Applications Web et servlets Java

Application Web

Une application Web répartie sur trois couches (*three-tier Web application*) est composée selon l'architecture suivante :

- **Client mince :** réalisé par un fureteur Web, le client mince est l'interface utilisateur (GUI) qui s'occupe de la présentation et la saisie des données. Aucun traitement des données n'est fait dans cette couche.
- **Serveur application :** s'occupe de traiter les requêtes provenant du client à travers le réseau et sert d'interface avec la base de données. La logique de contrôle de l'application réside dans cette couche.
- **Serveur BD :** normalement situé sur une machine différente au serveur application, le serveur BD s'occupe du stockage et de la gestion des données. La base de données (Oracle, MySQL, PostgreSQL) réside dans cette couche et communique au serveur application à travers le réseau.

Puisque le serveur BD (`disjkstra.logti.etsmtl.ca`) contient déjà une base de données créée au premier laboratoire, et comme plusieurs fureteurs Web sont disponibles, votre travail consistera essentiellement à implémenter le serveur application. La technologie qui sera utilisée pour ce serveur est celle des *Servlets* Java.

Servlets Java

Un servlet est une application Java qui permet de créer dynamiquement des données au sein d'un serveur HTTP. Ces données sont le plus souvent présentées sous le format HTML, mais elles peuvent également l'être en format XML ou tout autre format destiné aux fureteurs Web. Tel qu'illustré à la Figure 1, le processus d'invocation d'un servlet, dans une application à trois couches, comporte 7 étapes :

1. Le fureteur se connecte à l'adresse du site Web de l'application et lui envoie une requête `HTTP GET` demandant de produire une page HTTP associée à cette adresse.
2. Le serveur Web (ex : *Tomcat*) retourne au fureteur une page HTML comportant une saisie de données (FORM HTML). Cette page Web précise que les paramètres entrés par l'utilisateur seront traitées par un servlet.
3. L'utilisateur entre l'information requise par la page Web qui est ensuite envoyée par le fureteur au serveur Web sous la forme d'une requête `HTTP POST`.

4. Le serveur Web délègue la requête au conteneur de servlet qui...
5. ...démarre un fil du servlet correspondant.
6. Le servlet décode les paramètres provenant du client, établit une connexion au serveur BD et traite la requête à l'aide des données de la BD.
7. Le servlet produit une réponse à la requête sous la forme d'une page HTML qui est retournée au client.

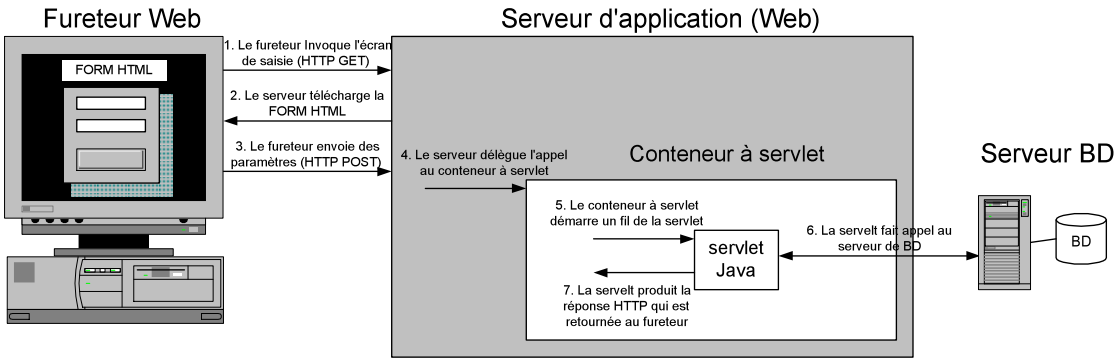

FIGURE 1 – Invocation d'un servlet Java (©*Robert Godin*)

Pour développer une servlet HTTP, il faut créer une classe Java spécialisant la classe `javax.servlet.http.HttpServlet` et redéfinir les méthodes `doGet` et `doPost` de la classe `HttpServlet`. Ces méthodes, produisant toutes les deux une page HTML en réponse à une requête HTTP, diffèrent par la manière avec laquelle les paramètres sont transmis au serveur. Ainsi, dans la méthode `doGet`, les paramètres sont envoyés directement dans l'URL, sans encryption. Cette méthode est souvent utilisée pour demander une information au serveur. En revanche, la méthode `doPost` insère les paramètres de façon encryptée dans le corps de la requête, et est utilisée pour mettre à jour l'information du serveur. Dans le cadre de ce laboratoire, nous ne ferons pas la distinction entre ces deux types de requête et aurons une même implémentation pour les méthodes `doGet` et `doPost`.

