


© 2003 Marty Hall



Invoking Java Code with JSP Scripting Elements

JSP and Servlet Training Courses: <http://courses.coreservlets.com>
 JSP and Servlet Books from Sun Press: <http://www.coreservlets.com>

Agenda

- Static vs. dynamic text
- Dynamic code and good JSP design
- JSP expressions
- Servlets vs. JSP pages for similar tasks
- JSP scriptlets
- JSP declarations
- Predefined variables
- Comparison of expressions, scriptlets, and declarations

Uses of JSP Constructs

Simple Application

↓

Complex Application

- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Servlet/JSP combo (MVC)
- MVC with JSP expression language
- Custom tags

Design Strategy: Limit Java Code in JSP Pages

- You have two options
 - Put 25 lines of Java code directly in the JSP page
 - Put those 25 lines in a separate Java class and put 1 line in the JSP page that invokes it
- Why is the second option *much* better?
 - **Development.** You write the separate class in a Java environment (editor or IDE), not an HTML environment
 - **Debugging.** If you have syntax errors, you see them immediately at compile time. Simple print statements can be seen.
 - **Testing.** You can write a test routine with a loop that does 10,000 tests and reapply it after each change.
 - **Reuse.** You can use the same class from multiple pages.

Basic Syntax

- **HTML Text**
 - `<H1>Blah</H1>`
 - Passed through to client. Really turned into servlet code that looks like
 - `out.print("<H1>Blah</H1>");`
- **HTML Comments**
 - `<!-- Comment -->`
 - Same as other HTML: passed through to client
- **JSP Comments**
 - `<%-- Comment --%>`
 - Not sent to client
- **To get `<%` in output, use `<%=`**

Types of Scripting Elements

- **Expressions**
 - Format: `<%= expression %>`
 - Evaluated and inserted into the servlet's output. I.e., results in something like `out.print(expression)`
- **Scriptlets**
 - Format: `<% code %>`
 - Inserted verbatim into the servlet's `_jspService` method (called by service)
- **Declarations**
 - Format: `<%! code %>`
 - Inserted verbatim into the body of the servlet class, outside of any existing methods

JSP Expressions

- **Format**
 - `<%= Java Expression %>`
- **Result**
 - Expression evaluated, converted to String, and placed into HTML page at the place it occurred in JSP page
 - That is, expression placed in `_jspService` inside `out.print`
- **Examples**
 - Current time: `<%= new java.util.Date() %>`
 - Your hostname: `<%= request.getRemoteHost() %>`
- **XML-compatible syntax**
 - `<jsp:expression>Java Expression</jsp:expression>`
 - You cannot mix versions within a single page. You must use XML for *entire* page if you use `jsp:expression`.

JSP/servlet training: <http://www.coreservlets.com>

JSP/Servlet Correspondence

- **Original JSP**

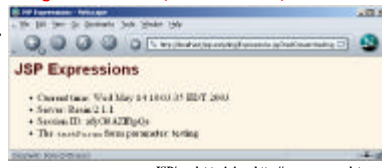
```
<H1>A Random Number</H1>
<%= Math.random() %>
```
- **Representative resulting servlet code**

```
public void _jspService(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    HttpSession session = request.getSession(true);
    JspWriter out = response.getWriter();
    out.println("<H1>A Random Number</H1>");
    out.println(Math.random());
    ...
}
```

JSP/servlet training: <http://www.coreservlets.com>

JSP Expressions: Example

```
...<BODY>
<H2>JSP Expressions</H2>
<UL>
<LI>Current time: <%= new java.util.Date() %>
<LI>Server: <%= application.getServerInfo() %>
<LI>Session ID: <%= session.getId() %>
<LI>The <CODE>testParam</CODE> form parameter:
    <%= request.getParameter("testParam") %>
</UL>
</BODY></HTML>
```



JSP/servlet training: <http://www.coreservlets.com>

Predefined Variables

- **request**
 - The `HttpServletRequest` (1st argument to `service/doGet`)
- **response**
 - The `HttpServletResponse` (2nd arg to `service/doGet`)
- **out**
 - The `Writer` (a buffered version of type `JspWriter`) used to send output to the client
- **session**
 - The `HttpSession` associated with the request (unless disabled with the `session` attribute of the page directive)
- **application**
 - The `ServletContext` (for sharing data) as obtained via `getServletContext()`.

