隧道检测程序大文件设计文档V1.0

**目录**

[1引言 1](#_Toc475718935)

[1.1编写目的 1](#_Toc475718936)

[1.2背景 1](#_Toc475718937)

[1.3定义 1](#_Toc475718938)

[2软件架构及数据结构 2](#_Toc475718939)

[2.1软件架构 2](#_Toc475718940)

[2.2数据结构 3](#_Toc475718941)

[3 接口函数 4](#_Toc475718942)

[3.1接口函数列表 4](#_Toc475718943)

[3.2 RSSFWirterCreateRssfFile 4](#_Toc475718944)

[3.3 RSSFWirterCreateRssfFileAndIndex 4](#_Toc475718945)

[3.4 RSSFReaderReadRssfFile 4](#_Toc475718946)

[3.5 RSSFWirterCreateRssfFileAndIndex 5](#_Toc475718947)

[3.6 writer->SetFileHead 5](#_Toc475718948)

[3.7 writer->SaveImage 5](#_Toc475718949)

[3.8 writer->SaveImageFileWithoutIndex 5](#_Toc475718950)

[3.9 writer->Final 5](#_Toc475718951)

[3.10 reader->GetUniversalTrackInfo 5](#_Toc475718952)

[3.11 reader-> GetNextTrackImage 5](#_Toc475718953)

[3.12 reader-> GetTrackImage 6](#_Toc475718954)

[3.13 reader-> ReleaseImage 6](#_Toc475718955)

[3.14 reader-> RefreshIndex 6](#_Toc475718956)

[3.15 reader-> GetTrackImageList 6](#_Toc475718957)

[3.16 reader-> ReleaseImageList 6](#_Toc475718958)

[3.17 reader-> GetImageIDByMileAge 6](#_Toc475718959)

[3.18 reader-> Final 6](#_Toc475718960)

[4卷宗管理模块设计 8](#_Toc475718961)

[4.1卷宗文件结构 8](#_Toc475718962)

[5生成RSSF图像卷宗 9](#_Toc475718963)

[6从RSSF图像卷宗获取图像 9](#_Toc475718964)

[7使用方法 10](#_Toc475718965)

[7.1生成图像卷宗 10](#_Toc475718966)

[7.2顺序读取图像卷宗 10](#_Toc475718967)

[7.3查询图像卷宗 10](#_Toc475718968)

[8常用WINAPI函数 11](#_Toc475718969)

[8.1 VirtualAlloc() 11](#_Toc475718970)

[8.2 VirtualFree() 11](#_Toc475718971)

[8.3 VirtualLock() 12](#_Toc475718972)

[8.4 CreateEvent() 12](#_Toc475718973)

[8.5 SetEvent () 12](#_Toc475718974)

[8.6 ResetEvent () 13](#_Toc475718975)

[8.7 SetFilePointer () 13](#_Toc475718976)

[8.8 WaitForSingleObject () 13](#_Toc475718977)

[8.9 WaitForMultipleObjectsEx () 14](#_Toc475718978)

[8.10 ZeroMemory () 15](#_Toc475718979)

[8.11 CopyMemory () 15](#_Toc475718980)

# 1引言

## 1.1编写目的

本文档主要说明海量图像卷宗化管理软件的基本功能和实现方案。预期读者包括本软件的开发人员以及本软件的测试人员。

## 1.2背景

## 1.3定义

**图像卷宗：**分Windows下文件大小超过10GB的文件，它需要通过特定的技术进行读写操作。

# 2软件架构及数据结构

## 2.1软件架构



在获取隧道卷宗文件的过程中，用户并不是直接通过以上定义的类CWriteHugeFile和CReadHugeFile来实现的，而是通过顶层的UI让用户来操作，其底层具体细节都是封装过的，对用户来说是隐藏的，如上图所示。

