Windows NT File System Internals

第二部分　　管理器

第六章 NT 缓存管理器（一）

- 功能
- 文件流
- 虚拟块缓存
- 在读和写操作中缓存
- 缓存管理器接口
- 缓存管理器客户
- 一些重要数据结构
- 文件大小考虑

虽然存储技术的发展使得产生了更快更廉价的辅助存储器，但是从辅助存储介质上访问数据仍然比从系统内存中的数据缓冲区访问数据慢得多。因此，为了那些管理大量数据的应用程序（比如，数据库管理应用程序）实现更好的性能，在数据被访问之前就把数据放进系统内存（“预读“（read-ahead）功能）就变得重要起来，在内存中保留数据直到数据不再需要（数据缓存），延迟修改数据写出到磁盘以此来获得更好的性能（“延迟写”功能）。

大多数现代操作系统提供文件数据缓存支持（注：即使名声不好的MICROSOFT DOS环境都具有“SmartDrive”缓存模块），通常由个别文件系统或者例如UNIX系统上的全系统的缓冲区缓存模块来执行这个任务，NT缓存管理器（NT Cache Manager）封装了缓存文件数据的功能（注：实际上缓存管理器缓存字节流（不解释），这就可以用任何文件系统定义的数据布局存储在磁盘上，因此文件系统元数据能够被NT缓存管理器缓存）。为了执行这个任务，NT缓存管理器和文件系统驱动以及NT虚拟内存管理器交互。缓存管理器是WINDOWS NT 执行环境的一个完整组件。只要通过WINDOWS NT访问文件数据，我们就利用了缓存管理器提供的服务。如果我们请求访问数据好象很快完成，甚至不用访问磁盘驱动，我们知道是缓存管理器努力地把我们的数据预先读到了内存中。如果请求复制文件或者修改文件几乎立即返回，可能修改的数据被缓存在内存中。当我们注意到硬盘周期性的活动的时候，我们知道修改的数据被延迟写入到磁盘上了。最后，当我们因为系统崩溃而导致数据丢失的时候，很明显缓存管理器应该为此负责。

在这一章中，还有下一章，我会讨论NT缓存管理器的细节，关注缓存管理器的职责，他使用来缓存数据的方法，还有和文件系统驱动以及NT虚拟内存管理器的交互。

功能（Functionality）

NT缓存管理器是NT执行体的一个独一无二的组件，他和NT虚拟内存管理器紧密相关。

它提供一个全系统的一致的辅助存储设备上的数据的缓存。

这个缓存是和适当的文件系统，NT虚拟内存管理器以及I/O管理器共同协作管理的。

它执行文件数据的“预读“

缓存管理器试图根据每个文件被用户应用程序的数据访问模式来调节“预读”策略。因为对被缓存文件的所有I/O请求都通过缓存管理器，缓存管理器能够保存文件数据访问模式的轨迹。因此，如果一个用户应用程序要读（假设）这个文件的前面10K字节，NT缓存管理器通常会试图预读这个文件接下来的64K字节到内存中。然后，如果应用程序试图得到这些数据，他就能够简单地从系统缓存中把数据给应用程序，因此避免了使应用程序等待数据从辅助存储器读进来。对于序列化的文件访问，缓存管理器提供的预读功能会得到很大的性能提升，因为应用程序要求访问的这些数据已经被读到系统内存中。

提供修改数据延迟写的功能

通过把修改数据在系统中保留一段时间然后再实际写到磁盘，NT缓存管理器向用户应用程序提供更快的响应。还能够把多个成批的内存中连续的写操作在一个单个的I/O操作中写所有的修改数据，这通常比执行多个单独的小的写操作更有效率。最后，一个用户进程可能重复修改相同字节范围。通过延迟到磁盘的I/O操作，这写修改只在内存中进行，完全避免了重复写介质的负载。

文件流（File Streams）

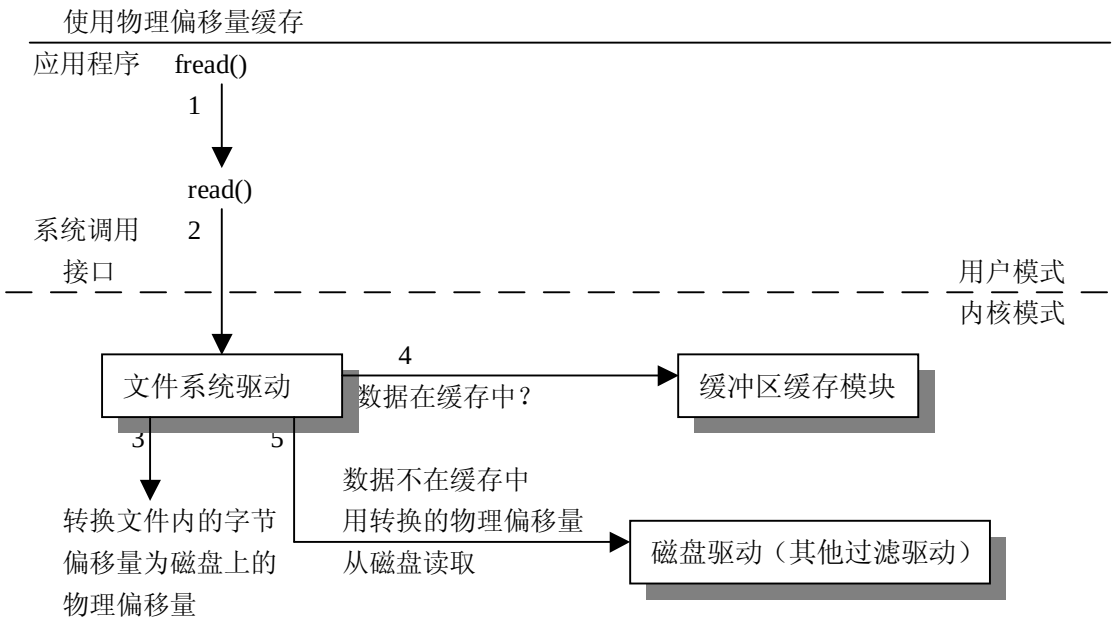
WINDOWS NT中用文件对象数据结构来代表每个打开文件的实例。任何和文件对象关联的线性字节流可以被定义为一个文件流。例如，文件流包括这个文件的数据（注：一个文件可能有多个数据流，如果文件系统支持这个特性，例如，NTFS支持多个数据流。NTFS使用两个单独的字节流（对于一个相同的命名文件）来存储资源和数据来分别关联存储在NT服务器NT文件系统上的Macintosh文件），目录（包含存储在磁盘上的其他文件信息），访问控制列表（ACL），还有和这个文件存储的扩展属性。

NT文件系统创建，删除和操作文件流都是外部产生的用户请求读或者写文件，或者请求操作文件系统特定的数据结构的结果。文件系统识别他要支持和缓存的文件流。例如，除非用户直接访问文件，文件系统将会缓存一个文件的用户数据。对于每个被缓存的文件流，文件系统通常支持缓存和非缓存的访问。

缓存管理器使用内存映射来支持“文件流缓存”，他还完整地缓存内存管理器用于其他用处的分页内存的策略。从缓存管理器的观点看，流只是应该保留在内存中表示信息的一个随机的字节序列。因此，缓存管理器提供的相同的系列服务可以被文件系统驱动用来缓存用户文件数据或者文件系统元数据。

