



Exploring Java Server Pages Technology

In the Web application development process, the Web designer is responsible for developing the static content of the Web application, such as the page layout. In addition, the Web developer is responsible for developing the dynamic content of the application that includes the presentation or business logic.

However, developing the static content of a Web application using servlets is complex and time consuming. This is because Servlet programming involves extensive coding.

Therefore, identification of the static code content and dynamic code content is required to incorporate any changes. To simplify this process, JSP was introduced. JSP facilitates this by segregating the work of the Web designer and the Web developer.

This chapter identifies the various components of a JSP page. In addition, it explains the lifecycle of a JSP page.

Objectives

In this chapter, you will learn to:

- 📖 Understand JSP technology
- 📖 Understand JSP lifecycle

Understanding JSP Technology

A typical Web application consists of the presentation logic, which contains the design and the structure, such as the page layout, of a Web page. In addition, it consists of the business logic or the dynamic content, which involves application of business-specific requirements. During the Web applications development, time is often lost in situations where the developer is required to write code for the static content. For example, consider an online website of Grant University. This website allows the students to register by using the registration page, as shown in the following figure.

The screenshot shows the Grant University website's registration page. The header is dark blue with 'GRANT UNIVERSITY' in yellow. A left sidebar contains a navigation menu with links: Home, About Us, Registration, Admissions, Contact Us, and Course Details. Below the menu is an 'Account Login' section with 'Id:' and 'Password:' labels, input fields, and 'Submit' and 'Reset' buttons. The main content area is titled 'Registration Form' and contains a registration form with the following fields: 'First Name:', 'Last Name:', 'Select Gender:' (with radio buttons for 'Male' and 'Female'), 'Date of birth:', 'Address:', 'Phone number:', 'E-mail Id:', 'User Id:', 'Password:', and 'Confirm password:'. Each field has a corresponding input field. At the bottom of the form are 'Submit' and 'Reset' buttons.

The Grant University Registration Page

As a developer, to create the registration page using servlets, you would need to write many `out.println()` statements. For example, to display the **First Name** label along with the text box field in the registration form, you would have to write the following code snippet in a servlet:

```
out.println("<Table>");
out.println("<Tr>");
out.println("<Td>First Name:");
out.println("</Td>");
out.println("<Td>");
out.println("<input type='text'>");
out.println("</Td>");
out.println("</Tr>");
...
```

Similarly, you will have to write these many lines of code to design and display the other components of the UI. As a developer, this becomes a tedious and time-consuming task. To overcome this limitation and simplify the UI creation of Web applications, JSP technology was introduced.

For example, the preceding code snippet can be simplified by using the following JSP code snippet:

```
<Table>
<Tr>
<Td>First Name:</Td>
<Td><input type="text"></Td>
</Tr>...
```

The preceding JSP code snippet is simple and easy to understand. Therefore, it becomes simple and easy for a Web designer to design and formulate the layout for the Web page by using HTML. In addition, a Web developer can work independently and use Java code and other JSP specific tags to code the business logic. This simultaneous construction of the static and dynamic content facilitates development of quality applications with increased productivity.

Identifying the Components of a JSP

A JSP page consists of regular HTML tags representing the static content, and the code enclosed within special tags representing the dynamic content. These tags begin with the “<%” symbol and end with the “%>” symbol.

Typically, a JSP page consists of various components that you can use in your Web page for additional functionality. They are:

- JSP comments
- JSP directives
- JSP declarations
- JSP scriptlets
- JSP expression
- JSP actions
- JSP implicit objects

The following table lists the syntax of some of the components of a JSP page.

<i>Component</i>	<i>Syntax</i>
<i>JSP comments</i>	<code><%-- comment --%></code>
<i>JSP directives</i>	<code><%@ directive %></code>
<i>JSP declarations</i>	<code><%! declaration %></code>
<i>JSP scriptlets</i>	<code><% %></code>
<i>JSP expression</i>	<code><%= expression %></code>

The Components of a JSP Page

JSP Comments

Comments explain the JSP code and make it more readable. It is not compiled and hence, is not included in the HTTP response. A comment can be added to a JSP page in the following ways:

- `<%-- comments --%>`
- `<% /** this is a comment ... **/ %>`
- `<!-- comments ... -->`

JSP Directives

JSP directives provide global information about a particular JSP page. To add a directive to the JSP page, you need to use the symbol, `<%@`, as the prefix and the symbol, `%>`, as the suffix of the directive name. A directive can have more than one attributes. The syntax for defining a directive is:

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

The various JSP directives are:

- The page directive
- The taglib directive
- The include directive

The page Directive

The page directive defines the attributes that notify the Web container about the general settings of a JSP page. You can specify different attributes with the page directive. The syntax of the page directive is:

```
<%@ page attribute_list %>;
```

The following table describes the various attributes supported by the page directive.

