# 学习大纲

|  |
| --- |
|  |

## TextOut函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TextOut** 函数使用当前所选字体、背景色和文本颜色在指定位置写入字符串。 语法 C++复制  BOOL TextOutA(  [in] HDC hdc,  [in] int x,  [in] int y,  [in] LPCSTR lpString,  [in] int c  ); 参数 [in] hdc  设备上下文的句柄。  [in] x  系统用于对齐字符串的引用点的 x 坐标（以逻辑坐标表示）。  [in] y  系统用于对齐字符串的引用点的 y 坐标（以逻辑坐标表示）。  [in] lpString  指向要绘制的字符串的指针。 字符串不需要以零结尾，因为 *cchString* 指定字符串的长度。  [in] c  *lpString* 指向[的字符串的长度](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/specifying-length-of-text-output-string)（以字符为单位）。 返回值 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。 注解 参考点的解释取决于当前文本对齐模式。 应用程序可以通过调用 [GetTextAlign](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-gettextalign) 函数来检索此模式;应用程序可以通过调用 [SetTextAlign](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-settextalign) 函数来更改此模式。 可以使用以下值进行文本对齐。 只能从影响水平和垂直对齐的标志中选择一个标志。 此外，只能选择更改当前位置的两个标志中的一个。  展开表   |  |  | | --- | --- | | **术语** | **说明** | | TA\_BASELINE | 引用点将位于文本的基行上。 | | TA\_BOTTOM | 引用点将位于边界矩形的下边缘。 | | TA\_TOP | 引用点将位于边界矩形的上边缘。 | | TA\_CENTER | 参考点将与边界矩形的中心水平对齐。 | | TA\_LEFT | 引用点将位于边界矩形的左边缘。 | | TA\_RIGHT | 引用点将位于边界矩形的右边缘。 | | TA\_NOUPDATECP | 每次文本输出调用后，当前位置不会更新。 引用点将传递给文本输出函数。 | | TA\_RTLREADING | **Windows 中东语言版本：** 文本按从右到左的阅读顺序进行布局，而不是默认的从左到右的顺序。 仅当在设备上下文中选择的字体是希伯来语或阿拉伯语时才适用。 | | TA\_UPDATECP | 每次文本输出调用后，当前位置都会更新。 当前位置用作参考点。 |     默认情况下，此函数不使用或更新当前位置。 但是，应用程序可以调用 [SetTextAlign](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-settextalign) 函数，并将 *fMode* 参数设置为 TA\_UPDATECP，以允许系统在每次应用程序调用指定设备上下文的 **TextOut** 时使用和更新当前位置。 设置此标志后，系统会在后续 **TextOut** 调用中忽略 *nXStart* 和 *nYStart* 参数。  将 **TextOut** 函数置于路径括号内时，系统会为包含每个字符及其字符框的 TrueType 文本生成路径。 生成的区域是字符框减去文本，而不是文本本身。 在将 **TextOut** 函数置于路径括号中之前，可以通过将背景模式设置为透明来获取 TrueType 文本轮廓所包围的区域。 下面是演示此过程的示例代码。  C++复制  // Obtain the window's client rectangle  GetClientRect(hwnd, &r);  // THE FIX: by setting the background mode  // to transparent, the region is the text itself  // SetBkMode(hdc, TRANSPARENT);  // Bracket begin a path  BeginPath(hdc);  // Send some text out into the world  TCHAR text[ ] = "Defenestration can be hazardous";  TextOut(hdc,r.left,r.top,text, ARRAYSIZE(text));  // Bracket end a path  EndPath(hdc);  // Derive a region from that path  SelectClipPath(hdc, RGN\_AND);  // This generates the same result as SelectClipPath()  // SelectClipRgn(hdc, PathToRegion(hdc));  // Fill the region with grayness  FillRect(hdc, &r, GetStockObject(GRAY\_BRUSH)); 示例 有关示例，请参阅 [枚举已安装的字体](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/enumerating-the-installed-fonts)。  **备注**  wingdi.h 标头将 TextOut 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)。 |

## TabbedTextOut函数的用法

|  |
| --- |
| **TabbedTextOut** 函数在指定位置写入字符串，将制表符扩展到制表位位置数组中指定的值。 文本以当前所选字体、背景色和文本颜色书写。 语法 C++复制  LONG TabbedTextOutA(  [in] HDC hdc,  [in] int x,  [in] int y,  [in] LPCSTR lpString,  [in] int chCount,  [in] int nTabPositions,  [in] const INT \*lpnTabStopPositions,  [in] int nTabOrigin  ); 参数 [in] hdc  设备上下文的句柄。  [in] x  字符串起点的 x 坐标，以逻辑单位表示。  [in] y  字符串起点的 y 坐标（以逻辑单元为单位）。  [in] lpString  指向要绘制的字符串的指针。 字符串不需要以零结尾，因为 *nCount* 指定字符串的长度。  [in] chCount  *lpString* 指向[的字符串的长度](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/specifying-length-of-text-output-string)。  [in] nTabPositions  制表位位置数组中的值数。  [in] lpnTabStopPositions  指向包含制表位位置（以逻辑单元为单位）的数组的指针。 制表位必须按升序排列；最小的 x 值应该是数组中的第一项。  [in] nTabOrigin  展开选项卡的起始位置的 x 坐标（以逻辑单元为单位）。 返回值 如果函数成功，则返回值为字符串的维度（以逻辑单位为单位）。 高度在高序字中，宽度在低位字中。  如果函数失败，则返回值为零。 注解 如果 *nTabPositions* 参数为零， *lpnTabStopPositions* 参数为 **NULL**，则选项卡将扩展为平均字符宽度的八倍。  如果 *nTabPositions* 为 1，则制表位由 *lpnTabStopPositions* 数组中的第一个值指定的距离分隔。  如果 *lpnTabStopPositions* 数组包含多个值，则为数组中的每个值设置制表位，最多为 *nTabPositions* 指定的数字。  *nTabOrigin* 参数允许应用程序为单行多次调用 **TabbedTextOut** 函数。 如果应用程序多次调用 **TabbedTextOut** ，每次将 *nTabOrigin* 设置为相同的值，则函数将相对于 *nTabOrigin* 指定的位置展开所有选项卡。  默认情况下， **TabbedTextOut** 函数不使用或更新当前位置。 如果应用程序在调用 **TabbedTextOut** 时需要更新当前位置，则应用程序可以使用 *wFlags* 参数设置为 TA\_UPDATECP调用 [SetTextAlign](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-settextalign) 函数。 设置此标志后，系统会在后续调用 **TabbedTextOut** 函数时忽略 *X* 和 *Y* 参数，改用当前位置。  **注意** 对于 Windows Vista 及更高版本， **TabbedTextOut** 在绘制文本时忽略文本对齐方式。    **备注**  winuser.h 标头将 TabbedTextOut 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)。 |

