H2K Infosys, LLC
Dream, Strive & Achieve Victory

H2K Infosys, LLC
Dream, Strive & Achieve Victory

H2K Infosys, LLC
Dream, Strive & Achieve Victory

www.H2KINFOSYS.com
USA: 770-777-1269
Training@H2KInfosys.com
UK : (020) 3371 7615

JSP
java server pages

What is JavaServer Pages?

JavaServer Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

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

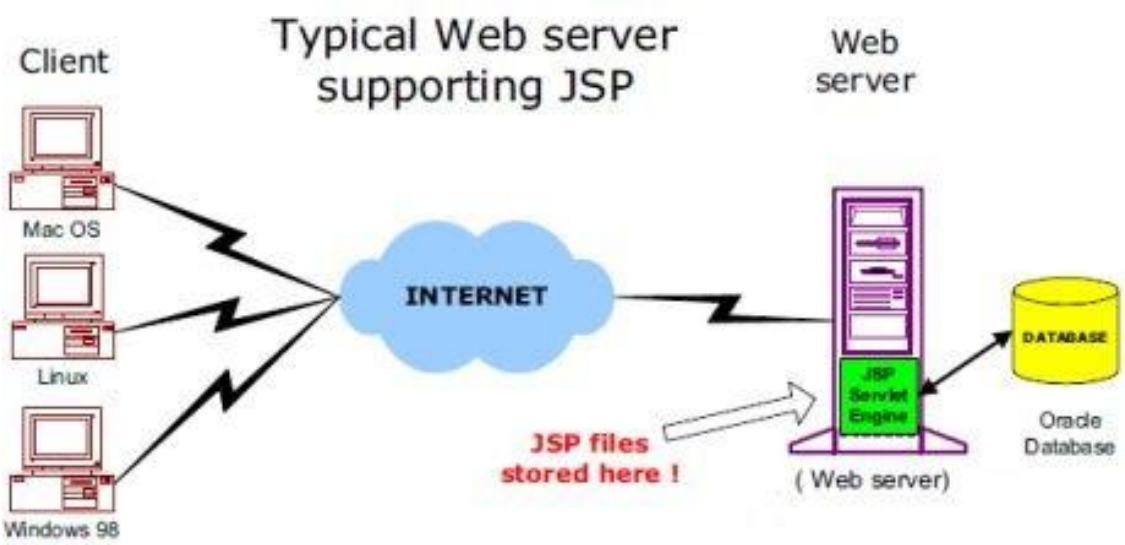
Why Use JSP?

Following is the list of other advantages of using JSP over other technologies:

- **vs. Active Server Pages (ASP):** The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.
- **vs. Pure Servlets:** It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.
- **vs. Server-Side Includes (SSI):** SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
- **vs. JavaScript:** JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.
- **vs. Static HTML:** Regular HTML, of course, cannot contain dynamic information.

JSP – Architecture

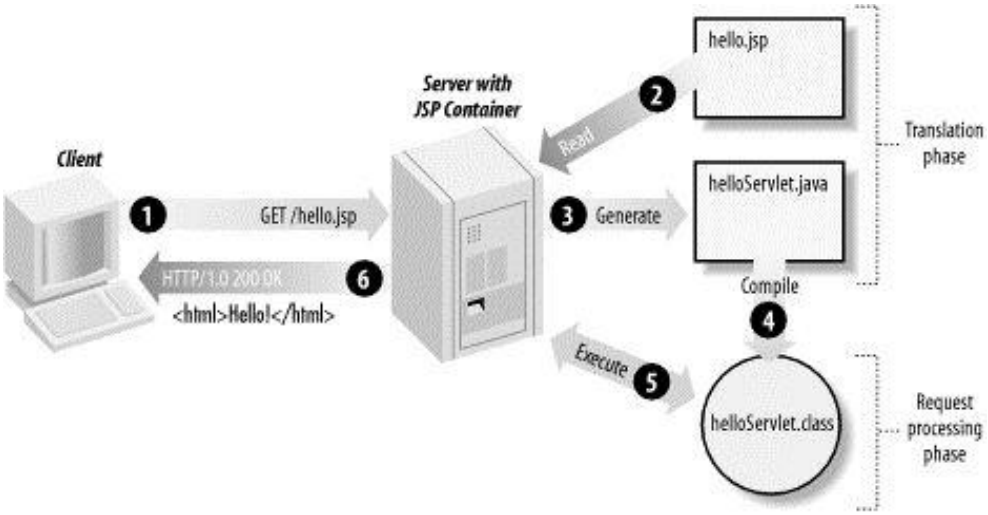
A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs



