**字节流:InputStream OutputStream**

**字符流:Reader,Writer**

## 1.OutputStream:字节输出流

### 1.1定义

public abstract class OutputStream

extends Object

implements Closeable, Flushable

最重要的三个输出的方法:

|  |  |
| --- | --- |
| 输出单个字节 | public abstract void write(int b)  throws IOException |
| 输出全部字节数组 | public void write(byte[] b)  throws IOException |
| 输出部分字节数组 | public void write(byte[] b,int off,int len)  throws IOException |

### 1.2FileOutputStream

说明:此类为OutputStream的直接子类

|  |  |
| --- | --- |
| 创建或者覆盖文件(字符路径) | public FileOutputStream(String name)  throws FileNotFoundException |
| 文件追加 | publicFileOutputStream(String name,boolean append)  throws FileNotFoundException |
| 创建或者覆盖文件 | public FileOutputStream(File file)  throws FileNotFoundException |
| 文件追加 | public FileOutputStream(File file, boolean append)  throws FileNotFoundException |

范例:写入文件:

public class test {  
 public static void main(String args[]) throws Exception { //注意这里  
 //定义输出文件的路径  
 File file = new File("F:" + File.*separator* + "demo" + File.*separator* + "01.txt");  
 if(!file.getParentFile().exists()){ //如果目录不存在,则创建  
 file.getParentFile().mkdirs();  
 }  
 OutputStream out = new FileOutputStream(file);  
 String str = "hello,world 你好世界";  
 byte data[] = str.getBytes(); //将字符串变为字节数组  
 //out.write(data); //输出全部字节数组内容  
 out.write(data,3,5); //输出部分字节数组  
 out.close();} }

## 2.InputStream:字节输入流

### 2.1定义

public abstract class InputStream

extends Object

implements Closeable //不考虑接口的存在,考虑本身类就可以了

|  |  |
| --- | --- |
| 读取单个字节  返回值:返回读取的字节内容,如果到文件末尾了返回-1 | public abstract int read()  throws IOException |
| 将读取的数组保存在字节数组中  返回值:返回读取的数据长度,如果到文件末尾了返回-1 | public int read(byte[] b)  throws IOException |
| 将读取的数据保存在部分字节数组中  返回值: 返回读取的数据长度,如果到文件末尾了返回-1 | public int read(byte[] b, int off,int len)  throws IOException |

### 2.2FileInputStream

说明:此类是InputStream的直接子类

|  |  |
| --- | --- |
| 构造方法接受File类对象 | public FileInputStream(File file)  throws FileNotFoundException |
| 构造方法接受字符串文件路径 | public FileInputStream(String name)  throws FileNotFoundException |

范例:从文件里面读取数据保存到字节数组里面

public class test {  
 public static void main(String args[]) throws Exception { //注意这里  
 //定义输出文件的路径  
 File file = new File("F:" + File.*separator* + "demo" + File.*separator* + "01.txt");  
 if(file.exists()) { //注意  
 InputStream in = new FileInputStream(file);  
 byte data[] = new byte[1024];  
 int len = in.read(data); //注意返回的是读取内容的字节长度  
 in.close();  
 System.*out*.println(new String(data,0,len));  
 }  
 }  
}

范例:从文件读取数据,每次读取一个字节

public class test {  
 public static void main(String args[]) throws Exception { //注意这里  
 //定义输出文件的路径  
 File file = new File("F:" + File.*separator* + "demo" + File.*separator* + "01.txt");  
 if(file.exists()) { //注意  
 InputStream in = new FileInputStream(file);  
 byte data[] = new byte[1024];  
 int temp = 0;  
 int foot = 0;  
 while((temp=in.read())!=-1){  
 data[foot++] = (byte)temp; //保存读取的字节到数组之中  
 }  
 in.close();  
 System.*out*.println(new String(data,0,foot));  
 }  
 }  
}

## 3.Writer:字符输出流

### 3.1定义

public abstract class Writer

extends Object

implements Appendable, Closeable, Flushable

//这里比OutputStream多了个Appendable

Appendable的定义:

public interface Appendable{

Appendable append(char c) throws IOException;

Appendable append(CharSequence csq) throws IOException;

Appendable append(CharSequence csq,int start,int end)

throws IOException

}

Writer的输出方法:

|  |  |
| --- | --- |
| 输出全部字符数组 | public void write(char[] cbuf)  throws IOException |
| 输出部分字符数组 | public abstract void write(char[] cbuf,int off,int len)  throws IOException |
| 输出字符串 | public void write(String str)  throws IOException |
| 输出部分字符串 | public void write(String str,int off,int len)  throws IOException |
| 输出单个字符 | public void write(int c)  throws IOException |

