

## **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**

- 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.

### 4) **Less code than Servlet**

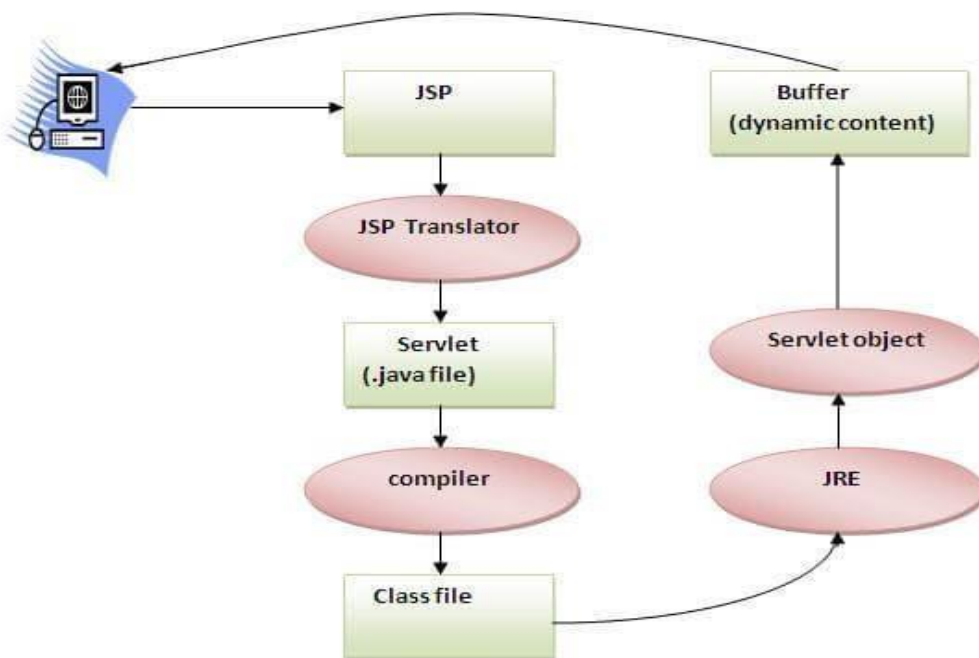
- In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code.
- Moreover, we can use EL, implicit objects, etc.

## The Lifecycle of a JSP Page

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization ( the container invokes jspInit() method).
- Request processing ( the container invokes \_jspService() method).
- Destroy ( the container invokes jspDestroy() method).

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



- As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator.

- The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet.
- After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

## Creating a simple JSP Page

- To create the first JSP page, write some HTML code as given below, and save it by .jsp extension.
- We have saved this file as index.jsp.
- Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

### index.jsp

- Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page.
- We will learn scriptlet tag later.

```
<html>
<body>
<% out.print(2*5); %>
</body>
</html>
```

It will print **10** on the browser.

## How to run a simple JSP Page?

Follow the following steps to execute this JSP page:

- Start the server
- Put the JSP file in a folder and deploy on the server

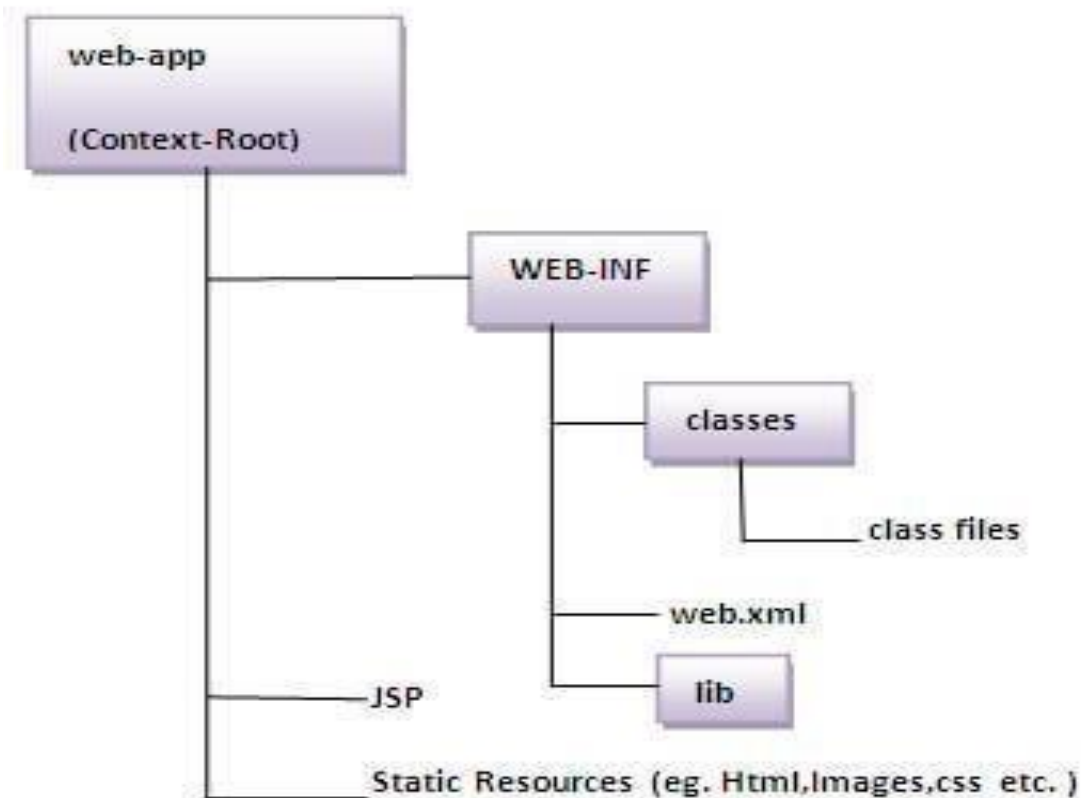
Visit the browser by the URL <http://localhost:portno/contextRoot/jspfile>, for example, <http://localhost:8888/myapplication/index.jsp>

### Do I need to follow the directory structure to run a simple JSP?

- No, there is no need of directory structure if you don't have class files or TLD files.
- For example, put JSP files in a folder directly and deploy that folder. It will be running fine.
- However, if you are using Bean class, Servlet or TLD file, the directory structure is required.

### The Directory structure of JSP

- The directory structure of JSP page is same as Servlet.
- We contain the JSP page outside the WEB-INF folder or in any directory.



## The JSP API

The JSP API consists of two packages:

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

### javax.servlet.jsp package

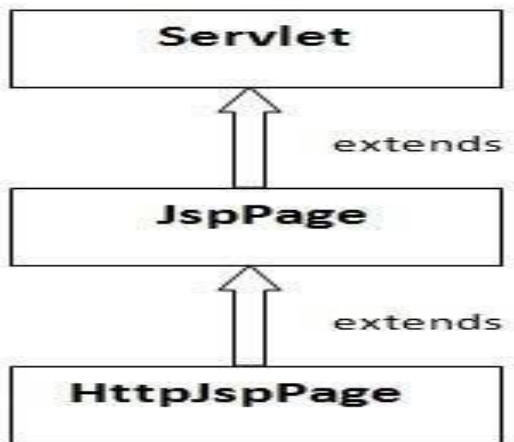
- The javax.servlet.jsp package has two interfaces and classes.
- The two interfaces are as follows:
- JspPage
- HttpJspPage

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

- **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.
- **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.

### The HttpJspPage interface

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

Method of HttpJspPage interface:

- **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 Scriptlet tag (Scripting elements)

- 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: scriptlet tag expression tag 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>
```

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

- In this example, we have created two files index.html and welcome.jsp.
- The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message. *File: index.html*



```
    form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

## 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

```
<html>
<body>
<
<%= statement %>
```

## 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>
```

Note: Do not end your statement with semicolon in case of expression tag.

## Example of JSP expression tag that prints current time

- To display the current time, we have used the getTime() method of Calendar class.
- The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

- 
- 
- *index.jsp*

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

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

- In this example, we are printing the username using the expression tag.
- The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

*File: index.jsp*

```
    form action="welcome.jsp">
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
</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.

```
<html>
<body>
<
```

## Syntax of JSP declaration tag

The syntax of the declaration tag

is as follows:

```
<%! field or method declaration %>
```

### 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 <code>_jspService()</code> method.	The declaration of jsp declaration tag is placed outside the <code>_jspService()</code> method.

