Programmation réseau et concurrente

Benoît Barbot

Département Informatique, Université Paris-Est Créteil, M1

Mardi 5 Janvier 2015, Cours 1 : Introduction et rappel

Information Pratique

- Cours : 7 séances de 3H.
- TD : 4 séances de 3H
- TP : 9 séances de 3H
- Enseignants : Benoît Barbot (CM+TD+TP) et Daniele Varacca (TD+TP).
- Évaluation : projet + examen.

Motivation

Dans les cours précédant

- Couche réseau “basse”
- Protocole textuel (http,smtp, . . .)
- Scénario de communication simple (1 serveur, quelques clients)

Motivation

Dans les cours précédant

- Couche réseau “basse”
- Protocole textuel (http,smtp, . . .)
- Scénario de communication simple (1 serveur, quelques clients)

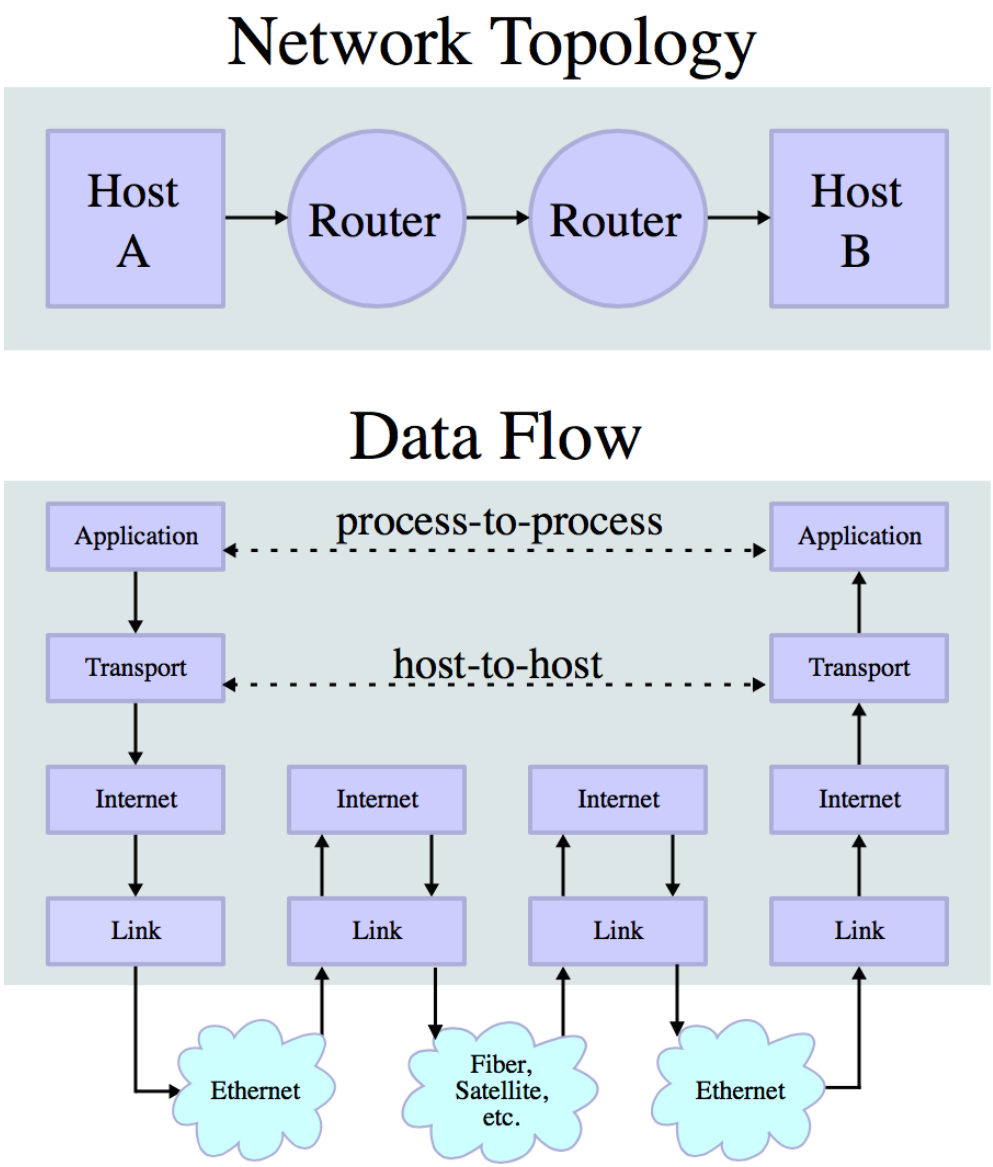
Dans ce cours

- Centré sur la couche application
- Protocole de haut niveau
- Échange de structure de donnée, de programme, d'objets
- Programmation réseau robuste avec forte charge
- Problématique et gestion de la concurrence
- Création de *middleware*

