目录

[一、MFC简介 2](#_Toc492721281)

[二、VS MFC应用程序工程设置向导 3](#_Toc492721282)

[三、VS应用程序工程文件组成结构 3](#_Toc492721283)

[3.1 解决方案相关文件 3](#_Toc492721284)

[3.2 工程相关文件 3](#_Toc492721285)

[3.3 应用程序头文件和源文件 3](#_Toc492721286)

[3.4 资源文件 4](#_Toc492721287)

[3.5 预编译头文件 4](#_Toc492721288)

[3.6 编译链接生成文件 4](#_Toc492721289)

[四、MFC应用程序框架分析 4](#_Toc492721290)

[4.1 SDK应用程序和MFC应用程序运行过程 4](#_Toc492721291)

[4.2 MFC框架主要类介绍 4](#_Toc492721292)

[五、MFC消息映射机制 5](#_Toc492721293)

[5.1 什么是消息 5](#_Toc492721294)

[5.2 消息映射机制 5](#_Toc492721295)

[5.3 Windows消息分类 5](#_Toc492721296)

[5.4 消息映射表 5](#_Toc492721297)

[5.5添加消息处理函数 5](#_Toc492721298)

[5.6 各种windows消息的消息处理函数 6](#_Toc492721299)

[六、对话框 6](#_Toc492721300)

[6.1 控件（control） 6](#_Toc492721301)

[6.2 创建类和添加控件变量 7](#_Toc492721302)

[6.3 为控件添加消息处理函数 7](#_Toc492721303)

[6.4 设置对话框控件的Tab顺序 7](#_Toc492721304)

[6.5 模态对话框及其弹出过程 7](#_Toc492721305)

[6.6 非模态对话框的创建及显示 7](#_Toc492721306)

[6.7 属性页对话框的介绍 8](#_Toc492721307)

[6.8 消息对话框 11](#_Toc492721308)

[6.9 文件对话框 12](#_Toc492721309)

[6.10 字体对话框 14](#_Toc492721310)

[6.11 颜色对话框 15](#_Toc492721311)

[七、常用控件 16](#_Toc492721312)

[7.1 静态文本框Static Text 16](#_Toc492721313)

[7.2 编辑框Edit Control 17](#_Toc492721314)

[7.3 按钮控件Button、Radio Button和Check Box 19](#_Toc492721315)

[7.4 列表框控件ListBox 21](#_Toc492721316)

[7.5 组合框控件Combo Box 24](#_Toc492721317)

[7.6 滚动条控件Scroll Bar 27](#_Toc492721318)

[7.7 图片控件Picture Control 29](#_Toc492721319)

[7.8 列表视图控件List Control 29](#_Toc492721320)

[7.9 树形控件 Tree Control 34](#_Toc492721321)

[7.10 标签控件Tab Control 38](#_Toc492721322)

[八、菜单，工具栏与状态栏 40](#_Toc492721323)

[8.1 VS2010菜单资源详解 40](#_Toc492721324)

[8.2工具栏资源详解 42](#_Toc492721325)

[8.3状态栏资源详解 44](#_Toc492721326)

[九、文档，视图和框架 46](#_Toc492721327)

[9.1 概述 46](#_Toc492721328)

[9.2 分割窗口 48](#_Toc492721329)

[十、MFC常用辅助类 50](#_Toc492721330)

[10.1 CString类 50](#_Toc492721331)

[10.2 CTime类与CTimeSpan类 52](#_Toc492721332)

[10.3 定时器Timer 55](#_Toc492721333)

[10.4 CFile文件操作类 58](#_Toc492721334)

[10.5 MFC异常处理 58](#_Toc492721335)

[十一、字体和文本输出 58](#_Toc492721336)

[11.1CFont字体类 58](#_Toc492721337)

[11.2文本输出 58](#_Toc492721338)

[十二、图形图像 58](#_Toc492721339)

[12.1CDC类及其屏幕绘图函数 58](#_Toc492721340)

[12.2GDI对象之画笔CPen 58](#_Toc492721341)

[12.3GDI对象之画刷CBrush 58](#_Toc492721342)

[十三、Ribbon界面开发 58](#_Toc492721343)

[13.1创建Ribbon样式的应用程序框架 58](#_Toc492721344)

[13.2为Ribbon Bar添加控件 58](#_Toc492721345)

[13.3使用更多控件并为其添加消息处理函数 58](#_Toc492721346)

# 一、MFC简介

CUI（command line user interface）界面：命令行模式的人机接口，比如控制台应用程序（MS-DOS程序）。

GUI（graphical user interface）界面：图形用户界面，如windows

MFC（microsoft fundamental class）：微软基础类库，是一个C++类库。主要封装了大部分的windows API函数，即用来创建windows界面程序的类库。MFC既是类库也是框架，MFC工程或程序有特有的文件配置。

窗口：一个标准的用户界面，含有多种辅助特性与工具，比如资源管理器，VS的主界面都是窗口。

对话框：一个简单的窗口，一般没有菜单、工具栏等，只有简单的消息提示与信息交互功能。用户操作也仅限于几个按钮，比如确认，关闭，取消等。当然不能通过这种定义严格区分二者，这二者只是一个模板提供给开发者选择。对话框上也可以任意增添功能实现成为窗口但显然不符合高效率。

字符串前加了个\_T，这是因为工程创建的时候用的默认的Unicode字符集，而如果前不加\_T就是ASCII字符串。\_T实际上是一个宏，工程的字符集选择为Unicode时字符串就转为Unicode字符串，选择为Muli-Byte时就转为ASCII字符串。我们可以在Solution Explorer的Addition根节点上点右键，在右键菜单上选择“Properties”，弹出工程的属性对话框，Configuration Properties->General右侧列表中的Character Set就显示选择的字符集。

# 二、VS MFC应用程序工程设置向导

Application Type：  
single document（单文档）：单文档应用程序运行时是一个单窗口界面  
multiple documents（多文档），dialog based（基于对话框），multiple top-level documents

Compound document support：  
OLE

Document template properties：  
File extension：设置程序能处理的文件的扩展名

Database support：  
Header files only：只包含定义了数据库类的头文件，但不生成对应特定表的数据库类或视图类  
Database view without file support：创建对应指定表的一个数据库类和一个视图类，不附加标准文件支持  
Database view with file support：类似上例  
None：完全不使用数据库特性

User interface features：常用用户界面特性

高级特性：打印特性，最近打开文件等

生成类：将要生成的类名和基类名的设置，常见的有视图类CHelloWorldView，应用类CHelloWorldApp，文档类CHelloWorldDoc，主框架窗口类CMainFrame

# 三、VS应用程序工程文件组成结构

## 3.1 解决方案相关文件

.sdf文件和ipch目录一般占用空间比较大，几十兆甚至上百兆，与智能提示、错误提示、代码恢复和团队本地仓库等相关。如果你觉得不需要则可以设置不生成它们，方法是点击菜单栏Tools->Options，弹出Options对话框，选择左侧面板中Text Editor->C/C++->Advanced，右侧列表中第一项Disable Database由False改为True就可以了，最后关闭VS2010再删除.sdf文件和ipch目录以后就不会再产生了。但关闭此选项以后也会有很多不便，例如写程序时的智能提示没有了。

.sln文件和.suo文件为MFC自动生成的解决方案文件，它包含当前解决方案中的工程信息，存储解决方案的设置。

## 3.2 工程相关文件

.vcxproj文件是MFC生成的工程文件，它包含当前工程的设置和工程所包含的文件等信息。

.vcxproj.filters文件存放工程的虚拟目录信息，也就是在解决方案浏览器中的目录结构信息。

## 3.3 应用程序头文件和源文件

HelloWorld.h：应用程序的主头文件。主要包含由CWinAppEx类派生的CHelloWorldApp类的声明，以及CHelloWorldApp类的全局对象theApp的声明。

HelloWorld.cpp：应用程序的主源文件。主要包含CHelloWorldApp类的实现，CHelloWorldApp类的全局对象theApp的定义等。

MainFrm.h和MainFrm.cpp：通过这两个文件从CFrameWndEx类派生出CMainFrame类，用于创建主框架、菜单栏、工具栏和状态栏等。

HelloWorldDoc.h和HelloWorldDoc.cpp：这两个文件从CDocument类派生出文档类CHelloWorldDoc，包含一些用来初始化文档、串行化（保存和装入）文档和调试的成员函数。

HelloWorldView.h和HelloWorldView.cpp：它们从CView类派生出名为CHelloWorldView的视图类，用来显示和打印文档数据，包含了一些绘图和用于调试的成员函数。

ClassView.h和ClassView.cpp：由CDockablePane类派生出CClassView类，用于实现应用程序界面左侧面板上的Class View。

FileView.h和FileView.cpp：由CDockablePane类派生出CFileView类，用于实现应用程序界面左侧面板上的File View。

OutputWnd.h和OutputWnd.cpp：由CDockablePane类派生出COutputWnd类，用于实现应用程序界面下侧面板Output。

PropertiesWnd.h和PropertiesWnd.cpp：由CDockablePane类派生出CPropertiesWnd类，用于实现应用程序界面右侧面板Properties。

ViewTree.h和ViewTree.cpp：由CTreeCtrl类派生出CViewTree类，用于实现出现在ClassView和FileView等中的树视图。

## 3.4 资源文件

res目录：工程文件夹下的res目录中含有应用程序默认图标、工具栏使用图标等图标文件。

HelloWorld.rc：包含默认菜单定义、字符串表和加速键表，指定了默认的About对话框和应用程序默认图标文件等。

Resource.h：含有各种资源的ID定义。

## 3.5 预编译头文件

stdafx.h：几乎所有的MFC程序的文件都要包含afxwin.h等文件，如果每次都编译一次则会大大减慢编译速度。所以把常用的MFC头文件都放到了该文件中，然后由stdafx.cpp包含stdafx.h文件，编译器对stdafx.cpp只编译一次，并生成编译之后的预编译头HelloWorld.pch，大大提高了编译效率。

## 3.6 编译链接生成文件

工程文件夹下的Debug或Release子文件夹中包含了编译链接时产生的中间文件，解决方案文件夹下的Debug或Release子文件夹中主要包含有应用程序的可执行文件。

# 四、MFC应用程序框架分析

## 4.1 SDK应用程序和MFC应用程序运行过程

SDK：

MFC：程序的外观界面（或模板样式）是由在资源视图resource view下控制设计的，是非常直观的设计方式。首先在资源视图下创建相关的资源。然后设计程序内部的逻辑采用OOP，为每个界面上的控件或元素创建一个对应类来实现运作。创建对应类和主要的函数时VS会提供窗口化的向导wizard来帮助完成。

## 4.2 MFC框架主要类介绍

CHelloWorldApp：处理消息，将收到的消息分发给相应的对象

CMainFrame：是视图CHelloWorldView的父窗口，后者就显示在前者的客户区中

CHelloWorldView：用来显示文档类CHelloWorldDoc中的数据，并根据对视图类的操作修改文档类的数据。一个视图类只能跟一个文档类相联系，而一个文档类可以跟多个视图类相联系。

# 五、MFC消息映射机制

## 5.1 什么是消息

 窗口消息一般由三个部分组成：1.一个无符号整数，是消息值；(2)消息附带的wParam类型的参数；包含有通知消息码（如BN\_CLICKED表示单击）和控件ID；(3)消息附带的lParam类型的参数，包含了控件的句柄。其实我们一般所说的消息是狭义上的消息值，也就是一个无符号整数，经常被定义为宏。

## 5.2 消息映射机制

MFC使用一种消息映射机制来处理消息，在应用程序框架中的表现就是一个消息与消息处理函数一一对应的消息映射表，以及消息处理函数的声明和实现等代码。当窗口接收到消息时，会到消息映射表中查找该消息对应的消息处理函数，然后由消息处理函数进行相应的处理。SDK编程时需要在窗口过程中一一判断消息值进行相应的处理，相比之下MFC的消息映射机制要方便好用的多。

## 5.3 Windows消息分类

Windows消息分为系统消息和用户自定义消息。Windows系统消息有三种：  
1.标准Windows消息。除WM\_COMMAND外以WM\_开头的消息是标准消息。例如，WM\_CREATE、WM\_CLOSE。  
2.命令消息。消息名为WM\_COMMAND，消息中附带了标识符ID来区分是来自哪个菜单、工具栏按钮或加速键的消息。  
3.通知消息。通知消息一般由列表框等子窗口发送给父窗口，消息名也是WM\_xxx，其中附带了控件通知码来区分控件。

CWnd的派生类都可以接收到标准Windows消息、通知消息和命令消息。命令消息还可以由文档类等接收。

用户自定义消息是实际上就是用户定义一个[宏](http://www.jizhuomi.com/software/70.html)作为消息，此宏的值应该大于等于WM\_USER，然后此宏就可以跟系统消息一样使用，窗口类中可以定义它的处理函数。

## 5.4 消息映射表

除了一些没有基类的类或CObject的直接派生类外，其他的类都可以自动生成消息映射表。

在BEGIN\_MESSAG\_MAP和END\_MESSAGE\_MAP之间的内容成为消息映射入口项。消息映射除了在CMainFrame的实现文件中添加消息映射表外，在类的定义文件MainFrm.h中还会添加一个宏调用：

       DECLARE\_MESSAGE\_MAP()

       一般这个宏调用写在类定义的结尾处。

## 5.5添加消息处理函数

       如何添加消息处理函数呢？不管是自动还是手动添加都有三个步骤：

       1.在类定义中加入消息处理函数的函数声明，注意要以afx\_msg打头。例如MainFrm.h中WM\_CREATE的消息处理函数的函数声明：afx\_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);。

       2.在类的消息映射表中添加该消息的消息映射入口项。例如WM\_CREATE的消息映射入口项：ON\_WM\_CREATE()。

       3.在类实现中添加消息处理函数的函数实现。例如，MainFrm.cpp中WM\_CREATE的消息处理函数的实现：

          int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)  
         {  
                  ......  
         }

       通过以上三个步骤以后，WM\_CREATE等消息就可以在窗口类中被消息处理函数处理了。

## 5.6 各种windows消息的消息处理函数

标准Windows消息的消息处理函数都与WM\_CREATE消息类似。

命令消息的消息映射入口项形式如：ON\_COMMAND(ID\_VIEW\_CUSTOMIZE, &CMainFrame:: OnViewCustomize)，消息为ID\_VIEW\_CUSTOMIZE，消息处理函数为OnViewCustomize。如果想要使用某个处理函数批量处理某些命令消息，则可以像CMainFrame消息映射表中的ON\_COMMAND\_RANGE(ID\_VIEW\_APPLOOK\_WIN\_2000, ID\_VIEW\_APPLOOK\_WINDOWS\_7, &CMainFrame::OnApplicationLook)一样添加消息映射入口项，这样值在ID\_VIEW\_APPLOOK\_WIN\_2000到ID\_VIEW\_APPLOOK\_WINDOWS\_7之间的菜单项等的命令消息都由CMainFrame的OnApplicationLook函数处理。函数原型为afx\_msg void OnApplicationLook(UINT id);，参数id为用户操作的菜单项等的ID。

在操作列表框等控件时往往会给父窗口发送WM\_NOTIFY通知消息。WM\_NOTIFY消息的wParam参数为发送通知消息的控件的ID，lParam参数指向一个结构体，可能是NMHDR结构体，也可能是第一个元素为NMHDR结构体变量的其他结构体。NMHDR结构体的定义如下（仅作了解）：  
Typedef sturct tagNMHDR{  
                HWND hwndFrom;  
                UINT idFrom;  
                UINT code;  
       } NMHDR;  
hwndFrom为发送通知消息控件的句柄，idFrom为控件ID，code为要处理的通知消息的通知码，例如NM\_CLICK。  
通知消息的消息映射入口项形式如：  
ON\_NOTIFY(wNotifyCode,id,memberFxn)  
wNotifyCode为要处理的通知消息通知码，例如：NM\_CLICK。id为控件标识ID。MemberFxn为此消息的处理函数。  
通知消息的处理函数的原型为：  
afx\_msg void memberFxn( NMHDR \* pNotifyStruct, LRESULT \* result);

如果需要使用用户自定义消息，首先要定义消息宏，如：#define WM\_UPDATE\_WND (WM\_USER+1)，再到消息映射表中添加消息映射入口项：ON\_MESSAGE(WM\_UPDATE\_WND, &CMainFrame::OnUpdateWnd)，然后在MainFrm.h中添加消息处理函数的函数声明：afx\_msg LRESULT OnUpdateWnd(WPARAM wParam, LPARAM lParam);，最后在MainFrm.cpp中实现此函数。

# 六、对话框

VS操作界面上的设计方法和VB类似，都是直接基于窗口实体模型和控件实体图标的直观方式。这些模型和控件图标都在资源文件中由资源视图进行管理。

默认创建的对话框包括项目同名的主对话框和ABOUTBOX（“关于”对话框）。

## 6.1 控件（control）

静态文本框static text：无法给用户编辑的内嵌文本

编辑框edit control：用来输入的文本框

……

## 6.2 创建类和添加控件变量

**注意，**一般类名都以C打头，又比如，CTestDlg。

**注意，**类的成员变量名一般以m\_打头，以标识它是一个成员变量。

(1) 为对话框模板创建一个对应类

(2) 为对话框中的控件添加值变量

(3) 对话框类的数据交换和检验  
控件的属性改变反映到关联变量上的操作由CAdditionDlg::DoDataExchange()来实现。这个函数由VS自动生成代码和改动。但需要手动调用，通过调用CDialogEx::UpdateData()来调用。  
BOOL UpdateData(BOOL bSaveAndValidate = TRUE);  
参数：bSaveAndValidate用于指示数据传输的方向，TRUE表示从控件传给变量，FALSE表示从变量传给控件。默认值是TRUE，即从控件传给变量。  
返回值：CDialogEx::UpdateData()函数的返回值表示操作是否成功，成功则返回TRUE，否则返回FALSE。

## 6.3 为控件添加消息处理函数

添加消息处理函数其实也就是事件（event）回调函数，有以下几种方法：  
(1) 使用Class Wizard添加消息处理函数  
(2) 通过“Add Event Handler...”添加消息处理函数  
(3) 在按钮的属性视图中添加消息处理函数  
(4) 双击按钮添加消息处理函数

推荐使用按钮属性视图方法，简单并且全面而直接。添加完函数后，即可编辑函数内容实现程序逻辑。

## 6.4 设置对话框控件的Tab顺序

Tab顺序，是按下tab键时控件的轮转顺序。为便于依靠键盘快速连续输入多个文本框数据，必须合理设计Tab顺序。

在主菜单中选择“Format”->"Tab Order"，或者按快捷键Ctrl+D，对话框模板上就会显示各个控件的Tab顺序数字。

从自己认为Tab顺序应该为1的控件开始依次单击，最后按ESC键，确认设置并退出对话框模板的Tab顺序设置状态。

## 6.5 模态对话框及其弹出过程

Windows对话框分为两类：模态对话框和非模态对话框。

模态对话框是这样的对话框，当它弹出后，本应用程序其他窗口将不再接受用户输入，只有该对话框响应用户输入，在对它进行相应操作退出后，其他窗口才能继续与用户交互。

非模态对话框则是，它弹出后，本程序其他窗口仍能响应用户输入。非模态对话框一般用来显示提示信息等。

弹出过程：  
通过对话框对象的成员DoModal()弹出对话框，其返回值为用户关闭该对话框时点击按钮的ID（一般是OK或Cancel）。根据返回值进行对应操作。另外返回值为整数值，并指定了传递给CDialog::EndDialog（该函数用于关闭对话框）的nResult参数值。如果函数不能创建对话框，则返回-1；如果出现其它错误，则返回IDABORT。

## 6.6 非模态对话框的创建及显示

弹出过程：  
首先在头文件中创建对话框类型指针，然后动态创建一个对话框对象，并使用Create()成员函数完成非模态对话框的创建，最后使用ShowWindow()成员显示对话框。因为动态创建，所以在析构函数中应删除。

## 6.7 属性页对话框的介绍

属性页对话框即用来设置属性的一种对话框，具体根据细节特征分为两种：一种标签切换式的一般属性页对话框，另一种是“next”按钮切换式的向导对话框。属性页对话框即将多个对话框集成于一身，通过标签或按钮来切换页面。

属性页对话框使用的主要类有：

CPropertyPage：  
CPropertyPage类继承自CDialog类，它被用于处理某单个的属性页，所以要为每个属性页都创建一个继承自CPropertyPage的子类。  
(1) [构造函数](http://www.jizhuomi.com/software/51.html)  
函数原型为：  
CPropertyPage( );  
explicit CPropertyPage(  
                UINT nIDTemplate,  
                UINT nIDCaption = 0,  
                DWORD dwSize = sizeof(PROPSHEETPAGE));  
explicit CPropertyPage(  
                LPCTSTR lpszTemplateName,  
                UINT nIDCaption = 0,  
                DWORD dwSize = sizeof(PROPSHEETPAGE));  
第一个是没有任何参数的构造函数。  
第二个构造函数中，参数nIDTemplate是属性页的[对话框资源](http://www.jizhuomi.com/software/149.html)ID，参数nIDCaption是属性页对话框选项卡的标题所用字符串资源的ID，若设为0，则选项卡标题就使用该属性页的对话框资源的标题。  
第三个构造函数中，参数lpszTemplateName为属性页的对话框资源的名称字符串，不能为NULL。参数nIDCaption同上。  
(2) CancelToClose()函数  
在模态属性页对话框的属性页进行了某不可恢复的操作后，使用CancelToClose()函数将“OK”按钮改为“Close”按钮，并禁用“Cancel”按钮。函数原型为：  
void CancelToClose( );  
(3) SetModified()函数  
调用此函数可激活或禁用“Apply”按钮，函数原型为：  
void SetModified(BOOL bChanged = TRUE);  
(4) 可[重载函数](http://www.jizhuomi.com/software/45.html)  
CPropertyPage类提供了一些[消息处理函数](http://www.jizhuomi.com/software/156.html)，来响应属性页对话框的各种消息。我们重载这些消息处理函数，就可以自定义对属性页对话框操作的处理。可重载的消息处理函数包括：  
       OnApply：处理属性页的“Apply”按钮被单击的消息  
       OnCancel：处理属性页的“Cancel”按钮被单击的消息  
       OnKillActive：处理属性页当前活动状态被切换的消息，常用于数据验证  
       OnOK：处理属性页的“OK”按钮、“Apply”按钮或者“Close”按钮被单击的消息  
       OnQueryCancel：处理属性页的“Cancel”按钮被单击前发出的消息  
       OnReset：处理属性页的“Reset”按钮被单击的消息  
       OnSetActive：处理属性页被切换为当前活动页的消息  
       OnWizardBack：处理属性页的“Back”按钮被单击的消息，仅在向导对话框中有效  
       OnWizardFinish：处理属性页的“Finish”按钮被单击的消息，仅在向导对话框中有效  
       OnWizardNext：处理属性页的“Next”按钮被单击的消息，仅在向导对话框中有效

CPropertySheet：  
CPropertySheet类继承自CWnd类，它是属性表类，负责加载、打开或删除属性页，并可以在属性页对话框中切换属性页。它跟对话框类似，也有模态和非模态两种。  
(1) 构造函数  
CPropertySheet( );  
explicit CPropertySheet(  
               UINT nIDCaption,  
               CWnd\* pParentWnd = NULL,  
               UINT iSelectPage = 0 );  
explicit CPropertySheet(  
               LPCTSTR pszCaption,  
               CWnd\* pParentWnd = NULL,  
               UINT iSelectPage = 0 );  
参数nIDCaption：标题的字符串资源的ID。  
参数pParentWnd：属性页对话框的父窗口，若设为NULL，则父窗口为应用程序的主窗口。  
参数iSelectPage：初始状态时，活动属性页的索引，默认为第一个添加到属性表的属性页。  
参数pszCaption：标题字符串。  
(2) GetActiveIndex()函数  
获取当前活动属性页的索引。函数原型为：  
int GetActiveIndex( ) const;  
返回值：当前活动属性页的索引。  
(3) GetActivePage()函数  
获取当前活动属性页对象。函数原型为：  
CPropertyPage\* GetActivePage( ) const;  
返回值：当前活动属性页对象的指针。  
(4) GetPage()函数  
获取某个属性页对象。函数原型为：  
CPropertyPage\* GetPage(int nPage) const;  
参数nPage：目标属性页的索引。  
返回值：目标属性页对象的指针。  
(5)GetPageCount()函数  
获取属性页的数量。函数原型为：  
int GetPageCount( ) const;  
返回值：属性页的数量。  
(6)GetPageIndex()函数  
获取某属性页在属性页对话框中的索引。函数原型为：  
int GetPageIndex(CPropertyPage\* pPage);  
参数pPage：要获取索引的属性页对象的指针。  
返回值：属性页对象在属性页对话框中的索引。  
(7)SetActivePage()函数  
设置某个属性页为活动属性页。函数原型为：     
BOOL SetActivePage(  
                 int nPage   
       );  
BOOL SetActivePage(  
                 CPropertyPage\* pPage   
       );  
参数nPage：要设置为活动属性页的索引。  
参数pPage：要设置为活动属性页的对象指针。  
(8)SetWizardButtons()函数  
在向导对话框上启用或禁用Back、Next或Finish按钮，应在调用DoModal之前调用此函数。函数原型为：  
void SetWizardButtons(  
                DWORD dwFlags   
       );  
参数dwFlags：设置向导按钮的外观和功能属性。可以是以下值的组合：  
PSWIZB\_BACK                    启用“Back”按钮，如果不包含此值则禁用“Back”按钮。  
       PSWIZB\_NEXT                    启用“Next”按钮，如果不包含此值则禁用“Next”按钮。  
       PSWIZB\_FINISH                  启用“Finish”按钮。  
       PSWIZB\_DISABLEDFINISH   显示禁用的“Finish”按钮。  
(9)SetWizardMode()函数  
设置属性页对话框为向导对话框模式，应在调用DoModal之前调用此函数。函数原型为：  
void SetWizardMode( );  
(10)SetTitle()函数  
设置属性对话框的标题。函数原型为：  
void SetTitle(  
               LPCTSTR lpszText,  
               UINT nStyle = 0   
       );  
参数lpszText：标题字符串。  
参数nStyle：指定属性表标题的风格。应当为0或PSH\_PROPTITLE。如果设为PSH\_PROPTITLE，则单词“Properties”会出现在指定标题之后。例如，SetTitle("Simple",PSH\_PROPTITLE)这种调用会使得属性表标题为“Simple Properties”。  
(11)AddPage()函数  
为属性对话框添加新的属性页。函数原型为：  
void AddPage(  
               CPropertyPage \*pPage   
       );  
参数pPage：要添加的新的属性页的对象指针。  
(12)PressButton()函数  
模拟按下某指定的按钮。函数原型为：     
void PressButton(  
               int nButton   
       );  
参数nButton：要模拟按下的按钮，它可以是下列值之一：  
       PSBTN\_BACK   选择“Back”按钮。   
       PSBTN\_NEXT   选择“Next”按钮。  
       PSBTN\_FINISH   选择“Finish”按钮。  
       PSBTN\_OK   选择“OK”按钮。  
       PSBTN\_APPLYNOW   选择“Apply”按钮。  
       PSBTN\_CANCEL   选择“Cancel”按钮。  
       PSBTN\_HELP   选择“帮助”按钮。  
(13)RemovePage()函数  
删除某属性页。函数原型为：  
void RemovePage(  
               CPropertyPage \*pPage   
       );  
void RemovePage(  
               int nPage   
       );  
参数pPage：要删除的属性页的对象指针。  
参数nPage：要删除的属性页的索引。

