

Environnement Client/Serveur

Cours 7

JavaServer Pages (1)

kn@lri.fr



Comprendre le monde,
construire l'avenir

université
PARIS-SACLAY



1 Rappels sur les systèmes d'exploitations / Communication par mémoire partagée ✓

2 Réseaux généralités, IP, UDP ✓

3 Multi-thread en Java, TCP ✓

4 Broadcast UDP et étude de cas ✓

5 Rappels Java/Java 8/Sérialisation ✓

6 Appel de méthode à distance/Rappels JDBC ✓

7 JavaServer Pages (1)

7.1 Principe

7.2 Rappels HTTP

7.3 Le serveur web Tomcat

Programmation Web coté serveur



(rappel) génération de pages-web dynamiques (*i.e.* dont le contenu est calculé en fonction de la requête HTTP). Plusieurs choix de langage côté serveur.

- ◆ PHP (déploiement de site très simple, langage ~~merdique~~ particulier)
- ◆ Python, Ruby (manque de standardisation, plusieurs framework concurrents, problèmes de performances)
- ◆ ASP .NET (microsoft)
- ◆ **Java/JSP** (langage raisonnable, déploiement complexe)



JSP (JavaServer Pages) est un *framework* permettant de créer des pages Web dynamiques en Java. Il fait partie de la suite Java EE (Entreprise Edition). Rappel :

- ◆ Java Card (Java pour cartes de crédit, très peu de choses, pas de GC)
- ◆ Java ME (Micro Edition, pour les périphériques embarqués, mobiles, etc.)
- ◆ Java SE (Standard Edition, java « normal »)
- ◆ Java EE (Entreprise Edition, SE + packages pour JSP, et autres)



Nécessite un serveur Web particulier. Le standard est Apache Tomcat.

- ◆ Le programmeur écrit des fichiers `.jsp`, contenant du HTML + du java dans des balises spéciales
- ◆ (Le programmeur déploie les fichiers sur le serveur Tomcat)
- ◆ L'utilisateur navigue vers une page `foo.jsp`
- ◆ Le serveur Tomcat génère `fooServlet.class`
- ◆ La classe est chargée dans la JVM java et (sa méthode principale) est exécutée, produisant une page HTML
- ◆ La page HTML est envoyée au navigateur

Example



```
<%@ page contentType="text/html; charset=UTF-8" %>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <title>test JSP</title>
```

```
    <meta charset="UTF-8" />
```

```
  </head>
```

```
  <body>
```

```
    Page generated on
```

```
    <%
```

```
    java.util.Date d = new java.util.Date();
```

```
    out.println(d.toString());
```

```
    %>
```

```
  </body>
```

```
</html>
```

balises spéciales JSP



JSP introduit 4 balises spéciales qui sont interprétée par le serveur Tomcat.

Balise de configuration

`<%@ ... %>` (options HTML, import de packages, ...)

Balise de déclarations

`<%! ... %>` (déclarer des attributs et des méthodes)

Balises d'instructions

`<% ... %>` (permet de mettre une suite d'instructions)

Balises d'expressions

`<%= ... %>` (permet de mettre une expression dont le résultat est converti en `String`)

Exemple complet



```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="java.util.Date" %>
<!DOCTYPE html>
<html>
  <head><title>test JSP</title>
<meta charset="UTF-8" />
  </head>
  <%!
Date maDate;

Date maMethode() {
  return new Date();
}
  %>
  <body>
<%
maDate = maMethode();
%>
    Page created on <%= maDate %>
  </body>
</html>
```



Le code d'une page JSP est transformé en classe Java et compilé par le serveur Web (tomcat). La classe générée hérite de *HttpServlet*.

- ◆ On **ne doit pas écrire** de constructeur dans une page JSP, ni déclarer la classe courante avec `class Toto { ...`. Le compilateur JSP le fait automatiquement.
- ◆ Il **est inutile** de mettre une méthode `public static void main` (elle ne sera jamais appelée).
- ◆ Il **n'est pas recommandé** de déclarer des variables `static` dans un bloc de déclaration d'une page JSP. Par définition, une même page peut être appelée par plusieurs clients (navigateurs) différents de manière concurrent, l'accès aux ressources partagées doit être bien contrôlé.



