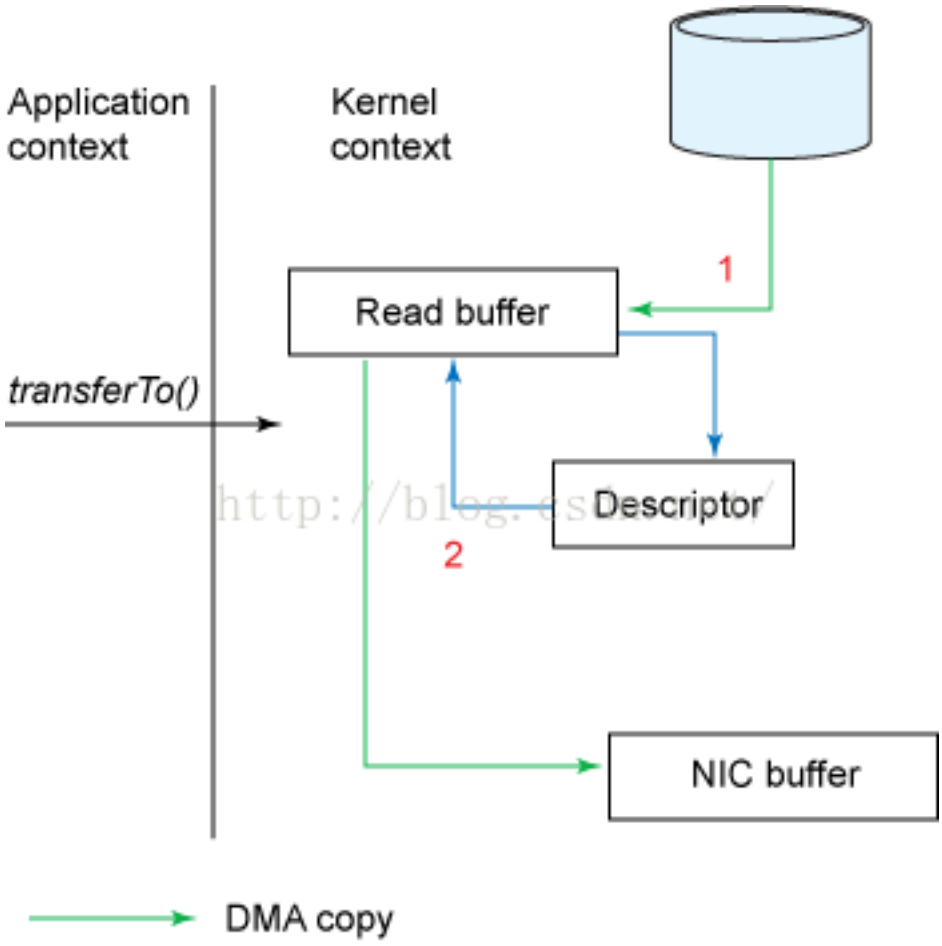
为何要懂零拷贝原理？因为 rocketmq 存储核心使用的就是零拷贝原理。

1. io 读写的方式
    1. 中断
    2. DMA
2. 中断方式
0. 中断方式的流程图如下：

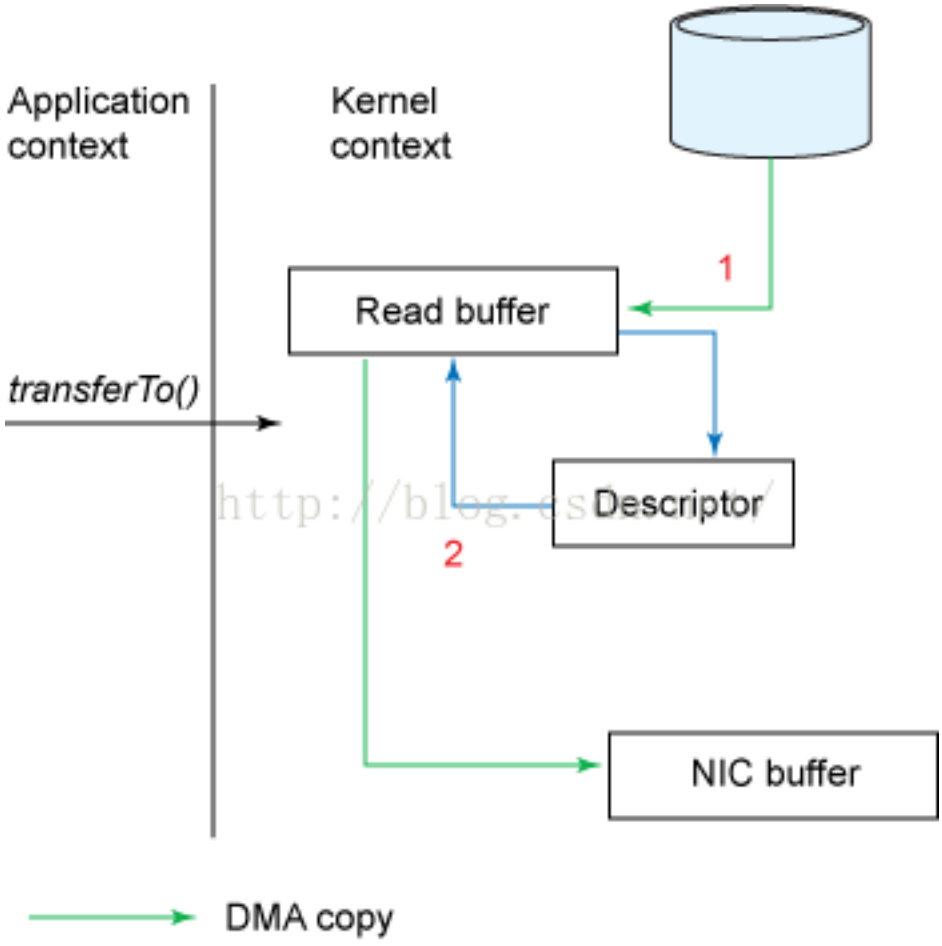


1. 用户进程发起数据读取请求
    2. 系统调度为该进程分配 cpu
    3. cpu 向 io 控制器(ide,scsi)发送 io 请求
    4. 用户进程等待 io 完成，让出 cpu
    5. 系统调度 cpu 执行其他任务
    6. 数据写入至 io 控制器的缓冲寄存器
    7. 缓冲寄存器满了向 cpu 发出中断信号

