Les sockets java

Introduction

- Point d'entrée entre 2 applications du réseau
- Permet l'échange de donnée entre elles à l'aide des mécanismes d'E/S (java.io + java.net)

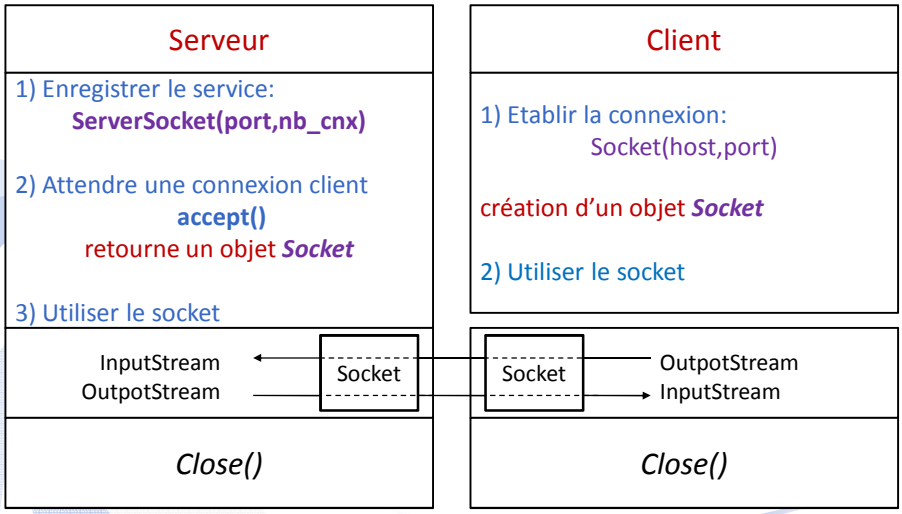
- Différents types de sockets

- Stream Sockets (TCP)
  - ✓ établir une communication en mode connecté
  - ✓ si connexion interrompue : applications informées

- Datagram Sockets (UDP)

  - ✓ établir une communication en mode non connecté
  - ✓ données envoyées sous forme de paquets
  - ✓ indépendants de toute connexion. Plus rapide, moins fiable que TCP

Les sockets en mode connecté



Serveur TCP

- il utilise la classe **java.net.ServerSocket** pour accepter des connexions de clients

- Quand un client se connecte à un port sur lequel un **ServerSocket** écoute,
- **ServerSocket** crée une nouvelle instance de la classe **Socket** pour supporter les communications côté serveur :

```
int port = …;
ServerSocket server = new ServerSocket(port);
Socket connection = server.accept();
```

⇦ Appel bloquant

java.net.ServerSocket

```
final int PORT = …;
try {
ServerSocket serveur = new ServerSocket(PORT,5);
    while (true) {
        Socket socket = serveur.accept();
    }
}
 catch (IOException e){
    ….
}
```

Client TCP

- Le client se connecte au serveur en créant une instance de la classe **java.net.Socket** : connexion synchrone

```
String host = …;
int port = …;
Socket connection = new Socket (host,port);
```

- Le socket permet de supporter les communications côté client
- La méthode **close()** ferme (détruit) le socket
- Les constructeurs et la plupart des méthodes peuvent générer une **IOException**
- Le serveur doit être démarré avant le client. Dans le cas contraire, si le client se connecte à un serveur inexistant, une exception sera levée après un time-out

java.net.Socket

```
String HOST = "… ";
int PORT = …;
try {
        Socket socket = new Socket (HOST,PORT);
}
try {
                socket.close(); } catch (IOException e){}
```

java.net.InetAddress

Le constructeur de l'objets Socket :

Socket (InetAddr addr, int port)

Méthodes statiques

- InetAddress getLocalHost()
- InetAddress getByName(String host_name)
- InetAddress[] getAllByName(String host_name)

Méthodes d'instances

- String getHostName ()
- byte[] getAddress ()
- String toString ()

Les flux de données (1)

- Une fois la connexion réalisée, il faut obtenir les ***streams*** d'E/S (**java.io**) auprès de l'instance de la classe ***Socket*** en cours

- Flux entrant

  ✓ obtention d'un stream
  ```
  InputSteam in = socket.getInputStream();
  ```

  ✓ création d'un stream convertissant les bytes reçus en char
  ```
  InputSteamReader reader = new InputStreamReader(in);
  ```

  ✓ création d'un stream de lecture avec tampon: pour lire ligne par ligne dans un stream de caractères
  ```
  BufferedReader istream = new BufferedReader(reader);
  ```

