

De l'esprit à la machine
L'approche Professo-Académique

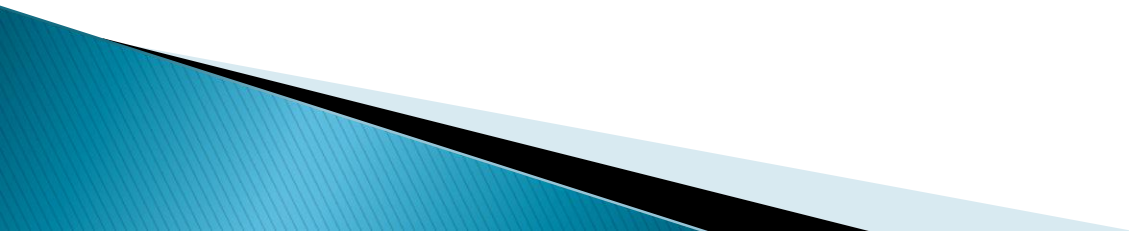
J2EE

Abdelahad SATOUR

Séances 13–16

JSP

Introduction à JSP



Mais qu'est ce que c'est ?

- ▶ Standard pour construire des applis Web
 - Portable
 - En mélangeant le HTML et des directives
 - Basé sur Java
- ▶ Equivalent PHP et ASP
 - Plus "propre" que PHP
 - Plus indépendant que ASP
- ▶ Plusieurs spécifications
 - JSP 2.0 = la dernière
- ▶ De nombreuses librairies : les TagLibs

JSP et Servlets ?

- ▶ Les deux sont basés sur Java
- ▶ Les Servlets sont peu adaptés à la génération de contenu
- ▶ Les JSP sont peu adaptés à l'extension de fonctions du serveur
- ▶ Note : Le JSP engine qui interprète les pages JSP est un Servlet
- ▶ Les JSP sont un moyen de construire des Servlet de façon déclarative
 - Une page JSP est transformée en un programme java (servlet) puis compilée et exécutée

JSP?

Java Server Pages (JSP)

Programme Java s'exécutant **côté serveur Web**

servlet prog. "autonome" stockés dans un fichier **.class** sur le serveur
JSP prog. **source** Java embarqué dans une page **.html**

	côté client	côté serveur
.class autonome	applet	servlet
embarqué dans .html	JavaScript	JSP

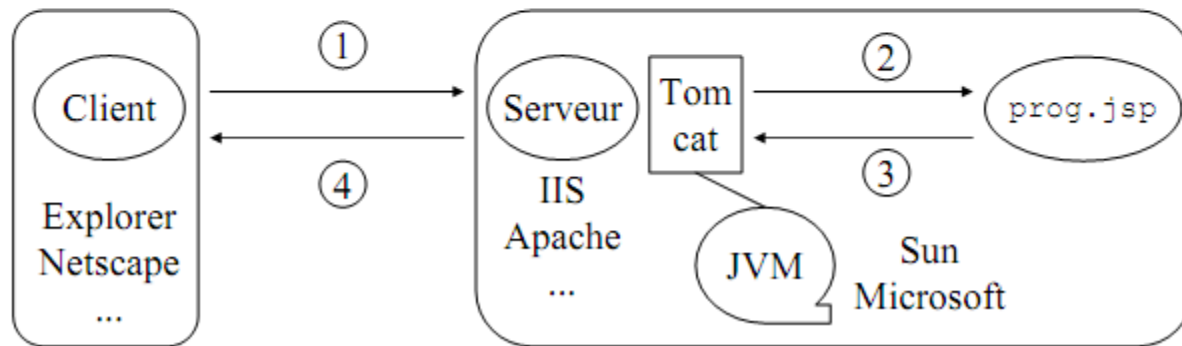
Servlet et JSP

- exécutable avec tous les serveurs Web (Apache, IIS, ...)
- auxquels on a ajouté un "moteur" de servlet/JSP (le plus connu : **Tomcat**)
- JSP compilées automatiquement en servlet par le moteur

JSP Cycle d'appel

Java Server Pages (JSP)

- du code Java **embarqué** dans une page HTML entre les balises `<%` et `%>`
- extension `.jsp` pour les pages JSP
- les fichiers `.jsp` sont stockés sur le serveur (comme des docs)
- ils sont désignés par une **URL** `http://www.lip6.fr/prog.jsp`
- le **chargement** de l'URL provoque l'**exécution** de la JSP côté serveur

