Java 2 Serializable 对象中 transient 和 static 型字段序列化方式的探讨

The Java an object inside transient turns the study of the = way with the static type word a preface row

曹大有 （湖北丹江口 郧阳师范高等专科学校计算机科学系 442700）

摘要：Java 类对象的持续性是通过对象的系列化（Serialization）过程来完成的，但默认的系列化方式只能处理非 transient 和非 static 型成员字段。本文则探讨了如何通过自定义系列化方式来完成对 transient 和 static 型成员字段的系列化。

关键词：虚拟机 持续性 系列化 本地方法

Java 虚拟机（JVM）中的对象通常是随生成该对象的程序终止而消失，当不存在对象的引用时，Java 虚拟机中的自动垃圾收集器（garbage collector）就会回收分配给该对象的内存空间。我们如何保存整个对象呢？要完成这个任务，通常情况下就要分别写入对象的每个成员字段的数值，然后在将来的某个时刻，再将写入的每个成员字段的数值读出并指定给某个对象，但这是一个非常复杂的过程。最好的解决方案就是利用对象的持续性，持续（persistence）就是对象记录自己的状态以便将来有再生的能力，对象的持续性允许对象的存在时间比作为其宿主机的 JVM 更长。在 Java 中对象的持续是通过对象的系列化（Serialization）过程来完成的，系列化过程就是对象通过写出描述自己状态的数值来记录自己，系列化过程的主要任务是写出对象实例字段的数值，如果字段是另一对象的引用，则引用的对象也要系列化，这一过程是递归进行的[1]。系列化的条件是只有实现了 Serializable 或 Externalizable 接口的对象才能成功地系列化。在实现了 Serializable 接口的可系列化对象中，并不是所有成员字段都能被系列化，其中标记有 transient、static 关键字的成员字段在默认条件下是不能被系列化的。本文则探讨在 Java 类定义中使用关键字 transient、static 的意义和被标记为 transient、static 的成员字段如何被系列化。

1 关键字 transient、static 的意义

通常情况下，对象中的某些成员字段是不能被系列化的，比如用来保存文件对象句柄或窗口对象句柄的整数，它们只有在本地方法（Native Method）面前才有意义，如果以后重新装载一个对象或将其传到一台不同的机器，这样的信息保证是没有意义的。事实上假如这些成员字段使用了错误的值，完全可能造成本地方法的崩溃。对此 Java 提供了一种便利的机制，可有效保障此类成员字段不会以默认的方式系列化，即用关键词 transient 将其标识即可。另外 static 标记的成员字段对于一台 JVM 来说是全局性数据，因此用单一的对象实例进行传送它是没有意义[2]。相反当接收方 JVM 在最初加载类文件时，类中 static 成员字段都能被正常初始化的。所以默认条件下，对象中被标记为 transient、static 的成员字段是不被系列化的。这个机制可让程序员有机会决定类中那些成员字段是不可以系列化。

2 对象中 transient、static 成员字段的系列化方式

虽然在实现了 Serializable 接口的可系列化对象

中,被标记为 transient、static 的成员字段在默认条件下是不能被系列化的,但我们参考 Java 2 的 java/io/Serializable. java、java/io/ObjectOutputStream. java、java/io/ObjectInputStream. java 等 API 文档时就可以发现(虽然接口 java. io. Serializable 只是一个可系列化的标记,并没有包含任何方法),Java 也给出了代替默认系列化方式的行为,那就是让某一给定的类实现以下两个方法:

private void writeObject (java. io. ObjectOutputStream out) throws IOException;

private void readObject(java. io. ObjectInputStream in) throws IOException, ClassNotFoundException;

来代替默认的系列化方式,当我们通过 java. io. ObjectOutputStream 中的:

public final void writeObject(Object obj) throws IOException 方法来写该类对象时便会调用该类同名私有方法:private void writeObject(java. io. ObjectOutputStream out) throws IOException 来进行对象的系列化操作。同样当我们通过 java. io. ObjectInputStream 中的:

public final Object readObject() throws IOException, ClassNotFoundException 来读该类对象时也会调用该类中同名私有方法:private void readObject(java. io. ObjectInputStream in) throws IOException, ClassNotFoundException 来进行对象的反系列化操作。这样类中标记为 transient、static 的成员字段我们就可以在这两个方法中进行系列化的处理。但在使用这两个方法时要注意以下问题:在 private void writeObject(java. io. ObjectOutputStream out)方法中,应该在用标准 ObjectOutputStream 方法来写任何自己定义的数据之前先调用 defaultWriteObject()方法,即使没有要通过缺省机制进行系列化的数据,这将保证与希望利用默认系列化选项以后的版本相兼容。同样在 private void readObject(java. io. ObjectInputStream in)方法中,应该在用标准 java. io. ObjectInputStream 方法来读任何自己定义的数据之前先调用 defaultReadObject()方法,即使没有要通过缺省机制进行反系列化的数据,这将保证与希望利用默认系列化选项的以后的版本相兼容。具体内容可参考后面的程序实例。

