Network Computing Lecture 15

Professor Louise E. Moser

Spring 2012

Java Server Pages

- Like Java Servlets, **Java Server Pages (JSPs)** let you mix static and dynamic HTML
  - JSPs can’t do anything more than Java Servlets can do
- JSPs are arguably better than Servlets alone, because JSPs allow you to write and modify HTML without so many print statements
  - However, JSPs should not contain large blocks of code
- JSPs are better than Service Side Includes (SSIs), because JSPs let you use Servlets to generate the dynamic part and to create "real" programs that use data from forms, databases, etc.

Java Servlets vs. Java Server Pages

- Java Servlets and JSPs are similar in that they both
  - Perform server-side processing
  - Provide the same results to the end user
  - Allow you to create dynamic Web pages
- Java Servlets and JSPs are different in that
  - **Java Servlets – HTML code in Java code**
    - HTML code is inaccessible to the GUI designer
    - HTML code and Java code are accessible to the application programmer
  - **Java Server Pages - Java code in HTML code**
    - HTML code is accessible to the GUI designer
    - Java code is accessible to the application programmer

Example: Java Server Pages

- **Java code in HTML code**

<html>

<body>

<b>I'm HTML code. <\b><\br>

<% out.println("I'm Java code.") %>

<\body>

<\html>

Using Java Server Pages

- To use JSPs, you write what looks like a regular HTML file and enclose the Java code for the dynamic parts in special tags, most of which start with ***<%*** and end with ***%>***
- You give your file a ***.jsp*** extension and install it where you would install an HTML file
- The first time it is accessed, the JSP is converted into a Servlet with the static HTML printed to the output stream associated with the Servlet's *service()* method
- With each subsequent request, the Servlet code is executed (and the JSP is ignored)

JSP Constructs

- The main JSP constructs are
  - **JSP Scripting Elements**
  - **JSP Directives**
  - **JSP Beans**

Kinds of JSP Scripting Elements

- **JSP Expressions** use the format ***<%= ... %>***
  - Evaluated and inserted into the Servlet's *_jspService()* method (called by the *service()* method), resulting in something like *out.println(expression)*
- **JSP Scriptlets** use the format ***<% ... %>***
  - Inserted into the Servlet's *_jspService()* method (called by the *service()* method)
- **JSP Declarations** use the format ***<%! ... %>***
  - Inserted into the body of the *Servlet* class, outside existing methods

JSP Expressions

- Format <%= ... %>
- Result
  - Expression is evaluated, converted to a String, placed into the HTML page where it occurs in the JSP
  - Expression is inserted into the Servlet's ***_jspService()*** **method inside** ***out.println()***
- Examples
  - *Current time: <%= new java.util.Date() %>*
  - *Host name: <%= request.getRemoteHost() %>*

JSP Expression / Servlet Correspondence

- **JSP Expression**

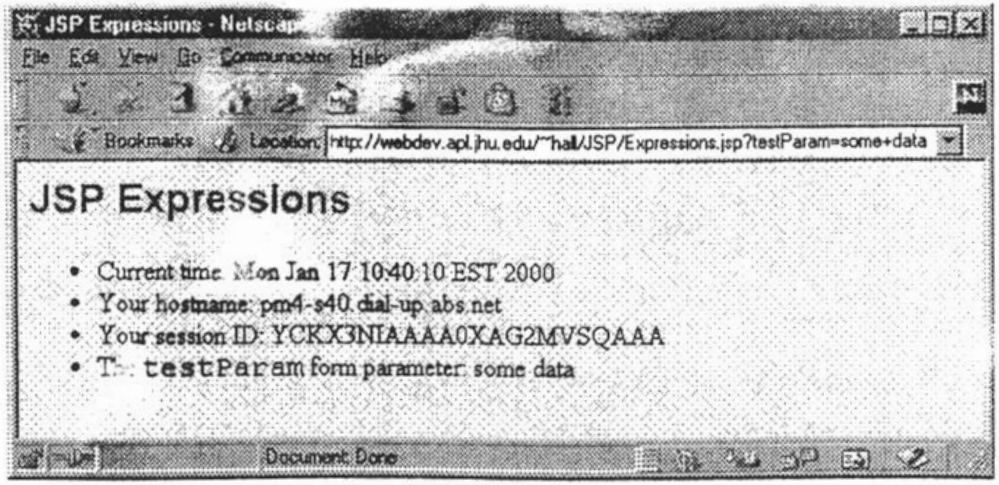
  <%= Math.random() %>

- **Servlet**

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

Example: JSP Expressions

```
<BODY>
<H2> JSP Expressions </H2>
<UL>
  <L1>Current time: <%= new java.util.Date() %>
  <L1>Your host name: <%= req.getRemoteHost() %>
  <L1>Your session id: <%= session.getId() %>
  <L1>The <CODE>testparam</CODE> form parameter:
     <%= req.getParameter("testparam") %>
</UL>
</BODY>
```