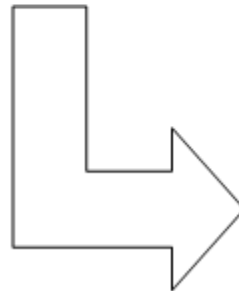


Utilisation

Illustration du fonctionnement

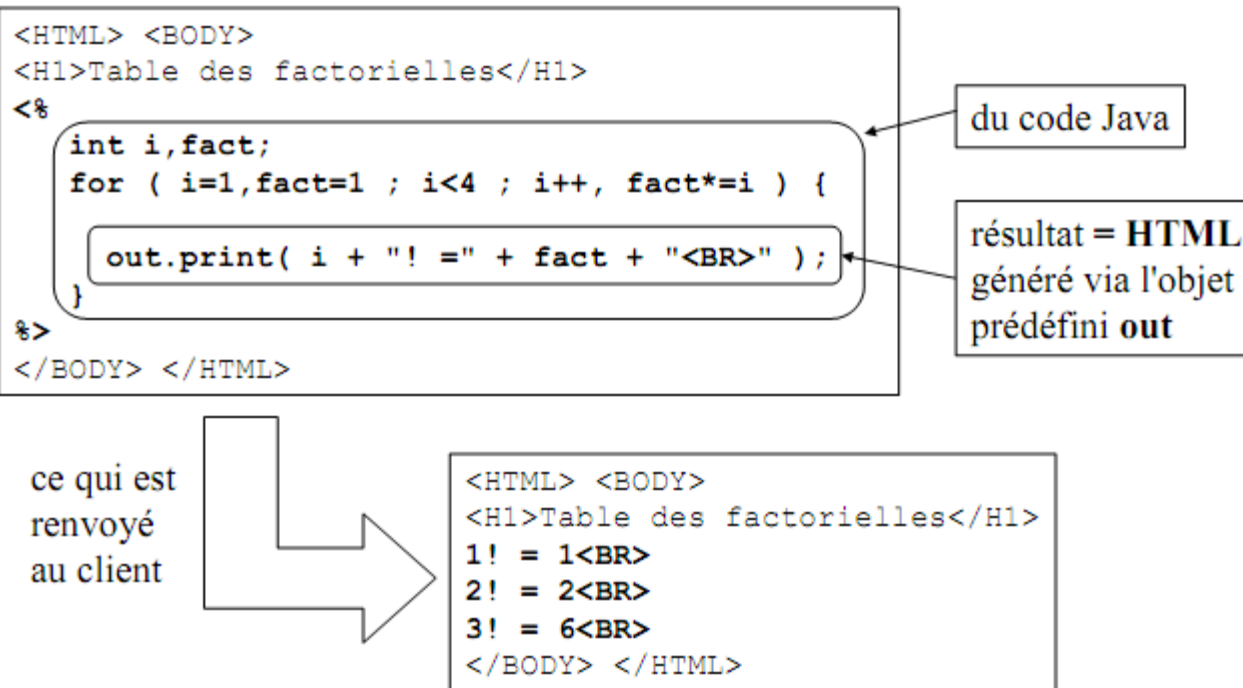
```
<HTML> <BODY>
<H1>Table des factorielles</H1>
<%
    int i,fact;
    for ( i=1,fact=1 ; i<4 ; i++, fact*=i ) {
        out.print( i + "! =" + fact + "<BR>" );
    }
%>
</BODY> </HTML>
```

invocation
⇒
exécution
côté serveur

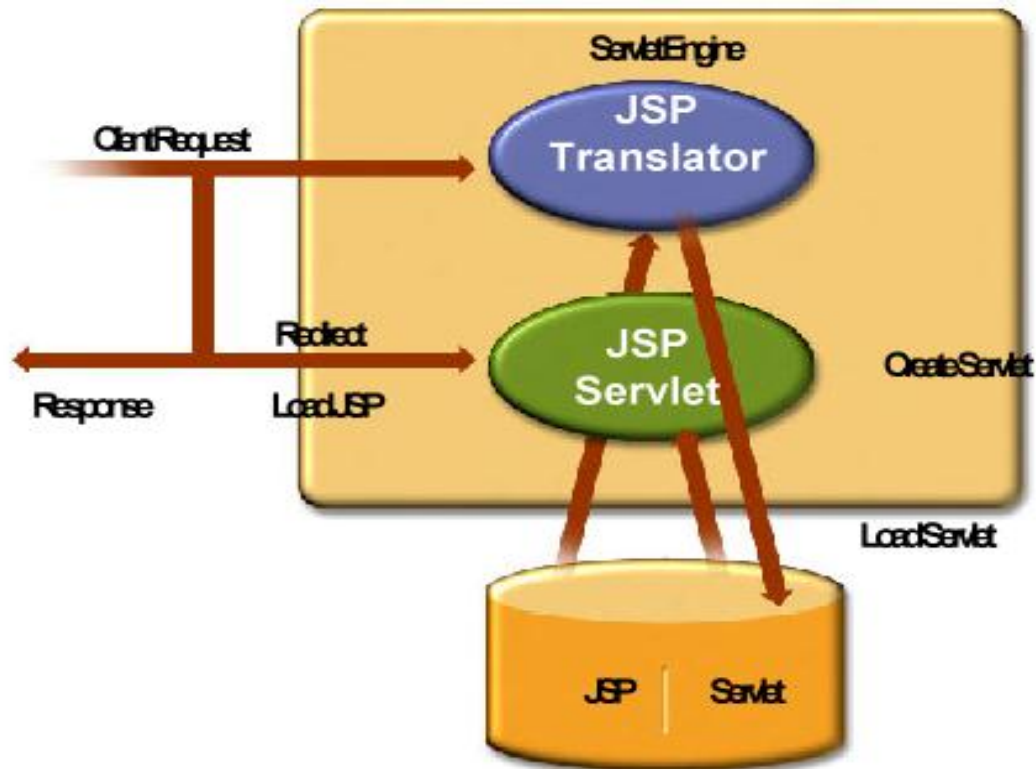


JSP?

