TD/TME 8 : Sérialisation et Appels Distants

T. Lieu, P. Trébuchet, (R. Demangeon)

19 novembre 2013

L'objectif de cette séance de TD/TME est d'étudier les mécanismes inhérents à la sérialisation d'objets dans l'exercice 1 puis de s'intéresser à RMI dans l'exercice 2.La sérialisation des objets est un procédé qui permet de rendre un objet persistant, c'est-à-dire que l'objet est représenté sous une forme grâce à laquelle il pourra être reconstitué à l'identique après sauvegarde et lecture depuis un disque dur ou après transfert à travers un réseau. Dans ce dernier cas, l'objet peut être transmis par copie à une machine (virtuelle dans le cas de Java) "distante" de celle dans laquelle il a été initialement instancié. On parlera alors d'objets distants, un mécanisme repris dans le standard RMI (Remote Method Invocation) qui permet à plusieurs programmes écrits en Java de disposer des méthodes d'objets distants de manière quasi-transparente.

Exercice 1 – Sérialisation en Java

Question 1

Créer une classe `Individu` qui décrit un individu par 3 champs `(Nom, Prénom, Age)` et qui surcharge la méthode `toString()` pour renvoyer un texte type : *"Prénom Nom, Age an(s)"*.

Que faut-il faire pour que cette classe puisse être sérialisée ?

Solution:

Pour qu'une classe puisse etre serialisee il suffit de declarer qu'elle implante l'interface `java.io.Serializable`.

```
import java.io.*;

public class Individu implements java.io.Serializable{

    private String nom, prenom;
    private int age;

    public Individu(String nom, String prenom, int age){
        this.nom = nom;
        this.prenom = prenom;
        this.age = age;
    }

    public void setNom(String nom){this.nom = nom;}
    public void setPrenom(String prenom){this.prenom = prenom;}
    public void setAge(int age){this.age = age;}

    public String getNom(){return(this.nom);}
    public String getPrenom(){return(this.prenom);}
    public int getAge(){return(this.age);}

    public String toString(){return(nom + " " + prenom + ", " + age);}
```

```
}
```

Question 2

Ecrire une classe `CreationIndividu` qui permet l'instanciation d'un objet individu via la saisie au clavier de ses différents champs et qui sauvegarde les informations relatives à cet individu dans le fichier « individu.ser ». (On utilisera pour cela la méthode `writeObject()` de la classe `ObjectOutputStream`).

Solution:

```
import java.io.*;

public class CreationIndividu{

    public static void main(String[] argv){
        try{
            BufferedReader clavier = new BufferedReader(new InputStreamReader(System.in));
            String nom = clavier.readLine();
            String prenom = clavier.readLine();
            int age = new Integer (clavier.readLine()).intValue();

            Individu ind = new Individu(nom,prenom,age);
            FileOutputStream fichier = new FileOutputStream("individu.ser");
            ObjectOutputStream oos = new ObjectOutputStream(fichier);
            oos.writeObject(ind);
            oos.flush();
            oos.close();
        }
        catch (IOException ioe){ioe.printStackTrace();}
    }
}
```

Question 3

Ecrire une classe LectureIndividu qui instancie un objet Individu à partir des valeurs lues dans le fichier « individu.ser » et qui utilise la méthode toString de ce dernier pour indiquer son contenu. (On utilisera pour cela la méthode `readObject()` de la classe `ObjectInputStream`).

Solution:

```
import java.io.*;

public class LectureIndividu{

    public static void main(String argv[]) {
        try {
            FileInputStream fichier = new FileInputStream("individu.ser");
            ObjectInputStream ois = new ObjectInputStream(fichier);
            Individu ind = (Individu) ois.readObject();
            System.out.println(ind.toString());
        }
```

```
        catch (java.io.IOException ioe) {ioe.printStackTrace();}
        catch (ClassNotFoundException cnfe) {cnfe.printStackTrace();}
    }
}
```

Question 4

Ecrire une application de type client-serveur dont les fonctionnalités sont les suivantes : le serveur est en attente permanente de la transmission d'un objet Individu par le client sur le port 12345 et affiche les données relatives à cet individu lorsqu'un appel d'un client est accepté ; le client a pour simple rôle d'instancier un objet Individu et le transmettre au serveur.

