https://mp.weixin.qq.com/s/6Yf-7QFXzPiH95BOXJl\_IA

# 什么是零拷贝技术？

这是一道高频的面试题，而且在很多技术中都使用到了，比如java NIO、kafka、Netty、Linux等等。作为一个非常重要的知识点，而且又是高频面试题，有必要从零开始好好地认识一下。

OK，开始今天的文章。

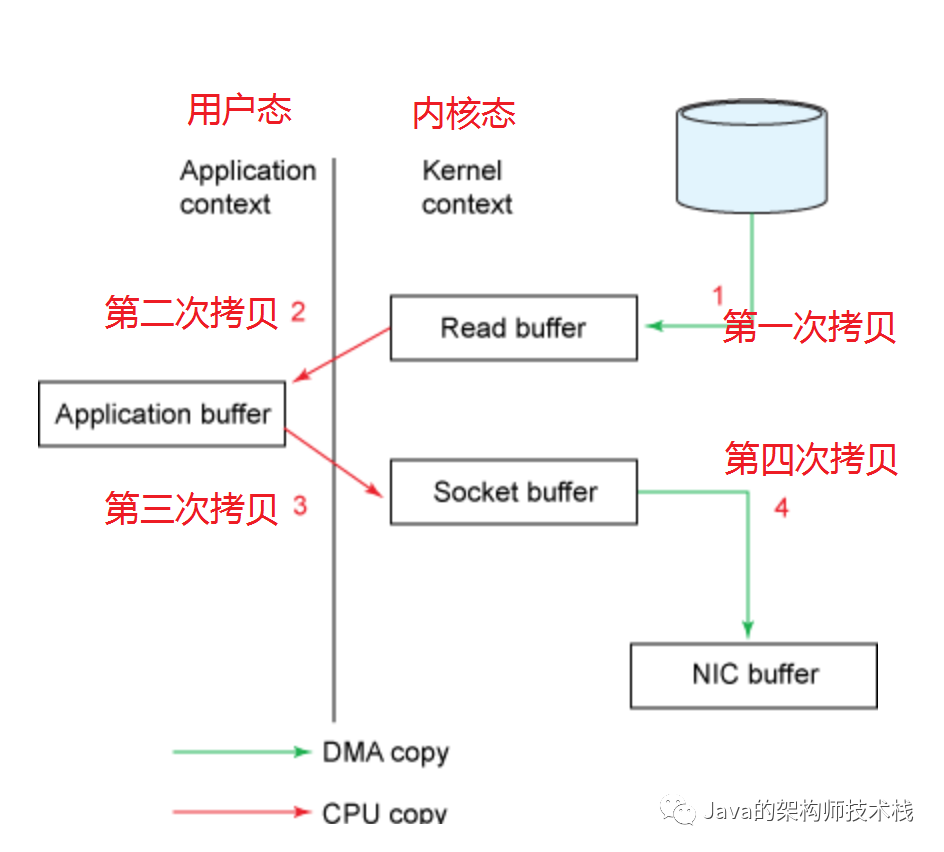
### 一、什么是零拷贝？

**1、从一个案例说起**

为了解释这个概念，我们先要从一个需求说起，说某天某领导给你下发了一个任务，完成一个从文件中读取数据，并传输到网络上的一个小程序。代码很简单：

首先我们在我们的操作系统中找到这个文件，然后把数据先读到缓冲区，最后把缓冲区的数据发送到网络上。

代码是很简单，现在我们考虑一下，这个数据从电脑到网络整个传输的过程：



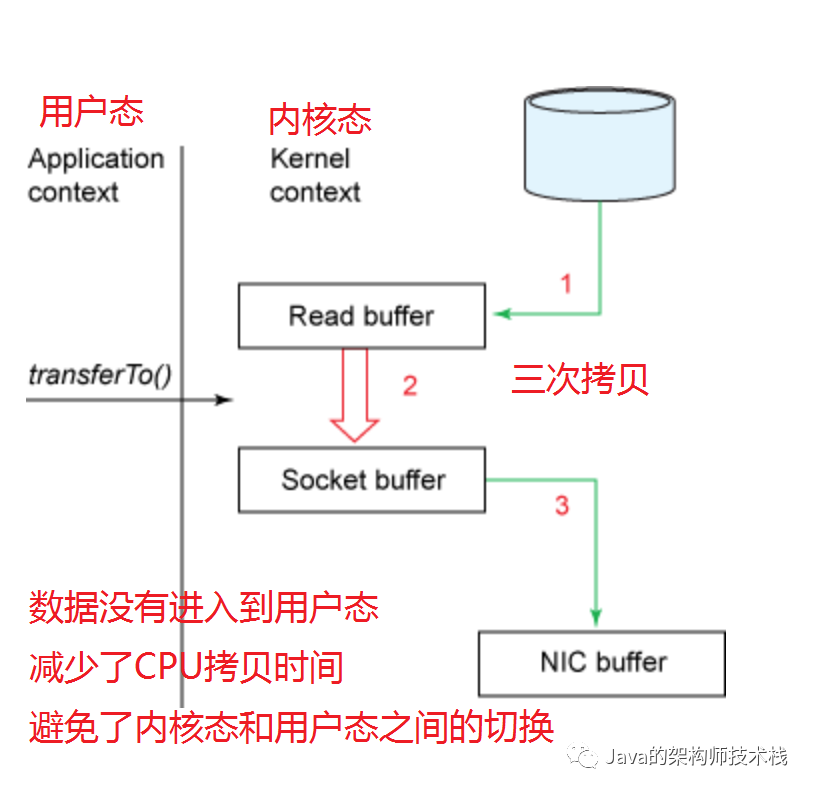
现在我们可以看到1->2->3->4的整个过程一共经历了四次拷贝的方式，**但是真正消耗资源和浪费时间的是第二次和第三次，因为这两次都需要经过我们的CPU拷贝，而且还需要内核态和用户态之间的来回切换。**想想看，我们的CPU资源是多么宝贵，要处理大量的任务。还要去拷贝大量的数据。如果能把CPU的这两次拷贝给去除掉，岂不快哉！！！既能节省CPU资源，还可以避免内核态和用户态之间的切换。

这里还要先说一下用户态和内核态的区别：

* 处于用户态执行时，进程所能访问的内存空间和对象受到限制，其所处于占有的处理器是可被抢占的；
* 处于内核态执行时，则能访问所有的内存空间和对象，且所占有的处理器是不允许被抢占的。

