JSP

(Java Server Pages)

**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.

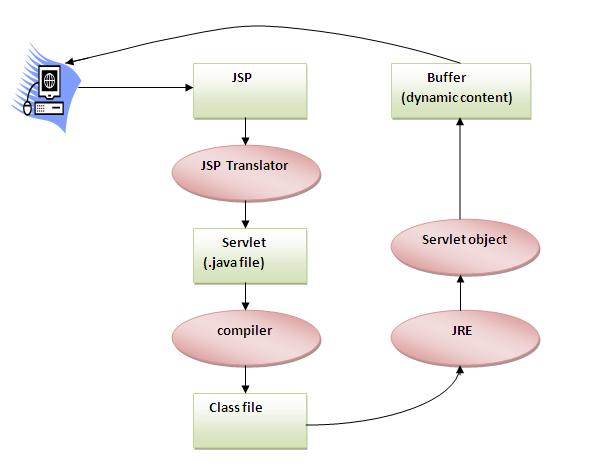
|  |  |
| --- | --- |
| JSP | Servlets |
| JSP is a webpage scripting language that can generate dynamic content. | Servlets are Java programs that are already compiled which also creates dynamic web content. |
| JSP run slower compared to Servlet as it takes compilation time to convert into Java Servlets. | Servlets run faster compared to JSP. |
| It’s easier to code in JSP than in Java Servlets. | Its little much code to write here. |
| In MVC, jsp act as a view. | In MVC, servlet act as a controller. |
| JSP are generally preferred when there is not much processing of data required. | servlets are best for use when there is more processing and manipulation involved. |
| The advantage of JSP programming over servlets is that we can build custom tags which can directly call Java beans. | There is no such custom tag facility in servlets. |

### Life cycle of a JSP Page

The JSP pages follows these phases:

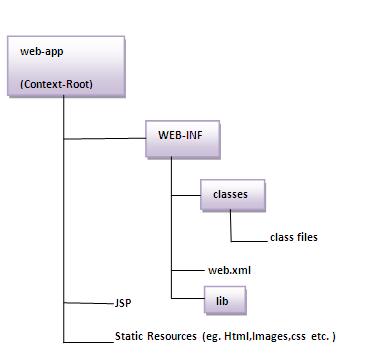
* Translation of JSP Page
* Compilation of JSP Page
* Classloading (.class file is loaded by the classloader)
* Instantiation (Object of the Generated Servlet is created).
* Initialization “**jspInit()”** method is invoked by the container).
* Request processing “ **\_jspservice()”** method is invoked by the container).
* Destroy “**jspDestroy()”** method is invoked by the container).

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



 JSP page is translated into servlet by the help of JSP translator. The JSP translator is a part of webserver that is responsible to translate the JSP page into servlet. After that Servlet page is compiled by the compiler and gets converted into the class file.

### Deployment Directory structure of JSP



# The JSP API

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:

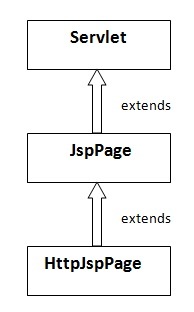
1. JspPage
2. HttpJspPage

The classes are as follows:

* JspWriter
* PageContext
* JspFactory
* JspEngineInfo
* JspException
* 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 cleanup operation.

## The HttpJspPage interface

The HttpJspPage interface provides the one life cycle method of JSP. It extends the JspPage interface.

### Method of HttpJspPage interface:

1. **public void \_jspService():** It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore \_ signifies that you cannot override this method.

