TP3 - Servlets

Un peu de vocabulaire ... et de technique

Servlet ... keskecé ?

Une servlet est un composant logiciel, utilisé dans un serveur web, qui peut être invoqué par les navigateurs clients au travers d'une URL. Le protocole de communication est dans ce cas HTTP. Même si le *web dynamique* est l'utilisation majeure de l'API Servlet, elle permet théoriquement de couvrir d'autres domaines d'application.

Le principe de fonctionnement est très simple : ce composant logiciel reçoit une requête, et il envoie une réponse. Cette réponse est transmise au client, qui l'interprète enfin.

Techniquement, l'API Servlet est un ensemble d'interfaces et de classes Java, rangées dans les packages *javax.servlet* et *javax.servlet.http*.

Dans la mesure où les servlets sont des composants programmés entièrement en Java, elles sont portables sur toutes les architectures munies d'une machine Java.

L'API Servlet a subi de nombreuses évolutions et améliorations depuis qu'elle est apparue. La première de ces évolutions est maintenant placée dans l'API JSP (Java Server Pages).

Web container ... et ça, kessdonc ?

Un *web container*, ou *servlet container*, est un logiciel qui est en capacité d'assurer l'exécution de *servlets*. Ces logiciels sont parfois appelés *moteur web* ou *moteur de servlets*.

Web Application Archive ... encore un nouveau format ?

Un fichier WAR (pour Web Application Archive) est un fichier JAR organisé pour contenir un ensemble de *JavaServer Pages*, de *servlets*, de classes Java, de fichiers XML, de pages Web statiques (et fichiers joints), le tout constituant une application web.

Ces fichiers doivent obligatoirement contenir certains répertoires et fichiers ...

Le répertoire racine d'une application web (celui dans lequel seront déposées toutes les ressources de l'application) est appelé **document root** de l'application. Il correspond à l'attribut **docBase** d'un **Context**, dans la configuration du serveur.

Dans le **document root**, on doit obligatoirement trouver un répertoire nommé **WEB-INF**. Ce répertoire doit en contenir deux autres, **WEB-INF/classes** et **WEB-INF/lib**. Ceux-ci contiendront respectivement les classes et les librairies accessibles uniquement à cette application web. Ce répertoire ne doit pas nécessairement contenir que des *servlets*. Toutes les classes Java utilisées par l'application doivent s'y trouver. Attention toutefois à respecter les *packages* Java. Si la classe *PremiereServlet* était

située dans le package *org.test.servlet*, la classe compilée devrait se trouver dans *WEB-INF/classes/org/test/servlet*.

Le **document root** doit également contenir un fichier **web.xml**, appelé descripteur de déploiement de l'application web. Il contient les différentes caractéristiques et paramètres de l'application, notamment la description des *servlets* ou des paramètres d'initialisation.

Le format WAR ... pour livrer ...

La plupart des applications J2EE sont livrées dans un fichier **WAR** et sont destinées à être déployées par une application de type *web container*. En pratique, il suffit de déposer le fichier *war* dans un répertoire spécifique du *web container* pour qu'un déploiement soit réalisé (soit au lancement du *web* container, soit automatiquement pendant son exécution, le *web container* vérifiant régulièrement la disponibilité de nouveaux fichiers *war* ou la mise à jour de fichiers déjà présents).

La plupart des *web containers* gèrent un répertoire nommé *webapps* contenant l'ensemble des applications déployées.

L'installation d'une application sur un *web container* respecte donc généralement le déroulé suivant :

- Un fichier *war* caractérisant une application est déposé dans le répertoire *webapps* du serveur

- Le contenu du fichier *war* (l'équivalent d'un fichier *zip*) est décompressé dans un sous répertoire du répertoire *webapps*, dont le nom sera généralement le nom du fichier *war* sans suffixe (le dépôt du fichier *MonAppli.war* dans le répertoire *webapps* va par exemple permettre la création d'un sous répertoire *MonAppli* contenant l'arborescence complète des fichiers contenus dans *MonAppli.war*).

Une fois déployée, comment accéder à une application ?