虚拟块缓存（Virtual Block Caching）

有的操作系统使用物理偏移量（或者磁盘块寻址）来在系统内存中缓存文件数据。而NT缓存管理器通过文件映射的方法缓存文件流来提供虚拟块缓存。图6-1显示这两种数据缓存方法的区别（对于被缓存数据来说）。注意，其中是数字表示操作执行的逻辑序列。



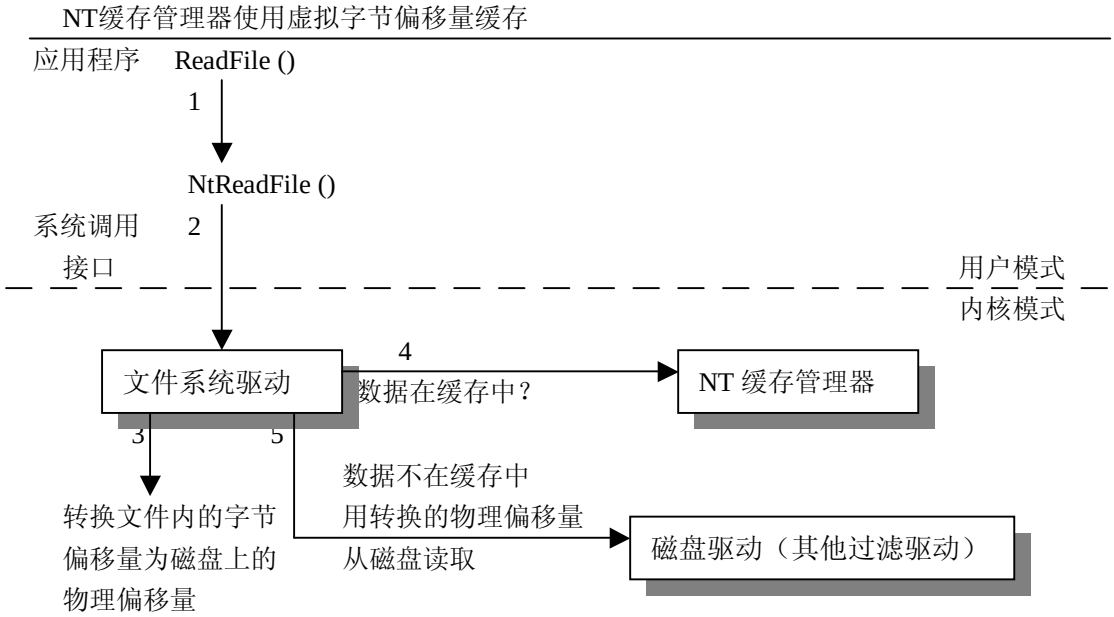


图6-1 比较虚拟块地址缓存个物理块地址缓存

在使用物理块地址来缓存数据的操作系统中（“旧缓冲区缓存（old buffer cache）”在UNIX SVR4中实现），在检查系统缓存中的数据是否有用之前文件系统或者缓存模块必须首先文件中的虚拟字节偏移量为磁盘上的物理块偏移量，因为缓存模块（缓冲区缓存）使用物理磁盘地址来索引缓存数据。但是，在图6-1中显示的，NT缓存管理器只使用文件中的虚拟字节偏移量来索引缓存的信息。NT缓存管理器不需要理解被访问的数据的物理块地址。因此，WINDOWS NT操作系统中的文件系统驱动程序通常只有不能在缓存管理器管理的内存中的缓存中得到数据是时候才把文件中的虚拟字节偏移量转换为磁盘上的物理块偏移量。

使用虚拟块缓存有下面这些好处（和物理块缓存的比较）：

- 某些应用程序可能使用本地系统调用来访问文件数据，例如，NtReadFile或者NtWriteFile（注：通常应用程序使用子系统提供的接口（如WIN32子系统提供的ReadFile函数）来执行读/写操作。调用这些接口例程实际上导致调用本地系统服务，附录A，WINDOWS NT 系统服务，给出一个I/O管理器为数据访问提供的完整系统服务列表），但是并发执行的其他应用程序可能把这个文件数据映射到他们的地址空间中进行读或者读写访问。通过使用虚拟块缓存，通过文件映射，还有适当的同步，给所有应用显示最当前的数据是可能的（注：不管FASTFAT还是NTFA本地文件系统的当前实现都不保证应用程序使用普通系统调用总能得到最新的数据，如果有其他应用程序把文件映射为读或者写访问。但是，在大多数情况下，文件系统尽量保证实际是这样）。
- 概念上，在文件数据被NT缓存管理器映射和被应用程序映射之间没有差别。通过使用文件映射模式，所有的缓存数据的物理内存都变得可用。前面提到，分配物理内存由NT虚拟内存管理器控制；分配给缓存管理器的物理内存数量依赖于系统中的其他组件对内存需求的变化（例如，为印象文件分配的页相对于数据文件分页）。
- I/O管理器常常直接调用缓存管理器，完全绕过文件系统驱动或者网络重定向器驱动。在这

种情况下，使得缓存管理器通过一个硬件虚拟地址查找来解决文件访问成为可能。这比在检查系统缓冲中的数据有效之前先转换虚拟地址到物理磁盘地址要更有效得多。

在读和写操作中缓存

WINDOWS NT操作系统中，用户进程在打开一个文件的时候可以指定文件数据是否缓存在内存中。只有那些打开的时候没有IRP_NOCACHE标志的文件的数据在系统内存中缓存，这个标志用来指出文件数据不在系统内存中缓存。为了理解在上一节中描述的NT缓存管理器是怎样提供缓存功能的，可以把缓存管理器认为是一个应用程序，执行在系统中，他和恰好要打开和那些被执行在同一系统上的其他应用程序打开的文件相同的文件。

为了缓存数据，缓存管理器需要利用系统内存。在第五章里看到，每个执行在WINDOWS NT环境的进程有4GB的虚拟地址空间可用。底半部分地址空间是进程独有的，但是高半部分是操作系统保留的，在每个执行在系统中的进程中共享。这个虚拟地址模式也适用与系统进程，这是一个在系统初始化的时候创建的特殊进程。在系统初始化的时候，NT缓存管理器保留系统进程虚拟地址空间的2GB中的一个虚拟地址范围。因为这个保留给NT缓存管理器单独使用的虚拟地址范围是存在于虚拟地址空间的高2GB中，因此每个执行在系统中的进程都访问这个保留给NT缓存管理器的地址范围。图6-2描绘这个保留给NT缓存管理器的虚拟地址范围。

