# 网络编程-NIO

https://mp.weixin.qq.com/s/fg36xDx--otVPACj391X0w

## Reactor 模式

Reactor 模式称为叫反应堆或则反应器。大量的线程休眠导致资源浪费，在 BIO 中，通过 Reactor 来优化，下面举个简单的例子：

有个高级游戏乐园，里面有 5 个不同游戏的玩法，玩家进来的时候，有一对一的工作人员给玩家讲解每个玩法。

### 传统 BIO 模式：

A 玩家进来，工作人员 1 接待 A，工作人员 1 讲解完第一个游戏玩法，然后A玩家开始玩，此时，工作人员 1 就在等待 A 玩家玩完（比如 br.readLine()）。玩家玩第二个游戏的时候，工作人员 1 开始讲解第二个游戏玩法，然后 A 玩家继续玩，一直玩到第五个游戏。

如果还有玩家进来呢？那派出第二个工作人员，第三个工作人员。。。

我们可以看到，工作人员在玩家玩的时候，他是处于休息空闲的状态，而且玩家越来越多的时候，工作人员就需要越来越多。

### Reactor 模式：

工作人员 1 给玩家 A 讲解完了，他就去处理其他事情，而玩家开始玩游戏，当玩家准备玩下一个游戏的时候，他就呼叫工作人员，工作人员就过来讲解下一个游戏玩法。本来一个工作人员只能服务一个玩家，但是通过这个模式，他可以同时服务多个玩家，在玩家 A 玩游戏的时候，他可以为其他玩家提供讲解服务。

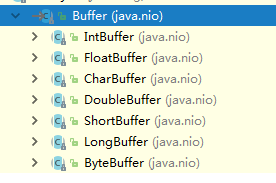
**在 Reactor 模式中，应用程序并不会调用某个方法直至完成，而是设置了事件处理流程，具体事件处理程序向反应器注册一个事件处理器，等到事件来了，具体事件处理程序再处理相关事件。**

## NIO 三大组件

### 1. Buffer

Buffer 在 NIO 中，本质是一块内存，用于和 NIO 通道进行交互。我们可以把数据从通道读取中出来，写入到 Buffer，也可以把 Buffer 的数据读到出来，写到通道中。

在 NIO 中，java 定义了 IntBuffer、FloatBuffer、ByteBuffer 等，我们比较常用的是 ByteBuffer。

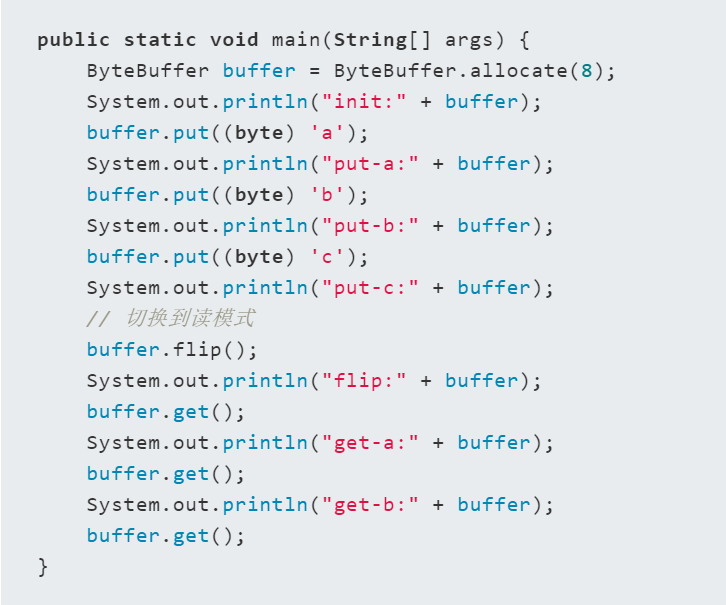


#### 主要属性

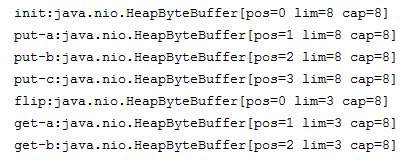
Buffer 有几个重要的属性：**position、limit、capacity**。