Une URL permet d'accéder aux ressources statiques et dynamiques d'une application web. Par exemple, une application contenue dans le sous-répertoire MonAppli du répertoire *webapps* sera accessible à travers une URL de la forme ...

```
http://localhost/MonAppli
```

Pour une ressource statique (une page HTML), il suffit de préciser dans l'URL le chemin d'accès à la ressource dans la *WebApp*, ainsi que son nom. Par exemple ...

```
http://localhost/MonAppli/rubrique1/page2.html
```

... permet d'accéder à la page HTML caractérisée par le fichier *page2.html* et rangée dans le sous-répertoire *rubrique1* de l'application *MonAppli*.

Pour accéder à une *servlet*, ressource dynamique, le chemin et la ressource utilisées dans l'URL doivent correspondre au *mapping* qui est décrit dans le fichier de configuration *web.xml* et qui définit un lien entre classe Java et URL.

Web.xml ... keskecé ?

Le fichier web.xml est essentiel au fonctionnement d'une *web application*. Appelé *descripteur de déploiement*, ce fichier est à installer dans le répertoire WEB-INF de l'application. Il contient les caractéristiques et

paramètres de l'application. Cela inclut le description des *servlets* utilisés, ou les différents paramètres d'initialisation.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="M4201C"
version="2.5">

   <display-name>M4102C</display-name>

   <description>Cours M4102C - François MERCIOL</description>

   <servlet>
      <servlet-name>ServletAction</servlet-name>
      <servlet-class>com.iut.servlets.Action</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>ServletAction</servlet-name>
      <url-pattern>/action</url-pattern>
   </servlet-mapping>

</web-app>
```

Deux balises relèvent de la description de l'application. La première, **display-name** permet de donner un nom à l'application. Certains *web containers* utilisent cet identifiant pour reconnaître l'URL utilisé pour l'accès aux ressources de l'application (et ignorent donc le nom du sous-répertoire de déploiement). La balise **description** permet de fournir une description plus détaillée de l'application, tout en étant techniquement inopérante.

Les balises *servlet* permettent de décrire les *servlets* mis à disposition de l'application. Ces *servlets* sont potentiellement accessibles par les clients, au travers d'une URL. Ainsi, la balise *servlet* permet d'affecter un nom à une *servlet* et de préciser la classe Java qui en assurera la gestion (classe Java dérivée de *HttpServlet*), et la balise *servlet-mapping* permet d'indiquer au serveur quelle servlet charger pour telle requête du client (telle URL demandée) – les URL des servlets sont relatives à l'URL du *Context* (la *webapp*) auquel elles appartiennent.
Aussi, en utilisant la configuration ci-dessus, l'appel de la *servlet* nommée *ServletAction* sera par exemple possible par une utilisation de l'URL *http://localhost/M4102C/action*.

Mettre en œuvre une *servlet ...*

Les fonctionnalités d'une *servlet* sont donc gérées par une classe Java surchargeant *HttpServlet*. Pour gérer une requête http de type *GET* en provenance d'un client, une *servlet* devra donc fournir le code de la méthode *doGet* de sa super classe.

```
   public void doGet(HttpServletRequest req, HttpServletResponse resp)
{
      /*
      analyse de la requête

      réalisation des traitements adéquats

      définition de la réponse fournie

      */
   }
```

D'autres méthodes de la classe *HttpServlet* peuvent être surchargées pour gérer différents types de requêtes HTTP :

- doPost() : pour les requêtes http de type POST

- 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

doGet ... komenkonfé ?

Les opérations réalisées dans une méthode *doGet* sont généralement ...

- La lecture des données associées à la requête (les paramètres de la requête)
- La génération des *headers* de la réponse
- La récupération d'une instance de *OutputStream* ou de *PrintWriter* pour permettre l'émission des données de la réponse
- L'émission des données de la réponse

Tapahinexampe ?

Ecrivons une simple *servlet* retournant à l'appelant un texte (avec quelques balises HTML) fournissant l'heure courante ...

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

public class HorlogeServlet extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException {
      SimpleDateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
      PrintWriter pw;

      response.setContentType("text/html");
      pw=response.getWriter();
     pw.println("<h1>"+ dateFormat.format(new Date())+"<h2>");
   }
}
```

Allez ... au boulot !

