programmation réseau

Introduction

Java propose un ensemble de classes pour la programmation réseau. Ces classes sont regroupées dans le paquetage `java.net`.
On peut programmer très facilement des architectures client/serveur en mode TCP ou UDP à l'aide des classes proposées.

Serveur TCP

```
// Exemple inspire du livre _Java in a Nutshell_
import java.io.*;
import java.net.*;

public class MonServeur extends Thread {
   public final static int DEFAULT_PORT = 6789;
   protected int port;
   protected ServerSocket listen_socket;
    //  traitement des erreurs
   public static void fail(Exception e, String msg) {
      System.err.println(msg + ": " +  e);
      System.exit(1);
   }
    // Cree un serveur TCP : c'est un objet de la
    // classe ServerSocket
      // Puis lance l'ecoute du serveur.
   public MonServeur(int port) {
      if (port == 0) port = DEFAULT_PORT;
      this.port = port;
     try { listen_socket = new ServerSocket(port); }
      catch (IOException e) {
         fail(e, "Pb lors de la creation de la socket
server"); }
System.out.println("Serveur lance sur le port " +
port);
      this.start(); }
```



```
    // Le corps de la thread d'ecoute :
   // Le serveur ecoute et accepte les connections.
   // pour chaque connection, il cree une thread
   // (objet de la classe Connection,
   // classe derivee de la classe Thread)
   // qui va la traiter.
    public void run() {
    try {
     while(true) {
       Socket client_socket = listen_socket.accept();
    Connection c = new Connection(client_socket);
     }
    }
    catch (IOException e) {
    fail(e, "Pb lors de l'ecoute");
      }
   }

    // Le lancement du programme :
       // - initialise le port d'ecoute
       // - lance la construction du serveurTCP
   public static void main(String[] args) {
      int port = 0;
      if (args.length == 1) {
         try { port = Integer.parseInt(args[0]);  }
         catch (NumberFormatException e) { port =
0; }
      }
       new MonServeur(port);
   }
}
```



```
//  la classe Connection : c'est une thread
class Connection extends Thread {
   protected Socket client;
   protected DataInputStream in;
   protected PrintStream out;

   // Initialise les streams and lance la thread
   public Connection(Socket client_socket) {
      client = client_socket;
      try {
        in = new
DataInputStream(client.getInputStream());
         out = new
PrintStream(client.getOutputStream());
      }
      catch (IOException e) {
         try { client.close(); } catch (IOException e2)
{ ; }
          System.err.println("Exception lors de
l'ouverture des sockets : " + e);
         return;
      }
      this.start();
   }
```

```
    // Fournit le service (inverse la ligne recue et la
    // retourne)
    public void run() {
        String line;
        StringBuffer revline;
        int len;
        try {
            for(;;) {
                // lit la ligne
                line = in.readLine();

            // Si cette ligne est vide, le serveur se termine
            if (line == null) break;
                // sinon l'ecrit a l'envers
                len = line.length();
                revline = new StringBuffer(len);
                for(int i = len-1; i >= 0; i--)
                    revline.insert(len-1-i, line.charAt(i));
                // et l'envoie dans la socket
                out.println(revline);
            }
        }
        catch (IOException e) { ; }
        finally { try {client.close();} catch (IOException
e2) {;} }
    }
}
```

Conclusion : serveur TCP

On utilise donc la structure de code :

```
ServerSocket listen_socket = new ServerSocket(port);
Socket client_socket = listen_socket.accept();
DataInputStream in = new
DataInputStream(client.getInputStream());
PrintStream out = new
PrintStream(client.getOutputStream());
```


Client TCP

```
// Exemple inspire du livre _Java in a Nutshell_
import java.io.*;
import java.net.*;

public class MonClient {
    public static final int DEFAULT_PORT = 6789;
    public static void usage() {
        System.out.println("Usage: java MonClient
<machineServeur> [<port>]");
        System.exit(0);
    }
    public static void main(String[] args) {
        int port = DEFAULT_PORT;
        Socket s = null;
        // initialise le port
        if ((args.length != 1) && (args.length != 2))
            usage();
        if (args.length == 1) port = DEFAULT_PORT;
        else {
            try { port = Integer.parseInt(args[1]); }
            catch (NumberFormatException e) {
usage(); }
        }

        try {
            // Cree une socket pour communiquer
            // avec le service se trouvant sur la
            // machine args[0] au port port
            s = new Socket(args[0], port);
            // Cree les streams pour lire et ecrire
            // du texte dans cette socket
            DataInputStream sin = new
DataInputStream(s.getInputStream());
            PrintStream sout = new
PrintStream(s.getOutputStream());
            // Cree le stream pour lire du texte a partir
```



```
du clavier
            DataInputStream in = new
DataInputStream(System.in);

            // Informe l'utilisateur de la connection
            System.out.println("Connected to " +
s.getInetAddress() + ":"+ s.getPort());

            String line;
            while(true) {
                // le prompt
                System.out.print("> "); System.out.flush();
                // lit une ligne du clavier
                line = in.readLine();
                if (line == null) break;
                // et l'envoie au serveur
                sout.println(line);
                // lit une ligne provenant de la socket,
                // donc du serveur
                line = sin.readLine();
                // Verifie si la connection est fermee.
                // Si oui on sort de la boucle
            if (line == null) {
                System.out.println("Connection ferme par
le serveur."); break;
            }
        // Ecrit la ligne traite par le serveur et envoye
        // par lui.
                System.out.println(line);
            }
        }
        catch (IOException e) { System.err.println(e); }
        // Refermer dans tous les cas la socket
        finally { try { if (s != null) s.close(); }
                    catch (IOException e2) { ; } }
    }
}
```


