序列化和反序列化作为Java里一个较为基础的知识点，大家心里也有那么几句要说的，但我相信很多小伙伴掌握的也就是那么几句而已，如果再深究问一下Java如何实现序列化和反序列化的，就可能不知所措了！遥记当年也被问了这一个问题，自信满满的说了一大堆，什么是序列化、什么是反序列化、什么场景的时候才会用到等，然后面试官说：那你能说一下序列化和反序列化底层是如何实现的吗？一脸懵逼，然后回家等通知！

**一、基本概念**

**1、什么是序列化和反序列化**

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

（2）序列化：对象序列化的最主要的用处就是在传递和保存对象的时候，保证对象的完整性和可传递性。序列化是把对象转换成有序字节流，以便在网络上传输或者保存在本地文件中。序列化后的字节流保存了Java对象的状态以及相关的描述信息。序列化机制的核心作用就是对象状态的保存与重建。

（3）反序列化：客户端从文件中或网络上获得序列化后的对象字节流后，根据字节流中所保存的对象状态及描述信息，通过反序列化重建对象。

（4）本质上讲，序列化就是把实体对象状态按照一定的格式写入到有序字节流，反序列化就是从有序字节流重建对象，恢复对象状态。

**2、为什么需要序列化与反序列化**

我们知道，当两个进程进行远程通信时，可以相互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。

那么当两个Java进程进行通信时，能否实现进程间的对象传送呢？答案是可以的！如何做到呢？这就需要Java序列化与反序列化了！

换句话说，一方面，发送方需要把这个Java对象转换为字节序列，然后在网络上传送；另一方面，接收方需要从字节序列中恢复出Java对象。

当我们明晰了为什么需要Java序列化和反序列化后，我们很自然地会想Java序列化的好处。其好处一是实现了数据的持久化，通过序列化可以把数据永久地保存到硬盘上（通常存放在文件里），二是，利用序列化实现远程通信，即在网络上传送对象的字节序列。

总的来说可以归结为以下几点：

（1）永久性保存对象，保存对象的字节序列到本地文件或者数据库中；

（2）通过序列化以字节流的形式使对象在网络中进行传递和接收；

（3）通过序列化在进程间传递对象；

**3、序列化算法一般会按步骤做如下事情：**

（1）将对象实例相关的类元数据输出。

（2）递归地输出类的超类描述直到不再有超类。

（3）类元数据完了以后，开始从最顶层的超类开始输出对象实例的实际数据值。

（4）从上至下递归输出实例的数据

**二、Java如何实现序列化和反序列化**

1、JDK类库中序列化和反序列化API

（1）java.io.ObjectOutputStream：表示对象输出流；

它的writeObject(Object obj)方法可以对参数指定的obj对象进行序列化，把得到的字节序列写到一个目标输出流中；

（2）java.io.ObjectInputStream：表示对象输入流；

它的readObject()方法源输入流中读取字节序列，再把它们反序列化成为一个对象，并将其返回；

**2、实现序列化的要求**

只有实现了Serializable或Externalizable接口的类的对象才能被序列化，否则抛出异常！

**3、实现Java对象序列化与反序列化的方法**

假定一个User类，它的对象需要序列化，可以有如下三种方法：

（1）若User类仅仅实现了Serializable接口，则可以按照以下方式进行序列化和反序列化

ObjectOutputStream采用默认的序列化方式，对User对象的非transient的实例变量进行序列化。

ObjcetInputStream采用默认的反序列化方式，对对User对象的非transient的实例变量进行反序列化。

（2）若User类仅仅实现了Serializable接口，并且还定义了readObject(ObjectInputStream in)和writeObject(ObjectOutputSteam out)，则采用以下方式进行序列化与反序列化。

ObjectOutputStream调用User对象的writeObject(ObjectOutputStream out)的方法进行序列化。

ObjectInputStream会调用User对象的readObject(ObjectInputStream in)的方法进行反序列化。

（3）若User类实现了Externalnalizable接口，且User类必须实现readExternal(ObjectInput in)和writeExternal(ObjectOutput out)方法，则按照以下方式进行序列化与反序列化。

