目录

VS2010/MFC 编程入门之前言

鸡啄米的 C++编程入门系列给大家讲了 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。

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

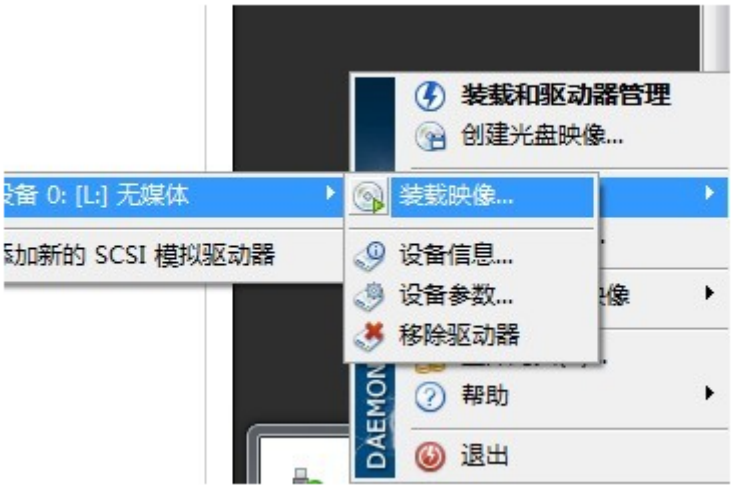
首先我们需要下载 VS2010，大家可以在网上下载 VS2010 破解正式版，建议选择英文版，养成使用英文工具的习惯。鸡啄米使用 VS2010 旗舰试用版 VS2010UltimTrial.iso 为例介绍安装过程，旗舰试用版官方下载地址为：
http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=12187。正式版的安装过程与试用版类似。

二. 安装 VS2010

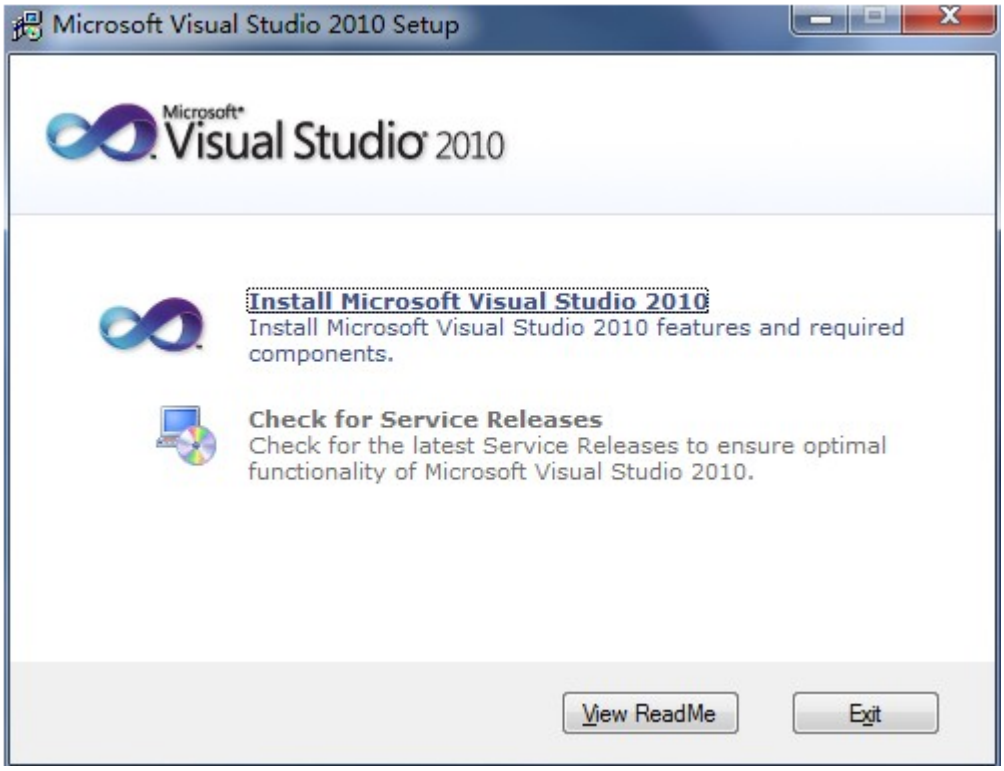
下载后进行安装。安装方法与一般的 iso 文件一样，可以使用虚拟光驱软件 Daemon Tools 安装，也可以将其解压后点击 setup.exe 进行安装。

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。



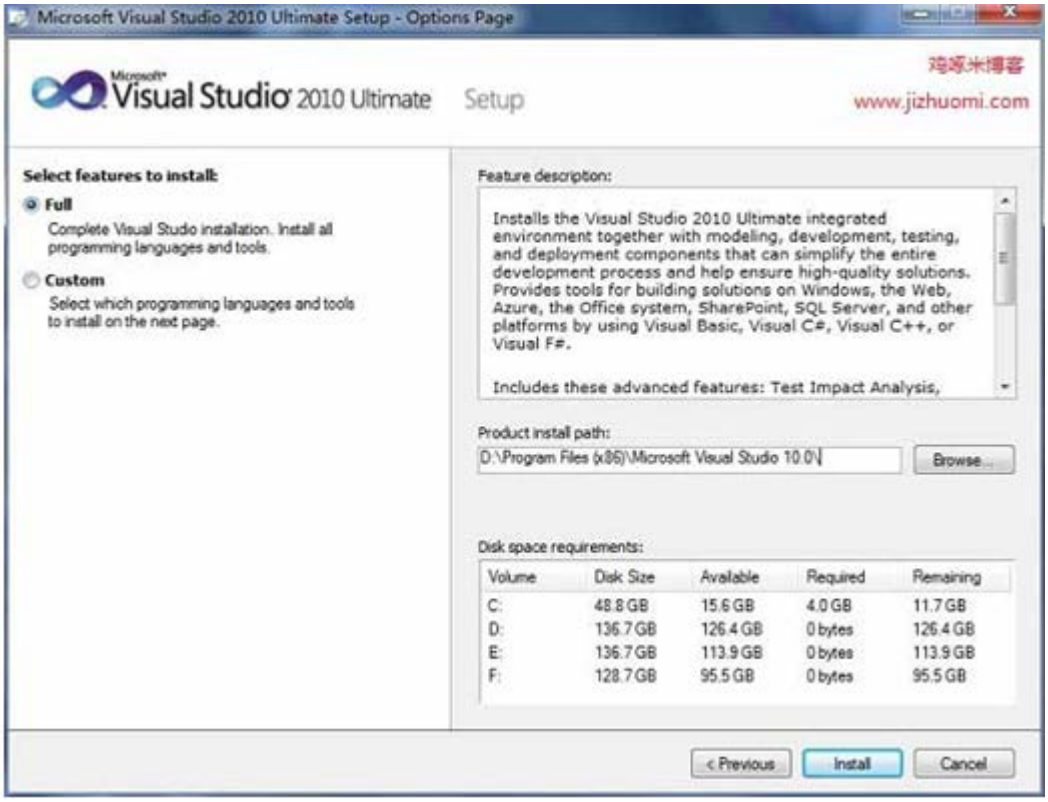
这样虚拟光驱就会打开此 iso 文件，弹出自动安装的提示，选择“运行 autorun.exe”就可以了，如果没有弹出提示就通过资源管理器进入虚拟光驱，用 setup.exe 安装。接着会弹出下面的对话框：



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：

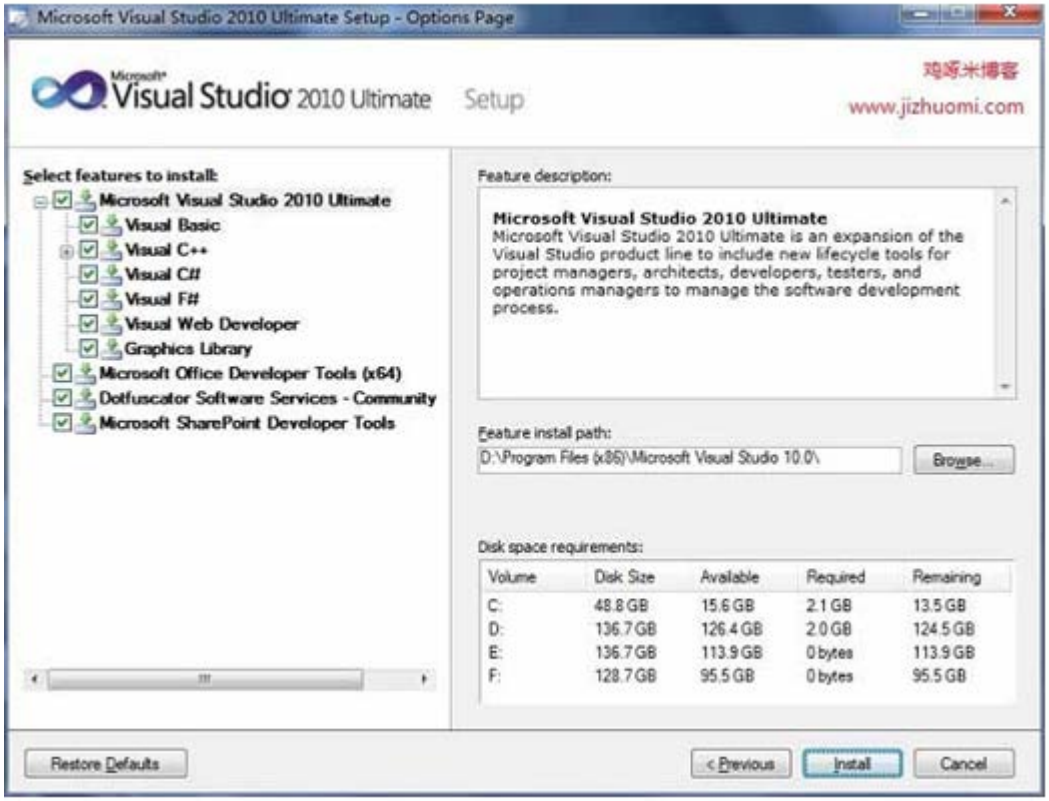
点“Next”后：

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



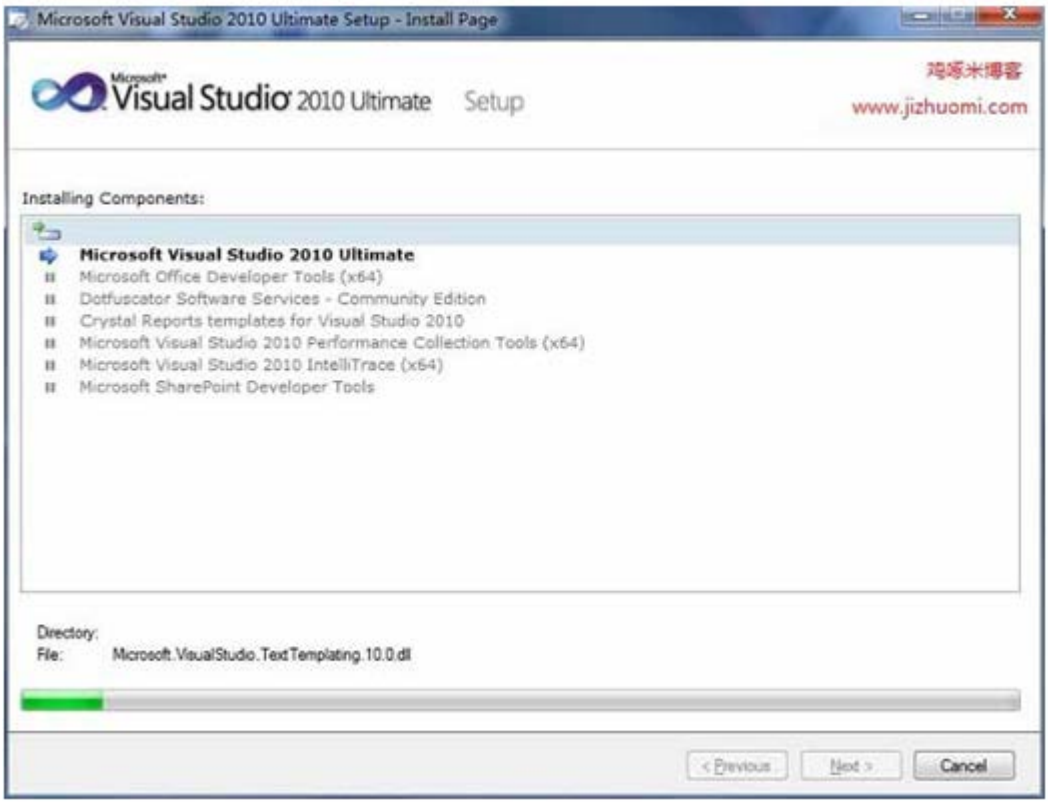
此处是让我们选择要安装的功能，有两种：Full（完全）和 Custom（自定义）。Full 选项表示安装所有编程语言和工具，Custom 选择表示可以自定义要安装的编程语言和工具。右侧可以更改安

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：

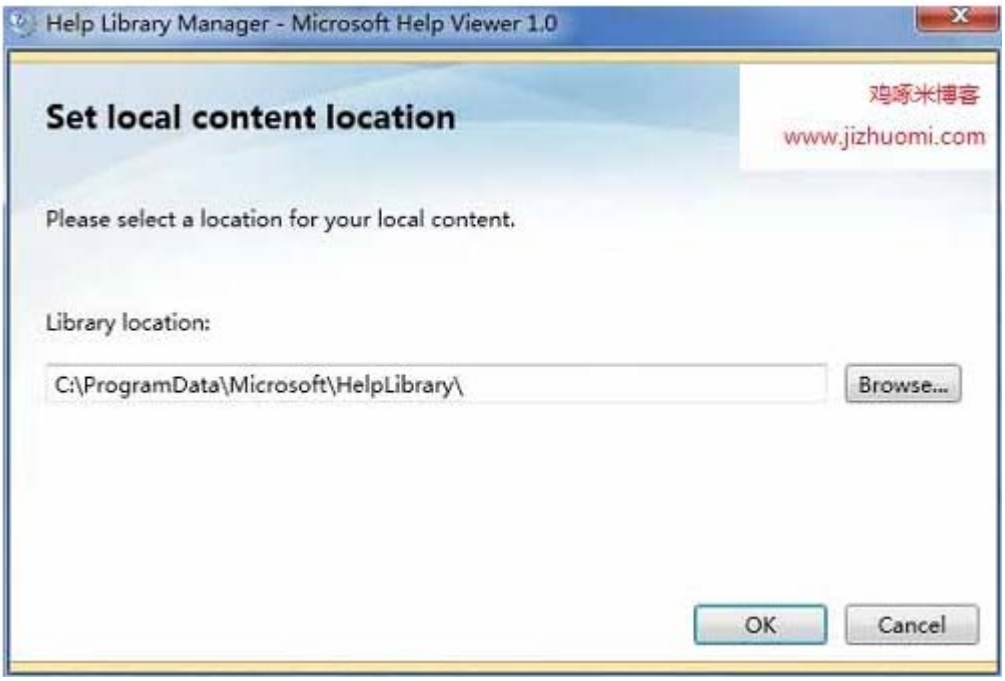
如果要继续安装 MSDN，先不要卸载虚拟光驱映像。

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，全部添加。

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。

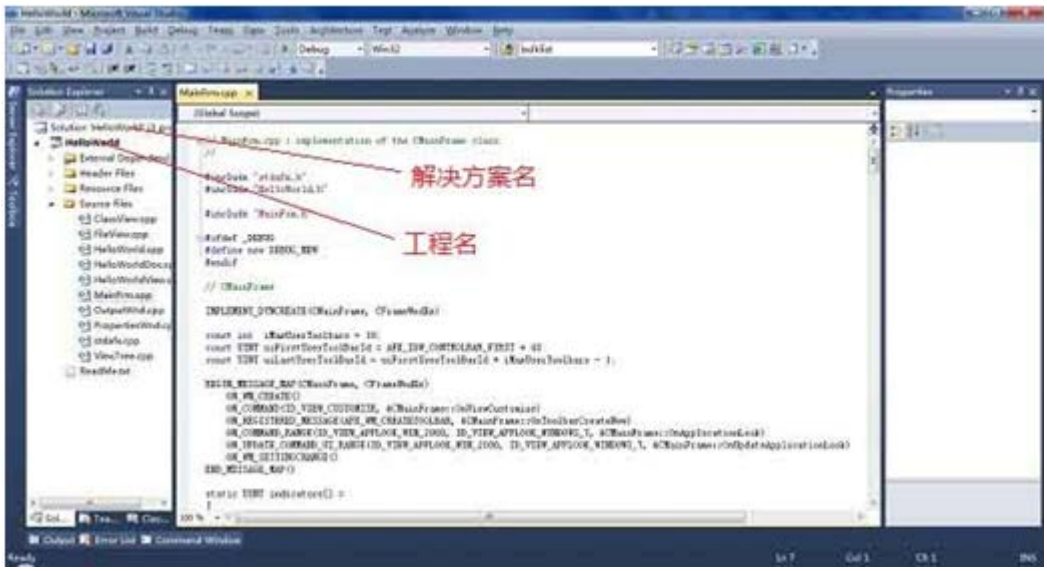
VS2010/MFC编程入门之二（利用MFC向导生成单文档应用程序框架）

上一讲中讲了VS2010和MSDN如何安装，相信大家都已经安装好了。这一讲给大家一个简单的例子，演示如何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。

