# 这一节我们来学习递归遍历目录

# 学习大纲

|  |
| --- |
|  |

## FindFirstFile函数的语法

|  |
| --- |
| 语法 HANDLE FindFirstFileA(  [in] LPCSTR lpFileName,  [out] LPWIN32\_FIND\_DATAA lpFindFileData  ); 参数 [in] lpFileName  目录或路径以及文件名。文件名可以包含通配符，例如星号 （\*） 或问号 （？）。  此参数不应为 **NULL**、无效字符串（例如，空字符串 或缺少终止 null 字符的字符串），或以尾部反斜杠 （\） 结尾。  如果字符串以通配符、句点 （.） 或目录名称结尾，则用户必须具有以下权限 路径上的根目录和所有子目录。  默认情况下，名称限制为 MAX\_PATH 个字符。要将此限制扩展到 32,767 个宽字符，请在路径前面加上 “\\？\”。有关更多信息，请参见[命名文件、路径和命名空间](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，您可以选择在不附加“\\？\”的情况下删除MAX\_PATH限制。有关详细信息[**，请参阅命名文件、路径和命名空间**](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)的“最大路径长度限制”部分。  [out] lpFindFileData  指向 [WIN32\_FIND\_DATA](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-win32_find_dataa) 结构的指针，该 接收有关找到的文件或目录的信息。 返回值 如果函数成功，则返回值是后续调用 [FindNextFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-findnextfilea) 或 [FindClose](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-findclose) 时使用的搜索句柄，*并且 lpFindFileData* 参数包含有关第一个文件或目录的信息 发现。  如果函数失败或无法从 *lpFileName* 参数中的搜索字符串中找到文件，则返回值为 **INVALID\_HANDLE\_VALUE** 并且 *lpFindFileData* 的内容为 定。要获取扩展的错误信息，请调用 [GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数。  如果函数因找不到匹配的文件而失败，[则 GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数将返回 **ERROR\_FILE\_NOT\_FOUND**。 言论 **FindFirstFile** 函数将打开一个搜索句柄，并且 返回有关文件系统找到的名称与指定 模式。这可能是也可能不是目录列表应用程序（例如 作为 dir 命令）。这是因为 **FindFirstFile** 不对搜索结果进行排序。为 其他信息，请参阅 [FindNextFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-findnextfilea)。  以下列表标识了一些其他搜索特征：   * 严格对文件名执行搜索，而不是对日期或文件等任何属性执行 类型（有关其他选项，请参阅 [FindFirstFileEx](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-findfirstfileexa)）。 * 搜索包括长文件名和短文件名。 * 尝试打开带有尾部反斜杠的搜索总是失败。 * 为 *lpFileName* 参数传递无效字符串、**NULL** 或空字符串不是此函数的有效使用。在这种情况下，结果是 定义。   **注意**在极少数情况下或在负载较重的系统上，NTFS 文件系统上的文件属性信息可能不是 current 调用此函数时。确保获取当前的 NTFS 文件系统文件 属性，调用 [**GetFileInformationByHandle**](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-getfileinformationbyhandle) 函数。    建立搜索句柄后，您可以使用它来搜索与相同模式匹配的其他文件 通过使用 [FindNextFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-findnextfilea) 函数。  当不再需要搜索句柄时，请使用 [FindClose](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-findclose) 函数（而不是 [CloseHandle](https://learn.microsoft.com/en-us/windows/desktop/api/handleapi/nf-handleapi-closehandle)）将其关闭。  如前所述，不能在 **FindFirstFile** 的 *lpFileName* 输入字符串中使用尾部反斜杠 （\），因此它可能不是 显而易见如何搜索根目录。如果要查看文件或获取根目录的属性，则 以下选项将适用：   * 要检查根目录中的文件，可以使用 “C：\\*” 并逐步执行 [目录中。](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-findnextfilea) * 要获取根目录的属性，请使用 [GetFileAttributes](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-getfileattributesa) 函数。   **注意**在字符串 “\\？\” 前面加上不允许访问根 目录。    在网络共享上，可以采用以下形式使用 *lpFileName*： “\\Server\Share\\*”。但是，不能使用指向共享本身的 *lpFileName*;例如，“\\Server\Share” 无效。  要检查不是根目录的目录，请使用该目录的路径，不要带有尾部 反斜線。例如，参数 “C：\Windows” 返回有关 目录 “C：\Windows”，与 “C：\Windows” 的 Windows 版本。要检查 “C：\Windows”，请使用 *lpFileName* “C：\Windows\\*”。  请注意，在您 查询结果和您对信息执行操作的时间。如果这是应用程序的潜在问题， 一种可能的解决方案是将 [CreateFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-createfilea) 函数与 **CREATE\_NEW**（如果文件存在，则失败）或 **OPEN\_EXISTING**（如果文件不存在，则失败）一起使用。  如果您正在编写一个 32 位应用程序来列出目录中的所有文件，并且该应用程序可能会运行 在 64 位计算机上，应调用 [Wow64DisableWow64FsRedirection](https://learn.microsoft.com/en-us/windows/desktop/api/wow64apiset/nf-wow64apiset-wow64disablewow64fsredirection) 函数 在调用 **FindFirstFile** 之前，并在 最后一次调用 [FindNextFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-findnextfilea)。有关更多信息，请参阅[文件系统重定向器](https://learn.microsoft.com/en-us/windows/desktop/WinProg64/file-system-redirector)。  如果路径指向符号链接，[则 WIN32\_FIND\_DATA](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-win32_find_dataa) 缓冲区包含有关 符号链接，而不是目标。 |

