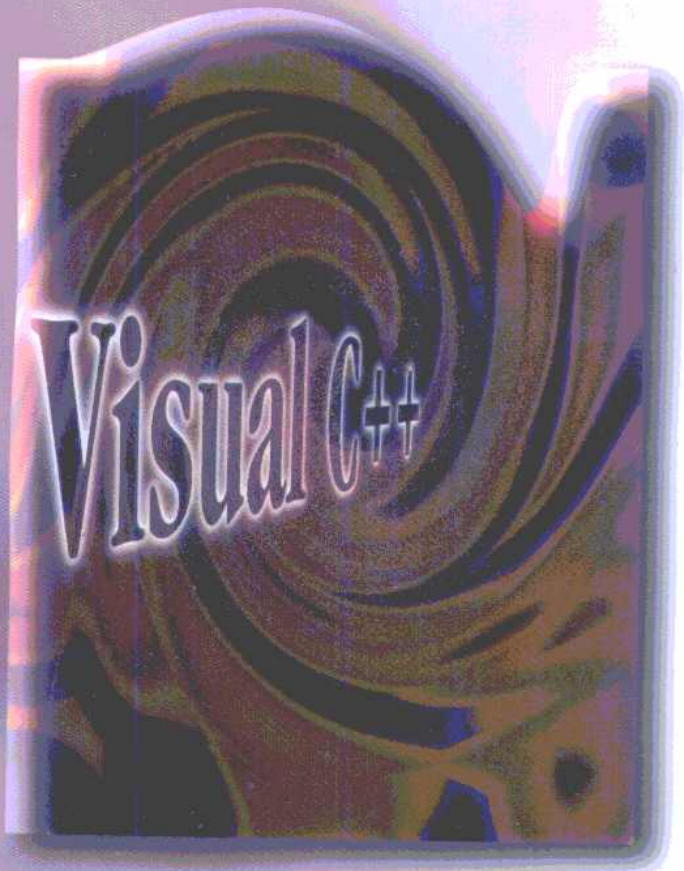
计算机语言函数应用丛书

Visual C++

计算机语言函数应用

童长武 王 博 贾春丽 魏居作 编著



科学出版社

计算机语言函数应用丛书

Visual C++

计算机语言函数应用

童长武　王　博　贾春丽　魏启佐　编著

科学出版社
2000

内 容 简 介

Visual C++伴随 Wintel 体系的日益巩固而风靡全球。本书按功能和类别对 Visual C++ 6.0 类进行了归纳总结，并具体分析和介绍了它们的功能、用法，以及所在头文件、成员函数的返回值及其参数调用等信息。对于重要的成员函数，结合作者多年来开发 Visual C++ 程序的经验和体会，给出了简单易懂的实例。对于 C 语言库函数，由于目前相关参考书较多，这里不作介绍。本书的特点是追求实用，通俗易懂，对使用 Visual C++ 的软件开发人员和大专院校的学生具有极高的参考价值。

图书在版编目（CIP）数据

计算机语言函数应用——Visual C++/童长武等编 . - 北京：科学出版社，2000

ISBN 7-03-008142-0

Ⅰ.计… Ⅱ.童… Ⅲ.C 语言-函数 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（1999）第 72828 号

科学出版社 出版

北京东黄城根北街 16 号

邮政编码：100717

北京双青印刷厂 印刷

新华书店北京发行所发行 各地新华书店经售

*

2000 年 3 月第 一 版 开本：787×1092 1/16

2000 年 3 月第一次印刷 印张：55

印数：1—5 100 字数：1300 000

定价：72.00 元

（如有印装质量问题，我社负责调换〈环伟〉）

前　言

Visual C++ 6.0是微软公司重点推出的核心产品之一，它在原MFC类库基础上进行了完善和扩充，功能更强，性能更优，使用更方便。MFC是指Microsoft Foundation Class(Microsoft基础类)，它封装了Windows API(应用程序接口)的数据结构、函数及宏，以面向对象的类的形式供程序开发者编写各种类型的程序，包括Internet、数据库和多媒体方面的程序。

Visual C++ 6.0类库在以下方面比前期版本功能更强。

1. Visual C++ 6.0支持动态HTML，通过AppWizard可以很轻松地编写基于Web浏览器的应用程序。新的CHtmlView类封装了Internet Explorer浏览器的功能，这样可以方便地编写浏览器风格的应用程序。

2. Visual C++ 6.0支持Windows DNA(分布式Internet应用程序)架构。DNA技术完全支持Microsoft Back Office服务器方面的控件，包括微软的SQL Server，Transaction Server和Message Server。

3. Visual C++ 6.0支持更多类型的ATL(ActiveX模板库)，包括DLL库、EXE文件、Web控件、数据库和Microsoft Back Office应用程序。

4. Visual C++ 6.0访问数据库的能力更强。它完全支持最新的微软数据库标准OLE DB，通过COleDBRecordView类可以在应用程序中方便地浏览OLE DB数据库。另外，VC 6.0可以支持Oracle 8，并且在支持Microsoft SQL 6.5数据库方面，功能更强。

