彻底明白Java的IO系统 作 者: pdfjn5198

一． Input和Output

1. stream代表的是任何有能力产出数据的数据源，或是任何有能力接收数据的接收源。在Java的IO中，所有的stream（包括Input和Out stream）都包括两种类型：

1.1 以字节为导向的stream

以字节为导向的stream，表示以字节为单位从stream中读取或往stream中写入信息。以字节为导向的stream包括下面几种类型：

1) input　stream：

1) ByteArrayInputStream：把内存中的一个缓冲区作为InputStream使用

2) StringBufferInputStream：把一个String对象作为InputStream

3) FileInputStream：把一个文件作为InputStream，实现对文件的读取操作

4) PipedInputStream：实现了pipe的概念，主要在线程中使用

5) SequenceInputStream：把多个InputStream合并为一个InputStream

2) Out　stream

1) ByteArrayOutputStream：把信息存入内存中的一个缓冲区中

2) FileOutputStream：把信息存入文件中

3) PipedOutputStream：实现了pipe的概念，主要在线程中使用

4) SequenceOutputStream：把多个OutStream合并为一个OutStream

1.2 以Unicode字符为导向的stream

以Unicode字符为导向的stream，表示以Unicode字符为单位从stream中读取或往stream中写入信息。以Unicode字符为导向的stream包括下面几种类型：

1) Input　Stream

1) CharArrayReader：与ByteArrayInputStream对应

2) StringReader：与StringBufferInputStream对应

3) FileReader：与FileInputStream对应

4) PipedReader：与PipedInputStream对应

2) Out　Stream

1) CharArrayWrite：与ByteArrayOutputStream对应

2) StringWrite：无与之对应的以字节为导向的stream

3) FileWrite：与FileOutputStream对应

4) PipedWrite：与PipedOutputStream对应

以字符为导向的stream基本上对有与之相对应的以字节为导向的stream。两个对应类实现的功能相同，只是在操作时的导向不同。如CharArrayReader：和ByteArrayInputStream的作用都是把内存中的一个缓冲区作为InputStream使用，所不同的是前者每次从内存中读取一个字节的信息，而后者每次从内存中读取一个字符。

1.3 两种不现导向的stream之间的转换

InputStreamReader和OutputStreamReader：把一个以字节为导向的stream转换成一个以字符为导向的stream。

2. stream添加属性

2.1 “为stream添加属性”的作用

运用上面介绍的Java中操作IO的API，我们就可完成我们想完成的任何操作了。但通过FilterInputStream和FilterOutStream的子类，我们可以为stream添加属性。下面以一个例子来说明这种功能的作用。

如果我们要往一个文件中写入数据，我们可以这样操作：

FileOutStream fs = new FileOutStream(“test.txt”);

然后就可以通过产生的fs对象调用write()函数来往test.txt文件中写入数据了。但是，如果我们想实现“先把要写入文件的数据先缓存到内存中，再把缓存中的数据写入文件中”的功能时，上面的API就没有一个能满足我们的需求了。但是通过FilterInputStream和FilterOutStream的子类，为FileOutStream添加我们所需要的功能。

2.2 FilterInputStream的各种类型

2.2.1 用于封装以字节为导向的InputStream

1) DataInputStream：从stream中读取基本类型（int、char等）数据。

2) BufferedInputStream：使用缓冲区

3) LineNumberInputStream：会记录input stream内的行数，然后可以调用getLineNumber()和setLineNumber(int)

