Servlet

Laurent Réveillère


laurent.reveillere@u-bordeaux.fr


http://www.reveillere.fr/

université de BORDEAUX

Contexte

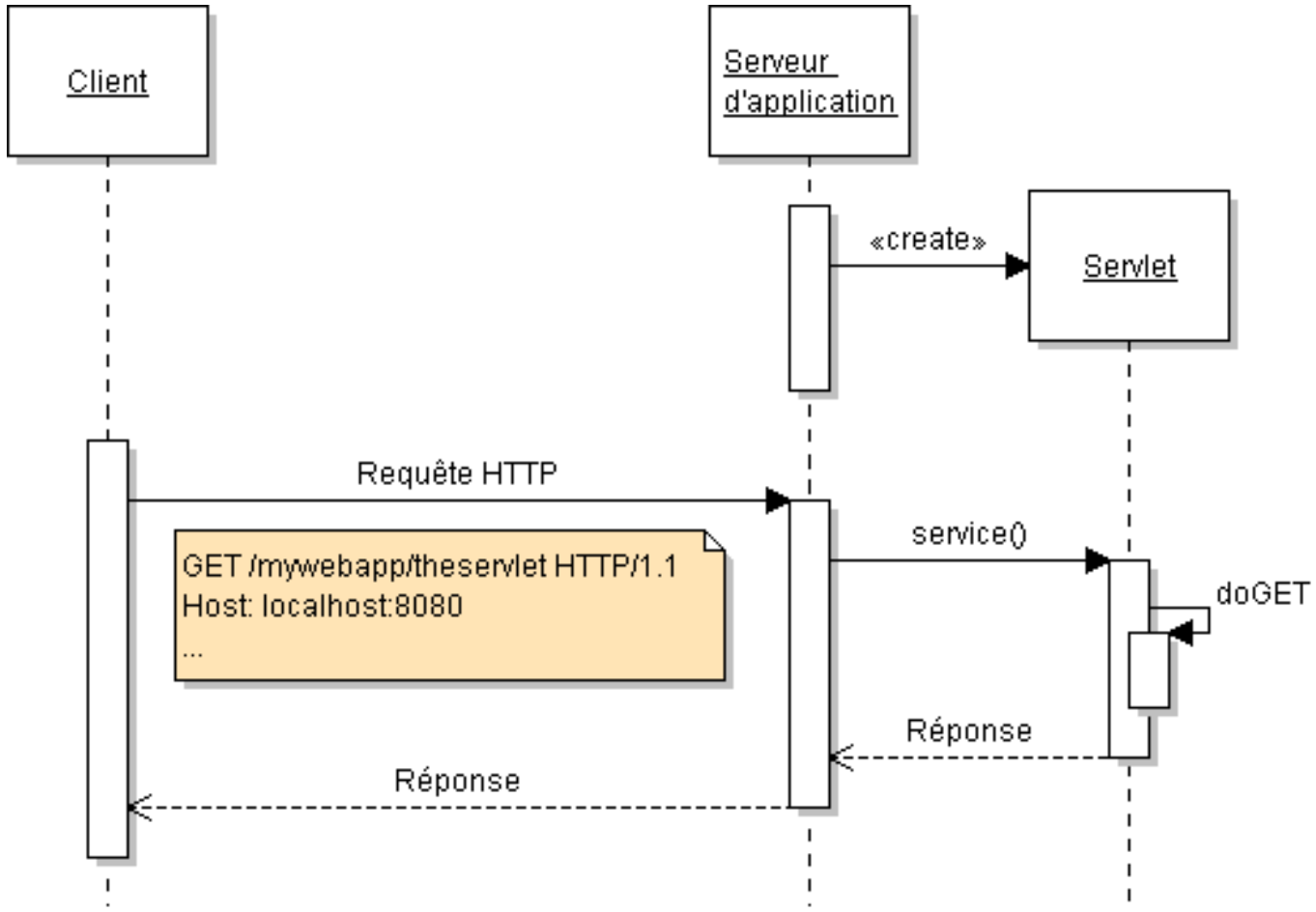
- Un serveur Web peut héberger plusieurs applications Web
- Les requêtes HTTP sont traitées par une application spécifique en fonction de l'URL de la requête
- Chaque application a son propre contexte qui est la partie de l'URL placée juste après le nom de la machine serveur :

```
http://www.machin.com/nomAppli/rep/index.html
```

