Table des matières

I. Présentation:

1. Définition:

La JavaServer Pages Standard Tag Library (JSTL) est un composant de la plate-forme JEE de développement. Elle étend la spécification JSP en ajoutant une bibliothèque de balises pour les tâches courantes, comme le travail sur des fichiers XML, l'exécution conditionnelle, les boucles et l'internationalisation. JSTL a été développée par la Java Community Process (JCP). Le 8 mai 2006, la JSTL 1.2 est sortie.

2. Objectifs:

JSTL a surtout permis de simplifier le travail des développeurs JEE. Le développement de la vue n'a jamais été aussi facile et aussi simple à manipuler.

En plus, la JSTL a permis de développer des JSL en utilisant une syntaxe très proche des langages utilisés par les web designers (HTML et XHTML).

Finalement, on est capable maintenant de développer des pages JSP entières sans connaissances préalables du langage Java.

3. Versions :

La JSTL est disponible sous les versions suivantes :

- La JSTL 1.0 nécessite un conteneur JSP 1.2 (J2EE 1.3).
- La JSTL 1.1 nécessite un conteneur JSP 2.0 (J2EE 1.4).
- La JSTL 1.2 nécessite un conteneur JSP 2.1 (J2EE 1.5).

La JSTL se base sur l'utilisation des Expressions Languages en remplacement des scriptlets Java. Toutefois, ce mécanisme n'est disponible qu'avec un conteneur JSP 2.0.

4. Librairies :

La JSTL sous ses versions 1.1 et 1.2 présente les bibliothèques suivantes :

Bibliothèque	URI	Prefixe
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
Function	http://java.sun.com/jsp/jstl/functions	fn

NB : Pour développer une page JSP a base de JSTL il suffit de déclarer la bibliothèque et de configurer le fichier web.xml qui contient le paramétrage des servlets d'une application JEE.

II. Les librairies:

1. Librairie de base:

Les librairies de base contiennent les actions de base d'une application web et donc les actions les plus utilisées. Elles sont divisées en quatre catégories: la gestion des variables de scope, les actions conditionnelles, les itération et les URLs.

Pour utiliser la librairie de base, on doit l'importer de la manière suivante:

Déclaration de la librairie 'core' :

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

a. Gestion des variables de scope:

Ces balises permettent d'afficher, créer, modifier et supprimer des variables en plus de gérer les exception.

- **<c:out/>:** Permet d'afficher une expression après l'avoir évalué tout en protégeant nos variables contre la faille XSS en protégeant les caractères spéciaux HTML.
  Elle peut comporter 3 attributs: value, défault et escapeXml

Exemple

```
<!-- Afficher l'user-agent du navigateur ou "Inconnu" si il est absent : -->
<c:out value="${header['user-agent']}" default="Inconnu"/>

<!-- Même chose en utilisant le corps du tag : -->
<c:out value="${header['user-agent']}">
        Inconnu
</c:out>
```

- **<c:set/>:** Permet de définir une variable dans une portée particulière (page, requête, session ou application). Elle permet aussi de modifier la valeur d'une propriété d'un bean.
  Elle peut comporter 5 attributs: valeur, var, scope, target et property

Exemple

```
<!-- Mettre ${expression} dans l'attribut "varName" de la session : -->
<c:set scope="session" var="varName" value="${expression}" />

<!-- Changer la propriété "name" de l'attribut "varName" de la session :
<c:set target="${session['varName']" property="name" value="new value"/>

<!-- Changer la propriété "name" de l'attribut "varName" de la session
<c:set target="${session['varName']" property="name">
        Nouvelle valeur de la propriété "name"
</c:set>
```

- **<c:remove/>:** Permet de supprimer une variable de scope indiqué.
  Elle peut comporter 2 attributs: var et scope

Exemple

```
<!-- Supprime l'attribut "varName" de la session -->
<c:remove var="varName" scope="session"/>
```

- **<c:catch/>:** Permet d'intercepter les exceptions qui peuvent être générées par son corps.
  Elle peut avoir un attribut var qui, s'il est spécifié contiendra l'exception interceptée (ou null dans le cas échéant). Si var n'est pas spécifié, les exceptions interceptées seront perdues.