本书按功能和类别对Visual C++ 6.0类进行了归纳总结，并具体分析和介绍它们的功能、用法、所在头文件、成员函数的返回值及其参数调用等信息。本书的内容如下：

- 第1章介绍根类和应用体系类。根类是微软基础类库(MFC)的原基类，大多数MFC类都是由它派生的。
- 第2章介绍应用框架体系结构方面的MFC类。这些类提供了大多数应用程序都需要的一些函数。
- 第3章介绍窗口、对话框和控件方面的类，这些是用户设计用户界面的有力工具。
- 第4章介绍绘图与打印方面的类。
- 第5章介绍简单数据类型类，包括CString、CTime等数据类型。
- 第6章介绍数组、列表、映射方面的类。
- 第7章介绍文件与数据库方面的类，包括文件类、DAO类和ODBC类。
- 第8章介绍Internet和网络类，包括ISAPI类、Windows套接字类和Internet类。
- 第9章介绍OLE类，通过OLE类与其它程序框架类可以方便地访问ActiveX API，并使用ActiveX强大的功能。
- 第10章介绍调试和异常方面类。

由于本人水平有限，再加上时间仓促，错误及不当之处在所难免，敬请广大读者批评指正。

1 根类和应用体系类

1.1 CObject 类

CObject 是 Microsoft 基础类库(MFC)的原基类。它不仅是类库中诸如 CFile 和 CDocument 类的基类,也是用户自己编写的所有类的基类。CObject 提供以下基本功能:

1. 序列化支持;
2. 运行时间信息;
3. 对象诊断输出;
4. 与集合类的兼容。

注意 CObject 不支持多继承的特性。派生类中只能有一个 CObject 基类,并且 CObject 类必须在类层次图的最左边。但是,可以通过结构体和非 CObject 派生类来进行多继承。

程序中若在类声明和类实现时使用宏定义,将会体现 CObject 类的功能。

通过第一级宏定义 DECLARE _DYNAMIC 和 IMPLEMENT _DYNAMIC 可以在运行时刻获得类名及其在类层次图中的位置。这样可以存储诊断信息。

第二级宏定义 DECLARE _SERIAL 和 IMPLEMENT _SERIAL 包括第一级宏的所有功能。通过它们的应用可以序列化文档。

构造函数/析构函数

CObject　缺省构造函数。

operator new　特殊 new 运算符。

operator delete　特殊 delete 运算符。

operator =　赋值运算符。

诊断函数

AssertValid　确认对象完整性。

Dump　产生该对象的一个诊断缓冲池。

序列化函数

IsRuntimeClass　测试该对象是否可以序列化。

Serialize　序列化文档。

杂务函数

GetRuntimeClass　返回对应于此对象类的 CRuntimeClass 结构。

IsKindOf　测试本对象与给定类之间的关系。

成员函数

CObject::CObject

```
CObject( );
```

CObject(const CObject& objectSrc);

参数:objectSrc　对另一个 CObject 对象的引用。

说明:标准 CObject 构造函数。用户派生类的构造函数运行时自动调用缺省构造函数。如果用户构造的类是可序列化的(与 IMPLEMENT _SERIAL 宏一致),那么在其声明的类中必须构造一个不带参数的缺省构造函数。如果不需要缺省构造函数,则声明一个无参数的私有或保护构造函数。标准 C++ 缺省情况下复制构造函数是成员到成员的复制。若用户构造的类需要一个拷贝构造函数却无法找到,则声明为私有的 CObject 拷贝构造函数能确保提供一个编译器错误信息提示。所以,如果用户类需要这项功能,则必须提供一个拷贝构造函数。

CObject::AssertValid

virtual void AssertValid( ) const;

说明:AssertValid 通过检查该类的内部状态来检查其合法性。在 Debug 中, AssertValid 可用来确保正确性,在诊断失败后终止程序并返回出错时所在的文件名和行号。在编写自己的类时,用户可以重载该函数,以便提供诊断服务。重载的 AssertValid 函数通常在检查派生类之前调用基类的 AssertValid 函数。由于 AssertValid 是一个 const 函数,故在诊断过程中不能改变其对象的状态。用户自己派生类的 AssertValid 函数不可抛出异常,应自己维护其合法性。

"合法性"的定义依赖于对象的类。通常,函数进行"浅检查"操作,即检查指针是否为空,但对指针所指向的对象不进行合法性检测。

CObject::Dump

virtual void Dump(CDumpContext& dc) const;

参数:dc　诊断转储设备,通常是 afxDump。

说明:将用户对象中的内容转储到 CDumpContext 对象中。用户编写自己的类时,应该重载 Dump 函数以便为该类提供诊断功能。重载的 Dump 函数在打印数据成员(对于派生类是唯一的)之前通常调用基类的 Dump 函数。如果用户使用了 IMPLEMENT _DYNAMIC 或 IMPLEMENT _SERIAL 宏,则 CObject::Dump 打印类名。

