1. 对象序列化
2. 概述

对于存储相同类型的数据个固定长度的记录格式是高效的，但是对于存储对象类型，无法确定该对象所需要的数据格式，因此我们使用对象序列化的机制将任何对象写出到输出流之中，并且可以将其读回。

1. 保存和加载序列化对象

为了保存对象数据，首先要打开一个ObjectOutputStream对象，通过writeObject方法来保存对象到输出流中，ObjectOutputStream类浏览对象的所有域，并存储它们的内容，如下所示：

var out = new ObjectOutputStream(new FileOutputStream(“a.dat”));

var harry = new Employe(“harry”, 1);

out.writeObject(harry);

同时要修改的是对希望在对象输出流中存储或者恢复的类必须实现Serializable接口，该接口没有任何方法，只是一个标记接口，标记这个类可以序列化：

class Employee implements Serializable{}

每个对象都是用一个序列号保存的，这个序列号是唯一的，当第一次遇到该对象时，保存该对象到输出流中，如果该对象之前已经被保存过，则直接引用该对象的序列号即可。序列号另一种应用就是通过网络将对象集合传送到另一台计算机中。

1. 序列化的文件格式

对象序列化是以特殊的文件格式存储对象数据的，过于复杂，只要知道是将对象的类，超类，接口，域类型和方法签名按照规范方式排序，然后将安全散列（SHA）算法应用于这些数据获得的指纹，即序列号，序列化的版本的唯一ID。

1. 修改默认的序列化机制

某些数据域是不能被序列化的，如只在本地方法有意义的对象信息，Java中为了防止这种域被序列化，可以将它们标记为transient，如果这些域属于不可序列化的类，也要将它们标记为transient的。

同时，单个类还可以修改默认的序列化机制，自定义readObject和writeObject方法，取代默认的序列化方法。

1. 序列化单例和类型安全的枚举

对于单例和类型安全的枚举，默认的序列化机制是不适用的，它会创建一个新的对象与之前已存在的对象不一致，为了解决这个问题，可以定义另一种称为readResolve的特殊序列化方法，返回恰当的对象。

1. 版本管理

使用序列化来保存对象，对象可能发生变化，它的 SHA指纹也会发生变化，而对象输入流将拒绝读入不同指纹的对象。但是类可以表明它对早期版本是兼容的，要实现这样的效果，就要首先获得这个类的早期版本的指纹，我们可以使用JDK中的程序serialver来获得这个数字，如下所示：

serialver Employee

将会打印出：

Employee: static final long serialVersionUID = -1814239825517340645L;

这个类的所有较新的版本都必须把serialVersionUID常量定义为与最初版本的指纹相同，这样，就可以兼容早期的版本了，可以读入这个类的不同版本，如下所示：

class Employee implements Serializable{

public static final long serialVersionUID = -1814239825517340645L;

}

只是读入该对象的版本并不代表什么，如果该对象只是方法发生了变化，则读入新对象数据时不会有问题，如果数据发生了变化，就会有问题。对象输入流会将这个类当前版本的数据域与被序列化的版本中的数据域进行比较，只考虑非瞬时（transient）和非静态（static）的数据域，如果这两部分数据域之间名字匹配而类型不匹配，则对象输入流不会尝试将一种类型转换为另一种类型，因为两个对象不兼容；如果有当前版本对象所没有的数据域，则忽略这些额外的数据；如果当前版本对象新增了一些数据，则设置为默认值（对象设置为null , 数字设置为0，boolean设置为false）。