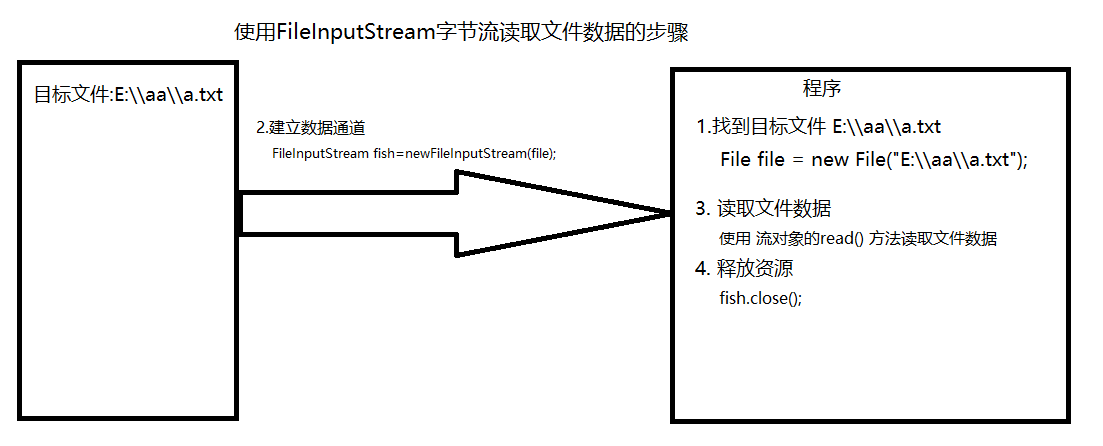
# 1. 输入字节流（读数据）

----| InputStream 所有输入字节流的基类，继承Object类。是abstract抽象类。

--------| FileInputStream读取文件数据的输入字节流。

## 1.1 读取数据的步骤（使用FileInputStream读取文件数据）

**1.找到目标文件 File file = new File("E:\\aa\\bb\\a.txt")  
2.搭建数据传输通道（桥梁） FileInputStream fis = new FileInputStream(file);  
3.传输数据(读取数据) fis.read()  
4.关闭通道（释放资源）fis.close()**



## 1.2 案例

|  |
| --- |
| **public class** Dome1 {  **public static void** main(String[] args) **throws** IOException {  // 1.确定目标文件  File file = **new** File(**"E:\\aa\\bb\\aaa.wmv"**);   // 2.搭建桥梁  FileInputStream fileInputStream = **new** FileInputStream(file);   // 3.读取数据  // 创建缓冲字节数组  **byte**[] bytes = **new byte**[1024 \* 10];//存储读取到的数据。缓冲数组的长度一般是1024的倍数，与计算机的处理单位保持一致。理论上缓冲数组越大，效率越高  **int** length = 0;//保存每次读取到的字节个数。  // 循环读取文件内容  **while** ((length = fileInputStream.read(bytes)) != -1) {//read()读取不到数据时返回-1  System.***out***.println(**"读取到的内容是："** + **new** String(bytes, 0, length));// 使用String将字符数组构建成字符串  }   // 4.释放资源  fileInputStream.close();  } } |

## 1.3读取文件注意细节

问题1：FileInputStream读取文件的时候，不关闭资源会有什么影响？

答案：资源文件一旦使用完毕，需要马上释放资源，否则其他的程序无法对该资源文件进行其他操作。

问题2：使用String将字符数构建字符串时，new String(bytes) 与 new String(bytes,0,length)的区别

答案：

length是数组长度。（另外，本人知道这两种模式的区别，但是不知道该怎么表示，欢迎大佬们使用专业术语进行更正，万分感谢。）

（1）使用new String(bytes)将数组构建成字符串时，使用的是**覆盖机制。**

第一次读取length个字节，

第二次再读取length个字节，直接覆盖掉第一组，

依次类推……

当最后一组不足length个字节时，会将最后一组的字节覆盖到倒数第二组对应的位置，不足length个字节的部分，会使用倒数第二组未被覆盖的部分补足，这样就产生了弊端。**不建议使用**。

