Web, Proxy et Cie

Par Jean-Marc QUERE

Le langage de l'internet, «java», n'a pas pour seul objet l'animation de nos pages HTML (même s'il y excelle) et sa présence sous forme d'applet n'est pas son unique forme d'utilisation. Ses qualités ont déjà été mise en évidence dans le cadre d'application autonome (couramment désignée sous le label «strong java») dans le cadre d'applications orientées base de données : JDBC, etc. Ses étonnantes facultés d'interropérabilités et autres traitements coopératifs ont également été abordés, notamment de façon connexe à Borland Delphi et dans le cadre d'un déploiement basé sur RMI. Un aspect plus commun n'a été que brièvement survolé dans le cadre d'un exemple : celui de l'implémentation de serveur de pages et de données. Je vous propose donc d'aborder «java» par l'autre bout de la «lorgnette» : celui des applications de services.

Mon Proxy à moi...

La réalisation proposée consiste en la mise en œuvre d'un serveur Proxy. Situé, dans le flux d'informations, entre le navigateur et le serveur de page, l'objet de ce dernier est d'assurer l'activation d'application java aux fins de compléter le contenu des pages délivrées. L'intérêt de ce type de solution est multiple. D'une part, elle permet la mise en œuvre de traitements dans notre langage favori sans imposer à l'utilisateur l'usage d'un navigateur particulier, d'autre part les traitements effectués sont totalement sécurisés. Le désassemblage du bytecode d'un éventuel applet offrant à un tiers la faculté de se connecter à vos bases de données est définitivement proscrit. Attention, même si le mécanisme implémenté s'approche du principe des servlets (applets coté serveur), il n'a pas la vocation de s'y substituer. Dans ce premier volet, le serveur Proxy est abordé ainsi que deux exemples d'applications. Le (ou les prochains) volet(s) seront principalement axés sur les différentes possibilités de mise en œuvre par l'étude de cas concrets.

Middle.java

```
import java.io.*;
import java.net.*;
import java.util.*;

class Middle extends Thread {
  private static int port;

  Socket browserSocket;
  Middle(Socket theSocket) {
    browserSocket=theSocket;
  }
```

La classe Middle (notre proxy) est un Thread dont une instance est créée pour chaque requête utilisateur. Pour simplifier, l'amorce de serveur est également incluse dans la classe d'où la présence d'élément statique, notamment le numéro du port employé (port).

```
  public void run() {
    BufferedReader fromBrowser=null;
    PrintWriter toBrowser=null;
```

La seule méthode employée (et implémentée) dans le cadre du service à assurer aux connectés est «run». Elle comporte le traitement à effectuer suite à l'activation du Thread par la méthode «start». Pour recevoir et émettre des informations en provenance et à l'attention du navigeur à l'origine de la requête un BufferedReader et un PrintWriter sont employés. Ils sont déclarés préalablement à leur emploi afin d'être visible au niveau du «catch» assurant la prise en charge des messages d'erreur. L'affectation de la valeur «null» évite l'erreur de compilation liée à la possibilité de leur utilisation sans initialisation préalable.

```
    try {
      fromBrowser=new BufferedReader(
        new InputStreamReader(browserSocket.getInputStream()));
      toBrowser=new PrintWriter(browserSocket.getOutputStream());
      String line,pageArgs="",providerArgs="";
```

La première opération du traitement consiste en l'initalisation du BufferedReader et du PrintWriter à partir des flux d'entrées (InputStream) et de sorties (OutputStream) du «socket» dédié au navigateur.

