Recursive and non-recursive method to serialize object at J2ME

Abstract

This paper shows serialization at J2ME in two ways, recursively and non-recursively. This paper shows that recursive methods are not appropriated for small platform. A comparative measures of recursive and non-recursive is shown to a serialization methods.

Fname:Antonio J.
Surname:Sierra
Jobtitle:Assistant Professor
Address:
Affil:University of Sevilla
Subaffil:Escuela Superior de Ingenieros
Departament of Sistemas and Automatic
Area of Engineering Telematic
C/Camino de los Descubrimientos s/n
City:Sevilla
Cntry:Spain
Postcode:41092
Phone:+34 95 446 73 68
Fax:+34 95 448 73 85
Email:antonio@trajano.us.es
Web:http://trajano.us.es/~antonio/

Introduction

Many of today's distributed systems use objects as the means of communication between nodes.

Object serialization is the ability of an object to write a complete state of itself and of any object that it references to an output stream, so that it can be recreated from the serialized representation at a later time.

Pickling is the process of creating a serialized representation of objects.

Tree (or like-tree) structures, such as, internal representation of XML document or abstract data types of serialization, is prone to use method's recursive.

J2ME technology, Java 2 Micro Edition, specifically addresses the vast consumer space, which covers the range of extremely tiny commodities such as smart cards or a pager all the way up to the set-top box, an appliance almost as powerful as a computer.

Under the JavaTM 2 Micro Edition (J2ME) technology, two notions have been introduced: a **configuration**, and a **profile**.

- A *configuration* is defined as the combination of a Virtual Machine (VM) and "core" APIs that represent an underlying development platform for a broad class of devices.
- A *profile* is defined as a set of APIs for a specific vertical market and relies upon an underlying configuration's capabilities to create new, market-specific APIs.

Mobile Information Device Profile (MIDP), define a *profile* that will extend and enhance the "J2ME Connected, Limited Device Configuration" [H]. By building upon this *configuration*, this *profile* will provide a standard platform for small, resource-limited, wireless-connected mobile information devices.

The CLDC does not support object serialization. This means that there is no built-in way to serialize objects into a byte stream for CLDC-based profiles. Applications that need to serialize objects must build their own serialization mechanisms.

Method's recursive can hurt scalability on object serialization's algorithm.

This paper presents a comparative of a recursive and a non-recursive serialization's method at J2ME.

Serialization

Related Work

J2SE provides a generic pickling mechanism, which is able to serialize and deserialize any object that implements the **java.io.Serializable** interface.

A method for transferring Abstract Data Types is presented in

They are many problems that have not been addressed by current implementation. One of them has to do with performance and specifically with the **size** of pickles produced by the system.

When serialization is used to transfer objects in a distributed application, **large pickles** adversely affect the performance because of greater network latencies for large messages.

Many algorithms are expressed most concisely as tail-recursive methods. The tail-recursive methods in the Java language can result in unexpectedly large memory usage, and can be inefficient because numerous recursive calls risk overlowing the stack. The stack size of KVM is very small.

The two Algorithms

This section presents the two algorithms to serialize (deserialize) recursively and non-recursively.

Both methods use the interface **Serializable.** Any object to be serialized must have implemented the two methods of interface **Serializable:**

- **serialize**, to serialize an object, and
- **deSerialize**, to deserialize an object.

The Figure 1 shows the interface **Serializable**, for a recursive algorithm. We can see that both methods (**serialize/deSerialize**) uses only one parameter (**DataInputStream/DataOutputStream**) to read/write the simple data types.

```
import java.io.IOException;
import java.io.DataOutputStream;
import java.io.DataInputStream;


public interface Serializable {
      public void serialize(DataOutputStream dos) throws IOException;
      public void deSerialize(DataInputStream dis)
             throws IOException, ClassNotFoundException,
             IllegalAccessException,  InstantiationException;
}
```

Figure 1: Interface Serializable for recursive algorithm.

Appendix E shows a class "**Antonio**" that implements the interface of Figure 1. This class has an integer as attribute and a reference to the same type of the class (**Antonio**).
The **serialize** method is recursive, write the attribute (integer) and call recursively to the **serialize** method of the reference.

