JSP
Java Server Page

Yves Bekkers

Java server page - jsp

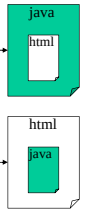
- Qu'est-ce que c'est ?
  - langage de script qui combine
    - un langage à balises (html ou xml)
    - des fragments de code java
  - chaque page est compilée en une servlet
- Défini par qui
  - sun

Exemple de page jsp

```
<%@ page language="java" %>
<html>
   <head>
      <title>Untitled</title>
   </head>
   <body>
   <h1>Résultat</h1>
   <% int francs = Integer.parseInt(
     request.getParameter("somme"));%>
   <%= francs + " Frs valent " +
     francs/6.55957 + " Euro"%>
   </body>
</html>
```

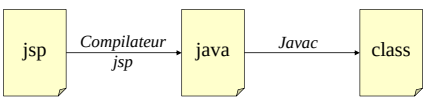

Servlet versus jsp

- Un même objectif
  - construction coté serveur de pages dynamiques
  - concurents de php, cgi, asp
- Deux logiques
  - servlet : priorité à java
  - jsp : priorité à html (ou xml)
- Un même avantage
  - ouverture sur l'univers Java



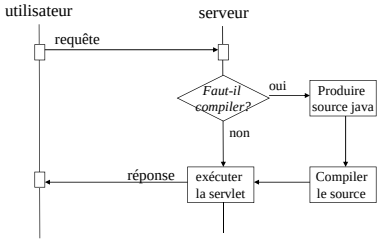
Jsp comment

- On dépose les pages jsp à coté des pages html
  - Pas de compilation au moment du dépôt
  - Visibilité identique aux pages html
  - url : pas déclaration de service
    - `nomDePage.jsp`
- Compilation automatique en dynamique sur le serveur WEB



Déroulement d'un service jsp

- La compilation ne se fait qu'une fois



Une page jsp s'occupe de la sortie

1. Lire les données expédiées par le demandeur
2. Lire les méta-informations sur la demande et le demandeur
3. Calculer la réponse (accés aux SGBD …)
4. Formater la réponse (généralement en html)
5. Envelopper la réponse http (type de document en retour, cookies …)
6. Transmettre le document dans un format adapté
   - texte (eg. html)
   - binaire (eg. gif)
   - compressé (eg. gzip)

Trois étapes simplifiées par jsp

Qu'est-ce qui est généré ?

- Un programme java contenant une classe qui implémente l'interface `HttpJspPage`
  - Une interface qui hérite de `JspPage` qui elle même hérite de `Servlet`
  - Son point d'entrée est la méthode `_jspService`

Interface `HttpJspPage`

- Le corps de la page jsp correspond au bloc de la méthode `_jspService()`
  - Point d'entrée de la page jsp
  - `void _jspService(HttpServletRequest request, HttpServletResponse response)`
  - les paramètres `request` et `response` sont disponibles au sein de la page
- Les méthodes `jspInit()` et `jspDestroy()` peuvent être redéfinie par l'auteur au sein de la page jsp

Scripts d'une page jsp

- Elements jsp:
  - *Directives*
    `<%@ ... %>`
  - *Déclarations* (attributs et méthodes)
    `<%! Déclarations de variables et de méthodes %>`
  - *Scriplets*
    `<% bloc d'instructions %>`
  - *Expressions*
    `<%= expression java %>`
  - *Commentaires* (ne sont pas produits en sortie)
    `<%-- expression java --%>`

Code généré par les scripts JSP

JSP	Code Java généré
`<html>`	`out.println("<html>");`
`<%! int i = 0; %>`	`int i = 0;`
`<% int i = 0; %>`	`int i = 0;`
`<%= i+2 %>`	`out.println(""+(i+2));`
`<!-- coucou -->`	`²out.println("<!-- coucou -->"):`
`<%-- salut --%>`	

Déclarations

```
<%!
  private int i;

  public void jspInit() {
    ...
  }
  public void jspDestroy() {
    ...
  }
%>
```

Scriptlet

Scriptlet : code java

```
<%
  Iterator i = cart.getItems().iterator();
  while (i.hasNext()) {
    ShoppingCartItem item =
      (ShoppingCartItem)i.next();
    BookDetails bd = (BookDetails)item.getItem();
  }
%>
```


La conditionnelle

```
<% if (clock.getHours() < 12) {%>
   matin
<% } else if (clock.getHours()<14) {%>
   midi
<% } else if (clock.getHours()<17) {%>
   après-midi
<% } else {%>
   soir
<%}%>
```