一般属性页对话框和向导型对话框的class基本构架一致，只是调用函数有区别。

## 6.8 消息对话框

MFC提供了两个函数可以方便直接的生成固定风格的消息对话框，用以作为提示信息之用。不必每次都需创建资源和类等。

(1) CWnd::MessageBox()函数  
CWnd::MessageBox()的函数原型如下：  
int MessageBox(  
           LPCTSTR lpszText,  
           LPCTSTR lpszCaption = NULL,  
           UINT nType = MB\_OK   
       );  
参数说明：  
lpszText：需要显示的消息字符串。  
lpszCaption：消息对话框的标题字符串。默认值为NULL。取值为NULL时使用默认标题。  
nType：消息对话框的风格和属性。默认为MB\_OK风格，即只有“确定”按钮。  
nType的取值可以是下面两个表中任取一个值，也可以是各取一个值的任意组合。即可以指定一个对话框类型，也可以指定一个对话框图标，还可以两者都设定。

|  |  |
| --- | --- |
| nType 取值 | 参数说明 |
| MB\_ABORTRETRY | 有“终止”、“重试”和“忽略”按钮 |
| MB\_OK | 有“确定”按钮 |
| MB\_OKCANCEL | 有“确定”和“取消”按钮 |
| MB\_RETRYCANCEL | 有“重试”和“取消”按钮 |
| MB\_YESNO | 有“是”和“否”按钮 |
| MB\_YESNOCANCEL | 有“是”、“否”和“取消”按钮 |

对话框类型表

|  |  |
| --- | --- |
| nType 取值 | 显示图标 |
| MB\_ICONEXCLAMTION MB\_ICONWARNING | VS2010/MFC编程入门之十六（对话框：消息对话框） |
| MB\_ICONASTERISK MB\_ICONINFORMATION | VS2010/MFC编程入门之十六（对话框：消息对话框） |
| MB\_ICONQUESTION | VS2010/MFC编程入门之十六（对话框：消息对话框） |
| MB\_ICONHAND MB\_ICONSTOP MB\_ICONERROR | VS2010/MFC编程入门之十六（对话框：消息对话框） |

对话框图标表  
如果想要设置nType的值为类型和图标的组合，可以像这样取值：MB\_OKCANCEL | MB\_ICONQUESTION。按位取或就可以了。

(2)AfxMessageBox()函数  
AfxMessageBox()的函数原型为：  
int AfxMessageBox(  
           LPCTSTR lpszText,  
           UINT nType = MB\_OK,  
           UINT nIDHelp = 0   
       );  
参数说明：  
lpszText：同CWnd::MessageBox()函数  
nType：CWnd::MessageBox()函数  
nIDHelp：此消息的帮助的上下文ID。默认值为0，取0时表示要使用应用程序的默认帮助上下文。  
我们在调用了上面两个函数后，都可以弹出模态消息对话框。消息对话框关闭后，我们也都可以得到它们的返回值。两者的返回值就是用户在消息对话框上单击的按钮的ID，可以是以下值：  
      IDABORT：单击“终止”按钮。  
      IDCANCEL：单击“取消”按钮。  
      IDIGNORE：单击“忽略”按钮。  
      IDNO：单击“否”按钮。  
      IDOK：单击“确定”按钮。  
      IDRETRY：单击“重试”按钮。  
      IDYES：单击“是”按钮。

## 6.9 文件对话框

文件对话框分为打开文件对话框和保存文件对话框

文件对话框类CFileDialog  
MFC使用文件对话框类CFileDialog封装了对文件对话框的操作。CFileDialog类的[构造函数](http://www.jizhuomi.com/software/51.html)原型如下：

explicit CFileDialog(  
   BOOL bOpenFileDialog,  
   LPCTSTR lpszDefExt = NULL,  
   LPCTSTR lpszFileName = NULL,  
   DWORD dwFlags = OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT,  
   LPCTSTR lpszFilter = NULL,  
   CWnd\* pParentWnd = NULL,  
   DWORD dwSize = 0,  
   BOOL bVistaStyle = TRUE  
);  
参数说明：  
bOpenFileDialog：指定要创建的文件对话框的类型。设为TRUE将创建打开文件对话框，否则将创建保存文件对话框。  
lpszDefExt：默认的文件扩展名。如果用户在文件名编辑框中没有输入扩展名，则由lpszDefExt指定的扩展名将被自动添加到文件名后。默认为NULL。  
lpszFileName：文件名编辑框中显示的初始文件名。如果为NULL，则不显示初始文件名。  
dwFlags：文件对话框的属性，可以是一个值也可以是多个值的组合。关于属性值的定义，可以在MSDN中查找结构体OPENFILENAME，元素Flags的说明中包含了所有属性值。默认为OFN\_HIDEREADONLY和OFN\_OVERWRITEPROMPT的组合，OFN\_HIDEREADONLY表示隐藏文件对话框上的“Read Only”复选框，OFN\_OVERWRITEPROMPT表示在保存文件对话框中如果你选择的文件存在了，就弹出一个消息对话框，要求确定是否要覆盖此文件。  
lpszFilter：文件过滤器，它是由若干字符串对组成的一个字符串序列。如果指定了文件过滤器，则文件对话框中只有符合过滤条件的文件显示在文件列表中待选择。给大家看看VS2010 MSDN中给出的一个例子：  
static TCHAR BASED\_CODE szFilter[] = \_T("Chart Files (\*.xlc)|\*.xlc|Worksheet Files (\*.xls)|\*.xls|Data Files (\*.xlc;\*.xls)|\*.xlc; \*.xls|All Files (\*.\*)|\*.\*||");  
这样设置过滤器以后，文件对话框的扩展名组合框中将有四个选项：Chart Files (\*.xlc)、Worksheet Files (\*.xls)、Data Files(\*.xlc;\*.xls)和All Files (\*.\*)，大家可以看到每种文件的扩展名规定都是一个字符串对，例如Chart Files的过滤字符串是Chart Files(\*.xlc)和\*.xlc成对出现的。  
pParentWnd：文件对话框的父窗口的指针。  
dwSize：OPENFILENAME结构体的大小。不同的操作系统对应不同的dwSize值。MFC通过此参数决定文件对话框的适当类型（例如，创建Windows 2000文件对话框还是XP文件对话框）。默认为0，表示MFC将根据程序运行的操作系统版本来决定使用哪种文件对话框。  
bVistaStyle：指定文件对话框的风格，设为TRUE则使用Vista风格的文件对话框，否则使用旧版本的文件对话框。此参数仅在Windows Vista中编译时适用。  
文件对话框也是模态对话框，所以在打开时也需要调用CFileDialog类的DoModal()成员函数。在打开文件对话框中点了“打开”或者在保存文件对话框中点了“保存”以后，我们可以使用CFileDialog类的成员函数GetPathName()获取选择的文件路径。

下面列出几个CFileDialog类的成员函数，我们可以使用它们获得文件对话框中的各种选择。  
GetFileExt()：获得选定文件的后缀名。  
GetFileName()：获得选定文件的名称，包括后缀名。  
GetFileTitle()：获得选定文件的标题，即不包括后缀名。  
GetFolderPath()：获得选定文件的目录。  
GetNextPathName()：获得下一个选定的文件的路径全名。  
GetPathName()：获得选定文件的路径全名。  
GetReadOnlyPref()：获得是否“以只读方式打开”。  
GetStartPosition()：获得文件名列表中的第一个元素的位置。

## 6.10 字体对话框

CFontDialog类的[构造函数](http://www.jizhuomi.com/software/51.html)  
我们先来了解CFontDialog类。它的常用构造函数原型如下：

CFontDialog(  
   LPLOGFONT lplfInitial = NULL,  
   DWORD dwFlags = CF\_EFFECTS | CF\_SCREENFONTS,  
   CDC\* pdcPrinter = NULL,  
   CWnd\* pParentWnd = NULL   
);  
参数说明：  
lplfInitial：指向LOGFONT结构体数据的指针，可以通过它设置字体的一些特征。  
dwFlags：指定选择字体的一个或多个属性，详情可在MSDN中查阅。  
pdcPrinter：指向一个打印设备上下文的指针。  
pParentWnd：指向字体对话框父窗口的指针。  
上面的构造函数中第一个参数为LOGFONT指针，LOGFONT结构体中包含了字体的大部分特征，包括字体高度、宽度、方向、名称等等。下面是此结构体的定义：

typedef struct tagLOGFONT {  
    LONG lfHeight;  
    LONG lfWidth;  
    LONG lfEscapement;  
    LONG lfOrientation;  
    LONG lfWeight;  
    BYTE lfItalic;  
    BYTE lfUnderline;  
    BYTE lfStrikeOut;  
    BYTE lfCharSet;  
    BYTE lfOutPrecision;  
    BYTE lfClipPrecision;  
    BYTE lfQuality;  
    BYTE lfPitchAndFamily;  
    TCHAR lfFaceName[LF\_FACESIZE];  
} LOGFONT;

获取字体对话框中所选字体

       我们在字体对话框中选择了字体后，如何获取选定的字体呢？我们可以通过CFontDialog类的成员变量m\_cf间接获得选定字体的CFont对象。m\_cf是CHOOSEFONT类型的变量，CHOOSEFONT结构体定义如下：

typedef struct {  
    DWORD lStructSize;  
    HWND hwndOwner;  
    HDC hDC;  
    LPLOGFONT lpLogFont;  
    INT iPointSize;  
    DWORD Flags;  
    COLORREF rgbColors;  
    LPARAM lCustData;  
    LPCFHOOKPROC lpfnHook;  
    LPCTSTR lpTemplateName;  
    HINSTANCE hInstance;  
    LPTSTR lpszStyle;  
    WORD nFontType;  
    INT nSizeMin;  
    INT nSizeMax;  
} CHOOSEFONT, \*LPCHOOSEFONT;  
CHOOSEFON结构体中有个成员lpLogFont，它是指向LOGFONT结构体变量的指针，就像上面所说，LOGFONT中包含了字体特征，例如，我们可以通过LOGFONT的lfFaceName得知字体名。

我们最终要获得的是所选择字体的CFont对象，有了字体的LOGFONT怎样获得对应的CFont对象呢？使用CFont类的成员函数CreateFontIndirect可以达到此目的。函数原型如下：  
BOOL CreateFontIndirect(const LOGFONT\* lpLogFont );  
参数是LOGFONT指针类型，我们可以传入CFontDialog类成员变量m\_cf的lpLogFont成员，就可以得到所选字体的CFont对象了。

## 6.11 颜色对话框

CColorDialog类的[构造函数](http://www.jizhuomi.com/software/51.html)

CColorDialog(  
   COLORREF clrInit = 0,  
   DWORD dwFlags = 0,  
   CWnd\* pParentWnd = NULL   
);  
参数说明：  
clrInit：默认选择颜色的颜色值，类型为COLORREF，实际上就是unsigned long类型。如果没有设置它的值，则默认为RGB(0,0,0)，即黑色。  
注：RGB(r,g,b)是[宏](http://www.jizhuomi.com/software/70.html)，可以计算颜色值。括号中的三个值分别为红、绿、蓝分量的值。  
dwFlags：自定义颜色对话框功能和外观的属性值。详情可在MSDN中查阅。  
pParentWnd：颜色对话框的父窗口的指针。

获取颜色对话框中所选颜色值  
我们使用颜色对话框的最终目的还是要获得在颜色对话框中选择的颜色值。为此CColorDialog类的成员函数GetColor()能够很好的实现我们的要求。GetColor()函数的原型为：  
COLORREF GetColor( ) const;  
它返回所选颜色的COLORREF值。  
如果我们想获得R、G、B各分量的值呢？可以根据GetColor得到的COLORREF颜色值，通过使用GetRValue、GetGValue和GetBValue三个宏获得。GetRValue的语法形式为：  
BYTE GetRValue(DWORD rgb);  
参数rgb就是COLORREF颜色值，返回值即是R分量值。其他两个宏的形式与之类似。例如，GetColor()函数返回的COLORREF为10000，则R分量值就是GetRValue(10000)。

# 七、常用控件

## 7.1 静态文本框Static Text

如果我们想在程序中动态创建静态文本框，而不是像前面那样直接从Toolbox中拖到对话框模板上，那么就需要使用CStatic类的成员函数Create。Create函数的原型如下：

virtual BOOL Create(  
   LPCTSTR lpszText,  
   DWORD dwStyle,  
   const RECT& rect,  
   CWnd\* pParentWnd,  
   UINT nID = 0xffff   
);  
参数说明：  
lpszText：指定要在控件中显示的文字。如果为NULL则不会显示任何文字。  
dwStyle：指定静态控件的风格。静态文本框一般都是对话框或其他窗口的子窗口，而且是可见的，所以应该包含WS\_CHILD 和WS\_VISIBLE风格，另外，MSDN中说明，还可以为其设置“static control styles”中风格的任意组合。下面大概为大家说明几个风格：

|  |  |
| --- | --- |
| SS\_BITMAP | 一个位图将显示在静态控件中，Create函数的lpszText参数字符串是资源文件中定义的位图名。此风格忽略宽度和高度参数，静态控件自动调整它的尺寸来适应位图 |
| SS\_BLACKFRAME | 指定一个具有与窗口边界同色的框，默认为黑色 |
| SS\_BLACKRECT | 指定一个具有与窗口边界同色的实矩形，默认为黑色 |
| SS\_CENTER | 使显示的正文居中对齐，正文可以换行 |
| SS\_GRAYFRAME | 指定一个具有与屏幕背景同色的边框 |
| SS\_GRAYRECT | 指定一个具有与屏幕背景同色的实矩形 |
| SS\_ICON | 使控件显示一个在资源中定义的图标，图标的名字由Create 函数的lpszText 参数指定，图标自动调整它的尺寸 |
| SS\_LEFT | 左对齐正文，正文能回绕 |
| SS\_LEFTNOWORDWRAP | 左对齐正文，正文不能回绕 |
| SS\_NOTIFY | 使控件能向父窗口发送鼠标事件消息 |
| SS\_RIGHT | 右对齐正文，可以回绕 |
| SS\_SIMPLE | 使静态正文在运行时不能被改变并使正文显示在单行中 |
| SS\_WHITEFRAME | 指定一个具有与窗口背景同色的框，默认为白色 |
| SS\_WHITERECT | 指定一个具有与窗口背景同色的实心矩形，默认为白色 |

  rect：指定静态控件的位置和大小，它可以是RECT结构体类型，也可以是CRect类的对象。  
pParentWnd：指定静态控件的父窗口，通常是一个CDialog对象，不能是NULL。  
nID：指定静态控件的ID。

CStatic类的成员函数简介

|  |  |
| --- | --- |
| GetBitmap | 获取由SetBitmap函数设置的位图的句柄 |
| GetCursor | 获取由SetCurSor设置的光标的句柄 |
| GetEnhMetaFile | 获取由SetEnhMetaFile设置的增强图元文件的句柄 |
| GetIcon | 获取由SetIcon设置的图标的句柄 |
| SetBitmap | 设置要在静态控件中显示的位图 |
| SetCursor | 设置要在静态控件中显示的光标图片 |
| SetEnhMetaFile | 设置要在静态控件中显示的增强图元文件 |
| SetIcon | 设置要在静态控件中显示的图标 |

除了上述成员函数外，由于CStatic是CWnd的派生类，CWnd的很多成员函数也可以使用，例如，GetWindowText、GetWindowRect、SetWindowText等。

## 7.2 编辑框Edit Control

 编辑框的通知消息  
编辑框发生某些事件时会向父窗口发送通知消息。在对话框模板中的编辑框上点右键，选择“Add Event Handler”，为编辑框添加消息处理函数时，可以在“Message type”列表中看到这些消息。下面简单介绍编辑框的部分通知消息。  
EN\_CHANGE：编辑框的内容被用户改变了，与EN\_UPDATE 不同，该消息是在编辑框显示的正文被刷新后才发出的  
EN\_ERRSPACE： 编辑框控件无法申请足够的动态内存来满足需要  
EN\_HSCROLL： 用户在水平滚动条上单击鼠标  
EN\_KILLFOCUS： 编辑框失去输入焦点  
EN\_MAXTEXT：输入的字符超过了规定的最大字符数。在没有ES\_AUTOHSCROLL 或  
ES\_AUTOVSCROLL： 的编辑框中，当正文超出了编辑框的边框时也会发出该消息  
EN\_SETFOCUS： 编辑框获得输入焦点  
EN\_UPDATE： 在编辑框准备显示改变了的正文时发送该消息  
EN\_VSCROLL： 用户在垂直滚动条上单击鼠标

编辑框的创建  
[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)为编辑框提供了CEdit类。编辑框的所有操作都封装到了CEdit类中。  
与静态文本框的创建类似，除了可以在[对话框模板](http://www.jizhuomi.com/software/149.html)上拖进一个编辑框，然后关联一个变量或通过API函数使用，也可以在程序中动态创建编辑框，即调用CEdit类的成员函数Create。Create成员函数的原型如下：  
virtual BOOL Create(  
   DWORD dwStyle,  
   const RECT& rect,  
   CWnd\* pParentWnd,  
   UINT nID   
);  
参数说明：  
dwStyle：指定编辑框的风格。可以是[MSDN](http://www.jizhuomi.com/software/139.html)中“edit styles”包含风格的任意组合。下面是“edit styles”的所有风格说明。  
ES\_AUTOHSCROLL：当用户在行尾键入一个字符时，正文将自动向右滚动10 个字符，当用户按回车键时，正文总是滚向左边  
ES\_AUTOVSCROLL： 当用户在最后一个可见行按回车键时，正文向上滚动一页  
ES\_CENTER： 在多行编辑框中使正文居中  
ES\_LEFT ：左对齐正文  
ES\_LOWERCASE： 把用户输入的字母统统转换成小写字母  
ES\_MULTILINE：指定一个多行编辑器。若多行编辑器不指定ES\_AUTOHSCROLL 风格，则会自动换行，若不指定ES\_AUTOVSCROLL，则多行编辑器会在窗口中正文装满时  
发出警告声响  
ES\_NOHIDESEL：默认时，当编辑框失去输入焦点后会隐藏所选的正文，当获得输入焦点时又显示出来。设置该风格可禁止这种默认行为  
ES\_NUMBER ：编辑框中只允许输入数字  
ES\_OEMCONVERT：使编辑框中的正文可以在ANSI 字符集和OEM 字符集之间相互转换。这在编辑框中包含文件名时是很有用的  
ES\_PASSWORD： 使所有键入的字符都用“\*”来显示  
ES\_READONLY： 将编辑框设置成只读的  
ES\_RIGHT ：右对齐正文  
ES\_UPPERCASE： 把用户输入的字母统统转换成大写字母  
ES\_WANTRETURN：使多行编辑器接收回车键输入并换行。如果不指定该风格，按回车键会选择默认的命令按钮，这往往会导致对话框的关闭  
除了上面的风格外，编辑款一般还会设置WS\_CHILD、WS\_VISIBLE、WS\_BORDER等窗口风格。另外，编辑框可以是多行的，也就是在编辑框中显示多行文字，这就需要设置ES\_MULTILINE风格，如果想要多行编辑框支持回车键，则还要设置ES\_WANTRETURN。  
对于在对话框模板中创建的编辑框，它的属性中包含了上述的风格，例如，Multiline属性对应的就是ES\_MULTILINE风格，Want Return属性对应ES\_WANTRETURN风格。  
其他三个参数与静态文本框的Create函数的参数类似，就不介绍了。

CEdit类的主要成员函数

       使用编辑框最重要的莫过于，获取和设置编辑框中的正文，它们对应的成员函数分别是GetWindowText和SetWindowText，这两个函数都是继承自CWnd类的成员函数，另外，还可以使用CWnd类的GetWindowTextLength函数获取编辑框中正文的长度。  
下面简单介绍CEdit类的其他几个主要的成员函数：  
int LineFromChar(int nIndex = –1) const;

       返回多行编辑框中指定索引的字符所在行的行号（从零开始），只适用于多行编辑框。nIndex等于-1则返回所选择正文的第一个字符所在行的索引。如果没有选择正文，则返回当前行的行号。  
int LineIndex(int nLine = –1) const;  
返回由nLine指定行的起始字符在编辑框的整个字符串中的索引，只适用于多行编辑框。如果指定行超过编辑框的最大行数，则返回-1，而如果nLine为-1，则返回当前插入符所在行的起始字符的索引。  
void GetSel(int& nStartChar,int& nEndChar) const;  
获取选择正文的索引范围。nStartChar返回被选择正文的起始索引，nEndChar返回被选择正文的终止索引（不包括在选择范围内）。如果没有选择正文，则两者均为当前插入符的索引。  
void SetSel(int nStartChar,int nEndChar,BOOL bNoScroll=FALSE);  
选择编辑框中的正文。nStartChar为选择开始处的索引，nEndChar为选择结束处的索引。如果nStartChar为0并且nEndChar为-1，则选择所有正文，而如果nStartChar为-1则取消所有选择。bNoScroll为FALSE时滚动插入符并使之可见，为TRUE时不滚动。  
void ReplaceSel(LPCTSTR lpszNewText,BOOL bCanUndo = FALSE);  
用lpszNewText指向的字符串来替换选择的正文。如果bCanUndo为TRUE则替换可以被撤销。  
int GetLineCount() const;  
获取正文的行数，只适用于多行编辑框。如果编辑框没有正文则返回1。  
int LineLength( int nLine = –1 ) const;  
获取指定字符索引所在行的字节长度（行尾的回车和换行符不计算在内），参数nLine 说明了为字符索引。如果nLine 的值为-1，则函数返回当前行的长度（假如没有正文被选择），或选择正文占据的行的字符总数减去选择正文的字符数（假如有正文被选择）。若用于单行编辑框，则函数返回整个正文的长度。  
int GetLine( int nIndex, LPTSTR lpszBuffer ) const;  
       int GetLine( int nIndex, LPTSTR lpszBuffer, int nMaxLength ) const;  
用来获得指定行的正文（不包括行尾的回车和换行符），只适用于多行编辑框。参数nIndex 是行号，lpszBuffer 指向存放正文的缓冲区，nMaxLength 规定了拷贝的最大字节数。若指定的行号小于编辑框的实际行数，函数返回实际拷贝的字节数，若指定的行号大于编辑框的实际行数，则函数返回0。需要注意的是，GetLine 函数不会在缓冲区中字符串的末尾添加字符串结束符（NULL）。  
UINT GetLimitText( ) const;  
获取编辑框能够接受的正文的最大字节数。  
void LimitText(int nChars = 0);  
设置用户在编辑框中可以输入的正文的最大长度（字节数）。如果nChars为0，则最大长度为UINT\_MAX个字节。

## 7.3 按钮控件Button、Radio Button和Check Box

按钮控件包括命令按钮（Button）、单选按钮（Radio Button）和复选框（Check Box）等。

按钮控件的创建  
[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)提供了CButton类封装按钮控件的所有操作。  
之前的教程中，我们是在对话框模板上直接添加的按钮控件资源，但某些特殊情况下需要我们动态创建按钮控件，即通过CButton类的成员函数Create来创建按钮。下面是Create函数的原型：  
virtual BOOL Create(  
   LPCTSTR lpszCaption,  
   DWORD dwStyle,  
   const RECT& rect,  
   CWnd\* pParentWnd,  
   UINT nID   
);  
参数说明：  
lpszCaption：指定按钮控件显示的文本。  
dwStyle：指定按钮控件的风格，可以设置为以下按钮风格的任意组合。  
BS\_AUTOCHECKBOX ：同BS\_CHECKBOX，不过单击鼠标时按钮会自动反转  
BS\_AUTORADIOBUTTON： 同BS\_RADIOBUTTON，不过单击鼠标时按钮会自动反转  
BS\_AUTO3STATE ：同BS\_3STATE，不过单击按钮时会改变状态  
BS\_CHECKBOX：指定在矩形按钮右侧带有标题的选择框  
BS\_DEFPUSHBUTTON：指定默认的命令按钮，这种按钮的周围有一个黑框，用户可以按回车键来快速选择该按钮  
BS\_GROUPBOX：指定一个组框  
BS\_LEFTTEXT：使控件的标题显示在按钮的左边  
BS\_OWNERDRAW：指定一个自绘式按钮  
BS\_PUSHBUTTON：指定一个命令按钮  
BS\_RADIOBUTTON：指定一个单选按钮，在圆按钮的右边显示正文  
BS\_3STATE：同BS\_CHECKBOX，不过控件有3 种状态—选择、未选择和变灰  
当然，除了以上列出的风格，一般还会为按钮设置WS\_CHILD、WS\_VISIBLE和WS\_TABSTOP等风格，WS\_TABSTOP风格使按钮控件具有tab停止属性，即按tab键切换焦点控件时能够将焦点停在按钮控件上。创建一组单选按钮时，第一个按钮的风格应设置为WS\_CHILD|WS\_VISIBLE|WS\_TABSTOP|WS\_GROUP|BS\_AUTORADIOBUTTON，其他单选按钮的风格应为WS\_CHILD|WS\_VISIBLE|BS\_AUTORADIOBUTTON，不包含WS\_TABSTOP和WS\_GROUP。  
在对话框模板上直接添加按钮控件时，它的属性中包含了上述风格，例如，复选框的Tri\_state属性实际上代表的就是BS\_3STATE风格。  
剩下的三个参数与静态文本框的Create函数中的相应参数类似，大家可以参考前面静态文本框的讲解，也可以查阅[MSDN](http://www.jizhuomi.com/software/139.html)。

