Cours 9 – Les servlets Java

Programmation objets, web et mobiles en Java
Licence 3 Professionnelle - Multimédia

Pierre Talbot (talbot@ircam.fr)

UPMC/IRCAM

6 mars 2018

Le menu

- Introduction
- Servlets
- Java Server Pages (JSP)
- Mapping URL vers Servlets
- Session
- Contexte de servlet
- Et ensuite...

Serveur web

- Un explorateur internet permet à l'utilisateur de demander une ressource.
- Cette ressource est servie par un serveur web.
- L'URL est l'adresse de cette ressource.
- Une ressource peut être une page HTML, une image, etc.

Absence de ressource

Si la ressource n'est pas là, une erreur 404 est générée par le serveur.

Protocole HTTP

Le client et le serveur communique via le protocole HTTP.

- Requête GET : Récupérer de l'information sans changer l'état visible du serveur.
- Requête POST : Demande de modification des informations côté serveur.
- ...

Protocole HTTP

Le client et le serveur communique via le protocole HTTP.

- Requête GET : Récupérer de l'information sans changer l'état visible du serveur.
- Requête POST : Demande de modification des informations côté serveur.
- ...

Codes de statut

- 1xx Information
- 2xx Succès : 200 pour OK, 201 pour "Created" (une ressource a bien été créée), etc.
- 3xx Redirection
- 4xx Erreur client : 400 si mauvaise requête, 404 si ressource inexistante, etc.
- 5xx Erreur serveur : 500 si erreur interne au serveur, etc.

GET vs POST

- Les données transportées par un GET sont limitées.
- Une requête POST a un corps pour les données.
- Les informations d'un GET sont transportées par l'URL sous la forme `example.fr/blah?arg=12&arg2=bonjour`.
- Attention à l'aspect sécurité d'une requête GET.
- Néanmoins si besoin de transporter un mot de passe, il faut utiliser un chiffrement (https) car le corps du POST est en clair.

Sémantique des requêtes

- GET = Récupérer de l'information.
- POST = Mettre à jour une information.

Requêtes idempotentes

- Idempotent en math : $f(x) = f(f(x))$
- Dans le web, ça signifie que deux mêmes requêtes d'affilées donne le même résultat qu'une seule requête.

Scénario

- L'utilisateur paie son panier.
- Il y a un petit lag et il décide de recharger la page : la requête est donc ré-envoyée.
- Si la requête n'est pas idempotente, alors son compte sera débité deux fois.

Requêtes idempotentes II

Dans la norme HTTP, on considère que :

- GET est idempotent.
- POST n'est pas idempotent.

C'est notamment pour ça que les navigateurs demandent si on est sûr de vouloir ré-envoyer la requête (car ce n'est pas idempotent).

Requête POST

C'est au développeur de rendre les requêtes POST idempotentes.

Pages statiques et dynamiques

Limites d'un serveur web

- Peut seulement servir des pages statiques.
- Ne peut pas sauvegarder de données dans une base de données ou fichiers.

- CGI (Common Gateway Interface) spécifie comment le serveur web et un programme externe communiquent.
- Est coûteuse car on lance le programme à chaque requête...

Servlets

En Java, au lieu de créer des processus systèmes, assez lourd, ce sont des *threads* qui sont utilisés.

- Un seul programme Java est lancé dans la JVM, le *servlet container*.
- Quand une requête arrive, le serveur exécute la requête dans un nouveau *thread*.
- Le container dirige la requête vers un servlet qui se charge de traiter l'information utilisateur et de renvoyer une page HTML.

Le menu

- Introduction
- Servlets
  - Configuration
  - Premier Servlet
- Java Server Pages (JSP)
- Mapping URL vers Servlets
- Session
- Contexte de servlet

Serveur de servlets

- On utilisera *Tomcat*, archive zip à télécharger `https://tomcat.apache.org/download-80.cgi`
- Voir les deux slides suivantes pour la configuration sous Eclipse ou Netbeans.

Configuration sous Eclipse

Ajouter le serveur Tomcat

1. Window > Preferences > Server > Runtime Environnement > Add
2. Sélectionner "Apache Tomcat v8.0" et ajouter le chemin du serveur téléchargé précédemment.

Projet Web avec Maven

1. File > New > Other > Maven > Maven Project > Next > Next
2. Sélectionner l'archétype *group id : org.apache.maven.archetypes* et *artifact id : maven-archetype-webapp*.
3. Finir la création du projet Maven en ajoutant le group id (par exemple *upmc*) et l'artifact qui est le nom du projet.

Configuration sous Netbeans

Ajouter le serveur Tomcat

1. Dans les onglets à gauche, sélectionner *Services*.
2. Servers > Add Server > Apache Tomcat
3. Dans *Select location*, préciser le chemin du serveur téléchargé précédemment.

