**Web.xml pour config :**

<jsp-config>

<jsp-property-group>

<url-pattern>\*.jsp</url-pattern>

<el-ignored>true</el-ignored>

</jsp-property-group>

</jsp-config>

**Objet Implicites de la JSPype** de l'objet

Description

| **Identifiant** | **Type de l'objet** | **Description** |
| --- | --- | --- |
| **pageContext** | PageContext | Il fournit des informations utiles relatives au contexte d'exécution. Entre autres, il permet d'accéder aux attributs présents dans les différentes portées de l'application. Il contient également une référence vers tous les objets implicites suivants. |
| **application** | ServletContext | Il permet depuis une page JSP d'obtenir ou de modifier des informations relatives à l'application dans laquelle elle est exécutée. |
| **session** | HttpSession | Il représente une session associée à un client. Il est utilisé pour lire ou placer des objets dans la session de l'utilisateur courant. |
| **request** | HttpServletRequest | Il représente la requête faite par le client. Il est généralement utilisé pour accéder aux paramètres et aux attributs de la requête, ainsi qu'à ses en-têtes. |
| **response** | HttpServletResponse | Il représente la réponse qui va être envoyée au client. Il est généralement utilisé pour définir le Content-Type de la réponse, lui ajouter des en-têtes ou encore pour rediriger le client. |
| **exception** | Throwable | Il est uniquement disponible dans les pages d'erreur JSP. Il représente l'exception qui a conduit à la page d'erreur en question. |
| **out** | JspWriter | Il représente le contenu de la réponse qui va être envoyée au client. Il est utilisé pour écrire dans le corps de la réponse. |
| **config** | ServletConfig | Il permet depuis une page JSP d'obtenir les éventuels paramètres d'initialisation disponibles. |
| **page** | objet this | Il est l'équivalent de la référence this et représente la page JSP courante. Il est déconseillé de l'utiliser, pour des raisons de dégradation des performances notamment. |

**-------------------------------------PAREIL MAIS POUR EXPRESSION LANGUAGE-------------------------------------**

| **Catégorie** | **Identifiant** | **Description** |
| --- | --- | --- |
| JSP | **pageContext** | Objet contenant des informations sur  l'environnement du serveur. |
| Portées | **pageScope** | Une Map qui associe les noms et valeurs des attributs  ayant pour portée la page. |
|  | **requestScope** | Une Map  qui associe les noms et valeurs des attributs ayant pour portée la requête. |
|  | **sessionScope** | Une Map  qui associe les noms et valeurs des attributs ayant pour portée la session. |
|  | **applicationScope** | Une Map  qui associe les noms et valeurs des attributs ayant pour portée l'application. |
| Paramètres de requête | **param** | Une Map qui associe les noms et valeurs des paramètres  de la requête. |
|  | **paramValues** | Une Map  qui associe les noms et multiples valeurs \*\* des paramètres de la requête sous forme de tableaux de String. |
| En-têtes de requête | **header** | Une Map qui associe les noms et valeurs des paramètres  des en-têtes HTTP. |
|  | **headerValues** | Une Map  qui associe les noms et multiples valeurs \*\* des paramètres des en-têtes HTTP sous forme de tableaux de String. |
| Cookies | **cookie** | Une Map qui associe les noms et instances des cookies. |
| Paramètres d’initialisation | **initParam** | Une Map qui associe les données contenues dans les  champs <param-name> et <param-value> de  la section <init-param> du fichier web.xml. |

**JSTL CORE**

**Liste**

<c:forEach var="item" items="${tirage}" **begin**="0" **end**="9" > (Limiter éléments)

<c:out value="${item}" />

</c:forEach>

--------------------------------------------------------------

<table>

<tr>

<th>Valeur</th>

<th>Cube</th>

</tr>

<c:forEach var="i" begin="0" end="7" step="1">

<tr>

<td><c:out value="${i}"/></td>

<td><c:out value="${i \* i \* i}"/></td>

</tr>

</c:forEach>

</table>

-----------------------------------------------------------------

Pour une MAP :

<c:forEach items="${maListe}" var="news" varStatus="status">

<div class="news">

<div class="titreNews">

<c:out value="${news['titre']}" />

</div>

<div class="corpsNews">

<c:out value="${news['contenu']}" />

</div>

</div>

</c:forEach>

----------------------------------------------------------------------------

Chaine de caractères

<c:forTokens var="sousChaine" items="salut; je suis un,gros;zéro+!" delims=";,+">

