# MFC的概念和作用

## MFC的概念和作用什么是MFC?

全称是Microsoft Foundation Class Library，我们称微软基础类库。

它封装了windows应用程序的各种API以及相关机制的C++类库总结：

1.MFC是一个大的类库

2.MFC是一个应用程序框架

## 为什么需要MFC？

基于应用程序框架之上，使用提供的类库快速开发，提高项目的开发效率，减少开发周期和成本。

MFC类库常用的头文件

afx.h-将各种MFC头文件包含在内

afxwin.h-包含了各种MFC窗口类。包含了afx.h和windows.h。

afxext.h-提供了扩展窗囗类的支持，例如工具栏，状态栏等。

## MFC的控制台程序

和win32的控制台程序的差别

main函数不同于普通的控制台程序

CWinApp theApp；多了一个全局对象

经验之谈：

以Afx开头可以确定为MFC库中的全局函数。

以：开头可以确定为win32的API函数。

## MFC的库程序·

使用MFC制作自己的静态库程序

MFC的动态库

使用静态的MFC库制作自己的动态库

使用动态的MFC库制作自己的动态库

MFC扩展库

规则库可以被各种程序所调用扩展库只能被MFC程序调用

## MFC的窗口程序

相关函数

·单文档视图架构程序

CWinApp-应用程序类，负责管理应用程序的流程

CFrameWnd-框架窗口类，负责管理框架窗口

CView-视图窗口类，负责显示数据

CDocument-文档类，负责管理数据

·多文档视图架构程序

CWinApp-应用程序类

CMDIFrameWnd-多文档主框架窗口类

CMDIChildWnd-多文档子框架窗口类

CView-视图窗口类，显示数据

CDocument-文档类，管理数据

·对话框应用程序

CWinApp-应用程序类

CDialog-对话框窗口类

## MFC库中相关类简介

·CObject类MFC类库中绝大部分类的父类，提供了MFC类库中一些基本的机制。

对运行时类信息的支持

对动态创建的支持

对序列化的支持

·CWinApp类应用程序类，封装了应用程序、线程等信息。

·CDocument类，文档类，管理数据

·Frame Windows,框架窗口类，封装了窗口程序组成的各种框架窗口

·CSplitterWnd-用来完成拆分窗囗的类

·Control Bars-控件条类

·Dialog Boxes-对话框类，封装了各种对话框，通用的对话框

·Views-视图类，封装了各种显示窗口

·Controls-控件类，封装了各种常用的控件

·Exceptions-异常处理类，封装了MFC中常用的各种异常

·File-文件类，各种文件的I/O操作等

·绘图类，包括CDC类和CGdiObject类。

·数据集合类，CArray/Clist/CMap,封装了相应的数据结构的管理

·非CObject类的子类

提供了各种数据结构相关的管理，CPoint，CTime，CString等

## 把Win32程序变为MFC程序：

设置开发环境

·头文件为<afxwin.h>

·setting当中设置使用MFC库

总结：win32程序和MFC程序区别仅仅为能不能使用MFC而已。

## MFC程序执行流程

程序的启动，构造theApp对象，调用父类CWinApp的构造函数。

将theApp对象的地址保存到线程状态信息中

将theApp对象的地址保存到模块状态信息中

进入WinMain函数，调用AfxWinMain函数

进入入口函数WinMain

获取应用程序类对象theApp的地址

利用theApp地址调用InitApplication，初始化当前应用程序的数据

利用theApp地址调用Initlnstance函数初始化程序，在函数中我们创建窗口并显示。