CButton类的主要成员函数  
下面是CButton类的一些主要的成员函数，至于其他的函数大家可以在MSDN中查看。  
HBITMAP SetBitmap(HBITMAP hBitmap);  
设置要在按钮中显示的位图。参数hBitmap为位图的句柄。返回值为按钮原来位图的句柄。  
HBITMAP GetBitmap( ) const;  
获取之前由SetBitmap函数设置的按钮位图的句柄。  
void SetButtonStyle(UINT nStyle,BOOL bRedraw = TRUE);  
设置按钮的风格。参数nStyle指定按钮的风格，bRedraw指定按钮是否重绘，为TRUE则重绘，否则不重绘，默认为重绘。  
UINT GetButtonStyle( ) const;  
获取按钮控件的风格。  
void SetCheck(int nCheck);  
设置按钮的选择状态。参数nCheck为0表示未选中状态，1表示选中状态，2表示不确定状态（仅用于复选框）。  
int GetCheck( ) const;  
获取按钮的选择状态。返回值的意义同SetCheck函数的nCheck参数。  
HCURSOR SetCursor(HCURSOR hCursor);  
设置要显示到按钮上的光标图。参数hCursor指定了光标的句柄。返回值为按钮原来光标的句柄。  
HCURSOR GetCursor( );  
获取之前由SetCursor设置的光标的句柄。  
HICON SetIcon(HICON hIcon);  
设置要在按钮上显示的图标。参数hIcon指定了图标的句柄。返回值为按钮原来图标的句柄。  
HICON GetIcon( ) const;  
获取之前由SetIcon设置的图标的句柄。  
void SetState(BOOL bHighlight);  
设置按钮的高亮状态。参数bHighlight指定按钮是否高亮显示，非0则高亮显示，否则取消高亮显示状态。  
UINT GetState( ) const;

       获取按钮控件的选择状态、高亮状态和焦点状态。我们可以通过将返回值与各个掩码相与来获得各种状态值，掩码与对应的相与结果说明如下：  
掩码0x0003：用来获取单选按钮或复选框的状态。相与结果为0表示未选中，1表示被选中，2表示不确定状态（仅用于复选框）。  
掩码0x0004：用来判断按钮是否是高亮显示。相与结果为非0值表示按钮是高亮显示的。当单击按钮并按住鼠标左键时，按钮会呈高亮显示。  
掩码0x0008：相与结果为非零值表示按钮拥有输入焦点。

下面再列出几个继承自CWnd类的成员函数，通过它们获取或设置按钮控件的状态非常方便，只需要知道按钮的ID。  
void CheckDlgButton(int nIDButton,UINT nCheck);  
用来设置按钮的选择状态。参数nIDButton指定了按钮的ID。nCheck的值为0表示按钮未被选择，为1表示按钮被选择，为2表示按钮处于不确定状态（仅用于复选框）。  
UINT IsDlgButtonChecked(int nIDButton) const;  
返回复选框或单选按钮的选择状态。返回值为0表示按钮未被选择，为1表示按钮被选择，为2表示按钮处于不确定状态（仅用于复选框）。  
void CheckRadioButton(int nIDFirstButton,int nIDLastButton,int nIDCheckButton);  
用来选择组中的一个单选按钮。参数nIDFirstButton指定了组中第一个按钮的ID，nIDLastButton指定了组中最后一个按钮的ID，nIDCheckButton指定了要选择的按钮的ID。  
int GetCheckedRadioButton(int nIDFirstButton, int nIDLastButton);  
用来获得一组单选按钮中被选中按钮的ID。参数nIDFirstButton 说明了组中第一个按钮的ID，nIDLastButton 说明了组中最后一个按钮的ID。  
另外，CWnd类的成员函数GetWindowText()、SetWindowText()等也可以用来获取或设置按钮中显示的文本。

## 7.4 列表框控件ListBox

列表框控件简介  
列表框给出了一个选项清单，允许用户从中进行单项或多项选择，被选中的项会高亮显示。列表框可分为单选列表框和多选列表框，顾名思义，单选列表框中一次只能选择一个列表项，而多选列表框可以同时选择多个列表项。

列表框也会向父窗口发送通知消息。这些通知消息及含义如下：  
LBN\_DBLCLK ：用户用鼠标双击了一列表项，只有具有LBS\_NOTIFY 的列表框才能发送该消息  
LBN\_ERRSPACE ：列表框不能申请足够的动态内存来满足需要  
LBN\_KILLFOCUS ：列表框失去输入焦点  
LBN\_SELCANCEL： 当前的选择被取消，只有具有LBS\_NOTIFY 的列表框才能发送该消息  
LBN\_SELCHANGE：单击鼠标选择了一列表项，只有具有LBS\_NOTIFY 的列表框才能发送该消息  
LBN\_SETFOCUS：列表框获得输入焦点  
WM\_CHARTOITEM：当列表框收到WM\_CHAR 消息后， 向父窗口发送该消息， 只有具有LBS\_WANTKEYBOARDINPUT 风格的列表框才会发送该消息  
WM\_VKEYTOITEM：当列表框收到WM\_KEYDOWN 消息后，向父窗口发送该消息，只有具有LBS\_WANTKEYBOARDINPUT 风格的列表框才会发送该消息

列表框控件的创建

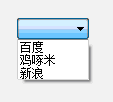
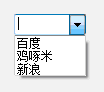
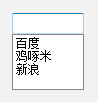
[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)将列表框控件的所有操作都封装到了CListBox类中。

        创建列表框控件时，可以在对话框模板中直接拖入列表框控件Listbox，然后添加控件变量使用。但如果需要动态创建列表框，就要用到CListBox类的Create成员函数了。Create成员函数的原型如下：  
virtual BOOL Create(  
   DWORD dwStyle,  
   const RECT& rect,  
   CWnd\* pParentWnd,  
   UINT nID   
);  
参数rect指定了列表框的位置和尺寸，pParentWnd为父窗口的指针，nID用于指定列表框控件的ID。最后重点讲讲参数dwStyle，它指定了列表框控件的风格，以下是各种风格说明：  
LBS\_EXTENDEDSEL：支持多重选择，在点击列表项时按住Shift 键或Ctrl 键即可选择多个项  
LBS\_HASSTRINGS：指定一个含有字符串的自绘式列表框  
LBS\_MULTICOLUMN：指定一个水平滚动的多列列表框， 通过调用CListBox::SetColumnWidth 来设置每列的宽度  
LBS\_MULTIPLESEL：支持多重选择。列表项的选择状态随着用户对该项单击或双击鼠标而翻转  
LBS\_NOINTEGRALHEIGHT：列表框的尺寸由应用程序而不是Windows 指定。通常，Windows指定尺寸会使列表项的某些部分隐藏起来  
LBS\_NOREDRAW：当选择发生变化时防止列表框被更新，可发送消息改变该风格  
LBS\_NOTIFY：当用户单击或双击鼠标时通知父窗口  
LBS\_OWNERDRAWFIXED：指定自绘式列表框，即由父窗口负责绘制列表框的内容，并且列表项有相同的高度  
LBS\_OWNERDRAWVARIABLE：指定自绘式列表框，并且列表项有不同的高度  
LBS\_SORT：使插入列表框中的项按升序排列  
LBS\_STANDARD：相当于指定了WS\_BORDER|WS\_VSCROLL|LBS\_SORT  
LBS\_USETABSTOPS：使列表框在显示列表项时识别并扩展制表符(‘\t’)，默认的制表宽度是32 个对话框单位  
LBS\_WANTKEYBOARDINPUT：允许列表框的父窗口接收WM\_VKEYTOITEM 和WM\_CHARTOITEM 消息，以响应键盘输入  
LBS\_DISABLENOSCROLL：使列表框在不需要滚动时显示一个禁止的垂直滚动条  
dwStyle可以是以上所列风格的组合。与其他控件一样，除了这些风格一般还要为列表框控件设置WS\_CHILD、WS\_VISIBLE、WS\_TABSTOP、WS\_BORDER、WS\_VSCROLL等风格。一般创建单选列表框时，风格要设置为：  
WS\_CHILD|WS\_VISIBLE|WS\_TABSTOP|LBS\_STANDARD，如果不希望列表框项排序显示则应去掉LBS\_STANDARD。创建多选列表框时，只需要在单选列表框风格后添加LBS\_MULTIPLESEL或LBS\_EXTENDEDSEL风格。

       对于对话框模板中直接添加的列表框控件，其属性页中的属性包含了以上风格，例如属性Multicolumn对应的就是LBS\_MULTICOLUMN风格。

CListBox类的主要成员函数  
int GetCount( ) const;  
       返回值：返回列表框中列表项的数目，如果发生错误则返回LB\_ERR。  
int GetSel(int nIndex) const;  
       参数：nIndex指定某个列表项的索引。  
       返回值：返回nIndex指定列表项的状态。如果此列表项被选择了则返回一个正值， 否则返回0，若发生错误则返回LB\_ERR。  
int SetSel(int nIndex,BOOL bSelect = TRUE);  
       此函数只用于多选列表框，使用它可以选择或取消选择指定的列表项。  
       参数：nIndex指定某个列表项的索引，若为-1则相当于指定了所有列表项。bSelect 为TRUE时选择指定列表项，否则取消选择指定列表项。  
       返回值：如果发生错误则返回LB\_ERR。  
int AddString(LPCTSTR lpszItem);  
       此函数用来向列表框中添加字符串。如果列表框指定了LBS\_SORT风格，字符串就 被以排序顺序插入到列表框中，如果没有指定LBS\_SORT风格，字符串就被添加到 列表框的结尾。  
       参数：lpszItem指定了要添加的字符串。  
       返回值：返回字符串在列表框中添加的位置。如果发生错误则返回LB\_ERR,内存不 够则返回LB\_ERRSPACE。  
int InsertString(int nIndex, LPCTSTR lpszItem);  
       该函数用来在列表框中的指定位置插入字符串。与AddString函数不同的是， InsertString函数不会导致LBS\_SORT风格的列表框重新排序。不要在具有LBS\_SORT 风格的列表框中使用InsertString函数，以免破坏列表项的次序。  
       参数：。参数nIndex 给出了插入位置（索引），如果值为-1，则字符串将被添加到 列表的末尾。参数lpszItem 指定了要插入的字符串。  
       返回值：返回实际的插入位置，若发生错误，会返回LB\_ERR 或LB\_ERRSPACE。  
int DeleteString(UINT nIndex);  
       该函数用于删除指定的列表项。  
       参数：nIndex 指定了要删除项的索引。  
       返回值：函数的返回值为剩下的列表项数目，如果nIndex 超过了实际的表项总数， 则返回LB\_ERR。  
void ResetContent();  
       该函数用于清除所有列表项。  
int GetText(int nIndex,LPTSTR lpszBuffer) const;  
void GetText(int nIndex,CString& rString) const;  
       这两个成员函数用于获取指定列表项的字符串。参数nIndex 指定了列表项的索引。 参数lpszBuffer 指向一个接收字符串的缓冲区。引用参数rString 则指定了接收字 符串的CString对象。第一个版本的函数会返回获得的字符串的长度，若出错，则 返回LB\_ERR；第二个版本的函数则不会。  
int GetTextLen(int nIndex) const;  
       该函数返回指定列表项的字符串的字节长度。  
       参数：nIndex 指定了列表项的索引。  
       返回值：若出错则返回LB\_ERR。  
int GetCurSel() const;  
       该函数仅适用于单选列表框，用来返回当前被选择项的索引，如果没有列表项被选 择或有错误发生，则函数返回LB\_ERR。  
int SetCurSel(int nSelect);  
       该函数仅适用于单选列表框，用来选择指定的列表项。该函数会滚动列表框以使选 择项可见。参数nIndex 指定了列表项的索引，若为-1，那么将清除列表框中的选 择。若出错函数返回LB\_ERR。  
int GetSelCount() const;  
       该函数仅用于多重选择列表框，它返回选择项的数目，若出错函数返回LB\_ERR。  
int FindString(int nStartAfter,LPCTSTR lpszItem) const;  
       该函数用于对列表项进行与大小写无关的搜索。参数nStartAfter 指定了开始搜索 的位置，合理指定nStartAfter 可以加快搜索速度，若nStartAfter 为-1，则从头开 始搜索整个列表。参数lpszItem 指定了要搜索的字符串。函数返回与lpszItem 指 定的字符串相匹配的列表项的索引，若没有找到匹配项或发生了错误，则会返回 LB\_ERR。FindString 函数先从nStartAfter指定的位置开始搜索，若没有找到匹配项， 则会从头开始搜索列表。只有找到匹配项，或对整个列表搜索完一遍后，搜索过程 才会停止，所以不必担心会漏掉要搜索的列表项。  
int SelectString(int nStartAfter,LPCTSTR lpszItem);  
       该函数仅适用于单选列表框，用来选择与指定字符串相匹配的列表项。该函数会滚 动列表框以使选择项可见。参数的意义及搜索的方法与函数FindString 类似。如果 找到了匹配的项，函数返回该项的索引，如果没有匹配的项，函数返回LB\_ERR 并 且当前的选择不被改变。

## 7.5 组合框控件Combo Box

组合框控件简介  
组合框其实就是把一个[编辑框](http://www.jizhuomi.com/software/181.html)和一个列表框组合到了一起，分为三种：简易（Simple）组合框、下拉式（Dropdown）组合框和下拉列表式（Drop List）组合框。下面讲讲它们的区别。  
简易组合框中的列表框是一直显示的，效果如下图：  
  
下拉式组合框默认不显示列表框，只有在点击了编辑框右侧的下拉箭头才会弹出列表框   
下拉列表式组合框的编辑框是不能编辑的，只能由用户在下拉列表框中选择了某项后，在编辑框中显示其文本。

组合框被操作时会向父窗口发送通知消息，这些通知消息及其含义如下：  
CBN\_CLOSEUP：组合框的列表框组件被关闭，简易组合框不会发送该通知消息  
CBN\_DBLCLK：用户在某列表项上双击鼠标，只有简易组合框才会发送该通知消息  
CBN\_DROPDOWN：组合框的列表框组件下拉，简易式组合框不会发送该通知消息  
CBN\_EDITUPDATE：在编辑框准备显示改变了的正文时发送该消息，下拉列表式组合框不会发送该消息  
CBN\_EDITCHANGE：编辑框的内容被用户改变了，与CBN\_EDITUPDATE不同，该消息是在编辑框显示的正文被刷新后才发出的，下拉列表式组合框不会发送该消息  
CBN\_ERRSPACE：组合框无法申请足够的内存来容纳列表项  
CBN\_SELENDCANCEL：表明用户的选择应该取消，当用户在列表框中选择了一项，然后又在组合框控件外单击鼠标时就会导致该消息的发送  
CBN\_SELENDOK：用户选择了一项，然后按了回车键或单击了下滚箭头，该消息表明用户确认了自己所作的选择  
CBN\_KILLFOCUS：组合框失去了输入焦点  
CBN\_SELCHANGE：用户通过单击或移动箭头键改变了列表的选择  
CBN\_SETFOCUS：组合框获得了输入焦点

 组合框控件的创建  
[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)将组合框控件的所有操作都封装到了CComboBox类中。  
我们在对话框中加入组合框时，可以往对话框模板中拖入Combo Box控件，而后添加CComboBox类型的控件变量使用，但如果我们想在程序中动态创建的话，就要使用CComboBox类的成员函数Create了。Create函数的原型如下：  
virtual BOOL Create(  
   DWORD dwStyle,  
   const RECT& rect,  
   CWnd\* pParentWnd,  
   UINT nID   
);  
大家可以看出，CComboBox类的Create成员函数同前面几个控件类的Create成员函数非常类似，dwStyle指定组合框控件的风格，rect为列表框弹出后组合框的位置和尺寸，pParentWnd是指向父窗口的指针，不能为NULL，nID指定组合框控件的ID。最后还是重点讲讲dwStyle参数。组合框控件的风格包括以下几种，并给出了相应说明：  
CBS\_AUTOHSCROLL：使编辑框组件具有水平滚动的风格  
CBS\_DISABLENOSCROLL：使列表框在不需要滚动时显示一个禁止的垂直滚动条  
CBS\_DROPDOWN：指定一个下拉式组合框  
CBS\_DROPDOWNLIST：指定一个下拉列表式组合框  
CBS\_HASSTRINGS：指定一个含有字符串的自绘式组合框  
CBS\_LOWERCASE：将编辑框和列表框中的所有文本都自动转换为小写字符  
CBS\_NOINTEGRALHEIGHT：组合框的尺寸由应用程序而不是Windows 指定，通常，由Windows指定尺寸会使列表项的某些部分隐藏起来  
CBS\_OEMCONVERT：使编辑框组件中的正文可以在ANSI 字符集和OEM字符集之间相互转换。这在编辑框中包含文件名时是很有用的  
CBS\_OWNERDRAWFIXED：指定自绘式组合框，即由父窗口负责绘制列表框的内容，并且列表项有相同的高度  
CBS\_OWNERDRAWVARIABLE：指定自绘式组合框，并且列表项有不同的高度  
CBS\_SIIMPLE：指定一个简易组合框  
CBS\_SORT：自动对列表框组件中的项进行排序  
CBS\_UPPERCASE：将编辑框和列表框中的所有文本都自动转换为大写字符  
dwStyle参数可以是以上风格的组合。跟其他控件一样，创建时一般也还要指定WS\_CHILD、WS\_VISIBLE、WS\_TABSTOP和WS\_VSCROLL等风格。  
在对话框模板中直接添加组合框控件时，其属性页中的属性包含了以上风格，例如属性Uppercase设为True就相当于指定了CBS\_UPPERCASE风格。

CComboBox类的主要成员函数  
因为组合框是由编辑框和列表框组合而成的，所以组合框的操作和编辑框与列表框的操作有很多相似之处，同样的，CComboBox类的成员函数也和CEdit类与CListBox类的成员函数有很多相似之处，不但功能相似，甚至函数名和参数也很相似。鸡啄米下面大概讲解下CComboBox类的主要成员函数，更详细的内容可以参见MSDN。  
int GetCount( ) const;  
       获取组合框控件的列表框中列表项的数量。  
int GetCurSel( ) const;  
       获取组合框控件的列表框中选中项的索引，如果没有选中任何项，该函数返回 CB\_ERR。  
int SetCurSel(int nSelect);  
       在组合框控件的列表框中选择某项。nSelect参数指定了要选择的列表项的索引，如 果为-1则列表框中当前选择项被取消选中，编辑框也被清空。  
DWORD GetEditSel( ) const;  
       获取组合框控件的编辑框中当前选择范围的起始和终止字符的位置。该函数返回一 个32位数，低16位存放起始位置，高16位存放选择范围后第一个非选择字符的 位置。如果该函数用于下拉列表式组合框时，会返回CB\_ERR。  
BOOL SetEditSel(int nStartChar,int nEndChar);  
       用于在组合框控件的编辑框中选择字符。nStartChar参数指定起始位置，nEndChar 参数指定终止位置。  
DWORD\_PTR GetItemData(int nIndex) const;  
       获取组合框中指定项所关联的32位数据。nIndex参数指定组合框控件的列表框某 项的索引（从0开始）。  
int SetItemData(int nIndex,DWORD\_PTR dwItemData);  
       为某个指定的组合框列表项设置一个关联的32位数。nIndex参数指定要进行设置 的列表项索引。dwItemData参数指定要关联的新值。  
void GetLBText(int nIndex,CString& rString) const;  
       从组合框控件的列表框中获取某项的字符串。nIndex参数指定要获取字符串的列表 项的索引，CString参数用于接收取到的字符串。  
int GetLBTextLen(int nIndex) const;  
       获取组合框控件的列表框中某项的字符串长度。nIndex参数指定要获取字符串长度 的列表项的索引。    
int GetTopIndex( ) const;  
       获取组合框控件的列表框中第一个可见项的索引。  
int SetTopIndex(int nIndex);  
       将组合框控件的列表框中某个指定项设置为可见的。nIndex参数指定了该列表项的 索引。该函数成功则返回0，有错误发生则返回CB\_ERR。  
BOOL LimitText(int nMaxChars);  
       用于限制用户在组合框控件的编辑框中能够输入的最大字节长度。nMaxChars参数 指定了用户能够输入文字的最大字节长度，如果为0则长度被限制为65535个字节。  
int AddString(LPCTSTR lpszString);  
       为组合框控件中的列表框添加新的列表项。lpszString参数是指向要添加的字符串的 指针。该函数的返回值如果大于等于0，那么它就是新列表项的索引，而如果有错 误发生则会返回CB\_ERR，如果没有足够的内存存放新字符串则返回CB\_ERRSPACE。  
int DeleteString(UINT nIndex);  
       删除组合框中某指定位置的列表项。nIndex参数指定了要删除的列表项的索引。该 函数的返回值如果大于等于0，那么它就是组合框中剩余列表项的数量。如果nIndex 指定的索引超出了列表项的数量则返回CB\_ERR。  
int FindString(int nStartAfter,LPCTSTR lpszString) const;  
       在组合框控件的列表框中查找但不选中第一个包含指定前缀的列表项。nStartAfter 参数指定了第一个要查找的列表项之前的那个列表项的索引。lpszString指向包含 要查找的前缀的字符串。该函数的返回值如果大于等于0，那么它是匹配列表项的 索引，如果查找失败则返回CB\_ERR。  
int InsertString(int nIndex,LPCTSTR lpszString);  
       向组合框控件的列表框中插入一个列表项。nIndex参数指定了要插入列表项的位置， lpszString参数则指定了要插入的字符串。该函数返回字符串被插入的位置，如果 有错误发生则会返回CB\_ERR，如果没有足够的内存存放新字符串则返回 CB\_ERRSPACE。  
int SelectString(int nStartAfter,LPCTSTR lpszString);  
       在组合框控件的列表框中查找一个字符串，如果查找到则选中它，并将其显示到编 辑框中。参数同FindString。如果字符串被查找到则返回此列表项的索引，如果查 找失败则返回CB\_ERR，并且当前选择项不改变。

此外，CComboBox类还继承了CWnd类的成员函数GetWindowText、SetWindowText等。

## 7.6 滚动条控件Scroll Bar

滚动条控件简介  
滚动条大家也很熟悉了，Windows窗口中很多都有滚动条。前面讲的[列表框](http://www.jizhuomi.com/software/186.html)和组合框设置了相应属性后，如果列表项显示不下也会出现滚动条。滚动条分为水平滚动条（Horizontal Scroll Bar）和垂直滚动条（Vertical Scroll Bar）两种。滚动条中有一个滚动块，用于标识滚动条当前滚动的位置。我们可以拖动滚动块，也可以用鼠标点击滚动条某一位置使滚动块移动。  
从滚动条的创建形式来分，有标准滚动条和滚动条控件两种。像列表框和组合框设置了WS\_HSCROLL 或WS\_VSCROLL风格以后出现的滚动条，不是一个独立的窗口，而是这些窗口的一部分，这就是标准滚动条。而滚动条控件是一个独立的窗口，它可以获得焦点，响应某些操作。

滚动条控件的创建  
[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)也为滚动条控件的操作提供了类，即为CScrollBar类。  
滚动条控件的创建依然有两种方式，一种是直接在Toolbox中将滚动条控件拖入对话框模板，然后[添加控件变量](http://www.jizhuomi.com/software/153.html)使用，另一种就是用CScrollBar类的Create成员函数动态创建。这两种方式适用于不同的场合。  
CScrollBar类的成员函数Create的函数原型如下：  
virtual BOOL Create(  
   DWORD dwStyle,  
   const RECT& rect,  
   CWnd\* pParentWnd,  
   UINT nID   
);  
此函数与其他控件类的Create函数原型基本相同。参数dwStyle指定滚动条控件的风格，rect指定滚动条控件的位置和尺寸，pParentWnd为指向滚动条控件父窗口的指针，nID指定滚动条控件的ID。下面鸡啄米简单介绍几个主要的滚动条控件风格，更加具体的可以查阅MSDN。  
SBS\_HORZ：指定滚动条为水平滚动条。如果没有指定SBS\_BOTTOMALIGN或SBS\_TOPALIGN风格，则滚动条的高度、宽度和位置由Create函数的rect参数给出。  
SBS\_VERT：指定滚动条为垂直滚动条。如果没有指定SBS\_RIGHTALIGN或SBS\_LEFTALIGN风格，则滚动条的高度、宽度和位置由Create函数的rect参数给出。  
SBS\_TOPALIGN：与SBS\_HORZ配合使用。滚动条的上边缘与Create函数的rect参数指定矩形的上边缘对齐。滚动条高度为系统滚动条的默认高度。  
SBS\_BOTTOMALIGN：与SBS\_HORZ配合使用。滚动条的下边缘与Create函数的rect参数指定矩形的下边缘对齐。滚动条高度为系统滚动条的默认高度。  
SBS\_LEFTALIGN：与SBS\_VERT配合使用。滚动条的左边缘与Create函数的rect参数指定矩形的左边缘对齐。滚动条宽度为系统滚动条的默认宽度。  
SBS\_RIGHTALIGN：与SBS\_VERT配合使用。滚动条的右边缘与Create函数的rect参数指定矩形的右边缘对齐。滚动条宽度为系统滚动条的默认宽度。  
dwStyle参数可以是以上风格中某几个的组合，另外一般也会用到WS\_CHILD、WS\_VISIBLE风格。例如，创建一个水平滚动条控件，dwStyle参数应该为WS\_CHILD|WS\_VISIBLE|SBS\_HORZ，创建垂直滚动条控件时dwStyle参数应该为WS\_CHILD|WS\_VISIBLE|SBS\_VERT。