The Figure 2 shows the interface **Serializable**, for a non-recursive algorithm. We can see that both methods (**serialize/deSerialize**) uses two parameters (**DataInputStream/DataOutputStream** and **Stack**) to read/write the simple data types and to maintain the references of the objects associated with the actual object.

```
import java.io.IOException;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.util.Stack;

 public interface Serializable {
       public void serialize(DataOutputStream dos,Stack stack)
                 throws IOException;
       public void deSerialize(DataInputStream dis,Stack stack)
                   throws IOException, ClassNotFoundException,
                   IllegalAccessException, InstantiationException;
 }
```

Figure 2: Interface Serializable for non-recursive algorithm.

Note that this two parameter can be avoided using a public attribute in other class.

Appendix A shows a class "**Antonio**" that implements the interface of Figure 2. This class has an integer as attribute and a reference to the same type of the class (**Antonio**).

Any object can be preceded by an integer number to determine the name of any class.

Any attribute's value can be preceded by an integer number to determine the data type.

```
null      = 0;
int       = 1;
long      = 2;
boolean   = 3;
String    = 4;
byte      = 5;
char      = 6;
short     = 7;
double    = 8;
float     = 9;
Object    = 0x7F;
```

Optimization of this method can be use a byte unless an integer. An integer has four byte, so they are less we translate information's object more efficiently.

But we implement ourselves the serialization methods and the order in which in implements has information of the attribute that is considered in any moment, so we can avoid this number in preceding any attribute.

An implementation of a non-recursive serialization algorithm is shown in Appendix A. Note that the attribute is not preceding by an integer (byte) number. If the object is **null** we write **0x0**, if the object is not **null**, we write **0x7F** and push in the stack the object to serialize.

DataOutputStream/DataInputStream

To write a simple data type we can use DataOutputStream's (DataInputStream) methods

Simple Datatype	Method
boolean	**void writeBoolean(boolean v)**
byte	**void writeByte(int v)**
short	**void writeShort(int v)**
char	**void writeChar(int v)**
int	**void writeInt(int v)**
long	**void writeLong(long v)**
float	**void writeFloat(float v)**
double	**void writeDouble(double v)**
double	**void writeUTF(String str)**

Figure 3: DataOutputStream's methods to write simple datatypes.
All this methods can throw **IOException.**

(rec)ObjectOutputStream/(rec)ObjectInputStream

To write/read an object we use as support the **ObjectOutputStream/ObjectInputStream** class. These classes are shown in the Appendix B, and Appendix D, respectively. The recursive option is supported by **recObjectOutputStream/recObjectInputStream** class. These classes are shown in the Appendix G, and Appendix H, respectively.

These classes extend the class Thread.
(**rec**)**ObjectOutputStream** class uses a constructor with the object to serialize/deserialize and the stream to write the object's transformation.

(**rec**)**ObjectInputStream** class uses a constructor with the stream to read the object's transformed. We can use getObject (**public Object getObject()**) to obtain the object after deserialized.

Creating a serialized representation of an object.

We need to establish different end-point to write and read an object. The Appendix C (for the non-recursive) and Appendix F (for the recursive) show the code for creating a transformation objects. This transformation is save in a file. This file is move to a serve.

Scenario

The file (**ser_p256, in the previous case**) is put in the network (**http://trajano.us.es/~antonio/ser_p256**). The code at Appendix C and Appendix I is executed in at Wireless Toolkit Version 2.2 Beta, to read the object and translate to a Java object. Different depth of recursion is proved in the measures.

Results

This section presents the measures of time to realize the process to translate an object from the serialize format to a Java object. The Figure 4 and Figure 5 also shown the size of serialized format obtain with the code of Appendix F and Appendix C. These tables also shown the memory used to realize this process. We can see that the non-recursive algorithm present a better performance in time (less time to execute) and memory used.