Fonctionnalité de base

- De base un serveur Web ne fournit que des pages HTML statiques
- Un servlet est :
  - un composant logiciel écrit en Java
  - qui s'exécute en programme « compagnon » d'un serveur Web
- Il reçoit des requêtes de clients HTTP et construit des pages dynamiques écrites en HTML, qui sont renvoyées en réponse aux clients

Fonctionnement d'un servlet



Exemple de servlet

```
package fr.ub.m2gl.servlet;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

public class HelloServlet extends HttpServlet {
  protected void doGet(HttpServletRequest req, HttpServletResponse resp)
  throws ServletException, IOException {
    resp.setContentType("text/html");
    PrintWriter out = resp.getWriter();
    out.println("<html>");
    out.println("<body>");
    out.println("Hello World!");
    out.println("</body>");
    out.println("</html>");
  }
}
```

Conteneur de servlets

- Les serveurs HTTP ne savent pas exécuter directement le code Java d'un servlet
- Les servlets nécessitent un conteneur de servlets qui se charge de la gestion des servlets :
  - gestion des noms des servlets (associés à une classe Java)
  - création et initialisation des servlets
  - suppression des servlets
- Le serveur Web lui passe la main quand il reçoit d'un client une requête qui doit être traitée par un servlet (reconnaissable à l'URL)

URL d’un servlet

- Le client ne peut donner une adresse directe du servlet
- L’application Web doit établir une correspondance (un *mapping*) entre le servlet et un URL (avec une annotation ou le fichier web.xml)
- Demander cet URL passera la main au conteneur de servlets qui lancera l’exécution du servlet

Annotations

- Depuis la spécification Servlet 3.0, il est possible de se passer de fichier web.xml pour déclarer un servlet
  - Version 3.1 (async, websocket, …)
  - Version 4.0 (HTTP/2, *multiplexing, push,* …)

- Il suffit d'annoter la classe du servlet par l'annotation **@WebServlet** qui possède des attributs pour donner toutes les informations nécessaires au fonctionnement du servlet : nom, pattern URL,…

Exemple

```
package fr.ub.m2gl.servlet;
…

@WebServlet(urlPatterns="/foo")
public class HelloServlet extends HttpServlet {
   …
   }
}
```

http://www.machin.com/**nomAppli**/foo

Fichier web.xml

- Le *mapping* peut aussi être donné dans le fichier **web.xml** qui décrit toute application Web Java (étudié plus loin en détails)

- Ce fichier sera contenu dans un fichier WAR (extension « .war »), fichier ZIP qui enveloppe tous les composants, code et ressources de l'application Web

