Java Server Pages

A. Madani

madaniabdellah@gmail.com

Introduction aux JSP : Rappel d'une Servlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class FirstServlet extends HttpServlet {
   public void doPost(HttpServletRequest request,
                HttpServletResponse response)
   throws ServletException,IOException{
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
   // Récupérer les données de la requête
        String n=request.getParameter("nom");
   // Traitement
   …….
   //Affichage de srésultats
        out.println("<html><body>");
        out.println("<h1>Votre nom est:"+n+"</h1>");
        out.printlnout.println("<body></html>");
   …….
        out.close();
   }
}
```

Introduction

- Les servlets Java sont très intéressantes afin de fournir des traitements, exécutés coté serveur, sur les pages HTML à renvoyer au client.
- Cela permet, notamment, de renvoyer des pages dont le contenu provient de bases de données.
- Mais peut-être, vous avez trouvé que certains aspects de cette technologie sont ennuyeux.
- Plus précisément, au niveau de l'intégration du code HTML au sein de la classe Java.
- En effet, chaque tag Java est renvoyé au client, par l'intermédiaire de *println*. out.println("<HTML>"); out.println("<BODY>");
- Cela n'est pas d'une lisibilité absolue.
- De plus, imaginez que vous ayez un web designer dans votre équipe. Lui demander d'intervenir au niveau d'une classe de code Java n'est peut-être pas la meilleure idée.

Introduction

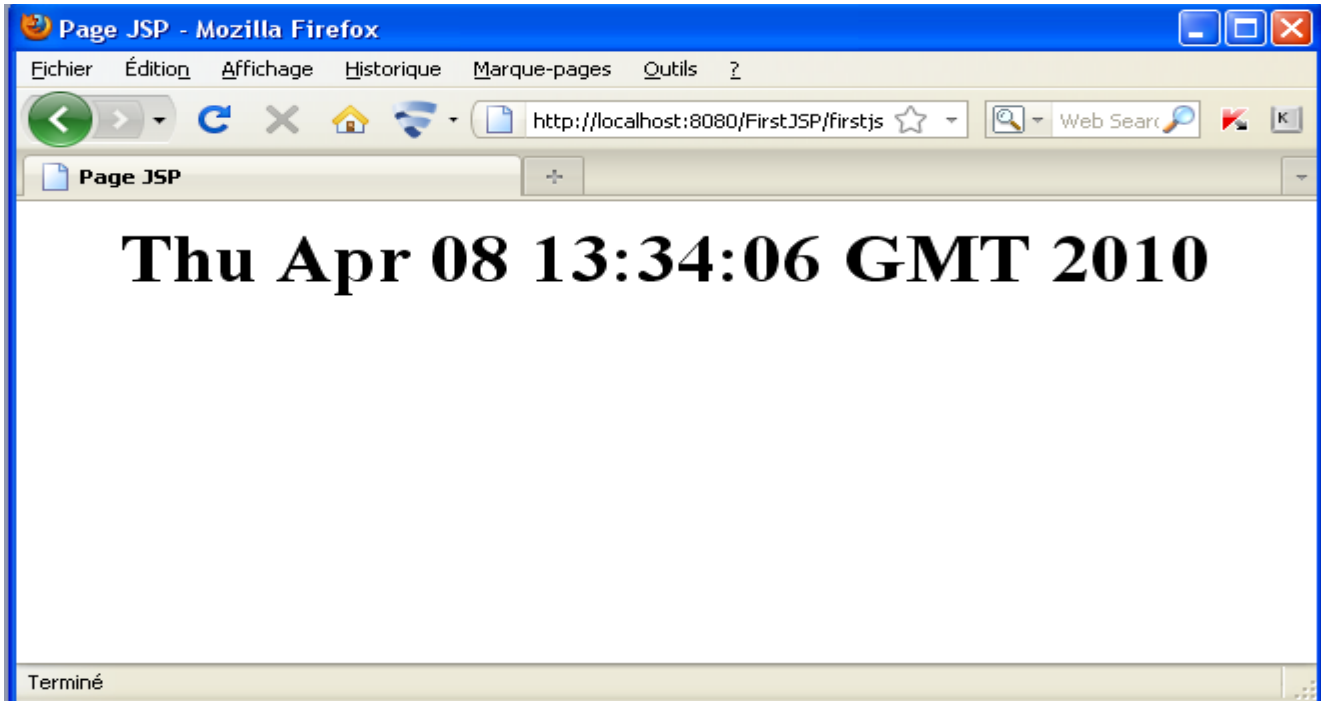
- En réponse à ce besoin de simplification de l'intégration des deux langages (HTML et Java), SUN Microsystems vous propose une seconde technologie : les JSP (Java Server Pages).
- Une page JSP ressemble plus à une page HTML qu'à une classe Java.
- Elle comporte malgré tout des bouts de code java.
- Un autre avantage sérieux à utiliser cette technologie réside dans le fait qu'elle est automatiquement transformée par le serveur d'applications en servlet Java et déployée sur le serveur.
- De plus, différentes constructions syntaxiques vous sont fournies pour répondre aux différentes exigences que vous pourriez avoir.
-
- JSP est une technologie analogue aux technologie ASP et PHP
- On peut dire que la servlet est du code html à l’intérieur du code java
- Alors que la JSP, c’est du code java à l’intérieur du code HTML

Première JSP

