Introduction aux communications inter-applications

Tout sur les Sockets

Bachir Djafri

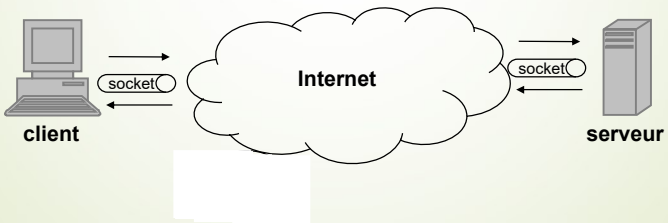
Université d'Évry – Paris-Saclay

1

Le modèle des sockets

2

- Interface (point de communication) client/serveur utilisée à l'origine dans le monde UNIX et TCP/IP
  - primitives pour le support de communications reposant sur les protocoles (TCP/IP, UDP/IP)
  - les applications client/serveur ne voient les couches de communication qu'à travers l'API socket (abstraction)



2

Protocoles TCP

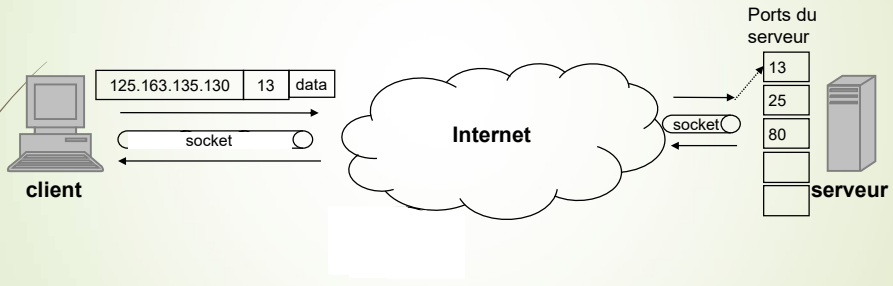
3

- TCP
  - Garantie l'arrivée dans l'ordre des paquets de données
  - Fiabilité de transmission
  - Lenteur des transmissions (http par exemple)
  - Vu comme un «service téléphonique»

Connexion réseau

4

- Adresse Internet de la machine
- Numéro du port



Notion de port

5

- Pourquoi les ports ?
  - Sur une même machine, plusieurs services sont accessibles simultanément (web, email, etc.)
  - Points d'accès : ports logiques (65535)
  - Rien à voir avec les ports physiques (USB, HDMI, série et parallèle)
- Désignation des ports
  - Port : numéro entier allant de 1 à 65535
  - Les ports de 1 à 1023 sont réservés aux services courants : finger, ftp, http (80), SMTP (25), etc.
  - Fichier d'assignation de ports : /etc/services (Unix)

Adresses Internet

6

- Connexion réseau
  - Adresse Internet de la machine
  - Numéro : 193.49.192.193 (IP V4)
- Désignation par des noms symboliques
  - Association de noms symboliques aux adresses numériques
  - Domain Name Server (ou DNS)
  - Exemple : www.univ-evry.fr : 194.199.84.39

Client-serveur en mode connecté

7

- Le client
  - ouvre une connexion avec le serveur avant de pouvoir lui adresser des appels, puis ferme la connexion à la fin de la suite d'opération
    - délimitation temporelle des échanges
    - maintien de l'état de connexion pour la gestion des paramètres de qualité de service
    - traitement des pannes, propriété d'ordre
  - orienté vers
    - traitement ordonné d'une suite d'appels
    - ordre local (requêtes d'un client traitées dans leur ordre d'émission), global ou causal
    - la gestion de données persistantes ou de protocole avec état

7

Mode connecté : caractéristiques

8

- Caractéristiques
  - établissement préalable d'une connexion (circuit virtuel) : le client demande au serveur s'il accepte la connexion
  - fiabilité assurée par le protocole de transport utilisé : TCP
  - mode d'échange par flots d'octets : le récepteur n'a pas connaissance du découpage des données effectué par l'émetteur
  - possibilité d'émettre et de recevoir des caractères urgents (OOB : Out Of Band)
  - après initialisation, le serveur est **passif**, il est activé lors de l'arrivée d'une demande de connexion d'un client
  - un serveur peut répondre aux demandes de services de plusieurs clients : les requêtes arrivées et non traitées sont stockées dans une file d'attente

