**JSP (J**ava **S**erver **P**ages**)**

Java Server 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 %>.

A Java Server Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

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.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages and sharing information between requests, pages etc.

## Why Use JSP?

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

* Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
* JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
* Java Server Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.
* JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Finally, JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

## Advantages of JSP:

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

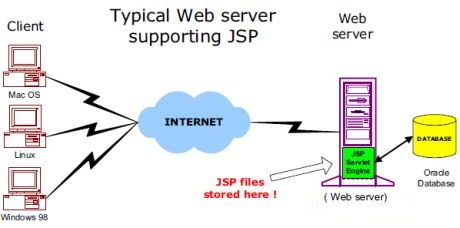
* **vs. Active Server Pages (ASP):** The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.
* **vs. Pure Servlets:** 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

The web server needs a JSP engine ie. Container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. This tutorial makes use of Apache which has built-in JSP container to support JSP pages development.

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

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

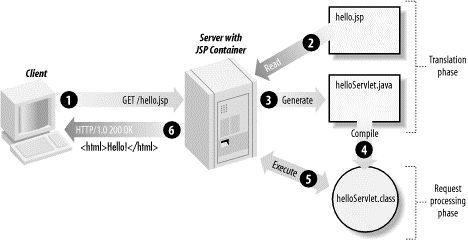


## JSP Processing:

The following steps explain how the web server creates the 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.

All the above mentioned steps can be shown below in the following diagram:



Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with other scripting languages (such as PHP) and therefore faster.

So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

# JSP - Life Cycle

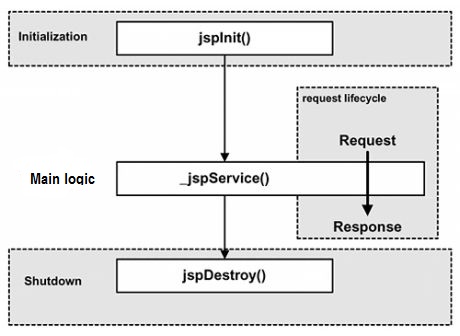
The key to understanding the low-level functionality of JSP is to understand the simple life cycle they follow.

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

The four major phases of JSP life cycle are very similar to Servlet Life Cycle and they are as follows:



## JSP Compilation:

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps:

* + Parsing the JSP.
  + Turning the JSP into a servlet.
  + Compiling the servlet.

## JSP Initialization:

When a container loads a JSP it invokes the jspInit() method before servicing any requests. If you need to perform JSP-specific initialization, override the jspInit() method:

public void jspInit(){

// Initialization code...

}

Typically initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

## JSP Execution:

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **\_jspService()** method in the JSP.

The \_jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows:

void \_jspService(HttpServletRequest request,

HttpServletResponse response)

{

// Service handling code...

}

The \_jspService() method of a JSP is invoked 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.

## JSP Cleanup:

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

The jspDestroy() method has the following form:

public void jspDestroy()

{

// Your cleanup code goes here.

}

## JSP Directives:

A JSP directive affects the overall structure of the servlet class. It usually has the following form:

<%@ directive attribute="value" %>

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 |

## 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 and there are following JSP actions available:

|  |  |
| --- | --- |
| **Syntax** | **Purpose** |
| jsp:include | Includes a file at the time the page is requested |
| jsp:include | Includes a file at the time the page is requested |
| jsp:useBean | Finds or instantiates a JavaBean |
| jsp:setProperty | Sets the property of a JavaBean |
| jsp:getProperty | Inserts the property of a JavaBean into the output |
| jsp:forward | Forwards the requester to a new page |
| jsp:plugin | Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin |
| jsp:element | Defines XML elements dynamically. |
| jsp:attribute | Defines dynamically defined XML element's attribute. |
| jsp:body | Defines dynamically defined XML element's body. |
| jsp:text | Use to write template text in JSP pages and documents. |

We would explain JSP actions in separate chapter [JSP - Actions](http://www.tutorialspoint.com/jsp/jsp_actions.htm)

## JSP Implicit Objects:

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

|  |  |
| --- | --- |
| **Objects** | **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. |

## 

## 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 - 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. |

# JSP - Standard Tag Library (JSTL) Tutorial

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

JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating existing custom tags with JSTL tags.