${sousChaine}<br/>

</c:forTokens>

**Affectation Variable**

<c:set var="message" value="Salut les zéros !" scope="request" />

<c:out value="${requestScope.client.nom}" default="Aucune Valeur n'a été rentrée"/>

**Affectation en variable Session**

<c:set var="locale" scope="session">

<c:out value="${param.lang}" default="FR"/>

</c:set>

**MODIFICATION**

<c:set target="${coyote}" property="prenom" value="Wile E."/>

**SUPPRESSION Variable**

<c:remove var="maVariable" scope="session" />

**Test IF**

<c:if test="${ 12 > 7 }" var="maVariable" scope="session">

Ce test est vrai.

</c:if>

**IF / ELSE IF / ELSE**

<c:choose>

<c:when test="${expression}">Action ou texte.</c:when>

...

<c:otherwise>Autre action ou texte.</c:otherwise>

</c:choose>

**LIEN**

<c:url value="/monSiteWeb/countZeros.jsp">

<c:param name="nbZeros" value="${countZerosBean.nbZeros}"/>

<c:param name="date" value="22/06/2010"/>

</c:url>

**REDIRECTION**

<c:redirect url="http://www.siteduzero.com">

<c:param name="mascotte" value="zozor"/>

<c:param name="langue" value="fr"/>

</c:redirect>

**INCLUSION Page**

<%-- Importer une page distante dans une variable

Le scope par défaut est ici page si non précisé. --%>

<c:import url="http://www.siteduzero.com/zozor/biographie.html" var="bio" scope="page"/>

<%-- Les attributs valables pour <c:url/> le sont aussi pour la redirection.

Ici par exemple, l'utilisation de paramètres --%>

<c:import url="footer.jsp">

<c:param name="design" value="bleu"/>

</c:import>

**JSTL XML**

<news>

<article id=*"1"*>

<auteur>Pierre</auteur>

<titre>Foo...</titre>

<contenu>...bar !</contenu>

</article>

<article id=*"27"*>

<auteur>Paul</auteur>

<titre>Bientôt un LdZ J2EE !</titre>

<contenu>Woot ?</contenu>

</article>

<article id=*"102"*>

<auteur>Jacques</auteur>

<titre>Coyote court toujours</titre>

<contenu>Bip bip !</contenu>

</article>

</news>

**Parse le doc XML**

<!-- Contenu Brut -->

<c:import url=*"monDocument.xml"* varReader=*"monReader"*>

<!-- Contenu parsé XML : on stocke la structure du XML dans le var-->

<x:parse var=*"doc"* doc=*"*${monReader}*"* scope=*"request"*/>

<x:out select=*"count($doc/news/article/auteur)"* />

</c:import>

l'expression XPath *($doc/news/article/auteur)* renvoie bel et bien un ensemble de nœuds, en l'occurrence les nœuds "auteur" ; cet ensemble de nœuds est stocké dans une structure de type NodeSet, un type propre à XPath qui implémente le type Java standard NodeList.

Dans une expression XPath, pour faire référence à une variable nommée nomVar on n'utilise pas${nomVar} comme c'est le cas dans une EL, mais $nomVar

<%-- Sélectionne le nœud 'article' dont l'attribut 'id' a pour valeur le contenu de la variable

nommée 'idArticle' qui a été passée en paramètre de la requête, via l'objet implicite param --%>

<x:out select="$doc/news/article[@id=$param:idArticle]"/>

**IF**

<%-- Afficher le titre de la news postée par 'Paul' --%>

<x:if select="$doc/news/article[auteur='Paul']">

Paul a déjà posté une news dont voici le titre :

<x:out select="$doc/news/article[auteur='Paul']/titre" />

</x:if>

**IF / ELSE IF**

<%-- Affiche le titre de la news postée par 'Nicolas' si elle existe, et un simple message sinon --%>

<x:choose>

<x:when select="$doc/news/article[auteur='Nicolas']">

Nicolas a déjà posté une news dont voici le titre :

<x:out select="$doc/news/article[auteur='Nicolas']/titre" />

</x:when>

<x:otherwise>

Nicolas n'a pas posté de news.

