登录 | 注册

sinat_32955803的博客

想了好久 实在想不到一句装13的描述

个人资料

起晚贪黑

访问： 15791次

积分： 458

等级： BLOG 2

排名： 千里之外

原创： 22篇 转载： 53篇

译文： 0篇 评论： 1条

文章搜索

黑马程序员 www.itheima.com
人工智能掘金时代

【活动】Python创意编程活动开始啦！！！ CSDN日报20170502 ——《程序学徒与导师》 深入浅出，带你学习 Unity

java 的序列化和反序列化的概念及简单使用

标签： java 序列化和反序列化 基础

2016-05-05 13:51 2559人阅读 评论(

分类：

java 笔记（9）

版权声明：本文为博主原创文章，未经博主允许不得转载。

一、序列化和反序列化的含义

注：序列化的其实是对象在某一时刻的状态（冻结对象状态）。如person.setAge(100)，就是保存的person年龄为100的状态，无论用何种方式，控制序列化细节，不控 制序列化细节也好，序列化的都是对象在某一时刻的状态，也就是持久化对象这一时刻的状态；反序列化就是将这一对象状态解冻。

反序列化出来的对象还是原来的对象吗，或者他是一个新的对象实例？答案是一个新的对象实例，具体请看第六点。

1. 序列化：将java对象转换为可保持或可传输的格式，即转换为二进制字节序列（字节流）的形式的过程。
2. 反序列化：将二进制字节序列（字节流）形式的java对象解析出来的过程。

- 发送方需要将java对象进行序列化，即转换为二进制字节序列（字节流）形式。
- 接收方需要将二进制字节序列（字节流）形式的对象恢复出来。

二、为什么要进行序列化

- 当两个进程间进行通讯时可以相互发送任何资源（数据），这些资源包括图片、文本、音频、视频等，而这些资源在网络上传输的格式均为二进制序列（字节流）。
- 当两个java进程进行通讯时能否实现进程间的对象传输呢。是可以的，但必须完成对象的序列化，即将对象转换为二进制序列（字节流）的形式。

三、序列化的用途及好处

1. 序列化可以实现数据的持久化，也就是说可以将数据永久的保存在磁盘上。
2. 序列化可以实现远程通讯，即在网络上以二进制字节序列的格式传送对象。
3. 序列化可以将对象状态保存，方便下次取出利用。
4. 有了序列化，两个进程就可以在网络上任性的交互数据了。

举例：

1. java的HashMap类实现了Serializable接口，所以在服务器上使用map.put("key",value)方法添加数据，然后将这个HashMap以二进制字节序列在网络上传送出去，可以到手机，可以到另一个终端……。在终端在将此HashMap解析出来，将数据显示或利用。
2. 以android程序和服务器来说，两个进程进行数据的传递。服务器将数据封装进一个bean里或者HashMap里，或以List集合的形式，，一般是服务器向将对象打成json字符串的形式，然后以二进制字节序列的形式传送到手机终端，手机终端或其他终端在将此二进制字节序列（字节流）反序列化成json字符

评论排行

android studio 设置大全 (1)
android 的外部存储的挂 (0)
内存和File (0)
android Uri利用及解析（ (0)
android 获取文件、存储 (0)
相对路径和绝对路径的理 (0)
android 的内存、内部存 (0)
android 根目录 File Expl (0)
java Uri Url Urn android (0)
retrofit 注解含义 (0)

推荐文章

* CSDN日报20170502 ——《程序学徒与导师》
* 抓取网易云音乐歌曲热门评论生成词云
* Android NDK开发之从环境搭建到Demo级十步流
* 个人的中小型项目前端架构浅谈
* 基于卷积神经网络(CNN)的中文垃圾邮件检测
* 四无年轻人如何逆袭

