# VS2010/MFC编程入门教程之目录

**第一部分：VS2010/MFC开发环境**

[VS2010/MFC编程入门之前言](#_VS2010/MFC编程入门之前言)

[VS2010/MFC编程入门之一（VS2010与MSDN安装过程图解）](#_VS2010/MFC编程入门之一（VS2010与MSDN安装过程图解）)

**第二部分：VS2010/MFC应用程序框架**

[VS2010/MFC编程入门之二（利用MFC向导生成单文档应用程序框架）](#_VS2010/MFC编程入门之二（利用MFC向导生成单文档应用程序框架)

[VS2010/MFC编程入门之三（VS2010应用程序工程中文件的组成结构）](#_VS2010/MFC编程入门之三（VS2010应用程序工程中文件的组成)

[VS2010/MFC编程入门之四（MFC应用程序框架分析）](#_VS2010/MFC编程入门之四（MFC应用程序框架分析）)

[VS2010/MFC编程入门之五（MFC消息映射机制概述）](#_VS2010/MFC编程入门之五（MFC消息映射机制概述）)

**第三部分：对话框**

[VS2010/MFC编程入门之六（对话框：创建对话框模板和修改对话框属性）](#_VS2010/MFC编程入门之六（对话框：创建对话框模板和修改对话框属)

[VS2010/MFC编程入门之七（对话框：为对话框添加控件）](#_VS2010/MFC编程入门之七（对话框：为对话框添加控件）)

[VS2010/MFC编程入门之八（对话框：创建对话框类和添加控件变量）](#_VS2010/MFC编程入门之八（对话框：创建对话框类和添加控件变量）)

[VS2010/MFC编程入门之九（对话框：为控件添加消息处理函数）](#_VS2010/MFC编程入门之九（对话框：为控件添加消息处理函数）)

[VS2010/MFC编程入门之十（对话框：设置对话框控件的Tab顺序）](#_VS2010/MFC编程入门之十（对话框：设置对话框控件的Tab顺序）)

[VS2010/MFC编程入门之十一（对话框：模态对话框及其弹出过程）](#_VS2010/MFC编程入门之十一（对话框：模态对话框及其弹出过程）)

[VS2010/MFC编程入门之十二（对话框：非模态对话框的创建及显示）](#_VS2010/MFC编程入门之十二（对话框：非模态对话框的创建及显示）)

[VS2010/MFC编程入门之十三（对话框：属性页对话框及相关类的介绍）](#_VS2010/MFC编程入门之十三（对话框：属性页对话框及相关类的介绍)

[VS2010/MFC编程入门之十四（对话框：向导对话框的创建及显示）](#_VS2010/MFC编程入门之十四（对话框：向导对话框的创建及显示）)

[VS2010/MFC编程入门之十五（对话框：一般属性页对话框的创建及显示）](#_VS2010/MFC编程入门之十五（对话框：一般属性页对话框的创建及显)

[VS2010/MFC编程入门之十六（对话框：消息对话框）](#_VS2010/MFC编程入门之十六（对话框：消息对话框）)

[VS2010/MFC编程入门之十七（对话框：文件对话框）](#_VS2010/MFC编程入门之十七（对话框：文件对话框）)

[VS2010/MFC编程入门之十八（对话框：字体对话框）](#_VS2010/MFC编程入门之十八（对话框：字体对话框）)

[VS2010/MFC编程入门之十九（对话框：颜色对话框）](#_VS2010/MFC编程入门之十九（对话框：颜色对话框）)

**第四部分：常用控件**

[VS2010/MFC编程入门之二十（常用控件：静态文本框）](#_VS2010/MFC编程入门之二十（常用控件：静态文本框）)

[VS2010/MFC编程入门之二十一（常用控件：编辑框Edit Control）](#_VS2010/MFC编程入门之二十一（常用控件：编辑框Edit_Con)

[VS2010/MFC编程入门之二十二（常用控件：按钮控件Button、Radio Button和Check Box）](#_VS2010/MFC编程入门之二十二（常用控件：按钮控件Button、)

[VS2010/MFC编程入门之二十三（常用控件：按钮控件的编程实例）](#_VS2010/MFC编程入门之二十三（常用控件：按钮控件的编程实例）)

[VS2010/MFC编程入门之二十四（常用控件：列表框控件ListBox）](#_VS2010/MFC编程入门之二十四（常用控件：列表框控件ListBo)

[VS2010/MFC编程入门之二十五（常用控件：组合框控件Combo Box）](#_VS2010/MFC编程入门之二十五（常用控件：组合框控件Combo_)

[VS2010/MFC编程入门之二十六（常用控件：滚动条控件Scroll Bar）](#_VS2010/MFC编程入门之二十六（常用控件：滚动条控件Scroll)

[VS2010/MFC编程入门之二十七（常用控件：图片控件Picture Control）](#_VS2010/MFC编程入门之二十七（常用控件：图片控件Picture)

[VS2010/MFC编程入门之二十八（常用控件：列表视图控件List Control 上）](#_VS2010/MFC编程入门之二十八（常用控件：列表视图控件List_)

[VS2010/MFC编程入门之二十九（常用控件：列表视图控件List Control 下）](#_VS2010/MFC编程入门之二十九（常用控件：列表视图控件List_)

[VS2010/MFC编程入门之三十（常用控件：树形控件Tree Control 上）](#_VS2010/MFC编程入门之三十（常用控件：树形控件Tree_Con)

[VS2010/MFC编程入门之三十一（常用控件：树形控件Tree Control 下）](#_VS2010/MFC编程入门之三十一（常用控件：树形控件Tree_Co)

[VS2010/MFC编程入门之三十二（常用控件：标签控件Tab Control 上）](#_VS2010/MFC编程入门之三十二（常用控件：标签控件Tab_Con)

[VS2010/MFC编程入门之三十三（常用控件：标签控件Tab Control 下）](#_VS2010/MFC编程入门之三十三（常用控件：标签控件Tab_Con)

**第五部分：菜单、工具栏与状态栏**

[VS2010/MFC编程入门之三十四（菜单：VS2010菜单资源详解）](#_VS2010/MFC编程入门之三十四（菜单：VS2010菜单资源详解）)

[VS2010/MFC编程入门之三十五（菜单：菜单及CMenu类的使用）](#_VS2010/MFC编程入门之三十五（菜单：菜单及CMenu类的使用）)

[VS2010/MFC编程入门之三十六（工具栏：工具栏资源及CToolBar类）](#_VS2010/MFC编程入门之三十六（工具栏：工具栏资源及CToolB)

[VS2010/MFC编程入门之三十七（工具栏：工具栏的创建、停靠与使用）](#_VS2010/MFC编程入门之三十七（工具栏：工具栏的创建、停靠与使用)

[VS2010/MFC编程入门之三十八（状态栏的使用详解）](#_VS2010/MFC编程入门之三十八（状态栏的使用详解）)

**第六部分：文档、视图和框架**

[VS2010/MFC编程入门之三十九（文档、视图和框架：概述）](#_VS2010/MFC编程入门之三十九（文档、视图和框架：概述）)

[VS2010/MFC编程入门之四十（文档、视图和框架：各对象之间的关系）](#_VS2010/MFC编程入门之四十（文档、视图和框架：各对象之间的关系)

[VS2010/MFC编程入门之四十一（文档、视图和框架：分割窗口）](#_VS2010/MFC编程入门之四十一（文档、视图和框架：分割窗口）)

**第七部分：MFC常用类**

[VS2010/MFC编程入门之四十二（MFC常用类：CString类）](#_VS2010/MFC编程入门之四十二（MFC常用类：CString类）)

[VS2010/MFC编程入门之四十三（MFC常用类：CTime类和CTimeSpan类）](#_VS2010/MFC编程入门之四十三（MFC常用类：CTime类和CT)

[VS2010/MFC编程入门之四十四（MFC常用类：定时器Timer）](#_VS2010/MFC编程入门之四十四（MFC常用类：定时器Timer）)

[VS2010/MFC编程入门之四十五（MFC常用类：CFile文件操作类）](#_VS2010/MFC编程入门之四十五（MFC常用类：CFile文件操作)

[VS2010/MFC编程入门之四十六（MFC常用类：MFC异常处理）](#_VS2010/MFC编程入门之四十六（MFC常用类：MFC异常处理）)

**第八部分：字体和文本输出**

[VS2010/MFC编程入门之四十七（字体和文本输出：CFont字体类）](#_VS2010/MFC编程入门之四十七（字体和文本输出：CFont字体类)

[VS2010/MFC编程入门之四十八（字体和文本输出：文本输出）](#_VS2010/MFC编程入门之四十八（字体和文本输出：文本输出）)

**第九部分：图形图像**

[VS2010/MFC编程入门之四十九（图形图像：CDC类及其屏幕绘图函数）](#_VS2010/MFC编程入门之四十九（图形图像：CDC类及其屏幕绘图函)

[VS2010/MFC编程入门之五十（图形图像：GDI对象之画笔CPen）](#_VS2010/MFC编程入门之五十（图形图像：GDI对象之画笔CPen)

[VS2010/MFC编程入门之五十一（图形图像：GDI对象之画刷CBrush）](#_VS2010/MFC编程入门之五十一（图形图像：GDI对象之画刷CBr)

**第十部分：Ribbon界面开发**

[VS2010/MFC编程入门之五十二（Ribbon界面开发：创建Ribbon样式的应用程序框架）](#_VS2010/MFC编程入门之五十二（Ribbon界面开发：创建Rib)

[VS2010/MFC编程入门之五十三（Ribbon界面开发：为Ribbon Bar添加控件）](#_VS2010/MFC编程入门之五十三（Ribbon界面开发：为Ribb)

[VS2010/MFC编程入门之五十四（Ribbon界面开发：使用更多控件并为控件添加消息处理函数）](#_VS2010/MFC编程入门之五十四（Ribbon界面开发：使用更多控)

# VS2010/MFC编程入门之前言

       鸡啄米的C++[编程入门](http://www.jizhuomi.com/catalog.asp?tags=%E7%BC%96%E7%A8%8B%E5%85%A5%E9%97%A8)系列给大家讲了C++的编程入门知识，大家对C++语言在语法和设计思想上应该有了一定的了解了。但是教程中讲的例子只是一个个简单的例程，并没有可视化窗口。鸡啄米在这套VS2010/MFC编程入门教程中将会给大家讲解怎样使用VS2010进行可视化编程，也就是基于窗口的程序。

       C++编程入门系列主要偏重于理论方面的知识，目的是让大家打好底子，练好内功，在使用VC++编程时不至于丈二和尚摸不着头脑。本套教程也会涉及到VC++的原理性的东西，同样更重视实用性，让大家学完本套教程以后，基本的界面程序都能很容易编写出来。

**VC++简介**

       VC++全称是Visual C++，是由微软提供的C++开发工具，它与C++的根本区别就在于，C++是语言，而VC++是用C++语言编写程序的工具平台。VC++不仅是一个编译器更是一个集成开发环境，包括编辑器、调试器和编译器等，一般它包含在Visual Studio中。Visual Studio包含了VB、VC++、C#等编译环境。当然我们在使用VC++ 6.0的时候为了轻便，总是只单独安装VC++ 6.0。但自微软2002年发布Visual Studio.NET以来，微软建立了在.NET框架上的代码托管机制，一个项目可以支持多种语言开发的组件，VC++同样被扩展为支持代码托管机制的开发环境，所以.NET Framework是必须的，也就不再有VC++的独立安装程序，不过可以在安装Visual Studio时只选择VC++进行安装。

**VC++版本的选择:VS2010**

       因为VC++ 6.0以后的版本不再有独立的安装程序，所以鸡啄米在教程中将不会称VC++ 6.0以后的版本为VC++ 7.0等等，而是用VC++所属的Visual Studio的版本名称代替，比如VS2003。

       近些年VC++主要的版本包括：VC++ 6.0、VS2003、VS2005、VS2008和[VS2010](http://www.jizhuomi.com/software/27.html)。

       VC++ 6.0占用的系统资源比较少，打开工程、编译运行都比较快，所以赢得很多软件开发者的青睐。但因为它先于C++标准推出，所以对C++标准的支持不太好。举个例子：

       for(int i=0; i<5; i++)  
       {  
                a[i] = i;  
       }

       for语句中声明的变量i，对于VC++ 6.0来说，出了for循环仍能使用。但很显然这与C++标准对于变量[生存期](http://www.jizhuomi.com/software/60.html)的规定不符合。

       随着VC++版本的更新，对C++标准的支持越来越好，对各种技术的支持也越来越完善。但同时新版本所需的资源也越来越多，对处理器和内存的要求越来越高。到VS2010，光安装文件就2G多，安装后的文件占3G多空间，其运行也经常受处理器和内存等性能的限制。但鸡啄米还是推荐大家使用VS2010，毕竟它是最新版本，类库和开发技术都是最完善的，本教程也将使用VS2010为大家做例程的演示。当然如果系统配置确实比较低，可以选择VS2005，VS2005和VS2010相比还是要轻量级一些的。VC++ 6.0已经过时，奉劝大家尽量别用了。

**VC++与MFC**

       讲VC++免不了要提MFC，MFC全称Microsoft Foundation Classes，也就是微软基础类库。它是VC++的核心，是C++与Windows API的结合，很彻底的用C++封装了Windows SDK（Software Development Kit，软件开发工具包）中的结构和功能，还提供了一个应用程序框架，此应用程序框架为软件开发者完成了一些例行化的工作，比如各种窗口、工具栏、菜单的生成和管理等，不需要开发者再去解决那些很复杂很乏味的难题，比如每个窗口都要使用Windows API注册、生成与管理。这样就大大减少了软件开发者的工作量，提高了开发效率。

       当然VC++不是只能够创建MFC应用程序，同样也能够进行Windows SDK编程，但是那样的话就舍弃了VC++的核心，放弃了VC++最强大的部分。MFC也不是只能用于VC++中，它同样也可以用在Borland C++等编译器中，当然没有几个人这样做。

       本节旨在让大家对VC++、VS2010和MFC有基本的概念上的认识，后面鸡啄米会带大家进入VS2010/MFC的世界，让大家轻松的开发各种包含窗口、图形等的可视化程序。

# VS2010/MFC编程入门之一（VS2010与MSDN安装过程图解）

       上一讲中鸡啄米对VC++和MFC做了一些简单介绍。在本套教程中鸡啄米将使用VS2010为大家讲解如何使用VC++和MFC进行编程，所以首先要安装VS2010。

**一.下载VS2010**

       首先我们需要下载VS2010，大家可以在网上下载VS2010破解正式版，建议选择英文版，养成使用英文工具的习惯。鸡啄米使用VS2010旗舰试用版VS2010UltimTrial.iso为例介绍安装过程，旗舰试用版官方下载地址为：<http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=12187>。正式版的安装过程与试用版类似。

**二.安装VS2010**

       下载后进行安装。安装方法与一般的iso文件一样，可以使用虚拟光驱软件Daemon Tools安装，也可以将其解压后点击setup.exe进行安装。

       鸡啄米为了让大家更直观的看到安装过程，我将在自己机子上再重新安装一次，并截图为大家讲解。

       这里使用Daemon Tools安装VS2010。首先打开Daemon Tools，屏幕右下角会出现托盘图标，在图标上点右键，会弹出菜单，再把鼠标移到菜单项“虚拟设备”上，然后再移到子菜单项“设备 0：[L:] 无媒体”上，最后点击下一级子菜单项“装载映像”，弹出对话框选择VS2010UltimTrial.iso文件。

       这样虚拟光驱就会打开此iso文件，弹出自动安装的提示，选择“运行autorun.exe”就可以了，如果没有弹出提示就通过资源管理器进入虚拟光驱，用setup.exe安装。接着会弹出下面的对话框：

       当然选择“Install Microsoft Visual Studio 2010”进入下一步，加载安装组件后如下显示：

    点“Next”后：

         选择“I have read and accept the license terms”后点“Next”弹出对话框：

       此处是让我们选择要安装的功能，有两种：Full（完全）和Custom（自定义）。Full选项表示安装所有编程语言和工具，Custom选择表示可以自定义要安装的编程语言和工具。右侧可以更改安装路径，鸡啄米建议不要安装到C盘，因为它占用的空间比较大。鸡啄米安装到了D盘，使用Full完全安装。如果选择Custom安装，点“Next”则出现如下画面：

       大家可以根据自己的需要取消某些语言或工具的安装，比如不想安装Visual C#，取消选择它就可以了。如果觉得以后都有可能会用到，那就像鸡啄米一样选择完全安装吧。

       Full或Custom方式和安装路径设置好后，点“Install”进行安装：

         可能正式版的安装文件在安装过程中会有重启过程。鸡啄米使用的试用版中间并没有重启。安装完成：

         如果要继续安装MSDN，先不要卸载虚拟光驱映像。

**三.安装MSDN**

         我们使用VS2010进行软件开发同样离不开帮助文档，即MSDN。在本地安装MSDN的方法如下：

         在开始菜单的“所有程序”->“Microsoft Visual Studio 2010”->“Visual Studio Tools”下选择“Manage Help Settings - ENU”：

         弹出对话框：

         可以将帮助库存在默认路径，也可以修改存放路径。鸡啄米使用默认路径，点“OK”出现：

        选择“Install Content From Disk”后弹出对话框选择帮助所在文件，这时需要在加载了VS2010的虚拟光驱中找，选择图中所示路径：

        点OK后出现如下对话框，可以点“Add”选择要添加的帮助库，鸡啄米全部添加了。

         点“Update”进行安装，等待其完成就可以了。

         使用MSDN时点击开始菜单的“所有程序”->“Microsoft Visual Studio 2010”->“Microsoft Visual Studio 2010 Documentation”即可。

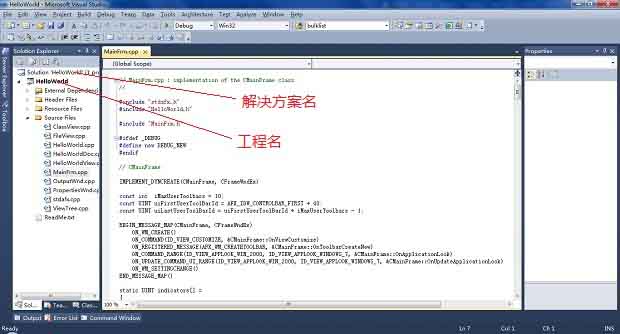
         到此VS2010和MSDN的安装过程就结束了。以后就可以正式使用VS2010进行软件开发了。至于[VS2010的使用方法](http://www.jizhuomi.com/software/27.html)在鸡啄米的C++编程入门系列中已经介绍过，大家可以看看。

# VS2010/MFC编程入门之二（利用MFC向导生成单文档应用程序框架）

**解决方案与工程**

鸡啄米在[VS2010的使用介绍](http://www.jizhuomi.com/software/27.html)中已经讲了解决方案与工程的概念，这里再重提一下。每个应用程序都作为一个工程来处理，它包含了头文件、源文件和资源文件等，这些文件通过工程集中管理。在VS2010中，工程都是在解决方案管理之下的。一个解决方案可以管理多个工程，可以把解决方案理解为多个有关系或者没有关系的工程的集合。VS2010提供了一个Solution Explorer解决方案浏览器视图，可以显示当前解决方案的内容，当新建一个工程时可以选择新建一个解决方案还是加入当前解决方案。

       下图左侧面板中正在显示的视图就是Solution Explorer，视图中有一个解决方案-HelloWorld，此解决方案下有一个同名的工程-HelloWorld。



      在应用程序向导生成应用程序后，VS2010会在用户设置的路径下，以解决方案名为名称建立一个目录，里面存放自动生成的文件。

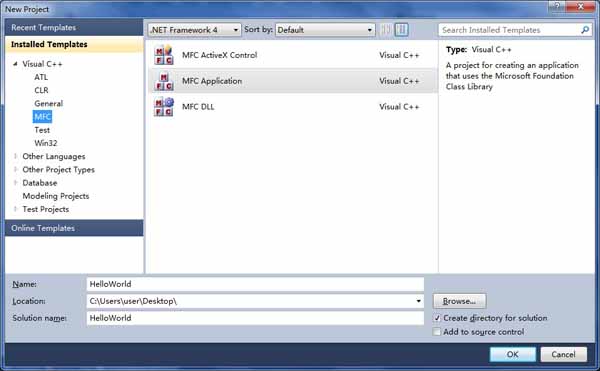
**使用VS2010应用程序向导生成单文档应用程序框架**

鸡啄米这里简略演示下怎样生成单文档应用程序框架，让大家先有个直观的了解，有不理解的地方可以留着以后回来再看。下面按照操作步骤一步步讲解：

      1.点菜单栏File->New->Project，弹出New Project对话框，我们可以选择工程类型。

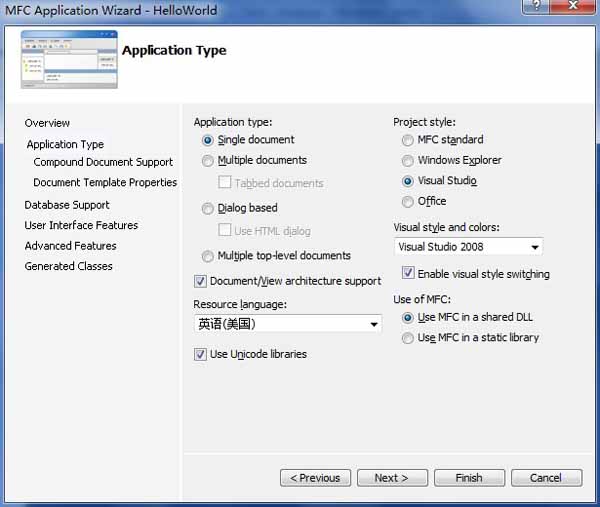
      如果安装完VS2010以后第一启动时已经设置为VC++，则Installed Templates->Visual C++项会默认展开，而如果没有设置VC++，则可以展开到Installed Templates->Other Languages->Visual C++项。因为我们要生成的是MFC程序，所以在“Visual C++”下选择“MFC”，对话框中间区域会出现三个选项：MFC ActiveX Control、MFC Application和MFC DLL。MFC ActiveX Control用来生成MFC ActiveX控件程序。MFC Application用来生成MFC应用程序。MFC DLL用来生成MFC动态链接库程序。当然我们要选择MFC Application。

      在对话框下部有Name、Location和Solution name三个设置项。意义如下：Name--工程名，Location--解决方案路径，Solution name--解决方案名称。这里Name我们设为“HelloWorld”，Location设置为“桌面”的路径，Solution name默认和Name一样，当然可以修改为其他名字，这里我们不作修改，也使用“HelloWorld”。点“OK”按钮。



      2.这时会弹出“MFC Application Wizard”对话框，上部写有“Welcome to the MFC Application Wizard”，下面显示了当前工程的默认设置。第一条“Tabbed multiple document interface (MDI)”是说此工程是多文档应用程序。如果这时直接点下面的“Finish”按钮，可生成具有上面列出设置的多文档程序。但我们此例是要建立单文档应用程序，所以点“Next”按钮再继续设置吧。

      3.接下来弹出的对话框上部写有“Application Type”，当然是让选择应用程序类型，我们看到有四种类型：Single document（单文档）、Multiple documents（多文档）、Dialog based（基于对话框）和Multiple top-level documents。我们选择Single document类型，以生成一个单文档应用程序框架。单文档应用程序运行时是一个单窗口界面。



      此对话框的“Resource language”还提供语言的选择，这里默认选择英语。“Project style”可选择工程风格，我们选择默认的“Visual Studio”风格。“Use of MFC”有两个选项：Use MFC in a shared DLL（动态链接库方式使用MFC）和Use MFC in a static library（静态库方式使用MFC）。选择Use MFC in a shared DLL时MFC的类会以动态链接库的方式访问，所以我们的应用程序本身就会小些，但是发布应用程序时必须同时添加必要的动态链接库，以便在没有安装VS2010的机子上能够正常运行程序。选择Use MFC in a static library时MFC的类会编译到可执行文件中，所以应用程序的可执行文件要比上种方式大，但可以单独发布，不需另加包含MFC类的库。这里我们使用默认的Use MFC in a shared DLL。点“Next”按钮。

      4.此时弹出上部写有“Compound Document Support”的对话框，可以通过它向应用程序加入OLE支持，指定OLE选项的复合文档类型。本例不需要OLE特性，使用默认值“None”。点“Next”按钮。

      5.弹出的新对话框上部写有“Document Template Properties”。“File extension”可以设置程序能处理的文件的扩展名。对话框其他选项还可以更改程序窗口的标题。我们都使用默认设置，点“Next”按钮。

      6.此时弹出的对话框主题是“Database Support”。用于设置数据库选项。此向导可以生成数据库应用程序需要的代码。它有四个选项：

      None：忽略所有的数据库支持；

      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：创建对应指定表的一个数据库类和一个视图类，并附加标准文件支持。

      本例选择默认值“None”，不使用数据库特性。点“Next”按钮。

      7.这时弹出的对话框是关于“User Interface Features”，即用户界面特性。我们可以设置有无最大化按钮、最小化按钮、系统菜单和初始状态栏等。还可以选择使用菜单栏和工具栏生成简单的应用程序还是使用ribbon。这里我们都选择默认设置。点“Next”进入下一步。

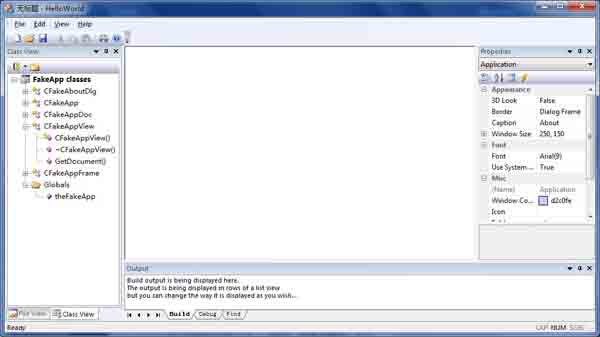
      8.此时弹出“高级特性”对话框。可以设置的高级特性包括有无打印和打印预览等。在“Number of files on recent file list”项可以设置在程序界面的文件菜单下面最近打开文件的个数。我们仍使用默认值。点“Next”按钮。

      9.弹出“生成类”对话框。在对话框上部的“生成类”列表框内，列出了将要生成的4 个类：一个视图类（CHelloWorldView）、一个应用类（CHelloWorldApp）、一个文档类（CHelloWorldDoc）和一个主框架窗口类（CMainFrame）。在对话框下面的几个编辑框中，可以修改默认的类名、类的头文件名和源文件名。对于视图类，还可以修改其基类名称，默认的基类是CView，还有其他几个基类可以选择。这里我们还是使用默认设置。点“Finish”按钮。

      应用程序向导最后为我们生成了应用程序框架，并在Solution Explorer中自动打开了解决方案（见上面第一张图）。

**编译运行生成的程序**

点菜单中的Build->Build HelloWorld编译程序，然后点Debug->Start Without Debugging（快捷键Ctrl+F5）运行程序，也可以直接点Debug->Start Without Debugging，这时会弹出对话框提示是否编译，选择“Yes”，VS2010将自动编译链接运行HelloWorld程序。结果页面如下所示：



       终于看见界面了。鸡啄米在以后的教程中会继续讲解各种界面和控件的使用方法。欢迎到鸡啄米博客交流，您的关注是我前进的动力。

# VS2010/MFC编程入门之三（VS2010应用程序工程中文件的组成结构）

       鸡啄米在上一讲中为大家演示了如何[利用应用程序向导创建单文档应用程序框架](http://www.jizhuomi.com/software/141.html)。这一节将以上一讲中生成应用程序HelloWorld的文件结构为例，讲解[VS2010](http://www.jizhuomi.com/catalog.asp?tags=VS2010)应用程序工程中文件的组成结构。

       用应用程序向导生成框架程序后，我们可以在之前设置的Location下看到以解决方案名命名的文件夹，此文件夹中包含了几个文件和一个以工程名命名的子文件夹，这个子文件夹中又包含了若干个文件和一个res文件夹，创建工程时的选项不同，工程文件夹下的文件可能也会有所不同。

       如果已经以Debug方式编译链接过程序，则会在解决方案文件夹下和工程子文件夹下各有一个名为“Debug”的文件夹，而如果是Release方式编译则会有名为“Release”的文件夹。这两种编译方式将产生两种不同版本的可执行程序：**Debug版本和Release版本**。Debug版本的可执行文件中包含了用于调试的信息和代码，而Release版本则没有调试信息，不能进行调试，但可执行文件比较小。

**鸡啄米将所有文件分为6个部分：解决方案相关文件、工程相关文件、应用程序头文件和源文件、资源文件、预编译头文件和编译链接生成文件。**

**1.解决方案相关文件**

       解决方案相关文件包括解决方案文件夹下的.sdf文件、.sln文件、.suo文件和ipch文件夹。

       .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。

**2.工程相关文件**

       工程相关文件包括工程文件夹下的.vcxproj文件和.vcxproj.filters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.应用程序头文件和源文件**

       应用程序向导会根据应用程序的类型（单文档、多文档或基于对话框的程序）自动生成一些头文件和源文件，这些文件是工程的主体部分，用于实现主框架、文档、视图等。鸡啄米下面分别简单介绍下各个文件：

       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。

**4.资源文件**

       一般我们使用MFC生成窗口程序都会有对话框、图标、菜单等资源，应用程序向导会生成资源相关文件：res目录、HelloWorld.rc文件和Resource.h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。

**5.预编译头文件**

       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stdafx.h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。

**6.编译链接生成文件**

       如果是Debug方式编译，则会在解决方案文件夹和工程文件夹下都生成Debug子文件夹，而如果是Release方式编译则生成Release子文件夹。

       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。

       关于应用程序工程文件的组成结构鸡啄米就先讲到这了。其中包含了很多专有名词，以后大家会慢慢熟悉的。欢迎来鸡啄米博客交流。谢谢。

# VS2010/MFC编程入门之四（MFC应用程序框架分析）

       上一讲鸡啄米讲的是[VS2010应用程序工程中文件的组成结构](http://www.jizhuomi.com/software/143.html)，可能大家对工程的运行原理还是很模糊，理不出头绪，毕竟跟C++编程入门系列中的例程差别太大。这一节鸡啄米就为大家分析下MFC应用程序框架的运行流程。

