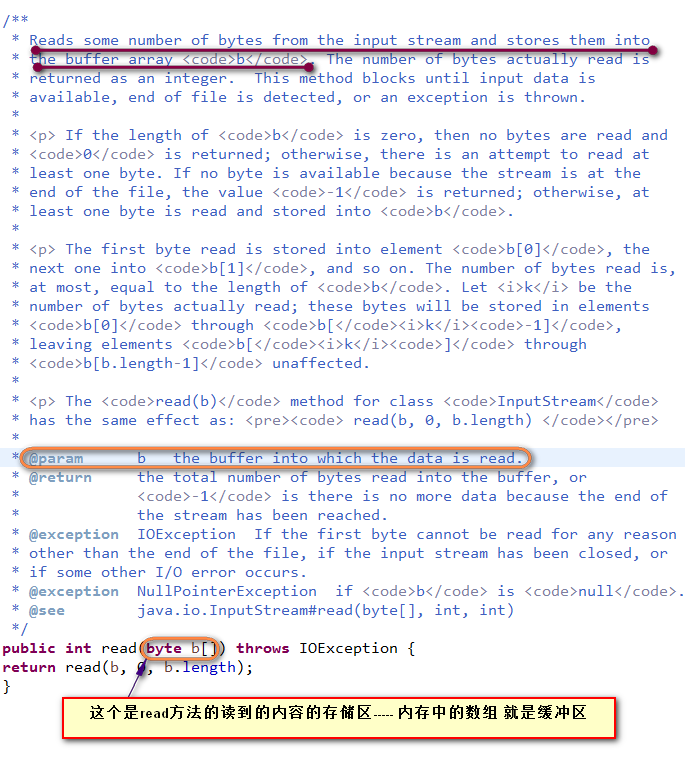
对Java IO流的新理解

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

首先看一下InputStream和OutStream的read和write方法的英文解释

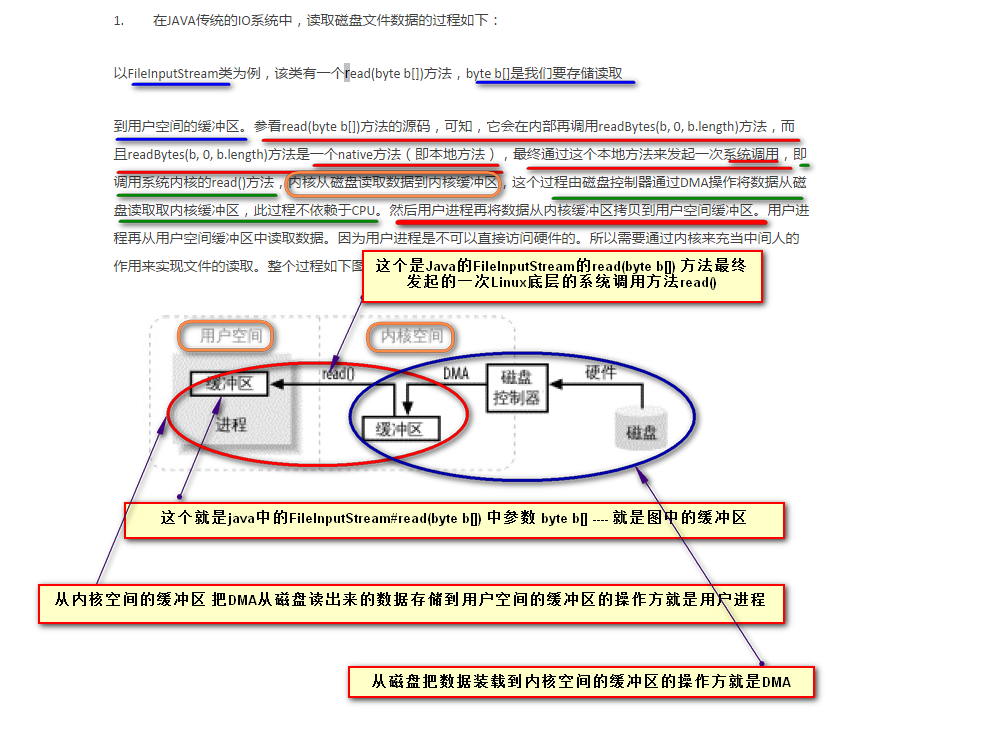
**java.io.InputStream#read(byte b[])**



其中参数byte b[] ----- the buffer into which the data is read ---- 就是读入的数据存入的位置就是这个入参的字节数组

