MFC 查询手册

龙马工作室搜集整理制作

索引

CArchive

CButton

CClientDC

CCmdTarget

CDialog

CDocument

CEdit

CFile

CFrameWnd

CListView

CObject

CSocket

CSocketFile

CString

CStringArray

CStringList

CTime

CView

CWinApp

CWindowDC

CWinThread

CWnd

CArchive

CArchive 没有基类。

CArchive 允许以一个永久二进制（通常为磁盘存储）的形式保存一个对象的复杂网络，它可以在对象被删除时，还能永久保存。可以从永久存储中装载对象，在内存中重新构造它们。使得数据永久保留的过程就叫作“串行化”。

可以把一个归档对象看作一种二进制流。象输入/输出流一样，归档与文件有关并允许写缓冲区以及从硬盘读出或读入数据。输入/输出流处理一系列 ASCII 字符，但是归档文件以一种有效率、精练的格式处理二进制对象。

必须在创建一个 CArchive 对象之前，创建一个 CFile 对象。另外，必须确信归档文件的装入/存储与文件的打开模式是兼容的。每一个文件只限于一个活动归档文件。

当构造一个 CArchive 对象时，要把它附加给表示一个打开文件的类 CFile（或派生类）的对象上。还要指定归档文件将用于装载还是存储。

CArchive 对象不仅可以处理首要类型，而且还能处理为串行化而设计的 CObject 派生类的对象。一个串行化类通常有一个 Serialize 成员函数并且使用 DECLARE_SERIAL 和 IMPLEMENT_SERIAL 宏。这些在 CObject 类中有所描述。

重载提取（>>）和插入（<<）是方便的归档编程接口。它支持主要类型和 CObject 派生类。

CArchive

参数：

pFile　　CFile 对象的指针。CFile 对象是永久数据的最终的源或目标。

nMode　标识。它指定了对象是否从归档文件中装载或存储到文件中去。nMode 参数必须有下列值之一：

· CArchive::load　　从归档文件装载数据。CFile 只读。

· CArchive::store 把数据保存到归档文件中。允许 CFile 写操作。

· CArchive::bNoFlushOnDelete 当归档文件析构程序被调用时，防止归档文件自动调用 Flush。如果设定了此标识，则在析构程序被调用之前必须负责调用 Close。如果不这样做，数据就会崩溃。

nBufSize 指定内部文件缓冲区大小的整数，以字节计算。注意缺省的缓冲区大小为 4096 字节。如果例程归档大的对象，使用大一些的缓冲区，即多个文件缓冲区，那么将会提高例程的执行效率。

lpBuf 指向 nBufSize 大小的提供缓冲区的指针。如果不指定这个参数，归档文件从本地堆为归档文件分配一个缓冲区并且当对象被毁弃时，释放缓冲区。归档文件不能释放一个提供的缓冲区。

说明：

构造 CArchive 对象并且指定它将用于装载或存储对象。在创建归档文件之后，不能改变这个指定内容。

不能使用 CFile 操作来改变文件的状态直到已经关闭归档文件时。任何这样的操作都将会毁弃归档文件的完整性。通过由 GetFile 成员函数获得归档文件的文件对象使得可在串行化过程中的任何时候访问文件指针的位置。然后使用 CFile::GetPosition 函数。应该在获得文件指针位置之前，调用 CArchive::Flush。

示例：

```
extern char* pFileName;
CFile f;
char buf[512];
if(!f.Open( pFileName,CFile::modeCreate| CFile::modeWrite))
{
  #ifdef_DEBUG
     afxDump<< "Unable to open file"<< "\n"
     exit(1);
  #endif
}
```

Close

说明：

冲掉保存在缓冲区中的任何数据，关闭归档文件并且释放归档文件与文件的链接。对于归档文件没有允许的其它操作。在关闭一个归档文件之后，可以为一个同样文件创建另一个归档文件或者关闭文件。

成员函数 Close 保证所有数据从归档文件传输到文件并且使归档文件无效。为了完成从文件到存储介质的传输，必须首先使用 CFile::Close 并且再毁弃 CFile 对象。

Flush( )

说明：

迫使保留在归档文件中的数据写入文件。

成员函数 Flush 保证所有的数据从归档文件传输到文件。必须调用 CFile::Close 来完成从文件到存储介质的传输。

CFile* GetFile( ) const

返回值：指向正在使用的 CFile 对象的指针。

说明：

为此归档文件获得 CFile 对象指针。必须在使用 GetFile 之前冲掉归档文件。

示例：

```
extern CArchive ar;
const CFile* fp = ar.GetFile( );
```

MapObject

参数：

pOb　　指向正在存储的对象的指针。

说明：

调用此成员函数在映射中放置对象。此映射没有真的对文件串行化，但是对参考的子对象有效。举例来说，可能不想串行化一个文档，但是想对作为文档一部分的项串行化。通过调用 MapObject，可允许那些项或子对象参考文档。而且，串行化子项还可以串行化它们的 m_pDocument 向后的指针。

当要把数据存储到 CArchive 对象和从它装载数据时，可以调用 MapObject。在串行化和非串行化的过程中，MapObject 向由 CArchive 保持的内部数据结构添加特定的对象，但是它不象 ReadObject 和 WriteObject 那样，它不会对对象串行化。

示例：

```
//MyDoc.h
//Document should have DECLEAR_SERIAL and IMPLEMENT_SERIAL
class CMyDocument: public CDocument
{
  CObList m_listOfSubItems;
  ...
  DECLARE_SERIAL(CMyDocument)
};
//MyDoc.cpp
...
IMPLEMENT_SERIAL(CMyDocument.CObject,1)
...
void CMyDocument::Serialize(CArchive& ar)
{
  if (ar.IsStoring( ))
  {
    //TODO:add storing code here
  }
  else
  {
    //TODO:add loading code here
  }
  ar.MapObject(this);
  //serialize the subitems in the documents;
  //they will be able to serialize their m_pDoc
  //back pointer
  m_listOfSubItems.Serialize(ar);
}
//SubItem.h
class CSubItem:public CObject
{
  public:
   CSubItem(CMYDocument * pdoc) { m_pDoc = pDoc }
```

```
    //back pointer to owning document
    CMYDocument* m_pDoc
    WORD m_I; //other item data
    virtual void Serialize(CArchive& ar);
};
//SubItem.cpp
void CSubItem:Serialize(CArchive& ar)
{
  if (ar.IsStoring( ))
  {
    //will serializing a reference
    //to the “mapped”document pointer
    ar <<M_PDOC;
    ar<<M_I;
  }
  else
  {
    //will load a reference to
    //the “mapped”document pointer
    ar >>m_pDoc;
    ar >>m_i;
  }
}
```

Read

返回值：

包含实际读入字节数的无符号整数。如果返回值小于所需要的数值，则说明已达到了文件的末尾。达到文件末尾时，没有异常。

参数：

lpBuf　指向提供的缓冲区的指针。这个提供的缓冲区接收从归档文件读取的数据。

nMax　标明从归档文件读取的字节数的无符号指针。

说明：

从归档文件读取指定的字节数。归档文件不解释字节数。

可以在 Serialize 函数里使用 Read 成员函数，读取包含在对象中的普通结构。

示例：

```
extern CArchive ar;
char pb[100];
UINT nr=ar.Read(pb.100);
```

ReadClass

返回值：指向 CRuntimeClass 结构的指针。

参数：

pClassRefRequested　指向 CRuntimeClass 结构的指针，此结构对应于所需要的类的参考。可以为 NULL。

pSchema　指向原先存储的运行时类的大纲的指针。

obTag　代表对象的唯一标识的数值。通过 ReadObject 的实现，在内部使用。只用于高级编程。obTag 通常应为 NULL。

说明：

调用此成员函数来读取一个原先存储在 WriteClass 中的类的参考。

如果 pClassRefReguested 不为 NULL，ReadClass 证明了归档文件类信息与例程类是兼容的。

如果不兼容，ReadClass 将产生一个 CArchiveException。

例程必须使用 DECLARE_SERIAL 和 IMPLEMENT_SERIAL，否则，ReadClass 将产生一个 CNotSupportedException。

如果 pSchema 为 NULL，则存储类的大纲可通过调用 CArchive::GetObjectSchema 来恢复，否则，*pSchema 将会包含原先存储的运行时类的大纲。

可以使用 SerializeClass 来代替 ReadClass，它可以处理类参考的读和写。

ReadObject

返回值：

CObject 指针。它必须通过使用 CObject::IsKindOf 安全地指向当前的派生类。

参数：

pClass　指向 CRumtimeClass 结构的常量指针。此结构对应于期待读入的对象。

说明：

从归档文件中读入对象并构造一个合适类型的对象。

此函数通常由 CArchive 提取（>>）运算符重载一个 CObject 指针调用它。ReadObject，反过来，调用归档文件类的 Serlialize 函数。

如果提供一个非零的 pClass **参数**，它可以从 RUNTIME_CLASS 获得，则函数确认归档文件的运行类。假设在类的实现中已经使用 IMPLEMENT_SERIAL 宏。

ReadString

返回值：

在返回逻辑值的版本中，True 代表成功；False 相反。

在返回 LPTSTR 的版本中，LPTSTR 是一个指向包含文字数据的缓冲区的指针，NULL 代表到达文件尾。

参数：

rString　CString 的一个参考。CString 将包含从与 CArchive 对象有关的文件中读出的结果字符串。

lpsz　指定一个指向提供的缓冲区的指针。此缓冲区将接收一个以空终止符结尾的文字串。

nMax　指定读入字符的最大数。它不允许小于 lpsz 缓冲区的大小。

说明：

调用此函数从与 CArchive 对象有关的文件中读取数据，放入缓冲区。在带有 nMax 参数的成员函数的版本中，缓冲区最多有 nMax-1 字符。读入的操作由一个回车换行符终止。在新行符号后面的字符都被丢弃了。在每种情况下，都有一个空字符（“/0”）。

CArchive::Read 还对文本-模式输入有效，但是它不以回车换行符终止。

SerializeClass

参数：

pRuntimeClass　　指向基类的运行类对象的指针。

说明：

当想存储和装载一个基类的版本信息时，调用此成员函数。SerializeClass 读或对 CArchive 对象写一个类的参考，取决于 CArchive 的方向。使用 SerializeClass 来代替 ReadClass 和 WriteClass，可以使得更方便地串行化基类对象。因为 SerializeClass 需要较少的代码和参数。

像 ReadClass 一样，SerializeClass 证明了归档文件信息与的例程类兼容。如果不兼容，则 SerializeClass 将一个 CArchiveException。

运行类必须使用 DECLARE_SERIAL 和 IMPLEMENT_SERIAL,否则，SerializeClass 将会一个 CNotSupportException。

使用 RUMTIME_CLASS 宏，恢复 pRuntimeClass 参数的值。基类必须已经使用 IMPLEMENT_SERIAL 宏。

示例：

```
class CBaseClass:public CObject{...};
class CDerivedClass:public CBaseClass{...};
void CDerivedClass::Serialize(CArchive& ar)
{
  if(ar.IsStoring( ))
  {
    //normal code for storing contents
    //of this object
  }
  else
  {
    //normal code for reading contents
    //of this object
  }
  //allow the base class to serialize along
  // with its version information
  ar.SerializeClass(RUNTIME_CLASS(CBaseClass));
  CBaseClass::Serialize(ar);
```

```
}
```

Write

参数：

lpBuf　指向提供的缓冲区的指针。缓冲区中包含了将被写入归档文件的数据。

nMax　指定要被写入归档文件的字节数的整数。

说明：

向归档文件写入特定的字节数。归档文件不会格式化字节。

可以在 Serialize 函数中使用 Write 成员函数，写入一个包含在对象中的普通结构。

示例：

```
extern CArchive ar;
char pb[100];
ar.Write(pb.100);
```

WriteClass

参数：

pClassRef　指向 CRuntimeClass 结构的指针。此结构对应于所需要的类的参考。

说明：

在一个派生类的串行化过程中，使用 WriteClass 存储基类的版本和类信息。WriteClass 将基类的 CRuntimeClass 参考写入 CArchive。使用 CArchive::ReadClass 恢复参考。

WriteClass 证明了归档文件信息与例程类兼容。如果不兼容，WriteClass 将 CArchive::Exception。

例程必须使用 DECLARE_SERIAL 和 IMPLEMENT_SERIAL。否则，WriteClass 将一个 CNotSuppotedException。

可以使用 SerializeClass 代替 WriteClass，它将处理类参考的读与写。

WriteObject

参数：

pOb　　被存储对象的常量指针。

说明：

将特定的 CObject 存储到归档文件中。

此函数通常由重载 CObject 的 CArchive 插入（<<）运算符调用。WriteClass 反过来，调用归档类的 Serialize 函数。

必须使用 IMPLEMENT_SERIAL 宏归档。WriteObject 将 ASCII 类名写入归档文件。在装载过程之后，此类名会被验证。一个特定的编码设计会防止对于多个对象类名的不必要的重复使用。还能防止是多个指针目标的对象的多余存储。

正确的对象编码方法（包括在现在的 ASCII 类名中）是一个实现细节并且在库的以后版本中可以改变。

注意：

在归档之前，结束创建、删除和修改所有对象。如果把归档和对象修改混合，则归档文件将会被废弃。

WriteString

参数：

lpsz　　使用此成员函数将数据从一个缓冲区写入与 CArchive 对象有关的文件中。结束的空字符（‘\0’）不被写入文件，新行也不自动写入。

说明：

WriteString 对于几种情况都可以异常，包括磁盘满时。Write 还是有效的，但不是以空终止符结尾，而是它向文件写入需要的字节数。

CButton

类 CButton 提供了对 Windows 按钮控件的操作。按钮控件是一个小的矩形子窗口，可以通过单击选中（按下）或不选中。按钮可以单独使用，也可以成组使用，它还可以具有文本标题。在用户单击它的时候，按钮通常要改变显示外观。

典型的按钮控件有：复选框、单选钮和下压式按钮（push button）。一个 CButton 对象可以是它们中的一种，这由它的按钮风格和成员函数 Create 的初始化决定。

此外，类 CButtonBitmap 是从类 CButton 继承而来的，不过它支持按钮的图像标签。一个 CButtonBitmap 对象可以分别为它的四种状态（未按下、按下、获得焦点和禁止存取）设置不同的位图。

既可以从对话框模板中创建一个按钮控件，也可以直接在代码中创建。

无论哪种情况，都要先调用构造函数 CButton 构造一个 CButton 对象，然后调用成员函数 Create 创建 Windows 按钮控件并应用到 CButton 对象上。

在一个从类 CButton 派生出来的类中，构造可以一步完成。程序员可以为这个派生类编写一个构造函数，并在其中调用 Create 函数。

如果想处理 Windows 的通知消息，如位图按钮控件发给它的父对象（通常是从 CDialog 继承来的）的消息，就要在父对象中加入消息映射入口以及处理每个消息的成员函数。

每个消息映射入口的格式如下：

ON_Notification(id, memberFxn)

其中 id 指定了发送通知的控件的子窗口的 ID，而 memberFxn 指定了处理该通知的父对象中的成员函数名。

父对象的函数原型格式如下：

afx_msg void memberFxn( );

可能的消息映射入口如下：

映射入口　　何时向父对象发送消息

ON_BN_CLICKED　　用户单击按钮时

ON_BN_DOUBLECLICKED　　用户双击按钮时

如果在对话框资源中创建了 CButton 对象，则在用户关闭该对话框时会自动撤消这个 CButton 对象。如果在窗口中创建了 CButton 对象，就可能需要自己撤消它。如果是用 new 函数在内存的堆中创建该对象的，则在用户关闭该窗口按钮控件时，必须用 delete 函数撤消它。如果在堆栈中创建了该对象，或者它嵌入在父对话框对象中，系统会自动撤消它。

CButton

说明：

本函数用于构造一个 CButton 对象。

Create

返回值：调用成功时返回非零值，否则为 0。

参数：

lpszCaption　指定按钮控件上的文本。

dwStyle　指定按钮控件的风格。可以采用控件风格的各种组合。

rect　指定按钮控件的大小和位置。既可以是一个 CRect 对象，也可以是一个 RECT 结构。

pParentWnd　指定按钮控件的父窗口，通常是一个 CDialog 对象。注意不能为 NULL。

nID　　指定按钮控件的 ID 号。

说明：

构造一个 CButton 对象需要两步：首先调用构造函数，然后调用 Create 函数创建 Windows 按钮控件并在 CButton 对象上应用它。

如果设置了 WS_VISIBLE 风格，Windows 将向该按钮控件发送所有用来激活和显示该按钮的消息。

按钮控件上可用的窗口风格如下：

- WS_CHILD　总是设置
- WS_VISIBLE　　通常要设置

- WS_DISABLED　很少使用
- WS_GROUP　成组按钮
- WS_TABSTOP　　按钮按制表键次序排列

GetCursor

返回值：

调用成功时返回一个指向光标图像的句柄，如果此前没有设置相应的光标图像，则返回NULL。

说明：

本函数用于取得此前用 SetCursor 在按钮上设置的光标的句柄。

GetIcon( )