利用theApp地址调用CWinApp的Run函数进行消息循环

如果没有消息，利用theApp地址调用Onldle虚函数实现空闲处理

程序退出利用theApp地址调用Exitinstance虚函数实现退出前的善后处理工作

CWinApp的成员成员虚函数

Initlnstance-程序的初始化函数，完成了窗囗创建等初始化处理

ExitInstance-程序退出时调用，清理资源等善后工作

Run-消息循环

Onldle-空闲处理函数

成员变量

m\_pMainWnd-当前应用程序的主窗口

# MFC窗口和消息

钩子函数（Win32函数）

创建钩子

HHOOK SetWindowsHookEx(

int idHook,//钩子类型（WH\_CBT）

HOOKPROC Ipfn,//钩子处理函数

HINSTANCE hMod,//应用程序实例句柄

DWORD dwThreadld //线程ID

）；

钩子处理函数

LRESULT CALLBACK CBTProc(

int nCode,//钩子码（HCBT\_CREATEWND） 对应钩子类型

WPARAM wParam,//刚刚创建成功的窗口句柄

LPARAM IParam //..…

}

更改窗囗处理函数

LONG\_PTR SetWindowLongPtr(

HWND hWnd,//窗口句柄

int nIndex,//GWLP\_WNDPROC

LONG\_PTR dwNewLong //新的窗口处理函数名（函数地址）

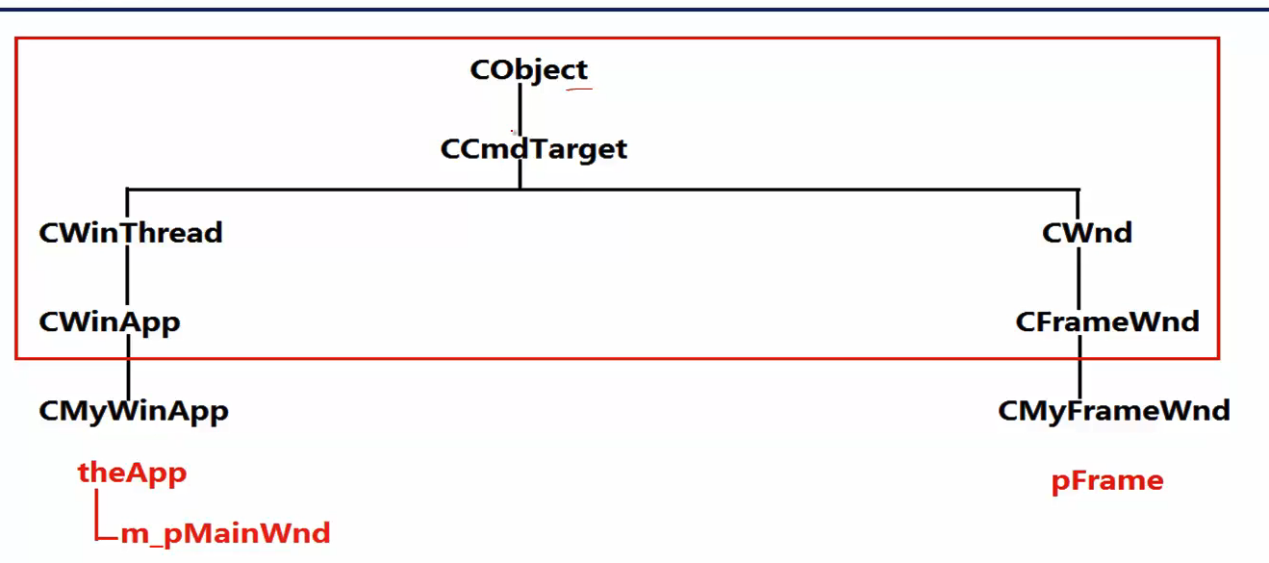
}

代码书写

定义自己的框架类（CMyFrameWnd），派生自CFrameWnd

定义自己的应用程序类（CMyWinApp)，派生自CWinApp，并重写父类成员虚函数InitInstance

定义全局对象CMyWinApp theApp；（爆破点）



## MFC窗口如何创建

\*加载菜单

\*调用CWnd::CreateEx函数创建窗口

调用PreCreateWindow函数设计和注册窗口类调用AfxDeferRegisterClass函数，在这个函数中设计窗囗类：

WNDCLASS wndcls;//设计窗囗类

...

∥定义窗囗的处理函数为DefWindowProc

wndcls.IpfnWndProc=DefWindowProc;

调用\_AfxRegisterWithlcon函数

在函数内部，加载图标，并调用AfxReqisterClass函数，在函数内部，调用::RegisterClass win32ApI函数注册窗口类

\*调用AfxHookWindowCreate函数。

在函数内部，调用SetWindowsHookEx创建WH\_CBT类型的钩子，钩子的处理函数是\_AfxCbtFilterHook。

将框架类对象地址（pFrame)保存到当前程序线程信息中

\*调用CreateWindowEx函数创建窗口，马上调用钩子处理函数

\*钩子处理函数AfxCbtFilterHook

将窗口句柄和框架类对象地址建立一对一的绑定关系。

使用SetWindowLong函数，将窗口处理的函数设置AfxWndProc

# MFC消息

MFC消息处理

当收到消息时，进入AfxWndProc函数。

AfxWndProc函数根据消息的窗口句柄，查询对应框架类对象的地址（pFrame）。

利用框架类对象地址（pFrame）调用框架类成员虚函数WindowProc，完成消息的处理。

## 消息映射机制的作用

在不重写WindowProc虚函数的大前提下，仍然可以处理消息。

消息映射机制的使用

类必须具备的要件类内必须添加声明宏 DECLARE\_MESSAGE\_MAP()

