Programmation Réseau
Socket

Plan

- ProTocole TCP
- ProTocole UDP

Introduction

- L'API "**sockets**" est une bibliothèque de Classes de communication entre machines sur TCP/IP contenu dans le paquetage *java.net*
- Le mode connecté => protocole TCP. Le protocole établit une connexion virtuelle et se charge alors de maintenir l'intégrité de la communication et de gérer les erreurs de transmission.
- Le mode non connecté => protocole UDP. Ce protocole fait l'envoi au mieux ("best effort"). C'est à l'application de maintenir la qualité de la transmission.

Le PORT

On rappel qu'un **service** est un programme qui fonctionne en permanence (démon sous UNIX) dès le démarrage de l'ordi.

- **Il** est également à l'écoute du réseau pour capturer la moindre requête venant de l'extérieur.
- Il est possible d'avoir plusieurs services sur une même machine.
- Pour les différencier, nous utilisons un numéro qui est appelé numéro de port
- Les ports numérotés de 0 à 511 sont les "well known ports" de l'architecture TCP/IP. (FTP port 21),(SMTP port 25) , (HTTP port80).
- De 512 à 1023, on trouve les services Unix.
- Au delà, (1024 ...) ce sont les ports "utilisateurs" disponibles pour placer un service applicatif quelconque.

InetAddress

Afin que deux applications communiquent, il est nécessaire de connaître le (Nom ou IP) *de l'autre machine.*

- L'*API standard permet de les manipuler en utilisant la classe* ***InetAddress.***
- *InetAddress* ***permet de manipuler IPv4 et IPv6.***
- la plupart des *IP rencontrées sont en IPv4 et ont une notation sous la forme* ***127.0.0.1.***
- Méthode statique **getLocalHost**() : récupère son adresse IP.
- **getHostName**() : récupère le nom d'hôte.
- **getHostAddress**() :récupère l'adresse sous forme pointée.

Exemple 1

```
try {
        //Get an InetAddress by a HostName
        InetAddress inet = InetAddress.getByName("localhost");
        // Display the Inet Address : IP, Name and Loop
   System.out.println("IP : " + inet.getHostAddress());
   System.out.println("Name : " + inet.getHostName());


}
catch (UnknownHostException e) {
        // If an exception happens ...
        e.printStackTrace();
}
```

Exemple2 : avec Add IP

```
try {
        //Get an InetAddress by a HostName
        InetAddress inet = InetAddress.getByAddress(
            new byte[] { (byte) 10, (byte) 20, (byte) 72, (byte) 5});
        // Display the Inet Address : IP, Name and Loop
   System.out.println("IP : " + inet.getHostAddress());
   System.out.println("Name : " + inet.getHostName());
}
catch (UnknownHostException e) {
        // If an exception happens ...
        e.printStackTrace();
}
```

Socket

La plupart des systèmes d'exploitations proposent une abstraction pour manipuler des données réseaux que l'on nomme **socket**.

- Du point vue **bas-niveau**, un *socket se comporte de la même manière qu'un fichier.*

- Une fois une **connexion établie**, la gestion des sockets est **similaire** à la gestion des fichiers.

- Classe **Socket** possède 2 méthodes : **getOutputStream()** et **getInputStream()** qui va vous permettre de manipuler les données comme pour un fichier.

- La classe **Socket** **correspond à un *socket client et la classe* *ServerSocket* *correspond à un socket serveur.***

ServerSocket

- *La classe* ***ServerSocket*** ***est dédiée à l'attente de demande de connexion de la part d'une autre machine.***

- pour pouvoir écouter les demandes de connexion, il est nécessaire d'assigner une adresse et un port d'écoute.

```
ServerSocket server = new ServerSocket(300);

server.close();
```

- La méthode **accept()** attend jusqu'à obtenir une connexion extérieure.

```
ServerSocket server = new ServerSocket(300);
Socket client = server.accept();

client.close();
server.close();
```

ServerSocket

- Voici les méthodes les plus couramment utilisées :

ServerSocket(int port) : crée un socket serveur écoutant sur un certain port

Socket accept() : attente de connexion d'un client

void close() : fermeture du socket

void setSoTimeout(int timeout) : spécifie un timeout (en millisecondes) lors des demandes de connexion

Socket

Voici les méthodes de la classe:

Socket() : création d'un socket non connecté

Socket(String host, Int port) : Création d'un socket connecté

void close() : fermeture du socket

void connect(SocketAddress endpoint) : connexion à un serveur

InetAddress getInetAddress() : Permet d'obtenir l'adresse auquel est connecté le socket

InetAddress getLocalAddress() : Permet d'obtenir l'adresse local qu'utilise votre machine pour se connecter au serveur

Int getLocalPort() : Permet d'obtenir le port local qu'utilise la machine

Int getPort() : Permet d'obtenir le port (serveur) avec laquelle la machine s'est connectée