```
<%@ page language="java" %>
<%@ page import="java.util.*"%>
<html>
 <head>
  <title>Page JSP</title>
 </head>
 <body bgcolor="#FFFFFF">
<h1 align='center'>
  <%
   out.println(new Date());
  %>
</h1>
</body>
</html>
```

Première page JSP

- Enregistrer ce fichier sous le nom Page.jsp
- Tester votre fichier



Eléments syntaxiques d’une JSP

- Une page JSP peut être formée par les éléments suivants :
  - Les directives
  - Les scriptlets
  - Les expressions
  - Les actions

Directives

- Une directive permet de spécifier des informations qui vont servir à configurer et à influer sur le code de la servlet générée.
- Ce type de construction se repère facilement étant donné qu'une directive commence par les trois caractères <%@.
- Notons principalement deux directives :
  - *<%@ page ... %>* et
  - *<%@ include ... %>*
- Voyons de plus près quelques unes des possibilités qui vous sont offertes.

Directive <%@ page .. %>

- La directive <%@ page .. %> permet de pouvoir spécifier des informations utiles pour la génération et la compilation de la servlet.

- En fait, cette directive accepte de nombreux paramètres dont les principaux sont:
  - <%@ page language="java" %>
  - <%@ page import="package|classe %>
  - <%@ page session="true|false" %>
  - <%@ page extends="classe" %>
  - <%@ page errorPage="url" %>
  - <%@ page isErrorPage="true|false" %>

Directive <%@ include .. %>

- La directive <%@ *include ... %*> est très utile si plusieurs pages se doivent de partager un même ensemble d'information.

- C'est souvent le cas avec les entêtes et les pieds de pages. Dans ce cas, codez ces parties dans des fichiers séparés et injectez les, via cette directive, dans tous les autre fichiers qui en ont besoin.

- Voici un petit exemple d'utilisation de cette directive:

```
<%@ page language="java" %>
<HTML>
<BODY>
<%@ include file="header.jsp" %>
 <!-- Contenu de la page à générer -->
<%@ include file="footer.jsp" %>
</BODY><HTML>
```

Scriptlets

- Les scriptlets correspondent aux blocs de code introduit par le caractères <% et se terminant par %>.
- Ils servent à ajouter du code dans la méthode de service.
- Le code Java du scriptlet est inséré tel quel dans la servlet générée : la vérification, par le compilateur, du code aura lieu au moment de la compilation totale de la servlet équivalent.
- L'exemple complet de JSP présenté précédemment, comportait quelques scriptlets :