Measures for non-recursive:

# object	**256 Objects**	**100 Objects**	**50 Objects**
Size of serialized format	1.290 bytes	510 bytes	260 bytes
Time to process	~240 miliseg.	~230 miliseg.	~220 miliseg.
Memory used	6532 bytes	6916 bytes	6188 bytes

Figure 4: Measures for the non-recursive algorithm.

Recursive:

# object	**256 Object**	**100 Object**	**50 Objects**
Size of serialized format	4.356 bytes	2104 bytes	1054 bytes
Time to process	~8072 mseg.	~811 mseg.	~341 mseg.
Memory used	87744 bytes	34304 bytes	24560 bytes

Figure 5: Measures for the recursive algorithm.

The Figure 6 show the time used to serialize object.

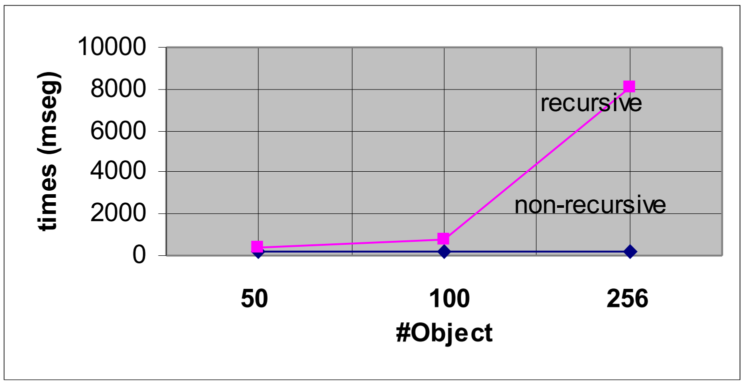

Figure 6: Time to execute both algorithms.

Conclusions

We does not use recursive methods at J2ME.

Two reason:

- Time (in execute Java code) and
- memory (recursive methods uses more memory).

References

[M] Eric Allen, Tail-recursive transformations can speed up your apps, but not all JVMs can perform the task, http://www-128.ibm.com/developerworks/library/j-diag8.html.

[A] Arnold, Ken, and James Gosling, The Java Programming Language, Addison-Wesley (1996).

[B] Birrell, Andrew, Michael B. Jones, and Edward P. Wobber, A simple and efficient implementation for small databases, Digital Equipment Corporation Systems Research Center Technical Report 24 (1987).

[C] Birrell, Andrew, Greg Nelson, Susan Owicki, and Edward Wobber, Network Objects. Digital Equipment Corporation Systems Research Center Technical Report 115 (1994).

[D] Gosling, James, and Bill Joy, Guy Steele, The Java Language Specification, in preparation.

[E] Herlihy, M and B. Liskov, A Value Transmission Method for Abstract Data Types, ACM Transactions on Programming Languages and Systems, Volume 4, Number 4, (1982).

[F] Lindholm, Tim and Frank Yellin, The Java Virtual Machine Specification, Addison-Wesley (1996), http://java.sun.com/docs/books/vmspec/.

[H]Antero Taivalsaari, JSR 30: J2ME Connected, Limited Device Configuration, http://jcp.org/en/jsr/detail?id=30.

[I] Jim Van Peursem, JSR 37: Mobile Information Device Profile for the J2ME Platform, http://jcp.org/en/jsr/detail?id=37.

[J] Wireless Tech Tips, J2ME Tech Tips: February 26, 2002. http://java.sun.com/developer/J2METechTips/2002/tt0226.html.

[L] Wollrath, Ann and Roger Riggs, Jim Waldo, A Distributed Object Model for the Java system, Proceedings of the USENIX 2nd Conference on Object-Oriented Technologies and Systems (1996).

