

JSP – JAVA SERVER PAGES



JSP

- Objectif : simplifier l'écriture de servlets :
 - ▣ Servlets = "HTML inside Java"
 - ▣ JSP = "Java inside HTML"

- Avertissements :
 - ▣ JSP = servlet
 - Similaire à Javascript/AJAX-JQuery
 - ▣ On est vite limité sans servlet
 - Sauf en utilisant des frameworks qui reposent dessus

Servlet – Hello World

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Hello extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        PrintWriter out = res.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("Hello World");
        out.println("</body>");
        out.println("</html>");
    }
}
```

JSP – Hello World

```
<html>
<body>
<% out.println("Hello World"); %>
</body>
</html>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
  <display-name>test</display-name>
  <description>test</description>
</web-app>
```

Conversion JSP en Servlet

- Moteur Jasper compile les fichiers JSP en code Java
 - ▣ Héritage de la classe `javax.servlet.jsp.HttpJspBase`
 - ▣ Implémente la méthode `_jspService`
 - ▣ Compilé de manière classique :
 - Le premier accès à la page jsp est donc potentiellement plus lent.

- Le code source de la servlet intermédiaire n'est pas forcément conservé
 - ▣ Dépend du moteur de servlet/JSP
 - ▣ Sous Jboss elle est en principe conservée par défaut :
 - `[Jboss DIR]\standalone\tmp\work\jboss.web\default-host\...`

Conversion JSP en Servlet

```
public void _jspService(HttpServletRequest request,
    HttpServletResponse response)
    throws java.io.IOException, ServletException {

    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;
```

Conversion JSP en Servlet

```
try {
    response.setContentType("text/html");
    response.setHeader("X-Powered-By", "JSP/2.1");
    pageContext = _jspxFactory.getPageContext(this, request, response, null, true, 8192, true);
    _jspx_page_context = pageContext;
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    _jspx_out = out;
    _jspx_resourceInjector = (org.glassfish.jsp.api.ResourceInjector)
application.getAttribute("com.sun.appserv.jsp.resource.injector");

    out.write("<html>\r\n");
    out.write("<body>\r\n");
    out.println("Hello World");
    out.write("\r\n");
    out.write("</body>\r\n");
    out.write("</html>\r\n");
    out.write("\r\n");
} catch ...
```

Conversion JSP en Servlet

- 3 méthodes essentielles:
 - `jspInit()` appelée après le premier chargement
 - `_jspService()` appelée à chaque appel à la JSP
 - `jspDestroy()` appelée lors du déchargement

- Équivalent aux méthodes de Servlet :
 - `init()`
 - `service()`
 - `destroy()`

Les objets implicites

- Définis dans la fonction `_jspService` et donc utilisables dans du code `jsp`.
- Il suffit de regarder la servlet générée :
 - ▣ `request` et `response` : cf servlets.
 - ▣ `out` : utilisé pour envoyer la réponse au client.
 - ▣ `session` : objet commun aux pages gérant les sessions.
 - ▣ `application` : contient le contexte de la servlet.
 - ▣ `config`: permet d'accéder à la configuration de la servlet.
 - ▣ `page` : `this`.
 - ▣ `context` : informations sur l'environnement du serveur.

La classe PageContext

- Permet de manipuler une variable quelque soit le contexte
 - ▣ Object `getAttribute(String name,int scope)`
 - ▣ void `setAttribute(String name, Object o, int scope)`
 - ▣ void `removeAttribute(String name,int scope)`
 - ▣ Scopes possibles : `PAGE_SCOPE`, `REQUEST_SCOPE`, `SESSION_SCOPE`, `APPLICATION_SCOPE`
 - Cf plus loin

- Recherche dans tous les contextes :
 - ▣ Object `getAttribute(String name)`
 - ▣ void `setAttribute(String name, Object o)`
 - ▣ void `removeAttribute(String name)`

JSP ou Servlets ?

- Servlets = "HTML inside Java"
- JSP = "Java inside HTML"

- Le code JSP est compilé :
 - Potentiellement plus rapide à l'exécution qu'un langage interprété (Php ou autre)
 - Lié aussi à la qualité du code et au serveur.
 - La compilation peut prendre du temps lors de la première requête sur la ressource.
 - Les requêtes ultérieures utilisent le fichier compilé si le jsp n'a pas été modifié.

SYNTAXE JSP



Les balises JSP

- Commentaires : `<%--`
- Scriptlet : `<%`
 - ▣ Insertion du code Java : `out.println("Hello");`
- Expression : `<%=`
 - ▣ Afficher une valeur
- Déclaration : `<%!`
 - ▣ Déclarer des variables ou fonctions
- Directive : `<%@`
 - ▣ Modifier la traduction du code JSP

Commentaires JSP

- `<%-- ... --%>`
 - ▣ Pas inclus dans la réponse envoyée au navigateur
 - ▣ A utiliser pour commenter le JSP, pas le HTML