Les variables pageArgs et providerArgs sont déclarées, puis initialisées à «» (chaîne vide). Ces deux variables ont pour objet de mémoriser la ligne d'argument passée en paramètre à la page (lors de la requête «GET» sous la forme «GET http://<identification de la page>?<paramètres>») et celle utilisée pour l'application java.

```
     if ((line=fromBrowser.readLine())!=null) {
       StringTokenizer myStringTokenizer=new StringTokenizer(line);
       String theCommand=myStringTokenizer.nextToken();
       if (myStringTokenizer.hasMoreTokens() && theCommand.equals("GET")) {
```

La classe StringTokenizer est très utile. Elle permet le découpage de la chaîne reçue en sous-chaîne en cherchant le séparateur (en règle générale l'espace). A noter, qu'il est possible d'en changer avec la méthode «nextToken(String delim)». Si la requête est une commande «GET» et qu'une indication complémentaire est fournie alors il s'agit de l'url demandée (theRequestedUrl).

```
         String theRequestedUrl=myStringTokenizer.nextToken();
         if (theRequestedUrl.startsWith("/"))
           theRequestedUrl="http://"+
             browserSocket.getInetAddress().getHostAddress()+
             theRequestedUrl;
         if (theRequestedUrl.indexOf("?")>-1) {
           pageArgs=theRequestedUrl.substring(theRequestedUrl.indexOf("?")+1,
           theRequestedUrl.length());
           theRequestedUrl=theRequestedUrl.substring(0,
             theRequestedUrl.indexOf("?"));
         }
```

Concernant l'url, plusieurs opérations sont effectuées. Lorsque l'url commence par «/» alors l'appel a été effectué de façon directe par l'intermédiaire d'une indication «http://<adresse>/<page>:<port>» où l'indication <port> correspond à celui employé par le Proxy et que ce dernier fonctionne sur le même poste que le navigateur utilisé. L'url est alors complétée par l'adresse du poste concerné. Le plus souvent, l'emploi du Proxy est explicitement indiqué dans les paramètres de connexion du navigateur employé, l'adresse communiquée est alors toujours complète. L'url peut éventuellement être suivi d'un ensemble de paramètres destiné aux applications hébergées par la page demandée. Ils sont isolés, puis stockés dans la variable pageArgs.

```
         URL hostUrl=new URL(theRequestedUrl);
         if (theRequestedUrl.toUpperCase().endsWith(".MIDDLE")) {

           BufferedReader fromHost=
             new BufferedReader(new InputStreamReader(hostUrl.openStream()));
           toBrowser.println("HTTP/1.0 200 OK\nContent-Type:text/html\n");
```

Afin de ne pas avoir à analyser l'intégralité du flux, les applications java ne peuvent être hébergées que par des pages comportant l'extension «middle» (convention). Tout autre type de document est retransmis en l'état : le Proxy est alors transparent. Le contenu des pages «middle» est parcouru ligne par ligne à la recherche de la balise «<!MIDDLE». Conformément à la norme HTML, il s'agit d'une balise de commentaires. Elle n'est donc pas prise en compte par les navigateurs, ni même les outils de conception de page. Veillez toutefois à rester vigilant quant à l'utilisation de logiciel d'optimisation de page : ils suppriment les commentaires afin d'alléger le volume des informations transmises pour la page. Dans ce cas, modifier ses options pour éviter la disparition du balisage ajouté. La syntaxe exacte de la balise est la suivante : « **<!MIDDLE=** *nom_de_l'application* ? *paramètres* **>** ». Le *nom_de_l'application* et les *paramètres* sont respectivement mémorisés dans les variables theRequestedProvider et providerArgs.

```
            while ((line=fromHost.readLine())!=null) {
           int p,q;
          String result="";
           while ((p=line.indexOf("<!MIDDLE="))!=-1) {
             String theRequestedProvider="";
             if ((q=line.indexOf(">",p))!=-1) {
               theRequestedProvider=line.substring(p+9,q);
               if (theRequestedProvider.indexOf("?")>-1) {
                 providerArgs=theRequestedProvider.substring(
                   theRequestedProvider.indexOf("?")+1,
                   theRequestedProvider.length());
                 theRequestedProvider=theRequestedProvider.substring(
                   0,theRequestedProvider.indexOf("?"));
```

```
              }
            }
```