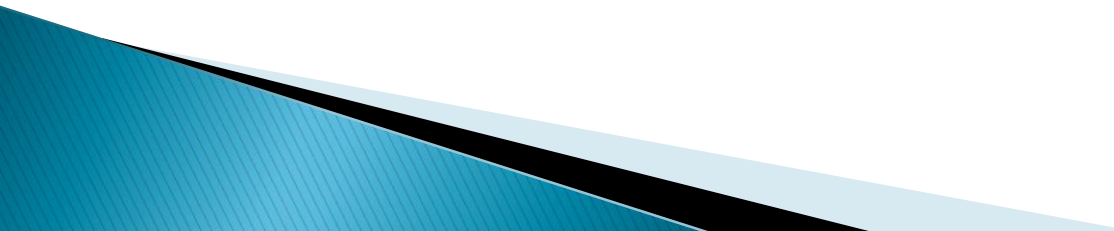
Principe de fonctionnement



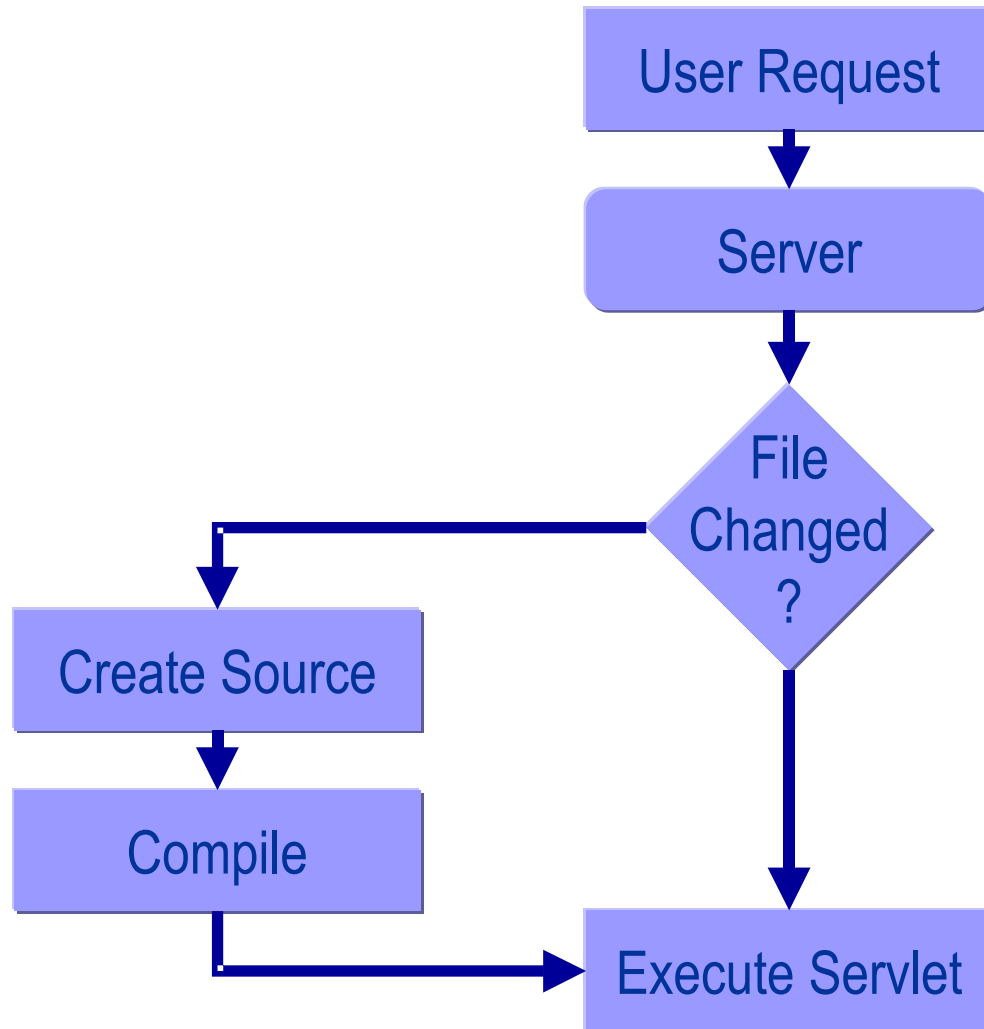
Architecture



Le cycle de vie d'un JSP

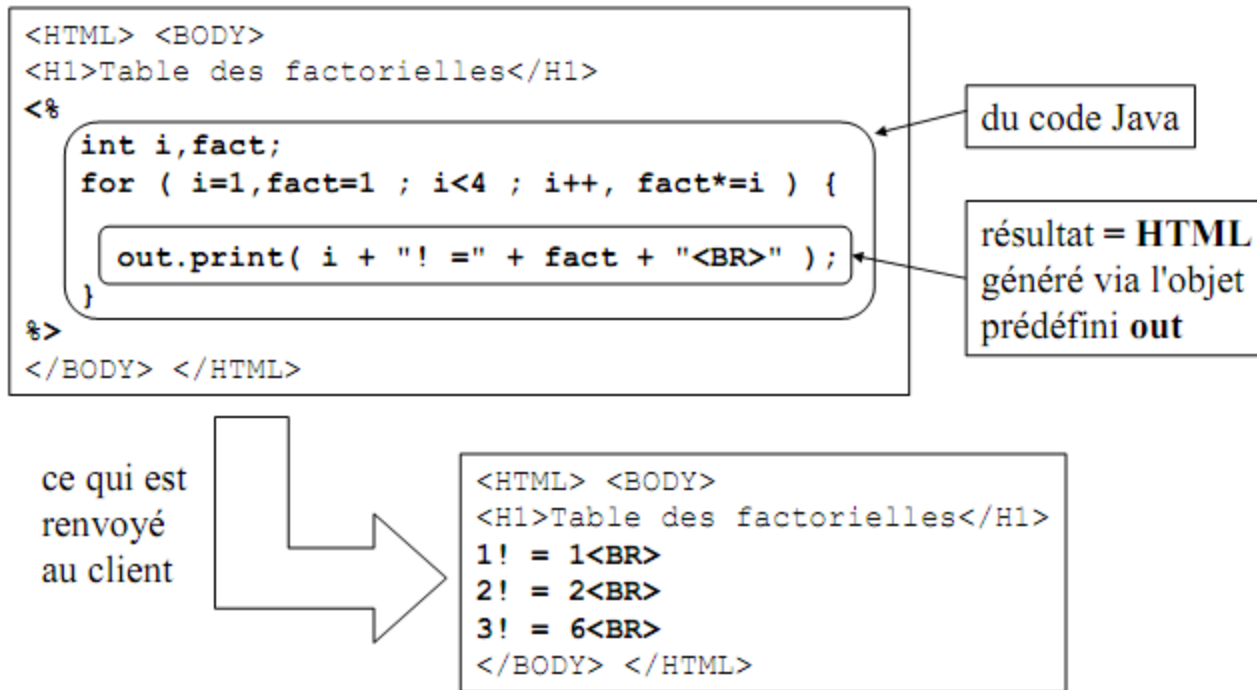
- ▶ JspInit()
 - ▶ JspService
 - Accept request
 - Generate response
 - ▶ JspDestroy()
- 

Cycle de vie



La syntaxe de base scriptlets

Principe de fonctionnement



La syntaxe de base scriptlets

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
 - la **compile**
 - l'**instancie**

⇒ JSP = objet Java présent dans le moteur
- 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

