

Cours n°4

Programmation réseau et sockets

1 Master Langue et Informatique – Internet et Bases de Données – Claude Montacié

Sommaire

1. Package Java.net

- Adresses internet
- Socket de communication

2. Communication en mode connecté

- Ouverture de la connexion
- Flots de lecture et d'écriture

3. Communication en mode non connecté

- Envoi d'un datagramme
- Réception d'un datagramme

2 Master Langue et Informatique – Internet et Bases de Données – Claude Montacié

INTRODUCTION

Bibliographie

Jan Graba, An Introduction to Network Programming with Java, Springer-Verlag London, 2006

E.-R. Harold, Java Network Programming, O'Reilly, 2004

G. Roussel, E. Doris, N. Bedon, R. Forax, Java et Internet, Vuibert, 2002

3 Master Langue et Informatique – Internet et Bases de Données – Claude Montacié

1. PACKAGE JAVA.NET

34 classes et 12 types d'exception

Ensemble de classes permettant une interaction avec le réseau pour recevoir et envoyer des données.

Développement de programmes établissant des communications avec d'autres applications distantes

Communication réseau

Gestion des protocoles des couches réseaux et transport

Encapsulation de la notion de Socket

Accélération par l'utilisation de caches

Utilisation simplifiée par rapport aux bibliothèques systèmes (Unix et Windows)

Programmation internet

Gestion des protocoles de la couche application (ssh, sftp, http, ...)

Sécurisation par l'utilisation de mots de passe

4 Master Langue et Informatique – Internet et Bases de Données – Claude Montacié

Classe InetAddress

Représentation d'une adresse internet avec deux attributs

hostName représentant le nom sous forme d'une chaîne de caractères

Address représentant le numéro IP sous forme d'un tableau d'octets

Méthodes statiques pour obtenir une InetAddress

Pas de constructeur,

`InetAddress` `InetAddress.getLocalHost()`
obtention de l'`InetAddress` correspondant à la machine locale

`InetAddress` `InetAddress.getByName(String host)`
obtention de l'`InetAddress` d'une machine d'un nom donné

`InetAddress` `InetAddress.getByAddress(byte[] addr)`
obtention de l'`InetAddress` d'une machine d'un numéro IP donné

`InetAddress` `InetAddress.getByAddress(String host, byte[] addr)`
obtention de l'`InetAddress` d'une machine d'un nom et d'un numéro IP donné

Levée de l'exception `UnknownHostException` en cas de machine inconnue

Méthodes (1/2)

Obtenir le nom d'une machine

`String` `getHostName()` `String` `getHostAddress()` Nom de la machine

Obtenir le numéro IP d'une machine

`byte[]` `getAddress()` Adresse IP sous forme de 4 octets

Caractéristiques

`boolean` `isAnyLocalAddress()` Adresse non spécifiée (0.0.0.0)
`boolean` `isLoopbackAddress()` Adresse locale de test (127.0.0.0) localhost
`boolean` `isLinkLocalAddress()` Adresse locale (169.254.0.0)
`boolean` `isSiteLocalAddress()` Adresse privée (127.0.0.0)
`boolean` `isMulticastAddress()` Adresse multicast (224.0.0.0)
`boolean` `isReachableAddress(int timeout)` Test de connectivité

Méthodes (2/2)

getInetAddress.java

```
try{
    InetAddress monAdresse = InetAddress.getByName("smtp.laposte.net");
    System.out.println(monAdresse.getHostName());
    System.out.println(monAdresse.getHostAddress());
    System.out.print(monAdresse.isAnyLocalAddress() + " ");
    System.out.print(monAdresse.isLoopbackAddress() + " ");
    System.out.print(monAdresse.isLinkLocalAddress() + " ");
    System.out.println(monAdresse.isReachable(100));

} catch (UnknownHostException exp){
    System.out.println("machine inconnue");
} catch (IOException e) {
    System.out.println("machine non atteignable");
}
```

```
smtp.laposte.net
81.255.54.9
false false false false
```

Principes

Lien de communication inter-processus

Développé initialement sous UNIX (1983)

Extension de la notion de tube nommé (pipe) pour des machines distantes

Utilisation des mécanismes classiques d'E/S (java.io)

Restriction au domaine AF_INET pour les sockets java (couche réseau)

Une adresse (nom, numéro IP) et un numéro de port

Diagramme de communication

Un réseau et deux sockets

Type de services IP (Internet Protocol)

Socket STREAM

Protocole TCP (Transmission Control Protocol)
 Mode connecté avec contrôle de flux
 Connexion établie entre les machines distantes
 Fiabilité de la transmission des données
 Comparaison avec une communication téléphonique