## FindNextFile函数的语法

|  |
| --- |
| 语法 C++复制  BOOL FindNextFileA(  [in] HANDLE hFindFile,  [out] LPWIN32\_FIND\_DATAA lpFindFileData  ); 参数 [in] hFindFile  上一次调用 [FindFirstFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-findfirstfilea) 或 [FindFirstFileEx](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-findfirstfileexa) 函数返回的搜索句柄。  [out] lpFindFileData  指向 [WIN32\_FIND\_DATA](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-win32_find_dataa) 结构的指针 接收有关找到的文件或子目录的信息。 返回值 如果函数成功，则返回值为非零，*并且 lpFindFileData* 参数包含有关找到的下一个文件或目录的信息。  如果函数失败，则返回值为零，并且 *lpFindFileData* 的内容是不确定的。要获取扩展的错误信息，请调用 [GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数。  如果函数由于找不到更多匹配文件而失败，[则 GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数将返回 **ERROR\_NO\_MORE\_FILES**。 言论 此函数使用用于创建在 *hFindFile* 参数中传递的搜索句柄的相同搜索筛选器。有关更多信息，请参阅 [FindFirstFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-findfirstfilea) 和 [FindFirstFileEx](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-findfirstfileexa)。  搜索返回文件的顺序（如字母顺序）无法保证，并且 取决于文件系统。如果必须对数据进行排序， 应用程序必须在获取所有结果后进行排序。  **注意**在极少数情况下或在负载较重的系统上，NTFS 文件系统上的文件属性信息可能不是 current 调用此函数时。确保获取当前的 NTFS 文件系统文件 属性，调用 [**GetFileInformationByHandle**](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-getfileinformationbyhandle) 函数。    此函数返回文件名的顺序取决于文件系统类型。使用 NTFS 文件系统和 CDFS 文件系统，则名称通常按字母顺序返回。使用 FAT 文件系统， 名称通常按照文件写入磁盘的顺序返回，这可能是也可能不是 按字母顺序排列。但是，如前所述，不能保证这些行为。  如果路径指向符号链接，[则 WIN32\_FIND\_DATA](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-win32_find_dataa) 缓冲区包含有关 符号链接，而不是目标。 |