Middleware

- Entre le programme et le système
- Abstraction de l'architecture (ex : windows, linux)
- Abstraction d'une architecture distribuée
- Agrégation de base donnée

Rappel Couche Réseau 1



Système de couche

- L'information traverse les couches
- Chaque couche fournit des garanties
- Une couche peut aider une couche supérieure

Rappel Couche Réseau 2

Point de vue du programmeur

- Le système gère toutes les couches sauf application
- Programmation réseau $\approx$ appels systèmes
- Appels systèmes simplifié par bibliothèque langage (ex libc, java.nio ...)

Rappel Couche Réseau 2

Point de vue du programmeur

- Le système gère toutes les couches sauf application
- Programmation réseau $\approx$ appels systèmes
- Appels systèmes simplifié par bibliothèque langage (ex libc, java.nio ...)

Choix de conceptions

- Choix de protocole de transport (TCP,UDP, ...)
- Choix de protocole réseau (domaine) (IPv4,IPv6,UNIX,...)
- En fonctions des performances et garanti

Protocole TCP - UDP

Transmission Control Protocol

- Connecté (flux de donnée)
- Transmission fiable des données
- Gestion de congestion
- Optimisation de flux (algorithme de Nagle)
- Surcoûp due aux accusés

Protocole TCP - UDP

Transmission Control Protocol

- Connecté (flux de donnée)
- Transmission fiable des données
- Gestion de congestion
- Optimisation de flux (algorithme de Nagle)
- Surcoûp due aux accusés

User Datagram Protocol

- Non connecté (message)
- Taille de message limité
- Garanti d'intégrité
- Pas de garantie de réception
- Pas de garantie d'ordre
- Duplication possible des paquets
- Léger et rapide

Protocole TCP - UDP

Transmission Control Protocol

- Connecté (flux de donnée)
- Transmission fiable des données
- Gestion de congestion
- Optimisation de flux (algorithme de Nagle)
- Surcoûp due aux accusés

