

5.0 - JAVA SERVLETS & JAVA SERVER PAGES

François Major
Université de Montréal

Francois.Major@UMontreal.CA

OBJECTIVES

- Implement dynamic Web pages using the JavaServer technology
- Learn the syntactic elements of the JavaServer pages
- Structure a Web application as a sequence of JavaServer pages
- Understand the relationships between JavaServer pages and servlets

CODES

- date.jsp
- localtime.java et time.jsp
- pizza.html
- manylocaltimes
- branching
- zonedb

Dynamic Web content

- JavaServer pages (*JSP*) can be used to implement dynamic Web pages
- To use *JSP*, we need an integrated server that possesses a *JSP* container
- The ***Apache Tomcat*** server is free!

Tomcat server at DIRO

- *NetBeans* comes with its own **Tomcat** server
- The DIRO has installed **Tomcat/Jakarta** on its Web server: www2 port 8080.
- To use it, follows the instructions given by the support team at DIRO:
<http://www.iro.umontreal.ca/support/servlet.html>
- One must wait at least four hours before using the server, the time required by the server to update the *xml* context of a user

Dynamic Web

- A *JavaServer* page contains HTML annotations, as well as Java instructions
- The **Java** instructions are executed each time the page is delivered to a browser
- For example, an instruction to insert the current date and time in a Web page is:

```
<%= new java.util.Date() %>
```

Let's go see the description of the date *JSP...*

To deploy the JSP date page

1. Introduce the JSP page code in a file using a text editor
2. Move the file in the Web application directory of your JSP engine. For instance, if you use Tomcat, create
C: \j akarta-tomcat\webapps\l ecture_5_0
3. Place the date. j sp file in the above directory
4. Start the Web server
5. Load http://localhost:8080/lecture_5_0/date.jsp
(or <http://localhost:8081/date.jsp> under **NetBeans**)

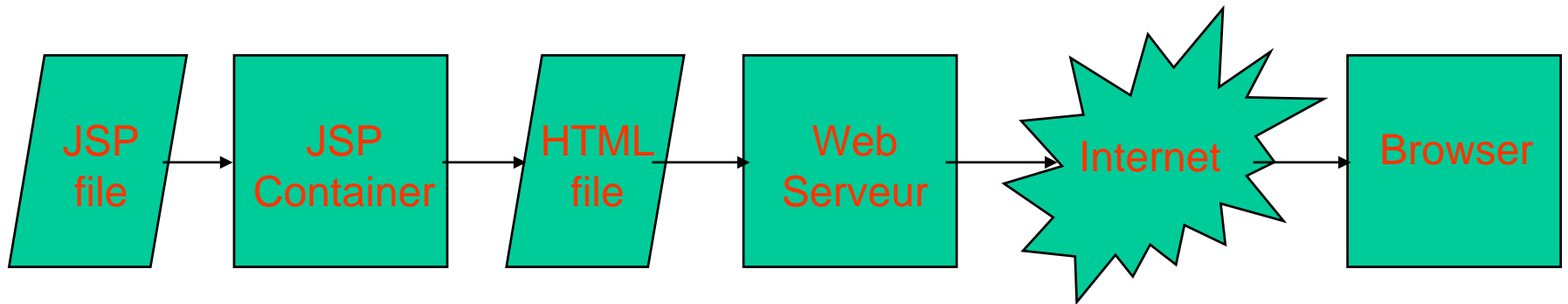
Dynamic Web Execution...

- The *JSP* container reads the requested page and transform it in HTML
- The HTML annotations are kept “as it”
- The *JSP* annotations (`<%= . . . %>`) are processed
- The expressions in the ***JSP*** annotations are evaluated and converted in character strings by invoking the `toString` method

... Dynamic Web execution

- The strings are introduced in the HTML page
- The resulting document contains HTML annotations only
- The Web server sends the document to the browser

The JSP Container Rewrite the Requested Page



Embedding Calculations Using *JavaBeans*

- Most Web page development on the *Internet* requires the contribution of two types of experts:
 - The programmer who understands and develops methods needed to compute and display the results
 - The graphics designer who determines how to display the results
- In this context, it is better to separate the *Java* code and the *HTML* annotations
- Every non-trivial computations should be placed and executed in a separated Java class (the *JavaBeans*)
- We can connect one or several *JavaBeans* to a *JSP* page

JavaBeans...

- A **JavaBean** is a Java class
- It needs a default constructor
- The **JavaBean** exposes its properties through the get and set methods
- The properties can be obtained and modified through methods which names follow a convention

JavaBeans...

- If the name of the property is *property* and its type is *Type*

- o access method

- Type* *getProperty()*

- o modification method

- void* *setProperty(Type* *newValue*)

- A Boolean property uses a different convention:

- boolean* *isProperty()*

- void* *setProperty(boolean* *newValue*)

- By convention, the name of the bean ends by **Bean**
- As with normal classes, no assumption must be made concerning the internal representation of beans

JavaBeans...

