



Cognizant
Technology
Solutions



Cognizant
Academy



JSP

Basic

C3: Protected

About the Author

Created By:	Mohamed, Hajura (Cognizant), 117960
Credential Information:	Executive – Cognizant Academy
Version and Date:	J2EE/PPT/0306/1.0

Cognizant Certified Official Curriculum



Icons Used



Questions



Hands-on Exercise



A Welcome Break



Test Your Understanding



Coding Standards



Reference



Demo









Key Contacts

Introduction:

The Java Server Page (JSP) is a Web-tier specification that supplements the Servlet specification. It is useful in the development of Web interfaces for enterprise applications. JSP technology makes available all the dynamic capabilities of Java Servlet technology, but provides a more natural approach to creating static content. It segregates content generation from content presentation.

Objective:

After completing this chapter, you will be able to:

-  Explain a JSP page
-  Describe the lifecycle of a JSP page
-  Explain static and dynamic content
-  Explain JSP scripting elements
-  Transfer control to a Web component
-  Explain the JavaBean component

The main features of JSP technology are:

- It provides a language for developing JSP pages, which are text-based documents that describe how to process a request and construct a response.
- It is an expression language for accessing server-side objects.
- It provides mechanisms for defining extensions to the JSP language.

What is a JSP Page?

- JSPs are server resident Java classes, like servlets, and they execute in the J2EE Server environment.
- Servlets generate the content as well as the necessary HTML syntax to present them to the browser.
- JSPs differentiate content from presentation.
- JSPs appear like an HTML document, embedded with JSP specific tags.
- JSPs have a file extension `.jsp`.
- The JSP directives are responsible for generating dynamic content.
- The HTML part takes care of formatting and presentation.
- JSPs are ultimately implemented as Servlets.
- A `.jsp` file is always compiled to servlets when they are loaded for the first time, and subsequently whenever the JSP Page is modified.
- JSPs are much simpler than servlets and are easier to develop.

What happens during a JSP Page Request?

- When a request is mapped to a JSP page, the Web container first checks whether the JSP page's servlet is older than the JSP page.
- If the servlet is older, then the Web container translates the JSP page into a servlet class and compiles the class. During development, one of the advantages of JSP pages over servlets is that the build process is performed automatically.

The lifecycle of a JSP page contains the following phases:

- Translation and compilation phase
- Execution phase

The Lifecycle of a JSP Page (Contd.)

Translation and compilation phase:

- During the translation phase, static data is transformed into code that will emit the data into the response stream.
- JSP elements are treated as follows:
 - Directives are used to control how the Web container translates and executes the JSP page.
 - Scripting elements are inserted into the JSP page's servlet class.
 - Expression language expressions are passed as parameters to calls to the JSP expression evaluator.
 - `jsp:[set|get]Property` elements are converted into method calls to JavaBeans components.

The Lifecycle of a JSP Page (Contd.)

- `jsp:[set|get]Property` elements are converted into method calls to JavaBeans components.
- `jsp:[include|forward]` elements are converted into invocations of the Java Servlet API.
- The `jsp:plugin` element is converted into browser-specific markup for activating an applet.
- Custom tags are converted into calls to the tag handler that implements the custom tag.

The Lifecycle of a JSP Page (Contd.)

Execution phase:

This phase can control various JSP page execution parameters by using page directives. The following are the two parts of the Execution phase:

- **Buffering:**

- When a JSP page is executed, the output written to the response object is automatically buffered.
- The size of the page buffer can be set using the following page directive:
`<%@ page buffer="none|xxxkb" %>`
- A larger buffer allows more content to be written before anything is actually sent back to the client, thus providing the JSP page with more time to set appropriate status codes and headers or to forward to another Web resource. A smaller buffer decreases server memory load.
- Allows the client to start receiving data more quickly.

