Cours Web

Java Server Pages (JSP)

Lionel Seinturier

Université Pierre & Marie Curie

Lionel.Seinturier@lip6.fr

11/7/02

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

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**

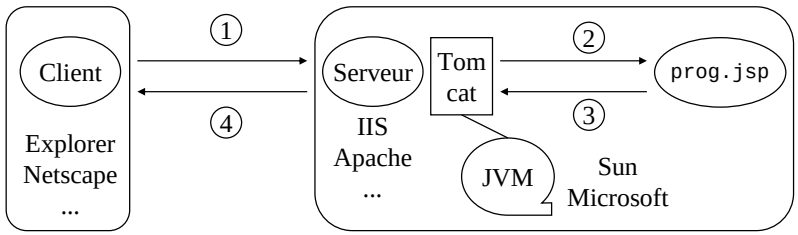
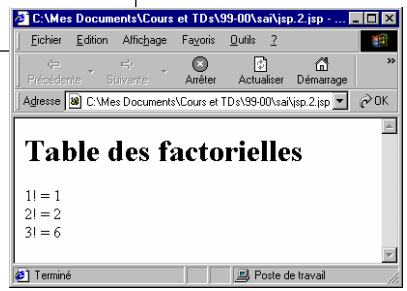


Illustration du fonctionnement

```
<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



Principe de 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>
```

du code Java

résultat = **HTML** généré via l'objet prédéfini **out**

ce qui est renvoyé au client

```
<HTML> <BODY>
<H1>Table des factorielles</H1>
1! = 1<BR>
2! = 2<BR>
3! = 6<BR>
</BODY> </HTML>
```


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***

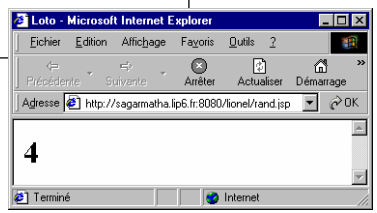
⇨ délai d'attente lors de la 1ère invocation dû à la compilation

⇨ en cas d'erreur de syntaxe dans le code Java de la JSP
message récupéré dans le navigateur

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>
```



Méthodes et variables d'instance

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

```
<HTML> <BODY>
<H1>Compteur</H1>
<%!
   int cpt = 0;

   int getCpt() {
     return cpt++;
   }
%>

<H1> <%= getCpt() %> </H1>

</BODY> </HTML>
```

Variable d'instance
- initialisée à l'instanciation de la JSP
- **persiste** entre 2 invocations tant que la JSP ne change pas

Méthode d'instance
- attachée à l'objet correspondant à la JSP

Variables d'instance

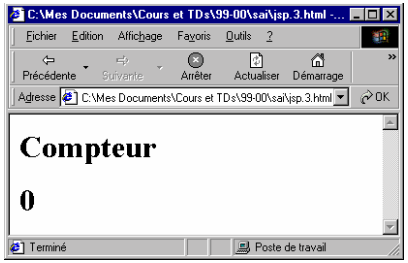
Attention !!

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

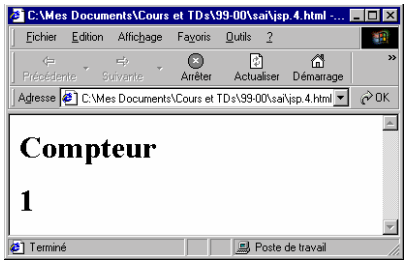
- variable **d'instance** de la JSP (**persiste**)
- variable **locale** à la JSP (**réinitialisée** à chaque invocation de la JSP)

Exemple

1ère invocation



2ème invocation



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 +sieurs clients à la fois (valeur par défaut)

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

Les objets implicites

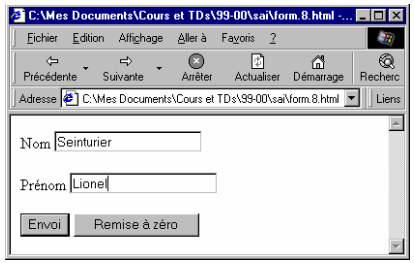
Objets prédéclarés utilisables dans le code Java des JSPs

`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

Récupération des données d'un formulaire

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

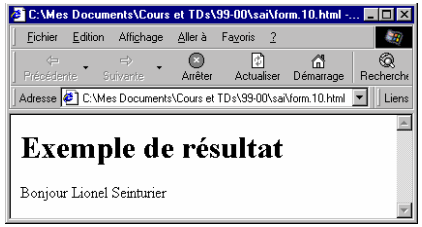
⇨ retourne le texte saisi
⇨ ou null si le nom de paramètre n'existe pas




