# 学习大纲

|  |
| --- |
|  |

## LOGFONT结构体

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **LOGFONT** 结构定义字体的属性。 语法 C++复制  typedef struct tagLOGFONTA {  LONG lfHeight;  LONG lfWidth;  LONG lfEscapement;  LONG lfOrientation;  LONG lfWeight;  BYTE lfItalic;  BYTE lfUnderline;  BYTE lfStrikeOut;  BYTE lfCharSet;  BYTE lfOutPrecision;  BYTE lfClipPrecision;  BYTE lfQuality;  BYTE lfPitchAndFamily;  CHAR lfFaceName[LF\_FACESIZE];  } LOGFONTA, \*PLOGFONTA, \*NPLOGFONTA, \*LPLOGFONTA; 成员 lfHeight  字体的字符单元格或字符的高度（以逻辑单位为单位）。 字符高度值 (也称为 em height) 是字符单元格高度值减去内部前导值。 字体映射器按以下方式解释 **lfHeight** 中指定的值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | > 0 | 字体映射器将此值转换为设备单位，并将其与可用字体的单元格高度匹配。 | | 0 | 字体映射器在搜索匹配项时使用默认高度值。 | | < 0 | 字体映射器将此值转换为设备单位，并将其绝对值与可用字体的字符高度匹配。 |     对于所有高度比较，字体映射器会查找不超过请求大小的最大字体。  首次使用该字体时，会发生此映射。  对于MM\_TEXT映射模式，可以使用以下公式指定具有指定点大小的字体的高度：  lfHeight = -MulDiv(PointSize, GetDeviceCaps(hDC, LOGPIXELSY), 72);  lfWidth  字体中字符的平均宽度（以逻辑单位为单位）。 如果 **lfWidth** 为零，则设备的纵横比将与可用字体的数字化纵横比进行匹配，以查找由差值的绝对值确定的最接近匹配项。  lfEscapement  转义向量与设备的 x 轴之间的角度（以十分之一度为单位）。 转义向量与文本行的基线平行。  当图形模式设置为“GM\_ADVANCED”时，可以独立于字符串字符的方向角度指定字符串的转义角度。  当图形模式设置为GM\_COMPATIBLE时， **lfEscapement** 同时指定转义和方向。 应将 **lfEscapement** 和 **lfOrientation** 设置为相同的值。  lfOrientation  每个字符的基线和设备 x 轴之间的角度（以十分之一度为单位）。  lfWeight  字体的粗细，范围为 0 到 1000。 例如，400 为正常值，700 为粗体。 如果此值为零，则使用默认权重。  为方便起见，定义了以下值。  展开表   |  |  | | --- | --- | | **值** | **重量** | | FW\_DONTCARE | 0 | | FW\_THIN | 100 | | FW\_EXTRALIGHT | 200 | | FW\_ULTRALIGHT | 200 | | FW\_LIGHT | 300 | | FW\_NORMAL | 400 | | FW\_REGULAR | 400 | | FW\_MEDIUM | 500 | | FW\_SEMIBOLD | 600 | | FW\_DEMIBOLD | 600 | | FW\_BOLD | 700 | | FW\_EXTRABOLD | 800 | | FW\_ULTRABOLD | 800 | | FW\_HEAVY | 900 | | FW\_BLACK | 900 |   lfItalic  如果设置为 **TRUE**，则为斜体字体。  lfUnderline  如果设置为 **TRUE**，则为带下划线的字体。  lfStrikeOut  如果设置为 **TRUE**，则为删除线字体。  lfCharSet  字符集。 以下值是预定义的：   * ANSI\_CHARSET * BALTIC\_CHARSET * CHINESEBIG5\_CHARSET * DEFAULT\_CHARSET * EASTEUROPE\_CHARSET * GB2312\_CHARSET * GREEK\_CHARSET * HANGUL\_CHARSET * MAC\_CHARSET * OEM\_CHARSET * RUSSIAN\_CHARSET * SHIFTJIS\_CHARSET * SYMBOL\_CHARSET * TURKISH\_CHARSET * VIETNAMESE\_CHARSET   Windows 的朝鲜语版本：   * JOHAB\_CHARSET   Windows 中东语言版本：   * ARABIC\_CHARSET * HEBREW\_CHARSET   Windows 的泰语版本：   * THAI\_CHARSET   OEM\_CHARSET值指定依赖于操作系统的字符集。  DEFAULT\_CHARSET根据当前系统区域设置设置为值。 例如，当系统区域设置为英语 (美国) 时，它设置为 ANSI\_CHARSET。  操作系统中可能存在具有其他字符集的字体。 如果应用程序使用字符集未知的字体，则不应尝试翻译或解释使用该字体呈现的字符串。  此参数在字体映射过程中非常重要。 为确保创建字体时的结果一致，请勿指定OEM\_CHARSET或DEFAULT\_CHARSET。 如果在 **lfFaceName** 成员中指定字样名称，请确保 **lfCharSet** 值与 **lfFaceName** 中指定的字样字符集匹配。  lfOutPrecision  输出精度。 输出精度定义输出与所请求字体的高度、宽度、字符方向、转义、间距和字体类型的匹配程度。 可以是下列值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | OUT\_CHARACTER\_PRECIS | 未使用。 | | OUT\_DEFAULT\_PRECIS | 指定默认字体映射器行为。 | | OUT\_DEVICE\_PRECIS | 当系统包含多个同名字体时，指示字体映射器选择设备字体。 | | OUT\_OUTLINE\_PRECIS | 此值指示字体映射器从 TrueType 和其他基于大纲的字体中进行选择。 | | OUT\_PS\_ONLY\_PRECIS | 指示字体映射器仅从 PostScript 字体中进行选择。 如果系统中没有安装 PostScript 字体，字体映射器将返回到默认行为。 | | OUT\_RASTER\_PRECIS | 当系统包含多个同名字体时，指示字体映射器选择光栅字体。 | | OUT\_STRING\_PRECIS | 字体映射器不使用此值，但在枚举光栅字体时会返回此值。 | | OUT\_STROKE\_PRECIS | 字体映射器不使用此值，但在枚举 TrueType、其他基于轮廓的字体和矢量字体时返回此值。 | | OUT\_TT\_ONLY\_PRECIS | 指示字体映射器仅从 TrueType 字体中进行选择。 如果系统中没有安装 TrueType 字体，字体映射器将返回到默认行为。 | | OUT\_TT\_PRECIS | 当系统包含多个同名字体时，指示字体映射器选择 TrueType 字体。 |     当操作系统包含多个具有指定名称的字体时，应用程序可以使用OUT\_DEVICE\_PRECIS、OUT\_RASTER\_PRECIS、OUT\_TT\_PRECIS和OUT\_PS\_ONLY\_PRECIS值来控制字体映射器选择字体的方式。 