链滴

序列化 | 谈一个不分手的对象

作者：douniwan
原文链接：https://ld246.com/article/1583336391194
来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20171107.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
“找不到对象，那就 new 一个吧！什么？你怕她记不住你们的点点滴滴？怕分手？怕撒
呀，有序列化啊！”
<h3 id="序列化">序列化</h3>
序列化是将对象转换为可传输格式的过程。 是一种数据的持久化手段。
<h4 id="Java中的序列化">Java 中的序列化</h4>
在 java 中我们可以创建一个对象。这个对象跟随程序的启动而存在，程序的结束而消亡。即对
的生命周期不会比 jvm 的生命周期更长。但在很多应用中，会要求 jvm 停止以后能够保存指定的对
，并在需要的时候重新读取被保存的对象。java 对象序列化就能够实现这一功能。
<h3 id="如何对Java对象进行序列化和反序列化">如何对 Java 对象进行序列化和反序列化</h3>
在 Java 中，一个类能被序列化的前提是，实现了 <code>java.io.Serializable</code> 接口（
序列化类的所有子类型本身都是可序列化的）。代码示例如下：
GirlFriend 类，用于序列化和反序列化
<pre><code class="language-java highlight-chroma"><span c
ass="highlight-cl">package <span class="highlight-nn
>geektomya;

<span class="high
ight-kn">import java.io.Serializable<span class
"highlight-o">;
<span class="high
ight-kn">import java.util.Date<span class="hi
hlight-o">;

<span class="high
ight-cm">/**
<span cla
s="highlight-cm"> * @author yaoqiuhong
<span cla
s="highlight-cm"> * @create 2020-02-28 16:01
<span cla
s="highlight-cm"> * @description
<span cla
s="highlight-cm"> */
<span class="high
ight-kd">public class <span class="highlight-n
">GirlFriend implements <span class="highligh
-n">Serializable {

    <span class="h
ghlight-kd">private String <span class="highlig
t-n">name;

    <span class="h
ghlight-kd">private Integer <span class="highli
ht-n">age;

    <span class="h
ghlight-kd">private transient <span class="hig
light-n">Date birthday<span class="highlight-o
>;

    <span class="h

ghlight-kd">private static <span class="highli
ht-kd">final long se
ialVersionUID = 86
9051849453497849L;

  <span class="h
ghlight-kd">public String <span class="highligh
-nf">getName() {</
pan>
  <span class=
highlight-k">return name<span class="highligh
-o">;
  <span class="h
ghlight-o">}

  <span class="h
ghlight-kd">public void <span class="highlight
nf">setName(String
/span> name) <span
class="highlight-o">{
  <span class=
highlight-k">this.n
me = name
span class="highlight-o">;
  <span class="h
ghlight-o">}

  <span class="h
ghlight-kd">public Integer <span class="highli
ht-nf">getAge() {</
pan>
  <span class=
highlight-k">return age<span class="highlight-
">;
  <span class="h
ghlight-o">}

  <span class="h
ghlight-kd">public void <span class="highlight
nf">setAge(Integer<
span> age) <span cl
ss="highlight-o">{
  <span class=
highlight-k">this.a
e = age<sp
n class="highlight-o">;
  <span class="h
ghlight-o">}

  <span class="h
ghlight-kd">public Date <span class="highlight
nf">getBirthday() {<
span>
  <span class=

highlight-k">return birthday<span class="highli
ht-o">;
    <span class="h
ghlight-o">}

    <span class="h
ghlight-kd">public void <span class="highlight
nf">setBirthday(Dat
 birthday) <
pan class="highlight-o">{
        <span class=
highlight-k">this.bi
thday = birthday</s
an>;
    <span class="h
ghlight-o">}

    <span class="h
ghlight-kd">public static <span class="highlig
t-kt">long getSerialVersionUID<span class="hi
hlight-o">() {
        <span class=
highlight-k">return serialVersionUID<span clas
="highlight-o">;
    <span class="h
ghlight-o">}

