# 1.IO流

流动的是数据---用于传输数据的一套API---IO->Input和Output---输入流和输出流---当数据是从外部流向程序的时候，输入流；数据是从程序流向外部的时候，输出流。读取文件---将数据从文件读到程序中---输入流；向文件中写入数据---数据从程序流向了文件---输出流。

根据数据的传输方向：输入流、输出流

根据数据的传输形式：字节流、字符流

|  |  |  |
| --- | --- | --- |
|  | 输入流 | 输出流 |
| 字节流 | InputStream | OutputStream |
| 字符流 | Reader | Writer |

这四个基本流都是抽象类。

数据来源(目的地)：硬盘，网络，输入设备，内存

## 1.1.FileWriter

向txt文件写入一个字符串---输出流，字符流，和文件有关的流---FileWriter

FileWriter(String path)---创建一个新文件。如果文件不存在，会创建一个新文件。

FileWriter继承了OutputStreamWriter。

### 1.1.1.流操作的过程

创建流→流出数据→冲刷缓冲区→关闭流。

最后将流对象赋值为null标记为一个垃圾对象。

### 1.1.2.构造函数

①FileWriter(File file);

无论文件是否存在，都会创建一个新的文件覆盖原来的文件。

②FileWriter(File file,boolean append);

是否追加内容。如果存在目标文件，不会删除原文件。

③FileWriter(String fileName);

根据给定的文件名fileName构造一个对象。

④FileWriter(String fileName,boolean append);

判断是否追加数据。

### 1.1.3.重要方法

①write(String s);

写入数据。

数据不是直接写到文件中个，而是先写到缓冲区，缓冲区满了之后，才会将数据挪到文件中。如果缓冲区没有满而程序已经结束了，那么数据就死在来缓冲区。

②flush();

冲刷缓冲区。

③close();

关流。为了释放文件。流在关闭之前会自动的冲刷一次缓冲区，以防有数据死在缓冲区中。流关闭了，流对象依然存在。

## 1.2.流中的异常处理

1.将流对象放在try外定义并且赋值为null，放在try中初始化

2.在关流之前需要判断流对象是否初始化成功

3.需要将流强制置为null以防关流失败产生数据丢失

4.需要在写完数据之后手动添加flush操作以防关流失败产生数据丢失

## 1.3.FileReader

读取一个txt文件---输入流 字符流 和文件有关的流---FileReader

FileReader继承了InputStreamReader。是缓冲流的一个子类。

### 1.3.1.流操作的过程

创建流→读取数据→关闭流

在输入流中没有缓冲区，可以自己指定缓冲区。

### 1.3.2.构造函数

①FileReader(File file);

读取给定的文件。

②FileReader(String fileName);

读取给定的文件名称的文件。

### 1.3.3.重要方法

①read();

每次只读取一个字符。返回值是一个int类型，表示的是这个字符对应的编码。

②read(char cs);

返回值表示读取到字符的个数，或者读入数组的字符个数。

练习：复制文件

import java.io.FileReader;

import java.io.FileWriter;

public class CopyFileDemo {

public static void main(String[] args) {

// 创建一个输入流指向要复制的文件

FileReader reader = null;

// 创建一个输出流指向要存储的文件

FileWriter writer = null;

try {

reader = new FileReader("E:\\a.txt");

writer = new FileWriter("F:\\a.txt");

// 定义一个变量来记录读取的字符个数

int i = -1;

// 定义一个数组作为缓冲区使用

char[] cs = new char[4];

// 读取文件

while ((i = reader.read(cs)) != -1) {

// 将读取的数据写出

writer.write(cs, 0, i);

}

// 需要手动冲刷缓冲区

writer.flush();

} catch (Exception e) {

e.printStackTrace();

} finally {

if (reader != null) {

try {

reader.close();

} catch (Exception e) {

e.printStackTrace();

} finally {

reader = null;

}

}

if (writer != null) {

try {

writer.close();

} catch (Exception e) {

e.printStackTrace();

} finally {

writer = null;

}

}

}

}

}

## 1.4.缓冲流

### 1.4.1.BufferedReader

BufferedReader---需要传入字符输入流，真正读取数据的是传入的这个字符输入流，缓冲流仅仅提供了一个缓冲区。

继承了Reader

在读取文件的过程中，考虑到FileReader没有缓冲区，读取效率比较低，就利用bufferedReader包装上一层缓冲区。关闭的时候只关闭外层流即可，因为关闭外层流的时候，会自动关闭里层的流。也可以手动关闭，关闭顺序要从里往外关闭。

#### 1.4.1.1.构造函数

①BufferedReader(Reader in);

使用默认大小输入缓冲区的缓冲字符输入流。

②BufferedReader(Reader in ,int sz)

创建一个使用指定大小输入缓冲区的缓冲字符输入流。

#### 1.4.1.2.重要方法

①readLine();

读取一行（以回车换行作为一行的结束），返回的是String类型。

### 1.4.2.BufferedWriter

用其他字符输出流来写出数据，提供来一个更大的缓冲区，同时提供来newLine();能够屏蔽不同系统之间换行的差异性。

#### 1.4.2.1.构造函数

①BufferedWriter(Writer out);

②BufferedWriter(Writer out , int sz);

#### 1.4.2.2.重要方法

①newLine();

换行。在不同的系统中，转换为对应的换行。

练习：统计Java代码的行数

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

public class GetRowCountDemo {

// 记录行数

static int count = 0;

public static void main(String[] args) {

// 创建一个File对象指向工作空间

File file = new File("E:\\workspace");

count(file);

System.out.println(count);

}

private static void count(File file) {

// 判断file是否为空

if (file == null) {

throw new NullPointerException("亲，文件不能为空哦~~~");

