Résumé de l'épisode précédent (Les Servlets)

servlet = ?

- Une servlet est un programme (plug-in) à ajouter à un serveur (quel qu'il soit).
- Ce cours a trait à la programmation Java coté serveur (Java EE)
- Pour l'instant les serveurs acceptant des servlets sont plutôt des serveurs Web.

Comment ça marche ?

- Le serveur (Web) possède désormais un interpréteur Java (JVM)
- => il n'y a pas de processus créé lors de l'exécution de code Java
- Cf. les clients Web possèdent un interpréteur Java permettant de lancer des applets.
- D'où le nom de servlets.

Moteurs de servlets (et de JSP) : Tomcat

- Plug-in de Apache v1.3 ou plus, Microsoft IIS v4.0 ou plus, …
- Est aussi un mini-serveur Web.

Une servlet : code complet

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

public class MaPremiereServlet extends HttpServlet {

     public void doGet(HttpServletRequest request, HttpServletResponse response)
       throws ServletException, IOException {

          // Etape 1.  Spécifier le type MIME du contenu de la réponse
          response.setContentType("text/html");

          // Etape 2.  Récupère le PrintWriter pour envoyer des données au client
          PrintWriter out = response.getWriter();

          // Step 3.  Envoyer l'information au client
          out.println("<html>");
          out.println("<head><title>Bonjour Servlet</title></head>");
          out.println("<body>");
          out.println("<h1> Bonjour à tous </h1>");
          out.println("Il est : " + new java.util.Date());
          out.println("</body></html>");
     }
     public void doPost(HttpServletRequest request, HttpServletResponse response)
       throws ServletException, IOException {
     doGet(request, response);
     }
}
```

Démonstration

- On lance le serveur Web
- La servlet est rangée sous `REP_INSTAL_TOMCAT\webapps\examples\WEB-INF\classes`
- correspond à l'URL : `http://localhost:8080/examples/servlet/MaPremiereServlet`

Euh + bidouilles dans le fichier `REP_INSTAL_TOMCAT\webapps\examples\WEB-INF\web.xml`

application web = ?

- "Une application web est une extension dynamique d'un serveur web ou applicatif.
- Par exemple :
  - les application web orientées présentation qui génèrent des pages web (HTML, XML) dynamiquement

Architecture d'une application web 1/3

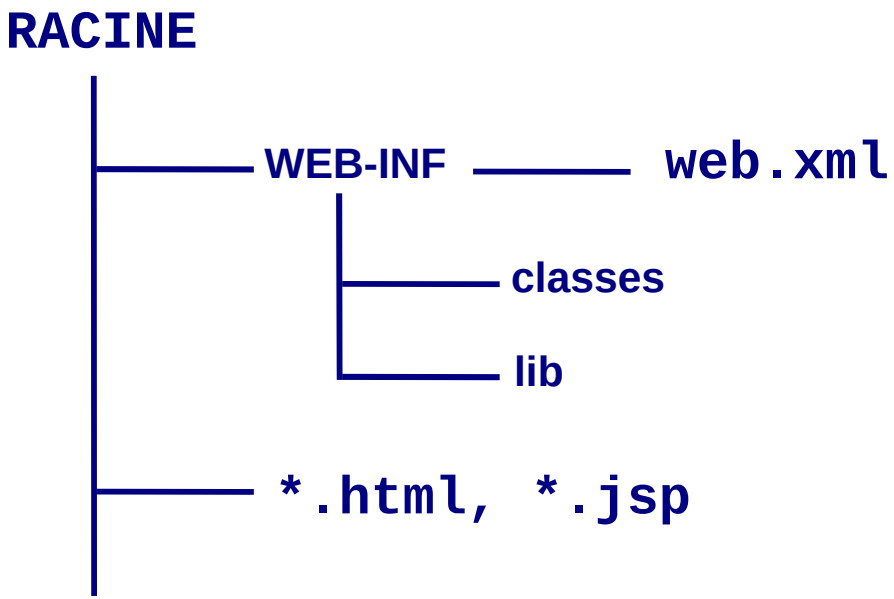