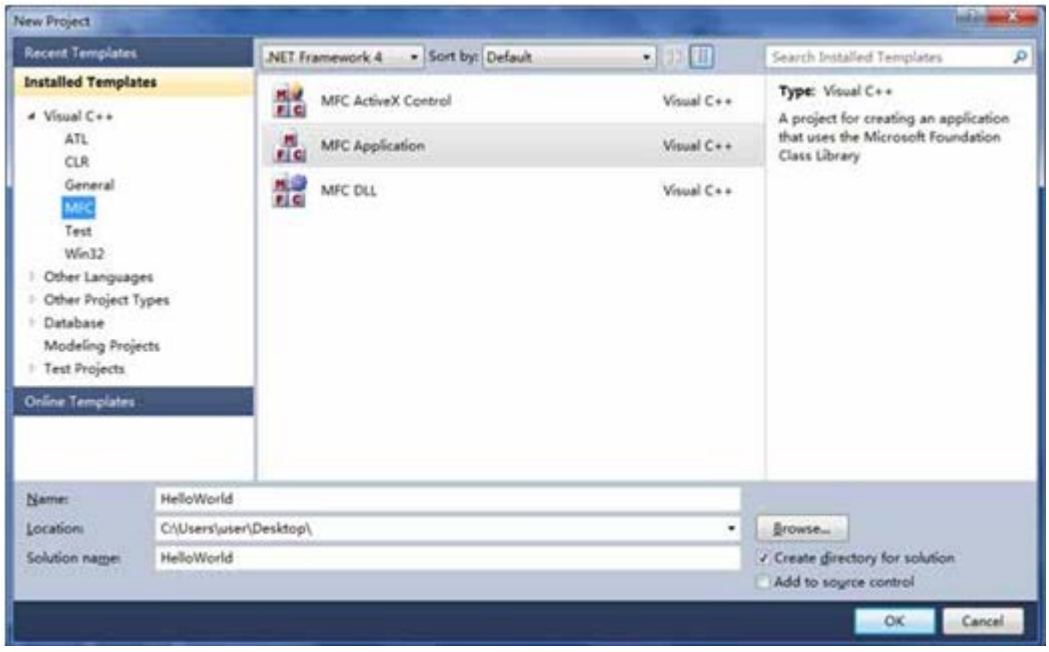
使用 VS2010 应用程序向导生成单文档应用程序框架

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。

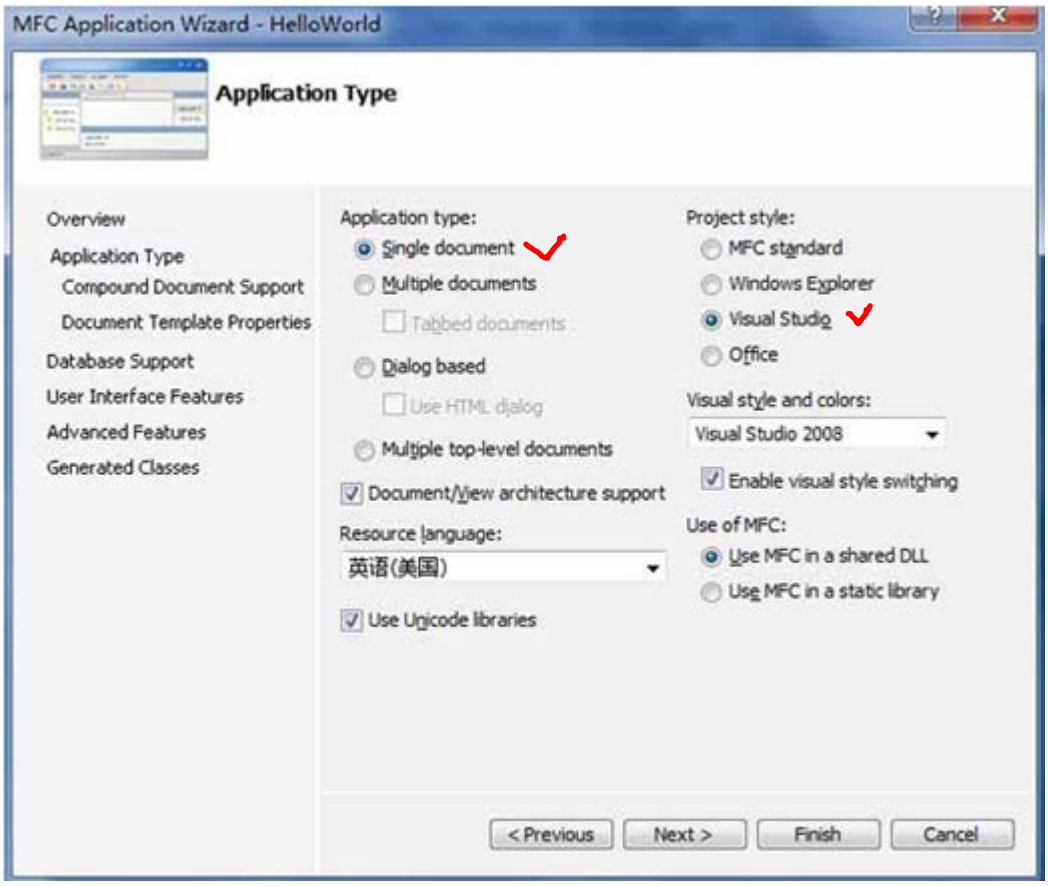
如果安装完 VS2010 以后第一启动时已经设置为 VC++，则 Installed Templates->Visual C++项会默认展开，而如果没有设置 VC++，则可以展开到 Installed Templates->Other Languages->Visual C++项。因为我们要生成的是 MFC 程序，所以在“Visual C++”下选择“MFC”，对话框中间区域会出现三个选项：MFC ActiveX Control、MFC Application 和 MFC DLL。MFC ActiveX Control 用来生成 MFC ActiveX 控件程序。MFC Application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

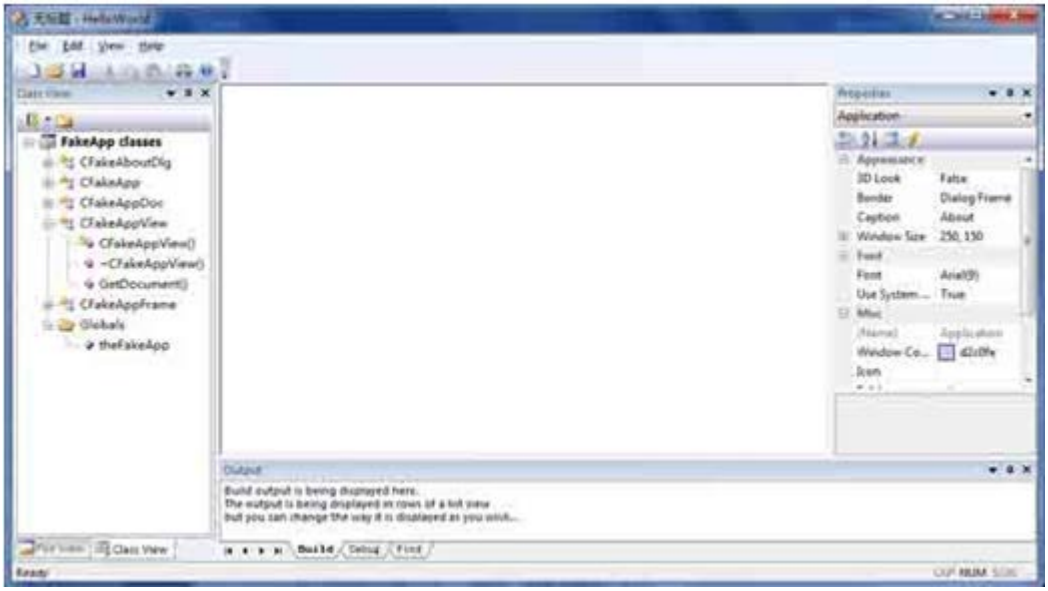
8. 此时弹出“高级特性”对话框。可以设置的高级特性包括有无打印和打印预览等。在“Number of files on recent file list”项可以设置在程序界面的文件菜单下面最近打开文件的个数。我们仍使用默认值。点“Next”按钮。

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：



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。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 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。

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，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。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

```
include <windows>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine, int
iCmdShow)
{
```

```
    const static TCHAR appName[] = TEXT("Hello world");
    WNDCLASSEX myWin;
    myWin.cbSize = sizeof(myWin);
    myWin.style = CS_HREDRAW | CS_VREDRAW;
    myWin.lpfnWndProc = myWndProc;
    myWin.cbClsExtra = 0;
    myWin.cbWndExtra = 0;
    myWin.hInstance = hInstance;
    myWin.hIcon = 0;
    myWin.hIconSm   = 0;
    myWin.hCursor = 0;
    myWin.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    myWin.lpszMenuName = 0;
    myWin.lpszClassName = appName;
    //Register
    if (!RegisterClassEx(&myWin)) return 0;
    const HWND hWindow = CreateWindow(
        appName,
        appName,
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        0,
        0,
        hInstance,
        0);
    ShowWindow(hWindow,iCmdShow);
    UpdateWindow(hWindow);
    {
        MSG msg;
        while(GetMessage(&msg,0,0,0))
        {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
        }
        return (int)msg.wParam;
    }
}

LRESULT CALLBACK myWndProc(HWND hWindow, UINT msg, WPARAM wParam, LPARAM lParam)
{
    if (msg==WM_PAINT)
    {
        PAINTSTRUCT ps;
```

```
        const HDC hDC = BeginPaint(hWindow,&ps);
        RECT rect;
        GetClientRect(hWindow,&rect);
        DrawText(hDC,TEXT("HELLO WORLD"),-1,&rect, DT_SINGLELINE | DT_CENTER |
DT_VCENTER);
        EndPaint(hWindow,&ps);
        return 0;
    }
    else if (msg==WM_DESTROY)
    {
        PostQuitMessage(0);
        return 0;
    }
    return DefWindowProc(hWindow,msg,wParam,lParam);
}
```

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 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

```
extern "C" int WINAPI
_tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
        _In_ LPTSTR lpCmdLine, int nCmdShow)
#pragma warning(suppress: 4985)
{
        // call shared/exported WinMain
        return AfxWinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow);
}
```

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

```
int AFXAPI AfxWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,LPTSTR lpCmdLine, int
nCmdShow)
{
    .............略
```

```
    // App global initializations (rare)
    if (pApp != NULL && !pApp->InitApplication())
       goto InitFailure;

    if (!pThread->InitInstance())
    {
       .........略
    }

    // Run 函数位于 THRDCORE.cpp 中，由此函数进入消息循环
    nReturnCode = pThread->Run();

    ..............略

    return nReturnCode;
}
```

上面 InitInstance 函数的代码如下：

C++代码

```
BOOL CTestApp::InitInstance()
{
    .............略
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
       IDR_MAINFRAME,
       RUNTIME_CLASS(CTestDoc),
       RUNTIME_CLASS(CMainFrame),     // main SDI frame window
       RUNTIME_CLASS(CTestView));
    if (!pDocTemplate)
          return FALSE;
    AddDocTemplate(pDocTemplate);
    // Parse command line for standard shell commands, DDE, file open

    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    //ProcessShellCommand 位于 AppUI2.cpp 中，注册并创建窗口
    if (!ProcessShellCommand(cmdInfo))
          return FALSE;

    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    return TRUE;
}
```

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

```
int CWinThread::Run()
{
     .............略
     // phase2: pump messages while available
     do
     {
       // pump message, but quit on WM_QUIT
       if (!PumpMessage())
          return ExitInstance();

       // reset "no idle" state after pumping "normal" message
       if (IsIdleMessage(&m_msgCur))
       {
          bIdle = TRUE;

          lIdleCount = 0;

       }
    } while (::PeekMessage(&m_msgCur, NULL, NULL, NULL, PM_NOREMOVE));
    ..............略
}

BOOL CWinThread::PumpMessage()
{
    return AfxInternalPumpMessage();
}

BOOL AFXAPI AfxInternalPumpMessage()
{
    _AFX_THREAD_STATE *pState = AfxGetThreadState();

    if (!::GetMessage(&(pState->m_msgCur), NULL, NULL, NULL))
    {
            .............略
    }
    ..............略
    if (pState->m_msgCur.message != WM_KICKIDLE
&& !AfxPreTranslateMessage(&(pState->m_msgCur)))
    {
            ::TranslateMessage(&(pState->m_msgCur));
```

```
            ::DispatchMessage(&(pState->m_msgCur));
        }

        return TRUE;
}
```

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

```
LRESULT CALLBACK AfxWndProc(HWND hWnd,UINT nMsg,WPARAM wParam, LPARAM lParam)
{
     ……
     CWnd*pWnd=CWnd::FromHandlePermanent(hWnd);
     ReturnAfxCallWndProc(pWnd,hWnd,nMsg,wParam,lParam);
}
```

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，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。

