

## java 中的 IO 流操作大全

### 一、流的概念

流(stream)的概念源于 UNIX 中管道(pipe)的概念。在 UNIX 中，管道是一条不间断的字节流，用来实现程序或进程间的通信，或读写外围设备、外部文件等。

一个流，必有源端和目的端，它们可以是计算机内存的某些区域，也可以是磁盘文件，甚至可以是 Internet 上的某个 URL。

流的方向是重要的，根据流的方向，流可分为两类：输入流和输出流。用户可以从输入流中读取信息，但不能写它。相反，对输出流，只能往输入流写，而不能读它。

实际上，流的源端和目的端可简单地看成是字节的生产者和消费者，对输入流，可不必关心它的源端是什么，只要简单地从流中读数据，而对输出流，也可不知道它的目的端，只是简单地往流中写数据。

形象的比喻——水流，文件=====程序，文件和程序之间连接一个管道，水流就在之间形成了，自然也就出现了方向：可以流进，也可以流出。便于理解，这么定义流：流就是一个管道里面有流水，这个管道连接了文件和程序。

### 二、流的分类

1. java.io 包中的类对应两类流，一类流直接从指定的位置（如磁盘文件或内存区域）读或写，这类流称为**结点流(node stream)**，其它的流则称为**过滤器(filters)**。过滤器输入流往往是以其它输入流作为它的输入源，经过过滤或处理后再以新的输入流的形式提供给用户，过滤器输出流的原理也类似。
2. Java 的常用输入、输出流

java.io 包中的 stream 类根据它们操作对象的类型是字符还是字节可分为两大类：字符流和字节流。

- Java 的字节流

InputStream 是所有字节输入流的祖先，而 OutputStream 是所有字节输出流的祖先。

- Java 的字符流

Reader 是所有读取字符串输入流的祖先，而 writer 是所有输出字符串的祖先。

结合开始所说的输入/输出流，出现了个一小框架。

	字节流	字符流
输入流	InputStream	Reader

## 输出流      **OutputStream**      **Writer**

### 【案例 1】创建一个新文件

[java] view plaincopy

```
1. import java.io.*;
2. class hello{
3.     public static void main(String[] args) {
4.         File f=new File("D:\\hello.txt");
5.         try{
6.             f.createNewFile();
7.         }catch (Exception e) {
8.             e.printStackTrace();
9.         }
10.    }
11. }
```

#### 【运行结果】:

程序运行之后，在 d 盘下会有一个名字为 `hello.txt` 的文件。

### 【案例 2】File 类的两个常量

[java] view plaincopy

```
1. import java.io.*;
2. class hello{
3.     public static void main(String[] args) {
4.         System.out.println(File.separator);
5.         System.out.println(File.pathSeparator);
6.     }
7. }
```

#### 【运行结果】:

\  
;

此处多说几句：有些同学可能认为，我直接在 windows 下使用\进行分割不行吗？当然是可以的。但是在 linux 下就不是\了。所以，要想使得我们的代码跨平台，更加健壮，所以，大家都采用这两个常量吧，其实也多写不了几行。呵呵、

现在我们使用 File 类中的常量改写上面的代码：

[java] view plaincopy

```
1. import java.io.*;
2. class hello{
3.     public static void main(String[] args) {
4.         String fileName="D:"+File.separator+"hello.txt";
```

```
5.         File f=new File(fileName);
6.         try{
7.             f.createNewFile();
8.         }catch (Exception e) {
9.             e.printStackTrace();
10.        }
11.    }
12. }
```

你看，没有多写多少吧，呵呵。所以建议使用 `File` 类中的常量。

### 删除一个文件

[java] view plaincopy

```
1.  /**
2.   * 删除一个文件
3.   * */
4.  import java.io.*;
5.  class hello{
6.      public static void main(String[] args) {
7.          String fileName="D:"+File.separator+"hello.txt";
8.          File f=new File(fileName);
9.          if(f.exists()){
10.             f.delete();
11.          }else{
12.              System.out.println("文件不存在");
13.          }
14.
15.      }
16. }
```

### 创建一个文件夹

[java] view plaincopy

```
1.  /**
2.   * 创建一个文件夹
3.   * */
4.  import java.io.*;
5.  class hello{
6.      public static void main(String[] args) {
7.          String fileName="D:"+File.separator+"hello";
8.          File f=new File(fileName);
9.          f.mkdir();
```

```
10.    }  
11. }
```

【运行结果】:

D 盘下多了一个 hello 文件夹

列出指定目录的全部文件（包括隐藏文件）：

[java] view plaincopy

```
1.  /**  
2.  * 使用 list 列出指定目录的全部文件  
3.  * */  
4.  import java.io.*;  
5.  class hello{  
6.      public static void main(String[] args) {  
7.          String fileName="D:"+File.separator;  
8.          File f=new File(fileName);  
9.          String[] str=f.list();  
10.         for (int i =0; i < str.length; i++) {  
11.             System.out.println(str[i]);  
12.         }  
13.     }  
14. }
```

【运行结果】:

\$RECYCLE.BIN

360

360Downloads

360Rec

360SoftMove

Config.Msi

da

Downloads

DriversBackup

eclipse

java web 整合开发和项目实战

Lenovo

MSOCCache

Program

Program Files

python

RECYGLER.{8F92DA15-A229-A4D5-B5CE-5280C8B89C19}

System Volume Information

Tomcat6

var

vod\_cache\_data

新建文件夹

（你的运行结果应该和这个不一样的，呵呵）

但是使用 `list` 返回的是 `String` 数组，。而且列出的不是完整路径，如果想列出完整路径的话，需要使用 `listFiles`。他返回的是 `File` 的数组

列出指定目录的全部文件（包括隐藏文件）：

[java] view plaincopy

```
1. /**
2.  * 使用 listFiles 列出指定目录的全部文件
3.  * listFiles 输出的是完整路径
4.  * */
5. import java.io.*;
6. class hello{
7.     public static void main(String[] args) {
8.         String fileName="D:"+File.separator;
9.         File f=new File(fileName);
10.        File[] str=f.listFiles();
11.        for (int i =0; i < str.length; i++) {
12.            System.out.println(str[i]);
13.        }
14.    }
15. }
```

【运行结果】:

D:\\$RECYCLE.BIN

D:\360

D:\360Downloads

D:\360Rec

D:\360SoftMove

D:\Config.Msi

D:\da

D:\Downloads

D:\DriversBackup

D:\eclipse  
D:\java web 整合开发和项目实战  
D:\Lenovo  
D:\MSOCache  
D:\Program  
D:\Program Files  
D:\python  
D:\RECYGLER.{8F92DA15-A229-A4D5-B5CE-5280C8B89C19}  
D:\System Volume Information  
D:\Tomcat6  
D:\var  
D:\vod\_cache\_data  
D:\新建文件夹

