Les servlets

(1ere partie)

Bibliographies: Les notes de cours sur les servlets est une revue littéraire des documents mentionnés ci-dessus. Plusieurs extraits ont été tirés ou traduits de ces références. Cette revue tient compte des dernières mises à jour apportées au logiciel Tomcat et à la spécification (2.3) des servlets.

Site web officiel de SUN sur les servlets: http://java.sun.com/products/servlet/

Des documents sur les servlets:
http://www-sop.inria.fr/acacia/personnel/itey/Francais/Cours/PDF/servlets.pdf
http://home.nordnet.fr/~fadelannoy/servlets.html
http://java.sun.com/docs/books/tutorial/servlets/index.html
http://java.sun.com/products/servlet/articles/tutorial/

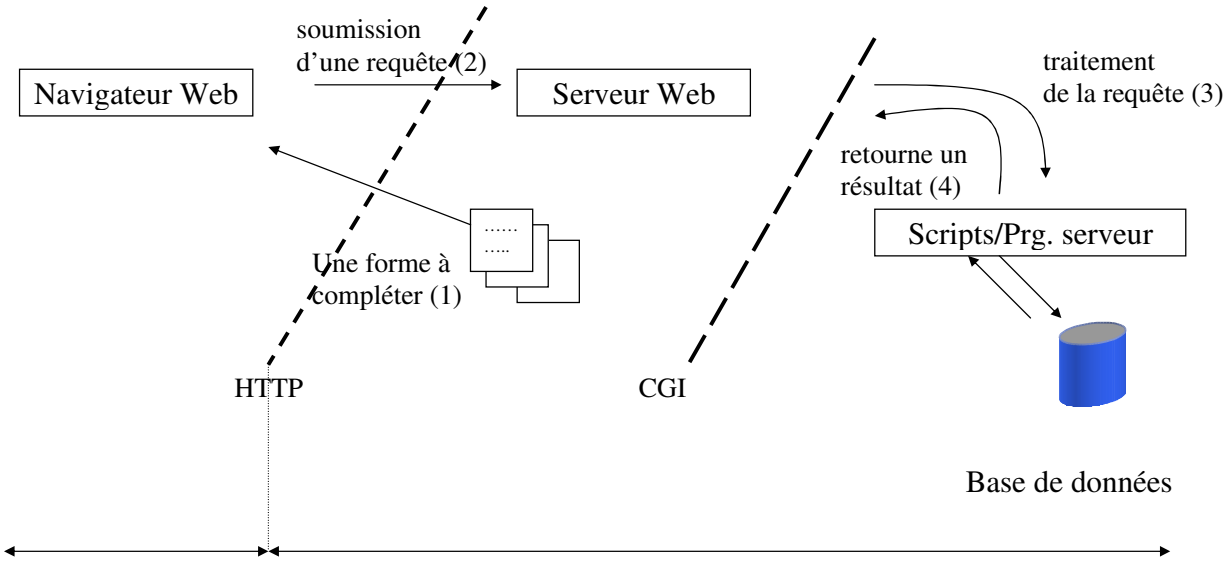
Des documents sur Tomcat:
http://jakarta.apache.org/tomcat/tomcat-4.0-doc/index.html
http://darktigrou.free.fr/Servlets-book2_der/node1.html

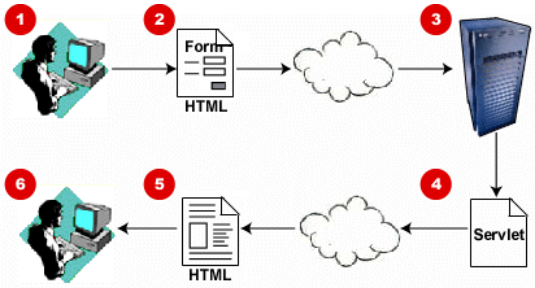
Dans cette présentation, nous allons répondre aux questions suivantes :

- C'est quoi une application web?
- Qu'est ce qu'une servlet Java?
- A quoi servent les servlets?
- Quels sont les avantages et les inconvénients des servlets?
- C'est quoi les différences entre les scripts CGI et les servlets?
- A quoi ressemble le code d'une servlet?
- Comment est constitué le paquetage Servlet?
- Comment se déroulent les interactions avec des clients?
- C'est quoi le cycle de vie d'une servlet?
- Comment configurer Tomcat4.0.1?
- Comment déployer une application dans Tomcat4.0.1?

C'est quoi un application web?

Une application web est constituée de 2 niveaux, le client émettant une requête et le serveur traitant cette requête.