JSP/servlet training: <http://www.coreservlets.com>

Comparing Servlets to JSP: Reading Three Params (Servlet)

```
public class ThreeParams extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
    ...
    out.println(docType +
        "<HTML>\n" +
        "<HEAD><TITLE>"+title + "</TITLE></HEAD>\n" +
        "<BODY BGCOLOR=\"#FDF5E6\">\n" +
        "<H1 ALIGN=\"CENTER\">"+ title + "</H1>\n" +
        "<UL>\n" +
        "  <LI><B>param1</B>: "
        + request.getParameter("param1") + "\n" +
        "  <LI><B>param2</B>: "
        + request.getParameter("param2") + "\n" +
        "  <LI><B>param3</B>: "
        + request.getParameter("param3") + "\n" +
        "</UL>\n" +
        "</BODY></HTML>");
    }
}
```

JSP/servlet training: <http://www.coreservlets.com>

Reading Three Params (Servlet): Result



JSP/servlet training: <http://www.coreservlets.com>

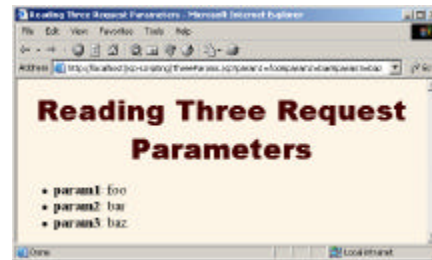
Comparing Servlets to JSP: Reading Three Params (JSP)

```
<!DOCTYPE ...>
<HTML>
<HEAD>
<TITLE>Reading Three Request Parameters</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H1>Reading Three Request Parameters</H1>
<UL>
<LI><B>param1</B>:
      <%= request.getParameter("param1") %>
<LI><B>param2</B>:
      <%= request.getParameter("param2") %>
<LI><B>param3</B>:
      <%= request.getParameter("param3") %>
</UL>
</BODY></HTML>
```

13

JSP/Servlet training: <http://www.coreservlets.com>

Reading Three Params (Servlet): Result



14

JSP/Servlet training: <http://www.coreservlets.com>

JSP Scriptlets

- **Format**
 - <% Java Code %>
- **Result**
 - Code is inserted verbatim into servlet's `_jspService`
- **Example**
 - <%
String queryData = request.getQueryString();
out.println("Attached GET data: " + queryData);
%>
 - <% response.setContentType("text/plain"); %>
- **XML-compatible syntax**
 - <jsp:scriptlet>Java Code</jsp:scriptlet>

15

JSP/Servlet training: <http://www.coreservlets.com>

JSP/Servlet Correspondence

- **Original JSP**

```
<H2>foo</H2>
<%= bar() %>
<% baz(); %>
```
- **Representative resulting servlet code**

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    HttpSession session = request.getSession();
    JspWriter out = response.getWriter();
    out.println("<H2>foo</H2>");
    out.println(bar());
    baz();
    ...
}
```

16

JSP/Servlet training: <http://www.coreservlets.com>

JSP Scriptlets: Example

- **Suppose you want to let end users customize the background color of a page**
 - What is wrong with the following code?
- ```
<BODY BGCOLOR=
 "<%= request.getParameter("bgColor") %>">
```

17

JSP/Servlet training: <http://www.coreservlets.com>

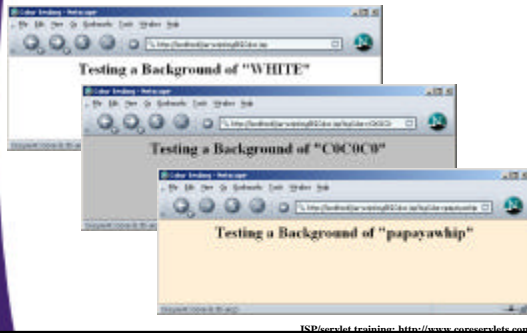
## JSP Scriptlets: Example