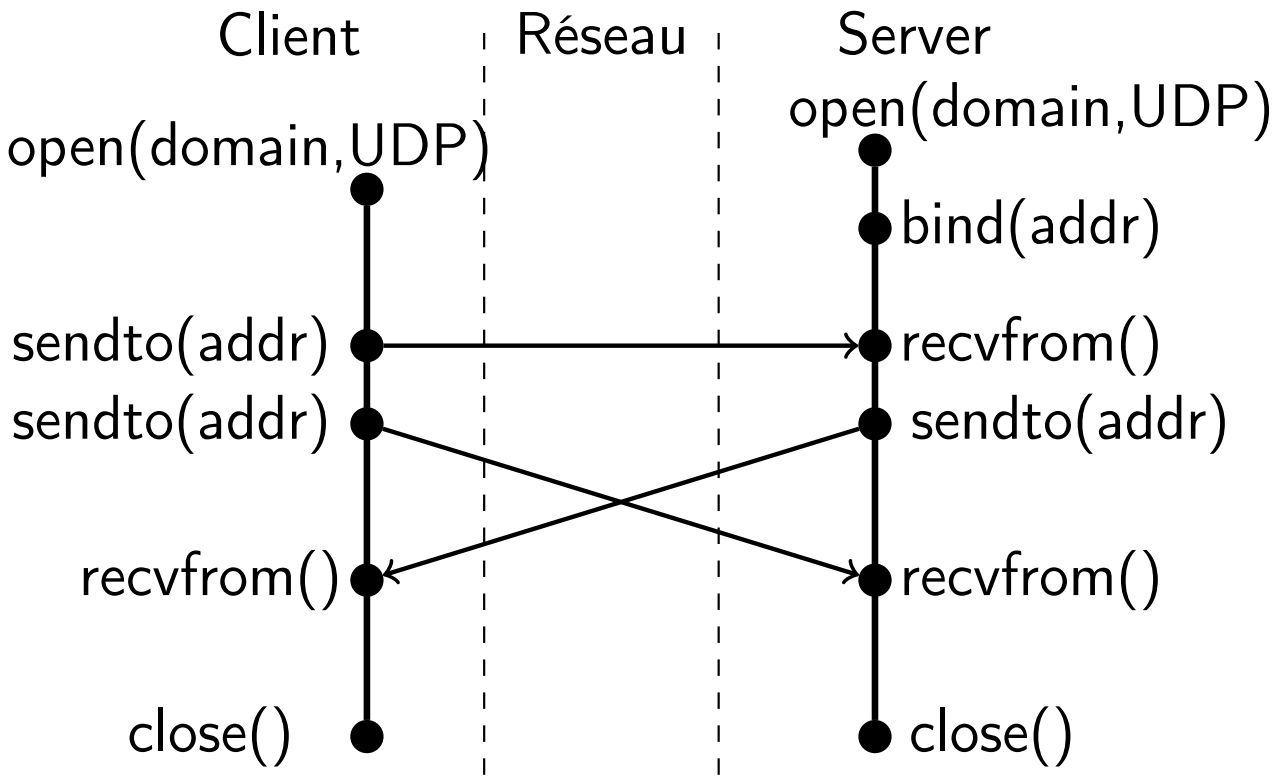
User Datagram Protocol

- Non connecté (message)
- Taille de message limité
- Garanti d'intégrité
- Pas de garantie de réception
- Pas de garantie d'ordre
- Duplication possible des paquets
- Léger et rapide