## ExtTextOut函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ExtTextOut** 函数使用当前所选字体、背景色和文本颜色绘制文本。 可以选择提供用于剪辑和/或不透明的尺寸。 语法 C++复制  BOOL ExtTextOutA(  [in] HDC hdc,  [in] int x,  [in] int y,  [in] UINT options,  [in] const RECT \*lprect,  [in] LPCSTR lpString,  [in] UINT c,  [in] const INT \*lpDx  ); 参数 [in] hdc  设备上下文的句柄。  [in] x  用于定位字符串的引用点的 x 坐标（以逻辑坐标表示）。  [in] y  用于定位字符串的引用点的 y 坐标（以逻辑坐标为单位）。  [in] options  指定如何使用应用程序定义的矩形。 此参数可使用以下一个或多个值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **ETO\_CLIPPED** | 文本将被剪裁为矩形。 | | **ETO\_GLYPH\_INDEX** | *lpString* 数组是指从 [GetCharacterPlacement](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-getcharacterplacementa) 返回的数组，应该由 GDI 直接分析，因为不需要进一步特定于语言的处理。 字形索引仅适用于 TrueType 字体，但标志可用于位图和矢量字体，以指示不需要进一步的语言处理，GDI 应直接处理字符串。 请注意，所有字形索引都是 16 位值，即使假定字符串是光栅字体的 8 位值的数组。  对于 ExtTextOutW，字形索引将保存到图元文件。 但是，若要显示正确的字符，必须使用相同的字体播放图元文件。 对于 ExtTextOutA，不保存字形索引。 | | **ETO\_IGNORELANGUAGE** | 预留给系统使用。 如果应用程序设置此标志，它将失去国际脚本支持，在某些情况下，它可能根本不显示任何文本。 | | **ETO\_NUMERICSLATIN** | 若要显示数字，请使用欧洲数字。 | | **ETO\_NUMERICSLOCAL** | 若要显示数字，请使用适合区域设置的数字。 | | **ETO\_OPAQUE** | 应使用当前背景色来填充矩形。 | | **ETO\_PDY** | 设置此项后， *lpDx* 指向的数组包含值对。 与往常一样，每对的第一个值是相邻字符单元格的原点之间的距离，但第二个值是沿字体垂直方向的位移。 | | **ETO\_RTLREADING** | **Windows 中东语言版本：** 如果指定了此值，并在设备上下文中选择了希伯来语或阿拉伯语字体，则使用从右到左的阅读顺序输出字符串。 如果未指定此值，则按从左到右的顺序输出字符串。 可以通过在 [SetTextAlign](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-settextalign) 中设置TA\_RTLREADING值来实现相同的效果。 保留此值是为了向后兼容。 |     ETO\_GLYPH\_INDEX值和ETO\_RTLREADING值不能一起使用。 由于 ETO\_GLYPH\_INDEX 表示所有语言处理都已完成，因此如果还指定了ETO\_RTLREADING标志，则函数将忽略该标志。  [in] lprect  指向可选 [RECT](https://learn.microsoft.com/zh-cn/windows/desktop/api/windef/ns-windef-rect) 结构的指针，该结构指定用于剪裁和/或不透明矩形的维度（以逻辑坐标表示）。  [in] lpString  指向指定要绘制的文本的字符串的指针。 字符串不需要以零结尾，因为 *cbCount* 指定字符串的长度。  [in] c  *lpString* 指向[的字符串的长度](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/specifying-length-of-text-output-string)。  此值不能超过 8192。  [in] lpDx  指向可选值数组的指针，这些值指示相邻字符单元格的原点之间的距离。 例如，lpDx[*i*] 逻辑单元分隔字符单元格 *i* 和字符单元格 *i* + 1 的原点。 返回值 如果绘制字符串，则返回值为非零值。 但是，如果使用 ETO\_GLYPH\_INDEX 调用 **ExtTextOut** 的 ANSI 版本，则即使函数不执行任何操作，函数也会返回 **TRUE** 。  如果函数失败，则返回值为零。 注解 指定设备上下文的当前文本对齐设置决定了如何使用引用点来定位文本。 通过调用 [GetTextAlign](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-gettextalign) 函数检索文本对齐设置。 通过调用 [SetTextAlign](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-settextalign) 函数更改文本对齐设置。 可以使用以下值进行文本对齐。 只能从影响水平和垂直对齐的标志中选择一个标志。 此外，只能选择更改当前位置的两个标志中的一个。  展开表   |  |  | | --- | --- | | **术语** | **说明** | | TA\_BASELINE | 引用点将位于文本的基行上。 | | TA\_BOTTOM | 引用点将位于边界矩形的下边缘。 | | TA\_TOP | 引用点将位于边界矩形的上边缘。 | | TA\_CENTER | 参考点将与边界矩形的中心水平对齐。 | | TA\_LEFT | 引用点将位于边界矩形的左边缘。 | | TA\_RIGHT | 引用点将位于边界矩形的右边缘。 | | TA\_NOUPDATECP | 每次文本输出调用后，当前位置不会更新。 引用点将传递给文本输出函数。 | | TA\_RTLREADING | **Windows 中东语言版本：** 文本按从右到左的阅读顺序进行布局，而不是默认的从左到右的顺序。 仅当在设备上下文中选择的字体是希伯来语或阿拉伯语时才适用。 | | TA\_UPDATECP | 每次文本输出调用后，当前位置都会更新。 当前位置用作参考点。 |     如果 *lpDx* 参数为 **NULL**， **则 ExtTextOut** 函数使用字符之间的默认间距。 字符单元格原点和 *lpDx* 参数指向的数组的内容按逻辑单元指定。 字符单元格原点定义为字符单元格的左上角。  默认情况下，此函数不使用或更新当前位置。 但是，应用程序可以调用 [SetTextAlign](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-settextalign) 函数，并将 *fMode* 参数设置为 TA\_UPDATECP，以允许系统在每次为指定设备上下文调用 **ExtTextOut** 时使用和更新当前位置。 设置此标志后，系统会在后续 **ExtTextOut** 调用中忽略 *X* 和 *Y* 参数。  对于 **ExtTextOut** 的 ANSI 版本， *lpDx* 数组的 INT 值数与 *lpString* 中的字节数相同。 对于 DBCS 字符，只要两个字节的总和加起来达到所需的 dx，就可以在前导字节和尾部字节之间的 *lpDx* 条目中分配 dx。 对于具有 **Unicode 版本的 ExtTextOut** 的 DBCS 字符，每个 Unicode 字形获取一个 *pdx* 条目。  请注意，[GetTextExtentExPoint](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-gettextextentexpointa) 中的 *alpDx* 值与 **ExtTextOut** 的 *lpDx* 值不同。 若要在 *lpDx 中使用 alpDx* 值，必须先处理它们。  **ExtTextOut** 将在必要时使用 [Uniscribe](https://learn.microsoft.com/zh-cn/windows/win32/intl/uniscribe) ，从而导致字体回退。 ETO\_IGNORELANGUAGE标志将禁止此行为，不应传递。  此外， **ExtTextOut** 将在转换到内核模式之前执行内部批处理调用，从而缓解 **了在权衡 PolyTextOut** 与 **ExtTextOut** 使用情况时的一些性能问题。  **提示**  强烈建议使用 **ExtTextOut** 替代 **PolyTextOut** 进行新式开发，因为它能够处理不同语言的显示。 示例 有关示例，请参阅 [使用菜单中](https://learn.microsoft.com/zh-cn/windows/desktop/menurc/using-menus)的“设置 Menu-Item 文本字符串的字体”。  **备注**  wingdi.h 标头将 ExtTextOut 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)。 |

