

图文 61 探秘黑科技：基于mmap内存映射实现磁盘文件的高性能读写

497 人次阅读 2019-12-23 07:00:00

[详情](#) [评论](#)

探秘黑科技：基于mmap内存映射实现磁盘文件的高性能读写

石杉老哥重磅力作：《互联网Java工程师面试突击》（第3季）【强烈推荐】：



全程真题驱动，精研Java面试中**6大专题**的高频考点，从面试官的角度剖析面试

(点击下方蓝字试听)

[《互联网Java工程师面试突击》（第3季）](#)

1、mmap：Broker读写磁盘文件的核心技术

今天我们要给大家介绍一个非常关键的黑科技，很多人可能都不太熟悉，这个技术就是mmap技术，而Broker中就是大量的使用mmap技术去实现CommitLog这种大磁盘文件的高性能读写优化的。

通过之前的学习，我们知道了一点，就是Broker对磁盘文件的写入主要是借助直接写入os cache来实现性能优化的，因为直接写入os cache，相当于就是写入内存一样的性能，后续等os内核中的线程异步把cache中的数据刷入磁盘文件即可。

那么今天我们就对这个过程中涉及到的mmap技术进行一定的分析。

2、传统文件IO操作的多次数据拷贝问题

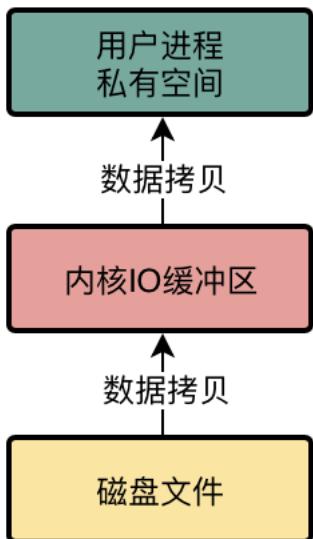
首先我们先来给大家分析一下，假设RocketMQ没有使用mmap技术，就是使用最传统和基本的普通文件IO操作去进行磁盘文件的读写，那么会存在什么样的性能问题？

答案是：多次数据拷贝问题

首先，假设我们有一个程序，这个程序需要对磁盘文件发起IO操作读取他里面的数据到自己这儿来，那么会经过以下一个顺序：

首先从磁盘上把数据读取到内核IO缓冲区里去，然后再从内核IO缓冲区里读取到用户进程私有空间里去，然后我们才能拿到这个文件里的数据

大家看下图



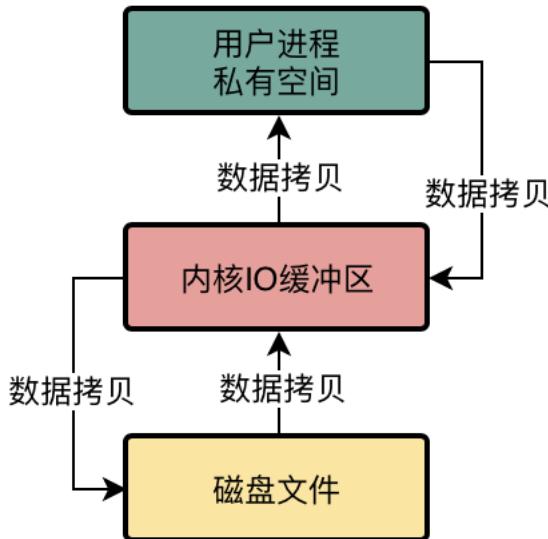
为了读取磁盘文件里的数据，是不是发生了两次数据拷贝？

没错，所以这个就是普通的IO操作的一个弊端，必然涉及到两次数据拷贝操作，对磁盘读写性能是有影响的。

那么如果我们要将一些数据写入到磁盘文件里去呢？

那这个就是一样的过程了，必须先把数据写入到用户进程私有空间里去，然后从这里再进入内核IO缓冲区，最后进入磁盘文件里去

我们看下面的图



在数据进入磁盘文件的过程中，是不是再一次发生了两次数据拷贝？没错，所以这就是传统普通IO的问题，有两次数据拷贝问题。

3、RocketMQ是如何基于mmap技术+page cache技术优化的？

接着我们来看一下，RocketMQ如何利用mmap技术配合page cache技术进行文件读写优化的？

首先，RocketMQ底层对CommitLog、ConsumeQueue之类的磁盘文件的读写操作，基本上都会采用mmap技术来实现。

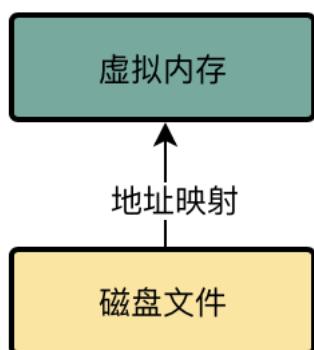
如果具体到代码层面，就是基于JDK NIO包下的MappedByteBuffer的map()函数，来先将一个磁盘文件（比如一个CommitLog文件，或者是一个ConsumeQueue文件）映射到内存来

