**什么是序列化，如何实现序列化？**

Java 中对象的序列化就是将对象转换成二进制序列，反序列化则是将二进制序列转换成对象。  
采用Java序列化与反序列化技术：

* 一是可以实现数据的持久化，在MVC模式中很是有用；
* 二是可以对象数据的远程通信。

Java 实现序列化的多种方式

1. 首先需要使用到工具类 ObjectInputStream 和ObjectOutputStream 两个IO类
2. 实现 Serializable 接口
3. 实现 Externalizable 接口

**一、什么是序列化与反序列化？**

Java 序列化是指把 Java 对象转换为字节序列的过程；  
Java 反序列化是指把字节序列恢复为 Java 对象的过程；

**二、为什么要用序列化与反序列化？**

在 **为什么要用序列化与反序列化** 之前我们先了解一下对象序列化的两种用途：

1. 把对象的字节序列永久地保存到硬盘上，通常存放在一个文件中；
2. 在网络上传送对象的字节序列。

我们可以想想如果没有序列化之前，又是怎样一种情景呢？

**举例：**

Web 服务器中的 Session 会话对象，当有10万用户并发访问，就有可能出现10万个 Session 对象，显然这种情况内存可能是吃不消的。

于是 Web 容器就会把一些 Session 先序列化，让他们离开内存空间，序列化到硬盘中，当需要调用时，再把保存在硬盘中的对象还原到内存中。

我们知道，当两个进程进行远程通信时，彼此可以发送各种类型的数据，包括文本、图片、音频、视频等， 而这些数据都会以二进制序列的形式在网络上传送。

同样的序列化与反序列化则实现了 **进程通信间的对象传送**，发送方需要把这个Java对象转换为字节序列，才能在网络上传送；接收方则需要把字节序列再恢复为Java对象。

初步总结：Java 序列化和反序列化，其一，实现了数据的持久化，通过序列化可以把数据永久的保存在硬盘上；其二，利用序列化实现远程通信，即在网络上传递对象的字节序列。

**三、如何实现序列化与反序列化？**

**3.1 JDK 类库中序列化 API**

使用到JDK中关键类 **ObjectOutputStream**(对象输出流) 和**ObjectInputStream**(对象输入流)

ObjectOutputStream 类中：通过使用 writeObject(Object object) 方法，将对象以二进制格式进行写入。

ObjectInputStream 类中：通过使用 readObject（）方法，从输入流中读取二进制流，转换成对象。

**3.2 目标对象实现 Serializable 接口**

我们创建一个 User 类，实现 Serializable 接口，并生成一个版本号 ：

**public** **class** **User** **implements** **Serializable** {  
  
    **private** **static** **final** **long** serialVersionUID = 3604972003323896788L;  
    **private** **transient** **int** age;  
    **private** String name;  
    **private** String sex;  
  
