Pour aller plus loin - Custom tags

D.Ledanseur
Année 2010/2011

Table des matières

1 Que sont-ils ?

Les customs tags (ou en français, balises personnalisées) sont des balises possédant une syntaxe particulière, définie par le développeur, qui permettent de réduire considérablement la volumétrie et la redondance de code dans les pages JSP.

Pour bien comprendre l'utilité des customs tags, il suffit de prendre l'exemple récurrent parmi les customs tags : la grille. A travers une application Web homogène, il va être nécessaire d'afficher à plusieurs endroits une grille contenant des données.

Supposons donc que le tableau objs contienne un ensemble d'objets possédant une méthode getKey et getValue pour obtenir nos données, et que nous souhaitons afficher l'ensemble de ces combinaisons dans un tableau à deux colonnes.

En programmation JSP standard, l'affichage d'une telle table pourrait donner quelque chose comme ça :

```
<table>
	<thead>
		<tr><td>Key</td><td>Value</td></tr>
	</thead>
	<tbody>
<%
	for (int i=0;i<objs.length;i++) {
%>
	<tr>
		<td><%=objs[i].getKey() %></td>
		<td><%=objs[i].getValue() %></td>
	</tr>
<%
	}

%>
	</tbody>
</table>
```

Deux problèmes peuvent être mis en valeur par cet exemple. Premièrement, avec cette syntaxe, le code devient rapidement difficile à lire, encore qu'il s'agisse ici d'un exemple simple. Le mélange entre code Java et HTML, rend difficile l'isolation entre les deux domaines, ce qui donne un code fonctionnel, mais difficile à maintenir.

L'autre problème est la redondance du code. Il n'est pas très pratique de devoir à chaque fois définir les entêtes de ligne et colonne, de devoir créer une boucle pour afficher nos objets.

Grâce aux customs tags, il est possible d'avoir une syntaxe telle que celle-ci :

```
<ct:table objets="${objs}>">
	<ct:column title="Key" value="key"/>
	<ct:column title="Value" value="value"/>
</ct:table>
```

Bien entendu, il n'y a aucune magie la dessous, et la complexité demandée pour générer la table HTML est cachée par l'implémentation du custom tag. L'implémentation de celui-ci en sera d'ailleurs complexifiée, puisqu'il devra s'adapter à des cas génériques. Mais on peut déjà noter que les pages JSP qui vont utiliser notre custom tag vont réellement

gagner en simplicité et en clarté.

Avant de voir comment réaliser notre custom tags, décryptons tout d'abord les quelques lignes que nous avons écrites dans la page JSP.

Nous avons ici un cas relativement complexe de custom tags, puisque en réalité, nous en avons deux imbriqués. En effet, les tags table et column sont en réalités deux customs tags distincts. Le préfixe devant le tag (« ct ») identifie la *bibliothèque* de tag, c'est à dire un regroupement de custom tags.

L'exemple ci dessus ne le montre pas, mais la bibliothèque doit être déclaré dans la page Jsp, de la manière suivante (ici dans le cas d'utilisation de la JSTL, cf chapitre ci-après):

Page JSP

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

La directive `<%@ taglib` permet de déclarer l'utilisation d'une bibliothèque de tags dans la page JSP. La valeur de l'attribut prefix permet de définir comment appeler les tags en question dans le reste de la page JSP. L'attribut uri doit faire référence à la déclaration d'une taglib dans le fichier web.xml, et doit donc correspondre à l'élément taglib-uri de notre exemple.

Par convention, l'URI utilisé par les customs tags est une URL permettant aux utilisateurs de votre bibliothèque de tags de trouver de la documentation ainsi que les fichiers nécessaires à l'utilisation de la bibliothèque de tags. Ceci n'est cependant qu'une convention et pourrait être n'importe quelle chaîne permettant d'identifier de manière unique votre bibliothèque, mais qui respecte les conventions des URI.

