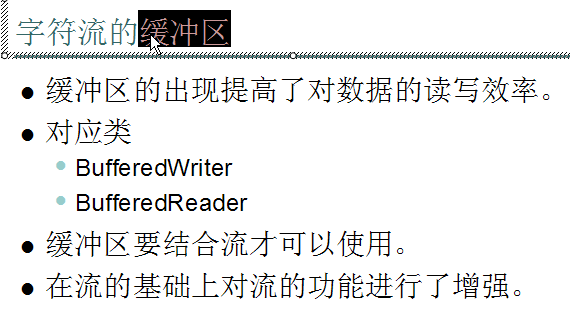
33

很多有读写操作的软件一般都有自己的缓冲区，如迅雷：



迅雷缓冲区的作用就是从网络上下载的数据先保存在内存中，达到缓存指定的大小时再一次性写入硬盘，这样可以提高性能，避免读一个字节写一个字节。就像搬东西一样，如果一样一样的搬会很慢，而如果先把东西放在一个箱子里，箱子装满了再搬这个箱子就会比较快。

查看BufferedWriter和BufferedReader这两个类，它们的构造函数都必须要传一个字符流进来，这说明缓冲流只是一个缓冲作用，它们读取文件或写文件最终还是要通过字符流来完成。

缓冲区的出现是为了提高流的操作效率而出现的。

所以在创建缓冲区之前，必须要先有流对象。

该缓冲区中提供了一个跨平台的换行符。

newLine();

实例1：

//创建一个字符写入流对象。

FileWriter fw = new FileWriter("buf.txt");

//为了提高字符写入流效率。加入了缓冲技术。

//只要将需要被提高效率的流对象作为参数传递给缓冲区的构造函数即可。

BufferedWriter bufw = new BufferedWriter(fw);

for(int x=1; x<5; x++)

{

bufw.write("abcd"+x);

bufw.newLine();

bufw.flush(); // 为了预防停电等意外情况，所以写一次刷新一次

}

//记住，只要用到缓冲区，就要记得刷新。

//bufw.flush();

//其实关闭缓冲区，就是在关闭缓冲区中的流对象(fw)，所以不需要再调用fw.close()。

bufw.close();

实例2：

字符读取流缓冲区：

该缓冲区提供了一个一次读一行的方法 readLine，方便于对文本数据的获取。

当返回null时，表示读到文件末尾。

readLine方法返回的时候只返回回车符之前的数据内容。并不返回回车符。

//创建一个读取流对象和文件相关联。

FileReader fr = new FileReader("buf.txt");

//为了提高效率。加入缓冲技术。将字符读取流对象作为参数传递给缓冲对象的构造函数。

BufferedReader bufr = new BufferedReader(fr);

String line = null;

while((line=bufr.readLine())!=null) {

System.out.print(line);

}

bufr.close();

实例3：

用缓存流实现文件复制

BufferedReader bufr = null;

BufferedWriter bufw = null;

try {

bufr = new BufferedReader(new FileReader("BufferedWriterDemo.java"));

bufw = new BufferedWriter(new FileWriter("bufWriter\_Copy.txt"));

String line = null;

while((line=bufr.readLine())!=null) {

bufw.write(line);

bufw.newLine();

bufw.flush();

}

}

catch (IOException e) {

throw new RuntimeException("读写失败");

} finally {

try {

if(bufr!=null)

bufr.close();

} catch (IOException e) {

throw new RuntimeException("读取关闭失败");

}

try {

if(bufw!=null)

bufw.close();

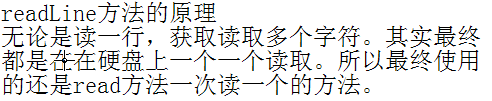
} catch (IOException e) {

throw new RuntimeException("写入关闭失败");

}

}

# readLine原理



/\*

明白了BufferedReader类中特有方法readLine的原理后，

可以自定义一个类中包含一个功能和readLine一致的方法。

来模拟一下BufferedReader

\*/

import java.io.\*;

class MyBufferedReader extends Reader {

private Reader r;

MyBufferedReader(Reader r) {

this.r = r;

