

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

J2EE


Abdelahad SATOUR

Séances 5–8

partie 2

Un pas vers j2EE
Servlet

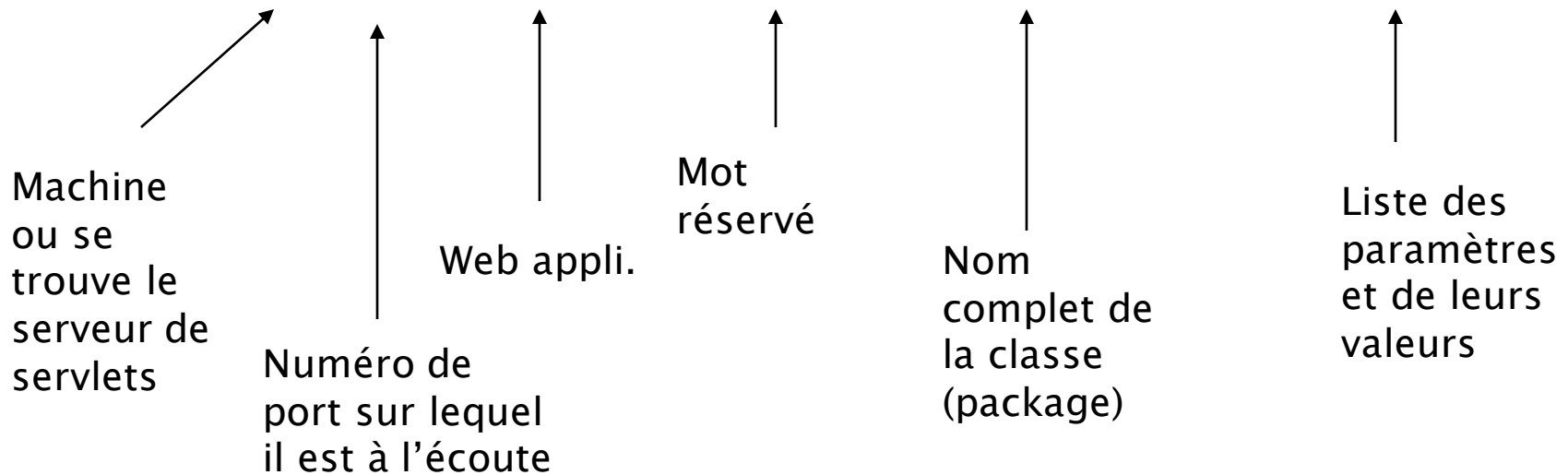
Plan

- ▶ URL Servlet
 - ▶ Réponse HTTP (redirect, error)
 - ▶ Init params
 - ▶ Cookies
 - ▶ Sessions
 - ▶ Filtres
- 

Charger et invoquer une Servlet

► URL

`http://host:port/<web-app>/servlet/<servlet_class>?param1=value1&...`



Réponse HTTP

L'objet **HttpServletResponse** permet en plus de renvoyer des codes d'erreurs

```
public class HttpRedirectServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
  
        if ("/index.html".equals(request.getPathInfo()))  
            response.sendRedirect("/demo/header/index.html");  
        else  
            response.sendError(HttpServletResponse.SC_NOT_FOUND);  
    }  
}
```

Redirection HTTP

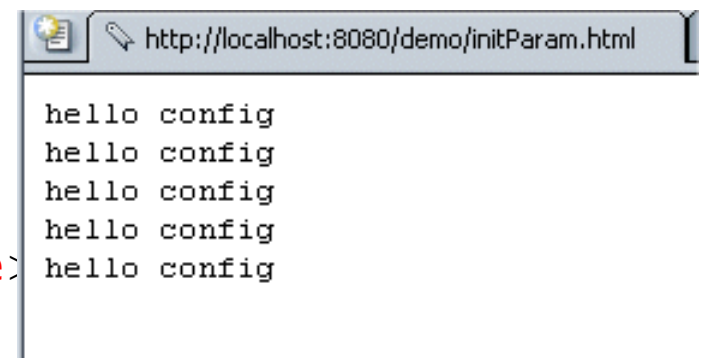
SC = Status Code

Etat HTTP 404 -	
type	Rapport d'état
message	
description	La ressource demandée () n'est pas disponible.
Apache Tomcat/4.1.18-LE-jdk14	

Paramètres d'initialisation

Les paramètres sont déclarés dans le fichier web.xml

```
<servlet>
  <servlet-name>initParam</servlet-name>
  <servlet-class>fr.umlv.servletdemo.InitParamServlet</servlet-class>
  <init-param>
    <param-name>count</param-name>
    <param-value>5</param-value>
  </init-param>
  <init-param>
    <param-name>message</param-name>
    <param-value>hello config</param-value>
  </init-param>
</servlet>
```



Paramètres d'initialisation (2)

L'objet ServletConfig permet de récupérer les paramètres

```
public class InitParamServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
  
        response.setContentType("text/plain");  
        PrintWriter out= response.getWriter();  
        for(int i=0;i<count;i++){  
            out.println(message);  
        }  
    }  
  
    public void init(ServletConfig config) throws ServletException {  
        count = Integer.parseInt(config.getInitParameter("count"));  
        message = config.getInitParameter("message");  
    }  
  
    private int count;  
    private String message;  
}
```

Demande la valeur du paramètre "count"

Paramètres d'initialisation (3)

Le destroy doit libérer les ressources !!