类外必须添加实现宏BEGIN\_MESSAGE\_MAP(theClass,baseClass)

END\_MESSAGE\_MAP()

总结：当一个类具备上述两个要件，这个类就可以按照消息映射机制来处理消息。

消息映射机制的具体实施

以WM\_CREATE消息为例：

BEGIN\_MESSAGE\_MAP(..)和END\_MESSAGE\_MAP0之间添加ON\_MESSAGE(WM\_CREATE,OnCreate)宏

在CMyFrameWnd类内添加OnCreate函数的声明和定义

数据结构

结构体

struct AFX\_MSGMAP\_ENTRY(静态数组每个元素的类型）{

UINT nMessage;//消息

IDUINT nCode;//通知码

UINT nID;//命令

IDUINT nLastID;//最后一个命令

IDUINT\_PTR nSig;//处理消息的函数类型

AFX\_PMSG pfn;//处理消息的函数名（地址）

}；

结构体

struct AFX\_MSGMAP（静态变量的类型）{

const AFX\_MSGMAP\*(PASCAL\* pfnGetBaseMap)0;

∥父类宏展开的静态变量地址

const AFX\_MSGMAP\_ENTRY\* IpEntries;

∥本类宏展开的静态数组首地址

}；

## 宏展开各部分的作用

GetThisMessageMap0静态函数

作用：定义静态变量和静态数组，并返回本类静态变量地址（获取链表头）\_messageEntries[ ]静态数组（进程级生命周期）

作用：数组每个元素，保存的为消息ID和处理消息的函数名（地址）

messageMap 静态变量（进程级生命周期）

作用：第一个成员，保存父类宏展开的静态变量地址（负责连接链表）

第二个成员，保存本类的静态数组首地址

GetMessageMap)虚函数

作用：返回本类静态变量地址（获取链表头）

## 创建窗囗的原理

·消息产生进入窗口处理函数（AfxWndProc)

·根据已知窗口句柄，找到和它绑定在一起的框架类对象地址（pFrame）。

·利用框架类对象地址（pFrame）调用框架类成员虚函数WindowProc

·获取本类对应的静态变量，并到对应数组中匹配查找。

·如果没有找到获取父类对应的静态变量，并到对应数组中匹配查找。

·如果找到了，利用找到的数组元素的最后一个成员，并调用之，完成消息处理。

## 消息的分类

标准windows消息 ON\_WM\_XXX

自定义消息 ON\_MESSAGE

命令消息 ON\_COMMAND

## 菜单相关问题

Win32-HMENU

MFC-CMenu类对象

菜单相关类

CMenu类简介封装了关于菜单的各种操作成员函数，另外还封装了一个非常重要的成员变量m\_hMenu（菜单句柄）

菜单的使用

添加菜单资源·将菜单设置到窗口

（1）利用pFrame调用Create函数时，传参。

（2）在处理框架窗口的WM\_CREATE消息时CMenu menu；menu.LoadMenu(..)；

CMenu类简介

封装了关于菜单的各种操作成员函数，另外还封装了一个非常重要的成员变量m\_hMenu（菜单句柄）

消息的处理

ON\_COMMAND(菜单项ID，处理消息的函数名）

设置菜单项状态

Win32

WM\_INITMENUPOPUP

：：CheckMenultem/::EnableMenultem

MFC

ON\_WM\_INITMENUPOPUP

CMenu::CheckMenultem/CMenu:EnableMenultem

设置菜单项状态(右键菜单)