<i>Attribute</i>	<i>Description</i>	<i>Syntax</i>
<i>language</i>	<i>Defines the scripting language of the JSP page.</i>	<code><%@page language="java"%></code>
<i>extends</i>	<i>Defines the parent class that the JSP generated servlet extends.</i>	<code><%@page extends="myapp.Validation"%></code>
<i>import</i>	<i>Imports the list of packages, classes, or interfaces into the generated servlet.</i>	<code><%@page import="java.util.Date"%></code>
<i>session</i>	<i>Specifies if the generated servlet can access the session or not. An implicit object, <code>session</code>, is generated if the value is set to <code>true</code>. The default value of the <code>session</code> attribute is <code>true</code>.</i>	<code><%@page session="false"%></code>
<i>buffer</i>	<i>Specifies the size of the out buffer. If size is set to <code>none</code>, no buffering is performed. The default value of <code>buffer</code> size is 8 KB.</i>	<code><%@page buffer="10kb"%></code>
<i>autoFlush</i>	<i>Specifies that the out buffer be flushed automatically if the value is set to <code>true</code>. If the value is set to <code>false</code>, an exception is raised when the buffer is full. The default value of the <code>autoFlush</code> attribute is <code>true</code>.</i>	<code><%@page autoFlush = "false"%></code>
<i>isThreadSafe</i>	<i>Specifies whether a JSP page is thread-safe or not. The default value of the <code>isThreadSafe</code> attribute is <code>true</code>.</i>	<code><%@page isThreadSafe = "false"%></code>

<i>Attribute</i>	<i>Description</i>	<i>Syntax</i>
<i>errorPage</i>	<i>Specifies the URL of a JSP page that will handle any unchecked Java exception.</i>	<code><%@page errorPage="ErrorPage.jsp"%></code>
<i>isErrorPage</i>	<i>Specifies that the current JSP page is an error page, if the attribute value is set to true. The default value of the isErrorPage attribute is false.</i>	<code><%@page isErrorPage="true"%></code>
<i>isELIgnored</i>	<i>Specifies that the current JSP page will ignore all the EL expressions, if this attribute is set to true. The default value of the isELIgnored attribute is false.</i>	<code><%@page isELIgnored="true"%></code>
<i>info</i>	<i>Provides a description of a JSP page.</i>	<code><%@page info="This JSP page will display the Home Page of Grant University website"%></code>
<i>pageEncoding</i>	<i>Specifies the language used by the JSP page to send the response to the Web browser.</i>	<code><%@page pageEncoding="UTF-8"%></code>
<i>contentType</i>	<i>Defines the MIME type for a response. The default value of the contentType attribute is text/html.</i>	<code><%@page contentType="text/html"%></code>

The Attributes of the page Directive

The following code snippet imports the `java.util.Date` package in a JSP page:

```
<%@page import="java.util.Date" contentType="text/html"
pageEncoding="UTF-8"%>
```

The preceding code snippet imports the package, `java.util.Date`, by using the `import` attribute of the page directive.

The following code snippet displays the use of the `errorPage` attribute of the page directive:

```
%@page errorPage="ErrorPage.jsp" contentType="text/html"
pageEncoding="UTF-8"%>
```

The preceding code snippet redirects a user to the error page named **ErrorPage.jsp**, whenever there is an unhandled exception in the current page. This is implemented by assigning the name of the error page as a value to the `errorPage` attribute of the page directive.

Now, if you want the **ErrorPage.jsp** page should display the exception message to the user stating the cause of exception, you can use the `isErrorPage` attribute in the page directive of the JSP page, as shown in the following code snippet:

```
<%@page isErrorPage="true"contentType="text/html" pageEncoding=
"UTF-8"%>
```

The preceding page code snippet sets the `isErrorPage` attribute to `true`. This enables the **Error.jsp** page to use the `getMessage()` method of the `Exception` class to retrieve the exception message. This retrieved message is then displayed in the **ErrorPage.jsp** page.

The include Directive

The include directive includes the output of the file specified using the `file` attribute in the form of its relative URL in the calling JSP page. The file is included in the calling page at the translation time. Therefore, if you have made any changes to the included file, it will be reflected only after the next compilation of the JSP page. The syntax to define the include directive is:

```
<%@ include file = "URLname" %>
```

Consider an example, where for the Grant University Web application you have created a JSP page named `University.html`, which consists of the logo, as shown in the following figure.



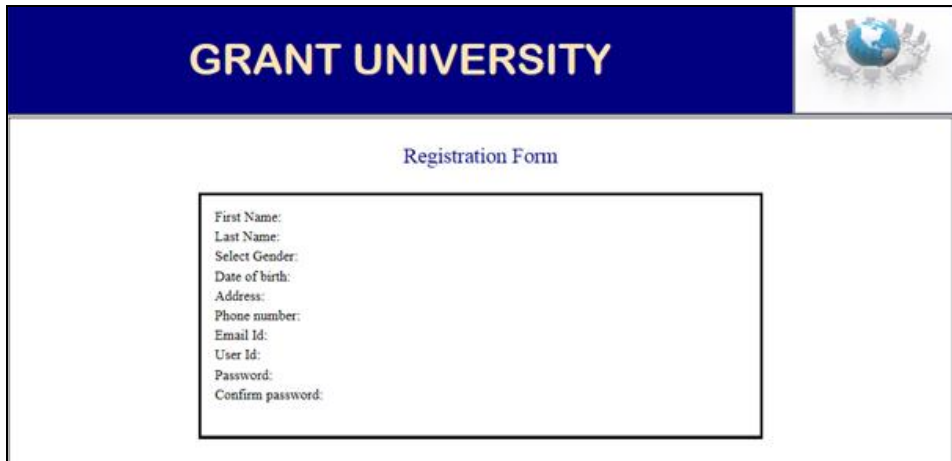
The University.html Page

Now, you want that this logo should be visible in all the other Web pages of the application. This can be achieved either by copying and then pasting the logo code to all the other Web pages or by simply using the include directive in the respective pages.

