在文件的处理中使用了BufferedReader时提到了一个特征---当BufferedReader读取输入流的数据时，如果没有读取到有效数据，程序将在此处阻塞该线程的执行（使用InputStream的read方法从流中读取数据时，如果数据源中没有数据，它也会阻塞该线程），也就是说，前面介绍的输入流、输出流都是阻塞式的输入与输出。不仅如此，传统的输入流和输出流都是通过字节的移动来处理的（即使我们不直接去处理字节流，但底层的实现还是依赖于字节处理），也就是说，面向流的输入/输出系统依次只能处理一个字节，因此效率不高。

从jdk 1.4开始，java气功了一系列改进的新功能，这些功能被统称为新IO（New IO，简称NIO），新的IO采用了内存映射文件的方式来处理输入/输出，新IO将文件或文件的一段区域映射到内存中，这样就可以像访问内存一样来访问文件了（这种操作模拟了操作系统上的虚拟内存的概念），通过这种方式来进行输入/输出比传统的输入/输出要快得多

Channel（通道）和Buffer（缓冲）是新IO中的两个核心概念，Channel是对传统的输入/输出系统的模拟，在新IO系统中所有的数据都需要通过通道传输：Channel与传统的InputStream、OutputStream最大的区别在于它提供了一个map方法，通过该方法可以直接将“一块数据”映射到内存中，如果说传统的输入/输出系统是面向流的处理，则新IO则是面向块的处理。

Buffer可以被理解成一个容器，它的本质是一个数组，发送到Channel中的所有对象都必须首先放到Buffer中，而从Channel中读取的数据也必须先方位Buffer中。除了Channel和Buffer之外，新IO还提供了用于将Unicode字符串映射成字节序列以及逆映射操作的Charset类，也提供了用于支持非阻塞式输入/输出的Selector类。

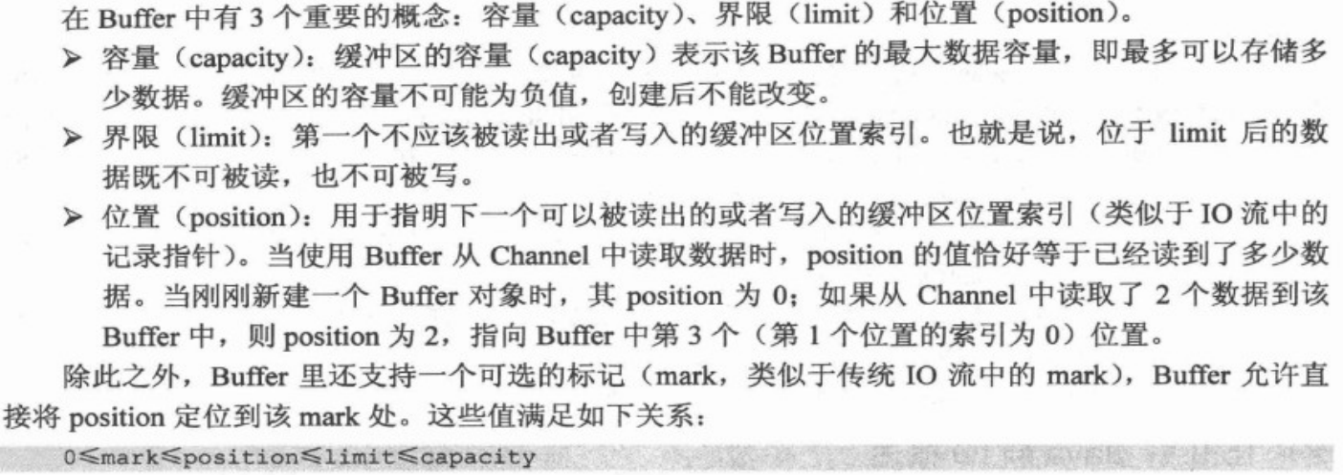
**使用Buffer**

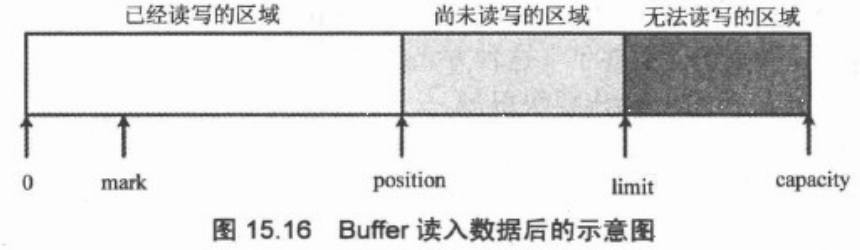
从内部结构上看，Buffer就像一个数组，它可以保存多个类型相同的数据，Buffer是一个抽象类，其最常用的子类是ByteBuffer，它可以在底层字节数组上进行get/set操作，除了ByteBuffer之外，对应于其他基本数据类型（boolean除外）都有相应的Buffer类：CharBuffer、ShortBuffer、IntBuffer、LongBuffer、FloatBuffer、DoubleBuffer等

