# 这一节我们来学习菜单相关操作

## 1.使用vs2019新建一个MFC工程，取名MenuDemo，注意取消勾选一些我们不需要的功能

|  |
| --- |
|  |

## 2.默认MFC帮我们生成了一个默认菜单，而且实现了一些基本的功能，我们如果需要添加新菜单项，可以打开resource文件，点击资源文件夹，双击MenuDemo.rc然后点击菜单文件夹，然后双击IDR\_MAINFRAME菜单，就会出现菜单设计界面

|  |
| --- |
|  |

## 3.先在帮助的旁边添加一个菜单Hello，然后给他添加一个SayHello子项，可以看到属性面板会帮他生成一个ID

|  |
| --- |
|  |

### 注意，默认这些菜单的顶级菜单都是弹出菜单，是不能修改ID的，如果你需要修改ID，可以把弹出属性改为False，但是这样子会删除他的使用子菜单。建议不要随便修改。新版本的MFC的菜单ID不再以IDM\_开头而是直接以ID\_开头。如我们的SayHallo菜单项的ID是ID\_HELLO\_SAYHELLO

## 4.点击保存保存我们的菜单项，然后打开类向导，点击命令选项卡，找到我们的菜单ID，也就是上面的ID，点击右边listBox里面的WM\_COMMAND,然后点击添加处理程序，会弹出一个对话框，如图

|  |
| --- |
|  |

### 点击确定，然后点击应用，如图

|  |
| --- |
|  |

### 可以看到系统帮我们生成了一个菜单命令响应函数，点击右边的编辑代码就可以进入到这个函数里面

|  |
| --- |
|  |

## 5.在里面添加一个弹出消息框的代码，如下

|  |
| --- |
|  |

### 编译运行程序，点击我们的菜单项，效果如下：

|  |
| --- |
|  |

### 注意：菜单其实是属于MainFrame类的，虽然把响应函数写在view类里面也能够工作，不过感觉好像不太对。还是使用MainFrame的好一点。其实MFC程序里面的这几个类都能够添加菜单命令响应函数。那么如果给他们都添加对同一个菜单目录的响应函数，究竟那个类的响应函数先执行？

### 是view类的响应函数最先执行！然后就是文档类的响应函数，第三个就是框架类MainFrame的响应函数。最后就是App类的响应函数。

# 信息其实是有分类的，我们将他们分为3类

|  |
| --- |
|  |

# 消息也有路由

|  |
| --- |
|  |

## MainFrame=》view=》doc，如果doc没有消息处理函数，它又把字符让我重新交给view类，如果view类也没有处理函数，它又会把它传递给MainFrame类，如果MainFrame也没有，那么他就交给CApp类。

# 下面我们来学习标题菜单的编程实现

## 所谓的标题菜单，就是菜单命令前面有一个✔的菜单项

## 6.我们来尝试给文件-新建创建标题菜单，进入CMainFrame菜单类的OnCreate函数

|  |
| --- |
|  |

## 在编写代码之前，我们有必要了解一下菜单的结构

|  |
| --- |
|  |

## 编程思想：先获取这个菜单对象，也就是菜单栏，然后再获取整个文件子菜单，然后利用索引或者ID找到新建菜单项。然后进行编程，实现整个功能需要用到下面的API

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

## CheckMenuItem使用说明：

|  |  |  |
| --- | --- | --- |
| CMenu::CheckMenuItem 向弹出菜单中的菜单项添加复选标记或从其删除复选标记。  复制  UINT CheckMenuItem(  UINT nIDCheckItem,  UINT nCheck); 参数 *nIDCheckItem* 指定要检查的菜单项，由 *nCheck* 确定。  *nCheck* 指定如何检查菜单项以及如何确定该项在菜单中的位置。 *nCheck* 参数可以是 MF\_CHECKED 或 MF\_UNCHECKED 与 MF\_BYPOSITION 或 MF\_BYCOMMAND 标志的组合。 可以使用按位 OR 运算符组合这些标志。 它们具有以下含义：   * MF\_BYCOMMAND 指定该参数提供现有菜单项的命令 ID。 这是默认值。 * MF\_BYPOSITION 指定该参数提供现有菜单项的位置。 第一项位于位置 0。 * MF\_CHECKED 充当使用 MF\_UNCHECKED 的转换，将默认复选标记放置在项旁边。 * MF\_UNCHECKED 充当使用 MF\_CHECKED 的转换，删除项目旁边的复选标记。  返回值 项的先前状态：MF\_CHECKED 或 MF\_UNCHECKED，或 0xFFFFFFFF（如果菜单项不存在）。 注解 *nIDCheckItem* 参数指定要修改的项。  *nIDCheckItem* 参数可标识弹出菜单项以及菜单项。 检查弹出菜单项时无需特殊步骤。 无法选中顶级菜单项。 必须按位置检查弹出菜单项，因为它没有与之关联的菜单项标识符。 示例    |  | | --- | | 示例 1 C++复制  // CMainFrame::OnToggleTestMenuInfo() is a menu command handler for  // "Toggle Info" menu item (whose resource id is ID\_MENU\_TOGGLEINFO). It  // toggles the checked or unchecked state of the "Toggle Info" menu item.  // CMainFrame is a CFrameWnd-derived class.  void CMainFrame::OnToggleTestMenuItemInfo()  {  // Get the popup menu which contains the "Toggle Info" menu item.  CMenu\* mmenu = GetMenu();  CMenu\* submenu = mmenu->GetSubMenu(4);  // Check the state of the "Toggle Info" menu item. Check the menu item  // if it is currently unchecked. Otherwise, uncheck the menu item  // if it is not currently checked.  MENUITEMINFO info;  info.cbSize = sizeof (MENUITEMINFO); // must fill up this field  info.fMask = MIIM\_STATE; // get the state of the menu item  VERIFY(submenu->GetMenuItemInfo(ID\_MENU\_TOGGLEINFO, &info));  if (info.fState & MF\_CHECKED)  submenu->CheckMenuItem(ID\_MENU\_TOGGLEINFO, MF\_UNCHECKED | MF\_BYCOMMAND);  else  submenu->CheckMenuItem(ID\_MENU\_TOGGLEINFO, MF\_CHECKED | MF\_BYCOMMAND);  } |  CMenu::GetMenuState 返回指定菜单项的状态或弹出菜单中的项数。  复制  UINT GetMenuState(  UINT nID,  UINT nFlags) const; 参数 *nID* 指定由 *nFlags* 确定的菜单项 ID。  *nFlags* 指定 *nID* 的性质。 可以为下列值之一：   * MF\_BYCOMMAND 指定该参数提供现有菜单项的命令 ID。 这是默认值。 * MF\_BYPOSITION 指定该参数提供现有菜单项的位置。 第一项位于位置 0。  返回值 如果指定的项不存在，则为值 0xFFFFFFFF。 如果 *nId* 标识弹出菜单，则高位字节包含弹出菜单中的项数，低位字节包含与弹出菜单相关联的菜单标志。 否则，返回值是以下列表中值的掩码（布尔 OR）（此掩码描述 *nId* 标识的菜单项的状态）：   * MF\_CHECKED 充当使用 MF\_UNCHECKED 的转换，将默认复选标记放置在项旁边。 当应用程序提供复选标记位图时（参见 SetMenuItemBitmaps 成员函数），将显示“复选标记打开”位图。 * MF\_DISABLED 禁用菜单项，使其无法被选中，但不将其灰显。 * MF\_ENABLED 启用菜单项，使其可被选中，并消除其灰显状态。 注意，该常数的值为 0；当使用此值时，应用程序不应针对 0 测试失败结果。 * MF\_GRAYED 禁用菜单项，使其无法被选中，同时将其灰显。 * MF\_MENUBARBREAK 将项目放置在静态菜单中的新行上或弹出菜单中的新列中。 将使用垂直分隔线将新弹出菜单列与旧列隔开。 * MF\_MENUBREAK 将项目放置在静态菜单中的新行上或弹出菜单中的新列中。 列之间无分隔线。 * MF\_SEPARATOR 绘制一条水平分割线。 只能在弹出菜单中使用。 此线条不能灰显、禁用或突出显示。 将忽略其他参数。 * MF\_UNCHECKED 充当使用 MF\_CHECKED 的转换，删除项目旁边的复选标记。 当应用程序提供复选标记位图时（参见 SetMenuItemBitmaps 成员函数），将显示“复选标记关闭”位图。 注意，该常数的值为 0；当使用此值时，应用程序不应针对 0 测试失败结果。  示例2  |  | | --- | | // CMainFrame::OnToggleTestMenuState() is a menu command handler for  // "Toggle State" menu item (whose resource id is ID\_MENU\_TOGGLESTATE).  // It toggles the checked or unchecked state of the "Toggle State" menu item.  // CMainFrame is a CFrameWnd-derived class.  void CMainFrame::OnToggleTestMenuState()  {  // Get the popup menu which contains the "Toggle State" menu item.  CMenu \*mmenu = GetMenu();  CMenu \*submenu = mmenu->GetSubMenu(4);  // Check the state of the "Toggle State" menu item. Check the menu item  // if it is currently unchecked. Otherwise, uncheck the menu item  // if it is not currently checked.  UINT state = submenu->GetMenuState(ID\_MENU\_TOGGLESTATE, MF\_BYCOMMAND);  ASSERT(state != 0xFFFFFFFF);  if (state & MF\_CHECKED)  submenu->CheckMenuItem(ID\_MENU\_TOGGLESTATE, MF\_UNCHECKED | MF\_BYCOMMAND);  else  submenu->CheckMenuItem(ID\_MENU\_TOGGLESTATE, MF\_CHECKED | MF\_BYCOMMAND);  } | |

