# 学习大纲

|  |
| --- |
|  |

# API用法

## AddClipboardFormatListener 函数用法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 在系统维护的剪贴板格式侦听器列表中Places给定窗口。 语法 C++复制  BOOL AddClipboardFormatListener(  [in] HWND hwnd  ); 参数 [in] hwnd  类型：**HWND**  要放置在剪贴板格式侦听器列表中的窗口的句柄。 返回值 类型： **BOOL**  如果成功，则返回 **TRUE** ;否则返回 **FALSE** 。 有关更多详细信息，请调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 。 注解 将窗口添加到剪贴板格式侦听器列表后，每当剪贴板的内容发生更改时，该窗口将发布 [WM\_CLIPBOARDUPDATE](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/wm-clipboardupdate) 消息。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 (Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-misc-l1-5-1) |  另请参阅 [GetClipboardSequenceNumber](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getclipboardsequencenumber)  [RemoveClipboardFormatListener](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-removeclipboardformatlistener)  [WM\_CLIPBOARDUPDATE](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/wm-clipboardupdate) |

## OpenClipboard函数的用法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 打开剪贴板以供检查，并阻止其他应用程序修改剪贴板内容。 语法 C++  BOOL OpenClipboard(  [in, optional] HWND hWndNewOwner  ); 参数 [in, optional] hWndNewOwner  类型：**HWND**  要与打开的剪贴板关联的窗口的句柄。 如果此参数为 **NULL**，则打开的剪贴板与当前任务相关联。 返回值 类型： **BOOL**  如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 注解 如果另一个窗口打开了剪贴板，**OpenClipboard** 将失败。  每次成功调用 [OpenClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-closeclipboard) 后，应用程序都应调用 **CloseClipboard** 函数。  除非调用 [EmptyClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-emptyclipboard) 函数，否则*由 hWndNewOwner* 参数标识的窗口不会成为剪贴板所有者。  如果应用程序调用 **OpenClipboard** 且 hwnd 设置为 **NULL**， [则 EmptyClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-emptyclipboard) 会将剪贴板所有者设置为 **NULL**;这会导致 [SetClipboardData](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-setclipboarddata) 失败。 示例 有关示例，请参阅 [将信息复制到剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/using-the-clipboard)。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 Windows 8.1) 中引入的 ext-ms-win-ntuser-misc-l1-2-0 ( |  另请参阅 [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  [CloseClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-closeclipboard)  **概念性**  [EmptyClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-emptyclipboard)  **引用** |

## ChangeClipboardChain 函数用法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 从剪贴板查看器链中删除指定的窗口。 语法 C++复制  BOOL ChangeClipboardChain(  [in] HWND hWndRemove,  [in] HWND hWndNewNext  ); 参数 [in] hWndRemove  类型：**HWND**  要从链中删除的窗口的句柄。 句柄必须已传递到 [SetClipboardViewer](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-setclipboardviewer) 函数。  [in] hWndNewNext  类型：**HWND**  剪贴板查看器链中 *hWndRemove* 窗口后面的窗口的句柄。 (这是 [SetClipboardViewer](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-setclipboardviewer) 返回的句柄，除非序列已更改以响应 [WM\_CHANGECBCHAIN](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/wm-changecbchain) 消息。) 返回值 类型： **BOOL**  返回值指示将 [WM\_CHANGECBCHAIN](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/wm-changecbchain) 消息传递到剪贴板查看器链中的窗口的结果。 由于链中的窗口在处理**WM\_CHANGECBCHAIN**时通常返回 **FALSE**，**因此 ChangeClipboardChain** 的返回值通常为 **FALSE**。 如果链中只有一个窗口，则返回值通常为 **TRUE**。 注解 *由 hWndNewNext* 标识的窗口替换链中的 *hWndRemove* 窗口。 [SetClipboardViewer](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-setclipboardviewer) 函数将[WM\_CHANGECBCHAIN](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/wm-changecbchain)消息发送到剪贴板查看器链中的第一个窗口。  有关示例，请参阅 [从剪贴板查看器链中删除窗口](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/using-the-clipboard)。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 (Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-misc-l1-5-1) |  另请参阅 [ChangeClipboardChain](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-changeclipboardchain)  [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  **概念性**  **引用**  [SetClipboardViewer](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-setclipboardviewer)  [WM\_CHANGECBCHAIN](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/wm-changecbchain) |

## CloseClipboard函数的用法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 关闭剪贴板。 语法 C++复制  BOOL CloseClipboard(); 返回值 类型： **BOOL**  如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 注解 当窗口完成检查或更改剪贴板后，通过调用 **CloseClipboard** 关闭剪贴板。 这使其他窗口能够访问剪贴板。  调用 **CloseClipboard** 后，请勿在剪贴板上放置对象。 示例 有关示例，请参阅 [剪贴板查看器的示例](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/using-the-clipboard)。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 Windows 8.1) 中引入的 ext-ms-win-ntuser-misc-l1-2-0 ( |  另请参阅 [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  **概念性**  [GetOpenClipboardWindow](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getopenclipboardwindow)  [OpenClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-openclipboard)  **引用** |

## CountClipboardFormats

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索剪贴板上当前不同数据格式的数量。 语法 C++复制  int CountClipboardFormats(); 返回值 类型： **int**  如果函数成功，则返回值是剪贴板上当前不同数据格式的数目。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll |  另请参阅 [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  **概念性**  [EnumClipboardFormats](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-enumclipboardformats)  **引用**  [RegisterClipboardFormat](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-registerclipboardformata) |

## EmptyClipboard 函数

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 清空剪贴板并释放剪贴板中数据的句柄。 然后， 函数将剪贴板的所有权分配给当前已打开剪贴板的窗口。 语法 C++复制  BOOL EmptyClipboard(); 返回值 类型： **BOOL**  如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 注解 在调用 **EmptyClipboard** 之前，应用程序必须使用 [OpenClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-openclipboard) 函数打开剪贴板。 如果应用程序在打开剪贴板时指定 **NULL** 窗口句柄， **EmptyClipboard** 将成功，但将剪贴板所有者设置为 **NULL**。 请注意，这会导致 [SetClipboardData](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-setclipboarddata) 失败。 示例 有关示例，请参阅 [将信息复制到剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/using-the-clipboard)。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 Windows 8.1) 中引入的 ext-ms-win-ntuser-misc-l1-2-0 ( |  另请参阅 [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  **概念性**  [OpenClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-openclipboard)  **引用**  [SetClipboardData](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-setclipboarddata)  [WM\_DESTROYCLIPBOARD](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/wm-destroyclipboard) |

