<https://blog.csdn.net/zhuwenchao90/article/details/54601993>

**1、IO的分类**

A、按照流向【参照物JVM】   
输入流 : 读取数据   
输出流 : 写出数据   
B、按照数据类型   
(1)字节流   
a、字节输入流 读取数据 InputStream   
b、字节输出流 写出数据 OutputStream   
(2)字符流   
a、字符输入流 读取数据 Reader   
b、字符输出流 写出数据 Writer

注意：一般我们在探讨IO流的时候，如果没有明确说明按哪种分类来说，默认情况下是按照数据类型来分的。

注意：每种基类的子类都是以父类名作为后缀名。   
XxxOutputStream   
XxxInputStream   
XxxReader   
XxxWriter

**2、字节流的抽象父类 及基本方法**

**InputStream**【抽象类】输入流:JVM从中连续读取字节的对象。

int read()：从输入流中读取数据的下一个字节。返回 0 到 255 范围内的 int 字节值，如果返回-1表示遇到流的末尾，结束。

int read(byte[] b)：读入b.length个字节放到b中，并返回实际读入的字节。

int read(byte[] b,int off,int len)：这个方法表示把流中的数据读到,数组b中,第off个开始的len个数组元素中。

void close()：在操作完一个流后要使用此方法将其关闭， 系统就会释放与这个流相关的资源。

* 1
* 2
* 3
* 4

**OutputStream**【抽象类】输出流：JVM向其中连续写入的对象。

void write(int b)：将指定的字节写入此输出流。

void write(byte[] b)：将 b.length 个字节从指定的 byte 数组写入此输出流。

void write(byte[] b,int off, int len)：将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此输出流。

void flush()：刷新此输出流，并强制写出所有缓冲的输出字节。【彻底完成输出并清空缓冲区】。

* 1
* 2
* 3
* 4

**3、为何需要调用close()方法？为什么需要缓冲？使用缓冲区技术的优缺点？**

**3.1为什么需要调用close()方法？**

尽管在调用流对象的，在没有引用变量指向它时会变成垃圾，最终垃圾回收器会自动回收。在程序创建一个IO流对象，除了Java程序中我们可见的实例对象外，还有系统本身的资源，Java垃圾回收器只能管理程序中的类的实例对象，没法去管理系统产生的资源，所以程序需要调用close方法，去通知系统释放其自身产生的资源。

**3.2 为何血需要缓冲区？缓冲技术的优缺点？**

计算机访问外部设备,要比直接访问内存慢得多,如果我们每一次write方法的调用都直接写到外部设备(如直接写入硬盘文件),CPU就要花费更多的时间等待外部设备；如果我们开辟一个内存缓冲区,程序的每一次write方法都是写到这个内存缓冲区中,只有这个缓冲区被装满后,系统才将这个缓冲区的内容一次集中写到外部设备。   
优点：有效的提高了CPU效率。   
缺点：由于缓冲区，数据并没有立即写入到目标中去，就会造成一定的滞后。   
缓冲区技术的实现是由编程语言本身决定的：   
C语言：默认情况下就会使用缓冲区。   
Java:有的类使用了缓冲区，有的类没有使用缓冲区。   
flush()方法在缓冲区没有满的情况下，也将缓冲区的内容强制写入外设【刷新】。在调用close()方法，系统在关闭这个流之前也会将缓冲区的内容刷新到硬盘文件。

**4、OutputStream输出流的write()方法使用**

如何向文本中写入字节？如何实现数据追加？如何实现数据换行？

**4.1 直接写入字节**

public class FileOutputStreamDemo {

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

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

fos.write("hello,IO".getBytes());//获取本地编码的字节数组，并写入文本文件中。

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

fos.close();

}

}

/\* IO.txt的内容：

hello,IOjava

\*/

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

**4.2 写入字节的3种方式**

public class FileOutputStreamDemo2 {

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

//如果第二个参数为 true，则将字节写入文件末尾处，而不是写入文件开始处。

FileOutputStream fos = new FileOutputStream("IO.txt", true);

fos.write(65); //65 -- 底层二进制数据 -- 通过记事本打开 -- 找65对应的字符值 -- a

byte[] bys={97,98,99,100,101};

fos.write(bys);

fos.write(bys,1,3);

fos.close();

