# java序列化机制Serialize接口

## java本身的序列化机制存在的问题

1. 序列化数据结果比较大、传输效率比较低
2. 不能跨语言对接

以至于在后来的很长一段时间，基于XML格式编码的对象序列化机制成为了主流，一方面解决了多语言兼容问题，另一方面比二进制的序列化方式更容易理解。以至于基于XML的SOAP协议及对应的WebService框架在很长一段时间内成为各个主流开发语言的必备的技术。

再到后来，基于JSON的简单文本格式编码的HTTP REST接口又基本上取代了复杂的Web Service接口，成为分布式架构中远程通信的首要选择。但是JSON序列化存储占用的空间大、性能低等问题，同时移动客户端应用需要更高效的传输数据来提升用户体验。在这种情况下与语言无关并且搞笑的二进制编码协议就成为了大家追求的热点技术之一。首先诞生的一个开源的二进制序列化框架-MessagePack。它比google的Protocol Buffers出现得还要早

恰当的序列化协议不仅可以提高系统的通用性、强壮型、安全性、优化性能。同时还能让系统更加易于调试和扩展

## 序列化和反序列化的概念

把对象转化为字节序列的过程称之为对象的序列化

反之，称之为反序列化

# 怎么去实现一个序列化操作

1. 实现Serializable接口
2. ObjectInputStream : 表示读取指定的字节数据转换成对象
3. ObjectOutputStream ：

# 科普小知识

## serialVersionUID的作用

文件流中的class和classpath中的class，也就是修改过后的class，不兼容了，处于安全机制考虑，程序抛出了错误，并且拒绝载入。从错误结果来看，如果没有为指定的class配置serialVersionUID，那么java编译器会自动给这个class进行一个摘要算法，类似于指纹算法，只要这个文件有任何改动，得到的UID就会截然不同的，可以保证在这么多类中，这个编号是唯一的。所以，由于没有显指定 serialVersionUID，编译器又为我们生成了一个UID，当然和前面保存在文件中的那个不会一样了，于是就出现了2个序列化版本号不一致的错误。因此，只要我们自己指定了serialVersionUID，就可以在序列化后，去添加一个字段，或者方法，而不会影响到后期的还原，还原后的对象照样可以使用，而且还多了方法或者属性可以用。

## 静态变量的序列化

序列化并不保存静态变量的状态

## Transient关键字

transient关键字表示指定属性不参与序列化

## 父子类问题

如果父类没有实现序列化，而子类实现列序列化。那么父类中的成员没办法做序列化操作

## 序列化的存储规则

对同一个对象进行多次写入，打印出的第一次存储结果和第二次存储结果，只多了5个字节的引用关系。

并不会导致文件累加

# 序列化实现深度克隆

**浅拷贝（浅复制、浅克隆）**：被复制对象的所有变量都含有与原来的对象相同的值，而所有的对其他对象的引用仍然指向原来的对象。

　　换言之，浅拷贝仅仅复制所拷贝的对象，而不复制它所引用的对象。

**深拷贝（深复制、深克隆）**：被复制对象的所有变量都含有与原来的对象相同的值，除去那些引用其他对象的变量。

　　那些引用其他对象的变量将指向被复制过的新对象，而不再是原有的那些被引用的对象。

　　换言之，深拷贝把要复制的对象所引用的对象都复制了一遍

# 总结

1. 在java中，只要一个类实现了java.io.Serializable接口，那么它就可以被序列化
2. 通过ObjectOutputStream和ObjectInputStream对对象进行序列化合反序列化操作
3. 对象是否允许被反序列化，不仅仅是取决于对象的代码是否一致，同时还有一个重要的因素（UID）
4. 序列化不保存静态变量
5. 要想父类对象也参与序列化操作，那么必须要让父类也实现Serializable接口
6. Transient关键字，主要是控制变量是否能够被序列化。如果没有被序列化的成员变量反序列化后，会被设置成初始值，比如String -> null
7. 通过序列化操作实现深度克隆

# 主流的序列化技术有哪些

JSON/Hessian(2) /xml/protobuf/kryo/MsgPack/FST/thrift/protostuff/Avro