## GetClipboardData函数的用法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 从剪贴板中检索指定格式的数据。 剪贴板之前必须已打开。 语法 C++复制  HANDLE GetClipboardData(  [in] UINT uFormat  ); 参数 [in] uFormat  类型： **UINT**  剪贴板格式。 有关标准剪贴板格式的说明，请参阅 [标准剪贴板格式](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard-formats)。 返回值 类型： **句柄**  如果函数成功，则返回值是指定格式的剪贴板对象的句柄。  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 **谨慎** 剪贴板数据不受信任。 在应用程序中使用数据之前，请仔细分析数据。    应用程序可以使用 [EnumClipboardFormats](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-enumclipboardformats) 函数提前枚举可用的格式。  剪贴板控制 **GetClipboardData** 函数返回的句柄，而不是应用程序。 应用程序应立即复制数据。 应用程序不得释放句柄，也不能将其保持锁定状态。 在调用 [EmptyClipboard 或 CloseClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-emptyclipboard) 函数后，或者在使用相同的剪贴板格式调用 [SetClipboardData](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-setclipboarddata) 函数之后，应用程序不得使用句柄。  当应用程序调用 **GetClipboardData** 函数时，系统会在某些剪贴板格式之间执行隐式数据格式转换。 例如，如果 [CF\_OEMTEXT](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/standard-clipboard-formats) 格式位于剪贴板上，则窗口可以检索 [CF\_TEXT](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/standard-clipboard-formats) 格式的数据。 剪贴板上的格式将按需转换为请求的格式。 有关详细信息，请参阅 [合成剪贴板格式](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard-formats)。 示例 有关示例，请参阅 [将信息复制到剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/using-the-clipboard)。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 Windows 8.1) 中引入的 ext-ms-win-ntuser-misc-l1-2-0 ( |  另请参阅 [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  [CloseClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-closeclipboard)  **概念性**  [EmptyClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-emptyclipboard)  [EnumClipboardFormats](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-enumclipboardformats)  **引用**  [SetClipboardData](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-setclipboarddata) |

## GetClipboardOwner 函数用法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 检索剪贴板当前所有者的窗口句柄。 语法 C++复制  HWND GetClipboardOwner(); 返回值 类型：**HWND**  如果函数成功，则返回值是拥有剪贴板的窗口的句柄。  如果剪贴板不是拥有的，则返回值为 **NULL**。 要获得更多的错误信息，请调用 GetLastError。 注解 即使剪贴板当前不是拥有者，剪贴板仍可以包含数据。  通常，剪贴板所有者是最后一个将数据放入剪贴板的窗口。 [EmptyClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-emptyclipboard) 函数分配剪贴板所有权。 示例 有关示例，请参阅 [剪贴板查看器的示例](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/using-the-clipboard)。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 Windows 8.1) 中引入的 ext-ms-win-ntuser-misc-l1-2-0 ( |  另请参阅 [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  **概念性**  [EmptyClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-emptyclipboard)  [GetClipboardViewer](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getclipboardviewer)  **引用** |

## GetClipboardSequenceNumber 函数

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 检索当前窗口工作站的剪贴板序列号。 语法 C++复制  DWORD GetClipboardSequenceNumber(); 返回值 类型：DWORD  返回值为剪贴板序列号。 如果没有 **WINSTA\_ACCESSCLIPBOARD** 访问窗口工作站，该函数将返回零。 注解 系统为每个窗口工作站保留剪贴板的序列号。 每当剪贴板的内容更改或剪贴板被清空时，此数字将递增。 可以跟踪此值以确定剪贴板内容是否已更改并优化创建 DataObject。 如果剪贴板呈现延迟，则在呈现更改之前，序列号不会递增。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-misc-l1-5-1 () |  另请参阅 [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard) |

## GetClipboardViewer 函数

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 检索剪贴板查看器链中第一个窗口的句柄。 语法 C++  HWND GetClipboardViewer(); 返回值 类型：**HWND**  如果函数成功，则返回值是剪贴板查看器链中第一个窗口的句柄。  如果没有剪贴板查看器，则返回值为 **NULL**。 要获得更多的错误信息，请调用 GetLastError。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-misc-l1-5-1 () |  另请参阅 [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  **概念性**  [GetClipboardOwner](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getclipboardowner)  **引用**  [SetClipboardViewer](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-setclipboardviewer) |

## GetOpenClipboardWindow 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 索当前打开剪贴板的窗口的句柄。 语法 C++复制  HWND GetOpenClipboardWindow(); 返回值 类型：**HWND**  如果函数成功，则返回值是打开剪贴板的窗口的句柄。 如果没有打开剪贴板的窗口，则返回值为 **NULL**。 要获得更多的错误信息，请调用 GetLastError。 注解 如果应用程序或 DLL 在调用 [OpenClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-openclipboard) 函数时指定 **NULL** 窗口句柄，则会打开剪贴板，但不与窗口关联。 在这种情况下， **GetOpenClipboardWindow** 返回 **NULL**。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll |  另请参阅 [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  **概念性**  [GetClipboardOwner](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getclipboardowner)  [GetClipboardViewer](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getclipboardviewer)  [OpenClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-openclipboard)  **引用** |

## GetUpdatedClipboardFormats函数用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 检索当前支持的剪贴板格式。 语法 C++  BOOL GetUpdatedClipboardFormats(  [out] PUINT lpuiFormats,  [in] UINT cFormats,  [out] PUINT pcFormatsOut  ); 参数 [out] lpuiFormats  类型： **PUINT**  剪贴板格式的数组。 有关标准剪贴板格式的说明，请参阅 [标准剪贴板格式](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/standard-clipboard-formats)。  [in] cFormats  类型： **UINT**  *lpuiFormats* 指向的数组中的条目数。  [out] pcFormatsOut  类型： **PUINT**  *lpuiFormats* 指向的数组中剪贴板格式的实际数量。 返回值 类型： **BOOL**  如果成功，函数返回 **TRUE** ;否则为 **FALSE**。 有关更多详细信息，请调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 。 要求 展开表   | **要求** | **值** | | --- | --- | | **最低受支持的客户端** | Windows Vista [仅限桌面应用] | | **最低受支持的服务器** | Windows Server 2008 [仅限桌面应用] | | **目标平台** | Windows | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | |

## IsClipboardFormatAvailable 函数

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 确定剪贴板是否包含指定格式的数据。 语法 C++复制  BOOL IsClipboardFormatAvailable(  [in] UINT format  ); 参数 [in] format  类型： **UINT**  标准或已注册的剪贴板格式。 有关标准剪贴板格式的说明，请参阅 [标准剪贴板格式](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/standard-clipboard-formats) 。 返回值 类型： **BOOL**  如果剪贴板格式可用，则返回值为非零值。  如果剪贴板格式不可用，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 注解 通常，仅识别一种剪贴板格式的应用程序会在处理 [WM\_INITMENU](https://learn.microsoft.com/zh-cn/windows/desktop/menurc/wm-initmenu) 或 [WM\_INITMENUPOPUP](https://learn.microsoft.com/zh-cn/windows/desktop/menurc/wm-initmenupopup) 消息时调用此函数。 然后，应用程序将启用或禁用“粘贴”菜单项，具体取决于返回值。 识别多种剪贴板格式的应用程序应为此使用 [GetPriorityClipboardFormat](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getpriorityclipboardformat) 函数。 示例 有关示例，请参阅 [粘贴剪贴板中的信息](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/using-the-clipboard)。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 Windows 8.1) 中引入的 ext-ms-win-ntuser-misc-l1-2-0 ( |  另请参阅 [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  **概念性**  [CountClipboardFormats](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-countclipboardformats)  [EnumClipboardFormats](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-enumclipboardformats)  [GetPriorityClipboardFormat](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getpriorityclipboardformat)  **引用**  [RegisterClipboardFormat](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-registerclipboardformata)  [WM\_INITMENU](https://learn.microsoft.com/zh-cn/windows/desktop/menurc/wm-initmenu)  [WM\_INITMENUPOPUP](https://learn.microsoft.com/zh-cn/windows/desktop/menurc/wm-initmenupopup) |