2 JSTL (JavaServer Pages Standard Tag Library)

Nous avons vu la manière dont le développeur peut utiliser des bibliothèques de tags pour accélérer le processus de développement de son application. Bien entendu, il existe des multitudes de bibliothèques déjà existantes qui permet de simplifier considérablement la vie du développeur.

Il en est cependant une qu'il convient de présenter particulièrement, puisqu'elle est normée et est apparue conjointement à la norme JSP 2.0.

En effet, la bibliothèque de tags JSTL fournit un ensemble de tags standard qui sont utilisés par un grand nombre d'applications.

L'un des buts de la JSTL, conjointement avec les expressions EL et les frameworks MVC, est de faire disparaître l'usage des scriptlets, c'est à dire les tags JSP standard (<%..%>), qui comme nous l'avons vu, rendent les pages JSP difficiles à lire et maintenir.

La JSTL est en réalité regroupée en plusieurs bibliothèques que le développeur peut inclure dans son application suivant ses besoins. En voici la liste, accompagnée de l'URI à déclarer dans les pages JSP lors de leur utilisation :

- *Core* (http://java.sun.com/jsp/jstl/core) : ce sont les principaux tags de la JSTL, ils contiennent des fonctionnalités diverses comme le contrôle de flux (conditions, itérations), la manipulation de variable et d'URL, entre autres choses.
- *XML* (http://java.sun.com/jsp/jstl/xml) : ces tags permettent la manipulation de document XML directement dans la page JSP.
- *Internationalization* (http://java.sun.com/jsp/jstl/fmt) : ces tags permettent d'internationaliser, c'est à dire de « traduire » la page en plusieurs langues. Ceci s'applique aux traductions littérales bien entendu, mais également au formatage des dates et nombres en fonction de la locale (comprendre localisation) de l'utilisateur.
- *SQL* (http://java.sun.com/jsp/jstl/sql) : ces tags permettent d'effectuer des requêtes à une base de données depuis la page JSP. Ces tags sont utilisés dans les petites applications de quelques pages, ou un framework dédié serait trop contraignant. Ils deviennent cependant rapidement limités dans le cadre d'applications plus importantes.
- *Functions*: http://java.sun.com/jsp/jstl/functions) : cette dernière bibliothèques contient les fonctions permettant de manipuler les chaînes de caractères, ainsi qu'une fonction permettant de connaître la taille d'une collection.

Il serait bien trop long de décrire dans ce document l'ensemble des tags de la JSTL, et ce n'est de toute façon pas son objectif. Plusieurs de ces tags seront utilisées à travers les différents documents composant le cours, et c'est pourquoi, je vous invite à consulter les spécifications correspondantes sur le site de SUN pour de plus amples informations :

http://jcp.org/aboutJava/communityprocess/mrel/jsr052/index2.html.

3 Ecrire ses propres custom tags

Nous avons vu dans la partie précédente comment utiliser un custom tag déjà écrit. Nous allons voir maintenant comment écrire nos propres customs tags.

Comme dit dans la partie précédente, deux tags sont nécessaires pour parvenir à réaliser l'exemple précédent : les tags table et column.

Commençons par nous interroger sur ce que feront ces deux tags:

- Le tag table doit générer l'en-tête de table, puis pour chaque ligne du tableau d'objet, appeler le tag column pour la génération. Enfin, le tag table génère le pied du tableau.
- Pour chaque objet qui lui est transmis, le tag column doit générer une « ligne » html, qui contient soit le titre de la colonne (première génération) soit la valeur d'une des propriétés d'un objet (générations suivantes).
- Pour permettre la communication entre les deux tags, nous allons utiliser le contexte Request et y stocker nos variables. Deux variables seront nécessaires:
  - __ctTable__titleGen : un booléen permettant de déterminer si nous sommes en train de générer le titre ou le corps de la table
  - __ctTable__currentObject : l'objet qui doit être utilisé pour afficher les valeurs de la table, c'est à dire l'objet en cours de notre tableau d'objets.

