Les TagLibs

P.Mathieu

Université de Lille

LP DA2I Lille
http://www.iut-a.univ-lille.fr
prenom.nom@univ-lille.fr

7 janvier 2020

Les EL expressions

Avantages

Université de Lille

- Introduites par SUN depuis JSP 2.0 (version J2EE 1.4)
- Eloigner encore plus le code Java des JSP
- Elles apportent plusieurs avantages :
  - Une meilleure lisibilité (s'affranchir de la syntaxe Java)
  - Evitent les cast
  - Unifient l'accès aux Map et aux objets
  - Permettent les accès en cascade (Maps de Maps ou d'objets)
  - Gestion par défaut des exceptions (null, arrayOutOfBounds)

Les EL expressions

Principe

Université de Lille

- Une EL expressions est de la forme `${expression}`
- Elle est remplacée à l'exécution par le résultat de son évaluation.

L'expression peut être composée d'opérateurs qui s'appuient sur :

- un type de base java (int,long,double, ... String)
- Une Map
- un objet implicite prédéfini
- un attribut d'un Bean
- une fonction définie par l'utilisateur

Les EL expressions

Les types de base

```
<% int age=25;
pageContext.setAttribute("age", new Integer(age));
%>
Le candidat est ${(age >=18) ? "majeur" : "mineur"}. <br/>
J'ai acheté ${6 div 2} pommes.      <br/>
J'ai acheté ${2003 % 8} pommes.     <br/>
Il me reste ${23/54} euros.         <br/>
cette chaine est vide ? ${empty ""}  <br/>
cette variable est non affectée ou déclarée ? ${empty x} <b
```

Attention : Les EL ne traitent que des objets. les types primitifs sont donc automatiquement transformés via les wrappers correspondants

Les EL expressions

Accès aux tableaux et listes

```
<%
  String[] animaux = {"chien", "chat", "souris"};
%>

${animaux[2]} <br />
${animaux['2']} <br />
${animaux["2"]} <br />
```

Le même code fonctionne avec une `List`

Les EL expressions

Accès aux Maps et aux objets

Uniformatisation des accès aux Maps et aux objets
Utilisation d'une notation pointée, éventuellement en cascade

```
${map.cle}
ou
${objet.attribut}
```

Les notations `${map["name"]}` ou `${map['name']}` sont aussi autorisées. (Cette notation avec crochets [] fonctionne aussi avec les List et les Array).

Les EL expressions

Des maps implicites sont créées !

pageContext : Accès à l'objet PageContext de la page JSP.

pageScope : Map d'accès aux différents attributs de 'page'.

requestScope : Map d'accès aux différents attributs de 'request'.

sessionScope : Map d'accès aux différents attributs de 'session'.

applicationScope : Map d'accès aux attributs de 'application'.

param : Map d'accès aux paramètres de la requête

paramValues : idem mais sous forme de tableau de String.

header : Map d'accès au Header HTTP

headerValues : idem mais sous forme de tableau de String.

cookie : Map d'accès aux différents Cookies.

initParam : Map d'accès aux init-params du web.xml

Les EL expressions

Accès aux maps implicites

Les objets implicites sont tous de type Map ! !

`${param.nom}` affiche la valeur du paramètre "nom".
`${initParam.driver}` affiche la valeur du paramètre "driver" du web.xml.
`${header.host}` affiche le nom du serveur host

Les EL expressions

Accès aux objets

Les accesseurs sont construits par réflexivité pour accéder aux objets (pour accéder à la propriété `nom`, la méthode `getNom()` est appelée).

```
<%!
public class Personne
{
private String nom="paul";
public String getNom(){return nom;}
}
%>

<% request.setAttribute("p",new Personne()); %>

${requestScope.p.nom}
```


Les EL expressions

Attention !

Si la variable `nom` n'est pas définie ...

```
${param.nom} -> ""

<%= request.getParameter("nom") %> -> null
```


Les EL expressions

Recherche automatique

Lors de l'évaluation d'un terme, si celui ci n'est ni un type primaire, ni un objet implicite, le conteneur JSP recherchera alors un attribut du même nom dans les différents scopes de l'application.