## 7.在CMainFrame::OnCreate函数中添加如下代码

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

### 也可以这么写

|  |
| --- |
|  |

### 注意第二个参数的组合方式，如果你菜单索引，需要使用：MF\_BYPOSITION | MF\_CHECKED如果你使用ID，需要使用：MF\_BYCOMMAND | MF\_CHECKED

# 下面我们来学习如何创建缺省菜单项

## 先把上面的打勾代码注释了

|  |
| --- |
|  |

## 这个功能需要用到一个叫做SetDefaultItem的API

|  |
| --- |
|  |

## 8.这里，我们想把文件-打开作为文件子菜单的缺省菜单，所谓的缺省菜单就是以粗体显示的菜单项

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

## 当然，还可以这么写，这一次我们用新建菜单项来测试

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

### 需要注意的是：

#### 1.子菜单上面的分割线也算是子菜单的一个项，不要数漏了，否则你得不到需要的效果

#### 2.一个子菜单里面只能有一个缺省菜单项，即使你设置了多个，只有最后一个起作用

# 下面我们来学习图形标记菜单

## 需要用到一个API叫做CMenu::SetMenuItemBitmaps

|  |
| --- |
|  |

### 他需要两个bitmap，第一个代表菜单没有选中的图片，第二个表示选中的图片，这些图片其实可以道微软网站，只需要在google搜索中输入

### [Visual Studio image library 就可以进入下载页面](https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjggvyskJOIAxVWSzABHc71JxQQFnoECBoQAQ&url=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Fvisualstudio%2Fdesigners%2Fthe-visual-studio-image-library%3Fview%3Dvs-2022&usg=AOvVaw2nzqrnZ7_8bsNK3bFK8bC5&opi=89978449)

