# JSP Tutorial

**JSP** technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

## Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

1. Extension to Servlet- JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
2. Easy to maintain- JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.
3. Fast Development: No need to recompile and redeploy
4. If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.
5. Less code than Servlet- In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

## Disadvantages of JSP

1. JSP pages can become difficult to maintain as they often mix presentation and business logic.
2. JSP pages can be less performant than other Java-based technologies such as Servlets because they are translated into Servlets at runtime.
3. JSP does not provide as much control over the generated HTML as other technologies, making it harder to create complex layouts and designs.
4. Some developers find JSP's use of Java code within the HTML difficult to read and understand.
5. JSP requires Java Servlet container to run, where as some other alternatives can run on other web server environments.

# The Lifecycle of a JSP Page

The JSP pages follow these phases:

1. Translation of JSP Page
2. Compilation of JSP Page
3. Classloading (the classloader loads class file)
4. Instantiation (Object of the Generated Servlet is created).
5. Initialization ( the container invokes jspInit() method).
6. Request processing ( the container invokes \_jspService() method).
7. Destroy ( the container invokes jspDestroy() method).

*Note: jspInit(), \_jspService() and jspDestroy() are the life cycle methods of JSP.*

# The JSP API consists of two packages:

1. javax.servlet.jsp
2. javax.servlet.jsp.tagext

## javax.servlet.jsp package

The javax.servlet.jsp package has two interfaces and classes.The two interfaces are as follows:

1. JspPage
2. HttpJspPage

The classes are as follows:

1. JspWriter
2. PageContext
3. JspFactory
4. JspEngineInfo
5. JspException
6. JspError

### The JspPage interface

According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface. It provides two life cycle methods.

**Methods of JspPage interface**

1. public void jspInit(): It is invoked only once during the life cycle of the JSP when JSP page is requested firstly. It is used to perform initialization. It is same as the init() method of Servlet interface.
2. public void jspDestroy(): It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.

# JSP Tags

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what are the scripting elements first.

JSP Scripting elements

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

1. scriptlet tag
2. expression tag
3. declaration tag

## JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

*<% java source code %>*

Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

<html>

<body>

<% out.print("welcome to jsp"); %>

</body>

</html>

In JSP (JavaServer Pages), the scriptlet tag <% ... %> is used to include Java code within a JSP page. The code within the scriptlet tag is executed on the server side, before the result is sent to the client.

The scriptlet tag is used to include small blocks of Java code that perform some logic or operations that are required by the JSP page. For example, the following scriptlet tag increments a variable named "counter" by 1:

<% counter++; %>

It can also be used to call methods or to include control flow statements, for example:

<%

if(user.isLoggedIn()){

out.println("Welcome, " + user.getName());

}else{

out.println("Please login to continue.");

}

%>

It's important to note that scriptlet tags and expressions tags should be used sparingly and judiciously in JSP pages, as they can make the code difficult to maintain and less readable, and it's recommended to use JSP Standard Tag Library (JSTL) and custom tags for most of the logic required in the JSP pages.

## JSP expression tag

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

**Syntax of JSP expression tag**

*<%= statement %>*

In JSP (JavaServer Pages), the expression tag <%= ... %> is used to insert the result of an expression or a variable into the output stream, which is sent to the client. The expression within the tag is evaluated, converted to a string, and then inserted into the output. For example, if a JSP page has a variable named "username" with the value "John", the following expression tag would insert the value of the "username" variable into the HTML output sent to the client:

<p>Hello, <%= username %>!</p>

This would be rendered as:

<p>Hello, John!</p>

It's important to note that the expression tag is executed on the server side, and it's results are sent to the client.

**Example of JSP expression tag**

In this example of jsp expression tag, we are simply displaying a welcome message.

<html>

<body>

<%= "welcome to jsp" %>

</body>

</html>

## JSP Declaration Tag

The JSP declaration tag is used to declare fields and methods. The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.

**Syntax of JSP declaration tag**

The syntax of the declaration tag is as follows:

<%! field or method declaration %>

In JSP (JavaServer Pages), the declaration tag <%! ... %> is used to declare variables and methods that will be available to the entire JSP page, as opposed to the expression tag <%= ... %> which is used to insert the result of an expression or a variable into the output stream.