Exemple de web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app …>
  <session-config>
     <session-timeout>30</session-timeout>
  </session-config>
  <servlet>
     <servlet-name>HelloWorld</servlet-name>
     <servlet-class>fr.ub.m2gl.servlet.Hello</servlet-class>
  </servlet>
  <servlet-mapping>
     <servlet-name>HelloWorld</servlet-name>
     <url-pattern>hello</url-pattern>
  </servlet-mapping>
</web-app>
```

Joker dans les mappings

- Le modèle pour l'URL peut contenir un joker (*) à la fin d'une URL qui commence par « / » ou juste avant une extension

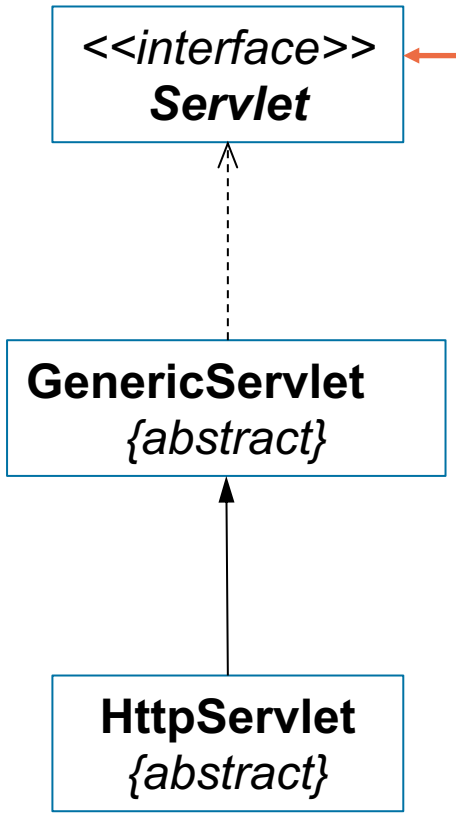
- Exemple
  - **<url-pattern>/servlet/*</url-pattern>**
  - **<url-pattern>*.serv</url-pattern>**

Les paquetages

- Le paquetage *javax.servlet* contient les classes et interfaces utilisées pour écrire des servlets

- Le paquetage *javax.servlet.http* contient les classes et interfaces pour écrire des servlets qui fonctionnent avec le protocole HTTP
  - c'est presque toujours le cas

Classes et interfaces de base



init

appelé par le conteneur à la création du servlet

service

traite une requête

destroy

appelé par le conteneur à la suppression du servlet

getServletConfig

donne accès aux paramètres d'initialisation du servlet (web.xml)

getServletInfo

informations sous forme de String

Classes et interfaces

- L’interface *ServletRequest* représente la requête du client au serveur
- L'interface *ServletResponse* représente la réponse du serveur
- Les interfaces *HttpServletRequest* et *HttpServletResponse* sont des interfaces filles pour travailler avec le protocole HTTP (ce qui est presque toujours le cas)

Traitements effectués par les servlets

- Traite les requêtes des clients Web
  - **public void service(ServletRequest, ServletResponse)**
- Méthode abstraite
  - Définir si on hérite de **GenericServlet**
- Majorité des servlets sont des servlets HTTP
  - Héritent de **HttpServlet**
  - Éviter de modifier la méthode service
  - Redéfinir les méthodes **doGet**, **doPost**,...

Traitements effectués par les servlets HTTP

- La méthode **service** de la classe **HTTPServlet** délègue le traitement à une autre méthode, selon le type de requête HTTP envoyée par le client :
  - **doGet()**
  - **doPost()**
  - **doPut()**
  - **doDelete()**
  - **doHead()**
  - **doOptions()**
  - **doTrace()**

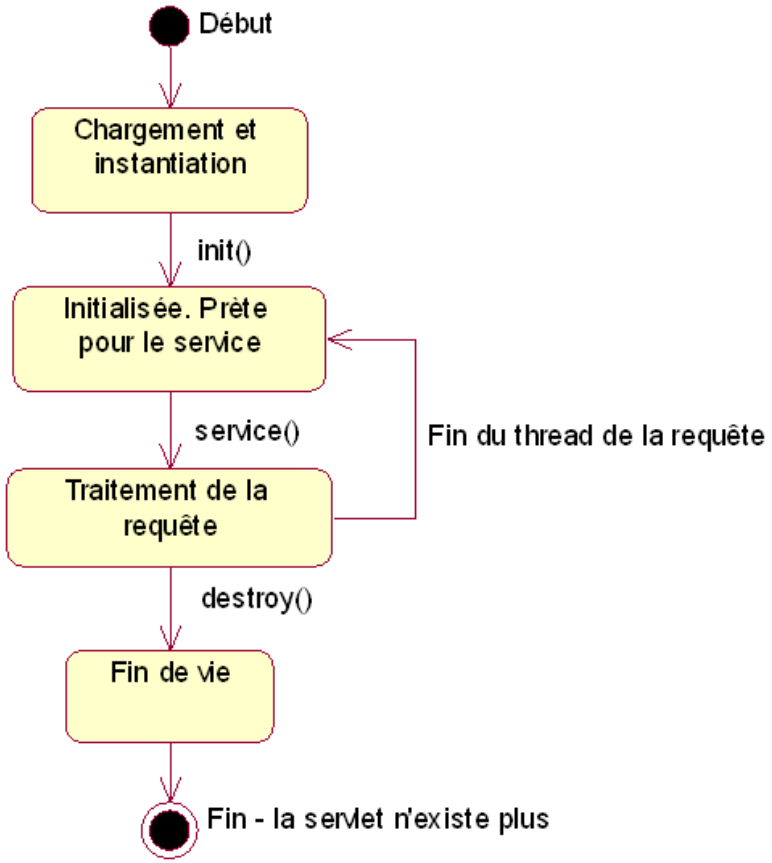
Cycle de vie d'un servlet

- Le serveur d'application ne crée qu'une seule instance de servlet par application et par type de servlet
- Le servlet est créé à la première connexion à une URL qui correspond (*url-mapping*) au servlet
- Le servlet est supprimé quand l'application est retirée du serveur (*undeploy*)

Un thread par client

- Création d'un thread par le container de servlet
  - Pour traiter chaque connexion ultérieure qui doit être traitée par le même type de servlet
- Variables d'instance des servlets
  - Accessibles par tous les clients
  - Ne pas garder d'informations spécifiques à un client
- Éventuelles informations conservée dans le servlet
  - Protection contre les accès concurrents

Cycle de vie



- Init
  - Initialiser des ressources qui seront utiles pendant l'existence du servlet
    Exemple: accès à une BDD
- Destroy
  - Libérer les ressources
    Exemple: connexions à une BDD

Code d’un servlet

- Écrire un servlet HTTP c’est
  - écrire une classe qui hérite de **HttpServlet**
  - redéfinir au moins une des méthodes **doGet**, **doPost**,…
  - redéfinir éventuellement une des méthodes **init** ou **destroy** si le servlet gère des ressources qu’il faut initialiser ou supprimer

HttpServlet(Request|Response)

- HttpServletRequest
  - Objet correspondant à la requête HTTP
    - headers, cookies envoyés par la requête, paramètres
- HttpServletResponse
  - Objet correspondant à la réponse qui sera envoyée
    - Code de statut de la réponse
    - Headers et cookies
  - Contient la méthode `getWriter()` pour obtenir le flot de sortie pour générer la page HTML résultat
  - Contient une méthode pour envoyer au client une réponse de redirection ou un message d'erreur

Paramètres d'initialisation d'un servlet

- Informations qui pourront être utilisées par le servlet
  - Exemple: adresse d'une base de données
- Valeurs données dans le fichier web.xml ou dans l'annotation @WebServlet
- Le fichier web.xml l'emporte sur les annotations
  - Permet de modifier ces informations sans devoir recompiler l'application

Exemple d’initialisation dans web.xml