通过比较可以指定，使用 `listFiles` 更加方便、

### 判断一个指定的路径是否为目录

[java] view plaincopy

```
1. /**
2.  * 使用 isDirectory 判断一个指定的路径是否为目录
3.  * */
4. import java.io.*;
5. class hello{
6.     public static void main(String[] args) {
7.         String fileName="D:"+File.separator;
8.         File f=new File(fileName);
9.         if(f.isDirectory()){
10.             System.out.println("YES");
11.         }else{
12.             System.out.println("NO");
13.         }
14.     }
15. }
```

【运行结果】: YES

### 搜索指定目录的全部内容

[java] view plaincopy

```
1. /**
```

```
2.  * 列出指定目录的全部内容
3.  * */
4.  import java.io.*;
5.  class hello{
6.      public static void main(String[] args) {
7.          String fileName="D:"+File.separator;
8.          File f=new File(fileName);
9.          print(f);
10.     }
11.     public static void print(File f){
12.         if(f!=null){
13.             if(f.isDirectory()){
14.                 File[] fileArray=f.listFiles();
15.                 if(fileArray!=null){
16.                     for (inti = 0; i < fileArray.length; i++) {
17.                         //递归调用
18.                         print(fileArray[i]);
19.                     }
20.                 }
21.             }
22.             else{
23.                 System.out.println(f);
24.             }
25.         }
26.     }
27. }
```

【运行结果】:

D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\framepages\web4welcome\_jsp.  
java  
D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\help\_005fhome\_jsp.class  
D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\help\_005fhome\_jsp.java  
D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\home\_jsp.class  
D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\home\_jsp.java  
D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\index\_jsp.class  
D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\index\_jsp.java  
D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\login\_jsp.class  
D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\login\_jsp.java  
D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\modify\_005fuser\_005finfo\_jsp.  
class

D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\modify\_005fuser\_005finfo\_jsp.j  
ava  
D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\register\_005fnotify\_jsp.class  
D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\register\_005fnotify\_jsp.java  
D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\sign\_005fup\_jsp.class  
D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\sign\_005fup\_jsp.java  
D:\Tomcat6\work\Catalina\localhost\nevel\org\apache\jsp\transit\_jsp.class  
.....

### 【使用 RandomAccessFile 写入文件】

[java] view plaincopy

```
1. /**
2.  * 使用 RandomAccessFile 写入文件
3.  * */
4. import java.io.*;
5. class hello{
6.     public static void main(String[] args) throws IOException {
7.         String fileName="D:"+File.separator+"hello.txt";
8.         File f=new File(fileName);
9.         RandomAccessFile demo=new RandomAccessFile(f,"rw");
10.        demo.writeBytes("asdsad");
11.        demo.writeInt(12);
12.        demo.writeBoolean(true);
13.        demo.writeChar('A');
14.        demo.writeFloat(1.21f);
15.        demo.writeDouble(12.123);
16.        demo.close();
17.    }
18. }
```

如果你此时打开 hello.txt 查看的话，会发现那是乱码。

## 字节流

### 【向文件中写入字符串】

[java] view plaincopy

```
1. /**
2.  * 字节流
3.  * 向文件中写入字符串
```



```
4.  * */
5.  import java.io.*;
6.  class hello{
7.      public static void main(String[] args) throws IOException {
8.          String fileName="D:"+File.separator+"hello.txt";
9.          File f=new File(fileName);
10.         OutputStream out =new FileOutputStream(f);
11.         String str="你好";
12.         byte[] b=str.getBytes();
13.         out.write(b);
14.         out.close();
15.     }
16. }
```

查看 hello.txt 会看到“你好”

当然也可以一个字节一个字节的写。

[java] view plaincopy

```
1.  /**
2.   * 字节流
3.   * 向文件中一个字节一个字节的写入字符串
4.   * */
5.  import java.io.*;
6.  class hello{
7.      public static void main(String[] args) throws IOException {
8.          String fileName="D:"+File.separator+"hello.txt";
9.          File f=new File(fileName);
10.         OutputStream out =new FileOutputStream(f);
11.         String str="你好";
12.         byte[] b=str.getBytes();
13.         for (int i =0; i < b.length; i++) {
14.             out.write(b[i]);
15.         }
16.         out.close();
17.     }
18. }
```

结果还是：“你好”

向文件中追加新内容：

[java] view plaincopy

```
1.  /**
```

```
2.  * 字节流
3.  * 向文件中追加新内容:
4.  * */
5.  import java.io.*;
6.  class hello{
7.      public static void main(String[] args) throws IOException {
8.          String fileName="D:"+File.separator+"hello.txt";
9.          File f=new File(fileName);
10.         OutputStream out =new FileOutputStream(f,true);
11.         String str="Rollen";
12.         //String str="\r\nRollen"; 可以换行
13.         byte[] b=str.getBytes();
14.         for (int i =0; i < b.length; i++) {
15.             out.write(b[i]);
16.         }
17.         out.close();
18.     }
19. }
```

#### 【运行结果】:

你好 Rollen

#### 【读取文件内容】

[java] view plaincopy

```
1.  /**
2.  * 字节流
3.  * 读文件内容
4.  * */
5.  import java.io.*;
6.  class hello{
7.      public static void main(String[] args) throws IOException {
8.          String fileName="D:"+File.separator+"hello.txt";
9.          File f=new File(fileName);
10.         InputStream in=new FileInputStream(f);
11.         byte[] b=new byte[1024];
12.         in.read(b);
13.         in.close();
14.         System.out.println(new String(b));
15.     }
16. }
```

#### 【运行结果】

你好 Rollen

Rollen\_

但是这个例子读取出来会有大量的空格，我们可以利用 `in.read(b);` 的返回值来设计程序。如下：

[java] view plaincopy

```
1. /**
2.  * 字节流
3.  * 读文件内容
4.  * */
5. import java.io.*;
6. class hello{
7.     public static void main(String[] args) throws IOException {
8.         String fileName="D:"+File.separator+"hello.txt";
9.         File f=new File(fileName);
10.        InputStream in=new FileInputStream(f);
11.        byte[] b=new byte[1024];
12.        int len=in.read(b);
13.        in.close();
14.        System.out.println("读入长度为: "+len);
15.        System.out.println(new String(b,0,len));
16.    }
17. }
```

【运行结果】:

读入长度为: 18

你好 Rollen

Rollen

读者观察上面的例子可以看出，我们预先申请了一个指定大小的空间，但是有时候这个空间可能太小，有时候可能太大，我们需要准确的大小，这样节省空间，那么我们可以这样干：

[java] view plaincopy

```
1. /**
2.  * 字节流
3.  * 读文件内容,节省空间
4.  * */
5. import java.io.*;
6. class hello{
7.     public static void main(String[] args) throws IOException {
8.         String fileName="D:"+File.separator+"hello.txt";
9.         File f=new File(fileName);
```

```
10.      InputStream in=new FileInputStream(f);
11.      byte[] b=new byte[(int)f.length()];
12.      in.read(b);
13.      System.out.println("文件长度为: "+f.length());
14.      in.close();
15.      System.out.println(new String(b));
16.  }
17. }
```