- Une bonne référence est : `http://tomcat.apache.org/tomcat-6.0-doc/appdev/deployment.html`
- Depuis la version 2.2 des spécifications des servlets, les ressources doivent être rangées comme indiqué diapo suivante :

Architecture d'une application web 2/3

- Les fichiers `.html` et `.jsp` doivent être rangés à partir de la racine de votre site web (= application web).
- `/WEB-INF/web.xml` : le fichier descripteur de déploiement de votre application web
- `/WEB-INF/classes/` : le répertoire racine de rangement des `.class` (servlets compilés, etc.). Si les classes sont dans des packages, la hiérarchie des packages doit être respectée à partir de `/WEB-INF/classes/`. Par exemple la classe `cnam.ihm.MaServlet` doit être mise dans `/WEB-INF/classes/cnam/ihm/MaServlet.class`

Architecture d'une application web 3/3

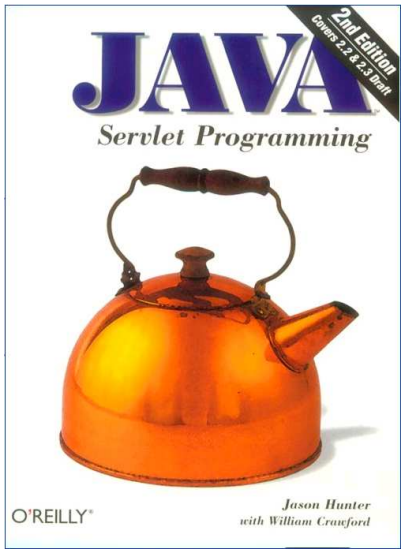
- `/WEB-INF/lib/` : le répertoire contenant les `.jar` nécessaires à votre application web (driver JDBC, etc.)



- On peut mettre tout cela dans un fichier compressé : un `.war`

Bibliographie

- Page de départ de la technologie servlets : `http://java.sun.com/products/servlet/index.html`
- Java servlets, Jason Hunter, ed O'Reilly traduit en français



Fin du résumé de l'épisode précédent

JavaServer Pages (JSP)

ssi : la technique server side include

- Une page ssi (`shtml`) est demandée par un client web
- Le serveur Web passe la main au programme adéquat qui traite la partie de la page le concernant.
- Ce programme génère la partie dynamique
- La page HTML créée dans son ensemble est retournée au serveur puis au client Web.

JavaServer Pages

- = JSP = la technique des ssi en Java
- = une page HTML contenant du code Java
- => meilleure division des tâches :
  - présentation générale par les graphistes
  - coté dynamique par des programmeurs (Java)

Comment ça marche ?

- Concrêtement :
  - toute la page HTML est convertie en une servlet
  - cette servlet est traitée par le moteur Java intégré au serveur Web (technologie des servlets) et retourne la page HTML construite

JSP vs. Servlets

- Servlet = du code Java contenant de l'HTML
- JSP = une page HTML contenant du code Java
- Concrètement avec les JSP :
  - les parties statiques de la page HTML sont écrites en HTML
  - les parties dynamiques de la page HTML sont écrites en Java

Notre première JSP

- fichier `MaDate.jsp`

