Java Avance

Programmation réseau

Emmanuel ADAM

Université Polytechnique des Hauts-De-France

UPHF/INSA HdF

Communication par sockets

package java.net

Présentation

- Les classes de java.net.* permettent d'établir des flux d'entrées/sorties entre machines à l'aide de sockets
- Il existe deux protocoles utilisant les sockets en Java :
  - TCP : Stream Socket
    - Communication en mode connectéS
    - Les applications sont averties lors de la déconnexion
  - UDP : Datagram Socket
    - Communication en mode non connectéS
    - Plus rapide mais moins fiable que TCP

Protocole TCP

Fonctionnement du TCP

Le serveur	Le client
Crée une socket	.
Attend une connection	⟵ Demande une connection
Accepte la demande ⟶	création de socket
Récupération d'un flux et filtrage	Récupération d'un flux et filtrage
échange de données	échange de données

ServerSocket

package java.net

Présentation

- La classe *ServerSocket* définit une socket sur le serveur.
- Pour créer une *ServerSocket*, il faut lui donner un numéro de port.
- Une fois créée, la socket Serveur attend une connection par la méthode *accept()*.

```
ServerSocket s = new ServerSocket (8888)
Socket socket = s.accept();
// attente d'une connection
```

Socket client

package java.net

Présentation

- Se connecter sur une socket d'un serveur, implique de connaître son adresse.
- *InetAddress* permet de récupérer une adresse à partir d'un nom.
- La liaison d'une socket à la socket serveur peut alors être demandée.

```
InetAddress addr = InetAddress.getByName("44.22.33.11");
Socket socket = new Socket(addr, 8888);
```

Communication inter-socket

package java.net

Communications

- Les communication entre socket s'effectue "classiquement",
  - par les flux d'entrées, issus de la méthode *getInputStream*
  - par les flux de sorties, issus de la méthode *getOutputStream*.

Le client-serveur en TCP

package java.net

Exemple

- Un exemple classique de programmation réseau : le client-serveur.
- Un serveur est lancé sur une machine et attend la connexion d'un client.
- Puis, il récupère les lignes envoyées par le client,
- il les affiche et les lui renvoie
- jusqu'à ce que le client envoie "FIN"

Solution : le serveur en TCP I

package java.net

```
import java.io.*;
import java.net.*;
class ServeurTCP  {
 public static void main(String args[]) throws IOException {
 // creation d'une socket serveur
 try (ServerSocket serveur = new ServerSocket(8888);) {
  System.out.println("Socket lancee : " + serveur);
  try ( // Attendre une connection
    Socket socket = serveur.accept();
    // recuperation du flux d'entree
    Scanner in = new Scanner(socket.getInputStream());
    // recuperation du flux de sortie
    PrintWriter out = new PrintWriter(socket.getOutputStream
        (), true);)
    {
      System.out.println("Connection acceptee : " + socket);
      String str = "";
```

Solution : le serveur en TCP II

package java.net

```
        // attente de donnees
        while (!(str.equalsIgnoreCase("FIN"))) {
            //attente de texte
            str = in.nextLine();
            System.out.println("serveur -> message recu : " +
                str);
            //retour a l'envoyeur
            out.println("AR du msg : " + str);
    } } } } }
```

Solution : le client en TCP I

package java.net

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

class LeClient {
  public static void main(String[] args) throws IOException
      {
    // recupere l'adresse internet du host
    // null permet de tester les applis sur une machine
        unique
    InetAddress addr = InetAddress.getByName(null);
    try (
      Socket socket = new Socket(addr, 8888);
      // recuperation du flux d'entree
      Scanner in = new Scanner( socket.getInputStream());
      // recuperation du flux de sortie
      PrintWriter out = new PrintWriter( socket.
          getOutputStream(),true);
      Scanner scanKeyboard = new Scanner(System.in);)
      {
```

Solution : le client en TCP II

package java.net

```
            System.out.println("chez le client , socket = " +
                socket);


