UNIT IV: Including files and applets in JSP pages and Using Java Beans components in JSP documents

Including pages at request time: the *jsp:include*action, Including pages at page translation time: the *include* directive, Forwarding request with *jsp:Forward,* Including applets for java plug-in, Why use Beans?, What are Beans?, Using Beans: basic task, Example: *StrignBean*

Including pages at request time: the *jsp:include*action

The jsp:include action lets you include the output of a page at request time. Its main advantage is that it saves you from changing the main page when the included pages change. Its main disadvantage is that since it includes the output of the secondary page, not the secondary page's actual code as with the include directive, the included pages cannot use any JSP constructs that affect the main page as a whole. The advantages generally far outweigh the disadvantages, and you will almost certainly use it much more than the other inclusion mechanisms.

Suppose you have a series of pages, all of which have the same navigation bar, contact information, or footer. What can you do? Well, one common "solution" is to cut and paste the same HTML snippets into all the pages. This is a bad idea because when you change the common piece, you have to change every page that uses it. Another common solution is to use some sort of server-side include mechanism whereby the common piece gets inserted as the page is requested. This general approach is a good one, but the typical mechanisms are server specific. Enter jsp:include, a portable mechanism that lets you insert any of the following into the JSP output:

• The content of an HTML page.
• The content of a plain text document.
• The output of JSP page.
• The output of a servlet.

The jsp:include action includes the output of a secondary page at the time the main page is requested. Although the output of the included pages cannot contain JSP, the pages can be the result of resources that use servlets or JSP to create the output. That is, the URL that refers to the included resource is interpreted in the normal manner by the server and thus can be a servlet or JSP page. The server runs the included page in the usual way and places the output into the main page

Syntax:

<jsp:include page="relative-path-to-resource" />

Relative URLs that do not start with a slash are interpreted relative to the location of the main page. Relative URLs that start with a slash are interpreted relative to the base Web application directory, not relative to the server root. For example, consider

a JSP page in the headlines Web application that is accessed by the URL http://host/headlines/sports/table-tennis.jsp. The table-tennis.jsp file is in the sports subdirectory of whatever directory is used by the headlines Web application. Now, consider the following two include statements.

```
<jsp:include page="bios/cheng-yinghua.jsp" />
<jsp:include page="/templates/footer.jsp" />
```

Example

```
<P>
Here is a summary of our three most recent news stories:
<OL>
<LI>
<jsp:include page="/WEB-INF/Item1.html" />
<LI>
<jsp:include page="/WEB-INF/Item2.html" />
<LI>
<jsp:include page="/WEB-INF/Item3.html" />
</OL>
</BODY></HTML>
```

Including pages at page translation time: the *include* directive

The include directive is used to to include a file in the main JSP document at the time the document is translated into a servlet (which is typically the first time it is accessed).
The syntax is as follows:
<%@ include file="Relative URL" %>

include directive as works as a preprocessor: the included file is inserted character for character into the main page, then the resultant page is treated as a single JSP page. So, the fundamental difference between jsp:include and include directive is the time at which they are invoked: jsp:include is invoked at request time, whereas the include directive is invoked at page translation time. The difference in between two is summarize in following Table.

	jsp:include Action	include Directive
What does basic syntax look like?	`<jsp:include page="..." />`	`<%@ include file="..." %>`
When does inclusion occur?	Request time	Page translation time
What is included?	Output of page	Actual content of file
How many servlets result?	Two (main page and included page each become a separate servlet)	One (included file is inserted into main page, then that page is translated into a servlet)
Can included page set response headers that affect the main page?	No	Yes
Can included page define fields or methods that main page uses?	No	Yes
Does main page need to be updated when included page changes?	No	Yes
What is the equivalent servlet code?	`include` method of `RequestDispatcher`	None

Forwarding request with *jsp:Forward*

JSP forward action tag is used for forwarding a request to the another resource (It can be a JSP, static page such as html or Servlet). Request can be forwarded with or without parameter. In this tutorial we will see examples of <jsp:forward> action tag.