具体实现过程如下：



文件功能说明：

|  |  |
| --- | --- |
| 文件 | 作用 |
| Rssf.h | 写图像卷宗类WrssfFile、读图像卷宗类RrssfFile的定义以及所需方法（函数）声明的头文件 |
| trackimagestruct.h | 大文件头信息结构以及隧道图像结构定义的头文件 |
| WriteHugeFile.h | 写大文件类CWriteHugeFile的定义以及所需方法（函数）声明的头文件 |
| ReadHugeFile.h | 读大文件类CReadHugeFile的定义以及所需方法（函数）声明的头文件 |
| WriteHugeFile.cpp | 写入图像卷宗所需方法（函数）的定义 |
| ReadHugeFile.cpp | 读取图像卷宗所需方法（函数）的定义 |
| Rssf.CLR.h | 操作大文件的帮助类RrssfFileHelper的定义以及所需方法（函数）声明的头文件 |
| Rssf.CLR.cpp | RrssfFileHelper操作大文件所需的方法（函数）的定义 |

## 2.2数据结构

|  |  |  |
| --- | --- | --- |
| 结构体名 | 作用 | 头文件 |
| UniversalTrackInfo | 大文件头信息 | Rssf/trackimagestruct.h |
| TrackImageJPG | 需要写入大文件的隧道图片及对应的隧道信息 | Rssf/trackimagestruct.h |

|  |  |  |
| --- | --- | --- |
| 类名 | 作用 | 头文件 |
| WrssfFile | 写入RSSF图像卷宗类 | Rssf/Rssf.h |
| RrssfFile | 读取RSSF图像卷宗类 | Rssf/Rssf.h |
| TrackImageInfo | 隧道图像信息 | Rssf.CLR.h |
| UniversalLineInfo | 大线路文件信息 | Rssf.CLR.h |
| RrssfFileHelper | 操作大文件的帮助类 | Rssf.CLR.h |

# 3 接口函数

## 3.1接口函数列表

|  |  |  |
| --- | --- | --- |
|  | 方法（函数）名 | 功能 |
| 全局 | RSSFWirterCreateRssfFile | 创建RSSF图像卷宗 |
| RSSFWirterCreateRssfFileAndIndex | 创建RSSF图像卷宗并新建索引 |
| RSSFReaderReadRssfFile | 读取RSSF图像卷宗 |
| RSSFReaderReadRssfFileAndIndex | 读取RSSF图像卷宗，打开/创建索引 |
| Writer | writer->SetFileHead | 向图像卷宗中写入RSSF文件头信息 |
| writer->SaveImage | 向RSSF图像卷宗中写入单幅隧道图像 |
| writer->Final | 写图像卷宗结束 |
| writer->SaveImageFileWithoutIndex | 向RSSF图像卷宗中写入单幅隧道图像 |
| Reader | reader->GetUniversalTrackInfo | 从RSSF图像卷宗中获取卷宗头信息 |
| reader-> GetNextTrackImage | 从RSSF图像卷宗中顺序读取下一幅图像 |
| reader-> GetTrackImage | 根据图片编号从RSSF图像卷宗中获取图像 |
| reader-> ReleaseImage | 释放Reader提供的一幅隧道图像 |
| reader-> RefreshIndex | 刷新索引中的信息，更新内存索引数据 |
| reader-> GetTrackImageList | 根据公里指标从图像卷宗中查询多幅图像 |
| reader->ReleaseImageList | 释放由Reader提供的隧道图像数组 |
| reader->GetImageIDByMileAge | 根据公里指标获取RSSF图像卷宗中图像号 |
| reader-> Final | 读图像卷宗结束 |

## 3.2 RSSFWirterCreateRssfFile

|  |  |
| --- | --- |
| WrssfFile\* \_\_stdcall RSSFWirterCreateRssfFile(const char\* pSzfilePath) | |
| 功能 | 创建RSSF图像卷宗 |
| 参数 | char\* pSzfilePath：RSSF图像卷宗路径 |
| 返回值 | WrssfFile\*：WrssfFile指针 |

