Assignment

Course Code: CSE 307
Course Title: Internet and Web Programming

Date: October 13, 2020

Submitted to:	Submitted by:
DR. TUSHAR KANTI SAHA Associate Professor, Department of Computer Science & Engineering Jatiya Kabi Kazi Nazrul Islam University Trishal, Mymensingh-2220.	MD.ZAWHARUL ISLAM B.Sc.(Eng.) in CSE ID: 17102014 Session: 2016-17 Department of Computer Science & Engineering Jatiya Kabi Kazi Nazrul Islam University Trishal, Mymensingh-2220.

Jatiya Kabi Kazi Nazrul Islam University

Trishal, Mymensingh-2220
Bangladesh.

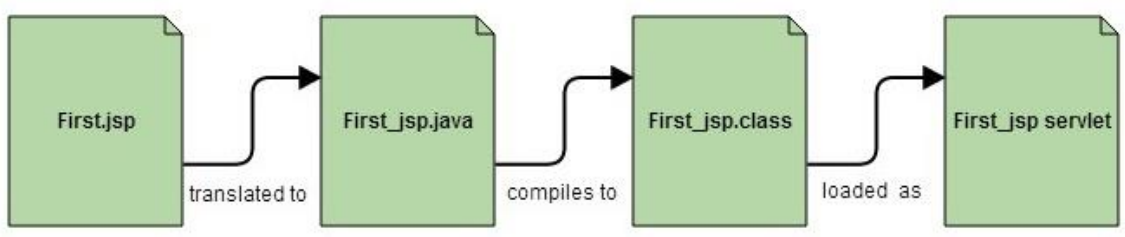
Introduction to JSP

JSP means **Java Server Pages. JSP** technology is used to create dynamic web applications.

JSP pages are easier to maintain then a **Servlet**. JSP pages are opposite of Servlets as a servlet adds HTML code inside Java code, while JSP adds Java code inside HTML using JSP tags. Everything a Servlet can do, a JSP page can also do it.

In the end a JSP becomes a Servlet

JSP pages are converted into **Servlet** by the Web Container. The Container translates a JSP page into servlet **class source(.java)** file and then compiles into a Java Servlet class.



Why JSP is preferred over servlets?

- JSP provides an easier way to code dynamic web pages.
- JSP does not require additional files like, java class files, web.xml etc.
- Any change in the JSP code is handled by Web Container (Application server like tomcat), and doesn't require re-compilation.
- JSP pages can be directly accessed, and web.xml mapping is not required like in servlets.

Advantage of JSP

- Easy to maintain and code.
- High Performance and Scalability.
- JSP is built on Java technology, so it is platform independent.

Lifecycle of a JSP page

1. JSP page translation:

A Java servlet file is created from the JSP source file. In the translation phase, the container validates the correctness of the JSP pages and tag files.

2. JSP page compilation:

The created java servlet file is compiled into a Java servlet class.

3. Class loading:

The java servlet class that was compiled from the JSP source is now loaded into the container.

4. Execution phase:

In the execution phase, the container creates one or more instances of this class in response to the requests. The interface JSP Page contains jspInit() and jspDestroy(). JSP provides special interface HttpJspPage for JSP pages specifically for the HTTP requests and this interface contains _jspService().

5. Initialization:

jspInit() method is called immediately after the instance is created.

6. jspDestroy() execution:

This method is called when JSP is destroyed. With this call, the servlet completes its purpose and goes into the garbage collection This ends the JSP life cycle.

Syntax of JSP

The syntax for the following in JSP:

1. JSP Expression

<%= expression %>

Example: &<%marks1 = marks1 + marks2 %>

2. Declaration Tag

<%! Dec var %>

Example: <%! int var = 50; %>

3. JSP scriptlet

<% java code %>

Here, you can insert the respective Java code.

4. JSP comments

<% -- JSP Comments %>

What is a servlet?

Java servlets were the first attempt to get access to the full power of Java in Web applications. They are written in Java.

Now, let me show you a code which will teach you to create a JSP page.

A Simple JSP page

```
<HTML>

<HEAD>

<TITLE>A Web Page</TITLE>

</HEAD>

<BODY>

<% out.println ("JSP in JAVA") ; %>

</BODY>

</HTMl>
```

As you can see in the above code, how easily a JSP page is created. This easier approach has helped JSP to take off so well. Simple HTML tags have been used. An additional element <% out.println (“JSP in JAVA”) ; %> can be seen. This element is called a scriptlet! It includes a java code used in an HTML-JSP code.

