# Programmation réseau et concurrente

Benoît Barbot

Département Informatique, Université Pariaient Créteil, M1

Mardi 12 janvier 2015, Cours 2 : Sérialisation et Threads

# Plan

# Motivation - Donnée en mémoire

## Format binaire

“foo”,”bar”,”upec” $\Rightarrow$

```
0000000: aced 0005 7400 0366  ....t..f
0000008: 6f6f 7400 0362 6172  oot..bar
0000010: 7400 0475 7065 63    t..upec
```

- Format dépend de la machine (Endianness)
- Contient des structures complexes (Graphe de pointeur)
- Format dépend du programme (langage)
- Non lisible pour un humain

# Motivation

## Échange de données

- Format indépendant de la machine (ex TCP -> big-endian)
- Structure linéaire (flux d'octet)
- Format standardisé

# Motivation

## Échange de données

- Format indépendant de la machine (ex TCP -> big-endian)
- Structure linéaire (flux d'octet)
- Format standardisé

## Format ad hoc

Besoin d'écrire du code pour chaque structure à stocker ;
$\Rightarrow$ "Boilerplate code"

## Sérialisation

Automatiser au maximum la conversion mémoire <-> fichier

# Caractéristique format de sérialisation

- Richesse des objets serialisable
- Lisible par un humain (HR)
- Représentation compacte
- Traduction depuis/vers format mémoire rapide

# Utilisation

## Échange entre processus

- Format binaire
- Rapidité de la conversion
- *Lecture/écriture sur la même machine*

# Utilisation

## Échange entre processus

- Format binaire
- Rapidité de la conversion
- *Lecture/écriture sur la même machine*

## Format de fichier

- Disque lent (pas besoin de conversion rapide)
- Selon besoin : compact/HR
- Pérenne

# Utilisation

## Échange entre processus

- Format binaire
- Rapidité de la conversion
- *Lecture/écriture sur la même machine*

## Format de fichier

- Disque lent (pas besoin de conversion rapide)
- Selon besoin : compact/HR
- Pérenne

## Réseau

- Variable ...

# 2 paradigmes

## La structure est connue du parser

- Plus fiable
- Plus facile à étendre

# 2 paradigmes

## La structure est connue du parser

- Plus fiable
- Plus facile à étendre

## Exemple Java

42 $\Rightarrow$ `0000 002a`

3.14159265359 $\Rightarrow$ `4009 21fb 5444 2eea`

“UPEC” $\Rightarrow$ `5550 4543`

# 2 paradigmes

## La structure est connue du parser

- Plus fiable
- Plus facile à étendre

## Exemple Java

42 $\Rightarrow$ `0000 002a`

3.14159265359 $\Rightarrow$ `4009 21fb 5444 2eea`

"UPEC" $\Rightarrow$ `5550 4543`

## La structure est encodée de manière non ambiguë

- Plus lourd
- Plus lent à encoder/decoder

# 2 paradigmes

## La structure est connue du parser

- Plus fiable
- Plus facile à étendre

## Exemple Java

42 $\Rightarrow$ `0000 002a`

3.14159265359 $\Rightarrow$ `4009 21fb 5444 2eea`

“UPEC” $\Rightarrow$ `5550 4543`

## La structure est encodée de manière non ambiguë

- Plus lourd
- Plus lent à encoder/decoder

## Exemple JSON

```
"UPEC", 42, true, 3.14159265359
```

# Exemple de format de sérialisation

- Comma Separated Value (CSV)
  HR, simple, bon compromis HR/compacité, pas de binaire, structure pauvre

# Exemple de format de sérialisation

- Comma Separated Value (CSV)
  HR, simple, bon compromis HR/compacité, pas de binaire, structure pauvre
- Extensible Markup Language (XML)
  HR, très utilisé, structure d'arbre, *pas de binaire, verbeux*