例如，如果操作系统在光栅和 TrueType 窗体中包含名为 Symbol 的字体，则指定OUT\_TT\_PRECIS将强制字体映射器选择 TrueType 版本。 指定OUT\_TT\_ONLY\_PRECIS会强制字体映射器选择 TrueType 字体，即使它必须替换另一个名称的 TrueType 字体。  lfClipPrecision  剪裁精度。 剪裁精度定义如何剪裁部分超出剪裁区域的字符。 它可以是以下一个或多个值。  有关坐标系的方向的详细信息，请参阅 *nOrientation* 参数的说明。  展开表   |  |  | | --- | --- | | **值** | **含义** | | CLIP\_CHARACTER\_PRECIS | 未使用。 | | CLIP\_DEFAULT\_PRECIS | 指定默认剪辑行为。 | | CLIP\_DFA\_DISABLE | **Windows XP SP1：** 关闭字体的字体关联。 请注意，此标志不保证在 Windows Server 2003 之后对任何平台产生任何影响。 | | CLIP\_EMBEDDED | 必须指定此标志才能使用嵌入的只读字体。 | | CLIP\_LH\_ANGLES | 使用此值时，所有字体的旋转取决于坐标系的方向是左手还是右手。如果未使用，设备字体始终逆时针旋转，但其他字体的旋转取决于坐标系的方向。 | | CLIP\_MASK | 未使用。 | | CLIP\_DFA\_OVERRIDE | 关闭字体的字体关联。 这与CLIP\_DFA\_DISABLE相同，但在某些情况下可能会有问题：建议使用的标志是CLIP\_DFA\_DISABLE。 | | CLIP\_STROKE\_PRECIS | 字体映射器不使用，但在枚举光栅、矢量或 TrueType 字体时返回。 为了兼容，枚举字体时始终返回此值。 | | CLIP\_TT\_ALWAYS | 未使用。 |   lfQuality  输出质量。 输出质量定义图形设备接口 (GDI) 必须尝试将逻辑字体属性与实际物理字体属性匹配的谨慎程度。 可以是下列值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | ANTIALIASED\_QUALITY | 如果字体支持该字体，并且字体大小不是太小或太大，则字体始终为抗锯齿。 | | CLEARTYPE\_QUALITY | 如果设置，则尽可能使用 ClearType 抗锯齿方法) 呈现文本 (。 有关更多信息，请参见备注。 | | DEFAULT\_QUALITY | 字体的外观并不重要。 | | DRAFT\_QUALITY | 字体的外观不如使用PROOF\_QUALITY时重要。 对于 GDI 光栅字体，会启用缩放，这意味着可以使用更多字号，但质量可能较低。 如有必要，将合成粗体、斜体、下划线和删除线字体。 | | NONANTIALIASED\_QUALITY | 字体永远不会抗锯齿。 | | PROOF\_QUALITY | 字体的字符质量比逻辑字体属性的精确匹配更重要。 对于 GDI 光栅字体，将禁用缩放，并选择大小最接近的字体。 虽然在使用PROOF\_QUALITY时可能无法精确映射所选字号，但字体质量较高，外观不会失真。 如有必要，将合成粗体、斜体、下划线和删除线字体。 |     如果ANTIALIASED\_QUALITY和NONANTIALIASED\_QUALITY均未选中，则仅当用户在控制面板中选择平滑屏幕字体时，字体才会抗锯齿。  lfPitchAndFamily  字体的间距和系列。 两个低位指定字体的间距，可以是以下值之一。   * DEFAULT\_PITCH * FIXED\_PITCH * VARIABLE\_PITCH   成员的位 4 到 7 指定字体系列，可以是以下值之一。   * FF\_DECORATIVE * FF\_DONTCARE * FF\_MODERN * FF\_ROMAN * FF\_SCRIPT * FF\_SWISS   通过使用布尔 OR 运算符将一个音调常量与一个系列常量相联接，可以获得适当的值。  字体系列以一般方式描述字体的外观。 它们用于在所需的确切字样不可用时指定字体。 字体系列的值如下所示。  展开表   |  |  | | --- | --- | | **值** | **含义** | | FF\_DECORATIVE | 新奇字体。 例如 Old English。 | | FF\_DONTCARE | 使用默认字体。 | | FF\_MODERN | 具有固定笔划宽度的字体 (带衬线或无衬线) 的空白字体。 Monospace 字体通常是新式字体。 例如，Pica、Elite 和 CourierNew。 | | FF\_ROMAN | 笔划宽度可变的字体 (比例) 和衬线。 例如 MS Serif。 | | FF\_SCRIPT | 设计为类似于手写的字体。 例如 Script 和 Cursive。 | | FF\_SWISS | 笔划宽度可变的字体 (成比例) 且不带衬线。 例如 MS Sans Serif。 |   lfFaceName[LF\_FACESIZE]  以 NULL 结尾的字符串指定字体的字体名称。 此字符串的长度不能超过 32 **个 TCHAR** 值，包括终止 **NULL**。 [EnumFontFamiliesEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-enumfontfamiliesexa) 函数可用于枚举所有当前可用字体的字体名称。 如果 **lfFaceName** 是空字符串，则 GDI 使用与其他指定属性匹配的第一个字体。 注解 以下情况不支持 ClearType 抗锯齿：   * 文本在打印机上呈现。 * 显示设置为 256 种或更少颜色。 * 文本呈现到终端服务器客户端。 * 该字体不是 TrueType 字体或具有 TrueType 轮廓的 OpenType 字体。 例如，以下不支持 ClearType 抗锯齿：键入 1 字体、不带 TrueType 轮廓的 Postscript OpenType 字体、位图字体、矢量字体和设备字体。 * 该字体已针对包含嵌入位图的任何字号调整嵌入位图。 例如，这通常发生在东亚字体中。   **备注**  wingdi.h 标头将 LOGFONT 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。 将非特定编码别名的使用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** | wingdi.h (包括 Windows.h) |  请参阅 [CreateFont](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-createfonta)  [CreateFontIndirect](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-createfontindirecta)  [EnumFontFamiliesEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-enumfontfamiliesexa)  [字体和文本结构](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/font-and-text-structures)  [字体和文本概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/fonts-and-text) |