(1) Un serveur web soumet une forme d'un document à compléter (une requête),

(2) Le client complète la forme et la retourne au serveur,

(3) Le serveur la réceptionne et la redirige vers la servelt invoquée dans la page html. En exécutant cette requête, le programme peut communiquer avec une ou plusieurs bases

de données si c'est nécessaire. Ainsi donc, une servlet analyse les données soumises, peut éventuellement sauvegarder celles-ci ou collecter d'autres pour construire dynamiquement une page HTML.

(4) ces programmes retournent les résultats au serveur qu'il se charge de les transmettre au client.

Le client peut utiliser :

- un formulaire HTML, permettant de saisir les champs ou bien il peut y avoir validation via des scripts (javaScript) ;

- des applets et communiquer avec le serveur via des sockets/RMI.

Les requêtes http vers le serveur contiennent :

- l'url de la ressource à accéder,

- la requête GET pour extraire des informations sur le serveur,

- la requête POST pour modifier les données sur le serveur.

Le serveur identifie avec la requête le type d'environnement d'exploitation à charger en fonction de :

- l'extension du fichier (.cgi, .jsp etc.) ou

- le répertoire où il se trouve (cgi-bin/, servlet/ etc.).

Le serveur charge par la suite l'environnement d'exécution Perl (cgi-perl), JVM (servlets) etc.

Le script ou programme précise le type du contenu (HTML, images etc.) et intègre la réponse dans un flot de sortie.

Le navigateur définit le type MIME (l'encodage utilisé pour le transfert de documents multimédias à travers le réseau) texte/html audio/basic image/gif etc. et affiche les données en fonction.

Qu'est qu'une servlet Java?

Servlet : **Serv**er-side app**let**

Les servlets sont aux serveurs ce que sont les applets aux browsers mais sans interface graphique utilisateur …

Elles sont limitées à la puissance du langage HTML … par contre, elles ne sont pas astreintes aux mêmes règles de sécurité que les *applets*

Les servlets correspondent à des programmes Java normaux qui utilisent des modules supplémentaires (ainsi que les classes et les méthodes associées) figurant dans l'API des servlets Java.

Les servlets s'exécutent sur une machine de serveur Web à l'intérieur d'un serveur compatible avec Java. Elles permettent l'extension des fonctions du serveur.

Une servlet peut être chargé :

- automatiquement lors du démarrage du serveur Web,

- lorsque le premier client demande les services de la servlet.

Une fois chargées, les servlets restent actives dans l'attente d'autres requêtes du client.

Les servlets permettent l'extension des fonctions du serveur grâce à la création d'un environnement de prestation de services de requête/réponse via le Web :

> Un client envoie une requête au serveur. Ce dernier transmet au servlet les informations relatives à la requête. La servlet créée ensuite une réponse que le serveur renvoie au client.

Dans la mesure où il s'agit d'un programme Java :

- La servlet peut utiliser toutes les fonctions du langage Java lors de la création de la réponse.

- Elle peut également communiquer avec des ressources externes tels que des fichiers ou des bases de données, ou avec d'autres

applications (également écrites en langage Java ou dans d'autres langages), afin de créer la réponse et éventuellement de sauvegarder des informations relatives à l'interaction requête/réponse.

La réponse envoyée au client peut donc être une réponse dynamique et unique conçue pour une interaction particulière et non une page HTML statique existante.

A quoi servent les servlets?

Les servlets exécutent un grand nombre de fonctions, par exemple:

- Une servlet peut créer et renvoyer une page Web HTML complète dont le contenu dynamique dépend de la nature de la requête du client.

- Une servlet peut simplement créer une partie d'une page Web HTML qui est intégrée à une page HTML statique existante.

- Une servlet peut communiquer avec d'autres ressources du serveur, y compris des bases de données, d'autres applications Java et des applications écrites dans d'autres langages.

- Une servlet peut traiter les connexions avec plusieurs clients en acceptant les données en entrée de plusieurs clients et en diffusant à ces derniers des résultats. Une servlet peut, par exemple, correspondre à un serveur de jeux électroniques faisant intervenir plusieurs joueurs.

Quels sont les avantages et les inconvénients des servlets?

Avantages

Les servlets sont indépendantes des OS (Unix ou NT) et des serveurs Web (Apache, IIS etc.) ;

Peuvent produire de l'HTML côté client (notamment pour la consultation de la base), sur la base d'http;

Peuvent dialoguer avec des applets Java côté client avec un protocole à objets distribués de type RMI;

S'appuient sur un langage vraiment standard : Java (et non pas Java script ou Visual Basic);

Par rapport aux applets, le client est « allégé » ;

Par rapport aux CGI, les servlets prennent en charge les connexions des utilisateurs en multi-thread, qui n'est pas le cas des CGI (même avec FastCGI).