# JSP Scripting elements:

### Scripting elements

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

* scriptlet tag (<% java code %>)
* expression tag (<%= expr %>)
* declaration tag (<%! Variable/methods %>)
* directive tag (<%@ %>)

### JSP scriptlet tag

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

1. <%  java source code %>

### Simple Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

1. <html>
2. <body>
3. <% out.print("welcome to jsp"); %>
4. </body>
5. </html>

### Example of JSP scriptlet tag that prints the user name

### index.html

<html>

<body>

<form action="welcome.jsp">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>

</body>

</html>

### welcome.jsp

1. <html>
2. <body>
3. <%
4. String name=request.getParameter("uname");
5. out.print("welcome "+name);
6. %>
7. </form>
8. </body>
9. </html>

# JSP expression tag

The code placed within 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

1. <%=  statement %>

### Example of JSP expression tag

1. <html>
2. <body>
3. <%= "welcome to jsp" %>
4. </body>
6. </html>

### Example of JSP expression tag that prints current time

### index.jsp

1. <html>
2. <body>
3. Current Time: <%= java.util.Calendar.getInstance().getTime() %>
4. </body>
5. </html>

### Example of JSP expression tag that prints the user name

### index.html

1. <html>
2. <body>
4. <form action="welcome.jsp">
5. <input type="text" name="uname"><br/>
6. <input type="submit" value="go">
7. </form>
8. </body>
9. </html>

### welcome.jsp

1. <html>
2. <body>
3. <%= "Welcome "+request.getParameter("uname") %>
4. </form>
5. </body>
6. </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

1. <%!  field or method declaration %>

### Difference between the jsp scriptlet tag and jsp 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. |

### Example of JSP declaration tag that declares field

### index.jsp

1. **<html>**
2. **<body>**
4. **<**%! int data=50; %**>**
5. **<**%= "Value of the variable is:"+data %**>**
7. **</body>**
8. **</html>**

### Example of JSP declaration tag that declares method

### index.jsp

1. **<html>**
2. **<body>**
4. **<**%!
5. int cube(int n){
6. return n\*n\*n\*;
7. }
8. %**>**
10. **<**%= "Cube of 3 is:"+cube(3) %**>**
12. **</body>**
13. **</html>**

# JSP Implicit Objects

There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.

|  |  |
| --- | --- |
| Object | Type |
| out | JspWriter |
| request | HttpServletRequest |
| response | HttpServletResponse |
| config | ServletConfig |
| application | ServletContext |
| session | HttpSession |
| pageContext | PageContext |
| page | Object |
| exception | Throwable |

### 1) JSP out implicit object

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter.

### Example of out implicit object

In this example we are simply displaying date and time.

### index.jsp

1. <html>
2. <body>
3. <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
4. </body>
5. </html>

JSP request implicit object

The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

It can also be used to set, get and remove attributes from the jsp request scope.

### Example of JSP request implicit object

### index.html

1. **<form** action="welcome.jsp"**>**
2. **<input** type="text" name="uname"**>**
3. **<input** type="submit" value="go"**><br/>**
4. **</form>**

### welcome.jsp

1. <%
2. String name=request.getParameter("uname");
3. out.print("welcome "+name);
4. %>

# JSP response implicit object

In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.

It can be used to add or manipulate response such as redirect response to another resource, send error etc.

Let's see the example of response implicit object where we are redirecting the response to the Google.

### Example of response implicit object

**index.html**

1. **<form** action="welcome.jsp"**>**
2. **<input** type="text" name="uname"**>**
3. **<input** type="submit" value="go"**><br/>**
4. **</form>**

**welcome.jsp**

**<**%

response.sendRedirect("http://www.google.com");

%**>**

JSP config implicit object

In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page.

Generally, it is used to get initialization parameter from the web.xml file.

**index.html**

1. **<form** action="welcome"**>**
2. **<input** type="text" name="uname"**>**
3. **<input** type="submit" value="go"**><br/>**
4. **</form>**

**web.xml file**

**<web-app>**

<servlet>

<servlet-name>configservlet</servlet-name>

<jsp-file>/welcome.jsp</jsp-file>

<init-param>

<param-name>driver</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>configservlet</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>

**</web-app>**

**welcome.jsp**

1. **<**%
2. out.print("Welcome "+request.getParameter("uname"));
4. String driver=config.getInitParameter("driver");
5. out.print("driver name is="+driver);
6. %**>**

# JSP application implicit object

In JSP, application is an implicit object of type *ServletContext*.

The instance of ServletContext is created only once by the web container when application or project is deployed on the server.