La conditionnelle (code généré)

```
if (clock.getHours() < 12) {
  out.println("matin");
} else if (clock.getHours()<14) {
  out.println("midi");
} else if (clock.getHours()<17) {
  out.println("après-midi");
} else {
  out.println("soir");
}
```


La boucle

```
<ol>
<%
   String[] items = cart.getItems();
   for (int i=0; i<items.length; i++)
  {
%>
<li> <%= items[i] %> </li>
<%
   }
%>
</ol>
```


HTML/JavaScript versus JSP (1)

```
<html><body>
<script type="text/javascript">
  var mycars = new Array()
  mycars[0] = "Renault"
  mycars[1] = "Peugeot"
  mycars[2] = "Citroen"
  for (i=0;i<mycars.length;i++) {
    document.write(mycars[i] + "<br />")
  }
</script>
</body></html>
```


HTML/JavaScript versus JSP (2)

```
<html><body>
<%
  String[] mycars = new String[3];
  mycars[0] = "Renault";
  mycars[1] = "Peugeot";
  mycars[2] = "Citroen";
  for (i=0;i<mycars.length;i++) {
    out.println(mycars[i] + "<br />");
  }
%>
</body></html>
```

Évaluation d'une expression

- Expression dont le résultat est sorti sur le sortie courante de la servlet

```
<%= (12+1)*2 %>
```

- Code généré

```
out.println(""+(12+1)*2);
```


Les commentaires

- Commentaires html ou xml

  générés dans la page en sortie

```
<!-- blabla -->
```

- Commentaires jsp

  Jamais générés dans le page en sortie

```
<%-- blabla --%>
```


Directives

- Syntaxe

```
<%@ directive attribut="valeur" %>
```

- Trois directives

```
page, include, Taglib
```

- Exemples

```
<%@ page language="java" %>
<%@ page buffer="5" autoFlush="false" %>
<%@ page import="java.sql.*, cart.*" %>
<%@ include file="foo.jsp" %>
<%@ taglib
     uri="http://java.sun.com/jstl/core"
     prefix="c" %>
```


Objets accessibles automatiquement au sein d'une page jsp

Nom de variable	Type java
`request`	`javax.servlet.http.HttpServletRequest`
`response`	`javax.servlet.http.HttpServletResponse`
`session`	`javax.servlet.http.HttpSession`
`application`	`javax.servlet.ServletContext`
`out`	`javax.servlet.jsp.JspWriter`
`pageContext`	`javax.servlet.jsp.PageContext`
`config`	`javax.servlet.ServletConfig`
`page`	`java.lang.Object (HttpJspPage)`
`exception`	`java.lang.Trowable`

Contexte d'une requète

Notion de context en JSP

- On a quatre contextes d'exécution imbriqués les uns dans les autres
- Application>session>requète>page
  - *Page* : Les variables déclarées dans un scriptlet sont locales à la page
  - *Requète* : Pages successives collaborant au traitement d'une même requète (voir "action" `<jsp:forward>` plus loin)
  - *session* : requètes collaborant à une même tâche
  - *application* : un ensemble de requètes regroupées sous un même thème

Extension des pages jsp
Actions

Librairies de « tags »

"*actions*" au sein de pages jsp

- *Syntaxe*
  - Eléments XML "actifs" au sein d'une page JSP
    `<prefixe:action attr="…"> … </prefixe:action>`
- *Effet*
  - Exécuter du code à chaque passage
- Deux sortes d'actions
  - Actions standards (*prefixe* `jsp`) exemple `<jsp:action>`
  - Actions définies par l'utilisateur : JSP tags (*autres prefixes*)
- Les actions sont regroupées par librairies

Actions JSP standard

- Modularité
  - `<jsp:include>` inclure un fichier en place
  - `<jsp:forward>` transférer le contrôle à une autre page
- Utiliser les beans
  - `<jsp:useBean>` trouver (instancier) un bean
  - `<jsp:getProperty>` valeur d'un attribut de bean
  - `<jsp:setProperty>` initialise un attribut de bean
- Appel d'une applet
  - `<jsp:plugin>` génére du code `<OBJET>` ou `<EMBED>`

Actions standards
modularité des pages jsp

Servir une requète
à l'aide de plusieurs pages jsp

- A quoi cela sert
  - Récupérer les erreurs
  - Simplifier le contrôle
  - ou simplement séparer contrôle et traitements
  - Modulariser les programmes
- Quels outils sont nécessaires
  - Des outils pour passer le contrôle d'une page à une autre
  - Des outils pour passer de l'information d'une page à l'autre