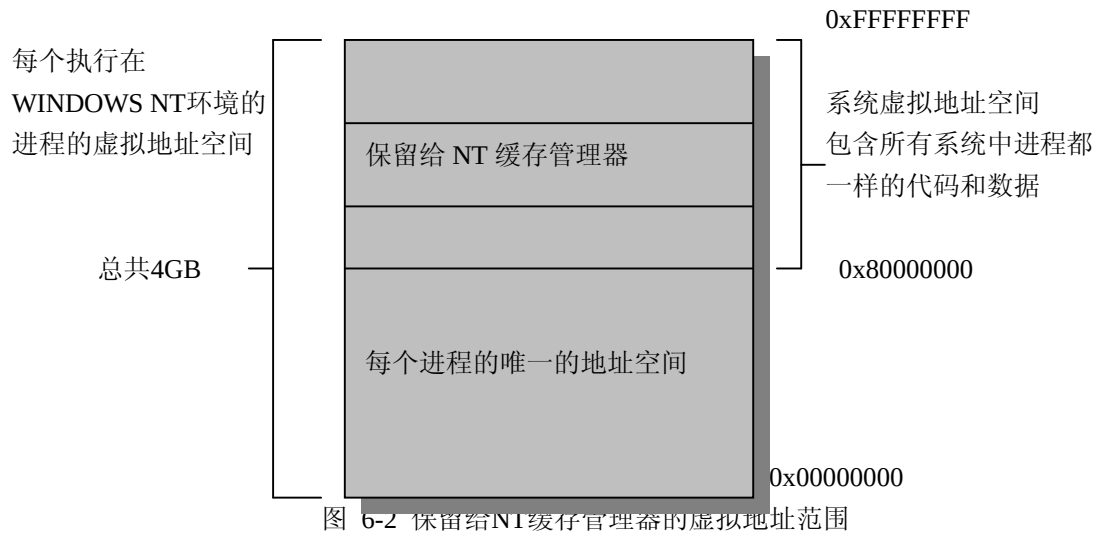


图 6-2 保留给NT缓存管理器的虚拟地址范围

虽然一定的虚拟地址范围是保留给NT缓存管理器单独使用，但是不必给这个虚拟地址范围分配物理页。分配给NT缓存管理器的物理页面是由NT VMM决定，而且是不断调整的。在其他用户进程或者系统组件对物理页的需求降低的时候，VMM可能选择增加分配给缓存管理器的物理内存数量。另一方面，在系统高负载可用物理内存不足的时候，内存管理器可能减少分配给缓存管理器用于缓存文件数据的物理页面。

要注意重要的一点是关于决定物理内存的分配是NT VMM的特权。

缓存管理器使用文件映射来缓存文件数据。文件流的缓存初始化通过文件系统驱动对缓存管理器的一个调用完成。收到这样一个请求的时候，缓存管理器调用VMM来创建表示文件映射的段对象，这是对整个文件流。接下来，当一个进程试图访问属于这个文件流的数据的时候，缓存管理器动态映射这个文件流的视图到系统虚拟地址空间中为他保留的虚拟地址范围的适当的地方。注意因为保留给缓存管理器的地址范围是固定的，所以缓存管理器可能需要丢弃一些以前的视图才能创建新视图。

为了更好地理解NT缓存管理器在服务I/O请求中扮演的角色，让我们检查在响应用户读和写操作时候通常执行的步骤序列。

缓存读操作

假设一个用户应用程序发起一个读操作。这个读操作被NT I/O管理器传给文件系统（注：后面将看到，I/O管理器在很多情况下绕过文件系统。但是，为了简化，我们假设I/O操作首先被NT I/O管理器子系统传给文件系统驱动）。在图6-3图示响应读请求时执行的操作序列（使用NT缓存管理器提供的复制接口（copy interface）（注：在本章后面，我会讨论NT缓存管理器提供给其他系统组件的不同的接口方法，copy interface是四个可用接口中的一个））。

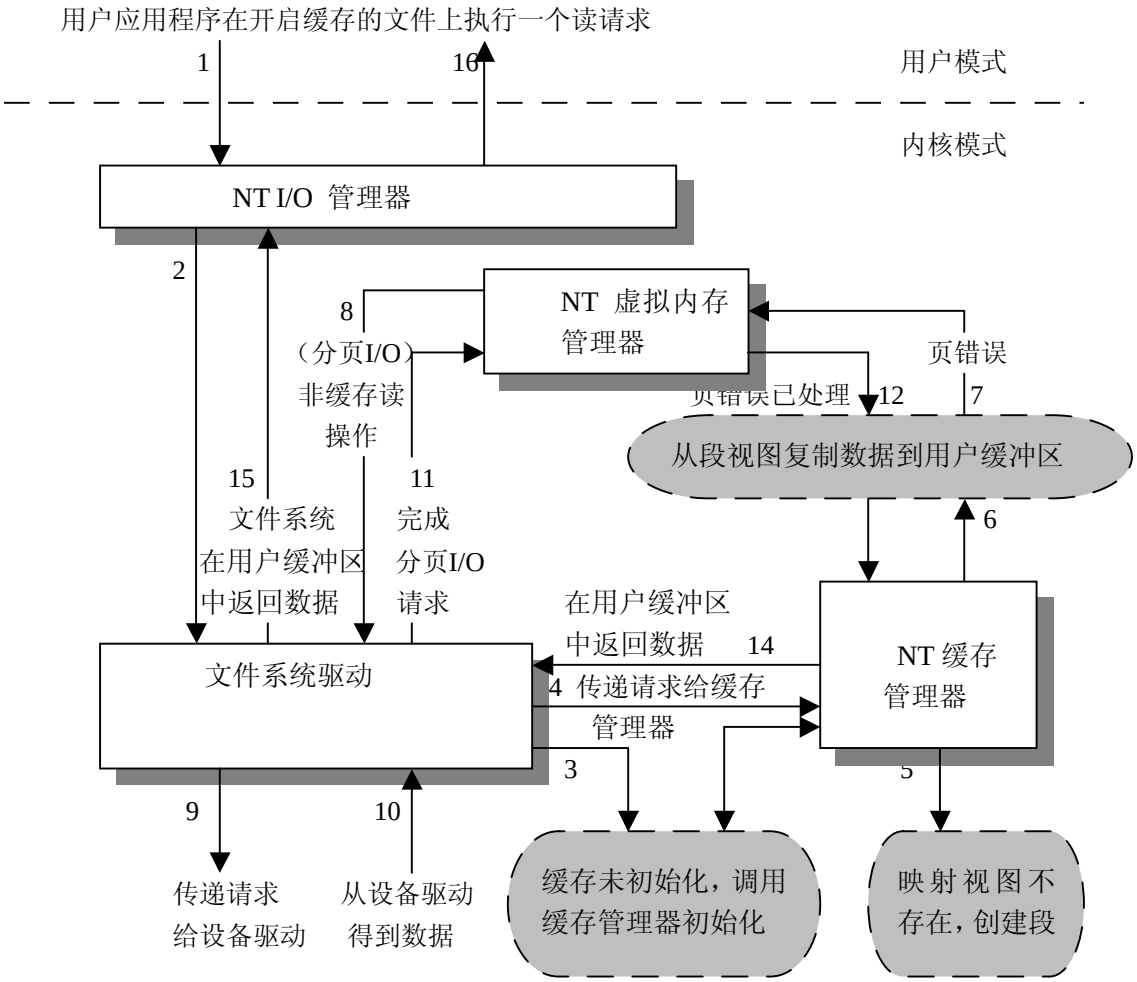


图6-3 对缓存文件执行读请求的执行步骤序列

下面给出图中列出的每个步骤的解释，注意图中的箭头表示控制流向。

1， 用户应用程序执行读操作，导致控制传递给内核中的I/O管理器。

2， I/O管理器用IRP把读请求发送给适当的文件系统驱动。用户缓冲区可以映射到系统虚拟地址空间中，或者I/O管理器可以分配MDL来表示这个缓冲区然后锁定相关的页面，或者I/O管理器可以直接传递未修改的缓冲区地址。在第三部分你会看见文件系统控制这些I/O管理器执行的操作。

