笔记总链接：http://bbs.itheima.com/thread-200600-1-1.html

8、IO

8.2 IO流

8.2.11 IO包中的其他类

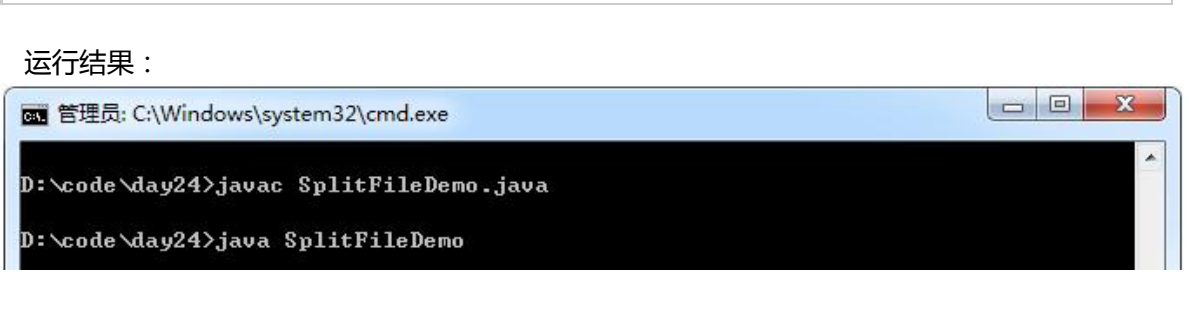
需求：文件切割器。

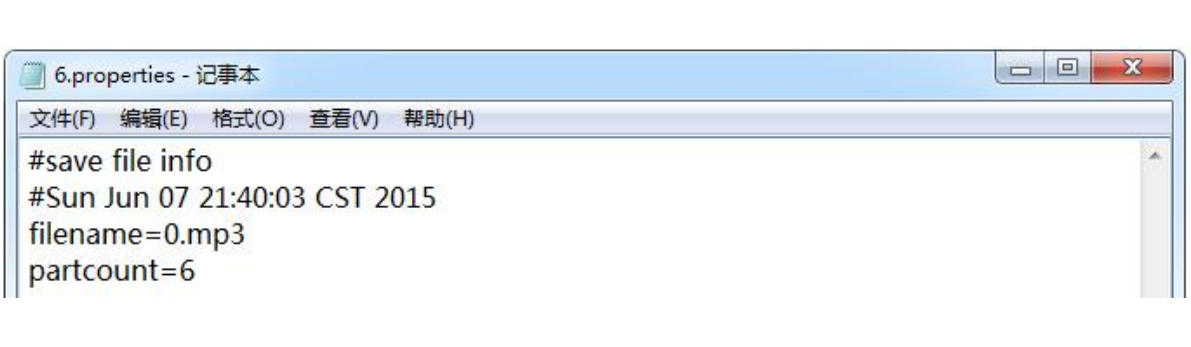
代码：

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Properties;

public class SplitFileDemo{

       private static final int SIZE = 1024*1024;

       public static void main(String[] args) throws IOException{
            File file = new File("0.mp3" );
            splitFile(file);
      }

       public static void splitFile(File file) throws IOException {
             //用读取流关联源文件
            FileInputStream fis = new FileInputStream(file);

             //定义一个1M的缓冲区
             byte[] buf = new byte[SIZE];

             //创建目的
            FileOutputStream fos = null;

             int len = 0;
             int count = 1;

             //切割文件时，必须记录住被切割文件的名称，以及切割出来碎片文件的个数，以方便于合并。
             //这个信息为了进行描述，使用键值对的方式，用到了properties对象。

            Properties prop = new Properties();

            File dir = new File("c:\\partFiles" );
             if(!dir.exists())
                  dir.mkdirs();

             while((len = fis.read(buf)) != -1){
                  fos = new FileOutputStream(new File(dir,(count++) + ".part"));
                  fos.write(buf,0,len);
                  fos.close();
            }

             //将被切割文件的信息保存到prop集合中
            prop.setProperty( "partcount",count + "" );
            prop.setProperty( "filename",file.getName());

            fos = new FileOutputStream(new File(dir,count + ".properties" ));

             //将prop集合中的数据存储到文件中
            prop.store(fos, "save file info");

            fis.close();
            fos.close();
      }
}
```

复制代码

运行结果：





需求：文件合并器。

代码：

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FilenameFilter;
import java.io.IOException;
import java.io.SequenceInputStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.Properties;

public class MergeFile{
       public static void main(String[] args) throws IOException {
            File dir = new File("c:\\partFiles" );
            mergeFile(dir);
      }

       public static void mergeFile(File dir) throws IOException {

             //获取指定目录下的配置文件对象
            File[] files = dir.listFiles( new SuffixFilter(".properties" ));

             if(files.length!=1)
                   throw new RuntimeException(dir + ",该目录下没有properties扩展名的文件或者不唯一" );

             //记录配置文件对象
            File confile = files[0];

             //获取该文件中的信息
            Properties prop = new Properties();
            FileInputStream fis = new FileInputStream(confile);

            prop.load(fis);

            String filename = prop.getProperty( "filename");

             int count = Integer.parseInt(prop.getProperty("partcount"));

             //获取该目录下的所有碎片文件
            File[] partFiles = dir.listFiles( new SuffixFilter(".part" ));

             if(partFiles.length != (count - 1)){
                   throw new RuntimeException("碎片文件不符合要求，个数不对！应该是" + count + "个");
            }

             //将碎片文件和流对象关联并存储到集合中
            ArrayList<FileInputStream> al = new ArrayList<FileInputStream>();

             for(int x = 1; x <= partFiles.length; x++){
                  al.add( new FileInputStream(partFiles[x-1]));
            }

             final Iterator<FileInputStream> it = al.iterator();

             //将多个流合并成一个序列流
            Enumeration<FileInputStream> en = Collections.enumeration(al);

            SequenceInputStream sis = new SequenceInputStream(en);

            FileOutputStream fos = new FileOutputStream(new File(dir,filename));

             byte[] buf = new byte[1024*1024];

             int len = 0;

             while((len = sis.read(buf)) != -1){
                  fos.write(buf,0,len);
            }

            fos.close();
            sis.close();
      }
}

class SuffixFilter implements FilenameFilter{
       private String suffix;

       public SuffixFilter(String suffix){
             super();
             this.suffix = suffix;
      }

       public boolean accept(File dir,String name){
             return name.endsWith(suffix);
      }
}
```

复制代码

运行结果：

操作对象

ObjectInputStream与ObjectOutputStream

P.S.