Exemple

```
<!-- Ignorer toutes les exceptions d'une partie de la page : -->
<c:catch>
        <c:set target="beans" property="prop" value="1"/>
</c:catch>

<!-- Stocker dans le scope page l'exception intercepté : -->
<c:catch var="varName">
        <c:set target="beans" property="prop" value="1"/>
</c:cath>
```

b. Actions conditionnelles:

Ces balises permettent d'effectuer des tests conditionnels de la même manière que les mots-clefs if ou switch du langage Java.

- **<c:if/>:** Permet d'effectuer un traitement conditionnel de la même manière que if dans java
  Elle peut avoir 3 attributs: test, var et scope.

Exemple

```
<c:if test="${empty param['page']}">
        Le paramètre page est absent !
</c:if>
```

- **<c:choose/>:** Permet un traitement conditionnel de la même manière que switch dans java. C'est à dire plusieurs possibilit"es dont une seule sera réalisée et évaluée.
  Elle exacute le corps du premier tag <c:when/> dont la condition est vérifiée.
  Elle n'a pas d'attributs.

- **<c:when/>:** Permet de définir une option de l'action <c:choose/>. Elle doit impérativement avoir la balise <c:choose/> comme parent.
  Elle a un attribut: test

Exemple

```
<!-- Afficher un message différent selon la valeur du bean 'value' : -->
<c:choose>
        <c:when test="${value==1}"> value vaut 1 (Un) </c:when>
        <c:when test="${value==2}"> value vaut 2 (Deux) </c:when>
        <c:when test="${value==3}"> value vaut 3 (Trois) </c:when>
</c:choose>
```

- **<c:otherwise/>:** Permet de définir le traitement à exécuter si aucun tag <c:when/> n'a été évalué. Elle doit impérativement avoir la balise <c:choose/> comme parent et doit être placée après la dernière balise <c:when/>. Elle est équivalente au mot-clef default dans switch dans java.

Exemple

```
<c:choose>
        <c:when test="${value==1}"> value vaut 1 (Un) </c:when>
        <c:when test="${value==2}"> value vaut 2 (Deux) </c:when>
        <c:when test="${value==3}"> value vaut 3 (Trois) </c:when>
        <c:otherwise>
                value vaut ${value} (?)
        </c:otherwise>
</c:choose>
```

c. Itérations:

Ces balises permettent d'effectuer des boucles de la même manière que les mots-clefs for ou while en Java.

Les balises d'itération peuvent avoir 5 attributs communs: var, varStatus, begin, end et step.

- **<c:forEach/>:** Permet d'effectuer des itérations sur un ensemble de collections de données.
  Elle peut avoir un autre attribut item qui peut contenir un iterator, une énumération, un map, un String, ...

Exemple

```
<!-- Afficher tous les éléments d'une collection dans le request-->
<c:forEach var="entry" items="${requestScope['myCollection']}" >
        ${entry}<br/>
</c:forEach>
```

- **<c:forTokens/>:** Permet de découper une chaine de caractères en plusieurs segments selon un certain délimiteur et ensuite, d'itérer sur ces segments.
  Elle a 2 attributs: items et delims.

Exemple

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

d. URLs:

Ces balises permettent de gérer les URLs.

- **<c:param/>:** Permet d'ajouter un paramètre à notre URL. Elle doit avoir comme balise parent (direct ou indirect) une balise <c:url/>, <c:import/> ou <c:redirect/>.
  Ele peut avoir 2 attributs: name et value.

Exemple

```
<!-- La forme suivante : -->
<c:url value="/mapage.jsp?paramName=paramValue"/>

<!-- est equivalente à : -->
<c:url value="/mapage.jsp">
        <c:param name="paramName" value="paramValue"/>
</c:url><br/>
```

- **<c:url/>:** Permet de créer des URLs absolues, relatives au contexte, ou relatives à un autre contexte.
  Elle peut avoir 4 attributs: value, context, var et scope.

Exemple

```
<!-- Création d'un lien dont les paramètres viennent d'une MAP -->
<c:url value="/index.jsp" var="variableURL">
        <c:forEach items="${parameterMap}" var="entry">
                <c:param name="${entry.key}" value="${entry.value}"/>
        </c:forEach>
</c:url>
<a href="${variableURL}">Mon Lien</a>
```

- **<c:redirect/>:** Permet de rediriger le client vers d'autres pages en envoyant une commande de redirection HTTP
  Elle peut avoir 2 attributs: url et context et utilise les mêmes règles pour réécrire les URL que la balise <c:url/>.