最新评论

android studio 设置大全
我养了几条鱼: 你这个大全真的很"全"。。。。。

串，然后在将此json字符串进一步解析，最后解析成原本的对象（数据），实现数据的任性交互，反之亦然，其实不是jdk的序列化。

四、transient关键字（影响序列化）

- **有些情况下不能使用默认的序列化方式，比如，某个变量不想序列化、简化序列化过程等，需要用到影响序列化，影响序列化可以使我们自身控制序列化的行为。**
- 当某个字段被声明为transient后，默认序列化机制就会忽略该字段

五、怎样实现序列化

- 为什么实现了Serializable就可以序列化，这源于objectOutputStream类，如果写入对象是对象的类型是String，或数组，或Enum，或Serializable，那么就可以对该对象进行序列化，否则将抛出NotSerializableException。

1. 继承Serializable接口，使用默认的方式来进行序列化，这种序列化方式仅仅对对象的非tran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，所以我们对不想持久化的变量可以加上transient关键字。注意使用默认机制，在序列化对象时，不仅会序列化当前对象本身，还会对该对象引用的其它对象也进行序列化，同样地，这些其它对象引用的另外对象也将被序列化，以此类推。所以，如果一个对象包含的成员变量是容器类对象，而这些容器所含有的元素也是容器类对象，那么这个序列化的过程就会较复杂，开销也较大。
2. 继承Externalnalizable（Externalnalizable为Serializable的子类）接口，自身控制序列化的行为。

方式一（默认序列化）：直接继承Serializable接口，采用默认的序列化方式，需要注意的是还原出来的对象并没有调用任何构造器，就像是直接将对象还原出来一样，再赋予我们的引用（其实在反序列化的时候反序列化出来的对象已经进行了重构并实例化，当然不需要调用构造器啦）！当另一个线程反序列化Person对象时，必须保证反序列化的线程存在Person.class,否则报ClassNotFound异常。代码如下：

```
[java]
01.       /**
02.  * 将person对象序列化
03.  * @Title: SerializePerson
04.  * void
05.  * @author shimy
06.  * @since 2016-5-5 V 1.0
07.  */
08. private String pathString = "D:" + File.separator + "test2"+File.separator;
09. File files = new File("pathString");
10. private void SerializePerson() throws IOException{
11.
12.
13.     Person person = new Person();
14.     person.setAge(11000);
15.     person.setName("寂寞高手一时去无踪");
16.     person.setSex("男");
17.
18.
19.     //找一个文件夹
20.
21.     if (!files.exists()) {//如果这个资源不存在
22.         files.mkdirs();
23.     }
24.     File file = new File(files, "person.txt");
25.     //创建一个文件输出流，向磁盘写入数据
26.     FileOutputStream outputStream = new FileOutputStream(file);
27.     //创建一个对象输出流，将对象序列化并写入到磁盘上
28.     ObjectOutputStream objectOutputStream = new ObjectOutputStream(outputStream);
29.     //将对象序列化并写入磁盘持久化
30.     objectOutputStream.writeObject(person);
31.     //记得关闭流
32.     objectOutputStream.flush();
33.     objectOutputStream.close();
34.     area.setText("序列化成功");
35.
36. }
37. /**
38.  * 将person对象反序列化
39.  * @Title: DeserializePerson
40.  * void
```


```
 * @author shimy
 * @throws IOException
 * @throws ClassNotFoundException
 * @since 2016-5-5 V 1.0
 */
private void DeserializePerson() throws IOException, ClassNotFoundException{
    File file = new File(files, "person.txt");
    FileInputStream fileInputStream = new FileInputStream(file);
    ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);
    Person person = (Person) objectInputStream.readObject();
    area.setText(person.getAge() + "\n"
            +person.getName() + "\n"
            +person.getSex());
    objectInputStream.close();
}
```

方式二（影响序列化）：

序列化对象中实现writeObject()方法与readObject()方法，当实现这两个方式的时候，序列化的细节就可以在这里

由我们自己来控制,需要注意的是这两个方法是由java的反射机制来调用的。

为什么这两个方法可以影响序列化：如下：

ObjectOutputStream源码： writeObject() ->writeObject0() ->writeOrdinaryObject() ->writeSerialD...

