# 1.先搞清楚一个概念

## 1.什么是内存映射文件？内存映射文件有什么用？

内存映射文件是一种内存管理功能，它允许应用程序将磁盘上的文件内容映射到进程的地址空间。这样做的好处是可以像访问普通内存那样访问文件数据，通过指针直接读写文件，而不是调用传统的文件I/O操作（如read和write）。这种方式可以提高处理大型文件时的性能，特别是需要频繁随机访问文件部分内容的场景，常见的如mmap：

### 使用内存映射文件处理大文件

1. **创建内存映射**: 在Unix-like系统中，可以使用mmap函数来创建内存映射；在Windows系统中，可以使用CreateFileMapping和MapViewOfFile等函数。
2. **访问数据**: 一旦文件被映射到内存，你就可以使用指针来访问文件的内容，就像它是一个巨大的数组或者是连续内存块。
3. **同步和清理**: 当对映射区域进行了写操作后，你可能需要确保数据刷新到磁盘。在Unix系统中，可以使用msync来同步；在完成操作后，使用munmap撤销映射。在Windows系统中，则使用UnmapViewOfFile来撤销映射，使用FlushViewOfFile来同步数据。

## **2.应用场景**

### 1.使用内存映射文件来读写文件

### 2.系统加载exe文件或者dll文件的时候也需要使用内存映射文件

### 3.多个进程之间使用内存映射文件来交换数据

# 2. 创建文件映射对象

映射文件的第一步是通过调用 [**CreateFile**](https://learn.microsoft.com/zh-cn/windows/win32/api/fileapi/nf-fileapi-createfilea) 函数打开文件。 若要确保其他进程无法写入映射的文件部分，应以独占访问权限打开文件。 此外，文件句柄应保持打开状态，直到进程不再需要文件映射对象。 获取独占访问权限的一种简单方法是在 **CreateFile** 的 fdwShareMode 参数中指定零。 **CreateFileMapping** 函数使用 [**CreateFileMapping**](https://learn.microsoft.com/zh-cn/windows/desktop/api/WinBase/nf-winbase-createfilemappinga) 返回的句柄来创建文件映射对象。

[**CreateFileMapping**](https://learn.microsoft.com/zh-cn/windows/desktop/api/WinBase/nf-winbase-createfilemappinga) 函数返回文件映射对象的句柄。 [创建文件视图](https://learn.microsoft.com/zh-cn/windows/win32/memory/creating-a-file-view)时将使用此句柄，以便可以访问共享内存。 调用 **CreateFileMapping** 时，可以指定对象名称、要从文件映射的字节数，以及映射内存的读/写权限。 调用 **CreateFileMapping** 的第一个进程创建文件映射对象。 为现有对象调用 **CreateFileMapping** 的进程会收到现有对象的句柄。 可以通过调用 [**GetLastError**](https://learn.microsoft.com/zh-cn/windows/win32/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数来判断对 **CreateFileMapping** 的成功调用是否创建或打开了文件映射对象。 **GetLastError** 将 **NO\_ERROR** 返回到创建进程， **并将ERROR\_ALREADY\_EXISTS** 返回到后续进程。

如果访问标志与 [**CreateFile**](https://learn.microsoft.com/zh-cn/windows/desktop/api/WinBase/nf-winbase-createfilemappinga) 函数打开文件时指定的标志冲突， [**CreateFileMapping**](https://learn.microsoft.com/zh-cn/windows/win32/api/fileapi/nf-fileapi-createfilea) 函数将失败。 例如，若要读取和写入文件，请执行以下操作：

* 在 [**CreateFile**](https://learn.microsoft.com/zh-cn/windows/win32/api/fileapi/nf-fileapi-createfilea) 的 fdwAccess 参数中指定**GENERIC\_READ**和**GENERIC\_WRITE**值。
* 在 [**CreateFileMapping**](https://learn.microsoft.com/zh-cn/windows/desktop/api/WinBase/nf-winbase-createfilemappinga) 的 fdwProtect 参数中指定**PAGE\_READWRITE**值。

创建文件映射对象不会提交物理内存，只保留它。

## 文件映射大小

文件映射对象的大小与所映射的文件的大小无关。 但是，如果文件映射对象大于文件，系统会在 [**CreateFileMapping**](https://learn.microsoft.com/zh-cn/windows/desktop/api/WinBase/nf-winbase-createfilemappinga) 返回之前扩展该文件。 如果文件映射对象小于文件，则系统仅映射文件中的指定字节数。

[**CreateFileMapping**](https://learn.microsoft.com/zh-cn/windows/desktop/api/WinBase/nf-winbase-createfilemappinga) 的 dwMaximumSizeHigh 和 dwMaximumSizeLow 参数允许您指定要从文件映射的字节数：

* 如果不希望文件大小更改 (例如，将只读文件映射) 时，请调用 [**CreateFileMapping**](https://learn.microsoft.com/zh-cn/windows/desktop/api/WinBase/nf-winbase-createfilemappinga) 并为 dwMaximumSizeHigh 和 dwMaximumSizeLow 指定零。 执行此操作会创建与文件完全相同的文件映射对象。 否则，必须计算或估计完成文件的大小，因为文件映射对象的大小是静态的;创建后，无法增加或减少其大小。 尝试以这种方式映射长度为零的文件失败，错误代码 **为 ERROR\_FILE\_INVALID**。 程序应测试长度为零的文件，并拒绝此类文件。
* 由命名文件支持的文件映射对象的大小受磁盘空间限制。 文件视图的大小限制为未保留虚拟内存的最大可用连续块。 这最多为 2 GB 减去进程已保留的虚拟内存。

