Formation Java EE

Web - JSTL, Filtre, Ecouteur

JSTL

JSTL

Un tag

- Déclarer une librairie de tags

```
<%@ taglib uri="/WEB-INF/mes-tags.tld"
prefix="mt" %>
```

- Utiliser un tag

```
<mt:tag> contenu </mt:tag>
```

JSTL

JSTL ?

- JSTL = JSP Standard Tag Library
- Une librairie de tags standard utilisable dans une page JSP.
- Ces tags permettent d'adresser les besoins récurrents d'une page JSP
  - Effectuer des itérations
  - Créer une variable
  - …
- Ils représentent généralement une alternative recommandée aux scripts Java.
    - Remplacement du code Java par du code XML

JSTL

Librairie	URI	Préfixe
Core	/jsp/jstl/core	c
Format	/jsp/jstl/fmt	fmt
XML	/jsp/jstl/xml	x
SQL	/jsp/jstl/sql	sql
Functions	/jsp/jstl/functions	fn

JSTL

Installer JSTL

- Ajouter les librairies (jstl.jar et standard.jar) dans le répertoire WEB-INF/lib (ou ajouter les dépendances Maven associées)

JSTL

Core

- Support des variables :
    - <c:set>, <c:remove>
- Structures de contrôle :
    - <c:choose>,<c:forEach>, <c:if>, …
- …

JSTL

Core

- Déclarer l'utilisation de JSTL Core

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

JSTL

Core - Variables

```
<c:set var="idUtilisateur" scope="session"
    value="${param.idUtilisateur}" />

<c:set var="monNom" scope="page" value="Jean
Jacques" />

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

JSTL

Core - URL

```
<a href="<c:url value='page.jsp' />">Lien</a>
```

JSTL

Core - URL

```
<a href="<c:url value='editionPizza.jsp?
pizza_id=${pizza.id}' />">Lien 2</a>
```

JSTL

Core - forEach

```
<c:forEach var="pizza" items="${listePizzas}">
 <tr>
  <td> ${pizza.id} </td>
  <td> ${pizza.nom} </td>
  <td> ${pizza.description} </td>
 </tr>
</c:forEach>
```

JSTL

Core - forToken

```
<!-- codes = 12, 56, 78 -->
<c:forTokens var="code" items="${codes}" delims=",">
  <li> ${code} </li>
</c:forTokens>
```

JSTL

Core - if

```
<c:if test="${!empty client}">
  <c:if test="${client.nom == 'Bob'}">
    <p>Nom :
    <c:out value="${client.nom}" /></p>
  </c:if>
  <c:if test="${client.age > 18}" >
    <p>Age (>18) :
    <c:out value="${client.age}" /></p>
  </c:if>
</c:if>
```

JSTL

Core - choose

```
<c:choose>
  <c:when test="${client.age < 18 }" >
  ...
  </c:when>
  <c:otherwise>
  ...
  </c:otherwise>
</c:choose>
```

JSTL

Format

- Définition de la langue
    - <fmt:setLocale>
- Formatage de messages
    - <fmt:bundle>,<fmt:message>, <fmt:setBundle>, …
- Formatage de dates et nombres
    - <fmt:formatNumber>, <fmt:parseNumber >, <fmt:formatDate>, <fmt:parseDate>, <fmt:setTimeZone>, <fmt:timeZone>

JSTL

Format

- Déclarer l'utilisation de JSTL Format

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

Filtre

Filtre

- Permettent de réaliser des tâches récurantes
- L'API des filtres est composée de 3 classes :
  - Filter : le filtre
  - FilterChain : une chaine de filtres
  - FilterConfig : données d'initialisation

Filtre

Exemple de filtre

Pour calculer le temps d'exécution de chaque filtrage

```
public class TimerFilter implements Filter {
   private FilterConfig config = null;
   @Override
   public void init(FilterConfig config) throws ServletException {
      this.config = config;
      config.getServletContext().log("TimerFilter initialized");
   }
   @Override
   public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
         throws IOException, ServletException {
      long before = System.currentTimeMillis();
      chain.doFilter(req, resp);
      long after = System.currentTimeMillis();
      String path = ((HttpServletRequest) req).getRequestURI();
      config.getServletContext().log(path + " : " + (after - before));
   }

   @Override
   public void destroy() {  }
}
```

il continue vers le prochain filtre

Filtre

Configurer le filtre

- Dans le fichier WEB-INF/web.xml

```
<filter>
  <filter-name>TimerFilter</filter-name>
  <filter-class>test.TimerFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>TimerFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Ecouteurs

Ecouteurs

Ecouteurs

- Les écouteurs sont des objets dont les méthodes sont invoquées en fonction du cycle de vie d'une servlet
- A la création et à la destruction d'un contexte (une application)
  - javax.servlet.ServletContextListener
- Quand on modifie les attributs du contexte
  - javax.servlet.ServletContextAttributeListener
- A la création, la suppression, le timeout d'une session
  - javax.servlet.HttpSessionListener
- A la création, modification, suppression d'un attribut de session
  - javax.servlet.HttpSessionAttributeListener

Exemple écouteur

```
public class CompteurSessionListener implements HttpSessionListener{

  @Override
  public void sessionCreated(HttpSessionEvent event) {
    Integer compteur = (Integer) event.getSession().getServletContext().getAttribute("compteur");
    event.getSession().getServletContext().setAttribute("compteur", compteur + 1);
    System.out.println("Nombre de sessions ouvertes = " + (compteur + 1));
  }

  @Override
  public void sessionDestroyed(HttpSessionEvent arg0) {}
}
```

Ecouteurs

Configurer le fichier WEB-INF/web.xml

```
<listener>
   <listener-class>test.CompteurSessionListener</listener-class>
 </listener>
```