Writer能直接输出字符串

### 3.2FileWriter

java.lang.Object

java.io.Writer

java.io.OutputStreamWriter

java.io.FileWriter

public class FileWriter

extends OutputStreamWriter

构造方法接受的参数和FileoutputStream一样(File类型和String类型)

范例:输出字符串到文件中

public class test {  
 public static void main(String args[]) throws Exception { //注意这里  
 //定义输出文件的路径  
 File file = new File("F:" + File.*separator* + "demo" + File.*separator* + "01.txt");  
 if(file.getParentFile().exists()) {  
 file.getParentFile().mkdirs();  
 }  
 Writer out = new FileWriter(file);  
 out.write("hello,world"); //输出字符串数据  
 out.close(); //关闭输出流  
 }  
}

## 4.Reader:字符输入流

### 4.1定义

public abstract class Reader

extends Object

implements Readable, Closeable

//InputStream多了一个Readable接口,不过这个接口不用关注

**Reader里的一些方法:**

public int read() throws IOException

public int read(char[] cbuf) throws IOException

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

### 4.2FileReader

java.lang.Object

java.io.Reader

java.io.InputStreamReader

java.io.FileReader

*FileReader的构造函数和FileWriter一样,接受File类型和String类型*

范例:将文件内容读入到字符数组中

public class test {  
 public static void main(String args[]) throws Exception { //注意这里  
 //定义输出文件的路径  
 File file = new File("F:" + File.*separator* + "demo" + File.*separator* + "01.txt");  
 if(file.exists()){  
 Reader in = new FileReader(file);  
 char data[] = new char[1024];  
 int len = in.read(data);  
 in.close();  
 System.*out*.println(new String(data,0,len));  
 }  
 }  
}

**此时和InputStream一样,只是一个是字节数组一个是字符数组**

## 5.字节流和字符流的区别?

通过字节流是将数据和文件(内存)直接写入和读出,而字符流会利用到缓冲区

字节数据:图片,音乐,电影,文字,字节流为主

字符流数据:中文

## 6.字节流和字符流的转化(了解)

|  |  |  |
| --- | --- | --- |
| 名称 | InputStreamReader | OutputStreamWriter |
| 定义 | Public class InputStreamReader  extends Reader | public class OutputStreamWriter  extends Writer |
| 构造方法 | public InputStreamReader(InputStream in) | public OutputStreamWriter(OutputStream out) |

转换发生的情况:对于中文的处理

范例:将字节输出流转化为字符输出流

public class test {  
 public static void main(String args[]) throws Exception { //注意这里  
 //定义输出文件的路径  
 File file = new File("F:" + File.*separator* + "demo" + File.*separator* + "01.txt");  
 if(!file.getParentFile().exists()){  
 file.getParentFile().mkdirs();  
 }  
 OutputStream outputStream = new FileOutputStream(file);//字节流  
 Writer out = new OutputStreamWriter(outputStream); //将字节流变为字符流  
 out.write("Ni 好,世界"); //用的Writer的方法  
 out.close();  
 outputStream.close();  
 }  
}

## 7.实战-文件拷贝

public class test{  
 public static void main(String args[]) throws Exception {  
 long start = System.*currentTimeMillis*();  
 if(args.length != 2){  
 System.*out*.println("命令执行错误");  
 System.*exit*(1);//程序退出执行  
 }  
 File inFile = new File(args[0]);  
 if(!inFile.exists()){ //源文件不存在  
 System.*out*.println("源文件不存在");  
 System.*exit*(1);//程序退出执行  
 }  
 File outFile = new File(args[1]);  
 if(!outFile.getParentFile().exists()){ //输出文件路径不存在  
 outFile.getParentFile().mkdirs(); //创建目录  
 }  
 InputStream input = new FileInputStream(inFile);  
 OutputStream output = new FileOutputStream(outFile);  
 int len = 0;//保存每次读取字节个数  
 byte data[] = new byte[1024];  
 while((len = input.read(data))!=-1){ //读取数据到字节数组中  
 output.write(data,0,len); //输出单字节数组  
 }  
 input.close();  
 output.close();  
 long end = System.*currentTimeMillis*();  
 System.*out*.println("拷贝时间；" + (end - start));  
 }  
}

## 8.内存流

情况:需要进行I/O操作,有不想产生文件,就可以使用内存来实现输入和输出的操作

字节内存流:ByteArrayInputStream(输入) ByteArrayOutputStream(输出)

字符内存流:CharArrayReader(输入) CharArrayWriter(输出)