* capacity：指的是内存块的固定大小，一旦设定，就不能再修改。往内存写满数据后，就不能再写，除非将其清空。
* position：指的是读和写的下一个位置。每次读或写的时候，就会加 1。从写模式切换到读模式的时候（flip 方法），position 就会重置为 0。
* limit：在写模式下，不超过 capacity，在读模式下，position 不能大于 limit。意思其实很简单，就是写的时候，不能超过指定的 capacity 大小。切换到读的时候，limit 等于写的长度，读取的 position 不能超过写的数据 limit。

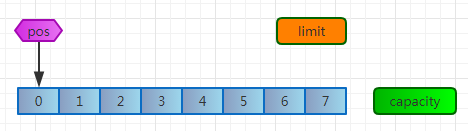
下面通过一个简单的例子深入了解一下这几个属性。



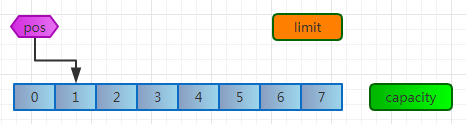
**输出结果如下：**

****

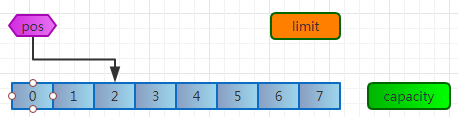
初始化时，pos 指向 0，capacity 和 limit 都等于指定大小 8。



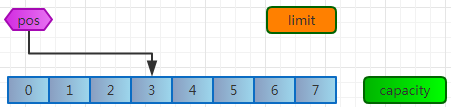
put-a 时，pos+1，等于 1，capacity 和 limit 不变。



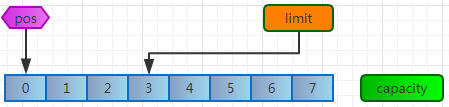
put-b 时，pos+1，等于 2，capacity 和 limit 不变。



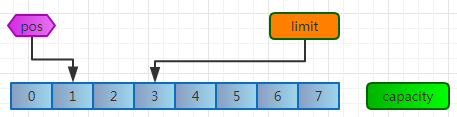
put-c 时，pos+1，等于 3，capacity 和 limit 不变。



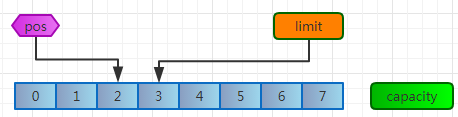
flip 后，pos 把值赋值给 limit，并重置为 0,capacity 不变。此时，pos 等于 0，limit 等于 3，capacity 等于 8。