Win32

WM\_ CONTEXTMENU

::TrackPopupMenu

MFC

ON WM CONTEXTMENU

CMenu::TrackPopupMenu

## 工具栏相关类

CToolBarCtrl

-父类CWnd，封装了关于工具栏控件的各种操作。

CToolBar

-父类CControlBar，封装了关于工具栏的操作，以及和框架窗口的关系。

工具栏的使用

添加工具栏资源

创建工具栏 CToolBar：：CreateEx

加载工具栏 CToolBar：：Load ToolBar

设置工具栏的停靠

CToolBar::EnableDocking

CFrameWnd::EnableDocking

CFrameWnd::DockControlBar

运行时类信息机制的使用

·类必须派生自CObject

·类内必须添加声明宏 DECLARE\_DYNAMIC(theClass)

·类外必须添加实现宏 IMPLEMENT\_DYNAMIC(theClass，baseClass)

当一个类具备上述三个要件后，CObject：：IsKindOf函数就可以正确判断对象是否属于某个类。

数据结构

struct CRuntimeClass(宏展开的静态变量的类型）{

LPCSTR m\_lpszClassName;//类名称

int m\_nObjectSize;//类大小

UINT m\_wSchema;//类版本（统一为0xFFFF）

CObject\*(PASCAL\* m\_pfnCreateObject)0;/动态创建机制使用，这里为NULL CRuntimeClass\* m\_pBaseClass;//父类宏展开静态变量地址

CRuntimeClass\* m\_pNextClass;//不使用为NULL

const AFX\_CLASSINIT\* m\_pClassInit;//不使用为NULL }；

宏展开各部分作用

·classCDog 静态变量保存为类名和类大小等信息，以及父类静态变量地址（负责连接链表）

·GetRuntimeClass()虚函数获取本类的静态变量地址（获取链表头结点）

执行过程

·一个非常非常非常重要的宏

RUNTIME\_CLASS(theClass)->&theClass::classtheClass获取theClass这个类的静态变量地址

·利用对象（yellowdog）的地址调用宏展开的虚函数GetRuntimeClass0获取本类静态变量的地址（链表头）

·利用本类静态变量的地址（链表头）和目标进行比对。

·如果相同，证明对象属于这个类。

·如果不相同获取链表的下一个结点（父类静态变量地址）循环比对，只要有一次相同也能证明对象属于这个类。循环结束一次都没有比对成功，证明对象不属于这个类。

动态创建机制的使用

·类必须派生自CObject

·类内必须添加声明宏DECLARE\_DYNCREATE( theClass)

·类外必须添加实现宏 IMPLEMENT\_DYCREATE(theClass，baseClass)

当一个类具备上述三个要件后，CRuntimeClass：：CreateObject（对象加工厂）函数就可以将类的对象创建出来。

与运行时类信息机制区别

·多了一个静态函数

CObject\* PASCAL CDog:CreateObject（）{

return new CDog;//new了一个CDog类对象，并返回对象地址。

}

·静态变量的第四个成员不再为NULL，而为新增加的这个静态函数的地址

执行过程

·利用本类（CDog）的静态变量，调用CRuntimeClass的成员函数CreateObject（对象加工厂函数）

·获取静态变量的第四个成员（新增加静态函数），并调用之。

·新增加的静态函数内部，完成类对象的创建，并返回对象地址。

# 视图窗口

## 视图窗口的相关类

·CView及其子类，父类为CWnd类，封装了关于视图窗口的各种操作，以及和文档类的数据交互

## 视图窗口的使用

·定义一个自己的视图类（CMyView），派生自CView，并重写父类成员纯虚函数OnDraw(可以用于绘图）

·其余框架类和应用程序类代码不变。

·在处理框架窗口的WM\_CREATE消息时，定义CMyView类对象，并调用Create函数创建视图窗口

视图窗口的ID为AFX\_IDW\_PANE\_FIRST

## 对象关系图

m\_pMainWnd=pFrame;

m\_pViewActive=pView;

·theApp

|->m\_pMainWnd(pFrame 框架类对象地址）

|->m\_pViewActive(pView 视图类对象地址）

# 文档类

## 文档类的作用

·相关类 CDocument

提供了一个用于管理数据的类，封装了关于数据的管理（数据提取，数据转换，数据存储等），并和视图类进行数据交互。

## 文档类的使用

·定义一个自己的文档类（CMyDoc），派生自CDocument

·其余具体见代码

## 程序的创建过程

·复习内容

WM\_CREATE消息的两个附带信息

wParam-不使用

IParam-传来的信息为指针（CREATESTRUCT类型），

IParam-这个指针指向位置保存了创建窗口函数（：：CreateWindowEx）的全部12个参数

·结合消息映射机制

ON\_WM\_CREATE()处理

WM\_CREATE消息的函数为：

afx msg int OnCreate(LPCREATESTRUCT pcs);