文件长度为: 18

你好 Rollen

Rollen

将上面的例子改为一个一个读:

[java] view plaincopy

```
1.  /**
2.   * 字节流
3.   * 读文件内容,节省空间
4.   * */
5.  import java.io.*;
6.  class hello{
7.      public static void main(String[] args) throws IOException {
8.          String fileName="D:"+File.separator+"hello.txt";
9.          File f=new File(fileName);
10.         InputStream in=new FileInputStream(f);
11.         byte[] b=new byte[(int)f.length()];
12.         for (int i =0; i < b.length; i++) {
13.             b[i]=(byte)in.read();
14.         }
15.         in.close();
16.         System.out.println(new String(b));
17.     }
18. }
```

输出的结果和上面的一样。

细心的读者可能会发现,上面的几个例子都是在知道文件的内容多大,然后才展开的,有时候我们不知道文件有多大,这种情况下,我们需要判断是否读到文件的末尾。

[java] view plaincopy

```
1.  /**
```

```
2.  * 字节流
3.  *读文件
4.  * */
5.  import java.io.*;
6.  class hello{
7.      public static void main(String[] args) throws IOException {
8.          String fileName="D:"+File.separator+"hello.txt";
9.          File f=new File(fileName);
10.         InputStream in=new FileInputStream(f);
11.         byte[] b=new byte[1024];
12.         int count =0;
13.         int temp=0;
14.         while((temp=in.read())!=(-1)){
15.             b[count++]=(byte)temp;
16.         }
17.         in.close();
18.         System.out.println(new String(b));
19.     }
20. }
```

### 【运行结果】

你好 Rollen

Rollen\_

提醒一下，当读到文件末尾的时候会返回-1.正常情况下是不会返回-1 的

## 字符流

### 【向文件中写入数据】

现在我们使用字符流

[\[java\] view plaincopy](#)

```
1.  /**
2.  * 字符流
3.  * 写入数据
4.  * */
5.  import java.io.*;
6.  class hello{
7.      public static void main(String[] args) throws IOException {
8.          String fileName="D:"+File.separator+"hello.txt";
9.          File f=new File(fileName);
10.         Writer out =new FileWriter(f);
11.         String str="hello";
```

```
12.         out.write(str);
13.         out.close();
14.     }
15. }
```

当你打开 **hello.txt** 的时候，会看到 **hello**

其实这个例子上之前的例子没什么区别，只是你可以直接输入字符串，而不需要你将字符串转化为字节数组。

当你如果想向文件中追加内容的时候，可以使用将上面的声明 **out** 的哪一行换为：

**Writer out =new FileWriter(f,true);**

这样，当你运行程序的时候，会发现文件内容变为：

**hellohello** 如果想在文件中换行的话，需要使用“**\r\n**”

比如将 **str** 变为 **String str="\r\nhello";**

这样文件追加的 **str** 的内容就会换行了。

从文件中读内容：

[java] view plain copy

```
1.  /**
2.   * 字符流
3.   * 从文件中读出内容
4.   * */
5.  import java.io.*;
6.  class hello{
7.      public static void main(String[] args) throws IOException {
8.          String fileName="D:"+File.separator+"hello.txt";
9.          File f=new File(fileName);
10.         char[] ch=new char[100];
11.         Reader read=new FileReader(f);
12.         int count=read.read(ch);
13.         read.close();
14.         System.out.println("读入的长度为: "+count);
15.         System.out.println("内容为"+newString(ch,0,count));
16.     }
17. }
```

【运行结果】：

读入的长度为：17

内容为 hellohello

hello

当然最好采用循环读取的方式，因为我们有时候不知道文件到底有多大。

[java] view plaincopy

```
1.  /**
2.   * 字符流
3.   * 从文件中读出内容
4.   * */
5.  import java.io.*;
6.  class hello{
7.      public static void main(String[] args) throws IOException {
8.          String fileName="D:"+File.separator+"hello.txt";
9.          File f=new File(fileName);
10.         char[] ch=new char[100];
11.         Reader read=new FileReader(f);
12.         int temp=0;
13.         int count=0;
14.         while((temp=read.read())!=(-1)){
15.             ch[count++]=(char)temp;
16.         }
17.         read.close();
18.         System.out.println("内容为"+newString(ch,0,count));
19.     }
```

【运行结果】:

内容为 hellohello

hello

## 关于字节流和字符流的区别

实际上字节流在操作的时候本身是不会用到缓冲区的，是文件本身的直接操作的，但是字符流在操作的时候下后是会用到缓冲区的，是通过缓冲区来操作文件的。

读者可以试着将上面的字节流和字符流的程序的最后一行关闭文件的代码注释掉，然后运行程序看看。你就会发现使用字节流的话，文件中已经存在内容，但是使用字符流的时候，文件中还是没有内容的，这个时候就要刷新缓冲区。

## 使用字节流好还是字符流好呢？

答案是字节流。首先因为硬盘上的所有文件都是以字节的形式进行传输或者保存的，包括图片等内容。但是字符只是在内存中才会形成的，所以在开发中，字节流使用广泛。

## 文件的复制

其实DOS下就有一个文件复制功能，比如我们想把d盘下面的hello.txt文件复制到d盘下面的rollen.txt文件中，那么我们就可以使用下面的命令：

```
copy d:\hello.txt d:\rollen.txt
```

运行之后你会在d盘中看见hello.txt，并且两个文件的内容是一样的，（这是屁话）

下面我们使用程序来复制文件吧。

基本思路还是从一个文件中读入内容，边读边写入另一个文件，就是这么简单。

首先编写下面的代码：

[java] view plaincopy

```
1.  /**
2.   * 文件的复制
3.   * */
4.  import java.io.*;
5.  class hello{
6.      public static void main(String[] args) throws IOException {
7.          if(args.length!=2){
8.              System.out.println("命令行参数输入有误，请检查");
9.              System.exit(1);
10.         }
11.         File file1=new File(args[0]);
12.         File file2=new File(args[1]);
13.
14.         if(!file1.exists()){
15.             System.out.println("被复制的文件不存在");
16.             System.exit(1);
17.         }
18.         InputStream input=new FileInputStream(file1);
19.         OutputStream output=new FileOutputStream(file2);
20.         if((input!=null)&&(output!=null)){
21.             int temp=0;
22.             while((temp=input.read())!=(-1)){
23.                 output.write(temp);
24.             }
25.         }
26.         input.close();
27.         output.close();
28.     }
29. }
```