## 这两个函数都需要一个 [WIN32\_FIND\_DATA](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-win32_find_dataa)结构体来保存数据

## [WIN32\_FIND\_DATA](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-win32_find_dataa)的结构如下

|  |
| --- |
| 语法 复制  typedef struct \_WIN32\_FIND\_DATAA {  DWORD dwFileAttributes;  FILETIME ftCreationTime;  FILETIME ftLastAccessTime;  FILETIME ftLastWriteTime;  DWORD nFileSizeHigh;  DWORD nFileSizeLow;  DWORD dwReserved0;  DWORD dwReserved1;  CHAR cFileName[MAX\_PATH];  CHAR cAlternateFileName[14];  DWORD dwFileType; // Obsolete. Do not use.  DWORD dwCreatorType; // Obsolete. Do not use  WORD wFinderFlags; // Obsolete. Do not use  } WIN32\_FIND\_DATAA, \*PWIN32\_FIND\_DATAA, \*LPWIN32\_FIND\_DATAA; 成员 dwFileAttributes  文件的 file 属性。  有关可能的值及其描述，请参阅 [File Attribute Constants](https://learn.microsoft.com/en-us/windows/desktop/FileIO/file-attribute-constants)。  如果任何流**FILE\_ATTRIBUTE\_SPARSE\_FILE** 的文件曾经稀疏。  ftCreationTime  一个 [FILETIME](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构，用于指定文件或 目录。  如果基础文件系统不支持创建时间，则此成员为零。  ftLastAccessTime  [FILETIME](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构。  对于文件，该结构指定上次从可执行文件读取、写入或为可执行文件读取文件的时间。  对于目录，结构指定创建目录的时间。如果底层文件系统执行 不支持上次访问时间，此成员为零。  在 FAT 文件系统上，文件和目录的指定日期都是正确的，但一天中的时间为 始终设置为午夜。  ftLastWriteTime  [FILETIME](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构。  对于文件，该结构指定文件的上次写入、截断或覆盖时间 例如，当使用 [WriteFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-writefile) 或 [SetEndOfFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-setendoffile) 时。日期和时间不是 在更改文件属性或安全描述符时更新。  对于目录，结构指定创建目录的时间。如果底层文件系统执行 不支持上次写入时间，此成员为零。  nFileSizeHigh  文件大小的高阶 **DWORD** 值，以字节为单位。  除非文件大小大于 **MAXDWORD，**否则此值为零。  文件大小等于 （**nFileSizeHigh** \* （**MAXDWORD**+1））+ **nFileSizeLow**。  nFileSizeLow  文件大小的低阶 **DWORD** 值（以字节为单位）。  dwReserved0  如果 **dwFileAttributes** 成员包含 **FILE\_ATTRIBUTE\_REPARSE\_POINT** 属性，则此成员指定重新分析点 标记。  否则，此值为 undefined，不应使用。  有关更多信息，请参阅[重新分析点标签](https://learn.microsoft.com/en-us/windows/desktop/FileIO/reparse-point-tags)。 IO\_REPARSE\_TAG\_CSV （0x80000009）IO\_REPARSE\_TAG\_DEDUP （0x80000013）IO\_REPARSE\_TAG\_DFS （0x8000000A）IO\_REPARSE\_TAG\_DFSR （0x80000012）IO\_REPARSE\_TAG\_HSM （0xC0000004）IO\_REPARSE\_TAG\_HSM2 （0x80000006）IO\_REPARSE\_TAG\_MOUNT\_POINT （0xA0000003）IO\_REPARSE\_TAG\_NFS （0x80000014）IO\_REPARSE\_TAG\_SIS （0x80000007）IO\_REPARSE\_TAG\_SYMLINK （0xA000000C）IO\_REPARSE\_TAG\_WIM （0x80000008） dwReserved1  保留供将来使用。  cFileName[MAX\_PATH]  文件的名称。  cAlternateFileName[14]  文件的备用名称。  此名称采用经典的 8.3 文件名格式。 言论 如果文件具有较长的文件名，则完整名称将显示在 **cFileName** 成员中。 并且名称的 8.3 格式截断版本显示在 **cAlternateFileName** 成员中。否则， **cAlternateFileName** 为空。如果在 *fInfoLevelId* 参数中使用 **FindExInfoBasic** 值调用 [FindFirstFileEx](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-findfirstfileexa) 函数，**则 cAlternateFileName** 成员将始终包含 **NULL** 字符串 价值。对于对 [FindNextFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-findnextfilea) 函数的所有后续调用，情况仍然如此。作为 要检索文件名的 8.3 格式版本，您可以使用 [GetShortPathName](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-getshortpathnamea) 函数。有关 文件名，请参见[文件名、路径和命名空间](https://learn.microsoft.com/en-us/windows/desktop/FileIO/naming-a-file)。  并非所有文件系统都可以记录创建和上次访问时间，也不是所有文件系统都可以将它们记录在 同样的方式。例如，在 FAT 文件系统上，创建时间的分辨率为 10 毫秒，写入时间的分辨率为 10 毫秒 分辨率为 2 秒，Access Time 的分辨率为 1 天。这 NTFS 文件系统将文件的更新延迟到上次访问时间后，最多在上次访问后 1 小时。为 有关更多信息，请参阅 [File Times](https://learn.microsoft.com/en-us/windows/desktop/SysInfo/file-times)。  **注意**  minwinbase.h 标头将 WIN32\_FIND\_DATA 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将 encoding-neutral 别名与非编码中性代码混合使用可能会导致不匹配，从而导致编译或运行时错误。有关更多信息，请参见[**函数原型的约定**](https://learn.microsoft.com/en-us/windows/win32/intl/conventions-for-function-prototypes)。 |