8

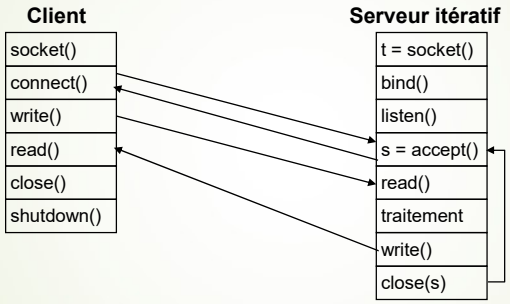
Caractéristiques du mode connecté

9

- Contrainte
  - le client doit avoir accès à l'adresse du serveur (adresse IP et numéro de port)
- Modes de gestion des requêtes
  - itératif : le processus serveur traite les requêtes les unes après les autres
  - **concurrent** : par création de processus fils pour les échanges de chaque requête

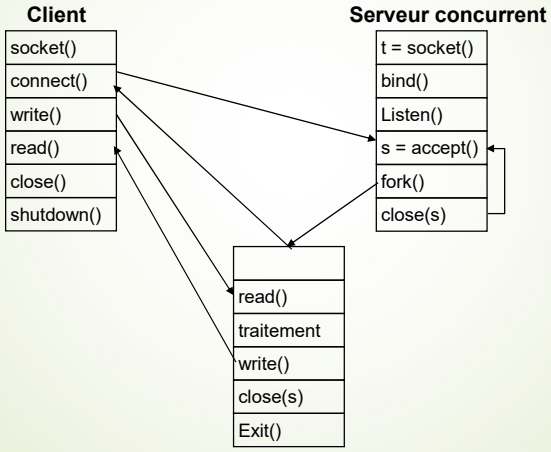
Enchaînement des opérations (1)

10



Enchaînement des opérations (2)

11



Java

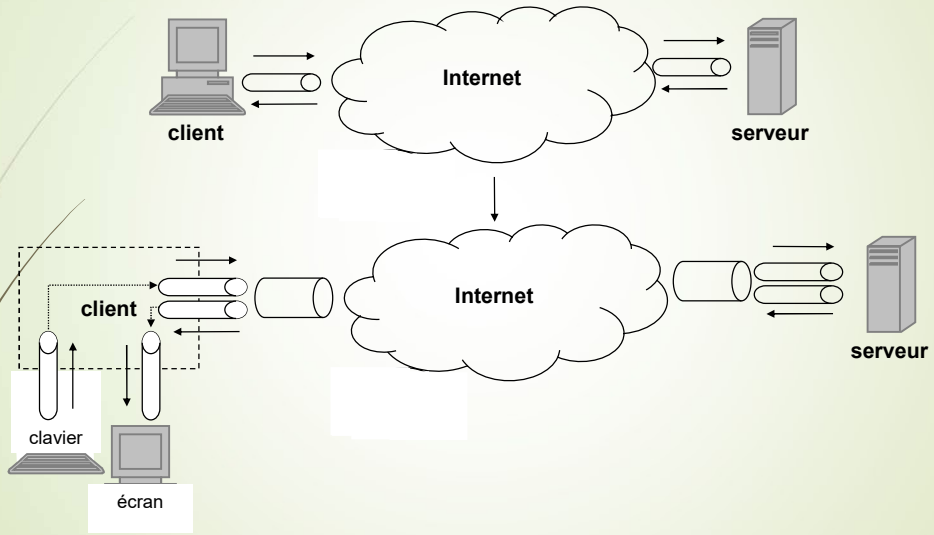
les sockets TCP sous Java

12

- Deux classes interviennent:
- java.net.Socket
  - Coté client (mais aussi coté serveur)
  - Elle permet une communication 1-1
  - C'est un couple de canaux de communication (asynchrone)
- java.net.ServerSocket
  - Utilisée uniquement coté serveur
  - C'est un point d'entrée
  - Req : demande d'établissement d'une connexion
  - Rep : établissement d'une Socket entre le serveur et le client