（2） 使用 new String(bytes,0,length)将数组构建成字符串时，使用的是**先清空再读取的机制。**

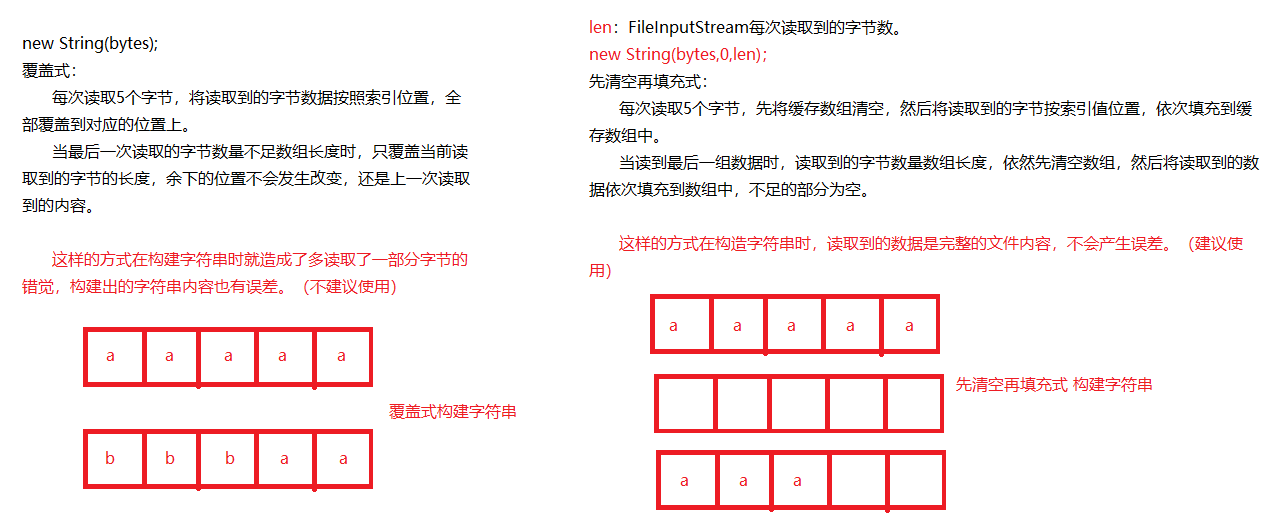
第一次读取length个字节，

清空第一组字节，然后读取第二组length个字节，

以此类推……

当最后一组不足length个字节时，将最后一组覆盖到数组中，不足部分为空。

注意：new String(bytes) 与 new String(bytes,0,length)的区别示意图如下



# 2. 输出字节流（写数据）

* 输出字节流 FileOutputStream

- - - - | OutputStream 是所有 字节输出流的基类，抽象类

- - - - - - - - | FileOutputStream 向文件输出数据的输出字节流

## 2.1 输出字节流使用步骤FileOutputStream

1.找到目标文件

File file = new File("E:\\aa\\bb\\a.txt");

2.搭建数据通道

FileOutputStream fot = new FileOutputStream(file);先清空目标文件中的数据，再将新的数据写出到硬盘。

FileOutputStream fot = new FileOutputStream(file, true);向文件中数据的末尾追加需要写出的数据内容。

3.传输数据

fileOutputStream.write( byte[] bytes);//将需要写出(输出)的数据以byte数组的形式传输出去。

fileOutputStream.write( byte[] bytes, int off, int len);

bytes：需要写出的数据，需要以byte数组的形式传输。

off：数组中的下标，表示从数组中索引值off出开始写出数据。

len：写出多少个字节的数据。

4.释放资源

fot.close();

## 2.2 输出字节流要注意的细节FileOutputStream

1. 使用FileOutputStream的时候，如果目标文件不存在，那么会自动创建目标文件，然后向文件中写入数据。

2. 使用FileOutputStream的时候，如果目标文件已经存在，那么会先清空文件中的数据，再向文件中写入数据。

3. 使用FileOutputStream的时候，如果目标文件已经存在，需要向文件中原来数据的基础上追加数据的时候，那么需要使用 FileOutputStream(File file,Boolean append)构造函数，第二个参数为true

4. 使用FileOutputStream方法写出int类型的数据时，虽然接受的是int类型的数据，但是写出的只是一个字节的数据，即只是将**最低八位的二进制数据写出，其他二十四位数据全部丢弃。**

