SERVLET

Claude Duvallet

Université du Havre
UFR Sciences et Techniques
25 rue Philippe Lebon - BP 540
76058 LE HAVRE CEDEX
Claude.Duvallet@gmail.com
http://litis.univ-lehavre.fr/∼duvallet/

Les Servlet en Java

Introduction (1/2)

- Avec les servlets, nous plongeons au cœur de l'API servlet qui est un des composants de base pour le développement d'applications Web.
- Une servlet est un programme qui s'exécute côté serveur en tant qu'extension du serveur.
- Elle reçoit une requête du client, elle effectue des traitements et renvoie le résultat.
- La liaison entre la servlet et le client peut être directe ou passer par un intermédiaire comme par exemple un serveur http.

Introduction (2/2)

- Même si pour le moment la principale utilisation des servlets est la génération de pages html dynamiques utilisant le protocole http et donc un serveur web, n'importe quel protocole reposant sur le principe de requête/réponse peut faire usage d'une servlet.
- Écrite en java, une servlet en retire ses avantages : la portabilité, l'accès à toutes les API de java dont JDBC pour l'accès aux bases de données, ...
- Une servlet peut être invoquée plusieurs fois en même temps pour répondre à plusieurs requêtes simultanées.
- La servlet se positionne dans une architecture Client/Serveur trois tiers dans le tiers du milieu entre le client léger chargé de l'affichage et la source de données.

Historique (1/2)

2.0 1997.

2.1 Novembre 1998 :

- Création d'un partage d'informations grâce au Servletcontext.
- La classe GenericServlet implémente l'interface ServletConfig.
- Mise en place d'une méthode `log()` standard pour envoyer des informations dans le journal du conteneur.
- L'objet `RequestDispatcher` est utilisé pour le transfert du traitement de la requête vers une autre ressource ou inclure le résultat d'une autre ressource.

2.2 Aout 1999 :

- Apparition du format war pour un déploiement standard des applications web.
- Mise en buffer de la réponse.
- Standard inclu dans J2EE 1.2.

Historique (2/2)

2.3 Septembre 2001 :

- JSR 053 : nécessite le JDK 1.2 minimum.
- Ajout d'un mécanisme de filtre.
- Ajout de méthodes pour la gestion d'événements liés à la création et la destruction du context et de la session.
- Standard inclu dans J2EE 1.3.

2.4 Novembre 2003 :

- JSR 154.
- Standard inclu dans J2EE 1.4.

3.0 2010 : En cours de validation finale.

- JSR 315.
- Standard inclu dans Java EE 6.

Fonctionnement d'une Servlet

- Un serveur d'application (Exemple : Tomcat) permet de charger et d'exécuter les servlets dans une JVM.
- C'est une extension du serveur web.
- Ce serveur d'application contient entre autre un moteur de servlets qui se charge de manager les servlets qu'il contient.
- Pour exécuter une servlet, il suffit de saisir une URL qui désigne la servlet dans un navigateur.

Exécution d'une Servlet (1/2)

1. Le serveur reçoit la requête http qui nécessite une servlet de la part du navigateur.
2. Instanciation :
   - Si c'est la première sollicitation de la servlet, le serveur l'instancie.
   - Les servlets sont stockées (sous forme de fichiers .class) dans un répertoire particulier du serveur.
   - Ce répertoire dépend du serveur d'applications utilisé. La servlet reste en mémoire jusqu'à l'arrêt du serveur.
   - Certains serveurs d'applications permettent aussi d'instancier des servlets dès le lancement du serveur.
   - La servlet en mémoire, peut être appelée par plusieurs threads lancés par le serveur pour chaque requête.
   - Ce principe de fonctionnement évite d'instancier un objet de type servlet à chaque requête et permet de maintenir un ensemble de ressources actives telle qu'une connexion à une base de données.