  ✓ lecture d'une chaîne de caractères
  ```
  String line = istream.readline();
  ```

Les flux de données (2)

- Flux sortant

  ✓ Obtention du flot de données sortantes : bytes
  ```
  OutputSteam out = socket.getOutputStream();
  ```

  ✓ création d'un stream convertissant les bytes en chaînes de caractères
  ```
  PrintWriter ostream = new PrintWriter(out);
  ```

  ✓ envoi d'une ligne de caractères
  ```
  ostream.println(str);
  ```

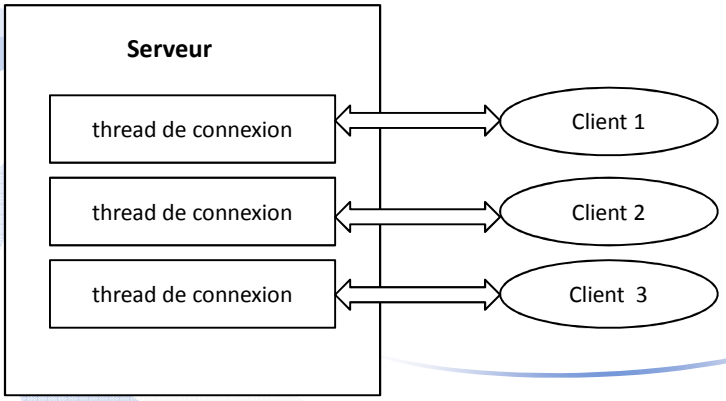
  ✓ envoi effectif sur le réseau des bytes (important)
  ```
  ostream.flush();
  ```

Les flux de données (3)

```
try {
    Socket socket = new Socket(HOST,PORT);
    //Lecture du flux d'entrée en provenance du serveur
    InputStreamReader reader = new InputStreamReader(socket.getInputStream());
    BufferedReader istream = new BufferedReader(reader);
    String line = istream.readLine();
    System.out.println(line);
    //Echo la ligne lue vers le serveur
    PrintWriter ostream = new PrintWriter(socket.getOutputStream());
    ostream.println(line);
    ostream.flush();
}catch (IOException e) {…}
finally {try{
        socket.close();
        }catch (IOException e){}}
```

Les serveurs multiclients

- Le serveur utilise une classe **Connexion** implémentant l'interface **Runnable** (thread) pour gérer les échanges de données en tâche de fond. C'est ce thread qui réalise le service demandé (**java.lang.Thread**).



Les serveurs multiclients (2)

```
//Déclaration de la clase connexion pour l'exécution du thread
class Connexion implements Runnable
{……
    public Connexion (Socket s) {
    this.s = s;
    try{
        in=new BufferedReader( new InputStreamReader(s.getInputStream()));
        out = new PrintWriter(s.getOutputStream());
} catch (IOExeption e) {…}
}
```

Les serveurs multiclients (3)

```
//Méthode run de la classe Connexion
public void run() {
try{
    while (true) {
        String ligne = in.readLine();
        if (ligne == null) break; //fin de connexion côté client
        output.println(ligne);
        out.flush();
}
} catch (IOExeption e) {…}
finally {try{
        s.close();
}catch (IOException e){}}
```

Les serveurs multiclients (4)

```
Class MyServer {
....
//Serveur multhreads
public static void main (String args[]){
try{
        ServerSocket serveur = new ServerSocket(PORT, 10);
        while (true) {
        //accepter une connexion
        Socket socket = serveur.accept();
        // créer un thread : pour échanger les données avec le client
        Connexion c = new Connexion(socket);
        Thread processus_connexion = new Thread(c);
        processus_connexion.start();
}
} catch (IOExeption e) {…}
…}
}
```

Socket en mode datagramme

- Il faut utiliser les classes **DatagramPacket** et **DatagramSocket**
- Ces objets sont initialisés différemment selon qu'ils sont utilisés pour envoyer (***send()***) ou recevoir (***receive ()***) des paquets

Envoi de datagrammes

1. Créer un DatagramPacket en spécifiant :
   - Les données à envoyer
   - Leur longuer
   - La machine réceptrice et le port
2. Utiliser la méthode send(DatagramPacket) de DatagramSocket

   Pas d'arguments pour le constructeur car toutes les informations se trouvent dans le paquet envoyé