## 2.3 案例

### 2.3.1 向文件中写出数据 （先清空再写入，即新内容覆盖掉原来的内容）

|  |
| --- |
| **public class** Dome3 {  **public static void** main(String[] args) **throws** IOException {  // 1.找到目标文件  File file = **new** File(**"E:\\aa\\bb\\a.txt"**);  // 2.搭建数据传输通道（先清空文件原来内容，再将新的内容写出到文件中）  FileOutputStream fileOutputStream = **new** FileOutputStream(file);  // 3.写出数据  String str = **"hello word!!!"**;//需要向文件写出的数据  **byte**[] bytes = str.getBytes();// 将字符串转换成字节数组  fileOutputStream.write(bytes, 0, bytes.**length**);//向硬盘写出数据  // 4.释放资源  fileOutputStream.close();  } } |
| **程序执行前文件内数据：aaaaaaaaaaaaa** |
| **程序执行后文件内数据：hello word!!!** |

### 2.3.2 向文件中写出数据 （向文件中原内容的末尾追加新内容）

|  |
| --- |
| **public class** Dome4 {  **public static void** main(String[] args) **throws** IOException {  // 1.找到目标文件  File file = **new** File(**"E:\\aa\\bb\\a.txt"**);  // 2.搭建数据传输通道(向文件原内容末尾处，追加新的内容)  FileOutputStream fileOutputStream = **new** FileOutputStream(file, **true**);  // 3.写出数据  String str = **"世界 你好！！！"**;//需要向文件写出的数据  **byte**[] bytes = str.getBytes();// 将字符串转换成字节数组  fileOutputStream.write(bytes, 0, bytes.**length**);//向硬盘写出数据  // 4.释放资源  fileOutputStream.close();  } } |
| **执行程序前文件内容：hello word!!!** |
| **执行程序后文件内容：hello word!!!世界 你好！！！** |

## 2.4 FileOutputStream的工作原理

每当创建一个新的字节输出流对象（new FileOutputStream(file)）时，默认情况下指针都会指向文件开始的位置。每当写出一次数据，指针都会出现相对的移动。（这就是先清空再写出）

当使用FileOutputStream(File file, boolean append)时，当append=true时，指针指向的是文件的结束部分，此时再写出的数据，就是在文件原数据的基础上，追加新的数据。（这就是追加数据）

# 3.处理异常（字节流应用）

## 需求：拷贝一份视频

要求：边读边写，并且处理异常

提示：

1. 处理的异常的目的(步骤)：

(1)阻止下面代码的执行（结束程序）  
(2)通知调用者这里出现了错误

2. 处理异常throw RuntimeException(IOException e)；

(1) 将 IOException异常传递给 RuntimeException包装一层，然后再抛出。这样做的目的是为了让调用者使用变得更灵活。

(2) RuntimeException运行时异常，出现这种异常时，jvm会自动阻断后边程序的运行，并将异常信息通知给调用者。

|  |
| --- |
| **public class** Dome2 {  **public static void** main(String[] args) **throws** IOException {  // 1.找到目标文件  File readFile = **new** File(**"E:\\aa\\bb\\aaa.wmv"**);//被读入的目标文件  File writeFile = **new** File(**"E:\\aa\\bb\\复制的视频.wmv"**);//被写出的目标文件   // 2.搭建桥梁  FileInputStream fileInputStream = **new** FileInputStream(readFile);//输入  FileOutputStream fileOutputStream = **new** FileOutputStream(writeFile);//输出   // 3.传输数据，边读边写  **byte**[] bytes = **new byte**[1024];//建立缓冲数组  **int** length = 0;// 声明变量，用于存储输入流读到的数据  **while** ((length = fileInputStream.read(bytes)) != -1) {//读入（当字节输入流对象读不到数据时，即读到文件数据结尾时，返回-1）  fileOutputStream.write(bytes);//写出  }   // 4.释放资源(原则：先开后关，后开先关)  fileOutputStream.close();  fileInputStream.close();  } } |

# 4.BufferedInputStream 缓冲输入字节流

## 4.1 BufferedInputStream：

通过使用FileInputStream我们可以知道，使用缓冲数组能提高读取效率。因此sun给我们提供了一个缓冲输入字节流对象，让我们可以更高效的读取文件。

## 4.2 输入字节流体系

- - - - | InputStream 输入字节流的基类，抽象类

- - - - - - - - - | FileInputStream 读取文件数据的输入字节流（直接从硬盘读取），每次读取一个字节

- - - - - - - - - | BufferedInputStream 缓冲输入字节流，它的出现主要是为了提高读取文件数据的效率。其实BufferedInputStream只不过是在内部维护了一个8192字节(8kb)的字节数组而已。

