Java 2 定制对象序列化方式的探讨

Java 2 make to order the study that the object sequence turns the way

曹大有　朱 琳　（郧阳师范高等专科学校 计算机科学系 湖北 丹江口 442700）

摘要：在利用 Java 2 缺省的对象序列化机制对对象进行序列化的过程中，会暴露对象一些私有数据，这对有经验的用户来说完全有能力修改对象数据。本文则探讨如何利用 Java 2 序列化机制中提供的特殊方式，来定制对象的序列化方式。

关键词：对象 序列化 反射

1 引言

Java 2 具有对对象进行序列化的能力，而且序列化后的结果采用特殊的文件格式[1]，这种文件格式对有经验的用户来说，是很容易利用这些文件信息来修改对象的内容，使我们在文件重新装载时读入无效或经过修改了的对象。那么怎样才能克服这种缺陷呢？一种有效的办法就是定制对象的序列化方式，控制对象如何序列化，甚至可以对一个对象所拥有的数据以所有者的形式进行加密，以便控制对该对象数据的访问。这种情况下就不能采用 Java 2 的缺省序列化机制，那么怎样定制对象的序列化方式呢？

2 定制对象序列化的方式

在 Java 2 中定制对象序列方式有三种，下面我们以序列化日期类为例来进行探讨。

第一是使用签名在自己的类中定义以下两个方法：

```
private void writeObject ( java. io. ObjectOutput-
Stream out) throws IOException;
private void readObject( java. io. ObjectInputStream
in) throws IOException, ClassNotFoundException;
```

来代替默认的系列化方式。关于这一方式，大家可能要问，以上两个方法根本就不在 Serializable 接口中，Serializable 接口只是对象能否序列化的标志，它没有定义任何方法，序列化系统是如何管理这些方法的调用的呢？答案是序列化系统使用了称为反射（Reflection）的机制，反射是允许程序基于它所知道的方法签名，来访问组件的方法和构造器。但在使用反射时，一定要注意反射需要准确的方法签名，当以上方法签名有误时，Java 2 会自动启用其缺省的序列化机制。

在这种方式中，用户的类不必去关心 super. writeObject( ) 或 super. readObject( ) 的调用，也不必关心子类将如何使用序列化机制，在对象的每个部分处理时将对对象进行分别进行序列化。另外，如果用户需要在 writeObject( ) 方法中使用缺省的机制，调用 out. defaultWriteObject( ) 方法即可，或从 readObject( ) 方法调用 In. defaultReadObject( ) 方法。下面是一个称为 DateTest 的类，在该类中我们使用三个单独的整数来写出日期的值，并没有使用缺省的序列化机制（具体参见程序 DateTest. java 所示）。

```
public class DateTest implements Serializable{
  transient GregorianCalendar myDate;
  public void newDate( ){
    myDate = new GregorianCalendar( );
  }
  private void writeObject ( ObjectOutputStream
out) throws IOException{
    int year = myDate. get( Calendar. YEAR);
    int month = myDate. get( Calendar. MONTH);
    int day = myDate. get ( Calendar. DAY_OF_
```

```
MONTH);
        out.writeInt(year);
        out.writeInt(month);
        out.writeInt(day);
    }
    private void readObject(ObjectInputStream in)
throws IOException,ClassNotFoundException{
        int year=in.readInt();
        int month=in.readInt();
        int day=in.readInt();
        myDate = new GregorianCalendar(year,
month,day);
    }
    public String toString(){
        DateFormat df=DateFormat.getDateInstance
();
        return "DateTest:" +df.format(myDate.get-
Time());
    }
}
```

二是可以不让序列化机制来保存及恢复数据，一个类亦可定义自己的机制。为做到这一点，该类必须实现 Externalizable（外部具体化）接口。而这个接口要求定义两个方法：

```
public void readExternal(ObjectInput in) throws IO-
Exception,ClassNotFoundException;
public void writeExternal(ObjectOutput out) throws
IOException;
```