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，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

```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWndEx)
        ON_WM_CREATE()
        ON_COMMAND(ID_VIEW_CUSTOMIZE, &CMainFrame::OnViewCustomize)
        ON_REGISTERED_MESSAGE(AFX_WM_CREATETOOLBAR,
&CMainFrame::OnToolbarCreateNew)
        ON_COMMAND_RANGE(ID_VIEW_APPLOOK_WIN_2000, ID_VIEW_APPLOOK_WINDOWS_7,
&CMainFrame::OnApplicationLook)
        ON_UPDATE_COMMAND_UI_RANGE(ID_VIEW_APPLOOK_WIN_2000, ID_VIEW_APPLOOK_WINDOWS_7,
&CMainFrame::OnUpdateApplicationLook)
        ON_WM_SETTINGCHANGE()
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。

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，属于原理方面的知识。对于 VC++编程入门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，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。

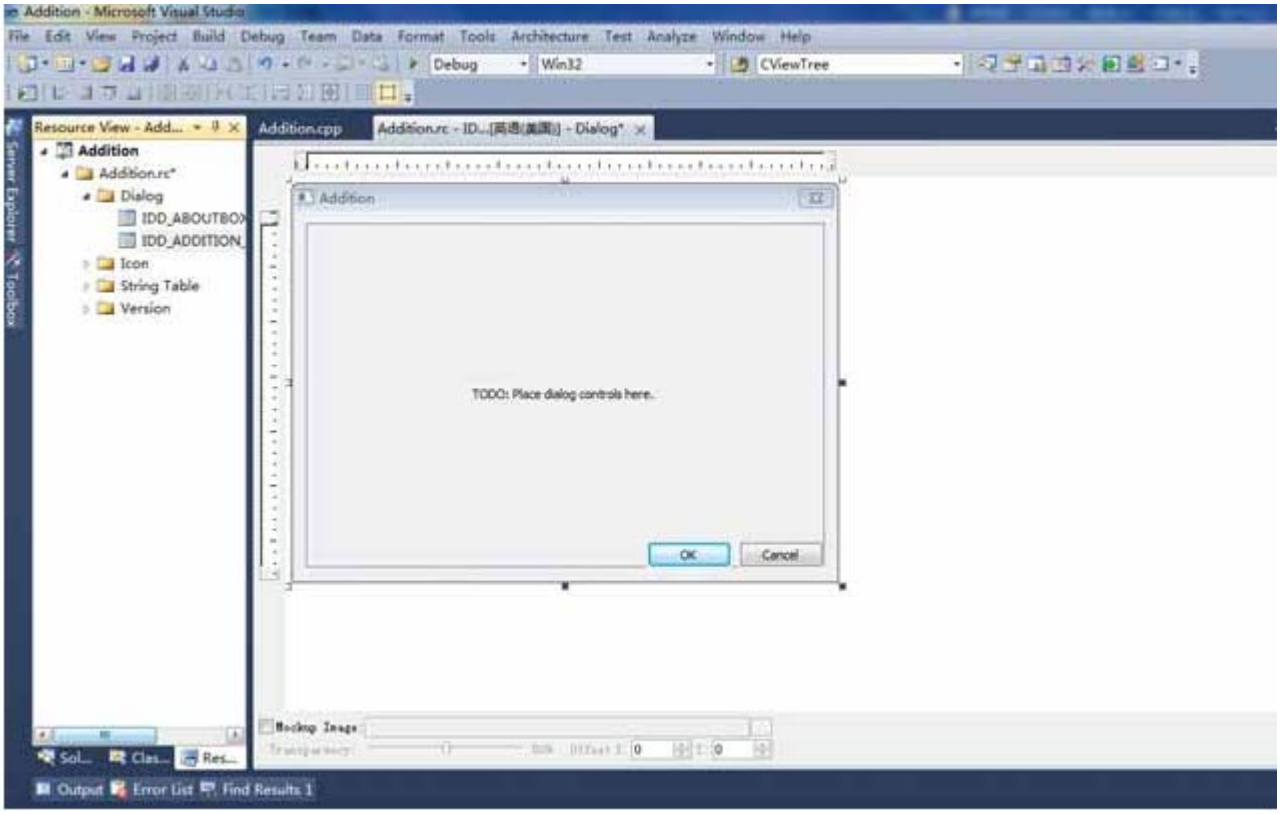
注：如果在 VS2010 中找不到 Solution Explorer 或 Class View 等视图，可以在菜单项 View 下找到对应视图选项选择即可。在 VS2010 的使用介绍中已经有讲解。

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

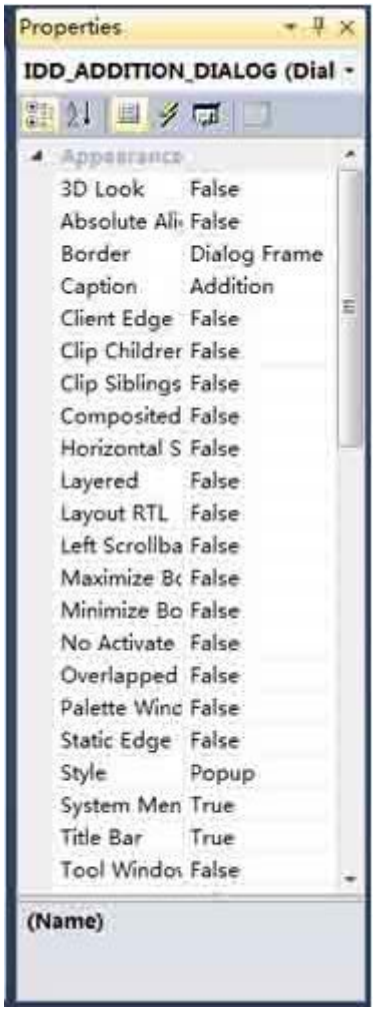
可见对于主对话框来说，创建对话框第一步中的创建新的对话框模板已经由系统自动完成了。而如果是再添加对话框需要创建新的对话框模板时,需要在 Resource View 的“Dialog”节点上点右键，在右键菜单中选择“Insert Dialog”，就会生成新的对话框模板，并且会自动分配 ID。

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。

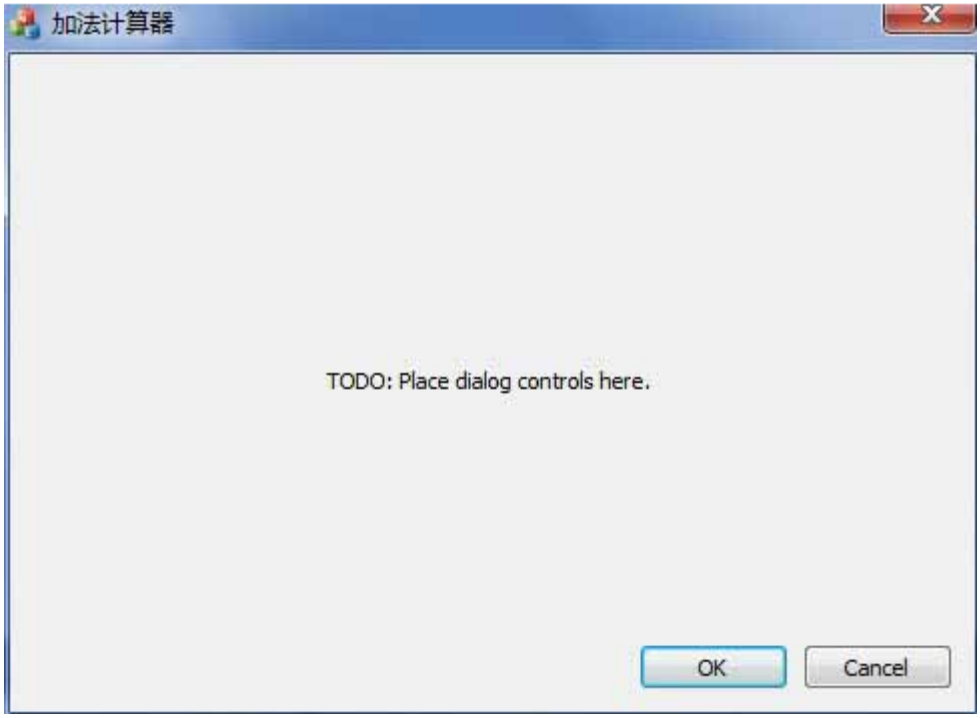
6.Style：对话框类型。有三种类型：Overlapped（重叠窗口）、Popup（弹出式窗口）和 Child（子窗口）。弹出式窗口比较常见。我们使用默认的 Popup 类型。

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：



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，本节继续讲如何为对话框添加控件。

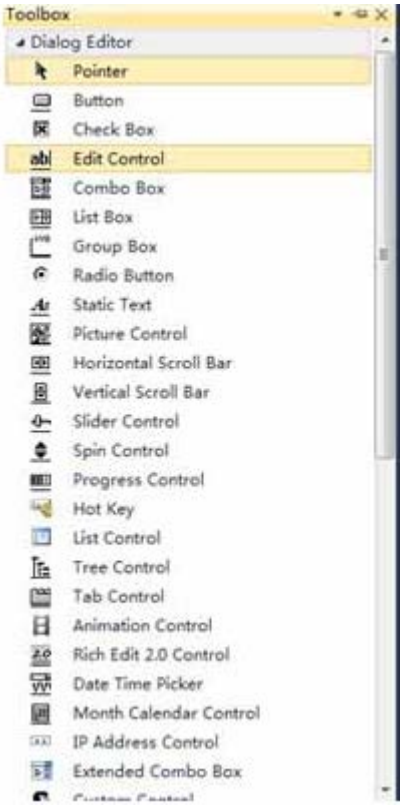
上一讲中鸡啄米创建了一个名为“Addition”的工程，目的是生成一个实现加法运算的应用程序。实现加法计算有几个必要的因素：被加数、加数、和。被加数和加数需要输入，和需要输出显示。那么这几个因素都需要相应的控件来输入或显示，下面鸡啄米就一步步讲解如何添加这些控件。

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。

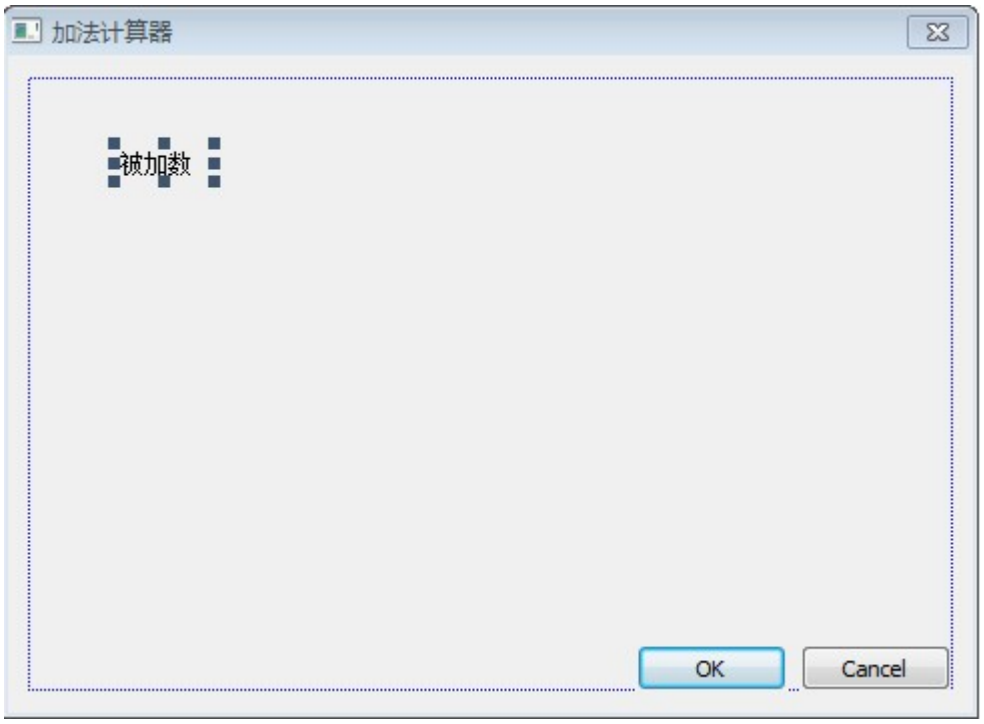
在添加新的静态文本框以前，先看看 Toolbox 视图是否显示了，如果没有显示，在菜单栏上点击 View->Toolbox 即可。Toolbox 视图如下图：



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。

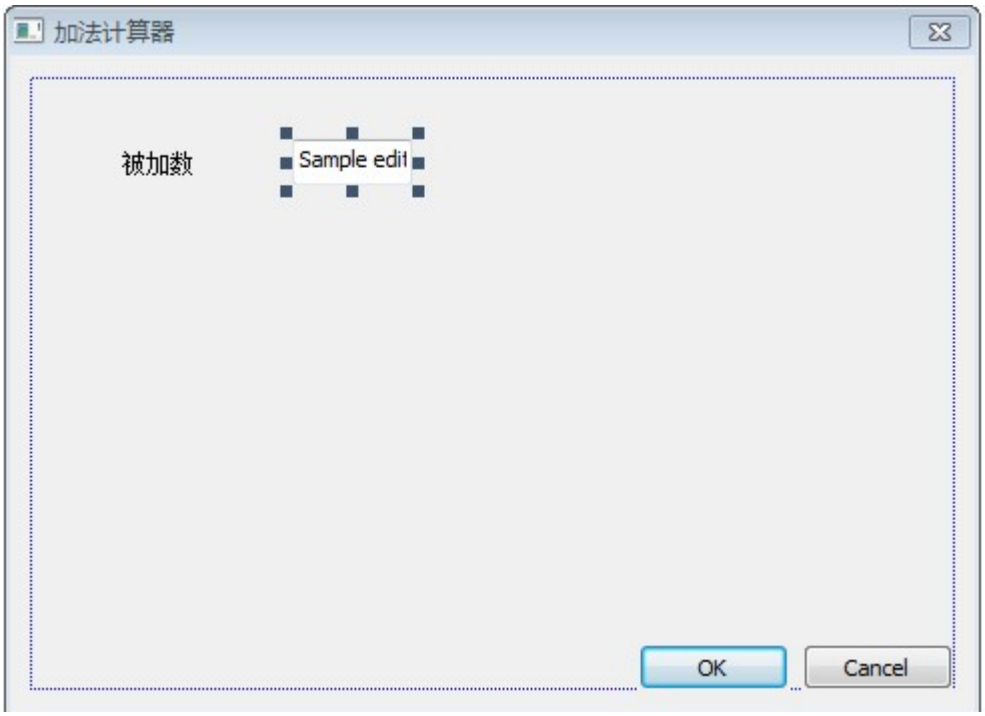
接下来就该修改静态文本框的文字了。鼠标右键点击静态文本框，在右键菜单中选择“Properties”，Properties 面板就会显示出来，在面板上修改 Caption 属性为“被加数”，ID 修改为 IDC_SUMMAND_STATIC。此时模板如下图：



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。

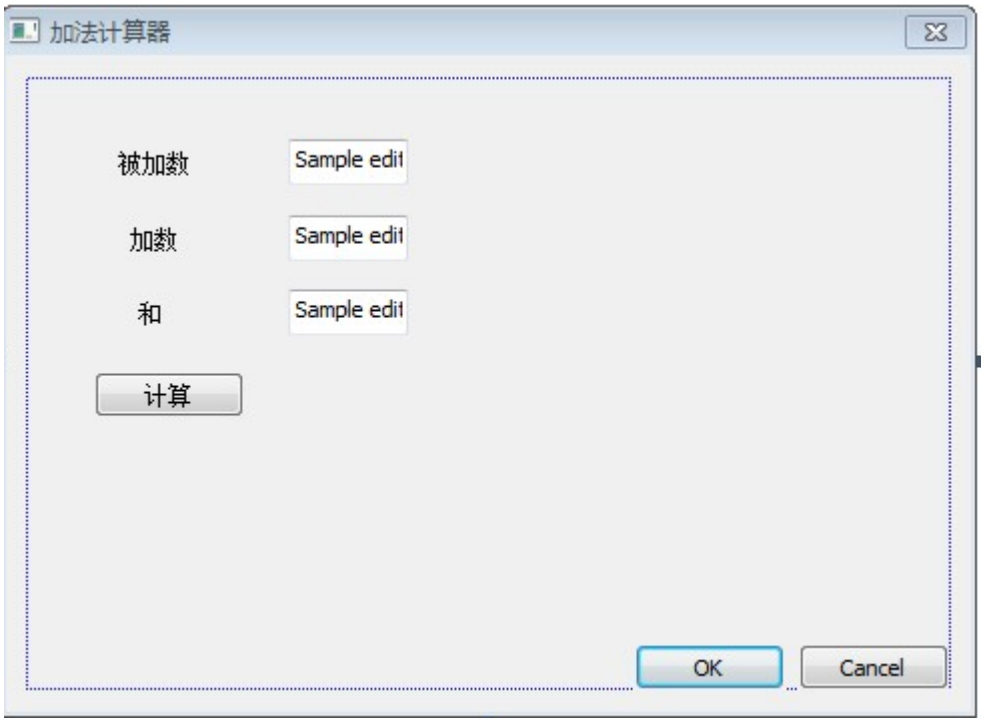
4.按照 2 的方法添加一个 ID 为 IDC_ADDEND_EDIT 的编辑框，用来输入加数。

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。

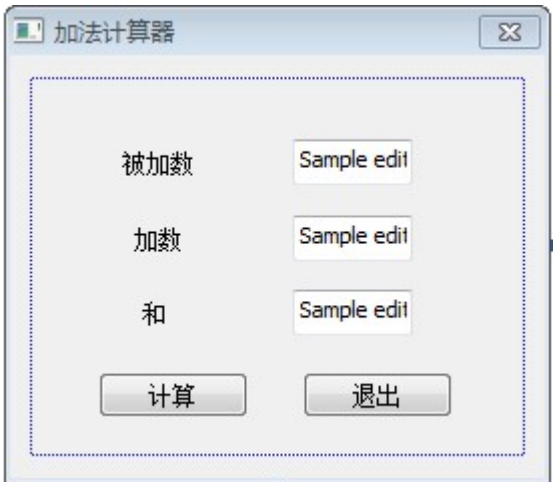
到此，对话框模板如图：



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。

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。

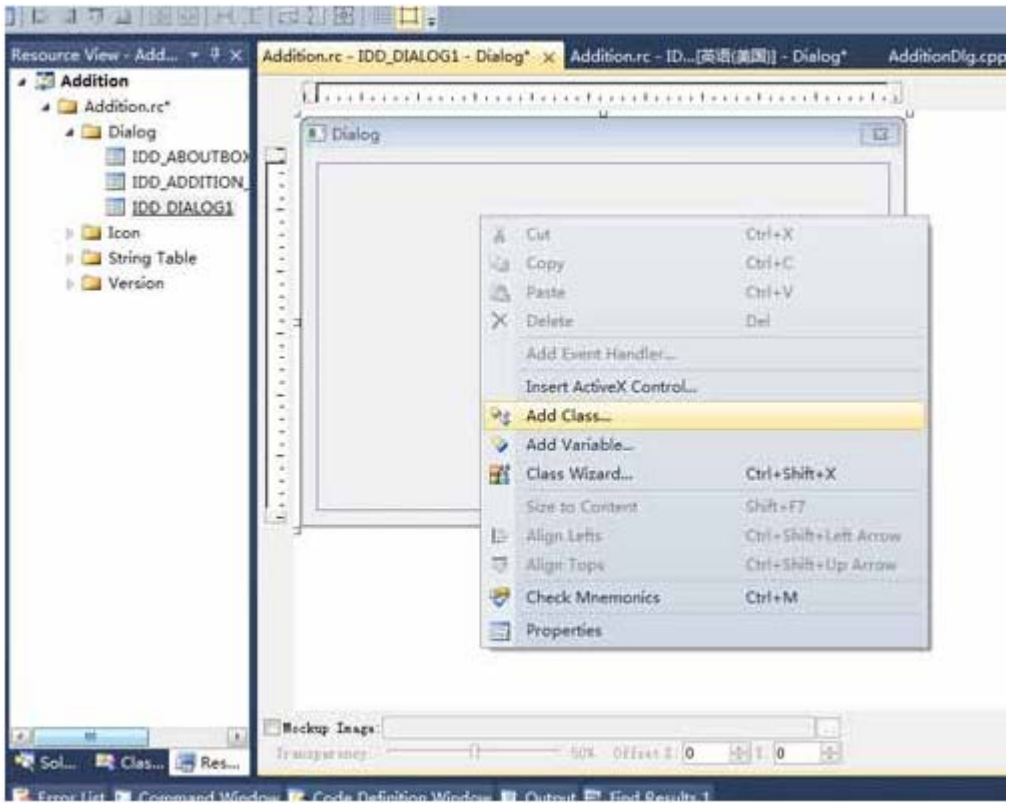
因为鸡啄米给大家的例程 Addition 是基于对话框的程序，所以程序自动创建了对话框模板 IDD_ADDITION_DIALOG，并自动生成了对话框类 CAdditionDlg，它是从 CDialogEx 类派生的。大家用过 VC++ 6.0 的可能记得，我们定义的对话框类都是从 CDialog 类派生的，但在 VS2010 中，一般对话框类都是继承自 CDialogEx 类。

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，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。

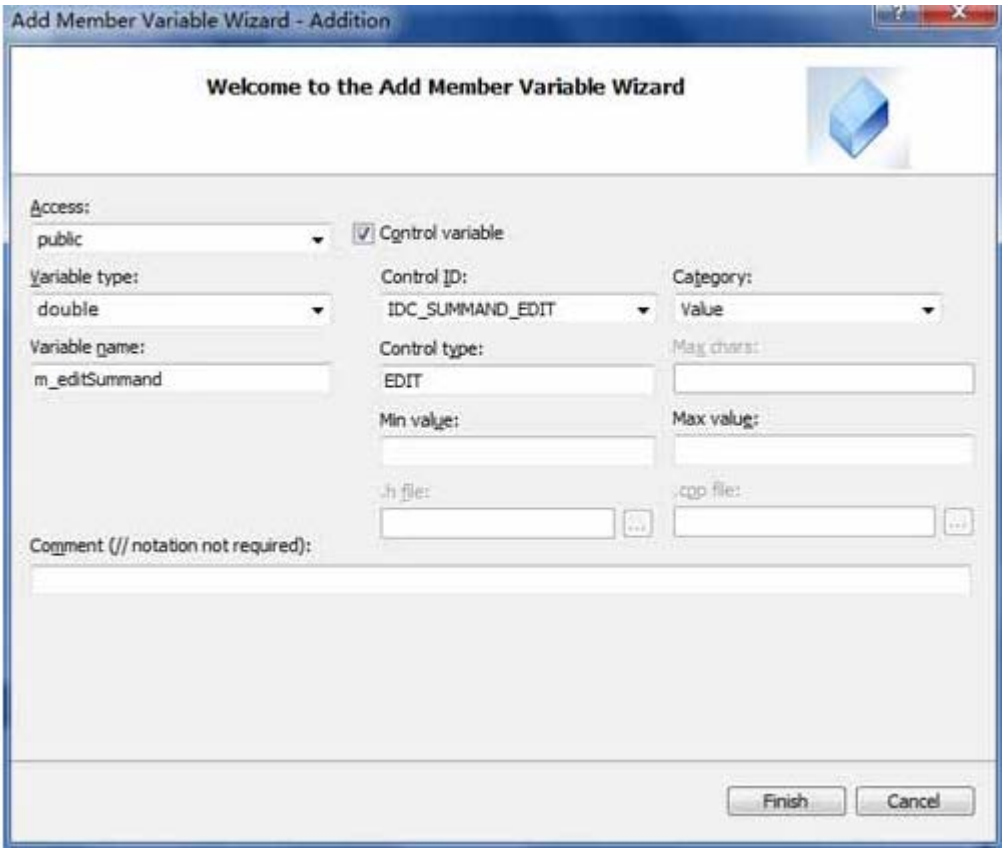
首先为被加数的编辑框 IDC_SUMMAND_EDIT 添加变量。

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 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