## DrawText函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **DrawText** 函数在指定矩形中绘制带格式的文本。 它根据指定的方法格式化文本 (展开制表符、对齐字符、断行等) 。  若要指定其他格式设置选项，请使用 [DrawTextEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-drawtextexa) 函数。 语法 int DrawText(  [in] HDC hdc,  [in, out] LPCTSTR lpchText,  [in] int cchText,  [in, out] LPRECT lprc,  [in] UINT format  ); 参数 [in] hdc  设备上下文的句柄。  [in, out] lpchText  指向指定要绘制的文本的字符串的指针。 如果 *nCount* 参数为 -1，则字符串必须以 null 结尾。  如果 *uFormat* 包含DT\_MODIFYSTRING，该函数最多可为此字符串添加四个附加字符。 包含字符串的缓冲区应足够大，以便容纳这些额外的字符。  [in] cchText  字符串的长度（以字符为单位）。 如果 *nCount* 为 -1，则 *假定 lpchText* 参数是指向以 null 结尾的字符串的指针， **DrawText** 会自动计算字符计数。  [in, out] lprc  指向 [RECT](https://learn.microsoft.com/zh-cn/windows/desktop/api/windef/ns-windef-rect) 结构的指针，该结构包含要设置文本格式) 逻辑坐标 (矩形。  [in] format  设置文本格式的方法。 此参数可使用以下一个或多个值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **DT\_BOTTOM** | 将文本与矩形底部对齐。 此值仅用于DT\_SINGLELINE值。 | | **DT\_CALCRECT** | 确定矩形的宽度和高度。 如果存在多行文本， **DrawText** 将使用 *lpRect* 参数指向的矩形宽度，并扩展矩形的基底以绑定最后一行文本。 如果最大字比矩形宽，则宽度将展开。 如果文本小于矩形的宽度，宽度将减小。 如果只有一行文本， **DrawText** 将修改矩形的右侧，使其绑定行中的最后一个字符。 在任一情况下， **DrawText** 均返回带格式文本的高度，但不绘制文本。 | | **DT\_CENTER** | 在矩形中水平居中放置文本。 | | **DT\_EDITCONTROL** | 复制多行编辑控件的文本显示特征。 具体而言，平均字符宽度的计算方式与编辑控件的计算方式相同，并且 函数不显示部分可见的最后一行。 | | **DT\_END\_ELLIPSIS** | 对于显示的文本，如果字符串的末尾不适合矩形，则会将其截断并添加省略号。 如果不在字符串末尾的单词超出矩形的限制，则会将其截断，且不带省略号。  除非指定了DT\_MODIFYSTRING标志，否则不会修改字符串。  与DT\_PATH\_ELLIPSIS和DT\_WORD\_ELLIPSIS进行比较。 | | **DT\_EXPANDTABS** | 扩展制表符。 每个制表符的默认字符数是 8。 DT\_WORD\_ELLIPSIS、DT\_PATH\_ELLIPSIS和DT\_END\_ELLIPSIS值不能与DT\_EXPANDTABS值一起使用。 | | **DT\_EXTERNALLEADING** | 在行高中包括字体外部间隙。 通常，外部前导不包括在文本行的高度中。 | | **DT\_HIDEPREFIX** | 忽略文本中的与 (&) 前缀字符。 后面的字母不会加下划线，但仍会处理其他助记符前缀字符。  例如：  输入字符串：“A&bc&&d”  normal：“Abc&d”  DT\_HIDEPREFIX：“Abc&d”  与DT\_NOPREFIX和DT\_PREFIXONLY进行比较。 | | **DT\_INTERNAL** | 使用系统字体计算文本规格。 | | **DT\_LEFT** | 将文本左对齐。 | | **DT\_MODIFYSTRING** | 修改指定的字符串以匹配显示的文本。 除非指定DT\_END\_ELLIPSIS或DT\_PATH\_ELLIPSIS，否则此值无效。 | | **DT\_NOCLIP** | 绘制时不剪裁。 使用 DT\_NOCLIP 时**，DrawText** 的速度会稍快一些。 | | **DT\_NOFULLWIDTHCHARBREAK** | 防止在 DBCS (双宽字符串) 换行，使换行规则等效于 SBCS 字符串。 例如，这可以在朝鲜语窗口中使用，以便提高图标标签的可读性。 除非指定DT\_WORDBREAK，否则此值无效。 | | **DT\_NOPREFIX** | 关闭对前缀字符的处理。 通常， **DrawText** 将助记键前缀字符 & 解释为指令以下划线后面的字符，将助记键前缀字符 && 作为指令来打印单个 &。 通过指定DT\_NOPREFIX，此处理将关闭。 例如，  例如：  输入字符串：“A&bc&&d”  normal：“Abc&d”  DT\_NOPREFIX：“A&bc&&d”  与DT\_HIDEPREFIX和DT\_PREFIXONLY进行比较。 | | **DT\_PATH\_ELLIPSIS** | 对于显示的文本，将字符串中间的字符替换为省略号，以便结果适合指定的矩形。 如果字符串包含反斜杠 (\\) 个字符，DT\_PATH\_ELLIPSIS将尽可能多地保留最后一个反斜杠后的文本。  除非指定了DT\_MODIFYSTRING标志，否则不会修改字符串。  与DT\_END\_ELLIPSIS和DT\_WORD\_ELLIPSIS进行比较。 | | **DT\_PREFIXONLY** | 仅在与字符后的位置绘制下划线， (&) 前缀字符。 不绘制字符串中的任何其他字符。 例如，  例如：  输入字符串：“A&bc&&d”n  normal：“Abc&d”  DT\_PREFIXONLY：“\_ ”  与DT\_HIDEPREFIX和DT\_NOPREFIX进行比较。 | | **DT\_RIGHT** | 使文本向右对齐。 | | **DT\_RTLREADING** | 当在 *hdc* 中选择的字体是希伯来语或阿拉伯语字体时，双向文本的从右到左阅读顺序布局。 所有文本的默认阅读顺序都是从左到右。 | | **DT\_SINGLELINE** | 仅显示单行上的文本。 回车符和换行符不会中断该行。 | | **DT\_TABSTOP** | 设置制表位。 15-8 位 (*uFormat* 参数的低序字) 的高阶字节指定每个选项卡的字符数。每个选项卡的默认字符数为 8。 DT\_CALCRECT、DT\_EXTERNALLEADING、DT\_INTERNAL、DT\_NOCLIP和DT\_NOPREFIX值不能与DT\_TABSTOP值一起使用。 | | **DT\_TOP** | 使文本与矩形顶部对齐。 | | **DT\_VCENTER** | 垂直居中文本。 此值仅用于DT\_SINGLELINE值。 | | **DT\_WORDBREAK** | 中断字词。 如果单词超出 *lpRect* 参数指定的矩形边缘，则单词之间的行会自动断开。 回车符换行序列也会中断该行。  如果未指定此项，则输出位于一行中。 | | **DT\_WORD\_ELLIPSIS** | 截断矩形中不适合的任何单词，并添加省略号。  与DT\_END\_ELLIPSIS和DT\_PATH\_ELLIPSIS进行比较。 |  返回值 如果函数成功，则返回值是文本的高度（以逻辑单位为单位）。 如果指定了DT\_VCENTER或DT\_BOTTOM，则返回值是从 lpRect->top 到绘制文本底部的偏移量。  如果函数失败，则返回值为零。 注解 **DrawText** 函数使用设备上下文的所选字体、文本颜色和背景色来绘制文本。 除非使用DT\_NOCLIP格式， **否则 DrawText** 将剪裁文本，使其不显示在指定矩形之外。 请注意，可能会剪裁具有明显悬垂的文本，例如，文本字符串中的初始“W”或斜体文本。 除非指定了DT\_SINGLELINE格式，否则假定所有格式都包含多行。  如果所选字体对于指定的矩形来说太大， **则 DrawText** 函数不会尝试替换较小的字体。  设备上下文的文本对齐模式必须包括TA\_LEFT、TA\_TOP和TA\_NOUPDATECP标志。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 Windows 8) 中引入的 ext-ms-win-ntuser-misc-l1-1-0 ( |  另请参阅 [DrawTextEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-drawtextexa)  [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)  [GrayString](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-graystringa)  [矩形](https://learn.microsoft.com/zh-cn/windows/desktop/api/windef/ns-windef-rect)  [TabbedTextOut](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-tabbedtextouta)  [TextOut](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-textouta) |

