# Visual C++MFC入门教程

## 第一章 VC入门

### 1.1 如何学好VC

这个问题很多朋友都问过我，当然流汗是必须的，但同时如果按照某种思路进行有计划的学习就会起到更好的效果。万事开头难，为了帮助朋友们更快的掌握VC开发，下面我将自己的一点体会讲一下：

1、需要有好的C/C++基础。正所谓“磨刀不误砍柴工”，最开始接触VC时不要急于开始Windows程序开发，而是应该进行一些字符界面程序的编写。这样做的目的主要是增加对语言的熟悉程度，同时也训练自己的思维和熟悉一些在编程中常犯的错误。更重要的是理解并能运用C++的各种特性，这些在以后的开发中都会有很大的帮助，特别是利用MFC进行开发的朋友对C++一定要能熟练运用。

2、理解Windows的消息机制，窗口句柄和其他GUI句柄的含义和用途。了解和MFC各个类功能相近的API函数。

3、一定要理解MFC中消息映射的作用。

4、训练自己在编写代码时不使用参考书而是使用Help Online。

5、记住一些常用的消息名称和参数的意义。

6、学会看别人的代码。

7、多看书，少买书，买书前一定要慎重。

8、闲下来的时候就看参考书。

9、多来我的主页。^O^

后面几条是我个人的一点意见，你可以根据需要和自身的情况选用适用于自己的方法。

此外我将一些我在选择参考书时的原则：

对于初学者：应该选择一些内容比较全面的书籍，并且书籍中的内容应该以合理的方式安排，在使用该书时可以达到循序渐进的效果，书中的代码要有详细的讲解。尽量买翻译的书，因为这些书一般都比较易懂，而且语言比较轻松。买书前一定要慎重如果买到不好用的书可能会对自己的学习积极性产生击。

对于已经掌握了VC的朋友：这种程度的开发者应该加深自己对系统原理，技术要点的认识。需要选择一些对原理讲解的比较透彻的书籍，这样一来才会对新技术有更多的了解，最好书中对技术的应用有一定的阐述。尽量选择示范代码必较精简的书，可以节约银子。

此外最好涉猎一些辅助性的书籍。

## 1.2 理解Windows消息机制

Windows系统是一个消息驱动的OS，什么是消息呢？我很难说得清楚，也很难下一个定义（谁在嘘我），我下面从不同的几个方面讲解一下，希望大家看了后有一点了解。

1、消息的组成：一个消息由一个消息名称（UINT），和两个参数（WPARAM，LPARAM）。当用户进行了输入或是窗口的状态发生改变时系统都会发送消息到某一个窗口。例如当菜单选中之后会有WM\_COMMAND消息发送，WPARAM的高字中（HIWORD(wParam)）是命令的ID号，对菜单来讲就是菜单ID。当然用户也可以定义自己的消息名称，也可以利用自定义消息来发送通知和传送数据。

2、谁将收到消息：一个消息必须由一个窗口接收。在窗口的过程（WNDPROC）中可以对消息进行分析，对自己感兴趣的消息进行处理。例如你希望对菜单选择进行处理那么你可以定义对WM\_COMMAND进行处理的代码，如果希望在窗口中进行图形输出就必须对WM\_PAINT进行处理。

3、未处理的消息到那里去了：M$为窗口编写了默认的窗口过程，这个窗口过程将负责处理那些你不处理消息。正因为有了这个默认窗口过程我们才可以利用Windows的窗口进行开发而不必过多关注窗口各种消息的处理。例如窗口在被拖动时会有很多消息发送，而我们都可以不予理睬让系统自己去处理。

4、窗口句柄：说到消息就不能不说窗口句柄，系统通过窗口句柄来在整个系统中唯一标识一个窗口，发送一个消息时必须指定一个窗口句柄表明该消息由那个窗口接收。而每个窗口都会有自己的窗口过程，所以用户的输入就会被正确的处理。例如有两个窗口共用一个窗口过程代码，你在窗口一上按下鼠标时消息就会通过窗口一的句柄被发送到窗口一而不是窗口二。

5、示例：下面有一段伪代码演示如何在窗口过程中处理消息

LONG yourWndProc(HWND hWnd,UINT uMessageType,WPARAM wP,LPARAM)

{

switch(uMessageType)

