Not a client language. Server side language that can be used to generate client side code.

Chapter 12

SECTION III: JAVA SERVER PAGES

Introduction To Java Server Pages

Since modern enterprise applications are moving from two-tier towards three-tier and N-tier architectures, there arises a need of discovering different ways to deliver applications and data to users.

The major drawback with the traditional thick client that holds the entire application on the local computer is that it becomes difficult distributing and updating the application.

Web based clients provide an excellent alternative for building Intranet and Internet enterprise applications. The new Java Server Pages [JSP] technology, part of the Java EE 5, gives web and Java developers a simple yet powerful mechanism for creating such web applications.

The JSP technology:

- Is a language for developing JSP pages, which are text-based documents that describe how to process a request and construct a response

- Is an expression language for accessing server-side objects
- Allows creating web content with both static and dynamic components
- Makes available all the dynamic capabilities of Java Servlet technology but provides a more natural approach to creating static content
- Provides mechanisms for defining extensions to the JSP language
- Provides developers with the ability to access remote data via mechanisms such as Enterprise JavaBeans [EJB], Remote Method Invocation [RMI] and Java DataBase Connectivity [JDBC]
- Provides developers with the ability to encapsulate and separate program logic from the presentation i.e. the HTML elements, to help maximize code reuse and flexibility. This separation of logic and presentation is a major advantage over the web application architectures such as Java Servlets and CGI scripts

A JSP page is a text document that contains two types of text:

- **Static data**, which can be expressed in any text-based format such as HTML, SVG, WML and XML
- **JSP elements**, which construct dynamic content

The recommended file extension for the source file of a JSP page is **.jsp**. The page can be composed of a top file that includes other files that contain either a complete JSP page or a fragment of a JSP page. The recommended extension for the source file of a fragment of a JSP page is **.jspf**.

Why Use Java Server Pages

The benefits of using JSP:

Nobody Can Borrow The Code

When a web site that does something really cool, which attracts the attention of the users, developers look at the source code of the page and copy the JavaScript or other code into their own pages which then do the same cool stuff.

With JSP, this issue does not arise at all. The code written runs and remains on the Web server. All of its functionality is handled before the page is sent to a Browser.

Occupies A Lot Of Space

Java Server Pages consume extra hard drive and memory [RAM] space. For every 30K JSP file on the Web server there will be a corresponding much larger class file created. This essentially more than doubles the hard drives, space requirements, to store JSP pages. Considering how easily a JSP file can **include** a large data file for display, this is a real concern. Each JSP's, class file data, must be loaded into the Web server's memory. This means the Web server may eventually store the entire JSP document tree in its memory.

A few JVMs have the ability to remove such class file data from memory [i.e. RAM], however, the programmer generally has no control over the rules for reclaiming used memory. For large sites memory reclamation may not be aggressive enough, this will slow down Web Server performance. Using template engines there's no need to duplicate page data into a second file, so hard drive space is spared. Template engines also give the programmer complete control over how templates are cached in memory.

JSP v/s Servlets

Servlets provide the ability to build dynamic content for Web sites using Java and are supported by all major Web servers.

Why JSP If Java Already Has An API For Answering HTTP Requests

Since Java Server Pages get automatically translated into Java Servlets. Hence, there is no difference between what can be done by a JSP or a Servlet. Both JSP and Servlets are executed by a Java Virtual Machine [JVM]. This eliminates the need for a Web server to create a new process each time a web page request arrives. A really huge advantage over CGI scripts.

The distinct advantage of JSP over Servlets is that the JSP allows a logical division between what is displayed [i.e. the generated HTML] and the Web server side code spec that dictates what content fills the page. It is easy to modify the look and feel of what is delivered by a JSP without having to alter any Web server side, Java code spec.

While it's true that anything done with a JSP can also be done with using a Servlet, JSPs provide a nice clean separation of the application's presentation layer from its data manipulating layer. JSP's are simpler to craft than Servlets. Servlets and JSPs work well together.

Life Cycle Of A JSP Page

Since JSP architecture is based on Servlet architecture, a **Java enabled** Web server provides the mechanism that deals with both. Individual JSPs are actually text files stored on the Web server and accessed via their path. For example: if a JSP page named main.jsp resides at the root of a web application named **sample**, it would be accessed by a request to http://localhost:8080/sample/main.jsp, assuming that the Web server is bound to port 8080 on localhost, when the JSP page is requested.

The Web server's JSP engine uses the content of the JSP file to generate the required Servlet's source code. The generated source is then compiled and run by the Web server's Servlet engine. Servlet code execution services a client's request.

Once JSP code spec has been converted to a Servlet and Servlet code spec has been compiled, the compiled version is saved and used to service additional client requests according to the standard Servlet life cycle.