JSP Processing:

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

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



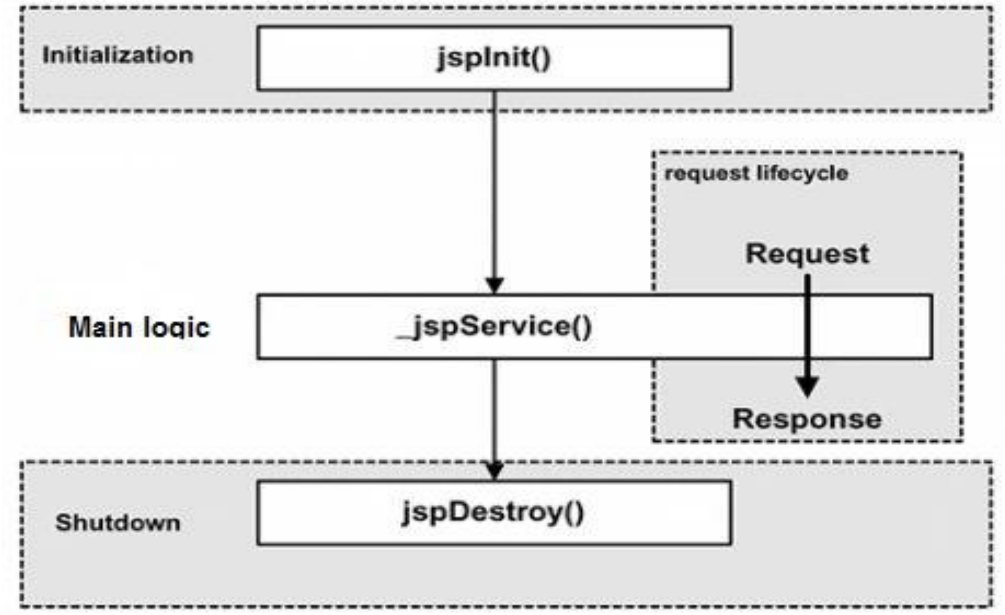
JSP - Life Cycle

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

The following are the paths followed by a JSP

- Compilation
- Initialization
- Execution

- Cleanup



JSP Compilation:

The compilation process involves three steps:

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

JSP Initialization:

When a container loads a JSP it invokes the **jspInit**() method before servicing any requests.

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

JSP Execution:

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

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

JSP Cleanup:

The **jspDestroy**() method is the JSP equivalent of the destroy method for servlets.

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

The Scriptlet:

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

<% **java** code fragment %>
OR
<jsp:scriptlet>
  Java code fragment
</jsp:scriptlet>

Let's write a small Scriptlet:

```
<html>
<head><title>Hello H2K Students</title></head>
<body>
<br>Hello H2K Students!<br/>
<%
out.println("Your IP address is " + request.getRemoteAddr());
%>
</body>
</html>
```

JSP Declarations:

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Following is the syntax of JSP Declarations:

<%! declaration; [declaration;]+ ... **%>**
OR

<jsp:declaration>
  code fragment
</jsp:declaration>

E.g.: <%! int i = 0; %>

JSP Expression:

The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

Following is the syntax of JSP Expression:

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

OR

```
<jsp:expression>
  Expression
</jsp:expression>
```

Following is the simple example for JSP Expression:

```
<html>
<head><title>A Comment Test</title></head>
<body>
<p>
   Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
</body>
</html>
```

JSP Comments:

Following is the syntax of JSP comments:

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

JSP - Directives

JSP directives provide directions and instructions to the container, telling it how to handle certain aspects of JSP processing.