    <span class="h
ghlight-nd">@Override
    <span class="h
ghlight-kd">public String <span class="highligh
-nf">toString() {</s
an>
        <span class=
highlight-k">return "GirlFriend{" <span class="h
ghlight-o">+
                <span c
ass="highlight-s">"name='" + <span class="hig
light-n">name + '\'
 +
                <span c
ass="highlight-s">", age=" + <span class="highl
ght-n">age +
                <span c
ass="highlight-s">", birthday=" + <span class=
highlight-n">birthday +
                <span c
ass="highlight-sc">'}';
    <span class="h
ghlight-o">}
<span class="high
ight-o">}

</code></pre>

原文链接：序列化 | 谈一个不分手的对象

SerializationDemo 类，进行序列化
<pre><code class="language-java highlight-chroma"><span c
ass="highlight-cl">package <span class="highlight-nn
>geektomya;

<span class="high
ight-kn">import java.io.*<span class="highligh
-o">;
<span class="high
ight-kn">import java.util.Date<span class="hi
hlight-o">;

<span class="high
ight-cm">/**
<span cla
s="highlight-cm"> * @author yaoqiuhong
<span cla
s="highlight-cm"> * @create 2020-02-28 16:07
<span cla
s="highlight-cm"> * @description
<span cla
s="highlight-cm"> */
<span class="high
ight-kd">public class <span class="highlight-n
">SerializationDemo {

    <span class="h
ghlight-kd">public static <span class="highlig
t-kt">void main(</
pan>String[] <span c
ass="highlight-n">args) <span class="highlight
o">{
        <span class=
highlight-c1">// 创建一个对象用于序列化
<span cla
s="highlight-c1">   GirlFriend <span class="h
ghlight-n">girlFriend = <span class="highlight-
">new GirlFriend();<
span>
        <span class=
highlight-n">girlFriend.<span class="highlight-
a">setName("MyGirl
riend");
        <span class=
highlight-n">girlFriend.<span class="highlight-
a">setAge(18</sp
n>);
        <span class=
highlight-n">girlFriend.<span class="highlight-
a">setBirthday(new
/span> Date());
        <span class=
highlight-n">GirlFriend.<span class="highlight-

a">love = "geektom
a";
  <span class=
highlight-n">System.<span class="highlight-na
>out.println</span
("序列化前的对象："<
pan class="highlight-o">+girlFriend<span class
"highlight-o">+" static love :"<span class="highl
ght-o">+girlFriend.
/span>love);

  <span class=
highlight-c1">// 将girlFriend序列化
<span cla
s="highlight-c1"> try<span class="highlight
o">(ObjectOutputStream <span class="highlig
t-n">oos = new</sp
n> ObjectOutputStream(</s
an>new FileOutputStream</
pan>("girlFriend"<s
an class="highlight-o"))) {
  <span clas
="highlight-n">oos.<span class="highlight-na"
writeObject(girlFrien
);
  <span class=
highlight-o">} catch
IOException e</spa
>) {
  <span clas
="highlight-n">e.pr
ntStackTrace();
  <span class=
highlight-o">}

  <span class="h
ghlight-o">}
<span class="high
ight-o">}

<span class="high
ight-cm">/* output:

</code></pre>

SerializationDemo 类，进行反序列化

<pre><code class="language-java highlight-chroma"><span c
ass="highlight-cl">

</code></pre>

```
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl"><span class="high
ight-kn">import</span> <span class="highlight-nn">java.io.*</span><span class="highligh
-o">;</span>
</span></span><span class="highlight-line"><span class="highlight-cl"><span class="high
ight-kn">import</span> <span class="highlight-nn">java.util.Date</span><span class="hi
hlight-o">;</span>
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl"><span class="high
ight-cm">/**
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-cm"> * @author yaoqiuhong
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-cm"> * @create 2020-02-28 16:07
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-cm"> * @description
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-cm"> */</span>
</span></span><span class="highlight-line"><span class="highlight-cl"><span class="high
ight-kd">public</span> <span class="highlight-kd">class</span> <span class="highlight-n
">SerializationDemo</span> <span class="highlight-o">{</span>
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    <span class="h
ghlight-kd">public</span> <span class="highlight-kd">static</span> <span class="highlig
t-kt">void</span> <span class="highlight-nf">main</span><span class="highlight-o">(</
pan><span class="highlight-n">String</span><span class="highlight-o">[]</span> <span c
ass="highlight-n">args</span><span class="highlight-o">)</span> <span class="highlight
o">{</span>
</span></span><span class="highlight-line"><span class="highlight-cl">        <span class=
highlight-c1">// 将girlFriend反序列化
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-c1"></span>        <span class="highlight-n">FileInputStream</span> <span cl
ss="highlight-n">fis</span> <span class="highlight-o">=</span> <span class="highlight-k
">null</span><span class="highlight-o">;</span>
</span></span><span class="highlight-line"><span class="highlight-cl">        <span class=
highlight-k">try</span> <span class="highlight-o">{</span>
</span></span><span class="highlight-line"><span class="highlight-cl">            <span clas
="highlight-n">fis</span> <span class="highlight-o">=</span> <span class="highlight-k"
new</span> <span class="highlight-n">FileInputStream</span><span class="highlight-o">
</span><span class="highlight-s">"girlFriend"</span><span class="highlight-o">);</span>
</span></span><span class="highlight-line"><span class="highlight-cl">        <span class=
highlight-o">}</span> <span class="highlight-k">catch</span> <span class="highlight-o">
</span><span class="highlight-n">FileNotFoundException</span> <span class="highlight-
">e</span><span class="highlight-o">)</span> <span class="highlight-o">{</span>
</span></span><span class="highlight-line"><span class="highlight-cl">            <span clas
="highlight-n">e</span><span class="highlight-o">.</span><span class="highlight-na">pr
ntStackTrace</span><span class="highlight-o">();</span>
</span></span><span class="highlight-line"><span class="highlight-cl">        <span class=
highlight-o">}</span>
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">        <span class=
highlight-k">try</span><span class="highlight-o">(</span><span class="highlight-n">Obj
ctInputStream</span> <span class="highlight-n">ois</span> <span class="highlight-o">=
/span> <span class="highlight-k">new</span> <span class="highlight-n">ObjectInputStre
```

m(fis)){
            GirlFriend myGirlFriend <span cl ss="highlight-o">= (<span class="highlight-n" GirlFriend)ois</spa >.readObject();
            System.out.println</sp n>("反序列化的对象："</span +myGirlFriend+" static love :"<span class "highlight-o">+myGirlFriend.love);</sp n>
        <span class= highlight-o">}catch IOException e</spa >){
            e.pr ntStackTrace();
        }catch ClassNotFoundException e){
            e.pr ntStackTrace();
        }
    }
}
/* output
* 反序列化的对象：GirlFriend{name='MyGirlFriend', age=18, birthday=null}
* static love :Me
* */
</code></pre>
<h4 id="输出对比">输出对比</h4>
<code>girlFriend </code> 对象被序列化的时候的值与反序列化得到的对象 <code>myGirlFri nd</code> 的值进行对比可以发现：<code>birthday</code> 字段的值和 <code>love</code> 字段的值不一样。还有
之所以会出现这个原因是因为在 GirlFriend 类中 <code>birthday</code> 被 <code>transient</ ode> 修饰，而 <code>love</code> 被 <code>static</code> 修饰。

Transient 关键字的作用是控制变量的序列化，在变量声明前加上该关键字，可以阻止该变量被

列化到文件中，在被反序列化后，transient 变量的值被设为初始值，如 int 型的是 0，对象型的是 nul
。
static 关键字修饰的变量属于类变量，而序列化是将对象的状态进行序列化，所以序列化的时候
不会将 static 修饰的字段进行序列化。

<h4 id="总结">总结</h4>

在 Java 中，只要一个类实现了 <code>java.io.Serializable</code> 接口，那么它就可以被序
化

通过 <code>ObjectOutputStream</code> 和 <code>ObjectInputStream</code> 对对
进行序列化及反序列化

虚拟机是否允许反序列化，不仅取决于类路径和功能代码是否一致，一个非常重要的一点是两个
的序列化 ID 是否一致（就是 <code>private static final long serialVersionUID</code>），如果
序列化时两个类的 serialVersionUID 不一致，那么就会失败。