InputStream getInputStream() : Obtention d'un flux d'entrée

OutputStream getOutputStream() : Obtention d'un flux de sortie

void setSoTimeout(Int timeout) : Règle le timeout (en millisecondes)

Socket avec TCP

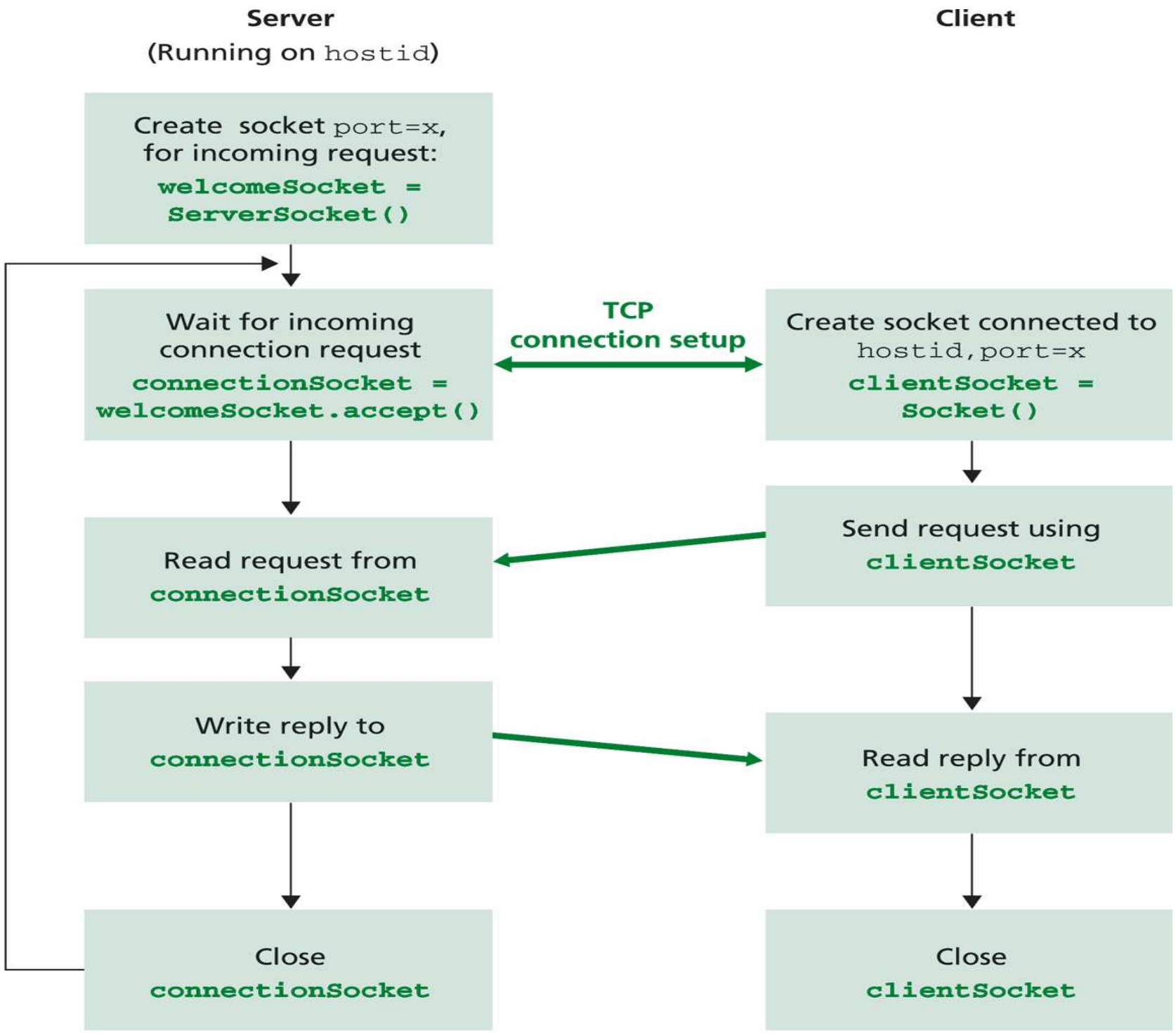
- **Etape 1 : Le serveur se met en attente**

1. le processus du serveur doit être démarrer
2. le serveur doit avoir crée une socket(port) pour les clients.

- **Etape 2 : client contact le serveur.**

1. crée Localement une Socket TCP.
2. spécifie l'adresse IP et port du serveur.