Inconvénients

Par rapport aux applets, interface graphique utilisateur limitée à HTML

Par rapport au CGI, voir plus loin dans le document.

C'est quoi les différences entre les scripts CGI et les servlets?

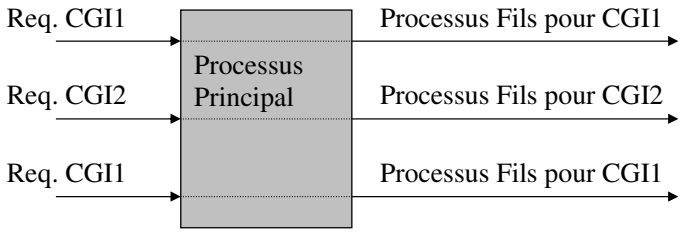
La plus grosse différence entre les CGI et les servlets est la performance.

- Il n'y a qu'une seule machine virtuelle Java qui tourne sur le serveur.

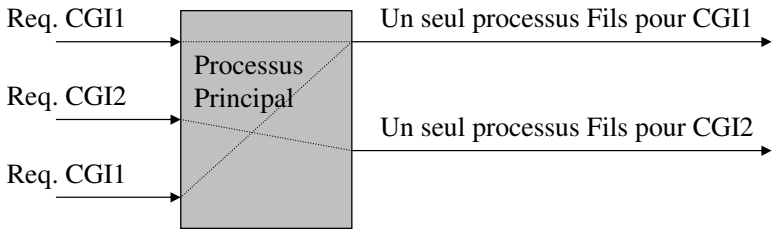
- La servlet est placée en mémoire une fois qu'elle est appelée. Elle n'est pas remise en mémoire jusqu'à ce que la servlet change.

- Une servlet dont le code a été modifié peut être réactivée (c'est à dire replacée en mémoire) sans redémarrer le serveur ou l'application.

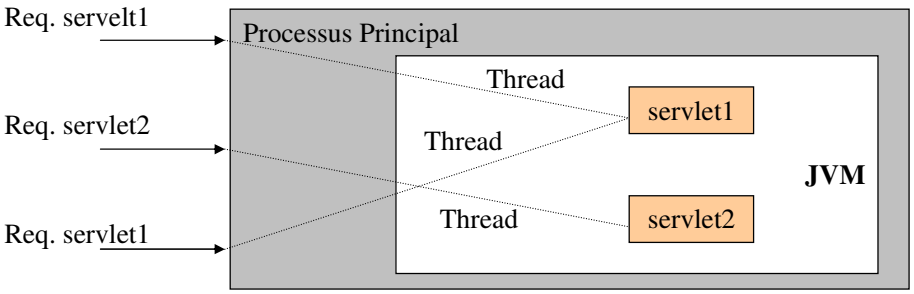
Common Gateway Interface (CGI)



FastCGI



Les servlets



Les servlets résident en mémoire, de ce fait leur exécution est très rapide. L'information statique peut être partagée par plusieurs invocations de la servlet, vous autorisant donc de partager cette information entre plusieurs utilisateurs.

Les servlets sont modulaires, chaque servlet peut accomplir une tache spécifique et ainsi vous pouvez les rendre communicantes.

Moins il y a de calcul à faire côté client, plus l'approche *servlet* est intéressante.

A quoi ressemble le code d'une servlet?

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

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

      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
```

```
        out.println("<html>");
        out.println("<body>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```


Comment est constitué le paquetage Servlet?

Le paquetage **javax.servlet** fournit les interfaces et classes pour réaliser des servlets.

À un moment donné, il était question d'intégrer ce paquetage dans la version 1.2 de JDK ; puis il a été décidé autrement pour permettre des modifications et corrections dans le paquetage de manière indépendante.

Par la suite, le paquetage a été intégré dans Java2 Platform, Enterprise Edition (J2EE)

http://java.sun.com/j2ee/

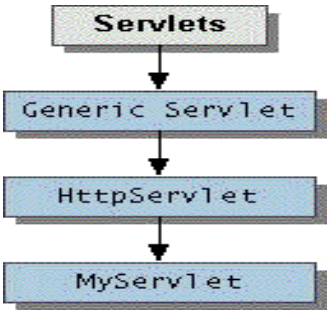
La relation avec Tomcat

Tomcat est un logiciel gratuit, open-source, qui implante les technologies Java Servlet et JavaServer Pages développées dans le cadre du projet Jakarta à la fondation "Apache Software Foundation."

http://jakarta.apache.org

Sun a adapté et intégré le code de base de Tomcat dans J2EE. Sun détient et continue de mettre à jour les spécifications de JavaServer PagesTM et JavaTM Servlets dans un processus ouvert à toute la communauté Java.