Exemple

```
<!-- Redirection vers le portail de developpez.com : -->
<c:redirect url="http://www.developpez.com"/>

<!-- 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. Cette ressource peut appartenir à un autre contexte ou même à un autre serveur.
  Elle peut avoir 6 attributs: url, context, var, scope, charEncoding et varReader.

Exemple

```
<!-- Importer un fichier de l'application (similaire à <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"/>

<!-- Importe une ressource distante dans un Reader -->
<c:url value="http://www.server.com/file.jsp" var="fileUrl">
        <c:param name="file" value="filename"/>
        <c:param name="page" value="1"/>
</c:url>
```

2. Libraires de formatage:

La librairie de formatage se divise en deux partie, la bibliothèque de formatage et la bibliothèque i18n.

a. La bibliothèque i18n:

Elle facilite l'internationalisation d'une page JSP. Pour l'utiliser il faut la déclarer dans le fichier web.xml du répertoire WEB-INF de l'application web de la manière suivante:

```
<taglib>
    <taglib-uri>http://java.sun.com/jstl/fmt<;/taglib-uri>
    <taglib-location>/WEB-INF/tld/fmt.tld</taglib-location>
</taglib>
```

La bibliothèque doit être déclarée avec la directive taglib pour chaque JSP qui utilise un ou plusieurs tags.

Exemple :

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

Pour mettre en œuvre la localisation des messages, on définit un ensemble de fichiers (Bundle) ; un fichier pour la langue par défaut et un fichier pour chaque langue particulière. Tous ces fichiers ont un préfixe commun appelé basename et doivent avoir comme extension .properties. Les fichiers pour les langues particulières utilisent le préfixe commun suivit d'un underscore puis du code langue et éventuellement d'un underscore suivi du code pays. Ces fichiers doivent être inclus dans le classpath : le plus simple est de les copier dans le répertoire WEB-INF/classes de l'application web.

Il est également possible d'utiliser une variante plus spécifique afin d'apporter une différenciation sur des critères spécifiques (Systèmes d'exploitation, variante d'une langue,…).

Lorsqu'un utilisateur se connecte la valeur du header http « Accept-language » est utilisée pour rechercher la meilleure locale à utiliser.

C'est ici que les variables de la bibliothèque de i18n interviennent.

- **<fmt:setLocale/>** : permet de changer la locale à utiliser dans les tags de la librairie. Elle peut avoir 3 attributs: value, variant et scope.

Exemple

```
<fmt:setLocale value="en" scope="page"/>

<!-- Force l'affichage en français pour un utilisateur : -->
<fmt:setLocale value="fr" scope="session"/>
```

- **<fmt:setBundle/>** : Permet de changer la Locale à utiliser dans les tags de la librairie, ou de créer une variable LocalizationContext afin de la réutiliser dans d'autres tags de la librairie.
  Elle peut avoir 3 attributs: basename, var et scope.

Exemple

```
<!-- Change le ResourceBundle par défaut de la page : -->
<fmt:setBundle basename="package.MyRessource"/>

<!-- Création d'un autre Bundle : -->
<fmt:setBundle basename="package.MyOtherRessource" var="bundleName"/>
```

- **<fmt:Bundle/>:** Permet d'utiliser un ResourceBundle limité à une partie de la page JSP (le corps du tag).
  Elle peut avoir 2 attributs: basename et prefix.

Exemple

```
<!-- Utilisation d'un autre Bundle pour afficher des messages
<fmt:bundle basename="package.MessageRessource">
        <fmt:message key="msg.error.key1"/><br/>
        <fmt:message key="msg.error.key2"/><br/>
        <fmt:message key="msg.error.key3"/><br/>
</fmt:bundle>
```

- **<fmt:message/>** : Permet l'affichage d'un message depuis un ResourceBundle.
  Elle peut avoir 4 attributs: key, bundle, var et scope

Exemple