·利用框架类对象地址（pFame）调用LoadFrame函数，创建框架窗口

·在处理框架窗口的WM\_CREATE消息时，动态创建视图类对象，并创建视图窗口。

·在处理视图窗口的WM\_CREATE消息时，将文档类对象和视图类对象建立关联关系。

## 对象关系图

·theApp

|->m\_pMainWnd（框架类对象地址pFrame)

|->m\_pViewActive（视图类对象地址pView)

|->m\_pDocument（文档类对象地址pDoc）

|->m\_viewList（所有视图类对象地址）

·文档类对象用一个链表成员变量，保存视图类对象地址

·视图类对象用一个普通成员变量，保存文档类对象地址

经过分析可知一个文档类对象可以对应多个视图类对象（视图窗口），而一个视图类对象（视图窗口）只能对应一个文档类对象。

## 窗口切分

相关类CSplitterWnd

-不规则框架窗口类，封装了关于不规则框架窗口的操作。

窗口切分的使用

重写CFrameWnd类的成员虚函数OnCreateClient

在虚函数中调用CSplitterWnd:CreateStatic创建不规则框架窗口

在虚函数中调用CSplitterWnd:CreateView创建视图窗口。

## 命令消息处理顺序

·theApp

|->m\_pMainWnd（框架类对象地址pFrame)

|->m\_pViewActive（视图类对象地址pView）

|->m\_pDocument(文档类对象地址pDoc)

|->m\_viewList（所有视图类对象地址）

视图类>文档类>框架类>应用程序类（默认顺序）《可手动调换》

## 文档类和视图类关系

·theApp

|->m\_pMainWnd（框架类对象地址pFrame)

|->m\_pViewActive（视图类对象地址pView)

|->m\_pDocument（文档类对象地址pDoc）

|->m\_viewList（所有视图类对象地址）

文档类和视图类的关系

视图类成员函数

获取和视图类对象关联的文档类对象，调用GetDocument0

文档类成员函数

当文档类数据发生变化时，调用UpDataAllViews刷新和文档类对象相关联的视图类对象（视图窗口）

# 单文档视图

单文档视图架构特点

·只能管理一个文档（只有一个文档类对象）。

## 单文档视图架构使用

参与架构的类

CFrameWnd/CWinApp/CView/Cdocument

需要用到的类

CDocTemplate(文档模板类）

|->CSingleDocTemplate(单文档模板类）

CDocManager - 文档管理类

执行过程

theApp

|>m\_pDocManager(文档管理类对象地址）

|->m\_templateList(单文档模板类对象地址）

|->

CSingleDocTemplate\*pTemplate

|->m\_pOnlyDoc//(唯一的文档类对象地址） I->m\_pDocClass//RUNTIME\_CLASS(CMyDoc) I->m\_pFrameClass//RUNTIME\_CLASS(CMyFrameWnd)

|->m\_pViewClass//RUNTIME\_CLASS(CMyView)

OnFileNew的执行过程：

1》利用theApp获取文档管理类对象地址。

2》利用文档管理类对象获取单文档模板类对象地址。

3》利用单文档模板类对象获取 RUNTIME\_CLASS（CMyFrameWnd)，调用CreateObject(对象加工厂函数）动态创建框架类对象。

4》利用单文档模板类对象获取RUNTIME\_CLASS（CMyDoc)，调用CreateObject（对象加工厂函数）动态创建文档类对象。

5》利用框架类对象调用LoadFrame函数，创建框架窗口。

# 多文档视图架构

多文档视图架构特点

·可以管理多个文档（可以有多个文档类对象）。

## 多文档视图架构使用

·参与架构的类

CMDIFrameWnd /CMDIChildWnd /CWinApp/ CView/ CDocument

·需要用到的类

CDocTemplate(文档模板类）

|->CMultiDocTemplate(单文档模板类）

CDocManager-文档管理类

执行过程

theApp

|->m\_pDocManager(文档管理类对象地址）

|->m\_templateList(单文档模板类对象地址）

|->

CMultiDocTemplate\*p Template

|->m\_docList/（保存多个文档类对象地址） |->m\_pDocClass//RUNTIME\_CLASS(CMyDoc)