8. cpu 读取数据至内存
    1. 缺点：中断次数取决于缓冲寄存器的大小

3. DMA ： 直接内存存取

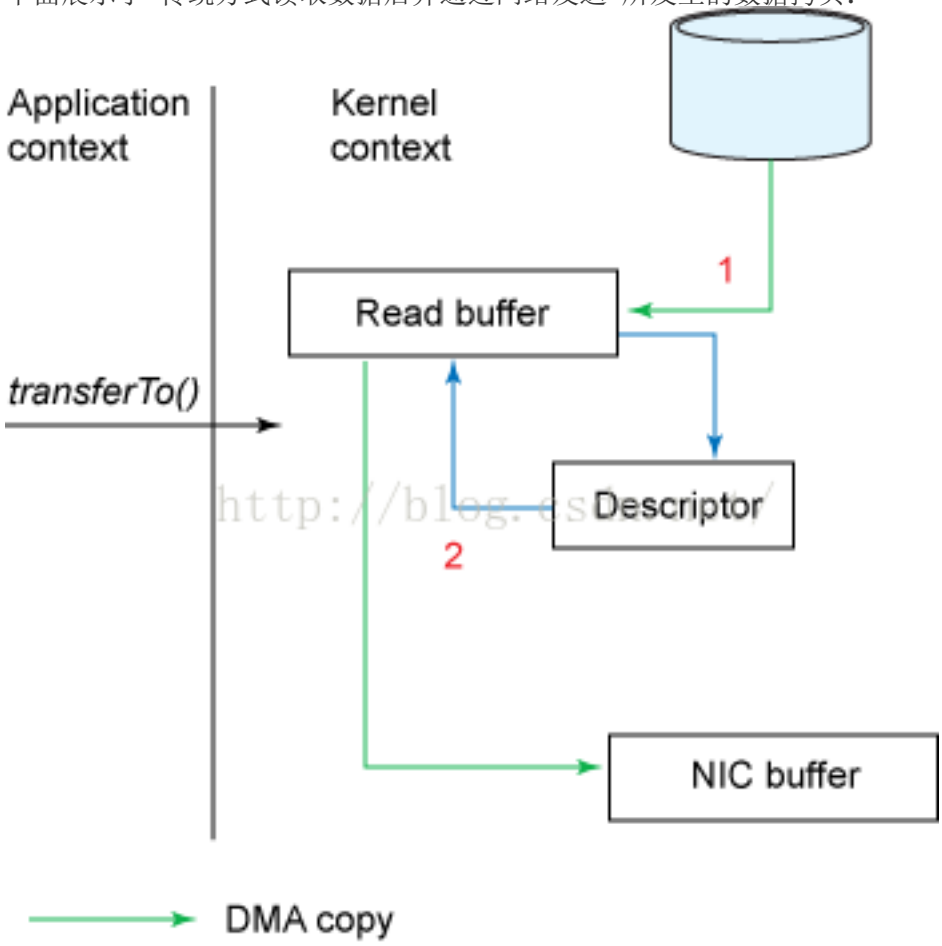
0. DMA 方式的流程图如下：



0. 用户进程发起数据读取请求
    1. 系统调度为该进程分配 cpu
    2. cpu 向 DMA 发送 io 请求
    3. 用户进程等待 io 完成，让出 cpu
    4. 系统调度 cpu 执行其他任务
    5. 数据写入至 io 控制器的缓冲寄存器
    6. DMA 不断获取缓冲寄存器中的数据（需要 cpu 时钟）
    7. 传输至内存（需要 cpu 时钟）
    8. 所需的全部数据获取完毕后向 cpu 发出中断信号
        1. 优点：减少 cpu 中断次数，不用 cpu 拷贝数据

4. 数据拷贝