## GetPriorityClipboardFormat 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 检索指定列表中的第一个可用剪贴板格式。 语法 C++复制  int GetPriorityClipboardFormat(  [in] UINT \*paFormatPriorityList,  [in] int cFormats  ); 参数 [in] paFormatPriorityList  类型： **UINT\***  剪贴板格式（按优先级顺序排列）。 有关标准剪贴板格式的说明，请参阅 [标准剪贴板格式](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/standard-clipboard-formats) 。  [in] cFormats  类型： **int**  *paFormatPriorityList* 数组中的条目数。 此值不得大于列表中的条目数。 返回值 类型： **int**  如果函数成功，则返回值是列表中第一个提供数据的剪贴板格式。 如果剪贴板为空，则返回值为 NULL。 如果剪贴板包含数据，但不包含任何指定格式的数据，则返回值为 –1。 要获得更多的错误信息，请调用 GetLastError。 要求 展开表   |  |  | | --- | --- | | **最低受支持的客户端** | Windows 2000 Professional [仅限桌面应用] | | **最低受支持的服务器** | Windows 2000 Server [仅限桌面应用] | | **目标平台** | Windows | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 (Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-misc-l1-5-1) |  另请参阅 [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  **概念性**  [CountClipboardFormats](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-countclipboardformats)  [EnumClipboardFormats](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-enumclipboardformats)  [GetClipboardFormatName](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getclipboardformatnamea)  [IsClipboardFormatAvailable](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-isclipboardformatavailable)  **引用**  [RegisterClipboardFormat](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-registerclipboardformata) |

## SetClipboardViewer函数的语法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将指定的窗口添加到剪贴板查看器链。 每当剪贴板内容发生更改时，剪贴板查看器窗口都会收到 [WM\_DRAWCLIPBOARD](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/wm-drawclipboard) 消息。 此函数用于向后兼容早期版本的 Windows。 语法 C++  HWND SetClipboardViewer(  [in] HWND hWndNewViewer  ); 参数 [in] hWndNewViewer  类型：**HWND**  要添加到剪贴板链的窗口的句柄。 返回值 类型：**HWND**  如果函数成功，则返回值将标识剪贴板查看器链中的下一个窗口。 如果发生错误或剪贴板查看器链中没有其他窗口，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 属于剪贴板查看器链的窗口（称为剪贴板查看器窗口）必须 [WM\_CHANGECBCHAIN](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/wm-changecbchain) 和 [WM\_DRAWCLIPBOARD](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/wm-drawclipboard)处理剪贴板消息。 每个剪贴板查看器窗口调用 [SendMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-sendmessage) 函数，以将这些消息传递到剪贴板查看器链中的下一个窗口。  剪贴板查看器窗口必须最终通过调用 [ChangeClipboardChain](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-changeclipboardchain) 函数（例如，响应 [WM\_DESTROY](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/wm-destroy) 消息）从剪贴板查看器链中删除自身。  **SetClipboardViewer** 函数的存在可提供与早期版本的 Windows 的向后兼容性。 无法正确处理剪贴板链消息的应用程序可能会破坏剪贴板查看器链。 新应用程序应使用更可靠的技术，例如剪贴板序列号或剪贴板格式侦听器的注册。 有关这些替代方法的更多详细信息，请参阅 [监视剪贴板内容](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/using-the-clipboard)。 示例 有关示例，请参阅 [将窗口添加到剪贴板查看器链](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/using-the-clipboard)。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 (Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-misc-l1-5-1) |  另请参阅 [ChangeClipboardChain](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-changeclipboardchain)  [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  **概念性**  [GetClipboardViewer](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getclipboardviewer)  **引用**  [SendMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-sendmessage) |

