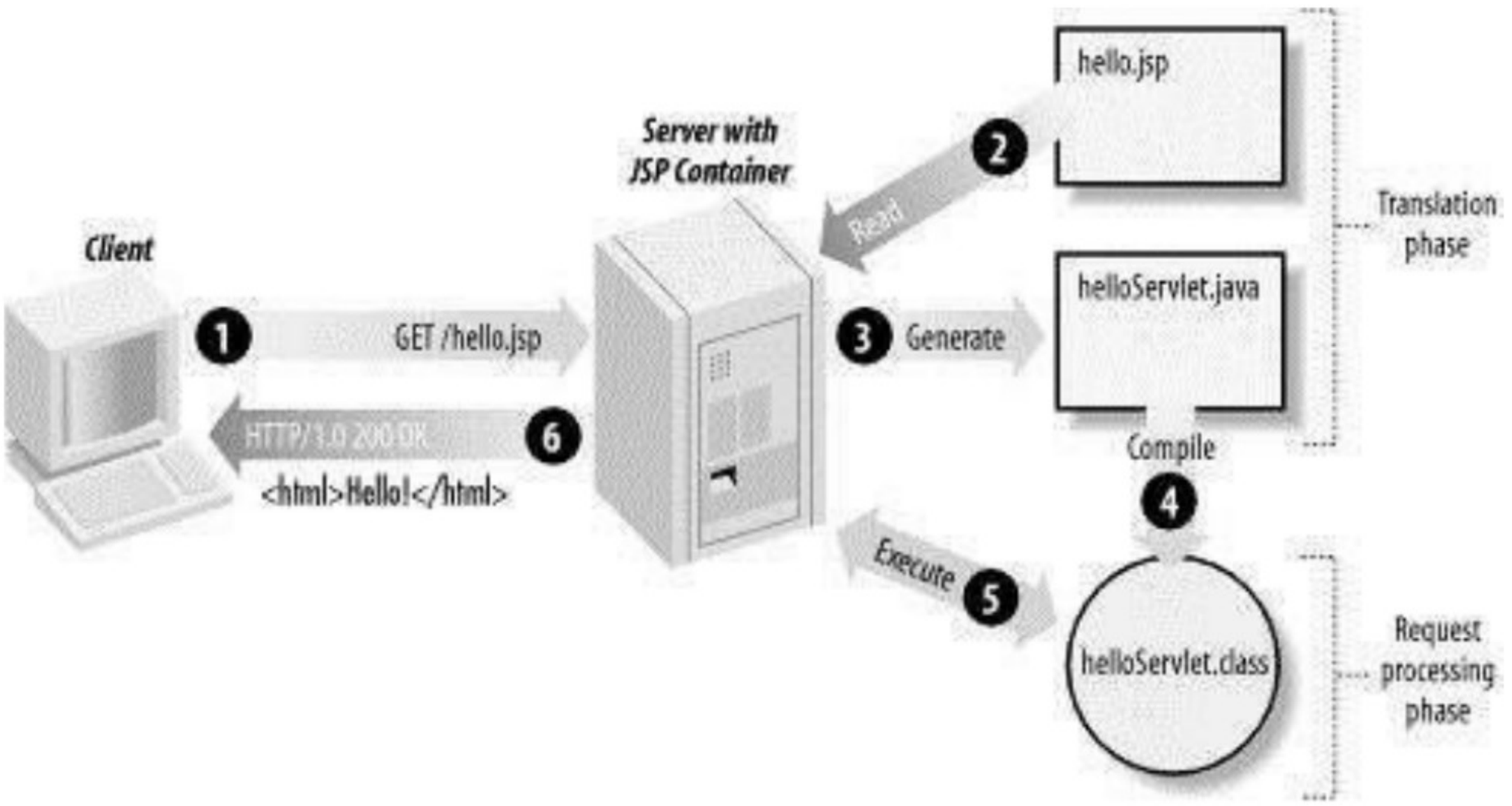
- Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications.
- It focuses more on presentation logic of the web apllication. **JSP** pages are easier to maintain then a **Servlet**. JSP pages are opposite of Servlets. Servlet adds HTML code inside Java code while JSP adds Java code inside HTML.
- JSP files are **HTML files** with **special tags** containing **Java source code** that provide the **dynamic content.**
- It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, jstl etc.