反射请求到我们自己的序列化对象里面的writeObject()方法，反序列化大同小异，不在陈述。

需要注意的是：在需要序列化的对象（如Person）里添加writeObject()和readObject()时，ObjectOutputStream对象在将对象序列化的过程中调用writeObject()

将会通过反射查找序列化对象（Person）是否存在writeObject()方法，如果有，优先调用需要序列化的对象（Person）d的writeObject()，而不是调用自己的

writeObject()方法，没有就调用自己的writeObject()方法，即默认的。ObjiecInputStream相同。举例如下：

需要序列化的类的代码：

```
[java]
package com.yue.xlh;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.HashMap;
/**
 * 定义的一个可序列化对象，一个标准的bean形式，用于测试
 * @ClassName: Person
 * @Description: TODO
 * @date 2016-5-5 下午3:20:53
 *
 */
public class WRPerson implements Serializable{

    /**
     *
     */
    private static final long serialVersionUID = -6827072970889122O1L;
    /**
     * 定义一个序列化id
     */
    private int age;
    private String name;
    private String sex;
    private transient String fujia;


    public String getFujia() {
        return fujia;
    }
    public void setFujia(String fujia) {
        this.fujia = fujia;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getName() {
```


```
43.         return name;
44.     }
45.     public void setName(String name) {
46.         this.name = name;
47.     }
48.     public String getSex() {
49.         return sex;
50.     }
51.     public void setSex(String sex) {
52.         this.sex = sex;
53.     }
54.     /**
55.      * 通过反射机制调用,序列化时ObjectOutputStream就会通过java的反射机制调用这个写入方法，
56.      * 然后再调用自己的写入方法。
57.      * 这个最后的o.writeObject(userter);在实现的过程中，会通过反射在userter中寻找方法名为
writeObject，
58.      * 参数为ObjectOutputStream的方法，如果找到了就会调用userter.writeObject(o)的方法；没找到的话
会使用默认的实现，这种情况下你的这个类中的password字段由于是transient的就会丢失掉。
59.      * 反过来读取对象的时候也是这样的。
60.      * @Title: writeObject
61.      * @param out
62.      * @throws IOException
63.      * void
64.      * @author shimy
65.      * @since 2016-5-5 V 1.0
66.      */
67.     private void writeObject(ObjectOutputStream out) throws IOException{
68.         HashMap<String, String> map = new HashMap<String, String>();
69.         map.put("ll", "小明");
70.         out.writeObject(map);
71.     }
72.     /**
73.      * 反序列时调用的方法,上面序列化的内容可以从这儿被解读出来
74.      * @Title: readObject
75.      * @param in
76.      * @throws IOException
77.      * @throws ClassNotFoundException
78.      * void
79.      * @author shimingyue
80.      * @since 2016-5-5 V 1.0
81.      */
82.     private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException{
83.         HashMap<String, String> map = (HashMap<String, String>) in.readObject();
84.         System.out.println(map.get("ll"));
85.         this.setFujia(map.get("ll").toString());
86.     }
87. }
```