Projet Web avec Maven

1. File > New Project > Maven > Web Application
2. Finir la création du projet Maven en ajoutant le group id (par exemple *upmc*) et l'artifact qui est le nom du projet.

Télégraphe

- Nous allons développez une application web de chat en ligne nommée Télégraphe en utilisant les servlets.
- On développe cette application incrémentalement afin d'aborder les différents principes des servlets :
  1. V1 : Une page de login où l'utilisateur rentre un pseudo, l'application répond avec une page contenant ce pseudo.
  2. V2 : Utilisation des JSPs pour décrire la page avec le login utilisateur.
  3. V3 : Création d'une nouvelle servlet pour envoyer des messages.
  4. V4 : Garder le pseudo dans une session.
  5. V5 : Utilisation d'un objet pour stocker les messages d'un salon de discussion.

Dissection d'un Servlet

```
public class WebChat extends HttpServlet
{
  protected void doGet(
    HttpServletRequest request,
    HttpServletResponse response)
   throws ServletException, IOException
  {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
      out.println("<!DOCTYPE html>");
      out.println("<html>");
      out.println("<head><title>Servlet Chat</title></head>");
      out.println("<body>");
      out.println("<h1>Hello! Servlet Chat at " +
        request.getContextPath() + "</h1>");
      out.println("</body></html>");
    }
  }
}
```

Récupérer les paramètres d'une requête

La requête HTML :

```
<form action="sendMessage.do">
  <textarea rows="4" cols="50" name="msg">
    Enter your message here...
  </textarea>
  <input type="submit">
</form>
```

Traitement côté serveur :

```
protected void doPost(HttpServletRequest request,
 HttpServletResponse response)
{
  String msg = (String) request.getParameter("msg");
}
```

Note : `getParameterValues()` permet de récupérer les paramètres à plusieurs valeurs (pensez aux *checkbox*).

Points clés

- On a deux méthodes `doGet` et `doPost` selon que l'utilisateur envoie une requête GET ou POST.
- On écrit la page HTML de retour dans l'objet `response.getWriter()`, comme quand on print dans la console avec `System.out.println`.
- `request` contient les arguments de la requête utilisateur, par exemple si on a un formulaire avec un champ `id="toto"` on récupère sa valeur avec :

```
request.getParameter("toto")
```

  Si l'objet retourné est `null` alors l'utilisateur n'a pas rempli ce champ.

Un premier Servlet

Une page de login où l'utilisateur rentre un pseudo, l'application répond avec une page contenant ce pseudo

- Télécharger sur le site du cours l'archive *TelegrapheV1Template.zip* et exécutez le code.
- Prendre en compte le cas où l'utilisateur a entré un pseudo vide, dans ce cas ré-afficher la page de login.
- En cas de succès, afficher "Super login JFdu67" si l'utilisateur a entré "JFdu67" comme login.

Consulter l'adresse `http://localhost:8080/Telegraphe`, *Telegraphe* étant le nom de la ressource.

Cycle de vie d'un Servlet

1. Une requête HTTP arrive sur le container.
2. Grâce à l'URL, le container constate que c'est pour un Servlet (ou non).
3. Le container crée les objets `HttpServletResponse` et `HttpServletRequest` qui encapsule la requête et la réponse HTTP.
4. Le container appelle la méthode `service()` sur le Servlet dans un nouveau *thread*.
5. Quand la méthode `service()` se termine, le container envoie la réponse et nettoie les deux objets créés à l'étape 3.

Gestion des requêtes dans le Servlet

- La méthode `service()` est une méthode de `HttpServlet` et appelle `doGet` ou `doPost` en fonction de la requête HTTP.
- Ces méthodes sont surchargées dans le Servlet de l'utilisateur.

```
protected void service(
  HttpServletRequest req,
  HttpServletResponse resp)
 throws ServletException, IOException
{
  String method = req.getMethod();
  if (method.equals(METHOD_GET)) {
    doGet(req, resp);
  } else if (method.equals(METHOD_POST)) {
    doPost(req, resp);
  } else if (. . .)
  . . .
  }else {
    resp.sendError(HttpServletResponse.SC_NOT_IMPLEMENTED, errMsg);
  }
}
```

Une seule instance du Servlet

Un servlet n'est instancié qu'une fois par le container, la même instance est ré-utilisée pour toutes les requêtes.

Règles d'utilisation

- Pas de variable d'instance dans un Servlet (sauf si on veut que les requêtes partagent les mêmes données, néanmoins attention à la persistance).
- Ne pas mettre les méthodes `doPost` ou `doGet` en `synchronized` (sinon on perd l'avantage du parallélisme) ; la section critique doit être aussi petite que possible.

