Construction d’applications web

Chapitre 4

Applicatifs côté serveur en java
Java Server Pages

Servlets : Rappel

- **Servlets :**
  - Classes Java pour la gestion des requêtes et réponses HTTP
- **Outil de base bas niveau**
  - Utilisé par l'ensemble des frameworks Web de plus haut niveau
- **Inconvénient :**
  - Approche programme pour la génération de contenu
  - Beaucoup de code pour générer des balises statiques
- **Solution :**
  - Approche donnée
  - Mettre un peu de code Java dans des balises HTML

5. Java Server Pages

JSP (Java Server Pages)

- du code Java **embarqué** dans une page HTML entre les balises <% et %>
- extension .jsp pour les pages JSP

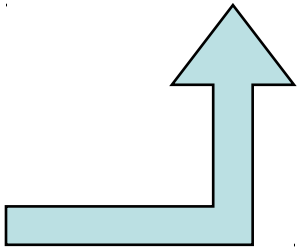
```
<html> ... <body>
<h1>Table des factorielles</h1><ul>
<%
int i, fact;
for ( i=1, fact=1 ; i<4 ;  i++, fact*=i ) {
   out.print( "<li>" + i + "! =" + fact + "</li>" );
}
%>
</ul></body> </html>
```

5. Java Server Pages

JSP (Java Server P

- du code Java **emb**
  entre les balises <
- extension .jsp pou

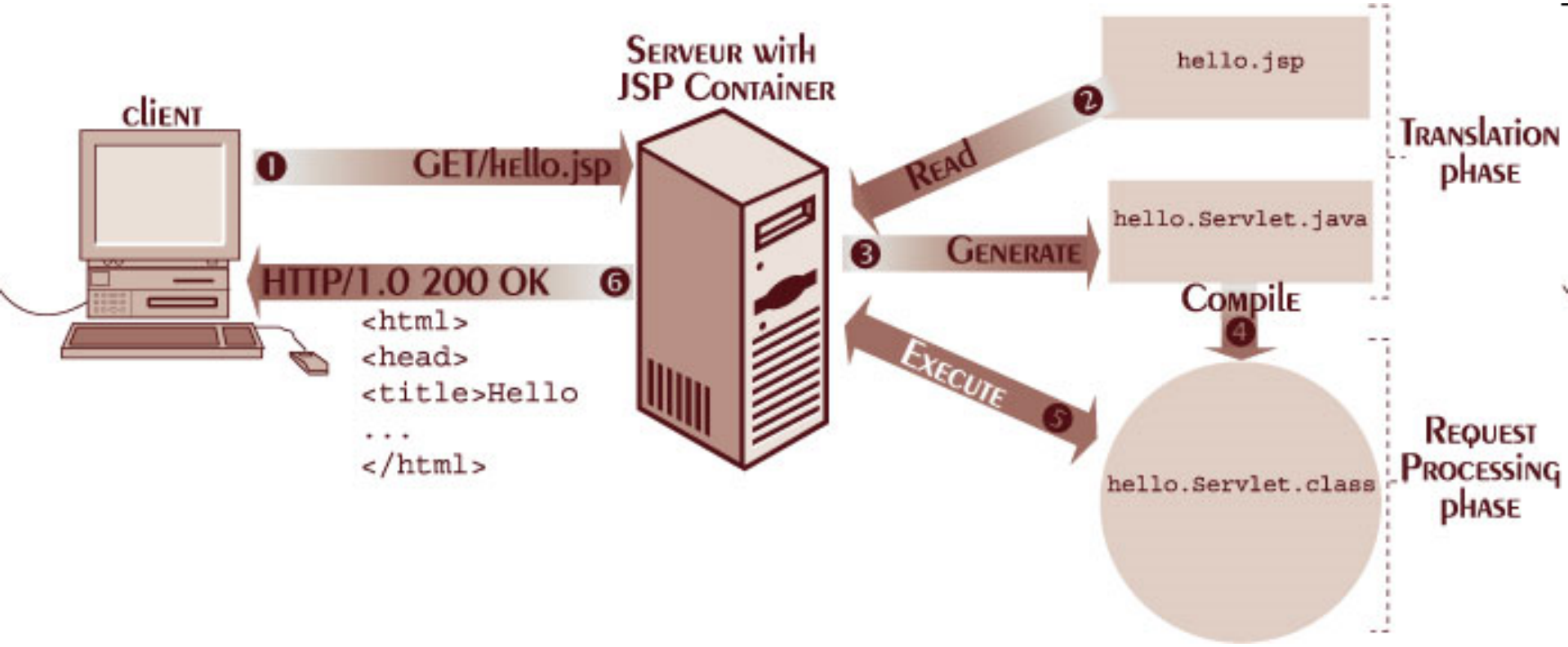

```
<html> ... <body>
<h1>Table des factorielles</h1><ul>
<%
int i, fact;
for ( i=1, fact=1 ; i<4 ;  i++, fact*=i ) {
   out.print( "<li>" + i + "! =" + fact + "</li>" );
}
%>
</ul></body> </html>
```