```
<servlet>
   <servlet-name>Servlet1</servlet-name>
   <init-param>
      <param-name>p1</param-name>
      <param-value>v1</param-value>
   </init-param>
   <init-param>
      <param-name>p2</param-name>
      <param-value>v2</param-value>
   </init-param>
</servlet>
```

Exemple d’initialisation avec annotation

```
@WebServlet(
   name = "Servlet1",
   urlPatterns = {"/add1"},
   initParams = {
      @InitParam(name="p1", value="v1"),
      @InitParam(name="p2", value="v2")} )
public class Servlet1 extends HttpServlet {
...
```

Récupérer les paramètres d'initialisation

- Méthodes de l'interface **ServletConfig**
  - **HttpServlet** implémente **ServletConfig**

- ServletConfig contient les méthodes
  - **`Enumeration<String> getInitParameterNames()`**
    - retourne une énumération des noms des paramètres
  - **`String getInitParameter(String)`**
    - retourne la valeur du paramètre d'initialisation dont le nom est passé en paramètre

```
public class Servlet1 extends HttpServlet {
   ...
   Enumeration<String> e = getInitParameterNames();
   while (e.hasMoreElements()) {
     String nom = e.nextElement();
     String v = getInitParameter(nom);
     ...
   }
...
}
```

Contexte d'une application

- Plusieurs applications hébergées par le même serveur Web
  - Chaque application Web décrite par un fichier web.xml (ou par des annotations) définit un contexte
    - interface **ServletContext** qui représente le contexte dans lequel l'application s'exécute
- Durant l'exécution
  - URL donné par le client détermine le contexte utilisé
  - Obtenir le contexte : **request.getContextPath()**
  - Exemple pour « http://machin.com**/appli**/liste »

Paramètres d'initialisation d'un contexte

- Paramètres seront connus de tous les servlets de l'application
- Initialisation d'un paramètre dans web.xml :

```
<web-app>
   <context-param>
      <param-name>p1</param-name>
      <param-value>v1</param-value>
   </context-param>
   ...
```