    **public** **int** **getAge**() {  
        **return** age;  
    }  
  
    **public** String **getName**() {  
        **return** name;  
    }  
  
    **public** String **getSex**() {  
        **return** sex;  
    }  
  
    **public** **void** **setAge**(**int** age) {  
        **this**.age = age;  
    }  
  
    **public** **void** **setName**(String name) {  
        **this**.name = name;  
    }  
  
    **public** **void** **setSex**(String sex) {  
        **this**.sex = sex;  
    }

首先：

1、Serializable 接口的作用只是用来标识我们这个类是需要进行序列化，并且 Serializable 接口中并没有提供任何方法。

2、SerialVersionUid 序列化版本号的作用是用来区分我们所编写的类的版本，用于判断反序列化时类的版本是否一直，如果不一致会出现版本不一致异常。

3、transient 关键字，主要用来忽略我们不希望进行序列化的变量

由于第一种形式太不常见，顾不再啰嗦演示，直接来看第二种实现 Serializable 接口的写入方式：

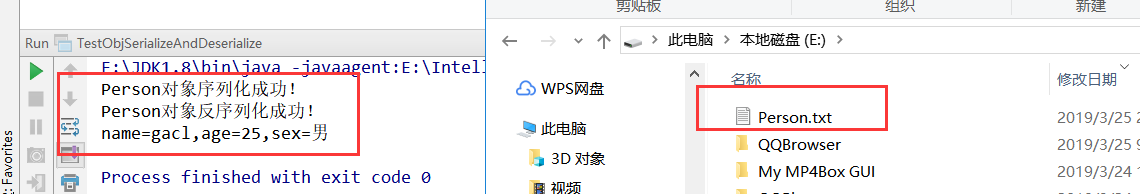
*定义一个Person类，实现Serializable接口*

**public** **class** **Person** **implements** **Serializable** {  
  
    **private** **static** **final** **long** serialVersionUID = -5809452578272945389L;  
    **private** **int** age;  
    **private** String name;  
    **private** String sex;  
  
    get..  
  
    set...  
}

*序列化和反序列化Person类对象*

**import** java.io.FileInputStream;  
**import** java.io.FileNotFoundException;  
**import** java.io.FileOutputStream;  
**import** java.io.IOException;  
**import** java.io.ObjectInputStream;  
**import** java.io.ObjectOutputStream;  
**import** java.text.MessageFormat;  
  
*/\*\*  
 \* <p>Description: 测试对象的序列化和反序列<p>  
 \*/*  
**public** **class** **TestObjSerializeAndDeserialize** {  
  
    **public** **static** **void** **main**(String[] args) **throws** Exception {  
　　　　 */\*\*序列化Person对象\*\*/*  
        SerializePerson();  
　　　　 */\*\*反序列Perons对象\*\*/*  
        Person p = DeserializePerson();  
        System.out.println(MessageFormat.format("name={0},age={1},sex={2}",p.getName(), p.getAge(), p.getSex()));  
    }  
  
    */\*\*  
     \* Description: 序列化Person对象  
     \*/*  
    **private** **static** **void** **SerializePerson**() **throws** FileNotFoundException,  
            IOException {  
        Person person = **new** Person();  
        person.setName("gacl");  
        person.setAge(25);  
        person.setSex("男");  
        */\*\* ObjectOutputStream 对象输出流，将Person对象存储到E盘的Person.txt文件中，完成对Person对象的序列化操作 \*\*/*  
        ObjectOutputStream oo = **new** ObjectOutputStream(**new** FileOutputStream(  
                **new** File("E:/Person.txt")));  
        oo.writeObject(person);  
        System.out.println("Person对象序列化成功！");  
        oo.close();  
    }  
  
    */\*\*  
     \* Description: 反序列Perons对象  
     \*/*  
    **private** **static** Person **DeserializePerson**() **throws** Exception, IOException {  
        ObjectInputStream ois = **new** ObjectInputStream(**new** FileInputStream(  
                **new** File("E:/Person.txt")));  
        Person person = (Person) ois.readObject();  
        System.out.println("Person对象反序列化成功！");  
        **return** person;  
    }  
  
}

*代码运行结果:*



疑问：*Person 实体中的 serialVersionUID 是个什么鬼？*

**答：**序列化版本号，取值是 Java 运行时环境根据类的内部细节自动生成的。如果对类的源代码作了修改，再重新编译，新生成的类文件的 serialVersionUID 的取值有可能也会发生变化。。

**既然程序帮我门自动生成，那为何还要去定义该属性？**

序列化和反序列化就是通过对比其 SerialversionUID 来进行的，我们修改一个实现 Serializable 接口的实体类，重新编译后，显然程序会重新会生成新值，那么一旦SerialversionUID 跟之前不匹配，反序列化就无法成功。

在实际的生产环境中，我们可能会建一系列的中间 Object 来反序列化我们的 pojo，为了解决这个问题，我们就需要在实体类中自定义 SerialversionUID，就像上方示例，不管我们序列化之后如何更改我们的 实体（不删除原有字段），最终都可以反序列化成功。。