The declaration tag is used to declare member variables and methods that are used in the JSP page. These variables and methods are defined at the class level, rather than in a particular method, so they can be accessed from any method in the JSP page. For example, the following declaration tag declares a variable named "counter" with an initial value of 0:

<%! int counter = 0; %>

It can also be used to declare methods, for example:

<%! public int addTwoNumbers(int a, int b) { return a+b;} %>

It's important to note that the declarations are translated into fields and methods of the generated servlet class and that the declaration tag is executed on the server side, at the time when the JSP is translated into a servlet.

## Difference between JSP Scriptlet tag and Declaration tag

|  |  |
| --- | --- |
| **Jsp Scriptlet Tag** | **Jsp Declaration Tag** |
| The jsp scriptlet tag can only declare variables not methods. | The jsp declaration tag can declare variables as well as methods. |
| The declaration of scriptlet tag is placed inside the \_jspService() method. | The declaration of jsp declaration tag is placed outside the \_jspService() method. |

# JSP Implicit objects

In JSP (JavaServer Pages), implicit objects are objects that are automatically created by the JSP container and are available to the JSP page without the need to be explicitly declared or instantiated. These objects can be used in any part of the JSP page, including scriptlets, expressions, and standard and custom tags.

Here are some of the most commonly used JSP implicit objects:

1. request: Represents the HTTP request made by the client. It can be used to access request headers and parameters.
2. response: Represents the HTTP response that will be sent to the client. It can be used to set response headers and write to the response body.
3. session: Represents the HTTP session associated with the request. It can be used to store and retrieve information about the current user's session.
4. application: Represents the servlet context, which is the container for the web application. It can be used to share data and resources across all JSP pages and servlets in the application.
5. out: Represents the output stream used to send the response to the client. It can be used to write to the response body.
6. config: Represents the servlet configuration for the JSP page. It can be used to access initialization parameters defined in the web.xml file.
7. page: Represents the current JSP page. It can be used to access the implicit variables this and super.
8. pageContext: Represents the context for the current JSP page. It provides access to all the other implicit objects and provides support for several other features.

It's important to note that the availability of these objects depends on the context in which the JSP page is executed, for example, session object is not available in some situations like if the session is disabled in the browser.

# JSP page directives

In JSP (JavaServer Pages), directives are used to provide instructions to the JSP container on how to handle the JSP page. There are three types of JSP directives:

1. **page directive**: This directive is used to provide information about the JSP page, such as the language used, the content type, and the error page to be used if an exception occurs.
2. **include directive**: This directive is used to include the content of another resource, such as a static HTML file or another JSP page, in the current JSP page.
3. **taglib directive**: This directive is used to import a tag library and make the tags defined in the library available for use in the JSP page.

JSP directives are always enclosed in <%@ and %> and are not executed when the JSP is executed. They provide information to the JSP container, which uses that information to properly process the JSP page.

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

This tells the JSP container that the page is written in Java, the content type is text/html and the character encoding is UTF-8

<%@ include file="header.jsp" %>

This tells the JSP container to include the content of the file "header.jsp" at this point in the current JSP page.

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

This tells the JSP container to import the JSTL Core library and to associate it with the prefix "c".

**Attributes of JSP page directive**

1. language: Specifies the programming language used in the JSP page.
2. extends: Specifies a superclass for the generated servlet.
3. import: Allows for the import of classes or packages.
4. session: Indicates whether or not the page participates in a session.
5. buffer: Specifies the buffer size for the JSP output.
6. autoFlush: Indicates whether or not the buffer should be automatically flushed.
7. isThreadSafe: Indicates whether or not the page is thread-safe.
8. info: Provides a string that can be used as a comment in the generated servlet's Java code.
9. errorPage: Specifies a URL for the page to handle errors that occur during the execution of the JSP.
10. isErrorPage: Indicates whether or not the page is an error page.
11. contentType: The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".
12. isELIgnored: We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. Expression Language is enabled by default. We see Expression Language later.

<%@ page isELIgnored="true" %>//Now EL will be ignored

1. pageEncoding :

These are some of the attributes of JSP page directive, there are many other attributes as well.