序列化并不保存静态变量。

Transient 关键字的作用是控制变量的序列化，在变量声明前加上该关键字，可以阻止该变量被
列化到文件中

反序列化后对象的全类名（包名 + 类名）需要和序列化之前对象的全类名一致

<h3 id="如何实现自定义序列化">如何实现自定义序列化</h3>
<h4 id="1-被序列化的类中增加writeObject-和-readObject方法">1.被序列化的类中增加 writeObj
ct 和 readObject 方法</h4>
<blockquote>
在序列化和反序列化过程中，如果被序列化的类中定义了 writeObject 和 readObject 方法，虚
机会试图调用对象类里的 writeObject 和 readObject 方法，进行用户自定义的序列化和反序列化。<
p>
如果没有这样的方法，则默认调用是 ObjectOutputStream 的 defaultWriteObject 方法以及 O
jectInputStream 的 defaultReadObject 方法。
</blockquote>
<h5 id="ArrayList的自定义序列化">ArrayList 的自定义序列化</h5>
在 ArrayList 中，他的定义如下：
<pre><code class="language-java highlight-chroma"><span c
ass="highlight-cl">public c
ass ArrayList<</s
an>E> <span clas
="highlight-kd">extends AbstractList<span clas
="highlight-o"><E<span class="highlight-o"
>
  <span class=
highlight-kd">implements List<span class="high
ight-o"><E<span class="highlight-o">>,</
pan> RandomAccess,</span

Cloneable, <span cl
ss="highlight-n">java.<span class="highlight-n
">io.Serializable</
pan>
<span class="high
ight-o">{
    <span class="h
ghlight-kd">private static <span class="highli
ht-kd">final long se
ialVersionUID = 86
3452581122892189L;
    <span class="h
ghlight-kd">transient Object<span class="highl
ght-o">[] elementData<span class="highlight-o
>; // non-private to simplify nested class access
<span cla
s="highlight-c1">   private <span class="highl
ght-kt">int size;</s
an>
    <span class="h
ghlight-cm">/*
<span cla
s="highlight-cm">   * 省略其他成员
<span cla
s="highlight-cm">   */
<span class="high
ight-o">}
</code></pre>
其中可以看到 <code>elementData</code>（就是用来保存列表中的元素的）被 <code>tran
ient</code> 修饰，那么意味着在 ArrayList 中的元素不能被序列化。然而事实并不这样</
trong>，ArrayList 中的元素能被序列化，因为在 ArrayList 中自定义了 writeObject 和 re
dObject 方法。
那么为什么 ArrayList 要 transient 后再自定义序列方式呢？

ArrayList 实际上是动态数组，每次在放满以后自动增长设定的长度值，如果数组自动增长长度
为 100，而实际只放了一个元素，那就会序列化 99 个 null 元素。为了保证在序列化的时候不会将这
多 null 同时进行序列化，ArrayList 把元素数组设置为 transient。
为了防止一个包含大量空对象的数组被序列化，为了优化存储，所以，ArrayList 使用 <code>tr
nsient</code> 来声明 <code>elementData</code>。 但是，作为一个集合，在序列化过程中还
须保证其中的元素可以被持久化下来，所以，通过重写 <code>writeObject</code> 和 <code>re
dObject</code> 方法的方式把其中的元素保留下来。