}

//可以一次读一行数据的方法。

public String myReadLine()throws IOException {

//定义一个临时容器。原BufferReader封装的是字符数组。

//为了演示方便。定义一个StringBuilder容器。因为最终还是要将数据变成字符串。

StringBuilder sb = new StringBuilder();

int ch = 0;

while((ch=r.read())!=-1) {

if(ch=='\r')

continue;

if(ch=='\n')

return sb.toString();

else

sb.append((char)ch);

}

if(sb.length()!=0) // 如果最后一行没有按回车进行换行，则上面不会识别为一行，所以这里把没有换行标志的内容全部返回

return sb.toString();

return null;

}

/\*

覆盖Reader类中的抽象方法。

\*/

public int read(char[] cbuf, int off, int len) throws IOException {

return r.read(cbuf,off,len) ;

}

public void close()throws IOException {

r.close();

}

public void myClose()throws IOException {

r.close();

}

}

class MyBufferedReaderDemo {

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

FileReader fr = new FileReader("buf.txt");

MyBufferedReader myBuf = new MyBufferedReader(fr);

String line = null;

while((line=myBuf.myReadLine())!=null) {

System.out.println(line);

}

myBuf.myClose();

}

}

这里BufferedReader就是增加了Reader的功能，因为读的操作最终是调用Reader来完成的，像这种就是一种设计模式—装饰设计模式

# 装饰设计模式

IO体系中就使用了这个模式，如BufferedReader、BufferedWriter等类

/\*

装饰设计模式：

当想要对已有的对象进行功能增强时，

可以定义类，将已有对象传入，基于已有的功能，并提供加强功能。

那么自定义的该类称为装饰类。

装饰类通常会通过构造方法接收被装饰的对象。

并基于被装饰的对象的功能，提供更强的功能。

\*/

class Person {

public void chifan() {

System.out.println("吃饭");

}

}

如上定义了一个人的吃饭方法，假如说多年以后人们的生活水平提高了，吃饭的方法也提高了，这时候不能直接修改吃饭方法，因为这样的话会对原来老代码使用这个类的类产生影响，所以这个时候就可以使用一个增加强类来完成。

class SuperPerson {

private Person p ;

SuperPerson(Person p) {

this.p = p;

}

public void superChifan() {

System.out.println("开胃酒");

p.chifan();

System.out.println("甜点");

System.out.println("来一根");

}

}

class PersonDemo {

public static void main(String[] args) {

Person p = new Person();

//p.chifan();

SuperPerson sp = new SuperPerson(p); // 人还是原来那个人，但是吃饭的功能就增加了！！

sp.superChifan();

}

}

这里SuperPerson为什么不直接继承Person呢，这样的话SuperPerson的chifan方法中就可以调用super.chifan()方法了，原因如下：

假如有两个类如下：

MyTextReader 用于读取文本文件

MyMediaReader 用于读取多媒体文件

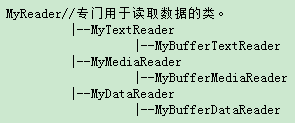
因为都有共同的特性读取，所以抽取了一个父类为MyReader，这时候的体系为:

MyReader

|--MyTextReader

|--MyMediaReader

这时候觉得MyTextReader和MyMediaReader的读取效率太低，想要对这两个类进行增强增加缓冲，如果使用继承的话，则这时候体系为：



这样的问题就是要为每个都原始类都增加一个Buffered类型的类，如果用传构造函数的方式就比较好：

class MyBufferReader {

MyBufferReader(MyTextReader text){}

MyBufferReader(MyMediaReader media){}

}

上面这个类扩展性很差。

找到其参数的共同类型。通过多态的形式。可以提高扩展性。

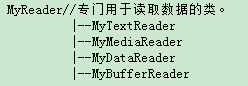
class MyBufferReader extends MyReader { // 为了让MyBufferReader也属性MyReader体系，所以继承MyReader类。

private MyReader r;