## 3.3 RSSFWirterCreateRssfFileAndIndex

|  |  |
| --- | --- |
| WrssfFile\* \_\_stdcall RSSFWirterCreateRssfFileAndIndex(const char\* pSzfilePath, const char\* pSzIndexFilePath); | |
| 功能 | 创建RSSF图像卷宗，并新建索引 |
| 参数 | char\* pSzfilePath：RSSF图像卷宗路径  char\* pSzIndexFilePath：索引文件路径 |
| 返回值 | WrssfFile\*：WrssfFile指针 |

## 3.4 RSSFReaderReadRssfFile

|  |  |
| --- | --- |
| RrssfFile\* \_\_stdcall RSSFReaderReadRssfFile(const char\* pSzfilePath); | |
| 功能 | 读取RSSF图像卷宗 |
| 参数 | char\* pSzfilePath：RSSF图像卷宗路径 |
| 返回值 | RrssfFile \*：RrssfFile指针 |

## 3.5 RSSFWirterCreateRssfFileAndIndex

|  |  |
| --- | --- |
| RrssfFile\* \_\_stdcall RSSFReaderReadRssfFileAndIndex(const char\* pSzfilePath,const char\* pSzIndexFilePath, bool createIndex) | |
| 功能 | 读取RSSF图像卷宗，打开/创建索引 |
| 参数 | char\* pSzfilePath：RSSF图像卷宗路径  char\* pSzIndexFilePath：索引文件路径  bool createIndex：是否创建索引文件 |
| 返回值 | RrssfFile \*：RrssfFile指针 |

## 3.6 writer->SetFileHead

|  |  |
| --- | --- |
| BOOL SetFileHead(UniversalTrackInfo\* pFileHeader) | |
| 功能 | 向图像卷宗中写入RSSF文件头信息 |
| 参数 | UniversalTrackInfo\* pFileHeader：RSSF头文件数据结构 |
| 返回值 | BOOL：操作是否成功的布尔值 |

## 3.7 writer->SaveImage

|  |  |
| --- | --- |
| \_\_int64 SaveImage(TrackImageJPG\* pTrackimage,BOOL bWriteIdx = TRUE) | |
| 功能 | 向RSSF图像卷宗中保存一幅隧道图像 |
| 参数 | TrackImageJPG\* pTrackimage：隧道图像数据结构 |
| 返回值 | \_\_int64：该图像在RSSF图像卷宗中的地址偏移量 |

## 3.8 writer->SaveImageFileWithoutIndex

|  |  |
| --- | --- |
| \_\_int64 SaveImageFileWithoutIndex(TrackImageJPG\* pTrackimage) | |
| 功能 | 向RSSF图像卷宗中保存一幅隧道图像 |
| 参数 | TrackImageJPG\* pTrackimage：隧道图像数据结构 |
| 返回值 | \_\_int64：该图像在RSSF图像卷宗中的地址偏移量 |

## 3.9 writer->Final

|  |  |
| --- | --- |
| BOOL Final() | |
| 功能 | 写入图像卷宗结束 |
| 参数 | 无 |
| 返回值 | BOOL：RSSF图像卷宗是否成功关闭 |

## 3.10 reader->GetUniversalTrackInfo

|  |  |
| --- | --- |
| UniversalTrackInfo\* GetUniversalTrackInfo(); | |
| 功能 | 从RSSF卷宗文件中获取图像卷宗头信息 |
| 参数 | 无 |
| 返回值 | UniversalTrackInfo\*：RSSF文件头信息结构体 |

## 3.11 reader-> GetNextTrackImage

|  |  |
| --- | --- |
| BOOL GetNextTrackImage(TrackImageJPG\* pTrackImage); | |
| 功能 | 从RSSF卷宗文件中顺序读取下一幅图像 |
| 参数 | TrackImageJPG\* pTrackImage：隧道图像结构体 |
| 返回值 | BOOL：成功查询到下一张图像的布尔值 |