### LOGFONT结构体不需要初始化就可以使用

## LOGFONT结构体成员含义

|  |
| --- |
| 每个字段的含义如下：  lfHeight：指定逻辑单位的字符或者字符元高度。  lfWidth：指定逻辑单位的字体字符的平均宽度。  lfEscapement：指定每行文本输出时相对于设备x轴的角度，其单位为1/10度。  lfOrientation：指定字符基线相对于设备x轴的角度，其单位为1/10度。此值在Win9X中和lfEscapement具有相同的值，而在WinNT下有时候可能不同。  lfWeight：指定字体的重量，Windows中字体重量表示字体的粗细程度，其范围在0～1000之间，正常为400，粗体为700，若此值为空，则使用默认的字体重量。  lfItalic：此值为TRUE时，字体为斜体。  lfUnderline：此值为TRUE时，字体带下划线。  lfStrikeOut：此值为TRUE时，字体带删除线。  lfCharSet：指定所使用的字符集，如GB2312\_CHARSET,CHINESEBIG5\_CHARSET等。  lfOutPrecision：指定输出精度，它定义了输出与所要求的字体高度、宽度、字符方向及字体类型等相接近的程度。  lfClipPrecision：指定剪辑精度，它定义了当字符的一部分超过剪辑区域时对字符的剪辑方式。  lfQuality：指定输出质量，它定义了GDI在匹配逻辑字体属性到实际的物理字体时所使用的方式。  lfPitchAndFamily：指定字体的字符间距和族。  lfFaceName：指向NULL结尾的字符串的指针，此字符串即为所使用的字体名称，其长度不能超过32个字符，如果为空，则使用系统默认的字体 |

## CeateFontIndirect函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CreateFontIndirect** 函数创建具有指定特征的逻辑字体。 随后可以选择该字体作为任何设备上下文的当前字体。 语法 HFONT CreateFontIndirectA(  [in] const LOGFONTA \*lplf  ); 参数 [in] lplf  指向定义逻辑字体特征的 [LOGFONT](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/ns-wingdi-logfonta) 结构的指针。 返回值 如果函数成功，则返回值是逻辑字体的句柄。  如果函数失败，则返回值为 NULL。 注解 **CreateFontIndirect** 函数使用 [LOGFONT](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/ns-wingdi-logfonta) 结构中指定的特征创建逻辑字体。 使用 [SelectObject](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-selectobject) 函数选择此字体时，GDI 的字体映射器会尝试将逻辑字体与现有物理字体匹配。 如果找不到完全匹配项，它将提供一个替代项，其特征与尽可能多的请求的特征匹配。  若要在不同语言版本的操作系统上获取适当的字体，请使用 [LOGFONT](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/ns-wingdi-logfonta) 结构中所需的字体特征调用 [EnumFontFamiliesEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-enumfontfamiliesexa)，检索适当的字样名称，并使用 [CreateFont](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-createfonta) 或 **CreateFontIndirect** 创建字体。  如果不再需要该字体，请调用 [DeleteObject](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-deleteobject) 函数将其删除。  许多东亚语言的字体有两个字样名称：英文名称和本地化名称。 [CreateFont](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-createfonta) 和 **CreateFontIndirect** 仅在与语言匹配的系统区域设置上采用本地化的字样名称，而在所有其他系统区域设置上采用英文字样名称。 最好的方法是尝试一个名称，如果失败，请尝试另一个名称。 请注意，如果系统区域设置与字体的语言不匹配，[EnumFontFamilies 和 EnumFontFamiliesEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-enumfontfamiliesexa) 将返回英文字样名称。  [CreateFont](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-createfonta)、**CreateFontIndirect** 和 [CreateFontIndirectEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-createfontindirectexa) 的字体映射器可识别英语和本地化的字样名称，而不考虑区域设置。 示例 有关示例，请参阅 [创建逻辑字体](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/creating-a-logical-font)。  **备注**  wingdi.h 标头将 CreateFontIndirect 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** | wingdi.h (包括 Windows.h) | | **Library** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [CreateFont](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-createfonta)  [CreateFontIndirectEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-createfontindirectexa)  [DeleteObject](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-deleteobject)  [EnumFontFamilies](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-enumfontfamiliesa)  [EnumFontFamiliesEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-enumfontfamiliesexa)  [EnumFonts](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-enumfontsa)  [字体和文本函数](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/font-and-text-functions)  [字体和文本概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/fonts-and-text)  [LOGFONT](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/ns-wingdi-logfonta)  [SelectObject](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-selectobject) |