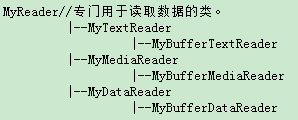
MyBufferReader(MyReader r){}

}

所以此时的MyReader体系为：



这个体系就要比下面的体系好很多：



以前是通过继承将每一个子类都具备缓冲功能,那么继承体系会复杂，并不利于扩展。

现在优化思想,单独描述一下缓冲内容,将需要被缓冲的对象传递进来,也就是，谁需要被缓冲，谁就作为参数传递给缓冲区。

这样继承体系就变得很简单,优化了体系结构。

装饰模式比继承要灵活，避免了继承体系臃肿，而且降低了类于类之间的关系。

装饰类因为增强已有对象，具备的功能和已有的是相同的，只不过提供了更强功能，所以装饰类和被装饰类通常是都属于一个体系中的。

# LineNumberReade

这也是一个包装类

class LineNumberReaderDemo {

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

FileReader fr = new FileReader("PersonDemo.java");

LineNumberReader lnr = new LineNumberReader(fr);

String line = null;

lnr.setLineNumber(100); // 设置起始编号，也可以不设置

while((line=lnr.readLine())!=null) {

System.out.println(lnr.getLineNumber()+":"+line);

}

lnr.close();

}

}

//练习：模拟一个带行号的缓冲区对象。

**class** LineNumberReader **extends** Reader{

**private** BufferedReader bufReader;

**private** **int** lineNumber;

**public** LineNumberReader(Reader reader) {

**super**();

**if** (reader == **null**) {

**throw** **new** RuntimeException("构造参数不能为null");

}

**if** (reader **instanceof** BufferedReader) {

bufReader = (BufferedReader) reader;

} **else** {

bufReader = **new** BufferedReader(reader);

}

}

**public** **int** getLineNumber() {

**return** lineNumber;

}

**public** **void** setLineNumber(**int** lineNumber) {

**this**.lineNumber = lineNumber;

}

**public** String readLine() **throws** IOException {

lineNumber++;

**return** bufReader.readLine();

}

@Override

**public** **int** read(**char**[] cbuf, **int** off, **int** len) **throws** IOException {

**return** bufReader.read(cbuf, off, len);

}

@Override

**public** **void** close() **throws** IOException {

bufReader.close();

}

}

也可以直接继承BufferedReader，如下：

**class** LineNumberReader **extends** BufferedReader{

**private** **int** lineNumber;

**public** LineNumberReader(Reader reader) {

**super**(reader);

}

**public** **int** getLineNumber() {

**return** lineNumber;

}

**public** **void** setLineNumber(**int** lineNumber) {

**this**.lineNumber = lineNumber;

}

**public** String readLine() **throws** IOException {

lineNumber++;

**return** **super**.readLine(); // 注，这里一定要加上super，不然就是自己调用自己，形成死循环

}

}

# 字节流

InputStream有read()方法用于读取一个字节，也有read(byte[] buf)一次读取多个，但是这个方法内部也是调用read()方法一个一个读的，然后保存到buf数组中。

InputStream OutputStream

需求，想要操作图片数据。这时就要用到字节流。

复制一个图片.

\*/

import java.io.\*;

class FileStream

{

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

{

readFile\_3();

}

public static void readFile\_3()throws IOException

{

FileInputStream fis = new FileInputStream("fos.txt");

// int num = fis.available();

byte[] buf = new byte[fis.available()];//定义一个刚刚好的缓冲区。不用在循环了。注：用这种方式要小心，如读一个几个G的文件，这内存就溢出了，因为虚拟机启动的时候分配的虚拟机内存为64M，所以这个方法用于读取比较小的文件还可以，读大文件就不行了。

fis.read(buf); // 注：这里不需要调用flush方法数据就写到文件里去了，字节流都不需要调用flush，为什么字节流需要呢？因为字符流也是对字节流进行了包装，中文一个字符是两个字节，字符流读取到一个字节的时候是不能进行操作，需要读取完之后，再去查找码表，这样才知道两个字节是一个汉字还是只是两个普通字母。

System.out.println(new String(buf));

fis.close();

}

public static void readFile\_2()throws IOException

{

FileInputStream fis = new FileInputStream("fos.txt");

byte[] buf = new byte[1024];

int len = 0;

while((len=fis.read(buf))!=-1)

{

System.out.println(new String(buf,0,len));