```
<html>
<body>

<!-- commentaire visible en HTML -->

<%-- commentaire JSP invisible en HTML --%>

</body>
</html>
```

Scriptlets JSP

- `<jsp:scriptlet ... />` ou `<% ... %>`
 - ▣ Inclusion de code Java dans la JSP
 - ▣ Permet de faire du code plus ou moins lisible

```
<html>
<body>
<%
for(int i=0;i<5;i++) {
%>
Hello<br>
<%
}
%>
</body>
</html>
```

```
<html>
<body>

<%
for(int i=0;i<5;i++) {
    out.println("Hello<br>");
}
%>

</body>
</html>
```

Expressions JSP

- `<jsp:expression ... />` ou `<%= ... %>`
 - ▣ Affichage d'une valeur ou expression
 - ▣ Pas de ; à la fin d'une expression

```
<%@ page import="java.util.*" %>
<html>
<body>
<%
for (Enumeration e = request.getParameterNames () ; e.hasMoreElements() ;) {
    String name= (String) e.nextElement();
%>
<%= name %> : <%= request.getParameter(name) %>
<%
}
%>
</body>
</html>
```


Déclarations JSP

- `<jsp:declaration ... />` ou `<%! ...%>`
 - ▣ Déclaration de variables d'instance ou de méthodes.
 - ▣ Le code est inséré tel quel au début de la classe générée.

```
<html>
<body>
<%!
String Chaine = "bonjour";
public String bold(String s) {return "<b>" + s + "</b>";}
%>
<%= bold(Chaine) %>
</body>
</html>
```

Un exemple

```
<html>
<body>
<!-- variable locale --%>
<% int n = 0; %>
Page chargée <%= ++n %> fois

<!-- variable d'instance --%>
<%! int m = 0; %>
Page chargée <%= ++m %> fois
</body>
</html>
```

Directives JSP

- `<jsp:directive ... />` ou `<%@ ... %>`

- **include :**
 - ▣ Inclusion d'un fichier dans la JSP au moment de la compilation
 - ▣ Equivalent à faire un copier-coller du code inclus
 - ▣ `<%@ include file = "test.html" %>`
 - ▣ `<%@ include page = "test.jsp" %>`

- **page :**
 - ▣ attributs de la page (type de contenu, librairies java...).
 - ▣ `<%@ page import="java.util.Date" contentType="application/vnd.ms-excel" isErrorPage="false" %>`

Directives JSP – nouveaux tags

- Utilisation d'autres tag du genre `<jsp:... />`

- Possibilité de définir des librairies de tags :
 - ▣ Chemin et préfixe utilisé pour une librairie de tag.
 - `<%@ taglib prefix="p" uri="WEB-INF/library.tld" %>`
 - ▣ Cf plus loin.

Inclusion dynamique

- `<jsp:include page= "/test.jsp"/>` .
 - ▣ Inclusion dynamique, effectuée au moment de la requête.
 - ▣ Possibilité de transmettre des infos à la page incluse.