**一.SDK应用程序与MFC应用程序运行过程的对比**

       程序运行都要有入口函数，在之前的C++教程中都是main函数，而Windows应用程序的入口函数是WinMain函数，MFC程序也是从WinMain函数开始的。下面鸡啄米就给出用Windows SDK写的“HelloWorld”程序，与应用程序框架进行对比，这样能更好的了解框架是怎样运行的。Windows SDK开发程序就是不使用MFC类库，直接用Windows API函数进行[软件开发](http://www.jizhuomi.com)。鸡啄米不是要讲解SDK开发，只是为了对比而简单介绍，至于SDK开发可以在大家学完MFC以后选择是否要研究，一般来说有简单了解就可以了。

**SDK应用程序**

       首先，给出Windows SDK应用程序“HelloWorld”的源码：

**C++代码**

1. #include <windows.h>
3. LRESULT CALLBACK myWndProc(HWND hWindow, UINT msg, WPARAM wParam, LPARAM lParam);
5. int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine, int iCmdShow)
6. {
7. const static TCHAR appName[] = TEXT("Hello world");
8. WNDCLASSEX myWin;
9. myWin.cbSize = sizeof(myWin);
10. myWin.style = CS\_HREDRAW | CS\_VREDRAW;
11. myWin.lpfnWndProc = myWndProc;
12. myWin.cbClsExtra = 0;
13. myWin.cbWndExtra = 0;
14. myWin.hInstance = hInstance;
15. myWin.hIcon = 0;
16. myWin.hIconSm  = 0;
17. myWin.hCursor = 0;
18. myWin.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);
19. myWin.lpszMenuName = 0;
20. myWin.lpszClassName = appName;
21. //Register
22. if (!RegisterClassEx(&myWin)) return 0;
23. const HWND hWindow = CreateWindow(
24. appName,
25. appName,
26. WS\_OVERLAPPEDWINDOW,
27. CW\_USEDEFAULT,
28. CW\_USEDEFAULT,
29. CW\_USEDEFAULT,
30. CW\_USEDEFAULT,
31. 0,
32. 0,
33. hInstance,
34. 0);
35. ShowWindow(hWindow,iCmdShow);
36. UpdateWindow(hWindow);
37. {
38. MSG msg;
39. while(GetMessage(&msg,0,0,0))
40. {
41. TranslateMessage(&msg);
42. DispatchMessage(&msg);
43. }
44. return (int)msg.wParam;
45. }
46. }
48. LRESULT CALLBACK myWndProc(HWND hWindow, UINT msg, WPARAM wParam, LPARAM lParam)
49. {
50. if (msg==WM\_PAINT)
51. {
52. PAINTSTRUCT ps;
53. const HDC hDC = BeginPaint(hWindow,&ps);
54. RECT rect;
55. GetClientRect(hWindow,&rect);
56. DrawText(hDC,TEXT("HELLO WORLD"),-1,&rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);
57. EndPaint(hWindow,&ps);
58. return 0;
59. }
60. else if (msg==WM\_DESTROY)
61. {
62. PostQuitMessage(0);
63. return 0;
64. }
65. return DefWindowProc(hWindow,msg,wParam,lParam);
66. }

       上面的程序运行的流程是：进入WinMain函数->初始化WNDCLASSEX，调用RegisterClassEx函数注册窗口类->调用ShowWindow和UpdateWindow函数显示并更新窗口->进入消息循环。关于消息循环再简单说下，Windows应用程序是消息驱动的，系统或用户让应用程序进行某项操作或完成某个任务时会发送消息，进入程序的消息队列，然后消息循环会将消息队列中的消息取出，交予相应的窗口过程处理，此程序的窗口过程函数就是myWndProc函数，窗口过程函数处理完消息就完成了某项操作或任务。本例是要显示“HELLO WORLD”字符串，UpdateWindow函数会发送WM\_PAINT消息，但是此消息不经过消息队列而是直接送到窗口过程处理，在窗口过程函数中最终绘制了“HELLO WORLD”字符串。

**MFC应用程序**

       下面是MFC应用程序的运行流程，通过MFC库中代码进行分析：

       首先在HelloWorld.cpp中定义[全局对象](http://www.jizhuomi.com/software/60.html)theApp：CHelloWorldApp theApp;。调用CWinApp和CHelloWorldApp的构造函数后，进入WinMain函数（位于appmodul.cpp中）。

**C++代码**

1. extern "C" int WINAPI
2. \_tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
3. \_In\_ LPTSTR lpCmdLine, int nCmdShow)
4. #pragma warning(suppress: 4985)
5. {
6. // call shared/exported WinMain
7. return AfxWinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow);
8. }

       在TCHAR.h中，有此定义：#define \_tWinMain   WinMain，所以这里的\_tWinMain就是WinMain函数。它调用了AfxWinMain函数（位于WinMain.cpp中）。

**C++代码**

1. int AFXAPI AfxWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,LPTSTR lpCmdLine, int nCmdShow)
2. {
3. .............略
4. // App global initializations (rare)
5. if (pApp != NULL && !pApp->InitApplication())
6. goto InitFailure;
8. if (!pThread->InitInstance())
9. {
10. .........略
11. }
13. // Run函数位于THRDCORE.cpp中，由此函数进入消息循环
14. nReturnCode = pThread->Run();
16. ..............略
18. return nReturnCode;
19. }

       上面InitInstance函数的代码如下：

**C++代码**

1. BOOL CTestApp::InitInstance()
2. {
3. .............略
4. CSingleDocTemplate\* pDocTemplate;
5. pDocTemplate = new CSingleDocTemplate(
6. IDR\_MAINFRAME,
7. RUNTIME\_CLASS(CTestDoc),
8. RUNTIME\_CLASS(CMainFrame),      // main SDI frame window
9. RUNTIME\_CLASS(CTestView));
10. if (!pDocTemplate)
11. return FALSE;
12. AddDocTemplate(pDocTemplate);
13. // Parse command line for standard shell commands, DDE, file open
15. CCommandLineInfo cmdInfo;
16. ParseCommandLine(cmdInfo);
18. //ProcessShellCommand位于AppUI2.cpp中，注册并创建窗口
19. if (!ProcessShellCommand(cmdInfo))
20. return FALSE;
22. m\_pMainWnd->ShowWindow(SW\_SHOW);
23. m\_pMainWnd->UpdateWindow();
25. return TRUE;
26. }

       InitInstance中的ProcessShellCommand函数又调用了CMainFrame的LoadFrame函数注册并创建了窗口，执行完ProcessShellCommand函数以后，调用了m\_pMainWnd的ShowWindow和UpdateWindow函数显示并更新框架窗口。这些是不是与上面的SDK程序十分类似？

       接下来该是消息循环了，上面的AfxWinMain函数中调用了pThread的Run函数（位于THRDCORE.cpp中），在Run中包含了消息循环。Run函数的代码如下：

**C++代码**

1. int CWinThread::Run()
2. {
3. .............略
4. // phase2: pump messages while available
5. do
6. {
7. // pump message, but quit on WM\_QUIT
8. if (!PumpMessage())
9. return ExitInstance();
11. // reset "no idle" state after pumping "normal" message
12. if (IsIdleMessage(&m\_msgCur))
13. {
14. bIdle = TRUE;
16. lIdleCount = 0;
18. }
19. } while (::PeekMessage(&m\_msgCur, NULL, NULL, NULL, PM\_NOREMOVE));
20. ..............略
21. }
23. BOOL CWinThread::PumpMessage()
24. {
25. return AfxInternalPumpMessage();
26. }
28. BOOL AFXAPI AfxInternalPumpMessage()
29. {
30. \_AFX\_THREAD\_STATE \*pState = AfxGetThreadState();
32. if (!::GetMessage(&(pState->m\_msgCur), NULL, NULL, NULL))
33. {
34. .............略
35. }
36. ...............略
37. if (pState->m\_msgCur.message != WM\_KICKIDLE && !AfxPreTranslateMessage(&(pState->m\_msgCur)))
38. {
39. ::TranslateMessage(&(pState->m\_msgCur));
40. ::DispatchMessage(&(pState->m\_msgCur));
41. }
43. return TRUE;
44. }

       我们看到PumpMessage中通过调用GetMessage、TranslateMessage、DispatchMessage等建立了消息循环并投递消息。

       窗口过程函数AfxWinProc形式如下：

**C++代码**

1. LRESULT CALLBACK AfxWndProc(HWND hWnd,UINT nMsg,WPARAM wParam, LPARAM lParam)
2. {
3. ……
4. CWnd\*pWnd=CWnd::FromHandlePermanent(hWnd);
5. ReturnAfxCallWndProc(pWnd,hWnd,nMsg,wParam,lParam);
6. }

**两者运行过程对比**

       到此，通过对比可以发现，MFC应用程序的运行流程与SDK程序是类似的，都是先进行一些初始化过程，再注册并创建窗口，然后显示、更新窗口，最后进入消息循环，消息都由窗口过程函数处理。现在大家是不是觉得有些头绪了？在运行流程上有基本的掌握即可。

**二.MFC应用程序框架主要类之间的关系**

       在第二讲中，给大家演示了如何[利用应用程序向导生成单文档应用程序框架](http://www.jizhuomi.com/software/141.html)，可以看到程序的基本框架和必要的代码都自动生成了，上一讲又讲解了文件组成结构，实际上在前面自动生成的框架中比较重要的类包括以下几个：CHelloWorldApp、CMainFrame、CHelloWorldDoc和CHelloWorldView，至于其他的类比如CClassView、CFileView等都是在框架窗口（CMainFrame）上创建的面板等，不是必要的。

       鸡啄米就四个主要类的关系简单讲下，CHelloWorldApp类处理消息，将收到的消息分发给相应的对象。CMainFrame是视图CHelloWorldView的父窗口，视图CHelloWorldView就显示在CMainFrame的客户区中。视图类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。关于视图类和文档类的关系后面会详细讲解。

       本节VC++/MFC[编程入门](http://www.jizhuomi.com/catalog.asp?tags=%E7%BC%96%E7%A8%8B%E5%85%A5%E9%97%A8)教程内容比较多，主要是让大家对MFC应用程序的运行原理有大概的了解。对于以后的MFC开发有很多好处。如果有问题请在鸡啄米博客留言交流。谢谢。

# VS2010/MFC编程入门之五（MFC消息映射机制概述）

       上一讲鸡啄米为大家简单分析了[MFC应用程序框架](http://www.jizhuomi.com/software/145.html)，这一讲是关于MFC消息映射机制的内容。

       前面已经说过，Windows应用程序是消息驱动的。在MFC[软件开发](http://www.jizhuomi.com)中，界面操作或者线程之间通信都会经常用到消息，通过对消息的处理实现相应的操作。比较典型的过程是，用户操作窗口，然后有消息产生，送给窗口的消息处理函数处理，对用户的操作做出响应。

**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；(3)消息附带的LPARAM类型的参数。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。

**什么是消息映射机制**

       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。

**Windows消息分类**

       先讲下Windows消息的分类。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\_COMMAND，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。

**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。下面的讲解都以前面例程HelloWorld的CMainFrame为例。消息映射表如下：

**C++代码**

1. BEGIN\_MESSAGE\_MAP(CMainFrame, CFrameWndEx)
2. ON\_WM\_CREATE()
3. ON\_COMMAND(ID\_VIEW\_CUSTOMIZE, &CMainFrame::OnViewCustomize)
4. ON\_REGISTERED\_MESSAGE(AFX\_WM\_CREATETOOLBAR, &CMainFrame::OnToolbarCreateNew)
5. ON\_COMMAND\_RANGE(ID\_VIEW\_APPLOOK\_WIN\_2000, ID\_VIEW\_APPLOOK\_WINDOWS\_7, &CMainFrame::OnApplicationLook)
6. ON\_UPDATE\_COMMAND\_UI\_RANGE(ID\_VIEW\_APPLOOK\_WIN\_2000, ID\_VIEW\_APPLOOK\_WINDOWS\_7, &CMainFrame::OnUpdateApplicationLook)
7. ON\_WM\_SETTINGCHANGE()
8. END\_MESSAGE\_MAP()

       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。

**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。

**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。

       鸡啄米本节对MFC消息映射机制只是做了比较简单的讲解，让大家对它有一定的认识，[编程入门](http://www.jizhuomi.com/catalog.asp?tags=%E7%BC%96%E7%A8%8B%E5%85%A5%E9%97%A8)者不必强求完全掌握。在以后的教程中会经常涉及到消息的使用，大家会逐渐熟悉MFC的消息映射机制。

# VS2010/MFC编程入门之六（对话框：创建对话框模板和修改对话框属性）

       鸡啄米在上一讲中介绍了[MFC的消息映射机制](http://www.jizhuomi.com/software/147.html)，属于原理方面的知识。对于[VC++编程入门](http://www.jizhuomi.com/catalog.asp?tags=%E7%BC%96%E7%A8%8B%E5%85%A5%E9%97%A8)学习者来说可能有些抽象，鸡啄米会把消息映射的知识渗透到后面的教程中。本节开始为大家讲解偏应用的知识-创建对话框。

       对话框，大家应该很熟悉了，在我们常用的软件中大多都有对话框界面，例如，360安全卫士的主界面其实就是个对话框，只是它做了很多美工方面的工作，将其大大美化了。

**创建对话框主要分两大步，第一，创建对话框资源，主要包括创建新的对话框模板、设置对话框属性和为对话框添加各种控件；第二，生成对话框类，主要包括新建对话框类、添加控件变量和控件的消息处理函数等。**鸡啄米在本节中先讲讲怎样创建对话框模板和设置对话框属性。

**创建基于对话框的应用程序框架**

       之前鸡啄米创建的HelloWorld程序是[单文档应用程序](http://www.jizhuomi.com/software/141.html)，生成了多种窗口，如果用它来将讲创建对话框的话可能有些复杂，对大家单纯理解对话框有点影响，所以这里鸡啄米就再创建一个基于对话框的应用程序，用来实现加法运算的功能。创建步骤同单文档应用程序大同小异，简单步骤如下：

       1.选择菜单项File->New->Project，弹出“New Project”对话框。

       2.左侧面板中Installed Templated的Visual C++下选择MFC，中间窗口中选择MFC Application，然后在下面的Name编辑框中键入工程名称，本例取名“Addition”，在Location编辑框中设置工程的保存路径。点“OK”。

       3.点“Next”到“Application Type”对话框，在Application type下选择Dialog based，其他使用默认设置，点“Finish”。

       我们可以在Solution Explorer视图中看到，此工程的文件要比单文档应用程序少的多，在Class View中主要有三个类：CAboutDlg、CAdditionApp和CAdditionDlg。CAboutDlg是应用程序的“关于”对话框类，CAdditionApp是由CWinApp派生的类，CAdditionDlg是主对话框类，主对话框也就是此应用程序运行后显示的主要界面。

**注：**如果在VS2010中找不到Solution Explorer或Class View等视图，可以在菜单项View下找到对应视图选项选择即可。在[VS2010的使用介绍](http://www.jizhuomi.com/software/27.html)中已经有讲解。

       在Resource View视图中可以看到工程Addition的资源树，展开Addition.rc，下面有四个子项：Dialog（对话框）、Icon（图标）、String Table（字符串表）和Version（版本）。然后展开Dialog项，下面有两个对话框模板，其ID分别为：IDD\_ABOUTBOX和IDD\_ADDITION\_DIALOG，前者是“关于”对话框的模板，后者是主对话框的模板。ID是资源的唯一标识，本质上是一个无符号整数，一般ID代表的整数值由系统定义，我们无需干涉。

**对话框模板**

       可见对于主对话框来说，创建对话框第一步中的创建新的对话框模板已经由系统自动完成了。而如果是再添加对话框需要创建新的对话框模板时，需要在Resource View的“Dialog”节点上点右键，在右键菜单中选择“Insert Dialog”，就会生成新的对话框模板，并且会自动分配ID。

       在Resource View的资源树中双击某个ID，可在中间区域内显示相应的资源界面。双击IDD\_ADDITION\_DIALOG时，中间区域就会显示Addition对话框模板。如下图：

**设置对话框属性**

       在Addition对话框模板上点右键，然后在右键菜单中选择Properties，则在右侧面板中会显示对话框的属性列表。如下图：

       鸡啄米在这里对经常使用的几个属性作简单说明，并对Addition对话框进行属性设置说明。

       1.ID：对话框ID，唯一标识对话框资源，可以修改。此处为IDD\_ADDITION\_DIALOG，我们不修改它。

       2.Caption：对话框标题。此处默认为Addition，我们将其修改为“加法计算器”。

       3.Border：边框类型。有四种类型：None、Thin、Resizing和Dialog Frame。我们使用默认的Dialog Frame。

       4.Maximize：是否使用最大化按钮。我们使用默认的False。

       5.Minimize：是否使用最小化按钮。同样我们使用默认的False。

       6.Style：对话框类型。有三种类型：Overlapped（重叠窗口）、Popup（弹出式窗口）和Child（子窗口）。弹出式窗口比较常见。我们使用默认的Popup类型。

       7.System Menu：是否带有标题栏左上角的系统菜单，包括移动、关闭等菜单项。我们使用默认的True。

       8.Title Bar：是否带有标题栏。我们使用默认的True。

       9.Font(Size)：字体类型和字体大小。如果将其修改为非系统字体，则Use System自动改为False。而如果Use System原来为False，将其修改为True，则Font(Size)自动设置为系统字体。这里我们使用默认的系统字体。

       根据以上说明，其实我们只修改了标题属性。这时我们运行此程序后的界面如下：

       这一讲就先讲到这里了，对于创建对话框第一步中的为对话框添加各种控件下一讲为大家演示。欢迎来鸡啄米博客交流学习。

# VS2010/MFC编程入门之七（对话框：为对话框添加控件）

       创建对话框资源需要创建对话框模板、修改对话框属性、为对话框添加各种控件等步骤，前面一讲中鸡啄米已经讲了[创建对话框模板和修改对话框属性](http://www.jizhuomi.com/software/149.html)，本节继续讲如何为对话框添加控件。

       上一讲中鸡啄米创建了一个名为“Addition”的工程，目的是生成一个实现加法运算的应用程序。实现加法计算有几个必要的因素：被加数、加数、和。被加数和加数需要输入，和需要输出显示。那么这几个因素都需要相应的控件来输入或显示，下面鸡啄米就一步步讲解如何添加这些控件。

**1.为对话框添加一个静态文本框（Static Text），用于显示字符串--“被加数”。**

       上一讲中生成的资源模板中自动添加了一个标题为“TODO:Place dialog controls here.”的静态文本框，我们可以修改它的标题继续使用，也可以删掉它。这里为了从头讲解静态文本框的添加过程，将它删掉，继续添加新的静态文本框。

       删除控件时，可以使用鼠标左键点击选中它，选中后控件的周围会出现虚线框，然后按Delete键就可以将其删除了。在“Addition”工程的Resource View中打开上一讲中创建的对话框模板IDD\_ADDITION\_DIALOG，自动添加的静态文本框就可以使用这种方法删除。

       在添加新的静态文本框以前，先看看Toolbox视图是否显示了，如果没有显示，在菜单栏上点击View->Toolbox即可。Toolbox视图如下图：

       Toolbox中列出了一些常用控件，其中有一个是Static Text，即是我们要添加的控件。在Toolbox中的Static Text上点下鼠标左键不放开，并拖到IDD\_ADDITION\_DIALOG对话框模板上，模板上会出现一个虚线框，我们找到合适的位置松开鼠标左键放下它。

       用鼠标左键选中控件后周围出现虚线框，然后鼠标移到虚线框上几个黑点的位置会变成双向箭头的形状，此时就可以按下鼠标左键并拖动来改变控件大小了。我们可以这样改变新添加的静态文本框控件的大小，以更好的显示标题。当然，整个对话框模板也可以用这种方法改变大小。

       接下来就该修改静态文本框的文字了。鼠标右键点击静态文本框，在右键菜单中选择“Properties”，Properties面板就会显示出来，在面板上修改Caption属性为“被加数”，ID修改为IDC\_SUMMAND\_STATIC。此时模板如下图：

**2.为对话框添加一个编辑框（Edit Control），用来输入被加数。**

       添加编辑框的过程与静态文本框类似，在Toolbox中选中Edit Control控件拖到对话框模板上，并使其与之前的静态文本框水平对齐（为了美观），然后调整其大小使之适合被加数的输入。

       在编辑框上点右键，仍然在右键菜单中选择“Properties”显示出属性（Properties）面板，修改其ID为IDC\_SUMMAND\_EDIT。此时模板如下图：

**3.按照1的方法添加一个标题为“加数”的静态文本框，用于显示字符串--“加数”。**并将其ID改为IDC\_ADDEND\_STATIC。

**4.按照2的方法添加一个ID为IDC\_ADDEND\_EDIT的编辑框，用来输入加数。**

**5.按照1的方法添加一个标题为“和”的静态文本框，用于显示文字--“和”。**并修改其ID为IDC\_SUM\_STATIC。

**6.按照2的方法添加一个ID为IDC\_SUM\_EDIT的编辑框，用来显示最终的加和。**

**7.类似的添加按钮（Button）控件到对话框模板，用于在被点击后触发加法计算。**修改其标题为“计算”，ID为IDC\_ADD\_BUTTON。

       到此，对话框模板如图：

**8.删除OK按钮。打开Cancel按钮的属性面板，将标题改为“退出”，并使其与“计算”按钮水平对齐。**

**9.根据控件的布局，适当调整整个对话框模板的大小，使其相对控件布局来说大小合适，界面美观。**

       这样在对话框模板中就把我们在本例中需要用到的控件就添加完了。最终效果如下：

       至此，我们的对话框资源就基本创建完了。应用程序运行后的界面效果已经很清楚了。后面鸡啄米会讲如何在对话框类中实现加法计算功能，并能很好的和界面交互。欢迎继续到鸡啄米博客交流。

# VS2010/MFC编程入门之八（对话框：创建对话框类和添加控件变量）

       前两讲中鸡啄米为大家讲解了如何创建对话框资源。创建好对话框资源后要做的就是生成对话框类了。鸡啄米再声明下，生成对话框类主要包括新建对话框类、添加控件变量和控件的消息处理函数等。

       因为鸡啄米给大家的例程Addition是基于对话框的程序，所以程序自动创建了对话框模板IDD\_ADDITION\_DIALOG，并自动生成了对话框类CAdditionDlg，它是从CDialogEx类派生的。大家用过[VC++](http://www.jizhuomi.com/catalog.asp?tags=VC%EF%BC%8B%EF%BC%8B) 6.0的可能记得，我们定义的对话框类都是从CDialog类派生的，但在[VS2010](http://www.jizhuomi.com/software/27.html)中，一般对话框类都是继承自CDialogEx类。

**创建对话框类**

       如果是自己新添加的对话框模板，怎样为它创建对话框类呢？

       1.首先鸡啄米就按[第六讲：创建对话框模板和修改对话框属性](http://www.jizhuomi.com/software/149.html)中说的那样，在Resource View的“Dialog”节点上右键，然后在右键菜单中选择“Insert Dialog”创建一个新的对话框模板，ID就使用默认的IDD\_DIALOG1。

       2.在中间区域会显示新建的对话框模板，然后选中此对话框模板，点右键，在右键菜单中选择Add Class。

       3.选择“Add Class”后会弹出一个对话框，在对话框中“Class name”下的编辑框中写入自定义的类名就可以了，例如CMyDialog。

       4.最后点“Finish”完成。

       最终你就可以在Class View中看到新生成的对话框类CMyDialog了，并且在Solution Explorer中有相应的MyDialog.h头文件和MyDialog.cpp源文件生成。CMyDialog类同样派生于CDialogEx类。

**注意，一般类名都以C打头，**又比如，CTestDlg。

**为对话框中的控件添加变量**

       在上一讲中[为对话框添加了几个控件](http://www.jizhuomi.com/software/151.html)，包括三个静态文本框，三个编辑框，一个按钮控件。程序自动生成的Cancel按钮保留，作为退出按钮，而OK按钮删除掉了。

       静态文本框只是为了说明后面紧跟的编辑框中数据的意义，是被加数、加数还是和，所以它们是不会变的，我们就不为它们添加变量了。按钮控件是用来操作的，这里也不为它们添加变量。编辑框中的数据可能会经常变化，有必要为它们每个控件关联一个变量。

       首先为被加数的编辑框IDC\_SUMMAND\_EDIT添加变量。

       1.在编辑框上点右键，在右键菜单中选择“Add Variable”。弹出添加成员变量的向导对话框。

       2.我们想为其添加值变量而不是控件变量，所以对话框中“Category”下的组合框中选择Value。

       3.“Variable type”下的组合框此时默认选中的是“CString”，CString是字符串类，显然不能进行加法运算。我们可以选择double、float、int等。这里我们选择double，即编辑框关联一个double类型的变量。

       4.在“Variable name”中写入自定义的变量名。鸡啄米为其取名m\_editSummand。

       5.点“Finish”完成。

**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。**

       参照此方法，再分别为加数的编辑框IDD\_ADDEND\_EDIT添加double型变量m\_editAddend、和的编辑框IDD\_SUM\_EDIT添加double型变量m\_editSum。

**对话框类的数据交换和检验**

       在程序运行界面中，用户往往会改变控件的属性，例如，在编辑框中输入字符串，或者改变组合框的选中项，又或者改变复选框的选中状态等。控件的属性改变后[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)会相应修改控件关联变量的值。这种同步的改变是通过MFC为对话框类自动生成的成员函数DoDataExchange()来实现的，这也叫做对话框的数据交换和检验机制。

       我们为三个编辑框添加了变量以后，在AdditionDlg.cpp中CAdditionDlg的DoDataExchange()函数的函数体中多了三条DDX\_Text调用语句。下面是函数体代码和鸡啄米添加的注释。

**C++代码**

1. void CAdditionDlg::DoDataExchange(CDataExchange\* pDX)
2. {
3. // 处理MFC默认的数据交换
4. CDialogEx::DoDataExchange(pDX);
5. // 处理控件IDC\_SUMMAND\_EDIT和变量m\_editSummand之间的数据交换
6. DDX\_Text(pDX, IDC\_SUMMAND\_EDIT, m\_editSummand);
7. // 处理控件IDC\_ADDEND\_EDIT和变量m\_editAddend之间的数据交换
8. DDX\_Text(pDX, IDC\_ADDEND\_EDIT, m\_editAddend);
9. // 处理控件IDC\_SUM\_EDIT和变量m\_editSum之间的数据交换
10. DDX\_Text(pDX, IDC\_SUM\_EDIT, m\_editSum);
11. }

       鸡啄米再以Addition程序为例简单说下数据交换机制。如果我们在程序运行界面中输入被加数，则通过CAddition的DoDataExchange()函数可以将输入的值保存到m\_editSummand变量中，反之如果程序运行中修改了变量m\_editSummand的值，则通过CAddition的DoDataExchange()函数也可以将新的变量值显示到被加数的编辑框中。

       但是这种数据交换机制中，DoDataExchange()并不是被自动调用的，而是需要我们在程序中调用CDialogEx::UpdateData()函数，由UpdateData()函数再去自动调用DoDataExchange()的。

       CDialogEx::UpdateData()函数的原型为：

       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表示从变量传给数据。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。

       在下一讲中鸡啄米将具体演示CDialogEx::UpdateData()函数如何使用。

       鸡啄米本节主要讲的是新建对话框类和添加控件变量，控件的消息处理函数将在下一讲详细介绍。依然欢迎大家常回鸡啄米博客学习和讨论。

# VS2010/MFC编程入门之九（对话框：为控件添加消息处理函数）

[创建对话框类和添加控件变量](http://www.jizhuomi.com/software/153.html)在上一讲中已经讲过，这一讲的主要内容是如何为控件添加消息处理函数。

[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)为对话框和控件等定义了诸多[消息](http://www.jizhuomi.com/software/147.html)，我们对它们操作时会触发消息，这些消息最终由消息处理函数处理。比如我们点击按钮时就会产生BN\_CLICKED消息，修改编辑框内容时会产生EN\_CHANGE消息等。一般为了让某种操作达到效果，我们只需要实现某个消息的消息处理函数。

**一.添加消息处理函数**

       鸡啄米仍以前面的加法计算器的程序为例，说明怎样为“计算”按钮控件添加消息处理函数。添加方法列出4种：

**1.使用Class Wizard添加消息处理函数**

       用过的[VC++](http://www.jizhuomi.com/catalog.asp?tags=VC%EF%BC%8B%EF%BC%8B) 6.0的朋友应该对Class Wizard很熟悉了，添加[类](http://www.jizhuomi.com/software/49.html)、消息处理函数等经常会用到它，可以说是一个很核心的功能。但从VS2002开始就见不到Class Wizard了，大部分功能都集成到对话框和控件等的属性中了，使用很方便。到[VS2010](http://www.jizhuomi.com/software/27.html)，久违的Class Wizard又回来了。但鸡啄米已经习惯了使用属性中的功能了，对于从VC++ 6.0直接转VS2010的朋友可能觉得还是使用Class Wizard比较习惯。

       大家应该记得，“计算”按钮的ID为IDC\_ADD\_BUTTON，上图中Commands标签下，Oject IDs列表中有此ID，因为我们是想实现点击按钮后的消息处理函数，所以在Messages列表中选择BN\_CLICKED消息，然后点右上方的Add Handler就可以添加BN\_CLICKED消息处理函数OnClickedAddButton了。当然你也可以改名，但一般用的默认的就可以。

**2.通过“Add Event Handler...”添加消息处理函数**

       在“计算”按钮上点右键，然后在右键菜单中选择菜单项“Add Event Handler...”，弹出“Event Handler Wizard”对话框，如下图：

       可见“Message type”中默认选中的就是BN\_CLICKED消息，函数名和所在类都已经自动给出，直接点“Add and Edit”就可以了。

**3.在按钮的属性视图中添加消息处理函数**

       上面说过，从VS2002开始就主要从属性视图添加消息处理函数了。我们在“计算”按钮上点右键，在右键菜单中选择“Properties”，右侧面板中会显示按钮的属性视图。