L'application préalablement sérialisée (voir «***Et ses applications...***») est chargée en mémoire, sa méthode «get» est ensuite appelée afin de disposer du complément inhérent aux traitements effectués par celle-ci (la méthode et donc de façon intrinsèque l'application). Le fichier est ensuite fermé. Une évolution possible est d'autoriser le chargement à partir d'une autre source que le disque : url, rmi, etc. Le contenu est ensuite concaténé au résultat (result). La balise traitée est éliminée de la ligne en cours d'analyse et le traitement est répété. Il se termine après épuisement des balises «<! MIDDLE=» par l'ajout de la chaîne résiduelle au résultat. Le flux est fermé après transmission du résultat au navigateur.

```
              FileInputStream fromFile=new FileInputStream(theRequestedProvider);
              ObjectInputStream theObjectInputStream=
                new ObjectInputStream(fromFile);
              MiddleProvider provider=
                (MiddleProvider)theObjectInputStream.readObject();
              fromFile.close();

              result=line.substring(0,p)+result+provider.get(new String[] {
                pageArgs, providerArgs });
              line=line.substring(++q,line.length());
            }
            result=result+line;
            toBrowser.println(result);
          }
          fromHost.close();
        }
        else {
          InputStream fromHost=hostUrl.openStream();
          byte buffer[]=new byte[fromHost.available()];
          fromHost.read(buffer);
          toBrowser.println(new String(buffer));
          fromHost.close();
```

Dans le cas d'un document classique (comprendre non doté de l'extension «middle») son contenu est chargé dans un buffer, puis ré-acheminer au navigateur.

```
        }
        toBrowser.flush();
        toBrowser.close();
      }
    }
  }
```

En cas d'anomalie, dans la mesure du possible (d'où le second try-catch), la nature de l'erreur rencontrée est transmise au navigateur. La mise au point des applications hébergées et des évolutions du Proxy n'en est que plus facile.

```
    catch (Exception e) {
     try {
       toBrowser.println("HTTP/1.0 404 Not Found\nContent-Type: text/html\n");
       toBrowser.println("<HTML><HEAD><TITLE>404 Not Found</TITLE></HEAD>"+
         "<BODY><H1>404 Not Found</H1><P>"+
         e.toString()+
         "</BODY></HTML>");
       toBrowser.flush();
       toBrowser.close();
     }
     catch (Exception f) {};
   }
 }
```

```
 private static void usage() {
   System.err.println("usage: java Middle <port>");
   System.exit(1);
 }
```

En l'absence du paramètre (indication du port à employer), un message rappelle à l'utilisateur la syntaxe utilisée pour l'activation du Proxy.

```
  public static void main(String args[]) {
    if (args.length==1) {
      port = Integer.parseInt(args[0]);
      try {
        ServerSocket serverSocket = new ServerSocket(port);
        for (;;) {
          Socket browserSocket=serverSocket.accept();
          new Middle(browserSocket).start();
        }
      }
      catch (Exception e) {
        e.printStackTrace();
      }
    }
    else
      usage();
  }
}
```

L'amorce du programme se limite à la création d'une nouvelle instance de la classe «middle» suite à la connexion d'un nouvel utilisateur. Le processus est itéré sans fin («for (;;)»).

Et ses applications...

Les applications hébergées par les pages «middle» sont stockées sous forme sérialisée. Elles implémentent toute la même interface «MiddleProvider». Celle-ci permet le chargement de l'application et son utilisation (via la méthode «get») indifférement de leur propre spécificité (merci java).

MiddleProvider.java

```
public interface MiddleProvider {
  String get(String[] Argv);
}
```

L'implémentation des applications va être abordée sous la forme d'un ensemble de page HTML («.html» ou «.middle») et de deux exemples : «Essai.java» et «Select.java».