### DrawText函数适合输出大段的文本

## DrawTextEx函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **DrawTextEx** 函数在指定的矩形中绘制格式化文本。 语法 int DrawTextExA(  [in] HDC hdc,  [in, out] LPSTR lpchText,  [in] int cchText,  [in, out] LPRECT lprc,  [in] UINT format,  [in] LPDRAWTEXTPARAMS lpdtp  ); 参数 [in] hdc  要在其中绘制的设备上下文的句柄。  [in, out] lpchText  指向包含要绘制的文本的字符串的指针。 如果 *cchText* 参数为 -1，则字符串必须以 null 结尾。  如果 *dwDTFormat* 包含DT\_MODIFYSTRING，则函数最多可为此字符串添加四个附加字符。 包含字符串的缓冲区应足够大，以容纳这些额外的字符。  [in] cchText  *lpchText* 指向[的字符串的长度](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/specifying-length-of-text-output-string)。 如果 *cchText* 为 -1，则 *假定 lpchText* 参数是指向以 null 结尾的字符串的指针， **DrawTextEx** 会自动计算字符计数。  [in, out] lprc  指向 [RECT](https://learn.microsoft.com/zh-cn/windows/desktop/api/windef/ns-windef-rect) 结构的指针，该结构包含要设置文本格式的逻辑坐标矩形。  [in] format  格式设置选项。 此参数可使用以下一个或多个值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **DT\_BOTTOM** | 使文本与矩形底部对齐。 此值仅用于DT\_SINGLELINE值。 | | **DT\_CALCRECT** | 确定矩形的宽度和高度。 如果存在多行文本， **DrawTextEx** 使用 *lprc* 参数指向的矩形的宽度，并扩展矩形的基以绑定最后一行文本。 如果只有一行文本， **DrawTextEx** 将修改矩形的右侧，使其绑定行中的最后一个字符。 在任一情况下， **DrawTextEx** 都返回格式化文本的高度，但不绘制文本。 | | **DT\_CENTER** | 在矩形中水平居中居中文本。 | | **DT\_EDITCONTROL** | 复制多行编辑控件的文本显示特征。 具体而言，平均字符宽度的计算方式与编辑控件的计算方式相同，并且函数不显示部分可见的最后一行。 | | **DT\_END\_ELLIPSIS** | 对于显示的文本，将字符串的末尾替换为省略号，以便结果适合指定的矩形。 超出矩形限制的字符串) 末尾的任何单词 (将被截断，而不用省略号。 除非指定了DT\_MODIFYSTRING标志，否则不会修改字符串。  与DT\_PATH\_ELLIPSIS和DT\_WORD\_ELLIPSIS进行比较。 | | **DT\_EXPANDTABS** | 扩展制表符。 每个制表符的默认字符数是 8。 | | **DT\_EXTERNALLEADING** | 在行高中包括字体外部间隙。 通常，外部前导不包括在文本行的高度中。 | | **DT\_HIDEPREFIX** | 忽略文本中的和号 (&) 前缀字符。 后面的字母不会加下划线，但仍会处理其他助记前缀字符。  例如：  输入字符串：“A&bc&&d”  normal： “Abc&d”  DT\_HIDEPREFIX：“Abc&d”  与 DT\_NOPREFIX 和 DT\_PREFIXONLY 进行比较。 | | **DT\_INTERNAL** | 使用系统字体计算文本规格。 | | **DT\_LEFT** | 将文本向左对齐。 | | **DT\_MODIFYSTRING** | 修改指定的字符串以匹配显示的文本。 除非指定了DT\_END\_ELLIPSIS或DT\_PATH\_ELLIPSIS，否则此值不起作用。 | | **DT\_NOCLIP** | 绘制时不进行剪裁。 使用DT\_NOCLIP时**，DrawTextEx** 的速度会稍快一些。 | | **DT\_NOFULLWIDTHCHARBREAK** | 防止 DBCS (双宽字符串) 的换行，使换行规则等效于 SBCS 字符串。 例如，这可以在朝鲜语窗口中使用，以便提高图标标签的可读性。 除非指定了DT\_WORDBREAK，否则此值不起作用。 | | **DT\_NOPREFIX** | 关闭对前缀字符的处理。 通常， **DrawTextEx** 将和和 (&) 助记前缀字符解释为指令，以突出显示后面的字符，将双和和 (&&) 助记前缀字符解释为指令以打印单个和号。 通过指定DT\_NOPREFIX，会关闭此处理。 与DT\_HIDEPREFIX和DT\_PREFIXONLY进行比较 | | **DT\_PATH\_ELLIPSIS** | 对于显示的文本，将字符串中间的字符替换为省略号，以便结果适合指定矩形。 如果字符串包含反斜杠 (\\) 字符，DT\_PATH\_ELLIPSIS在最后一个反斜杠之后保留尽可能多的文本。 除非指定了DT\_MODIFYSTRING标志，否则不会修改字符串。  与 DT\_END\_ELLIPSIS 和 DT\_WORD\_ELLIPSIS 进行比较。 | | **DT\_PREFIXONLY** | 仅在字符的后部绘制一个下划线，并 (&) 前缀字符。 不绘制字符串中的任何字符。  例如：  输入字符串：“A&bc&&d”  normal： “Abc&d”  PREFIXONLY： “ \_ ”  与DT\_NOPREFIX和DT\_HIDEPREFIX进行比较。 | | **DT\_RIGHT** | 将文本向右对齐。 | | **DT\_RTLREADING** | 当 *hdc* 中选择的字体是希伯来语或阿拉伯语字体时，双向文本的从右到左阅读顺序布局。 所有文本的默认阅读顺序都是从左到右。 | | **DT\_SINGLELINE** | 仅显示单行上的文本。 回车符和换行符不会中断该行。 | | **DT\_TABSTOP** | 设置制表位。 *lpDTParams* 参数指向的 [DRAWTEXTPARAMS](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/ns-winuser-drawtextparams) 结构指定每个制表位的平均字符宽度数。 | | **DT\_TOP** | 使文本与矩形顶部对齐。 | | **DT\_VCENTER** | 垂直居中文本。 此值仅用于DT\_SINGLELINE值。 | | **DT\_WORDBREAK** | 中断字词。 如果单词超出 *lprc* 参数指定的矩形边缘，则单词之间的行会自动断开。 回车换行序列也会中断该行。 | | **DT\_WORD\_ELLIPSIS** | 截断矩形中不适合的任何单词，并添加省略号。  与DT\_END\_ELLIPSIS和DT\_PATH\_ELLIPSIS进行比较。 |   [in] lpdtp  指向 [DRAWTEXTPARAMS](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/ns-winuser-drawtextparams) 结构的指针，该结构指定其他格式设置选项。 此参数可以为 NULL。 返回值 如果函数成功，则返回值为文本高度（以逻辑单元为单位）。 如果指定了DT\_VCENTER或DT\_BOTTOM，则返回值是绘制 lprc->top 文本从到底部的偏移量  如果函数失败，则返回值为零。 注解 **DrawTextEx** 函数仅支持转义和方向均为零的字体。  设备上下文的文本对齐模式必须包括TA\_LEFT、TA\_TOP和TA\_NOUPDATECP标志。  **备注**  winuser.h 标头将 DrawTextEx 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 [DRAWTEXTPARAMS](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/ns-winuser-drawtextparams)  [DrawText](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-drawtext)  [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) |

