

# Servlets et JSP

Gaël Thomas

[gael.thomas@lip6.fr](mailto:gael.thomas@lip6.fr)

(basé sur un cours de Lionel Seinturier)

Université Pierre et Marie Curie

Master Informatique

M2 – Spécialité SAR

## Introduction

### Servlet et 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
Classe autonome	Applet	<b>Servlet</b>
Embarqué dans du HTML	JavaScript	<b>JSP</b>

### Servlet et JSP

- ✓ Exécutables avec tous les serveurs Web (Apache, IIS, ...)
- ✓ Auxquels on a ajouté un "moteur" de servlet/JSP (le plus connu : **Tomcat**)
- ✓ Les JSP sont compilés automatiquement en servlet par le moteur

## Introduction

### Gestion de page HTML à contenu dynamique

- ✓ Contenu généré au moment de la consultation/affichage
- ✓ Nécessaire pour
  - Afficher des données dépendantes d'une base de donnée (qui évolue au cours du temps)
  - Créer des sessions HTTP avec les clients (préférences, paniers etc...)
  - Enrichir une page HTML avec du code additionnel

### Création dynamique du contenu chez

- ✓ Le client : nécessite un navigateur Web enrichi
- ✓ Le serveur : nécessite un serveur Web enrichi

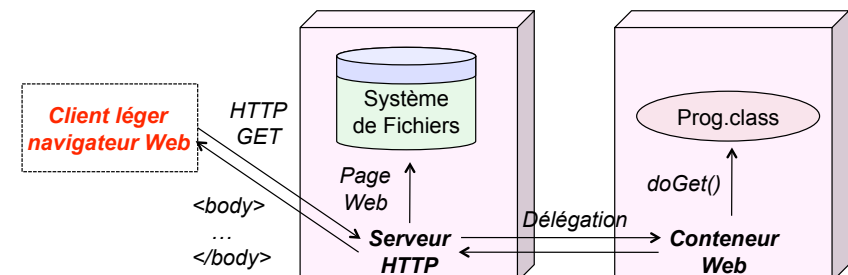
### Création dynamique du contenu par

- ✓ Du code embarqué dans le code HTML
- ✓ Du code extérieur à la page HTML

## Introduction

### Principe des Servlets

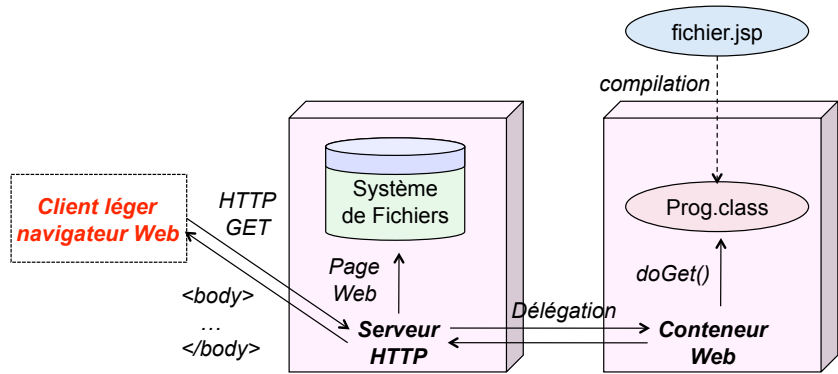
- ✓ Les fichiers de *bytecode* (.class) sont stockés sur le serveur
- ✓ Gérés dans un conteneur Web (Tomcat par exemple)
- ✓ Elles sont désignées par une **URL** <http://www.lip6.fr/maservlet/Prog>
- ✓ Le **chargement** de l'URL provoque l'**exécution** de la servlet Prog.class via le conteneur Web (parfois appelé conteneur de Servlet)



# Introduction

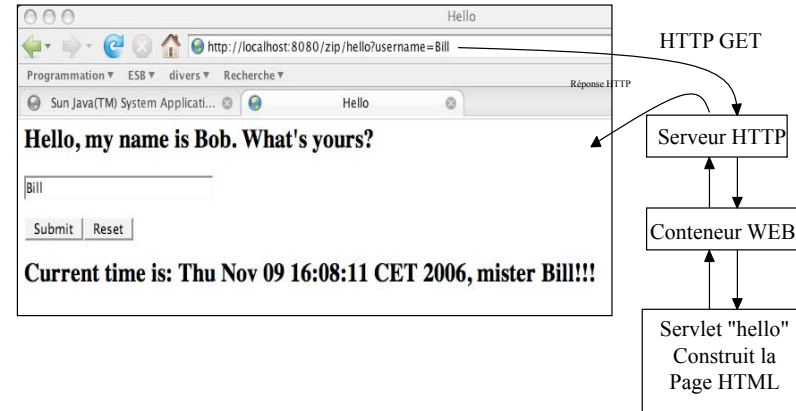
## Principe des JSP

Un fichier JSP est compilé vers une servlet!