Exemple de servlet

Voici un exemple de servlet Java pour obtenir la liste des films dont le titre renferme une certaine chaîne de caractères :

```
// Classe Java: ServletWebflix.java
package webflix;

import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.StringWriter;
import java.sql.Connection;
import javax.servlet.*;
import oracle.jdbc.pool.*;
import javax.naming.Context;
```

```
import javax.naming.InitialContext;

public class ServletWebflix extends HttpServlet {

    // Initialisation du parent
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
    }

    // Methode doGet: on utilise l'implementation de doPost
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException,
                                                          IOException {
        doPost(request,response);
    }

    // Traitement de la requete:
    // On recoit une chaine de charactere et on retourne un page HTML
    // contenant le titre des films de la BD contenant cette chaine
    // dans leur titre.
    public void doPost(HttpServletRequest requete,
                       HttpServletResponse reponse) throws ServletException, IOException {

     // Specifier le type et l'encodage des donnees
     reponse.setContentType("text/html");
     reponse.setCharacterEncoding("utf-8");

     // Creer un PrintWriter pour imprimer la page Web de la reponse
     OutputStreamWriter osw = new OutputStreamWriter(reponse.getOutputStream());
     PrintWriter out = new PrintWriter(osw);

     // Entete de la page
     out.println("<html>");
     out.println("<head><title>Reponse du Servlet Webflix</title></head>");
     out.println("<body>");

     String chaineRecherche = "";
     Connection conn = null;
     PreparedStatement ps = null;

     try {
        // Recuperer le parametre provenant de la page HTML d'entree
        chaineRecherche = requete.getParameter("chaineRecherche");

        if (chaineRecherche != null)
        {
           // Ouvrir une connexion en passant par un DataSource
           Context initContext = new InitialContext();
           Context envContext = (Context) initContext.lookup("java:/comp/env");
           OracleDataSource ds = (OracleDataSource) envContext.lookup("jdbc/webflix");
           conn = ds.getConnection();

           // Creer une requete au serveur BD
           ps = conn.prepareStatement(
```

```
                "SELECT * " +
                "FROM Film " +
                "WHERE titre like ?");

            ps.setString(1,"%" + chaineRecherche + "%");

            // Decoder les resultats
            ResultSet rs = ps.executeQuery();

            while (rs.next())
               out.println(rs.getString("titre") + " (" + rs.getInt("annee") + ")<br>");
         }
      }
      catch (Exception e) {
         // Debug: afficher la trace d'erreur directement dans la page
         StringWriter sw = new StringWriter();
         PrintWriter pw = new PrintWriter(sw);
         e.printStackTrace(pw);
         out.println("<p>" + sw.string() + "</p>");
      }
      finally{
         try{
            // Liberer les connections et resources
            out.println("</body></html>");
            out.close();
            ps.close();
            conn.close();
         }
         catch(Exception lException){
            lException.printStackTrace();
         }
      }
   }
}
```

Tel qu'illustré dans cet exemple, le traitement d'une requête par un servlet, dans la méthode `doPost` comporte plusieurs étapes :

1. Configurer le type et l'encodage des données de la réponse. Dans notre cas, les données sont de type HTML et l'encodage utilisé est celui de la base de données (UTF-8).
2. Créer un `PrintWriter` afin d'imprimer la page HTML de la réponse.
3. Récupérer les paramètres de la requête à l'aide de la fonction `getParameter`.
4. Créer une connexion à la base données. Lors du laboratoire 1, les paramètres de la connexion (URL, ID et mot de passe pour la BD) étaient entrés directement dans le code Java. Cette approche comporte plusieurs problèmes, principalement au niveau de la sécurité et de la flexibilité. Ainsi, si l'adresse du serveur BD ou l'ID/mot de passe du compte de la BD change, il faut modifier et redéployer le servlet. Dans ce laboratoire, nous utiliserons une meilleure approche basée sur les `DataSource`. Cette approche sera expliquée plus en détails dans l'énoncé.
5. Une fois la connexion à la BD établit, utiliser un ou plusieurs `PreparedStatement` pour faire une requête ou mettre à jour les données de la BD.

6. Imprimer sous la forme d'une page HTML la réponse à la requête du client à l'aide du `PrintWriter`.
7. Dans `tous les cas`, libérer les resources en fermant le `PrintWriter`, le `PreparedStatement` ainsi que la connexion à la BD.