<h5 id="writeObject-和-readObject如何被调用的-">writeObject 和 readObject 如何被调用的？
/h5>
都是通过 ObjectXxxxxStream 来调用，以 ObjectInputStream 为例，图示如下：
<img src="https://ld246.com/images/img-loading.svg" alt="20170927193034672.png" data-s
c="https://b3logfile.com/file/2020/03/20170927193034672-2f9a7616.png?imageView2/2/inte
lace/1/format/jpg">
也就是调用栈为：
<code>readObject ---> readObject0 --->readOrdinaryObject--->readSerialData---
gt;invokeReadObject</code>
其中在这里发挥作用的是：<code>invokeReadObject</code>,其中 <code>readObjectMethod.
nvoke(obj, new Object[]{ in });</code> 是关键，通过反射的方式调用 readObjectMethod 方法。
方是这么解释这个 readObjectMethod 的：

```
<blockquote>
<p>class-defined readObject method, or null if none</p>
</blockquote>
<p><strong>结论</strong>：通过反射自动调用的。</p>
<h4 id="2-被序列化的类中增加增加writeReplace和readResolve方法-">2.被序列化的类中增加增加
writeReplace 和 readResolve 方法。</h4>
<blockquote>
<p>在序列化和反序列化过程中，如果被序列化的类中定义了 writeReplace 和 readResolve 方法，
么在<strong>序列化过程中就会调用 writeReplace</strong> 方法，实际序列化得到的对象将是 wr
teReplace 方法返回值的对象；在<strong>反序列化过程中就会调用 readResolve</strong> 方法
实际反序列化得到对象将是作为 readResolve 方法返回值的对象。</p>
</blockquote>
<h5 id="writeReplace">writeReplace</h5>
<p>这里需要注意的是 writeReplace 中返回的对象<strong>如果不是本类对象</strong>，那么<s
rong>返回的对象那个类</strong>也应该实现 <code>java.io.Serializable</code> 接口，然后在
序列化的时候，需要用<strong>返回的对象那个类来接收</strong>。</p>
<ul>
<li>如果<strong>返回的对象那个类</strong>没有实现 <code>java.io.Serializable</code> 接
，那么序列化的时候将会报 <code>java.io.NotSerializableException</code>。</li>
<li>如果反序列化的时候，没有用<strong>返回的对象那个类来接收</strong>，那么反序列化就
报 <code>java.lang.ClassCastException</code>。</li>
<li>代码示例如下：<br>
writeReplace 方法中返回其他类对象的 Demo1 类</li>
</ul>
<pre><code class="language-java highlight-chroma"><span class="highlight-line"><span c
ass="highlight-cl"><span class="highlight-kn">package</span> <span class="highlight-nn
>geektomya</span><span class="highlight-o">;</span>
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl"><span class="high
ight-kn">import</span> <span class="highlight-nn">java.io.Serializable</span><span class
"highlight-o">;</span>
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl"><span class="high
ight-cm">/**
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-cm"> * @author yaoqiuhong
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-cm"> * @create 2020-02-28 21:52
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-cm"> * @description
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-cm"> */</span>
</span></span><span class="highlight-line"><span class="highlight-cl"><span class="high
ight-kd">public</span> <span class="highlight-kd">class</span> <span class="highlight-n
">Demo1</span> <span class="highlight-kd">implements</span> <span class="highlight-
">Serializable</span> <span class="highlight-o">{</span>
</span></span><span class="highlight-line"><span class="highlight-cl">    <span class="h
ghlight-kd">public</span> <span class="highlight-n">String</span> <span class="highligh
-n">Demo1_Name</span><span class="highlight-o">;</span>
</span></span><span class="highlight-line"><span class="highlight-cl">    <span class="h
ghlight-kd">private</span> <span class="highlight-kd">static</span> <span class="highli
ht-kd">final</span> <span class="highlight-kt">long</span> <span class="highlight-n">se
ialVersionUID</span> <span class="highlight-o">=</span> <span class="highlight-mi">47
1659151754920854L</span><span class="highlight-o">;</span>
```

    <span class="h
ghlight-kd">public Object <span class="highlig
t-nf">writeReplace(){
        <span class=
highlight-k">return new <span class="highlight
n">Demo2();
    <span class="h
ghlight-o">}
    <span class="hi
hlight-c1">// 省略的getter and setter
<span cla
s="highlight-c1">}

</code></pre>
writeReplace 方法中返回的 Demo2 类
<pre><code class="language-java highlight-chroma"><span c
ass="highlight-cl">package <span class="highlight-nn
">geektomya;

<span class="high
ight-kn">import java.io.Serializable<span class
"highlight-o">;

<span class="high
ight-cm">/**
<span cla
s="highlight-cm"> * @author yaoqiuhong
<span cla
s="highlight-cm"> * @create 2020-02-28 21:53
<span cla
s="highlight-cm"> * @description 实现了Serializable接口
<span cla
s="highlight-cm"> */
<span class="high
ight-kd">public class <span class="highlight-n
">Demo2 implements <span class="highlight
n">Serializable {

<span class="high
ight-o">}
</code></pre>
测试类
<pre><code class="language-go highlight-chroma">
<span cla
s="highlight-cl">package
eektomya;