```
<jsp:include page= "/test.jsp">  
  <jsp:param name= "maVariable" value= "laValeur" />  
</jsp:include>
```
 - ▣ Récupération du paramètre :

```
<% out.println(request.getParameter(maVariable));%>
```
- C'est un include de servlet : même contraintes !

Redirection dynamique

- `<jsp:forward page= "/test.jsp"/>` .
 - ▣ redirection dynamique, effectuée au moment de la requête.
 - ▣ Possibilité de transmettre des infos à la page incluse.

```
<jsp:forward page= "/test.jsp">  
  <jsp:param name= "maVariable" value= "laValeur" />  
</jsp:forward>
```
 - ▣ Récupération du paramètre :
 - `<% out.println(request.getParameter(maVariable)); %>`
- C'est un forward de servlet : même contraintes !

Directives de page

```
<%@ page language="scriptingLanguage"  
  extends="className" //étend une classe  
  import="importList" //packages à importer  
  session="true | false" // utilisation d'une session ou pas  
  buffer="none | sizekb"  
  autoFlush="true | false"  
  isThreadSafe="true | false"  
  info="info_text"  
  errorPage="error_url" // fixe une page en cas d'erreur (cf cours servlets)  
  isErrorPage="true | false"  
  contentType="ctinfo"  
  pageEncoding="peinfo"  
  isElIgnored="true | false"
```

>

Gestion des erreurs

- Plusieurs types d'erreurs :
 - ▣ Erreur au moment de la conversion de la jsp en servlet
 - Aucune solution, mis à part corriger la jsp
 - ▣ Erreur au moment de l'exécution (exception)
 - Récupérable facilement via une page d'erreur

- Dans la déclaration d'une page on peut déclarer :
 - ▣ Vers quelle page faire suivre les erreurs
 - ▣ Si la page courante est une page d'erreur

Gestion des erreurs

```
<!-- fichier index.jsp -->

<%@ page errorPage="/error.jsp" %>
<html>
<body>

<%
int i=0/0;
%>

</body>
</html>
```

```
<!-- fichier error.jsp -->

<%@ page
  isErrorPage="true"
  import="java.io.*"
  contentType="text/plain" %>

<%
exception.printStackTrace(
  new PrintWriter(out)
);
%>
```

UTILISATION D'OBJETS EN JSP : LES BEANS



Création de beans

- Bean = classe annexe utilisée dans le code JSP
 - ▣ Objectif : séparer la présentation du modèle
 - ▣ Cf la classe Chat vue en TME1
- `<jsp:useBean id= "..." class="..." scope= "..." />`
 - ▣ Instanciation de la classe et ajout dans le "contexte"
 - id : identifiant du bean.
 - class : classe associée au bean.
 - scope : portée du bean.
 - ▣ Si le bean existe déjà dans le contexte choisi alors il est récupéré et utilisable

Utilisation de beans

- `jsp:useBean` :
 - ▣ instantiation de la classe / récupération de l'objet
- `jsp:getProperty property="xxx"`
 - ▣ Obtenir la valeur d'un attribut de l'objet
 - ▣ L'objet doit contenir un getter associé : `getxxx()`
- `jsp:setProperty property="yyy" value="valeur"`
 - ▣ Modifier la valeur d'un attribut de l'objet
 - ▣ L'objet doit contenir un setter associé : `setyyy()`
- Toute méthode définie dans le bean accessible
 - ▣ `<%= monObjet.getPrice() %>`

setProperty

- Modification directe :
 - `<jsp:setProperty name="myBean" property="test" value="toto" />`
 - `<jsp:setProperty name="myBean" property="test" value="<%=myBean.getTest()+1%>" />`

- Modification via des paramètres reçus par GET ou POST :
 - De manière générale, via l'objet request :
 - `<jsp:setProperty name="myBean" property="name" value="<%=request.getParameter("nom")%>" />`
 - Plus simplement en connaissant le nom du paramètre et celui de l'attribut :
 - `<jsp:setProperty name="myBean" property="name" param="nom"/>`
 - Encore plus simplement si le paramètre et l'attribut sont identiques :
 - `<jsp:setProperty name="myBean" property="name"/>`
 - "équivalent à" `myBean.name = request.getParameter("name")`

Exemple