There are three types of directive tag:

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

The *page* Directive:

The page directive is used to provide instructions to the container that pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Following is the syntax of page directive:

<%@ page attribute="value" %>

OR:

<jsp:directive.page attribute="value" />

Following is the list of attributes associated with page directive:

Attribute	Purpose	Example
buffer	Specifies a buffering model for the output stream.	`buffer="8kb"` `buffer="none"`
autoFlush	Controls the behavior of the servlet output buffer.	`autoFlush="false"` `autoFlush="true"`
contentType	Defines the character encoding scheme.	`contentType="text/xml"`
errorPage	Defines the URL of another JSP that reports on Java unchecked runtime exceptions.	`errorPage="MyErrorPage.jsp"`
isErrorPage	Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.	`isErrorPage="true"`
extends	Specifies a superclass that the generated servlet must extend	`extends="package.SomeClass"`
import	Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.	`import="java.sql.*"`
info	Defines a string that can be accessed with the servlet's getServletInfo() method.	`info="H2KInfosys JSP Page”`
isThreadSafe	Defines the threading model for the generated servlet.	`isThreadSafe="false"`
language	Defines the programming language used in the JSP page.	`language="java"`
session	Specifies whether or not the JSP page participates in HTTP sessions	`session="true"`
isELIgnored	Specifies whether or not EL expression within the JSP page will be ignored.	`isELIgnored="false"`
isScriptingEnabled	Determines if scripting elements are allowed for use.	`isScriptingEnabled="false"`

The *include* Directive:

The **include** directive is used to includes a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code include directives anywhere in your JSP page.

The general usage form of this directive is as follows:

```
<%@ include file="relative url" >
```

OR:

```
<jsp:directive.include file="relative url" />
```

Let's keep footer.jsp as:

```
<html><body>
<center>
        <p>Copyright © 2010</p>
</center>
</body>
</html>
```

Now in main.jsp, you can include this file as:

```
<html><body>
<center>
<p>Thanks for visiting H2KInfosys</p>
</center>
<%@ include file="footer.jsp" %>
</body>
</html>
```

The *taglib* Directive:

The JavaServer Pages API allows you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.

The taglib directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides a means for identifying the custom tags in your JSP page.

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

You can write XML equivalent of the above syntax as follows:

```
<jsp:directive.taglib uri="uri" prefix="prefixOfTag" />
```

We will be learning more about this in **Struts**.

JSP - Implicit Objects

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

JSP supports nine Implicit Objects which are listed below:

Object	Description
request	This is the **HttpServletRequest** object associated with the request.
response	This is the **HttpServletResponse** object associated with the response to the client.
out	This is the **PrintWriter** object used to send output to the client.
session	This is the **HttpSession** object associated with the request.
application	This is the **ServletContext** object associated with application context.
config	This is the **ServletConfig** object associated with the page.
pageContext	This encapsulates use of server-specific features like higher performance **JspWriters**.
page	This is simply a synonym for **this**, and is used to call the methods defined by the translated servlet class.
Exception	The **Exception** object allows the exception data to be accessed by designated JSP.

What's the difference between page and pageContext?

Page: The implicit variable page is of class java.lang.Object and it refers to instance of generated servlet.

PageContext: variable is of type javax.servlet.jsp.PageContext. The PageContext class is the abstract class and JSP engine vendor (let's say Tomcat) provides its concrete subclass.

- Store reference to implicit objects
- Provide method to get and set attributes in different scopes.
- Provide convenience methods for transferring request to other resources in web application.

JSP Scopes:

The *scope* of an object describes how widely it's available and who has access to it. For example, if an object is defined to have page scope, then it's available only for the duration of the current request on that page before being destroyed by the container. In this case, only the current page has access to this data, and no one else can read it. At the other end of the scale, if an object has application scope, then any page may use the data because it lasts for the duration of the application, which means until the container is switched off.

Page Scope:

Objects with page scope are accessible **only within the page in which they're created**. The data is valid only during the processing of the current response; once the response is sent back to the browser, the data is no longer valid. If the request is forwarded to another page or the browser makes another request as a result of a redirect, the data is also lost.

Defined as: *scope="page"*

Request Scope

Objects with request scope **are accessible from pages processing the same request in which they were created**. Once the container has processed the request, the data is released. Even if the request is forwarded to another page, the data is still available though not if a redirect is required.

Defined as: *scope="request"*

Session Scope

Objects with session scope **are accessible from pages processing requests that are in the same session** as the one in which they were created. A session is the time users spend using the application, which ends when they close their browser, when they go to another Web site, or when the application designer wants (after a logout, for instance). So, for example, when users log in, their username could be stored in the session and displayed on every page they access. This data lasts until they leave the Web site or log out.