```
<!-- Affichage d'un message directement dans le JSP :
<fmt:message key="message.key"/>
```

- **<fmt:param/> :** Permet de paramétrer l'affichage d'un message obtenu avec <fmt:message/>.
  Elle a un attribut: value.

Exemple

```
<!-- Affichage d'un message avec deux paramètres -->
<fmt:message key="message.key">
        <fmt:param value="${mailbox.userName}"/>
        <fmt:param value="${mailbox.messageCount}"/>
</fmt:message>
```

- **<fmt:requestEncoding/>** : Permet de définir l'encodage de caractère utilisé par le navigateur du client.
  Elle peut avoir un attribut: value

b. La bibliothèque i18n:

La JSTL propose des tags facilitant le formatage des données numériques et des dates/heures. Ces tags prennent en compte la Locale* de l'utilisateur pour paramétrer l'affichage.

*locale : définit les propriétés régionales qui devront etre utilisées à la place de celles définis dans header HTT « Accept-language ».

- **<fmt:setTimeZone/>:** Permet de changer le fuseau horaire à utiliser dans les tags de la librairie, ou de créer une variable TimeZone afin de la réutiliser dans d'autres tags de la librairie.
  Elle peut avoir 3 attributs: value, var et scope.

Exemple

```
<!-- Change le TimeZone par défaut de la page
<fmt:setTimeZone value="GMT-8"/>
```

- **<fmt:TimeZone/>:** Permet d'utiliser un timeZone limité à une partie de la page JSP (le corps du tag).
  Elle a un attribut: value

Exemple

```
<!-- Affichage d'une date GMT : -->
<fmt:timeZone value="GMT">
        <fmt:formatDate value="${myDate}"/>
</fmt:timeZone>
```

- **<fmt:parseDate/>:** Permet de créer des dates en analysant une chaîne de caractère. Le TimeZone et la Locale peuvent modifier lecomportement de ce tag.
  Elle peut avoir 9 attributs: value, type, dateStyle, timeStyle, pattern, timeZone, parseLocate, var et scope

Exemple

```
<!-- Créer une date initialisé au premier janvier 2005 : -->
<fmt:parseDate value="01/01/2005" pattern="dd/MM/yyyy" var="date"/><br/>
```

- **<fmt:formatDate/>:** Permet de formater une date afin de l'afficher à l'utilisateur. Le TimeZone et la Locale peuvent modifier le résultatde ce tag.
  Elle peut avoir 8 attributs: Value, type, dateStyle, timeStyle, pattern, timeZone, var et scope.

Exemple

```
<!-- Afficher une date selon un format spécifique : -->
<fmt:formatDate value="${dateBeans}" pattern="dd/MM/yyyy"/>
```

- **<fmt:parseNumber/>** : Permet d'analyser des chaînes de caractères afin de créer des objets représentant un nombre (sous-class de java.lang.Number).
  Elle peut avoir 6 attributs: value, type, pattern, parseLocate, integerOnly, var et scope.

Exemple

```
<!-- Créer un objet nombre : -->
<fmt:parseNumber value="1250,00" var="num"/>
```

- **<fmt:formatNumber/>** : Permet de formater un nombre afin de l'afficher à l'utilisateur final.
  Elle peut avoir 12 attributs: value, type, pattern, currencyCode, currencySymbol, groupingUsed, maxIntegerDigits, minIntegerDigits, maxFractionDigits, minFractionDigits, var et scope

Exemple

```
<!-- Afficher un pourcentage -->
<fmt:formatNumber value="0.25" type="percent"/>
```

3. Librairies SQL:

Cette librairie permet à la page JSP d'intéragir avec les bases de données relationelles comme mySQL, Oracle, Microsoft SQL Server..

Pour déclarer cette librairie, on l'importe de la manière suivante :

Déclaration de la librairie 'SQL' :

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

Balises :

Balises SQL	Descriptions
sql: setDataSource	Il est utilisé pour créer une source de données simple adaptée uniquement au prototypage.
sql: requête	Il est utilisé pour exécuter la requête SQL définie dans son attribut sql ou le corps.
sql: mise à jour	Il est utilisé pour exécuter la mise à jour SQL définie dans son attribut sql ou dans le corps de la balise.
sql: param	Il est utilisé pour définir le paramètre dans une instruction SQL sur la valeur spécifiée.
sql: dateParam	Il est utilisé pour définir le paramètre dans une instruction SQL sur une valeur java.util.Date spécifiée.
sql: transaction	Il est utilisé pour fournir l'action de base de données imbriquée avec une connexion commune.

La librairie SQL permet de configurer le DataSource et le nombre de ligne maximum d'une requête. On peut bien sûr utiliser des valeurs différentes grâce aux attributs du même nom des différents tags...

a. DataSource :

Cette variable défini la source de données qui sera utilisé dans les tags de la librairie. Cette variable accepte les
éléments suivants :

- Une instance de DataSource.
- Une chaîne représentant le nom JNDI du DataSource (si le conteneur supporte JNDI).
- Une chaîne paramétrée contenant les informations sur la connection (url,driver,user,password)

Configuration du web.xml