L'architecture du paquetage est décrite comme suit :



java.lang.Object
|
+--javax.servlet.GenericServlet
|
+--javax.servlet.http.HttpServlet

L'interface Servelt

Toutes les servlets implantent l'interface Servlet: **javax.servlet.Servlet**

soit directement ou bien via une classe qui implante cette interface, comme:

- **javax.servlet.GenericServlet**
(paquetage javax.servlet)
- **javax.servlet.http.HttpServlet**
(paquetage javax.servlet.http)
  - particulièrement désignée pour des requêtes et réponses HTTP

Comme toute interface, l'interface Servlet déclare mais n'implante pas les méthodes qui gèrent les servlets et les communications avec des clients. Cette charge revient aux concepteurs des servlets.

Cette interface possède les méthodes pour :

- initialiser la *servlet* : init()
- recevoir et répondre aux requêtes des clients : service()
- détruire la *servlet* et ses ressources : destroy()
- obtenir des informations sur la configuration de la servlet: getServletConfig()
- obtenir des informations comme l'auteur et la version de la servlet: getServletInfo()

Comment se déroulent les interactions avec des clients?

Les servlets suivent un modèle de programmation requête-service-réponse :

Requête : objet **javax.servlet.ServletRequest**
contient les informations nécessaires pour une communication du client vers le serveur

Service : méthode **service()** invoquée

Réponse : objet **javax.servlet.ServletResponse**
contient les informations nécessaires pour une communication du serveur vers le client.


```
import javax.servlet.*;

public class first implements Servlet {

  public void init(ServletConf config)
                throws ServletException {…}

  public void service(ServletRequest req,
                      ServletResponse rep)
         throws ServletException, IOException {…}

  public void destroy() {…}
}
```

L'interface **ServletRequest** permet à la servelt d'accéder à:

- Les noms des paramètres passés par le client, le protocole utilisé, les noms des machines ayant émis les requêtes, et le nom du serveur recevant ces requêtes.

- Le flux en entrée, **ServletInputStream**. Les servlets utilisent le flux en entrée pour prendre les données du client et utilise des protocoles d'accès comme le protocole HTTP.

L'interface **ServletResponse** permet à la servelt de répondre aux clients:

- permet à la servlet de fixer la taille de la réponse et son type.

- fournit un flux en sortie, **ServletOutputStream** utilisé par la servlet pour envoyer des données.

C'est quoi le cycle de vie d'une servlet?

Une *servlet* suit le cycle de vie suivant :

1. la *servlet* est créée puis initialisée (**`init()`**)
   - cette méthode n'est appelée par le serveur qu'une seule fois lors du chargement

2. le service du client est implémenté (**`service()`**)
   - cette méthode est appelée automatiquement par le serveur à chaque requête de client

3. la *servlet* est détruite (**`destroy())`**
   - cette méthode n'est appelée par le serveur qu'une seule fois à la fin
   - permet de libérer des ressources

Comment configurer Tomcat4.0.1?

Téléchargez à partir de

http://jakarta.apache.org/builds/jakarta-tomcat-4.0/release/v4.0.1/bin/

le fichier:

UNIX: jakarta-tomcat-4.0.1.tar.gz
Windows: jakarta-tomcat-4.0.1.zip

Décompressez le fichier dans un répertoire donné

n'installez pas ce paquetage sur une des machines du laboratoire

Ajouter les variables d'environnement suivantes à:

Win98: autoexec.bat

```
REM Pour Tomcat

REM Le chemin où a été installé JDK, par exemple:
SET JAVA_HOME=C:\jdk1.3

REM Le chemin où a été installé le paquetage de jakara-tomcat, par exemple:
SET CATALINA_HOME=C:\Program Files\jakarta-tomcat\jakarta-tomcat-4.0.1

REM Le class-path pour retrouver le fichier utilisé pour compiler les servlets
SET CLASSPATH=.
SET CLASSPATH=%CLASSPATH%; %CATALINA_HOME%\common\lib\servlet.jar
```


Il se peut que vous ayez l'erreur suivante "out of environment space" au lancement de Tomcat:
Cliquez sur le bouton droit de la souris sur les fichiers STARTUP.BAT et SHUTDOWN.BAT. Cliquez sur "Properties" puis "Memory". Dans le champ "Initial environment", entrez par exemple la valeur 4096.