}

fis.close();

}

public static void readFile\_1()throws IOException

{

FileInputStream fis = new FileInputStream("fos.txt");

int ch = 0;

while((ch=fis.read())!=-1)

{

System.out.println((char)ch);

}

fis.close();

}

public static void writeFile()throws IOException

{

FileOutputStream fos = new FileOutputStream("fos.txt");

fos.write("abcde".getBytes());

fos.close();

}

}

# 实例：复制图片

/\*

复制一个图片

思路：

1，用字节读取流对象和图片关联。

2，用字节写入流对象创建一个图片文件。用于存储获取到的图片数据。

3，通过循环读写，完成数据的存储。

4，关闭资源。

\*/

import java.io.\*;

class CopyPic {

public static void main(String[] args) {

FileOutputStream fos = null;

FileInputStream fis = null;

try {

fos = new FileOutputStream("c:\\2.bmp");

fis = new FileInputStream("c:\\1.bmp");

byte[] buf = new byte[1024];

int len = 0;

while((len=fis.read(buf))!=-1) {

fos.write(buf,0,len);

}

} catch (IOException e) {

throw new RuntimeException("复制文件失败");

} finally {

try {

if(fis!=null)

fis.close();

} catch (IOException e) {

throw new RuntimeException("读取关闭失败");

}

try {

if(fos!=null)

fos.close();

} catch (IOException e) {

throw new RuntimeException("写入关闭失败");

}

}

}

}

# 实例：通过缓冲类复制mp3

/\*

演示mp3的复制。通过缓冲区。

BufferedOutputStream

BufferedInputStream

\*/

import java.io.\*;

class CopyMp3{

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

long start = System.currentTimeMillis();

copy\_2();

long end = System.currentTimeMillis();

System.out.println((end-start)+"毫秒");

}

public static void copy\_2()throws IOException{

MyBufferedInputStream bufis = new MyBufferedInputStream(new FileInputStream("c:\\9.mp3"));

BufferedOutputStream bufos = new BufferedOutputStream(new FileOutputStream("c:\\3.mp3"));

int by = 0;

//System.out.println("第一个字节:"+bufis.myRead());

while((by=bufis.myRead())!=-1) {

bufos.write(by);

}

bufos.close();

bufis.myClose();

}

//通过字节流的缓冲区完成复制。

public static void copy\_1()throws IOException{

BufferedInputStream bufis = new BufferedInputStream(new FileInputStream("c:\\0.mp3"));

BufferedOutputStream bufos = new BufferedOutputStream(new FileOutputStream("c:\\1.mp3"));

int by = 0;

while((by=bufis.read())!=-1){ // 这里的bufis实际上一次从硬盘读取了好多数据，这里调用bufis.read()只是从读取的缓存中返回了一个，当缓存中的都取完之后bufis再去硬盘一次读一堆数据放在缓冲区。

bufos.write(by); // 这里的bufos.write(by)是往缓存区里存了1个字节，这时并没有写入硬盘，只在往缓冲区里存了足够多的数据的时候才把这些数据一次性写入硬盘，或者在调用close方法的时候才写入硬盘。

}

bufos.close();

bufis.close();

}

}

# 自定义字节缓冲类

class MyBufferedInputStream {

private InputStream in;

private byte[] buf = new byte[1024\*4];

private int pos = 0,count = 0;

MyBufferedInputStream(InputStream in) {

this.in = in;

}

//一次读一个字节，从缓冲区(字节数组)获取。

public int myRead()throws IOException {

//通过in对象读取硬盘上数据，并存储buf中。

if(count==0){

count = in.read(buf);

if(count<0)

return -1;

pos = 0;

byte b = buf[pos];

count--;

pos++;

return b&255;

}else if(count>0){

byte b = buf[pos];

count--;

pos++;

return b&0xff; // 0xff即为255

}

return -1;

}

public void myClose()throws IOException{

in.close();

}

}

# 为什么read()方法返回int而不是byte

上面的代码在用read()方法读一个字节的时候为什么不直接返回byte类型而是返回了int类型呢？

查看InputStream的read()方法返回的也是int类型，它读的是一个字节的内容为什么不是返回byte呢？因为读取到文件的末尾是用-1来表示的，如果文件中本来就有内容为-1，则就会出问题了，所以使用返回int类型就可以解决这个问题：