### 需要注意的是，你是在CMainFrame的OnCreate函数里面写代码，二你的菜单需要在别处使用，如果此时你创建的bitmap是句柄变量，他将帮在菜单上面显示，最好把它设置为成员变量

## 9、我们在项目中插入一个新建文件的bmp文件，作为我们的图标资源

|  |
| --- |
|  |

## 10.在CMainFrame类的头文件里面定义一个位图成员变量

|  |
| --- |
|  |

### 然后在MainFrame.cpp的类的OnCreate函数里面加载我们的位图资源，并且设置到菜单项里面

|  |
| --- |
|  |

### 编译运行程序，效果如下

|  |
| --- |
|  |

### 注意：如果你自己在资源编辑器里面新建位图资源，不要把它的大小设置得太大。否则只能显示左上角的一部分，如果你不清楚该用多大的尺寸，可以借助一个叫做GetSystemMetrics的函数来获取

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 语法 C++复制  int GetSystemMetrics(  [in] int nIndex  ); 参数 [in] nIndex  类型： **int**  要检索的系统指标或配置设置。 此参数的取值可为下列值之一： 请注意，所有SM\_CX\* 值为宽度，所有SM\_CY\* 值为高度。 另请注意，设计为返回布尔数据的所有设置将 **TRUE** 表示为任何非零值， **FALSE** 表示零值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **SM\_ARRANGE**  56 | 指定系统如何排列最小化窗口的标志。 有关详细信息，请参阅本主题中的“备注”部分。 | | **SM\_CLEANBOOT**  67 | 指定系统启动方式的 值：   * 0 正常启动 * 1 故障安全启动 * 2 通过网络启动实现故障安全   故障安全启动 (也称为 SafeBoot、安全模式或干净启动) 会绕过用户启动文件。 | | **SM\_CMONITORS**  80 | 桌面上的显示监视器数。 有关详细信息，请参阅本主题中的“备注”部分。 | | **SM\_CMOUSEBUTTONS**  43 | 鼠标上的按钮数;如果未安装鼠标，则为零。 | | **SM\_CONVERTIBLESLATEMODE**  0x2003 | 反映笔记本电脑或平板模式的状态，0 表示板模式，否则为非零。 当此系统指标发生更改时，系统会通过 LPARAM 中带有“ConvertibleSlateMode” [的WM\_SETTINGCHANGE](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/wm-settingchange) 发送广播消息。 请注意，此系统指标不适用于台式电脑。 在这种情况下，请使用 [GetAutoRotationState](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getautorotationstate)。 | | **SM\_CXBORDER**  5 | 窗口边框的宽度（以像素为单位）。 这等效于具有 3D 外观的窗口的 SM\_CXEDGE 值。 | | **SM\_CXCURSOR**  13 | 光标的标称宽度（以像素为单位）。 | | **SM\_CXDLGFRAME**  7 | 此值与 SM\_CXFIXEDFRAME 相同。 | | **SM\_CXDOUBLECLK**  36 | 矩形围绕双击序列中第一次单击的位置的宽度（以像素为单位）。 第二次单击必须在由 SM\_CXDOUBLECLK 和 SM\_CYDOUBLECLK 定义的矩形内发生，系统才能将两次单击视为双击。 两次单击也必须在指定时间内发生。  若要设置双击矩形的宽度，请使用SPI\_SETDOUBLECLKWIDTH调用 [SystemParametersInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-systemparametersinfoa) 。 | | **SM\_CXDRAG**  68 | 鼠标指针在拖动操作开始之前可以移动的鼠标向下点任一侧的像素数。 这允许用户轻松单击并释放鼠标按钮，而不会无意中启动拖动操作。 如果此值为负值，则从鼠标向下点的左侧减去该值，并将其添加到其右侧。 | | **SM\_CXEDGE**  45 | 三维边框的宽度（以像素为单位）。 此指标是SM\_CXBORDER的三维对应指标。 | | **SM\_CXFIXEDFRAME**  7 | 窗口周围具有描述文字但不是相当大的（以像素为单位）的框架的粗细。 SM\_CXFIXEDFRAME是水平边框的高度，SM\_CYFIXEDFRAME是垂直边框的宽度。  此值与 SM\_CXDLGFRAME 相同。 | | **SM\_CXFOCUSBORDER**  83 | [DrawFocusRect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-drawfocusrect) 绘制的焦点矩形的左边缘和右边缘的宽度。 此值以像素为单位。  **Windows 2000：**不支持此值。 | | **SM\_CXFRAME**  32 | 此值与 SM\_CXSIZEFRAME 相同。 | | **SM\_CXFULLSCREEN**  16 | 主显示器上全屏窗口的工作区宽度（以像素为单位）。 若要获取系统任务栏或应用程序桌面工具栏未遮挡的屏幕部分的坐标，请使用SPI\_GETWORKAREA值调用 [SystemParametersInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-systemparametersinfoa) 函数。 | | **SM\_CXHSCROLL**  21 | 水平滚动条上箭头位图的宽度（以像素为单位）。 | | **SM\_CXHTHUMB**  10 | 水平滚动条中拇指框的宽度（以像素为单位）。 | | **SM\_CXICON**  11 | 图标的系统大宽度（以像素为单位）。 [LoadIcon](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-loadicona) 函数只能加载具有SM\_CXICON和SM\_CYICON指定尺寸的图标。 有关详细信息 [，请参阅图标大小](https://learn.microsoft.com/zh-cn/windows/win32/menurc/about-icons#icon-sizes) 。 | | **SM\_CXICONSPACING**  38 | 大图标视图中项的网格单元格的宽度（以像素为单位）。 每个项都适合在排列时按SM\_CYICONSPACING SM\_CXICONSPACING大小的矩形。 此值始终大于或等于 SM\_CXICON。 | | **SM\_CXMAXIMIZED**  61 | 主显示监视器上最大化的顶级窗口的默认宽度（以像素为单位）。 | | **SM\_CXMAXTRACK**  59 | 具有描述文字和大小调整边框（以像素为单位）的窗口的默认最大宽度。 此指标是指整个桌面。 用户无法将窗口框架拖动到大于这些尺寸的大小。 窗口可以通过处理 [WM\_GETMINMAXINFO](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/wm-getminmaxinfo) 消息来替代此值。 | | **SM\_CXMENUCHECK**  71 | 默认菜单的宽度检查标记位图（以像素为单位）。 | | **SM\_CXMENUSIZE**  54 | 菜单栏按钮的宽度，例如在多个文档界面中使用的子窗口关闭按钮（以像素为单位）。 | | **SM\_CXMIN**  28 | 窗口的最小宽度（以像素为单位）。 | | **SM\_CXMINIMIZED**  57 | 最小化窗口的宽度（以像素为单位）。 | | **SM\_CXMINSPACING**  47 | 最小化窗口的网格单元格的宽度（以像素为单位）。 每个最小化窗口在排列时适合此大小的矩形。 此值始终大于或等于 SM\_CXMINIMIZED。 | | **SM\_CXMINTRACK**  34 | 窗口的最小跟踪宽度（以像素为单位）。 用户无法将窗口框架拖动到小于这些尺寸的大小。 窗口可以通过处理 [WM\_GETMINMAXINFO](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/wm-getminmaxinfo) 消息来替代此值。 | | **SM\_CXPADDEDBORDER**  92 | 带字幕窗口的边框填充量（以像素为单位）。  **Windows XP/2000：**不支持此值。 | | **SM\_CXSCREEN**  0 | 主显示器的屏幕宽度（以像素为单位）。 这是通过调用 [GetDeviceCaps](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-getdevicecaps) 获取的相同值，如下所示： GetDeviceCaps( hdcPrimaryMonitor, HORZRES)。 | | **SM\_CXSIZE**  30 | 窗口中按钮的宽度描述文字或标题栏（以像素为单位）。 | | **SM\_CXSIZEFRAME**  32 | 可调整大小的窗口周边的大小边框的粗细（以像素为单位）。 SM\_CXSIZEFRAME是水平边框的宽度，SM\_CYSIZEFRAME是垂直边框的高度。  此值与 SM\_CXFRAME 相同。 | | **SM\_CXSMICON**  49 | 图标的系统小宽度（以像素为单位）。 小图标通常显示在窗口标题和小图标视图中。 有关详细信息 [，请参阅图标大小](https://learn.microsoft.com/zh-cn/windows/win32/menurc/about-icons#icon-sizes) 。 | | **SM\_CXSMSIZE**  52 | 小描述文字按钮的宽度（以像素为单位）。 | | **SM\_CXVIRTUALSCREEN**  78 | [虚拟屏幕](https://learn.microsoft.com/zh-cn/windows/win32/gdi/the-virtual-screen)的宽度（以像素为单位）。 虚拟屏幕是所有显示监视器的边框。 SM\_XVIRTUALSCREEN指标是虚拟屏幕左侧的坐标。 | | **SM\_CXVSCROLL**  2 | 垂直滚动条的宽度（以像素为单位）。 | | **SM\_CYBORDER**  6 | 窗口边框的高度（以像素为单位）。 这等效于具有 3D 外观的窗口的 SM\_CYEDGE 值。 | | **SM\_CYCAPTION**  4 | 描述文字区域的高度（以像素为单位）。 | | **SM\_CYCURSOR**  14 | 光标的标称高度（以像素为单位）。 | | **SM\_CYDLGFRAME**  8 | 此值与 SM\_CYFIXEDFRAME 相同。 | | **SM\_CYDOUBLECLK**  37 | 矩形围绕双击序列中第一次单击的位置的高度（以像素为单位）。 第二次单击必须在由 SM\_CXDOUBLECLK 定义的矩形内发生，SM\_CYDOUBLECLK系统会将两次单击视为双击。 两次单击也必须在指定时间内发生。  若要设置双击矩形的高度，请使用SPI\_SETDOUBLECLKHEIGHT调用 [SystemParametersInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-systemparametersinfoa) 。 | | **SM\_CYDRAG**  69 | 鼠标指针在拖动操作开始之前可以移动的鼠标向下点上方和下方的像素数。 这允许用户轻松单击并释放鼠标按钮，而不会无意中启动拖动操作。 如果此值为负值，则从鼠标向下点上方减去该值，并将其添加到其下方。 | | **SM\_CYEDGE**  46 | 三维边框的高度（以像素为单位）。 这是SM\_CYBORDER的三维对应项。 | | **SM\_CYFIXEDFRAME**  8 | 窗口周围具有描述文字但不是相当大的（以像素为单位）的框架的粗细。 SM\_CXFIXEDFRAME是水平边框的高度，SM\_CYFIXEDFRAME是垂直边框的宽度。  此值与 SM\_CYDLGFRAME 相同。 | | **SM\_CYFOCUSBORDER**  84 | [DrawFocusRect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-drawfocusrect) 绘制的焦点矩形的上边缘和下边缘的高度。 此值以像素为单位。  **Windows 2000：**不支持此值。 | | **SM\_CYFRAME**  33 | 此值与 SM\_CYSIZEFRAME 相同。 | | **SM\_CYFULLSCREEN**  17 | 主显示器上全屏窗口的工作区高度（以像素为单位）。 若要获取系统任务栏或应用程序桌面工具栏未遮挡的屏幕部分的坐标，请使用 SPI\_GETWORKAREA 值调用 [SystemParametersInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-systemparametersinfoa) 函数。 | | **SM\_CYHSCROLL**  3 | 水平滚动条的高度（以像素为单位）。 | | **SM\_CYICON**  12 | 图标的系统高度（以像素为单位）。 [LoadIcon](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-loadicona) 函数只能加载具有SM\_CXICON和SM\_CYICON指定尺寸的图标。 有关详细信息 [，请参阅图标大小](https://learn.microsoft.com/zh-cn/windows/win32/menurc/about-icons#icon-sizes) 。 | | **SM\_CYICONSPACING**  39 | 大图标视图中项的网格单元格的高度（以像素为单位）。 每个项都适合在排列时按SM\_CYICONSPACING SM\_CXICONSPACING大小的矩形。 此值始终大于或等于 SM\_CYICON。 | | **SM\_CYKANJIWINDOW**  18 | 对于系统的双字节字符集版本，这是屏幕底部的汉字窗口的高度（以像素为单位）。 | | **SM\_CYMAXIMIZED**  62 | 主显示监视器上最大化的顶级窗口的默认高度（以像素为单位）。 | | **SM\_CYMAXTRACK**  60 | 具有描述文字和大小调整边框的窗口的默认最大高度（以像素为单位）。 此指标是指整个桌面。 用户无法将窗口框架拖动到大于这些尺寸的大小。 窗口可以通过处理 [WM\_GETMINMAXINFO](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/wm-getminmaxinfo) 消息来替代此值。 | | **SM\_CYMENU**  15 | 单行菜单栏的高度（以像素为单位）。 | | **SM\_CYMENUCHECK**  72 | 默认菜单的高度检查标记位图（以像素为单位）。 | | **SM\_CYMENUSIZE**  55 | 菜单栏按钮（例如在多个文档界面中使用的子窗口关闭按钮）的高度（以像素为单位）。 | | **SM\_CYMIN**  29 | 窗口的最小高度（以像素为单位）。 | | **SM\_CYMINIMIZED**  58 | 最小化窗口的高度（以像素为单位）。 | | **SM\_CYMINSPACING**  48 | 最小化窗口的网格单元格的高度（以像素为单位）。 