The following code snippet uses the include directive in the registration page:

```
<%@ include file = "University.html" %>
```


The following figure displays the output of the registration page after using the include directive.



The Registration Page

The taglib Directive

The taglib directive is used to import a custom tag into the current JSP page. Custom tag is a user-defined tag, which is used to perform repetitive tasks in a JSP page. For example, you can create a custom tag that implements the functionality of login controls, which can be used in various Web pages of a Web application. The Tag Library Descriptor (TLD) file defines the functionality of a custom tag.

The taglib directive associates itself with a Uniform Resource Identifier (URI) to uniquely identify a custom tag. It also associates a tag prefix string that will distinguish a custom tag from the other tag library used in the JSP page. The syntax to import a taglib directive in the JSP page is:

```
<%@ taglib uri="tag_lib_URI" prefix="prefix" %>
```

The taglib directive accepts two attributes. The following table lists the two attributes of the taglib directive along with their description.

<i>Attribute</i>	<i>Description</i>
<i>Uri</i>	<i>Locates the tld file of a custom tag.</i>
<i>Prefix</i>	<i>Defines a prefix string to be used for distinguishing a custom tag instance.</i>

The Attributes of the taglib Directive

JSP Declarations

The JSP declarations provide mechanism to define variables and methods in a JSP page. The declarative statements are placed within the `<%!` and `%>` symbols and end with a semicolon.

The following code snippet uses JSP declarations to define variables and methods:

```
<%!  
int i=5;  
int add()  
{  
i=i+5;  
return i;  
}  
%>
```

The preceding code snippet declares a variable `i` of `int` data type and initializes it with the value 5. In addition, the method `add()` is defined, which modifies and returns the value of the `i` variable.

JSP Expressions

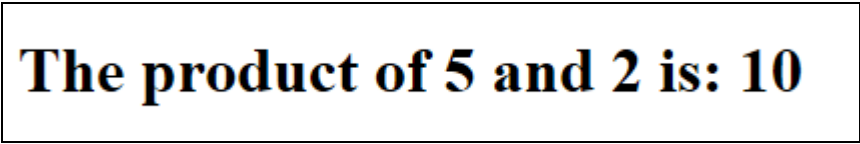
JSP expressions are used to directly insert values into the response output. The JSP expressions are evaluated when a user makes an HTTP request. The syntax to include a JSP expression in the JSP page is:

```
<%= expression%>
```

The following code snippet uses JSP expressions to evaluate the value of the expression specified within the `<%=` and `%>` symbols:

```
<h1>The product of 5 and 2 is: <%= (2 * 5) %></h1>
```

The output of the preceding code snippet is shown in the following figure.



The product of 5 and 2 is: 10

The Output of the JSP Expression

JSP Scriptlets

A JSP scriptlet consists of Java code snippets that are enclosed within the `<%` and `%>` symbols. They are executed at the request time. The syntax to include valid Java code in JSP scriptlets is:

```
<% //Java code %>
```

The following code snippet uses JSP scriptlets to include Java code in a JSP page:

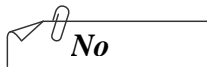
```
<% int i=10;
    if(i>0)
    {
        out.println("i is a positive number");
    }
    else
    {
        out.println("i is a negative number");
    }
%>
```

The preceding code snippet declares a local variable `i` of `int` data type and initializes it with a value `10`. In addition, it displays the message, **i is a positive number** to the users.

JSP Actions

JSP actions are the tags that follow the XML syntax. By using JSP actions you perform tasks, such as inserting files, reusing beans, forwarding a user to another page, and instantiating objects. The following depicts the syntax of JSP actions:

```
<jsp:actionname attribute="">
```



A Java bean is a simple Java class that exposes internal fields as properties using the corresponding getter and setter methods.

The following table lists the various JSP action tags along with their description, attribute supported, and description of attributes.