然后在命令行下面

```
javac hello.java
```

```
java hello d:\hello.txt d:\rollen.txt
```

现在你将会在 d 盘看到 rollen.txt 了，

## OutputStreramWriter 和 InputStreamReader 类

整个 IO 类中除了字节流和字符流还包括字节和字符转换流。

OutputStreramWriter 将输出的字符流转化为字节流

InputStreamReader 将输入的字节流转换为字符流

但是不管如何操作，最后都是以字节的形式保存在文件中的。

## 将字节输出流转化为字符输出流

[java] view plaincopy

```
1.  /**
2.   * 将字节输出流转化为字符输出流
3.   * */
4.  import java.io.*;
5.  class hello{
6.      public static void main(String[] args) throws IOException {
7.          String fileName= "d:"+File.separator+"hello.txt";
8.          File file=new File(fileName);
9.          Writer out=new OutputStreamWriter(new FileOutputStream(file));
10.         out.write("hello");
11.         out.close();
12.     }
13. }
```

### 【运行结果】:

文件中内容为: hello

## 将字节输入流变为字符输入流

[java] view plaincopy

```
1.  /**
2.   * 将字节输入流变为字符输入流
3.   * */
4.  import java.io.*;
5.  class hello{
6.      public static void main(String[] args) throws IOException {
```

```
7.         String fileName= "d:"+File.separator+"hello.txt";
8.         File file=new File(fileName);
9.         Reader read=new InputStreamReader(new FileInputStream(file));
10.        char[] b=new char[100];
11.        int len=read.read(b);
12.        System.out.println(new String(b,0,len));
13.        read.close();
14.    }
15. }
```

【运行结果】: hello

前面列举的输出输入都是以文件进行的，现在我们以内容为输出输入目的地，使用内存操作流

ByteArrayInputStream 主要将内容写入内容

ByteArrayOutputStream 主要将内容从内存输出

使用内存操作流将一个大写字母转化为小写字母

[java] view plaincopy

```
1.  /**
2.   * 使用内存操作流将一个大写字母转化为小写字母
3.   * */
4.  import java.io.*;
5.  class hello{
6.      public static void main(String[] args) throws IOException {
7.          String str="ROLLENHOLT";
8.          ByteArrayInputStream input=new ByteArrayInputStream(str.getBytes());
9.
10.         ByteArrayOutputStream output=new ByteArrayOutputStream();
11.         int temp=0;
12.         while((temp=input.read())!=-1){
13.             char ch=(char)temp;
14.             output.write(Character.toLowerCase(ch));
15.         }
16.         String outStr=output.toString();
17.         input.close();
18.         output.close();
19.         System.out.println(outStr);
20.     }
```

【运行结果】:

rollenholt

内容操作流一般使用来生成一些临时信息采用的，这样可以避免删除的麻烦。

## 管道流

管道流主要可以进行两个线程之间的通信。

PipedOutputStream 管道输出流

PipedInputStream 管道输入流

### 验证管道流

[java] view plaincopy

```
1.  /**
2.   * 验证管道流
3.   * */
4.  import java.io.*;
5.
6.  /**
7.   * 消息发送类
8.   * */
9.  class Send implements Runnable{
10.     private PipedOutputStream out=null;
11.     public Send() {
12.         out=new PipedOutputStream();
13.     }
14.     public PipedOutputStream getOut(){
15.         return this.out;
16.     }
17.     public void run(){
18.         String message="hello , Rollen";
19.         try{
20.             out.write(message.getBytes());
21.         }catch (Exception e) {
22.             e.printStackTrace();
23.         }try{
24.             out.close();
25.         }catch (Exception e) {
26.             e.printStackTrace();
27.         }
28.     }
29. }
30.
31. /**
32.  * 接受消息类
33.  * */
```

```
34. class Recive implements Runnable{
35.     private PipedInputStream input=null;
36.     public Recive(){
37.         this.input=new PipedInputStream();
38.     }
39.     public PipedInputStream getInput(){
40.         return this.input;
41.     }
42.     public void run(){
43.         byte[] b=new byte[1000];
44.         int len=0;
45.         try{
46.             len=this.input.read(b);
47.         }catch (Exception e) {
48.             e.printStackTrace();
49.         }try{
50.             input.close();
51.         }catch (Exception e) {
52.             e.printStackTrace();
53.         }
54.         System.out.println("接受的内容为 "+(newString(b,0,len)));
55.     }
56. }
57. /**
58.  * 测试类
59.  */
60. class hello{
61.     public static void main(String[] args) throws IOException {
62.         Send send=new Send();
63.         Recive recive=new Recive();
64.         try{
65. //管道连接
66.             send.getOut().connect(recive.getInput());
67.         }catch (Exception e) {
68.             e.printStackTrace();
69.         }
70.         new Thread(send).start();
71.         new Thread(recive).start();
72.     }
73. }
```

【运行结果】:

接受的内容为 hello , Rollen

## 打印流

[java] view plaincopy

```
1. /**
2.  * 使用 PrintStream 进行输出
3.  * */
4. import java.io.*;
5.
6. class hello {
7.     public static void main(String[] args) throws IOException {
8.         PrintStream print = new PrintStream(new FileOutputStream(new File("d
9.             + File.separator + "hello.txt")));
10.        print.println(true);
11.        print.println("Rollen");
12.        print.close();
13.    }
14. }
```

### 【运行结果】:

true

Rollen

当然也可以格式化输出

[java] view plaincopy

```
1. /**
2.  * 使用 PrintStream 进行输出
3.  * 并进行格式化
4.  * */
5. import java.io.*;
6. class hello {
7.     public static void main(String[] args) throws IOException {
8.         PrintStream print = new PrintStream(new FileOutputStream(new File("d
9.             + File.separator + "hello.txt")));
10.        String name="Rollen";
11.        int age=20;
12.        print.printf("姓名: %s. 年龄: %d.", name, age);
13.        print.close();
14.    }
15. }
```

【运行结果】:

姓名: Rollen. 年龄: 20.