Essai.java

```
import java.io.*;

public class Essai implements java.io.Serializable, MiddleProvider {
  public String get(String Argv[]) {
    return("Hi...");
  }

  public static void main(String Argv[]) {
    try {
      Essai essai1=new Essai();
      FileOutputStream fileOutputStream1=new FileOutputStream("Essai");
      ObjectOutputStream objectOutputStream1=new ObjectOutputStream(fileOutputStream1);
      objectOutputStream1.writeObject(essai1);
      objectOutputStream1.flush();
      objectOutputStream1.close();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

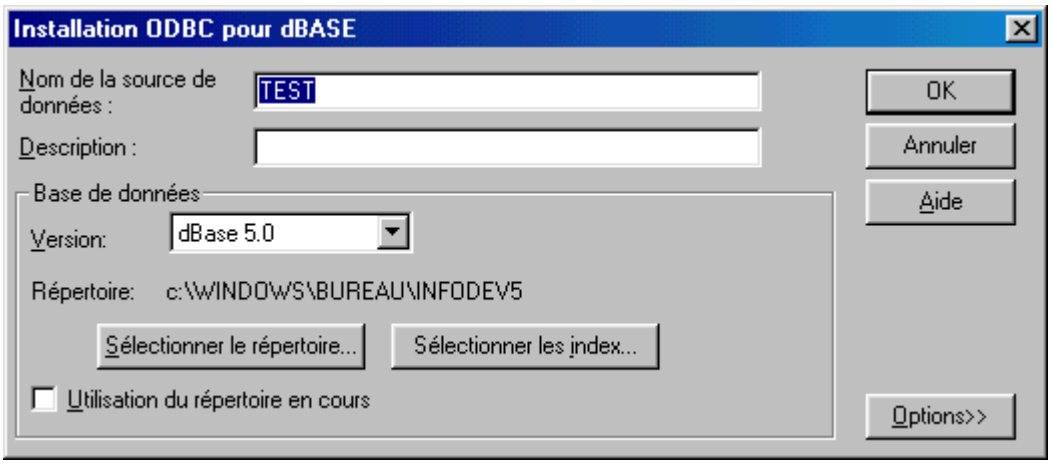
L'application type (en l'occurrence «Essai.java») se décompose en deux parties. L'implémentation de la méthode «get» conformément à l'interface «MiddleProvider» et une amorce (main). Le traitement effectué par cette méthode se limite à la sérialisation d'une instance de la classe dans le fichier «Essai». La méthode «get» est réduite à sa plus simple expression : return(«Hi...»). La page hébergeant l'application «Essai» se nomme «Essai.middle» (originale, non ?). Lors de l'appel de la page «Essai.middle», le Proxy intercepte la requête (un «GET») et retourne la page suivante : «<HTML> Exemple d'appel : " **Hi...** " .».

Essai.middle

```
<HTML>
Exemple d'appel : "<!MIDDLE=essai>".
</HTML>
```

Ce premier exemple est on-ne-peut-plus simple. Les perpectives relatives aux possibilités offertes sont largement plus importante qu'il ne le laisse présager. Ainsi, la seconde application «Select.java» devrait éveiller plus que de la curiosité. L'objet de l'application est d'effectuer une requête d'extraction de données sur la base de données «TEST». Il s'agit d'une source de donnée ODBC définie conformément à la figure 1. Le nom du répertoire employé devra être adapté en fonction du répertoire réellement employé. La base ne comporte qu'une table qui se nomme également «TEST» est donc la description est : CODE, C5 ; LIBELLE, C20 ; PRIX, N8.2. Elle comporte les cinq enregistrements suivants :

Code	Libelle	Prix
MOBRI	MONITEUR 15'' COULEUR LCD	8 000
MODLA	DALLE ANTI-REFLET	300
M105D	MONITEUR 15'' AUTOSYNC	1 000
M107A	MONITEUR 17'' AQUA	2 600
SP001	SUPPORT 15'' ORIENTABLE	100



- figure 1 : la source ODBC -

Select.java