Syntax:

1) Forwarding along with parameters.

<jsp:forward page="display.jsp">
<jsp:param ... />
<jsp:param ... />
<jsp:param ... />

...
<jsp:param ... />
</jsp:forward>

2) Forwarding without parameters.

<jsp:forward page="Relative_URL_of_Page" />

Relative_URL_of_Page: If page is in the same directory where the main page resides then use page name itself as I did in the below examples.

JSP Forward Example 1 – without passing parameters

In this example we are having two JSP pages – index.jsp and display.jsp. We have used <jsp:forward> action tag in index.jsp for forwarding the request to display.jsp. Here we are not passing any parameters while using the action tag. In the next example we will pass the parameters as well to another resource.

index.jsp

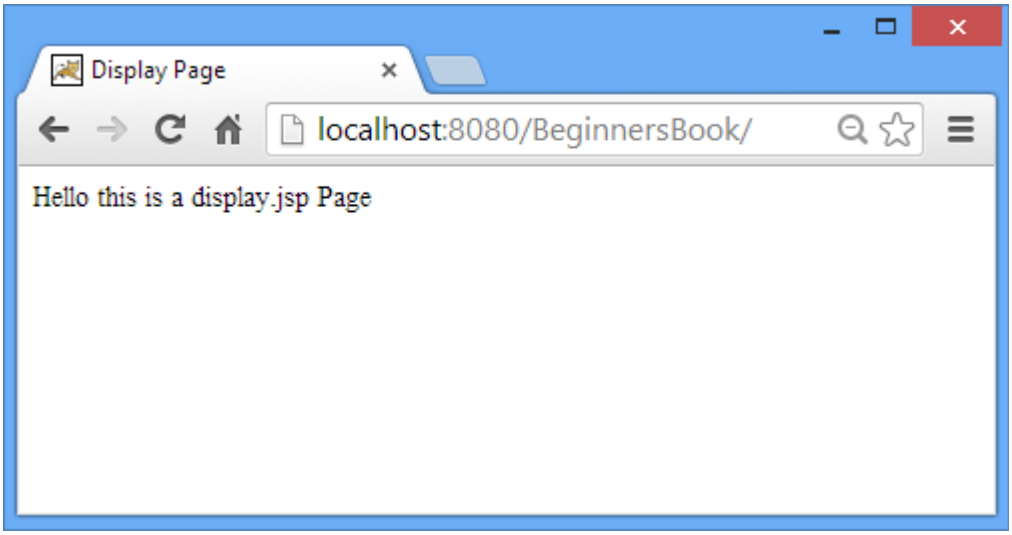
```
<html>
<head>
<title>JSP forward action tag example</title>
</head>
<body>
<p align="center">My main JSP page</p>
<jsp:forward page="display.jsp" />
</body>
</html>
```

display.jsp

```
<html>
<head>
<title>Display Page</title>
</head>
<body>
Hello this is a display.jsp Page
</body>
</html>
```

Output:

Below is the output of above cpde. It is basically the content of display.jsp, which clearly shows that index.jsp didn't display as it forwarded the request to the **display.jsp** page.



jsp:include is used to combine output from the main page and the auxiliary page. Instead of that jsp:forward is used to obtain the complete output from the auxiliary page.

For example, here is a page that randomly selects either page1.jsp or page2.jsp to output.

```
<% String destination;
if (Math.random() > 0.5)
{ destination =
"/examples/page1.jsp";
} else {
destination = "/examples/page2.jsp";
}
%>
```

Including applets for java plug-in

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.