### 问文本题：这些输出函数有什么区别？

## 扩展：SetTextAlign函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SetTextAlign** 函数为指定的设备上下文设置文本对齐标志。 语法 C++复制  UINT SetTextAlign(  [in] HDC hdc,  [in] UINT align  ); 参数 [in] hdc  设备上下文的句柄。  [in] align  使用以下列表中值的掩码的文本对齐方式。 只能从影响水平和垂直对齐的标志中选择一个标志。 此外，只能选择更改当前位置的两个标志中的一个。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **TA\_BASELINE** | 参考点将位于文本的基行上。 | | **TA\_BOTTOM** | 参考点将位于边框的下边缘。 | | **TA\_TOP** | 参考点将位于边框的上边缘。 | | **TA\_CENTER** | 参考点将与边框中心水平对齐。 | | **TA\_LEFT** | 参考点将位于边框的左边缘。 | | **TA\_RIGHT** | 参考点将位于边框的右边缘。 | | **TA\_NOUPDATECP** | 每次文本输出调用后，当前位置不会更新。 引用点将传递给文本输出函数。 | | **TA\_RTLREADING** | **Windows 中东语言版本：** 文本按从右到左的阅读顺序排列，而不是默认的从左到右的顺序。 仅当在设备上下文中选择的字体是希伯来语或阿拉伯语时，这才适用。 | | **TA\_UPDATECP** | 每次文本输出调用后，当前位置都会更新。 当前位置用作参考点。 |     如果当前字体具有垂直默认基线（与汉字一样），则必须使用以下值，而不是TA\_BASELINE和TA\_CENTER。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **VTA\_BASELINE** | 参考点将位于文本的基行上。 | | **VTA\_CENTER** | 参考点将与边框的中心垂直对齐。 |     默认值为 TA\_LEFT、TA\_TOP 和 TA\_NOUPDATECP。 返回值 如果函数成功，则返回值为以前的文本对齐设置。  如果函数失败，则返回值GDI\_ERROR。 注解 [TextOut](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-textouta) 和 [ExtTextOut](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-exttextouta) 函数使用文本对齐标志在显示器或其他设备上放置文本字符串。 标志指定引用点与绑定文本的矩形之间的关系。 引用点是当前位置或传递给文本输出函数的点。  绑定文本的矩形由文本字符串中的字符单元格构成。  获取左对齐文本的最佳方式是使用任一  C++复制  SetTextAlign (hdc, GetTextAlign(hdc) & (~TA\_CENTER))  或  C++复制  SetTextAlign (hdc,TA\_LEFT | <other flags>)  还可以将 **SetTextAlign** (hdc TA\_LEFT) 用于此目的，但这会丢失任何垂直或从右到左的设置。  **注意**使用 [**ScriptStringOut**](https://learn.microsoft.com/zh-cn/windows/desktop/api/usp10/nf-usp10-scriptstringout) 时，不应将 **SetTextAlign** 与 TA\_UPDATECP 一起使用，因为所选文本未正确呈现。 如果必须使用此标志，可以根据需要取消设置并重置它以避免此问题。   示例 有关示例，请参阅 [设置文本对齐方式](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/setting-the-text-alignment)。   |  | | --- | | UINT uAlignPrev;  char szCount[8];  HRESULT hr;  size\_t \* pcch;    uAlignPrev = SetTextAlign(hdc, TA\_UPDATECP);  MoveToEx(hdc, 10, 50, (LPPOINT) NULL);  TextOut(hdc, 0, 0, "Number of Arial fonts: ", 23);  itoa(cArial, szCount, 10);  hr = StringCchLength(szCount, 9, pcch);  if (FAILED(hr))  {  // TODO: write error handler  }    TextOut(hdc, 0, 0, (LPSTR) szCount, \*pcch);  SetTextAlign(hdc, uAlignPrev); |  要求  |  |  | | --- | --- | | **目标平台** | Windows | | **标头** | wingdi.h (包括 Windows.h) | | **Library** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [ExtTextOut](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-exttextouta)  [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)  [GetTextAlign](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-gettextalign)  [ScriptStringOut](https://learn.microsoft.com/zh-cn/windows/desktop/api/usp10/nf-usp10-scriptstringout)  [TextOut](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-textouta) |

