# MFC常用类

1. CWinApp类(执行程序)

负责程序的运行流程。该类型的全局对象theApp，可将全局变量声明为该类型的成员变量。如m\_pMainWnd成员变量是主框架窗口对象地址。

2. CFrameWnd类(单文档或多文档界面框架)

3. CDocument类()

4. CView类()

5. CDialog类(对话框界面框架)

CFileDialog类：文件对话框CFileDialog fileDlg(FALSE);//另存为对话框,TRUE打开对话框

fileDlg.m\_ofn.lpstrTitle = "我的文件保存" //标题

fileDlg.m\_ofn.lpstrFilter = "Text Files(\*.txt)\0\*.txt\0All File(\*.\*)\0\*.\*\0\0";用\0或|分隔,2个结束

6. CSplitterWnd类(切分窗口)

不通过向导生成MFC类窗口:

1.#include <afxwin.h>和选择支持MFC类库选项。

2.继承CWinApp父类,再定义CWinApp的子类全局对象。

3.重写CWinApp的InitInstance虚函数。InitInstance函数中new一个CFrameWnd类窗口对象指针，并Create该窗口。

并将该窗口指针赋值给全局对象的m\_pMainWnd成员。

## CWnd

### 风格

### 函数

#### 根据资源ID获取资源指针对象

CWnd\* GetDlgItem(INT) //参数：资源ID，返回值：该资源ID对应的指针。

#### 更改窗体风格

BOOL ModifyStyle(0, 风格) //修改某个控件风格

返回值：修改成功返回1， 否则0。

#### 判断窗口是否显示

BOOL IsWindowVisible();

#### 根据HWND获取CWND

CWnd\* FromHandle(HWND);

#### 安全获取窗口对象的HWND

HWND GetSafeHwnd()

//若使用CWnd->m\_hWnd获取到的HWND， 可能CWnd窗口不存在，则出现错误。

若使用CWnd->GetSafeHwnd(); CWnd窗口对象不存在， 则函数会返回NULL。

#### 获取父窗口对象

CWnd\* GetParent()

#### 更新窗口界面上绑定的非控件类型的控件

BOOL UpdateData(BOOL)

//TRUE从控件获得数据, FALSE设置数据到控件。

#### 滚动窗口

void ScrollWindow(int, int, LPCRECT, LPCRECT);

第一个参数:是X方向, 每次滚动的距离, 正数窗口往右移动

第二个参数:是Y方向, 每次滚动的距离, 正数窗口往下移动

参数三: 默认值NULL

参数四: 默认值NULL

#### 窗体透明

SetLayeredWindowAttributes( RGB(0,0,0),200,LWA\_ALPHA)

//第二个参数：透明度，0~255， 0为全透明。

//第三个参数：LAW\_ALPHA:将无视第一个参数。

PS：需设置WS\_EX\_LAYERED风格才能使窗口透明。SetWindowLong(m\_hWnd,GWL\_EXSTYLE,GetWindowLong(m\_hWnd,GWL\_EXSTYLE) ^ WS\_EX\_LAYERED);

## CDialog

## CListCtrl类(列表控件)

### 风格

LVS\_EX\_TRACKSELECT：当光标移动到某行数据自动高亮。

LVS\_SINGLESEL: 列表行只能单选

### 消息

### 函数

HCURSOR SetHotCursor( HCURSOR hc );

功能:设置CListCtrl光标移动到有数据的行时显示的光标。

参数:光标句柄，::LoadCursor(NULL,MAKEINTRESOURCE(IDC\_ARROW))

1. 删除选定行

while(m\_listCtrl.GetNextItem(-1,LVNI\_ALL | LVNI\_SELECTED) != -1)

{

int nItem = m\_listCtrl.GetNextItem(-1,LVNI\_ALL | LVNI\_SELECTED);

m\_listCtrl.DeleteItem(nItem);

} //如果使用for(int i = 0; i<m\_listStock.GetItemCount(); i++)

{

if(m\_listStock.GetItemState(i, LVNI\_ALL | LVNI\_SELECTED) == LVIS\_SELECTED)

m\_listStock.DeleteItem(i);

}则删除不干净。

#### 表头插入图片

CHeaderCtrl\* head = GetHeaderCtrl(); //获取表头指针

head ->SetImageList(&m\_ImgList); //将表头类设置图片列表(CImageList);

HDITEM hdItem;

hdItem.mask= HDI\_IMAGE | HDI\_FORMAT;

VERIFY(head ->GetItem(0, &hdItem) );

hdItem.iImage= 1; //显示的图片，图片列表(CImageList)的图片位置（从0开始和数组下表一样）;

hdItem.fmt|= HDF\_IMAGE;

VERIFY(head ->SetItem(0, &hdItem) );

PS:表头复选框功能，根据点击表头事件来替换（head ->SetItem）表头不同的复选框图片。 目前未解决表头排序问题，没法区别点击位置是否是在复选框图片上，待以后解决。

### 常用

1. 单元格可编辑。

方法：通过点击某个单元格事件，将编辑控件显示到对应单元格位置。

## CComboBox(组合下拉框)

### 风格

CBS\_DROPDOWNLIST //下拉风格(不可写)

CBS\_DROPDOWN //下拉风格(可写)

### 消息

ON\_CBN\_SELCHANGE

当点击下拉框的选择时(不管是否发生变化选择，只要点击则触发)，则触发该消息响应函数，

在响应函数中，GetWindowText是获取点击之前的文本内容， 应该通过GetLBText(GetCurSel(),strText);获取点击之后的文本内容。

### 函数

void GetLBText( int, CString&)

功能:根据下拉框中的序号获取对应的文本。

## CStatic (静态文本框)

### 风格

SS\_NOTIFY | SS\_ENDELLIPSIS //该属性在静态文本不够长时显示..., 若没有该属性，若最后一个单词不够长显示时会不显示的。

SS\_BITMAP|SS\_CENTERIMAGE //在静态文本中设置bmp格式图片。

### 消息

### 函数

## CButton (按钮)

### 风格

BS\_DEFPUSHBUTTON :普通按钮风格

BS\_AUTORADIOBUTTON | WS\_GROUP : 单选按钮风格(第一个单选按钮设置WS\_GROUP属性)

### 消息

### 函数

## CEdit (编辑框)

### 风格

ES\_MULTILINE : 多行，使用文本”\r\n”换行回车才显示。

### 消息

### 函数

## CRichEdit (拓展编辑框)

### 风格

### 消息

### 函数

## CTabCtrl (标签)

### 风格

### 消息

### 函数

### 禁用tab标签按钮

在点击标签的响应的函数中， 再设置(SetCurSel)回原来的tab标签。

## CImageList(图片列表)

图像列表控件（CImageList）是相同大小图像的一个集合，每个集合中均以0为图像的索引序号基数，（可以把这些图标看成是以数组方式存储的），图像列表通常由大图标(ICON)或位图(BMP)构成。由于图像列表不是窗口，它们本身并不是控件，不能单独使用，但它们和几种不同类型的控件一起使用包括列表控件（CListCtrl）、树控件（CTreeCtrl）和选项卡控件（CTabCtrl）。

### 风格

### 消息

### 函数

#### 根据资源ID直接创建图片列表

BOOL Create(UINT, int, int, COLORREF)

第一个参数: 位图资源ID。

第二个参数: 图像宽度像素。

第三个参数: 图像列表可控制的图像数量。

第四个参数: RGB(255,255,255) 颜色屏蔽位



//以上BMP图片像素为48\*16，背景色为0xFF00FF；

//创建列表参数为：图像宽度为16，图像数量为3，颜色屏蔽位：RGB(0xFF,0x00,0xFF)。

#### 根据不同的资源ID创建图片列表

BOOL Create(int, int, UINT, int, int)

第一个参数: 列表中每个图像宽度像素。

第二个参数: 列表中每个图像高度像素。

第三个参数: ILC\_MASK|ILC\_COLOR32

第四个参数: 图像列表包含的最初的图像数量

第五个参数: 图像列表最大的图像数量。

//Create(32, 32, ILC\_COLOR32, 0, 4);表示创建了初始图像数量为0，最大图像数量为4的图像列表(可以看成数组)，此时改列表无图片数据， 需要调用其Add成员函数加入图片数据。

