### 什么是java序列化，如何实现java序列化？或者请解释Serializable接口的作用。

我们有时候将一个java对象变成字节流的形式传出去或者从一个字节流中恢复成一个java对象，例如，要将java对象存储到硬盘或者传送给网络上的其他计算机，这个过程我们可以自己写代码去把一个java对象变成某个格式的字节流再传输，但是，jre本身就提供了这种支持，我们可以调用OutputStream的writeObject方法来做，如果要让java 帮我们做，要被传输的对象必须实现serializable接口，这样，javac编译时就会进行特殊处理，编译的类才可以被writeObject方法操作，这就是所谓的序列化。需要被序列化的类必须实现Serializable接口，该接口是一个mini接口，其中没有需要实现的方法，implements Serializable只是为了标注该对象是可被序列化的。

例如，在web开发中，如果对象被保存在了Session中，tomcat在重启时要把Session对象序列化到硬盘，这个对象就必须实现Serializable接口。如果对象要经过分布式系统进行网络传输或通过rmi等远程调用，这就需要在网络上传输对象，被传输的对象就必须实现Serializable接口。

序列化概念的提出和实现，可以使我们的应用程序的设置信息保存和读取更加方便。

序列化有很多好处，比如，在一台机器上产生一个实例，初始化完毕，然后可以序列化，通过网络传送到另一台机器，然后反序列化，得到对象实例，之后再执行某些业务逻辑，得到结果，再序列化，返回第一台机器，第一台机器得到对象实例，得到结果。  
这个例子是目前比较先进的“智能代理”的原理。

当前比较热火的web services使用soap协议，soap协议也是以对象的可序列化为基础的。

### 序列化接口的id有什么用？

对象经常要通过IO进行传送，让你写程序传递对象，你会怎么做？把对象的状态数据用某种格式写入到硬盘，Person->“zxx,male,28,30000”🡪Person，既然大家都要这么干，并且没有个统一的干法，于是，sun公司就提出一种统一的解决方案，它会把对象变成某个格式进行输入和输出，这种格式对程序员来说是透明（transparent）的，但是，我们的某个类要想能被sun的这种方案处理，必须实现Serializable接口。

ObjectOutputStream.writeObject(obj);

Object obj = ObjectInputStream.readObject();

假设两年前我保存了某个类的一个对象，这两年来，我修改该类，删除了某个属性和增加了另外一个属性，两年后，我又去读取那个保存的对象，或有什么结果？未知！sun的jdk就会蒙了。为此，一个解决办法就是在类中增加版本后，每一次类的属性修改，都应该把版本号升级一下，这样，在读取时，比较存储对象时的版本号与当前类的版本号，如果不一致，则直接报版本号不同的错!

### [什么是java序列化，如何实现java序列化？](http://blog.csdn.net/yakihappy/article/details/3979373)

Java 串行化技术可以使你将一个对象的状态写入一个Byte 流里，并且可以从其它地方把该Byte 流里的数据读出来，重新构造一个相同的对象。这种机制允许你将对象通过网络进行传播，并可以随时把对象持久化到数据库、文件等系统里。Java的串行化机制是RMI、EJB等技术的技术基础。用途：利用对象的串行化实现保存应用程序的当前工作状态，下次再启动的时候将自动地恢复到上次执行的状态。

序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。序列化是为了解决在对对象流进行读写操作时所引发的问题。

序列化的实现：将需要被序列化的类实现Serializable接口，然后使用一个输出流(如：FileOutputStream)来构造一个ObjectOutputStream(对象流)对象，接着，使用ObjectOutputStream对象的writeObject(Object obj)方法就可以将参数为obj的对象写出(即保存其状态)，要恢复的话则用输入流。

2、串行化的特点：

    （1）如果某个类能够被串行化，其子类也可以被串行化。如果该类有父类，则分两种情况来考虑，如果该父类已经实现了可串行化接口。则其父类的相应字段及属性的处理和该类相同；如果该类的父类没有实现可串行化接口，则该类的父类所有的字段属性将不会串行化。

  （2）声明为static和transient类型的成员数据不能被串行化。因为static代表类的状态， transient代表对象的临时数据；

  （3）相关的类和接口：在java.io包中提供的涉及对象的串行化的类与接口有ObjectOutput接口、ObjectOutputStream类、ObjectInput接口、ObjectInputStream类。

    （1）ObjectOutput接口：它继承DataOutput接口并且支持对象的串行化，其内的writeObject()方法实现存储一个对象。ObjectInput接口：它继承DataInput接口并且支持对象的串行化，其内的readObject()方法实现读取一个对象。

    （2）ObjectOutputStream类：它继承OutputStream类并且实现ObjectOutput接口。利用该类来实现将对象存储（调用ObjectOutput接口中的writeObject()方法）。ObjectInputStream类：它继承InputStream类并且实现ObjectInput接口。利用该类来实现读取一个对象（调用ObjectInput接口中的readObject()方法）。

  对于父类的处理，如果父类没有实现串行化接口，则其必须有默认的构造函数（即没有参数的构造函数）。否则编译的时候就会报错。在反串行化的时候，默认构造函数会被调用。但是若把父类标记为可以串行化，则在反串行化的时候，其默认构造函数不会被调用。这是为什么呢？这是因为Java 对串行化的对象进行反串行化的时候，直接从流里获取其对象数据来生成一个对象实例，而不是通过其构造函数来完成。

import java.io.\*;

public class Cat implements Serializable {

        private String name;

        public Cat () {

                this.name = "new cat";