Pour saisir et envoyer les données au servlet, il est nécessaire d'avoir une page HTML faisant référence au servlet :

```
<!-- Page HTML FormWebflixServlet.html -->
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Recherche de films dans la BD Webflix</title>
</head>
<body>
<form action = "servlet-webflix" method = "post">
<p>Entrer une chaîne de caractères contenue dans le titre du film</p>
<table>
  <tr>
    <td>Chaîne de recherche: </td>
    <td><input type = "text" name = "chaineRecherche" /></td>
  </tr>
</table>
<input type = "submit" value = "Soumettre" />
</form>
</body>
</html>
```

À noter les éléments `action="servlet-webflix"` et `method="post"` de la balise `form`. Le premier indique que le servlet qui recevra les données est situé à l'URL (virtuelle) :

`<adresse application Web>/servlet-webflix`.

Nous verrons plus loin comment associer cette adresse virtuelle avec une classe Java (ex : `ServletWebflix.class`). Le second élément spécifie que la requête sera envoyée par un `POST HTTP` et que la méthode `doPost` du servlet sera appelée.

Gestion de session dans les servlets

Dans une application Web l'interaction entre l'utilisateur et l'application se fait souvent à travers différentes pages. Afin de transmettre les données entrées dans une page (servlet) à une autre page (servlet), dans une même session, on doit utiliser le mécanisme de `HttpSession`. Prenons le cas d'une application Web qui demande d'entrer deux nombres, dans des pages différentes, et retourne une page HTML affichant la somme de ces deux nombres. Dans un premier temps, il faut une page HTML initiale demandant d'entrer le premier nombre :

```
<!-- Page HTML FormSommeServlet.html -->
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Somme de deux nombres</title>
```

```
</head>
<body>
<form action = "somme1" method = "post">
<table>
  <tr>
    <td>Entrer un premier nombre: </td>
    <td><input type = "text" name = "premierNombre" /></td>
  </tr>
</table>
<input type = "submit" value = "Soumettre" />
</form>
</body>
</html>
```

Ensuite, dans le premier servlet, il faut récupérer le paramètre envoyé par cette page, le sauver dans un `HttpSession`, et imprimer dans la réponse une page Web demander

```
// Classe Java: SommeServlet1.java
...
public void doPost(HttpServletRequest requete,
                   HttpServletResponse reponse) throws ServletException, IOException {
  ...
  // Creer une session
  HttpSession session = requete.getSession(true);

  try {
    // Recuperer le premier nombre
    String premierNombre = requete.getParameter("premierNombre");

    // Sauver le premier parametre dans la session
    session.setAttribute("premierNombre", premierNombre);

    // Afficher la seconde page
    out.println("<html>");
    out.println("<head><title>Somme de deux nombres (2)</title></head>");
    out.println("<body>");
    out.println("<form action = \"somme2\" method = \"post\">");
    out.println("<table><tr>");
    out.println("<td>Entrer un deuxième nombre: </td>");
    out.println("<td><input type = \"text\" name = \"deuxiemeNombre\" /></td>");
    out.println("</tr></table>");
    out.println("<input type = \"submit\" value = \"Soumettre\" />");
    out.println("</form></body></html>");
  }
  ...
}
```

Puisque que la session est créée dans ce servlet, il faut donner le valeur `true` au paramètre de la méthode `getSession`. Lorsque l'on veut récupérer une session déjà créée, on emploie plutôt la valeur `false`. Ensuite, il faut créer un second servlet qui va récupérer les deux nombre et produire une page HTML donnant leur somme :

```
// Classe Java: SommeServlet2.java
...
```

```
public void doPost(HttpServletRequest requete,
                   HttpServletResponse reponse) throws ServletException, IOException {
  ...
  // Creer une session
  HttpSession session = requete.getSession(false);

  try {
    // Recuperer le premier nombre
    String premierNombre = session.getAttribute("premierNombre");

    // Recuperer le second nombre
    String deuxiemeNombre = requete.getParameter("deuxiemeNombre");

    // Afficher la page contenant la somme
    out.println("<html>");
    out.println("<head><title>Somme de deux nombres (3)</title></head>");
    out.println("<body>");
    out.println("<p>La somme des deux nombres est: </p>");
    out.println(Integer.parseInt(premierNombre) + Integer.parseInt(deuxiemeNombre));
    out.println("</body></html>");
  }
  ...
}
```

Déploiement de l'application Web

Le déploiement de votre application Web sera faite sur la plateforme Apache Tomcat 6.0. Sous Tomcat, une application Web est structurée sous la forme d'une hiérarchie de répertoires, dont la racine porte le nom de l'application (`<mon_app>`), situés dans le répertoire Tomcat des applications Web (ex :`/var/lib/tomcat6/webapps/`) :

`<mon_app>/` : Contient les fichiers HTML, JSP, JavaScript, CSS, images, ainsi que tout autre fichier directement visible par le fureteur. Par exemple, c'est ici que l'on met le fichier `FormWebflixServlet.html` dans l'exemple précédent.