Socket DATAGRAM

Protocole UDP (User Datagram Protocol)
 Mode non connecté avec transmission par paquet sans contrôle de flux
 Pas de connexion entre le client et le serveur
 Pas de sécurité de transmission des données
 Comparaison à une communication par lettre

Port d'entrée-sortie

Canal de communication accessibles à travers un réseau

Identification par un entier (16 bits),

Correspondance avec un service spécifique

Services standard internet
 terminal sécurisé (SSH port 22),
 courrier sortant (SMTP port 25),
 serveur web (HTTP port 80),
 courrier entrant (POP3 port 110),
 transfert de fichiers sécurisé (SFTP port 115),
 courrier entrant (IMAP3 port 220),
 authentification (LDAP, port 389),
 serveur web sécurisé (HTTPS port 443),
 accès à la base de données Mysql (port 3306)

Ports utilisateurs (entre 30 000 et 60 000)
 consultation de la table des services
 enregistrement à l'IANA

Classe InetAddress

Représentation d'une adresse de socket avec deux attributs

Adresse IP sous la forme d'une InetAddress
 Port d'entrée/sortie sous la forme d'un entier

Constructeurs

`InetAddress(String hostName, int port)`
`InetAddress(InetAddress addr, int port)`
`InetAddress(int port)` adresse IP non spécifiée (utilisée en écoute)

Accesseurs

`InetAddress getAddress()` accès à l'InetAddress
`String getHostName()` accès au nom
`int getPort()` accès au numéro de port

Principes

Protocole fiable de la couche réseau (RFC 793)

Flot découpé en paquets (~536 octets)
 Mécanismes de contrôle de flot (contrôle de congestion)
 Mécanisme d'acquittement des paquets
 alarme activée à l'émission d'un paquet,
 désactivation en cas de réception de l'acquittement,
 réémission après un temps donné
 Ajout à chaque paquet d'un numéro de séquence
 préservation de l'ordre,
 élimination des doublons

Demande de connexion (1/2)

Choix de l'adresse des sockets

Adresse de la socket locale : saddr1

addr1 : numéro IP de la machine locale

port1 : numéro de port choisie de la machine locale

Adresse de la socket distante : saddr2

addr2 : numéro IP de la machine distante

port2 : numéro de port choisie de la machine distante

Trois étapes

1. Construction d'une socket de communication `Socket s = new Socket()`
2. Attachement du port de connexion `s.bind (saddr1)`
3. Demande de connexion à une socket distante `s.connect (saddr2)`

Constructeur alternatif

`Socket s = new Socket(addr2, port2, addr1, port 1)`

Levée de l'exception `IOException` en cas de refus de connexion

Demande de connexion (2/2)

```
// création de la socket
Socket s = new Socket(); InetAddress addr1 = null;
int port1 = 32005, port2 = 32006;
try { addr1 = InetAddress.getLocalHost(); }
catch (UnknownHostException exp){ }
try {
    // attachement
    InetSocketAddress saddr1 = new InetSocketAddress(addr1, port1);
    s.bind(saddr1);

    // adresse de la machine distante
    InetAddress addr2 = InetAddress.getByName("192.168.1.102");
    InetSocketAddress saddr2 = new InetSocketAddress(addr2, port2);

    // demande de connexion
    s.connect(saddr2);
    System.out.println("Connexion établie entre " +
        s.getLocalSocketAddress() + " et " +
        s.getRemoteSocketAddress());
}
```

Connexion établie entre /192.168.1.101:32005 et /192.168.1.102:32006

Acceptation de la connexion (1/2)

Choix des caractéristiques du service

Adresse de la socket de service : saddr

addr : numéro IP de la machine locale

port : numéro de port choisie pour le service

max : nombre maximum de connexions

Trois étapes

1. Construction d'un écouteur de connexion `ServerSocket ss = new ServerSocket()`
2. Attachement du port de service `ss.bind (addr, max)`
3. Acceptation de la connexion et récupération de la socket `Socket s = ss.accept ()`

Acceptation de la connexion (2/2)

```
int port = 32006;
InetSocketAddress saddr = null;
try {
    InetAddress addr = InetAddress.getLocalHost();
    saddr = new InetSocketAddress(addr, port);
} catch (UnknownHostException exp){ }
try {
    // création d'un écouteur de connexion
    ServerSocket ss = new ServerSocket();
    // attachement
    ss.bind(saddr);
    // acceptation de la connexion
    Socket s = ss.accept();
    System.out.println("Connexion établie entre " +
        s.getLocalSocketAddress() + " et " +
        s.getRemoteSocketAddress());
} catch (IOException exp){ }
```

Connexion établie entre /192.168.1.102:32006 et /192.168.1.101:32005

Gestion des flots

Canal de communication à double sens (full duplex)