```
ServletContext contexte = getServletContext();
Enumeration<String> e = contexte.getInitParameterNames();
while (e.hasMoreElements()) {
   String nom = e.nextElement();
   String v = contexte.getInitParameter(nom);
   ...
}
```

Forward et redirect

- Déléguer des traitement à une autre partie de l'application ou à une autre application
- *forward*
  - Servlet passe la main à une autre partie de l'application
    - Envoie la requête reçue et la référence pour répondre au client
  - Transparent pour le client

```
public void doGet (HttpServletRequest request,
                   HttpServletResponse response)
        throws ServletException, IOException {
    ...
    request.getRequestDispatcher("/servlet1").forward(request, response);
}
```

Forward et redirect

- Déléguer des traitement à une autre partie de l'application ou à une autre application
- *redirect*
  - Permet de passer la main en dehors de l'application
  - Réponse demandant au client d'envoyer une requête vers une autre URL

```
public void doGet (HttpServletRequest request,
                   HttpServletResponse response)
       throws ServletException, IOException {
   ...
   response.sendRedirect("http://.../index.html?p1=val1");
}
```

Conserver des données

- Une requête peut être traitée par plusieurs servlets
  - Exemple du *forward*
- Les servlets permettent de conserver des données pendant une requête de l'utilisateur ou même entre 2 requêtes, pendant une session de travail de l'utilisateur
- Ils peuvent même partager des données avec tous les servlets de l'application
  - y compris ceux qui sont utilisés par d'autres utilisateurs

3 portées pour conserver des données

- *Requête*
  - depuis l'arrivée de la requête jusqu'au renvoi de la réponse au client
  - Accès par : **HttpServletRequest**
- *Session*
  - de la première connexion de l'utilisateur jusqu'à expiration par timeout (optionnel) ou fermeture explicite de la session (ou arrêt du serveur)
  - Accès par **HttpSession getSession()**
- *Application*
  - toute la durée de vie de l'application
  - Accès par **ServletContext getServletContext()**

Attributs

- Conserver des objets (types primitifs interdits) dans les différentes portées en utilisant la notion d'attribut
- Ranger un objet val1 dans la session :

```
HttpSession session = request.getSession();
session.setAttribute("p1", val1);
```

- Récupérer l'objet dans la session :

```
HttpSession session = request.getSession();
Type1 v1 = (Type1)session.getAttribute("p1");
```

- Code semblable pour les portées application et requête

Gestion de sessions dans les Servlets

- Il existe une « Servlet session API »
  - Permet de récupérer un objet `HttpSession` à partir de la requête (`HTTPServletRequest`)
  - L'objet HTTPSession est une HashMap Java
  - Gestion des objets dans la session
    - ajout, modification, retrait, recherche, etc.
  - Gestion des méta-information à propos de la session
    - date de création, identiant de la session, etc.

Récupérer l'objet session

- Utiliser cette méthode : `HttpServletRequest.getSession()`
- Exemple:

```
HttpSession session = request.getSession(true);
```

- Retourne la session en cours, en crée une (true) si il n'en existe pas.
- Pour savoir si il s'agit d'une nouvelle session utiliser la méthode `isNew()` de la session

En coulisse…

- Quand on appelle `getSession(true)` chaque utilisateur se voit attribuer un Session ID
- Le Session ID est communiqué au client
  - Option 1: si le navigateur supporte les cookies, la Servlet crée un cookie avec le session ID, dans Tomcat, ce cookie est appelé `JSESSIONID`
  - Option 2: si le navigateur ne supporte pas les cookies, la servlet va essayer de récupérer le session ID depuis l'URL

Récupérer des données de la session

- L'objet session fonctionne comme une `HashMap`
  - Peut stocker n'importe quel type d'objet,
  - Les objets ont une « clé d'accès » comme dans les Maps Java
- Exemple de récupération :

```
Integer accessCount =(Integer) session.getAttribute("accessCount");
```

- Récupérer toutes les « clés » de tous les objets dans la session :

```
Enumeration attributes = request.getAttributeNames();
```

Mettre des données dans la session

- Les objets que l'on met dans la sessions sont des "attributs" :

```
HttpSession session = request.getSession();
session.setAttribute("nom", "Michel Buffa");
```

- Et on peut les supprimer :

```
session.removeAttribute("name");
```

Autres informations de session

- Récupérer le session ID, par exemple : `gj9xswvw9p`