I->m\_ pFrameClass//RUNTIME\_CLASS(CMyChild) I>m\_pViewClass//RUNTIME\_CLASS(CMyView)

OnFileNew的执行过程：

1》利用theApp获取文档管理类对象地址。

2》利用文档管理类对象获取单文档模板类对象地址。

3》利用单文档模板类对象获取RUNTIME\_CLASS（CMyChild），调用CreateObject(对象加工厂函数）动态创建框架类对象。

4》利用单文档模板类对象获取RUNTIME\_CLASS（CMyDoc)，调用

CreateObject(对象加工厂函数）动态创建文档类对象。

5》利用框架类对象调用LoadFrame函数，创建子框架窗口。

# MFC绘图

相关问题

WINDOWS下的绘图，需要绘图设备

·WIN32绘图设备句柄（HDC）

·MFC类对象

## 绘图相关类

·CDC类（绘图设备类）

封装了各种绘图相关的函数，以及两个非常重要的成员变量m\_hDC和m\_hAttribDC

CPaintDC类，封装了在WM\_PAINT消息中绘图的绘图设备

CClientDC类，封装了在客户区绘图的绘图设备

·CGdiObject类（绘图对象类）

封装了各种绘图对象相关的操作，以及一个非常重要的成员变量m\_hObject（绘图对象句柄）

CPen类，封装了画笔的操作

CBrush类，封装了画刷的操作

CFont类，封装了字体的操作

CBitmap,封装了位图的操作

# 序列化机制

## 文件操作相关类

CFile-文件操作类，封装了关于文件读写等操作。

CFile:Open

CFile:Write/Read

CFile:Close

CFile:SeekToBegin/SeekToEnd/Seek

# 序列化基本类型

序列化机制的作用

·以二进制流形式读写硬盘文件。但效率很高。

序列化机制相关类

·CFile-文件操作类，完成硬盘文件的读写操作。

·CArchive-归档类，完成内存数据的读写操作。

序列化机制的使用

·创建或打开文件 CFile:Open

·定义归档类对象 CArchive ar；

·数据序列化（存储/写）ar<<数据

·关闭归档类对象 ar.Close)

·关闭文件 CFile:Close)；

反序列化机制的使用

·打开文件 CFile:Open

·定义归档类对象 CArchive ar；

·数据反序列化（加载/读）ar>>变量

·关闭归档类对象 ar.Close)

·关闭文件 CFile:Close)；

序列化和反序列化执行过程

·数据结构

class CArchive{

enum Mode{store=0,load=1...};

BOOL m\_nMode;//访问方式

int m\_nBufSize;//buff的大小

CFile\* m\_pFile;//操作的文件对象

BYTE\* m\_lpBufCur;//当前指向

BYTE\*m\_lpBufMax;//终止指向

BYTE\* m\_IpBufStart;//开始指向

...

}

序列化执行过程

·ar对象维护一个缓冲区。

·将各个数据依次序列化（存储）到ar对象维护的缓冲区中，并将m\_lpBufCur的指针指向移动相应字节。

·如果ar维护的缓冲区不足，则将ar维护的缓冲区的数据写入硬盘文件，并重置m\_lpBufCur为开始指向。

·当关闭ar对象时，将ar对象维护的缓冲区数据写入硬盘文件，并释放ar对象维护的缓冲区。

反序列化执行过程

·ar对象维护一个缓冲区。

·当反序列化第一个数据时，将文件数据全部读取到ar维护的缓冲区中，并将第一个数据反序列化到第一个变量，并将m\_lpBufCur移动相应的字节数。

·依次反序列化每个数据到变量中。

·当关闭ar对象时，释放ar维护的缓冲区。

## 序列化类对象使用

序列化类对象的本质

·序列化类对象，就是序列化对象的各个成员变量

序列化类对象的使用

·类必须派生自CObject

·类内必须添加声明宏 DECLARE\_SERIAL(theClass)

·类外必须添加实现宏 IMPLEMENT\_SERIAL(theClass，baseClass,1)