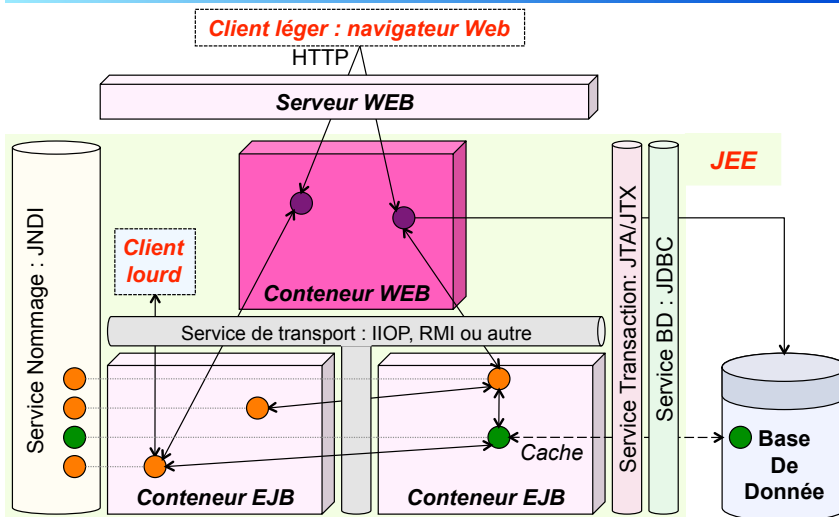


# Introduction

## Illustration du fonctionnement des servlets



# Introduction



# Première étape : les Servlets

## Servlet : mise en œuvre

Écriture d'une servlet = écriture d'une **classe Java**

Lors du premier chargement d'une servlet (ou après modification), le **conteneur Web**

- ✓ **instancie** (alloue et initialise) la servlet
- ✓ servlet = objet Java présent dans le moteur

Puis, ou lors des chargements suivants, le **conteneur Web**

- ✓ Exécute le code dans un **thread**
- ✓ Le code produit un résultat qui est envoyé au client
- ✓ En cas d'erreur dans le code Java de la servlet, un message est récupéré dans le navigateur

## Servlet : mise en œuvre

Développement d'une Servlet

Utilisation des packages Java `javax.servlet.*` et `javax.servlet.http.*`

- ✓ Extension de la classe `javax.servlet.http.HttpServlet`
- ✓ Redéfinition de la méthode `doGet` ou `doPost` de cette classe
  - `doGet` : correspond à une requête HTTP GET
  - `doPost` : correspond à une requête HTTP POST
- ✓ Définit le code à exécuter lorsque la servlet est invoquée

Appelée automatiquement par le conteneur Web lors d'une requête

```
void doGet(HttpServletRequest request, HttpServletResponse response);
```

**Requête** envoyée par le client  
! renseigné automatiquement par le conteneur Web

**Réponse** HTML retournée par la servlet  
! à **renseigner** dans le code de la servlet

## Servlet : mise en œuvre

Aperçu de l'API servlet

Méthodes importantes d'un objet **request**

- ✓ `String getParameter(String param)`  
Retourne la **valeur du champ** param transmis dans les données du formulaire
- ✓ `java.util.Enumeration getParameterNames()`  
retourne l'ensemble des noms de paramètres transmis à la servlet
- ✓ `String getMethod()`  
retourne la méthode HTTP (GET ou POST) utilisée pour invoquer la servlet

## Servlet : mise en œuvre

Aperçu de l'API servlet

Méthodes importantes d'un objet **response**

- ✓ `void setContentType(String type)`  
définit le **type MIME** du document retourné par la servlet
- ✓ `PrintWriter getWriter()`  
retourne un **flux de sortie** permettant à la servlet de produire son résultat  
la servlet écrit le code HTML sur ce flux de sortie

## Servlet : mise en œuvre

Exemple de servlet : le code de la servlet

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

public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response ) throws
        ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // écriture du HTML pour le client
        out.println("<html><body>Current time is: " +
            new Date() + ".</body></html>");
        out.close(); } }
```

## Cycle de vie des servlets

Chaque servlet n'est instanciée qu'une seule fois  
⇒ persistance des variables d'instance entre 2 invocations