被操作的对象需要实现Serializable。

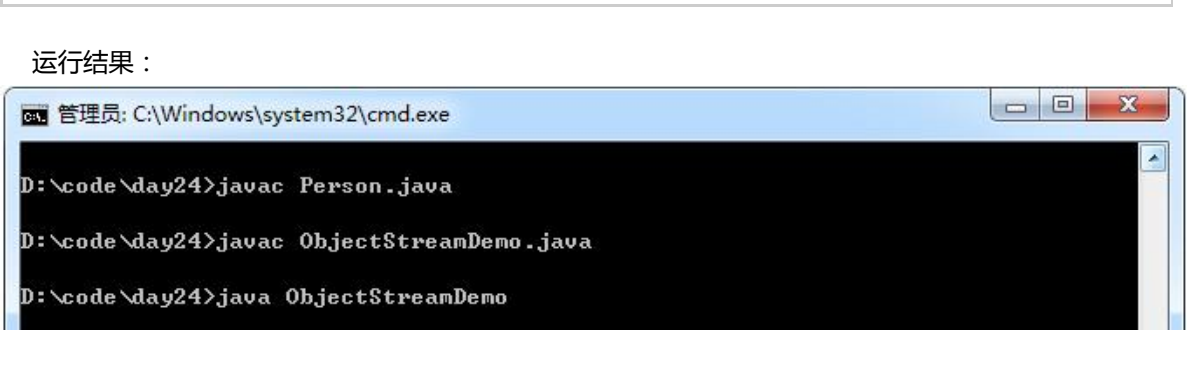
类通过实现java.io.Serializable接口以启用序列化功能，Serializable只是一个标记接口。

示例1：

```java
import java.io.Serializable;

class Person implements Serializable{
       private String name;
       private int age;

       public Person(String name,int age){
             this.name = name;
             this.age = age;
      }

       public String getName(){
             return name;
      }

       public void setName(String name){
             this.name = name;
      }

       public int getAge(){
             return age;
      }

       public void setAge(int age){
             this.age = age;
      }
}
```

复制代码

运行结果：

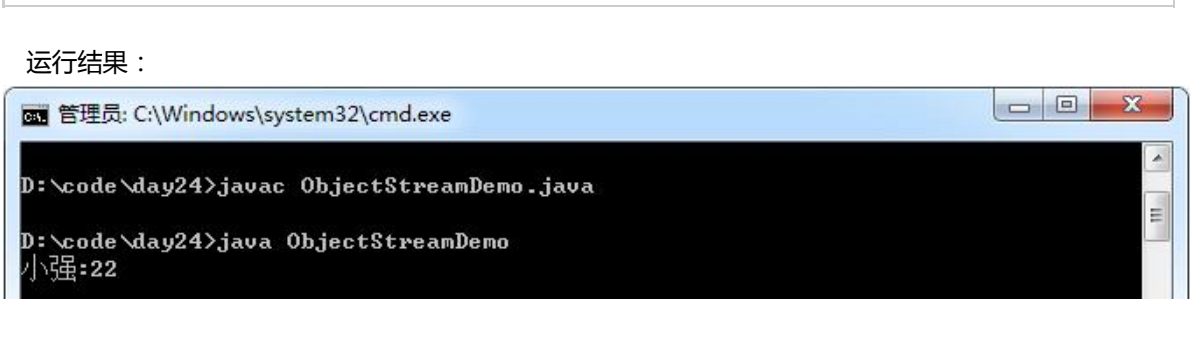


示例2：

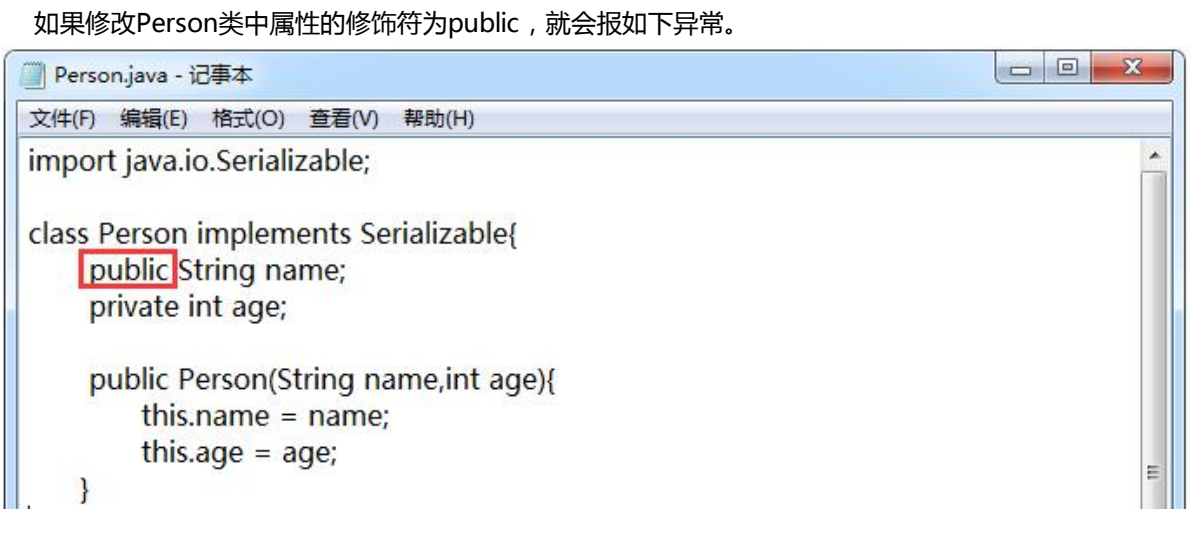
```java
import java.io.FileInputStream;
import java.io.ObjectInputStream;

public class ObjectStreamDemo{
       public static void main(String[] args) throws Exception {
            readObj();
      }

       public static void readObj() throws Exception {
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream("obj.object" ));

            Person p = (Person)ois.readObject();

            System.out.println(p.getName() + ":" + p.getAge());

            ois.close();
      }
}
```