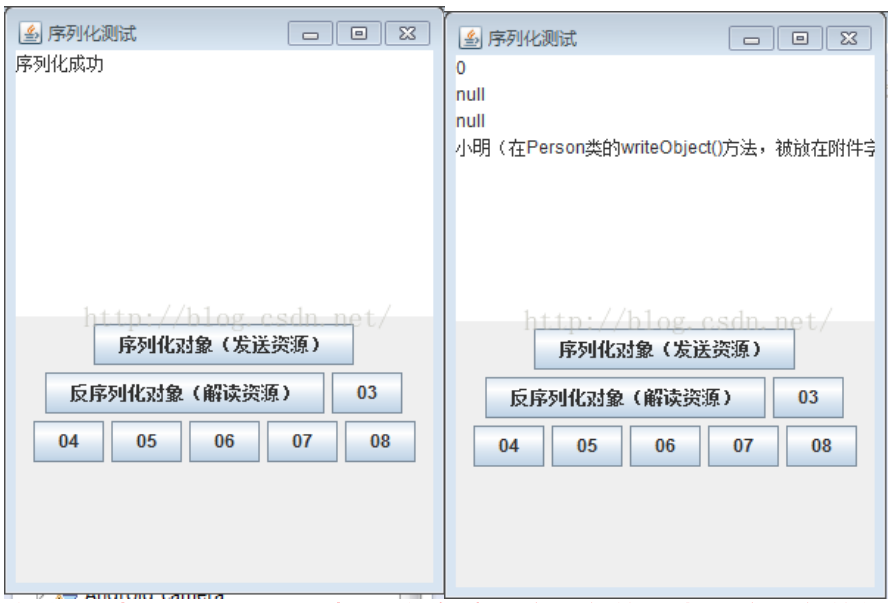
main方法里执行序列化和发序列化的方法：

```
[java]
01. private String pathString = "D:" + File.separator + "test2"+File.separator;
02.     File files = new File("pathString");
03.     private void SerializePerson() throws IOException{
04.
05.
06.         WRPerson person = new WRPerson();
07.         person.setAge(11000);
08.         person.setName("寂寞高手一时去无踪");
09.         person.setSex("男");
10.
11.
12.         //找一个文件夹
13.
14.         if (!files.exists()) {//如果这个资源不存在
15.             files.mkdirs();
16.         }
17.         File file = new File(files, "person.txt");
18.         //创建一个文件输出流，向磁盘写入数据
19.         FileOutputStream outputStream = new FileOutputStream(file);
20.         //创建一个对象输出流，将对象序列化并写入到磁盘上
21.         ObjectOutputStream objectOutputStream = new ObjectOutputStream(outputStream);
22.         //将对象序列化并写入磁盘持久化
23.         objectOutputStream.writeObject(person);
24.         //记得关闭流
25.         objectOutputStream.flush();
26.         objectOutputStream.close();
```


```
27.             area.setText("序列化成功");
28.         }
29.         /**
30.          * 将person对象反序列化
31.          * @Title: DeserializePerson
32.          * void
33.          * @author shimy
34.          * @throws IOException
35.          * @throws ClassNotFoundException
36.          * @since 2016-5-5 V 1.0
37.          */
38.         private void DeserializePerson() throws IOException, ClassNotFoundException{
39.             File file = new File(files, "person.txt");
40.             FileInputStream fileInputStream = new FileInputStream(file);
41.             ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);
42.             WRPerson person = (WRPerson) objectInputStream.readObject();
43.             area.setText(person.getAge() + "\n"
44.                     +person.getName() + "\n"
45.                     +person.getSex()+"\n"
46.                     +person.getFujia());
47.             objectInputStream.close();
48.         }
```

最终的结果：可以看到序列化执行的是Person类里面的writeObject()方法，并将里面的HashMap序列化。可以Person的writeObject()方法里调用了ObjcetOutputStream
类的writeObject()方法最终完成对象状态的序列化。反序列化的是Person的readObject()方法，并在其内调用了ObjectInputStream的readObjcet()方法最终完成对
HashMap的最终反序列化，并将内容取出放在fujia字段里，所以最终可以取出显示。而我们的age，name,sex字段状态并没有被序列化到文件里，因为他们根本就
没有被序列化。

而使用默认序列化的话就是在序列化对象某一时刻的状态是里面的transient和静态的过滤掉，其余状态全部序列化。



方式三（Externalizable）：自身控制序列化的行为和序列化的细节，可以对密码进行加密等操作。

首先：它与Person(需要序列化的对象)实现writeObject()和readObjce()方法有什么区别呢，他们不都可以控制序列化的行为吗?