{ //使用SWITCH语句将各种消息分开

case(WM\_PAINT):

doYourWindow(...);//在窗口需要重新绘制时进行输出

break;

case(WM\_LBUTTONDOWN):

doYourWork(...);//在鼠标左键被按下时进行处理

break;

default:

callDefaultWndProc(...);//对于其它情况就让系统自己处理

break;

}

}

接下来谈谈什么是消息机制：系统将会维护一个或多个消息队列，所有产生的消息都回被放入或是插入队列中。系统会在队列中取出每一条消息，根据消息的接收句柄而将该消息发送给拥有该窗口的程序的消息循环。每一个运行的程序都有自己的消息循环，在循环中得到属于自己的消息并根据接收窗口的句柄调用相应的窗口过程。而在没有消息时消息循环就将控制权交给系统所以Windows可以同时进行多个任务。下面的伪代码演示了消息循环的用法：

while(1)

{

id=getMessage(...);

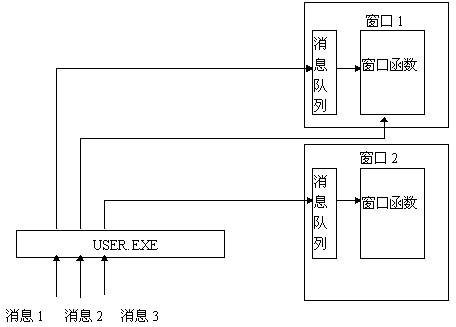
if(id == quit)

break;

translateMessage(...);

}

当该程序没有消息通知时getMessage就不会返回，也就不会占用系统的CPU时间。 下图为消息投递模式



在16位的系统中系统中只有一个消息队列，所以系统必须等待当前任务处理消息后才可以发送下一消息到相应程序，如果一个程序陷如死循环或是耗时操作时系统就会得不到控制权。这种多任务系统也就称为协同式的多任务系统。Windows3.X就是这种系统。而32位的系统中每一运行的程序都会有一个消息队列，所以系统可以在多个消息队列中转换而不必等待当前程序完成消息处理就可以得到控制权。这种多任务系统就称为抢先式的多任务系统。Windows95/NT就是这种系统。

## 1.3 利用Visual C++/MFC开发Windows程序的优势

MFC借助C++的优势为Windows开发开辟了一片新天地，同时也借助ApplicationWizard使开发者摆脱离了那些每次都必写基本代码，借助ClassWizard和消息映射使开发者摆脱了定义消息处理时那种混乱和冗长的代码段。更令人兴奋的是利用C++的封装功能使开发者摆脱Windows中各种句柄的困扰，只需要面对C++中的对象，这样一来使开发更接近开发语言而远离系统。（但我个人认为了解系统原理对开发很有帮助）

正因为MFC是建立在C++的基础上，所以我强调C/C++语言基础对开发的重要性。利用C++的封装性开发者可以更容易理解和操作各种窗口对象；利用C++的派生性开发者可以减少开发自定义窗口的时间和创造出可重用的代码；利用虚拟性可以在必要时更好的控制窗口的活动。而且C++本身所具备的超越C语言的特性都可以使开发者编写出更易用，更灵活的代码。

在MFC中对消息的处理利用了消息映射的方法，该方法的基础是宏定义实现，通过宏定义将消息分派到不同的成员函数进行处理。下面简单讲述一下这种方法的实现方法：

代码如下

BEGIN\_MESSAGE\_MAP(CMainFrame, CFrameWnd)

//{{AFX\_MSG\_MAP(CMainFrame)

ON\_WM\_CREATE()

//}}AFX\_MSG\_MAP

ON\_COMMAND(ID\_FONT\_DROPDOWN, DoNothing)

END\_MESSAGE\_MAP()

经过编译后，代码被替换为如下形式（这只是作讲解，实际情况比这复杂得多）：

//BEGIN\_MESSAGE\_MAP(CMainFrame, CFrameWnd)

CMainFrame::newWndProc(...)

{

switch(...)

