Java 序列化与多线程

- ✓ 出处：站长百科
- ✓ 原文地址：http://www.zzbaike.com/wiki/JAVA_序列化
- ✓ 本电子书整理自站长百科JAVA_序列化词条，查看内容请访问网站。

目录

✓ 出处：站长百科
✓ 原文地址：http://www.zzbaike.com/wiki/JAVA_序列化
✓ 本电子书整理自站长百科 JAVA_序列化词条，查看内容请访问网站。

JAVA 序列化

各种类型的数据。无论是何种类型的数据，都会以二进制序列的形式在网络上传送。发送方需要把这个 Java 对象转换为字节序列，才能在网络上传送;接收方则需要把字节序列再恢复为 Java 对象。把 Java 对象转换为字节序列的过程称为对象的序列化。

把字节序列恢复为 Java 对象的过程称为对象的反序列化。

对象的序列化主要有两种用途：

1) 把对象的字节序列永久地保存到硬盘上，通常存放在一个文件中;

2) 在网络上传送对象的字节序列。

JDK类库中的序列化API

java.io.ObjectOutputStream 代表对象输出流，它的writeObject(Object obj)方法可对参数指定的obj对象进行序列化，把得到的字节序列写到一个目标输出流中。

java.io.ObjectInputStream 代表对象输入流，它的readObject()方法从一个源输入流中读取字节序列，再把它们反序列化为一个对象，并将其返回。、

只有实现了 Serializable 和 Externalizable 接口的类的对象才能被序列化。Externalizable 接口继承自 Serializable 接口，实现 Externalizable 接口的类完全由自身来控制序列化的行为，而仅实现 Serializable 接口的类可以采用默认的序列化方式 。

对象序列化包括如下步骤：

1）创建一个对象输出流，它可以包装一个其他类型的目标输出流，如文件输出流；

2）通过对象输出流的 writeObject()方法写对象。

对象反序列化的步骤如下：

1）创建一个对象输入流，它可以包装一个其他类型的源输入流，如文件输入流；

2）通过对象输入流的 readObject()方法读取对象。

下面让我们来看一个对应的例子，类的内容如下：

```
import java.io.*;

import java.util.Date;  /**  * 对象的序列化和反序列化测试类.

* @author AmigoXie  * @version 1.0  * Creation date:
```

```
2007-9-15 - 下午 21:45:48 */ public class ObjectSaver { /** * @param args * @author AmigoXie * Creation date: 2007-9-15 - 下午 21:45:37 */ public static void main(String[] args) throws Exception { ObjectOutputStream out = new ObjectOutputStream (new FileOutputStream("D:""objectFile.obj")); //序列化对象

    Customer customer = new Customer("阿蜜果", 24);

    out.writeObject("你好!");

    out.writeObject(new Date());

    out.writeObject(customer);

    out.writeInt(123); //写入基本类型数据

    out.close(); //反序列化对象

    ObjectInputStream in = new ObjectInputStream (new FileInputStream("D:""objectFile.obj"));

    System.out.println("obj1=" + (String) in.readObject());

    System.out.println("obj2=" + (Date) in.readObject());

    Customer obj3 = (Customer) in.readObject();
```

```
System.out.println("obj3=" + obj3);

int obj4 = in.readInt();

System.out.println("obj4=" + obj4);

in.close(); } } class Customer implements Serializable { private String name;

private int age;

public Customer(String name, int age) { this.name = name; this.age = age; } public String toString() { return "name=" + name + ", age=" + age; } }
```

输出结果如下：

```
obj1=你好!

obj2=Sat Sep 15 22:02:21 CST 2007

obj3=name=阿蜜果, age=24

obj4=123
```

因此例比较简单，在此不再详述。

实现Serializable接口

ObjectOutputStream 只能对 Serializable 接口的类的对象进行序列化。默认情况下，ObjectOutputStream 按照默认方式序列化，这种序列化方式仅仅对对象的非 transient 的实例变量进行序列化，而不会序列化对象的 transient 的实例变量，也不会序列化静态变量。

当 ObjectOutputStream 按照默认方式反序列化时，具有如下特点：

1）如果在内存中对象所属的类还没有被加载，那么会先加载并初始化这个类。如果在 classpath 中不存在相应的类文件，那么会抛出 ClassNotFoundException;

2）在反序列化时不会调用类的任何构造方法。

如果用户希望控制类的序列化方式，可以在可序列化类中提供以下形式的 writeObject()和 readObject()方法。

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

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

当 ObjectOutputStream 对一个 Customer 对象进行序列化时，如果该对象具有 writeObject()方法，那么就会执行这一方法，否则就按默认方式序列化。在该对象的 writeObjectt()方法中，可以先调用 ObjectOutputStream 的 defaultWriteObject()方法，使得对象输出流先执行默认的序列化操作。同理可得出反序列化的情况，不过这次是 defaultReadObject()方法。

有些对象中包含一些敏感信息，这些信息不宜对外公开。如果按照默认方式对它们序列化，那么它们的序列化数据在网络上传输时，可能会被不法份子窃取。对于这类信息，可以对它们进行加密后再序列化，在反序列化时则需要解密，再恢复为原来的信息。

默认的序列化方式会序列化整个对象图，这需要递归遍历对象图。如果对象图很复杂，递归遍历操作需要消耗很多的空间和时间，它的内部数据结构为双向列表。

在应用时，如果对某些成员变量都改为 transient 类型，将节省空间和时间，提高序列化的性能。

实现Externalizable接口

Externalizable 接口继承自 Serializable 接口，如果一个类实现了 Externalizable 接口，那么将完全由这个类控制自身的序列化行为。Externalizable 接口声明了两个方法： public void

writeExternal(ObjectOutput out) throws IOException

public void readExternal(ObjectInput in) throws IOException , ClassNotFoundException

