CSE 510
Web Data Engineering

Java Server Pages (JSPs)

cse@buffalo

Java Server Pages: Embedding Java Code in Static Content

Why JSPs?

Need to separate

- the business logic implementation
  - done by web developer
- from implementing the look-and-feel
  - done by web designer

The Key Idea Behind JSPs

- HTML page with embedded Java code (in the form of JSP elements)

```
<html>
  <head>
    <title>Date JSP (Textbook Listing 5.1)</title>
  </head>
  <body>
    <big>
      Today's date is <%= new java.util.Date() %>
    </big>
  </body>
</html>
```

Deploying JSPs

- JSP file has .jsp suffix
- Store JSP file (in text) in app directory
- Invoke as

  `http://<host>/<web-app>/<file>.jsp`

Compilation

At first access of JSP:

- Jasper translator generates Java servlet code
  - Loads in `<CATALINA_HOME>/work/Catalina/<host>/<web app>`
- Jasper compiler generates Java Servlet class file
  - Loads in same directory

date_jsp.java

```
package org.apache.jsp.jsp.dates;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;

public final class date_jsp extends HttpJspBase
    implements JspSourceDependent {
  ...
  /* Similar to doGet() */
  public void _jspService(HttpServletRequest request,
                          HttpServletResponse response)
    throws java.io.IOException, ServletException {
```

Implicitly Declared Objects

- You may use the following objects in the Java code of your JSP
- **`request`**: well-known HttpServletRequest object
  - transfers parameters
- **`response`**: still important for writing non-body fields of HTTP response
- **`session`**: maintain parameters accessed by all steps of a session
  - Very important, we'll come back to it
- **`application`**: maintain parameters accessed by all JSPs of a web application

date_jsp.java

```
/* Implicit objects defined next */
PageContext pageContext = null;
HttpSession session = null;
ServletContext application = null;
ServletConfig config = null;
JspWriter out = null;
...

try {
  response.setContentType("text/html");
  pageContext = _jspxFactory.getPageContext(...);
  application = pageContext.getServletContext();
  config = pageContext.getServletConfig();
  session = pageContext.getSession();
  out = pageContext.getOut();
```

date_jsp.java

```
        /* Output of HTML code of JSP */
        out.write("<html>\r\n");
        out.write("  <head>\r\n");
        out.write("    <title>Date JSP (...)</title>\r\n");
        out.write("  </head>\r\n");
        out.write("  <body>\r\n");
        out.write("    <big>\r\n");
        out.write("      Today's date is ");
        out.print(new java.util.Date());
        out.write("\r\n");
        out.write("    </big>\r\n");
        ...
      }
    }
}
```

JSP Elements

- JSP Directives
  - Includes, imports, etc
- JSP Scripting Elements
  - Java code, expressions, variable declarations
- JSP Action Elements
  - Beans, tag libraries, etc
  - We'll discuss later

JSP Directives

- `<%@ <directive> { <attr>="<value>" }* %>`
- `<%@ include file="<file>.html" %>`
- `<%@ page import="<package name>" %>`

```
<html>
  <head><title>dateWithImport.jsp</title></head>
  <body><big>
    <%@ page import="java.util.*" %>
    Today's date is <%= new Date() %>
  </big></body>
</html>
```

- Recall: some packages automatically imported
  - More on page 86 of textbook

JSP Scripting Elements

- **Expressions**
  - `<%= <java_expression> %>`
  - Example: `<%= i+1 %>`
  - Evaluates expression, casts into String, places in output
- **Scriptlets**
  - `<% <java_code> %>`
  - Example:

```
<% int times;
   times = 3; %>
```

  - Code inlined in `_jspService()`
- Scriptlets have semicolons, expressions don't

Two Kinds of Declarations in JSP Scripting Elements

- **Local variables** simply part of Scriptlets
  - See code of `<CATALINA_HOME>/work/Catalina/localhost/multiplierJSP/.../multiplyJSP_jsp.java`
- **Class variables** (not in `_jspService()`)
  - `<%! int times; %>`
  - If we have in JSP Sciptlet

    `<% times += 1; %>`

    It will be incremented every time JSP is called, from same or different sessions
  - See `multiplyJSPWithClassVariable.jsp`

Deployment Revisited

- All uses of servlet names also apply to JSPs
- You may not want someone to know that you have used (a particular) .jsp to implement your page and you want to use URL mapping to hide name
- Declaration of name almost same with servlets

```
<servlet>
    <servlet-name>multiplier</servlet-name>
    <jsp-file>multiplierJSP.jsp</jsp-file>
</servlet>
```