<jsp:plugin type= "applet | bean" code= "nameOfClassFile" codebase= "directoryNameOfClass File"

</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.

index.jsp

```
1.  <html>
2.      <head>
3.          <title>Mouse Drag</title>
4.      </head>
5.      <body bgcolor="khaki">
6.  <h1>Mouse Drag Example</h1>
7.
8.   <jsp:plugin align="middle" height="500" width="500"
9.       type="applet" code="MouseDrag.class" name="clock" codebase="."/>
10.
11.     </body>
12. </html>
```

Attributes Description of jsp:plugin tag

Attribute name	Required	Description
Type	**Yes**	Specifies type of the component: applet or bean.
Code	**Yes**	Class name of the applet or JavaBean, in the form of *packagename.classname*.class.
codebase	**Yes**	Base URL that contains class files of the component.
align	Optional	Alignment of the component. Possible values are: left, right, top, middle, bottom.
archive	Optional	Specifies a list of JAR files which contain classes and resources required by the component.
height, width	Optional	Specifies height and width of the component in pixels.
hspace, vspace	Optional	Specifies horizontal and vertical space between the component and the surrounding components, in pixels.
jreversion	Optional	Specifies JRE version which is required by the component. Default is 1.2.
name	Optional	Name of the component.
title	Optional	Title of the component.

nspluginurl, **iepluginurl**	Optional	Specifies URL where JRE plugin can be downloaded for Netscape or IE browser, respectively.
mayscript	Optional	Accepts true/false value. If true, the component can access scripting objects (Javascript) of the web page.

WHY USE BEANS?

Beans are regular Java classes that follow some simple conventions defined by the JavaBeans specification; beans extend no particular class, are in no particular package, and use no particular interface.

There are several advantages we can achieve while using bean. With beans in general, visual manipulation tools and other programs can automatically discover information about classes that follow this format and can create and manipulate the classes without the user having to explicitly write any code.

In JSP in particular, use of JavaBeans components provides three advantages over script lets and JSP expressions that refer to normal Java classes.

1. No Java syntax. By using beans, page authors can manipulate Java objects using only XML-compatible syntax: no parentheses, semicolons, or curly braces. This promotes a stronger separation between the content and the presentation and is especially useful in large development teams that have separate Web and Java developers.

2. Simpler object sharing. When you use the JSP bean constructs, you can much more easily share objects among multiple pages or between requests than if you use the equivalent explicit Java code.

3. Convenient correspondence between request parameters and object properties. The JSP bean constructs greatly simplify the process of reading request parameters, converting from strings, and putting the results inside objects.

WHAT ARE BEANS?

Beans are simply Java classes that are written in a standard format. beans are the three simple points outlined in the following list.

• A bean class must have a zero-argument (default) constructor.

You can satisfy this requirement either by explicitly defining such a constructor or by omitting all constructors, which results in a zero-argument constructor being created automatically.

• A bean class should have no public instance variables (fields).

To be a bean that is accessible from JSP, a class should use accessor methods instead of allowing direct access to the instance variables. It is an important design strategy in object-oriented programming.

• Persistent values should be accessed through methods called getXxx and setXxx.

For example, if your Car class stores the current number of passengers, you might have methods named getNumPassengers (which takes no arguments and returns an int) and setNumPassengers (which takes an int and has a void return type).

In such a case, the Car class is said to have a property named numPassengers (notice the lowercase n in the property name, but the uppercase N in the method names). If the class has a getXxx method but no corresponding setXxx, the class is said to have a read-only property named xxx.

The one exception to this naming convention is with Boolean properties: they are permitted to use a method called isXxx to look up their values. So, for example, your Car class might have methods called isLeased (which takes no arguments and returns a boolean) and setLeased (which takes a boolean and has a void return type), and would be said to have a boolean property named leased (again, notice the lowercase leading letter in the property name).

USING BEANS: BASIC TASK:

You use three main constructs to build and manipulate JavaBeans components in JSP

pages:

- **jsp:useBean.** In the simplest case, this element builds a new bean.

  It is normally used as follows:

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

  If you supply a scope attribute the jsp:useBean element can either build a new bean or access a preexisting one.

