EL (Expression Language)

Naveen Kumar K S

adith.naveen@gmail.com

http://naveenks.com

Scriptless JSP

- A JSP page without any scripting elements(scriplets, script declaration and expressions) is a script-less JSP.
- Scriptless JSP attempts to reduce the clutter of java code in the JSP.
- Also the page designers do not have the burden of learning java and also maintenance becomes easy.
- With standard JSP action tags we can achieve some script-less-ness can be achieved.
- But surely this is not enough. For instance how will you print request parameters in a script-less way or a bean’s property’s property?

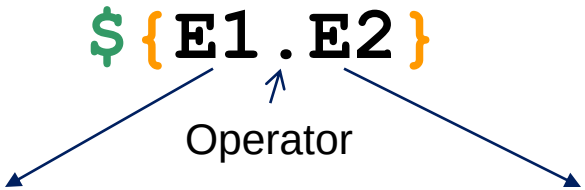
EL

- Expression language
- Primary feature of JSP technology from version 2.0
- EL expressions can be used:
  - In static text
    - The value of an el expression in static text is computed and inserted into the current output
  - In any standard or custom tag attribute that can accept an expression
    - If the static text appears in a tag body, the expression will not be evaluated if the body is declared to be tagdependent

We will see what this is in the custom tag section

EL Syntax

- EL Expressions are always within curly braces, and prefixed with a dollar sign



EL implicit object

OR

an attribute either in page, request, session or application scope (searched in that sequence)

OR

Map(java.util.Map)

If E1 is implicit object , then E2 is attribute

If E1 is attribute, then E2 is the property of the object.

If E1 is Map, then E2 is the key

Example: Using EL with bean

Using EL to print bean property of a bean

```
package a;
import java.io.Serializable;
public class Emp implements Serializable{
   private String empno;
   private Name name;
   public Emp(){ this("111", "Ravi", "Shankar"); }
   public Emp(String e,String f, String l){
   empno=e;
   name = new Name(f,l);}
   public void setEmpno(String empno){ this.empno=empno;}
   public void setName(Name n){ this.name=n; }
   public String getEmpno(){return empno;}
   public Name getName(){ return name;}
}
```

Another bean

To make it easier for the sake of example

```
package a;

public class Name {
private String fname;
private String lname;

public String getFname() {return fname;}
public void setFname(String fname) {this.fname =
fname;}
public String getLname() {return lname;}
public void setLname(String lname) {this.lname =
lname;}
Name(String f, String l){fname=f;lname=l;}
public Name(){}
}
```

JSP:

```
<body>
<jsp:useBean class="a.Emp" id="e"/>
<br>
Emp Name: Mr./Mrs./Miss ${e.name.fname} ${e.name.lname}
</body>
```

The expressions are evaluated from left to right. Each expression is evaluated to a String and then concatenated with any intervening text. The resulting String is then coerced to the attribute's expected type.

Displays

Emp Name: Mr./Mrs./Miss Ravi Shankar

EL Implicit Objects

- `pageScope`
- `requestScope`
- `sessionScope`
- `applicationScope`
- `param`
- `paramValues`
- `header`
- `headerValues`
- `cookie`
- `initParam`
- `pageContext` → Like JSP implicit object

Map objects

Note that these objects are not the same as JSP implicit object

Examples

- **`${cookie.uname.valu}`**
  - Returns the value of uname cookie
- **`${initParam.color}`**
  - Same as

    `<%=application.getInitParameter("color")%>`
- **`${pageScope.e}`** where e the **`a.Emp of last example`**

Some common errors

```
<jsp:useBean class="a.Name" id="n">
<jsp:setProperty name="n" property="fname"
value="Rahim"/>
<jsp:setProperty name="n" property="lname"
value="Raja"/>
</jsp:useBean>
<jsp:useBean class="a.Emp" id="e">
<jsp:setProperty name="e" property="name"
value="{pageScope.n}"/>
</jsp:useBean>
```

Because **`${pageScope.n}`** returns a string!

- Write an EL to print the session id.

${session.id} → WRONG, No 'session' in EL implicit object

${pageContext.session.id} → CORRECT

- **session.setAttribute("flower", "rose");**

…

${pageContext.session.flower} → WRONG

${flower} → WRONG

${sessionScope.size} → CORRECT

Using “[ ]” operator

- When the variable is on the left side of the [], it can be

```
${E1["E2"]}
```

  - If **E1** is **Map** → **E2** is **key**
  - If is **bean** → **E2** is **property**
  - **List** → **E2** is **index**
  - array → **E2** is **index**

In case of **List** and arrays , **String** index is forced(converted) to an **int**.

Example using [] and .

```
protected void doGet(…){
java.util.Map carmap=new java.util.HashMap();
        carmap.put("Large","Mercedes");
        carmap.put("Medium","Getz");
        carmap.put("Small","Alto");
String[] cartype={"Large","Medium","Small"};
request.setAttribute("carmap",carmap);
request.setAttribute("cartype",cartype);
request.setAttribute("size","Medium");
request.getRequestDispatcher("i2.jsp").forward(request
   ,response);
}
```

In JSP

```
<body>
    My car is: ${carmap.Large}
    <br> or  <br>
    My car is: ${carmap["Large"]}
<br>
```

Both print **Mercedes**

prints **Large**

```
Car type : ${cartype[0]} or Car type : ${cartype["1"]}
<br>
   My car is: ${carmap[size]}
<br>

My car is:
${carmap[cartype[0]]}
  </body>
```

prints **Medium**

prints **Getz**. Without double quotes inside the square brackets the string value is treated as attribute. The attribute evaluates and its value is substituted.

prints **Mercedes**

Getting Form Params

- **`${param.eno}`**
  - Gets the request parameter matching ‘eno’
- **`${paramValues.hobbies}`**
  - Gets the first value from the multiple value request attribute
- **`${paramValues.hobbies[0]}`**
  - Same as the above
- You could also say **`${param.eno[0]}`** for a single value parameter.

Arithmetic Operators

- `+`
- `-`
- `*`
- `/,div`
- `% ,mod`

Example:

`${42/0}` →gives INFINITY as output and not Exception

`${42%0}`→ gives an Exception

In arithmetic expressions, EL treats the unknown variable (null value) as Zero(0).

Relational and Logical Operators

Relational

- `==, eq`
- `!=, ne`
- `< , lt`
- `> , gt`
- `<=, le`
- `>=, ge`

Logical

- `&& , and  AND`
- `|| , or  OR`
- `!, not NOT`

In Logical expressions (using logical operators and/or relational operator, EL treats the unknown variable (null value) as false.

EL reserved words

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

`${empty obj}` returns `true` if obj is `null` or empty.

EL Functions

- The expression language allows a function to be invoked in an expression provided the function is defined as **`public static`** method in a java class.
- For this to happen the **`public static`** method must be specified in a special file with extension TLD.
- The TLD is a tag library descriptor file that is used when custom tags are created.

Disabling EL in JSP and DD

- In a jsp page we can disable the EL using the is **ELIgnored** page directive attribute

```
<%@ page isELIgnored= “true” %>
```

- In the DD we do it as

```
<web-app>
    <jsp-config>
  <jsp-property-group>
    <url-pattern>*.jsp </url-pattern>
    <el-ignored>true</el-ignored>
   </jsp-property-group>
  </jsp-config>
</web-app>
```

Ensuring that JSPs are scriptless

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