`<jsp:forward>`

- Action `<jsp:forward>`
  `<jsp:forward page="page1.jsp"/>`
- Attribut `page`
  - Reférence relative à la racine de l'application
    `page="/rep/page1.jsp"`
  - Reférence relative à la page d'appel
    `page="rep/page1.jsp"`
- Effet : arrête l'exécution de la page courante, lance l'exécution d'une autre page

<jsp:forward> suite

- Effets (suite)
  - L'URI d'appel reste inchangée (en cas de rechargement dans le navigateur, c'est la page d'appel qui est rechargée …)
  - Les variables request, result … restent accessibles
  - Le buffer de sortie est réinitialisé (si le transfert avait déjà commencé, une erreur est générée)
- Passer de l'information entre pages :

```
<jsp:forward page="myPage.jsp">
   <jsp:param name="nom" value="val"/>
 </jsp:forward>
```


Récupération d'erreurs

```
<% try {
…
   } catch(Exception e) {
   %>
   <jsp:forward page="erreur.jsp">
    <jsp:param name="titre" value="Erreur
 pendant la lecture de la base"/>
    <jsp:param name="message"
         value="<%= e.getMessage()%>"/>
 </jsp:forward>
<%}%>
```


<jsp:include>

- Inclure en ligne la réponse d'une autre page jsp, servlet ou page statique
  - L'URI reste celle de la page d'appel
  - Les informations sur la requète restent inchangées
  - Le buffer de sortie est vidé sur la sortie avant de commencer
  - Des paramètres peuvent être passés

```
<jsp:include page="nav.jsp">
   <jsp:param name="nom" value="val"/>
</jsp:include>
```


<jsp:plugin>

- Génération d'un élément <embed> ou <object> (en fonction du navigateur utilisé)

```
<jsp:plugin type="applet"
   code="Clok.class" jreversion="1.2"
   width="160 value="150">
   <jsp:params>
      <jsp:param name="nom" value="val"/>
   </jsp:param>
</jsp:plugin>
```


Actions standards pour utiliser les java beans

Les Beans - conventions

- bean = classe qui respecte des conventions

```
public class MonBean {
   // Attributs privés
      private type XXX;
   // constructeur
      public MonBean() {…}
    …
    // Lecture/écriture d'un attribut
   type getXXX() { … }
   setXXX(type val) { … }
}
```

Attribut caché

Constructeur (sans paramètre)

Visibilité gérée par des méthodes d'accès

Actions jsp standards – les "beans"

- Trois actions standards pour les beans

```
<jsp:useBean id="XXX" class="…"/>
<jsp:getProperty name="XXX" property="…"/>
<jsp:setProperty name="XXX" property="…"
   value="…"/>
```

- Trois éléments vides (en général) :
  - usage des paramètres pour passer l'information
  - Attention *XXX* doit être un identificateur java valide
- Trois traitements correspondants
  - *Déclaration/création d'une instance d'objet* (constructeur sans paramètre)
  - *Lecture d'un attribut de classe*
  - *Écriture d'un attribut de classe*

Création d'un bean

- Création d'un bean dans une page jsp

```
<jsp:useBean id="horloge"
      class="myPackage.DateFormatee"/>
```

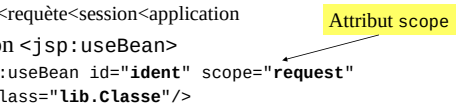
- Code Généré : deux fonctions
  - Création d'une instance d'objet

```
myPackage.DateFormatee horloge = null;
horloge = new myPackage.DateFormatee();
```

  - Réutilisation du bean : le bean est sauvé par défaut dans le contexte de la requête

Contexte d'un bean

- Quatre contextes d'exécution imbriqués (rappel)
  Page<requête<session<application
- Action `<jsp:useBean>`




```
<jsp:useBean id="ident" scope="request"
   class="lib.Classe"/>
```

- Si l'objet bean existe déjà dans la portée on utilise celui-ci et on ne crée pas de nouvel objet

Création d'un bean attribut `@scope`

- Création par défaut au sein de la requête

```
<jsp:useBean id="horloge"
    class="myPackage.DateFormatee"/>
```

- Création dans le contexte de la session

```
<jsp:useBean id="horloge"
    class="myPackage.DateFormatee"
    scope="session"/>
```


Portée (attribut `scope`)

- `scope="page"`
  - objet est visible dans la page (valeur par défaut)
- `scope="request"`
  - objet est visible au fils des jsp:forward
- `scope="session"`
  - objet est visible pour la session
- `scope="application"`
  - objet est visible pour toute l'application