{

//{{AFX\_MSG\_MAP(CMainFrame)

// ON\_WM\_CREATE()

case(WM\_CREATE):

OnCreate(...);

break;

//}}AFX\_MSG\_MAP

// ON\_COMMAND(ID\_FONT\_DROPDOWN, DoNothing)

case(WM\_COMMAND):

if(HIWORD(wP)==ID\_FONT\_DROPDOWN)

{

DoNothing(...);

}

break;

//END\_MESSAGE\_MAP()

}

}

newWndProc就是窗口过程只要是该类的实例生成的窗口都使用该窗口过程。

所以了解了Windows的消息机制在加上对消息映射的理解就很容易了解MFC开发的基本思路了。

## 1.4 利用MFC进行开发的通用方法介绍

以下是我在最初学习VC时所常用的开发思路和方法，希望能对初学VC的朋友有所帮助和启发。

1、开发需要读写文件的应用程序并且有简单的输入和输出可以利用单文档视结构。

2、开发注重交互的简单应用程序可以使用对话框为基础的窗口，如果文件读写简单这可利用CFile进行。

3、开发注重交互并且文件读写复杂的的简单应用程序可以利用以CFormView为基础视的单文档视结构。

4、利用对话框得到用户输入的数据，在等级提高后可使用就地输入。

5、在对多文档要求不强烈时尽量避免多文档视结构，可以利用分隔条产生单文档多视结构。

6、在要求在多个文档间传递数据时使用多文档视结构。

7、学会利用子窗口，并在自定义的子窗口包含多个控件达到封装功能的目的。

8、尽量避免使用多文档多视结构。

9、不要使用多重继承并尽量减少一个类中封装过多的功能。

## 1.5 MFC中常用类，宏，函数介绍

常用类

CRect：用来表示矩形的类，拥有四个成员变量：top left bottom right。分别表是左上角和右下角的坐标。可以通过以下的方法构造：

CRect( int l, int t, int r, int b ); 指明四个坐标

CRect( const RECT& srcRect ); 由RECT结构构造

CRect( LPCRECT lpSrcRect ); 由RECT结构构造

CRect( POINT point, SIZE size ); 有左上角坐标和尺寸构造

CRect( POINT topLeft, POINT bottomRight ); 有两点坐标构造

下面介绍几个成员函数：

int Width( ) const; 得到宽度

int Height( ) const; 得到高度

CSize Size( ) const; 得到尺寸

CPoint& TopLeft( ); 得到左上角坐标

CPoint& BottomRight( ); 得到右下角坐标

CPoint CenterPoint( ) const; 得当中心坐标

此外矩形可以和点（CPoint）相加进行位移，和另一个矩形相加得到“并”操作后的矩形。

CPoint：用来表示一个点的坐标，有两个成员变量：x y。 可以和另一个点相加。

CString：用来表示可变长度的字符串。使用CString可不指明内存大小，CString会根据需要自行分配。下面介绍几个成员函数：

GetLength 得到字符串长度

GetAt 得到指定位置处的字符

operator + 相当于strcat

void Format( LPCTSTR lpszFormat, ... ); 相当于sprintf

Find 查找指定字符，字符串

Compare 比较

CompareNoCase 不区分大小写比较

MakeUpper 改为小写

MakeLower 改为大写

CStringArray：用来表示可变长度的字符串数组。数组中每一个元素为CString对象的实例。下面介绍几个成员函数：

Add 增加CString

RemoveAt 删除指定位置CString对象

RemoveAll 删除数组中所有CString对象

GetAt 得到指定位置的CString对象

SetAt 修改指定位置的CString对象

InsertAt 在某一位置插入CString对象

常用宏

RGB

TRACE

ASSERT

VERIFY

常用函数

CWindApp\* AfxGetApp();

HINSTANCE AfxGetInstanceHandle( );

HINSTANCE AfxGetResourceHandle( );

int AfxMessageBox( LPCTSTR lpszText, UINT nType = MB\_OK, UINT nIDHelp = 0 );用于弹出一个消息框

## 第二章 图形输出

### 2.1 和GUI有关的各种对象

### 2.2 在窗口中输出文字

### 2.3 使用点，刷子，笔进行绘图

### 2.4 在窗口中绘制设备相关位图，图标，设备无关位图

### 2.5 使用各种映射方式

### 2.6 多边形和剪贴区域

## 第三章 文档视结构

### 3.1 文档 视图 框架窗口间的关系和消息传送规律

### 3.2 接收用户输入

### 3.3 使用菜单

### 3.4 文档，视，框架之间相互作用

### 3.5 利用序列化进行文件读写

### 3.6 MFC中所提供的各种视类介绍

## 第四章 窗口控件

### 4.1 Button

### 4.2 Static Box

### 4.3 Edit Box

### 4.4 Scroll Bar

### 4.5 List Box/Check List Box

### 4.6 Combo Box

### 4.7 Tree Ctrl

### 4.8 List Ctrl

### 4.9 Tab Ctrl

### 4.A Tool Bar

### 4.B Status Bar

### 4.C Dialog Bar

### 4.E General Window

### 4.F 关于WM\_NOTIFY的使用方法

## 第五章 对话框

### 5.1 使用资源编辑器编辑对话框

在Windows开发中弹出对话框是一种常用的输入/输出手段，同时编辑好的对话框可以保存在资源文件中。Visual C++提供了对话框编辑工具，利用编辑工具可以方便的添加各种控件到对话框中，而且利用ClassWizard可以方便的生成新的对话框类和映射消息。