Le menu

- Introduction
- Servlets
- Java Server Pages (JSP)
- Mapping URL vers Servlets
- Session
- Contexte de servlet
- Et ensuite...

Java Server Pages (JSP)

- Utiliser des `out.println(" <html>")` n'est pas très élégant et adapté...
- Au lieu de mettre du HTML dans du Java, pourquoi ne pas mettre du Java dans du HTML ?
- Ce qui a donné naissance aux JSP.

JSP

- Une JSP est contenue dans un fichier .jsp
- Elle est compilée par le serveur vers un Servlet utilisant les `out.println(...)`.

Exemple de JSP

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<body>
<% if(request.getParameter("msg") == null ||
      request.getParameter("msg").isEmpty()) { %>
  Vous devez envoyer un message.
<% } else { %>
  Le message est <%= request.getParameter("msg") %>
<% } %>
</body>
</html>
```

C'est le code Java qu'on doit échapper maintenant. Ce qui est entre <% et %> s'appelle un *scriptlet*.

Variables implicitement accessibles

Dans les *scriptlets*, on a accès à un certain de variables pré-déclarées :

- `JspWriter out` : Permet d'appeler `println` dans un *scriptlet*.
- `HttpServletRequest request` : La requête HTTP.
- `HttpServletResponse response` : La réponse HTTP.
- `HttpSession session` : La session courante.
- ...

Balises JSP

- **Scriptlet** <% `int count = 0;` %> : N'importe quel code Java peut être entre ces balises.
- **Expression** <%= `count++` %> : Équivalent à `out.println(count++);`, noter l'absence de `;`.
- **Directive de page** <%@ `page import java.util.*;` %> : Ici un import nécessaire si on utilise `ArrayList` dans un scriptlet par exemple.
- **Directive de page** <%@ `page contentType="text/html"` %>
- ...

Traduction du JSP vers un Servlet

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

public class PrintMsg_jsp extends HttpJspBase {
  // Tout ce qui est entre <%! %> vient ici.

  public void _jspService(HttpServletRequest request,
    HttpServletResponse response)
   throws ServletException, IOException
  {
    JspWriter out = ...;
    HttpSession session = ...;

    out.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Strict//EN\"");
    ...
    if(request.getParameter("msg") == null ||
      request.getParameter("msg").isEmpty()) {
      out.println("Vous devez envoyer un message.");
    } else {
      out.println("Le message est " + request.getParameter("msg"));
    }
    ...
    out.println("</html>");
  }
}
```

Télégraphe V2

Utilisation des JSPs pour décrire la page avec le login utilisateur.

- Création d'une JSP "room.jsp" (ça deviendra notre salon de discussion plus tard) dans le répertoire `Web pages`.
- Afficher le pseudo de l'utilisateur en utilisant les scriplets.
- Dans `Login.java`, vous pouvez utiliser `request.getRequestDispatcher("room.jsp")` comme pour afficher une page HTML.

Le menu

- Introduction
- Servlets
- Java Server Pages (JSP)
- Mapping URL vers Servlets
- Session
- Contexte de servlet
- Et ensuite...

Dispatch des requêtes utilisateurs

- Lorsque le container reçoit une requête utilisateur, il faut bien qu'il sache vers quel servlet la distribuer.
- On lui donne l'information dans le fichier `WEB-INF/web.xml`.

`web.xml`

```
<servlet>
    <servlet-name>Log</servlet-name>
    <servlet-class>upmc.telegraphe.Login</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Log</servlet-name>
    <url-pattern>/login.do</url-pattern>
</servlet-mapping>
```

`servlet-name` est un nom interne du service, `servlet-class` est le chemin complet vers la classe et `url-pattern` est l'URL tel que décrite par exemple dans `<form action="login.do">`.

Télégraphe V3

Création d'une nouvelle servlet pour envoyer des messages.

- Dans la JSP `room.jsp`, ajouter un formulaire qui récupère un message de l'utilisateur et envoie une requête POST vers `sendMessage.do`.
- Créer une nouvelle servlet `Room.java` qui ne fait que rediriger vers la JSP `room.jsp`.
- Créer le mapping dans `web.xml`.
- Mettre à jour la JSP `room.jsp` pour afficher le dernier message envoyé.
- PS : on ne se tracasse pas du pseudo pour le moment.

Le menu

- Introduction
- Servlets
- Java Server Pages (JSP)
- Mapping URL vers Servlets
- Session
- Contexte de servlet
- Et ensuite...

Session

- Problème : Après avoir envoyer un message, le serveur a “oublié” le pseudo de l'utilisateur : il n'y pas retransmis dans les futures requêtes.
- Le protocole HTTP est *stateless*, après une requête/réponse, le serveur a oublié le client.