- **Handling errors:**
 - To specify an error, the Web container should forward the control to an error page.
 - If an exception occurs, include the following page directive at the beginning of your JSP page:

```
<%@ page errorPage="file_name"%>
```

Example:

```
<%@ page errorPage="errorpage.jsp"%>
```

The following page directive at the beginning of `errorpage.jsp` indicates that it is serving as an error page:

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

- JSP is a tag-based language
- The tags help the container to interpret the enclosed script suitably.
- JSP tags fall into the following categories:
 - Directives
 - Scripting Elements
 - Standard Actions
- JSP tags are case sensitive

- Directives affect the overall structure of the servlet that results from the translation of JSP.
- These are used to set global values for the JSP file as a whole.

page directive:

- Defines the attributes like import, session, and so on, which is common to the entire JSP page.
- The syntax for the `page` directive is:

```
<% page import="java.rmi.*,java.utils.*" session="true"%>
```

include directive:

- It directs the container to include the specified JSP, HTML, and other file types, in the current file.
- The specified resource is copied inline. This happens during translation time.
- Any subsequent changes to the included resource will not be reflected in the JSP, unless the JSP undergoes some modification forcing the container to recompile it.
- The file is included as follows:

```
<% include file="/hello.html" %>
```

The taglib directive:

- Allows the page to use custom JSP tags.

- Scripting elements are further subdivided into
 - **Declarations:**
 - Declarations enclose Java code that defines class-wide variables and methods.
 - They are declared using `<% ! javacode %>` tag sets.
 - Declarations are initialized when the JSP page is initialized.
 - They are automatically made available to other declarations, expressions, and code within that page.

– Scriptlets:

- A scriptlet is a block of Java code that will be executed during run time (request processing time) by the JSP container.
- It is enclosed with `<% java statements %>`
- Multiple scriptlets in the same JSP page are combined together in the same order when the container generates the servlet.
- Because the scriptlets allow you to write fully functional Java code, these are powerful tools in the hands of the JSP programmer.

– Expressions:

- An expression is a scriptlet that is evaluated by the JSP Container and *“sent to the client”* for being displayed.
- The tags discussed earlier normally instruct the container and are not involved in display.
- You enclose an expression in a tag as:

```
<%= "the value of i = " i %>
```



Standard Actions



- Standard actions have the tags that define the behavior of the JSP during run time.
- The result of a standard action command is normally sent to the client.
- A standard action tag is incorporated as follows:

```
<jsp:include page="myjsp.jsp" flush="true" />
```
- The `include` action covers the file contents during run time (request processing time) unlike the `include` directive, which includes the file contents during compile time (translation time).
- So, if the included file is modified subsequently, the `include` action output will reflect the change, whereas the `include` directive will not.
- Other frequently used standard action tags are:

```
<jsp:useBean>  
<jsp:setProperty>  
<jsp:getProperty>  
<jsp:param>  
<jsp:include>  
<jsp:forward> etc.,
```

Static and Dynamic Content

- Fixed (hard coded) contents of a JSP page are referred as static content.
- Static content can be expressed in any text-based format, such as HTML, WML, and XML. The default format is HTML.
- Content type is included in the `page` directive. For example:

```
<%@ page contentType="text/vnd.wap.wml"%>
```
- The content generated by the JSP page during execution phase are referred as dynamic contents.
- Dynamic contents are generated by the use of various JSP tags.

Example of scriptlet tag:

```
<%  
for(int i=0;i< 10;i++) {  
    out.println("<b>Hello World. This is a scriptlet test " + i  
    + "</b><br>");  
    System.out.println("This goes to the System.out stream " +  
    i);  
}  
%>
```

Sample JSP Code

```
<html>
<body>
  <%@ page language = java%> ← Directive
  <%! int count = 0           %> ← Declaration
  <% count++                  %> ← Scriptlet
  Welcome you are visitor no:
  <%=count                    %> ← Expression
</body>
</html>
```



Creating and Using a JavaBeans Component



- The JavaBeans is both a specification and a framework of APIs.
- It is platform independent.
- It enables the developer to write portable and reusable components.
- It should have no public instance variables (fields):
 - Use accessor methods instead of allowing direct access to fields
 - Persistent values should be accessed through methods
- It called `getXxx` and `setXxx` methods:
 - If a class has method `getTitle` that returns a `String`, the class is said to have a `String` property named `title`.
 - Boolean properties use `isXxx` instead of `getXxx`.



Creating and Using a JavaBeans Component (Contd.)