```
void CAdditionDlg::DoDataExchange(CDataExchange* pDX)
{
	// 处理 MFC 默认的数据交换
	CDialogEx::DoDataExchange(pDX);
	// 处理控件 IDC_SUMMAND_EDIT 和变量 m_editSummand 之间的数据交换
	DDX_Text(pDX, IDC_SUMMAND_EDIT, m_editSummand);
	// 处理控件 IDC_ADDEND_EDIT 和变量 m_editAddend 之间的数据交换
	DDX_Text(pDX, IDC_ADDEND_EDIT, m_editAddend);
	// 处理控件 IDC_SUM_EDIT 和变量 m_editSum 之间的数据交换
	DDX_Text(pDX, IDC_SUM_EDIT, m_editSum);
}
```

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。

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 为对话框和控件等定义了诸多消息，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。

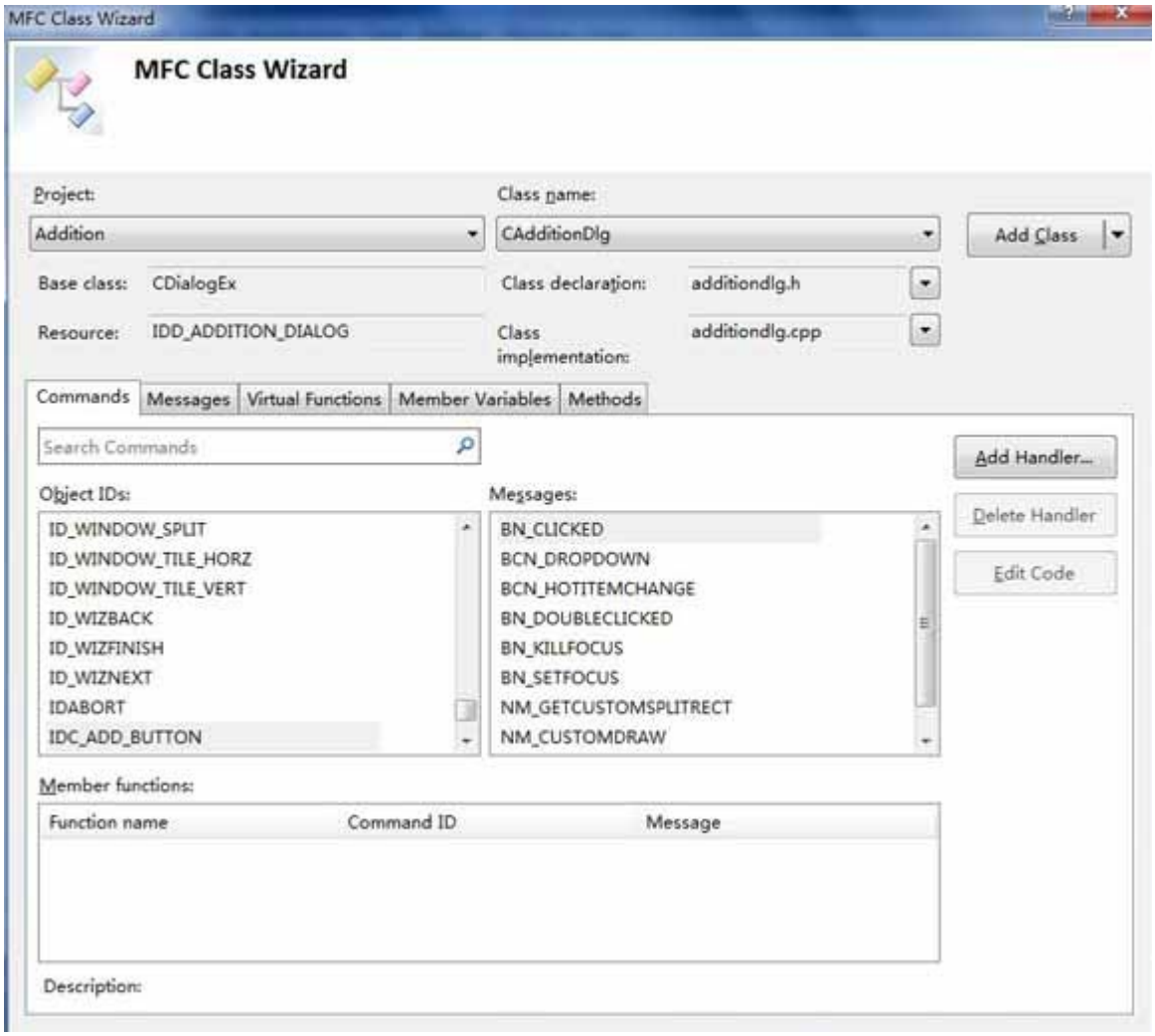
一.添加消息处理函数

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++ 6.0 的朋友应该对 Class Wizard 很熟悉了，添加类、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，久违的 Class Wizard 又回来了。但鸡啄米已经习

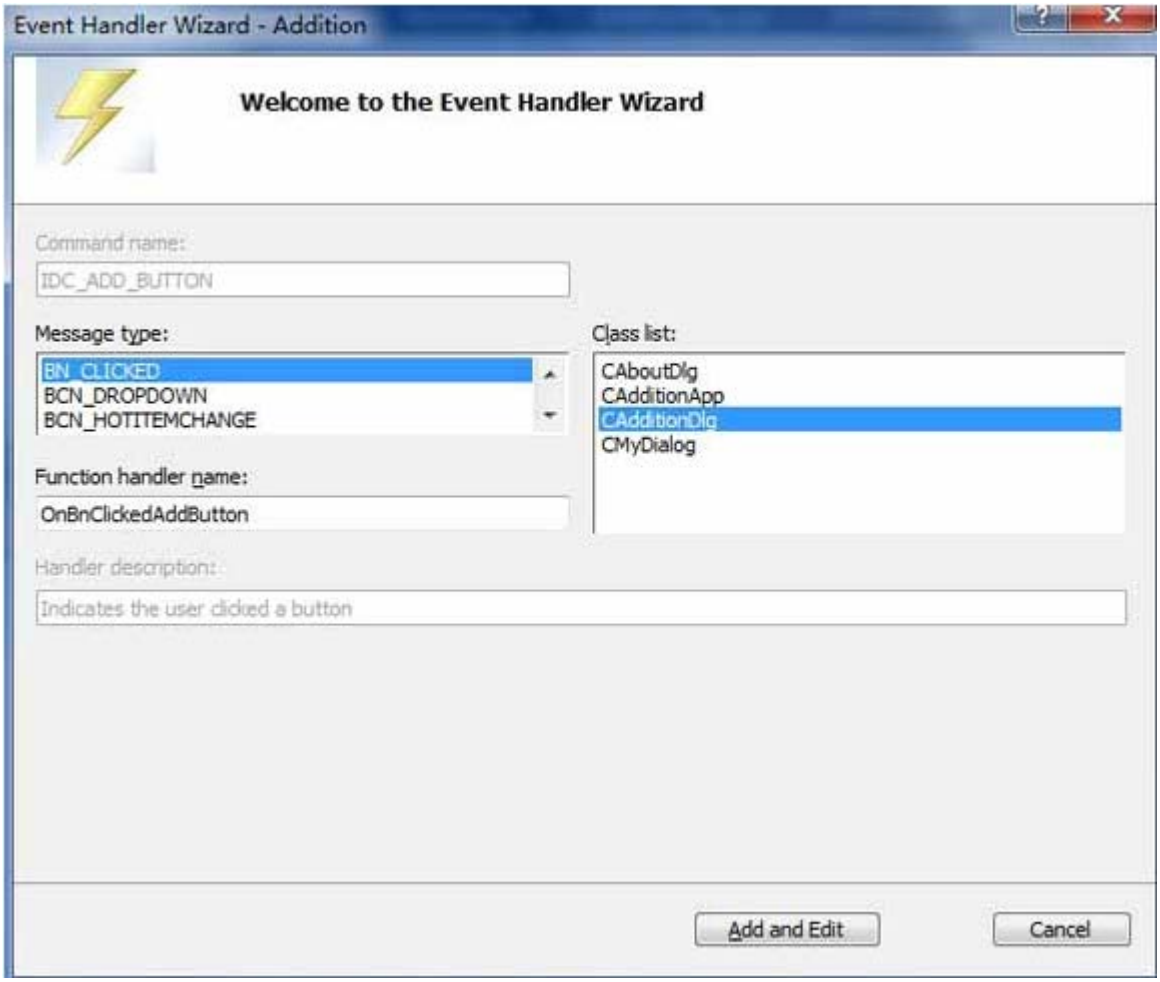
惯了使用属性中的功能了，对于从 VC++ 6.0 直接转 VS2010 的朋友可能觉得还是使用 Class Wizard 比较习惯。



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

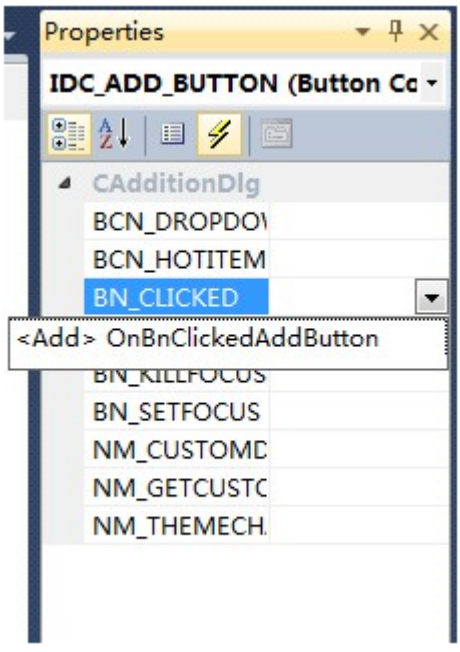
在“计算”按钮上点右键，然后在右键菜单中选择菜单项“Add Event Handler...”，弹出“Event Handler Wizard”对话框，如下图：



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

```
void CAdditionDlg::OnBnClickedAddButton()
{
	// TODO: Add your control notification handler code here
	// 将各控件中的数据保存到相应的变量
	UpdateData(TRUE);

	// 将被加数和加数的加和赋值给 m_editSum
	m_editSum = m_editSummand + m_editAddend;

	// 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
	UpdateData(FALSE);
}
```

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。

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

```
CAdditionDlg dlg;       // 定义对话框类 CAdditionDlg 的对象 dlg
m_pMainWnd = &dlg;      // 将 dlg 设为主窗口
```

```
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
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）
{
        // TODO: Place code here to handle when the dialog is
        //   dismissed with OK
}
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”）
{
        // TODO: Place code here to handle when the dialog is
        //   dismissed with Cancel
}
```

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”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”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”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

```
void CAdditionDlg::OnBnClickedAddButton()
{
	// TODO: Add your control notification handler code here
	INT_PTR nRes;		// 用于保存 DoModal 函数的返回值

	CTipDlg tipDlg;		// 构造对话框类 CTipDlg 的实例
	nRes = tipDlg.DoModal();	// 弹出对话框
	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
	 return;

	// 将各控件中的数据保存到相应的变量
	UpdateData(TRUE);

	// 将被加数和加数的加和赋值给 m_editSum
	m_editSum = m_editSummand + m_editAddend;

	// 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
	UpdateData(FALSE);
}
```

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。

到此，大家对于模态对话框的基本使用方法应该掌握了吧。

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，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和对话框类

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

```
void CAdditionDlg::OnBnClickedAddButton()
{
	// TODO: Add your control notification handler code here
	/*INT_PTR nRes;		// 用于保存 DoModal 函数的返回值

	CTipDlg tipDlg;		// 构造对话框类 CTipDlg 的实例
	nRes = tipDlg.DoModal();	// 弹出对话框
	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
	return;*/

	// 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
	if (NULL == m_pTipDlg)
	{
	// 创建非模态对话框实例
	m_pTipDlg = new CTipDlg();
	m_pTipDlg->Create(IDD_TIP_DIALOG, this);
	}
	// 显示非模态对话框
	m_pTipDlg->ShowWindow(SW_SHOW);

	// 将各控件中的数据保存到相应的变量
	UpdateData(TRUE);

	// 将被加数和加数的加和赋值给 m_editSum
	m_editSum = m_editSummand + m_editAddend;

	// 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
	UpdateData(FALSE);
}
```

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

```
CAdditionDlg::~CAdditionDlg()
{
        // 如果非模态对话框已经创建则删除它
        if (NULL != m_pTipDlg)
        {
      // 删除非模态对话框对象
      delete m_pTipDlg;
        }
}
```

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。

本节教程就到这里了，相信大家对对话框的使用更上了一个台阶了，在不同的情况下可以选择使用模态对话框和非模态对话框了。

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和非模态对话框，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 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 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

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 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

CPropertyPage 类提供了一些消息处理函数，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。

属性对话框和相关的两个类鸡啄米就先介绍到这，主要是为后面使用属性页对话框做准备

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，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”。

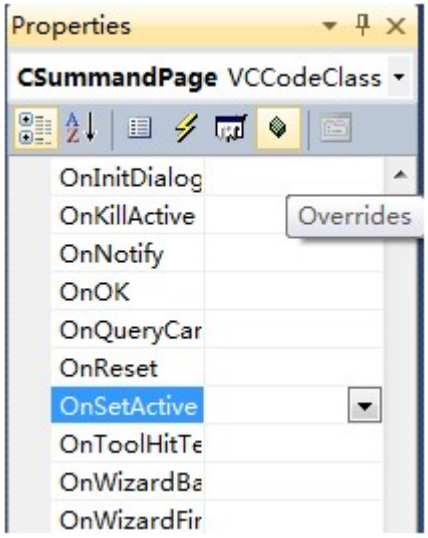
按照上述步骤，继续添加第二个和第三个对话框资源。第二个对话框模板的 ID 设为 IDD_ADDEND_PAGE，Caption 属性改为“加数页”，也添加一个静态文本框，Caption 设为“请继续输入 double 型加数”,其他属性同第一个对话框。第三个对话框模板的 ID 设为 IDD_ADD_PAGE,Caption 属性改为“计算页”，添加静态文本框的 Caption 属性改为“最后请按下“计算”按钮”，其他属性也第一个对话框一样。

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

```
BOOL CSummandPage::OnSetActive()
{
	// TODO: Add your specialized code here and/or call the base class

	// 获得父窗口，即属性表 CPropertySheet 类
	CPropertySheet* psheet = (CPropertySheet*) GetParent();
	// 设置属性表只有“下一步”按钮
	psheet->SetWizardButtons(PSWIZB_NEXT);

	return CPropertyPage::OnSetActive();
}
```

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

```
BOOL CAddPage::OnSetActive()
{
	// TODO: Add your specialized code here and/or call the base class

	// 获得父窗口，即属性表 CPropertySheet 类
	CPropertySheet* psheet = (CPropertySheet*) GetParent();
	//设置属性表只有“完成”按钮
	psheet->SetFinishText(_T("完成"));

	return CPropertyPage::OnSetActive();
}
```

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

```
BOOL CAddPage::OnWizardFinish()
{
	// TODO: Add your specialized code here and/or call the base class

	// 提示向导完成
	MessageBox(_T("使用说明向导已阅读完！"));

	return CPropertyPage::OnWizardFinish();
}
```

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

```
CAddSheet::CAddSheet(UINT nIDCaption, CWnd* pParentWnd, UINT iSelectPage)
	:CPropertySheet(nIDCaption, pParentWnd, iSelectPage)
{
	// 添加三个属性页到属性表
	AddPage(&m_summandPage);
	AddPage(&m_addendPage);
	AddPage(&m_addPage);
}

CAddSheet::CAddSheet(LPCTSTR pszCaption, CWnd* pParentWnd, UINT iSelectPage)
```

```
        :CPropertySheet(pszCaption, pParentWnd, iSelectPage)
{
        // 添加三个属性页到属性表
        AddPage(&m_summandPage);
        AddPage(&m_addendPage);
        AddPage(&m_addPage);
}
```

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