This object can be used to get initialization parameter from configuaration file (web.xml). It can also be used to get, set or remove attribute from the application scope.

This initialization parameter can be used by all jsp pages.

### Example of application implicit object:

**index.html**

1. **<form** action="welcome"**>**
2. **<input** type="text" name="uname"**>**
3. **<input** type="submit" value="go"**><br/>**
4. **</form>**

**web.xml file**

**<web-app>**

<context-param>

<param-name>driver</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</context-param>

<servlet>

<servlet-name>configservlet</servlet-name>

<jsp-file>/welcome.jsp</jsp-file>

</servlet>

<servlet-mapping>

<servlet-name>configservlet</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>

**</web-app>**

**welcome.jsp**

**<**%

out.print("Welcome "+request.getParameter("uname"));

String driver=application.getInitParameter("driver");

out.print("driver name is="+driver);

%**>**

# session implicit object

|  |
| --- |
| In JSP, session is an implicit object of type HttpSession.The Java developer can use this object to set,get or remove attribute or to get session information. |

### Example of session implicit object

### index.html

<html>

<body>

<form action="welcome.jsp">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>

</body>

</html>

### welcome.jsp

<html>

<body>

<%

String name=request.getParameter("uname");

out.print("Welcome "+name);

session.setAttribute("user",name);

%>

<a href="second.jsp">second jsp page</a>

</body>

</html>

### second.jsp

<html>

<body>

<%

String name=(String)session.getAttribute("user");

out.print("Hello "+name);

%>

</body>

</html>

pageContext implicit object

|  |
| --- |
| In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes:   * page * request * session * application |
| In JSP, page scope is the default scope. |

### Example of pageContext implicit object

### index.html

1. <html>
2. <body>
3. <form action="welcome.jsp">
4. <input type="text" name="uname">
5. <input type="submit" value="go"><br/>
6. </form>
7. </body>
8. </html>

### welcome.jsp

1. <html>
2. <body>
3. <%
5. String name=request.getParameter("uname");
6. out.print("Welcome "+name);
8. pageContext.setAttribute("user",name,PageContext.SESSION\_SCOPE);
10. <a href="second.jsp">second jsp page</a>
12. %>
13. </body>
14. </html>

### second.jsp

1. <html>
2. <body>
3. <%
5. String name=(String)pageContext.getAttribute("user",PageContext.SESSION\_SCOPE);
6. out.print("Hello "+name);
8. %>
9. </body>
10. </html>

exception implicit object

In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.

### Example of exception implicit object:

### error.jsp

1. <%@ page isErrorPage="true" %>
2. <html>
3. <body bgcolor=”red”>
5. Sorry following exception occured:<%= exception %>
7. </body>
8. </html>

#### process.jsp

1. <%@ page errorPage="error.jsp" %>
2. <%
4. String num1=request.getParameter("n1");
5. String num2=request.getParameter("n2");
7. **int** a=Integer.parseInt(num1);
8. **int** b=Integer.parseInt(num2);
9. **int** c=a/b;
10. out.print("division of numbers is: "+c);
12. %>

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

### JSP page directive

The page directive defines attributes that apply to an entire JSP page.

### Syntax of JSP page directive

<%@ page attribute="value" %>

### Attributes of JSP page directive

* import
* contentType
* extends
* info
* buffer
* language
* isELIgnored
* isThreadSafe
* autoFlush
* session
* pageEncoding
* errorPage
* isErrorPage

Jsp Include Directive

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time

### Advantage of Include directive

Code Reusability

### Syntax of include directive

<%@ include file="resourceName" %>

### Example of include directive

In this example, we are including the content of the header.html file

**Header.jsp**

<html>

<body>

<%@ include file="header.html" %>

Today is: <%= java.util.Calendar.getInstance().getTime() %>

</body>

</html>

#### Note: The include directive includes the original content, so the actual page size grows at runtime.

JSP Action Tags (Standard Actions)

There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.

The action tags are used to control the flow between pages and to use Java Bean. The Jsp action tags are given below.

|  |  |
| --- | --- |
| JSP Action Tags | Description |
| 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.

