JSP

**JavaServer Pages (JSP):**

JSP is a technology for developing Web pages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

**Life cycle of JSP:**

JSP Life Cycle is defined as translation of JSP Page into servlet as a JSP Page needs to be converted into servlet first in order to process the service requests. The Life Cycle starts with the creation of JSP and ends with the disintegration of that.

**Following steps explain the JSP life cycle:**

1. Translation of JSP page
2. Compilation of JSP page (Compilation of JSP page into \_jsp.java)
3. Classloading (\_jsp.java is converted to class file \_jsp.class)
4. Instantiation (Object of generated servlet is created)
5. Initialisation(\_jspinit() method is invoked by container)
6. Request Processing(\_jspservice() method is invoked by the container)
7. Destroy (\_jspDestroy() method invoked by the container)

**1.Translation of JSP page:**

A[Java](https://www.guru99.com/java-tutorial.html)servlet file is generated from a JSP source file. This is the first step of JSP life cycle. In translation phase, container validates the syntactic correctness of JSP page and tag files.

**2.Compilation of the JSP Page**:

The generated java servlet file is compiled into java servlet class. The translation of java source page to its implementation class can happen at any time between the deployment of JSP page into the container and processing of the JSP page. jsp.java is compiled to a class jsp.class.

**3.Classloading**

Servlet class that has been loaded from JSP source is now loaded into the container

**4.Instantiation**

* In this step the object i.e. the instance of the class is generated.
* The container manages one or more instances of this class in the response to requests and other events. Typically, a JSP container is built using a servlet container. A JSP container is an extension of servlet container as both the container support JSP and servlet.
* A JSPPage interface which is provided by container provides init() and destroy () methods.

There is an interface HttpJSPPage which serves HTTP requests, and it also contains the service method.

**5.Initialization**

* jspinit() method will initiate the servlet instance which was generated from JSP and will be invoked by the container in this phase.
* Once the instance gets created, init method will be invoked immediately.
* It is only called once during a JSP life cycle.

**6.Request processing**

* \_jspservice() method is invoked by the container for all the requests raised by the JSP page during its life cycle
* For this phase, it has to go through all the above phases and then only service method can be invoked.
* It passes request and response objects
* This method cannot be overridden
* It is responsible for generating of all HTTP methods i.eGET, POST, etc.

**7.Destroy**

* \_jspdestroy() method is also invoked by the container
* This method is called when container decides it no longer needs the servlet instance to service requests.
* When the call to destroy method is made then, the servlet is ready for a garbage collection
* This is the end of the life cycle.
* We can override jspdestroy() method when we perform any cleanup such as releasing database connections or closing open files.

**JSP Scripting elements**

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

* scriptlet tag
* expression tag
* declaration tag

**Scriplets: <% %>**

* Scriplets will be used to write any valid java statements like local variables or any other java code.
* All statements inside the scriplet must be terminated with semicolon.
* All statements inside the scriplet will be placed inside the jspService().
* Method declaration are not allowed inside a scriplet.

**Syntax:** <%  java source code %>

Example:

**<html>**

**<body>**

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

**</body>**

**</html>**

**Expressions: <%= %>**

* Expression is the short cut form of out.println.
* All the expressions inside the jsp will be placed inside the jspService() method.
* We should not use semi colon at the end of expression.
* Expressions cannot be used inside a scriplets.

**Syntax:** **<**%=  statement %**>**

Example:

**<html>**

**<body>**

**<**%= "welcome to jsp" %**>**

**</body>**

**</html>**

**Declarations: <%! %>**

* Declarations are used to write class level members i.e instance variables and methods.
* Java statements are not allowed inside declarations.
* All the declarations inside the JSP will be placed directly inside the servlet class and outside the jspservice() method.

**Syntax: <**%!  field or method declaration %**>**

**Example:**

**<html>**

**<body>**

**<**%! int data=50; %**>**

**<**%= "Value of the variable is:"+data %**>**

**</body>**

**</html>**

**JSP Directives**

The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.

There are three types of directives:

* page directive
* include directive
* taglib directive

**1.Page Directive:**

The **page** directive is used to provide instructions to the container. These instructions 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 basic syntax of the page directive −

<%@ page attribute = "value" %>

### Attributes

Following table lists out the attributes associated with the page directive −

|  |  |
| --- | --- |
| **S.No.** | **Attribute & Purpose** |
| 1 | **buffer**  Specifies a buffering model for the output stream. |
| 2 | **autoFlush**  Controls the behavior of the servlet output buffer. |
| 3 | **contentType**  Defines the character encoding scheme. |
| 4 | **errorPage**  Defines the URL of another JSP that reports on Java unchecked runtime exceptions. |
| 5 | **isErrorPage**  Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute. |
| 6 | **extends**  Specifies a superclass that the generated servlet must extend. |
| 7 | **import**  Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes. |
| 8 | **info**  Defines a string that can be accessed with the servlet's **getServletInfo()** method. |
| 9 | **isThreadSafe**  Defines the threading model for the generated servlet. |
| 10 | **language**  Defines the programming language used in the JSP page. |
| 11 | **session**  Specifies whether or not the JSP page participates in HTTP sessions |
| 12 | **isELIgnored**  Specifies whether or not the EL expression within the JSP page will be ignored. |
| 13 | **isScriptingEnabled**  Determines if the scripting elements are allowed for use. |

## 2.The include Directive:

The **include** directive is used to include 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 the ***include***directives anywhere in your JSP page.

The general usage form of this directive is as follows −

<%@ include file = "relative url" >

The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.