}

// 判断是否是一个目录

if (file.isDirectory()) {

// 获取子文件和子目录

File[] fs = file.listFiles();

for (File f : fs) {

count(f);

}

} else if (file.getName().endsWith(".java")) {

// 统计代码行数---按行读取

BufferedReader br = null;

try {

// 创建一个缓冲流指向要统计的文件

br = new BufferedReader(new FileReader(file));

// 按行读取

while (br.readLine() != null) {

count++;

}

} catch (IOException e) {

e.printStackTrace();

} finally {

if (br != null) {

try {

br.close();

} catch (Exception e) {

e.printStackTrace();

} finally {

br = null;

}

}

}

}

}

}

## 1.5.StringReader

读取字符串的流

## 1.6.字节流

以字节形式来读写文件。

### 1.6.1.FileOutputStream

FileOutputStream class 继承了OutputStream class。

创建一个字节流→写入数据→关流

#### 1.6.1.1.构造函数

①FileOutputStream(File file);

②FileOutputStream(File file);

③FileOutputStream(String name);

④FileOutputStream(String name,boolean append)

#### 1.6.1.2.重要方法

①write(int b);

②write(byte[] b);

③write(byte[] b,int off,int len);

### 1.6.2.FileInputStream

FileInputStream class继承了InputStream class。

创建一个字节流→创建缓冲区→读出数据→关流

#### 1.6.2.1.构造函数

①FileInputStream(File file);

②FileInputStream(String name);

#### 1.6.2.2.重要方法

①read();

②read(byte[] b);

③read(byte[] b ,int off,int len);

练习：复制文件

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class CopyFileExer {

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

long begin = System.currentTimeMillis();

// 创建一个字节流来读取指定的文件

FileInputStream in = new FileInputStream("E:\\aaa.zip");

// 创建有一个字节流来指向存储的文件

FileOutputStream out = new FileOutputStream("F:\\b.zip");

// 构建一个字节数组作为缓冲区

byte[] bs = new byte[1024 \* 1024 \* 15];

// 定义一个变量来记录读取到的字节个数

int len = -1;

// 读取数据

while ((len = in.read(bs)) != -1) {

// 将读取到的数据写出

out.write(bs, 0, len);

}

// 关流

out.close();

in.close();

long end = System.currentTimeMillis();

System.out.println(end - begin);

}

}

## 1.7.转换流

### 1.7.1.InputStreamReader

转换输入流---将字节流转化为字符流---底层是以字节流形式读取，但是表层是以字符流形式操作

#### 1.7.1.1.构造函数

①InputStreamReader(InputStream in);

创建一个使用默认编码参数为字节流的对象。

②InputStreamReader(InputStream in,Charset cs);

创建一个使用指定编码参数为字节流的对象。

③InputStreamReader(InputStream in,String charsetName);

④InputStreamReader(InputStream in,CharsetDecoder dec);

#### 1.7.1.2.重要方法

①read();

读取单个字符。

②read(char[] cbuf,int offset,int length);

读入字符数组中的某一部分。

### 1.7.2.OutputStreamWriter

转换输出流---将字符流转化为字节流---表层是以字符流形式写出，底层是以字节流形式来传输

#### 1.7.2.1.构造函数

①OutputStreamWriter(OutputStream out);

创建一个使用默认编码参数为字节流的对象。

②OutputStreamWriter(OutputStream out,Charset cs);

③OutputStreamWriter(OutputStream out,CharsetEncoder enc);

④OutputStreamWriter(OutputStream out,String charsetName);

练习：转换文件编码

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

public class ChangeEncodeExer {

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

// 创建一个File对象指向源文件

File old = new File("F:\\java基础增强.txt");

// 创建一个File对象作为临时文件

File temp = new File("F:\\temp");

// 准备缓冲流对象以实现按行读取

BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream(old), "utf-8"));

// 创建一个缓冲流对象将读取的数据写到临时文件中

BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(temp), "gbk"));

// 定义一个字符串记录每行的数据

String line = null;

// 读取数据

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

// 将读取的数据写出

bw.write(line);

bw.newLine();

}

// 关流

br.close();

bw.close();

// 删除源文件

old.delete();

// 将临时文件命名为原文件的名字

temp.renameTo(old);

}

}

## 1.8.系统流/标准流

都是字节流，系统一旦关闭无法二次使用。

System.in 标准输入流

System.out 标准输出流，线程不安全的。

System.err 标准错误流，线程不安全的。

练习：从控制台获取一行数据

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

String str = br.readLine();

System.out.println(str);

Br.close;

## 1.9.打印流

打印流只有输出流，没有输入流。

### 1.9.1PrintStream

提供来一系列格式化打印一个对象的方法。是一个字节流。

#### 1.9.1.1.构造方法

①PrintStream(File file);

②PrintStream(OutputStream out);

#### 1.9.1.2.重要方法

①print(int/long/float/double/char/char[]/boolean/String/Object s);

打印。

②println(/int/long/float/double/char/char[]/boolean/String/Object s);

打印换行。

### 1.9.2.PrintWriter

是一个字符流。

#### 1.9.2.1.构造方法

①PrintWriter(File file);

②PrintWriter(OutputStream out);

③PrintWriter(String fileName);

④PrintWriter(Writer out);

#### 1.9.2.2.重要方法

①print();

②println();

③write();

#### 1.9.2.3.代码示例：

public static void main(String[] args) {

// 表示将数据打印到控制台上

PrintStream ps = new PrintStream(System.out);

// ps.print(new Object());

ps.print(new char[] { 'a', 'b', 'c' });

// BufferedWriter---newLine()

ps.println("abc");

ps.close();

}