Puis cliquez sur "apply". Windows va créer deux raccourcis (shortcuts) dans le répertoire, que vous allez dorénavant utiliser pour démarrer et arrêter Tomcat.

WIN2000: voir dans le panneau des variables Systemes

Unix: Cela dépend du type de l'installation. En tant que root une seule instance de Tomcat ou plusieurs instances de Tomcat ; ou en

tant qu'usager ordinaire? La procédure à suivre vous sera communiquée dans la prochaine démonstration.

Comment déployer une application dans Tomcat4.0.1?

Un contexte constitue pour chaque servlet d'une même application une vue sur le fonctionnement de cette application. Une application web peut être composée de :

servlets
JSP
classes utilitaires
documents statiques (pages html, images, sons, etc.)
beans, applications clients
méta informations décrivant la structure de l'application

Dans le code source d'une servlet, un contexte est représenté par un objet de type ServletContext qui peut être obtenu par l'intermédiaire d'un objet de type ServletConfig, par exemple de la façon suivante :

```
public void init(ServletConfig config) {

  ServletContext contexte = config.getServletContext();

  /* suite du code */

}
```


Grâce à ce contexte, il est possible d'accéder à chacune des ressources de l'application web correspondantes au contexte.

À noter: une application correspond à un et un seul contexte, et un contexte correspond une et une seule application => il n'est pas possible de partager des ressources entre applications différentes.

Un contexte est une abstraction supplémentaire qui permet de matérialiser les relations privilégiées que connaissent les modules d'une même application et le fait que chaque application est différente.

Configuration avec Tomcat

A chaque contexte correspond une arborescence dans le système de fichiers qui contient les ressources accédées lors des requêtes vers le moteur de servlets. Cette arborescence est identique pour chaque contexte. Voici comment se décompose la structure des répertoires :

Tout doit se faire à partir de :

CATALINA_HOME\webapps

la racine : elle fait office de répertoire racine pour les ressources qui font partie du contexte.

Par exemple l'URL
http://www.unserveur.com/essai/index.html

Elle fait référence au fichier index.html de ce répertoire racine. Vous pouvez créer les répertoires que vous désirez ou déposer les fichiers que vous voulez dans ce répertoire, comme par exemple un répertoire images/ qui sera accessible via l'URL
http://www.unserveur.com/essai/images/

le répertoire WEB-INF : situé à la racine, il contient un fichier web.xml qui est le descripteur de déploiement du contexte. Il contient tous les paramètres de configuration utilisés par le contexte.

le répertoire WEB-INF/classes/ : c'est le répertoire dans lequel vous déposez les classes de vos servlets et des classes personnalisées utilisées par celles-ci. Le chargeur de classe de l'application vient chercher les classes à charger dans ce répertoire.

le répertoire WEB-INF/lib/ : il permet de déposer les archives au format jar (Java ARchive Files) qui contiennent des servlets, des beans ou des classes utilitaires utilisées par l'application. Le chargeur de classe de l'application recherche aussi dans ces archives pour trouver les classes ce dont il a besoin.

2e partie

Configuration UdM:

chaque usager a un compte dans www2, accessible comme suit:

/home/www2/nom_login

dans ce répertoire, il faudra créer l'arborescence suivante:

/home/www2/nom_login/webapps

Soit moncontexte, un contexte associé à un exemple donné:

webapps/moncontexte
webapps/moncontexte/servlets
webapps/moncontexte/WEB-INF
webapps/moncontexte/WEB-INF/classes
webapps/moncontexte/WEB-INF/lib

ou bien:

webapps/examples
webapps/examples/servlets
webapps/examples/WEB-INF
webapps/examples/WEB-INF/classes
webapps/examples/WEB-INF/lib

Le contexte examples est associé (mapé) par défaut dans la configuration UdM de Tomcat.

Tomcat charge les contextes après leur démarrage. Un contexte crée par la suite, pour être chargé, il faudra arrêter Tomcat et le démarrer de nouveau.

Le contexte examples est utilisé dans le cas des servlets modifiées ou bien nouvellement créées. Ces servlets sont automatiquement chargées sans avoir recours à la séquence arrêt-démarrage de Tomcat.

javax.servlet.GenericServlet

(paquetage javax.servlet)

- **javax.servlet.http.HttpServlet**

(paquetage javax.servlet.http)

- particulièrement désignée pour des requêtes et réponses HTTP

GenericServlet

- Définit une servelt à base d'un protocole générique.
- Elle implante l'interface Servlet et ServletConfig.
- Pour écrire une Servlet générique, il suffit tout juste de définir la méthode service de l'interface Servlet.


