Desarrollo de Aplicaciones en Red

José Rafael Rojano Cáceres
http://www.uv.mx/rrojano

EL

Expression Language

Write the code in something else, just let EL call it.

EL (1)

- *EL stand for Expression Language which is part of JSP 2.0 specification*
- *The idea is to define your own XML tags to be used with JSP.*
  - **Our tags are instance of XML, and**
  - **Our tags are member of a namespace**
- *For example*
  - **<name:myTag>whatever this tag does...</name:myTag >**
- *In EL there are features of XML for that is case sensitive.*
- *A collection of tags that provide similar functionality or that logically collaborate with each other is called a tag library.*

EL (2)

- *A java class which is the tag handler provides the functionality to the tag in the JSP.*
- *Custom tag let's write code without scripting.*
- *The standard tags like <jsp:useBean> let's minimize scripting, but we need to do a lot of scripting for something.*
- *For example you can use EL in situation like this:*
  - **${param.variable}**
    - Param is an implicit object like request in the JSP
- *To define the behavior of tag we create a TLD file which is a XML file that map tag to the java class*

Implicit objects in EL

- *Mapping for scope for objects*
  - **pageScope:** Maps page-scoped variable names to their values
  - **requestScope:** Maps request-scoped variable names to their values
  - **sessionScope:** Maps session-scoped variable names to their values
  - **applicationScope:** Maps application-scoped variable names to their values
- *Mapping in request parameter*
  - **param:** Maps a request parameter name to a single value
  - **paramValues:** Maps a request parameter name to an array of values
- *Mapping from pageContext object*
  - **pageContext**

Literals

- **Boolean**: true and false
- **Integer**: as in Java
- **Floating point**: as in Java
- **String**: with single and double quotes; " is escaped as \", ' is escaped as \', and \ is escaped as \\.
- **Null**: null

Operators

- **In addition to the operators** . and [] we have:
- **Arithmetic**: +, - (binary), *, / and div, % and mod, - (unary)
- **Logical**: and, &&, or, ||, not, !
- **Relational**: ==, eq, !=, ne, <, lt, >, gt, <=, ge, >=, le. Comparisons can be made against other values, or against boolean, string, integer, or floating point literals.
- **Empty**: The empty operator is a prefix operation that can be used to determine whether a value is null or empty.
- **Conditional**: A ? B : C. Evaluate B or C, depending on the result of the evaluation of A.

7

Reserved words

- and eq
- gt true instanceof
- or ne
- le false empty
- not lt ge
- null div mod

8

Examples

EL Expression	Result
${1 > (4/2)}	false
${4.0 >= 3}	true
${100.0 == 100}	true
${(10*10) ne 100}	false
${'a' < 'b'}	true
${'hip' gt 'hit'}	false
${4 > 3}	true
${1.2E4 + 1.4}	12001.4
${3 div 4}	0.75
${10 mod 4}	2
${empty param.Add}	True if the request parameter named Add is null or an empty string
${pageContext.request.contextPath}	The context path
${sessionScope.cart.numberOfItems}	The value of the numberOfItems property of the session-scoped attribute named cart
${param['mycom.productId']}	The value of the request parameter named mycom.productId
${header["host"]}	The host
${departments[deptName]}	The value of the entry named deptName in the departments map

9

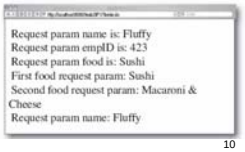
Example

- ***HTML page***

```
<form action="TestBean.jsp">
  Name: <input type="text" name="name">
  ID#: <input type="text" name="empID">

  First food: <input type="text" name="food">
  Second food: <input type="text" name="food">

  <input type="submit">
</form>
```

- ***JSP page***

```
Request param name is: ${param.name} <br>
Request param empID is: ${param.empID} <br>
Request param food is: ${param.food} <br>

First food request param: ${paramValues.food[0]} <br>
Second food request param: ${paramValues.food[1]} <br>

Request param name: ${paramValues.name[0]}
```



10

Functions (1)

- ***To define a function you program it as a public static method in a public class.***
- ***Example: The mypkg.Myclass is a class that defines a function that tests the equality of two Strings as follows:***

```
package mypkg;
public class MyClass {
   ...
   public static boolean equals(String l1, String l2)
   {
        return l1.equals(l2);
   }
}
```

11

Functions (2)

