EL + JSTL

Advanced Topics in Java

Khalid Azim Mughal
khalid@ii.uib.no
http://www.ii.uib.no/~khalid/atij/

Version date: 2006-09-04

Overview

- Accessing Java Code using JSP Expression Language (EL)
- Introduction to using JSP Standard Tag Library (JSTL) in JSP Pages

Accessing Java Code using JSP Expression Language (EL)

- Simplifies using the "Pull Model".
  - Provides a navigational notation to pull information from the model and into a JSP page.
  - Makes using the server state easier in a JSP page.
- The expression language automatically handles typecasting, `null` values, and error handling.
- Notation for evaluating and returning the textual representation of values that are stored in the standard scopes:

```
${ expression }
```


Topics on JSP EL

- Activating EL Evaluation
- Accessing Attributes in the Standard Scopes
- Accessing Properties of Beans stored as Attributes in the Standard Scopes
- Accessing Collections stored as Attributes in the Standard Scopes
- Using EL Implicit Objects
- Using EL Operators
- EL Conditional Evaluation

Activating EL Evaluation

- The `web.xml` file should be compatible with JSP 2.0:

```
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/
web-app_2_4.xsd">
  <display-name>Shopping</display-name>
  ...
</web-app>
```


EL Syntax

- Syntax: **`${ expression }`**
- The result of expression evaluation replaces the expression and its delimiters.
- EL expressions are comprised of literals, operators and variables.
- Used in template text and within tags:
  - The *value of an attribute stored in a scope* can be accessed in an EL expression.
  - The *value of an attribute in tags/elements* can be specified by an EL expression.
  - *Functions* can also be called from EL expressions (covered later in these notes).
- To escape `${`, use `\${`.

Accessing Attributes in the Standard Scopes

- The variables in EL expressions can reference attributes that are stored in the standard scopes.
- The variable is looked up in the standard scopes in the following *order*: page, request, session, or application.
- The first match is returned. If no match, return `null`.
- The value of the variable is converted to a string if necessary by calling the `toString()` method.

```
User name: ${username} <%-- Assume username is an attribute in session scope --%>
User name: ${sessionScope.username} <%-- Restricting scope search--%>
```

are equivalent to:

```
User name: <%= session.getAttribute("username") %>
```

Standard Scope	EL Implicit Object with Attributes from the Standard Scope (*Maps*)
page	`pageScope`
request	`requestScope`
session	`sessionScope`
application	`applicationScope`

Accessing Properties of Beans stored as Attributes in the Standard Scopes

- Syntax: **`${ beanAttr.prop`$_1$`.prop`$_2$`. ... .prop`$_k$` }`**
- All `prop`$_i$ are legal Java identifier.
- The dot operator allows navigating from one property to another every time it is used in an expression.

```
<%--
     Assume that userinfo refers to a bean in a standard scope.
     Assume that this bean has a property called address which is also a bean.
--%>
${userinfo.username} <%-- (1) --%>
${userinfo.address.street}
${userinfo.address.zipcode}
```

Accessing Collections stored as Attributes in the Standard Scopes

- If the attribute is an array, a List or a Map, then the following syntax can be used:

  `${ beanAttr[indexValue] }`

```
${userTab[0]}    <%-- userTab is the name for an array attribute.
                      The value inside [] is used as an index in the array --%>
${userTab["0"]}  <%-- userTab is the name for an array attribute. --%>
${userTab.0}     <%-- Illegal. Not legal Java identifier. --%>

${itemList[0]}   <%-- itemList is the name of an List attribute --%>


${callMap["Monday"]}  <%-- (1) callMap is the name of an Map attribute with
                              entries <name of day, no. of calls>.
                              "Monday" is used as a key for the lookup.  --%>
${callMap[Monday]}    <%--     null, no attribute named Monday in scope
${callMap.Monday}     <%--     Equivalent to (1)  --%>
${callMap.Holiday}    <%--     null, does not throw an exception  --%>
${itemList[callMap["Monday"]]} <%--    ok  --%>

${shows["Holiday-on-ice"]}     <%--    ok  --%>
${shows.Holiday-on-ice}        <%--    0, illegal property name  --%>
```

Note that the name is the key used to store the object in the scope.