<span class="high
ight-kn">import java<span class="highlight-p
>.io.<span
class="highlight-o">*;

<span class="high
ight-cm">/**

<span cla

```
s="highlight-cm"> * @author yaoqiuhong
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-cm"> * @create 2020-02-28 22:00
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-cm"> * @description
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-cm"> */</span>
</span></span><span class="highlight-line"><span class="highlight-cl"><span class="high
ight-nx">public</span> <span class="highlight-nx">class</span> <span class="highlight-n
">SerializableDemo2</span><span class="highlight-p">{</span>
</span></span><span class="highlight-line"><span class="highlight-cl">    <span class="h
ghlight-c1">//为了便于理解，忽略关闭流操作及删除文件操作。真正编码时千万不要忘记
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-c1"></span>    <span class="highlight-c1">//Exception直接抛出
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-c1"></span>    <span class="highlight-nx">public</span> <span class="highli
ht-nx">static</span> <span class="highlight-nx">void</span> <span class="highlight-nf"
main</span><span class="highlight-p">(</span><span class="highlight-nx">String</span
<span class="highlight-p">[]</span> <span class="highlight-nx">args</span><span class=
highlight-p">)</span>  <span class="highlight-nx">throws</span> <span class="highlight-
x">FileNotFoundException</span><span class="highlight-p">,</span> <span class="highli
ht-nx">IOException</span> <span class="highlight-p">{</span>
</span></span><span class="highlight-line"><span class="highlight-cl">        <span class=
highlight-nx">Demo1</span> <span class="highlight-nx">demo1</span> <span class="hi
hlight-p">=</span> <span class="highlight-nx">new</span> <span class="highlight-nf">D
mo1</span><span class="highlight-p">();</span>
</span></span><span class="highlight-line"><span class="highlight-cl">        <span class=
highlight-nx">demo1</span><span class="highlight-p">.</span><span class="highlight-nf
>setDemo1_Name</span><span class="highlight-p">(</span><span class="highlight-s">"
emo1"</span><span class="highlight-p">);</span>
</span></span><span class="highlight-line"><span class="highlight-cl">        <span class=
highlight-c1">// 序列化
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-c1"></span>        <span class="highlight-nx">ObjectOutputStream</span> <s
an class="highlight-nx">oos</span> <span class="highlight-p">=</span> <span class="hi
hlight-nx">new</span> <span class="highlight-nf">ObjectOutputStream</span><span clas
="highlight-p">(</span><span class="highlight-nx">new</span> <span class="highlight-nf
>FileOutputStream</span><span class="highlight-p">(</span><span class="highlight-s">"
emo1"</span><span class="highlight-p">));</span>
</span></span><span class="highlight-line"><span class="highlight-cl">        <span class=
highlight-c1">// 由于Demo1类中的writeReplace方法返回Demo2对象，如果Demo2没实现Seriali
able接口，将会抛出NotSerializableException
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-c1"></span>        <span class="highlight-nx">oos</span><span class="highli
ht-p">.</span><span class="highlight-nf">writeObject</span><span class="highlight-p">(
/span><span class="highlight-nx">demo1</span><span class="highlight-p">);</span>
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">        <span class=
highlight-c1">// 反序列化
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-c1"></span>        <span class="highlight-nx">ObjectInputStream</span> <sp
n class="highlight-nx">ois</span> <span class="highlight-p">=</span> <span class="highl
ght-nx">new</span> <span class="highlight-nf">ObjectInputStream</span><span class="h
ghlight-p">(</span><span class="highlight-nx">new</span> <span class="highlight-nf">Fi
```

eInputStream("demo
"

