http://wiki.jikexueyuan.com/project/java-nio-zh/java-nio-buffer.html

Java NIO Buffers用于和NIO Channel交互。正如你已经知道的，我们从channel中读取数据到buffers里，从buffer把数据写入到channels.
buffer本质上就是一块内存区，可以用来写入数据，并在稍后读取出来。这块内存被NIO Buffer包裹起来，对外提供一系列的读写方便开发的接口。

Buffer基本用法（Basic Buffer Usage）

利用Buffer读写数据，通常遵循四个步骤：

- 把数据写入buffer；
- 调用flip；
- 从Buffer中读取数据；
- 调用buffer.clear()或者buffer.compact()

当写入数据到buffer中时，buffer会记录已经写入的数据大小。当需要读数据时，通过flip()方法把buffer从写模式调整为读模式；在读模式下，可以读取所有已经写入的数据。
当读取完数据后，需要清空buffer，以满足后续写入操作。清空buffer有两种方式：调用clear()或compact()方法。clear会清空整个buffer，compact则只清空已读取的数据，未被读取的数据会被移动到buffer的开始位置，写入位置则近跟着未读数据之后。
这里有一个简单的buffer案例，包括了write，flip和clear操作：

```
RandomAccessFile aFile
    = new RandomAccessFile("data/nio-data.txt", "rw");
FileChannel inChannel = aFile.getChannel();

//create buffer with capacity of 48 bytes
ByteBuffer buf = ByteBuffer.allocate(48);
//read into buffer.
int bytesRead = inChannel.read(buf);
while (bytesRead != -1) {
```

```
  buf.flip();  //make buffer ready for read
  while(buf.hasRemaining()){
      System.out.print((char) buf.get()); // read 1 byte at a
time
  }
  buf.clear(); //make buffer ready for writing
  bytesRead = inChannel.read(buf);
}
aFile.close();
```

Buffer的容量，位置，上限（Buffer Capacity, Position and Limit）

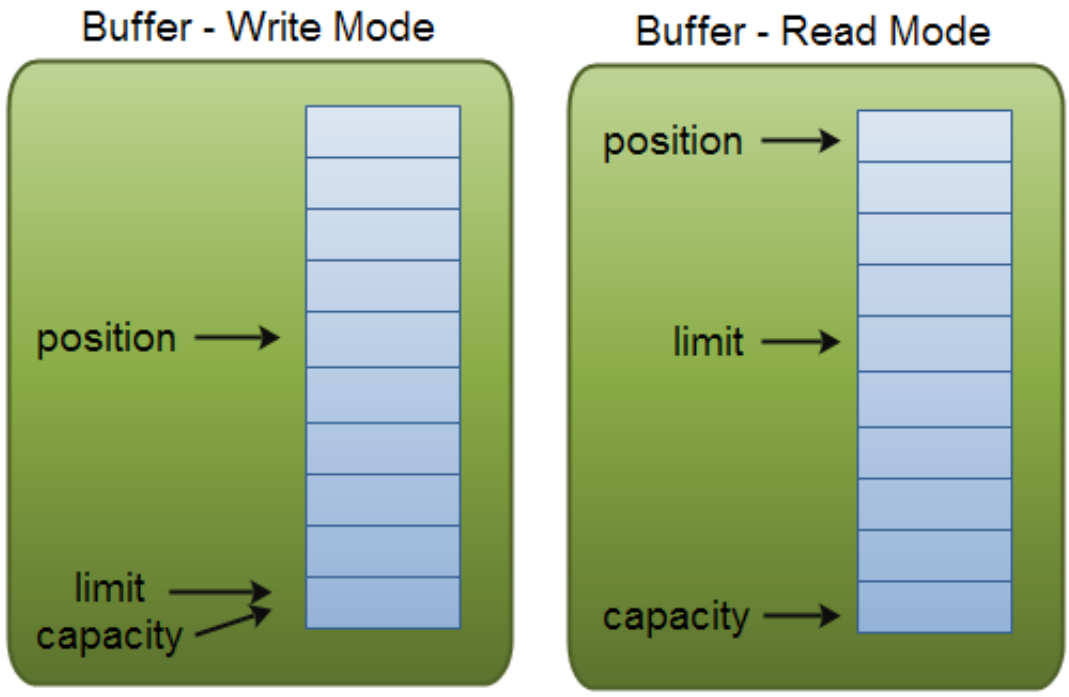
buffer缓冲区实质上就是一块内存，用于写入数据，也供后续再次读取数据。这块内存被NIO Buffer管理，并提供一系列的方法用于更简单的操作这块内存。

一个Buffer有三个属性是必须掌握的，分别是：

- capacity容量
- position位置
- limit限制

position和limit的具体含义取决于当前buffer的模式。capacity在两种模式下都表示容量。

下面有张示例图，描诉了不同模式下position和limit的含义：



Buffer capacity, position and limit in write and read mode.