## 4.3使用BufferedInputStream的步骤：

1.找到目标文件

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

2.建立数据通道

BufferedInputStream bufferedInputStream = new BufferedInputStream(newFileInputStream(file));

3.读取数据

int content = bufferedInputStream.read()；

4.释放流资源(实际上关闭的是 FileInputStream的资源。)

bufferedInputStream.close();

## 4.4 关于 BufferedInputStream 的问题：

### （1）为什么创建BufferedInputStream对象的时候，要传入FileInputStream对象？

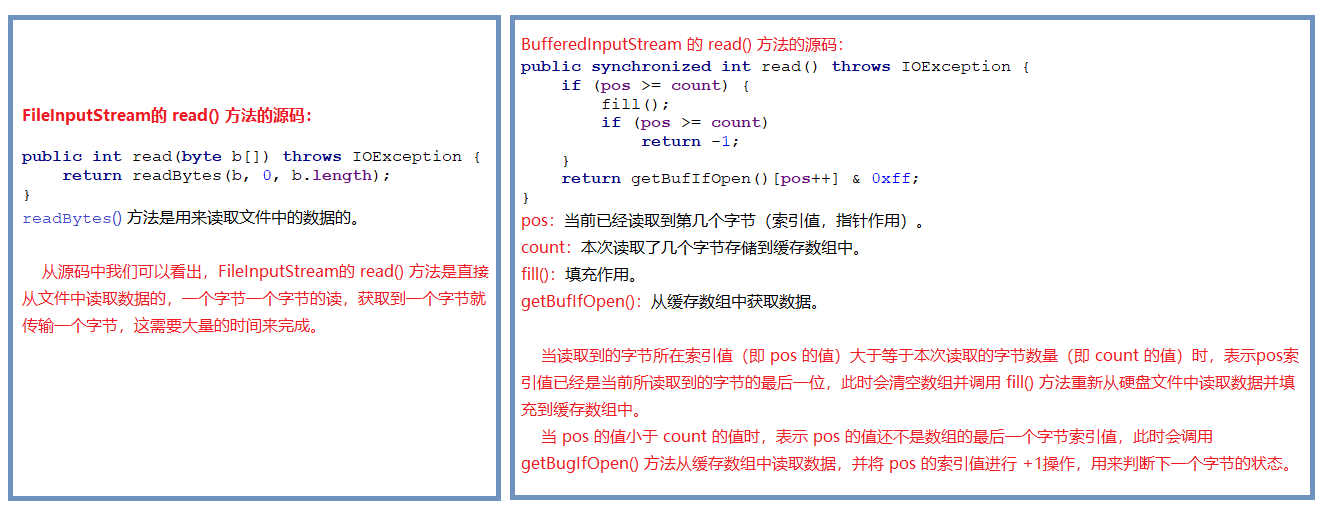
答：因为所有的缓冲字节流都不具备读写文件的能力。所以BufferedInputStream不具备读写文件的能力，不能直接从硬盘中读取文件中的数据。需要借助FileInputStream的read()的读取文件数据的能力来完成文件数据的读取操作。

### （2）BufferedInputStream出现的目的是为了提高读取数据的效率，但是BufferedInputStream每次读取一个字节的数据，而FileInputStream也是每次读写一个字节的数据，那么BufferedInputStream的高效率从何而来呢？

答：BufferedInputStream只不过是在类的内部维护了一个8192字节(8kb)的字节数组而已。

从BufferedInputStream的源码看它的运行原理：

|  |
| --- |
| BufferedInputStream的源码:  public synchronized int read() throws IOException {  if (pos >= count) {  fill();  if (pos >= count)  return -1;  }  return getBufIfOpen()[pos++] & 0xff;  } |
| pos：当前已经读取到了第几个字节(指针作用)  count：本次读取了几个字节存储到了缓冲数组中。  getBufIfOpen():获取字节数组中的数据。  fill()：填充。获取getBufIfOpen()获取到的字节数组中的数据。使用BufferedInputStream内部维护的数组去读取文件数据，每次最多读取8kb(8192字节)的数据进入缓冲数组。 |
| 通过源码可以得知BufferedInputStream缓冲字节输入流的运行原理：  BufferedInputStream是从字节数组(也就是内存中)读取数据的，它没有读写文件的能力。  第一次读取时pos和count都是0，此时进入if内部调用fill()方法，fill()维护getBufIfOpen()方法一次性读取的最多8kb的数据。  fill()方法起到一个指针的作用，每读取完一个字节指针索引值+1,直到pos>=count，return -1读取完字节数组中的所有数据，然后pos指针清零，继续下一轮的读取操作，直到将文件中的所有数据都读取完成。 |



