https://www.cnblogs.com/xiohao/p/4234184.html

有的时候我们想要把一个Java对象变成字节流的形式传出去---对象到字节流为序列化

，有的时候我们想要从一个字节流中恢复一个Java对象--字节流中恢复到对象为反序列化。

例如，有的时候我们想要把一个Java对象写入到硬盘或者传输到网路上面的其它计算机，这时我们就需要自己去通过java把相应的对象写成转换成字节流。对于这种通用的操作，我们为什么不使用统一的格式呢？没错，这里就出现了java的序列化的概念。

在Java的OutputStream类下面的子类ObjectOutputStream类就有对应的WriteObject(Object object) 其中要求对应的object实现了java的序列化的接口。

 Java对象的序列化有两种方式。

           a.是相应的对象实现了序列化接口Serializable，这个使用的比较多，对于序列化接口Serializable接口是一个空的接口，它的主要作用就是

             标识这个对象时可序列化的，jre对象在传输对象的时候会进行相关的封装

public class Article implements java.io.Serializable/\*表示可序列化 \*/{

    private static final long serialVersionUID = 1L;/\*当implements Serializable时，会有提示用固定的还是生成随机的\*/

……

}

　b.实现序列化的第二种方式为实现接口Externalizable，Externlizable的部分源代码如下:

public interface Externalizable extends java.io.Serializable {

    /\*\*

     \* The object implements the writeExternal method to save its contents

     \* by calling the methods of DataOutput for its primitive values or …… \*/

      没错，Externlizable接口继承了java的序列化接口，并增加了两个方法:

     void writeExternal(ObjectOutput out) throws IOException;

     void readExternal(ObjectInput in) throws IOException, ClassNotFoundException;

     首先，我们在序列化对象的时候，由于这个类实现了Externalizable 接口，在writeExternal()方法里定义了哪些属性可以序列化，哪些不可以序列化，所以，对象在经过这里就把规定能被序列化的序列化保存文件，不能序列化的不处理，然后在反序列的时候自动调用readExternal()方法，根据序列顺序挨个读取进行反序列，并自动封装成对象返回，然后在测试类接收，就完成了反序列。

所以说Exterinable的是Serializable的一个扩展.

class Person implements Externalizable{

        private static final long serialVersionUID = 1L;/\*没有强制提醒要设置\*/

……

/\*\*

     \* 序列化操作的扩展类

     \*/

    @Override

    public void writeExternal(ObjectOutput out) throws IOException {

        //增加一个新的对象

        Date date=new Date();

        out.writeObject(userName);

        out.writeObject(password);

        out.writeObject(date);

    }

    /\*\*

     \* 反序列化的扩展类

     \*/

    @Override

    public void readExternal(ObjectInput in) throws IOException,

            ClassNotFoundException {

        //注意这里的接受顺序是有限制的哦，否则的话会出错的

        // 例如上面先write的是A对象的话，那么下面先接受的也一定是A对象...

        userName=(String) in.readObject();

        password=(String) in.readObject();

        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");

        Date date=(Date)in.readObject();

        System.out.println("反序列化后的日期为:"+sdf.format(date));

    }

}

1.java中的序列化时transient变量(这个关键字的作用就是告知JAVA我不可以被序列化)和静态变量不会被序列化。（transient只作用于实现 Serializable 接口）

2.最应该注意的，如果你先序列化对象A后序列化B，那么在反序列化的时候一定记着JAVA规定先读到的对象是先被序列化的对象，不要先接收对象B，那样会报错.尤其在使用上面的Externalizable的时候一定要注意读取 的先后顺序。

