Java EE

Cours 6

JSTL et EL

Cours de 2e année ingénieur

Expression Language (EL)

- Depuis la version 2.0 des JSP, il est possible de placer **à n'importe quel endroit d'une page JSP** des expressions qui sont évaluées et remplacées par le résultat de leur évaluation : les **Expressions Languages** (**EL**) .

- Elles permettent de manipuler les données au sein d'une page JSP plus simplement qu'avec les **scriptlets Java** (<% ... %>) .

- Utilisées conjointement avec des librairies de tags (telles que les JSTL que nous verront plus tard), elles permettent de se passer totalement des scriptlets.

- Une EL permet également d'accéder simplement aux **beans** des différents **scopes** de l'application web (**page**, **request**, **session** et **application**).

Expression Language (EL)

- La syntaxe d'une EL est de la forme suivante :

```
${ expression }
```

- La chaîne **expression** correspond à l'expression à interpréter. Une expression peut être composée de plusieurs **termes** séparés par des **opérateurs**.
  - Ex :

```
${ (10 + 2)*2 }   => 24
  ${ a && b }     => a ET b
```

Ou, comme nous avions vu pour les beans :

```
${bean.property}
```

Expression Language (EL)

- Les différents **termes** peuvent être :
  - **Un type primaire** *(null, Long, Double, String, Boolean, …)*
  - **Un objet implicite** *(requestScope, sessionScope, param, paramValues, header, headerValues, …)*

    Ex: *${sessionScope.myBean.property}*

    => pour rechercher le bean dans le scope de session
  - **Un attribut d'un des scopes de l'application**. Cet attribut pouvant être stocké dans n'importe quel scope (page, request, session ou application). **L'EL cherchera dans tous les scopes dans cet ordre**. On a donc :

    `${name}` ⇔ `<%= pageContext.findAttribute("name"); %>`

    Il est conseillé de n'utiliser cette forme que pour accéder aux objets du scope **page**, afin d'éviter d'éventuels conflits. Pour les autres scopes, il est préférable d'utiliser les objets implicites *${requestScope.name}*, *${sessionScope.name}*, *${applicationScope.name}*, ...
  - **Une fonction EL** (cf JSTL)

Expression Language (EL)

- Les EL permettent d'accéder simplement aux propriétés (attributs) des objets ou des beans:
  - `${ object.property }` <!-- objet normal -->
  - `${ object["index property"] }` <!-- map -->
  - `${ object['index property'] }` <!-- map -->
  - `${ object[index] }` <!-- tableau/List -->

  Note : Les **chaînes de caractères** peuvent être définies entre simples quotes **' '** ou double quotes **" "**

- Il est possible d'indiquer au conteneur JSP 2.0 de ne pas interpréter les **EL** d'une page. Il suffit pour cela d'utiliser l'attribut **isELIgnored** de la directive **page** :

```
<%@ page isELIgnored="true" %>
```

- Il est également possible de ne pas interpréter une **EL** en particulier en la protégeant avec un anti-slash :

```
\${ ceci ne sera pas interprété comme une EL }
```

Expression Language (EL)

- **Gestion des Exceptions :** Les EL gèrent un certains nombres d'exceptions afin de ne pas avoir à les traiter dans les JSP.
  - **NullPointerException**
    - Les différentes propriétés d'un élément ne sont pas forcément renseignées et peuvent très bien être *null*. Par exemple, dans l'expression :

      ```
      ${ sessionScope['data'].information.date }
      ```

      plusieurs éléments peuvent être *null*. En effet, l'attribut *data* peut ne pas être présent dans la session, ou alors sa méthode *getInformation()* peut renvoyer *null...*
      Dans tous les cas, l'expression prendra la valeur ***null*** mais **aucune exception ne sera levée**.
    - De plus, lors de l'affichage dans la page JSP, toutes les valeurs *null* sont remplacées par des chaînes vides, afin de ne pas afficher le texte « null » à l'utilisateur…

Expression Language (EL)

- **ArrayOutOfBoundException :**
  - De la même manière, l'accès à un index incorrect d'un élément d'un tableau ou d'une **List** ne provoquera pas d'exception mais renverra une valeur **null**.