## CreateFont函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CreateFont** 函数创建具有指定特征的逻辑字体。 随后可以选择逻辑字体作为任何设备的字体。 语法 C++复制  HFONT CreateFontW(  [in] int cHeight,  [in] int cWidth,  [in] int cEscapement,  [in] int cOrientation,  [in] int cWeight,  [in] DWORD bItalic,  [in] DWORD bUnderline,  [in] DWORD bStrikeOut,  [in] DWORD iCharSet,  [in] DWORD iOutPrecision,  [in] DWORD iClipPrecision,  [in] DWORD iQuality,  [in] DWORD iPitchAndFamily,  [in] LPCWSTR pszFaceName  ); 参数 [in] cHeight  字体的字符单元格或字符的高度（以逻辑单位为单位）。 字符高度值 (也称为 em height) 是字符单元格高度值减去内部前导值。 字体映射器按以下方式解释 *在 nHeight* 中指定的值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | > 0 | 字体映射器将此值转换为设备单位，并将其与可用字体的单元格高度进行匹配。 | | 0 | 字体映射器在搜索匹配项时使用默认高度值。 | | < 0 | 字体映射器将此值转换为设备单位，并将其绝对值与可用字体的字符高度相匹配。 |     对于所有高度比较，字体映射器查找不超过请求大小的最大字体。  首次使用该字体时会发生此映射。  对于MM\_TEXT映射模式，可以使用以下公式指定具有指定点大小的字体的高度：  C++复制  nHeight = -MulDiv(PointSize, GetDeviceCaps(hDC, LOGPIXELSY), 72);  [in] cWidth  所请求字体中字符的平均宽度（以逻辑单位为单位）。 如果此值为零，则字体映射器选择最接近的匹配值。 最接近的匹配值是通过比较当前设备的纵横比与可用字体的数字化纵横比之差的绝对值来确定的。  [in] cEscapement  转义向量和设备 x 轴之间的角度（以十分之一度为单位）。 转义向量与文本行的基线平行。  当图形模式设置为GM\_ADVANCED时，可以指定字符串的转义角度，而与字符串的字符的方向角度无关。  当图形模式设置为GM\_COMPATIBLE时， *nEscapement* 将同时指定转义和方向。 应将 *nEscapement* 和 *nOrientation* 设置为相同的值。  [in] cOrientation  每个字符的底线和设备 x 轴之间的角度（以十分之一度为单位）。  [in] cWeight  0 到 1000 范围内的字体粗细。 例如，400 为正常值，700 为粗体。 如果此值为零，则使用默认权重。  为方便起见，定义了以下值。  展开表   |  |  | | --- | --- | | **重量** | **值** | | **FW\_DONTCARE** | 0 | | **FW\_THIN** | 100 | | **FW\_EXTRALIGHT** | 200 | | **FW\_ULTRALIGHT** | 200 | | **FW\_LIGHT** | 300 | | **FW\_NORMAL** | 400 | | **FW\_REGULAR** | 400 | | **FW\_MEDIUM** | 500 | | **FW\_SEMIBOLD** | 600 | | **FW\_DEMIBOLD** | 600 | | **FW\_BOLD** | 700 | | **FW\_EXTRABOLD** | 800 | | **FW\_ULTRABOLD** | 800 | | **FW\_HEAVY** | 900 | | **FW\_BLACK** | 900 |   [in] bItalic  如果设置为 **TRUE**，则指定斜体字体。  [in] bUnderline  如果设置为 **TRUE**，则指定带下划线的字体。  [in] bStrikeOut  如果设置为 **TRUE**，则为删除线字体。  [in] iCharSet  字符集。 预定义了以下值：   * ANSI\_CHARSET * BALTIC\_CHARSET * CHINESEBIG5\_CHARSET * DEFAULT\_CHARSET * EASTEUROPE\_CHARSET * GB2312\_CHARSET * GREEK\_CHARSET * HANGUL\_CHARSET * MAC\_CHARSET * OEM\_CHARSET * RUSSIAN\_CHARSET * SHIFTJIS\_CHARSET * SYMBOL\_CHARSET * TURKISH\_CHARSET * VIETNAMESE\_CHARSET   Windows 的朝鲜语版本：   * JOHAB\_CHARSET   Windows 中东语言版本：   * ARABIC\_CHARSET * HEBREW\_CHARSET   Windows 的泰语版本：   * THAI\_CHARSET   OEM\_CHARSET值指定依赖于操作系统的字符集。  DEFAULT\_CHARSET根据当前系统区域设置设置为值。 例如，当系统区域设置为英语 (美国) 时，它将设置为ANSI\_CHARSET。  操作系统中可能存在具有其他字符集的字体。 如果应用程序使用具有未知字符集的字体，则不应尝试翻译或解释使用该字体呈现的字符串。  为确保创建字体时的结果一致，请不要指定OEM\_CHARSET或DEFAULT\_CHARSET。 如果在 *lpszFace* 参数中指定字样名称，请确保 *fdwCharSet* 值与 *lpszFace* 中指定的字样字符集匹配。  [in] iOutPrecision  输出精度。 输出精度定义输出与所请求字体的高度、宽度、字符方向、转义、间距和字体类型的匹配程度。 可以是下列值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **OUT\_CHARACTER\_PRECIS** | 未使用。 | | **OUT\_DEFAULT\_PRECIS** | 默认字体映射器行为。 | | **OUT\_DEVICE\_PRECIS** | 当系统包含多个同名字体时，指示字体映射器选择设备字体。 | | **OUT\_OUTLINE\_PRECIS** | 此值指示字体映射器从 TrueType 和其他基于大纲的字体中进行选择。 | | **OUT\_PS\_ONLY\_PRECIS** | 指示字体映射器仅从 PostScript 字体中进行选择。 如果系统中没有安装 PostScript 字体，字体映射器将返回到默认行为。 | | **OUT\_RASTER\_PRECIS** | 当系统包含多个同名字体时，指示字体映射器选择光栅字体。 | | **OUT\_STRING\_PRECIS** | 字体映射器不使用此值，但在枚举光栅字体时会返回此值。 | | **OUT\_STROKE\_PRECIS** | 字体映射器不使用此值，但在枚举 TrueType、其他基于大纲的字体和矢量字体时返回此值。 | | **OUT\_TT\_ONLY\_PRECIS** | 指示字体映射器仅从 TrueType 字体中进行选择。 如果系统中没有安装 TrueType 字体，字体映射器将返回到默认行为。 | | **OUT\_TT\_PRECIS** | 当系统包含多个同名字体时，指示字体映射器选择 TrueType 字体。 |     当操作系统包含多个具有指定名称的字体时，应用程序可以使用OUT\_DEVICE\_PRECIS、OUT\_RASTER\_PRECIS、OUT\_TT\_PRECIS和OUT\_PS\_ONLY\_PRECIS值来控制字体映射器选择字体的方式。 例如，如果操作系统在光栅和 TrueType 窗体中包含名为 Symbol 的字体，则指定OUT\_TT\_PRECIS将强制字体映射器选择 TrueType 版本。 指定OUT\_TT\_ONLY\_PRECIS将强制字体映射器选择 TrueType 字体，即使它必须替换另一个名称的 TrueType 字体。  [in] iClipPrecision  剪裁精度。 剪裁精度定义如何剪裁部分超出剪裁区域的字符。 它可以是下面的一个或多个值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **CLIP\_CHARACTER\_PRECIS** | 未使用。 | | **CLIP\_DEFAULT\_PRECIS** | 指定默认剪辑行为。 | | **CLIP\_DFA\_DISABLE** | Windows XP SP1：关闭字体的字体关联。 请注意，此标志不保证在 Windows Server 2003 之后对任何平台产生任何影响。 | | **CLIP\_EMBEDDED** | 必须指定此标志才能使用嵌入的只读字体。 | | **CLIP\_LH\_ANGLES** | 使用此值时，所有字体的旋转取决于坐标系的方向是左手还是右手。  如果未使用，设备字体始终逆时针旋转，但其他字体的旋转取决于坐标系的方向。  有关坐标系的方向的详细信息，请参阅 *nOrientation* 参数的说明 | | **CLIP\_MASK** | 未使用。 | | **CLIP\_DFA\_OVERRIDE** | 关闭字体的字体关联。 这与CLIP\_DFA\_DISABLE相同，但在某些情况下可能会有问题：建议使用的标志是CLIP\_DFA\_DISABLE。 | | **CLIP\_STROKE\_PRECIS** | 字体映射器不使用，但在枚举光栅、矢量或 TrueType 字体时返回。  为了兼容，枚举字体时始终返回此值。 | | **CLIP\_TT\_ALWAYS** | 未使用。 |   [in] iQuality  输出质量。 输出质量定义 GDI 尝试将逻辑字体属性与实际物理字体属性匹配的谨慎程度。 可以是下列值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **ANTIALIASED\_QUALITY** | 如果字体支持该字体，并且字体大小不是太小或太大，则字体已抗锯齿或平滑。 | | **CLEARTYPE\_QUALITY** | 如果设置，则尽可能使用 ClearType 抗锯齿方法) (呈现文本。 有关更多信息，请参见备注。 | | **DEFAULT\_QUALITY** | 字体的外观并不重要。 | | **DRAFT\_QUALITY** | 字体的外观不如使用PROOF\_QUALITY值时重要。 对于 GDI 光栅字体，会启用缩放，这意味着可以使用更多字号，但质量可能较低。 如有必要，将合成粗体、斜体、下划线和删除线字体。 | | **NONANTIALIASED\_QUALITY** | 字体永远不会抗锯齿，也就是说，字体平滑处理未完成。 | | **PROOF\_QUALITY** | 字体的字符质量比逻辑字体属性的精确匹配更重要。 对于 GDI 光栅字体，将禁用缩放，并选择大小最接近的字体。 虽然在使用PROOF\_QUALITY时可能无法精确映射所选字号，但字体质量较高，外观不会失真。 如有必要，将合成粗体、斜体、下划线和删除线字体。 |     如果输出质量为DEFAULT\_QUALITY、DRAFT\_QUALITY或PROOF\_QUALITY，则当SPI\_GETFONTSMOOTHING系统参数为 **TRUE** 时，字体将抗锯齿。 用户可以从控制面板控制此系统参数。 (控制面板中设置的精确措辞取决于 Windows 的版本，但对于“屏幕字体的平滑边缘”效果而言，其效果是单词。)  [in] iPitchAndFamily  字体的间距和系列。 两个低位指定字体的间距，可以是以下值之一：   * DEFAULT\_PITCH * FIXED\_PITCH * VARIABLE\_PITCH   四个高阶位指定字体系列，可以是以下值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **FF\_DECORATIVE** | 新奇字体。 例如 Old English。 | | **FF\_DONTCARE** | 使用默认字体。 | | **FF\_MODERN** | 笔划宽度恒定的字体，带或不带衬线。 例如，Pica、Elite 和 Courier New。 | | **FF\_ROMAN** | 具有可变笔划宽度和衬线的字体。 例如 MS Serif。 | | **FF\_SCRIPT** | 设计为类似于手写的字体。 例如 Script 和 Cursive。 | | **FF\_SWISS** | 笔划宽度可变且不带衬线的字体。 女士？Sans Serif 就是一个例子。 |     应用程序可以使用布尔 OR 运算符将间距常量与系列常量联接，从而指定 *fdwPitchAndFamily* 参数的值。  字体系列以一般方式描述字体的外观。 它们用于在请求的确切字样不可用时指定字体。  [in] pszFaceName  指向以 null 结尾的字符串的指针，该字符串指定字体的字样名称。 此字符串的长度不得超过 32 个字符，包括终止 null 字符。 [EnumFontFamilies](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-enumfontfamiliesa) 函数可用于枚举所有当前可用字体的字样名称。 有关详细信息，请参阅“备注”部分。  如果 *lpszFace* 为 **NULL** 或空字符串，GDI 将使用与其他指定属性匹配的第一个字体。 返回值 如果函数成功，则返回值是逻辑字体的句柄。  如果函数失败，则返回值为 NULL。 注解 如果不再需要该字体，请调用 [DeleteObject](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-deleteobject) 函数将其删除。  为了帮助保护为 Windows 提供字体的供应商的版权，应用程序应始终报告所选字体的确切名称。 由于可用字体可能因系统而异，因此不要假定所选字体始终与请求的字体相同。 例如，如果请求名为 Palatino 的字体，但系统上没有此类字体，则字体映射器将替换具有类似属性但名称不同的字体。 始终向用户报告所选字体的名称。  若要在操作系统的不同语言版本上获取适当的字体，请在 [LOGFONT](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/ns-wingdi-logfonta) 结构中调用具有所需字体特征的 [EnumFontFamiliesEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-enumfontfamiliesexa)，然后检索相应的字样名称并使用 **CreateFont** 或 [CreateFontIndirect](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-createfontindirecta) 创建字体。  **CreateFont**、[CreateFontIndirect](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-createfontindirecta) 和 [CreateFontIndirectEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-createfontindirectexa) 的字体映射器可识别英语和本地化的字样名称，而不考虑区域设置。  以下情况不支持 ClearType 抗锯齿：   * 在打印机上呈现的文本。 * 设置为 256 种或更少颜色的显示器。 * 呈现到终端服务器客户端的文本。 * 该字体不是 TrueType 字体，也不是带有 TrueType 轮廓的 OpenType 字体。 例如，以下不支持 ClearType 抗锯齿：类型 1 字体、不带 TrueType 轮廓的 Postscript OpenType 字体、位图字体、矢量字体和设备字体。 * 字体已优化嵌入位图，仅适用于包含嵌入位图的字号。 例如，这通常发生在东亚字体中。  示例 C++复制  LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  {  int wmId, wmEvent;  PAINTSTRUCT ps;  HDC hdc;  switch (message)  {      case WM\_PAINT:  {  RECT rect;  HFONT hFontOriginal, hFont1, hFont2, hFont3;  hdc = BeginPaint(hWnd, &ps);    //Logical units are device dependent pixels, so this will create a handle to a logical font that is 48 pixels in height.  //The width, when set to 0, will cause the font mapper to choose the closest matching value.  //The font face name will be Impact.  hFont1 = CreateFont(48,0,0,0,FW\_DONTCARE,FALSE,TRUE,FALSE,DEFAULT\_CHARSET,OUT\_OUTLINE\_PRECIS,  CLIP\_DEFAULT\_PRECIS,CLEARTYPE\_QUALITY, VARIABLE\_PITCH,TEXT("Impact"));  hFontOriginal = (HFONT)SelectObject(hdc, hFont1);    //Sets the coordinates for the rectangle in which the text is to be formatted.  SetRect(&rect, 100,100,700,200);  SetTextColor(hdc, RGB(255,0,0));  DrawText(hdc, TEXT("Drawing Text with Impact"), -1,&rect, DT\_NOCLIP);    //Logical units are device dependent pixels, so this will create a handle to a logical font that is 36 pixels in height.  //The width, when set to 20, will cause the font mapper to choose a font which, in this case, is stretched.  //The font face name will be Times New Roman. This time nEscapement is at -300 tenths of a degree (-30 degrees)  hFont2 = CreateFont(36,20,-300,0,FW\_DONTCARE,FALSE,TRUE,FALSE,DEFAULT\_CHARSET,OUT\_OUTLINE\_PRECIS,  CLIP\_DEFAULT\_PRECIS,CLEARTYPE\_QUALITY, VARIABLE\_PITCH,TEXT("Times New Roman"));  SelectObject(hdc,hFont2);    //Sets the coordinates for the rectangle in which the text is to be formatted.  SetRect(&rect, 100, 200, 900, 800);  SetTextColor(hdc, RGB(0,128,0));  DrawText(hdc, TEXT("Drawing Text with Times New Roman"), -1,&rect, DT\_NOCLIP);    //Logical units are device dependent pixels, so this will create a handle to a logical font that is 36 pixels in height.  //The width, when set to 10, will cause the font mapper to choose a font which, in this case, is compressed.  //The font face name will be Arial. This time nEscapement is at 250 tenths of a degree (25 degrees)  hFont3 = CreateFont(36,10,250,0,FW\_DONTCARE,FALSE,TRUE,FALSE,DEFAULT\_CHARSET,OUT\_OUTLINE\_PRECIS,  CLIP\_DEFAULT\_PRECIS,ANTIALIASED\_QUALITY, VARIABLE\_PITCH,TEXT("Arial"));  SelectObject(hdc,hFont3);  //Sets the coordinates for the rectangle in which the text is to be formatted.  SetRect(&rect, 500, 200, 1400, 600);  SetTextColor(hdc, RGB(0,0,255));  DrawText(hdc, TEXT("Drawing Text with Arial"), -1,&rect, DT\_NOCLIP);  SelectObject(hdc,hFontOriginal);  DeleteObject(hFont1);  DeleteObject(hFont2);  DeleteObject(hFont3);    EndPaint(hWnd, &ps);  break;  }  case WM\_DESTROY:  PostQuitMessage(0);  break;  default:  return DefWindowProc(hWnd, message, wParam, lParam);  }  return 0;  }  有关另一个示例，请参阅 [Using Menus](https://learn.microsoft.com/zh-cn/windows/desktop/menurc/using-menus) 中的“设置 Menu-Item 文本字符串的字体”。  **备注**  wingdi.h 标头将 CreateFont 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)。   |  |  | | --- | --- | | **标头** | wingdi.h (包括 Windows.h) | | **Library** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [CreateFontIndirect](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-createfontindirecta)  [CreateFontIndirectEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-createfontindirectexa)  [DeleteObject](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-deleteobject)  **EnumFontFamilies**  [EnumFontFamiliesEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-enumfontfamiliesexa)  [EnumFonts](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-enumfontsa)  [字体和文本函数](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/font-and-text-functions)  [字体和文本概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/fonts-and-text)  [LOGFONT](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/ns-wingdi-logfonta)  [SelectObject](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-selectobject) |