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

public class SimpleServlet extends GenericServlet {
   public void service(ServletRequest req,
          ServletResponse res)
          throws ServletException, IOException {

       ServletOutputStream out = res.getOutputStream();
       out.println("<HEAD><TITLE> SimpleServlet Output ");
       out.println("</TITLE></HEAD><BODY>");
       out.println("<H1> SimpleServlet Output </H1>");
       out.println("</BODY>");
   }
}
```

compilation (attention à la variable CLASSPATH)

javac SimpleServlet.java

déploiement

Sous webapps dans CATALINA_HOME, on a créé par exemple un contexte Essai :

Essai\servlets va contenir le fichier index.html:

```
<HTML>
<HEAD>
<TITLE> Essai </TITLE>
</HEAD>
<BODY>
<a href="../servlet/SimpleServlet">Execute</a>
</BODY>
</HTML>
```


Essai\WEB-INF va contenir:

Le répertoire :

Essai\WEB-INF\classes

nous déposerons SimpleServlet.class dans le répertoire classes.

Le fichier web.xml

à propos des fichiers server.xml et web.xml

le fichier server.xml est résidant dans le répertoire conf de Tomcat. Il n'est pas accessible dans la configuration UdM. C'est dans ce fichier que les contextes doivent y figurer.

Une application web est reliée à un prefixe URI (Uniform Resource Identifier) (ou URL) spécifique, pour qu'une application serveur puisse la reconnaître et envoyer toutes les requêtes à cette adresse URI. Pour déployer une nouvelle application web, il faut définir un nouveau contexte qu'il faudra ajouter à l'application server. Cette application serveur est représentée par le fichier server.xml

La syntaxe à utiliser est comme suit :

```
<tomcat home>/conf/server.xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<Server>
...
<ContextManager debug="0" workDir="work" >
...
```



```
   <Context path="/examples"
      docBase="webapps/examples"
      debug="0"
      reloadable="true" />
   </ContextManager>
</Server>
```

path

Le chemin du contexte de l'application (le préfixe URI vers l'application web)
attribut obligatoire
Doit commencer par un /

docBase

C'est le répertoire root de l'application web

attribut obligatoire
Peut utiliser un chemin relative ou absolu

reloadable
Si vraie, Tomcat doit recharger la servelt en cas de mise à jour de cette dernière.
Si non avoir recours à la séquence arrête-démarrage du serveur.
Par défaut la valeur est false: false

debug
le niveau du débogage utilisé pour enregistrer des messages de **0** à **9**
Par défaut: 0

web.xml

Un minimum web.xml:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD
Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
</web-app>
```

```
http://localhost:8080/Essai/servlet/SimpleServlet
http://localhost:8080 protocol, host, et port
/Essai le contexte
/servlet/ le préfixe par défaut pour les servlets
SimpleServlet le nom de la classe de la servlet
```

Sinon que peut-il contenir …

```
<web-app>
   <!-- General description of your Web application -->
   <display-name>
      IFT1176 - Servlets
   </display-name>

   <description>
      Ceci est un exemple sur le contenu du fichier web.xml
       à partir de l'exemple de la servelt SimpleServlet.
   </description>

   <!-- Servlet definitions -->
   <servlet>
      <servlet-name>SimpleServlet</servlet-name>

      <description>
         Un affichage d'un message de bienvenue en sortie
      </description>
```



```
      <servlet-class>
         SimpleServlet
      </servlet-class>
   </servlet>


   <!-- Servlet mappings -->
   <servlet-mapping>
      <servlet-name>SimpleServlet</servlet-name>
      <url-pattern>/test</url-pattern>
   </servlet-mapping>


</web-app>
```

http://localhost:8080/Essai/servlet/SimpleServlet
http://localhost:8080/Essai/test
http://localhost:8080/Essai/servlets/index.html

Puisque nous avons affaire à des applications web, Il est profitable d'utiliser HttpServelt.

HttpServelt

C'est une classe permettant donc de traiter des requêtes du type HTTP.

La classe qui hérite de HttpServelt doit redéfinir au moins l'une des méthodes suivantes:

doGet, pour les requêtes HTTP GET (récupérer des informations du serveur, un fichier html, une image etc.)

doPost, pour les requêtes HTTP POST (envoyer des informations au serveur, des données par exemple etc.)

doPut, pour les requêtes HTTP PUT (déposer un fichier sur un serveur)

doDelete, pour les requêtes HTTP DELETE (effacer un fichier du serveur)

init et destroy, (étapes appelées lors du chargement et resp. lors de la destruction de la servelt)

getServletInfo (des informations sur la servlet: auteur, version etc.)