#### 往图片列表添加图片数据

重载一：INT Add(ICON) //Add(AfxGetApp()->LoadIcon(IDI\_ICON1))

重载二：INT Add(CBitmap\*, RGB) //参数二：颜色屏蔽位。

//CBitmap bmp;

//bmp.LoadBitmap(IDB\_BITMAP1)

//Add(&bmp, RGB(0, 0, 0));

//bmp.DeleteObject();

//bmp.LoadBitmap(IDB\_BITMAP2)

//Add(&bmp, RGB(0, 0, 0));

从文件导入：

HBITMAP hBitmap;

CBitmap \*pBitmap;

CString FilePathName = "e:/TIME.bmp";

pBitmap = new CBitmap;

HBitmap=(HBITMAP)LoadImage(AfxGetInstanceHandle(),FilePathName, //从文件导入

IMAGE\_BITMAP,0,0,LR\_LOADFROMFILE);

pBitmap->Attach(hBitmap);

Add(pBitmap, RGB(0,0,0));

## CFont

(1)  int GetLogFont(LOGFONT\*) 用来填充LOGFONT结构

LOGFONT lf;

GetLogFont(&lf);

lf.lfUnderline = 1  //设置字体下划线。

(2)  BOOL CreateFontIndirect (const LOGFONT\*) 初始化字体

LOGFONT lf;

GetLogFont(&lf);

CFont \*font = new CFont;

font->CreateFontIndirect(&lf);

设置某够控件字体SetFont(font);

font->DeleteObject(); //字体不用后需要删除。

1. CreateFont 创建字体

## CDC

利用MFC的CDC类绘图: GetDC和ReleaseDC是CWnd类成员函数

CDC \*pDC = GetDC();

pDC->MoveTo(CPoint);

pDC->LineTo(CPoint);

ReleaseDC(pDC);

利用MFC的CDC派生CClientDC类绘图:该类构造函数会调用GetDC,获得设备描述表对象，析构会调用ReleaseDC。

CClientDC dc(this);//在this中绘图

dc.MoveTo(CPoint);

dc.LineTo(CPoint);

(1) CSize GetTextExtent(字符串， 字符串大小) 获取某个字符串的大小，cx为字符串宽度，cy为字符串高度。

     应用:

  在窗口类中使用GetDC()获取CDC\*指针, 然后使用CDC\*指针获取m\_hDC的HDC句柄。 (HDC = GetDC(m\_hWnd))

## CBitmap

1.根据位图ID获取位图大小

CBitmap bitmap;

bitmap.LoadBitmap(位图ID);

方法一：

BITMAPINFO temp;

bitmap.GetObject(sizeof(temp), &temp);

int bwidth = temp.bmiHeader.biWidth;

int bHeight = temp.bmiHeader.biHeight;

方法二：

BITMAP bInfo;

Bitmap.GetBitmap(&bInfo);

int bWidth = bInfo.bmWidth;

int bHeight = bInfo.bmHeight;

## CBrush

介绍:

用画刷填充区域,因为设备描述表默认是白色画刷，当填充一个区域时，该区域颜色覆盖白色。所以并不需要把画刷选入设备描述表

应用:

CBrush brush(RGB(255,0,0));

CClientDC dc(this);

dc.FillRect(CRect, &brush); //填充区域

---------------------------------------------------------------------------

加载位图画刷

CBitmap bitmap;

bitmap.LoadBitmap(IDB\_BITMAP);

CBrush brush(&bitmap);

CClientDC dc(this);

dc.FillRect(CRect, &brush); //填充区域,此函数不需要将画刷选入设备描述表

---------------------------------------------------------------------------

创建透明画刷

CBrush \*pBrush = CBrush::FromHandle((HBRUSH)GetStockObject(NULL\_BRUSH));

CClientDC dc(this);

CBrush \*pOldBrush = dc.SelectObject(pBrush);

dc.Rectangle(CRect);   //需要将画刷选入设备描述表

dc.SelectObject(pOldBrush);

## CPen

介绍:

默认是黑线画笔。

应用:

 将画笔(GDI)对象选入设备描述表

CPen pen(PS\_SOLID, 1, RGB(255,0,0)); //宽度小于1，虚线才有效

CClientDC dc(this);

CPen\* pOldPen = dc.SelectObject(&pen);

dc.MoveTo(CPoint1);

dc.LineTo(CPoint2);

dc.SelectObject(pOldPen); //还原画笔

## CString

成员函数:

Trim(); //移除字符串前后空格。

Trim(char[]); //移除字符串中指定字符串的前后空格。

## CFile

## CArchive

序列化: 将CArchive对象从内存中的数据放到永久性存储地方(如文件)。 并无追加功能，只能覆盖。

反序列化:从文件中读取数据到CArchive对象。 若读取数据出错这抛出...异常，可用来判断是否文件中读完或错误。

序列化与反序列化数据位置要一致，不然读取数据会错乱。

当构造一个CArchive对象时，要把它附加给表示一个打开文件的类CFile（或派生类）的对象上。还要指定归档文件将用于装载还是存储。

功能：

(in)参数一()：

(in)参数二()：

(in)参数三()：

(in)参数四()：

(in)参数五()：

返回值(BOOL)：

## 单文档

CFrameWnd - 框架窗口类，负责管理主框架窗口。

CWinApp - 应用程序类，负责整个程序执行流程。

CDocument - 文档类，负责管理数据（提取/转换/存储）。

CView - 视图窗口类，负责管理视图窗口以及数据的显示。

最简单的代码框架:

#include <afxwin.h>

1 在框架窗口的WM\_CREATE消息的处理中创建视图窗口

2 CMyView类必须要重写CView类成员纯虚函数OnDraw(CEditView等可以不用OnDraw)

3 必须指明第一个视图窗口的ID（AFX\_IDW\_PANE\_FIRST）

class CMyFrameWnd : public CFrameWnd

{

DECLARE\_MESSAGE\_MAP(); //可以为私有

};

BEGIN\_MESSAGE\_MAP(CMyFrameWnd, CFrameWnd)

ON\_WM\_CREATE()

END\_MESSAGE\_MAP()

class CMyWinApp : public CWinApp

{

public:

//CMyWinApp( );

virtual BOOL InitInstance( );

};

class CMyView : public CView

{

public:

virtual void OnDraw(CDC\* pDC);

}

void CMyView::OnDraw(CDC\* pDC)

{

pDC->TextOut( 0, 0, "CMyView::OnDraw" );

}

int CMyFrameWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)

{

CMyView \*pView = new CMyView( );

pView->Create( NULL, "", WS\_CHILD|WS\_VISIBLE, //创建视图

CRect(0,0,200,200), this,

AFX\_IDW\_PANE\_FIRST );

this->m\_pViewActive = pView;

return 0;

}

CMyWinApp theApp;//全局对象

BOOL CMyWinApp::InitInstance( )

{

CMyFrameWnd \*pFrame = new CMyFrameWnd( ); //调用构造时， 可以直接赋值给m\_pMainWnd指针

pFrame->Create( NULL, "MFCBase" ); //Create方法可以在CMyFrameWnd()构造函数中调用

m\_pMainWnd = pFrame;

pFrame->ShowWindow( SW\_SHOW );

pFrame->UpdateWindow( );

return TRUE;

}

之二:

#include <afxwin.h>

class CMyFrameWnd : public CFrameWnd

{

};

class CMyWinApp : public CWinApp

{

public:

//CMyWinApp( );

virtual BOOL InitInstance( );

};

CMyWinApp theApp;//全局对象

BOOL CMyWinApp::InitInstance( )

{

CMyFrameWnd \*pFrame = new CMyFrameWnd( ); //调用构造时， 可以直接赋值给m\_pMainWnd指针

pFrame->Create( NULL, "MFCBase" ); //Create方法可以在CMyFrameWnd()构造函数中调用

m\_pMainWnd = pFrame;

pFrame->ShowWindow( SW\_SHOW );

pFrame->UpdateWindow( );

return TRUE;

}

* + - * 1. 窗口的切分

动态切分 - 程序在执行过程中，根据用户需要实时完成切分。

静态切分 - 窗口在创建出来时，就已经完成切分。

CSplitterWnd - 不规则框架类。

静态切分:

定义一个CSplitterWnd类对象

重写CFrameWnd成员虚函数OnCreateClient,注意返回TRUE

添加动态创建声明和实现

利用CSplitterWnd::CreateStatic创建不规则框架窗口

利用CSplitterWnd::CreateView添加视图窗口。

split1.CreateStatic( this, 2, 1 );

split2.CreateStatic( &split1, 1, 2, WS\_CHILD|WS\_VISIBLE,split1.IdFromRowCol(0,0) );

split2.CreateView( 0, 0, RUNTIME\_CLASS(CMyView), CSize(200,200), pContext );

split2.CreateView( 0, 1, RUNTIME\_CLASS(CEditView),CSize(200,200), pContext );

split1.CreateView( 1, 0, RUNTIME\_CLASS(CHtmlView),CSize(200,200), pContext );

CHtmlView \*pView = (CHtmlView\*)split1.GetPane( 1, 0 ); //获取所在位置的视图

pView->Navigate( "C:/" );

split1.SetRowInfo( 0, 200, 100); //第二个参数设置分割窗口的高度

动态切分:

定义一个CSplitterWnd类对象

重写CFrameWnd成员虚函数OnCreateClient

添加动态创建声明和实现

利用CSplitterWnd::Create创建切分， 最多只能2\*2

显示是同类型视图窗口。

CCreateContext cxt;

cxt.m\_pNewViewClass = RUNTIME\_CLASS(CMyView);

split1.Create( this, 2, 2, CSize(100,100), &cxt );

* + - * 1. 工具栏

添加工具栏资源

需要添加#include <afxext.h>

工具栏1:

CToolBar toolbar

toolbar.CreateEx( this, TBSTYLE\_TRANSPARENT, WS\_CHILD|

WS\_VISIBLE|CBRS\_ALIGN\_TOP|CBRS\_GRIPPER|

CBRS\_SIZE\_DYNAMIC|CBRS\_TOOLTIPS|CBRS\_FLYBY ); //创建工具栏

toolbar.LoadToolBar( IDR\_TOOLBAR1 ); //加载工具栏

工具栏的停靠和移动

toolbar.EnableDocking( CBRS\_ALIGN\_ANY );//工具栏准备停靠的位置

this->EnableDocking( CBRS\_ALIGN\_ANY ); //this指的是框架对象,框架窗口允许停靠的位置

this->DockControlBar( &toolbar, AFX\_IDW\_DOCKBAR\_TOP );//确定工具栏的停靠位置

CFrameWnd::ShowControlBar( &toolbar, FALSE, FALSE );

toolbar::IsWindowVisible - 判断窗口是否处于显示状态

* + - * 1. 状态栏

创建状态栏

CStatusbar statusbar

statusbar.CreateEx( this );

UINT g\_hIndicator[] ={

0,

IDS\_TIME, //String Table资源中添加

};

statusbar.SetIndicators( g\_hIndicator, sizeof(g\_hIndicator)/sizeof(UINT) );设置指示器

statusbar.SetPaneInfo( 1, IDS\_TIME, SBPS\_POPOUT, 200 ); 设置指示器的风格及宽度

statusbar.SetPaneText(1, "hello");设置指示器的文本内容

CStatusbar::GetItemRect(int nIndex, LPRECT lpRect)//获取窗格的大小

## 多文档

CMDIChildWnd - 子框架窗口类，负责管理子框架窗口。

MDIFrameWnd - 主框架窗口类，负责管理主框架窗口。

WinApp - 应用程序类，负责整个程序执行流程。

Document - 文档类，负责管理数据（提取/转换/存储）。

View - 视图窗口类，负责管理视图窗口以及数据的显示。

## 对话框

CWinApp - 应用程序类，负责整个程序执行流程。

CDialog - 对话框窗口类，负责管理对话框窗口。

Domodal创建的对话框， 如果想在其他线程中销毁该对话框，使SendMessage(WM\_CLOSE)。 而EndDialog函数就算使用GetSafeHwnd函数判断句柄存在都会调用失败，也不能使用OnCancel()关闭。

1. 模态对话框(阻塞)

DoModal()来创建

EndDialog函数用于终止模态对话框，使非模态不可见

Domodal调用之处类似于调用了消息循环，阻塞了当前函数，消息循环处理估计屏蔽了非自身的大部分消息使得模态窗口的父类窗口不可点击等操作，但未屏蔽除了父类定时器消息，若有定时器在跑此时消息循环接收定时消息后就会去执行定时器函数，因此异步了。 但是都在一个线程里。

1. 非模态对话框(非阻塞)

Create(IDD\_DIALOG1, NULL)

ShowWindow(SW\_SHOW)

DestroyWindow函数用于非模态对话框

1. 文件对话框

CFileDialog fileDlg(FALSE); //另存为对话框,TRUE打开对话框

fileDlg.m\_ofn.lpstrTitle = "我的文件保存" //标题

fileDlg.m\_ofn.lpstrFilter = "Text Files(\*.txt)\0\*.txt\0All File(\*.\*)\0\*.\*\0\0";用\0或|分隔,2个结束

fileDlg.m\_ofn.lpstrDefExt = "txt";//默认拓展名

fileDlg.GetFileName();//获取文件名

fileDlg.GetPathName();//获取全路径(路径+文件名)

if(IDOK == fileDlg.DoModal())

{//创建文件};

//允许选择多个文件

CFileDialog filedlg(TRUE,NULL,NULL,OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT |OFN\_ALLOWMULTISELECT,\_T("Source 　Files(\*.cpp;\*.c;\*.cc;\*.cxx)|\*.cpp;\*.c;\*.cc;\*.cxx|Head Files(\*.h;\*.hpp;\*.hxx)|\*.h;\*.hpp;\*.hxx||"));

TCHAR \*pBuffer = new TCHAR[MAX\_PATH \* 20];//最多允许同时打开20个文件

filedlg.m\_ofn.lpstrFile = pBuffer;

filedlg.m\_ofn.nMaxFile = MAX\_PATH \* 20;

filedlg.m\_ofn.lpstrFile[0] = '\0';

if (filedlg.DoModal() == IDOK)

{

CString cstrfilepath = \_T("");

POSITION pos = filedlg.GetStartPosition();

while (pos != NULL)

{

cstrfilepath = filedlg.GetNextPathName(pos);//取得文件路径

　　}

}

## 原理部分

### MFC六大机制

#### 消息映射机制

MFC消息机制解决消息处理函数的动态约束问题（如果为每个消息处理函数设计成虚函数将消耗大量内存）

1. 各类消息:

常用的宏:LOWORD、HIWORD、MAKELONG

低16位数据 = LOWORD(32位数据)

32位数据 = MAKELONG(低16位，高16位)

1 windows的标准消息（键盘/鼠标.....）

ON\_WM\_XXX

2 自定义消息

定义消息宏#define WM\_MYMESSAGE WM\_USER+n

消息映射: ON\_MESSAGE(WM\_MYMESSAGE, OnMessagePro)

发送消息:SendMeesage(WM\_MYMESSAGE,WPARAM,LPARAM);

响应函数:LRESULT OnMessagePro(WPARAM wParam, LPARAM lParam)

3 命令消息 //当用户操作某个控件时发出的消息

ON\_COMMAND( 命令ID， 成员处理函数)

ON\_COMMAND\_RANGE(起始ID,终止ID,成员处理函数)

4 通知消息 //控件自身发生变化时（包含用户操作改变）发送消息

ON\_通知码

发送消息:

NMHDR hmh;

hmh.code = TCN\_SELCHANGE; //消息ID

hmh.idFrom = IDC\_TAB1; //控件ID

hmh.hwndFrom = \*GetDlgItem(IDC\_TAB1); //控件的句柄

SendMessage(WM\_NOTIFY, (WPARAM)GetDlgItem(IDC\_TAB1), (LPARAM)&hmh); //WPARAM可不用

1. 消息响应函数中调用其父类的消息响应函数， 是为了将消息传递下去， 若不传递，相当于屏蔽该消息。

原理:

消息映射机制使用:

1.1 类必须派生自 CCmdTarget

1.2 类内添加声明宏 DECLARE\_MESSAGE\_MAP( )

1.3 类外添加实现宏

BEGIN\_MESSAGE\_MAP( theClass,baseClass )

ON\_MESSAGE( WM\_CREATE, OnCreate )

END\_MESSAGE\_MAP( )

执行过程:

1 利用 本类的对象（pFrame）调用成员虚函数

GetMessageMap获取本类静态变量messageMap的

地址 pMessageMap（链表头节点地址）

2 利用pMessageMap的第二个成员获取到相应类的静

态数组首地址，然后在数组中 查找 消息ID对应处理

函数如果找到了 执行4， 如果未找到执行3

3 利用pMessageMap的第一个成员获取父类的静态变

量地址如果为NULL，结束查找，如果不为NULL执行2 //查找顺序:子->父，查找每个类就是查找每个节点

4 利用找到的数组元素，并调用这个数组元素的第六个

数组中存放: struct AFX\_MSGMAP\_ENTRY

{

UINT nMessage; // 消息ID

UINT nCode; // 通知码

UINT nID; // 命令ID/控件ID

UINT nLastID; // 最后一个控件ID

UINT nSig; // 消息处理函数的类型

AFX\_PMSG pfn; // 消息处理函数指针

};

链表节点:

struct AFX\_MSGMAP

{

const AFX\_MSGMAP\* pBaseMap;//每个类的静态变量地址

const AFX\_MSGMAP\_ENTRY\* lpEntries;//数组的首地址

};

AfxWndProc(...)