每个最小化窗口在排列时适合此大小的矩形。 此值始终大于或等于 SM\_CYMINIMIZED。 | | **SM\_CYMINTRACK**  35 | 窗口的最小跟踪高度（以像素为单位）。 用户无法将窗口框架拖动到小于这些尺寸的大小。 窗口可以通过处理 [WM\_GETMINMAXINFO](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/wm-getminmaxinfo) 消息来替代此值。 | | **SM\_CYSCREEN**  1 | 主显示器的屏幕高度（以像素为单位）。 这是通过调用 [GetDeviceCaps](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-getdevicecaps) 获取的相同值，如下所示： GetDeviceCaps( hdcPrimaryMonitor, VERTRES)。 | | **SM\_CYSIZE**  31 | 窗口中按钮的高度描述文字或标题栏（以像素为单位）。 | | **SM\_CYSIZEFRAME**  33 | 可调整大小的窗口周边的大小边框的粗细（以像素为单位）。 SM\_CXSIZEFRAME是水平边框的宽度，SM\_CYSIZEFRAME是垂直边框的高度。  此值与 SM\_CYFRAME 相同。 | | **SM\_CYSMCAPTION**  51 | 小描述文字的高度（以像素为单位）。 | | **SM\_CYSMICON**  50 | 图标的系统小高度（以像素为单位）。 小图标通常显示在窗口标题和小图标视图中。 有关详细信息 [，请参阅图标大小](https://learn.microsoft.com/zh-cn/windows/win32/menurc/about-icons#icon-sizes) 。 | | **SM\_CYSMSIZE**  53 | 小描述文字按钮的高度（以像素为单位）。 | | **SM\_CYVIRTUALSCREEN**  79 | [虚拟屏幕](https://learn.microsoft.com/zh-cn/windows/win32/gdi/the-virtual-screen)的高度（以像素为单位）。 虚拟屏幕是所有显示监视器的边框。 SM\_YVIRTUALSCREEN指标是虚拟屏幕顶部的坐标。 | | **SM\_CYVSCROLL**  20 | 垂直滚动条上箭头位图的高度（以像素为单位）。 | | **SM\_CYVTHUMB**  9 | 垂直滚动条中拇指框的高度（以像素为单位）。 | | **SM\_DBCSENABLED**  42 | 如果 User32.dll 支持 DBCS，则为非零值;否则为 0。 | | **SM\_DEBUG**  22 | 如果安装了 User.exe 的调试版本，则为非零;否则为 0。 | | **SM\_DIGITIZER**  94 | 如果当前操作系统是 Windows 7 或 Windows Server 2008 R2 并且平板电脑输入服务已启动，则为非零;否则为 0。 返回值是一个位掩码，用于指定设备支持的数字化器输入的类型。 有关详细信息，请参阅“备注”。  **Windows Server 2008、Windows Vista 和 Windows XP/2000：**不支持此值。 | | **SM\_IMMENABLED**  82 | 如果启用了输入法管理器/输入法编辑器功能，则为非零值;否则为 0。  SM\_IMMENABLED指示系统是否已准备好在 Unicode 应用程序上使用基于 Unicode 的输入法。 若要确保依赖于语言的 IME 正常工作，检查 SM\_DBCSENABLED 和系统 ANSI 代码页。 否则，ANSI 到 Unicode 的转换可能无法正确执行，或者某些组件（如字体或注册表设置）可能不存在。 | | **SM\_MAXIMUMTOUCHES**  95 | 如果系统中存在数字化器，则为非零;否则为 0。  SM\_MAXIMUMTOUCHES返回系统中每个数字化器支持的最大触点数的聚合最大值。 如果系统只有单点触控数字化器，则返回值为 1。 如果系统具有多点触控数字化器，则返回值是硬件可以提供的同时触点数。  **Windows Server 2008、Windows Vista 和 Windows XP/2000：**不支持此值。 | | **SM\_MEDIACENTER**  87 | 如果当前操作系统为 Windows XP Media Center Edition，则为非零值;否则为 0。 | | **SM\_MENUDROPALIGNMENT**  40 | 如果下拉菜单与相应的菜单栏项右对齐，则为非零;如果菜单左对齐，则为 0。 | | **SM\_MIDEASTENABLED**  74 | 如果为希伯来语和阿拉伯语启用系统，则为非零值;否则为 0。 | | **SM\_MOUSEPRESENT**  19 | 如果安装了鼠标，则为非零;否则为 0。 此值很少为零，因为支持虚拟鼠标，并且某些系统检测到端口的存在，而不是鼠标的存在。 | | **SM\_MOUSEHORIZONTALWHEELPRESENT**  91 | 如果安装了水平滚轮的鼠标，则为非零值;否则为 0。 | | **SM\_MOUSEWHEELPRESENT**  75 | 如果安装了带垂直滚轮的鼠标，则为非零值;否则为 0。 | | **SM\_NETWORK**  63 | 如果存在网络，则设置最小有效位;否则，会将其清除。 其他位保留供将来使用。 | | **SM\_PENWINDOWS**  41 | 如果安装了 Microsoft Windows for Pen 计算扩展，则为非零;否则为零。 | | **SM\_REMOTECONTROL**  0x2001 | 此系统指标在终端服务环境中用于确定当前终端服务器会话是否受到远程控制。 如果当前会话是远程控制的，则其值为非零值;否则为 0。  可以使用终端服务管理工具（如终端服务管理器 (tsadmin.msc) 和 shadow.exe）来控制远程会话。 远程控制会话时，另一个用户可以查看该会话的内容，并可能与之交互。 | | **SM\_REMOTESESSION**  0x1000 | 此系统指标在终端服务环境中使用。 如果调用进程与终端服务客户端会话相关联，则返回值为非零值。 如果调用进程与终端服务控制台会话相关联，则返回值为 0。 **Windows Server 2003 和 Windows XP：**控制台会话不一定是物理主机。 有关详细信息，请参阅 [WTSGetActiveConsoleSessionId](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-wtsgetactiveconsolesessionid)。 | | **SM\_SAMEDISPLAYFORMAT**  81 | 如果所有显示监视器具有相同的颜色格式，则为非零值，否则为 0。 两个显示器可以具有相同的位深度，但颜色格式不同。 例如，红色、绿色和蓝色像素可以使用不同的位数进行编码，或者这些位可以位于像素颜色值的不同位置。 | | **SM\_SECURE**  44 | 应忽略此系统指标;它始终返回 0。 | | **SM\_SERVERR2**  89 | 如果系统为 Windows Server 2003 R2，则为内部版本号;否则为 0。 | | **SM\_SHOWSOUNDS**  70 | 如果用户要求应用程序在仅以有声形式呈现信息的情况下直观呈现信息，则为非零值;否则为 0。 | | **SM\_SHUTTINGDOWN**  0x2000 | 如果当前会话正在关闭，则为非零值;否则为 0。  **Windows 2000：**不支持此值。 | | **SM\_SLOWMACHINE**  73 | 如果计算机具有低端 (慢速) 处理器，则为非零值;否则为 0。 | | **SM\_STARTER**  88 | 如果当前操作系统为 Windows 7 简易版 Edition、Windows Vista 入门版 或 Windows XP Starter Edition，则为非零值;否则为 0。 | | **SM\_SWAPBUTTON**  23 | 如果交换鼠标左右键的含义，则为非零值;否则为 0。 | | **SM\_SYSTEMDOCKED**  0x2004 | 反映停靠模式的状态，0 表示未停靠模式，否则为非零。 当此系统指标发生更改时，系统会通过 LPARAM 中带有“SystemDockMode” [的WM\_SETTINGCHANGE](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/wm-settingchange) 发送广播消息。 | | **SM\_TABLETPC**  86 | 如果当前操作系统为 Windows XP Tablet PC 版本，或者当前操作系统为 Windows Vista 或 Windows 7 且平板电脑输入服务已启动，则为非零值;否则为 0。 SM\_DIGITIZER设置指示运行 Windows 7 或 Windows Server 2008 R2 的设备支持的数字化器输入类型。 有关详细信息，请参阅“备注”。 | | **SM\_XVIRTUALSCREEN**  76 | [虚拟屏幕](https://learn.microsoft.com/zh-cn/windows/win32/gdi/the-virtual-screen)左侧的坐标。 虚拟屏幕是所有显示监视器的边框。 SM\_CXVIRTUALSCREEN指标是虚拟屏幕的宽度。 | | **SM\_YVIRTUALSCREEN**  77 | [虚拟屏幕](https://learn.microsoft.com/zh-cn/windows/win32/gdi/the-virtual-screen)顶部的坐标。 虚拟屏幕是所有显示监视器的边框。 SM\_CYVIRTUALSCREEN指标是虚拟屏幕的高度。 |  返回值 类型： **int**  如果函数成功，则返回值为请求的系统指标或配置设置。  如果函数失败，返回值为 0。 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 不提供扩展错误信息。 注解 系统指标可能因显示而异。  **GetSystemMetrics** (SM\_CMONITORS) 计数仅显示可见的显示器。 这不同于 [EnumDisplayMonitors](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-enumdisplaymonitors)，后者同时枚举可见的显示监视器以及与镜像驱动程序关联的不可见伪监视器。 不可见的伪监视器与用于镜像应用程序绘图以进行远程处理或其他目的的伪设备相关联。  SM\_ARRANGE设置指定系统如何排列最小化窗口，由起始位置和方向组成。 起始位置可以是以下值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | ARW\_BOTTOMLEFT | 从屏幕左下角开始。 默认位置。 | | ARW\_BOTTOMRIGHT | 从屏幕右下角开始。 等效于 ARW\_STARTRIGHT。 | | ARW\_TOPLEFT | 从屏幕左上角开始。 等效于 ARW\_STARTTOP。 | | ARW\_TOPRIGHT | 从屏幕右上角开始。 等效于 ARW\_STARTTOP |SRW\_STARTRIGHT。 |     排列最小化窗口的方向可以是以下值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | ARW\_DOWN | 垂直排列，从上到下。 | | ARW\_HIDE | 通过将最小化的窗口移出屏幕的可见区域来隐藏这些窗口。 | | ARW\_LEFT | 从左到右水平排列。 | | ARW\_RIGHT | 从右到左水平排列。 | | ARW\_UP | 垂直排列，从下到上。 |     SM\_DIGITIZER设置指定在运行 Windows 7 或 Windows Server 2008 R2 的设备上安装的数字化器的类型。 返回值是指定以下一个或多个值的位掩码。  展开表   |  |  | | --- | --- | | **值** | **含义** | | NID\_INTEGRATED\_TOUCH  0x01 | 该设备具有集成的触摸数字化器。 | | NID\_EXTERNAL\_TOUCH  0x02 | 设备具有外部触摸数字化器。 | | NID\_INTEGRATED\_PEN  0x04 | 该设备具有集成的笔数字化器。 | | NID\_EXTERNAL\_PEN  0x08 | 设备具有外部笔数字化器。 | | NID\_MULTI\_INPUT  0x40 | 设备支持多个数字化器输入源。 | | NID\_READY  0x80 | 设备已准备好接收数字化器输入。 |     此 API 不可识别 DPI，如果调用线程按监视器感知 DPI，则不应使用此 API。 有关此 API 的 DPI 感知版本，请参阅 [GetSystemMetricsForDPI](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getsystemmetricsfordpi)。 有关 DPI 感知的详细信息，请参阅 [Windows 高 DPI 文档。](https://learn.microsoft.com/zh-cn/windows/desktop/hidpi/high-dpi-desktop-application-development-on-windows) 示例 以下示例使用 [GetSystemMetrics](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getsystemmetrics) 函数确定是否安装了鼠标以及是否交换了鼠标按钮。 该示例还使用 [SystemParametersInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-systemparametersinfoa) 函数检索鼠标阈值和速度。 它在控制台中显示信息。  syntax复制  #include <windows.h>  #include <stdio.h>  #pragma comment(lib, "user32.lib")  void main()  {  BOOL fResult;  int aMouseInfo[3];    fResult = GetSystemMetrics(SM\_MOUSEPRESENT);    if (fResult == 0)  printf("No mouse installed.\n");  else  {  printf("Mouse installed.\n");  // Determine whether the buttons are swapped.  fResult = GetSystemMetrics(SM\_SWAPBUTTON);    if (fResult == 0)  printf("Buttons not swapped.\n");  else printf("Buttons swapped.\n");    // Get the mouse speed and the threshold values.    fResult = SystemParametersInfo(  SPI\_GETMOUSE, // get mouse information  0, // not used  &aMouseInfo, // holds mouse information  0); // not used  if( fResult )  {  printf("Speed: %d\n", aMouseInfo[2]);  printf("Threshold (x,y): %d,%d\n",  aMouseInfo[0], aMouseInfo[1]);  }  }  } |