Code généré à la création d'un bean

```
myPackage.DateFormatee horloge = null;
synchronized (_jspx_page_context) {
  horloge = (myPackage.DateFormatee)
  _jspx_page_context.getAttribute("horloge",
     PageContext.PAGE_SCOPE);
  if (horloge == null){
    horloge = new myPackage.DateFormatee();
    _jspx_page_context.setAttribute(
       "horloge", horloge,
       PageContext.PAGE_SCOPE
    );
  }
}
```

Attribut `@class` ou `@type`

- Au moins un attribut `class` ou `type` doit être présent
  - `class="`*nom*`"` : Un nom complet de classe
  - `type="`*nom*`"` : Un nom complet de super classe de l'objet ou d'interface implémentée

Elément `<jsp:useBean>` avec contenu

```
<jsp:useBean id="myBean" class="myClass">
   <jsp:setProperty name="myBean"
       property="prop" value="val"/>
</jsp:useBean>
```

Le corps de l'élément est exécuté une seule fois lors de la création de l'instance de l'objet bean

`<jsp:useBean>` avec contenu

```
<%@ page language="java" %>
<html><body>
  <h1>Utilisation du Bean
      <code>String</code></h1>

  <jsp:useBean id="monTexte" class="String">
     <p>On crée une chaîne nommée monTexte</p>
     <%monTexte = "blabla";%>
  </jsp:useBean>

  <p>monTexte =  <%= monTexte%></p>
</body></html>
```


`<jsp:useBean>` avec contenu - bis

```
<%@ page language="java" %>
<html><body>
  <h1>Utilisation du Bean <code>String</code></h1>

  <jsp:useBean id="monTexte" class="String">
     <p>On crée une chaîne nommée monTexte</p>
     <%monTexte = "blabla";%>
  </jsp:useBean>

  <p>monTexte =  <%= monTexte%></p>
</body></html>
```

Exemple de Bean -"*date courante formatée*"

```
package myPackage;
public class DateFormatee {
  static SimpleDateFormat sdf =
    new SimpleDateFormat(
     "EEEEEEEEE dd MMMMMMMMMMM yyyy");
  private Date dateCreation;
  public DateFormatee () {
    dateCreation = new Date();
  }
  static public String getDate() {
    return sdf.format(new Date());
  }
  public String getDateCreation() {
    return sdf.format(dateCreation);
  }
}
```

Attention « default package » interdit

Attention les attributs **static** ne sont pas des propriétés du bean

Une seule propriété

Utilisation du bean

```
<%@ page language="java" %>
<html><body>
  <h1>Utilisation du Bean DateFormatee</h1>
  <p>Date :
    <%= myPackage.DateFormatee.getDate()%>
  </p>
  <jsp:useBean id="horloge"
    class="myPackage.DateFormatee"/>
  <p>Heure de création :
    <jsp:getProperty name="horloge"
      property="dateCreation"/>
  </p>
</body></html>
```

Attention les attributs **static** ne sont des propriétés du bean

Code généré pour l'accès à une propriété d'un bean

- Code JSP

```
<jsp:getProperty name="horloge"
      property="dateCreation"/>
```

- Code Java

```
(
  (myPackage.DateFormatee)
  _jspx_page_context.findAttribute(
    "horloge")
).getDateCreation()
```


Elément <jsp:setProperty>

Trois formes d'appel

- Valeur directe

```
<jsp:setProperty name="XXX"
   property="…" value="chaîne"/>
```

- Valeur résultant du calcul

```
<jsp:setProperty name="XXX"
   property="…" value="<%= expression %>"/>
```

- Valeur provenant d'un paramètre saisi dans un formulaire

```
<jsp:setProperty name="XXX"
   property="prop" Param="param"/>
```

(*prop*:=*param*)

Elément <jsp:setProperty>

```
<jsp:setProperty name="XXX"
   property="…" value="…"/>
```

- Le type d'une propriété est quelconque
- Si la valeur est une chaîne une conversion implicite est opérée

Conversions implicites de chaîne vers un type primitif de Java

Type de propriété	Méthode de convertion
`Boolean ou boolean`	`Boolean.valueOf(String)`
`Byte ou byte`	`Byte.valueOf(String)`
`Character ou char`	`String.charAt(int)`
`Double ou double`	`Double.valueOf(String)`
`Integer ou int`	`Integer.valueOf(String)`
`Float ou float`	`Float.ValueOf(String)`
`Long ou long`	`Long.valueOf(String)`

Initialisation d'un "bean"

- *Condition* : chaque paramêtre récupéré dans le formulaire possède un attribut de même nom dans la classe bean

```
<jsp:useBean id="monObjet"
   class="lib.MaClasse">
  <jsp:setProperty name="monObjet"
     property="*"
  />
</jsp:useBean>
```