使用 `OutputStream` 向屏幕上输出内容

[java] view plaincopy

```
1. /**
2.  * 使用 OutputStream 向屏幕上输出内容
3.  * */
4. import java.io.*;
5. class hello {
6.     public static void main(String[] args) throws IOException {
7.         OutputStream out=System.out;
8.         try{
9.             out.write("hello".getBytes());
10.        }catch (Exception e) {
11.            e.printStackTrace();
12.        }
13.        try{
14.            out.close();
15.        }catch (Exception e) {
16.            e.printStackTrace();
17.        }
18.    }
19. }
```

【运行结果】:

hello

输入输出重定向

[java] view plaincopy

```
1. import java.io.File;
2. import java.io.FileNotFoundException;
3. import java.io.FileOutputStream;
4. import java.io.PrintStream;
5.
6. /**
7.  * 为 System.out.println() 重定向输出
8.  * */
9. public class systemDemo{
10.     public static void main(String[] args){
11.         // 此刻直接输出到屏幕
```

```
12.         System.out.println("hello");
13.         File file = new File("d:" + File.separator + "hello.txt");
14.         try{
15.             System.setOut(new PrintStream(new FileOutputStream(file)));
16.         }catch(FileNotFoundException e){
17.             e.printStackTrace();
18.         }
19.         System.out.println("这些内容在文件中才能看到哦！");
20.     }
21. }
```

【运行结果】:

eclipse 的控制台输出的是 hello。然后当我们查看 d 盘下面的 hello.txt 文件的时候，会在里面看到：这些内容在文件中才能看到哦！

[java] view plaincopy

```
1. import java.io.File;
2. import java.io.FileNotFoundException;
3. import java.io.FileOutputStream;
4. import java.io.PrintStream;
5.
6. /**
7.  * System.err 重定向 这个例子也提示我们可以使用这种方法保存错误信息
8.  */
9. public class systemErr{
10.     public static void main(String[] args){
11.         File file = new File("d:" + File.separator + "hello.txt");
12.         System.err.println("这些在控制台输出");
13.         try{
14.             System.setErr(new PrintStream(new FileOutputStream(file)));
15.         }catch(FileNotFoundException e){
16.             e.printStackTrace();
17.         }
18.         System.err.println("这些在文件中才能看到哦！");
19.     }
20. }
```

【运行结果】:

你会在 eclipse 的控制台看到红色的输出：“这些在控制台输出”，然后在 d 盘下面的 hello.txt 中会看到：这些在文件中才能看到哦！

[java] view plaincopy

```
1. import java.io.File;
```

```
2. import java.io.FileInputStream;
3. import java.io.FileNotFoundException;
4. import java.io.IOException;
5.
6. /**
7.  * System.in 重定向
8.  */
9. public class systemIn{
10.     public static void main(String[] args){
11.         File file = new File("d:"+ File.separator + "hello.txt");
12.         if(!file.exists()){
13.             return;
14.         }else{
15.             try{
16.                 System.setIn(new FileInputStream(file));
17.             }catch(FileNotFoundException e){
18.                 e.printStackTrace();
19.             }
20.             byte[] bytes = newbyte[1024];
21.             int len = 0;
22.             try{
23.                 len = System.in.read(bytes);
24.             }catch(IOException e){
25.                 e.printStackTrace();
26.             }
27.             System.out.println("读入的内容为: "+ new String(bytes, 0, len));
28.         }
29.     }
30. }
```

#### 【运行结果】:

前提是我的 d 盘下面的 hello.txt 中的内容是：“这些文件中的内容哦！”，然后运行程序，输出的结果为：读入的内容为：这些文件中的内容哦！

#### BufferedReader 的小例子

注意：BufferedReader 只能接受字符流的缓冲区，因为每一个中文需要占据两个字节，所以需要先将 System.in 这个字节输入流变为字符输入流，采用：  
BufferedReader buf = new BufferedReader(  
 new InputStreamReader(System.in));

下面给一个实例：

[java] view plaincopy

```
1. import java.io.BufferedReader;
```



```
2. import java.io.IOException;
3. import java.io.InputStreamReader;
4.
5. /**
6.  * 使用缓冲区从键盘上读入内容
7.  * */
8. public class BufferedReaderDemo{
9.     public static void main(String[] args){
10.         BufferedReader buf = new BufferedReader(
11.             new InputStreamReader(System.in));
12.         String str = null;
13.         System.out.println("请输入内容");
14.         try{
15.             str = buf.readLine();
16.         }catch(IOException e){
17.             e.printStackTrace();
18.         }
19.         System.out.println("你输入的内容是: "+ str);
20.     }
21. }
```

【运行结果】:

请输入内容

dasdas

你输入的内容是: dasdas

## Scanner 类

其实我们比较常用的是采用 Scanner 类来进行数据输入，下面来给一个 Scanner 的例子吧

[java] view plaincopy

```
1. import java.util.Scanner;
2.
3. /**
4.  * Scanner 的小例子，从键盘读数据
5.  * */
6. public class ScannerDemo{
7.     public static void main(String[] args){
8.         Scanner sca = new Scanner(System.in);
9.         // 读一个整数
10.        int temp = sca.nextInt();
11.        System.out.println(temp);
12.        //读取浮点数
13.        float flo=sca.nextFloat();
```

```
14.         System.out.println(flo);
15.         //读取字符
16.         //...等等的，都是一些太基础的，就不师范了。
17.     }
18. }
```

其实 `Scanner` 可以接受任何的输入流

下面给一个使用 `Scanner` 类从文件中读出内容

[java] view plaincopy

```
1. import java.io.File;
2. import java.io.FileNotFoundException;
3. import java.util.Scanner;
4.
5. /**
6.  * Scanner 的小例子，从文件中读内容
7.  * */
8. public class ScannerDemo{
9.     public static void main(String[] args){
10.
11.         File file = new File("d:" + File.separator + "hello.txt");
12.         Scanner sca = null;
13.         try{
14.             sca = new Scanner(file);
15.         }catch(FileNotFoundException e){
16.             e.printStackTrace();
17.         }
18.         String str = sca.next();
19.         System.out.println("从文件中读取的内容是: " + str);
20.     }
21. }
```

【运行结果】:

从文件中读取的内容是：这些文件中的内容哦！

**数据操作流 `DataOutputStream`、`DataInputStream` 类**

[java] view plaincopy

```
1. import java.io.DataOutputStream;
2. import java.io.File;
3. import java.io.FileOutputStream;
4. import java.io.IOException;
5.
6. public class DataOutputStreamDemo{
7.     public static void main(String[] args) throws IOException{
```

```
8.         File file = new File("d:" + File.separator + "hello.txt");
9.         char[] ch = { 'A', 'B', 'C' };
10.        OutputStream out = null;
11.        out = new FileOutputStream(new FileOutputStream(file));
12.        for(char temp : ch){
13.            out.writeChar(temp);
14.        }
15.        out.close();
16.    }
17. }
```

【运行结果】:

A B C

现在我们在上面例子的基础上，使用 `DataInputStream` 读出内容

[java] view plaincopy

```
1. import java.io.DataInputStream;
2. import java.io.File;
3. import java.io.FileInputStream;
4. import java.io.IOException;
5.
6. public class DataOutputStreamDemo{
7.     public static void main(String[] args) throws IOException{
8.         File file = new File("d:" + File.separator + "hello.txt");
9.         DataInputStream input = new DataInputStream(new FileInputStream(file));
10.        char[] ch = new char[10];
11.        int count = 0;
12.        char temp;
13.        while((temp = input.readChar()) != 'C'){
14.            ch[count++] = temp;
15.        }
16.        System.out.println(ch);
17.    }
18. }
```