- **jsp:getProperty.** This element reads and outputs the value of a

property. Reading a property is a shorthand notation for calling a

method of the form getXxx. This element is used as follows:

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

- **jsp:setProperty**. This element modifies a bean property (i.e., calls a method of the form setXxx). It is normally used as follows:

  <jsp:setProperty name="beanName" property="propertyName" value="propertyValue" />

The following subsections give details on these elements. Building Beans: jsp:useBean The jsp:useBean action lets you load a bean to be used in the JSP page. Beans provide a very useful capability because they let you exploit the reusability of Java classes without sacrificing the convenience that JSP adds over servlets alone.

The simplest syntax for specifying that a bean should be used is the following.

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

For example, the JSP action

<jsp:useBean id="book1" class="coreservlets.Book" />

can normally be thought of as equivalent to the scriptlet

<% coreservlets.Book book1 = new coreservlets.Book(); %>

Installing Bean Classes

The bean class definition should be placed in the same directories where servlets can be installed, not in the directory that contains the JSP file. Just remember to use packages. Thus, the proper location for individual bean

classes is WEB-INF/classes/subdirectoryMatchingPackageName

EXAMPLE: STRINGBEAN

Following program presents a simple class called StringBean that is in the coreservlets package. Because the class has no public instance variables (fields) and has a zero-argument constructor since it doesn't declare any explicit constructors, it satisfies the basic criteria for being a bean.

Since StringBean has a method called getMessage that returns a String and another method called setMessage that takes a String as an argument, in beans terminology the class is said to have a

String property called message.

Following Program shows a JSP file that uses the StringBean class. First, an instance of

StringBean is created with the jsp:useBean action as follows.

<jsp:useBean id="stringBean" class="coreservlets.StringBean" />

After this, the message property can be inserted into the page in either of the following

two ways.

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

<%= stringBean.getMessage() %>

StringBean.java

```
package coreservlets;
/** A simple bean that has a single String property
* called message.
*/
public class StringBean {
private String message = "No message specified";
public String getMessage() {
return(message);
}
public void setMessage(String message) {
this.message = message;
}
}
```

Following Program shows a JSP file that uses the StringBean class. First, an instance of StringBean is created with the jsp:useBean action as follows.

<jsp:useBean id="stringBean" class="coreservlets.StringBean" />

After this, the message property can be inserted into the page in either of the following two ways.

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

<%= stringBean.getMessage() %>

The message property can be modified in either of the following two ways.

<jsp:setProperty name="stringBean" property="message" value="some message" />

<% stringBean.setMessage("some message"); %>

StringBean.jsp

```
<HTML>
<HEAD>
<TITLE>Using JavaBeans with JSP</TITLE>
<LINK REL=STYLESHEET
HREF="JSP-Styles.css"
TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
<TR><TH CLASS="TITLE">
Using JavaBeans with JSP</TABLE>
<jsp:useBean id="stringBean" class="coreservlets.StringBean" />
<OL>
<LI>Initial value (from jsp:getProperty):
<I><jsp:getProperty name="stringBean"
property="message" /></I>
<LI>Initial value (from JSP expression):
<I><%= stringBean.getMessage() %></I>
<LI><jsp:setProperty name="stringBean"
property="message"
value="Best string bean: Fortex" />
Value after setting property with jsp:setProperty:
<I><jsp:getProperty name="stringBean"
property="message" /></I>
<LI><% stringBean.setMessage("My favorite: Kentucky Wonder"); %>
Value after setting property with scriptlet:
<I><%= stringBean.getMessage() %></I>
</OL>
</BODY></HTML>
```


Using JavaBeans with JSP - Netscape
File Edit View Go Bookmarks Tools Window Help
http://localhost/beans/StringBean.jsp
Using JavaBeans with JSP
1. Initial value (from jsp:getProperty): No message specified
2. Initial value (from JSP expression): No message specified
3. Value after setting property with jsp:setProperty: Best string bean: Fortex
4. Value after setting property with scriptlet: My favorite: Kentucky Wonder
Document: Done (0.06 secs)