注意:Dump 函数在输出结尾不能换行。Dump 只能在 MFC 类库的调试版中使用。用户应将该函数的说明、实现及调用括在 #ifdef _DEBUG/#endif 语句中作为条件编译。此外,Dump 是个 const 函数,用户在转储过程中不能改变对象的状态。当插入 CObject 对象指针时,CDumpContext 插入符(<<)调用 Dump 函数。Dump 只允许对象的"非循环"转储。例如,在转储对象列表时,如果其中的某一对象就是该列表,就会造成堆栈溢出。

CObject::IsSerializable

BOOL IsSerializable( ) const;

返回值:若此对象可以序列化,返回非 0;否则返回 0。

说明:测试对象是否可以序列化。对序列化的类,其声明中应包含 DECLARE _SERIAL 宏,实现时包含 IMPLEMENT _SERIAL 宏。

注意:不要重载此函数。

CObject::Serialize

virtual void Serialize(CArchive& ar);

throw(CMemoryException);
throw(CArchiveException);
throw(CFileException);

参数：ar　将被序列化的对象。

说明：从文档中读进对象或将对象写到文档中。被序列化的每一个类都必须重载 Serialize 函数。重载的 Serialize 首先调用基类的 Serialize 函数。当然，用户必须在类的声明中调用 DECLARE _ SERIAL 宏，在类的实现时调用 IMPLEMENT _ SERIAL 宏。使用 CArchive::IsLoading 函数测试文档是否正被调入，使用 CArchive::IsStoring 函数测试文档是否正被保存。序列化由 CArchive::ReadObject 和 CArchive::WriteObject 函数调用。它们是与 CArchive 的插入符(< <)和提取符(> >)相关联的。

CObject::GetRuntimeClass

virtual CRuntimeClass * GetRuntimeClass( ) const;

返回值：返回与该对象对应的 CRuntimeClass 结构体指针。

说明：每一个从 CObject 继承的类都有一个 CRuntimeClass 结构体。其结构体成员如下：

- LPCSTR m _ lpszClassName　ASCII 码类名，是以'\0'结尾的字符串。
- int m _ nObjectSize　对象的大小，单位为字节。如果对象中有成员被分配内存，则该内存大小不计其内。
- UINT m _ wSchema　预定义的数目(对于不可序列化的类为 -1)。参见 IMPLEMENT _ SERIAL 宏。
- CObject * (PASCAL * m _ pfnCreateObject)( )　创建用户类对象的缺省构造函数的指针。(只有当类支持动态创建时才有效；否则返回 NULL)。
- CRuntimeClass * (PASCAL * m _ pfn _ GetBaseClass)( )　如果用户应用程序与 MFC 的 AFXDLL 动态链接，该指针指向返回基类的 CRuntimeClass 结构体的函数。
- CRuntimeClass * m _ pBaseClass　如果用户程序与 MFC 静态链接，该成员表示指向基类的 CRuntimeClass 结构体指针。

该函数应用时需要在类实现时使用 IMPLEMENT _ DYNAMIC 宏和 IMPLEMENT _ SERIAL 宏。

参见：CObject::IsKindOf, RUNTIME _ CLASS11

CObject::IsKindOf

BOOL IsKindOf(const CRuntimeClass * pClass) const;

返回值：如果对象与类相对应，返回非 0；否则返回 0。

参数：pClass 与 CObject 派生类相关联的 CRuntimeClass 结构体指针。

说明：测试一个 pClass：

(1)是否是指定类的对象；

(2)是否从指定类继承的对象。

该函数只有在声明了 DECLARE _ DYNAMIC 宏和 DECLARE _ SERIAL 宏的类中有效。不要太多使用此函数，因为它与 C + + 的多态性相抵触，应尽量使用虚拟函数。

参见：CObject::GetRuntimeClass, RUNTIME _ CLASS

运算符

CObject::operator new

void * operator new(size _ t nSize);

throw(CMemoryException);

void * operator new(size_t nSize, LPCSTR lpszFileName, int nLine);

throw(CMemoryException);

说明:在 Release 版中,new 操作符与 malloc 函数相似,优化分配内存;在 Debug 中,new 操作符进行内存泄露检测。

如果用户在.cpp 文件中的所有成员函数实现之前使用了下述代码行 # define new DEBUG_NEW 则使用 new 运算符的第二个版本。它将文件名和行号存放在分配的块中,以便备用。用户无需提供额外参数;C++中有特定的宏完成这些工作。即使用户在 Debug 模式中没有定义 DEBUG_NEW,程序仍然进行泄露检测,但在报告窗口中不显示源文件的行号。

注意:如果用户重载了本运算符,同时必须重载 delete 运算符。不可使用标准库函数 _new_handler。

参见:CObject::operator delete

CObject::operator delete

void operator delete(void * p);

说明:在 Release 版中,delete 操作符仅仅是释放有 new 操作符分配的内存;在 Debug 版中,delete 还进行内存泄露检测。如果你重载了操作符 new 和 delete,则新的操作符将没有诊断功能。

参见:CObject::operator new

CObject::operator =

void operator = (const CObject& src);