l'exécution donne

Sur la machine serveur :

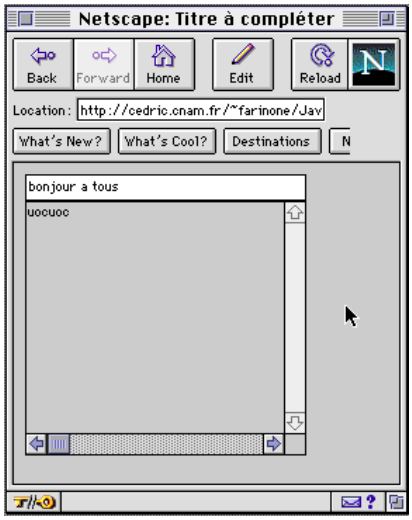
```
java MonServeur &
```

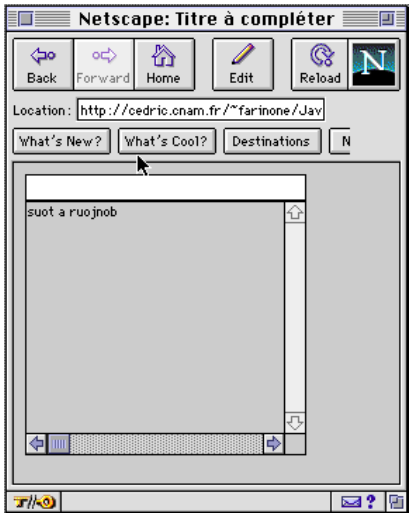
Sur la machine client :

```
java MonClient MachineDist
Connected to MachineDist/163.173.XXX.XXX:6789
> bonjour
ruojnob
> ca marche tres bien
neib sert ehcram ac
> au revoir
riover ua
```

applet cliente

Une applet peut se connecter sur le site d'où elle provient. En supposant que ce site possède le serveur ci dessus, voici une applet qui affiche une zone de saisie et une fenêtre de texte.
L'utilisateur écrit une chaîne dans la zone de saisie, appuie sur la touche Entree. Cette chaîne est envoyée au serveur, qui la traite et la retourne (à l'envers).
Le résultat retourné est affiché dans la fenêtre de texte.





Un exemple

```
// Exemple fortement inspiré du livre "Java in a Nutshell" de David Flanagan.

import java.applet.*;
import java.awt.*;
import java.io.*;
import java.net.*;

public class AppletClient extends Applet {
    public static final int PORT = 6789;
    Socket s;
    DataInputStream in;
    PrintStream out;
    TextField inputfield;
    TextArea outputarea;
    ThreadDecoutelistener;

    // Create a socket to communicate with a server on port 6789 of the
    // host that the applet's code is on.  Create streams to use with
    // the socket.  Then create a TextField for user input and a TextArea
    // for server output.  Finally, create a thread to wait for and
    // display server output.
```



```
    public void init() {
        try {
            s = new Socket(this.getCodeBase().getHost(), PORT);
            in = new DataInputStream(s.getInputStream());
            out = new PrintStream(s.getOutputStream());

            inputfield = new TextField();
            outputarea = new TextArea();
            outputarea.setEditable(false);
            this.setLayout(new BorderLayout());
            this.add("North", inputfield);
            this.add("Center", outputarea);

            listener = new ThreadDecoute(in, outputarea);

            this.showStatus("Connected to "
                + s.getInetAddress().getHostName()
                + ":" + s.getPort());
        }
        catch (IOException e)
            this.showStatus(e.toString());
    }

    // Quand l'utilsateur ecrit une ligne et appui
    // RETURN, cette ligne est envoyée au serveur
    public boolean action(Event e, Object what) {
        if (e.target == inputfield) {
            out.println((String)e.arg);
            inputfield.setText("");
            return true;
        }
        return false;
    }
}
```

```
// Thread qui attend la réponse du serveur puis
// l'affiche dans la TextArea.

class ThreadDecoute extends Thread {
   DataInputStream in;
   TextArea output;
   public ThreadDecoute(DataInputStream in,
TextArea output) {
      this.in = in;
      this.output = output;
      this.start();
   }
   public void run() {
      String line;
      try {
         for(;;) {
            line = in.readLine();
            if (line == null) break;
            output.setText(line);
         }
      }
      catch (IOException e)
output.setText(e.toString());
      finally output.setText("Connection closed by
server.");
   }
}
```