- 无论是通过transient关键字，还是实现writeObject()和readObjec()方法来影响序列化的过程，他们本身还是基于Serializable接口的，

而jdk提供了另一种Externalizable接口来实现序列化，虽然Externalizable接口继承自Serializable接口，但他们的序列化机制是完全
不同的，也就是说序列化继承此接口的话，Serializable所有序列化机制全部失效。

- 使用Serializable的所有方式，在反序列化是不会调用任何的序列化对象构造器，而使用Externalizable是会调用一个无参构造方法的，原因如下：

- Externalizable序列化的过程：使用Externalizable序列化时，在进行反序列化的时候，会重新实例化一个对象，然后再将被反序列化的对象的状态全部复制到这个新的实例化对象当中去，这也就是为什么会调用构造方法啦，也因此必须有一个无参构造方法供其调用，并且权限是public。
- 示例代码：

```
[java]
01. public class EXPerson implements Externalizable{
02. 
03.     /**
04.      *
05.      */
06.     private static final long serialVersionUID = -6827072970889122O1L;
07.     /**
08.      * 定义一个序列化id
09.      */
10.     private int age;
11.     private String name;
12.     private String sex;
13.     private transient String fujia;
14. 
15. 
16.     public EXPerson() {
17.         super();
18.         System.out.println("调用了EXPerson的构造方法");
19.     }
20. 
21.     /**
22.      *
23.      */
24.     private void writeObject(ObjectOutputStream out) throws IOException{
25.         HashMap<String, String> map = new HashMap<String, String>();
26.         map.put("ll", "小明（在Person类的writeObject()方法，被放在附件字段里）");
27.         out.writeObject(map);
28.     }
29. 
30.     private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException{
31.         HashMap<String, String> map = (HashMap<String, String>) in.readObject();
32.         System.out.println(map.get("ll"));
33.         this.setFujia(map.get("ll").toString());
34.     }
35. 
36.     @Override
37.     public void writeExternal(ObjectOutput out) throws IOException {
38.         // TODO Auto-generated method stub
39. 
40.     }
41.     @Override
42.     public void readExternal(ObjectInput in) throws IOException,
43.             ClassNotFoundException {
44.         // TODO Auto-generated method stub
45.     }
46. 
47. 
48. 
49. }
```

- 打印结果：[调用了EXPerson的构造方法]
  [0][null][null][null]
- 可以看到全部字段都没有被序列化，而且在Person里writeObject()和readObject()也没有被通过反射调用，但是在反序列化时调用了构造方法。为什么会这样，因为我们没有writeExternal()和readExternal()这两个方法里做任何自己的序列化和反序列化处理，而这两个方法也是由我们自身控制序列化细节的方法。怎么用，加入我想序列化name 和fujia两个字段，其他的不被序列化，代码如下：

```
[java]
01. /**
02.      * 序列化两个参数
03.      */
04.     @Override
05.     public void writeExternal(ObjectOutput out) throws IOException {
06.         out.writeObject(getName());
```


```
        out.writeObject(getFujia());
    }
    /**
     * 将两个参数反序列化
     */
    @Override
    public void readExternal(ObjectInput in) throws IOException,
            ClassNotFoundException {
        setName((String)in.readObject());
        setFujia((String)in.readObject());
    }
```

- 打印结果:[调用了EXPerson的构造方法][0][寂寞高手一时去无踪][null][小明(附加字段)]
- 可以看到我们将自己想要序列化的参数序列化了，而不想序列化的参数就没有被序列化，其实他们都在对象状态中。

六、单例模式和枚举的兼容：readResolve()和writeReplace()方

重构：对原有的“坏味道”的代码进行整理，美化代码。

在使用ObjectIInputStream实现对对象状态的解冻（即反序列化）的过程中，会对解冻出来的对象进行重构，并生成一个新的对象实例，这个新的对象实例拥有原来对象的所有状态。其实我们反序列化后获得的并不是原来的对象，而是经过重构的新的对象实例。但Java中枚举和单例都存在唯一性，所以如果对单例和枚举进行序列化，在反序列化的时候就违反了单例和枚举的唯一性原则，这是不允许的。那怎么解决这个问题呢。