所选文件映射对象的大小控制了使用内存映射“查看”到文件的距离。 如果创建大小为 500 Kb 的文件映射对象，则无论文件大小如何，您都只能访问该文件的前 500 Kb。 由于创建更大的文件映射对象不会花费任何系统资源，因此创建文件映射对象的大小 (将 [**CreateFileMapping**](https://learn.microsoft.com/zh-cn/windows/desktop/api/WinBase/nf-winbase-createfilemappinga) 的 dwMaximumSizeHigh 和 dwMaximumSizeLow 参数设置为零) 即使不希望查看整个文件。 系统资源的成本来自创建视图并访问它们。

# 3.学习大纲

|  |
| --- |
|  |

## 注意：操作过程分6个步骤

## 第一步，使用创建文件api

|  |
| --- |
|  |

## 第二步，使用内存映射API

|  |
| --- |
|  |

### CreateFileMapping语法详解

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HANDLE CreateFileMappingA(  [in] HANDLE hFile,  [in, optional] LPSECURITY\_ATTRIBUTES lpFileMappingAttributes,  [in] DWORD flProtect,  [in] DWORD dwMaximumSizeHigh,  [in] DWORD dwMaximumSizeLow,  [in, optional] LPCSTR lpName  ); 参数 [in] hFile  要从中创建文件映射对象的文件的句柄。  必须使用与 *flProtect* 参数指定的保护标志兼容的访问权限打开文件。 这不是必需的，但建议打开要映射的文件进行独占访问。 有关详细信息，请参阅 [文件安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-security-and-access-rights)。  如果 *hFile***是INVALID\_HANDLE\_VALUE**，则调用进程还必须在 *dwMaximumSizeHigh* 和 *dwMaximumSizeLow* 参数中指定文件映射对象的大小。 在此方案中， **CreateFileMapping** 会创建一个指定大小的文件映射对象，该对象由系统分页文件而不是文件系统中的文件提供支持。  [in, optional] lpFileMappingAttributes  指向 [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85)) 结构的指针，该结构确定是否可由子进程继承返回的句柄。 **SECURITY\_ATTRIBUTES** 结构的 **lpSecurityDescriptor** 成员为新的文件映射对象指定安全描述符。  如果 *lpFileMappingAttributes* 为 **NULL**，则无法继承句柄，并且文件映射对象将获取默认的安全描述符。 访问控制列出了来自创建者的主要令牌或模拟令牌的文件映射对象的默认安全描述符中 (ACL) 。 有关详细信息，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/file-mapping-security-and-access-rights)。  [in] flProtect  指定文件映射对象的页保护。 对象的所有映射视图都必须与此保护兼容。  此参数的取值可为下列值之一：  展开表   |  |  | | --- | --- | | **值** | **含义** | | **PAGE\_EXECUTE\_READ**  0x20 | 允许映射视图，以便进行只读、写入时复制或执行访问。  必须使用**GENERIC\_READ**创建 *hFile* 参数指定的文件句柄，**并GENERIC\_EXECUTE**访问权限。  **Windows Server 2003 和 Windows XP：**此值在 Windows XP SP2 和 Windows Server 2003 SP1 之前不可用。 | | **PAGE\_EXECUTE\_READWRITE**  0x40 | 允许映射视图，以便进行只读、写入时复制、读/写或执行访问。  必须使用**GENERIC\_READ**、**GENERIC\_WRITE**和**GENERIC\_EXECUTE**访问权限创建 *hFile* 参数指定的文件句柄。  **Windows Server 2003 和 Windows XP：**此值在 Windows XP SP2 和 Windows Server 2003 SP1 之前不可用。 | | **PAGE\_EXECUTE\_WRITECOPY**  0x80 | 允许映射视图，以便进行只读、写入时复制或执行访问。 此值等效于 **PAGE\_EXECUTE\_READ**。  必须使用 **GENERIC\_READ** 创建 *hFile* 参数指定的文件句柄，**并GENERIC\_EXECUTE**访问权限。  **Windows Vista：**此值在 Windows Vista SP1 中之前不可用。  **Windows Server 2003 和 Windows XP：**不支持此值。 | | **PAGE\_READONLY**  0x02 | 允许映射视图，以便进行只读或写入时复制访问。 尝试写入特定区域会导致访问冲突。  必须使用**GENERIC\_READ**访问权限创建 *hFile* 参数指定的文件句柄。 | | **PAGE\_READWRITE**  0x04 | 允许映射视图，以便进行只读、写入时复制或读/写访问。  必须使用 **GENERIC\_READ** 创建 *hFile* 参数指定的文件句柄，**GENERIC\_WRITE**访问权限。 | | **PAGE\_WRITECOPY**  0x08 | 允许映射视图，以便进行只读或写入时复制访问。 此值等效于 **PAGE\_READONLY**。  必须使用**GENERIC\_READ**访问权限创建 *hFile* 参数指定的文件句柄。 |     应用程序可以通过将以下一个或多个属性与前面的页面保护值组合来为文件映射对象指定以下一个或多个属性。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **SEC\_COMMIT**  0x8000000 | 如果文件映射对象由操作系统分页文件提供支持， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) ，则指定在将文件的视图映射到进程地址空间时，将提交而不是保留整个页面范围。 