# Netty

## 1. BIO/NIO/AIO

### ByteBuffer

1. 可通过 `allocateDirect()` 方法，分配堆外内存；
2. 绝对/相对的 `get/put` 方法；
3. mark、position、limit、capacity；
4. `clear()` -position=0,limit=capacity,mark=丢弃；
5. `flip()` -limit=position,position=0,mark=丢弃；
6. `reset()`
7. `A view buffer is simply another buffer whose content is backed by the byte buffer.`，操作 `view buffer` 或者 `backed buffer` 会相互影响（类似，如果 `backed array`，直接操作数组，也会影响buffer）；

### ByteBuf

1. 建议使用 `Unpooled` 进行buf创建；
2. readIndex、writeIndex、capacity；
3. 实现了 `ReferenceCounted` 接口；
4. `discardReadBytes()` -丢弃已读的字节；
5. `clear()` -readIndex、writeIndex均设置为0，但是buf的内容不会清空（和ByteBuffer类似）；
6. 如：`mark/resetReaderIndex()`，可以分别对readIndex、writeIndex进行mark和reset；
7. buf复制方法：`duplicate()`、`slice()`；修改复制得到的buf或原buf的内容，会互相影响；

### 同步阻塞/异步非阻塞

1. 要明白 `同步/异步` 指啥？ `是否阻塞` 又指啥？
2. 两个目标：`IO操作`、`线程`；
3. `AsynchronousChannelGroup` - AIO；

## 2. Netty相关类

- **ChannelInboundHandlerAdapter**
- **NioEventLoopGroup**

```
EventLoopGroup bossGroup = new NioEventLoopGroup(); // (1)
EventLoopGroup workerGroup = new NioEventLoopGroup();
```

> The first one, often called 'boss', accepts an incoming connection.
> The second one, often called 'worker', handles the traffic of the accepted connection once the boss accepts the connection and registers the accepted connection to the worker.

- **ServerBootstrap** - `bind(port)`
- **Bootstrap** - `connect(host, port)`

- **ChannelHandlerContext#write(ByteBuf)** 会自动将ByteBuf给release掉；

```
@Override
public void channelRead(ChannelHandlerContext ctx, Object msg) {
    ctx.write(msg); // (1)
    ctx.flush(); // (2)
}
```

> ctx.write(Object) does not make the message written out to the wire. It is buffered internally and then flushed out to the wire by ctx.flush(). Alternatively, you could call ctx.writeAndFlush(msg) for brevity.

- **ByteToMessageDecoder**、**MessageToMessageCodec**、**MessageToByteEncodor**
- **ReplayingDecoder** - 解决"未知长度的frame读取"的场景；

## 3. Netty零散点

- bossGroup 线程池则只是在 Bind 某个端口后，获得其中一个线程作为 MainReactor，专门处理端口的 Accept 事件，每个端口对应一个 Boss 线程。
- 虽然 Netty 的线程模型基于主从 Reactor 多线程，借用了 MainReactor 和 SubReactor 的结构。但是实际实现上 SubReactor 和 Worker 线程在同一个线程池中
- **Netty 的Zero-copy 体现在如下几个个方面** （参考：对于Netty ByteBuf 的零拷贝(Zero Copy) 的理解 ）：
  - Netty 提供了CompositeByteBuf 类, 它可以将多个 ByteBuf 合并为一个逻辑上的 ByteBuf, 避免了各个 ByteBuf 之间的拷贝.
  - 通过 wrap 操作, 我们可以将 byte[] 数组、ByteBuf、ByteBuffer等包装成一个** Netty** ByteBuf 对象, 进而避免了拷贝操作.
  - ByteBuf 支持 slice 操作, 因此可以将 ByteBuf 分解为多个共享同一个存储区域的 ByteBuf, 避免了内存的拷贝.
  - 通过 FileRegion 包装的FileChannel.tranferTo 实现文件传输, 可以直接将文件缓冲区的数据发送到目标 Channel, 避免了传统通过循环** write** 方式导致的内存拷贝问题.

---

## 博客

用户指南&文档 | User guide for 4.x | 浅谈NIO与零拷贝 | netty性能优化 | 用Netty开发中间件：高并发性能优化 | 计算机组成 -- DMA | 同步/异步/阻塞/非阻塞/BIO/NIO/AIO讲的最清楚的好文章 | AIO示例 | IO 基础——用户空间与内核空间 | 零拷贝原理 | ByteBuf内存规格 | [内存规格] | Netty堆外内存泄漏排查