{

CWnd\* pWnd = CWnd::FromHandlePermanent(hWnd);

//获取和窗口句柄 绑定在一起的框架类对象地址（pFrame）

AfxCallWndProc(pWnd....）//参数pWnd===pFrame

{

pWnd->WindowProc(...)//函数内部this为pFrame==pWnd

{

OnWndMsg(...)//函数内部this为pFrame==pWnd

{

AFX\_MSGMAP\* pMessageMap = GetMessageMap()

{

return &CMyFrameWnd::messageMap;

//返回链表头节点地址(子类节点地址)

}

for (; pMessageMap != NULL;

pMessageMap = pMessageMap->pBaseMap)

//遍历 链表

{

lpEntry = AfxFindMessageEntry(

pMessageMap->lpEntries,message...）；

if( lpEntry != NULL )

{

goto LDispatch;//一旦找到相应的消息处理函数并没执行跳转出for循环

}

}

LDispatch:

union MessageMapFunctions mmf;

mmf.pfn = lpEntry->pfn;

//将OnCreate成员函数地址 赋给联合体的第一个成员

int nSig = lpEntry->nSig; //AfxSig\_lwl

switch (nSig)

{

case AfxSig\_lwl:

(this->\*mmf.pfn\_lwl)(wParam, lParam);

break;

}

}

}

}

}

#### 动态创建机制

动态创建机制的作用:

在不知道类名的情况下，将类的对象创建出来。

2 动态创建机制的使用

2.1 类必须派生自CObject

2.2 类内添加声明宏 DECLARE\_DYNCREATE( theClass )

2.3 类外添加实现宏

IMPLEMENT\_DYNCREATE( theClass，baseClass )

利用CRuntimeClass::CreateObject将类的对象动态创建出来

3 实现（和运行时类信息的区别）

3.1 静态变量 （classCDog）的第四个成员不再为NULL,保存了

一个 新增加的静态函数的地址（CDog::CreateObject）

3.2 函数的变化，多了一个静态函数

CObject\* PASCAL CDog::CreateObject()

{

return new CDog;

}

4 作用

CreateObject - 静态函数

new了一个CDog类的对象，并返回对象地址。

classCDog - 静态变量

第四个成员保存 CreateObject 这个静态函数地址。

5 动态创建对象的执行过程

5.1 利用CDog::classCDog调用CRuntimeClass::CreateObject

5.2 在函数内部获取 CDog::classCDog的第四个成员

CDog::CreateObject并调用

5.3 在函数内部 new了一个CDog类的对象，并返回对象地址。

CObject \*pObject = RUNTIME\_CLASS(CDog)->CreateObject( )

//函数内部this指针为 &CDog::classCDog

{

pObject = (\*m\_pfnCreateObject)()

//调用静态变量的第四个成员保存的

//新增加的 静态函数（CDog::CreateObject）

{

return new CDog; //返回new出来的CDog类的对象地址

}

return pObject;

}

#### 启动机制

第一步：构造theApp全局对象，执行CWinApp::CWinApp()

AFX\_MODULE\_STATE aaa;//当前程序模块状态信息

AFX\_MODULE\_THREAD\_STATE bbb;//当前程序线程状态信息

CWinApp::CWinApp(..)

{

AFX\_MODULE\_STATE\* pModuleState = AfxGetModuleState();

//获取全局变量 &aaa

AFX\_MODULE\_THREAD\_STATE\* pThreadState = pModuleState->m\_thread;

//获取全局变量 &bbb

pThreadState->m\_pCurrentWinThread = this;

//将 &theApp保存到 全局变量bbb的一个成员中

AfxGetThread()

{

AFX\_MODULE\_THREAD\_STATE\* pState = AfxGetModuleThreadState();

//重新获取全局变量 &bbb

CWinThread\* pThread = pState->m\_pCurrentWinThread;

//从bbb的成员 重新要回 &theApp

return pThread; //&theApp

}

pModuleState->m\_pCurrentWinApp = this;

//将 &theApp 保存到全局变量 aaa的一个成员中

AfxGetApp()

{

return AfxGetModuleState()->m\_pCurrentWinApp;

//返回保存在aaa中的 &theApp

}

}

第二步: 进入AfxWinMain全局函数

WinMain(....)

{

AfxWinMain(...)

{

CWinThread\* pThread = AfxGetThread();

CWinApp\* pApp = AfxGetApp();

//以上两句 获取 &theApp

AfxWinInit(...)//初始化MFC库

pApp->InitApplication();//初始化工作（成员虚函数）

pThread->InitInstance()//(应用程序类的成员虚函数)

{

//回到自己代码(创建、显示窗口)Create函数中

{

//加载菜单资源

//调用PreCreateWindow 设计并注册窗口类

WNDCLASS wndcls;

wndcls.lpfnWndProc =DefWindowProc; //会勾取WM\_CREATE消息在钩子处理函数中更改窗口处理函数

}

}

pThread->Run()//函数内部this指针为&theApp(成员虚函数)

{

CWinThread::Run()//函数内部this指针为&theApp

{

for(;;)

{

while( !PeekMessage()函数如果没消息 )

{

OnIdle(..);//（应用程序类的成员虚函数）空闲处理

}

do

{

PumpMessage()

{

如果::GetMessage抓到WM\_QUIT返回FALSE; //这里是if(GetMessage)

翻译/派发消息

}

if( PumpMessage返回FALSE )

ExitInstance();

//(应用程序类的成员虚函数)善后处理

}while(PeekMessage()函数)

}

}

}

}

}

#### 序列化(永久保存)机制

把一个对象的信息存储到硬盘上,读取数据和写入数据必须一致。

//序列化写文件

CFile file("1.txt", CFile::modeCreate | CFile::modeWrite);

//构造存档对象

CArchive ar(&file, CArchive::store);

int i = 4;

char ch = 'a';

float f = 1.3f;

CString str("123456");

//保存数据

ar << i << ch << f << str;

//序列化读文件

CFile file("1.txt", CFile::modeRead);

//构造存档对象

CArchive ar(&file, CArchive::load);

int i;

char ch;

float f;

CString str;

//读取数据

ar >> i >> ch >> f >> str;

#### 运行时类信息机制

运行时类信息机制的使用

1. 类必须从COjbect类派生

2. 类内添加声明宏 DECLARE\_DYNAMIC( theClass )

3. 类外添加实现宏 IMPLEMENT\_DYNAMIC(theClass，baseClass )

利用CObject::IsKindOf来判断类的对象 是否属于该类以及父类(父类根据实现宏确定)。

struct CRuntimeClass

{

LPCSTR m\_lpszClassName;//类名称

int m\_nObjectSize; //类的大小（sizeof）

UINT m\_wSchema; //类的版本（0xFFFF）

CObject\* (PASCAL\* m\_pfnCreateObject)();

//用于动态创建机制，运行时类信息机制为NULL

CRuntimeClass\* m\_pBaseClass;

//父类的静态变量地址（连接链表）

CRuntimeClass\* m\_pNextClass;

//用于序列化机制，运行时类信息机制为NULL

};

