Servlets et JSP : la JSTL

1 Utiliser le langage d'expression (EL)

Documentions :

Présentation du langage d'expressions `http://adiguba.developpez.com/tutoriels/j2ee/jsp/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. La syntaxe est la suivante :

```
${ expression }
```

- Commencez par vous amuser avec les exemples d'expressions [1] livrés avec tomcat.
- Prenez un peu de temps pour lire la présentation du langage d'expressions (voir lien ci-dessus).
- Vous pouvez maintenant modifier votre TP précédent en utilisant les *expressions* à la place des *scriptlets* pour initialiser les champs de votre formulaire et/ou visualiser les propriétés du bean personne.

2 Java Standard Tag Library

2.1 Récupérer et installer la JSTL 1.2

Documentations :

La page sur la JSTL chez Java Soft `http://www.oracle.com/technetwork/java/index-jsp-138231.html`

Les transparents sur les librairies de balises sont disponibles [2]

La documentation de la JSTL 1.2 `http://tomcat.apache.org/taglibs/standard/apidocs/`

Un cours sur la JSTL `http://adiguba.developpez.com/tutoriels/j2ee/jsp/jstl/`

Travail :

- Ajoutez à votre projet Web la fonction *Maven*.
- Ajoutez à votre fichier `pom.xml` la dépendance vers l'API JSTL [3] (partie spécification).
- Ajoutez à votre fichier `pom.xml` la dépendance vers la version Apache de la JSTL [4] (partie implantation).
- Ajoutez à votre fichier `pom.xml` la dépendance vers le Driver JDBC MySQL [5].
- Vous pouvez déployer les WAR d'exemples [6] livrés avec cette implantation de la JSTL en les **recopiant dans le répertoire** `webapps` de `Tomcat`.
- Vous pouvez notamment tester les exemples SQL.

Remarque : Je vous encourage fortement à lire le cours sur la JSTL (lien ci-dessus). Il explique bien le fonctionnement de ses balises et donne de nombreux exemples.

1. http://localhost:8080/examples/jsp/
2. jsp.html#taglib
3. http://mvnrepository.com/artifact/org.apache.taglibs/taglibs-standard-spec
4. http://mvnrepository.com/artifact/org.apache.taglibs/taglibs-standard-impl
5. http://mvnrepository.com/artifact/mysql/mysql-connector-java
6. ref:ress-jsp

2.2 Utiliser la JSTL

Pour utiliser la JSTL 1.2.x dans une application basée sur un conteneur WEB respectant le standard JSP 2.0, il suffit de copier dans le répertoire `WEB-INF/lib` de votre application WEB les fichiers `*.jar` qui se trouvent dans la distribution de la JSTL. **Ce travail est déjà réalisé par Maven**.

Il est notamment inutile de copier les fichiers `.tld`. Ceux-ci se trouvent dans les `.jar` et sont pris en compte automatiquement. Une fois le contexte de votre application relancé, vous pouvez essayer la page suivante :

```
<html>
<body>

<!-- controle, iterations, tests, variables -->
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!-- traitement XML -->
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<!-- formattage des donnees -->
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!-- SQL/JDBC -->
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>


<p>Les parametres:</p>

<ul>
<c:forEach var="aParam" items="${param}">
  <li>un parametre :
    <c:out value="${aParam.key}"/> = <c:out value="${aParam.value}"/>
  </li>
</c:forEach>
</ul>

<c:choose>
  <c:when test="${param['question'] == 'oui'}">
     <p>OUI</p>
  </c:when>
  <c:otherwise>
     <p>NON</p>
  </c:otherwise>
</c:choose>

</body>
</html>
```

Travail :

- Modifiez vos pages JSP afin d'utiliser la balise `c:out` (voir exemple ci-dessus). Vous ne devriez plus avoir d'injection de code HTML.
- Utilisez dans vos pages JSP la balise `c:url` pour construire les URL des liens vers les servlets ou les autres pages JSP (exemple ci-dessous). Vérifiez le code généré par cette balise.

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

...

<c:url var="lister" value="/lister.jsp" />

<p><a href="${lister}">Lister</a></p>

...
```

- Ajoutez à la définition de la personne (TP précédent) une propriété `statut` qui peut être « Etudiant », « Enseignant » ou « Extérieur ».
- Nous allons modifier votre formulaire pour traiter cette propriété. Commencez (dans la page JSP) par définir un bean `statuts` de portée page qui est une `HashMap`. Lors de sa création (dans le corps du `<jsp:useBean>`), placez-y les trois statuts. Faites ensuite dans le formulaire une boucle (`c:forEach`) qui parcourt cette table et construit le code du `select` en HTML.
- Prenez soin que lors du retour au formulaire (par exemple si nous avons oublié de renseigner une propriété), le champs `statut` soit correctement initialisé (il faut mettre en place l'attribut `selected` de la balise HTML `option` qui se trouve dans le `select`).

3 Les filtres

Créez dans votre application le filtre ci-dessous (en adaptant l'URL des adresses à traiter) :

```
package fr.exemple.web;

import java.io.IOException;
import javax.servlet.DispatcherType;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;

@WebFilter(
    dispatcherTypes = {
        DispatcherType.REQUEST, DispatcherType.FORWARD,
        DispatcherType.INCLUDE, DispatcherType.ERROR
    },
    urlPatterns = { "/edition" },
    servletNames = { "simpleServlet" }
)
public class SimpleFilter implements Filter {

    public void init(FilterConfig fConfig) throws ServletException {
    }

    public void destroy() {
    }

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException
    {
        System.err.printf("Before %s\n", request);
        if (request instanceof HttpServletRequest) {
            HttpServletRequest hr = (HttpServletRequest) request;
            System.err.printf("Before Http Request %s\n", hr);
        }
        chain.doFilter(request, response);
        System.err.printf("After %s\n", request);
    }

}
```

et vérifier son fonctionnement. Il permet d'insérer des traitements automatisés avant et après chaque requête.

4 Les listeners

Ajoutez à votre application, la classe d'écoute :

```
package fr.exemple.web;

import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpSessionAttributeListener;
import javax.servlet.http.HttpSessionBindingEvent;

@WebListener
public class SimpleListener implements HttpSessionAttributeListener {

    @Override
    public void attributeAdded(HttpSessionBindingEvent event) {
    }

    @Override
    public void attributeRemoved(HttpSessionBindingEvent event) {
    }

    @Override
    public void attributeReplaced(HttpSessionBindingEvent event) {
    }

}
```

et explorez les possibilités de l'argument passé à chaque méthode. Vous pouvez également vous intéresser aux interfaces

- `ServletContextListener`
- `ServletContextAttributeListener`
- `ServletRequestListener`
- `ServletRequestAttributeListener`
- `HttpSessionListener` création/suppression de sessions