返回值：

返回图标的句柄。如果此前没有设置图标，则返回 NULL。

说明：

本函数用于取得此前调用 SetIcon 在按钮上设置的图标的句柄。

SetButtonStyle

参数：

nStyle　　指定按钮的风格。

bRedraw　　指明按钮是否需要重绘。非零值表示需要重绘，0 表示不需要重绘。缺省时需要重绘。

说明：

本函数用于改变按钮的风格。

函数 GetButtonStyle 用于取得按钮的风格。按钮风格的完整名称中的小写单词指明了与具体按钮相关的风格。

SetCursor;

返回值：返回此前在按钮上设置的光标的句柄。

参数：

hCursor　光标的句柄。

说明：

本函数用于设置按钮的光标。

光标将会被自动地放到按钮上，缺省时居中放置。如果光标太大，则会自动剪裁。可以选择的对齐方式有：

- BS_TOP
- BS_LEFT
- BS_RIGHT
- BS_CENTER
- BS_BOTTOM
- BS_VCENTER

CBitmapButton 对象可以用四个位图，而 SetCursor 只为每个按钮设置一个光标。在按钮被按下时，光标看起来也向右下角倾斜。

SetIcon

返回值：返回此前在按钮上设置的图标的句柄。

参数：

hIcon　　图标的句柄。

说明：

本成员函数用于设置按钮的图标。

图标将会被自动地放到按钮的上面，缺省时居中放置。如果图标太大，则会自动剪裁。可以选择的对齐方式有：

- BS_TOP
- BS_LEFT
- BS_RIGHT
- BS_CENTER
- BS_BOTTOM
- BS_VCENTER

CbitmapButton

SetState

参数：

bHighlight，指定按钮是否被加亮显示。非零值将加亮显示按钮，0 将不加亮显示。

说明：

本函数用于设置是否加亮显示按钮。

加亮显示影响控件的外观，但对单选钮和复选框控件的选中状态没有影响。

当用户单击并且保持鼠标左键为按下状态时，按钮控件自动地加亮显示。当用户放开鼠标按钮时，按钮控件将不再加亮显示。

CClientDC

类 CClientDC 派生于 CDC，在构造时调用了 Windows 函数 GetDC，在析构时调用了 ReleaseDC。这意味着和 CClientDC 对象相关的设备上下文是窗口的客户区。

CClientDC

参数：

pWnd　设备上下文将要存取的客户区所在的窗口。

说明：

本函数构造一个 CClientDC 对象,它将存取 pWnd 指向的 CWnd 的客户区。此构造函数调用了 Windows 函数 GetDC。

如果 Windows 函数 GetDC 调用失败，则产生一个 CResourceException 类型的异常。如果 Windows 已经分配出了所有可用的设备上下文，则没有新的设备上下文可用。无论何时，应用总在竞争使用 Windows 提供的五个公共显示上下文。

CCmdTarget

类 CCmdTarget 是 MFC 类库中消息映射体系的一个基类。消息映射把命令或消息引导给用户为之编写的响应函数（命令是由菜单项、命令按钮或者加速键产生的消息）。

从CCmdTarget继承来的按键框架类包括：CView、CWinApp、CDocument、CWnd和CFrameWnd。如果想生成一个处理按键消息的类，可以选择其中的一个派生一个子类。很少需要直接从CCmdTarget派生类。

相关的命令目标和 OnCmdMsg 例程的其它信息，请参阅联机文档“VisualC++程序员指南”中的“命令目标”、“命令路由”和“映射消息”部分。

类 CmdTarget 包括了处理沙漏形光标显示的成员函数。当某个命令的执行时间比较长时，可以显示沙漏标提示用户命令正在执行。

和消息映射类似，分派映射用于列出 OLE 自动的 IDispatch 功能。列出这个接口后，其它的应用（如 VB）就能调用这个应用了。有关 IDispatch 接口的更详细的信息，请参阅“Win32 SDK OLE 程序员参考”中的“创建 IDPatch 接口”和“分派接口与 API 函数”。

BeginWaitCursor

说明：

本函数用于显示沙漏标（通常在命令执行时间较长时采用）。框架调用本函数显示沙漏标，告诉用户系统忙，例如在加载一个 CDocument 对象或把它保存到文件时。

在不是处理单个消息时，BeginWaitCursor 可能不象其它函数那样有效，例如 OnSetCursor 的处理也能改变光标形状。调用函数 EndWaitCursor 可以恢复此前的光标。

示例：

```
//The following example illustrates the most common case
//of displaying the hourglass cursor during some lengthy
//processing of a command handler implemented in some
//CCmdTaget-derived class,such as a document or view.
```

```
void CMyView::OnSomeCommand( )
{
  BeginWaitCursor( ); //display the hourglass cursor
  //do some lengthy processing
  EndWaitCursor( ); //remove the hourglass cursor
}
//The next example illustrates RestoreWaitCursor
void CMyView::OnSomeCommand( )
{
  BeginWaitCursor( ); //display the hourglass cursor
  //do some lengthy processing
  // The dialog box will normally change the cursor to
  // the standard arrow cursor,and leave the cursor in
  // as the standard arrow cursor when the dialog box is
  //closed.
  CMyDialog dlg;
  dlg.DoModal( );
  // It is necessary to call RestoreWaitCursor here in order
  // to change the cursor back to the hourglass cursor.
  RestoreWaitCursor( );
  // do some more lengthy processing
  EndWaitCursor( ); //remove the hourglass cursor
}
// In the above example,the dialog was clearly invoked between
// the pair of calls to BeginWaitCursor and EndWaitCursor.
// Sometimes it may not be clear whether the dialog is invoked
// in between a pair of calls to BeginWaitCursor and EndWaitCursor.
// It is permissable to call RestoreWaitCursor,even if
// BeginWaitCursor was not previously called.This case is
// illustrated below,where CMyView::AnotherFunction does not
// need to know whether it was called in the context of an
// hourglass cursor.
void CMyView::AnotherFunction( )
{
  // some processing...
  MyDialog dlg;
  dlg.DoModal( );
  RestoreWaitCursor( );
```

```
  //some more processing
}
// If the dialog is invoked from a member function of
// some non-CCmdTarget,the you call CWinApp::DoWaitCursor
// with a 0 parameter value to restore the hourglass cursor.
void CMyObject::AnotherFunction( )
{
  CMyDialog dlg;
  dlg.DoModal( );
  AfxGetApp( )->DoWaitCursor(0); //same as CCmdTarget::RestoreWaitCursor
}
```

EnableAutomation

说明：

本函数设置对象的 OLE 自动功能。一般在对象的构造函数里调用。调用时要保证已经为类声明了分派映射。有关自动功能的更详细信息，请参阅联机文档“Visual C++程序员指南”中的“自动客户”和“自动服务器”。

EndWaitCursor

说明：

本函数在 BeginWaitCursor 之后调用。它用于撤消沙漏标，并恢复以前的光标。框架在调用沙漏标之后也调用本函数。

示例：

```
// The following example illustrates the most common case
// of displaying the hourglass cursor during some lengthy
// processing of a command handler implemented in some
// CCmdTarget-derived class,such as a document or view.
void CMyView::OnSomeCommand( )
{
  BeginWaitCursor( ); // display the hourglass cursor
  // do some lengthy processing
  EndWaitCursor( ); // remove the hourglass cursor
```

```
}
// The next example illustrates RestoreWaitCursor
void CMyView::OnSomeCommand( )
{
  BeginWaitCursor( ); // display the hourglass cursor
  // do some lengthy processing
  // The dialog box will normally change the cursor to
  // the standard arrow cursor,and leave the cursor in
  // as the standard arrow cursor when the dialog box is
  // closed.
  CMyDialog dlg;
  dlg.DoModal( );
  // It is necessary to call RestoreWaitCursor here in order
  // to change the cursor back to the hourglass cursor.
  RestoreWaitCursor( );
  // do some more lengthy processing
  EndWaitCursor( ); // remove the hourglass cursor
}
// In the above example,the dialog was clearly invoked between
// the pair of calls to BeginWaitCursor and EndWaitCursor.
// Sometimes it may not be clear whether the dialog is invoked
// in between a pair of calls to BeginWaitCursor and EndWaitCursor.
// It is permissable to call RestoreWaitCursor,even if
// BeginWaitCursor was not previously called.This case is
// illustrated below,where CMyView::AnotherFunction does not
// need to know whether it was called in the context of an
// hourglass cursor.
void CMyView::AnotherFunction( )
{
  // some processing...
  MyDialog dlg;
  dlg.DoModal( );
  RestoreWaitCursor( );
  // some more processing ...
}
// If the dialog is invoked from a member function of
// some non-CCmdTarget,the you call CWinApp::DoWaitCursor
// with a 0 parameter value to restore the hourglass cursor.
```

```
void CMyObject::AnotherFunction( )
{
  CMyDialog dlg;
  dlg.DoModal( );
  AfxGetApp( )->DoWaitCursor(0); // same as CCmdTarget::RestorWaitCursor
}
```

FromIDispatch

返回值：

返回一个和 lpDistpatch 相关的 CCmdTarget 对象的指针。如果该 IDistpatch 对象不是一个 MFC IDistpatch 对象，则返回 NULL。

参数：

lpDispatch　　指向 IDistpatch 对象的指针。

说明：

本函数把从类的自动成员函数中得到的一个 IDistpatch 指针映射到一个实现了 IDistpatch 对象接口的 CCmdTarget 对象。

函数的结果与调用函数 GetIDispatch 的结果相反。

GetIDispatch

返回值：返回一个和该对象相关的 IDistpatch 指针。

参数：

bAddRef　　指明是否增加对该对象的参考记数。

说明：

本成员函数用于检索一个自动方法的 IDistpatch 指针，该自动方法返回一个 IDistpatch 指针或参考一个 IDistpatch 指针。

对于那些在构造函数中调用了 EnableAutomation 的对象，本函数使它们自动激活，并返回一个指向 IDistpatch 在 MFC 中的实现的指针。该 IDistpatch 被那些通过 IDistpatch 接口通

信的客户所使用。本函数的调用自动增加一个对该指针的参考，因而不需要调用IUnknow::AddRef。

IsResultExpected

返回值：如果自动函数要返回一个值，则返回非零值。否则为 0。

说明：

本函数确定客户是否期待从它对自动函数的调用中返回一个值。OLE 接口告诉 MFC 客户是使用了还是忽略了调用返回值。MFC 则使用这些信息决定 IsResultExpected 的返回值。如果返回值的计算比较耗费时间或其它资源，可以在计算返回值之前调用本函数以提高效率。

本函数只返回一次 0，这样，如果在一个已被客户端调用的自动函数中调用了其它的自动函数，就可以获得有效的返回值。

如果在没有进行自动函数调用时调用本函数，将返回非零值。

OnCmdMsg

返回值：如果消息被处理了，则返回非零值；否则为 0。

参数：

nID　命令的 ID。

nCode　命令的通知代码。

pExtra　根据 nCode 的值使用。

pHandlerInfo　如果非空，OnCmdMsg 将填充 pHandlerInfo 结构的 pTarget 和 pmf 成员，而不是分派该命令。此参数通常为 NULL。

说明：

本函数由框架来调用，它分派命令并处理那些提供了命令用户接口的对象的更新。这是框架的命令体系中实现的一个主要例程。

在运行时，OnCmdMsg 把命令分派到其它对象上或者调用 CCmdTarget::OnCmdMsg（此函数进行真正的消息映射查找）自己处理命令。有关这个缺省命令例程的完整描述，请参阅联机文档“Visual C++程序员指南”中的“消息处理”和“映射主题”部分。

偶尔需要覆盖本函数以扩展 MFC 框架的标准命令例程。有关命令例程体系中的更多细节，请参阅联机文档中的“技术指南 21”。

示例：

```
//This example illustrates extending the framework's standard command
//route from the view to objects managed by the view. This example
//is from an object-oriented drawing application, similar to the
//DRAWCLI sample application, which draws and edits "shapes".
BOOL CMyView::OnCmdMsg(UINT nID, int nCode, void* pExtra,
                       AFX_CMDHANDLERINFO* pHandlerInfo )
{
  //Extend the framework's command route from the view to
  //the application-specific CMyShape that is currently selected.
  //in the view. m_pActiveShape is NULL if no shape object
  //is currently selected in the view.
  if( (m_pActiveShape!=NULL)&&
      m_pActiveShape->OnCmdMsg(nID, nCode, pExtra, pHandlerInfo
    ) )
  return TRUE;
  //If the object(s) in the extended command route don't handle
  //the command, then let the base class OnCmdMsg handle it.
  return CView::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo );
}
//The command handler for ID_SHAPE_COLOR (menu command used to
change
//the color of the currently selected shape ) was added to
//the message map of CMyShape (note, not CMyView ) using ClassWizzard.
//The menu item will be automatically enabled or disabled, depending
//on whether a CMyShape is currently selected in the view, that is
//depending on whether CMyView::m_pActiveView is NULL. It is not
//necessary to implement an ON_UPDATE_COMMAND_UI handler to enable
//or disable the menu item.
BEGIN_MESSAGE_MAP( CMyShape, CCmdTarget )
//{{AFX_MSG_MAP( CMyShape )
   ON_COMMAND( ID_SHAPE_COLOR, OnShapeColor )
//}}AFX_MSG_MAP( )
END_MESSAGE_MAP( )
```

OnFinalRelease

说明：

本函数在对对象的最后一个 OLE 参考或对象对别人的后一个 OLE 参考被释放时，由框架调用。可以覆盖它以进行所需的处理。缺省的实现是删除该对象。

RestoreWaitCursor

说明：

本函数用于在系统光标改变后重置沙漏标（例如，在一个耗时操作的过程中打开一个消息窗口而后又关闭它。）

示例：

```
// The following example illustrates the most common case
// of displaying the hourglass cursor during some lengthy
// processing of a command handler implemented in some
// CCmdTarget-derived class,such as a document or view.
void CMyView::OnSomeCommand( )
{
  BeginWaitCursor( ); // display the hourglass cursor
  // do some lengthy processing
  EndWaitCursor( ); // remove the hourglass cursor
}
// The next example illustrates RestoreWaitCursor
void CMyView::OnSomeCommand( )
{
  BeginWaitCursor( ); // display the hourglass cursor
  // do some lengthy processing
  // The dialog box will normally change the cursor to
  // the standard arrow cursor,and leave the cursor in
  // as the standard arrow cursor when the dialog box is
  // closed.
  CMyDialog dlg;
  dlg.DoModal( );
  // It is necessary to call RestoreWaitCursor here in order
```

```
  // to change the cursor back to the hourglass cursor.
  RestoreWaitCursor( );
  // do some more lengthy processing
  EndWaitCursor( ); // remove the hourglass cursor
}
// In the above example,the dialog was clearly invoked between
// the pair of calls to BeginWaitCursor and EndWaitCursor.
// Sometimes it may not be clear whether the dialog is invoked
// in between a pair of calls to BeginWaitCursor and EndWaitCursor.
// It is permissable to call RestoreWaitCursor,even if
// BeginWaitCursor was not previously called.This case is
// illustrated below,where CMyView::AnotherFunction does not
// need to know whether it was called in the context of an
// hourglass cursor.
void CMyView::AnotherFunction( )
{
  // some processing...
  CMyDialog dlg;
  dlg.DoModal( );
  RestoreWaitCursor( );
  // some more processing...
}
// If the dialog is invoked from a member function of
// some non-CCmdTarget,the you call CWinApp::DoWaitCursor
// with a 0 parameter value to restore the hourglass cursor.
void CMyObject::AnotherFunction( )
{
  CMyDialog dlg;
  dlg.DoModal( );
  AfxGetApp( )->DoWaitCursor(0); // same as CCmdTarget::RestorWaitCursor
}
```

CDialog

CDialog 类是在屏幕上显示的对话框基类。对话框有两类：模态对话框和非模态对话框。模态对话框在应用继续进行之前必须关闭。非模态对话框允许用户执行另外的操作而不必取消或删除该对话框。

一个 CDialog 对象是对话框模板与一个 CDialog 派生类的组合。使用对话框编辑器创建对话框并存入资源之中，然后使用 ClassWizard 创建一个 CDialog 派生类。

同其它窗口一样，对话框从 Windows 中获取信息。在对话框中你会对来自对话框控件的处理消息感兴趣，因为它说明了对话框是如何与用户交互的。ClassWizard 观察对话框每个控件可能产生的消息，可以选择你所希望处理的消息。ClassWizard 将适当的消息映射器入口和消息处理成员函数加到一个新类中。你只需为成员函数编写应用的代码。

如果喜欢，在使用 ClassWizard 时可以编写自己的消息映射器入口和成员函数。

对于大多数普通对话框，可以向对话框派生类中添加成员变量以存储数据。数据是用户向对话框中输入的，或者为用户显示数据。ClassWizard 观察对话框中与数据映射的控件并提示为每一控件创建一个成员变量。同时，为每一变量选择变量类型和取值范围。ClassWizard 将成员变量加入对话框派生类中。