Scope Issues in JSPs

Interaction Across HTTP Calls: Four Scoping Levels

- **Application**
  - Servlet initialization parameters
  - Exchange information across calls of same application (same app context)
- **Session** (most important)
  - Session: Set of calls from same browser process
  - Multiple browser windows may be in same process
  - Exchange information within session
  - Non-obvious how given HTTP statelessness
- **Request**
  - Exchange information across HTTP calls
- **Page** (almost useless)

Application Level Attributes

- `application` is an implicit variable within JSPs
- In servlet, it is obtained by

```
application=getServletContext();
```

- Exchange attribute info across all calls
  - `application.getAttribute(name);`
  - `application.setAttribute(name, object);`
  - Can do the same with class variables, or with a database, at a higher cost but with persistence
  - No synchronization and ACID properties

Counter Example

```
<html>
 <head><title>Counter Web Application</title></head>
 <body>
  <% Integer i =
       (Integer)(application.getAttribute("counter"));
     if (i == null) { i = new Integer(0); }
     else { i = new Integer(i.intValue() + 1); }
     application.setAttribute("counter", i);
  %>
  Your application has visited <%= i %> times this page
 </body>
</html>
```

Getting Web Application Initialization Parameters

- Define application initialization parameters in the deployment descriptor

```
<web-app>
  <!-- other stuff we've seen... -->
  <context-param>
    <param-name>developer</param-name>
    <param-value>mpetropo@buffalo.edu</param-value>
  </context-param>
  <!-- other stuff we've seen... -->
</web-app>
```

- `application.getInitParameter(name)`

Session Level Attributes

- HTTP is stateless
- But your applications most often involve stateful sessions
- Session-level attributes pass data across the requests of a session
- App server provides implicit `session` object
- In servlets: `req.getSession()`, where `req` is the `HttpServletRequest` parameter
- Behind the scenes Tomcat employs cookies and/or URL rewriting to implement the session object

Maintaining Session Information with Implicit `session` Object

```
<html>
  <head><title>Counter Web Application</title></head>
  <body>
    <% Integer i =
         (Integer) (session.getAttribute("counter"));
       if (i == null) { i = new Integer(0); }
       else { i = new Integer(i.intValue() + 1); }
       session.setAttribute("counter", i);
    %>
    Your session has visited <%= i %> times this page.
  </body>
</html>
```

Session Duration

- Session data are automatically deleted after
  - client is inactive for a period
    - Tomcat default is 30 minutes
  - call of `HttpSession.invalidate()`
- Dynamic reset of session duration with `HttpSession.setMaxInactiveInterval()`
  - in seconds
- Set the default for all web apps following path `web-app/session-config/session-timeout` in `<CATALINA_HOME>/conf/web.xml`

Other Methods of Passing Information

Direct Use of the `response` Object

- Set values for various headers
  - `response.setContentType(String <MIME type>)`
- Add extra HTTP headers
  - **`addHeader`**`(java.lang.String name, java.lang.String value)`
  - Other "versions" for `int` and `Date` types
- Add cookies (discussed next)
- Send error responses
- …and other (see pg 118)

Cookies

- Way to store information on the client side
- Server includes `Set-Cookie` header
  - `Set-Cookie: multiply5Fid=%7BE2; path=/`
  - Implicitly associated with URL of server that provided
  - Explicitly associated with provided `path`
- Web client stores on cookie repository
  - If cookies from this site are enabled
  - Until expiration
    - Default is the browser session

Cookies (cont'd)

- When web client makes subsequent HTTP requests to domain/path, all matching cookies are attached
  - `Cookie: multiply5Fid =%7BE2`
- Constructor:

```
javax.servlet.http.Cookie(String name,
                          String value)
```

- `response.addCookie(Cookie value)`
- `request.getCookies()` returns `Cookie[]`
- Bunch of setter methods for changing default path, id, lifetime properties of cookie
  - More on pages 138-140 of textbook

When Should One Use Cookies?

- Use cookies if
  - No confidential info is released
  - You have to utilize their longevity
    - Cookies that live across browser startup/shutdown
  - Web app does not fall apart if cookies are disabled by client
- Example: preset some forms
- Do not use for standard session management aspects

Hidden Fields

- Passing (non-user input) information across requests
- You need an HTML form to be present
  - Not applicable with HTML links

```
<input type="hidden"
       name="<parameter>" value="<value>" />
```

- Prefer POST forms if you need to hide the hidden field from the URL
- Database keys are typical hidden fields
  - Example in databases section

URL Rewriting

What is Wrong with JSPs?

- Business logic & HTML content (presentation) mixed together
- Especially hard to maintain/evolve a program
- Still not very clean separation of web designer and web developer tasks