## SetClipboardData函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将数据以指定的剪贴板格式放置在剪贴板上。 窗口必须是当前剪贴板所有者，并且应用程序必须已调用 [**OpenClipboard**](https://learn.microsoft.com/zh-cn/windows/win32/api/winuser/nf-winuser-openclipboard) 函数。 (响应[WM\_RENDERFORMAT](https://learn.microsoft.com/zh-cn/windows/win32/dataxchg/wm-renderformat)消息时，剪贴板所有者不得再调用 **SetClipboardData**.) 语法 C++  HANDLE SetClipboardData(  [in] UINT uFormat,  [in, optional] HANDLE hMem  ); 参数 [in] uFormat  类型： **UINT**  剪贴板格式。 此参数可以是注册格式或任何标准剪贴板格式。 有关详细信息，请参阅 [标准剪贴板格式](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/standard-clipboard-formats) 和 [已注册的剪贴板格式](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard-formats)。   | **常量/值** | **说明** | | --- | --- | | **CF\_BITMAP**  2 | 位图 (**HBITMAP**) 的句柄。 | | **CF\_DIB**  8 | 包含 [**BITMAPINFO**](https://learn.microsoft.com/zh-cn/windows/win32/api/wingdi/ns-wingdi-bitmapinfo) 结构的内存对象，后跟位图位。 | | **CF\_DIBV5**  17 | 包含 [**BITMAPV5HEADER**](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/ns-wingdi-bitmapv5header) 结构的内存对象，后跟位图颜色空间信息和位图位。 | | **CF\_DIF**  5 | Software Arts 的数据交换格式。 | | **CF\_DSPBITMAP**  0x0082 | 与专用格式关联的位图显示格式。 *hMem* 参数必须是可以以位图格式显示的数据的句柄，而不是专用格式的数据。 | | **CF\_DSPENHMETAFILE**  0x008E | 与专用格式关联的增强型图元文件显示格式。 *hMem* 参数必须是可以以增强型图元文件格式显示的数据的句柄，而不是专用格式的数据。 | | **CF\_DSPMETAFILEPICT**  0x0083 | 与专用格式关联的图元文件图片显示格式。 *hMem* 参数必须是可以以图元文件图片格式显示的数据的句柄，而不是专用格式的数据。 | | **CF\_DSPTEXT**  0x0081 | 与专用格式关联的文本显示格式。 *hMem* 参数必须是可以以文本格式显示的数据的句柄，而不是专用格式的数据。 | | **CF\_ENHMETAFILE**  14 | 增强型图元文件的句柄 (**HENHMETAFILE**) 。 | | **CF\_GDIOBJFIRST**  0x0300 | 应用程序定义的 GDI 对象剪贴板格式的整数值范围的开头。 范围的末尾为 **CF\_GDIOBJLAST**。 清空剪贴板时，不会使用 [**GlobalFree**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalfree) 函数自动删除与此范围内剪贴板格式关联的句柄。 此外，在此范围内使用值时， *hMem* 参数不是 GDI 对象的句柄，而是由 [**GlobalAlloc**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 函数使用 **GMEM\_MOVEABLE** 标志分配的句柄。 | | **CF\_GDIOBJLAST**  0x03FF | 请参阅 **CF\_GDIOBJFIRST**。 | | **CF\_HDROP**  15 | 类型 **HDROP** 的句柄，用于标识文件列表。 应用程序可以通过将句柄传递给 [**DragQueryFile**](https://learn.microsoft.com/zh-cn/windows/win32/api/shellapi/nf-shellapi-dragqueryfilea) 函数来检索有关文件的信息。 | | **CF\_LOCALE**  16 | 数据是 HGLOBAL () 与剪贴板中的文本关联的区域设置标识符 ([LCID](https://learn.microsoft.com/zh-cn/windows/win32/intl/locale-identifiers)) 的句柄。 关闭剪贴板时，如果剪贴板包含 **CF\_TEXT** 数据但没有 **CF\_LOCALE** 数据，系统会自动将 **CF\_LOCALE** 格式设置为当前输入语言。 可以使用 **CF\_LOCALE** 格式将不同的区域设置与剪贴板文本相关联。 从剪贴板粘贴文本的应用程序可以检索此格式，以确定用于生成文本的字符集。 请注意，剪贴板不支持多个字符集中的纯文本。 若要实现此目的，请改用带格式的文本数据类型，例如 RTF。 系统使用与 **CF\_LOCALE** 关联的代码页从 **CF\_TEXT** 隐式转换为 **CF\_UNICODETEXT**。 因此，使用正确的代码页表进行转换。 | | **CF\_METAFILEPICT**  3 | [**METAFILEPICT**](https://learn.microsoft.com/zh-cn/windows/win32/api/wingdi/ns-wingdi-metafilepict) 结构定义的图元文件图片格式的句柄。 通过 DDE 传递 **CF\_METAFILEPICT** 句柄时，负责删除 *hMem* 的应用程序还应释放 **CF\_METAFILEPICT** 句柄引用的图元文件。 | | **CF\_OEMTEXT**  7 | 包含 OEM 字符集中字符的文本格式。 每行以回车符/换行符 (CR-LF) 组合结束。 null 字符表示数据结束。 | | **CF\_OWNERDISPLAY**  0x0080 | 所有者显示格式。 剪贴板所有者必须显示和更新剪贴板查看器窗口，并接收 [**WM\_ASKCBFORMATNAME**](https://learn.microsoft.com/zh-cn/windows/win32/dataxchg/wm-askcbformatname)、 [**WM\_HSCROLLCLIPBOARD**](https://learn.microsoft.com/zh-cn/windows/win32/dataxchg/wm-hscrollclipboard)、 [**WM\_PAINTCLIPBOARD**](https://learn.microsoft.com/zh-cn/windows/win32/dataxchg/wm-paintclipboard)、 [**WM\_SIZECLIPBOARD**](https://learn.microsoft.com/zh-cn/windows/win32/dataxchg/wm-sizeclipboard)和 [**WM\_VSCROLLCLIPBOARD**](https://learn.microsoft.com/zh-cn/windows/win32/dataxchg/wm-vscrollclipboard) 消息。 *hMem* 参数必须为 **NULL**。 | | **CF\_PALETTE**  9 | 调色板的句柄。 每当应用程序在依赖于或假定调色板的剪贴板中放置数据时，它也应该将调色板放在剪贴板上。 如果剪贴板包含 **CF\_PALETTE** (逻辑调色板) 格式的数据，则应用程序应使用 [**SelectPalette**](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-selectpalette) 和 [**RealizePalette**](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-realizepalette) 函数实现 (剪贴板中) 任何其他数据与该逻辑调色板进行比较。 显示剪贴板数据时，剪贴板始终使用剪贴板上采用 **CF\_PALETTE** 格式的任何对象作为其当前调色板。 | | **CF\_PENDATA**  10 | Microsoft Windows for Pen Computing 的笔扩展的数据。 | | **CF\_PRIVATEFIRST**  0x0200 | 专用剪贴板格式的整数值范围的开头。 范围以 **CF\_PRIVATELAST**结尾。 与专用剪贴板格式关联的句柄不会自动释放;剪贴板所有者必须释放此类句柄，通常是为了响应 [**WM\_DESTROYCLIPBOARD**](https://learn.microsoft.com/zh-cn/windows/win32/dataxchg/wm-destroyclipboard) 消息。 | | **CF\_PRIVATELAST**  0x02FF | 请参阅 **CF\_PRIVATEFIRST**。 | | **CF\_RIFF**  11 | 表示的音频数据比以 **CF\_WAVE** 标准波形格式表示的音频数据更为复杂。 | | **CF\_SYLK**  4 | Microsoft 符号链接 (SYLK) 格式。 | | **CF\_TEXT**  1 | 文本格式。 每行以回车符/换行符 (CR-LF) 组合结束。 null 字符表示数据结束。 将此格式用于 ANSI 文本。 | | **CF\_TIFF**  6 | 标记图像文件格式。 | | **CF\_UNICODETEXT**  13 | Unicode 文本格式。 每行以回车符/换行符 (CR-LF) 组合结束。 null 字符表示数据结束。 | | **CF\_WAVE**  12 | 表示其中一种标准波形的音频数据，例如 11 kHz 或 22 kHz PCM。 |   [in, optional] hMem  类型： **HANDLE**  指定格式的数据的句柄。 此参数可以为 **NULL**，表示窗口以指定的剪贴板格式提供数据， (请求时呈现格式) ;这称为 [延迟呈现](https://learn.microsoft.com/zh-cn/windows/win32/dataxchg/clipboard-operations#delayed-rendering)。 如果窗口延迟呈现，则必须处理 [WM\_RENDERFORMAT](https://learn.microsoft.com/zh-cn/windows/win32/dataxchg/wm-renderformat) 并 [WM\_RENDERALLFORMATS](https://learn.microsoft.com/zh-cn/windows/win32/dataxchg/wm-renderallformats) 消息。  如果 **SetClipboardData** 成功，则系统拥有 *由 hMem* 参数标识的对象。 将所有权转让给系统后，应用程序可能不会写入或释放数据，但在调用 [CloseClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-closeclipboard) 函数之前，应用程序可以锁定数据并从中读取数据。 (必须在关闭剪贴板之前解锁内存。) 如果 *hMem* 参数标识内存对象，则必须使用带有 **GMEM\_MOVEABLE** 标志的 函数分配对象。 返回值 类型： **HANDLE**  如果函数成功，则返回值是数据的句柄。  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 **Windows 8：** 要与 Windows 应用商店应用共享的位图必须采用与设备相关的 **位** 图) (CF\_BITMAP格式。  如果应用程序调用 **SetClipboardData** 以响应 [WM\_RENDERFORMAT](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/wm-renderformat) 或 [WM\_RENDERALLFORMATS](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/wm-renderallformats)，则调用 **SetClipboardData** 后，应用程序不应使用句柄。  如果应用程序调用 [OpenClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-openclipboard) 且 hwnd 设置为 **NULL**， [则 EmptyClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-emptyclipboard) 会将剪贴板所有者设置为 **NULL**;这会导致 **SetClipboardData** 失败。  当应用程序调用 [GetClipboardData](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getclipboarddata) 函数时，系统会在某些剪贴板格式之间执行隐式数据格式转换。 例如，如果 **CF\_OEMTEXT** 格式位于剪贴板上，则窗口可以检索 **CF\_TEXT** 格式的数据。 剪贴板上的格式将按需转换为请求的格式。 有关详细信息，请参阅 [合成剪贴板格式](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard-formats)。 示例 有关示例，请参阅 [将信息复制到剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/using-the-clipboard)。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 Windows 8.1) 中引入的 ext-ms-win-ntuser-misc-l1-2-0 ( |  另请参阅 [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  [CloseClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-closeclipboard)  **概念性**  [GetClipboardData](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getclipboarddata)  [OpenClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-openclipboard) |