- **ELException :**
  - Certaines exceptions peuvent toujours être levées, mais elles ne concernent pas directement les données mais l'expression **EL**. Par exemple, lorsque l'expression est mal formé, que l'on tente d'accéder à une propriété sans accesseur publique, ou que l'on accède à une propriété indexée avec un index qui ne correspond pas à un nombre entier…

Expression Language (EL)

- **Avantages des EL :**
  - Une meilleure **lisibilité** : le code se limite quasiment au nom du bean et de sa propriété.
  - Pas de **cast**, et de ce fait pas d'import (on accède aux propriétés par réflexion).
  - **Gestion des Exceptions :** Soit la donnée à afficher est valide alors on l'affiche, soit elle n'est pas présente et on n'affiche rien... Cela permet de se passer d'effectuer plusieurs vérifications au sein des JSP avant d'afficher des données...

Expression Language (EL)

- **Conclusion :**

  - Les **EL** permettent donc de simplifier le code des pages JSP tout en améliorant la sécurité du code grâce à la gestion de certaine exception de base.

  - La notion d'**Expressions Languages** a été introduite afin de faciliter la conception de pages JSP, en particulier afin de pouvoir accéder et utiliser des données sans devoir maîtriser un langage aussi complexe que Java...

  - En effet, la logique de conception des pages JSP se rapproche de la logique de conception d'une page HTML ou d'un fichier XML. Ainsi, une formation de base peut permettre à un web designer de concevoir des pages JSP dynamiques en utilisant des beans créées par un développeur Java...

Java Standard Tag Library (JSTL)

- De nombreux frameworks facilitent le développement d'application Java EE. **JSTL** propose une librairie standard pour la plupart des fonctionnalités de base d'une application Java EE.
- Le but de la **JSTL** est de simplifier le travail des auteurs de page JSP, c'est à dire la personne responsable de la couche présentation d'une application web J2EE.
  En effet, un web designer peut avoir des problèmes pour la conception de pages JSP du fait qu'il est confronté à un langage de script complexe qu'il ne maîtrise pas forcément.
- La **JSTL** permet de développer des pages JSP en utilisant des balises XML, donc avec une syntaxe proche des langages utilisés par les web designers, et leur permet donc de concevoir des pages dynamiques complexes sans connaissances du langage Java.
- La **JSTL** se base sur l'utilisation des **EL** en remplacement des scriptlets Java.
- Il existe différentes versions des **JSTL** (1.0, 1.1, 1.2). La plus récente se base sur les **JSP 2.0** qui intègre un moteur d'**EL**.

Java Standard Tag Library (JSTL)

- **Utilisation:**
  - Pour utiliser les JSTL, il suffit de copier dans le répertoire WEB-INF/lib de votre application WEB les fichiers **jstl.jar** et **standard.jar** qui se trouvent dans le répertoire lib de la distribution JSTL.
  - Les JSTL disposent d'un ensemble de librairies de fonctionnalités différentes
  - Pour inclure une librairie jstl à une page JSP il faut ajouter une déclaration en tête de page (directive taglib) de cette façon:

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

  - Cette ligne permet d'inclure la librairie "core" de la JSTL qui contient les actions de base d'une application web.
  - Voici une liste des différentes librairies existantes :

Librairie	URI	Préfixe
Core	http://java.sun.com/jsp/jstl/core	c
Format	http://java.sun.com/jsp/jstl/fmt	fmt
XML	http://java.sun.com/jsp/jstl/xml	x
SQL	http://java.sun.com/jsp/jstl/sql	sql
Fonctions	http://java.sun.com/jsp/jstl/functions	fn

Librairie Core

- Cette librairie contient les actions de base d'une application web.
- **Gestion des variables de scope :**
  - **<c:out/>** : Afficher une expression (⇔ <%= ... %>).