}

}

/\* IO.txt文本的内容：

Aabcdebcd

\*/

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

**4.3关于数据换行**

不同的系统针对不同的换行符号识别是不一样的。   
1、windows:\r\n   
2、linux:\n   
3、Mac:\r

public class FileOutputStreamDemo3 {

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

FileOutputStream fos = new FileOutputStream("IO.txt", true);

// 写数据

for (int x = 0; x < 3; x++) {

fos.write(("Hello Java" + x).getBytes());

fos.write("\r\n".getBytes());

}

fos.close();

}

}

/\* IO.txt文本的内容：

Hello Java0

Hello Java1

Hello Java2

\*/

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16

**4.4 IO流异常处理的代码**

开发中还必须考虑异常处理情况下的输入流的操作

public class FileOutputStreamDemo4 {

public static void main(String[] args) {

//为了在finally里面能够看到该对象就必须定义到外面，为了访问不出问题，还必须给初始化值

FileOutputStream fos = null;

try {

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

fos.write("Java,Hello".getBytes());

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} finally {

//如果fos不是null，才需要close()

if (fos != null) {

try {

fos.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23

**5、InputStream输入流的read()方法使用**

输入流操作

public class FileInputStreamDemo {

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

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

int by = 0;

//读取，赋值，判断

while ((by = fis.read()) != -1) {

System.out.print((char)by);

}

fis.close();

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

**6、通过IO流进行文件复制**

**6.1 基本的文件复制**

public class CopyFileDemo {

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

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

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

int by = 0;

while ((by = fis.read()) != -1) {

fos.write(by);

}

fos.close();

fis.close();

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12

**6.2 计算机是如何识别应该把2个字节转换为一个汉字？**

在GBK字符编码集中，汉字是由2个字节组成，因为GBK兼容ISO-8859-1,正数的单字节已被占用   
所以汉字的第一个字节必须为负数第二个字节大多也为负数。如：

public class StringDemo {

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

String s1 = "Xyz123";

String s2 = "手可摘星辰";

byte[] bys1 = s1.getBytes();

byte[] bys2 = s2.getBytes();

System.out.println(Arrays.toString(bys1));

System.out.println(Arrays.toString(bys2));

}

}

/\* GBK编码字符集下：

[88, 121, 122, 49, 50, 51]

[-54, -42, -65, -55, -43, -86, -48, -57, -77, -67]

\* \*/

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

**6.3 IO操作时定义一个字节数组作为缓存**

public class CopyFileDemo {

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

FileInputStream fis = new FileInputStream("c:\\a.txt");

FileOutputStream fos = new FileOutputStream("d:\\b.txt");

byte[] bys = new byte[1024];

int len = 0;

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

fos.write(bys, 0, len);

}

fos.close();

fis.close();

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13

**6.5 通过带有缓冲区的字节类【高效类】**

写数据：BufferedOutputStream   
读数据：BufferedInputStream   
看源码:   
BufferedOutputStream 继承【过滤流】 FilterOutputStream。进一步地重写过滤流方法中的一些方法，并且还可以提供一些额外的方法和字段。   
FilterOutputStream继承自抽象类OutputStream，过滤类本身只是简单地重写那些将所有请求传递给所包含输出流的 OutputStream 的所有方法。   
实际上把缓冲写在包装在类中，BufferedInputStream原理类似。   
通过定义数组的方式比一次读取一个字节的方式快很多，拥有缓冲区效率提升很多。   
Java在设计时提供缓冲区的字节类BufferedOutputStream和BufferedInputStream。   
真正的底层读写数据还是依靠基本流对象来实现，见源码解析。   
【BufferedOutputStream】源代码解析

public class BufferedOutputStream extends FilterOutputStream {

protected byte buf[]; //内部缓冲区

protected int count; //缓冲区存储的字节个数

public BufferedOutputStream(OutputStream out) {

this(out, 8192);

}

public BufferedOutputStream(OutputStream out, int size) {

super(out);

if (size <= 0) {

throw new IllegalArgumentException("Buffer size <= 0");

}

buf = new byte[size];//开辟一个缓冲区

}

//刷新内部缓冲区

private void flushBuffer() throws IOException {

if (count > 0) {

out.write(buf, 0, count);

count = 0;

}

}

//将指定的字节写入此缓冲的输出流

public synchronized void write(int b) throws IOException {

if (count >= buf.length) {

flushBuffer();

}

buf[count++] = (byte)b;