Pas besoin de surdéfinir la méthode service.

service gère les requêtes standards du protocole HTTP.
Une requête arrive au serveur, la méthode service détermine sa nature (Post, Get etc.), puis appelle la méthode appropriée pour gérer cette requête (doPost, doGet etc.).

Squelette d'une servlette HTTP-Get

```
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleServlet extends HttpServlet {

   public void init(HttpServletConfig c)
                  throws ServletException {…}
   public void doGet(HttpServletRequest req,
                        HttpServletResponse res)
            throws ServletException, IOException {…}
```



```
   public void destroy() {…}
   public String getServletInfo() {…}
}
```

Squelette d'une servlette HTTP-Put

```
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleServlet extends HttpServlet {

   public void init(HttpServletConfig c)
                  throws ServletException {…}
   public void doPut(HttpServletRequest req,
                        HttpServletResponse res)
            throws ServletException, IOException {…}
   public void destroy() {…}
   public String getServletInfo() {…}
}
```

Utiliser les objets **HttpServletRequest** et **HttpServletResponse** passés en paramètres de ces méthodes pour implémenter le service :

HttpServletRequest contient les renseignements sur le formulaire HTML initial (utile pour **doPost()**) :

- la méthode **getParameter()** récupère les paramètres d'entrée.
- la méthode **getParameterName()** récupère les noms des paramètres d'entrée envoyés avec la méthode Post.
- la méthode **getHeaderNames()** récupère les paramètres se trouvant dans l'entête.

HttpServletResponse contient le flux de sortie pour la génération de la page HTML résultat (**getWriter()**).

Dans le cas d'un envoi binaire, on utilisera :

ServletOutputStream getOutputStream() ;

Pour fixer le type de l'encodage que le navigateur devra utiliser, on utilisera la méthode :

void setContentType (String type) ;

Code Source pour RequestHeader

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

public class RequestHeaderExample extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
       throws IOException, ServletException {
         response.setContentType("text/html");
         PrintWriter out = response.getWriter();
         Enumeration e = request.getHeaderNames();
         while (e.hasMoreElements()) {
             String name = (String)e.nextElement();
             String value = request.getHeader(name);
             out.println(name + " = " + value);
         }
    }
}
```


Code source pour Request Parameter

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

public class RequestParamExample extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
    throws IOException, ServletException


   {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("GET Request. No Form Data Posted");
   }
```

```
    public void doPost(HttpServletRequest request,
                       HttpServletResponse res)
     throws IOException, ServletException
     {
         Enumeration e = request.getParameterNames();
         PrintWriter out = res.getWriter ();
         while (e.hasMoreElements()) {
             String name = (String)e.nextElement();
             String value = request.getParameter(name);
             out.println(name + " = " + value);
         }
     }
}
```


Fichier HTML accompagnant la requête :

```
<h3>Request Parameters Example</h3>
Parameters in this request:<br>
No Parameters, Please enter some
<P>
<form action="RequestParamExample" method=POST>
First Name:
<input type=text size=20 name=firstname>
<br>
Last Name:
<input type=text size=20 name=lastname>
<br>
<input type=submit>
</form>
```

On peut soumettre aussi la requête sous la forme suivante :

http://localhost:8080/examples/servlet/RequestParamExample?firstname=toto&lastname=titi

La méthode init()

void init()
Redéfinir celle ci pour initialiser la servelt.

void init(ServletConfig config)
Pas celle ci, sinon appel obligatoire à super.init(config)

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloHTML extends HttpServlet {
   protected String bgcolor;
   protected String fgcolor;
```



```
   public void init() {
      bgcolor = getInitParameter("bgcolor");
      fgcolor = getInitParameter("fgcolor");
   }

   <doGet() method>
}
```

Suivi de sessions en Java

Une session est une série de dialogues entre un client et un serveur pour accomplir certaines tâches.

HTTP protocole fournit très peu d'informations sur le suivi de sessions, car chaque requête est traitée séparément (non apparentées)

Plusieurs applications nécessitent d'avoir l'état entre deux requêtes:

e-commerce,
publicité,

Éviter la saisie d'informations à répétition login, password, adresse, téléphone…
Gérer des « préférences utilisateur »

Les suivis de sessions est un mécanisme que le serveur utilise pour garder un historique ou cumuler certaines informations sur les actions réalisées par les clients sur ce serveur.

Les servlets dépendent de quelques extensions pour réaliser cela, comme :

Cookies
Sessions
Databases

Le serveur assigne un identificateur unique pour chaque client, appelé sessionID. Le client va s'identifier avec cet identificateur pour chaque requête.

Il y a plusieurs mécanismes pour envoyer cet identificateur au client.

À partir de l'url :

Nous allons inclure l'identificateur du client dans l'URL :
http://www.xyz.com/catalog.html;jsessionid=1234

Un cookie :

