Java Server Pages, Session Management & Project Overview
ECE 155B – Spring 2012
Discussion Section 8.2

JSP Review

JSP Expression
<%= expression %>
Expression is evaluated and placed in the output. The XML equivalent is

<jsp:expression>
expression
</jsp:expression>

Predefined variables are request, response, out, session, application, config, and pageContext (available in scriptlets also).

JSP Scriptlet
<% code %>
Code is inserted into the service() method. The XML equivalent is

<jsp:scriptlet>
code
</jsp:scriptlet>.

JSP Declaration
<%! code %>
Code is inserted into the body of the servlet class, outside of the service() method. The XML equivalent is

<jsp:declaration>
code
</jsp:declaration>

JSP page Directive
<%@ page att="val" %>
Provides directions to the servlet engine about the general setup. The XML equivalent is
<jsp:directive.page att="val"\>.

Legal attributes, with default values in bold, are

- import="*package.class*"
- contentType="*MIME-Type*"
- isThreadSafe="**true**|false"
- session="**true**|false"
- buffer="*size*kb|none"
- autoflush="**true**|false"
- extends="*package.class*"
- info="*message*"

- errorPage="*url*"
- isErrorPage="true|**false**"
- language="java"

JSP include Directive

<%@ include file="url" %>

The file at the given url on the local system is to be included when the JSP page is translated into a servlet. The XML equivalent is
<jsp:directive.include file="url"\>.

The URL must be a relative path name. Use the jsp:include action to include a file at request time, instead of translation time.

JSP Comment
<%-- comment --%>

Comment. Ignored when the JSP page is translated into a servlet. If you want a comment in the resultant HTML, use the regular syntax for an HTML comment, i.e.,
<-- comment -->

The jsp:include Action
<jsp:include
  page="*relative URL*"
  flush="true"/>

Includes a file at the time that the page is requested. If you want to include the file at the time the page is translated, use the page directive with the include attribute instead. **Warning!** On some servers, the included file must be an HTML file or JSP file, as determined by the server (usually based on the file extension).

The jsp:useBean Action
<jsp:useBean att=val*/> or
<jsp:useBean att=val*>
...
</jsp:useBean>

Finds or builds a bean. The possible attributes are

- id="*name*"
- scope="**page**|request|session|application"
- class="*package.class*"
- type="*package.class*"
- beanName="*package.class*"

The jsp:setProperty Action
<jsp:setProperty att=val*/>

Sets bean properties, either explicitly or by designating that val comes from a request parameter. Legal attributes are

- name="*beanName*"
- property="*propertyName*|*"
- param="*parameterName*"
- value="*val*"

The jsp:getProperty Action

```
<jsp:getProperty
   name="propertyName"
   value="val"/>
```

Retrieves and outputs bean properties.

The jsp:forward Action

```
<jsp:forward
   page="relative URL"/>
```

Forwards the request to another page.

The jsp:plugin Action

```
<jsp:plugin attribute="value"*>
  ...
</jsp:plugin>
```

Generates OBJECT or EMBED tags, as appropriate to the browser type, asking that an applet be run using the Java Plugin.

JSP Expressions

<%= Java expression **%>**

```
Current time: <%= new java.util.Date() %>
<%= request.getRemoteHost() %>
```

- request, the HttpServletRequest
- response, the HttpServletResponse
- session, the HttpSession associated with the request (if any)
- out, the PrintWriter (a buffered version of type JspWriter) used to send output to the client.

JSP Scriptlets

<% Java Code **%>**

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

JSP Declarations

<%! Java Code **%>**

```
<%! private int accessCount = 0; %>
 Accesses to page since server reboot:
<%= ++accessCount %>
```