       我们可以像上图中那样，点属性视图的“Control Events”按钮（类似闪电标志），下面列出了“计算”按钮的所有消息。我们要处理的是BN\_CLICKED消息，点其右侧空白列表项，会出现一个带下箭头的按钮，再点此按钮会出现“<Add> OnBnClickedAddButton”选项，最后选中这个选项就会自动添加BN\_CLICKED处理函数了。

**4.双击按钮添加消息处理函数**

       最直接最简单的方法就是，双击“计算”按钮，MFC会自动为其在CAdditionDlg类中添加BN\_CLICKED消息的处理函数OnBnClickedAddButton()。

**二.在消息处理函数中添加自定义功能**

       在我们使用任意一种方法添加了消息处理函数以后，都只能得到一个空的OnBnClickedAddButton()函数的函数体，要实现我们想要的功能，还需要在函数体中加入自定义功能代码。

       在加法计算器程序中，我们想要“计算”按钮实现的功能是，获取被加数和加数的数值，然后计算它们的和并显示到和的编辑框里。那么，OnBnClickedAddButton()的函数体就应修改为：

**C++代码**

1. void CAdditionDlg::OnBnClickedAddButton()
2. {
3. // TODO: Add your control notification handler code here
4. // 将各控件中的数据保存到相应的变量
5. UpdateData(TRUE);
7. // 将被加数和加数的加和赋值给m\_editSum
8. m\_editSum = m\_editSummand + m\_editAddend;
10. // 根据各变量的值更新相应的控件。和的编辑框会显示m\_editSum的值
11. UpdateData(FALSE);
12. }

       鸡啄米在上面的代码中已经添加注释，大家应该很容易理解了。对于UpdateData()函数的说明在上一讲中已经介绍过，如果忘了可以再回上一讲了解了解。

       接下来我们运行下此应用程序。在运行结果界面中，输入被加数5.1，加数2.3，然后点“计算”：

       在上图中可以看到，点“计算”按钮后，和的编辑框中显示了正确结果：7.4。

       鸡啄米简单分析下运行过程：输入被加数和加数，点“计算”按钮后产生点击消息，从而调用OnBnClickedAddButton()函数。进入此函数后，首先由UpdateData(TRUE)函数将被加数的值5.1和加数的值2.3分别保存到变量m\_editSummand和m\_editAddend，然后通过语句m\_editSum = m\_editSummand + m\_editAddend;计算出被加数和加数的和为7.4，并把7.4赋值给m\_editSum。最后调用UpdateData(FALSE)根据被加数、加数、和的值更新三个编辑框的显示值，就得到了上图中的结果。

       到此，一个具有简单的加法运算功能的加法计算器应用程序就基本完成了。如果大家想实现其他功能，可以修改控件资源和消息处理函数来练习下。本节就讲到这里了，有问题欢迎到鸡啄米博客或者我们的编程入门qq群讨论。

# VS2010/MFC编程入门之十（对话框：设置对话框控件的Tab顺序）

       前面几节鸡啄米为大家演示了加法计算器程序完整的编写过程，本节主要讲**对话框上控件的Tab顺序如何调整**。

       上一讲为“计算”按钮[添加了消息处理函数](http://www.jizhuomi.com/software/156.html)后，加法计算器已经能够进行浮点数的加法运算。但是还有个遗留的小问题，就是对话框控件的Tab顺序问题。

       运行加法计算器程序，显示对话框后不进行任何操作，直接按回车，可以看到对话框退出了。这是因为“退出”按钮是Tab顺序为1的控件，也就是第一个接受用户输入的控件。但是按照我们的输入习惯，应该是被加数的编辑框首先接受用户输入，然后是加数编辑框，再接下来是“计算”按钮，最后才是“退出”按钮。

       我们先来直观的看看各个控件的Tab顺序吧。打开“Resource View”视图，然后在资源中找到对话框IDD\_ADDITION\_DIALOG，双击ID后中间客户区域出现其模板视图。**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。**如下图：

       上图中每个控件左上角都有一个数字，这就是它的Tab响应顺序。对话框刚打开时输入焦点就在Tab顺序为1的“退出”按钮上，不做任何操作按下Tab键，输入焦点就会转移到Tab顺序为2的“被加数”静态文本框上，但是因为静态文本框不接受任何输入，所以输入焦点继续自动转移到Tab顺序为3的被加数编辑框，再按Tab键，输入焦点又会转移到Tab顺序为4的“加数”静态文本框上，同样由于它是静态文本框，输入焦点不停留继续转移到加数编辑框，后面的控件同理。

       我们认为这个顺序不合理，那怎么修改呢？很简单，**从自己认为Tab顺序应该为1的控件开始依次单击**，随着单击的完成，各控件的Tab响应顺序也按我们的想法设置好了。

       例如，此例中我们可以依次单击被加数编辑框、“被加数”静态文本框、加数编辑框、“加数”静态文本框、和编辑框、“和”静态文本框、“计算”按钮和“退出”按钮。设置完后如下图：

       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。

       现在我们再运行程序，可以看到对话框打开后最初的输入焦点在被加数编辑框上，然后我们按Tab键，输入焦点移到加数编辑框上，继续多次按Tab键时，输入焦点会按“和编辑框--‘计算’按钮--‘退出’按钮--被加数编辑框--加数编辑框--和编辑框......”的顺序循环转移。这样就达到了我们的目的。

# VS2010/MFC编程入门之十一（对话框：模态对话框及其弹出过程）

       加法计算器对话框程序大家照着做一遍后，相信对基于对话框的程序有些了解了，有个好的开始对于以后的学习大有裨益。趁热打铁，鸡啄米这一节讲讲什么是模态对话框和非模态对话框，以及模态对话框怎样弹出。

**一.模态对话框和非模态对话框**

       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。

       大家对Windows系统很了解，相信这两种对话框应该都遇到过。之前的加法计算器对话框其实就是模态对话框。

**二.模态对话框是怎样弹出的**

       毕竟加法计算器程序大部分都是MFC自动生成的，对话框怎么弹出来的大家可能还不是很清楚。鸡啄米下面简单说说它是在哪里弹出来的，再重新建一个新的对话框并弹出它，这样大家实践以后就能更灵活的使用模态对话框了。

       大家打开Addition.cpp文件，可以看到CAdditionApp类有个InitInstance()函数，在[MFC应用程序框架分析](http://www.jizhuomi.com/software/145.html)中提到过此函数，不过那是单文档应用程序App类中的，函数体不太相同，但都是进行App类实例的初始化工作。

       InitInstance()函数的后半部分有一段代码就是定义对话框对象并弹出对话框的，鸡啄米下面给出这段代码并加以注释：

**C++代码**

1. CAdditionDlg dlg;        // 定义对话框类CAdditionDlg的对象dlg
2. m\_pMainWnd = &dlg;       // 将dlg设为主窗口
3. INT\_PTR nResponse = dlg.DoModal();   // 弹出对话框dlg，并将DoModal函数的返回值（退出时点击按钮的ID）赋值给nResponse
4. if (nResponse == IDOK)               // 判断返回值是否为OK按钮（其ID为IDOK，鸡啄米已经将它删除）
5. {
6. // TODO: Place code here to handle when the dialog is
7. //  dismissed with OK
8. }
9. else if (nResponse == IDCANCEL)      // 判断返回值是否为Cancel按钮（其ID为IDCANCEL，鸡啄米将它的Caption改为了“退出”）
10. {
11. // TODO: Place code here to handle when the dialog is
12. //  dismissed with Cancel
13. }

       弹出对话框比较关键的一个函数，就是对话框类的DoModal()函数。CDialog::DoModal()函数的原型为：

       virtual INT\_PTR DoModal();

       返回值：整数值，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。

       调用了它对话框就会弹出，返回值是退出对话框时所点的按钮的ID，比如，我们点了“退出”按钮，那么DoModal返回值为IDCANCEL。

**三.添加一个新对话框并弹出它**

       鸡啄米再为加法计算器程序添加一个对话框，以在计算之前询问用户是否确定要进行计算。大家可以完整的看下对话框的添加和弹出过程。

       1.根据“[创建对话框模板和修改对话框属性](http://www.jizhuomi.com/software/149.html)”中所讲的方法，在Resource View中的“Dialog”上点右键选择“Insert Dialog”，创建一个新的对话框模板，修改其ID为IDD\_TIP\_DIALOG，Caption改为“提示”，然后参考“[为对话框添加控件](http://www.jizhuomi.com/software/151.html)”中所讲，在对话框模板上添加一个静态文本框（static text），Caption改为“您确定要进行加法计算吗？”，接下来修改OK按钮的Caption为“确定”，Cancel按钮的Caption为“取消”，最后调整各个控件的位置和对话框的大小。最终的对话框模板如下图：

       2.根据“[创建对话框类和添加控件变量](http://www.jizhuomi.com/software/153.html)”中创建对话框类的方法，在对话框模板上点右键选择“Add Class...”，弹出添加类的对话框，设置“Class name”为CTipDlg，点“OK”。在Solution Explorer中可以看到生成了CTipDlg类的头文件TipDlg.h和源文件TipDlg.cpp。

       3.我们要在点“计算”按钮之后弹出此提示对话框，那么就要在“计算”按钮的消息处理函数OnBnClickedAddButton()中访问提示对话框类，所以为了访问CTipDlg类，在AdditionDlg.cpp中包含CTipDlg的头文件：#include "TipDlg.h"。

       4.修改OnBnClickedAddButton()的函数体，在所有代码前，构造CTipDlg类的对象tipDlg，并通过语句tipDlg.DoModal();弹出对话框，最后判断DoModal()函数的返回值是IDOK还是IDCANCEL来确定是否继续进行计算。OnBnClickedAddButton()函数修改后如下：

**C++代码**

1. void CAdditionDlg::OnBnClickedAddButton()
2. {
3. // TODO: Add your control notification handler code here
4. INT\_PTR nRes;             // 用于保存DoModal函数的返回值
6. CTipDlg tipDlg;           // 构造对话框类CTipDlg的实例
7. nRes = tipDlg.DoModal();  // 弹出对话框
8. if (IDCANCEL == nRes)     // 判断对话框退出后返回值是否为IDCANCEL，如果是则return，否则继续向下执行
9. return;
11. // 将各控件中的数据保存到相应的变量
12. UpdateData(TRUE);
14. // 将被加数和加数的加和赋值给m\_editSum
15. m\_editSum = m\_editSummand + m\_editAddend;
17. // 根据各变量的值更新相应的控件。和的编辑框会显示m\_editSum的值
18. UpdateData(FALSE);
19. }

       5.测试。编译运行程序后，在对话框上输入被加数和加数，点“计算”，弹出提示对话框询问是否进行计算，如果选择“确定”，则提示对话框退出，并在主对话框上显示被加数和加数的和，而如果选择“取消”，则提示对话框也会退出，但主对话框显示的和不变，即没有进行加法计算。

# VS2010/MFC编程入门之十二（对话框：非模态对话框的创建及显示）

       上一节鸡啄米讲了[模态对话框及其弹出过程](http://www.jizhuomi.com/software/160.html)，本节接着讲另一种对话框--非模态对话框的创建及显示。

       鸡啄米已经说过，非模态对话框显示后，程序其他窗口仍能正常运行，可以响应用户输入，还可以相互切换。鸡啄米会将上一讲中创建的Tip模态对话框改为非模态对话框，让大家看下效果。

**非模态对话框的**[**对话框资源**](http://www.jizhuomi.com/software/149.html)**和**[**对话框类**](http://www.jizhuomi.com/software/153.html)

       实际上，模态对话框和非模态对话框在创建对话框资源和生成对话框类上是没有区别的，所以上一讲中创建的IDD\_TIP\_DIALOG对话框资源和CTipDlg类都不需要修改。

**创建及显示非模态对话框的步骤**

       需要修改的是，对话框类实例的创建和显示，也就是之前在CAdditionDlg::OnBnClickedAddButton()函数体中添加的对话框显示代码。下面是具体步骤：

       1.在AdditionDlg.h中包含CTipDlg头文件并定义CTipDlg类型的指针成员变量。详细操作方法是，在AdditionDlg.cpp中删除之前添加的#include "TipDlg.h"，而在AdditionDlg.h中添加#include "TipDlg.h"，这是因为我们需要在AdditionDlg.h中定义CTipDlg类型的指针变量，所以要先包含它的头文件；然后在AdditionDlg.h中为CAdditionDlg类添加private成员变量CTipDlg  \*m\_pTipDlg;。

       2.在CAdditionDlg类的[构造函数](http://www.jizhuomi.com/software/51.html)中初始化成员变量m\_pTipDlg。如果cpp文件中函数太多，我们可以在Class View上半个视图中找到CAdditionDlg类，再在下半个视图中找到其构造函数双击，中间客户区域即可马上切到构造函数的实现处。在构造函数体中添加m\_pTipDlg = NULL;，这是个好习惯，鸡啄米在C++编程入门系列的[指针的赋值和指针运算](http://www.jizhuomi.com/software/78.html)中说到过，在任何指针变量使用前都初始化，可以避免因误访问重要内存地址而破坏此地址的数据。

       3.将上一讲中添加的模态对话框显示代码注释或删除掉，添加非模态对话框的创建和显示代码。VC++中注释单行代码使用“//”，注释多行代码可以在需注释的代码开始处添加“/\*”，结束处添加“\*/”。修改后的CAdditionDlg::OnBnClickedAddButton()函数如下：

**C++代码**

1. void CAdditionDlg::OnBnClickedAddButton()
2. {
3. // TODO: Add your control notification handler code here
4. /\*INT\_PTR nRes;             // 用于保存DoModal函数的返回值
6. CTipDlg tipDlg;           // 构造对话框类CTipDlg的实例
7. nRes = tipDlg.DoModal();  // 弹出对话框
8. if (IDCANCEL == nRes)     // 判断对话框退出后返回值是否为IDCANCEL，如果是则return，否则继续向下执行
9. return;\*/
11. // 如果指针变量m\_pTipDlg的值为NULL，则对话框还未创建，需要动态创建
12. if (NULL == m\_pTipDlg)
13. {
14. // 创建非模态对话框实例
15. m\_pTipDlg = new CTipDlg();
16. m\_pTipDlg->Create(IDD\_TIP\_DIALOG, this);
17. }
18. // 显示非模态对话框
19. m\_pTipDlg->ShowWindow(SW\_SHOW);
21. // 将各控件中的数据保存到相应的变量
22. UpdateData(TRUE);
24. // 将被加数和加数的加和赋值给m\_editSum
25. m\_editSum = m\_editSummand + m\_editAddend;
27. // 根据各变量的值更新相应的控件。和的编辑框会显示m\_editSum的值
28. UpdateData(FALSE);
29. }

       4.因为此非模态对话框实例是动态创建的，所以需要手动删除此动态对象来销毁对话框。我们在CAdditionDlg类的[析构函数](http://www.jizhuomi.com/software/51.html)中添加删除代码，但是MFC并没有自动给出析构函数，这时需要我们手动添加，在对话框对象析构时就会调用我们自定义的析构函数了。在AdditionDlg.h文件中为CAdditionDlg添加析构函数声明：~CAdditionDlg();，然后在AdditionDlg.cpp文件中添加析构函数的实现，函数体如下：

**C++代码**

1. CAdditionDlg::~CAdditionDlg()
2. {
3. // 如果非模态对话框已经创建则删除它
4. if (NULL != m\_pTipDlg)
5. {
6. // 删除非模态对话框对象
7. delete m\_pTipDlg;
8. }
9. }

       这样，非模态对话框创建和显示的代码就添加修改完了。让我们运行下看看效果吧。

       在加法计算器对话框上输入被加数和加数，然后点“计算”按钮，依然像上节一样弹出了提示对话框，但是先不要关闭它，你可以拖动它后面的加法计算器对话框试试，我们发现加法计算器对话框竟然可以拖动了，而且“和”编辑框里已经显示了运算结果，这表明提示对话框显示以后还没有关闭，OnBnClickedAddButton() 就继续向下执行了，不仅如此，加法计算器的每个编辑框还都可以响应输入。

       这只是一个简单的例子，非模态对话框的用处有很多，以后大家在软件开发中会用到。

# VS2010/MFC编程入门之十三（对话框：属性页对话框及相关类的介绍）

       前面讲了[模态对话框](http://www.jizhuomi.com/software/160.html)和[非模态对话框](http://www.jizhuomi.com/software/162.html)，本节开始鸡啄米讲一种特殊的对话框--属性页对话框。另外，本套教程所讲大部分对VC++各个版本均可适用或者稍作修改即可，但考虑到终究还是基于[VS2010](http://www.jizhuomi.com/catalog.asp?tags=VS2010)版本的，所以将《VC++/MFC编程入门》改为《VS2010/MFC编程入门》。

**属性页对话框的分类**

       属性页对话框想必大家并不陌生，XP系统中桌面右键点属性，弹出的就是属性页对话框，它通过标签切换各个页面。另外，我们在创建MFC工程时使用的向导对话框也属于属性页对话框，它通过点击“Next”等按钮来切换页面。

       属性页对话框就是包含一般属性页对话框和向导对话框两类。它将多个对话框集成于一身，通过标签或按钮来切换页面。

**属性页对话框相关类**

我们使用属性页对话框时，用到的类主要有两个：CPropertyPage类和CPropertySheet类。

**1.CPropertyPage类**

       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。大家可以在[VS2010的MSDN](http://www.jizhuomi.com/software/139.html)中查找CPropertyPage类以及它的成员的详细说明。下面鸡啄米就为大家讲解MSDN中列出的CPropertyPage类的部分主要成员函数。

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

        这里讲三个CProperty类的构造函数，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“Next”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“下一步”按钮被单击的消息，仅在向导对话框中有效

**2.CPropertySheet类**

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。下面鸡啄米就讲解CPropertySheet类的部分成员函数。

      （1）构造函数

       这里依然列出CPropertySheet类的三个构造函数：

       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。

# VS2010/MFC编程入门之十四（对话框：向导对话框的创建及显示）

       上一讲鸡啄米讲了[属性页对话框和相关的两个类CPropertyPage类和CPropertySheet类](http://www.jizhuomi.com/software/164.html)，对使用属性页对话框做准备。本节将为大家演示如何创建向导对话框。

       仍然以前面的“加法计算器”的例子为基础，在其中加入向导对话框，我们可以用它来说明加法计算器的使用方法，一步一步引导用户操作，这也是比较常见的用法。

       加法计算器使用时大概可以分为三步：输入被加数、输入加数、点“计算”按钮。

       鸡啄米就详细说明向导对话框的创建步骤：

**1.创建属性页对话框资源**

       根据[创建对话框模板和修改对话框属性](http://www.jizhuomi.com/software/149.html)中所讲方法，在“Resource View”的Dialog”节点上点右键，然后在右键菜单中选择“Insert Dialog”创建第一个对话框模板，对话框的ID属性设置为IDD\_SUMMAND\_PAGE，Caption属性改为“被加数页”，Style属性在下拉列表中选择“Child”，Border属性在下拉列表中选择“Thin”。

       删除“OK”和“Cancel”按钮，再按照[为对话框添加控件](http://www.jizhuomi.com/software/151.html)中所讲方法，添加一个静态文本框，并修改静态文本框的Caption属性为“请先输入double型被加数”。

       按照上述步骤，继续添加第二个和第三个对话框资源。第二个对话框模板的ID设为IDD\_ADDEND\_PAGE，Caption属性改为“加数页”，也添加一个静态文本框，Caption设为“请继续输入double型加数”，其他属性同第一个对话框。第三个对话框模板的ID设为IDD\_ADD\_PAGE，Caption属性改为“计算页”，添加静态文本框的Caption属性改为“最后请按下“计算”按钮”，其他属性也第一个对话框一样。

**2.创建属性页类**

       按照[创建对话框类和添加控件变量](http://www.jizhuomi.com/software/153.html)中的方法，在第一个对话框模板上点右键，在右键菜单中选择“Add Class”，弹出类向导对话框，在“Class name”编辑框中输入类名“CSummandPage”，与之前不同的是，因为属性页类都应继承于CPropertyPage类，所以要修改下面“Base class”的选项，在下拉列表中选择“CPropertyPage”。

       因为是第一个属性页，所以它应该有一个“下一步”按钮，在哪里添加呢？上一讲[CPropertyPage类的可重载函数](http://www.jizhuomi.com/software/164.html)中提到，OnSetActive函数用于处理属性页被切换为当前活动页的消息，所以我们可以在OnSetActive函数中进行相关设置。

       那怎样重载OnSetActive函数呢？我们可以在“Class View”中找到“CSummandPage”节点，点右键弹出右键菜单，选择“Properties”，然后VS2010右侧面板上会显示对话框的属性列表，属性列表的工具栏上有个tip信息为“Overrides”的按钮，按下它，下方列表中就列出了重载函数，找到“OnSetActive”，点其右侧空白列表项出现向下箭头，再点箭头就在下面出现了“<Add>OnSetActive”的选项，选择它就会自动在CSummandPage类中添加函数OnSetActive。

       我们只需在OnSetActive函数体中添加相关代码就可以实现添加“下一步”按钮的效果了。新的函数体如下：

**C++代码**

1. BOOL CSummandPage::OnSetActive()
2. {
3. // TODO: Add your specialized code here and/or call the base class
5. // 获得父窗口，即属性表CPropertySheet类
6. CPropertySheet\* psheet = (CPropertySheet\*) GetParent();
7. // 设置属性表只有“下一步”按钮
8. psheet->SetWizardButtons(PSWIZB\_NEXT);
10. return CPropertyPage::OnSetActive();
11. }

       为第二个和第三个对话框也分别添加属性页类CAddendPage和CAddPage。但第二个对话框的属性页不需要重载OnSetActive函数。第三个对话框是最后一个对话框，所以不需要“下一步”按钮，而应该换成“完成”按钮，所以也需要重载OnSetActive函数设置“完成”按钮。重载后的OnSetActive如下：

**C++代码**

1. BOOL CAddPage::OnSetActive()
2. {
3. // TODO: Add your specialized code here and/or call the base class
5. // 获得父窗口，即属性表CPropertySheet类
6. CPropertySheet\* psheet = (CPropertySheet\*) GetParent();
7. //设置属性表只有“完成”按钮
8. psheet->SetFinishText(\_T("完成"));
10. return CPropertyPage::OnSetActive();
11. }

       上面的代码段中，字符串“完成”前加了个\_T，这是因为本工程创建的时候用的默认的Unicode字符集，而如果“完成”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。

       那点了第三个属性页上的“完成”按钮我们想进行某些处理的话，就重载OnWizardFinish函数，方法同OnSetActive函数。重载后的OnWizardFinish函数如下：

**C++代码**

1. BOOL CAddPage::OnWizardFinish()
2. {
3. // TODO: Add your specialized code here and/or call the base class
5. // 提示向导完成
6. MessageBox(\_T("使用说明向导已阅读完！"));
8. return CPropertyPage::OnWizardFinish();
9. }

**3.创建属性表类**

       属性页资源和属性页类创建完以后，还不能生成向导对话框，我们还需要一个属性表类，来容纳这些属性页。

       在Solution Explorer视图中的根节点“Addition”上点右键，在右键菜单中选择Add->Class，弹出“Add Class”对话框，然后在中间区域中选择“MFC Class”，点“Add”按钮，弹出另一个类向导对话框，设置Class name为CAddSheet，Base class选择“CPropertySheet”，点“Finish”按钮，这样就属性表类就建好了。

       接下来，在新生成的AddSheet.h中包含三个属性页类的头文件：

       #include "SummandPage.h"  
       #include "AddendPage.h"  
       #include "AddPage.h"

       之后在AddSheet.h中添加private变量：

       CSummandPage    m\_summandPage;  
       CAddendPage     m\_addendPage;  
       CAddPage        m\_addPage;

       然后在AddSheet.cpp文件中修改CAddSheet的两个构造函数为：

**C++代码**

1. CAddSheet::CAddSheet(UINT nIDCaption, CWnd\* pParentWnd, UINT iSelectPage)
2. :CPropertySheet(nIDCaption, pParentWnd, iSelectPage)
3. {
4. // 添加三个属性页到属性表
5. AddPage(&m\_summandPage);
6. AddPage(&m\_addendPage);
7. AddPage(&m\_addPage);
8. }
10. CAddSheet::CAddSheet(LPCTSTR pszCaption, CWnd\* pParentWnd, UINT iSelectPage)
11. :CPropertySheet(pszCaption, pParentWnd, iSelectPage)
12. {
13. // 添加三个属性页到属性表
14. AddPage(&m\_summandPage);
15. AddPage(&m\_addendPage);
16. AddPage(&m\_addPage);
17. }

**4.显示向导对话框**

        我们在加法计算器对话框上添加一个按钮，点击它就打开向导对话框。此按钮的ID设为IDC\_INSTRUCT\_BUTTON，Caption属性设为“使用说明”。

        按照[为控件添加消息处理函数](http://www.jizhuomi.com/software/156.html)中所讲方法，为IDC\_INSTRUCT\_BUTTON按钮在CAdditionDlg类中添加点击消息的处理函数OnBnClickedInstructButton。然后在AdditionDlg.cpp文件中包含CAddSheet的头文件：#include "AddSheet.h"。最后修改OnBnClickedInstructButton函数如下：

**C++代码**

1. void CAdditionDlg::OnBnClickedInstructButton()
2. {
3. // TODO: Add your control notification handler code here
5. // 创建属性表对象
6. CAddSheet sheet(\_T(""));
7. // 设置属性对话框为向导对话框
8. sheet.SetWizardMode();
9. // 打开模态向导对话框
10. sheet.DoModal();
11. }

       到此，向导对话框就完整的创建完成了，并可以在加法计算器对话框上点“使用说明”按钮显示出来。我们来看看效果吧：

       上图只是被加数页的效果，点其上“下一步”按钮就可以继续显示后面的两个页面。

# VS2010/MFC编程入门之十五（对话框：一般属性页对话框的创建及显示）

       属性页对话框包括向导对话框和一般属性页对话框两类，上一节鸡啄米讲了[如何创建并显示向导对话框](http://www.jizhuomi.com/software/166.html)，本节将继续介绍一般属性页对话框的创建和显示。

       实际上，一般属性页对话框的创建和显示过程和向导对话框是很类似的。鸡啄米将上一节中的向导对话框进行少量修改，使其成为一般属性页对话框。

       一般属性页对话框的创建步骤：

**1.创建属性页对话框资源**

       属性页对话框资源的创建方法同向导对话框是一样的，上一讲中的对话框资源不需进行任何修改。

**2.创建属性页类**

       属性页类的创建和向导对话框的属性页类也基本一样，只是一般属性页对话框中不需要“下一步”和“完成”等按钮，所以上一讲中属性页类的OnSetActive和OnWizardFinish等重载函数可以去掉。即CSummandPage类中的OnSetActive函数、CAddPage类中的OnSetActive函数和OnWizardFinish函数可以删除或注释掉。其他部分不需作任何修改。

**3.创建属性表类**

       创建属性表类的过程同向导对话框属性表类也是一样的，所以上一讲中的CAddSheet类不需修改。

**4.显示一般属性页对话框**

       上一讲向导对话框的显示是在OnBnClickedInstructButton函数中实现的，其中语句sheet.SetWizardMode();旨在设置属性表为向导对话框模式，所以显示一般属性页对话框时不需调用SetWizardMode成员函数。另外，我们可以将属性页对话框的标题设为“使用说明”，在构造属性表对象时将此字符串作为[构造函数](http://www.jizhuomi.com/software/51.html)的参数传入。OnBnClickedInstructButton函数修改如下：

**C++代码**

1. void CAdditionDlg::OnBnClickedInstructButton()
2. {
3. // TODO: Add your control notification handler code here
5. // 创建属性表对象
6. CAddSheet sheet(\_T("使用说明"));
8. // 打开模态一般属性页对话框
9. sheet.DoModal();
10. }

       这样一般属性页对话框的创建和显示就讲完了，我们运行下程序，在结果对话框上点“使用说明”按钮看看效果吧：

       再总结下，一般属性页对话框和向导对话框的创建和显示的不同包括，**是否需要OnSetActive和OnWizardFinish等重载函数，是否需要调用属性表类的SetWizardMode函数设置为向导对话框模式。**

# VS2010/MFC编程入门之十六（对话框：消息对话框）

       前面几节鸡啄米讲了[属性页对话框](http://www.jizhuomi.com/software/164.html)，我们可以根据所讲内容方便的建立自己的属性页对话框。本节讲解Windows系统中最常用最简单的一类对话框--消息对话框。

       我们在使用Windows系统的过程中经常会见到消息对话框，提示我们有异常发生或提出询问等。因为在软件开发中经常用到消息对话框，所以MFC提供了两个函数可以直接生成指定风格的消息对话框，而不需要我们在每次使用的时候都要去创建对话框资源和生成对话框类等。这两个函数就是CWnd类的成员函数MessageBox()和全局函数AfxMessageBox()。

**一.CWnd::MessageBox()函数和AfxMessageBox()函数的用法**

       下面鸡啄米就分别讲解两个函数的用法。

**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 |  |
| MB\_ICONASTERISK MB\_ICONINFORMATION |  |
| MB\_ICONQUESTION |  |
| MB\_ICONHAND MB\_ICONSTOP MB\_ICONERROR |  |

 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。

**二.CWnd::MessageBox()和AfxMessageBox()的返回值**

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。

**三.应用举例**

我们还是拿前面加法计算器的程序做例子。

       大家是否记得，在[模态对话框及其弹出过程](http://www.jizhuomi.com/software/160.html)中我们修改了CAdditionDlg::OnBnClickedAddButton()函数，在点了“计算”按钮以后先弹出了一个模态对话框，询问用户是否确定要进行加法计算，并通过模态对话框DoModal函数的返回值判断用户选择了“确定”还是“取消”。这些功能很明显消息对话框完全能够实现，鸡啄米就使用消息对话框来替代原来的模态对话框。