读的时候提升为了int，数据就会有8位二进制数变化32位二进制数

如byte的-1为：1111-1111，提升为int为：1111-1111-1111-1111-1111-1111-1111-1111

所以在提升为int之后，我们需要的内容只是int数据的最后8个二进制位，我们看OutputStream有一个[**write**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/OutputStream.html#write(int))(int b)方法，它接收的是int类型，但是写到硬盘的时候其实只会写入int数据的最后8个二进制（即(byte) int强转即可取出int的后8位），虽然读一个byte返回了一个int，但是写的时候只写了int的后8位，所以不会有问题。

反回int的时候，如果我们把除了后8位之外的二进制位都设置为0的话，那这个int就不会有负数的情况出现，这就是为什么需要read()方法返回int而不返回byte的原因。

11111111 11111111 11111111 11111111 （这是-1的二进制）

&00000000 00000000 00000000 11111111 （这是255的二进制）

------------------------------------

00000000 00000000 00000000 11111111

结论：

字节流的读一个字节的read方法为什么返回值类型不是byte，而是int。

因为有可能会读到连续8个二进制1的情况，8个二进制1对应的十进制是-1.

那么就会数据还没有读完，就结束的情况。因为我们判断读取结束是通过结尾标记-1来确定的。

所以，为了避免这种情况将读到的字节进行int类型的提升。

并在保留原字节数据的情况前面了补了24个0，变成了int类型的数值。

而在写入数据时，只写该int类型数据的最低8位。

# 键盘录入-屏幕输出

FileInputStream-FileOutputStream和FileReader-FileWriter对应了文件的读和写，那么它们的读的源是文件，写的源也是文件，如果这些源不是文件呢？

如输入源从键盘获取，输出源是输出到屏幕上，我们应该怎样去创建这些源对应的IO流对象呢，其实不需要我们去创建，Java的api中已经提供了相应的对象：

System.out:对应的是标准输出设备：控制台

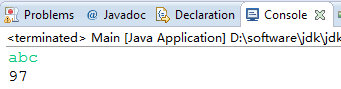
System.in: 对应的标准输入设备：键盘

InputStream in = System.*in*;

**int** by1 = in.read();

System.*out*.println(by1);

在控制台输入“abc”，然后按下回车键，结果如下：



如果想要读取更多的数据，则可以多次调用read()方法，如下：

InputStream in = System.*in*;

**int** by1 = in.read();

**int** by2 = in.read();

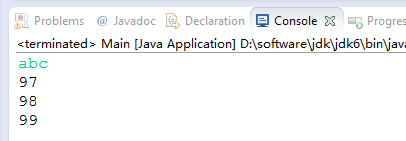
**int** by3 = in.read();

System.*out*.println(by1);

System.*out*.println(by2);

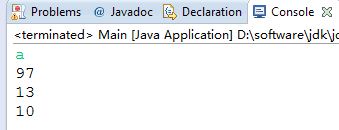
System.*out*.println(by3);

在控制台输入“abc”，然后按下回车键，结果如下：



字母a、b、c对应的ASCII码分别为97、98、99

再次运行程序，此时只输入字母“a”，再按下回车，结果如下：

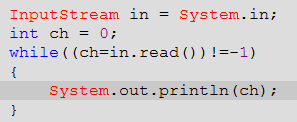


13和10分别是\r和\n的ASCII码，验证如下：

System.*out*.println('\r' == 13);

System.*out*.println('\n' == 10);

结果为true



运行这一段代码，发现会循环的要求录入并输出，如何结束呢？输入-1再按回车会发现不管用，因为-1读取的时候是读取了两个字符，在命令行中可以按Ctrl + C停止命令行的执行。

实例：

/\*

需求：

通过键盘录入数据。

当录入一行数据后，就将该行数据进行打印。

如果录入的数据是over，那么停止录入。

\*/

import java.io.\*;

class ReadIn{

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

InputStream in = System.in;