```
<!-- Utilisation de JNDI :         -->
<context-param>
        <param-name>javax.servlet.jsp.jstl.sql.dataSource</param-name>
        <param-value>jdbc/myDataBase</param-value>
</context-param>

<!-- Utilisation d'une chaîne paramétrée sur une base mySQL :         -->
<context-param>
        <param-name>javax.servlet.jsp.jstl.sql.dataSource</param-name>
        <param-value>jdbc:mysql://localhost/base,org.gjt.mm.mysql.Driver,login,password</param-value>
</context-param>
```

b. Nombre de lignes maximum :

On peut spécifier un nombre maximum de lignes aux requêtes SQL. Si le nombre maximum de ligne vaut -1 où qu'il n'est pas spécifié, cela signifie qu'aucune limite ne sera appliquée aux requêtes SQL.

- **<sql:setDataSource/>:** Permet de définir le Datasource à utiliser pour les connections à la base de données, ou de créer un objet DataSource.

Exemple

```
<!-- Changer le DataSource a utiliser sur la page (avec JNDI) -->
<sql:setDataSource dataSource="jdbc/myDataBase" scope="page"/>

<!-- Changer le DataSource pour l'utilisateur courant (avec JNDI) -->
<sql:setDataSource dataSource="jdbc/myDataBase" scope="session"/>
```

- **<sql:query/>:** Permet d'exécuter des requêtes (SELECT) sur la base de données.

Exemple

```
<!-- Rechercher tous les éléments d'une table -->
<sql:query sql="select * from tableName" var="result"/>

<!-- Même chose en utilisant le corps du tag -->
<sql:query var="result">
        select * from tableName
</sql:query>
```

- **<sql:update/>:** Permet d'exécuter des commandes SQL tel que INSERT, UPDATE ou DELETE ainsi que les commandes SQL qui ne retournent pas de résultats.

Exemple

```
<!-- Supprimer tous les éléments d'une table -->
<sql:query sql="delete from tableName" var="nbRowDeleted"/>
```

- **<sql:transaction/>:** Permet d'exécuter des commandes SQL tel que INSERT, UPDATE ou DELETE ainsi que les commandes SQL qui ne retournent pas de résultats.

Exemple

```
<!--
<sql:transaction>
        <sql:query sql="requete SQL 1 ... " />
        <sql:query sql="requete SQL 1 ... " />
        <sql:query sql="requete SQL 1 ... " />
</sql:transaction>
```

- **<sql:param/>:** Permet de définir la valeur d'un paramètre d'une requête lorsque le marqueur "?" est utilisé. Il doit obligatoirement être un sous tag de <sql:query/> ou de <sql:update/>.

Exemple

```
<!-- Exemple de requête paramétrée : -->
<sql:query var="result">
        SELECT * FROM tableName
        WHERE row1 = ?
        WHERE row1 = ?

        <sql:param value="valeur1"/>
        <sql:param>
                valeur2
        </sql:param>
</sql:query>
```

- **<sql:dateParam/>:** Permet de définir la valeur d'un paramètre d'une requête lorsque le marqueur "?" est utilisé. Il doit obligatoirement être un sous tag de <sql:query/> ou de <sql:update/>.

Exemple

```
<!-- Exemple de requête paramétrée avec deux dates : -->
<sql:query var="result">
        SELECT * FROM tableName
        WHERE date BETWEEN ? AND ?

