Java NIO 原理概述

1. 什么是NIO？

NIO的全称是 Non-blocking IO，就是非阻塞的IO。NIO和传统的IO有很大的区别，传统

的IO是一种面向流的阻塞式IO，在处理速度上是比较慢的。传统的IO是一个字节一个字节的处理，处理速度很慢。

NIO 处理数据是以块为单位，并且NIO是非阻塞的，处理数据的速度是比较快的。

在NIO中需要掌握如下介个概念：缓冲区（buffer）、通道（channel）、选择器（selector）

1. 缓存区Buffer

缓冲区是一个指定类型的数据容器，实质是一个数组。在NIO库中，所有数据都是用缓

冲区来处理的。在读数据的时候，也是将数据读入到缓冲区中；写数据时也是将数据写入到缓冲区中。注意：任何时候访问NIO中的数据都是先放入到缓冲区中。传统的IO是直接将数据放入到流中。

在NIO中，所有的缓冲区都是继承Buffer类；最常用的是ByteBuffer，所有的基本类型的都有对应的缓冲区。

1. 缓冲区的属性
2. position :用于记录下一个元素的位置，调用buffer的get或put方法时，都会改变position的值。（read/write方法同样也会），初始化为0；
3. limit ：读或写数据时，用于记录当前position的位置，也就是读或写数据的界限。初始化值为capacity的值。
4. capacity ：当前缓冲区的容量。固定不变的。

以上三个属性具有如下的关系：position <= limit <= capacity

4）如果我们创建一个新的容量大小为10的ByteBuffer对象，在初始化的时候，position设置为0，limit和 capacity被设置为10，在以后使用ByteBuffer对象过程中，capacity的值不会再发生变化，而其它两个个将会随着使用而变化。四个属性值分别如图所示：

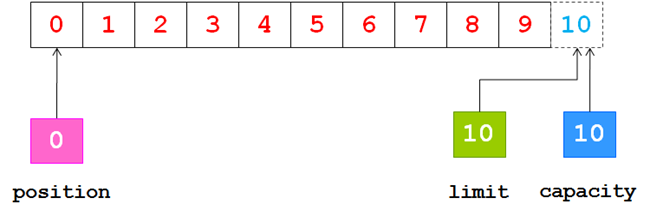


图1 初始时

此时若通过通道读取4个字节的数据到buffer中，相当于是往buffer中写数据。此时position的值为4，limit和capacity值不变。三个属性的位置如图2所示。

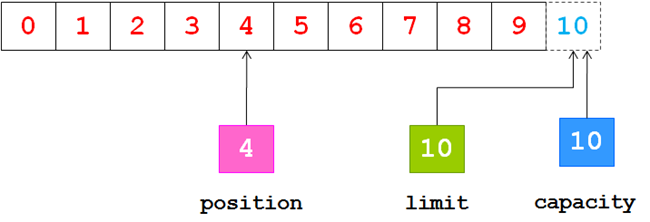


图2 buffer写数据时

假设现在要从buffer中写数据到通道时，首先应该调用buffer的flip()方法，该方法会做如下的几步操作：

1. 将limit的值设置当前的position的值
2. 将position设置为0

以上两步操作都是为读取buffer中的值做准备的。调用flip()方法后三个属性的位置变化如图3所示。

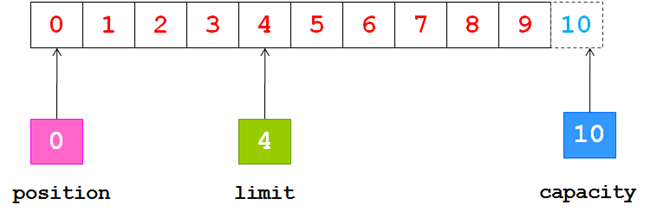


图3 调用flip()方法后

现在通道调用write(buffer)方法时，通过调用buffer的get()方法读取buffer中的数据，调用get()方法position增加，但不会操作limit的值，读取数据完毕后position就等于limit的值。此时三个属性的位置变化如下图所示：

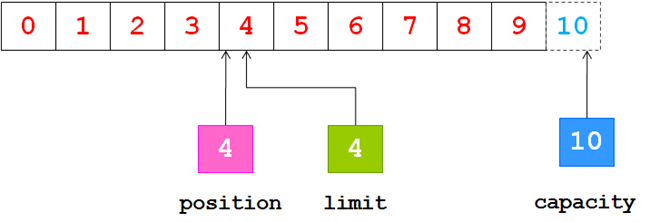


图4 通道读取buffer数据后

再假设此时又需要往该buffer中写入数据时，此时应该调用clear()方法重置buffer缓冲区使用position = 0、limit=capacity。此时三个属性的位置变化如图5所示。

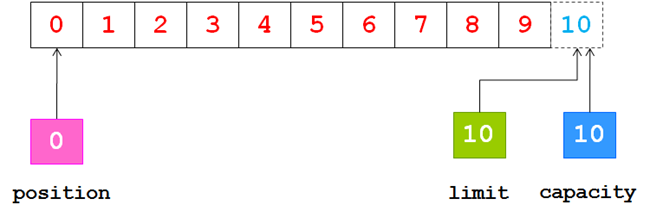


图5 调用clear()方法后

1. 缓冲区API介绍

在缓冲区中，具有如下的常用操作的API：

1. clear()：该方法作用是重置缓存区，恢复到最初的状态，注意：在往缓冲区中写数据时，通常先调用该方法重置缓存区，这里重置主要是对position、limit重置
2. flip()：该方法是作用是，为读取到的数据定界限，将缓存区的limit的值设置为当前position的值，然后将position设置为0。在读取缓冲区的数据之前，须调用该方法，已确定那些数据是刚才读取的。
3. get()：从buffer缓冲区中，读取一个数据。导致position的值增加1
4. put()：往buffer缓冲区中写入一个数据，同样也会导致position的值递增
5. remaining():返回position和limit之间元素的数量。
6. hasRemaining()：判断position和limit之间是否还有可取的元素。