无论是实现Serializable接口，或是Externalizable接口，当从I/O流中读取对象时，readResolve()方法都会被调用到。实际上就是用readResolve()中返回的对象直接替换在反序列化过程中重构的对象。

- 这两个方法究竟是怎么一回事：摘自：http://blog.sina.com.cn/s/blog_4e345ce70100rt86.html
- 对于实现 Serializable 或 Externalizable 接口的类来说，writeReplace()方法可以使对象被写入流以前，用一个对象来替换自己。当序列化时，可序列化的类要将对象写入流，如果我们想要另一个对象来替换当前对象来写入流，则可以要实现下面这个方法，方法的签名也要完全一致：ANY-ACCESS-MODIFIERObject writeReplace() throws ObjectStreamException;
- writeReplace()方法在 ObjectOutputStream 准备将对象写入流以前调用，ObjectOutputStream 会首先检查序列化的类是否定义了writeReplace()方法，如果定义了这个方法，则会通过调用它，用另一个对象替换它写入流中（没有就调用默认的）。方法返回的对象要么与它替换的对象类型相同，要么与其兼容，否则，会抛出ClassCastException 。
- 同理，当反序列化时，要将一个对象从流中读出来，我们如果想将读出来的对象用另一个对象实例替换，则要实现跟下面的方法的签名完全一致的方法。ANY-ACCESS-MODIFIERObject readResolve() throws ObjectStreamException;
- readResolve 方法在对象从流中读取出来的时候调用， ObjectInputStream会检查反序列化的对象是否已经定义了这个方法（没有就调用默认的），如果定义了，则读出来的对象返回一个替代对象。同writeReplace()方法，返回的对象也必须是与它替换的对象兼容，否则抛出ClassCastException。如果序列化的类中有这些方法，那么它们的执行顺序是这样的：
- a. writeReplace()
- b. writeObject()或者writeExternal()
- c. readObject()或者readExternal()
- d. readResolve()

下面是 Java doc 中关于 readResolve() 与 writeReplace()方法的英文描述:

Serializable classes that need to designate an alternative objectto be used when writing an object to the stream should implementthis special method with the exact signature:

ANY-ACCESS-MODIFIER Object writeReplace() throwsObjectStreamException;

This writeReplace method is invoked byserialization if the method exists and it would be accessible froma method defined within the class of the object being serialized.Thus, the method can have private, protected and package-privateaccess. Subclass access to this method follows java accessibilityrules.

Classes that need to designate a replacement when an instance ofit is read from the stream should implement this special methodwith the exact signature.

ANY-ACCESS-MODIFIER Object readResolve() throwsObjectStreamException;

This readResolve method follows the sameinvocation rules and accessibility rules as writeReplace.

说了一大堆，我们可以在这两个方法中完成序列化对单例和枚举的兼容。代码如下：

要序列化的类：

```
[java]
01. public class EXPersonSingle implements Externalizable{
02. 
03. 
04.     private static final long serialVersionUID = -6827072970889122O1L;
05. 
06. 
07. 
08.     /**
09.      * 这个是通过java 的反射机制来实现调用的,在执行writeObject()或者writeExternal()
10.      * 前调用
11.      * 用于替换序列化的对象
12.      */
13.     private Object writeReplace()throws ObjectStreamException{
14.         System.out.println("writeReplace");
15.         return this;
16.     }
17.     /**
18.      * 这个是通过java 的反射机制来实现调用的，在执行readObject()或者readExternal()
19.      * 前调用
20.      * 用于替换发序列化出来的对象
21.      */
22.     private Object readResolve() throws ObjectStreamException {
23.         System.out.println("readResolve");
24.         return this;
25.     }
26.     /**
27.      * 序列化两个参数,ObjectOutputStream会通过反射优先执行这里的，如果没有则执行默认的
28.      */
29.     @Override
30.     public void writeExternal(ObjectOutput out) throws IOException {
31.         System.out.println("writeExternal");
32.         out.writeObject(getName());
33.         out.writeObject(getFujia());
34. 
35.     }
36.     /**
37.      * 将两个参数反序列化,ObjectInputStream会通过反射优先执行这里的，如果没有则执行默认的
38.      */
39.     @Override
40.     public void readExternal(ObjectInput in) throws IOException,
41.             ClassNotFoundException {
42.         System.out.println("readExternal");
43.         setName((String)in.readObject());
44.         setFujia((String)in.readObject());
45.             }
46. }
```