### （3）当使用完流资源，要第一时间将流资源关闭，为什么只关闭BufferedInputStream的流资源，而不关闭FileInputStream的流资源？

答：关闭BufferedInputStream的资源，实际上关闭的就是FileInputStream的资源，因为BufferedInputStream不具备读取文件的能力，所以不需要释放资源。而它读取文件的能力是借助的FileInputStream的read()方法，所以关闭的实际上是FIleInputStream的资源。

## 4.5 BufferedInputStream与FileInputStream的使用选取问题？

看个人习惯，总的来说，使用FileInputStream搭配数组的效率应该比BufferedInputStream的效率要高。

从以下几点可以分析出：

1.执行流程：从BufferedInputStream的源码可以看出，每次读取字节都需要进行if判断，这就很影响工作效率了。

2.程序员所需写的代码量：两种字节输入流使用时，所需的代码量是相同的。只不过FileInputStream需要自己去维护一个byte数组而BufferedInputStream却需要将FileInputStream传入它构建的对象中。

3.维护数组：BufferedInputStream内部维护的数组是8kb(8192个字节)，也就是1024\*8个字节，是固定的。而程序员自己去维护数组的话，会更加灵活。

## 4.6 案例

|  |
| --- |
| **public class** Dome1 {  /\*  3.使用BufferedInputStream的步骤：  1.找到目标文件  2.建立数据通道  3.读取数据  4.释放流资源  \*/  **public static void** main(String[] args) {  // 1.找到目标文件  File file = **new** File(**"E:\\aa\\bb\\a.txt"**);  FileInputStream fileInputStream = **null**;  BufferedInputStream bufferedInputStream = **null**;   **try** {  // 2.搭建数据通道  // 疑问1：问什么在创建BufferedInputStream的时候，要传入FileInputStream？ 因为BufferedInputStream(所有缓冲字节流)不具备读写文件的能力，需要借助FileInputStream的read()方法读取文件数据内容。  fileInputStream = **new** FileInputStream(file);  bufferedInputStream = **new** BufferedInputStream(fileInputStream);   // 3.读取数据  /\*  缓冲字节输入流的运行原理：BufferedInputStream只不过是在类的内部维护了一个8192字节(8kb)的字节数组而已。  BufferedInputStream源码：  public synchronized int read() throws IOException {  if (pos >= count) {  fill();  if (pos >= count)  return -1;  }  return getBufIfOpen()[pos++] & 0xff;  }  pos：当前已经读取到了第几个字节(指针作用)  count：本次读取了几个字节存储到了缓冲数组中。  getBufIfOpen():获取字节数组中的数据。  fill()：填充。获取getBufIfOpen()获取到的字节数组中的数据。使用BufferedInputStream内部维护的数组去读取文件数据，每次最多读取8kb(8192字节)的数据进入缓冲数组。   通过源码可以得知BufferedInputStream缓冲字节输入流的运行原理：  BufferedInputStream是从字节数组(也就是内存中)读取数据的，  第一次读取时pos和count都是0，此时进入if内部调用fill()方法，fill()维护getBufIfOpen()方法一次性读取的最多8kb的数据。  fill()方法起到一个指针的作用，每读取完一个字节指针索引值+1,直到pos>=count，return -1读取完文件中的所有数据。  \*/  **int** content = 0;  **while** ((content = bufferedInputStream.read()) != -1) {  System.***out***.println((**char**) content);  }  } **catch** (FileNotFoundException e) {  System.***out***.println(**"寻找目标文件失败..."**);  **throw new** RuntimeException(e);// 将异常封装到运行时异常处理掉  } **catch** (IOException e) {  System.***out***.println(**"读取文件内容失败..."**);  **throw new** RuntimeException(e);  } **finally** {  **try** {  // 4.释放资源  System.***out***.println(**"关闭流资源成功......"**);  bufferedInputStream.close();//实际上释放的是FileInputStream的资源，因为BufferedInputStream不具备读写文件的能力，所以不能搭建与文件之间的通道，它借助的是FileInputStream的read()方法来读写文件的数据。  } **catch** (IOException e) {  System.***out***.println(**"关闭流资源失败......"**);  **throw new** RuntimeException(e);  }  }  } } |