```
<%
for(int i=1; i<=6; i++) {
 out.println("<H" + i + " align=\"center\">Heading " +i+
   "</H" + i + ">");
 }
%>
```

Expressions

- Les expressions JSP sont, des expressions Java qui vont être évaluées à l'intérieur d'un appel de méthode *print*.
- Une expression commence par les caractères <%= et se termine par les caractères %>.
- Comme l'expression est placée dans un appel de méthode, il est interdit de terminer l'expression via un point-virgule.
- Syntaxe: <%=expression%>
- Equivalent à: <% out.println(expression) ; %>
- Exemple : <%=new Date()%>
- Equivalent à: <% out.println(new Date()) ; %>

Les actions

- Les actions constituent une autre façon de générer du code Java à partir d'une page JSP.
- Les actions se reconnaissent facilement, syntaxiquement parlant : il s'agit de tag XML ressemblant à <jsp:tagName ... />.
- Le tag peut, bien entendu comporter plusieurs attributs.
- Il existe plusieurs actions différentes. Les principales sont les suivantes

Les actions

- <jsp:include> : Inclusion coté serveur
  - Exemple : <jsp:include page= "entete.jsp" />
- <jsp:forward> : Poursuivre le traitement de la requête vers une autre ressource statique ou dynamique de l'application web.
  - Exemple : <jsp:forward page="affiche.jsp" />
- <jsp:useBean> : Instanciation d'un objet java (java bean)
  - Exemple :
    - <jsp:useBean id="jbName" class="TheClass" scope="session" />
- <jsp:setProperty> : Cette action, permet de modifier une propriété sur un objet créé via l'action *<jsp:useBean ...>*
  - Exemple :
    - <jsp:setProperty name="jbName" property="XXX" value="<%= javaExpression %>" />
- <jsp:getProperty> : cette action est l'inverse de la précédente : elle permet de retourner dans le flux HTML, la valeur de la propriété considérée.
  - Exemple :
    - <jsp:getProperty name="jbName" property="XXX" />

Les objets implicites

- Les objets implicites sont des variables qui référencent des objets de l'API Servlets/JSP, accessibles dans le contexte de la page JSP.

- L'avantage des objets implicites et qu'ils vous permettent d'invoquer directement leurs méthodes sans avoir à les initialiser au préalable. Cela permet donc de diminuer le nombre d'instructions de code Java dans la page JSP. Par exemple, vous n'avez pas à écrire de code Java pour récupérer le contexte **session** associé à l'utilisateur.

Les objets implicites

- L'objet request (**HttpServletRequest**)

Permet de référencer le contexte de requête HTTP, représenté par une instance du type
javax.servlet.http.HttpServletRequest

- L'objet response (**HttpServletResponse**)
- L'objet session (**HttpSession**)
- L'objet application (**ServletContext**)
- L'objet out (**javax.servlet.jsp.JspWriter**)
- L'objet config (**ServletConfig**)

JSP ou Servlet

- En fait, il n'y a pas une technologie qui est, de manière absolue, meilleur qu'une autre.
- Elles ont, toutes les deux des points forts et des inconvénients. Dans les deux cas, l'inconvénient majeur est qu'on mixe deux types de code : du code HTML de description de la page, et du code Java implémentant le comportement souhaité.
- Or, on peut au moins dire qu'il y a deux types d'informaticiens pouvant intervenir sur le développement d'une application web : les infographistes et les développeurs.
- Chacun d'eux cherchant au maximum à éviter l'autre type de code. Le mieux est donc de chercher à séparer ces deux types d'informations. Dans ce but, on peut dire qu'il y a deux solutions intéressantes.
- Vous pouvez utiliser le modèle MVC (Modèle Vue Contrôleur). Chaque URL invoque une servlet.
- Les servlets ont pour but de construire les objets JavaBeans chargés de prendre en charge la logique métier.
- Ensuite, le contrôleur (la servlet) donne la main à la vue (la JSP) qui devra se charger de construire la réponse HTML.