### Example of JSP declaration tag that declares field

- In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

### Example of JSP declaration tag that declares method

- In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag.
- But we can also use jsp scriptlet tag to call the declared method.

index.jsp

```
%!  
int cube(int n){  
    return n*n*n;  
}  
%>  
<%= "Cube of 3 is:"+cube(3) %>  
</body>  
</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.

The available implicit objects are out, request, config, session, application etc.

A list of the 9 implicit objects is given below:

Object	Type
out	JspWriter
request	HttpServletRequest

&lt;html&gt;

&lt;body&gt;

&lt;

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. In case of servlet you need to write:

```
PrintWriter out=response.getWriter();
```

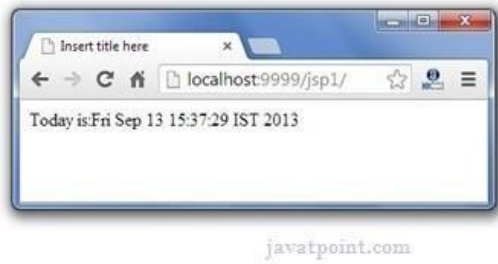
But in JSP, you don't need to write this code.

### Example of out implicit object

- In this example we are simply displaying date and time. index.jsp

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

### Output



## **Upcoming topics in JSP implicit Objects**

**Example of Request implicit Object**

**Example of Response implicit Object**

**Example of config implicit object**

**Example of application implicit object**

**Example of session implicit object**

**Example of pageContext implicit object**

**Example of page implicit object**

**Example of exception implicit object**

## **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.

**Let's see the simple example of request implicit object where we are printing the name of the user with welcome message.**

## **Example of JSP request implicit object index.html**

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

## **welcome.jsp**

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

## **Output**





### 3) 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

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
response.sendRedirect("http://www.google.com");
%>
```

### Output



### 4) 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.

## Example of config implicit object:

index.html

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

web.xml file

```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>

<init-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>

</servlet>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

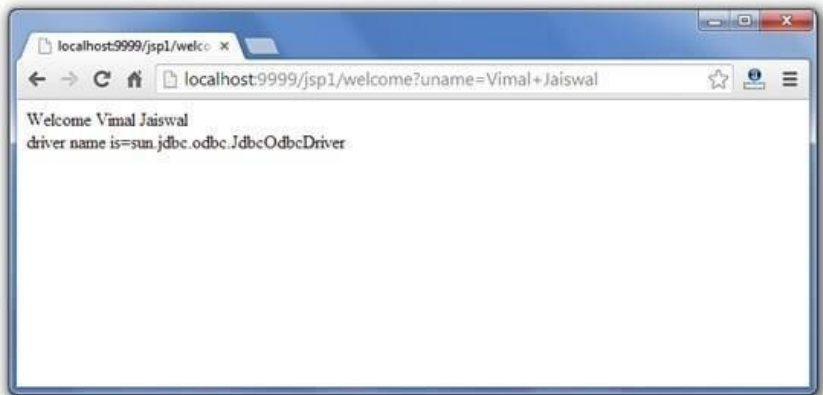
</web-app>
```

welcome.jsp

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

String driver=config.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

## Output



## 5) 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 configuration 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**

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

**web.xml file**

```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>
</servlet>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

<context-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>

</web-app>
```