JSP Action	Description	Attribute	Description of Attribute
<code><jsp:useBean></code>	<i>Invokes and searches for an existing bean.</i>	<i>Id</i> <i>class</i> <i>scope</i> <i>beanName</i>	<i>Uniquely identifies the instance of the bean.</i> <i>Identifies the class from which the bean objects are to be implemented.</i> <i>Defines the scope of the bean.</i> <i>Defines the referential name for the bean.</i>
<code><jsp:getProperty></code>	<i>Retrieves the property of a bean.</i>	<i>name</i> <i>property</i>	<i>Defines the name for the bean.</i> <i>Defines the property from which the values are to be retrieved.</i>
<code><jsp:setProperty></code>	<i>Is used to set the property of a bean.</i>	<i>name</i> <i>property</i> <i>value</i> <i>param</i>	<i>Specifies a name for the bean.</i> <i>Defines the property for which values are to be set.</i> <i>Defines an explicit value for the bean property.</i> <i>Defines the name of the request parameter to be used.</i>
<code><jsp:forward></code>	<i>Is used to forward a request to a target page.</i>	<i>Page</i>	<i>Specifies the URL of the target page.</i>
<code><jsp:include></code>	<i>Includes a file in the current JSP page.</i>	<i>Page</i> <i>flush</i>	<i>Specifies the URL of the resource to be included. The resource is included in the calling page at the run time. Therefore, if you have made any changes to the included resource, it will be reflected in the next request.</i> <i>Specifies whether the buffer should be flushed or not. The flush value can be either true or false.</i>

JSP Action	Description	Attribute	Description of Attribute
<code><jsp:param></code>	<i>Defines a parameter to be passed to an included or forwarded page.</i>	Name value	<i>Defines the name of the reference parameter.</i> <i>Defines the value of the specified parameter.</i>
<code><jsp:plugin></code>	<i>Executes a Java applet or a Java bean.</i>	type code codebase	<i>Defines the type of plug-in to be included.</i> <i>Defines the name of the class to be executed by the plug-in.</i> <i>Defines the path of the code.</i>

The JSP Action Tags

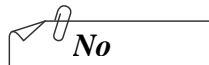
JSP Implicit Objects

JSP implicit objects are predefined objects provided by the container that can be included in JSP expressions and scriptlets. These implicit objects are mapped to the classes and interfaces of the servlet API. The following table describes the various implicit objects of JSP.

Implicit Object	Class	Description
<code>request</code>	<code>javax.servlet.http.HttpServletRequest</code>	<i>It represents the <code>HttpServletRequest</code> object associated with the request.</i>
<code>response</code>	<code>javax.servlet.http.HttpServletResponse</code>	<i>It represents the <code>HttpServletResponse</code> object associated with the response that is sent back to the browser.</i>
<code>out</code>	<code>javax.servlet.jsp.JspWriter</code>	<i>It represents the <code>JspWriter</code> object associated with the output stream of the response.</i>

<i>Implicit Object</i>	<i>Class</i>	<i>Description</i>
<code>session</code>	<code>javax.servlet.http.HttpSession</code>	<i>It represents the HttpSession object associated with the session for the given user of the request. This variable does not exist if JSP is not participating in the session.</i>
<code>application</code>	<code>javax.servlet.ServletContext</code>	<i>It represents the ServletContext object for the Web application.</i>
<code>config</code>	<code>javax.servlet.ServletConfig</code>	<i>It represents the ServletConfig object associated with the servlet for the JSP page.</i>
<code>page</code>	<code>java.lang.Object</code>	<i>It represents the current instance of the JSP page that, in turn, is used to refer to the current instance of the generated servlet.</i>
<code>pageContext</code>	<code>javax.servlet.jsp.PageContext</code>	<i>It represents the page context for a JSP page.</i>
<code>exception</code>	<code>java.lang.Throwable</code>	<i>It represents the Throwable exception in a JSP page.</i>

The JSP Implicit Objects



The `JspWriter` class extends from the `java.io.Writer` class and is used to render output to a JSP page. The object of the `JspWriter` class is referenced by the implicit variable, `out`.

The JSP implicit objects, such as request and response can be used to retrieve the request data from the client and send a response to the client. For example, you have created an application, which accepts and validates the user name and password entered by a user on the login page.

To accept and validate the user name and password, you have created the following JSP pages:

- **UserInput.jsp:** Displays the interface that allows the user to enter the user name and the password. These input values are passed as request parameters to the Welcome.jsp page.
- **Welcome.jsp:** Processes the input given by the user and displays a customized welcome message to the user.

The **UserInput.jsp** page consists of the following code:

[illegible]

The output of the preceding code is displayed in the following figure.

MY FORM

Enter the user name:

Enter the password:

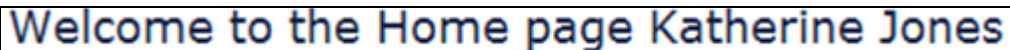
The Output of the UserInput.jsp Page

When a user types **Katherine Jones** and **password@123** in the first and second text boxes, respectively, and then clicks the **SUBMIT** button, the **Welcome.jsp** page is invoked. This page retrieves and validates the value stored in the user name and password fields of the **UserInput.jsp** page and generates a customized welcome message to the user.