Other EL Implicit Objects

Name of EL Implicit Object	Type of Implicit Object	Description	Example
`pageContext`	`PageContext`	Allows access to the `PageContext` of the current page. This class has properties: `request`, `response`, `session`, `servletContext`	`${pageContext.session.id}` `${pageContext.request.method}`
`param` `paramValues`	`Map`	Allows access to the request parameters	`${param.sign}` `${param["sign"]}`
`header` `headerValues`	`Map`	Allows access to the request header	`${header.Accept}` `${header["Accept-Encoding"]}`
`initParam`	`Map`	Allows access to the *context* initialization parameters	`${initparam.e_mail_addr}` `${initparam["e_mail_addr"]}`
`cookie`	`Map`	Allows access to the incoming cookie using the property `value`	`${cookie.userCookie.value}` `${cookie["userCookie"].value}`

EL Implicit Object (cont.)

- Getting the HTTP request method.

```
${request.method}              <%-- No. There is no implicit request object --%>
${requestScope.method}         <%-- No. This implicit object doe not have request
                                    info, only attributes stored in it.--%>
${pageContext.request.method} <%-- ok. Request properties thru pageContext --%>
```


Using EL Operators

- Used for simple tasks related to presentation logic.
- Operators have the standard precedence and associativity rules.
- EL expressions with only literals and operators are evaluated as arithmetic expressions.
- Predefined EL literals: `true`, `false`, `null`.

Type of Operators	Operators	Example	Result

Indexing	`[]`
	`.`
	`( )`
		`${initparam["dt_table"]}`
		`${array["0"]}`
		`${initparam.dt_table}`
		`${ 2 * (3 + 4)}`	

Arithmetic	`-`
	`* / div`
	`% mod`
	`+ -`
		`${1 + 2 * 3}`
		`${"10" + 2}`
		`${100 % 98}`
		`${var * 2}`
			`7`
			`12`
			`2`
			`var coerced to number before operation`

Type of Operators	Operators	Example	Result

Relational	`== eq`
	`!= ne`
	`< lt`
	`> gt`
	`<= le`
	`>= ge`
		`${obj == obj}`
		`${2 == 3}`
		`${obj == null}`
		`${bigI == bigJ}`
		`${obj1 == obj2}`
		`${"10" < "2"}`
			`true`
			`false`
			`false`
			`dep. on compareTo(bigI, bigJ)`
			`value of obj1.equals(obj2)`
			`true`

Logical	`&& and`
	`|| or`
	`! not`
		`${obj1 != null &&`
		`obj2 != null}`
		`${!(2 == 3)}`
			`true`

The Empty Operator	`empty`	`${empty null}` `${empty attr.prop}`	`true` `true` if `attr.prop` is `null`, an empty string, an empty array, an empty map, or an empty collection, i.e an empty container.

The Ternary Operator	`test ? trueExpr : falseExpr`	`${(shopping.total > 1000)`
		`? "Esteemed Customer"`
		`: "Dear Customer" }`
			Dependent on the test.

`null`-friendly JSP EL

- In expressions containing dot or `[]` operators, an undefined variable is treated as having the `null` value.
- In arithmetic and relational expressions, an undefined variable is treated as 0 (zero).
- In logical expressions, an undefined variable is treated as `false`.

```
<%-- ying and yang are not defined. --%>
${ying}        <%-- No output --%>
${ying.yang}   <%-- No output --%>

${10 - yang}   <%-- 10 --%>
${10 / yang}   <%-- infinity --%>
${10 % yang}   <%-- Exception --%>

${10 < yang}   <%-- false --%>
${not yang}    <%-- true --%>
```

Implementing JSP EL Functions

1. Implement a Java class with a `public static` method.
   - Can have a non-empty parameter list, and is normally declared as non-`void`.
   - The class file is installed in `/WEB-INF/classes` directory
2. Create a Tag Library Descriptor (TLD) file specifies a mapping between the function and the JSP page.
   - The `<taglib>` element is used to specify the pertinent information.
   - A `<uri>` subelement specifies the name that will be used in a JSP page.
   - A `<function>` subelement specifies the name (`<name>`) that will be used to invoke the function from a JSP page, the class of the function (`<function-class>`) , the signature of the method (`<function-signature>`) .
   - The TDL file has a `.tld` extension and can be installed in `/WEB-INF` directory.
3. Create a `taglib` directive in the JSP page.
   - The `prefix` attribute specifies the *namespace* for invoking the method.
   - The `uri` attribute value matches the one in the TLD file.