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:

* **Core Tags**
* **Formatting tags**
* **SQL tags**
* **XML tags**
* **JSTL Functions**

## Install JSTL Library:

If you are using Apache Tomcat container then follow the following two simple steps:

* Download the binary distribution from [Apache Standard Taglib](http://tomcat.apache.org/taglibs/index.html) and unpack the compressed file.
* To use the Standard Taglib from its Jakarta Taglibs distribution, simply copy the JAR files in the distribution's 'lib' directory to your application's webapps\ROOT\WEB-INF\lib directory.

To use any of the libraries, you must include a <taglib> directive at the top of each JSP that uses the library.

## Core Tags:

The core group of tags are the most frequently used JSTL tags. Following is the syntax to include JSTL Core library in your JSP:

<%@ taglib prefix="c"

uri="http://java.sun.com/jsp/jstl/core" %>

There are following Core JSTL Tags:

|  |  |
| --- | --- |
| **Tag** | **Description** |
| [<c:out >](http://www.tutorialspoint.com/jsp/jstl_core_out_tag.htm) | Like <%= ... >, but for expressions. |
| [<c:set >](http://www.tutorialspoint.com/jsp/jstl_core_set_tag.htm) | Sets the result of an expression evaluation in a 'scope' |
| [<c:remove >](http://www.tutorialspoint.com/jsp/jstl_core_remove_tag.htm) | Removes a scoped variable (from a particular scope, if specified). |
| [<c:catch>](http://www.tutorialspoint.com/jsp/jstl_core_catch_tag.htm) | Catches any Throwable that occurs in its body and optionally exposes it. |
| [<c:if>](http://www.tutorialspoint.com/jsp/jstl_core_if_tag.htm) | Simple conditional tag which evaluates its body if the supplied condition is true. |
| [<c:choose>](http://www.tutorialspoint.com/jsp/jstl_core_choose_tag.htm) | Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise> |
| [<c:when>](http://www.tutorialspoint.com/jsp/jstl_core_choose_tag.htm) | Subtag of <choose> that includes its body if its condition evalutes to 'true'. |
| [<c:otherwise >](http://www.tutorialspoint.com/jsp/jstl_core_choose_tag.htm) | Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'. |
| [<c:import>](http://www.tutorialspoint.com/jsp/jstl_core_import_tag.htm) | Retrieves an absolute or relative URL and exposes its contents to either the page, a String in 'var', or a Reader in 'varReader'. |
| [<c:forEach >](http://www.tutorialspoint.com/jsp/jstl_core_foreach_tag.htm) | The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality . |
| [<c:forTokens>](http://www.tutorialspoint.com/jsp/jstl_core_foreach_tag.htm) | Iterates over tokens, separated by the supplied delimeters. |
| [<c:param>](http://www.tutorialspoint.com/jsp/jstl_core_param_tag.htm) | Adds a parameter to a containing 'import' tag's URL. |
| [<c:redirect >](http://www.tutorialspoint.com/jsp/jstl_core_redirect_tag.htm) | Redirects to a new URL. |
| [<c:url>](http://www.tutorialspoint.com/jsp/jstl_core_url_tag.htm) | Creates a URL with optional query parameters |

## Formatting tags:

The JSTL formatting tags are used to format and display text, the date, the time, and numbers for internationalized Web sites. Following is the syntax to include Formatting library in your JSP:

<%@ taglib prefix="fmt"

uri="http://java.sun.com/jsp/jstl/fmt" %>

Following is the list of Formatting JSTL Tags:

|  |  |
| --- | --- |
| **Tag** | **Description** |
| [<fmt:formatNumber>](http://www.tutorialspoint.com/jsp/jstl_format_formatnumber_tag.htm) | To render numerical value with specific precision or format. |
| [<fmt:parseNumber>](http://www.tutorialspoint.com/jsp/jstl_format_parsenumber_tag.htm) | Parses the string representation of a number, currency, or percentage. |
| [<fmt:formatDate>](http://www.tutorialspoint.com/jsp/jstl_format_formatdate_tag.htm) | Formats a date and/or time using the supplied styles and pattern |
| [<fmt:parseDate>](http://www.tutorialspoint.com/jsp/jstl_format_parsedate_tag.htm) | Parses the string representation of a date and/or time |
| [<fmt:bundle>](http://www.tutorialspoint.com/jsp/jstl_format_bundle_tag.htm) | Loads a resource bundle to be used by its tag body. |
| [<fmt:setLocale>](http://www.tutorialspoint.com/jsp/jstl_format_setlocale_tag.htm) | Stores the given locale in the locale configuration variable. |
| [<fmt:setBundle>](http://www.tutorialspoint.com/jsp/jstl_format_setbundle_tag.htm) | Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable. |
| [<fmt:timeZone>](http://www.tutorialspoint.com/jsp/jstl_format_timezone_tag.htm) | Specifies the time zone for any time formatting or parsing actions nested in its body. |
| [<fmt:setTimeZone>](http://www.tutorialspoint.com/jsp/jstl_format_settimezone_tag.htm) | Stores the given time zone in the time zone configuration variable |
| [<fmt:message>](http://www.tutorialspoint.com/jsp/jstl_format_message_tag.htm) | To display an internationalized message. |
| [<fmt:requestEncoding>](http://www.tutorialspoint.com/jsp/jstl_format_requestencoding_tag.htm) | Sets the request character encoding |

## SQL tags:

The JSTL SQL tag library provides tags for interacting with relational databases (RDBMSs) such as Oracle, mySQL, or Microsoft SQL Server.

Following is the syntax to include JSTL SQL library in your JSP:

<%@ taglib prefix="sql"

uri="http://java.sun.com/jsp/jstl/sql" %>

Following is the list of SQL JSTL Tags:

|  |  |
| --- | --- |
| **Tag** | **Description** |
| [<sql:setDataSource>](http://www.tutorialspoint.com/jsp/jstl_sql_setdatasource_tag.htm) | Creates a simple DataSource suitable only for prototyping |
| [<sql:query>](http://www.tutorialspoint.com/jsp/jstl_sql_query_tag.htm) | Executes the SQL query defined in its body or through the sql attribute. |
| [<sql:update>](http://www.tutorialspoint.com/jsp/jstl_sql_update_tag.htm) | Executes the SQL update defined in its body or through the sql attribute. |
| [<sql:param>](http://www.tutorialspoint.com/jsp/jstl_sql_param_tag.htm) | Sets a parameter in an SQL statement to the specified value. |
| [<sql:dateParam>](http://www.tutorialspoint.com/jsp/jstl_sql_dateparam_tag.htm) | Sets a parameter in an SQL statement to the specified java.util.Date value. |
| [<sql:transaction >](http://www.tutorialspoint.com/jsp/jstl_sql_transaction_tag.htm) | Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction. |

## XML tags:

The JSTL XML tags provide a JSP-centric way of creating and manipulating XML documents. Following is the syntax to include JSTL XML library in your JSP.

The JSTL XML tag library has custom tags for interacting with XML data. This includes parsing XML, transforming XML data, and flow control based on XPath expressions.

<%@ taglib prefix="x"

uri="http://java.sun.com/jsp/jstl/xml" %>

Before you proceed with the examples, you would need to copy following two XML and XPath related libraries into your <Tomcat Installation Directory>\lib:

* **XercesImpl.jar:** Download it from <http://www.apache.org/dist/xerces/j/>
* **xalan.jar:** Download it from <http://xml.apache.org/xalan-j/index.html>