## 3.12 reader-> GetTrackImage

|  |  |
| --- | --- |
| BOOL GetTrackImage(TrackImageJPG\* pTrackImage, int photoNum) | |
| 功能 | 根据图像编号，从RSSF卷宗文件中查询图像 |
| 参数 | TrackImageJPG\* pTrackImage：隧道图像结构体  int photoNum：图像编号 |
| 返回值 | BOOL：操作成功的布尔值 |

## 3.13 reader-> ReleaseImage

|  |  |
| --- | --- |
| void ReleaseImage(TrackImageJPG\* pTrackImage) | |
| 功能 | 释放Reader提供的一幅隧道图像 |
| 参数 | TrackImageJPG\* pTrackImage：待释放的隧道图像数据 |
| 返回值 | 无 |

## 3.14 reader-> RefreshIndex

|  |  |
| --- | --- |
| BOOL RefreshIndex() | |
| 功能 | 刷新索引中的信息，更新内存索引数据 |
| 参数 | 无 |
| 返回值 | 无 |

## 3.15 reader-> GetTrackImageList

|  |  |
| --- | --- |
| TrackImageJPG\*\* GetTrackImageList(short KM, short M, int\* count) | |
| 功能 | 根据公里指标从图像卷宗中查询多幅图像 |
| 参数 | short KM：查询的KM条件  short M：查询的M条件  int\* count：用于返回查询到的图像数量的指针 |
| 返回值 | TrackImageJPG\*\*：包含图像结构体指针的数组 |

## 3.16 reader-> ReleaseImageList

|  |  |
| --- | --- |
| void ReleaseImageList(TrackImageJPG\*\* imageList, size\_t size) | |
| 功能 | 释放由Reader提供的隧道图像数组 |
| 参数 | TrackImageJPG\*\* imageList：待释放的隧道图像数组  size\_t size：图像数组长度 |
| 返回值 | 无 |

## 3.17 reader-> GetImageIDByMileAge

|  |  |
| --- | --- |
| int GetImageIDByMileAge(short KM, short M) | |
| 功能 | 根据公里指标获取RSSF图像卷宗中图像编号 |
| 参数 | short KM：查询的KM条件  short M：查询的M条件 |
| 返回值 | int：查询得到的图像编号 |

## 3.18 reader-> Final

|  |  |
| --- | --- |
| BOOL Final() | |
| 功能 | 读图像卷宗文件结束 |
| 参数 | 无 |
| 返回值 | BOOL：操作是否成功的布尔值 |

# 4卷宗管理模块设计

## 4.1卷宗文件结构

卷宗文件中需要保存的信息主要包括隧道图像的灰度矩阵。为了实际操作的方便，在保存图像灰度矩阵时转换为一维uchar数组。

隧道图像卷宗文件结构如下：

DEX文件结构



RSSF文件结构



# 5生成RSSF图像卷宗

|  |
| --- |
| 功能：生成隧道图像卷宗  输入：RSSF图像卷宗路径、索引文件路径  输出：左、右隧道对应的卷宗文件 |
| 步骤一：加载左、右半幅隧道图片  步骤二：拼接左、右半幅隧道图片  步骤三：构造左右隧道图片卷宗文件。创建图像卷宗并新建索引；向图像卷宗中写入文件头信息；最后将图片批量写入大文件中  步骤四：写入图像卷宗结束  步骤五：返回左、右隧道图片卷宗文件 |

# 6从RSSF图像卷宗获取图像

|  |
| --- |
| 功能：获取图像卷宗中的图片  输入：RSSF图像卷宗  输出：隧道图片 |
| 步骤一：打开图像卷宗和索引文件  步骤二：获得RSSF图像卷宗头信息  步骤三：根据图像编号、里程等信息查询  步骤四：写入图像卷宗结束  步骤五：返回左、右隧道图片 |

# 7使用方法

## 7.1生成图像卷宗

1. RSSFWirterCreateRssfFile 创建新的RSSF图像卷宗
2. writer->SetFileHead() 保存RSSF图像卷宗的文件头信息
3. writer->SaveImage() 向RSSF图像卷宗保存图像数据
4. writer->Final () 写入RSSF图像卷宗结束
5. delete writer