   ```
   <%@ taglib prefix="namespaceInTaglibDirective" uri="uriValueInTLD" %>
   ```
4. Invoke the EL function from the JSP page.

   ```
   ${namespaceInTaglibDirective:nameinTLD()}
   ```


Example: JSP EL function (ELFunction application)

- The class (`DateWrapper.java`) implementing the function

```
package kam;
import java.util.Date;
public class DateWrapper {
  public static Date getDate() {
    return new Date();
  }}
```

- The JSP file (`callingDateFunction.jsp`)

```
<%@ taglib prefix="kam" uri="/mughal/data" %>
<html>
  <head><title>Calling EL Function</title></head>
  <body><h1>Calling EL Function</h1>
  <h1>${kam:dateMe()}</h1>
  </body>
</html>
```

- The TLD file (`DatingFunction.tld`)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<taglib version="2.0" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/
web-jsptaglibrary_2_0.xsd">
  <tlib-version>1.2</tlib-version>
  <uri>/mughal/data</uri>
  <function>
    <name>dateMe</name>
    <function-class>kam.DateWrapper</function-class>
    <function-signature>java.util.Date getDate()</function-signature>
  </function>
</taglib>
```

Introduction to using JSP Standard Tag Library (JSTL) in JSP Pages

- **The Core Library:** `<c: ... >`
- The Formatting Library: `<fmt: ... >`
- The SQL Library: `<sql: ... >`
- The XML Library: `<x: ... >`

Installing JSTL 1.1

- From `TOMCAT_HOME\webapps\jsp-examples\WEB-INF\lib`, copy the following files to `myApp\WEB-INF\lib`:

  ```
  - jstl.jar
  - standard.jar
  ```

- These `jar` files can also be installed in `TOMCAT_HOME\shared\lib`, so that all applications can share them.

Core JSLT

General purpose

- Output: `<c:out>`
- Set property: `<c:set>`
- Removing attributes: `<c:remove>`
- Exception handling: `<c:catch>`

Conditionals

- Conditional Include: `<c:if>`
- Multiple Choice: `<c:choose>`, `<c:when>`, `<c:otherwise>`

URL-related

- Importing: `<c:import>`
- URL rewriting: `<c:url>`
- Redirecting: `<c:redirect>`
- Parameter passing: `<c:param>`

Iteration

- Iteration: `<c:forEach>`
- Tokenizing: `<c:forTokens>`

The `<c:set>` Tag

- Version for setting *attribute variables*:

```
<c:set var=... value=... scope=... />
```

or

```
<c:set var=... scope=... >
  ...value inside body...
</c:set>
```

- Version for setting *bean properties* and *Map values*:

```
<c:set target=... property=... value=... />
```

or

```
<c:set target=... property=... >
  ...value inside body...
</c:set>
```

  - The `target` must evaluate to a reference.

Example: <c:set>

- See files `setTag.jsp` and `kam.Item.java` in `myJSTLExamples` application.

