第11章 序列化

2018年7月23日

10:47

**第74条：谨慎地实现Serializable接口**

* 实现Serialization接口而付出的最大代价是，一旦这个类被发布，就大大降低了”改变这个类实现“的灵活性。
* 实现Serializable的第二个代价是，他增加了出现Bug和安全漏洞的可能性。
* 实现Serialization的第三个代价是，随着类发行新的版本，相关的测试负担也增加了。

序列版本UID与流的唯一识别符有关。

为了继承而设计的类应该尽可能少实现Serializable接口，用户接口也应该尽可能少地继承Serialization接口。

内部类不应该实现Serializatable接口。

静态成员类可以实现Serializable接口。

**第75条：考虑使用自定义的序列化形式。**

如果没有先认真考虑默认的序列化形式是否合适，则不要贸然接受。

如果一个对象的物理表示法等同于它的逻辑内容，可能就适合使用默认的序列化形式。

即使你确定了默认的序列化形式是合适的，通常还必须提供一个readObject方法以确保约束关系和安全性。

当一个对象的物理表示法和它的逻辑内容有实质性的区别时，使用默认序列化形式会有以下4个缺点：

* 它使这个类的导出API永远地束缚在这个类的内部表示法上。
* 它会消耗过多的空间。
* 它会消耗过多的时间。
* 它会引起栈溢出。

如果所有实例都是瞬时的，从技术角度而言，不调用defaultWriteObject和defaultReadObject也是允许的，但是不推荐这样做。

无论怎么，都要为自己编写的每个可序列化的类声明一个显式的序列版本UID。

**第76条：保护性地编写readObject方法**

当一个对象被反序列化的时候，对于包含客户端不应该拥有的对象引用的域，必须做出保护性拷贝。

编写更加健壮的readObject方法：

* 对于对象引用域必须保持为私有的类，要保护性地拷贝这些域中的每个对象。不可变类的可变组件就属于这一类组件。
* 对于任何约束条件，如果检查失败，则抛出一个InvalidObjectException异常。这些检查动作应该跟在所有的保护性拷贝之后。
* 如果整个对象图在被反序列化之后必须进行验证，就应该使用ObjectInputValidation接口。
* 无论是直接当方式还是间接方式，都不要调用类中任何可被覆盖的方法。

**第77条：对于实例控制，枚举类型优先于readResolve**

如果依赖readResolve进行实例控制，带有对象引用类型的所有实例域都必须声明为transient的。

readResolve的可访问性很重要。

应该尽可能地使用枚举类型来实施实例控制的约束条件。如果做不到，同时又需要一个既可序列化又是实例可控的类，就必须提供一个read Resolve方法，并确保该类的所有实例域都为基本类型，或者transient。