  Ex : Afficher « Bonjour »:

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

  Afficher la propriété *Nom* du **bean** *Etudiant* ou "Inconnu" s'il est *null*:

```
<c:out value="${Etudiant.nom}"> Inconnu </c:out>
```

  Note : Le conteneur JSP 2.0 gère lui-même les EL, ainsi :

```
${expression} ⇔ <c:out value="${expression}" escapeXml="false" />
```

  - **<c:set/>** : Définir une variable de scope ou une propriété d'un attribut ou d'un bean.

  Ex : Stocker *valeur* dans une variable *maVariable* ayant une portée sur la request.

```
<c:set var="maVariable" value="valeur" scope="request" />
```

  Changer la propriété *name* de l'attribut *att* de la session :

```
<c:set target="${session['att']}" property="name" value="new value"/>
```

  - **<c:remove/>** : Supprimer une variable de scope.

  Ex : Supprimer l'attribut *varName* de la session:

```
<c:remove var="varName" scope="session"/>
```

  - **<c:catch/>** : Intercepter les exceptions qui sont levées lors de l'exécution du code inclus dans son corps. (cf exemple Slide 14).

Librairie Core

- **Structures conditionnelles :**
  - **<c:if/> :** Traitement conditionnel (le même que celui du langage Java).

  Ex : Dans un Select, si le paramètre *select* a pour valeur *choix1* alors on met cette option en sélection.

```
<option value="choix1"
   <c:if test="${param.select == 'choix1'}">SELECTED</c:if>
                          >choix 1</option>
```

  - **<c:choose /> :** Traiter différents cas mutuellement exclusifs (idem que Switch en Java).

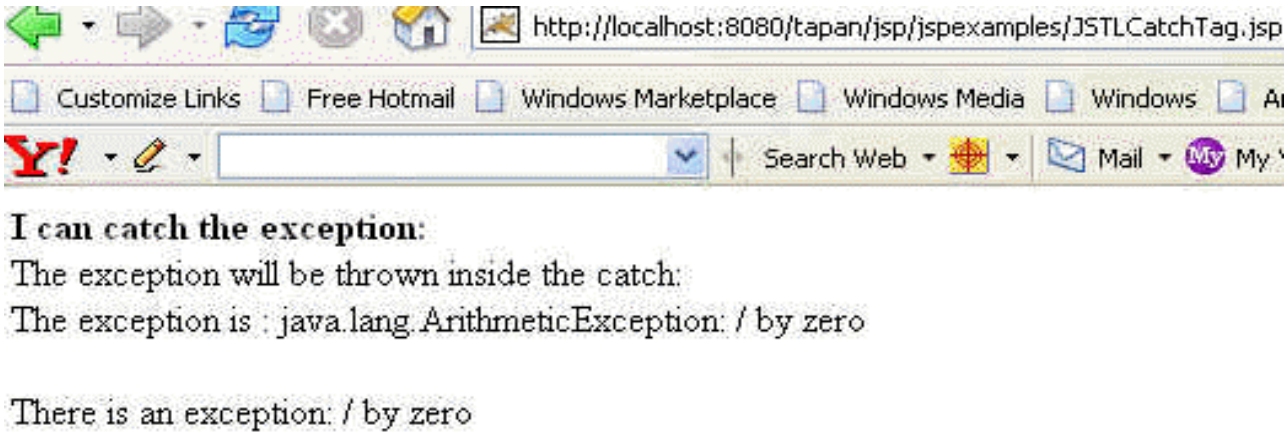
  Ex : Afficher un message d'accueil différent en fonction de la property *civilite* du bean *personne*.

```
<c:choose>
   <c:when test="${personne.civilite=='Mr'}">Bonjour Monsieur</c:when>
   <c:when test="${personne.civilite=='Mme'}">Bonjour Madame</c:when>
   <c:when test="${personne.civilite=='Mlle'}">Bonjour Melle</c:when>
   <c:otherwise>Bonjour</c:otherwise>
</c:choose>
```

