# IO流

## 概述与分类

Java中的IO流技术主要用于传输数据。典型的应用场景有：读写本地文件、上传下载文件等。按照数据传输的方向可以分为两种：

* **输入流**（Input）：即让数据进入应用程序中。
* **输出流**（Output）：即让数据离开应用程序。

按照数据的种类还可以有另外一种分法：

* **字节流**：可以处理任何一种数据。
* **字符流**：只能处理纯文本数据。

现阶段我们主要学习：字符输入流、字符输出流，来实现读写文件的功能。

## FileWriter写文件

### 一般步骤

使用java.io.FileWriter类可以将数据写到文本文件当中，使用步骤为三步：

1. 调用构造方法来创建一个FileWriter对象：  
   public FileWriter(String fileName)：构造参数为文件路径名。  
   备注：创建对象时，如果文件不存在，则会自动创建该文件。
2. 调用写数据的成员方法向文件中写入数据：  
   public void write(String text)：参数为需要写入的字符串。
3. 调用关闭的成员方法释放相关资源：  
   public void close()

三步走：“**创、写、关**”。

### 基本代码

**public** **class** Demo01FileWriter {

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

// 1. 创建一个FileWriter对象

FileWriter fw = **new** FileWriter("file01.txt");

// 2. 调用write方法写数据

fw.write("Hello, World!");

// 3. 调用close方法关闭流

fw.close();

}

}

### 覆盖与追加

在使用FileWriter写文件的时候，如果指定的文件不存在，则会自动创建；如果文件已经存在，则会覆盖写入；如果不希望覆盖写入，实现追加写入的效果，则需要使用另一种重载的构造方法：

**public FileWriter(String fileName, boolean append)**

当第2个参数为true时，将会实现追加写入的效果。

**public** **class** Demo02FileWriter {

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

FileWriter fw = **new** FileWriter("file02.txt", **true**); // 创

fw.write("Java"); // 写

fw.close(); // 关

}

}

### 换行符

不同的平台使用不同的字符来代表换行：

* Windows操作系统：**\r\n**（两个字符）
* macOS操作系统：\r（早期）或者\n（现在）
* Linux操作系统：**\n**

**public** **class** Demo03FileWriterLine {

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

FileWriter fw = **new** FileWriter("file03.txt");

fw.write("迪丽热巴\r\n");

fw.write("\r\n"); // 代表一个Windows当中的换行符

fw.write("古力娜扎");

fw.close();

}

}

### 数字与文字的转换

其实所有数据在计算机底层都是二进制数字，所以一切数据都是数字。文件当中存放的其实并不是文字图形，而是数字序列。之所以打开文件可以看到文字图形，是因为软件对文件当中的数字进行了“翻译”。而翻译就需要一个对照词典，如：ASCII码表或Unicode码表。

ASCII当中的字符种类比较少；而Unicode包含了ASCII的内容，还扩展了数以万计的字符，可以涵盖全世界所有的文字。

FileWriter提供了一种重载形式的write方法，可以省去将文字翻译成为数字的步骤，直接在文件当中写对应的数字：

**public void write(int ch)**：参数就是ASCII或Unicode表当中的数字，最小为0。

**public** **class** Demo04FileWriterCharacter {

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

FileWriter fw = **new** FileWriter("file04.txt");

fw.write("Hello");

fw.write(20013); // 代表“中”

fw.write("国");

fw.close();

}

}

### write方法的重载

FileWriter中写数据的方法write其实有5种重载形式，可以在写字符到文件中时提供更加灵活的选择：

* public void write(int ch)：参数为单个文字对应的数字（参考对照码表）。
* public void write(String str)：写完整字符串。
* public void write(String str, int offset, int count)：写字符串的一部分。
* public void write(char[] array)：写完整字符数组。
* public void write(char[] array, int offset, int count)：写字符数组的一部分。

**public** **class** Demo05FileWriterOverload {

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

FileWriter fw = **new** FileWriter("file05.txt");

// 写单个字符

fw.write(65); // A

fw.write(66); // B

fw.write(67); // C

fw.write(48); // 0

fw.write(97); // a

// ===================

// 写一个完整的字符串

fw.write("World\r\n");

fw.write("Hello");