Pourquoi utiliser le contexte Request, alors que le contexte page aurait suffit ? Les tags-file étant des pages JSP à part entière, elle dispose de leur propre context Page. Cependant, en choisissant correctement le nom des variables pour communiquer entre les deux tags, on peut s'assurer d'une relative sécurité quant à un éventuel conflit de variable...

Pour cela, nous allons créer un fichier column.tag contenant le code suivant:

Fichier column.tag

```
<%@ tag display-name="column" description="Génère une ligne d'une table HTML"
%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ attribute name="title" description="Le titre de la column" required="false"
type="String" %>
<%@ attribute name="value" description="La propriété de l'objet à afficher"
required="true" type="String" %>

<td>
	<c:choose>
		<c:when test="${__ctTable__titleGen==true}">
			${title}
		</c:when>
		<c:otherwise>
			${__ctTable__currentObject[value]}
		</c:otherwise>
	</c:choose>
```

```
</td>
```

La première directive `tag` permet de donner quelques informations sur le tag en question : son nom et sa description.

Il est important de noter que ce tag fait appel à la JSTL par la directive `taglib`. En effet, il est tout à fait possible dans un tag de faire appel à une autre bibliothèque de tags.

Les deux directives `attribute` permettent de définir un attribut pour l'utilisation du tag. Ces déclarations correspondent aux attributs que le développeur va renseigner dans la page JSP qui utilise nos custom tags.

Fichier table.tag

```
<%@ tag display-name="table" description="Génère une table HTML" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ attribute name="objets" description="Le titre de la column" required="true"
type="java.lang.Object[]" %>

<table>
		<c:set var="__ctTable__titleGen" value="true" scope="request"/>
		<thead>
			<tr><jsp:doBody/></tr>
		</thead>

		<c:set var="__ctTable__titleGen" scope="request" value="false"/>
		<tbody>
			<c:forEach items="${objets}" var="obj">
				<c:set var="__ctTable__currentObject" value="${obj}"
scope="request"/>
					<tr><jsp:doBody/></tr>
			</c:forEach>
		</tbody>

</table>
```

Pour générer l'en-tête du tableau, on définit la propriété `__ctTable__titleGen` de la portée Request à true, puis on demande au moteur JSP d'évaluer le contenu du tag par l'intermédiaire de `<jsp:doBody/>`. On définit en suite la propriété `__ctTable__titleGen` à false puis, pour chaque objet du tableau passé en argument, on définit la variable `__ctTable__currentObject`, puis on fait appel à chaque fois à une nouvelle évaluation du corps du tag.

Ainsi, les tags column sont évalués une fois pour les titres, et une fois pour chaque objet du tableau d'objet.

Depuis la version JSP 2.1., les tags-files permettent aux développeurs d'accélérer le développement des customs tags, et nous ferons donc l'impasse sur les autres façons de faire, un peu plus compliquées. Il reste cependant des situations où leur utilisation reste nécessaire. Vous pouvez vous référer au tutoriel officiel de

Sun pour de plus amples informations :
http://java.sun.com/javaee/5/docs/tutorial/doc/bnalj.html.

4 TLD et packaging

4.1 TLD

Le TLD (Tag Library Descirptor) est un fichier xml qui permet de décrire les tags et fonctions disponible dans la bibliothèque de tags.

Ce fichier prend la forme suivante, avec ici la déclaration d'un tag-file et d'une fonction (cf. chapitre suivant).

```
<?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>1.0</tlib-version>
    <short-name>fns</short-name>
    <uri>http://organisation.com/taglib/funcs.tld</uri>

      <tag-file>
            <name>link</name>
            <path>/WEB-INF/tags/testFn/link.tag</path>
      </tag-file>

      <function>
    <name>toUpper</name>
    <function-class>test.JSPFuncs</function-class>
    <function-signature>
        java.lang.String toUpper(java.lang.String)
    </function-signature>
</function>

</taglib>
```

Ce TLD est normalement obligatoire pour chaque création de custom tags. Il existe cependant certains cas particuliers ou le serveur d'application génère ce fichier en fonction de la liste des tags qu'il peut trouver dans le projet.