CScrollBar类的主要成员函数  
BOOL GetScrollInfo(LPSCROLLINFO lpScrollInfo, UINT nMask = SIF\_ALL);

       获取的滚动条的参数信息，该信息为SCROLLINFO结构体的形式。参数lpScrollInfo为指向SCROLLINFO结构体变量的指针。SCROLLINFO结构体的定义如下：

1. typedef struct tagSCROLLINFO {
2. UINT cbSize;         // 结构的尺寸（字节为单位）
3. UINT fMask;          // 说明结构中的哪些参数是有效的，可以是屏蔽值的组合,如SIF\_POS|SIF\_PAGE，若为SIF\_ALL则整个结构都有效
4. int  nMin;           // 滚动范围最小值，当fMask 中包含SIF\_RANGE 时有效
5. int  nMax;           // 滚动范围最大值，当fMask 中包含SIF\_RANGE 时有效
6. UINT nPage;          // 页尺寸，用来确定比例滚动框的大小，当fMask中包含SIF\_PAGE时有效
7. int  nPos;           // 滚动框的位置，当fMask 中包含SIF\_POS 有效
8. int  nTrackPos;      // 滚动时滚动框的位置，当fMask 中包含SIF\_TRACKPOS 时有效，该参数只能查询，不能设置，最好不要用该参数来查询拖动时滚动框的位置
9. }   SCROLLINFO, \*LPSCROLLINFO;
10. typedef SCROLLINFO CONST \*LPCSCROLLINFO;    
    参数nMask 的含义与SCROLLINFO 结构体中的fMask一样。该函数在获取信息成功则返回TRUE，否则返回FALSE。  
    BOOL SetScrollInfo(LPSCROLLINFO lpScrollInfo, BOOL bRedraw = TRUE);

       用于设置滚动条的各种参数信息。参数lpScrollInfo为指向SCROLLINFO结构体变量的指针，参数bRedraw表示是否需要重绘滚动条，如果为TRUE，则重绘。该函数操作成功则返回TRUE，否则返回FALSE。  
int GetScrollPos( ) const;

       获取滚动块的当前位置。如果失败则返回0。  
int SetScrollPos(int nPos, BOOL bRedraw = TRUE);  
将滚动块移动到指定位置。参数nPos指定了滚动块的新位置，参数bRedraw 表示是否需要重绘滚动条，如果为TRUE，则重绘。函数返回滚动框原来的位置，若操作失败则返回0。  
void GetScrollRange(LPINT lpMinPos, LPINT lpMaxPos) const;  
获取滚动条的滚动范围。参数lpMinPos指向滚动条滚动范围的最小值，参数lpMaxPos指向滚动条滚动范围的最大值。  
void SetScrollRange(int nMinPos, int nMaxPos, BOOL bRedraw = TRUE);

       用于指定滚动条的滚动范围。参数nMinPos 和nMaxPos 分别指定了滚动范围的最小值和最大值，两者的差不得超过32767。当两者都为0 时，滚动条将被隐藏。参数bRedraw 表示是否需要重绘滚动条，如果为TRUE，则重绘。  
OnHScroll()与OnVScroll()函数

       无论是标准滚动条，还是滚动条控件，滚动条的通知消息都是用WM\_HSCROLL 和WM\_VSCROLL消息发送出去的。对这两个消息的默认处理函数是CWnd::OnHScroll和CWnd::OnVScroll，一般需要在派生类中对这两个函数进行重载，以实现滚动功能。也就是说，假设在一个对话框中放入了一个水平滚动条，我们可以在对话框类中重载OnHScroll函数，并在OnHScroll函数中实现滚动功能。

       这两个函数的声明如下：

       afx\_msg void OnHScroll(UINT nSBCode,UINT nPos,CScrollBar\* pScrollBar);  
       afx\_msg void OnVScroll(UINT nSBCode,UINT nPos,CScrollBar\* pScrollBar);

       参数nSBCode是通知消息码，主要通知码及含义的介绍下面已列出。nPos 是滚动框的位置，只有在nSBCode为SB\_THUMBPOSITION或SB\_THUMBTRACK时，该参数才有意义。如果通知消息是滚动条控件发来的，那么pScrollBar 是指向该控件的指针，如果是标准滚动条发来的，则pScrollBar 为NULL。  
       SB\_BOTTOM/SB\_RIGHT：滚动到底端（右端）  
       SB\_TOP/SB\_LEFT：滚动到顶端（左端）  
       SB\_LINEDOWN/SB\_LINERIGHT：向下（向右）滚动一行（列）  
       SB\_LINEUP/SB\_LINELEFT：向上（向左）滚动一行（列）  
       SB\_PAGEDOWN/SB\_PAGERIGHT：向下（向右）滚动一页  
       SB\_PAGEUP/SB\_PAGELEFT：向上（向左）滚动一页  
       SB\_THUMBPOSITION：滚动到指定位置  
       SB\_THUMBTRACK：滚动框被拖动。可利用该消息来跟踪对滚动框的拖动  
       SB\_ENDSCROLL：滚动结束

## 7.7 图片控件Picture Control

图片控件和前面讲到的[静态文本框](http://www.jizhuomi.com/software/179.html)都是静态文本控件，因此两者的使用方法有很多相同之处，所属类都是CStatic类，有关成员函数已在前面介绍，这里就不重复了。

Picture Control控件，在图片控件的属性页中有一个Type属性，Type属性下拉列表中有8种类型，下面分别介绍下：  
Frame：显示一个无填充的矩形框，边框颜色可以通过Color属性的下拉列表设定  
Etched Horz：显示一条横分割线  
Etched Vert：显示一条竖分割线  
Rectangle：显示一个填充的矩形框，矩形颜色可通过Color属性的下拉列表设定  
Icon：显示一个图标（Icon），图标通过Image 下拉列表来设置图标资源ID  
Bitmap：显示一个位图（Bitmap），位图通过Image 下拉列表来设置位图资源ID  
Enhanced Metafile：显示一个加强的元数据文件（Metafile）  
Owner Draw：自绘

图片控件可以使用静态加载的方法，即手动在资源窗口加载，也可以使用代码动态加载。

## 7.8 列表视图控件List Control

列表视图控件是对前面讲到的[列表框控件List Box](http://www.jizhuomi.com/software/186.html)的改进和延伸。列表视图控件的列表项一般有图标（Icon）和标签（Label）两部分。图标是对列表项的图形描述，标签是文字描述。当然列表项可以只包含图标也可以只包含标签。  
列表视图控件有4种风格：Icon、Small Icon、List和Report。下面简单说下4种风格各自的特点：  
Icon大图标风格：列表项的图标通常为32×32像素，在图标的下面显示标签。  
Small Icon小图标风格：列表项的图标通常为16×16像素，在图标的右面显示标签。  
List列表风格：与小图标风格类似，图标和文字的对齐方式不同。  
Report报表风格：列表视图控件可以包含一个列表头来描述各列的含义。每行显示一个列表项，通常可以包含多个列表子项。最左边的列表子项的标签左边可以添加一个图标，而它右边的所有子项则只能显示文字。这种风格的列表视图控件很适合做各种报表。

列表视图控件的通知消息  
LVN\_ITEMCHANGING 和LVN\_ITEMCHANGED：当列表视图的状态发生变化时，会发送这两个通知消息。例如，当用户选择了新的列表项时，程序就会收到这两个消息。  
消息会附带一个指向NMLISTVIEW 结构的指针，消息处理函数可从该结构中获得状态信息。两个消息的不同之处在于，前者的消息处理函数如果返回TRUE，那么就阻  
止选择的改变，如果返回FALSE，则允许改变。

       LVN\_KEYDOWN：该消息表明了一个键盘事件。消息会附带一个指向NMLVKEYDOWN结构的指针，通过该结构程序可以获得按键的信息。

       LVN\_BEGINLABELEDIT 和LVN\_ENDLABELEDIT：分别在用户开始编辑和结束编辑标题时发送。消息会附带一个指向NMLVDISPINFO结构的指针。在前者的消息处理函数中，可以调用GetEditControl成员函数返回一个指向用于编辑标题的编辑框的指针，如果处理函数返回FALSE，则允许编辑，如果返回TRUE，则禁止编辑。在后者的消息处理函数中，NMLVDISPINFO结构中的item.pszText指向编辑后的新标题，如果pszText 为NULL，那么说明用户放弃了编辑，否则，程序应负责更新表项的标题，这可以由SetItem或SetItemText函数来完成。

列表视图控件的相关结构体

       下面我们来介绍一下与列表视图控件有关的一些结构体。  
1. NMHDR结构体

1. **typedef** **struct** tagNMHDR {
2. **HWND** hwndFrom;     // 控件窗口的句柄
3. **UINT\_PTR** idFrom;   // 控件ID
4. **UINT** code;         // 控件的通知消息码
5. } NMHDR;

        此结构体在很多情况下都是其他扩充结构体的第一个元素，比如上面的NMITEMACTIVATE结构体：

1. **typedef** **struct** tagNMITEMACTIVATE {
2. NMHDR hdr;
3. **int** iItem;
4. **int** iSubItem;
5. **UINT** uNewState;
6. **UINT** uOldState;
7. **UINT** uChanged;
8. POINT ptAction;
9. **LPARAM** lParam;
10. **UINT** uKeyFlags;
11. } NMITEMACTIVATE, \*LPNMITEMACTIVATE;

2. LVITEM 结构体该结构体包含了列表视图控件中列表项或列表子项的各种属性。

1. **typedef** **struct** \_LVITEM {
2. **UINT** mask;           // 掩码位的组合（下面有对应掩码的元素都已在括号中标出掩码），表明哪些元素是有效的
3. **int** iItem;           // 列表项的索引
4. **int** iSubItem;        // 列表子项的索引
5. **UINT** state;          // 状态，下面会列出。（LVIF\_STATE）
6. **UINT** stateMask;      // 状态掩码，用来说明要获取或设置哪些状态。下面会列出
7. **LPTSTR** pszText;      // 指向列表项或列表子项的标签字符串。（LVIF\_TEXT）
8. **int** cchTextMax;      // pszText指向缓冲区的字符的个数，包括字符串结束符。（LVIF\_TEXT）
9. **int** iImage;          // 图标的索引。(LVIF\_IMAGE)
10. **LPARAM** lParam;       // 32位的附加数据。(LVIF\_PARAM)
11. **#if (\_WIN32\_IE >= 0x0300)**
12. **int** iIndent;
13. **#endif**
14. **#if (\_WIN32\_WINNT >= 0x501)**
15. **int** iGroupId;
16. **UINT** cColumns; // tile view columns
17. **PUINT** puColumns;
18. **#endif**
19. **#if (\_WIN32\_WINNT >= 0x0600)**
20. **int**\* piColFmt;
21. **int** iGroup;
22. **#endif**
23. } LVITEM, \*LPLVITEM;

        下面是state和stateMask的取值及含义：  
状态                     对应的状态掩码                   含义  
LVIS\_CUT                     同左                 列表项或列表子项被选择用来进行剪切和粘贴操作  
LVIS\_DROPHILITED      同左                 列表项或列表子项成为拖动操作的目标  
LVIS\_FOCUSED            同左                 列表项或列表子项具有输入焦点  
LVIS\_SELECTED           同左                 列表项或列表子项被选中  
3. LVCOLUMN结构体  
该结构体仅适用于Report报表式列表视图控件。在向列表控件中插入一列时需要用到此结构体。它包含了列表控件某列的各种属性。

1. **typedef** **struct** \_LVCOLUMN {
2. **UINT** mask;              // 掩码位的组合（下面有对应掩码的元素都已在括号中标出掩码），表明哪些元素是有效的
3. **int** fmt;                    // 该列的表头和列表子项的标签正文显示格式，可以是LVCFMT\_CENTER、LVCFMT\_LEFT或LVCFMT\_RIGHT。（LVCF\_FMT）
4. **int** cx;                     // 以像素为单位的列的宽度。（LVCF\_FMT）
5. **LPTSTR** pszText;    // 指向列表头标题正文的字符串。（LVCF\_TEXT）
6. **int** cchTextMax;     // pszText指向缓冲区的字符的个数，包括字符串结束符。（LVCF\_TEXT）
7. **int** iSubItem;          // 该列的索引。（LVCF\_SUBITEM）
8. **#if (\_WIN32\_IE >= 0x0300)**
9. **int** iImage;
10. **int** iOrder;
11. **#endif**
12. **#if (\_WIN32\_WINNT >= 0x0600)**
13. **int** cxMin;
14. **int** cxDefault;
15. **int** cxIdeal;
16. **#endif**
17. } LVCOLUMN, \*LPLVCOLUMN;

4. NMLISTVIEW结构体  
该结构体存放了列表视图控件通知消息的相关信息。列表视图控件的大部分通知消息都会附带指向该结构体的指针。

1. **typedef** **struct** tagNMLISTVIEW {
2. NMHDR hdr;       // 标准的NMHDR 结构
3. **int** iItem;       // 列表项的索引
4. **int** iSubItem;    // 列表子项的索引
5. **UINT** uNewState;  // 列表项或列表子项的新状态
6. **UINT** uOldState;  // 列表项或列表子项原来的状态
7. **UINT** uChanged;   // 取值与LVITEM的mask成员相同，用来表明哪些状态发生了变化
8. POINT ptAction;  // 事件发生时鼠标的客户区坐标
9. **LPARAM** lParam;   //32位的附加数据
10. } NMLISTVIEW, \*LPNMLISTVIEW;

列表视图控件的创建  
[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)同样为列表视图控件的操作提供了CListCtrl类。  
如果我们不想在对话框模板中直接拖入List Control来使用列表视图控件，而是希望动态创建它，则要用到CListCtrl类的成员函数Create函数，原型如下：  
virtual BOOL Create(  
   DWORD dwStyle,  
   const RECT& rect,  
   CWnd\* pParentWnd,  
   UINT nID   
);  
参数rect为列表视图控件的位置和尺寸，pParentWnd为指向父窗口的指针，nID指定列表视图控件的ID，最复杂的一个参数同样还是dwStyle，它用于设定列表视图控件的风格，可以是以下风格的组合：  
风格                                                含义  
LVS\_ALIGNLEFT                        显示格式是大图标或小图标时，标签放在图标的左边  
LVS\_ALIGNTOP                         显示格式是大图标或小图标时，标题放在图标的上边  
LVS\_AUTOARRANGE                显示格式是大图标或小图标时，自动排列控件中的列表项  
LVS\_EDITLABELS                      用户可以修改标签文本  
LVS\_ICON                                  指定大图标显示格式  
LVS\_LIST                                   指定列表显示格式  
LVS\_NOCOLUMNHEADER        在报表格式中不显示列的表头  
LVS\_NOLABELWRAP             显示格式是大图标时，使标签文本单行显示。默认是多行显示  
LVS\_NOSCROLL                       列表视图控件无滚动条，此风格不能与LVS\_LIST或LVS\_REPORT组合使用  
LVS\_NOSORTHEADER              报表格式的列表视图控件的表头不能作为排序按钮使用  
LVS\_OWNERDRAWFIXED        由控件的拥有者负责绘制表项  
LVS\_REPORT                           指定报表显示格式  
LVS\_SHAREIMAGELISTS           使列表视图共享图像序列  
LVS\_SHOWSELALWAYS           即使控件失去输入焦点，仍显示出项的选择状态  
LVS\_SINGLESEL                       指定只能有一个列表项被选中。默认时可以多项选择  
LVS\_SMALLICON                      指定小图标显示格式  
LVS\_SORTASCENDING             按升序排列列表项  
LVS\_SORTDESCENDING          按降序排列列表项  
与前面的控件一样，除了以上风格一般我们还要为列表视图控件设置WS\_CHILD和WS\_VISIBLE风格。对于直接在对话框模板中创建的列表视图控件，其属性页中的属性与上述风格是对应的，例如，属性Alignment默认为Left，也就等价于指定了LVS\_ALIGNLEFT风格。

CListCtrl类的主要成员函数  
CListCtrl类有很多成员函数，鸡啄米这里就为大家介绍几个常用的主要成员函数。  
UINT GetSelectedCount( ) const;  
该函数返回列表视图控件中被选择列表项的数量。  
POSITION GetFirstSelectedItemPosition( ) const;

       获取列表视图控件中第一个被选择项的位置。返回的POSITION值可以用来迭代来获取其他选择项，可以当作参数传入下面的GetNextSelectedItem函数来获得选择项的索引。如果没有被选择项则返回NULL。  
int GetNextSelectedItem(POSITION& pos) const;

       该函数获取由pos指定的列表项的索引，然后将pos设置为下一个位置的POSITION值。参数pos为之前调用GetNextSelectedItem或GetFirstSelectedItemPosition得到的POSITION值的引用。返回值就是pos指定列表项的索引。  
int GetItemCount( ) const;

       获取列表视图控件中列表项的数量。  
int InsertColumn(int nCol,const LVCOLUMN\* pColumn );  
       int InsertColumn(int nCol,LPCTSTR lpszColumnHeading,int nFormat = LVCFMT\_LEFT,int nWidth = -1,int nSubItem = -1 );

       这两个函数用于在报表式列表视图控件中插入列。第一个函数中，nCol参数为插入列的索引，pColumn参数指向LVCOLUMN结构，其中包含了插入列的属性。第二个函数中，nCol参数也是插入列的索引，lpszColumnHeading参数为列标题字符串，nFormat参数为列中文本的对齐方式，可以是LVCFMT\_LEFT、LVCFMT\_RIGHT或LVCFMT\_CENTER，nWidth参数为列宽，nSubItem为插入列对应列表子项的索引。两个函数在成功时都返回新列的索引，失败都返回-1。  
BOOL DeleteColumn(int nCol);

       该函数用于删除列表视图控件中的某列。参数nCol为删除列的索引。删除成功则返回TRUE，失败返回FALSE。  
int InsertItem(int nItem,LPCTSTR lpszItem);

       向列表视图控件中插入新的列表项。参数nItem为要插入项的索引，参数lpszItem为要插入项的标签字符串。如果插入成功则返回新列表项的索引，否则返回-1。  
BOOL DeleteItem(int nItem);

       从列表视图控件中删除某个列表项。参数nItem指定了要删除的列表项的索引。删除成功则返回TRUE，否则返回FALSE。  
CString GetItemText(int nItem,int nSubItem) const;

       获取指定列表项或列表子项的显示文本。参数nItem指定了列表项的索引，参数nSubItem指定了列表子项的索引。   
BOOL SetItemText(int nItem,int nSubItem,LPCTSTR lpszText);

       设置指定列表项或列表子项的显示文本。参数nItem和nSubItem同GetItemText。参数lpszText为要设置的显示文本字符串。如果设置成功则返回TRUE，否则返回FALSE。  
DWORD\_PTR GetItemData(int nItem) const;  
该函数用于获取指定列表项的附加32位数据。参数nItem为列表项的索引。返回值就是由nItem指定列表项的附加32位数据。  
BOOL SetItemData(int nItem,DWORD\_PTR dwData);

       该函数用于为指定列表项设置附加32位是数据。参数nItem为列表项的索引，参数dwData为列表项的附加32位数据。

## 7.9 树形控件 Tree Control

树形控件在Windows系统中是很常见的，例如资源管理器左侧的窗口中就有用来显示目录的树形视图。树形视图中以分层结构显示数据，每层的缩进不同，层次越低缩进越多。树形控件的节点一般都由标签和图标两部分组成，图标用来抽象的描述数据，能够使树形控件的层次关系更加清晰。

树形控件的[通知消息](http://www.jizhuomi.com/software/147.html)  
下面列出树形控件特有的通知消息中比较常用的几个：  
TVN\_SELCHANGING和TVN\_SELCHANGED：在用户改变了对树节点的选择时，控件会发送这两个消息。消息会附带一个指向NMTREEVIEW结构的指针，程序可从该结构中获得必要的信息。两个消息都会在该结构的itemOld成员中包含原来的选择项信息，在itemNew成员中包含新选择项的信息，在action成员中表明是用户的什么行为触发了该通知消息(若是TVC\_BYKEYBOARD则表明是键盘，若是TVC\_BYMOUSE则表明是鼠标，若是TVC\_UNKNOWN则表示未知）。两个消息的不同之处在于，如果TVN\_SELCHANGING的消息处理函数返回TRUE，那么就阻止选择的改变，如果返回FALSE，则允许改变。  
TVN\_KEYDOWN：该消息表明了一个键盘事件。消息会附带一个指向NMTVKEYDOWN结构的指针，通过该结构程序可以获得按键的信息。  
TVN\_BEGINLABELEDIT和TVN\_ENDLABELEDIT：分别在用户开始编辑和结束编辑节点的标签时发送。消息会附带一个指向NMTVDISPINFO结构的指针，程序可从该结构中获得必要的信息。在前者的消息处理函数中，可以调用GetEditControl()成员函数返回一个指向用于编辑标题的编辑框的指针。如果处理函数返回FALSE，则允许编辑，如果返回TRUE，则禁止编辑。在后者的消息处理函数中，NMTVDISPINFO结构中的item.pszText指向编辑后的新标题，如果pszText为NULL，那么说明用户放弃了编辑，否则，程序应负责更新节点的标签，这可以由SetItem()或SetItemText()函数来完成。

树形控件的相关数据结构  
1. HTREEITEM句柄

       树形控件中的每个节点都可以由一个HTREEITEM类型的句柄表示。我们通过CTreeCtrl类的成员函数对树进行访问和操作时，很多时候都要用到HTREEITEM句柄。  
2. TVITEM结构体  
TVITEM结构体描述了树形控件节点的属性，定义如下：

1. **typedef** **struct** tagTVITEM {
2. **UINT** mask;       // 包含一些掩码位（下面的括号中列出）的组合，用来表明结构的哪些成员是有效的
3. HTREEITEM hItem; // 树节点的句柄(TVIF\_HANDLE)
4. **UINT** state;      // 树节点的状态(TVIF\_STATE)
5. **UINT** stateMask;  // 状态的掩码组合(TVIF\_STATE)
6. **LPTSTR** pszText;  // 树节点的标签文本(TVIF\_TEXT)
7. **int** cchTextMax;  // 标签文本缓冲区的大小(TVIF\_TEXT)
8. **int** iImage;      // 树节点的图像索引(TVIF\_IMAGE)
9. **int** iSelectedImage;  // 选中项的图像索引(TVIF\_SELECTEDIMAGE)
10. **int** cChildren;   // 表明节点是否有子节点，为1则有，为0则没有(TVIF\_CHILDREN)
11. **LPARAM** lParam;   // 一个32 位的附加数据(TVIF\_PARAM)
12. } TVITEM, \*LPTVITEM;

       此结构体中多个元素涉及到了图像和状态等，有必要具体解释下。  
树形控件节点需要显示图标时，就要为树形控件关联一个图像序列，上面的iImage成员就代表了该结构体对应的树节点的图标在图像序列中的索引，iSelectedImage则代表该树节点被选中时显示的图标在图像序列中的索引。对于如何为树形控件关联图像序列，鸡啄米将在后面的实例中讲到。  
stateMask用来说明要获取或设置树节点的哪些状态。下面是state和stateMask的一些常用值及含义：  
state                            对应的stateMask                       含义           
TVIS\_CUT                          TVIS\_CUT                        节点被选择用来进行剪切和粘贴操作  
TVIS\_DROPHILITED           TVIS\_DROPHILITED         节点成为拖动操作的目标  
TVIS\_EXPANDED               TVIS\_EXPANDED              节点的子节点被展开  
TVIS\_EXPANDEDONCE      TVIS\_EXPANDEDONCE     节点的子节点曾经被展开过  
TVIS\_SELECTED                TVIS\_SELECTED               节点被选中  
lParam在实际开发中常用来存放与树节点有关的附加数据。  
3. NMTREEVIEW结构体  
NMTREEVIEW结构体中包含了树形控件通知消息的相关信息。树形控件的大多数通知消息都会带有指向该结构体的指针。NMTREEVIEW结构体的定义如下：

1. **typedef** **struct** tagNMTREEVIEW {
2. NMHDR hdr;      // 标准的NMHDR结构
3. **UINT** action;    // 表明是用户的什么行为触发了该通知消息
4. TVITEM itemOld; // 原节点的属性
5. TVITEM itemNew; // 新节点的属性
6. POINT ptDrag;   // 事件发生时鼠标的客户区坐标
7. } NMTREEVIEW, \*LPNMTREEVIEW;

       4. TVINSERTSTRUCT结构体  
向树形控件中插入新节点时需要用到TVINSERTSTRUCT结构体，它常与TVM\_INSERTITEM消息一起使用。定义如下：

1. **typedef** **struct** tagTVINSERTSTRUCT {
2. HTREEITEM hParent;      // 父节点的句柄
3. HTREEITEM hInsertAfter; // 指明插入到同层中哪一项的后面
4. **#if (\_WIN32\_IE >= 0x0400)**
5. **union**
6. {
7. TVITEMEX itemex;
8. TVITEM item;
9. } DUMMYUNIONNAME;
10. **#else**
11. TVITEM item;            // 要添加的新节点的属性
12. **#endif**
13. } TVINSERTSTRUCT, \*LPTVINSERTSTRUCT;

       若hParent成员为TVI\_ROOT或NULL，那么新节点将被作为树的根节点插入。hInsertAfter除了可以是某个节点的句柄，还可以有四种取值：TVI\_FIRST（插入到树形控件的最前面）、TVI\_LAST（插入到树形控件的最后面）、TVI\_ROOT（作为根节点插入）和TVI\_SORT（按字母顺序插入）。  
5. NMTVDISPINFO结构体  
NMTVDISPINFO结构体中包含了与树节点的显示有关的信息。定义如下：

1. **typedef** **struct** tagNMTVDISPINFO {
2. NMHDR hdr;
3. TVITEM item;
4. } NMTVDISPINFO, \*LPNMTVDISPINFO;