```
<%@ page import="java.util.*, kam.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
  String[] daysArray = {"Monday","Tuesday","Wednesday","Thursday",
                        "Friday","Saturday","Sunday"};
  session.setAttribute("arrayOfDays", daysArray);

  List daysList = new ArrayList();
  daysList.add("Monday"); daysList.add("Tuesday"); daysList.add("Wednesday");
  daysList.add("Thursday"); daysList.add("Friday"); daysList.add("Saturday");
  daysList.add("Sunday");
  request.setAttribute("listOfDays", daysList);
```



```
  Map daysMap = new HashMap();
  daysMap.put(new Integer(1), "Monday");
  daysMap.put(new Integer(2), "Tuesday");
  daysMap.put(new Integer(3), "Wednesday");
  daysMap.put(new Integer(4), "Thursday");
  daysMap.put(new Integer(5), "Friday");
  daysMap.put(new Integer(6), "Saturday");
  daysMap.put(new Integer(7), "Sunday");
  request.setAttribute("mapOfDays", daysMap);

  Item item = new Item("Shoes", "Ekin", 999.90, 1);

  Item[] itemsArray = {
      new Item("Shoes", "Ekin", 999.90, 10),
      new Item("Shirt", "Van Heusen", 9.90, 30),
      new Item("Shampoo", "Simple", 99.90, 15),
      new Item("Socks", "Golden Toe", 900.00, 9),
      new Item("Suit", "Armani", 49.90, 17)
  };
  request.setAttribute("arrayOfItems", itemsArray);
%>
```

```
<HTML><HEAD><TITLE>myJSTLExamples</TITLE></HEAD>
<BODY>
<h1>JSTL Examples</h1>

1: <c:out value="Hello" /><br/>

<c:set var="myShoes" value="<%= item %>" scope="request" />
2: <c:out value="${myShoes}"/><br/>

<c:set var="day" value="${listOfDays[2]}" scope="request" />
3: <c:out value="${day}"/><br/>

<c:set var="attribute" value="in session" scope="session" />
4: <c:out value="attribute ${attribute}" /><br/>

<c:set target="${myShoes}" property="itemDescription" value="sadida" />
5: <c:out value="${myShoes}"/><br/>
6: ${myShoes}<br/>

<c:set target="<%= item %>" property="itemDescription" value="sadida" />
7: <c:out value="<%= item %>"/><br/>
8: <%= item %><br/>
```



```
9: <c:out value="${mapOfDays}"/><br/>
<%-- This property specification for the key does not work in a map. --%>
<%--
<c:set target="${mapOfDays}" property="<%= new Integer(7) %>" value="FREE DAY" />
--%>
<%-- String property specification for a key works in a map. --%>
<c:set target="${mapOfDays}" property="7" value="FREE DAY" />
10: <c:out value="<%= daysMap %>"/><br/>

</BODY></HTML>
```

Output in browser window:

```
1: Hello
2: Shoes Ekin 999.9 1
3: Wednesday
4: attribute in session
5: Shoes sadida 999.9 1
6: Shoes sadida 999.9 1
7: Shoes sadida 999.9 1
8: Shoes sadida 999.9 1
9: {2=Tuesday, 4=Thursday, 6=Saturday, 1=Monday, 3=Wednesday, 7=Sunday, 5=Friday}
10: {2=Tuesday, 4=Thursday, 6=Saturday, 1=Monday, 3=Wednesday, 7=FREE DAY, 5=Friday}
```

The <c:forEach> Tag

- Used to iterate over the elements in an iterable: an array, `Collection`, `Map`, `ResultSet` or a comma-separated `String`s.

```
<c:forEach var="anItem" items="some iterable">
    ${anItem.property} <%-- The variable anItem is only accessible in the
                            local scope of the forEach loop. --%>
    ...
</c:forEach>
```

- See files `forEachTag.jsp` and `kam.Item.java` in `myJSTLExamples` application.
- Examples:

The loop

```
<c:forEach var="item" items="en, to, tre, fire" >
     ${item}<br/>
</c:forEach>
```

prints:

```
en
to
tre
fire
```


The loop

```
<table>
  <c:forEach var="item" items="${arrayOfItems}" varStatus="itemLoopCount" >
  <tr>
      <td>Item no. ${itemLoopCount.count}:</td>
      <td>${item.itemName}</td>
      <td>${item.itemDescription}</td>
      <td>${item.price}</td>
      <td>${item.quantity}</td>
  </tr>
  </c:forEach>
</table>
```

prints:

```
Item no. 1: Shoes   Ekin       999.9 10
Item no. 2: Shirt   Van Heusen 9.9   30
Item no. 3: Shampoo Simple     99.9  15
Item no. 4: Socks   Golden Toe 900.0 9
Item no. 5: Suit    Armani     49.9  17
```

- An optional attribute called `varStatus` can be defined. It provides access to the iteration number.
- `forEach`-loops can be nested.

The <c:if> Tag

- Allows the contents of its body to be executed or skipped, depending on the test condition. Note there is no <c:else> tag.