The JSP include Directive

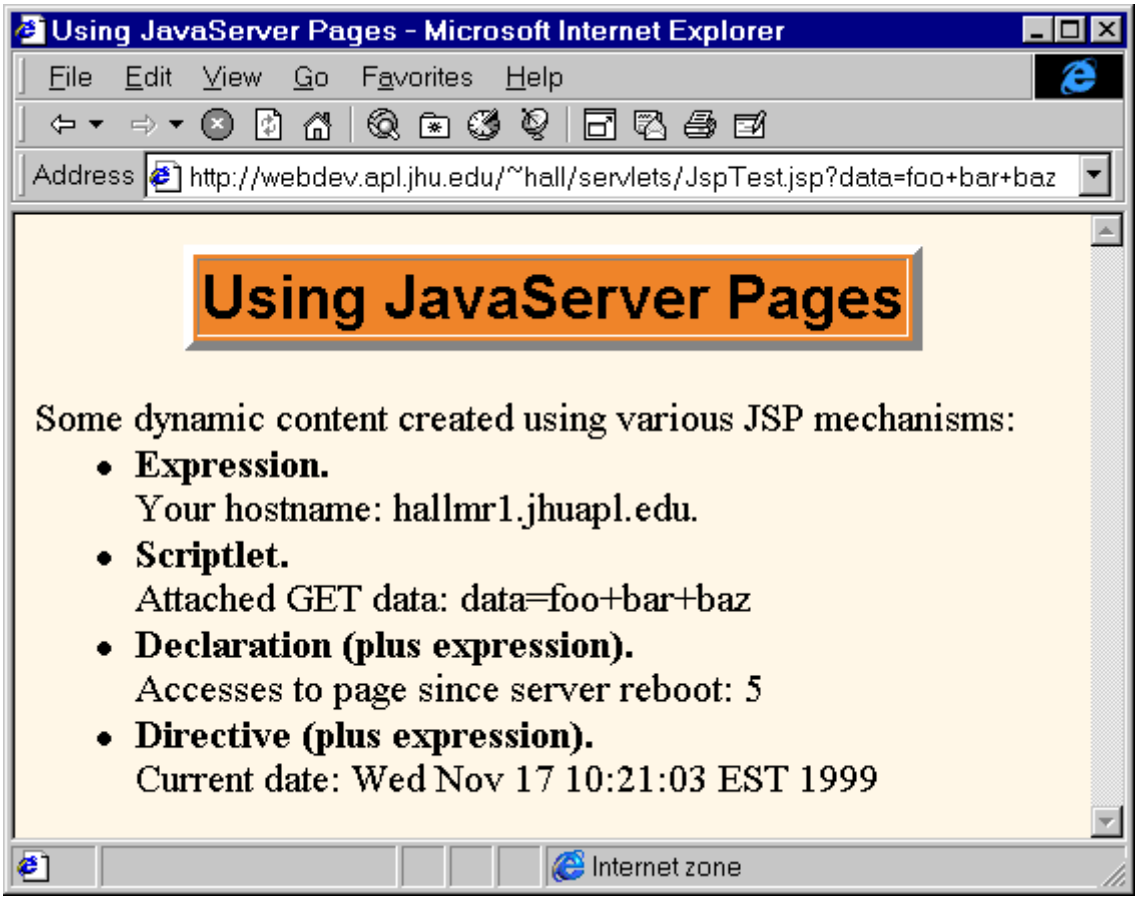
```
<%@ include file="relative url" %>
 <HTML>
<HEAD>
<TITLE>Servlet Tutorial</TITLE>
</HEAD>
<BODY>
<%@ include file="/navbar.html" %>
<!-- Part specific to this page ... -->
</BODY>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE>Using JavaServer Pages</TITLE>
</HEAD>

<BODY>

<CENTER>
<TABLE BORDER=5 BGCOLOR="#EF8429">
  <TR><TH CLASS="TITLE">
     Using JavaServer Pages</TABLE>
</CENTER>
<P>

Some dynamic content created using various JSP mechanisms:
<UL>
  <LI><B>Expression.</B><BR>
     Your hostname: <%= request.getRemoteHost() %>.
  <LI><B>Scriptlet.</B><BR>
     <% out.println("Attached GET data: " +
                    request.getQueryString()); %>
  <LI><B>Declaration (plus expression).</B><BR>
     <%! private int accessCount = 0; %>
     Accesses to page since server reboot: <%= ++accessCount %>
  <LI><B>Directive (plus expression).</B><BR>
     <%@ page import = "java.util.*" %>
     Current date: <%= new Date() %>
</UL>
</BODY>
</HTML>
```