How JSP works

- When browser sends an **HTTP request** to the **web server**.
- The **web server recognizes** that HTTP request 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 **servlet** content. This conversion is very simple in which **all text template** is converted to **println( )**statements and all **JSP elements** are converted to **Java code** that implements the corresponding dynamic behaviour of the page.
- The **JSP engine compiles** the servlet into an executable class and forwards the original request to a **servlet engine**.
- Then, web server calls the **servlet engine** loads **the Servlet class** and **executes** it.
- During execution, the servlet produces an **output in HTML format**, which servlet engine passes to the web server inside of HTTP response.
- The web server forwards the **HTTP response** to the browser in terms of **static HTML content**.

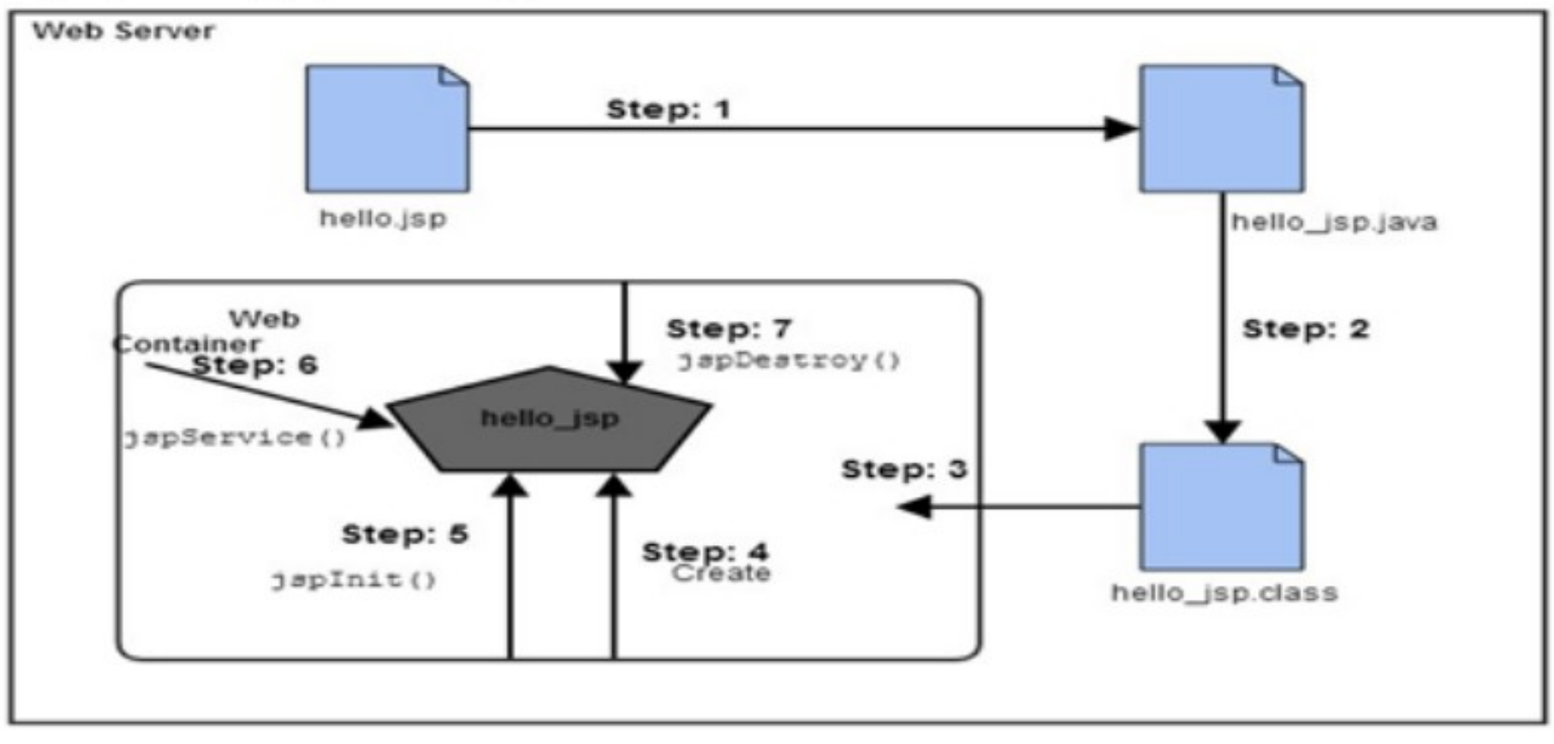


Advantages

- **Extension to Servlet**
  - We can use all the features of servlet in JSP. We can also use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
- **Easy to maintain**
  - JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.
- **Fast Development, No need to recompile and redeploy**
  - If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.
- **Less code than Servlet**
  - In JSP, we can use a lot of tags such as **action tags, JSTL (Java Standarad Tag Library), custom tags** etc. that reduces the code. Also, we can use **Expression Language (EL)**, **implicit objects** etc.
- **Platform independent**
  - It is built in java technology.

JSP Life Cycle

- A JSP page is converted into **Servlet** in order to service requests. The translation of a JSP page to a Servlet is called Lifecycle of JSP.
- JSP Lifecycle consists of following steps:
  1) **Translation of JSP to Servlet code.**
  2) **Compilation of Servlet to bytecode.**
  3) **Loading Servlet class.**
  4) **Creating servlet instance.**
  5) **Initialization by calling jspInit() method**
  6) **Request Processing by calling _jspService() method**
  7) **Destroying by calling jspDestroy() method**



1) **Translation of JSP to Servlet code**
   - JSP container checks the **JSP page code** and **parse** it to generate the **servlet source code**.