        <sql:dateParam value="${startDate}" type="date"/>
        <sql:dateParam value="${endDate}" type="date"/>

</sql:query>
```

4. Librairies XML:

Cette librairie permet d'utiliser et traiter des fichiers XML dans une page JSP.

Pour utiliser cette librairie, on doit l'importer de la manière suivante:

Déclaration de la librairie 'XML' :

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

a. XPath:

XPath est un langage de formulation d'expression par rapport à un fichier XML et donc il est utilisé pour accéder aux données des pages XML. C'est une recommandation de W3C qui fournit sa documentation.

b. Actions XML de Base:

Ces balises permettent l'analyse de fichiers XML et l'accès à ses variables en utilisant des expressions du langage XPath.

- **<x:parse/>:** Permet d'analyser un document XML.
  Elle peut avoir 8 attributs: doc, xml, systemId, filter, var, scope, varDom et scopeDom.

Exemple

```
<!-- Chargement du XML dans un Reader : -->
<c:import url="/docs/myFile.xml" varReader="myFileReader"/>

<!-- Analyse du fichier XML chargé avec c:import -->
<x:parse doc="${myFileReader}" var="parsedDoc"/>
```

- **<x:out/>:** Permet d'évaluer et d'afficher une expression XPath.
  Elle peut avoir 2 attributs: select et escapeXML.

Exemple

```
<!-- Afficher la quantité de PC en stock : -->
<x:out select="$parsedDoc/stock/produit[nom='PC']/quantite"/>
```

- **<x:set/>:** Permet de définir une variable XPath dans une portée particulière (page, requête, session ou application).
  Elle peut avoir 3 attributs: select, var et scope.

Exemple

```
<!-- Stocker le noeud du PC dans un bean : -->
<x:set select="$parsedDoc/stock/produit[name='PC']" var="pc" />
```

c. Actions de contrôle XML:

Ces balises permettent d'effectuer des actions de contrôle de la même manière que ceux de la librairie standard, sauf qu'ils appliquent une action XPath sur un document XML.

- **<x:if/>:** Permet d'évaluer une expression XPath et de déterminer s'il faut exécuter le corps.
  Elle peut avoir 3 attributs: select, var et scope.

Exemple
```
<!-- Afficher un message si la quantité de PC est inférieur au minimum :
<x:if select="$parsedDoc/stock/produit[nom='PC'][quantite<min]">
        Il faut commander des PCs !!!!!
</x:if>
```

- **<x:choose/>:** Permet un traitement conditionnel de la même manière que<c:choose/> de la librairie standard, sauf qu'elle s'applique à une expression XPath.
  Elle n'a pas d'attributs.

- **<x:when/>:** Permet de définir une option de l'action <x:choose/>. Elle doit impérativement avoir la balise <x:choose/> comme parent.
  Elle a un attribut: select

Exemple
```
<x:choose>
        <x:when select="parsedDoc/stock/produit[quantite<min]">
                Le stock est en dessous de la quantité minimum !
        </x:when>
        <x:when select="parsedDoc/stock/produit[quantite=0]">
                Il n'y a plus de stock !
        </x:when>
</x:choose>
```

- **<x:otherwise/>:** Permet de définir le traitement à exécuter si aucun tag <x:when/> n'a été évalué. Elle doit impérativement avoir la balise <x:choose/> comme parent et doit être placée après la dernière balise <x:when/>.

Exemple
```
<code langage="xml" titre="Exemple"><![CDATA[
<!-- Afficher un message différent selon la quantité en stock :
<x:choose>
        <x:when select="parsedDoc/stock/produit[quantite<min]">
                Le stock est en dessous de la quantité minimum !
        </x:when>
        <x:when select="parsedDoc/stock/produit[quantite=0]">
                Il n'y a plus de stock !
        </x:when>
        <x:otherwise>
                La quantité en stock est suffisante.
        </x:otherwise>
</x:choose>
```

- **<x:forEach/>:** Permet d'effectuer des itérations sur les éléments d'un fichier XML.
  Elle peut avoir un autre attribut select

Exemple