**welcome.jsp**

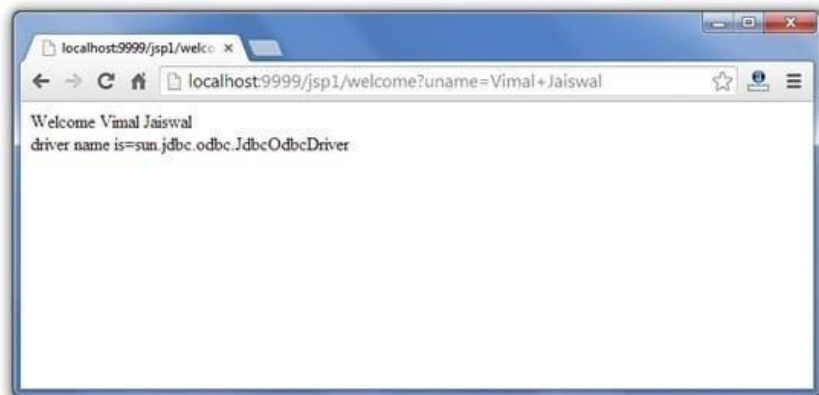
```
<%

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

String driver=application.getInitParameter("dname");
out.print("driver name is="+driver);

%>
```

**Output**



## 6) 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

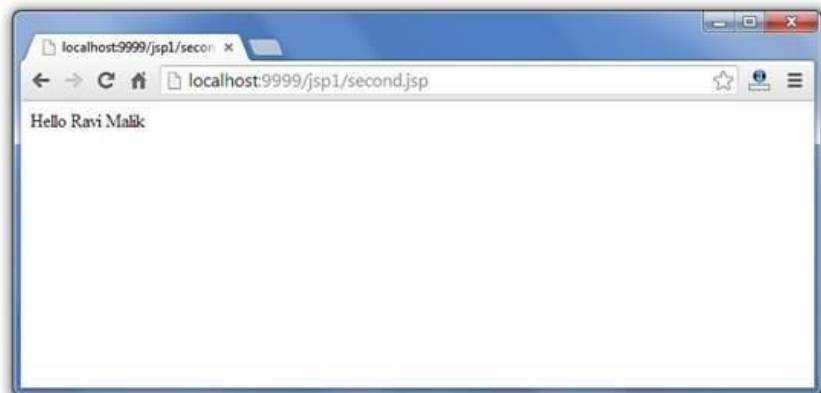
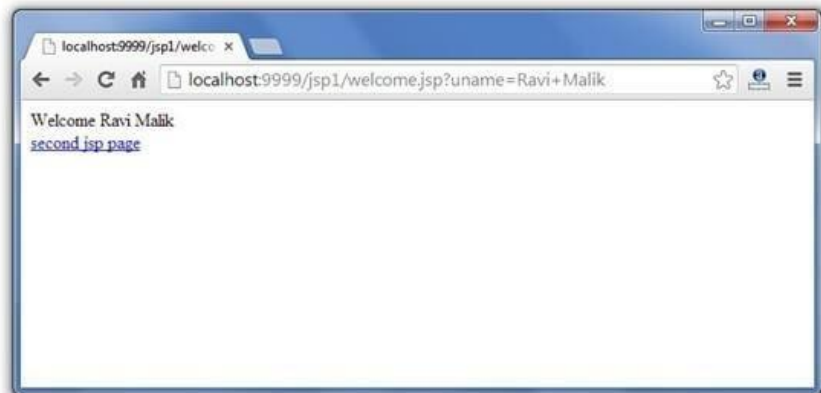
```

<html>
<body>
<%

String name=(String)session.getAttribute("user");
out.print("Hello "+name);

%>
</body>
</html>

```



## 7) **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

```
<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);

pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);

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

%>
</body>
</html>
```

## second.jsp

```
<html>
<body>
<%

String name=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
out.print("Hello "+name);

%>
</body>
</html>
```

## Output



## 8) page implicit object:

- In JSP, page is an implicit object of type Object class.
- This object is assigned to the reference of auto generated servlet class. It is written as:
- Object page=this;
- For using this object it must be cast to Servlet type. For example:

```
<% (HttpServletRequest)page.log("message"); %>
```

Since, it is of type Object it is less used because you can use this object directly in jsp. For example:

```
<% this.log("message"); %>
```

## 9) 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.
- It is better to learn it after page directive. Let's see a simple example:

### Example of exception implicit object: error.jsp

```
<%@ page isErrorPage="true" %>
<html>
<body>

Sorry following exception occurred: <%= exception %>

</body>
</html>
```

To get the full example, [click here](#) full example of exception handling in jsp. But, it will be better to learn it after the JSP Directives.