## 7.2顺序读取图像卷宗

1. RSSFReaderReadRssfFile 打开RSSF图像卷宗
2. reader->GetUniversalTrackInfo() 获得RSSF图像卷宗头信息
3. reader->GetNextTrackImage(…) 获得RSSF图像卷宗中下一幅图像的信息

*pTrackImageJPG ptrImage = new TrackImageJPG;*

*reader-> GetNextTrackImage (ptrImage);*

*…*

*reader->ReleaseImage(ptrImage);*

*delete ptrImage;*

1. reader->Final () 顺序读取RSSF图像卷宗结束
2. delete reader

## 7.3查询图像卷宗

1. RSSFReaderReadRssfFileAndIndex 打开图像卷宗和索引文件（createIndex参数为false）
2. reader->GetUniversalTrackInfo() 获得RSSF图像卷宗头信息
3. 查询RSSF图像卷宗中的图像
   1. reader->GetTrackImage(…) 根据图像编号查询

*pTrackImageJPG ptrImage = new TrackImageJPG;*

*reader->GetTrackImage(ptrImage, PhotoNum);*

*…*

*reader->ReleaseImage(ptrImage);*

*delete ptrImage;*

* 1. reader-> GetTrackImageList (…) 根据KM、M条件查询

*int count = 0;*

*pTrackImageJPG\* ptrImages = reader->GetTrackImageList(KM, M, &count);*

*…*

*reader->ReleaseImageList(ptrImages,count);*

1. reader->RefreshIndex 更新内存中的索引数据（如果需要）
2. reader->Final () 查询RSSF图像卷宗结束
3. delete reader

# 8常用WINAPI函数

**说明**：

1.此处的WINAPI函数的使用文件主要为RSSF目录下的ReadHugeFile.cpp以及WriteHugeFile.cpp两个文件

2.\_\_In：输入参数； \_\_Out：输出参数 ；\_\_In\_opt：可选的输入参数

## 8.1 VirtualAlloc()

|  |  |
| --- | --- |
| 功能 | 在调用进程的虚地址空间,预定或者提交一部分页，简单点的意思就是申请内存空间 |
| 原型 | LPVOID  WINAPI  VirtualAlloc(  \_In\_opt\_ LPVOID lpAddress,  \_In\_ SIZE\_T dwSize,  \_In\_ DWORD flAllocationType,  \_In\_ DWORD flProtect  ); |
| 参数 | \_In\_opt\_ LPVOID lpAddress, //分配的起始位置  //如果要保留一段内存区域，函数会将其自动向最近的一个分配粒度对齐；  //如果要提交一段内存区域，函数将会向最近的一个页面对齐；  //如果为NULL，系统自行决定在什么地方分配  \_In\_ SIZE\_T dwSize //所需要分配的内存字节大小  \_In\_ DWORD flAllocationType //分配类型：MEM\_COMMIT(提交)、MEM\_RESERVED(保留)  \_In\_ DWORD flProtect //内存保护属性：PAGE\_READWRITE、PAGE\_EXECUTE… |
| 返回值 | 成功时，返回指向分配到的内存的起始地址的指针；  失败时，返回NULL |
| 备注 |  |

## 8.2 VirtualFree()

|  |  |
| --- | --- |
| 功能 | 将当前进程内存状态从提交变为保留，或将保留变为空闲，或同时进行 |
| 原型 | BOOL  WINAPI  VirtualFree(  LPVOID lpAddress,  \_In\_ SIZE\_T dwSize,  \_In\_ DWORD dwFreeType  ); |
| 参数 | LPVOID lpAddress：//需要改变状态的内存区域的起始地址  \_In\_ SIZE\_T dwSize：//需要改变状态的内存区域的字节大小  \_In\_ DWORD dwFreeType：//MEM\_DECOMMIT—将内存变为保留状态  //MEM\_RELEASE—释放内存，将内存变为空闲状态 |
| 返回值 | 成功时，返回非零值；  失败时，返回零值 |
| 备注 |  |