// ===================

// 写一个字符串当中的部分内容

String str = "生前何必久睡，死后自会长眠。";

fw.write(str, 7, 6); // 死后自会长眠

// ===================

**char**[] array = { 'J', 'a', 'v', 'a', '!' };

// 写一个完整的字符数组

fw.write(array);

// 写一个字符数组的部分内容

fw.write(array, 1, 3);

fw.close();

}

}

## FileReader读文件

### 一般步骤

使用java.io.FileReader类可以读取文本文件当中的字符（对应的ASCII或Unicode值）。使用步骤为：

1. 调用构造方法来创建一个FileReader对象：  
   public FileReader(String fileName)：参数仍然为文件的路径名。  
   备注：如果文件不存在，不会自动创建，而会发生异常。
2. 调用读取数据的read方法获取单个字符：  
   public int read()：返回值为单个字符对应的ASCII或Unicode值。  
   备注：每次调用都会读取下一个字符，如果没有更多，将返回-1值。
3. 调用close方法关闭流：  
   public void close()：释放相关资源。

由于并不确定文件当中包含多少个字符，所以次数不确定的重复代码，应该使用while循环。当得到-1值时退出循环。

### 读取单个字符

**public** **class** Demo02FileReaderWhile {

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

FileReader fr = **new** FileReader("file06.txt"); // 创建

// 下面的错误！因为每次read方法执行，都会读取一个新字符，直到-1为止。

// while (fr.read() != -1) {

// System.out.println(fr.read());

// }

**int** ch; // 代表一个单个字符（对应的ASCII/Unicode）

**while** ((ch = fr.read()) != -1) {

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

}

// (x + y) > z

fr.close(); // 关闭

}

}

### 读取字符数组

一个字一个字地读取，很明显效率很低。如果能够五个五个地批量读取，那么效率会有大幅度的提升。FileReader中提供另一种重载形式的read方法：

**public int read(char[] buf)**：参数buf是用来装载多个字符结果的数组，返回值代表数组当中读取到的字符**有效个数**。

**public** **class** Demo04FileReaderArrayWhile {

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

FileReader fr = **new** FileReader("file07.txt");

**char**[] buf = **new** **char**[2];

**int** len; // 代表有效个数

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

String str = **new** String(buf, 0, len);

System.***out***.println(str);

}

fr.close();

}

}

## 读写数据的过程

Java程序并不会直接去操作硬盘，而是交给JVM；JVM再与操作系统交互；操作系统去操作硬盘。然而硬盘的速度比内存慢很多，所以如果每一个单个字符都去操作一次硬盘，性能非常低下。为了提高性能，应该使用数组来达到批量操作，缓冲的目的。

## BufferedWriter

为了提高写文件的效率，java.io.BufferedWriter类内部包含了一个缓冲数组，可以大幅度提高FileWriter写数据的性能。

### 缓冲原理

BufferedWriter类内部包含一个长度为8192的char[]字符数组，每次写数据都是在向数组当中添加字符。如果数组满了，将会整体写到硬盘中；如果数组没满，那么在调用close方法的时候，也会将最后剩余的有效部分写到硬盘中。

### 基本使用

BufferedWriter类的构造方法可以将一个普通的FileWriter作为参数，而使用起来与FileWriter基本一样，性能却大幅度提高。构造方法：

**public BufferedWriter(FileWriter fw)**：构造参数就是普通的FileWriter对象。

**public** **class** Demo01BufferedWriter {

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

// 首先创建一个普通的FileWriter

FileWriter fw = **new** FileWriter("file08.txt");

// 将这个普通的FileWriter对象传递给BufferedWriter构造方法即可。

BufferedWriter bw = **new** BufferedWriter(fw);

// 后面bw的时候，和fw基本没有区别。

bw.write("Hello");

// 关闭流

bw.close();

}

}

### newLine方法

除了性能比FileWriter更高之外，BufferedWriter还提供一个换行的方法：

public void newLine()：该方法会根据操作系统的不同，自动选择写入\r\n或者\r或者\n。

**public** **class** Demo02NewLine {

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

FileWriter fw = **new** FileWriter("file09.txt");

BufferedWriter bw = **new** BufferedWriter(fw);

bw.write("Hello");