```
<html><head><title>Obtenu par une JSP</title></head>
<body>

<h3>Bonjour de ma part </h3> <hr>
La date courante est :  <%= new java.util.Date() %>
</body>
</html>
```

- Traité quand le client demande l'URL de la JSP : `http://serveurWeb:<port>/.../MaDate.jsp`

Tomcat et JSP

- Des exemples de JSP (code + liens pour l'exécution) sont disponibles à partir de `REP_INSTALL_TOMCAT/webapps/examples/jsp` pour Tomcat 6.0

Exécution de JSP

- Il faut mettre les pages JSP dans un endroit particulier du serveur Web
- Cet endroit dépend du serveur Web et de sa configuration
- Pour tomcat en configuration standard,
  `http://localhost:8080/examples/jsp/MaDate.jsp`
  ~
  `REP_INSTAL_TOMCAT\webapps\examples\jsp\MaDate.jsp`
  pour tomcat 6.0
- Et sans bidouille !!

Exécution de JSP (suite)

- Une démo:
- Le résultat de `MaDate.jsp` est :

Bonjour de ma part

La date courante est : Sun Dec 23 18:04:36 CET 2001

- Une autre exécution donne une autre date => dynamicité

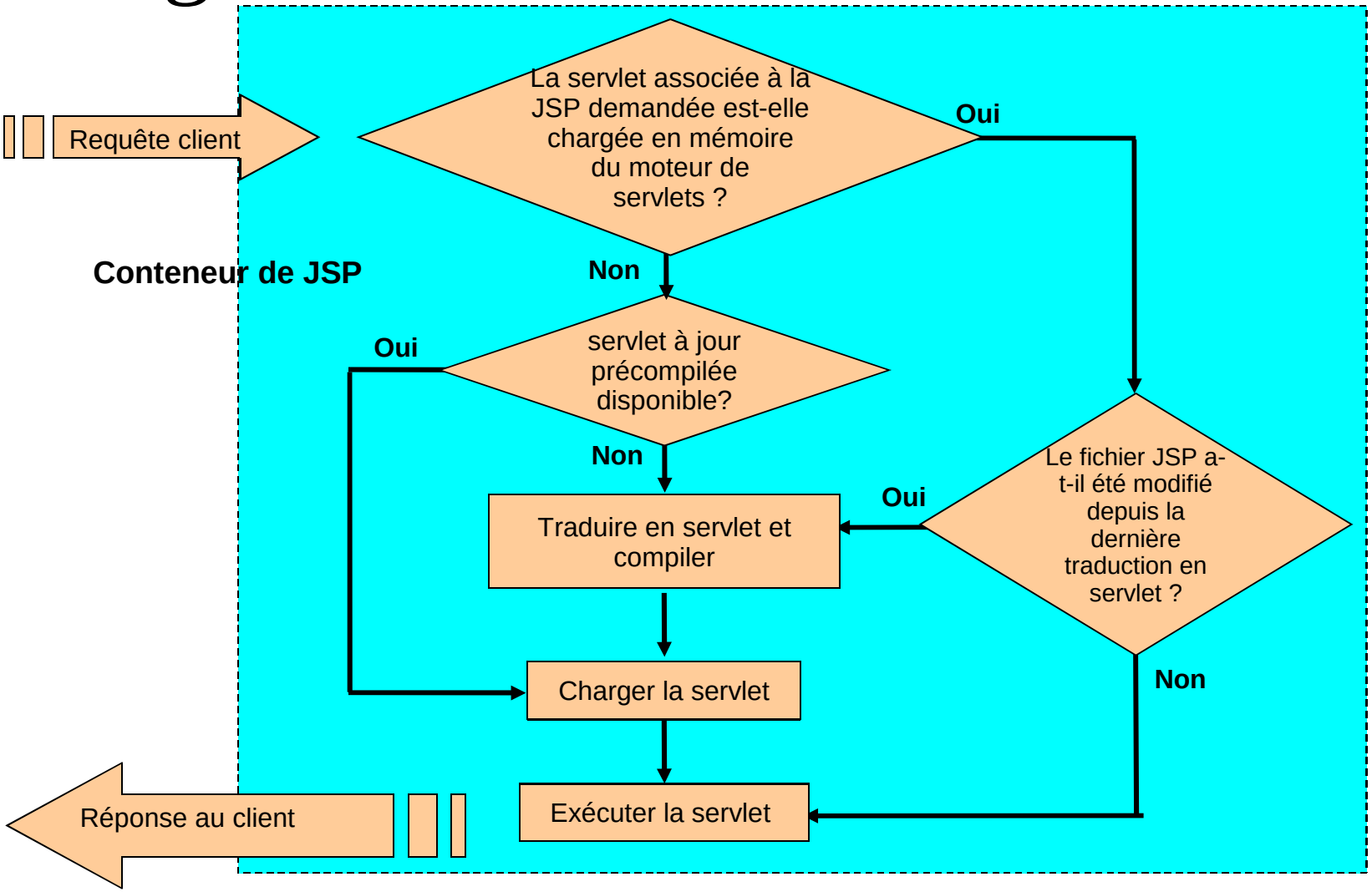
Que s'est il passé ?

- Le moteur de JSP a construit une servlet (`MaDate_jsp.java` sous l'arborescence `work` pour Tomcat 6.0)
- Cette phase est parfois appelée la traduction de la JSP (en servlet)
- Puis a compilé et exécuté la servlet

La servlet construite

```
package org.apache.jsp;
...
public class MaData_jsp extends HttpJspBase {
     ...
    public void _jspService(HttpServletRequest request, HttpServletResponse
response) throws IOException, ServletException {
       ...
            pageContext = _jspxFactory.getPageContext(...);
            session = pageContext.getSession();
            out = pageContext.getOut();
            // HTML
   // begin [file="C:\\...\\examples\\jsp\\MaDate.jsp";from=(0,0);to=(4,24)]
         out.write("<html><head><title>Obtenu par une JSP</title></head>\r\n
<body>\r\n\r\n<h3>Bonjour de ma part</h3> <hr>\r\n
     La date courante est :  ");
     // end
//begin [file="C:\\...\\examples\\jsp\\MaDate.jsp";from=(4,27)to=(4,49)]
                out.print( new java.util.Date() );
            // end
            // HTML
// begin [file="C:\\...\\examples\\jsp\\date.jsp";from=(4,51);to=(6,7)]
                out.write("\r\n</body>\r\n</html>"); // end
            ...
    }
}
```

Algorithme d'exécution de la JSP



3 parties d'une JSP

- scriptlets <% %>
- déclarations <%! %>
- expressions <%= %>

Scriptlets <% %>

- contient du code Java
- insérer dans `_jspService()` de la servlet, donc peut utiliser `out`, `request`, `response`, etc.
- Exemple :