上面的这些Buffer类都没有构造器，可以通过武侠方法来得到一个Buffer对象：

Static XxxBuffer allocate(int capacity)

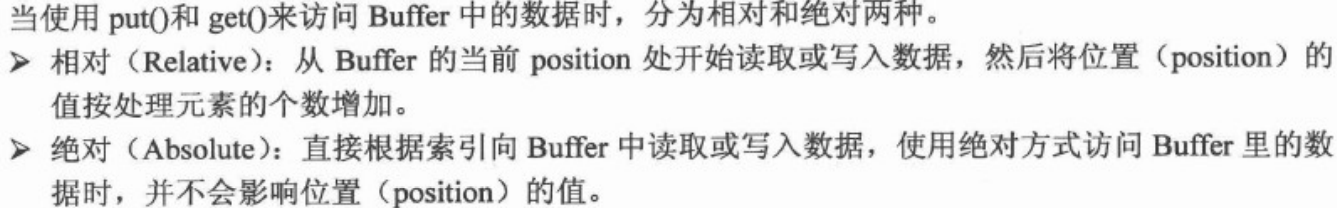
实际使用较多的是ByteBuffer和CharBuffer，其他Buffer子类则较少使用，其中ByteBuffer类还有一个子类：MappedByteBuffer，它用于表示Channel将磁盘文件的部分或全部内容映射到内存中后得到的结果，通常MappedByteBuffer对象由Channel的map方法返回。





初始化时，Buffer的position为0，limit为capacity，程序可通过put方法向Buffer中放入一些数据（或者从Channel中获取一些数据），每放入一些数据，Buffer的position相应地向后移动一些位置

当Buffer装入数据结束后，调用Buffer的flip方法，该方法将limit设置为position所在的位置，并将position设为0，这就使得Buffer的读写指针又移到了开始位置。也就是说，Buffer调用flip方法之后，Buffer为输出数据做好准备；当Buffer输出数据结束之后，Buffer调用clear方法，clear方法不是清空Buffer的数据，它仅仅将position置为0，将limit置为capacity，这样为再次向Buffer中装入数据做好准备，对Buffer执行clear方法后，该Buffer对象里的数据仍然存在。



**public** **static** **void** testBuffer(){

// 创建Buffer

CharBuffer charBuffer = CharBuffer.*allocate*(8);

System.*out*.println("capacity = " + charBuffer.capacity());

System.*out*.println("limit = " + charBuffer.limit());

System.*out*.println("position = " + charBuffer.position());

// 放入元素

charBuffer.put('a');

charBuffer.put('b');

charBuffer.put('c');

System.*out*.println("加入三个元素后 position = " + charBuffer.position());

// 调用flip

charBuffer.flip();

System.*out*.println("执行flip后 limit = " + charBuffer.limit());

System.*out*.println("position = " + charBuffer.position());

// 取出第一个元素

charBuffer.get();

System.*out*.println("取出第一个元素后 limit = " + charBuffer.limit());

System.*out*.println("position = " + charBuffer.position());

// 调用clear

charBuffer.clear();

System.*out*.println("调用clear后 capacity = " + charBuffer.capacity());

System.*out*.println("limit = " + charBuffer.limit());

System.*out*.println("position = " + charBuffer.position());

// 再取出第一个元素

**char** ch1 = charBuffer.get();

**char** ch2 = charBuffer.get();

**char** ch3 = charBuffer.get();

**char** ch4 = charBuffer.get();

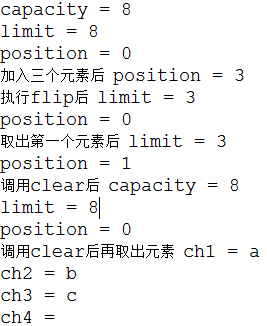
System.*out*.println("调用clear后再取出元素 ch1 = " + ch1);

System.*out*.println("ch2 = " + ch2);

System.*out*.println("ch3 = " + ch3);

System.*out*.println("ch4 = " + ch4);

}



**使用Channel**

Channel类似于传统的流对象，但与传统的刘对象有两个主要的区别。

1. Channel可以直接将指定文件的部分或全部直接映射成Buffer
2. 程序不能直接访问Channel中的数据，包括读取、写入都不行，Channel只能与Buffer进行交互，也就是说，如果要从Channel中取得数据，必须先用Buffer从Channel中读取一些数据，然后让程序从Buffer取出这些数据；如果要将程序中的数据写入Channel，一样先让程序将数据放入Buffer中，程序再将Buffer里的数据写入Channel中。