## EnumClipboardFormats函数用法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 枚举剪贴板上当前可用的数据格式。  剪贴板数据格式存储在有序列表中。 若要执行剪贴板数据格式的枚举，需要对 **EnumClipboardFormats** 函数进行一系列调用。 对于每次调用， *format* 参数指定可用的剪贴板格式，函数返回下一个可用的剪贴板格式。 语法 C++  UINT EnumClipboardFormats(  [in] UINT format  ); 参数 [in] format  类型： **UINT**  已知可用的剪贴板格式。  若要启动剪贴板格式的枚举，请将 *格式* 设置为零。 当 *format* 为零时，函数将检索第一个可用的剪贴板格式。 对于枚举期间的后续调用，请将 *格式* 设置为上一个 **EnumClipboardFormats** 调用的结果。 返回值 类型： **UINT**  如果函数成功，则返回值是遵循指定格式的剪贴板格式，即下一个可用的剪贴板格式。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 如果剪贴板未打开，函数将失败。  如果没有其他要枚举的剪贴板格式，则返回值为零。 在这种情况下， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数 **ERROR\_SUCCESS**返回值。 这使你可以区分函数失败和枚举结束。 注解 在枚举剪贴板格式之前，必须打开剪贴板。 使用 [OpenClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-openclipboard) 函数打开剪贴板。 如果剪贴板未打开， **则 EnumClipboardFormats** 函数失败。  **EnumClipboardFormats** 函数按格式放置在剪贴板上的顺序枚举格式。 如果要将信息复制到剪贴板，请按从最具描述性剪贴板格式到描述性最低的剪贴板格式的顺序添加剪贴板对象。 如果要粘贴剪贴板中的信息，请检索可以处理的第一种剪贴板格式。 这是可以处理的最具描述性的剪贴板格式。  系统为某些剪贴板格式提供自动类型转换。 对于此类格式，此函数枚举指定的格式，然后枚举可以转换为的格式。 有关详细信息，请参阅 [标准剪贴板格式](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard-formats) 和 [合成剪贴板格式](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard-formats)。 示例 有关示例，请参阅 [剪贴板查看器的示例](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/using-the-clipboard)。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 (Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-misc-l1-5-1) |  另请参阅 [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  **概念性**  [CountClipboardFormats](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-countclipboardformats)  [OpenClipboard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-openclipboard)  **引用**  [RegisterClipboardFormat](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-registerclipboardformata) |

## RegisterClipboartFormat函数用法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 注册新的剪贴板格式。 然后，可以将此格式用作有效的剪贴板格式。 语法 C++复制  UINT RegisterClipboardFormatA(  [in] LPCSTR lpszFormat  ); 参数 [in] lpszFormat  类型： **LPCTSTR**  新格式的名称。 返回值 类型： **UINT**  如果函数成功，则返回值标识已注册的剪贴板格式。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 注解 如果已存在具有指定名称的已注册格式，则不会注册新格式，并且返回值标识现有格式。 这使多个应用程序能够使用相同的注册剪贴板格式复制和粘贴数据。 请注意，格式名称比较不区分大小写。  注册的剪贴板格式由0xC000到0xFFFF范围内的值标识。  当注册的剪贴板格式放置在剪贴板上或从剪贴板中检索时，它们必须采用 **HGLOBAL** 值的形式。 示例 有关示例，请参阅 [注册剪贴板格式](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/using-the-clipboard)。  **备注**  winuser.h 标头将 RegisterClipboardFormat 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。 将非特定编码别名与非非特定编码的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。 有关详细信息，请参阅 [**函数原型的约定**](https://learn.microsoft.com/zh-cn/windows/win32/intl/conventions-for-function-prototypes)。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 Windows 8.1) 中引入的 ext-ms-win-ntuser-misc-l1-2-0 ( |  另请参阅 [剪贴板](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/clipboard)  **概念性**  [CountClipboardFormats](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-countclipboardformats)  [EnumClipboardFormats](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-enumclipboardformats)  [GetClipboardFormatName](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getclipboardformatnamea)  **引用** |

## RemoveClipboardFormatListener 函数用法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 从系统维护的剪贴板格式侦听器列表中删除给定窗口。 语法 C++复制  BOOL RemoveClipboardFormatListener(  [in] HWND hwnd  ); 参数 [in] hwnd  类型：**HWND**  要从剪贴板格式侦听器列表中删除的窗口的句柄。 返回值 类型： **BOOL**  如果成功，则返回 **TRUE** ;否则返回 **FALSE** 。 有关更多详细信息，请调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 。 注解 从剪贴板格式侦听器列表中删除窗口后，它将不再接收 [WM\_CLIPBOARDUPDATE](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/wm-clipboardupdate) 消息。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 (Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-misc-l1-5-1) |  另请参阅 [AddClipboardFormatListener](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-addclipboardformatlistener)  [GetClipboardSequenceNumber](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getclipboardsequencenumber)  [WM\_CLIPBOARDUPDATE](https://learn.microsoft.com/zh-cn/windows/desktop/dataxchg/wm-clipboardupdate) |

