KLEFSPRO
Learn & Grow

EL And JSTL

Objectives

- To understand Expression Language (EL) in Jsp.
- Discuss JSTL core tags.
- Introduction to JSTL formatting tags.
- Introduction to JSTL SQL tags.
- Introduction to JSTL XML tags.

Revision no./Date:00/Oct.09 2008

Problem With <jsp:useBean>

- What if we want to print a property of a property??? e.g. What if we want to print the name of Person's dog??
- **Using scripting**

```
<%=((p1.Person)
request.getAttribute("person")).
 getDog().getName() %>
```

- **Using standard actions**

```
<jsp:useBean id="p" class="p1.Person">
   Dog name is
<jsp:getProperty name="p"
property="dog"/>
```

O/p in browser window

Dog name is foo.Dog@228994

Revision no./Date:00/Oct.09 2008

Problem With <jsp:useBean>

- The <jsp:getProperty> lets you access only the properties of the bean attribute.
- There is no capability for nested properties, where you want a property of a property, rather than a property of an attribute.

Revision no./Date:00/Oct.09 2008

Expression Language

- **JSP Code**

```
Dog Name is
${person.dog.name}
```

O/p in browser window

Dog Name is Tommy

- EL makes it easy to print nested properties

Revision no./Date:00/Oct.09 2008

Expression Language

- EL expressions are always written within curly braces, and prefixed with the '$' sign.
- The first named variable in the expression is either an implicit object or an attribute.
  - `${person.name}`

Revision no./Date:00/Oct.09 2008

Implicit Objects

- EL provides several implicit objects:
  - `pageScope`
  - `requestScope`
  - `sessionScope`
  - `applicationScope`
  - `param`
  - `header`
  - `cookie`
  - `initParam`
  - `pageContext`

Revision no./Date:00/Oct.09 2008

Expression Language

- `${person.name}`
  - What if person is
    - an array?
    - List?
    - Or a Map?

Revision no./Date:00/Oct.09 2008

Expression Language

- If an expression has a variable followed by a bracket [], the left hand variable can be a **Map**, a **bean**, a **List** or an **array.**
- E.g.
  - `${songList["something"]}`

Revision no./Date:00/Oct.09 2008

Using [ ] with an array

- E.g.

```
String[] myFavSongs =
{“Nothing Gonna”, “Titanic”, “BT”};
request.
setAttribute(“musicList”, myFavSongs);
```

- In JSP
  - `Second  Song is: ${musicList[“1”]}`
  - `Third  Song is: ${musicList[2]}`

- O/p in browser window
  - `Second Song is: Titanic`
  - `Third  Song is: BT`

Revision no./Date:00/Oct.09 2008

Working with Map

- E.g.

```
Map songMap = new HashMap();
songMap.put("Ambient", "Zero 7");
songMap.put("Surf", "Tahiti 80");
songMap.put("DJ", "BT");
request.setAttribute("songMap", songMap);
```

- In JSP
  - `Ambient is:  ${songMap.Ambient}`
  - `DJ is: ${songMap[„DJ"]}`
- O/p in browser window
  - `Ambient is: Zero 7`
  - `DJ is: BT`

Revision no./Date:00/Oct.09 2008

Implicit Objects

- The `param` implicit object is fine when you know you have only one parameter for that particular parameter name.
- Use `paramValues` when you might have more than one parameter value for a given parameter name.

Revision no./Date:00/Oct.09 2008

Implicit Objects

- HTML Code

```
First Car#:
<input type="text" name="car">
Second Car#:
<input type="text" name="car">
```


Revision no./Date:00/Oct.09 2008

Implicit Objects

- JSP Code

```
Car is : ${param.car}
 <%--Will display the first car only--%>
First car is : ${paramValues.car[0]}
Second car is: ${paramValues.car[1]}
```


Revision no./Date:00/Oct.09 2008

EL Implicit objects

- Use `requestScope` to get request attributes, not request properties.
  - If we need to print name of a person and we really don't care what scope the person is in, then just `${person.name}` is sufficient.
  - If we are concerned about a potential naming conflict, we can be explicit about which person we want by saying `${requestScope.person.name}`.