</x:otherwise>

</x:choose>

**BOUCLE**

<!-- Affiche les auteurs et titres de tous les articles -->

<p>

<x:forEach var="element" select="$doc/news/article">

<strong><x:out select="$element/auteur" /></strong> :

<x:out select="$element/titre" />.<br/>

</x:forEach>

</p>

**FORMAT XSL**

<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Mise en forme avec XSLT</title>

</head>

<body>

<table width="1000" border="1" cellspacing="0" cellpadding="0">

<tr>

<th scope="col">Id</th>

<th scope="col">Auteur</th>

<th scope="col">Titre</th>

<th scope="col">Contenu</th>

</tr>

<xsl:for-each select="/news/article">

<tr>

<td>

<xsl:value-of select="@id" />

</td>

<td>

<xsl:value-of select="auteur" />

</td>

<td>

<xsl:value-of select="titre" />

</td>

<td>

<xsl:value-of select="contenu" />

</td>

</tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

Application transformation :

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

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

<c:import varReader="xslFile" url="test.xsl">

<c:import varReader="xmlFile" url="monDocument.xml">

<x:transform doc="${xmlFile}" xslt="${xslFile}"/>

</c:import>

</c:import>

----------------------------------------------------------------------------------------------------

<c:import var="xslFile" url="test.xsl"/>

<c:import var="xmlFile" url="monDocument.xml"/>

<x:transform doc="${xmlFile}" xslt="${xslFile}">

<x:param name="couleur" value="orange" />

</x:transform>

**WEB.XML**

<web-app>

<jsp-config>

<jsp-property-group>

<url-pattern>\*.jsp</url-pattern>

<include-prelude>/WEB-INF/taglibs.jsp</include-prelude>

</jsp-property-group>

</jsp-config>

**FORMULAIRES**

<p class="${empty erreurs ? 'succes' : 'erreur'}">${resultat}</p> // Test Unitaire sur class

La solution qui s'offre à nous est l'utilisation de la directive **page**. Voici la ligne à ajouter en début de page pour empêcher la création d'une session :

<%@ page session="false" %>

Dans notre page **connexion.jsp**, nous allons modifier une ligne de code :

<!-- Dans la page connexion.jsp, remplacez la ligne suivante : -->

<form method="post" action="connexion">

<!-- Par cette ligne : -->

<form method="post" action="<c:url value="/connexion" />">

Si vous reconnaissez ici [la balise <c:url> de la JSTL Core](http://www.siteduzero.com/tutoriel-3-285009-la-bibliotheque-core.html#ss_part_4), vous devez également vous souvenir qu'elle est équipée pour la gestion automatique des sessions. Je vous avais en effet déjà expliqué que cette balise avait l'effet suivant :

<%-- L'url ainsi générée --%>

<c:url value="test.jsp" />

<%-- Sera rendue ainsi dans la page web finale,

si le cookie est présent --%>

test.jsp

<%-- Et sera rendue sous cette forme si le cookie est absent --%>

test.jsp;jsessionid=BB569C7F07C5E887A4D

**Modifier le mode de déclenchement d'un filtre**

Je vous ai implicitement fait comprendre à travers ces quelques exemples qu'un filtre était déclenché lors de la réception d'une requête HTTP uniquement. Eh bien sachez qu'il s'agit là d'un comportement par défaut ! En réalité, un filtre est tout à fait capable de s'appliquer à un *forwarding*, mais il faut pour cela modifier sa déclaration dans le fichier **web.xml** :

<filter>

<filter-name>RestrictionFilter</filter-name>

<filter-class>com.sdzee.filters.RestrictionFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>RestrictionFilter</filter-name>

<url-pattern>/\*</url-pattern>

<dispatcher>REQUEST</dispatcher>

<dispatcher>FORWARD</dispatcher>

</filter-mapping>

Il suffit, comme vous pouvez l'observer, de rajouter un champ <dispatcher> à la fin de la section<filter-mapping>.

De même, si dans votre projet vous mettez en place des inclusions et souhaitez leur appliquer un filtre, alors il faudra ajouter cette ligne à la déclaration du filtre :

<dispatcher>INCLUDE</dispatcher>

Nous n'allons pas nous amuser à vérifier le bon fonctionnement de ces changements. Retenez simplement qu'il est bel et bien possible de filtrer les *forwardings* et inclusions en plus des requêtes directes entrantes, en modifiant au cas par cas les déclarations des filtres à appliquer. Enfin, n'oubliez pas que ces ajouts au fichier**web.xml** ne sont pris en compte qu'après un redémarrage du serveur.