`<mon_app>/WEB-INF/web.xml` : Décrit les servlets de l'application, leur association avec une URL virtuelle, leurs paramètres, ainsi que les ressources utilisées par ces servlets. Par exemple. c'est dans ce fichier que la classe Java `ServletWebflix.class` est associée avec l'URL virtuelle `servlet-webflix` référencée dans le fichier `FormWebflixServlet.html`.

`<mon_app>/WEB-INF/classes/` : Contient tous les classes Java (fichiers `.class`) requis par l'application, incluant à la fois les classes des servlets et celles utilisées par ces servlets. À noter que l'emplacement des classes doit **respecter la position de la classe dans son paquetage Java**. Par exemple, comme la classe `ServletWebflix.java` renferme la ligne "`package webflix;`", le fichier `ServletWebflix.class` doit se trouver dans le sous-répertoire `<mon_app>/WEB-INF/classes/webflix/`.

`<mon_app>/WEB-INF/lib/`	: Contient toutes les libraires JAR requises par votre application. À noter que certaines JAR, comme celles renfermant les pilotes JDBC se trouvent déjà dans un répertoire partagé les rendant disponibles à toutes les applications Web dans Tomcat.
`<mon_app>/META-INF/context.xml`	: Définit les resources disponibles dans le contexte de l'application Web. C'est dans ce fichier que l'on définit les paramètres de la connexion JDBC à la BD Oracle.

La description du déploiement de l'application se fait dans le fichier `web.xml` situé dans le répertoire `<mon_app>/WEB-INF`. Pour chacun des servlets de l'application ce fichier doit avoir une balise

```
<servlet>
  <servlet-name> nom_servlet </servlet-name>
  <servlet-class> classe_servlet </servlet-class>
</servlet>
```

ainsi qu'une balise

```
<servlet-mapping>
  <servlet-name> nom_servlet </servlet-name>
  <url-pattern> url_servlet </url-pattern>
</servlet-mapping>
```

Ici, `nom_servlet` est un identifiant choisit pour le servlet, `classe_servlet` est la classe du servlet correspondant, incluant le préfixe du paquetage Java, et `url_servlet` est l'URL virtuelle associée au servlet, soit `url_serveur_tomcat/url_servlet`. Par ailleurs, ce fichier peut également définir certaines ressources utilisées par l'application, avec une balise :

```
<resource-ref>
  <description> texte_descriptif </description>
  <res-ref-name> nom_ressource </res-ref-name>
  <res-type> classe_ressource </res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

Voici un exemple de fichier pour le servlet décrit précédemment :

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                             http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5" xmlns="http://java.sun.com/xml/ns/javaee">
  <servlet>
    <servlet-name>ServletWebflix</servlet-name>
    <servlet-class>webflix.ServletWebflix</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServletWebflix</servlet-name>
    <url-pattern>/servlet-webflix</url-pattern>
  </servlet-mapping>
  <session-config>
      <session-timeout>30</session-timeout>
   </session-config>
   <mime-mapping>
```

```
      <extension>html</extension>
      <mime-type>text/html</mime-type>
   </mime-mapping>
   <mime-mapping>
      <extension>txt</extension>
      <mime-type>text/plain</mime-type>
   </mime-mapping>
   <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
      <welcome-file>index.html</welcome-file>
   </welcome-file-list>
   <resource-ref>
      <description>JDBC connexion datasource</description>
      <res-ref-name>jdbc/webflix</res-ref-name>
      <res-type>oracle.jdbc.pool.OracleDataSource</res-type>
      <res-auth>Container</res-auth>
   </resource-ref>
</web-app>
```

La ressource décrite dans cet exemple est une data source Oracle permettant de se connecter à la BD Webflix. Les paramètres de cette ressource doivent être définis dans le fichier `context.xml` :

```
<?xml version='1.0' encoding='utf-8'?>
<Context reloadable="true">
     <Resource name="jdbc/webflix"
                auth="Container"
                type="oracle.jdbc.pool.OracleDataSource"
                factory="oracle.jdbc.pool.OracleDataSourceFactory"
                maxActive="30"
                maxIdle="2"
                maxWait="1000"
                user="nomLogin"
                password="motDePasseLogin"
                driverClassName="oracle.jdbc.driver.OracleDriver"
                url="jdbc:oracle:thin:@dijkstra.logti.etsmtl.ca:1521:log660" />
</Context>
```

Évidemment, les valeurs des paramètres `user` et `password` doivent correspondre à ceux utilisés pour se connecter à la BD.

Références

- Documentation Tomcat6 : `http://tomcat.apache.org/tomcat-6.0-doc/index.html`