 树形控件的创建  
[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)为树形控件提供了CTreeCtrl类，它封装了树形控件的所有操作。

       树形控件的创建也是有两种方式，一种是在对话框模板中直接拖入Tree Control控件创建，另一种就是通过CTreeCtrl类的Create成员函数创建。下面主要讲后者。  
CTreeCtrl类的Create成员函数的原型如下：  
       virtual BOOL Create(  
              DWORD dwStyle,  
              const RECT& rect,  
              CWnd\* pParentWnd,  
              UINT nID   
       );

       此函数的原型与前面讲到的所有控件类的Create函数都类似。dwStyle指定树形控件风格的组合，rect指定树形控件窗口的位置和大小，pParentWnd为指向树形控件父窗口的指针，nID指定树形控件的ID。下面还是主要讲讲树形控件的主要风格以及含义。

       TVS\_DISABLEDRAGDROP：禁止树形控件发送TVN\_BEGINDRAG通知消息，即不支持拖动操作  
TVS\_EDITLABELS：用户可以编辑节点的标签文本  
TVS\_HASBUTTONS：显示带有＂+＂或＂-＂的小方框来表示某项能否被展开或已展开  
TVS\_HASLINES：在父节点与子节点间连线以更清晰地显示树的结构  
TVS\_LINESATROOT：在根节点处连线  
TVS\_SHOWSELALWAYS：即使控件失去输入焦点，仍显示出项的选择状态

       同样，动态创建树形控件时，除了能够指定上述风格的组合外，一般还要指定WS\_CHILD和WS\_VISIBLE风格。

       在[对话框模板](http://www.jizhuomi.com/software/149.html)中直接拖入Tree Control创建树形控件时，可以在树形控件的属性页中设置其风格，与上面的风格是对应的，例如，属性Has Lines对应的就是TVS\_HASLINES风格。

CTreeCtrl类的主要成员函数

**CImageList\* SetImageList(CImageList \* pImageList,int nImageListType);**

       如果树节点需要显示图标时，则必须先创建一个CImageList类的对象，并为其添加多个图像组成一个图像序列，然后调用SetImageList函数为树形控件设置图像序列，在用InsertItem插入节点时传入所需图像在图像序列中的索引即可。后面的例子中会演示。参数pImageList为指向图像序列类CImageList的对象的指针，若为NULL则删除树形控件的所有图像。参数nImageListType指定图像序列的类型，可以是TVSIL\_NORMAL（普通图像序列）或TVSIL\_STATE（状态图像序列，用图像表示节点的状态）。

**UINT GetCount( ) const;**

       获取树形控件中节点的数量。

**DWORD\_PTR GetItemData(HTREEITEM hItem) const;**

       获取树形控件中某个指定节点的附加32位数据。参数hItem为指定的树节点的句柄。

**BOOL SetItemData(HTREEITEM hItem,DWORD\_PTR dwData);**

       为树形控件中某个指定节点设置附加的32位数据。参数hItem同上，dwData为要设置的32位数据。

**CString GetItemText(HTREEITEM hItem) const;**

       获取树形控件中某个指定节点的标签文本。参数hItem同上。返回值是包含标签文本的字符串。

**BOOL SetItemText(HTREEITEM hItem,LPCTSTR lpszItem);**

       为树形控件中某个指定节点设置标签文本。参数hItem同上，lpszItem为包含标签文本的字符串的指针。

**HTREEITEM GetNextSiblingItem(HTREEITEM hItem) const;**

       获取树形控件中某个指定节点的下一个兄弟节点。参数hItem同上。返回值是下一个兄弟节点的句柄。

**HTREEITEM GetPrevSiblingItem(HTREEITEM hItem) const;**

       获取树形控件中某个指定节点的上一个兄弟节点。参数hItem同上。返回值是上一个兄弟节点的句柄。

**HTREEITEM GetParentItem(HTREEITEM hItem) const;**

       获取树形控件中某个指定节点的父节点。参数hItem同上。返回值是父节点的句柄。

**HTREEITEM GetRootItem( ) const;**

       获取树形控件根节点的句柄。

**HTREEITEM GetSelectedItem( ) const;**

       获取树形控件当前选中节点的句柄。

**BOOL DeleteAllItems( );**

       删除树形控件中的所有节点。删除成功则返回TRUE，否则返回FALSE。

**BOOL DeleteItem(HTREEITEM hItem);**

       删除树形控件中的某个节点。参数hItem为要删除的节点的句柄。删除成功则返回TRUE，否则返回FALSE。

**HTREEITEM InsertItem(LPCTSTR lpszItem,int nImage,int nSelectedImage,HTREEITEM hParent = TVI\_ROOT,HTREEITEM hInsertAfter = TVI\_LAST);**

       在树形控件中插入一个新节点。参数lpszItem为新节点的标签文本字符串的指针，参数nImage为新节点的图标在树形控件图像序列中的索引，参数nSelectedImage为新节点被选中时的图标在图像序列中的索引，参数hParent为插入节点的父节点的句柄，参数hInsertAfter为新节点的前一个节点的句柄，即新节点将被插入到hInsertAfter节点之后。

**BOOL SelectItem(HTREEITEM hItem);**

       选中指定的树节点。参数hItem为要选择的节点的句柄。若成功则返回TRUE，否则返回FALSE。

## 7.10 标签控件Tab Control

标签控件也比较常见。它可以把多个页面集成到一个窗口中，每个页面对应一个标签，用户点击某个标签时，它对应的页面就会显示。 标签控件相当于是一个页面的容器，可以容纳多个对话框，而且一般也只容纳对话框，所以我们不能直接在标签控件上添加其他控件，必须先将其他控件放到对话框中，再将对话框添加到标签控件中。最终我们点击标签切换页面时，切换的不是控件的组合，而是对话框。

标签控件的通知消息

       在对标签控件进行一些操作，比如点击标签时，标签控件也会向父窗口发送一些通知消息。我们可以为这些通知消息[添加处理函数](http://www.jizhuomi.com/software/156.html)，实现各种功能。标签控件的主要通知消息及含义如下所示：  
TCN\_SELCHANGE：通知父窗口控件的标签选择项已经改变  
TCN\_SELCHANGING 通知父窗口控件的标签选择项正在改变  
TCN\_KEYDOWN：通知父窗口在控件范围内键盘被按下  
TCN\_GETOBJECT：具有TCS\_EX\_REGISTERDROP扩展特性并且对象被拖动时的通知消 息  
TCN\_FOCUSCHANGE：通知父窗口控件的按钮聚焦已经改变  
NM\_CLICK：通知父窗口用户在控件区域范围内点击了鼠标左键  
NM\_RCLICK：通知父窗口用户在控件区域范围内点击了鼠标右键  
NM\_RELEASEDCAPTURE：通知父窗口在控件区域范围内释放鼠标捕获消息

标签控件的相关结构体

       标签控件在使用中也有一些相关的结构体经常用到，主要以下几个：  
1. TCITEMHEADER结构体  
该结构体用来指定或获取标签控件本身的属性。用在TCM\_INSERTITEM、TCM\_GETITEM和TCM\_SETITEM消息中。

1. **typedef** **struct** tagTCITEMHEADER {
2. **UINT** mask;   // 掩码，可以为TCIF\_IMAGE（iImage成员有效）、TCIF\_RTLREADING、TCIF\_TEXT（pszText成员有效）
3. **UINT** lpReserved1;   // 预留
4. **UINT** lpReserved2;   // 预留
5. **LPTSTR** pszText;     // 标签文本字符串
6. **int** cchTextMax;
7. **int** iImage;         // 图标在标签控件图像序列中的索引
8. } TCITEMHEADER, \*LPTCITEMHEADER;

       2. TCITEM结构体  
该结构体用来指定或获取标签页的属性。用在TCM\_INSERTITEM、TCM\_GETITEM和TCM\_SETITEM消息中。

1. **typedef** **struct** tagTCITEM {
2. **UINT** mask;  // 掩码，可以是TCIF\_IMAGE（iImage成员有效）、TCIF\_PARAM（lParam成员有效）、TCIF\_RTLREADING、TCIF\_STATE、TCIF\_TEXT（pszText成员有效）
3. **#if (\_WIN32\_IE >= 0x0300)**
4. **DWORD** dwState;
5. **DWORD** dwStateMask;
6. **#else**
7. **UINT** lpReserved1;
8. **UINT** lpReserved2;
9. **#endif**
10. **LPTSTR** pszText;
11. **int** cchTextMax;
12. **int** iImage;
13. **LPARAM** lParam;     // 与标签页关联的32位数据
14. } TCITEM, \*LPTCITEM;

       3. TCHITTESTINFO结构体  
该结构体包含了鼠标单击测试的信息。

1. **typedef** **struct** tagTCHITTESTINFO {
2. POINT pt;  // 鼠标点击测试的客户区坐标
3. **UINT** flags; // 接收点击测试的结果。有以下几种：TCHT\_NOWHERE（坐标点不在标签上）、TCHT\_ONITEM（坐标点在标签上但不在标签文本或图标上）、TCHT\_ONITEMICON（坐标点在标签图标上）、TCHT\_ONITEMLABEL（坐标点在标签文本上）
4. } TCHITTESTINFO, \*LPTCHITTESTINFO;

4. NMTCKEYDOWN结构体该结构体包含了标签控件中键盘按下的相关信息。主要用在TCN\_KEYDOWN通知消息中。

1. **typedef** **struct** tagNMTCKEYDOWN {
2. NMHDR hdr;
3. **WORD** wVKey;
4. **UINT** flags;
5. } NMTCKEYDOWN;

标签控件的创建

[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)为标签控件的操作提供了CTabCtrl类。

       与之前的控件类似，创建标签控件可以在对话框模板中直接拖入Tab Control，也可以使用CTabCtrl类的Create成员函数创建。Create函数的原型如下：  
virtual BOOL Create(  
  DWORD dwStyle,  
  const RECT& rect,  
  CWnd\* pParentWnd,  
  UINT nID   
);

       参数dwStyle为标签控件的风格，rect为标签控件的位置和大小，pParentWnd为指向标签控件父窗口的指针，nID指定标签控件的ID。这里还是要具体说下dwStyle，下面列出了几种主要的控件风格：

       TCS\_BUTTONS：标签（控件上部用来选择标签页的位置）外观为按钮风格，且整个控件周围没有边框。

       TCS\_FIXEDWIDTH ：所有标签具有相同的宽度。

       TCS\_MULTILINE：标签以多行显示，如果需要，可以显示所有标签。

       TCS\_SINGLELINE：只显示一行标签，用户可以滚动着看其他标签。

       TCS\_TABS：标签以普通标签样式显示，且整个控件周围有边框。

       如果想了解标签控件的所有风格，可以查阅MSDN。

CTabCtrl类的主要成员函数

**int GetCurSel( ) const;**

       获取标签控件中当前选择标签的索引。如果成功则返回选择标签的索引，否则返回-1。

**BOOL GetItem(int nItem,TCITEM\* pTabCtrlItem) const;**

       获取标签控件中某个标签的信息。参数nItem为标签索引，pTabCtrlItem为指向TCITEM结构体的指针，用来接收标签信息。若获取成功返回TRUE，否则返回FALSE。

**int GetItemCount( ) const;**

       获取标签控件中标签的数量。

**int SetCurSel(int nItem);**

       在标签控件中选择某标签。参数nItem为要选择的标签的索引。如果成功则返回之前选择标签的索引，否则返回-1。

**BOOL SetItem(int nItem,TCITEM\* pTabCtrlItem);**

       设置某标签的所有或部分属性。参数nItem为标签的索引，pTabCtrlItem为指向TCITEM结构体的指针，包含了新的标签属性。成功则返回TRUE，否则返回FALSE。

**BOOL DeleteAllItems( );**

       删除标签控件中所有标签。

**BOOL DeleteItem(int nItem);**

       删除标签控件中的某个标签。参数nItem为要删除标签的索引。

**LONG InsertItem(int nItem,LPCTSTR lpszItem);**

       在标签控件中插入新的标签。参数nItem为新标签的索引，lpszItem为标签文本字符串。如果插入成功则返回新标签的索引，否则返回-1。

# 八、菜单，工具栏与状态栏

## 8.1 VS2010菜单资源详解

菜单简介  
菜单可以分为下拉式菜单和弹出式菜单。

       下拉式菜单一般在窗口标题栏下面显示，它的运行结果窗口的标题栏下就是下拉式菜单。下拉式菜单通常是由主菜单栏、子菜单及子菜单中的菜单项和分隔条所组成的。弹出式菜单一般可以通过单击鼠标右键等操作显示。它的主菜单不可见，只显示子菜单。

热键定义  
那么热键是如何定义的呢？我们可以看下“File”菜单项的属性，Caption为“&File”，很明显，只要在要定义为热键的字母前加&就可以了。

快捷键定义和功能实现  
 快捷键如何定义？我们再来看看“Open”菜单项的Caption属性，为“&Open...\tCtrl+O”，这里的\t表示在显示前面的文本后跳格再显示快捷键Ctrl+O，但这样设置其Caption属性只是能显示出快捷键，要实现快捷键的功能还需要在Accelerator资源中设定。资源视图中展开Example34.rc->Accelerator，双击打开下面的IDR\_MAINFRAME，即可添加快捷键的功能。

 CMenu类的主要成员函数（VS2008起提供CMFCMenuBar，用法类似，功能更强）  
[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)为菜单的操作提供了CMenu类，下面鸡啄米就常用的几个成员函数进行简单的介绍。

**BOOL LoadMenu(UINT nIDResource);**

       加载菜单资源，并将其附加到CMenu对象上。参数nIDResource指定了要加载的菜单资源的ID。如果菜单加载成功则返回TRUE，否则返回FALSE。

**BOOL DeleteMenu(UINT nPosition,UINT nFlags);**

       在菜单中删除一个菜单项。参数nPosition指定要删除的菜单项。参数nFlags就用来解释nPosition的意义，为MF\_BYCOMMAND时说明nPosition表示菜单项的ID，为MF\_BYPOSITION时说明nPosition表示菜单项的位置，第一个菜单项的位置为0。如果删除菜单项成功则返回TRUE，否则返回FALSE。

**BOOL TrackPopupMenu(UINT nFlags,int x,int y,CWnd\* pWnd,LPCRECT lpRect = 0);**

       用来在指定位置显示一个浮动的弹出式菜单。参数nFlags指定屏幕坐标和鼠标位置的标志，可以是以下取值：  
TPM\_CENTERALIGN：菜单在水平方向上相对于参数x指定的坐标值居中显示  
TPM\_LEFTALIGN：菜单的左侧与参数x指定的坐标值对齐  
TPM\_RIGHTALIGN：菜单的右侧与参数x指定的坐标值对齐  
TPM\_BOTTOMALIGN：菜单的底部与参数y指定的坐标值对齐  
TPM\_TOPALIGN：菜单项的顶部与参数y指定的坐标值对齐  
TPM\_VCENTERALIGN：菜单在垂直方向上相对于参数y指定的坐标值居中显示

       这里先介绍这几个比较常用的，其他可参见MSDN。参数x指定弹出式菜单的水平方向的屏幕坐标，参数y指定菜单顶部垂直方向上的屏幕坐标，参数pWnd指明哪个窗口拥有此弹出式菜单，不能为NULL，参数lpRect忽略。

**UINT CheckMenuItem(UINT nIDCheckItem,UINT nCheck);**

       在弹出菜单中为菜单项增加选中标记或移除选中标记。参数nIDCheckItem指定要选中或取消选中的菜单项。参数nCheck指定菜单项的选中状态和如何根据nIDCheckItem确定菜单项的位置，可以是MF\_CHECKED或MF\_UNCHECKED与MF\_BYPOSITION或MF\_BYCOMMAND的组合，这几个标志的含义如下：  
MF\_BYCOMMAND：为默认值。说明参数nIDCheckItem表示菜单项的ID  
MF\_BYPOSITION：说明参数nIDCheckItem表示菜单项的位置，第一个菜单项的位置是0  
MF\_CHECKED：为菜单项添加选中标记  
MF\_UNCHECKED：为菜单项移除选中标记

       该函数返回菜单项之前的状态：MF\_CHECKED或MF\_UNCHECKED, 如果菜单项不存在则返回0xFFFFFFFF。

**UINT EnableMenuItem(UINT nIDEnableItem,UINT nEnable);**

       激活、禁用菜单项或使其变灰。参数nIDEnableItem指定要激活、禁用或变灰的菜单项。参数nEnable指定操作的类型，可以是MF\_DISABLED、MF\_ENABLED或MF\_GRAYED与MF\_BYCOMMAND或MF\_BYPOSITION的组合，这些值的含义如下：  
MF\_BYCOMMAND：同CheckMenuItem  
MF\_BYPOSITION：同CheckMenuItem  
MF\_DISABLED：禁用菜单项，使其不能被选择但不变灰  
MF\_ENABLED：激活菜单项，使其能够被选择并由变灰状态恢复  
MF\_GRAYED：禁用菜单项，使其不能被选择并变灰

       该函数返回菜单项之前的状态：MF\_DISABLED、MF\_ENABLED或MF\_GRAYED

**CMenu\* GetSubMenu(int nPos) const;**

       获取弹出菜单的CMenu对象。参数nPos指定弹出菜单在菜单中的位置，不能使用ID。返回值是CMenu对象的指针，该CMenu对象的m\_hMenu成员为由nPos指定的弹出菜单的句柄，如果不存在这样的CMenu对象则返回NULL，然后创建一个临时弹出菜单。

       CMenu类的成员函数先讲这些，如果大家需要用其他的函数可以到MSDN中查看，解释的很清楚。

菜单消息

       菜单主要能发送两种消息：COMMAND消息和UPDATE\_COMMAND\_UI消息。下面分别讲解：

       COMMAND消息：在菜单项被点击时发送该消息。

       UPDATE\_COMMAND\_UI消息：用来维护菜单项的各项状态，包括激活、禁用、变灰、选中、未选中等。在下拉菜单每次打开的时候，所有菜单项的此消息都会被发送出去。如果所属类中为菜单项的该消息添加了处理函数，则执行相应函数更新菜单状态，如果菜单项没有此消息处理函数，也没有COMMAND消息的处理函数，那么它就会变灰。

## 8.2工具栏资源详解

工具栏一般位于主框架窗口的上部，菜单栏的下方，由一些带图片的按钮组成。当用户用鼠标单击工具栏上某个按钮时，程序会执行相应的操作，如果鼠标没有点击，只是停留在某个按钮上一会后，会弹出一个小窗口显示提示信息。  
一般工具栏中的按钮在菜单栏中都有对应的菜单项中，即点击工具栏按钮与点击菜单项的效果相同。但工具栏中的按钮都显式的排列出来，操作很方便，而且按钮上的图片描述功能更直观，所以工具栏作为用户操作接口来说比菜单更加便捷。

工具栏按钮的各项属性  
 ID属性：ID\_FILE\_NEW。不知大家是否还记得，菜单IDR\_MAINFRAME的菜单项File->New的ID也是ID\_FILE\_NEW，两者ID相同，正是如此才使得工具栏第一个按钮与菜单项File->New能实现相同的功能。所以大家一定要记住，如果想让工具栏某个按钮与菜单栏某个菜单项点击后执行的操作相同，就要为两者设置相同的ID。  
Prompt属性：Create a new document\nNew。此属性为工具栏按钮的提示文本。在鼠标指向此按钮时，状态栏中会显示“Create a new document”，当弹出提示信息窗口时会显示包含“New”的提示信息。“\n”是两者的分隔转义符。  
Height属性：15。此属性为工具栏按钮的像素高度。  
Width属性：16。此属性为工具栏按钮的像素宽度。

       工具栏资源的最右边总是会有一个待编辑的按钮，我们对其进行编辑后，工具栏资源会自动增加一个新的空白按钮，这也实现了按钮的添加操作。如果我们想要删除某个按钮，就可以用鼠标左键点住它，拖出工具栏资源的范围即可。

       另外，我们看到，第三个按钮（保存按钮）和第四个按钮（剪切按钮）之间有一些间隙，在运行程序后会出现一个竖的分隔线，所以想要在两个按钮之间添加分隔线的话，可以用鼠标左键拖住右边的按钮往右稍移动一些就可以了。

CToolBar类的主要成员函数（VS2008起自动使用CMFCToolBar，用法类似功能丰富，非派生）

[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)为工具栏的操作提供了CToolBar类。下面介绍CToolBar类的主要成员函数。

**virtual BOOL CreateEx(  
                 CWnd\* pParentWnd,  
                 DWORD dwCtrlStyle = TBSTYLE\_FLAT,  
                 DWORD dwStyle = WS\_CHILD | WS\_VISIBLE | CBRS\_ALIGN\_TOP,  
                 CRect rcBorders = CRect(0, 0, 0, 0),  
                 UINT nID = AFX\_IDW\_TOOLBAR  
        );**创建工具栏对象。参数pParentWnd为工具栏父窗口的指针。参数dwCtrlStyle为工具栏按钮的风格，默认为TBSTYLE\_FLAT，即“平面的”。参数dwStyle为工具栏的风格，默认取值WS\_CHILD | WS\_VISIBLE | CBRS\_ALIGN\_TOP，由于是主框架窗口的子窗口，所以要有WS\_CHILD和WS\_VISIBLE风格，CBRS\_ALIGN\_TOP风格表示工具栏位于父窗口的顶部， 各种风格可以参见MSDN的Toolbar Control and Button Styles中的定义。参数rcBorders为工具栏边框各个方向的宽度，默认为CRect(0, 0, 0, 0)，即没有边框。参数nID为工具栏子窗口的ID，默认为AFX\_IDW\_TOOLBAR。

**BOOL LoadBitmap(UINT nIDResource);**

       为工具栏加载位图。参数nIDResource为位图资源的ID。成功则返回TRUE，否则返回FALSE。注意，这里的位图资源应当为每个工具栏按钮都提供位图，如果图片不是标准大小（16像素宽，15像素高），则需要调用SetSizes成员函数调整按钮大小和图片大小。

**BOOL LoadToolBar(UINT nIDResource);**

       加载由nIDResource指定的工具栏。参数nIDResource为要加载的工具栏的资源ID。成功则返回TRUE，否则返回FALSE。

**void SetSizes(SIZE sizeButton,SIZE sizeImage);**

       设置工具栏按钮的大小和图片的大小。参数sizeButton为工具栏按钮的像素大小。参数sizeImage为图片的像素大小。

**void SetButtonStyle(int nIndex,UINT nStyle);**

       设置工具栏按钮或分隔线的风格，或者为按钮分组。参数nIndex为将要进行设置的按钮或分隔线的索引。参数nStyle为按钮风格，可以是以下取值：  
TBBS\_BUTTON   标准按钮(默认)  
TBBS\_SEPARATOR   分隔条   
TBBS\_CHECKBOX   复选框   
TBBS\_GROUP   标记一组按钮的开始  
TBBS\_CHECKGROUP   标记一组复选框的开始  
TBBS\_DROPDOWN   创建下拉列表按钮  
TBBS\_AUTOSIZE   按钮的宽度根据按钮文本计算，而不基于图片大小   
TBBS\_NOPREFIX   按钮的文本没有快捷键前缀

**UINT GetButtonStyle(int nIndex) const;**

       获取工具栏按钮或分隔条的风格。风格可参考SetButtonStyle。参数nIndex为按钮或分隔条的索引。

**BOOL SetButtonText(int nIndex,LPCTSTR lpszText);**

       设置工具栏按钮的文本。参数nIndex为工具栏按钮的索引。参数lpszText为指向要设置的文本字符串的指针。设置成功则返回TRUE，否则返回FALSE。

**CString GetButtonText(int nIndex) const;**

       获取工具栏按钮上显示的文本。参数nIndex为工具栏按钮的索引。

工具栏的使用

       上一节中鸡啄米提到过，一般情况下工具栏中的按钮在菜单栏中都有对应的菜单项，两者实现的功能相同，要想实现这种效果，只需要将工具栏按钮的ID与对应的菜单栏中菜单项的ID设置为相同值即可。

       在实际使用工具栏时，除了前面讲的资源编辑外，其他使用与菜单类似。例如，对COMMAND消息和UPDATE\_COMMAND\_UI消息，可以像[VS2010/MFC编程入门之三十五（菜单：菜单及CMenu类的使用）](http://www.jizhuomi.com/software/212.html)中的菜单应用实例那样为工具栏按钮添加消息处理函数。

       如果工具栏按钮对应的菜单项已经添加了消息处理函数，那么就不必再为它添加了，因为它的ID与菜单项相同，所以会调用同样的消息处理函数。这样点击工具栏按钮与点击相应菜单项执行相同的功能，在菜单项为选中、激活或禁用等状态时，工具栏按钮会有一样的状态。

工具栏的创建

       大家在第三十四讲创建的Example34工程的CMainFrame类中看到，它创建工具栏所使用的类并不是常用的CToolBar类，而是CMFCToolBar类。CMFCToolBar类是自VS2008以来MFC提供的类，它与CToolBar类有些类似，但功能更丰富。这里要注意，CMFCToolBar类与CToolBar类并没有任何派生关系。  
鸡啄米这里就以CMFCToolBar类来讲讲工具栏的创建步骤：  
1. 创建工具栏资源。  
2. 构造CMFCToolBar类的对象。  
3. 调用CMFCToolBar类的Create或CreateEx成员函数创建工具栏。  
4. 调用LoadToolBar成员函数加载工具栏资源。  
因为创建框架窗口时需要调OnCreate函数，所以工具栏的创建也是在OnCreate中完成的。

工具栏的停靠

       在创建好工具栏后，如果想要停靠工具栏，也需要添加相应的停靠代码。工具栏停靠的步骤及需要调用的函数如下（前两个步骤可以颠倒顺序）：

1. 在框架窗口中启用停靠。

           若要将工具栏停靠到某个框架窗口，则必须启用该框架窗口（或目标）以允许停靠。可以在CFrameWndEx类中调用下面的成员函数来实现：

           BOOL EnableDocking(DWORD dwDockStyle);