## 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

### Syntax of JSP Directive

```
<%@ directive attribute="value" %>
```

## JSP page directive

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

### Syntax of JSP page directive

```
<%@ page attribute="value" %>
```

AdChoices

### Attributes of JSP page directive

- import
- contentType
- extends
- info
- buffer
- language
- isELIgnored
- isThreadSafe
- autoFlush

- session
- pageEncoding
- errorPage
- isErrorPage

## 1)import

- The import attribute is used to import class,interface or all the members of a package.
- It is similar to import keyword in java class or interface.

### Example of import attribute

```
<html>
<body>

<%@ page import="java.util.Date" %>
Today is: <%= new Date() %>

</body>
</html>
```

## 2)contentType

- The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.
- The default value is "text/html;charset=IS O-8859-1".

## Example of contentType attribute

```
<html>
<body>

<%@ page contentType=application/msword %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

### 3)extends

- The extends attribute defines the parent class that will be inherited by the generated servlet.
- It is rarely used.

### 4)info

- This attribute simply sets the information of the JSP page which is retrieved later by using `getServletInfo()` method of `Servlet` interface.

## Example of info attribute

```
<html>
<body>

<%@ page info="composed by Sonoo Jaiswal" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

The web container will create a method `getServletInfo()` in the resulting servlet. For example:

```
public String getServletInfo() {
    return "composed by Sonoo Jaiswal";
}
```

### 5)buffer

- The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.

- The default size of the buffer is 8Kb.

### Example of buffer attribute

```
<html>
<body>

<%@ page buffer="16kb" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

## 6)language

- The language attribute specifies the scripting language used in the JSP page. The default value is "java".

## 7)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
```

## 8)isThreadSafe



- Servlet and JSP both are multithreaded. If you want to control this behaviour of JSP page, you can use `isThreadSafe` attribute of page directive.
- The value of `isThreadSafe` value is `true`.
- If you make it `false`, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it. If you make the value of `isThreadSafe` attribute like:

`<%@ page isThreadSafe="false" %>`

The web container in such a case, will generate the servlet as:

```
public class SimplePage_jsp extends HttpJspBase
implements SingleThreadModel{
.....
}
```

## 9)errorPage

- The `errorPage` attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

Example of `errorPage` attribute

```
//index.jsp
<html>
<body>

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

<%= 100/0 %>

</body>
</html>
```

## 10)isErrorPage

- The isErrorPage attribute is used to declare that the current page is the error page.

Note: The exception object can only be used in the error page.

### Example of isErrorPage attribute

```
//myerrorpage.jsp

<html>
<body>

<%@ page isErrorPage="true" %>

  Sorry an exception occurred!<br/>
  The exception is: <%= exception %>

</body>
</html>
```

## 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 (the jsp page is translated only once so it will be better to include static resource).

## 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. To run this example you must create an header.html file.

```
<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 Taglib directive

- The JSP taglib directive is used to define a tag library that defines many tags.
- We use the TLD (Tag Library Descriptor) file to define the tags.

- In the custom tag section we will use this tag so it will be better to learn it in custom tag.

## Syntax JSP Taglib directive

```
<%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>
```

## Example of JSP Taglib directive

- In this example, we are using our tag named currentDate.
- To use this tag we must specify the taglib directive so the container may get information about the tag.