// bw.write("\r\n"); // 这种写法不太好，一旦换了操作系统，那么就要修改代码

bw.newLine(); // 自动根据操作系统选择对应的换行符

bw.write("World");

bw.close();

}

}

## BufferedReader

### 缓冲原理

类似地，BufferedReader类内部包含一个长度为8192的char[]字符数组，读数据的时候将会一次最多读取8192个字符放在数组中，而read方法只是从数组中取出字符使用。如果数组被“取空”，会自动再次读取最多8192个字符供使用。从而达到减少硬盘读取次数的目的。

### 基本使用

BufferedReader类的构造方法可以将一个普通的FileReader作为参数，而使用起来与FileReader基本一样，性能却大幅度提高。构造方法：

**public BufferedReader(FileReader fr)**：构造参数就是普通的FileReader对象。

### readLine方法

除了性能比FileReader更高之外，BufferedReader还额外提供一个方法，可以读取一整行字符串：

**public String readLine()**：返回值为一整行字符串内容，不包含换行符。

**public** **class** Demo04ReadLine {

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

FileReader fr = **new** FileReader("file11.txt");

BufferedReader br = **new** BufferedReader(fr);

// String line = br.readLine();

// System.out.println(line); // Hello

//

// line = br.readLine();

// System.out.println(line); // World

//

// line = br.readLine();

// System.out.println(line); // Java

//

// line = br.readLine();

// System.out.println(line); // null

// System.out.println(line == null); // true

String line; // 代表一行字符串

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

System.***out***.println(line);

}

br.close();

}

}

## 练习

### 将集合中的字符串写到文件中

定义一个集合用于存储多个字符串，向其中添加一些字符串，然后将集合的所有字符串内容写到文件中。要求每个字符串独占一行。

**public** **class** Demo01FromListToFile {

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

ArrayList<String> list = **new** ArrayList<>(); // 创建集合

// 加入字符串

list.add("迪丽热巴");

list.add("古力娜扎");

list.add("玛尔扎哈");

BufferedWriter bw = **new** BufferedWriter(**new** FileWriter("练习1.txt"));

// 遍历集合

**for** (**int** i = 0; i < list.size(); i++) {

String str = list.get(i); // 当前字符串

bw.write(str);

bw.newLine(); // 不要忘记换行

}

bw.close(); // 不要忘记关闭流

}

}

### 将文件中的字符串读到集合中

与上一题相反，将文件中的字符串文本读取到集合当中，并且每一行文本作为集合中的一个字符串元素。

**public** **class** Demo02FromFileToList {

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

BufferedReader br = **new** BufferedReader(**new** FileReader("练习1.txt"));

ArrayList<String> list = **new** ArrayList<>();

String line; // 代表一行字符串

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

list.add(line); // 将读取得到的一行字符串绝蛮王到集合当中

}

br.close(); // 关闭流

// 遍历集合

**for** (**int** i = 0; i < list.size(); i++) {

System.***out***.println(list.get(i));

}

}

}

### 反转藏头情诗

将文件当中的每一行字符串读取到集合当中，然后倒序写到另一个文件中，实现行与行之间的反转。

**public** **class** Demo03Reverse {

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

BufferedReader br = **new** BufferedReader(**new** FileReader("练习2-表白信-之前.txt"));

ArrayList<String> list = **new** ArrayList<>();

String line; // 代表一行字符串

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

list.add(line);

}

br.close(); // 关闭输入流

BufferedWriter bw = **new** BufferedWriter(**new** FileWriter("练习2-表白信-之后.txt"));

**for** (**int** i = list.size() - 1; i >= 0 ; i--) {

bw.write(list.get(i));

bw.newLine();

}

bw.close();

}

}

# 方法参数与返回值

方法的参数和返回值可以是任何类型，无论是基本类型，还是引用类型中的数组、字符串甚至是自定义的类，都可以作为方法的参数和返回值。

## 参数

**public** **class** Demo01MethodParam {

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

*method1*(100);

System.***out***.println("============");

**int**[] arr = { 10, 20, 30 };

*method2*(arr);

System.***out***.println("============");

*method3*("How do you do?");

System.***out***.println("============");