说明:标准 C++缺省的类赋值操作是成员到成员的复制。若使用了非重载操作符,该私有赋值运算符给出一个编译器错误信息。因此,用户要对一个用户派生类的对象赋值,则必须提供一个赋值操作符。

1.2 CRuntimeClass 类

CRuntimeClass 没有基类。

每一个由 CObject 派生的类与 CRuntimeClass 结构体相关联,这样用户在运行时就能得到该对象及其基类的有关信息。当函数参数需要类型检查或者用户要在对象类的基础上编写一些特殊要求的代码时,用户就需要在运行时刻确定对象的类。C++不能直接支持运行时刻类的信息。

该结构体有如下成员:

- LPCSTR m_lpszClassName ASCII 类名,是一个以'\0'结尾的字符串;
- int m_nObjectSize 对象大小,单位字节。如果对象含有已分配内存的指针成员,则对象大小不包括这个内存大小。
- UINT m_wSchema 预定数目(如果类不能序列化,则为 -1)。
- CObject * (PASCAL * m_pfnCreateObject)( ) 函数指针,指向创建用户类对象的缺省指针。(只有该类支持动态创建时才有效;否则为 NULL)。
- CRuntimeClass * (PASCAL * m_pfn_GetBaseClass)( ) 如果用户应用程序与 MFC 的 AFXDLL 动态链接,该指针指向返回基类的 CRuntimeClass 结构体的函数。
- CRuntimeClass * m_pBaseClass 如果用户程序与 MFC 静态链接,该成员表示指向基类的 CRuntimeClass 结构体指针。

• CObject * CreateObject( ) 从 CObject 继承的类具有在运行时刻动态创建的能力。例如,文档、视图和框架类都应当支持动态创建特性。CreateObject 成员函数可以用来实施这个功能,它在运行时刻创建类的对象。

• BOOL IsDerivedFrom(const CRuntimeClass * pBaseClass) const 如果该类是从具有参数为 CRuntimeClass 结构体的类中派生来的,则返回 TRUE。否则返回 FALSE。

注意:如果需要使用 CRuntimeClass 结构体,必须在用户类的实现时添加 IMPLEMENT _ DYNAMIC 宏、IMPLEMENT _ DYNCREATE 宏或 IMPLEMENT _ SERIAL 宏。

参 见: CObject:: GetRuntimeClass, CObject:: IsKindOf, RUNTIME _ CLASS, IMPLEMENT _ DYNAMIC, IMPLEMENT _ DYNCREATE, IMPLEMENT _ SERIAL

2 MFC 应用体系类

本章介绍应用框架体系结构方面的 MFC 类。这些类提供了大多数应用程序都需要的一些函数。用户使用时只需从这些类派生出自己的类，并增加和重载一些成员函数即可。AppWizard 能自动产生几种类型的应用程序框架。SDI(单文档界面)和 MDI(多文档界面)都充分利用了文档/视图的体系框架。其它几种类型应用，例如基于对话框的应用、基于表单的应用和 DLLs，只利用了文档/视图体系的部分特征。

文档/视图应用程序包括一组或多组文档、视图和框架窗口。文档模板对象与每一组文档/视图/框架类相关联。虽然用户在 MFC 应用程序中可以不使用文档/视图结构，但使用文档/视图结构的确有所裨益。MFC 的 OLE 容器和 OLE 服务器都基于文档/视图架构，打印和打印预览也是如此。

所有 MFC 应用程序至少包含两个对象：从 CWinApp 派生的应用对象和从 CWnd 直接或间接派生的主窗口对象。通常主窗口是从 CFrameWnd、CMDIFrameWnd 或 CDialog 派生的，它们都由 CWnd 派生的。

使用文档/视图结构的应用程序还包含其它对象，其中基本的几种对象如下：

- 应用对象，从 CWinApp 类派生；
- 一个或多个文档类对象，由 CDocument 类派生。文档类负责视图中所需数据的表示和操作，一般与一个数据文件相关联；
- 一个或多个视图对象，由 CView 类派生。每一个视图即一个窗口，它与一个文档和框架窗口相关联。视图中显示并处理文档中的数据。

文档/视图应用程序还包含框架窗口和文档模板，其中框架窗口从 CFrameWnd 派生的，文档模板从 CDocTemplate 派生的。

2.1 应用程序和线程类

2.1.1 Win 应用类

CWinApp 类成员

数据成员

m_pszAppName　应用程序名。

m_hInstance　应用程序当前句柄。

m_hPrevInstance　32 位程序中设为 NULL。

m_lpCmdLine　命令行字符串。

m_nCmdShow　设置窗口初始化时是否显示。

m_bHelpMode　设置是否使用帮助上下文模式。通常由 SHIFT + F1 进入帮助状态。

m_pActiveWnd　当 OLE 服务器被激活在相应位置时，该成员指向容器的主窗口。

m_pszExeName　程序的模块名称。

m_pszHelpFilePath　帮助文件的路径。

m_pszProfileName　文件名。

m_pszRegistryKey　存储程序设置的注册键。

构造函数

CWinApp　构造一个 CWinApp 对象。

操作函数

LoadCursor　调入光标资源。

LoadStandardCursor　从文件 WINDOWS.H 中调入 IDC_常量预定义的光标。

LoadOEMCursor　从文件 WINDOWS.H 中调入 OCR_常量预定义的 Windows OEM 光标。

LoadIcon　调入图标资源。

LoadStandardIcon　从文件 WINDOWS.H 中调入 IDI_常量预定义的 Windows 图标。

LoadOEMIcon　从文件 WINDOWS.H 中调入 OIC_常量预定义的 Windows OEM 图标。

RunAutomated　测试选项/Automation 在命令行中是否起作用。该项已不用,现用 CCommandLineInfo::m_bRunEmbedded 代替。