Invocation
Exécution côté serveur

5. Java Server Pages

- **JSP (Java Server Pages)**



5. Java Server Pages

JSP (Mécanismes mis en œuvre)

- **plusieurs** zones **<%** ... **%>** peuvent cohabiter dans une même JSP
- lors du premier chargement d'une JSP (ou après modification), le **moteur :**
  - rassemble **tous** les fragments <% ...%> de la JSP dans une classe (un servlet)
  - la **compile**
  - l'**instancie**
- puis, ou lors des chargements suivants, le **moteur**
  - exécute le code dans un ***thread***
  - délai d'attente lors de la 1ère invocation dû à la compilation
  - en cas d'erreur de syntaxe dans le code Java de la JSP : message récupéré dans le navigateur

5. Java Server Pages

JSP (Code généré)

- Un squelette de servlet standard est généré
- Le contenu de la page JSP est placé dans la méthode _jspService() (équivalent de processRequest()) :
  - tout ce qui n'est pas entre balises <% ... %> est transformé en une série de out.write(...)
  - ce qui est entre les balises est collé tel quel
- Le résultat peut donc ne pas compiler !
- Les erreurs n’apparaissent pas au déploiement mais lors de l’accès à la page par le navigateur
- Une fois qu’on a accédé à la page au moins une fois, Netbeans permet l’accès au servlet généré par une jsp (clic droit + view servlet)

5. Java Server Pages

JSP

- Directive **<%= ... %>**
- Permet l'affichage de la valeur d'une expression
- <%= *expr* %> : abréviation de
  <% out.print(expr); %>

```
<html>
    ...
    <body>
        <% int alea = (int) (Math.random() * 5); %>
        <p> <%= alea %> </p>
    </body>
</html>
```

5. Java Server Pages

JSP

- Directive **<%! ... %>**
- Le code est placé directement dans la classe générée et non dans la méthode _jspService()
- Permet la déclaration d'attributs et de méthodes

```
<html> ... <body>
<h1>Compteur</h1>
<%!
    int cpt = 0;
    int getCpt() {
        return cpt++;
    }
%>
<%= getCpt() %>
</body> </html>
```

Attribut

- apparaît directement dans le corps de la classe (variable globale)
- **persiste** entre 2 invocations tant que la JSP ne change pas (sinon elle est recompilée)

Méthode

- méthode supplémentaire de l'objet servlet généré

5. Java Server Pages

JSP

- Directive **<%@ page ... %>**
- Donne des informations à la JSP
  - **<%@ page import="..."%>**
    (exemple <%@ page import="java.io.*"%>)
    les "import" nécessaires au code Java de la JSP
  - **<%@ page errorPage="..."%>**
    (exemple <%@ page errorPage="err.jsp"%>)
    fournit l'URL de la JSP à charger en cas d'erreur (exception)
  - **<%@ page isThreadSafe="..." %>** true ou false
    true : la JSP peut être exécutée par plusieurs clients à la fois
  - **<%@ page isErrorPage="..." %>** true ou false
    true : la JSP est une page invoquée en cas d'erreur
  - ...

5. Java Server Pages

JSP

- Directive **<jsp:include …/>**
- Permet d'inclure une sortie générée par un autre fichier (autre jsp, servlet ou autre)

```
<html> ... <body>
    <jsp:include page="inc.jsp" />
</body> </html>
```

inc.jsp

```
<p>
    <%= (int) (Math.random() * 15) %>
</p>
```

La requête HTTP est transmise à l'autre fichier et le résultat est inclus à l'emplacement du jsp:include

5. Java Server Pages

JSP

- Directive **<jsp:forward .../>**
- Permet de déléguer le traitement à une autre JSP, servlet ou autre

```
<html> ... <body>
    <jsp:forward page="inc.jsp" />
</body> </html>
```

Tout est ignoré sauf la directive : la page en cours de génération est supprimée et remplacée par le résultat de inc.jsp

inc.jsp

```
<html> ... <body>
    <p>
        <%= (int) (Math.random() * 15) %>
    </p>
</body> </html>
```

5. Java Server Pages

JSP (Les objets implicites)

- Objets prédéclarés utilisables dans le code Java des JSPs
  - **out** : le flux de sortie pour générer le code HTML
  - **request** : la requête qui a provoqué le chargement de la JSP
  - **response** : la réponse à la requête de chargement de la JSP
  - **page** : l'instance de servlet associée à la JSP courante (≡ this)
  - **exception** : l'exception générée en cas d'erreur sur une page
  - **session** : suivi de session pour un même client
  - **application** : espace de données partagé entre toutes les JSP

5. Java Server Pages

JSP (Récupération des données d'un formulaire)