Solution:

```
/* classe Client pour illustrer la sérialisation d'objets en Java via le réseau */
import java.io.*;
import java.net.*;

public class Client{

    public static void main(String[] argv){

        try{
            if (argv.length !=2) {
                System.out.println("Usage : java Client <hostame> <port>");
                System.exit(0);
            }
            String hostname = argv[0];
            int port = new Integer(argv[1]).intValue();

            Socket client = new Socket(hostname,port);
            Individu ind = new Individu("Monnom","Monprénom",20);
            ObjectOutputStream oos = new ObjectOutputStream(client.getOutputStream());
            oos.writeObject(ind);
            oos.flush();
            oos.close();
            System.out.println("Objet individu envoyé au serveur");
            System.out.println(ind.toString());
        }
        catch(Exception e){e.printStackTrace();}

    }
}
```

Pour le serveur

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

public class Serveur{
    public static void main(String[] argv){
        try{
            if (argv.length !=1) {
```

```
                System.out.println("Usage : java Serveur <port>");
                System.exit(0);
            }
            int port = new Integer(argv[0]).intValue();
            ServerSocket serv = new ServerSocket(port);
            System.out.println("Serveur en attente sur le port " + port);
            while (true){
                Socket client = serv.accept();
                ObjectInputStream ois = new ObjectInputStream(client.getInputStream());
                Individu ind = (Individu) ois.readObject();
                ois.close();
                System.out.println("Objet individu reçu par le serveur :");
                System.out.println(ind.toString());
                client.close();
            }
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Exercice 2 – RMI un premier exercice

Cet exercice a pour objectif de mettre en oeuvre les principes fondamentaux de RMI (Remote Method Invocation). On s'intéresse ici au développement d'un additionneur distant qui a les fonctionnalités suivantes :

- `eval(a,b)` prend 2 entiers (int ou Integer) en entrée et en retourne la somme ;
- `cumul(val)` permet le cumul des valeurs envoyées une par une ;
- `getCumul()` permet d'obtenir le cumul de toutes les valeurs fournies ;
- enfin `resetCumul()` permet de réinitialiser le cumul à 0.

Question 1 – Interface de l'objet

Ecrire l'interface de l'objet distant `IRemoteCalculette`.

Solution:

```
//IRemoteCalculette.java

import java.rmi.*;
public interface  IRemoteCalculette extends Remote{
public Integer eval(Integer op1, Integer op2)  throws RemoteException;
public void cumuler(Integer val)  throws RemoteException;
public Integer getCumul()  throws RemoteException;
public void resetCumul()  throws RemoteException;
}
```

Question 2 – Implantation de l'objet pour le serveur

Ecrire la classe RemoteCalculette implémentant cette interface par un objet contactable à distance.

Solution:

```
import java.util.*;
import java.rmi.*;
import java.rmi.server.*;

public class RemoteCalculette  extends UnicastRemoteObject // Objets distants
    implements IRemoteCalculette{

    int cumul = 0;

    public RemoteCalculette()  throws RemoteException{super();}
    public Integer eval(Integer op1, Integer op2)  throws RemoteException{
        return new Integer(op1.intValue() + op1.intValue()) ;
    }
    public void cumuler(Integer val)  throws RemoteException {
        cumul += val.intValue();
    }
    public Integer getCumul()  throws RemoteException {
        return new Integer(cumul);
    }
    public void resetCumul()  throws RemoteException {
        cumul = 0;
    }
}
```

Question 3 – Création des classes, de la couche (stub) et du squelette (skel)

Compiler RemoteCalculette (et, donc par dépendance, IRemoteCalculette). Exécuter ensuite la commande rmic RemoteCalculette, qui va créer `RemoteCalculette_Stub.class` ainsi que le fichier `RemoteCalculette_Skel.class`.