- **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" />
```

- **jsp:getProperty**

This element reads and outputs the value of a bean property. It is used as follows:

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

- **jsp:setProperty**

This element modifies a bean property (it calls a method of the form `setXxx`). It is normally used as follows:

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



HTML Form:

```
<HTML>
<BODY>
<FORM METHOD=POST ACTION="SaveName.jsp">
What's your name? <INPUT TYPE=TEXT NAME=username SIZE=20>
<BR>
What's your e-mail address? <INPUT TYPE=TEXT NAME=email
    SIZE=20>
<BR>
What's your age? <INPUT TYPE=TEXT NAME=age SIZE=4> <P>
<INPUT TYPE=SUBMIT>
</FORM>
</BODY>
</HTML>
```




Cognizant
Technology
Solutions

Demo1 - Beans and Form Processing (Contd.)



Cognizant
Academy

Java Bean

```
public class UserData {  
    private String username;  
    private String email;  
    private int age;  
    public void setUsername( String value ) {  
        username = value;  
    }  
    public void setEmail( String value ) {  
        email = value;  
    }  
}
```



Cognizant
Technology
Solutions

Demo1 - Beans and Form Processing (Contd.)



Cognizant
Academy

```
public void setAge( int value ) {  
    age = value;  
}  
  
public String getUsername() {  
    return username;  
}  
  
public String getEmail() {  
    return email;  
}  
  
public int getAge() {  
    return age;  
}  
  
}
```



Cognizant
Technology
Solutions

Demo1 - Beans and Form Processing (Contd.)



Cognizant
Academy

To collect data from the bean:

```
<jsp:useBean id="user" class="UserData" scope="session"/>
<HTML>
  <BODY> You entered<BR>
    Name: <%= user.getUsername() %>
    <BR> Email: <%= user.getEmail() %>
    <BR> Age: <%= user.getAge() %>
    <BR>
  </BODY>
</HTML>
```



Invoking another Web component through `include` directive and `include` action:

```
<!-- File Name: includeAction.jsp - ->
<html>
<head>
    <title>Include Action test page</title>
</head>
<body>
    <h1>Include Action test page</h1>
    <h2>Using the include directive</h2>
    <%@ include file="included2.html" %>
    <%@ include file="included2.jsp" %>
    <h2>Using the include action</h2>
    <jsp:include page="included2.html" flush="true" />
    <jsp:include page="included2.jsp" flush="true" />
</body>
</html>
```

Demo2 - JSP: include (Contd.)



```
<!-- included2.jsp -->  
<%@ page import="java.util.Date" %>  
<%= "Current date is " + new Date() %>  
<!-- included2.html -->  
<p>This is some static text in the html file</p>
```

Demo3 - JSP: forward



```
--forward.jsp
-- demonstrates the use of forward action
<%
if ((request.getParameter("userName").equals("Richard")) &&
    (request.getParameter("password").equals("xyzzzy"))) {
%>
<jsp:forward page="forward2.jsp" />
<% } else { %>
<%@ include file="forward.html" %>
<% } %>
```

Demo3 - JSP: forward (Contd.)



File name: forward2.jsp

```
<html>
  <head>
    <title>Forward action test: Login successful!</title>
  </head>
  <body>
    <h1>Forward action test: Login successful</h1>
    <p>Welcome, <%= request.getParameter("userName") %>
  </body>
</html>
```

Demo3 - JSP: forward (Contd.)



File name: forward.html

```
<html>
  <head>
    <title>Forward action test page</title>
  </head>
  <body>
    <h1>Forward action test page</h1>
    <form method="post" action="forward.jsp">
      <p>Please enter your username:
      <input type="text" name="userName">
      <br>and password:
      <input type="password" name="password">
      </p>
      <p><input type="submit" value="Log in">
    </form>
  </body>
</html>
```


- **Allow time for questions from participants**





JSP: Summary



- JSP (Java Server Pages) is a dynamic Web page generation technology. It is a layer above the Servlet API.
- Usage of a JSP allows users to embed dynamic content generation chunks into a HTML page, during the access to the JSP the JSP page gets converted on the fly to a servlet, which is equivalent in functionality to the coded JSP.
- JSPs contain predefined tags, which provide specific functionality by getting converted to appropriate artifacts, when the JSP is translated to a servlet.
- JSPs can use JavaBeans to isolate view / user interface generation from logic implementation.



Cognizant
Technology
Solutions

JSP: Source



Cognizant
Academy

- http://java.sun.com/j2ee/tutorial/1_3-fcs/index.html
- Professional Java Server Programming J2EE Edition By Wrox Author Team

Disclaimer: Parts of the content of this course is based on the materials available from the Web sites and books listed above. The materials that can be accessed from linked sites are not maintained by Cognizant Academy and we are not responsible for the contents thereof. All trademarks, service marks, and trade names in this course are the marks of the respective owner(s).



Cognizant
Technology
Solutions



Cognizant
Academy



You have
successfully
completed JSP.