```
public String getId();
```

- Voir si la session vient juste d'être créée :

```
public boolean isNew();
```

- Récupérer la date de création :

```
public long getCreationTime();
```

- Dernière fois que la session a été activée (ex dernière date de connexion)

```
public long getLastAccessedTime();
```

Session Timeout

➔ Récupérer le plus grand temps (secondes) d'inactivité de la session (ex : on veut la fermer si pendant 5mn on ne fait rien)

```
public int getMaxInactiveInterval();
```

› Si on spécifie cet intervalle, la session sera fermée (invalidée) automatiquement lorsqu'un utilisateur ne fait rien pendant un temps plus long:

```
public void setMaxInactiveInterval (int seconds);
```

› Si la valeur est négative : jamais d'interruption

Terminer une session

- Pour terminer (invalider) une session :

```
public void invalidate();
```

  - Typiquement, on fait ça au logout, ou au passage d'une commande sur un site de e-commerce
- Les sessions peuvent se terminer automatiquement lors de périodes d'inactivité

Login / Logout – Exemple

- Une appli web protégée par login / password
  - On utilise la session pour stocker l'utilisateur loggué
    - On utilise la clé "username"
    - Lorsqu'elle est présente, la valeur = le nom de l'utilisateur loggué
  - Lors de l'authentification on rajoute la clé si les login/password sont valides
  - Cliquer sur logout invalide la session
  - La servlet principale vérifie que l'utilisateur courant est loggué

Formulaire de login

LoginForm.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Login</title>
</head>
<body>
    <form action="LoginServlet" method="post">
        <h1>Login</h1>
        Username <input type="text" name="username"/>
        Password <input type="password" name="password" />
    </form>
</body>
</html>
```

LoginServlet

LoginServlet.java

```
public class LoginServlet extends HttpServlet {
  public void doPost(HttpServletRequest req,
    HttpServletResponse resp)
  throws IOException, ServletException {

    String username = req.getParameter("username");
    String password = req.getParameter("password");

    if (isLoginValid(username, password)) {
      HttpSession session = req.getSession();
      session.setAttribute("USER", username);
      response.sendRedirect("MainServlet");
    } else {
      response.sendRedirect("InvalidLogin.html");
    }
} }
```

LogoutServlet

LogoutServlet.java

```
public class LogoutServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request,
    HttpServletResponse response)
  throws ServletException, IOException {

    HttpSession session = request.getSession();
    session.invalidate();

    response.setContentType("text/html");
    ServletOutputStream out =
      response.getOutputStream();
    out.println("<html><head>");
    out.println("<title>Logout</title></head>");
    out.println("<body>");
    out.println("<h1>Logout successfull.</h1>");
    out.println("</body></html>");
} }
```

MainServlet

MainServlet.java

```
public class MainServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
    HttpServletResponse resp)
  throws ServletException, IOException {

    HttpSession session = request.getSession();
    String userName = (String)
      session.getAttribute("USER");

    if (userName != null) {
      response.setContentType("text/html");
      ServletOutputStream out = resp.getOutputStream();
      out.println("<html><body><h1>");
      out.println("Hello, " + userName + "! ");
      out.println("</h1></body></html>");
    } else {
      response.sendRedirect("LoginForm.html");
} } }
```

Page InvalidLogin.html

InvalidLogin.html

```
<html>
  <head>
    <title>Error</title>
  </head>
  <body>
    <h1>Invalid login!</h1>
    Please <a href="LoginForm.html">try again</a>.
  </body>
</html>
```

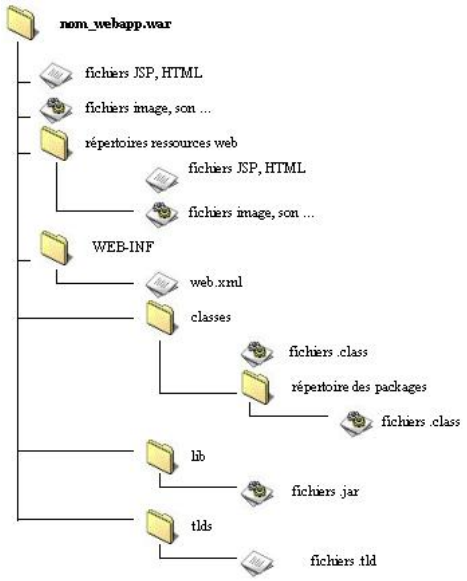
Problèmes avec le cache du navigateur

- La plupart des navigateurs utilisent un cache pour les pages et les images
  - L'utilisateur peut voir "l'ancien état d'une page"
    - Peut paraitre pour un bug, surtout dans le cas d'une page de login
- Pour éviter cela, il faut désactiver le cache dans la réponse HTTP :

```
response.setHeader("Pragma", "No-cache");
response.setDateHeader("Expires", 0);
response.setHeader("Cache-Control", "no-cache");
```