   - For example in Tomcat you will find generated **servlet class files** at **<tomcat>/WEBAPP/org/apache/jsp** directory.
   - If the JSP page name is **FirstJSP.jsp**, usually the generate **servlet class name** as **FirstJSP_jsp.class** after compilation and java file name is **FirstJSP_jsp.java** after translation.
2) **Compilation of Servlet to bytecode.**
   - JSP container **compiles** the **JSP class source** code and produces the **class file** in this phase.
3) **Loading Servlet class**
   - Web Container **loads the class** in this phase.
4) **Creating servlet instance**
   - Web Container invokes the **no-argument constructor** of generated class to **load and instantiate** it.
5) **Initialization (by calling jspInit() method)**
   - Web Container invokes the **init method** of JSP class object and initializes the **servlet config** with **init parms** configured in development descriptor.
   - After this phase, JSP is ready to handle client requests.
   - Usually form **translation to initialization** of JSP happens when **first request** for **JSP** comes.
6) **Request Processing (by calling _jspService() method)**
   - This is **the longest lifecycle phase** of JSP page and JSP page process the client requests.
   - The processing is **multi-threaded** and similar to servlets and for **every request a new thread** is created and **ServletRequest** and **ServletResponse** object is created and **JSP service method _jspService()** is invoked.
7) **Destroying (by calling jspDestroy() method)**
   - When the Web Container removes the **servlet instance from service**, it calls the **jspDestroy()** method to perform any required clean up.

JSP Life Cycle Methods

- 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 four major phases of JSP life cycle are very similar to Servlet Life Cycle and they are as follows:
  1) **Compilation (Convert JSP to Servlet)**
  2) **Initialization (jspInit())**
  3) **Execution (_jspService())**
  4) **Cleanup (jspDestroy())**

1) Compilation

- JSP engine compiles the page.
- The compilation process involves three steps:
  1. Parsing the JSP.
  2. Turning the JSP into a servlet.
  3. Compiling the servlet.

