Content

- Need and benefits for JSP
- Comparing JSP to other technologies
- JSP lifecycle
- Dynamic code and good JSP design
- JSP expressions: `<%= ... %>`
- Servlets vs. JSP pages for similar tasks
- Predefined variables: out, session, ...
- JSP scriptlets: `<% ... %>`
- JSP declarations: `<%! ... %>`
- Comparison of expressions, scriptlets, and declarations
- Page directive: `<%@ ... %>`
  - Designating which classes are imported
  - The content type
  - Including files in a JSP (at translation time and at execution time).

Most of the slides were made available by www.coreservlets.com
The Need for JSP

- With servlets, it is **easy** to
  - Read form data
  - Read HTTP request headers
  - Set HTTP status codes and response headers
  - Use cookies and session tracking
  - Share data among servlets (*we shall see*)
  - Remember data between requests

- But, it is a **pain** to:
  - *use those println statements to generate HTML*
  - *maintain* that HTML.
The JSP Framework

- Use regular HTML for most of page
- Entire JSP page gets translated into a servlet (once), and servlet is what actually gets invoked (for each request)
- Mark servlet code with special tags

```html
<!DOCTYPE ...>
<html>
<head>
<title>Order Confirmation</title>
<link rel=stylesheet
type="text/css">
<body>
<h2>Order Confirmation</h2>
Thanks for ordering
<i><%= request.getParameter("title") %></i>!
</body></html>
```
Benefits of JSP

- Although JSP technically can't do anything servlets can't do, JSP makes it easier to:
  - **Write** HTML
  - Read and maintain the HTML

- JSP makes it possible to:
  - **Use** standard HTML **tools** such as Macromedia DreamWeaver or Adobe GoLive
  - Have **different members** of your team do the HTML layout than do the Java programming

- JSP encourages you to
  - **Separate** the (Java) code that creates the content from the (HTML) code that presents it.
Advantages of JSP Over Competing Techs

- Versus client-side JavaScript (in browser)
  - Capabilities mostly do not overlap with JSP, but
    - You control server, not client
    - Richer language

- Versus pure servlets
  - More convenient to create HTML
  - Can use standard tools (e.g., DreamWeaver)
  - Divide and conquer
  - JSP programmers still need to know servlet programming

- Versus static HTML
  - Dynamic features
  - Adding dynamic features no longer "all or nothing" decision.
Setting Up Your Environment

- Set your CLASSPATH. Not.
- Compile your code. Not.
- Use packages to avoid name conflicts. Not.
- Put JSP page in special directory. Not.
  - `install_dir\webapps\ROOT\` (HTML and JSP -- Tomcat)
- Use special URLs to invoke JSP page. Not.
  - Use same URLs as for HTML pages (except for file extensions)

Caveats

- Previous rules about CLASSPATH, install dirs, etc., 
  **still apply to regular Java classes** used by a JSP page.
Example

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>JSP Expressions</title>
<meta name="keywords" content="JSP,expressions,JavaServer Pages">
<meta name="description" content="A quick example of JSP expressions.">
<link rel="stylesheet" href="css/JSP-Styles.css" type="text/css">
</head>
</html>
<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>
Example: Result

- If the context of your application is coresjsp and the location was
  - C:\jakarta-tomcat-xx\webapps\ROOT\coresjsp\Expressions.jsp

- URL would be
  - http://localhost/coresjsp/Expressions.jsp

- Your jsp sources are written in myprj/web directory, and Netbeans copies them in the myprj/build/web directory
Request and Translation Times

- **What happens at page translation time?**
  - JSP constructs get translated into servlet code

- **What happens at request time?**
  - Servlet code gets executed. *No* interpretation of JSP occurs at request time. The original JSP page is totally ignored at request time; only the servlet that resulted from it is used.

- **When does page translation occur?**
  - Typically, **the first time JSP page is accessed after it is modified** - this should never happen to real user (developers should test all JSP pages they install)
  - Page translation **does not occur for each request**.
JSP/Servlet Correspondence

- **Original JSP**

  ```
  <H1>A Random Number</H1>
  <%= Math.random() %>
  ```

- **Representative resulting servlet code**

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

<table>
<thead>
<tr>
<th>Page first written</th>
<th>Request #1</th>
<th>Req. #2</th>
<th>Request #3</th>
<th>Req. #4</th>
<th>Request #5</th>
<th>Request #6</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSP page translated into servlet</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Servlet compiled</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Servlet instantiated and loaded into server's memory</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>init (or equivalent) called</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>doGet (or equivalent) called</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
JSP/Servlets in the Real World: Airlines