实现宏：(都是CRuntimeClass类型)定义子类静态变量， 获取子类静态变量地址

### MFC常见问题

#### 对话框作为子对话框显示到父对话框

需要设置的属性: style:child,border:none

#### DestroyWindow与PostNcDestroy区别

窗口销毁调用函数顺序: DestroyWindow->PostNcDestroy

#### PreTranslateMessage虚函数

该函数只接受PostMessage的消息，不接收SendMessage消息

if (WM\_MOUSEWHEEL == pMsg->message)

{return TRUE; //TRUE不响应该消息， FLASE响应}

#### CButton类 (按钮)

1.动态创建: CButton \*m\_wndBtn = new CButton

m\_wndBtn->Create(名称,风格, 位置(相对父窗口位置), 父窗口(this),控件ID);

2.风格:

BS\_DEFPUSHBUTTON | WS\_VISIBLE | WS\_CHILD :普通按钮风格

BS\_AUTORADIOBUTTON | WS\_GROUP : 单选按钮风格(第一个单选按钮设置WS\_GROUP属性)

BS\_GROUPBOX: 组静态框

BS\_BITMAP: 支持位图

3.动态创建事件:

(1)ON\_BN\_CLICKED(控件ID,响应函数); //单击事件

4.特有函数:

5:应用:

(1)画背景色:

重载DrawItem虚函数， 并时控件设置BS\_OWNERDRAW风格

(2)绘制单选按钮背景色:

使用WM\_CTRLCOLOR消息:在响应函数中使用

if (nCtlColor == CTLCOLOR\_STATIC)

{

pDC->SetBkMode(TRANSPARENT);

return CreateSolidBrush(RGB(0,255,0));

}

#### CEdit类 (编辑框)

1.风格:

2. 动态创建事件:

EN\_CHANGE, 输入一个字符显示后发生。 WM\_COMMAD HIWORD(wParam)

3.特有函数:

(1) SetSel(0, -1)

选定编辑框内容

(2) BOOL GetModify()

编辑框内容若改变返回1，否则0。

(3)void SetLimitText(UINT)

设置编辑框字符最大个数。

参数:字符最大个数

4.通过向导和某个控件绑定值的范围， 然后调用UpdateData(TRUE)，若数据不对则会出现提示。

#### CStatic类 (静态文本框)

1.风格:

SS\_NOTIFY | SS\_ENDELLIPSIS //该属性在静态文本不够长时显示..., 若没有该属性，若最后一个单词不够长显示时会不显示的。

2. 动态创建事件:

3.特有函数:

#### CComboBox类 (组合框)

注意: CRect设定控件宽度包括下拉列表宽度

1.风格:

CBS\_DROPDOWNLIST //下拉风格(不可写)

CBS\_DROPDOWN //下拉风格(可写)

2. 动态创建事件:

3. 特有函数:

(1)int GetCurSel() 获取当前选中的位置, 从0开始计算。

(2)int SetCurSel(int nSelect) 设置nSelect的位置, 从0开始计算。

如果 nSelect 为 -1 ，则清除当前的选择并返回 CB\_ERR 。

如果 nSelect 大于列表中项的个数，不产生效果，返回 CB\_ERR。

成功返回下标。

(3) int AddString(const char\*) 组合框末尾添加一个字符串

(4) void ResetContent() 清空组合框列表数据。

(5)int DeleteString(UINT uiSelect) 清空组合框列表指定位置数据, 从0开始计算。

(6)int FindStringExact(int, const char\*)

第一个参数为-1表示从头开始查找

第二个参数:表示要查找的字符串

返回值: 找到该字符串所在列表的位置,从0开始计算。

#### CToolTipCtrl类 (气泡提示)

1.风格：

2.动态创建事件:

3.特有函数:

4.应用 :

在构造函数中加入(设置)

EnableToolTips(TRUE);

m\_ToolTip.Create(this,TTS\_ALWAYSTIP);

m\_ToolTip.Activate(TRUE);

m\_ToolTip.AddTool(this,"");//添加 tips

m\_ToolTip.SetMaxTipWidth(1000); //换行

m\_ToolTip.SetDelayTime(TTDT\_INITIAL, 1);

m\_ToolTip.SetDelayTime(TTDT\_AUTOPOP, 500); //提示持续时间

m\_ToolTip.SetDelayTime(TTDT\_RESHOW, 500); //多久时间出现提示。

然后再PreTranslateMessage中加入(设置):

if (m\_pTip)

{m\_pTip->RelayEvent( pMsg );}

最后在OnCreate或OnInitDialog中加入(使用)

m\_ToolTip(GetDlgItem(IDC\_), "123");

#### CDateTimeCtrl类(日期控件)

1.风格：

DTS\_RIGHTALIGN | DTS\_UPDOWN | WS\_TABSTOP 该风格与“MM-dd HH:mm”格式搭配

2.动态创建事件:

3.特有函数:

(1) BOOL SetFormat(const char\*) 设置日期格式

格式有: "yyyy-MM-dd HH:mm:ss"

“MM-dd HH:mm”

//获取格式没有的时间， 则是当前时间。 例如“MM-dd HH:mm”,获取的年就是2014年。

(2) BOOL SetTime(LPSYSTEMTIME) 设置UTC时间(系统时间)

例子一: time\_t转换SYSTEMTIME

time\_t uiTotalSecond = 3600;

FILETIME ft;

SYSTEMTIME pst;

LONGLONG nLL = Int32x32To64(uiTotalSecond, 10000000) + 116444736000000000;//后面数字表示是UTC时间

ft.dwLowDateTime = (DWORD) nLL;

ft.dwHighDateTime = (DWORD) (nLL >> 32);

FileTimeToSystemTime(&ft, &pst);

例子二: SYSTEMTIME转换time\_t

将SYSTEMTIME 赋值给 struct tm结构， 然后再通过mktime函数转换

(3) BOOL SetTime(CTime\*) 设置当地时间(包含时区)

time\_t tSecond = 3600;

m\_Date->SetTime(tSencond); //显示是:1970年1月1号1小时+时区

SetTime(time(NULL));//设置控件为当前时间(其实不设置，创建后默认当前时间

(4) BOOL GetTime(LPSYSTEMTIME) 获取当前控件上面显示的时间,

控件显示多少时间，SYSTEMTIME结构就被填充多少。

(5) BOOL GetTime(CTime) 获取控件时间的秒数。从1970年1月1号

如果小于1970年1月1号， 获得秒数小于0，会崩溃。

（6） BOOL SetRang(CTime\*, CTime\*) 设定时间控件的范围

CTime tMinRang(2000,1,1,0,0,0);

CTime tMaxRang(2069,12,31,23,59,59);

m\_dateAlarmStart.SetRange(&tMinRang, &tMaxRang);

#### CListCtrl类(列表控件)

* + - 1. 风格：

LVS\_REPORT | LVS\_ALIGNLEFT | WS\_BORDER | WS\_TABSTOP

拓展风格:

LVS\_EX\_FULLROWSELECT

LVS\_EX\_ONECLICKACTIVATE

LVS\_EX\_GRIDLINES

LVS\_EX\_SUBITEMIMAGES

2.动态创建事件:

HDN\_ITEMCLICK , LVN\_COLUMNCLICK : 点击某项列表头.