【运行结果】:

AB

### 合并流 `SequenceInputStream`

`SequenceInputStream` 主要用来将 2 个流合并在一起，比如将两个 txt 中的内容合并为另外一个 txt。下面给出一个实例：

[java] view plaincopy

```
1. import java.io.File;
2. import java.io.FileInputStream;
3. import java.io.FileOutputStream;
4. import java.io.IOException;
5. import java.io.InputStream;
6. import java.io.OutputStream;
7. import java.io.SequenceInputStream;
8.
9. /**
10.  * 将两个文本文件合并为另外一个文本文件
11.  */
12. public class SequenceInputStreamDemo{
13.     public static void main(String[] args) throws IOException{
14.         File file1 = new File("d:" + File.separator + "hello1.txt");
15.         File file2 = new File("d:" + File.separator + "hello2.txt");
16.         File file3 = new File("d:" + File.separator + "hello.txt");
17.         InputStream input1 = new FileInputStream(file1);
18.         InputStream input2 = new FileInputStream(file2);
19.         OutputStream output = new FileOutputStream(file3);
20.         // 合并流
21.         SequenceInputStream sis = new SequenceInputStream(input1, input2);
22.         int temp = 0;
23.         while((temp = sis.read()) != -1){
24.             output.write(temp);
25.         }
26.         input1.close();
27.         input2.close();
28.         output.close();
29.         sis.close();
30.     }
31. }
```

### 【运行结果】

结果会在 hello.txt 文件中包含 hello1.txt 和 hello2.txt 文件中的内容。

### 文件压缩 ZipOutputStream 类

先举一个压缩单个文件的例子吧：

[java] view plain copy

```
1. import java.io.File;
2. import java.io.FileInputStream;
3. import java.io.FileOutputStream;
4. import java.io.IOException;
5. import java.io.InputStream;
6. import java.util.zip.ZipEntry;
```

```
7. import java.util.zip.ZipOutputStream;
8.
9. public class ZipOutputStreamDemo1{
10.     public static void main(String[] args) throws IOException{
11.         File file = new File("d:" + File.separator + "hello.txt");
12.         File zipFile = new File("d:" + File.separator + "hello.zip");
13.         InputStream input = new FileInputStream(file);
14.         ZipOutputStream zipOut = new ZipOutputStream(new FileOutputStream(
15.             zipFile));
16.         zipOut.putNextEntry(new ZipEntry(file.getName()));
17.         // 设置注释
18.         zipOut.setComment("hello");
19.         int temp = 0;
20.         while((temp = input.read()) != -1){
21.             zipOut.write(temp);
22.         }
23.         input.close();
24.         zipOut.close();
25.     }
26. }
```

### 【运行结果】

运行结果之前，我创建了一个 hello.txt 的文件，原本大小 56 个字节，但是压缩之后产生 hello.zip 之后，居然变成了 175 个字节，有点搞不懂。

不过结果肯定是正确的，我只是提出我的一个疑问而已。

上面的这个例子测试的是压缩单个文件，下面的们来看看如何压缩多个文件。

[java] view plaincopy

```
1. import java.io.File;
2. import java.io.FileInputStream;
3. import java.io.FileOutputStream;
4. import java.io.IOException;
5. import java.io.InputStream;
6. import java.util.zip.ZipEntry;
7. import java.util.zip.ZipOutputStream;
8.
9. /**
10.  * 一次性压缩多个文件
11.  */
12. public class ZipOutputStreamDemo2{
```

```
13.     public static void main(String[] args) throws IOException{
14.         // 要被压缩的文件夹
15.         File file = new File("d:" + File.separator + "temp");
16.         File zipFile = new File("d:" + File.separator + "zipFile.zip");
17.         InputStream input = null;
18.         ZipOutputStream zipOut = new ZipOutputStream(new FileOutputStream(
19.             zipFile));
20.         zipOut.setComment("hello");
21.         if(file.isDirectory()){
22.             File[] files = file.listFiles();
23.             for(int i = 0; i < files.length; ++i){
24.                 input = new FileInputStream(files[i]);
25.                 zipOut.putNextEntry(newZipEntry(file.getName()
26.                     + File.separator + files[i].getName()));
27.                 int temp = 0;
28.                 while((temp = input.read()) != -1){
29.                     zipOut.write(temp);
30.                 }
31.                 input.close();
32.             }
33.         }
34.         zipOut.close();
35.     }
36. }
```

### 【运行结果】

先看看要被压缩的文件吧：

接下来看看压缩之后的：

大家自然想到，既然能压缩，自然能解压缩，在谈解压缩之前，我们会用到一个 `ZipFile` 类，先给一个这个例子吧。java 中的每一个压缩文件都是可以使用 `ZipFile` 来进行表示的

[java] view plaincopy

```
1. import java.io.File;
2. import java.io.IOException;
3. import java.util.zip.ZipFile;
4.
5. /**
6.  * ZipFile 演示
```

```
7.  * */
8.  public class ZipFileDemo{
9.      public static void main(String[] args) throws IOException{
10.          File file = new File("d:" + File.separator + "hello.zip");
11.          ZipFile zipFile = new ZipFile(file);
12.          System.out.println("压缩文件的名称为: " + zipFile.getName());
13.      }
14. }
```

【运行结果】:

压缩文件的名称为: d:\hello.zip

现在我们呢是时候来看看如何加压缩文件了,和之前一样,先让我们来解压单个压缩文件(也就是压缩文件中只有一个文件的情况),我们采用前面的例子产生的压缩文件 **hello.zip**

[java] view plaincopy

```
1.  import java.io.File;
2.  import java.io.FileOutputStream;
3.  import java.io.IOException;
4.  import java.io.InputStream;
5.  import java.io.OutputStream;
6.  import java.util.zip.ZipEntry;
7.  import java.util.zip.ZipFile;
8.
9.  /**
10.   * 解压缩文件（压缩文件中只有一个文件的情况）
11.   * */
12.  public class ZipFileDemo2{
13.      public static void main(String[] args) throws IOException{
14.          File file = new File("d:" + File.separator + "hello.zip");
15.          File outFile = new File("d:" + File.separator + "unZipFile.txt");
16.          ZipFile zipFile = new ZipFile(file);
17.          ZipEntry entry = zipFile.getEntry("hello.txt");
18.          InputStream input = zipFile.getInputStream(entry);
19.          OutputStream output = new FileOutputStream(outFile);
20.          int temp = 0;
21.          while((temp = input.read()) != -1){
22.              output.write(temp);
23.          }
24.          input.close();
25.          output.close();
26.      }
27. }
```

### 【运行结果】:

解压缩之前:

这个压缩文件还是 175 字节

解压之后产生:

又回到了 56 字节，表示郁闷。

现在让我们来解压一个压缩文件中包含多个文件的情况吧

### ZipInputStream 类

当我们需要解压缩多个文件的时候，ZipEntry 就无法使用了，如果想操作更加复杂的压缩文件，我们就必须使用 ZipInputStream 类

[java] view plain copy

```
1. import java.io.File;
2. import java.io.FileInputStream;
3. import java.io.FileOutputStream;
4. import java.io.IOException;
5. import java.io.InputStream;
6. import java.io.OutputStream;
7. import java.util.zip.ZipEntry;
8. import java.util.zip.ZipFile;
9. import java.util.zip.ZipInputStream;
10.
11. /**
12.  * 解压缩一个压缩文件中包含多个文件的情况
13.  * */
14. public class ZipFileDemo3{
15.     public static void main(String[] args) throws IOException{
16.         File file = new File("d:" + File.separator + "zipFile.zip");
17.         File outFile = null;
18.         ZipFile zipFile = new ZipFile(file);
19.         ZipInputStream zipInput = new ZipInputStream(new FileInputStream(file));
20.         ZipEntry entry = null;
21.         InputStream input = null;
22.         OutputStream output = null;
23.         while((entry = zipInput.getNextEntry()) != null){
24.             System.out.println("解压缩" + entry.getName() + "文件");
25.             outFile = new File("d:" + File.separator + entry.getName());
```



```
26.         if(!outFile.getParentFile().exists()){
27.             outFile.getParentFile().mkdir();
28.         }
29.         if(!outFile.exists()){
30.             outFile.createNewFile();
31.         }
32.         input = zipFile.getInputStream(entry);
33.         output = new FileOutputStream(outFile);
34.         int temp = 0;
35.         while((temp = input.read()) != -1){
36.             output.write(temp);
37.         }
38.         input.close();
39.         output.close();
40.     }
41. }
42. }
```

#### 【运行结果】:

被解压的文件:

解压之后再 D 盘下会出现一个 temp 文件夹, 里面内容:

#### PushBackInputStream 回退流

[java] view plaincopy

```
1. import java.io.ByteArrayInputStream;
2. import java.io.IOException;
3. import java.io.PushbackInputStream;
4.
5. /**
6.  * 回退流操作
7.  */
8. public class PushBackInputStreamDemo{
9.     public static void main(String[] args) throws IOException{
10.         String str = "hello,rollenholt";
11.         PushbackInputStream push = null;
12.         ByteArrayInputStream bat = null;
13.         bat = new ByteArrayInputStream(str.getBytes());
14.         push = new PushbackInputStream(bat);
15.         int temp = 0;
16.         while((temp = push.read()) != -1){
```

```
17.         if(temp == ','){
18.             push.unread(temp);
19.             temp = push.read();
20.             System.out.print("(回退"+ (char) temp + ") ");
21.         }else{
22.             System.out.print((char) temp);
23.         }
24.     }
25. }
26. }
```

【运行结果】:

hello(回退,)rollenholt

[java] view plaincopy

```
1.  /**
2.   * 取得本地的默认编码
3.   * */
4.  public class CharSetDemo{
5.      public static void main(String[] args){
6.          System.out.println("系统默认编码为:
7.          "+ System.getProperty("file.encoding"));
8.      }
9.  }
```

【运行结果】:

系统默认编码为: GBK

乱码的产生:

[java] view plaincopy

```
1.  import java.io.File;
2.  import java.io.FileOutputStream;
3.  import java.io.IOException;
4.  import java.io.OutputStream;
5.
6.  /**
7.   * 乱码的产生
8.   * */
9.  public class CharSetDemo2{
10.     public static void main(String[] args) throws IOException{
11.         File file = new File("d:"+ File.separator + "hello.txt");
12.         OutputStream out = new FileOutputStream(file);
```

```
13.         byte[] bytes = "你好".getBytes("ISO8859-1");
14.         out.write(bytes);
15.         out.close();
16.     }
17. }
```

【运行结果】:

??

一般情况下产生乱码，都是由于编码不一致的问题。

## 对象的序列化

对象序列化就是把一个对象变为二进制数据流的一种方法。

一个类要想被序列化，就必须实现 `java.io.Serializable` 接口。虽然这个接口中没有任何方法，就如同之前的 `cloneable` 接口一样。实现了这个接口之后，就表示这个类具有被序列化的能力。

先让我们实现一个具有序列化能力的类吧：

[java] view plaincopy

```
1. import java.io.*;
2. /**
3.  * 实现具有序列化能力的类
4.  * */
5. public class SerializableDemo implements Serializable{
6.     public SerializableDemo(){
7.
8.     }
9.     public SerializableDemo(String name, int age){
10.         this.name=name;
11.         this.age=age;
12.     }
13.     @Override
14.     public String toString(){
15.         return "姓名: "+name+" 年龄: "+age;
16.     }
17.     private String name;
18.     private int age;
19. }
```

这个类就具有实现序列化能力，

在继续将序列化之前，先将一下 `ObjectInputStream` 和 `ObjectOutputStream` 这两个类先给一个 `ObjectOutputStream` 的例子吧：

[java] view plaincopy

```
1. import java.io.Serializable;
2. import java.io.File;
3. import java.io.FileOutputStream;
4. import java.io.IOException;
5. import java.io.ObjectOutputStream;
6.
7. /**
8.  * 实现具有序列化能力的类
9.  * */
10. public class Person implements Serializable{
11.     public Person(){
12.
13.     }
14.
15.     public Person(String name, int age){
16.         this.name = name;
17.         this.age = age;
18.     }
19.
20.     @Override
21.     public String toString(){
22.         return "姓名: " + name + " 年龄: " + age;
23.     }
24.
25.     private String name;
26.     private int age;
27. }
28. /**
29.  * 示范 ObjectOutputStream
30.  * */
31. public class ObjectOutputStreamDemo{
32.     public static void main(String[] args) throws IOException{
33.         File file = new File("d:" + File.separator + "hello.txt");
34.         ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(
35.             file));
36.         oos.writeObject(new Person("rollen", 20));
37.         oos.close();
38.     }
39. }
```

#### 【运行结果】:

当我们查看产生的 `hello.txt` 的时候, 看到的是乱码, 呵呵。因为是二进制文件。

虽然我们不能直接查看里面的内容，但是我们可以使用 `ObjectInputStream` 类查看：

[java] view plaincopy

```
1. import java.io.File;
2. import java.io.FileInputStream;
3. import java.io.ObjectInputStream;
4.
5. /**
6.  * ObjectInputStream 示范
7.  */
8. public class ObjectInputStreamDemo{
9.     public static void main(String[] args) throws Exception{
10.         File file = new File("d:"+ File.separator + "hello.txt");
11.         ObjectInputStream input = new ObjectInputStream(new FileInputStream(
12.             file));
13.         Object obj = input.readObject();
14.         input.close();
15.         System.out.println(obj);
16.     }
17. }
```