- Delta Airlines
- United Airlines
- AirTran
- American Airlines
- British Airways
- KLM
- Air China
- Saudi Arabian Airlines
- Iceland Air
JSP/Servlets in the Real World: Travel Sites

- Travelocity.com
- Orbitz.com
- HotWire.com
- Hotels.com
- CheapTickets.com
- National Car Rental
- Avis Car Rental
- Enterprise Car Rental
- Hertz Car Rental
JSP/Servlets in the Real World: Financial Services

- American Century
- Vanguard
- Fidelity
- NY Stock Exchange
- First USA Bank
- Royal Bank of Scotland
- Banco Popular de Puerto Rico
- Bank of America
- China Construction Bank
JSP/Servlets in the Real World: Retail

- Sears.com
- Walmart.com
- HomeDepot.com
- SamsClub.com
- Macys.com
- llbean.com
- Kohls.com
- Ikea.com
- REI.com
- Longaberger.com
- Nike.com
JSP/Servlets in the Real World: Search/Portals

- Parts of Google
- All of Ebay
- netscape.com
- excite.com
- dice.com
- hi5
- Paypal
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 (IDE), not an HTML environment
  - **Debugging.** If you have syntax errors, you see them immediately at compile time
  - **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 JSP 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 - results in something like `out.println(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 1) `evaluated`, 2) `converted to String`, and 3) `placed into HTML page` at the place it occurred in JSP page

- 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 - use XML for `entire` page if you use `jsp:expression`. 
...<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>
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 – see later)

- **application**
  - The `ServletContext` (for sharing data) as obtained via `getServletContext()`.
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);
  %>
  ```

- **XML-compatible syntax**
  - `<jsp:scriptlet>Java Code</jsp:scriptlet>`
JSP/Servlet Correspondence

- **Original JSP**
  
  ```html
  <H2>foo</H2>
  <%= bar() %>
  <% baz(); %>
  ```

- **Representative resulting servlet code**

  ```java
  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();
    ...
  }
  ```
Suppose you want to let end users customize the background color of a page.

What is wrong with the following code?

```html
<BODY BGCOLOR="\"<%= request.getParameter("bgColor") %>\">
```

What happens if the parameter is not provided in the request? **Badcall goodcall**
JSP Scriptlets: Example

```html
<!DOCTYPE ...>
<html>
<head>
  <title>Color Testing</title>
</head>
<body bgcolor=""><%= bgColor %">
  <h2 align="center">Testing a Background of "<%= bgColor %></h2>
</body></html>
```
Testing a Background of "WHITE"

Testing a Background of "green"
Scriptlets: Make Parts of JSP File Conditional

- 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>`
Original JSP

```jsp
<H1>Some Heading</H1>
<br />
<%!  
    private String randomHeading() {
        return("<H2>" + Math.random() + "</H2>"');
    }
%>

<%= randomHeading() %>
```

Alternative: make randomHeading a static method in a separate Java class.
Possible resulting servlet code

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

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

- The predefined variables (request, response, out, session, etc.) are **local to the _jspService method**
- They are not available to methods defined by JSP declarations or to methods in helper classes

Solution: pass them as arguments, e.g.

```jsp
<%!  
private void someMethod(HttpSession s) {  
    doSomethingWith(s);  
}  
%>
<someMethod(session);%>
```

Same issue with separate static methods

- And they are usually preferred over JSP declarations.
Expressions, Scriptlets and Declarations

- Task 1: Output a bulleted list of five random integers 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.
package coreservlets; // Always use packages!!

/** Simple utility to generate random integers. */

class RanUtilities {

    /** A random int from 1 to range (inclusive). */

    public static int randomInt(int range) {
        return(1 + ((int)(Math.random() * range))); // cast to int truncates
    }

    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)

```html
<!DOCTYPE ...>
<html>
<head>
<title>Random Numbers</title>
<link rel=stylesheet
    href="css/JSP-Styles.css"
    type="text/css">
</head>
<body>
<h1>Random Numbers</h1>
<ul>
    <li><%= coreservlets.RanUtilities.randomInt(10) %></li>
    <li><%= coreservlets.RanUtilities.randomInt(10) %></li>
    <li><%= coreservlets.RanUtilities.randomInt(10) %></li>
    <li><%= coreservlets.RanUtilities.randomInt(10) %></li>
    <li><%= coreservlets.RanUtilities.randomInt(10) %></li>
</ul>
</body>
</html>
```
Task 1: JSP Expressions (Result)

Random Numbers

- 8
- 1
- 4
- 8
- 9
Task 2: JSP Scriptlets (Code: Version 1)