系统必须有足够的可提交页面来保存整个映射。 否则， **CreateFileMapping** 将失败。  对于由可执行图像文件或数据文件支持的文件映射对象， (*hfile* 参数是文件) 句柄，则此属性不起作用。  **SEC\_COMMIT** 不能与 **SEC\_RESERVE**结合使用。  如果未指定任何属性，则假定 **SEC\_COMMIT** 。 | | **SEC\_IMAGE**  0x1000000 | 指定 *hFile* 参数指定的文件是可执行映像文件。  **SEC\_IMAGE** 属性必须与页面保护值（如**PAGE\_READONLY**）结合使用。 但是，此页面保护值对可执行映像文件的视图没有影响。 对可执行映像文件视图的页面保护由可执行文件本身确定。  没有其他属性对 **SEC\_IMAGE**有效。 | | **SEC\_IMAGE\_NO\_EXECUTE**  0x11000000 | 指定 *hFile* 参数指定的文件是不会执行的可执行映像文件，并且加载的映像文件不会运行强制完整性检查。 此外，映射使用 **SEC\_IMAGE\_NO\_EXECUTE** 属性创建的文件映射对象的视图不会调用使用 [PsSetLoadImageNotifyRoutine](https://learn.microsoft.com/zh-cn/windows-hardware/drivers/ddi/content/ntddk/nf-ntddk-pssetloadimagenotifyroutine) 内核 API 注册的驱动程序回调。  **SEC\_IMAGE\_NO\_EXECUTE** 属性必须与**PAGE\_READONLY**页保护值结合使用。 SEC\_IMAGE\_NO\_EXECUTE中没有其他属性**有效。**  **Windows Server 2008 R2、Windows 7、Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP：**在Windows Server 2012和Windows 8之前，不支持此值。 | | **SEC\_LARGE\_PAGES**  0x80000000 | 使大型页面可用于操作系统分页文件支持的文件映射对象， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) 。 由可执行图像文件或数据文件支持的文件映射对象不支持此属性， (*hFile* 参数是可执行映像或数据文件的句柄) 。  文件映射对象的最大大小必须是 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数返回的大页最小大小的倍数。 否则， **CreateFileMapping** 将失败。 映射使用 **SEC\_LARGE\_PAGES** 创建的文件映射对象的视图时，基址和视图大小也必须是最小大页面大小的倍数。  **SEC\_LARGE\_PAGES** 要求在调用方令牌中启用 [SeLockMemoryPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 特权。  如果指定 **了SEC\_LARGE\_PAGES** ，还必须指定 **SEC\_COMMIT** 。  **Windows Server 2003：**直到 Windows Server 2003 SP1 才支持此值。  **Windowsxp：**不支持此值。 | | **SEC\_NOCACHE**  0x10000000 | 将所有页面设置为不可缓存。  应用程序不应使用此属性，除非设备显式需要。 将互锁 **函数与SEC\_NOCACHE** 映射的内存配合使用可能会导致 **EXCEPTION\_ILLEGAL\_INSTRUCTION** 异常。  **SEC\_NOCACHE** 需要设置 **SEC\_RESERVE** 或 **SEC\_COMMIT** 属性。 | | **SEC\_RESERVE**  0x4000000 | 如果文件映射对象由操作系统分页文件提供支持， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) ，则指定当文件的视图映射到进程地址空间时，将保留整个页面范围供进程以后使用，而不是提交。  可以在后续调用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc) 函数时提交保留页。 提交页面后，无法使用 [VirtualFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfree) 函数释放或取消提交页面。  此属性对由可执行映像文件或数据文件支持的文件映射对象无效， (*hfile* 参数是文件) 的句柄。  **SEC\_RESERVE** 不能与 **SEC\_COMMIT** 结合使用。 | | **SEC\_WRITECOMBINE**  0x40000000 | 将所有页面设置为写合并。  除非设备明确需要，否则应用程序不应使用此属性。 将互锁函数 **与SEC\_WRITECOMBINE映射** 的内存结合使用可能会导致 **EXCEPTION\_ILLEGAL\_INSTRUCTION** 异常。  **SEC\_WRITECOMBINE** 要求设置 **SEC\_RESERVE** 或 **SEC\_COMMIT** 属性。  **Windows Server 2003 和 Windows XP：**在 Windows Vista 之前，不支持此标志。 |   [in] dwMaximumSizeHigh  文件映射对象最大大小的高序 **DWORD** 。  [in] dwMaximumSizeLow  文件映射对象最大大小的低序 **DWORD** 。  如果此参数和 *dwMaximumSizeHigh* 为 0 (零) ，则文件映射对象的最大大小等于 *hFile* 标识的文件的当前大小。  尝试映射长度为 0 (零的文件) 失败，错误代码 **为 ERROR\_FILE\_INVALID**。 应用程序应测试长度为 0 (零的文件) 并拒绝这些文件。  [in, optional] lpName  文件映射对象的名称。  如果此参数与现有映射对象的名称匹配，则函数会请求访问具有 *flProtect* 指定的保护的对象。  如果此参数为 **NULL**，则创建不带名称的文件映射对象。  如果 *lpName* 与现有事件、信号灯、互斥体、可等待计时器或作业对象的名称匹配，则函数将失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数将返回 **ERROR\_INVALID\_HANDLE**。 发生这种情况是因为这些对象共享相同的命名空间。  