4) PushbackInputStream：很少用到，一般用于编译器开发

2.2.2 用于封装以字符为导向的InputStream

1) 没有与DataInputStream对应的类。除非在要使用readLine()时改用BufferedReader，否则使用DataInputStream

2) BufferedReader：与BufferedInputStream对应

3) LineNumberReader：与LineNumberInputStream对应

4) PushBackReader：与PushbackInputStream对应

2.3 FilterOutStream的各种类型

2.2.3 用于封装以字节为导向的OutputStream

1) DataIOutStream：往stream中输出基本类型（int、char等）数据。

2) BufferedOutStream：使用缓冲区

3) PrintStream：产生格式化输出

2.2.4 用于封装以字符为导向的OutputStream

1) BufferedWrite：与对应

2) PrintWrite：与对应

3. RandomAccessFile

1) 可通过RandomAccessFile对象完成对文件的读写操作

2) 在产生一个对象时，可指明要打开的文件的性质：r，只读；w，只写；rw可读写

3) 可以直接跳到文件中指定的位置

4. I/O应用的一个例子

import java.io.\*;

public class TestIO{

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

//1.以行为单位从一个文件读取数据

BufferedReader in =

new BufferedReader(

new FileReader("F:\\nepalon\\TestIO.java"));

String s, s2 = new String();

while((s = in.readLine()) != null)

s2 += s + "\n";

in.close();

//1b. 接收键盘的输入

BufferedReader stdin =

new BufferedReader(

new InputStreamReader(System.in));

System.out.println("Enter a line:");

System.out.println(stdin.readLine());

//2. 从一个String对象中读取数据

StringReader in2 = new StringReader(s2);

int c;

while((c = in2.read()) != -1)

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

in2.close();

//3. 从内存取出格式化输入

try{

DataInputStream in3 =

new DataInputStream(

new ByteArrayInputStream(s2.getBytes()));

while(true)

System.out.println((char)in3.readByte());

}

catch(EOFException e){

System.out.println("End of stream");

}

//4. 输出到文件

try{

BufferedReader in4 =

new BufferedReader(

new StringReader(s2));

PrintWriter out1 =

new PrintWriter(

new BufferedWriter(

new FileWriter("F:\\nepalon\\ TestIO.out")));

int lineCount = 1;

while((s = in4.readLine()) != null)

out1.println(lineCount++ + "：" + s);

out1.close();

in4.close();

}

catch(EOFException ex){

System.out.println("End of stream");

}

//5. 数据的存储和恢复

try{

DataOutputStream out2 =

new DataOutputStream(

new BufferedOutputStream(

new FileOutputStream("F:\\nepalon\\ Data.txt")));

out2.writeDouble(3.1415926);

out2.writeChars("\nThas was pi:writeChars\n");

out2.writeBytes("Thas was pi:writeByte\n");

out2.close();

DataInputStream in5 =

new DataInputStream(

new BufferedInputStream(

new FileInputStream("F:\\nepalon\\ Data.txt")));

BufferedReader in5br =

new BufferedReader(

new InputStreamReader(in5));

System.out.println(in5.readDouble());

System.out.println(in5br.readLine());

System.out.println(in5br.readLine());

}

catch(EOFException e){

System.out.println("End of stream");

}

//6. 通过RandomAccessFile操作文件

RandomAccessFile rf =

new RandomAccessFile("F:\\nepalon\\ rtest.dat", "rw");

for(int i=0; i<10; i++)

rf.writeDouble(i\*1.414);

rf.close();

rf = new RandomAccessFile("F:\\nepalon\\ rtest.dat", "r");

for(int i=0; i<10; i++)

System.out.println("Value " + i + "：" + rf.readDouble());

rf.close();

rf = new RandomAccessFile("F:\\nepalon\\ rtest.dat", "rw");

rf.seek(5\*8);

rf.writeDouble(47.0001);

rf.close();

rf = new RandomAccessFile("F:\\nepalon\\ rtest.dat", "r");

for(int i=0; i<10; i++)

System.out.println("Value " + i + "：" + rf.readDouble());

rf.close();

}

}

关于代码的解释（以区为单位）：

1区中，当读取文件时，先把文件内容读到缓存中，当调用in.readLine()时，再从缓存中以字符的方式读取数据（以下简称“缓存字节读取方式”）。

1b区中，由于想以缓存字节读取方式从标准IO（键盘）中读取数据，所以要先把标准IO（System.in）转换成字符导向的stream，再进行BufferedReader封装。