       在[非模态对话框的创建及显示](http://www.jizhuomi.com/software/162.html)中，鸡啄米注释了模态对话框的相关代码，加入了非模态对话框的创建和显示代码，我们在加入消息对话框之前将非模态对话框的代码也注释或删除掉，确保此函数中不再生成原来的模态对话框或非模态对话框。

       修改后的CAdditionDlg::OnBnClickedAddButton()函数如下：

**C++代码**

1. void CAdditionDlg::OnBnClickedAddButton()
2. {
3. // TODO: Add your control notification handler code here
5. INT\_PTR nRes;
7. // 显示消息对话框
8. nRes = MessageBox(\_T("您确定要进行加法计算吗？"), \_T("加法计算器"), MB\_OKCANCEL | MB\_ICONQUESTION);
9. // 判断消息对话框返回值。如果为IDCANCEL就return，否则继续向下执行
10. if (IDCANCEL == nRes)
11. return;
13. // 将各控件中的数据保存到相应的变量
14. UpdateData(TRUE);
16. // 将被加数和加数的加和赋值给m\_editSum
17. m\_editSum = m\_editSummand + m\_editAddend;
19. // 根据各变量的值更新相应的控件。和的编辑框会显示m\_editSum的值
20. UpdateData(FALSE);
21. // 设置属性对话框为向导对话框
22. //sheet.SetWizardMode();
23. }

        编译运行，在运行结果对话框上点“计算”按钮弹出以下消息对话框：

       大家也可以将MessageBox函数换为AfxMessageBox()函数，同时参数进行相应修改，运行下看看效果。

# VS2010/MFC编程入门之十七（对话框：文件对话框）

       上一讲鸡啄米介绍的是[消息对话框](http://www.jizhuomi.com/software/171.html)，本节讲解文件对话框。文件对话框也是很常用的一类对话框。

**文件对话框的分类**

文件对话框分为打开文件对话框和保存文件对话框，相信大家在Windows系统中经常见到这两种文件对话框。例如，很多编辑软件像记事本等都有“打开”选项，选择“打开”后会弹出一个对话框，让我们选择要打开文件的路径，这个对话框就是打开文件对话框；除了“打开”选项一般还会有“另存为”选项，选择“另存为”后往往也会有一个对话框弹出，让我们选择保存路径，这就是保存文件对话框。

       正如上面举例说明的，打开文件对话框用于选择要打开的文件的路径，保存文件对话框用来选择要保存的文件的路径。

**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。

**文件对话框实例**

根据前面所讲内容，鸡啄米给大家做个文件对话框实例。

       1.创建一个基于对话框的MFC应用程序工程，名称设为“Example17”。

       2.修改主对话框IDD\_EXAMPLE17\_DIALOG的模板，删除自动生成的“TODO: Place dialog controls here.”静态文本框，添加两个编辑框，ID分别为IDC\_OPEN\_EDIT和IDC\_SAVE\_EDIT，再添加两个按钮，ID分别设为IDC\_OPEN\_BUTTON和IDC\_SAVE\_BUTTON，Caption分别设为“打开”和“保存”。按钮IDC\_OPEN\_BUTTON用于显示打开文件对话框，编辑框IDC\_OPEN\_EDIT显示在打开文件对话框中选择的文件路径。按钮IDC\_SAVE\_BUTTON用于显示保存文件对话框，编辑框IDC\_SAVE\_BUTTON显示在保存文件对话框中选择的文件路径。

       3.分别为按钮IDC\_OPEN\_BUTTON和IDC\_SAVE\_BUTTON添加点击消息的[消息处理函数](http://www.jizhuomi.com/software/156.html)CExample17Dlg::OnBnClickedOpenButton()和CExample17Dlg::OnBnClickedSaveButton()。

       4.修改两个消息处理函数如下：

**C++代码**

1. void CExample17Dlg::OnBnClickedOpenButton()
2. {
3. // TODO: Add your control notification handler code here
4. // 设置过滤器
5. TCHAR szFilter[] = \_T("文本文件(\*.txt)|\*.txt|所有文件(\*.\*)|\*.\*||");
6. // 构造打开文件对话框
7. CFileDialog fileDlg(TRUE, \_T("txt"), NULL, 0, szFilter, this);
8. CString strFilePath;
10. // 显示打开文件对话框
11. if (IDOK == fileDlg.DoModal())
12. {
13. // 如果点击了文件对话框上的“打开”按钮，则将选择的文件路径显示到编辑框里
14. strFilePath = fileDlg.GetPathName();
15. SetDlgItemText(IDC\_OPEN\_EDIT, strFilePath);
16. }
17. }

20. void CExample17Dlg::OnBnClickedSaveButton()
21. {
22. // TODO: Add your control notification handler code here
23. // 设置过滤器
24. TCHAR szFilter[] = \_T("文本文件(\*.txt)|\*.txt|Word文件(\*.doc)|\*.doc|所有文件(\*.\*)|\*.\*||");
25. // 构造保存文件对话框
26. CFileDialog fileDlg(FALSE, \_T("doc"), \_T("my"), OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT, szFilter, this);
27. CString strFilePath;
29. // 显示保存文件对话框
30. if (IDOK == fileDlg.DoModal())
31. {
32. // 如果点击了文件对话框上的“保存”按钮，则将选择的文件路径显示到编辑框里
33. strFilePath = fileDlg.GetPathName();
34. SetDlgItemText(IDC\_SAVE\_EDIT, strFilePath);
35. }
36. }

       上面显示编辑框内容时，鸡啄米使用了Windows API函数SetDlgItemText，当然也可以先给编辑框关联变量，然后再使用鸡啄米在[创建对话框类和添加控件变量](http://www.jizhuomi.com/software/153.html)中介绍的  
CDialogEx::UpdateData()函数，但是鸡啄米比较习惯使用SetDlgItemText函数，感觉比较灵活。

       5.运行此程序，在结果对话框上点“打开”按钮，显示打开文件对话框如下：

       点“保存”按钮后，显示保存文件对话框：

       在打开文件对话框和保存文件对话框都选择了文件路径后，主对话框如下：

       到此，文件对话框就讲完了，是不是依然很简单？如果忘记了文件对话框类构造函数的参数意义，可以回到鸡啄米来看看或者在MSDN上查阅。

# VS2010/MFC编程入门之十八（对话框：字体对话框）

       鸡啄米在上一节为大家讲解了[文件对话框](http://www.jizhuomi.com/software/173.html)的使用，本节则主要介绍字体对话框如何应用。

       字体对话框的作用是用来选择字体。我们也经常能够见到。MFC使用CFontDialog类封装了字体对话框的所有操作。字体对话框也是一种[模态对话框](http://www.jizhuomi.com/software/160.html)。

**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。

**字体对话框应用实例**

       鸡啄米给大家做一个字体对话框的实例。先介绍此实例要实现的功能，生成一个对话框，对话框中放置一个“字体选择”按钮和一个编辑框。点击“字体选择”按钮将弹出字体对话框。编辑框用于显示所选字体名，并以选定的字体来显示字体名字符串，例如，如果选择了宋体，则在编辑框中以宋体显示字符串“宋体”。

       以下是创建此实例的步骤：

       1.创建一个基于对话框的MFC工程，名字为“Example18”。

       2.在自动生成的主对话框IDD\_EXAMPLE18\_DIALOG的模板中，删除“TODO: Place dialog controls here.”静态文本框，添加一个按钮，ID设为IDC\_FONT\_BUTTON，Caption设为“字体选择”，用于显示字体对话框来选择字体，再添加一个编辑框，ID设为IDC\_FONT\_EDIT，用来以所选字体显示字体名字符串。

       3.在Example18Dlg.h中为CExample18Dlg类添加private成员变量：CFont m\_font;，用来保存编辑框中选择的字体。

       4.为按钮IDC\_FONT\_BUTTON添加点击消息的消息处理函数CExample18Dlg::OnBnClickedFontButton()。

       5.修改[消息处理函数](http://www.jizhuomi.com/software/156.html)CExample18Dlg::OnBnClickedFontButton()如下：

**C++代码**

1. void CExample18Dlg::OnBnClickedFontButton()
2. {
3. // TODO: Add your control notification handler code here
4. CString strFontName;    // 字体名称
5. LOGFONT lf;             // LOGFONT变量
7. // 将lf所有字节清零
8. memset(&lf, 0, sizeof(LOGFONT));
10. // 将lf中的元素字体名设为“宋体”
11. \_tcscpy\_s(lf.lfFaceName, LF\_FACESIZE, \_T("宋体"));
13. // 构造字体对话框，初始选择字体名为“宋体”
14. CFontDialog fontDlg(&lf);
16. if (IDOK == fontDlg.DoModal())     // 显示字体对话框
17. {
18. // 如果m\_font已经关联了一个字体资源对象，则释放它
19. if (m\_font.m\_hObject)
20. {
21. m\_font.DeleteObject();
22. }
23. // 使用选定字体的LOGFONT创建新的字体
24. m\_font.CreateFontIndirect(fontDlg.m\_cf.lpLogFont);
25. // 获取编辑框IDC\_FONT\_EDIT的CWnd指针，并设置其字体
26. GetDlgItem(IDC\_FONT\_EDIT)->SetFont(&m\_font);
28. // 如果用户选择了字体对话框的OK按钮，则获取被选择字体的名称并显示到编辑框里
29. strFontName = fontDlg.m\_cf.lpLogFont->lfFaceName;
30. SetDlgItemText(IDC\_FONT\_EDIT, strFontName);
31. }
32. }

       6.最后，编译运行程序。显示结果对话框，点击“字体选择”按钮，将弹出字体对话框，默认选择为“宋体”，我们改而选择“华文彩云”字体点“确定”，编辑框中会像如下显示：

       到此，我们又学会了字体对话框的使用，对于以后在界面开发中控制显示的字体很有帮助。有问题欢迎在鸡啄米留言。

# VS2010/MFC编程入门之十九（对话框：颜色对话框）

       鸡啄米在上一节中为大家讲解了[字体对话框](http://www.jizhuomi.com/software/175.html)的使用方法，熟悉了字体对话框，本节继续讲另一种通用对话框--颜色对话框。

       颜色对话框大家肯定也不陌生，我们可以打开它选择需要的颜色，简单说，它的作用就是用来选择颜色。MFC中提供了CColorDialog类封装了颜色对话框的所有操作，我们可以通过它显示颜色对话框，并获取颜色对话框中选择的颜色。颜色对话框跟字体对话框一样，也是一种[模态对话框](http://www.jizhuomi.com/software/160.html)。

**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)。

**颜色对话框应用实例**

       鸡啄米下面给大家做一个颜色对话框的小例子。此例要实现的功能简单介绍下：生成一个对话框，对话框中放置一个“颜色选择”按钮，四个静态文本框和四个编辑框。四个静态文本框分别显示Color：、R：、G：、B：，每个静态文本框后面跟一个编辑框，分别用来显示颜色对话框中选择的颜色值和所选颜色值的红色分量、绿色分量、蓝色分量。

       以下是实例创建的步骤：

       1.创建一个基于对话框的MFC工程，名字为“Example19”。

       2.在自动生成的主对话框IDD\_EXAMPLE19\_DIALOG的模板中，删除“TODO: Place dialog controls here.”静态文本框，添加一个按钮，ID设为IDC\_COLOR\_BUTTON，Caption设为“颜色选择”，用于显示颜色对话框来选择颜色。再添加四个静态文本框，ID分别为IDC\_COLOR\_STATIC、IDC\_R\_STATIC、IDC\_G\_STATIC、IDC\_B\_STATIC，Caption分别设为“Color：”、“R：”、“G：”、“B：”，然后每个静态文本框后添加一个编辑框，四个编辑框的ID分别为IDC\_COLOR\_EDIT、IDC\_R\_EDIT、IDC\_G\_EDIT、IDC\_B\_EDIT，分别用来显示颜色对话框中选择的颜色值和所选颜色值的红色分量、绿色分量、蓝色分量。

       3.为按钮IDC\_COLOR\_BUTTON添加点击消息的消息处理函数CExample19Dlg::OnBnClickedColorButton()。

       4.修改[消息处理函数](http://www.jizhuomi.com/software/156.html)CExample19Dlg::OnBnClickedColorButton()如下：

**C++代码**

1. void CExample19Dlg::OnBnClickedColorButton()
2. {
3. // TODO: Add your control notification handler code here
4. COLORREF color = RGB(255, 0, 0);      // 颜色对话框的初始颜色为红色
5. CColorDialog colorDlg(color);         // 构造颜色对话框，传入初始颜色值
7. if (IDOK == colorDlg.DoModal())       // 显示颜色对话框，并判断是否点击了“确定”
8. {
9. color = colorDlg.GetColor();      // 获取颜色对话框中选择的颜色值
10. SetDlgItemInt(IDC\_COLOR\_EDIT, color);         // 在Color编辑框中显示所选颜色值
11. SetDlgItemInt(IDC\_R\_EDIT, GetRValue(color));  // 在R编辑框中显示所选颜色的R分量值
12. SetDlgItemInt(IDC\_G\_EDIT, GetGValue(color));  // 在G编辑框中显示所选颜色的G分量值
13. SetDlgItemInt(IDC\_B\_EDIT, GetBValue(color));  // 在B编辑框中显示所选颜色的B分量值
14. }
15. }

       5.最后编译运行程序，在结果对话框中点击“颜色选择”按钮，弹出颜色对话框。初始状态下，选择框在红色上，我们选另一种颜色，此时的颜色对话框如下：

        点“确定”，主对话框上的四个编辑框中分别显示了选择的颜色值、R分量、G分量和B分量：

       我们在实际开发中，可以用获取到的颜色值来设置其他对象的颜色，使用还是很方便的。

# VS2010/MFC编程入门之二十（常用控件：静态文本框）

       上一节鸡啄米讲了[颜色对话框](http://www.jizhuomi.com/software/177.html)之后，关于对话框的使用和各种通用对话框的介绍就到此为止了。从本节开始鸡啄米将讲解各种常用控件的用法。常用控件主要包括：静态文本框、编辑框、单选按钮、复选框、分组框、列表框、组合框、图片控件、列表控件、树形控件和进度条控件等等。本节教程先来讲解静态文本框的使用。

**控件的通知消息**

       在将静态文本框的使用之前，先大概讲讲控件的通知消息。

       当控件有事件发生时，它会向父窗口发送通知消息。最常发生的事件就是鼠标单击了，此时控件会向父窗口发送BN\_CLICKED消息，实际上也就是给父窗口发送WM\_COMMAND消息，在wParam参数中包含有通知消息码（鼠标单击时的通知消息码就是BN\_CLICKED）和控件ID，lParam参数中包含了控件的句柄。在[MFC消息映射机制概述](http://www.jizhuomi.com/software/147.html)中，鸡啄米讲过，消息就是由三个部分组成：消息值、wParam参数和lParam参数。

       为控件通知消息添加消息映射和[消息处理函数](http://www.jizhuomi.com/software/156.html)的方法，之前不止一遍讲过了。现在再来具体说明下，控件的消息映射宏的格式大致是：

       ON\_通知消息码(nID, memberFun)

       nID参数是控件的ID，memberFun参数是消息处理函数名。例如，ON\_BN\_CLICKED(IDC\_BUTTON1, &CDlg::OnBnClickedButton1)。此消息映射宏应添加到BEGIN\_MESSAGE\_MAP和END\_MESSAGE\_MAP之间。

       消息处理函数声明的语法形式为：

       afx\_msg void memberFun();

**静态文本框的使用**

       在前面鸡啄米的举例中，大家应该也清楚了静态文本框的一般作用，就是用于显示文字说明。[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)提供了CStatic类，封装了对静态文本框的所有操作。

       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 |

       我们在对话框模板添加静态文本框时，可以在静态文本框的属性页中设置它的风格，很多都与上面的风格是对应的，例如，Simple属性就相当于SS\_SIMPLE风格。

       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**

       简单介绍下CStatic类的主要成员函数，下面是成员函数列表。

|  |  |
| --- | --- |
| 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。

# VS2010/MFC编程入门之二十一（常用控件：编辑框Edit Control）

       鸡啄米上一节讲了[静态文本框](http://www.jizhuomi.com/software/179.html)，本节要讲的编辑框（Edit Control）同样是一种很常用的控件，我们可以在编辑框中输入并编辑文本。在前面加法计算器的例子中已经演示了编辑框的基本应用。下面具体讲解编辑框的使用。

**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。

**CEdit类应用实例**

       下面鸡啄米为大家写一个简单的例子，来说明CEdit类的几个成员函数的使用方法。此例的功能是，首先在编辑框中显示一行正文，然后替换其中部分字符为另一个含有回车符的字符串，最终显示为两行正文。下面是简单的步骤介绍：

       1.创建基于对话框的MFC程序，名称为“Example21”。

       2.在自动生成的对话框模板IDD\_EXAMPLE21\_DIALOG中，删除静态文本框“TODO: Place dialog controls here.”，添加一个编辑框，ID设为IDC\_MULTI\_LINE\_EDIT，属性Multiline设置为true。

       3.为编辑框IDC\_MULTI\_LINE\_EDIT[添加CEdit类型的控件变量](http://www.jizhuomi.com/software/153.html)m\_editMultiLine。

       4.修改CExample21Dlg::OnInitDialog()函数为：

**C++代码**

1. BOOL CExample21Dlg::OnInitDialog()
2. {
3. CDialogEx::OnInitDialog();
5. // Add "About..." menu item to system menu.
7. // IDM\_ABOUTBOX must be in the system command range.
8. ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);
9. ASSERT(IDM\_ABOUTBOX < 0xF000);
11. CMenu\* pSysMenu = GetSystemMenu(FALSE);
12. if (pSysMenu != NULL)
13. {
14. BOOL bNameValid;
15. CString strAboutMenu;
16. bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);
17. ASSERT(bNameValid);
18. if (!strAboutMenu.IsEmpty())
19. {
20. pSysMenu->AppendMenu(MF\_SEPARATOR);
21. pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);
22. }
23. }
25. // Set the icon for this dialog.  The framework does this automatically
26. //  when the application's main window is not a dialog
27. SetIcon(m\_hIcon, TRUE);         // Set big icon
28. SetIcon(m\_hIcon, FALSE);        // Set small icon
30. // TODO: Add extra initialization here
31. m\_editMultiLine.SetWindowText(\_T("鸡啄米博客/software"));  // 设置编辑框正文为“鸡啄米博客.com”
32. m\_editMultiLine.SetSel(3, 5);                            // 选择起始索引为3，终止索引为5（不包括在选择范围内）的正文，即“博客”
33. m\_editMultiLine.ReplaceSel(\_T("\r\nwww.jizhuomi.com"));  // 将选择的“博客”替换为“\r\nwww.jizhuomi.com”
35. return TRUE;  // return TRUE  unless you set the focus to a control
36. }

      5.编译运行程序，结果对话框如下：

      关于编辑框的介绍就到这里了。CEdit类成员函数的更详细的讲解可以查阅MSDN。鸡啄米谢谢您的持续关注。

# VS2010/MFC编程入门之二十二（常用控件：按钮控件Button、Radio Button和Check Box）

       因为私人问题，鸡啄米暂停更新了几天，首先向关注鸡啄米动态的朋友说一声抱歉。

       言归正传，鸡啄米上一节中讲了[编辑框](http://www.jizhuomi.com/software/181.html)的用法，本节继续讲解常用控件--按钮控件的使用。

**按钮控件简介**

       按钮控件包括命令按钮（Button）、单选按钮（Radio Button）和复选框（Check Box）等。命令按钮就是我们前面多次提到的狭义的按钮控件，用来响应用户的鼠标单击操作，进行相应的处理，它可以显示文本也可以嵌入位图。单选按钮使用时，一般是多个组成一组，组中每个单选按钮的选中状态具有互斥关系，即同组的单选按钮只能有一个被选中。

       命令按钮是我们最熟悉也是最常用的一种按钮控件，而单选按钮和复选框都是一种比较特殊的按钮控件。单选按钮有选中和未选中两种状态，为选中状态时单选按钮中心会出现一个蓝点，以标识选中状态。一般的复选框也是有选中和未选中两种状态，选中时复选框内会增加一个“√”，而三态复选框（设置了BS\_3STATE风格）有选中、未选中和不确定三种状态，不确定状态时复选框内出现一个灰色“√”。

       按钮控件会向父窗口发送通知消息，最常用的通知消息莫过于BN\_CLICKED和BN\_DOUBLECLICKED了。用户在按钮上单击鼠标时会向父窗口发送BN\_CLICKED消息，双击鼠标时发送BN\_DOUBLECLICKED消息。

**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。

       关于按钮控件Button、Radio Button和Check Box的使用基础就介绍到此，下一节中鸡啄米将举实例为大家演示各种按钮控件的使用方法，希望大家能继续关注。

# VS2010/MFC编程入门之二十三（常用控件：按钮控件的编程实例）

       上一节VS2010/MFC[编程入门](http://www.jizhuomi.com/catalog.asp?tags=%E7%BC%96%E7%A8%8B%E5%85%A5%E9%97%A8)教程中鸡啄米讲了[按钮控件Button、Radio Button和Check Box](http://www.jizhuomi.com/software/182.html)的基本用法，本节就继续讲按钮控件的内容，通过一个实例让大家更清楚按钮控件在实际的[软件开发](http://www.jizhuomi.com/software/)中如何使用。

       因为Button控件在前面的例子中涉及到了，比较简单，本文就不作深入分析了，而是重点讲解单选按钮Radio Button、复选框Check Box的使用。

**按钮控件实例的功能**

       首先介绍此实例实现的功能。此实例用来根据网站类型选择网站，并将选择的网站的名称显示到编辑框中。网站类型有“门户”、“论坛”和“博客”三种，为单选按钮。网站有六个：[鸡啄米](http://www.jizhuomi.com)、新浪、天涯论坛、韩寒博客、网易和凤凰网论坛，均为复选框。

       当选中某种网站类型即点了某个单选按钮时，其对应的网站的复选框就激活，其他则禁用，不允许选择，且为非选中状态。例如，如果选中了“门户”单选按钮，则“新浪”、“网易”复选框激活，允许用户选择，而其他复选框则禁用。

**按钮控件实例的实现**

       鸡啄米下面为大家详细阐述此实例的编写步骤。

       1. 创建一个基于对话框的MFC工程，名称设为“Example23”。

       2. 在自动生成的主对话框IDD\_EXAMPLE23\_DIALOG的模板中，删除“TODO: Place dialog controls here.”[静态文本框](http://www.jizhuomi.com/software/179.html)，添加两个Group Box，属性Caption分别改为“网站类型”、“网站”。

       3. 在Group Box“网站类型”中加入三个Radio Button，Caption分别设为“门户”、“论坛”和“博客”，ID分别设为IDC\_PORTAL\_RADIO、IDC\_FORUM\_RADIO和IDC\_BLOG\_RADIO。

       4. 在Group Box“网站”中加入六个Check Box，Caption分别设为“鸡啄米”、“新浪”、“天涯论坛”、“韩寒博客”、“网易”和“凤凰网论坛”，ID分别设为IDC\_CHECK1、IDC\_CHECK2、IDC\_CHECK3、IDC\_CHECK4、IDC\_CHECK5和IDC\_CHECK6。然后为每个复选框添加CButton类型的变量m\_check1、m\_check2、m\_check3、m\_check4、m\_check5和m\_check6。

       5. 在两个Group Box下面，添加一个静态文本框和一个[编辑框](http://www.jizhuomi.com/software/181.html)。静态文本框的Caption设为“选择的网站：”。编辑框的ID设为IDC\_WEBSITE\_SEL\_EDIT，属性Read Only改为True，使此编辑框为只读状态，不允许用户编辑。

       6. 将“OK”按钮的Caption修改为“确定”，“Cancel”按钮的Caption修改为“退出”。到此，对话框模板就修改好了，如下图：

        7. 为“门户”、“论坛”和“博客”三个单选按钮分别添加点击消息的[消息处理函数](http://www.jizhuomi.com/software/156.html)CExample23Dlg::OnBnClickedPortalRadio()、CExample23Dlg::OnBnClickedForumRadio()和CExample23Dlg::OnBnClickedBlogRadio()。

        在某个单选按钮被点击之后，我们可以先将六个网站复选框都禁用且置为非选中状态，而后将选择的网站类型对应的网站复选框激活。为了代码复用，我们将置所有复选框为禁用且非选中状态的操作写到一个函数里，此函数为CExample23Dlg::InitAllCheckBoxStatus()，然后就可以在三个单选按钮的消息处理函数中调用InitAllCheckBoxStatus()，实现复选框状态的初始化。

       三个消息处理函数及InitAllCheckBoxStatus()函数的实现如下：

**C++代码**

1. void CExample23Dlg::OnBnClickedPortalRadio()
2. {
3. // TODO: Add your control notification handler code here
4. // 如果选择了“门户”单选按钮，则激活复选框“新浪”和“网易”，其他复选框禁用并非选中
5. InitAllCheckBoxStatus();
6. m\_check2.EnableWindow(TRUE);
7. m\_check5.EnableWindow(TRUE);
8. }

11. void CExample23Dlg::OnBnClickedForumRadio()
12. {
13. // TODO: Add your control notification handler code here
14. // 如果选择了“论坛”单选按钮，则激活复选框“天涯论坛”和“凤凰网论坛”，其他复选框禁用并非选中
15. InitAllCheckBoxStatus();
16. m\_check3.EnableWindow(TRUE);
17. m\_check6.EnableWindow(TRUE);
18. }

21. void CExample23Dlg::OnBnClickedBlogRadio()
22. {
23. // TODO: Add your control notification handler code here
24. // 如果选择了“博客”单选按钮，则激活复选框“鸡啄米”和“韩寒博客”，其他复选框禁用并非选中
25. InitAllCheckBoxStatus();
26. m\_check1.EnableWindow(TRUE);
27. m\_check4.EnableWindow(TRUE);
28. }
30. // 初始化所有复选框的状态，即全部禁用，全部非选中
31. void CExample23Dlg::InitAllCheckBoxStatus()
32. {
33. // 全部禁用
34. m\_check1.EnableWindow(FALSE);
35. m\_check2.EnableWindow(FALSE);
36. m\_check3.EnableWindow(FALSE);
37. m\_check4.EnableWindow(FALSE);
38. m\_check5.EnableWindow(FALSE);
39. m\_check6.EnableWindow(FALSE);
41. // 全部非选中
42. m\_check1.SetCheck(0);
43. m\_check2.SetCheck(0);
44. m\_check3.SetCheck(0);
45. m\_check4.SetCheck(0);
46. m\_check5.SetCheck(0);
47. m\_check6.SetCheck(0);
48. }

       8. 程序运行后，我们希望网站类型默认选择为“门户”，则修改对话框初始化函数CExample23Dlg::OnInitDialog()为：

**C++代码**

1. BOOL CExample23Dlg::OnInitDialog()
2. {
3. CDialogEx::OnInitDialog();
5. // Add "About..." menu item to system menu.
7. // IDM\_ABOUTBOX must be in the system command range.
8. ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);
9. ASSERT(IDM\_ABOUTBOX < 0xF000);
11. CMenu\* pSysMenu = GetSystemMenu(FALSE);
12. if (pSysMenu != NULL)
13. {
14. BOOL bNameValid;
15. CString strAboutMenu;
16. bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);
17. ASSERT(bNameValid);
18. if (!strAboutMenu.IsEmpty())
19. {
20. pSysMenu->AppendMenu(MF\_SEPARATOR);
21. pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);
22. }
23. }
25. // Set the icon for this dialog.  The framework does this automatically
26. //  when the application's main window is not a dialog
27. SetIcon(m\_hIcon, TRUE);         // Set big icon
28. SetIcon(m\_hIcon, FALSE);        // Set small icon
30. // TODO: Add extra initialization here
31. // 默认选中“门户”单选按钮
32. CheckDlgButton(IDC\_PORTAL\_RADIO, 1);
33. OnBnClickedPortalRadio();
35. return TRUE;  // return TRUE  unless you set the focus to a control
36. }

       9. 点击“确定”后，将选择的网站名字显示到编辑框中，那么需要修改“确定”按钮（原来的OK按钮）的消息处理函数CExample23Dlg::OnBnClickedOk()如下：

**C++代码**

1. void CExample23Dlg::OnBnClickedOk()
2. {
3. // TODO: Add your control notification handler code here
4. CString strWebsiteSel;         // 选择的网站
6. // 若选中“鸡啄米”则将其加入结果字符串
7. if (1 == m\_check1.GetCheck())
8. {
9. strWebsiteSel += \_T("鸡啄米 ");
10. }
11. // 若选中“新浪”则将其加入结果字符串
12. if (1 == m\_check2.GetCheck())
13. {
14. strWebsiteSel += \_T("新浪 ");
15. }
16. // 若选中“天涯论坛”则将其加入结果字符串
17. if (1 == m\_check3.GetCheck())
18. {
19. strWebsiteSel += \_T("天涯论坛 ");
20. }
21. // 若选中“韩寒博客”则将其加入结果字符串
22. if (1 == m\_check4.GetCheck())
23. {
24. strWebsiteSel += \_T("韩寒博客 ");
25. }
26. // 若选中“网易”则将其加入结果字符串
27. if (1 == m\_check5.GetCheck())
28. {
29. strWebsiteSel += \_T("网易 ");
30. }
31. // 若选中“凤凰网论坛”则将其加入结果字符串
32. if (1 == m\_check6.GetCheck())
33. {
34. strWebsiteSel += \_T("凤凰网论坛 ");
35. }
37. // 将结果字符串显示于“选择的网站”后的编辑框中
38. SetDlgItemText(IDC\_WEBSITE\_SEL\_EDIT, strWebsiteSel);
40. // 为了避免点“确定”后对话框退出，将OnOk注掉
41. //CDialogEx::OnOK();
42. }