Depuis JSP 2.0, il n'est plus nécessaire de déclarer le fichier TLD dans lefichier Web.xml. En effet, le serveur d'application doit chercher les TLD disponibles dans les répertoires suivants :

- Dans le répertoire /WEB-INF
- Dans les sous-répertoires de /WEB-INF/tags
- Dans les sous-répertoires de /META-INF de chaque fichier jar inclus dans le projet Web (donc dans le répertoire /WEB-INF/lib)

4.2 Utilisation spécifique à un projet

Des custom tags peuvent être créés dans le but d'éliminer le code redondant pour un projet unique. Dans ce cas, le serveur d'application est capable de générer la TLD correspondante, sans que le développeur ne soit contraint à en écrire un lui-même.

Pour ce faire, il est nécessaire de placer ces tag-files à un endroit particulier de l'application Web. En effet, tout tag-file peut être placé dans un sous-répertoire de /WEB-INF/tags.

Par exemple, dans le projet d'exemple des custom tags, aucun TLD n'a été écrit pour les tags générant la table. Ceux-ci se trouvent dans le répertoire /WEB-INF/tags/ct.

La déclaration dans les pages JSP est alors un peu différente que dans le cas de l'utilisation d'un custom tag « packagé » :

```
<%@ taglib prefix="ct" tagdir="/WEB-INF/tags/ct" %>
```

Dans ce cas de figure, le serveur d'application va tenter de trouver tous les fichiers .tag dans le répertoire /WEB-INF/tags/ct, et créera un TLD implicite à l'aide des fichiers trouvés.

4.3 Cas des fonctions

Les fonctions quant à elles, doivent être déclarées dans un TLD créé par le développeur. Il n'existe pas de mécanisme implicite pour celles-ci.

Il conviendra donc d'utiliser le mécanisme étudié au chapitre 5 pour déclarer des fonctions , même si celle-ci n'ont pour but d'exister que dans l'application où elles sont utilisées.

4.4 Packaging dans des fichiers Jars

Les custom tags et fonctions peuvent être « packagés » dans un fichier Jar qui leur est propre, pour permettre leur réutilisation à travers plusieurs applications Web.

La création d'un fichier TLD est ici indispensable. Celui-ci doit se trouver dans le répertoire /META-INF du fichier jar en question.

Dans ce jar, les tag-files doivent se trouver dans le répertoire /META-INF/tags ou l'un de ses sous-répertoires.

5 Fonctions

Les fonctions sont un ajout à la norme JSP 2.0 qui permet d'appeler des méthodes java statiques depuis une page JSP, dans le but de les utiliser dans les expressions El. Ces fonctions se déclarent de la même manière que les custom tags, c'est à dire dans un fichier TLD:

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/JEE"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/JEE web-
jsptaglibrary_2_0.xsd"
    version="2.0">

    <tlib-version>1.0</tlib-version>
    <short-name>fns</short-name>
    <uri>http://organisation.com/taglib/funcs.tld</uri>

      <function>
    <name>toUpper</name>
    <function-class>test.JSPFuncs</function-class>
    <function-signature>
        java.lang.String toUpper(java.lang.String)
    </function-signature>
</function>

</taglib>
```

La fonction correspondante dans le code Java doit impérativement être statique:

```
package test;

public class JSPFuncs {
      public static String toUpper(String value) {
            return value.toUpperCase();
      }
}
```

Et voici pour finir, son utilisation dans la page JSP:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="fns" uri="http://organisation.com/taglib/funcs.tld" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
      ${fns:toUpper("bonjour") }
</body>
</html>
```