            String ligne = "";
            while (!ligne.equalsIgnoreCase("FIN")) {
              System.out.println("client -> entrez une chaine :
                  " );
              //lecture au clavier
              ligne = scanKeyboard.nextLine();
              //envoi au serveur
              out.println(ligne);
              //attente du retour
              String str = in.nextLine();
              System.out.println("client -> recu du serveur :"
                  + str);
}   }   } }
```

Communication par UDP

package java.net

Présentation

- Emission de paquets de données, rapide mais peu fiable
- Définir un paquet à envoyer ou à recevoir par la classe *DatagramPacket*
- Le paquet de données contient l'adresse de destination et la longueur du paquet
- Définir une socket par la classe *DatagramSocket*

Communication par UDP

un serveur attend des données

```
import java.net.*;
public class UDPServeur
{
    public static void main(String arg[])
    {
        try
        {
            // Serveur
            DatagramSocket ds = new DatagramSocket(1234);

            while(true)
            {
                DatagramPacket packet = new DatagramPacket(new
                    byte[1024], 1024);
                ds.receive(packet);
                System.out.println("Message: " + packet.getData
                    ());
            }
        }
        catch(Exception e){}
    }
}
```

Communication par UDP

un client envoit un paquet

```
import java.net.*;
public class UDPClient
{
    public static void main(String arg[])
    {
        try
        {
            String chaine = "un message";
            byte[] data = chaine.getBytes();
            InetAddress addr = InetAddress.getByName(null);
            DatagramPacket packet = new DatagramPacket(data,
                data.length, addr, 1234);
            DatagramSocket ds = new DatagramSocket();
            ds.send(packet);
            ds.close();
        }
        catch(Exception e){}
    }
}
```

Une classe réseau : URL

package java.net

- *URL* permet de créer des liens vers des pages internets

Exemple :

```
URL url = new URL("http://www.univ-valenciennes.fr/index.
    html");
DataInputStream dis = new DataInputStream(url.openStream()
    );
String line;
while ((line = dis.readLine()) != null)
  System.out.println(line);
```

Une classe réseau : URL

package java.net

- *URLConnection* , qui représente le lien http entre le serveur et le client

Exemple :

```
try {
  URL monSite = new URL("http://emmanuel.adam.free.fr/site")
      ;
  URLConnection conn = url.openConnection() ;
  String type = conn.getContentType();
  String encoding = conn.getContentEncoding() ;
  int len = conn.getContentLength() ;
  Date lastModified = new Date(conn.getLastModified()) ;
  .....
  }
  catch(MalformedURLException e) {...;} // echec new URL()
  catch(IOException e) {...;} // echec openConnection()
```

R.M.I

PACKAGE JAVA.RMI.*

PRÉSENTATION

- R.M.I. = Remote Method Invocation : invoquer des méthodes d'objets distants ⇒ applications Java distribuées.
  - applications distribuées ⇒ classes distribuées sur un réseau
  - R.M.I. ≻ R.P.C. (uniquement transfert de données)
  - R.M.I. ≺ CORBA (Common Object Request Broker Architecture) : standard par l'OMG (Object Management Group), fonctionne sous plusieurs langages ;
  - I.I.O.P. (Internet InterObject Protocol) permet l'interopérabilité entre RMI et CORBA.

R.M.I

PACKAGE JAVA.RMI.*

PRÉSENTATION

- Les méthodes des objets distants peuvent être invoquées depuis des JVM différentes via le réseau.
- RMI assure la communication entre le serveur et le client via TCP/IP,
  transparent pour le développeur.
- RMI utilise des sockets et RMP (Remote Method Protocol).
- Gestion de la sécurité par la classe RMISecurityManager
- Gestion de la mémoire distribuée par le DGC (Distibuted Gabage Collector).

Architecture

package java.rmi.*

Principe

- Interfaçage : le client doit posséder une vue de ce qu'il peut appeler → une interface décrivant ce qui est appelable chez l'objet distant
- Le client comme le serveur possède une copie de la même interface
- seul le serveur possède une implémentation de l'interface
- Si le client appelle une méthode distante, l'appel est transmis à son "proxy"
- RMI envoit la requête à l'implémentation côté serveur
- Les valeurs de retour sont transmise au proxy client puis au client

Interface partagée

package java.rmi.*

Interface

- Il est nécessaire de définir l'interface de l'objet distant et décrire les méthodes appelables
- L'interface doit étendre l'interface Remote et être présente chez le client et chez le serveur
- Les méthodes appelables doivent pouvoir propager l'exception RemoteException
- Les objets retournés doivent être Serializable

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface TrucDistant extends Remote
{
  String donneChaine() throws RemoteException;
}
```

