Java Avance

Programmation réseau

Emmanuel ADAM

LAMIH UMR CNRS 8530
Université de Valenciennes et du Hainaut-Cambrésis
FRANCE

22 janvier 2008

Communication par sockets

package java.net

Présentation

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

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 :

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 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

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

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 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

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.

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

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.

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

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

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.

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

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("crabe");
  Socket socket = new Socket(addr, 8888);
```

Communication inter-socket

package java.net

Communications

Communication inter-socket

package java.net

Communications

- Les communication entre socket s'effectue "¡classiquement"¿,

Communication inter-socket

package java.net

Communications

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

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*.

Communication inter-socket

un serveur lit des données

```
// création d'une socket serveur
ServerSocket s = new ServerSocket(8888);
System.out.println("Socket lancee : " + s);
try {
  Socket socket = s.accept(); // Attendre une connection
  try {
    System.out.println( "Connection acceptee : "+ socket);
    // récupération du flux d'entrée
    InputStream socket_in =
                    new InputStream(socket.getInputStream());
    // filtrage pour récupérer des données
    DataInputStream in = new DataInputStream(socket_in);
    // attente d'une donnee
    String str = in.readLine(); }
  catch (Exception e) {...}  }
 catch (Exception e) {...}
```

Communication par UDP

package java.net

Présentation

Communication par UDP

package java.net

Présentation

- Emission de paquets de données, rapide mais peu fiable

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*

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

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) {...;} // échec new URL()
  catch(IOException e) {...;} // échec openConnection()
```

le client-serveur

package java.net

Exemple

le client-serveur

package java.net

Exemple

- Un exemple classique de programmation réseau : le client-serveur.

le client-serveur

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.

le client-serveur

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,

le client-serveur

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.

Solution : le serveur I

package java.net

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

class LeServeur  {
 // choisir un port hors de 1-1024:
 public static final int PORT = 8080;
 public static void main(String[] args) throws IOException {
   ServerSocket s = new ServerSocket(PORT);
   System.out.println("Socket lancee : " + s);
   try {  // Attendre une connection
     Socket socket = s.accept();
     try {
     System.out.println( "Serveur : Connection acceptee : "+ socket)
         ;
       InputStreamReader socket_in = new InputStreamReader(socket.
           getInputStream());
     BufferedReader in = new BufferedReader( socket_in );
       OutputStreamWriter socket_out =  new OutputStreamWriter(
           socket.getOutputStream());
```

Solution : le serveur II

package java.net

```
        PrintWriter out = new PrintWriter( new BufferedWriter(
            socket_out),true);
          while (true) {
           String str = in.readLine();
           if (str.equals("END")) break;
           System.out.println("Recu : " + str);
           out.println("serveur, retour d'info sur " + str); }
        // Toujours fermer les sockets...
        }
        finally {
          System.out.println("fermeture...");
          socket.close();
        } }
    finally { s.close(); }
} }
```

Solution : le client I

package java.net

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

class LeClient {
 public static void main(String[] args) throws IOException {
    // récupère l'adresse internet du host
    // null permet de tester les applis sur une machine unique
    InetAddress addr = InetAddress.getByName(null);
    System.out.println("addr = " + addr);
    Socket socket = new Socket(addr, 8080); // utilisation du même
        no de port...
    try {
      System.out.println("chez le client socket = " + socket);
      InputStreamReader socket_in = new InputStreamReader( socket.
          getInputStream());
      BufferedReader in = new BufferedReader(socket_in);
      OutputStreamWriter socket_out = new OutputStreamWriter(
          socket.getOutputStream());
      PrintWriter out = new PrintWriter( new BufferedWriter(
          socket_out ),true);
```

Solution : le client II

package java.net

```
        for(int i = 0; i < 10; i ++) {
          out.println("client : envoie de la ligne " + i);
          String str = in.readLine();
          System.out.println(str);
}
        out.println("END");
    }
finally {
        System.out.println("fermeture...");
        socket.close();
} } }
```

le client-serveur

package java.net

utilisation de canaux

le client-serveur

package java.net

utilisation de canaux

- Pour la communication 1-N par socket, il faut :

le client-serveur

package java.net

utilisation de canaux

- Pour la communication 1-N par socket, il faut :
  - Utilisation d'un sélecteur en écoute sur un port

le client-serveur

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

le client-serveur

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

le client-serveur

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

le client-serveur

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

le client-serveur

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 :

le client-serveur

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 Frédéric 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

```
// définir 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()) {
 //récupérer 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();
  } }
}
```