名称可以具有“Global”或“Local”前缀，以在全局命名空间或会话命名空间中显式创建对象。 名称的其余部分可以包含除反斜杠字符 (\) 以外的任何字符。 从会话 0 以外的会话在全局命名空间中创建文件映射对象需要 [SeCreateGlobalPrivilege](https://learn.microsoft.com/zh-cn/windows/win32/secauthz/privilege-constants#constants) 特权。 有关详细信息，请参阅 [内核对象命名空间](https://learn.microsoft.com/zh-cn/windows/desktop/TermServ/kernel-object-namespaces)。  使用终端服务会话实现快速用户切换。 第一个登录用户使用会话 0 (零) ，下一个登录的用户使用会话 1 (一个) ，依此而过。 内核对象名称必须遵循终端服务概述的准则，以便应用程序可以支持多个用户。 返回值 如果函数成功，则返回值是新创建的文件映射对象的句柄。  如果对象在函数调用之前存在，则函数将返回一个句柄，该句柄指向现有对象 (其当前大小，而不是指定大小 [) ，GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_ALREADY\_EXISTS**。  如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 创建文件映射对象后，文件大小不得超过文件映射对象的大小;如果存在，并非所有文件内容都可用于共享。  如果应用程序为文件映射对象指定的大小大于磁盘上实际命名文件的大小，并且页面保护允许写入访问 (即 *flProtect* 参数指定 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE**) ，则磁盘上的文件将增加以匹配文件映射对象的指定大小。 如果文件已扩展，则不保证文件旧端与文件新端之间的文件内容为零;行为由文件系统定义。 如果磁盘上的文件无法增加， **CreateFileMapping** 将失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 将返回 **ERROR\_DISK\_FULL**。  操作系统分页文件支持的文件映射对象中页面的初始内容为 0 (零) 。  **CreateFileMapping** 返回的句柄对新的文件映射对象具有完全访问权限，并且可以与需要文件映射对象的句柄的任何函数一起使用。  多个进程可以通过使用单个共享文件映射对象或创建由同一文件支持的单独文件映射对象来共享同一文件的视图。 通过继承进程创建时的句柄、复制句柄或按名称打开文件映射对象，多个进程可以共享单个文件映射对象。 有关详细信息，请参阅 [CreateProcess](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-createprocessa)、 [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle) 和 [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga) 函数。  创建文件映射对象实际上不会将视图映射到进程地址空间。 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile) 和 [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex) 函数将文件的视图映射到进程地址空间中。  有一个重要例外，从同一文件支持的任何文件映射对象派生的文件视图在特定时间是一致的或相同的。 对于进程内的视图以及由不同进程映射的视图，可以保证一致性。  异常与远程文件相关。 尽管 **CreateFileMapping** 适用于远程文件，但它不会使它们保持一致。 例如，如果两台计算机都将一个文件映射为可写文件，并且都更改了同一页，则每台计算机只能看到自己对页面的写入。 在磁盘上更新数据时，不会合并数据。  映射文件和使用输入和输出访问的文件 (I/O) 函数 ([ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile) 和 [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile)) 不一定是一致的。  文件映射对象的映射视图保留对该对象的内部引用，并且文件映射对象在释放对其的所有引用之前不会关闭。 因此，若要完全关闭文件映射对象，应用程序必须通过调用 [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) 取消映射文件映射对象的所有映射视图，并通过调用 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 关闭文件映射对象句柄。 可以按任意顺序调用这些函数。  通过映射视图修改文件时，上次修改时间戳可能不会自动更新。 如果需要，调用方应使用 [SetFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 设置时间戳。  从会话 0 以外的会话在全局命名空间中创建文件映射对象需要 [SeCreateGlobalPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 特权。 请注意，此特权检查仅限于创建文件映射对象，不适用于打开现有对象。 例如，如果服务或系统在全局命名空间中创建文件映射对象，则在任何会话中运行的任何进程都可以访问该文件映射对象，前提是调用方具有所需的访问权限。  **Windowsxp：**上一段中所述的要求是在 Windows Server 2003 和 Windows XP SP2 中引入的  使用结构化异常处理来保护写入文件视图或从文件视图中读取的任何代码。 有关详细信息，请参阅 [从文件视图中读取和写入](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/reading-and-writing-from-a-file-view)。  若要具有具有可执行权限的映射，应用程序必须使用 **PAGE\_EXECUTE\_READWRITE** 或 **PAGE\_EXECUTE\_READ** 调用 **CreateFileMapping**，然后使用 或 FILE\_MAP\_EXECUTE | FILE\_MAP\_READ调用 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile)FILE\_MAP\_EXECUTE | FILE\_MAP\_WRITE。 |