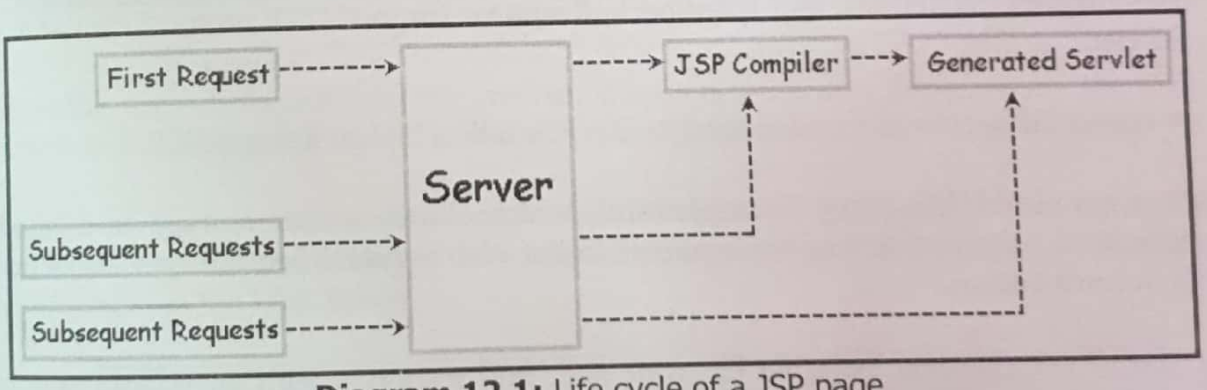

Diagram 12.1: Life cycle of a JSP page

Whenever the JSP file changes, the Web server automatically detects the change and rebuilds the corresponding Servlet. The JSP to Servlet compilation phase imposes a slight delay the first time a page is retrieved. Many web servers permit pre-compilation of JSPs to get around this problem.

How Does A JSP Function

JSP page code spec can be broken into 2 categories:

- **Elements** that are processed by the JSP engine on the Web server
- **Template data** or everything other than such elements, that the JSP engine **ignores**

Chapter

13

SECTION III: JAVA SERVER PAGES

Getting Started With Java Server Pages

A JSP page is very similar to an HTML or a XML page. Every JSP page holds text marked with a variety of tags. A standard JSP page is not a valid XML page however there is a separate JSP tag syntax, which allows using JSP tags within XML documents.

In JSP, [when compared to regular HTML] the special JSP tags are not processed by the browser instead they are processed by the Web server, allowing the page content to change dynamically. JSP thus functions in the same manner in which other dynamic page generation systems such as Microsoft's **A**ctive **S**erver **P**ages [ASP] or PHP functions.

JSP tags are nothing but holders of Java code spec in an HTML page. This code spec is then executed by the web server whenever the page is requested.

Following are the components, which a JSP page will have:

Comments

Comments in HTML are as follows:

```
<!-- Comments -->
```

Comments are useful especially when building and maintaining pages is of prime importance. Comments identify a particular block of code which otherwise is in pure HTML language. Comment is always a part of the page and hence it will also be seen along with the HTML code spec if the **View Source** option of the Web browser is chosen. This is never considered as a disadvantage but the download time usually increases by a tiny fraction of a second longer.

A serious concern is that sometimes HTML comments might contain implementation details that might be confidential. This issue can be solved by using JSP's comments instead of HTML comments as shown below:

```
<%-- Comments --%>
```

When the JSP engine comes across such a tag **<%-- --%>**, it recognizes it as a comment and is not inserted in the Servlet it builds. The comment will therefore not be sent to the user [Web browser] and thus will not be visible using the browser's View Source option.

This clearly indicates that what goes into a JSP file, can and will be different from what comes out.

There are two more ways of commenting the code spec:

1. One-liner Comment:

```
<% //This is the one line comment. %>
```

2. Using /**/ comment tags inside the <% %> tags:

```
<% /* %>
  This is also considered as comment.
<% */ %>
```

The second commenting method is a bit difficult and also time consuming. The reason being that when the JSP page is translated, the text within the /**/ comment tag is still converted into Java statements, which discharge the text. However, because the Java statements are surrounded by /* */ comment, they are ignored by the compiler. Therefore, it is better to use the <%-- --%> comment instead of /**/ comment.

This feature plays a significant role during the debugging and testing processes as developers can use JSP comments to selectively block out code spec or tags from compilation.

Template Text

A JSP page as mentioned earlier is a regular web page with some JSP elements embedded in the code spec. These JSP elements help create dynamic content for each request.

Template text is everything in the web page that is not a JSP element. Template text can be any text such as HTML, WML, XML or even plain text. Hence everything that is not a JSP directive, action or scripting element is template text. Template text is sent to the Web browser as it is, i.e. no interpretation is done on it. JSP can thus be used to generate any kind of text based output such as WML, XML or even plain text. The JSP container is no where dependent on what the template text represents.

When a Web server processes a JSP page, the template text and dynamic content generated by JSP elements available in the JSP page are merged together and the result is then sent as the response to the Web browser. In all respects, the output looks just like normal HTML, follows all the same syntax rules and is simply **passed through** to the client by the Servlet created to handle the page.

The one minor exception to the template text being passed straight through is that, if the developer wants to have <% in the output, then the developer needs to put a backslash before the percentage sign like <\% in the template text.

REMINDER

JSP has no dependency on HTML. It can be used with any markup language.

```
<%@ page contentType="text/html" %>                          JSP Element
<%@ taglib prefix="c" uri="http://java.sun.com" %>
<HTML>
  <HEAD>
     <TITLE>Template Text</TITLE>                            Template Text
  </HEAD>

  <BODY BGCOLOR="pink">
     <H1>Template Text Of JSP</H1>
  <%-- Calculate the sum of 2 + 2 dynamically --%>           JSP Element
     2 + 2 = <c:out value="${2 + 2}" />
  </BODY>                                                    Template Text
</HTML>
```

Diagram 13.1

In the diagram 13.1 besides JSP elements, notice that the JSP page also contains regular HTML.

JSP Elements

Let's take a look at the following code spec:

```
<HTML>
  <HEAD>
     <TITLE>JSP Code Spec</TITLE>
  </HEAD>
  <BODY>
     Hello World!
  </BODY>
</HTML>
```

The above code spec does not hold Java code. But these HTML elements do form a valid JSP file. The above code can be saved as HelloWorld.jsp and then installed into a web application and the server would later access it as a JSP resource. This indicates that a JSP page looks a lot like an HTML page.