容量（Capacity）

作为一块内存，buffer有一个固定的大小，叫做capacity容量。也就是最多只能写入容量值得字节，整型等数据。一旦buffer写满了就需要清空已读数据以便下次继续写入新的数据。

位置（Position）

当写入数据到Buffer的时候需要中一个确定的位置开始，默认初始化时这个位置position为0，一旦写入了数据比如一个字节，整型数据，那么position的值就会指向数据之后的一个单元，position最大可以到capacity-1.

当从Buffer读取数据时，也需要从一个确定的位置开始。buffer从写入模式变为读取模式时，position会归零，每次读取后，position向后移动。

上限（Limit）

在写模式，limit的含义是我们所能写入的最大数据量。它等同于buffer的容量。

一旦切换到读模式，limit则代表我们所能读取的最大数据量，他的值等同于写模式下

position的位置。(0到limit表示buffer中存的数据量, capacity表示buffer能存的最大数据量)

Buffer Types

Java NIO有如下具体的Buffer类型:

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

正如你看到的，Buffer的类型代表了不同数据类型，换句话说，Buffer中的数据可以是上述的基本类型;

MappedByteBuffer稍有不同，我们会单独介绍。

分配一个Buffer（Allocating a Buffer）

为了获取一个Buffer对象，你必须先分配。每个Buffer实现类都有一个allocate()方法用于分配内存。下面看一个实例,开辟一个48字节大小的buffer:

```
ByteBuffer buf = ByteBuffer.allocate(48);
```

开辟一个1024个字符的CharBuffer:

```
CharBuffer buf = CharBuffer.allocate(1024);
```

写入数据到Buffer（Writing Data to a Buffer）

写数据到Buffer有两种方法:

- 从Channel中写数据到Buffer（`int bytesRead = inChannel.read(buf);`）
- 手动写数据到Buffer，调用put方法（`buf.put(127);`）

翻转（flip()）

flip()方法可以吧Buffer从写模式切换到读模式。调用flip方法会把position归零，并设置limit为之前的position的值。 也就是说，现在position代表的是读取位置，

limit标示的是已写入的数据位置。

从Buffer读取数据（Reading Data from a Buffer）

冲Buffer读数据也有两种方式。

- 从buffer读数据到channel（`int bytesWritten = inChannel.write(buf);`）
- 从buffer直接读取数据，调用get方法（`byte aByte = buf.get();`）

rewind()

Buffer.rewind()方法将position置为0，这样我们可以重复读取buffer中的数据。limit保持不变。

clear() and compact()

一旦我们从buffer中读取完数据，需要复用buffer为下次写数据做准备。只需要调用clear或compact方法。

clear方法会重置position为0，limit为capacity，也就是整个Buffer清空。实际上Buffer中数据并没有清空，我们只是把标记为修改了。

如果Buffer还有一些数据没有读取完，调用clear就会导致这部分数据被"遗忘"，因为我们没有标记这部分数据未读。

针对这种情况，如果需要保留未读数据，那么可以使用compact。 因此compact和clear的区别就在于对未读数据的处理，是保留这部分数据还是一起清空。

mark() and reset()

通过mark方法可以标记当前的position，通过reset来恢复mark的位置，这个非常像canva的save和restore：

```
buffer.mark();

//call buffer.get() a couple of times, e.g. during parsing.

buffer.reset();  //set position back to mark.
```

equals() and compareTo()

可以用eqauls和compareTo比较两个buffer

equals()

判断两个buffer相对，需满足：

- 类型相同
- buffer中剩余字节数相同
- 所有剩余字节相等

从上面的三个条件可以看出，equals只比较buffer中的部分内容，并不会去比较每一个元素。

compareTo()

compareTo也是比较buffer中的剩余元素，只不过这个方法适用于比较排序的：