## 第三步，使用这个API

|  |
| --- |
|  |

### MapViewOfFile函数语法详解

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LPVOID MapViewOfFile(  [in] HANDLE hFileMappingObject,  [in] DWORD dwDesiredAccess,  [in] DWORD dwFileOffsetHigh,  [in] DWORD dwFileOffsetLow,  [in] SIZE\_T dwNumberOfBytesToMap  ); parameters [in] hFileMappingObject  文件映射对象的句柄。 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga) 和 [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga) 函数返回此句柄。  [in] dwDesiredAccess  文件映射对象的访问类型，该对象确定页面的页面保护。 此参数可以是以下值之一，也可以是多个值的按位 OR 组合（如果适用）。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **FILE\_MAP\_ALL\_ACCESS** | 映射文件的读/写视图。 文件映射对象必须已创建 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE** 保护。  与 **MapViewOfFile** 函数一起使用时， **FILE\_MAP\_ALL\_ACCESS** 等效于 **FILE\_MAP\_WRITE**。 | | **FILE\_MAP\_READ** | 映射文件的只读视图。 尝试写入文件视图会导致访问冲突。  创建文件映射对象时必须具有 **PAGE\_READONLY**、 **PAGE\_READWRITE**、 **PAGE\_EXECUTE\_READ**或 **PAGE\_EXECUTE\_READWRITE** 保护。 | | **FILE\_MAP\_WRITE** | 映射文件的读/写视图。 文件映射对象必须已创建 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE** 保护。  与 **MapViewOfFile** 一起使用时， (**FILE\_MAP\_WRITE** | **FILE\_MAP\_READ**) 和 **FILE\_MAP\_ALL\_ACCESS** 等效于 **FILE\_MAP\_WRITE**。 |     使用按位 OR，可以将上述值与这些值组合在一起。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **FILE\_MAP\_COPY** | 映射文件的写入时复制视图。 文件映射对象必须已**创建PAGE\_READONLY、PAGE\_READ\_EXECUTE**、**PAGE\_WRITECOPY**、**PAGE\_EXECUTE\_WRITECOPY**、**PAGE\_READWRITE**或**PAGE\_EXECUTE\_READWRITE**保护。  当进程写入写入时复制页时，系统会将原始页面复制到进程专用的新页。 新页面由分页文件提供支持。 对新页面的保护从写入时复制更改为读/写。  指定写入时复制访问权限时，系统将和进程提交费用用于整个视图，因为调用进程可能会写入视图中的每个页面，从而使所有页面都成为私有页面。 新页面的内容永远不会写回到原始文件，并且当视图未映射时会丢失。 | | **FILE\_MAP\_EXECUTE** | 映射文件的可执行视图 (映射的内存可以作为代码) 运行。 文件映射对象必须已创建 **PAGE\_EXECUTE\_READ**、 **PAGE\_EXECUTE\_WRITECOPY**或 **PAGE\_EXECUTE\_READWRITE** 保护。  **Windows Server 2003 和 Windows XP：**此值从具有 SP2 的 Windows XP 和 SP1 的 Windows Server 2003 开始可用。 | | **FILE\_MAP\_LARGE\_PAGES** | 从 Windows 10 版本 1703 开始，此标志指定应使用[大页面支持](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/large-page-support)映射视图。 视图的大小必须是 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数报告的大型页面大小的倍数，并且文件映射对象必须已使用 **SEC\_LARGE\_PAGES** 选项创建。 如果为 *lpBaseAddress* 提供非 null 值，则该值必须是 **GetLargePageMinimum** 的倍数。  **注意：**在 Windows 10 版本 1703 之前的 OS 版本上，**FILE\_MAP\_LARGE\_PAGES**标志不起作用。 在这些版本中，如果分区是使用 **SEC\_LARGE\_PAGES标志创建的** ，则使用大页面自动映射视图。 | | **FILE\_MAP\_TARGETS\_INVALID** | 将映射文件中的所有位置设置为控制流防护 (CFG) 的无效目标。 此标志类似于 **PAGE\_TARGETS\_INVALID**。 将此标志与执行访问权限 **FILE\_MAP\_EXECUTE**结合使用。 对这些页面中位置的任何间接调用都将失败 CFG 检查，并且进程将终止。 分配的可执行页面的默认行为是标记为 CFG 的有效调用目标。 |     对于使用 **SEC\_IMAGE** 属性创建的文件映射对象， *dwDesiredAccess* 参数不起作用，应设置为任何有效值，例如 **FILE\_MAP\_READ**。  