How to run a JSP page

Execution of JSP involves several steps. These are mentioned below:

1. Firstly, create an HTML file, say, ana.html, from here a request will be sent to the server.
2. Secondly, create a .jsp file, say, ana1.jsp, this would tackle the request of the user.
3. Thirdly, create a project folder structure.
4. Now, you need to create an XML file and then a WAR file.
5. After that, start Tomcat
6. Finally, you are ready to run the application.

JSP Directive Tag

Directive Tag gives special instruction to Web Container at the time of page translation.
Directive tags are of three types: **page**, **include** and **taglib**.

Directive	**Description**

<%@ page ... %>	defines page dependent properties such as language, session, error page etc.
<%@ include ... %>	defines file to be included.
<%@ taglib ... %>	declares tag library used in the page

You can place page directive anywhere in the JSP file, but it is good practice to make it as the first statement of the JSP page.

Page directive

The **Page directive** defines a number of page dependent properties which communicates with the Web Container at the time of translation.

Basic syntax of using the page directive is <%@ page attribute="value" %> where attributes can be one of the following :

- import attribute
- language attribute
- extends attribute
- session attribute
- isThreadSafe attribute
- isErrorPage attribute
- errorPage attribute
- contentType attribute
- autoFlush attribute
- buffer attribute

import attribute

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

<%@ page import="java.util.Date" %>

or

<%@ page import="java.util.Date,java.net.*" %>

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

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 Include Directive

The include directive tells the Web Container to copy everything in the included file and paste it into current JSP file. Syntax of **include** directive is:

```
<%@ include file="filename.jsp" %>
```

Example of include directive

welcome.jsp

```
<html>
   <head>
      <title>Welcome Page</title>
   </head>

  <body>
      <%@ include file="header.jsp" %>
      Welcome, User
```

```
  </body>
</html>
```

header.jsp

```
<html>
  <body>
     <img src="header.jpg" alt="This is Header image" / >
  </body>
</html>
```

The example above is showcasing a very standard practice. Whenever we are building a web application, with webpages, all of which have the top navbar and bottom footer same. We make them as separate jsp files and include them using the include directive in all the pages. Hence whenever we have to update something in the top navbar or footer, we just have to do it at one place.

JSP taglib Directive

The taglib directive is used to define tag library that the current JSP page uses. A JSP page might include several tag library.

Java Server Pages Standard Tag Library (JSTL), is a collection of useful JSP tags, which provides many commonly used core functionalities. It has support for many general, structural tasks such as iteration and conditionals, readymade tags for manipulating XML documents, internationalization tags, and for performing SQL operations.

Syntax of taglib directive is:

```
<%@ taglib prefix="prefixOfTag" uri="uriOfTagLibrary" %>
```

JSP Actions

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. Following table lists out the available JSP Actions –

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.
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 Objects

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

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.

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.

Control-Flow Statements

You can use all the APIs and building blocks of Java in your JSP programming including decision-making statements, loops, etc.

Decision-Making Statements

The if...else block starts out like an ordinary Scriptlet, but the Scriptlet is closed at each line with HTML text included between the Scriptlet tags.

```
<%! int day = 3; %>

<html>

<head><title>IF...ELSE Example</title></head>


 <body>
```

```
<% if (day == 1 || day == 7) { %>
<p> Today is weekend</p>
<% } else { %>
<p> Today is not weekend</p>
 <% } %>
 </body>
</html>
```

The above code will generate the following result –

Today is not weekend

switch...case:

Now look at the following switch...case block which has been written a bit differently using out.println() and inside scriptlets −

```
<%! int day = 3; %>
<html>
<head><title>SWITCH...CASE Example</title></head>
<body>
<%
switch(day) {
case 0:
out.println("It\'s Sunday.");
break;
case 1:
out.println("It\'s Monday.");
break;
case 2:
out.println("It\'s Tuesday.");
```

```
  break;
case 3:
out.println("It\'s Wednesday.");
break;
case 4:
out.println("It\'s Thursday.");
break;
case 5:
out.println("It\'s Friday.");
break;
default:
out.println("It's Saturday.");
}
%>
</body>
</html>
```

The above code will generate the following result −

```
It's Wednesday.
```

Loop Statements

You can also use three basic types of looping blocks in Java: for, while, and do…while blocks in your JSP programming.

