Les techniques réseau de base : les sockets

contenu de la section

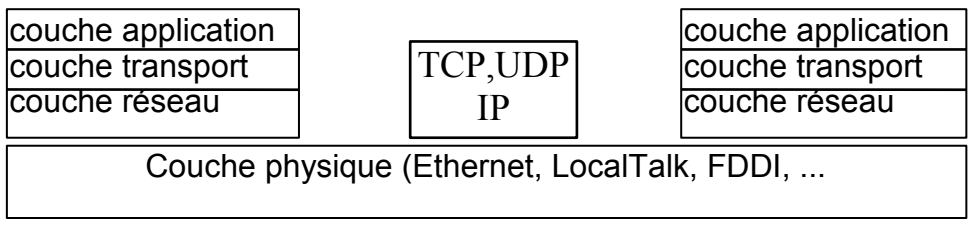
Quelques rappels

quelques concepts

les sockets sont des endpoints

- le point de communication (endpoint)
- l'adressage : la désignation des points de communication
  adresse IP = adresse réseau (machine) + adresse locale (numéro de port)
- le transport = sémantique de communication :
  - H1 : transfert de message avec maintien du séquencement
  - H2 : transfert de message sans duplication
  - H3 : transfert de message sans perte
  - H4 : transfert de message en préservant les frontières
  - H5 : transfert de messages urgents (hors bande)
  - H6 : service de connexion

- les protocoles réseaux IP (V4 ou V6) : format d'adresse réseau, format d'entête et de paquet, algorithme de routage (multi-routage)
- les protocoles de transport (UDP ou TCP) : format d'adresse (réseau + local), format d'entête et de paquet
  - UDP : de transport garantissant les propriétés H4
  - TCP : protocole de transport garantissant les propriétés H1, H2, H3, H5, H6



Internet, internets : réseaux basés sur IP

les adresses IP

- adresse machine sur 4 octets (IP v4)
- groupes A, B, C, D

A
B
C
D

```
0RESEAUX----------------------
10RESEAUX******----------------
110RESEAUX*************--------
1110GROUPES DE MULTICAST
11111réservé
```

- format IP : octet1.octet2.octet3.octet4
- format Internet : **entité.domaine.zone**

le pays (fr,jp,uk,de,...) ou le domaine d'activité :

com	:	commercial
edu	:	educational
gov	:	government
net	:	network

- quelques adresses réservées :
  - 10.xxx.yyy.zzz, 192.xxx.yyy.zzz non assignées
  - 127.xxx.yyy.zzz correspond à une boucle locale (en particulier 127.0.0.1)

l'adresse réseau (la classe InetAddress)

dans le package java.net

```
public static InetAddress getByName(String hostname) throws UnknownHostException ;
public static InetAddress[ ] getAllByName(String hostname)  throws UnknownHostException ;
public static InetAddress getLocalHost( ) throws UnknownHostException;

public byte[ ] getAddress( );
public String getHostName( ) throws UnknownHostException;
public byte[ ] getHostAddress( );
public boolean isMulticastAddress( );
.....
```

Internet, internets : réseaux basés sur IP

les numéros de port

proto applic	port	proto transport	description éventuelle
ftp-data	20	tcp	transfert de fichier par FTP
ftp-comm	21	tcp	transfert des commandes (get,put, ...) de FTP
telnet	23	tcp	
.....	.....	.....	
http	80	tcp	HyperText Transfer Protocol
RMI	1099	tcp	service d'enregistrement RMI

Les sockets : généralités

les objectifs

- offrir des interfaces de communication réseau de base

les modèles de base

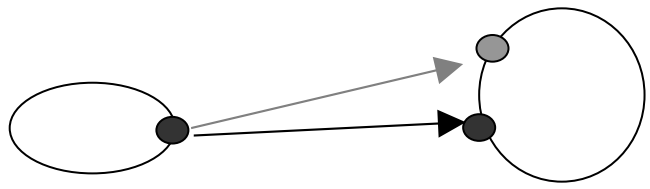
- toute communication met en jeu une socket locale
- toute socket a une **adresse**
- toute socket a un **type** (sémantique de communication)
  - SOCK_STREAM : H1 + H2 + H3 + H5 + H6
  - SOCK_DGRAM : H4

les sockets et java

- communication en mode connecté : Socket (socket TCP), ServerSocket (TCP connexion serveur)
- communication en mode déconnecté : DatagramPacket (message UDP), DatagramSocket (socket UDP)
- communication par diffusion : MulticastSocket (groupe de diffusion)

