Systèmes et Applications Distribués
Communication (Partie 2)

Synchronisation par passage de messages :
application sur les « socket » de Java

1

Plan du cours

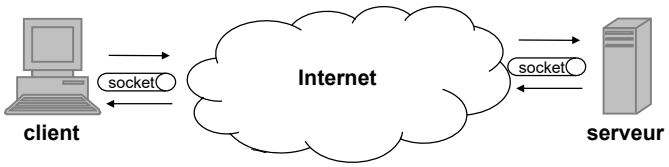
- Introduction
  - Définitions
  - Problématique
  - Architectures de distribution
- Distribution intra-applications
  - Notion de processus
  - Programmation multi-thread
- Distribution inter-applications et inter-machines
  - Les Sockets
  - middlewares par appel de procédures distantes (RPC)
  - middlewares par objets distribués (Java RMI)
  - middlewares par objets distribués hétérogènes (CORBA/GRPC)
- Conclusion

2

Distribution inter-applications et inter-machines

Le modèle des sockets [N. Melab (LIFL)]

- Interface (point de communication) client/serveur utilisée à l'origine dans le monde UNIX et TCP/IP
  - étendue aux PCs (Winsock) et mainframes
  - 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)

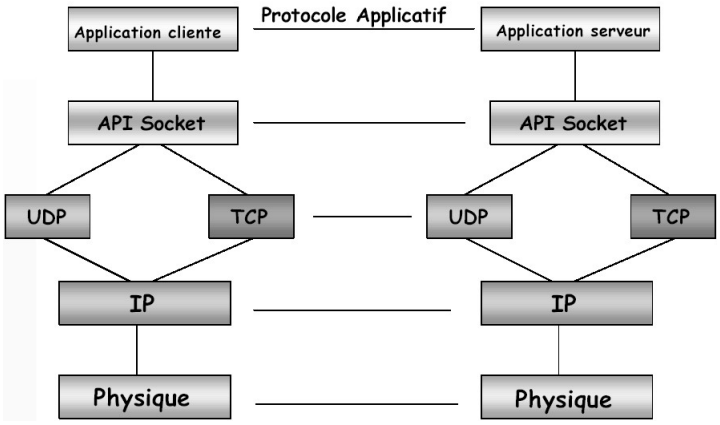


3

3

Distribution inter-applications et inter-machines – les sockets

Sockets et OSI [N. Melab (LIFL)]



4

4

Distribution inter-applications et inter-machines – les sockets

Protocoles TCP et UDP [N. Melab (LIFL)]

- TCP
  - Garantie d'arrivée dans l'ordre de paquets de données
  - Fiabilité de transmission
  - Lenteur des transmissions (http par exemple)
  - Vu comme un «service téléphonique»
- UDP
  - Arrivée dans le bon ordre non garantie
  - Non fiabilité des transmissions
  - Rapidité des transmissions
  - Applications audio/vidéo
  - Vu comme un «service postal»

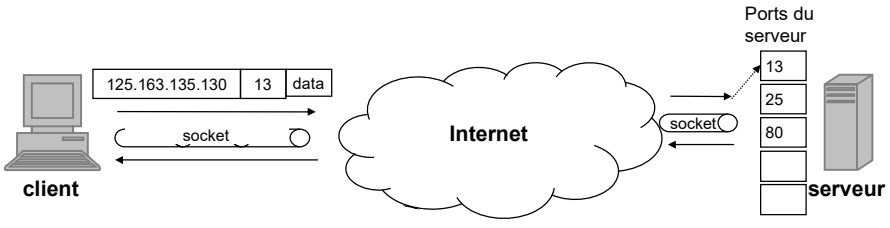
5

5

Distribution inter-applications et inter-machines – les sockets

Connexion réseau

- Adresse Internet de la machine (@ IP)
- Numéro du port (int)



6

6

Distribution inter-applications et inter-machines – les sockets

Notion de port [N. Melab (LIFL)]

- 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 (série et parallèle)
- Désignation des ports
  - Port : numéro 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

7

7

Distribution inter-applications et inter-machines – les sockets

Adresses Internet [N. Melab (LIFL)]

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

8

8

Distribution inter-applications et inter-machines – les sockets

Client-serveur en mode connecté [M. Riveill (INPG)]

- 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

9

9

Distribution inter-applications et inter-machines – les sockets

Mode connecté : caractéristiques [M. Riveill (INPG)]

- 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

10

10

Distribution inter-applications et inter-machines – les sockets

Caractéristiques du mode connecté [M. Riveill (INPG)]