3， 文件系统驱动收到读请求，且注意到是对一个以缓存访问打开的文件操作。如果这个文件的缓存还没有初始化，文件系统驱动调用缓存管理器来初始化这个文件的缓存。然后

缓存管理器请求VMM创建一个这个文件的文件映射（段对象）。

4， 文件系统驱动传递读请求给NT缓存管理器，使用CcCopyRead缓存管理器调用。缓存管理器现在负责所有需要传递数据到用户缓冲区的步骤。

5， 缓存管理器解释这个数据结构来决定用户请求的文件数据字节范围是否在映射的视图中。如果视图不存在，缓存管理器就创建一个。

6， 缓存管理器执行一个简单的从映射视图到用户缓冲区的复制内存的操作。

7， 如果请求的数据不在文件映射视图的当前物理页面中，就发生一个页错误，然后控制转到VMM。

8， VMM为页错误分配将要用来包含请求数据的物理页面。然后通过NT I/O管理器向文件系统驱动发出一个非缓存的分页I/O读取操作。虽然上图中没有指出分页I/O请求是通过NT I/O管理器，但是实际上是这样做的。

9， 接收到非缓存的读请求后，文件系统驱动创建适当的I/O请求来从辅助存储器介质取得数据，然后发送给低层驱动。

10，文件系统下面的设备驱动从辅助存储器（或者网络）取得数据然后完成这个请求。

11，文件系统驱动完成NT VMM的分页I/O请求。

12，这说明返回到重新执行的页错误中。

13，缓存管理器完成文件数据从映射视图到用户缓冲区的复制操作。这时候，这个复制操作应当能在不产生页错误的情况下完成（从理论上说，刚被放进内存的那个页面是有可能发生一个重复的页错误的，但是实际上几乎不会发生）。

14，缓存管理器在缓存数据复制到用户缓冲区后返回控制给文件系统驱动。注意，这个数据将在保留给缓存管理器的虚拟地址空间中被缓存（但是，这个数据可能在任何时候被NT VMM从系统内存中丢弃）。

15，文件系统驱动完成原来NT I/O管理器发送给他的IRP。

16，I/O管理器完成用户的请求。

缓存写操作

现在假设一个用户应用进程发出一个写操作。图6-4示例完成这个写请求的执行操作序列（使用缓存管理器的“复制”接口）（为了清晰的原因这个图被慎重的简化了。你将在第九章中看到，为了那种不完整的块传输，写操作可能导致文件系统实际上在执行写之前从磁盘读数据）。就象你看到的，操作序列和读操作类似。下面给出图中列出的每个步骤的解释：

1， 用户应用程序执行写操作，导致控制传递给内核中的I/O管理器。

2， I/O管理器用IRP把写请求发送给适当的文件系统驱动。和读操作一样，用户缓冲区可以映射到系统虚拟地址空间中，或者I/O管理器可以分配MDL来表示这个缓冲区然后锁定相关的页面，或者I/O管理器可以直接传递未修改的缓冲区地址。

3， 文件系统驱动注意到是对一个以缓存访问打开的文件操作。如果这个文件的缓存还没有初始化，文件系统驱动调用缓存管理器来初始化这个文件的缓存。然后缓存管理器请求VMM创建一个这个文件的文件映射（段对象）。

4， 文件系统驱动传递写请求给NT缓存管理器，使用CcCopyWrite缓存管理器调用。这是缓存管理器可用的复制接口的一部分。

5， 缓存管理器解释这个数据结构来决定用户请求修改的文件数据字节范围是否在映射的视图中。如果视图不存在，缓存管理器就创建一个。

6， 缓存管理器执行一个简单的从用户缓冲区到映射视图相联系的虚拟地址范围的复制内存的操作。

7， 如果请求修改的虚拟地址范围不在文件映射视图的当前物理页面中，就发生一个页错误，然后控制转到VMM。

8， VMM为页错误分配将要用来包含请求数据的物理页面。在图6-4中，假设整个页面都将被用户覆盖。在这种情况下，缓存管理器或者VMM都不需要在修改这些数据之前预先从磁盘读取已有的数据。但是如果是页面的一部分被修改，这个页错误将导致在允许修改数据之前，VMM发出分页I/O读操作。这说明导致页错误再次发生。

9， 缓存管理器完成从用户缓冲区到映射视图相联系的虚拟地址范围的复制内存的操作。

10，缓存管理器现在把控制返回给文件系统驱动，注意现在用户数据还在系统内存中，没有被写到辅助存储器上。缓存管理器将在以后执行实际的传输数据到辅助存储器的操作（不管是缓存管理器的延迟写者组件还是VMM的修改页写者都可能开始到辅助存储器的写操作。还有可能因为用户请求刷新系统内存缓冲区或者文件系统驱动（比如一个清除操作）开始的刷新都可能触发到磁盘的写操作。延迟写者组件会在下一章详细讨论）。