```
response.addCookie(new Cookie("jsessionid", "1234"));
```


Un champ caché :

Le serveur ajoute des informations supplémentaires dans la page web retournée au client, comme :

```
<input type="hidden" name="jsessionid" value="1234"/>
```

HttpSession :

Java offre une API pour le suivi de sessions représentées par la classe HttpSession.

Cookies

Cookies sont une petite quantité de données que le serveur envoie au client.

Le client renvoie le cookie au serveur à chaque requête émise par ce dernier.

Les cookies sont sous la forme de paire (nom, valeur) et sont stockées sur la machine du client.

Le client retourne ces données au serveur quand il se connecte au même site web, du même domaine à la même page URL.

Les cookies peuvent être désactivés via le navigateur. Le problème est plutôt relié à la vie privée qu'à la sécurité (!). En effet le serveur peut garder un historique sur toutes les connections, et partager cette information avec une tierce personne.

Une mauvaise utilisation des cookies (sauvegarder par exemple des données sensibles) peut constituer un réel risque. Un simple problème de sécurité dans un navigateur et vos cookies se retrouveront dans la nature.
Ne pas donc trop dépendre des cookies et ne pas mettre des informations trop sensibles dans des cookies. Par exemple en désactivant l'option des cookies de votre navigateur.

Le nombre de cookies peut-être limité à 20 par site.

Le nombre total à 300 cookies.
La taille du cookie peut-être elle aussi limité à 4ko.

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);

Code Source pour Cookie

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

public class CookieExample extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // print out cookies

        Cookie[] cookies = request.getCookies();
        for (int i = 0; i < cookies.length; i++) {
            Cookie c = cookies[i];
```

```
            String name = c.getName();
            String value = c.getValue();
            out.println(name + " = " + value);
        }

        // set a cookie

        String name = request.getParameter("cookieName");
        if (name != null && name.length() > 0) {
            String value =
                     request.getParameter("cookieValue");
            Cookie c = new Cookie(name, value);
            response.addCookie(c);
        }
    }
}
```


HttpSession

Avec HttpSession, une session est établie pour une durée préfixée, indépendamment du nombre de requêtes établies ou bien du nombre de pages sollicitées.

API

Dans la classe HttpServletRequest:

HttpSession getSession()
HttpSession getSession(boolean create)

Retourne la session HttpSession courante associée à cette requête ou bien une nouvelle session, dans le cas ou il n'y a pas de session courante et create est true

Par défaut: create est true

String getRequestedSessionId()
Retournes le ID de la session.

Dans la classe HttpServletResponse:

String encodeURL(String url)
code l'URL spécifiée en argument en lui incluant le ID de la session.

si l'encodage n'est pas nécessaire, elle se contente de retourner l'URL sans y introduire de modification.

Dans la classe HttpSession:
void setAttribute(String name, Object value)
Associé un objet à un nom spécifié en argument, pour cette session.

Object getAttribute(String name)
Retourne l'objet associé au nom spécifié en argument, ou bien null si aucun nom n'a été associé à l'objet.

void removeAttribute(String name)
Retire l'association de l'objet avec le nom spécifié en argument.

String getId()

Retourne le ID de la session.

Enumeration getAttributeNames()

Retourne un énumérateur d'objets string contenant les noms de tous les objets associés à cette session.

long getCreationTime()

Retourne la date de création de la session.

long getLastAccessedTime()

Retourne la date de la dernière requête envoyée par le client, associée à cette session.

Code Source pour Session

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

public class SessionExample extends HttpServlet {

    public void doGet(HttpServletRequest request,
                              HttpServletResponse response)
                throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        HttpSession session = request.getSession(true);

        // print session info
        Date created = new Date(session.getCreationTime());
        Date accessed =
```

```
                new Date(session.getLastAccessedTime());
        out.println("ID " + session.getId());
        out.println("Created: " + created);
        out.println("Last Accessed: " + accessed);
        // set session info if needed
        String dataName = request.getParameter("dataName");
        if (dataName != null && dataName.length() > 0) {
            String dataValue =
                        request.getParameter("dataValue");
            session.setAttribute(dataName, dataValue);
        }
        // print session contents
        Enumeration e = session.getAttributeNames();
        while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            String value =
                     session.getAttribute(name).toString();
            out.println(name + " = " + value);
        }
    }
}
```