Student xueSheng = **new** Student("赵丽颖", 20);

*method4*(xueSheng);

}

// 使用基本类型作为方法的参数，可以

**public** **static** **void** method1(**int** num) {

num += 20;

System.***out***.println(num); // 120

}

// 使用数组作为方法的参数，也可以

**public** **static** **void** method2(**int**[] array) {

System.***out***.println(array[0]); // 10

System.***out***.println(array[1]); // 20

System.***out***.println(array[2]); // 30

}

// 使用字符串作为方法的参数，还可以

**public** **static** **void** method3(String str) {

String result = str.replace("o", "\*");

System.***out***.println(result); // H\*w d\* y\*u d\*?

}

// 使用自定义的类作为方法的参数，照样可以！

**public** **static** **void** method4(Student stu) {

// 姓名：赵丽颖，年龄：20

System.***out***.println("姓名：" + stu.getName() + "，年龄：" + stu.getAge());

}

}

## 返回值

**public** **class** Demo02MethodReturn {

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

**int** result1 = *method1*();

System.***out***.println(result1); // 100

System.***out***.println("==================");

**int**[] result2 = *method2*();

System.***out***.println(result2[0]); // 10

System.***out***.println(result2[1]); // 20

System.***out***.println("==================");

String result3 = *method3*();

System.***out***.println(result3.replace("o", "\*")); // H\*w d\* y\*u d\*?

System.***out***.println("==================");

Student result4 = *method4*();

// 姓名：赵丽颖，年龄：20

System.***out***.println("姓名：" + result4.getName() + "，年龄：" + result4.getAge());

}

// 使用基本类型作为方法的返回值，可以

**public** **static** **int** method1() {

**int** num = 100;

**return** num;

}

// 使用数组类型作为方法的返回值，也可以

**public** **static** **int**[] method2() {

**int**[] array = { 10, 20 };

**return** array;

}

// 使用字符串作为方法的返回值，还可以

**public** **static** String method3() {

String str = "How do you do?";

**return** str;

}

// 使用自定义的类作为方法的返回值，仍然可以

**public** **static** Student method4() {

Student stu = **new** Student("赵丽颖", 20);

**return** stu;

}

}

# 知识总结

1. IO流是用来传输数据的技术，典型应用包括读写文件。
2. IO流的分类：
   1. 字节输入流
   2. 字节输出流
   3. 字符输入流：FileReader、BufferedReader
   4. 字符输出流：FileWriter、BufferedWriter
3. FileWriter一般使用步骤：创、写、关。
   1. 创：FileWriter fw = new FileWriter(“file01.txt”);
   2. 写：fw.write(“Hello”);
   3. 关：fw.close();
4. FileWriter如何才能追加写入呢？构造方法第二个参数为true即可。
5. 如何实现换行？
   1. Windows当中：\r\n
   2. macOS当中：\r或者\n
   3. Linux当中：\n
6. 五种write方法的重载：
   1. 写字符串的整体
   2. 写字符串的部分
   3. 写字符数组的整体
   4. 写字符谁书的部分
   5. 写单个字符对应的数字（参考ASCII码表或者Unicode表）
7. FileReader一般使用步骤：创、读、关。
   1. 创：FileReader fr = new FileReader(“file01.txt”);
   2. 读单个字符：  
      int ch; // 单个字符的ASCII值  
      while ((ch = fr.read()) != -1) {  
       // ...  
      }
   3. 读字符数组：  
      char[] buf = new char[1024];  
      int len; // 有效个数  
      while ((len = fr.read(buf)) != -1) {  
       String str = new String(buf, 0, len);  
      }
8. BufferedXxx和FileXxx有什么区别？BufferedXxx有8192长度的char[]字符数组作为缓冲，所以性能更加高一些。
9. BufferedWriter有额外的方法：public void newLine()，根据操作系统自动选择写入换行符。
10. BufferedReader有额外的方法：public String readLine()，读取一行字符串，不包含换行符。
11. 关于方法的参数和返回值类型：
    1. 任何数据类型（基本类型、数组、字符串、自定义的类）都可以作为参数。
    2. 任何数据类型（基本类型、数组、字符串、自定义的类）都可以作为返回值。