### 这个函数功能非常强大，需要慢慢学习，这里我们需要获取的参数是SM\_CXMENUCHECK和SM\_CYMENUCHECK，可以这么窗口菜单位图的大小

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

### 然后你就可以在资源编辑器里面调整你的位图的大小，直到看到状态栏里面的数字合适为止

# 下面我们来学习如何禁用/启用菜单项

## 可以使用CMenu::EnableMenuItem

|  |
| --- |
| CMenu::EnableMenuItem 启用、禁用或灰显菜单项。  复制  UINT EnableMenuItem(  UINT nIDEnableItem,  UINT nEnable); 参数 *nIDEnableItem* 指定要启用的菜单项，由 *nEnable* 确定。 此参数可以指定弹出菜单项以及标准菜单项。  *nEnable* 指定要执行的操作。 它可以是 MF\_DISABLED、MF\_ENABLED 或 MF\_GRAYED 与 MF\_BYCOMMAND 和 MF\_BYPOSITION 的组合。 可以使用 C++ 按位 OR 运算符 (|) 组合这些值。 这些值将具有以下含义：   * MF\_BYCOMMAND 指定该参数提供现有菜单项的命令 ID。 这是默认值。 * MF\_BYPOSITION 指定该参数提供现有菜单项的位置。 第一项位于位置 0。 * MF\_DISABLED 禁用菜单项，使其无法被选中，但不将其灰显。 * MF\_ENABLED 启用菜单项，使其可被选中，并消除其灰显状态。 * MF\_GRAYED 禁用菜单项，使其无法被选中，同时将其灰显。  返回值 以前的状态（MF\_DISABLED、MF\_ENABLED 或 MF\_GRAYED）或 -1（如果无效）。 |

### 实例

|  |
| --- |
| // The code fragment below shows how to disable (and gray out) the  // File\New menu item.  // NOTE: m\_bAutoMenuEnable is set to FALSE in the constructor of  // CMainFrame so no ON\_UPDATE\_COMMAND\_UI or ON\_COMMAND handlers are  // needed, and CMenu::EnableMenuItem() will work as expected.  CMenu \*mmenu = GetMenu();  CMenu \*submenu = mmenu->GetSubMenu(0);  submenu->EnableMenuItem(ID\_FILE\_NEW, MF\_BYCOMMAND | MF\_DISABLED | MF\_GRAYED); |

## 11.我们尝试把打印菜单禁用，因为我们这里不需要打印，我们写下面的代码

|  |
| --- |
|  |

### 这个代码是参考微软的例子写的，应该没有错误，但是运行程序，发现打印功能仍然没有被禁用，这是为什么？

## 12.因为CMainFrame类有一个m\_bAutoMenuEnable他是CMainFrame的父类的成员，他默认会自动启用使用菜单，所以我们需要在CMainFrame的构造函数里面把这个成员设置为FALSE.如图

|  |
| --- |
|  |

### 然后运行程序，效果如下，打印功能被禁用了

|  |
| --- |
|  |

### 这个坑需要避免，还有，当我们把这个变量设置为FALSE后，程序原来默认禁用的菜单项变为可用了，这是因为我们禁用了系统默认的自动处理菜单是否可用的功能。此时我们必须全部自己处理。

## 13.我们把代码还原回来

|  |
| --- |
|  |
|  |

# 下面我们来学习怎么把菜单栏改为我们自己创建的菜单栏

## 这里需要用到一个API，CWnd::SetMenu

|  |
| --- |
|  |

## 14.如果需要移除当前菜单，只需要用SetMenu(NULL)即可

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

## 15.需要注意的是，如果你把原来的菜单移除了，然后用下面的代码把它重新加载，

|  |
| --- |
|  |

### 编译运行程序，表面上看上去没有问题

|  |
| --- |
|  |

### 但是，当我们点击菜单是的时候，就会报异常

|  |
| --- |
|  |

### 这是为什么？因为我们这么写，这个menu是一个局部变量，他的生命周期非常短，当我们需要使用的时候，可能它已经被销毁了，所以就引发异常。解决办法有2个，第一个就是把这个菜单变量设置为类成员变量，第二个办法就是使用CWnd::Detach方法把菜单句柄和CMenu对象分离，这样子当菜单类对象析构的时候，就不会销毁这个菜单句柄。注意需要先把菜单设置给程序，然后再进行句柄分离

## 16.在设置菜单后，添加分离句柄代码

|  |
| --- |
|  |

### 然后编译运行程序，发现没有问题了

|  |
| --- |
|  |
|  |

### 这个坑，也是需要注意的。

### 以后我们学习动态生成菜单时，也需要这么处理。

# MFC的命令更新机制

|  |
| --- |
|  |

# 由于vs2019有工具栏样式的程序对菜单的操作不友好，我们改用vs2008来继续学习，使用的工程是：testmfc

## 17，我们来做一个操作，我的程序默认剪切是不能用的，我们想启用它，在vs2008中，选择CMainFrame类，在属性面板中点击事件选项卡，选中剪切的ID好，点击左边的+，在下拉列表中点击ON\_UPDATE\_COMMAND\_UI，然后点击生成OnUpdateEditCut就会生成这个函数，然后我们就可以在这里编辑代码