```
<!DOCTYPE ...>
<HTML>
<HEAD>
 <TITLE>Color Testing</TITLE>
</HEAD>
<%
String bgColor = request.getParameter("bgColor");
if ((bgColor == null) ||
 (bgColor.trim().equals(""))) {
 bgColor = "WHITE";
}
%>
<BODY BGCOLOR="<%= bgColor %>">
 <H2 ALIGN="CENTER">Testing a Background of
 "<%= bgColor %>"</H2>
</BODY></HTML>
```

18

JSP/Servlet training: <http://www.coreservlets.com>

## JSP Scriptlets: Result



## Using Scriptlets to Make Parts of the JSP File Conditional

- **Point**
  - Scriptlets are inserted into servlet exactly as written
  - Need not be complete Java expressions
  - Complete expressions are usually clearer and easier to maintain, however
- **Example**
  - `<% if (Math.random() < 0.5) { %>`  
Have a `<B>nice</B>` day!  
`<% } else { %>`  
Have a `<B>lousy</B>` day!  
`<% } %>`
- **Representative result**
  - `if (Math.random() < 0.5) {`  
out.println("Have a `<B>nice</B>` day!");  
`}`  
`else {`  
out.println("Have a `<B>lousy</B>` day!");  
`}`

## JSP Declarations

- **Format**
  - `<%! Java Code %>`
- **Result**
  - Code is inserted verbatim into servlet's class definition, outside of any existing methods
- **Examples**
  - `<%! private int someField = 5; %>`
  - `<%! private void someMethod(...) { ... } %>`
- **Design consideration**
  - Fields are clearly useful. For methods, it is usually better to define the method in a separate Java class.
- **XML-compatible syntax**
  - `<jsp:declaration>Java Code</jsp:declaration>`

## JSP/Servlet Correspondence

- **Original JSP**

```
<H1>Some Heading</H1>
<%!
 private String randomHeading() {
 return("<H2>" + Math.random() +
 "</H2>");
 }
%>
<%= randomHeading() %>
```
- **(Alternative: make randomHeading a static method in a separate Java class)**

## JSP/Servlet Correspondence

- **Possible resulting servlet code**

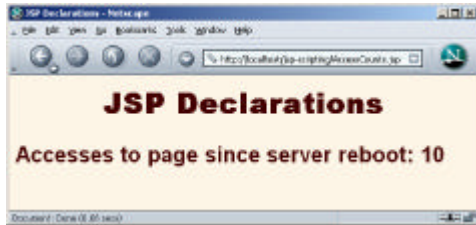
```
public class xxxx implements HttpJspPage {
 private String randomHeading() {
 return("<H2>" + Math.random() + "</H2>");
 }

 public void _jspService(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 HttpSession session = request.getSession(true);
 JspWriter out = response.getWriter();
 out.println("<H1>Some Heading</H1>");
 out.println(randomHeading());
 ...
 }
}
```

## JSP Declarations: Example

```
<!DOCTYPE ...>
<HTML>
<HEAD>
<TITLE>JSP Declarations</TITLE>
<LINK REL=STYLESHEET
 HREF="JSP-Styles.css"
 TYPE="text/css">
</HEAD>
<BODY>
<H1>JSP Declarations</H1>
<%! private int accessCount = 0; %>
<H2>Accesses to page since server reboot:
<%= ++accessCount %></H2>
</BODY></HTML>
```

## JSP Declarations: Result



## JSP Declarations: the jspInit and jspDestroy Methods

- JSP pages, like regular servlets, sometimes want to use init and destroy
- Problem: the servlet that gets built from the JSP page might already use init and destroy
  - Overriding them would cause problems.
  - Thus, it is illegal to use JSP declarations to declare init or destroy.
- Solution: use **jspInit** and **jspDestroy**.
  - The auto-generated servlet is guaranteed to call these methods from init and destroy, but the standard versions of jspInit and jspDestroy are empty (placeholders for you to override).

## JSP Declarations and Predefined Variables

- Problem
  - The predefined variables (request, response, out, session, etc.) are *local* to the `_jspService` method. Thus, they are not available to methods defined by JSP declarations or to methods in helper classes. What can you do about this?
- Solution: pass them as arguments. E.g.