Le modèle clients/serveur

le modèle



- la socket de connexion (ServerSocket) vs la socket de travail (Socket)
- le protocole de connexion :
  - création d'une connexion entre 2 sockets de travail (client et serveur)
  - transparent au client (via le constructeur de la socket de travail)
  - explicite dans le serveur (via la méthode accept sur la socket de connexion)
  - accept( ) ➔ création d'une socket de travail

les serveurs séquentiels vs les serveurs concurrents

- **serveur séquentiel** : les sessions sont toutes traitées séquentiellement dans le même thread

```
création de sockCx;
while (true) {
  sockW=sockCx.accept( );
  <communication via sockW>
   sockW.close()
}
sockCx.close();
```

- **serveur concurrent** : les sessions sont traitées parallèlement dans des threads différents (typiquement un nouveau thread est créé à chaque connexion ....)
  - attention aux problèmes de concurrence : méthodes ou instructions **synchronized**
  - **amélioration des performances : via des pools de Threads (en particulier cachedThreadPool)**

```
création de sockCx;
while (true) {
    sockW=sockCx.accept( );
    new WorkThread(sockW).start();
}
sockCx.close();
```

```
class WorkThread  extends Thread {
    Socket sockW;
    WorkThread(Socket sW) { sockW=sW; }
    public void run( ) {
      <communication via sockW>
      sockW.close()
    }
}
```

Java et le modèle clients/serveur

la socket de connexion (la classe ServerSocket)

```
public ServerSocket();
public ServerSocket(int port) throws IOException,BindException;
public ServerSocket(int port,int queuelength) throws IOException,BindException;
public ServerSocket(int port,int queuelength,InetAddress bindAddress) throws IOException,BindException;

public Socket accept( ) throws IOException;
public InetAddress getInetAddress( );

+ méthodes de paramétrage
```

```
public void setSoTimeout(int millisecs) throws SocketException;
public int getSoTimeout( ) throws IOException;
```

la socket de travail/communication (la classe Socket)

```
public Socket(String host,int port) throws UnknownHostException, IOException;
public Socket(InetAddress inAd,int port) throws IOException;

public InputStream getInputStream( ) throws IOException;
public OutputStream getOutputStream( ) throws IOException;

public synchronized void close( ) throws IOException;

public InetAddress getInetAddress( ) throws UnknownHostException;
public int getLocalPort( );
public InetAddress getLocalAddress( );

+ méthodes de paramétrage
```

```
public void setSoLinger(boolean on,int seconds) throws SocketException;
public int getSoLinger( ) throws SocketException;
public void setTcpNoDelay(boolean on) throws SocketException;
public boolean getTcpNoDelay( ) throws SocketException;
public void setSoTimeout(int millisecs) throws SocketException;
public int getSoTimeout( ) throws IOException;
```

Des exemples de communication clients/serveur (1)

un serveur "echo" séquentiel

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

public class EchoServer    {
   public static void main(String[ ] args) {
      try{
         ServerSocket cnxSck=new ServerSocket(5555);

         while(true) {
            Socket sck= cnxSck.accept( );
            DataOutputStream oStream = new DataOutputStream(sck. getOutputStream( ));
            DataInputStream  iStream = new DataInputStream(sck.getInputStream( ));

            for (int i=0;i<50;i++) {
               double val=iStream.readDouble();
               oStream.writeDouble(val);
            }

            sck.close();
         }

         cnxSck.close();

      }  catch(IOException e) { System.err.println(e); }
   }
}
```

Des exemples de communication clients/serveur (2)

un client "echo"

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

public class EchoClient     {
   public static void main(String[ ] args) {
      try{
         Socket sck=new Socket(args[0],5555);
         DataOutputStream oStream = new DataOutputStream(new BufferedOutputStream(sck. getOutputStream( )));
         DataInputStream  iStream = new DataInputStream(new BufferedInputStream(sck.getInputStream( )));

         for(int i=0;i<50;i++) oStream. writeDouble(Math.random());
         oStream.flush();
         for(int i=0;i<50;i++) System.out.println("val echo=" +iStream.readDouble());

         sck.close();
      } catch(UnknownHostException e) { System.err.println(e);
      }  catch(IOException e) { System.err.println(e); }
   }
}
```

Des exemples de communication clients/serveur (3)

un serveur concurrent "echo"

- le code client est inchangé....

```
public class EchoServer     {

    public static void main(String[ ] args)  {
        ServerSocket cnx=new ServerSocket(5555);
        while(true) {
          Socket s = cnx.accept( );
          new Thread(new ServTask(s)).start();
        }
        cnxSck.close();
    }
}
```

```
import java.net.*;
import java.io.*;
class ServTask implements Runnable {
    private Socket sck;

    public ServTask(Socket sW) { sck=sW; }

    public void run( ) {
        try {
            DataOutputStream oStream = new DataOutputStream(new BufferedOutputStream(sck. getOutputStream( )));
            DataInputStream  iStream = new DataInputStream(new BufferedInputStream(sck.getInputStream( )));

            for (int i=0;i<50;i++) {
                double val=iStream.readDouble();
                oStream.writeDouble(val);
            }
            oStream.flush();
            sck.close();

        }  catch(IOException e) { System.err.println(e);
        }  catch(SocketException e) { ..... }
    }
}
```

```
public class EchoServer1   {

    public static void main(String[ ] args)  {
        ServerSocket cnx=new ServerSocket(5555);
        ExecutorService exec = Executors.newCachedThreadPool();
        while(true) {
          Socket s = cnx.accept( );
          exec.execute(new ServTask(s));
        }
        cnxSck.close();
    }
}
```