RunEmbedded　测试选项/Embedding 在命令行中是否起作用。该项已不用,现用 CCommandLineInfo::m_bRunEmbedded 代替。

ParseCommandLine　分析命令行中每一个参数和标志符。

ProcessShellCommand　处理命令行参数和标志符。

GetProfileInt　从程序的.INI 文件中取得一个项目的整型值。

WriteProfileInt　向程序的.INI 文件写入一个项目的整型值。

GetProfileString　从程序的.INI 文件中取得一个项目的字符串。

WriteProfileString　向程序的.INI 文件写入一个项目的字符串。

AddDocTemplate　增加一个文档模板到文档模板列表中。

GetFirstDocTemplatePosition　取得第一个文档模板的位置。

GetNextDocTemplate　取得下一个模板的位置。可以重复使用。

OpenDocumentFile　由框架调用,用来从文件中打开文档。

AddToRecentFileList　将新的文件名加到最新使用过的文件列表中。

SelectPrinter　从打印机对话框中选择打印机。

CreatePrinterDC　创建一个打印机设备上下文。

GetPrinterDeviceDefaults　取得缺省打印机设备。

重载函数

InitInstance　重载该函数可以进行 Windows 的初始化实例工作。例如可以创建窗口对象。

Run　运行缺省的消息循环。重载该函数可以定制消息虚幻循环。

OnIdle　重载该函数可以处理特定的空闲进程。

ExitInstance　重载该函数可以在程序终止时完成一些清除工作。

HideApplication　在关闭所有文档之前,隐藏应用程序。

CloseAllDocuments　关闭所有打开的文档。

PreTranslateMessage　将消息发送到 Windows 函数::TranslateMessage 和::DispatchMessage 之前进行过滤。

SaveAllModified　提示用户将保存所有修改过的文档。

DoMessageBox　执行 AfxMessageBox 函数。

ProcessMessageFilter　截获并处理发送到应用程序的消息。

ProcessWndProcException　截获并处理所有被程序消息和命令抛出的异常。

DoWaitCursor　打开或关闭等待光标的状态。

OnDDECommand　由框架调用以响应动态数据交换(DDE)执行命令。

WinHelp　调用 WinHelp 函数。

初始化函数

LoadStdProfileSettings　调入标准的.INI 文件设置,并设置支持 MRU 文件列表特性。

SetDialogBkColor　设置对话框和消息框的缺省背景颜色。

SetRegistryKey　将应用程序设置保存到注册表中,而不是.INI 文件中。

EnableShellOpen　允许用户从 Windows 文件管理器中打开数据文件。

RegisterShellFileTypes　通过 Windows 文件管理器注册所有应用程序文档。

Enable3dControls　设置控件具有三维外观。

Enable3dControlsStatic　设置控件具有三维外观。

命令处理函数

OnFileNew　实现 ID_FILE_NEW 命令。

OnFileOpen　实现 ID_FILE_OPEN 命令。

OnFilePrintSetup　实现 ID_FILE_PRINT_SETUP 命令。

OnContextHelp　处理 SHIFT + F1 按键 。

OnHelp　处理 F1 按键。

OnHelpIndex　处理 ID_HELP_INDEX 命令并提供一个缺省帮助主题。

OnHelpFinder　处理 ID_HELP_FINDER 和 ID_DEFAULT_HELP 命令。

OnHelpUsing　处理 ID_HELP_USING 命令。

构造函数

CWinApp::CWinApp

```
CWinApp( LPCTSTR lpszAppName = NULL );
```

参数:lpszAppName　应用程序名,是一个以'\0'结尾的字符串。如果程序没有提供该参数或为 NULL,那么 CWinApp 使用资源字符串 AFX_IDS_APP_TITLE 或可执行文件名。

说明:构造一个 CWinApp 对象并将 lpszAppName 存储起来作为应用程序名。每一个应用程序只能有一个 CWinApp 构造的对象。

操作函数

CWinApp::LoadCursor

```
HCURSOR LoadCursor( LPCTSTR lpszResourceName ) const;
HCURSOR LoadCursor( UINT nIDResource ) const;
```

返回值:如果调用成功,返回光标的句柄;否则为 NULL。

参数:lpszResourceName　光标资源名,是一个以'\0'结尾的字符串。可以通过 CString 定义该参数。

nIDResource　光标资源的 ID。

说明:调用 lpszResourceName 标识的光标资源,或者调用由当前可执行文件中的 nIDResource 指定的光标资源。如果光标已调用,则返回其句柄;否则将其调入内存。使用 LoadStandardCursor 和 LoadOEMCursor 成员函数访问预定义的 Windows 光标。