Using JavaServer Pages - Microsoft Internet Explorer
File Edit View Go Favorites Help
Address http://webdev.apl.jhu.edu/~hall/servlets/JspTest.jsp?data=foo+bar+baz
Using JavaServer Pages
Some dynamic content created using various JSP mechanisms:
Expression.
Your hostname: hallmr1.jhuapl.edu.
Scriptlet.
Attached GET data: data=foo+bar+baz
Declaration (plus expression).
Accesses to page since server reboot: 5
Directive (plus expression).
Current date: Wed Nov 17 10:21:03 EST 1999
Internet zone

Predefined Variables

To simplify code in JSP expressions and scriptlets, you can use the following automatically defined variables, sometimes called *implicit objects*. These predefined variables are request, response, out, session, application, config, pageContext, and page.

7.1 request

This is the HttpServletRequest associated with the request, and lets you look at the request parameters (via getParameter), the request type (GET, POST, HEAD, etc.), and the incoming HTTP headers (cookies, Referer, etc). Request is allowed to be a subclass of ServletRequest other than HttpServletRequest, if the protocol in the request is something other than HTTP; however, this is almost never done in practice.

7.2 response

This is the HttpServletResponse associated with the response to the client. Note that, because the output stream (see out below) is buffered, it *is* legal to set HTTP status codes and response headers, even though this is not permitted in regular servlets once output has been sent to the client.

7.3 out

This is the PrintWriter used to send output to the client. However, to make the response object (described above) useful, this is a buffered version of PrintWriter called JspWriter. Note that you can adjust the buffer size, or even turn buffering off, through use of the buffer attribute of the page directive (discussed in Section 5). Also, note that out is used almost exclusively in scriptlets, because JSP expressions automatically get placed in the output stream, and thus rarely need to refer to out explicitly.

7.4 session

This is the HttpSession object associated with the request. Sessions are created automatically, so this variable is bound even if there was no incoming session reference. The one exception is if you use the session attribute of the page directive (see Section 5) turn sessions off, in which case attempts to reference the session variable cause errors at the time the JSP page is translated into a servlet.

7.5 application

This is the ServletContext as obtained via getServletConfig().getContext().

7.6 config

This is the ServletConfig object for the page.