11，缓存管理器完成请求。

12，文件系统驱动完成原来NT I/O管理器发送给他的IRP。

13，I/O管理器完成用户的写请求。

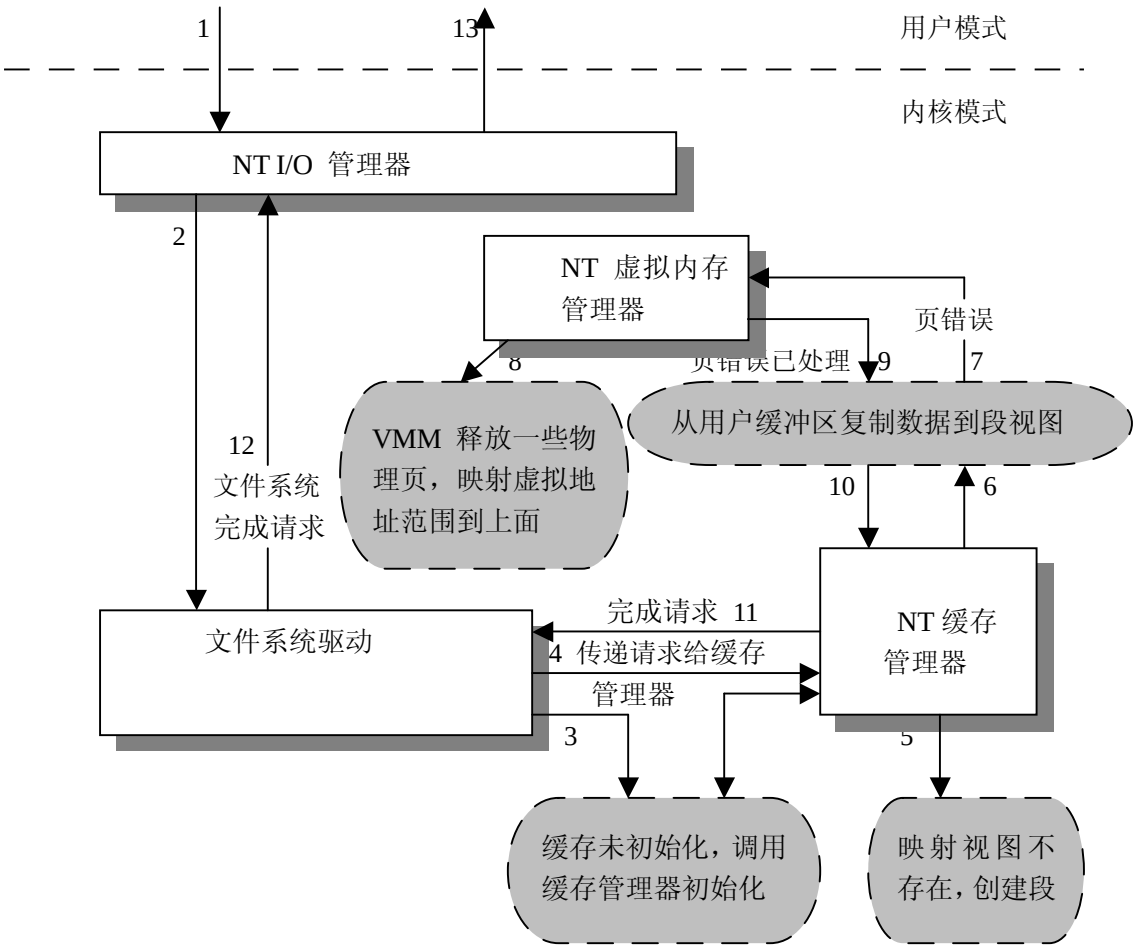


图6-4 对缓存文件执行写请求的执行步骤序列

缓存管理器接口

现在我们已经浏览了文件系统通常怎样使用缓存，我们再看看系统组件能够使用NT缓存管理器的不同途径。WINDOWS NT操作系统中的文件系统驱动和其他组件可以使用缓存管理器提供的四组接口例程得到服务。第一组接口例程支持基本文件流访问和操作，而其他的三组可以不同的方法访问系统缓存服务。
NT缓存管理器提供的四组接口是文件流操作函数，复制接口，MDL接口和销连接接口（pinning）。

文件流操作函数
NT缓存管理器提供初始化文件流缓存操作，刷新缓存数据到磁盘（按要求），修改文件大小，清除缓存数据，清0文件数据，记录文件流（一些文件系统（例如NTFS文件系统）使用一种叫做“记录到（logging to）”的方法来提供快速恢复和确保系统崩溃（或者未预期的关闭）的时候数据完整。这些文件系统需要确保一些操作序列，在其中记录项和文件元数据（数据）被写入磁盘。NT缓存管理器用下面列出的例程为这些文件系统驱动提供支持），还有其他通用的维护函数支持。NT缓存管理器在这个组中提供的函数如下：

- CcInitializeCacheMap
- CcUninitializeCacheMap
- CcSetFileSizes
- CcPurgeCacheSection
- CcSetDirtyPageThreshold
- CcFlushCache
- CcZeroData
- CcGetFileObjectFromSectionPtrs
- CcSetLogHandleForFile
- CcSetAdditionalCacheAttributes
- CcGetDirtyPages
- CcIsThereDirtyData
- CcGetLsnForFileObject

复制接口
复制接口是缓存访问最简单的部分。使用缓存管理器的客户端模块可以利用这个接口来复制缓冲区中的字节范围到缓存文件流的特定虚拟字节偏移处或者从缓存文件流的特定虚拟字节偏移处复制字节范围到内存中的缓冲区。
这个接口包括开始预读的调用，支持节流写的调用。节流写允许缓存管理器的客户（通常是一个文件系统驱动）延迟某些写操作，如果系统运行在底可用或者未修改页面的情况下。这种情况会在一些程序持续以极快的速率修改数据，比延迟写者或者修改页写者把数据写到磁盘上或者通过网络传输到存储服务器的速率更快的时候发生。注意很可能磁盘或者网络不能保持和延迟写者或者修改页写者发出的写数据到磁盘或网络的I/O请求的速率一致。这样就会导致可用的，未修改页面的减少。
缓存管理器提供的属于这组接口的函数有下面这些：

- CcCopyRead/CcFastCopyRead
- CcCopyWrite/CcFastCopyWrite
- CcCanlWrite
- CcDeferWrite
- CcSetReadAheadGranularity

- CcScheduleReadAhead

MDL接口

MDL是内存管理器定义的不透明的数据结构，用于映射特定的虚拟地址范围到由一个或者多个页面组成的物理地址范围。缓存管理器的MDL结构允许通过DMA（直接内存访问）访问系统缓存（注：DMA允许设备控制器在系统内存和辅助存储设备之间直接传输数据。这个处理方法使数据传输简化，从而得到更好的性能）。组成MDL接口的例程返回一个MDL给调用者，包含在请求包含的字节范围，随后调用者可以用来从系统缓存中传进或者传出数据。

这个接口对需要直接访问系统缓存的内容的子系统是有用的。例如，网络文件服务器需要在 网络设备和缓存管理器的虚拟地址范围之间使用MDL接口DMA交换数据实现较高的性能。如果没有这个接口，网络驱动从系统缓存中传出数据可能需要先分配临时缓冲区，从系统缓存中复制数据到临时缓冲区，然后网络设备再执行传输，最后释放临时缓冲区。如果网络设备通过网络直接传输数据给系统缓存，就可以避免分配/释放临时缓冲区和复制操作。这实际上可以利用CcMdlRead 和 CcMdlReadComplete调用序列来完成（注：这里使用的术语很重要：CcMdlRead在客户端希望从系统缓存中读取然后写到网络或者磁盘的时候使用。CcPrepareMdlWrite在客户端希望从网络或者磁盘写数据到系统缓存的时候使用）。

注意这个接口和复制接口共享预读调用。组成MDL接口和复制接口的调用可以同时在一个文件流中使用。缓存管理器在这个接口中提供的函数如下：

- CcMdlRead
- CcMdlReadComplete
- CcPrepareMdlWrite
- CcMdlWriteComplete

这里要注意的重要的一点是，大多数缓存管理器例程（例如，CcMdlRead, CcCopyRead）执行数据传输作为他们提供功能的一部分。但是CcPrepareMdlWrite只是创建一个包含原始数据的MDL，随后调用者在调用CcMdlWriteComplete之前可以修改它。因此，虽然CcPrepareMdlWrite被缓存管理器调用的时候可能执行一些数据传输（从磁盘或者网络取得当前文件流的数据然后放入MDL描述的页面中），但是他更象一个授权者例程，允许调用者使用返回的MDL来传输新的数据。

销连接接口（pinning）

缓存管理器提供的这个接口可以用来执行两个任务：

- 映射数据到系统缓存中以便使用一个缓冲区指针直接访问
- 固定（锁）存放映射数据的物理页

为了能够使用一个缓冲区指针直接读数据，调用者也能够直接修改系统缓存中的数据。

当映射数据不再需要访问的时候，数据可以解除固定。这也将导致被锁定的页面被解锁，可以被其他用户使用。一旦数据被解锁，数据指针就不能再使用。

锁定数据通常是因为性能的原因，当文件系统驱动或其他系统组件需要频繁直接使用内存中的数据结构（或者和文件流关联的其他数据）的时候。这也被用于确保数据要访问的数据不从系统内存中删除。但是，锁定映射数据消耗物理内存，因此减少了其他系统组件可以使用的内存数量。