```
import java.io.*;
import java.sql.*;

public class Select implements java.io.Serializable, MiddleProvider {
  public String get(String Argv[]) {
    String result="<table border><tr><td>code</td><td>libelle</td><td>prix</td></tr>";
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con=DriverManager.getConnection("jdbc:odbc:TEST","","");
      Statement stmt=con.createStatement();
      ResultSet res=stmt.executeQuery(Argv[1]);
      while (res.next()) {
         result+="<tr>";
          result+="<td>"+res.getString("code")+"</td>";
          result+="<td>"+res.getString("libelle")+"</td>";
          result+="<td>"+res.getFloat("prix")+"</td>";
         result+="</tr>";
      }
```

Le principe général de «Select.java» est similaire à celui de «Essai.java». Seule l'implémentation de la méthode «get» diffère. Les paramètres de l'application (Argv[1], Argv[0] stockant ceux de la page hôte) comporte une requête SQL à exécuter. Dans le cadre de l'exemple, il s'agit d'un «select * from TEST where ...». L'ensemble de données résultant est ensuite parcouru et restitué sous la forme

d'une table (balises exploitées : <table border> «<tr> <td> ... </td> </tr> ... </table>). En cas d'erreur l'objet de celle-ci est retourné en lieu et place de la table attendue.

```
    }
    catch(Exception e) {
      result=e.toString();
    }
    result+="</table>";
    return(result);
  }

  public static void main(String Argv[]) {
    try {
      Select select1=new Select();
      FileOutputStream fileOutputStream1=new FileOutputStream("Select");
      ObjectOutputStream objectOutputStream1=new ObjectOutputStream(fileOutputStream1);
      objectOutputStream1.writeObject(select1);
      objectOutputStream1.flush();
      objectOutputStream1.close();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

La page hôte comporte deux appels à l'application, le premier avec une requête listant l'ensemble des articles de la base, la seconde se limitant aux moniteurs.

Select.middle

```
<HTML>
<P>La liste</P>
<!MIDDLE=Select?select * from test>
<P>La liste des moniteurs</P>
<!MIDDLE=Select?select * from test where libelle like 'MONITEUR%'>
</HTML>
```

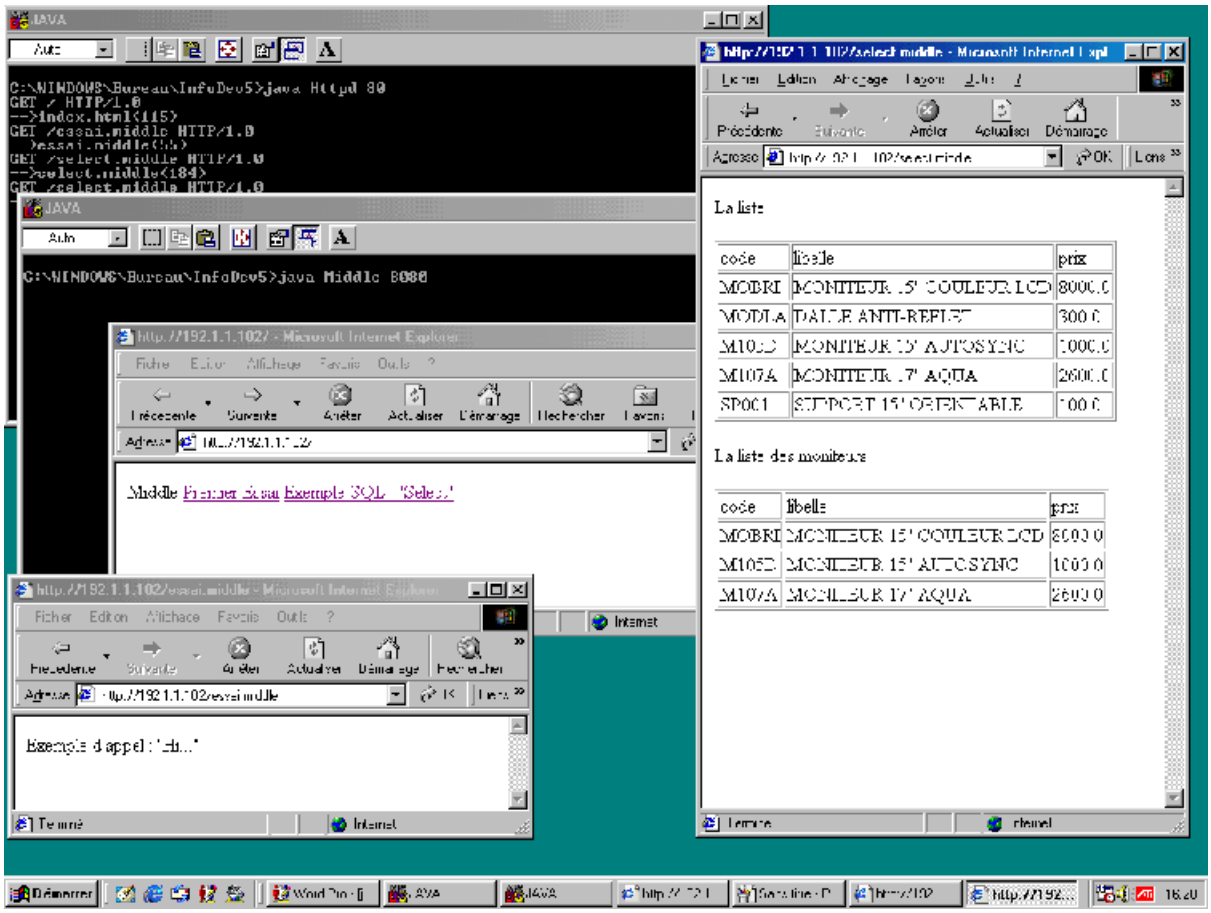
Il ne manque plus qu'une page d'index offrant le minimum d'interractivité requise. La page ci-dessous devrait convenir :