```
<%!
private void someMethod(HttpSession s) {
 doSomethingWith(s);
}
%>
<% someMethod(session); %>
```
- Note that the `println` method of `JspWriter` throws `IOException`
  - Use “throws `IOException`” for methods that use `println`

## Comparing Expressions, Scriptlets and Declarations

- Task 1
  - Output a bulleted list of five random ints from 1 to 10.
    - Since the structure of this page is fixed and we use a separate helper class for the `randomInt` method, **JSP expressions** are all that is needed.
- Task 2
  - Generate a list of between 1 and 10 entries (selected at random), each of which is a number between 1 and 10.
    - Because the number of entries in the list is dynamic, a **JSP scriptlet** is needed.
- Task 3
  - Generate a random number on the first request, then show the same number to all users until the server is restarted.
    - Instance variables (fields) are the natural way to accomplish this persistence. Use **JSP declarations** for this.

## Helper Class: RanUtilities

```
package coreservlets; // Always use packages!!

/** Simple utility to generate random integers. */
public class RanUtilities {

 /** A random int from 1 to range (inclusive). */
 public static int randomInt(int range) {
 return(1 + ((int)(Math.random() * range)));
 }

 public static void main(String[] args) {
 int range = 10;
 try {
 range = Integer.parseInt(args[0]);
 } catch(Exception e) { // Array index or number format
 // Do nothing: range already has default value.
 }
 for(int i=0; i<100; i++) {
 System.out.println(randomInt(range));
 }
 }
}
```

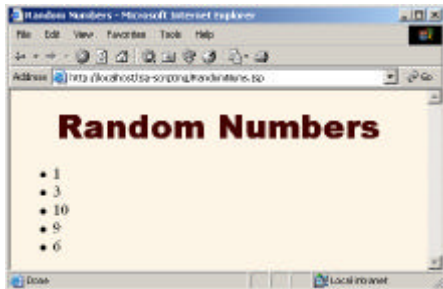
## Task 1: JSP Expressions (Code)

```
<!DOCTYPE ...>
<HTML>
<HEAD>
<TITLE>Random Numbers</TITLE>
<LINK REL=STYLESHEET
 HREF="JSP-styles.css"
 TYPE="text/css">
</HEAD>
<BODY>
<H1>Random Numbers</H1>

 <%= coreservlets.RanUtilities.randomInt(10) %>
 <%= coreservlets.RanUtilities.randomInt(10) %>
 <%= coreservlets.RanUtilities.randomInt(10) %>
 <%= coreservlets.RanUtilities.randomInt(10) %>
 <%= coreservlets.RanUtilities.randomInt(10) %>

</BODY></HTML>
```

## Task 1: JSP Expressions (Result)



ISP/coreservlet training: <http://www.coreservlets.com>

## Task 2: JSP Scriptlets (Code: Version 1)

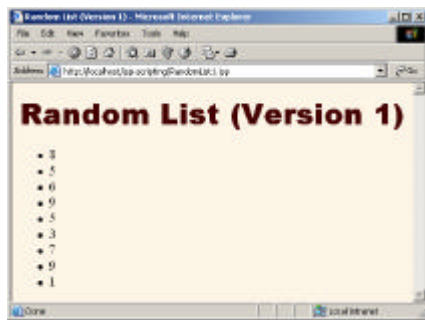
```
<!DOCTYPE ...>
<HTML>
<HEAD>
<TITLE>Random List (Version 1)</TITLE>
<LINK REL=STYLESHEET
 HREF="JSP-Styles.css"
 TYPE="text/css">
</HEAD>
<BODY>
<H1>Random List (Version 1)</H1>