在该消息中排序

if (pNMLV->iSubItem == m\_list.GetCurrentCol())

{

//降序¨°

m\_list.m\_bSortASC = !m\_list.m\_bSortASC;

}

else

{

//升序¨°

m\_list.m\_bSortASC = true;

m\_list.SetCurrentCol(pNMLV->iSubItem);

}

m\_list.SortItems(MyCompareProc, (DWORD)&m\_list);

NM\_KILLFOCUS : 离开焦点和时间控件是同一个通知消息码

3.特有函数:

(1) DWORD SetExtendedStyle(DWORD) 设置拓展风格

(2) int InsertColumn(int, const char\*, int, int, int) 插入一列

第一个参数: 列的位置, 从0开始计算。

第二个参数: 列的名称。

第三个参数: LVCFMT\_LEFT、LVCFMT\_RIGHT、LVCFMT\_CENTER 文字位置

第四个参数: 列的宽度。

第五个参数:属于谁的子列， 都不属于填-1

1. BOOL InserItem(int, const char\*, int) 插入一行的第一列
2. BOOL SetItemText(int, int, const char\*) 在某行中插入一项文本内容

使用该函数前， 必须使用InserItem插入某一行的一列

参数一:行号

参数二:列号

参数三:显示文本。

(5)CString GetItemText(int, int) 获取某项文本。

(3) BOOL SetItem(int, int, UINT, LPCTSTR, int,UINT,UINT, LPARAM ); 插入一项数据

第一个参数: 行号， 从0开始计算

第二个参数: 列号， 从0开始计算

第三个参数: LVCFMT\_LEFT、LVCFMT\_RIGHT、LVCFMT\_CENTER 文字位置

第四个参数:数据

第五个参数:该项的图片列表序号， 无图片为0

其他参数为NULL

1. BOOL DeleteAllItems(); 清空列表,不能清除列名称。
2. DeleteColumn(int nCol); 清除一列，包括列名称。
3. GetHeaderCtrl()->GetItemCount(); 获取列数,

清除所有列，利用循环每次调用DeleteColumn（0）

(8) BOOL SetColumnWidth(int, int); 设置某列宽度

第一个参数:列号，从0开始计算

第二个参数:宽度

1. BOOL SortItem(回调函数, 参数);

回调函数: int CALLBACK MyCompareProc(LPARAM, LPARAM, LPARAM)

回调参数: 参数一和参数二，是<某行调用SetItemData函数>设置的数据值。

回调返回值: :-1，第一列排在第二列前面, 1第一列排在第二列后面。

参数: 设置该参数， 可以从回调函数第三个参数得到。

#### CTreeCtrl类(树控件)

1.风格：

TVS\_HASLINES : 父子节点间绘制连线

TVS\_LINESATROOT: 表示树控制在根项之间存在连线；

TVS\_HASBUTTONS: 表示树控制视在父项左侧存在展开合拢控制按钮；

2.动态创建事件:

(1)ON\_NOTIFY(TVN\_SELCHANGED,控件ID, &CDlgTestDlg::OnSelchangedTree2) //节点选中后的通知消息。

响应函数:返回值void,参数NMHDR\* pNMHDR, LRESULT\* pResult。

SelectItem函数会触发该通知消息。

3.特有函数:

(1) HTREEITEM InsertItem(const char\*, int, int, HTREEITEM, HTREEITEM) 插入一个节点

第一个参数: 插入节点的文本

第二个参数: 未选中该节点显示的图片序号(CImageList)

第三个参数: 选中该节点显示的图片序号(CImageList)

第四个参数: 父节点， 默认值是根节点(TVI\_ROOT)

第五个参数: 插入节点的位置，默认值:兄弟节点的最下面(TVI\_LAST),

TVI\_SORT：排序 TVI\_FIRST:放在兄弟节点最上面

返回值;节点

(2) BOOL Expand(HTREEITEM, int) 让某个节点展开

第二个参数: TVE\_EXPAND 展开

(3) BOOL DeleteItem(HTREEITEM) 删除某个节点

(4) BOOL DeleteAllItems() 删除所有节点

(5) CString GetItemText(HTREEITEM) 获取某个节点的文本

(6)SHORT SetItemHeight(SHORT) 设置每个叶子节点的间隔。

(7) HTREEITEM GetSelectedItem() 获取选中的某个节点

(8) BOOL SetItemData(HTREEITEM, DWORD) 设置某个节点附加数据

(9)DWORD GetItemData(HTREEITEM ) 获取某个节点的附加数据

如果之前没设置， 则返回值0

(10) BOOL SelectItem(HTREEITEM) 设置选中当前节点， 成功非0。

(11)BOOL ItemHasChildren(HTREEITEM)判断某个节点是否有孩子节点，成功非0

(12) HTREEITEM GetChildItem(HTREEITEM) 返回值是某个节点的第一个孩子节点。

(13) HTREEITEM GetNextItem (HTREEITEM, UINT) 获得某个节点的兄弟节点

第二个参数: TVGN\_NEXT 下一个兄弟节点

(14) HTREEITEM GetRootItem()获取根节点

#### CScrollbar类(滚动条)

1.风格：

2.动态创建事件:

3.特有函数:

(1)void SetScrollRange(int, int, BOOL) 设置滚动条的范围

第一个参数: 小0

第二个参数: 大100,若为0，滚动条不显示。

第三个参数:默认值TRUE,是否重画

(2) int GetScrollPos( )获取滚动条当前位置。

(3) int GetScrollLimit()获取滚动条(最大范围值+1)。

例如：设置SetScrollRange(0,100),则返回值为101。

1. int SetScrollPos( int nPos, BOOL bRedraw = TRUE );设置滚动条的位置

(5) void SetScrollInfo(LPSCROLLINFO); //通过使用该函数能是固定大小对话框中动态创建无数控件。

参数: 结构中必须填cbSize = sizeof(SCROLLINFO),fMask = SIF\_ALL;

页大小设置(滑块大小设置),不足一页大小就不显示滚动条。

如果滚动条控件长度和窗口大小一样(也就是一页的大小):滚动条范围和窗口大小比例一致,以及ScrollWindow滚动的大小比例一致。

#### CProgressCtrl类(进度条)

1.风格：

2.动态创建事件:

3.特有函数:

(1) int SetPos(int) 设置当前进度(%)

#### CTabCtrl类(标签控件)

1.风格：

2.动态创建事件:

(1)Tab页切换的通知消息TCN\_SELCHANGE, 只有用户切换时才发生。

3.特有函数:

(1)LONG InsertItem(int, const char\*) 插入一个Tab标签

参数一:位置，从0开始

参数二:名称

(2)BOOL DeleteItem(int) 删除指定标签

参数一 ：位置,从0 开始

(3)BOOL DeleteAllItems() 删除所有标签

(4)int GetCurSel() //获取当前选择标签位置，从0开始

(5)BOOL SetCurSel(int) 设置当前标签位置，从0开始

4.使用例子:

例子一: 获取与strFileName相同的标签文本

TCITEM tcItem;

char buffer[256] = {0};

int i = 0;

for (i = 0; i < m\_tab.GetItemCount(); i++)

{

memset(buffer, 0, 256);

tcItem.pszText = buffer;

tcItem.cchTextMax = 256;

tcItem.mask = TCIF\_TEXT;

m\_tab.GetItem(i, &tcItem);

if (!strcmp(tcItem.pszText,strFileName) )

{

return;

}

}

#### CIPAddressCtrl (地址控件类)

1.风格：

2.动态创建事件:

3.特有函数:

(1) SetAddress(BYTE,BYTE,BYTE,BYTE) 设置界面上控件值(从左到右)， 参数也可:DWORD

### 各种技巧

#### 屏蔽按键F4+Ctrl 和回车和空格

在PreTanslateMessage虚函数中使用：

if ((pMsg->message==WM\_KEYDOWN && pMsg->wParam==VK\_SPACE)

|| (pMsg->message==WM\_KEYDOWN && pMsg->wParam==VK\_RETURN)

|| (WM\_SYSKEYDOWN == pMsg->message) && (VK\_F4 == pMsg->wParam) && (GetKeyState(VK\_MENU) & 0x8000))

{

return TRUE; //TRUE表示不往后传递消息

}

#### 遍历对话框所有控件

//好处:当对话框中的控件变化时，代码不做改变

char clsName[MAX\_PATH] = {0};

HWND hWnd = ::GetWindow(m\_hWnd, GW\_CHILD);

while (hWnd != NULL)

{

if (::GetClassName(hWnd, clsName, MAX\_PATH) != 0)

{

if (strcmp(clsName, "Edit") == 0)

{::SetWindowText(hWnd, "ee");}

if (strcmp(clsName, "ComboBox") == 0)

{::SetWindowText(hWnd, "cc");}

}

hWnd = ::GetNextWindow(hWnd, GW\_HWNDNEXT);

}

#### 修改可执行程序图标

图标资源值修改为：127

#### 修改应用程序大小图标

HICON m\_hIcon

m\_hIcon = LoadIcon(AfxGetInstanceHandle(), MAKEINTRESOURCE(IDI\_ICON1));