然后，ClassWizard 写入成员函数与对话框控件之间的交换数据自动映射的数据。数据映射使函数可以为对话框控件提供适当的初始值。检索数据并对数据进行有效性检测。

要建立模态对话框，先使用构造程序为对话框派生类构造一个对象。然后调用 DoModal 创建对话框窗口及其控件。如果要建立非模态对话框，在构造程序中为对话框类调用 Create 即可。

还可使用 DLGTEMPLATE 数据结构在内存中建立模板，该结构在联机文档“Win32 SDK”中有描述。当构造了一个 CDialog 对象之后，调用 CreateIndirect 创建非模态对话框，或调用 InitModalIndirect 和 DoModal 来创建模态对话框。

ClassWizard 在覆盖它为新对话框类加入的 CWnd::DoDataExchange 之后写入交换和有效性规则的数据映射。请参阅 CWnd 中的 DoDataExchange 成员函数。

程序和框架都通过调用 CWnd::UpdateData 间接调用 DoDataExchange。当用户单击 OK 按钮关闭模态对话框时，框架调用 UpdateData（如果单击了 Cancel 按钮，将不能获得数据）。

OnInitDialog 的缺省方式也是设置控件的初始值。为得到初始控件，通常要覆盖 OnInitDialog。在所有对话框控件建立且在对话框显示之前，调用 OnInitDialog。

在模态对话框与非模态对话框执行的任意时刻都可以调用 CWnd::UpdateData。

如果要手工创建对话框，应为对话框派生类添加必需的成员变量，并添加已获得数据值的成员函数。

关于 ClassWizard 的更多信息，请参阅联机文档“Visual C++程序员指南”中的“使用 ClassWizard”。

在应用中调用 CWinApp::SetDialogBkColor 为对话框设置背景色。

当用户按下 OK 或 Cancel 按钮，或者调用 EndDialog 时，模态对话框自动关闭。

当实现一个非模态对话框时，总是覆盖 OnCancel 函数并从中调用 DestroyWindows。不要调用 CDialog::OnCancel 基类。因为它会调用 EndDialog，那会使对话框虽然存在但并不可视。模态对话框还应覆盖 PostNcDestory 以防止删除自身。模态对话框厂在框架中构造，不需要用 PostNcDestory 清除。

CDialog

参数：

lpszTemplateName　　包含一个对话框模板资源的空终止字符串。

nIDTemplate　包含对话框模板资源的 ID 号。

pParentWnd　包含对话框的父窗口或所有者窗口对象的指针。如果其为 NULL，则对话框对象的父窗口设置为主应用程序窗口。

说明：

构造一个基于资源的模态对话框。调用构造程序的窗体，其中一个窗体通过模板便可访问对话框。另一个一般使用带 IDD_前缀（如 IDD_DIALOG1）的模板 ID 号实现访问。

从内存中模板来构造模态对话框，首先需要参数和受保护的构造程序，然后调用 InitModalIndirect。

使用上述方法之一构造好对话框之后，调用 DoModal。

构造非模态对话框，使用 CDialog 构造程序中受保护的窗体。构造程序受到保护，因为必须从自己的对话框类中派生得到一个非模态对话框。

构造非模态对话框分两步进行：首先调用构造程序，然后调用 Create 成员函数创建基于资源的对话框，或者从内存模板中调用 CreateIndirect 来创建对话框。

Create

返回值：

如果对话框创建和初始化成功，则返回非零值，否则为 0。

参数：

lpszTemplateName　　包含一个对话框模板资源的空终止字符串。

pParentWnd　指向含有对话框的父窗口对象的指针。如果为 NULL，对话框对象的父窗口设置为应用的主窗口。

nIDTemplate　包含对话框模板资源的 ID 数。

说明：

调用 Create，使用资源中对话框模板来创建非模态对话框。可将调用置于构造程序内部或者在构造程序启动之后调用。

Create 成员函数为访问对话框模板资源提供了两种方法，既可以通过模板名称，也可以模板 ID 号（如 IDD_DIALOG1）。

每种访问方法都会给父窗口传递一个指针。如果 pParentWnd 为 NULL，则应用的主窗口作为其父窗口或所有者窗口来进行创建。

当创建对话框后，Create 成员函数应立即返回。

在父窗口中创建之后，如果要求对话框出现，则使用模板中的 WS_VISIBLE 风格。否则，必须调用 ShowWindow。其它对话框风格及应用，请参阅联机文档“Win32 SDK”中的 DIGTEMPLATE 结构和“Microsoft Visual C++ 6.0 MFC 类库参考手册（二）”中的“Windows 风格”。使用 CWnd::DestroyWindow 函数来删除由 Create 函数创建的对话框。

DoModal

返回值：

整数值，指定了传递给 CDialog::EndDialog 的 nResult 参数值。该函数用于关闭对话框。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。

说明：

调用该成员函数使用模态对话框并返回对话框结果。当对话框处于活动状态时，该函数处理与用户的交互。这使得对话框是模态的，使用户在关闭对话框之前不能与其它窗口交互。

如果用户单击了对话框中的按钮，如 OK 或 Cancel，那么消息处理函数如 OnOK 或 OnCancel 被调用，从而关闭对话框。缺省的 OnOK 成员函数会对对话框数据进行有效性检验和更新，并关闭它得到结果 IDOK。缺省 OnCancel 函数关闭对话框得到结果 IDCANCEL，而不对对话框数据检验或更新，可以覆盖这些消息函数并改变它们的行为。注意 目前 PreTransMessage 被调用来处理模态对话框的消息。

EndDialog

参数：

nResult　对话框返回的值，用于调用者 DoModal。

说明：

调用该成员函数来中止一个模态对话框。该函数返回 nResult。无论模态对话框是何时被创建的，必须使用 EndDialog 来结束处理。

可以随时调用 EndDialog，即使在使用 OnInitDialog 时，即在对话框显示或获得输入焦点之前就关闭它。

EndDialog 不会立即关闭对话框。它设置了一个标记，用以指定在当前消息处理程序返回时就关闭对话框。

NextDlgCtrl

说明：

在对话框内将焦点移到下一个控件。如果焦点位于最后一个控件，则移到第一个控件上。

OnCancel

说明：

当用户在模态对话框或非模态对话框内单击 Cancel 按钮或按 ESC 键时，窗体调用这个成员函数。

覆盖该成员函数，执行 Cancel 按钮动作，缺省方式是调用 EndDialog 来简单中止模态对话框，并使 DoModal 返回 IDCANCEL。

如果在非模态对话框中实现 Cancel 按钮，必须覆盖 OnCancel 成员函数，并在其中调用 DestoryWindow。不能调用基类成员函数，那将会调用 EndDialog，使对话框虽然存在但不可视。

OnInitDialog

返回值：

指定对话框是否对它的一个控件设置输入焦点。如果 OnInitDialog 返回非零值，Windows 将输入焦点设在对话框的第一个控件上，只有在对话框明确将输入焦点设在某控件上，应用返回 0。

说明：

调用这个成员函数是对 WM_INITDIALOG 消息作出的反应。这条消息是在对话框即将显示之前，在 Create ，CreateIndirect 或 DoModal 调用期间发出的。

如果在对话框初始化后需要执行特别处理，覆盖该函数。首先调用基类 OnInitDialog，但不考虑其返回值。正常情况下，覆盖的函数返回 TRUE。Windows 调用 OnInitDialog 函数是通过标准的全局对话框过程（它们对于所有的 Microsoft 基础类库的对话框是通用的），而不是通过消息映射。因此该函数不需要消息映射入口。

OnOK

说明：

当用户按 OK 按钮（ID 是 IDOK）时调用。

覆盖该函数执行 OK 按钮动作。如果对话框包括自动数据检验和交换，缺省方式是对应用的某些变量进行数据的检验和更新。

如果在非模态对话框中实现 OK 按钮，必须覆盖 OnOK 成员函数，并在其中调用 DestoryWindow。不能调用基类成员函数，那将会调用 EndDialog，使对话框虽然存在但不可视。

OnSetFont

参数：

pFont　字体指针。用作对话框中所有字体使用的缺省值。

说明：

书写文本时为对话框控件指定字体。对话框控件使用指定字体作为所有对话框控件的缺省值。

对话框编辑器设置字体，将其作为对话框模板资源的一部分。

PrevDlgCtrl

说明：

把对话框中的焦点移到前一个控件。如果焦点在第一个控件上，则移到对话框中最后一个控件上。

SetDefID

参数：

nID　指定用作缺省按钮的按钮控件的 ID。

说明：为对话框改变缺省按钮。

SetHelpID

参数：

nIDR　指定上下文帮助 ID。

说明：为对话框指定上下文帮助 ID。

CDocument

CDocument 类为用户定义的文档类提供了基本的函数功能。文档类表示了通常用于 File Open 命令打开和使用 File Save 命令保存的数据。

CDocument 支持标准操作，如创建、装载、保存等。框架用 CDocument 定义的界面来操作文档。

应用可支持多种文档，例如文本文档和工作表。每种类型都有一个相关的文档模板。文档模板指定该类文档所使用的资源（如菜单、图标和加速符号表）。每个文档还含有一个 CDocTemplate 对象指针。

用户通过与文档相联系的 CView 对象来与之交互。视图在框架窗口内生成一个文档图象，并解释作用于该文档之上的用户输入。一份文档可以有多个相关的视图，当用户在文档上打开一个窗口时，框架创建一个视图并将其与文档连接。文档模板为每类文档指定了用于显示的视图类型和框架窗口。

文档作为窗口标准命令例程的一部分，接收标准用户界面组件（如 FileSave 菜单项）的命令。文档在活动视图之后接收命令。如果文档未能处理指定的命令，则将其交给管理它的文档模板。

当文档数据被修改时，各个视图都必须反应这些修改。CDocument 提供了 UpdateAllViews 成员函数为视图通知这些变化。框架在关闭之前会提示用户必须存储修改后的文件。

在一个典型的应用中生成一个文档，必须做到以下几点：

·为每种类型的文档从 CDocument 中派生一个类。

·添加保存在文档数据的成员变量。

·为阅读和修改文档数据提供成员函数，文档的视图是这些成员函数最重要的用户。

·在文档类中覆盖 CObject::Serialize 成员函数，从磁盘读取文档数据或将其写入磁盘。

CDocument 支持通过邮件发送文档，如果存在邮件支持（MAPI）的话。关于 CDocument 的更多信息，请参阅联机文档“Visual C++程序员指南”中的“串行化(对象持久化)”，“文档/视图结构主题”和“文档/视图创建”。

AddView

参数：

pView　被添加的视图指针。

说明：

调用该成员函数将视图添加到文档中。该函数将指定视图加入与文档相联系的视图列表之中。函数还设置指向文档的视图指针。当添加一个新创建的视图对象到文档时，框架调用该函数。这是对 File Open，FileNew，新的 Windows 或窗口被分隔等作出的反应。

仅在手工创建和添加视图时调用该函数。通常通过定义与文档类、视图类和框架窗口类相联系的 CDocTemplate 对象来连接框架窗口与视图、文档。

示例：

```
// The following example toggle two views in an SDI(single document
// interface) frame window.A design decision must be made as to
// whether to leave the inactive view connected to the document,
// such that the inactive view continues to receive OnUpdate
// keep the inactive view continuously in sync with the document,even
//though it is inactive However,doing so incurs a performance cost,
// as well as the programming cost of implementing OnUpdate hints.It may be less expensive,in terms of performance and/or programming,
// to re-sync the inactive view with the document only with it is
// reactivated. This example illustrates this latter approach,by
// reconnecting the newly active view and disconnecting the newly
// inactive view,via calls to CDocument::AddView and RemoveView.
BOOL CMainFrame::OnViewChange(UNIT nCmdID)
{
  CView* pViewAdd;CView* pViewRemove;
  CDocument* pDoc = GetActiveDocument();
  UNIT nCmdID;nCmdID = LOWORD(GetCurrentMessage()->wParam);
  if ((nCmdID = = ID_VIEW_VIEW1)&&(m_currentView = =1)) return;
  if ((nCmdID = = ID_VIEW_VIEW2)&&(m_currentView = =2))return;
  if (nCmdID = = ID_VIEW_VIEW2)
  {
    if (m_pView2 = = NULL)
    {
```

```
    m_pView1 = GetActiveView();
    m_pView2 = new CMyView2;
    //Note that if onSize has beenoverridden in CMy View2
    //and GetDoument()is used in this overrideit can
    //cause assertions and,if theassertions are ignored,
    //cause access violation.
    m_pView2->Create(NULL, NULL, AFX_WS_DEFAULT_VIEW, rectDefault,
                this, AFX_IDW_PANE_FIRST+1, NULL
               );
  }

  pViewAdd = m_pView2;pViewRemove = m_pView1;
  m_currentView = 2
}
else
{
   pViewAdd = m_pView1;
   pViewRemove = m_pView2;
   m_currentView = 1;
}
// Set the child i.d. of the active view to AFX_IDW_PANE_FIRST,
// so that CFrameWnd::RecalcLayout will allocate to this
// "first pane" that portion of the frame window's client area
// not allocated to control bars.Set the child i.d. of the
// other view to anything other than AFX_IDW_PANE_FIRST;this
// examples switches the child id's of the two views.

int nSwitchChildID = pViewAdd->GetDlgCtrlID();
pViewAdd->SetDlgCtrlID(AFX_LDW_PANE_FIRST);
pViewRemove->SetDlgCtrlID(nSwitchChildID);

// Show the newly active view and hide the inactive view.

pViewAdd->ShowWindow(SW_SHOW);
pViewRemove->ShowWindow(SW_HIDE);

// Connect the newly active view to the document,and
// disconnect the inactive view.
```

```
    pDoc->AddView(pViewAdd);
    pDoc->RemoveView(pViewRemove);
    SetActiveView(pViewAdd);
    RecalcLayout();
  }
```

AddView

参数：

pView　被添加的视图指针。

说明：

调用该成员函数将视图添加到文档中。该函数将指定视图加入与文档相联系的视图列表之中。函数还设置指向文档的视图指针。当添加一个新创建的视图对象到文档时，框架调用该函数。这是对 File Open，FileNew，新的 Windows 或窗口被分隔等作出的反应。

仅在手工创建和添加视图时调用该函数。通常通过定义与文档类、视图类和框架窗口类相联系的 CDocTemplate 对象来连接框架窗口与视图、文档。

示例：

```
// The following example toggle two views in an SDI(single document
// interface) frame window.A design decision must be made as to
// whether to leave the inactive view connected to the document,
// such that the inactive view continues to receive OnUpdate
// keep the inactive view continuously in sync with the document,even
//though it is inactive However,doing so incurs a performance cost,
// as well as the programming cost of implementing OnUpdate hints.It may be less expensive,in terms of performance and/or programming,
// to re-sync the inactive view with the document only with it is
// reactivated. This example illustrates this latter approach,by
// reconnecting the newly active view and disconnecting the newly
// inactive view,via calls to CDocument::AddView and RemoveView.
BOOL CMainFrame::OnViewChange(UNIT nCmdID)
{
  CView* pViewAdd;CView* pViewRemove;
  CDocument* pDoc = GetActiveDocument();
```

```
UNIT nCmdID;nCmdID = LOWORD(GetCurrentMessage()->wParam);
if ((nCmdID = = ID_VIEW_VIEW1)&&(m_currentView = =1)) return;
if ((nCmdID = = ID_VIEW_VIEW2)&&(m_currentView = =2))return;
if (nCmdID = = ID_VIEW_VIEW2)
{
  if (m_pView2 = = NULL)
  {
    m_pView1 = GetActiveView();
    m_pView2 = new CMyView2;
    //Note that if onSize has beenoverridden in CMy View2
    //and GetDoument()is used in this overrideit can
    //cause assertions and,if theassertions are ignored,
    //cause access violation.
    m_pView2->Create(NULL, NULL, AFX_WS_DEFAULT_VIEW, rectDefault,
                this, AFX_IDW_PANE_FIRST+1, NULL
               );
  }

  pViewAdd = m_pView2;pViewRemove = m_pView1;
  m_currentView = 2
}
else
{
   pViewAdd = m_pView1;
   pViewRemove = m_pView2;
   m_currentView = 1;
}
// Set the child i.d. of the active view to AFX_IDW_PANE_FIRST,
// so that CFrameWnd::RecalcLayout will allocate to this
// "first pane" that portion of the frame window's client area
// not allocated to control bars.Set the child i.d. of the
// other view to anything other than AFX_IDW_PANE_FIRST;this
// examples switches the child id's of the two views.

int nSwitchChildID = pViewAdd->GetDlgCtrlID();
pViewAdd->SetDlgCtrlID(AFX_LDW_PANE_FIRST);
pViewRemove->SetDlgCtrlID(nSwitchChildID);
```

```
    // Show the newly active view and hide the inactive view.

    pViewAdd->ShowWindow(SW_SHOW);
    pViewRemove->ShowWindow(SW_HIDE);

    // Connect the newly active view to the document,and
    // disconnect the inactive view.

    pDoc->AddView(pViewAdd);
    pDoc->RemoveView(pViewRemove);
    SetActiveView(pViewAdd);
    RecalcLayout();
}
```