La syntaxe de base expression

Directive `<%= ... %>`

La directive `<%= expr %>` génère l'affichage d'une valeur de l'expression *expr*

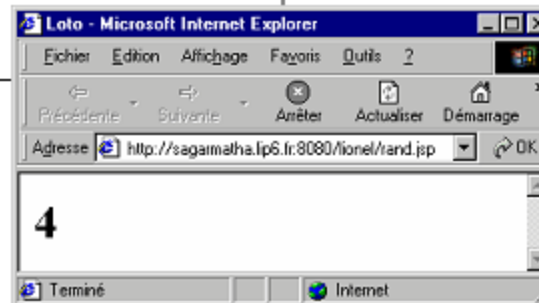
⇒ `<%= expr %>` raccourci pour `<% out.print(expr); %>`

```
<HTML> <BODY>

<% int aleat = (int) (Math.random() * 5); %>

<H1> <%= aleat %> </H1>

</BODY> </HTML>
```



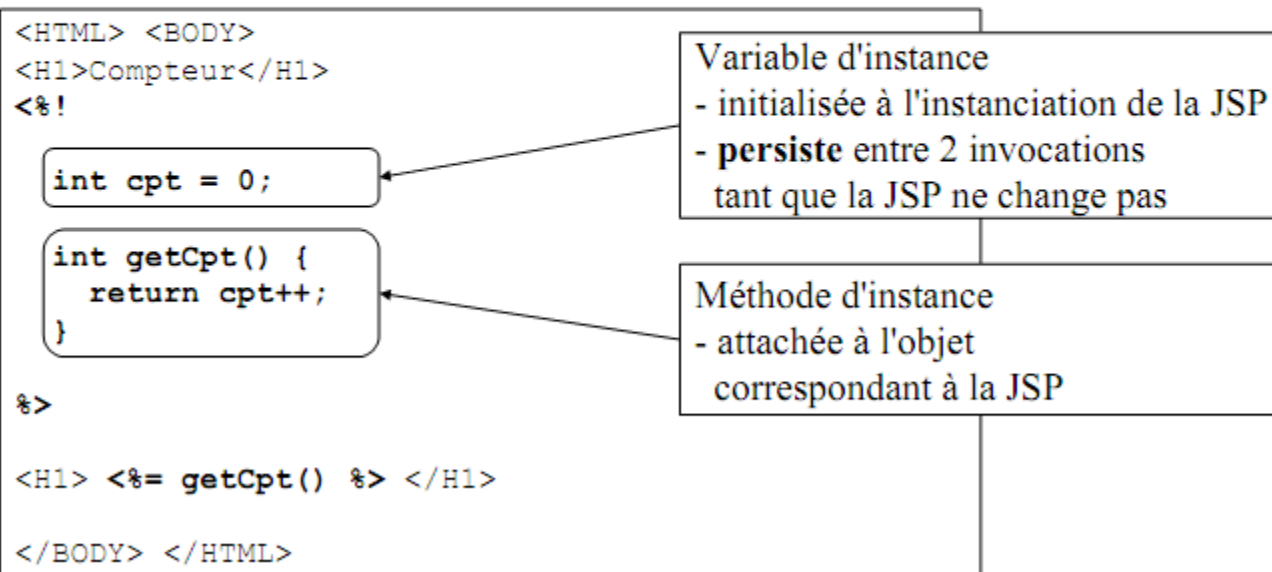
La syntaxe de base expression

```
<body>  
  <h2>Hello World!</h2>  
  Aujourd'hui on est le  
  <%=new java.util.Date(System.currentTimeMillis()).toString()%>  
</body>
```