参见:CWinApp::LoadStandardCursor, CWinApp::LoadOEMCursor, ::LoadCursor

CWinApp::LoadStandardCursor

HCURSOR LoadStandardCursor(LPCTSTR lpszCursorName) const;

返回值：如果调用成功，返回光标的句柄；否则返回 NULL。

参数：lpszCursorName　预定义的 Windows 光标名，为 WINDOWS.H 定义的 IDC_常量。其意义说明如下：

IDC_ARROW　标准箭头光标。

IDC_IBEAM　标准文本插入光标。

IDC_WAIT　沙漏状光标，表示 Windows 正在执行任务。

IDC_CROSS　十字光标，表示选择。

IDC_UPARROW　上箭头光标。

IDC_SIZE　已不用，现使用 IDC_SIZEALL。

IDC_SIZEALL　四个箭头的光标，重新设置窗口大小。

IDC_ICON　已不用，现用 IDC_ARROW.

IDC_SIZENWSE　光标的两个箭头分别指向左上角和右下角。

IDC_SIZENESW　光标的两个箭头分别指向右上角和左下角。

IDC_SIZEWE　具有两箭头的横向光标。

IDC_SIZENS　具有两箭头的纵向光标。

说明：调用 lpszCursorName 指定的 Windows 预定义的光标资源。使用 LoadStandardCursor 或 LoadOEMCursor 成员函数访问 Widnows 预定义的光标资源。

参见：CWinApp::LoadOEMCursor, CWinApp::LoadCursor,::LoadCursor

CWinApp::LoadOEMCursor

HCURSOR LoadOEMCursor(UINT nIDCursor) const;

返回值：如果调用成功，返回光标的句柄；否则返回 NULL。

参数：nIDCursor　预定义的 Windows 光标名，为 WINDOWS.H 定义的 OCR_常量。用户必须在 #include <afxwin.h> 之前定义 #define OEMRESOURCE，以便能访问 WINDOWS.H 中定义的 OCR_常量。

说明：调用由 nIDCursor 指定的 Windows 预定义资源光标。使用成员函数 LoadOEMCursor 或 LoadStandardCursor 访问 Windows 预定义的光标。

参见：CWinApp::LoadCursor, CWinApp::LoadStandardCursor,::LoadCursor

CWinApp::LoadIcon

HICON LoadIcon(LPCTSTR lpszResourceName) const;

HICON LoadIcon(UINT nIDResource) const;

返回值：如果调用成功，返回光标的句柄。否则返回 NULL。

参数：lpszResourceName　图标资源名，是一个以'\0'结尾的字符串。可以通过 CString 定义该参数。

nIDResource　图标资源的 ID。

说明：调用 lpszResourceName 标识的图标资源，或者调用由当前可执行文件中的 nIDResource 指定的图标资源。如果图标已调用，则返回其句柄；否则将其调入内存。使用 LoadStandardIcon 和 LoadOEMIcon 成员函数访问预定义的 Windows 图标。

注意：该成员函数实际是调用 Win32 API 函数 LoadIcon，它所调用的图标大小必须符合

SM _CXICON 和 SM _CYICON 所规定的值域。

参见:CWinApp::LoadStandardIcon, CWinApp::LoadOEMIcon,::LoadIcon

CWinApp::LoadStandardIcon

HICON LoadStandardIcon(LPCTSTR lpszIconName) const;

返回值:如果调用成功,返回图标的句柄。否则返回 NULL。

参数:lpszIconName 预定义的 Windows 图标名,为 WINDOWS.H 定义的常量。其意义说明如下:

IDI _APPLICATION 缺省图标。

IDI _HAND 手状图标,用于严重警告消息。

IDI _QUESTION 问号状图标,用于提示消息。

IDI _EXCLAMATION 惊叹号状图标,用于警告消息。

IDI _ASTERISK 星号状图标,用于信息提示。

说明:调用由 lpszIconName 指定的 Windows 预定义图标。使用 LoadStandardIcon 或 LoadOEMIcon 成员函数访问预定义的 Windows 图标。

参见:CWinApp::LoadOEMIcon, CWinApp::LoadIcon,::LoadIcon

CWinApp::LoadOEMIcon

HICON LoadOEMIcon(UINT nIDIcon) const;

返回值:如果调用成功,返回图标的句柄,否则返回 NULL。

参数:nIDIcon 预定义的 Windows 图标名,为 WINDOWS.H 定义的 OIC _常量。用户必须在 # include <afxwin.h> 之前定义 # define OEMRESOURCE,以便能访问 WINDOWS.H 中定义的 OIC _常量。

说明:调用由 nIDIcon 指定的 Windows 预定义资源图标。使用成员函数 LoadOEMIcon 或 LoadStandardIcon 访问 Windows 预定义的图标。