Librairie Core

- **Exemple de page JSP pour la gestion de l'exception : Division par zéro.**

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
<head>
      <title>Catching the Exception</title>
</head>
<body>
      <strong>I can catch the exception:</strong><br>
      <c:catch var ="catchException">
               The exception will be thrown inside the catch:<br>
               <% int x = 5/0;%>
      </c:catch>
      <c:if test = "${catchException!=null}">
               The exception is : ${catchException}<br><br>
               There is an exception: ${catchException.message}<br>
      </c:if>
</body>
</html>
```



Librairie Core

- **Structures itératives :**
- **<c:forEach/> :** Permet d'effectuer simplement des itérations sur plusieurs types de collection de données (idem que For et While en Java).

  Ex : Afficher « 1,2,3,4,5,6,7,8,9,10 »

```
<c:forEach var="i" begin="1" end="10"  step ="1"  >
        ${i} ,
</c:forEach>
```

  Afficher le contenu d'un tableau de String nommé *customers*

```
<c:forEach var="customer" items="${customers}">
        ${customer}<br>
</c:forEach>
```

- L'attribut **varStatus** permet d'obtenir des informations sur la boucle courante :

  Ex : On parcours le même tableau, mais en affichant à chaque itération certaines infos de boucle.

```
<c:forEach var="customer" items="${customers}" varStatus="status">
     <c:out value="${status.index}"/></td> // N° de l'itération courante
     <c:out value="${status.count}"/></td> // Nb Tours de l'itération.
     <c:out value="${status.first}"/></td> // True si la boucle courante
                                           //est la première de l'itération.
     <c:out value="${status.last}"/></td> // resp. la dernière.
</c:forEach>
```

Librairie Core

- **Attention** : ***count*** et ***index*** ne sont pas forcément identiques. En effet, ***count*** comptabilise le nombre de tours de l'itération et index représente l'index de l'élément courant.
- On peut également itérer sur les éléments d'une Map :

```
<c:forEach var="entry" items="${myMap}" >
  Le paramètre "${entry.key}" vaut "${entry.value}" <br />
</c:forEach>
```

- **<c:forTokens /> :** Permet de découper des chaînes de caractères selon un ou plusieurs délimiteurs. Chaque marqueur ainsi obtenu sera traité dans une boucle de l'itération.

  Ex : Décomposer *str* et afficher chaque sous-chaîne sur une ligne différente.

```
<% String str = "mot1;mot2;mot3;mot4" %>
<c:forTokens var="mot" delims=";" items=str>
        ${mot} <br />
</c:forTokens>
```

Librairie Core

- **Les URLS :**
- **<c:url /> :** Permet de créer des URLs absolues, relatives au contexte, ou relatives à un autre contexte.

  Ex : l'URL « /MaPage.jsp?nom=Jean » s'écrira :

```
<c:url value="/MaPage.jsp?nom=Jean" />
```

  Elle contient un paramètre *nom* qui a pour valeur *Jean.* On peut utiliser une balise **<c:param />** pour l'écrire plus simplement :

```
<c:url value="/MaPage.jsp">
        <c:param name="nom" value= "Jean" />
</c:url>
```

Note: Avec la balise <c:param />, le nom et la valeur du paramètre de l'URL sont automatiquement encodés afin de respecter le format des URLs (les 'espaces' sont remplacés par des '+' etc.). De plus, le corps du tag est bufférisé et tout ce qui est écrit à l'intérieur n'est pas reporté sur la page JSP mais ignoré (plus efficace que <jsp:include>)

Librairie Core

- Les URLs sont réécrites de la manière suivante :
  - Le chemin du contexte est ajouté aux URLs relatives à une application locale (URLs qui commencent par **'/').**
  - **Si les cookies ne sont pas pris en charge**, les URLs relatives à l'application courante sont encodées afin de rajouter le **jsessionid.**

  Ex: `/thread.html;jsessionid=970CACB1E5707A.app03_02?topicId=12&sid=1`

  Au lieu de : `/thread.html?topicId=12&sid=1`
  - Les paramètres ajoutés avec les balises **<c:param />** sont ajoutés à l'URL.
- **<c:redirect /> :** Envoi une commande de redirection HTTP au client.

  Ex : Redirection vers une page d'erreur avec des paramètres :