## GetStockObject函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **GetStockObject** 函数检索其中一支股票笔、画笔、字体或调色板的句柄。 语法 C++复制  HGDIOBJ GetStockObject(  [in] int i  ); 参数 [in] i  常用对象的类型。 此参数的取值可为下列值之一：  展开表   |  |  | | --- | --- | | **值** | **含义** | | **BLACK\_BRUSH** | 黑色画笔。 | | **DKGRAY\_BRUSH** | 深灰色画笔。 | | **DC\_BRUSH** | 纯色画笔。 默认颜色为白色。 可以使用 [SetDCBrushColor](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-setdcbrushcolor) 函数更改颜色。 有关详细信息，请参见“备注”部分。 | | **GRAY\_BRUSH** | 灰色画笔。 | | **HOLLOW\_BRUSH** | 空心画笔 (等效于NULL\_BRUSH) 。 | | **LTGRAY\_BRUSH** | 浅灰色画笔。 | | **NULL\_BRUSH** | null 画笔 (等效于 HOLLOW\_BRUSH) 。 | | **WHITE\_BRUSH** | 白色画笔。 | | **BLACK\_PEN** | 黑色触笔。 | | **DC\_PEN** | 纯色笔颜色。 默认颜色为黑色。 可以使用 [SetDCPenColor](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-setdcpencolor) 函数更改颜色。 有关详细信息，请参见“备注”部分。 | | **NULL\_PEN** | null 触笔。 null 触控笔不绘制任何内容。 | | **WHITE\_PEN** | 白色触笔。 | | **ANSI\_FIXED\_FONT** | Windows 固定间距 (正) 系统字体。 | | **ANSI\_VAR\_FONT** | Windows 可变间距 (比例空间) 系统字体。 | | **DEVICE\_DEFAULT\_FONT** | 设备依赖字体。 | | **DEFAULT\_GUI\_FONT** | 用户界面对象（如菜单和对话框）的默认字体。 不建议使用DEFAULT\_GUI\_FONT或SYSTEM\_FONT来获取对话框和窗口使用的字体;有关详细信息，请参阅备注部分。  默认字体为 Tahoma。 | | **OEM\_FIXED\_FONT** | 原始设备制造商 (OEM) 依赖固定间距 (正方形) 字体。 | | **SYSTEM\_FONT** | 系统字体。 默认情况下，系统使用系统字体来绘制菜单、对话框控件和文本。 不建议使用DEFAULT\_GUI\_FONT或SYSTEM\_FONT来获取对话框和窗口使用的字体;有关详细信息，请参阅备注部分。  默认系统字体为 Tahoma。 | | **SYSTEM\_FIXED\_FONT** | 固定间距 (单调) 系统字体。 此库存对象仅用于与 3.0 之前的 16 位 Windows 版本兼容。 | | **DEFAULT\_PALETTE** | 默认调色板。 此调色板由系统调色板中的静态颜色组成。 |  返回值 如果函数成功，则返回值是请求的逻辑对象的句柄。  如果函数失败，则返回值为 NULL。 注解 不建议使用此方法来获取对话框和窗口使用的当前字体。 请改用带有 SPI\_GETNONCLIENTMETRICS 参数的 [SystemParametersInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-systemparametersinfoa) 函数来检索当前字体。 [SystemParametersInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-systemparametersinfoa) 将考虑当前主题，并提供标题、菜单和消息对话框的字体信息。  仅在具有CS\_HREDRAW和CS\_VREDRAW样式的窗口中使用DKGRAY\_BRUSH、GRAY\_BRUSH和LTGRAY\_BRUSH库存对象。 在任何其他窗口样式中使用灰色股票画笔可能会导致在移动窗口或调整大小后画笔图案不对齐。 无法调整股票画笔的来源。  HOLLOW\_BRUSH和NULL\_BRUSH库存对象是等效的。  不需要 (但) 调用 [DeleteObject](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-deleteobject) 删除库存对象并不有害。  DC\_BRUSH和DC\_PEN都可以与其他库存对象（如BLACK\_BRUSH和BLACK\_PEN）互换使用。 有关检索当前笔或画笔颜色的信息，请参阅 [GetDCBrushColor](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-getdcbrushcolor) 和 [GetDCPenColor](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-getdcpencolor)。 有关设置颜色的示例，请参阅 [设置笔或画笔](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/setting-the-pen-or-brush-color) 颜色。 参数为 DC\_BRUSH 或 DC\_PEN 的 **GetStockObject** 函数可与 [SetDCPenColor](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-setdcpencolor) 和 [SetDCBrushColor](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-setdcbrushcolor) 函数互换使用。 示例 有关示例，请参阅 [设置笔或画笔颜色](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/setting-the-pen-or-brush-color)。 要求  |  |  | | --- | --- | | **标头** | wingdi.h (包括 Windows.h) | | **Library** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [DeleteObject](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-deleteobject)  [设备上下文函数](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/device-context-functions)  [设备上下文概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/device-contexts)  [SelectObject](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-selectobject) |