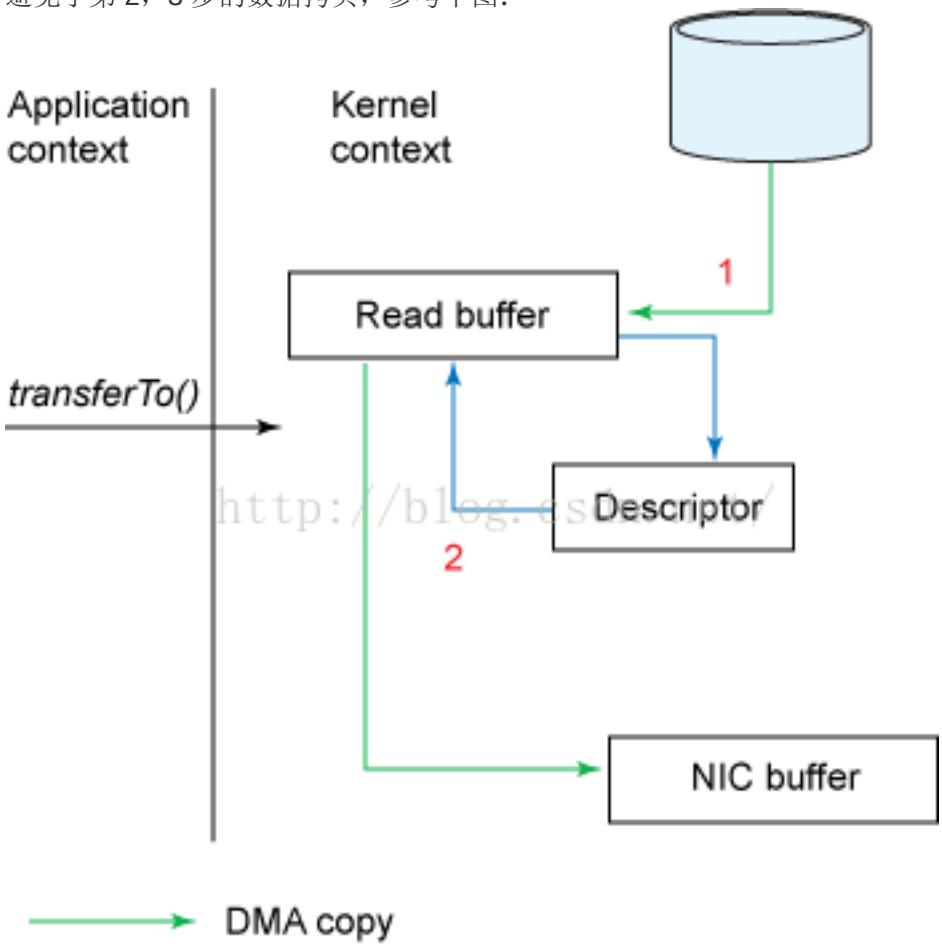
0. 下面展示了 传统方式读取数据后并通过网络发送 所发生的数据拷贝：



    0. 一个 read 系统调用后，DMA 执行了一次数据拷贝，从磁盘到内核空间
        1. read 结束后，发生第二次数据拷贝，由 cpu 将数据从内核空间拷贝至用户空间
        2. send 系统调用，cpu 发生第三次数据拷贝，由 cpu 将数据从用户空间拷贝至内核空间(socket 缓冲区)
        3. send 系统调用结束后，DMA 执行第四次数据拷贝，将数据从内核拷贝至协议引擎
        4. 另外，这四个过程中，每个过程都发生一次上下文切换
            1. 内存缓冲数据，主要是为了提高性能，内核可以预读部分数据，当所需数据小于内存缓冲区大小时，将极大的提高性能。
    2. 零拷贝是为了消除这个过程中冗余的拷贝