## 8.3 VirtualLock()

|  |  |
| --- | --- |
| 功能 | 将虚拟地址空间中内存页面的一片区域锁定在物理内存里 |
| 原型 | BOOL  WINAPI  VirtualLock(  \_In\_ LPVOID lpAddress,  \_In\_ SIZE\_T dwSize  ); |
| 参数 | \_In\_ LPVOID lpAddress：//要锁定内存页的基地址  \_In\_ SIZE\_T dwSize：//要锁定的区域大小，以字节为单位 |
| 返回值 |  |
| 备注 | 将页面锁定在物理内存中使用VirtualLock函数。这样页面将不会被交换到硬盘上，提高了访问的效率，如果要解除它，可以使用VirtualUnLock。 |

## 8.4 CreateEvent()

|  |  |
| --- | --- |
| 功能 | 用来创建或打开一个命名的或无名的事件对象 |
| 原型 | HANDLE  WINAPI  CreateEventA(  \_In\_opt\_ LPSECURITY\_ATTRIBUTES lpEventAttributes,  \_In\_ BOOL bManualReset,  \_In\_ BOOL bInitialState,  \_In\_opt\_ LPCSTR lpName  ); |
| 参数 | \_In\_opt\_ LPSECURITY\_ATTRIBUTES lpEventAttributes：//安全属性  \_In\_ BOOL bManualReset：//复位方式  \_In\_ BOOL bInitialState：//初始状态  \_In\_opt\_ LPCSTR lpName：//对象名称 |
| 返回值 | 成功时，函数返回事件对象的句柄；  失败时，函数返回值为NULL |
| 备注 |  |

## 8.5 SetEvent ()

|  |  |
| --- | --- |
| 功能 | 设置事件的状态为有标记 |
| 原型 | BOOL  WINAPI  SetEvent(  \_In\_ HANDLE hEvent  ); |
| 参数 | \_In\_ HANDLE hEvent：// 句柄 |
| 返回值 | 成功时，返回非零值；  失败时，返回零值 |
| 备注 |  |

## 8.6 ResetEvent ()

|  |  |
| --- | --- |
| 功能 | 用于手动重置事件对象 |
| 原型 | BOOL  WINAPI  ResetEvent(  \_In\_ HANDLE hEvent  ); |
| 参数 | \_In\_ HANDLE hEvent：//hEvent 指向事件对象的句柄，由 CreateEvent or OpenEvent 函数返回 |
| 返回值 | 成功时，返回非零值  失败时，返回零值 |
| 备注 |  |

## 8.7 SetFilePointer ()

|  |  |
| --- | --- |
| 功能 | 在一个文件中设置当前的读取位置 |
| 原型 | DWORD  WINAPI  SetFilePointer(  \_In\_ HANDLE hFile,  \_In\_ LONG lDistanceToMove,  \_Inout\_opt\_ PLONG lpDistanceToMoveHigh,  \_In\_ DWORD dwMoveMethod  ); |
| 参数 | \_In\_ HANDLE hFile：// 文件句柄  \_In\_ LONG lDistanceToMove：//偏移量（低位）  \_Inout\_opt\_ PLONG lpDistanceToMoveHigh：//偏移量（高位）  \_In\_ DWORD dwMoveMethod：// 基准位置  FILE\_BEGIN:文件开始位置  FILE\_CURRENT:文件当前位置  FILE\_END:文件结束位置 |
| 返回值 | 成功时，返回一个新位置 |
| 备注 |  |

## 8.8 WaitForSingleObject ()

|  |  |
| --- | --- |
| 功能 | 用来检测hHandle事件的信号状态 |
| 原型 | DWORD  WINAPI  WaitForSingleObject(  \_In\_ HANDLE hHandle,  \_In\_ DWORD dwMilliseconds  ); |
| 参数 | \_In\_ HANDLE hHandle：// 事件的句柄  \_In\_ DWORD dwMilliseconds：//时间间隔 |
| 返回值 | 有三种返回类型：  WAIT\_OBJECT\_0, 表示等待的对象有信号（对线程来说，表示执行结束）；  WAIT\_TIMEOUT, 表示等待指定时间内，对象一直没有信号（线程没执行完）；  WAIT\_ABANDONED 表示对象有信号，但还是不能执行 一般是因为未获取到锁或其他原因 |
| 备注 |  |