ObjectOutputStream调用User对象的writeExternal(ObjectOutput out))的方法进行序列化。

ObjectInputStream会调用User对象的readExternal(ObjectInput in)的方法进行反序列化。

**4、JDK类库中序列化的步骤**

步骤一：创建一个对象输出流，它可以包装一个其它类型的目标输出流，如文件输出流：



步骤二：通过对象输出流的writeObject()方法写对象：



**5、JDK类库中反序列化的步骤**

步骤一：创建一个对象输入流，它可以包装一个其它类型输入流，如文件输入流：



步骤二：通过对象输出流的readObject()方法读取对象：



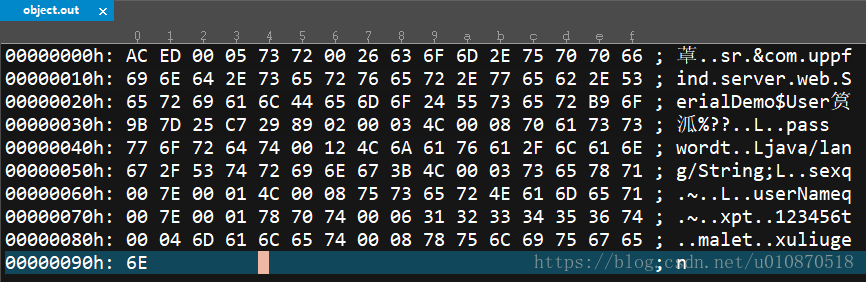
说明：为了正确读取数据，完成反序列化，必须保证向对象输出流写对象的顺序与从对象输入流中读对象的顺序一致。

**6、序列化和反序列化的示例**

为了更好地理解Java序列化与反序列化，举一个简单的示例如下：



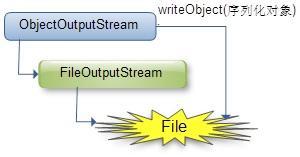
object.out文件如下（使用UltraEdit打开）：



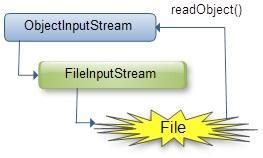
注：上图中0000000h-000000c0h表示行号；0-f表示列；行后面的文字表示对这行16进制的解释；对上述字节码所表述的内容感兴趣的可以对照相关的资料，查阅一下每一个字符代表的含义，这里不在探讨！

类似于我们Java代码编译之后的.class文件，每一个字符都代表一定的含义。序列化和反序列化的过程就是生成和解析上述字符的过程！

序列化图示：



反序列化图示：



**三、相关注意事项**

1、序列化时，只对对象的状态进行保存，而不管对象的方法；

2、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；

3、当一个对象的实例变量引用其他对象，序列化该对象时也把引用对象进行序列化；

4、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：

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；

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；

5、声明为static和transient类型的成员数据不能被序列化。因为static代表类的状态，transient代表对象的临时数据。

6、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。为它赋予明确的值。显式地定义serialVersionUID有两种用途：

在某些场合，希望类的不同版本对序列化兼容，因此需要确保类的不同版本具有相同的serialVersionUID；

在某些场合，不希望类的不同版本对序列化兼容，因此需要确保类的不同版本具有不同的serialVersionUID。

7、Java有很多基础类已经实现了serializable接口，比如String,Vector等。但是也有一些没有实现serializable接口的；

8、如果一个对象的成员变量是一个对象，那么这个对象的数据成员也会被保存！这是能用序列化解决深拷贝的重要原因；

**四、总结**

看到这里，可能已经让我们很满足了，毕竟已经知道了我们平时使用的序列化和反序列化是如何进行操作的，Java给我们提供了哪些接口可供使用，也比我们最初知道的简单的什么是序列化、反序列化以及作用多了很多！

看了这篇文章之后，我们应该清楚的知道如下问题了：

* 什么是序列化和反序列化；
* 为什么要进行序列化和反序列化？
* 序列化和反序列化的作用和应用场景有哪些？
* 序列化和反序列化中用到的ObjectOutputStream和ObjectInputStream，以及他们的readObject和writeObject方法，这也是底层API的具体实现，一般问到原理的时候，我们只需要回到出来这一个层面就可以了；
* 脑海中应该有一个印象就是，序列成的字节码和反序列化解析的字节码基本构造是什么样子的；
* 实现序列化和反序列化的要求，什么时候数据不会被序列化和反序列化？