       10. 到此程序编写完成。运行程序弹出结果对话框，选择网站后界面如下图：

       按钮控件的内容就这些了。掌握了按钮控件的基本用法，又动手编写了这个实例后，相信大家对按钮控件已经很熟悉了。

# VS2010/MFC编程入门之二十四（常用控件：列表框控件ListBox）

       前面两节讲了比较[常用的按钮控件](http://www.jizhuomi.com/software/182.html)，并通过[按钮控件实例](http://www.jizhuomi.com/software/184.html)说明了具体用法。本文要讲的是列表框控件（ListBox）及其使用实例。

**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。

**CListBox类应用实例**

最后鸡啄米给大家写一个简单的实例，说明CListBox的几个成员函数及[通知消息](http://www.jizhuomi.com/software/147.html)等的使用方法。此实例实现的功能：在单选列表框中显示一个网站列表，然后在用鼠标左键选择某列表项时，将选中列表项的文本显示到编辑框中。下面是具体实现步骤：

       1. 创建一个基于对话框的MFC工程，名称设置为“Example24”。

       2. 在自动生成的对话框模板IDD\_EXAMPLE24\_DIALOG中，删除“TODO: Place dialog controls here.”静态文本控件、“OK”按钮和“Cancel”按钮。添加一个Listbox控件，ID设置为IDC\_WEB\_LIST，Sort属性设为False，以取消排序显示。再添加一个静态文本控件和一个编辑框，静态文本控件的Caption属性设为“您选择的站点：”，编辑框的ID设为IDC\_SEL\_WEB\_EDIT，Read Only属性设为True。此时的对话框模板如下图：

       3. 为列表框IDC\_WEB\_LIST[添加CListBox类型的控件变量](http://www.jizhuomi.com/software/153.html)m\_listBox。

       4. 在对话框初始化时，我们将站点名加入到列表框中，那么需要修改CExample24Dlg::OnInitDialog()函数为：

**C++代码**

1. BOOL CExample24Dlg::OnInitDialog()
2. {
3. CDialogEx::OnInitDialog();
5. // Add "About..." menu item to system menu.
7. // IDM\_ABOUTBOX must be in the system command range.
8. ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);
9. ASSERT(IDM\_ABOUTBOX < 0xF000);
11. CMenu\* pSysMenu = GetSystemMenu(FALSE);
12. if (pSysMenu != NULL)
13. {
14. BOOL bNameValid;
15. CString strAboutMenu;
16. bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);
17. ASSERT(bNameValid);
18. if (!strAboutMenu.IsEmpty())
19. {
20. pSysMenu->AppendMenu(MF\_SEPARATOR);
21. pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);
22. }
23. }
25. // Set the icon for this dialog.  The framework does this automatically
26. //  when the application's main window is not a dialog
27. SetIcon(m\_hIcon, TRUE);         // Set big icon
28. SetIcon(m\_hIcon, FALSE);        // Set small icon
30. // TODO: Add extra initialization here
31. m\_listBox.AddString(\_T("新浪"));        // 在列表框结尾添加字符串“新浪”
32. m\_listBox.AddString(\_T("鸡啄米"));      // 在列表框结尾添加字符串“鸡啄米”
33. m\_listBox.AddString(\_T("猫扑"));        // 在列表框结尾添加字符串“猫扑”
34. m\_listBox.InsertString(2, \_T("百度"));  // 在列表框中索引为2的位置插入字符串“百度”
36. return TRUE;  // return TRUE  unless you set the focus to a control
37. }

       5. 我们希望在选中列表项改变时，将最新的选择项实时显示到编辑框中，那么这就要用到LBN\_SELCHANGE通知消息。为列表框IDC\_WEB\_LIST的通知消息LBN\_SELCHANGE[添加消息处理函数](http://www.jizhuomi.com/software/156.html)CExample24Dlg::OnLbnSelchangeWebList()，并修改如下：

**C++代码**

1. void CExample24Dlg::OnLbnSelchangeWebList()
2. {
3. // TODO: Add your control notification handler code here
4. CString strText;
5. int nCurSel;
7. nCurSel = m\_listBox.GetCurSel();           // 获取当前选中列表项
8. m\_listBox.GetText(nCurSel, strText);       // 获取选中列表项的字符串
9. SetDlgItemText(IDC\_SEL\_WEB\_EDIT, strText); // 将选中列表项的字符串显示到编辑框中
10. }

       6. 运行程序，弹出结果对话框，在对话框的列表框中用鼠标改变选中项时，编辑框中的显示会相应改变。效果图如下：

       关于列表框ListBox的讲解就到此为止了。大家如果想试验更多的列表框成员函数，可以在上面的小例子中加入更多的功能来体会

# VS2010/MFC编程入门之二十五（常用控件：组合框控件Combo Box）

       上一节鸡啄米讲了[列表框控件ListBox](http://www.jizhuomi.com/software/186.html)的使用，本节主要讲解组合框控件Combo Box。组合框同样相当常见，例如，在Windows系统的控制面板上设置语言或位置时，有很多选项，用来进行选择的控件就是组合框控件。它为我们的日常操作提供了很多方便。

**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，列表框弹出后如下图：

       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。下拉列表式组合框如下图：

       经过上面的介绍，大家应该知道，最常用的当属下拉式组合框和下拉列表式组合框了，它们在很多时候能使程序看起来更专业，更简洁，让用户在进行选择操作时更方便。

       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。

**CComboBox类应用实例**

       最后鸡啄米给大家写一个简单的实例，说明CComboBox的几个成员函数及通知消息等的使用方法。此实例实现的功能：在组合框中包含一个网站列表，切换组合框控件的列表框中选择的列表项时，将新选中的列表项的文本显示到编辑框中。下面是具体实现步骤：

       1. 创建一个基于对话框的MFC工程，名称设置为“Example25”。

       2. 在自动生成的对话框模板IDD\_EXAMPLE25\_DIALOG中，删除“TODO: Place dialog controls here.”静态文本控件、“OK”按钮和“Cancel”按钮。添加一个Combo Box控件，ID设置为IDC\_WEB\_COMBO，Type属性设为Drop List，为下拉列表式组合框，编辑框不允许用户输入，Sort属性设为False，以取消排序显示。再添加一个静态文本控件和一个编辑框，静态文本控件的Caption属性设为“您选择的网站：”，编辑框的ID设为IDC\_SEL\_WEB\_EDIT，Read Only属性设为True。此时的对话框模板如下图：

       3. 为组合框IDC\_WEB\_COMBO[添加CComboBox类型的控件变量](http://www.jizhuomi.com/software/153.html)m\_comboWeb。

       4. 在对话框初始化时，我们将站点名加入到组合框中，并默认选择第一项，那么需要修改CExample25Dlg::OnInitDialog()函数为：

**C++代码**

1. BOOL CExample25Dlg::OnInitDialog()
2. {
3. CDialogEx::OnInitDialog();
5. // Add "About..." menu item to system menu.
7. // IDM\_ABOUTBOX must be in the system command range.
8. ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);
9. ASSERT(IDM\_ABOUTBOX < 0xF000);
11. CMenu\* pSysMenu = GetSystemMenu(FALSE);
12. if (pSysMenu != NULL)
13. {
14. BOOL bNameValid;
15. CString strAboutMenu;
16. bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);
17. ASSERT(bNameValid);
18. if (!strAboutMenu.IsEmpty())
19. {
20. pSysMenu->AppendMenu(MF\_SEPARATOR);
21. pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);
22. }
23. }
25. // Set the icon for this dialog.  The framework does this automatically
26. //  when the application's main window is not a dialog
27. SetIcon(m\_hIcon, TRUE);         // Set big icon
28. SetIcon(m\_hIcon, FALSE);        // Set small icon
30. // TODO: Add extra initialization here
31. // 为组合框控件的列表框添加列表项“鸡啄米”
32. m\_comboWeb.AddString(\_T("鸡啄米"));
33. // 为组合框控件的列表框添加列表项“百度”
34. m\_comboWeb.AddString(\_T("百度"));
35. // 在组合框控件的列表框中索引为1的位置插入列表项“新浪”
36. m\_comboWeb.InsertString(1, \_T("新浪"));
38. // 默认选择第一项
39. m\_comboWeb.SetCurSel(0);
40. // 编辑框中默认显示第一项的文字“鸡啄米”
41. SetDlgItemText(IDC\_SEL\_WEB\_EDIT, \_T("鸡啄米"));
43. return TRUE;  // return TRUE  unless you set the focus to a control
44. }

       5. 我们希望在组合框中选中的列表项改变时，将最新的选择项实时显示到编辑框中，那么这就要用到CBN\_SELCHANGE通知消息。为列表框IDC\_WEB\_COMBO的通知消息CBN\_SELCHANGE添加[消息处理函数](http://www.jizhuomi.com/software/156.html)CExample25Dlg::OnCbnSelchangeWebCombo()，并修改如下：

**C++代码**

1. void CExample25Dlg::OnCbnSelchangeWebCombo()
2. {
3. // TODO: Add your control notification handler code here
4. CString strWeb;
5. int nSel;
7. // 获取组合框控件的列表框中选中项的索引
8. nSel = m\_comboWeb.GetCurSel();
9. // 根据选中项索引获取该项字符串
10. m\_comboWeb.GetLBText(nSel, strWeb);
11. // 将组合框中选中的字符串显示到IDC\_SEL\_WEB\_EDIT编辑框中
12. SetDlgItemText(IDC\_SEL\_WEB\_EDIT, strWeb);
13. }

       6. 运行程序，弹出结果对话框，在对话框的组合框中改变选择项时，编辑框中的显示会相应改变。效果图如下：

       组合框的内容就是这些了。相对于CComboBox类数量不少的成员函数来说，本节的实例只是用到了很少的几个，大家可以根据上面所讲试试其他的成员函数

# VS2010/MFC编程入门之二十六（常用控件：滚动条控件Scroll Bar）

       回顾上一节，鸡啄米讲的是[组合框控件Combo Box](http://www.jizhuomi.com/software/189.html)的使用。本节详解滚动条控件Scroll Bar的相关内容。

**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：

**C++代码**

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

**CScrollBar类应用实例**

       讲完了基础知识，鸡啄米还是给大家一个简单的实例。例子非常简单，就是在一个对话框中加入一个水平滚动条控件和一个[编辑框](http://www.jizhuomi.com/software/181.html)控件，无论滚动条控件是在滚动还是静止，编辑框中都显示滚动块的当前位置。以下是具体开发步骤：

       1. 创建一个基于对话框的MFC工程，名称设置为“Example26”。

       2. 在自动生成的对话框模板IDD\_EXAMPLE26\_DIALOG中，删除“TODO: Place dialog controls here.”静态文本控件、“OK”按钮和“Cancel”按钮。添加一个Horizontal Scroll Bar控件，ID设置为IDC\_HORI\_SCROLLBAR。再添加一个静态文本控件和一个编辑框，静态文本控件的Caption属性设为“滚动块当前位置：”，编辑框的ID设为IDC\_HSCROLL\_EDIT，Read Only属性设为True。此时的对话框模板如下图：

       3. 为滚动条IDC\_HORI\_SCROLLBAR添加CScrollBar类型的控件变量m\_horiScrollbar。

       4. 在对话框初始化时，我们需要设置滚动条的滚动范围和初始位置，并在编辑框中显示初始位置，那么需要修改CExample26Dlg::OnInitDialog()函数为：

**C++代码**

1. BOOL CExample26Dlg::OnInitDialog()
2. {
3. CDialogEx::OnInitDialog();
5. // Add "About..." menu item to system menu.
7. // IDM\_ABOUTBOX must be in the system command range.
8. ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);
9. ASSERT(IDM\_ABOUTBOX < 0xF000);
11. CMenu\* pSysMenu = GetSystemMenu(FALSE);
12. if (pSysMenu != NULL)
13. {
14. BOOL bNameValid;
15. CString strAboutMenu;
16. bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);
17. ASSERT(bNameValid);
18. if (!strAboutMenu.IsEmpty())
19. {
20. pSysMenu->AppendMenu(MF\_SEPARATOR);
21. pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);
22. }
23. }
25. // Set the icon for this dialog.  The framework does this automatically
26. //  when the application's main window is not a dialog
27. SetIcon(m\_hIcon, TRUE);         // Set big icon
28. SetIcon(m\_hIcon, FALSE);        // Set small icon
30. // TODO: Add extra initialization here
31. // 设置水平滚动条的滚动范围为1到100
32. m\_horiScrollbar.SetScrollRange(1, 100);
33. // 设置水平滚动条的初始位置为20
34. m\_horiScrollbar.SetScrollPos(20);
35. // 在编辑框中显示20
36. SetDlgItemInt(IDC\_HSCROLL\_EDIT, 20);
38. return TRUE;  // return TRUE  unless you set the focus to a control
39. }

       5. 现在滚动条还不能正常滚动，并且编辑框中数字也不随滚动改变。根据上面所讲，我们可以重载CExample26Dlg类的OnHScroll函数。具体操作为，在CExample26Dlg类的属性页面的工具栏上点“Messages”按钮，找到WM\_HSCROLL消息，添加响应函数就可以了。OnHScroll函数重写后如下：

**C++代码**

1. void CExample26Dlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar\* pScrollBar)
2. {
3. // TODO: Add your message handler code here and/or call default
4. int pos = m\_horiScrollbar.GetScrollPos();    // 获取水平滚动条当前位置
6. switch (nSBCode)
7. {
8. // 如果向左滚动一列，则pos减1
9. case SB\_LINEUP:
10. pos -= 1;
11. break;
12. // 如果向右滚动一列，则pos加1
13. case SB\_LINEDOWN:
14. pos  += 1;
15. break;
16. // 如果向左滚动一页，则pos减10
17. case SB\_PAGEUP:
18. pos -= 10;
19. break;
20. // 如果向右滚动一页，则pos加10
21. case SB\_PAGEDOWN:
22. pos  += 10;
23. break;
24. // 如果滚动到最左端，则pos为1
25. case SB\_TOP:
26. pos = 1;
27. break;
28. // 如果滚动到最右端，则pos为100
29. case SB\_BOTTOM:
30. pos = 100;
31. break;
32. // 如果拖动滚动块滚动到指定位置，则pos赋值为nPos的值
33. case SB\_THUMBPOSITION:
34. pos = nPos;
35. break;
36. // 下面的m\_horiScrollbar.SetScrollPos(pos);执行时会第二次进入此函数，最终确定滚动块位置，并且会直接到default分支，所以在此处设置编辑框中显示数值
37. default:
38. SetDlgItemInt(IDC\_HSCROLL\_EDIT, pos);
39. return;
40. }
42. // 设置滚动块位置
43. m\_horiScrollbar.SetScrollPos(pos);
45. CDialogEx::OnHScroll(nSBCode, nPos, pScrollBar);
46. }

       6. 编译运行程序，弹出结果对话框，可以自己拖动滚动块看是否能正常滚动，并且编辑框中也显示了正确的数值。效果如下：

        至于垂直滚动条，其实与水平滚动条类似，大家可以自己写写垂直滚动条的例子，鸡啄米就不再举例了。

# VS2010/MFC编程入门之二十七（常用控件：图片控件Picture Control）

       上一节中鸡啄米讲的是[滚动条控件](http://www.jizhuomi.com/software/191.html)，本节主要讲一种简单实用的控件，图片控件Picture Control。我们可以在界面某个位置放入图片控件，显示图片以美化界面。

**图片控件简介**

       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。

**图片控件静态和动态加载图片**

       鸡啄米下面为大家演示如何为图片控件静态和动态加载位图图片。

**1. 图片控件静态加载图片**

       1）创建一个基于对话框的[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)工程，名称设置为“Example27”。

       2）准备一张Bitmap图片，名称设为“test.bmp”，放到工程的res文件夹中，res文件夹路径为...\Example27\Example27\res。鸡啄米在这里用的是一张鸡啄米网站的截图。

       3）在Resource View中的“Example27.rc\*”节点上点右键，选择“Add Resource...”，弹出“Add Resource”对话框：

       然后在左侧的“Resource Type”中选择“Bitmap”，点按钮“Import”，显示一个文件对话框，我们选择res文件夹中的test.bmp图片文件，导入成功后会在Resource View的Example27.rc\*节点下出现一个新的子节点“Bitmap”，而在“Bitmap”节点下可以看到刚添加的位图资源IDB\_BITMAP1，这里的默认ID就不修改了。

       4.）在自动生成的对话框模板IDD\_EXAMPLE27\_DIALOG中，删除“TODO: Place dialog controls here.”静态文本控件、“OK”按钮和“Cancel”按钮。添加一个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

       因为我们要加载的是位图图片，所以Type属性选择Bitmap。

       5）在图片控件的Image属性的下拉列表中选择3）中导入的位图IDB\_BITMAP1。

       6）编译运行程序，弹出结果对话框，如下图所示：

**2. 图片控件动态加载图片**

        以上讲的是静态加载图片的方法，下面接着讲动态加载图片的方法。程序依然沿用上面的工程。步骤如下：

        1）将上面添加的图片控件的Image属性IDB\_BITMAP1清空，Type属性不变。

        2）修改图片控件的ID为IDC\_JIZHUOMI\_STATIC，然后为其添加CStatic类型控件变量m\_jzmPicture。（若不修改ID则无法为其[添加控件变量](http://www.jizhuomi.com/software/153.html)）

        3）在对话框下方添加一按钮控件，Caption属性改为“加载图片”，ID设为IDC\_LOAD\_PIC\_BUTTON。

        4）为按钮IDC\_LOAD\_PIC\_BUTTON[添加点击消息的处理函数](http://www.jizhuomi.com/software/153.html)CExample27Dlg::OnBnClickedLoadPicButton()，然后修改此函数的函数实现如下：

**C++代码**

1. void CExample27Dlg::OnBnClickedLoadPicButton()
2. {
3. // TODO: Add your control notification handler code here
4. CBitmap bitmap;  // CBitmap对象，用于加载位图
5. HBITMAP hBmp;    // 保存CBitmap加载的位图的句柄
7. bitmap.LoadBitmap(IDB\_BITMAP1);  // 将位图IDB\_BITMAP1加载到bitmap
8. hBmp = (HBITMAP)bitmap.GetSafeHandle();  // 获取bitmap加载位图的句柄
9. m\_jzmPicture.SetBitmap(hBmp);    // 设置图片控件m\_jzmPicture的位图图片为IDB\_BITMAP1
10. }

        5）编译运行程序，弹出结果对话框，点击按钮“加载图片”，结果如下：

       图片控件Picture Control的内容就讲到这里了。

# VS2010/MFC编程入门之二十八（常用控件：列表视图控件List Control 上）

       前面一节中，鸡啄米讲了[图片控件Picture Control](http://www.jizhuomi.com/software/193.html)，本节为大家详解列表视图控件List Control的使用。

**列表视图控件简介**

       列表视图控件List Control同样比较常见，它能够把任何字符串内容以列表的方式显示出来，这种显示方式的特点是整洁、直观，在实际应用中能为用户带来方便。

       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**

       鸡啄米在[VS2010/MFC编程入门之五（MFC消息映射机制概述）](http://www.jizhuomi.com/software/147.html)中的“各种Windows消息的消息处理函数”部分，就曾以列表视图控件为例简单讲了WM\_NOTIFY通知消息及其消息映射入口和消息处理函数的形式。如果你忘记了可以回到第五节看一看，回忆一下。

       鸡啄米这里给出下一节中将要演示的列表视图控件实例中，通知码为NM\_CLICK的通知消息的消息映射入口：

       ON\_NOTIFY(NM\_CLICK, IDC\_PROGRAM\_LANG\_LIST, &CExample29Dlg::OnNMClickProgramLangList)

       还有[消息处理函数](http://www.jizhuomi.com/software/156.html)自动生成时的形式：

**C++代码**

1. void CExample29Dlg::OnNMClickProgramLangList(NMHDR \*pNMHDR, LRESULT \*pResult)
2. {
3. LPNMITEMACTIVATE pNMItemActivate = reinterpret\_cast<LPNMITEMACTIVATE>(pNMHDR);
4. // TODO: Add your control notification handler code here
5. \*pResult = 0;
6. }

       我们看到，上面的代码中将NMHDR指针类型的pNMHDR强制转换为LPNMITEMACTIVATE类型的pNMItemActivate，那么我们就可以在函数中通过pNMHDR来访问NMHDR结构，也可以通过pNMItemActive指针变量来访问第一个元素为NMHDR结构体变量的扩充结构。

       当然列表视图控件还有一些自己特有的通知消息，下面就介绍几个其中比较常用的。

       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**

**C++代码**

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：

**C++代码**

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。

**C++代码**

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。

**C++代码**

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。

**C++代码**

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;

       有关列表视图控件的内容本节先讲这些，下节继续讲列表控件类CListCtrl的一些成员函数和应用实例

# VS2010/MFC编程入门之二十九（常用控件：列表视图控件List Control 下）

       上一节是关于[列表视图控件List Control的上半部分](http://www.jizhuomi.com/software/195.html)，简单介绍了列表视图控件，其通知消息的处理和有关结构体的定义。本节继续讲解下半部分，包括列表视图控件的创建、CListCtrl类的主要成员函数和CListCtrl类应用实例。

**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。

**CListCtrl类应用实例**

       最后鸡啄米还是给大家写一个简单的实例，说明CListCtrl类的几个成员函数及通知消息等的使用方法。因为在开发中最常用的要属报表风格的List Control了，所以鸡啄米给大家写的是一个报表List Control的例子。

       此实例实现的功能：在单选列表视图控件中显示一个简单的编程语言排行榜，然后在用鼠标左键选择某列表项时，将选中列表项的文本显示到编辑框中。下面是具体实现步骤：

       1. 创建一个基于对话框的MFC工程，名称设置为“Example29”。

       2. 在自动生成的对话框模板IDD\_EXAMPLE29\_DIALOG中，删除“TODO: Place dialog controls here.”静态文本控件、“OK”按钮和“Cancel”按钮。添加一个List Control控件，ID设置为IDC\_PROGRAM\_LANG\_LIST，View属性设为Report，即为报表风格，Single Selection属性设为True。再添加一个静态文本控件和一个编辑框，静态文本控件的Caption属性设为“选择的语言：”，编辑框的ID设为IDC\_LANG\_SEL\_EDIT，Read Only属性设为True。此时的对话框模板如下图：

       3. 为列表视图控件IDC\_PROGRAM\_LANG\_LIST[添加CListCtrl类型的控件变量](http://www.jizhuomi.com/software/153.html)m\_programLangList。

       4. 在对话框初始化时，我们将编程语言排行榜加入到列表视图控件中，那么需要修改CExample29Dlg::OnInitDialog()函数为：

**C++代码**

1. BOOL CExample29Dlg::OnInitDialog()
2. {
3. CDialogEx::OnInitDialog();
5. // Add "About..." menu item to system menu.
7. // IDM\_ABOUTBOX must be in the system command range.
8. ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);
9. ASSERT(IDM\_ABOUTBOX < 0xF000);
11. CMenu\* pSysMenu = GetSystemMenu(FALSE);
12. if (pSysMenu != NULL)
13. {
14. BOOL bNameValid;
15. CString strAboutMenu;
16. bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);
17. ASSERT(bNameValid);
18. if (!strAboutMenu.IsEmpty())
19. {
20. pSysMenu->AppendMenu(MF\_SEPARATOR);
21. pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);
22. }
23. }
25. // Set the icon for this dialog.  The framework does this automatically
26. //  when the application's main window is not a dialog
27. SetIcon(m\_hIcon, TRUE);         // Set big icon
28. SetIcon(m\_hIcon, FALSE);        // Set small icon
30. // TODO: Add extra initialization here
31. CRect rect;
33. // 获取编程语言列表视图控件的位置和大小
34. m\_programLangList.GetClientRect(&rect);
36. // 为列表视图控件添加全行选中和栅格风格
37. m\_programLangList.SetExtendedStyle(m\_programLangList.GetExtendedStyle() | LVS\_EX\_FULLROWSELECT | LVS\_EX\_GRIDLINES);
39. // 为列表视图控件添加三列
40. m\_programLangList.InsertColumn(0, \_T("语言"), LVCFMT\_CENTER, rect.Width()/3, 0);
41. m\_programLangList.InsertColumn(1, \_T("2012.02排名"), LVCFMT\_CENTER, rect.Width()/3, 1);
42. m\_programLangList.InsertColumn(2, \_T("2011.02排名"), LVCFMT\_CENTER, rect.Width()/3, 2);
44. // 在列表视图控件中插入列表项，并设置列表子项文本
45. m\_programLangList.InsertItem(0, \_T("Java"));
46. m\_programLangList.SetItemText(0, 1, \_T("1"));
47. m\_programLangList.SetItemText(0, 2, \_T("1"));
48. m\_programLangList.InsertItem(1, \_T("C"));
49. m\_programLangList.SetItemText(1, 1, \_T("2"));
50. m\_programLangList.SetItemText(1, 2, \_T("2"));
51. m\_programLangList.InsertItem(2, \_T("C#"));
52. m\_programLangList.SetItemText(2, 1, \_T("3"));
53. m\_programLangList.SetItemText(2, 2, \_T("6"));
54. m\_programLangList.InsertItem(3, \_T("C++"));
55. m\_programLangList.SetItemText(3, 1, \_T("4"));
56. m\_programLangList.SetItemText(3, 2, \_T("3"));
58. return TRUE;  // return TRUE  unless you set the focus to a control
59. }

       5. 我们希望在选中列表项改变时，将最新的选择项实时显示到编辑框中，那么可以使用NM\_CLICK通知消息。为列表框IDC\_PROGRAM\_LANG\_LIST的通知消息NM\_CLICK[添加消息处理函数](http://www.jizhuomi.com/software/156.html)CExample29Dlg::OnNMClickProgramLangList，并修改如下：

**C++代码**

1. void CExample29Dlg::OnNMClickProgramLangList(NMHDR \*pNMHDR, LRESULT \*pResult)
2. {
3. LPNMITEMACTIVATE pNMItemActivate = reinterpret\_cast<LPNMITEMACTIVATE>(pNMHDR);
4. // TODO: Add your control notification handler code here
5. \*pResult = 0;
7. CString strLangName;    // 选择语言的名称字符串
8. NMLISTVIEW \*pNMListView = (NMLISTVIEW\*)pNMHDR;
10. if (-1 != pNMListView->iItem)        // 如果iItem不是-1，就说明有列表项被选择
11. {
12. // 获取被选择列表项第一个子项的文本
13. strLangName = m\_programLangList.GetItemText(pNMListView->iItem, 0);
14. // 将选择的语言显示与编辑框中
15. SetDlgItemText(IDC\_LANG\_SEL\_EDIT, strLangName);
16. }
17. }

       6. 运行程序，弹出结果对话框，在对话框的列表框中用鼠标改变选中项时，编辑框中的显示会相应改变。效果图如下：

       关于列表视图控件List Control的内容总算讲完了，内容不少，但实际上这些还只是一部分，在实际开发中会遇到各种问题，需要大家去查阅MSDN或上网找资料等来解决。

# VS2010/MFC编程入门之三十（常用控件：树形控件Tree Control 上）

       前面两节为大家讲了[列表视图控件List Control](http://www.jizhuomi.com/software/195.html)，这一节开始介绍一种特殊的列表--树形控件Tree Control。

**树形控件简介**

       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。

       树形控件在插入新的树节点时会稍麻烦些，回顾之前的[列表框](http://www.jizhuomi.com/software/186.html)，插入新列表项时调用AddString成员函数就可以了，而对于树形控件则需要指定新节点与已有节点的关系。另外，树形控件与列表视图控件一样，可以在每一个节点的左边加入图标。这些都使得树形控件给人一种复杂的感觉，但我们在使用它一两次后会发现其实树形控件用起来还是很方便的。

**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：

**C++代码**

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：

**C++代码**

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：

**C++代码**

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：

**C++代码**

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

       关于树形控件的使用本节先讲这么多，在下节将继续讲解CTreeCtrl类的相关知识和实例

# VS2010/MFC编程入门之三十一（常用控件：树形控件Tree Control 下）

       前面一节讲了[树形控件Tree Control的简介、通知消息以及相关数据结构](http://www.jizhuomi.com/software/200.html)，本节继续讲下半部分，包括树形控件的创建、CTreeCtrl类的主要成员函数和应用实例。

**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。

**树形控件的应用实例**

       最后鸡啄米还是给大家写一个简单的实例，说明CListCtrl类的几个成员函数及树形控件通知消息等的使用方法。

       此实例实现的功能：在一个树形控件中显示鸡啄米网站的简单结构分层，共有三层，分别为鸡啄米网站、各个分类和文章。用鼠标左键单击改变选中节点后，将选中节点的文本显示到[编辑框](http://www.jizhuomi.com/software/181.html)中。另外，还要实现一个常见的效果，就是鼠标划过除根节点外的某个树节点时，显示相应的Tip提示信息。下面是具体实现步骤：

       1. 创建一个基于对话框的MFC工程，名称设置为“Example31”。

       2. 在自动生成的对话框模板IDD\_EXAMPLE31\_DIALOG中，删除“TODO: Place dialog controls here.”[静态文本框](http://www.jizhuomi.com/software/179.html)、“OK”按钮和“Cancel”按钮。添加一个Tree Control控件，ID设置为IDC\_WEB\_TREE，属性Has Buttons、Has Lines和Lines At Root都设为True，为了在鼠标划过某个节点时显示提示信息还需要将Info Tip属性设为True。再添加一个静态文本框和一个编辑框，静态文本框的Caption属性设为“您选择的节点：”，编辑框的ID设为IDC\_ITEM\_SEL\_EDIT，Read Only属性设为True。此时的对话框模板如下图：

       3. 导入需要为树形控件的节点添加的图标。鸡啄米在这里找了三个32x32的Icon图标，保存到工程的res目录下。然后在Resource View资源视图中，右键点击Icon节点，在右键菜单中选择“Add Resource...”，弹出“Add Resource”对话框，再从左边“Resource type”列表中选择“Icon”，点击右边的“Import...”按钮，就可以选择三个图标文件进行导入了。导入成功后，分别修改它们ID为IDI\_WEB\_ICON、IDI\_CATALOG\_ICON和IDI\_ARTICLE\_ICON。