```
<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>
```


Récupération des données d'un formulaire

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



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`)

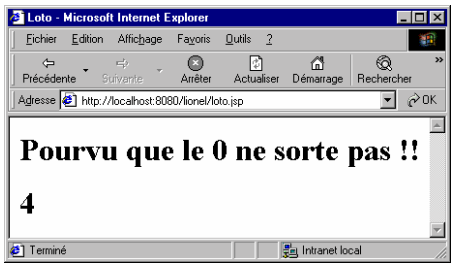
⇨ dans tous les cas, erreur récupérée dans le **navigateur** client

2 possibilités

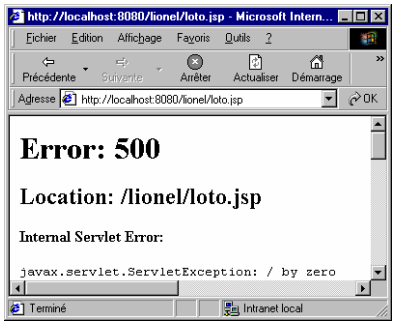
• conserver la **page par défaut** construite par le moteur
• en concevoir une adaptée aux besoins particuliers de l'application
⇨ utilisation des directives `<%@ page errorPage="..." %>` et `<%@ page isErrorPage="..." %>`

Exemple 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 de gestion d'erreur

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

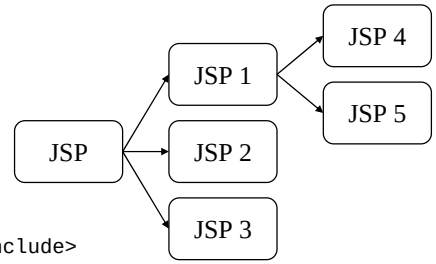
```
<HTML> <BODY>

<%@ 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`**

Inclusion de JSP

- aggrégation 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`

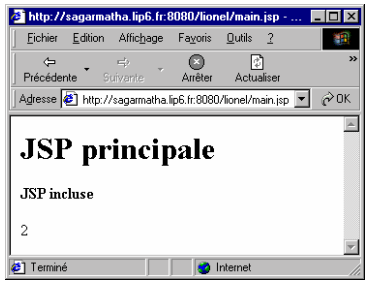
```
<B>JSP incluse</B>

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

Pas de `<HTML> <BODY>`

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>` — inclusion statique
2. directive `<%@ page include file="…"%>` — inclusion dynamique

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>
```

URL

Fichier `forw.jsp`

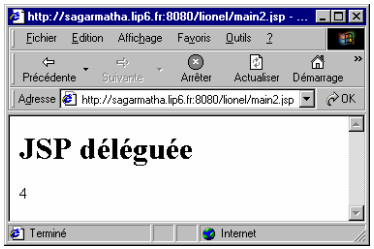
```
<HTML> <BODY>
<H1>JSP déléguée</H1>

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

`<HTML> <BODY>`

Délégation de JSP

Résultat

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

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

</BODY> </HTML>
```



Délégation et inclusion de JSP

Transmission de paramètres aux **inclus** et aux **délégués**
Utilisation de couples (`name`, `value`)

Directive `<jsp:param name="..." value="..." />`

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

<jsp:forward page="forw.jsp">
   <jsp:param name="nom" value="Seinturier" />
   <jsp:param name="prenom" value="Lionel" />
</jsp:forward>

</BODY> </HTML>
```


Délégation et inclusion de JSP

Récupération des paramètres
≡ à la récupération des paramètres transmis via un formulaire

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

Nom : <%= request.getParameter("nom") %>
Prénom : <%= request.getParameter("prenom") %>

</BODY> </HTML>
```


Compléments sur l'API

Méthodes appelables sur l'objet prédéfini `request`

- `String getProtocol()`
  retourne le protocole implanté par le serveur (ex. : HTTP/1.1)
- `String getServerName() / String getServerPort()`
  retourne le nom/port de la machine serveur
- `String getRemoteAddr() / String getRemoteHost()`
  retourne l'adresse/nom de la machine cliente (ayant invoqué la servlet)
- `String getScheme()`
  retourne le protocole utilisé (ex. : http ou https) par le client

Suivi de session

• HTTP protocole non connecté
• pour le serveur, 2 requêtes successives d'un même client sont **indépendantes**