Les sockets Java

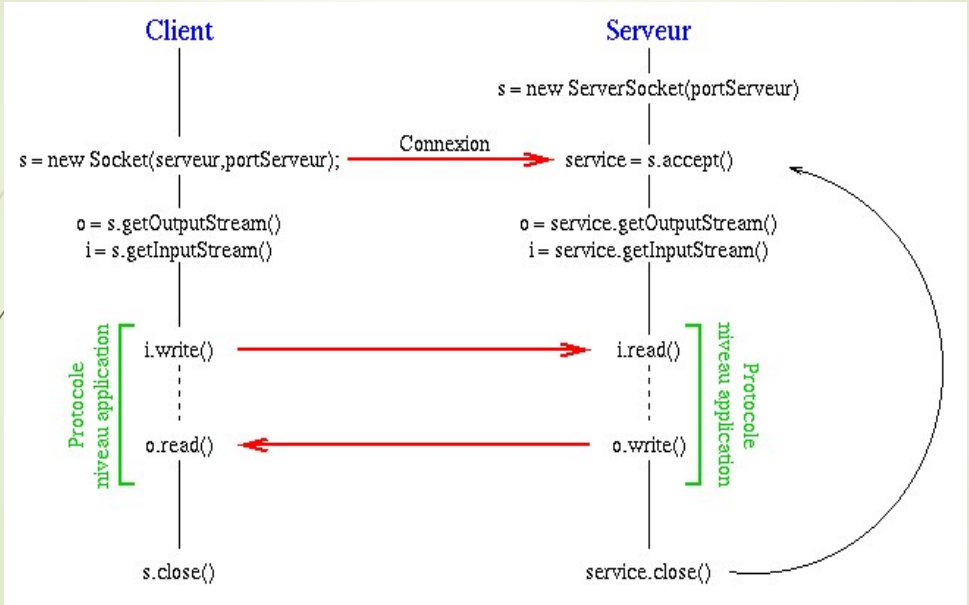
13



Java

Les sockets sous Java

14



Java

Java.net.Socket (TCP)

15

- `Socket(String host, int port)`
  - crée une socket et la connecte à un port de l'ordinateur distant (ou local)
- `void close()`
  - ferme la socket
- `InputStream getInputStream()`
  - récupère le flux de données pour lire sur la socket
- `OutputStream getOutputStream()`
  - récupère le flux de données pour écrire sur la socket
- `void setSoTimeout(int timeout)`
  - définit la valeur (en ms) de timeout en lecture sur cette socket
  - si la valeur de timeout est atteinte, une **`InterruptedIoException`** est déclenchée

Un premier client

16

- Interrogation du service « date » d'un serveur

```
$ telnet time-A.timefreq.bldrdoc.gov 13
$ 59310 21-04-06 09:28:23 50 0 0 793.3 UTC(NIST) *
```

- Implantation Java

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

public class SocketTest {
  public static void main(String[] args) {
    try {
      Socket s = new Socket("time-A.timefreq.bldrdoc.gov", 13);
      BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));

      boolean more = true;
      while (more) {
        String line = in.readLine();
        if (line == null)
          more = false;
        else
          System.out.println(line);
      }
    }
    catch (IOException e) { e.printStackTrace(); }
  }
}
```

Java

Attention ! la création d'une socket est une action bloquante

17

- Problème
  - si le serveur ne répond pas, le client est bloqué
- Solution
  - utilisation d'un timeout au bout duquel la socket sera fermée
  - quand la valeur de timeout est dépassée, toutes les opérations de lecture lancent une **`InterruptedIOException`**
- Exemple

```
Socket s = new Socket(...);
s.setSoTimeout(10000);
...
try {
  String line;
  while ((line = in.readLine()) != null) {
    // traitement de la ligne
  }
}
catch (InterruptedIoException exception) {
  // gestion du timeout
}
```


Java

Les adresses Internet

18

- La classe `java.net.InetAddress`
  - l'objet **`InetAddress`** encapsule la séquence de 4 octets
  - méthodes qui offrent les services d'un DNS
- Exemple