  <span class=
highlight-nx">try {
  <span clas
="highlight-cm">/**
<span cla
s="highlight-cm"> * 此处会抛出ClassCastException，因为前面的序列化得到的对象是De
o2
<span cla
s="highlight-cm"> * 此处正确的写法是：Demo2 demo22 = (Demo2) ois.readObject();
<span cla
s="highlight-cm"> * */
  <span clas
="highlight-nx">Demo1 demo11 <span class=
highlight-p">= (D
mo1) ois<
pan class="highlight-p">.readObject<span clas
="highlight-p">();
  <span class=
highlight-p">} catch <span class="highlight-p"
(ClassNotFoundException <span class="highlig
t-nx">e) {
  <span clas
="highlight-nx">System.<span class="highlight
nx">out.println</s
an>("异常错误java.lang.Class
otFoundException");
  <span class=
highlight-p">}
  <span class="h
ghlight-p">}
<span class="high
ight-p">}
</code></pre>
<h4 id="readResolve">readResolve</h4>
readResolve 注意这里如果返回的对象不是本类的对象，那么再反序列化
时候，也需要用与 readResolve 返回的对象一直的类对象来接受，否则同样会出现 <code>java.lang
ClassCastException</code>
<h4 id="3-通过实现Externalizable接口-">3.通过实现 Externalizable 接口。</h4>
Externalizable 继承了 Serializable，该接口中定义了两个抽象方法：writeExternal()与 readExte
nal()。当使用 Externalizable 接口来进行序列化与反序列化的时候需要开发人员重写 writeExternal(
与 readExternal()方法。
所以可以在 writeExternal()与 readExternal()方法中进行自定义的序列化和反序列化。

如果没有在这两个方法中定义序列化实现细节，那么序列化之后，对象内容为空。
类必须存在一个 pulic 的无参数构造方法
反序列化时的字段属性需要与序列化时一致，否则值顺序错乱。(Serializable 中如果定义了 <co
e>serialVersionUID</code> 且一直的话能顾自动兼容自动的增减)

<h3 id="为什么实现了Serializable接口就可以序列化">为什么实现了 Serializable</stro
g> 接口就可以序列化</h3>
<blockquote>

Serializable 明明就是一个空的接口，它是怎么保证只有实现了该接口的方法才能进行序列化与序列化的呢？**
这个问题可以从 ObjectOutputStream 的 writeObject 的调用栈来回答：
</blockquote>
<code>writeObject ---> writeObject0 --->writeOrdinaryObject--->writeSerialDa
a--->invokeWriteObject</code>
writeObject0 方法中有这么一段代码：
<pre><code class="language-java highlight-chroma"><span c
ass="highlight-cl">if (</span
obj instanceof <span
class="highlight-n">String) <span class="highli
ht-o">{
 <span c
ass="highlight-n">writeString((<span class="hi
hlight-n">String) ob
, unshared<
pan class="highlight-o">);
  <span clas
="highlight-o">} else <span class="highlight-k"
if (cl<span c
ass="highlight-o">.isArray<span class="highli
ht-o">()) {
  <span c
ass="highlight-n">writeArray(<span class="high
ight-n">obj, desc</
pan>, unshared<sp
n class="highlight-o">);
  <span clas
="highlight-o">} else <span class="highlight-k"
if (obj <spa
class="highlight-k">instanceof Enum<span clas
="highlight-o">) {
  <span c
ass="highlight-n">writeEnum((<span class="hi
hlight-n">Enum<?>) <span class="highligh
-n">obj, desc</spa
>, unshared<span cl
ss="highlight-o">);
  <span clas
="highlight-o">} else <span class="highlight-k"
if (obj <spa
class="highlight-k">instanceof Serializable<sp
n class="highlight-o">) {
  <span c
ass="highlight-n">writeOrdinaryObject(<span c
ass="highlight-n">obj, <span class="highlight-n
>desc, unshared</s
an>);
  <span clas
="highlight-o">} else <span class="highlight-o"
{
  <span c
ass="highlight-k">if (<span class="highlight-n"
extendedDebugInfo) <span class="highlight-o"

{
  throw new NotSerializableException(
  cl.getName() + "\n" + debugInfoStack.toString());
  } else {
  throw new NotSerializableException(cl.getName());
  }
  }
</code></pre>
在进行序列化操作时，会判断要被序列化的类是否是 Enum、Array 和 Serializable 类型，如果是则直接抛出 <code>NotSerializableException</code>。
<hr>
本文参考
深入分析 Java 的序列化与反序列化-Hollishuang's Blog
序列化与自定义序列
JAVA 对象流序列化时的eadObject，writeObject，readResolve 是怎么被调用的