### Example of jsp:forward action tag without parameter

### forward.jsp

<html>

<body>

<h2>this is forward demo</h2>

<jsp:forward page="printdate.jsp" />

</body>

</html>

### printdate.jsp

<html>

<body>

<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>

</body>

</html>

### Example of jsp:forward action tag with parameter

<html>

<body>

<h2>this is forward demo</h2>

<jsp:forward page="printdate.jsp">

<jsp:param name="name" value="impulsesoftware.com" />

</jsp:forward>

</body>

</html>

### printdate.jsp

<html>

<body>

<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>

<%= request.getParameter("name") %>

</body>

</html>

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.

### Difference between jsp include directive and include action

|  |  |
| --- | --- |
| JSP include directive (<%@) | JSP include action <jsp:include> |
| 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. |

### Example of jsp:include action tag without parameter

**include.jsp**

<h2>this is include page</h2>

<jsp:include page="printdate.jsp" />

<h2>end section of includepage</h2>

Java Bean

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

* According to Java white paper, it is a reusable software component.
* It should have a default constructor.
* It should be Serializable. (Optional)
* It should provide methods to set and get the values of the properties, known as getter and setter methods.
* It must be placed in a package.
* It is generally known as **POJO** (Plain Old Java Object).

### Example of java bean class

**Employee.java**

package beans;

public class Employee implements Serializable {

private int id; **// property**

private String name; **// property**

public Employee() {

}

public Employee(int id, String name) {

this.id = id;

this.name = name;

}

public int getId() { // getter method

return id;

}

public void setId(int id) { // setter method

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

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.

### 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.
3. **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.
4. **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.
5. **beanName:**instantiates the bean using the java.beans.Beans.instantiate() method.

jsp:setProperty and jsp:getProperty action tags

The setProperty and getProperty action tags are used for developing web application with Java Bean. In web devlopment, bean class is mostly used because it is a reusable software component that represents data.

The jsp:setProperty action tag sets a property value or values in a bean using the setter method.

### Using java bean in jsp

<body>

<jsp:useBean id="**emp**" class="beans.Employee"></jsp:useBean>

<jsp:setProperty property="id" value="1001" name="**emp**"/>

<jsp:setProperty property="name" value="srinivas" name="**emp**"/>

Data : <br />

<jsp:getProperty property="id" name="**emp**"/><br>

<jsp:getProperty property="name" name="**emp**"/><br>

</body>

Expression Language (EL) in JSP

The **Expression Language** (EL) simplifies the accessibility of data stored in the Java Bean component, and other objects like request, session, application etc.

There are many implicit objects, operators and reserve words in EL.

### Syntax for Expression Language (EL)

${ expression }

### Implicit Objects in Expression Language (EL)

|  |  |
| --- | --- |
| Implicit Objects | Usage |
| pageScope | it maps the given attribute name with the value set in the page scope |
| requestScope | it maps the given attribute name with the value set in the request scope |
| sessionScope | it maps the given attribute name with the value set in the session scope |
| applicationScope | it maps the given attribute name with the value set in the application scope |
| param | it maps the request parameter to the single value |
| paramValues | it maps the request parameter to an array of values |
| header | it maps the request header name to the single value |
| headerValues | it maps the request header name to an array of values |
| cookie | it maps the given cookie name to the cookie value |
| initParam | it maps the initialization parameter |
| pageContext | it provides access to many objects request, session etc. |

### Example of Expression Language that prints the name of the user

#### index.jsp

<form action="process.jsp">

Enter Name:<input type="text" name="name" /><br/><br/>

<input type="submit" value="go"/>

</form>

#### process.jsp

Welcome, ${ param.name }

### Example of Expression Language that prints the value set in the session scope

#### index.jsp

<h3>welcome to index page</h3>

<%

session.setAttribute("user","sonoo");

%>

<a href="process.jsp">visit</a>

#### process.jsp

Value is ${ sessionScope.user }

### Reserve words in EL

|  |  |  |  |
| --- | --- | --- | --- |
| lt | le | gt | ge |
| eq | ne | true | false |
| and | or | not | instanceof |
| div | mod | empty | null |

Custom Tags in JSP

**Custom tags** are user-defined tags.

### Advantages of Custom Tags

The key advantages of Custom tags are as follows:

1. **Eliminates the need of srciptlet tag** The custom tag eliminates the need of scriptlet tag which is considered bad programming approach in JSP.
2. **Separation of business logic from JSP**The custom tags separate the business logic from the JSP page so that it may be easy to maintain.
3. **Reusability** The custom tags makes the possibility to reuse the same business logic again and again.