在方法的解释：**从输入流读出一些数量的字节** 并且 **把这些读出来的字节存入缓冲的数组**

看一下Linux的读入数据的过程：



从这张图上线 可以看出 FileInputStream#read(byte[] b)总结几点：

\*\*Java的读操作属于用户进程发起的请求

\*\*读数据的请求最后转化成一次系统调用read()的发生

=====> 这个系统调用 实际上就会把数据从内核空间的缓冲区 读入InputStream流对象中------ 然后流再把数据存储到用户空间的缓冲区【暂时的理解】

\*\*磁盘数据进入到内存的最开始的操作者实际上是DMA

\*\*缓冲区之间的数据的复制过程是由用户进程 ---- 这里面就是java进程完成的

------ XXInputStream#read(byte[]) 实际上 是系统调用让InputStream流中充满了数据 ---- 可以看做是写入流的过程 ----- 最后再把数据流入到用户空间的缓冲区【暂时理解】

以前错误的理解：仅仅理解了把XX中的数据读入到了流对象中就没了 ----- 其实 把数据读入到流对象中由系统调用完成 --- 然后 流对象在用户请求的情况下，把数据再写入到目标内存区域 ======== 这个过程 可以看做数据通过流对象从A**流入**到B[内存空间] ===== 这就是为什么这个对象叫Stream 流的原因 很形象

为什么叫输入流？ ------- 数据最终是写入/存储到了内存空间 --- 内存空间已经数据智能系统的操作范围(CPU可以掌控的范围) ------- 对于CPU而言 就是输入到自己的管辖范围 所以是叫做输入

上面两段解释了为什么叫**Stream** 为什么叫**输入**

A可以是磁盘文件(FileInputStream)、控制台、内存中的数组(ByteArrayInputStream)

结论：流中的数据我们不能通过程序直接使用！必须转化成内存的标准存储单位才能使用 -------- 内存的标准存储单位就是那些常用的数据结构 --- 细化到最小就是 一个存储单元 或者 就是数组 ------ 规划到基本的单位就是 byte和byte[]

========

所以可以看出 使用输入流来读数据 ---- 实际上 ---- 数据就是通过InputStream从一端流入到另一端（形象的看）----- 从一端流入 实际上就是**对输入流进行写操作**，**但是从CPU的角度来看是读到自己的管辖范围** ===== 没有写操作，输入流怎么能从空变成有数据呢？------- 流入到另一端 实际上就是输入流主动把数据写入到内存空间 ------- CPU角度还是读 ===🡺**所以 整个过程流对象先是被输入数据（被写）-> 然后主动把数据输出去（主动写出去）**----- 整个过程对于CPU而言就是输入到自己的可以管辖的范围

===-> **这样如果持续的流入数据，可以把输入流对象看做数据流动的管道，源的数据量再大，都可以通过流对象读取完成。**

**=====》这就是流的优势：可以传输任意大小的数据【动态流动】**

**=====》那么你的目标缓冲区却是一个byte[] ----这个是开始大小固定的！====》要求我的目标存储区byte[]很有可能存储的数据是一部分而已，如果我们要把数据全部都读到内存，又不知道源数据的大小，那么这个byte[]就是一个中转站而已**，为了保证数据都能读入到内存中，这个中转站不是最终的存储目标，必须转移出去 ------ 所以 这个时候需要输出流来配合把byte[]中的数据转移到你想要转移的目标

======》**这就是为什么我的输入流操作完成之后需要使用输出流来配合**

======》这样临时存储数据的区域byte[] 就是起到一个缓冲的作用 ---- 所以 inputStream的read参数名是buffer ===== **byte[]是缓冲区的由来**