2) jspInit() (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...
}
```

- This method is called **only once** in the JSP lifecycle to initialize it.
- JSP **declaration scripting** element is used to initialize.

3) _jspService() (Execution)

- Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP container invokes the **_jspService()** in the JSP.
- The _jspService() method takes an **HttpServletRequest** and **HttpServletResponse** as its parameters as below:

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

- The **_jspService()** of a JSP is invoked **once per 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 **HTTPmethods** i.e. GET, POST, DELETE etc.

4) jspDestroy() (Cleanup)

- This method is called **only once** in JSP lifecycle.
- When the **Container** removes **the servlet instance**, it calls the **jspDestroy()** method to perform any required clean up.
- The jspDestroy () method has the following form:

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

Difference between Servlet and JSP

Servlet	JSP
Servlets are **java classes.**	JSP is **webpage scripting language**.
Servlets **run faster** compared to JSP.	JSP run **slower** compared to Servlet as it takes compilation time to convert into Java Servlets.
In MVC, servlet act as a **controller**.	In MVC, jsp act as a **view**.
servlets are best for use when there is more **processing and manipulation** involved.	JSP are generally preferred when there **is not much processing** of data required.
We **cannot** build **custom tags** using Servlet.	We **can** build **custom tags** using JSP.
To make servlet**, java code knowledge** must be there.	JSP are compiled to servlet effectively allowing you to produce a servlet by just writing the **HTML page**, **without knowing Java**.
In servlet **implicit objects** are **not present**.	In JSP **implicit objects** are **present.**
We **cannot** achieve functionality of servlets at **client side.**	We **can** achieve functionality of JSP at client side by running JavaScript at **client side**.
Consists of an **html file** for **static content** & **java file** for **dynamic content**.	Contains **java code embedded** directly to in an **html page** by using **special tags**.

JSP Scripting Elements

- The scripting elements provide the **ability to insert java code inside JSP.**
- JSP Scripting element is written inside **<% %>** tags. These code inside <% %> tags are processed by **the JSP engine during translation** of the JSP page.
- Any other text in the JSP page is considered as **HTML code or plain text**.

Scripting Element	Syntax
Comment	<%-- comment --%>
Scriptlet	<% scriplets %>
Declaration	<%! declarations %>
Expression	<%= expression %>
Directive	<%@ directive %>

1) JSP Comment :

- JSP Comment is used when you are creating a JSP page and want to put in comments about what you are doing.

- JSP comments are only seen in the JSP page. These comments are **neither included in servlet source code** during translation phase, nor they appear in **the HTTP response**.

  Syntax: <%-- JSP comment --%>

2) JSP Scriptlet :

- JSP Scriptlet Tag allows you **to write java code** inside JSP page.

  Syntax: <% java source code %>

3) JSP Declaration :

- The **JSP declaration tag** is used to declare **fields and methods**.
- The code written inside the jsp declaration tag, the declaration is made inside the **Servlet class** but outside the **service** (or any other) method.
- You can declare **static member, instance variable and methods** inside Declaration Tag.
- **Code** placed in this tag must end in a **semicolon**(;).
- Declarations do not generate output so are used with **JSP expressions or scriptlets**.

  Syntax: <%! Declaration %>

  Example: <%! int count=0; %>

4) JSP Expression :

- The code placed within **JSP expression tag** is written to **the output stream** of the response.
- So, you need not write **out.print()** to display data. It is mainly used to **print** the values of **variable or method**.

  Syntax : <%= Java Expression %>

  Example : <%= (10*2) %> it turns into **out.println((10*2));**
- Between **<%= %>** we can put anything and that will **converted to the String** and that will be **displayed**.

5) JSP Directive:

- It gives special instruction to Web Container at the time of page translation (JSP to Servlet).

  Syntax : <%@ **directive_Name** attribute_name="value" %>
- There are **three** types of directive tag:
  1) **page directive**
  2) **include directive**
  3) **taglib directive**

1) JSP page directive :

- The page directive defines attributes that apply to an entire JSP page.

  Syntax: <%@ **page** attribute_name="value" %>

- There are many attributes as given in below table:

Attribute	Description	Example
import	▪ The import attribute is used to **import class, interface** or all the members of a **package**. ▪ It is similar to import keyword in **java class or interface**.	<%@ page **import**="java.util.Date" %>
extends	▪ It defines the parent class that will be inherited by the generated **servlet**. ▪ It is rarely used.	-