```
<c:redirect url="/error.jsp">
      <c:param name="from" value="${pageContext.request.requestURI}" />
</c:redirect>
```

- **<c:import />** : Permet d'importer une ressource selon son URL.

  Ex : Importer un fichier (⇔ <jsp:include />)

```
<c:import url="/file.jsp">
      <c:param name="page" value="1" />
</c:import>
```

  Importer une ressource distante FTP dans une variable

```
<c:import url="ftp://server.com/path/file.ext" var="file" scope="page"/>
```

Librairie de Fonctions EL

- La création de librairies de fonctions est une nouveauté des **JSP 2.0 qui utilise à la fois les EL et les librairies de tags (telles que JSTL).**

- Cette librairie n'est disponible que dans la **JSTL 1.1 puisqu'elle** nécessite un conteneur **JSP 2.0. En effet,** la **JSTL 1.0 est prévue pour fonctionner avec un conteneur JSP 1.2 et ne peux donc pas utiliser des fonctions EL.**

- La plupart des fonctions de cette librairie concernent la gestion des chaînes de caractères. Toutes ces fonctions interprète la valeur **null comme une chaîne vide ("").**

Librairie de Fonctions EL

Fonctions EL :

- **<fn:length> :** Nb éléments d'une collection (tableau, List, Set, Map,…), Nb caract. d'une String
- **<fn:toUpperCase>, <fn:toLowerCase>** : Changer la casse d'une String.
- **<fn:substring>, <fn:substringBefore>**, **<fn:substringAfter> :** Sous-chaînes d'une String.
- **<fn:trim> :** Supprime les espaces au début et à la fin de la String.
- **<fn:replace> :** Remplacer toutes les occurrence d'une sous-chaîne dans une String.
- **<fn:indexOf>, <fn:startsWith>, <fn:endsWith>, <fn:containsIgnoreCase>** : Vérifier si une String contient une autre String.
- **<fn:split>, <fn:join> :** Découper une String dans un Tableau de sous-chaînes. Ou inversement, joindre tous les éléments d'un tableau de chaîne (*String[]*) dans une unique chaîne(*String*).
- **<fn:escapeXml> :** Protège les caractères qui peuvent être interprétés comme des marqueurs XML.

  Ex :

```
${fn:escapeXml("les <balises> xml & html")}
    ⟶ "les &lt;balises&gt; xml &amp; html"
```

Librairie de Fonctions EL

Exemples d'applications :

- Tronquer *name* à 30 caractères et l'afficher en Majuscules :

```
${fn:toUpperCase(fn:substring(name, 0, 30))}
```

- Afficher la sous-chaîne (de la String *text*) qui précède le premier caractère * :

```
${fn:substringBefore(text, '*')}
```

- Afficher la sous-chaîne (de la String *text*) qui est entre parenthèses :

```
${fn:substring(text, fn:indexOf(text, '(')+1, fn:indexOf(text, ')'))}
```

- Afficher *name* s'il contient *searchString* :

```
<c:if test="${fn:containsIgnoreCase(name, searchString)}">
   Found name: ${name}
</c:if>
```

- Afficher *text* avec des bullets à la place des - :

```
${fn:replace(text, '-', '&#149;')}
```

Webographie

- http://blog.erdener.org/archives/files/jstl-quick-reference.pdf
  => Mémo exhaustif sur les JSTL (en anglais)
- http://www.jcp.org/aboutJava/communityprocess/first/jsr052/
  => Documentation sur la JSTL 1.0 (en anglais)
- http://adiguba.developpez.com/tutoriels/j2ee/jsp/jstl/
  => Présentation assez complète des JSTL et des différentes librairies (en français)
- http://www.jmdoudoux.fr/java/dej/chap051.htm
  => Une autre présentation sur les JSTL (en français)