programmation UDP

UDP est une autre couche transport, plus simple mais moins sécurisée que TCP. Elle est orientée sans connexion et les messages peuvent être perdus ou déséquencés. Cette couche est quand même utilisée pour son efficacité et ce sont alors les couches supérieures qui implantent la sécurité.

Les messages UDP sont appelés des datagrammes. UDP signifiant d'ailleurs "Unreliable Datagram Protocol".

Un émetteur UDP

```
// Exemple du livre "Java in a Nutshell"
import java.io.*;
import java.net.*;

// classe permettant d'envoyer du texte par UDP
public class UDPSend {
   static final int port = 6010;
   public static void main(String args[]) throws
Exception {
      if (args.length != 2) {
         System.out.println("Usage: java UDPSend
<hostname> <message>");
         System.exit(0); }
      // On récupère l'adresse IP de l'hôte distant.
      InetAddress address =
InetAddress.getByName(args[0]);

      // Convertion du texte en tableau de byte
      int msglen = args[1].length();
      byte[] message = new byte[msglen];
      args[1].getBytes(0, msglen, message, 0);

      // On construit le datagram.
      // c'est dans le datagram qu'est précisé la
      // destination finale hôte et le numéro de port
      // de la destination finale.
      DatagramPacket packet = new
DatagramPacket(message, msglen,
       address, port);
      // Crée la socket pour l'envoi de datagrams,
      // et effectue cet envoi.
      DatagramSocket socket = new
DatagramSocket();  socket.send(packet);
   }
}
```


Un récepteur UDP

```
// Exemple du livre "Java in a Nutshell"
// Ce programme écoute le port 6010 et affiche
// la chaine de caractères reçue sur ce port.

import java.io.*;
import java.net.*;
public class UDPReceive {
   static final int port = 6010;
   public static void main(String args[]) throws
Exception
   {  byte[] buffer = new byte[1024];
      String s;
      // Crée une socket d'écoute sur ce port.
      DatagramSocket socket = new
DatagramSocket(port);
      for(;;) {
         // Création du packet de réception.
         DatagramPacket packet = new
DatagramPacket(buffer, buffer.length);
         // écoute et réception d'un datagram.
         socket.receive(packet);
         // Convertion byte -> String à l'aide
         // du constructeur
//String(ta_byte[], octpoidsfort, inddeb, longueur)
         s = new String(buffer, 0, 0,
packet.getLength());
         // affichage du packet reçu en indiquant la
         // machine émettrice et le numéro de port
         // de la machine émettrice.
         System.out.println("UDPReceive: received
from " + packet.getAddress().getHostName() + ":"
+packet.getPort() + ": " + s);
      }
   }
}
```

Une exécution

La machine `MachEmet` est émettrice, la machine `Recept` est réceptrice. On lance le récepteur :

```
Recept: % java UDPReceive
```

on envoie des données UDP :

```
MachEmet: % java UDPSend Recept 'ca marche super'
MachEmet: % java UDPSend Recept '2ieme essai'
```

et on les reçoie au fur et à mesure :

```
Recept: % java UDPReceive
UDPReceive: received from MachEmet:35913:  ca marche super
UDPReceive: received from MachEmet:35914: 2ieme essai
```


Connexion sur des URL

Java propose des classes permettant de manipuler des URL.

La classe `URL`

principales méthodes

La classe `URL` permet de construire une URL à partir d'une chaîne de caractères.

principales méthodes

```
public URL(String) throws MalformedURLException

public URL(String, String hote, String fic) throws MalformedURLException

public URL(String protocol, String hote, int port, String fic) throws MalformedURLException
```

sont des constructeurs retournant un objet `URL`.

Les méthodes `getFile()`, `getHost()`, `getPort()`, `getProtocol()` retournent des informations sur l'instance.

La classe `URLConnection`

principales méthodes

On permet d'obtenir des informations plus précises d'une ressource repérée par son URL.

