# TODO

# Serializable

一、源码解析

public interface Serializable {

}

是一个没有任何待实现方法的接口,作用是提供一个标志而已

序列化的机理，看一下使用ObjectOutputStream类持久化一个对象的过程，writeObject0(Object obj, boolean unshared) 函数的源码1170~1185行(JDK 1.7.0\_45)

if (obj instanceof String) {//注意语句1 writeString((String) obj, unshared); } elseif (cl.isArray()) { writeArray(obj, desc, unshared); } elseif (obj instanceof Enum) { writeEnum((Enum) obj, desc, unshared); } elseif (obj instanceof Serializable) {//注意语句2

writeOrdinaryObject(obj, desc, unshared);

} else { if (extendedDebugInfo) { thrownew NotSerializableException( cl.getName() + "\n" + debugInfoStack.toString()); } else { thrownew NotSerializableException(cl.getName()); } }

对象如果是String、是数组、是枚举、**是Serializable**就相应的函数把对象写成文件否则抛出错

注意多个类的顺序判断很重要(//注意语句1和注意语句2顺序不能换了)

|  |
| --- |
| String s = "I AM an Object!";    System.out.println( s instanceof Object);//true    System.out.println( s instanceof Serializable);//true    System.out.println( s instanceof Comparable<?>);//true    System.out.println( s instanceof CharSequence );//true |

二、demo

package myjava.lang;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.Serializable;

public class SerializableDemo implements Serializable {

   private static final long serialVersionUID = -3450064362986273896L;

   private int myint = 1;

   public int getMyint() {

       return myint;

   }

   public void setMyint(int myint) {

       this.myint = myint;

   }

   public int public\_int=2;

   static int static\_int = 3;

   transient int transient\_int = 4;

   public static void main(String[] args) {

       SerializableDemo myBox=new SerializableDemo();

       myBox.setMyint(10);

       myBox.public\_int = 20;

       myBox.static\_int = 30;

       myBox.transient\_int = 40;

       try {

           FileOutputStream fs=new FileOutputStream("foo.ser");

           ObjectOutputStream os=new ObjectOutputStream(fs);

           os.writeObject(myBox);

           os.close();

           myBox.static\_int =3;

           FileInputStream fi=new FileInputStream("foo.ser");

           ObjectInputStream oi=new ObjectInputStream(fi);

           SerializableDemo box=(SerializableDemo)oi.readObject();

           oi.close();

           System.out.println(

            " Myint="+box.getMyint()

            +"\n public\_int="+box.public\_int

            +"\n static\_int="+box.static\_int

            +"\n transient\_int="+box.transient\_int);

       } catch (Exception e) {

           e.printStackTrace();

       }

   }

}

输出:

Myint=10

public\_int=20

static\_int=3

transient\_int=0

**相关注意事项**

　　a）当一个父类实现序列化，子类自动实现序列化，不需要显式实现Serializable接口；

　　b）当一个对象的实例变量引用其他对象，序列化该对象时也必须手动把引用对象进行序列化；

　　c）并非所有的对象都可以序列化，至于为什么不可以，有很多原因了,比如：

　　1.安全方面的原因，比如一个对象拥有private，public等field，对于一个要传输的对象，比如写到文件，或者进行rmi传输 等等，在序列化进行传输的过程中，这个对象的private等域是不受保护的。

　　2. 资源分配方面的原因，比如socket，thread类，如果可以序列化，进行传输或者保存，也无法对他们进行重新的资源分配，而且，也是没有必要这样实现。

　 d).对于可序列化类中的static、transient对象变量，在序列化时无法保存其状态或值，static对象变量在反序列化时取得的值为当前jvm中对应类中对应static变量的值，而transient（瞬态）关键字则一般用于标识那些在序列化时不需要传递的状态变量

(e)serialVersionUID作用

:意思是序列化的版本号。

【作用】一些java类中为什么需要重载 serialVersionUID 属性。在Java中，软件的兼容性是一个大问题，尤其在使用到对象串行性的时候，那么在某一个对象已经被串行化了，可是这个对象又被修改后重新部署了，那么在这种情况下， 用老软件来读取新文件格式虽然不是什么难事，但是有可能丢失一些信息。serialVersionUID来解决这些问题，新增的serialVersionUID必须定义成下面这种形式：static final long serialVersionUID=-2805284943658356093L;。其中数字后面加上的L表示这是一个long值。 通过这种方式来解决不同的版本之间的串行话问题。