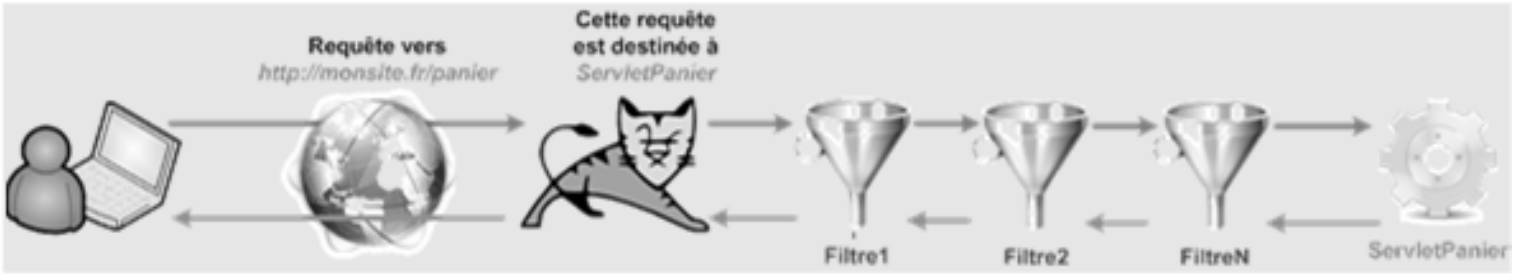
Structure d'une application Web

- Spécification des servlets normalise la présentation des applications Web
  - Améliore la portabilité des applications
  - Fichier au format jar mais d'extension war
- Structure d'un fichier .war
  - Répertoire racine
    - Pages statiques html
    - Ressources (images, fichiers css, fichiers js,…)
  - Sous-répertoire WEB-INF
    - Classes Java utilisées par l'application
    - fichier de configuration de l'application



Filtres

- Objet qui intercepte
  - Requêtes avant qu'elles ne soient traitées par un servlet
  - Réponses juste après qu'elles aient été générées par un servlet
- Chaînage des filtres
  - À l'image des pipes sous Unix



Déclaration d'un filtre

- Balise **<filter>** du fichier web.xml ou annotation **@WebFilter** de la classe du filtre

- Indication des requêtes qui seront filtrées par la balise **<filter-mapping>** de web.xml ou un attribut de l'annotation **@WebFilter**
  - nom d'un ou plusieurs servlets
  - ou un ou plusieurs modèles d'URL

Paramètres d’initialisation

- Comme pour les servlets, on peut donner des paramètres d’initialisation à un filtre

- On peut utiliser pour cela
  - balise <init-param> (dans la balise <filter>) dans le fichier web.xml ou
  - annotation @WebInitParam, directement dans l’attribut initParams de l’annotation @WebFilter

Exemples

```
<filter>
   <filter-name>Filtre1</filter-name>
   <filter-class>fr.foo.bar.Filtre1</filter-class>
   <init-param>
      <param-name>p1</param-name>
      <param-value>v1</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>Filtre1</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

```
@WebFilter(
   filterName = "Filtre1",
   urlPatterns = {"/*"},
   initParams = {
      @WebInitParam(name="p1", value="v1")
   }
)
public class Filtre1 implements Filter { ... }
```

Ordre d’exécution des filtres

- ➔ Si plusieurs filtres peuvent s’appliquer
  - › Ordre d’exécution déterminé par l’ordre de déclaration des **<filter-mapping>** dans le fichier web.xml
    - pas possible avec l’annotation

- ➔ En appelant doFilter, on appelle le prochain filtre
  - › Cas du dernier filtre ➔ envoie de la requête au servlet ou envoie de la réponse au client