Ordre : page, request, session, application

les `*Scope` sont donc facultatifs (au prix d'un léger ralentissement)

```
<%
Map notes=new HashMap();
notes.put("paul","5");
notes.put("pierre","15");
notes.put("jean","25");
notes.put("jacques","35");
session.setAttribute("notes",notes);
%>
```

`${sessionScope.notes.paul}` s'écrit plus simplement `${notes.paul}`

Définir ses propres Tags

Etendre les tags

Université de Lille

JSP permet de créer des nouveaux tags associés à des méthodes statiques publiques de n'importe quelle classe Java

Technique :

1. Créer la classe Java qui définit les actions à réaliser par le nouveau Tag
2. Créer le fichier de description (TLD) qui établit le lien entre ce tag et la classe/methode précédemment décrite

Définir ses propres Tags

Creation de la classe Java

Université de Lille

On définit ses méthodes statiques dans un objet ...

```
package tools;

import java.util.Locale;

public class MesFonctions {
    public static String reverse(String text) {
        return new StringBuilder(text).reverse().toString();
    }

    public static String upper(String text) {
        return text.toUpperCase(Locale.ENGLISH);
    }
    ....
}
```


Définir ses propres Tags

Déclaration des nouveaux tags

Université de Lille

Fichier *Tag Lib Descriptor* (TLD) à placer dans `WEB_INF`

Ex `MesFonctions.tld`

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
        PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
        "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>

  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>simple</short-name>
  <uri>http://localhost:8080</uri>
  <description>
    Un exemple trival
  </description>

    <function>
        <name>reverse</name>
        <function-class>tools.MesFonctions</function-class>
        <function-signature>String reverse(java.lang.String)</function-signature>
    </function>
```

Définir ses propres Tags

utilisation des nouveaux tags

Université de Lille

Il suffit ensuite de déclarer la TLD pour pouvoir utiliser les nouveaux tags en notation EL

```
<%@page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="perso" uri="/WEB_INF/MesFonctions.tld"%>

${perso:reverse('alibaba')}

${perso:upper('alibaba')}
```


JSTL

Principe

- JSTL (Jsp Standard Tag Library)
- Tags XML codant les actions les plus courantes
- Evite le Java dans la JSP
- Librairie standard initialement réalisée par SUN
- Actuellement JSTL1.2

JSTL

5 sous-librairies

Type	URI	Prefixe	Exemple
Core	http://java.sun.com/jsp/jstl/core	c	<c:tagname ...>
XML	http://java.sun.com/jsp/jstl/xml	x	<x:tagname ...>
I18N	http://java.sun.com/jsp/jstl/fmt	fmt	<fmt:tagname ...>
DB access	http://java.sun.com/jsp/jstl/sql	sql	<sql:tagname ...>
Functions	http://java.sun.com/jsp/jstl/functions	fn	<fn:functionName(...)

Doc : http://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/

JSTL

Installation

La JSTL dépend du conteneur. Pour Tomcat il est préférable de la télécharger chez Apache.

- Télécharger la JSTL API sous forme de jar
- `https://jstl.java.net/` (implémentation de reference)
- `http://tomcat.apache.org/taglibs/` (implémentation Apache)
  Actuellement un seul fichier : `jstl-1.2.jar`
- Placez ce fichier dans lib ou WEB-INF/lib (ou via Maven)

Dans chaque page utilisant les tags, indiquer :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
```


JSTL

La librairie Core

Fonction	Tags
Manip variables	remove set
Controle du flux	choose - when - otherwise forEach forTokens if
Gestion d'URL	import (+ param) redirect (+ param) url (+ param)
Divers	catch out

JSTL

Exemple 1 utilisant Core

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>

<html>
Si on passe un nom en parametre, le voici : ${nom}

<p/>

et via la JSTL <c:out value="${param.nom}" />
</html>
```

`c:out` encode les caractères HTML, contrairement à `${param.nom}`.
il permet donc d'éviter facilement le cross-site scripting
(à tester avec `?nom=<h1>paul</h1>`)

JSTL

Exemple 2 utilisant Core

```
<%@ taglib uri='http://java.sun.com/jsp/jstl/core' prefix='c' %>

<h2>Lister les entetes</h2>

<c:forEach var='hv' items='${headerValues}'>
   <ul>
     <li><b><c:out value='${hv.key}'/></b>:<br>

          <c:forEach var='value' items='${hv.value}'>
                <c:out value='${value}'/>
          </c:forEach>
   </ul>
</c:forEach>
```

JSTL

La librairie SQL

Fonction	Tags
Connexion	setDataSource
SQL	query (+dateParam, +param) transaction update (+dateParam, +param)

JSTL

Exemple Core+SQL

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>

<sql:setDataSource var="conn" driver="org.postgresql.Driver"
url="jdbc:postgresql://psqlserv/da2i" user="dupont" password="moi" />

<html>
<body>

<sql:query var = "rs" dataSource="${conn}">
select * from personnes
</sql:query>

<table border=1>
<c:forEach var="ligne" items="${rs.rows}">
<tr>
<td><c:out value="${ligne.pno}"/></td>
<td><c:out value="${ligne.nom}"/></td>
<td><c:out value="${ligne.prenom}"/></td>
<td><c:out value="${ligne.adresse}"/></td>
</tr>
</c:forEach>
</table>
```