1 Rappels sur les systèmes d'exploitations / Communication par mémoire partagée ✓

2 Réseaux généralités, IP, UDP ✓

3 Multi-thread en Java, TCP ✓

4 Broadcast UDP et étude de cas ✓

5 Rappels Java/Java 8/Sérialisation ✓

6 Appel de méthode à distance/Rappels JDBC ✓

7 JavaServer Pages (1)

7.1 Principe ✓

7.2 Rappels HTTP

7.3 Le serveur web Tomcat

Caractéristiques du protocole HTTP



- ◆ Sans connexion permanente:
 - ◆ Le client se connecte au serveur, envoie sa requête, se déconnecte
 - ◆ Le serveur se connecte au client, envoie sa réponse, se déconnecte
- ◆ Indépendant du contenu : permet d'envoyer des documents (hyper) texte, du son, des images, ...
- ◆ Sans *état*: chaque paire requête/réponse est indépendante (le serveur ne maintient pas d'information sur le client entre les requêtes)
- ◆ Protocole en mode *texte*

Format des messages HTTP



Les messages ont la forme suivante

- ◆ Ligne initiale CR LF
- ◆ zéro ou plusieurs lignes d'option CR LF
- ◆ CR LF
- ◆ Corp du message (document envoyé, paramètres de la requête, ...)

- ◆ *Requête* la première ligne contient un nom de *méthode* (GET, POST, HEAD, ...), le paramètre de la méthode et la version du protocole
- ◆ *Réponse* la version du protocole, le code de la réponse (200, 404, 403, ...) et un message informatif

Formulaire HTML (version simple)



L'élément `<form>` permet de créer des formulaires HTML. Un formulaire est constitué d'un ensemble de widgets (zones de saisies de textes, boutons, listes déroulantes, cases à cocher, ...) et d'un bouton submit. Lorsque l'utilisateur appuie sur le bouton, les données du formulaires sont envoyées au serveur. Exemple, fichier `age.html` :

```
<html>
  <body>
<form method="get" action="display.jsp" >
  Entrez votre âge  :
      <input type="text" name="val_age" />
<input type="submit" />
  </form>
</body>
</html>
```

The screenshot shows a simple web form rendered in a browser. It consists of a rectangular container with a thin border. Inside the container, on the left, is a text input field with a light gray border and a small cursor. To the right of the input field is a button with a light gray border and the text "Submit Query" centered on it.



Les paramètres envoyés au serveur Web par la méthode `get`, sont accessibles en Java l'objet implicite *request*. Exemple : `display.jsp`

```
<html>
<body>
  Vous avez <%= request.getParameter("val_age") %> ans!
</body>
</html>
```

Différence entre get et post



La seule différence entre get et post est la manière dont les paramètres sont passés :

get :

les paramètres sont encodés dans l'url : `display.jsp?val_age=22`

post :

les paramètres sont stockés dans la requête HTTP

Il est donc judicieux d'utiliser post lorsque les paramètres sont longs ou contiennent des caractères ne pouvant pas être présents dans une URL (upload de fichier par exemple).



Rappel : une session HTTP est un ensemble de connexions et d'échange de requêtes HTTP entre le serveur et un même client. C'est le serveur qui décide de la durée d'une session et qui gère le mécanisme de reconnaissance du client (usuellement grâce à des cookies).

Objets implicites



Le code placé dans les balises spéciales a accès à certains objets automatiquement déclarés (*JSP implicit objects*). Parmi les principaux :

`JspWriter out` : comme `System.out` mais écrit dans la page Web générée.

`HttpSession session` : représente la session HTTP courante et contient les variables de session.

`ServletContext application` : représente le **contexte d'exécution** de l'application JSP et permet de définir des variables d'application.

`HttpServletRequest request` : représente la requête HTTP ayant provoqué le chargement de la page en cours. L'objet permet de récupérer les paramètres passés dans la requête HTTP (par exemple les valeurs d'un formulaire).

`HttpServletResponse response` : correspond à l'objet qui sera envoyé au client permet de spécialiser la réponse envoyée au client (en-têtes HTTP, cookies, ...).

Classe JspWriter



Fonctionne comme `System.out` (on peut donc appeler `.print/.println`) mais correspond a un endroit particulier de la page HTML

Classe HttpSession



Propose plusieurs méthodes :

```
//Renvoie la valeur stockée sous le nom name  
Object getAttribute(String name)
```

```
//Stocke l'objet value sous le nom name  
void setAttribute(String name, Object value)
```

```
//Supprime l'association name value  
void removeAttribute(String name)
```

```
//Définit la durée (en secondes) d'inactivité d'une session  
void setMaxInactiveInterval(int interval)
```

```
//Renvoie la durée (en secondes) d'inactivité d'une session  
int getMaxInactiveInterval()
```

```
//Renvoie la date (en mili-secondes depuis EPOCH) de dernière utilisation  
long getLastAccessedTime()
```

```
//Détruit la session en cours  
void invalidate()
```