Complément : le debug distant avec Eclipse

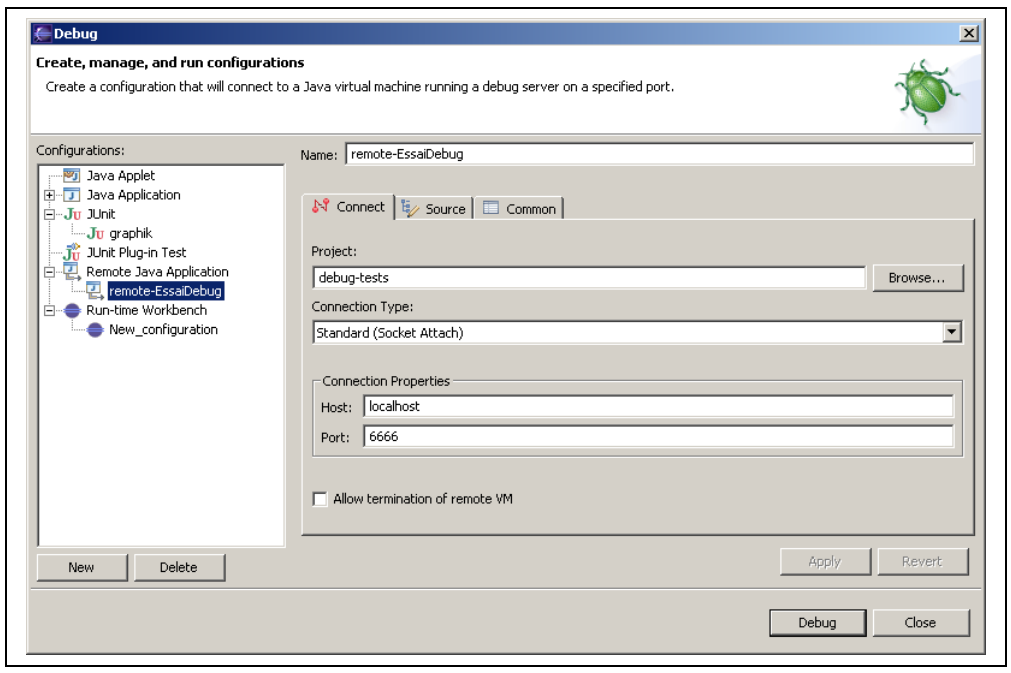
la marche à suivre

- **lancer l'application distante avec les commandes suivantes :**
  java –Xdebug –Xnoagent –Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=8000 -Djava.compiler=NONE …….

  –Xdebug : mode debug
  –Xnoagent
  –Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=6666 : lancement du protocole java debug wire protocol
  -Djava.compiler=NONE : pas de compilation JIT

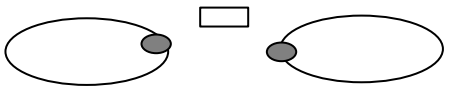
- **lancer l'application locale avec une configuration remote application**

- **debugger …**



Compléments sur les sockets : la communication par datagram

le modèle de communication par datagram



- socket de communication (DatagramSocket) = file d'attente
- pas de protocole de connexion
- adresse de la socket destination associée au datagram

le paquet Datagram

```
public DatagramPacket(byte buffer[ ],int length) throws IllegalArgumentException
public DatagramPacket(byte buffer[ ],int length,InetAddress ia,int port) throws IllegalArgumentException

public InetAddress getAddress( )
public int getPort( )
public byte[ ] getData( )
public int getLength( )
```

packet en réception
packet en émission

la socket Datagram

```
public DatagramSocket( ) throws SocketException
public DatagramSocket(int port ) throws SocketException
public DatagramSocket(int port,InetAddress inA) throws SocketException

public void send(DatagramPacket dp) throws IOException
public void receive(DatagramPacket dp) throws IOException
public void close( )
public int getLocalPort()
```

```
public void setSoTimeout(int millisecs) throws SocketException
public  int getSoTimeout( ) throws IOException
```