```
import java.net.*;

public class InetAddressTest {
  public static void main(String[] args) {
    try {
      if (args.length > 0) {
        String host = args[0];
        InetAddress[] addresses = InetAddress.getAllByName(host);
        for (int i = 0; i < addresses.length; i++)
          System.out.println(addresses[i]);
      }
      else {
        InetAddress localHostAddress = InetAddress.getLocalHost();
        System.out.println(localHostAddress);
      }
    }
    catch (Exception e) { e.printStackTrace(); }
  }
}
```

Java

Java.net.InetAddress

19

- `static InetAddress getByName(String host)`
  - récupère l'addresse IP associée à un nom d'hôte
- `static InetAddress[] getAllByName(String host)`
  - récupère toutes les addresses IP associées à un nom d'hôte
- `static InetAddress getLocalHost()`
  - récupère l'addresse IP de l'ordinateur local
- `byte[] getAddress()`
  - renvoie un tableau d'octets contenant une addresse numérique
- `String getHostAddress`
  - renvoie une chaîne de caractères sous la forme de valeurs décimales séparées par des points
- `String getHostName()`
  - renvoie le nom de l'ordinateur

Java

Java.net.ServerSocket

20

- Elle sert à mettre en place un serveur sur une machine (@IP)
- Le relier à un port (port logique de la machine, un entier)
- Permet de traiter les requêtes de demande de connexion:
  - Les requêtes sont stockées dans le tampon du port et traitées une à une (clients mis en attente)
  - Le résultat de chaque traitement de requête est un objet Socket reliant le serveur et le client

Java.net.ServerSocket

21

- `ServerSocket(int port) throws IOException`
  - crée une socket serveur qui examine un port
- `Socket accept() throws IOException`
  - attend une connexion
  - bloque le thread courant jusqu'à une demande de connexion
  - renvoie un objet **`Socket`** pour communiquer avec le client
- `void close() throws IOException`
  - ferme la socket du serveur

Un premier serveur

22

- **Affichage en écho des messages reçus du client**
  - Implantation Java

```
Bonjour, tapez OK pour sortir
Bonjour, comment allez-vous
Echo : Bonjour, comment allez-vous
Très bien et vous ?
Echo : Très bien et vous ?
OK
Echo : OK
```

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

public class EchoServer {
  public static void main(String[] args) {
    try {
      ServerSocket s = new ServerSocket(4444);
      Socket incoming = s.accept();
      BufferedReader in = new BufferedReader(new InputStreamReader(incoming.getInputStream()));
      PrintWriter out = new PrintWriter(incoming.getOutputStream(), true /* autoFlush */);
      out.println( "Bonjour, tapez OK pour sortir" );
      boolean done = false;
      while (!done) {
        String line = in.readLine();
        if (line == null) done = true;
        else {
          out.println("Echo: " + line);
          if (line.trim().equals("OK")) done = true;
        }
      }
      incoming.close();
    }
    catch (Exception e) {  e.printStackTrace(); }
  }
}
```

Un serveur multi-threadé

23

- Problème
  - impossible de servir plusieurs clients en même temps
- Solution
  - créer un nouveau **thread** à chaque nouvelle connexion
  - chaque thread est chargé de la gestion entre le serveur et un client particulier
  - cette gestion s'effectue dans la méthode **`run`** du thread
- Implantation

```
...
ServerSocket s = new ServerSocket(8189);

for (;;) {  // while(true){}
  Socket incoming = s.accept( );
  Thread t = new ThreadedEchoHandler(incoming);
  t.start();
}
```


Rappel sur Les Flux d'Entrée/Sortie

Tout sur les flux java

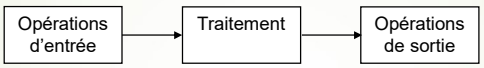
Bachir Djafri

Université d'Évry – Paris-Saclay

Rappels sur les flux Java (`java.io`)