这个接口应被希望提供人工序列化功能的类来实现，该对象流仅将类的名称写到流中，所有其它数据必须通过这个接口进行交换，使用这种机制的类并没有获取缺省序列化机制提供的版本控制功能。实现 Externalizable（外部具体化具体化）接口的对象必须提供一个无参构造器，通过这个构造器能在从流中进行反序列化之前，创建该对象的一个实例，然后调用 readExternal() 方法。和前面定义的 readObject() 及 writeObject() 方法不同，这些方法将完全负责整个对象的保存及恢复，其中包括超类数据。下面是一个称为 DateTest1 的类，在该类中我们使用 Externalizable（外部具体化）接口同样对日期类进行自定义的序列化（具体参见程序 DateTest1.java 所示）。

```
public class DateTest1 implements Externalizable{
    transient GregorianCalendar myDate;
    public void newDate(){
        myDate=new GregorianCalendar();
    }
    public void writeExternal(ObjectOutput out)
throws IOException{
        int year=myDate.get(Calendar.YEAR);
        int month=myDate.get(Calendar.MONTH);
        int day=myDate.get(Calendar.DAY_OF_
MONTH);
        out.writeInt(year);
        out.writeInt(month);
        out.writeInt(day);
    }
    public void readExternal(ObjectInput in) throws
IOException{
        int year=in.readInt();
        int month=in.readInt();
        int day=in.readInt();
        myDate = new GregorianCalendar(year,
month,day);
    }
    public String toString(){
        DateFormat df=DateFormat.getDateInstance
();
        return "DateTest1:" +df.format(myDate.
getTime());
    }
}
```

三是关闭序列化机制的默认选择，用其它值来进行序列化。默认的序列化机制是对象中所有非静态、非临时的字段都会得到序列化，但有时出于安全方面的考虑，会对这种默认行为感到不满意，想关闭默认机制。为此需要专门指定一个由 ObjectStreamField（对象

流字段)对象组成的数组,其中每个对象都必须定义一个值的名字及类型。必须定义一个 private static final 的数组,并将其称作 serialPersistentFields。如我们想保存日期对象的状态,但不想保存年、月、日字段,而是将日期对象保存到单独一个数字:

10000 * year + 100 * month + day

例如:1962 年 2 月 28 日将保存为数字 19960228。我们将这个值称为 date。随后我们必须告诉日期类,它的序列化形式将由单独一个字段 date 组成,类型为 long,即在日期类中要有以下语句:

```
private static final ObjectStreamField[ ]
    serialPersistentFields = { new ObjectStreamField
("date",long.class),};
```

现在我们需要接管这个日期类的流式处理。在 writeObject( ) 方法中,我们用 putFields( ) 方法取得对象的字段集合(这个方法会返回一个对象,其中封装了字段集 - - - 其类型为内部类 ObjectOutputStream.PutField)。随后我们设置 date 字段的值,最后将该字段集写入流中:

```
private void writeObject(ObjectOutputStream out)
throws IOException{
        int year = myDate.get(Calendar.YEAR);
        int month = myDate.get(Calendar.MONTH);
        int day = myDate.get(Calendar.DAY_OF_
MONTH);
        ObjectOutputStream.PutField fields = out.put-
Fields( );
        fields.put("date",year * 10000L + month *
100 + day);
        out.writeFields( );
    }
```

要读回对象,同样需要覆盖 readObject( ) 方法。首先用 readFields( ) 方法读入所有字段,随后用内部类 ObjectInputStream.GetField 的已经覆盖的 get( ) 方法取得每个字段的值。get( ) 方法的第一个参数是字段名,第二个是默认值,在字段不存在时使用。常用的 get 方法如下所示:

```
int get(String name,int defval);
long get(String name,int defval);
float get(String name,int defval);
double get(String name,int defval);
char get(String name,int defval);
short get(String name,int defval);
```

下面是我们修改过的日期类的 readObject( ) 方法,它会读取 date 值,并将其分解为年、月、日数据。

```
private void readObject(ObjectInputStream in)
throws IOException,ClassNotFoundException{
        ObjectInputStream.GetField fields = in.read-
Fields( );
        long date = fields.get("date",0L);
        int day = (int)(date% 100);
        int month = (int)((date/100)% 100);
        int year = (int)(date/10000);
        myDate = new GregorianCalendar(year,
month,day);
    }
```

具体可参见程序:DateTest2.java 所示。

3 总结

以上是我们对 Java 2 中定制对象序列化方式的探讨。Java 的对象序列化机制是一个强大、方便的工具,它能使我们很容易的存取 Java 对象。它的缺点是在存取过程中暴露了对象的内部数据,造成数据保存时对象的不安全性。定制对象序列化是解决这个缺点的关键所在,本文只探讨了定制对象的序列化过程,其实可以在这个过程之上进行数据的加密和解密,使对象数据的保存更加安全。

参考文献

1 (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],北京 机械工业出版社,2000.489 - 497。