# Exemple de format de sérialisation

- Comma Separated Value (CSV)
  HR, simple, bon compromis HR/compacité, pas de binaire, structure pauvre
- Extensible Markup Language (XML)
  HR, très utilisé, structure d'arbre, *pas de binaire, verbeux*
- JavaScript Object Notation (JSON)
  HR, très utilisé, structure d'arbre, *pas de binaire*

# Exemple de format de sérialisation

- Comma Separated Value (CSV)
  HR, simple, bon compromis HR/compacité, pas de binaire, structure pauvre
- Extensible Markup Language (XML)
  HR, très utilisé, structure d'arbre, *pas de binaire, verbeux*
- JavaScript Object Notation (JSON)
  HR, très utilisé, structure d'arbre, *pas de binaire*
- S-expression
  HR/binaire, très simple à parser, structure d'arbre binaire,

# Exemple de format de sérialisation

- Comma Separated Value (CSV)
  HR, simple, bon compromis HR/compacité, pas de binaire, structure pauvre
- Extensible Markup Language (XML)
  HR, très utilisé, structure d'arbre, *pas de binaire, verbeux*
- JavaScript Object Notation (JSON)
  HR, très utilisé, structure d'arbre, *pas de binaire*
- S-expression
  HR/binaire, très simple à parser, structure d'arbre binaire,
- Protocol Buffers (protobuf)
  Compact, structure riche

# Exemple de format de sérialisation

- Comma Separated Value (CSV)
  HR, simple, bon compromis HR/compacité, pas de binaire, structure pauvre
- Extensible Markup Language (XML)
  HR, très utilisé, structure d'arbre, *pas de binaire, verbeux*
- JavaScript Object Notation (JSON)
  HR, très utilisé, structure d'arbre, *pas de binaire*
- S-expression
  HR/binaire, très simple à parser, structure d'arbre binaire,
- Protocol Buffers (protobuf)
  Compact, structure riche
- Format spécifique a un langage :
  Java Serializable, OCaml Marshalling, ...
  structure riche, automatique, liée à un langage, non pérenne

# Sérialisation des types primitifs

## Encoder les types primitifs

1. Booléen
2. Entier
3. Flottant
4. Caractère
5. String

# Sérialisation de structure / d'objet

## Sérialisation d'objets

- Un identifiant de la classe de l'objet
- Chaque membre est linéarisé dans l'ordre
- Un objet est serialisable si ses membres le sont

# Sérialisation de structure / d'objet

## Sérialisation d'objets

- Un identifiant de la classe de l'objet
- Chaque membre est linéarisé dans l'ordre
- Un objet est serialisable si ses membres le sont

## Problème des doublons

```
1 myObject1->a
2 myObject2->a
3 Serialize(myObject1, myObject2)
```

Combien de copie de "a" ?

# Sérialisation de structure / d'objet

## Sérialisation d'objets

- Un identifiant de la classe de l'objet
- Chaque membre est linéarisé dans l'ordre
- Un objet est serialisable si ses membres le sont
- Un identifiant de l'objet

## Problème des doublons

```
1  myObject1->a
2  myObject2->a
3  Serialize(myObject1, myObject2)
```

Combien de copie de "a" ?

# Exemple

```
1  class Msg implements Serializable {
2    int n;
3    String msg;
4    private Msg follow;
5
6    public Msg(int n, String msg) {
7      this.n = n;
8      this.msg=msg;
9      this.follow = null;
10   }
11   public Msg(int n, String msg, Msg follow) {
12     this.n = n;
13     this.msg=msg;
14     this.follow = follow;
15   }
16 }
```

# Méthode générale

## Chaque objet sait se “serialiser/deserialiser”

- Une méthode “serialise : Object -> ByteBuffer”
- Une méthode statique “deserialise : ByteBuffer -> Object”

# Méthode générale

## Chaque objet sait se “serialiser/deserialiser”

- Une méthode “serialise : Object -> ByteBuffer”
- Une méthode statique “deserialise : ByteBuffer -> Object”