+ méthodes de paramétrage

Compléments sur les sockets : un exemple de communication par datagram (1)

l'entité initiatrice de l'échange

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

public class echoClient      {
    public static void main(String[ ] args) {
        String chaine="bonjour les amis";
        byte[ ] bufR=new byte[32];
        byte[ ] bufE=chaine.getBytes();

        try{
            InetAddress inAd=InetAddress.getByName("igloo.ibp.fr");
            DatagramSocket sck=new DatagramSocket( );

            DatagramPacket dSnd=new DatagramPacket(bufE,bufE.length,inAd,5555);
            sck.send( dSnd);

            DatagramPacket dRep=new DatagramPacket(bufR,bufR.length);
            sck.receive(dRep);

            System.out.println(new String(bufR));
            sck.close();

        }   catch(UnknownHostException e) { System.err.println(e);
        }   catch(IOException e) { System.err.println(e); }
        }   catch(IllegalArgumentException e) { System.err.println(e); }
    }
}
```

Compléments sur les sockets : un exemple de communication par datagram (2)

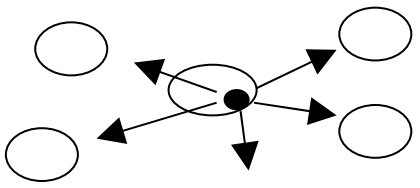
l'autre entité ....

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

public class echoServer{
   public static void main(String[ ] args) {
      byte[ ] buffer = new byte[255];
      try{
         DatagramSocket sck=new DatagramSocket(5555);

         DatagramPacket dRec=new DatagramPacket(buffer,buffer.length);
         sck.receive(dRec);

         DatagramPacket dSnd= new DatagramPacket(buffer, dRec.getLength(),dRec.getAddress(),dRec.getPort( ));
         sck.send(dSnd);

         sck.close();
      }  catch(IOException e) { System.err.println(e); }
      }  catch(IllegalArgumentException e) { System.err.println(e); }
   }
}
```

Compléments sur les sockets : la diffusion (1)

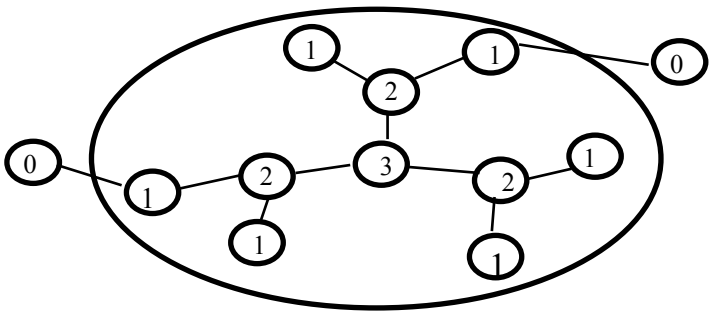
le modèle de communication multicast

- adresse de diffusion
- routeurs multicast
- adresses comprises entre 224.0.0.0 et 239.255.255.255



BASE-ADDRESS.MCAST.NET	224.0.0.0	non utilisé
ALL-SYSTEMS.MCAST.NET	224.0.0.1	tous les systèmes du sous réseau
NTP.MCASR.NET	224.0.1.1	Network Time Protocol
NSS.MCAST.NET	224.0.1.6	Service de nom
IETF-1-LOW-AUDIO.MCAST.NET	224.0.1.10	canal 1 audio pour meetings IETF
.....		
MBONE (Multicast Backbone on the Internet)	224.2.*.*	
+	239.0.0.0 → 239.255.255.255	

le TTL (Time to live)



QUELQUES VALEURS HEURISTIQUES	
machine locale	0
sous-réseau local	1
site local	16
réseau prim. national	32
national	40
réseaux prim. internat	128
partout	255

Compléments sur les sockets : la diffusion (2)

la socket Multicast (la classe MulticastSocket)

```
public MulticastSocket() throws SocketException
public MulticastSocket(int port) throws SocketException

public void joinGroup(InetAddress mcastAddr) throws SocketException
public void leaveGroup(InetAddress mcastAddr) throws SocketException

public synchronized void send(DataGramPacket dp,byte ttl) throws IOException,SocketException ;
public synchronized void receive(DataGramPacket dp) throws IOException,SocketException ;
…….
```

Compléments sur les sockets : la diffusion (2)

un exemple de réception de données multicast

```
import .....

public MCastExemple {
   public static main(String[ ] args) {
      InetAddress inAd ;
      MulticastSocket sck;

      try {
         inAd=InetAddress.getByName(args[0]);
      }  catch(UnknownHostException e) { System.err.println(e); }

      int port= ……..;
      try {
         sck=new MulticastSocket(port);
         sck.joinGroup(inAd);

         byte[ ] buffer=new byte[65000];
         DatagramPacket dp=new DatagramPacket(buffer,buffer.length);

         while(…….) {
            sck.receive(dp);
            ……….
         }
         sck.leaveGroup();
         sck.close();
      }  catch(SocketException e) { System.err.println(e); }
   }
}
```