```html
<!DOCTYPE ...>
<html>
<head>
<title>Random List (Version 1)</title>
<link rel=stylesheet href="css/JSP-Styles.css" type="text/css">
</head>
<body>
<h1>Random List (Version 1)</h1>
<ul>
<% int numEntries = coreservlets.RanUtilities.randomInt(10);
for(int i=0; i<numEntries; i++) {
    out.println("<li>" + coreservlets.RanUtilities.randomInt(10));
}
%>
</ul>
</body>
</html>
```
Task 2: JSP Scriptlets (Result: Version 1)

Random List (Version 1)

- 4
- 8
- 10
- 2
- 5
- 9
- 1
- 4
- 5
- 1
Task 2: JSP Scriptlets (Code: Version 2)

```html
<!DOCTYPE ...>
<html>
<head>
<title>Random List (Version 2)</title>
<link rel=stylesheet href="css/JSP-Styles.css" type="text/css">
</head>
<body>
<h1>Random List (Version 2)</h1>
<ul>
  <% int numEntries = coreservlets.RanUtilities.randomInt(10);
      for(int i=0; i<numEntries; i++) {
        %>
  <li><%= coreservlets.RanUtilities.randomInt(10) %></li>
  <% } %>
</ul>
</body></html>
```
Task 2: JSP Scriptlets (Result: Version 2)

Random List (Version 2)

- 10
- 5
- 1
- 4
- 1
- 3
- 8
- 1
Task 3: JSP Declarations

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

<H1>Semi-Random Number:<BR><%= randomNum %></H1>
</BODY>
</HTML>
Task 3: JSP Declarations (Result)

Semi-Random Number: 10
Purpose of the page **Directive**

- Give high-level information about the servlet that will result from the JSP page
- Can control
  - Which classes are **imported**
  - What class the servlet **extends**
  - What MIME type is **generated**
  - How **multithreading** is handled
  - If the servlet participates in **sessions**
  - The size and behavior of the **output buffer**
  - What page handles unexpected **errors**
The import Attribute

- **Format**
  - `<%@ page import="package.class" %>`
  - `<%@ page import="package.class1,...,package.classN" %>`

- **Purpose**
  - Generate import statements at top of servlet definition

- **Notes**
  - Although JSP pages can be almost anywhere on server, classes used by JSP pages must be in normal servlet dirs
  - E.g.:
    - `.../WEB-INF/classes` or `.../WEB-INF/classes/directoryMatchingPackage`
    - *Always use packages for utilities that will be used by JSP!*
The import Attribute: Example (Code)

...<H2>The import Attribute</H2>
<%@ page import="java.util.*,coreservlets.*" %>
<%!
    private String randomID() {
        int num = (int)(Math.random()*10000000.0);
        return("id" + num);
    }

    private final String NO_VALUE = "<I>No Value</I>";
%>
<%
    String oldID = CookieUtilities.getCookieValue(request, "userID", NO_VALUE);
    if (oldID.equals(NO_VALUE)) {
        String newID = randomID();
        Cookie cookie = new LongLivedCookie("userID", newID);
        response.addCookie(cookie);
    }
%>
This page was accessed on <%= new Date() %> with a userID cookie of <%= oldID %>.
</BODY></HTML>
The `contentType` and `pageEncoding` Attributes

- **Format**
  - `<%@ page contentType="MIME-Type" %>`
  - `<%@ page contentType="MIME-Type; charset=Character-Set" %>`
  - `<%@ page pageEncoding="Character-Set" %>`

- **Purpose**
  - Specify the MIME type of the page generated by the servlet that results from the JSP page

- **Notes**
  - `contentType` value cannot be computed at request time
  - See section on response headers for table of the most common MIME types.
The session Attribute

- Format
  - `<%@ page session="true" %>`  `<%-- Default --%>`
  - `<%@ page session="false" %>`

- Purpose
  - To designate that page not be part of a session

- Notes
  - By **default, it is part of a session**
  - Saves memory on server if you have a high-traffic site
  - *All* related pages have to do this for it to be useful
Including Files at Request Time: jsp:include

Format
- `<jsp:include page="Relative address" />`

Purpose
- To **reuse** JSP, HTML, or plain text content
- To permit updates to the included content **without changing the main** JSP page(s)

Notes
- Included JSP **cannot affect main page**: only output of included JSP page is used
- Relative URLs that starts with slashes are interpreted relative to the Web app, not relative to the server root
- If it does not start with slash it is relative to the position of the JSP that use the `jsp:include` action
- You are permitted to include files from WEB-INF.
jsp:include Example: Main Page

...
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
    What's New at JspNews.com</TH></TR>
</TABLE>
<P>
Here is a summary of our three
most recent news stories:
<OL>
  <LI><jsp:include page="/WEB-INF/includes/Item1.jsp" /></LI>
  <LI><jsp:include page="WEB-INF/includes/Item2.jsp" /></LI>
  <LI><jsp:include page="/WEB-INF/includes/Item3.jsp" /></LI>
</OL>
</BODY></HTML>

Relative to the main JSP: there should be
a subfolder WEB-INF in the directory where
the main JSP is.

Relative to the base of
the web application –
not the server root.
<B>Bill Gates acts humble.</B> In a startling and unexpected development, Microsoft big wig Bill Gates put on an open act of humility yesterday.

<A HREF="http://www.microsoft.com/Never.html">More details...</A>

- Note that the page is <i>not</i> a complete HTML document; it has only the tags appropriate to the place that it will be inserted
What's New at JspNews.com

Here is a summary of our three most recent news stories:

1. **Bill Gates acts humble.** In a startling and unexpected development, Microsoft big wig Bill Gates put on an open act of humility yesterday. [More details...]
2. **Scott McNealy acts serious.** In an unexpected twist, wisecracking Sun head Scott McNealy was sober and subdued at yesterday’s meeting. [More details...]
3. **Larry Ellison acts conciliatory.** Catching his competitors off guard yesterday, Oracle prez Larry Ellison referred to his rivals in friendly and respectful terms. [More details...]
The jsp:param Element: Augmenting Request Parameters

- Code
  ```jsp
  <jsp:include page="/fragments/StandardHeading.jsp">
    <jsp:param name="bgColor" value="YELLOW" />
  </jsp:include>
  ```
- URL
  ```http://host/path/MainPage.jsp?fgColor=RED```
- Main page
  - fgColor: RED
  - bgColor: null
  - Regardless of whether you check before or after inclusion
- Included page
  - fgColor: RED
  - bgColor: YELLOW