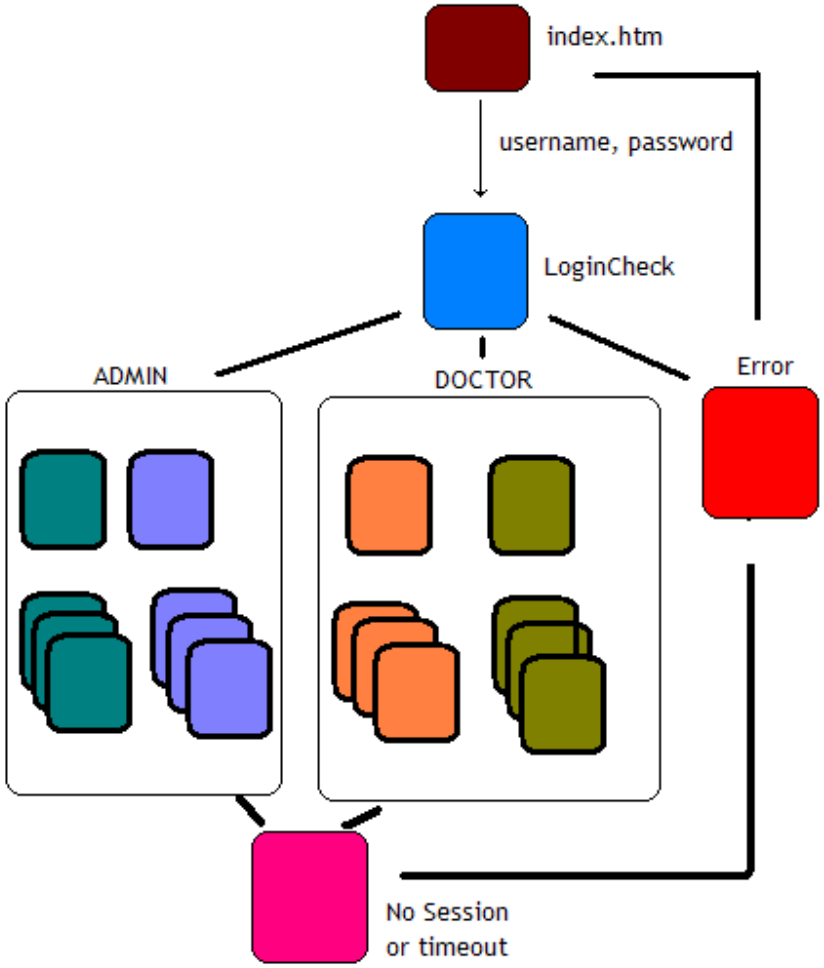
7.7 pageContext

PageContext is used to encapsulate use of server-specific features like higher performance JspWriters. If you access them through this class rather than directly, your code will still run on "regular" servlet/JSP engines.

7.8 page

page is simply a synonym for this, and is not very useful in Java. It was created as a placeholder when the scripting language is something other than Java.

Http Session Management



Session management is important in network programming. When users log into the system, for security reasons, the users' information is kept so that one cannot access pages without being authenticated.

To do this, in the index.html page, you ask for the username and password and send this information to the logincheck page where the user is validated and provided access to the database. If the user is valid, you create a HttpSession for the user.

```
User user = new User(name,password)
HttpSession thesession = request.getSession(true)
thesession.putValue("user",user)
```

Here, you can add an object to HttpSession, for example, vectors, objects of your own types (in our case User class). Then, you forward the authenticated user to the appropriate page. If you have more than one user type, like admin user, normal user, etc, you can check for this too. In this example, there are two types of users, ADMIN and DOCTOR, which are forwarded accordingly to the appropriate page.

```
<%
String name  = request.getParameter("name");
String pass    = request.getParameter("pass");

MediDB db = new MediDB ();

User user = db.FindUser(name);

if(user != null && user.getPass().equals(pass))
{
          HttpSession thesession = request.getSession(true);
          thesession.putValue("user",user);
           if(user.getType() == User.DOCTOR){
%>
           <jsp:forward page="logged.jsp" />
<%
          }else{
          if(user.getType() == User.ADMIN)
%>
           <jsp:forward page="loggedadmin.jsp" />
<%
               }
}
else
{
%>
          <p align="center">
          No such user/wrong password,<br>click to <A href="index.jsp"> continue </a>
          </p>
<%
}
%>
```

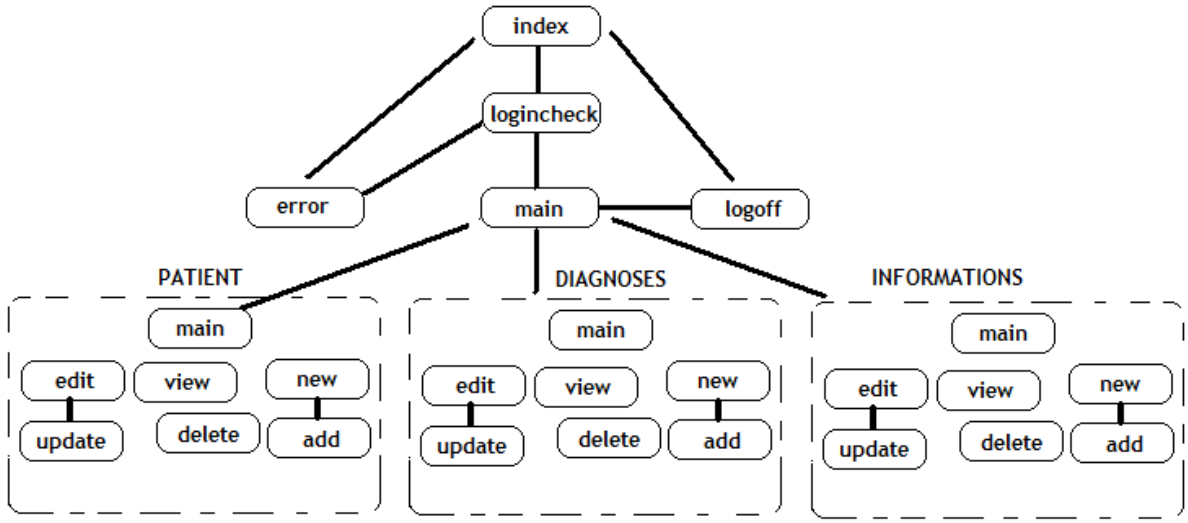
Now, we have a session with the provided user information, and the user is authenticated.