get-a 时，pos+1，等于 1，capacity 和 limit 不变。



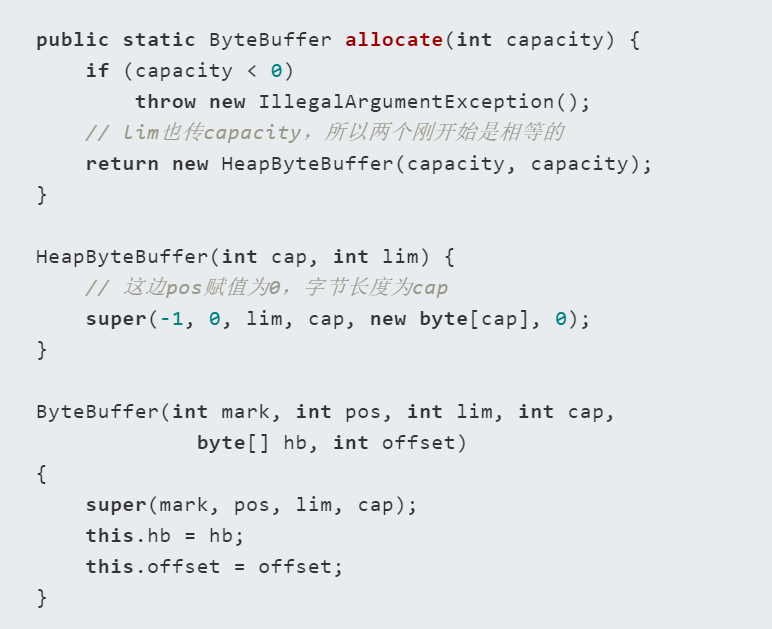
get-b 时，pos+1，等于 1，capacity 和 limit 不变。



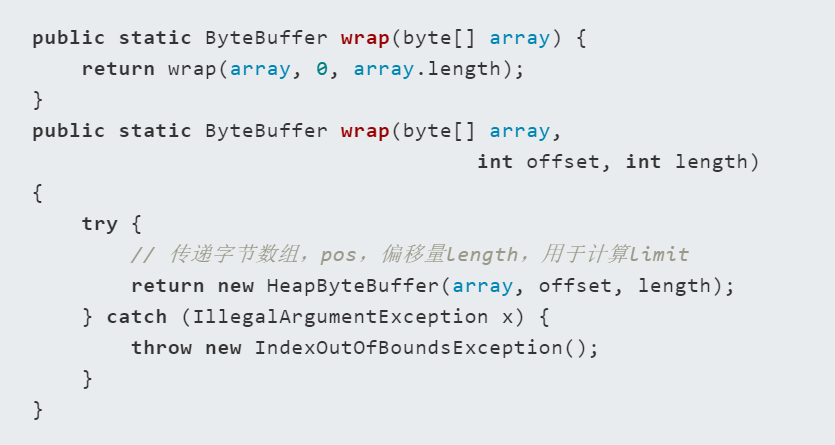
#### 主要方法

##### 初始化 buffer

**allocate 方法**，在上面例子中，我们看到了 ByteBuffer.allocate(8) 的使用。

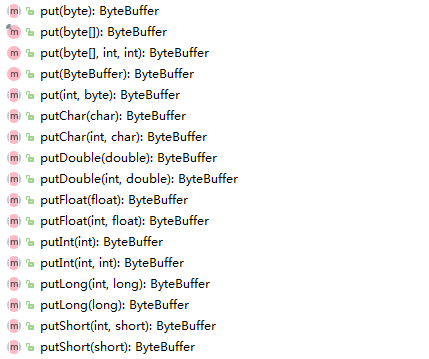


**wrap 方法**，跟 allocate 方法都可以初始化 buffer，不同的是可以指定 pos 和 limit，以及指定字节数组的初始值。

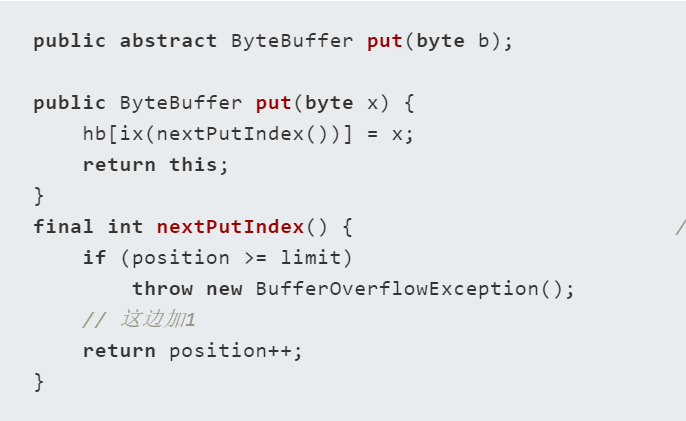


##### 2. 写数据

除了上面例子演示的，put(byte)，还有以下这些:



从源码中看 pos 会加 1 的原因：



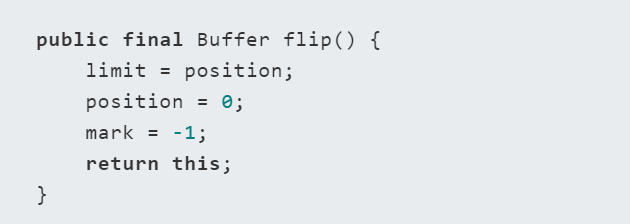
也可以把通道中的数据写入到 buffer：

// 这边用read指的是把通道的数据读取出来，再写入buffer，read返回的是写入buffer的数据大小。

channel.read(buf)

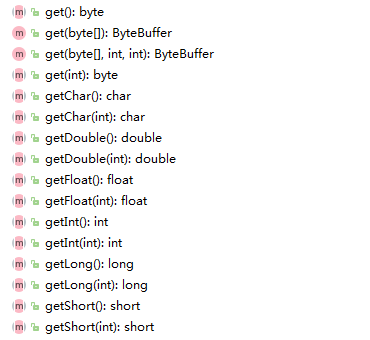
##### 3. flip(读写切换)

从源码中也可以看出，把 pos 的值赋值给 limit，并重置为 0。

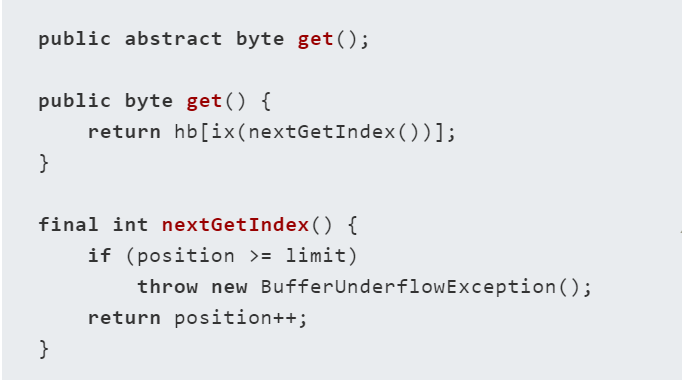


##### 读数据

除了上面例子演示的，get，还有以下这些：



从源码中看 pos 会加 1 的原因：

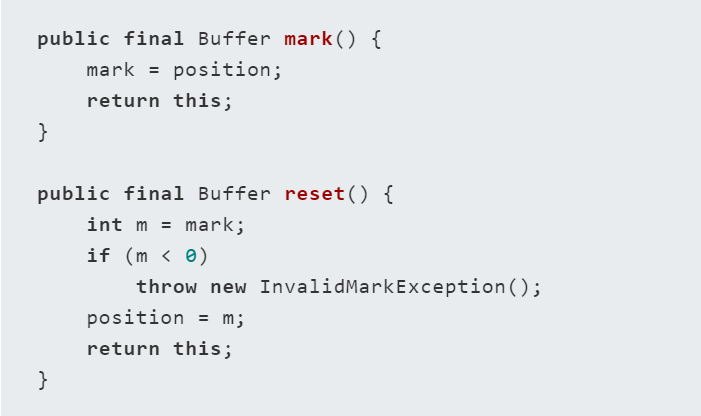


也可以把buffer的数据写入到通道中：



##### 5. 标记与重置

调用 mark 的时候，会把 pos 的值给 mark，调用 reset 的时候，会把 mark 的值给 pos。在实际过程中，我们在读操作的时候，先调用 mark 方法标记位置，比如此时为 4，当我们读到 7 的时候，再调用 reset 方法，此时又重新从 4 开始读。