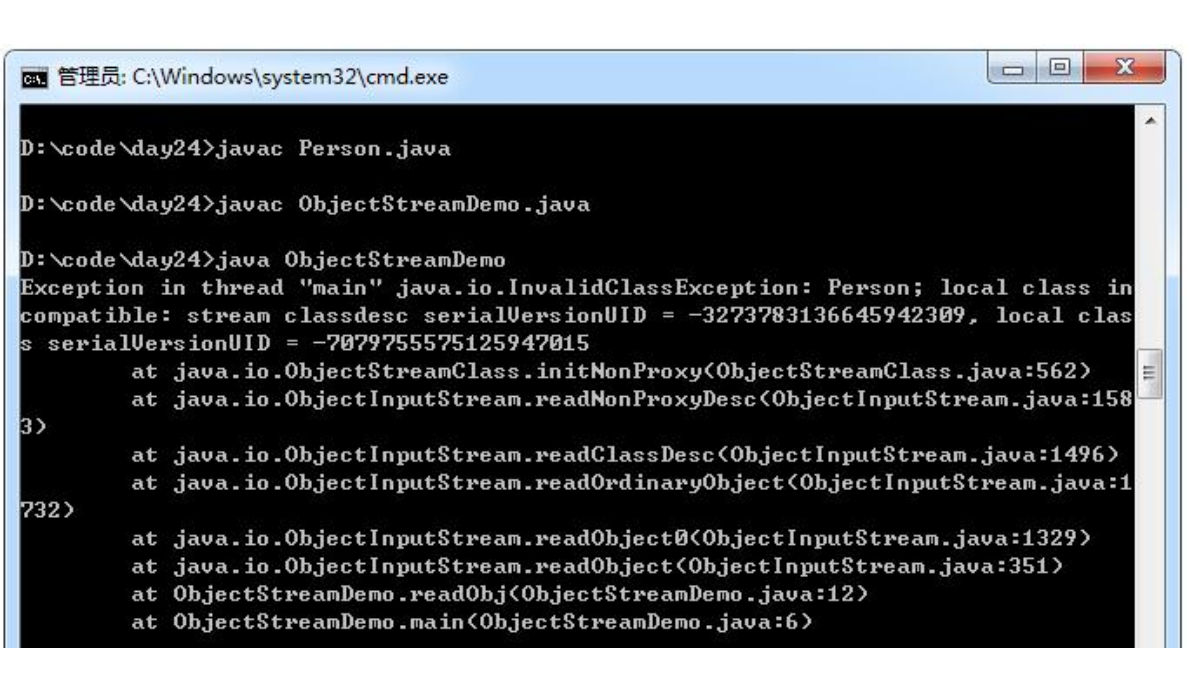
复制代码

运行结果：



如果修改Person类中属性的修饰符为public，就会报如下异常。

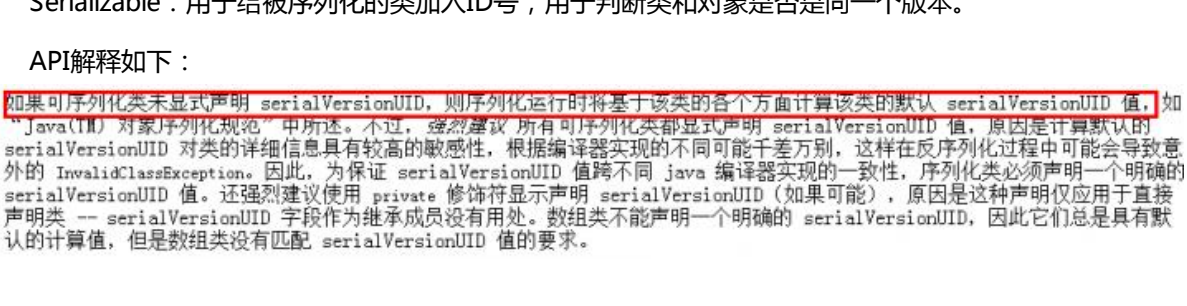




原因分析：

Serializable：用于给被序列化的类加入ID号，用于判断类和对象是否是同一个版本。

API解释如下：


如果可序列化类未显式声明 serialVersionUID，则序列化运行时将基于该类的各个方面计算该类的默认 serialVersionUID 值，如"Java(TM) 对象序列化规范"中所述。不过，强烈建议 所有可序列化类都显式声明 serialVersionUID 值，原因是计算默认的 serialVersionUID 对类的详细信息具有较高的敏感性，根据编译器实现的不同可能千差万别，这样在反序列化过程中可能会导致意外的 InvalidClassException。因此，为保证 serialVersionUID 值跨不同 java 编译器实现的一致性，序列化类必须声明一个明确的 serialVersionUID 值。还强烈建议使用 private 修饰符显示声明 serialVersionUID（如果可能），原因是这种声明仅应用于直接声明类 -- serialVersionUID 字段作为继承成员没有用处。数组类不能声明一个明确的 serialVersionUID，因此它们总是具有默认的计算值，但是数组类没有匹配 serialVersionUID 值的要求。

为Person类添加序列号属性。此时，再将Person类中属性修饰符修改为public，也不会出现任何异常。

```java
class Person implements Serializable{
       private static final long serialVersionUID = 9527;

       public String name ;
       public int age ;

       public Person(String name,int age){
             this.name = name;
             this.age = age;
      }

       public String getName(){
             return name ;
      }

       public void setName(String name){
             this.name = name;
      }

       public int getAge(){
             return age ;
      }

       public void setAge(int age){
             this.age = age;
      }
}
```

复制代码

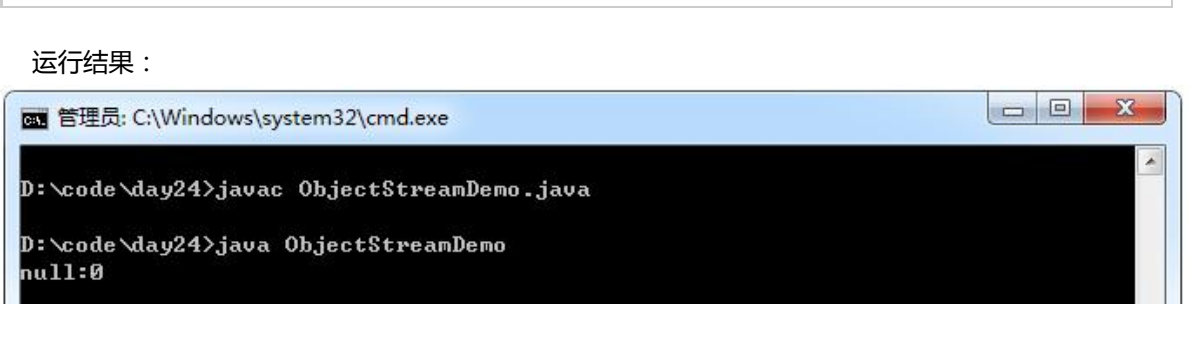
writeObject方法不能写入类及其所有超类型的瞬态和静态字段的值。

示例3：

```java
import java.io.Serializable;

class Person implements Serializable{
       private static final long serialVersionUID = 9527;

       private transient String name;
       private static int age;

       public Person(String name,int age){
             this.name = name;
             this.age = age;
      }

       public String getName(){
             return name;
      }

       public void setName(String name){
             this.name = name;
      }

       public int getAge(){
             return age;
      }

       public void setAge(int age){
             this.age = age;
      }
}
```

复制代码

运行结果：



RandomAccessFile

随机访问文件，自身具备读写的方法。

通过skipBytes(int x),seek(int x)等方法来达到随机访问。

特点：

1. 该对象即能读，又能写。
2. 该对象内部维护了一个byte数组，并通过指针可以操作数组中的元素。
3. 可以通过getFilePointer方法获取指针的位置，和通过seek方法设置指针的位置。
4. 其实该对象就是将字节输入流和输出流进行了封装。
5. 该对象的源或者目的只能是文件。通过构造函数就可以看出。

P.S.