The **Welcome.jsp** page consists of the following code:

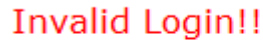
```
<!-- Import the Java packages --%>
<%@ page language="java"%>
<%@ page import="java.lang.*"%>
<html>
<font size=4 face="Verdana" color=#112244>
<body>
<%
    String str1=request.getParameter("t1");
    String str2=request.getParameter("t2");
    if(str1.equals("Katherine
Jones")&&str2.equals("password@123"))
    {
        out.println("Welcome to the Home page " + str1);
    }
    else
}
%>
    <jsp:forward page="ErrorPage.jsp"></jsp:forward>
    <%
}
%>
</font>
</body>
</html>
```

In the preceding code, the user name and password are retrieved using the `getParameter()` method of the `request` object. If the user name and password entered in the **UserInput.jsp** page is valid, a welcome message is displayed to the user, as shown in the following figure.



The Output of the Welcome.jsp Page

Otherwise, the user is forwarded to the **ErrorPage.jsp** page, as shown in the following figure.



Invalid Login!!

The Output of the ErrorPage.jsp Page

The **ErrorPage.jsp** page consists of the following code:

```
<%@ page language="java"%>
<%@ page import="java.lang.*"%>

<html>
<head>
<title>Error Page</title>
</head>
<body>
    <font color='red' size='+2'>Invalid Login!!
</body>
</html>
```



Which one of the following components of the JSP page consists of valid Java code snippets that are enclosed within the <% and %> symbols?

1. *JSP scriptlets*
2. *JSP declarations*
3. *JSP expression*
4. *JSP implicit objects*

Answer:

1. *JSP scriptlets*

Understanding JSP Lifecycle

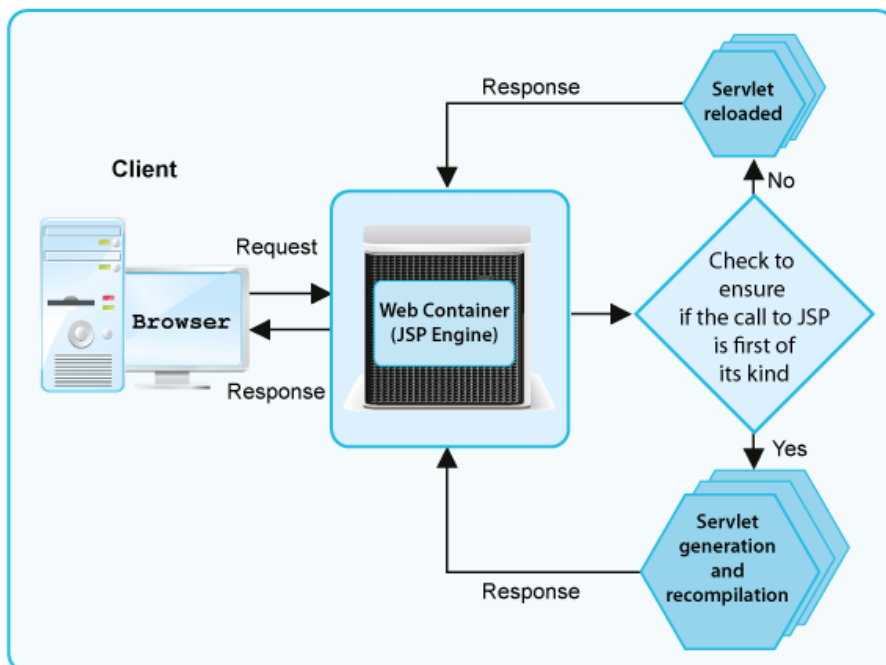
When a client requests for a JSP page, the corresponding JSP page goes through a lifecycle. Similar to servlets, the lifecycle of the JSP page is managed by the Web container. The lifecycle of a JSP page starts when the Web container receives the request for the page and continues for the subsequent request-response cycle of the same JSP page. It ends when no more requests are received for the same JSP page. The JSP page lifecycle is carried out in two phases. They are:

- Translation phase
- Request-processing phase

Lifecycle of a JSP Page

Whenever the client browser requests for a particular JSP page, the server, in turn, sends a request to the JSP engine. A JSP engine is a part of a Web container that compiles a JSP page to a servlet.

The following figure represents the process of the flow of events that occur after a client requests for a JSP page.



The JSP Lifecycle

The Web container determines whether the request for the JSP page is first of its kind. If the JSP page is requested for the first time, it is translated to a servlet. The Web container compiles and executes the servlet to generate the response for the request. However, if the request for the JSP page is not first of its kind, the corresponding compiled servlet is reloaded to generate the response. The generated response is sent to the client.

The lifecycle of a JSP page is managed using the following lifecycle methods of the `javax.servlet.jsp.JspPage` interface:

- `jspInit()`: This method initializes the servlet and is invoked when the corresponding servlet of the requested JSP page is loaded in the Web container for the first time.
- `_jspService()`: This method is invoked when the servlet corresponding to the requested JSP page is initialized and is ready to process the client requests. The Web container invokes this method to execute the initialized servlet to process the JSP page request.
- `jspDestroy()`: This method is invoked by the Web container when the corresponding servlet of the JSP page is not required to service any subsequent requests and needs to be removed from the Web container's memory.