La syntaxe de base déclaration

Méthodes et variables d'instance

Des **méthodes** et des **variables** d'instance peuvent être associées à une JSP entre les directives `<%!` et `%>`



La syntaxe de base déclaration

Variables d'instance

Attention !!

```
<%! int cpt = 0; %>
```

≠

```
<% int cpt = 0; %>
```

- variable **d'instance** de la JSP (**persiste**)
- variable **locale** à la JSP (**réinitialisée** à chaque invocation de la JSP)

La syntaxe de base directive

- ▶ Les directives permettent de préciser des informations globales sur la page JSP. Les spécifications des JSP définissent trois directives :
 - page : permet de définir des options de configuration
 - include : permet d'inclure des fichiers statiques dans la JSP avant la génération de la servlet
 - taglib : permet de définir des tags personnalisés
- ▶ Leur syntaxe est la suivante :
- ▶ `<%@ directive attribut="valeur" ... %>`

La syntaxe de base directive :: page

La directive `<%@ page ... %>`

Donne des informations sur la JSP (non obligatoire, valeurs par défaut)

`<%@ page import="..." %>` (ex. `<%@ page import="java.io.*" %>`)
les "import" nécessaires au code Java de la JSP

`<%@ page errorPage="..." %>` (ex. `<%@ page errorPage="err.jsp" %>`)
fournit l'URL de la JSP à charger en cas d'erreur

`<%@ page contentType="..." %>` (ex. `<%@ page contentType="text/html" %>`)
le type MIME du contenu retourné par la JSP

`<%@ page isThreadSafe="..." %>` true ou false
true la JSP peut être exécutée par plusieurs clients à la fois (valeur par défaut)

`<%@ page isErrorPage="..." %>` true ou false
true la JSP est une page invoquée en cas d'erreur

La syntaxe de base directive :: page

Option	Valeur	Valeur par défaut	Autre valeur possible
autoFlush	Une chaîne	«true»	«false»
buffer	Une chaîne	«8kb»	«none» ou «nnnkb» (nnn indiquant la valeur)
contentType	Une chaîne contenant le type mime		
errorPage	Une chaîne contenant une URL		
extends	Une classe		
import	Une classe ou un package.*		
info	Une chaîne		
isErrorPage	Une chaîne	«false»	«true»
isThreadSafe	Une chaîne	«true»	«false»
langage	Une chaîne	«java»	
session	Une chaîne	«true»	«false»

La syntaxe de base directive :: page

Exemple :

```
<%@ page import="java.util.*" %> <%@ page import="java.util.Vector" %> <%@ page  
info="Ma premiere JSP"%>
```

La syntaxe de base directive :: page

- ▶ options directive page.pdf

La syntaxe de base directive :: include

- ▶ La directive include
- ▶ (html, java, jsp)

```
<HTML> <HEAD> <TITLE>Essai de page JSP</TITLE>  
</HEAD> <BODY> <p align="center">Test d'inclusion d'un  
fichier dans la JSP</p> <%@ include file="bonjour.htm"%> <p  
align="center">fin</p> </BODY> </HTML>e
```


La syntaxe de base

directive :: taglib

- ▶ **La directive taglib**

Cette directive permet de déclarer l'utilisation d'une bibliothèque de tags personnalisés. L'utilisation de cette directive est détaillée dans la section consacrée aux bibliothèques de tags personnalisés.

La syntaxe de base

- ▶ Les directives
 - Instruction pour le moteur JSP
 - Encadrées par `<%@ %>`
- ▶ Les déclarations
 - Déclarations de variables ou de méthodes utilisables dans la page
 - Encadrées par `<%! %>`
- ▶ Les expressions
 - Une expression est évaluée, transformée en chaîne et incluse dans la page
 - Encadrées par `<%= %>`
- ▶ Les scriptlets
 - Morceau de code java exécuté dans la page
 - Encadrés par `<% %>`

les commentaires

- ▶ Les commentaires sont les mêmes qu'en XML
 - `<%-- tagada --%>`

Les objets implicites

- ▶ Objets utilisables sans déclaration dans les expressions et les scriptlets
 - request (request scope) : HttpServletRequest
 - response (page scope) : HttpServletResponse
 - pageContext (page scope) : PageContext
 - L'objet représentant le contexte de la page
 - session (session scope) : HttpSession
 - L'objet représentant le contexte de la session
 - page (=this) (page scope) : HttpJSPPage
 - out (page scope) : JspWriter
 - application (application scope) ServletContext
 - config (page scope) : ServletConfig
 - exception (page scope) : Throwable
 - Définit dans les pages de traitement des erreurs