CDocument

说明：

构造一个 CDocument 对象。由框架负责创建文档。覆盖 OnNewDocument 进行文档初始化，这在 SDI 应用中特别重要。

DeleteContents

说明：

由框架调用来删除文档数据而不删除 CDocument 对象本身。在文档被删除之前调用。它还被调用来确保在重新使用文档之前该文档为 NULL。这对 SDI 应用尤其重要。因为它只使用一个文档。无论用户是何时创建和打开文档的，文档都可以被再次使用。调用这个函数来实现“Edit Clear All”或类似命令将文档数据全部删除。缺省方式是函数什么也不做。覆盖该函数来删除文档中的数据。

示例：

```
// This example is the handler for an Edit Clear All Command.
void CMyDoc::OnEditClearAll()
{
  DeleteContents();
  UpdateAllViews(NULL);
```

```
}

void CMyDoc::DeleteContents()
{
  // Reinitialize document data here.
}
```

GetDocTemplate

返回值：

返回该文档类型模板的指针。如果文档不由该文档模板管理，则返回 NULL。

说明：

调用该函数为文档类型的文档模板获取指针。

示例：

```
// This example accesses the doc template object to construct
// a default document name such as SHEET.XLS,where "sheet"
// is the base document name and *.xls* is the file extension
// for the document type.
CString strDefaultDocName, strBaseName, strExt;
CDocTemplate* pDocTemplate = GetDocTemplate();
if (!pDocTemplate->GetDocString(strBaseName,CDocTemplate::docName) ||
    !pDocTemplate->GetDocString(strExt, CDocTemplate::filterExt)
   )
{
  AfxThrowUserException();
  // These doc template strings will
  // be available if you created the application using AppWizard
  // and specified the file extension as an option for
  // the document class produced by AppWizard.
}
strDefaultDocName = strBaseName + strExt;
```

GetFile

返回值：指向 CFile 对象的指针。

参数：

lpszFileName　　所需要的文件的路径的字符串。该路径可为相对路径或绝对路径。

pError　已经存在的文件异常对象的指针，该对象表明操作的进行状态。

nOpenFlags　共享和访问模式。在打开文件时指定要进行的动作。可使用位操作符 OR 来组合在 CFile 构造程序 CFile::CFile 列出的选项,需要一个访问许可和一个共享选项。modeCreate 和 modeNoInherit 模式是可选的。

说明：调用该函数为 CFile 对象获得指针。

GetFirstViewPosition

返回值：

返回 POSITION 值，它用于 GetNextView 成员函数的迭代。

说明：

调用该函数获得与文档相关的视图列表中的第一个视图。

示例：

```
// To get the first view in the list of views:
POSITION pos = GetFirstViewPositon ();
CView* pFirstView = GetNextView( pos );
// This example uses CDocument::GetFirstViewPosition
// and GetNextView to repaint each view.
void CMyDoc::OnRepaintAllViews()
{
  POSITION pos = GetFirstViewPosition();
  while (pos != NULL)
  {
    CView* pView = GetNextView(pos);
    pView->UpdateWindow();
  }
```

```
}
// An easier way to accomplish the same result is to call
// UpdateAllViews(NULL);
```

GetNextView

返回值：返回由 rPosition 标识的视图指针。

参数：

rPosition　　POSITION 值的一个参考。它是由前一次调用 GetNextView 和 GetFirstViewPosition 成员函数返回得到的。其值不能为 NULL。

说明：

调用该函数来迭代所有的文档视图。函数通过 rPosition 返回视图，并将 rPosition 设置为列表中下一个视图的 POSITION 值。如果获得的视图是列表中的最后一个，则设置 rPosition 为 NULL。

示例：

```
// This example uses CDocument::GetFirstViewPosition
// and GetNextView to repaint each view.
void CMyDoc::OnRepaintAllViews()
{
  POSITION pos = GetFirstViewPosition();
  while (pos != NULL)
  {
    CView* pView = GetNextView(pos);
    pView->UpdateWindow();
  }
}
// An easier way to accomplish the same result is to call
// UpdateAllViews(NULL);
```

GetPathName

返回值：

返回文档全限定路径。如果文档不是保存在磁盘文件中或不与其相关，则返回空。

说明：调用该函数获得文档的磁盘路径。

GetTitle

返回值：文档标题。

说明：调用该函数获得文档标题，它常由文档文件名派生出。

IsModified

返回值：

如果自从上一次保存以来被修改过，则返回非零值，否则为 0。

说明：

调用该成员函数判断文档自从上一次保存以来是否被修改过。

OnChangedViewList

说明：

在往文档中添加或移去视图后，由框架调用该函数。缺省方式是函数检查最后一个视图是否被移去，如果是，则删除该文档。如果要在框架添加/移去视图时进行特别的处理，覆盖该函数。例如，要一份文档保持打开，即使已没有与之相连的视图，则可覆盖该函数。

OnCloseDocument

说明：

在文档关闭时被框架调用，通常作为 File Close 命令的一部分。缺省方式是函数删除所有用于观察该文档的框架，关闭视图，清除文档内容，然后调用 DeleteContents 成员函数删除文档数据。如果框架关闭文档时需要进行特别清除处理，覆盖此函数。例如，如果文档表示数据库中的一个记录，用户可能需要覆盖该函数来关闭数据库。应从函数覆盖中调用这个函数的基类。

OnFileSendMail

说明：

通过常驻邮件主机将文档作为附件来发送消息。OnFileSendMail 调用 OnSave-Document 将未命名的和修改的文档串行化（保存）为临时文件，然后由电子邮件发送。如果文档未被修改，则不需要临时文件发送原始文档。如果没有装入 MAPI32. DLL，OnFileSendMail 装入它。

COleDocument 使用的 OnFileSendMail 需要正确地组合文件。

如果有邮件支持，OnFileSendMail 通过邮件发送文档。

要了解更多的详细信息，请参阅联机文档“Visual C++程序员指南”中的“MFC 中的 MAPI 支持”。

OnNewDocument

返回值：如果文档初始化成功，则返回非零值，否则为 0。

说明：

由框架调用，作为 File New 命令的一部分。缺省实现方式是调用该函数确保文档为空，并标明新文档是未被修改的。覆盖该函数进行新文档数据结构的初始化，应从函数覆盖中调用该函数的基类版本。

如果用户在 SDI 应用中选择了 File New 命令，框架使用该函数对存在的文档进行重新初始化，而不是创建一个新的文档。如果用户在 MDI 应用中选择了 File New 命令，框架每次建立一个新的文档并调用该函数来进行初始化。必须将初始化代码放在这个函数中而不是放在 FileNew 命令的构造程序中，该程序在 SDI 应用中有效。

示例：

```
// The following examples illustrate alternative methods of
// initializing a document object.
// Method 1: In an MDI application,the simplest place to do
// initialization is in the document constructor.The framework
// always creates a new document object for File New or File Open.
CMyDoc::CMyDoc()
{
  // Do initialization of MDI document here.
  // ...
```

```
}

// Method 2: In an SDI or MDI application,do all initialization
// in an override of OnNewDocument,if you are certain that
// the initialization is effectively saved upon File Save
// and fully restored upon File Open,via serialization.
BOOL CMyDoc::OnNewDocument()
{
  if (!CDocument::OnNewDocument()) return FALSE;
  // Do initialization of new document here.
  return TRUE;
}

// Method 3: If the initialization of your document is not
// effectively saved and restored by serialization(during File Save
// and File Open),the implement the initialization in single
// function(named InitMyDocument in this example).Call the
// shared initialization function from override of both
// OnNewDocument and OnOpenDocument.
BOOL CMyDoc::OnNewDocument()
{
  if (!CDocument::OnNewDocument()) return FALSE;
  InitMyDocument(); //call your shared initialization function
  // If your new document object requires additional initialization
  // not necessary when the document is deserialized via File Open,
  // then perform that additional initialization here.
  return TRUE;
}
```

OnOpenDocument

返回值：如果文档装载成功，则返回非零值，否则为 0。

参数：

lpszPathName　　指向打开的文档的路径的指针。

说明：

由框架调用该函数，作为 File Open 命令的一部分。缺省实现方式是打开指定文件，调用 DeleteContents 来确保文档是空的，调用 CObject::Serialize 阅读文件内容，然后标明文档未被修改。如果不使用归档机制或文件机制，则覆盖该函数。例如，应用中的文档表示一个数据库的记录而非文件。

如果用户在 SDI 应用中选择了 File Open 命令，框架使用该函数对 CDocument 对象进行重新初始化，而不是创建一个新的文档。如果用户在 MDI 应用中选择了 File Open 命令，框架每次建立一个新的 CDocument 对象并调用该函数来进行初始化。必须将初始化代码放在这个函数中而不是放在 File Open 命令的构造程序中，该程序在 SDI 中有效。

示例：

```
// The following examples illustrate alternative methods of
// initializing a document object.
// Method 1: In an MDI application,the simplest place to do
// initialization is in the document constructor.The framework
// always creates a new document object for File New or File Open.
CMyDoc::CMyDoc()
{
  // Do initialization of MDI document here.
  // ...
}

// Method 2: In an SDI or MDI application,do all initialization
// in an override of OnNewDocument,if you are certain that
// the initialization is effectively saved upon File Save
// and fully restored upon File Open,via serialization.
BOOL CMyDoc::OnNewDocument()
{
  if (!CDocument::OnNewDocument()) return FALSE;
  // Do initialization of new document here.
  return TRUE;
}

// Method 3: If the initialization of your document is not
// effectively saved and restored by serialization(during File Save
// and File Open),the implement the initialization in single
// function(named InitMyDocument in this example).Call the
// shared initialization function from overrides of both
```

```
// OnNewDocument and OnOpenDocument.
BOOL CMyDoc::OnNewDocument()
{
  if (!CDocument::OnNewDocument()) return FALSE;
  InitMyDocument(); //call your shared initialization function
  // If your new document object requires additional initialization
  // not necessary when the document is deserialized via File Open,
  // then perform that additional initialization here.
  return TRUE;
}
```

OnSaveDocument

返回值：如果文档保存成功，则返回非零值，否则为 0。

参数：

lpszPathName　　用于保存文件的全限定路径指针。

说明：

由框架调用该函数作为 File Save 或 File Save As 命令的一部分。缺省实现方式是打开指定文件，调用 CObject::Serialize 将文档数据写入文件，然后标识文档是未被修改的。如果在框架中保存文档时要进行特别的处理，覆盖该函数。例如，可以用应用中的文档表示一个数据库的记录而非文件。

PreCloseFrame

参数：

pFrame　含有 CDocument 对象的 CFrameWnd 的指针。

说明：

在框架窗口被删除前由框架调用该成员函数。它可以被覆盖以提供定制的清除，但要保证基类也被调用。

缺省地，PreCloseFrame 在 CDocument 中什么也不做。CDocument 派生类 COleDocument 和 CRichEditDoc 使用该函数

ReleaseFile

参数：

pFile　　将要释放的 CFile 对象的指针。

bAbort　指定是否用 CFile::Close 或 CFile::Abort 来释放文件。如果使用 CFile::Close 来释放文件，则返回 FALSE；如果使用 CFile::Abort ，则返回 FALSE。

说明：

成员函数调用来释放文件，使其可以为其它应用使用。如果 bAbort 为 TRUE，ReleaseFile 调用 CFile::Abort，并释放文件。CFile::Abort 不产生异常。如果 bAbort 为 FALSE，ReleaseFile 调用 CFile::Close ，并释放文件。

在文件释放之前需要用户动作时，覆盖该函数。

RemoveView

参数：

pView　被移去的视图指针。

说明：

调用该成员函数使视图与文档分离。函数将与文档相关的视图列表中的特定视图移去。同时设置该视图的文档指针为 NULL。当框架窗口关闭或一块被拆分的窗口关闭关闭时，由框架调用该函数。

只有在人工分离视图时才调用该函数。函数定义与文档类、视图类和框架窗口类相关的 CDocTemplate 对象，使框架与文档和视图分离。

SaveModified

返回值：

如果安全地继续并关闭文档，则返回非零值；如果不能关闭文档，则返回 0。

说明：

在被修改的文档关闭之前由框架调用。缺省方式是如果文档作了修改，函数显示消息框，询问用户是否保存了文档变化。如果程序需要不同的提示程序，则覆盖该函数。这是一个高级函数覆盖。

SetModifiedFlag

参数：

bModified　　文档是否被修改的标记。

说明：

在对文档作了修改之后调用该函数。连续调用以确保在关闭之前框架提示用户保存这些变化。通常使用 bModified 参数的缺省值 TRUE。要标记文档是未被修改的，调用带 FALSE 值的函数。

SetPathName

参数：

lpszPathName　　用作文档路径的字符串的指针。

bAddToMRU　　决定文件名是否加在 MRU 文件列表中，如果为 TRUE，则表示加入；如果为 FALSE，表示不加入。

说明：

调用该函数为文档的磁盘文件指定全限定路径。路径是否添加到应用的 MRU 列表上，取决于 bAddToMRU 的取值。要注意某些文档并未与磁盘文件相联系。只有在覆盖窗口所使用的缺省打开方式时，才调用该函数。

SetTitle

参数：

lpszTitle 用作文档标题的字符串的指针。

说明：

调用该函数来指定文档标题（显示在框架窗口的标题栏上）。调用该函数更新所有显示该文档的框架窗口标题。

UpdateAllViews

参数：

pSender　修改文档的视图的指针。如果所有视图都被更新，则返回 NULL。

lHint　包含修改的信息。

pHint　包含修改信息的对象指针。

说明：

在文档修改之后调用这个函数。应在调用 SetModifiedFlag 之后调用。除了 pSender 指定的视图之外，函数通知与文档连接的各个视图文档已被修改的消息。通常在用户通过视图改动文档后调用该函数。

除了发送视图外，这个函数为每个文档视图调用 CView::OnUpdate ，并传递 lHint 和 pHint 值，使用这些参数将文档的修改情况传递给视图。可以使用 lHint 为信息编码，（或者）定义一个 CObject 派生类保存修改信息，并使用 pHint 传递一个该类对象。为优化视图更新，在 CView 派生类中可覆盖 CView::OnUpdate 成员函数。

CEdit

CEdit 类提供了 Windows 编辑控件中的功能。编辑控件是一个子窗口矩形，用户可以向其中输入文本。

可以通过对话模板或直接从代码中创建一个编辑控件。在两种情形下，首先调用 CEdit 构造程序构造 CEdit 对象，再调用 Create 成员函数创建 Windows 编辑控件并将其与 CEdit 对象连接。

构造在 CEdit 的派生类中可以单步实现。为派生类编写构造程序并从构造程序中调用 Create。

CEdit 从 CWnd 继承了重要的功能，要在 CEdit 对象中设置或获取文本，使用 CWnd 成员函数 SetWindowText 和 GetWindowText，可以设置和得到编辑控件的全部内容，即使它是一个多行控件。如果编辑控件是多行的，使用 CEdit 成员函数 GetLine，GetSel，GetSel 和 ReplaceSel 来获取和写入控件的部分文本。

如果要处理编辑控件发往其父类（通常是一个 CDialog 派生类）的通知消息，则向父类中为每一消息添加一个消息映射入口和消息处理成员函数。

各消息映射入口可采用如下形式：

ON_Notification(id,memberFxn)

其中 id 指定了发送通知的编辑控件的子窗口 ID，memberFxn 为你写好的处理通知的父成员函数的名字。

父函数形式如下：

afx_msg void memberFxn();

下面是一组可能的消息映射入口，以及在何种情况下向父类发送的描述：

· ON_EN_CHANGE 用户采取的行动可能会改变编辑控件的文本。与 EN_UPDATE 通知消息不同，该通知是在 Windows 更新显示之后发送的。

· ON_EN_ERRSPACE 编辑控件不能为特定请求分配足够的空间。

· ON_EN_HSCROLL 用户单击了编辑控件中的水平滚动条，父窗口在屏幕更新之前被通知。

· ON_EN_KILLFOCUS 编辑控件失去输入焦点。

· ON_EN_MAXTEXT 当前输入超过了为编辑控件指定的数目，并作截尾处理。当编辑控件不具有 ON_EN_HSCROLL 风格且要输入的字符会超过编辑控件的宽度时，发送消息。当编辑控件不具有 ON_EN_VSCROLL 风格且要输入的字符会超过编辑控件的高度时，也会发送消息。