StringBuilder sb = new StringBuilder();

while(true){

int ch = in.read();

if(ch=='\r')

continue;

if(ch=='\n'){

String s = sb.toString();

if("over".equals(s))

break;

System.out.println(s.toUpperCase());

sb.delete(0,sb.length());

} else

sb.append((char)ch);

}

}

}

通过刚才的键盘录入一行数据并打印其大写，发现其实就是读一行数据的原理。

也就是readLine方法。

能不能直接使用readLine方法来完成键盘录入的一行数据的读取呢？

readLine方法是字符流BufferedReader类中的方法。

而键盘录入的read方法是字节流InputStream的方法。

那么能不能将字节流转成字符流在使用字符流缓冲类的readLine方法呢？通过InputStreamReader即可，那它应该属于字节流还是属于字符流？我们知道字节流的底层其实就是使用字节流来读取的，InputStreamReader也不例外，并且它的功能就是用来操作字符的，所以InputStreamReader属于字符流体系。从类名也可看出，因为类名的前面是功能名，后面是体系名。

class TransStreamDemo

{

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

{

//获取键盘录入对象。

//InputStream in = System.in;

//将字节流对象转成字符流对象，使用转换流。InputStreamReader

//InputStreamReader isr = new InputStreamReader(in);

//为了提高效率，将字符串进行缓冲区技术高效操作。使用BufferedReader

//BufferedReader bufr = new BufferedReader(isr);

//键盘的最常见写法。

BufferedReader bufr = new BufferedReader(new InputStreamReader(System.in));

// OutputStream out = System.out;

// OutputStreamWriter osw = new OutputStreamWriter(out);

// BufferedWriter bufw = new BufferedWriter(osw);

BufferedWriter bufw = new BufferedWriter(new OutputStreamWriter(System.out));

String line = null;

while((line=bufr.readLine())!=null) {

if("over".equals(line))

break;

bufw.write(line.toUpperCase());

bufw.newLine();

bufw.flush();

}

bufr.close();

}

}

这是一个比较经常用的操作，就是把一个源的数据写到另一个目的中，所以上面的代码可以抽取成方法，方面复用，如下：

**public** **class** Main {

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

// 将键盘录入的内容用大写的方式显示到控制台

//readAndWrite(System.in, System.out);

// 将文件中的内容用大写的方式显示到控制台

//InputStream in = new FileInputStream("D:\\Main.java");

//readAndWrite(in, System.out);

// 将键盘录入的内容用大写的方式写入文件

OutputStream out = **new** FileOutputStream("D:\\haha.txt");

*readAndWrite*(System.*in*, out);

}

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

BufferedReader bufr = **new** BufferedReader(**new** InputStreamReader(in));

BufferedWriter bufw = **new** BufferedWriter(**new** OutputStreamWriter(out));

String line = **null**;

**while** ((line = bufr.readLine()) != **null**) {

**if** ("over".equals(line))

**break**;

bufw.write(line.toUpperCase());

bufw.newLine();

bufw.flush();

}

bufr.close();

}

}

/\*

流操作的基本规律：

最痛苦的就是流对象有很多，不知道该用哪一个。

通过三个明确来完成。

1，明确源和目的。

源：输入流。InputStream Reader

目的：输出流。OutputStream Writer。

2，操作的数据是否是纯文本。

是：字符流。

不是：字节流。

3，当体系明确后，在明确要使用哪个具体的对象。

通过设备来进行区分：

源设备：内存，硬盘。键盘

目的设备：内存，硬盘，控制台。

1，将一个文本文件中数据存储到另一个文件中。复制文件。

源：因为是源，所以使用读取流。InputStream Reader

是不是操作文本文件。

是！这时就可以选择Reader

这样体系就明确了。

接下来明确要使用该体系中的哪个对象。

明确设备：硬盘。上一个文件。

Reader体系中可以操作文件的对象是 FileReader

是否需要提高效率：是！。加入Reader体系中缓冲区 BufferedReader.