- Méthode **String getParameter(String)** de l'objet prédéfini **request**
  - retourne le texte saisi
  - ou **null** si le nom du paramètre n'existe pas

```
<form action="afficher.jsp" method="post">
    nom : <input type="text" name="nom" />
    prénom : <input type="text" name="prenom" />
    <input type="submit" value="Envoyer" />
</form>
```

afficher.jsp

```
<html> ... <body> ...
  <%= request.getParameter("prenom") %>
  <%= request.getParameter("nom") %>
</body> </html>
```

5. Java Server Pages

- Directive **<jsp:param .../>**
- Transmission de paramètres aux délégués (ou inclus)

```
<jsp:forward page="inc.jsp">
    <jsp:param name="nom" value="Dupont"/>
    <jsp:param name="prenom" value="Paul"/>
</jsp:forward>
```

Les paramètres sont ajoutés à la requête (comme s'ils venaient d'un formulaire) avant transmission à inc.jsp

inc.jsp

```
<html> <body>
    Nom : <%= request.getParameter("nom") %>
    Prénom : <%= request.getParameter("prenom") %>
</body> </html>
```

JSTL et Expression Language

- JSP = HTML + code Java
- Transformation automatique en Servlet
- But
  - Ecrire le moins de code Java possible
  - JSP 2.0 : Expression Language
- Possibilité de définir ses propres balises (tags)
  - JSTL (Java standard tag library)

Expression Language

- Exemple :

```
<% Facture facture = (Facture) request.getAttribute("facture");
....
 <% for (int i=0; i<facture.getNbProduits(); i++)
 { %>
 <tr><td><%=facture.getProduit(i).getDescription()%></td>
      <td><%=facture.getProduit(i).getPrix()%></td>
 </tr>
```

- 2 opérations :
  - Récupérer un java-bean.
  - Accéder à ses attributs.

Expression Language

- Objectifs :
  - Manipulation plus simple des variables
  - S'affranchir de la notation Java
- Accès aux "javabeans" (= objets avec méthodes getPropriété() et éventuellement setPropriété() pour différentes propriétés) :
  - ${nomVariable} (voir transparent suivant)
- Accès aux propriétés :
  - ${nomVariable.propriété} (transformé en un getPropriété() ou setPropriété() suivant le contexte)

Variables

- `${var}` : var est recherché :
  - dans la page
  - dans la requête
    - *request.getAttribute(“var”);*
  - dans la session
    - *request.getSession().getAttribute(“var”);*
  - dans l'application
    - *getServletContext.getAttribute(“var”) ;*

Propriétés « simples »

- propriétés des Beans :
  - ${p.nom}
  - ${p.age}
  - ${p.adresse.rue}
  - ${p.nomComplet}

```
public class Personne {

        private String nom ;
        private String prenom ;
        private int age ;
        private Adresse adresse ;
        private String[] passeTemps ;
        private Vector<String> telephones ;
        private HashMap<String, String> emails ;

        public String getNom() {
                return nom;
        }

        public void setNom(String nom) {
                this.nom = nom;
        }
        ........
        public String getNomComplet()
        {
                return nom + " " + prenom ;
        }
}
```

Propriétés « complexes »

- Tableaux :
  - ${p.passeTemps[0]}
- Vecteurs :
  - ${p.telephones[0]}
- Maps :
  - ${p.emails.personnel} ou
  - ${p.emails["personnel"]}

Variables Prédéfinies

- param
  - *http://localhost/test?nom=dupond*
  - *${param.nom}*
- paramValues
  - *http://localhost/test?a=3&a=4*
- header (headerValues) :
  - *${header["user-agent"]}*
- cookies

JSTL

- 5 sous-parties :
  - core (c)
  - XML (x)
  - i18n (fmt)
  - database (sql)
  - functions (fn)
- On ne s'intéresse ici qu'au « core »
  - pour inclure la bibliothèque :

    <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
  - permet d'accéder aux balises du core en les préfixant par "c:"

JSTL : Core

- Support des variables :
  - *<c:set scope="session" var="test" value="${var}" />*
- Test :

```
<c:if test="${var.x == 3}"> ... </c:if>
<c:choose>
      <c:when test="${empty maliste}">...</c:when>
      <c:otherwise>...</c:otherwise>
</c:choose>
```

JSTL : Core

- Boucles

```
<c:forEach items="${liste}" var="element">
    .... ${element.prop}.....
</c:forEach>
```

- URL :

```
<c:import url=”.....”/>
```

Facture.jsp

```
<c:choose>
   <c:when test="${empty facture.produits}">
       <c:import url="vide.html"/>
   </c:when>
   <c:otherwise>
        <h2>Facture</h2>
        <table>
          <tr><th>Produit</th><th>Prix</th></tr>
          <c:forEach items="${facture.produits}" var="produit">
             <tr>
               <td>${produit.description}</td>
               <td>${produit.prix}</td>
             </tr>
          </c:forEach>
        </table>
        <p> Prix total : ${facture.prixTotal}</p>
   </c:otherwise>
</c:choose>
```