### Syntax to use custom tag

There are two ways to use the custom tag. They are given below:

<prefix:tagname attr1=value1....attrn=valuen />

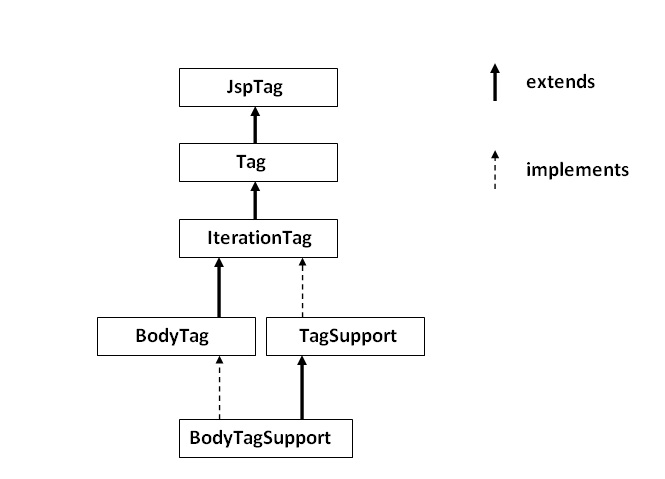
<prefix:tagname attr1=value1....attrn=valuen >

body code

</prefix:tagname>

### JSP Custom Tag API

The **javax.servlet.jsp.tagext** package contains classes and interfaces for JSP custom tag API. The **JspTag** is the root interface in the Custom Tag hierarchy.



**SimpleTagSupport (doTag() method)**

### JspTag interface

The JspTag is the root interface for all the interfaces and classes used in custom tag. It is a marker interface.

### Tag interface

The Tag interface is the sub interface of JspTag interface. It provides methods to perform action at the start and end of the tag.

### Fields of Tag interface

There are four fields defined in the Tag interface. They are:

|  |  |
| --- | --- |
| Field Name | Description |
| public static int EVAL\_BODY\_INCLUDE | it evaluates the body content. |
| public static int EVAL\_PAGE | it evaluates the JSP page content after the custom tag. |
| public static int SKIP\_BODY | it skips the body content of the tag. |
| public static int SKIP\_PAGE | it skips the JSP page content after the custom tag. |

### Methods of Tag interface

The methods of the Tag interface are as follows:

|  |  |
| --- | --- |
| Method Name | Description |
| public void setPageContext(PageContext pc) | it sets the given PageContext object. |
| public void setParent(Tag t) | it sets the parent of the tag handler. |
| public Tag getParent() | it returns the parent tag handler object. |
| public int doStartTag()throws JspException | it is invoked by the JSP page implementation object. The JSP programmer should override this method and define the business logic to be performed at the start of the tag. |
| public int doEndTag()throws JspException | it is invoked by the JSP page implementation object. The JSP programmer should override this method and define the business logic to be performed at the end of the tag. |
| public void release() | it is invoked by the JSP page implementation object to release the state. |

### IterationTag interface

The IterationTag interface is the sub interface of the Tag interface. It provides an additional method to reevaluate the body.

### Field of IterationTag interface

There is only one field defined in the IterationTag interface.

* **public static int EVAL\_BODY\_AGAIN**it reevaluates the body content.

### Method of Tag interface

There is only one method defined in the IterationTag interface.

* **public int doAfterBody()throws JspException**it is invoked by the JSP page implementation object after the evaluation of the body. If this method returns EVAL\_BODY\_INCLUDE, body content will be reevaluated, if it returns SKIP\_BODY, no more body cotent will be evaluated.