JSP pages usually provide dynamic behavior. This means they are supposed to change the response as per specific client requests. JSP pages can be given dynamic behavior by embedding Java code into the page. This concludes that a JSP page is a Web page with bits of Java code embedded in them.

Java code cannot be written anywhere in the page. The JSP translator needs to be informed, which bits of code is Java and which bits of code is regular HTML. To do this, the JSP specification provides HTML elements. These elements enclose the Java code spec in a JSP page and are categorized as follows:

- Directives
- Scripting Elements
- Action Elements [Discussed in Chapter 15: Action Elements Of JSP]

Directives

JSP directives serve special processing information about the page to the JSP server. JSP directives are used to:

- Set global values such as class declaration
- Methods to be implemented
- The scripting language for the page and so on

They do not produce any output that is visible to the client. They are typically used to provide a direction to the JSP server processing the page.

REMINDER

JSP directives affect only the JSP file that holds it.

In a regular JSP page, directives are characterized using the @ character within the <% %> tags.

Syntax:

```
<%@ <DirectiveName> attributes="<Value>" %>
```

HINT

The **white space** following the characters **<%@** and before the characters **%>** is optional.

Following are the directives available in JSP:

- The **page** directive
- The **include** directive
- The **taglib** directive

The page Directive

The page directive allows importing classes, customizing the Servlet superclass, setting the content type and so on. A page directive can be placed anywhere within the document.

The page directive supports several attributes that affect the whole page. A single JSP page can contain multiple page directives. During the translation, all page directives are **assimilated and applied** together to the same page along with any of its **static include files**. The page directive **does not apply** to any **dynamic include files**.

Page directives can be placed anywhere in the document, but are often placed at the very top:

```
<%@ page language="java" contentType="text/html" import="java.util.*" %>
<HTML>
   <HEAD>
      <TITLE>JSP page Directive Example</TITLE>
   </HEAD>
   <BODY>
      Hello World!
   </BODY>
</HTML>
```


HINT

There can be only one occurrence of any attribute/value pair defined by the page directive in a given JSP.

Syntax:

```
<%@ page attributes="<Value>" %>
```

Following are the attributes associated with the page directive:

- **language:** Defines which scripting language the file uses. The scripting language can be used in scriptlets, declarations and expressions in a JSP page and any other included files. The default value is java [Till version 1.2, the only allowed value was java]

  The scope of this directive spans the entire file. When used more than once, only the first occurrence of the directive is significant.

  Syntax:

  ```
  <%@ page language="java" %>
  ```

- **extends:** Holds the name of the Java language class that the Servlet extends. It is an optional fully qualified class name of a class implementing javax.servlet.jsp.JspPage or javax.servlet.jsp.HttpJspPage. The class mentioned here need not be a Servlet class but has to be a valid class

  The scope of this directive spans the entire JSP file. When used more than once, only the first occurrence of the directive is significant.

  Syntax:

  ```
  <%@ page extends="javax.servlet.http.HttpServlet" %>
  ```

REMINDER

The **extends** attribute should be used with extreme caution, since it restricts some of the decisions that a JSP server can make. For example, the server looks for javax.servlet.**Servlet** as being the contract agreement between itself and the page. Implicit variables such as request, response, application and so on would not be available to the JSP if some other classes were extended.

- **import:** Imports all the available classes from the specified Java package into the current JSP page. This page directive helps the JSP page to use other Java classes. This attribute can hold a comma-separated list of Java language package names or class name that the Servlet imports. These packages / classes are then available to scriptlets, expressions and declarations within the JSP page

  This directive can be specified multiple times within a JSP file to import different packages.

  Syntax:

  ```
  <%@ page import="java.io.*, java.util.Hashtable" %>
  ```

- **session:** Allows the page to participate in an HTTP session. By default, the first time a user accesses a JSP on a site, a session is created for that user and the user is issued a cookie

  If this attribute is set to true, a predefined session variable of type **HttpSession** is bound to an existing session if one exists, otherwise a new session is created and bound to it. If this attribute is set to false, no sessions are used and any attempt to access the session variable will result in errors whilst the JSP page is being translated into a Servlet.

  Syntax:

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

- **buffer:** Holds the data stream size through which the application will pass data to the browser. In other words, it controls the buffered output for a JSP page

  The default is 8kb. If the value is set to none then no buffering occurs and all output is sent directly to the **ServletResponse** using a **PrintWriter**. If a buffer size is specified, example: 24kb, then output is buffered with a buffer size not less than what is specified.

  Syntax:

  ```
  <%@ page buffer="8kb" %>
  ```

- **autoFlash:** Flushes output buffer when full

  If buffering is turned on and this value is set to **true**, data is automatically sent to the user when the buffer is full. If buffering is turned on and this value is set to **false**, then an exception is raised when the buffer overflows. If buffering is not turned on, this value has no effect. The default value is true.

  Syntax:

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

- **isThreadSafe:** States whether JSP engine can pass multiple requests simultaneously to the page

A value set to **true** [the **default**] indicates that a new thread is started and therefore requests are handled simultaneously i.e. the JSP engine can send multiple, concurrent client requests to the JSP page. In case the value **true** is used, a code has to be written in the JSP page to synchronize the multiple client threads. A value set to **false** indicates that the JSP engine sends client requests one at a time to the JSP page.