```
public class CompteurServlet extends HttpServlet {
    int compteur = 0;

    public void doGet( HttpServletRequest request, HttpServletResponse response )
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter(); out.println("<html><body>");
        out.println("<h1> " + compteur++ + "</h1>");
        out.println("</body></html>");
    }
}
```

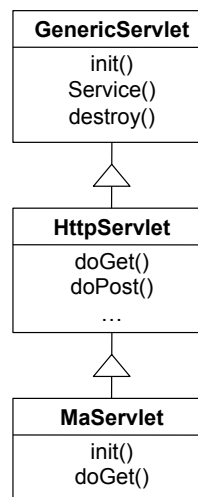
## Cycle de vie des servlets

Cycle de vie d'une servlet

- ✓ void init(ServletConfig conf)
  - méthode appelée par le moteur au démarrage de la servlet
  - peut être utilisée pour initialiser la servlet
  - propager l'initialisation par super.init(conf)
  - ne jamais utiliser de constructeur pour initialiser une Servlet
- ✓ void destroy()
  - Méthode appelé avant la destruction de la servlet

Différenciation des méthodes HTTP

- ✓ service() traite toutes les requêtes HTTP
- ✓ doGet(), doHead(), doPost(), doPut() doDelete(), doTrace()  
Traitement différencié de chaque requête HTTP



## Chaînage de Servlets

Agrégation des résultats fournis  
par plusieurs servlets

- ✓ Meilleure modularité
- ✓ Meilleure réutilisation

Utilisation d'un *RequestDispatcher*

- ✓ obtenu via l'objet request

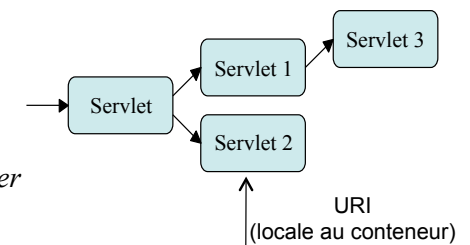
RequestDispatcher rd = request.getRequestDispatcher("servlet2");

Inclusion du résultat d'une autre servlet

- ✓ rd.include(request, response);

Délégation du traitement à une autre servlet

- ✓ rd.forward(request, response);



## Complément sur l'API des Servlets

### Méthodes appelables sur un objet 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
- ✓ `java.io.BufferedReader getReader()` :  
retourne un flux d'entrée permettant à une servlet chaînée de récupérer le résultat produit par la servlet précédente
  - permet à la servlet chaînée de modifier le résultat

## Cookies et Sessions dans les Servlets

### Cookie = donnée stockée par un serveur Web chez un client

- ✓ Moyen pour savoir "par où passe" un client, quand, en venant d'où, ...
- ✓ Débat éthique ??
- ✓ L'utilisateur a la possibilité d'interdire leur dépôt dans son navigateur

### Définis dans la classe `javax.servlet.http.Cookie`

- ✓ Créé en donnant un nom (String) et une valeur (String) `Cookie`  
`uneCookie = new Cookie("sonNom", "saValeur");`
- ✓ Positionné via un objet `response` `response.addCookie(uneCookie);`
- ✓ Récupéré via un objet `request` `Cookie[] desCookies = request.getCookies();`
- ✓ Quelques méthodes : `String getName()` / `String getValue()`

## Complément sur l'API des Servlets

### Gestion de la concurrence

- ✓ Par défaut les servlets sont exécutées de façon *multi-threadée*
- ✓ Si une servlet doit être exécutée en exclusion mutuelle (ex. : accès à des ressources partagées critiques)  
implantation de l'interface marqueur `SingleThreadModel`

