### 1、序列化和反序列化

* 列化：把对象转换为字节序列的过程称为对象的序列化。
* 反序列化：把字节序列恢复为对象的过程称为对象的反序列化。

### 2、什么时候需要用到序列化和反序列化

当我们只在本地JVM里运行下Java实例，这个时候是不需要什么序列化和反序列化的，但当我们需要将内存中的对象持久化到磁盘，数据库中时，当我们需要与浏览器进行交互时，当我们需要实现RPC时，这个时候就需要序列化和反序列化了。

前两个需要用到序列化和反序列化的场景, 是不是让我们有一个很大的疑问? 我们在与浏览器交互时, 还有将内存中的对象持久化到数据库中时, 好像都没有去进行序列化和反序列化, 因为我们都没有实现Serializable接口, 但一直正常运行。

下面先给出结论:

只要我们对内存中的对象进行持久化或网络传输，这个时候都需要序列化和反序列化。

理由：

服务器与浏览器交互时真的没有用到Serializable接口吗？JSON格式实际上就是将一个对象转化为字符串，所以服务器与浏览器交互时的数据格式其实是字符串，我们来看来String类型的源码：

String类型实现了Serializable接口，并显示指定serialVersionUID的值。

然后我们再来看对象持久化到数据库中时的情况，Mybatis数据库映射文件里的insert代码：

<insert id="insertUser" parameterType="org.tyshawn.bean.User">

INSERT INTO t\_user(name, age) VALUES (#{name}, #{age})

</insert>

实际上我们并不是将整个对象持久化到数据库中，而是将对象中的属性持久化到数据库中，而这些属性都是实现了Serializable接口的基本属性。

### 3、实现序列化和反序列化为什么要实现Serializable接口

在Java中实现了Serializable接口后，JVM会在底层帮我们实现序列化和反序列化，如果我们不实现Serializable接口，那自己去写一套序列化和反序列化代码也行。

### 4、实现Serializable接口后，为什么还要显示指定serialVersionUID的值

如果不显示指定serialVersionUID, JVM在序列化时会根据属性自动生成一个serialVersionUID, 然后与属性一起序列化, 再进行持久化或网络传输. 在反序列化时, JVM会再根据属性自动生成一个新版serialVersionUID, 然后将这个新版serialVersionUID与序列化时生成的旧版serialVersionUID进行比较, 如果相同则反序列化成功，否则报错。

如果显示指定了serialVersionUID, JVM在序列化和反序列化时仍然都会生成一个serialVersionUID, 但值为我们显示指定的值, 这样在反序列化时新旧版本的serialVersionUID就一致了。

在实际开发中, 不显示指定serialVersionUID的情况会导致什么问题？如果我们的类写完后不再修改，那当然不会有问题，但这在实际开发中是不可能的，我们的类会不断迭代，一旦类被修改了，那旧对象反序列化就会报错。所以在实际开发中，我们都会显示指定一个serialVersionUID，值是多少无所谓，只要不变就行。

### 5、序列化的其他特性

被transient关键字修饰的属性不会被序列化，static属性也不会被序列化。

### 6、static属性为什么不会被序列化

因为序列化是针对对象而言的，而static属性优先于对象存在，随着类的加载而加载，所以不会被序列化。

看到这个结论，是不是有人会问，serialVersionUID也被static修饰，为什么serialVersionUID会被序列化？其实serialVersionUID属性并没有被序列化，JVM在序列化对象时会自动生成一个serialVersionUID，然后将我们显示指定的serialVersionUID属性值赋给自动生成的serialVersionUID。