```
<%
    String[] langages = {"Java", "C++", "Smalltalk", "Simula 67"};
    out.println("<h3>Principaux langages orientés objets : </h3>");
    for (int i=0; i < langages.length; i++) {
              out.println("<p>" + langages[i] + "</p>");
    }
%>
```

Déclarations <%! %>

- Sont des déclarations Java.
- Seront insérées comme des membres de la servlet
- Permet de définir des méthodes ou des données membres
- Exemples :

```
<%!
      int random4() {
            return (int)(Math.random() * 4);
      }
%>
```

```
<%!
      int nombreFetiche = 2;
%>
```

Expressions <%= %>

- En fait expression Java qui renvoie un objet `String` ou un type primitif.
- Un raccourci pour `<% out.println(…); %>`
- `<%= XXX %> ~ <% out.println(XXX); %>`
- attention au `;`
- est donc converti en `out.println(…)` dans la méthode `_jspService(…)` de la servlet.

```
La somme est:  <%= (195 + 9 + 273) %>

Je vous réponds à l'adresse :  <%= request.getParameter("email_address") %>
```

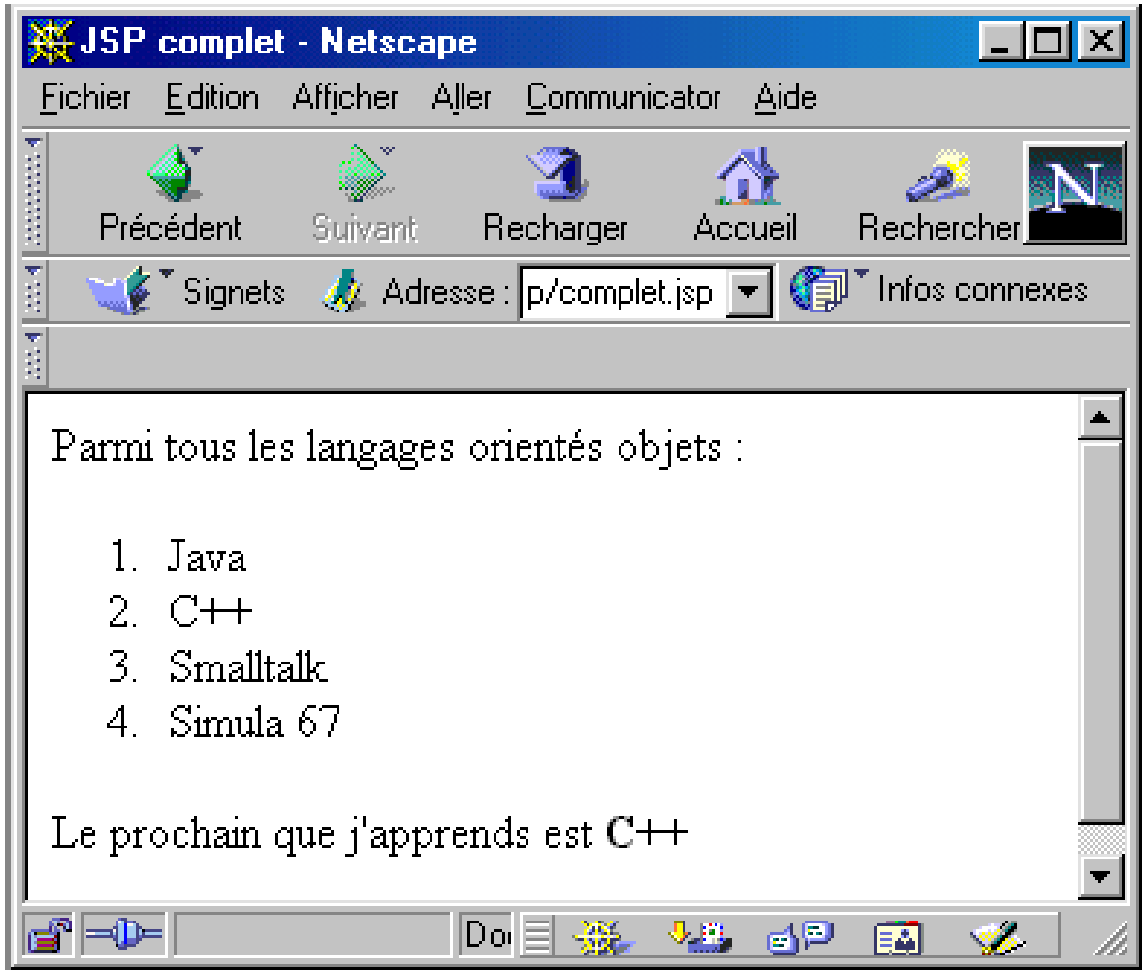
Objets prédéfinis dans une JSP

- 3 objets peuvent être immédiatement utilisés dans une expression ou une scriptlet d'une JSP :
  - `out` : le canal de sortie
  - `request` (`HttpServletRequest`) : l'objet requête
  - `response` (`HttpServletResponse`) : l'objet réponse
- Il y en a d'autres
- Cf. ces mêmes objets dans une servlet

Un exemple complet : `complet.jsp`