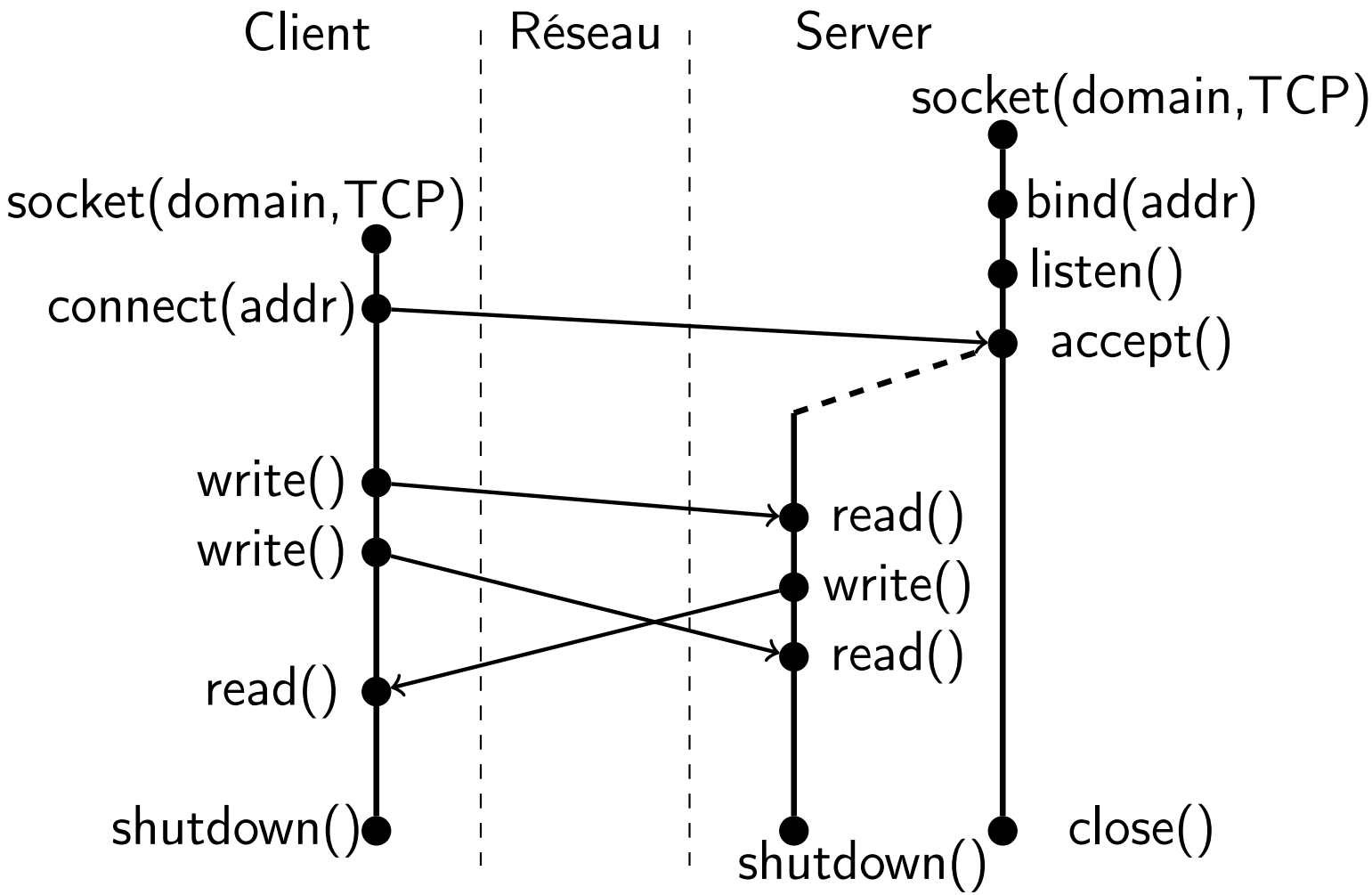
Raw Socket

- Pas de garantie
- Pas de couche de transport

Connexion UDP



Connexion TCP



Résolution d'adresse web

Problèmes :

- Adresse sous forme textuelle (ex : http ://www.u-pec.fr)
- Les appels système demandent une adresse IP (ex : 193.48.143.10)
- Le protocole et le domaine souvent implicite (ex : IPv4, IPv6, UNIX)

Résolution d'adresse web

Problèmes :

- Adresse sous forme textuelle (ex : http ://www.u-pec.fr)
- Les appels système demandent une adresse IP (ex : 193.48.143.10)
- Le protocole et le domaine souvent implicite (ex : IPv4, IPv6, UNIX)

Solutions :

- Besoin d'interroger un DNS
- libc utilise *getaddrinfo()*
- Java utilise la classe *InetSocketAddress*

I/O bloquantes/non bloquantes

I/O bloquantes

- *accept()*, *read()*, *write()* peuvent bloquer
- Interruption du *thread* appelant
- Déblocage quand l'action peut avoir lieu (des données à lire, écrire,. . .)
- Plusieurs *threads*, problème de concurrence

I/O bloquantes/non bloquantes

I/O bloquantes

- *accept()*, *read()*, *write()* peuvent bloquer
- Interruption du *thread* appelant
- Déblocage quand l'action peut avoir lieu (des données à lire, écrire,. . .)
- Plusieurs *threads*, problème de concurrence

I/O non bloquantes

- *accept()*, *read()*, *write()* retournent immédiatement
- Utilisation de *select()* (ou *poll()*, *epoll()*)
- Un seul *thread* peut gérer toutes les I/O
- Pas d'exécution parallèle, car 1 seul *thread*

I/O bloquantes/non bloquantes

I/O bloquantes

- *accept()*, *read()*, *write()* peuvent bloquer
- Interruption du *thread* appelant
- Déblocage quand l'action peut avoir lieu (des données à lire, écrire,...)
- Plusieurs *threads*, problème de concurrence

I/O non bloquantes

- *accept()*, *read()*, *write()* retournent immédiatement
- Utilisation de *select()* (ou *poll()*, *epoll()*)
- Un seul *thread* peut gérer toutes les I/O
- Pas d'exécution parallèle, car 1 seul *thread*

Solution intermédiaire

- *thread* léger
- répartition des connexions sur plusieurs threads

Boucle *select()*

en pseudo-code :

```
1  openConnections();
2  Set s = {all socket waiting for I/O}
3  while(true){
4     Set s2 = select(s,timeout);
5     for( c : s2 ){
6        readWriteAccept(c);
7     }
8  }
```

Exemple serveur de chat Java 1

```
1  public class Server {
2    private ServerSocketChannel incomeSocket;
3    private Selector selector;
4
5    public Server( int port) throws IOException {
6      incomeSocket = ServerSocketChannel.open();
7      incomeSocket.configureBlocking(false);
8      InetSocketAddress addr = new InetSocketAddress(port);
9      incomeSocket.socket().bind(addr);
10
11     selector = Selector.open();
12     incomeSocket.register(selector, SelectionKey.OP_ACCEPT);
13   }
14         ...
15 }
```

Exemple serveur de chat Java 2

```
1  public class Server {
2          ...
3  public void run() throws IOException {
4      while (true) {
5        selector.select();
6        Set<SelectionKey> selectedKeys = selector.selectedKeys()
7        for (SelectionKey k : selectedKeys){
8            if (k.isAcceptable()) {
9              accept();
10           } else {
11             repeat(k);
12           }
13       }
14       selectedKeys.clear();
15     }
16   }
17         ...
18 }
```