·类必须重写虚函数Serialize

当类具备上述三个要件后，类对象就可以序列化到文件中保存了。

# 对话框

## WIN32对话框回顾

间接方式创建无模式对话框

·添加对话框资源

·查找资源FindResource

·加载资源LoadResource

·锁定资源LockResource

·创建无模式对话框CreateDialogIndirect

销毁无模式对话框

·销毁无模式对话框，使用DestroyWindow函数，不能使用EndDialog函数，因为EndDialog函数只能将无模式对话框隐藏，不能销毁。

## MFC对话框分类

模式对话框（假）/无模式对话框

### 无模式对话框架构使用

参与架构的类 CDialog/CWinApp

代码书写·添加对话框资源。

定义一个自己的对话框类（CMyDIg），管理对话框资源，派生自CDialog或CDialogEx均可。

其余代码，详见代码。

执行过程

·程序启动进入入口函数WinMain

获取应用程序类对象theApp的地址

利用theApp地址调用InitApplication，初始化当前应用程序的数据

利用theApp地址调用Initlnstance函数，在函数中我们创建无模式对话框并显示。

利用theApp地址调用CWinApp的Run函数进行消息循环。

如果没有消息，利用theApp地址调用OnIdle虚函数实现空闲处理

当对话框销毁（必须利用DestroyWindow)，消息循环才可退出。

退出前利用theApp地址调用ExitInstance虚函数实现退出前的善后处理工作。

### 模式对话框（假

参与架构的类 CDialog/CWinApp

代码书写·添加对话框资源。

·定义一个自己的对话框类（CMyDlg），管理对话框资源，派生自CDialog或CDialogEx均可。

·其余代码，详见代码。

执行过程

程序启动进入入口函数WinMain

获取应用程序类对象theApp的地址

利用theApp地址调用InitApplication，初始化当前应用程序的数据

利用theApp地址调用Initlnstance函数，在函数中我们调用DoModal函数。

DoModal内部执行过程

将父窗口设置为不可用

创建无模式对话框

进入消息循环（自带的）

退出消息循环（父类的OnOK/OnCancel导致循环退出）

将父窗口设置为可用状态

销毁无模式对话框

不再执行CWinApp的Run函数进行的消息循环。

程序结束。

# MFC 控件的使用

## 对象和控件绑定

绑定的作用

·将控件窗口和类对象绑定具有两大作用

如果和数据类对象绑定，对象和控件可以进行数据交换。

如果和控件类对象绑定，对象就可以代表整个控件。

与数据类型对象绑定的使用

数据类型对象和控件可实现数据交互。

重写父类成员虚函数DoDataExchange在函数内部通过一系列的DDX\_xxx函数，实现控件和数据类型对象的数据交互。

如果需要实现数据交互，调用UpdateData函数

UpdateData(TRUE):控件>变量

UpdateData(TRUE):变量>控件

与控件类型对象绑定

控件类型对象和控件可实现对象代表整个控件。

重写父类成员虚函数DoDataExchange在函数内部通过一系列的DDX\_xxx函数，实现控件句柄和控件类型对象的绑定。

控件类型对象，就代表这个控件。

控件消息的基本原理·

控件消息的处理

WM\_COMMAND消息

LOWORD(wParam)-菜单项ID，控件ID

HIWORD(wParam)-菜单项为0控件，通知码（控件发生的事情）

IParam-均无用

## 基本控件（一）

关于控件的介绍

控件如何与数据类型对象绑定（做数据交换）。

控件如何与控件类对象绑定（对象代表控件）。

控件的消息如何处理。

下压式按钮

相关类

CButton类：父类为CWnd，封装了关于下压式按钮控件的各种操作。

消息

当下压式按钮被单击WM\_COMMAND消息（通知码BN\_CLICKED）

**编辑框**

·相关类

CEdit类：父类为CWnd，封装了关于编辑框控件的各种操作。

·常见操作

获取编辑框文本内容

设置编辑框文本内容

·消息

当编辑框文本内容发生变化WM\_COMMAND消息（通知码EN\_CHANGE）

## 基本控件（二）

**复选框按钮**

相关类

CButton类：父类为CWnd，封装了关于复选框按钮控件的各种操作。

常见操作

获取复选框状态CButton:GetCheck

设置复选框状态CButton:SetCheck

消息

当下压式按钮被单击WM\_COMMAND消息（通知码BN\_CLICKED）

**单选框按钮**

相关类

CButton类：父类为CWnd，封装了关于单选框按钮控件的各种操作。

常见操作

获取复选框状态CButton:GetCheck

设置复选框状态CButton:SetCheck

消息

当下压式按钮被单击WM\_COMMAND消息（通知码BN\_CLICKED）

