

Java NIO 之 Channel 和 Buffer



六尺帐篷 (/u/f8e9b1c246f1)

2017.08.01 16:20 字数 3085 阅读 237 评论 1 喜欢 21

(/u/f8e9b1c246f1)

[编辑文章 \(/writer#/notebooks/14906121/notes/15267891\)](#)

- Channel
- Channel Characteristics
- Java NIO Channel Classes
- buffer
- 什么是缓冲区?
- 缓冲区类型
- 缓冲区内部细节
- NIO Buffer Characteristics
- How to Read from NIO Buffer
- How to Write to NIO Buffer
- Java NIO 读写文件实例程序

Channel

Java NIO中，channel用于数据的传输。类似于传统IO中的流的概念。channel的两端是buffer和一个entity，不同于IO中的流，channel是双向的，既可以写入，也可以读取。而流则是单向的，所以channel更加灵活。我们在读取数据或者写入数据的时候，都必须经过channel和buffer，也就是说，我们在读取数据的时候，先利用channel将IO设备中的数据读取到buffer，然后从buffer中读取，我们在写入数据的时候，先将数据写入到buffer，然后buffer中的数据再通过channel传到IO设备中。

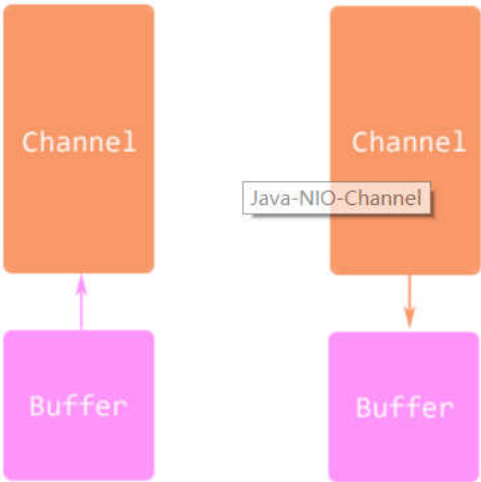
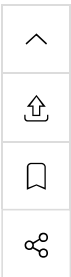


image.png

我们知道NIO的特点就是将IO操作更加类似于底层IO的流程。
我们可以通过底层IO的机制更好的理解channel。



所有的系统I/O都分为两个阶段：等待就绪和操作。

- 等待就绪就是从IO设备将数据读取到内核中的过程。
- 操作就是将数据从内核复制到进程缓冲区的过程。

channel就可以看作是IO设备和内核区域的一个桥梁，凡是与IO设备交互都必须通过channel，而buffer就可以看作是内核缓冲区。这样整个过程就很好理解了。

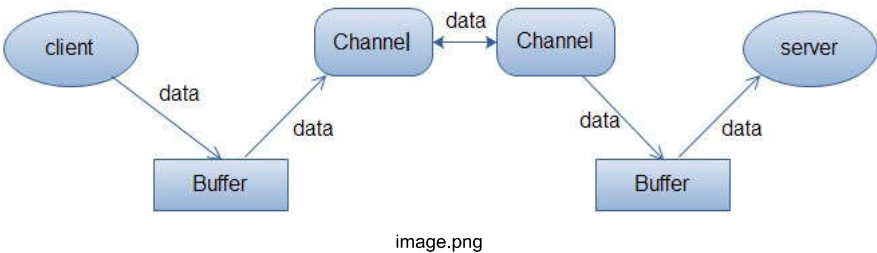
我们看一下读取的过程

先从IO设备，网卡或者磁盘将内容读取到内核中，对应于NIO就是从网卡或磁盘利用channel将数据读到buffer中

然后就是内核中的数据复制到进程缓冲区，对应于就是从buffer中读取数据

写入的过程则是：

先从进程将数据写到内核中，对应于就是进程将数据写入到buffer中，然后内核中的数据再写入到网卡或者磁盘中，对应于就是，buffer中的数据利用channel传输到IO设备中。