注意销连接接口现在不能和复制接口或者MDL接口联合使用。

这个接口常常被文件系统处理缓存的文件系统元数据的时候使用。由下面的函数组成：

- CcMapData
- CcPinMappedData
- CcPinRead

- CcSetDirtyPinnedData
- CcPreparePinWrite
- CcUnpinData
- CcUnpinDataForThread
- CcRepinBcb
- CcUnpinRepinnedBcb
- CcGetFileObjectFromBcb

上面的函数在第七章会更详细地说明。

缓存管理器的客户

下面的组件通常使用缓存管理器提供的接口。这些组件也叫做缓存管理器的客户。

- 文件系统驱动比如NTFS，FASTFAT，CDFS和其他使用缓存管理器的复制接口来执行每个用户文件数据的第三方文件系统。这允许更好的性能，因为一旦用户数据在系统内存中缓存，随后对数据的访问能够立即满足而不需要从辅助存储器中再次读取。

  文件系统驱动也使用缓存管理器来缓存文件系统元数据，包括卷结构，目录信息，磁盘自由空间的位图，和文件关联的扩展属性，以及其他类似信息。许多这样的结构被文件系统驱动固定在内存中。注意缓存管理器并不解释被缓存数据的类型；他只知道文件对象数据结构和和这个文件对象关联的数据流。

  文件系统驱动通常还使用混村管理器提供的预读和延迟写功能，虽然很可能某些高级的文件系统实现可能加入自己的支持预读或者延迟写操作。最后，所有的文件系统驱动必须使用缓存管理器提供的文件流操作函数来和缓存管理器交互。
- 网络重定向器类似于文件系统的实现；但是，这些模块通过网络从文件服务器得到数据，而不是直接从附加到本地系统的辅助存储器上取数据。这些组件通常在系统内存中缓存各种数据流来提供能够和本地文件系统媲美的非常快速的性能。

  网络重定向器通常使用缓存管理器提供的复制接口。他们也可以使用MDL接口来直接访问系统缓存。这些组件还使用缓存管理器提供的预读或者延迟写功能。为了在特定数据流上开始和结束缓存或者执行其他的缓存操作函数，网络重定向器要使用文件流操作函数。
- 网络文件服务器是缓存管理器的间接客户，因为他们使用本地文件系统来最终访问文件数据。这些驱动从不直接调用缓存管理器例程。文件服务器因为性能的原因通常实现为内核驱动。他们通过为他的请求服务的文件系统驱动使用复制接口。他们也通常使用MDL接口来直接访问系统缓存。因此文件服务器就使用缓存管理器的MDL接口。因为文件服务器不能直接调用缓存管理器，因此他们在发送给文件系统驱动的读/写IRP中使用特定标志来请求为文件流中特定虚拟地址范围创建MDL。当数据传输完成以后，文件服务器通知文件系统先前创建的MDL可以删除了。第九章包含一个文件服务器请求创建和删除系统缓存中的数据缓冲区的MDL的标志的说明。
- 过滤驱动或者其他为了特殊目的而使用NT文件系统驱动接口的驱动是缓存管理器的间接客户。考虑为存储在低速介质上如软磁盘或者光介质上的数据提供硬盘缓存的过滤驱动。这样的驱动使用本地文件系统来存储缓存信息。因此，过滤驱动是缓存管理器的间接客户，因为文件系统使用复制接口直接访问系统内存来支持过滤驱动。类似的，考虑提供HSM（分级存储管理）（注：HSM指有效管理使用配置由快速的昂贵的介质和低速但是廉价的介质组成可用存储，达到最小化存储数据每字节的花费，但数据在请求的使用又可用。通常，这要自动从快速的，昂贵的硬盘传输那些很少访问的数据到慢速的，廉价的介质上，比如软盘。当这些数据以后被访问的时候，驱动自动从软盘传输到硬盘。HSM还有其他的方面

特征但是已经超出了这里讨论的范围）功能的过滤驱动。这样的驱动必须从相关的快速存储设备比如一个快速硬盘上迁移数据到低速设备比如软盘。为了提高这个处理的速度，过滤驱动使用DMA直接从系统缓存传输数据到软盘，因此使用了缓存管理器的MDL接口（通过在发送给文件系统驱动的读/写IRP中使用特殊的标志）。在传输完成以后，过滤驱动通知文件系统先前创建的MDL可以删除了。

表 6-1 概括缓存管理器的客户使用他的不同接口的方式

表 6-1 缓存管理器的客户

	本地文件系统	网络重定向器	网络文件服务	过滤驱动
文件流操作	▲	▲		
复制接口	▲	▲	▲	▲
MDL接口		▲	▲	▲
销连接接口	▲			

一些重要数据结构

缓存管理器提供的服务被网络重定向器和文件系统驱动大量利用来为用户的I/O请求服务。下面描述的数据接口和域在正确理解缓存管理器提供的接口的时候很重要。

文件对象（file object）中的域

正如在第四章中说明的，每个创建或者打开的文件流有一个文件对象结构（类型FILE_OBJECT）被I/O管理器为他创建。虽然文件对象结构中大多数域由I/O管理器填充，但是网络重定向器和文件系统驱动这些相关文件流的I/O请求的接收者被要求填充某些特定的域。下面的三个域必须被初始化：

- FsContext域
- SectionObjectPointer域
- PrivateCacheMap域

通常在文件流打开或者创建的时候初始化；但是也可能网络重定向器和文件系统驱动会在这个文件流的缓存被第一次初始化之前延迟这个操作。

FsContext

如果缓存管理器被要求为一个打开的文件流（由文件对象结构表示）建立缓存，FsContext域被初始化为一个指向FSRTL_COMMON_FCB_HEADER结构类型的指针。这个结构定义如下：

```
typedef struct _FSRTL_COMMON_FCB_HEADER {
    CSHORT          NodeTypeCode;
    CSHORT          NodeByteSize;
    UCHAR           Flags;
    UCHAR           IsFastIoPossible;
    /*************************************************/
    // The following two fields are only present in Version 4.0+ of the
    // the Windows NT operating system.
    // Second Flags Field.
    UCHAR           Flags2;
```

```
    // The following reserved field should always be 0.
    UCHAR           Reserved;
    //
    /*********************************************/
    PERESOURCE      Resource ;
    PERESOURCE      PagingIoResource;
    LARGE_INTEGER AllocationSize;
    LARGE_INTEGER FileSize;
    LARGE_INTEGER ValidDataLength;
} FSRTL_COMMON_FCB_HEADER;
```

上面的结构也被叫做CommonFCBHeader结构。文件系统或者网络驱动必须从非分页核心内存中分配他们。就象在第九章中看见的，每个文件流在内存中用一个文件控制块（FCB）结构唯一表示。

注意：对于UNIX背景的读者来说，文件控制块就类似于UNIX中的表示文件或者目录的V接点结构。

虽然对同一文件的同时的多个打开操作可能导致多个文件对象结构被创建，但是这个文件只有一个唯一的FCB，所有的文件对象结构都必须指向他。

类似的，每个文件流只能存在一个CommonFCBHeader结构。因此，在文件系统驱动或者网络驱动实现中把CommonFCBHeader结构作为FCB结构的一部分来分配来表示文件流就并不罕见。但是，注意文件系统驱动并不要求一对一的分配CommonFCBHeader结构和FCB结构的逻辑关系。