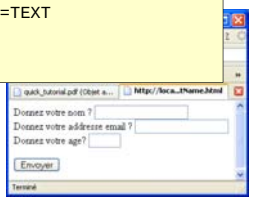
Utilisation de *

Usage des Java beans et des formulaires

exemple

Formulaire de saisie `getName.html`

```
<HTML>
<BODY>
<FORM METHOD=POST ACTION="saveName.jsp">
Donnez votre nom ? <INPUT TYPE=TEXT
      NAME="username" SIZE=20><BR>
Donnez votre addresse email ? <INPUT TYPE=TEXT
      NAME="email" SIZE=20><BR>
Donnez votre age? <INPUT TYPE=TEXT
      NAME="age" SIZE=4>
<P><INPUT TYPE=SUBMIT>
</FORM>
</BODY>
</HTML>
```



Java beans (*cf* nom des attributs)

```
public class UserData {
    String username;
    String email;
    int age;

    public void setUsername(String value)
    {
        username = value;
    }
    public void setEmail(String value) {
        email = value;
    }
    public void setAge(int value) {
        age = value;
    }
```

```
public String getUsername() {
  return username;
}
public String getEmail() {
  return email;
}
public int getAge() {
  return age;
}
}
```

Les getters et setters doivent être public, le nom des attribut est le même que dans le formulaire, la classe ne doit pas être dans le paquetage par défaut

Récolter les valeurs, page JSP `saveName.jsp`

```
<jsp:useBean id="user"
  class="test.UserData" scope="session"/>
<jsp:setProperty name="user"
  property="*"/>
<HTML>
<BODY>
<A HREF="nextPage.jsp">Continue</A>
</BODY>
</HTML>
```

Afficher les valeurs `nextPage.jsp`

```
<jsp:useBean id="user"
  class="test.UserData" scope="session"/>
<HTML>
<BODY>
You entered<BR>
Name: <%= user.getUsername() %><BR>
Email: <%= user.getEmail() %><BR>
Age: <%= user.getAge() %><BR>
</BODY>
</HTML>
```

5 librairies standard d'actions

- Classées en 5 familles
  1. Core (préfixe `c`)
  2. XML (préfixe `x`)
  3. Internationalisation i18n (préfixe `fmt`)
  4. Base de données (préfixe `sql`)
  5. Fonction diverses (préfixe `fn`)
     - sur les chaînes …

Librairies standards

- Définie par qui ?
  - Sun
- Disponibles où ?
  - Sur le site d'apache on en trouve une implémentation

URIs pour les librairies

1. *Core*: `http://java.sun.com/jsp/jstl/core`
2. *XML*: `http://java.sun.com/jsp/jstl/xml`
3. *Internationalisation*: `http://java.sun.com/jsp/jstl/fmt`
4. *SQL*: `http://java.sun.com/jsp/jstl/sql`
5. *Fonctions*: `http://java.sun.com/jsp/jstl/functions`

- Exemple de déclaration

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


Attention

- Les librairies de tag standard ne viennent pas par défaut :
  - il faut les charger sur le site d'Apache !

Éléments actifs « `core` »

Gestion de variables	`remove, set`
Gestion du contrôle	`choose (when, otherwise),` `forEach , forTokens, if`
Gestion des urls	`import (param),` `redirect (param),` `url (param)`
Divers	`catch, out`

Exemple gestion des variables

- Initialiser

```
<c:set var="bookId"
   value="${param.Remove}"/>
```

- Effacer

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

- Référencer

```
${bookId}
```


Affichage d'une liste

```
<ul>
<%   List bookList = (List) request.getAttribute ("books-list");
   if (bookList!=null) {
      Iterator iterator = bookList.iterator();
      int i = 0;
      while ( iterator.hasNext() ) {
         Book b = (Book) iterator.next();
         out.print ("<li class='"+(i%2==0?"pair":"impair")+"'>");
         out.print (b.getName()+" ("+b.getPrice()+" &euro;)");
         if ( b.getPrice() < 30.0 )
            out.print (" <img src='hot.png'
             alt='Moins de 30 &euro;'/>");
         out.println ("</li>");
         i++;
      }
   }
%>
</ul>
```


Utiliser la librairie `core`

```
<ul>
<c:forEach items="${requestScope['books-list']}"
  var="book" varStatus="status">
  <li
  class="${status.index%2==0?'pair':'impair'}">
       ${book.name} (${book.price} &euro;)
      <c:if test="${book.price < 30.0}">
         <img src='hot.png'
             alt='Moins de 30 &euro;'/>
      </c:if>
  </li>
</c:forEach>
</ul>
```