# 5. BufferedOutputStream

## 5.1 输出字节流

- - - - | OutputStream 输出字节流的基类，抽象类

- - - - - - - - | FileOutputStream向文件输出数据的 输出字节流

- - - - - - - - | BufferedOutputStream缓冲输出字节流。BufferedOutputStream出现的目的是为了提高向文件输出数据的效率。其内部维护了一个8kb(8192字节)的数组。

## 5.2使用BufferedOutputStream的步骤：

1. 找到目标文件

2. 搭建数据通道

3. 将数据写出到BufferedOutputStream 维护的字节数组中

4. 将字节数组(内存)中的数据写出到硬盘文件中。

## 5.3 BufferedOutputStream需要注意的细节

1. 在使用BufferedOutputStream写数据的时候，它的write方法是将数据写入到它内部维护的数组中的，而不是直接写入到内存中。因为BufferedOutputStream不具备向文件中写入数据的能力。

2.使用BufferedOutputStream向文件写入数据的三种情况

* + 首先使用BufferedOutputStream的write方法，将数据写入到字节数组中（也就是内存中）

1. 第1种情况：使用flush方法，将内存中的数据刷入硬盘文件中。
2. 第2种情况：使用close方法，将内存中的数据刷入硬盘文件中。其实它内部调用的是flush和close两个方法，向将输入flush刷入硬盘文件中，再关闭FileOutputStream流资源。
3. 第3种情况：当字节数组已经填满数据时（即字节数组已经存储了8kb字节），它将会自动将数组中的数据刷入硬盘文件中。

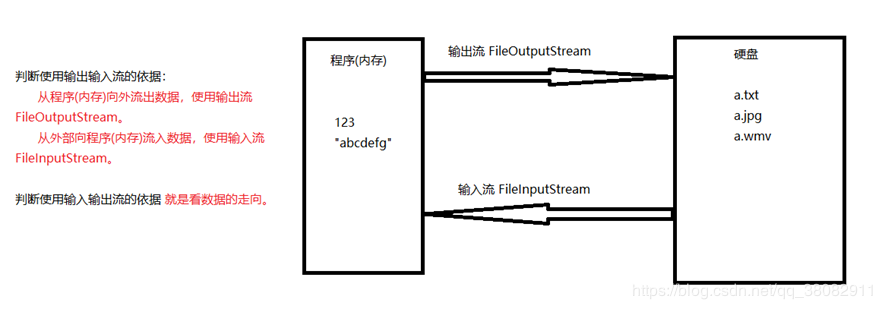
## 5.4 案例

|  |
| --- |
| **public class** Dome3 {  **public static void** main(String[] args) {  // 1.找到目标文件  File readFile = **new** File(**"E:\\aa\\bb\\aaa.wmv"**);  File writeFile = **new** File(**"E:\\aa\\bb\\复制品.wmv"**);   // 2.搭建数据通道  FileInputStream fileInputStream = **null**;  FileOutputStream fileOutputStream = **null**;  BufferedInputStream bufferedInputStream = **null**;  BufferedOutputStream bufferedOutputStream = **null**;   **try** {  fileInputStream = **new** FileInputStream(readFile);  bufferedInputStream = **new** BufferedInputStream(fileInputStream);   fileOutputStream = **new** FileOutputStream(writeFile);  bufferedOutputStream = **new** BufferedOutputStream(fileOutputStream);  // 3.读写数据  **int** len = 0;  /\*byte[] bytes = new byte[1024];  int content = bufferedInputStream.read(bytes, 0, bytes.length); \*/ //如果read读取的文件数据传入了缓冲数组，那么读取的内容存储到缓冲数组中，返回值是存储到缓冲数组中的字节数量。  **while** ((len = bufferedInputStream.read()) != -1) { // 如果read读取的文件数据没有存储到缓冲数组中，那么返回值是读取到的数据内容。注意：这里说的缓冲数组是程序员自己维护的字节数组，不是缓冲流类中默认维护的8kb的数组。  bufferedOutputStream.write(len);  }   } **catch** (FileNotFoundException e) {  System.***out***.println(**"寻找目标文件发生异常......"**);  **throw new** RuntimeException(e);  } **catch** (IOException e) {  System.***out***.println(**"读写数据发生异常......"**);  **throw new** RuntimeException(e);  } **finally** {  // 4.关闭流资源  **try** {  System.***out***.println(**"关闭缓冲字节输出流成功......"**);  bufferedOutputStream.close();  System.***out***.println(**"关闭缓冲字节输入流成功......"**);  bufferedInputStream.close();  } **catch** (IOException e) {  System.***out***.println(**"关闭流资源失败......"**);  **throw new** RuntimeException(e);  }  }  } } |