3 transient、static 成员字段系列化和反系列化的实例

下面我们通过一个具体实例来说明上述操作。类定义如下:

```
class MyCircle implements Serializable
{
 protected double x,y,radius;
 public MyCircle(double x,double y,double radius)
 {
  this.x=x;
  this.y=y;
  this.radius=radius;
 }
 public boolean isWithin(double x,double y)
 {
  return (this.x-x)*(this.x-x)+(this.y-y)*(this.y-y)<=radius*radius;
 }
 public String toString()
 {
  return "x="+x+";y="+y+";radius="+radius;
 }
}
class MoreCircle extends MyCircle
{
 private transient double radiusSquared;
 private transient int id;
 private transient Socket socket;
 private static int globalId;
 private synchronized static int getId()
 {
  return globalId++;
 }
 public MoreCircle(double x,double y,double radius,String host,int port) throws IOException
 {
  super(x,y,radius);
  radiusSquared=radius*radius;
```

```
        id = getId( ) ;
        socket = new Socket( host, port) ;
       }
        public boolean isWithin( double x, double y)
       {
        return ( this. x - x) * ( this. x - x) + ( this. y - y)
* ( this. y - y) < = radiusSquared;
       }
       public String toString( )
       {
        return " x = " + x + " ; y = " + y + " ; radius = " +
radius + " ; radiusSquared = " + radiusSquared + " ; id
= " + id + " ; socket = " + socket;
       }
       //private void writeObject ( ObjectOutputStream
objectOut) throws IOException
       //private void readObject( ObjectInputStream ob-
jectIn) throws IOException, ClassNotFoundException
      }
```

类 MyCircle **实现了** Serializable **接口，但类** MyCircle **中没有** transient、static **成员字段。类** MoreCircle **继承于** MyCircle **类，所以它也是可系列化的。我们在类** MoreCircle **中加入一些** transient、static **成员字段，如：** radiusSquared **只是为了计算的方便，**globalId **是全局性数据，**id **是一个** JVM - unique **标识，同时我们也加入了一个** Socket **类对象** socket**，类** Socket **是不可系列化的，所以以上数据分别加上** transient、static **标记。**

private void writeObject( ) **方法的实现为：**

```
private void writeObject ( ObjectOutputStream ob-
jectOut) throws IOException
       {
        objectOut. defaultWriteObject( ) ;
        objectOut. writeObject ( socket. getInetAddress
( ) ) ;
        objectOut. writeInt( socket. getPort( ) ) ;
       }
```

在语句 objectOut. defaultWriteObject( ) **中，我们利用默认系列化机制将类** MyCircle **中的数据成员系列化了，由于类** MoreCircle **中的** radiusSquared **可通过** radius **计算出，**id **是一个标识，它们都没有必要传送，接下来只要将** socket **的** IP **地址和端口号写入即可。**

private void readObject( ) **方法的实现为：**

```
private void readObject ( ObjectInputStream objec-
tIn) throws IOException, ClassNotFoundException
       {
        objectIn. defaultReadObject( ) ;
        radiusSquared = radius * radius;
        id = getId( ) ;
        InetAddress address = ( InetAddress) objectIn.
readObject( ) ;
        int port = objectIn. readInt( ) ;
        socket = new Socket( address, port) ;
       }
```

我们首先调用 objectIn. defaultReadObject( ) **语句使用默认反系列化机制来对所有非** transient、static **成员字段进行反系列化，这样就得到** x, y, radius **的值，然后计算** radiusSquared**，分配一个新的** id**，再按写入的顺序读出** socket **的** IP **地址和端口号，生成** socket**。这样就完成对类中** transient、static **成员字段的系列化过程，具体见所附程序：**ObjectStreamTest. java **所示。示例中的** socket **连接的是美国科罗拉多洲博尔德市的国家标准与技术研究所的时间服务器上，它的** host = " time - A. timefreq. bldrdoc. gov" , port = 13**。**

4 总结

上面我们通过具体实例介绍了如何通过自定义方法来代替默认系列化机制实现 transient、static **成员字段的系列化和反系列化的方法。实现该过程的重点是两个私有方法：**private void readObject( ) **和** private void writeObject( ) **的实现。当然这两个方法的意义不仅在于可以实现类中** transient、static **成员字段的系列化和反系列化，也可以对类中敏感数进行加密和解密（另文讨论）。**

参考文献

1 （澳）David Reilly Michael Reilly **著，沈风等译，**Java **网络编程与分布式计算**[M]**，机械工业出版社，**2003. 76 - 82。

2 （美）Cay S. Horstmann Gary Cornell **著，京京工作室译，**Java 2 **核心技术**（I）**基础知识**[M]**，机械工业出版社，**2000. 486 - 499。