# 演练

## 1.新建一个win32项目起名：Lesson41-logfont-demo

|  |
| --- |
|  |

## 2.定位到WndProc函数中，先添加一下变量，注意是静态的

|  |
| --- |
|  |

## 3.添加对WM\_CREATE消息的响应处理代码，在这里创建字体

|  |
| --- |
|  |

## 也可以不在这里创建，在WM\_PAINT消息处理函数里面创建

|  |
| --- |
|  |

## 4.然后我们在WM\_PAINT消息处理代码中使用新建的字体输出文字，注意在这里，如果你在WM\_CREATE消息处理函数里面创建的字体，不能把新字体删除，否则一旦改变窗口大小，原来的字体就消失，在WM\_PAINT代码创建的字体就可以删除

|  |
| --- |
|  |

## 5.创建字体还可以直接使用CreateFont函数，它把LOGFONT结构体的所有成员都放到参数里面，好像参数很多，但是绝大部分都可以用0来代替。也有它便捷的地方，可以根据你的需要使用。

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 6.其实建议你使用CreateFontIndirect函数，因为可以先把logfont结构体所有成员设置为0，然后设置几个不为0的成员，这样子更加便捷，

### 注意：在windows编程中，很多API的参数你传递0给他，并不是就是一定是0，他的意思是使用系统默认的值，这一点需要注意，因为很多时候0相当于FALSE

## 7.其实这个程序改变没有必要响应WM\_CREATE消息，我们可以把这个代码移动到WM\_PAINT消息响应代码中

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

### 注意：字体的宽度是系统自动计算的，一般设置为0，除非你有特别需要，不要去设置，在一下API中，如这里如果是字体粗细属性，0代表FW\_DONTCARE,也就是不介意，也就是用系统默认值

## 8.如果你不想自己创建字体，也可以使用系统字体，可以用GetStockObject(SYSTEM\_FONT)来获取系统字体

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 11.其实微软也预先定义了一下字体,可以直接拿来用

|  |
| --- |
|  |

## 如,我们使用一个14的字体

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 12.其实还是建议根据你的需要自己创建,

|  |
| --- |
|  |
|  |

# 这一节学习到此为止,完整代码如下

## Lesson41-logfont-demo.cpp

|  |
| --- |
| // Lesson41-logfont-demo.cpp : 定义应用程序的入口点。  //  #include "stdafx.h"  #include "Lesson41-logfont-demo.h"  #define MAX\_LOADSTRING 100  // 全局变量:  HINSTANCE hInst; // 当前实例  TCHAR szTitle[MAX\_LOADSTRING]; // 标题栏文本  TCHAR szWindowClass[MAX\_LOADSTRING]; // 主窗口类名  // 此代码模块中包含的函数的前向声明:  ATOM MyRegisterClass(HINSTANCE hInstance);  BOOL InitInstance(HINSTANCE, int);  LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);  INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);  int APIENTRY \_tWinMain(HINSTANCE hInstance,  HINSTANCE hPrevInstance,  LPTSTR lpCmdLine,  int nCmdShow)  {  UNREFERENCED\_PARAMETER(hPrevInstance);  UNREFERENCED\_PARAMETER(lpCmdLine);  // TODO: 在此放置代码。  MSG msg;  HACCEL hAccelTable;  // 初始化全局字符串  LoadString(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);  LoadString(hInstance, IDC\_LESSON41LOGFONTDEMO, szWindowClass, MAX\_LOADSTRING);  MyRegisterClass(hInstance);  // 执行应用程序初始化:  if (!InitInstance (hInstance, nCmdShow))  {  return FALSE;  }  hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_LESSON41LOGFONTDEMO));  // 主消息循环:  while (GetMessage(&msg, NULL, 0, 0))  {  if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))  {  TranslateMessage(&msg);  DispatchMessage(&msg);  }  }  return (int) msg.wParam;  }  //  // 函数: MyRegisterClass()  //  // 目的: 注册窗口类。  //  // 注释:  //  // 仅当希望  // 此代码与添加到 Windows 95 中的“RegisterClassEx”  // 函数之前的 Win32 系统兼容时，才需要此函数及其用法。调用此函数十分重要，  // 这样应用程序就可以获得关联的  // “格式正确的”小图标。  //  ATOM MyRegisterClass(HINSTANCE hInstance)  {  WNDCLASSEX wcex;  wcex.cbSize = sizeof(WNDCLASSEX);  wcex.style = CS\_HREDRAW | CS\_VREDRAW;  wcex.lpfnWndProc = WndProc;  wcex.cbClsExtra = 0;  wcex.cbWndExtra = 0;  wcex.hInstance = hInstance;  wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_LESSON41LOGFONTDEMO));  wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);  wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);  wcex.lpszMenuName = MAKEINTRESOURCE(IDC\_LESSON41LOGFONTDEMO);  wcex.lpszClassName = szWindowClass;  wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));  return RegisterClassEx(&wcex);  }  //  // 函数: InitInstance(HINSTANCE, int)  //  // 目的: 保存实例句柄并创建主窗口  //  // 注释:  //  // 在此函数中，我们在全局变量中保存实例句柄并  // 创建和显示主程序窗口。  //  BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)  {  HWND hWnd;  hInst = hInstance; // 将实例句柄存储在全局变量中  hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,  CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);  if (!hWnd)  {  return FALSE;  }  ShowWindow(hWnd, nCmdShow);  UpdateWindow(hWnd);  return TRUE;  }  //  // 函数: WndProc(HWND, UINT, WPARAM, LPARAM)  //  // 目的: 处理主窗口的消息。  //  // WM\_COMMAND - 处理应用程序菜单  // WM\_PAINT - 绘制主窗口  // WM\_DESTROY - 发送退出消息并返回  //  //  LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  {  int wmId, wmEvent;  PAINTSTRUCT ps;  HDC hdc;  static LOGFONT lf;  static HFONT hFont,hOrgFont;  switch (message)  {  case WM\_COMMAND:  wmId = LOWORD(wParam);  wmEvent = HIWORD(wParam);  // 分析菜单选择:  switch (wmId)  {  case IDM\_ABOUT:  DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);  break;  case IDM\_EXIT:  DestroyWindow(hWnd);  break;  default:  return DefWindowProc(hWnd, message, wParam, lParam);  }  break;  case WM\_PAINT:  hdc = BeginPaint(hWnd, &ps);  // TODO: 在此添加任意绘图代码...  memset(&lf,0,sizeof(lf));  lf.lfEscapement = 100;//文字旋转10°  lf.lfCharSet =1 ;//使用系统默认的字符集  lstrcpy(lf.lfFaceName ,\_T("黑体"));  lf.lfHeight = 64;  lf.lfItalic=1;//使用斜体  hFont = CreateFontIndirect(&lf);//创建新字体方法1：使用LOGFONT结构体创建字体,推荐使用这种方法  //hFont = CreateFont(70,0,0,0,0,0,0,0,1,0,0,0,0,\_T("黑体")); //方法2.使用CreateFont函数，参数挺多，不过可以使用0代替  //方法3.使用系统字体  /\*hFont =(HFONT)GetStockObject(SYSTEM\_FONT);\*/  //hFont =(HFONT)GetStockObject(DEVICE\_DEFAULT\_FONT);  hOrgFont = (HFONT)SelectObject(hdc,hFont);  TextOut(hdc,100,180,\_T("好好学习vc++"),8);  SelectObject(hdc,hOrgFont);  DeleteObject(hFont);  EndPaint(hWnd, &ps);  break;  case WM\_DESTROY:  PostQuitMessage(0);  break;  default:  return DefWindowProc(hWnd, message, wParam, lParam);  }  return 0;  }  // “关于”框的消息处理程序。  INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)  {  UNREFERENCED\_PARAMETER(lParam);  switch (message)  {  case WM\_INITDIALOG:  return (INT\_PTR)TRUE;  case WM\_COMMAND:  if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)  {  EndDialog(hDlg, LOWORD(wParam));  return (INT\_PTR)TRUE;  }  break;  }  return (INT\_PTR)FALSE;  } |