```
<html><head><title>JSP complet</title></head>
<body>
<%! String[] langages = {"Java", "C++", "Smalltalk", "Simula 67"};
    int random4() {
            return (int) (Math.random() * 4);
    }
%>
<p>Parmi tous les langages orientés objets :</p>
<ol>
<%
    for (int i=0; i < langages.length; i++) {
             out.println("<li>" + langages[i] + "</li>");
    }
%>
</ol>
<p>Le prochain que j'apprends est <b><%= langages[random4()] %> </b></p>
</body>
</html>
```

complet.jsp


JSP complet - Netscape
Fichier Edition Afficher Aller Communicator Aide
Précédent Suivant Recharger Accueil Rechercher
Signets Adresse : p/complet.jsp Infos connexes
Parmi tous les langages orientés objets :
1. Java
2. C++
3. Smalltalk
4. Simula 67
Le prochain que j'apprends est C++

Déboguer les JSP

- La fenêtre de lancement du serveur Web donne des indications. Suivant les serveurs, une page HTML est retournée avec des indications.
- Ces éléments sont très souvent relatifs à la servlet et pas à la page JSP.
- Directives `<%@ page errorPage= ...%>` et
  `<%@ page isErrorPage="true"%>`

Déboguer les JSP (suite)

- Un page JSP peut référencer une page erreur par `<%@ page errorPage="page.jsp"%>`
- La page erreur est indiquée par l'entête `<%@ page isErrorPage="true"%>`
- Si une exception est levée le traitement est dérouté vers la page erreur qui connaît la référence `exception` qui repère l'exception

Déboguer les JSP : exemple

langages.jsp

```
<%@ page errorPage="erreur.jsp"%>
<%! String[] langages = {"Java", "C++", "Smalltalk", "Simula 67"};
%>
<p>Parmi tous les langages orientés objets :</p>
<ol>
<%
    // levée d'une ArrayIndexOutOfBoundsException
    for (int i=0; i < 7; i++) {
              out.println("<li>" + langages[i] + "</li>");
    }
%>
```

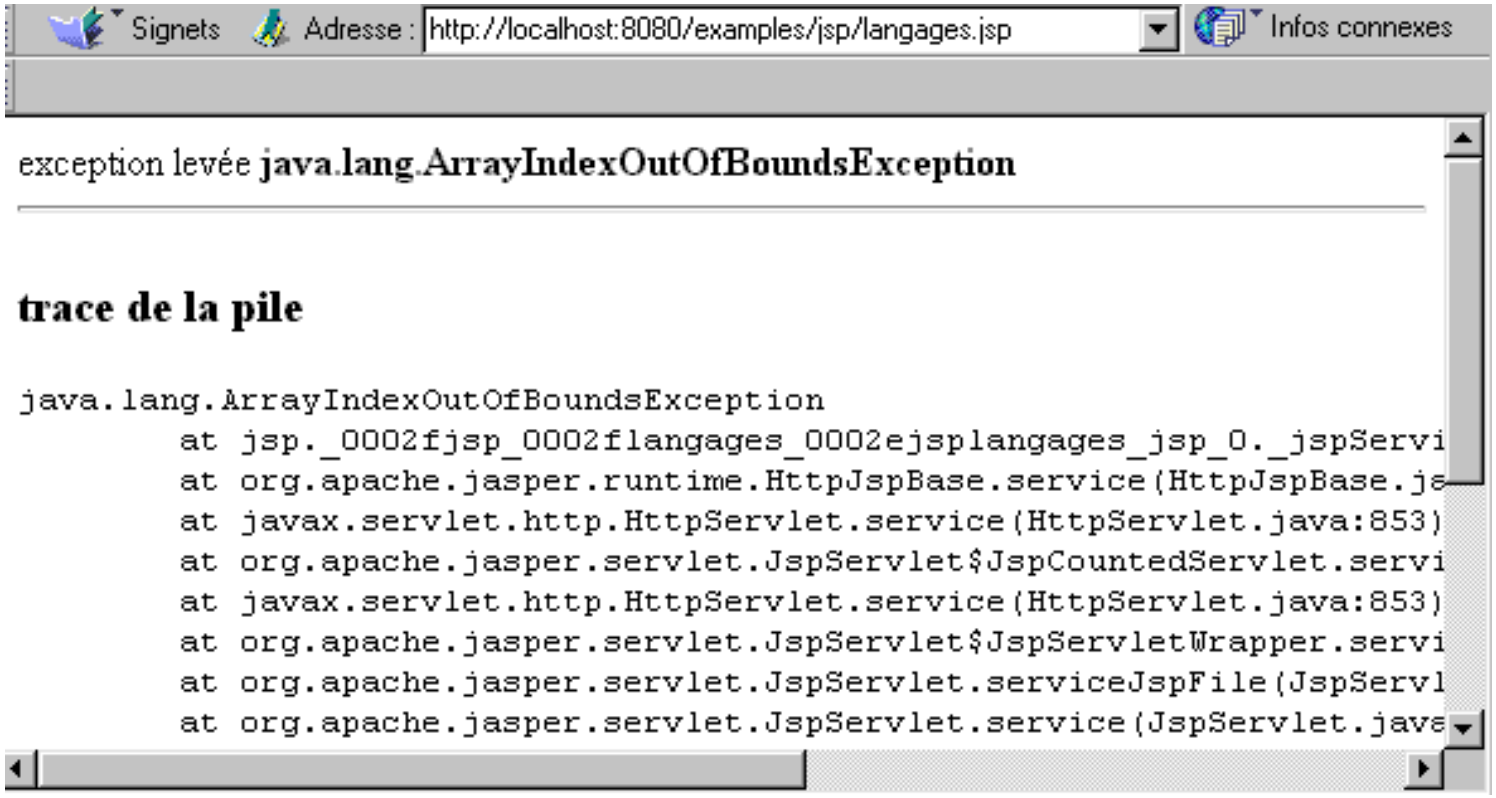
Déboguer les JSP : exemple (suite)

erreur.jsp