Tiny Java Web Server (*tjws*) est une application de type *servlet container* portable, permettant d'exploiter des *servlets.* Et comme toute application de ce type, *tjws* est en capacité de déployer des applications distribuées sous la forme de fichiers respectant le standard WAR.

Pour installer et exécuter *tjws* ...

- Téléchargez l'archive d'installation
- Décompressez cette archive (un répertoire Tjws est créé)
- Dans le répertoire *Tjws*, éditez le fichier *tjws.sh* et complétez la déclaration de la variable d'environnement JDK_HOME (ou supprimez la si votre environnement la défini déjà)
- Exécutez *./tjws.sh*

Par défaut, *Tjws* écoute sur le port TCP 8080. Vérifiez son bon fonctionnement en utilisant l'URL *http://127.0.0.1:8080* sur un navigateur.

Pour stopper le serveur *Tjws*, tapez ***q***, puis validez.

Yapluka !

Installez sur votre serveur *Tjws* une application proposant une page statique (une page de bienvenue, intitulée par exemple *index.html*) et une *servlet* permettant de retourner la racine carrée d'un nombre entier passé en paramètre.

La *servlet* devra être en capacité d'interpréter une URL de type ...

```
http://127.0.0.1:8080/MonAppli/racine?entier=45
```

... et de fournir une réponse texte de type ...

```
la racine carrée de 45 est 6,7
```

Les sessions ... keskecéhenkor ?

Une application *web* est basée sur le protocole http, qui est un protocole dit « sans état ». Cela signifie que le serveur, une fois qu'il a envoyé une réponse à la requête formulée par un client, ne conserve pas les données le concernant. Autrement dit, le serveur traite chaque requête qui lui parvient dans le même environnement vide d'historique ... pour lui, chaque nouvelle requête provient d'un nouveau client.

Pour pallier cette lacune, qui peut être très gênante dans certaines situations, le concept de session a été créé ... il permet au serveur de mémoriser des informations relatives à un client, entre deux requêtes émises par ce client.

La session représente un espace mémoire alloué pour chaque utilisateur et identifié par une valeur (un identifiant) générée par les serveur, valeur transportée dans un *cookie* http dont le nom est *JSESSIONID*.

La méthode *getSession()* de la classe *HttpServletRequest* permet d'obtenir une instance de *HttpSession* caractérisant la session associée à l'utilisateur courant. La méthode *getID()* de *HttpSession* permet d'obtenir l'identifiant associé à la session.

Le contenu d'une session (une instance de *HttpSession*) est conservée par le serveur tant qu'un utilisateur n'est pas resté inactif trop longtemps ou que la session n'a pas été désactivée volontairement par la *servlet*.

Les méhodes *setAttribute* et *getAttribute* de *HttpSession* permettent la mise en place d'instances d'objets au sein d'une session et leur récupération.

Inkapratik ?

Modifiez la *servlet* que vous avez développé précédemment pour faire en sorte qu'elle numérote les réponses fournies dans la cadre d'une même session utilisateur, et génère des réponses du type ...

```
Calcul n° 12 : la racine carrée de 45 est 6,7
```

Kestionpratiks ...

- Précisez, dans le *classpath* Java l'utilisation de la librairie *Tjws/lib/servlet.jar*

- Organisez votre espace de travail. Par exemple, à partir du répertoire TP3 ...

```
TP3
|----src
|     |----com
|     |----iut
|            |----servlets
|                   |---- MonServlet.java
|
|----WEB-INF
|     |----classes
|     |----lib
|     |----web.xml
|
|----index.html
```

- Dans le répertoire TP3, générez un script de compilation intégrant une commande de type ...

```
java -cp ../Tjws/lib/servlet.jar -d WEB-INF/classes
src/com/iut/servlets/*
```

- Toujours dans ce répertoire, écrivez un script de génération du fichier WAR permettant de déployer l'application sur votre *web container* Tjws. Un fichier WAR respectant le format JAR, utilisez la commande *jar*.

Exemple :

```
jar cf MonAppli.war WEB-INF
jar uf MonAppli.war index.html
```