The following code can be added to all pages to authenticate a session.

```
HttpSession thesession = request.getSession();
User user= (User) thesession.getValue("user");
if(user == null)
{
          out.println("No session is created, you need to login: "+
          "<a href=\"index.jsp\"> do login </a> ");
          return;
}
```

In the above code, we get the HttpSession object and get the “user” value. If it is null, this means that there is no session created. The user either did not type the address or timed out. In either case, we will not allow any further operations.

Possible Web Layout of MediCare



Main Patient Information Statistics

Patients

There are 4 patient(s), new patient

	Social Security No	Name, Surname
1	111-11-1111	Firat, Kart
2	123-45-6789	Mücahit, Kart
3	222-22-1234	Cavit, Kart
4	233-44-5987	Ece155b

This is a possible main page for the Patients Web page. We display a list of patients (or something else). From this page, we can create, view, edit, and delete patients. It is nice to see an image in the User Interface. Here the patient information is not completely displayed. We enable only some of the fields to be seen on this page. When the user wants to see more patient information, we provide more information.

Here, we are viewing a patient. Remember that, if you want to edit patient, you should request a lock. So you cannot use this page to update patient information. Here is the edit patient page.

It is almost the same, but you request a lock for the patient. When you update this information, you release the lock.

You also need to be able to delete a patient.

It is nice to inform the user after all operations.

"*Patient information updated*"

"The patient is under modification, you cannot view it now!
Sorry for inconvenience..
Try again"

"The patient is under modification, you cannot edit it now!
Sorry for inconvenience..
Try again"

"Cannot update patient without lock on it"

"Patient deleted"

"Patient with ssn 111-11-1111 exists"

This is the Information page. In this page you can do the following:

Check if there is a "SSN" parameter passed to this page. If so, display the Information items for that patient. If not, display all information items for all patients.

How to inform the user of a problem with the database connection

```
MediDB medidb = null;
try
{
        medidb = new MediDB();
}
catch (Exception ex)
{
        out.println ("<br><br><center>Error connecting database <br>");
        out.println ("<img src='images/dbconnect.gif' /></center>");
        return;
}
```

How to list all patients

```
out.println("<table>");
out.println("<tr><td style='border-bottom:1px dashed crimson'><b>Social Security No</b></td><td
style='border-bottom:1px dashed crimson'><b>Name, Surname</b></td></tr>");
String colors [] = {"Cornsilk","Lavender"};

for(int i=0; i<list.size(); i++)
{
BasicPatient bpat = (BasicPatient) list.elementAt(i);
        out.println("<tr bgcolor="+colors[i%2]+"><td>"+(i+1)+
                "</td><td>"+bpat.GetSsn()+
                "</td><td>"+bpat.GetName()+
                "</td><td><a href='viewp.jsp?ssn="+bpat.GetSsn()+"'><img
src='images/addressbook.gif' border=0 alt='View'></a>"+
"</td><td><a href='editp.jsp?ssn="+bpat.GetSsn()+"'><img src='images/edit.gif' border=0
alt='Edit'></a>"+
"</td><td><a href='deletep.jsp?ssn="+bpat.GetSsn()+"'><img src='images/delete.gif' border=0
alt='Delete'></a>"+
                "</td></tr>");
 }
 out.println("</table>");
```