Java IO中常用类：

File：文件类，文件的基本操作都是此类，例如生成新目录，修改文件名，删除文件，判断文件所在路径等。

RandomAccessFile：随机存取文件类，file对文件操作，RandomAccessFile对文件内容操作

InputStream/OutputStream：字节输入/输出流

Reader/Writer：字符输入/输出流

字节流/字符流

Java采用Unicode编码，一个字符占用两个字节

字节流每次输出/输入是按字节操作，字符流每次输出/输入是按字符操作

字节流在操作是本身不会用缓冲区，对文件本身直接操作，而字符流在操作时使用了缓存区，通过缓存区再操作文件，实际开发中字节流使用的更多

字节流和字符流的转换：

InputStreamReader(InputStream)：字节到字符的桥梁

OutputStreamWriter(OutStreamout)：字符流到字节的桥梁

System类对IO的支持：

System.in：代表键盘的输入，通过Scanner类接收输入

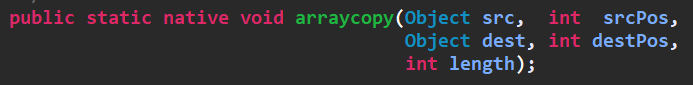
System.out：向控制台输出

System.error：向控制台输出错误信息

System类：在System类下有系统级的属性和控制方法(都是static，可以直接使用)

成员变量：in，out，error

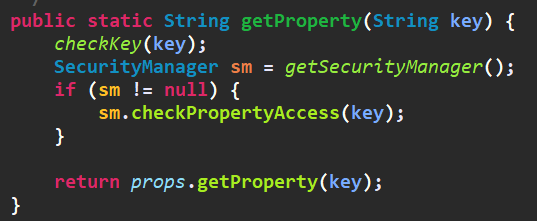
成员方法：



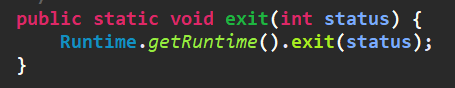
native调用c/c++进行对数组的复制



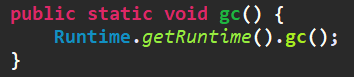
获取当前计算机的时间，与格林威治时间(1970.1.1,0:0:0)所相差的毫秒数

获取系统的属性对应的值



退出程序，status为0表示正常退出，非0为异常退出



手动调用gc

Runtime类封装了java虚拟机的启动进程，每个java虚拟机对应一个Runtime实例

序列化/反序列化：

序列化是将对象转换为字节序列的过程，这样java对象就可以在网络中传输，对象就可以在线程之间通信，当关闭JVM的时候，可以见对象序列化到外存中，启动JVM时，在反序列化到JVM内存(二进制的byte流)

实现：

ObjectOutputStream中的writeObject(Object obj)方法可以将对象以二进制格式写入

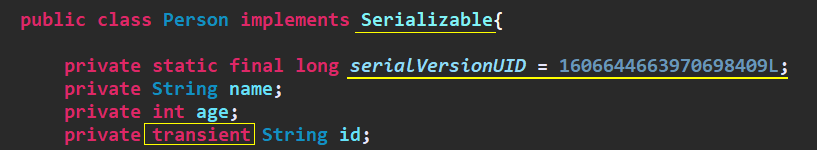
ObjectInputStream中的readObject()方法可以从输入流中读取二进制，转换为对象

被序列化的对象必须实现Serializable接口，这样此对象才能被序列化

private transient String id；transient表示此属性在序列化的时候不被序列化/static关键不会被序列化

static为什么不能序列化：序列化是对象，而不是类，static的属性属于类，如果static可以被序列化，那么当前对象序列化后，其他对象对static进行了修改，那么反序列出来的对象和类中的static属性就不一样了，可以添加final，final static常量，不可变，所以可以序列化(可能在输出的时候，会发现static的属性被序列化了，其实是没有的，在通过对象.static属性名获取static属性时，获取到的类的属性，而不是当前对象的属性，所以static属性是不能被序列化的)