· ON_EN_SETFOCUS 编辑控件获得焦点。

· ON_EN_UPDATE 编辑控件将要显示变动的文本。在控件对文本格式化之后但在显示文本之前发送消息，以便在必要时改变窗口尺寸。

· ON_EN_VSCROLL 用户单击了编辑控件中的垂直滚动条，父窗口在屏幕更新之前被通知。

如果在对话框内创建 CEdit 对象，CEdit 对象在用户关闭对话框时自动被删除。

如果使用对话框编辑器从对话资源中创建 CEdit 对象，CEdit 对象在用户关闭对话框时自动被删除。

如果在窗口内创建 CEdit 对象，也需要删除它。如果在栈上创建 CEdit 对象，它被自动删除。如果使用 new 函数在堆上创建 CEdit 对象，在用户中止编辑控件时，必须对其调用 delete 来删除它。如果在 CEdit 对象中分配存储空间，覆盖 CEdit 析构程序来处理分配情况。

CEdit

说明：

构造一个 CEdit 对象。使用 Create 来创建 Windows 编辑控件。

Clear

说明：

调用该函数来删除（清除）编辑控件中当前选中的文本。

由 Clear 进行的操作可以通过调用 Undo 成员函数撤销。

要删除当前选定文本并将其拷贝到剪贴板上，调用 Cut 成员函数。要了解更多的信息，请参阅 Win 32 文档中的 WM_CLEAR。

Create

返回值：初始化成功，则返回非零值，否则为 0。

参数：

dwStyle 指定编辑控件的风格。可以组合使用控件的编辑风格。

rect 指定控件的尺寸和位置。可以是 RECT 结构或 CRect 对象。

pParentWnd 指定编辑控件的父窗口（通常使用 CDialog ）。其值不能为 NULL。

nID 指定编辑控件的 ID。

说明：

构造 CEdit 对象分两步。首先调用 CEdit 构造程序，再调用 Create，这样就创建了一个 Windows 编辑控件，并将其与 CEdit 对象连接。当执行 Create 时，Windows 发送 WM_NCCREATE ，WM_NCCALCSIZE ，WM_CREATE 和 WM_GETMINMAXINFO 消息到编辑控件。

缺省地，这些消息由 CWnd 基类中的 OnNcCalcSize，OnCreate，OnNcCreate 和 OnGetMinMaxInfo 成员函数处理。要扩展缺省的消息处理，先从 CEdit 派生一个类，为新类添加消息映射并覆盖上述消息处理成员函数。例如，覆盖 OnCreate 为新类执行所需要的初始化操作。

可以为编辑控件应用如下的风格：

- WS_CHILD 总是采用
- WS_VISIBLE 经常采用
- WS_DISABLED 很少采用
- WS_GROUP 组合控件
- WS_TABSTOP 按制表键次序包含编辑控件

Cut

说明：

调用该函数来删除（剪切）在编辑控件中的当前选定文本，并将其用 CF_TEXT 格式拷贝到剪贴板中。

由 Cut 执行的删除可以由 Undo 成员函数来撤销。

删除当前选定部分而不将已删除文本置于剪贴板，调用 Clear 成员函数。要了解更多的信息，请参阅 Win 32 文档中的 WM_CUT。

GetFirstVisibleLine

返回值：

可视的最顶端行的行号（行号由 0 开始），对单行编辑控件来说，返回值为 0。

说明：

调用此成员函数决定编辑控件中可视的最顶端行的行号。

要了解更多信息，请参阅 Win32 文档中的 EM_GETFIRSTVISIBLELINE。

GetHandle

返回值：

一个用于标识编辑控件内容的局部内存句柄。如果发生错误，例如发送信息到一个单行编辑控件，则返回值为 0。

说明：

调用此成员函数来获取一个多行编辑控件中当前分配的内存句柄。此句柄是一个局部内存句柄，可被任何局部 Windows 存储函数作为一个参数来获得。

GetHandle 仅仅被多行编辑控件处理。

在一个多行编辑控件的对话框中调用此成员函数时，对话框必须由 DS_LOCALEDIT 的样式标志集生成。如果不是，虽然也可以得到一个非零的返回值，但此返回值不可被使用。

注意：

GetHandle 不可在 Windows 95 下运行。如果在 Windows 95 下调用，会返回 NULL。GetHandle 可在 Windows NT 3.51 版及以上使用。

要了解有关的更多信息，请参阅 Win32 文档中的 EM_GETHANDLE。

GetLimitText

返回值：对当前 CEdit 对象的文本大小限制，以字节计算。

说明：

调用此成员函数来获取该 CEdit 对象的文本大小限制，文本限制是此编辑控件可以接收的文本的最大长度（以字节计算）。

注意：此成员函数仅在 Windows 95 和 Windows NT 4.0 以上版本中可用。

要了解更多信息，请参阅 Win32 文档中的 EM_GETLIMITTEXT。

GetLine

返回值：

实际拷贝的字节数。如果由 nIndex 指定的行号大于此编辑控件的行数，则返回值为 0。

参数：

nIndex　指定从多行编辑控件中检索的行的行号，行号由 0 指定。对单行编辑控件，此参数被忽略。

lpszBuffer　指向获取此行备份的缓冲区。缓冲区的第一个字必须指定能被拷贝到缓冲区的最大字节数。

nMaxLength　指定能被拷贝到缓冲区的最大字节数，GetLine 在调用 Windows 之前将此值放置到 lpszBuffer 的第一个字中。

说明：

调用此成员函数从编辑控件中获取文本的一行并将其放置到 lpszBuffer 缓冲区。

被拷贝的行不包括空终止符。

要了解更多信息，请参阅 Win32 文档中的 EM_GETLINE。

GetLineCount

返回值：

在多行编辑控件中的包含的一个整数总行数。如果没有向控件输入任何文本，则返回值为 1。

说明：

调用此成员函数获取一个多行编辑控件中的总行数。

此函数仅应用于多行编辑控件。

GetModify

返回值：

如果编辑控件的内容被改变，则返回值为非零，否则为 0。

说明：

调用此成员函数测试编辑控件的内容是否被改变。Windows 有一个内部标记来表明编辑控件的内容是否被改变。当编辑控件首次被创建时此标记被清除，在调用 SetModify 成员函数时也被清除。

要了解更多信息，请参阅 Win32 文档中的 EM_GETMODIFY。

GetPasswordChar

返回值：

指定在用户输入字符处显示的字符。如果无密码，则返回 NULL。

说明：

调用此成员函数获取在用户输入密码时所显示的密码字符。

如果编辑控件是用 ES_PASSWORD 风格建立的，则缺省的密码字符为一个星号（* ）。

要了解更多信息，请参阅 Win32 文档中的 EM_GETPASSWORDCHAR。

GetRect

参数：

lpRect　指向 RECT 结构以接收格式化矩形。

说明：

调用此成员函数获取一个编辑控件的格式化矩形。此格式化矩形为文本的边界矩形，与编辑控件窗口的大小无关。

多行编辑控件的格式化矩形可以被 SetRect 和 SetRectNP 成员函数改变。

要了解更多信息，请参阅 Win32 文档中的 EM_GETRECT。

LineFromChar

返回值：

返回由 nIndex 指定的字符索引的行号，此行号从 0 开始。如果 nIndex 为-1，则返回所选部分第一个字符的行号，如果无选定部分，则返回当前行号。

参数：

nIndex　包含编辑控件文本中所需字符的基于 0 的索引值，或者包含-1。如果为-1 则指定为当前行，即包含脱字符的行。

说明：

调用此成员函数获取包含指定字符索引的行的行号，字符索引指编辑控件中从开始到指定字符的字符数。

此成员函数仅适用于多行编辑控件。

要了解更多信息，请参阅 Win32 文档中的 EM_LINEFROMCHAR。

Paste

说明：

调用此成员函数将剪贴板上的数据插入 CEdit 的插入点，仅在剪贴板上的数据具有 CF_TEXT 格式时数据才可以被插入。

要了解更多信息，请参阅 Win32 文档中的 WM_PASTE。

SetHandle

参数：

hBuffer 包含一个指向局部内存的句柄。此句柄必须已由 LocalAlloc Windows 函数使用 LMEM_MOVEABLE 标记创建。该存储区被认为包含一个带空终止符的字符串，如果不是这样，则缓冲区的第一个字符应被设置为 0。

说明：

调用此成员函数设置一个可被多行编辑控件使用的局部内存句柄。编辑控件便可以使用此缓冲区来存储当前显示的文本，而不必分配自己的缓冲区。

此函数仅对多行编辑控件有效。

当应用设置一个新的存储句柄时，应使用 GetHandle 成员函数获取一个当前缓冲内存句柄，并使用 LocalFree Windows 函数释放此缓冲区。

此函数清除撤消缓冲区（CanUndo 成员函数返回 0）和内部修改标记（GetModify 成员函数返回 0），编辑控件被重新设置。

仅在使用 DS_LOCALEDIT 风格标志设置构造一个多行编辑控件对话框后，才可以在此对话框中使用此成员函数。

注意：

此函数不可在 Windows 95 中使用，如果在 Windows 95 中使用 GetHandle 会返回 NULL。此函数仅在 Windows NT 3.51 以上的版本中使用。

要了解更多信息，请参阅 Win32 文档中的 EM_SETHANDLE，LocalAlloc，LocalFree。

SetModify

参数：

bModified TRUE 表示文本被改变了，FALSE 表示没有改变，缺省情况下设定了改变标志。

说明：

调用此成员函数设置或清除编辑控件的改变标志。改变标记表明文本是否被改变。当用户改变文本时，此标志被自动设置，它的值在调用 GetModify 成员函数时获取。

要了解更多信息，请参阅 Win32 文档中的 EM_SETMODIFY。

GetPasswordChar

参数：

ch　指定在用户输入字符处显示的字符。如果值为 0，则显示输入的实际字符。

说明：

调用此成员函数在编辑控件中设置或清除用户输入文本时所显示的密码字符。

此成员函数对多行编辑控件无效。

当调用 SetPasswordChar 成员函数时，CEdit 将用 ch 所指定的字符替代所有可视的字符。

如果编辑控件是用 ES_PASSWORD 风格建立的，则缺省的密码字符被设置为一个星号（* ）。此风格在 SetPasswordChar 以 ch=0 调用时删除。

要了解更多信息，请参阅 Win32 文档中的 ES_SETPASSWORDCHAR。

SetReadOnly

返回值：

操作成功，则返回非零值；当发生错误时为 0。

参数：

bReadOnly　指定设置还是去掉编辑控件的只读状态。如果为 TRUE 值，则设置为只读状态；如果为 FALSE 值，则设置为可读写状态。

说明：

调用此成员函数设置编辑控件的只读状态。当前的只读状态可由 CWnd::GetStyle 的返回值的 ES_READONLY 标志测出。要了解更多信息，请参阅 Win32 文档中的 EM_SETREADONLY。

GetRect

参数：

lpRect　指向 RECT 结构或 CRect 对象的指针，指定格式化矩形的新的尺寸。

说明：

调用此成员函数用指定坐标设置一个编辑控件的矩形的尺寸。此成员函数仅对多行编辑控件有效。

使用 SetRect 函数设置一个对多行编辑控件的格式化矩形。此格式化矩形为文本的边界矩形，与编辑控件窗口的大小无关。当编辑控件首次被创建时，格式化矩形与用户的编辑控

件窗口区一样。使用 SetRect 成员函数后，应用程序可以使格式化矩形大于或小于编辑控件窗口。

如果编辑控件没有滚动条，在格式化矩形大于窗口时，文本将被剪切而不是被覆盖。如果编辑控件包括一个边界，则格式化矩形的大小将被边界的大小变低。如果用 GetRect 成员函数的返回值来调整矩形的大小，在传递矩形之前应去掉边界大小。

调用 SetRect 函数时，编辑控件的文本格式和显示方式将被重新设置。

要了解更多信息，请参阅 Win32 文档中的 EM_SETRECT。

SetTabStops

返回值：如果制表键被设置，则返回非零值，否则为 0。

参数：

cxEachStop　指定在每个 cxEachStop 对话单位设置制表键停止。

nTabStops　指定包含在 rgTabStops 中的制表键停止个数。此个数必须大于 1。

rgTabStops　是一个指向无符号整数数组的指针，此数组指定了对话单位的制表键停止个数。一个制表键单元是一个水平或垂直距离，一个水平制表键单元等于宽度相同的当前对话框的四分之一，一个垂直制表键单元等于高度相同的当前对话框的八分之一，对话基本单元是基于当前系统字体的高度和宽度计算的，WindowsGetDialogBaseUnits 函数以像素形式返回当前对话基本单元。

说明：

调用此成员函数在一个多行编辑控件中设置制表键停止。当文本被拷贝到多行编辑控件时，文本中的任何制表键之间均会产生一段空白。

要将缺省的制表键大小为 32 个对话单位，可不带参数调用此成员函数。如果大小比 32 大，用 cxEachStop 作参数调用；设置一个数组形式的制表键停止，可使用双参数调用。

此成员函数仅适用于多行编辑控件。

SetTabStops 不会自动重画编辑窗口。如果要改变已在文本控件中的文本的制表键停止，应调用 CWnd::InvalidateRect 来重画编辑窗口。

要了解更多信息，请参阅 Win32 文档中的 EM_SETTABSTOPS。

Undo

返回值：

对于单行编辑控件总是返回非零值。对于多行编辑控件，如果操作成功，则返回非零值，失败则返回 0。

说明：

调用此成员函数撤消编辑控件的最后一次操作。

撤销操作也可以被撤消。例如，可以第一次调用撤销来保存被删文本，在没有别的操作发生时，可以再次调用撤消操作将文本删除。

要了解更多信息，请参阅 Win32 文档中的 EM_UNDO。

CFile

CFile 是 MFC 文件类的基类，它直接提供非缓冲的二进制磁盘输入/输出设备，并直接地通过派生类支持文本文件和内存文件。CFile 与 CArchive 类共同使用，支持 MFC 对象的串行化。

该类与其派生类的层次关系让程序通过多形 CFile 接口操作所有文件对象。例如，一个内存文件相当一个磁盘文件。

使用 CFile 及其派生类进行一般目的的磁盘 I/O，使用 ofstream 或其它 Microsoft 输入输出流类将格式化文本送到磁盘文件。

通常，一个磁盘文件在 CFile 构造时自动打开并在析构时关闭。静态成员函数使你可以在不打开文件的情况下检查文件状态。

要了解关于使用 CFile 的更多信息，可参阅联机文档“Visual C++程序员指南”中的“MFC 中的文件”和“Microsoft Visual C++ 6.0 运行库参考”中的“文件处理”。

CFile

参数：

hFile　已打开的文件句柄。

lpszFileName　所需文件的路径字符串，此路径可为相对的也可为绝对的路径。

nOpenFlags　共享和访问模式，指定当打开文件时进行的动作，可以将以下所列用 OR(|)操作符联起来。至少应有一个访问权限和一个共享选项，modeCreate 和 modeNoInherit 是可选的。值如下所示：

· CFile::modeCreate　调用构造函数构造一个新文件，如果文件已存在，则长度变成 0。

· CFile::modeNoTruncate　此值与 modeCreate 组合使用。如果所创建的文件已存在则其长度不变为 0。因而此文件被打开，或者作为一个新文件或者作为一个已存在的文件。

这将是很有用的，例如当打开一个可能存在也可能不存在的文件时。这个选项也可用于CStdioFile.