# 演练，学习DrawText函数

## 项目1练习输出长文本时的换行，取消裁剪，文本左中右对齐等等

## 1.新建一个win32应用程序项目，起名：Lesson37-drawtext-demo

|  |
| --- |
|  |

## 2.次数向导会帮我们生成一个应用程序，可以编译运行，然后我们找到WM\_PAINT消息的处理代码，在那里输出一些文字。首先我们需要在项目目录里面创建一个文本文件，叫做demo.txt。

|  |
| --- |
|  |
|  |

## 3.我们把文件的内容复制过来，放到一个字符数组里面,由于文本很长，如果只是用普通的左对齐方式，只能显示一点点

|  |
| --- |
|  |
|  |

### 效果

|  |
| --- |
|  |

## 4.此时我们可以把对齐方式修改一下，使用下面的代码

|  |
| --- |
|  |

### 效果，尽管我们设置的矩形还是不够大，但是你可以看到，他的显示效果好多了

|  |
| --- |
|  |

## 5.我们再来显示一个，这一次使用右对齐,矩形宽度不够，我们把他改为550

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 6. 文本居中对齐+文本换行

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 7.上面的输出方式都是没有包括外边距的，我们可以修改一下代码使得它包含外边距，为了方便学习，我们把一下代码注释了

|  |
| --- |
| case WM\_PAINT:  hdc = BeginPaint(hWnd, &ps);  // TODO: 在此添加任意绘图代码...  rc.top = 10;  rc.left = 10;  rc.right =rc.left + 550;  rc.bottom = rc.top + 200;  //绘制矩形  Rectangle(hdc,rc.left,rc.top,rc.right,rc.bottom);  //左对齐+文本换行  DrawText(hdc,szContent,-1,&rc,DT\_LEFT|DT\_WORDBREAK);  ////右对齐+文本换行  //rc.left = 10;  //rc.top = 260;  //rc.right =rc.left + 550;  //rc.bottom = rc.top + 200;  ////绘制矩形  //Rectangle(hdc,rc.left,rc.top,rc.right,rc.bottom);  //DrawText(hdc,szContent,-1,&rc,DT\_RIGHT|DT\_WORDBREAK);  ////文本居中对齐+文本换行  //rc.left = 600;  //rc.top = 10;  //rc.right =rc.left + 550;  //rc.bottom = rc.top + 200;  ////绘制矩形  //Rectangle(hdc,rc.left,rc.top,rc.right,rc.bottom);  //DrawText(hdc,szContent,-1,&rc,DT\_CENTER|DT\_WORDBREAK);  //坐对齐+文本换行+外边距  rc.left = 10;  rc.top = 260;  rc.right =rc.left + 550;  rc.bottom = rc.top + 200;  Rectangle(hdc,rc.left,rc.top,rc.right,rc.bottom);  DrawText(hdc,szContent,-1,&rc,DT\_LEFT|DT\_WORDBREAK|DT\_EXTERNALLEADING);  EndPaint(hWnd, &ps);  break; |