FileReader fr = new FileReader("a.txt");

BufferedReader bufr = new BufferedReader(fr);

目的：OutputStream Writer

是否是纯文本。

是！Writer。

设备：硬盘，一个文件。

Writer体系中可以操作文件的对象FileWriter。

是否需要提高效率：是！。加入Writer体系中缓冲区 BufferedWriter

FileWriter fw = new FileWriter("b.txt");

BufferedWriter bufw = new BufferedWriter(fw);

练习：将一个图片文件中数据存储到另一个文件中。复制文件。要按照以上格式自己完成三个明确。

---------------------------------------

2，需求：将键盘录入的数据保存到一个文件中。

这个需求中有源和目的都存在。

那么分别分析

源：InputStream Reader

是不是纯文本？是！Reader

设备：键盘。对应的对象是System.in.

不是选择Reader吗？System.in对应的不是字节流吗？

为了操作键盘的文本数据方便。转成字符流按照字符串操作是最方便的。

所以既然明确了Reader，那么就将System.in转换成Reader。

用了Reader体系中转换流,InputStreamReader

InputStreamReader isr = new InputStreamReader(System.in);

需要提高效率吗？需要！BufferedReader

BufferedReader bufr = new BufferedReader(isr);

目的：OutputStream Writer

是否是存文本？是！Writer。

设备：硬盘。一个文件。使用 FileWriter。

FileWriter fw = new FileWriter("c.txt");

需要提高效率吗？需要。

BufferedWriter bufw = new BufferedWriter(fw);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

扩展一下，可以识别中文的编码有GBK和UTF-8，想要把录入的数据按照指定的编码表（utf-8），将数据存到文件中。

目的：OutputStream Writer

是否是存文本？是！Writer。

设备：硬盘。一个文件。使用 FileWriter。

但是FileWriter是使用的默认编码表。GBK.

但是存储时，需要加入指定编码表utf-8。而指定的编码表只有转换流可以指定。

所以要使用的对象是OutputStreamWriter。

而该转换流对象要接收一个字节输出流，而且是可以操作文件的字节输出流，所以用：FileOutputStream

OutputStreamWriter osw = new OutputStreamWriter(new FileOutputStream("d.txt"),"UTF-8");

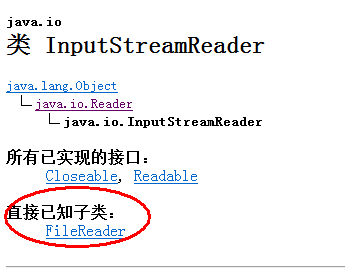
需要高效吗？需要。

BufferedWriter bufw = new BufferedWriter(osw);

所以，记住，转换流什么时候用：字符和字节之间的桥梁，通常涉及到字符编码转换时需要用到转换流。

练习：将一个文本数据打印在控制台上。要按照以上格式自己完成三个明确。

\*/



如上图，发现FileReader是InputStream的子类，因为FileReader是操作字符的，涉及到字符的编码表，所以它继承了转换流，但是不幸的是，FileReader中使用的字符编码默认写死了用本地编码，无法修改。所以如果要使用UTF-8编辑的话还是得使用转换流。如果new InputStreamReader时指定的是GBK编码，则这个对象可以说功能和FileReader就是一样的了。

FileWriter也一样，它也继承了转换流，而且只是使用了默认平台编码而已，没有什么特殊的。

假如使用转换流写了一个文件，如：new OutputStreamWriter(new FileOutputStream(“D:\\haha.txt”), “UTF-8”);

则，这个haha.txt文件使用FileReader来读取将会是乱码，因为FileReader是写死用本地码表（GBK）来读取的，

所以这个时候应该使用转换流来读取：new InputStreamReader(new FileInputStream(“D:\\haha.txt”), “UTF-8”);

使用GBK写入“你好”，则文件是6字节大小，如果用UTF-8写入“你好”，则文件是8字节大小。所以操作字符的时候一般要指定码表，如GBK的中文件，指定GBK码表后则读取的时候就会读取3个字节，然后把这3个字节拿去查找GBK码表，这样就能查到正确的中文字符，如果指定UTF-8，则是读取4个字节去查UTF-8码表。

# 改变标准输入、输出的源

默认的标准输入是键盘，输出是控制台，其实只是可以改变的，如下：

System.setIn(new FileInputStream("PersonDemo.java"));

