Java.net
Sockets

IFT 3880-IFT 6835 APPLICATIONS DISTRIBUÉES

Par Laurent Magnin

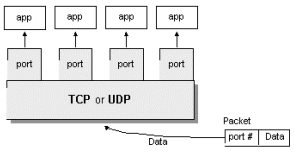
Université de Montréal

Concepts de base (1)

- **TCP (Transmission Control Protocol) est un protocole à base de *connection*, fournissant une flot *fiable* de données entre deux ordinateurs.**
- **UDP (User Datagram Protocol) est un protocole qui envoie des *paquets indépendants* de données entre deux ordinateurs *sans assurer* que ces paquets sont bien reçus ou non.**

Concepts de base (2)

- **Ces deux protocoles utilisent le port pour diriger les données arrivées vers un processus particulier s'exécutant sur l'ordinateur**



Package java.net

- **Tous les classes Java concernant le réseau sont groupées ensembles dans le package java.net.**
- **Les classes URL, URLConnection, Socket et ServerSocket utilisent TCP pour la communication sur le réseau.**
- **Les classes DatagramPacket, DatagramSocket et MulticastSocket utilisent UDP**

URL (Uniform Resource Locator)

- **Contient une chaîne de caractères décrivant un lien d'accès à une ressource sur Internet.**
- **2 composants principaux:**
  - protocole nécessaire pour accéder à la ressource
  - position de la ressource

Création et analyse d 'une URL

```
try {
    monURL = new URL("http://www.iro.umontreal.ca:80/")
}
catch (MalformedURLException e) {
    . . .
    // exception handler code here
    . . .
}
```

- **getProtocol()**
- **getHost()**
- **getPort()**

URLConnection (1)

- **Initialisation d'un lien de communication entre le programme Java et l'URL sur le réseau**

```
try {
   URL yahoo = new URL("http://www.yahoo.com/");
   URLConnection yahooConnection = yahoo.openConnection();
}
catch (MalformedURLException e) {  // new URL() failed
   . . .
}
catch (IOException e) {    // openConnection() failed
   . . .
}
```


URLConnection (2)

- **Recevoir des données de la connection :**

```
BufferedReader in = new BufferedReader(
    new InputStreamReader(connection.getInputStream()));
```

- **Envoyer des données à travers la connection :**

```
PrintWriter out =
    new PrintWriter(connection.getOutputStream());
```


Socket : définition

- Est le point terminal d'un lien de communication bi-directionnel entre deux programmes sur le réseau.
- Est représenté et identifié par un numéro de port
- Java offre deux classes: **Socket** et **ServerSocket** pour implanter respectivement la part client et la part serveur de la connection.

Mode d'emploi d'une *Socket*

- **1. Ouverture du *Socket.***
- **2. Ouverture des flots d'entrée (input stream) et de sortie (output stream) de la *Socket.***
- **3. Lecture et écriture à partir de ces deux flots (dépendant de l 'application).**
- **4. Fermeture des flots.**
- **5. Fermeture de la *Socket.***

ServerSocket côté serveur

- **Le serveur se met à l'écoute sur un port :**

```
ServerSocket  server = new ServerSocket(4250);
```

- **et doit accepter la connexion d'un client :**

```
Socket sClient = server.accept();
```

- **Il envoie alors sur le OutputStrem de la socket**

```
OutputStream out = sClient.getOutputStream();
out.write('blabla');
```

- **et reçois sur le InputStream de la socket**

```
InputStream in = sClient.getInputStream();
String str = in.read();
```


Manipulation de ServerSocket

- Principle :

```
while (true) {
        accepter une connection ;
        créer un thread pour traiter ce client;
}
```

Socket (TCP) côté client

- **On fournit l'adresse URL ou le nom du domaine et le numéro de port pour créer une socket:**

```
Socket s = new Socket('arabica.udm.ca',4250);
```

- **On envoie sur le OutputStream de la socket**

```
OutputStream out = s.getOutputStream();
out.write('balabala');
```

- **On reçois sur le InputStream de la socket**

```
InputStream in = s.getInputStream();
String str = in.read();
```