以上其实就是NIO基本的利用channel和buffer进行读取和写入的流程。

Channel Characteristics

- 与传统IO中的流不同，channel是双向的，可读可写
- channel从buffer中读取数据，写入数据也是先写入到buffer
- channel可以实现异步读写操作
- channel可以设置为阻塞和非阻塞的模式
- 非阻塞模式意味着，当读不到数据或者缓冲区已满无法写入的时候，不会把线程睡眠
- 只有socket的channel可以设置为非阻塞模式，文件的channel是无法设置的。文件的IO一定是阻塞的
- 如果是文件channel的话，channel可以在channel之间传输数据

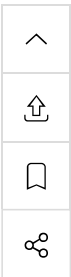
Java NIO Channel Classes

channel主要有两大类，四个具体的类

- FileChannel
文件的读写是不可以设置为非阻塞模式
- SocketChannel
根据tcp和udp，服务端和客户端，又可以分为，SocketChannel，ServerSocketChannel and DatagramChannel.它们是可以设置为非阻塞模式的

buffer

什么是缓冲区？



Buffer 是一个对象，它包含一些要写入或者刚读出的数据。在 NIO 中加入 Buffer 对象，体现了新库与原 I/O 的一个重要区别。在面向流的 I/O 中，您将数据直接写入或者将数据直接读到 Stream 对象中。

在 NIO 库中，所有数据都是用缓冲区处理的。在读取数据时，它是直接读到缓冲区中的。在写入数据时，它是写入到缓冲区中的。任何时候访问 NIO 中的数据，您都是将它放到缓冲区中。

缓冲区实质上是一个数组。通常它是一个字节数组，但是也可以使用其他种类的数组。但是一个缓冲区不 仅仅 是一个数组。缓冲区提供了对数据的结构化访问，而且还可以跟踪系统的读/写进程。

缓冲区类型

最常用的缓冲区类型是 ByteBuffer。一个 ByteBuffer 可以在其底层字节数组上进行 get/set 操作(即字节的获取和设置)。

ByteBuffer 不是 NIO 中唯一的缓冲区类型。事实上，对于每一种基本 Java 类型都有一种缓冲区类型：

- ByteBuffer
- CharBuffer
- ShortBuffer
- IntBuffer
- LongBuffer
- FloatBuffer
- DoubleBuffer

每一个 Buffer 类都是 Buffer 接口的一个实例。除了 ByteBuffer，每一个 Buffer 类都有完全一样的操作，只是它们所处理的数据类型不一样。因为大多数标准 I/O 操作都使用 ByteBuffer，所以它具有所有共享的缓冲区操作以及一些特有的操作。

缓冲区内部细节

本节将介绍 NIO 中两个重要的缓冲区组件：状态变量和访问方法 (accessor)。

状态变量是前一节中提到的"内部统计机制"的关键。每一个读/写操作都会改变缓冲区的状态。通过记录和跟踪这些变化，缓冲区就可能内部地管理自己的资源。

在从通道读取数据时，数据被放入到缓冲区。在有些情况下，可以将这个缓冲区直接写入另一个通道，但是在一般情况下，您还需要查看数据。这是使用 访问方法 get() 来完成的。同样，如果要将原始数据放入缓冲区中，就要使用访问方法 put()。

状态变量

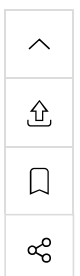
可以用三个值指定缓冲区在任意时刻的状态：position, limit, capacity

这三个变量一起可以跟踪缓冲区的状态和它所包含的数据。我们将在下面的小节中详细分析每一个变量，还要介绍它们如何适应典型的读/写(输入/输出)进程。在这个例子中，我们假定要将数据从一个输入通道拷贝到一个输出通道。

Position

您可以回想一下，缓冲区实际上就是美化了的数组。在从通道读取时，您将所读取的数据放到底层的数组中。position 变量跟踪已经写了多少数据。更准确地说，它指定了下一个字节将放到数组的哪一个元素中。因此，如果您从通道中读三个字节到缓冲区中，那么缓冲区的 position 将会设置为3，指向数组中第四个元素。

同样，在写入通道时，您是从缓冲区中获取数据。position 值跟踪从缓冲区中获取了多少数据。更准确地说，它指定下一个字节来自数组的哪一个元素。因此如果从缓冲区写了5个字节到通道中，那么缓冲区的 position 将被设置为5，指向数组的第六个元素。



Limit

limit 变量表明还有多少数据需要取出(在从缓冲区写入通道时)，或者还有多少空间可以放入数据(在从通道读入缓冲区时)。
position 总是小于或者等于 limit。