```
void CAdditionDlg::OnBnClickedInstructButton()
{
        // TODO: Add your control notification handler code here

        // 创建属性表对象
        CAddSheet sheet(_T(""));
        // 设置属性对话框为向导对话框
        sheet.SetWizardMode();
        // 打开模态向导对话框
        sheet.DoModal();
}
```

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，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。

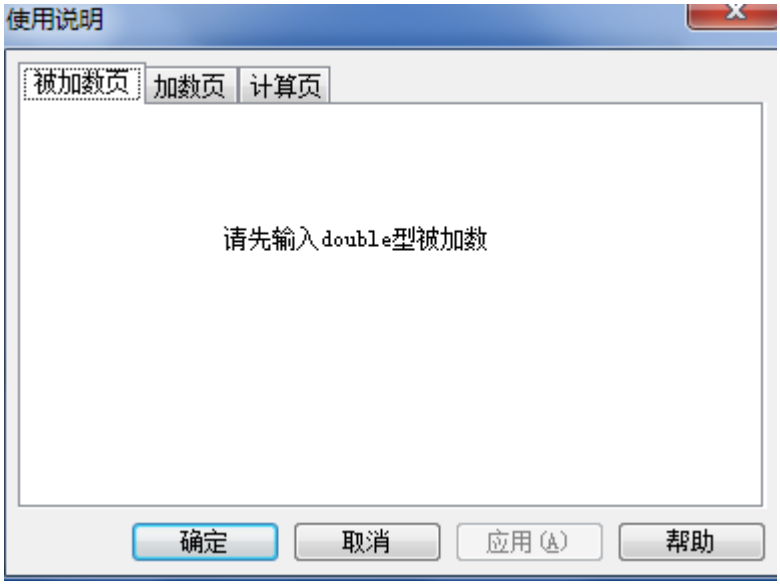
4.显示一般属性页对话框

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的参数传入。OnBnClickedInstructButton 函数修改如下：

C++代码

```
void CAdditionDlg::OnBnClickedInstructButton()
{
	// TODO: Add your control notification handler code here

	// 创建属性表对象
	CAddSheet sheet(_T("使用说明"));

	// 打开模态一般属性页对话框
	sheet.DoModal();
}
```

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。

是不是一般属性页对话框的创建和显示也很简单？到此，属性页对话框就讲完了。鸡啄米欢迎大家继续关注后面的内容。

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，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	

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。

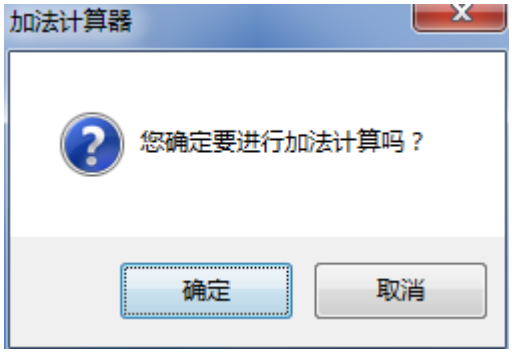
在非模态对话框的创建及显示中，鸡啄米注释了模态对话框的相关代码，加入了非模态对话框的创建和显示代码，我们在加入消息对话框之前将非模态对话框的代码也注释或删除掉，确保此函数中不再生成原来的模态对话框或非模态对话框。

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

C++代码

```
void CAdditionDlg::OnBnClickedAddButton()
{
	// TODO: Add your control notification handler code here

	INT_PTR nRes;

	// 显示消息对话框
	nRes = MessageBox(_T("您确定要进行加法计算吗？"), _T("加法计算器"), MB_OKCANCEL | MB_ICONQUESTION);
	// 判断消息对话框返回值。如果为 IDCANCEL 就 return，否则继续向下执行
	if (IDCANCEL == nRes)
		return;

	// 将各控件中的数据保存到相应的变量
	UpdateData(TRUE);

	// 将被加数和加数的加和赋值给 m_editSum
	m_editSum = m_editSummand + m_editAddend;

	// 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
	UpdateData(FALSE);
	// 设置属性对话框为向导对话框
	//sheet.SetWizardMode();
}
```

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。

消息对话框就讲到这里了。在以后的软件开发中用到它的频率很高，希望大家慢慢熟悉并掌握它。

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，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 CExample17Dlg::OnBnClickedOpenButton()和 CExample17Dlg::OnBnClickedSaveButton()。

4.修改两个消息处理函数如下：

C++代码

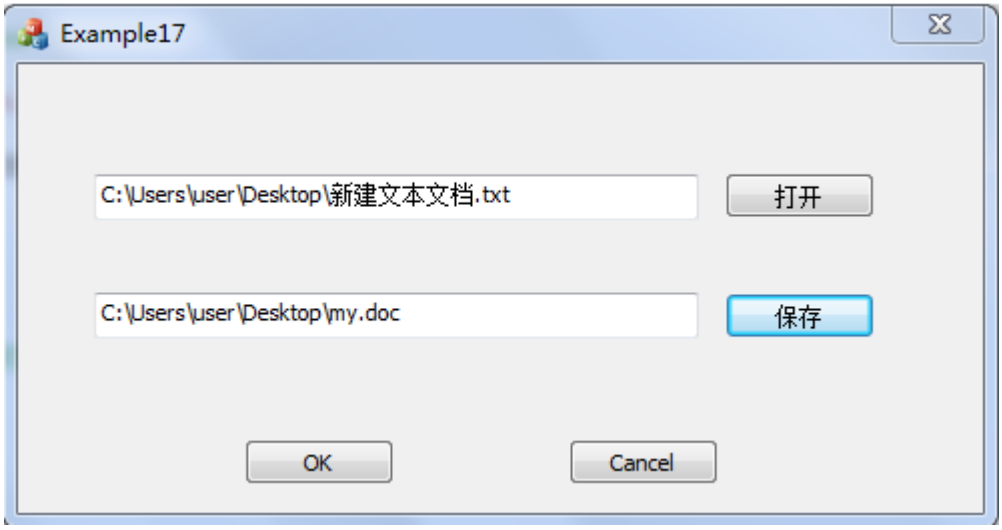
```
void CExample17Dlg::OnBnClickedOpenButton()
{
	// TODO: Add your control notification handler code here
	// 设置过滤器
	TCHAR szFilter[] = _T("文本文件(*.txt)|*.txt|所有文件(*.*)|*.*||");
	// 构造打开文件对话框
	CFileDialog fileDlg(TRUE, _T("txt"), NULL, 0, szFilter, this);
	CString strFilePath;

	// 显示打开文件对话框
	if (IDOK == fileDlg.DoModal())
	{
    // 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
    strFilePath = fileDlg.GetPathName();
    SetDlgItemText(IDC_OPEN_EDIT, strFilePath);
	}
}


void CExample17Dlg::OnBnClickedSaveButton()
{
	// TODO: Add your control notification handler code here
	// 设置过滤器
	TCHAR szFilter[] = _T("文本文件(*.txt)|*.txt|Word 文件(*.doc)|*.doc|所有文件
(*.*)|*.*||");
	// 构造保存文件对话框
	CFileDialog fileDlg(FALSE, _T("doc"), _T("my"), OFN_HIDEREADONLY |
OFN_OVERWRITEPROMPT, szFilter, this);
	CString strFilePath;

	// 显示保存文件对话框
	if (IDOK == fileDlg.DoModal())
	{
    // 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
    strFilePath = fileDlg.GetPathName();
    SetDlgItemText(IDC_SAVE_EDIT, strFilePath);
	}
}
```

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：



VS2010/MFC 编程入门之十八（对话框：字体对话框）分类标签: 编程入门　VS2010　VC＋＋　MFC　-

鸡啄米在上一节为大家讲解了文件对话框的使用，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。

CFontDialog 类的构造函数

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。

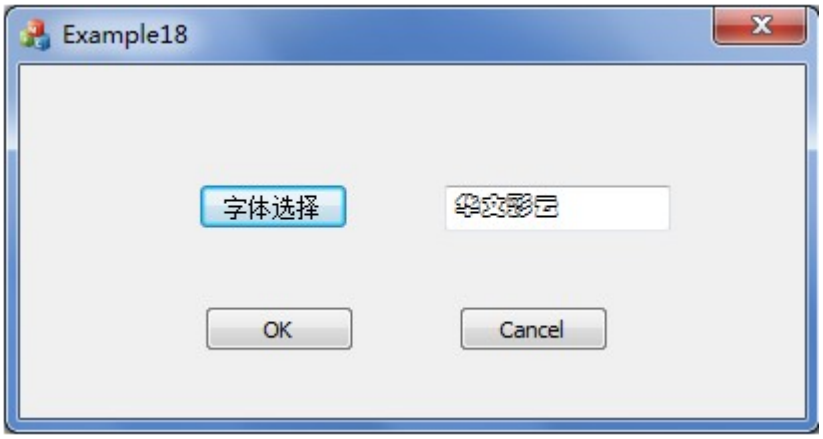
4.为按钮 IDC_FONT_BUTTON 添加点击消息的消息处理函数 CExample18Dlg::OnBnClickedFontButton()。

5.修改消息处理函数 CExample18Dlg::OnBnClickedFontButton()如下：

C++代码

```
void CExample18Dlg::OnBnClickedFontButton()
{
	// TODO: Add your control notification handler code here
	CString strFontName;		// 字体名称
	LOGFONT lf;			// LOGFONT 变量

	// 将 lf 所有字节清零
	memset(&lf, 0, sizeof(LOGFONT));

	// 将 lf 中的元素字体名设为“宋体”
	_tcscpy_s(lf.lfFaceName, LF_FACESIZE, _T("宋体"));

	// 构造字体对话框，初始选择字体名为“宋体”
	CFontDialog fontDlg(&lf);

	if (IDOK == fontDlg.DoModal())		// 显示字体对话框
	{
	// 如果 m_font 已经关联了一个字体资源对象，则释放它
	if (m_font.m_hObject)
	{
		m_font.DeleteObject();
	}
	// 使用选定字体的 LOGFONT 创建新的字体
	m_font.CreateFontIndirect(fontDlg.m_cf.lpLogFont);
	// 获取编辑框 IDC_FONT_EDIT 的 CWnd 指针，并设置其字体
	GetDlgItem(IDC_FONT_EDIT)->SetFont(&m_font);

	// 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
	strFontName = fontDlg.m_cf.lpLogFont->lfFaceName;
	SetDlgItemText(IDC_FONT_EDIT, strFontName);
	}
}
```

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：



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。

CColorDialog 类的构造函数

```
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，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CExample19Dlg::OnBnClickedColorButton()如下：

C++代码

```
void CExample19Dlg::OnBnClickedColorButton()
{
	// TODO: Add your control notification handler code here
	COLORREF color = RGB(255, 0, 0);	// 颜色对话框的初始颜色为红色
	CColorDialog colorDlg(color);	// 构造颜色对话框，传入初始颜色值

	if (IDOK == colorDlg.DoModal())	// 显示颜色对话框，并判断是否点击了“确定”
	{
		color = colorDlg.GetColor();	// 获取颜色对话框中选择的颜色值
		SetDlgItemInt(IDC_COLOR_EDIT, color);	// 在Color编辑框中显示所选颜色值
		SetDlgItemInt(IDC_R_EDIT, GetRValue(color));	// 在R编辑框中显示所选颜色的R分量值
		SetDlgItemInt(IDC_G_EDIT, GetGValue(color));	// 在G编辑框中显示所选颜色的G分量值
		SetDlgItemInt(IDC_B_EDIT, GetBValue(color));	// 在B编辑框中显示所选颜色的B分量值
	}
}
```

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。

关于颜色对话框就讲到这里了。其实各种对话框的使用都有很多相似之处，相信大家越来越熟悉了。

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 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。

静态文本框的基本应用方法在前面已经讲过，鸡啄米就不再举例，大家可以根据本节所讲进行试验，以对静态文本框有更多的认识。

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

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 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 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 m_editMultiLine。

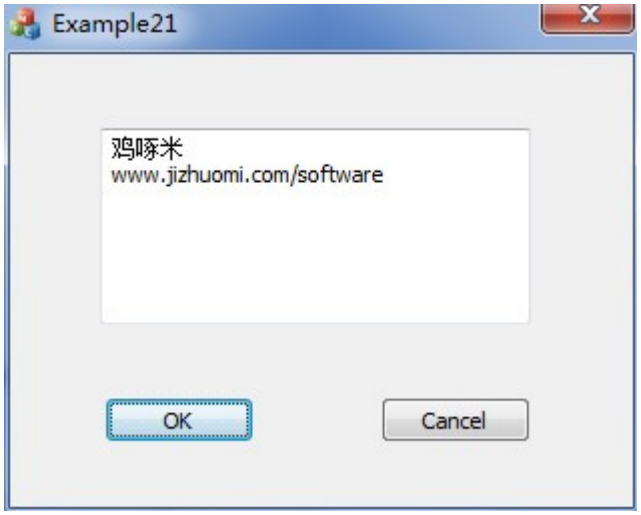
4.修改 CExample21Dlg::OnInitDialog()函数为：

C++代码

```
BOOL CExample21Dlg::OnInitDialog()
{
	CDialogEx::OnInitDialog();
```

```
    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
BOOL bNameValid;
CString strAboutMenu;
bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
ASSERT(bNameValid);
if (!strAboutMenu.IsEmpty())
{
     pSysMenu->AppendMenu(MF_SEPARATOR);
     pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
}
    }

    // Set the icon for this dialog.   The framework does this automatically
    //   when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);     // Set big icon
    SetIcon(m_hIcon, FALSE);     // Set small icon

    // TODO: Add extra initialization here
    m_editMultiLine.SetWindowText(_T("鸡啄米博客/software"));   // 设置编辑框正文为
“鸡啄米博客.com”
    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”
    m_editMultiLine.ReplaceSel(_T("\r\nwww.jizhuomi.com"));   // 将选择的“博客”替
换为“\r\nwww.jizhuomi.com”

    return TRUE;   // return TRUE   unless you set the focus to a control
}
```

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。

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

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 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。

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。

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 编程入门教程中鸡啄米讲了按钮控件 Button、Radio Button 和 Check Box 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、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，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。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 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

```
void CExample23Dlg::OnBnClickedPortalRadio()
{
	// TODO: Add your control notification handler code here
	// 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
	InitAllCheckBoxStatus();
	m_check2.EnableWindow(TRUE);
	m_check5.EnableWindow(TRUE);
}


void CExample23Dlg::OnBnClickedForumRadio()
{
	// TODO: Add your control notification handler code here
	// 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
	InitAllCheckBoxStatus();
	m_check3.EnableWindow(TRUE);
	m_check6.EnableWindow(TRUE);
}


void CExample23Dlg::OnBnClickedBlogRadio()
{
	// TODO: Add your control notification handler code here
	// 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
	InitAllCheckBoxStatus();
	m_check1.EnableWindow(TRUE);
	m_check4.EnableWindow(TRUE);
}

// 初始化所有复选框的状态，即全部禁用，全部非选中
void CExample23Dlg::InitAllCheckBoxStatus()
{
	// 全部禁用
	m_check1.EnableWindow(FALSE);
	m_check2.EnableWindow(FALSE);
	m_check3.EnableWindow(FALSE);
	m_check4.EnableWindow(FALSE);
```

```
        m_check5.EnableWindow(FALSE);
        m_check6.EnableWindow(FALSE);

        // 全部非选中
        m_check1.SetCheck(0);
        m_check2.SetCheck(0);
        m_check3.SetCheck(0);
        m_check4.SetCheck(0);
        m_check5.SetCheck(0);
        m_check6.SetCheck(0);
}
```

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

```
BOOL CExample23Dlg::OnInitDialog()
{
       CDialogEx::OnInitDialog();

       // Add "About..." menu item to system menu.

       // IDM_ABOUTBOX must be in the system command range.
       ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
       ASSERT(IDM_ABOUTBOX < 0xF000);

       CMenu* pSysMenu = GetSystemMenu(FALSE);
       if (pSysMenu != NULL)
       {
     BOOL bNameValid;
     CString strAboutMenu;
     bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
     ASSERT(bNameValid);
     if (!strAboutMenu.IsEmpty())
     {
          pSysMenu->AppendMenu(MF_SEPARATOR);
          pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
     }
       }

       // Set the icon for this dialog.   The framework does this automatically
       //   when the application's main window is not a dialog
       SetIcon(m_hIcon, TRUE);     // Set big icon
       SetIcon(m_hIcon, FALSE);     // Set small icon

       // TODO: Add extra initialization here
       // 默认选中“门户”单选按钮
```

```
	CheckDlgButton(IDC_PORTAL_RADIO, 1);
	OnBnClickedPortalRadio();

	return TRUE;  // return TRUE  unless you set the focus to a control
}
```