参见:CWinApp::LoadStandardIcon, CWinApp::LoadIcon,::LoadIcon

CWinApp::RunAutomated

BOOL RunAutomated( );

返回值:如果发现选项返回非 0;否则返回 0。

说明:调用该成员函数确定"/Automation"或"-Automation"选项是否存在,以表明服务器应用程序是否被客户程序启动。如果选项存在,则从命令行中移去该选项。

参见:CWinApp::RunEmbedded

CWinApp::RunEmbedded

BOOL RunEmbedded( );

返回值:如果发现选项返回非 0;否则返回 0。

说明:调用该成员函数确定"/Embedding" 或"-Embedding"选项是否存在,以表明服务器应用程序是否被客户程序启动。如果选项存在,则从命令行中移去该选项。

参见:CWinApp::RunAutomated

CWinApp::ParseCommandLine

void ParseCommandLine(CCommandLineInfo& rCmdInfo);

参数:rCmdInfo　对 CCommandLineInfo 对象的引用。

说明:调用该成员函数分析命令行并且向 CCommandLineInfo::ParseParam 发送参数。当用户通过 AppWizard 新建一个新的 MFC 项目时,AppWizard 将创建一个本地的 CCommandLineInfo 实例,然后调用 InitInstance 成员函数的 ProcessShellCommand 和 ParseCommandLine。

命令行遵循以下过程。

(1)在 InitInstance 中创建了 CCommandLineInfo 对象后,将该对象发送到 ParseCommandLine。

(2)ParseCommandLine 反复调用 CCommandLineInfo::ParseParam,每个参数调用一次。

(3)ParseParam 填写 CCommandLineInfo 对象,然后发送到 ProcessShellCommand。

(4)ProcessShellCommand 处理命令行参数和标志符。

注意:如果需要的话,可以直接调用 ParseCommandLine。

参见: CCommandLineInfo, CWinApp::InitInstance, CCommandLineInfo::ParseParam, CWinApp::ProcessShellCommand, CCommandLineInfo::m _nShellCommand

CWinApp::ProcessShellCommand

```
BOOL ProcessShellCommand( CCommandLineInfo& rCmdInfo );
```

返回值:如果成功处理了 Shell 命令,返回非 0;如果为 0,则从 InitInstance 中返回 FALSE。

参数:rCmdInfo　对 CCommandLineInfo 的引用。

说明:InitInstance 调用该成员函数从 rCmdInfo 标识的 CCommandLineInfo 对象接受参数,并处理有关操作。当用户通过 AppWizard 新建一个新的 MFC 项目时,AppWizard 将创建一个本地的 CCommandLineInfo 实例,然后调用 InitInstance 成员函数的 ProcessShellCommand 和 ParseCommandLine。

命令行遵循以下过程。

(1)在 InitInstance 中创建了 CCommandLineInfo 对象后,将该对象发送到 ParseCommandLine。

(2)ParseCommandLine 反复调用 CCommandLineInfo::ParseParam,每个参数调用一次。

(3)ParseParam 填写 CCommandLineInfo 对象,然后发送到 ProcessShellCommand。

(4)ProcessShellCommand 处理命令行参数和标志符。

CCommandLineInfo 对象的数据成员 CCommandLineInfo::m _nShellCommand 有如下枚举类型。这些类型在 CCommandLineInfo 中得到定义。

```
enum{
  FileNew,
  FileOpen,
  FilePrint,
  FilePrintTo,
  FileDDE,
};
```

参见:CWinApp::ParseCommandLine, CCommandLineInfo,
CCommandLineInfo::ParseParam, CCommandLineInfo::m _nShellCommand

CWinApp::GetProfileInt

```
UINT GetProfileInt( LPCTSTR lpszSection, LPCTSTR lpszEntry, int nDefault );
```

返回值:返回指定条目字符串的整形值。如果没有发现该条目,返回 nDefault 参数的值。如果与指定条目相对应的值不是个整型值,则返回 0。该函数支持.INI 文件中有 16 进制的值。如果要检索一个有符号的整型,那么要将其转换为 int 型。

参数:lpszSection　指定包含条目的段,为一个字符串。

lpszEntry　要检索的条目。

nDefault　指定没有发现指定的条目时返回的缺省值。该参数可以是个无符号整型值(0 到 65,535),也可以是个符号整型值(从 -32,768 到 32,767)。

说明:调用该函数从程序注册表或.INI 文件中检索指定段中的条目值。条目存储情况如下:

在 Windows NT 中,该值存为一个注册键。

在 Windows 3.x 中,该值存在 WIN.INI 文件中。

在 Windows 95 中,该值存在 WIN.INI 文件的一个高速缓存副本中。

该成员函数大小写不敏感。

参见:CWinApp::GetProfileString, CWinApp::WriteProfileInt,::GetPrivateProfileInt

CWinApp::WriteProfileInt

BOOL WriteProfileInt(LPCTSTR lpszSection, LPCTSTR lpszEntry, int nValue);