Capacity

缓冲区的 capacity 表明可以储存在缓冲区中的最大数据容量。实际上，它指定了底层数组的大小 — 或者至少是指定了准许我们使用的底层数组的容量。
limit 决不能大于 capacity。

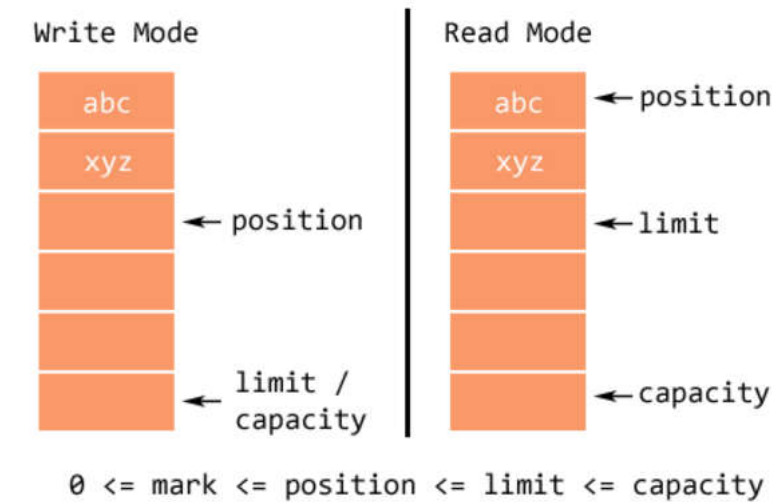


image.png

实例：
我们首先观察一个新创建的缓冲区。出于本例子的需要，我们假设这个缓冲区的 总容量为8个字节。Buffer 的状态如下所示：

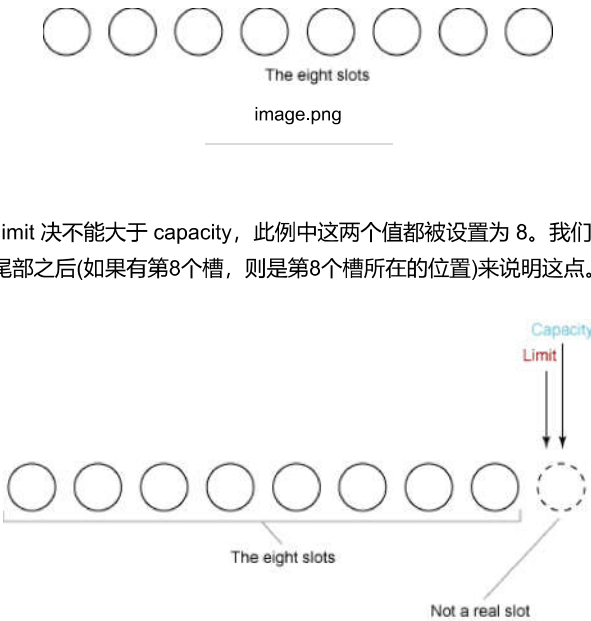
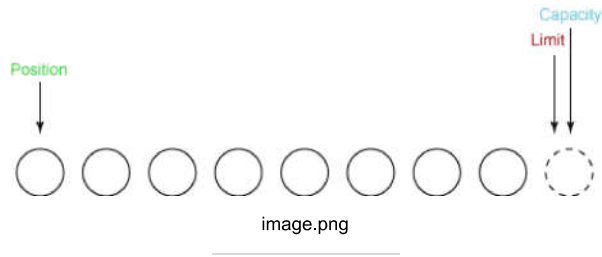


image.png

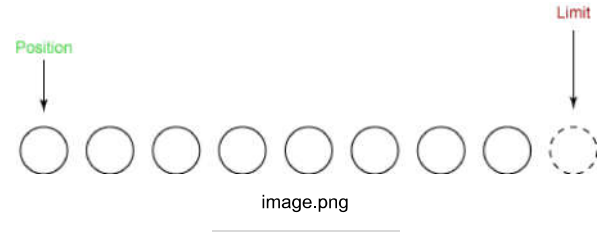
position 设置为0。如果我们读一些数据到缓冲区中，那么下一个读取的数据就进入 slot 0。如果我们从缓冲区写一些数据，从缓冲区读取的下一个字节就来自 slot 0。 position 设置如下所示：



