

Instructor Notes:

Add instructor notes here.

Java Server Pages (JSP)

Lesson 06: JSP Scripting Elements



Instructor Notes:

This lesson explains
the various JSP
Scripting elements

Lesson Objectives

In this lesson, you will learn about:

- Forms of Scripting Elements
 - JSP Expressions
 - JSP Scriptlets
 - JSP Declarations
- Predefined Variables
- Examples using Scripting Elements



Instructor Notes:

Introduce the three forms of scripting elements.

6.1: Forms of Scripting Elements

JSP Scripting Elements

JSP scripting elements insert Java code into the generated servlet. There are three forms of scripting elements, namely:

- Expressions: `<%= expression %>`
- Scriptlets: `<% scriptlet code %>`
- Declarations: `<%! Declarative code %>`

Forms of Scripting Elements:**JSP Scripting Elements:**

- With JSP scripting elements Java code into the servlet that will be generated from the current JSP page.
- There are three forms of scripting elements:
 - **Expressions** of the form `<%= expression %>` that are evaluated and inserted into the output.
 - **Scriptlets** of the form `<% code %>` that are inserted into the servlet's service method.
 - **Declarations** of the form `<%! code %>` that are inserted into the body of the servlet class, outside of any existing methods.
- Each of these are described in more detail in the subsequent slides.

Instructor Notes:

Explain the significance of JSP expressions. Do mention that we cannot use a semicolon to end an expression otherwise it will result into exception. The expression is equivalent to a single `out.println (...)` statement. The results of the expression are displayed as part of the JSP output.

6.1.1: JSP Expressions JSP Expressions



A JSP expression is used to insert Java values directly into the output.

- Form :

```
<%= Java expression%>
```

- Example:

```
hostname: <%=request.getRemoteHost( )%>
```

- The result of the expression is evaluated, converted to a String, and inserted in the JSP page.

Forms of Scripting Elements:

JSP Expressions:

- The **JSP expression** is evaluated at run-time (when the page is requested), and thus has full access to information about the request. The JSP expression example shown on the above slide will retrieve the remote host information from the pre-defined **request object** and insert it in the **JSP** page. The result is in fact stored in the **out object** and inserted where the **expression** appears in the JSP page.
- To simplify these expressions, there are a number of predefined variables that can be used. These implicit objects are discussed in more detail later, but for the purpose of expressions, the most important ones are as follows:
 - **request:** the `HttpServletRequest`;
 - **response:** the `HttpServletResponse`;
 - **session:** the `HttpSession` associated with the request (if any); and
 - **out:** the `PrintWriter` (a buffered version of type `JspWriter`) used to send output to the client.
- When using Java as the scripting language, remember that:
 - A semicolon should not be used to end an expression.
 - The expression tag can contain any expression that is valid according to the Java Language Specification.
 - Expressions are evaluated in left-to-right order as they appear in the tag.

Instructor Notes:

For JSP expression as well as JSP scriptlet the expression results or the Java code is inserted into the **Service** method (`_jspService()`) of the translated Servlet.

6.1.2: Scriptlets JSP Scriptlets



A JSP scriptlet is used to insert Java code into the `_jspService` method.

- Form :

```
<%Java code%>
```

- Example:

```
<%
String queryData = request.getQueryString();
out.println ("Attached GET data: " + queryData);
%>
```

Forms of Scripting Elements:

JSP Scriptlets:

Scriptlets have the following form: `<% Java Code %>`. Scriptlets are executed when the JSP engine processes the client request. If the scriptlet produces output, then the output is stored in the `out` object, from which it can be displayed.

Note that code inside a scriptlet gets inserted exactly as written. Any static HTML (template text) before or after a scriptlet gets converted to print statements. This means that scriptlets need not contain complete Java statements, and blocks left open can affect the static HTML situated outside of the scriptlets.

For example:

Consider the following JSP fragment, containing mixed template text and scriptlets:

```
<% if (Math.random() < 0.5) { %>
Have a <B>nice</B> day!
<% } else { %>
Have a <B>lousy</B> day!
<% } %>
```

```
if (Math.random() < 0.5) {
    out.println("Have a <B>nice</B> day!");
} else {
    out.println("Have a <B>lousy</B> day!");
}
```

Instructor Notes:

The Java code in the JSP declaration is put into the translated Servlet class outside the service method. Hence, it can be used to declare instance variables or define methods (private or public) in the generated Servlet class. We can even override the methods inherited from the base class of the Servlet (i.e. `jspInit`, `jspDestroy`, etc.)

6.1.3: JSP Declarations JSP Declarations



A JSP declaration is used to define methods or fields that are inserted in the Servlet class outside the `_jspService()` method.

- Form :

```
<%! Java Code%>
```

- Example: Count accesses to page since server reboot:

```
<%! private int accessCount = 0; %>
<%= ++accessCount %>
```

- Declaration can be used to override `jspInit()` and `jspDestroy()` methods of servlet.