这里我必须给大家解释一下，这个所谓的内存映射是什么意思

其实有的人可能会误以为是直接把那些磁盘文件里的数据给读取到内存里来了，类似这个意思，但是并不完全是对的。

因为刚开始你建立映射的时候，并没有任何的数据拷贝操作，其实磁盘文件还是停留在那里，只不过它把物理上的磁盘文件的一些地址和用户进程私有空间的一些虚拟内存地址进行了一个映射

我们看下面的图



这个地址映射的过程，就是JDK NIO包下的MappedByteBuffer.map()函数干的事情，底层就是基于mmap技术实现的。

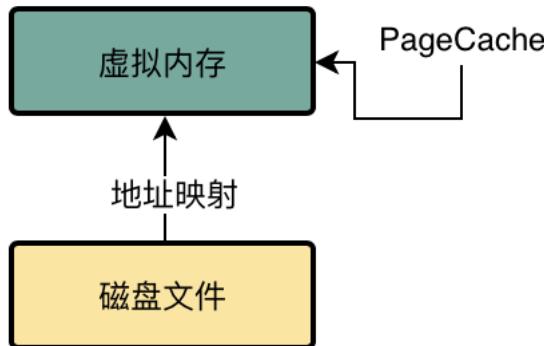
另外这里给大家说明白的一点是，这个mmap技术在进行文件映射的时候，一般有大小限制，在1.5GB~2GB之间

所以RocketMQ才让CommitLog单个文件在1GB，ConsumeQueue文件在5.72MB，不会太大。

这样限制了RocketMQ底层文件的大小，就可以在进行文件读写的时候，很方便的进行内存映射了。

然后接下来要给大家讲的一个概念，就是之前给大家说的PageCache，实际上在这里就是对应于虚拟内存

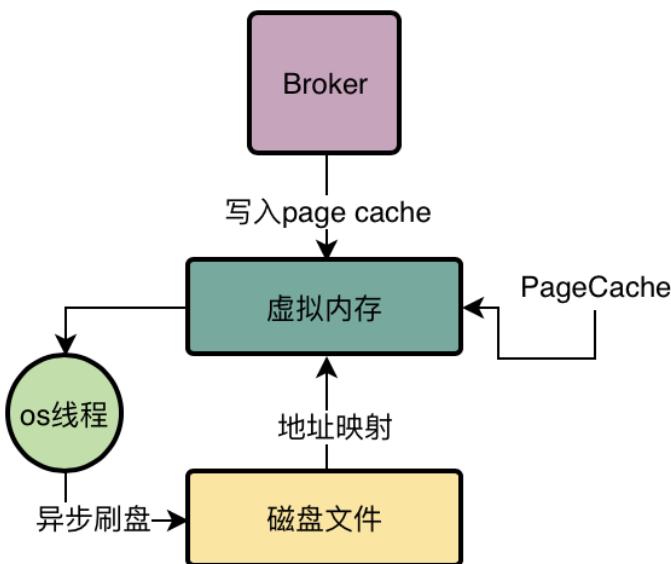
所以我们在下面的图里就给大家画出了这个示意。



4. 基于mmap技术+pagecache技术实现高性能的文件读写

接下来就可以对这个已经映射到内存里的磁盘文件进行读写操作了，比如要写入消息到CommitLog文件，你先把一个CommitLog文件通过MappedByteBuffer的map()函数映射其地址到你的虚拟内存地址。

接着就可以对这个MappedByteBuffer执行写入操作了，写入的时候他会直接进入PageCache中，然后过一段时间之后，由os的线程异步刷入磁盘中，如下图我们可以看到这个示意。



看到这里我们有没有发现什么问题？

对了！就是上面的图里，似乎只有一次数据拷贝的过程，他就是从PageCache里拷贝到磁盘文件里而已！这个就是你使用mmap技术之后，相比于传统磁盘IO的一个性能优化。