Let us look at the following for loop example −

```
<%! int fontSize; %>
```

```
<html>
<head><title>FOR LOOP Example</title></head>

<body>
<%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>
<font color = "green" size = "<%= fontSize %>">
JSP Tutorial
</font><br />
<%}%>
</body>
</html>
```

The above code will generate the following result −

JSP Tutorial

JSP Tutorial

JSP Tutorial

Above example can be written using the while loop as follows −

```
<%! int fontSize; %>
<html>
<head><title>WHILE LOOP Example</title></head>
<body>
<%while ( fontSize <= 3){ %>
<font color = "green" size = "<%= fontSize %>">
JSP Tutorial
</font><br />
<%fontSize++;%>
<%}%>
</body>
```

</html>

The above code will generate the following result −

JSP Tutorial

JSP Tutorial

JSP Tutorial

JSP array

Example of array

<%

String[]arr={“apple”,”orange”,cherry”};

%>

>%

Int i;

Out.println(“<p>The array element are</p>”);

For(i=0;i<arr.length;i++){

Out.println(arr[i]);

}

%>

Result:

The array element are

Apple orange cherry

JSP Operators

JSP supports all the logical and arithmetic operators supported by Java. Following table lists out all the operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.

Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right

Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

JSP Literals

The JSP expression language defines the following literals −

- **Boolean** − true and false
- **Integer** − as in Java
- **Floating point** − as in Java
- **String** − with single and double quotes; " is escaped as \", ' is escaped as \', and \ is escaped as \\.
- **Null** − null

JSP Exception Handling

Exception Handling is a process of handling exceptional condition that might occur in your application. Exception Handling in JSP is much easier than Java Technology exception handling. Although JSP Technology also uses the same exception class objects.

It is quite obvious that you don't want to show error stack trace to any random user surfing your website. You can't prevent all errors in your application but you can at least give a userfriendly error response page.

Ways to perform Exception Handling in JSP

JSP provide 3 different ways to perform exception handling:

1. Using **isErrorPage** and **errorPage** attribute of page directive.
2. Using **<error-page>** tag in **Deployment Descriptor**.

3. Using simple try...catch block.

JSP Standard Tag(Action Element)

JSP specification provides **Standard**(Action) tags for use within your JSP pages. These tags are used to remove or eliminate scriptlet code from your JSP page because scriptlet code are technically not recommended nowadays. It's considered to be bad practice to put java code directly inside your JSP page.

Standard tags begin with the jsp: prefix. There are many JSP Standard Action tag which are used to perform some specific task.

The following are some JSP Standard Action Tags available:

Action Tag	Description
jsp:forward	forward the request to a new page Usage : <jsp:forward page="Relative URL" />
jsp:useBean	instantiates a JavaBean Usage : <jsp:useBean id="beanId" />
jsp:getProperty	retrieves a property from a JavaBean instance. Usage : <jsp:useBean id="beanId" ... /> ... <jsp:getProperty name="beanId" property="someProperty" .../> Where, **beanName** is the name of pre-defined bean whose property we want to access.
jsp:setProperty	store data in property of any JavaBeans instance. Usage : <jsp:useBean id="beanId" ... /> ... <jsp:setProperty name="beanId" property="someProperty"

	value="some value"/> Where, **beanName** is the name of pre-defined bean whose property we want to access.
jsp:include	includes the runtime response of a JSP page into the current page.
jsp:plugin	Generates client browser-specific construct that makes an OBJECT or EMBED tag for the Java Applets
jsp:fallback	Supplies alternate text if java plugin is unavailable on the client. You can print a message using this, if the included jsp plugin is not loaded.
jsp:element	Defines XML elements dynamically
jsp:attribute	defines dynamically defined XML element's attribute
jsp:body	Used within standard or custom tags to supply the tag body.
jsp:param	Adds parameters to the request object.
jsp:text	Used to write template text in JSP pages and documents. Usage : <jsp:text>Template data</jsp:text>

What is JSTL?

JSTL stands for Java server pages standard tag library, and it is a collection of custom JSP tag libraries that provide common web development functionality.

Advantages of JSTL

1. **Standard Tag**: It provides a rich layer of the portable functionality of JSP pages. It's easy for a developer to understand the code.
2. **Code Neat and Clean**: As scriptlets confuse developer, the usage of JSTL makes the code neat and clean.

3. **Automatic JavabeansInterospection Support**: It has an advantage of JSTL over JSP scriptlets. JSTL Expression language handles JavaBean code very easily. We don't need to downcast the objects, which has been retrieved as scoped attributes. Using JSP scriptlets code will be complicated, and JSTL has simplified that purpose.
4. **Easier for humans to read**: JSTL is based on XML, which is very similar to HTML. Hence, it is easy for the developers to understand.
5. **Easier for computers to understand**: Tools such as Dreamweaver and front page are generating more and more HTML code. HTML tools do a great job of formatting HTML code. The HTML code is mixed with the scriplet code. As JSTL is expressed as XML compliant tags, it is easy for HTML generation to parse the JSTL code within the document.