25

- Les entrées/sorties
  - Communication entre le programme et le monde extérieur



- Philosophie en Java
  - les opérations E/S peuvent avoir des origines très diverses
    - Entrées : clavier, fichier, réseau, autre programme ou application
    - Sorties : écran, fichier, réseau, autre programme ou application
  - les flux ont une interface standard
    - les opérations effectuées sont indépendantes de la nature du périphérique concerné

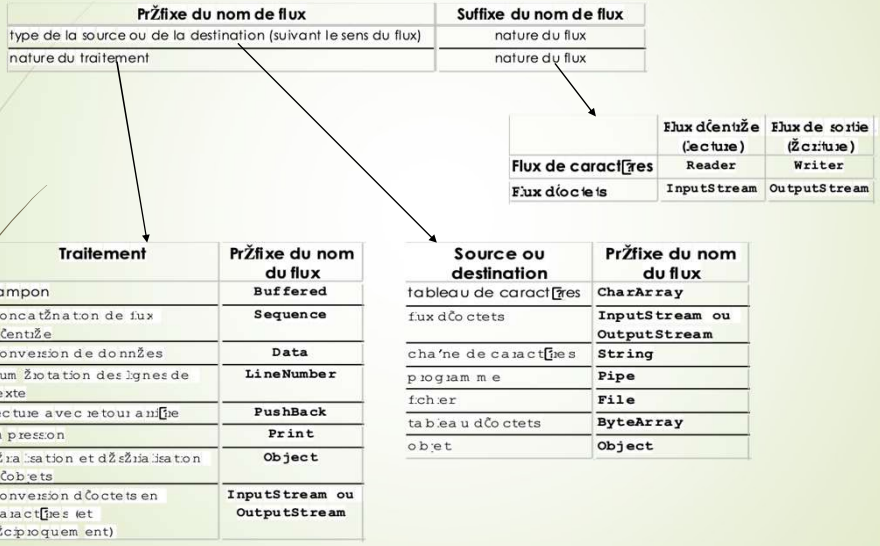
Différents types de flux

26

- Flux à accès séquentiel
  - les données sont traitées les unes après les autres dans un ordre qui ne peut pas être changé
    - majorité des flux Java
    - flux unidirectionnels (lecture OU écriture)
    - diverses catégories en fonction de la nature (type) des données, du sens de transfert, du type de source ou de destination
- Flux à accès indexé
  - permet d'accéder à un fichier en choisissant directement la position (méthode **`seek`**) à laquelle lire ou écrire
  - flux bidirectionnel = un seul flux permet à la fois la lecture et l'écriture
  - une seule classe = **`RandomAccessFile`**

Construction des noms de flux Java (1)

27



27

Construction des noms de flux Java (2)

28

- Liste des flux séquentiels de **`java.io`** (les classes en italique sont abstraites)

Reader	*Writer*	*InputStream*	*OutputStream*
BufferedReader	BufferedWriter	BufferedInputStream	BufferedOutputStream
CharArrayReader	CharArrayWriter	ByteArrayInputStream	ByteArrayOutputStream
FileReader	FileWriter	DataInputStream	DataOutputStream
FilterReader	*FilterWriter*	FileInputStream	FileOutputStream
InputStreamReader	InputStreamWriter	*FilterInputStream*	*FilterOutputStream*
LineNumberReader		ObjectInputStream	ObjectOutputStream
PipedReader	PipedWriter	PipedInputStream	PipedOutputStream
	PrintWriter		PrintStream
PushBackReader		PushbackInputStream	
StringReader	StringWriter	SequenceInputStream	