在CommonFCBHeader结构最前面两个域，NodeTypeCode和NodeByteSize没有被缓存管理器使用。组成这个结构的域描述如下。注意许多域要求理解在后面的章节（特别是9-11章）中说明的概念；当所有这些要求的概念出现的时候这些域的初始化会被重新考察：

Flags

CommonFCBHeader结构有两个同步类型ERESOURCE结构的指针。PagingIoResource由修改页写者线程获得。通过在Flags域中设置适当的值，文件系统驱动或者网络重定向器驱动允许指示MPW线程MainResource（见下面）应该被请求来代替PagingIoResource。在第十一章中会讨论为什么文件系统或者网络重定向器可以设置这样一个标志。

Flags2

这个域是在NT 4.0操作系统中加入的。在本书后面讨论的，他可能被一个FSD用于指定延迟写操作不要在一个缓存文件流上执行。但是，如果域的FSRTL_FLAG2_DO_MODIFIED_WRITE（定义为0x01）标志被设置，缓存管理器将忽略FDS请求禁止延迟写操作而为文件流执行延迟写I/O。

IsFastIoPossible

因为性能的原因，I/O管理器试图绕过缓存文件的文件系统驱动或者网络重定向器来直接从缓存管理器取得文件数据。这个处理叫做快速I/O操作。IsFastIoPossible域允许文件系统驱动或者网络重定向器控制对于特定的文件流是否允许快速I/O操作。这个域的内容由文件系统驱动或者网络重定向器设置，可以是下面三个枚举值之一：FastIoIsNotPossible，FastIoIsPossible，或者 FastIolsQuestionable。

Resource和PagingIoResource

访问和文件流关联的数据必须使用结构同步。这是一个对于文件系统驱动和网络重定向器

的要求，为了能够和缓存管理器和内存管理器组件正确的交互。

这些资源的内存必须由文件系统或者网络重定向器从非分页池中分配，而且CommonFCBHeader中的这个域必须初始化为指向这个分配的结构。

因为这是资源提供共享的读和互斥写的语意，缓存管理器期望文件系统驱动或者网络重定向器互斥地得到MainResource来同步对文件流的所有修改操作。类似的，读操作可以共享地得到MainResource来同步。

AllocationSize

这是为文件流分配的在磁盘上的存储空间的实际大小。通常，这是多个介质扇区大小或者文件系统串（cluster）大小（注：在辅助存储器上分配空间的单位叫扇区（sector），每个扇区由固定的字节数组成，例如，一个扇区可以等于512字节。为了避免碎片，某些文件系统驱动用串（cluster）来作为分配空间的单位，这里的每个串（cluster）等于多个扇区。例如，一个串（cluster）可以等于8个物理扇区）。这个域必须由文件系统驱动或者网络重定向器初始化为适当的值。随后，每当这个值改变的时候缓存管理器必须得到通知。在下一章你会看到文件系统怎样通知缓存管理器分配尺寸的改变。

FileSize

这是文件呈现给用户的大小；这个值指出这个文件流中包含的字节数。任何超过这个值的读操作将返回一个文件结尾(STATUS_END_OF_FILE)的错误消息给应用进程。任何超出这个值的读操作将被裁剪为这个值。

例如，如果FileSize是45字节而用户希望从文件流的偏移量为40的地方读取30个字节，那么实际上文件系统驱动（或者缓存管理器）只会返回5个字节给用户。但是，如果还是这个用户希望在偏移量45的地方读30个字节（假设文件开始处的偏移量为0），那么就会返回一个STATUS_END_OF_FILE错误给用户。

文件系统驱动或者网络重定向器初始化为适当的值。随后，每当这个值改变的时候缓存管理器必须得到通知。

ValidDataLength

考虑这样的情况：一个文件流的FileSize是100字节。但是这个文件流只有开始的10字节的有效数据，而后面的90字节没有被任何进程写入。那么这个文件流的ValidDataLength就被设置为10。任何超过这个值的访问企图将自动得到0个返回字节。这有助于避免不必要的到磁盘的I/O操作，还有助于提高数据安全（因为某些先前的文件流存储在介质上的旧信息不会不小心的返回给用户）。

很少文件系统在磁盘上存储维护一个ValidDataLength概念来和一个文件流关联。NTFS和HPFS支持这个概念。但是，不管文件系统时候支持有效数据长度的概念，缓存管理器都期望文件系统驱动或者网络重定向器把他初始化为适当的值。

SectionObjectPointer（段对象指针）

这个域被初始化为指向类型SECTION_OBJECT_POINTERS的指针。这个结构必须由文件系统驱动或者网络重定向器从非分页的核心内存中分配，然后由VMM和缓存管理器共享。用来存储文件流的文件映射和缓存相关的信息。这个结构有下面的格式：

```
typedef struct _SECTION_OBJECT_POINTERS {
    PVOID       DataSectionObject;
    PVOID       SharedCacheMap;
    PVOID       ImageSectionObject;
} SECTION_OBJECT_POINTERS;
typedef SECTION_OBJECT_POINTERS *PSECTION_OBJECT_POINTERS;
```

任何时候一个文件流只有一个这样的结构和他关联。但是，完全可能，非常有可能的是在用户打开的文件有多个文件对象，每个都代表一个文件流的实例，同时存在在一个接点上的。这种情况下，在每个文件对象结构中的所有SectionObjectPointer域必须用单独分配的这个类型的结构的地址来初始化。因此，这个结构通常和文件流的FCB关联。

在分配上，缓存管理器的客户有责任清除SECTION_OBJECT_POINTERS数据结构中的所有的域。在清除了这个结构后，客户不需要关心任何这些域的操作。包含在这个结构中的域的一个说明如下（记住只有VMM或者缓存管理器能够操作这些域）：

DataSectionObject

VMM用这个指针来指向一个为文件流创建的数据段对象的内部数据结构。因此，这个域由VMM在文件流的缓存被初始化的时候初始化。

SharedCacheMap

缓存管理器创建一个叫做缓存位图的数据结构来跟踪特定数据流的映射视图。这个域由缓存管理器在文件流的缓存被初始化的时候用一个SharedCacheMap结构（会在 这一接后面描述）的地址初始化。

ImageSectionObject

VMM在每当这个文件流的一个印象段创建的时候用一个私有数据结构的地址初始化这个域。

PrivateCacheMap（私有缓存映射）

缓存管理器的客户被期望为每个文件对象结构初始化这个域为NULL。注意对于一个文件流可以在内存中同时存在多个文件对象结构。也有可能，缓存可能已经被一些文件对象结构，但不是全部，初始化了。

我们知道文件系统驱动，网络重定向器和其他的缓存管理器的客户和缓存管理器合作来向所有用户提供一个数据的一致的视图；不管是用缓存途径访问数据还是不是的线程都一样。对于文件系统或者网络重定向器来说确定一个文件流的缓存是否已经使用一个文件对象初始化的唯一的途径是检查PrivateCacheMap域是否为非空。这个检查必须在得到MainResource以后执行，不管是共享得到还是互斥得到。