```
<!-- Afficher tous les produits du stock : -->
<x:forEach var="produit" select="$stockXml/stock/produit">
        <b><x:out select="$produit/nom"/></b> :
        <x:out select="$produit/quantite"/> unité(s) en stock.<br/>
</x:forEach>
```

d. Transformation XSLT:

XSLT (eXtensible Stylesheet Language Transformations) est un langage de transformation XML, défini par W3C. Il permet de transformer un document XML en un autre format (PDF, HTML, ...). Il utilise XPath pour désigner une partie d'un arbre XML.

- **<x:transform/>:** Permet d'appliquer une transformation XSLT sur un document XML. Elle peut avoir 9 attributs: doc, xml, xslt, docSystemId, xmlSystemId, xsltSystemId, var scope et result.

Exemple

```
<!-- chargement des fichiers XML et XLS : -->
<c:import url="/stock.xml" var="stockXml"/>
<c:import url="/stock.xsl" var="stockXsl"/>

<!-- Transformation XLS : -->
<x:transform doc="${stockXml}" xslt="${stockXsl}"/>
```

- **<x:param/>:** Permet d'ajouter un paramètre à une transformation XSLT et doit avoir comme balise parente <x:tranform/>
  Elle peut avoir deux attributs: name et value.

Exemple

```
<!-- Transformation XLS avec un paramètre : -->
<x:transform doc="${stockXml}" xslt="${stockXsl}">
        <x:param name="param" value="01"/>
</x:transform>
```

5. Librairies de fonctions EL:

Dans le but de fournir un moyen simple d'accéder aux données nécessaire à une JSP, JSTL propose un langage particulier constitué d'expressions permettant d'utiliser et de faire référence à des objets java accessible dans les différents contextes de la page JSP.

La syntaxe de base d'une EL (Expression Langugae) est de la forme ${nomVariable}

Voici une petite comparaison :

Exemple :

//accéder à l'attribut nom d'un objet personne situé dans la session avec Java

<%= session.getAttribute("personne").getNom() %>

//accéder à l'attribut nom d'un objet personne situé dans la session avec EL

${sessionScope.personne.nom}

EL possède par défaut plusieurs variables (Cookie, PageScope, Header, RequestScope, SessionScope,) et propose aussi différents opérateurs logique, arithmétique et de test.

L'inconvénient des EL c'est qu'elle ne permet pas l'accès aux variables locales. Pour pouvoir accéder à de telles variables, il faut obligatoirement en créer une copie dans une des portées particulières : page, request, session ou application.

La librairie de fonction EL est une nouveauté des JSP 2.0 qui utilise à la fois les Expressions

Languages et les librairies de tags. En effet, les fonctions EL doivent être définit dans un descripteur de taglib. Ainsi, cette librairie n'est disponible que dans la JSTL 1.1.

Sa déclaration se fait à travers la balise

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

Cette librairie contient environ 16 fonctions dont la plupart concerne la gestion des chaines de caractères.

NB : Toutes ces fonctions interprète NULL comme chaine vide.

- ${fn:contains()}: Vérifie si une chaîne contient une autre chaîne .
- ${fn:containsIgnoreCase()}: Vérifie si une chaîne contient une autre chaîne en ignorant la case.
- ${fn:endsWith()}: Vérifie si une chaîne se termine par le suffixe indiqué
- ${fn:escapeXml()}: Protège les caractères qui peuvent être interprétés comme des marqueurs XML.
- ${fn:indexOf()}: Retourne l'index de la sous chaîne dans la chaîne
- ${fn:join()}: Joint tous les éléments d'un tableau de chaîne dans une unique chaîne
- ${fn:length()}: Indique le nombre d'éléments d'une collection (tableau, List, Set, Map...) ou le nombre de caractères d'une String
- ${fn:replace()}: Retourne la chaîne après avoir remplacé toutes les occurrences d'une chaîne par une autres
- ${fn:split()}:Permet de découper une chaîne de caractère en plusieurs sous chaîne
- ${fn:startsWith()}: Vérifie si une chaîne commence par le préfixe indiqué
- ${fn:substring()}: Retourne une partie d'une chaîne de caractère selon deux index
- ${fn:substringAfter()}: Retourne la sous chaîne de caractère situé après la sous chaîne spécifiée

- ${fn:substringBefore()}: Retourne la sous chaîne de caractère situé avant la sous chaîne spécifié
- ${fn:toLowerCase()}: Convertit tous les caractères de la chaîne en minuscule
- ${fn:toUpperCase()}: Convertit tous les caractères de la chaîne en majuscule
- ${fn:trim()}: Supprime les espaces au début et à la fin de la chaîne

III. Création de taglib en utilisant JSTL:

1. Configuration :

La classe javax.servlet.jsp.jstl.core.Config définit un certain nombre de méthode statique permettant d'accéder/modifier les différents éléments de la configuration.

Config.find(PageContext,String)	Recherche la valeur de la configuration dans l'ordre des scopes.
Config.get(PageContext,String) Config.get(ServletRequest,String) Config.get(HttpSession,String) Config.get(Servletcontext,String)	Recherchent la valeur de la configuration dans le scope spécifié.
Config.set(PageContext,String,Object) Config.get(HttpSession,String,Object) Config.get(Servletcontext,String,Object)	Recherchent la valeur de la configuration dans le scope spécifié.

2. Tags conditionnels :

La classe javax.servlet.jsp.jstl.core.ConditionalTagSupport est une classe abstraite permettant de faire un tag conditionnel simplement. Il suffit ainsi d'implémenter le code de la méthode condition() qui déterminera si le corps doit être affiché ou pas. Il n'y a pas à se soucier du fonctionnement réel du tag.

Exemple