# 6. 字节流总结

## 6.1 判断使用输入流还是输出流的依据

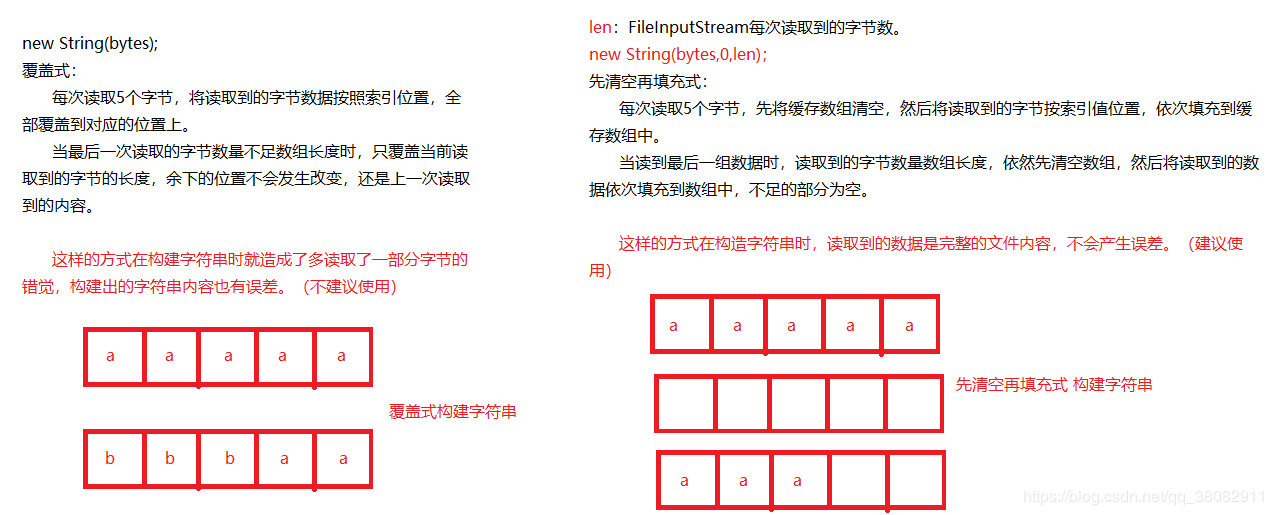
* 以当前程序（也就是内存）为参照物，从硬盘向内存中读取数据，使用输入流 FileInputStream。从内存向硬盘写出数据，使用输出流 FileOutputStream。



## 6.2 FileInputStream 读取文件

    使用FIleInputStream读取文件时，我们通常维护一个缓冲数组来提高读取效率。假如我们维护的这个缓冲字节数组的长度是5，而我们读取的文件内数据的长度是8。那么当我们使用 new String() 来构建字符串时，它是怎么读取的呢？