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    ...
}
```

Stockage des paramètres

```
public void init(ServletConfig config) throws ServletException {
    super.init(config);
```

```
    count = Integer.parseInt(config.getInitParameter("count"));
    message = config.getInitParameter("message");
}
```

```
public void destroy() {
    message=null;
}
```


Libération des paramètres

```
private int count;
private String message;
```



Gestion des cookies



Introduction

- ▶ Morceaux d'informations envoyés par le serveur
... et renvoyés par le client quand il revient visiter le même URL
 - ▶ Durée de vie réglable
 - ▶ Permet la persistance
- 

A quoi ça sert ?

- ▶ Identification des utilisateurs (e-commerce)
 - ▶ Eviter la saisie d'informations à répétition
 - login, password, adresse, téléphone...
 - ▶ Gérer des « préférences utilisateur »
 - sites portails...
 - ▶ ...
- 

Cookie et sécurité

- ▶ Jamais interprété ou exécuté : pas de virus
- ▶ Un cookie est limité à 4KB et les navigateurs se limitent à 300 cookies (20 par site) : pas de surcharge de disque
- ▶ Bien pour rendre privées des données non sensibles
 - nom, adresse, ... mais pas No CB !
- ▶ ... mais ne constitue pas un traitement sérieux de la sécurité

Manipuler les cookies

- ▶ Utiliser les fonctions de l'API des servlets...
 - créer un cookie : classe `Cookie`,
 - écrire/lire un cookie : `addCookie(cookie)`, `getCookies()`,
 - positionner des attributs d'un cookie : `cookie.setXxx(...)`

- ▶ Exemple d'envoi d'un cookie :

...

```
String nom = request.getParameter("nom");
```

```
Cookie unCookie = new Cookie("nom", nom);
```

...ici positionner des attributs si on le désire

```
response.addCookie(unCookie);
```

...

Création d'un cookie

- ▶ `Cookie unCookie = new Cookie(name, value);`
- 2 arguments de type `java.lang.String` :
 - `name` et `value`
- caractères non autorisés :
 - espace blanc
 - `[]() = , " / ? @ : ;`

Attributs des cookies

- ▶ `getValue/setValue`
- ▶ `getName/setName`
- ▶ `getComment/setComment`
- ▶ `getMaxAge/setMaxAge` : délai restant avant expiration du cookie (en seconde)
 - par défaut : pour la session courante
- ▶ `getPath/setPath` : répertoire où s'applique le cookie
 - dir. courant ou pages spécifiques

Récupération des cookies

► Exemple de récupération des cookies

```
Cookie [] cookies = request.getCookies();  
String nom = getCookieValue(cookies, "nom", "non trouvé");  
...  
public static String getCookieValue(Cookie [] cookies,  
                                     String cookieName, String defaultValue) {  
    for(int i=0; i < cookies.length; i++) {  
        Cookie cookie = cookies[i];  
        if(cookieName.equals(cookie.getName()))  
            return(cookie.getValue());  
    }  
    return(defaultValue);  
}
```


Temps d'expiration

- ▶ Par défaut, durée de vie d'un cookie = la connexion.
- ▶ Si on veut que le cookie soit sauvé sur disque, modifier sa durée de vie :

```
public static final int SECONDS_PER_YEAR =  
    60*60*24*365;  
cookie.setMaxAge(SECONDS_PER_YEAR) ;
```

TP Manipulation Cookies

- ▶ Lien 2

TP 30 minutes Cookies

- ▶ Authentication memorize function

Sessions

Problématique

- ▶ Protocole HTTP = protocole Internet déconnecté
 - différent de Telnet, Ftp, ...
 - traite les requêtes et les réponses comme transactions simples et isolées (requêtes non apparentées)
- ▶ Certaines applications Web (e-commerce : caddie) ont besoin de maintenir une "mémoire" entre deux requêtes
 - ie. maintenir une connexion de l'utilisateur sur le serveur
 - pour se faire : concept de "suivi de sessions"

Suivi de session : qu'est-ce que c'est ?

- ▶ Mémoire de ce que fait l'utilisateur d'une page à l'autre
 - consiste au transfert de données générées par une requête vers les requêtes suivantes
- ▶ 4 méthodes avec les servlets Java
 - 1) utilisation des cookies (déjà vu)
 - 2) réécriture d'URL : passage de paramètres
 - 3) utilisation des champs de formulaire "hidden"
 - 4) utilisation du JSDK (HttpSession API)

Réécriture d'URL

► Principe :

- ajouter dans la chaîne de requête de la servlet des informations supplémentaires identifiant la session