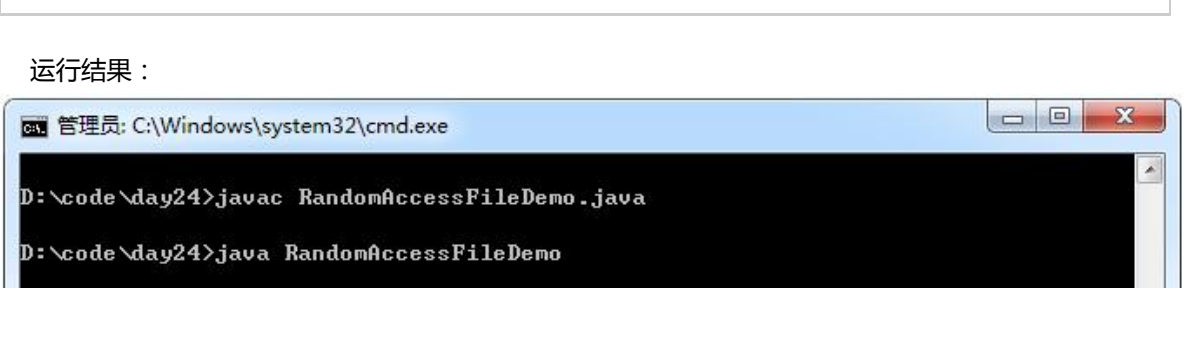
RandomAccessFile不是io体系中的子类。

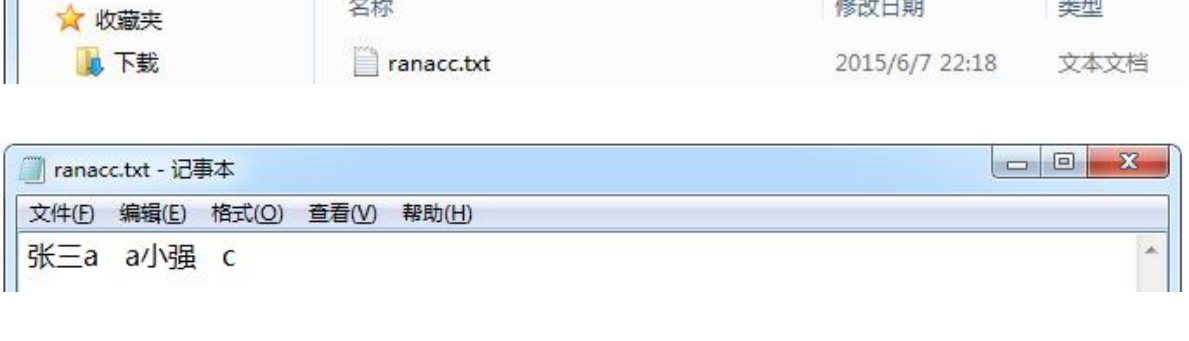
示例1：

```java
import java.io.IOException;
import java.io.RandomAccessFile;

public class RandomAccessFileDemo{
       public static void main(String[] args) throws IOException {
            writeFile();
      }

       //使用RandomAccessFile对象写入一些人员信息，比如姓名和年龄
       public static void writeFile() throws IOException {
             //如果文件不存在，则创建，如果文件存在，不创建
            RandomAccessFile raf = new RandomAccessFile("ranacc.txt" ,"rw" );

            raf.write( "张三".getBytes());
             //使用write方法之写入最后一个字节
            raf.write(97);
             //使用writeInt方法写入四个字节（int类型）
            raf.writeInt(97);

            raf.write( "小强".getBytes());
            raf.writeInt(99);

            raf.close();
      }
}
```

复制代码

运行结果：



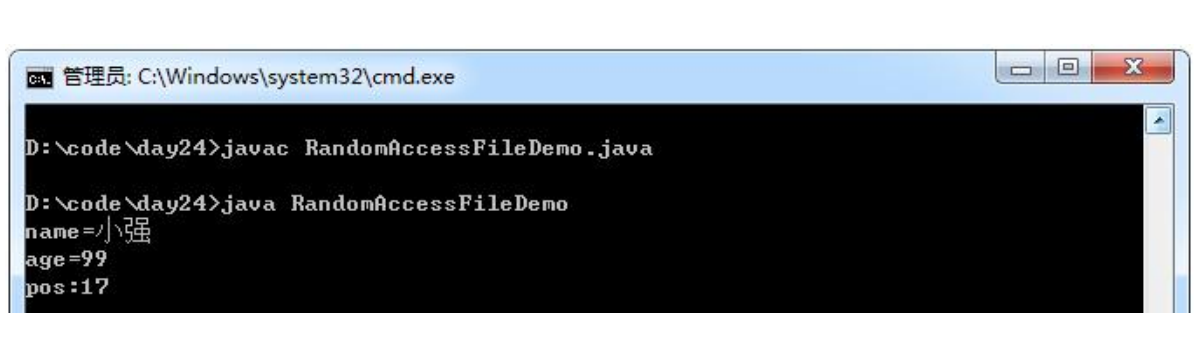


示例2：

```java
import java.io.IOException;
import java.io.RandomAccessFile;

public class RandomAccessFileDemo{
       public static void main(String[] args) throws IOException {
            readFile();
      }

       public static void readFile() throws IOException {
            RandomAccessFile raf = new RandomAccessFile("ranacc.txt" ,"r" );

             //通过seek设置指针的位置
            raf.seek(9); //随机的读取，只要指定指针的位置即可

             byte[] buf = new byte[4];
            raf.read(buf);

            String name = new String(buf);
            System.out.println( "name=" + name);

             int age = raf.readInt();
            System.out.println( "age=" + age);

            System.out.println( "pos:" + raf.getFilePointer());

            raf.close();
      }
}
```

复制代码

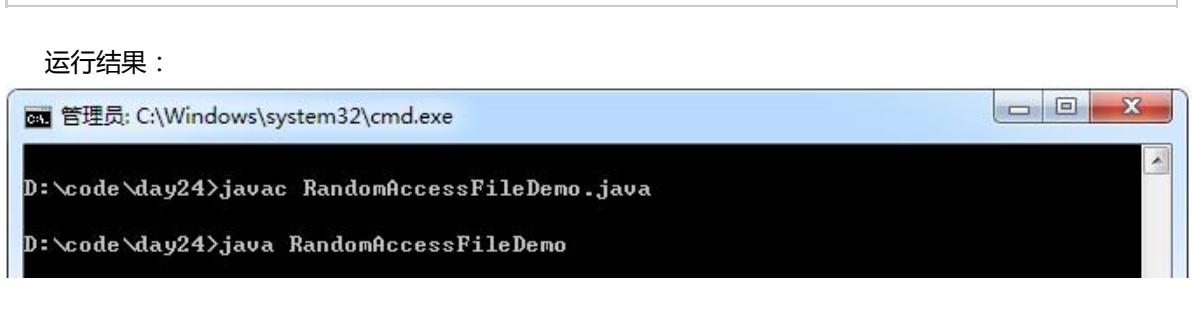
运行结果：



示例3：

```java
import java.io.IOException;
import java.io.RandomAccessFile;

public class RandomAccessFileDemo{
       public static void main(String[] args) throws IOException {
            randomWrite();
       }

       public static void randomWrite() throws IOException {
            RandomAccessFile raf = new RandomAccessFile("ranacc.txt" ,"rw" );

             //往指定位置写入数据
            raf.seek(3*8);

            raf.write( "哈哈".getBytes());
            raf.writeInt(102);

            raf.close();
       }
}
```
复制代码