## Exemple

```
1  public void serialize(ByteBuffer bb) {
2    bb.putInt(n);
3    Charset c = Charset.forName("UTF-16");
4    ByteBuffer cb = c.encode(msg);
5    bb.putInt(cb.remaining());
6    bb.put(cb);
7    if( follow != null){
8      bb.put((byte)1);
9      follow.serialize(bb);
10   }else bb.put((byte)0);
11 }
```

# Méthode générale - Exemple

## Exemple

```
1  public static Msg unserialize(ByteBuffer bb){
2    int n = bb.getInt();
3    int length = bb.getInt();
4    Charset c = Charset.forName("UTF-16");
5    int limit = bb.limit();
6    bb.limit(bb.position() +length);
7    String msg = c.decode(bb).toString();
8    bb.limit(limit);
9    Msg follow = null;
10   byte b = bb.get();
11   if(b>0)follow = unserialize(bb);
12   return new Msg(n,msg,follow);
13 }
```

# Méthode générale - Introspection

## Utilisation d'introspection

Un objet est déclaré Sérialisable.
Les fonctions de sérialisation / desérialisation sont générées.

# Méthode générale - Introspection

## Utilisation d'introspection

Un objet est déclaré Sérialisable.

Les fonctions de sérialisation / desérialisation sont générées.

```
1  Msg m = new Msg(3,"test",null);
2  Field fd[] = m.getClass().getDeclaredFields();
3  System.out.println(m.getClass() );
4  for(Field f : fd){
5     System.out.print(f.getType());
6     System.out.print(" ");
7     System.out.println(f.getName());
8  }
```

# Méthode générale - Introspection

## Utilisation d'introspection

Un objet est déclaré Sérialisable.

Les fonctions de sérialisation / desérialisation sont générées.

```
1  Msg m = new Msg(3,"test",null);
2  Field fd[] = m.getClass().getDeclaredFields();
3  System.out.println(m.getClass() );
4  for(Field f : fd){
5     System.out.print(f.getType());
6     System.out.print(" ");
7     System.out.println(f.getName());
8  }
```

---

```
1  class Msg
2  long serialVersionUID
3  int n
4  class java.lang.String msg
5  class Msg follow
```

---

# Problème des fonctions / méthodes

## Liée à la machine

- Code compilé $\Rightarrow$ pas portable
- Métodes $\Rightarrow$ besoin de toute la hiérarchie

# Problème des fonctions / méthodes

## Liée à la machine

- Code compilé $\Rightarrow$ pas portable
- Métodes $\Rightarrow$ besoin de toute la hiérarchie

## Classe connue des deux cotée de la sérialisation

Le fichier est compilé des deux côtés, un identifiant partagé.

# Exercice - Bencode

Bencode utilise des caractères ASCII comme délimiteur et chiffres.

- Un entier est encodé tel que “i<entier en base 10>e”.
- Une chaîne d’octets (string) est encodée “<longueur> :<contenu>”. La longueur est encodée en base 10, comme un entier.
- Une liste de valeurs est encodée “l<contenu>e”. Le contenu est constitué des éléments Bencodé de la liste dans l’ordre, concaténé.
- Un dictionnaire est encodé “d<contenu>e”. Les éléments sont chacun encodés chaque clé suivie de sa valeur. Toutes les clés doivent être constituées d’une chaîne de caractère et apparaître dans l’ordre lexicographique.

# Exercice - Bencode

Bencode utilise des caractères ASCII comme délimiteur et chiffres.

- Un entier est encodé tel que “i<entier en base 10>e”.
- Une chaîne d’octets (string) est encodée “<longueur> :<contenu>”. La longueur est encodée en base 10, comme un entier.
- Une liste de valeurs est encodée “l<contenu>e”. Le contenu est constitué des éléments Bencodé de la liste dans l’ordre, concaténé.
- Un dictionnaire est encodé “d<contenu>e”. Les éléments sont chacun encodés chaque clé suivie de sa valeur. Toutes les clés doivent être constituées d’une chaîne de caractère et apparaître dans l’ordre lexicographique.