序列化方法：

```
[java]
01. private String pathString = "D:" + File.separator + "test2"+File.separator;
02.     File files = new File("pathString");
03.     private void SerializePerson() throws IOException{
04. 
05. 
06.         EXPersonSingle person = new EXPersonSingle();
07.         person.setAge(11000);
08.         person.setName("寂寞高手一时去无踪");
09.         person.setSex("男");
10.         person.setFujia("小明(附加字段)");
11. 
12. 
13.         //找一个文件夹
14. 
15.         if (!files.exists()) {//如果这个资源不存在
```


```
            files.mkdirs();
        }
        File file = new File(files, "person.txt");
        //创建一个文件输出流，向磁盘写入数据
        FileOutputStream outputStream = new FileOutputStream(file);
        //创建一个对象输出流，将对象序列化并写入到磁盘上
        ObjectOutputStream objectOutputStream = new ObjectOutputStream(outputStream);
        //将对象序列化并写入磁盘持久化
        objectOutputStream.writeObject(person);
        //记得关闭流
        objectOutputStream.flush();
        objectOutputStream.close();
        System.out.println("序列化成功");
        area.setText("序列化成功");
    }
    /**
     * 将person对象反序列化
     * @Title: DeserializePerson
     * 由于反序列化时会重新生成一个新的对象实例，
     * 这与单例模式和枚举类实现唯一性原则相违背，为了使它们不矛盾，必须修改反 序列化的流程
性
     * void
     * @author shimy
     * @throws IOException
     * @throws ClassNotFoundException
     * @since 2016-5-5 V 1.0
     */
    private void DeserializePerson() throws IOException, ClassNotFoundException{
        File file = new File(files, "person.txt");
        FileInputStream fileInputStream = new FileInputStream(file);
        ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);
        EXPersonSingle person = (EXPersonSingle) objectInputStream.readObject();
        area.setText(person.getAge() + "\n"
                +person.getName() + "\n"
                +person.getSex()+"\n"
                +person.getFujia());
        System.out.print("["+person.getAge() + "]["
                +person.getName() + "]["
                +person.getSex()+"]["
                +person.getFujia()+"]");
        objectInputStream.close();
    }
```

- 打印结果如下，可以看出代码的执行顺序，也可以看到并没有执行默认的序列化方法而是执行的person类里的writeExternal()和readExternal()方法里的序列化，在里面writeReplace和readResolve返回的是this，当前对象，我们可以在这里做手脚，张冠李戴，随意替换成其他对象。
- writeReplace和readResolve兼容单例和枚举
- writeReplace
- writeExternal
- 序列化成功
- readExternal
- readResolve
- [0][寂寞高手一时去无踪][null][小明(附加字段)]