有关访问文件映射对象的详细信息，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/file-mapping-security-and-access-rights)。  [in] dwFileOffsetHigh  视图开始位置的文件偏移量的高阶 **DWORD** 。  [in] dwFileOffsetLow  视图开始位置的文件偏移量的低序 **DWORD** 。 高偏移量和低偏移量的组合必须在文件映射中指定偏移量。 它们还必须匹配系统的内存分配粒度。 也就是说，偏移量必须是分配粒度的倍数。 若要获取系统的内存分配粒度，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数，该函数填充 [SYSTEM\_INFO](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-system_info) 结构的成员。  [in] dwNumberOfBytesToMap  要映射到视图的文件映射的字节数。 所有字节都必须在 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga) 指定的最大大小内。 如果此参数为 0 (零) ，则映射将从指定的偏移量扩展到文件映射的末尾。 返回值 如果函数成功，则返回值是映射视图的起始地址。  如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 映射文件会使文件的指定部分在调用进程的地址空间中可见。  对于大于地址空间的文件，一次只能映射文件数据的一小部分。 第一个视图完成后，可以取消映射它并映射新视图。  若要获取视图的大小，请使用 [VirtualQuery](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualquery) 函数。  如果文件 (或文件映射对象的多个视图及其映射文件) 在指定时间包含相同的数据，则它们是 *一致的* 。 如果文件视图派生自由同一文件支持的任何文件映射对象，则会发生此情况。 进程可以使用 [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle) 函数将文件映射对象句柄复制到另一个进程，或者另一个进程可以使用 [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga) 函数按名称打开文件映射对象。  有一个重要例外，派生自由同一文件支持的任何文件映射对象的文件视图在特定时间是一致的或相同的。 对于进程内的视图以及由不同进程映射的视图，可以保证一致性。  异常与远程文件相关。 尽管 **MapViewOfFile** 适用于远程文件，但无法使其保持一致。 例如，如果两台计算机都将一个文件映射为可写文件，并且都更改了同一页，则每台计算机只看到自己对页面的写入。 在磁盘上更新数据时，不会合并数据。  不保证文件的映射视图与 [ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile) 或 [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) 函数访问的文件一致。  不要将指针存储在内存映射文件中;存储与文件映射基的偏移量，以便可以在任何地址使用映射。  若要防范 **EXCEPTION\_IN\_PAGE\_ERROR** 异常，请使用结构化异常处理来保护任何写入或读取页面文件以外的文件的内存映射视图的代码。 有关详细信息，请参阅 [从文件视图读取和写入](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/reading-and-writing-from-a-file-view)。  通过映射视图修改文件时，上次修改时间戳可能不会自动更新。 如果需要，调用方应使用 [SetFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 设置时间戳。  如果文件映射对象由分页文件提供支持， ([调用 CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga) 并将 *hFile* 参数设置为 **INVALID\_HANDLE\_VALUE**) ，则分页文件必须足够大才能保存整个映射。 否则， **MapViewOfFile** 将失败。 由分页文件支持的文件映射对象中页面的初始内容为 0 (零) 。  创建由分页文件支持的文件映射对象时，调用方可以指定 **是 mapViewOfFile** 应同时保留和提交页面 (**SEC\_COMMIT**) ，还是仅保留页面 (**SEC\_RESERVE**) 。 映射文件会使整个映射的虚拟地址范围无法用于进程中的其他分配。 提交保留范围内的页面后，无法通过调用 [VirtualFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfree) 来释放或取消提交该页。 取消映射视图且文件映射对象关闭时，将释放保留页和提交的页。 有关详细信息，请参阅 [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) 和 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 函数。  若要拥有具有可执行权限的文件，应用程序必须使用 **PAGE\_EXECUTE\_READWRITE** 或 **PAGE\_EXECUTE\_READ** 调用 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga)，然后使用 **FILE\_MAP\_EXECUTE FILE\_MAP\_WRITE** | 或 **FILE\_MAP\_EXECUTE** | **FILE\_MAP\_READ** 调用 **MapViewOfFile**。 |