Fourniture d'un flot de lecture des données arrivant sur la connexion

`InputStream` `getInputStream()`

Fourniture d'un flot d'écriture des données sortant de la connexion

`OutputStream` `getOutputStream()`

Fermeture du flot de lecture `shutdownInput()` `boolean isOutputShutdown()`

Fermeture du flot d'écriture `shutdownOutput()` `boolean isInputShutdown()`

Fermeture du canal `Close()` `boolean isClosed()`

Lecture/Ecriture

Méthodes read/write

Composition de classes `new InputStreamReader(s.getInputStream());`

Flots (1/2)

```
// Création des flots
BufferedReader br = null; PrintStream ps = null;
try {
    br = new BufferedReader(new InputStreamReader(s.getInputStream()));
    ps = new PrintStream(s.getOutputStream());
} catch (IOException exp){
    System.out.println("erreur de création des flots");
}

// Communication
try {
    ps.println("début de la communication");
    String ligne = br.readLine();
    System.out.println(ligne);
}
catch (IOException exp){
    System.out.println("erreur d'entrée-sortie");
}
```

fin de la communication

Flots (2/2)

AcceptionConnexion.java

```
// Création des flots
BufferedReader br = null; PrintStream ps = null;
try {
    br = new BufferedReader(new InputStreamReader(s.getInputStream()));
    ps = new PrintStream(s.getOutputStream());
} catch (IOException exp){
    System.out.println("erreur de création des flots");
}

// Communication
try {
    String ligne = br.readLine();
    System.out.println(ligne);
    ps.println("fin de la communication");
    br.readLine();
} catch (IOException exp){
    System.out.println("erreur d'entrée-sortie");
}
```

début de la communication

Introduction

Protocole non fiable de la couche réseau (RFC 768)

Pas de flot,

Acheminement de bloc de données (datagramme)

Taille max des données transportées: ~64Ko

Checksum optionnel en IP v4, obligatoire en IP v6

Classe DatagramSocket

Socket permettant un service UDP

Création d'une socket sur chaque machine

Classe DatagramPacket

Empaquetage des données envoyées ou reçues

Adresse de la socket de destination

Principes

Adresse de la socket distante : saddr

addr : numéro IP de la machine distante

port : numéro de port choisie de la machine distante

Quatre étapes

1. Construction d'une socket de communication
`DatagramSocket ds = new DatagramSocket()`
2. Empaquetage des données dans un tableau de bytes : b
3. Construction du datagramme à transmettre
`DatagramPacket dp = new DatagramPacket(b, b.length, saddr)`
4. Envoi du datagramme `ds.send(dp)`
Levée des exceptions `SocketException` et `IOException`

`void setBroadcast(boolean on)`

autorisation d'envoi à toutes les machines du sous-réseau

`void setSendBufferSize(int size)`

modification de la taille du tampon

Exemple

```
// création de la socket
DatagramSocket ds = null; int port = 32006;
try { ds = new DatagramSocket();
}
catch (SocketException exp){ }

// adresse de la machine distante
InetSocketAddress saddr = null;
try { InetAddress addr = InetAddress.getByName("192.168.1.102");
    saddr = new InetSocketAddress(addr, port);
} catch (UnknownHostException exp){ }

// construction et envoi du datagramme
try { String s = "début de la communication";
    byte[] b = s.getBytes();
    DatagramPacket dp = new DatagramPacket(b, b.length, saddr );
    ds.send(dp);
} catch (SocketException exp){ } catch (IOException exp){ }
```

Principes

Quatre étapes

1. Construction d'une socket de communication écoutant le port p
`DatagramSocket ds = new DatagramSocket(p)`
2. Construction d'un datagramme contenant un tableau de bytes vide : b
`DatagramPacket dp = new DatagramPacket(b, b.length)`
3. Réception du datagramme `ds.receive(dp)`
4. Extraction des données du paquet
Levée des exceptions `SocketException` et `IOException`

`byte[] getData()`

tableau de byte contenant les données

`int getLength()`

taille en octet des données reçues

`SocketAddress getSocketAddress()`

adresse de la socket émettrice

Exemple

```
DatagramSocket ds = null; int port = 32006;
try { ds = new DatagramSocket(port);
} catch (SocketException exp){ }

// réception du datagramme
try {
    byte b[] = new byte [1024];
    DatagramPacket dp = new DatagramPacket(b, b.length);

    ds.receive(dp);
    InetSocketAddress saddr = (InetSocketAddress)dp.getSocketAddress();
    System.out.println("datagramme en provenance de " + saddr.toString());

    System.out.println(new String(b, 0, dp.getLength()));
} catch (SocketException exp){ } catch (IOException exp){ }
```

datagramme en provenance de /192.168.1.101:1131

début de la communication