JSTL Core

The core tags are most frequently used tags in JSP. They provide support for

- Iteration
- Conditional logic
- Catch exception
- url forward
- Redirect, etc.

To use core tags we need to define tag library first and below is the syntax to include a tag library.

Syntax :

<%@ taglib prefix="c" uri=http://java.sun.com/jsp/jstl/core%>

JSP Custom Tags

- It is a user-defined JSP language element.
- When JSP is translated into a servlet, custom tag is converted into a class which takes action on an object and is called as a tag handler.
- Those actions when the servlet is executed are invoked by the web container.
- To create the user-defined custom tag, we need to create the tag handler which will be extending the Simple Tag Support and have to override do Tag() method.
- We need to create TLD where we need to map the class file in TLD.

Advantages of custom tags in JSP:

- **Portable** - An action described in a tag library must be usable in any JSP container.
- **Simple** - Unsophisticated users must be able to understand and use this mechanism. Vendors of JSP functionality must find it easy to make it available to users as actions.
- **Expressive** - The mechanism must support a wide range of actions, including nested actions, scripting elements inside action bodies, creation, use and updating of scripting variables.
- **Usable from different scripting languages** - Although the JSP specification currently only defines the semantics for scripts in the Java programming language, we want to leave open the possibility of other scripting languages.
- **Built upon existing concepts and machinery**- We do not want to reinvent what exists elsewhere. Also, we want to avoid future conflicts whenever we can predict them.

Syntax:

Consider we are creating testGuru tag and we can use tag handler test Tag class, which will override doTag() method.

<ex:testGuru/>

Class testTag extends SimpleTagSupport{ public void doTag()}

JSP Client Request

- When the web page is requested, it sends information to the web server in the HTTP header.
- We can use this information using HTTP Servlet Request object.
- Information sent by the browser is stored in the request header of HTTP request.
- We are using different headers to send information to the request object.

Different headers are described below:

Header	Description	Example
Accept	It specifies MIME types that browser or other clients can handle	Image/png or image/jpeg

Header	Description	Example
Accept-charset	It uses the character set used by the browser to display the information	ISO-8859-1
Accept-Encoding	It specifies type of encoding handled by the browser	Gzip or compress
Accept-language	It specifies clients specified language	En,en_us
Authorization	Header used by clients when trying to access password protected web pages	
Connection	It indicates whether client can handle persistent HTTP connections(browser can retrieve multiple files)	Keep-alive
Content-length	Applicable to post requests. It gives size of post data of bytes	
Cookie	Returns cookie to server(those which were previously sent to the browser)	
Host	Specifies the host and port of the original URL	
If modified since	It indicates that it requires only a page if it has been changed or modified	
If unmodified since	It indicates that it requires a page only if it has not been changed or modified	
Referrer	Indicates URL of referring URL page	
User-agent	Identifies browser or client making request	

Following methods are used to read the HTTP header in JSP page:

1. **Cookie[] getCookies() –** returns an array containing cookie objects that the client has sent
2. **Enumeration getAttributeNames() –** contains enumeration of names of attributes for request
3. **Enumeration getHeaderNames() -** contains enumeration of names of header .
4. **Enumeration getParameterNames() –** contains enumeration of getting parameter names in the request.
5. **HttpSessiongetSession() –** returns the current session associated with the request or if does not have a session then it will create a new one.

6. **Locale getLocale() –** returns the preferred locale that client will accept content in.It has been assigned to the response. By default, the value will be default locale of the server.
7. **Object getAttribute(String name) –** returns the value of named attribute as an object.
8. **ServletInputStreamgetInputStream() –** retrievesbody of request as binary data.
9. **String getAuthType() –** returns the name of authentication scheme to protect servlet
10. **String getCharacterEncoding() –** returns name of the character encoding used in the body of the request.
11. **String getContentType() –** returns the MIME type of body of the request.
12. **String getContextPath() –** returns the part of request URI indicates context path of URI
13. **String getHeader(String name) –** returns the request header as a string
14. **String getMethod() –** returns the name of the HTTP method like GET, POST
15. **String getParameter(String name) –** returns the parameter of the request as a string.
16. **String getPathInfo() –** returns the path information associated with the URL
17. **String getQueryString() –** returns the query string that is associated with the request URL
18. **String getServletPath() –** returns the part of URLs of the request that calls the JSP
19. **String[] getParameterValues(String name) –** returns the array of string objects containing the values that request parameter has