Defined as: *scope="session"*

Application Scope

Objects with application scope are accessible from JSP pages that reside in the same application. This creates a global object that's available to all pages.

Application scope uses a single namespace, which means all your pages should be careful not to duplicate the names of application scope objects or change the values when they're likely to be read by another page (this is called thread safety). Application scope variables are typically created and populated when an application starts and then used as read-only for the rest of the application.

Defined as: *scope="application"*

Apart from the standard way, you can set pageContext attribute in particular scope as:

```
pageContext.setAttribute("AttributeName", "AttributeValue", PageContext.PAGE_SCOPE);
```

Difference between sendRedirect() and forward() in JSP Servlet

sendRedirect() - This is declared in **HttpServletResponse** Interface.

This method is used to redirect client request to some other location for further processing ,the new location is available on different server or different context / web container handle this and transfer the request using browser ,and this request is visible in browser as a new request. This is also called as **client side redirect**.

forward() - This method is declared in RequestDispatcher Interface.

This method is used to pass the request to another resource for further processing within the same server, another resource could be any servlet, jsp page any kind of file. This process is taken care by web container when we call forward method request is sent to another resource without the client being informed, which resource will handle the request it has been mention on requestDispatcher object which we can get by two ways either using ServletContext or Request. This is also called server side redirect.

```
RequestDispatcher rd = request.getRequestDispatcher("pathToResource");
rd.forward(request, response);
```

Or

```
RequestDispatcher rd = servletContext.getRequestDispatcher("/pathToResource");
rd.forward(request, response);
```

JSP - Standard Tag Library (JSTL)

The JavaServer Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates core functionality common to many JSP applications.

The JSTL tags can be classified, according to their functions, into following JSTL tag library groups that can be used when creating a JSP page:

Type	Declaration
Core Tags	<%@ taglib prefix="c" uri="*http://java.sun.com/jsp/jstl/core*" %>
Formatting tags	<%@ taglib prefix="fmt" uri="*http://java.sun.com/jsp/jstl/fmt*" %>
SQL tags	<%@ taglib prefix="sql" uri="*http://java.sun.com/jsp/jstl/sql*" %>
XML tags	<%@ taglib prefix="x" uri="*http://java.sun.com/jsp/jstl/xml*" %>
JSTL Functions	<%@ taglib prefix="fn" uri="*http://java.sun.com/jsp/jstl/functions*" %>

Core Tags:

Tag	Description	Example
<c:out >	Like <%= ... >, but for expressions.	*<c:out value="${'to print'}"/>*
<c:set >	Sets the result of an expression evaluation in a 'scope'	*<c:set var="salary" scope="session" value="${2000*2}"/>*
<c:remove >	Removes a scoped variable (from a particular scope, if specified).	*<c:remove var="salary"/>*
<c:catch>	Catches any Throwable that occurs in its body and optionally exposes it.	*<c:catch var ="catchException"> </c:catch>*
<c:if>	Simple conditional tag which evaluates its body if the supplied condition is true.	*<c:if test="${salary > 2000}">*
<c:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>	*<c:choose> <c:when test="${salary <= 0}"> </c:when> </c:choose>*
<c:when>	Subtag of <choose> that includes its body if its condition evaluates to 'true'.	*<c:when test="${salary <= 0}"> </c:when>*
<c:otherwise>	Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'.	*<c:otherwise> ---- </c:otherwise>*
<c:import>	Retrieves an absolute or relative URL and exposes its contents to the page, a String in 'var', or a Reader in 'varReader'.	*<c:import var="data" url="http://www.h2kinfosys.com"/>*
<c:forEach >	The basic iteration tag, accepting many different collection types and supporting sub setting and other functionality.	*<c:forEach var="i" begin="1" end="5"> Item <c:out value="${i}"/><p> </c:forEach>*
<c:forTokens>	Iterates over tokens, separated by the supplied delimiters.	*<c:forTokens items="Zara,nuha,roshy" delims="," var="name"> <c:out value="${name}"/><p> </c:forTokens>*
<c:param>	Adds a parameter to a containing 'import' tag's URL.	*<c:url value="/index.jsp" var="myURL"> <c:param name="trackingId" value="1234"/> <c:param name="reportType" value="summary"/> </c:url>*
<c:url>	Creates a URL with optional query parameters	
<c:redirect >	Redirects to a new URL.	*<c:redirect url="http://www.h2kinfosys.com"/>*