           该函数采用一个DWORD参数，用来指定框架窗口的哪个边可以接受停靠，可以有四种取值：CBRS\_ALIGN\_TOP（顶部）、CBRS\_ALIGN\_BOTTOM（底部）、CBRS\_ALIGN\_LEFT（左侧）、CBRS\_ALIGN\_RIGHT（右侧）。如果希望能够将控制条停靠在任意位置，将CBRS\_ALIGN\_ANY作为参数传递给EnableDocking。

       2. 工具栏启用停靠。

           框架窗口启用停靠准备好后，必须以相似的方式准备工具栏。为想要停靠的每一个工具栏CMFCToolBar对象调用下面的函数：

           virtual void EnableDocking(DWORD dwAlignment);

           允许工具栏停靠到框架窗口，并指定工具栏应停靠的目标边。此函数指定的目标边必须与框架窗口中启用停靠的边匹配，否则工具栏无法停靠，为浮动状态。

       3. 停靠工具栏。

           当用户试图将工具栏放置在允许停靠的框架窗口某一边时，需要框架CFrameWndEx类调用以下函数：

           void DockPane(CBasePane\* pBar,UINT nDockBarID=0,LPCRECT lpRect=NULL);

           参数pBar为要停靠的控制条的指针，参数nDockBarID为要停靠的框架窗口某条边的ID，可以是以下四种取值：AFX\_IDW\_DOCKBAR\_TOP、AFX\_IDW\_DOCKBAR\_BOTTOM、AFX\_IDW\_DOCKBAR\_LEFT、AFX\_IDW\_DOCKBAR\_RIGHT。

## 8.3状态栏资源详解

状态栏简介

       状态栏相信大家在很多窗口中都能见到，它总是用来显示各种状态。状态栏实际上也是一个窗口，一般分为几个窗格，每个窗格分别用来显示不同的信息和状态等，如菜单项和工具栏按钮的提示信息。

CStatusBar类（VS2008之后系统自动使用CMFCStatusBar，用法类似但功能更丰富）

[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)为状态栏提供了CStatusBar类，封装了状态栏的属性和操作。

       下面是CStatusBar类几个主要的成员函数：

       virtual BOOL Create(CWnd\* pParentWnd, DWORD dwStyle = WS\_CHILD | WS\_VISIBLE | CBRS\_BOTTOM, UINT nID = AFX\_IDW\_STATUS\_BAR);

       创建一个状态栏。参数pParentWnd为状态栏父窗口的指针，参数dwStyle为状态栏的风格，除了标准的Windows风格外，它还支持：

       CBRS\_TOP：位于框架窗口的顶部。

       CBRS\_BOTTOM：位于框架窗口的底部。

       CBRS\_NOALIGN：父窗口大小改变时状态栏不会被重新定位。

       参数nID指定状态栏的ID。

       BOOL SetIndicators(const UINT\* lpIDArray, int nIDCount);

       为每个指示器设置显示文本，具体来说，就是用lpIDArray数组中的对应元素为每个指示器设置一个ID，然后加载每个ID代表的字符串，设置为这些指示器的显示文本。参数lpIDArray为指向一个ID数组的指针，参数nIDCount为lpIDArray数组的元素个数。

       UINT GetItemID(int nIndex) const;

       获取由nIndex指定的指示器的ID。参数nIndex为要获取ID的指示器索引。

       CString GetPaneText(int nIndex) const;

       获取状态栏窗格中显示的文本。参数nIndex为要获取文本的窗格的索引。返回值为包含窗格文本的CString对象。

       BOOL SetPaneText(int nIndex, LPCTSTR lpszNewText, BOOL bUpdate = TRUE);

       设置状态栏窗格的显示文本。参数nIndex为要设置文本的窗格的索引，参数lpszNewText为指向新的窗格文本的指针，参数bUpdate表示是否设置后立即更新显示。如果设置成功则返回TRUE，否则返回FALSE。

状态栏的创建  
1. 构造一个CMFCStatusBar类的对象。  
在MainFrm.h文件中，为CMainFrame类定义了一个成员对象：CMFCStatusBar     m\_wndStatusBar;。  
2. 调用CMFCStatusBar::Create函数来创建状态栏窗口。  
在CMainFrame::OnCreate函数的实现中，我们可以找到CMFCStatusBar::Create函数的调用：

1. **if** (!m\_wndStatusBar.Create(**this**))
2. {
3. TRACE0("Failed to create status bar\n");
4. **return** -1;      // fail to create
5. }

       3. 调用CMFCStatusBar::SetIndicators函数为状态栏划分窗格，并为每个指示器设置显示文本。  
CMFCStatusBar::SetIndicators函数需要一个ID数组的参数，在MainFrm.cpp中，如下定义了一个窗格ID的数组：

1. **static** **UINT** indicators[] =
2. {
3. ID\_SEPARATOR,           // status line indicator
4. ID\_INDICATOR\_CAPS,
5. ID\_INDICATOR\_NUM,
6. ID\_INDICATOR\_SCRL,
7. };

       indicators数组定义了状态栏窗格的划分信息。第一个元素一般为ID\_SEPARATOR，对应的窗格用来显示命令提示信息，上面数组中的后三项为指示器文本的字符串ID，可以根据这些ID在String Table字符串资源中找到相应的字符串，查找方法是，在Resource View资源视图中，打开String Table字符串资源，可以看到有ID、Value和Caption三列，在ID列中找到需要的ID，对应的Caption列文本就是要查找的字符串。ID\_INDICATOR\_CAPS、ID\_INDICATOR\_NUM和ID\_INDICATOR\_SCRL对应的字符串分别是CAP、NUM、SCRL，对应的三个窗格分别为Caps Lock指示器、Num Lock指示器和Scroll Lock指示器。  
定义了指示器数组就可以使用CMFCStatusBar::SetIndicators函数为状态栏划分窗格了，依然是在CMainFrame::OnCreate函数中调用：

1. m\_wndStatusBar.SetIndicators(indicators, **sizeof**(indicators)/**sizeof**(**UINT**));

       这样状态栏就创建完成了，之后我们可以通过CMFCStatusBar::SetPaneText设置窗格的文本。

# 九、文档，视图和框架

## 9.1 概述

文档、视图和框架简介

**文档**

       文档对象用于管理和维护数据，包括保存数据、取出数据以及修改数据等操作，在数据被修改以后，文档可以通知其对应的所有视图更新显示。

**视图**

       视图对象将文档中的数据可视化，负责从文档对象中取出数据显示给用户，并接受用户的输入和编辑，将数据的改变反映给文档对象。视图充当了文档和用户之间媒介的角色。

**框架**

       一个文档可能有多个视图界面，这就需要有框架来管理了。框架就是用来管理文档和视图的。框架窗口是应用程序的主窗口，应用程序执行时会先创建一个最顶层的框架窗口。视图窗口是没有菜单和边界的子窗口，它必须包含在框架窗口中，即置于框架窗口的客户区内。

**文档模板**

       文档模板中存放了与文档、视图和框架相关的信息。应用程序通过文档模板创建文档对象、框架窗口对象和视图对象。另外，文档、视图和框架之间的关系也是由文档模板来管理的。  
在构造文档模板类CSingleDocTemplate的对象时，第一个参数是资源ID IDR\_MAINFRAME，它包括框架窗口图标等，后面的三个参数都是RUNTIME\_CLASS宏的调用，RUNTIME\_CLASS用于获取类的运行时信息，文档模板可以根据这些动态创建信息来创建相应类的对象，即文档对象、框架窗口对象和视图对象。AddDocTemplate函数用来注册文档模板对象。

框架类、文档类和视图类

       在[VS2010](http://www.jizhuomi.com/catalog.asp?tags=VS2010)自动生成的代码中，框架类继承于CFrameWndEx类，文档类继承于CDocument类，视图类继承于CView类。

**CFrameWndEx类又继承于CFrameWnd类，CFrameWnd类中用于管理文档和视图的成员函数包括：**

**virtual CDocument\* GetActiveDocument( );**

       获得当前活动视图对应文档对象的指针，如果不存在则返回NULL。

**CView\* GetActiveView( ) const;**

       获得当前活动视图对象的指针，如果不存在则返回NULL。

**void SetActiveView(CView\* pViewNew, BOOL bNotify = TRUE);**

       设置活动视图。参数pViewNew为要激活的视图对象的指针，参数bNotify指定视图是否接收激活通知。

**CDocument类的主要成员函数：**

**virtual BOOL OnNewDocument( );**

       创建新文档。可以重载使用。

**virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);**

       打开文档。参数lpszPathName为要打开的文档的路径。可以重载使用。

**virtual BOOL OnSaveDocument(LPCTSTR lpszPathName);**

       保存文档。参数lpszPathName指定文档保存到的全路径。可以重载使用。

**CDocTemplate\* GetDocTemplate( ) const;**

       获取此文档类型对应的文档模板对象的指针。如果此文档没有被文档模板管理则返回NULL。

**virtual POSITION GetFirstViewPosition( ) const;**

       获取文档中视图列表的第一个视图的位置。

**virtual CView\* GetNextView(POSITION& rPosition) const;**

       利用此函数可以迭代处理文档的所有视图。参数rPosition为上一次调用GetFirstViewPosition或GetNextView成员函数返回的POSITION值的引用。

**void AddView(CView\* pView);**

       为文档增加一个视图。参数pView为要增加的视图对象的指针。

**void RemoveView(CView\* pView);**

       移除某个视图与文档的关联。参数pView为要移除的视图对象的指针。

**void UpdateAllViews(CView\* pSender, LPARAM lHint = 0L, CObject\* pHint = NULL);**

       在文档被更改后调用此函数更新视图。参数pSender指向修改文档的视图，实际应用时常用来指定哪个视图不需要更新，如果更新所有视图则设为NULL，参数lHint包含了文档修改的信息，参数pHint指向存储文档修改信息的对象。

**CView类中与文档/视图结构相关的成员函数包括：**

**CDocument\* GetDocument( ) const;**

       获取视图关联的文档对象的指针。如果视图没有关联到文档上则返回NULL。

各个对象之间的关系

       文档、视图、框架结构中涉及到的对象主要有：应用程序对象、文档模板对象、文档对象、视图对象和框架窗口对象等。根据上一节的概述，大家对它们的概念已经有所了解了，下面就对它们之间的关系进行总结和概括，并对各个关系中用到的类的成员函数进行介绍。

       1. 应用程序对象保存了一个文档模板的列表。在任何对象中调用全局函数AfxGetApp都可以获得应用程序对象的指针。通过调用CWinAppEx::GetFirstDocTemplatePosition、CWinAppEx::GetNextDocTemplate函数可以遍历所有的文档模板。

       2. 文档模板对象用于维护文档、视图和框架窗口的映射关系，它包含有一个已打开文档的列表。我们可以通过调用CDocTemplate::GetFirstDocPosition、CDocTemplate::GetNextDoc来遍历该文档模板对应的所有文档。

       3. 框架窗口对象中包含有指向当前活动视图对象的指针。AfxGetApp()->m\_pMainWnd即为主框架窗口对象的指针。我们可以通过调用CFrameWndEx::GetActiveView来获取当前活动视图对象的指针，并且使用CFrameWndEx::GetActiveDocument函数可以获得当前活动视图对应的文档。

       4. 文档对象中维护着该文档的视图列表，以及创建该文档的文档模板对象的指针。我们可以通过调用CDocument::GetFirstViewPosition，CDocument::GetNextView来遍历该文档关联的所有视图，调用CDocument::GetDocTemplate获取创建该文档的文档模板对象的指针。

       5. 视图是框架窗口的子窗口，它保存有指向对应的文档对象的指针。我们可以通过调用CView::GetParentFrame获取其所属的框架窗口对象的指针，调用CView::GetDocument获取该视图对应的文档对象的指针。

       另外，在MDI多文档程序中，调用CMDIFrameWnd::MDIGetActive可以获取当前活动的MDI子窗口。

文档和视图的关系

       应用程序可以是单文档程序也可以是多文档程序。单文档程序中主框架窗口和文档框架窗口重合，而多文档程序的主框架窗口中有客户窗口，客户窗口中又包含了多个文档框架窗口。

       文档和视图是一对多的关系。一个文档可以对应多个视图，例如在Word中一个文档有普通视图、大纲视图、Web版式视图、阅读版式视图等多种视图。而一个视图只能属于一个文档。最简单的应用程序是单文档单视图程序，除此之外还有单文档多视图程序、多文档程序等。

       每个文档对象都保存着一个视图列表，可以通过CDocument::AddView函数添加视图，通过CDocument::RemoveView函数删除视图，在数据发生变化时调用CDocument::UpdateAllViews函数更新所有视图。

       在[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)中文档可以有三种视图模式：

       1. 文档有多个视图对象，它们是同一个视图类的对象，每个视图对象位于一个独立的文档框架窗口中。

        2. 文档的基于同一个视图类的多个视图对象，位于同一个文档框架窗口中。Word的子窗口就是这种视图模式。

  　 3．文档的视图对象属于不同的视图类，但所有的视图对象位于同一文档框架窗口中。

## 9.2 分割窗口

**分割窗口概述**

    分割窗口，顾名思义，就是将一个窗口分割成多个窗格，在每个窗格中都包含有视图，或者是同一类型的视图，或者是不同类型的视图。MFC分割窗口的方式有两种，动态分割和静态分割。

       动态分割窗口通常用于创建同一个文档对应的多个视图，而且这些视图一般都是同一类型的视图，能够在用户编辑文档的不同部分时提供方便。动态分割窗口最多可以有两行两列。

       静态分割窗口比较常见。我们经常能看到某个软件打开后，界面窗口默认被分割成了几个窗格，这就是静态分割窗口。静态分割窗口指在窗口创建时，分割的窗格就已经生成了，而且用户不能改变窗格的数量和顺序。静态分割窗口最多支持16行16列。通常静态分割窗口的每个窗格中包含不同类的视图，当然也可以是同一类的视图。

 CSplitterWnd类

       MFC中的分割窗口类-CSplitterWnd类提供了分割窗口的功能。CSplitterWnd类中包含一个分割器窗口，该分割器窗口就是一个包含多个窗格的窗口。我们分割窗口时就是直接在此分割器窗口中分割的。  
下面介绍三个最常用的成员函数：

1. **virtual** **BOOL** Create(
2. CWnd\* pParentWnd,
3. **int** nMaxRows,
4. **int** nMaxCols,
5. SIZE sizeMin,
6. CCreateContext\* pContext,
7. **DWORD** dwStyle = WS\_CHILD | WS\_VISIBLE | WS\_HSCROLL | WS\_VSCROLL | SPLS\_DYNAMIC\_SPLIT,
8. **UINT** nID = AFX\_IDW\_PANE\_FIRST
9. );

       创建动态分割窗口。参数pParentWnd为分割器窗口的父框架窗口；参数nMaxRows为分割器窗口的最大行数，不能超过2；参数nMaxCols为分割器窗口的最大列数，也不能超过2；参数sizeMin为窗格能显示的最小尺寸，如果窗格尺寸小于sizeMin则不显示；参数pContext为指向CCreateContext结构的指针，大多数情况下可以赋值为父框架窗口的pContext；参数dwStyle指定窗口风格；参数nID为分割窗口的ID，除非分割器窗口嵌入到另一个分割器窗口中，否则可以取值AFX\_IDW\_PANE\_FIRST。

1. **virtual** **BOOL** CreateStatic(
2. CWnd\* pParentWnd,
3. **int** nRows,
4. **int** nCols,
5. **DWORD** dwStyle = WS\_CHILD | WS\_VISIBLE,
6. **UINT** nID = AFX\_IDW\_PANE\_FIRST
7. );

       创建静态分割窗口。参数pParentWnd、dwStyle和nID同上；参数nRows为行数，不能超过16；参数nCols为列数，同样不能超过16。

1. **virtual** **BOOL** CreateView(
2. **int** row,
3. **int** col,
4. CRuntimeClass\* pViewClass,
5. SIZE sizeInit,
6. CCreateContext\* pContext
7. );

       为静态分割窗口创建窗格视图。参数row指定分割器窗口中放置新视图的行；参数col指定放置新视图的列；参数pViewClass指定新视图的CRuntimeClass对象；参数sizeInit指定新视图的初始大小；参数pContext为指向CCreateContext结构的指针，通常可以赋值为传递给父框架窗口的重载函数CFrameWnd::OnCreateClient的pContext参数值。

**动态分割窗口**

       创建动态分割窗口的步骤为：

       1. 在父框架类中定义一个CSplitterWnd类型的成员对象。

       2. 重载父框架类的CFrameWnd::OnCreateClient成员函数。

       3. 在重载的CFrameWnd::OnCreateClient函数中调用CSplitterWnd成员对象的Create函数。

**静态分割窗口**

       创建静态分割窗口的步骤为：

       1. 在父框架类中定义一个CSplitterWnd类型的成员对象。

       2. 重载父框架类的CFrameWnd::OnCreateClient成员函数。

       3. 在重载的CFrameWnd::OnCreateClient函数中调用CSplitterWnd成员对象的CreateStatic成员函数，然后可以调用CSplitterWnd成员对象的CreateView成员函数为每个窗格创建视图。

# 十、MFC常用辅助类

## 10.1 CString类

 CString类简介  
使用[VS2010](http://www.jizhuomi.com/catalog.asp?tags=VS2010)的话，可能会见到CStringT，实际上它是一个操作可变长度字符串的模板类。CStringT模板类有三个实例：CString、CStringA和CStringW，它们分别提供对TCHAR、char和wchar\_t字符类型的字符串的操作。char类型定义的是Ansi字符，wchar\_t类型定义的是Unicode字符，而TCHAR取决于MFC工程的属性对话框中的Configuration Properties->General->Character Set属性，如果此属性为Use Multi-Byte Character Set，则TCHAR类型定义的是Ansi字符，而如果为Use Unicode Character Set，则TCHAR类型定义的是Unicode字符。三个字符串类的操作是一样的，只是处理的字符类型不同。

CString类的字符串操作

**1. CString类的构造函数**

       CString类有很多构造函数，这里只介绍几个比较常用的：  
       CString(const CString& stringSrc);  
       将一个已经存在的CString对象stringSrc的内容拷贝到该CString对象。例如：

1. CString str1(\_T("www.jizhuomi.com"));  // 将常量字符串拷贝到str1
2. CString str2(str1);       // 将str1的内容拷贝到str2

       CString(LPCTSTR lpch, int nLength);

       将字符串lpch中的前nLength个字符拷贝到该CString对象。例如：

1. CString str(\_T("www.jizhuomi.com"),3); // 构造的字符串对象内容为"www"

       CString(TCHAR ch, int nLength = 1);

       使用此函数构造的CString对象中将含有nLength个重复的ch字符。例如：

1. CString str(\_T('w'),3);  // str为"www"

**2. CString类的大小写转换及顺序转换函数**

       CString& MakeLower();

       将字符串中的所有大写字符转换为小写字符。

       CString& MakeUpper();

       将字符串中的所有小写字符转换为大写字符。

       CString& MakeReverse();

       将字符串中所有字符的顺序颠倒。

       例如：

1. CString str(\_T("JiZhuoMi"));
2. str.MakeLower();   // str为"jizhuomi"
3. str.MakeUpper();   // str为"JIZHUOMI"
4. str.MakeReverse(); // str为"IMOUHZIJ"

**3. CString对象的连接**

       多个CString对象的连接可以通过重载运算符+、+=实现。例如：

1. CString str(\_T("jizhuomi"));      // str内容为"jizhuomi"
2. str = \_T("www.") + str + \_T("."); // str为"www.jizhuomi."
3. str += \_T("com");                 // str为"www.jizhuomi.com"

**4. CString对象的比较**

       CString对象的比较可以通过==、!=、<、>、<=、>=等重载运算符实现，也可以使用Compare和CompareNoCase成员函数实现。

       int Compare(PCXSTR psz) const;

       将该CString对象与psz字符串比较，如果相等则返回0，如果小于psz则返回值小于0，如果大于psz则返回值大于0。

       int CompareNoCase(PCXSTR psz) const throw();

       此函数与Compare功能类似，只是不区分大小写。例如：

1. CString str1 = \_T("JiZhuoMi");
2. CString str2 = \_T("jizhuomi");
3. **if** (str1 == str2)
4. {
5. // 因为str1、str2不相等，所以不执行下面的代码
6. ...
7. }
8. **if** (0 == str1.CompareNoCase(str2))
9. {
10. // 因为不区分大小写比较时，CompareNoCase函数返回0，所以执行下面的代码
11. ...
12. }

**5. CString对象字符串的提取操作**

       CString Left(int nCount) const;  
       提取该字符串左边nCount个字符的子字符串，并返回一个包含这个子字符串的拷贝的CString对象。

       CString Right(int nCount) const;  
       提取该字符串右边nCount个字符的子字符串，并返回一个包含这个子字符串的拷贝的CString对象。

       CString Mid(int iFirst,int nCount) const;  
       提取该字符串中以索引iFirst位置开始的nCount个字符组成的子字符串，并返回一个包含这个子字符串的拷贝的CString对象。

       CString Mid(int iFirst) const;  
       提取该字符串中以索引iFirst位置开始直至字符串结尾的子字符串，并返回一个包含这个子字符串的拷贝的CString对象。例如：

1. CString str1 = \_T("jizhuomi");
2. CString str2 = str1.Left(3);    // str2为"jiz"
3. str2 = str1.Right(2);           // str2为"mi"
4. str2 = str1.Mid(1,3);           // str2为"izh"
5. str2 = str1.Mid(5);             // str2为"omi"

**6. CString对象字符串的查找操作**

       int Find(PCXSTR pszSub,int iStart=0) const throw( );  
       int Find(XCHAR ch,int iStart=0) const throw( );

       在CString对象字符串的iStart索引位置开始，查找子字符串pszSub或字符ch第一次出现的位置，如果没有找到则返回-1。

       int FindOneOf(PCXSTR pszCharSet) const throw( );

       查找pszCharSet字符串中的任意字符，返回第一次出现的位置，找不到则返回-1。

       int ReverseFind(XCHAR ch) const throw();

       从字符串末尾开始查找指定的字符ch，返回其位置，找不到则返回-1。这里要注意，尽管是从后向前查找，但是位置的索引还是要从开始算起。

1. CString str = \_T("jizhuomi");
2. **int** nIndex1 = str.Find(\_T("zh"));   // nIndex1的值为2
3. **int** nIndex2 = str.FindOneOf(\_T("mui")); // nIndex2的值为1
4. **int** nIndex3 = str.ReverseFind(\_T('i'));  // nIndex3的值为7

**7. CString类对象字符串的替换与删除**

       int Replace(PCXSTR pszOld,PCXSTR pszNew);

       用字符串pszNew替换CString对象中的子字符串pszOld，返回替换的字符个数。

       int Replace(XCHAR chOld,XCHAR chNew);

       用字符chNew替换CString对象中的字符chOld，返回替换的字符个数。

       int Delete(int iIndex,int nCount = 1);

       从字符串中删除iIndex位置开始的nCount个字符，返回删除操作后的字符串的长度。

       int Remove(XCHAR chRemove);

       删除字符串中的所有由chRemove指定的字符，返回删除的字符个数。例如：

1. CString str = \_T("jizhuomi");
2. **int** n1 = str.Replace(\_T('i'), \_T('j'));  // str为"jjzhuomj"，n1为2
3. **int** n2 = str.Delete(1,2);        // str为"jhuomj"，n2为6
4. **int** n3 = str.Remove(\_T('j'));    // str为"huom"，n3为2

       8. CString类的格式化字符串方法

       使用CString类的Format成员函数可以将int、short、long、float、double等数据类型格式化为字符串对象。

       void \_\_cdecl Format(PCXSTR pszFormat,[, argument]...);

       参数pszFormat为格式控制字符串；参数argument可选，为要格式化的数据，一般每个argument在pszFormat中都有对应的表示其类型的子字符串，int型的argument对应的应该是"%d"，float型的应对应"%f"，等等。例如：

1. CString str;
2. **int** a = 1;
3. **float** b = 2.3f;
4. str.Format(\_T("a=%d,b=%f"), a, b);  // str为"a=1,b=2.300000"

## 10.2 CTime类与CTimeSpan类

**日期和时间类简介**CTime类的对象表示的时间是基于格林威治标准时间（GMT）的。CTimeSpan类的对象表示的是时间间隔。CTime类和CTimeSpan类一般不会被继承使用。两者对象的大小都是8个字节。CTime表示的日期上限是3000年12月31日，下限是1970年1月1日 12:00:00 AM GMT。

**CTime类的主要成员函数**

       下面列出CTime类的主要成员函数，并加以讲解。

**CTime();**

       构造一个未经初始化的CTime对象。此构造函数使我们可以定义一个CTime对象的数组，在使用数组前需要以有效的时间值为其初始化。

**CTime(\_\_time64\_t time);**

       以一个\_\_time64\_t（注意：最前面的下划线有两条）类型的数据来构造一个CTime对象。参数time是一个\_\_time64\_t类型的值，表示自GMT时间1970年1月1日零点以来的秒数，这里要注意的是，参数time代表的时间会转换为本地时间保存到构造的CTime对象中。例如，我们传递参数0构造一个CTime对象，然后调用CTime对象的GetHour成员函数将返回8，因为参数0代表的GMT时间转换为北京时间后为1970年1月1日 8:00:00。  
**CTime(  
   int nYear,  
   int nMonth,  
   int nDay,  
   int nHour,  
   int nMin,  
   int nSec,  
   int nDST = -1   
);**

       以本地时间的年、月、日、小时、分钟、秒等几个时间分量构造CTime对象。参数nYear、nMonth、nDay、nHour、nMin、nSec分别表示年、月、日、小时、分钟、秒，取值范围如下：

|  |  |
| --- | --- |
| 时间分量 | 取值范围 |
| nYear | 1970-3000 |
| nMonth | 1-12 |
| nDay | 1-31 |
| nHour | 0-23 |
| nMin | 0-59 |
| nSec | 0-59 |

