1.任务目标：使用字节流InputStream、OutputStream、BufferedInputStream、BufferedOutputStream类的读写方法对一个文件进行复制，并比较其运行效率，分析原因。

2.建立一个测试类并编写4个copy方法，在main方法中调用copy方法并进行时间运算。

源码：

**package xhu.edu.io;**

**import java.io.BufferedInputStream;**

**import java.io.BufferedOutputStream;**

**import java.io.FileInputStream;**

**import java.io.FileOutputStream;**

**import java.io.IOException;**

**import java.io.InputStream;**

**import java.io.OutputStream;**

**public class Test1 {**

**public static void main(String[] agrs) throws IOException {**

**String pathF = "d:/（贝多芬的悲伤）.mp3";**

**String pathT = "d:/（贝多芬的悲伤）copy.mp3";**

**InputStream in = new FileInputStream(pathF);**

**OutputStream out = new FileOutputStream(pathT);**

**long start = System.currentTimeMillis();**

**copy4(in, out);**

**long end1 = System.currentTimeMillis();**

**System.out.println("copy4用时" + (end1 - start) + "ms");**

**in.close();**

**out.close();**

**}**

**// 使用InputStream read（）单个字节读取方法进行读，OutputStream write（）进行写**

**public static void copy1(InputStream in, OutputStream out) throws IOException {**

**int b;**

**while ((b = in.read()) != -1) {**

**out.write(b);**

**}**

**}**

**// 使用InputStream read（byte[]）单个字节读取方法进行读，OutputStream write（byte[]）进行写**

**public static void copy2(InputStream in, OutputStream out) throws IOException {**

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

**while (in.read(b) != -1) {**

**out.write(b);**

**}**

**}**

**// 使用BufferedInputStream read（）单个字节读取方法进行读，BufferedOutputStream**

**// write（）单个字节进行写**

**public static void copy3(InputStream in, OutputStream out) throws IOException {**

**BufferedInputStream bufferedInputStream = new BufferedInputStream(in);**

**BufferedOutputStream bufferedOutputStream = new BufferedOutputStream(out);**

**int b;**

**while ((b = bufferedInputStream.read()) != -1) {**

**bufferedOutputStream.write(b);**

**}**

**}**

**/\***

**\* 使用BufferedInputStream read（byte[]）读取方法进行读，BufferedOutputStream**

**\* write（byte[]）进行写 问题：为什么使用read（byte[]）、 write（byte[]） 会与read(b, 0,**

**\* b.length)、write(b, 0, b.length)的速度相差无几?**

**\* 虽然前面是使用Filter的方法但是，后面是调用的是this.read(b, 0,b.length),**

**\* 也就是还是使用的buffered对象的(b, 0,b.length),看仔细点！！！！！！！！！**

**\*/**

**public static void copy4(InputStream in, OutputStream out) throws IOException {**

**BufferedInputStream bufferedInputStream = new BufferedInputStream(in);**

**BufferedOutputStream bufferedOutputStream = new BufferedOutputStream(out);**

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

**while (bufferedInputStream.read(b) != -1) {**

**bufferedOutputStream.write(b);**

**}**

**}**

**}**

运行截图：









结果分析：

（

首先要注意的几点：

基本字节输入输出流：

1，read()方法是读取一个字节（虽然读取的是字节但是返回的却是int），然后write()是写一个字节（虽然传入的是int但是写的却是字节），所以中间涉及到一个转化，通过查询源码得到将byte准确转化为int无影响的方法是将(byte&0xff)

2. read(byte[])方法是读取到自定义的数组中，然后write(byte[])是将指定数组中字节写到相应位置

高效缓冲输入输出流：（仅仅只是使用基本流加上缓冲区进行装饰了一下，如果不指定缓冲区大小默认为int最大值-8）

1，read()方法是先从输入流中读取到缓冲区，再从自带的缓冲区中读取一个字节中（虽然读取的是字节但是返回的却是int），然后write()是写一个字节到自带的缓冲区中，再将缓冲区中的输出（虽然传入的是int但是写的却是字节），中间仍然涉及到一个转化。

2. read(byte[])方法是先从输入流中读取到缓冲区，再从自带的缓冲区中读取到自定义的数组中，然后write(byte[])是将数组中的字节写到自带的缓冲区中，再将缓冲区中的输出将指定数组中字节写到相应位置。

所以使用缓冲流速度总体会快于使用基本流。而单个字节读取会比定义一个字节数组要快，因为一个一个字节进行读取会造成大量的系统开销，所以效率会下降，自定义字节数组相当于自定义了一个内存缓冲区。减少了系统开销，所以效率会高于单个字节读取。

）

注意：*Read()*方法每次只能读取一个字节，所以也只能读取由*ASCII*码范围内的一些字符。这些字符主要用于显示现代英语和其他西欧语言。而对于汉字等*unicode*中的字符则不能正常读取。只能以乱码的形式显示。