Exécution d'une Servlet (2/2)

3. Le serveur crée un objet qui représente la requête http.
4. Il crée aussi un objet qui contiendra la réponse et l'envoi à la servlet.
5. La servlet crée dynamiquement la réponse sous forme de pages html transmises via un flux dans l'objet contenant la réponse.
6. La création de cette réponse utilise bien sûr la requête du client mais aussi un ensemble de ressources incluses sur le serveur tels de que des fichiers ou des bases de données.
7. Le serveur récupère l'objet "réponse" et envoie la page html au client.

Présentation de l'API Servlet

- Les servlets sont conçues pour agir selon un modèle de requête/réponse. Tous les protocoles utilisant ce modèle peuvent être utilisés tel que http, ftp, etc.
- L'API servlet est une extension du jdk de base, et en tant que telle elle est regroupée dans des packages préfixés par javax.
- L'API servlet regroupe un ensemble de classes dans deux packages :
  - javax.servlet : contient les classes pour développer des servlets génériques indépendantes d'un protocole.
  - javax.servlet.http : contient les classes pour développer des servlets qui reposent sur le protocole http utilisé par les serveurs web.

Les interfaces du package `javax.servlet`

- **RequestDispatcher :** Définition d'un objet qui permet le renvoi d'une requête vers une autre ressource du serveur (une autre servlet, une JSP, ...).
- **Servlet :** Définition de base d'une servlet.
- **ServletConfig :** Définition d'un objet pour configurer la servlet.
- **ServletContext :** Définition d'un objet pour obtenir des informations sur le contexte d'exécution de la servlet.
- **ServletRequest :** Définition d'un objet contenant la requête du client.
- **ServletResponse :** Définition d'un objet qui contient la réponse renvoyée par la servlet.
- **SingleThreadModel :** Permet de définir une servlet qui ne répondra qu'à une seule requête à la fois.

Les classes et les exceptions du package `javax.servlet`

Les classes :

- **GenericServlet :** Classe définissant une servlet indépendante de tout protocole.
- **ServletInputStream :** Flux permet la lecture des données de la requête cliente.
- **ServletOutPutStream :** Flux permettant l'envoie de la réponse de la servlet.

Les exceptions :

- **SevletException :** Exception générale en cas de problème durant l'exécution de la servlet.
- **UnavailableException :** Exception levée si la servlet n'est pas disponible.

Le package `javax.servlet.http`

Les interfaces :

- **HttpServletRequest :** Hérite de ServletRequest : définit un objet contenant une requête selon le protocole http.
- **HttpServletResponse :** Hérite de ServletResponse : définit un objet contenant la réponse de la servlet selon le protocole http.
- **HttpSession :** Définit un objet qui représente une session.

Les classes :

- **Cookie :** Classe représentant un cookie (ensemble de données sauvegardées par le navigateur WEB sur le poste client).
- **HttpServlet :** Hérite de GenericServlet : classe définissant une servlet utilisant le protocole http.
- **HttpUtils :** Classe proposant des méthodes statiques utiles pour le développement de servlet http.

L'interface `Servlet`

- Une servlet est une classe Java qui implémente l'interface `javax.servlet.Servlet`.
- Cette interface définit 5 méthodes qui permettent au conteneur web de dialoguer avec la servlet.
  - **void service (ServletRequest req, ServletResponse res) :** Cette méthode est exécutée par le conteneur lorsque la servlet est sollicitée par un client.
  - **void init(ServletConfig conf) :** Initialisation de la servlet. Cette méthode est appelée une seule fois après l'instanciation de la servlet en mode bloquant.
  - **ServletConfig getServletConfig() :** Renvoie l'objet `ServletConfig` passé à la méthode `init()`.
  - **void destroy() :** Cette méthode est appelée lors de la destruction de la servlet par le serveur d'application. Elle permet de libérer proprement certaines ressources (fichiers, bases de données ...).
  - **String getServletInfo() :** Renvoie des informations sur la servlet.