Utiliser la librairie core (bis)

```
<%
   String [] liste = Personne.listIds();
   pageContext.setAttribute ("liste", liste);
%>
<select name="personneId" size="<%= liste.length %>">
  <optgroup label="">
    <c:forEach items="${liste}" var="item">
      <option value="${item}">
        <%=
   Personne.get((String)pageContext.getAttribute("item")
   ).getNom() %>
      </option>
    </c:forEach>
  </optgroup>
</select>
```


Conditionnelle à choix multiple

```
<c:choose>
  <c:when test="${customer.category=='trial'}" >
    ...
  </c:when>
  <c:when test="${customer.category=='member'}" >
    ...
  </c:when>
    <c:when
   test="${customer.category=='preferred'}" >
    ...
  </c:when>
  <c:otherwise>
    ...
  </c:otherwise>
</c:choose>
```


Éléments actifs « xml »

Noyau	`out, parse, set`
Gestion du contrôle	`choose (when, otherwise), forEach , if`
Transformation	`transform (param),`

Evaluations xpath

```
<x:forEach var="book"
select="$applicationScope:booklist/books/*">
…
</x:forEach>
```


URL et documents xml

- Importer un document

```
<c:import url="/books.xml"
  var="xml" />
```

- Compiler le document

```
<x:parse doc="${xml}"
  var="booklist"   scope="applicatio
  n" />
```


i18n

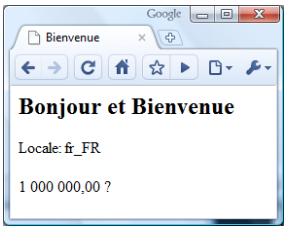
Définir de la locale	`setLocale requestEncoding`
Gestion des messages	`Bundle, message param, setBundle`
Nombres et dates	`formatNumber, formatDate parseDate, parseNumber setTimeZone, timeZone`

```
<h3><fmt:message key="Choose"/></h3>
```


Exemple1

```
<%@ taglib uri="http://java.sun.com/jstl/fmt"
  prefix="fmt" %>
<%@ page isELIgnored="false" %>
<html><fmt:setBundle basename="message"/>
<head><title><fmt:message key="Welcome" />
</title></head><body>
<h2><fmt:message key="Hello" /> <fmt:message key="and"
  /> <fmt:message key="Welcome" /></h2>
Locale:
  ${pageContext.request.locale.language}_${pageContex
  t.request.locale.country}
<br /><br />
<fmt:formatNumber value="1000000" type="currency" />
</body></html>
```

Résultat



Properties file

- `message._fr.properties` placé à la racine des sources

```
Welcome=Bienvenue
Hello=Bonjour
and=et
```


Exemple i18n

```
<jsp:useBean id="now"
  class="java.util.Date" />
<jsp:setProperty name="now"
  property="time"   value="${now.tim
  e + 432000000}" />
<fmt:message key="ShipDate"/>
<fmt:formatDate value="${now}"
  type="date"   dateStyle="full"/>.
```


EL : Expression Language

Manipuler des donnée plus simplement qu'en Java

Utilisation dans les attributs de la librairie standard JSTL

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


Où utiliser les EL

- Dans les attributs des tags JSP.
- Dans du texte simple de la page JSP.
  - Exemple

```
${monBean}
<prefix:actionTag id="newBean"
  param="${monBean}">
    ${newBean}
</prefix:actionTag>
```

Spécification

- Valable pour
  - **J2EE 1.4** (Servlet 2.4 / JSP 2.0)
  - **J2EE 1.3**. (**JSTL 1.0** -Java Standard Tag Library- avec un serveur d'application **J2EE 1.3)**
- La syntaxe est

`${ <expression> }`

- Exemples

```
${personne.age + 4}
${tab[index]% 4}
```


Quelques exemples

- **`${ pageContext.response.contentType }`**
  - affiche le content type de la réponse.
- **`${ pageScope["name"] }`**
  - affiche l'attribut "**name**" du scope page.
- **`${ param["page"] }`**
  - affiche la valeur du paramètre "**page**".
- **`${ header["user-agent"] }`**
  - affiche l'header "**user-agent**" envoyé par le navigateur.