3.字符流的使用：

package xhu.edu.io;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.util.Arrays;

public class Test2 {

/\*

\* 经过测试证明了一些事情：

\* 1.eclipse的控制台输入输出的编码以及文件的编码是统一的（file.encoding），没有设置时，默认为系统编码GBK

\*

\*

\* 2.InputStreamReader读取字符的过程： 底层都是用字节流在流动，所以我们在键盘上输入的字符会先转化为字节，

\* 然后InputStreamReader会根据字节流，以及字符集（查看源码得没有设置则是file.encoding,即是eclipse文件编码）

\* 将字节转化为字符，注意后面并不涉及Java得Unicode的中间转换,

\* 最后转化出的字符串由OutputStreamWriter会根据指定的字符集转化为字节流输出到相应位置

\* 所以如果InputStreamReader指定的字符集如果本身就和字节流来源所采用的字符集不一样，那么读取的就是乱码，

\* 所以你最后无论用什么字符集进行输出，都应该是乱码

\* 反之，如果与字节流来源所采用的字符集一致，那么进行输出时采用的一致的字符集就能正确解码显示。

\*

\* （！！！！！！注意：因为我们使用的时String进行中间暂存，如果它使用的编码不一样会影响结果码？

\* 不会，因为字符的传递即使码表不同，也会自动进行字节的转化，字符不变，只要字符不变最后用相应的

\* 码表都是能转化回来的

\*

\* ）

\* （！！！如果在网络通信中，我们必须的准确的知道对方传输时使用字符集）

\*

\* 3.char是采用Java本身的Unicode编码，而String则是file.encoding。

\*

\* 4.BufferedReader.readLine()在读取时，会舍弃换行符，而BufferedWriter在写时，它并不会读一行写一行

\* 它会先将读取到的放在缓冲区中，然后统一写，一次最多会写缓冲区大小.

\*

\* 5.不同操作系统中的换行符不一致，windows中\n、\r,linux中\n,Mac中\r,所以需要newLine()来进行动态添加换行符

\*

\* 问题：为什么char不能转化为byte[]，如果通过String间接转化不会出现由编码所引起的错误吗?

\* char为两个字节，byte为一个字节，但是Java并不允许char直接转化为byte[],

\* 而需要将其转化为String后，才可以转化为byte[],这里需要注意一点

\* String字符的编码比较奇特，是不确定的，也就是不要将它与char(使用的是Unicode编码)等价，

\* 如果由char转化为String，最后出来的byte[]就是Unicode编码，

\* 如果是由文件中定义的常量字符串，转化出的byte[]是file.encoding.

\* 所以通过将char或者char[]转化String再转化为byte[]始终是Unicode编码

\*/

public static void main(String[] args) throws IOException {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in, "utf-8"));

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream("d:/a.txt"), "utf-8"));

System.out.println(System.getProperty("file.encoding"));

String string = null;

String sbString = "你好";

char[]a=new char[2];

a[0]='你';

a[1]='好';

byte b=(byte)a[0];

System.out.println(b);

System.out.println(Arrays.toString(a.toString().getBytes()));

System.out.println(Arrays.toString(sbString.getBytes()));

while (true) {

string = reader.readLine();

System.out.println(string);

System.out.println(Arrays.toString(string.getBytes()));

if (string.equals("--over--")) {

break;

} else {

writer.write(string);

writer.newLine();

}

}

reader.close();

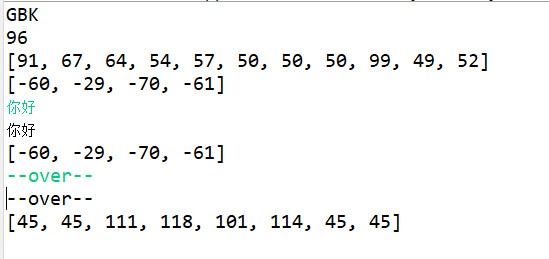
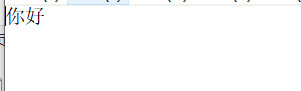
writer.close();

}

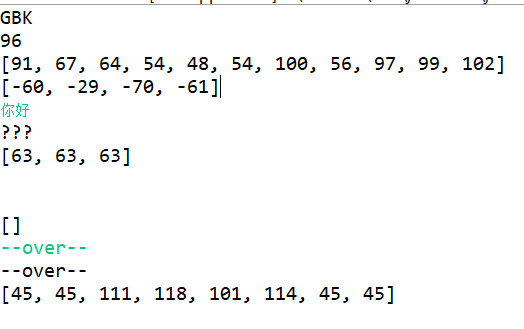
}

运行截图：

第一次测试使用默认的GBK编码，数据正常显示：



第二次测试使用UTF-8,结果乱码：





所以证实了读取时采用的编码需要与数据来源采用编码一致，否则读取的时候就是乱码，而写出时采用的编码最好是保持一致，否则遇见中文，即便是你显示端支持输出时的该编码，仍可能乱码。