## GlobalAlloc函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 从堆中分配指定的字节数。  **注意** 与其他内存管理功能相比，全局函数具有更大的开销，并且提供的功能更少。 除非文档指出应使用全局函数，否则新的应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局函数和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++  DECLSPEC\_ALLOCATOR HGLOBAL GlobalAlloc(  [in] UINT uFlags,  [in] SIZE\_T dwBytes  ); 参数 [in] uFlags  内存分配属性。 如果指定零，则默认值为 **GMEM\_FIXED**。 此参数可以是以下一个或多个值，但专门指出的不兼容组合除外。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **GHND**  0x0042 | 将 **GMEM\_MOVEABLE** 和GMEM\_ZEROINIT组合在 **一**起。 | | **GMEM\_FIXED**  0x0000 | 分配固定内存。 返回值为指针。 | | **GMEM\_MOVEABLE**  0x0002 | 分配可移动内存。 内存块永远不会在物理内存中移动，但它们可以在默认堆中移动。  返回值是内存对象的句柄。 若要将句柄转换为指针，请使用 [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock) 函数。  此值不能与 **GMEM\_FIXED** 组合使用。 | | **GMEM\_ZEROINIT**  0x0040 | 将内存内容初始化为零。 | | **GPTR**  0x0040 | 将 **GMEM\_FIXED** 和GMEM\_ZEROINIT组合在 **一**起。 |     以下值已过时，但提供这些值是为了兼容 16 位 Windows。 忽略它们。  **GMEM\_DDESHARE**  **GMEM\_DISCARDABLE**  **GMEM\_LOWER**  **GMEM\_NOCOMPACT**  **GMEM\_NODISCARD**  **GMEM\_NOT\_BANKED**  **GMEM\_NOTIFY**  **GMEM\_SHARE**  [in] dwBytes  要分配的字节数。 如果此参数为零，并且 *uFlags* 参数指定 **GMEM\_MOVEABLE**，则函数将返回一个标记为已放弃的内存对象的句柄。 返回值 如果函数成功，则返回值是新分配的内存对象的句柄。  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 Windows 内存管理不提供单独的本地堆和全局堆。 因此， **GlobalAlloc** 和 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc) 函数本质上是相同的。  可移动内存标志 **GHND** 和 **GMEM\_MOVABLE** 增加了不必要的开销，并且需要锁定才能安全使用。 除非文档明确指出应使用它们，否则应避免使用它们。  除非文档明确指出应使用全局函数，否则新应用程序应使用 [堆函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 来分配和管理内存。 例如，全局函数仍与 Dynamic Data Exchange (DDE) 、剪贴板函数和 OLE 数据对象一起使用。  如果 **GlobalAlloc** 函数成功，则至少分配请求的内存量。 如果实际分配的金额大于请求的金额，则进程可以使用整个金额。 若要确定实际分配的字节数，请使用 [GlobalSize](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalsize) 函数。  如果堆中没有足够的可用空间来满足请求， **则 GlobalAlloc** 返回 **NULL**。 由于 **NULL** 用于指示错误，因此永远不会分配虚拟地址零。 因此，很容易检测 **NULL** 指针的使用。  使用此函数分配的内存保证在 8 字节边界上对齐。 若要执行动态生成的代码，请使用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc) 函数分配内存，使用 [VirtualProtect](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualprotect) 函数授予 **PAGE\_EXECUTE** 访问权限。  若要释放内存，请使用 [GlobalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalfree) 函数。 使用 [LocalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localfree) 释放 **GlobalAlloc** 分配的内存是不安全的。 示例 以下代码演示 **了 GlobalAlloc** 和 [GlobalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalfree) 的简单用法。  C++复制  #include <windows.h>  #include <stdio.h>  #include <tchar.h>  void \_cdecl main()  {  PSECURITY\_DESCRIPTOR pSD;  pSD = (PSECURITY\_DESCRIPTOR) GlobalAlloc(  GMEM\_FIXED,  sizeof(PSECURITY\_DESCRIPTOR));  // Handle error condition  if( pSD == NULL )  {  \_tprintf(TEXT("GlobalAlloc failed (%d)\n"), GetLastError());  return;  }  //see how much memory was allocated  \_tprintf(TEXT("GlobalAlloc allocated %d bytes\n"), GlobalSize(pSD));  // Use the memory allocated  // Free the memory when finished with it  GlobalFree(pSD);  } 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalDiscard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globaldiscard)  [GlobalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalfree)  [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock)  [GlobalSize](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalsize)  [堆函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## GlobalLock函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 锁定全局内存对象并返回指向对象内存块的第一个字节的指针。  **注意** 与其他内存管理功能相比，全局函数的开销更大，提供的功能更少。 除非文档指出应使用全局函数，否则新应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++复制  LPVOID GlobalLock(  [in] HGLOBAL hMem  ); 参数 [in] hMem  全局内存对象的句柄。 此句柄由 [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 或 [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc) 函数返回。 返回值 如果函数成功，则返回值是指向内存块的第一个字节的指针。  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 每个内存对象的内部数据结构包括最初为零的锁计数。 对于可移动内存对象， **GlobalLock** 将计数递增 1， [GlobalUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalunlock) 函数将计数递减 1。 进程对对象的 **GlobalLock** 进行的每个成功调用都必须通过对 **GlobalUnlock** 的相应调用进行匹配。 除非使用 [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc) 函数重新分配内存对象，否则不会移动或丢弃锁定的内存。 锁定的内存对象的内存块将保持锁定状态，直到其锁计数减为零，此时可以移动或丢弃它。  使用 **GMEM\_FIXED** 分配的内存对象的锁计数始终为零。 对于这些对象，返回的指针的值等于指定句柄的值。  如果指定的内存块已被丢弃，或者内存块的大小为零字节，则此函数返回 **NULL**。  放弃的对象始终具有锁计数为零。 要求 展开表   |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll | |  |  | |  |  | |  |  |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc)  [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc)  [GlobalUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalunlock)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## GlobalUnlock函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 递减与使用 **GMEM\_MOVEABLE** 分配的内存对象关联的锁计数。 此函数对使用 **GMEM\_FIXED** 分配的内存对象没有影响。  **注意** 与其他内存管理功能相比，全局函数具有更大的开销，并且提供的功能更少。 除非文档指出应使用全局函数，否则新的应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局函数和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++复制  BOOL GlobalUnlock(  [in] HGLOBAL hMem  ); 参数 [in] hMem  全局内存对象的句柄。 此句柄由 [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 或 [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc) 函数返回。 返回值 如果内存对象在减少锁计数后仍被锁定，则返回值为非零值。 如果内存对象在减少锁计数后解锁，则函数返回零， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)**返回NO\_ERROR**。  如果函数失败，则返回值为零， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **NO\_ERROR**以外的值。 注解 每个内存对象的内部数据结构包括最初为零的锁计数。 对于可移动内存对象， [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock) 函数将计数递增 1， **GlobalUnlock** 将计数递减 1。 对于进程对 **对象的 GlobalLock** 进行的每个调用，它最终必须调用 **GlobalUnlock**。 除非使用 [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc) 函数重新分配内存对象，否则不会移动或丢弃锁定的内存。 锁定的内存对象的内存块将保持锁定状态，直到其锁定计数减为零，此时可以移动或放弃该内存块。  使用 **GMEM\_FIXED** 分配的内存对象始终具有零锁计数。 如果指定的内存块是固定内存，则此函数返回 **TRUE**。  如果内存对象已解锁，**GlobalUnlock** 将返回 **FALSE**，**ERROR\_NOT\_LOCKED**[GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 报告。  进程不应依赖返回值来确定它随后必须为内存对象调用 **GlobalUnlock** 的次数。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc)  [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock)  [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## GlobalReAlloc 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 更改指定全局内存对象的大小或属性。 大小可以增加或减少。  **注意** 与其他内存管理功能相比，全局函数的开销更大，提供的功能更少。 除非文档指出应使用全局函数，否则新应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++复制  DECLSPEC\_ALLOCATOR HGLOBAL GlobalReAlloc(  [in] \_Frees\_ptr\_ HGLOBAL hMem,  [in] SIZE\_T dwBytes,  [in] UINT uFlags  ); 参数 [in] hMem  要重新分配的全局内存对象的句柄。 此句柄由 [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 或 **GlobalReAlloc** 函数返回。  [in] dwBytes  内存块的新大小（以字节为单位）。 如果 *uFlags* 指定 **GMEM\_MODIFY**，则忽略此参数。  [in] uFlags  重新分配选项。 如果指定 **了GMEM\_MODIFY** ，则函数仅在忽略 *dwBytes* 参数 (修改内存对象的属性。) 否则，函数将重新分配内存对象。  可以选择将 **GMEM\_MODIFY** 与以下值组合在一起。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **GMEM\_MOVEABLE**  0x0002 | 分配可移动内存。  如果内存是锁定 **GMEM\_MOVEABLE** 内存块或 **GMEM\_FIXED** 内存块，并且未指定此标志，则只能就地重新分配内存。 |     如果此参数未指定 **GMEM\_MODIFY**，则可以使用以下值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **GMEM\_ZEROINIT**  0x0040 | 如果内存对象的大小增大，则会导致其他内存内容初始化为零。 |  返回值 如果函数成功，则返回值是重新分配的内存对象的句柄。  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 如果 **GlobalReAlloc** 重新分配可移动对象，则返回值是内存对象的句柄。 若要将句柄转换为指针，请使用 [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock) 函数。  如果 **GlobalReAlloc** 重新分配固定对象，则返回的句柄值是内存块的第一个字节的地址。 若要访问内存，进程只需将返回值强制转换为指针。  如果 **GlobalReAlloc** 失败，则不会释放原始内存，并且原始句柄和指针仍然有效。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll | | 另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc)  [GlobalDiscard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globaldiscard)  [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |  | |