SetIcon(m\_hIcon, TRUE); // Set big icon

SetIcon(m\_hIcon, FALSE); // Set small icon

#### 给直线增加气泡提示

 一旦设定宽度，\r\n和空格就会同时起作用，只是空格是在一行宽度超过设定宽度时起作用的

/\*给一条直线增加提示\*/

EnableToolTips(TRUE);

m\_ToolTip.Create(this,TTS\_ALWAYSTIP);

m\_ToolTip.Activate(TRUE);

m\_ToolTip.AddTool(this,"");//添加 tips

m\_ToolTip.SetMaxTipWidth(1000); //换行

m\_ToolTip.SetDelayTime(TTDT\_INITIAL, 1);

m\_ToolTip.SetDelayTime(TTDT\_AUTOPOP, 500);

m\_ToolTip.SetDelayTime(TTDT\_RESHOW, 500);

//要显示Tooltip控件

BOOL CBatteryTestReportDlg::PreTranslateMessage(MSG\* pMsg)

{

// TODO: Add your specialized code here and/or call the base class

m\_ToolTip.RelayEvent( pMsg );

return CDialog::PreTranslateMessage(pMsg);

}

//鼠标移动消息

void CBatteryTestReportDlg::OnMouseMove(UINT nFlags, CPoint point)

{

// TODO: Add your message handler code here and/or call default

TRACKMOUSEEVENT csTME;

csTME.cbSize=sizeof(csTME);

csTME.dwFlags=TME\_LEAVE|TME\_HOVER;

csTME.hwndTrack=m\_hWnd;

csTME.dwHoverTime=100;

::\_TrackMouseEvent(&csTME); //发送消息响应OnMouseHover函数

CDialog::OnMouseMove(nFlags, point);

}

void CBatteryTestReportDlg::OnMouseHover(UINT nFlags, CPoint point)

{

//根据坐标点显示不同内容

ClientToScreen(&point);

m\_ToolTip.UpdateTipText(strData,FromHandle(m\_hWnd));

}

//获取一直线所有的坐标点

void CBatteryTestReportDlg::GetLineAllPoint(CPoint point1, CPoint point2, int nIndex)

{

ClientToScreen(&point1);

ClientToScreen(&point2);

double k = 0; //斜率

double y = 0; //Y坐标

CPoint tempPoint;

vector<CPoint> vcPoint;

//保证第二个参数的x坐标一定比第一个大

if (point1.x > point2.x)

{

tempPoint = point1;

point1 = point2;

point2 = tempPoint;

}

vcPoint.push\_back(point1);

for (int i = point1.x + 1; i < point2.x + 1; i++)

{

if (point1.y < point2.y)

{

k = ((double)(point2.y - point1.y)) / (point2.x - point1.x);

y = k \* (i - point1.x) + point1.y;

}

else

{

k = ((double)(point1.y - point2.y)) / (point2.x - point1.x);

y = -k \* (i - point1.x) + point1.y;

}

tempPoint.x = i;

tempPoint.y = (int)y;

m\_mapPointData[nIndex].push\_back(tempPoint);

}

}

#### 多个控件消息响应同一个函数

1.添加消息映射宏

ON\_COMMAND\_EX(控件ID, OnInfoManage)

ON\_COMMAND\_EX(控件ID, OnInfoManage)

ON\_COMMAND\_EX(控件ID, OnInfoManage)

2.添加处理函数

BOOL 类::OnInfoManage(UINT nID)

{

switch(nID)

{

case IDC\_BUTTON1:

AfxMessageBox("a");

break;

case IDC\_BUTTON2:

AfxMessageBox("b");

break;

case IDC\_BUTTON3:

AfxMessageBox("c");

break;

}

return TRUE;

}

#### 右下角弹出对话框

#define WIN\_WIDTH 100 //弹出窗口的宽度

#define WIN\_HEIGHT 200//弹出窗口的高度

//设置定时器

SetTimer(ID\_TIMER\_POP\_WINDOW,20,NULL);

void CMyDlg::OnTimer(UINT\_PTR nIDEvent)

{

static int nHeight=0;

RECT rect;

SystemParametersInfo(SPI\_GETWORKAREA,0,&rect,0); //获取屏幕的大小

int y=rect.bottom-rect.top;

int x=rect.right-rect.left;

x=x-WIN\_WIDTH;

switch(nIDEvent)

{

case ID\_TIMER\_POP\_WINDOW:

if(nHeight<=WIN\_HEIGHT)

{

++nHeight;

MoveWindow(x,y-nHeight,WIN\_WIDTH,nHeight);

Invalidate(FALSE);//不用重绘

}

else

{

KillTimer(ID\_TIMER\_POP\_WINDOW);

SetTimer(ID\_TIMER\_DISPALY\_DELAY,5000,NULL);

}

break;

case ID\_TIMER\_CLOSE\_WINDOW:

if(nHeight>=0)

{

nHeight--;

MoveWindow(x,y-nHeight,WIN\_WIDTH,nHeight);

}

else

{

KillTimer(ID\_TIMER\_CLOSE\_WINDOW);

SendMessage(WM\_CLOSE);

}

break;

case ID\_TIMER\_DISPALY\_DELAY:

KillTimer(ID\_TIMER\_DISPALY\_DELAY);

SetTimer(ID\_TIMER\_CLOSE\_WINDOW,20,NULL);

break;

}

}

#### EXCEL操作

* + - * 1. 写入Excel

myexcel::\_Application m\_TempExlApp; //定义进程实例

myexcel::Workbooks m\_TempBooks; //定义books实例

myexcel::\_Workbook m\_TempBook; //定义book实例

myexcel::Worksheets m\_TempSheets; //定义Sheets实例

myexcel::\_Worksheet m\_TempSheet; //定义sheet实例

myexcel::Range m\_TempExlRge; //定义Range实例

BOOL OpenExcel(CString szFilePath);//打开excel和写入数据要在同一个线程中。

BOOL CloseExcel();//关闭excel

void SaveExcel(CString szReportPath);

BOOL ExportStatistical();

//写入时，该参数为""

BOOL CStatisticalPageDlg::OpenExcel(CString szFilePath)

{

try

{

//创建Excel服务器(启动Excel)

OleInitialize(NULL);

if (!m\_TempExlApp.CreateDispatch("Excel.Application"))

{

return FALSE;

}

//加载excel

m\_TempBooks.AttachDispatch(m\_TempExlApp.GetWorkbooks(), TRUE);

m\_TempBook.AttachDispatch(m\_TempBooks.Add((\_variant\_t)szFilePath), TRUE);

m\_TempSheets.AttachDispatch(m\_TempBook.GetSheets(), TRUE); //加载Sheet页面

}

catch (...)

{

return FALSE;

}

return TRUE;

}

BOOL CStatisticalPageDlg::CloseExcel()

{

try

{

m\_TempExlRge.ReleaseDispatch();

m\_TempSheet.ReleaseDispatch();

m\_TempSheets.ReleaseDispatch();

m\_TempBook.ReleaseDispatch();

m\_TempBooks.ReleaseDispatch();

m\_TempExlApp.SetDisplayAlerts(FALSE);

m\_TempExlApp.Quit();

m\_TempExlApp.SetDisplayAlerts(TRUE);

m\_TempExlApp.ReleaseDispatch();

OleUninitialize();

}

catch (...)

{

return FALSE;

}

return TRUE;

}

void CStatisticalPageDlg::SaveExcel(CString szReportPath)

{

\_variant\_t OptinVal((long)DISP\_E\_PARAMNOTFOUND, VT\_ERROR);

CString szVersion = m\_TempExlApp.GetVersion();

if (szVersion == "9.0" || szVersion == "10.0" ||szVersion == "11.0" ) //EXCEL2000,2003,2002

{

m\_TempBook.SaveAs(COleVariant(szReportPath), OptinVal, OptinVal, OptinVal, OptinVal, OptinVal, 1, OptinVal,

OptinVal, OptinVal, OptinVal, OptinVal);

}

else //EXCEL 2007

{

m\_TempBook.SaveAs(COleVariant(szReportPath), \_variant\_t(56), OptinVal, OptinVal, OptinVal, OptinVal, 1, OptinVal,

OptinVal, OptinVal, OptinVal, OptinVal);

}

m\_TempBook.Close(OptinVal, OptinVal, OptinVal);

return;

}

BOOL CStatisticalPageDlg::ExportStatistical()

{

int iRowIndex = 1;

int iColumnIndex = 1;

CString szText;

//打开EXCEL

if (FALSE == OpenExcel(""))

{

return FALSE;

}

//将第一行信息写到文件中

try

{

m\_TempSheet.AttachDispatch(m\_TempSheets.GetItem(\_variant\_t((long)1)), TRUE);

m\_TempExlRge.AttachDispatch(m\_TempSheet.GetCells(), TRUE); //加载所有单元格

szText = "No.";

m\_TempExlRge.SetItem(\_variant\_t((long)(iRowIndex)), \_variant\_t((long)(iColumnIndex++)), \_variant\_t(szText));

szText = "Device Name";

m\_TempExlRge.SetItem(\_variant\_t((long)(iRowIndex)), \_variant\_t((long)(iColumnIndex++)), \_variant\_t(szText));

}

catch (...)