}

//将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此缓冲的输出流

public synchronized void write(byte b[], int off, int len) throws IOException {

if (len >= buf.length) {

flushBuffer();

out.write(b, off, len);

return;

}

if (len > buf.length - count) {

flushBuffer();

}

System.arraycopy(b, off, buf, count, len);

count += len;

}

public synchronized void flush() throws IOException {

flushBuffer();

out.flush();

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49

**6.7 比较4中IO复制操作的效率**

/\*

\* 需求：把F:\\红玫瑰.mp3【9.25M】复制到当前项目目录下的copy.mp4中

\*/

public class CopyMp4Demo {

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

long start = System.currentTimeMillis();

method4("F:\\红玫瑰.mp3", "copy.mp3");

long end = System.currentTimeMillis();

System.out.println("共耗时：" + (end - start) + "毫秒");

}

//1、基本字节流一次读写一个字节

public static void method1(String srcString, String destString)throws IOException {

FileInputStream fis = new FileInputStream(srcString);

FileOutputStream fos = new FileOutputStream(destString);

int by = 0;

while ((by = fis.read()) != -1) {

fos.write(by);

}

fos.close();

fis.close();

}

//2、基本字节流一次读写一个字节数组

public static void method2(String srcString, String destString)throws IOException {

FileInputStream fis = new FileInputStream(srcString);

FileOutputStream fos = new FileOutputStream(destString);

byte[] bys = new byte[1024];

int len = 0;

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

fos.write(bys, 0, len);

}

fos.close();

fis.close();

}

//3、高效字节流一次读写一个字节：

public static void method3(String srcString, String destString)throws IOException {

BufferedInputStream bis = new BufferedInputStream(new FileInputStream(srcString));

BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(destString));

int by = 0;

while ((by = bis.read()) != -1) {

bos.write(by);

}

bos.close();

bis.close();

}

//4、高效字节流一次读写一个字节数组：

public static void method4(String srcString, String destString)throws IOException {

BufferedInputStream bis = new BufferedInputStream(new FileInputStream(srcString));

BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(destString));

byte[] bys = new byte[1024];

int len = 0;

while ((len = bis.read(bys)) != -1) {

bos.write(bys, 0, len);

}

bos.close();

bis.close();

}

}

/\*

\* 字节流4种方式复制文件所耗费的时间：

\* method1基本字节流一次读写一个字节： 共耗时：115906毫秒

\* method2基本字节流一次读写一个字节数组：共耗时：342毫秒

\* method3高效字节流一次读写一个字节： 共耗时：550毫秒

\* method4高效字节流一次读写一个字节数组：共耗时：61毫秒

\*\*/

**7、IO流字符集可能出现的乱码**

/\*

\* 字节流读取中文可能出现的乱码问题：

\* a.txt文件的具体内容

\* 手可摘星辰

\* 2017.01.18.Java

\*/

public class FileInputStreamDemo {

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

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

/\*//读取数据，英文正常，中文乱码异常。【此方案仅对单字节字符有效】

int by = 0;

while ((by = fis.read()) != -1) {

System.out.print((char) by);

} \*/

/\*运行结果：??????????

2017.01.18.Java \*/

//解决：采用系统默认的编码字符集

byte[] bys = new byte[1024];

int len = 0;

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

System.out.print(new String(bys, 0, len));

}

fis.close();

/\* 手可摘星辰

\* 2017.01.18.Java\*/

}

}

关于字符串的编解码说明

/\*

\* 乱码产生原因：编解码采用不同方案所致。

\* 解决方案：编解码采用同一套字符集。

\* 编码：byte[] -- String : new String(byte[] bytes, String CharsetName )

\* 解码：String -- byte[] : getBytes(String CharsetName);

\* \*/

public class StringDemo {

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

String s = "中国";

byte[] bys1 = s.getBytes();

byte[] bys2 = s.getBytes("GBK");

byte[] bys3 = s.getBytes("UTF-8");

System.out.println(Arrays.toString(bys1));

System.out.println(Arrays.toString(bys2));

System.out.println(Arrays.toString(bys3));

String s1 = new String(bys1);

String s2 = new String(bys2, "GBK");

String s3 = new String(bys3, "UTF-8");

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

}

}

/\* 运行结果：

[-42, -48, -71, -6]

[-42, -48, -71, -6]

[-28, -72, -83, -27, -101, -67]

中国

中国

中国

\* \*