Processing of a JSP Page

A JSP page needs to be converted to a servlet before it can service a client request. The processing of a JSP page is carried out in the following phases:

- **Translation:** Is responsible for translating the JSP code to the servlet code.
- **Compilation:** Is responsible for the compilation of the servlet code to the servlet/bytecode class.
- **Servlet class loading:** Is responsible for loading of the servlet class in the Web container.
- **Servlet instance creation:** Is responsible for creating an instance of the loaded servlet.
- **Servlet initialization:** Is responsible for initializing the servlet instance by calling the `jspinit()` method.
- **Servicing client requests:** Is responsible for servicing the client requests by calling the `jspservice()` method.
- **Servlet destruction:** Is responsible for destroying the servlet by calling the `jspdestroy()` method.

Consider a Web application, which consists of a **Hello.jsp** page. This page displays a hello message to the user along with the number of users accessing the Web application.

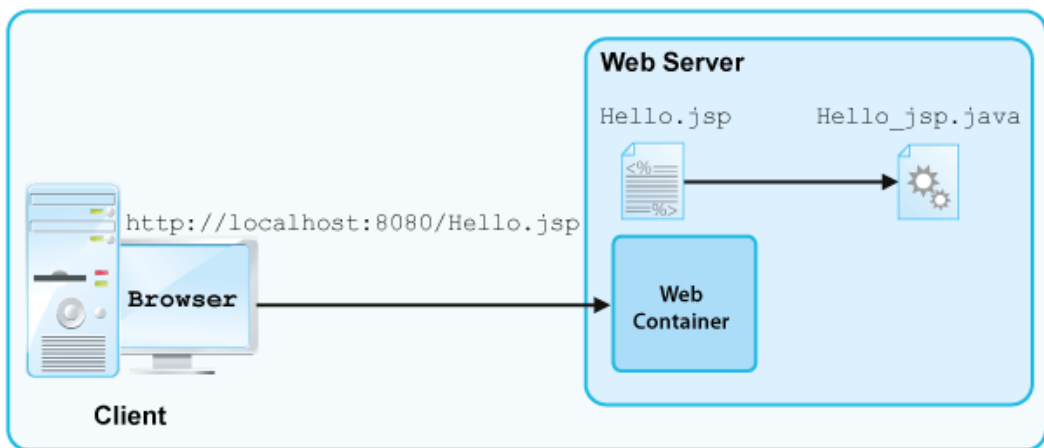
The following code snippet implements this:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <body>
    <%! int i=0; %>
    Hello user number:
    <%=++i %>
  </body>
</html>
```

Whenever a client requests for the **Hello.jsp** page, the processing of the JSP page occurs to process the client request.

Translation

When a client requests for the Hello.jsp page, the Web container translates the **Hello.jsp** page to **Hello_jsp.java** servlet, as shown in the following figure.



The Translation of the JSP Code to the Servlet Code

For the **Hello.jsp** page, the Web container creates a servlet named `Hello_Jsp` that extends the `HttpServlet` class, as shown in the following code snippet:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```

public class Hello_Jsp extends HttpServlet

{

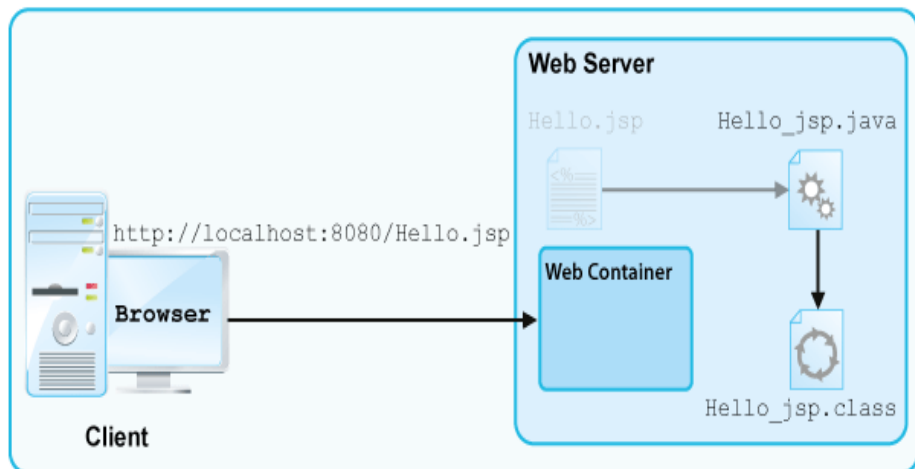
    int i=0;
    public void _jspService(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("Hello user number:");
        out.println(++i);
        out.println("</body>");
        out.println("</html>");
    }
}

```

In the preceding code snippet, the `<%! int i=0; %>` declarative expression in the **Hello.jsp** page is translated to the global declaration of variable, `i`, just after the `Hello_JspServlet` class declaration. The HTML tags of the **Hello.jsp** page is translated to the `out.println()` statements in the `_jspService()` method. In addition, the `<%=++i %>` expression statement of the **Hello.jsp** page is translated to `out.println(++i)` statement in the `_jspService()` method.

Compilation

The Web container compiles the **Hello_jsp.java** file to bytecode. This results in the creation of the **Hello_jsp.class** file, as shown in the following figure.