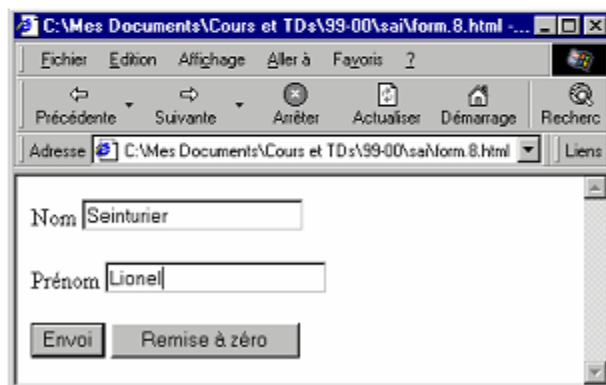
Exemple

Récupération des données d'un formulaire

Méthode `String getParameter(String)` de l'objet prédéfini `request`

⇒ retourne le texte saisi

⇒ ou `null` si le nom de paramètre n'existe pas



The screenshot shows a web browser window with the title "C:\Mes Documents\Cours et TDs\99-00\sa\Norm.8.html". The address bar shows the local file path. The form contains two text input fields: "Nom" with the value "Seinturier" and "Prénom" with the value "Lionel". Below the inputs are two buttons: "Envoi" and "Remise à zéro".

```
<HTML> <BODY>
<FORM ACTION="http://..."
      METHOD=POST>
Nom <INPUT NAME="nom"> <P>
Prénom <INPUT NAME="prenom"> <P>
<INPUT TYPE=SUBMIT VALUE="Envoi">
<INPUT TYPE=RESET
      VALUE="Remise à zéro">
</FORM>
</BODY> </HTML>
```

Exemple

Récupération des données d'un formulaire

```
<HTML> <BODY>  
<H1>Exemple de résultat</H1>  
Bonjour  
<%= request.getParameter("prenom") %>  
<%= request.getParameter("nom") %>  
</BODY> </HTML>
```



Exemple

Gestion des erreurs

Erreur de syntaxe

- dans les directives JSP (ex. : oubli d'une directive %>)
- dans le code Java

Erreur d'exécution du code Java (ex. : `NullPointerException`)

⇒ dans tous les cas, erreur récupérée dans le **navigateur** client

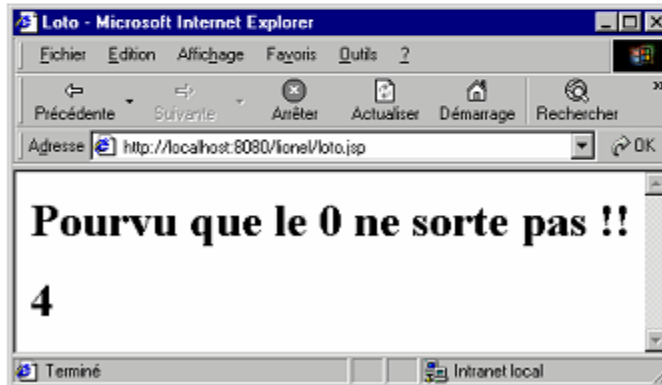
2 possibilités

- conserver la **page par défaut** construite par le moteur
- en concevoir une adaptée aux besoins particuliers de l'application
 - ⇒ utilisation des directives `<%@ page errorPage="..." %>` et `<%@ page isErrorPage="..." %>`

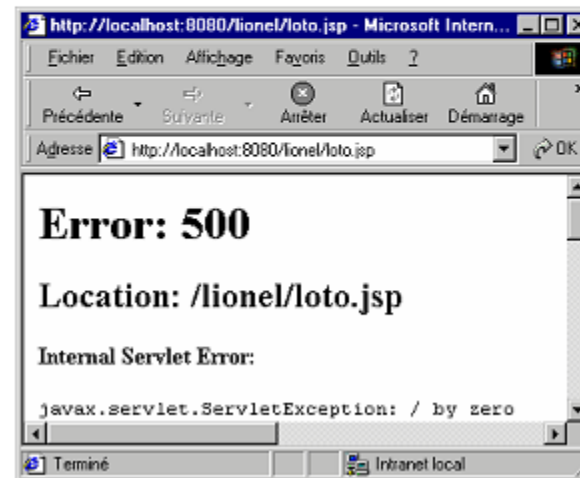
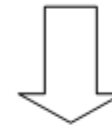
Exemple

Exemple de gestion d'erreur

```
<HTML> <BODY>
<H1>Pourvu ... !!</H1>
<% int hasard =
    (int) ( Math.random() * 5 );
%>
<H1> <%= 12 / hasard %> </H1>
</BODY> </HTML>
```



Si hasard = 0
page d'erreur par défaut



Exemple

Exemple de gestion d'erreur

```
<HTML> <BODY>
<H1>Pourvu ... !!</H1>
<%@ page
    errorPage="err.jsp" %>
<% int hasard = ... %>
<H1> <%= 12 / hasard %> </H1>
</BODY> </HTML>
```

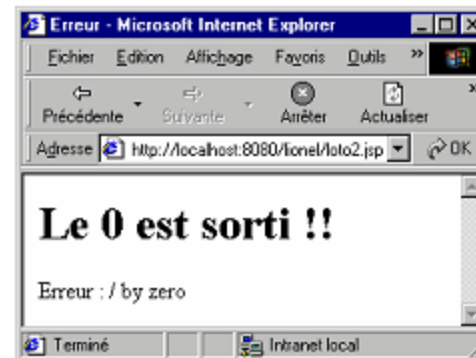
```
<HTML> <BODY>

<%@ page isErrorPage="true" %>
<h1>Le 0 est sorti !!</h1>
Erreur :
<%= exception.getMessage() %>

</BODY> </HTML>
```

