# IOcopy

建议实现读取定长数据的时候, 使用make定长slice + io.ReadFull的方式来代替 bytes.Buferr + io.CopyN, bytes.Buffer的初始长度很屌, 每次扩容长度\*2, 都需要拷贝数据, 大包就会影响性能, 发现包大了性能相对于其他框架性能下降, 就是因为io copy的多了.

测试pbrpc server性能的时候发现，如果消息大小比较大的场景下(>1MB)，性能会有明显的退化，跟C++版的brpc有比较大的差距，看了一下代码，gdp里面有些不必要的copy操作，改掉之后性能就能有所优化，例如packing/reader.go里的DiscardMessage，可以用io.ReadFull来代替两次copy操作



见的 bytes、bufio、ringbuffer 等 buffer 库，均存在 growth 需要 copy 原数组数据，以及只能扩容无法缩容，占用大量内存等问题。因此我们希望引入一种新的 Buffer 形式，一举解决上述两方面的问题。

go官方本身就提供了一个net包，然后底层也是用的epoll，但是，采用的是来一个链接就启动一个goroutine，对于短链接来说，可以通过goroutine池进行复用，可是对于长链接服务或者工具如rpc等，如果继续使用官方自带的net库就会有大量的goroutine进行常驻，熟悉go的调度模型mpg就会知道，goroutine并不是越多，速度就越快。当goroutine以几十万或者上百万个的时候程序就会非常缓慢。不仅仅如此，因为goroutine是属于有栈协程，所以会消耗大量的内存。所以对于高并发流量或者大量长连接服务场景来说，go的net包就不是特别的适合，所以需要单独开发调度epoll，只需要几十个goroutine就可以维护几十万上百万链接以及百万流量，可以大大节省内存资源以及延迟消耗

经过上述过程，数据只经过了2次copy就从磁盘传送出去了。这个才是真正的Zero-Copy(这里的零拷贝是针对kernel来讲的，数据在kernel模式下是Zero-Copy)。

正是Linux2.4的内核做了改进，Java中的TransferTo()实现了Zero-Copy,如下图：

Zero-Copy技术的使用场景有很多，比如Kafka, 又或者是Netty等，可以大大提升程序的性能。