Example:

In the example below, we are using different methods using request object

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>

<%@ page import="java.io.* java.util.*" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Client Request Guru JSP</title>
</head>
<body>
<h2>Client Request Guru JSP</h2>

<table border="1">
<tr>
<th>guru header</th><th>guru header Value(s)</th>
</tr>
<%
        HttpSession gurusession = request.getSession();
        out.print("<tr><td>Session Name is </td><td>" +gurusession+ "</td.></tr>");
        Locale gurulocale = request.getLocale ();
        out.print("<tr><td>Locale Name is</td><td>" +gurulocale + "</td></tr>");
        String path = request.getPathInfo();
        out.print("<tr><td>Path Name is</td><td>" +path+ "</td></tr>");
        String lpath = request.get();
        out.print("<tr><td>Context path is</td><td>" +lipath + "</td></tr>");
        String servername = request.getServerName();
        out.print("<tr><td>Server Name is </td><td>" +servername+ "</td></tr>");
        int portname = request.getServerPort();
        out.print("<tr><td>Server Port is </td><td>" +portname+ "</td></tr>");
        Enumeration hnames = request.getHeaderNames();
        while(hnames.hasMoreElements()) {
                String paramName = (String)hnames.nextElement();
                out.print ("<tr><td>" + paramName + "</td>" );
```

```
            String paramValue = request.getHeader(paramName);
            out.println("<td> " + paramValue + "</td></tr>");
        }
%>
```

JSP Server Response

- When a request is processed and then the response is generated from the web server. It consists of a status line, response headers, a blank line and document.
- It is the object of HTTPServletResponseclass, which is a response object.
- The status line is a version of HTML.

Response headers are mentioned below:

Header	Description
Allow	It specifies the request methods like GET, POST that server is requesting
Cache-control	The response document can be cached. It can be public, private and no cache. No cache specifies that document should not be cached
Connection	It instructs whether the browser should use saved HTTP Connections or not. Close value represents that browser should not use persistent in HTTP Connections and "keep-alive" means using persistent connections
Content-disposition	To ask the user whether to save the response to disk or not
Content-encoding	Page has to be encoded during transmission
Content-length	Number of bytes in the response
Content type	It specifies the MIME type of response
Expires	Specifies till when the content should be considered out of date and should not be cached

Header	Description
Last modified	It indicates when the document was last modified
Location	It should be included with all responses which have status code has 300 as status code
Refresh	It specifies how to find the updated page.
Retry-after	It can be used with 503 response to tell client of how soon it can repeat request
Set-cookie	Specifies the cookie associated with the page

Following are the methods using response object:

1. **String encodeRedirectURL(String URL) –** encodes the URL in redirectURL method.
2. **String encodeURL(String URL) –** encodes the URL by including session ID.
3. **Boolean containsHeader(String name) –** it contains a header in the JSP or not.
4. **Boolean isCommited() –** response has been committed or not.
5. **Void addCookie(Cookie cookie) – adds cookie to the response**
6. **Void addDateHeader(String name, String value) –** adds response header date name and value
7. **Void addHeader(String name, String value) –** adds response header with name and value
8. **Void addIntHeader(String name,int value) –** adds response header with name and integer value
9. **Void flushBuffer() –** forces content in the buffer to the output to the client.
10. **Void reset() –** clears data in the buffer.
11. **Void resetBuffer –** clears the content buffer in the response without clearing status codes.
12. **Void sendError(intsc,Stringmsg) –** sends an error response to the client using status code.
13. **Void sendRedirect(String location) –** sends a temporary redirect response to the client.
14. **Void setBufferSize(int size) –** sets buffer size of the body
15. **Void setCharacterEncoding(String charset) –** sets character encoding
16. **Void setContentType(String type) –** sets the content type of the response

17. **Void setContentLength(intlen) –** sets the content length of the response
18. **Void setLocale(Locale lcl) –** sets the locale type of the response
19. **Void setStatus(intsc) –** sets the status code of the response

Example:

In this example, we are covering different methods getLocale,flushbuffer, getWriter, get ContentType, setIntHeader.

Example:

In the example below, we are using different methods using request object

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>

<%@ page import="java.io.* java.util.*" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Client Request Guru JSP</title>

</head>

<body>

<h2>Client Request Guru JSP</h2>


<table border="1">

<tr>

<th>guru header</th><th>guru header Value(s)</th>
```

```
</tr>
<%
        HttpSession gurusession = request.getSession();
        out.print("<tr><td>Session Name is </td><td>" +gurusession+ "</td.></tr>");
        Locale gurulocale = request.getLocale ();
        out.print("<tr><td>Locale Name is</td><td>" +gurulocale + "</td></tr>");
        String path = request.getPathInfo();
        out.print("<tr><td>Path Name is</td><td>" +path+ "</td></tr>");
        String lpath = request.get();
        out.print("<tr><td>Context path is</td><td>" +lipath + "</td></tr>");
        String servername = request.getServerName();
        out.print("<tr><td>Server Name is </td><td>" +servername+ "</td></tr>");
        int portname = request.getServerPort();
        out.print("<tr><td>Server Port is </td><td>" +portname+ "</td></tr>");
        Enumeration hnames = request.getHeaderNames();
        while(hnames.hasMoreElements()) {
                String paramName = (String)hnames.nextElement();
                out.print ("<tr><td>" + paramName + "</td>" );

                String paramValue = request.getHeader(paramName);
                out.println("<td> " + paramValue + "</td></tr>");
        }

%>
```