Syntax:

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

HINT

It is recommended to always use the value true for the **isThreadSafe** page directive and handle multithread issues by avoiding JSP declarations and ensure that all objects used by the page are thread-safe.

- **info:** Is used by the developer to add information for a page i.e. it is used by the developer to add details such as author, version, copyright, date and so on. In other words, it is the text that is incorporated verbatim into the compiled JSP page, which can be retrieved with the Servlet.getServletInfo() method. The text entered can be used by a server administration tool

  Syntax:

  ```
  <%@ page info="<Message>" %>
  ```

- **errorPage:** A page or context relative URI path to a JSP page, Servlet or a static page to redirect to in case an exception is thrown by code spec in the current page

  By default in such a situation, a standard Error 500 page is thrown, however if the value of this attribute is set to the path of another page, the contents of that page are sent instead. In other words it is a page that should process any Throwables thrown but not caught in the current page

  Syntax:

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

- **isErrorPage:** Indicates whether or not the current page can act as the error page for another JSP page

  The default value for this attribute is false. When the value is set to true, an exception object can be used in the JSP page.

  Syntax:

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

- **contentType:** Sets the mime type and character set of the JSP page. The default value for **MIME type** is:
  - **text/html** for JSP style JSP tags

- text/xml for XML style JSP tags
- ISO-8859-1 for Character set

When used more than once, only the first occurrence of this directive is significant. This can have either of the form:

TYPE-MIME type

OR

TYPE-MIME type; charset=CHARSET

with an optional white space after the ;.

CHARSET or character encoding if specified must be the IANA value for character encoding.

HINT

IANA is the Internet Assigned Numbers Authority. The names and other details of the different character encoding can be found at http://www.iana.org/.

Charsets indicate how written characters are encoded, so that pages can support languages that use different scripts.

Syntax:

```
<%@ page contentType="text/html; charset=Shift_JIS" %>
```

- **pageEncoding:** Is the page source character encoding i.e. it is the character set of the current page

The value held is the IANA character set name. The default character encoding is **ISO-8859-1** [Latin script] for **JSP** style tags and **UTF-8** [8-bit Unicode encoding] for **XML** style tags.

Syntax:

```
<%@ page pageEncoding="ISO-8859-1" %>
```

The include Directive

This directive allows including files at the time the JSP page is translated into a Servlet. In other words, this directive is used to include an HTML file, JSP or Servlet file into a JSP file. The included files are usually used for navigation, tables, headers and footers that are common to multiple pages.

The included file should be a static file i.e. it is included into a JSP file at the time of compilation and once the JSP file is compiled any changes in the included file will not be reflected. Most JSP engines usually keep track of the included file and **recompile** the JSP if it **changes**.

REMINDER

There are **no restrictions** on the number of include directives that may appear in a single JSP file.

Syntax:

```
<%@ include file="<Filename/Relative URL>" %>
```

The URL specified is usually interpreted relative to the JSP page that refers to it. With relative URLs in general, the system can be asked to interpret the URL relative to the home directory of the Web server by beginning the URL with a forward slash.

Since the contents of the included file are parsed as regular JSP text, the included file can have static HTML, scripting elements, directives and actions.

For example: There are many web sites on the internet that include a small menu bar on each page. Due to issues with HTML frames, this menu bars are usually implemented by way of a small table across the top of the page or down the left-hand side, with the HTML code spec forming the menu bar repeated for each page in the site. The include directive can be of great use for doing this, saving the developers from the maintenance nightmare of actually copying the HTML into each separate file. Since include allows the developer to write the menu bar once and then include the same on multiple website pages.

Example:

```
<HTML>
  <HEAD>
     <TITLE>JSP include Directive Example</TITLE>
  </HEAD>
  <BODY>
     <%@ include file="/navigate.html" %>
  </BODY>
</HTML>
```

REMINDER

Since the include directive inserts the file at the time the page is translated, if the menu bar changes, a need arises to re-translate all the JSP pages that refer to it. But in reality the menu bar changes very infrequently. If, however, the included files change more often, then use the **jsp:include** action instead. This includes the file at the time the JSP page is requested.

Following is the only attribute associated with the include directive:

- **file:** A page-relative or context-relative URI path to the file that will be included at the current position in the file. This attribute includes a static file, merging its content with the including page before the combined result is converted to a JSP page implementation class. A page can contain multiple include directives

  Syntax:

```
<%@ include file="blogs.html" %>
```

Example:
The contents of the **include.jsp** file are:

```
<HTML>
  <HEAD>
     <TITLE>JSP include directive Test</TITLE>
  </HEAD>
  <BODY>
     <FONT COLOR="Red">
        Current date and time:<%@ include file="date.jsp" %>
     </FONT>
  </BODY>
</HTML>
```

The contents of the **date.jsp** file are:

```
<%@ page import="java.util.*" %>
<%= (new java.util.Date()).toLocaleString() %>
```

Explanation:
In the above example, **include.jsp** is the main JSP application. In include.jsp, **include** directive is used to include the **date.jsp** file. Here, an include directive is used to insert a JSP file. The JSP page recompiles, in case the included file changes.

In date.jsp, **page** directive is used to import the required package. The next line of the date.jsp file uses an expression statement to print the current date and time.

When the main file i.e. the include.jsp file is run, the output sent to the browser is as follows:

```
The current date and time are Aug 28, 2006 3:09:27 PM
```

The taglib Directive

A taglib is a collection of Custom tags that can be used by the page. In other words, this directive allows the page to use **Custom tags** inside the JSP page.

Custom tags were introduced in JSP version 1.1. These tags allow developers to hide complex server side code spec from web designers.