JSP Scriptlets

- Format *<% ... %>*
- Result
  - Java code is inserted **verbatim** into the Servlet's ***_jspService()* method**
- Examples
  - *<% String.queryData = req.getQueryString();*

    out.println("Attached GET data: " + queryData); %>
  - *<% res.setContentType("text/plain"); %>*

JSP Scriptlet / Servlet Correspondence

- JSP Expression and **JSP Scriptlet**

  <%= foo() %> // JSP Expression

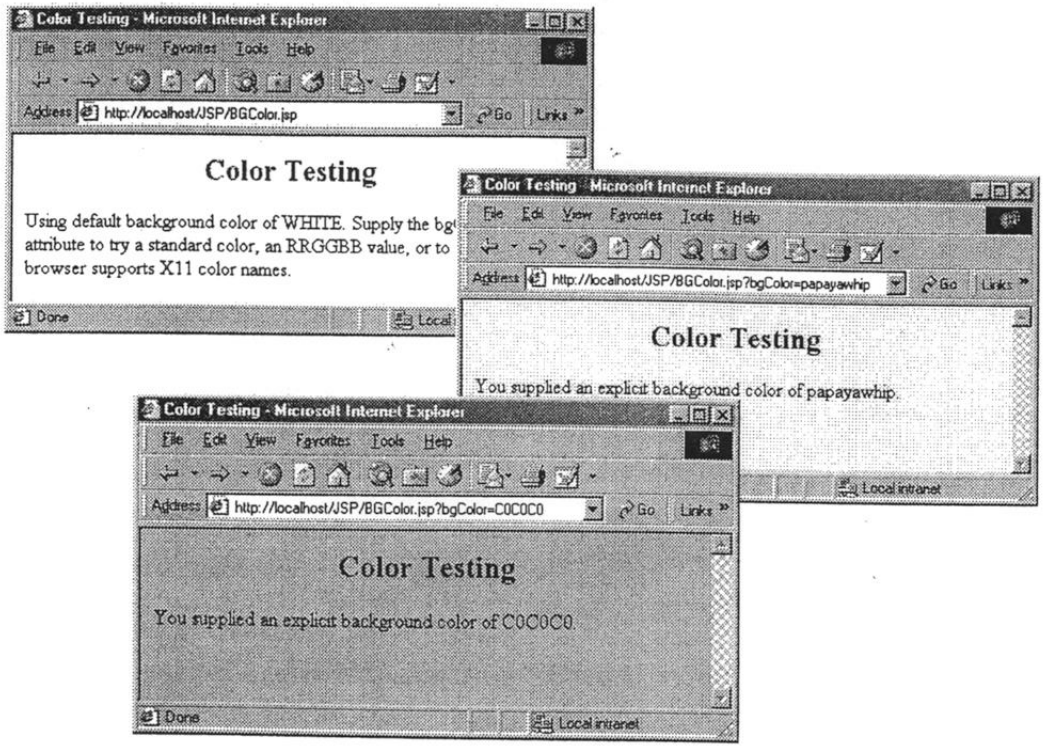
  <% bar(); %> // JSP Scriptlet

- **Servlet**

```
public void _jspService(HttpServletRequest req,
                        HttpServletResponse res)
        throws ServletException, IOException {
  req.SetContentType("text/html");
  HTTPSession session = req.getSession(true);
  JspWriter out = res.getWriter();
  out.println(foo());
  bar(); ...
}
```

Example: JSP Scriptlets

```
<!DOCTYPE HTML PUBLIC
        "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Color Testing</TITLE>
</HEAD>
<%                          // JSP Scriptlet
String bgColor =
    req.getParameter("bgColor");
boolean hasExplicitColor;
if (bfColor != null) {
  hasExplicitColor = true;
}
else {
  hasExplicitColor = false;
  bgColor = "WHITE";
}
%>
<BODY BGCOLOR = "<%= bgColor %>">   // JSP Expression
```



JSP Declarations

- Format *<%! ... %>*
- Result
  - Code is inserted **verbatim** into the Servlet's **class** definition, **outside existing methods** such as the *_jspService()* method
- Examples
  - *<%! private int someField = 5; %>*
  - *<%! private void someMethod(...); %>*
  - *<%! res.setContentType("text/plain"); %>*