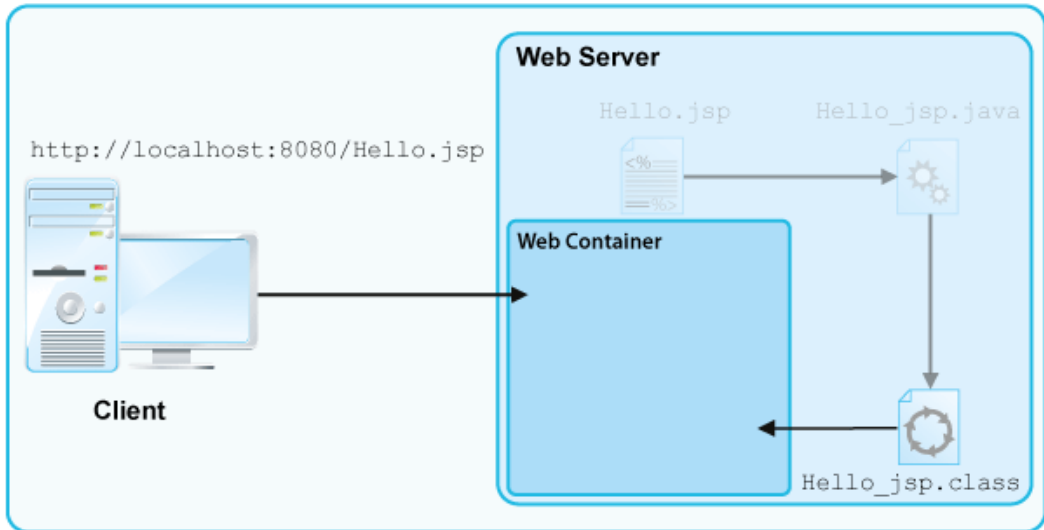


The JSP Page Compilation

Once the code has been compiled, it is ready for execution.

The Servlet Class Loading

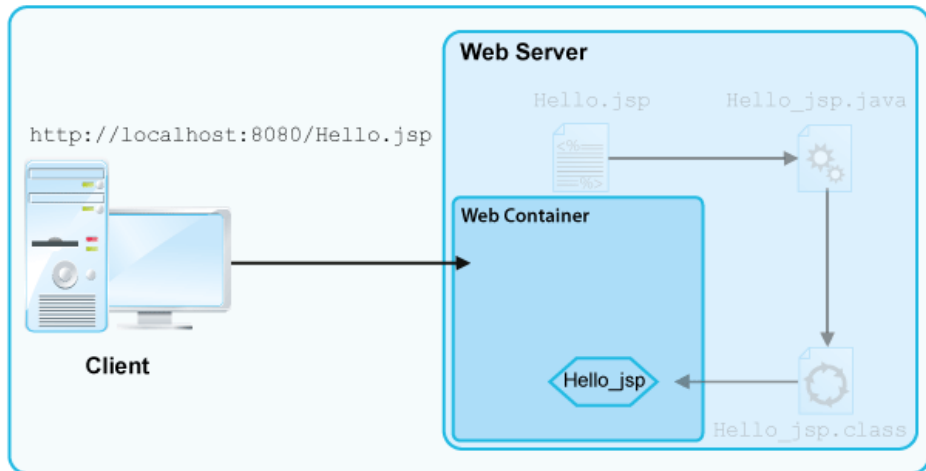
The class loader of the Web container loads the **Hello_jsp.class** file in the Web container's memory, as shown in the following figure.



The Servlet Class Loading

Creation of a Servlet Instance

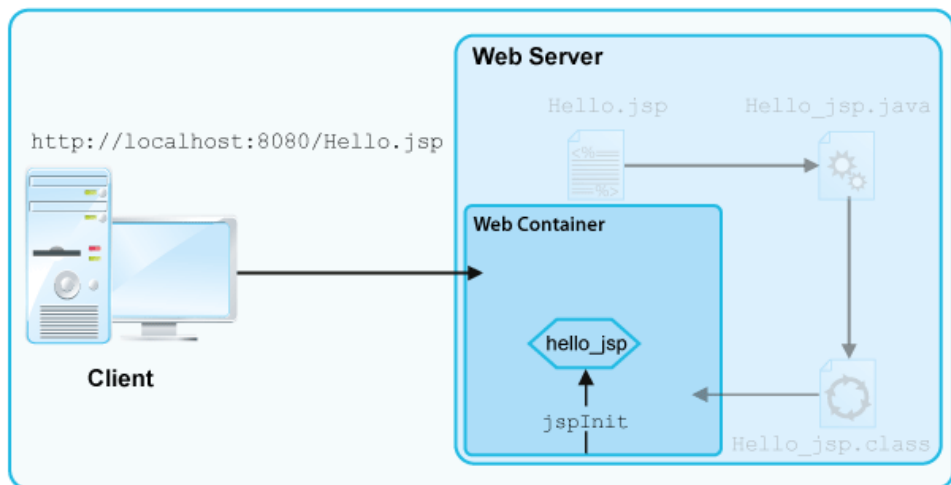
The Web container creates an instance of the **Hello_jsp.class** file, as shown in the following figure.