(TME uniquement) Utiliser `rmic -keepgenerated` pour obtenir les fichiers sources Java intermédiaires.

Solution:

Bon ici il s'agit juste de de tapper les commandes

Question 4 – Serveur de création de l'objet

Ecrire le serveur `ServCalculette` qui a pour rôle de : · créer une instance de `RemoteCalculette`, l'enregistrer sous le nom calculette auprès de la base des objets nommés contactables à distance du registre des objets. Dans un premier temps, on utilisera la base de localhost dont le serveur est supposé à l'écoute sur localhost sur le port `1234`.

Solution:

Ne pas oublier de dire rmiregistry !

```
import java.net.*;
import java.rmi.*;
import java.rmi.server.*;
public class ServCalculette { //classe ordinaire, coté serveur
    public static void main (String args[]) {
        String URL;

        if (args.length < 3)
            {URL="rmi://localhost:1234/calculette";
                System.err.println("AlistRegister : usage host port object"
                                   + " par défaut   " + URL);
```

```
                //                System.exit(1);
            }

        else { URL = "rmi://"+args[0]+":"+args[1]+"/"+args[2];}

        try {
            RemoteCalculette calculatrice = new RemoteCalculette();
            Naming.bind(URL,calculatrice);
        }
        catch(RemoteException e) {
            System.err.println(URL+":  serveur rmi indisponible"+"("+e.getMessage()+")");
            System.exit(1);
        }
        catch(AlreadyBoundException e) {
            System.err.println(URL+": nom déjà associé à un objet ");
            System.exit(1);
        }
        catch(MalformedURLException e) {
            System.err.println(URL+" :  URL syntaxiquement incorrecte");
            System.exit(1);
        }
    }
}
```

Parler ici de l'autonomie avec les objets de type rmiregistry

```
import java.net.*;
import java.rmi.*;
import java.rmi.registry.*;

/**
   cree un registre rmi, y enregistre un objet IRemoteCalculette de nom calc
   attend le numero de port en parametre ligne commande, defaut 1234
*/
public class ServAutonomeCalculette { //classe ordinaire, coté serveur
    public static void main (String args[]) {
        String URL;int port;

        try {
            if (args.length>0){port=Integer.valueOf(args[0]).intValue();}
            else {port =1234;}
            Registry reg= LocateRegistry.createRegistry(port);

            RemoteCalculette calculatrice = new RemoteCalculette();
            reg.rebind("/calc",calculatrice);
            System.out.println("apres bind");
            /*
              IRemoteCalculette calc = (IRemoteCalculette)
              reg.lookup("/calc");
              System.out.println("apres lookup");
              System.out.println("directe " +
              calculatrice.eval(new Integer(5),new Integer(5)) + " " +
              calc.eval(new Integer(5),new Integer(5)));
```

```
            */
            //System.out.println( calculatrice.getClientHost());

            //les groupes de threads

            //consultation du ThreadGroup
            // et du gpe parent
            int k,nb;Thread[] tt=new Thread[50];
            System.out.println("\tle groupe de thread courant");
            ThreadGroup g= Thread.currentThread().getThreadGroup();
            nb = g.enumerate(tt,false);
            for (k=0;k<nb;k++) System.out.println("\t" + tt[k].toString());
            System.out.println("\tle groupe de thread parent du groupe precedent");
            ThreadGroup gp = g.getParent();

            nb = gp.enumerate(tt,false);//pour eviter les precedents
            for (k=0;k<nb;k++) System.out.println("\t" + tt[k].toString());
            /*
              System.out.println("\tle groupe de thread parent du groupe precedent");
              ThreadGroup gpp = gp.getParent();

              nb = gpp.enumerate(tt,false);//pour eviter les precedents
              for (k=0;k<nb;k++) System.out.println("\t" + tt[k].toString());
            */


        }
        catch(RemoteException e) {
            System.err.println(":  serveur rmi indisponible"+"("
                               + e.getMessage()+")");
            System.exit(1);
        }
        catch(Exception e) {
            System.err.println(":  serveur rmi indisponible"+"("+
                               e.getMessage()+")");
            System.exit(1);
        }
    }
}

/* inutiles ici :

   catch(AlreadyBoundException e) {
   System.err.println(": nom déjà associé à un objet ");
   System.exit(1);
   }
   catch(MalformedURLException e) {
   System.err.println(" :  URL syntaxiquement incorrecte");
   System.exit(1);
   }
*/
```