```
<%@ page isErrorPage="true"%>
<html><body>
exception levée <b> <%= exception %> </b>
<hr>
<h3>trace de la pile</h3>
<pre>
<%
     java.io.PrintWriter myWriter = new java.io.PrintWriter(out);
     exception.printStackTrace(myWriter);
%>
</pre>
</body></html>
```

Déboguer les JSP : exemple (fin)

- Charger la page `langages.jsp` amène à :



Enchaîner les pages

- Un page JSP peut en appeler une autre par la directive : `<jsp:forward>`
- Syntaxe :

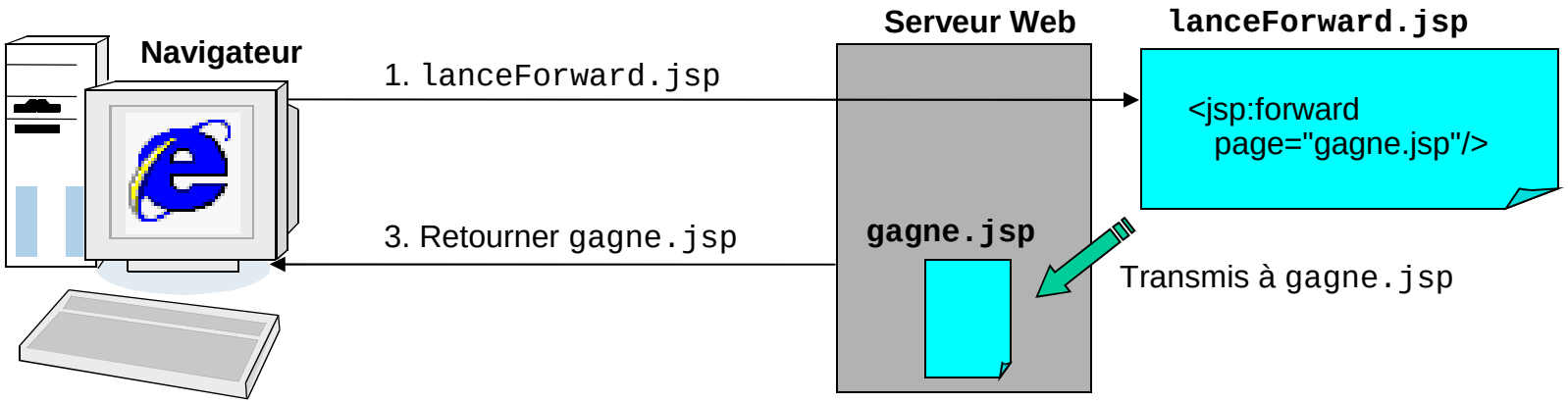
```
<jsp:forward page="pageDeRedirection" />
```

`lanceForward.jsp`