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

### 你会发现此时剪刀图标的颜色变深了，说明他现在是可以使用的，因为工具栏通常会和菜单栏对于的ID号是一样的，所以我们可以看出剪切菜单也可用。

## 18.同理，我们也可以把一个可用的菜单变为不可用，如禁用文件-新建菜单

|  |
| --- |
|  |

### 然后它就不能使用了

|  |
| --- |
|  |
|  |

### 也可以判断一下再来禁用，如图

|  |
| --- |
|  |

### 也可以使用索引

|  |
| --- |
|  |

### 效果是一样的

|  |
| --- |
|  |

### 不过使用索引有一个问题，

|  |
| --- |
|  |

### 发现工具栏不再联动了

|  |
| --- |
|  |
|  |

### 我们启用了剪切菜单项，可是工具栏的剪切没有启用，这是为什么？

## 19、我们使用ID来做判断

|  |
| --- |
|  |

### 效果如下

|  |
| --- |
|  |
|  |

### 发现工具栏又联动了，说明菜单项的索引和对应工具栏上的按钮的索引是不一样的。

### 所以，这里建议使用ID号否则就会重新前面的不联动想象

# 下面我们来实现应用程序的右键弹出菜单功能

## 在旧版本的vc++开发工具中，有这样子的菜单项

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

## 20.在vs2008中没有这个功能，只能手动添加，先在资源编辑器里面编辑需要使用的菜单资源，

|  |
| --- |
|  |

## 其实它是一个函数叫做OnContextMenu

|  |
| --- |
|  |

### 先这么写看看有什么效果

|  |
| --- |
|  |

### 效果如下

|  |
| --- |
|  |

### 有一个很明显的问题，就是菜单显示的坐标不太对，它并不是出现在鼠标右键点击的坐标，而是出现在别处，那是因为它默认使用屏幕坐标系，而我们的程序在客户区里面使用客户坐标系，这两个坐标系的坐标不一样！

### 所以响应做一个坐标转换，响应使用CWnd::ClientToScreen，用法如下

|  |
| --- |
|  |

### 我们可用这么写

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

### 另外我们需要用到一个叫做TrackPopupMenu的API函数，用法如下

|  |
| --- |
| CMenu::TrackPopupMenu **BOOL** **TrackPopupMenu(** **UINT** *nFlags***,** **int** *x***,** **int** *y***,** **CWnd\*** *pWnd***,** **LPCRECT** *lpRect* **=** **NULL** **);**  **Return Value**  Nonzero if the function is successful; otherwise 0.  **Parameters**  *nFlags*  Specifies a screen-position flag and a mouse-button flag. The screen-position flag can be one of the following:   * **TPM\_CENTERALIGN**   Centers the pop-up menu horizontally relative to the coordinate specified by *x*. * **TPM\_LEFTALIGN**   Positions the pop-up menu so that its left side is aligned with the coordinate specified by *x*. * **TPM\_RIGHTALIGN**   Positions the pop-up menu so that its right side is aligned with the coordinate specified by *x*.   The mouse-button flag can be either of the following:   * **TPM\_LEFTBUTTON**   Causes the pop-up menu to track the left mouse button. * **TPM\_RIGHTBUTTON**   Causes the pop-up menu to track the right mouse button.   *x*  Specifies the horizontal position in screen coordinates of the pop-up menu. Depending on the value of the *nFlags* parameter, the menu can be left-aligned, right-aligned, or centered relative to this position.  *y*  Specifies the vertical position in screen coordinates of the top of the menu on the screen.  *pWnd*  Identifies the window that owns the pop-up menu. This window receives all **WM\_COMMAND** messages from the menu. In Windows versions 3.1 and later, the window does not receive **WM\_COMMAND** messages until **TrackPopupMenu** returns. In Windows 3.0, the window receives **WM\_COMMAND** messages before **TrackPopupMenu** returns.  *lpRect*  Points to a **RECT** structure or **CRect** object that contains the screen coordinates of a rectangle within which the user can click without dismissing the pop-up menu. If this parameter is **NULL**, the pop-up menu is dismissed if the user clicks outside the pop-up menu. This must be **NULL** for Windows 3.0.  For Windows 3.1 and later, you can use the following constants:   * **TPM\_CENTERALIGN** * **TPM\_LEFTALIGN** * **TPM\_RIGHTALIGN** * **TPM\_RIGHTBUTTON**   **Remarks**  Displays a floating pop-up menu at the specified location and tracks the selection of items on the pop-up menu. A floating pop-up menu can appear anywhere on the screen. |

### 注意，右键菜单的响应函数一定不能够添加到CMainFrame类里面，否则，不会响应即使你给他添加了响应代码都无法执行到！！！因为上面的代码指定了右键菜单的所有者是view类。

### 如果你响应框架类来响应这个右键菜单命令，需要这么写

|  |
| --- |
|  |

## 21.选择需要添加右键菜单的类一般是view类，然后给他响应WM\_CONTEXMENU消息，在里面加载右键菜单，然后进入这个函数里面编辑代码

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

## 22.我们尝试给右键菜单的d一个菜单项添加命令响处理函数，在类视图中选择view类，在属性面板中点击事件选项卡，找到右键菜单项SayHello的ID点击+号，点击COMMAND,会弹出创建响应函数的选项，点击一下，就可以进入对象的函数，我们编辑代码，弹出一个消息框。

|  |
| --- |
|  |

### 编译运行程序，效果如下

|  |
| --- |
|  |
|  |

# 下面我们来学习动态添加菜单项

## 1.关闭上面的工程，然后新建一个MFC工程叫做DynamicMenu选择单文档应用程序

|  |
| --- |
|  |

## 这里使用到一个API，CMenu::CreatePopupMenu，语法如下：

|  |
| --- |
| CMenu::CreatePopupMenu 创建弹出菜单并将其附加到 CMenu 对象。  BOOL CreatePopupMenu(); 返回值 如果弹出菜单成功创建，则为非零；否则为 0。 备注 菜单最初为空。 可以使用 AppendMenu 或 InsertMenu 成员函数添加菜单项。 应用程序可以将弹出菜单添加到现有菜单或弹出菜单。 TrackPopupMenu 成员函数可用于将此菜单显示为浮动弹出菜单，并跟踪弹出菜单上的选择情况。  如果将菜单指定给窗口，则在窗口被销毁时将自动销毁该菜单。 如果将菜单添加到某现有菜单，则当该现有菜单被销毁时，会自动销毁该菜单。  在退出之前，如果未将某个弹出菜单分配给窗口，应用程序需要释放与该菜单关联的系统资源。 应用程序通过调用 [DestroyMenu](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cmenu-class?view=msvc-170#destroymenu) 成员函数释放菜单。 |

## 还有一个，叫做CMenu::AppendMenu，语法如下

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CMenu::AppendMenu 将新项追加到菜单末尾。  BOOL AppendMenu(  UINT nFlags,  UINT\_PTR nIDNewItem = 0,  LPCTSTR lpszNewItem = NULL);  BOOL AppendMenu(  UINT nFlags,  UINT\_PTR nIDNewItem,  const CBitmap\* pBmp); 参数 *nFlags* 指定在将新菜单项添加到菜单时该项的状态的相关信息。 它包含“备注”部分中列出的一个或多个值。  *nIDNewItem* 指定新菜单项的命令 ID，或如果 *nFlags* 设置为 MF\_POPUP，则指定弹出菜单的菜单句柄 (HMENU)。 如果 *nFlags* 设置为 MF\_SEPARATOR，则忽略 *nIDNewItem* 参数（不需要）。  *lpszNewItem* 指定新菜单项的内容。 *nFlags* 参数用于按以下方式解释 *lpszNewItem*：  展开表   | **nFlags** | **解释 lpszNewItem** | | --- | --- | | MF\_OWNERDRAW | 包含应用程序提供的 32 位值，应用程序可使用该值维护与菜单项关联的其他数据。 当应用程序处理 WM\_MEASUREITEM 和 WM\_DRAWITEM 消息时，应用程序可使用此 32 位值。 该值存储在随这些消息一起提供的结构的 itemData 成员中。 | | MF\_STRING | 包含指向以 null 结尾的字符串的指针。 这是默认解释。 | | MF\_SEPARATOR | 将忽略 *lpszNewItem* 参数（不需要）。 |   *pBmp* 指向将用作菜单项的 CBitmap 对象。 返回值 如果该函数成功，则为非 0；否则为 0。 备注 应用程序可以通过在 *nFlags* 中设置值来指定菜单项的状态。 当 *nIDNewItem* 指定一个弹出菜单时，它将成为所附加菜单的一部分。 如果该菜单被销毁，附加菜单也将被销毁。 附加菜单应与 CMenu 对象分离以避免冲突。 请注意，MF\_STRING 和 MF\_OWNERDRAW 对 AppendMenu 的位图版本无效。  以下列表描述了 *nFlags* 中可能设置的标志：   * MF\_CHECKED 充当使用 MF\_UNCHECKED 的转换，将默认复选标记放置在项旁边。 当应用程序提供复选标记位图时（参见 [SetMenuItemBitmaps](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cmenu-class?view=msvc-170#setmenuitembitmaps) 成员函数），将显示“复选标记打开”位图。 * MF\_UNCHECKED 充当使用 MF\_CHECKED 的转换，删除项目旁边的复选标记。 当应用程序提供复选标记位图时（参见 SetMenuItemBitmaps 成员函数），将显示“复选标记关闭”位图。 * MF\_DISABLED 禁用菜单项，使其无法被选中，但不将其灰显。 * MF\_ENABLED 启用菜单项，使其可被选中，并消除其灰显状态。 * MF\_GRAYED 禁用菜单项，使其无法被选中，同时将其灰显。 * MF\_MENUBARBREAK 将项目放置在静态菜单中的新行上或弹出菜单中的新列中。 将使用垂直分隔线将新弹出菜单列与旧列隔开。 * MF\_MENUBREAK 将项目放置在静态菜单中的新行上或弹出菜单中的新列中。 列之间无分隔线。 * MF\_OWNERDRAW 指定该项是所有者绘制项。 第一次显示菜单时，拥有菜单的窗口将收到一条 WM\_MEASUREITEM 消息，该消息检索菜单项的高度和宽度。 每当所有者需要更新菜单项的视觉外观时，都会发送 WM\_DRAWITEM 消息。 此选项对顶级菜单项无效。 * MF\_POPUP 指定菜单项具有与其关联的弹出菜单。 ID 参数指定要与项关联的弹出菜单的句柄。 这用于将顶级弹出菜单或分层弹出菜单添加到弹出菜单项。 * MF\_SEPARATOR 绘制一条水平分割线。 只能在弹出菜单中使用。 此线条不能灰显、禁用或突出显示。 将忽略其他参数。 * MF\_STRING 指定菜单项为字符串。   以下每个组列出了相互排斥、不能一起使用的标志：   * MF\_DISABLED、MF\_ENABLED 和 MF\_GRAYED * MF\_STRING、MF\_OWNERDRAW、MF\_SEPARATOR 和位图版本 * MF\_MENUBARBREAK 和 MF\_MENUBREAK * MF\_CHECKED 和 MF\_UNCHECKED   每当窗口中的菜单发生更改时（无论是否显示该窗口），应用程序都应调用 [CWnd::DrawMenuBar](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cwnd-class?view=msvc-170#drawmenubar)。 |

## 2、进入CMainFrame类的OnCreate函数里面，添加下面的代码

|  |
| --- |
|  |

### 运行程序，没有报错，而且出现了我们动态创建的菜单

|  |
| --- |
|  |

### 但是，当我们点击这个菜单的时候，程序抛异常了

|  |
| --- |
|  |

### 原因和前面的是一样，因为这个menu对象是一个局部变量。当OnCreate执行完成，他就被销毁了，而我们在别处尝试使用一个被销毁的对象，就会引发异常。解决办法就是在添加了菜单后调用menu对象的Detach把菜单句柄和menu对象断开连接。是的menu对象被销毁时不会销毁这个菜单句柄