{

CloseExcel();

return FALSE;

}

CloseExcel();

return TRUE;

}

* + - * 1. 插入新表单

\_Application ExcelApp;

Workbooks books;

\_Workbook book;

Worksheets sheets;

\_Worksheet sheet;

Range range;

LPDISPATCH lpDisp = NULL;

lpDisp = sheets.Add(vtMissing, vtMissing, \_variant\_t((long)1), vtMissing);

sheet.AttachDispatch(lpDisp);

sheet.SetName(strSheetName); //从前往后

//插入3个(默认表单)以上的表单

try

{

m\_TempSheet.AttachDispatch(m\_TempSheets.GetItem(variant\_t("sheet1")), TRUE);

}

catch(...)

{

try

{

m\_TempSheet.AttachDispatch(m\_TempSheets.GetItem(variant\_t("sheet2")), TRUE);

}

catch(...)

{

try

{

m\_TempSheet.AttachDispatch(m\_TempSheets.GetItem(variant\_t("sheet3")), TRUE);

}

catch(...)

{

LPDISPATCH lpDisp = m\_TempSheets.Add(vtMissing, vtMissing, \_variant\_t((long)1), vtMissing);

m\_TempSheet.AttachDispatch(lpDisp);

m\_TempSheet.AttachDispatch(m\_TempSheets.GetItem(\_variant\_t((long)1)), TRUE);

}

}

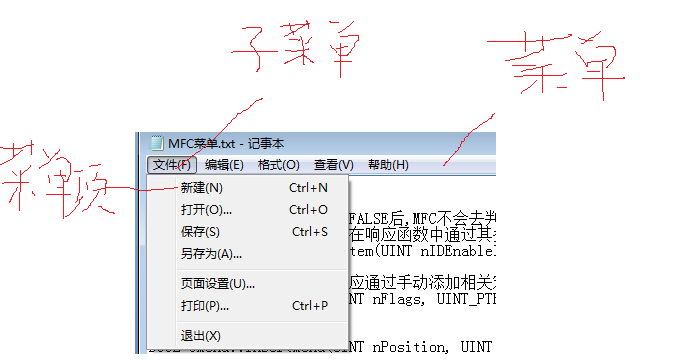
}

m\_TempExlRge.AttachDispatch(m\_TempSheet.GetCells(), TRUE); //加载所有单元格

m\_TempSheet.SetName("BatteryTest"); //更改名字

### CMenu类 菜单

介绍:



菜单项的索引位置从0开始计算（包括分隔符）, 将工具栏ID设置菜单项ID一致，即可关联。

菜单项的访问可以通过菜单项标识ID或者索引位置， 而子菜单只能通过索引位置。

点击菜单发送命令消息ON\_COMMAND( 菜单项ID, 处理函数 )。

菜单(WM\_COMMAND)响应顺序: 视图类->文档类->框架类->应用程序类，可以通过重OnCommand虚函数截取WM\_COMMAND消息

应用:

实例一: 获取系统菜单

CMenu \* m\_pMenu = GetSystemMenu(FALSE);

m\_pMenu->EnableMenuItem(SC\_CLOSE, MF\_BYCOMMAND | MF\_ENABLED);//设置X按钮可用

实例二: 加载一个菜单

CMenu m\_menu;

m\_menu.LoadMenu(菜单资源ID)；

SetMenu(&menu);

成员函数:

(1)BOOL LoadMenu(菜单资源ID); //加载菜单

(2) BOOL AppendMenu(UINT nFlags, UINT\_PTR nIDNewItem = 0, LPCTSTR lpszNewItem = NULL)可以在各种菜单末尾添加一个菜单项,该函数已被InsertMenuItem取代

(3) CMenu\* GetSubMenu(int nPos); 获得子菜单

参数:子菜单的索引位置， 从0开始。

返回值:子菜单对象指针。

1. BOOL InsertMenu (UINT,UINT,UINT, char\*) 在一个子菜单中插入一个菜单项

参数一: 该参数是标识符ID还是索引位置，由第二个参数MF\_BYPOSITION和MF\_BYCOMMAND(没指定则默认它)

参数二:菜单外观等: MF\_SEPARATOR分隔符, MF\_CHECKED 菜单项前面勾选,

MF\_GRAYED菜单项变灰且禁用。

参数三: 新菜单项的标识符，或者当参数二设置为MF\_POPUP时，指定下拉式菜单或子菜单的句柄

参数四: 新菜单项的内容(名称)

1. UINT CheckMenuItem(UINT, UINT) 设置菜单项勾选或非勾选,参数和(4)函数前2个参数一样。
2. BOOL TrackPopupMenu(UINT, int ,int, HWND) 在屏幕任何位置显示一个已有的菜单。

参数一: PM\_CENTERALIGN|TPM\_VCENTERALIGN 水平|垂直

参数二:屏幕x

参数三:屏幕y

参数四:菜单所在的窗口句柄。

1. BOOL SetDefaultItem(UINT uItem, BOOL fByPos = FALSE); //设置默认菜单项(粗体)，第二个参数是FALSE,则第一个参数是菜单项ID，否则为索引位置
2. BOOL DeleteMenu(UINT, UINT)删除菜单

参数一:索引或ID,根据参数二

调用者是子菜单则删除的是菜单项，调用者是菜单则删除子菜单。

1. BOOL EnableMenuItem禁止，允许菜单
2. BOOL CMenu::SetMenuItemBitmaps(UINT, UINT, const CBitmap\* , constCBitmap\*);

给菜单项添加位图

参数一二:位置和标识

参数三:未选中该菜单项图标

参数四:选中该菜单项图标

使用该函数前需要加载位图。

可通过GetSystemMetrics(int nIndex)获取菜单项的前面位图位置的大小

参数: SM\_CXMENUCHECK表示宽度 , SM\_CYMENUCHECK表示高度

### CFileFind类

介绍:

应用:

遍历目录:

void MFCFileFind(CString strPath)

{

CFileFind find;

//"c:\\\*.\*"

BOOL bRet=find.FindFile(strPath+"\\\*.\*");

while(bRet)

{

bRet=find.FindNextFile();

//CString strFileName=find.GetFileName();

CString strFilePath=find.GetFilePath();

if (!find.IsDots())

{

if (find.IsDirectory())

{

printf("目录：%s\n",strFilePath);

//递归

MFCFileFind(strFilePath);

}

else

{

printf("文件:%s\n",strFilePath);

}

}

}

find.Close();//结束查找

}

### MFC常用函数

#### void UpdateWindow() 立即重绘窗口

类似于SendMessag WM\_PAINT消息。

更多操作请参见RedrawWindow函数

#### Invalidate(BOOL == TRUE) 不立即重绘窗口 类似于PostMessage WM\_PAINT。

参数:决定了是否要在WM\_PAINT消息前发送WN\_ERASEBKGND(擦除重绘区背景)

#### RedrawWindow

#### CWinThread\* AfxGetThread( ); //返回当前的线程

#### CWinApp\* AfxGetApp( ) 当前应用进程的指针

#### const char\* AfxGetAppName( ); 返回当前应用程序名称

#### HINSTANCE AfxGetInstanceHandle( ); 返回应用程序实例句柄

#### CWnd\* AfxGetMainWnd( ); 若是OLE服务器，返回应用程序的活动主窗口指针。否则与m\_pMainWnd等价

#### HINSTANCE AfxGetResourceHandle(); 返回应用程序默认的资源

#### AfxParseURLEx分析一个URL信息 <afxinet.h>

#### 判断一个对象的类型

CEdit\* pEdit = new CEdit;

if (pEdit->IsKindOf(RUNTIME\_CLASS(CWnd)))

{

是CWnd类或子类

}