Server-side Web Programming

Lecture 20: The JSP Expression Language (EL)

Advantages of EL

• EL has more elegant and compact syntax than standard JSP tags
• EL lets you access nested properties
• EL let you access collections such as maps, arrays, and lists
• EL does a better job of handling null values
• EL provides more functionality
Disadvantages of EL

• EL doesn’t create a JavaBean if it doesn’t exist
• EL doesn’t provide a way to set properties
Syntax

- `${attribute}`: access an attribute name
- **Servlet code**
  ```java
  Date currentDate = new Date();
  request.setAttribute(“currentDate”, currentDate)
  ```
- **JSP code**
  ```html
  <p> The current date is ${currentDate} </p>
  ```

Syntax

- `${attribute.property}`: access the property of an attribute
- **Servlet code**
  ```java
  User user = new User(firstName, lastName, emailAddress);
  session.setAttribute(“user”, user)
  ```
- **JSP code**
  ```html
  <p> Hello ${user.firstName} </p>
  ```
When you use the dot operator, the code to the left of the operator must specify a JavaBean or a map, and the code to the right of the operator must specify a JavaBean or a map key.

When you use this syntax, EL looks up the attribute starting with the smallest scope (page scope) and moving towards the largest scope (application scope).

<table>
<thead>
<tr>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>page</td>
<td>stored in the pageContext object</td>
</tr>
<tr>
<td>request</td>
<td>stored in the HttpServletRequest object</td>
</tr>
<tr>
<td>session</td>
<td>stored in the HttpSession object</td>
</tr>
<tr>
<td>application</td>
<td>stored in the ServletContext object</td>
</tr>
</tbody>
</table>

Use this when you have a naming conflict.
• \${scope.attribute}
  Servlet code
  ```java
  Date currentDate = new Date();
  request.setAttribute("currentDate", currentDate)
  ```
  JSP code
  ```html
  <p> The current date is \${requestScope.currentDate} </p>
  ```
• \${scope.attribute.property}
  Servlet code
  ```java
  User user = new User(firstName, lastName, emailAddress);
  session.setAttribute("user", user)
  ```
  JSP code
  ```html
  <p> Hello \${sessionScope.user.firstName} </p>
  ```

**Use [ ] operator to work with arrays and lists**

• \${attribute["propertyKeyOrIndex"]}
  Servlet code
  ```java
  String[] colors = {"Red", "Green", "Blue");
  ServletContext application = this.getServletContext();
  application.setAttribute("colors", colors);
  ```
  JSP code
  ```html
  <p> The first color is \${colors[0]} </p>
  <br>
  The second color is \${colors[1]} </p>
  ```
  Another way to write JSP code
  ```html
  <p> The first color is \${colors["0"]} </p>
  <br>
  The second color is \${colors["1"]} </p>
  ```
Servlet code
   ArrayList<User> users = UserIO.getUsers(path);
   session.setAttribute("users", users);

JSP code
   <p> The first address on our list is ${users[0].emailAddress} <br>
       The second address on our list is ${users[1].emailAddress} </p>

Another way to write JSP code
   <p> The first address on our list is ${users["0"].emailAddress} <br>
       The second address on our list is ${users["1"].emailAddress} </p>

Use Dot operator to access nested properties
  • ${attribute.property1.property2}

Servlet code
   Product p = new Product();
   p.setCode("pf01");
   LineItem lineItem = new LineItem(p,10);
   session.setAttribute("item", lineItem);

JSP code
   <p> Product code: ${item.product.code} </p>
Another way to access the nested property

Syntax: \${attribute["property1"].property2}

Servlet code

Product p = new Product();
p.setCode("pf01");
LineItem lineItem = new LineItem(p,10);
session.setAttribute("item", lineItem);

JSP code

<p> Product code: \${item["product"].code} </p>

There is no limit to the number of nested properties that you can access with the dot operator