运行结果：



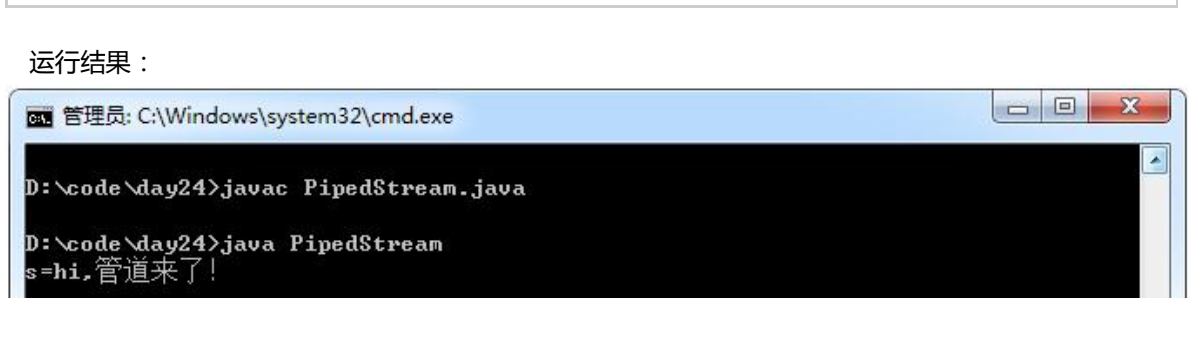
管道流

PipedInputStream和PipedOutputStream：输入输出可以直接进行连接，通过结合线程使用。

示例1：

```java
import java.io.PipedInputStream;
import java.io.PipedOutputStream;

public class PipedStream{
      public static void main(String[] args) throws Exception {
          PipedInputStream input = new PipedInputStream();
          PipedOutputStream output = new PipedOutputStream();

          input.connect(output);

           new Thread(new Input(input)).start();
           new Thread(new Output(output)).start();
      }
}

class Input implements Runnable{
      private PipedInputStream in;

     Input(PipedInputStream in){
           this.in = in;
     }

      public void run(){
           try{
                byte[] buf = new byte[1024];
                int len = in.read(buf);

                String s = new String(buf,0,len);
                System.out.println( "s=" + s);
                in.close();
           } catch(Exception e){
                e.printStackTrace();
           }
     }
}

class Output implements Runnable{
      private PipedOutputStream out;

     Output(PipedOutputStream out){
           this.out = out;
     }

      public void run(){
           try{
                out.write( "hi,管道来了！" .getBytes());
                out.close();
           } catch(Exception e){
                e.printStackTrace();
           }
     }
}
```
复制代码

运行结果：



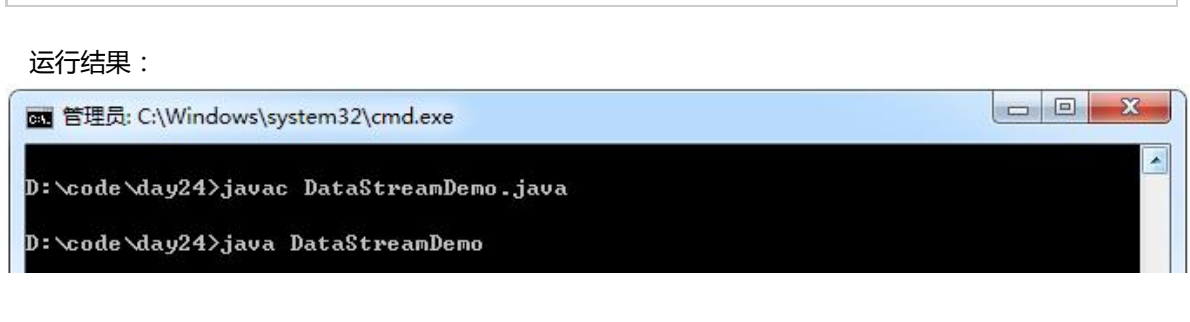
操作基本数据类型

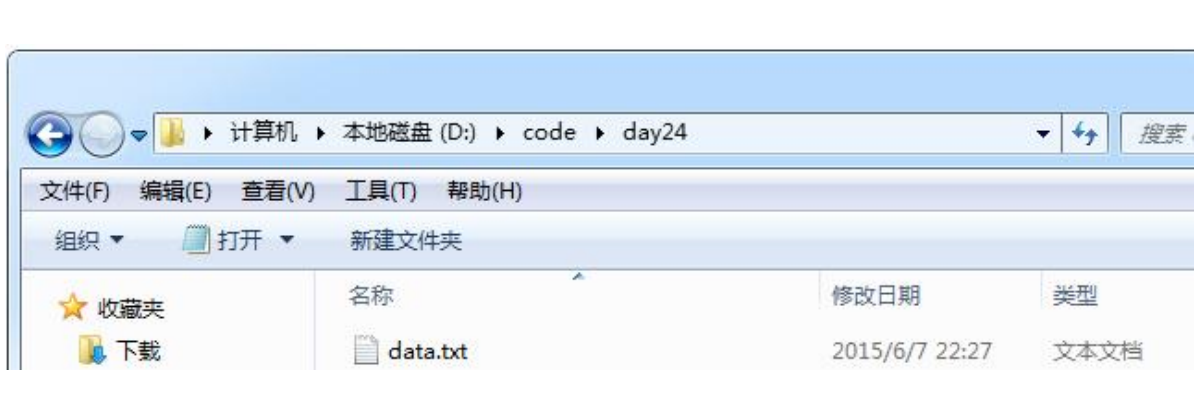
DataInputStream与DataOutputStream

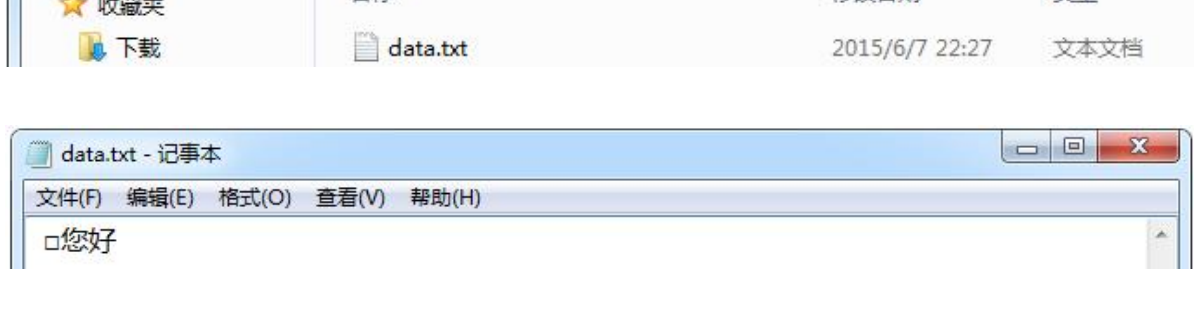
示例1：

```java
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class DataStreamDemo{
      public static void main(String[] args) throws IOException {
          writeData();
     }

      public static void writeData() throws IOException {
          DataOutputStream dos = new DataOutputStream(new 
FileOutputStream("data.txt" ));

          dos.writeUTF( "您好");

          dos.close();
     }
}
```
复制代码