返回值:如果成功返回非 0;否则返回 0。

参数:lpszSection　指定包含条目的段。如果该段不存在,则创建一个。段名大小写不敏感。

lpszEntry　指定需写入值的条目。如果指定段中的条目不存在,则创建一个。

nValue　写入条目的值。

说明:调用该函数将指定的值写入程序注册表或.INI 文件中的指定段中。条目存储情况如下:

在 Windows NT 中,该值存为一个注册键。

在 Windows 3.x 中,该值存在 WIN.INI 文件中。

在 Windows 95 中,该值存在 WIN.INI 文件的一个高速缓存副本中。

参见:CWinApp::GetProfileInt, CWinApp::WriteProfileString, CWinApp::SetRegistryKey

CWinApp::GetProfileString

CString GetProfileString(LPCTSTR lpszSection, LPCTSTR lpszEntry, LPCTSTR lpszDefault = NULL);

返回值:返回应用程序的.INI 文件中的字符串。如果没有发现字符串,返回 lpszDefault。字符串的最大长度为_MAX_PATH。如果 lpszDefault 为 NULL,返回空的字符串。

参数:lpszSection　指定包含条目的段。

lpszEntry　需要检索其中字符串的条目,不能为 NULL。

lpszDefault　如果指定条目没有发现时返回的缺省字符串。

说明:检索与指定应用程序的注册表或.INI 文件的段相关的字符串。条目存储情况如下:

在 Windows NT 中,该值存为一个注册键。

在 Windows 3.x 中,该值存在 WIN.INI 文件中。

在 Windows 95 中,该值存在 WIN.INI 文件的一个高速缓存副本中。

参见:CWinApp::GetProfileInt, CWinApp::WriteProfileString,::GetPrivateProfileString

CWinApp::WriteProfileString

BOOL WriteProfileString(LPCTSTR lpszSection, LPCTSTR lpszEntry, LPCTSTR lpszValue);

返回值:成功返回非 0;否则返回 0。

参数:lpszSection　指向包含条目的段。如果这个段不存在,则创建一个段。段名大小写不敏感。

lpszEntry　指定需写入值的条目。如果指定段中的条目不存在,则创建一个。

lpszValue　字符串指针。如果该参数为 NULL,则删除由 lpszEntry 参数指定的条目。

说明:调用此函数向应用程序注册表或.INI 文件中指定的段写入字符串。条目存储情况如下:

在 Windows NT 中,该值存为一个注册键。

在 Windows 3.x 中,该值存在 WIN.INI 文件中。

在 Windows 95 中,该值存在 WIN.INI 文件的一个高速缓存副本中。

参见:CWinApp::GetProfileString, CWinApp::WriteProfileInt,::WritePrivateProfileString, CWinApp::SetRegistryKey

CWinApp::AddDocTemplate

```
void AddDocTemplate( CDocTemplate* pTemplate );
```

参数:pTemplate 指向需要添加的 CDocTemplate 对象指针。

说明:调用该成员函数向文档模板列表中增加新的文档模板。用户在调用 RegisterShellFileTypes 函数之前应向应用程序添加所有的文档模板。

参见:CWinApp::RegisterShellFileTypes, CMultiDocTemplate, CSingleDocTemplate

CWinApp::GetFirstDocTemplatePosition

```
POSITION GetFirstDocTemplatePosition( ) const;
```

返回值:返回一个 POSITION 值;如果列表为空,则返回 NULL。

说明:取得第一个文档模板的位置。通过 GetNextDocTemplate 返回的 POSITION 值可以得到第一个 CDocTemplate 对象。

参见:CWinApp::AddDocTemplate, CWinApp::GetNextDocTemplate

CWinApp::GetNextDocTemplate

```
CDocTemplate* GetNextDocTemplate( POSITION& pos ) const;
```

返回值:返回 CDocTemplate 对象的指针。

参数:pos 通过调用 GetNextDocTemplate 或 GetFirstDocTemplatePosition 取得的 POSITION 值。调用该函数后,该值更新为下一个位置。

说明:取得由 pos 指定的模板,调用后将 pos 设置为 POSITION 值。如果通过 GetFirstDocTemplatePosition 创建了初始化位置,那么就能使用 GetNextDocTemplate 进行迭代循环。POSITION 必须是有效的。如果取得的文档模板是最后一个,则新的 pos 值将设为 NULL。

参见:CWinApp::AddDocTemplate, CWinApp::GetFirstDocTemplatePosition

CWinApp::OpenDocumentFile

```
virtual CDocument* OpenDocumentFile( LPCTSTR lpszFileName );
```

返回值:如果调用成功,返回 CDocument 指针;否则返回 NULL。

参数:lpszFileName 将被打开的文件名。

说明:程序框架调用该函数打开指定的 CDocument 文档。如果该文档已打开,那么包含该文档的第一个框架窗口将被激活。如果程序支持多文档模板,程序框架则通过文件扩展名找到合适的文档模板以调入文档。如果调入成功,则文档模板将创建一个框架窗口和一个视图以浏览编辑该文档。

CWinApp::AddToRecentFileList

```
virtual void AddToRecentFileList( LPCTSTR lpszPathName );
```

参数:lpszPathName 文件路径。