```
<c:if test="some condition" >
    ...if-body...
</c:if>
```

- Example: See file conditionalTags.jsp in myJSTLExamples application.

```
<c:forEach var="item" items="${arrayOfItems}" varStatus="itemLoopCount" >
  <c:if test="${itemLoopCount.count % 2 == 0}">
    <tr>
        <td>Item no. ${itemLoopCount.count}:</td>
        <td>${item.itemName}</td>
        <td>${item.itemDescription}</td>
        <td>${item.price}</td>
        <td>${item.quantity}</td>
    </tr>
  </c:if>
</c:forEach>
```

Output:

```
Item no. 2: Shirt Van Heusen 9.9   30
Item no. 4: Socks Golden Toe 900.0 9
```


The Multiple Choice Tags: <c:choose>, <c:when>, <c:otherwise>

- Allows the body of *only one* <c:when> tag to be executed.
- If none of the <c:when> tags match, the body of the <c:otherwise> tag is executed.
- There is no fall-through.

```
<c:choose>
  <c:when test="..." >
    ...
  </c:when>
  <c:when test="..."  >
    ...
  </c:when>
  <c:when test="..."  >
    ...
  </c:when>
  ...
  <c:otherwise>    <%-- OPTIONAL --%>
    ...
  </c:otherwise>
</c:choose>
```

- Example: See file `conditionalTags.jsp` in `myJSTLExamples` application.

  The following code:

```
<c:forEach var="day" items="${listOfDays}" >
<c:choose>
  <c:when test="${day == 'Saturday'}" >
  HOLIDAY
  </c:when>
  <c:when test="${day == 'Sunday'}" >
  HOLIDAY
  </c:when>
  <c:otherwise>
  ${day}
  </c:otherwise>
</c:choose>
</c:forEach>
```

  generates:

```
Monday Tuesday Wednesday Thursday Friday HOLIDAY HOLIDAY
```


Example: the `myELShop` application

- The `myELShop` web application is a rewrite of the `myEShop` application using EL and JSTL.
- See the following files for the `myELShop` web application:
  - `index.html`,
    `WEB-INF\web.xml`,
    `WEB-INF\src\kam\shopping\Item.java`,
    `WEB-INF\src\kam\shopping\ELShoppingServlet.java`,
    `ELShop.jsp`, `ELCart.jsp`, `ELCheckout.jsp`, `ELError.html`
  - `myELShop.xml`

JSTL Libraries Overview

CORE LIBRARY	FORMATTING LIBRARY	XML LIBRARY	SQL LIBRARY
GENERAL PURPOSE	**INTERNATIONALIZATION**	**CORE XML**	**DATABASE**
<c:out>	<fmt: message>	<x:parse>	<sql:query>
<c:set>	<fmt: setLocale>	<x:out>	<sql:update>
<c:remove>	<fmt: bundle>	<x:set>	<sql:setDatasource>
<c:catch>	<fmt: setBundle>	**XML FLOW**	<sql:param>
CONDITIONAL	<fmt: param>	<x:if>	<sql:dateParam>
<c:if>	<fmt: requestEncoding>	<x:choose>	
<c:choose>	**FORMATTING**	<x:when>	
<c:when>	<fmt: timeZone>	<x:otherwise>	
<c:otherwise>	<fmt: setTimeZone>	<x:forEach>	
URL-RELATED	<fmt: formatNumber>	**TRANSFORMATION**	
<c:import>	<fmt: parseNumber>	<x:transform>	
<c:url>	<fmt: parseDate>	<x:param>	
<c:redirect>			
<c:param>			

CORE LIBRARY	FORMATTING LIBRARY	XML LIBRARY	SQL LIBRARY
ITERATION			
<c:forEach>			
<c:forTokens>			