A taglib directive in a JSP is a link to an XML document that describes a set of Custom tags. This XML document also determines which Tag Handler class implements the action of each tag. The XML document names the tag library [a compressed file], which holds the custom tags. The JSP engine uses this tag library to determine what to do when it comes across custom tags in the JSP.

Once this directive has been used to indicate the dependence of a page on a specific tag library, all the custom tags defined in that library are made available for use on that page.

WARNING

If the Custom tags are developed or used, then the tag prefixes such as jsp, jspx, java, Servlet, sun and sunw cannot be used as they are reserved by Sun Microsystems.

Syntax:

```
<%@ taglib uri="<TagLibraryURI>" prefix="<TagPrefix>" %>
```

Following are the attributes associated with the taglib directive:

- **uri:** A **U**niform **R**esource **I**dentifier [URI] that identifies the tag library descriptor, which is used to uniquely name the set of custom tags and inform the server what to do with the specified tags. There is no need to represent the actual location i.e. path or URL instead it simply has to be a name. It means that the JSP engine does not try to request something from the **uri** in the taglib directive. The **uri** attribute is not used as a location

  Syntax:

```
<%@ taglib uri="http://www.sharanamshah.com/tags" %>
```

- **prefix:** Defines the prefix string in <prefix>:<tagname> pair. This is used to define the custom tag. It is the prefix, which is used in the action element names for all actions in the library

For example, if this value is set to **mytag** then when the server comes across any element that starts as <mytag:tagname .../> in the JSP, it references the tag library descriptor specified in the URI. Prefixes such as jsp, jspx, javax, servlet, sun and sunw are reserved by Sun Microsystems and thus cannot be used.

Syntax:

```
<%@ taglib prefix="mytag" %>
```

Scripting Elements

Scripting elements are used to include scripting code spec, which is the usual Java code spec, within a JSP page. These elements allow:

- Declaring variables and methods
- Including arbitrary scripting code
- Evaluating an expression

Using scripting elements, developers can directly embed the Java code spec into a JSP page. This code spec may also include the code spec that generates output that is to be sent back to the user i.e. Web browser.

HINT

Scripting elements are always written in a specific scripting language that was designated for the page using the **Language** attribute of the **page** directive.

In the absence of an explicit scripting language specification, the JSP server assumes Java as the default scripting language.

Following are the type of Scripting Elements available in JSP:

- Declarations
- Scriptlets
- Expressions

Declarations

A declaration is a Java code block that holds code spec for class-wide variables and methods declarations in the generated class file. In other words, JSP declaration allows the developer to declare variables and methods.

Declarations are initialized when a JSP page is initialized and has **class** scope. Variables or methods declared within the declaration element can be called by any other code blocks. In other words, anything defined in the declaration is available throughout the JSP file, to other declarations, expressions or code. Declarations are found within the <%! ... %> tag.

WARNING

Declarations do not generate output and so they are used with Expressions or Scriptlets.

Syntax:

```
<%!
    Java variable and method declaration(s)
%>
```

REMINDER

Always end variable declarations with a semicolon, as any content must be valid Java statements.

```
<%!
    Int num = 0;
%>
```

Methods can be declared as follows:

```
<%!
    public void countNum()
    {
        Int num = 10;
    }
%>
```

Any method declared within the declaration element becomes an instance of the JSP page implementation class and therefore, it is global to the entire JSP page.

Scriptlets

A Scriptlet is a block of Java code spec that is executed at run time. Scriptlets also known as JSP code fragments are embedded within <% ... %> tags. A Scriptlet can produce output passed via an output stream back to the client.

Syntax:

```
<%
    Scriptlet Code Spec
%>
```

Multiple scriptlets when used are combined in the compiled class exactly in the order in which they appear in the JSP. Scriptlets can also modify objects inside them as a result of method invocations.

In reality, a JSP gets compiled into a Servlet, all the code appearing between the <% and %> in the JSP, gets into the **service()** method of this Servlet, as is, in the order it is laid out. Hence it is processed for every request that the Servlet processes.

There can be just about any valid Java code within a scriptlet. For example, the following displays a string Hello World! using <H1>, <H2> and <H3> HTML elements, combining the use of expressions and scriptlets:

```
<%
    for (int num=1; num<=3; num++)
    {
%>
    <H<%=num%>>Hello World!</H<%=num%>>
<%
    }
%>
```

WARNING

While making use of scriptlets ensure that all the start and end braces are in place. Even if one of the braces is found missing, the code that the JSP engine generates is not syntactically correct and the error message that the developer gets is not that easy to interpret.

Since Java statements are included in the scriptlets, the following code spec is a valid scriplet:

```
<%
    Int num=0;
    /* Some code */
%>
```

The above code looks similar to that of declaration element, but scriptlets are different from declarations in the following ways:

- Methods can be defined only by Declarations and not by Scriptlets
- Variables declared in a Scriptlet are **local** to a method in the JSP page implementation class i.e. they are visible only within their defining code block. Variables declared in a

- Declaration are instance variables of the JSP page implementation class i.e. these variables are visible to all other code statements or methods in the page

Expressions

An **expression** is a shorthand notation for a scriptlet that outputs a value to a response stream bound to a client. JSP expression element is explicitly used for output generation.

After an expression is evaluated the result is converted to a string and displayed. If any part of the expression is an object, then the conversion is done using the **toString()** method of the object.

REMINDER

An expression begins within **<%= ... %>**.