首先资源列表中按下右键，可以在弹出菜单中选择“插入对话框”，[如图5.1](http://tech.china.com/zh_cn/netschool/programme/c/656/20001207/51_g1.gif)。

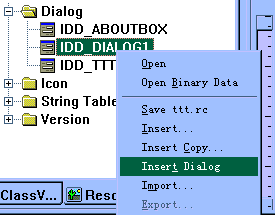


图5.1

然后再打开该对话框进行编辑，屏幕上看到一个控件板，如图5.2。

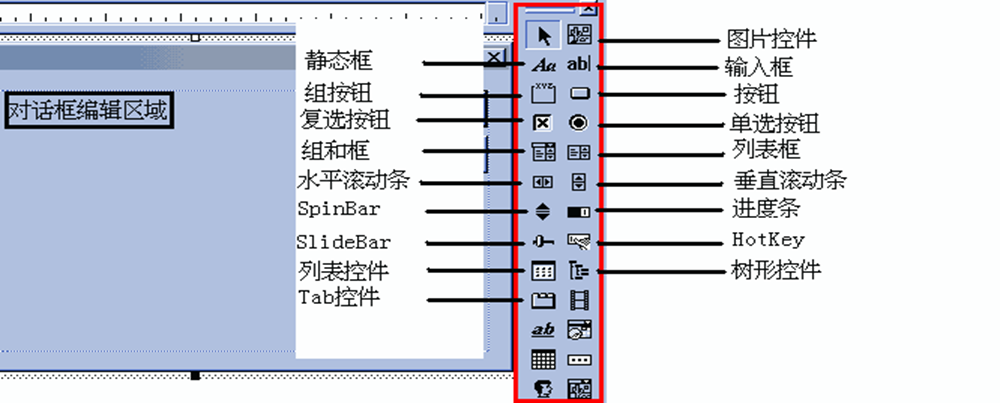


图 5.2

你可以将所需要添加的控件拖到对话框上，或是先选中后再在对话框上用鼠标画出所占的区域。

接下来我们在对话框上产生一个输入框，和一个用于显示图标的图片框。之后我们使用鼠标右键单击产生的控件并选择其属性，如图5.3。

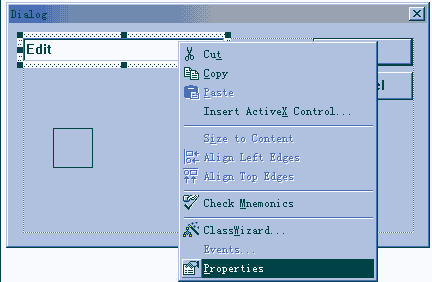


图5.3

我们可以在属性对话框中编辑控件的属性同时也需要指定控件ID，[如图5.4](http://tech.china.com/zh_cn/netschool/programme/c/656/20001207/51_g4.gif)，如果在选择对话框本身的属性那么你可以选择对话框的一些属性，包括字体，外观，是否有系统菜单等等。