**My SQL**

**Se Connecter**

Sudo service mysql start

Mysql -u root -p

**Créer Database**

CREATE DATABASE bdd\_sdzee DEFAULT CHARACTER SET utf8 COLLATE utf8\_general\_ci;

**Créer Utilisateur**

CREATE USER 'java'@'localhost' IDENTIFIED BY 'SdZ\_eE';

GRANT ALL ON bdd\_sdzee.\* TO 'java'@'localhost' IDENTIFIED BY 'SdZ\_eE';

L'appel à cette méthode peut retourner des erreurs de type [SQLException](https://docs.oracle.com/javase/6/docs/api/java/sql/SQLException.html) :

* si une erreur **SQLException: No suitable driver** est envoyée, alors cela signifie que le driver JDBC n'a pas été chargé ou que l'URL n'a été reconnue par aucun des drivers chargés par votre application ;
* si une erreur **SQLException: Connection refused** ou **Connection timed out** ou encore**CommunicationsException: Communications link failure** est envoyée, alors cela signifie que la base de données n'est pas joignable.

Si un de ces derniers cas survient, vous trouverez ci-dessous une liste des causes possibles et les pistes de résolution associées :

| **Cause éventuelle** | **Piste de résolution** |
| --- | --- |
| Le serveur MySQL est éteint ? | Démarrez le serveur MySQL... |
| Le numéro de port dans l'URL est manquant ou incorrect ? | Ouvrez le fichier de configuration **my.cnf** de votre serveur MySQL, et vérifiez le port qui y est spécifié. |
| Le nom d'hôte ou l'adresse IP dans l'URL est incorrect(e) ? | Testez la connectivité en effectuant un simple ping. |
| Le serveur MySQL n'accepte pas de connexions TCP/IP ? | Vérifiez que MySQL a été lancé sans l'option --skip-networking. |
| Il n'y a plus aucune connexion disponible sur le serveur MySQL ? | Redémarrez MySQL, et corrigez le code de votre application pour qu'il libère les connexions efficacement. |
| Quelque chose entre l'application Java et le serveur MySQL bloque la connexion, comme un pare-feu ou un proxy ? | Configurez votre pare-feu et/ou proxy pour qu'il(s) autorise(nt) le port écouté par votre serveur MySQL. |
| Le nom d'hôte dans l'URL n'est pas reconnu par votre serveur DNS local ? | Utilisez l'adresse IP dans l'URL au lieu du nom d'hôte, ou actualisez si possible votre DNS. |
|  |  |

/\* Connexion à la base de données \*/

String url = "jdbc:mysql://localhost:3306/bdd\_sdzee";

String utilisateur = "java";

String motDePasse = "$dZ\_£E";

Connection connexion = null;

try {

connexion = DriverManager.getConnection( url, utilisateur, motDePasse );

/\* Ici, nous placerons nos requêtes vers la BDD \*/

/\* ... \*/

} catch ( SQLException e ) {

/\* Gérer les éventuelles erreurs ici \*/

} finally {

if ( connexion != null )

try {

/\* Fermeture de la connexion \*/

connexion.close();

} catch ( SQLException ignore ) {

/\* Si une erreur survient lors de la fermeture, il suffit de l'ignorer. \*/

}

}

/\* Création de l'objet gérant les requêtes \*/

Statement statement = connexion.createStatement();

/\* Exécution d'une requête de lecture \*/

ResultSet resultat = statement.executeQuery( "SELECT id, email, mot\_de\_passe, nom FROM Utilisateur;" );

/\* Exécution d'une requête d'écriture \*/

int statut = statement.executeUpdate( "INSERT INTO Utilisateur (email, mot\_de\_passe, nom, date\_inscription) VALUES ('jmarc@mail.fr', MD5('lavieestbelle78'), 'jean-marc', NOW());" );

/\* Exécution d'une requête de lecture \*/

ResultSet resultat = statement.executeQuery( "SELECT id, email, mot\_de\_passe, nom FROM Utilisateur;" );

/\* Récupération des données du résultat de la requête de lecture \*/

while ( resultat.next() ) {

int idUtilisateur = resultat.getInt( "id" );

String emailUtilisateur = resultat.getString( "email" );

String motDePasseUtilisateur = resultat.getString( "mot\_de\_passe" );

String nomUtilisateur = resultat.getString( "nom" );

/\* Traiter ici les valeurs récupérées. \*/

}

Connection connexion = null;

Statement statement = null;

ResultSet resultat = null;

try {

/\*

\* Ouverture de la connexion, initialisation d'un Statement, initialisation d'un ResultSet, etc.