Classe ServletContext



Propose plusieurs méthodes :

```
//Renvoie le chemin sur le système de fichier du serveur  
//correspondant à la portion d'URL donnée  
String getRealPath(String path)
```

```
//Stocke l'objet donné dans l'espace global de l'application. Toutes  
//les pages de l'application et toutes les sessions y ont accès :  
void setAttribute(String name, Object o)
```

```
//Récupère l'objet précédemment stocké ou null si aucun objet  
//n'existe sous ce nom  
Object getAttribute(String name)
```

```
//Supprime l'objet stocké sous ce nom s'il en existe un  
void removeAttribute(String name)
```

Classe HttpServletRequest



Propose plusieurs méthodes :

```
//Récupère la valeur des cookies:
```

```
Cookie[] getCookies()
```

```
//Récupère les paramètres passés par un formulaire :
```

```
Map<String, String[]> getParameterMap()
```

```
//Récupère un paramètre particulier
```

```
String[] getParameter(String name)
```

```
//Renvoie le type de la requête (POST/GET/PUT)
```

```
String getMethod()
```

Classe HttpServletResponse



Propose plusieurs méthodes :

```
//Renvoie une erreur HTTP (404 par exemple)
```

```
void sendError(int code)
```

```
//Ajoute un cookie au site
```

```
void addCookie(Cookie c)
```

```
//Effectue une redirection temporaire
```

```
void sendRedirect(String url)
```

Classe Cookie



Propose plusieurs méthodes :

```
//Constructeur
```

```
Cookie(String name, String value)
```

```
//Expiration en secondes
```

```
void setMaxAge(int a)
```



Coté client

Première connexion à une page PHP

Deuxième accès une page PHP (envoi du cookie ("JSESSIONID", "12345"))

Coté serveur (JSP)

Démarrage de session

- génération d'un ID unique "12345"
- dépôt d'un cookie "JSESSIONID", valeur "12345", durée 10 minutes - création dans une hashmap globale d'une entrée:

```
"12345" → new HttpSession();
```

```
"12345" → objet session existant
```



1 Rappels sur les systèmes d'exploitations / Communication par mémoire partagée ✓

2 Réseaux généralités, IP, UDP ✓

3 Multi-thread en Java, TCP ✓

4 Broadcast UDP et étude de cas ✓

5 Rappels Java/Java 8/Sérialisation ✓

6 Appel de méthode à distance/Rappels JDBC ✓

7 JavaServer Pages (1)

7.1 Principe ✓

7.2 Rappels HTTP ✓

7.3 Le serveur web Tomcat

Architecture d'une application JSP



Une *application JSP* est l'ensemble des classes et ressources (pages JSP, pages HTML, scripts javascripts, images, ...) se trouvant sous un même répertoire.

Le serveur Web tomcat utilise une JVM par application. Quelle sont les conséquences pour deux applications (tomcat) qui souhaitent communiquer ? Elles doivent utiliser un système de communication sans mémoire partagée : fichiers partagés, connexion réseau, ou requête POST/GET en HTTP.

Pour faciliter les tests et déploiements, on configurera un serveur tomcat de test *dans l'éditeur Eclipse*.

Structure d'un projet Eclipse/JSP



Un projet JSP (sous Eclipse) ressemble à un projet Java classique. Il a la structure suivante :

`src/` :

répertoire contenant les sources Java

`build/` :

répertoire contenant les `.class` générés

WebContent/ :

racine de votre application Web. Sous Eclipse, le fichier `WebContent/index.jsp` sera présenté lors de l'exécution à l'URL : `http://localhost:8080/PROJET/index.jsp` (où PROJET est le nom du projet Eclipse).

WebContent/META-INF/ :

Répertoire contenant des fichiers utilisés pour la génération de l'archive et le déploiement. *Ne pas Modifier.*

WebContent/WEB-INF/ :

Répertoire contenant des fichiers (notamment `.jar`) utilisés par votre site.

Déploiement d'un projet JSP



1. créer un répertoire avec la structure suivante :

```
fichier1.jsp
...
fichiern.jsp
META-INF/MANIFEST.MF
WEB-INF/classes/...
```

Le répertoire WEB-INF/classes/ contient tous les .class générés.

2. Créer un .war de la manière suivante (en se plaçant dans le répertoire ci-dessus):

```
jar -cfv projet.war *
```

3. Arrêter le server Tomcat. Copier le fichier projet.war dans le sous-répertoire webapps/ présent à la racine du serveur
4. Redémarrer le serveur Tomcat. Les fichiers .war sont redéployés
5. Le projet est accessible sur `http://serveur.com/projet/`

En TP, toutes ces étapes sont prises en charge par Eclipse.