Implémentation de l'objet distant

Côté serveur : Implémentation de l'objet distant

Objet Distant

- Le nom de l'objet implémenant a pour suffixe Impl (convention)
- La classe doit étendre la classe RemoteObject, ou une sous classe (UnicastRemoteObject)
- Le constructeur doit propager une exception RemoteException

```
public class TrucDistantImpl extends UnicastRemoteObject
    implements TrucDistant  {

  String chaine;

  protected TrucDistantImpl() throws RemoteException {
    chaine = "coucou du serveur ! ";  }

  public String donneChaine()  throws RemoteException
  { return chaine;     }
```

Enregistrement / Recherche : Naming

Enregistrement / Recherche : Naming

- La classe Naming :
  - enregistrer un objet serveur
  - diffuser des références à des objets d'un registre distant (comme rmiregistry, JNDI, ...).
- avec adresse = "rmi ://host :port/nom" ou "//host/nom" ou ... et nom = nom de l'objet dans l'annuaire
  - bind(adresse, objet) : enregistre objet dans l'annuaire sous le nom de l'adresse
  - rebind(adresse, objet) : enregistre un nouvel objet dans l'annuaire sous nom déjà donné
  - unbind(adresse, objet) : desinscrit l'objet de l'annuaire sous l'adresse donnée
  - list() : retourne tous les enregistrements d'objets
  - lookup(adresse) : retourne une référence à l'objet distant spécifié dans adresse

Côté serveur : Exemple d'Enregistrement

Démarrer l'annuaire : lancer `rmiregistry`

```
//Le serveur :
public class LeServeur {
public static void main(String []args)
{
 try {
  // Creation d'un objet a rendre accessible :
  TrucDistantImpl truc = new TrucDistantImpl();
  // publication de l'objet dans l'annuaire
  java.rmi.Naming.bind("rmi://localhost/UnTruc", truc);
  System.out.println("Serveur lance.");
 }
 catch (Exception e) {
 System.out.println("Exception levee : " + e.getMessage());
} } }
```

Côté client : Exemple de recherche et d'utilisation

```
//Le client :
public class LeClient {
 public static void main(String []args) {
  try {
  //recuperer un objet de type TrucDistant (interface)
   TrucDistant truc = (TrucDistant) java.rmi.Naming.lookup("
       rmi://localhost/UnTruc");
   System.out.println("chaine recuperee : " + truc.
       donneChaine());
  }
  catch(Exception e){System.out.println(e);}
} }
```

Recherche de classes

Recherche de classes accessoires

- Un appel à une méthode d'un objet distant peut nécessiter des classes non présentes chez le client
  - il faut les téléchager !
- La classe RMIClassLoader est appelée automatiquement pour télécharger les classes manquantes
- Ces classes doivent être Serializable
- Il faut indiquer à Java où télécharger les classes manquantes dans le lancement du Serveur
  - Utilisation du paramètre java.rmi.server.codebase :
    - java -Djava.rmi.server.codebase=http: //172.16.12.23/lesClasses/ LeServeur
    - L'adresse peut être de type ftp ; possibilité d'indiquer un no de port

Stratégie de sécurité

Sécurité

- Le chargement dynamique de classe peut être refusé pour des raisons de sécurité.
  - Naming retourne une erreur de type AccessControlException
- Utilisation d'un gestionnaire de sécurité : RMISecurityManager.
  - `System.setSecurityManager(new RMISecurityManager());`
- Définition des permissions dans un fichier `.policy` lu automatiquement
  - exemple de contenu :