## 3.The taglib Directive:

The JavaServer Pages API allow 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 means for identifying the custom tags in your JSP page.

The taglib directive follows the syntax given below −

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

Here, the **uri** attribute value resolves to a location the container understands and the **prefix** attribute informs a container what bits of markup are custom actions.

**JSP Standard 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. |

## 1.The <jsp:include> Action:

This action lets you insert files into the page being generated. The syntax looks like this

<jsp:include page = "relative URL" flush = "true" />

Unlike the **include** directive, which inserts the file at the time the JSP page is translated into a servlet, this action inserts the file at the time the page is requested.

Following table lists out the attributes associated with the include action −

|  |  |
| --- | --- |
| **S.No.** | **Attribute & Description** |
| 1 | **page**  The relative URL of the page to be included. |
| 2 | **flush**  The boolean attribute determines whether the included resource has its buffer flushed before it is included. |

### Example

Let us define the following two files **(a)date.jsp** and **(b) main.jsp** as follows −

Following is the content of the **date.jsp** file −

<p>Today's date: <%= (new java.util.Date()).toLocaleString()%></p>

Following is the content of the **main.jsp** file −

<html>

<head>

<title>The include Action Example</title>

</head>

<body>

<center>

<h2>The include action Example</h2>

<jsp:include page = "date.jsp" flush = "true" />

</center>

</body>

</html>

Let us now keep all these files in the root directory and try to access **main.jsp**. You will receive the following output − **Today's date: 12-Sep-2010 14:54:22**

## 2.The <jsp:forward> Action:

The **forward** action terminates the action of the current page and forwards the request to another resource such as a static page, another JSP page, or a Java Servlet.

Following is the syntax of the **forward** action −

<jsp:forward page = "Relative URL" />

Following table lists out the required attributes associated with the forward action −

|  |  |
| --- | --- |
| **S.No.** | **Attribute & Description** |
| 1 | **page**  Should consist of a relative URL of another resource such as a static page, another JSP page, or a Java Servlet. |

### Example

Let us reuse the following two files **(a) date.jsp** and **(b) main.jsp** as follows −

Following is the content of the **date.jsp** file −

<p>Today's date: <%= (new java.util.Date()).toLocaleString()%></p>

Following is the content of the **main.jsp** file −

<html>

<head>

<title>The include Action Example</title>

</head>

<body>

<center>

<h2>The include action Example</h2>

<jsp:forward page = "date.jsp" />

</center>

</body>

</html>

Let us now keep all these files in the root directory and try to access **main.jsp**. This would display result something like as below.

Here it discarded the content from the main page and displayed the content from forwarded page only.

Today's date: 12-Sep-2010 14:54:22

## 3.The <jsp:useBean> Action

The **useBean** action is quite versatile. It first searches for an existing object utilizing the id and scope variables. If an object is not found, it then tries to create the specified object.

The simplest way to load a bean is as follows −

<jsp:useBean id = "name" class = "package.class" />

Once a bean class is loaded, you can use **jsp:setProperty** and **jsp:getProperty** actions to modify and retrieve the bean properties.

Following table lists out the attributes associated with the useBean action −

|  |  |
| --- | --- |
| **S.No.** | **Attribute & Description** |
| 1 | **class**  Designates the full package name of the bean. |
| 2 | **type**  Specifies the type of the variable that will refer to the object. |
| 3 | **beanName**  Gives the name of the bean as specified by the instantiate () method of the java.beans.Beans class. |

**4. The <jsp:setProperty> Action:**

The **setProperty** action sets the properties of a Bean. The Bean must have been previously defined before this action. There are two basic ways to use the setProperty action −

You can use **jsp:setProperty** after, but outside of a **jsp:useBean** element, as given below −

<jsp:useBean id = "myName" ... />

...

<jsp:setProperty name = "myName" property = "someProperty" .../>

In this case, the **jsp:setProperty** is executed regardless of whether a new bean was instantiated or an existing bean was found.

A second context in which jsp:setProperty can appear is inside the body of a **jsp:useBean** element, as given below −

<jsp:useBean id = "myName" ... >

...

<jsp:setProperty name = "myName" property = "someProperty" .../>

</jsp:useBean>

Here, the jsp:setProperty is executed only if a new object was instantiated, not if an existing one was found.

Following table lists out the attributes associated with the **setProperty** action −

|  |  |
| --- | --- |
| **S.No.** | **Attribute & Description** |
| 1 | **name**  Designates the bean the property of which will be set. The Bean must have been previously defined. |
| 2 | **property**  Indicates the property you want to set. A value of "\*" means that all request parameters whose names match bean property names will be passed to the appropriate setter methods. |
| 3 | **value**  The value that is to be assigned to the given property. The the parameter's value is null, or the parameter does not exist, the setProperty action is ignored. |
| 4 | **param**  The param attribute is the name of the request parameter whose value the property is to receive. You can't use both value and param, but it is permissible to use neither. |

**5.The <jsp:getProperty> Action:**

The **getProperty** action is used to retrieve the value of a given property and converts it to a string, and finally inserts it into the output.

<jsp:useBean id = "myName" ... />

...

<jsp:getProperty name = "myName" property = "someProperty" .../>

Following table lists out the required attributes associated with the **getProperty** action −

|  |  |
| --- | --- |
| **S.No.** | **Attribute & Description** |
| 1 | **name**  The name of the Bean that has a property to be retrieved. The Bean must have been previously defined. |
| 2 | **property**  The property attribute is the name of the Bean property to be retrieved. |