    使用缓冲数组读取文件内容，然后构建字符串，有两种情况，第一种是覆盖式的读取，第二种是先清空再填充。两种方式如下图：

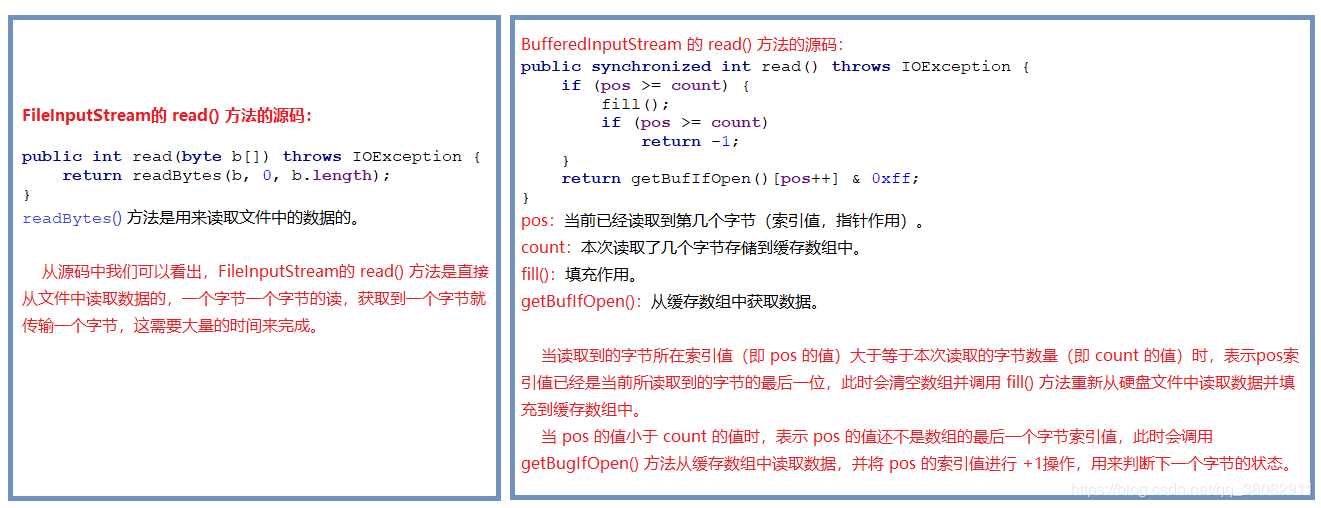
​

## 6.3 BufferedInputStream 读取效率比 FileInputStream 高的原因？

     BufferedInputStream 的类内部，维护了一个缓存数组。但是它不具备读文件的功能，需要依赖 FileInputStream 的read()方法将文件中的数据读取到内存中，BufferedInputStream 在类的内部维护了一个 8kb的数组，FileInputStream 的 read() 方法将硬盘中的数据读取到这个缓存数组中，然后 BufferedInputStream才能使用 read() 方法从这个数组中读取数据。

    BufferedInputStream 每次读取一个字节的数据，而 FileInputStream 每次也读取一个字节的数据，那么为什么 BufferedInputStream 读取效率要高于 FileInputStream呢？

**看段源码比较一下**

​

通过源码我们可以看出：

FileInputStream每次读取完一个字节，就传输一个字节，这消耗了大量的传输时间。

而 BufferedInputStream，每次读取一个字节，然后将这个字节放入缓存数组中，当读取了8kb的数据或者没有数据可读时，会一次性传输一个数组长度（即8kb）的数据，这样节省了大量的传输时间。

**因此可以得出结论，BufferedInputStream 读取数据的效率比 FileInputStream 读取数据的效率要高的多。**

既然BufferedInputStream 的工作效率要比 FileInputStream 读取数据的工作效率高的多，那么我们是不是以后再工作中要是用 BufferedInputStream来进行硬盘与内存之间的数据传输呢？

在使用 FileInputStream读取文件数据时，我们并没有一个字节一个字节的去读取文件，而是经常会自己去维护一个缓存字节数组，这个数组的大小是1024的倍数。而BufferedInputStream 内部也是维护了一个长度为 8192 字节（即8kb大小）的缓存数组。BufferedInputStream 的缓冲数组大小是固定的，而我们自己维护的缓存数组的大小是可变的，更具有灵活性。

使用FileInputStream 可以直接读取文件数据内容，而 BufferedInputStream 需要借助FileInputStream 的 read()方法先将文件内容读取到缓存数组中，再使用 BufferedInputStream 的 read() 方法读取缓存数组中的数据，通过源码我们也能看出来，BufferedInputStream 的 read() 方法在读取缓存数组中的数据时，每次读取读需要进行 if判断，这样浪费了大量的时间，理想上来说，使用缓冲数组的 FileInputStream的工作效率要高于 BufferedInputStream的工作效率。

**所以从习惯上或者理想状态来讲，个人都比较推荐使用 FileInputStream。**

如有错误处，欢迎大佬进行指正。