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

```
void CExample23Dlg::OnBnClickedOk()
{
	// TODO: Add your control notification handler code here
	CString strWebsiteSel;	// 选择的网站

	// 若选中“鸡啄米”则将其加入结果字符串
	if (1 == m_check1.GetCheck())
	{
	strWebsiteSel += _T("鸡啄米 ");
	}
	// 若选中“新浪”则将其加入结果字符串
	if (1 == m_check2.GetCheck())
	{
	strWebsiteSel += _T("新浪 ");
	}
	// 若选中“天涯论坛”则将其加入结果字符串
	if (1 == m_check3.GetCheck())
	{
	strWebsiteSel += _T("天涯论坛 ");
	}
	// 若选中“韩寒博客”则将其加入结果字符串
	if (1 == m_check4.GetCheck())
	{
	strWebsiteSel += _T("韩寒博客 ");
	}
	// 若选中“网易”则将其加入结果字符串
	if (1 == m_check5.GetCheck())
	{
	strWebsiteSel += _T("网易 ");
	}
	// 若选中“凤凰网论坛”则将其加入结果字符串
	if (1 == m_check6.GetCheck())
	{
	strWebsiteSel += _T("凤凰网论坛 ");
	}

	// 将结果字符串显示于“选择的网站”后的编辑框中
```

```
        SetDlgItemText(IDC_WEBSITE_SEL_EDIT, strWebsiteSel);

        // 为了避免点“确定”后对话框退出，将 OnOk 注掉
        //CDialogEx::OnOK();
}
```

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，并通过按钮控件实例说明了具体用法。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 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 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

```
BOOL CExample24Dlg::OnInitDialog()
{
```

```
	CDialogEx::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
BOOL bNameValid;
CString strAboutMenu;
bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
ASSERT(bNameValid);
if (!strAboutMenu.IsEmpty())
{
	pSysMenu->AppendMenu(MF_SEPARATOR);
	pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);		// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
	m_listBox.AddString(_T("新浪"));		// 在列表框结尾添加字符串“新浪”
	m_listBox.AddString(_T("鸡啄米"));		// 在列表框结尾添加字符串“鸡啄米”
	m_listBox.AddString(_T("猫扑"));		// 在列表框结尾添加字符串“猫扑”
	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”

	return TRUE;  // return TRUE  unless you set the focus to a control
}
```

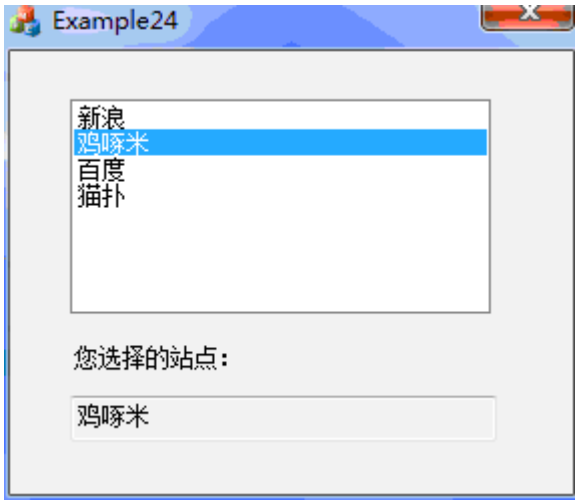
5. 我们希望在选中列表项改变时，将最新的选择项实时显示到编辑框中，那么这就要用到 LBN_SELCHANGE 通知消息。为列表框 IDC_WEB_LIST 的通知消息 LBN_SELCHANGE 添加消息处理函数 CExample24Dlg::OnLbnSelchangeWebList()，并修改如下：

C++代码

```
void CExample24Dlg::OnLbnSelchangeWebList()
{
	// TODO: Add your control notification handler code here
	CString strText;
	int nCurSel;
```

```
        nCurSel = m_listBox.GetCurSel();          // 获取当前选中列表项
        m_listBox.GetText(nCurSel, strText);      // 获取选中列表项的字符串
        SetDlgItemText(IDC_SEL_WEB_EDIT, strText); // 将选中列表项的字符串显示到编辑框
中
}
```

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 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 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。

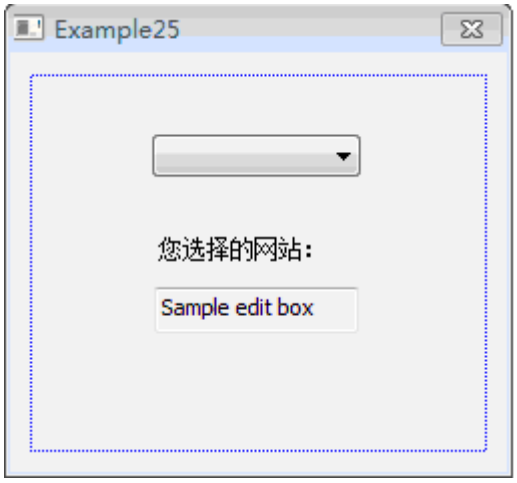
此外，CComboBox 类还继承了 CWnd 类的成员函数 GetWindowText、SetWindowText 等。

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 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

```
BOOL CExample25Dlg::OnInitDialog()
{
	CDialogEx::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
	BOOL bNameValid;
	CString strAboutMenu;
	bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
	ASSERT(bNameValid);
	if (!strAboutMenu.IsEmpty())
	{
		pSysMenu->AppendMenu(MF_SEPARATOR);
		pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
	}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);		// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon
```

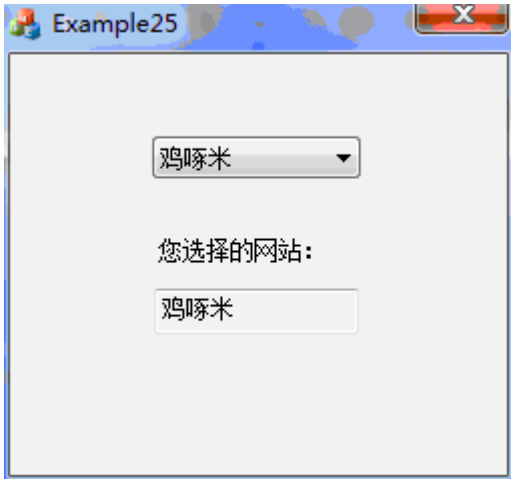
```
        // TODO: Add extra initialization here
        // 为组合框控件的列表框添加列表项“鸡啄米”
        m_comboWeb.AddString(_T("鸡啄米"));
        // 为组合框控件的列表框添加列表项“百度”
        m_comboWeb.AddString(_T("百度"));
        // 在组合框控件的列表框中索引为 1 的位置插入列表项“新浪”
        m_comboWeb.InsertString(1, _T("新浪"));

        // 默认选择第一项
        m_comboWeb.SetCurSel(0);
        // 编辑框中默认显示第一项的文字“鸡啄米”
        SetDlgItemText(IDC_SEL_WEB_EDIT, _T("鸡啄米"));

        return TRUE;   // return TRUE   unless you set the focus to a control
}
```

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 CExample25Dlg::OnCbnSelchangeWebCombo()，并修改如下：

C++代码

```
void CExample25Dlg::OnCbnSelchangeWebCombo()
{
        // TODO: Add your control notification handler code here
        CString strWeb;
        int nSel;

        // 获取组合框控件的列表框中选中项的索引
        nSel = m_comboWeb.GetCurSel();
        // 根据选中项索引获取该项字符串
        m_comboWeb.GetLBText(nSel, strWeb);
        // 将组合框中选中的字符串显示到 IDC_SEL_WEB_EDIT 编辑框中
        SetDlgItemText(IDC_SEL_WEB_EDIT, strWeb);
}
```

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 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 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

```
typedef struct tagSCROLLINFO {
       UINT cbSize;      // 结构的尺寸（字节为单位）
       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
       int   nMin;         // 滚动范围最大值，当 fMask 中包含 SIF_RANGE 时有效
       int   nMax;         // 滚动范围最小值，当 fMask 中包含 SIF_RANGE 时有效
       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
       int   nPos;         // 滚动框的位置，当 fMask 中包含 SIF_POS 有效
       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
}     SCROLLINFO, *LPSCROLLINFO;
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

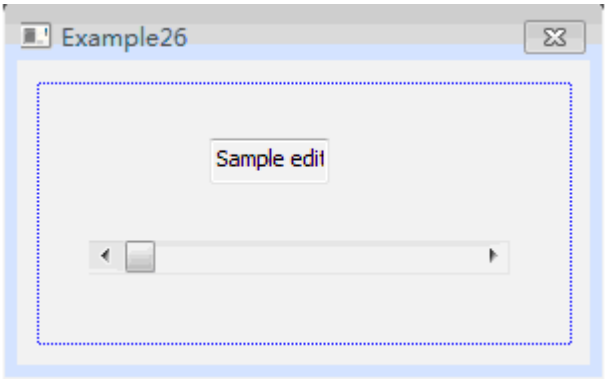
SB_ENDSCROLL：滚动结束

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

```
BOOL CExample26Dlg::OnInitDialog()
{
	CDialogEx::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
	BOOL bNameValid;
	CString strAboutMenu;
	bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
	ASSERT(bNameValid);
	if (!strAboutMenu.IsEmpty())
	{
		pSysMenu->AppendMenu(MF_SEPARATOR);
		pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
	}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);		// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
	// 设置水平滚动条的滚动范围为 1 到 100
	m_horiScrollbar.SetScrollRange(1, 100);
	// 设置水平滚动条的初始位置为 20
	m_horiScrollbar.SetScrollPos(20);
	// 在编辑框中显示 20
	SetDlgItemInt(IDC_HSCROLL_EDIT, 20);

	return TRUE;  // return TRUE  unless you set the focus to a control
}
```

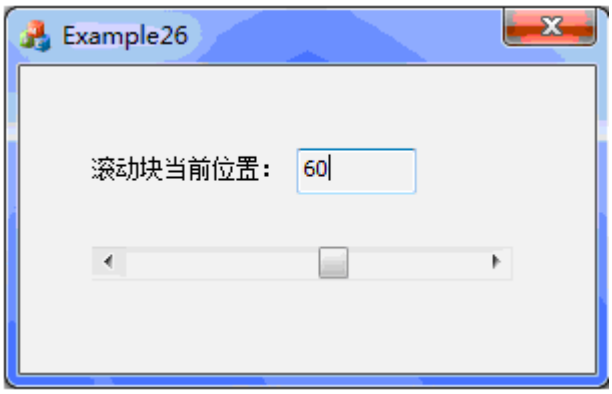
5. 现在滚动条还不能正常滚动，并且编辑框中数字也不随滚动改变。根据上面所讲，我们可以重载 CExample26Dlg 类的 OnHScroll 函数。具体操作为，在 CExample26Dlg 类的属性页面的工具栏上点“Messages”按钮，找到 WM_HSCROLL 消息，添加响应函数就可以了。OnHScroll 函数重写后如下：

C++代码

```
void CExample26Dlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
	// TODO: Add your message handler code here and/or call default
	int pos = m_horiScrollbar.GetScrollPos();	// 获取水平滚动条当前位置

	switch (nSBCode)
	{
	// 如果向左滚动一列，则 pos 减 1
	case SB_LINEUP:
		pos -= 1;
		break;
	// 如果向右滚动一列，则 pos 加 1
	case SB_LINEDOWN:
		pos   += 1;
		break;
	// 如果向左滚动一页，则 pos 减 10
	case SB_PAGEUP:
		pos -= 10;
		break;
	// 如果向右滚动一页，则 pos 加 10
	case SB_PAGEDOWN:
		pos   += 10;
		break;
	// 如果滚动到最左端，则 pos 为 1
	case SB_TOP:
		pos = 1;
		break;
	// 如果滚动到最右端，则 pos 为 100
	case SB_BOTTOM:
		pos = 100;
		break;
	// 如果拖动滚动块滚动到指定位置，则 pos 赋值为 nPos 的值
	case SB_THUMBPOSITION:
		pos = nPos;
		break;
	// 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
	default:
		SetDlgItemInt(IDC_HSCROLL_EDIT, pos);
		return;
	}
```

```
        // 设置滚动块位置
        m_horiScrollbar.SetScrollPos(pos);

        CDialogEx::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

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。

滚动条控件的内容就讲到这里了，比较基础，但这些是以后应用滚动条控件的必知内容。

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，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

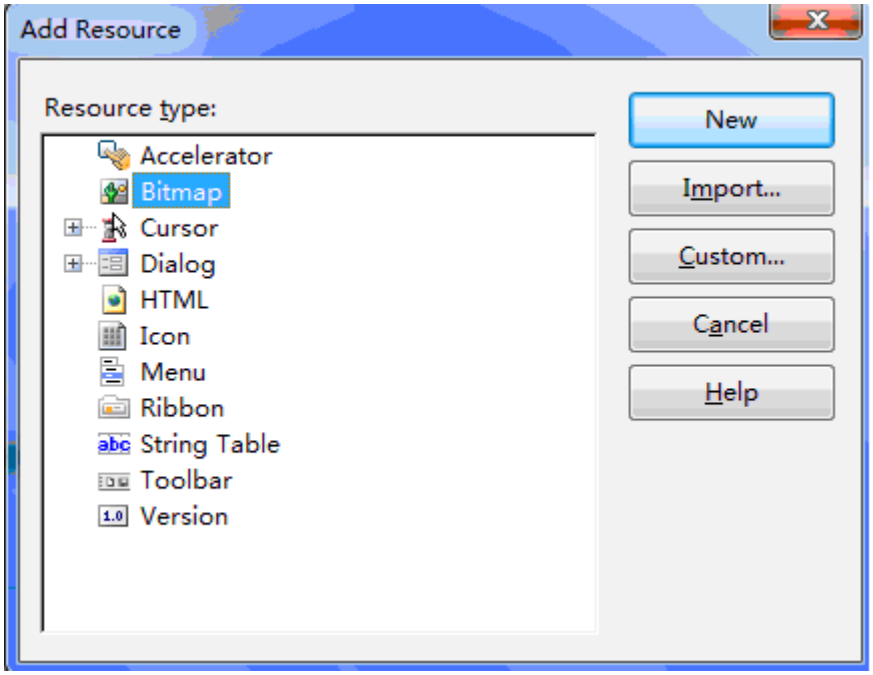
鸡啄米下面为大家演示如何为图片控件静态和动态加载位图图片。

1. 图片控件静态加载图片

1）创建一个基于对话框的 MFC 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）

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 CExample27Dlg::OnBnClickedLoadPicButton()，然后修改此函数的函数实现如下：

C++代码

```
void CExample27Dlg::OnBnClickedLoadPicButton()
{
	// TODO: Add your control notification handler code here
	CBitmap bitmap;  // CBitmap 对象，用于加载位图
	HBITMAP hBmp;  // 保存 CBitmap 加载的位图的句柄

	bitmap.LoadBitmap(IDB_BITMAP1);  // 将位图 IDB_BITMAP1 加载到 bitmap
	hBmp = (HBITMAP)bitmap.GetSafeHandle();  // 获取 bitmap 加载位图的句柄
	m_jzmPicture.SetBitmap(hBmp);  // 设置图片控件 m_jzmPicture 的位图图片为 IDB_BITMAP1
}
```

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。应该说还是比较简单的。

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，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 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）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)
```

还有消息处理函数自动生成时的形式：

C++代码

```
void CExample29Dlg::OnNMClickProgramLangList(NMHDR *pNMHDR, LRESULT *pResult)
{
	LPNMITEMACTIVATE pNMItemActivate =
reinterpret_cast<LPNMITEMACTIVATE>(pNMHDR);
	// TODO: Add your control notification handler code here
	*pResult = 0;
}
```

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

```
typedef struct tagNMHDR {
	HWND hwndFrom;		// 控件窗口的句柄
	  UINT_PTR idFrom;	// 控件 ID
	UINT code;	// 控件的通知消息码
} 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

```
typedef struct tagNMITEMACTIVATE {
	NMHDR hdr;
	int iItem;
	int iSubItem;
```

```
        UINT uNewState;
        UINT uOldState;
        UINT uChanged;
        POINT ptAction;
        LPARAM lParam;
        UINT uKeyFlags;
} 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

```
typedef struct _LVITEM {
        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
        int iItem;          // 列表项的索引
        int iSubItem;       // 列表子项的索引
        UINT state;         // 状态，下面会列出。（LVIF_STATE）
        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
        LPTSTR pszText;     // 指向列表项或列表子项的标签字符串。（LVIF_TEXT）
        int cchTextMax;     // pszText 指向缓冲区的字符的个数，包括字符串结束符。（LVIF_TEXT）
        int iImage;         // 图标的索引。(LVIF_IMAGE)
        LPARAM lParam;      // 32 位的附加数据。(LVIF_PARAM)
#if (_WIN32_IE >= 0x0300)
        int iIndent;
#endif
#if (_WIN32_WINNT >= 0x501)
        int iGroupId;
        UINT cColumns; // tile view columns
        PUINT puColumns;
#endif
#if (_WIN32_WINNT >= 0x0600)
        int* piColFmt;
        int iGroup;
#endif
} 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
```
typedef struct _LVCOLUMN {
       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
       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）
       int cx;          // 以像素为单位的列的宽度。（LVCF_FMT）
       LPTSTR pszText;       // 指向列表头标题正文的字符串。（LVCF_TEXT）
       int cchTextMax;          // pszText 指向缓冲区的字符的个数，包括字符串结束符。（LVCF_TEXT）
       int iSubItem;      // 该列的索引。（LVCF_SUBITEM）
#if (_WIN32_IE >= 0x0300)
       int iImage;
       int iOrder;
#endif
#if (_WIN32_WINNT >= 0x0600)
       int cxMin;
       int cxDefault;
       int cxIdeal;
#endif
} 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
```
typedef struct tagNMLISTVIEW {
       NMHDR hdr;    // 标准的 NMHDR 结构
         int iItem;    // 列表项的索引
         int iSubItem;       // 列表子项的索引
         UINT uNewState;   // 列表项或列表子项的新状态
         UINT uOldState;   // 列表项或列表子项原来的状态
         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
         POINT ptAction;   // 事件发生时鼠标的客户区坐标
         LPARAM lParam;     //32 位的附加数据
} NMLISTVIEW, *LPNMLISTVIEW;
```

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，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 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。

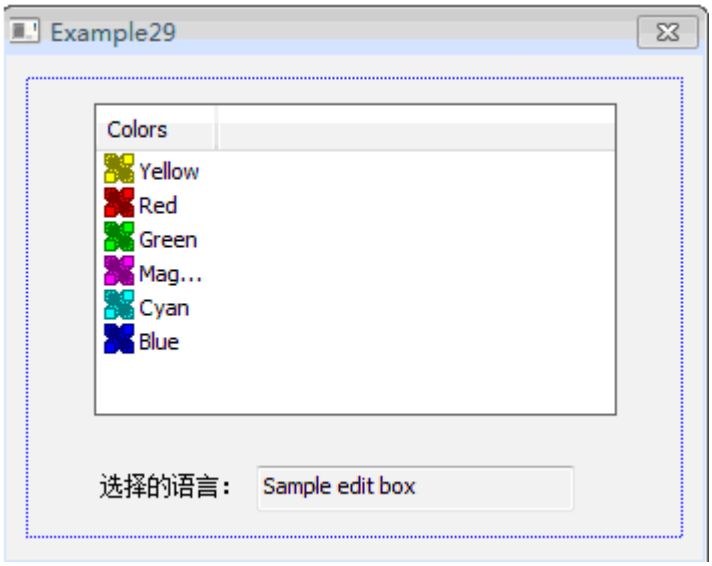
CListCtrl 类应用实例

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 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