Exemple serveur de chat Java 3

```
1  public class Server {
2          ...
3    void accept() {
4      try {
5        SocketChannel ns = incomeSocket.accept();
6        System.out.println("New client" + ns);
7        ns.configureBlocking(false);
8        ns.register(selector, SelectionKey.OP_READ);
9      } catch (IOException e) {
10       System.out.println("Fail to accept new client: " + e);
11     }
12   }
13         ...
14 }
```

Java nio buffer

Buffer pour les I/O

- Implémente des I/O efficaces proches du système
- Adapté aux échanges de données binaire
- Sous classé pour chacun des types primitifs
- Gestion propre des encodages

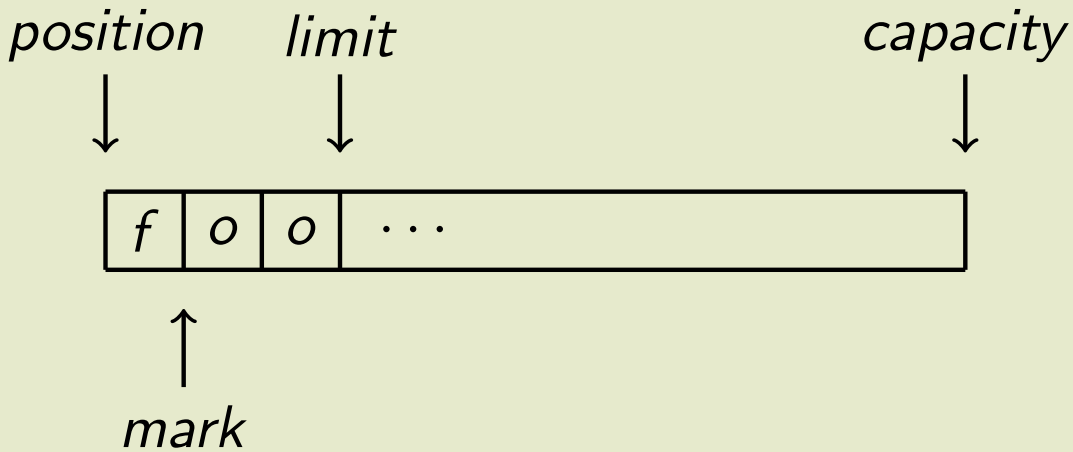
Java nio buffer

Buffer pour les I/O

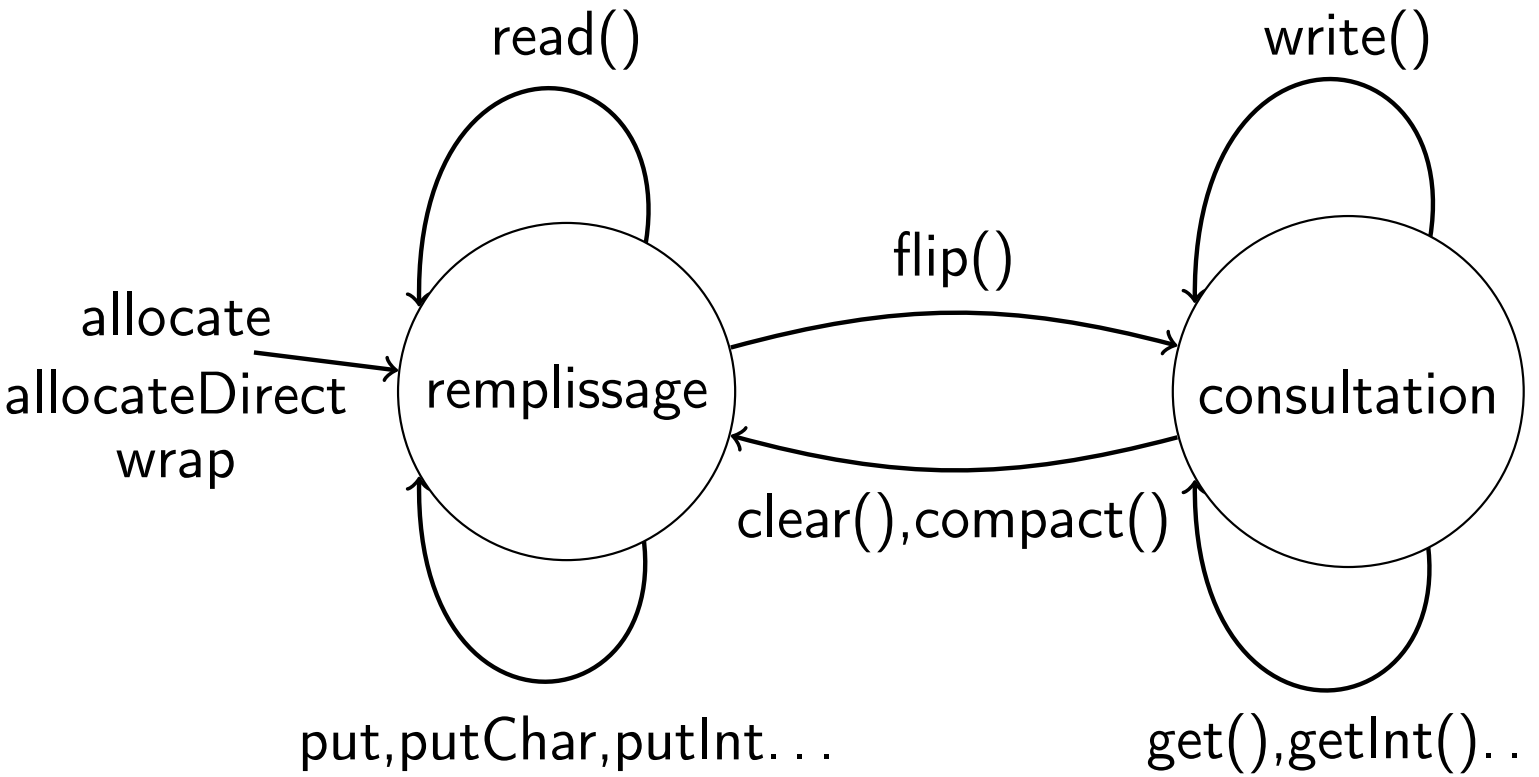
- Implémente des I/O efficaces proches du système
- Adapté aux échanges de données binaire
- Sous classé pour chacun des types primitifs
- Gestion propre des encodages

Description

- Un tableau
- 4 pointeurs
- Deux modes : remplissage et consultation



Utilisation



Encodage

Charset

- Spécifie l'encodage
- ex de charset US-ASCII, ISO646-US, ISO-8859-1,UTF-8,UTF-16

Encodage

Charset

- Spécifie l'encodage
- ex de charset US-ASCII, ISO646-US, ISO-8859-1,UTF-8,UTF-16

String -> ByteBuffer

```
1 Charset c = Charset.forName("UTF-8");
2 ByteBuffer bb = c.encode("test UTF-8");
```

Encodage

Charset

- Spécifie l'encodage
- ex de charset US-ASCII, ISO646-US, ISO-8859-1,UTF-8,UTF-16

String -> ByteBuffer

```
1  Charset c = Charset.forName("UTF-8");