Index.html

```
<HTML>
Middle
<A HREF=essai.middle>Premier Essai</A>
<A HREF=select.middle>Exemple SQL : "Select"</A>
</HTML>
```

Au delà de la théorie... la pratique

Les «images» des applications «Essai» et «Select» sous forme sérialisées sont obtenues après compilation par leur activation : «javac Essai.java», puis «java Essai» (idem pour «Select). Les fichiers «Index.html», «Essai.middle», «Select.middle» doivent être copiés dans le répertoire racine du serveur HTTP. Les fichiers «Essai» et «Select» doivent être copiés dans le répertoire d'activation du Proxy. Si vous ne disposez pas d'un serveur HTTP en état de marche, ou que vous ne souhaitez pas altérer une configuration existante, vous pouvez copier tous les fichiers dans le même répertoire. Compilez alors le serveur HTTP décrit dans PDBF n°94 («Httpd.java») dans ce même répertoire et activez le par la commande «java Httpd 80». Toujours dans le même répertoire, suite à sa compilation, activez le Proxy pas «java Middle 8080». Activez votre navigateur favori et demandez l'url suivante : «http:// *adresse_du_poste_serveur* ». Vous devez alors optenir un affichage similaire à la figure 2. ATTENTION, la source de données ODBC précédemment décrite doit être préalablement convenable configurée.



- figure 2 : l'ensemble des donnés consultable -

Des bizarreries ?!

Les seules anomalies susceptibles de perturber les tests sont connexes aux méthodes de résolution des adresses concernant l'utilisation du réseau. Assurez-vous que votre fichier «c:\windows\hosts» est convenablement configuré. S'il n'existe pas renommez le fichier installé par défaut «c:\windows\hosts.sam» en «c:\windows\hosts.sam». Pensez à configurer la connexion de votre navigateur. Il doit employer le Proxy et pas interroger directement le serveur Http (sous IE5, le réglage s'effectue dans le menu «Outils \ Options Internet \ Connexions \ Paramètres LAN» . Cochez la case «utiliser un serveur proxy» et indiquez son adresse. Lorsque toutes les applications : Httpd, Middle et le navigateur sont utilisées à partir d'un même poste, l'adresse renseignée est celle du poste lui-même. L'option «ne pas utiliser de serveur proxy pour les adresses locales» ne doit pas être cochée. Si la configuration d'un Proxy, vous laisse dubitatif ou si vous ne souhaitez pas toucher au réglage de votre navigateur (pas confiance ?), vous pouvez appeler directement la Proxy en indiquant au navigateur le port à exploiter. L'url «http:// *adresse_du_poste_serveur*» devient alors «http:// *adresse_du_poste_serveur* **: 8080**» où 8080 est le port affecté au Proxy (comme quoi effectivement tous les chemins mènent à Rome).