Forms of Scripting Elements:

JSP Declarations:

- A JSP *declaration* lets helps to define methods or fields that are inserted in the servlet class (outside of the service method processing the request).
- It has the following form: **<%! Java Code %>**
- The example, shown on the slide prints out the number of times the current page has been requested since the server booted (or the servlet class was changed and reloaded). The **accessCount** variable is now an instance variable that is defined only once within the lifecycle of JSP. If, however, **accessCount** was part of a scriptlet, then it would have been a local variable whose scope would be the **service()** method!
- The JSP usually runs as multiple “**threads**” of one single instance. Different threads interfere with **variable access**, because it will be the same variable for all of them. If variables has to be used in JSP, it should be used with “**synchronized access**”, but that hurts the performance. In general, any data should go either in the “**session object**” or the “**request object**” if passing data between different JSP pages.
- JSP declarations can be used to override **jspInit()** and **jspDestroy()** methods

```
<%! public void jspInit() {
    ..... }
    public void jspDestroy(){
    ..... } %>
```

Instructor Notes:

The participants have used these pre-defined variables in Servlet – check whether they recollect the usage of each of these variables. In Servlet they had to get these objects and use them whereas in JSP they are available as Implicit objects. Do explain the difference between the `PrintWriter` object available in Servlet & JSP.

6.2: Predefined Variables Implicit Objects



Let us see some of the implicit objects in JSP:

- **request:**
 - It is the `HttpServletRequest` associated with the request.
- **response:**
 - It is the `HttpServletResponse` associated with the response.
- **out:**
 - It is the `Buffered PrintWriter` (`JspWriter`) used to send output to the client.
- **session:**
 - It is the `HttpSession` object associated with the session.

Predefined Variables:

To simplify code in **JSP expressions** and **scriptlets**, JSPs are supplied with eight automatically defined variables, sometimes are called **implicit objects**. The available variables are as follows:

- **request:** This is the **`HttpServletRequest`** associated with the request, and lets to look at the request parameters (via `getParameter`), the request type (GET, POST, HEAD, and so on), and the incoming HTTP headers (cookies, Referrer, and so on).
- **response:** This is the **`HttpServletResponse`** associated with the response to the client. Note that, since the output stream is buffered, it is legal to set HTTP status codes and response headers, even though this is not permitted in regular Servlets once any output has been sent to the client.
- **out:** This is the **`PrintWriter`** used to send output to the client. However, in order to make the response object useful, this is a buffered version of `PrintWriter` called **`JspWriter`**. Note that we can adjust the buffer size, or even turn buffering off, through use of the `buffer` attribute of the page directive. In scriptlets, need to refer to `out` explicitly in case we need to display anything.
- **session:** This is the **`HttpSession` object** associated with the session. As sessions are created automatically, this variable is bound even if there is no incoming session reference. The one exception is that if the `session` attribute of the page directive is used to turn sessions off. In this case, attempts to reference the session variable cause errors at the time the JSP page is translated into a servlet.

Instructor Notes:

The pageContext, page (listed in notes page) & exception object were not there in Servlets so explain the significance of these implicit objects in JSP.

6.2: Predefined Variables Implicit Objects



- application:
 - It is the ServletContext object.
- config:
 - It is the ServletConfig object.
- pageContext:
 - It refers to the current page.
- exception:
 - It refers to the java.lang.Exception object that represents the uncaught exception.

Predefined Variables:

- **Application:** This is the **ServletContext** as obtained via the **getServletConfig().getContext()**.
- **config:** This is the **ServletConfig** object for this page.
- **pageContext:** JSP introduced a new class called **PageContext** to encapsulate use of server-specific features like higher performance JspWriters. The advantage is that these can be accessed through this class rather than directly, code will still run on “regular” servlet/JSP engines.
- **page:** This is simply a synonym for **this**, and is not very useful in Java. It was created as a placeholder for the time when the scripting language could be something other than Java.
- **exception:** This implicit object applies only to JSP error pages – these are pages to which processing is forwarded when an exception is thrown from another JSP page. They must have the page directive **isErrorPage** attribute set to **true**. The implicit exception object is a **java.lang.Exception** instance that represents the uncaught exception that was thrown from another JSP page and that resulted in the current error page being invoked. The exception object is accessible only from the JSP error page instance to which processing was forwarded when the exception was encountered.

Instructor Notes:

Demonstrate the examples for all the three forms of Scripting elements (i.e. Expressions, Scriptlet & Declarations).

Demo: JSP Scripting Elements**JSP Expressions**

- remotehost.jsp
- predefined.jsp

JSP Scriptlets

- scriptlet.jsp
- conditional.jsp

JSP Declarations

- declaration1.jsp
- declaration2.jsp
- declaration3.jsp
- fact.jsp



Deploy web application **Lesson3-JSPScriptlets** and show demo by executing each of the above JSP pages.

Instructor Notes:

Lab on all the three forms of Scripting elements.

Lab: JSP Scripting Elements**Lab 3.1**

Instructor Notes:

Summary



In this lesson, you have learnt the following concepts:

- JSP Expressions
- JSP Scriptlets
- JSP Declarations
- Predefined Variables



Instructor Notes:**Answers for the
Review Questions:**

Answer 1: JSP
Scriptlet

Answer 2: False

Answer 3: JSP
Declaration

Review – Questions

Question 1: ____ inserts java code into `_jspService()` method.

Question 2: A JSP expression can end with a semicolon (;) . True/False

Question 3: ____ can be used to override `jspInit()` method.



Instructor Notes:**Answers for the
Review Questions:**

Answer 4:
JSPWriter

Answer 5:
application

Review – Questions

Question 4: ____ is a buffered PrintWriter used in JSP to send output to client.

Question 5: ____ implicit object is used to share data across users of the same web application.