文件流是否已经被特定文件对象缓存的信息不能被客户在其他地方维护。这是因为缓存保留强行结束这个文件流关联的一些或者全部文件对象缓存的权利。因此，象前面提到的，PrivateCacheMap域为非空是客户检查的缓存当前被文件对象初始化的唯一可靠指示。

缓存位图（Cache Maps）

缓存管理器必须维护他帮助缓存数据的每个文件流的信息。这个信息用缓存位图来维护。对于每个文件流，缓存管理器分配一个共享缓存位图（Shared Cache Map）结构来保存这个文件流和和这个文件流关联的其他信息的映射视图。这个共享缓存位图结构在这个文件流的缓存在文件系统驱动或者网络重定向器请求首次初始化的时候分配。

另外，共享缓存位图结构对于每个文件流是唯一的，因此只有在首次为文件流建立缓存的时候分配，每次一个客户用一个特定的文件对象结构发出一个请求初始化缓存的时候，缓存管理器就分配一个私有缓存位图（Private Cache Map）结构。缓存管理器用这个结构来标记缓存已经使用特定文件对象初始化的事实。其中还包含缓存管理器用于预读控制的信息和其他的数据。

注意，共享缓存位图（Shared Cache Map）结构和私有缓存位图（Private Cache Map）结构都是由缓存管理器分配和维护。

缓冲区控制块（Buffer Control Blocks）

前面提到的缓存管理器提供的接口之一是销连接接口。缓存管理器的客户要使用这个接口必须使用缓冲区控制块结构。这个结构分为两部分：公共BCB，导出给缓存管理器的客户，还有私有BCB由缓存管理器内部使用。

公共BCB定义如下：

```
typedef struct _PUBLIC_BCB {
    CSHORT          NodeTypeCode;
    CSHORT          NodeByteSize;
    ULONG           MappedLength;
    LARGE_INTEGER   MappedFileOffset;
} PUBLIC_BCB, *PPUBLIC_BCB;
```

公共BCB是非常简单的，充当缓存管理器的客户的上下文用来销连接数据和解除数据的销连接。当文件系统驱动或者网络重定向器请求为文件流销连接数据的请求成功返回，缓存管理器就返回一个BCB结构的指针。这个BCB结构的内存由缓存管理器分配。

文件系统驱动以一个不透明的方式使用BCB结构的指针：MappedLength和MappedFileOffset向客户提供实际偏移量的信息，被销连接在内存中数据开始的地方和数据的字节数。

随后客户请求重新销连接或者解除内存结构的销连接必须使用这个BCB指针作为环境来执行，这个BCB指针被返回给缓存管理器。在下一章将说明，返回给缓存管理器的这个BCB很可能会在作为环境传进的时候被改变。因此，客户一定不要试图复制返回的BCB结构然后使用。BCB的私有部分没有被缓存管理器暴露出来。

文件大小的考虑

有三个不同的文件大小值：

- 文件流的AllocationSize代表磁盘上为这个文件流保留的实际磁盘空间，这是由文件所在的介质的最小分配单位组成的。
- 文件流的FileSize是一个读操作超过这个值就会返回一个文件结尾错误的值（注：注意完全有可能在某些文件系统实现中，FileSize可能比AllocationSize大。这会在文件系统驱动支持稀疏文件实现的时候发生。WINDOWS NT当前提供的文件系统中没有支持稀疏文件的）。
- ValidDataLength是一个文件流中包含的有效数据的数量。
  进程试图读任何超过这个值（小于FileSize）的访问将得到0的结果。

对于缓存管理器的客户有两个重要的考虑是谁改变了这里的一个或者多个文件大小。

所有客户必须遵循的一个基本的规则是改变AllocationSize和FileSize必须和其他读/写请求同步，而且缓存管理器必须立即得到任何的改变。

改变FileSize和读/写请求的同步是通过确保在执行这样的改变的时候文件流的FCB被互斥的得到来完成的。在改变任一文件大小值之前，PagingIoResource和MainResource都必须被互斥的得到。用文件的FCB调用CcSetFileSizes将使得缓存管理器得到通知。

上面的规则后面的道理很简单：文件系统驱动或者网络重定向器常常被I/O管理器绕开而直接和文件流交换数据，通过快速I/O途径来使用缓存管理器。在这种情况下，如果缓存管理器没有被正确的通知FileSize改变。可能会给试图执行数据传输应用程序返回无效结果（注：在某些情况下，当文件被剪裁的时候，缓存管理器或者内存管理器可能拒绝操作的执行。这个主题将在第十章更详细的说明，但是，重要的一点是文件系统或者网络重定向器必须和缓存管理器模块协调文件流的文件大小的改变）。例如，如果当前文件大小被应用程序扩大而缓存管理器没有被通知现在的文件大小，很可能应用程序试图读取超过旧的文件结尾偏移量的信息的时候收到

STATUS_END_OF_FILE错误。

第二个重要的一点是改变FileSize通常不需要和分页I/O读和写请求同步。注意分页I/O请求通常从延迟写者或者修改块写者组件发出，或者由用户直接操作映射文件而产生。分页I/O请求将在9-11章更详细地处理，读者应该认识到下面几点：

- 分页I/O读从当前文件结尾之后开始的以错误STATUS_END_OF_FILE结束。
- 分页I/O读请求从当前文件结尾之前开始而超过了当前文件结尾的将被剪裁到当前文件结尾偏移量处。但是，客户必须小心地设置要写入的字节数和最初要求的字节数一样（虽然实际上没有I/O被执行）。
- 分页I/O写请求从当前文件结尾之后开始的必须被文件系统或者网络重定向器空处理并返回STATUS_SUCCESS。

但是ValidDataLength的概念很少被盘上文件系统驱动支持。如果文件系统支持并在磁盘上记录ValidDataLength值，他应该在文件流第一次打开的时候用当前值来初始化CommonFCBHeader。随后，缓存管理器在这个值改变的时候通知文件系统驱动，然后文件系统驱动或者网络重定向器可以在磁盘上记录修改了值。注意缓存管理器可能直接被I/O管理器调用来处理用户的写请求，这样可能导致有效数据长度的改变。

缓存管理器用SetFilelnformation IRP来通知客户有效数据长度的改变。这个IRP和缓存管理器用来通知客户的方法将在第十章详细讨论。

如果客户在磁盘上不支持有效数据长度的概念，因此而不希望从缓存管理器收到这个值改变的通知，那么客户必须这样初始化ValidDataLength域：底32位必须初始化为0xFFFFFFFF而高32位必须初始化为0x7FFFFFFF。

即使客户不在磁盘上记录有效数据长度，当文件流保持打开的时候维护有效数据的长度可能对客户也是有用的。考虑一下用户进程扩展文件长度的情形，随后，用户进程发出一个超过了旧文件结尾偏移量的写请求。这个请求将会交给缓存管理器，缓存管理器首先试图在产生一个页错误来来得到页面来接收用户的数据。这个页错误最终需要文件系统驱动或者网络重定向器来完成。如果文件系统驱动在内存中维护有效数据长度的概念，他可能意识到没有读操作的要求，因此这个文件流仅仅需要被扩展，那么立刻就可以返回0来完成页错误请求。