Le serveur embryonnaire Httpd ci-dessous peut être employé au même titre que tout serveur HTML pour la mise en œuvre des exemples.

Httpd.java

```
import java.net.*;
import java.io.*;
import java.util.*;

class Client extends Thread {
  private Socket browserSocket;

  Client(Socket socket) {
    browserSocket = socket;
  }
```

```
  public void run() {
    Socket serverSocket = null;

    BufferedReader fromBrowser = null;
    OutputStream toBrowser = null;
    PrintWriter toBrowserWriter = null;

    try {
      String line;

      fromBrowser = new BufferedReader(new
InputStreamReader(browserSocket.getInputStream()));
      toBrowser = browserSocket.getOutputStream();
      toBrowserWriter = new PrintWriter(toBrowser);

      if ((line = fromBrowser.readLine()) != null) {
        System.out.println(line);
        StringTokenizer t = new StringTokenizer(line);
        String command = t.nextToken();
        if (t.hasMoreTokens() && command.equals("GET")) {
          String urlString = t.nextToken();
          if (urlString.equals("/"))
            urlString="index.html";
          if (urlString.startsWith("/"))
            urlString=urlString.substring(1,urlString.length());
          FileInputStream fis = new FileInputStream(urlString);
          System.out.println("-->"+urlString+"("+fis.available()+")");
          byte b[]=new byte[fis.available()];
          fis.read(b);
          fis.close();
          toBrowserWriter.println("HTTP/1.0 200 OK\nContent-Type: text/html\n");
          toBrowserWriter.println(new String(b));
          toBrowserWriter.flush();
        }
      }
    }
    catch (Exception e) {
      if (toBrowser != null)
        try {
          toBrowserWriter.println("HTTP/1.0 404 Not Found\nContent-Type: text/html\n");
          toBrowserWriter.println("<HTML><HEAD><TITLE>404 Not
Found</TITLE></HEAD><BODY><H1>404 Not Found</H1><P>"+e.toString()+"</BODY></HTML>");
          toBrowserWriter.flush();
        } catch (Exception f) {}
    }
    finally {
      if (fromBrowser != null)
        try {fromBrowser.close();} catch (Exception e) {}
      if (toBrowserWriter != null)
        try {toBrowserWriter.close();} catch (Exception e) {}
      try {browserSocket.close();} catch (Exception e) {}
    }
  }
}


class Httpd {
  private static int port;

  private static void usage() {
    System.err.println("usage: java httpd <port>");
    System.exit(1);
  }

  public static void main(String[] args) {
    if (args.length==1) {
      try {
        port = Integer.parseInt(args[0]);
        try {
          ServerSocket server = new ServerSocket(port);



          for (;;) {
            Socket s = server.accept();
            Client client = new Client(s);
            client.start();
          }
```

```
        } catch (Exception e) {
          System.err.println("Erreur: " + e);
          System.exit(1);
        }
      } catch (NumberFormatException e) {
        usage();
      }
    }
    else
      usage();
  }
}
```

La foire aux questions...

Pourquoi implémenter un Proxy plutôt que compléter les possibilités du serveur Web «Httpd» ?

- L'optimisation liée à l'intégration de la gestion des balises au sein même du serveur est évidente : une seule application en mémoire, un trafic plus simple (moins de socket), etc. Toutefois, elle impose l'usage du serveur Http développé. L'intérêt du Proxy est qu'il n'impose aucun serveur de page en particulier d'une part et d'autre part, il permet de décharger le poste doté du serveur HTTP de l'exécution des applications hébergées lorsque le (ou les) Proxy sont installés sur d'autres équipements.

L'utilisation du Proxy permet-elle le dialogue entre un applet et une application hébergée ?

- Oui, via l'utilisation des paramètres de la page avec une url du type «http:// *adresse_du_poste* / *nom de_la_page* ? *requête_de_l'applet*» .