       参数nDST指定是否实行夏令时，为0时表示实行标准时间，为正数时表示实行夏令时，为负数时由系统自动计算实行的是标准时间还是夏令时。

**CTime(const SYSTEMTIME& st,int nDST = - 1) ;**

       以一个SYSTEMTIME结构体变量来构造CTime对象。SYSTEMTIME结构体也是我们对日期时间的常用表示方式。参数st为以本地时间表示的SYSTEMTIME对象，参数nDST同上。

**static CTime WINAPI GetCurrentTime( );**

       获取系统当前日期和时间。返回表示当前日期和时间的CTime对象。

**int GetYear( ) const；**

       获取CTime对象表示时间的年份。范围从1970年1月1日到2038年（包括2038年）1月18日。

**int GetMonth( ) const;**

       获取CTime对象表示时间的月份。范围为1到12。

**int GetDay( ) const;**

       获取CTime对象表示时间的日期。范围为1到31。

**int GetHour( ) const;**

       获取CTime对象表示时间的小时。范围为0到23。

**int GetMinute( ) const;**

       获取CTime对象表示时间的分钟。范围为0到59。

**int GetSecond( ) const;**

       获取CTime对象表示时间的秒。范围为0到59。

**int GetDayOfWeek( ) const;**

       此函数的返回值表示CTime对象代表的是星期几，1表示是周日，2表示是周一，以此类推。

**CString Format(LPCTSTR pszFormat) const;**

       将CTime对象中的时间信息格式化为字符串。参数pszFormat是格式化字符串，与printf中的格式化字符串类似，格式化字符串中带有%前缀的格式码将会被相应的CTime时间分量代替，而其他字符会原封不动的拷贝到返回字符串中。格式码及含义如下：  
%a：周的英文缩写形式。  
%A：周的英文全名形式。  
%b： 月的英文缩写形式。  
%B：月的英文全名形式。  
%c： 完整的日期和时间。  
%d：十进制形式的日期（01-31）。  
%H：24小时制的小时（00-23）。  
%I： 12小时制的小时（00-11）。  
%j： 十进制表示的一年中的第几天（001-366）。  
%m： 月的十进制表示（01-12）。  
%M：十进制表示的分钟（00-59）。  
%p： 12小时制的上下午标示（AM/PM）。  
%S： 十进制表示的秒（00-59）。  
%U： 一年中的第几个星期（00-51），星期日是一周的第一天。  
%W： 一年中的第几个星期（00-51），星期一是一周的第一天。  
%w： 十进制表示的星期几（0-6）。  
%Y： 十进制表示的年。

**CTime operator +(CTimeSpan timeSpan) const;**

       将CTime对象和CTimeSpan对象相加，返回一个CTime对象。实际意义就是在一个时间的基础上推后一个时间间隔，得到一个新的时间。

**CTime operator -(CTimeSpan timeSpan) const;**

       将CTime对象和一个CTimeSpan相减，返回一个CTime对象。实际意义就是在一个时间的基础上提前一个时间间隔，得到一个新的时间。

**CTimeSpan operator -(CTime time) const;**

       将该CTime对象和另一个CTime对象相减，返回一个CTimeSpan对象。实际意义就是计算两个时间点的间隔，得到一个CTimeSpan对象。

**CTime& operator +=(CTimeSpan span);**

       为该CTime对象增加一个span表示的时间间隔。

**CTime& operator -=(CTimeSpan span);**

       为该CTime对象减去一个span表示的时间间隔。

**CTime& operator =(\_\_time64\_t time);**

       为该CTime对象赋予一个新的时间值。

       简单说下剩下的几个重载i运算符：  
       **operator ==**：    比较两个绝对时间是否相等。  
       **operator !=**：    比较两个绝对时间是否不相等。  
       **operator >** ：    比较两个绝对时间，是否前一个大于后一个。  
       **operator <** ：    比较两个绝对时间，是否前一个小于后一个。  
       **operator >=**： 比较两个绝对时间，是否前一个大于等于后一个。  
      **operator <=**： 比较两个绝对时间，是否前一个小于等于后一个。

**CTimeSpan类的主要成员函数**

       前面介绍了CTime类的成员函数，再来看CTimeSpan类的成员函数就比较容易了，这里只做简单介绍。

**CTimeSpan( );**

       构造一个未经初始化的CTimeSpan对象。

**CTimeSpan(\_\_time64\_t time);**

       以一个\_\_time64\_t类型的数据来构造CTimeSpan对象，参数time的含义上面CTime(\_\_time64\_t time)的讲解。  
**CTimeSpan(  
   LONG lDays,  
   int nHours,  
   int nMins,  
   int nSecs   
);**

       以天、小时、分钟、秒等时间分量来构造CTimeSpan对象。每个时间分量的取值范围如下：

|  |  |
| --- | --- |
| 时间分量 | 取值范围 |
| lDays | 0-25000（大约） |
| nHours | 0-23 |
| nMins | 0-59 |
| nSecs | 0-59 |

**GetDays()：**获得CTimeSpan类对象中包含的完整的天数。

**GetHours()：**获得当天的小时数，取值范围为-23到23。

**GetTotalHours()：**获得CTimeSpan类对象中包含的完整的小时数。

**GetMinutes()：**获得当前小时包含的分数，取值范围为-59到59。

**GetTotalMinutes()：**获得CTimeSpan类对象中包含的完整的分数。

**GetSeconds()：**获得当前分钟包含的秒数，取值范围为-59到59。

**GetTotalSeconds()：**获得CTimeSpan类对象中包含的完整的秒数。

**CString Format(LPCTSTR pszFormat) const;**

       将一个CTimeSpan对象格式化为字符串。使用方式与CTime::Format类似，格式码及含义如下：

       %D：CTimeSpan对象中的总天数；  
       %H：不足整天的小时数；  
       %M：不足1小时的分钟数；  
       %S：不足1分钟的秒数；  
       %%：百分号。

       另外，CTimeSpan类也重载了运算符“=”，“+”，“-”，“+=”，“-=”，“==”，“!=”，“<”，“>”，“<=”，“>=”，用于CTimeSpan对象的赋值、加减运算及两个CTimeSpan对象的比较。

## 10.3 定时器Timer

**定时器简介**

       定时器并不是一个类，可以帮助开发者或者用户定时完成某项任务。在使用定时器时，我们可以给系统传入一个时间间隔数据，然后系统就会在每个此时间间隔后触发定时处理程序，实现周期性的自动操作。例如，我们可以在数据采集系统中，为定时器设置定时采集时间间隔为1个小时，那么每隔1个小时系统就会采集一次数据，这样就可以在无人操作的情况下准确的进行操作。

[**MFC**](http://www.jizhuomi.com/catalog.asp?tags=MFC)**定时器**

[VS2010](http://www.jizhuomi.com/catalog.asp?tags=VS2010)编程中，我们可以使用MFC的CWnd类提供的成员函数SetTimer实现定时器功能，也可以使用Windows API函数SetTimer来实现。两者使用方法实际上很类似，但也有不同。CWnd类的SetTimer成员函数只能在CWnd类或其派生类中调用，而API函数SetTimer则没有这个限制，这是一个很重要的区别。  
 **1、启动定时器。**

       启动定时器就需要使用CWnd类的成员函数SetTimer。CWnd::SetTimer的原型如下：

       UINT\_PTR SetTimer(  
             UINT\_PTR nIDEvent,  
             UINT nElapse,  
             void (CALLBACK\* lpfnTimer  
       )(HWND,  
          UINT,  
          UINT\_PTR,  
          DWORD  
       )   
       );

       参数nIDEvent指定一个非零的定时器ID；参数nElapse指定间隔时间，单位为毫秒；参数lpfnTimer指定一个回调函数的地址，如果该参数为NULL，则WM\_TIMER消息被发送到应用程序的消息队列，并被CWnd对象处理。如果此函数成功则返回一个新的定时器的ID，我们可以使用此ID通过KillTimer成员函数来销毁该定时器，如果函数失败则返回0。

       通过SetTimer成员函数我们可以看出，处理定时事件可以有两种方式，**一种是通过WM\_TIMER消息的消息响应函数，一种是通过回调函数**。

**如果要启动多个定时器就多次调用SetTimer成员函数。另外，在不同的CWnd中可以有ID相同的定时器，并不冲突。  
2、为WM\_TIMER消息添加消息处理函数，或者定义回调函数。**

       如果调用CWnd::SetTimer函数时最后一个参数为NULL，则通过WM\_TIMER的消息处理函数来处理定时事件。添加WM\_TIMER消息的处理函数的方法是，在VS2010工程的Class View类视图中找到要添加定时器的类，点击右键，选择Properties，显示其属性页，然后在属性页工具栏上点击Messages按钮，下面列表就列出了所有消息，找到WM\_TIMER消息，添加消息处理函数。添加后，cpp文件中会出现类似如下内容：

1. BEGIN\_MESSAGE\_MAP(CExample44Dlg, CDialogEx)
2. ......
3. ON\_WM\_TIMER()
4. END\_MESSAGE\_MAP()
6. **void** CExample44Dlg::OnTimer(**UINT\_PTR** nIDEvent)
7. {
8. // TODO: Add your message handler code here and/or call default
10. CDialogEx::OnTimer(nIDEvent);
11. }

之后就可以在OnTimer函数中进行相应的处理了。OnTimer的参数nIDEvent为定时器ID，即在SetTimer成员函数中指定的定时器ID，如果有多个定时器，我们可以像下面这样处理：

1. **void** CExample44Dlg::OnTimer(**UINT\_PTR** nIDEvent)
2. {
3. // TODO: Add your message handler code here and/or call default
4. **switch** (nIDEvent)
5. {
6. **case** 1:
7. // 如果收到ID为1的定时器的消息则调用func1函数
8. func1();
9. **break**;
10. **case** 2:
11. // 如果收到ID为2的定时器的消息则调用func2函数
12. fun2();
13. **break**;
14. ......
15. **default**:
16. **break**;
17. }
19. CDialogEx::OnTimer(nIDEvent);
20. }

 如果调用CWnd::SetTimer函数时最后一个参数不为NULL，则需要定义回调函数。回调函数的形式如下：

1. **void** CALLBACK EXPORT TimerProc(
3. **HWND** hWnd, // handle of CWnd that called SetTimer
5. **UINT** nMsg, // WM\_TIMER
7. **UINT** nIDEvent // timer identification
9. **DWORD** dwTime // system time
11. );

参数hWnd为调用SetTimer成员函数的CWnd对象的句柄，即拥有此定时器的窗口的句柄；参数nMsg为WM\_TIMER，而且总是为WM\_TIMER；参数nIDEvent为定时器ID；参数dwTime为系统启动以来的毫秒数，即GetTickCount函数的返回值。

       这样CWnd::SetTimer函数最后一个参数就可以为TimerProc。

       这里注意下，**回调函数的名称不一定为TimerProc，可以取其他名字，但返回值类型、参数的类型和个数不能改变。**

**3、销毁定时器。**

       不再使用定时器时，可以销毁它。销毁定时器需使用CWnd类的KillTimer成员函数，CWnd::KillTimer函数的原型如下：

1. **BOOL** KillTimer(**UINT\_PTR** nIDEvent);

参数nIDEvent为要销毁的定时器的ID，是调用CWnd::SetTimer函数时设置的定时器ID。如果定时器被销毁则返回TRUE，而如果没有找到指定的定时器则返回FALSE。

       如果要销毁多个定时器，则多次调用KillTimer函数并分别传入要销毁的定时器的ID。

## 10.4 CFile文件操作类

**CFile类概述**

       如果你学过C语言，应该知道文件操作使用的是文件指针，通过文件指针实现对它指向的文件的各种操作。这些文件操作函数中有的最终还是调用了操作系统的API函数或者处理过程与之类似，例如在Windows系统中，fread函数就调用了API函数ReadFile。

       Windows系统的API函数除了ReadFile，还有CreateFile、WriteFile等函数。而MFC基于[面向对象的思想](http://www.jizhuomi.com/software/47.html)，将这些Windows API函数封装到了CFile类中，实现对文件的打开、关闭、读、写、获取文件信息等操作。使用CFile类对文件进行操作非常便捷。

**CFile类的成员函数**      **CFile( );  
       CFile(HANDLE hFile);  
       CFile(LPCTSTR lpszFileName,UINT nOpenFlags);**

       以上三个成员函数都是CFile的[构造函数](http://www.jizhuomi.com/software/51.html)，用于构造CFile对象。参数hFile为要关联到CFile对象的文件的句柄。参数lpszFileName为要关联到CFile对象的文件的相对路径或者绝对路径；参数nOpenFlags为文件访问选项的组合，通过各选项的按位或运算实现组合，下面的5个表列出了nOpenFlags参数可能取的选项：

       下面的文件访问模式选项表中只能选择一个进行组合，默认取CFile::modeRead。

|  |  |
| --- | --- |
| 取值 | 描述 |
| CFile::modeRead | 只读方式访问文件 |
| CFile::modeWrite | 写入方式访问文件 |
| CFile::modeReadWrite | 读写方式访问文件 |

       下面的文件共享模式选项表中也只能选择一个进行组合，默认的共享模式是CFile::shareExclusive。

|  |  |
| --- | --- |
| 取值 | 描述 |
| CFile::shareDenyNone | 允许其他进程对文件进行读写 |
| CFile::shareDenyRead | 不允许其他进程读取文件 |
| CFile::shareDenyWrite | 不允许其他进程写文件 |
| CFile::shareExclusive | 禁止其他进程对文件的所有访问 |

       下面的文件创建模式选项列表中可选择第一个或两者都选进行组合。

|  |  |
| --- | --- |
| 取值 | 描述 |
| CFile::modeCreate | 如果文件不存在则创建文件，而如果存在则将它关联到此CFile对象并将长度截取为0 |
| CFile::modeNoTruncate | 如果文件不存在则创建文件，而如果存在则将它关联到此CFile对象而不进行截取 |

       注意，选择CFile::modeNoTruncate时需要与CFile::modeCreate一起使用，即CFile::modeCreate | CFile::modeNoTruncate。

       另外，还有一个文件缓冲选项列表和一个文件安全选项。文件缓冲选项不太常用，鸡啄米这里就不讲了，有兴趣的可以查阅MSDN。文件安全选项是CFile::modeNoInherit，意为禁止子进程继承使用此文件。

       当然，在实际使用时，以上各个表并不是都要用到，大家可以根据自己的需要选择用哪个表，选择哪个选项。

**virtual BOOL Open(LPCTSTR lpszFileName,UINT nOpenFlags,CFileException\* pError = NULL);**

       打开文件。它通常与默认构造函数CFile::CFile()一起使用。参数lpszFileName和nOpenFlags同构造函数。参数pError为指向文件异常对象的指针，默认为NULL。

**virtual void Close( );**

       关闭文件。如果你没有在执行析构函数前调用此成员函数关闭文件，则析构函数会为你关闭。

**virtual UINT Read(void\* lpBuf,UINT nCount);**

       读取文件数据到缓存。参数lpBuf是由用户提供的指向接收文件数据的缓存的指针；参数nCount为读取的最大字节数。返回值是实际读取到缓存的字节数，如果到达文件尾则返回值可能会小于nCount，此时继续读取的话，会返回0，所以通常我们都会判断返回值是否小于nCount或者等于0来确定是否到达文件尾。

**virtual void Write(const void\* lpBuf,UINT nCount);**

       将缓存中的数据写入文件。参数lpBuf也是由用户提供，指向包含写入数据的缓存的指针；参数nCount为缓存中要被写入文件的数据的字节数。

**virtual ULONGLONG Seek(LONGLONG lOff,UINT nFrom);**

       在一个打开的文件中重定位文件指针。参数lOff为文件指针移动的字节个数，为正数时表示向文件尾移动，为负数时表示向文件开头移动；参数nFrom为lOff的基准位置，即由nFrom位置开始移动lOff个字节，它可以取下面几个值中的一个：  
CFile::begin       从文件开头开始移动   
CFile::current    从文件指针的当前位置开始移动   
CFile::end          从文件尾开始移动

       文件打开时，文件指针被置于0，即文件开头处。

       如果此函数成功则返回文件指针的位置。

**void SeekToBegin( );**

       将文件指针移动到文件开头。它等价于Seek( 0L, CFile::begin )。

**ULONGLONG SeekToEnd( );**

       将文件指针移动到文件末尾。返回值是文件的字节长度。它等价于CFile::Seek( 0L, CFile::end )。

**virtual ULONGLONG GetLength( ) const;**

       获取文件的字节长度。

**virtual void SetLength(ULONGLONG dwNewLen);**

       改变文件的长度。参数dwNewLen为文件的新长度，它可能比文件的当前长度值要大或者小，文件会相应的被扩展或截取。

**virtual CString GetFileName( ) const;**

       获取文件名称。

**virtual CString GetFilePath( ) const;**

       获取文件的绝对路径。

**virtual CString GetFileTitle( ) const;**

       获取文件的显示名称。举个例子，与GetFileName区分一下，如果你系统中的文件不显示扩展名，则它获取到的文件名称就不包含扩展名，否则就显示扩展名。

**virtual ULONGLONG GetPosition( ) const;**

       获取文件指针的当前位置。

**static void PASCAL Remove(LPCTSTR lpszFileName,CAtlTransactionManager\* pTM = NULL);**

       删除文件。参数lpszFileName为要删除的文件路径，可以是相对路径、绝对路径或者网络路径；参数pTM指向一个CAtlTransactionManager对象。

**static void PASCAL Rename(LPCTSTR lpszOldName,LPCTSTR lpszNewName,CAtlTransactionManager\* pTM = NULL);**

       重命名文件。参数lpszOldName为老的文件路径；参数lpszNewName为新的文件路径；参数pTM指向一个CAtlTransactionManager对象。实际上此函数的意义已经不只是重命名文件，还可以移动文件到其他目录下，例如，lpszOldName取"d:\\1.txt"，lpszNewName取"e:\\2.txt"，这样可以将D盘中的1.txt文件转移到E盘并重命名为2.txt。

**CFile类应用实例**

       这里鸡啄米只给大家演示几个简单的代码片段，从这些代码片段中熟悉CFile类的文件操作。

       实例一：构造CFile对象时就打开文件，然后向文件中写入数据，最后以Seek函数移动文件指针，读取文件内容。

1. **char** writeBuffer[500];     // 要写入的数据的缓存
2. **char** readBuffer[500];      // 存放读取数据的缓存
3. **LONGLONG** lOff = 0;         // 文件指针的偏移量，也是读取到的数据的总字节数
4. // 构造CFile对象，同时以创建和读写的方式打开文件E:\1.txt
5. CFile file(\_T("e:\\1.txt"), CFile::modeCreate | CFile::modeReadWrite);
7. // 将写入数据的缓存中每个字节都赋值为字符c
8. memset(writeBuffer, 'c', **sizeof**(writeBuffer));
9. // 将数据写入到文件中
10. file.Write(writeBuffer, **sizeof**(writeBuffer));
12. **while** (**true**)
13. {
14. // 以文件开头为基准，移动文件指针到lOff的位置
15. file.Seek(lOff, CFile::begin);
16. // 读取100个字节的数据到存放读取数据的缓存的readBuffer + lOff位置处
17. **int** nRet = file.Read(readBuffer + lOff, 100);
18. // 根据实际读取的字节数，增加文件指针的移动量
19. lOff += nRet;
20. // 如果读取数据时返回值小于指定的100，说明已到文件尾，跳出循环
21. **if** (nRet < 100)
22. **break**;
23. }
25. // 关闭文件
26. file.Close();

实际上，在Write函数和Read函数执行后，文件指针会自动移动到最后操作的位置，所以其实上面的代码中无须使用Seek函数再去手动移动文件指针。这将在下面的实例二中体现出来。

       实例二：构造CFile对象，然后使用Open成员函数打开文件，再写入一个[结构体](http://www.jizhuomi.com/software/38.html)数组，最后读取出来。

       先贴上结构体的定义：

1. **struct** student
2. {
3. **int**  nNum;
4. **float** fScore;
5. };

 下面是文件操作的代码片段：

1. student s1[2];   // 存放要写入文件的数据
2. student s2[2];   // 存放从文件读取的数据
3. CFile file;      // CFile对象
4. **int** nReadBytes = 0;   // 从文件中读取到的总字节数
6. // 为s1数组各元素赋值
7. s1[0].nNum = 22;
8. s1[0].fScore = 91.5;
9. s1[1].nNum = 23;
10. s1[1].fScore = 85;
12. // 以创建、读写方式打开文件E:\1.txt
13. **if** (file.Open(\_T("E:\\1.txt"), CFile::modeCreate | CFile::modeReadWrite))
14. {
15. // 写入数据s1结构体数组
16. file.Write(s1, **sizeof**(s1));
17. // 因为上面调用Write以后文件指针在文件尾，所以需要将其移动到文件开头
18. file.SeekToBegin();
20. **while** (**true**)
21. {
22. // 读取数据到s2
23. **int** nRet = file.Read((**BYTE**\*)s2 + nReadBytes, **sizeof**(student));
24. // 计算已经读取到的总字节数
25. nReadBytes += nRet;
26. // 如果读取数据时返回值小于指定的sizeof(student)，则说明已到文件尾，跳出循环
27. **if** (nRet < **sizeof**(student))
28. **break**;
29. }
31. // 关闭文件
32. file.Close();
33. }

## 10.5 MFC异常处理

[**MFC**](http://www.jizhuomi.com/catalog.asp?tags=MFC)**异常宏**

       MFC提供的异常处理宏包括TRY、CATCH、AND\_CATCH、END\_CATCH、THROW、THROW\_LAST等，大家看着名称是不是与C++标准异常处理的关键字相似？它们实际上就是在try、catch和throw的基础上定义的。

**MFC异常类**

       MFC将对异常的处理封装到了异常类--CException类及其子类中。其实即使我们不使用MFC异常宏而是使用C++标准异常处理的话，也是会用到MFC的CException类及其子类的。MFC异常类及其含义如下表：

|  |  |
| --- | --- |
| MFC异常类 | 含义 |
| CSimpleException | 资源紧张异常的基类 |
| CInvalidArgException | 无效参数异常 |
| CMemoryException | 内存不足 |
| CNotSupportedException | 响应对不支持服务的请求 |
| CArchiveException | 存档/序列化异常 |
| CFileException | 文件异常 |
| CResourceException | Windows 资源分配异常 |
| COleException | OLE 异常 |
| CDBException | 数据库异常（ODBC 类） |
| COleDispatchException | 调度（自动化）异常 |
| CUserException | 用消息框警告用户然后引发一般CException 的异常 |
| CDaoException | 数据库异常（DAO 类） |
| CInternetException | 网络异常 |

**MFC异常处理**

        MFC异常处理的TRY块的形式如下：  
TRY     
{     
         复合语句     
}     
CATCH (MFC异常类名, 变量名)     
{     
         复合语句     
}     
AND\_CATCH (MFC异常类名, 变量名)     
{     
         复合语句     
}     
AND\_CATCH (MFC异常类名, 变量名)     
{     
         复合语句     
}     
......     
END\_CATCH

       说明：TRY后的一对大括号内包含了可能会抛出异常的代码块；用CATCH子句捕获并处理异常，它捕获的是指向异常对象的指针，小括号中的“MFC异常类名”就是CException类或其子类的名称，变量名代表的就是“MFC异常类名”类型的指针变量；如果抛出的异常类型与CATCH子句中的不一致，则对后面的所有AND\_CATCH子句依次检查，若子句的异常类型与抛出异常类型一致则由其捕获并处理此异常；最后用END\_CATCH结束整个TRY块。

       注意：**MFC异常宏只能捕获处理CException及其子类类型的异常**。

       我们在使用MFC类时，有些会自动抛异常，当然我们可以在需要的情况下使用AfxThrow\*\*\*\*\*\*()自己抛异常，这里的\*\*\*\*\*\*与上面MFC异常类列表中的各个异常类对应，例如抛文件异常可以使用AfxThrowFileException()，参数可以查阅MSDN。

**MFC异常处理实例**

鸡啄米给大家一个简单的MFC异常处理的代码段，了解下如何使用MFC异常处理即可。

1. TRY
2. {
3. CFile file(\_T("C:\\1.txt"), CFile::modeRead);   // 构造CFile对象file,并以只读模式打开一个文件，如果不存在则抛出CFileException异常
4. }
5. CATCH (CFileException, e)
6. {
7. **if** (e->m\_cause == CFileException::fileNotFound)
8. {
9. // 如果捕获到CFileException异常且为文件未找到时，弹出提示对话框
10. MessageBox(\_T("file not found!"));
11. **return**;
12. }
13. }
14. END\_CATCH

上面这段代码的意义很简单，就是打开一个文件C:\1.txt，如果此文件不存在，则抛出CFileException异常，由CATCH子句捕获后判断是否是文件未找到，如果是则弹出提示对话框并返回。

       再将上面的代码稍微修改下，以演示AfxThrow\*\*\*\*\*\*()抛异常的用法：

1. TRY
2. {
3. AfxThrowFileException(CFileException::fileNotFound);   // 手动抛出CFileException异常
4. }
5. CATCH (CFileException, e)
6. {
7. **if** (e->m\_cause == CFileException::fileNotFound)
8. {
9. // 如果捕获到CFileException异常且为文件未找到时，弹出提示对话框
10. MessageBox(\_T("file not found!"));
11. **return** TRUE;
12. }
13. }
14. END\_CATCH

上面这段代码执行时，在CATCH子句中会捕获到文件异常。

       最后提醒大家一下，MFC的建议是不再使用MFC异常宏，而是直接使用C++标准异常，它更加灵活。

# 十一、字体和文本输出

## 11.1CFont字体类

**字体简介**

       GDI(Graphics Device Interface)，图形设备接口，是Windows提供的一些函数和结构，用于在显示器和打印机上显示图形。我们在[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)开发中经常会使用GDI来输出文本或图形图像（当然现在也有了GDI+，本教程主要讲解GDI）。文本实际上就是一种特殊的图形，它只不过是根据事先指定的“字体”绘制出来的图形。

       字体通常用来为字符集中每一个字符，如字母、数字、标点符号等，指定其形状等外表特征。窗口创建后，如果没有专门指定，一般会采用系统字体作为默认字体。我们可以使用API函数GetStockObject(SYSTEM\_FONT)获得系统字体的句柄。