Apendix A

Object to serialice/deserialice

```
import java.io.IOException;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.util.Stack;


public class Antonio implements Serializable {
    int valor = 14;
    Antonio ant = null;

    public Antonio () {}

    public void setValue(int i){
      this.valor = i;
    }
    public void setAnt(Antonio a){
      this.ant = a;
    }

    public void serialize(DataOutputStream dos,Stack stack) throws
IOException{
        dos.writeInt(this.valor);

        if(ant == null){
            dos.writeByte((byte)0x0);
        }else{
            dos.writeByte((byte)0x7F);
            stack.push(this.ant);
        }
    }

    public void deSerialize(DataInputStream dis,Stack stack)
                 throws IOException, ClassNotFoundException,
                 IllegalAccessException, InstantiationException{

        //this.valor =
        this.setValue(dis.readInt());

        if(0x0 != dis.readByte()){
            this.ant =
(Antonio)Class.forName("Antonio").newInstance();
            stack.push(this.ant);
        }
    }

}


        if (aux == 9){
            String s = dis.readUTF();
            this.ant = (Antonio)Class.forName(s).newInstance();
            ((Serializable )this.ant).deSerialize(dis);
        }else if(aux == 0){
            this.ant = null;
        }
    }
```

```
}
```

Apendix B

To generates serialized representation of an object in a non recursive process.

```
import java.util.Stack;
import java.io.DataOutputStream;
import java.io.IOException;

public class ObjectOutputStream extends Thread {

    private Object o;
    private DataOutputStream out;
    private Stack stack = null;

    public ObjectOutputStream (Object o, DataOutputStream ot){
        this.o   = o;
        this.out = ot;
        stack = new Stack();
        start();
    }

    public void run(){
        int tmp = 127;
        try{
            if(o != null){
                out.writeByte(tmp);
                out.writeUTF(o.getClass().getName());
                ((Serializable)o).serialize(out,stack);
                while( !(stack.empty()) ){
                    Object aux = stack.pop();
                     ((Serializable)aux).serialize(out,stack);
                }
            }else{
                tmp = 0;
                out.writeByte(tmp);
            }
        }catch(IOException e){
           e.printStackTrace();
        }
    }
}
```

Apendix C

To generates de serialized representation of an object in a file. Note that in this example generates the files **`ser_p256`**. This process is executed at J2SE.

```
import java.io.DataOutputStream;
import java.io.FileOutputStream;

public class Servidor {

    public static void main(String[] args) {
        Antonio prim = null;
        Antonio aux = new Antonio();
        prim = aux;
        aux.setValue(0);
        Antonio ant = aux;
        for(int i = 1; i<256 ;i++){
            aux = new Antonio();
            aux.setValue(i);
            ant.setAnt(aux);
            ant = aux;
        }
        muestra(prim);
        try{
            FileOutputStream os = new FileOutputStream("ser_p256");
            DataOutputStream o = new DataOutputStream(os);

            ObjectOutputStream oos = new ObjectOutputStream(prim, o);
            oos.join();
            os.close();
        } catch (Exception e){
                   System.out.println(e);
        }

    }
    static void muestra (Antonio a){
        System.out.println("Muestra el objeto");
        while(a!= null){
            System.out.println("Campo entero = " + a.valor);
            System.out.println();
            a = a.ant;
        }
    }
 }
```

Apendix C

The file (**`ser_p256, in the previous case`**) is put in the network. The following code is executed in at Wireless Toolkit Version 2.2 Beta.

```
import javax.microedition.midlet.MIDlet;
import java.io.InputStream;
import java.io.DataInputStream;
import javax.microedition.io.HttpConnection;
import javax.microedition.io.Connector;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Form;
import java.io.InputStream;
import java.io.ByteArrayInputStream;

public class ser_p256 extends MIDlet implements CommandListener {
    ObjectInputStream ois = null;
    Antonio v1 = null;
    Form mainForm = new Form ("ser_p256");

    public ser_p256 (){
        Command cmdExit = new Command("Salida",Command.EXIT,1);
        mainForm.addCommand(cmdExit );
        mainForm.setCommandListener(this);
    }
   public void commandAction(Command c, Displayable d) {
       if(c.getCommandType() == Command.EXIT){
            destroyApp(true);
            notifyDestroyed();
            return;
       }
   }

    public void startApp () {
      Display.getDisplay (this).setCurrent (mainForm);

           try {
               Runtime rr = Runtime.getRuntime();
               long mem1, mem2;
               mem1 = rr.freeMemory();

               HttpConnection http = (HttpConnection)
Connector.open("http://trajano.us.es/~antonio/ser_p256");

               long antes = System.currentTimeMillis();

               byte []tabla = new byte[(int)http.getLength()];

               InputStream is = http.openInputStream();
               is.read(tabla);
               ByteArrayInputStream iis = new ByteArrayInputStream
(tabla);
               ois = new ObjectInputStream(new DataInputStream(iis));

               long despues = System.currentTimeMillis();

               System.out.println("tiempo = " +(despues -antes ));
               mem2 = rr.freeMemory();
               System.out.println("memoria1Libre ="+mem1);
               System.out.println("memoria2Libre ="+mem2);
               System.out.println("memoriausada ="+(mem1-mem2));
            }catch(Exception e){
                e.printStackTrace();
            }
    }
    public void pauseApp () { }

    public void destroyApp (boolean unconditional) {
               v1 =  (Antonio)ois.getObject();
    }
}
```