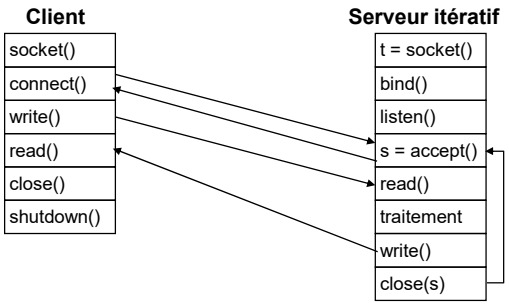
- Contrainte
  - le client doit avoir accès à l'adresse du serveur (adresse IP et numéro de port)
- Modes de gestion des requêtes
  - itératif : le processus serveur traite les requêtes les unes après les autres
  - concurrent : par création de processus fils pour les échanges de chaque requête

11

11

Distribution inter-applications et inter-machines – les sockets

Enchaînement des opérations (1) [M. Riveill (INPG)]

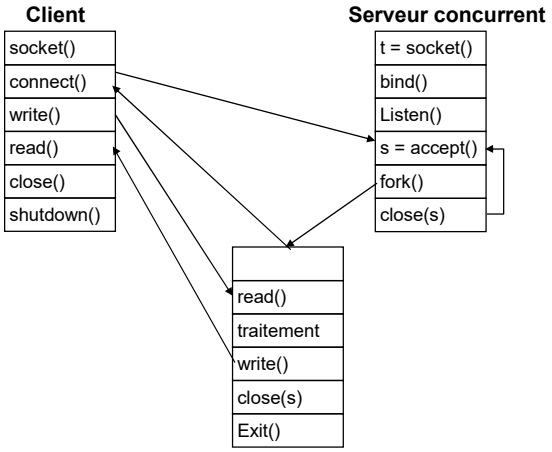


12

12

Distribution inter-applications et inter-machines – les sockets

Enchaînement des opérations (2) [M. Riveill (INPG)]



13

13

Distribution inter-applications et inter-machines – les sockets

Client-serveur en mode non connecté [M. Riveill (INPG)]

- le client peut envoyer des appels au serveur à n'importe quel moment
  - mode assez léger orienté
    - traitement non ordonné des appels
    - absence de mémoire entre appels successifs (serveur sans données rémanentes et sans état)
  - exemple :
    - calcul de fonction numérique
    - DNS
    - NFS

14

14

Distribution inter-applications et inter-machines – les sockets

Mode non connecté : caractéristiques [M. Riveill (INPG)]

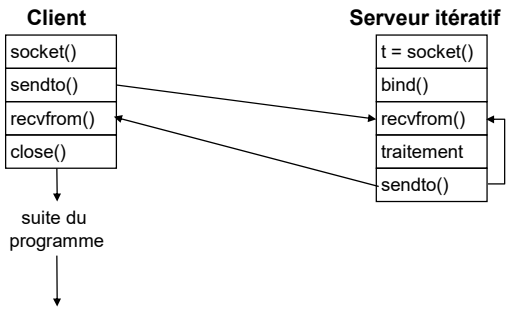
- Caractéristiques
  - pas d'établissement préalable d'une connexion
  - adapté aux applications pour lesquelles les réponses aux requêtes des clients sont courtes (un message)
  - protocole de transport utilisé : UDP
  - mode d'échange par messages : le récepteur reçoit les données suivant le même découpage que celui effectué par l'émetteur
- Contraintes
  - le client doit avoir accès à l'adresse du serveur (adresse IP et numéro de port)
  - pour répondre à chaque client, le serveur doit en récupérer l'adresse : il faut pour cela utiliser les primitives **`sendto`** et **`recvfrom`**
- Mode 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

15

15

Distribution inter-applications et inter-machines – les sockets

Enchaînement des opérations (1) [M. Riveill (INPG)]

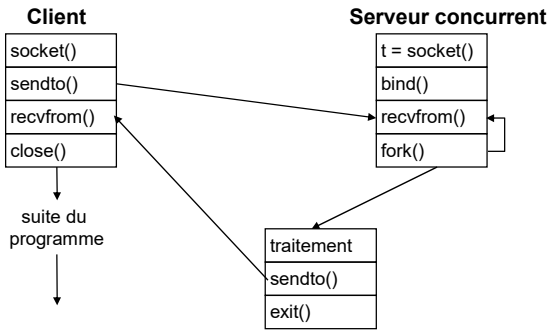


16

16

Distribution inter-applications et inter-machines – les sockets

Enchaînement des opérations (2) [M. Riveill (INPG)]