Formatting tags:

Tag	Description	Example
<fmt:formatNumber>	To render numerical value with specific precision or format.	<fmt:formatNumber value="${salary}" type="currency"/>
<fmt:parseNumber>	Parses the string representation of a number, currency, or percentage.	`<fmt:parseNumber var="i" type="number" value="${salary}" />`
<fmt:formatDate>	Formats a date and/or time using the supplied styles and pattern	`<fmt:formatDate type="both" value="${now}" />`
<fmt:parseDate>	Parses the string representation of a date and/or time	`<fmt:parseDate value="${now}" var="parsedEmpDate" pattern="dd-MM-yyyy" />`
<fmt:setBundle>	Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable.	`<fmt:setBundle basename="com.h2kinfosys.Example" var="lang"/>`
<fmt:timeZone>	Specifies the time zone for any time formatting or parsing actions nested in its body.	`<fmt:timeZone value="${zone}">` Where zone is java.util.TimeZone ID.
<fmt:setTimeZone>	Stores the given time zone in the time zone configuration variable	`<fmt:setTimeZone value="GMT-8" />`
<fmt:message>	To display an internationalized message.	`<fmt:setBundle basename="com.h2kinfosys.Example" var="lang"/>` `<fmt:message key="count.one" bundle="${lang}"/> `

The <fmt:***formatNumber***> tag has following attributes:

Attribute	Description	Required	Default
value	Numeric value to display	Yes	None
type	NUMBER, CURRENCY, or PERCENT	No	Number
pattern	Specify a custom formatting pattern for the output.	No	None
currencyCode	Currency code (for type="currency")	No	From the default locale
currencySymbol	Currency symbol (for type="currency")	No	From the default locale
groupingUsed	Whether to group numbers (TRUE or FALSE)	No	true
maxIntegerDigits	Maximum number of integer digits to	No	None

	print		
minIntegerDigits	Minimum number of integer digits to print	No	None
maxFractionDigits	Maximum number of fractional digits to print	No	None
minFractionDigits	Minimum number of fractional digits to print	No	None
var	Name of the variable to store the formatted number	No	Print to page
scope	Scope of the variable to store the formatted number	No	page

The <fmt:**parseNumber**> tag has following attributes:

Attribute	Description	Required	Default
value	Numeric value to read (parse)	No	Body
type	NUMBER, CURRENCY, or PERCENT	No	number
parseLocale	Locale to use when parsing the number	No	Default locale
integerOnly	Whether to parse to an integer (true) or floating-point number (false)	No	false
pattern	Custom parsing pattern	No	None
timeZone	Time zone of the displayed date	No	Default time zone
var	Name of the variable to store the parsed number	No	Print to page
scope	Scope of the variable to store the formatted number	No	page

The <fmt:**formatDate**> tag has following attributes:

Attribute	Description	Required	Default
value	Date value to display	Yes	None
type	DATE, TIME, or BOTH	No	date
dateStyle	FULL, LONG, MEDIUM, SHORT, or DEFAULT	No	default
timeStyle	FULL, LONG, MEDIUM, SHORT, or DEFAULT	No	default
pattern	Custom formatting pattern	No	None
timeZone	Time zone of the displayed date	No	Default time zone
var	Name of the variable to store the formatted date	No	Print to page
scope	Scope of the variable to store the formatted date	No	page

SQL tags:

Tag	Description
<sql:setDataSource>	Creates a simple DataSource suitable only for prototyping
<sql:query>	Executes the SQL query defined in its body or through the sql attribute.
<sql:update>	Executes the SQL update defined in its body or through the sql attribute.
<sql:param>	Sets a parameter in an SQL statement to the specified value.
<sql:dateParam>	Sets a parameter in an SQL statement to the specified java.util.Date value.
<sql:transaction >	Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction.