```
<a href="http://leo.inria.fr/servlet/foo?uid=itey">Acheter  
</a>
```

- l'ID utilisateur est transmis en même temps que la requête; il est accédé par chaque servlet mentionnée qui récupère les informations persistantes (BD, fichiers) à partir de cet ID

► Limitations :

- données volumineuses, caractères autorisés, longueur URL, données visibles (sécurité)

Champs de formulaires cachés

▶ Principe :

- on cache les données de session dans des champs "hidden" :

```
<INPUT TYPE="HIDDEN" NAME="uid" VALUE=itey>
```

▶ Limitations :

- idem la "réécriture d'URL" sauf pour la sécurité (utilisation de POST)

L'objet session

- ▶ Très simple avec l'API des servlets (JSDK)
 - objet `HttpSession`
- ▶ Principe :
 - Un objet "session" peut être associé *avec chaque requête*
 - Il va servir de "container" pour des informations persistantes
 - Durée de vie limitée et réglable

Session (2)

L'objet HttpSession gère les sessions

Création :

`request.getSession()`
`request.getSession(boolean create)`

Pourquoi?

Crée une session si
non existante

Gestion d'association clé/valeur :

`session.getAttributeNames()`
`session.getAttribute(String name)`
`session.setAttribute(String name, Object value)`
`session.removeAttribute(String name)`

Destruction :


`session.invalidate()`
`session.logout()`

Invalide toutes les sessions
pour un client

Modèle basique

```
HttpSession session = request.getSession(true);
Caddy caddy = (Caddy) session.getValue("caddy");

if(caddy != null) {
    // le caddy n'est pas vide !
    afficheLeContenuDuCaddy(caddy);
} else {
    caddy = new Caddy();
    ...
    caddy.ajouterUnAchat(request.getParameter("NoArticle"));
    session.putValue("caddy", caddy);
}....
```



Méthodes de la classe HttpSession

- ▶ `getID()`
- ▶ `isNew()`
- ▶ `getCreationTime()` / `getLastAccessedTime()`
- ▶ `getMaxInactiveInterval()`
- ▶ ...
- ▶ `getValue()` , `removeValue()` , `putValue()`
- ▶ ...
- ▶ `invalidate()`

TP manipulation

- ▶ Lien 2

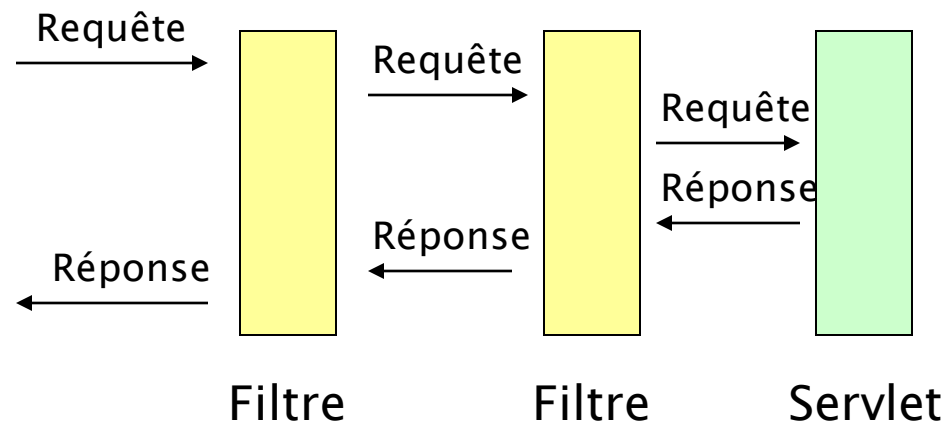
TP 30 minutes

- ▶ Auth avancée

Forward



Les filtres



Les filtres

Il est possible d'ajouter des filtres qui seront exécutés avant les servlets

```
<filter>  
  <filter-name>footer</filter-name>  
  <filter-class>fr.umlv.servletdemo.FooterFilter</filter-class>  
</filter>
```

```
<filter-mapping>  
  <filter-name>footer</filter-name>  
  <url-pattern>/filter/*</url-pattern>  
</filter-mapping>
```

Filtre à partir d'un URL

```
<filter-mapping>  
  <filter-name>footer</filter-name>  
  <servlet-name>hello</servlet-name>  
</filter-mapping>
```

Filtre à partir d'une servlet

Les filtres

Des wrappers permettent d'interagir avec la servlet

```
public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain) throws IOException, ServletException {

    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    out.println("<html><body bgcolor=\"white\"><h1>");

    HttpServletResponseWrapper newResponse=new HttpServletResponseWrapper(
        (HttpServletResponse)response) {
        public void setContentType(String contentType) {
        }
    };
    chain.doFilter(request,newResponse);

    // context.getRequestDispatcher("/footer.html").include(request,response);
    out.println("</h1></body></html>");
}
```

Exécuté avant

Appelle les autres filtres
ou la servlet

Exécuté après