```
BOOL CExample29Dlg::OnInitDialog()
{
	CDialogEx::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
	BOOL bNameValid;
	CString strAboutMenu;
	bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
	ASSERT(bNameValid);
	if (!strAboutMenu.IsEmpty())
	{
		pSysMenu->AppendMenu(MF_SEPARATOR);
		pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
	}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);		// Set big icon
```

```
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
	CRect rect;

	// 获取编程语言列表视图控件的位置和大小
	m_programLangList.GetClientRect(&rect);

	// 为列表视图控件添加全行选中和栅格风格
	m_programLangList.SetExtendedStyle(m_programLangList.GetExtendedStyle() |
LVS_EX_FULLROWSELECT | LVS_EX_GRIDLINES);

	// 为列表视图控件添加三列
	m_programLangList.InsertColumn(0, _T("语言"), LVCFMT_CENTER, rect.Width()/3,
0);
	m_programLangList.InsertColumn(1, _T("2012.02 排名"), LVCFMT_CENTER,
rect.Width()/3, 1);
	m_programLangList.InsertColumn(2, _T("2011.02 排名"), LVCFMT_CENTER,
rect.Width()/3, 2);

	// 在列表视图控件中插入列表项，并设置列表子项文本
	m_programLangList.InsertItem(0, _T("Java"));
	m_programLangList.SetItemText(0, 1, _T("1"));
	m_programLangList.SetItemText(0, 2, _T("1"));
	m_programLangList.InsertItem(1, _T("C"));
	m_programLangList.SetItemText(1, 1, _T("2"));
	m_programLangList.SetItemText(1, 2, _T("2"));
	m_programLangList.InsertItem(2, _T("C#"));
	m_programLangList.SetItemText(2, 1, _T("3"));
	m_programLangList.SetItemText(2, 2, _T("6"));
	m_programLangList.InsertItem(3, _T("C++"));
	m_programLangList.SetItemText(3, 1, _T("4"));
	m_programLangList.SetItemText(3, 2, _T("3"));

	return TRUE;  // return TRUE  unless you set the focus to a control
}
```

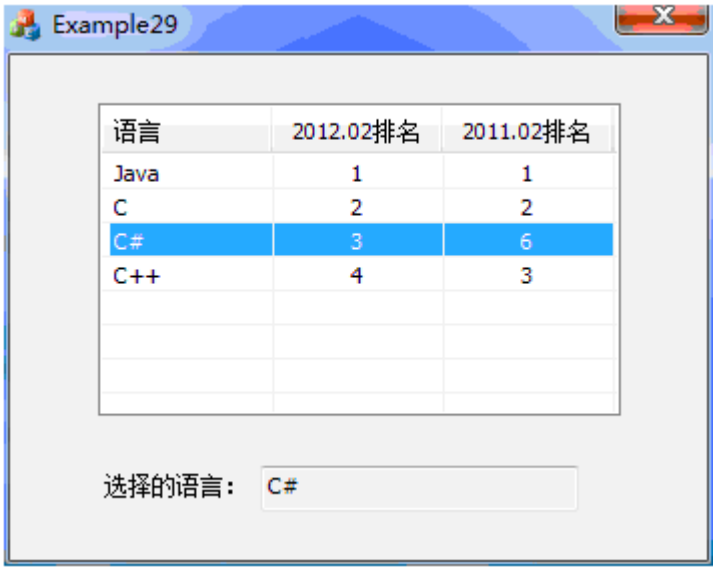
5. 我们希望在选中列表项改变时，将最新的选择项实时显示到编辑框中，那么可以使用 NM_CLICK 通知消息。为列表框 IDC_PROGRAM_LANG_LIST 的通知消息 NM_CLICK 添加消息处理函数 CExample29Dlg::OnNMClickProgramLangList，并修改如下：

C++代码

```
void CExample29Dlg::OnNMClickProgramLangList(NMHDR *pNMHDR, LRESULT *pResult)
{
	LPNMITEMACTIVATE pNMItemActivate =
reinterpret_cast<LPNMITEMACTIVATE>(pNMHDR);
	// TODO: Add your control notification handler code here
```

```
    *pResult = 0;

    CString strLangName;        // 选择语言的名称字符串
    NMLISTVIEW *pNMListView = (NMLISTVIEW*)pNMHDR;

    if (-1 != pNMListView->iItem)      // 如果 iItem 不是-1，就说明有列表项被选择
    {
  // 获取被选择列表项第一个子项的文本
  strLangName = m_programLangList.GetItemText(pNMListView->iItem, 0);
  // 将选择的语言显示与编辑框中
  SetDlgItemText(IDC_LANG_SEL_EDIT, strLangName);
    }
}
```

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，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，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

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

```
typedef struct tagTVITEM {
	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
	HTREEITEM hItem; // 树节点的句柄(TVIF_HANDLE)
	UINT state;	// 树节点的状态(TVIF_STATE)
	UINT stateMask;	// 状态的掩码组合(TVIF_STATE)
	LPTSTR pszText;	// 树节点的标签文本(TVIF_TEXT)
	int cchTextMax;	// 标签文本缓冲区的大小(TVIF_TEXT)
	int iImage;	// 树节点的图像索引(TVIF_IMAGE)
	int iSelectedImage;	// 选中项的图像索引(TVIF_SELECTEDIMAGE)
	int cChildren;	// 表明节点是否有子节点，为 1 则有，为 0 则没有(TVIF_CHILDREN)
	LPARAM lParam;	// 一个 32 位的附加数据(TVIF_PARAM)
} 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

```
typedef struct tagNMTREEVIEW {
       NMHDR hdr;     // 标准的 NMHDR 结构
       UINT action;       // 表明是用户的什么行为触发了该通知消息
       TVITEM itemOld; // 原节点的属性
       TVITEM itemNew; // 新节点的属性
       POINT ptDrag;     // 事件发生时鼠标的客户区坐标
} 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

```
typedef struct tagTVINSERTSTRUCT {
       HTREEITEM hParent;     // 父节点的句柄
       HTREEITEM hInsertAfter; // 指明插入到同层中哪一项的后面
#if (_WIN32_IE >= 0x0400)
       union
       {
     TVITEMEX itemex;
     TVITEM item;
       } DUMMYUNIONNAME;
#else
       TVITEM item;          // 要添加的新节点的属性
#endif
} 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

```
typedef struct tagNMTVDISPINFO {
```

```
    NMHDR hdr;
    TVITEM item;
} 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，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 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

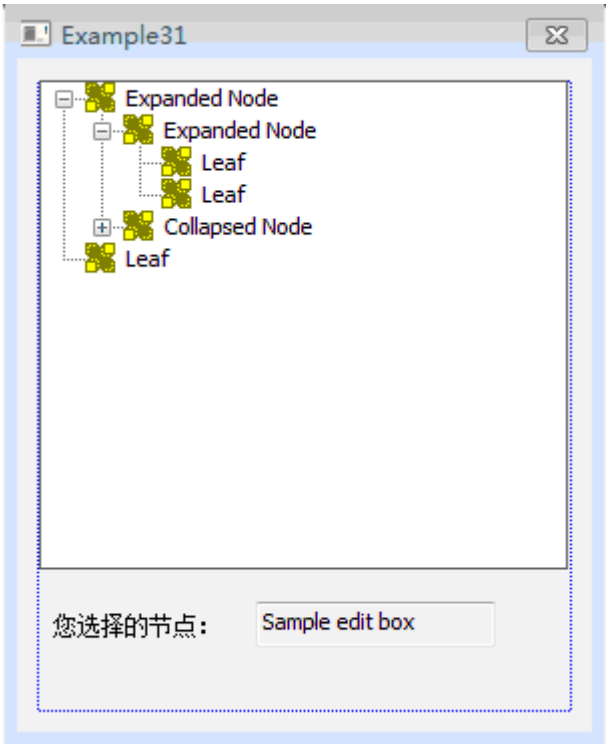
最后鸡啄米还是给大家写一个简单的实例，说明 CListCtrl 类的几个成员函数及树形控件通知消息等的使用方法。

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、“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。

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 CExample29Dlg::OnInitDialog()函数为：

C++代码

```
BOOL CExample31Dlg::OnInitDialog()
{
	CDialogEx::OnInitDialog();
	.....略

	// TODO: Add extra initialization here
	HICON hIcon[3];		// 图标句柄数组
	HTREEITEM hRoot;		// 树的根节点的句柄
	HTREEITEM hCataItem; // 可表示任一分类节点的句柄
	HTREEITEM hArtItem;	// 可表示任一文章节点的句柄

	// 加载三个图标，并将它们的句柄保存到数组
	hIcon[0] = theApp.LoadIcon(IDI_WEB_ICON);
	hIcon[1] = theApp.LoadIcon(IDI_CATALOG_ICON);
	hIcon[2] = theApp.LoadIcon(IDI_ARTICLE_ICON);
```

```
    // 创建图像序列 CImageList 对象
    m_imageList.Create(32, 32, ILC_COLOR32, 3, 3);
    // 将三个图标添加到图像序列
    for (int i=0; i<3; i++)
    {
  m_imageList.Add(hIcon[i]);
    }

    // 为树形控件设置图像序列
    m_webTree.SetImageList(&m_imageList, TVSIL_NORMAL);

    // 插入根节点
    hRoot = m_webTree.InsertItem(_T("鸡啄米"), 0, 0);
    // 在根节点下插入子节点
    hCataItem = m_webTree.InsertItem(_T("IT 互联网"), 1, 1, hRoot, TVI_LAST);
    // 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
    m_webTree.SetItemData(hCataItem, 1);
    // 在“IT 互联网”节点下插入子节点
    hArtItem = m_webTree.InsertItem(_T("百度文章 1"), 2, 2, hCataItem, TVI_LAST);
    // 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
    m_webTree.SetItemData(hArtItem, 2);
    // 在“IT 互联网”节点下插入另一子节点
    hArtItem = m_webTree.InsertItem(_T("谷歌文章 2"), 2, 2, hCataItem, TVI_LAST);
    // 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
    m_webTree.SetItemData(hArtItem, 3);
    // 在根节点下插入第二个子节点
    hCataItem = m_webTree.InsertItem(_T("数码生活"), 1, 1, hRoot, TVI_LAST);
    // 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
    m_webTree.SetItemData(hCataItem, 4);
    // 在“数码生活”节点下插入子节点
    hArtItem = m_webTree.InsertItem(_T("智能手机文章 1"), 2, 2, hCataItem,
TVI_LAST);
    // 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
    m_webTree.SetItemData(hArtItem, 5);
    // 在“数码生活”节点下插入另一子节点
    hArtItem = m_webTree.InsertItem(_T("平板电脑文章 2"), 2, 2, hCataItem,
TVI_LAST);
    // 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
    m_webTree.SetItemData(hArtItem, 6);
    // 在根节点下插入第三个子节点
    hCataItem = m_webTree.InsertItem(_T("软件开发"), 1, 1, hRoot, TVI_LAST);
    // 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
    m_webTree.SetItemData(hCataItem, 7);
    // 在“软件开发”节点下插入子节点
    hArtItem = m_webTree.InsertItem(_T("C++编程入门系列 1"), 2, 2, hCataItem,
TVI_LAST);
```

```
        // 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
        m_webTree.SetItemData(hArtItem, 8);
        // 在“软件开发”节点下插入另一子节点
        hArtItem = m_webTree.InsertItem(_T("VS2010/MFC 编程入门 2"), 2, 2, hCataItem,
TVI_LAST);
        // 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
        m_webTree.SetItemData(hArtItem, 9);
        // 在根节点下插入第四个子节点
        hCataItem = m_webTree.InsertItem(_T("娱乐休闲"), 1, 1, hRoot, TVI_LAST);
        // 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
        m_webTree.SetItemData(hCataItem, 10);
        // 在“娱乐休闲”节点下插入子节点
        hArtItem = m_webTree.InsertItem(_T("玛雅文明文章 1"), 2, 2, hCataItem,
TVI_LAST);
        // 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
        m_webTree.SetItemData(hArtItem, 11);
        // 在“娱乐休闲”节点下插入另一子节点
        hArtItem = m_webTree.InsertItem(_T("IT 笑话 2"), 2, 2, hCataItem, TVI_LAST);
        // 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
        m_webTree.SetItemData(hArtItem, 12);

        return TRUE;   // return TRUE   unless you set the focus to a control
}
```

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 CExample31Dlg::OnTvnSelchangedWebTree，并修改函数体如下：

C++代码

```
void CExample31Dlg::OnTvnSelchangedWebTree(NMHDR *pNMHDR, LRESULT *pResult)
{
        LPNMTREEVIEW pNMTreeView = reinterpret_cast<LPNMTREEVIEW>(pNMHDR);
        // TODO: Add your control notification handler code here
        *pResult = 0;

        CString strText; // 树节点的标签文本字符串

        // 获取当前选中节点的句柄
        HTREEITEM hItem = m_webTree.GetSelectedItem();
        // 获取选中节点的标签文本字符串
        strText = m_webTree.GetItemText(hItem);
        // 将字符串显示到编辑框中
        SetDlgItemText(IDC_ITEM_SEL_EDIT, strText);
}
```

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

```
void CExample31Dlg::OnTvnGetInfoTipWebTree(NMHDR *pNMHDR, LRESULT *pResult)
{
	LPNMTVGETINFOTIP pGetInfoTip = reinterpret_cast<LPNMTVGETINFOTIP>(pNMHDR);
	// TODO: Add your control notification handler code here
	*pResult = 0;
	NMTVGETINFOTIP* pTVTipInfo = (NMTVGETINFOTIP*)pNMHDR;	// 将传入的 pNMHDR 转换为 NMTVGETINFOTIP 指针类型
	HTREEITEM hRoot = m_webTree.GetRootItem();	// 获取树的根节点
	CString strText;		// 每个树节点的提示信息

	if (pTVTipInfo->hItem == hRoot)
	{
	// 如果鼠标划过的节点是根节点，则提示信息为空
	strText = _T("");
	}
	else
	{
	// 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
	strText.Format(_T("%d"), pTVTipInfo->lParam);
	}

	// 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
	wcscpy(pTVTipInfo->pszText, strText);
}
```

8. 运行程序，弹出结果对话框。效果如下图：

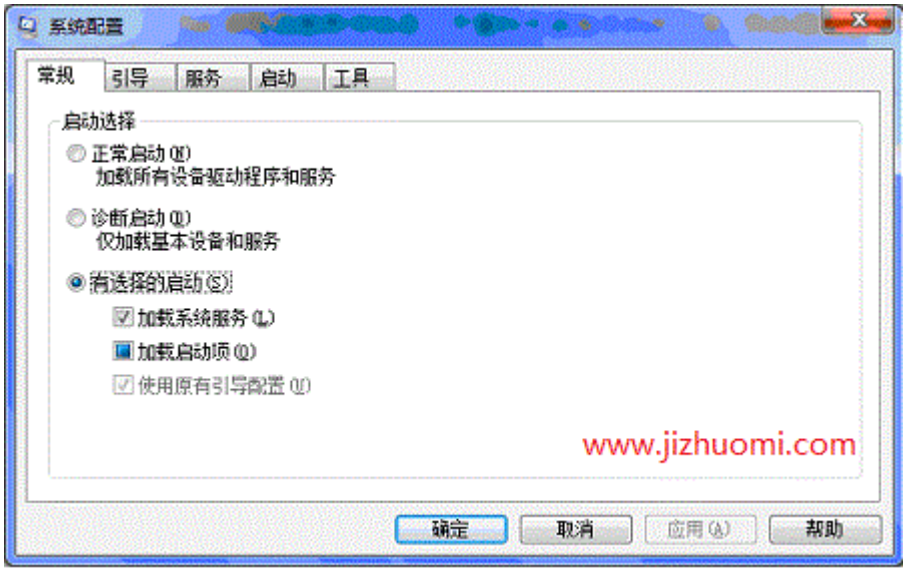
树形控件的知识就讲到这里了，相比之前的控件可能稍有复杂。

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，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，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