```
public class CompteurServlet extends HttpServlet implements SingleThreadModel {  
    public void doGet( ServletRequest request, ServletResponse response )  
        throws ServletException, IOException {  
        /* Tout le code de la Servlet est en exclusion avec lui-même */ } }  
}
```

### Autre solution : définir du code `synchronized` dans la servlet

Déconseillé car le conteneur Web possède des verrous  
⇒ peut conduire à des inter-bloquages

## Cookies et Sessions dans les Servlets

### Suivi de session

- ✓ HTTP protocole non connecté  
⇒ 2 requêtes successives d'un même client sont **indépendantes** pour le serveur

### Notion de session : **suivi** de l'activité du client sur plusieurs pages

- ✓ Un objet `Session` associé à toutes les **requêtes** d'un utilisateur (= adresse IP + navigateur)
- ✓ Les sessions **expirent** au bout d'un délai fixé (pas de requête pendant n secondes ⇒ expiration de la session)

### Consultées/crées à partir d'un objet request

- ✓ `HttpSession session = request.getSession(true);`  
retourne la session courante pour cet utilisateur ou une nouvelle session
- ✓ `HttpSession session = request.getSession(false);`  
retourne la session courante pour cet utilisateur ou null

## Cookies et Sessions dans les Servlets

### Méthodes appelables sur un objet de type HttpSession

- ✓ 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 **setMaxInactiveInterval**( int seconds );  
spécifie le temps avant la fermeture 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);

## Cookies et Sessions dans les Servlets

### Partage de données entre servlets

Contexte d'exécution = ensemble de **couples** (name, value) partagées par toutes les servlets **instanciées**

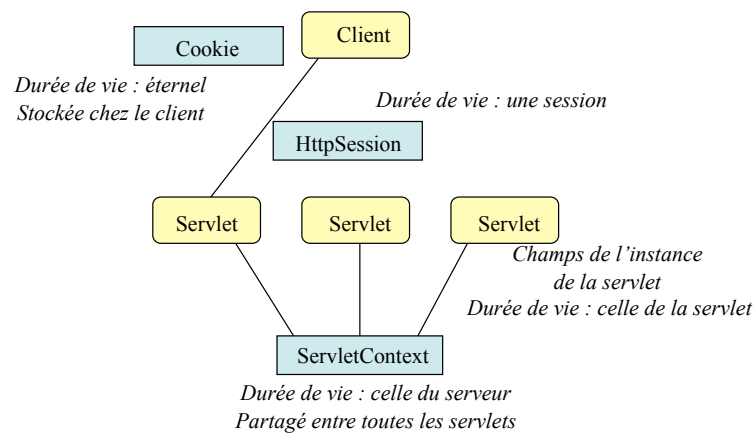
ServletContext ctx = **getServletContext**()

### Méthodes appelables sur un objet de type ServletContext

- ✓ 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

## Cookies et Sessions dans les Servlets

### Récapitulatif des objets de données



## Servlet : conclusion

### Servlets : étendent le comportement des serveurs Web avec des programme Java

- ✓ Portabilité, facilité d'écriture (**Java**)
- ✓ Définition du code, du paquetage, du déploiement
- ✓ **Persistance des données** dans les servlets  
Servlet chargée et instanciée **une seule fois**
- ✓ Exécutée en // avec des processus légers (**threads**)

### Mais :

- ✓ Difficile d'écrire du code HTML dans du code Java  
Introduction de la technologie Java Server Pages (JSP)
- ✓ Pas de mécanisme intégré de distribution  
Introduction de la technologie Enterprise Java Beans (EJB)

## Deuxième étape : les JSP

## JSP : 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-1) + "! = " + fact + "<br>" );
   } %>
</body></html>
```

Invocation  
⇒  
Exécution  
Côté serveur



## JSP : illustration du fonctionnement

## JSP : illustration du fonctionnement

<pre>&lt;html&gt;&lt;body&gt; &lt;h1&gt;Table des factorielles&lt;/h1&gt;</pre>	Code HTML
<pre>&lt;% int i,fact;    for ( i=1,fact=1 ; i&lt;4 ; i++, fact*=i ) {        out.print( (i-1) + "! = " + fact + "&lt;br&gt;" );</pre>	Code Java
<pre>   } %&gt;</pre>	Produit HTML
<pre>&lt;/body&gt;&lt;/html&gt;</pre>	Code HTML

Code  
Envoyé  
Au  
Client

```
<html><body>
<h1>Table des factorielles</h1>
0!=1<br>
1!=1<br>
2!=2<br>
3!=6<br>
</body></html>
```

<pre>&lt;html&gt;&lt;body&gt; &lt;h1&gt;Table des factorielles&lt;/h1&gt; &lt;% int i,fact;    for ( i=1,fact=1 ; i&lt;4 ; i++, fact*=i ) {        out.print( (i-1) + "! = " + fact + "&lt;br&gt;" );    } %&gt; &lt;/body&gt;&lt;/html&gt;</pre>	
Après Compilation	<pre>public final class fact_jsp /* ... */ { public void _jspService(HttpServletRequest request,                         HttpServletResponse response) /*  * ... */ {     PrintWriter out = response.getWriter();     response.setContentType("text/html");     /* ... */     out.write("&lt;html&gt;&lt;body&gt;\n");     out.write("&lt;h1&gt;Table des factorielles&lt;/h1&gt;\n");     int i,fact;     for ( i=1,fact=1 ; i&lt;4 ; i++, fact*=i ) {         out.print( (i-1) + "! = " + fact + "&lt;br&gt;" );     }     out.write("\n");     out.write("&lt;/html&gt;&lt;/body&gt;\n"); /* ... */ }</pre>