### 效果不太明显

|  |
| --- |
|  |

## 8.仔细一看，我们的程序输出有乱码，我们改用读文件的方法来加载长字符串

|  |
| --- |
|  |
|  |

## 效果，现在就舒服多了

|  |
| --- |
|  |

## 9.这个函数默认对于\t处理也是不好的，如果我们需要把\t处理为空白字符，我们需要使用DT\_EXPANDTABS,为了练习这个功能我们定义一个字符串szMsg\_t里面有很多\t。

|  |
| --- |
|  |

### 然后我们可以把DrawText的字符串改为这个字符串，并且使用上面这个标记

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

### 注意，默认情况下DrawText是会有一个裁剪矩形的，当字符串的长度超过裁剪矩形的宽度时，超过的部分被裁剪，如果你不想它被裁剪可以添加DT\_NOCLIP标记

## 10.我们把第一段文本买的DT\_WORDBREAK标记去掉，换成DT\_NOCLIP

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

### 项目1的文字代码如下

## Lesson37-drawtext-demo.cpp

|  |
| --- |
| // Lesson37-drawtext-demo.cpp : 定义应用程序的入口点。  //  #include "stdafx.h"  #include "Lesson37-drawtext-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\_LESSON37DRAWTEXTDEMO, 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\_LESSON37DRAWTEXTDEMO));  // 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\_LESSON37DRAWTEXTDEMO));  wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);  wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);  wcex.lpszMenuName = MAKEINTRESOURCE(IDC\_LESSON37DRAWTEXTDEMO);  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;  TCHAR szContent[4096];  //用读取文件的方法来加载长字符串，注意，使用unicode编程文本文件的编码必须使用UTF-16LE  DWORD readSize;  ZeroMemory(szContent,4096);  HANDLE hFile;  hFile = CreateFile(\_T("demo.txt"),GENERIC\_READ,0,NULL,OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);  if(INVALID\_HANDLE\_VALUE == hFile)  {  MessageBox(hWnd,\_T("打开文件失败"),\_T("错误"),0);  return -1;  }  ReadFile(hFile,szContent,4096,&readSize,NULL);  CloseHandle(hFile);  RECT rc;  TCHAR szMsg\_t[]=\_T("If\tyou\tsee\ta\tpretty\tgirl\t,showly\tyou\twill\trun\tafter\ther.\tright?\twill\tthis\tis\tquite\tnormal.\twho\tdoesn't\twant\ta\t\pretty\tgirl?");  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...  rc.top = 10;  rc.left = 10;  rc.right =rc.left + 550;  rc.bottom = rc.top + 200;  //绘制矩形  Rectangle(hdc,rc.left,rc.top,rc.right,rc.bottom);  //左对齐+文本换行  //DrawText(hdc,szContent,-1,&rc,DT\_LEFT|DT\_WORDBREAK);  //不换行，不裁剪  DrawText(hdc,szContent,-1,&rc,DT\_LEFT|DT\_NOCLIP);  //坐对齐+文本换行+外边距  rc.left = 10;  rc.top = 260;  rc.right =rc.left + 550;  rc.bottom = rc.top + 200;  Rectangle(hdc,rc.left,rc.top,rc.right,rc.bottom);  //DrawText(hdc,szContent,-1,&rc,DT\_LEFT|DT\_WORDBREAK|DT\_EXTERNALLEADING);  //处理\t  DrawText(hdc,szMsg\_t,-1,&rc,DT\_LEFT|DT\_EXPANDTABS|DT\_WORDBREAK|DT\_EXTERNALLEADING);  ////右对齐+文本换行  //rc.left = 10;  //rc.top = 260;  //rc.right =rc.left + 550;  //rc.bottom = rc.top + 200;  ////绘制矩形  //Rectangle(hdc,rc.left,rc.top,rc.right,rc.bottom);  //DrawText(hdc,szContent,-1,&rc,DT\_RIGHT|DT\_WORDBREAK);  ////文本居中对齐+文本换行  //rc.left = 600;  //rc.top = 10;  //rc.right =rc.left + 550;  //rc.bottom = rc.top + 200;  ////绘制矩形  //Rectangle(hdc,rc.left,rc.top,rc.right,rc.bottom);  //DrawText(hdc,szContent,-1,&rc,DT\_CENTER|DT\_WORDBREAK);    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;  } |

## 项目2.练习段文本的常见对齐方式

## 1.把上面的项目复制一份,改名Lesson37-drawtext-demo2,把多余的内容删除.把文本改为”Hello Cv++”我们先来看左上角对齐,代码如下

|  |
| --- |
|  |
|  |

### 效果:

|  |
| --- |
|  |

## 2. 短文本，单行显示，左下角对齐

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 3.短文本，单行显示，水平左对齐，垂直居中对齐

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 4. 短文本，单行显示，水平居中，垂直顶对齐

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 5. 短文本，单行显示，水平居中，垂直居中对齐

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 6. 短文本，单行显示，水平居中，垂直底对齐

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 7. 短文本，单行显示，水平居右，垂直顶对齐

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 8. 短文本，单行显示，水平居右，垂直居中对齐

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 9. 短文本，单行显示，水平居右边，垂直底对齐

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

### 注意:在实际开发中不需要把这个矩形绘制出来,这个函数默认是利用矩形来限定文本的,我们需要理解这一点

# 项目2的完整代码如下

## Lesson37-drawtext-demo2.cpp

|  |
| --- |
| // Lesson37-drawtext-demo.cpp : 定义应用程序的入口点。  //  #include "stdafx.h"  #include "Lesson37-drawtext-demo2.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\_LESSON37DRAWTEXTDEMO, 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\_LESSON37DRAWTEXTDEMO));  // 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\_LESSON37DRAWTEXTDEMO));  wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);  wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);  wcex.lpszMenuName = MAKEINTRESOURCE(IDC\_LESSON37DRAWTEXTDEMO);  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;  RECT rc;  TCHAR szMsg\_t[]=\_T("Hello CV++");  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...  rc.top = 10;  rc.left = 10;  rc.right =rc.left + 300;  rc.bottom = rc.top + 100;  //绘制矩形  Rectangle(hdc,rc.left,rc.top,rc.right,rc.bottom);  //短文本，单行显示，左上角对齐  DrawText(hdc,szMsg\_t,-1,&rc,DT\_LEFT|DT\_TOP|DT\_SINGLELINE);  //短文本，单行显示，左下角对齐  DrawText(hdc,szMsg\_t,-1,&rc,DT\_LEFT|DT\_BOTTOM|DT\_SINGLELINE);  //短文本，单行显示，水平左对齐，垂直居中对齐  DrawText(hdc,szMsg\_t,-1,&rc,DT\_LEFT|DT\_VCENTER|DT\_SINGLELINE);  //短文本，单行显示，水平居中，垂直顶对齐  DrawText(hdc,szMsg\_t,-1,&rc,DT\_TOP|DT\_CENTER|DT\_SINGLELINE);  //短文本，单行显示，水平居右，垂直居中对齐  DrawText(hdc,szMsg\_t,-1,&rc,DT\_CENTER|DT\_VCENTER|DT\_SINGLELINE);  //短文本，单行显示，水平居中，垂直底对齐  DrawText(hdc,szMsg\_t,-1,&rc,DT\_CENTER|DT\_BOTTOM|DT\_SINGLELINE);  // 短文本，单行显示，水平居右，垂直顶对齐  DrawText(hdc,szMsg\_t,-1,&rc,DT\_RIGHT|DT\_TOP|DT\_SINGLELINE);  // 短文本，单行显示，水平居右，垂直居中对齐  DrawText(hdc,szMsg\_t,-1,&rc,DT\_RIGHT|DT\_VCENTER|DT\_SINGLELINE);  // 短文本，单行显示，水平居右边，垂直底对齐  DrawText(hdc,szMsg\_t,-1,&rc,DT\_RIGHT|DT\_BOTTOM|DT\_SINGLELINE);  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;  } |