- ***Each function must be mapped into a TLD file.***

```
<function>
    <name>equals</name>
    <function-class>
      mypkg.MyClass
    </function-class>
    <function-signature>
        boolean equals( java.lang.String, java.lang.String )
    </function-signature>
</function>
```

12

TLD

- *A Tag Library Descriptor (TLD) provides a mapping between a plain old java class and a JSP.*
- *When you define a class it can be invoked through EL just defining it in a TLD file.*
- *A TLD file is saved in the WEB-INF directory.*
- *In order to invoke the tag you must set the namespace into the taglib directive:*
  - <%@ taglib prefix="alias" uri="Funciones"%>
- *Accessing the function from EL expressions is written something like:*
  - ${alias:name()}

Example Class

```
package Clases;
public class hacerAlgo
{
  public static int valor()
  {
     return (int)((Math.random() * 6) + 1);
  }
}
```


Example TLD

```
<?xml version="1.0" encoding="utf-8" ?>
<taglib 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"
        version="2.0">
  <tlib-version>1.2</tlib-version>
  <short-name>Funciones</short-name>          (1)
  <uri>Funciones</uri>
  <function>
    <name>calcular</name>                     (2)
    <function-class>Clases.hacerAlgo</function-class>
    <function-signature>int valor()</function-signature>   (3)
  </function>
</taglib>
```


Example JSP invocation

```
<%@ taglib prefix="alias" uri="Funciones"%>   (1)
  <html>
    <body>
      ${alias:calcular()}
    </body>
  </html>
                                              (2)
```


Taglib

- *To declare that a JSP page will use tags defined in a tag library, you include a taglib directive in the page before any custom tag from that tag library is used.*

<%@ taglib prefix="tt" [tagdir=/WEB-INF/tags/dir | uri=URI] %>

JSTL

Java Server Page Standard Tag Library

JSTL

- ***The JSTL encapsulates core functionality common to many JSP applications.***
- ***JSTL has tags such as iterators and conditionals, for manipulating XML, databases access, etc.***
- ***A complete list of tags can be found in:***
  - http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html

19

Namespaces

- JSTL includes a wide variety of tags that fit into discrete functional areas.
- This functionality is organized in the following namespaces:
  - *Core*: http://java.sun.com/jsp/jstl/core
  - *XML*: http://java.sun.com/jsp/jstl/xml
  - *Internationalization*: http://java.sun.com/jsp/jstl/fmt
  - *SQL*: http://java.sun.com/jsp/jstl/sql
  - *Functions*: http://java.sun.com/jsp/jstl/functions

20

Functionality into namespaces

Area	Subfunction	Prefix
Core	Variable support Flow control URL management Miscellaneous	c
XML	Core Flow control Transformation	x
I18n	Locale Message formatting Number and date formatting	fmt
Database	SQL	sql
Functions	Collection length String manipulation	fn

21

Terminology

- *custom tags* are a mechanism for encapsulating other types of dynamic functionality.
- Custom tags are distributed in a *tag library*, which defines a set of related custom tags and contains the objects that implement the tags.
- The object that implements a custom tag is called a *tag handler*.
- There are two types of tags: simple and classic.
  - *Simple* tag handlers can be used only for tags that do not use scripting elements in attribute values or the tag body.
  - *Classic* tag handlers must be used if scripting elements are required.

22

Tags

- ***Tags with and without body***
  - <tt:tag> *body* </tt:tag>
  - <tt:tag /> or <tt:tag></tt:tag>
  - In case a tag has body it would be like:

```
<c:if test="${param.Clear}">
    <font color="#ff0000" size="+2"><strong>
      You just cleared your shopping cart!
      </strong>
    </font>
</c:if>
```

- ***Tag also can contain attributes***
  - **Attributes are evaluated by the container before the tag handler, for example:**

```
<sc:catalog bookDB ="${bookDB}" color="#cccccc">
```

23

Examples

- ***For the context of core we have:***

```
<c:if test="${empty param.l}">
  escribe un valor para login
</c:if>
<c:if test="${empty param.p}">
  escribe un valor password
</c:if>
<c:if test="${alias:comprobar()}">
  si es valido
</c:if>
```

24

Creating your own tag

- *Define the rules in TLD file*
- *Create the handler*
- *Use the tag in you JSP page*