- A *JSP* page gives access to the properties of **JavaBeans** without having to develop **Java** codes
- To use a bean in a *JSP* page, we use the `jsp:useBean` statement
- We give a name to the object and we specify the name of the bean class

```
<jsp:useBean id = "tune" class="TuneBean" />
```

- This statement invokes the default constructor of the `TuneBean`
- It makes a new instance of a `tune` object

JavaBeans...

- To modify a property in the bean, we use the `setProperty` statement:

```
<jsp:setProperty name="tune" property="tempo" value="140" />
```

- To read a property from the bean, we use the `getProperty` statement:

```
<jsp:getProperty name="tune" property="tempo" />
```

- The latter returns a character string that is inserted in the resulting HTML page

JavaBeans...

- If, for example, we want to display the date only, without the time, from our `date.jsp`
- We can extract it using the `getTimeInstance` method of the `DateFormat` class
- We place this code in a bean

Let's go see some code...

TimeFormatterBean.java

time.jsp

JavaBeans

- It is good practice to put the bean statements at the beginning of the *JSP* file, before the HTML annotations.
- `time.jsp` and `TimeFormatterBean.java` must be placed in the appropriate directories:
 - `time.jsp` in `c:`
`C:\Jakarta-tomcat\webapps\lecture_5_0`
 - `TimeFormatterBean.java` in
`c:\Jakarta-tomcat\webapps\lecture_5_0\WEB-INF\classes`
- Remember, we use *JavaBeans* to separate HTML annotations from *Java* computations

Reading parameters...

- We want to modify our *JSP* page to present the local time in the user's city
- The *Java* library contains the `TimeZone` class:
 - A time zone is identified by a `String`:
 - "America/Montreal"
 - "Europe/Barcelona"
- The static method `getAvailableIDs` returns an array of `Strings` that contains all available zone identifiers (ID)
- The static method `getTimeZone` returns an object of the class `TimeZone` for a given ID:

```
String zoneID = "America/Montreal";  
TimeZone zone = TimeZone.getTimeZone(zoneID);
```

Reading parameters...

- The user enters a city such as Montreal
- Our *Java* bean verifies if the String “Montreal” exists
- The *JSP* page formats the current time for this time zone or prints an error message because the city has not been found or is not available

Reading parameters...

- We need an HTML form to get the user's input
- This form contains a text field and a button to submit the form to the *JSP* server

Let's go see `localtime.html`

Reading parameters...

- When a browser submits a form, it sends the names and values of all elements in the form:
 - Here, the name is “city”
 - Here, the value is the content of the corresponding field
- The `action` attribute of the form specifies the URL of the program-server that is processing the elements of the form
 - Here `localtime.jsp`
- In a *JSP* page, we can access the form data through the predefined object `request`
- The `getParameter` method of the class `ServletRequest` returns the value of an element from its name
- Here, to get the city typed by the user:

```
<%= request.getParameter("city") %>
```

Reading parameters

- To set the “city” property of Local TimeBean

```
<jsp:setProperty name="time" property="city"
  value="<%= request.getParameter(\"city\")%> />
```

- We can also use the shortcut:

```
<jsp:setProperty name="time" property="city" param="city" />
```

Let's go see localtime.jsp and LocalTimeBean.java...

HTML forms...

As previously seen, HTML forms can contain user-interface elements, such as:

- Text fields
- Password fields
- Text area
- Radio buttons
- Check boxes
- Selection lists
- Submit buttons
- Hidden text fields
- ...

Let's go see [pizza.html...](#)

HTML forms...

- Most of the elements of a HTML form use the following syntax:

`<i nput type=name_type_element_attributes/>`

- We must include the attribute names.
- Default values can be defined.

HTML forms...

- Text fields

```
<input type="text" name="title"  
value="" size="16" maxlength="30" />
```

- Passwords

```
<input type="password" name="Credit card  
number" size="16" maxlength="19" />
```

- Text area

```
<textarea name="..." rows="..." cols="...">  
default text  
</textarea>
```

HTML forms...

- Radio buttons

```
<input type="radio" name="Grosseur" value="S" />Small 10 po.  
<input type="radio" name="Grosseur" value="M" />Medium 12 po.  
<input type="radio" name="Grosseur" value="L" checked="checked" />Large 14 po.  
<input type="radio" name="Grosseur" value="XL" />X-Large 16 po.
```

Seulement un bouton radio peut être coché à la fois

- Check boxes

```
<input type="checkbox" name="Ingredients" value="Pe" />Peperoni  
<input type="checkbox" name="Ingredients" value="Ch" />Champignons  
<input type="checkbox" name="Ingredients" value="Ex" />Extra Fromage  
<input type="checkbox" name="Ingredients" value="Ba" />Bacon  
<input type="checkbox" name="Ingredients" value="Oi" />Olives  
<input type="checkbox" name="Ingredients" value="Ol" />Olives  
<input type="checkbox" name="Ingredients" value="Pi" />Piments  
<input type="checkbox" name="Ingredients" value="Po" />Poulet
```

Here, many boxes can be checked at the same time and will have the attribute “checked” set to “checked”. If two boxes are checked (cf. Bacon and Olives), then the browser will send the value `Ingredients=Ba+Ingredients=Ol` to the server.