Question 5 – Client utilisant l'objet distant

Ecrire une application utilisatrice de l'objet qui : • prend contact avec le serveur de la base d'objets pour obtenir une référence distante ; • réalise quelques calculs, cumule les nombres de 1 à 100 et récupère le résultat avant de faire un reset et en affiche les résultats sur la console client.

Solution:

Ne pas oublier de dire rmiregistry !

```
import java.net.*;
import java.rmi.*;

public class UtilCalculette{//classe ordinaire locale
    static String umsg =
        "UtilCalculette : usage host port object ";
    public static void main(String args[]) {
        String URL;

        if (args.length < 3)
            { URL="rmi://localhost:1234/calculette";
                System.err.println(umsg + "val défaut : " + URL);
                //               System.exit(1);

            }
        else { System.out.println(args.length +  args[0] + args[1] +args[2]);
            URL = "rmi://"+args[0]+":"+args[1]+"/"+args[2];}

        try{
            //obtenir un représentant local de l'objet distant
            IRemoteCalculette calc = (IRemoteCalculette)Naming.lookup(URL);

            Integer res = calc.eval ( new Integer(3),  new Integer(3));
            System.out.println("resultat 3+3 calculé par l'objet distant : " +
                               res.intValue());
            int k;
            for (k=1 ;k <= 10 ;k++)
                {calc.cumuler(new Integer(k));}
            res = calc.getCumul();
            System.out.println("resultat cumul 1 à 10 calculé par l'objet distant : " +
                               res.intValue());
            calc.resetCumul();res = calc.getCumul();
            System.out.println("resultat apres reinitialisation de l'objet distant : " +
                               res.intValue());
        }
        catch(RemoteException e) {
            System.err.println(URL+" : serveur rmi indisponible "+"("+e.getMessage()+")");
            System.exit(1);
        }
        catch(NotBoundException e) {
            System.err.println(URL+" : nom inconnu sur serveur rmi"+"("+e.getMessage()+")");
            System.exit(1);
        }
        catch(MalformedURLException e) {
            System.err.println(URL+"  URL syntaxiquement incorrecte");
```

```
            System.exit(1);
        }
    }
}
```

Expliquer ici aussi l'autonomie

```
import java.net.*;
import java.rmi.*;
import java.rmi.registry.*;
public class ClientPourServAutonomeCalc { //classe ordinaire, coté serveur
    public static void main (String args[]) {
        String URL="";
        int port;

        try {
            if (args.length>0){port=Integer.valueOf(args[0]).intValue();}
            else {port =1234;}

            Registry reg= LocateRegistry.getRegistry(port);


            IRemoteCalculette calc = (IRemoteCalculette)
                reg.lookup("/calc");
            System.out.println("apres lookup");
            System.out.println("apres eval " +
                               calc.eval(new Integer(5),new Integer(5)));

            //les groupes de threads

            //consultation du ThreadGroup
            // et du gpe parent
            int k,nb;Thread[] tt=new Thread[50];
            System.out.println("\tle groupe de thread courant");
            ThreadGroup g= Thread.currentThread().getThreadGroup();
            nb = g.enumerate(tt,false);
            for (k=0;k<nb;k++) System.out.println("\t" + tt[k].toString());
            System.out.println("\tle groupe de thread parent du groupe precedent");
            ThreadGroup gp = g.getParent();

            nb = gp.enumerate(tt,false);//pour eviter les precedents
            for (k=0;k<nb;k++) System.out.println("\t" + tt[k].toString());


        }
        catch(RemoteException e) {
            System.err.println(URL+":  serveur rmi indisponible"+"("+e.getMessage()+")");
            System.exit(1);
        }

        catch(Exception e) {
            System.err.println(URL+":  serveur rmi indisponible"+"("+e.getMessage()+")");
            System.exit(1);
```

```
        }
    }
}
```