25

Define the TLD example

```
<?xml version="1.0" encoding="utf-8" ?>

<taglib 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"
      version="2.0">

 <tlib-version>1.2</tlib-version>
 <short-name>funciones</short-name>
 <uri>funciones</uri>

 <tag>
   <name>pruebas</name>
   <tag-class>Clases.prueba</tag-class>
   <body-content>empty</body-content>
   <attribute>
     <name>parametro</name>
     <required>true</required>
     <rtexprvalue>true</rtexprvalue>
   </attribute>
 </tag>
</taglib>
```

26

Create the handler

```
package Clases;

import java.io.IOException;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class prueba extends SimpleTagSupport
{
    private String parametro;

    public void doTag() throws JspException, IOException
    {
          getJspContext().getOut().write("<br> hola <strong>");
          getJspContext().getOut().write(parametro + "</strong><br>");
    }

    public void setParametro(String parametro)
    {
          this.parametro = parametro;
    }
}
```

27

Using the defined tag

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib prefix="rr" uri="WEB-INF/rr.tld"%>


    <c:if test="${alias:comprobar()}">
          si eres valido, <rr:pruebas parametro="${param.l}"></rr:pruebas>
    </c:if>
```

28

MVC

Model – View – Controller

29

A little bit of history

- **MVC is the paradigm behind Smalltalk-80.**
- **The user interface of Smalltalk was the precedence for Macintosh and Windows.**
- **MVC is one of the first design patterns. It was developed by Trygve Reenskaug in 1979.**
- **The MVC paradigm has been copied from Smalltalk-80 and used in other UI frameworks. The Java Swing UI and Struts, for example, are based on MVC.**
  - **in Struts, the View is represented by a set of JSP pages,**
  - **the Controller by a Struts servlet that delegates the real work to action classes**
  - **and the Model by application classes that typically interact with a database or some other permanent storage.**

30

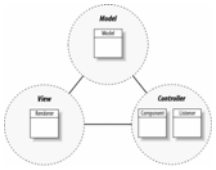
MVC (1)

- *MVC drives to put in layers different responsibilities from the application*
- *MVC defines how the user input, the processing and the answers is handled by an application:*
  - The view manages the graphical and/or textual output.
  - The controller interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate.
  - Finally, the model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).

31

MVC in JSF

- *The Model is represented by properties of application objects, e.g., a username property on an application object holding user information.*
- *JSF UI components declare what events they can fire, such as "value changed" and "button clicked" events, and external event listeners (representing the Controller) attached to the components handle these events.*
  - An event listener may set properties of the JSF components—for instance, adjust the set of rows shown in a table—or invoke backend code that processes the submitted data (say, verify that a credit card is valid or update a database).
- *A separate renderer class (representing the View) renders each JSF UI component, making it possible to render instances of the same component type in different ways (e.g., either as a button or a link, or using different markup languages) just by attaching different renderers.*



32

MVC

- *In the original MVC the Servlet controller talked to the model (java class with business logic), then it set an attribute in the request scope before forwarding to the jsp view.*
- *The jsp has to get the attribute from the request scope, and use it to render a response send back to client.*

33

A fast review about learned concepts

Java Server Faces

34

JSF

- *JavaServer Faces (JSF) simplifies development of sophisticated web application user interfaces, primarily by defining a user interface component model tied to a well-defined request processing lifecycle.*
  - Less code in the user interface
  - More Modular User Interface Code

35

A fast view (1)

- There is not any control flow structure or logic.
- The real logic is inside *<h:selectManyCheckbox>* and *<f:selectItem>*
- When the form is sent all the selection are saved in an application object defined by *#{cust.SeleccionComida}*.

```
-<h:form>
   <table>
   ...
    <tr>
     <td>Mi Comida Favorita:</td>
     <td>
      <h:selectManyCheckbox value="#{cust.SeleccionComida}">
       <f:selectItem itemValue="z" itemLabel="Chilaquiles" />
       <f:selectItem itemValue="a" itemLabel="Pizza" />
       <f:selectItem itemValue="p" itemLabel="Pasta" />
       <f:selectItem itemValue="c" itemLabel="China" />
      </h:selectManyCheckbox>
     </td>
    </tr>
   ...
   </table>
</h:form>
```

36

Example with

MyEclipse

37

- ***http://www.onjava.com/pub/a/onjava/2000/12/15/jsp_custom_tags.html***
- ***http://jakarta.apache.org/taglibs/tutorial.html#tag_library***
- ***http://www.onjava.com/pub/a/onjava/2004/02/11/jspcookbook.html***
- ***Java servlet and JSP cookbook***

38