HINT

The most important thing to note is that the expression must not end with semi-colon, as expressions will not necessarily be legal Java code. If the expression ends with a semi colon, then the code will not be syntactically correct.

Syntax:

```
<%= Expressions to be evaluated %>
```

Example:

```
Date: <%= new java.util.Date() %>
```

Explanation:
The above examples display the current date and time.

Output

```
Date: Tue Aug 29 12:25:59 GMT+05:30 2006
```

A Sample JSP Page

After understanding what are:

- Directives
- Scripting elements

Lets use them in the following example. This example has a user interface that accepts data using an HTML form and displays the same on page submission. To do so, use the same application named **MyWebApplication** that was deployed earlier in Chapter 12: Introduction To Java Server Pages.

This example will therefore require creating the following files:

- AuthorMasterForm.jsp: The User Interface
- header.jsp: The menu bar [To demonstrate the include directive]
- AuthorMaster.jsp: The file that will process the data captured by d/e form contained in AuthorMasterForm.jsp

Create a JSP page named AuthorMasterForm.jsp with the following code spec. This file after it is created is placed under the **jsp** directory of the same application named **MyWebApplication** that was deployed earlier in *Chapter 12: Introduction To Java Server Pages*

```
<!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=UTF-8">
    <TITLE>Author Information</TITLE>
  </HEAD>
  <BODY BGCOLOR="pink">
  <%-- Including a JSP file to form the page header. --%>
    <%@ include file="header.jsp" %>
  <!-- Initializing a form object, which will submit data
  captured on the form. -->
    <FORM ACTION="/MyWebApplication/MyJSP/AuthorMaster" METHOD="post"
          NAME="FrmAuthMast">
    <!-- Parent Table begins. -->
      <TABLE ALIGN="center" BGCOLOR="PINK" BORDER="0" CELLPADDING="0"
             CELLSPACING="0" NAME="tblOuter" WIDTH="50%">
      <!-- Parent Table: First Row. -->
        <TR HEIGHT="300" VALIGN="top">
          <TD ALIGN="center" BORDER="1" COLSPAN="10">
          <BR>
          <!-- First Child Table to hold the d/e Form
          objects, to Capture / Control data begins. -->
          <!-- HTML code spec follows for designing the form
          fields Author Name, Degree, Speciality and Date Of
          Birth and button such as Save. -->
            <TABLE ALIGN="center" BORDER="1"
                   BORDERCOLOR="SKYBLUE" CELLPADDING="2"
                   CELLSPACING="0" NAME="tblFirstChild"
                   WIDTH="100%">
```



```
<!-- First Child Table: First Row: Column to
hold the Form Title . -->
<!-- Inserting table row to display the form
title -->
    <TR BGCOLOR="BLACK">
      <TD ALIGN="left" COLSPAN="2">
      <!-- Formatting the font of title -->
        <FONT SIZE="2" COLOR="white"><B>Author Master
          </B></FONT>
      </TD>
    </TR>
<!-- First Child Table: Second, Third, Fourth &
Fifth Rows: Two columns, first column to hold
the label and the second to hold the data
capture object that captures data. -->
<!-- Inserting table row that holds table data
label and text box of Author Name -->
    <TR>
      <TD ALIGN="right" WIDTH="25%">Author Name:
      </TD>
      <TD ALIGN="left">
        <INPUT MAXLENGTH="35" NAME="txtAuthorName"
          TYPE="text" SIZE="25">
      </TD>
    </TR>
<!-- Inserting table row that holds table data
label and text box of Degree -->
    <TR>
      <TD ALIGN="right">Degree:</TD>
      <TD ALIGN="left">
        <INPUT MAXLENGTH="255" NAME="txtDegree"
          TYPE="text" SIZE="42">
      </TD>
    </TR>
<!-- Inserting table row that holds table data
label and text box of Speciality -->
    <TR>
      <TD ALIGN="right">Speciality:</TD>
      <TD ALIGN="left">
        <INPUT MAXLENGTH="255" NAME="txtSpeciality"
          TYPE="text" SIZE="42">
      </TD>
    </TR>
<!-- Inserting table row that holds table data
label and text box of Date of birth -->
    <TR>
      <TD ALIGN="right">Date Of Birth:</TD>
```

```
                        <TD ALIGN="left">
                            <INPUT MAXLENGTH="255" NAME="txtDOB"
                                TYPE="text" SIZE="">
                        </TD>
                    </TR>
                    <!-- First Child Table: Last Row: Columns to
                    hold data control objects. -->
                    <!-- Inserting table row that holds inputs for
                    Save button -->
                        <TR>
                            <TD COLSPAN="2" ALIGN="RIGHT">
                                <INPUT NAME="cmdSubmit" TYPE="submit".
                                    VALUE="Save">
                            </TD>
                        </TR>
                    </TABLE>
                </TD>
            </TR>
        </TABLE>
    </FORM>
</BODY>
</HTML>
```

Explanation:
This is a basic HTML form contained within a JSP file. The most significant part of the code spec is:

```
<%@ include file="header.jsp" %>
```

Here, a header is being included, which, when rendered / executed will be displayed as shown in diagram 13.2.1. The header code spec is held inside the header.jsp file.

