遇到这个 Java Serializable 序列化这个接口，我们可能会有如下的问题

* 什么叫序列化和反序列化；
* 作用，为啥要实现这个Serializable接口，也就是为啥要序列化；
* serialVersionUID 这个的值到底是在怎么设置的，有什么用。有的是1L，有的是一长串数字，迷惑ing。

在处理这个问题之前，你要先知道一个问题，这个比较重要。这个Serializable接口，以及相关的东西，全部都在Java io里面的。

### 1、序列化和反序列化的概念

* 序列化：把对象转换为字节序列的过程称为对象的序列化。
* 反序列化：把字节序列恢复为对象的过程称为对象的反序列化。

简而言之。在代码运行的时候，我们可以看到很多的对象(debug过的都造吧)，可以是一个，也可以是一类对象的集合，很多的对象数据，这些数据中，有些信息我们想让他持久的保存起来，那么这个就叫序列化。

就是把内存里面的这些对象给变成一连串的字节(bytes)描述的过程。常见的就是变成文件，我不序列化也可以保存文件啥的呀，有什么影响呢？我也是这么问的。

### 2、什么情况下需要序列化

* 当你想把的内存中的对象状态保存到一个文件中或者数据库中时候；
* 当你想用套接字在网络上传送对象的时候；
* 当你想通过RMI传输对象的时候；

(老实说，上面的几种，我可能就用过个存数据库的。)

### 3、Java如何实现序列化

实现Serializable接口即可。下面实际代码看看这个序列化到底能干啥，以及会产生的bug问题。

transient修饰的属性，是不会被序列化的。

静态static属性不会序列化。

serialVersionUID的作用和用法：

model里没有明确给serialVersionUID赋值，但是，Java会自动赋值，这个值跟这个model的属性相关计算出来的。

保存的时候，也就是序列化的时候，那时候还没有addTip属性呢，所以，自动生成的serialVersionUID 这个值，在我反序列化的时候Java自动生成的这个serialVersionUID值是不同的，他就抛异常啦。

注意：在实现这个Serializable 接口的时候，一定要给这个 serialVersionUID 赋值。可以简单的赋值个 1L，这就可以啦。。这样可以确保代码一致时反序列化成功。

下面是摘自 jdk api 文档里面关于接口 Serializable 的描述

* 类通过实现 java.io.Serializable 接口以启用其序列化功能。
* 未实现此接口的类将无法使其任何状态序列化或反序列化。
* 可序列化类的所有子类型本身都是可序列化的。因为实现接口也是间接的等同于继承。
* 序列化接口没有方法或字段，仅用于标识可序列化的语义。

静态属性不会被序列化，但是却又有一个特殊的静态属性，会被序列化，没办法，这个静态属性是亲生的。自带的。

序列化运行时使用一个称为 serialVersionUID 的版本号与每个可序列化类相关联，该序列号在反序列化过程中用于验证序列化对象的发送者和接收者是否为该对象加载了与序列化兼容的类。如果接收者加载的该对象的类的 serialVersionUID 与对应的发送者的类的版本号不同，则反序列化将会导致 InvalidClassException。可序列化类可以通过声明名为 "serialVersionUID" 的字段（该字段必须是静态 (static)、最终 (final) 的 long 型字段）显式声明其自己的 serialVersionUID：

如果可序列化类未显式声明 serialVersionUID，则序列化运行时将基于该类的各个方面计算该类的默认 serialVersionUID 值，如“Java(TM) 对象序列化规范”中所述。不过，强烈建议 所有可序列化类都显式声明 serialVersionUID 值，原因是计算默认的 serialVersionUID 对类的详细信息具有较高的敏感性，根据编译器实现的不同可能千差万别，这样在反序列化过程中可能会导致意外的 InvalidClassException。

因此，为保证 serialVersionUID 值跨不同 java 编译器实现的一致性，序列化类必须声明一个明确的 serialVersionUID 值。还强烈建议使用 private 修饰符显示声明 serialVersionUID（如果可能），原因是这种声明仅应用于直接声明类 -- serialVersionUID 字段作为继承成员没有用处。数组类不能声明一个明确的 serialVersionUID，因此它们总是具有默认的计算值，但是数组类没有匹配 serialVersionUID 值的要求。