```
<html><body>
<!-- création du bean --%>
  <jsp:useBean id="myFirstBean" scope="session" class="fr.ntw.myBean"/>
<!-- récupération éventuelle d'un champ name envoyé par post et ajout dans le bean --%>
  <jsp:setProperty name="myFirstBean" property="name"/>
<!-- affichage du contenu du bean --%>
  <jsp:getProperty name="myFirstBean" property="name"/>

<form action="index.jsp" method=GET>
Nom : <input type=text size=20 name="name"><br>
<input type=submit value="Envoi">
</form>
</body></html>
```

```
package fr.ntw;
public class myBean {
  private String name;
  public String getName() {return name;}
  public void setName(String name) {this.name = name;}
}
```

Portée des beans

- Page :
 - ▣ Visible pour une page JSP uniquement
 - ▣ Similaire à une variable locale dans la méthode doGet

- Request :
 - ▣ Comme page mais le bean est utilisable en cas d'include ou de forward
 - ▣ Similaire à un `request.setAttribute()` avant le forward

Portée des beans

- Session :
 - ▣ Visible durant toute la vie d'une session (donc pour un utilisateur donné)
 - Utilisable pour un panier par exemple
 - ▣ Similaire à un ajout dans la session

- Application :
 - ▣ Accessible dans toute l'application (jusqu'à son redémarrage) par toutes les pages et tous les utilisateurs. A tester avec deux navigateurs.
 - Utilisable pour une connexion à une BD par exemple
 - ▣ Similaire à un ajout dans le contexte

Portée des objets - exemple

```
<html><body>
```

```
Avant : <jsp:include page="testInclude.jsp"/>
```

```
<jsp:useBean id="myBean1" scope="page" class="fr.ntw.myBean"/>
```

```
<jsp:setProperty name="myBean1" property="name" value="p"/>
```

```
<jsp:useBean id="myBean2" scope="request" class="fr.ntw.myBean"/>
```

```
<jsp:setProperty name="myBean2" property="name" value="r"/>
```

```
<jsp:useBean id="myBean3" scope="session" class="fr.ntw.myBean"/>
```

```
<jsp:setProperty name="myBean3" property="name" value="s"/>
```

```
<jsp:useBean id="myBean4" scope="application" class="fr.ntw.myBean"/>
```

```
<jsp:setProperty name="myBean4" property="name" value="a"/>
```

```
Après : <jsp:include page="testInclude.jsp"/>
```

```
</body></html>
```

Portée des objets - exemple

```
<%-- Fichier testInclude.jsp --%>

<jsp:useBean id="myBean1" scope="page" class="fr.ntw.myBean"/>
<jsp:useBean id="myBean2" scope="request" class="fr.ntw.myBean"/>
<jsp:useBean id="myBean3" scope="session" class="fr.ntw.myBean"/>
<jsp:useBean id="myBean4" scope="application" class="fr.ntw.myBean"/>

<ul>
  <li>page : <jsp:getProperty name="myBean1" property="name"/></li>
  <li>request : <jsp:getProperty name="myBean2" property="name"/></li>
  <li>session : <jsp:getProperty name="myBean3" property="name"/></li>
  <li>appli : <jsp:getProperty name="myBean4" property="name"/></li>
</ul>
```

Portée des objets - exemple

- A gauche : premier chargement de la servlet
- Au milieu : après rechargement dans le même navigateur
- A droite : après rechargement dans un autre navigateur

```
Avant :  
page : null  
request : null  
session : null  
application : null
```

```
Après :  
page : null  
request : r  
session : s  
application : a
```

```
Avant :  
page : null  
request : null  
session : s  
application : a
```

```
Après :  
page : null  
request : r  
session : s  
application : a
```

```
Avant :  
page : null  
request : null  
session : null  
application : a
```