## 3.我们添加分离句柄的代码，如下

|  |
| --- |
|  |

### 编译运行程序，效果如下

|  |
| --- |
|  |

### 此时点击菜单，就没有问题了

# 现在我们来学习动态插入菜单

## 需要用到一个API，用法如下

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CMenu::InsertMenu 在 *nPosition* 指定的位置插入新菜单项，并向下移动菜单中的其他项。  复制  BOOL InsertMenu(  UINT nPosition,  UINT nFlags,  UINT\_PTR nIDNewItem = 0,  LPCTSTR lpszNewItem = NULL);  BOOL InsertMenu(  UINT nPosition,  UINT nFlags,  UINT\_PTR nIDNewItem,  const CBitmap\* pBmp); 参数 *nPosition* 指定要在其前插入新菜单项的菜单项。 *nFlags* 参数可用于按以下方式解释 *nPosition*：  展开表   | **nFlags** | **解释 nPosition** | | --- | --- | | MF\_BYCOMMAND | 指定该参数提供现有菜单项的命令 ID。 如果既未设置 MF\_BYCOMMAND 也未设置 MF\_BYPOSITION，则这是默认值。 | | MF\_BYPOSITION | 指定该参数提供现有菜单项的位置。 第一项位于位置 0。 如果 *nPosition* 为 -1，则新菜单项将附加到菜单的末尾。 |   *nFlags* 指定如何解释 *nPosition*，并指定新菜单项添加到菜单时的状态信息。 有关可设置的标志列表，请参阅 [AppendMenu](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cmenu-class?view=msvc-170#appendmenu) 成员函数。 要指定多个值，使用按位 OR 运算符将它们与 MF\_BYCOMMAND 或 MF\_BYPOSITION 标志组合。  *nIDNewItem* 指定新菜单项的命令 ID，或如果 *nFlags* 设置为 MF\_POPUP，则指定弹出菜单的菜单句柄 (HMENU)。 如果 *nFlags* 设置为 MF\_SEPARATOR，则忽略 *nIDNewItem* 参数（不需要）。  *lpszNewItem* 指定新菜单项的内容。 *nFlags* 可用于按以下方式解释 *lpszNewItem*：  展开表   | **nFlags** | **解释 lpszNewItem** | | --- | --- | | MF\_OWNERDRAW | 包含应用程序提供的 32 位值，应用程序可使用该值维护与菜单项关联的其他数据。 可在 [WM\_MEASUREITEM](https://learn.microsoft.com/zh-cn/windows/win32/Controls/wm-measureitem) 和 [WM\_DRAWITEM](https://learn.microsoft.com/zh-cn/windows/win32/Controls/wm-drawitem) 消息提供的结构的 itemData 成员中将此 32 位值提供给应用程序使用。 这些消息在菜单项最初显示或更改时发送。 | | MF\_STRING | 包含指向以 null 结尾的字符串的长指针。 这是默认解释。 | | MF\_SEPARATOR | 将忽略 *lpszNewItem* 参数（不需要）。 |   *pBmp* 指向将用作菜单项的 CBitmap 对象。 返回值 如果该函数成功，则为非 0；否则为 0。 备注 应用程序可以通过在 *nFlags* 中设置值来指定菜单项的状态。  每当窗口中的菜单发生更改时（无论是否显示该窗口），应用程序都应调用 CWnd::DrawMenuBar。  当 *nIDNewItem* 指定弹出菜单时，它将成为其插入的菜单的一部分。 如果该菜单被销毁，则插入的菜单也将被销毁。 插入的菜单应与 CMenu 对象分离以避免冲突。  如果激活的多文档界面 (MDI) 子窗口为最大化状态，并且应用程序通过调用此函数并指定 MF\_BYPOSITION 标志将弹出菜单插入 MDI 应用程序的菜单中，则菜单将插入到比预期更靠左的位置。 这是因为活动 MDI 子窗口的控制菜单插入到了 MDI 框架窗口菜单栏的第一个位置。 若要正确定位菜单，应用程序必须将其他情况下使用的位置值增加 1。 应用程序可以使用 WM\_MDIGETACTIVE 消息来决定是否将当前活动的子窗口最大化。 示例 C++复制  // CMainFrame::OnChangeFileMenu() is a menu command handler for  // CMainFrame class, which in turn is a CFrameWnd-derived class.  // It modifies the File menu by inserting, removing and renaming  // some menu items. Other operations include associating a context  // help id and setting default menu item to the File menu.  // CMainFrame is a CFrameWnd-derived class.  void CMainFrame::OnChangeFileMenu()  {  // Get the menu from the application window.  CMenu \*mmenu = GetMenu();  // Look for "File" menu.  int pos = FindMenuItem(mmenu, \_T("&File"));  if (pos == -1)  return;  // Remove "New" menu item from the File menu.  CMenu \*submenu = mmenu->GetSubMenu(pos);  pos = FindMenuItem(submenu, \_T("&New\tCtrl+N"));  if (pos > -1)  submenu->RemoveMenu(pos, MF\_BYPOSITION);  // Look for "Open" menu item from the File menu. Insert a new  // menu item called "Close" right after the "Open" menu item.  // ID\_CLOSEFILE is the command id for the "Close" menu item.  pos = FindMenuItem(submenu, \_T("&Open...\tCtrl+O"));  if (pos > -1)  submenu->InsertMenu(pos + 1, MF\_BYPOSITION, ID\_CLOSEFILE, \_T("&Close"));  // Rename menu item "Exit" to "Exit Application".  pos = FindMenuItem(submenu, \_T("E&xit"));  if (pos > -1)  {  UINT id = submenu->GetMenuItemID(pos);  submenu->ModifyMenu(id, MF\_BYCOMMAND, id, \_T("E&xit Application"));  }  // Associate a context help ID with File menu, if one is not found.  // ID\_FILE\_CONTEXT\_HELPID is the context help ID for the File menu  // that is defined in resource file.  if (submenu->GetMenuContextHelpId() == 0)  submenu->SetMenuContextHelpId(ID\_FILE\_CONTEXT\_HELPID);  // Set "Open" menu item as the default menu item for the File menu,  // if one is not found. So, when a user double-clicks the File  // menu, the system sends a command message to the menu's owner  // window and closes the menu as if the File\Open command item had  // been chosen.  if (submenu->GetDefaultItem(GMDI\_GOINTOPOPUPS, TRUE) == -1)  {  pos = FindMenuItem(submenu, \_T("&Open...\tCtrl+O"));  submenu->SetDefaultItem(pos, TRUE);  }  }  // FindMenuItem() will find a menu item string from the specified  // popup menu and returns its position (0-based) in the specified  // popup menu. It returns -1 if no such menu item string is found.  int FindMenuItem(CMenu \*Menu, LPCTSTR MenuString)  {  ASSERT(Menu);  ASSERT(::IsMenu(Menu->GetSafeHmenu()));  int count = Menu->GetMenuItemCount();  for (int i = 0; i < count; i++)  {  CString str;  if (Menu->GetMenuString(i, str, MF\_BYPOSITION) &&  str.Compare(MenuString) == 0)  return i;  }  return -1;  } |

## 1.我们此时在帮助菜单前面插入一个弹出菜单，创建菜单的代码是一样的，使用我们复用了这些代码

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 2.我们想个我们的子菜单添加菜单项，可用使用AppendMenu来实现。我们把NewMenu改为Tools，然后添加两个菜单项，如图