17

17

– Java

la communication Sous Java

- La classe Socket (et ServerSocket)
  - Mode connecté
  - TCP.
  - Un mélange entre RDV pour obtenir un canal asynchrone (cf plus loin)
- La classe DatagramSocket (et DatagramPacket)
  - Mode non connecté
  - UDP.
  - Asynchrone

18

18

- Mode connecté

19

19

-java

les sockets TCP sous Java

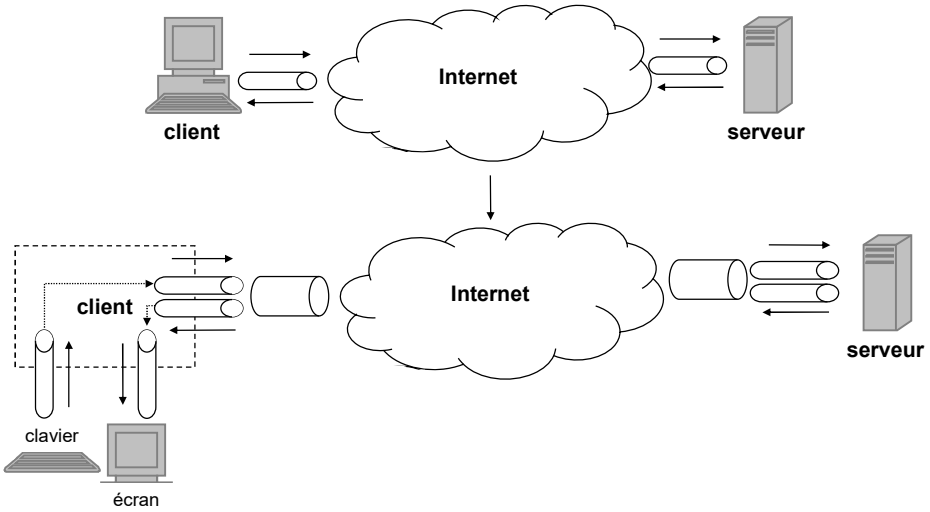
- Deux classes interviennent:
- java.net.Socket
  - Coté client (mais aussi coté serveur)
  - Elle permet une communication 1-1
  - C'est un couple de canaux (asynchrone)
- java.net.ServerSocket
  - Utilisé uniquement coté serveur
  - C'est un point d'entrée
  - Req : demande d'établissement d'une connexion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

Rappels sur les Flux

20

Distribution inter-applications et inter-machines – les sockets

Les sockets Java

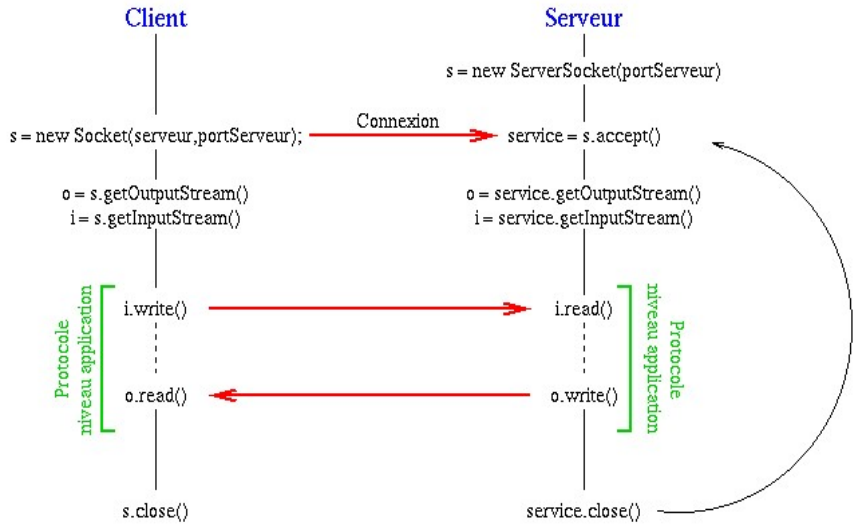


21

21

-java

Les sockets sous Java



Rappels sur les Flux

22

-java

Java.net.Socket (TCP)

- `Socket(String host, int port)`
  - crée une socket et la connecte à un port de l'ordinateur distant
- `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

23

23

Distribution inter-applications et inter-machines – les sockets

Un premier client

- Interrogation du service « date » d'un serveur

```
$ telnet time-A.timefreq.bldrdoc.gov 13
$ 50692 05-01-10 10:27:15 50 0 0 50.0 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();
    }
  }
}
```