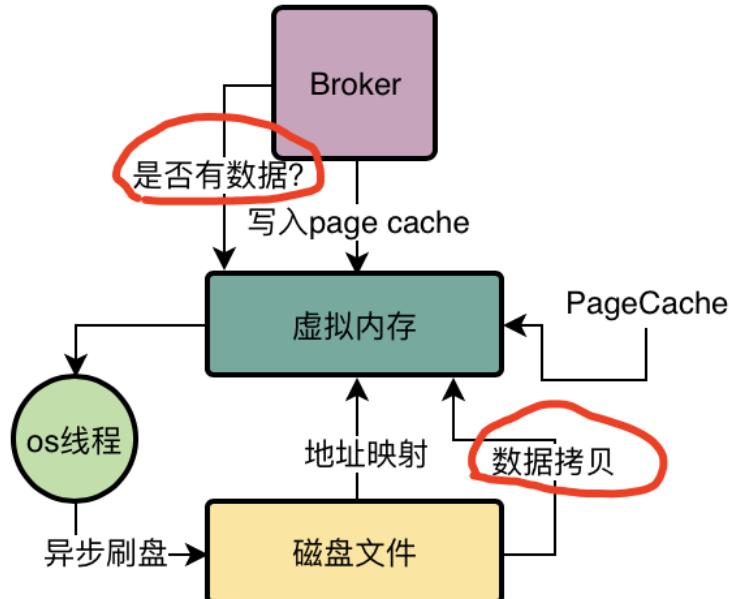
接着如果我们要从磁盘文件里读取数据呢？

那么此时就会判断一下，当前你要读取的数据是否在PageCache里？如果在的话，就可以直接从PageCache里读取了！

比如刚写入CommitLog的数据还在PageCache里，此时你Consumer来消费肯定是从PageCache里读取数据的。

但是如果PageCache里没有你要的数据，那么此时就会从磁盘文件里加载数据到PageCache中去，如下图

而且PageCache技术在加载数据的时候，还会将你加载的数据块的临近的其他数据块也一起加载到PageCache里去。



大家可以看到，在

你读取数据的时候，其实也仅仅发生了一次拷贝，而不是两次拷贝，所以这个性能相较于传统IO来说，肯定又是提高了。

5、预映射机制 + 文件预热机制

接着给大家说几个Broker针对上述的磁盘文件高性能读写机制做的一些优化：

(1) 内存预映射机制：Broker会针对磁盘上的各种CommitLog、ConsumeQueue文件预先分配好MappedFile，也就是提前对一些可能接下来要读写的磁盘文件，提前使用MappedByteBuffer执行map()函数完成映射，这样后续读写文件的时候，就可以直接执行了。

(2) 文件预热：在提前对一些文件完成映射之后，因为映射不会直接将数据加载到内存里来，那么后续在读取尤其是CommitLog、ConsumeQueue的时候，其实有可能会频繁的从磁盘里加载数据到内存中去。

所以其实在执行完map()函数之后，会进行madvise系统调用，就是提前尽可能多的把磁盘文件加载到内存里去。

通过上述优化，才真正能实现一个效果，就是写磁盘文件的时候都是进入PageCache的，保证写入高性能；同时尽可能多的通过map + madvise的映射后预热机制，把磁盘文件里的数据尽可能多的加载到PageCache里来，后续对CosumeQueue、CommitLog进行读取的时候，才能尽可能从内存里读取数据。

6、对今天文章的一点总结

今天我们在之前给大家讲解的PageCache技术基础之上，引入了Broker底层大量采用的mmap技术

实际上在Broker读写磁盘的时候，是大量把mmap技术和pagecache技术结合起来使用的，通过mmap技术减少数据拷贝次数，然后利用pagecache技术实现尽可能优先读写内存，而不是物理磁盘。

End

专栏版权归公众号**狸猫技术窝**所有

未经许可不得传播，如有侵权将追究法律责任

狸猫技术窝其他精品专栏推荐：

[《从零开始带你成为JVM实战高手》](#)

[《21天Java面试突击训练营》（分布式篇）](#) (现更名为：[互联网Java工程师面试突击第2季](#))

[互联网Java工程师面试突击（第1季）](#)

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情在评论区留言提问，我会逐一答疑

如何加群：购买了狸猫技术窝专栏的小伙伴都可以加入**狸猫技术交流群**

具体加群方式, 请参见[目录菜单](#)下的文档: 《付费用户如何加群? 》 (购买后可见)

Copyright © 2015-2019 深圳小鹅网络技术有限公司 All Rights Reserved. 粤ICP备15020529号

 小鹅通提供技术支持