介绍Java对象序列化使用基础Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/620/2021_2022__E4_BB_8B_E7_BB_8DJava_c104_620992.htm 序列化的过程就是对象写入字节流和从字节流中读取对象。将对象状态转换成字节流之后，可以用java.io包中的各种字节流类将其保存到文件中，管道到另一线程中或通过网络连接将对象数据发送到另一主机。对象序列化功能非常简单、强大，在RMI、Socket、JMS、EJB都有应用。对象序列化问题在网络编程中并不是最激动人心的课题，但却相当重要，具有许多实用意义。 1.对象序列化可以实现分布式对象。主要应用例如：RMI要利用对象序列化运行远程主机上的服务，就像在本地机上运行对象时一样。 2.Java对象序列化不仅保留一个对象的数据，而且递归保存对象引用的每个对象的数据。可以将整个对象层次写入字节流中，可以保存在文件中或在网络连接上传递。利用对象序列化可以进行对象的“深复制”，即复制对象本身及引用的对象本身。序列化一个对象可能得到整个对象序列。从上面的叙述中，我们知道了对象序列化是java编程中的必备武器，那么让我们从基础开始，好好学习一下它的机制和用法。 Java序列化比较简单，通常不需要编写保存和恢复对象状态的定制代码。只需要实现接口(java.io.Serializable)的类对象可以转换成字节流或从字节流恢复，不需要在类中增加任何代码。只有极少数情况下才需要定制代码保存或恢复对象状态。这里要注意：不是每个类都可序列化，有些类是不能序列化的，例如涉及线程的类与特定JVM有非常复杂的关系。 序列化机制： 序列化分为两大部分：序列化和反序列化。序列化

是这个过程的第一部分，将数据分解成字节流，以便存储在文件中或在网络上传输。反序列化就是打开字节流并重构对象。对象序列化不仅要将基本数据类型转换成字节表示，有时还要恢复数据。恢复数据要求有恢复数据的对象实例。ObjectOutputStream中的序列化过程与字节流连接，包括对象类型和版本信息。反序列化时，JVM用头信息生成对象实例，然后将对象字节流中的数据复制到对象数据成员中。下面我们分两大部分来阐述： 处理对象流：(序列化过程和反序列化过程) java.io包有两个序列化对象的类。ObjectOutputStream负责将对象写入字节流，ObjectInputStream从字节流重构对象。 我们先了解ObjectOutputStream类吧。ObjectOutputStream类扩展DataOutput接口。 writeObject()方法是最重要的方法，用于对象序列化。如果对象包含其他对象的引用，则writeObject()方法递归序列化这些对象。每个ObjectOutputStream维护序列化的对象引用表，防止发送同一对象的多个拷贝。(这点很重要)由于writeObject() 可以序列化整组交叉引用的对象，因此同一ObjectOutputStream实例可能不小心被请求序列化同一对象。这时，进行反引用序列化，而不是再次写入对象字节流。 下面，让我们从例子中来了解ObjectOutputStream这个类吧。 // 序列化 today ’ s date 到一个文件中. FileOutputStream f = new FileOutputStream("tmp"). ObjectOutputStream s = new ObjectOutputStream(f). s.writeObject("Today"). s.writeObject(new Date()). s.flush(). 现在，让我们来了解ObjectInputStream这个类。它与ObjectOutputStream相似。它扩展DataInput接口。ObjectInputStream中的方法镜像DataInputStream中读取Java