System.setOut(new PrintStream("zzz.txt"));

//键盘的最常见写法。

BufferedReader bufr = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufw = new BufferedWriter(new OutputStreamWriter(System.out));

String line = null;

while((line=bufr.readLine())!=null) {

if("over".equals(line))

break;

bufw.write(line.toUpperCase());

bufw.newLine();

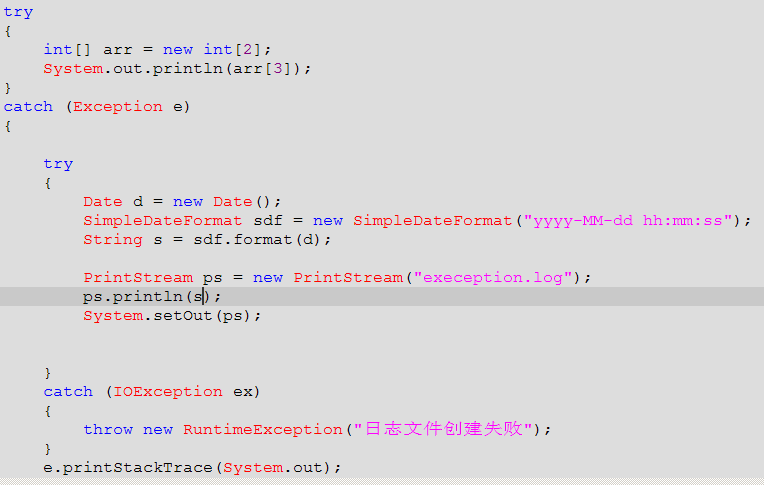
bufw.flush();

}

bufr.close();

# 保存异常信息

项目开发中，把异常信息打印到控制台没有用，看不见，所以最好保存到文件里，如下：



一般写异常信息使用log4j即可，这个更方便更专业。

查看Throwable类如有下方法：

|  |  |
| --- | --- |
| void | [**printStackTrace**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/lang/Throwable.html#printStackTrace())()  将此 throwable 及其追踪输出至标准错误流。 |
| void | [**printStackTrace**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/lang/Throwable.html#printStackTrace(java.io.PrintStream))([PrintStream](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/PrintStream.html) s)  将此 throwable 及其追踪输出到指定的输出流。 |
| void | [**printStackTrace**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/lang/Throwable.html#printStackTrace(java.io.PrintWriter))([PrintWriter](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/PrintWriter.html) s)  将此 throwable 及其追踪输出到指定的 PrintWriter。 |

System.out是一个PrintStream，其实printStackTrace()方法内部就是调用了printStackTrace(System.out)来把异常信息打印到控制台的。

# Properties与流结合使用

系统属性中的file.encoding=UTF-8属性为平台的默认编码

Properties prop = System.*getProperties*();

System.*out*.println(prop);// 这样打印的系统信息是不换行的

|  |  |
| --- | --- |
| void | [**list**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Properties.html#list(java.io.PrintStream))([PrintStream](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/PrintStream.html) out)  将属性列表输出到指定的输出流。 |
| void | [**list**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Properties.html#list(java.io.PrintWriter))([PrintWriter](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/PrintWriter.html) out)  将属性列表输出到指定的输出流。 |
| void | [**load**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Properties.html#load(java.io.InputStream))([InputStream](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/InputStream.html) inStream)  从输入流中读取属性列表（键和元素对）。 |
| void | [**load**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Properties.html#load(java.io.Reader))([Reader](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/Reader.html) reader)  按简单的面向行的格式从输入字符流中读取属性列表（键和元素对）。 |
| void | [**loadFromXML**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Properties.html#loadFromXML(java.io.InputStream))([InputStream](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/InputStream.html) in)  将指定输入流中由 XML 文档所表示的所有属性加载到此属性表中。 |

我们的System.out是一个PrintStream，因此可以使用list(PrintStream out)方法来将系统信息打印在控制上：

Properties prop = System.*getProperties*();

System.*out*.println(prop);

prop.list(System.*out*);

也可以保存到文件中：

prop.list(**new** PrintStream("sysinfo.txt"));