       4. 为树形控件IDC\_WEB\_TREE添加CTreeCtrl类型的控件变量m\_webTree。并在Example31Dlg.h文件中为CExample31Dlg类添加成员对象：CImageList m\_imageList;。

       5. 在对话框初始化时，我们在树形控件中添加鸡啄米网站的树形结构，那么需要修改CExample31Dlg::OnInitDialog()函数为：

**C++代码**

1. BOOL CExample31Dlg::OnInitDialog()
2. {
3. CDialogEx::OnInitDialog();
4. ......略
6. // TODO: Add extra initialization here
7. HICON hIcon[3];      // 图标句柄数组
8. HTREEITEM hRoot;     // 树的根节点的句柄
9. HTREEITEM hCataItem; // 可表示任一分类节点的句柄
10. HTREEITEM hArtItem;  // 可表示任一文章节点的句柄
12. // 加载三个图标，并将它们的句柄保存到数组
13. hIcon[0] = theApp.LoadIcon(IDI\_WEB\_ICON);
14. hIcon[1] = theApp.LoadIcon(IDI\_CATALOG\_ICON);
15. hIcon[2] = theApp.LoadIcon(IDI\_ARTICLE\_ICON);
17. // 创建图像序列CImageList对象
18. m\_imageList.Create(32, 32, ILC\_COLOR32, 3, 3);
19. // 将三个图标添加到图像序列
20. for (int i=0; i<3; i++)
21. {
22. m\_imageList.Add(hIcon[i]);
23. }
25. // 为树形控件设置图像序列
26. m\_webTree.SetImageList(&m\_imageList, TVSIL\_NORMAL);
28. // 插入根节点
29. hRoot = m\_webTree.InsertItem(\_T("鸡啄米"), 0, 0);
30. // 在根节点下插入子节点
31. hCataItem = m\_webTree.InsertItem(\_T("IT互联网"), 1, 1, hRoot, TVI\_LAST);
32. // 为“IT互联网”节点添加附加的编号数据，在鼠标划过该节点时显示
33. m\_webTree.SetItemData(hCataItem, 1);
34. // 在“IT互联网”节点下插入子节点
35. hArtItem = m\_webTree.InsertItem(\_T("百度文章1"), 2, 2, hCataItem, TVI\_LAST);
36. // 为“百度文章1”节点添加附加的编号数据，在鼠标划过该节点时显示
37. m\_webTree.SetItemData(hArtItem, 2);
38. // 在“IT互联网”节点下插入另一子节点
39. hArtItem = m\_webTree.InsertItem(\_T("谷歌文章2"), 2, 2, hCataItem, TVI\_LAST);
40. // 为“谷歌文章2”节点添加附加的编号数据，在鼠标划过该节点时显示
41. m\_webTree.SetItemData(hArtItem, 3);
42. // 在根节点下插入第二个子节点
43. hCataItem = m\_webTree.InsertItem(\_T("数码生活"), 1, 1, hRoot, TVI\_LAST);
44. // 为“数码生活”节点添加附加的编号数据，在鼠标划过该节点时显示
45. m\_webTree.SetItemData(hCataItem, 4);
46. // 在“数码生活”节点下插入子节点
47. hArtItem = m\_webTree.InsertItem(\_T("智能手机文章1"), 2, 2, hCataItem, TVI\_LAST);
48. // 为“智能手机文章1”节点添加附加的编号数据，在鼠标划过该节点时显示
49. m\_webTree.SetItemData(hArtItem, 5);
50. // 在“数码生活”节点下插入另一子节点
51. hArtItem = m\_webTree.InsertItem(\_T("平板电脑文章2"), 2, 2, hCataItem, TVI\_LAST);
52. // 为“平板电脑文章2”节点添加附加的编号数据，在鼠标划过该节点时显示
53. m\_webTree.SetItemData(hArtItem, 6);
54. // 在根节点下插入第三个子节点
55. hCataItem = m\_webTree.InsertItem(\_T("软件开发"), 1, 1, hRoot, TVI\_LAST);
56. // 为“软件开发”节点添加附加的编号数据，在鼠标划过该节点时显示
57. m\_webTree.SetItemData(hCataItem, 7);
58. // 在“软件开发”节点下插入子节点
59. hArtItem = m\_webTree.InsertItem(\_T("C++编程入门系列1"), 2, 2, hCataItem, TVI\_LAST);
60. // 为“C++编程入门系列1”节点添加附加的编号数据，在鼠标划过该节点时显示
61. m\_webTree.SetItemData(hArtItem, 8);
62. // 在“软件开发”节点下插入另一子节点
63. hArtItem = m\_webTree.InsertItem(\_T("VS2010/MFC编程入门2"), 2, 2, hCataItem, TVI\_LAST);
64. // 为“VS2010/MFC编程入门2”节点添加附加的编号数据，在鼠标划过该节点时显示
65. m\_webTree.SetItemData(hArtItem, 9);
66. // 在根节点下插入第四个子节点
67. hCataItem = m\_webTree.InsertItem(\_T("娱乐休闲"), 1, 1, hRoot, TVI\_LAST);
68. // 为“娱乐休闲”节点添加附加的编号数据，在鼠标划过该节点时显示
69. m\_webTree.SetItemData(hCataItem, 10);
70. // 在“娱乐休闲”节点下插入子节点
71. hArtItem = m\_webTree.InsertItem(\_T("玛雅文明文章1"), 2, 2, hCataItem, TVI\_LAST);
72. // 为“玛雅文明文章1”节点添加附加的编号数据，在鼠标划过该节点时显示
73. m\_webTree.SetItemData(hArtItem, 11);
74. // 在“娱乐休闲”节点下插入另一子节点
75. hArtItem = m\_webTree.InsertItem(\_T("IT笑话2"), 2, 2, hCataItem, TVI\_LAST);
76. // 为“IT笑话2”节点添加附加的编号数据，在鼠标划过该节点时显示
77. m\_webTree.SetItemData(hArtItem, 12);
79. return TRUE;  // return TRUE  unless you set the focus to a control
80. }

       6. 我们希望在选中节点改变时，将最新的选择项实时显示到编辑框中，那么可以响应TVN\_SELCHANGED通知消息。为树形控件IDC\_WEB\_TREE的通知消息TVN\_SELCHANGED[添加消息处理函数](http://www.jizhuomi.com/software/156.html)CExample31Dlg::OnTvnSelchangedWebTree，并修改函数体如下：

**C++代码**

1. void CExample31Dlg::OnTvnSelchangedWebTree(NMHDR \*pNMHDR, LRESULT \*pResult)
2. {
3. LPNMTREEVIEW pNMTreeView = reinterpret\_cast<LPNMTREEVIEW>(pNMHDR);
4. // TODO: Add your control notification handler code here
5. \*pResult = 0;
7. CString strText; // 树节点的标签文本字符串
9. // 获取当前选中节点的句柄
10. HTREEITEM hItem = m\_webTree.GetSelectedItem();
11. // 获取选中节点的标签文本字符串
12. strText = m\_webTree.GetItemText(hItem);
13. // 将字符串显示到编辑框中
14. SetDlgItemText(IDC\_ITEM\_SEL\_EDIT, strText);
15. }

       7. 还有一个功能需要实现，那就是鼠标划过除根节点外的某个树节点时，显示相应的Tip提示信息，本实例中提示信息为节点的编号。这需要响应TVN\_GETINFOTIP通知消息。为树形控件IDC\_WEB\_TREE的通知消息TVN\_GETINFOTIP添加消息处理函数CExample31Dlg::OnTvnGetInfoTipWebTree，并修改函数体如下：

**C++代码**

1. void CExample31Dlg::OnTvnGetInfoTipWebTree(NMHDR \*pNMHDR, LRESULT \*pResult)
2. {
3. LPNMTVGETINFOTIP pGetInfoTip = reinterpret\_cast<LPNMTVGETINFOTIP>(pNMHDR);
4. // TODO: Add your control notification handler code here
5. \*pResult = 0;
6. NMTVGETINFOTIP\* pTVTipInfo = (NMTVGETINFOTIP\*)pNMHDR;   // 将传入的pNMHDR转换为NMTVGETINFOTIP指针类型
7. HTREEITEM hRoot = m\_webTree.GetRootItem();      // 获取树的根节点
8. CString strText;     // 每个树节点的提示信息
10. if (pTVTipInfo->hItem == hRoot)
11. {
12. // 如果鼠标划过的节点是根节点，则提示信息为空
13. strText = \_T("");
14. }
15. else
16. {
17. // 如果鼠标划过的节点不是根节点，则将该节点的附加32位数据格式化为字符串
18. strText.Format(\_T("%d"), pTVTipInfo->lParam);
19. }
21. // 将strText字符串拷贝到pTVTipInfo结构体变量的pszText成员中，这样就能显示内容为strText的提示信息
22. wcscpy(pTVTipInfo->pszText, strText);
23. }

       8. 运行程序，弹出结果对话框。效果如下图：

       树形控件的知识就讲到这里了，相比之前的控件可能稍有复杂。

# VS2010/MFC编程入门之三十二（常用控件：标签控件Tab Control 上）

       前面两节鸡啄米讲了[树形控件Tree Control](http://www.jizhuomi.com/software/203.html)，本节开始讲解标签控件Tab Control，也可以称为选项卡控件。

**标签控件简介**

       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。下图是Windows系统配置中标签控件的例子：

       使用标签控件我们可以同时加载多个有关联的页面，用户只需点击标签即可实现页面切换，方便灵活的进行操作。每个标签除了可以显示标签文本，还可以显示图标。

       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。

**C++代码**

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。

**C++代码**

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。

**C++代码**

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。

**C++代码**

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

       标签控件的上半部分就讲到这里了

# VS2010/MFC编程入门之三十三（常用控件：标签控件Tab Control 下）

       上一节中鸡啄米讲了[标签控件知识的上半部分](http://www.jizhuomi.com/software/205.html)，本节继续讲下半部分。

**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。

**标签控件的应用实例**

       最后鸡啄米依然是给大家写一个简单的实例，说明CTabCtrl类的几个成员函数及标签控件通知消息等的使用方法。

       此实例实现的功能：在一个标签控件中加入两个标签页，标签文本分别为“鸡啄米”和“Android开发网”，点击不同的标签显示不同的标签页。下面是具体实现步骤：

       1. 创建一个基于对话框的MFC工程，名称设置为“Example33”。

       2. 在自动生成的对话框模板IDD\_EXAMPLE33\_DIALOG中，删除“TODO: Place dialog controls here.”[静态文本框](http://www.jizhuomi.com/software/179.html)、“OK”[按钮](http://www.jizhuomi.com/software/182.html)和“Cancel”按钮。添加一个Tab Control控件，并为其关联一个CTabCtrl类型的控件变量m\_tab。

       3. 创建两个新的对话框，ID分别设为IDD\_JIZHUOMI\_DIALOG、IDD\_ANDROID\_DIALOG，两者都将Border属性设为None，Style属性设为Child。在对话框模板IDD\_JIZHUOMI\_DIALOG中加入一个静态文本框，Caption属性设为“鸡啄米 www.jizhuomi.com”，并为其生成对话框类CJzmDlg；在对话框模板IDD\_ANDROID\_DIALOG中也加入一个静态文本框，Caption属性设为“Android开发网 www.jizhuomi.com/android”，并为其生成对话框类CAndroidDlg。

       4. 在“Example33Dlg.h”文件中包含“JzmDlg.h”和“AndroidDlg.h”两个头文件，然后继续在“Example33Dlg.h”文件中为CExample33Dlg类添加两个成员变量：

       CJzmDlg m\_jzmDlg;  
       CAndroidDlg m\_androidDlg;

       5.  在CExample33Dlg对话框初始化时，我们也初始化标签控件。修改CExample33Dlg::OnInitDialog()函数如下：

**C++代码**

1. BOOL CExample33Dlg::OnInitDialog()
2. {
3. CDialogEx::OnInitDialog();
5. // Add "About..." menu item to system menu.
7. // IDM\_ABOUTBOX must be in the system command range.
8. ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);
9. ASSERT(IDM\_ABOUTBOX < 0xF000);
11. CMenu\* pSysMenu = GetSystemMenu(FALSE);
12. if (pSysMenu != NULL)
13. {
14. BOOL bNameValid;
15. CString strAboutMenu;
16. bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);
17. ASSERT(bNameValid);
18. if (!strAboutMenu.IsEmpty())
19. {
20. pSysMenu->AppendMenu(MF\_SEPARATOR);
21. pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);
22. }
23. }
25. // Set the icon for this dialog.  The framework does this automatically
26. //  when the application's main window is not a dialog
27. SetIcon(m\_hIcon, TRUE);         // Set big icon
28. SetIcon(m\_hIcon, FALSE);        // Set small icon
30. // TODO: Add extra initialization here
31. CRect tabRect;   // 标签控件客户区的位置和大小
33. m\_tab.InsertItem(0, \_T("鸡啄米"));         // 插入第一个标签“鸡啄米”
34. m\_tab.InsertItem(1, \_T("Android开发网"));  // 插入第二个标签“Android开发网”
35. m\_jzmDlg.Create(IDD\_JIZHUOMI\_DIALOG, &m\_tab);    // 创建第一个标签页
36. m\_androidDlg.Create(IDD\_ANDROID\_DIALOG, &m\_tab); // 创建第二个标签页
38. m\_tab.GetClientRect(&tabRect);    // 获取标签控件客户区Rect
39. // 调整tabRect，使其覆盖范围适合放置标签页
40. tabRect.left += 1;
41. tabRect.right -= 1;
42. tabRect.top += 25;
43. tabRect.bottom -= 1;
44. // 根据调整好的tabRect放置m\_jzmDlg子对话框，并设置为显示
45. m\_jzmDlg.SetWindowPos(NULL, tabRect.left, tabRect.top, tabRect.Width(), tabRect.Height(), SWP\_SHOWWINDOW);
46. // 根据调整好的tabRect放置m\_androidDlg子对话框，并设置为隐藏
47. m\_androidDlg.SetWindowPos(NULL, tabRect.left, tabRect.top, tabRect.Width(), tabRect.Height(), SWP\_HIDEWINDOW);
49. return TRUE;  // return TRUE  unless you set the focus to a control
50. }

       6. 运行程序，查看结果，这时我们发现切换标签时，标签页并不跟着切换，而总是显示CJzmDlg对话框。

       7. 我们要实现的是标签页的切换效果，所以还要为m\_tab标签控件的通知消息TCN\_SELCHANGE[添加处理函数](http://www.jizhuomi.com/software/156.html)，并修改如下：

**C++代码**

1. void CExample33Dlg::OnTcnSelchangeTab1(NMHDR \*pNMHDR, LRESULT \*pResult)
2. {
3. // TODO: Add your control notification handler code here
4. \*pResult = 0;
5. CRect tabRect;    // 标签控件客户区的Rect
7. // 获取标签控件客户区Rect，并对其调整，以适合放置标签页
8. m\_tab.GetClientRect(&tabRect);
9. tabRect.left += 1;
10. tabRect.right -= 1;
11. tabRect.top += 25;
12. tabRect.bottom -= 1;
14. switch (m\_tab.GetCurSel())
15. {
16. // 如果标签控件当前选择标签为“鸡啄米”，则显示m\_jzmDlg对话框，隐藏m\_androidDlg对话框
17. case 0:
18. m\_jzmDlg.SetWindowPos(NULL, tabRect.left, tabRect.top, tabRect.Width(), tabRect.Height(), SWP\_SHOWWINDOW);
19. m\_androidDlg.SetWindowPos(NULL, tabRect.left, tabRect.top, tabRect.Width(), tabRect.Height(), SWP\_HIDEWINDOW);
20. break;
21. // 如果标签控件当前选择标签为“Android开发网”，则隐藏m\_jzmDlg对话框，显示m\_androidDlg对话框
22. case 1:
23. m\_jzmDlg.SetWindowPos(NULL, tabRect.left, tabRect.top, tabRect.Width(), tabRect.Height(), SWP\_HIDEWINDOW);
24. m\_androidDlg.SetWindowPos(NULL, tabRect.left, tabRect.top, tabRect.Width(), tabRect.Height(), SWP\_SHOWWINDOW);
25. break;
26. default:
27. break;
28. }
29. }

       8. 再运行程序，最终的标签页切换效果如下面两图：

       经过两讲内容，终于把标签控件的主要知识讲完了。

# VS2010/MFC编程入门之三十四（菜单：VS2010菜单资源详解）

       上一节讲了[标签控件Tab Control](http://www.jizhuomi.com/software/207.html)以后，常用控件的内容就全部讲完了，当然并没有包括所有控件，主要是一些很常用很重要的控件。本节开始鸡啄米将为大家讲解菜单的概念及使用。

**菜单简介**

       菜单在界面设计中是经常使用的一种元素，包括Windows系统中的窗口、智能终端设备的应用界面等都会经常见到菜单的身影。我们在对可视化窗口操作时，菜单确实提供了很大方便。

       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，大家还记得我们在[VS2010/MFC编程入门之二（利用MFC向导生成单文档应用程序框架）](http://www.jizhuomi.com/software/141.html)中创建的HelloWorld单文档工程吗？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。

**VS2010菜单资源详解**

       菜单也可以在VS2010的资源视图中直接创建编辑。我们先来创建一个新的[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)单文档工程，具体看看菜单的组成结构及各种标记的意义。

       按照VS2010/MFC编程入门之二中的步骤创建一个名为“Example34”的MFC单文档工程。打开Resource View资源视图，展开Example34->Example34.rc->Menu，我们可以看到有一个ID为IDR\_MAINFRAME菜单资源，双击打开，菜单资源显示如下图：

       上边包含“File”的一栏是主菜单栏，点击“File”弹出子菜单，可以看到子菜单中有多个菜单项和分隔条。菜单项中含有“...”则表示点击后会弹出对话框。

       除了这些，我们还注意到，很多菜单项的标题文本中都有一个字母带下划线，带下划线的字母为热键，例如，主菜单栏上的“File”中字母“F”带下划线，F就是热键，程序运行并显示窗口时，在键盘上点击Alt+F就等同于直接点菜单项File，弹出File下的子菜单后，点击“Open”的热键O就可以实现与直接点菜单项Open相同的功能。

       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。

       有些菜单项的右侧还显示了一些字符串，例如，“New”的右侧显示有“Ctrl+N”，这些代表的是快捷键，也就是“New”菜单项的快捷键是Ctrl+N，“Open”菜单项的快捷键是Ctrl+O，用这些组合键就能实现与相应菜单项一样的功能。

       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，如下图：

       Accelerator中有四列，分别为：ID、Modifier、Key和Type。ID就是菜单项的ID，Modifer和Key就代表了组合键。例如，Open菜单项的ID为ID\_FILE\_OPEN，Modifer为“Ctrl”，Key为“O”。

**VS2010菜单资源编辑**

       我们试着在Example34的IDR\_MAINFRAME菜单资源中添加菜单项。

       在主菜单栏的“Help”菜单项上点右键，弹出右键菜单，选择“Insert New”，就在“Help”菜单项前添加了一个空的菜单项，我们可以直接在其中输入标题，也可以在属性页中设置Caption属性，标题设为“&Tools”。

       然后编辑Tools下子菜单的第一个菜单项，标题设为“&Draw\tCtrl+D”，即热键为D，快捷键为Ctrl+D。其ID默认为ID\_TOOLS\_DRAW。为了实现快捷键的功能，还需要编辑Accelerator，打开Accelerator，在最下面的空白行中，ID选择为ID\_TOOLS\_DRAW，Modifier选择“Ctrl”，Key输入“D”，这样就设置好了快捷键。

       最终的菜单资源如下图：

       本节内容就是这些了，主要是关于菜单的一些基础知识，比较好理解

# VS2010/MFC编程入门之三十五（菜单：菜单及CMenu类的使用）

       鸡啄米在上一节中讲的是[VS2010的菜单资源](http://www.jizhuomi.com/software/210.html)，本节主要讲菜单及CMenu类的使用。

**CMenu类的主要成员函数**

[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。

**菜单的应用实例**

       鸡啄米先讲一下本实例要实现的功能，此实例是在上一节创建的单文档工程Example34的基础上完成的，上一节中为主菜单栏添加了“Tools”菜单项，并设置它的第一个子菜单项为“Draw”，另外我们还要为主菜单栏添加“Settings”项，然后为其添加一个子菜单项“Draw Enable”，我们通过“Draw Enable”菜单项的选中状态控制菜单项“Draw”的激活状态，如果“Draw Enable”菜单项选中，则“Draw”菜单项激活，点击它弹出一个MessageBox，否则“Draw”菜单项禁用。程序中已经在Example34View类中自动生成了OnRButtonUp(UINT /\* nFlags \*/, CPoint point)函数，并在其中实现了弹出右键菜单的功能，这里鸡啄米用CMenu类的TrackPopupMenu成员函数重新做一遍。

**注意：**Example34的CMainFrame类中定义的菜单并没有使用常用的CMenu类，而是用的CMFCMenuBar类（自VS2008起提供），但菜单的消息处理函数的添加是相同的。

       下面是具体步骤：

       1. 打开Example34工程的IDR\_MAINFRAME菜单资源，在“Help”菜单项前通过“Insert New”操作插入一个菜单项，Caption设为“Settings”，在新菜单项的子菜单中再添加一个菜单项，Caption设为“Draw Enable”，ID默认为ID\_SETTINGS\_DRAWENABLE。

       2. 因为此菜单为CMainFrame所拥有，所以我们在CMainFrame类中对菜单进行操作。在“MainFrm.h”中为CMainFrame类添加成员变量bool m\_bDraw，以标识当前是否可以点击Tools->Draw菜单项，并在CMainFrame类的[构造函数](http://www.jizhuomi.com/software/51.html)中为m\_bDraw初始化：m\_bDraw = TRUE。

       3. 为菜单项Tools->Draw的COMMAND消息和UPDATE\_COMMAND\_UI消息分别添加处理函数CMainFrame::OnToolsDraw()和OnUpdateToolsDraw(CCmdUI \*pCmdUI)，这里要注意，添加处理函数时class list中应选择CMainFrame，修改两个函数的实现为：

**C++代码**

1. void CMainFrame::OnToolsDraw()
2. {
3. // TODO: Add your command handler code here
4. // 弹出提示框
5. MessageBox(\_T("Draw"));
6. }
8. void CMainFrame::OnUpdateToolsDraw(CCmdUI \*pCmdUI)
9. {
10. // TODO: Add your command update UI handler code here
11. // 根据m\_bDraw的值设置是否激活
12. pCmdUI->Enable(m\_bDraw);
13. }

       4. 为菜单项Settings->Draw Enable的COMMAND消息和UPDATE\_COMMAND\_UI消息分别添加处理函数CMainFrame::OnSettingsDrawenable()和OnUpdateSettingsDrawenable(CCmdUI \*pCmdUI)，并将它们的实现修改为：

**C++代码**

1. void CMainFrame::OnSettingsDrawenable()
2. {
3. // TODO: Add your command handler code here
4. // 绘图使能标识取反
5. m\_bDraw = !m\_bDraw;
6. }

9. void CMainFrame::OnUpdateSettingsDrawenable(CCmdUI \*pCmdUI)
10. {
11. // TODO: Add your command update UI handler code here
12. // 根据m\_bDraw的值设置是否选中
13. pCmdUI->SetCheck(m\_bDraw);
14. }

       5. 运行程序，效果图如下：

       6. 接下来我们要重新实现右键菜单。大家以后可以仿照[VS2010](http://www.jizhuomi.com/catalog.asp?tags=VS2010)自动生成的代码实现右键菜单，也可以用鸡啄米下面讲到的方法。首先将CExample34View::OnRButtonUp(UINT /\* nFlags \*/, CPoint point)函数内的代码都注释掉，保证原来的弹出方法失效。

       7. 自动生成代码是在鼠标弹起时实现的右键菜单，我们这里改为在鼠标按下时就弹出右键菜单。在class view类视图中点击CExample34View，然后在属性页的messages列表中找到WM\_RBUTTONDOWN，添加其消息响应函数CExample34View::OnRButtonDown(UINT nFlags,CPoint point)，修改其实现为：

**C++代码**

1. void CExample34View::OnRButtonDown(UINT nFlags, CPoint point)
2. {
3. // TODO: Add your message handler code here and/or call default
4. CMenu menu;       // 菜单（包含主菜单栏和子菜单）
5. CMenu \*pSubMenu;  // 右键菜单
7. // 加载菜单资源到menu对象
8. menu.LoadMenu(IDR\_POPUP\_EDIT);
9. // 因为右键菜单是弹出式菜单，不包含主菜单栏，所以取子菜单
10. pSubMenu = menu.GetSubMenu(0);
11. // 将坐标值由客户坐标转换为屏幕坐标
12. ClientToScreen(&point);
13. // 弹出右键菜单，菜单左侧与point.x坐标值对齐
14. pSubMenu->TrackPopupMenu(TPM\_LEFTALIGN, point.x, point.y, this);
16. CView::OnRButtonDown(nFlags, point);
17. }

       8. 最终的右键菜单效果：

       本节内容不少，大家可以慢慢消化

# VS2010/MFC编程入门之三十六（工具栏：工具栏资源及CToolBar类）

       上一节中鸡啄米讲了[菜单及CMenu类的使用](http://www.jizhuomi.com/software/212.html)，这一节讲与菜单有密切联系的工具栏。

**工具栏简介**

       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。

**VS2010工具栏资源详解**

       鸡啄米仍然以[VS2010/MFC编程入门之三十四（菜单：VS2010菜单资源详解）](http://www.jizhuomi.com/software/210.html)中创建的单文档工程Example34为基础，讲解工具栏资源。

       在Example34工程中，打开Resource View资源视图，展开Example->Example34.rc->Toolbar，我们可以看到有一个ID为IDR\_MAINFRAME的工具栏资源，双击打开，工具栏资源显示如下：

       以IDR\_MAINFRAME工具栏的第一个按钮为例说明工具栏按钮的各项属性。用鼠标单击工具栏资源上的第一个按钮，属性页中就会显示其属性。下面分别讲解各项属性。

       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**

[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。

# VS2010/MFC编程入门之三十七（工具栏：工具栏的创建、停靠与使用）

       鸡啄米在上一节教程中讲了[工具栏资源及CToolBar类](http://www.jizhuomi.com/software/215.html)，本节继续讲解工具栏的相关知识，主要内容包括工具栏的创建、停靠与使用。

**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。**

       大家可以对应着看看Example34的CMainFrame类自动生成的代码中创建工具栏的过程。

       工具栏IDR\_MAINFRAME的资源已经自动创建好。在MainFrm.h文件对CMainFrame类的声明中，定义了CMFCToolBar类的对象作为成员对象：CMFCToolBar  m\_wndToolBar;。然后在CMainFrame::OnCreate函数的实现中可以看到工具栏的创建以及加载工具栏资源的代码，如下：

**C++代码**

1. int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
2. {
3. if (CFrameWndEx::OnCreate(lpCreateStruct) == -1)
4. return -1;
5. ......略
7. // 调用CreateEx函数创建工具栏，并调用LoadToolBar函数加载工具栏资源
8. if (!m\_wndToolBar.CreateEx(this, TBSTYLE\_FLAT, WS\_CHILD | WS\_VISIBLE | CBRS\_TOP | CBRS\_GRIPPER | CBRS\_TOOLTIPS | CBRS\_FLYBY | CBRS\_SIZE\_DYNAMIC) ||
9. !m\_wndToolBar.LoadToolBar(theApp.m\_bHiColorIcons ? IDR\_MAINFRAME\_256 : IDR\_MAINFRAME))
10. {
11. TRACE0("Failed to create toolbar\n");
12. return -1;      // fail to create
13. }
15. ......略
17. return 0;
18. }

       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。

        下面我们接着看Example34的CMainFrame类的OnCreate函数实现中，工具栏的停靠过程：

**C++代码**

1. int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
2. {
3. if (CFrameWndEx::OnCreate(lpCreateStruct) == -1)
4. return -1;
6. ......略
8. // 调用CreateEx函数创建工具栏，并调用LoadToolBar函数加载工具栏资源
9. if (!m\_wndToolBar.CreateEx(this, TBSTYLE\_FLAT, WS\_CHILD | WS\_VISIBLE | CBRS\_TOP | CBRS\_GRIPPER | CBRS\_TOOLTIPS | CBRS\_FLYBY | CBRS\_SIZE\_DYNAMIC) ||
10. !m\_wndToolBar.LoadToolBar(theApp.m\_bHiColorIcons ? IDR\_MAINFRAME\_256 : IDR\_MAINFRAME))
11. {
12. TRACE0("Failed to create toolbar\n");
13. return -1;      // fail to create
14. }
16. ......略
18. // TODO: Delete these five lines if you don't want the toolbar and menubar to be dockable
19. m\_wndMenuBar.EnableDocking(CBRS\_ALIGN\_ANY);
20. // 为m\_wndToolBar启用停靠
21. m\_wndToolBar.EnableDocking(CBRS\_ALIGN\_ANY);
22. // 为框架窗口启用停靠
23. EnableDocking(CBRS\_ALIGN\_ANY);
24. DockPane(&m\_wndMenuBar);
25. // 停靠工具栏
26. DockPane(&m\_wndToolBar);
28. ......略
30. return 0;
31. }

# VS2010/MFC编程入门之三十八（状态栏的使用详解）

       上一节中鸡啄米讲了[工具栏的创建、停靠与使用](http://www.jizhuomi.com/software/217.html)，本节来讲解状态栏的知识。

**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。

       用MFC向导生成的单文档或多文档程序都会自动创建状态栏，大家可以运行下[VS2010/MFC编程入门之三十四（菜单：VS2010菜单资源详解）](http://www.jizhuomi.com/software/210.html)中创建的Example34程序，在结果界面中可以看到窗口底部有个状态栏，该状态栏被分为了几个窗格，分别用来显示菜单项和工具栏按钮的提示信息及Caps Lock、Num Lock、Scroll Lock键的状态。