```
Après :  
page : null  
request : r  
session : s  
application : a
```

Gestion de formulaires

- Comme avec une servlet :
 - ▣ Utilisation de `request.getParameter()`
- Utilisation de Bean :
 - ▣ Création d'une classe correspondant aux différents champs du formulaire avec les méthodes `get` et `set` pour chaque champ.
 - ▣ Cf. exemple plus tôt.

Gestion (rapide) de formulaires

```
<html>
<body>

<jsp:useBean id="myBean" scope="session" class="fr.ntw.Hello"/>
<jsp:setProperty name="myBean" property="*" />

<jsp:getProperty name="myBean" property="param1"/>
<jsp:getProperty name="myBean" property="param2"/>

<hr>
<form action="index.jsp" method=GET>
String : <input type=text size=20 name="param1"><br>
int : <input type=text size=20 name="param2"><br>
<input type=submit value="Envoi">
</form>

</body>
</html>
```

Autres solutions

- Vu ici :
 - JSP + Beans

- Autres solutions :
 - Bibliothèques de tag (à définir ou existantes)
 - Xforms (formulaires en XML)
 - ...

LIBRAIRIES DE TAGS



Objectif

- Définir de nouveaux tags pour simplifier l'écriture

- Besoins :
 - Un fichier .tld de description des tags
 - Une classe associée (cf beans) pour définir comment créer les objets associés aux tags et les affichages :
 - Implémentation de Tag ou extension de TagSupport
 - Importer la librairie dans le fichier jsp :
 - `<%@ taglib uri="/WEB-INF/DemoTags.tld" prefix="p"%>`

Fichier jsp

```
<%@ taglib uri="/WEB-INF/DemoTags.tld" prefix="p" %>

<html>
<body>

<p:firsttag name="Test" /><br/>

<p:firsttag />

</body>
</html>
```

Fichier tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>FirstTag</shortname>
  <tag>
    <name>firsttag</name>
    <tagclass>fr.ntw.tags.FirstTag</tagclass>
    <bodycontent>empty</bodycontent>
    <attribute>
      <name>name</name>
      <required>>false</required>
    </attribute>
  </tag>
</taglib>
```

Classe associée

```
package fr.ntw.tags;

import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class FirstTag implements Tag, Serializable {
    private PageContext pc = null;
    private Tag parent = null;

    public void setPageContext(PageContext p) {pc = p;}
    public void setParent(Tag t) {parent = t;}
    public Tag getParent() {return parent;}
    public void release() {pc = null; parent = null; name = null;}
```

Classe associée (suite)

```
public int doStartTag() throws JspException {
    try {
        if(name != null) {
            pc.getOut().write("Bonjour " + name);
        } else {
            pc.getOut().write("Bonjour inconnu");
        }
    } catch(IOException e) {
        throw new JspTagException("An IOException occurred.");
    }
    return SKIP_BODY;
}

public int doEndTag() throws JspException {return EVAL_PAGE;}

private String name = null;
public void setName(String s) {name = s;}
public String getName() {return name;}
```

Valeurs de retour

- `doStartTag()` :
 - ▣ `EVAL_BODY_INCLUDE` : le contenu du tag est traité (à condition qu'il y ait un contenu, cf fichier tld).
 - ▣ `SKIP_BODY` : le contenu du tag est ignoré

- `doEndTag()` :
 - ▣ `EVAL_PAGE` : le reste de la page est évalué.
 - ▣ `SKIP_PAGE` : le reste de la page n'est pas évalué

TME :

INTERACTION JSP /SERVLET

Objectifs du TME

- Création d'un panier utilisateur pour un site web :
 - ▣ Partie client :
 - Ajout, suppression d'un produit dans le panier avec quantité.
 - ▣ Partie administration :
 - Ajout, suppression, modifications des produits disponibles sur le site web.
 - En général on fait ça avec une BD, ici on pourra utiliser un bean niveau application.

- Faire :
 - ▣ les formulaires (admin/client) en jsp.
 - ▣ Les pages de traitements ajout/suppression en servlet.
 - ▣ Créer de nouveaux tags pour simplifier la conception.