## JSP : mise en œuvre

Plusieurs zones `<% ... %>` peuvent cohabiter dans une même page

Lors du premier chargement d'une jsp

- ✓ Génération d'une servlet à partir de la jsp
- ✓ Compilation de la servlet
- ✓ Instanciation de la servlet
- ⇒ Délai d'attente lors de la première consultation
- ⇒ En cas d'erreur de syntaxe, message envoyé au navigateur (erreurs détectées à l'exécution)

Lors des chargements suivants

Exécution de la servlet dans un thread

## JSP : mise en œuvre

Les objets implicites

Objets pré-déclarés 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
- ✓ 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

## JSP : mise en œuvre

Les objets implicites : équivalents servlets

Objets pré-déclarés utilisables dans le code Java des JSPs

- ✓ out `response.getWriter()`
- ✓ request le paramètre `HttpServletRequest` de `service()`
- ✓ response le paramètre `HttpServletResponse` de `service()`
- ✓ Page `this`
- ✓ exception *pas d'équivalent immédiat*
- ✓ session `request.getSession(true)`
- ✓ application `getServletContext()`

## JSP : mise en œuvre

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> Nombre aléatoire : <%= aleat %> </h1>
</body> </html>
```

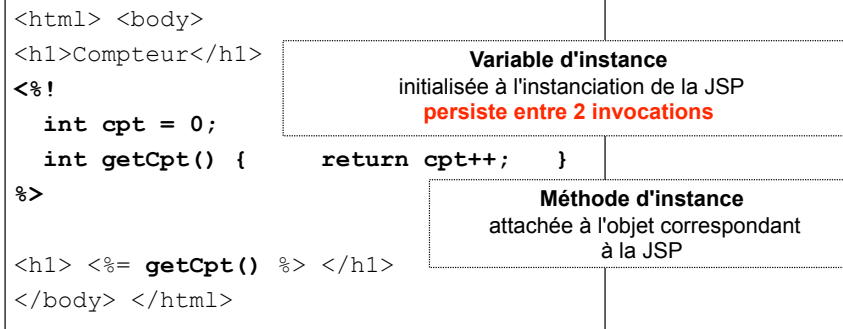


## JSP : mise en œuvre

### Méthodes et variables d'instance

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

#### Méthodes et variables d'instance de la servlet générée

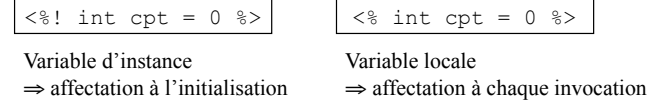


## JSP : mise en œuvre

### Méthodes et variables d'instance

- ✓ Méthode d'instance de la servlet générée à partir de la jsp
- ⇒ Pas d'accès aux objets implicites (out, request, page...) : ce sont des objets définis dans la méthode principale de la servlet (`_jspService()`)

#### Attention!



- `<%! ... %>` : définit une variable d'instance (persiste entre deux invocations)
- `<% ... %>` : définit une variable locale à la jsp (réinitialisée à chaque invocation)

## JSP : gestion des erreurs

### Erreur de syntaxe

- ✓ Dans le code HTML
- ✓ Dans les directives JSP (ex. : oubli d'une directive `%>`)
- ✓ Dans le code Java (ex. : oubli de ;)

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

Dans tous les cas, erreur récupérée dans le navigateur client

- ✓ Conserver la page par défaut construite par le moteur
- ✓ En concevoir une adaptée aux besoins particuliers de l'application
- ✓ Utilisation des directives :
  - `<%@ page errorPage="..." %>` : URL du gestionnaire d'erreurs
  - `<%@ page isErrorPage="..." %>` : vrai si page est un gestionnaire d'erreurs


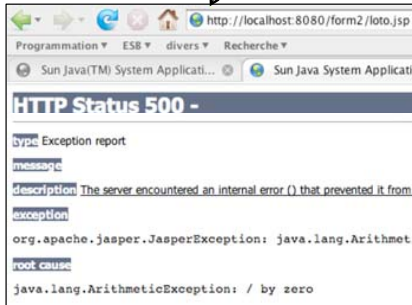
## JSP : gestion des erreurs

### Exemple