## 第四步，使用这个api

|  |
| --- |
|  |

## 第五，六步都是使用同一个API

|  |
| --- |
|  |

# 演练

## 利用内存映射文件来读取文件内容

## 1.新建一个MFC基于对话框的应用程序项目，取名：Lesson22-memory\_mapping,需要保留一个按钮改名退出，然后添加点击事件处理代码，防止用户按下回车键后程序马上退出。

|  |
| --- |
|  |

## 2.我们添加一个编辑框控件，把它拉大需要编辑框的multiline属性改为true，垂直滚动条设置为true，然后添加语法静态文本一个编辑框控件ID为ID\_FILE，再添加一个按钮，文本是读取ID：IDC\_BTN\_READ,在按钮的点击事件里面添加代码，一个比较好的办法是给对话框资源绑定变量，如果绑定的是控件变量，类型选择控件，如果绑定是不是控件或者窗口类型的变量选择value类型，否则报错，这里绑定控件，给大的编辑框也绑定一个CString类型变量

|  |
| --- |
|  |

### 注意：一个对话框控件资源只能绑定一个变量。

## 3.我们编写一个函数来实现核心功能，先在对话框的头文件里面声明函数原型

|  |
| --- |
|  |

## 4.先在对话框的cpp文件里面写一个骨架然后需要在读取按钮的点击事件处理函数里面调用这个函数

|  |
| --- |
|  |
|  |

## 5.然后我们来添加这个函数的具体实现代码

|  |
| --- |
| BOOL CLesson22memory\_mappingDlg::LoadFile(CString strFileName)  {  HANDLE hFile,hMapping;  void \*pData;  //1.创建文件对象得到文件句柄  hFile = CreateFile(strFileName,GENERIC\_READ,FILE\_SHARE\_READ,  0,OPEN\_EXISTING,FILE\_FLAG\_SEQUENTIAL\_SCAN,0); //注意这里和普通文件读取方式有点不一样  if(INVALID\_HANDLE\_VALUE == hFile)  {  MessageBox(TEXT("创建文件对象失败"));  return FALSE;  }  //2.创建内存映射文件  hMapping = CreateFileMapping(hFile,0,PAGE\_READONLY,0,0,0);//如果是进程间交换数据，最后一个参数需要指定名字  if(INVALID\_HANDLE\_VALUE == hMapping)  {  MessageBox(TEXT("创建文件映射对象失败"));  //出错了，需要关闭文件对象句柄  CloseHandle(hFile);  return FALSE;  }  //3.将文件数据映射到进程地址空间  pData = MapViewOfFile(hMapping,FILE\_MAP\_READ,0,0,0);//后面三个参数都设置为0，表示从文件开始位置一直映射到结束位置  if(!pData)  {  MessageBox(TEXT("映射文件对象数据到内存失败"));  //出错了，需要关闭映射文件句柄  CloseHandle(hMapping);  //出错了，需要关闭文件对象句柄  CloseHandle(hFile);  }  //3.2使用数据  m\_strFileContent = (LPTSTR)pData;  //4.撤销映射  UnmapViewOfFile(hMapping);  //5.关闭映射文件句柄  CloseHandle(hMapping);  //6.关闭文件对象句柄  CloseHandle(hFile);  return TRUE;  } |

### 注意，用这种方法读取文件，文件的字符编码需要使用UTF-16LE格式，否则会有乱码。

### 效果如下：

|  |
| --- |
|  |

# 这一节学习到此为止，完成代码如下

### Lesson22-memory\_mappingDlg.h

|  |
| --- |
| // Lesson22-memory\_mappingDlg.h : 头文件  //  #pragma once  #include "afxwin.h"  // CLesson22memory\_mappingDlg 对话框  class CLesson22memory\_mappingDlg : public CDialogEx  {  // 构造  public:  CLesson22memory\_mappingDlg(CWnd\* pParent = NULL); // 标准构造函数  // 对话框数据  enum { IDD = IDD\_LESSON22MEMORY\_MAPPING\_DIALOG };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  HICON m\_hIcon;  BOOL LoadFile(CString strFileName);  // 生成的消息映射函数  virtual BOOL OnInitDialog();  afx\_msg void OnSysCommand(UINT nID, LPARAM lParam);  afx\_msg void OnPaint();  afx\_msg HCURSOR OnQueryDragIcon();  DECLARE\_MESSAGE\_MAP()  public:  afx\_msg void OnBnClickedOk();  afx\_msg void OnBnClickedBtnRead();  private:  // 接受用户输入文件名称的编辑框成员变量  CEdit m\_edFileName;  // 给控件绑定值类型的成员变量  CString m\_strFileName;  // 接受文件内容的成员变量和编辑框控件绑定    public:  // 编辑框控件绑定一个CString成员变量来保存文件内容  CString m\_strFileContent;  }; |