2区中，要以字符的形式从一个String对象中读取数据，所以要产生一个StringReader类型的stream。

4区中，对String对象s2读取数据时，先把对象中的数据存入缓存中，再从缓冲中进行读取；对TestIO.out文件进行操作时，先把格式化后的信息输出到缓存中，再把缓存中的信息输出到文件中。

5区中，对Data.txt文件进行输出时，是先把基本类型的数据输出屋缓存中，再把缓存中的数据输出到文件中；对文件进行读取操作时，先把文件中的数据读取到缓存中，再从缓存中以基本类型的形式进行读取。注意in5.readDouble()这一行。因为写入第一个writeDouble()，所以为了正确显示。也要以基本类型的形式进行读取。

6区是通过RandomAccessFile类对文件进行操作。

[Edit on 2004-09-20 11:08:23 By aeonsun]

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

此日记TrackBack地址: http://aeonsun.polyhero.net/trackback.do?log\_id=16

In [转贴及收藏] By [aeonsun] at 04-9-20 10:48 | Comments [10] | TB [0] | 92 Views

≡≡≡ 网友评论 ≡≡≡

aeonsun 网友说:

重要提示：

LineNumberInputStream，StringBufferInputStream已经废除！大家不要再用！

LineNumberInputStream，This class does not properly convert characters into bytes！

StringBufferInputStream，Deprecated. This class incorrectly assumes that bytes adequately represent characters！

at 04-09-20 10:49

aeonsun 网友说:

DataInputStream流中已经没有readLine()整个方法！

替换为： BufferedReader d=new BufferedReader(new InputStreamReader(in));

--------把字节流转换为字符流接入缓存读取字符流中，再进行处理！

这个方法很大的优势！

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

-----------------Why use character streams?------------------

The primary advantage of character streams is that they make it easy to write programs that are not dependent upon a specific character encoding, and are therefore easy to internationalize.

Java stores strings in Unicode, an international standard character encoding that is capable of representing most of the world's written languages. Typical user-readable text files, however, use encodings that are not necessarily related to Unicode, or even to ASCII, and there are many such encodings. Character streams hide the complexity of dealing with these encodings by providing two classes that serve as bridges between byte streams and character streams. The InputStreamReader class implements a character-input stream that reads bytes from a byte-input stream and converts them to characters according to a specified encoding. Similarly, the OutputStreamWriter class implements a character-output stream that converts characters into bytes according a specified encoding and writes them to a byte-output stream.

A second advantage of character streams is that they are potentially much more efficient than byte streams. The implementations of many of Java's original byte streams are oriented around byte-at-a-time read and write operations. The character-stream classes, in contrast, are oriented around buffer-at-a-time read and write operations. This difference, in combination with a more efficient locking scheme, allows the character stream classes to make up for the added overhead of encoding conversion in many cases.

at 04-09-20 10:49

aeonsun 网友说:

----------标准设备System.in读取数据------------------

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

读取字节：BufferedInputStream

读取字符：BufferedReader + InputStreamReader

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

import java.io.\*;

public class systemin

{

public static void main(String args[])

{ try{ //流转换！

BufferedReader is=new BufferedReader(new InputStreamReader(System.in))

String inputline=null;

while((inputline=is.readLine())!=null)

System.out.println(inputline);

is.close();

}

catch(IOException e)

{ System,out.println("IOXE: "+e);

}

}

}

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

-----------------标准输出System.out是一个打印流PrintStream---------------------

import java.io.\*;

public class PrintStandardOutput {

public static void main(String[] args) {

String myAnswer = "No, and that's final,";

System.out.println("Hello World of Java");

System.out.println("The answer is " + myAnswer + " at this time.");

PrintWriter pw = new PrintWriter(System.out);

pw.println("The answer is " + myAnswer + " at this time.");

int i = 42;

pw.println(i + '=' + " the answer.");

pw.println("Note: " + i + '=' + " the answer.");

pw.println(i + "=" + " the answer.");

pw.println(i + ('=' + " the answer."));

pw.close();