JSP Declaration / Servlet Correspondence

- **JSP Declaration** and JSP Expression

```
<H1>Some Heading<\H1>
<%!                                          // JSP Declaration
  private String randomHeading() {
    return("<H2>" + Math.random() + "</H2>");
  }
%>
<%= randomHeading() %>                       // JSP Expression
```

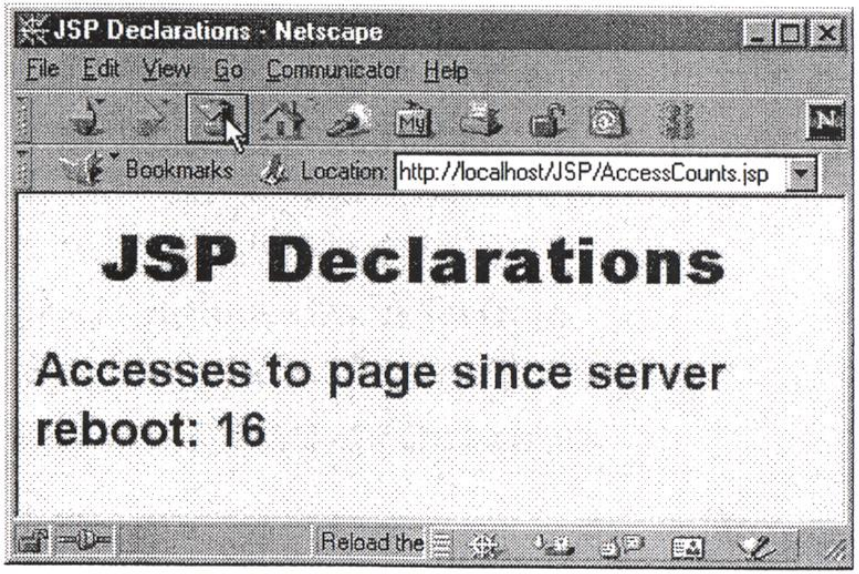
JSP Declaration / Servlet Correspondence (cont)

- **Servlet**

```
public class xxx implements HttpJspPage {

    private String randomHeading() {
        return("<H2>" + Math.random() + "</H2>");
    }

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

Example: JSP Declarations

```
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD HTML 4.0 Transitional//EN">
<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; %>                    // JSP Declaration
<H2>Accesses to page since server reboot:
<%= ++accessCount %></H2>                              // JSP Expression

</BODY>
</HTML>
```



JSP Directives

- Give high-level information about the Servlet that will result from a JSP
- Can be used to control
  - Which classes are imported
  - Which class the Servlet extends
  - Which MIME type is generated
  - How multithreading is handled
  - Whether the Servlet participates in sessions
  - Which page handles unexpected errors
  - Size of the output buffer

Kinds of JSP Directives

- **page** – for defining page attributes
- **include** – for including external reusable components into the current page, e.g., blocks of template text, other JSP scripts, applets, etc.
- **taglib** – for including taglib collections into the current page (we will not cover taglib)

JSP Directives have the format

<%@ directive attribute = "value" %>

which may have multiple attribute-value pairs separated by commas

JSP Page Directive

- **page** has the format

  *<%@ **page import** = "java.util.*"%>* or

  *<**jsp:page import** ="java.util.*">*

- **page** has the following **page attributes**
  - ***import*** – similar to Java import statements
  - ***contentType*** – for setting the page's *contentType*
  - ***isThreadSafe*** – default is true, assumes you handle synchronization
  - ***session*** – default is true, sets session participation
  - ***buffer*** – default is set by the server but can be overridden
  - ***autoflush*** – sets *autoflush* to true or false
  - ***extends*** – indicates the superclass (use with care)
  - ***info*** – for setting a String retrieved by *getServletInfo()*
  - ***errorPage*** – sets the Error page
  - ***isErrorPage*** – true or false for current page to handle errors

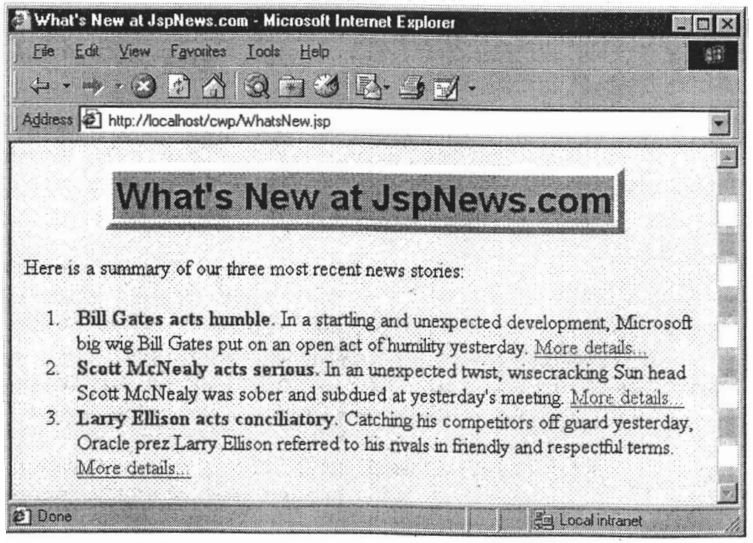
JSP Page Directive – Import Attribute

- Format
  - *<%@ page **import** ="package.class"%>*
  - *<%@ page **import** ="package.class1,…,package.classN"%>*
- Purpose
  - To generate *import* statements at the top of a Servlet
- Note
  - JSPs can be almost anywhere on the server, but classes that JSPs use must be in regular Servlet directories
  - For the Tomcat servlet engine, the directory structure is *install_dir/webapps/ROOT/WEB-INF/classes* or *…/ROOT/WEB-INF/classes/directorMatchingPackage*

JSP Include Directive

- Format
  - ***<jsp:include*** *page="URL_of_page">*
    includes the page at request time (cannot include JSP code)
  - *<%@ include file="URL_of_file"%>*
    includes the file (cannot include JSP code)
  - *<jsp:plugin type="applet" width="120" height="70">*
    includes the applet
- Purpose
  - To permit updates to the included content without changing the main JSPs

Example: JSP Include Directive