### 8.1字节内存流

|  |  |  |
| --- | --- | --- |
| 名称 | ByteArrayInputStream | ByteArrayOutputStream |
| 继承结构 | java.lang.Object  java.io.InputStream  java.io.ByteArrayInputStream | java.lang.Object  java.io.OutputStream  java.io.ByteArrayOutputStream |
| 构造方法 | public ByteArrayInputStream(byte[] buf)  ByteArrayInputStream(byte[] buf, int offset, int length)  将字节数组保存到内存中,最后通过read的参数提供的缓冲区,每次读取一定的数据 | public ByteArrayOutputStream()  public ByteArrayOutputStream(int size)  准备一个缓冲区，write的数据保存到缓冲区 |
|  | public int read()  每次返回读取的一个字节,-1表示到末尾了 | public void write(int b)  将单个字节保存到缓冲区中 |
|  | public int read(byte[] b,int off,int len)  指定一个字节数组来保存读入的内容 | public void write(byte[] b,int off,int len)将指定的字节数据保存到会缓冲区 |

范例:

public class test{  
 public static void main(String args[]) throws Exception {  
 String str = "hahahahahahha132424324,24234324234234321";  
 InputStream input = new ByteArrayInputStream(str.getBytes()); //将数据保存到内存中  
 byte data[] = new byte[2]; //提供一个缓冲区  
 while(input.read(data,0,data.length)!=-1){  
 System.*out*.println(new String(data)); //每次从内存中读取数据到缓冲区data中  
 }  
  
 OutputStream output = new ByteArrayOutputStream(); //定义一个输出缓冲区  
 int a = 97;  
 int b = 98;  
 int c = 99;  
 output.write(a); //将要写到内存的数据保存到输出缓冲区  
 output.write(b);//将要写到内存的数据保存到输出缓冲区  
 output.write(c);//将要写到内存的数据保存到输出缓冲区  
 System.*out*.println(output); //调用toString()  
 }  
}

## 9.打印流

范例:自定义打印流

public class test{  
 public static void main(String args[]) throws Exception {  
 PrintUtil p = new PrintUtil(new FileOutputStream("F:\\test.txt"));  
 p.println("hello ");  
 p.println("world");  
 p.close();  
 }  
}  
class PrintUtil{  
 private OutputStream out;  
 public void print(String x) {  
 try {  
 out.write(x.getBytes());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 public void print(int x) {  
 this.print(String.*valueOf*(x));  
 }  
 public void print(double x){  
 this.print(String.*valueOf*(x));  
 }  
 public void println(String x){  
 this.print(x+"\n");  
 }  
 public void println(int x){  
 this.println(String.*valueOf*(x));  
 }  
 public void println(double x){  
 this.println(String.*valueOf*(x));  
 }  
 public PrintUtil(OutputStream out){  
 this.out = out;  
 }  
 public void close(){  
 try {  
 this.out.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

打印流:PrintStream(打印字节流),PrintWriter(打印字符流),这两个类除了类名称不一样,其他都一样

### 9.1PrintStream:打印字节流

定义:

java.lang.Object

java.io.OutputStream

java.io.FilterOutputStream

java.io.PrintStream

在PrintStream里有一系列的print,println方法,这些方法支持各种类型的数据输出,比OutputStream里的write只支持三种类型的数据好多了,以后用print,println，而不用write,装饰设计模式

构造函数

PrintStream(File file)

PrintStream(File file, String csn)

PrintStream(OutputStream out)

PrintStream(OutputStream out, boolean autoFlush)

PrintStream(OutputStream out, boolean autoFlush, String encoding)

PrintStream(String fileName)

PrintStream(String fileName, String csn)

PrintStream里的format格式化输出标记:

整数(%d),字符串(%s),小数(%m.nf),字符(%c)

**范例:PrintStream格式化输出**

public class test{  
 public static void main(String args[]) throws Exception {  
 PrintStream pu = new PrintStream(new File("F:\\demo\\01.txt"));  
 String name = "林威";  
 int age = 19;  
 double score = 34.6;  
 pu.print("hello,wrold,你好");  
 pu.format("姓名:%s,年龄:%d,成绩:%5.2f",name,age,score);  
 pu.close();  
 }  
}

不过format基本不用,因为字符串自己也有format方法

### 9.2String的format格式化

定义:

Public static String format(String format,Object…args)

范例:

public class test{  
 public static void main(String args[]) throws Exception {  
 String name = "林威";  
 int age = 19;  
 double score = 34.6;  
 String str = String.*format*("姓名:%s,年龄:%d,成绩:%5.2f",name,age,score);  
 System.*out*.print(str);  
 }  
}

### 9.3System里的三个IO常量(java.lang.System)

错误输出: public static final PrintStream err (基本不用)