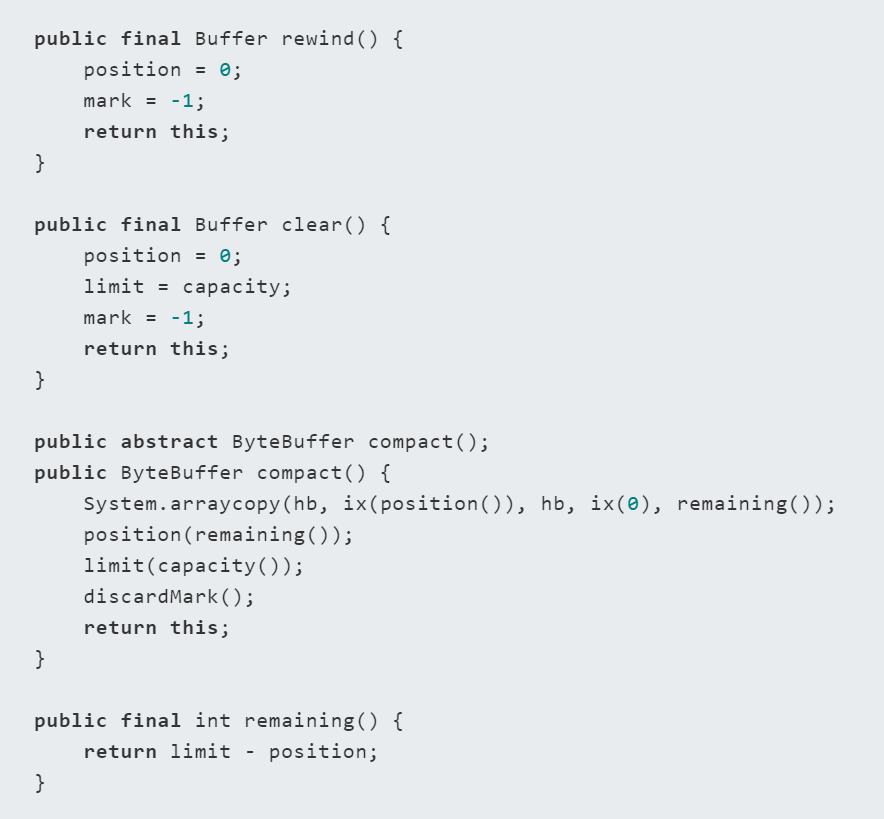


##### 6、rewind、clear、compact

从源码可以看出，rewind 把 pos 置为 0，所以就是从头开始读写。

clear 方法，把 pos 置 0，并重置 limit 为 capacity，这个时候进行写的时候，就是从第一个位置开始写，如果原先有数据，就是要被覆盖，相当于清空了整个内存。

compact 与 clear 不一样的是，他会把 pos 和 limit 之间的数据，移到前面去，并设置 pos 的值，写的时候，会从新的位置开始写。比如 pos 为 2，limit 为 4，他会把 2-4 之间的值移到 0，再把 pos 设置为 2，这样没读的数据，就不会被覆盖而消失消失。



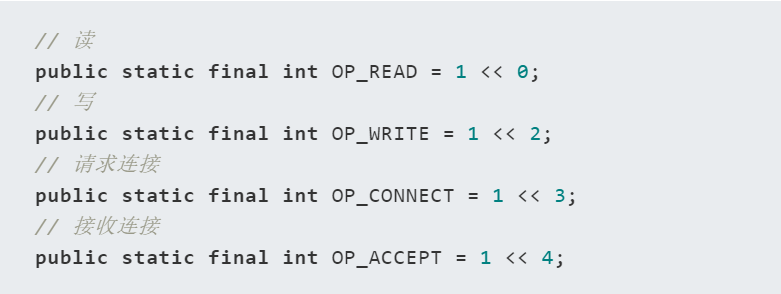
### Channel

Channel，通道，操作系统和应用程序之间的数据交互，就是通过通道来的。

* ServerSocketChannel：用于 TCP 的服务端
* SocketChannel：用于 TCP 的客户端
* DatagramChannel：用于 UDP

### 3. Selector

选择器，把 Channel 和需要的事件注册到 Selector 上面，让 Selector 进行监听。这些事件包括以下几种：



当需要监听多个事件时，比如 OP\_ACCEPT 和 OP\_CONNECT 可以这样写：

c174d7b8dcb28972ebebbf34b0ddc057