#### 【运行结果】

姓名: rollen 年龄: 20

到底序列化什么内容呢？

其实只有属性会被序列化。

#### Externalizable 接口

被 `Serializable` 接口声明的类的对象的属性都将被序列化，但是如果想自定义序列化的内容的时候，就需要实现 `Externalizable` 接口。

当一个类要使用 `Externalizable` 这个接口的时候，这个类中必须要有一个无参的构造函数，如果没有的话，在构造的时候会产生异常，这是因为在反序列化时候会默认调用无参的构造函数。

现在我们来演示一下序列化和反序列化：

[java] view plaincopy

```
1. package IO;
2.
3. import java.io.Externalizable;
4. import java.io.File;
```

```
5. import java.io.FileInputStream;
6. import java.io.FileOutputStream;
7. import java.io.IOException;
8. import java.io.ObjectInput;
9. import java.io.ObjectInputStream;
10. import java.io.ObjectOutput;
11. import java.io.ObjectOutputStream;
12.
13. /**
14.  * 序列化和反序列化的操作
15.  * */
16. public class ExternalizableDemo{
17.     public static void main(String[] args) throws Exception{
18.         ser(); // 序列化
19.         dser(); // 反序列化
20.     }
21.
22.     public static void ser() throws Exception{
23.         File file = new File("d:" + File.separator + "hello.txt");
24.         ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(
25.             file));
26.         out.writeObject(new Person("rollen",20));
27.         out.close();
28.     }
29.
30.     public static void dser() throws Exception{
31.         File file = new File("d:" + File.separator + "hello.txt");
32.         ObjectInputStream input = new ObjectInputStream(new FileInputStream(
33.             file));
34.         Object obj = input.readObject();
35.         input.close();
36.         System.out.println(obj);
37.     }
38. }
39.
40. class Person implements Externalizable{
41.     public Person(){
42.
43.     }
44.
45.     public Person(String name, int age){
46.         this.name = name;
```

```
47.         this.age = age;
48.     }
49.
50.     @Override
51.     public String toString(){
52.         return "姓名: " + name + "  年龄: " + age;
53.     }
54.
55.     // 复写这个方法, 根据需要可以保存的属性或者具体内容, 在序列化的时候使用
56.     @Override
57.     public void writeExternal(ObjectOutput out)throws IOException{
58.         out.writeObject(this.name);
59.         out.writeInt(age);
60.     }
61.
62.     // 复写这个方法, 根据需要读取内容 反序列化话的时候需要
63.     @Override
64.     public void readExternal(ObjectInput in)throws IOException,
65.         ClassNotFoundException{
66.         this.name = (String) in.readObject();
67.         this.age = in.readInt();
68.     }
69.
70.     private String name;
71.     private int age;
72. }
```

【运行结果】:

姓名: rollen 年龄: 20

本例中, 我们将全部的属性都保留了下来,

**Serializable** 接口实现的操作其实是吧一个对象中的全部属性进行序列化, 当然也可以使用我们上使用的是 **Externalizable** 接口以实现部分属性的序列化, 但是这样的操作比较麻烦, 当我们使用 **Serializable** 接口实现序列化操作的时候, 如果一个对象的某一个属性不想被序列化保存下来, 那么我们可以使用 **transient** 关键字进行说明:

下面举一个例子:

[java] view plaincopy

```
1. package IO;
2.
3. import java.io.File;
4. import java.io.FileInputStream;
5. import java.io.FileOutputStream;
```

```
6. import java.io.ObjectInputStream;
7. import java.io.ObjectOutputStream;
8. import java.io.Serializable;
9.
10. /**
11.  * 序列化和反序列化的操作
12.  * */
13. public class serDemo{
14.     public static void main(String[] args) throws Exception{
15.         ser(); // 序列化
16.         dser(); // 反序列化
17.     }
18.
19.     public static void ser() throws Exception{
20.         File file = new File("d:" + File.separator + "hello.txt");
21.         ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(
22.             file));
23.         out.writeObject(new Person1("rollen",20));
24.         out.close();
25.     }
26.
27.     public static void dser() throws Exception{
28.         File file = new File("d:" + File.separator + "hello.txt");
29.         ObjectInputStream input = new ObjectInputStream(new FileInputStream(
30.             file));
31.         Object obj = input.readObject();
32.         input.close();
33.         System.out.println(obj);
34.     }
35. }
36.
37. class Person1 implements Serializable{
38.     public Person1(){
39.
40.     }
41.
42.     public Person1(String name, int age){
43.         this.name = name;
44.         this.age = age;
45.     }
46.
47.     @Override
```



```
48.     public String toString(){
49.         return "姓名: " + name + "  年龄: " + age;
50.     }
51.
52.     // 注意这里
53.     private transient String name;
54.     private int age;
55. }
```

【运行结果】:

姓名: null 年龄: 20

最后在给一个序列化一组对象的例子吧:

[\[java\] view plaincopy](#)

```
1.  import java.io.File;
2.  import java.io.FileInputStream;
3.  import java.io.FileOutputStream;
4.  import java.io.ObjectInputStream;
5.  import java.io.ObjectOutputStream;
6.  import java.io.Serializable;
7.
8.  /**
9.   * 序列化一组对象
10.  */
11. public class SerDemo1{
12.     public static void main(String[] args) throws Exception{
13.         Student[] stu = { new Student("hello",20), new Student("world",30),
14.
15.             new Student("rollen",40) };
16.         ser(stu);
17.         Object[] obj = dser();
18.         for(int i =0; i < obj.length; ++i){
19.             Student s = (Student) obj[i];
20.             System.out.println(s);
21.         }
22.
23.         // 序列化
24.         public static void ser(Object[] obj) throws Exception{
25.             File file = new File("d:" + File.separator + "hello.txt");
26.             ObjectOutputStream out = new ObjectOutputStream(new FileOutputStrea
27. m(
28.                 file));
29.             out.writeObject(obj);
30.         }
```

```
29.         out.close();
30.     }
31.
32.     // 反序列化
33.     public static Object[] dser()throws Exception{
34.         File file = new File("d:"+ File.separator + "hello.txt");
35.         ObjectInputStream input = newObjectInputStream(new FileInputStream(
36.             file));
37.         Object[] obj = (Object[]) input.readObject();
38.         input.close();
39.         return obj;
40.     }
41. }
42.
43. class Student implements Serializable{
44.     public Student(){
45.
46.     }
47.
48.     public Student(String name, intage){
49.         this.name = name;
50.         this.age = age;
51.     }
52.
53.     @Override
54.     public String toString(){
55.         return "姓名:  " + name + "  年龄: " + age;
56.     }
57.
58.     private String name;
59.     private int age;
60. }
```

**【运行结果】:**

姓名: hello 年龄: 20

姓名: world 年龄: 30

姓名: rollen 年龄: 40