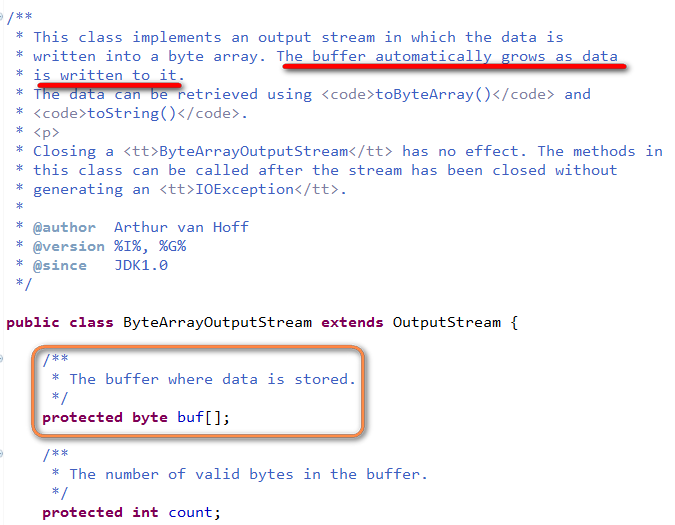
重新看张孝祥讲的数据安全的代码的理解：

重新叙述一遍需求：我的数据加密的密钥已经存储到文件中，关键是我的数据也是存储在文件中，存储加密文件的大小是未知的。

我现在想把这个加密的数据进行解密，解密方法是Cipher#doFinal(byte[]) 这个是一个字节数组

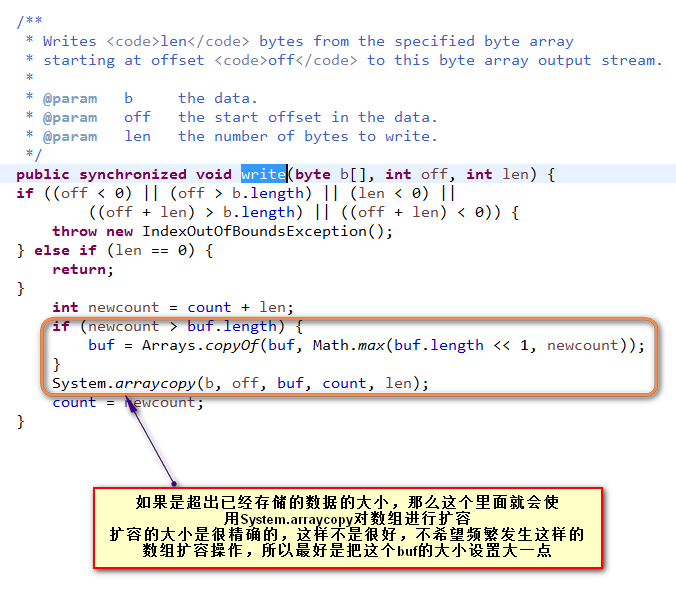
所以 很显然这个需求就是 file -> byte[] ----- 很显然 开始应该是使用FileInputStream读入数据到内存空间 byte[] ------ 但是 由于文件的大小未知 ----- 所以 刚才说了 read(byte[])这个byte[]仅仅是一个大小固定的缓冲区 ----- 所以必须把数据转移到一个符合新的文件的大小的内存byte[] ------ 这样 数据又从byte[] buffer读入数据 -> 输出 byte[] targetMem ----- 输入的目标是还是byte array ----- 所以使用的是貌似是输出流 ByteArrayOutputStream ----- 但是 如果文件任意大 这个OutputStream能承受的了么？

我们看一下 ByteArrayOutputStream这个类的源代码



这个类的介绍说了 ByteArrayOutputStream 这个类实现了一个输出流 这个输出流里面的数据被写入一个字节数组buffer。这个字节数组buffer可以随着写入里面的数据增长而**自动增大**。

那么 我们看一下写入的方法



所以 如果文件太大啊 这个方法不太合适 因为会不断申请大的内存空间 如果内存空间溢出，就麻烦了。

<http://www.open-open.com/lib/view/open1413518521372.html> ---- 这篇文章很好