}

}

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

-----------------------要读取(输出到—）一个文本文件-----------------------------

BufferedReader is=new BufferedReader(new FileReader("xxxx.text"));读取

BufferedOutputStream byteout=new BufferedOutputStream(new FileOutputStream("XX.dat"));

// 写出到文本！

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

at 04-09-20 10:49

aeonsun 网友说:

import java.io.\*;

public class OpenFileByName {

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

BufferedReader is = new BufferedReader(new FileReader("myFile.txt"));

BufferedOutputStream bytesOut = new BufferedOutputStream(

new FileOutputStream("bytes.dat"));

// Code here to read from is, write to bytesOut

bytesOut.close();

}

}

at 04-09-20 10:50

aeonsun 网友说:

import java.io.\*;

public class FileIO {

private FileIO() { }

public static void copyFile(String inName, String outName)

throws FileNotFoundException, IOException {

BufferedInputStream is =

new BufferedInputStream(new FileInputStream(inName));

BufferedOutputStream os =

new BufferedOutputStream(new FileOutputStream(outName));

copyFile(is, os, true);

}

/\*\* Copy a file from an opened InputStream to opened OutputStream \*/

public static void copyFile(InputStream is, OutputStream os, boolean close)

throws IOException {

int b; while ((b = is.read()) != -1) {

os.write(b);

}

is.close();

if (close)

os.close();

}

public static void copyFile(Reader is, Writer os, boolean close)

throws IOException {

int b; while ((b = is.read()) != -1) {

os.write(b);

}

is.close();

if (close)

os.close();

}

public static void copyFile(String inName, PrintWriter pw, boolean close)

throws FileNotFoundException, IOException {

BufferedReader is = new BufferedReader(new FileReader(inName));

copyFile(is, pw, close);

}

public static String readLine(String inName)

throws FileNotFoundException, IOException {

BufferedReader is = new BufferedReader(new FileReader(inName));

String line = null;

line = is.readLine();

is.close();

return line;

}

protected static final int BLKSIZ = 8192;

public void copyFileBuffered(String inName, String outName) throws

FileNotFoundException, IOException {

InputStream is = new FileInputStream(inName);

OutputStream os = new FileOutputStream(outName);

int count = 0;

byte b[] = new byte[BLKSIZ];

while ((count = is.read(b)) != -1) {

os.write(b, 0, count);

}

is.close();

os.close();

}

public static String readerToString(Reader is) throws IOException {

StringBuffer sb = new StringBuffer();

char[] b = new char[BLKSIZ];

int n;

while ((n = is.read(b)) > 0) {

sb.append(b, 0, n);

}

return sb.toString();

}

public static String inputStreamToString(InputStream is)

throws IOException {

return readerToString(new InputStreamReader(is));

}

public static void stringToFile(String text, String fileName)

throws IOException {

BufferedWriter os = new BufferedWriter(new FileWriter(fileName));

os.write(text);

os.flush();

os.close();

}

public static BufferedReader openFile(String fileName)

throws IOException {

return new BufferedReader(new FileReader(fileName));

}

}

at 04-09-20 10:50

aeonsun 网友说:

import java,io.\*;

class fileprocess

{

public static void main(String args[])

{

int b;

byte buffer[]=new byte[1000];

try

{ b=System.out.read(buffer); //存取数据，以备写出!

FileOutputStream out=new FileOotputStream("line.text");

out.write(buffer,0,b); //写出！注意字节缓冲语文件流的关系

]

catch(IOException e)

{ System.out.println(" error");

}

]

}

at 04-09-20 10:50

aeonsun 网友说:

import java,io.\*;

public class read

{

public static void main(String args[]0

{

int b;

byte buffer=new byte[2000];

try

{ FileInputStream readfile=new FileInputStream("XXXX.java");

b=readfile(buffer,0,25000);

try{String str=nre String(buffer,0,b,"Default");//构造String对象！

System.out.println(str);}

catch(UnsuportedEncodeingException e)