       当然，我们可以自定义状态栏，加入新的提示信息或指示器。

**CStatusBar类**

[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**

       在Example34程序中，我们在CMainFrame类中看到，创建状态栏时使用的是CMFCStatusBar类对象。CMFCStatusBar类是自VS2008以来提供的状态栏类，用法与CStatusBar类相似，甚至很多成员函数也类似，但它的功能更加丰富。关于CMFCStatusBar类的成员函数可以查阅MSDN了解。

       鸡啄米下面就以Example34程序的CMFCStatusBar类对象为例，来讲讲状态栏的创建步骤：

       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：

**C++代码**

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：

**C++代码**

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：

**C++代码**

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。

**状态栏应用实例**

       鸡啄米看到网上有很多人在问，怎样在状态栏添加一个时间窗格，用来显示系统时间，本节就给出这样一个实例。此实例依然是在Example34的基础上进行修改的。步骤如下：

       1. 在Resource View资源视图中打开String Table字符串资源，然后在最后一行的下一个空白行中，或者任意处点右键选择“New String”，添加一个新的字符串资源，ID为ID\_INDICATOR\_TIME，Value设为一个不与任何其他字符串资源重复的整数值，Caption设为"00:00:00"，这是为了给时间的显示预留空间，因为状态栏会根据字符串的长度为相应的窗格确定缺省宽度。

       2. 在indicators数组的第一个元素ID\_INDICATOR\_SCRL后插入ID\_INDICATOR\_TIME。

**C++代码**

1. static UINT indicators[] =
2. {
3. ID\_SEPARATOR,           // status line indicator
4. ID\_INDICATOR\_CAPS,
5. ID\_INDICATOR\_NUM,
6. ID\_INDICATOR\_SCRL,
7. ID\_INDICATOR\_TIME
8. };

       3. 要实时显示系统时间，就需要使用一个定时器，每秒钟更新一次时间显示。在CMainFrame::OnCreate函数中开启定时器，代码如下：

**C++代码**

1. int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
2. {
3. if (CFrameWndEx::OnCreate(lpCreateStruct) == -1)
4. return -1;
6. ......略
8. // 启动定时器，定时器ID为1，定时时间为1000ms，即1s
9. SetTimer(1, 1000, NULL);
11. return 0;
12. }

        4. 在Class View类视图中找到CMainFrame类，右键选择“Properties”，然后在显示出来的属性页中，点工具栏上的Messages按钮，即显示出消息列表，找到WM\_TIMER消息，添加其消息处理函数void CMainFrame::OnTimer(UINT\_PTR nIDEvent)，并修改此函数实现如下：

**C++代码**

1. void CMainFrame::OnTimer(UINT\_PTR nIDEvent)
2. {
3. // TODO: Add your message handler code here and/or call default
4. CString strTime;
5. // 获取系统当前时间，并保存到curTime
6. CTime curTime = CTime::GetCurrentTime();
8. // 格式化curTime，将字符串保存到strTime
9. strTime = curTime.Format(\_T("%H:%M:%S"));
10. // 在状态栏的时间窗格中显示系统时间字符串
11. m\_wndStatusBar.SetPaneText(4, strTime);
13. CFrameWndEx::OnTimer(nIDEvent);
14. }

       5. 运行程序，我们看到状态栏的最后一个窗格中能够实时显示系统时间，如下图：

# VS2010/MFC编程入门之三十九（文档、视图和框架：概述）

       前面几节讲了[菜单](http://www.jizhuomi.com/software/212.html)、[工具栏](http://www.jizhuomi.com/software/215.html)和[状态栏](http://www.jizhuomi.com/software/219.html)的使用，鸡啄米本节开始将为大家讲解文档、视图和框架的知识。

**文档、视图和框架简介**

       在[VS2010/MFC编程入门之三十四（菜单：VS2010菜单资源详解）](http://www.jizhuomi.com/software/210.html)创建的单文档工程Example34中，我们可以看到[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)向导自动为我们生成了CExample34Doc类、CExample34View类和CMainFrame类，它们就分别是文档类、视图类和框架窗口类。

       文档/视图结构是MFC提供的一种不错的设计，它将数据的处理和显示分开来，这样更便于我们对程序的维护和扩展。下面分别介绍这种结构中涉及到的几个重要概念。

**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。

       我们来看看Example34单文档程序中，CExample34App应用程序类的成员函数CExample34App::InitInstance()创建并注册文档模板的部分：

**C++代码**

1. BOOL CExample34App::InitInstance()
2. {
3. ......略
4. // Register the application's document templates.  Document templates
5. //  serve as the connection between documents, frame windows and views
6. CSingleDocTemplate\* pDocTemplate;
7. pDocTemplate = new CSingleDocTemplate(
8. IDR\_MAINFRAME,
9. RUNTIME\_CLASS(CExample34Doc),
10. RUNTIME\_CLASS(CMainFrame),       // main SDI frame window
11. RUNTIME\_CLASS(CExample34View));
12. if (!pDocTemplate)
13. return FALSE;
14. AddDocTemplate(pDocTemplate);
16. ......略
18. return TRUE;
19. }

       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。

# VS2010/MFC编程入门之四十（文档、视图和框架：各对象之间的关系）

       前面一节中鸡啄米进行了[文档、视图和框架的概述](http://www.jizhuomi.com/software/221.html)，本节主要讲解文档、视图、框架结构中各对象之间的关系。

**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。

       鸡啄米在网上找到了一张分别对应三种视图模式的图如下：

       有关文档、视图和框架等对象之间的关系就讲到这里了

# VS2010/MFC编程入门之四十一（文档、视图和框架：分割窗口）

      上一节中鸡啄米讲了[文档、视图和框架结构中各对象之间的关系](http://www.jizhuomi.com/software/223.html)，本节主要讲讲在[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)中如何分割窗口。

**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。

       大家看下Word里的动态分割窗口就很明白了，以Word 2007文档为例，在菜单中点击“视图”->“拆分”，就可以看到一条随鼠标移动的分隔条，当我们在文档中某个位置按下鼠标左键时，分割条就固定了下来，生成了上下两个分割窗格，通过滚动每个窗格中的垂直滚动条可以看到，两个窗格中的内容相同，这就是所说的对应同一个文档的同一类视图。

       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。

       鸡啄米下面介绍三个最常用的成员函数：

**C++代码**

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。

**C++代码**

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。

**C++代码**

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。

       下面鸡啄米给大家一个实例。同样以[VS2010/MFC编程入门之三十四（菜单：VS2010菜单资源详解）](http://www.jizhuomi.com/software/210.html)中创建Example34工程为例，我们要实现在主框架窗口的客户区中创建两行两列的动态分割窗口。以下是创建动态分割窗口的具体步骤：

       1. 在MainFrm.h文件中为CMainFrame类添加成员对象：CSplitterWnd m\_wndSplitter;。

       2. 在Class View类视图中找到CMainFrame类，右键点击，在右键菜单中选择Properties，就会显示属性页，然后在属性页的工具栏上点击Tip为Overrides的按钮，下面的列表中就列出了能够重载的函数，找到OnCreateClient生成重载函数。

       3. 在MainFrm.cpp文件中找到刚重载的OnCreateClient函数修改如下：

**C++代码**

1. BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext\* pContext)
2. {
3. // TODO: Add your specialized code here and/or call the base class
4. // 创建动态分割窗口，两行两列
5. return m\_wndSplitter.Create(this,2, 2, CSize(10, 10), pContext);
7. //return CFrameWndEx::OnCreateClient(lpcs, pContext);
8. }

       4. 在Resource View资源视图中，打开Menu下的IDR\_MAINFRAME菜单，在View下添加一个菜单项，Caption设为Splitter Window，ID设为（一定要设为）ID\_WINDOW\_SPLIT。这样在运行结果界面中点击此菜单项时MFC会执行一些操作显示动态分割窗口。

       5. 运行程序，点击菜单中的View->Splitter Window菜单项，创建动态分割窗口后效果如下：

**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。

       鸡啄米仍通过Example34工程给大家一个实例，目的是在主框架窗口中的客户区创建一个两行一列的静态分割窗口。如果已经试验过动态分割窗口的创建，那么麻烦撤销那些修改吧。创建静态分割窗口的具体步骤如下：

       1. 在MainFrm.h文件中为CMainFrame类添加成员对象：CSplitterWnd m\_wndSplitter;。

       2. 在Class View类视图中找到CMainFrame类，右键点击，在右键菜单中选择Properties，就会显示属性页，然后在属性页的工具栏上点击Tip为Overrides的按钮，下面的列表中就列出了能够重载的函数，找到OnCreateClient生成重载函数。

       3. 在MainFrm.cpp文件中找到刚重载的OnCreateClient函数进行修改。因为鸡啄米没有新建其他视图类，所以上下两个窗格的视图都是CExample34View。为了能识别CExample34View类，还需在MainFrm.cpp文件中添加#include "Example34View.h"，在Example34View.h文件中添加#include "Example34Doc.h"。最终OnCreateClient函数修改如下：

**C++代码**

1. BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext\* pContext)
2. {
3. // TODO: Add your specialized code here and/or call the base class
4. CRect rc;
6. // 获取框架窗口客户区的CRect对象
7. GetClientRect(&rc);
9. // 创建静态分割窗口，两行一列
10. if (!m\_wndSplitter.CreateStatic(this, 2, 1))
11. return FALSE;
13. // 创建上面窗格中的视图
14. if (!m\_wndSplitter.CreateView(0, 0, RUNTIME\_CLASS(CExample34View), CSize(rc.Width(), rc.Height()/2), pContext))
15. return FALSE;
17. // 创建下面窗格中的视图
18. if (!m\_wndSplitter.CreateView(1, 0, RUNTIME\_CLASS(CExample34View), CSize(rc.Width(), rc.Height()/2), pContext))
19. return FALSE;
21. return TRUE;
23. //return CFrameWndEx::OnCreateClient(lpcs, pContext);
24. }

       4. 运行程序，在结果界面中关掉其他面板后效果如下：

       如果大家想创建在其中某个窗格中再嵌套分割窗口，那么就需要再定义一个CSplitterWnd对象，以父窗格所在的CSplitterWnd对象为父框架窗口创建分割窗口即可。

# VS2010/MFC编程入门之四十二（MFC常用类：CString类）

       上一节鸡啄米讲了[分割窗口](http://www.jizhuomi.com/software/226.html)的有关知识，本节开始讲解[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)的一些常用类，先来说说CString类。

**CString类简介**

       CString类作为MFC的常用类，当之无愧。可以这样说，只要是从事MFC开发，基本都会遇到使用CString类的场合。因为字符串的使用比较普遍，而CString类又提供了对字符串的便捷操作，所以它给MFC开发人员带来了高的开发效率，受到了开发者的欢迎。

       大家使用[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。鸡啄米以CString类为讲解对象。

**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：

**C++代码**

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：

**C++代码**

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：

**C++代码**

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：

**C++代码**

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：

**C++代码**

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：

**C++代码**

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：

**C++代码**

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。

**C++代码**

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：

**C++代码**

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：

**C++代码**

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"

        好了，关于CString类的内容就讲到这里了，用法不少，但仍不全，大家可以查看MSDN深入学习。

# VS2010/MFC编程入门之四十三（MFC常用类：CTime类和CTimeSpan类）

       上一节中鸡啄米讲了[MFC常用类CString类](http://www.jizhuomi.com/software/228.html)的用法，本节继续讲另外两个[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)常用类-日期和时间类CTime类和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。

**CTime类和CTimeSpan类的应用实例**

       鸡啄米在下面将为大家演示如何得到当前时间、计算两个时间的时间差以及CTime对象怎样格式化为字符串等。具体步骤如下：

       1. 创建一个Win32 Console Application工程，Name设为“Example43”。

       2. 因为要使用到CTime类、CTimeSpan类和cout输出流，所以在Example43.cpp文件中包含相应的头文件：

**C++代码**

1. #include "atltime.h"
2. #include <iostream>
3. using namespace std;

        3. 修改main函数如下：

**C++代码**

1. int \_tmain(int argc, \_TCHAR\* argv[])
2. {
3. CString strTime;  // 用于将CTime对象格式化为字符串
4. // 获取当前时间并保存到curTime
5. CTime curTime = CTime::GetCurrentTime();
7. int nYear = curTime.GetYear();  // 获取当前年份
8. int nMonth = curTime.GetMonth(); // 获取当前月份
9. int nDay = curTime.GetDay();   // 获取当前日期
10. int nHour = curTime.GetHour();  // 获取当前小时时间
11. int nMin = curTime.GetMinute(); // 获取当前分钟时间
12. int nSec = curTime.GetSecond(); // 获取当前秒时间
14. // 输出当前时间
15. cout<<"当前时间："<<endl;
16. cout<<nYear<<"年"<<nMonth<<"月"<<nDay<<"日"<<nHour<<"时"<<nMin<<"分"<<nSec<<"秒"<<endl;
18. // 为计算时间差，设定一个起始时间并输出
19. CTime startTime = CTime(2010,7,3,10,20,30);
20. cout<<"起始时间："<<endl;
21. cout<<startTime.GetYear()<<"年"<<startTime.GetMonth()<<"月"<<startTime.GetDay()<<"日"<<startTime.GetHour()<<"时"<<startTime.GetMinute()<<"分"<<startTime.GetSecond()<<"秒"<<endl;
23. // 计算时间差
24. CTimeSpan timeSpan;
25. timeSpan = curTime - startTime;
26. cout<<"两时间相差："<<endl;
27. cout<<timeSpan.GetDays()<<"天"<<timeSpan.GetHours()<<"小时"<<timeSpan.GetMinutes()<<"分"<<timeSpan.GetSeconds()<<"秒"<<endl;
28. cout<<"总小时数："<<timeSpan.GetTotalHours()<<"小时"<<endl;
29. cout<<"总分钟数："<<timeSpan.GetTotalMinutes()<<"分"<<endl;
30. cout<<"总秒数："<<timeSpan.GetTotalSeconds()<<"秒"<<endl;
32. // 将当前时间curTime对象格式化为字符串
33. strTime = curTime.Format(\_T("%Y-%m-%d %H:%M:%S"));
34. // 输出格式化字符串，由于字符串使用Unicode字符，所以要使用wcout输出
35. wcout<<(LPCTSTR)strTime<<endl;
37. return 0;
38. }

       其中的注释已经解释的比较清楚，相信大家都能够理解。

       4. 运行程序，效果如下：

       好了，有关CTime类和CTimeSpan类的相关内容就讲到这里了，如果大家觉得还有必要了解更加详细的知识可以参阅MSDN文档。

# VS2010/MFC编程入门之四十四（MFC常用类：定时器Timer）

       前面一节鸡啄米讲了[CTime类和CTimeSpan类](http://www.jizhuomi.com/software/230.html)的使用，本节继续讲与时间有关的定时器。定时器并不是一个类，主要考虑到，提起时间的话就不能不说定时器，所以就把它放到CTime和CTimeSpan之后讲解。

**定时器简介**

       定时器，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。因为本教程主要是讲解MFC编程，所以这里就先重点讲解MFC定时器的用法，关于API函数SetTimer的用法鸡啄米会在MFC定时器讲解的基础上进行延伸。

       鸡啄米下面分步骤给出使用MFC定时器的方法。

**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：

**C++代码**

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：

**C++代码**

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：

**C++代码**

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。**

鸡啄米给出一个回调函数的例子：

**C++代码**

1. void CALLBACK EXPORT TimerProc(HWND hWnd,UINT nMsg,UINT nTimerid,DWORD dwTime)
2. {
3. switch(nTimerid)
4. {
5. case 1:
6. // 处理ID为1的定时器的事件
7. func1();
8. break;
9. case 2:
10. // 处理ID为2的定时器的事件
11. func2();
12. break;
13. ......
14. default:
15. break;
16. }
17. }

       回调函数为全局函数，需要写在使用它的位置的前面，或者写在后面然后在使用之前声明。

**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：

**C++代码**

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。

**通过Windows API函数使用定时器**

       如果我们不使用MFC定时器，而通过Windows API函数使用定时器，其实是很类似的。下面鸡啄米简单说下步骤吧。

**1、启动定时器。**

       使用API函数SetTimer启动定时器，SetTimer函数的原型如下：

**C++代码**

1. UINT\_PTR SetTimer(
2. HWND
3. hWnd,
4. UINT\_PTR
5. nIDEvent,
6. UINT
7. uElapse,
8. TIMERPROC
9. lpTimerFunc
10. );

       参数hWnd为与定时器关联的窗口的句柄；参数nIDEvent为非零的定时器ID，如果hWnd等于NULL，且还不存在ID为nIDEvent的定时器，那么nIDEvent参数被忽略，然后生成一个新ID的定时器，而如果hWnd不为NULL，且hWnd指定的窗口已存在ID为nIDEvent的定时器，那么这个已存在的定时器被新定时器所取代。参数uElapse和lpTimerFunc同CWnd::SetTimer函数。

**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SetTimer函数时最后一个参数为NULL，我们需要自己为WM\_TIMER消息添加处理函数，要注意的是，WM\_TIMER消息的附加数据wParam为定时器ID，lParam为回调函数的指针，如果调用SetTimer时回调函数为NULL，那么lParam也为NULL。

       而如果调用SetTimer函数时最后一个参数不为NULL，我们就需要定义回调函数。回调函数的定义同MFC定时器。

**3、销毁定时器。**

       销毁定时器使用KillTimer API函数，原型如下：

**C++代码**

1. BOOL KillTimer(HWND hWnd,UINT\_PTR uIDEvent);

       参数hWnd为与定时器关联的窗口的句柄，与启动定时器时SetTimer函数的hWnd参数值相同；参数uIDEvent为要销毁的定时器的ID，如果传递给SetTimer的参数hWnd有效，则uIDEvent应与传递给SetTimer的参数nIDEvent相同，而如果SetTimer的参数hWnd为NULL，则uIDEvent应为SetTimer返回的定时器ID。该函数成功则返回TRUE，否则返回FALSE。

**MFC定时器应用实例**

       鸡啄米给大家演示一个定时器的例子，该实例功能很简单，就是使用两个定时器，定时更新两个编辑框中的显示内容，第一个编辑框每秒刷新一次，从1刷新到10，然后销毁定时器，第二个编辑框每两秒刷新一次，从1刷新到5，然后销毁定时器。下面简单说下步骤：

       1、创建基于对话框的工程，名称设为“Example44”。

       2、在自动生成的对话框模板IDD\_EXAMPLE44\_DIALOG中，删除“TODO: Place dialog controls here.”静态文本控件。添加两个[静态文本框](http://www.jizhuomi.com/software/179.html)控件，Caption分别设为“1秒钟刷新一次”和“2秒钟刷新一次”，再添加两个个[Edit Control控件](http://www.jizhuomi.com/software/181.html)，ID使用默认的IDC\_EDIT1和IDC\_EDIT2，两者的Read Only属性都设为True。此时的对话框模板如下图：

       3、为CExample44Dlg类添加两个成员变量，分别为m\_nData1、m\_nData2，并在CExample44Dlg类的构造函数中初始化：

**C++代码**

1. CExample44Dlg::CExample44Dlg(CWnd\* pParent /\*=NULL\*/)
2. : CDialogEx(CExample44Dlg::IDD, pParent)
3. {
4. m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);
5. // 两个数据初始化为0
6. m\_nData1 = 0;
7. m\_nData2 = 0;
8. }

       4、在对话框模板上双击OK按钮，添加点击消息的处理函数，并修改如下：

**C++代码**

1. void CExample44Dlg::OnBnClickedOk()
2. {
3. // TODO: Add your control notification handler code here
4. // 启动ID为1的定时器，定时时间为1秒
5. SetTimer(1, 1000, NULL);
6. // 启动ID为2的定时器，定时时间为2秒
7. SetTimer(2, 2000, NULL);
9. //CDialogEx::OnOK();
10. }

       这样，点击OK按钮时就不会退出，而是启动两个定时器。

       5、根据上面MFC定时器讲解中为WM\_TIMER消息添加处理函数的方法，添加WM\_TIMER的消息处理函数OnTimer，并修改其实现如下：

**C++代码**

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m\_nData1已经达到10，则销毁ID为1的定时器
8. if (10 == m\_nData1)
9. {
10. KillTimer(1);
11. break;
12. }
13. // 刷新编辑框IDC\_EDIT1的显示
14. SetDlgItemInt(IDC\_EDIT1, ++m\_nData1);
15. break;
16. case 2:
17. // 如果m\_nData1已经达到5，则销毁ID为2的定时器
18. if (5 == m\_nData2)
19. {
20. KillTimer(2);
21. break;
22. }
23. // 刷新编辑框IDC\_EDIT2的显示
24. SetDlgItemInt(IDC\_EDIT2, ++m\_nData2);
25. default:
26. break;
27. }
29. CDialogEx::OnTimer(nIDEvent);
30. }

       6、运行程序，点击OK按钮，查看效果。

       关于定时器的内容就讲这些

# VS2010/MFC编程入门之四十五（MFC常用类：CFile文件操作类）

       上一节中鸡啄米讲了[定时器Timer](http://www.jizhuomi.com/software/232.html)的用法，本节介绍下文件操作类CFile类的使用。

**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。

**C++代码**

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：

**C++代码**

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

       下面是文件操作的代码片段：

**C++代码**

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. }

       本节内容就到这里，如果有其他语言的文件操作的经验的话，应该还是比较简单的。

# VS2010/MFC编程入门之四十六（MFC常用类：MFC异常处理）

       上一节中鸡啄米讲了[CFile文件操作类](http://www.jizhuomi.com/software/234.html)，本节主要来说说MFC异常处理。

       在[鸡啄米C++编程入门系列](http://www.jizhuomi.com/software/129.html)的最后一节[鸡啄米：C++编程入门系列之五十（异常处理）](http://www.jizhuomi.com/software/127.html)中，鸡啄米讲了C++标准异常的处理机制，如果你还没有学过这方面内容，可以到那节教程中去学习下。

       MFC异常处理与C++标准异常处理是类似的，只是它在抛出和捕获异常时使用了一些宏，另外还将异常封装到了CException类及其派生类中。下面就分别讲解这些宏与异常类。

[**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。鸡啄米下面贴出MFC中这些宏的定义：

**C++代码**

1. /////////////////////////////////////////////////////////////////////////////
2. // Exception macros using try, catch and throw
3. //  (for backward compatibility to previous versions of MFC)
5. #define TRY { AFX\_EXCEPTION\_LINK \_afxExceptionLink; try {
7. #define CATCH(class, e) } catch (class\* e) \
8. { ASSERT(e->IsKindOf(RUNTIME\_CLASS(class))); \
9. \_afxExceptionLink.m\_pException = e;
11. #define AND\_CATCH(class, e) } catch (class\* e) \
12. { ASSERT(e->IsKindOf(RUNTIME\_CLASS(class))); \
13. \_afxExceptionLink.m\_pException = e;
15. #define END\_CATCH } }
17. #define THROW(e) throw e
18. #define THROW\_LAST() (AfxThrowLastCleanup(), throw)
20. // Advanced macros for smaller code
21. #define CATCH\_ALL(e) } catch (CException\* e) \
22. { { ASSERT(e->IsKindOf(RUNTIME\_CLASS(CException))); \
23. \_afxExceptionLink.m\_pException = e;
25. #define AND\_CATCH\_ALL(e) } catch (CException\* e) \
26. { { ASSERT(e->IsKindOf(RUNTIME\_CLASS(CException))); \
27. \_afxExceptionLink.m\_pException = e;
29. #define END\_CATCH\_ALL } } }
31. #define END\_TRY } catch (CException\* e) \
32. { ASSERT(e->IsKindOf(RUNTIME\_CLASS(CException))); \
33. \_afxExceptionLink.m\_pException = e; } }

       可以看出这些宏的定义中都包含了相应的[C++](http://www.jizhuomi.com/catalog.asp?tags=C%EF%BC%8B%EF%BC%8B)异常处理关键字，本质上还是要通过try、catch和throw实现。

**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。

**C++代码**

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：

**C++代码**

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。

# VS2010/MFC编程入门之四十七（字体和文本输出：CFont字体类）

       上一节中鸡啄米讲了[MFC异常处理](http://www.jizhuomi.com/software/236.html)，本节的主要内容是字体CFont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类为字体的操作提供了几个成员函数，下面是对其各个成员函数的详细介绍。

**1、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。

**2、**

**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。

**3、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[VS2010/MFC编程入门之十八（对话框：字体对话框）](http://www.jizhuomi.com/software/175.html)。

**4、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。

**5、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。

**6、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。

**7、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。

# VS2010/MFC编程入门之四十八（字体和文本输出：文本输出）

       鸡啄米在上一节中讲了[CFont字体类](http://www.jizhuomi.com/software/239.html)，本节主要讲解文本输出的方法和实例。

**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：

**C++代码**

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：

**C++代码**

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。

**字体和文本输出的应用实例**

       鸡啄米下面给大家演示一个简单的关于字体和文本输出的实例。功能就是实现两个字符串分别在水平方向和垂直方向上定时滚动。实现步骤如下：

       1、创建一个基于对话框的[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)工程，名字设置为“Example48”。

       2、在自动生成的对话框模板IDD\_EXAMPLE48\_DIALOG中，删除“TODO: Place dialog controls here.”[静态文本框](http://www.jizhuomi.com/software/179.html)。

       3、在Example48Dlg.h文件中为CExample48类添加成员变量：

**C++代码**

1. int m\_nTextX;   // 水平滚动文本的起始点的x坐标
2. int m\_nTextY;   // 垂直滚动文本的起始点的y坐标
3. CFont m\_newFont;   // 新字体
4. CFont \*m\_pOldFont; // 选择新字体之前的字体

       4、在CExample48Dlg类的构造函数中，初始化新添加的成员变量：

**C++代码**

1. CExample48Dlg::CExample48Dlg(CWnd\* pParent /\*=NULL\*/)
2. : CDialogEx(CExample48Dlg::IDD, pParent)
3. {
4. m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);
6. m\_nTextX = 260;
7. m\_nTextY = 10;
8. m\_pOldFont = NULL;
9. }

       5、在CExample48Dlg对话框初始化函数中，创建新的字体，并开启[定时器](http://www.jizhuomi.com/software/232.html)：

**C++代码**

1. BOOL CExample48Dlg::OnInitDialog()
2. {
3. CDialogEx::OnInitDialog();
5. // Add "About..." menu item to system menu.
7. // IDM\_ABOUTBOX must be in the system command range.
8. ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);
9. ASSERT(IDM\_ABOUTBOX < 0xF000);
11. CMenu\* pSysMenu = GetSystemMenu(FALSE);
12. if (pSysMenu != NULL)
13. {
14. BOOL bNameValid;
15. CString strAboutMenu;
16. bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);
17. ASSERT(bNameValid);
18. if (!strAboutMenu.IsEmpty())
19. {
20. pSysMenu->AppendMenu(MF\_SEPARATOR);
21. pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);
22. }
23. }
25. // Set the icon for this dialog.  The framework does this automatically
26. //  when the application's main window is not a dialog
27. SetIcon(m\_hIcon, TRUE);         // Set big icon
28. SetIcon(m\_hIcon, FALSE);        // Set small icon
30. // TODO: Add extra initialization here
31. // 创建一种新的字体（18点，隶书）
32. m\_newFont.CreatePointFont(180, \_T("隶书"));
34. // 设置定时器，定时时间为200ms
35. SetTimer(1,200,NULL);
37. return TRUE;  // return TRUE  unless you set the focus to a control
38. }

       6、修改CExample48Dlg::OnPaint()函数，如果窗口没有最小化就在指定的位置输出文本，即在OnPaint函数中if(IsIconic())对应的else大括号内添加相应代码。CExample48Dlg::OnPaint()函数修改如下：

**C++代码**

1. void CExample48Dlg::OnPaint()
2. {
3. if (IsIconic())
4. {
5. CPaintDC dc(this); // device context for painting
7. SendMessage(WM\_ICONERASEBKGND, reinterpret\_cast<WPARAM>(dc.GetSafeHdc()), 0);
9. // Center icon in client rectangle
10. int cxIcon = GetSystemMetrics(SM\_CXICON);
11. int cyIcon = GetSystemMetrics(SM\_CYICON);
12. CRect rect;
13. GetClientRect(&rect);
14. int x = (rect.Width() - cxIcon + 1) / 2;
15. int y = (rect.Height() - cyIcon + 1) / 2;
17. // Draw the icon
18. dc.DrawIcon(x, y, m\_hIcon);
19. }
20. else
21. {
22. CPaintDC dc(this); // device context for painting
23. // 设置m\_newFont对象的字体为当前字体，并将之前的字体指针保存到m\_pOldFont
24. m\_pOldFont = (CFont\*)dc.SelectObject(&m\_newFont);
25. // 设置
26. dc.SetBkMode(TRANSPARENT); //设置背景为透明！
27. // 设置文本颜色为红色
28. dc.SetTextColor(RGB(255,0,0));
29. // 在指定位置输出文本
30. dc.TextOut(m\_nTextX,10,\_T("欢迎来到鸡啄米！"));
31. // 设置文本颜色为绿色
32. dc.SetTextColor(RGB(0,255,0));
33. // 在指定位置输出文本
34. dc.TextOut(10,m\_nTextY,\_T("谢谢关注www.jizhuomi.com"));
35. // 恢复以前的字体
36. dc.SelectObject(m\_pOldFont);
38. CDialogEx::OnPaint();
39. }
40. }

       7、在Class View类视图中找到CExample48Dlg，右键点Properties，显示出其属性页，在属性页工具栏上点击Messages按钮，找到WM\_TIMER消息，添加消息响应函数CExample48Dlg::OnTimer(UINT\_PTR nIDEvent)，并在此函数中修改两个文本输出的坐标位置。

**C++代码**

1. void CExample48Dlg::OnTimer(UINT\_PTR nIDEvent)
2. {
3. // TODO: Add your message handler code here and/or call default
4. LOGFONT logFont;
5. // 获取m\_newFont字体的LOGFONT结构
6. m\_newFont.GetLogFont(&logFont);
8. // 将m\_nTextX的值减5
9. m\_nTextX -= 5;
10. // 如果m\_nTextX小于10，则文本“欢迎来到鸡啄米”回到起始位置
11. if (m\_nTextX < 10)
12. m\_nTextX = 260;
14. // 将m\_nTextY的值加一个字符高度
15. m\_nTextY += abs(logFont.lfHeight);
16. // 如果m\_nTextY大于260，则文本“谢谢关注www.jizhuomi.com”回到起始位置
17. if (m\_nTextY >260)
18. m\_nTextY = 10;
20. // 使窗口客户区无效，之后就会重绘
21. Invalidate();
23. CDialogEx::OnTimer(nIDEvent);
24. }