# 演练

## 1.新建一个文件夹Lesson68-proc-comm-cliboard，然后新建一个MFC对话框项目取名ClipboardDataSender，然后在这个解决方案里面新建一个MFC对话框项目，取名：ClipboardDataReceiver,把这两个项目的对话框默认控件全部删除，然后重写PreTranslateMessage函数用来屏蔽回车键和esc键

|  |
| --- |
|  |

## 2.给发送程序的对话框添加一个静态文本，提示用户输入数据，然后添加一个编辑框控件，可以拉大一点，设置允许多行，用垂直滚动条和一个按钮

|  |
| --- |
|  |
|  |
|  |
|  |

## 3.给编辑框关联一个对话框类成员变量

|  |
| --- |
|  |

## 4.先添加一些内容为空的出来代码

|  |
| --- |
| void CClipboardDataSenderDlg::OnBnClickedBtnSend()  {  // TODO: 在此添加控件通知处理程序代码  CString str;  GetDlgItemText(IDC\_EDIT\_DATA,str);  if(str.GetLength()==0)  {  MessageBox(\_T("发送内容不能为空！！"));  m\_editData.SetFocus();  return;  }  MessageBox(str);  } |

### 运行程序，效果如下

|  |
| --- |
|  |
|  |

## 5.给接收应用程序添加一个按钮和一个编辑框控件，再添加一个保存数据到文件的按钮，默认是禁用的因为一开始没有数据，接收到数据后需要启用它

|  |
| --- |
|  |
|  |
|  |

## 6.然后给编辑框和保存按钮关联成员变量，方便我们编程

|  |
| --- |
|  |
|  |

## 7.回到发送端程序，我们来完成发送代码

|  |
| --- |
| void CClipboardDataSenderDlg::OnBnClickedBtnSend()  {  // TODO: 在此添加控件通知处理程序代码  CString str;  GetDlgItemText(IDC\_EDIT\_DATA,str);  if(str.GetLength()==0)  {  MessageBox(\_T("发送内容不能为空！！"));  m\_editData.SetFocus();  return;  }  //MessageBox(str);  //发送数据  //1.操作之前需要先打开剪贴板  if(!OpenClipboard())  {  MessageBox(\_T("Open Clipboard Failed！！"));  return;  }  //发送之前先把剪贴板清空  EmptyClipboard();  //动态分配内存  HANDLE hMem;  hMem = GlobalAlloc(GMEM\_MOVEABLE,(str.GetLength()+1)\*sizeof(TCHAR));//在当前进程的动态堆上分配内存  //因为上面的内存是可以移动的，我们需要把它锁住  TCHAR\* pData;  pData = (TCHAR\*)GlobalLock(hMem);  //拷贝数据到上面的内存  \_tcscpy(pData,str);  //解锁上面的锁定内存  GlobalUnlock(hMem);  //把数据设置到剪贴板  SetClipboardData(CF\_TEXT,hMem);  //关闭剪贴板  CloseClipboard();    } |

## 8.下面我们来做接收程序的接收按钮点击事件代码

|  |
| --- |
| void CClipboardDataReceiverDlg::OnBnClickedBtnRecv()  {  // TODO: 在此添加控件通知处理程序代码    HANDLE hMem;  TCHAR \*pData;  //操作剪贴板之前需要先打开她  if(!OpenClipboard())  {  MessageBox(\_T("Open Clipboard Failed！！"));  return;  }  //检查剪贴板的格式是否文本格式  if(IsClipboardFormatAvailable(CF\_TEXT))  {  hMem = GetClipboardData(CF\_TEXT);  pData = (TCHAR\*)GlobalLock(hMem);  GlobalUnlock(hMem);  SetDlgItemText(IDC\_EDIT\_DATA,pData);  }  CloseClipboard();  //让保存按钮可用  m\_btnSave.EnableWindow(TRUE);  } |

### 测试，重新生成所有项目，然后启动两个程序，在发送端里面输入一些文本，点击发送，然后在接收端的窗口中点击接受，效果如下

|  |
| --- |
|  |

### 此时点击保存按钮是没有反应的，我们来完成这个功能

## 9.给保存数据按钮添加点事件处理函数，然后代码下面的代码

|  |
| --- |
| void CClipboardDataReceiverDlg::OnBnClickedBtnSave()  {  // TODO: 在此添加控件通知处理程序代码  CStdioFile file;  CString str;  GetDlgItemText(IDC\_EDIT\_DATA,str);  if(str.GetLength()== 0)  {  MessageBox(\_T("No data,Please enter data!"));  m\_editData.SetFocus();  return;  }  file.Open(\_T("data.txt"),CFile::modeCreate|CFile::modeWrite|CFile::typeText);  file.WriteString(str);  file.Flush();  file.Close();  MessageBox(\_T("保存数据成功"));  } |

### 运行程序，效果如下

|  |
| --- |
|  |

### 关于剪贴板的格式问题：CF\_TEXT是ansi格式，UNICODE可用使用CF\_UNICODETEXT或者CF\_OEMTEXT

# 这一节的学习到此为止，完整代码如下，主要是对话框的代码

## ClipboardDataSenderDlg.h

|  |
| --- |
| // ClipboardDataSenderDlg.h : 头文件  //  #pragma once  #include "afxwin.h"  // CClipboardDataSenderDlg 对话框  class CClipboardDataSenderDlg : public CDialogEx  {  // 构造  public:  CClipboardDataSenderDlg(CWnd\* pParent = NULL); // 标准构造函数  // 对话框数据  enum { IDD = IDD\_CLIPBOARDDATASENDER\_DIALOG };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  HICON m\_hIcon;  // 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:  virtual BOOL PreTranslateMessage(MSG\* pMsg);  afx\_msg void OnBnClickedBtnSend();  // 关联编辑框控件的成员变量  CEdit m\_editData;  }; |