# JSP Include directive

The JSP include directive is used to include the content of another resource (such as another JSP or HTML file) in the current JSP page. The included resource can be a static or dynamic file. The syntax for the include directive is as follows:

<%@ include file="relative\_URL" %>

Where "relative\_URL" is the path to the resource you want to include, relative to the current JSP page.

For example, if you want to include the content of a file called "header.jsp" in the current JSP page, you would use the following syntax:

<%@ include file="header.jsp" %>

When the JSP page is executed, the content of the included file will be inserted at the location of the include directive. This can be useful for reusing common elements, such as headers and footers, across multiple pages.

Note: The include directive is processed at the translation-time, the included resource is included as a static text in the generated servlet.

It's also worth noting that there's another include action jsp:include, which allows you to include resources dynamically at runtime, rather than at translation time.

# JSP Exception handling

In JSP (JavaServer Pages), exception handling can be done using a combination of the JSP try and catch statements, and the Java try and catch statements. The JSP try statement is used to enclose the code that may throw an exception, and the JSP catch statement is used to catch and handle the exception. The Java try and catch statements can also be used within a JSP to catch and handle exceptions thrown by Java code called from the JSP.

Here is an example of exception handling in a JSP using the try and catch statements:

<%@ page errorPage="error.jsp" %>

<%

try {

// code that may throw an exception

} catch (Exception e) {

throw new ServletException(e);

}

%>

In this example, the errorPage attribute of the page directive is set to "error.jsp", which is the JSP that will be displayed in case an exception is caught. The try block encloses the code that may throw an exception, and the catch block catches any exception of type Exception that is thrown. The exception is then re-thrown as a ServletException so that the error page can handle it.

JSP also provide error-page element in web.xml to handle the exception globally for all JSP pages.

<error-page>

<exception-type>java.lang.Throwable</exception-type>

<location>/error.jsp</location>

</error-page>

This will redirect to error.jsp whenever any exception is thrown by any jsp page.

# JSP Action tags

JSP action tags are special tags that provide a way for a JSP page to communicate with the web container or other resources. They are used to perform a variety of tasks such as including other resources, forwarding to another page, or interacting with JavaBeans. Examples of JSP action tags include jsp:include, jsp:forward, jsp:useBean, and jsp:setProperty. These tags are processed by the web container at runtime, and their actions are performed before the JSP page is rendered to the client.

* jsp:forward-forwards the request and response to another resource.
* jsp:include-includes another resource.
* jsp:useBean-creates or locates bean object.
* jsp:setProperty-sets the value of property in bean object.
* jsp:getProperty-prints the value of property of the bean.
* jsp:plugin-embeds another components such as applet.
* jsp:param-sets the parameter value. It is used in forward and include mostly.
* jsp:fallback-can be used to print the message if plugin is working. It is used in jsp:plugin.

## jsp:forward action tag

The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

Syntax of jsp:forward action tag without parameter

<jsp:forward page="relativeURL | <%= expression %>" />

Syntax of jsp:forward action tag with parameter

<jsp:forward page="relativeURL | <%= expression %>">

<jsp:param name="parametername" value="parametervalue | <%=expression%>" />

</jsp:forward>

## jsp:include action tag

The jsp:include action tag is used to include the content of another resource it may be jsp, html or servlet.

The jsp include action tag includes the resource at request time so it is better for dynamic pages because there might be changes in future. The jsp:include tag can be used to include static as well as dynamic pages.

|  |  |
| --- | --- |
| **JSP include directive** | **JSP include action** |
| includes resource at translation time. | includes resource at request time. |
| better for static pages. | better for dynamic pages. |
| includes the original content in the generated servlet. | calls the include method. |

**Advantage of jsp:include action tag**

Code reusability : We can use a page many times such as including header and footer pages in all pages. So it saves a lot of time.

Syntax of jsp:include action tag without parameter

<jsp:include page="relativeURL | <%= expression %>" />

Syntax of jsp:include action tag with parameter

<jsp:include page="relativeURL | <%= expression %>">

<jsp:param name="parametername" value="parametervalue | <%=expression%>" />

</jsp:include>

## JavaBean

A JavaBean is a Java class that should follow the following conventions:

* It should have a no-arg constructor.
* It should be Serializable.
* It should provide methods to set and get the values of the properties, known as getter and setter methods.