```
<html><body>
<h1>Test d'une erreur</h1>
<% int rand = (int)(Math.random() *2); %>
<h1>Resultat: <%= 12/rand %></h1>
</html></body>
```

Si `rand = 0`,  
page d'erreur par défaut

Si `rand ≠ 0`,  
page d'erreur par défaut

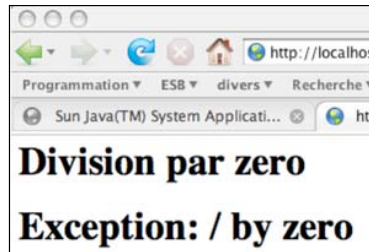


## JSP : gestion des erreurs

### Exemple de gestion d'erreur

```
<html><body>
<h1>Test d'une erreur</h1>
<%@ page
  errorPage="err.jsp" %>
<% int rand =
  (int) (Math.random() *2);%>
<h1>Resultat:
  <%= 12/rand %></h1>
</html></body>
```

```
<html><body>
<%@ page isErrorPage="true"%>
<h1>Division par zero</h1>
<h1>Exception:
  <%= exception.getMessage() %>
</h1> </html></body>
```



Si rand = 0,  
délégation de la requête à err.jsp  
Récupération de l'erreur via l'objet  
prédéfini exception

## Chaînage de JSP

### Inclusion de JSP

#### Deux types d'inclusion

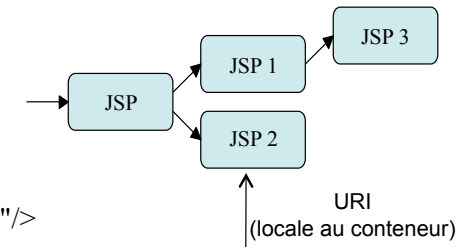
- ✓ `<jsp:include page="..." />` inclusion dynamique  
(délégation de servlets **deux servlets**)
- ✓ `<%@ include file="..." %>` inclusion statique  
(inclusion au niveau source : **une seule servlet**)

Inclusion statique
<pre>&lt;HTML&gt; &lt;BODY&gt; &lt;H1&gt;JSP déléguée&lt;/H1&gt; &lt;P&gt; &lt;%= (int) (Math.random()*5) %&gt; &lt;/P&gt; &lt;/BODY&gt; &lt;/HTML&gt;</pre>

## Chaînage de JSP

### Agrégation des résultats fournis par plusieurs JSP

- ✓ Meilleure modularité
- ✓ Meilleure réutilisation



### Directive `<jsp:include page="..." />`

main.jsp
<pre>&lt;html&gt; &lt;body&gt; &lt;h1&gt;JSP principale&lt;/h1&gt;  &lt;jsp:include   page="inc.jsp" /&gt; &lt;/body&gt; &lt;/html&gt;</pre>

inc.jsp
<pre>&lt;b&gt;JSP include&lt;/b&gt;  &lt;p&gt; &lt;%= (int) (Math.random()*5) %&gt; &lt;/p&gt;</pre>
<p><b>Fichier inclus pas de &lt;HTML&gt; &lt;BODY&gt;</b></p>

## Chaînage de JSP

### 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 page="..." />`

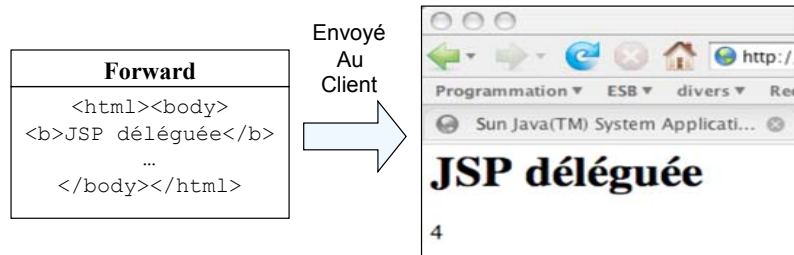
main.jsp
<pre>&lt;html&gt; &lt;body&gt; &lt;h1&gt;Code ignoré&lt;/h1&gt;  &lt;jsp:include   page="frow.jsp" /&gt;</pre>

forw.jsp
<pre>&lt;html&gt;&lt;body&gt; &lt;b&gt;JSP déléguée&lt;/b&gt; ... &lt;/body&gt;&lt;/html&gt;</pre>
<p><b>Fichier délégué pas de &lt;HTML&gt; &lt;BODY&gt;</b></p>

## Chaînage de JSP

### Délégation de JSP

La jsp d'origine est totalement ignorée

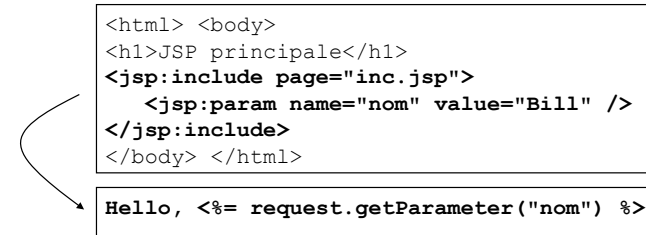


## Chaînage de JSP

### Délégation et inclusion de JSP

Transmission de paramètres aux jsp incluses et déléguées

- ✓ Utilisation de couples (name, value)
  - ✓ Directive `<jsp:param name="..." value="..." />`
- Récupération des paramètres : `request.getParameter("name")`



## Comparaison Servlet/JSP

JSP : compilé en servlet

Servlet : possibilité de distinguer les requêtes HTTP (doPut, doGet, doPost, ...)

JSP : bcp HTML, peu Java

Servlet : bcp Java, peu HTML

contenu autre que HTML (PDF, GIF, Excel, ...) : servlet oui / JSP oui mais

session, chaînage, redirection : oui dans les 2 cas : API vs directives

servlet : pur Java : facilement éditable dans IDE

JSP : plutôt éditeur de pages HTML

servlet compilation avant déploiement / JSP après

JSP à redéployer si erreur

## Déploiement et exécution

## Packaging

Composant Web : entité correspondant à une URL

- ✓ Une servlet
- ✓ Un fichier jsp

Application Web : unité de déploiement

- ✓ Ensemble de composants Web

Fichier de déploiement

- ✓ Description des différents composants constituant l'application Web  
**Standardisé**, placé dans un fichier appelé web.xml
- ✓ Description de la façon d'installer l'application Web  
**Non standardisé**, dépend du conteneur utilisé, en général nomduconteneur-web.xml

## Packaging

Description des composants Web (**web.xml**, standardisé)