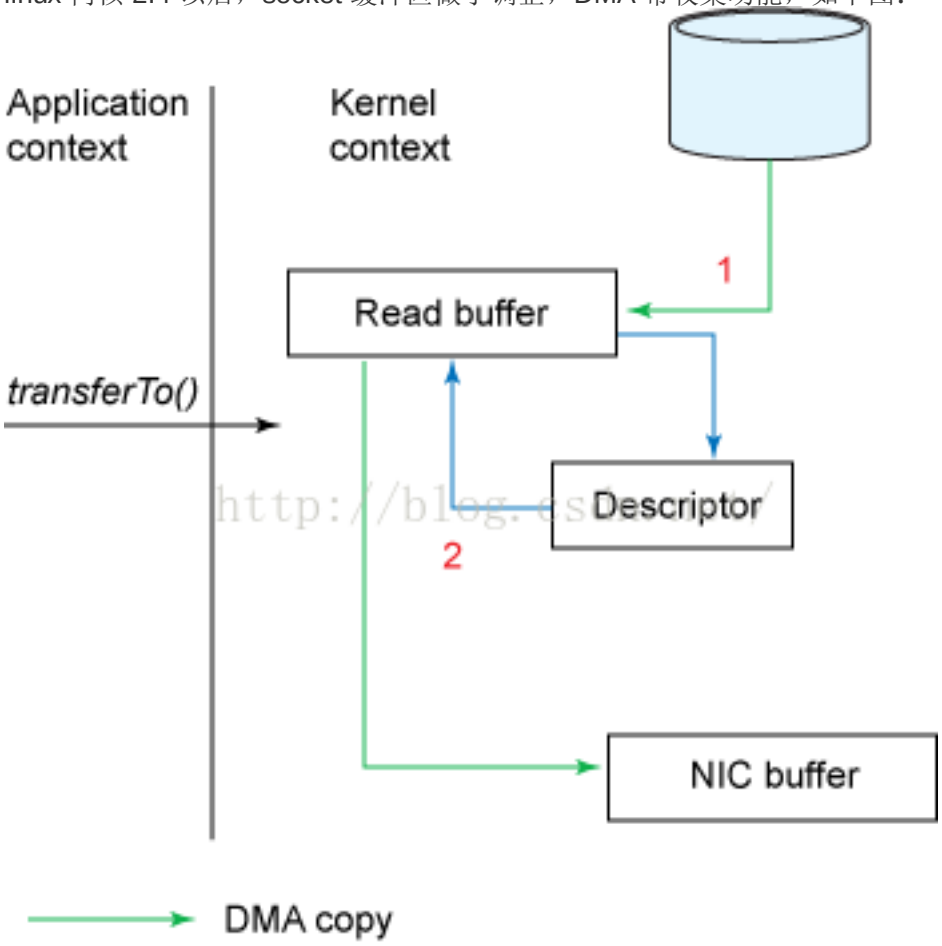
5. 零拷贝-sendfile 对应到 java 中
为 FileChannel.transferTo(long position, long count, WritableByteChannel target)//将数据从文件通道传输到了给定的可写字节通道

0. 避免了第 2，3 步的数据拷贝，参考下图：



0. DMA 从拷贝至内核缓冲区
    1. cpu 将数据从内核缓冲区拷贝至内核空间(socket 缓冲区)
    2. DMA 将数据从内核拷贝至协议引擎
    3. 这三个过程中共发生 2 次上下文切换，分别为发起读取文件和发送数据
        1. 以上过程发生了三次数据拷贝，其中有一次为 cpu 完成

2. linux 内核 2.4 以后，socket 缓冲区做了调整，DMA 带收集功能，如下图：



0. DMA 从拷贝至内核缓冲区
    1. 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)
    2. DMA 将数据从内核拷贝至协议引擎

6. 零拷贝-mmap 对应到 java 中
   为 MappedByteBuffer//文件内存映射

0. 数据不会复制到用户空间，只在内核空间，与 sendfile 类似，但是应用程序可以直接操作该内存。

7. 参考资料

0. http://blog.chinaunix.net/uid-25314474-id-3325879.html
    1. http://blog.chinaunix.net/uid-28874972-id-3725082.html
    2. https://www.ibm.com/developerworks/cn/java/j-zerocopy/#fig1
    3. http://www.ibm.com/developerworks/cn/linux/l-cn-zerocopy1/
    4. https://www.ibm.com/developerworks/cn/linux/l-cn-zerocopy2/