【生成方式】它有两种显示的生成方式：

一个是默认的1L，比如：private static final long serialVersionUID = 1L;。

一个是根据类名、接口名、成员方法及属性等来生成一个64位的哈希字段，

比如：private static final long serialVersionUID = xxxxL

三、一个重要的应用 -深度拷贝的实现

|  |
| --- |
| package myjava.lang;  import java.io.ByteArrayInputStream;  import java.io.ByteArrayOutputStream;  import java.io.IOException;  import java.io.ObjectInputStream;  import java.io.ObjectOutputStream;  import java.io.Serializable;  public class SerializableTest2  {      public static void main(String[] args) throws Exception      {          Teacher3 t = new Teacher3();          t.setName("Teacher Wang");          t.setAge(50);          Student3 s1 = new Student3();          s1.setAge(20);          s1.setName("ZhangSan");          s1.setTeacher(t);          Student3 s2 = (Student3) deepClone(s1);           s2 = (Student3) TTT.cloneTo(s1);          System.out.println("拷贝得到的信息:");          System.out.println(s2.getName());          System.out.println(s2.getAge());          System.out.println(s2.getTeacher().getName());          System.out.println(s2.getTeacher().getAge());          System.out.println("---------------------------");          // 将复制后的对象的老师信息修改一下：          s2.getTeacher().setName("New Teacher Wang");          s2.getTeacher().setAge(28);          System.out.println("修改了拷贝对象的教师后：");          System.out.println("拷贝对象的教师：");          System.out.println(s2.getTeacher().getName());          System.out.println(s2.getTeacher().getAge());          System.out.println("原来对象的教师：");          System.out.println(s1.getTeacher().getName());          System.out.println(s1.getTeacher().getAge());          // 由此证明序列化的方式实现了对象的深拷贝      }      public static Object deepClone(Object o) throws Exception      {          // 序列化          ByteArrayOutputStream bos = new ByteArrayOutputStream();          ObjectOutputStream oos = new ObjectOutputStream(bos);          oos.writeObject(o);          // 反序列化          ByteArrayInputStream bis = new ByteArrayInputStream(bos.toByteArray());          ObjectInputStream ois = new ObjectInputStream(bis);          return ois.readObject();      }  }  class Teacher3 implements Serializable  {      private String name;      private int age;      public String getName()      {          return name;      }      public void setName(String name)      {          this.name = name;      }      public int getAge()      {          return age;      }      public void setAge(int age)      {          this.age = age;      }  }  class Student3 implements Serializable  {      private String name;      private int age;      private Teacher3 teacher;      public String getName()      {          return name;      }      public void setName(String name)      {          this.name = name;      }      public int getAge()      {          return age;      }      public void setAge(int age)      {          this.age = age;      }      public Teacher3 getTeacher()      {          return teacher;      }      public void setTeacher(Teacher3 teacher)      {          this.teacher = teacher;      }      public <T> T deepClone(T src) throws RuntimeException, ClassNotFoundException, IOException      {          // 序列化          ByteArrayOutputStream bos = new ByteArrayOutputStream();          ObjectOutputStream oos = new ObjectOutputStream(bos);          oos.writeObject(src);          // 反序列化          ByteArrayInputStream bis = new ByteArrayInputStream(bos.toByteArray());          ObjectInputStream ois = new ObjectInputStream(bis);          return (T) ois.readObject();      }      public static <T> T deepClone2(T src) throws  RuntimeException{    ByteArrayOutputStream memoryBuffer = new ByteArrayOutputStream();    ObjectOutputStream out = null;    ObjectInputStream in = null;    T dist = null;    try {     out = new ObjectOutputStream(memoryBuffer);     out.writeObject(src);     out.flush();     in = new ObjectInputStream(new ByteArrayInputStream(       memoryBuffer.toByteArray()));     dist = (T) in.readObject();    } catch (Exception e) {     throw new RuntimeException(e);    } finally {     if (out != null)      try {       out.close();       out = null;      } catch (IOException e) {       throw new RuntimeException(e);      }     if (in != null)      try {       in.close();       in = null;      } catch (IOException e) {       throw new RuntimeException(e);      }    }    return dist;   }  } |

参考

1.[**如何理解java.io.Serializable接口**](http://segmentfault.com/q/1010000000713415)

<http://segmentfault.com/q/1010000000713415>

2.**java.io.Serializable浅析[举例]**

<http://www.cnblogs.com/gw811/archive/2012/10/10/2718331.html>

3.serialVersionUID是什么?

<http://zhidao.baidu.com/link?url=NbSCMIXx2H7arDbtKXCBICVX3sG8b72FmlVR0yF8sS98Mo5RS3_ZLMZZhdlMIfJrpGMcSqnR8sxhYqqMaWnY-a>

4.Java 深拷贝和浅拷贝

<http://www.cnblogs.com/mengdd/archive/2013/02/20/2917971.html>

# REF