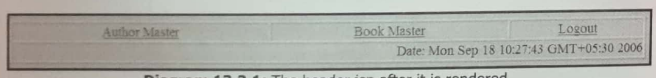

Diagram 13.2.1: The header.jsp after it is rendered

Create a JSP page named header.jsp with the following code spec. This file after it is created is placed under the **jsp** directory in the same application named **MyWebApplication** that was deployed earlier in *Chapter 12: Introduction To Java Server Pages*:

```
<%-- Directives --%>
<%@ page language="java" import="java.util.*" %>
<TABLE BORDER="1" ALIGN="center" WIDTH="100%">
    <TR>
        <TD ALIGN="center"><A HREF="AuthorMasterForm.jsp">Author Master</A>
        </TD>
```



```
        <TD ALIGN="center"><A HREF="BookMaster.html">Book Master</A></TD>
        <TD ALIGN="center"><A HREF="javascript:alert('Goodbye');">Logout</A></TD>
    </TR>
    <TR>
        <TD COLSPAN="3" ALIGN="right">
        <%-- Expressions --%>
            Date: <%= new java.util.Date() %>
        </TD>
    </TR>
</TABLE>
```

Explanation:
This is a JSP page that forms a menu bar with the following menu items:

- **AuthorMaster:** Is a Link to the AuthorMasterForm.jsp file
- **BookMaster:** Is a Link to the BookMaster.html file [created earlier in *Chapter 7: Working With Servlets*]
- **Logout:** Displays a message "Goodbye"

Additionally, this page also displays current date and time.

The primary purpose of this page is to serve the menu bar and should therefore be included using the **include directive** wherever a menu bar is required.

Create a JSP page named AuthorMaster.jsp with the following code spec. This is the file that will process the data captured by the Author Master Form [AuthorMasterForm.jsp]. This file after it is created is placed under the **jsp** directory in the same application named **MyWebApplication** that was deployed earlier in *Chapter 12: Introduction To Java Server Pages*:

```
<%-- Directives --%>
<%@ page language="java" contentType="text/html" %>
<HTML>
  <HEAD>
      <TITLE>Author Information</TITLE>
  <HEAD>
  <BODY>
  <%-- Declarations --%>
      <%!
          String AuthorName;
          String Degree;
          String Speciality;
          String DOB;
      %>
  <%-- Scriptlets --%>
      <%
          AuthorName = request.getParameter("txtAuthorName");
```

```
            Degree = request.getParameter("txtDegree");
            Speciality = request.getParameter("txtSpeciality");
            DOB = request.getParameter("txtDOB");
        %>
        <H1>This is Author Information</H1>
    <%-- Expressions --%>
        Author Name:  <%=AuthorName%><BR>
        Degree:  <%=Degree%><BR>
        Speciality:  <%=Speciality%><BR>
        Date of Birth:  <%=DOB%>
    </BODY>
</HTML>
```

Explanation:

Most of this file's content is plain HTML but it also has, interspersed within it, special JSP tags.

```
<%@ page language="java" contentType="text/html" %>
```

This is a page directive with **language** and **contentType** attributes set.

```
<%!
    String AuthorName;
    String Degree;
    String Speciality;
    String DOB;
%>
```

This is a Declaration section where four variables are declared. These variables will be available throughout the JSP file, to other declarations, expressions or code blocks.

```
<%
    AuthorName = request.getParameter("txtAuthorName");
    Degree = request.getParameter("txtDegree");
    Speciality = request.getParameter("txtSpeciality");
    DOB = request.getParameter("txtDOB");
%>
```

This is a Scriptlet that holds four variables declared earlier are assigned the data captured by the Author Master form. The data captured is available in the *POST* array and accessed using **request.getParameter()**.

```
    Author Name:  <%=AuthorName%>
    Degree:  <%=Degree%>
    Speciality:  <%=Speciality%>
    Date of Birth:  <%=DOB%>
```


To display data entered by the user which is held inside memory variables `<%= . . . %>` expression is used. In side the expression tag the variable declared is accessed.

Once the files are ready and placed appropriately the deployment can begin.

To do so **web.xml** file will undergo some changes as follows:

vi. Append the following lines [marked in dark shade] in the web.xml file:

```
<?xml version="1.0"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
                xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                                    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <servlet>
        <servlet-name>BookInformation</servlet-name>
        <display-name>BookInformation</display-name>
        <servlet-class>BookInformation</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>AuthorMaster</servlet-name>
        <display-name>AuthorMaster</display-name>
        <jsp-file>AuthorMaster.jsp</jsp-file>
    </servlet>
    <servlet-mapping>
        <servlet-name>BookInformation</servlet-name>
        <url-pattern>/MyServlets/BookInformation</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>AuthorMaster</servlet-name>
        <url-pattern>/MyJSP/AuthorMaster</url-pattern>
    </servlet-mapping>
</web-app>
```

Explanation:

```
<jsp-file>AuthorMaster.jsp</jsp-file>
```

In the above code spec, <jsp-file> element holds the path to a JSP file within the web application. Since in this case, the AuthorMaster.jsp file is placed in the root of the web application after it is deployed no path is mentioned.

vii. Now build the web application using the **asant** utility

viii. Next, deploy the web application

ix. Run the deployed web application, which now holds the AuthorMasterForm.jsp, AuthorMaster.jsp and header.jsp in addition to the previous examples, using the URL http://localhost:8080/MyWebApplication/AuthorMasterForm.jsp as shown in diagram 13.2.2

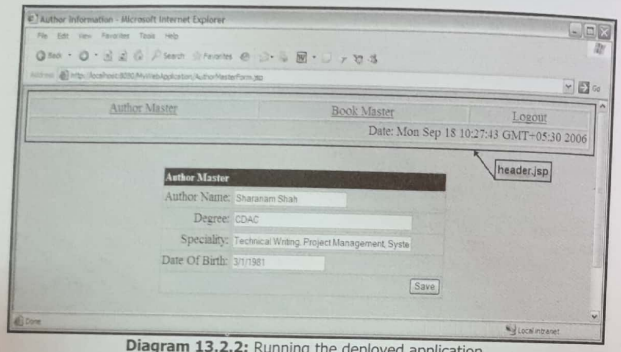

Diagram 13.2.2: Running the deployed application

REMINDER

The AuthorMasterForm.jsp file when executed as shown in diagram 13.2.2 also displays the header.jsp page, which holds the menu bar as defined earlier.