Exemple 1 : servlet implémentant l'interface `Servlet` (1/2)

```
import java.io.*;
import javax.servlet.*;

public class MaServlet implements Servlet {
  private ServletConfig cfg;

  public void init(ServletConfig config) throws ServletException {
     cfg = config;
  }

  public ServletConfig getServletConfig() {
    return cfg;
  }

  public String getServletInfo() {
    return "Une servlet de test";
  }

  public void destroy() {
  }
```


Exemple 1 : servlet implémentant l'interface `Servlet` (2/2)

```
  public void service (ServletRequest req,  ServletResponse res)
        throws ServletException, IOException  {
    res.setContentType( "text/html" );
    PrintWriter out = res.getWriter();
    out.println( "<HTML>" );
    out.println( "<HEAD>");
    out.println( "<TITLE>Page generee par une servlet</TITLE>" );
    out.println( "</HEAD>" );
    out.println( "<BODY>" );
    out.println( "<H1>Bonjour / Hello / Salam Alaikoum</H1>" );
    out.println( "<H2>Labass ?</H2>" );
    out.println( "</BODY>" );
    out.println( "</HTML>" );
    out.close();
  }
}
```

Exemple 1 : le code source XML

- Il faut aussi écrire un fichier `web.xml`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

  <display-name>Exemples de Servlet en 2.4</display-name>
    <description>
      Exemples de Servlet en 2.4.
    </description>

    <servlet>
        <servlet-name>MaServlet</servlet-name>
        <servlet-class>MaServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>MaServlet</servlet-name>
        <url-pattern>/servlet/MaServlet</url-pattern>
    </servlet-mapping>

</web-app>
```


Exemple 1 : Installation des fichiers

- Nous allons créer l'arborescence suivante :

```
cd $TOMCAT_HOME/webapps
mkdir -p MesExemples/WEB-INF/classes
```

- Plaçons le fichier `MaServlet.class` et `web.xml` au bon endroit :

```
cp web.xml $TOMCAT_HOME/webapps/MesExemples/WEB-INF/
cp MaServlet.class $TOMCAT_HOME/webapps/MesExemples/WEB-INF/classes/
```


Exemple 1 : exécution

- Redémarrer le serveur d'application Tomcat :

```
sh $TOMCAT_HOME/bin/shutdown.sh ;
sh $TOMCAT_HOME/bin/startup.sh
```

- Utiliser un navigateur WEB pour visualiser le résultat :
- Taper l'URL :

`http://localhost:8080/MesExemples/servlet/MaServlet`.

Les servlets HTTP (1/3)

- L'usage principal des servlets est la création de pages HTML dynamiques.
- Sun fourni une classe qui encapsule une servlet utilisant le protocole http : la classe `HttpServlet`.
- Cette classe hérite de `GenericServlet`, donc elle implémente l'interface `Servlet`.
- Ce type de servlet ne sert pas uniquement à générer des pages HTML bien que cela soit son principal usage.
- Elle peut aussi réaliser un ensemble de traitements tel que mettre à jour une base de données.
- Elle définit un ensemble de fonctionnalités très utiles : par exemple, elle contient une méthode `service()` qui appelle certaines méthodes à redéfinir en fonction du type de requête http (`doGet()`, `doPost()`, etc ...).

Les servlets HTTP (2/3)

- La requête du client est encapsulée dans un objet qui implémente l'interface `HttpServletRequest` et contient les données de la requête ainsi que des informations sur le client.
- La réponse de la servlet est encapsulée dans un objet qui implémente l'interface `HttpServletResponse`.
- Typiquement pour définir une servlet, il faut définir une classe qui hérite de la classe `HttpServlet` et redéfinir la méthode doGet et/ou doPost selon les besoins.
- La méthode service héritée de `HttpServlet` appelle l'une ou l'autre de ces méthodes en fonction du type de la requête http :
  - une requête GET : c'est une requête qui permet au client de demander une ressource.
  - une requête POST : c'est une requête qui permet au client d'envoyer des informations issues par exemple d'un formulaire.

Les servlets HTTP (3/3)

- Une servlet peut traiter un ou plusieurs types de requêtes grâce à plusieurs autres méthodes :
  - **doHead() :** pour les requêtes http de type HEAD
  - **doPut() :** pour les requêtes http de type PUT
  - **doDelete() :** pour les requêtes http de type DELETE
  - **doOptions() :** pour les requêtes http de type OPTIONS
  - **doTrace() :** pour les requêtes http de type TRACE
- La classe `HttpServlet` hérite aussi de plusieurs méthodes définies dans l'interface `Servlet` : `init(), destroy()` et `getServletInfo()`.

Exemple 2 : le code source Java

- Nous allons écrire une classe BonjourServlet qui hérite de la classe `HttpServlet`.

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

public class BonjourServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
   PrintWriter out = response.getWriter();
   out.println("<html>");
   out.println("<body>");
   out.println("<h1>Bonjour le monde depuis ma premiere Servlet</h1>");
   out.println("</body>");
   out.println("</html>");
  }
}
```