```
...
<BODY>
<TABLE BORDER=5 ALIGH="CENTER">
  <TR><TH CLASS="TITLE">
    What's New at Jsp.News.com
</TABLE>
<P>
Here is a summary of our three most recent news stories:
<OL>
  <LI><jsp:include  page="news/Item1.html"  flush=true" />
  <LI><jsp:include  page="news/Item2.html"  flush=true" />
  <LI><jsp:include  page="news/Item3.html"  flush=true" />
</OL>
</BODY>
</HTML>
```

JSP Beans

- **JSP Beans** are regular Java classes that follow certain conventions, including
  - A constructor with no arguments
  - No public instance variables
  - Use of *getXxx()* and *setXxx()* methods
    - Boolean properties use *isXxx()*, instead of *getXxx()*
    - If a class has a method *getTitle()* that returns a *String*, the class is said to have a String property named Title

JSP useBean

- Format

  <jsp:useBean *id="name" class="package.Class" />*

- Purpose
  - To allow instantiation of classes without explicit Java syntax
- Note
  - *<jsp:useBean id="book1" class="cwp.Book" />*
    is equivalent to the JSP Scriptlet

    <% cwp.Book book1 = new cwp.Book(); %>
  - But, *useBean* has two additional features:
    - Simplifies the setting of fields based on request parameters
    - Makes it easier to share Beans

Setting Bean Properties

- Format
  - ***<jsp:setProperty*** *name="name" property="property" value="value" />*
- Purpose
  - To allow the setting of Bean properties, using *setXxx()* methods, without explicit Java code
- Note
  - *<jsp:setProperty name="book1" property="title" value="Core Servlets and JSP" />*

    is equivalent to the JSP Scriptlet

    <% book1.setTitle("Core Servlets and JSP") %>

Getting Bean Properties

- Format
  - ***<jsp:getProperty*** *name="name" property="property" />*
- Purpose
  - To allow the getting of Bean properties, using *getXxx()* methods, without explicit Java code
- Note
  - *<jsp:getProperty name="book1" property="title" />*

    is equivalent to the JSP Expression

    <%= book1.getTitle() %>

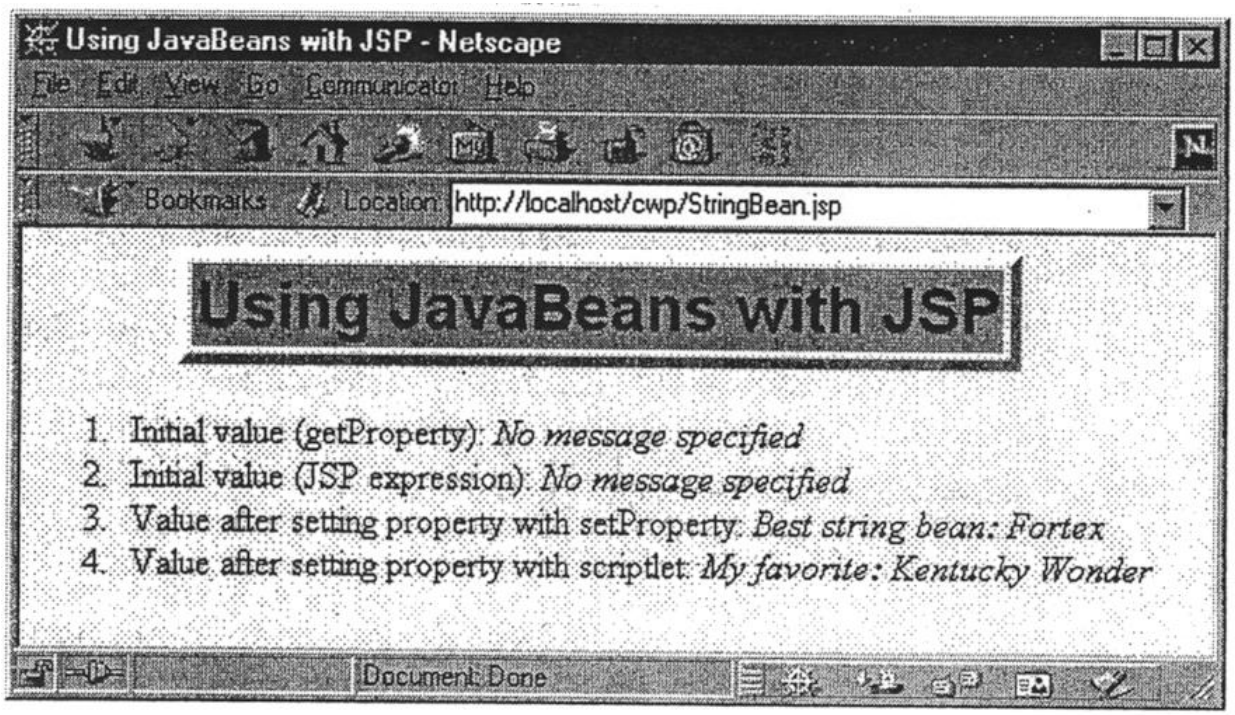
Example: JSP StringBean