Session

- Le serveur peut créer un cookie qui contient un identifiant unique.
- Ce cookie est envoyé dans la première réponse.
- Le client doit ré-envoyer ce cookie à chaque fois.

Créer et récupérer une session

```
protected void doPost(HttpServletRequest request,
 HttpServletResponse response)
{
  HttpSession session = request.getSession();
}
```

- La méthode `getSession()` crée la session si elle n'existait pas déjà et la retourne.
- L'ajout du cookie dans la réponse et le traitement de la requête est géré par le container.
- `getSession(false)` retourne `null` si il n'y a pas de session courante (et n'en crée pas).

Utiliser une session

- `String pseudo = (String) s.getAttribute("pseudo")`
- `s.setAttribute("pseudo", pseudo)`
- ...

Attention : Ne pas confondre `getParameter` et `getAttribute` où le premier contient les données de la requête courante et l'autre les données de session.

Télégraphe V4

Garder le pseudo dans une session.

- Lors de la première connexion, on enregistre le pseudo dans la session avec `session.setAttribute(...)`.
- Utilisez cette information dans `room.jsp` pour récupérer et afficher le pseudo.

Le menu

- Introduction
- Servlets
- Java Server Pages (JSP)
- Mapping URL vers Servlets
- Session
- Contexte de servlet
- Et ensuite...

Contexte d'exécution

On veut attaquer la dernière version de notre chat. Comment faire pour garder en mémoire les messages précédents ?

- Les paramètres de `request` ne sont valides que pour la requête courante.
- Les attributs de `session` ne sont valides que pour un seul utilisateur.

Information globale

- Il n'existe qu'une seule instance de servlet et donc toutes les requêtes partagent le même servlet.
- Dès lors, peut-on stocker les messages dans une variable d'instances du servlet ?
- Oui, cela marchera mais il y a un endroit dédié aux informations de servlet appelé `ServletContext`.
- L'avantage est que, sous certaines conditions, lorsque vous redéployer votre application, le contexte n'est pas perdu tandis que les variables d'instances le sont.

ServletContext

- On peut stocker des attributs dans le `ServletContext` d'une application web.
- Par exemple

```
getServletContext().setAttribute("roomCounter", new Integer(0))
```

stock un objet dans l'attribut "roomCounter" qui peut-être récupéré avec `getAttribute`.

ATTENTION

- Plusieurs requêtes simultanées signifient que l'objet `ServletContext` peut être accédé de manière concurrente vu qu'il y a un thread par requête.
- Il faut **protéger** la modification des attributs avec des méthodes `synchronized`.

Télégraphe V5

Utilisation d'un objet pour stocker les messages d'un salon de discussion.

- Créer un objet `upmc.telegraphe.model.MessageRoom.java` qui contiendra une liste de message avec une fonction `push` (`synchronized` !).
- Dans la méthode `public void init() throws ServletException` du Servlet `Room`, enregistrer une nouvelle instance de l'objet MessageRoom.
- Afficher les messages dans la JSP ; vous pouvez communiquer l'objet `MessageRoom` à la JSP en passant par les attributs de `request`.

Le menu

- Introduction
- Servlets
- Java Server Pages (JSP)
- Mapping URL vers Servlets
- Session
- Contexte de servlet
- Et ensuite...

AJAX

AJAX

- Pour éviter que l'utilisateur ait à recharger la page pour récupérer les données, on peut utiliser *ajax*.
- Les requêtes et réponses ne seront plus en HTML mais en JSON par exemple.
- Attention à bien changer le MIME.

Pour faire un chat en ligne, une technique est d'interroger le serveur toutes les secondes si il y a des nouveaux messages.

Web Socket

Web socket

- Avec les sessions et même AJAX, le serveur ne peut pas prévenir le client qu'une information vient d'arriver.
- Une technique est de laisser ouvert une requête HTTP et ainsi garder la connexion entre le client et le serveur, c'était une sorte de *hack* appelé *Comet*.
- Maintenant le protocole *Web socket* est plus adapté. C'est un mode du protocole HTTP.

Ceci sont des techniques générales, pouvant fonctionner avec tous langages et frameworks.

À ne pas manquer

Compléter vos connaissances Java

- Le mécanisme de réflection et les annotations.
- Des frameworks web complet comme *Spring* ou *Struts*.
- ...

À ne pas manquer

Connaissances générales

- Apprendre un langage fonctionnel (par exemple OCaml et puis Haskell).
- Lire le livre de Peter Van Roy (`https://www.info.ucl.ac.be/~pvr/book`), disponible en français aussi.
- Se renseigner sur les langages synchrones (Lustre, Esterel, ...).
- Langages pour le web expérimentaux : Hop et HipHop.
- Consulter notamment `http://www.college-de-france.fr/site/gerard-berry/_course.htm`