24

24

– Java

Attention la création d'un socket est une action bloquante

- 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
}
```

25

25

– Java

Les adresses Internet

- 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();
    }
  }
}
```

26

26

– Java

Java.net.InetAddress

- `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

27

27

-java

Java.net.ServerSocket

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

Rappels sur les Flux

28

Distribution inter-applications et inter-machines – les sockets

Java.net.ServerSocket

- `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

29

29

Distribution inter-applications et inter-machines – les sockets

Un premier serveur

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

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

public class EchoServer {
  public static void main(String[] args) {
    try {
      ServerSocket s = new ServerSocket(8189);
      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();
    }
  }
}
```

```
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
```

30

30

Distribution inter-applications et inter-machines – les sockets

Un serveur multi-threadé

- 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();
}
```

31

31

- Mode non connecté

32

32

-java

Les socket UDP sous Java

- Deux classes interviennent :
- DatagramPacket
  - Cette classe permet de créer des objets qui contiendront les données envoyées ou reçues ainsi que l'adresse de destination ou de provenance du datagramme.
  - Deux constructeurs disponibles (un pour le coté client et un pour le coté serveur).
- DatagramSocket
  - Cette classe permet de créer des sockets UDP qui permettent d'envoyer et de recevoir des datagrammes UDP.
  - La création d'une DatagramSocket est essentielle aussi bien coté client que coté serveur.

33

33

-java

DatagramPacket

- Constructeur pour recevoir les données (coté serveur) :
  - `DatagramPacket(byte buffer[], int taille)`
  - - buffer pour mettre les données
  - - taille maximale à lire (le reste est perdu)
- Constructeur pour envoyer les données
  - `DatagramPacket(byte buffer[], int taille, InetAddress adresse, int port)`
  - - buffer pour mettre les données
  - - taille maximale à envoyer (le reste est pas envoyé)
  - - plus l'adresse et le port du récepteur

34

34

-java

DatagramSocket (1)

- Constructeur pour client
  - `public DatagramSocket ()`
- - on ne spécifie pas le port d'attachement
- - notez bien que l'adresse de la destination ne figure pas dans la socket mais dans le DatagramPacket
- Constructeur pour le serveur
  - `public DatagramSocket (int port)`
- - on spécifie le port
- - il existe aussi un autre constructeur où on spécifie aussi l'adresse

35

35

-java

DatagramSocket (2)

- Envoi de message (asynchrone = non bloquant)
  - `public void send(DatagramPacket data) throws...`
- - envoi des données de data
- au serveur dont l'adresse est spécifiée dans data
- Réception (synchrone = bloquante)
- `public synchronized void receive(DatagramPacket data) throws...`
- - reçoit les données dans data
- - il est possible de mettre une garde sur la réception (méthode SetTimeout(int x))
- - après la réception data contient : les données, la taille réelle et l'adresse plus le port de l'émetteur.

36

36

Exemple Echo : le Serveur

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

class ServeurEcho
{
 final static int port = 8532;
 final static int taille = 1024;
 final static byte buffer[] = new byte[taille];

 public static void main(String argv[]) throws Exception
 {
 DatagramSocket socket = new DatagramSocket(port);
 while(true)
 {
 DatagramPacket data = new DatagramPacket(buffer,buffer.length);
 socket.receive(data);
 System.out.println(data.getAddress());
 socket.send(data);
 }
 }
}
```

37

37

Exemple Echo : le Client

```
import java.io.*;
import java.net.*;
public class ClientEcho
{
 final static int taille = 1024;
 final static byte buffer[] = new byte[taille];
 public static void main(String argv[]) throws Exception
 {
 InetAddress serveur = InetAddress.getByName(argv[0]);
 int length = argv[1].length();
 byte buffer[] = argv[1].getBytes();
 DatagramPacket dataSent = new DatagramPacket(buffer,length,serveur,ServeurEcho.port);
 DatagramSocket socket = new DatagramSocket();

 socket.send(dataSent);