HTML forms...

Selection lists

```
<select name="Crust">  
<option value="T" selected="selected">Traditional </option>  
<option value="M" >Thin</option>  
<option value="F" >Cheeze</option>  
</select>
```

- Submit buttons

```
<input type="submit" value="Place your order!" />
```

- Hidden fields (cannot be changed)

```
<input type="hidden" name="Credit card number"  
value="1234 5678 9012 3456" />
```

- o The browser does not display the hidden fields
- o It sends pairs name/value of the fields to the server when the form is submitted.

HTML Forms

- We place all elements of a form in an element of type form

```
<form acti on="pi zza. j sp" method="POST" >  
.  
.  
.  
</form>
```

- The HTML form must specify the URL of the program that process it
- The acti on attribute contains this URL
- We set the method attribute to the value POST because we do not impose any time limit for this form

Session...

- A session is a sequence of page queries from the same browser and Web server
- To capture a JSP page session, we use a bean and specify the session scope

```
<jsp:useBean id="tray" class="GroceryTrayBean"  
scope="session" />
```

- The user requests the JSP page a first time, a new tray is built
- The user returns to the same page or visit another page of the same application, the tray is still there
- If the bean does not have the scope attribute set to session, the bean has a scope of type page and a new object is built each time the page is visited

Session...

- Let's create a program that allows a user to add cities and their local times
- In the current version of our HTML application, we ask for the name of the first city
- The JSP page will display the local time of the first city and will ask for another city
- The user can add as many cities as he/she wishes
- All cities will be stored in an object of the class `ManyLocal TimesBean`

Session

- The bean object will have the scope set to session, and will thus keep the list of cities
- To keep the cities in a bean ManyLocalTimesBean

```
<jsp:useBean id="LocalTimes"  
    class="ManyLocalTimesBean" scope="session"/>
```

- One instance is created for each page
- The setVille method adds a city to the bean object
- The new city comes from the text field

```
<jsp:setProperty name="LocalTimes" property="city" param="city"/>
```

Let's go see [manylocaltimes.\[html | jsp\]](#) and [ManyLocalTimesBean.java...](#)

Page branching...

- A JSP page can send a query to another page
- We can use “forward” to implement page branching
- Send the query to a JSP page, which verifies the input but has no HTML output
- The page evaluates the input and uses the `jsp:forward` statement to select another page

Page branching...

- The formal syntax of the branching statement is:
`<jsp:forward page="url" />`
- Produce the loading of the page found at URL
- The current's bean data, as well as those of the beans with scope "request" are sent to the newly loaded page
- Branching page:

```
<%  
    if (condition)  
    {  
%>  
    <jsp:forward page="url1"/>  
%>  
    }  
    else  
    {  
%>  
    <jsp:forward page="url2"/>  
%>  
    }  
%>
```

NB. We said we were not including Java code in JSP pages, what is shown here!
Branching pages are an exception to this rule, as they do not contain any graphics, and in fact this page is not designed by the graphics designer

Page Branching

- We can use this technique to deal with the cities that are not found in any time zone
- We start with the HTML form, similar to the one built in the example
- Except we will send the data to a branching JSP page (no graphics)
- This page will then send the query to the appropriate page

Let's go see [branchingtime.\[html | jsp\]](#),
[timeresult.jsp](#) et [timeerror.jsp](#)...

Third Party Applications...

- Presentation party: The Browser
- Logical party: JSP engine and pages, and JavaBeans
- Storage party: MySQL database system

Third Party Applications...

- We will keep the time zones in a database
- We will build a class ZoneDBBean, which will consult the database to find a requested city
- If not available, we will lookup among the IDs as before
- We will use the class DataSourceBean to manage the connection with the DB

Third Party Applications...

- Only one instance of DataSourceBean is necessary for all JSP pages
- We set its scope to "application"
- We store the database properties in the file web.xml in sub-directory WEB-INF
- The JSP pages can access the information in web.xml
- We use the getInitParameter method of the predefined object application

Third Party Applications...

- We need to retrieve the information in `web.xml` to access the database and initialize it (login to SQL)
- `ZoneDBBean` makes a query to the database to find a requested city
- We need a SQL database: `ci_tzone.sql`, which contains the table `Ci_tZone`

cityzone.sql

1. CREATE TABLE CityZone(City CHAR(30), Zone CHAR(45))
2. INSERT INTO CityZone VALUES('San Francisco', 'America/Los_Angeles')
3. INSERT INTO CityZone VALUES('Quebec', 'America/Montreal')
4. ...

City	Zone
San Diego	America/Los_Angeles
Laval	America/Montreal
...	

Let's go see zonedb.[html | jsp],
DataSourceBean.java and
ZoneDBBean.java...

Servlets...

- When writing a servlet you extend the class `HttpServlet`
- JSP pages are automatically translated in servlets
- A servlet is a *Java* program that makes computations, some of which can produce HTML code