**Why use JavaBean?**

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance.

**How to access the JavaBean class?**

To access the JavaBean class, we should use getter and setter methods.

Note: There are two ways to provide values to the object. One way is by constructor and second is by setter method.

**JavaBean Properties**

A JavaBean property is a named feature that can be accessed by the user of the object. The feature can be of any Java data type, containing the classes that you define.

A JavaBean property may be read, write, read-only, or write-only. JavaBean features are accessed through two methods in the JavaBean's implementation class:

1. **getPropertyName ()**

For example, if the property name is firstName, the method name would be getFirstName() to read that property. This method is called the accessor.

1. **setPropertyName ()**

For example, if the property name is firstName, the method name would be setFirstName() to write that property. This method is called the mutator.

### Advantages and disadvantages of JavaBean

The following are the advantages of JavaBean:

1. The JavaBean properties and methods can be exposed to another application.
2. It provides an easiness to reuse the software components.

The following are the disadvantages of JavaBean:

1. JavaBeans are mutable. So, it can't take advantages of immutable objects.
2. Creating the setter and getter method for each property separately may lead to the boilerplate code.

## jsp:useBean action tag

The jsp:useBean action tag is used to locate or instantiate a bean class. If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of bean is not created, it instantiates the bean.

**Syntax of jsp:useBean action tag**

<jsp:useBean id= "instanceName" scope= "page | request | session | application" class= "packageName.className" type= "packageName.className" beanName="packageName.className | <%= expression >" > </jsp:useBean>

**Attributes and Usage of jsp:useBean action tag**

1. id: is used to identify the bean in the specified scope.
2. scope: represents the scope of the bean. It may be page, request, session or application. The default scope is page.

* page: specifies that you can use this bean within the JSP page. The default scope is page.
* request: specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
* session: specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
* application: specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.

1. class: instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.
2. type: provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attribute. If you use it without class or beanName, no bean is instantiated.
3. beanName: instantiates the bean using the java.beans.Beans.instantiate() method.

## Jsp:setProperty and jsp:getProperty action tags

The jsp:setProperty and jsp:getProperty action tags are used in JavaServer Pages (JSP) to set and retrieve the properties of JavaBeans components.

The jsp:setProperty tag is used to set the value of a property in a JavaBean, and it has the following attributes:

1. name: the name of the JavaBean
2. property: the name of the property to set
3. value: the value to set the property to

For example, the following code sets the value of the "message" property in a JavaBean named "myBean" to "Hello, World!":

<jsp:setProperty name="myBean" property="message" value="Hello, World!"/>

The jsp:getProperty tag is used to retrieve the value of a property from a JavaBean, and it has the following attributes:

1. name: the name of the JavaBean
2. property: the name of the property to retrieve

For example, the following code retrieves the value of the "message" property from a JavaBean named "myBean" and writes it to the JSP output:

<jsp:getProperty name="myBean" property="message"/>

Both jsp:setProperty and jsp:getProperty are used on JSP page to interact with the JavaBean, which can be created and instantiated in the servlet container, such as Tomcat.

**Syntax of jsp:setProperty action tag**

<jsp:setProperty name="instanceOfBean" property= "\*" | property="propertyName" param="parameterName" |

property="propertyName" value="{ string | <%= expression %>}" />

**Example of jsp:setProperty action tag if you have to set all the values of incoming request in the bean**

<jsp:setProperty name="bean" property="\*" />

**Example of jsp:setProperty action tag if you have to set value of the incoming specific property**

<jsp:setProperty name="bean" property="username" />

**Example of jsp:setProperty action tag if you have to set a specific value in the property**

<jsp:setProperty name="bean" property="username" value="Kumar" />

**jsp:getProperty action tag -**The jsp:getProperty action tag returns the value of the property.

**Syntax of jsp:getProperty action tag**

<jsp:getProperty name="instanceOfBean" property="propertyName" />

Simple example of jsp:getProperty action tag

<jsp:getProperty name="obj" property="name" />