- **Etape 3 : Echange de donnée via OUT/IN PUTSTREAM.**
- **Etape 4 : Close.**


Server
(Running on hostid)
Client
Create socket port=x, for incoming request:
welcomeSocket = ServerSocket()
Wait for incoming connection request
connectionSocket = welcomeSocket.accept()
TCP connection setup
Create socket connected to hostid,port=x
clientSocket = Socket()
Send request using clientSocket
Read request from connectionSocket
Write reply to connectionSocket
Read reply from clientSocket
Close connectionSocket
Close clientSocket

Coté Serveur

```
import java.net.*;
import java.io.*;
public class Server {

    public static void main(String[] args) {
        try{
        ServerSocket ecoute=new ServerSocket(1111,6);
        Socket service=null;
        while(true){
            service=ecoute.accept();
            OutputStream os=service.getOutputStream();
            InputStream is=service.getInputStream();
            os.write(is.read());
            service.close();

        }

        }
        catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

Coté Client

```
import java.io.*;
import java.net.*;
public class Client{

    public static void main(String[] args) {
        try{
            Socket s=new Socket("localhost",1111);
            OutputStream os=s.getOutputStream();
            InputStream is=s.getInputStream();
            os.write((int)'a');
            System.out.print(is.read());
        }
        catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

Application (mini-chat)

```
public class Serveur {

    public static void main(String[] zero) {

        ServerSocket socketserver  ;
        Socket socketduserveur ;
        BufferedReader in;
        PrintWriter out;

        try {

            socketserver = new ServerSocket(2009);
            System.out.println("Le serveur est à l'écoute du port "+socketserver.getLocalPort());
            socketduserveur = socketserver.accept();
                System.out.println("Un zéro s'est connecté");
            out = new PrintWriter(socketduserveur.getOutputStream());
                out.println("Vous êtes connecté zéro !");
                out.flush();

                socketduserveur.close();
                socketserver.close();

        }catch (IOException e) {

            e.printStackTrace();
        }
    }
```

Coté Client

```
public class Client {

    public static void main(String[] zero) {


        Socket socket;
        BufferedReader in;
        PrintWriter out;
        try {

            socket = new Socket(InetAddress.getLocalHost(),2009);
                System.out.println("Demande de connexion");
                in = new BufferedReader (new InputStreamReader (socket.getInputStream()));
                String message_distant = in.readLine();
                System.out.println(message_distant);

                socket.close();

        }catch (UnknownHostException e) {

            e.printStackTrace();
        }catch (IOException e) {

            e.printStackTrace();
        }
    }
```

Coté Serveur

```
String phraseClient;  String phraseMajuscule;

  ServerSocket socketEcoute = new ServerSocket(7777);
            while(true) { // boucle infinie
                Socket socketConnexion = socketEcoute.accept();

   BufferedReader entreeDepuisClient = new BufferedReader
(new InputStreamReader(socketConnexion.getInputStream()));
 DataOutputStream  sortieVersClient = new DataOutputStream(
socketConnexion.getOutputStream());
      phraseClient = entreeDepuisClient.readLine();
      System.out.println("RECU DU Client: " + phraseClient);
      phraseMajuscule = phraseClient.toUpperCase() + '\n';
      sortieVersClient.writeBytes(phraseMajuscule);
}//fin boucle
```

Coté Client

```
 BufferedReader entreeDepuisUtilisateur = new BufferedReader(new
InputStreamReader(System.in));

    Socket socketClient = new Socket("localhost", 7777);

 DataOutputStream sortieVersServeur = new
DataOutputStream(socketClient.getOutputStream());
    BufferedReader entreeDepuisServeur = new BufferedReader(new
InputStreamReader(socketClient.getInputStream()));
   String  phrase = entreeDepuisUtilisateur.readLine();

    sortieVersServeur.writeBytes(phrase + '\n');
    String phraseModifiee = entreeDepuisServeur.readLine();

    System.out.println("RECU DU SERVEUR: " + phraseModifiee);
    socketClient.close();
```

Exercice : calculette a dist.

- Le client a oublié sa calculette.
- Le client possède l'interface et le serveur qui fait le travail

Serveur TCP/IP (muti-clients)

Le serveur précédent accepte plusieurs connexions simultanées, mais ne traite qu'un client à la fois, les autres sont mis en attente?

Pour y remédier, utiliser les threads java (java.lang.Thread)

Serveur TCP/IP (muti-clients)

```
ServerSocket socketEcoute = new ServerSocket(Port);

while(true) { // boucle infinie

        //accepter une connexion
            Socket socketConnexion = socketEcoute.accept();

   //créer un thread : pour échanger les données avec le client
Connexion con = new  Connexion(socketConnexion);

Thread   processus_connexion = new  Thread(con);

processus_connexion .start();

   }
```

Serveur TCP/IP (muti-clients)

```
class Connexion implements Runnable{
...
Public Connexion (Socket cible){
                    this.cible=cible;
                In= new  BufferedReader(new
                    InputStreamReader(cible.getInputStream()));
                Out =  new PrintWriter( cible. getOutputStream());
}
public void  run() {
                    While (true) {
                                String ligne =  In.readLine();
                                If ( ligne == null) break;
                                                        //fin de connexion côté client
                            Out.println(ligne);
                            Out.flush();
                                        }
}
```

Serveur TCP/IP (muti-clients)

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