### TagSupport class

The TagSupport class implements the IterationTag interface. It acts as the base class for new Tag Handlers. It provides some additional methods also.

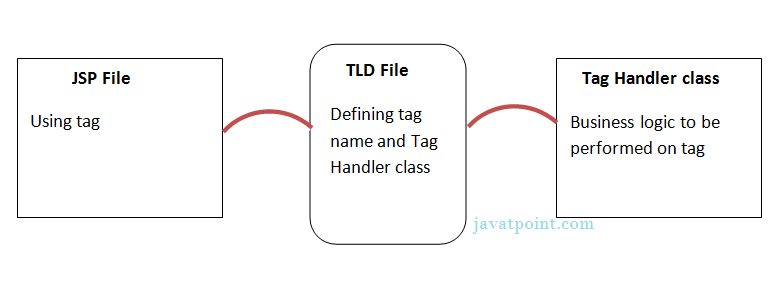
Example of JSP Custom Tag

we are going to create a **custom tag that prints the current date and time**. We are performing action at the start of tag.

For creating any custom tag, we need to follow following steps:

1. **Create the Tag handler class** and perform action at the start or at the end of the tag.
2. **Create the Tag Library Descriptor (TLD) file** and define tags
3. **Create the JSP file that uses the Custom tag defined in the TLD file**

### Understanding flow of custom tag in jsp



### 1) Create the Tag handler class

To create the Tag Handler, we are inheriting the **TagSupport class** and overriding its method **doStartTag()**.To write data for the jsp, we need to use the **JspWriter class**.

The **PageContext** class provides **getOut()** method that returns the instance of JspWriter class. TagSupport class provides instance of pageContext bydefault.

### 2) Create the TLD file

**Tag Library Descriptor** (TLD) file contains information of tag and Tag Handler classes. It must be contained inside the **WEB-INF** directory.

**It is an XML File.**

<?xml version="1.0" encoding="UTF-8"?>

<taglib>

<tlib-version>1.0</tlib-version>

<jsp-version>1.2</jsp-version>

<short-name>simple</short-name>

**<uri>http://www.impulsesoftware.com</uri>**

<tag>

<name>**today**</name>

<tag-class>tags.TodayHandler</tag-class>

</tag>

</taglib>

### 3) Create the JSP file

Let's use the tag in our jsp file. Here, we are specifying the path of tld file directly. But it is recommended to use the uri name instead of full path of tld file.

It uses **taglib** directive to use the tags defined in the tld file.

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

pageEncoding="ISO-8859-1"%>

**<%@ taglib uri="http://www.impulsesoftware.com" prefix="m" %>**

<!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>JSP Custom Tags</title>

</head>

<body>

Current Date and Time is: <**m:today**/>

</body>

</html>

Attributes in JSP Custom Tag

There can be defined too many attributes for any custom tag. To define the attribute, you need to perform two tasks:

* Define the property in the TagHandler class with the attribute name and define the setter method
* define the attribute element inside the tag element in the TLD file

### Simple example of attribute in JSP Custom Tag

**CubeHandler.java**

package tags;

import javax.servlet.jsp.JspException;

import javax.servlet.jsp.JspWriter;

import javax.servlet.jsp.tagext.TagSupport;

public class CubeHandler extends TagSupport{

int number;

public void setNumber(int number) {

this.number = number;

}

@Override

public int doStartTag() throws JspException {

JspWriter out=pageContext.getOut();

try{

out.print(number\*number\*number);

}catch(Exception e){

e.printStackTrace();

}

return SKIP\_BODY;

}

}

**Index.jsp**

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

pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://www.impusesoftware.com" prefix="is" %>