        }

        public String getName() {

                return this.name;

        }

        public void setName(String name) {

                this.name = name;

        }

        public static void main(String[] args) {

                Cat cat = new Cat();

                try {

                        FileOutputStream fos = new FileOutputStream("catDemo.out");

                        ObjectOutputStream oos = new ObjectOutputStream(fos);

                        System.out.println(" 1> " + cat.getName());

                        cat.setName("My Cat");

                        oos.writeObject(cat);

                        oos.close();

                } catch (Exception ex) {  ex.printStackTrace();   }

                try {

                        FileInputStream fis = new FileInputStream("catDemo.out");

                        ObjectInputStream ois = new ObjectInputStream(fis);

                        cat = (Cat) ois.readObject();

                        System.out.println(" 2> " + cat.getName());

                        ois.close();

                } catch (Exception ex) {

                        ex.printStackTrace();

                }

        }

}//writeObject和readObject本身就是线程安全的，传输过程中是不允许被并发访问的。所以对象能一个一个接连不断的传过来

### 注解transient是什么意思

java语言的关键字，变量修饰符，如果用transient声明一个[实例变量](http://zhidao.baidu.com/search?word=%E5%AE%9E%E4%BE%8B%E5%8F%98%E9%87%8F&fr=qb_search_exp&ie=utf8)，当对象存储时，它的值不需要维持。 Java的serialization提供了一种持久化对象实例的机制。当持久化对象时，可能有一个特殊的对象数据成员，我们不想用serialization机制来保存它。为了在一个特定对象的一个域上关闭serialization，可以在这个域前加上关键字transient。当一个对象被串行化的时候，transient型变量的值不包括在串行化的表示中，然而非transient型的变量是被包括进去的。

Hibernate持久化数据库时，用@Transient标记出某个字段，这个字段就不会参与对数据库表的操作，是一个辅助字段。

**public** **class** UseStudent {

**public** **static** **void** main(String[] args) {

Student st = **new** Student("Tom", 'M', 20, 3.6);

File file = **new** File("d:\\student.txt");

**try** {

file.createNewFile();

} **catch** (IOException e) {

e.printStackTrace();

}

**try** {

// Student对象序列化过程

FileOutputStream fos = **new** FileOutputStream(file);

ObjectOutputStream oos = **new** ObjectOutputStream(fos);

oos.writeObject(st);

oos.flush();

oos.close();

fos.close();

// Student对象反序列化过程

FileInputStream fis = **new** FileInputStream(file);

ObjectInputStream ois = **new** ObjectInputStream(fis);

Student st1 = (Student) ois.readObject();

System.***out***.println("name = " + st1.getName());

System.***out***.println("sex = " + st1.getSex());

System.***out***.println("year = " + st1.getYear());

System.***out***.println("gpa = " + st1.getGpa());

ois.close();

fis.close();

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**import** java.io.Serializable;

**public** **class** Student **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** String name;

**private** **char** sex;

**private** **int** year;

**private** **double** gpa;

**public** Student() {

}

**public** Student(String name, **char** sex, **int** year, **double** gpa) {

**this**.name = name;

**this**.sex = sex;

**this**.year = year;

**this**.gpa = gpa;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **void** setSex(**char** sex) {

**this**.sex = sex;

}

**public** **void** setYear(**int** year) {

**this**.year = year;

}

**public** **void** setGpa(**double** gpa) {

**this**.gpa = gpa;

}

**public** String getName() {

**return** **this**.name;

}

**public** **char** getSex() {

**return** **this**.sex;

}

**public** **int** getYear() {

**return** **this**.year;

}

**public** **double** getGpa() {

**return** **this**.gpa;

}

}

### Java通过网络传递对象（序列化,Socket编程）

**public** **class** Student **implements** Serializable {

**private** **static** **final** **long** *serialVersionUID* = 6687367257080711874L;

**private** **int** num;

**transient** **private** String name;

**public** Student(**int** num, String name) {

**this**.num = num;

**this**.name = name;

}

**public** **int** getNum() {

**return** num;

}

**public** **void** setNum(**int** num) {

**this**.num = num;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

@Override

**public** String toString() {

**return** ("num=" + num + ",name=" + name);

}

}

**public** **class** MyServer2 {

**public** **void** process() {

ServerSocket ss = **null**;

ObjectInputStream ois = **null**;

**try** {

ss = **new** ServerSocket(1111);

Socket socket = ss.accept();

ois = **new** ObjectInputStream(socket.getInputStream());

Student stu = (Student) ois.readObject();

System.*out*.println(stu);

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

**if** (ss != **null**) {

**try** {

ss.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

}

**public** **static** **void** main(String[] args) {

MyServer2 server = **new** MyServer2();

server.process();

}

}

**public** **class** MyClient2 {

**public** **void** process() {

Socket socket = **null**;

ObjectOutputStream oos = **null**;

**try** {

socket = **new** Socket("localhost", 1111);

oos = **new** ObjectOutputStream(socket.getOutputStream());

Student stu = **new** Student(1, "name1");

oos.writeObject(stu);

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

**if** (socket != **null**) {

**try** {

socket.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

**if** (oos != **null**) {

**try** {

oos.flush();

oos.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

}

**public** **static** **void** main(String[] args) {

MyClient2 client = **new** MyClient2();

client.process();

}

}