Si hasard = 0
page d'erreur **err.jsp** = 0

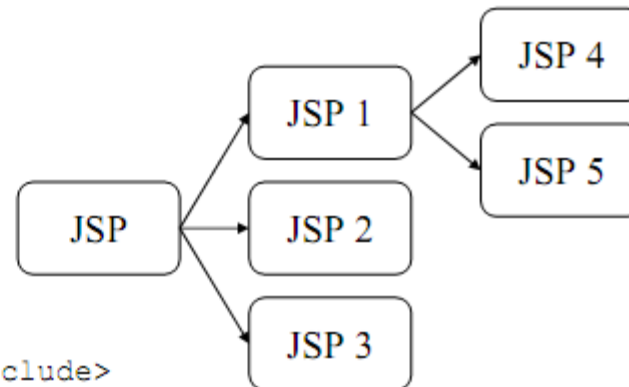
Récupération de l'erreur via
l'objet prédéfini **exception**



Exemple

Inclusion de JSP

- agrégation des résultats fournis par plusieurs JSP
 - ⇒ meilleure modularité
 - ⇒ meilleure réutilisation



Directives `<jsp:include>` et `</jsp:include>`

```
<HTML> <BODY>
<H1>JSP principale</H1>

<jsp:include
  page="inc.jsp" >
</jsp:include>

</BODY> </HTML>
```

URL

Fichier `inc.jsp`

```
<B>JSP incluse</B>

<P>
<%= (int) (Math.random()*5) %>
</P>
```

Pas de `<HTML>` `<BODY>`

Exemple

Inclusion de JSP

Résultat

```
<HTML> <BODY>  
<H1>JSP principale</H1>  
  
<B>JSP incluse</B>  
<P>  
<%= (int) (Math.random() * 5) %>  
</P>  
  
</BODY> </HTML>
```



Remarque

1. directives `<jsp:include>` et `</jsp:include>`
2. directive `<%@ page include file="..." %>`

inclusion statique
inclusion dynamique

Exemple

Inclusion de JSP

Résultat

```
<HTML> <BODY>
<H1>JSP principale</H1>

<B>JSP include</B>
<P>
<%= (int) (Math.random()*5) %>
</P>

</BODY> </HTML>
```



Remarque

1. directives `<jsp:include>` et `</jsp:include>`
2. directive `<%@ page include file="..." %>`

inclusion statique
inclusion dynamique

Syntaxe XML

- ▶ Depuis JSP 1.2
 - `<jsp:expression>Expression</jsp:expression>`
 - `<jsp:scriptlet>`
 - `<jsp:declaration>`
- ▶ Permet
 - La validation
 - Les transformations
 - L'édition

Example

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0">

  <jsp:directive.page contentType="text/html" pageEncoding="UTF-8"/>

  <!-- any content can be specified here, e.g.: -->
  <jsp:element name="text">
    <jsp:attribute name="lang">EN</jsp:attribute>
    <jsp:body>Hello World!<br/>
    <jsp:expression>new java.util.Date(System.currentTimeMillis()).toString()</jsp:ex
    </jsp:body>
  </jsp:element>

</jsp:root>
```

Jsp:forward

Délégation de JSP

Une JSP peut déléger le traitement d'une requête à une autre JSP

⇒ prise en compte **complète** de la requête par la JSP déléguée


Directives `<jsp:forward>` et `</jsp:forward>`

```
<HTML> <BODY>
<H1>JSP principale</H1>

<jsp:forward
  page="forw.jsp" >
</jsp:forward>

Ignoré !!

</BODY> </HTML>
```



Fichier forw.jsp

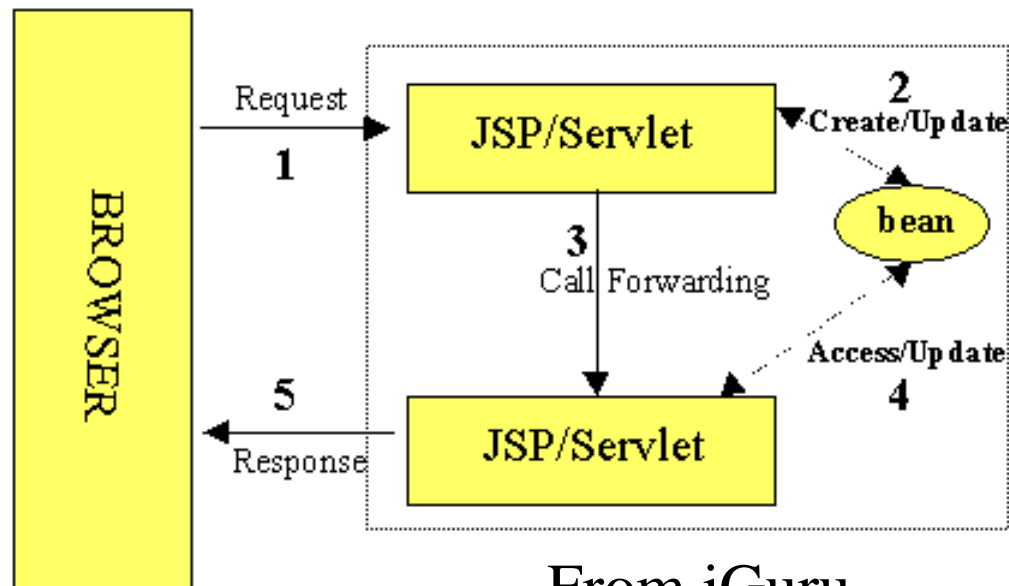
```
<HTML> <BODY>
<H1>JSP déléguée</H1>