Question 6 – Conception d'un fichier de règles de sécurité

Ecrire un fichier nommé `Calculette.policy` assouplissant les règles par défaut en :

- autorisant et acceptant la connection au port 1234 quelle que soit l'origine de la requête,
- autorisant la connection au port 80.

Solution:

```
grant{
permission java.net.SocketPermission "*:1024-65535", "connect,accept";
permission java.net.SocketPermission "*:80", "connect";
};
```

Question 7 – Exécution (TME uniquement)

Toutes les classes sont supposées compilées.

- Créer par rmic les classes couches et squelettes de l'objet distant RemoteCalculette
- Lancer le serveur de la base d'objets distants par rmiregistry 1234 & ;
- Lancer le serveur d'objet en pensant au gestionnaire de sécurité ;
- Lancer leclient en pensant au gestionnaire de sécurité.

Exercice 3 – RPC un petit exemple

Dans cet exercice nous allons mettre en place un service de RPC, qui à la reception d'une chaine de caractères renvoie sa longueur.

Question 1

Ecrire un fichier IDL pour les RPC, décrivant les types et les fonctions pour notre module RPC :

- déclaration d'une fonction prenant une `string` et rendant un `int`.

Solution:

```
program LONGUEUR{
  version VERSION_UN{
    int CALCUL_LONGUEUR(string<>) = 1;
  } = 1;
} = 0x20000001;
```

Question 2

Compléter la fonction `calcul_addition_1_svc` du fichier `long-server.c` et compiler le serveur.

Solution:

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
```

```
 */

#include<string.h>
#include "long.h"

int *
calcul_addition_1_svc(char **argp, struct svc_req *rqstp)
{
        static int  result;

        result=strlen(*argp);

        return &result;
}
```

Question 3

Compléter/Modifier le fichier `_client.c` pour que la fonction fasse l'appel RPC et affiche le resultat obtenu.

Solution:

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "long.h"


int
longueur_1(char *host,char *param)
{
        CLIENT *clnt;
        int  *result_1;
        char * calcul_longueur_1_arg;

#ifndef DEBUG
        clnt = clnt_create (host, LONGUEUR, VERSION_UN, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */
        /* ICI C LE DEBUT DU RAJOUT */
        result_1 = calcul_longueur_1(&param, clnt);
        /*ICI C LA FIN */
        if (result_1 == (int *) NULL) {
                clnt_perror (clnt, "call failed");
        }
#ifndef DEBUG
        clnt_destroy (clnt);
```

```
#endif   /* DEBUG */
        return *result_1;
}


int
main (int argc, char *argv[])
{
        char *host;
        /* ICI C LE DEBUT DU RAJOUT */
        int res;
        /* ICI C LA FIN */
        if (argc < 2) {
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
        host = argv[1];
        /* ICI C LE DEBUT DU RAJOUT */
        res=longueur_1 (host,argv[2]);
        printf("resultat %d\n",res);
        /* ICI C LA FIN */
}
```

Question 4

Lancer le serveur sur votre machine, puis le client et verifier que tout se passe bien.