Nio缓冲区分析二

本文主要讲解缓冲区的分配、缓冲区分片和只读缓冲区。由于ByteBuffer是经常使用的缓冲区，所有使用ByteBuffer为例。

1. **缓冲区的分配（也是创建buffer的方法）**

1）调用ByteBuffer的静态方法allocate(capacity)方法，去分配一个指定容量的ByteBuffer缓冲区，该缓冲区的元素的值都为0。实质是调用ByteBuffer的构造函数来分配的。

/\*\*

\* 缓冲区的分配

\* **@author** 21463

\*

\*/

**public** **class** BufferAllocate {

**public** **static** **void** main(String[] args) {

//1.分配一个容量为10 的缓冲区，实质指定了数组的大小为10

ByteBuffer buffer = ByteBuffer.*allocate*(10);

}

}

2）调用ByteBuffer的静态方法wrap(array)方法，通过一个byte数组来初始化ByteBuffer缓冲区。

/\*\*

\* 缓冲区的分配

\* **@author** 21463

\*

\*/

**public** **class** BufferAllocate {

**public** **static** **void** main(String[] args) {

//1.分配一个容量为10 的缓冲区，实质指定了数组的大小为10

ByteBuffer buffer = ByteBuffer.*allocate*(10);

//2.通过调用wrap方法来，使用一个byte数组来分配一个ByteBuffer缓冲区

//并使用数组来初始化缓冲区。

**byte**[] array = **new** **byte**[10];

**for**(**byte** i = 0;i < array.length ; i++){

array[i] = i;

}

ByteBuffer buffer2 = ByteBuffer.*wrap*(array);

ByteBuffer buffer3 = ByteBuffer.*wrap*(array, 1, 5);

}

}

1. **缓冲区的分片（实质就是获取buffer的子缓冲区，子缓冲区和父缓冲区共享数据）**

缓冲区的分片是调用ByteBuffer的slice()方法来实现的，调用该方法会返回buffer的一

个子buffer，并且改子缓冲区中数据的修改会体现到buffer中，说明slice()方法创建的子buffer和buffer是共享数据的。

slice()方法创建子buffer，实质是使用buffer中的hb即缓冲数据所有的数组的引用去创建的子buffer，因此buffer和其子buffer是共享数据的，更改子buffer中的数据可以体现到buffer中。ByteBuffer的slice()方法创建子ByteBuffer的代码如下：

**public** ByteBuffer slice() {

**return** **new** HeapByteBuffer(hb,

-1,

0,

**this**.remaining(),

**this**.remaining(),

**this**.position() + offset);

}

其中hb就是ByteBuffer实例中缓冲数据所用的数组，remaining()方法返回的是position和limit之间的元素个数，在这里就是子buffer的容量和limit的值。

缓冲区简单分片例子如下：

/\*\*

\* 缓冲区的分片

\* **@author** 21463

\*

\*/

**public** **class** BufferSlience {

**public** **static** **void** main(String[] args) {

//分配缓冲区

ByteBuffer buffer = ByteBuffer.*allocate*(10);

//向缓冲区中写入1-10

buffer.clear();//重置缓冲区，一般在写入数据前调用该方法

**for**(**byte** i = 1;i <= 10 ;i++){

buffer.put(i);

}

buffer.flip();//固定缓冲数据区域

//position()是返回当前缓冲区的position值

//将position的值定位到newPosition的位置

buffer.position(3);

//设置limit的值定位到newLimit的位置。

buffer.limit(7);

//获取子缓冲区

ByteBuffer slice = buffer.slice();

**for**(**byte** i = 0;i < slice.capacity();i ++){

**byte** b = slice.get(i);//通过索引获取buffer中的元素值

b \*= 10;

//将数据写入到指定的位置

slice.put(i, b);//通过元素索引，设置元素的值

}

buffer.clear();//重置将position 设置为0，limit设置为capacity

**while**(buffer.hasRemaining()){

System.***out***.println(buffer.get());

}

}

}