## 8.9 WaitForMultipleObjectsEx ()

|  |  |
| --- | --- |
| 功能 | 等待，直到一个或所有指定的对象是在信号状态，排队到线程I / O完成例程或异步过程调用（APC），或超时的时间间隔内。 |
| 原型 | DWORD  WINAPI  WaitForMultipleObjectsEx(  \_In\_ DWORD nCount,  \_In\_reads\_(nCount) CONST HANDLE \* lpHandles,  \_In\_ BOOL bWaitAll,  \_In\_ DWORD dwMilliseconds,  \_In\_ BOOL bAlertable  ); |
| 参数 | \_In\_ DWORD nCount：//句柄指向的数组由lpHandles在等待的对象。对象句柄的最大数量是64。此参数不能是零  \_In\_reads\_(nCount) CONST HANDLE \* lpHandles：//  \_In\_ BOOL bWaitAll：//如果此参数为TRUE，则函数返回时的lpHandles阵列中的所有对象的状态被设置为“有信号状态”。如果为FALSE时，该函数返回的任何的一个对象的状态被设置为“有信号状态”。在后者的情况下，返回值表示的是对象返回的列表下标。  \_In\_ DWORD dwMilliseconds：//如果指定一个非零值，这个函数等待，直到指定的对象的信号，I / O完成例程或APC入队列，或经过的时间间隔。dwMilliseconds是零，则不进入等待状态，如果不符合标准的，它总是立即返回。如果dwMilliseconds是无限的，只有当该函数将返回指定对象的信号或一个I / O完成例程或APC排队。  \_In\_ BOOL bAlertable：//如果此参数为TRUE，线程处于等待状态，系统队列时，该函数返回一个I / O完成例程或APC，线程运行的例行程序或功能。否则，该函数不返回，不执行完成例程或APC功能。 |
| 返回值 | https://msdn.microsoft.com/en-us/library/ms687028(VS.85).aspx |
| 备注 |  |

## 8.10 ZeroMemory ()

|  |  |
| --- | --- |
| 功能 | ZeroMemory宏用0来填充一块内存区域 |
| 原型 | void ZeroMemory(  PVOID Destination,  SIZE\_T Length  ); |
| 参数 | PVOID Destination：//指向一块准备用0来填充的内存区域的开始地址  SIZE\_T Length：//准备用0来填充的内存区域的大小，按字节来计算 |
| 返回值 | 无 |
| 备注 |  |

## 8.11 CopyMemory ()

|  |  |
| --- | --- |
| 功能 | 将一块内存的数据从一个位置复制到另一个位置 |
| 原型 | VOID CopyMemory(  PVOID Destination,  CONST VOID \*Source,  SIZE\_T Length  ); |
| 参数 | Destination：//要复制内存块的目的地址。  Source：//要复制内存块的源地址。  Length：//指定要复制内存块的大小，单位为字节 |
| 返回值 | 无 |
| 备注 |  |

# 9内存映射文件

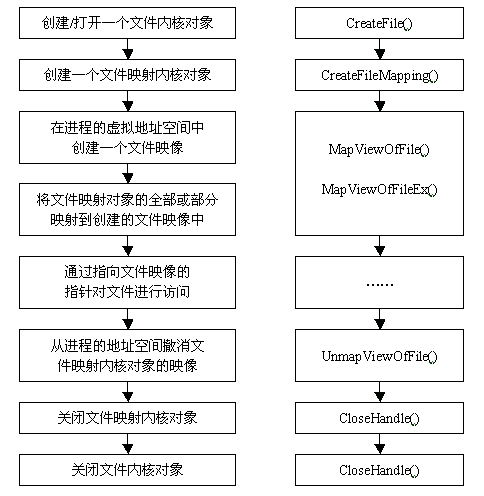
## 9.1引言

Win32 API和MFC均提供了支持文件处理的函数和类，比如Win32 API下常用的CreateFile()、WriteFile()、ReadFile()等。一般来说，以上这些函数可以满足大多数场合的需求，但是如果涉及到某些几十GB、几百GB、乃至几TB的海量数据，如果我们再以普通文件的处理方法进行处理显然是行不通的。针对以上这种大文件的操作一般是以内存文件映射的方式进行操作的。