28

Opérations courantes sur des flux d'octets

29

Signature de la méthode	Classes concernées	Donnée traitée	Type et valeur retournés	Exceptions levées
read()	**classes dérivées de InputStream**	octet extrait des 8 bits de poids faibles de la valeur retournée	int: • l'octet lu • -1, si fin de flux	IOException : erreur de lecture
write(int)	**classes dérivées de OutputStream**	octet extrait des 8 bits de poids faibles de l'argument		IOException : erreur d'écriture
read(byte[])	**classes dérivées de InputStream**	tableau d'octets	int: • le nombre d'octets lus • -1, si fin de flux	IOException : erreur de lecture
write(byte[])	**classes dérivées de OutputStream**	tableau d'octets		IOException : erreur d'écriture
readDouble()	**DataInputStream**	double	double: valeur lue	EOFException : fin de flux IOException : erreur de lecture
writeDouble(double)	**DataOutputStream**	double		IOException : erreur d'écriture
readTypeSimple()	**DataInputStream**	*TypeSimple* (pour Boolean, Char, Double, Float, Int, Long, Short)	*TypeSimple* : valeur lue	EOFException : fin de flux IOException : erreur de lecture
writeTypeSimple()	**DataOutputStream**	*TypeSimple* (pour Boolean, Char, Double, Float, Int, Long, Short)		IOException : erreur d'écriture

29

Lecture/écriture d'octets dans un fichier

30

```
// création d'un flux de lecture d'octets dans un fichier
DataInputStream fichEntree = new DataInputStream(new
FileInputStream("fichEntree.bin"));

void loadFile(DataInputStream entree) throws Exception {
  int valeurEntiere = entree.readInt();
  double valeurDouble = entree.readDouble();
  boolean valeurBooleenne = entree.readBoolean();
  byte valeurByte = entree.readByte();
  entree.close() ;
}

// création d'un flux d'écriture d'octets dans un fichier
DataOutputStream fichSortie = new DataOutputStream(new
FileOutputStream("fichSortie.bin"));

void toFile(DataOutputStream sortie) throws Exception {
  sortie.writeInt(valeurEntiere);
  sortie.writeDouble(valeurDouble);
  sortie.writeBoolean(valeurBooleenne);
  sortie.writeByte(valeurByte);
  sortie.close();
}
```

30

Opérations courantes sur des flux de caractères

31

Signature de la méthode	Classes concernées	Donnée traitée	Type et valeur retournés	Exceptions levées
`read()`	`Reader`	caractère	int: • le caractère lu • -1, si fin de flux	OExceptions : erreur de lecture
`write(int)`	`Writer`	caractère extrait des 16 bits de poids faible de l'argument		OExceptions : erreur d'écriture
`read(char[])`	`Reader`	tableau de caractères	int: • le nombre de caractères lus • -1, si fin de flux	OExceptions : erreur de lecture
`write(char[])`	`Writer`	tableau de caractères donnée en argument		OExceptions : erreur d'écriture
`readLine()`	`BufferedReader`	chaîne de caractères	String: • la chaîne lue sans les caractères de terminaison • null, si fin de flux	OExceptions : erreur de lecture
`write(String)`	`Writer`	chaîne de caractères donnée en argument		OExceptions : erreur d'écriture
`print(arg)` ou `println(arg)`	`PrintWriter`	arg: donnée de type simple, chaîne de caractères ou objet		aucune, CheckError() renvoie true en cas d'erreur

31

Lecture/écriture de caractères dans un fichier

32

```
// création d'un flux de lecture de caractères dans un fichier
BufferedReader fichEntree = new BufferedReader(new FileReader("fichEntree.txt"));

// utilisation d'un flux de lecture de caractères dans un fichier
void loadFile(BufferedReader entree) throws Exception {
  int valeurEntiere = Integer.valueOf(entree.readLine()).intValue();
  double valeurDouble = Double.valueOf(entree.readLine()).doubleValue() ;
  boolean valeurBooleenne = Boolean.valueOf(entree.readLine()).booleanValue() ;
  byte valeurByte = Byte.valueOf(entree.readLine()).byteValue() ;
  entree.close() ;
}

// création d'un flux d'écriture de caractères dans un fichier
PrintWriter fichSortie = new PrintWriter(new FileWriter("fichSortie.txt"));

void toFile(PrintWriter sortie) throws Exception {
  sortie.println(valeurEntiere);
  sortie.println(valeurDouble);
  sortie.println(valeurBooleenne);
  sortie.println(valeurByte);
  sortie.close();
}
```

32