serialVersionUID：在反序列化的时候，会判断反序列化出来的对象是否是目标类的实例化对象

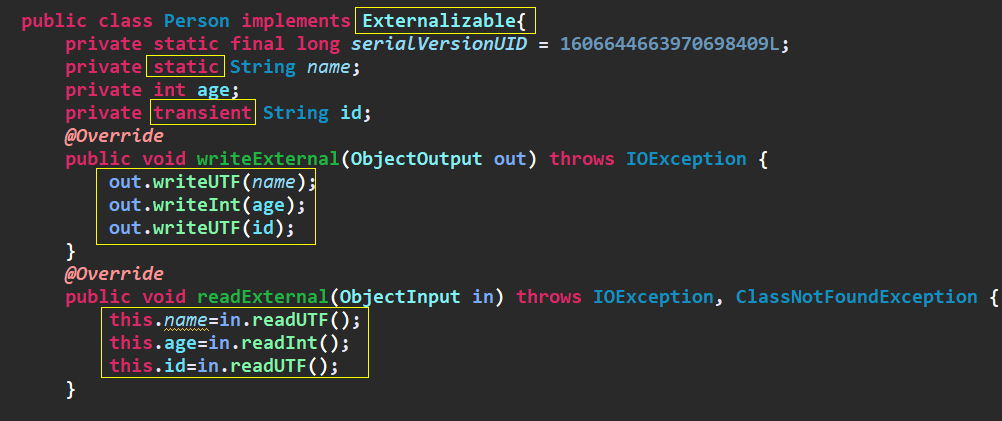


实现方式



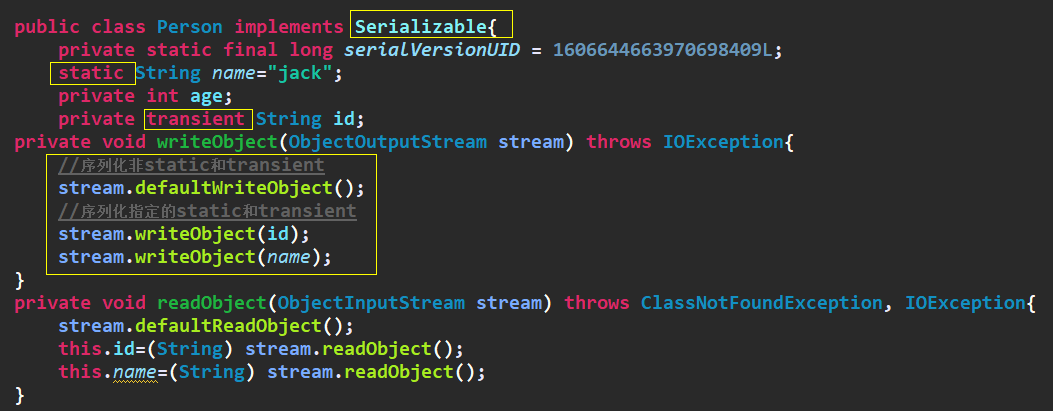
Externalizable：Externalizable是Serializable的子接口，在接口中声明了writeExternal和readExternal方法，通过此方法可以指定需要序列化的属性，并且还可以序列化static和transient属性

Externalizable接口只支持在writeExternal和readExternal方法中进行序列化



自定义序列化规则

Serializable接口也可以指定需要序列化的属性(包含static，transient)，但是要严格遵循方法签名writeObject和readObject



在执行ObjectOutputStream的writeObject(Object obj)时，首先会去判断方法中是否实现了write/readObject方法，如果实现了则使用类自定义的序列化方法，没有则使用ObjectOutputStream默认的

ObjectOutputStream是ObjectOutput的子类

其实在方法中声明write/readObject和实现Externalizable接口类似，只是实现接口更规范

在反序列化的时候一定要保证序列化ID一致，有两种生成策略：固定的1L，随机不重复的long类型数据，一般采用固定的1L，这样可以确保反序列化成功，随机的ID可以用来限制某些用户的使用

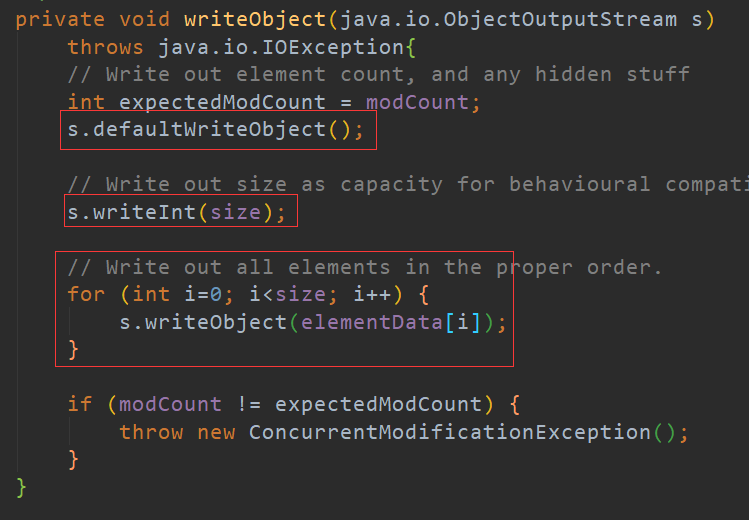
当一个对象的实例变量引用其他对象，序列化该对象时也把引用对象进行序列化；

序列化一个对象时，不会序列化其父对象，除非其父类也实现Serialiable接口，当反序列化出来的对象在获取其父类的属性时(父类没有实现Serialiable接口)，获取到的都是默认类型(0，null)(序列化的时候不会序列化继承的属性)

父类序列化的时候，子类会自动序列化，不用实现Serializable接口(这句话的意思应该是子类不用再实现Serializable接口，就可序列化)

在序列化过程中，虚拟机会试图调用对象类的writeOnObject和readObject方法(序列化方式3)，如果没有write/readObject方法就通过序列化方式1对对象进行序列化，用户可以通过writeObject自定义需要序列化的属性，并且可以对属性进行加密等操作

既然通过Externalizable和Serializable内的write/readObject方法可以序列化transient关键字的属性，那transient有什么作用，比如ArrayList自定义了write/readObject方法进行序列化，但是elementData还是被transient关键字修饰，如果elementData大小为100，但是只有第一个有元素，其他为空，如果直接使用defaultWriteObject()会将elementData中的null也序列化了，所以使用transient，当defaultWriteObject()只序列化非static和transient的属性，transient的属性通过writeObject(elementData[i]循环序列化，只序列化有元素



可以通过Jackson，Gson(google)，FastJson(阿里)等第三方工具类，将对象转化为JOSN格式，然后在将JSON进行序列化，这样可以减小序列化后的文件大小，通过JDK的序列化出来的对象占用字节大，但是只能序列化当前类，不能序列化父类和其子类

序列化时只对对象的状态进行保存，不管方法

序列化父类时，子类自动序列化，不需要显式实现Serializable接口

属性是引用类型也要实例化(深拷贝)