**分组框按钮**

·常见操作

用于控件的分组显示，提高界面友好性。

**文本静态框**

·常见操作

常用于显示字符串，提高界面友好性CWnd::SetWindowText

**图像静态框**

常见操作

常用于显示图像CStatic:Seticon/CStatic:SetBitmap

## 基本控件（三）

**组合框**

·分类

简单式组合框/下拉式组合框/下拉列表式组合框

·相关类

CComboBox类：父类为CWnd，封装了关于组合框控件的各种操作。

·常见操作

添加选项CComboBox::AddString

删除选项CComboBox:DeleteString

获取选择项索引CComboBox:GetCurSel

设置选择项 CComboBox:SetCurSel

获取选项文本内容 CComboBox:GetLBText

设置选项附加数据 CComboBox:SetltemData

获取选项附加数据 CComboBox:GetltemData

.消息

当选择项发生变化时

WM\_COMMAND消息（通知码CBN\_SELCHANGE)

当文本内容发生变化时

WM\_COMMAND消息（通知码CBN\_EDITCHANGE）

**列表框**

相关类

CListBox类:父类为CWnd,封装了关于列表框控件的各种操作

常见操作

添加选项CListBox: Addstring

删除选项CList Box. DeleteStrin

获取选择项CListBox∷ GetCurSel

设置选择项 CListBox SetCurSel

获取选项文本内容 CListBox Gettem Text

设置选项附加数据 CListBox: SetltemData

获取选项附加数据 CListBox: GetItem Data

消息

当选择项发生变化时WM\_ COMMAND消息(通知码 LBN SELCHANGE)

## 基本控件 (四)

**滑块控件**

●相关类

CSliderCtrl类:父类为CWnd,封装了关于滑块控件的各种操作。

●常见操作

设置范围CSliderCtrl: SetRange

设置位置CSliderCtrl: SetPos

获取位置CSliderCtrl: GetPos

●消息

当滑块的位置改变后

滑块控件发出 WM COMMAND消息(通知码 NM CUSTOMDRAW)

**进度条控件**

相关类

CProgressCtrl类:父类为CWnd,封装了关于进度条控件的各种操作。

常见操作

设置范围CProgressCtrl: SetRange

设置位置CProgressCtrl: SetPos

获取位置CProgressCtrl: GetPos

**文件搜索类**

●相关类

CFilefind类:封装了关于文件遍历的各种操作。

●常见操作

开启查找,成功返回TRUE,失败返回

FALSECFile Find: Find File

找到当前文件,返回下一个文件是否存在

CFile Find- Find NextFile

调用一系列 GetXXXI函数获取文件信息

CFile Find: GetFileName/ CFile Find: GetFilePath

调用一系列IsXXX函数判断文件属性

例如: CFile Find: IsDirectory/ CFile Find:sDot

关闭查找

例如: CFile find: Close

## 基本控件 (五)

**列表控件**

●相关类

CListCtrl类:父类为CWnd,封装了关于列表控件的各种操作。

●风格

图标( LVS ICON)

小图标( LVS SMALLICON)

列表( LVS LIST)

报表( LVS REPORT)

修改风格CWnd: Modifystyle

●常见操作

添加选项 CListctrl∵ Inserter

删除选项 CListCtrl: Deleteltem/DeleteAllItem

添加列 CListctrl: InsertColumn

设置文本内容 CListctrl. SetItem Text

设置图像 CListCtrl: SetImageList

设置附加数据 CListctrl: SetItemData

获取附加数据 CListCtrl- GetItemData

●消息

当列表控件被双击 发出 WM COMMAND消息(通知码 NM DBLCLK)

**树控件**

●相关类

CTreeCtrl类:父类为CWnd,封装了关于树控件的各种操作。

●常见操作

添加选项 CTreeCtrl:InsertItem

删除选项 CTreeCtrl: DeleteItem/ DeleteAllItems

获取选择项 CTreeCtrl GetSelectedItem

获取选项的文本内容 CTreeCtrl- GetItem Text

设置选项的高度 CTreeCtrl∴ GetItem Text

设置选项的图像 CTreeCtrl: SetImageList