- CFile::modeRead 打开文件仅供读。
- CFile::modeReadWrite 打开文件供读写。
- CFile::modeWrite 打开文件仅供写。
- CFile::modeNoInherit 阻止文件被子进程继承。
- CFile::ShareDenyNone 不禁止其它进程读或写访问，打开文件。如果文件已被其它进程以兼容模式打开，则 Create 失败。
- CFile::ShareDenyRead 打开文件，禁止其它进程读此文件。如果文件已被其它进程以兼容模式打开，或被其它进程读，则 Create 失败。
- CFile::ShareDenyWrite 打开文件，禁止其它进程写此文件。如果文件已被其它进程以兼容模式打开，或被其它进程写，则 Create 失败。
- CFile::ShareExclusive 以独占模式打开文件，禁止其它进程对文件的读写。如果文件已经以其它模式打开读写（即使被当前进程），则构造失败。
- CFile::ShareCompat 此标志在 32 位 MFC 中无效。此标志在使用 CFile::Open 时映射为 CFile::ShareExclusive。
- CFile::typeText 对回车换行设置特殊进程（仅用于派生类）。
- CFile::typeBinary 设置二进制模式（仅用于派生类）。

说明：

缺省的构造函数不打开文件，而是将 m_hFile 设置为 CFile::hFileNull。因为此构造函数不产生异常，故不使用 TRY/CATCH 逻辑操作。使用 Open 成员函数，然后直接测试异常状态。有关异常处理的策略，可参阅联机文档“Visual C++程序员指南”中的“异常”。

带一个参数的构造函数构造一个 CFile 对象，对应于由 hFile 标识的操作系统文件。对访问模式或文件类型不作检查。当 CFile 对象被析构时，操作系统文件不会关闭，必须手工关闭它。

带两个参数的构造函数构造一个 CFile 对象，并打开给定路径的相对应的操作系统文件。构造函数将第一个构造函数和 Open 成员函数组合起来。当打开文件出错时产生一个异常。通常这说明这种错误不可解决，用户将被警告。

示例：

```
// example for CFile::CFile
```

```
char* pFileName = "test.dat";
TRY
{
  CFile f(pFileName, CFile::modeCreate | CFile::modeWrite);
}

CATCH(CFileException,e)
{
  #ifdef _DEBUG
     afxDump<<"File could not be opened"<<e->m_cause<<"\n";
  #endif
}

END_CATCH
```

Close( )

说明：

关闭与该对象有关的文件并使之不可读写。当析构一个对象时未关闭文件，则构造函数关闭它。

如果用 new 操作放置 CFile 对象到堆顶，在关闭文件后必须删除它，Close 设置 m_hFile 为 CFile::hFileNull。

Duplicate

返回值：指向一个备份 CFile 对象。

说明：

为给定文件构造一个备份 CFile 对象，等价于 C 运行函数_dup。

Flush

说明：

强制滞留在缓冲区的数据写入文件。

使用 Flush 不能保证 CArchive 缓冲区的被清空。如果正在使用一个档案，可先调用 CArchive::Flush。

GetFileName

返回值：文件名。

说明：

调用此成员函数获取一个指定文件的文件名。例如当你调用 GetFileName 来产生一个关于文件 c:\windows\write\myfile.wri 的消息给用户时，文件名 myfile.wri 被返回。

要返回文件的完整路径及文件名，可调用 GetFilePath；要得到文件标题（myfile），可调用 GetTitle。

GetFilePath

返回值：指定文件的完整路径。

说明：

调用此成员函数获取指定文件的全路径。例如当你调用 GetFilePath 来产生一个关于文件 c:\windows\write\myfile.wri 的消息给用户时，文件路径 c:\windows\write\myfile.wri 被返回。

要返回文件名（myfile.wri）,可调用 GetFileName；要返回文件标题（myfile），可调用 GetFileTitle。

GetFileTitle

返回值：指定的文件的标题

说明：

调用此成员函数获取指定文件的标题，例如，当你调用 GetFileTitle 产生一个关于文件 c:\windows\write\myfile.wri 的消息给用户时，文件标题 myfile 被返回。

注意：在 Windows95 中，文件标题不包括扩展名，其解释见 Win32 文档中的 GetFileTitle。

要返回文件的全路径及文件名，可调用 GetFilePath；要返回文件名（myfile.wri），可调用 GetFileName。

GetLength

返回值：文件长度。

说明：获取文件的当前逻辑长度，以字节表示。

GetPosition

返回值：32 位双字文件指针。

说明：

获取文件指针的当前值，可用于以后的 Seek 调用。

示例：

```
// example for CFile::GetPosition

extern CFile cfile;

DWORD dwPosition = cfile.GetPosition( );
```

GetStatus

返回值：如果指定文件的状态信息成功获取，则为 TRUE，否则为 FALSE。

参数：

rStatus　用户提供的 CFileStatus 结构的参考，用来接收状态信息。CFileStatus 结构有以下字段：

- CTime m_ctime　文件创建的时间。
- CTime m_mtime　文件最后一次修改的时间。
- CTime m_atime　最后一次访问文件并读取的时间。
- LONG m_size　文件逻辑长度，以字节数表示，如同 DIR 命令报告的那样。
- BYTE m_attribute　文件属性字节。
- Char m_szFullName[_MAX_PATH]　Windows 字符集表示的全文件名。

lpszFileName　　Windows 字符集表示的文件路径，此路径可为绝对的或为相对的，但不包含网络名。

说明：

GetStatus 的虚拟版本获取与 CFile 对象有关的文件的状态，不把值插入到 m_szFullName 结构成员中。

静态版本获取文件状态并把文件名拷入 m_szFullName。此函数从文件目录入口获取文件状态而不打开文件，这对于测试已存在和访问权限十分有用。

m_attribute 是文件属性，MFC 提供一个 enum 类型的属性，这样就可以用符号指定属性：

```
enum Attribute
{
  normal    = 0x00,
  readOnly  = 0x01,
  hidden    = 0x02,
  system    = 0x04,
  volume    = 0x08,
  directory = 0x10,
  archive   = 0x20
};
```

示例：

```
// example for CFile::GetStatus
CFileStatus status;
Extern CFile cfile;
If(cfile.GetStatus(status)) // virtual member function
{
  #ifdef _DEBUG
     afxDump<<"File size ="<<status.m_size<<"\n";
  #endif
}
char* pFileName ="test.dat";
if(CFile::GetStatus(pFileName,status)) // status function
```

```
{
  #ifdef _DEBUG
    afxDump<<"Full File name ="<<status.m_szFullName <<"\n";
  #endif
}
```

LockRange

参数：

dwPos　封锁字符中从开始字节计算的偏移量。

dwCount　　封锁范围的字节数。

说明：

在一个打开文件中封锁一定范围内的字节。如果文件已封锁，则产生一个异常。在一个文件中封锁字节禁止其它进程输入到这些字节中。可以封锁一个以上的文件范围，但不可重叠。

当解锁一定范围时，可使用 UnlockRange 成员函数。其字节范围必须与以前封锁的范围相符合。LockRange 不连接相邻范围，如果两个相连区域被封锁，则应分别解锁。

注意：此函数不适用于CMemFile派生类。

示例：

```
// example for CFile::LockRange
extern DWORD dwPos;
extern DWORD dwCount;
extern CFile cfile;
cfile.LockRange(dwPos,dwCount);
```

Open

返回值：

如果成功打开，则返回非零值，否则为 0。pError 参数仅在返回 0 时才有意义。

参数：

lpszFileName　　待打开文件的路径，路径可为绝对、相对或网络名（UNC）。

nOpenFlags　一个定义了文件的共享和访问模式的 UINT。它指定了打开文件后的动作，可以用 OR（|）操作符将选项组合起来，至少应有一个访问权限和一个共享选项，modeCreate 和 modeNoInherit 模式是可选的。可参阅 CFile 构造函数中模式选项的列表。

pError　指向一个存在的文件异常对象，获取失败操作的状态。

说明：

Open 是设计来和缺省 CFile 构造函数共同使用的。这两个函数形成一个安全方式打开文件，此时失败是通常的、可预料的情况。

CFile 构造函数会在出错时产生一个异常，Open 在出错时返回 FALSE。Open 也可以初始化一个 CFileException 对象来描述一个错误，但是如果你不提供 pError 参数或将 NULL 传递给 pError，Open 将返回 FALSE 而不产生一个 CFileException。如果传递一个指针到一个存在的 CFileException，Open 会遇到错误，函数将用出错信息描述填充它。两种情况下 Open 都不产生异常。

下表描述了 Open 的可能结果：

pError	是否是遇到错误？	返回值	CFileException 内容
NULL	No	TRUE	n/a
ptr 指向 CFileException	No	TRUE	不变
NULL	Yes	FALSE	n/a
ptr 指向 CFileException	Yes	FALSE	被初始化，用来描述错误

示例：

```
// example for CFile::Open
CFile f;
CFileException e;
char* pFileName = "test.dat";
if(!f.Open(pFileName, CFile::modeCreate | CFile::modeWrite,&e))
{
  #ifdef _DEBUG
     afxDump<<"File could not be opened"<<e.m_cause<<"\n";
  #endif
}
```

Read

返回值：

传输到缓冲区的字节数。注意对所有 CFile 类，如果到达文件尾，则返回值可能比 nCount 小。

参数：

lpBuf　指向用户提供的缓冲区以接收从文件中读取的数据。

nCount　可以从文件中读出的字节数的最大值。对文本模式的文件，回车换行作为一个字符。

说明：从与 CFile 对象相关联的文件读数据到缓冲区。

示例：

```
// example for CFile::Read
extern CFile cfile;
char pbuf[100];
UINT nBytesRead = cfile.Read(pbuf,100);
```

ReadHuge

返回值：

传输到缓冲区的字节数。注意对所有 CFile 对象，如果达到文件末尾，则返回值可能要比 dwCount 小。

参数：

lpBuf　指向用户提供的缓冲区，以接收文件读入的数据。

dwCount　可从文件中读取的字节数的最大值。对文本模式的文件，回车换行作为一个字符。

说明：

从与 CFile 对象相关联的文件中读数据到缓冲区。

此函数与 Read 的区别在于：大于 64K-1 字节数的数据可以被读入 ReadHuge。此函数可被 CFile 派生的任何对象使用。

注意：ReadHuge 仅提供向后兼容，Win32 下的 ReadHuge 与 Read 有相同的语义。

Remove

参数：

lpszFileName　　表示所需文件的路径字符串。路径可为相对或绝对，但不可包含网络名。

说明：

此静态函数删除由路径指定的文件，但不可移去一个目录。

如果相关联的文件打开或文件不可移去，则函数产生一个异常，它等价于 DEL 命令。

示例：

```
// example for CFile::Remove
char* pFileName ="test.dat";
TRY
{
  CFile::Remove(pFileName);
}

CATCH(CFileException,e)
{
  #ifdef _DEBUG
     afxDump <<"File"<<pFileName<<"cannot be removed\n";
  #endif
}
END_CATCH
```

Rename

参数：

lpszOldName 原路径。

lpszNewName　　新路径。

说明：

此函数改名一个指定文件。目录不可改名，这个函数等价于 REN 命令。

示例：

```
// example for CFile::Rename
extern char* pOldName;
extern char* pNewName;
TRY
{
  CFile::Rename(pOldName,pNewName);
}

CATCH(CFileException,e)
{
  #ifdef _DEBUG
    afxDump << "File" << pOldName << "not found,cause =" << e->m_cause
<<"\n";
  #endif
}
END_CATCH
```

Seek

返回值：

如果要求的位置合法，则 Seek 返回从文件开始起的新字节偏移量。否则值未定义并产生 CFileException 异常。

参数：

lOff　　指针移动的字节数。

nFrom　　指针移动的模式，可为以下值之一：

· CFile::begin　从文件开始，把指针向后移动 lOff 字节。

· CFile::current　　从当前位置开始，把指针向后移动 lOff 字节。

· CFile::end　　从文件尾开始，把指针向前移动 lOff 字节。注意必须移动到存在的文件中，因而 lOff 应为负。如果为正值，则超出文件尾。

说明：

在以前打开的文件中重新定位一个指针。Seek 函数使我们可以随机访问一个文件的内容，这是通过指针移动一定量完成的，指针的移动可为绝对或相对。在查找时没有实际读文件。

当文件打开时，文件指针为偏移量 0 处，即文件开始处。

示例：

```
// example for CFile::Seek
extern CFile cfile;
LONG lOffset = 1000;
LONG lActual = cfile.Seek(lOffset, CFile::begin);
```

SeekToBegin

说明：

将文件指针指向文件开始处，等价于 Seek(0L, CFile::begin)。

示例：

```
// example for CFile::SeekToBegin
extern CFile cfile;
cfile.SeekToBegin( );
```

SeekToEnd

返回值：文件长度（字节数）。

说明：

将文件指针指向文件逻辑尾部，等价于 CFile::Seek(0L, CFile::End);

示例：

```
// example for CFile::SeekToEnd
extern CFile cfile;
DWORD dwActual = cfile.SeekToEnd( );
```

SetFilePath

参数：

lPszNewName　　指向指定新文件的路径字符串。

说明：

调用此成员函数指定文件路径。例如当构造一个 CFile 对象而文件的路径无效时，可调用此成员函数提供路径。

注意：SetFilePath 不打开或创建文件，只将 CFile 对象和路径名联系起来，然后即可使用。

SetLength

参数：

dwNewLen　以字节指定文件的长度。此值可比当前文件大或小，此文件将按需要被扩展或截取。

说明：

调用此成员函数改变文件长度。

注意：和CMemFile一起使用,此函数可能产生一个CMemeryException对象。

示例：

```
// example for CFile::SetLength
extern CFile cfile;
DWORD dwNewLength = 10000;
cfile.SetLength(dwNewLength);
```

SetStatus

参数：

lpszFileName　　所需文件的路径字符串。此路径可为相对的或为绝对的，但不可为网络名。

status　包含新状态信息的缓冲区。调用此成员函数用当前值预先填充 CFileStatus 结构，然后用要求的值改变它。如果值为 0，则相对应的状态项均不更新。可参阅 GetStatus 成员函数关于 CFileStatus 结构的描述。

说明：

设置与此文件位置有关的状态。

要设置时间，可改变 Status 的 m_mtime 字段。

注意，当调用 SetStatus 仅希望改变文件属性并且文件状态结构的成员 m_mtime 非零，属性也会受到影响（改变时间项也会对属性产生负作用）。如果希望仅改变文件属性，先设置文件状态结构的成员 m_mtime 为 0，然后调用 SetStatus。

示例：

```
// example for CFile::SetStatus
char* pFileName ="test.dat";
extern BYTE newAttribute;
CFile::Status Status;
CFile::GetStatus (pFileName,status);
status.m_attribute = newAttribute;
CFile::SetStatus(pFileName,status);
```

UnlockRange

参数：

dwPos　解锁范围开始处的字节偏移量。

dwCount　解锁范围的字节数。

说明：

在一个打开文件中解锁一定范围的字节。可参阅 LockRange 成员函数的描述了解细节。

注意：此函数对 CMemFile 派生类无效。

示例：

```
// example for CFile::UnlockRange
extern DWORD dwPos;
extern DWORD dwCount;
extern CFile cfile;
```

```
cfile.UnlockRange(dwPos,dwCount);
```

Write

参数：

lpBuf　指向用户提供的缓冲区，包含将写入文件中的数据。

nCount　从缓冲区内传输的字节数。对文本模式的文件，回车换行作为一个字符。

说明：

将数据从缓冲区写入与 CFile 对象相关联的文件。

Write 在几种情况下均产生异常，包括磁盘满的情况。

示例：

```
// example for CFile::Write
extern CFile cfile;
char pbuf[100];
cfile.Write(pbuf,100);
```

WriteHuge

参数：

lpBuf　指向用户提供的缓冲区，此缓冲区包含写入文件的数据。

dwCount　从缓冲区传输的字节数。对文本模式文件，回车换行作为一个字符。

说明：

将数据从缓冲区写入与 CFile 对象相关联的文件中。WriteHuge 在几种情况下产生异常，包括磁盘满的情况。

此函数与 Write 的区别在于：大于 64K-1 字节的数据可由 WriteHuge 写入。此函数可被任意 CFile 派生对象使用。

注意：WriteHuge 仅提供向后兼容，在 Win32 下 WriteHuge 与 Write 语义相同。

CFrameWnd

CFrameWnd 类提供 Windows 单文档界面重叠或弹出式框架窗口，以及管理窗口的成员。

要为应用构造有用的框架窗口，可从 CFrameWnd 中派生类。向派生类加入成员变量，以便存储指定给应用的数据。在派生类中实现消息处理成员函数和消息映射，指定当消息指向窗口时的动作。

有三种方法可以构造一个框架窗口：

- 用 Create 直接构造。
- 用 LoadFrame 直接构造。
- 用文档模板间接构造。

当调用 Create 和 LoadFrame 时，必须在堆中构造一个框架窗口（使用 C++ New 操作）。调用 Create 之前，也可以用 AfxRegisteerWndClass 全局函数登记一个窗口类来设置框架的图表和类风格。

使用 Create 成员函数传递框架构造参数作为立即参数。

LoadFrame 需要比 Create 少的参数，而从资源中获取大多数缺省值，例如框架标题，图标、加速表、菜单。为了能被 LoadFrame 访问，所有的资源必须有相同的 ID（例如，IDR_MAINFRAME）。

当一个 CFrameWnd 对象包含视图和文档时，它们由框架间接构造而不是直接由程序员直接构造。CDocTemplate 对象将框架构造、包含视图的构造、视图与文档相连接等混在一起。CDocTemplate 构造函数的参数指定了三种类（文档、框架、视图）的 CRuntimeClass。当用户指定新框架时（例如，使用 File New 命令或 MDI Windows New 命令），CRuntimeclass 对象被框架用于动态建立新的框架。

一个从 CFrameWnd 中派生出的框架窗口类必须由 DECLARE_DYNCREATE 声明以使上面的 RUNTIME_CLASS 机制正确运行。

CFrameWnd 包含缺省实现，用来执行 Windows 典型应用的主窗口的以下功能：

· 一个 CFrameWnd 框架窗口追踪一个与 Windows 活动窗口无关的窗口或当前输入焦点。当框架再次被激活时，活动视图通过 CView::OnActivateView 得到通知。

· 命令消息和一些常用框架通知消息由一个 CFrameWnd 框架发送到当前活动视图，包括 CWnd 中由 OnSetFocus，OnHScroll，OnVScroll 管理的消息。

· 当前活动窗口（或在 MDI 框架中当前活动的 MDI 子框架窗口）可以判断框架窗口的标题，这种特征可以由关闭框架窗口的 FMS_ADDTOTITLE 风格位使之无效。

· CFrameWnd 框架窗口管理控件条、视图、框架窗口用户区的其它子窗口的位置。一个框架窗口也更新工具条和其它控件条按钮。CFrameWnd 框架窗口也有缺省命令功能来打开和关闭工具条和状态条。