前者负责序列化操作，后者负责反序列化操作。

在对实现了 Externalizable 接口的类的对象进行反序列化时，会先调用类的不带参数的构造方法，这是有别于默认反序列方式的。如果把类的不带参数的构造方法删除，或者把该构造方法的访问权限设置为 private、默认或 protected 级别，会抛出 java.io.InvalidException: no valid constructor 异常。

可序列化类的不同版本的序列化兼容性

凡是实现 Serializable 接口的类都有一个表示序列化版本标识符的静态变量：

private static final long serialVersionUID;

以上 serialVersionUID 的取值是 Java 运行时环境根据类的内部细节自动生成的。如果对类的源代码作了修改，再重新编译，新生成的类文件的 serialVersionUID 的取值有可能也会发生变化。

类的 serialVersionUID 的默认值完全依赖于 Java 编译器的实现，对于同一个类，用不同的 Java 编译器编译，有可能会导致不同

的 serialVersionUID，也有可能相同。为了提高哦啊 serialVersionUID 的独立性和确定性，强烈建议在一个可序列化类中显示的定义 serialVersionUID，为它赋予明确的值。显式地定义 serialVersionUID 有两种用途：

1）在某些场合，希望类的不同版本对序列化兼容，因此需要确保类的不同版本具有相同的 serialVersionUID;

2）在某些场合，不希望类的不同版本对序列化兼容，因此需要确保类的不同版本具有不同的 serialVersionUID。

JAVA 多线程

Java 语言中有一个重要的特性是支持多线程。多线程是 java 的一项高级技术，它涉及到操作系统里面的知识，层次贴近系统层面。对于普通程序员一般 很少碰它。而且目前就是在 java EE（原来的 J2EE）的相关框架里，对线程这个东西都是尽量回避。程序员最理想的状态是专注业务逻辑，而不是天天想着线程这个东西怎么写。

思考一个问题程序的本质是什么？是 CPU 的指令序列的集合。到底什么顺序是程序员编写的让计算机赋值，它就赋值、写个循环它就循环、写个分 支语句它就分支、写个跳转它就跳转。每个指令流就是一个线程，并发执行多个指令流就是多线程。大家想，只有一个

CPU 怎么可能同时发出多个指令流呢？是 的，并发只是“逻辑”上的，物理上是不可能的除非是两个以上的 CPU.

多线程和传统的单线程的区别是由于各个线程的控制流彼此独立，使得各个线程之间的代码是乱序执行的，出现了并发访问带来的一切问题。正像是三个和尚的故事，和尚多了未必是好事。也就是刚才说的，程序员一般都不让他们碰这个东西。

在 java 中如何写线程呢，在 java 中就是很简单了。有两种方法：第一、继承 java.lang.Thread 第二、实现 Runnable 接口。

实践：

```
//继承 Thread 而重写了 run()方法

public class Hello extends Thread{

int i;

public void run(){

   while(true){

       System.out.println("Hello "+i++);

       if(i==10) break;
```

```
        }}}

public class HelloThread {

  public static void main(String[] args){

     Hello h1 = new Hello();

     Hello h2 = new Hello();

     h1.start(); //用两个线程执行那 10 次循环

     h2.start();

}} 上面的例子是第一种方法，下面是第二种方法

public class TestThread {

public static void main(String args[]) {

  Xyz r = new Xyz();

  Xyz r1 = new Xyz();

  Thread t1 = new Thread(r);

  Thread t2 = new Thread(r1);

  t1.start();//用两个线程执行那 50 次循环
```

```
        t2.start();
    }} //实现 Runnable 接口
     class Xyz implements Runnable {
     int i;
     public void run() {
       i = 0;
       while (true) {
         System.out.println("Hello " + i++);
         if ( i == 50 ) {
       break;
     }}}}
```

上面两种方法继承 Thread 类，是比较简单的，代码也比较少。但是我们不提倡使用这种方法。而第二种实现 Runnable 接口，更符合面向对象思 想，Thread 是把虚拟的 CPU 看成一个对象，封装了 CPU 的细节。但是 Thread 的构造线程的子类的方法中与 CPU 不相关，没

有必要把 CPU 的细节 都继承来。而实现 Runnable 则不影响 java.lang.Thread 的体系。而且便于其它类的继承。

更多电子书

SEO 方面的知识有很多，对于新手来说，如果你不知道，不清楚这方面的知识，那么，你可以看看这两部搜索引擎指南：

SEO搜索引擎优化基础教程：http://bbs.zzbaike.com/thread-9952-1-1.html

SEO搜索引擎优化高级教程：http://bbs.zzbaike.com/thread-12692-1-1.html

wordpress的中文翻译文档：http://www.wordpress.la/codex.html，wordpress 开发的相关知识都有，对WordPress开发感兴趣的博主会有一定的用处。

WordPress免费主题：http://www.wordpress.la/theme.html

WordPress免费插件：http://www.wordpress.la/plugin.html

WordPress主题制作电子书：http://bbs.zzbaike.com/thread-9954-1-1.html

Apache2.2 中文文档电子书：http://bbs.zzbaike.com/thread-9955-1-1.html

IXWeHosting 控制面板使用手册(在线版+PDF 电子书)：

http://bbs.zzbaike.com/thread-9953-1-1.html

更多电子书下载：http://down.zzbaike.com/ebook/

视频教程：

1. 美国主机 IXWebHosting 使用视频教程（在线观看及下载）

   http://bbs.zzbaike.com/thread-47008-1-1.html

2. Godaddy 主机及域名使用视频教程（在线观看及下载）

   http://bbs.zzbaike.com/thread-50005-1-1.html