Exemple 2 : le code source XML

- Il faut ajouter le bloc suivant dans le fichier `web.xml`.

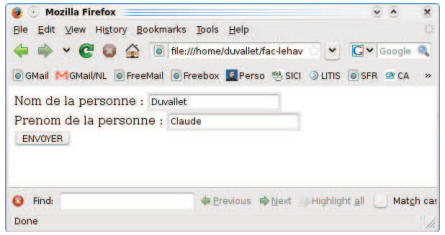
```
<servlet>
    <servlet-name>BonjourServlet</servlet-name>
    <servlet-class>BonjourServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>BonjourServlet</servlet-name>
    <url-pattern>/servlet/BonjourServlet</url-pattern>
</servlet-mapping>
```

- Les étapes suivantes sont les mêmes que pour l'exemple 1 : n'oublier pas de redémarrer le serveur Tomcat.

Exemple 3 : Création d'un formulaire de saisie

- Il s'agit de créer un formulaire HTML dont le traitement sera effectuée par une servlet.
- Le formulaire, qui sera sauvegardé dans une page html, est le suivant :

```
<FORM ACTION="http://localhost:8080/MesExemples/servlet/MonExemple3"
      METHOD="POST">
  Nom de la personne : <INPUT NAME="NOM"><br/>
  Prenom de la personne : <INPUT NAME="PRENOM"><br/>
  <INPUT TYPE="SUBMIT" VALUE="ENVOYER">
</FORM>
```



- Ensuite, il faut créer la servlet chargée de traiter ce formulaire.

Exemple 3 : La servlet MonExemple3

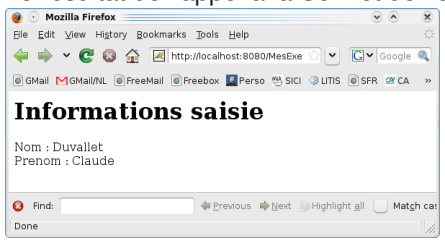
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MonExemple3 extends HttpServlet {