输出到标准输出设备(显示器): public static final PrintStream out

从标准输入设备输入(键盘): public static final InputStream in

**范例:利用OutputStream实现屏幕输出**

OutputStream out = System.out;

Out.write(“hello,world”.getBytes());

**范例:利用函数式接口引用print函数:**

Consumer<String> con = System.*out*::print;//引用某个对象的方法  
con.accept("hello,world");

**范例:利用System.in进行输入**

public class test{  
 public static void main(String args[]) throws Exception {  
 InputStream input = System.*in*;  
 byte data[] = new byte[3];  
 System.*out*.print("请输入数据:");  
 input.read(data);  
 System.*out*.print("输出的数据为:"+ new String(data));  
 }  
}

**注意:超过data数组长度的数据将不会被保存!!!!!!,在输入的过程中你不可能知道输入的数据长度是多少,所以这个设置输入长度的长度不靠谱!!!**

### 9.4缓冲区流

字符缓冲区流:BufferedReader BufferedWriter

字节缓冲区流:BufferedInputStream BufferedOutputStream

这四个类中最重要的就是BufferedReader,其他的可以忽略

#### 9.4.1BufferedReader:主要用来代替System.in用来输入的

***比System.in的优点:可以直接接受字符串,不用设定数组长度来接受数据***

定义:

java.lang.Object

java.io.Reader

java.io.BufferedReader

一个重要的方法:

public String readLine() throws IOException 读取一行数据,分隔符:换行符并且换行符不被保存,如果返回null则表示读取到末尾了

构造方法:

public BufferedReader(Reader in)

public BufferedReader(Reader in, int sz)

范例:利用System.in进行输入

public class test{  
 public static void main(String args[]) throws Exception {  
 BufferedReader in = new BufferedReader(new InputStreamReader(System.*in*));  
 System.*out*.print("请输入数据:");  
 String str = in.readLine();  
 System.*out*.print("输出数据:");  
 System.*out*.print(str);  
 }  
}

范例:对文件一行一行读取

public class test{  
 public static void main(String args[]) throws Exception {  
 File file = new File("F:\\demo\\01.txt");  
 if(file.exists()){  
 BufferedReader buf = new BufferedReader(new FileReader(file));  
 String str = "";  
 while((str = buf.readLine())!=null){  
 System.*out*.println(str);  
 }  
 }  
 }  
}

## 10.Scanner:java.util.Scanner

BufferedReader有两个问题:

1. 只能够返回字符串:public String readLine()
2. 分隔符是固定的为换行符

以后考虑输入,优先考虑Scanner

java.lang.Object

java.util.Scanner

|  |  |
| --- | --- |
| 构造函数 | Scanner(InputStream source) |
| 判断是否有某个数据 | Public boolean hasXxx() |
| 取出数据 | Public 数据类型nextXxx() |
|  | Public boolean hasNext(String pattern) |
|  | Public String next(String pattern) |
| 设置分隔符 | public Scanner useDelimiter(Pattern pattern) |

hasNext和next要同一

范例:对double类型的输入的数字进行验证

public class test{  
 public static void main(String args[]) throws Exception {  
 Scanner scan = new Scanner(System.*in*); //准备接受键盘输入数据  
 System.*out*.print("请输入:");  
 if(scan.hasNextDouble()){ //判断是否为Double  
 System.*out*.print("输入内容:" + scan.nextDouble()); //直接由Strign转为Double  
 }else{  
 System.*out*.print("输入错误");  
 }  
 scan.close();  
 }  
}

范例;正则验证

public class test{  
 public static void main(String args[]) throws Exception {  
 Scanner scan = new Scanner(System.*in*); //准备接受键盘输入数据  
 System.*out*.print("请输入生日:");  
 if(scan.hasNext("\\d{4}-\\d{2}-\\d{2}")){ //判断是否为Double  
 String bir = scan.next("\\d{4}-\\d{2}-\\d{2}");  
 System.*out*.print("输入内容:" + bir); //直接由Strign转为Double  
 }else{  
 System.*out*.print("生日格式错误");  
 }  
 scan.useDelimiter()  
 scan.close();  
 }  
}

范例:读取文件

public class test{  
 public static void main(String args[]) throws Exception {  
 Scanner scan = new Scanner(new FileInputStream("F:\\demo\\01.txt"));  
 scan.useDelimiter("\n\r");  
 while (scan.hasNext()){  
 System.*out*.print(scan.next());  
 }  
 scan.close();  
 }  
}

## 11.总结

括号内为增强版

字节流:InputStream(Scanner),OutputStream(PrintStream)

字符流:Reader(BufferedReader),Writer(PrintWriter)