### 可以看到还是使用上一节的api不过这里实现递归遍历而已

# 演练

## 1.新建一个项目，取名：Lesson20-recursive-travel-dir，然后新建一个cpp源文件，取名：recursive-travel-dir.cpp

|  |
| --- |
|  |

## 2.然后我们在上一节课的代码的基础上添加递归功能，完整代码如下

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  DWORD ListDirFiles(LPCTSTR pathName)  {  TCHAR szPath[MAX\_PATH];  TCHAR szSubPath[MAX\_PATH];  TCHAR szFullPath[MAX\_PATH];  HANDLE hFile;  WIN32\_FIND\_DATA w32fd;  lstrcpy(szPath,pathName);  lstrcat(szPath,"\\\*.\*");  hFile = FindFirstFile(szPath,&w32fd);  if(INVALID\_HANDLE\_VALUE == hFile)  {  printf("Find File Failed:%d\n",GetLastError());  return 1;  }  else  {  do  {  //过滤.和..  if((0==lstrcmp(TEXT("."),w32fd.cFileName)) || ((0==lstrcmp(TEXT(".."),w32fd.cFileName))))  {  continue;  }  printf("%s\\%s\t",pathName,w32fd.cFileName); //需要先输出文件全名  if(FILE\_ATTRIBUTE\_DIRECTORY & w32fd.dwFileAttributes)  {  printf("<Dir>\n");  lstrcpy(szFullPath,pathName);//接收跟路径  lstrcpy(szSubPath,w32fd.cFileName);//需要接收自路径，不能直接修改  lstrcat(szFullPath,"\\");//根路径和子路径分隔符  lstrcat(szFullPath,szSubPath); //拼接全路径  //printf("%s\n",szFullPath);//测试用，检测路径拼接对不对  ListDirFiles(szFullPath); //递归调用时，不要在+"\\\*.\*",因为函数一开始就会加上  }  else  {  printf("\n");  }      }while(FindNextFile(hFile,&w32fd));  }  return 0;  }  int main()  {  ListDirFiles(TEXT("E:\\vscode-kotlin"));  system("pause");  return 0;  } |

### 效果：

|  |
| --- |
|  |

# 扩展

## 把这个项目复制一份，改名Lesson20-recursive-travel-dir-argc，我们把它改为命令行参数的形式，为了方便使用，把项目名称改为travel\_argc.代码如下，只需要修改一下main函数即可

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  DWORD ListDirFiles(LPCTSTR pathName)  {  TCHAR szPath[MAX\_PATH];  TCHAR szSubPath[MAX\_PATH];  TCHAR szFullPath[MAX\_PATH];  HANDLE hFile;  WIN32\_FIND\_DATA w32fd;  lstrcpy(szPath,pathName);  lstrcat(szPath,"\\\*.\*");  hFile = FindFirstFile(szPath,&w32fd);  if(INVALID\_HANDLE\_VALUE == hFile)  {  printf("Find File Failed:%d\n",GetLastError());  return 1;  }  else  {  do  {  //过滤.和..  if((0==lstrcmp(TEXT("."),w32fd.cFileName)) || ((0==lstrcmp(TEXT(".."),w32fd.cFileName))))  {  continue;  }  printf("%s\\%s\t",pathName,w32fd.cFileName); //需要先输出文件全名  if(FILE\_ATTRIBUTE\_DIRECTORY & w32fd.dwFileAttributes)  {  printf("<Dir>\n");  lstrcpy(szFullPath,pathName);//接收跟路径  lstrcpy(szSubPath,w32fd.cFileName);//需要接收自路径，不能直接修改  lstrcat(szFullPath,"\\");//根路径和子路径分隔符  lstrcat(szFullPath,szSubPath); //拼接全路径  //printf("%s\n",szFullPath);//测试用，检测路径拼接对不对  ListDirFiles(szFullPath); //递归调用时，不要在+"\\\*.\*",因为函数一开始就会加上  }  else  {  printf("\n");  }      }while(FindNextFile(hFile,&w32fd));  }  return 0;  }  int main(int argc,char \*argv[])  {  if(argc < 2)  {  printf("Usage: travel\_argc pathname\n");  return -1;  }  ListDirFiles(argv[1]);  system("pause");  return 0;  } |