由于 capacity 不会改变，所以我们在下面的讨论中可以忽略它。

第一次读取

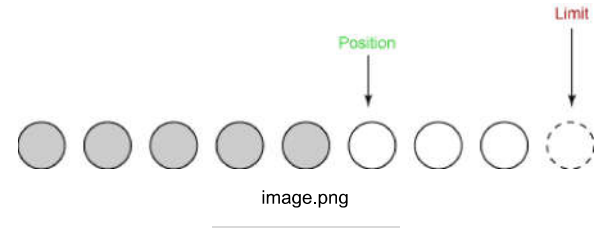
现在我们可以开始在新创建的缓冲区上进行读/写操作。首先从输入通道中读一些数据到缓冲区中。第一次读取得到三个字节。它们被放到数组中从 position 开始的位置，这时 position 被设置为 0。读完之后，position 就增加到 3，如下所示：



limit 没有改变。

第二次读取

在第二次读取时，我们从输入通道读取另外两个字节到缓冲区中。这两个字节储存在由 position 所指定的位置上，position 因而增加 2：



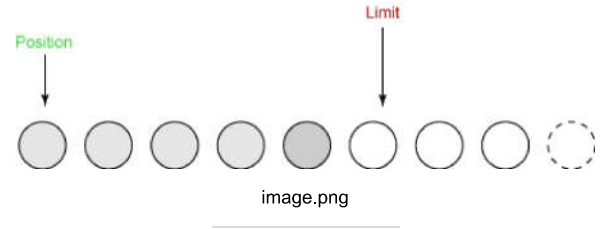
limit 没有改变。

flip

现在我们要将数据写到输出通道中。在这之前，我们必须调用 flip() 方法。这个方法做两件非常重要的事：

- 它将 limit 设置为当前 position。
- 它将 position 设置为 0。

前一小节中的图显示了在 flip 之前缓冲区的情况。下面是在 flip 之后的缓冲区：



我们现在可以将数据从缓冲区写入通道了。position 被设置为 0，这意味着我们得到的下一个字节是第一个字节。limit 已被设置为原来的 position，这意味着它包括以前读到的所有字节，并且一个字节也不多。

第一次写入

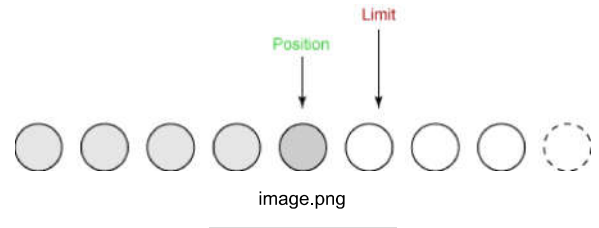
在第一次写入时，我们从缓冲区中取四个字节并将它们写入输出通道。这使得 position 增加到 4，而 limit 不变，如下所示：

^

⌂

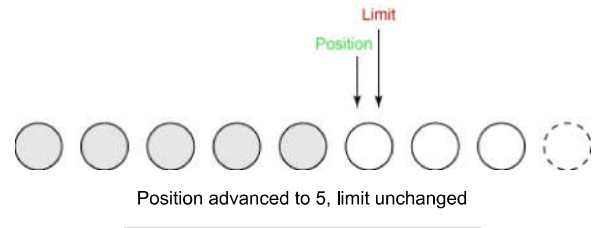
🔖

🔗



第二次写入

我们只剩下一个字节可写了。limit在我们调用 flip() 时被设置为 5，并且 position 不能超过 limit。所以最后一次写入操作从缓冲区取出一个字节并将它写入输出通道。这使得 position 增加到 5，并保持 limit 不变，如下所示：