\*/

} catch ( SQLException e ) {

/\* Traiter les erreurs éventuelles ici. \*/

} finally {

if ( resultat != null ) {

try {

/\* On commence par fermer le ResultSet \*/

resultat.close();

} catch ( SQLException ignore ) {

}

}

if ( statement != null ) {

try {

/\* Puis on ferme le Statement \*/

statement.close();

} catch ( SQLException ignore ) {

}

}

if ( connexion != null ) {

try {

/\* Et enfin on ferme la connexion \*/

connexion.close();

} catch ( SQLException ignore ) {

}

}

}

/\* Exécution d'une requête d'écriture avec renvoi de l'id auto-généré \*/

int statut = statement.executeUpdate( "INSERT INTO Utilisateur (email, mot\_de\_passe, nom, date\_inscription) VALUES ('jmarc2@mail.fr', MD5('lavieestbelle78'), 'jean-marc', NOW());" , Statement.RETURN\_GENERATED\_KEYS);

/\* Formatage pour affichage dans la JSP finale. \*/

messages.add( "Résultat de la requête d'insertion : " + statut + "." );

/\* Récupération de l'id auto-généré par la requête d'insertion. \*/

resultat = statement.getGeneratedKeys();

/\* Parcours du ResultSet et formatage pour affichage de la valeur qu'il contient dans la JSP finale. \*/

while ( resultat.next() ) {

messages.add( "ID retourné lors de la requête d'insertion :" + resultat.getInt( 1 ) );

}

Pour pallier à l’**injection SQL,** on utilise les **requêtes préparées**. Solution :

/\* Création de l'objet gérant la requête préparée définie \*/

PreparedStatement preparedStatement = connexion.prepareStatement( "SELECT id, email, mot\_de\_passe, nom FROM Utilisateur;" );

Elles sont précompilées !!!!

preparedStatement.executeQuery();

**Exemple :**

/\* Création de l'objet gérant les requêtes préparées \*/

preparedStatement = connexion.prepareStatement( "INSERT INTO Utilisateur (email, mot\_de\_passe, nom, date\_inscription) VALUES(?, MD5(?), ?, NOW());" );

/\* Récupération des paramètres d'URL saisis par l'utilisateur \*/

String paramEmail = request.getParameter( "email" );

String paramMotDePasse = request.getParameter( "motdepasse" );

String paramNom = request.getParameter( "nom" );

/\*

\* Remplissage des paramètres de la requête grâce aux méthodes

\* setXXX() mises à disposition par l'objet PreparedStatement.

\*/

preparedStatement.setString( 1, paramEmail );

preparedStatement.setString( 2, paramMotDePasse );

preparedStatement.setString( 3, paramNom );

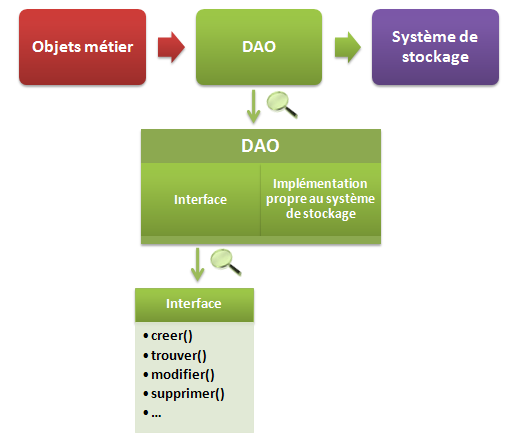
/\* Exécution de la requête \*/

int statut = preparedStatement.executeUpdate();

/\* Création d'un PreparedStatement avec renvoi de l'id auto-généré \*/

PreparedStatement preparedStatement = connexion.prepareStatement( "INSERT INTO Utilisateur (email, mot\_de\_passe, nom, date\_inscription) VALUES ('jmarc@mail.fr', MD5('lavieestbelle78'), 'jean-marc', NOW());", Statement.RETURN\_GENERATED\_KEYS );

**DAO**



Le principe du pattern DAO est de séparer la couche modèle d'une application en deux sous-couches distinctes :

* une couche gérant les traitements métier appliqués aux données, souvent nommée couche service ou métier. Typiquement, tout le travail de validation réalisé dans nos objets **InscriptionForm** et **ConnexionForm** en fait partie ;
* une couche gérant le stockage des données, logiquement nommée couche de données. Il s'agit là des opérations classiques de stockage : la création, la lecture, la modification et la suppression. Ces quatre tâches basiques sont souvent raccourcies à l'anglaise en CRUD.

Protection de mots de passe : Bibliothèque jasypt