<!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>JSP Custom Tags</title>

</head>

<body>

Current Date and Time is: <is:today/>

<br />

Cube of 5 = <is:cube number="5" />

</body>

</html>

**Mytags.tld**

<tag>

<name>cube</name>

<tag-class>tags.CubeHandler</tag-class>

<attribute>

<name>number</name>

<required>true</required>

</attribute>

</tag>

## JSP Custom Tag attribute example with database

**JobsHandler.java**

package tags;

import javax.servlet.jsp.JspException;

import javax.servlet.jsp.JspWriter;

import javax.servlet.jsp.tagext.SimpleTagSupport;

import javax.sql.rowset.CachedRowSet;

import oracle.jdbc.rowset.OracleCachedRowSet;

public class JobsHandler extends SimpleTagSupport {

@Override

public void doTag() throws JspException {

JspWriter out = getJspContext().getOut();

try {

CachedRowSet rs=new OracleCachedRowSet();

rs.setUrl("jdbc:oracle:thin:@localhost:1521:xe");

rs.setUsername("hr");

rs.setPassword("hr");

rs.setCommand("select job\_id, job\_title from jobs");

rs.execute();

while (rs.next()) {

out.println(rs.getString(1) + " - " + rs.getString(2) + "<br />");

}

}

catch (Exception ex) {

throw new JspException(ex.getMessage());

}

}

}

**index.jsp**

<body>

Current Date and Time is: <is:today/>

<br />

Cube of 5 = <is:cube number="5" />

<br />

<is:jobs />

</body>

**mytags.tld**

<tag>

<name>jobs</name>

<tag-class>tags.JobsHandler</tag-class>

<body-content>empty</body-content>

</tag>

JSTL (Java Server Pages Standard Tag Library)

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

#### Advantage of JSTL

1. **Fast Development** JSTL provides many tags that simplifies the JSP.
2. **Code Reusability** We can use the JSTL tags in various pages.
3. **No need to use scriptlet tag** It avoids the use of scriptlet tag.

There JSTL mainly provides 5 types of tags:

|  |  |
| --- | --- |
| Tag Name | Description |
| core tags | The JSTL core tag provide variable support, URL management, flow control etc. The url for the core tag is **http://java.sun.com/jsp/jstl/core** . The prefix of core tag is **c**. |
| sql tags | The JSTL sql tags provide SQL support. The url for the sql tags is **http://java.sun.com/jsp/jstl/sql** and prefix is **sql**. |
| xml tags | The xml sql tags provide flow control, transformation etc. The url for the xml tags is **http://java.sun.com/jsp/jstl/xml** and prefix is **x**. |
| internationalization tags | The internationalization tags provide support for message formatting, number and date formatting etc. The url for the internationalization tags is **http://java.sun.com/jsp/jstl/fmt** and prefix is **fmt**. |
| functions tags | The functions tags provide support for string manipulation and string length. The url for the functions tags is **http://java.sun.com/jsp/jstl/functions** and prefix is **fn**. |

#### For creating JSTL application, you need to load jstl-1.2.jar file.

## JSTL Core Tags

The JSTL core tags mainly provides 4 types of tags:

* **Miscellaneous tags**: catch and out.
* **url management tags**: import, redirect and url.
* **variable support tags**: remove and set.
* **flow control tags**: forEach, forTokens, if and choose.

#### Syntax for defining core tags

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

### c:out

It is just like JSP expression tag but it is used for exression. It renders data to the page.

#### Example of c:out

**index.jsp**

<form action="process.jsp" method="post">

FirstName:<input type="text" name="fname"/><br/>

LastName:<input type="text" name="lname"/><br/>

<input type="submit" value="submit"/>

</form>

**process.jsp**

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

First Name:<**c:out** value="${param.fname}"><**/c:out**><br/>

Last Name:<**c:out** value="${param.lname}"><**/c:out**>

### c:import

It is just like jsp include but it can include the content of any resource either within server or outside the server.

### c:forEach

It repeats the nested body content for fixed number of times or over collection.

#### Example of c:forEach

<c:forEach var="number" begin="5" end="10">

<c:out value="${number}"></c:out>

</c:forEach>

### c:if

It tests the condition.

#### Example of c:if

<c:set var="number" value="${param.num}" />

<c:if test="${param.num%2==0}">

<c:out value="number is even"></c:out>

</c:if>

### c:redirect

It redirects the request to the given url.

**<c:redirect** url="http://www.gmail.com"**></c:redirect>**