## 9.2内存映射文件概述

内存文件映射提供了一个统一的内存管理特征，使应用程序可以通过内存指针对磁盘上的文件进行访问，其过程就如同对加载了文件的内存的访问。通过文件映射这种使磁盘文件的全部或部分内容与进程虚拟地址空间的某个区域建立映射关联的能力，可以直接对被映射的文件进行访问，而不必执行文件I/O操作也无需对文件内容进行缓冲处理。内存文件映射的这种特性是非常适合于用来管理大尺寸文件的。

下面给出内存映射文件的一般流程框图：



## 9.3使用方法

BOOL CDemoDlg::LoadFile(CString strFileName)

{

//创建一个句柄

HANDLE hFile, hMapping;

void \*basepointer;

/\*

HANDLE CreateFile(

//文件名

PCSTR pszFileName,

//对文件操作的方式

DWORD dwDesiredAccess,

//指定共享的方式

DWORD dwShareMode,

//指定文件的安全属性，0表示Windows默认的

PSECURITY\_ATTRIBUTES psa,

//创建新的文件还是打开已经存在的文件

DWORD dwCreationDisposition,

//文件的标志和属性：比如顺序访问、随机存取

DWORD dwFlagsAndAttributes,

//创建文件对象的模板

HANDLE hTemplateFile);

HANDLE CreateFileMapping(

//第一步创建文件对象的句柄

HANDLE hFile,

//文件安全属性，跟第一步的一致

PSECURITY\_ATTRIBUTES psa,

//内存页的页面保护属性

DWORD fdwProtect,

//指定要映射的内存大小，32位

DWORD dwMaximumSizeHigh,

//32位

DWORD dwMaximumSizeLow,

//文件映射对象的名称，实现文件共享

PCTSTR pszName);

PVOID MapViewOfFile(

//第二步创建的文件映射对象的句柄

HANDLE hFileMappingObject,

//指定内存映射文件的访问权限

DWORD dwDesiredAccess,

//偏移，把文件的数据映射到进程的地址空间的时候是从开始映射还是从某个特定的位置（指定的偏移位置）开始映射

DWORD dwFileOffsetHigh,

//两个偏移都是32位

DWORD dwFileOffsetLow,

//映射文件内容的大小：0表示把文件内容全部映射

SIZE\_T dwNumberOfBytesToMap);

\*/

//如果是一个无效的句柄，则返回错误信息

if ((hFile = CreateFile(strFileName, GENERIC\_READ, FILE\_SHARE\_READ, 0, OPEN\_EXISTING,

FILE\_FLAG\_SEQUENTIAL\_SCAN, 0)) == INVALID\_HANDLE\_VALUE)

{

AfxMessageBox(TEXT("Can not open file"));

return FALSE;

}

//创建文件映射对象

if (!(hMapping = CreateFileMapping(hFile, 0, PAGE\_READONLY, 0, 0, 0)))

{

AfxMessageBox(TEXT("Mapping failed！"));

CloseHandle(hFile);

return FALSE;

}

//将文件的数据映射到进程的地址空间

if (!(basepointer = MapViewOfFile(hMapping, FILE\_MAP\_READ, 0, 0, 0)))

{

AfxMessageBox(TEXT("View failed!"));

CloseHandle(hMapping);

CloseHandle(hFile);

}

m\_strText = (LPTSTR)basepointer;

UnmapViewOfFile(basepointer);

CloseHandle(hMapping);

CloseHandle(hFile);

return TRUE;

}