The Servlet Instance Creation

Servlet Initialization

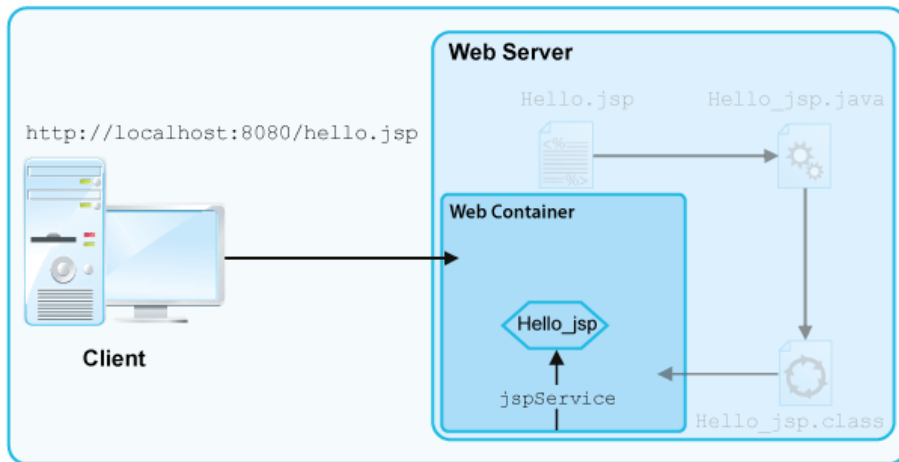
The Web container initializes the instance of the **Hello_jsp.class** file by calling the `jspinit()` method, as shown in the following figure.



The Servlet Initialization

Servicing Client Requests

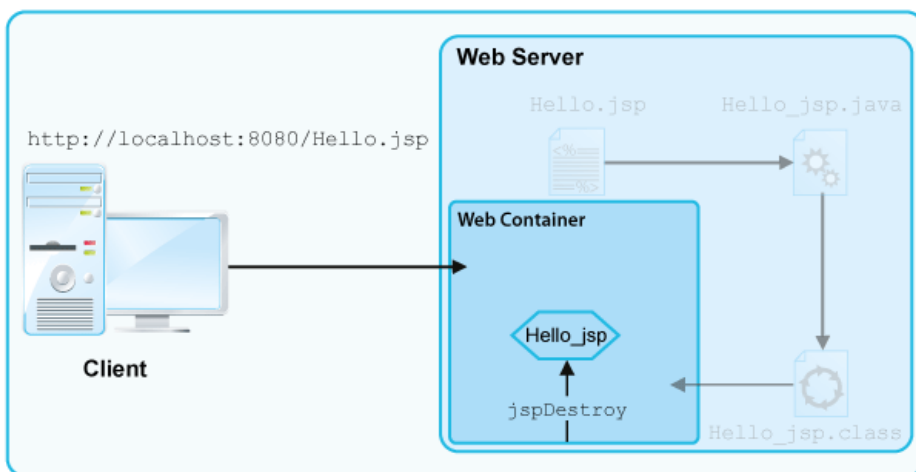
The Web container responds to the client's requests by calling the `_jspService()` method of the initialized `Hello_jsp.class` file, as shown in the following figure.



Servicing the Client Requests

Servlet Destruction

When the servlet is not required to service any client requests, the Web container removes the JSP servlet instance from the Web container's memory by calling the `jspDestroy()` method. This method performs the required clean up. The following figure shows the servlet destruction.



The Servlet Destruction



Activity 5.1: Creating JSP Pages



Just a

Which of the following steps are performed in the translation phase:

- a. Translation of the JSP code to the servlet code.*
 - b. Compilation of the servlet code to the servlet/bytecode class.*
 - c. Loading of the servlet class in the Web container.*
 - d. Creation of the servlet instance.*
 - e. Initialization of the servlet instance.*
-
- 1. a and b only*
 - 2. a, b, and c only*
 - 3. a, b, c, and d only*
 - 4. c, d, and e only*

Answer:

- 1. a and b only*

Practice Questions

1. Which one of the following components of a JSP page provides global information about the JSP page?
 - a. JSP directives
 - b. JSP scriptlets
 - c. JSP declarations
 - d. JSP action tags
2. Which one of the following attributes of the page directive specifies that the current JSP page is an error page?
 - a. `isErrorPage`
 - b. `autoflush`
 - c. `isThreadSafe`
 - d. `errorPage`
3. The `out` implicit object represents the _____ object associated with the output stream of the response.
 - a. `JspWriter`
 - b. `HttpRequest`
 - c. `HttpResponse`
 - d. `ServletConfig`
4. Which one of the following methods is invoked when the corresponding servlet of the requested JSP page is loaded in the Web container for the first time?
 - a. `init()`
 - b. `service()`
 - c. `jspInit()`
 - d. `jspService()`
5. Which one of the following methods is invoked when the servlet corresponding to a JSP page has been initialized and is ready to process the client requests?
 - a. `init()`
 - b. `service()`
 - c. `jspInit()`
 - d. `jspService()`