    ```
    grant {
    permission java.security.Allpermission;
    }
    ```
  - Signaler à java d'utiliser le fichier :
    - java -Djava.security.policy=chemin / monFichier.policy LeServeur

Remarques

Création d'objets distant

- Le client ne peut créer un nouvel objet distant
- Le serveur peut créer un nouvel objet distant et retourner un objet de ce type
  - Il suffit de créer une méthode retournant une nouvelle instance du type distant dans le type distant

Ramasse miette distribué

- Les objets distants non référencés sont supprimés du serveur
- Pour suivre un nettoyage d'un objet,il doit implémenter l'interface `java.rmi.server.Unreferenced`
  et coder la méthode `public void unreferenced();`

L'architecture client-serveur 1-N

package java.net

Utilisation de canaux

- Pour la communication 1-N par socket, il faut :
  - Utilisation d'un sélecteur en écoute sur un port
    - Chaque demande de connection crée une clé de connection
    - Pour chaque clé de connection : créer une clé de lecture

Exemple

- 1 serveur accepte la connection de N clients et
- suite à une lecture d'une chaine d'un client :
  - il balaie les clés et envoyer la chaine aux autres clients

Exemple de serveur 1-N I

package java.net

```
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.*;
import java.util.*;

/** d'apres Frederic Chuong */

public class ServeurNIO {
 public static void main(String[] args) {
  int port = 1080;
  try {
   // ouvrir un selecteur
   Selector selector = Selector.open();
   // ouvrir un serveur non bloquant sur le port
   ServerSocketChannel server = ServerSocketChannel.open();
   server.configureBlocking(false);
   server.socket().bind(new InetSocketAddress(port));
   server.register(selector, SelectionKey.OP_ACCEPT);
```

Exemple de serveur 1-N II

package java.net

```
// definir un tampon de 1024 bytes
ByteBuffer buffer = ByteBuffer.allocate(1024);
//tq il y a des elements dans le selecteur
while (selector.select() > 0) {
 //balayer les cles du selecteur
 Set<SelectionKey> selectedKeys = selector.selectedKeys()
     ;
 Iterator<SelectionKey> i = selectedKeys.iterator();
 while (i.hasNext()) {
  SelectionKey key = i.next();
  i.remove();
  // si la cle (canal) peut etre acceptee
  if (key.isAcceptable()) {
     // l'accepter et redefinir l'option du canal dans le
          selecteur (pour lire les donnees)
   SocketChannel socket = server.accept();
   socket.configureBlocking(false);
   socket.register(selector, SelectionKey.OP_READ);
  }
```

Exemple de serveur 1-N III

package java.net

```
// si la cle (canal) peut etre lue
else if (key.isReadable()) {
   //recuperer le canal
 SocketChannel channel = (SocketChannel) key.channel();
 int len;
 try {
   //lire dans le canal
  len = channel.read(buffer);
  // s'il n'y a plus d'elements, fermer le canal et le
       desinscrire du selecteur
  if (len <= 0) {
   channel.keyFor(selector).cancel();
   channel.close();
  } else {
   //sinon, afficher le buffer
   buffer.flip();
   System.out.write(buffer.array(), 0, len);
```

Exemple de serveur 1-N IV

package java.net

```
        // et le transmettre a tous les clients
        Set<SelectionKey> keys = selector.keys();
        Iterator<SelectionKey> i2 = keys.iterator();
        while (i2.hasNext()) {
         SelectionKey key2 =  i2.next();
         //verifier que la cle vient d'un client
         if (key2.channel() instanceof SocketChannel) {
          SocketChannel otherChannel = (SocketChannel) key2.
              channel();
          if (!channel.equals(otherChannel)) {
                //rembobiner le buffer
           buffer.rewind();
           // le transmettre au client
           otherChannel.write(buffer);
        } } }
        buffer.clear();
      }
    } catch (IOException e) {
     e.printStackTrace();
     try {
```

Exemple de serveur 1-N V

package java.net

```
                channel.keyFor(selector).cancel();
            } catch (Exception e2) {
        } } } } }
    } catch (IOException e) {
        e.printStackTrace();
    } }
}
```