## Question 1

Représenter l’encode du dictionnaire contenant [ “foo” -> 3 ; “bar” -> 32 ]

## Question 2

Écrire le code qui prend en entrée une chaîne de caractère et renvoi son encodage.

# Plan

# Using *threads*

## Définition

```
1  class Task implements Runnable {
2    public void run() {
3      //Execution
4    }
5  }
```

# Using *threads*

## Définition

```
1  class Task implements Runnable {
2    public void run() {
3      //Execution
4    }
5  }
```

## Création

```
1  Thread p = new Thread(new Task());
2  p.start();
```

# Using *threads*

## Définition

```
1  class Task implements Runnable {
2    public void run() {
3      //Execution
4    }
5  }
```

## Création

```
1  Thread p = new Thread(new Task());
2  p.start();
```

## Synchronisation

```
1  p.join();
```

# Exemple Thread

```
1  public class Client {
2    private SocketChannel sc;
3    private boolean connected = false;
4    public boolean isConnected(){ return connected;}
5    public void disconnected() { connected = false;}
6
7    public Client( String addr, int port ) throws Exception {
8      InetSocketAddress isa=new InetSocketAddress(addr, port);
9      if(isa.isUnresolved())throw
10       new Exception("Fail to resolve address:"+addr);
11     sc = SocketChannel.open();
12     sc.configureBlocking(true);
13     sc.connect(isa);
14     connected=true;
15   }
16   public void run() throws IOException{
17     Thread rnt = new Thread(new RepeatNetwork(sc, this));
18     Thread rkt = new Thread(new RepeatKeyboard(sc, this));
19     rnt.start();
20     rkt.start();
21     try {  rnt.join();  rkt.join();
22     } catch (InterruptedException e) {}
23   }
24 }
```

# Exemple Thread

```
1  class RepeatNetwork implements Runnable {
2    private SocketChannel sc;
3    private ByteBuffer buff;
4    private Client main;
5    public RepeatNetwork(SocketChannel sc, Client main){
6      this.sc =sc; this.main = main;
7      buff = ByteBuffer.allocateDirect(1024);
8    }
9    public void run() {
10     try {
11       while (main.isConnected()) {
12         buff.clear();
13         if (sc.read(buff) == -1)
14           throw new IOException("Connection Close");
15         buff.flip();
16         CharBuffer cb=Charset.forName("UTF-8").decode(buff);
17         System.out.println(cb.toString());
18       }
19     } catch (IOException e) {
20       main.disconnected();
21     };
22   }
23 }
```

# Exemple Thread

```
1  class RepeatKeyboard implements Runnable {
2    private SocketChannel sc;
3    private ReadableByteChannel keyboard;
4    private ByteBuffer buff;
5    private Client main;
6    public RepeatKeyboard(SocketChannel sc, Client main){
7      this.sc =sc; this.main = main;
8      buff = ByteBuffer.allocateDirect(1024);
9      keyboard = Channels.newChannel(System.in);
10   }
11
12   public void run() {
13     try {  while (main.isConnected()) {
14         buff.clear();
15         int n = keyboard.read(buff);
16         if (n == -1)
17         throw new IOException("User quit");
18         buff.flip();
19         while (buff.hasRemaining()) { sc.write(buff); }
20       } }catch(IOException e){
21       main.disconnected();
22     };
23   }
24 }
```

# Problème de concurrence

```
1  public class TestThread implements Runnable {
2    public static int t = 0;
3    public void run() {
4      for(int i=0; i< 10000; i++){
5        t = t + 1;
6      }
7    }
8    public static void main(String[] args) throws InterruptedEx
9      Thread t1 = new Thread(new TestThread());
10     Thread t2 = new Thread(new TestThread());
11     t1.start();
12     t2.start();
13
14     t1.join();
15     t2.join();
16
17     System.out.println("Value␣of␣t:"+ t );
18   }
19 }
```