Envoi de datagrammes (2)

```
//Machine destinataire
InetAddress address = InetAddress.getByName(" mpssr.isitcom.tn");
static final int PORT = 4562;
//Création du message à envoyer
String s = new String (" Hello World");
int longueur = s.length();
byte[] message = new byte[longueur];
s.getBytes(0, longueur, message, 0);
//Initialisation du paquet avec toutes les informations
DatagramPacket paquet = new DatagramPacket(message,longueur, address,PORT);
//Création du socket et envoi du paquet
DatagramSocket socket = new DatgramSocket();
socket.send(paquet);....
```

Réception de datagrammes

1. Créer un DatagramSocket qui écoute sur le port de la machine du destinataire
2. Créer un DatagramPacket pour recevoir les paquets envoyés par le serveur
   - ➔ Dimensionner le buffer assez grand
3. Utiliser la méthode receive() de DatagramPacket
   - ➔ Cette méthode est bloquante

Réception de datagrammes (2)

```
//Définir un buffer de réception
byte[] buffer = new byte[1024];
//On associe un paquet à un buffer vide pour la réception
DatagramPacket paquet =new DatagramPacket(buffer, buffer.length());
//On crée un socket pour écouter sur le port
DatagramSocket socket = new DatgramSocket(PORT);
while (true) {
    //attente de réception
    socket.receive(paquet);
    //affichage du paquet reçu
    String s = new String(buffer,0,0,paquet.getLength());
    System.out.println("Paquet reçu : "+ s + "du hôte"  + paquet.get Address() +
        "sur le port "  + paquet.getPort() );
}
```

Sérialisation des objets

- Cela consiste à stocker un objet sous une certaine forme en dehors de la JVM
- Pourquoi ?
  - Avoir une persistance des données
  - Les objets sérialisés peuvent être stockés dans des fichiers ou des bases de données
  - Pour pouvoir transmettre un objet
    - Sur le réseau
    - D'une application à une autre (Une application calcule un certain objet et le met ensuite à disposition d'une autre application)
- Autres appélations: *Marshalling / Unmarshalling*
  Linéarisation
- Existe aussi dans d'autres langages OO …
- Peut être passé à des objets de type *ObjectInputStream* ou *ObjectOutputStream*

Sérialisation des objets (2)

- Les objets qu'on souhaite sérialiser doivent implémenter l'interface *Serializable*
- Un objet n'implémentant pas *Serializable* ne peut pas être srialisé
  - ➔ **Tous les attributs d'un objet implémentant *Serializable* doivent aussi implémenter *Serializable***
- L'objet est décomposé en éléments de plus en plus petits (jusqu'à arriver aux éléments de base) et chacun de ces éléments est encodé
- On peut ensuite écrire des objets grâce à des flux d'écriture / lecture d'objets :
  - Classe ***ObjectOutputStream***
    - Méthode void writeObject(Object o)
    - Si l'objet ou un des objets appartenant à ces champs n'est pas sérialisable, une exception est levée
  - Classe ***ObjectIntputStream***
    - Méthode Object readObject()
    - Il faut caster l'objet dans la classe désirée

Sérialisation des objets *Exemple*

```
// création d'une classe Serializable

public class Truc implements Serializable {
      // La classe String est Serializable, donc le champ1 est légal
      private static final long serialVersionUID = 354054054054L;
      private String champ1;
      private int      champ2;

      // La classe Connection n'est pas Serializable,  il faut donc retirer ce champ
      //de la serialization
      private transient Connection con ;

      public Truc(String m, int i) {
         this.champ1 = m;
         this.champ2 = i;
      }
      String toString()
      {   return (this.champ1 + « : » + this.champ2);
      }
}
```

Sérialisation des objets *Exemple (2)*

- Sérialisation

```
//Bout de code du serveur
client = server.accept();
…
// Récupérer le flux de sortie de la socket client
out = new ObjectOutputStream(client.getOutputStream());

// Créer  un objet Truc et l'envoyer vers le client
Truc  truc =  new Truc("Bonjour", 10);
out.writeObject(truc).
out.flush();
…
} catch (IOException e) {  e.printStackTrace();
                           System.exit(-1); }
 catch (Exception e) {
                           e.printStackTrace();
                           System.exit(-1); }
```

Sérialisation des objets *Exemple (3)*

- Désérialisation

…

```
// Récupérer le flux d'entrée de la socket client
in = new ObjectInputStream(client.getInputStream());

// Caster l'objet envoyé par le client en un objet de type Truc
Truc  truc = (Truc) in.readObject();
System.out.println(truc);
…
 } catch (IOException e) {  e.printStackTrace();
                            System.exit(-1); }
 catch (Exception e) {
                            e.printStackTrace();
                            System.exit(-1); }
```