Enter the desired data and click Save. The AuthorMasterForm.jsp file will send the captured data [using the post method] to the AuthorMaster.jsp file for further processing. The AuthorMaster.jsp file displays the captured data as shown in diagram 13.2.3.

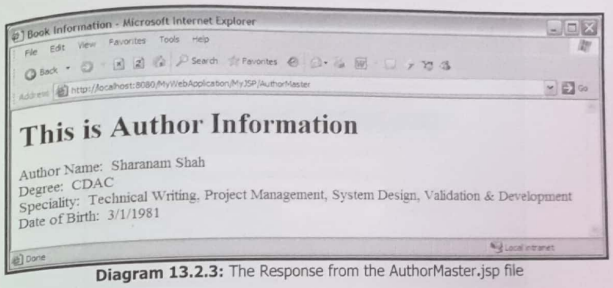

Diagram 13.2.3: The Response from the AuthorMaster.jsp file

Since the control is passed on to the JSP file, the URL changes to http://localhost:8080/MyWebApplication/MyJSP/AuthorMaster. This is the same URL that was specified in the ACTION attribute of the FORM element of the AuthorMasterForm.jsp file:

```
<FORM ACTION="/MyWebApplication/MyJSP/AuthorMaster" METHOD="post"
      NAME="FrmAuthMast">
```

REMINDER

Action Elements are discussed in Chapter 15: Action Elements Of JSP.

Hands On Exercises

Create an HTML page [user interface] for capturing Book information. This form will accept data from the user and display the same upon page submission i.e. when the user enters the required data and clicks **Save**.

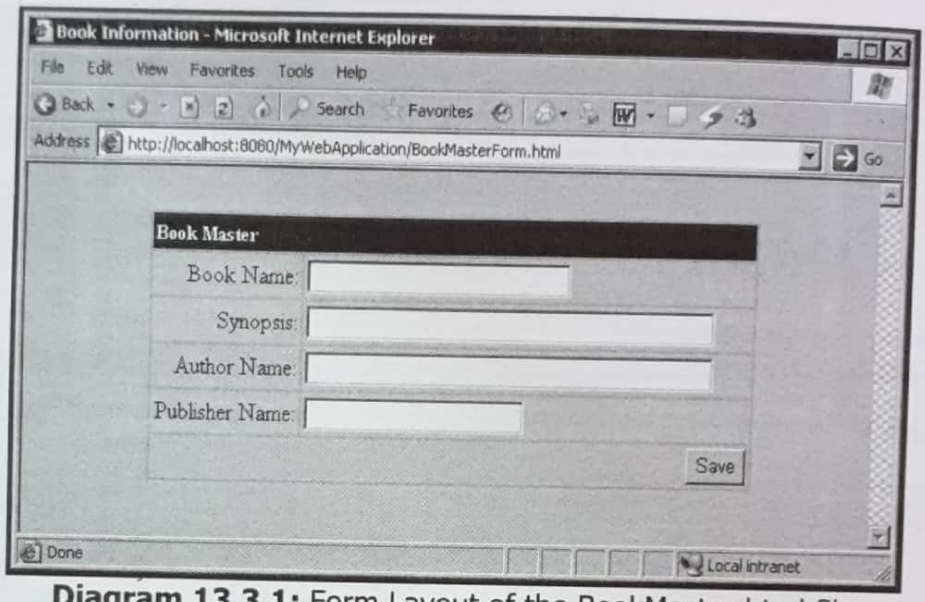

Diagram 13.3.1: Form Layout of the BookMaster.html file

When the data captured is submitted, the control should pass to a JSP page called **BookMaster**. This page should display the data captured by the earlier HTML form.

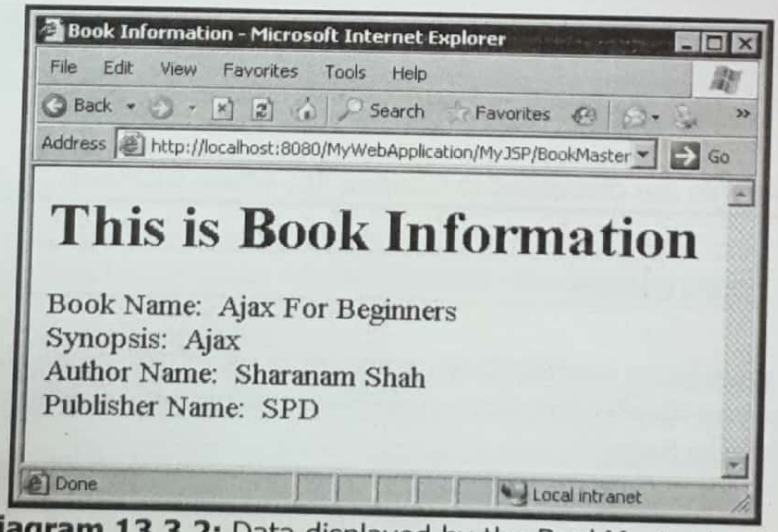

Diagram 13.3.2: Data displayed by the BookMaster.jsp file