### 效果：

|  |
| --- |
|  |

# 扩展2.宽字符版本

## 1.新建一个空项目,取名: RecursiveListDir,然后添加一个cpp文件,取名: rec-traveldir.cpp

|  |
| --- |
|  |

## 2.rec-traveldir.cpp的代码如下

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<locale.h>  void GetAttribStr(DWORD attr, WCHAR\*\* szAttr)  {  switch (attr)  {  case FILE\_ATTRIBUTE\_DIRECTORY:  lstrcpy(\*szAttr, L"[Directory]");  break;  case FILE\_ATTRIBUTE\_ENCRYPTED:  lstrcpy(\*szAttr, L"[Encrypted]");  break;  case FILE\_ATTRIBUTE\_COMPRESSED:  lstrcpy(\*szAttr, L"[Compressed]");  break;  case FILE\_ATTRIBUTE\_ARCHIVE:  lstrcpy(\*szAttr, L"[Archived]");  break;  default:  lstrcpy(\*szAttr, L"[Unknown]");  break;  }  }  void RecursiveListDir(WCHAR\* path)  {  WCHAR\* szAttr = (WCHAR\*)HeapAlloc(GetProcessHeap(), HEAP\_GENERATE\_EXCEPTIONS | HEAP\_ZERO\_MEMORY, 100);  WIN32\_FIND\_DATA w32fd;  WCHAR szFilePath[MAX\_PATH];  memset(szFilePath, 0, MAX\_PATH);  WCHAR szFullPath[MAX\_PATH]; //递归用的全路径  memset(szFullPath, 0, MAX\_PATH);  WCHAR szSubPath[MAX\_PATH];  memset(szSubPath, 0, MAX\_PATH);  lstrcpy(szFilePath, path);//接收参数,然后拷贝  lstrcat(szFilePath, L"\\\*.\*");//添加通配符表示查找所有  HANDLE hFile;  hFile = FindFirstFile(szFilePath, &w32fd);  if (hFile == INVALID\_HANDLE\_VALUE)  {  wprintf\_s(L"Find First File Failed,code:%d\n", GetLastError());  return;  }  do  {  if (lstrcmp(L".", w32fd.cFileName) != 0 && lstrcmp(L"..", w32fd.cFileName) != 0)  {  GetAttribStr(w32fd.dwFileAttributes, &szAttr);  wprintf\_s(L"%s\t%s\\%s\n", szAttr, path, w32fd.cFileName);  if (FILE\_ATTRIBUTE\_DIRECTORY & w32fd.dwFileAttributes) {  lstrcpy(szFullPath, path);  lstrcpy(szSubPath, w32fd.cFileName);  lstrcat(szFullPath, L"\\");  lstrcat(szFullPath, szSubPath);  RecursiveListDir(szFullPath);  }    }  } while (FindNextFile(hFile, &w32fd));  HeapFree(GetProcessHeap(), HEAP\_ZERO\_MEMORY, szAttr);  }  int wmain(int argc, WCHAR\* argv[])  {  setlocale(LC\_ALL, "");  if (argc < 2)  {  wprintf\_s(L"%s directory name\n", argv[0]);  return -1;  }  //wprintf\_s(L"List Of Files\n");  RecursiveListDir(argv[1]);  return 0;  } |

## 效果

|  |
| --- |
|  |

# 这一节的学习到此为止