七、序列化版本号serialVersionUID（主要用于版本控制）

摘自http://www.jb51.net/article/37145.htm

- 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；**

八、总结，需要注意的点

1. 序列化的概念，保存的是对象某一时刻的状态，将对象冻结在这一时刻，通过ObjectOutputStream将对象转换为二进制字节码然后以流的形式写入磁盘持久化。
2. 反序列化的概念，使用 ObjectInputStream将二进制字节码格式的对象以流的形式从磁盘读出并重构恢复。
3. 反序列化出来的对象并不是原来的对象了，而是已经完成了重构的对象实例。
4. Externalizable接口虽然继承了Serializable接口，但他们的序列化机制是完全不同的，Serializable的机制对于Externalizable是完全失效的。Externalizable反序列化时会重新实例化一个对象，然后将读出（重构）出来的对象赋予新实例化出来的对象。
5. writeReplace()和readResolve()方法实现序列化对象的偷换，可以用来对付单例和枚举的序列化。
6. writeReplace、readResolve、writeExternal、readExternal、writeObject、readObject是ObjectOutputStream或ObjectInputStream通过反射机制来调用的，反射总会先检查被序列化[illegible]这几个方法，存在，执行，不存在，执行默认。
7. 使用默认机制，在序列化对象时，不仅会序列化当前对象本身，还会对该对象引用的其它对化，同样地，这些其它对象引用的另外对象也将被序列化，以此类推。所以，如果一个对象量是容器类对象，而这些容器所含有的元素也是容器类对象，那么这个序列化的过程就会较复杂，开销也较大，所以这里的序列化细节就需要我们自己控制。
8. 至于其他请参看JAVA反序列化漏洞完整过程分析与调试和关于 Java 对象序列化您不知道的 5 件事

参考文章：

http://blog.sina.com.cn/s/blog_4e345ce70100rt86.html

http://drops.wooyun.org/papers/13244

http://www.ibm.com/developerworks/cn/java/j-5things1/

http://developer.51cto.com/art/201202/317181.htm

http://www.jb51.net/article/37145.htm

http://blog.csdn.net/wangloveall/article/details/7992448

http://www.cnblogs.com/xdp-gacl/p/3777987.html

顶 1 踩 0

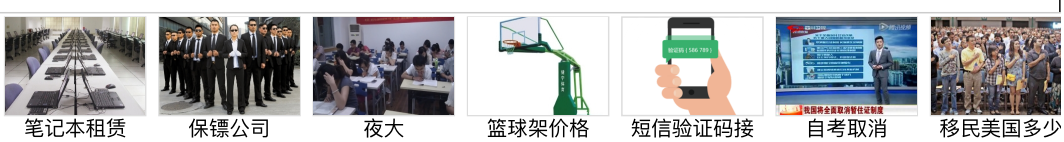
上一篇 短阶段总结

下一篇 eclipse 配置生成java头文件(JNI)

我的同类文章

java 笔记（9）

• 二进制的位数，字节、字符	2016-08-11 阅读 242	• 二进制 十进制 八进制 十六…	2016-08-08 阅读 54
• 对Rxjava1.0的map方法的源…	2016-07-15 阅读 417	• （error/warning）java Unsa…	2016-07-14 阅读 302
• java 输入输出流和File简单…	2016-05-04 阅读 161	• javaDoc和java文件的注释以…	2016-05-03 阅读 612
• 内存和File	2016-04-28 阅读 86	• 相对路径和绝对路径的理解…	2016-04-28 阅读 1914
• java Uri Url Urn android Uri…	2016-04-26 阅读 209		


笔记本租赁 保镖公司 夜大 篮球架价格 短信验证码接 自考取消 移民美国多少

参考知识库

Android知识库
33629 关注 | 2793 收录

Java 知识库
26202 关注 | 1457 收录

.NET知识库
3810 关注 | 836 收录

软件测试知识库
4582 关注 | 318 收录

Java SE知识库
25922 关注 | 479 收录

Java EE知识库
17945 关注 | 1324 收录

Git知识库
6530 关注 | 615 收录

猜你在找

Java基础核心技术：Java反射机制(day19-day20)
JAVA反射机制与单例模式
深入浅出Java的反射
Java基础核心技术：面向对象编程(day05-day07)
大数据培训（第一季） java基础

Java序列化和反序列化使用总结
JAVA面试整理--上
Java基础总结内部版
Google Protocol Buffer 的使用和原理
Google Protocol Buffer 的使用和原理


篮球架价格

希腊买房移民

团队拓展训练

笔记本租赁

保镖公司

app开发报价单

台球桌价格

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 | 江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2017, CSDN.NET, All Rights Reserved