· CFrameWnd 框架窗口管理主菜单条。当一个弹出式菜单条显示时，框架

窗口使用 UPDATE_COMMAND_UI 机制来决定哪一菜单项目可用、无效、复选。当用户选择了菜单项目时，框架窗口用那个命令的消息字符串更新状态条。

· CFrameWnd 框架窗口有可选的加速器表格来自动译出键盘加速器。

· CFrameWnd 框架窗口有一个与 LoadFrame 一起设置的可选帮助 ID，用于上下文敏感帮助。一个框架窗口是一个象上下文敏感帮助（SHIFT+F1）和打印预览模式那样的半模态的乐队。

· CFrameWnd 框架窗口可以打开一个从文件管理器拖出的文件并放置在框架窗口上。如果一个文件扩展名被登记并与应用有关，框架窗口应对动态数据交换（DDE）开放请求作出应答，此请求在用户从文件管理器中打开一个文件或调用 ShellExecute 函数时发生。

· 如果框架窗口是应用的主窗口（即 CWinThread::m_pMainWnd），当用户关闭应用时，框架窗口让用户存储改变过的文档（用 OnClose 和 OnQueryEndSession）。

· 如果框架窗口是应用的主窗口，框架窗口是运行 WinHelp 的上下文。关闭框架窗口时，如果它是为应用提供帮助，则会关闭 WINHELP.EXE。

不要使用 C++ delete 操作析构一个框架窗口，而应该用 CWnd::DestroyWindow。CFrameWnd 实现的 PostNcDestroy 会在窗体被析构时删除 C++对象。当用户关闭框架窗口时，缺省 OnClose 处理会调用 DestroyWindow。

CFrameWnd

说明：构造一个 CFrameWnd 对象，但不构造可视框架窗口。调用 Create 构造可视窗口。

Create

返回值：如果初始化成功，则返回非零值，否则为 0。

参数：

lpszClassName　指向一个用于命名 Windows 类的以空终止的字符串。类名可以是任何以 AfxRegisterJWndClass 全局函数登记或 RegisterClassWindows 函数登记的名。如果为 NULL，使用预定义的缺省 CFrameWnd 属性。

lpszWindowName 指向代表窗口名的以空终止的字符串，用作标题条的文本。

dwStyle 指定窗口风格属性。如果想标题条自动显示窗口代表的文档名，则应包含 FWS_ADDTOTITLE 风格。

rect 定义窗口大小和位置。rectDefault 值使 Windows 为一个新窗口指定大小和位置。

pParentWnd 指定框架窗口的父窗口，对最高层框架窗口来说应为 NULL。

lpszMenuName　指定与窗口一起使用的菜单资源名。如果菜单有一个整数 ID 而不是字符串 ID，则使用 MAKEINTRESOURCE。此参数可为 NULL。

dwExStyle　指定窗口扩展的风格属性。

pContext　指向 CCreateContext 结构的指针。可为 NULL。

说明：

分两步构造一个 CFrameWnd 对象。首先调用构造函数来构造 CFrameWnd 对象，然后调用 Create 来构造 Windows 框架窗口并把它附加给 CFrameWnd 对象，Create 初始化窗口的类名、窗口名，登记它的风格、父窗口和相关菜单的缺省值。

使用 LoadFrame 而不用 Create 来从资源而不是指定其参数加载一个框架窗口。

CreateView

返回值：如果成功，则指向一个 CWnd 对象，否则为 NULL。

参数：

pContext　定义视图和文档风格。

nID　视图的 ID 值。

说明：

调用 CreateView 在一个框架内构造一个视图，此函数构造非 CView 派生的视图。调用后必须手工设置视图活动和可视，这些功能不是由 CreateView 自动执行的。

注意 MFC 高级概念的示例 COLLECT 用 CreateView 获取 Windows 95 中正确的 3D 效果。

GetActiveDocument

返回值：指向当前 CDocument，如果无当前文档，则返回 NULL。

说明：获取一个指向附加给当前视图的当前 CDocument。

GetActiveFrame

返回值

指向活动的 MDI 子窗口。如果程序是一个 SDI 应用或 MDI 框架窗口没有活动文档，则返回 this 指针。

说明：

调用此成员函数获取一个指向 MDI 框架窗口的活动多文档界面（MDI）子窗口的指针。

如果没有活动 MDI 子窗口或应用是一个单文档界面（SDI），则强制返回 this 指针。

GetActiveView

返回值：指向当前 CView 的指针，如果无当前视图，则返回 NULL。

说明：

调用此成员函数获取一个附加给框架窗口（CFrameWnd）的活动视图（如果有）指针。

此函数返回值在一个 MDI 主框架窗口（CMDIFrameWnd）时为 NULL。

在一个 MDI 应用中，MDI 主框架窗口没有与之相关的视图，相反，每个子视图（CMDIChildWnd）都有一个或几个与之相关的视图。MDI 应用中的活动视图可先查找活动 MDI 子窗口，再查找这些子窗口的活动视图。活动 MDI 子窗口可由调用函数 MDIGetActive 或 GetActiveFrame 找到

示例：

```
CMDIFrameWnd* pFrame = (CMDIFrameWnd* )AfxGeteApp( )->m_pMainWnd;
```

```
// Get the active MDI child window .
CMDIChildWnd* pChild = (CMDIChildWnd* )pFrame->GetActiveFrame( );

// or CMDIChildWnd* pChild = pFrame->MDIGetsActive( );

// Get the active view attached to the active MDI child windows.
CMyView * pView = (CMyView * ) pChild->GetActiveView( );
```

GetControlBar

返回值：与 ID 相关联的控件条指针。

参数：

nID　　控件条的 ID 值。

说明：

调用此成员函数获取对 ID 相关联的控件条的访问。

即使控件条浮动而不是框架的当前子窗口，此函数也可返回控件条。

GetMessageString

参数：

nID　　所需消息的资源 ID。

rMessage　　放置消息的 CString 对象。

说明：

覆盖此函数为命令 ID 提供定制的字符串。缺省实现仅加载资源文件中由 nID 指定的字符串。此函数在状态条中的字符串需更新时，由框架调用。

LoadFrame

参数：

nIDResource　与框架窗口有关的共享资源 ID。

dwDefaultStyle　框架的风格。如果想窗口的文档在标题条自动显示，则应包含FWS_ADDTOTITLE 风格。

pParentWnd　指向框架的父窗口的指针。

pContext　指向 CCreateContext 结构，可为 NULL。

说明：

分两步构造一个 CFrameWnd 对象。首先调用构造函数构造 CFrameWnd 对象，然后调用 LoadFrame 加载 Windows 框架窗口和相关资源，并将框架窗口附加给 CFrameWnd 对象。

当希望指定所有框架窗口的构造参数时，应该用 Create 成员函数而不用 LoadFrame。

当用文档模板对象构造一个框架窗口时，框架调用 LoadFrame。

框架用 pContext 参数指定要与框架窗口相连的对象，包含任何包容视图对象。可以在调用 LoadFrame 时设置 pContext 为 NULL。

OnCreateClient

返回值：如果成功，则返回非零值，否则为 0。

参数：

lpcs　指向 Windows CREATESTRUCT 结构对象的指针。

pContext　指向 CCreateContext 结构的指针。

说明：

执行 OnCreate 时由框架调用。不用调用此成员函数。

缺省实现是从 pContext 提供的信息构造一个 CView 对象（如果可能）。

覆盖此函数，可覆盖 CCreateContext 对象传递的参数或改变框架窗口的主客户区控件的构造方式。你可以覆盖的 CCreateContext 成员在 CCreateContext 类中有描述。

注意 不要改变 CREATESTRUCT 结构传递的值。它们仅用于传递信息。如果想覆盖初始窗口矩形，则可覆盖 CWnd 的成员函数 PreCreateWindows

SetActiveView

参数：

pViewNew　指向 CView 对象，如果无活动视图，则为 NULL。

bNotify　指定视图是否被通知激活。如果为 TRUE，则从新视图调用 OnActiveView；如果为 FALSE，则不会。

说明：

调用此成员函数设置活动视图。当用户在框架窗口中改变焦点时，框架会调用此成员函数。你可以强制调用 SetActiveView 改变指定视图的焦点。

SetActiveView

参数：

pViewNew　指向 CView 对象，如果无活动视图，则为 NULL。

bNotify　指定视图是否被通知激活。如果为 TRUE，则从新视图调用 OnActiveView；如果为 FALSE，则不会。

说明：

调用此成员函数设置活动视图。当用户在框架窗口中改变焦点时，框架会调用此成员函数。你可以强制调用 SetActiveView 改变指定视图的焦点。

ShowOwnedWindows

参数：

bShow　指定拥有的窗口显示或隐藏。

说明：调用此成员函数显示或隐藏所有 CFrameWnd 对象的后代窗口。

CListView

CListView 类简化了列表控件和 CListCtrl 的使用。CListCtrl 类按照 MFC 文档或视图结构封装了列表控件的功能。如果要了解这些结构的更多信息，请参阅 CView 类的概述及交叉参考。

CListView

说明：构造一个 CListView 对象。

GetListCtrl

返回值：

返回与视图相关联的列表控件的参考。

说明：

调用该成员函数来获取与视图相关联的列表控件的参考。

CObject

CObject 为 Microsoft 基础类库中主要的基类。它不仅用作库类，例如 CFile 和 CObList 的根，而且也是自己编写的类的根。CObject 类提供了基本的服务，其中包括：

- 支持串行化
- 运行类信息
- 对象诊断输出
- 与收集类兼容

注意：CObject 类不支持多继承。派生的类仅仅拥有一个 CObject 基类，并且 CObject 在等级体系中必须位于最左边。不过，它也允许在多继承分支的右边有其结构及非 CObject 派生类。

如果在类的执行与声明过程中使用了可选的宏，那么你将发现 CObject 派生的主要优点。

第一层次的宏，DECLARE_DYNAMIC 和 IMPLEMENT_DYNAMIC，允许在等级体系中运行时访问类名及位置。这样，将允许依次地进行有意义的诊断转储。

第二层次的宏，DECLARE_SERIAL 和 IMPLEMENT_SERIAL，包含了第一层次宏的所有功能，并且能够使对象到“档案”有效地串行化。

如果要了解有关通常条件下派生的 Microsoft 基础类和 C++类，以及如何使用 CObject 类的更多信息，请参阅联机文档“Visual C++程序员指南”中的“CObject 类主题”及“串行化（对象永久化）”。

AssertValid

说明：

AssertValid 通过检测内部状态，对对象进行有效性检查。在库的调试版本中，AssertValid 将产生断言，发出消息，列出断言失败的行数和文件名并终止程序。

当你编写了自己的类后，应当调用覆盖 AssertValid 函数，来为你自己和使用你的类的用户提供诊断服务。被覆盖的 AssertValid 在检测对应于派生类的数据成员之前，通常调用它的基类的 AssertValid 函数。

由于 AssertValid 为一个常量函数，那么在测试期间，不允许改变对象的状态。自己派生类的 AssertValid 函数不应当产生异常，而应当断言是否检测到无效的对象数据。

“有效性”的诊断依赖于对象的类。作为规则，函数应当执行“浅度检测”（“shallow check”），也就是说，如果一个对象包含了其它对象的指针，那么它应当进行检测，以查看指针是否为空，而不是对指针指向的对象进行有效性测试。

示例：请参阅 CObList::CObList，了解所有 CObject 示例中使用的 CAge 类。

```
// example for CObject::AssertValid
void CAge::AssertValid() const
{
   CObject::AssertValid();
   ASSERT( m_years > 0 );
   ASSERT( m_years < 105 );
}
```

CObject

参数：

objectSrc　　另一个 CObject 对象的参考。

说明：

上述函数为标准的 CObject 构造函数。通过派生类的构造函数自动调用该函数的缺省形式。

如果你的类可串行化（它引入了 IMPLEMENT_SERIAL 宏），那么在类的派生中必须使用缺省的构造函数（即没有参数的构造函数）。若不需要缺省的构造函数，请事先声明私有的和受保护的“空”构造函数。如果需要更多信息，请参阅联机文档“Visual C++程序员指南”中的“CObject 类主题”。

标准 C++缺省类拷贝构造函数进行的是成员对成员的拷贝。如果需要你自己类的拷贝构造函数但现在没有，那么私有的 CObject 拷贝构造函数的存在将保证编译器发出错误消息。因此，若你的类需要这种能力，就必须提供拷贝构造函数。

Dump

参数：

dc　用于转储的诊断性转储文本，通常该值为 afxDump。

说明：

将对象的内容转储到 CDumpContext 对象中。

当编写了自己的类后，应当覆盖Dump函数，来为你自己和使用你的类的用户提供诊断服务。被覆盖的Dump在打印对应于派生类的数据成员之前，通常调用它基类的Dump函数。如果你的类使用了IMPLEMENT_DYNAMIC和IMPLEMENT_SERIAL宏，那么CObject::Dump将打印类名。

注意：Dump 函数在输出末尾不应打印换行符。

Dump 调用仅仅在 Microsoft 基础类库的调试版本中有意义。对于有条件的编译，应当使用#ifdef _DEBUG/#endif 声明将函数调用、函数声明、及函数执行括起来。

既然 Dump 为一个常量函数，那么在转储期间不允许改变对象的状态。当插入了 CObject 指针，CDumpContext 插入(<<)操作符将调用 Dump 函数。

Dump 仅仅允许进行对象的“非周期”转储。例如，可以转储一系列对象，但假如其中一个对象就是列表本身，那么最终将导致栈溢出。

示例：请参阅 CObList::CObList，了解所有 CObject 示例中使用的 CAge 类。

```
// example for CObject::Dump
void CAge::Dump( CDumpContext &dc ) const
{
   CObject::Dump( dc );
   dc << "Age = " << m_years;
}
```

GetRuntimeClass

返回值：返回对应于该对象类的 CRuntimeClass 结构的指针，不会为 NULL。

说明：

对于每个 CObject 派生的类，都有一个 CRuntimeClass 结构。该结构成员如下：

· LPCSTR m_lpszClassName 包含 ASCII 类名的以空值为终止符的字符串。

· int m_nObjectSize 以字节数表示的对象的大小。如果对象有指向分配内存的数据成员，那么将不包括内存的大小。

· UINT m_wSchema 图解号（对于非串行化类，该值为-1）。请参阅 IMPLEMENT_SERIAL 宏以获取图解号的详细描述。

· CObject* (PASCAL* m_pfnCreateObject)( )

指向缺省构造函数的函数指针，用于创建你自己类的对象（仅仅当类支持动态创建时有效，否则将返回 NULL)。

· CRuntimeClass* (PASCAL* m_pfn_GetBaseClass)( )

如果应用动态链接到 MFC 的 AFXDLL 形式，那么将返回基类 CRuntimeClass 结构的函数指针。

· CRuntimeClass* m_pBaseClass 如果应用静态地链接到 MFC，那么将返回基类 CRuntimeClass 结构的指针。

专业和企业版中独有的功能 仅仅 Visual C++专业和企业版支持静态地链接到 MFC 中。如果需要更多信息，请参阅 Visual C++ Editions。

该函数在类的执行过程中需要使用 IMPLEMENT_DYNAMIC 或 IMPLEMENT_SERIAL 宏，否则，将得不到正确的结果。

示例：请参阅 CObList::CObList，了解所有 CObject 示例中使用的 CAge 类。

```
// example for CObject::GetRuntimeClass
CAge a(21);
CRuntimeClass* prt = a.GetRuntimeClass();
ASSERT( strcmp( prt->m_lpszClassName, "CAge" ) == 0 );
```

IsKindOf

返回值：如果对象对应于该类，则返回非零值，否则为 0。

参数：

pClass 指向与 CObject 派生类相关联的 CRuntimeClass 结构的指针。

说明：

检测 pClass 来查看：(1)对象是否属于指定的类，(2)对象是否属于指定类派生的类。该函数仅仅当类声明了 DECLARE_DYNAMIC 或 DECLARE_SERIAL 宏时有效。

不要过多地使用该函数，其原因在于它破坏了 C++多形性功能。相反，请使用虚函数。

示例：请参阅 CObList::CObList，了解所有 CObject 示例中使用的 CAge 类。

```
// example for CObject::IsKindOfCAge a(21);
// Must use IMPLEMENT_DYNAMIC or IMPLEMENT_SERIAL
ASSERT( a.IsKindOf( RUNTIME_CLASS( CAge ) ) );
ASSERT( a.IsKindOf( RUNTIME_CLASS( CObject ) ) );
```

IsSerializable

返回值：如果该对象可被串行化，则返回非零值，否则为 0。

说明：

测试该对象是否有资格被串行化。对于被串行化的类，声明中必须包含 DECLARE_SERIAL 宏，并且在执行过程中必须包含 IMPLEMENT_SERIAL 宏。注意：不可覆盖该函数。

示例：请参阅 CObList::CObList，了解所有 CObject 示例中使用的 CAge 类。

```
// example for CObject::IsSerializable
CAge a(21);
ASSERT( a.IsSerializable() );
```