```
typedef struct tagTCITEMHEADER {
	UINT mask;	// 掩码，可以为 TCIF_IMAGE（iImage 成员有效）、TCIF_RTLREADING、TCIF_TEXT（pszText 成员有效）
	UINT lpReserved1;	// 预留
	UINT lpReserved2;	// 预留
	LPTSTR pszText;	// 标签文本字符串
	int cchTextMax;
	int iImage;	// 图标在标签控件图像序列中的索引
} 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

```
typedef struct tagTCITEM {
	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）
#if (_WIN32_IE >= 0x0300)
	DWORD dwState;
	DWORD dwStateMask;
#else
	UINT lpReserved1;
	UINT lpReserved2;
#endif
	LPTSTR pszText;
	int cchTextMax;
```

```
        int iImage;
        LPARAM lParam;          // 与标签页关联的 32 位数据
} 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

```
typedef struct tagTCHITTESTINFO {
        POINT pt;   // 鼠标点击测试的客户区坐标
        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）
} 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

```
typedef struct tagNMTCKEYDOWN {
        NMHDR hdr;
        WORD wVKey;
        UINT flags;
} NMTCKEYDOWN;
```

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，本节继续讲下半部分。

标签控件的创建

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、“OK”按钮和“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

```
BOOL CExample33Dlg::OnInitDialog()
{
	CDialogEx::OnInitDialog();
```

```
	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
BOOL bNameValid;
CString strAboutMenu;
bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
ASSERT(bNameValid);
if (!strAboutMenu.IsEmpty())
{
	pSysMenu->AppendMenu(MF_SEPARATOR);
	pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);		// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
	CRect tabRect;		// 标签控件客户区的位置和大小

	m_tab.InsertItem(0, _T("鸡啄米"));		// 插入第一个标签“鸡啄米”
	m_tab.InsertItem(1, _T("Android开发网"));	// 插入第二个标签“Android开发网”
	m_jzmDlg.Create(IDD_JIZHUOMI_DIALOG, &m_tab);		// 创建第一个标签页
	m_androidDlg.Create(IDD_ANDROID_DIALOG, &m_tab); // 创建第二个标签页

	m_tab.GetClientRect(&tabRect);		// 获取标签控件客户区Rect
	// 调整tabRect，使其覆盖范围适合放置标签页
	tabRect.left += 1;
	tabRect.right -= 1;
	tabRect.top += 25;
	tabRect.bottom -= 1;
	// 根据调整好的tabRect放置m_jzmDlg子对话框，并设置为显示
	m_jzmDlg.SetWindowPos(NULL, tabRect.left, tabRect.top, tabRect.Width(), tabRect.Height(), SWP_SHOWWINDOW);
	// 根据调整好的tabRect放置m_androidDlg子对话框，并设置为隐藏
```

```
	m_androidDlg.SetWindowPos(NULL, tabRect.left, tabRect.top, tabRect.Width(),
tabRect.Height(), SWP_HIDEWINDOW);

	return TRUE;  // return TRUE  unless you set the focus to a control
}
```

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，并修改如下：

C++代码

```
void CExample33Dlg::OnTcnSelchangeTab1(NMHDR *pNMHDR, LRESULT *pResult)
{
	// TODO: Add your control notification handler code here
	*pResult = 0;
	CRect tabRect;		// 标签控件客户区的 Rect

	// 获取标签控件客户区 Rect，并对其调整，以适合放置标签页
	m_tab.GetClientRect(&tabRect);
	tabRect.left += 1;
	tabRect.right -= 1;
	tabRect.top += 25;
	tabRect.bottom -= 1;

	switch (m_tab.GetCurSel())
	{
	// 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
	case 0:
    m_jzmDlg.SetWindowPos(NULL, tabRect.left, tabRect.top, tabRect.Width(),
tabRect.Height(), SWP_SHOWWINDOW);
    m_androidDlg.SetWindowPos(NULL, tabRect.left, tabRect.top, tabRect.Width(),
tabRect.Height(), SWP_HIDEWINDOW);
    break;
	// 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
	case 1:
    m_jzmDlg.SetWindowPos(NULL, tabRect.left, tabRect.top, tabRect.Width(),
tabRect.Height(), SWP_HIDEWINDOW);
    m_androidDlg.SetWindowPos(NULL, tabRect.left, tabRect.top, tabRect.Width(),
tabRect.Height(), SWP_SHOWWINDOW);
    break;
	default:
    break;
	}
}
```

8. 再运行程序，最终的标签页切换效果如下面两图：

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 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。

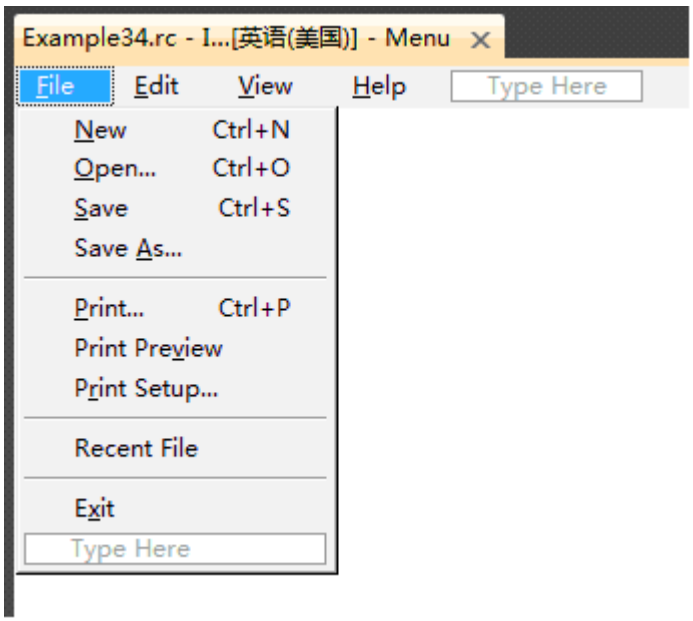
下拉式菜单一般在窗口标题栏下面显示，大家还记得我们在 VS2010/MFC 编程入门之二（利用 MFC 向导生成单文档应用程序框架）中创建的 HelloWorld 单文档工程吗？它的运行结果窗口的标题栏下就是下拉式菜单。下拉式菜单通常是由主菜单栏、子菜单及子菜单中的菜单项和分隔条所组成的。

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 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：

ID	Modifier	Key	Type
ID_EDIT_COPY	Ctrl	C	VIRTKEY
ID_EDIT_COPY	Ctrl	VK_INSERT	VIRTKEY
ID_EDIT_CUT	Shift	VK_DELETE	VIRTKEY
ID_EDIT_CUT	Ctrl	X	VIRTKEY
ID_EDIT_PASTE	Ctrl	V	VIRTKEY
ID_EDIT_PASTE	Shift	VK_INSERT	VIRTKEY
ID_EDIT_UNDO	Alt	VK_BACK	VIRTKEY
ID_EDIT_UNDO	Ctrl	Z	VIRTKEY
ID_FILE_NEW	Ctrl	N	VIRTKEY
ID_FILE_OPEN	Ctrl	O	VIRTKEY
ID_FILE_PRINT	Ctrl	P	VIRTKEY
ID_FILE_SAVE	Ctrl	S	VIRTKEY
ID_NEXT_PANE	None	VK_F6	VIRTKEY
ID_PREV_PANE	Shift	VK_F6	VIRTKEY

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”。

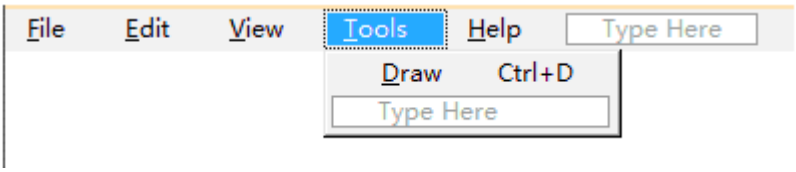
VS2010 菜单资源编辑

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：



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，本节主要讲菜单及 CMenu 类的使用。

CMenu 类的主要成员函数

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：

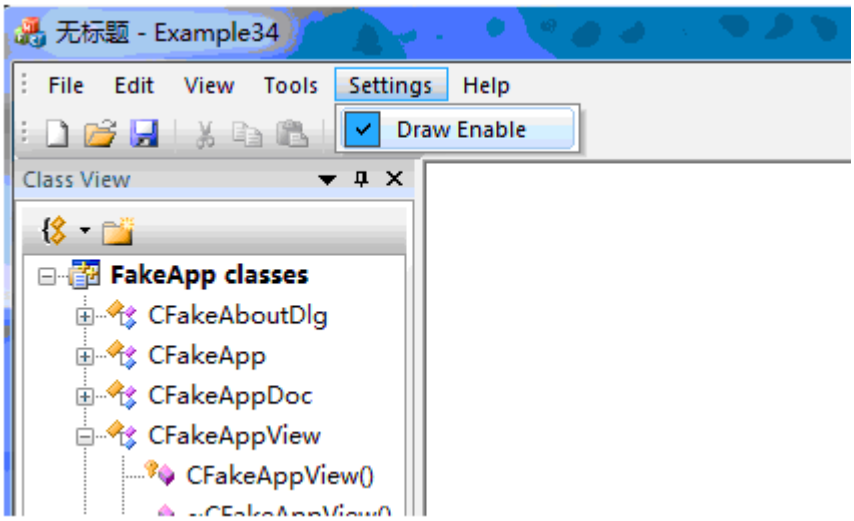
C++代码
```
void CMainFrame::OnToolsDraw()
{
	// TODO: Add your command handler code here
	// 弹出提示框
	MessageBox(_T("Draw"));
}

void CMainFrame::OnUpdateToolsDraw(CCmdUI *pCmdUI)
{
	// TODO: Add your command update UI handler code here
	// 根据 m_bDraw 的值设置是否激活
	pCmdUI->Enable(m_bDraw);
}
```

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
```
void CMainFrame::OnSettingsDrawenable()
{
	// TODO: Add your command handler code here
	// 绘图使能标识取反
	m_bDraw = !m_bDraw;
}
```

```
void CMainFrame::OnUpdateSettingsDrawenable(CCmdUI *pCmdUI)
{
	// TODO: Add your command update UI handler code here
	// 根据 m_bDraw 的值设置是否选中
	pCmdUI->SetCheck(m_bDraw);
}
```

5. 运行程序，效果图如下：



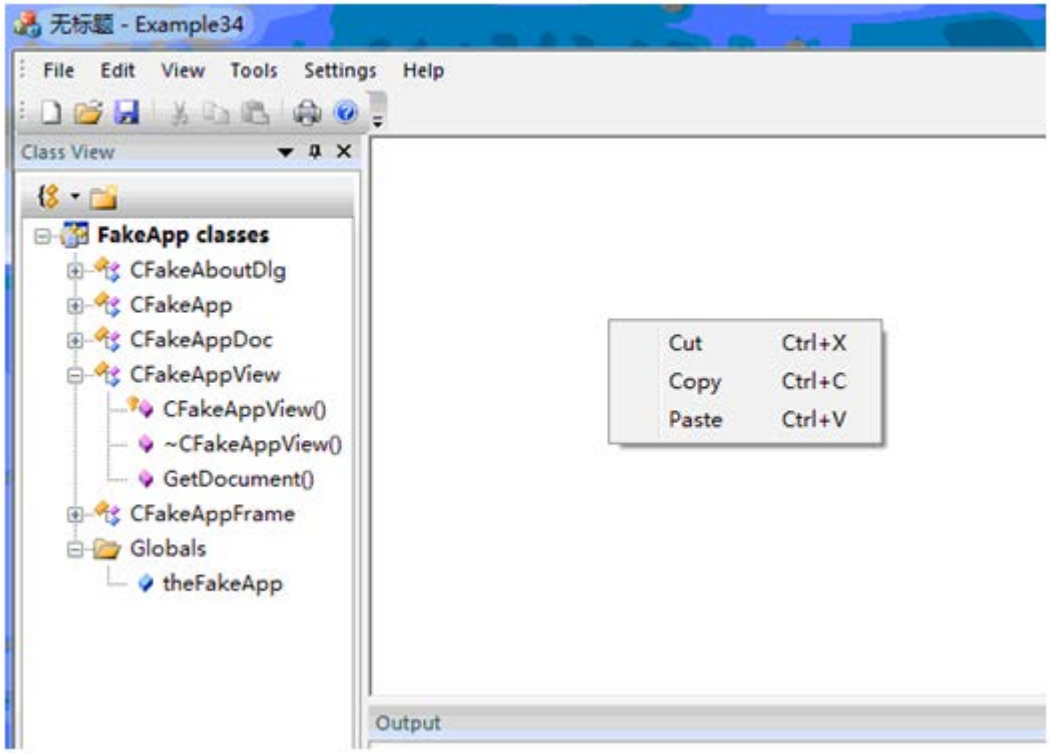
6. 接下来我们要重新实现右键菜单。大家以后可以仿照 VS2010 自动生成的代码实现右键菜单，也可以用鸡啄米下面讲到的方法。首先将 CExample34View::OnRButtonUp(UINT /* nFlags */, CPoint point)函数内的代码都注释掉，保证原来的弹出方法失效。

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

```
void CExample34View::OnRButtonDown(UINT nFlags, CPoint point)
{
	// TODO: Add your message handler code here and/or call default
	CMenu menu;		// 菜单（包含主菜单栏和子菜单）
	CMenu *pSubMenu;	// 右键菜单

	// 加载菜单资源到 menu 对象
	menu.LoadMenu(IDR_POPUP_EDIT);
	// 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
	pSubMenu = menu.GetSubMenu(0);
	// 将坐标值由客户坐标转换为屏幕坐标
	ClientToScreen(&point);
	// 弹出右键菜单，菜单左侧与 point.x 坐标值对齐
	pSubMenu->TrackPopupMenu(TPM_LEFTALIGN, point.x, point.y, this);

	CView::OnRButtonDown(nFlags, point);
}
```

8. 最终的右键菜单效果：



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，这一节讲与菜单有密切联系的工具栏。

工具栏简介

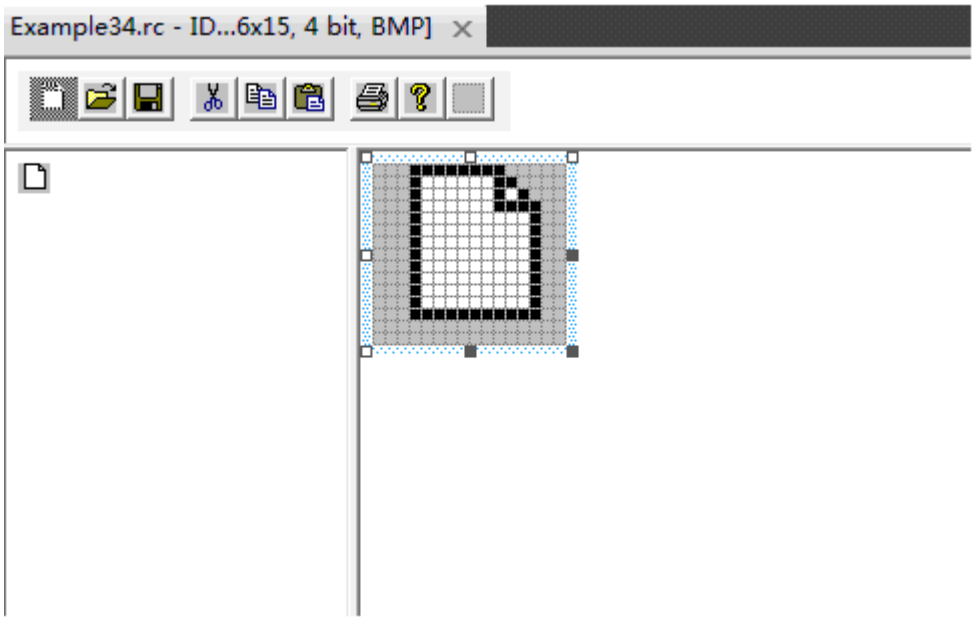
工具栏一般位于主框架窗口的上部，菜单栏的下方，由一些带图片的按钮组成。当用户用鼠标单击工具栏上某个按钮时，程序会执行相应的操作，如果鼠标没有点击，只是停留在某个按钮上一会后，会弹出一个小窗口显示提示信息。

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）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 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，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）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

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
       if (CFrameWndEx::OnCreate(lpCreateStruct) == -1)
     return -1;
     ......略

       // 调用CreateEx函数创建工具栏，并调用LoadToolBar函数加载工具栏资源
       if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP |
CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
     !m_wndToolBar.LoadToolBar(theApp.m_bHiColorIcons ? IDR_MAINFRAME_256 :
IDR_MAINFRAME))
       {
     TRACE0("Failed to create toolbar\n");
     return -1;     // fail to create
       }

       ......略

       return 0;
}
```

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

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CFrameWndEx::OnCreate(lpCreateStruct) == -1)
	return -1;

	.....略

	// 调用 CreateEx 函数创建工具栏，并调用 LoadToolBar 函数加载工具栏资源
	if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
	!m_wndToolBar.LoadToolBar(theApp.m_bHiColorIcons ? IDR_MAINFRAME_256 : IDR_MAINFRAME))
	{
	TRACE0("Failed to create toolbar\n");
	return -1;	// fail to create
	}

	.....略

	// TODO: Delete these five lines if you don't want the toolbar and menubar to be dockable
	m_wndMenuBar.EnableDocking(CBRS_ALIGN_ANY);
	// 为 m_wndToolBar 启用停靠
	m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
	// 为框架窗口启用停靠
	EnableDocking(CBRS_ALIGN_ANY);
```

```
    DockPane(&m_wndMenuBar);
  // 停靠工具栏
    DockPane(&m_wndToolBar);

  ......略

    return 0;
}
```