JSP HTTP Status Codes

- When the request is processed, the response is generated. The response status line consists of HTTP version, a status code and an associated message.

- The message is directly associated with the status code and HTTP version, and it is determined by the server.
- By default 200 is set as a status code in JSP, so we don't need to set explicitly.
- We can set as response.setStatus() method

The codes fall in following 5 categories:

- 100-199 - Here client indicates that it should respond with some action
- 200-299 - It signifies that request is successful
- 300-399 - They are used for files that have been moved and usually include a location header indicating new address
- 400-499 - Indicates error by the client
- 500-599 - Indicates error by the server

Some of the common status codes are below:

- 200 – Indicates everything is fine
- 301 – It has moved permanently
- 304 – Not modified since last change
- 400 – Bad request
- 404 – Not found
- 405 - Method not found
- 500 – Internal Server Error
- 503 - Service unavailable
- 505 – HTTP version not supported

Some of its methods are listed below:

1. Public void setStatus(intstatusCode)

   It sets the status code whichever we want to set in that JSP Page.This will give us the message of status code which has been set

2. Public void sendRedirect(String URL)

It generates 302 response along with the location header giving URL of the new document

3. Public void sendError(intcode,Stringmsg)

   It sends the status code along with the short message and it is formatted inside HTML document.

Example:

In this example, we are sending error to JSP page explicitly.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>


<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Guru Status Code</title>

</head>

<body>

<% response.sendError(404,"Guru Page Not Found"); %>

</body>

</html>
```

Cookies in JSP

What are Cookies?

- Cookies are the text files which are stored on the client machine.
- They are used to track the information for various purposes.
- It supports HTTP cookies using servlet technology
- The cookies are set in the HTTP Header.

- If the browser is configured to store cookies, it will keep information until expiry date.

Following are the cookies methods:

- Public void setDomain(String domain)

  It is used to set the domain to which the cookie applies

- Public String getDomain()

  It is used to get the domain to which cookie applies

- Public void setMaxAge(int expiry)

  It sets the maximum time which should apply till the cookie expires

- Public intgetMaxAge()

  It returns the maximum age of cookie

- Public String getName()

  It returns the name of the cookie

- Public void setValue(String value)

  Sets the value associated with the cookie

- Public String getValue()

  Get the value associated with the cookie

- Public void setPath(String path)

  It sets the path to which cookie applies

- Public String getPath()

  It gets the path to which the cookie applies

- Public void setSecure(Boolean flag)

  It should be sent over encrypted connections or not.

- Public void setComment(String cmt)

It describes the cookie purpose

- Public String getComment()

  It the returns the cookie comments which has been described.

How to Handle Cookies in JSP

1. Creating the cookie object
2. Setting the maximum age
3. Sending the cookie in HTTP response headers

Example:

In this example, we are creating cookies of username and email and add age to the cookie for 10 hours and trying to get the variable names in the action_cookie.jsp

Action_cookie.jsp.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Guru Cookie</title>
</head>
<body>
<form action="action_cookie_main.jsp" method="GET">
Username: <input type="text" name="username">
<br />
Email: <input type="text" name="email" />
<input type="submit" value="Submit" />
```

```
</form>
</body>
</html>
```

Action_cookie_main.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%

   Cookie username = new Cookie("username",
                                request.getParameter("username"));
   Cookie email = new Cookie("email",
                                request.getParameter("email"));



   username.setMaxAge(60*60*10);
   email.setMaxAge(60*60*10);

   // Add both the cookies in the response header.
   response.addCookie( username );
   response.addCookie( email );
%>



<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

```
<title>Guru Cookie JSP</title>
</head>
<body>

<b>Username:</b>
  <%= request.getParameter("username")%>
<b>Email:</b>
  <%= request.getParameter("email")%>

</body>
</html>
```

JSP Form Processing

Forms are the common method in web processing. We need to send information to the web server and that information.