**2、优化方案**

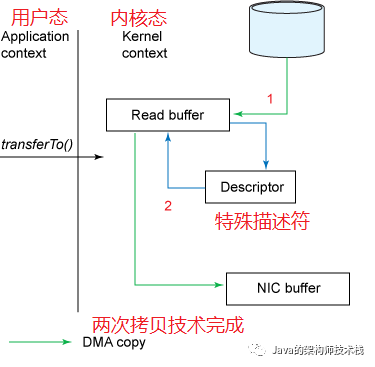
要去除第二次和第三次之间的拷贝，Linux开发人员也早就注意到了这个问题，于是在linux 2.1内核中，添加了 “**数据被copy到socket buffe**r”的动作，于是我们的java NIO，可以直接调用transferTo() 的方法，就可以实现这种现象。



现在一看，感觉性能资源都得到了很大的提升，不过现在还不并不是完美的。因为这三次拷贝还用到了CPU的拷贝技术，就是第二次。不过不要担心。Linux开发人员比我们更加深谋远虑。

**3、零拷贝优化方案**

在Linux2.4 内核做了优化，取而代之的是只包含关于数据的位置和长度的信息的描述符被追加到了socket buffer 缓冲区中。**DMA引擎直接把数据从内核缓冲区传输到协议引擎**（protocol engine），从而消除了最后一次CPU copy。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：



**注意：这里的零拷贝其实是根据内核状态划分的，在这里没有经过CPU的拷贝，数据在用户态的状态下，经历了零次拷贝，所以才叫做零拷贝，但不是说不拷贝。**

OK。现在我们已经了解了什么是零拷贝技术，下面我们再说一下那些数据结构会用到零拷贝技术。

### 二、哪些地方会用到零拷贝技术

**1、java的NIO**

先说java，是因为要给下面的netty做铺垫，在 Java NIO 中的通道（Channel）就相当于操作系统的内核空间（kernel space）的缓冲区，而缓冲区（Buffer）对应的相当于操作系统的用户空间（user space）中的用户缓冲区（user buffer）。