{ e.printStackTrace();}

catch( IOException e)

{ System.out.println(" error");}

}

}

at 04-09-20 10:51

aeonsun 网友说:

---------------利用 BufferedReader--FileReader读取文本文件！-----------

---------------在IO中始终要注意是字节流还是字符流----------------------

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

import java.io.\*;

import java.awt.\*;

import javax.swing.\*;

import java.awt.event.\*;

class filewindow extends JFrame implements ActionListener

{

JTextArea text;

BufferedReader in;

JButton button;

FileReader file;

filewindow()

{

super("文件字符流");

Container con=getContentPane();

text=new JTextArea(50,50);

text.setBackground(Color.blue);

try{

File f=new File("E:\\a.txt");

file=new FileReader(f);

in=new BufferedReader(file);

/\*\*BufferedReader（Reader in)构造函数，

\*文件自字符读取流FileReader接入BufferedReader

\*流中，以便用BufferedReader的对象方法readLine()高效成行读取！

\*/

}

catch(FileNotFoundException e){}

catch(IOException e){}

button=new JButton("读取");

button.addActionListener(this);

con.setLayout(new BorderLayout());

setSize(300,200);

setVisible(true);

con.add(text,"Center");

con.add(button,"South");

addWindowListener(new WindowAdapter()

{public void windowClosing(WindowEvent e)

{setVisible(false);System.exit(0);}});

}

public void actionPerformed(ActionEvent e)

{

String s;

if(e.getSource()==button)

try{

while((s=in.readLine())!=null)

text.append(s+'\n');

//在这里大家还可以用BufferString来暂时保存读取的字符数据！

}

catch(IOException e1){}

}

//---------main()----------

public static void main(String args[])

{

filewindow win=new filewindow();

win.pack();

}

}

at 04-09-20 10:51

aeonsun 网友说:

-------------------RandomAccessFile随机读取文件---------------

import java.io.\*;

public class RandomRead

{

final static String FILENAME="E:\\a.txt";

protected String fileName;

protected RandomAccessFile seeker;

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

RandomRead r = new RandomRead(FILENAME);

System.out.println("Offset is " + r.readOffset());

System.out.println("Message is \"" + r.readMessage() + "\".");

}

/\*\* Constructor: save filename, construct RandomAccessFile \*/

public RandomRead(String fname) throws IOException {

fileName = fname;

seeker = new RandomAccessFile(fname, "rw");

}

/\*\* Read the Offset field, defined to be at location 0 in the file. \*/

public int readOffset() throws IOException {

seeker.seek(0);

seeker.writeChars(FILENAME); // move to very beginning

return seeker.readInt(); // and read the offset

}

/\*\* Read the message at the given offset \*/

public String readMessage() throws IOException {

seeker.seek(120); // move to where

return seeker.readLine(); // and read the String

}

}

at 04-09-20 10:51

aeonsun 网友说:

一些评论：axman

写得很辛苦,我本来不想再说什么了,但本着对技术负责的精神还是说出来.

对于I/O的理解属于3级水平(如果java IO有十级的话)

错误太多.

对于I/O层次不熟悉

java IO主要包括

java.io包和java.nio包.

java.io主要从四个接口延伸:

字节:

InputStream/OutputStream,其下为封装,过滤,特定对象处理的具体实现类.

字符:

Reader/Writer(原文中连Writer接口全都写成Write,足以说明根本不了解这了接口,如果你经常使用Writer接口怎么会连Writer和Write都分不清,不是一处失误,而是全部都是Write)

以上四个接口中,底层全部是操作字节的阻塞方式流.

java.io主要以块操作块(Buffer)为主,通过可以设定的阻塞和非阴塞模式,极大地提过了数据输出输

入的性能,而且将Channel通过选选择器模式的控制,可以实现在同一输出输入通道上多用户并发进行数据传输入.比如一个Socket端口可以同时被无限多(理论上不受限制)个客户端并发访问.就是经典的I/O多路复用技术.