<%
int numEntries = coreservlets.RanUtilities.randomInt(10);
for(int i=0; i<numEntries; i++) {
 out.println("* " +
 coreservlets.RanUtilities.randomInt(10));
}
%>

</BODY></HTML>
```

ISP/coreservlet training: <http://www.coreservlets.com>

## Task 2: JSP Scriptlets (Result: Version 1)



ISP/coreservlet training: <http://www.coreservlets.com>

## Task 2: JSP Scriptlets (Code: Version 2)

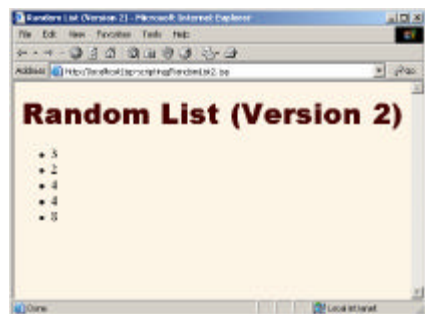
```
<!DOCTYPE ...>
<HTML>
<HEAD>
<TITLE>Random List (Version 2)</TITLE>
<LINK REL=STYLESHEET
 HREF="JSP-Styles.css"
 TYPE="text/css">
</HEAD>
<BODY>
<H1>Random List (Version 2)</H1>

<%
int numEntries = coreservlets.RanUtilities.randomInt(10);
for(int i=0; i<numEntries; i++) {
%>
<%= coreservlets.RanUtilities.randomInt(10) %>
<% } %>

</BODY></HTML>
```

ISP/coreservlet training: <http://www.coreservlets.com>

## Task 2: JSP Scriptlets (Result: Version 2)



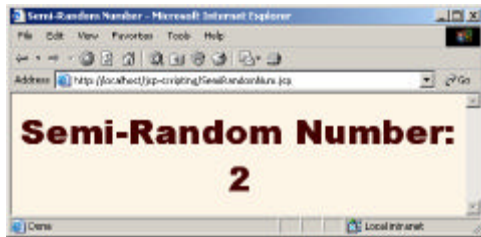
ISP/coreservlet training: <http://www.coreservlets.com>

## Task 3: JSP Declarations (Code)

```
<!DOCTYPE ...>
<HTML>
<HEAD>
<TITLE>Semi-Random Number</TITLE>
<LINK REL=STYLESHEET
 HREF="JSP-Styles.css"
 TYPE="text/css">
</HEAD>
<BODY>
<%!
private int randomNum =
 coreservlets.RanUtilities.randomInt(10);
%>
<H1>Semi-Random Number:
<%= randomNum %></H1>
</BODY>
</HTML>
```

ISP/coreservlet training: <http://www.coreservlets.com>

## Task 3: JSP Declarations (Result)



27

JSP/servlet training: <http://www.coreservlets.com>

## Summary

- **JSP Expressions**
  - Format: `<%= expression %>`
  - Wrapped in `out.print` and inserted into `_jspService`
- **JSP Scriptlets**
  - Format: `<% code %>`
  - Inserted verbatim into the servlet's `_jspService` method
- **JSP Declarations**
  - Format: `<%! code %>`
  - Inserted verbatim into the body of the servlet class
- **Predefined variables**
  - `request`, `response`, `out`, `session`, `application`
- **Limit the Java code that is directly in page**
  - Use helper classes, beans, servlet/JSP combo (MVC), JSP expression language, custom tags

28

JSP/servlet training: <http://www.coreservlets.com>

© 2003 Marty Hall



## Questions?

JSP and Servlet Training Courses: <http://courses.coreservlets.com>  
JSP and Servlet Books from Sun Press: <http://www.coreservlets.com>