```
<html>
<body>

<%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>

<mytag:currentDate/>

</body>
</html>
```

## JSP Action Tags

- 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.



The jsp:useBean, jsp:setProperty and jsp:getProperty tags are used for bean development. So we will see these tags in bean development.

### 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>
```

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

In this example, we are simply forwarding the request to the `printdate.jsp` file. `index.jsp`

```
<html>  
<body>  
<h2>this is index page</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

- In this example, we are forwarding the request to the `printdate.jsp` file with parameter and `printdate.jsp` file prints the parameter value with date and time.

### `index.jsp`

```
<html>
<body>
<h2>this is index page</h2>

<jsp:forward page="printdate.jsp" >
<jsp:param name="name" value="javatpoint.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.

## 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.

## Difference between jsp include directive and include action

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.

### 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>
```

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

In this example, index.jsp file includes the content of the printdate.jsp file.



File: *index.jsp*

```
<h2>this is index page</h2>

<jsp:include page="printdate.jsp" />

<h2>end section of index page</h2>
```

File: *printdate.jsp*

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

download this exampl e





## 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.

## Simple example of JavaBean class

```
//Employee.java

package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;
public Employee(){ }
public void setId(int id){this.id=id;}
public int getId(){return id;}
public void setName(String name){this.name=name;}
public String getName(){return name;}
}
```

## How to access the JavaBean class?

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

```
package mypack;  
public class Test{  
    public static void main(String args[]){  
        Employee e=new Employee();//object is created  
        e.setName("Arjun");//setting value to the object  
        System.out.println(e.getName());  
    }  
}
```

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.

### 2. **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 of JavaBean

The following are the advantages of JavaBean:/p>

- The JavaBean properties and methods can be exposed to another application.

It provides an easiness to reuse the software components.

### Disadvantages of JavaBean

The following are the disadvantages of JavaBean:

- 
- JavaBeans are mutable. So, it can't take advantages of immutable objects.

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.

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.

### Simple example of jsp:useBean action tag

In this example, we are simply invoking the method of the Bean class.

For the example of setProperty, getProperty and useBean tags, visit next page.

### Calculator.java (a simple Bean class)

```
package com.javatpoint;
public class Calculator{

    public int cube(int n){return n*n*n;}

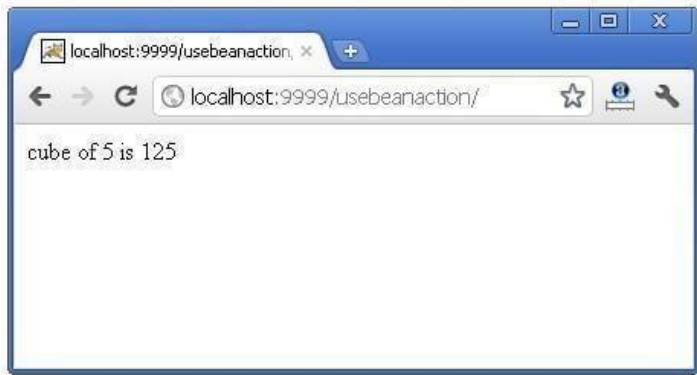
}
```

### index.jsp file

```
<jsp:useBean id="obj" class="com.javatpoint.Calculator"/>

<%
int m=obj.cube(5);
out.print("cube of 5 is "+m);
%>
```

download this example







## **jsp:setProperty and jsp:getProperty action tags**

The setProperty and getProperty action tags are used for developing web application with Java Bean. In web development, 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.

### **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" />
```

## Example of bean development in JSP In

this example there are 3 pages:

- index.html for input of values welocme.jsp file that sets
- the incoming values to the bean object and prints the one value

User.java bean class that have setter and getter methods

### index.html

```
<form action="process.jsp" method="post">  
Name: <input type="text" name="name"> <br>  
Password: <input type="password" name="password"> <br>  
Email: <input type="text" name="email"> <br>  
<input type="submit" value="register">  
</form>
```