```
<% String repUtilisateur = request.getParameter("repTextField");
   int rep = Integer.parseInt(repUtilisateur);
    if ((rep % 2) == 0) {
%>
    <jsp:forward page="gagne.jsp"/>
<% } else { %>
    <jsp:forward page="perdu.jsp"/>
<% } %>
On n'affiche jamais cela
```

Enchaîner les pages (suite)

- Après un `<jsp:forward>`, le traitement est entièrement pris en charge par nouvelle page

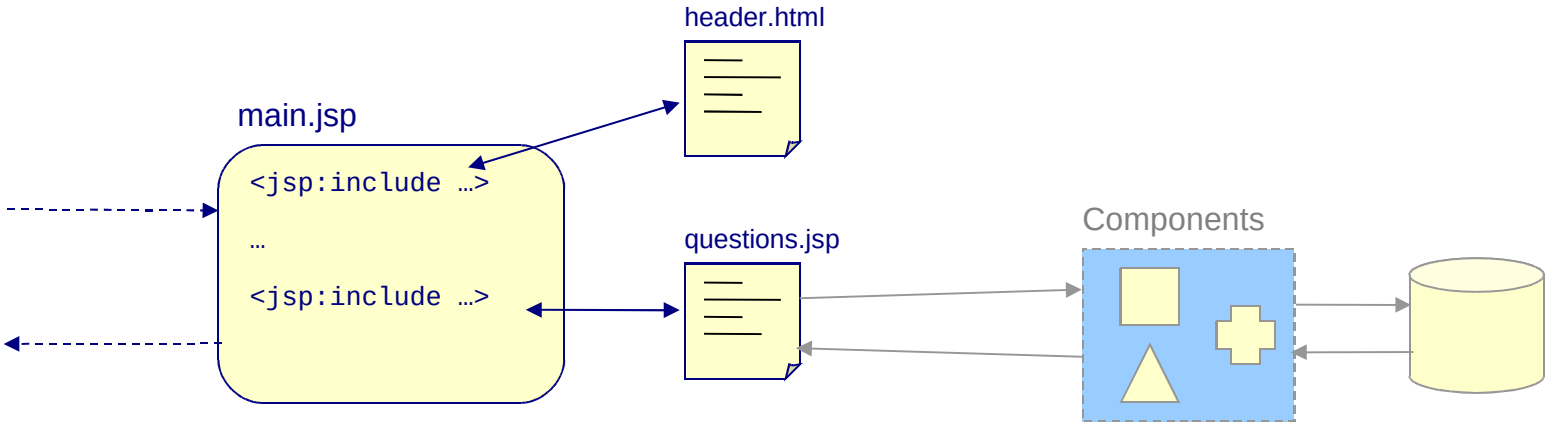


Inclusion de fichiers par <%@ include file="…" %>

- Utilisé pour faire des inclusions statiques de fichiers : en-tête ou pied de page ...
- Syntaxe :
  `<%@ include file="path to relative URL" %>`
- Le fichier est inclus au moment de la traduction

Inclusion de pages `<jsp:include page="…" />`

- Pour inclure des sorties de servlets ou JSPs
  - Le contenu du servlet ou de la JSP est traité avant l'inclusion
  - La page est incluse au moment de la requête
- Syntaxe :
  `<jsp:include page="path to relative URL" />`
- Le traitement de la page courante continue après l'inclusion des ressources



- Permet aussi d'inclure du HTML
  - On préférera `<%@ include %>` pour inclure du HTML

JSP et Java beans

- But : avoir le moins de code Java possible dans une page JSP (HTML)
- Sous-traiter le code à un Java bean
- balise XML : `<jsp:useBean>`

JSP et Java beans (suite)

- Syntaxe générale :

```
<jsp:useBean id="nomInstanceJavaBean"
   class="nomClasseDuBean"
scope="request|session|application|
page">
</jsp:useBean>
```

- Le bean est alors utilisable par *nomInstanceJavaBean*
- balise sans corps donc utilisation de `<jsp:useBean … />`

l'attribut `scope`

- Il indique la portée du bean.

valeur	Description
`request`	Le bean est valide pour cette requête. Il est utilisable dans les pages de redirection de la requête (`<jsp:forward>`). Il est détruit à la fin de la requête.
`page`	Similaire à `request`, mais le bean n'est pas transmis aux pages de redirection `<jsp:forward>`. C'est la portée par défaut
`session`	Le bean est valide pour la session courante. S'il n'existe pas encore dans la session courante, il est créé et placé dans la session du client. Il est réutilisé jusqu'à ce que la session soit invalidée
`application`	Le bean est valide pour l'application courante. Il est créé une fois et partagé par tous les clients de l'application web.

JSP et Java beans : exemple

- Soit le bean :

```
package cnam;

public class SimpleBean implements java.io.Serializable {

    private int compter;

    public SimpleBean() {
      compter = 0;
    }

    public void setCompter(int theValue) {
      compter = theValue;
    }

    public int getCompter() {
         return compter;
    }

    public void increment() {
      compter++;
    }
}
```

- Remarque : il faut mettre le bean dans un package

Utilisation du bean dans une JSP

- Utilisation à l'aide de son nom
- Récupération des propriétés :
  - Par appel de méthode `getXXX()` :
  - Par la balise `<jsp:getProperty …>`