堆外内存（DirectBuffer）在使用后需要应用程序手动回收，而堆内存（HeapBuffer）的数据在 GC 时可能会被自动回收。因此，在使用 HeapBuffer 读写数据时，为了避免缓冲区数据因为 GC 而丢失，NIO 会先把 HeapBuffer 内部的数据拷贝到一个临时的 DirectBuffer 中的本地内存（native memory），这个拷贝涉及到 sun.misc.Unsafe.copyMemory() 的调用，背后的实现原理与 memcpy() 类似。最后，将临时生成的 DirectBuffer 内部的数据的内存地址传给 I/O 调用函数，这样就避免了再去访问 Java 对象处理 I/O 读写。

**（1）MappedByteBuffer**

MappedByteBuffer 是 NIO 基于内存映射（mmap）这种零拷贝方式的提供的一种实现，意思是把一个文件从 position 位置开始的 size 大小的区域映射为内存映像文件。这样之添加地址映射，而不进行拷贝。

**（2）DirectByteBuffer**

DirectByteBuffer 的对象引用位于 Java 内存模型的堆里面，JVM 可以对 DirectByteBuffer 的对象进行内存分配和回收管理，是 MappedByteBuffer 的具体实现类。因此同样具有零拷贝技术。

**（3）FileChannel**

FileChannel 定义了 transferFrom() 和 transferTo() 两个抽象方法，它通过在通道和通道之间建立连接实现数据传输的。

我们直接看Linux2.4的版本，socket缓冲区做了调整，DMA带收集功能。

（1）DMA从拷贝至内核缓冲区

（2）将数据的位置和长度的信息的描述符增加至内核空间(socket缓冲区)

（3）DMA将数据从内核拷贝至协议引擎

这个复制过程是零拷贝过程。

**2、Netty**

Netty 中的零拷贝和上面提到的操作系统层面上的零拷贝不太一样, 我们所说的 Netty 零拷贝完全是基于（Java 层面）用户态的。

（1）Netty 通过 DefaultFileRegion 类对FileChannel 的 tranferTo() 方法进行包装，相当于是间接的通过java进行零拷贝。

（2）我们的数据传输一般都是通过TCP/IP协议实现的，在实际应用中，很有可能**一条完整的消息被分割为多个数据包进行网络传输，而单个的数据包对你而言是没有意义的，只有当这些数据包组成一条完整的消息时你才能做出正确的处理**，而Netty可以通过零拷贝的方式将这些数据包组合成一条完整的消息供你来使用。

此时零拷贝的作用范围仅在用户空间中。那Netty是如何实现的呢？为此我们就要找到Netty进行数据传输的接口，这个接口一定包含了可以实现零拷贝的功能，这个接口就是ChannelBuffer。

**既然有接口肯定就有实现类，一个最主要的实现类是CompositeChannelBuffer，这个类的主要作用是将多个ChannelBuffer组成一个虚拟的ChannelBuffer来进行操作**

**为什么说是虚拟的呢，因为CompositeChannelBuffer并没有将多个ChannelBuffer真正的组合起来，而只是保存了他们的引用，这样就避免了数据的拷贝，实现了Zero Copy。**

（3）ByteBuf 可以通过 wrap 操作把字节数组、ByteBuf、ByteBuffer 包装成一个 ByteBuf 对象, 进而避免了拷贝操作

（4）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

**3、kafka**

Kafka 的索引文件使用的是 mmap + write 方式，数据文件使用的是 sendfile 方式。适用于系统日志消息这种高吞吐量的大块文件的数据持久化和传输。

如果有10个消费者，传统方式下，数据复制次数为4\*10=40次，而使用“零拷贝技术”只需要1+10=11次，一次为从磁盘复制到页面缓存，10次表示10个消费者各自读取一次页面缓存。