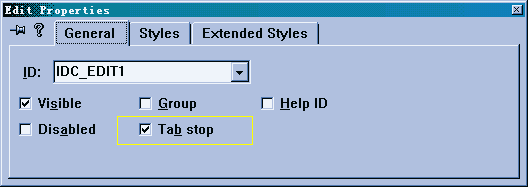


图 5.4

最后我们编辑图片控件的属性，如图5.5，我们设置控件的属性为显示图标并指明一个图标ID。

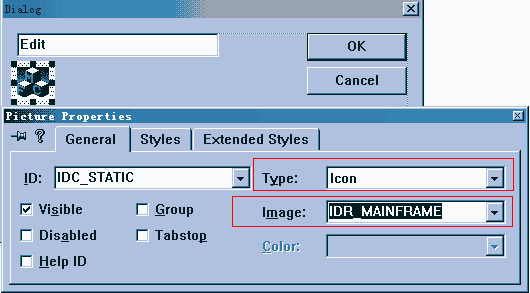


图5.5

接下来我们添加一些其他的控件，最后的效果如图5.6。

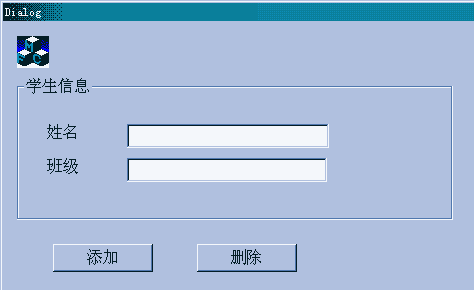


图 5.6

按下Ctrl-T可以测试该对话框。此外在对话框中还有一个有用的特性，就是可以利用Tab键让输入焦点在各个控件间移动，要达到这一点首先需要为控件设置在Tab键按下时可以接受焦点移动的属性Tab Stop，如果某一个控件不打算利用这一特性，你需要清除这一属性。然后从菜单“Layout”选择Tab Order来确定焦点移动顺序，如图5.7。

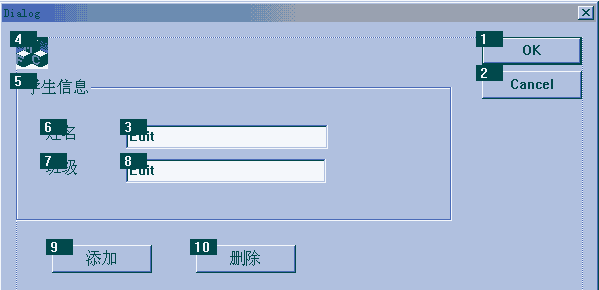


图 5.7

使用鼠标依此点击控件就可以重新规定焦点移动次序。最后按下Ctrl-T进行测试。

最后我们需要为对话框产生新的类，ClassWizard可以替我们完成大部分的工作，我们只需要填写几个参数就可以了。在编辑好的对话框上双击，然后系统回询问是否添加新的对话框，选择是并在接下来的对话框中输入类名就可以了。ClassWizard会为你产生所需要的头文件和CPP文件。然后在需要使用的地方包含相应的头文件，对于有模式对话框使用DoModal()产生，对于无模式对话框使用Create()产生。相关代码如下:

void CMy51\_s1View::OnCreateDlg()

{//产生无模式对话框

CTestDlg \*dlg=new CTestDlg;

dlg->Create(IDD\_TEST\_DLG);

dlg->ShowWindow(SW\_SHOW);

}

void CMy51\_s1View::OnDoModal()

{//产生有模式对话框

CTestDlg dlg;

int iRet=dlg.DoModal();

TRACE("dlg return %d\n",iRet);

}

下载例子。如果你在调试这个程序时你会发现程序在退出后会有内存泄漏，这是因为我没有释放无模式对话框所使用的内存，这一问题会在以后的章节5.3 创建无模式对话框中专门讲述。

关于在使用对话框时Enter键和Escape键的处理：在使用对话框是你会发现当你按下Enter键或Escape键都会退出对话框，这是因为Enter键会引起CDialog::OnOK()的调用，而Escape键会引起CDialog::OnCancel()的调用。而这两个调用都会引起对话框的退出。在MFC中这两个成员函数都是虚拟函数，所以我们需要进行重载，如果我们不希望退出对话框那么我们可以在函数中什么都不做，如果需要进行检查则可以添加检查代码，然后调用父类的OnOK()或OnCancel()。相关代码如下:

void CTestDlg::OnOK()

{

AfxMessageBox("你选择确定");

CDialog::OnOK();

}

void CTestDlg::OnCancel()

{

AfxMessageBox("你选择取消");

CDialog::OnCancel();

}

### 5.2 创建有模式对话框

### 5.3 创建无模式对话框

### 5.4 在对话框中进行消息映射

### 5.5 在对话框中进行数据交换和数据检查

### 5.6 使用属性对话框

### 5.7 使用通用对话框

### 5.8 建立以对话框为基础的应用

### 5.9 使用对话框作为子窗口

## 第六章 网络通信开发

### 6.1 WinSock介绍

### 6.2 利用WinSock进行无连接的通信

### 6.3 利用WinSock进行有连接的通信