Apendix D

Non-recursive process

```
//package com.sierra.Serializable;

//import com.sierra.Serializable.Serializable;
import java.util.Stack;
import java.io.DataInputStream;
import java.io.IOException;

public class ObjectInputStream extends Thread{

    private DataInputStream in = null;
    private Object o = null;


    public ObjectInputStream(DataInputStream in){
        this.in = in;
        try{
            if(0x0 != in.readByte()){
                o = Class.forName(in.readUTF()).newInstance();
                //System.out.println("Entra");
                start();
            }
        }catch(Exception e){}
    }

    public void run(){

        Stack stack = new Stack();
        //leo el nombre de la clase
        //String nombreObj = null;

        try{

            ((Serializable)o).deSerialize(in, stack);
            while( !(stack.empty()) ){
                Object local = stack.pop();
                ((Serializable)local).deSerialize(in, stack );
            }
        }catch(Exception e){}
    }

    public Object getObject(){
        return o;
    }
}
```

Apendix E

Object to serialize recursively.

```
import java.io.IOException;
import java.io.DataOutputStream;
import java.io.DataInputStream;


public class Antonio implements Serializable {
    int valor = 14;
    Antonio ant = null;

    public Antonio () {}

    public void setValue(int i){
      this.valor = i;
    }
    public void setAnt(Antonio a){
      this.ant = a;
    }

    public void serialize(DataOutputStream dos) throws IOException{

        //dos.writeInt(1);         //es un entero (redundante)
        dos.writeInt(this.valor);

        if(ant != null){
            dos.writeInt(9);        //lo que viene es
serializable(redundante)
            dos.writeUTF(ant.getClass().getName());
            ((Serializable )ant).serialize(dos);
        }else{
            dos.writeInt(0);
        }
    }

    public void deSerialize(DataInputStream dis)throws IOException,
        ClassNotFoundException, IllegalAccessException,
InstantiationException{
        int aux = -1;

        dis.readInt();  //leo 1, que indica que es un ENTERO
        this.valor = dis.readInt();

        aux = dis.readInt();  //leo 9, que indica que es un
SERIALIZABLE

        if (aux == 9){
            String s = dis.readUTF();
            this.ant = (Antonio)Class.forName(s).newInstance();
            ((Serializable )this.ant).deSerialize(dis);
        }else if(aux == 0){
            this.ant = null;
        }
    }
}
```

Apendix F

```
import java.io.FileOutputStream;
import java.io.FileInputStream;

public class Servidor {

       public static void main(String[] args) {
             Antonio prim = null;
             Antonio aux = new Antonio();
             prim = aux;
             aux.setValue(0);
             Antonio ant = aux;
             for(int i = 1; i<256 ;i++){
                 aux = new Antonio();
                 aux.setValue(i);
                 ant.setAnt(aux);
                 ant = aux;
             }
             muestra(prim);
       try{
            FileOutputStream os = new FileOutputStream("rec_p256");
            recObjectOutputStream oos = new recObjectOutputStream(os);
            oos.writeObject(prim);
            oos.join();
            os.close();

            FileInputStream in = new FileInputStream("rec_p256");
            recObjectInputStream iis =  new recObjectInputStream(in);
            prim = (Antonio)iis.readObject();
            oos.join();
            in.close();
            muestra(prim);

       } catch (Exception e){
           System.out.println(e);
           e.printStackTrace();
       }

       }
       static void muestra (Antonio a){
           System.out.println("Muestra el objeto");
           while(a!= null){
              System.out.println("Campo entero = " + a.valor);
              System.out.println();
              a = a.ant;
        }
    }
 }
```