## ClipboardDataSenderDlg.cpp

|  |
| --- |
| // ClipboardDataSenderDlg.cpp : 实现文件  //  #include "stdafx.h"  #include "ClipboardDataSender.h"  #include "ClipboardDataSenderDlg.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()  // CClipboardDataSenderDlg 对话框  CClipboardDataSenderDlg::CClipboardDataSenderDlg(CWnd\* pParent /\*=NULL\*/)  : CDialogEx(CClipboardDataSenderDlg::IDD, pParent)  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  }  void CClipboardDataSenderDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  DDX\_Control(pDX, IDC\_EDIT\_DATA, m\_editData);  }  BEGIN\_MESSAGE\_MAP(CClipboardDataSenderDlg, CDialogEx)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  ON\_BN\_CLICKED(IDC\_BTN\_SEND, &CClipboardDataSenderDlg::OnBnClickedBtnSend)  END\_MESSAGE\_MAP()  // CClipboardDataSenderDlg 消息处理程序  BOOL CClipboardDataSenderDlg::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 CClipboardDataSenderDlg::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 CClipboardDataSenderDlg::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  {  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 CClipboardDataSenderDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  BOOL CClipboardDataSenderDlg::PreTranslateMessage(MSG\* pMsg)  {  // TODO: 在此添加专用代码和/或调用基类  if(pMsg->message == WM\_KEYDOWN)  {  switch(pMsg->wParam)  {  case VK\_RETURN:  return TRUE;  case VK\_ESCAPE:  return TRUE;  }  }  return CDialogEx::PreTranslateMessage(pMsg);  }  void CClipboardDataSenderDlg::OnBnClickedBtnSend()  {  // TODO: 在此添加控件通知处理程序代码  CString str;  GetDlgItemText(IDC\_EDIT\_DATA,str);  if(str.GetLength()==0)  {  MessageBox(\_T("发送内容不能为空！！"));  m\_editData.SetFocus();  return;  }  //MessageBox(str);  //发送数据  //1.操作之前需要先打开剪贴板  if(!OpenClipboard())  {  MessageBox(\_T("Open Clipboard Failed！！"));  return;  }  //发送之前先把剪贴板清空  EmptyClipboard();  //动态分配内存  HANDLE hMem;  hMem = GlobalAlloc(GMEM\_MOVEABLE,(str.GetLength()+1)\*sizeof(TCHAR));//在当前进程的动态堆上分配内存  //因为上面的内存是可以移动的，我们需要把它锁住  TCHAR\* pData;  pData = (TCHAR\*)GlobalLock(hMem);  //拷贝数据到上面的内存  \_tcscpy(pData,str);  //解锁上面的锁定内存  GlobalUnlock(hMem);  //把数据设置到剪贴板  SetClipboardData(CF\_TEXT,hMem);  //关闭剪贴板  CloseClipboard();    } |

## ClipboardDataReceiverDlg.h

|  |
| --- |
| // ClipboardDataReceiverDlg.h : 头文件  //  #pragma once  #include "afxwin.h"  // CClipboardDataReceiverDlg 对话框  class CClipboardDataReceiverDlg : public CDialogEx  {  // 构造  public:  CClipboardDataReceiverDlg(CWnd\* pParent = NULL); // 标准构造函数  // 对话框数据  enum { IDD = IDD\_CLIPBOARDDATARECEIVER\_DIALOG };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  HICON m\_hIcon;  // 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:  virtual BOOL PreTranslateMessage(MSG\* pMsg);  // 关联保存按钮的成员变量  CButton m\_btnSave;  // 关联编辑框的成员变量  CEdit m\_editData;  afx\_msg void OnBnClickedBtnRecv();  afx\_msg void OnBnClickedBtnSave();  }; |

## ClipboardDataReceiverDlg.cpp

|  |
| --- |
| // ClipboardDataReceiverDlg.cpp : 实现文件  //  #include "stdafx.h"  #include "ClipboardDataReceiver.h"  #include "ClipboardDataReceiverDlg.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()  // CClipboardDataReceiverDlg 对话框  CClipboardDataReceiverDlg::CClipboardDataReceiverDlg(CWnd\* pParent /\*=NULL\*/)  : CDialogEx(CClipboardDataReceiverDlg::IDD, pParent)  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  }  void CClipboardDataReceiverDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  DDX\_Control(pDX, IDC\_BTN\_SAVE, m\_btnSave);  DDX\_Control(pDX, IDC\_EDIT1, m\_editData);  }  BEGIN\_MESSAGE\_MAP(CClipboardDataReceiverDlg, CDialogEx)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  ON\_BN\_CLICKED(IDC\_BTN\_RECV, &CClipboardDataReceiverDlg::OnBnClickedBtnRecv)  ON\_BN\_CLICKED(IDC\_BTN\_SAVE, &CClipboardDataReceiverDlg::OnBnClickedBtnSave)  END\_MESSAGE\_MAP()  // CClipboardDataReceiverDlg 消息处理程序  BOOL CClipboardDataReceiverDlg::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 CClipboardDataReceiverDlg::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 CClipboardDataReceiverDlg::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  {  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 CClipboardDataReceiverDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  BOOL CClipboardDataReceiverDlg::PreTranslateMessage(MSG\* pMsg)  {  // TODO: 在此添加专用代码和/或调用基类  if(pMsg->message == WM\_KEYDOWN)  {  switch(pMsg->wParam)  {  case VK\_RETURN:  return TRUE;  case VK\_ESCAPE:  return TRUE;  }  }  return CDialogEx::PreTranslateMessage(pMsg);  }  void CClipboardDataReceiverDlg::OnBnClickedBtnRecv()  {  // TODO: 在此添加控件通知处理程序代码    HANDLE hMem;  TCHAR \*pData;  //操作剪贴板之前需要先打开她  if(!OpenClipboard())  {  MessageBox(\_T("Open Clipboard Failed！！"));  return;  }  //检查剪贴板的格式是否文本格式  if(IsClipboardFormatAvailable(CF\_TEXT))  {  hMem = GetClipboardData(CF\_TEXT);  pData = (TCHAR\*)GlobalLock(hMem);  GlobalUnlock(hMem);  SetDlgItemText(IDC\_EDIT\_DATA,pData);  }  CloseClipboard();  //让保存按钮可用  m\_btnSave.EnableWindow(TRUE);  }  void CClipboardDataReceiverDlg::OnBnClickedBtnSave()  {  // TODO: 在此添加控件通知处理程序代码  CStdioFile file;  CString str;  GetDlgItemText(IDC\_EDIT\_DATA,str);  if(str.GetLength()== 0)  {  MessageBox(\_T("No data,Please enter data!"));  m\_editData.SetFocus();  return;  }  file.Open(\_T("data.txt"),CFile::modeCreate|CFile::modeWrite|CFile::typeText);  file.WriteString(str);  file.Flush();  file.Close();  MessageBox(\_T("保存数据成功"));  } |

# 扩展，可用尝试发送Bitmap到剪贴板然后用接收程序来获取