This parameter is passed to the included page

The included page normally sees the same parameters as the main page
Including Files at Page **Translation Time**

- **Format**
  - `<%@ include file="Relative address" %>`

- **Purpose**
  - To reuse JSP content in multiple pages, *where JSP content affects main page*

- **Notes**
  - Servers are not required to detect changes to the included file, and in practice they don't
  - Thus, you need to change the JSP files whenever the included file changes or to rebuild the project
  - You can use OS-specific mechanisms such as the Unix "touch" command.
Include Directive Example: Reusable Footers

<%@ page import="java.util.Date" %>
<%-- The following become fields in each servlet that
    results from a JSP page that includes this file. --%>
<%!
private int accessCount = 0;
private Date accessDate = new Date();
private String accessHost = "<I>No previous access</I>";
%
<P>
<HR>
This page © 2003
This page has been accessed <%= ++accessCount %>
times since server reboot. It was most recently
accessed from
<%= accessHost %> at <%= accessDate %>. //the previous access
<% accessHost = request.getRemoteHost(); %>
<% accessDate = new Date(); %>

This is the included page
Some Random Page

Information about our products and services.
Blah, blah, blah.
Yadda, yadda, yadda.
<%@ include file="/WEB-INF/includes/ContactSection.jsp" %>

SomeRandomPage.jsp
Reusing Footers: Result

Some Random Page

Information about our products and services.

Blah, blah, blah.

Yadda, yadda, yadda.

This page © 2003 my-company.com. This page has been accessed 2 times since server reboot. It was most recently accessed from 127.0.0.1 at Wed Apr 16 19:03:44 CEST 2008.
### jsp:include vs. `<%@ include ...%>`

<table>
<thead>
<tr>
<th></th>
<th>jsp:include</th>
<th><code>&lt;%@ include ...%&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic syntax</strong></td>
<td><code>&lt;jsp:include page=&quot;...&quot; /&gt;</code></td>
<td><code>&lt;%@ include file=&quot;...&quot; %&gt;</code></td>
</tr>
<tr>
<td><strong>When inclusion occurs</strong></td>
<td>Request time</td>
<td>Page translation time</td>
</tr>
<tr>
<td><strong>What is included</strong></td>
<td>Output of page</td>
<td>Contents of file</td>
</tr>
<tr>
<td><strong>Number of resulting servlets</strong></td>
<td>Two</td>
<td>One</td>
</tr>
<tr>
<td><strong>Can included page set response headers that affect the main page?</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Can included page define fields or methods that main page uses?</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Does main page need to be updated when included page changes?</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Which Should You Use?

- Use **jsp:include** whenever possible
  - Changes to included page do not require any manual updates
  - Speed difference between jsp:include and the include directive (@include) is insignificant

- The include directive (<%@ include ...%>) has additional power, e.g.>
  - Included page
    - <%! int accessCount = 0; %>
  - Main page
    - <%= accessCount++ %>