       到这一步，两个文本就可以分别在水平和垂直方向滚动了。鸡啄米再简单解释下这个过程：程序刚启动时，会调用OnPaint函数，在初始位置绘出两个文本，然后每次到了定时器的定时时间后，会执行OnTimer函数，修改两个文本的坐标值，并通过Invalidate使窗口重绘，又会重新调用OnPaint函数绘制两个文本。这样通过定时修改坐标值就实现了两个文本的滚动效果。

       8、运行程序，最终的效果如下图：

# VS2010/MFC编程入门之四十九（图形图像：CDC类及其屏幕绘图函数）

       上一节中鸡啄米讲了[文本输出](http://www.jizhuomi.com/software/241.html)的知识，本节的主要内容是CDC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）。

       好了，本节就讲到这里了，主要就是大概讲了讲CDC类，又介绍了CDC类一些常用的绘图函数。如果想了解更多的资料可以查阅MSDN。

# VS2010/MFC编程入门之五十（图形图像：GDI对象之画笔CPen）

       上一节中鸡啄米讲了[CDC类及其屏幕绘图函数](http://www.jizhuomi.com/software/244.html)，本节的主要内容是GDI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，即CPen类和CBrush类。本文就主要讲解画笔的使用。

**画笔的应用实例**

       鸡啄米在这里直接通过一个波形图的实例，来详细讲解画笔的使用方法。

       首先介绍此实例要实现的功能：在对话框上有一个Picture控件，将此控件的背景填充为黑色；启动一个定时器，每次定时器到时，所有波形数据都前移一个单位，并获取一个80以内的随机数作为波形的最后一个数据，然后以绿色画笔在绘图控件上绘制波形。这样就实现了波形的绘制及动态变化。

       下面是具体实施步骤：

       1、创建一个基于对话框的MFC工程，名字设为“Example50”。

       2、在自动生成的对话框模板IDD\_EXAMPLE50\_DIALOG中，删除“TODO: Place dialog controls here.”[静态文本框](http://www.jizhuomi.com/software/179.html)，添加一个[Picture控件](http://www.jizhuomi.com/software/193.html)，ID设为IDC\_WAVE\_DRAW。

       3、为Picture控件IDC\_WAVE\_DRAW添加CStatic变量，名称设为m\_picDraw。

       4、在文件Example50Dlg.h文件中CExample50Dlg类声明的上面添加宏定义：

**C++代码**

1. #define POINT\_COUNT 100

       此符号常量的意义是波形的点数，这里[用define将其定义为符号常量](http://www.jizhuomi.com/software/70.html)是为了方便以后可能的修改，假如我们以后想将点数改为200，则只改此宏定义就可以了：#define POINT\_COUNT 200，而如果没有使用符号常量，在程序中直接使用了100，那么就需要将所有使用100的位置找出来，并替换为200，这样不仅麻烦也很容易出错，所以最好是将其定义为符号常量。

       5、在CExample50Dlg.h文件中为CExample50Dlg类添加成员数组：

**C++代码**

1. int   m\_nzValues[POINT\_COUNT];

       此数组用于存放波形数据。

       6、在CExample50Dlg类的构造函数中为数组m\_nzValues的元素赋初值：

**C++代码**

1. CExample50Dlg::CExample50Dlg(CWnd\* pParent /\*=NULL\*/)
2. : CDialogEx(CExample50Dlg::IDD, pParent)
3. {
4. m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);
6. // 将数组m\_nzValues的元素都初始化为0
7. memset(m\_nzValues, 0, sizeof(int) \* POINT\_COUNT);
8. }

       7、在CExample50Dlg对话框的初始化成员函数CExample50Dlg::OnInitDialog()中，构造随机数生成器，并启动定时器。CExample50Dlg::OnInitDialog()修改如下：

**C++代码**

1. BOOL CExample50Dlg::OnInitDialog()
2. {
3. CDialogEx::OnInitDialog();
5. // Add "About..." menu item to system menu.
7. // IDM\_ABOUTBOX must be in the system command range.
8. ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);
9. ASSERT(IDM\_ABOUTBOX < 0xF000);
11. CMenu\* pSysMenu = GetSystemMenu(FALSE);
12. if (pSysMenu != NULL)
13. {
14. BOOL bNameValid;
15. CString strAboutMenu;
16. bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);
17. ASSERT(bNameValid);
18. if (!strAboutMenu.IsEmpty())
19. {
20. pSysMenu->AppendMenu(MF\_SEPARATOR);
21. pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);
22. }
23. }
25. // Set the icon for this dialog.  The framework does this automatically
26. //  when the application's main window is not a dialog
27. SetIcon(m\_hIcon, TRUE);         // Set big icon
28. SetIcon(m\_hIcon, FALSE);        // Set small icon
30. // TODO: Add extra initialization here
32. // 以时间为种子来构造随机数生成器
33. srand((unsigned)time(NULL));
35. // 启动定时器，ID为1，定时时间为200ms
36. SetTimer(1, 200, NULL);
38. return TRUE;  // return TRUE  unless you set the focus to a control
39. }

       8、为CExample50Dlg类添加波形绘制的成员函数CExample50Dlg::DrawWave(CDC \*pDC, CRect &rectPicture)，参数分别为设备上下文指针和绘图的矩形区域。

**C++代码**

1. void CExample50Dlg::DrawWave(CDC \*pDC, CRect &rectPicture)
2. {
3. float fDeltaX;     // x轴相邻两个绘图点的坐标距离
4. float fDeltaY;     // y轴每个逻辑单位对应的坐标值
5. int nX;      // 在连线时用于存储绘图点的横坐标
6. int nY;      // 在连线时用于存储绘图点的纵坐标
7. CPen newPen;       // 用于创建新画笔
8. CPen \*pOldPen;     // 用于存放旧画笔
9. CBrush newBrush;   // 用于创建新画刷
10. CBrush \*pOldBrush; // 用于存放旧画刷
12. // 计算fDeltaX和fDeltaY
13. fDeltaX = (float)rectPicture.Width() / (POINT\_COUNT - 1);
14. fDeltaY = (float)rectPicture.Height() / 80;
16. // 创建黑色新画刷
17. newBrush.CreateSolidBrush(RGB(0,0,0));
18. // 选择新画刷，并将旧画刷的指针保存到pOldBrush
19. pOldBrush = pDC->SelectObject(&newBrush);
20. // 以黑色画刷为绘图控件填充黑色，形成黑色背景
21. pDC->Rectangle(rectPicture);
22. // 恢复旧画刷
23. pDC->SelectObject(pOldBrush);
24. // 删除新画刷
25. newBrush.DeleteObject();
27. // 创建实心画笔，粗度为1，颜色为绿色
28. newPen.CreatePen(PS\_SOLID, 1, RGB(0,255,0));
29. // 选择新画笔，并将旧画笔的指针保存到pOldPen
30. pOldPen = pDC->SelectObject(&newPen);
32. // 将当前点移动到绘图控件窗口的左下角，以此为波形的起始点
33. pDC->MoveTo(rectPicture.left, rectPicture.bottom);
34. // 计算m\_nzValues数组中每个点对应的坐标位置，并依次连接，最终形成曲线
35. for (int i=0; i<POINT\_COUNT; i++)
36. {
37. nX = rectPicture.left + (int)(i \* fDeltaX);
38. nY = rectPicture.bottom - (int)(m\_nzValues[i] \* fDeltaY);
39. pDC->LineTo(nX, nY);
40. }
42. // 恢复旧画笔
43. pDC->SelectObject(pOldPen);
44. // 删除新画笔
45. newPen.DeleteObject();
46. }

       9、有了定时器和绘图成员函数，我们就可以在WM\_TIMER消息的响应函数中添加对波形数据的定时处理和对波形的定时绘制了。定时器及WM\_TIMER消息处理函数的添加方法如果忘记了，可以再到[VS2010/MFC编程入门之四十四（MFC常用类：定时器Timer）](http://www.jizhuomi.com/software/232.html)温习下。

       WM\_TIMER消息的处理函数修改如下：

**C++代码**

1. void CExample50Dlg::OnTimer(UINT\_PTR nIDEvent)
2. {
3. // TODO: Add your message handler code here and/or call default
4. CRect rectPicture;
6. // 将数组中的所有元素前移一个单位，第一个元素丢弃
7. for (int i=0; i<POINT\_COUNT-1; i++)
8. {
9. m\_nzValues[i] = m\_nzValues[i+1];
10. }
11. // 为最后一个元素赋一个80以内的随机数值（整型）
12. m\_nzValues[POINT\_COUNT-1] = rand() % 80;
14. // 获取绘图控件的客户区坐标
15. // （客户区坐标以窗口的左上角为原点，这区别于以屏幕左上角为原点的屏幕坐标）
16. m\_picDraw.GetClientRect(&rectPicture);
18. // 绘制波形图
19. DrawWave(m\_picDraw.GetDC(), rectPicture);
21. CDialogEx::OnTimer(nIDEvent);
22. }

       10、在对话框销毁时，定时器应关闭。所以为CExample50Dlg类添加WM\_DESTROY消息的处理函数，并修改如下：

**C++代码**

1. void CExample50Dlg::OnDestroy()
2. {
3. CDialogEx::OnDestroy();
5. // TODO: Add your message handler code here
7. // 关闭定时器
8. KillTimer(1);
9. }

       11、一切准备就绪，编译运行。最终的效果如下图：

       关于画笔，鸡啄米就讲到这里了，下一节将为大家简单讲讲画刷的使用。

# VS2010/MFC编程入门之五十一（图形图像：GDI对象之画刷CBrush）

       上一节中鸡啄米主要讲的是[画笔CPen](http://www.jizhuomi.com/software/246.html)的用法，前面也说了，GDI对象中最常用的就是画笔和画刷，本节就讲讲画刷CBrush。

       鸡啄米依然是通过实例的方式来说明画刷的用法。此实例要实现的功能是，对话框上有一个按钮控件和一个[图片控件](http://www.jizhuomi.com/software/193.html)，点击按钮弹出[颜色对话框](http://www.jizhuomi.com/software/177.html)，然后在颜色对话框中选择颜色并点击“确定”后，图片控件中将显示选择的颜色。

       其实此实例的功能，还可以通过重载对话框WM\_CTLCOLOR消息的响应函数等方法来实现，但为讲解画刷的CBrush的使用，鸡啄米采用了下面代码中的方法。

**以下是此实例的具体实施步骤：**

       1、创建一个基于对话框的[MFC](http://www.jizhuomi.com/catalog.asp?tags=MFC)工程，名字设为“Example51”。

       2、在自动生成的对话框模板IDD\_EXAMPLE51\_DIALOG中，删除“TODO: Place dialog controls here.”[静态文本框](http://www.jizhuomi.com/software/179.html)，添加一个按钮控件和一个Picture控件，ID分别设为IDC\_COLOR\_SEL\_BUTTON和IDC\_COLOR\_SHOW\_STATIC，按钮控件的Caption属性设为“选择颜色”，此时对话框模板如下图所示：

       3、为Picture控件IDC\_COLOR\_SHOW\_STATIC添加CStatic变量，名称设为m\_picColor。

       4、在对话框模板中双击“选择颜色”按钮，为其添加点击消息的响应函数CExample51Dlg::OnBnClickedColorSelButton()，修改该函数的实现如下：

**C++代码**

1. void CExample51Dlg::OnBnClickedColorSelButton()
2. {
3. // TODO: Add your control notification handler code here
4. COLORREF color = RGB(255, 0, 0);    // 颜色对话框的初始颜色
5. CColorDialog colorDlg(color);       // 构造颜色对话框，初始颜色为红色
6. CRect rectPicture;       // 图片控件的矩形区域坐标
7. CBrush newBrush;         // 创建的新画刷
8. CBrush \*pOldBrush;       // 旧画刷的指针
9. CClientDC clientDC(this); // 构造客户区的设备上下文对象
11. if (IDOK == colorDlg.DoModal())    // 显示颜色对话框
12. {
13. // 如果点击了颜色对话框的“确定”按钮，则执行以下操作
15. // 获取颜色对话框中选择的颜色
16. color = colorDlg.GetColor();
17. // 用选择的颜色创建新画刷
18. newBrush.CreateSolidBrush(color);
20. // 获取图片控件矩形区域的屏幕坐标
21. m\_picColor.GetWindowRect(&rectPicture);
22. // 将图片控件矩形区域的屏幕坐标转换为其父窗口即对话框的客户区坐标
23. ScreenToClient(&rectPicture);
25. // 选择新画刷，并保存旧画刷的指针到pOldBrush
26. pOldBrush = clientDC.SelectObject(&newBrush);
27. // 以新画刷为图片控件填充颜色
28. clientDC.Rectangle(rectPicture);
30. // 恢复旧画刷
31. clientDC.SelectObject(pOldBrush);
32. // 删除新画刷
33. newBrush.DeleteObject();
34. }
35. }

       代码中已经加了详细的注释，不过鸡啄米觉得有必要再简单介绍下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。

       讲到这里，有人会问，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。

       5、编译运行程序，在结果对话框中，点击“选择颜色”按钮弹出颜色对话框，并选择颜色后，效果如下图：

       本节教程除讲了画刷CBrush的使用外，还让大家简单温习了颜色对话框的用法。

       鸡啄米提醒大家，如果有什么函数或类的用法不是很清楚，均可以查阅MSDN。

# VS2010/MFC编程入门之五十二（Ribbon界面开发：创建Ribbon样式的应用程序框架）

      上一节中鸡啄米讲了[GDI对象之画刷CBrush](http://www.jizhuomi.com/software/248.html)，至此图形图像的入门知识就讲完了。从本节开始鸡啄米将为大家带来Ribbon界面开发的有关内容。本文先来说说如何创建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。

       鸡啄米将以图文结合的方式来说明利用MFC向导创建Ribbon样式的单文档应用程序框架的过程，其实Ribbon应用程序框架的创建有很多地方与普通单文档应用程序类似，所以如果有些步骤的选项不太明白，可以参见[VS2010/MFC编程入门之二（利用MFC向导生成单文档应用程序框架）](http://www.jizhuomi.com/software/141.html)。下面是具体步骤：

       1、打开[VS2010](http://www.jizhuomi.com/catalog.asp?tags=VS2010)，在菜单中选择File->New->Project，弹出标题为“New Project”的对话框。在此对话框左侧面板中选择Installed Templates->Visual C++->MFC，然后在中间区域中选择“MFC Application”。

       选择了工程类型后，可以看到对话框下边有三个设置项，分别是Name--工程名、Location--解决方案路径、Solution Name--解决方案名称。鸡啄米在这里将Name设为“Example52”，Location设为“桌面”的路径，Solution Name默认与Name一样，我们这里不作修改。如下图：

       点“OK”按钮。

       2、这时会弹出“MFC Application Wizard”对话框，上部写有“Welcome to the MFC Application Wizard”，下面显示了当前工程的默认设置。第一条“Tabbed multiple document interface (MDI)”是说此工程是多文档应用程序。如果这时直接点下面的“Finish”按钮，可生成具有上面列出设置的多文档程序。但我们此例是要建立Ribbon样式的单文档应用程序，所以点“Next”按钮再继续设置吧。

       3、接下来弹出的对话框上部写有“Application Type”，当然是让选择应用程序类型，我们选择“Single document”，说明要创建的是单文档应用程序框架。另外，在“Project Type”下选择“Office”，表示此应用程序为Office风格。如下图：

       点“Next”按钮。

       4、弹出上部写有“Compound Document Support”的对话框，可以通过它向应用程序加入OLE支持，这里使用默认值“None”。点“Next”按钮。

       5、弹出的新对话框上部写有“Document Template Properties”。这里都使用默认设置，点“Next”按钮。

       6、此时弹出的对话框主题是“Database Support”。用于设置数据库选项。依然使用默认值，点“Next”。

       7、这时弹出的对话框是关于“User Interface Features”，即用户界面特性的。由于我们要使用Ribbon界面，所以要确保“Command bars (menu/toolbar/ribbon)”下的单选按钮“Use a ribbon”选中。其他设置使用默认值。如下图：

       点“Next”按钮。

       8、此时弹出“高级特性”对话框。可以设置的高级特性包括有无打印和打印预览等。保持默认值不变，点“Next”。

       9、弹出最后一个“Generated Classes”（生成类）向导对话框，列出了要生成的四个类。这里不作修改，点“Finish”。

       这样我们就完成了创建Ribbon样式的应用程序框架的全部设置。编译运行程序，得到如下的的Ribbon界面（Office 2007(Blue Style)）：

       在Ribbon界面的右上角位置有个“Style”下拉菜单，我们可以选择不同的Style，在Office 2007 (Blue Style)、Office 2007 (Black Style)、Office 2007 (Silver Style)、Office 2007 (Aqua Style)、Windows 7等5中风格之间切换。

       关于Ribbon样式的应用程序框架的创建鸡啄米就讲到这里了，有了以前的基础，这些操作可以说都是小菜一碟。

# VS2010/MFC编程入门之五十三（Ribbon界面开发：为Ribbon Bar添加控件）

       前面一节中鸡啄米为大家简单介绍了如何[创建Ribbon样式的应用程序框架](http://www.jizhuomi.com/software/251.html)，本节教程就来初步讲讲怎样为Ribbon Bar添加Ribbon控件。

[VS2010](http://www.jizhuomi.com/catalog.asp?tags=VS2010)为Ribbon界面开发提供了Ribbon Designer，通过它我们可以为Ribbon Bar添加各种Ribbon控件、设置控件属性和进行界面布局。

**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。**

       鸡啄米以实例的方式讲解Ribbon控件的添加方法和属性，这样比较直观。此实例还是在上一节中创建的Example52工程的基础上进行修改。

       1、打开Example52工程，在资源视图Resource View中，展开Example52->Example52.rc->Ribbon，在Ribbon节点下，我们看到有一个系统自动生成的IDR\_RIBBON，双击IDR\_RIBBON节点，就可以在中间区域打开Ribbon Designer，如下图：

       上图中，Home标签下的整个界面就是类别，Clipboard和View对应的就是面板，每个面板都有一些按钮、复选框等基本控件。

       2、我们要向Ribbon界面中添加控件的话，也需要从Toolbox给出的Ribbon控件列表中选择控件拖入Ribbon bar。点击View菜单下的Toolbox，就会显示出Toolbox视图。下图就是Ribbon Designer的Toolbox：

       虽然与以前我们用的Toolbox有些不同，但还是很相似的。上图的Toolbox中的Category就是类别，Panel就是面板，其他的大部分都是基本控件。

       3、在Toolbox中选择Category拖入Ribbon bar，放到Home类别的后面，可以看到它的默认名称为“Category1”，并且默认带了一个面板“Panel1”。这里我们选择“Category1”标签，右键点击“Properties”，在显示出的属性页中，修改Caption属性为“Function”。然后选择“Panel1”面板，以同样的方法修改其Caption属性为“Big Button”。

       4、接下来我们再为“Function”类别添加一个新面板。在Toolbox中选择Panel拖到“Function”类别下，放到“Big Button”面板后面，Caption属性修改为“Small Button”。

       5、再往“Big Button”面板中添加一个按钮控件。在Toolbox中选择Button拖入“Big Button”面板中，Caption修改为“Open”。以同样的方式往“Small Button”面板中添加两个按钮控件，Caption分别修改为“Click”和“Check”。此时的Ribbon bar如下图所示：

       6、通过与Home下的按钮对比我们发现，新添加的几个按钮不太美观，因为没有加图片。那么怎样为按钮加图片呢？

       观察发现，Home下的按钮的图标有两种：大图标和小图标。**实际上大图标是像素为32×32的图标，小图标是像素为16×16的图标。**鸡啄米为三个按钮制作了两套图像序列，每个图像序列都由三个图标组成，第一个图像序列由三个大图标组成，第二个图像序列由三个小图标组成。如下面的两个图：

大图标(newicons-32.bmp)

小图标(newicons-16.bmp)

**要得到这样的图像序列，可以先找到每个图标，然后使用IconWorkshop等工具制作成图像序列即可。**

       将两个图片都复制到目录...\Example52\Example52\res下，然后在工程中Resource View资源视图的Example52.rc->Bitmap上点右键，选择“Add Resource”，弹出Add Resource对话框，在Add Resource对话框左侧的树中选择“Bitmap”，然后点击“Import”按钮，选择newicons-32.bmp文件后就成功导入了大图标文件，ID默认为IDB\_BITMAP1。以同样的方式导入小图标文件newicons-16.bmp，ID默认为IDB\_BITMAP2。

       在“Function”类别的属性页中可以看到有Large Images和Small Images两个属性，它们就是用来设置本类别下控件所要使用的大图标序列和小图标序列的。这里我们将Large Images属性设为IDB\_BITMAP1，Small Images属性设为IDB\_BITMAP2。

       然后我们为Open按钮设置图片。在Open按钮的属性页中有Image Index和Large Image Index两个属性，分别是其小图标在小图标序列中的索引和其大图标在大图标序列中的索引，这里我们设置其图标为大图标，且为大图标序列中的第一个，**那么直接设置Large Image Index属性为0，也可以通过在选择Large Image Index属性的编辑框后出现的浏览按钮上点击，弹出Image Collection对话框来选择图标。**

       再为Click按钮和Check按钮设置小图标。将Click按钮的Image Index属性设置为1，Large Image Index属性仍为-1，Check按钮的Image Index属性设置为2，Large Image Index属性也保持为-1。

       7、编译运行程序，最终界面的Function类别视图如下：

       因为上述三个按钮都没有添加任何事件的响应函数，所以都是灰色的。

       这一节就讲到这里了。大家可以试着添加其他控件看看效果

# VS2010/MFC编程入门之五十四（Ribbon界面开发：使用更多控件并为控件添加消息处理函数）

       上一节中鸡啄米讲了[为Ribbon Bar添加控件](http://www.jizhuomi.com/software/253.html)的方法。本节教程鸡啄米将继续完善前面的实例，讲解一些稍复杂的控件的添加方法，及如何为它们添加消息处理函数。

**一、为Ribbon Bar添加更多Ribbon控件**

       鸡啄米将在上一节实例的基础上，继续添加下拉菜单、Check Box、Combo Box等Ribbon控件。

       1、首先把“Small Button”面板上的“Click”按钮改造成一个下拉菜单。“Click”按钮有一个Behavior属性Menu Items，默认为Empty，选中它右侧会出现一个浏览按钮，点击浏览按钮会弹出“Items Editor”对话框，如下图：

       我们可以在上图Items下的组合框中选择按钮、分割线等，点击组合框右侧的Add按钮将其添加到下拉菜单中，添加按钮后在Properties分组中可以设置Caption（标题）、ID、Image（图片）等属性。鸡啄米这里添加两个按钮，Caption属性分别为One Click、Double Click，ID分别为ID\_ONE\_CLICK、ID\_DOUBLE\_CLICK，Image等属性就不设置了。此时的Ribbon Bar如下图：

       “Click”右侧多了一个向下的箭头，运行程序后点击此箭头会显示包含One Click和Double Click按钮的下拉菜单。另外，上图中有一个按钮鸡啄米用红线指示了其提示信息-“Test Ribbon”，点击了此按钮我们就可以不运行程序而直接查看Ribbon界面效果。

        2、在Small Button面板的右侧再添加一个面板“More Controls”，然后在Toolbox工具中找到Check Box和Combo Box控件拖入新面板，Check Box的Caption属性设为“Websites Enable”，Combo Box的属性设为“Websites”。效果图如下：

       我们为Websites组合框（Combo Box）添加两个下拉选项，方法是，右键点击Websites组合框，选择“Properties”，显示出其属性页，修改Data属性为“www.jizhuomi.com;www.jizhuomi.com/android”，这样就为此Combo Box添加两个网址选项。

**二、为Ribbon控件添加消息处理函数**

       前面控件都添加好了，接下来我们就为控件添加消息处理函数。

       1、首先为Open按钮添加单击事件的消息处理函数，其ID修改为ID\_OPEN\_BUTTON，然后右键点击Open按钮，选择“Add Event Handler”，弹出Event Handler Wizard对话框，右侧的Class list中选择“CMainFrame”，左侧的Message Type中选择“COMMAND”，最后点击“Add and Edit”按钮，CMainFrame类中就添加了void CMainFrame::OnOpenButton()成员函数。

       大家可能感觉到了，其实消息处理函数的添加过程与以前的普通控件是类似的。最后修改void CMainFrame::OnOpenButton()函数的函数实现如下：

**C++代码**

1. void CMainFrame::OnOpenButton()
2. {
3. // TODO: Add your command handler code here
4. MessageBox(\_T("Open Button!"));   // 弹出对话框，提示“Open Button!”
5. }

       因为只是为了讲解按钮的消息处理函数的添加，所以没有写复杂的代码，只写了一个弹出MessageBox的语句。运行程序，在结果界面的Function类别的Big Button面板中，点击Open按钮就会弹出一个对话框，并显示“Open Button!”。

       2、我们再为Websites组合框添加消息处理函数（方法同上），同样也为其在CMainFrame类中添加COMMAND消息处理函数--void CMainFrame::OnWebsitesCombo()，修改此函数实现如下：

**C++代码**

1. void CMainFrame::OnWebsitesCombo()
2. {
3. // TODO: Add your command handler code here
4. // 获取Combo Box控件的指针
5. CMFCRibbonComboBox\* pComboBox = DYNAMIC\_DOWNCAST(CMFCRibbonComboBox, m\_wndRibbonBar.FindByID(ID\_WEBSITES\_COMBO));
6. // 获取Combo Box控件当前选中项的索引
7. int nCurSel = pComboBox->GetCurSel();
9. if (nCurSel >= 0)
10. {
11. // 如果索引大于等于0，则说明有选中项，弹出对话框并显示选中项的信息
12. MessageBox(pComboBox->GetItem(nCurSel));
13. }
14. else
15. {
16. // 如果索引小于0，则说明没有选中项，弹出对话框提示用户进行选择
17. MessageBox(\_T("Please select one item!"));
18. }
19. }

       运行程序，在结果界面中，改变WebSites组合框的选中项，则会弹出对话框显示选中项的信息。如下图：

       3、然后我们为Websites Enable复选框添加消息处理函数。默认情况下Check Box控件不会因为用户的点击而改变状态，这就需要我们通过代码来实现正常的复选功能。

       我们需要一个变量保存Check Box的当前选中状态，所以在MainFrm.h文件中为CMainFrame类添加一个BOOL型的成员变量m\_bWebsitesEnable，并在CMainFrame类的构造函数中为其初始化：

**C++代码**

1. CMainFrame::CMainFrame()
2. {
3. // TODO: add member initialization code here
4. theApp.m\_nAppLook = theApp.GetInt(\_T("ApplicationLook"), ID\_VIEW\_APPLOOK\_OFF\_2007\_BLUE);
5. // 初始化为TRUE，即复选框为选中状态
6. m\_bWebsitesEnable = TRUE;
7. }

       接下来仍然采用1中的方法为Websites Enable复选框在CMainFrame类中添加COMMAND消息处理函数，并修改其函数体如下：

**C++代码**

1. void CMainFrame::OnWebsitesCheck()
2. {
3. // TODO: Add your command handler code here
4. // 为m\_bWebsitesEnable取反，即切换复选框的状态
5. m\_bWebsitesEnable = !m\_bWebsitesEnable;
6. }

       但是现在复选框的状态只是保存到了变量中，我们还要根据变量值改变复选框的显示状态，这就需要再为Check Box添加一个UPDATE\_COMMAND\_UI消息处理函数，方法仍旧是右键点击Check Box，选择“Add Event Handler”，在弹出的Event Handler Wizard对话框的Class list中选择“CMainFrame”，Message type中则选择UPDATE\_COMMAND\_UI，最后点“Add and Edit”，这样就添加了UPDATE\_COMMAND\_UI消息处理函数，修改此函数实现如下：

**C++代码**

1. void CMainFrame::OnUpdateWebsitesCheck(CCmdUI \*pCmdUI)
2. {
3. // TODO: Add your command update UI handler code here
4. // 根据当前变量值设置复选框状态
5. pCmdUI->SetCheck(m\_bWebsitesEnable);
6. }

       这时你可以试着运行下程序，点击Website Enable复选框，它已经能成功的改变状态了。

       4、除了以上功能，我们还要实现一个稍复杂的功能，就是如果选中Websites Enable复选框则激活Websites组合框，而如果取消选中则禁用Websites组合框。这就需要我们为Websites组合框添加UPDATE\_COMMAND\_UI消息处理函数了，添加方法同上，鸡啄米这里不再赘述。修改函数实现为：

**C++代码**

1. void CMainFrame::OnUpdateWebsitesCombo(CCmdUI \*pCmdUI)
2. {
3. // TODO: Add your command update UI handler code here
4. // 根据Websites Enable复选框的状态确实激活还是禁用
5. pCmdUI->Enable(m\_bWebsitesEnable);
6. }

        运行程序，试着改变Websites Enable复选框的状态，Websites组合框的使能状态也会跟着改变。

        最后，鸡啄米再简单讲讲如何为Ribbon Bar左上角的圆形菜单按钮和快速访问工具栏添加新项。

       圆形菜单按钮的属性页中有一个Buttons属性，可以点击其右侧浏览按钮弹出Items Editor对话框，使用该对话框可以在菜单按钮的弹出菜单窗口中添加右下角的按钮。还有一个Main Items属性，点其右侧浏览按钮也会弹出Items Editor对话框，通过它可以为圆形菜单按钮添加菜单项。

       快速访问工具栏的属性页中有一个QAT Items属性，点击其右侧浏览按钮弹出QAT Items Editor对话框，使用该对话框可以在快速访问工具栏中添加新项。