运行结果：



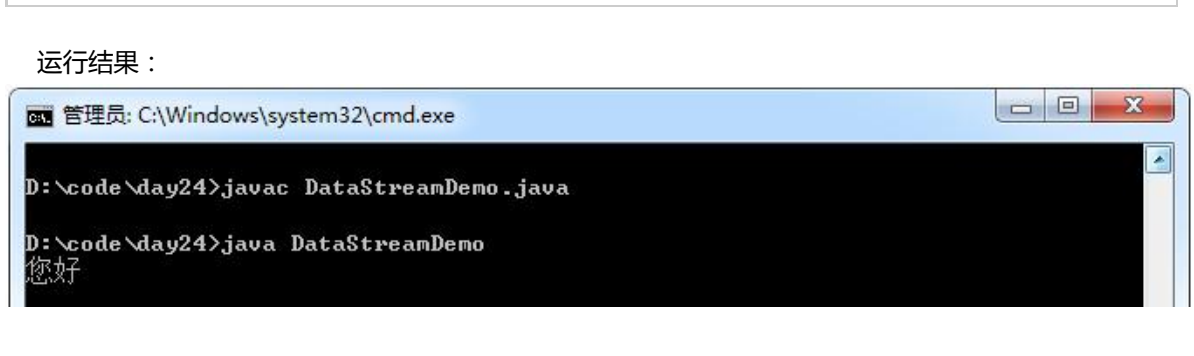




示例2：

```java
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;

public class DataStreamDemo{
       public static void main(String[] args) throws IOException {
            readData();
       }

       public static void readData() throws IOException {
            DataInputStream dis = new DataInputStream(new 
FileInputStream("data.txt" ));

            String str = dis.readUTF();

            System.out.println(str);

            dis.close();
       }
}
```
复制代码

运行结果：



操作字节数组

ByteArrayInputStream与ByteArrayOutputStream

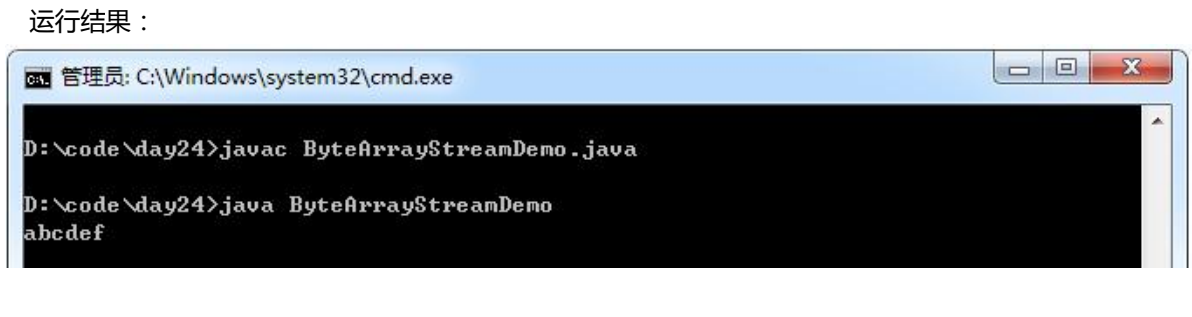
P.S.

关闭字节数组输出输出流无效，因为它们没有调用底层资源，所有的操作都是在内存中完成的。

示例1：

```java
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;

public class ByteArrayStreamDemo{
       public static void main(String[] args) throws IOException {
            ByteArrayInputStream bis = new ByteArrayInputStream("abcdef" 
.getBytes());
            ByteArrayOutputStream bos = new ByteArrayOutputStream();

             int ch = 0;

             while((ch = bis.read()) != -1){
                  bos.write(ch);
            }

            System.out.println(bos.toString());
       }
}
```
复制代码

运行结果：



8.2.12 编码表

编码表的由来

计算机只能识别二进制数据，早期由来是电信号。为了方便应用计算机，让它可以识别各个国家的文字。

就将各个国家的文字用数字来表示，并一一对应，形成一张表，这就是编码表。

常见的编码表

ASCII：美国标准信息交换码，用一个字节的7位可以表示。

ISO8859-1：拉丁码表。欧洲码表，用一个字节的8位表示。

GB2312：中国的中文编码表。

GBK：中国的中文编码表升级，融合了更多的中文文字符号。

Unicode：国际标准码，融合了多种文字。

所有文字都用两个字节来表示,Java语言使用的就是unicode

UTF-8：最多用三个字节来表示一个字符。

......

示例1：

```java
import java.io.IOException;

public class EncodeDemo{
       public static void main(String[] args) throws IOException{
             //字符串-->字节数组：编码
             //字符数组-->字符串：解码
            String str = "您好";

             //编码
             byte[] buf1 = str.getBytes("GBK" );
            printBytes(buf1);

             byte[] buf2 = str.getBytes("UTF-8" );
            printBytes(buf2);

             //解码
            String s1 = new String(buf1);
            System.out.println( "s1 = " + s1);

            String s2 = new String(buf2,"UTF-8" );
            System.out.println( "s2 = " + s2);
       }

       private static void printBytes(byte[] buf){
             for(byte b : buf){
                  System.out.print(b + " ");
            }
            System.out.println();
       }
}
```
复制代码