```
public class LoginConditionalTag extends ConditionalTagSupport {

        protected boolean condition() {
                HttpSession session = pageContext.getSession();
                if (session!=null && session.getAttribute("login-info"))
                        return true;
                return false;
        }

}
```

3. Tags itératifs :

La classe javax.servlet.jsp.jstl.core.LoopTagSupport est une classe abstraite permettant de faire un tag itératif simplement sans se soucier du fonctionnement réel du tag.

Il suffit d'implémenter les méthodes prepare(), hasNext(), et next() qui permettent respectivement de :

- Préparer les éléments de la boucle.
- Indiquer si il y a encore un élément.
- Accéder à l'élément suivant.

Ainsi, le tag appellera la méthode prepare() à l'initialisation, puis les méthodes hasNext() et next() autant de fois que nécessaire.

4. Accès aux données localisées :

La classe javax.servlet.jsp.jstl.fmt.LocaleSupport permet elle d'accéder aux données localisées de la même manière que le tag <fmt:message/>. Elle possèdent en effet plusieurs méthodes statiques getLocalizedMessage() afin d'accéder aux informations des ResourceBundles dans la locale courante en précisant ou non des arguments et/ou le nom du ResourceBundle :

- getLocalizedMessage(PageContext pageContext, String key)
- getLocalizedMessage(PageContext pageContext, key, Object[] args)
- getLocalizedMessage(PageContext pageContext, String key, String basename)
- getLocalizedMessage(PageContext pageContext, key, Object[] args, String basename)

Par exemple, pour accéder à un message du ResourceBundle par défaut, il suffit d'utiliser le code suivant :

```
String message = LocaleSupport.getLocalizedMessage(pageContext, "message.key");
```

IV. Conclusion:

En apportant la plupart des fonctionnalités de bases d'une application web, la JSTL devrait s'imposer dans le développement d'application J2EE. En effet, bien que de nombreuses librairies de tags proposent déjà les mêmes fonctionnalités, la JSTL ne se présente pas comme une énième librairie concurrente, mais propose une harmonisation de toutes ces librairies.

Il y a de fortes chances que les prochains frameworks J2EE s'appuie sur la JSTL afin de se concentrer sur leurs fonctionnalités propres. Ainsi, le framework Struts de Jakarta propose déjà une version compatible avec la JSTL 1.0 .

En effet, la JSTL propose également la possibilité d'utiliser ses ressources (Locale, ResourceBundle,DataSource,...) de manière portable (indépendamment de son implémentation). De ce fait un frameworks ou une librairie basée sur la JSTL peuvent permettre un déploiement et une intégration simple dans un projet JSP/JSTL.

Enfin, le tiercé JSTL/EL/Taglibs change radicalement la conception de pages JSP, ce qui peut troubler les développeurs Java. En effet, les scriptlets Java sont amenées à disparaître et les pages JSP s'apparentes désormais plus à des fichiers XML.