|  |
| --- |
|  |

### 编译运行程序，效果如下

|  |
| --- |
|  |

### 可见，AppendMenu既可以添加子菜单，也可以添加菜单项

## 3.也可以给系统创建的子菜单添加菜单项，如我们想给帮助菜单增加一个MSDN菜单项，我们可用这么做

|  |
| --- |
|  |

### 效果如下：

|  |
| --- |
|  |

## 4.如果我们想在关于菜单项和MSDN菜单项直接插入菜单项，可用使用InsertMenu成员函数来实现，如我们插入一个“动态帮助“菜单项，可以这么写、

|  |
| --- |
|  |

### 效果如下

|  |
| --- |
|  |

# 下面我们来学习动态删除子菜单

## 1.我们想把视图子菜单删除，可以这么写

|  |
| --- |
|  |

### 编译运行程序，效果如下

|  |
| --- |
|  |

### 视图子菜单就让我们给删除了

### 这个函数既可以删除子菜单，也可以删除菜单项

## 2.我们把我们插入的动态帮助菜单项删除，可以这么写，需要注意的是，如果我们删除了一个子菜单，他右边的子菜单的索引会减少1.比如帮助子菜单原来的索引是4，删除视图子菜单后他的索引就是3.

|  |
| --- |
|  |

### 编译运行程序，发现动态帮助菜单项被删除了

|  |
| --- |
|  |

# 下面我们来学习如何给动态创建的菜单添加目录响应函数

## 技术要点，根据MFC的运行原理，资源的标识可以在Resourse.h里面定义所以需要对动态创建的菜单项做命令响应，你需要在这个文件里面给你的菜单定义一个ID

## 1.我们打开Resource.h,我们给Tools子菜单的两个菜单项定义2个ID

|  |
| --- |
|  |

## 2.点击保存按钮保存文件然后改变Resource.h文件，回到我们的代码中

|  |
| --- |
|  |

## 3.然后需要这个这两个菜单项改为用我们定义的ID来创建

|  |
| --- |
|  |

# 手动设置消息映射

## 4.在CMainFrame的头文件里面定义消息响应函数声明代码

|  |
| --- |
|  |

## 5.然后需要在CMainFrame.cpp文件里面写消息映射宏注意，菜单是命令，所以需要使用ON\_COMMAND宏

|  |
| --- |
|  |

## 6.然后就是添加这两个函数的实现

|  |
| --- |
|  |

### 编译运行程序，效果如下

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

### 响应成功！！！

### 注意需要在3处地方添加代码，头文件的命令处理函数原型声明，cpp文件上面的消息映射宏，还有cpp文件里面的命令处理函数的代码实现

# 演练：利用动态菜单原理开发一个电话本程序，能够用电话号码动态生成菜单

## 1.新建一个项目，前面DynamicPhoneMenu，选择单文档应用程序，完成

|  |
| --- |
|  |

## 2.这个程序实现的功能是：当我们在view窗口里面输入人名+空格+电话号码，会把人名作为菜单项添加到当前菜单中，在输入再添加，然后当我们点击一个人名，会在view窗口客户区里面显示人名+空格+电话号码

## 3首先需要给view类添加一个对WM\_CHAR消息的响应函数，

|  |
| --- |
|  |

### 在里面需要对我们输入的人名进行判断，不要重复添加相同的人名。为了更好的实现这个功能，我们需要给view类添加一个成员变量，原来保存人名

|  |
| --- |
|  |
|  |

### 然后需要在view类的构造函数里面把这个成员变量赋值为-1，注意vs2008会自动初始化，但是我们需要把它改为-1

|  |
| --- |
|  |

## 4.回到OnChar函数，我们来添加代码，这里又需要一个成员变量，我们来添加一个菜单成员变量

|  |
| --- |
|  |

## 5.然后我们来编写创建联系人弹出菜单的代码

|  |
| --- |
|  |

### 编译运行程序，你会发现一个很有趣的现象，当你按下回车，联系人菜单并没有显示，但是当你用鼠标在框架上面滑动或者窗口尺寸变化了，这个联系人菜单才会显示，这是为什么？

### 这个坑需要避免。在View类创建菜单和在MainFrame类创建菜单是不太一样的，在MainFrame类创建菜单会马上生效，因为它设置创建创建的时候创建菜单的，窗口创建完成后对菜单的修改就需要重绘才能生效。在view类创建菜单后需要让他的父窗口也就是框架窗口的菜单栏重新绘制。这里需要用到一个API,叫做DrawMenuBar函数是win32 平台sdk函数

|  |
| --- |
| C++复制  BOOL DrawMenuBar(  [in] HWND hWnd  ); 参数 [in] hWnd  类型：**HWND**  要重绘菜单栏的窗口的句柄。 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。 |

## 6.我们在view类的OnChar函数里面添加菜单后调用一下这个函数，注意需要调用框架对象的窗口句柄成员而不是view类的窗口句柄成员

|  |
| --- |
|  |

### 当然CWnd类也有一个DrawMenuBar函数，所以代码还可以这么写

|  |
| --- |
|  |

### 这又是一个坑

## 7.然后就需要实现添加联系人信息的功能了，也就是要动态添加菜单项，而且这个菜单项需要能够需要用户点击，把对应的信息显示在view窗口里面，要实现这个功能我们需要一个CString类的成员变量来保存我们输入的字符。我们来添加这个成员变量

|  |
| --- |
|  |
|  |

### 我们发现，vs2008又自动帮我们初始化了这个成员变量

|  |
| --- |
|  |

## 8.回到OnChar函数，继续编写代码，注意，由于我们需要输出文本到窗口，我们需要在代码的最开始创建一个dc对象

|  |
| --- |
|  |

## 9.然后我们就可以用它来输出文本，注意这里先把每一个字符添加到字符串成员变量，然后再输出这个成员变量，每按一下按键，都需要输出所有的字符。

|  |
| --- |
|  |

### 注意这里有一个坑，就是把nChar累加到CString对象的时候需要继续类型转换：m\_strLine += (WCHAR)nChar;

### 编译运行一下程序，发现此时我们能够输入文本了，不过暂时没有退格功能。以后我们来完善他

|  |
| --- |
|  |

## 10但是此时并不完善，如果此时我们继续输入联系人，体会在第一个联系人的后面继续输出，我们并不想这样的效果，所以我们需要在按下回车键的处理代码中添加处理代码。而且我们需要把窗口之前的内容抹掉。这里我们先实现抹掉内容和清空需要字符串的功能，可以用背景色来抹掉，也可以用窗口重绘的办法抹掉，因为这里我们只需要文本暂时显示在窗口上。写法1，如下

|  |
| --- |
|  |

### 还有写法2.在这个案例里面这种写法非常简单

|  |
| --- |
|  |

## 11.我们需要实现但用户输入完毕并且按下回车，我们需要保存联系人信息到菜单项，这里需要用你的CString::Find函数和CString::Left函数，先查找空格的索引，然后使用Left函数截取这个索引左边的所以字符作为菜单项添加到联系人子菜单中

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |
|  |

## 12，现在我们需要设置当用户点击一个联系人的名字，就会在窗口中显示这个联系人的信息。我们只使用一个CString成员变量是做不到的，需要另外一个成员变量来保存数据，这个成员变量是MFC提供的CStringArray类的类型。这个集合类的对象的大小可以动态增加也就是可以存储不确定大小的集合，我们可以查看一下msdn

|  |
| --- |
|  |

### 他的用法和CObArray是类似的，只不过他是操作字符串的，类似的集合类还有这些

|  |
| --- |
|  |

### 我们给view类添加一个CStringArray类型的成员变量m\_strArr

|  |
| --- |
|  |
|  |

### 回到OnChar函数，我们在用户按回车的代码里面把联系人信息保存到集合中

|  |
| --- |
|  |

## 13.现在我们需要做一个功能就是但用户点击联系人子菜单的对应的人名，需要在窗口中显示联系人信息。这个有点难度，因为我们不知道有多少给菜单项，我们很难用前面学习过的方法来手动添加命令消息响应函数。为方便操作，我们把我们的插入子菜单改为添加AppendMenu

|  |
| --- |
|  |

### 这里有一个取巧的方法，打开resource视图，点开菜单，随便创建一个子菜单，如图

|  |
| --- |
|  |

### 然后打开Resource.h文件，旧版本的vc响应自己工具ID，vs2008已经帮我们做了

|  |
| --- |
|  |

### 给这些菜单项添加目录响应函数

|  |
| --- |
|  |

### 然后打开资源编辑器，把能够新建的子菜单删除

|  |
| --- |
|  |
|  |

### 我们这么做的目的只不过是用来创建命令响应函数，然后我们继续编写代码，我们先在第一个动态菜单项的命令响应函数里面添加代码

|  |
| --- |
|  |

### 其实，在实际开发中不会写这样子的程序，这不是给自己添麻烦吗？不过这里只是想练习一下如何动态添加菜单并且为菜单项添加命令响应

# 下面我们再学习一个知识点，就是让框架窗口自己处理命令消息

## 我们前面学习到了，如果用户点击菜单，其实是框架类先获取到命令，但是它不自己处理，而是交给他的子类也就是view类处理，那么我们如何能够让框架窗口自己处理命令消息？这里需要从CWnd：：OnCommand函数入手，因为这是一个虚函数，我们可以改写它。

## 14.我们给CMainFrame类添加一个虚函数，vs2008需要现选择CMainFrame类，然后在属性面板中点击重写选项卡，找到OnCommand函数，然后点击一下，右边会出现有小箭头，点击小箭头，然后点击编辑，就可以进入代码

|  |
| --- |
|  |
|  |

## 15，然后我们就可以添加如下代码，因为需要使用View类的成员变量，我们在MainFrame.cpp中包含view类的头文件，结果编译运行报错，解决办法是把view.cpp文件的包含文档类头文件的语句移动到view.h中

|  |
| --- |
|  |

### 一定要小心这个坑

## 16.然后我们来输入下面的代码，先输入MessageBox作为测试代码

|  |
| --- |
|  |

### 编译运行程序，添加运行联系人，然后点击一个联系人的名字，效果如下

|  |
| --- |
|  |
|  |

### 说明的确是截取到了消息给MainFrame执行，但是view类还是能够响应到命令消息，因为他最后还是执行了默认的代码。

## 17.为了让view不再响应命令消息，我们可以在MainFrame处理完消息后，return TRUE

|  |
| --- |
|  |

### 编译运行程序，效果如下

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

### 可以看到View类不再响应命令消息了

## 18.然后我们就可以添加下面的代码，通过MarnFrame类来需要命令消息

|  |
| --- |
|  |

## 运行程序，效果如下

|  |
| --- |
|  |
|  |