EL Implicit objects

- What if attribute is put like this??

```
request.setAttribute
("foo.person", p);
```

- Trouble if we say

```
${foo.person.name}
```

- To get the proper output

```
${requestScope["foo.person"].name}
```


Revision no./Date:00/Oct.09 2008

EL Implicit objects

- Printing the value of a context init parameter

  In web.xml

```
<context-param>
    <param-name>myEmail</param-name>
    <param-name>john@yahoo.com</param-name>
</context-param>
```

- With scripting

```
<%=application.getInitParameter("myEmail")%>
```

- With EL

```
${initParam.myEmail}
```


Revision no./Date:00/Oct.09 2008

JSTL

- JSTL grew out of the realization that with many developers creating tag libraries, many actions would be duplicated among the libraries.
- Because these libraries were developed separately, the duplicated actions would probably have different names, syntaxes and behaviors.
- The JSTL standardizes number of actions.

JSTL

- The JSTL tags have been divided into four categories. These categories are
  - Core actions
  - Internationalization(i18n) capable formatting
  - Relational database access (SQL)
  - XML processing
- The core action provides tag handlers for manipulating variables, dealing with errors, performing tests and conditional behavior, executing loops and iterations

Core actions- General purpose

Tag	Description
`<c:out value="" default="">`	Sends the value to the response stream. We can specify an optional default value so that if the **value** attribute is set with an EL expression and the expression is null, the **default** value will be output
`<c:set var="" value="">`	Sets the JSP-scoped variable identified by **var** to the given **value**
`<c:set target="" property="" value="">`	Sets the property of the given JavaBean or Map object to the given value.

Core actions- conditional

Tag	Description
`<c:if test="" var="">`	Used like standard Java if block. The **var** attribute is optional; if present, the result of the **test** is assigned to the variable identified by the var. If the test expression evaluates to true, the tag is evaluated; if false it is not
`<c:choose>,<c:when test="", <c:otherwise>`	The analog to Java if..elseif..else block. The **<c:choose>** action starts and ends the block. The test in each **<c:when test="">** tag is evaluated; the first test that evaluates to true causes that tag to be evaluated. If no **<c:when>** action evaluates to true, the **<c:otherwise>** tag is evaluated.
	21

Core actions- Iterator

Tag	Description
`<c:forEach var="" items="">`	Iterates over each item in the collection identified by **items**. Each item can be reference by **var**. When items is a Map, the value of the item is referenced by the var value. Items can be of type array, Collection, Map, comma delimited String
`<c:forEach var="" begin="" end="" step="">`	The tag for a for loop. The **step** attribute is optional
`<c:forTokens items="" delims="">`	Iterates over the tokens in the **items** string

Core actions- Miscellaneous

Tag	Description
<c:import url=“”> <c:import url=“” var=“”>	Adds the content from the value of **url** attribute to the current page, at request time; and if required you can set the url in a variable using **var** attribute.
<c:url value=“”>	Provides Hyperlink with URL rewriting capabilities. Value attribute has a value of relative URL.

Formatting Actions

Tag	Description
<fmt:formatDate value="date" [type="{time|date|both}"] [dateStyle="{default|short|medium|long|full}" [timeStyle="{default|short|medium|long|full}"] [pattern="customPattern"] [timeZone="timeZone"] [var="varName"] [scope="page|request|sessionapplication"] >	Only the **value** attribute is required. The other attributes define how to format the data. The **pattern** attribute can contain a custom pattern for formatting the date string

Revision no./Date:00/Oct.09 2008

SQL Actions

Tag	Description
<sql:query var="" dataSource="" > SQL Command </sql:query>	Queries the DB given by the **dataSource** attribute. The query that is performed is given in the body of the tag. The results of the query can be accessed by **var.rows.** The **dataSource** attribute uses the JDBC URL to access the DB
<sql:setDataSource dataSource="" var="" scope="" />	Set the datasource. Where **dataSource** attribute will be in the form of [JDBC URL],[driverName],[user name],[password]. **var** attribute is a reference to data source for forward referencing. **scope** decides the scope of a **var**

XML Actions

Tag	Description
`<x:parse xml="" var=""/>`	Builds a DOM tree based on the source specified using **xml** attribute and stores it into **var**.

`<x:forEach select="" var="">`
`----Some Commands----`
`</x:forEach>`
	Loops through a set of nodes referenced by **var** and fetched by XPath expression using **select**.

`<x:out select=""/>`	Fetches the value of the node based on XPath expression specified using **select**.
`<x:if select=""/>`	Used to test condition specifies using **select** attribute and generate filtered output.
`<x:transform xml="" xslt=""/>`	Converts an xml source file specified in the **xml** to HTML using a stylesheet specified in the **xslt**.