<P>
  <%= (int) (Math.random()*5) %>
</P>
</BODY> </HTML>
```

```
<HTML> <BODY>
```

jsp:forward

- ▶ Jsp:forward permet de chainer les requêtes pour invoquer
 - Une autre page jsp
 - Un servlet
- ▶ `<jsp:forward page="AutrePage.jsp"/>`
- ▶ `<jsp:forward page="/Servlet/control/">`



From jGuru

Jsp:forward avec paramètres

- ▶ Il est possible d'ajouter des paramètres à la requête.
- ▶ Accessibles par `request.getAttribute(name)`

```
<jsp:forward page="chaine.jsp">  
<jsp:param name="name1" value="v1"/>  
<jsp:param name="name2" value="v2"/>  
</jsp:forward>
```

Jsp:include

- ▶ Redirige la requête et inclut le résultat à l'endroit où se trouve l'action
 - `<jsp:include page="checkBean.jsp"/>`
 - Cette action est exécutée à chaque fois
- ▶ L'action exécutée ne peut pas modifier le Header de la page (pas de cookie ni de type de contenu)

Les tags JSP (ou actions)

- ▶ Les tags sont des actions incluses dans une page Web suivant la syntaxe XML
 - `<mod:tag attr="value">`
 - `<mod:tag attr="value">body</mod:tag>`
- ▶ Les actions de base font partie de la librairie jsp:
 - `<jsp:useBean>`
 - `<jsp:setProperty>`
 - `<jsp:getProperty>`
 - `<jsp:include>`
 - `<jsp:forward>`
 - `<jsp:text>`

Un JavaBean

- ▶ Composant simple.
Respecte des
conventions
d'écriture

```
public class MyBean {  
    private String nom;  
    private int compte;  
    private Date date;  
  
    public String getNom() {  
        return nom;  
    }  
  
    public void setNom(String nom) {  
        this.nom = nom;  
    }  
  
    public int getCompte() {  
        return compte;  
    }  
  
    public void setCompte(int compte) {  
        this.compte = compte;  
    }  
  
    public Date getDate() {  
        return date;  
    }  
}
```

Exemple UseBean

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h2>Use Bean</h2>
    <jsp:useBean id="mybean" scope="page" class="exemple.MyBean" />
    <jsp:setProperty name="mybean" property="nom" value="Test" />
    <jsp:setProperty name="mybean" property="compte" value="0" />

    <jsp:getProperty name="mybean" property="nom"/>
  </body>
</html>
```

Autre exemple

```
<body>
  <h2>Use Bean 2</h2>
  <jsp:useBean id="bean2" scope="page" class="exemple.MyBean" />
  <jsp:setProperty name="bean2" property="nom" param="nom" />
  Valeur du paramètre = <jsp:getProperty name="bean2" property="nom" />
</body>

<body>
  <h2>Use Bean 3</h2>
  <jsp:useBean id="mybean" scope="request" class="exemple.MyBean" />
  <jsp:setProperty name="mybean" property="*" />
  Nom= <jsp:getProperty name="mybean" property="nom" /> <br>
  Compte = <jsp:getProperty name="mybean" property="compte" />
</body>
```

 <http://localhost:8084/CoursWeb/useBean3.jsp?nom=test&compte=100>

Use Bean 3

Nom= test

Compte = 100

useBean et scope

```
<body>
  <h2>Scope 1</h2>
  <jsp:useBean id="aBean" scope="session" class="exemple.MyBean" />
  <jsp:setProperty name="aBean" property="*" />
</body>
```

```
<body>
  <h2>Scope 2</h2>
  <jsp:useBean id="aBean" scope="session" class="exemple.MyBean" />
  <jsp:getProperty name="aBean" property="nom" />
</body>
```

Scope 2

test

Code généré

```
synchronized (session) {  
    aBean = (exemple.MyBean) _jspx_page_context.getAttribute("aBean", PageContext.SESSION_SCOPE);  
    if (aBean == null){  
        aBean = new exemple.MyBean();  
        _jspx_page_context.setAttribute("aBean", aBean, PageContext.SESSION_SCOPE);  
    }  
}
```


JavaBean et JSP

- ▶ Les action `useBean`, `setProperty` et `getProperty` permettent de manipuler des JavaBean sans programmation
 - `jsp:usebean` pour nommer, créer ou désigner un bean
 - `jsp:getProperty` pour récupérer une propriété d'un bean
 - `jsp:setProperty` pour changer la valeur d'une propriété.

Les directives

- ▶ `<%@ directive {attr="value"}* %>`
 - Messages passés au moteur de JSP
- ▶ Page : propriétés de la page
 - **extends**="*ClassName*"
 - La page est une sous-classe de la classe indiquée
 - **import**="javax.jms.*,cour.test.util"
 - import des types ou des packages
 - Les packages lang, servlet et jsp sont importés par défaut
 - **session**="true" ou "false" (default=true)
 - La page participe à une session
 - **isThreadSafe**
 - **buffer**=16k
 - Taille du buffer par défaut pour le PrintWriter.
 - **autoFlush**="true" ou "false"
 - Vide le buffer automatiquement quand le buffer est plein

Les propriétés

- ▶ El-enabled : permet l'utilisation du langage d'expression
 - ▶ Scripting-enabled : permet l'utilisation du langage de script
 - ▶ Page-encoding :
 - ▶ Include-prelude : ajoute un header à toutes les pages
 - ▶ Include-code : ajoute un footer après la génération de la page
 - ▶ Is-xml : permet la validation XML des pages JSP
- 