```
<web-app>
  <servlet>
    <servlet-name>Hello</servlet-name> ←
    <servlet-class>HelloServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Hello</servlet-name> ←
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

Nom de la servlet

Localisation du composant web  
Dans l'application Web  
`http://server:port/??/hello`

Localisation de l'application Web dans le conteneur

## Packaging

Description de l'application Web pour le serveur Sun (**sun-web.xml**)  
**non standardisé, dépend du Conteneur Web**

```
<sun-web-app>
  <context-root>/test</context-root>
</sun-web-app>
```

Racine de l'application Web  
`http://server:port/test`

⇒ `http://server:port/test/hello` appelle de la méthode doGet de la classe HelloServlet

*Exemple non utilisé lors du déploiement*

## Packaging

Packaging de l'application Web : dans un fichier war  
(Web Archive, standardisé)

/ : Ressources directement projetées

- `/index.html` correspond à `http://.../test/index.html`
- `/hello.jsp` correspond à `http://.../test/hello.jsp`

/WEB-INF/ : descripteur de l'application Web

- `/WEB-INF/web.xml`
- `/WEB-INF/sun-web.xml`

/WEB-INF/classes/ : fichiers de classes

- `/WEB-INF/classes/HelloServlet.class` correspond à `http://.../test/hello`

## Déploiement

Déploiement : dans un conteneur Web

Tomcat : le conteneur Web le plus utilisé

- ✓ Moteur d'exécution de servlets et de jsp
- ✓ Projet soutenu conjointement par Apache et Sun
- ✓ Codé en Java (nécessite une machine virtuelle Java)

2 modes de fonctionnement

- ✓ Autonome (standalone)
  - Tomcat est aussi un serveur Web
  - Capable de servir des pages HTML, d'exécuter des servlets et des JSP
- ✓ Collaboratif (in-process et out-of-process)
  - Extension d'un serveur Web (Apache, Microsoft IIS ou Netscape NetServer) meilleures performances pour le service des pages HTML

## Déploiement

Installation de Tomcat

- ✓ Récupérer l'archive sur le site
- ✓ Décompresser l'archive
- ✓ Ca marche! Occupation disque ~10 Mo

Contenu de l'archive

- ✓ 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 web-archives (war)
- ✓ Webapps/ROOT Répertoire de dépôt des .jsp indépendantes
- ✓ Webapps/ROOT/WEB-INF/classes Répertoire de dépôt des servlets indépendantes

## Déploiement

Correspondance URI → répertoire Par défaut

Copié dans le répertoire ⇒ déployé (= décompacté et installé)