Objectif : être capable de "suivre" l'activité du client sur +sieurs pages

Notion de session

⇨ les **requêtes** provenant d'un **utilisateur** sont associées à une même session
⇨ les sessions ne sont pas éternelles, elles **expirent** au bout d'un délai fixé

Objet prédéfini `session` de type `HttpSession`

⇨ la session courante ou une nouvelle session

Suivi de session

Méthodes appelables sur l'objet prédéfini `session`

- `void setAttribute( String name, Object value )`
  ajoute un couple (`name`, `value`) pour cette session
- `Object getAttribute( String name )`
  retourne l'objet associé à la clé `name` ou `null`
- `void removeAttribute( String name )`
  enlève le couple de clé `name`
- `java.util.Enumeration getAttributeNames()`
  retourne tous les noms d'attributs associés à la session
- `void setMaxIntervalTime( int seconds )`
  spécifie la durée de vie maximum d'une session
- `long getCreationTime() / long getLastAccessedTime()`
  retourne la date de création / de dernier accès de la session
  en ms depuis le 1/1/1970, 00h00 GMT → `new Date(long)`

Partage de données entre JSP

Notion de contexte d'exécution
= ensemble de couples (`name`, `value`) partagées par toutes les JSP **instanciées**

⇨ objet prédéfini `application`

Méthodes appelables sur l'objet prédéfini `application`

- `void setAttribute( String name, Object value )`
  ajoute un couple (`name`, `value`) dans le contexte
- `Object getAttribute( String name )`
  retourne l'objet associé à la clé `name` ou `null`
- `void removeAttribute( String name )`
  enlève le couple de clé `name`
- `java.util.Enumeration getAttributeNames()`
  retourne tous les noms d'attributs associés au contexte

Conclusion

Permettent d'étendre le comportement des serveurs Web avec des prog. Java

Résumé des fonctionnalités

+ code embarqué dans un fichier HTML

+ portabilité, facilité d'écriture (**Java**)
+ gestion des applications requièrant un suivi entre plusieurs programmes (**persistance des données**)

+ JSP chargée et instanciée **une seule fois**
+ JSP exécutée avec des processus légers (***threads***)

Tomcat

Le moteur de JSP et de servlet le plus utilisé
Projet soutenu conjointement par Apache et Sun

Nécessite une machine virtuelle (Sun ou Microsoft)

2 modes de fonctionnement

Autonome (*standalone*)
- Tomcat est **aussi** un serveur Web
- il est capable de servir des pages HTML, d'exécuter des servlets et des JSP

Collaboratif (*in-process* et *out-of-process*)
- Tomcat peut s'installer comme une extension d'un serveur Web
  (Apache, Microsoft IIS ou Netscape NetServer)

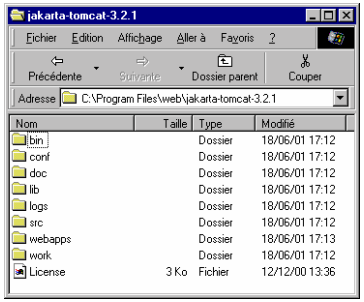
⇨ meilleures performances pour le service des pages HTML

Tomcat

Installation

- récupérer d'un `.zip` sur le site
- dézipper

⇨ occupation disque 11 Mo



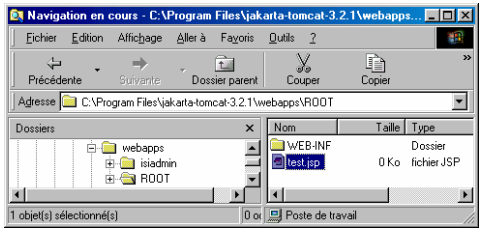
- bin — Scripts de démarrage/d'arrêt
- conf — Fichiers de configuration (en particulier *server.xml*)
- doc — Documentation
- lib — Librairies utilisées Tomcat
- logs — Répertoire pour les fichiers d'audit
- src — Sources de Tomcat

- webapps — Répertoire de **dépôt des .jsp et des servlets**

Tomcat

Répertoire webapps\ROOT

- dépose des fichiers
- création de sous-répertoires



Après lancement de Tomcat, tous les fichiers déposés dans webapps sont accessibles via l'URL `http://nom.de.ma.machine:8080/`

Ex. : fichier `test.jsp` dans webapps\ROOT
URL `http://nom.de.ma.machine:8080/test.jsp`

Rq : les `.class` des servlets sont à déposer dans WEB-INF\classes