Other Implicit EL Objects

- **pageContext.** The PageContext object.
  - E.g. \${pageContext.session.id}
- **param and paramValues.** Request params.
  - E.g. \${param.custID}
- **header and headerValues.** Request headers.
  - E.g. \${header.Accept} or \${header["Accept"]}
  - \${header["Accept-Encoding"]}
- **cookie.** Cookie object (not cookie value).
  - E.g. \${cookie.userCookie.value} or \${cookie["userCookie"].value}
- **initParam.** Context initialization param.
Example

```html
<!DOCTYPE ...>
...
<UL>
    <LI><B>test Request Parameter:</B> 
        ${param.test}
    <LI><B>User-Agent Header:</B> 
        ${header["User-Agent"]}
    <LI><B>JSESSIONID Cookie Value:</B> 
        ${cookie.JSESSIONID.value}
    <LI><B>Server:</B> 
        ${pageContext.servletContext.serverInfo}
</UL>
</BODY></HTML>
```

Example

```
- test Request Parameter: blah
- User-Agent Header: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.0.1) Gecko/20020823 Netscape/7.0
- JSESSIONID Cookie Value: 734b657be5bbad70d65ca475211d7cc7
- Server: Apache Tomcat/5.0.3
```
EL Operators

- **Arithmetic**
  - + - * / div % mod

- **Relational**
  - == eq != ne < lt > gt <= le >= ge

- **Logical**
  - && and || or ! Not

- **Empty**
  - Empty
  - True for null, empty string, empty array, empty list, empty map. False otherwise.

- **CAUTION**
  - Use extremely sparingly to preserve MVC model

Example

```
<TABLE BORDER=1 ALIGN="CENTER">
  <TR><TH CLASS="COLORED" COLSPAN=2>Arithmetic Operators</TH><TH CLASS="COLORED" COLSPAN=2>Relational Operators</TH></TR>
  <TR><TH>Expression</TH><TH>Result</TH><TH>Expression</TH><TH>Result</TH></TR>
  <TR ALIGN="CENTER">
    <TD>\${3+2-1}</TD>$\{3+2-1\}$
    <TD>\${1&lt;2}</TD>$\{1<2\}$
    <TD>\${3%2}</TD>$\{3%2\}$
    <TD>\${3/4 == 0.75}</TD>$\{3/4 == 0.75\}$
  </TR>
</TABLE>
```
Common (but Confusing) EL Problem

• Scenario
  – You use ${something} in a JSP page
  – You literally get "${something}" in the output
  – You realize you forgot to update the web.xml file to refer to servlets 2.4, so you do so
  – You redeploy your Web app and restart the server
  – You still literally get "${something}" in the output

• Why?
  – The JSP page was already translated into a servlet
    • A servlet that ignored the expression language

• Solution
  – Resave the JSP page to update its modification date
Preventing EL Evaluation

- **What if JSP page contains ${ ?**
- **Deactivating the EL in an entire Web application.**
  - Use a web.xml file that refers to servlets 2.3 (JSP 1.2) or earlier.
    
    ```
    <jsp-config>
        <jsp-property-group>
            <url-pattern>*.jsp</url-pattern>
            <el-ignored>true</el-ignored>
        </jsp-property-group>
    </jsp-config>
    ```

- **Deactivating the expression language in multiple JSP pages.**
  - Use the jsp-property-group web.xml element

- **Deactivating the expression language in individual JSP pages.**
  - Use `<%@ page isELIgnored="true" %>`
    
    - This is particularly useful in pages that use JSTL

Preventing Use of Standard Scripting Elements

- To enforce EL-only with no scripting, use scripting-invalid in web.xml

  ```
  <jsp-config>
      <jsp-property-group>
          <url-pattern>*.jsp</url-pattern>
          <scripting-invalid>true</scripting-invalid>
      </jsp-property-group>
  </jsp-config>
  ```
Summary

• The JSP 2.0 EL provides concise, easy-to-read access to
  – Bean properties
  – Collection elements
  – Standard HTTP elements such as request parameters, request headers, and cookies

• The JSP 2.0 EL works best with MVC
  – Use only to output values created by separate Java code

• Resist use of EL for business logic