Applications orientées données (NSY135)

5 – Vues: JSP et **tag libraries**

Auteurs: Raphaël Fournier-S'niehotta et Philippe Rigaux
(philippe.rigaux@cnam.fr,fournier@cnam.fr)

Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

Plan du cours

S2 Les tag libraries

Introduction

- tag libraries = “bibliothèques de balises”
- On va voir en détail ici ces tag libraries qui permettent d’étendre les JSP pour accéder à des bibliothèques Java
- Ces bibliothèques peuvent être définies par tout programmeur
- Certaines servent pour résoudre des problèmes récurrents et sont intégrées dans une librairie standard, la Java Standard Tag Library ou JSTL
- On va l’explorer un peu ici, en v1.2

JSTL et Tomcat

- Pour que les balises de la JSTL soit reconnues par Tomcat, il faut installer deux fichiers accessibles à https://jstl.java.net/download.html:
  - jsp-api-1.2.jar
  - jstl-1.2.jar
- à copier dans WEB-INF/lib
- redémarrer Tomcat pour valider les changements

Exemple

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

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

<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>Ma première JSTL</title>
</head>

<body>
  <c:out value="test" />
 </body>
</html>
```

Exemple suite

- la librairie est incluse avec la directive `taglib`
- L'URI (**Universal Resource Identifier**) identifie la librairie, que Tomcat va donc chercher dans son environnement.
- Le préfixe *c* va servir, lui, à reconnaître les balises de la librairie.
- exemple :

```
<c:out value="test" />
```

- Le préfixe agit comme un **espace de nommage** au sens de XML.
- Toutes les balises dont le nom est préfixé par *c:* sont considérées comme appartenant à la JSTL, et cette dernière leur associe une fonctionnalité particulière.
- Dans notre exemple, il n'est pas trop difficile d'imaginer que *c:out* permet d'afficher une valeur dans le document HTML produit.

Variables, expressions et JSTL

- La librairie JSTL est étroitement associée aux expressions.
- On peut donc combiner une expression, et l'affichage de son résultat, comme dans:

```
<c:out value="${2+2}" />
```

- l'utilisation de balise peut paraître inutile, on pourrait employer l'expression seule
- en fait, JSTL effectue un pré-traitement de la chaîne de caractères à afficher, pour la rendre compatible avec XHTML
- Les caractères réservés comme <, >, &, sont remplacés par des références à des **entités**, comme `<`, `>`, `&`, etc.
- Exemple avec des fragments de code tels que :

```
<langageC>if (x > 2 && x < 5) {}</langageC>
```

- on utilise c:out :

```
<c:out value="<langageC>if (x > 2 && x < 5) {}"/>
```

♠ La protection des caractères interdits est particulièrement importante quand on veut afficher du texte provenant d'une source externe (formulaire, base de données) pour éviter les attaques d'injection (de code javascript par exemple)

Variables, expressions et JSTL (suite)

- La balise c:out accepte une valeur par défaut si l'expression échoue (par exemple parce qu'un objet n'existe pas):

```
<c:out value="${bean}" default="Où est mon bean??" />
```

- La balise c:set sert à **définir** ou **modifier** une variable et à lui affecter une valeur:

```
<c:set var="uneVariable" value="${17 * 23}" />
```

- On peut ensuite l'afficher avec ${uneVariable}.
- Attention cependant : pas de programmation compliquée dans la vue, dont l'objectif reste d'afficher des informations produites par le modèle et le contrôleur, pas pour développer une algorithmique liée à l'application.

Conditions et boucles

- Dans une application Web typique, on voudra afficher une message de bienvenue si l'utilisateur est connecté, un formulaire de connexion sinon.
- il nous faut donc recourir à des conditions et tester la valeur d'une variable
- avec JSTL, c'est la balise <c:if>, avec un attribut `test` contenant une expression Booléenne
- voici un exemple, supposant qu'on a défini une variable `uneVariable` :

```
<c:if test="${uneVariable > 200 }">
Ma variable vaut plus que 200.
</c:if>
```

- Pour des tests un peu plus étendus, on peut recourir à une variante du `case/switch`:

```
<c:choose>
<c:when test="${uneVariable == 0}">Variable null</c:when>
<c:when test="${uneVariable > 0 && uneVariable < 200}">Valeur modérée</c:when>
...
<c:otherwise>Sinon...</c:otherwise>
</c:choose>
```

Boucles

- Pour traiter des variables de type liste ou ensemble, il est indispensable de disposer de boucles
- Cas d'un tableau indicé (on énumère toutes les valeurs de l'index) :

```
<table>
<c:forEach var="i" begin="0" end="5" step="1">
<tr>
<td><c:out value="${i}" /></td>
</tr>
</c:forEach>
</table>
```

- cas où la variable correspond à une **collection** dont on ne connaît pas a priori la taille (**HashMap**, **ArrayMap**, **Collection**) :

```
<c:forEach items="${personnes}" var="personne">
<c:out value="${personne.nom}" />
</c:forEach>
```

- Une variable locale à la boucle, de nom *personne*, est ici définie et successivement associée à toutes les valeurs de la collection. Dans la boucle, on se retrouve donc en situation d'avoir une variable simple, objet, à laquelle on peut appliquer les opérations déjà vues d'accès aux propriétés et d'affichage.

pageContext

- L'objet implicite `pageContext` est un peu différent.
- Ce n'est pas une `Map` mais un objet donnant accès au contexte de l'exécution d'une JSP par l'ensemble des attributs suivants:

Nom	Type	Description
`request`	`ServletRequest`	L'objet `request`
`response`	`ServletResponse`	L'objet `response`
`servletConfig`	`ServletConfig`	Configuration de la servlet
`servletContext`	`ServletContext`	Contexte de la servlet
`session`	`HttpSession`	La session

Liens

- A priori un lien peut être directement inclus comme du HTML, sous la forme standard d'une ancre <a/>. Exemple :

  `http://orm.bdpedia.fr`

- Dans de nombreux cas les choses sont plus compliquées, et des éléments dynamiques interviennent dans la création de l'URL du lien. Voici les principaux cas:
  - le lien mène à une autre page du site : faut-il indiquer une adresse relative ou une adresse absolue, et dans ce dernier cas laquelle ?
  - le lien comprend des paramètres : leur valeur doit être encodée selon la norme HTTP pour survivre au transfert réseau ;
  - enfin citons une fonctionnalité sur laquelle nous revenons plus loin: la gestion des sessions.

- La balise JSTL <c:url> fournit de manière transparente un support pour gérer ces besoins

Liens et contexte

- Dans notre 1e JSP, nous avions introduit une URL de la forme :
  `${pageContext.request.contextPath}/convertisseur`
- L'URL interne est `/convertisseur`.
- Pour ne pas dépendre, dans l'URL complète, du contexte qui peut varier (*dev* pour le développeur, *testapp* pour la machine d'intégration, *appname*, etc.), nous avons introduit un contexte **dynamique** représenté par `${pageContext.request.contextPath}`.
- il s'agit d'une expression, mais nous n'avions pas encore la possibilité d'utiliser les extensions JSTL. Avec, le lien devient simplement:
  ```
  <c:url value="/convertisseur"/>
  ```
- Notez bien que cette adresse est **absolue**. Elle est automatiquement étendue par le conteneur de servlet en ajoutant le chemin de l'application. Pour nous, cela donne:
  `/nsy135/convertisseur`

Liens (suite)

- On ne code surtout pas **en dur** valeurs dépendant de la configuration dans le code
- Le **tag** `c:url` est intégrable tel quel dans une balise HTML `a`, ce qui donne une syntaxe un peu étrange:

```
<a href="<c:url value="/convertisseur"/>">Lien vers mon convertisseur</a>
```

- Le second avantage de `c:url` est la prise en charge automatique de l'encodage des valeurs de paramètre. La syntaxe pour les paramètres est la suivante:

```
<c:url value="/convertisseur">
  <c:param name="param1" value="${personne.nom}"/>
  <c:param name="param2" value="${societe.id}"/>
</cr:url>
```

- Si un paramètre contient des caractères accentués, des espaces, ou autres caractères interdits, il faudra encoder la balise <c:param>.
  cf `http://www.w3schools.com/TAGs/ref_urlencode.asp`