On fait correspondre un objet `URL` à un objet `URLConnection` par

```
public URLConnection openConnection()
```

de la classe `URL` ou le constructeur

```
protected URLConnection(URL)
```

On peut obtenir des renseignements à l'aide de :

```
public String getContentType()
```

retourne la valeur du champ `content-type` (i.e. le format MIME (type/soustype)) du document HTML

```
public int getContentLength()
```

retourne la valeur du champ `content-length` (i.e. la taille) du document HTML

```
public long getLastModified()
```

retourne la valeur du champ `last-modified` (i.e. la date de dernière modification) du document HTML. Le résultat est le nombre de secondes depuis le 1er janvier 1970 GMT.

```
public long getExpiration()
```

retourne la valeur du champ `expires` (i.e. la date d'expiration) du document HTML ou 0 si ce champ est inconnu.

Exemple

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

public class GetURLInfo {
   public static void printinfo(URLConnection u)
throws IOException {
System.out.println(u.getURL().toExternalForm()
+ ":");
      System.out.println("  Content Type: " +
u.getContentType());
      System.out.println("  Content Length: " +
u.getContentLength());
      System.out.println("  Last Modified: " + new
Date(u.getLastModified()));
      System.out.println("  Expiration: " +
u.getExpiration());
      System.out.println("  Content Encoding: " +
u.getContentEncoding());

      // Read and print out the first five lines of the
URL.
      System.out.println("First five lines:");
      DataInputStream in = new
DataInputStream(u.getInputStream());
      for(int i = 0; i < 5; i++) {
         String line = in.readLine();
         if (line == null) break;
         System.out.println("  " + line);
      }
   }
```

```
    // Create a URL from the specified address,
open a connection to it,
    // and then display information about the URL.
    public static void main(String[] args)
        throws MalformedURLException,
IOException
    {
        URL url = new URL(args[0]);
        URLConnection connection =
url.openConnection();
        printinfo(connection);
    }
}
```


Une exécution

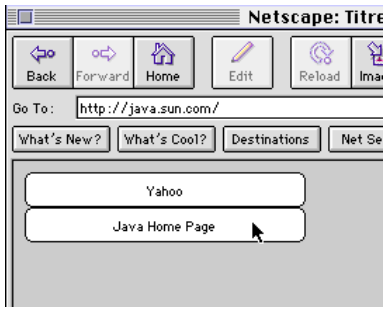
```
pouchan: % java GetURLInfo
'http://cedric.cnam.fr/personne/farinone/'
http://cedric.cnam.fr/personne/farinone/:
   Content Type: text/html
   Content Length: 1485
   Last Modified: Tue Sep 10 17:46:23 MET DST
1996
   Expiration: 0
   Content Encoding: null
First five lines:
   <html>
   <title>Page d'accueil de Jean Marc
Farinone</title>

   <body><TITLE>Page d'accueil de Jean Marc
Farinone</TITLE>
   <h2>Jean Marc Farinone</h2>
```


`showDocument()` de la classe `AppletContext`

On peut aussi, à l'intérieur d'une applet, lancer une connexion sur une URL et afficher son contenu dans le browser à l'aide de la méthode `showDocument()` de la classe `AppletContext`. On obtient un objet `AppletContext` à partir d'un objet `Applet` grâce à la méthode `getAppletContext()`.

```
/* inspiré de "le programmeur Java" */

import java.awt.*;
import java.net.*;

public class ButtonLink extends
java.applet.Applet {
  static final int nbBoutons = 2;
  Bookmark bmlist[] = new Bookmark[nbBoutons];
  Button bt[] = new Button[nbBoutons];

  public void init() {
    bmlist[0] = new Bookmark("Yahoo",
"http://www.yahoo.com");
    bmlist[1]= new Bookmark("Java Home Page",
"http://java.sun.com");

    setLayout(new GridLayout(bmlist.length,1));

    for (int i = 0; i < bmlist.length; i++) {
      add(bt[i] = new Button(bmlist[i].name));
    }
  }
```



```
public boolean action(Event evt, Object arg) {
      URL theURL = null;
      for (int i=0; i < bt.length; i++)
    if (evt.target == bt[i]) {
        theURL = bmlist[i].url;
     }
    if (theURL != null) {
      showStatus("chargement de: " + theURL);
      getAppletContext().showDocument(theURL);
      return true;
    }
    else return false;
  }
}

class Bookmark {
  String name;
  URL url;

  Bookmark(String name, String theURL) {
    this.name = name;
    try { this.url = new URL(theURL); }
    catch ( MalformedURLException e) {
      System.out.println("Bad URL: " + theURL);
    }
  }
}
```

Une exécution