  public void doPost(HttpServletRequest request, HttpServletResponse response)
  throws IOException, ServletException
  {
    String nom = request.getParameter("NOM");
    String prenom = request.getParameter("PRENOM");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<body>");
    out.println("<h2>Informations saisies</h2>");
    out.println("Nom :"+ nom+" <br/>");
    out.println("Prenom :"+ prenom+" <br/>");
    out.println("</body>");
    out.println("</html>");
  }
}
```


Exemple 3 : Résultat de l'exécution

- Il faut charger la page HTML contenant le serveur.
- Saisissez des informations dans le formulaire puis cliquer sur "ENVOYER".
- Le résultat de l'appel à la Servlet devrait être le suivant :



Les informations contenues dans la requête (1/2)

- De nombreuses informations en provenance du client peuvent être extraites de l'objet `ServletRequest` passé en paramètre par le serveur (ou de `HttpServletRequest` qui hérite de `ServletRequest`).
- Les informations les plus utiles sont les paramètres envoyés dans la requête.
- L'interface ServletRequest dispose de nombreuses méthodes pour obtenir ces informations :
  - **int getContentLength() :** Renvoie la taille de la requête, 0 si elle est inconnue.
  - **String getContentType() :** Renvoie le type MIME de la requête, null si il est inconnu
  - **ServletInputStream getInputStream() :** Renvoie un flux qui contient le corps de la requête

Les informations contenues dans la requête (2/2)

- Suite des méthodes :
  - **Enumeration getParameterNames() :** Renvoie une énumération contenant le nom de tous les paramètres
  - **String getProtocol() :** Retourne le nom du protocole et sa version utilisé par la requête
  - **BufferedReader getReader() :** Renvoie un flux qui contient le corps de la requête
  - **String getRemoteAddr() :** Renvoie l'adresse IP du client
  - **String getRemoteHost() :** Renvoie le nom de la machine cliente
  - **String getScheme() :** Renvoie le protocole utilisé par la requête (exemple : http, ftp ...)
  - **String getServerName() :** Renvoie le nom du serveur qui a reçu la requête
  - **int getServerPort() :** Renvoie le port du serveur qui a reçu la requête

Exemple 4 : Récupération des informations

- Il s'agit de créer un formulaire HTML dont le traitement affichera les informations obtenues par la servlet.
- Le formulaire qui sera sauvegardé dans une page html est le suivant :

```
<FORM ACTION="http://localhost:8080/MesExemples/servlet/MonExemple4"
      METHOD="POST">
  Nom de la personne : <INPUT NAME="NOM"><br/>
  Prenom de la personne : <INPUT NAME="PRENOM"><br/>
  <INPUT TYPE="SUBMIT" NAME="ENVOI" VALUE="ENVOYER">
  <INPUT TYPE="SUBMIT" NAME="ENVOI" VALUE="NE PAS ENVOYER">
</FORM>
```

- Ensuite il faut créer la servlet chargée de traiter ce formulaire.

Exemple 4 : La servlet MonExemple4

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException
{
  String nom = request.getParameter("NOM");
  String prenom = request.getParameter("PRENOM");
  String resultat = request.getParameter("ENVOI");
  PrintWriter out = response.getWriter();
  out.println("<html>");
  out.println("<body>");
  out.println("<head>");
  out.println("<title>Informations a disposition de la servlet</title>");
  out.println("</head>");
  out.println("<body>");
  out.println("<h2>Informations a disposition de la servlet</h2>");
  out.println("<p>Type Mime de la requete : "+request.getContentType()+"</p>");
  out.println("<p>Protocole de la requete : "+request.getProtocol()+"</p>");
  out.println("<p>Adresse IP du client : "+request.getRemoteAddr()+"</p>");
  out.println("<p>Nom du client : "+request.getRemoteHost()+"</p>");
  out.println("<p>Nom du serveur qui a recu la requete : "+request.getServerName()+"</p>");
  out.println("<p>Port du serveur qui a recu la requete : "+request.getServerPort()+"</p>");
  out.println("<p>Scheme : "+request.getScheme()+"</p>");
  out.println("<p>Liste des parametres </p>");
  for (Enumeration e =request.getParameterNames() ; e.hasMoreElements() ; ) {
      Object p = e.nextElement();
      out.println("<p>  nom : "+p+" valeur :"+request.getParameter(""+p)+"</p>");
  }
  String envoi = request.getParameter("ENVOI");
  out.println("<p><b>Vous avez appuyer sur le bouton : "+ envoi +"</b></p>");

  out.println("</body>");
  out.println("</html>");
  }
```


Exemple 4 : Résultat de l'exécution

- Saisissez des informations dans le formulaire puis cliquer sur "ENVOYER" ou "NE PAS ENVOYER".