基本数据类型的公开方法。readObject()方法从字节流中反序列化对象。每次调用readObject()方法都返回流中下一个Object。对象字节流并不传输类的字节码，而是包括类名及其签名。 readObject()收到对象时，JVM装入头中指定的类。如果找不到这个类，则readObject()抛出ClassNotFoundException,如果需要传输对象数据和字节码，则可以用RMI框架。ObjectInputStream的其余方法用于定制反序列化过程。 例子如下： //从文件中反序列化 string 对象和 date 对象 FileInputStream in = new FileInputStream("tmp"). ObjectInputStream s = new ObjectInputStream(in). String today = (String)s.readObject(). Date date = (Date)s.readObject(). 定制序列化过程: 序列化通常可以自动完成，但有时可能要对这个过程进行控制。java可以将类声明为serializable，但仍可手工控制声明为static或transient的数据成员。 例子：一个非常简单的序列化类。 public class simpleSerializableClass implements Serializable { String sToday="Today:". transient Date dtToday=new Date(). } 序列化时，类的所有数据成员应可序列化除了声明为transient或static的成员。将变量声明为transient告诉JVM我们会负责将变元序列化。将数据成员声明为transient后，序列化过程就无法将其加进对象字节流中，没有从transient数据成员发送的数据。后面数据反序列化时，要重建数据成员(因为它是类定义的一部分)，但不包含任何数据，因为这个数据成员不向流中写入任何数据。记住，对象流不序列化static或transient。我们的类要用writeObject()与readObject()方法以处理这些数据成员。使用writeObject()与 readObject()方法时，还要注意按写入的顺序读取这些数据成员。 关于如何使用定制

序列化的部分代码如下： //重写writeObject()方法以便处理transient的成员。 public void writeObject(ObjectOutputStream outputStream) throws IOException { outputStream.defaultWriteObject(). //使定制的writeObject()方法可以利用自动序列化中内置的逻辑。 outputStream.writeObject(oSocket.getInetAddress()). outputStream.writeInt(oSocket.getPort()). } //重写readObject()方法以便接收transient的成员。 private void readObject(ObjectInputStream inputStream) throws IOException,ClassNotFoundException { inputStream.defaultReadObject().//defaultReadObject()补充自动序列化 InetAddress oAddress=(InetAddress)inputStream.readObject(). int iPort =inputStream.readInt(). oSocket = new Socket(oAddress,iPort). iID=getID(). dtToday =new Date(). } 完全定制序列化过程: 如果一个类要完全负责自己的序列化，则实现Externalizable接口而不是Serializable接口。 Externalizable接口定义包括两个方法writeExternal()与readExternal()。利用这些方法可以控制对象数据成员如何写入字节流.类实现Externalizable时，头写入对象流中，然后类完全负责序列化和恢复数据成员，除了头以外，根本没有自动序列化。这里要注意了。声明类实现Externalizable接口会有重大的安全风险。writeExternal()与readExternal()方法声明为public，恶意类可以用这些方法读取和写入对象数据。如果对象包含敏感信息，则要格外小心。这包括使用安全套接或加密整个字节流。 处理对象流：(序列化过程和反序列化过程) java.io包有两个序列化对象的类

。ObjectOutputStream负责将对象写入字节流，ObjectInputStream从字节流重构对象。 我们先了解ObjectOutputStream类吧。ObjectOutputStream类扩展DataOutput接口。 writeObject()方法是最重要的方法，用于对象序列化。如果对象包含其他对象的引用，则writeObject()方法递归序列化这些对象。每个ObjectOutputStream维护序列化的对象引用表，防止发送同一对象的多个拷贝。(这点很重要)由于writeObject() 可以序列化整组交叉引用的对象，因此同一ObjectOutputStream实例可能不小心被请求序列化同一对象。这时，进行反引用序列化，而不是再次写入对象字节流。 下面，让我们从例子中来了解ObjectOutputStream这个类吧。 // 序列化 today ’ s date 到一个文件中. FileOutputStream f = new FileOutputStream("tmp"). ObjectOutputStream s = new ObjectOutputStream(f). s.writeObject("Today"). s.writeObject(new Date()). s.flush(). 现在，让我们来了解ObjectInputStream这个类。它与ObjectOutputStream相似。它扩展DataInput接口。ObjectInputStream中的方法镜像DataInputStream中读取Java基本数据类型的公开方法。readObject()方法从字节流中反序列化对象。每次调用readObject()方法都返回流中下一个Object。对象字节流并不传输类的字节码，而是包括类名及其签名。 readObject()收到对象时，JVM装入头中指定的类。如果找不到这个类，则readObject()抛出ClassNotFoundException,如果需要传输对象数据和字节码，则可以用RMI框架。ObjectInputStream的其余方法用于定制反序列化过程。 