 DatagramPacket dataRecieved = new DatagramPacket(new byte[length],length);
 socket.receive(dataRecieved);
 System.out.println("Data recieved : " + new String(dataRecieved.getData()));
 System.out.println("From : " + dataRecieved.getAddress() + ":" + dataRecieved.getPort());
 }
}
```

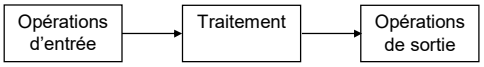
38

38

Distribution inter-applications et inter-machines

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

- 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
    - Sorties : écran, fichier, réseau, autre programme
  - les flux ont une interface standard
    - les opérations effectuées sont indépendantes de la nature du périphérique concerné

39

39

Distribution inter-applications et inter-machines – Rappels sur les flux Java

Différents types de flux

- 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 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`**

40

40

Distribution inter-applications et inter-machines – Rappels sur les flux Java

Construction des noms de flux Java (1)

Préfixe du nom de flux	Suffixe du nom de flux
type de la source ou de la destination (suivant le sens du flux)	nature du flux
nature du traitement	nature du flux

	Flux d'entrée (lecture)	Flux de sortie (écriture)
Flux de caractères	Reader	Writer
Flux d'octets	InputStream	OutputStream

Traitement	Préfixe du nom du flux
tampon	Buffered
concaténation de flux d'entrée	Sequence
conversion de données	Data
numérotation des lignes de texte	LineNumber
lecture avec retour arrière	PushBack
impression	Print
sérialisation et désérialisation d'objets	Object
conversion d'octets en caractères (et réciproquement)	InputStream ou OutputStream

Source ou destination	Préfixe du nom du flux
tableau de caractères	CharArray
flux d'octets	InputStream ou OutputStream
chaîne de caractères	String
programme	Pipe
fichier	File
tableau d'octets	ByteArray
objet	Object

41

41

Distribution inter-applications et inter-machines – Rappels sur les flux Java

Construction des noms de flux Java (2)

- 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	

42

42

Distribution inter-applications et inter-machines – Rappels sur les flux Java

Opérations courantes sur des flux de caractères

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	OException : erreur de lecture
`write(int)`	`Writer`	caractère extrait des 16 bits de poids faible de l'argument		OException : erreur d'écriture
`read(char[])`	`Reader`	tableau de caractères	int : • le nombre de caractères lus • -1, si fin de flux	OException : erreur de lecture
`write(char[])`	`Writer`	tableau de caractères donnée en argument		OException : erreur d'écriture
`readLine()`	`BufferedReader`	chaîne de caractères	String : • la chaîne lue sans les caractères de terminaison • null, si fin de flux	OException : erreur de lecture
`write(String)`	`Writer`	chaîne de caractères donnée en argument		OException : erreur d'écriture
`print(arg)` ou `println(arg)`	`PrintWriter`	arg : donnée de type simple, chaîne de caractères ou objet		aucune, CheckError) renvoyée en cas d'erreur

Distribution inter-applications et inter-machines – Rappels sur les flux Java

Opérations courantes sur des flux d'octets

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	OException : erreur de lecture
`write(int)`	`classes dérivées de OutputStream`	octet extrait des 8 bits de poids faibles de l'argument		OException : erreur d'écriture
`read(byte[])`	`classes dérivées de InputStream`	tableau d'octets	int : • le nombre d'octets lus • -1, si fin de flux	OException : erreur de lecture
`write(byte[])`	`classes dérivées de OutputStream`	tableau d'octets		OException : erreur d'écriture
`readDouble()`	`DataInputStream`	double	double : la valeur lue	EOFException : fin de flux OException : erreur de lecture
`writeDouble(double)`	`DataOutputStream`	double		OException : erreur d'écriture
`readTypeSimple()`	`DataInputStream`	`TypeSimple` (pour Boolean, Char, Double, Float, Int, Long, Short)	`TypeSimple` : la valeur lue	EOFException : fin de flux OException : erreur de lecture
`writeTypeSimple()`	`DataOutputStream`	`TypeSimple` (pour Boolean, Char, Double, Float, Int, Long, Short)		OException : erreur d'écriture

Distribution inter-applications et inter-machines – Rappels sur les flux Java

Lecture/écriture de caractères dans un fichier

```
// 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();
}
```

45

45

Distribution inter-applications et inter-machines – Rappels sur les flux Java

Lecture/écriture d'octets dans un fichier

```
// 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();
}
```

Retour aux sockets 46

46

Quelques références

- Ce cours a été réalisé en se basant sur :
- Cours Guillaume Hutzler pour la partie sur les Socket
  - http://www.lami.univ-evry.fr/%7Ehutzler/Cours/CPAR/Sockets.pdf
- Une interprétation libre du chapitre 10 du livre :
  - Concurrency: State Models & Java Programs de *Jeff Magee & Jeff Kramer*
  - *les 10 premiers chapitres sont dispo en ligne ainsi que les slides qui vont avec (En Anglais ;-).*
    - *http://www-dse.doc.ic.ac.uk/concurrency/*

47

47