### Lesson22-memory\_mappingDlg.cpp

|  |
| --- |
| // Lesson22-memory\_mappingDlg.cpp : 实现文件  //  #include "stdafx.h"  #include "Lesson22-memory\_mapping.h"  #include "Lesson22-memory\_mappingDlg.h"  #include "afxdialogex.h"  #ifdef \_DEBUG  #define new DEBUG\_NEW  #endif  // 用于应用程序“关于”菜单项的 CAboutDlg 对话框  class CAboutDlg : public CDialogEx  {  public:  CAboutDlg();  // 对话框数据  enum { IDD = IDD\_ABOUTBOX };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  DECLARE\_MESSAGE\_MAP()  };  CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)  {  }  void CAboutDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)  END\_MESSAGE\_MAP()  // CLesson22memory\_mappingDlg 对话框  CLesson22memory\_mappingDlg::CLesson22memory\_mappingDlg(CWnd\* pParent /\*=NULL\*/)  : CDialogEx(CLesson22memory\_mappingDlg::IDD, pParent)    , m\_strFileName(\_T(""))  , m\_strFileContent(\_T(""))  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  }  void CLesson22memory\_mappingDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  DDX\_Control(pDX, IDC\_FILENAME, m\_edFileName);  DDX\_Text(pDX, IDC\_CONTENT, m\_strFileContent);  }  BEGIN\_MESSAGE\_MAP(CLesson22memory\_mappingDlg, CDialogEx)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  ON\_BN\_CLICKED(IDOK, &CLesson22memory\_mappingDlg::OnBnClickedOk)  ON\_BN\_CLICKED(IDC\_BTN\_READ, &CLesson22memory\_mappingDlg::OnBnClickedBtnRead)  END\_MESSAGE\_MAP()  // CLesson22memory\_mappingDlg 消息处理程序  BOOL CLesson22memory\_mappingDlg::OnInitDialog()  {  CDialogEx::OnInitDialog();  // 将“关于...”菜单项添加到系统菜单中。  // IDM\_ABOUTBOX 必须在系统命令范围内。  ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);  ASSERT(IDM\_ABOUTBOX < 0xF000);  CMenu\* pSysMenu = GetSystemMenu(FALSE);  if (pSysMenu != NULL)  {  BOOL bNameValid;  CString strAboutMenu;  bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);  ASSERT(bNameValid);  if (!strAboutMenu.IsEmpty())  {  pSysMenu->AppendMenu(MF\_SEPARATOR);  pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);  }  }  // 设置此对话框的图标。当应用程序主窗口不是对话框时，框架将自动  // 执行此操作  SetIcon(m\_hIcon, TRUE); // 设置大图标  SetIcon(m\_hIcon, FALSE); // 设置小图标  // TODO: 在此添加额外的初始化代码    return TRUE; // 除非将焦点设置到控件，否则返回 TRUE  }  void CLesson22memory\_mappingDlg::OnSysCommand(UINT nID, LPARAM lParam)  {  if ((nID & 0xFFF0) == IDM\_ABOUTBOX)  {  CAboutDlg dlgAbout;  dlgAbout.DoModal();  }  else  {  CDialogEx::OnSysCommand(nID, lParam);  }  }  // 如果向对话框添加最小化按钮，则需要下面的代码  // 来绘制该图标。对于使用文档/视图模型的 MFC 应用程序，  // 这将由框架自动完成。  void CLesson22memory\_mappingDlg::OnPaint()  {  if (IsIconic())  {  CPaintDC dc(this); // 用于绘制的设备上下文  SendMessage(WM\_ICONERASEBKGND, reinterpret\_cast<WPARAM>(dc.GetSafeHdc()), 0);  // 使图标在工作区矩形中居中  int cxIcon = GetSystemMetrics(SM\_CXICON);  int cyIcon = GetSystemMetrics(SM\_CYICON);  CRect rect;  GetClientRect(&rect);  int x = (rect.Width() - cxIcon + 1) / 2;  int y = (rect.Height() - cyIcon + 1) / 2;  // 绘制图标  dc.DrawIcon(x, y, m\_hIcon);  }  else  {  m\_edFileName.SetFocus();//第一个编辑框控件需要在对话框初始化的时候获取焦点  CDialogEx::OnPaint();  }  }  //当用户拖动最小化窗口时系统调用此函数取得光标  //显示。  HCURSOR CLesson22memory\_mappingDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  void CLesson22memory\_mappingDlg::OnBnClickedOk()  {  // TODO: 在此添加控件通知处理程序代码  //解决按下回车就会退出程序的问题，需要使用全局的MessageBox  if(IDOK == ::MessageBox(this->m\_hWnd,TEXT("退出应用程序?"),TEXT("退出提示"),MB\_OKCANCEL))  {  CDialogEx::OnOK();  }  }  void CLesson22memory\_mappingDlg::OnBnClickedBtnRead()  {  // TODO: 在此添加控件通知处理程序代码  //处理编辑框内容为空的情况  GetDlgItemText(IDC\_FILENAME,m\_strFileName);  if(m\_strFileName.GetLength()==0)  {  MessageBox(TEXT("请输入文件名"));  m\_edFileName.SetFocus();  }  else  {  if(!LoadFile(m\_strFileName))  {  MessageBox(TEXT("读取文件内容失败"));  return;  }  UpdateData(FALSE);//给绑定控件的变量赋值后需要调用 UpdateData()更新数据，而且需要传递FALSE  }    }  BOOL CLesson22memory\_mappingDlg::LoadFile(CString strFileName)  {  HANDLE hFile,hMapping;  void \*pData;  //1.创建文件对象得到文件句柄  hFile = CreateFile(strFileName,GENERIC\_READ,FILE\_SHARE\_READ,  0,OPEN\_EXISTING,FILE\_FLAG\_SEQUENTIAL\_SCAN,0); //注意这里和普通文件读取方式有点不一样  if(INVALID\_HANDLE\_VALUE == hFile)  {  MessageBox(TEXT("创建文件对象失败"));  return FALSE;  }  //2.创建内存映射文件  hMapping = CreateFileMapping(hFile,0,PAGE\_READONLY,0,0,0);//如果是进程间交换数据，最后一个参数需要指定名字  if(INVALID\_HANDLE\_VALUE == hMapping)  {  MessageBox(TEXT("创建文件映射对象失败"));  //出错了，需要关闭文件对象句柄  CloseHandle(hFile);  return FALSE;  }  //3.将文件数据映射到进程地址空间  pData = MapViewOfFile(hMapping,FILE\_MAP\_READ,0,0,0);//后面三个参数都设置为0，表示从文件开始位置一直映射到结束位置  if(!pData)  {  MessageBox(TEXT("映射文件对象数据到内存失败"));  //出错了，需要关闭映射文件句柄  CloseHandle(hMapping);  //出错了，需要关闭文件对象句柄  CloseHandle(hFile);  }  //3.2使用数据  m\_strFileContent = (LPTSTR)pData;  //4.撤销映射  UnmapViewOfFile(pData);  //5.关闭映射文件句柄  CloseHandle(hMapping);  //6.关闭文件对象句柄  CloseHandle(hFile);  return TRUE;  } |