The <sql:setDataSource> tag has following attributes:

Attribute	Description	Required	Default
driver	Name of the JDBC driver class to be registered	No	None
url	JDBC URL for the database connection	No	None
user	Database username	No	None
password	Database password	No	None
password	Database password	No	None
dataSource	Database prepared in advance	No	None
var	Name of the variable to represent the database	No	Set default
scope	Scope of the variable to represent the database	No	Page

Example:

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ page import="java.util.Date,java.text.*" %>
<%@ 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>JSTL sql:dataParam Tag</title>
</head>
<body>

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

<%
Date DoB = new Date("2001/12/16");
int studentId = 100;
%>

<sql:update dataSource="${snapshot}" var="count">
  UPDATE Students SET dob = ? WHERE Id = ?
  <sql:dateParam value="<%=DoB%>" type="DATE" />
```

```
  <sql:param value="<%=studentId%>" />
</sql:update>

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

<table border="1" width="100%">
<tr>
  <th>Emp ID</th>
  <th>First Name</th>
  <th>Last Name</th>
  <th>DoB</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.dob}"/></td>
</tr>
</c:forEach>
</table></body></html>
```

XML tags:

Tag	Description
<x:out>	Like <%= ... >, but for XPath expressions.
<x:parse>	Use to parse XML data specified either via an attribute or in the tag body.
<x:set >	Sets a variable to the value of an XPath expression.
<x:if >	Evaluates a test XPath expression and if it is true, it processes its body. If the test condition is false, the body is ignored.
<x:forEach>	To loop over nodes in an XML document.
<x:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>
<x:when >	Subtag of <choose> that includes its body if its expression evalutes to 'true'
<x:otherwise>	Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'

Example:

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

<html>
<head>
  <title>JSTL x:choose Tags</title>
</head>
<body>
<h3>Books Info:</h3>

<c:set var="xmltext">
  <books>
```

```
    <book>
      <name>Padam History</name>
      <author>ZARA</author>
      <price>100</price>
    </book>
    <book>
      <name>Great Mistry</name>
      <author>NUHA</author>
      <price>2000</price>
    </book>
  </books>
</c:set>

<x:parse xml="${xmltext}" var="output"/>

<x:if select="$output//book">
   Document has at least one <book> element.
</x:if>
<br />
<x:if select="$output/books[1]/book/price > 100">
   Book prices are very high
</x:if>

<x:set var="fragment" select="$output//book"/>
<b>The price of the second book</b>:
<c:out value="${fragment}" />

<x:choose>
   <x:when select="$output//book/author = 'ZARA'">
      Book is written by ZARA
   </x:when>
   <x:when select="$output//book/author = 'NUHA'">
      Book is written by NUHA
   </x:when>
   <x:otherwise>
      Unknown author.
   </x:otherwise>
</x:choose>

<ul class="list">
<x:forEach select="$output/books/book/name" var="item">
   <li>Book Name: <x:out select="$item" /></li>
</x:forEach>
</ul>
</body>
</html>
```

JSTL Functions:

Function	Description
fn:contains()	Tests if an input string contains the specified substring.
fn:containsIgnoreCase()	Tests if an input string contains the specified substring in a case insensitive way.
fn:endsWith()	Tests if an input string ends with the specified suffix.

fn:indexOf()	Returns the index withing a string of the first occurrence of a specified substring.
fn:join()	Joins all elements of an array into a string.
fn:length()	Returns the number of items in a collection, or the number of characters in a string.
fn:replace()	Returns a string resulting from replacing in an input string all occurrences with a given string.
fn:split()	Splits a string into an array of substrings.
fn:startsWith()	Tests if an input string starts with the specified prefix.
fn:substring()	Returns a subset of a string.
fn:substringAfter()	Returns a subset of a string following a specific substring.
fn:substringBefore()	Returns a subset of a string before a specific substring.
fn:toLowerCase()	Converts all of the characters of a string to lower case.
fn:toUpperCase()	Converts all of the characters of a string to upper case.
fn:trim()	Removes white spaces from both ends of a string.

Example:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
<head>
<title>Using JSTL Functions</title>
</head>
<body>

<c:set var="string1" value="This is first String."/>
<c:set var="string2" value="This <abc>is second String.</abc>"/>

<p>Index (1) : ${fn:indexOf(string1, "first")}</p>
<p>Index (2) : ${fn:indexOf(string2, "second")}</p>

</body>
</html>
```