运行结果：

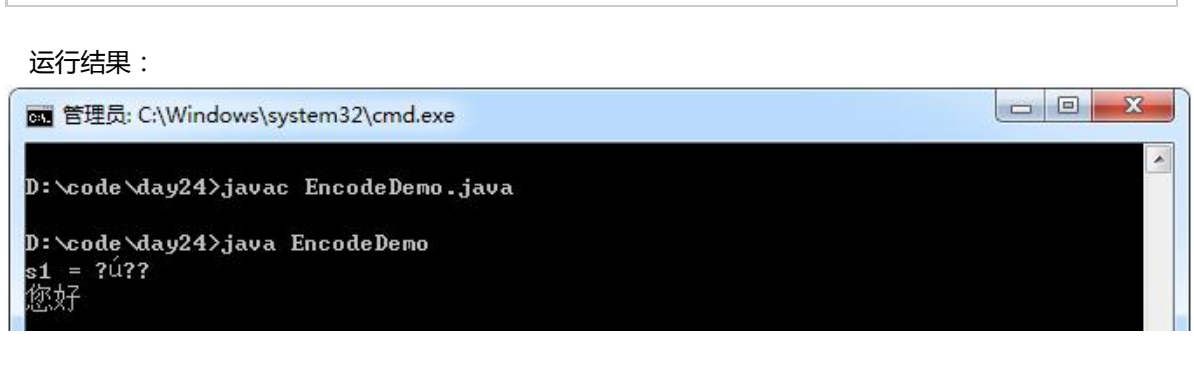
如果编码编错了，解不出来。

如果编对了，解错了，有可能有救。

示例2：

```java
import java.io.IOException;

public class EncodeDemo{
      public static void main(String[] args) throws IOException{
          String str = "您好";

           byte[] buf = str.getBytes("gbk" );

          String s1 = new String(buf,"iso8859-1" );

          System.out.println( "s1 = " + s1);

           byte[] buf2 = s1.getBytes("iso8859-1" );
          String s2 = new String(buf2,"gbk" );

          System.out.println(s2);
     }

      private static void printBytes(byte[] buf){
           for(byte b : buf){
                System.out.println(b);
           }
     }
}
```
复制代码

运行结果：

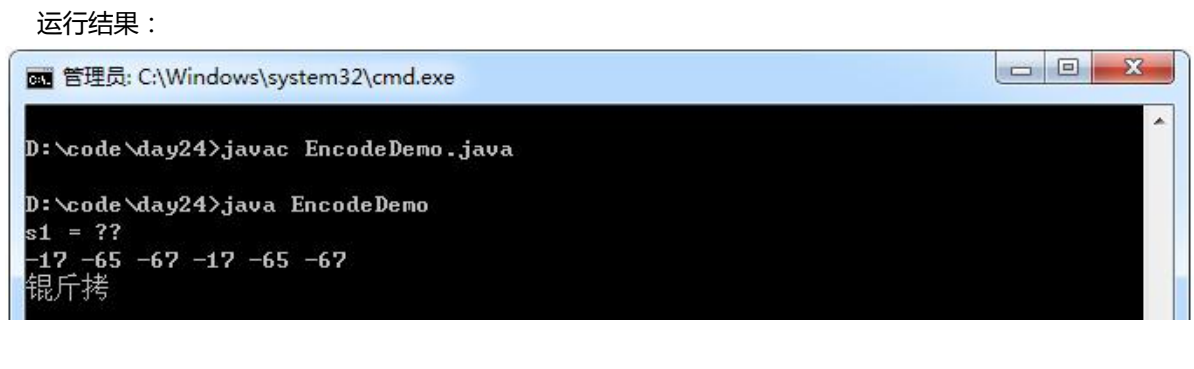


如果编对了，解错了，也可能没救了。

示例3：

```java
import java.io.IOException;

public class EncodeDemo{
       public static void main(String[] args) throws IOException{
            String str = "您好";

             byte[] buf = str.getBytes("gbk" );

            String s1 = new String(buf,"UTF-8" );

            System.out.println( "s1 = " + s1);

             byte[] buf2 = s1.getBytes("UTF-8" );

            printBytes(buf2);

            String s2 = new String(buf2,"gbk" );

            System.out.println(s2);
       }

       private static void printBytes(byte[] buf){
             for(byte b : buf){
                  System.out.print(b + " ");
            }
            System.out.println();
       }
}
```
复制代码

运行结果：



原因分析：

“您好”的gbk编码在UTF-8码表中查不到对应的字符，所以已经用“？”代替。

“？”在UTF-8中的编码为-17 -65 -67

故即使使用UTF-8码表进行解码，获取的字节也不是“您好”的gbk编码后的字节。

所以再也不能成功解码了。

P.S.