Java为Channel接口提供了DatagramChannel、FileChannel、Pipe.SinkChannel、Pipe.SourceChannel、SelectableChannel、ServerSocketChannel、SocketChannel等实现类。

所有的Channel都不应该通过构造器直接创建，而是通过传统的节点InputStream、OutputStream的getChannel方法来返回对应的Channel，不同的节点流获得得Channel不一样，

Channel中最常用的3类方法是map、read和write，其中map方法用于将Channel对应的部分或全部数据映射成ByteBuffer；而read或write方法都有一系列的重载形式，这些方法用于从Buffer中读取数据或向Buffer中写入数据

Map方法的方法签名为：MappedByteBuffer map(FileChannel.MapMode mode, long position, long size)，第一个参数执行映射时的模式，分别只有只读、读写等模式；而第二个、第三个参数用于控制将Channel的哪些数据映射成ByteBuffer

**public** **static** **void** testChannel(){

File file = **new** File("F:\\workspacing-coc\\Java\\src\\com\\test\\test\\Main.java");

**try**{

FileChannel inChannel = **new** FileInputStream(file).getChannel();

FileChannel outChannel = **new** FileOutputStream("copy.txt").getChannel();

// 将FileChannel中的全部数据映射成ByteBuffer

MappedByteBuffer buffer = inChannel.map(FileChannel.MapMode.*READ\_ONLY*, 0, file.length());

// 使用GBK的字符集来创建解码器

Charset charset = Charset.*forName*("utf-8");

// 直接将buffer中的数据全部输出

outChannel.write(buffer);

// 再次调用buffer的clear方法，恢复limit与position的值

buffer.clear();

// 创建解码器对象

CharsetDecoder charsetDecoder = charset.newDecoder();

// 使用解码器将byteBuffer转换为charBuffer

CharBuffer charBuffer = charsetDecoder.decode(buffer);

System.*out*.println(charBuffer);

}**catch**(FileNotFoundException e){

System.*out*.println("File not found!");

e.printStackTrace();

}**catch**(NonWritableChannelException e){

e.printStackTrace();

}**catch**(Exception e){

e.printStackTrace();

}

}

如果习惯了传统IO的“用竹筒多次重复取水”的过程，或者担心Channel对应的文件过大，使用map方法一次将所有的文件内容映射到内存中引起的性能下降，也可以使用Channel和Buffer传统的“用竹筒多次重复取水”的方式：

FileChannel inChannel = **new** FileInputStream(file).getChannel();

ByteBuffer buffer = ByteBuffer.*allocate*(1024);

**while**(inChannel.read(buffer) != -1){

buffer.flip(); // 锁定buffer

Charset charset = Charset.*forName*("utf-8");

CharsetDecoder charsetDecoder = charset.newDecoder();

CharBuffer charBuffer = charsetDecoder.decode(buffer);

System.*out*.print(charBuffer);

buffer.clear();

}

上面代码虽然使用了FileChannel和Buffer来读取文件，但处理方式和使用InputStream、byte[]来读取文件的方式几乎一样。

**字符集和Chatset**

计算机中的文件、数据、图片文件只是一种表面现象，所有文件在底层都是二进制文件，及全部都是字节码。



Jdk 1.4提供了Charset来处理自己序列和字符序列（字符串）之间的转换关系，该类包含了用于创建解码器和编码器的方法，还提供了获取Charset所支持字符集的方法，Charset类是不可变类。

获取全部可支持的字符集：

**public** **static** **void** testCharset(){

SortedMap<String, Charset> map = Charset.*availableCharsets*();

Set<String> set = map.keySet();

Iterator<String> iterator = set.iterator();

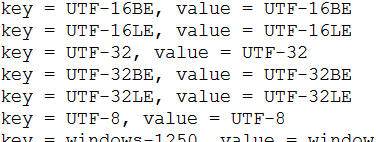
**while**(iterator.hasNext()){

String key = iterator.next();

System.*out*.println("key = " + key + ", value = " + map.get(key));

}

}



一旦知道了字符集的别名之后，我们就可以调用Charset的forName方法来创建对应的Charset对象，forName方法的参数就是相应字符集的别名。

获得Charset对象之后，就可以通过该对象的newDecoder、newEncoder这两个方法分别返回CharsetDecoder和CharsetEncoder对象，代表该Charset的解码器和编码器。调用CharsetDecoder的decode方法就可以将ByteBuffer转换为CharBuffer，调用CharsetEncoder的encoder方法就可以将CharBuffer或String转换成ByteBuffer