Apendix G

```
import java.io.InputStream;
import java.io.IOException;
import java.io.DataInputStream;

public class recObjectInputStream {

    private DataInputStream dis;

    public recObjectInputStream(InputStream is) {
        this.dis = new DataInputStream(is);
    }
    public Object readObject() throws IOException,
       ClassNotFoundException, IllegalAccessException,
           InstantiationException {
        Object obj = null;
        int tmp = -1;

        tmp = this.dis.readInt();
        if(tmp == 9){
            obj = (Serializable)
                Class.forName(dis.readUTF()).newInstance();
            ((Serializable)obj).deSerialize(dis);
        }else
           obj = null;
        return obj;
   }
}
```

Apendix H

```
import java.io.InputStream;
import java.io.IOException;
import java.io.DataInputStream;

public class recObjectInputStream {

    private DataInputStream dis;

    public recObjectInputStream(InputStream is) {
        this.dis = new DataInputStream(is);
    }
    public Object readObject() throws IOException,
       ClassNotFoundException, IllegalAccessException,
           InstantiationException {
        Object obj = null;
        int tmp = -1;

        tmp = this.dis.readInt();
System.out.println("en recObjectInputStream, tmp = " + tmp);
        if(tmp == 9){
            obj = (Serializable)
                Class.forName(dis.readUTF()).newInstance();
            ((Serializable)obj).deSerialize(dis);
        }else
           obj = null;
        return obj;
   }
}
```

Apendix I

```
import javax.microedition.midlet.MIDlet;
import java.io.InputStream;
import java.io.DataInputStream;
import javax.microedition.io.HttpConnection;
import javax.microedition.io.Connector;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Form;
import java.io.InputStream;
import java.io.ByteArrayInputStream;

public class rec_p256 extends MIDlet implements CommandListener {

    Antonio v2 = null;

    Form mainForm = new Form ("rec_p256");
    public void pauseApp () {
    }

    public void destroyApp (boolean unconditional) {
      // muestra(v2);
    }
    public rec_p256(){
        Command cmdExit = new Command("Salida",Command.EXIT,1);
        mainForm.addCommand(cmdExit );
        mainForm.setCommandListener(this);
    }
    public void commandAction(Command c, Displayable d) {
        if(c.getCommandType() == Command.EXIT){
            destroyApp(false);
            notifyDestroyed();
            return;
        }
    }


    public void startApp () {
        Runtime r = Runtime.getRuntime();
        System.out.println("memoria total= "+ r.totalMemory());
        System.out.println("memoria free = "+ r.freeMemory());

        Display.getDisplay (this).setCurrent (mainForm);
           v2 = new Antonio();
           try {
               HttpConnection http =
               (HttpConnection)
                           Connector.open("http://trajano.us.es/~anton
io/rec_p256");
               InputStream is = http.openInputStream();

               byte []tabla = new byte[(int)http.getLength()];
               long antes = System.currentTimeMillis();
               is.read(tabla);
               ByteArrayInputStream iis = new ByteArrayInputStream
(tabla);
```

```
               recObjectInputStream ois =  new 
recObjectInputStream(new DataInputStream(iis));
               v2 = (Antonio)ois.readObject();
               long despues = System.currentTimeMillis();
               System.out.println("tiempo = " +(despues -antes ));
               System.out.println("memoria total= "+ r.totalMemory());
               System.out.println("memoria free = "+ r.freeMemory());
               System.out.println("memoria total-libre = "+ 
(r.totalMemory()-r.freeMemory()));
```