Les objets implicites

- **pageContext** : Accès à l'objet **PageContext** de la page JSP.
- **pageScope** : Map permettant d'accéder aux différents attributs du scope '**page**'.
- **requestScope** : Map permettant d'accéder aux différents attributs du scope '**request**'.
- **sessionScope** : Map permettant d'accéder aux différents attributs du scope '**session**'.
- **applicationScope** : Map permettant d'accéder aux différents attributs du scope '**application**'.
- **param** : Map permettant d'accéder aux paramètres de la requête HTTP sous forme de **String**.
- **paramValues** : Map permettant d'accéder aux paramètres de la requête HTTP sous forme de **tableau de String**.
- **header** : Map permettant d'accéder aux valeurs du Header HTTP sous forme de **String**.
- **headerValues** : Map permettant d'accéder aux valeurs du Header HTTP sous forme de **tableau de String**.
- **cookie** : Map permettant d'accéder aux différents Cookies.
- **initParam** : Map permettant d'accéder aux **init-param**s du web.xml.

Recherche automatique d'un attribut

- `${name}`
  - Rechercher l'attribut "`name`" successivement dans les différentes portées (dans l'ordre : **page**, **request**, **session**, **application**).

Opérateurs

- Arithmétiques `+, - , *, /` (ou `div`) , `%` (ou `mod`)
- Relationnels `==` (ou `eq`) , `!=` (ou `ne`), < (ou `lt`), > (ou `gt`) , <= (ou `le`), >= (ou `ge`)
- Logiques `&&` (ou `and`) , `||` (ou `or`), `!` (ou `not[..]`)

Autres opérateurs

- `Empty[*]`

  true si l'opérande est **null**, une chaîne vide, un tableau vide, une Map vide ou une List vide. **false** sinon.
- `(c?v1:v2)`

  conditionnelle

Ecritures équivalentes

- `${ bean.name }`
- `${ bean["name"] }`
- `${ bean['name'] }`
- La valeur entre 'crochet' est une chaîne mais aussi n'importe quel objet.
- La méthode **toString()** est utilisé pour forcer le résultat à être une chaîne.
- `${ bean[ config.propertyName ] }`

Exemple d'écritures équivalentes

- `${ person.address.city }`
- `${ person.['address'].['city'] }`
- `${ person.["address"].["city"] }`
- `${ person.address.['city'] }`
- `${ person.["address"].city }`

Listes, tableaux et map

- Listes et tableaux
  - `${ list[0] }`
  - `${ list[1] }`
  - `${ list["2"] }`
  - `${ list["3"] }`
  - `${ list[ config.value ] }`
- Map
  - `${ map["clef1"] }`
  - `${ map['clef2'] }`
  - `${ map[ config.key ] }`

Prise en compte des expressions

- Dans une directive **page** ou dans un **tag** .
  - Utiliser l'attribut **isELIgnored**

```
<%@ page isELIgnored="false" %>
```

Attention elles sont ignorées par défaut …

- Utiliser anti-slash pour signifier que ceci n'est pas une expression:

```
\${ ceci n'est pas une EL }
```


Un exemple de librairie de tags dédiée

« Modèlisation de pages WEB»

Un exemple de modèle de page

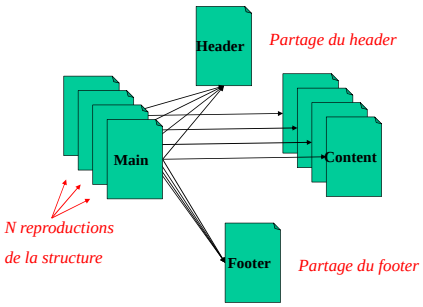
Solution 1

Utiliser la modulatité
Action `jsp:include`

Un modèle de page en JSP

```
<html><head><title>JSP Templates</title></head>
 <body background='graphics/background.jpg'>
  <table>
   <tr valign='top'>
    <td><%@include file='sidebar.html'%></td>
    <td><table>
      <tr><td><%@include file='header.html'%></td></tr>
      <tr><td><%@include file='intro.html'%></td></tr>
      <tr><td><%@include file='footer.html'%></td></tr>
     </table></td>
   </tr>
  </table>
 </body>
</html>
```


Structure du site WEB



Problèmes avec cette implémentation

- Le modèle de page est codé en dur dans la page elle même
  - Si plusieurs pages d'un site WEB ont le même modèle, celui-ci est dupliqué dans chaque page
  - Si on désire modifier la présentation il faut intervenir dans toutes les pages
- Nécessité d'un mécanisme pour inclure séparément le modèle de page et le contenu de la page