There are two commonly used methods to send and get back information to the web server.

1. GET Method:

- This is the default method to pass information from browser to web server.
- It sends the encoded information separated by ?character appended to URL page.
- It also has a size limitation, and we can only send 1024 characters in the request.
- We should avoid sending password and sensitive information through GET method.

2. POST Method:

- Post method is a most reliable method of sending information to the server.
- It sends information as separate message.
- It sends as text string after ?in the URL.
- It is commonly used to send information which are sensitive.

JSP handles form data processing by using following methods:

1. getParameter():

   It is used to get the value of the form parameter.

2. getParameterValues():

   It is used to return the multiple values of the parameters.

3. getParameterNames()

   It is used to get the names of parameters.

4. getInputStream()

   It is used to read the binary data sent by the client.

   Example:

   In this example, we have taken a form with two fields."username" and "password" with a submit button

   Action_form.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>


<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Guru Form</title>
</head>
<body>
<form action="action_form_process.jsp" method="GET">
UserName: <input type="text" name="username">
<br />
```

```
Password: <input type="text" name="password" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

Action_form_process.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

<h1>Form Processing</h1>

<p><b>Welcome User:</b>
   <%= request.getParameter("username")%>
</p>

</body>
</html>
```

JSP Current Date and Time

JSP Date Handling

All methods of core Java can be used in JSP is the biggest advantage of JSP.

In this section, we will be using Date class of java.util package, and it consists of date and time.

It supports two constructors:

Date() – It gives us current date and time

Date(long miilsec) – This takes parameter of milliseconds which has been elapsed on January 1 1970

Boolean after(Date date) – It gives after date of the given date parameter

Boolean before(Date date) – It gives the before date of the given date parameter

Object clone() – It creates a copy of the date object

IntcompareTo(Date date) – it compares the date class object with another one

IntcompareTo(Object date) – it compares with the object class object with another one

Boolean equals(Object date) – it checks whether two date objects are equal

Long getTime() – it fetches the time

InthashCode() – it fetches the hashcode of the given date

Void setTime(long Time) – It sets the time of given date object

String toString() – It converts into string object of date object.

Example:

In this example, we are fetching the current date and time using date object

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
   <%@ page import="java.util.*" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Guru current Date</title>
</head>
<body>
Today's date: <%= (new java.util.Date()).toLocaleString()%>
</body>
</html>
```

JSP Database Connection:

Create Table

To create the **Employees** table in the EMP database, use the following steps –

Step 1

Open a **Command Prompt** and change to the installation directory as follows −

```
C:\>
C:\>cd Program Files\MySQL\bin
C:\Program Files\MySQL\bin>
```

Step 2

Login to the database as follows −

```
C:\Program Files\MySQL\bin>mysql -u root -p
Enter password: ********
mysql>
```

Step 3

Create the **Employee** table in the **TEST** database as follows − −

```
mysql> use TEST;
mysql> create table Employees
(
    id int not null,
    age int not null,
    first varchar (255),
    last varchar (255)
  );
Query OK, 0 rows affected (0.08 sec)

mysql>
```

Create Data Records

Let us now create a few records in the **Employee** table as follows − −

```
mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');
Query OK, 1 row affected (0.00 sec)
 mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');
Query OK, 1 row affected (0.00 sec)
mysql>
```

SELECT Operation

Following example shows how we can execute the **SQL SELECT** statement using JTSL in JSP programming −

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>

<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix = "c"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>


<html>
  <head>
    <title>SELECT Operation</title>
  </head>


  <body>
    <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
      url = "jdbc:mysql://localhost/TEST"
      user = "root"  password = "pass123"/>


    <sql:query dataSource = "${snapshot}" var = "result">
      SELECT * from Employees;
    </sql:query>


    <table border = "1" width = "100%">
      <tr>
```

```
            <th>Emp ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Age</th>
         </tr>

         <c:forEach var = "row" items = "${result.rows}">
            <tr>
               <td><c:out value = "${row.id}"/></td>
               <td><c:out value = "${row.first}"/></td>
               <td><c:out value = "${row.last}"/></td>
               <td><c:out value = "${row.age}"/></td>
            </tr>
         </c:forEach>
      </table>