Following is the list of XML JSTL Tags:

|  |  |
| --- | --- |
| **Tag** | **Description** |
| [<x:out>](http://www.tutorialspoint.com/jsp/jstl_xml_out_tag.htm) | Like <%= ... >, but for XPath expressions. |
| [<x:parse>](http://www.tutorialspoint.com/jsp/jstl_xml_parse_tag.htm) | Use to parse XML data specified either via an attribute or in the tag body. |
| [<x:set >](http://www.tutorialspoint.com/jsp/jstl_xml_set_tag.htm) | Sets a variable to the value of an XPath expression. |
| [<x:if >](http://www.tutorialspoint.com/jsp/jstl_xml_if_tag.htm) | 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>](http://www.tutorialspoint.com/jsp/jstl_xml_foreach_tag.htm) | To loop over nodes in an XML document. |
| [<x:choose>](http://www.tutorialspoint.com/jsp/jstl_xml_choose_tag.htm) | Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise> |
| [<x:when >](http://www.tutorialspoint.com/jsp/jstl_xml_choose_tag.htm) | Subtag of <choose> that includes its body if its expression evalutes to 'true' |
| [<x:otherwise >](http://www.tutorialspoint.com/jsp/jstl_xml_choose_tag.htm) | Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false' |
| [<x:transform >](http://www.tutorialspoint.com/jsp/jstl_xml_transform_tag.htm) | Applies an XSL transformation on a XML document |
| [<x:param >](http://www.tutorialspoint.com/jsp/jstl_xml_param_tag.htm) | Use along with the transform tag to set a parameter in the XSLT stylesheet |

## JSTL Functions:

JSTL includes a number of standard functions, most of which are common string manipulation functions. Following is the syntax to include JSTL Functions library in your JSP:

<%@ taglib prefix="fn"

uri="http://java.sun.com/jsp/jstl/functions" %>

Following is the list of JSTL Functions:

|  |  |
| --- | --- |
| **Function** | **Description** |
| [fn:contains()](http://www.tutorialspoint.com/jsp/jstl_function_contains.htm) | Tests if an input string contains the specified substring. |
| [fn:containsIgnoreCase()](http://www.tutorialspoint.com/jsp/jstl_function_containsignorecase.htm) | Tests if an input string contains the specified substring in a case insensitive way. |
| [fn:endsWith()](http://www.tutorialspoint.com/jsp/jstl_function_endswith.htm) | Tests if an input string ends with the specified suffix. |
| [fn:escapeXml()](http://www.tutorialspoint.com/jsp/jstl_function_escapexml.htm) | Escapes characters that could be interpreted as XML markup. |
| [fn:indexOf()](http://www.tutorialspoint.com/jsp/jstl_function_indexof.htm) | Returns the index withing a string of the first occurrence of a specified substring. |
| [fn:join()](http://www.tutorialspoint.com/jsp/jstl_function_join.htm) | Joins all elements of an array into a string. |
| [fn:length()](http://www.tutorialspoint.com/jsp/jstl_function_length.htm) | Returns the number of items in a collection, or the number of characters in a string. |
| [fn:replace()](http://www.tutorialspoint.com/jsp/jstl_function_replace.htm) | Returns a string resulting from replacing in an input string all occurrences with a given string. |
| [fn:split()](http://www.tutorialspoint.com/jsp/jstl_function_split.htm) | Splits a string into an array of substrings. |
| [fn:startsWith()](http://www.tutorialspoint.com/jsp/jstl_function_startswith.htm) | Tests if an input string starts with the specified prefix. |
| [fn:substring()](http://www.tutorialspoint.com/jsp/jstl_function_substring.htm) | Returns a subset of a string. |
| [fn:substringAfter()](http://www.tutorialspoint.com/jsp/jstl_function_substringafter.htm) | Returns a subset of a string following a specific substring. |
| [fn:substringBefore()](http://www.tutorialspoint.com/jsp/jstl_function_substringbefore.htm) | Returns a subset of a string before a specific substring. |
| [fn:toLowerCase()](http://www.tutorialspoint.com/jsp/jstl_function_tolowercase.htm) | Converts all of the characters of a string to lower case. |
| [fn:toUpperCase()](http://www.tutorialspoint.com/jsp/jstl_function_touppercase.htm) | Converts all of the characters of a string to upper case. |
| [fn:trim()](http://www.tutorialspoint.com/jsp/jstl_function_trim.htm) | Removes white spaces from both ends of a string. |