Solution 2

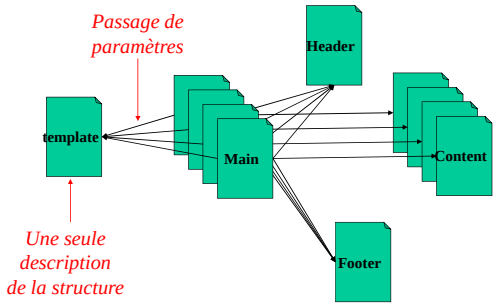
Utiliser un modèle de page unique
La libairie de tag *Template*

Un modèle de page générique

```
<%@ taglib uri='template.tld' prefix='template' %>
<html>
 <head><title><template:get name='title'/>
                                    </title></head>
 <body background='graphics/background.jpg'>
  <table>
   <tr valign='top'>
    <td><template:get name='sidebar'/></td>
    <td><table>
      <tr><td><template:get name='header'/></td></tr>
      <tr><td><template:get name='content'/></td></tr>
      <tr><td><template:get name='footer'/></td></tr>
     </table></td>
   </tr>
  </table>
 </body>
</html>
```

Déclaration de procédure
(5 paramètres)

Structurer un site WEB



Page utilisant un modèle

```
<%@ taglib uri='/WEB-INF/tlds/template.tld' prefix='template' %>

<template:insert template='/articleTemplate.jsp'>
  <template:put name='title' content='Templates'
    direct='true'/>
  <template:put name='header' content='/header.html' />
  <template:put name='sidebar' content='/sidebar.jsp' />
  <template:put name='content'
    content='/introduction.html'/>
  <template:put name='footer' content='/footer.html' />
</template:insert>
```

Appel de procédure
articleTemplate.jsp(
Templates, /header.html, /sidebar.jsp, /content.html, /footer.html
)

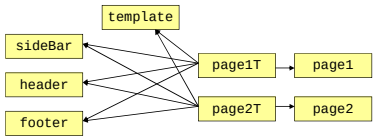
Fonctionnement

- Appel de procédure

```
<template:insert
  template='/articleTemplate.jsp'>
```

- Passage de paramètre effectif

```
<template:put name='footer'
  content='/footer.html'/>
```



Attribut direct

```
<template:put name='title' content='Templates'
  direct='true'/>
```

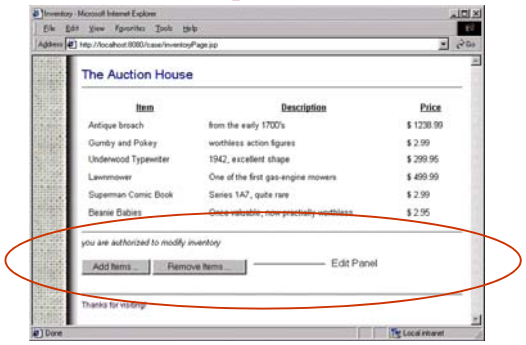
Le valeur de l'attribut `content` est directement inséré

Contenu optionnel

- Tout contenu d'un modèle est optionnel
  - Un même modèle peut donc servir pour plusieurs format …

Modèle à trois zones

Modèle à quatre zones



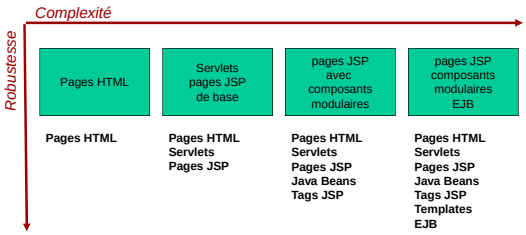
Conclusion sur les templates JSP

- Avantages
  - Paramétrage de la présentation d'un site web par l'utilisation de modèles de page
- Idées tirées de
  - *Java Tip 98: Reflect on the Visitor design pattern*
    Implement visitors in Java, using reflection, By David Geary
  - `http://www.javaworld.com`
- Où trouver la librairie
  - `http://www.javaworld.com/javaworld/jw-09-2000/jspweb/jw-0915-jspweb.zip`

Conclusion servlet/jsp ouverture sur le monde java

- Communiquer avec
  - des classes
  - des applets
- Librairies
  - jdbc
  - xml (dom, sax, transformation sxl, base de données native xml xindice)
  - soap, xmlrpc ...

Conception d'applications WEB



Références

- Rappel : Tutorial de sun (1500 pages)
  - (pdf) http://java.sun.com/j2ee/1.4/docs/tutorial-update2/doc/J2EETutorial.pdf
- Un tutorial en Français sur les tags
  - http://adiguba.developpez.com/tutoriels/j2ee/jsp/taglib/
- Un tutorial sur le développement J2EE avec Eclipse et Tomcat
  - ftp://ftp2.developpez.biz/developpo/users/tahe/fichiers/progwebjavaavececlipseettomcat.pdf