### 注意:在MFC中选择字体对话框需要使用CFontDialog,不要在MFC中使用ChooseFont函数,不会有字体对话框,看看下面的实例

|  |
| --- |
|  |

## 对话框类的源代码

|  |
| --- |
| // logfotdemoDlg.cpp: 实现文件  //  #include "pch.h"  #include "framework.h"  #include "logfotdemo.h"  #include "logfotdemoDlg.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  #ifdef AFX\_DESIGN\_TIME  enum { IDD = IDD\_ABOUTBOX };  #endif  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  DECLARE\_MESSAGE\_MAP()  };  CAboutDlg::CAboutDlg() : CDialogEx(IDD\_ABOUTBOX)  {  }  void CAboutDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)  END\_MESSAGE\_MAP()  // ClogfotdemoDlg 对话框  ClogfotdemoDlg::ClogfotdemoDlg(CWnd\* pParent /\*=nullptr\*/)  : CDialogEx(IDD\_LOGFOTDEMO\_DIALOG, pParent)  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  }  void ClogfotdemoDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(ClogfotdemoDlg, CDialogEx)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  ON\_WM\_LBUTTONDBLCLK()  END\_MESSAGE\_MAP()  // ClogfotdemoDlg 消息处理程序  BOOL ClogfotdemoDlg::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 != nullptr)  {  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 ClogfotdemoDlg::OnSysCommand(UINT nID, LPARAM lParam)  {  if ((nID & 0xFFF0) == IDM\_ABOUTBOX)  {  CAboutDlg dlgAbout;  dlgAbout.DoModal();  }  else  {  CDialogEx::OnSysCommand(nID, lParam);  }  }  void UseFont1(HWND hwnd)  {  HDC hdc;  HFONT hfOld, hfNew;  LOGFONT lf;  memset(&lf, 0, sizeof(LOGFONT));  lstrcpy(lf.lfFaceName, L"华文行楷");  lf.lfHeight = 48;  lf.lfCharSet = 1;  lf.lfEscapement = 100;//旋转10°  lf.lfItalic = 1;  hfNew = CreateFontIndirect(&lf);  hdc = ::GetDC(hwnd);  hfOld = (HFONT)SelectObject(hdc, hfNew);  SetBkMode(hdc, TRANSPARENT);//把背景设置为透明  TextOut(hdc, 10, 80, L"好好学习,天天进步", lstrlen(L"好好学习,天天进步"));  SelectObject(hdc, hfOld);  DeleteObject(hfNew);  ::ReleaseDC(hwnd, hdc);  }  void UseFont2(HWND hwnd) //这个函数不好,没有字体对话框  {  CHOOSEFONT cf;  LOGFONT lf;  HDC hdc;  HFONT hfOld, hfNew;  memset(&lf, 0, sizeof(LOGFONT));  lf.lfHeight = 48;  lf.lfCharSet = 1;  lf.lfEscapement = 100;//旋转10°  lf.lfItalic = 1;  cf.lStructSize = sizeof(CHOOSEFONT);  cf.lpLogFont = &lf;  cf.Flags = CF\_SCREENFONTS;  cf.nFontType = SCREEN\_FONTTYPE;  hdc = ::GetDC(hwnd);  ChooseFont(&cf);//在MFC中没有对话框窗口!!  hfNew = CreateFontIndirect(cf.lpLogFont);  hfOld = (HFONT)SelectObject(hdc, hfNew);  SetBkMode(hdc, TRANSPARENT);//把背景设置为透明  TextOut(hdc, 10, 80, L"恭喜发财,红包拿来", lstrlen(L"恭喜发财,红包拿来"));  SelectObject(hdc, hfOld);  DeleteObject(hfNew);  ::ReleaseDC(hwnd, hdc);  }  // 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 ClogfotdemoDlg::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  {  //UseFont2(m\_hWnd);  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 ClogfotdemoDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  BOOL ClogfotdemoDlg::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:  case VK\_ESCAPE:  return TRUE;  default:  break;  }  }  return CDialogEx::PreTranslateMessage(pMsg);  }  void ClogfotdemoDlg::OnLButtonDblClk(UINT nFlags, CPoint point)  {  // TODO: 在此添加消息处理程序代码和/或调用默认值  CClientDC dc(this);  HFONT hfNew;  CFont\* pNewFnt, \* pOldFnt;  LOGFONT lf;  CFontDialog dlg;  if (IDOK == dlg.DoModal())  {  dlg.GetCurrentFont(&lf);  hfNew = ::CreateFontIndirect(&lf);  pNewFnt = CFont::FromHandle(hfNew);  pOldFnt = dc.SelectObject(pNewFnt);  dc.TextOut(10, 10, L"恭喜发财,红包拿来");  dc.SelectObject(pOldFnt);  ::DeleteObject(hfNew);  }  CDialogEx::OnLButtonDblClk(nFlags, point);  } |