### process.jsp

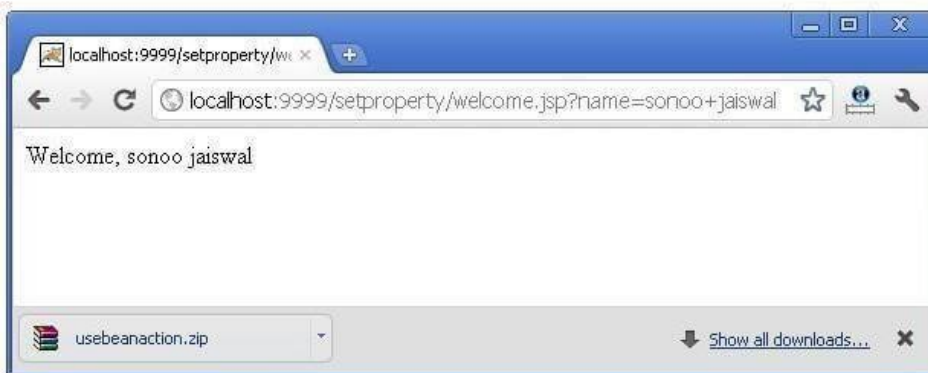
```
<jsp:useBean id="u" class="org.sssit.User"> </jsp:useBean>  
<jsp:setProperty property="*" name="u"/>  
  
Record: <br>  
<jsp:getProperty property="name" name="u"/> <br>  
<jsp:getProperty property="password" name="u"/> <br>  
<jsp:getProperty property="email" name="u" /> <br>
```

### User.java

```
<jsp:useBean id="u" class="org.sssit.User" scope="session"> </jsp:useBean>  
<jsp:setProperty property="*" name="u"/>
```

```
package org.sssit;  
  
public class User {  
    private String name,password,email;  
    //setters and getters  
}
```

download this examnl e



## Reusing Bean in Multiple Jsp Pages

Let's see the simple example, that prints the data of bean

object in two jsp pages. index.jsp Same as above.

User.java Same

as

above.

process.js

p

```
Record:<br>
<jsp:getProperty property="name" name="u"/><br>
<jsp:getProperty property="password" name="u"/><br>
<jsp:getProperty property="email" name="u" /><br>

<a href="second.jsp">Visit Page</a>
```

## second.jsp

```
<jsp:useBean id="u" class="org.sssit.User" scope="session"></jsp:useBean>
Record:<br>
<jsp:getProperty property="name" name="u"/><br>
<jsp:getProperty property="password" name="u"/><br>
<jsp:getProperty property="email" name="u" /><br>
```

## Using variable value in setProperty tag

In some case, you may get some value from the database, that is to be set in the bean object, in such case, you need to use expression tag. For example: process.jsp

```
<jsp:useBean id="u" class="org.sssit.User"></jsp:useBean>
<%
String name="arjun";
%>
<jsp:setProperty property="name" name="u" value="<%=name %>"/>

Record:<br>
<jsp:getProperty property="name" name="u"/><br>
```



## Displaying applet in JSP (jsp:plugin action tag)

The jsp:plugin action tag is used to embed applet in the jsp file. The jsp:plugin action tag downloads plugin at client side to execute an applet or bean.

### Syntax of jsp:plugin action tag

```
<jsp:plugin type= "applet | bean" code= "nameOfClassFile"  
codebase= "directoryNameOfClassFile"  
</jsp:plugin>
```

### Example of displaying applet in JSP

In this example, we are simply displaying applet in jsp using the jsp:plugin tag. You must have MouseDrag.class file (an applet class file) in the current folder where jsp file resides. You may simply download this program that contains index.jsp, MouseDrag.java and MouseDrag.class files to run this application. **index.jsp**

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Mouse Drag</title>
  </head>
  <body bgcolor="khaki">
    <h1>Mouse Drag Example</h1>

    <jsp:plugin align="middle" height="500" width="500"
      type="applet" code="MouseDrag.class" name="clock" codebase="." />