**CFont类**

       CFont类封装了一个Windows图形设备接口（GDI）字体，并为操作字体提供了成员函数。

       为了使用CFont对象，首先构造一个CFont对象，再通过调用CreateFont、CreateFontIndirect、CreatePointFont或CreatePointFontIndirect将一个Windows字体与此CFont对象关联，然后使用此CFont对象的成员函数就可以操作字体了。

       一般使用CreatePointFont或CreatePointFontIndirect比使用CreateFont或CreateFontIndirect要更简便，因为前两者会自动的将字体高度的单位由点转换为逻辑单位。

**注**：“点”是传统计量字大小的单位,是从英文Point来的,一般用小写p表示,俗称“磅”。其换算关系为：1p＝0.35146mm≈0.35mm，1英寸＝72p。

       先简单解释下下面用到的几个概念：

       设备上下文是包含某个设备（如显示器、打印机）的绘制属性信息的Windows数据结构，有了它就可以在Windows中进行与设备无关的绘图，而不用考虑此设备是显示器还是打印机等。CDC类就是设备上下文类。

       在绘图时还经常会提到逻辑和物理，例如逻辑单位、物理单位，通俗点说，逻辑的就是与具体设备无关的，甚至我们可以自定义，物理的就是由具体设备决定的了。

CFont类成员函数  
**CFont( );**

       构造一个CFont对象。此对象在使用之前应该先使用CreateFont、CreateFontIndirect、CreatePointFont或CreatePointFontIndirect初始化。

**BOOL CreateFont(  
   int nHeight,  
   int nWidth,  
   int nEscapement,  
   int nOrientation,  
   int nWeight,  
   BYTE bItalic,  
   BYTE bUnderline,  
   BYTE cStrikeOut,  
   BYTE nCharSet,  
   BYTE nOutPrecision,  
   BYTE nClipPrecision,  
   BYTE nQuality,  
   BYTE nPitchAndFamily,  
   LPCTSTR lpszFacename   
);**

       通过指定的一些特征初始化CFont对象。下面分别介绍每个参数：

       nHeight：指定字体高度（逻辑单位）。有三种取值：>0，字体映射器将高度值转换为设备单位，并与可用字体的字符元高度进行匹配；=0，字体映射器使用默认的高度值；<0，字体映射器将高度值转换为设备单位，用其绝对值与可用字体的字符高度进行匹配。nHeight转换后的绝对值不应超过16384个设备单位。

       nWidth：指定字体中字符的平均宽度（逻辑单位）。

       nEscapement：指定偏离垂线和显示界面X轴之间的角度，以十分之一度为单位。偏离垂线是穿过一行文本中第一个字符和最后一个字符的直线。

       nOrientation：指定每个字符的基线和设备X轴之间的角度，以十分之一度为单位。

       nWeight：指定字体磅数（每1000点中墨点像素数）。可取0到1000之间的任意整数值。

       bItalic：指定字体是否为斜体。

       bUnderline：指定字体是否带有下划线。

       bStrikeOut：指定字体是否带有删除线。

       nCharSet：指定字体的字符集。预定义的字符集：

       ANSI\_CHARSET;BALTIC\_CHARSET;CHINESEBIG5\_CHARSET;DEFAULT\_CHARSET;EASTEUROPE\_CHARSET; GB2312\_CHARSET; GREEK\_CHARSET;HANGUL\_CHARSET; MAC\_CHARSET; OEM\_CHARSET; RUSSIAN\_CHARSET; SHIFTJIS\_CHARSET;SYMBOL\_CHARSET; TURKISH\_CHARSET。韩国Windows：JOHAB\_CHARSET；中东地区Windows：HEBREW\_CHARSSET，ARABIC\_CHARSET；泰国Windows：THAI\_CHARSET。应用程序可以使用DEFAULT\_CHARSET以允许字体名和大小完全指定逻辑字体，如果指定的字体名不存在则可能会用任意字符集的字体来代替，所以为避免不可预料的结果，应谨慎使用DEFAULT\_CHARSET。

       nOutPrecision：指定输出精度。输出精度定义了输出与要求的字体高度、宽度、字符方向、移位和间距等的接近程度。它的取值及含义如下（只能取其一）：

       OUT\_CHARACTER\_PRECIS；未用。 　　  
OUT\_DEFAULT\_PRECIS：指定缺省的字体映射器状态。 　　  
OUT\_DEVICE\_PRECIS：在当系统里有多种字体使用同一个名字时指示字体映射器选择一种设备字体。 　　  
OUT\_OUTLINE\_PRCIS：在Windows NT中此值指示字体映射器从TrueType和其他基于边框的字体中选择。 　　  
OUT\_RASTER\_PRECIS：在当系统里有多种字体使用同一个名字时指示字体映射器选择一种光栅字体。 　　  
OUT\_STRING\_PRECIS：此值没有被字体映射器使用，但是当列举光栅字体时它会被返回。  　　  
OUT\_STROKE\_PRECIS：没有被字体映射器使用，但是当列举TrueType字体、其他基于边框的字体和向量字体时它会被返回。　OUT\_TT\_ONLY\_PRECIS：指示字体映射器仅从TrueType字体中选择，如果系统中没有安装TrueType字体，则字体映射返回缺省状态。　　  
OUT\_TT\_PRECIS：在当系统里有多种同名的字体时指示字体映射器选择一种TrueType字体。当操作系统含有多种与指定名字同名的字体时，应用程序可以使用OUT\_DEVICE\_PRECIS，OUT\_RASTER\_PRECIS和OUT\_TT\_PRECIS值来控制字体映射器如何选择一种字体，例如，如果操作系统含有名字Symbol的光栅和TrueType两种字体，指定OUT\_TT\_PRECIS使字体映射器选择TrueType方式（指定OUT\_TT\_ONLY\_PRECIS强制字体映射器选择一种TrueType字体，尽管这会给TrueType字体换一个名字）。

       nClipPrecision：指定裁剪精度。裁剪精度定义了怎样裁剪部分超出裁剪区域的字符。它的取值及含义如下（可取一个或多个值）：

　　CLIP\_DEFAULT\_PRECIS：指定缺省裁剪状态。  
CLIP\_CHARACTER\_PRECIS：未用。 　　  
CLIP\_STROKE\_PRECIS：未被字体映射器使用，但是当列举光栅字体、向量字体或TrueType字体时它会被返回。在Windows环境下，为保证兼容性，当列举字体时这个值总被返回。 　　  
CLIP\_MASK：未用。  
CLIP\_EMBEDDED：要使用嵌入式只读字体必须使用此标志。 　　  
CLIP\_LH\_ANGLES：当此值被使用时，所有字体的旋转依赖于坐标系统的定位是朝左的还是朝右的。如果未使用此值，设备字体总是逆时针方向旋转，但其他字体的旋转依赖于坐标系统的定向。 　　  
CLIP\_TT\_ALWAYS：未用。

       nQuality：指定字体的输出质量。输出质量定义了GDI将逻辑字体属性匹配到实际物理字体的细致程度。它的各个取值及含义如下（取其一）：

       DEFAULT\_QUALITY：字体的外观不重要。 　　  
DRAFT\_QUALITY：字体外观的重要性次于使用PROOF\_QUALITY时，对GDI光栅字体，缩放比例是活动的，这意味着多种字体大小可供选择，但质量可能不高，如果有必要，粗体、斜体、下划线、strikeout字体可被综合起来使用。 　　  
PROOF\_QUALITY：字符质量比精确匹配逻辑字体字体属性更重要。对GDI扫描字体，缩放比例是活动的，并选择最接近的大小。尽管当使用PROOF\_QUALITY时，选择字体大小并不完全匹配，但字体的质量很高，并没有外观上的变形。如果有必要，粗体、斜体、下划线、strikeout字体可被综合起来使用。

       nPitchAndFamily：指定字体间距和字体族。低2位用来指定字体的间距，可取下列值中的一个：DEFAULT\_PITCH，FIXED\_PITCH，VARIABLE\_PITCH。高4位指定字体族，取值及含义如下（取其一）：  
FF\_DECORATIVE：新奇的字体，如老式英语（Old English）。  
FF\_DONTCARE：不关心或不知道。 　　  
FF\_MDERN：笔划宽度固定的字体，有或者无衬线。如Pica、Elite和Courier New。 　　  
FF\_ROMAN：笔划宽度变动的字体，有衬线。如MS Serif。 　　  
FF\_SCRIPT：设计成看上去象手写体的字体。如Script和Cursive。 　　  
FF\_SWISS：笔划宽度变动的字体，无斜线。如MS Sans Serif。

       应用程序可以用运算符OR将字符间距和字体族组合起来给nPitchAndFamily赋值。

       字体族描述一种字体的普通外观，当所有的精确字样都不能使用时，可用它们来指定字体。

       lpszFacename：指定字体的字样名的字符串。此字符串的长度不应超过30个字符。Windows函数EnumFontFamilies可以枚举出当前所有可用字体的字样名。如果lpszFacename为NULL，则GDI使用一种与设备无关的字体。

       返回值：此函数成功则返回TRUE，否则返回FALSE。

       CreateFont函数初始化CFont对象后，此字体就能够被选作任何设备上下文的字体了。此函数并不会创建一个新的Windows GDI字体，只是从GDI的物理字体中选择了一个最匹配的字体。在创建一个逻辑字体时，大部分参数可以使用默认值，但一般情况下都会给出参数nHeight和lpszFacename的指定值，如果没有给nHeight和lpszFacename参数设定取值，则创建的逻辑字体与设备相关。当使用CreateFont函数初始化一个CFont对象完成后，就能够使用CDC::SelectObject函数来为设备上下文选择字体了，并且还能够在不再使用此CFont对象时删除它。

**BOOL CreateFontIndirect(const LOGFONT\* lpLogFont);**

       通过一个LOGFONT结构体变量给出的特征来初始化CFont对象。参数lpLogFont是指向LOGFONT结构体变量的指针，此LOGFONT结构体变定义了逻辑字体的特征。LOGFONT结构体的定义可以参见字体对话框部分知识。

**BOOL CreatePointFont(int nPointSize,LPCTSTR lpszFaceName,CDC\* pDC = NULL);**

       此函数提供了一种由指定字样和点数创建字体的简单方式。参数的意义如下：

       nPointSize：指定字体高度，以十分之一点为单位。例如，nPointSize为120则表示是12点的字体。

       lpszFacename：指定字体的字样名的字符串。此字符串的长度不应超过30个字符。Windows函数EnumFontFamilies可以枚举出当前所有可用字体的字样名。如果lpszFacename为NULL，则GDI使用一种与设备无关的字体。

       pDC：指向CDC对象，用来将nPointSize指定的高度转换为逻辑单位，如果为NULL，则使用屏幕设备上下文进行转换。

**BOOL CreatePointFontIndirect(const LOGFONT\* lpLogFont,CDC\* pDC = NULL);**

       此函数是通过指定的字样和点数创建字体的间接方式。参数lpLogFont指向一个LOGFONT结构体变量，此LOGFONT变量定义了逻辑字体的特征，它的lfHeight成员以十分之一点为单位，而不是逻辑单位。参数pDC指向CDC对象，用来将lfHeight表示的高度转换为逻辑单位，如果为NULL，则使用屏幕设备上下文进行转换。

       此函数与CreateFontIndirect很相似，但区别是LOGFONT变量中lfHeight成员的单位是十分之一点而不是逻辑单位。

**static CFont\* PASCAL FromHandle(HFONT hFont);**

       由Windows GDI字体的HFONT句柄获得相应的CFont对象指针。参数hFont是一个Windows字体的HFONT句柄。成功则返回CFont对象的指针，否则返回NULL。

**int GetLogFont(LOGFONT \* pLogFont);**

       获取CFont对象的LOGFONT结构体的拷贝。参数pLogFont指向用来接收字体信息的LOGFONT结构体变量。成功则返回非零值，否则返回零。

## 11.2文本输出

**文本输出过程**

       在文本输出到设备以前，我们需要确定字体、字体颜色和输出的文本内容等信息。Windows窗口的客户区由应用程序管理，所以我们还要在应用程序中控制输出文本的格式，例如后续字符的位置、换行等格式。

       由此，文本的输出过程大致包括确定字体信息、格式化文本和执行输出操作三个步骤。下面分别讲解。

**1、确定字体信息**

       文本在输出以前应该先确定字体信息，或者是当前正在使用的字体，或者是自定义的字体，之后就可以根据确定的字体来显示文本或者利用字体信息来设定文本的格式了，例如，我们可以根据当前字体的字符高度来确定下一行字符在什么位置输出。

       自定义字体可以通过CFont类的创建字体的几个成员函数完成。获取当前选择字体的信息可以使用API函数GetTextMetrics实现，此函数的原型如下：

       BOOL GetTextMetrics(\_\_in   HDC hdc,\_\_out  LPTEXTMETRIC lptm);

       参数hdc为设备上下文的句柄；参数lptm是指向TEXTMETRIC结构体变量的指针，此结构体变量用于接收字体信息。TEXTMETRIC结构体的定义如下：

1. **typedef** **struct** tagTEXTMETRIC {
2. **LONG**  tmHeight;        // 字符高度
3. **LONG**  tmAscent;        // 字符基线以上的高度
4. **LONG**  tmDescent;       // 字符基线以下的高度
5. **LONG**  tmInternalLeading; // 由tmHeight成员指定的字符高度顶部的空间
6. **LONG**  tmExternalLeading; // 行间距
7. **LONG**  tmAveCharWidth;  // 字符的平均宽度
8. **LONG**  tmMaxCharWidth;  // 字符的最大宽度
9. **LONG**  tmWeight;        // 字符的粗度
10. **LONG**  tmOverhang;      // 合成字体间附加的宽度
11. **LONG**  tmDigitizedAspectX; // 为输出设备设计的x轴尺寸
12. **LONG**  tmDigitizedAspectY; // 为输出设备设计的y轴尺寸
13. **TCHAR** tmFirstChar;     // 字体中第一个字符值
14. **TCHAR** tmLastChar;      // 字体中最后一个字符值
15. **TCHAR** tmDefaultChar;   // 替换字体中没有的字符
16. **TCHAR** tmBreakChar;     // 作为分隔符的字符
17. **BYTE**  tmItalic;        // 非0则表示字体为斜体
18. **BYTE**  tmUnderlined;    // 非0则表示字体有下划线
19. **BYTE**  tmStruckOut;     // 非0则表示字符带有删除线
20. **BYTE**  tmPitchAndFamily;// 字体间距和字体族
21. **BYTE**  tmCharSet;       // 字符集
22. } TEXTMETRIC, \*PTEXTMETRIC;

**2、格式化文本**

       格式化文本一般包括两种，一种是确定文本行中后续文本的位置，另一种是确定换行时下一行文本的位置。

**确定后续文本的位置**

       一般我们可以先获取当前字符串的宽度，根据此宽度确定文本行中后续文本的位置。当前字符串的宽度可以通过API函数GetTextExtentPoint32获得。GetTextExtentPoint32函数的原型如下：

       BOOL GetTextExtentPoint32(\_\_in   HDC hdc,\_\_in   LPCTSTR lpString,\_\_in   int c,\_\_out  LPSIZE lpSize);

       参数hdc为设备上下文的句柄；参数lpString为指向文本字符串缓存的指针，此字符串不是必须以结束符结尾的，因为参数c指定了长度；参数c为lpString指向的字符串的长度；参数lpSize为指向SIZE结构体变量的指针，此SIZE结构体变量用于接收字符串的宽度和高度信息。SIZE结构体定义如下：

1. **typedef** **struct** tagSIZE {
2. **LONG** cx;   // 宽度
3. **LONG** cy;   // 高度
4. } SIZE, \*PSIZE;

已知本字符串的起始水平坐标和宽度，两者相加即是后续文本的起始坐标。

**确定换行时下一行文本的位置**

       由GetTextMetrics函数获取了当前字体的信息并存入TEXTMETRIC结构体后，通过计算当前文本行的垂直坐标、当前字体的高度和行间距之和，就可以得到换行时下一行的垂直坐标。

**3、执行文本输出操作**

       最后，通过API函数TextOut执行文本输出操作。TextOut函数的原型如下：

       BOOL TextOut(\_\_in  HDC hdc,\_\_in  int nXStart,\_\_in  int nYStart,\_\_in  LPCTSTR lpString,\_\_in  int cbString);

       参数hdc为设备上下文的句柄；参数nXStart为起始点x坐标；参数nYStart为起始点y坐标；参数lpString为要输出的文本字符串；参数cbString为字符串中要输出的字符的数量。

       当然也可以使用设备上下文类CDC的成员函数TextOut来输出，CDC::TextOut函数的两种重载形式如下：

       virtual BOOL TextOut(int x,int y,LPCTSTR lpszString,int nCount);  
       BOOL TextOut(int x,int y,const CString& str);

       参数x指定文本起始点的x坐标；参数y指定文本起始点的y坐标；参数lpszString为要输出的文本字符串；参数nCount指定字符串中的字节个数；参数str为包含要输出的字符的CString对象。

# 十二、图形图像

## 12.1CDC类及其屏幕绘图函数

**CDC类简介**

       CDC类是一个设备上下文类。

       CDC类提供了用来处理显示器或打印机等设备上下文的成员函数，还有处理与窗口客户区关联的显示上下文的成员函数。使用CDC的成员函数可以进行所有的绘图操作，包括处理绘图工具、GDI对象的选择、颜色和调色板的处理、获取和设置绘图属性、映射、窗口范围、坐标转换、剪切以及绘制直线、简单图形、椭圆和多边形等，另外它还为文本输出、处理字体、使用打印机跳转和滚动等提供了成员函数。

       如上所述，CDC类几乎封装了所有的Windows GDI函数，另外，[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)中还有几个由CDC类派生的子类，包括CWindowDC、CPaintDC、CClientDC、CMetaFileDC，它们用来进行一些特定的绘图操作。

       一般我们在使用完CDC对象后要记得删除它，否则会有内存泄露。很多情况下我们可以调用CWnd::GetDC()函数来获取设备上下文指针，即CDC指针，这个时候记得用完后调用CWnd::ReleaseDC()函数释放设备上下文。

**CDC类的屏幕绘图成员函数**

       CDC类有很多成员函数，鸡啄米在这里只大概讲下比较常用的绘图函数，包括绘制点、直线、矩形、椭圆、多边形、文本以及位图等的成员函数。

**COLORREF SetPixel(int x,int y,COLORREF crColor);  
       COLORREF SetPixel(POINT point,COLORREF crColor);**

       上面两个成员函数用来将指定坐标点的像素设置为指定的颜色，这样就实现了画点功能。参数x为点的逻辑x坐标；参数y为点的逻辑y坐标；参数crColor为要为点设置的颜色；参数point指定点的逻辑x坐标和逻辑y坐标，可以为其传入POINT结构体变量或者CPoint对象。

**CPoint MoveTo(int x,int y);  
       CPoint MoveTo(POINT point);**

       将当前点移动到指定位置。参数x指定新位置的逻辑x坐标；参数y指定新位置的逻辑y坐标；参数point指定新位置的逻辑x坐标和逻辑y坐标，可以为其传入POINT结构体变量或者CPoint对象。

**BOOL LineTo(int x,int y);  
       BOOL LineTo(POINT point);**

       绘制一条从当前点到指定点（不包括指定点）的直线。参数x为指定点的逻辑x坐标；参数y为指定点的逻辑y坐标；参数point为指定点的逻辑x坐标和逻辑y坐标。一般我们绘制直线时就可以先调用MoveTo函数移动当前点到某个位置，然后调用LineTo画直线。

**BOOL Rectangle(int x1,int y1,int x2,int y2);  
       BOOL Rectangle(LPCRECT lpRect);**

       使用当前画笔绘制矩形。参数x1指定矩形左上角的x坐标；参数y1指定矩形左上角的y坐标；参数x2指定矩形右下角的x坐标；参数y2指定矩形右下角的y坐标；以上坐标均为逻辑单位。参数lpRect为矩形对象的指针，可以为其传入CRect对象或RECT结构体变量的指针。

**BOOL Ellipse(int x1,int y1,int x2,int y2);  
       BOOL Ellipse(LPCRECT lpRect);**

       绘制椭圆。参数x1指定椭圆的包围矩形左上角的x坐标；参数y1指定椭圆的包围矩形左上角的y坐标；参数x2指定椭圆的包围矩形右下角的x坐标；参数y2指定椭圆的包围矩形右下角的y坐标；以上坐标均为逻辑单位。参数lpRect指定椭圆的包围矩形，可以传入CRect对象或RECT结构体变量的指针。

**BOOL Polyline(LPPOINT lpPoints,int nCount);**

       由指定的多边形顶点绘制多边形。参数lpPoints为指向一个POINT结构体变量数组或CPoint对象数组的指针，其中的POINT结构体变量或CPoint对象代表了多边形顶点的坐标；参数nCount为数组中点的个数，至少为2。

**virtual BOOL TextOut(int x,int y,LPCTSTR lpszString,int nCount);  
       BOOL TextOut(int x,int y,const CString& str);**

       使用当前选择的字体在指定位置输出文本。 参数x指定文本起始点的x坐标；参数y指定文本起始点的y坐标；参数lpszString为要输出的文本字符串；参数nCount指定字符串中的字节个数；参数str为包含要输出的字符的CString对象。这两个函数在上一节中其实已经讲到了。  
       **BOOL BitBlt(  
              int x,  
              int y,  
              int nWidth,  
              int nHeight,  
              CDC\* pSrcDC,  
              int xSrc,  
              int ySrc,  
              DWORD dwRop   
       );**

       从源设备上下文拷贝一幅位图到当前设备上下文。参数x指定目标矩形区域左上角的逻辑x坐标；参数y指定目标矩形区域左上角的逻辑y坐标；参数nWidth指定目标矩形区域和源位图的宽度（逻辑单位）；参数nHeight指定目标矩形区域和源位图的高度（逻辑单位）；参数pSrcDC为指向源设备上下文的CDC对象的指针，如果dwRop指定了一个不包含源的光栅操作，那么pSrcDC可以为NULL；参数xSrc指定源位图左上角的逻辑x坐标；参数ySrc指定源位图左上角的逻辑y坐标；参数dwRop指定要执行的光栅操作，光栅操作码定义了GDI如何将当前画刷颜色、源位图颜色和目标位图颜色组合形成新的颜色，下面是一些常用的光栅操作码及含义：

       BLACKNESS：表示使用与物理调色板的索引0相关的色彩来填充目标矩形区域，（对缺省的物理调色板而言，该颜色为黑色）。   
DSTINVERT：表示使目标矩形区域颜色取反。   
MERGECOPY：表示使用布尔型的AND（与）操作符将源矩形区域的颜色与特定模式组合一起。   
MERGEPAINT：通过使用布尔型的OR（或）操作符将反向的源矩形区域的颜色与目标矩形区域的颜色合并。   
NOTSRCCOPY：将源矩形区域颜色取反，于拷贝到目标矩形区域。   
NOTSRCERASE：使用布尔类型的OR（或）操作符组合源和目标矩形区域的颜色值，然后将合成的颜色取反。   
PATCOPY：将特定的模式拷贝到目标位图上。   
PATPAINT：通过使用布尔OR（或）操作符将源矩形区域取反后的颜色值与特定模式的颜色合并。然后使用OR（或）操作符将该操作的结果与目标矩形区域内的颜色合并。  
PATINVERT：通过使用XOR（异或）操作符将源和目标矩形区域内的颜色合并。   
SRCAND：通过使用AND（与）操作符来将源和目标矩形区域内的颜色合并。   
SRCCOPY：将源矩形区域直接拷贝到目标矩形区域。   
SRCERASE：通过使用AND（与）操作符将目标矩形区域颜色取反后与源矩形区域的颜色值合并。   
SRCINVERT：通过使用布尔型的XOR（异或）操作符将源和目标矩形区域的颜色合并。   
SRCPAINT：通过使用布尔型的OR（或）操作符将源和目标矩形区域的颜色合并。   
WHITENESS：使用与物理调色板中索引1有关的颜色填充目标矩形区域。（对于缺省物理调色板来说，这个颜色就是白色）。

## 12.2GDI对象之画笔CPen

**GDI对象**

       在[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)中，CGdiObject类是GDI对象的基类，通过查阅MSDN我们可以看到，CGdiObject类有六个直接的派生类，GDI对象主要也是这六个，分别是：CBitmap、CBrush、CFont、CPalette、CPen和CRgn。在这六个GDI对象中，最常用的莫过于画笔和画刷了.

**CPen应用实例**见MFCExample实例

## 12.3GDI对象之画刷CBrush

CClientDC类  
CClientDC类是CDC类的派生类，它生成的设备上下文对应于窗口的客户区，另一个类CWindowDC也是CDC类的派生类，它对应的则包括客户区和非客户区，也就是说，CClientDC只能在客户区绘图，而CWindowDC可在客户区和非客户区绘图。

到底什么是客户区，什么是非客户区？非客户区包括窗口中的标题栏、菜单栏、状态栏、滚动条、边框等区域，客户区就是除去非客户区之外的区域。

# 十三、Ribbon界面开发

## 13.1创建Ribbon样式的应用程序框架

Ribbon界面就是微软从Office2007开始引入的一种为了使应用程序的功能更加易于发现和使用、减少了点击鼠标的次数的新型界面，从实际效果来看，不仅外观漂亮，而且功能直观，用户操作简洁方便。Ribbon的中文解释也称作“功能区”。

## 13.2为Ribbon Bar添加控件

Ribbon的界面元素可以分为类别、面板和基本控件（按钮、文本编辑框等），类别由面板组成，面板又由按钮、文本编辑框等基本控件组成。（联想Office2007的界面设计）

Ribbon控件的图标可以通过图片导入，图标图片实际上是若干个icon左右并列组成的图像序列。图像序列可以使用IconWorkshop来加工多个图标图片来制作。

Ribbon界面的效果测试可以不用编译运行程序，而使用工具栏按钮“Test Ribbon”

## 13.3使用更多控件并为其添加消息处理函数

很多的Ribbon菜单都可以通过其菜单按钮的“属性”中的“行为”部分设置其菜单内容。