session	▪ It defines whether the JSP page is participating in an **HTTP session**. ▪ The value is either true or false.	<%@ page **language**="java" **session**="true" %>
errorPage	▪ It is used to define the **error page.** ▪ if exception occurs in the current page, it will be **redirected to the error page**.	<%@ page **language**="java" **errorPage**="error.jsp" %>
isErrorPage	▪ It is used to declare that the **current page is the error page**. ▪ Default value is **false.**	<%@ page **isErrorPage**="true" %>
isELIgnored	▪ We can ignore the Expression Language (EL) in jsp by the **isELIgnored** attribute. ▪ By default its value is **false.** Means Expression Language is **enabled by default**.	<%@ page **isELIgnored**="true" %>
language	▪ It specifies the **scripting language** used in the JSP page. ▪ The default value is "**java**".	<%@ page **language**="java" %>
contentType	▪ It defines the **MIME** type for the **JSP response**. ▪ The default value is “text/html”.	<%@ page **contentType**="text/html” %>
autoFlush	▪ It defines whether the buffered **output is flushed automatically**.	<%@ page **autoFlush**="true" %>

	▪ The default value is "**true**".	
buffer	▪ It **sets the buffer size** in KB to handle output generated by the JSP page. ▪ The default size of the buffer is 8Kb.	<%@ page **buffer**="16kb" %>
isThreadSafe	▪ Servlet and JSP both are **multithreaded**. ▪ If you want to control this behaviour of JSP page, you can use isThreadSafe attribute of page directive. ▪ The value of isThreadSafe value is **true**. ▪ If you make it **false**, the web container will **serialize the multiple requests**. Means it will **wait** until the JSP finishes responding to a request before passing another request to it.	<%@ page **isThreadSafe**="false" %>
info	▪ It simply **sets the information of the JSP page** which is retrieved later by using **getServletInfo()** method of Servlet interface.	<%@ page **info**="Hello Welcome to info attribute" %>

2) JSP include directive :

- The include directive is used to include the contents of **any resource** it may be **jsp file, html file or text file**.
- It includes the original content of the included resource at page translation time.

  Syntax: <%@ **include** file="resourceName" %>

3) JSP taglib directive:

- The JSP taglib directive is used to define a **tag library** that defines many tags.
- We use the **TLD (Tag Library Descriptor) file** to define the tags.

  Syntax : <%@ **taglib** uri=" taglibURI " prefix="prefixOfTag" %>

- The **prefix** is used to distinguish the **custom tag from other libary custom tag**. Every custom tag must have a **prefix**.
- The **URI** is the **unique name** for Tag Library.

JSP implicit Objects/Predefined variables

- Implicit objects are created by **the web container** and are available to all **the jsp pages**.
- There are **9** JSP implicit objects.

Object	Description
out	▪ The **JspWriter** object associated with the output stream of the response. **Example :** <% **out**.println("Welcome") %>
request	▪ The **HttpServletRequest** object associated with the request. ▪ So, we can get request information parameter (getParameter()), header information (cookie), server port, content type etc. ▪ It can also be used to **set, get and remove attributes.** **Example:** <% String name=**request**.getParameter("uname"); %>
response	▪ The **HttpServletReponse** object associated with the response that is **sent back to the browser**. ▪ It can be used to **add or manipulate response** such as redirect response to another resource. **Example:** <% response.sendRedirect("http://www.google.com"); %>
config	▪ It is an implicit object of type **ServletConfig**. ▪ This object can be used to **get initialization parameter** for a particular JSP page from web.xml. **Example :** <% String name=**config**.getInitParameter("name"); %>
application	▪ It is an object of type **ServletContext.** ▪ The instance of **ServletContext** is created **only once** by the web container when application or project is deployed on the server. ▪ It can be used to get **initialization parameter** from configuration file (web.xml). ▪ It can also be used **to get, set or remove attribute** from the application scope. **Example:** <% String name=**application**.getInitParameter("name"); %>
Session	▪ It is an object of type **HttpSession**. ▪ The Java developer can use this object **to set, get or remove attribute or to get session information.** **Example:** <% session.setAttribute("user",name); %>
pageContext	▪ It is an object of type **PageContext** class. ▪ It can be used **to set, get or remove attribute** from **page, request, session or application** scope. ▪ page scope is the **default** scope. **Example:** <% **pageContext**.setAttribute("user",name,PageContext.**SESSION_SCOPE**); %>

page	▪ It is an object of type **Object class**. ▪ This object is assigned to the **reference** of auto generated **servlet** class.
exception	▪ It is an object of type **java.lang.Throwable** class. ▪ This object can be used to **print** the exception. ▪ This object is only available to pages that have **isErrorPage** set to **true** with the directive.