   </body>
</html>
```

Access the above JSP, the following result will be displayed −

Emp ID	First Name	Last Name	Age
100	Zara	Ali	18

101	Mahnaz	Fatma	25
102	Zaid	Khan	30
103	Sumit	Mittal	28

INSERT Operation

Following example shows how we can execute the SQL INSERT statement using JTSL in JSP programming −

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>

<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix = "c"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>


<html>

  <head>

    <title>JINSERT Operation</title>

  </head>


  <body>

    <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"

      url = "jdbc:mysql://localhost/TEST"

      user = "root"  password = "pass123"/>

      <sql:update dataSource = "${snapshot}" var = "result">
```

```
   INSERT INTO Employees VALUES (104, 2, 'Nuha', 'Ali');
</sql:update>

<sql:query dataSource = "${snapshot}" var = "result">
   SELECT * from Employees;
</sql:query>

<table border = "1" width = "100%">
   <tr>
      <th>Emp ID</th>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Age</th>
   </tr>

   <c:forEach var = "row" items = "${result.rows}">
      <tr>
         <td><c:out value = "${row.id}"/></td>
         <td><c:out value = "${row.first}"/></td>
         <td><c:out value = "${row.last}"/></td>
         <td><c:out value = "${row.age}"/></td>
      </tr>
   </c:forEach>
```

```
</table>


</body>
</html>
```

Access the above JSP, the following result will be displayed −

Emp ID	First Name	Last Name	Age
100	Zara	Ali	18
101	Mahnaz	Fatma	25
102	Zaid	Khan	30
103	Sumit	Mittal	28
104	Nuha	Ali	2

DELETE Operation

Following example shows how we can execute the **SQL DELETE** statement using JTSL in JSP programming −

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>

<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix = "c"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>
```

```
<html>
  <head>
    <title>DELETE Operation</title>
  </head>

  <body>
    <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
      url = "jdbc:mysql://localhost/TEST"
      user = "root" password = "pass123"/>

    <c:set var = "empId" value = "103"/>

    <sql:update dataSource = "${snapshot}" var = "count">
      DELETE FROM Employees WHERE Id = ?
      <sql:param value = "${empId}" />
    </sql:update>

    <sql:query dataSource = "${snapshot}" var = "result">
      SELECT * from Employees;
    </sql:query>

    <table border = "1" width = "100%">
      <tr>
```

```
      <th>Emp ID</th>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Age</th>
    </tr>

    <c:forEach var = "row" items = "${result.rows}">
      <tr>
        <td><c:out value = "${row.id}"/></td>
        <td><c:out value = "${row.first}"/></td>
        <td><c:out value = "${row.last}"/></td>
        <td><c:out value = "${row.age}"/></td>
      </tr>
    </c:forEach>
  </table>

 </body>
</html>
```

Access the above JSP, the following result will be displayed −

Emp ID	First Name	Last Name	Age
100	Zara	Ali	18

101	Mahnaz	Fatma	25
102	Zaid	Khan	30

UPDATE Operation

Following example shows how we can execute the **SQL UPDATE** statement using JTSL in JSP programming −

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>

<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>

<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c"%>

<%@ taglib uri = "http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>


<html>

  <head>

    <title>DELETE Operation</title>

  </head>


  <body>

    <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"

      url = "jdbc:mysql://localhost/TEST"

      user = "root" password = "pass123"/>


    <c:set var = "empId" value = "102"/>
```

```
<sql:update dataSource = "${snapshot}" var = "count">
   UPDATE Employees SET WHERE last = 'Ali'
   <sql:param value = "${empId}" />
</sql:update>

<sql:query dataSource = "${snapshot}" var = "result">
   SELECT * from Employees;
</sql:query>

<table border = "1" width = "100%">
   <tr>
      <th>Emp ID</th>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Age</th>
   </tr>

   <c:forEach var = "row" items = "${result.rows}">
      <tr>
         <td><c:out value = "${row.id}"/></td>
         <td><c:out value = "${row.first}"/></td>
         <td><c:out value = "${row.last}"/></td>
```

```
            <td><c:out value = "${row.age}"/></td>
         </tr>
      </c:forEach>
   </table>

</body>
</html>
```

Access the above JSP, the following result will be displayed −

Emp ID	First Name	Last Name	Age
100	Zara	Ali	18
101	Mahnaz	Fatma	25
102	Zaid	Ali	30