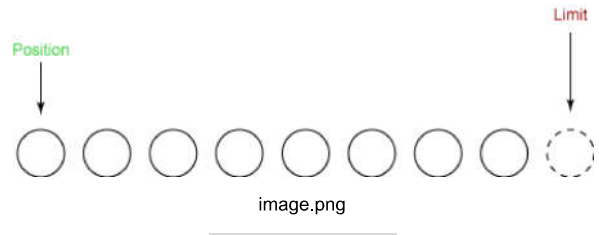
clear

最后一步是调用缓冲区的 clear() 方法。这个方法重设缓冲区以便接收更多的字节。Clear 做两种非常重要的事情：

它将 limit 设置为与 capacity 相同。

它设置 position 为 0。

下图显示了在调用 clear() 后缓冲区的状态：



缓冲区现在可以接收新的数据了。

NIO Buffer Characteristics

- buffer是java NIO中的块的基础
- buffer可以提供一个固定大小的容器来读取和写入数据
- 任意一个buffer都是可读的，只有选中的buffer才可写
- buffer是channel的端点
- 在只读的模式下，buffer的内容不可变，但是的他/她的几个变量，position，limit都是可变的
- 默认情况下，buffer不是线程安全的

How to Read from NIO Buffer

- 首先创建一个指定大小的buffer

```
ByteBuffer byteBuffer = ByteBuffer.allocate(512);
```
- 将buffer转换为读模式

```
byteBuffer.flip();
```

^

📄

🔖

🔗

- 然后从channel中读取数据到buffer中

```
int numberOfBytes = fileChannel.read(byteBuffer);
```

- 用户从buffer中读取数据

```
char c = (char)byteBuffer.get();
```

How to Write to NIO Buffer

- Create a buffer by allocating a size.

```
ByteBuffer byteBuffer = ByteBuffer.allocate(512);//512 becomes the capacity
```

- Put data into buffer

```
byteBuffer.put((byte) 0xff);
```

Java NIO 读写文件实例程序

下面的程序实现了一个简单的利用buffer和channel读取数据

```
package Channel;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;

public class BufferExample {

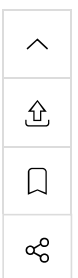
    public static void main(String[] args) throws IOException {
        Path path = Paths.get("temp.data");
        write(path);
        read(path);
    }

    private static void write(Path path) throws IOException {
        String input = "NIO Buffer Hello World!";
        byte[] inputBytes = input.getBytes();
        ByteBuffer byteBuffer = ByteBuffer.allocate(inputBytes.length);
        byteBuffer.put(inputBytes);
        byteBuffer.flip();
        FileChannel channelWrite = FileChannel.open(path,
            StandardOpenOption.CREATE, StandardOpenOption.WRITE);
        channelWrite.write(byteBuffer);
        channelWrite.close();
    }

    private static void read(Path path) throws IOException {
        FileChannel channelRead = FileChannel.open(path);
        ByteBuffer byteBuffer = ByteBuffer.allocate(512);
        int readBytes = channelRead.read(byteBuffer);
        if(readBytes > 0) {
            byteBuffer.flip();
            byte[] bytes = new byte[byteBuffer.remaining()];
            byteBuffer.get(bytes);
            String fileContent = new String(bytes, "utf-8");
            System.out.println("File Content: " + fileContent);
        }
        channelRead.close();
    }
}
```

参考

- <http://javapapers.com/java/java-nio-buffer/> (<http://javapapers.com/java/java-nio-buffer/>)
- <http://www.importnew.com/19816.html> (<http://www.importnew.com/19816.html>)
- <https://www.ibm.com/developerworks/cn/education/java/j-nio/j-nio.html> (<https://www.ibm.com/developerworks/cn/education/java/j-nio/j-nio.html>)





六尺帐篷 (/u/f8e9b1c246f1)

写了 245418 字，被 15240 人关注，获得了 1429 个喜欢

(/u/f8e9b1c246f1)

 喜欢

21








更多分享

(http://cwb.assets.jianshu.io/notes/images/1526789)


被以下专题收入，发现更多相似内容

投稿管理


+ 收入我的专题



Android知识 (/c/3fde3b545a35?utm_source=desktop&utm_medium=notes-included-collection)




Android... (/c/5139d555c94d?utm_source=desktop&utm_medium=notes-included-collection)




Android开发 (/c/d1591c322c89?utm_source=desktop&utm_medium=notes-included-collection)



程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-collection)



首页投稿 (/c/bDHhpK?utm_source=desktop&utm_medium=notes-included-collection)



自定义views (/c/e3e7d9e32134?utm_source=desktop&utm_medium=notes-included-collection)