  </body>
</html>
```

<https://www.javatpoint.com/displaying-applet-in-jsp> 1/4

## MVC in JSP

**MVC** stands for Model View and Controller. It is a **design pattern** that separates the business logic, presentation logic and data.

**Controller** acts as an interface between View and Model. Controller intercepts all the incoming requests.

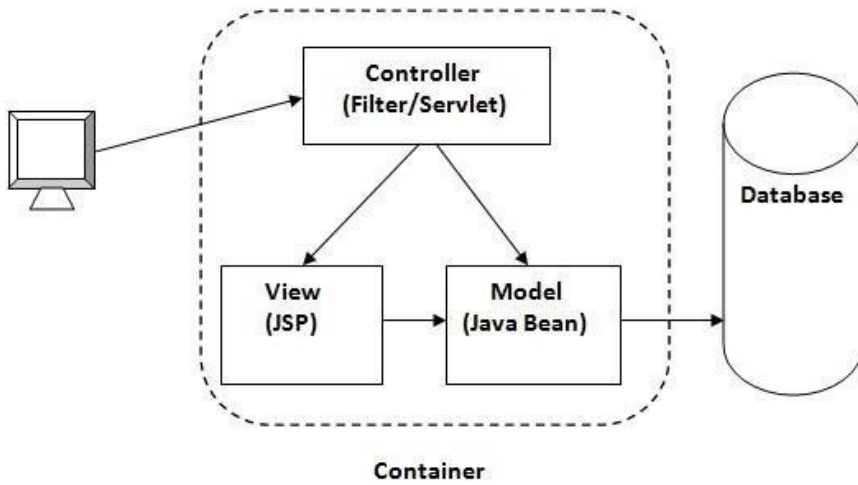
**Model** represents the state of the application i.e. data. It can also have business logic.

**View** represents the presentaion i.e. UI(User Interface).

**Advantage of MVC (Model 2) Architecture**

1. Navigation Control is centralized
2. Easy to maintain the large application





If you new to MVC, please visit [Model1 vs Model 2](#) first.

## MVC Example in JSP

In this example, we are using servlet as a controller, jsp as a view component, Java Bean class as a model.

In this example, we have created 5 pages:

- **index.jsp** a page that gets input from the user.
- **ControllerServlet.java** a servlet that acts as a controller.
- **login-success.jsp** and **login-error.jsp** files acts as view components.
- **web.xml** file for mapping the servlet.

File: *index.jsp*

```
<form action="ControllerServlet" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
<input type="submit" value="login">
</form>
```

File: *ControllerServlet*

```
package com.javatpoint;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ControllerServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

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

```
String password=request.getParameter("password");

LoginBean bean=new LoginBean();
bean.setName(name);
bean.setPassword(password);
request.setAttribute("bean",bean);

boolean status=bean.validate();

if(status){
    RequestDispatcher rd=request.getRequestDispatcher("login-success.jsp");
    rd.forward(request, response);
}
else{
    RequestDispatcher rd=request.getRequestDispatcher("login-error.jsp");
    rd.forward(request, response);
}

}

@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    doPost(req, resp);
}
}
```

File: LoginBean.java

```
package com.javatpoint;

public class LoginBean {
    private String name,password;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public boolean validate(){
        if(password.equals("admin")){
            return true;
        }
    }
}
```

```
}  
else{  
    return false;  
}  
}  
}
```

File: login-success.jsp

```
<%@page import="com.javatpoint.LoginBean"%>  
  
<p>You are successfully logged in!</p>  
<%  
LoginBean bean=(LoginBean)request.getAttribute("bean");  
out.print("Welcome, "+bean.getName());  
%>
```

File: login-error.jsp

```
<p>Sorry! username or password error</p>  
<%@ include file="index.jsp" %>
```

File: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-  
app_2_5.xsd"  
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-  
app_3_0.xsd"  
id="WebApp_ID" version="3.0">  
  
    <servlet>  
        <servlet-name>s1</servlet-name>  
        <servlet-class>com.javatpoint.ControllerServlet</servlet-class>  
    </servlet>  
    <servlet-mapping>  
        <servlet-name>s1</servlet-name>  
        <url-pattern>/ControllerServlet</url-pattern>  
    </servlet-mapping>  
</web-app>
```

download this example (developed using eclipse IDE)

Output