2  ByteBuffer bb = c.encode("test UTF-8");
```

ByteBuffer -> String

```
1  Charset c = Charset.forName("UTF-8");
2  CharBuffer cb = c.decode(bb);
3  String s = cb.toString();
```

Exemple serveur de chat Java 4

```
1  public class Server {
2    private ByteBuffer buff = ByteBuffer.allocate(1024);
3          ...
4    void repeat(SelectionKey k) {
5      SocketChannel s = (SocketChannel) k.channel();
6      try {
7        buff.clear();
8        int n = s.read(buff);
9        if (n == -1)
10         throw new IOException("Client Close Connection");
11               buff.flip();
12
13       for (SelectionKey c : selector.keys()) {
14         if ( ((c.interestOps() & SelectionKey.OP_READ) != 0)
15                                  && c != k) {
16           SocketChannel sc = (SocketChannel) c.channel();
17           buff.rewind();
18
19           while (buff.hasRemaining()) {
20             sc.write(buff);
21           }
22         }
23       }
24     } ...
```

Exemple serveur de chat Java 4 bis

```
1       } catch (IOException e) {
2         System.out.println("Client left");
3         try {
4           s.close();
5         } catch (IOException e2) {
6         };
7         k.cancel();
8       }
9     }
```

Exemple serveur de chat Java 5 + Demo

```
1       public static void main(String[] args) {
2         System.out.println("Build␣server");
3         Server s = new Server(9090);
4         System.out.println("Start␣server");
5         s.run();
6       } catch (IOException e) {
7         e.printStackTrace();
8       }
```