```
package cwp;

public class StringBean {
  private String message = "No message specified";

  public void setMessage(String message) {
    this.message = message;
  }

  public String getMessage() {
    return(message);
  }
}
```

JSP Page That Uses StringBean

```
<jsp:useBean id="stringBean" class="cwp.StringBean" />
<OL>
<LI>Initial value (getProperty);
    <I><jsp.getProperty name="stringBean"  property="message" /></I>

<LI>Initial value (JSP expression);
    <I><%= stringBean.getMessage() %></I>

<LI><jsp:setProperty name="stringBean"  property="message"
          value="Best string bean: Fortex" />
    Value after setting property with setProperty:
    <I><jsp:getProperty name="stringBean"  property="message" /></I>

<L1><% stringBean.setMessage("My favorite: Kentucky Wonder"); %>
    Value after setting property with scriptlet:
    <I><%= stringBean.getMessage() %></I>
</OL>
```

Result of JSP Page That Uses StringBean



Bean Properties and Request Parameters

- If property is a *String*, you can use it as follows

  <jsp:setProperty name = ... property = ...
  *value='<%= **request.getParameter("...")**%>' />*

- You can use ***** for *property* to indicate that
  - *value* should come from the request parameter whose name matches the property name
- You can use the *param* attribute to indicate that
  - *value* should come from the request parameter specified
- In the above two cases, simple type conversion is performed

Scope Attribute

- You can use the ***scope*** attribute to share Beans and to specify where a Bean is stored

  *<jsp:useBean id = "..." class = "..." **scope** = "..." />*

- The Bean is still bound to a local variable in the *_jspService()* method
- This mechanism allows multiple JSPs or Servlets to share Beans

Values of Scope Attribute

- In JSP, a Bean can be stored in one of four locations:
  - **page** - Bean is stored in the **Page context** for the duration of the request; methods in the same Servlet can access the Bean (which is the default)
  - **application** - Bean is stored in the ***ServletContext*** **object** (which is shared by all Servlets in the same application)
  - **session** - Bean is stored in the ***HttpSession*** **object** associated with the current request, where it can be accessed using *setAttribute()* and *getAttribute()*
  - **request** - Bean is stored in the ***ServletRequest*** **object** associated with the current request, where it can be accessed using *getAttribute()*
- The four locations above are the **values** of the *scope* attribute
- In all of the above four cases, the Bean is still bound to a local variable in the *_jspService()* method