## Complément : les formulaires

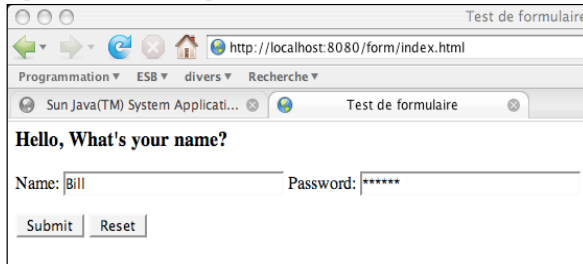
Définition d'un formulaire : balise **<form>**

- ✓ Attribut **method** :
  - **get** : méthode HTTP GET, les paramètres sont passés via l'URI
  - **post** : méthode HTTP POST, la page HTML avec le contenu des paramètres est envoyée au serveur
- ✓ Attribut **action** :
  - URI du composant Web qui va recevoir le formulaire
- ✓ Peut contenir n'importe quelle sous balise HTML valide
- ✓ Sous-balise **<input>** : définition d'un paramètre
  - Attribut **type**
    - Définit le type du paramètre (text, password, reset, submit...)
  - Attribut **name**
    - Définit le nom du paramètre
  - Attributs optionnels (**size**...)

## Complément : les formulaires

### Définition d'un formulaire en HTML

```
<html> <body>
<h3>Hello, What's your name?</h3>
<form method="post" action="url-servlet">
  Name: <input type="text" name="username" size="25">
  Password: <input type="password" name="password" size="25">
<p></p>
<input type="submit"
value="Submit">
<input type="reset"
value="Reset">
</form>
</body> </html>
```



## Complément : les formulaires

### Récupération des données d'un formulaire dans une servlet

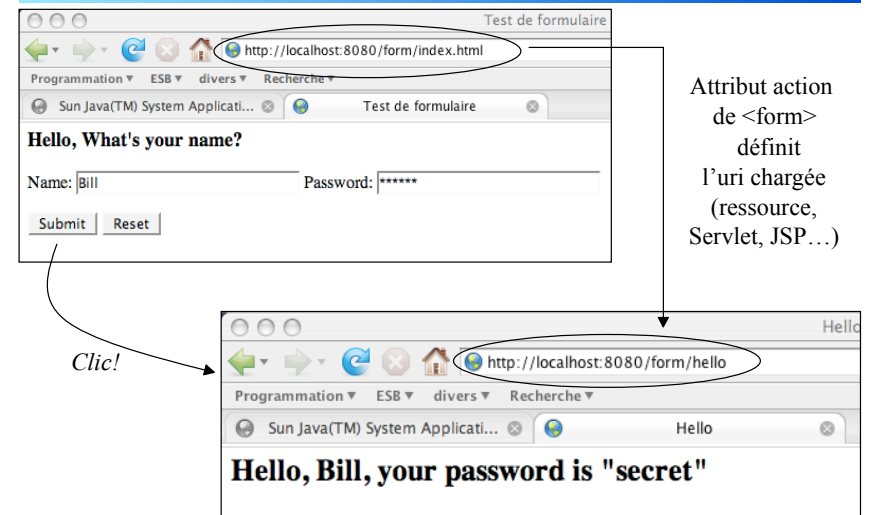
```
public class FormulaireServlet extends HttpServlet {
  public void doPost( HttpServletRequest request, HttpServletResponse response )
    throws ServletException, IOException {
    response.setContentType( "text/html" );
    PrintWriter out = response.getWriter();
    out.println( "<html><body>" );
    String name = request.getParameter("username");
    String password = request.getParameter("password");
    out.println( "<h2>Hello, " + name + ", your password is \"\"
      + password + "\"</body></html>");
  }
}
```

## Complément : les formulaires

### Récupération des données d'un formulaire dans une JSP

```
<html><body>
<h2>Hello,
  <%= request.getParameter("username") %>,
  your password is "
  <%= request.getParameter("password") %>
  "
</h2>
</body></html>
```

## Complément : les formulaires



## Conclusion

---

Servlet et Java Server Pages :

Étendent le comportement des serveurs Web avec des programmes Java

Résumé des fonctionnalités

- ✓ JAVA embarqué dans HTML ou HTML embarqué dans JAVA
- ✓ Portabilité, facilité d'écriture (Java)
- ✓ Notion de session au dessus d'HTTP
- ✓ Persistance des données entre deux appels  
(Pas de persistance si serveur redémarre)
- ✓ JSP chargée et instanciée une seule fois
- ✓ JSP exécutée dans des processus légers (threads)
- ⇒ Attention aux accès concurrents!