```
<p> on repere le bean par le nom nomBean<br>
<jsp:useBean id="nomBean" class="cnam.SimpleBean"
    scope="session" />
<p> On accede a une propriéte avec une expression:
<br> compteur = <%= nomBean.getCompter() %>
<hr>
On incrémente le compteur <% nomBean.increment(); %>
<p>On peut accéder à la propriété par une balise :<br>
<jsp:getProperty name="nomBean" property="compter" />
```

Positionner les propriétés du bean dans une JSP

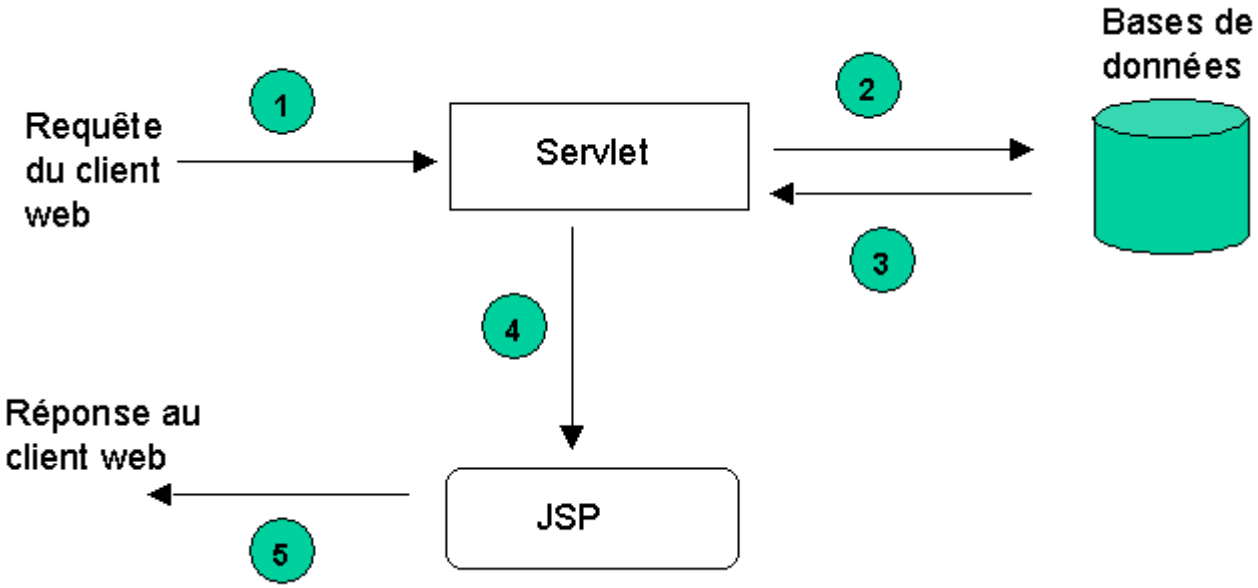
- Par appel de méthode `setXXX(...)` :
- Par la balise `<jsp:setProperty …>`

```
<p> on repere le bean par le nom nomBean<br>
<jsp:useBean id="nomBean" class="cnam.SimpleBean"
    scope="session" />

<p> On positionne une propriéte avec une expresion:
<br> compteur = <%= nomBean.setCompter(6) %>

<p>ou par une balise :<br>
<jsp:setProperty name="nomBean" property="compter" value="6" />
```

Architecture MVC

- modèle = les données accédées par un code Java (JDBC, RMI, EJB, etc.)
- vues = JSP
- contrôleur = servlets

Architecture MVC (suite)



- Syntaxe dans la servlet pour lancer la JSP :

```
public void doPost(HttpServletRequest request, HttpServletResponse response){
   ServletContext context = getServletContext();  // héritée de GenericServlet
   RequestDispatcher dispatcher =
     context.getRequestDispatcher("/maPageMiseEnForme.jsp");
   dispatcher.forward(request, response);
}
```

Architecture MVC (suite)

- La servlet peut passer des valeurs à la JSP appelée grâce à `setAttribute()`

```
public void doPost(HttpServletRequest request, HttpServletResponse response) {
     // appelle les méthodes sur les objets métiers
     ArrayList theList = // un objet à passer
     // ajoute à la requête
     request.setAttribute("nomDelObjet", theList);
    ServletContext context = getServletContext();
    RequestDispatcher dispatcher = context.getRequestDispatcher("/jspAAppeler.jsp");
    dispatcher.forward(request, response);
}
```

- La JSP extrait les objets de `request` grâce à `getAttribute()`

```
<%  ArrayList theList = (ArrayList)
       request.getAttribute("nomDelObjet");
// maintenant, utiliser l'ArrayList
%>
```