                3.实现序列化接口的对象并不强制声明唯一的serialVersionUID，是否声明serialVersionUID对于对象序列化的向上向下的兼容性有很大的影响

@SerializedName("uid")//gson用，[**com**](eclipse-javadoc:☂=dbc-api/\/home\/liht\/.gradle\/caches\/modules-2\/files-2.1\/com.google.code.gson\/gson\/2.8.0\/c4ba5371a29ac9b2ad6129b1d39ea38750043eff\/gson-2.8.0.jar<com)**.**[**google**](eclipse-javadoc:☂=dbc-api/\/home\/liht\/.gradle\/caches\/modules-2\/files-2.1\/com.google.code.gson\/gson\/2.8.0\/c4ba5371a29ac9b2ad6129b1d39ea38750043eff\/gson-2.8.0.jar<com.google)**.**[**gson**](eclipse-javadoc:☂=dbc-api/\/home\/liht\/.gradle\/caches\/modules-2\/files-2.1\/com.google.code.gson\/gson\/2.8.0\/c4ba5371a29ac9b2ad6129b1d39ea38750043eff\/gson-2.8.0.jar<com.google.gson)**.**[**annotations**](eclipse-javadoc:☂=dbc-api/\/home\/liht\/.gradle\/caches\/modules-2\/files-2.1\/com.google.code.gson\/gson\/2.8.0\/c4ba5371a29ac9b2ad6129b1d39ea38750043eff\/gson-2.8.0.jar<com.google.gson.annotations)**.SerializedName\*/**

@JSONField(name = "uid")//fastjson用，[**com**](eclipse-javadoc:☂=dbc-api/\/home\/liht\/.gradle\/caches\/modules-2\/files-2.1\/com.alibaba\/fastjson\/1.2.47\/73d66d7ee99e81373b4955f2ee3797791520329b\/fastjson-1.2.47.jar<com)**.**[**alibaba**](eclipse-javadoc:☂=dbc-api/\/home\/liht\/.gradle\/caches\/modules-2\/files-2.1\/com.alibaba\/fastjson\/1.2.47\/73d66d7ee99e81373b4955f2ee3797791520329b\/fastjson-1.2.47.jar<com.alibaba)**.**[**fastjson**](eclipse-javadoc:☂=dbc-api/\/home\/liht\/.gradle\/caches\/modules-2\/files-2.1\/com.alibaba\/fastjson\/1.2.47\/73d66d7ee99e81373b4955f2ee3797791520329b\/fastjson-1.2.47.jar<com.alibaba.fastjson)**.**[**annotation**](eclipse-javadoc:☂=dbc-api/\/home\/liht\/.gradle\/caches\/modules-2\/files-2.1\/com.alibaba\/fastjson\/1.2.47\/73d66d7ee99e81373b4955f2ee3797791520329b\/fastjson-1.2.47.jar<com.alibaba.fastjson.annotation)**.JSONField \*/**

**protected** String userId;

<https://www.ibm.com/developerworks/cn/java/j-5things1/>

Java 序列化允许随着时间的推移而改变类型.

ObjectInputStream/ObjectOutputStream 类、完全保真的元数据以及程序员愿意用 Serializable 标识接口标记他们的类，从而 “参与” 这个过程。

写入文件

FileOutputStream fos = new FileOutputStream("tempdata.ser");

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeObject(ted);

oos.close()

从文件中反序列化

FileInputStream fis = new FileInputStream("tempdata.ser");

ObjectInputStream ois = new ObjectInputStream(fis);

com.tedneward.Person ted = (com.tedneward.Person) ois.readObject();

ois.close();

<https://my.oschina.net/kaixuan1995/blog/625069?p=1>

参数是按值而不是按引用传递的说明 Java 应用程序有且仅有的一种参数传递机制，即按值传递。

在 Java 应用程序中永远不会传递对象，而只传递对象引用。Java 应用程序按引用传递对象这一事实并不意味着 Java 应用程序按引用传递参数。参数可以是对象引用，而 Java 应用程序是按值传递对象引用的。（引用是什么？不就是地址吗？地址是什么，不过就是一个整数值）

按引用传递意味着当将一个参数传递给一个函数时，函数接收的是原始值的内存地址，而不是值的副本

# GPG密钥的生成与使用

https://www.jianshu.com/p/7f19ceacf57c