“谢谢”的gbk编码在UTF-8码表中可以查到对应的字符，为“ππ”。

因此，使用UTF-8码表对“ππ”进行解码，获取的字节也依然是“您好”的gbk编码后的字节。

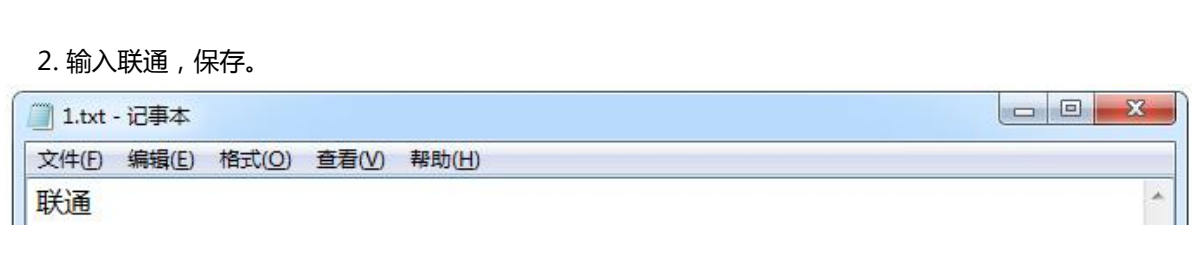
所以，不会出现“您好”发生的情况。

实验：联通乱码问题。

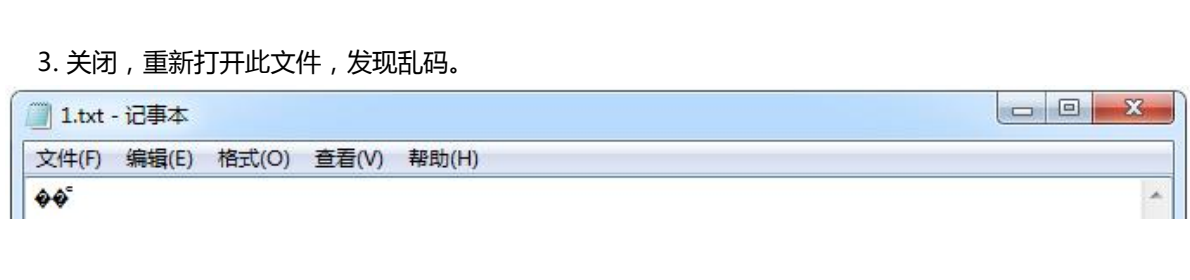
步骤：

1. 新建一个1.txt文件。

2. 输入联通，保存。



3. 关闭，重新打开此文件，发现乱码。



原因分析：

“联通”经过gbk编码后成四个字节：11000001、10101010、11001101、10101000。

正好符合UTF-8的编码规则。所以，记事本按照UTF-8进行了解码，从而出现了乱码现象。

练习：

在java中，字符串“abcd”与字符串“ab您好”的长度是一样，都是四个字符。

但对应的字节数不同，一个汉字占两个字节。

定义一个方法，按照最大的字节数来取子串。

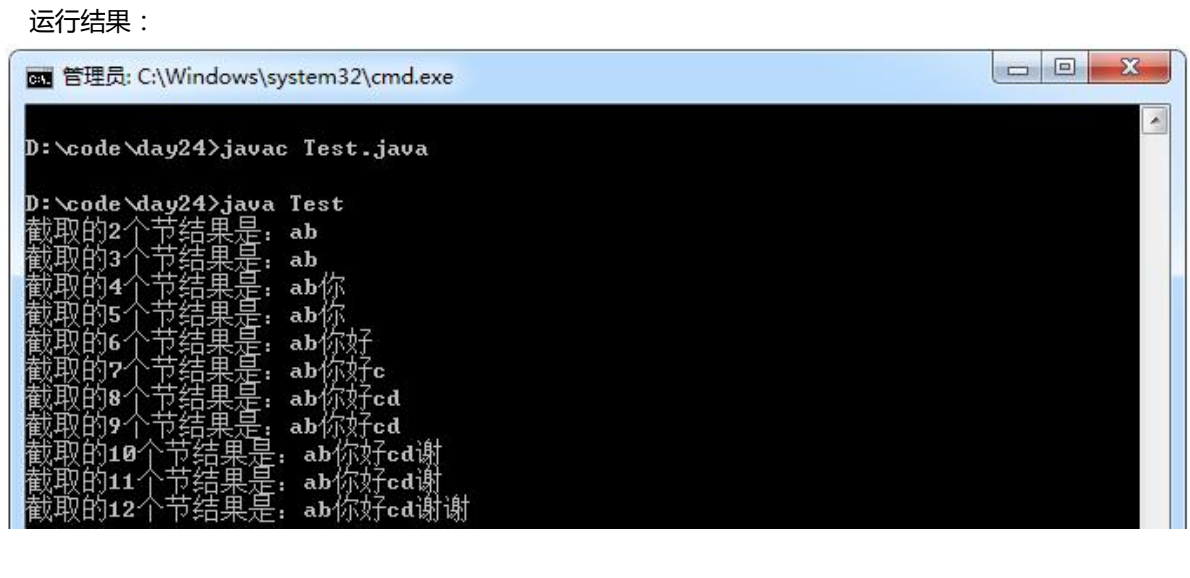
如：对于“ab你好”，如果取三个字节，那么子串就是ab与“你”字的半个。

那么半个就要舍弃。如果取四个字节就是“ab你”，取五个字节还是“ab你”。

代码：

```java
import java.io.IOException;

public class Test{
      public static void main(String[] args) throws IOException {
          String str = "ab你好cd谢谢" ;

           int len = str.getBytes("gbk" ).length;

           for(int x = 1; x < len; x++){
                System.out.println( "截取的" + (x + 1) + "个节结果是：" + 
cutStringByByte(str,x+1));
          }
     }

      public static String cutStringByByte(String str,int len) throws IOException {
           byte[] buf = str.getBytes("gbk" );

           int count = 0;
           for(int x = len - 1; x >= 0; x--){
                //gbk编码的值两个字节值一般都为负，记录连续的负数个数，如果为奇数，则舍
弃
                if(buf[x] < 0)
                    count++;
                else
                     break;
          }

           if(count % 2 == 0){
                return new String(buf,0,len,"gbk");
          } else{
                return new String(buf,0,len-1,"gbk");
          }
     }
}
```
复制代码

运行结果：



P.S.

中文经过gbk编码后，也有两个字节不都为负数的情况，例如“琲”，字节值为-84、105。

第一个字节值为负，第二个字节值为正。因此，上面的代码得出的结果依然正确。

定义一个方法，按照最大的字节数来取子串，使用UTF-8编码的代码如下：

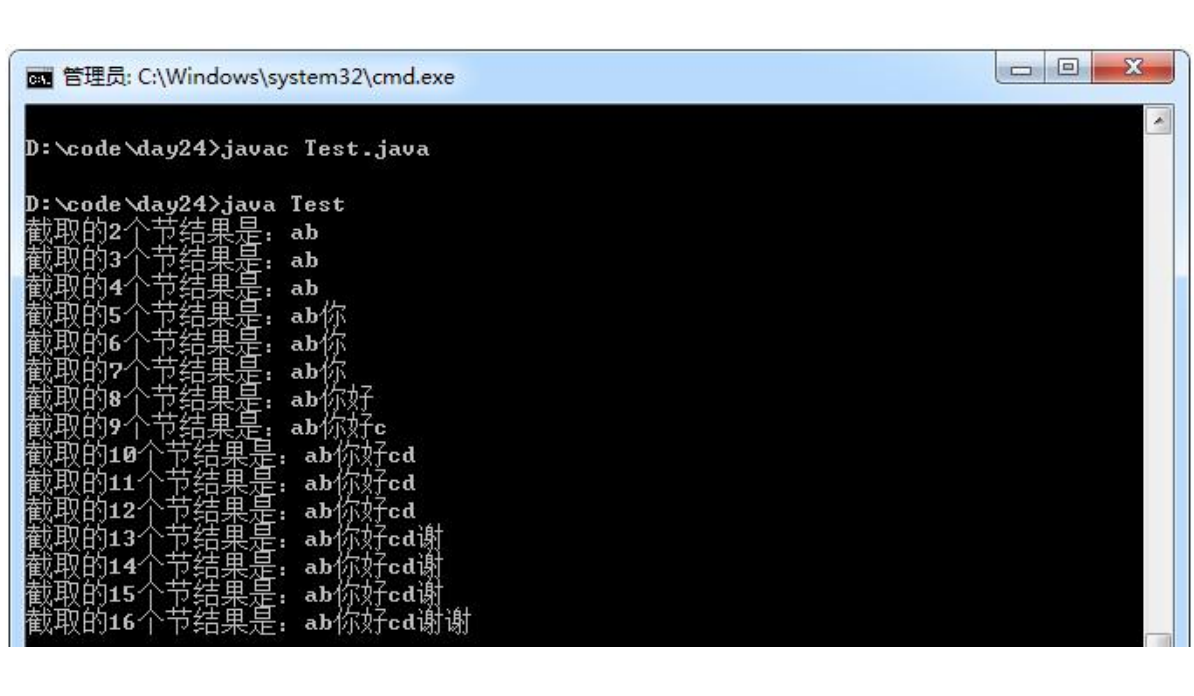
代码：

```java
public class Test{
       public static void main(String[] args) throws Exception {
            String str = "ab你好cd谢谢" ;

             int len = str.getBytes("utf-8" ).length;

             for(int x = 1; x < len; x++){
                  System.out.println( "截取的" + (x + 1) + "个节结果是：" + 
cutStringByByte(str,x+1));
            }
       }
```

```
        public static String cutStringByByte(String str,int len) throws Exception {
                byte[] buf = str.getBytes("utf-8" );

                int count = 0;
                for(int x = len - 1; x >= 0; x--){
                      if(buf[x] < 0)
                           count++;
                      else
                            break;
               }

                if(count % 3 == 0)
                      return new String(buf,0,len,"utf-8");
                else if (count % 3 == 1)
                      return new String(buf,0,len-1,"utf-8");
                else
                      return new String(buf,0,len-2,"utf-8");
        }
}
```

复制代码

运行结果：



~END~

~爱上海，爱黑马~