Serialize

参数：

ar　被串行化的 CArchive 对象。

说明：

从档案文件中读取该对象或向档案文件中写入该对象。

必须为希望串行化的每个类覆盖 Serialize。被覆盖的 Serialize 首先必须调用基类的 Serialize 函数。

在类的声明中必须使用 DECLARE_SERIAL 宏，并且在类的执行过程中也必须使用 IMPLEMENT_SERIAL 宏。

使用 CArchive::IsLoading 或 CArchive::IsStoring 函数，用于决定是否装载或存储了档案文件。

通过 CArchive::ReadObject 和 CArchive::WriteObject 来调用 Serialize 函数。这些函数与 CArchive 插入操作符(<<)和抽出操作符(>>)相关联。

如果要了解有关串行化的例子，请参阅联机文档“Visual C++程序员指南”中的“串行化（对象永久化）”。

示例：请参阅 ObList::CObList，了解所有 CObject 示例中使用的 CAge 类。

```
// example for CObject::Serialize
void CAge::Serialize( CArchive& ar )
{
  CObject::Serialize( ar );
  if( ar.IsStoring() ) ar << m_years;
  else ar >> m_years;
}
```

CSocket

CSocket 类是从 CAsyncsocket 派生而来的，它继承了 CAsyncsocket 对 WindowsSockets API 的封装。与 CAsyncsocket 对象相比，CSocket 对象代表了 WindowsSockets API 的更高一级的抽象化。CSocket 与类 CSocketFile 和 CArchive 一起来管理对数据的发送和接收。

一个 CSocket 对象也支持阻塞，这对于 CArchive 的同步操作来说是必要的。块操作函数，比如 Receive，Send，ReceiveFrom，SendTo，和 Accept（都是从 CAsyncsocket 继承来的），都不返回一个 CSocket 对象中的 WSAEWOULDBLOCK 错误。取而代之，这些函数等待，直到操作完成。另外，当这些函数中的某一个是阻塞的时，如果调用了 CancelBlockingCall，则原来的调用将因为 WSAEINTR 错误而终止。

要使用一个 CSocket 对象，调用构造函数，然后调用 Create 来创建基础 插槽句柄（插槽类型）。Create 的缺省参数创建一个插槽，但是如果你不是用一个 CArchive 对象来使用这个插槽，则你可以指定一个参数来创建一个数据包 插槽来代替，或者是结合一个指定的端口来创建一个服务器插槽。在客户方使用 Connect，则服务器方使用 Accept 来与一个客户插槽连接。然后再创建一个 CSocketFile 对象，并在 CSocketFile 的构造函数中将它连接到 CSocket 对象上。再接着，创建一个 CArchive 对象用来发送数据，一个用来接收数据（如果需要），然后在 CArchive 构造函数中将它们与 CSocketFile 对象连接。当通讯完成后，销毁 CArchive，CSocketFile，CSocket 对象。

更多的信息，参见 Win32 SDK 文档中的“Windows 插槽 2 概述”和“Windows 插槽设计思考

Attach

返回值：如果函数成功则返回非零值。

参数：

hsocket　包含一个插槽句柄。

说明：

此成员函数用来将一个 hsocket 句柄与一个 CSocket 对象连接。这个插槽句柄被保存在对象的 m_socket 数据成员中。

更多的信息，参见 Win32 SDK 文档中的“Windows Sockets 设计思考”。

CancelBlockingCall

说明：

此成员函数用来取消一个当前在进行中的阻塞调用。这个函数取消该插槽的任何未完的阻塞操作。原来的阻塞调用一有可能就终止，并给出 WSAEINTR 错误。

在一次阻塞的 Connect 操作中，Windows 插槽实现一有可能就终止这个阻塞的调用，但是这对于释放 socket 资源来说是不可能的，要直到连接完成（然后被重置）或时间到时，资源才被释放。只有在应用程序立即尝试打开一个新的插槽（如果没有插槽可利用），或连接到相同的同级者时，这一点才是需要注意的。

除了 Accept，取消任何操作都会使该插槽处于一种不确定的状态。如果一个应用程序取消了在一个插槽上的阻塞操作，则应用程序要能够在该插槽上显示，唯一能依赖的操作就是调用 Closet，虽然其它的操作可能会对某些 Windows 插槽起作用。如果你希望你的应用程序具有最大的可移植性，则你必须小心不要在一次取消后依赖于执行某些操作。

更多的信息，参见 Win32 SDK 文档中的“Windows Sockets 设计思考”。

Create

返回值：

如果函数成功则返回非零值；否则返回 0，并通过调用 GetLastError 可以获得特定的错误代码。

参数：

nSocketPort　一个要被此插槽使用的特别的端口，如果你希望 MFC 来选择一个端口，则这个值是 0。

nSocketType　是 SOCK_STREAM 或 SOCK_DGRAM。

lpszSocketAddress　　一个指向字符串的指针，该字符串包含了被连接插槽的网络地址。一个带点的数字，如“128.56.22.8”。

说明：

在构造一个插槽对象之后，调用 Create 成员函数来创建 Windows 插槽并连接它。然后 Create 调用 Bind 来将此插槽与指定的地址结合。下面就是所支持的插槽类型：

· SOCK_STREAM 提供连续的，可靠的，两种方法（two-way）的，基于连接的字节流。使用 Internet 地址家族的传输控制协议（TCP）。

· SOCK_DGRAM 支持无连接的数据包，它有具有固定（通常较小）最大长度的不可靠的缓冲。使用 Internet 地址家族的 User Datagram Protocol（UDP）。要使用这个选项，你必须不用一个 CArchive 对象来使用插槽。

注意：

Accept 成员函数将一个新的，空的 CSocket 对象的引用作为它的参数。你必须在调用 Accept 之前构造这个对象。记住，如果这个插槽对象超出了范围，则连接关闭。不要对这个新的 sock 对象调用 Create 函数。

有关流和数据包 插槽的更多信息，参见 Win32 SDK 文档中的“Windows Sockets 设计思考”。

CSocket

说明：

此成员函数用来构造一个 CSocket 对象。在构造之后，你必须调用 Create 成员函数。

更多的信息，参见 Win32 SDK 文档中的“Windows Sockets 设计思考”。

FromHandle

返回值：

返回一个指向 CSocket 对象的指针。如果没有与 hSocket 连接的 CSocket 对象，则返回 NULL。

参数：

hSocket 包含一个插槽句柄。

说明：

此成员函数返回一个指向 CSocket 对象的指针。当给予一个插槽句柄，如果没有一个 CSocket 对象连接到这个句柄，则此成员函数返回 NULL，并且不会创建一个临时对象。

更多的信息，参见 Win32 SDK 文档中的“Windows Sockets 设计思考”。

IsBlocking

返回值：

如果该插槽是阻塞的，则返回非零值。否则返回 0。

说明：

此成员函数用来确定一个阻塞的调用是否正在进行中。

更多的信息，参见 Win32 SDK 文档中的“Windows Sockets 设计思考”。

OnMessagePending

返回值：

如果消息被处理了则返回非零值；否则返回 0。

说明：

重载这个函数来查找来自 Windows 的特别消息，并在你的插槽中对它们作出响应。这是一个高级的可重载函数。

当插槽在转发 Windows 消息时，框架调用 OnMessagePending 来给你一个机会去处理你的应用程序感兴趣的消息。

更多的信息，参见 Win32 SDK 文档中的“Windows Sockets 设计思考”。

CSocketFile

一个 CSocketFile 对象是一个用来通过 Windows Sockets 在网络中发送和接收数据的 CFile 对象。为了这个目的，你可以将 CSocketFile 对象与一个 CSocket 对象连接。你也可以——并且通常是这样做——将 CSocketFile 对象与一个 CArchive 对象连接，以使用 MFC 系列来简化发送和接收数据。

对于连续（发送）的数据，你可以把它们插入到档案中，该档案调用 CSocketFile 成员函数来把数据写到 CSocket 对象中。对于不连续（接收）的数据，你从档案中提取它们。这导致该档案调用 CSocketFile 成员函数来从 CSocket 对象中读取数据。

提示：

除了象这儿所描述的一样使用 CSocketFile，你也可以将它作为一个标准文件对象，就象你使用它的基类 CFile 一样。你可以将 CSocketFile 与其它任何基于档案的 MFC 系列函数一起使用。因为 CSocketFile 并不支持 CFile 的所有性能，某些缺省的 MFC 串行化函数与 CSocketFile 是不兼容的。这一点对于 CEditView 类来说完全正确。你不要尝试使用 CEditView::SerializeRaw 并通过一个与 CSocketFile 对象连接的 CArchive 对象来串行化 CEditView 数据；而应当使用 CEditView::Serialize。SerializeRaw 函数希望文件对象具有一些 CSocketFile 所不拥有的函数，比如说 Seek。

CSocketFile

参数：

pSocket　连接到 CSocketFile 对象的插槽。

bArchiveCompatible　　指示该文件对象是否与一个 CArchive 对象一起使用。只有当你希望在单机方式下来使用这个 CSocketFile 对象时，才传递 FALSE。这种方式就象单机的 CFile 对象，具有一定的限制。这个标志将改变与 CSocketFile 对象连接的 CArchive 对象管理读缓冲区的方式。

说明：

此成员函数用来构造一个 CSocketFile 对象。当此对象超出范围或被删除时，它的析构函数将使它自己从插槽对象上分离。

注意：一个 CSocketFile 对象也可以在没有 CArchive 对象的情况下作为一个（受限制）的文件来使用。缺省的，CSocketFile 构造函数的 bArchiveCompatible 参数是 TRUE。这表明此文件对象是与一个档案一起使用的。要在没有档案的情况下使用这个文件对象，给 bArchiveCompatible 参数传递 FALSE。

在“档案兼容”模式下，一个 CSocketFile 对象可以提供更好的表现，并减少出现“死锁”的危险。当两个发送和接收插槽相互等待时，或是同时等待一个公用的资源时，就会发生死锁。如果与 CSocketFile 对象一起工作的 CArchive 对象按它与 CFile 对象一起工作的方式来工作时，这种情况就会发生。在与 CFile 一起工作时，档案可以假定如果它接收到的字节少于它所请求的，则文件已经到达文件结尾了。

但是，在使用 CSocketFile 工作时，数据是基于消息的；缓冲区可以保存多条消息，因此收到比所请求的要少的字节数并不表示到达了文件结尾。在这种情况下，应用程序不会象使用 CFile 时那样阻塞，它可以继续从缓冲区中读取数据，直到缓冲区为空。在这种情况下，CArchive::IsBufferEmpty 用于监视档案的缓冲区的状态是很有用的。

CString

CString 没有基类。

一个 CString 对象由可变长度的一队字符组成。CString 使用类似于 Basic 的语法提供函数和操作符。连接和比较操作符以及简化的内存管理使 CString 对象比普通字符串数组容易使用。

CString 是基于 TCHAR 数据类型的对象。如果在你的程序中定义了符号_UNICODE，则 TCHAR 被定义为类型 wchar_t，即 16 位字符类型；否则，TCHAR 被定义为 char，即 8 位字符类型。在 UNICODE 方式下，CString 对象由 16 位字符组成。非 UNICODE 方式下，CString 对象由 8 位字符组成。

当不使用_UNICODE 时，CString 是多字节字符集（MBCS，也被认为是双字节字符集，DBCS）。注意，对于 MBCS 字符串，CString 仍然基于 8 位字符来计算，返回，以及处理字符串，并且你的应用程序必须自己解释 MBCS 的开始和结束字节。

CString 对象还具有下列特征：

· CString 可作为连接操作的结果而增大。

· CString 对象遵循“值语义”。应将 CString 看作是一个真实的字符串而不是指向字符串的指针。

· 你可以使用 CString 对象任意替换 const char*和 LPCTSTR 函数参数。

· 转换操作符使得直接访问该字符串的字符就像访问一个只读字符（C-风格的字符）数组一样。

提示：如果可能的话，应在框架中而不是堆中分配这个 CString 对象。这可以节省内存并简化参数的传递。

CString 允许两个具有相同值的字符串共享相同的缓冲空间，这有助于你节省内存空间。但是，如果你初始直接改变该缓冲的内容（不使用 MFC），则有可能在无意中改变了两个字符串。CString 提供了两个成员函数 CString::LockBuffer 和 CString::UnlockBuffer 来帮助你保护你的数据。当你调用 LockBuffer 时，你就创建了一个字符串的一个拷贝，然后将引用计数设置为-1，这就“加锁”了该缓冲区。当缓冲区被加锁时，就没有其它的字符串可以引用该字

符串中的数据，被加锁的字符串也不能引用其它字符串的数据。通过加锁该缓冲区内的字符串，就可以保证该字符串对数据的持续独占。当你使用完数据后，调用 UnlockBuffer 来将引用计数恢复为 1。

Compare

返回值：

如果字符串是一样的则返回非零值;如果 CString 对象小于 lpsz 则返回值<0，如果 CString 对象大于 lpsz 则返回值>0。

参数：

lpsz　　要用于比较的另一个字符串。

说明：

此成员函数通过使用通用文本函数_tcscmp 来比较这个 CString 对象和另一个字符串。此通用文本函数_tcscmp 是在 TCHAR.H 中定义的，根据在编译时设置的字符来与 strcmp，wcscmp，或_mbscmp 对应。这些函数的每一个都根据当前使用的代码页来进行一次区分大小写的比较，而且不会被现场影响。更多的信息，参见“Microsoft Visual C++ 6.0 运行库参考”中的 strcmp，wcscmp，_mbscmp。

示例：下面的例子说明了如何使用 CString::Compare。

```
// CString::Compare 示例：
CString s1( "abc" );
CString s2( "abd" );
ASSERT( s1.Compare( s2 ) == -1 ); //  与另一个 CString 比较。
ASSERT( s1.Compare( "abe" ) == -1 ); //  与 LPTSTR 字符串比较。
```

CompareNoCase

返回值：

如果字符串是一样的（不区分大小写）则返回非零值；如果 CString 对象小于 lpsz（不区分大小写）则返回值<0，如果 CString 对象大于 lpsz（不区分大小写）则返回值>0。

说明：

此成员函数通过使用通用文本函数_tcsicmp 来比较这个 CString 对象和另一个字符串。此通用文本函数_tcsicmp 是在 TCHAR.H 中定义的，根据在编译时设置的字符来与_stricmp，

_wcsicmp，或_mbsicmp 对应。这些函数的每一个都根据当前使用的代码页来进行一次区分大小写的比较，而且不会被现场影响。更多的信息，参见“Microsoft Visual C++ 6.0 运行库参考”中的_stricmp，_wcsicmp，_mbsicmp。

示例：下面的例子说明了如何使用 CString::CompareNoCase。

```
// CString::CompareNoCase 示例：
CString s1( "abc" );
CString s2( "ABD" );
ASSERT( s1.CompareNoCase( s2 ) == -1 ); // 与一个 CString 比较。
ASSERT( s1.Compare( "ABE" ) == -1 ); // 与 LPTSTR 字符串比较。
```

CString

参数：

stringSrc　一个已经存在的 CString 对象，它要被拷贝到此 CString 对象中。

ch　要被重复 nRepeat 次的单个字符。

nRepeat　要对 ch 重复的次数。

lpch　一个指向字符数组的指针，该字符数组的长度是 nLength，不包括结尾的空字符。

nLength　pch 中的字符的个数。

psz　一个要被拷贝到此 CString 对象中去的以空字符结尾的字符串。

lpsz　一个要被拷贝到此 CString 对象中去的以空字符结尾的字符串。

说明：

这些构造函数的每一个都用来以指定的数据初始化一个新的 CString 对象。

由于构造函数是将输入数据拷贝到新分配的存储区，所以你应该注意可能会导致的内存异常。注意，这些构造函数中的某些函数的作用相当于一个转换函数。这就允许你使用替换物，例如在一个需要 CString 对象的地方用一个 LPTSTR 来代替。

此构造函数的某几种形式具有特殊的目的：

· CString(LPCSTR lpsz)　从一个 ANSI 字符串构造一个 Unicode CString。你可以象下面的例子那样用这个函数来加载一个字符串资源。

· CString(LPCWSTR lpsz)　从一个 Unicode 字符串构造一个 CString。

· CString(const unsigned char* psz) 从一个指向 unsigned char 的指针构造一个 CString。

想了解更多的信息，请参阅“Visual C++ 程序员指南”随机文中的“String:CString Exception Cleanup ”一节。

示例：下面的例子说明了如何使用 String::CString。

```
// CString::CString 示例：
CString s1; // 空字符串
CString s2( "cat" ); // 从一个文字的 C 字符串
CString s3 = s2; // 拷贝构造函数
CString s4( s2 + " " + s3 ); // 从一个字符串表达式
CString s5( 'x' ); // s5 = "x"
CString s6( 'x', 6 ); // s6 = "xxxxxx"
CString s7((LPCSTR)ID_FILE_NEW); // s7 = "Create a new document"
CString city = "Philadelphia"; // 不是赋值操作符
```

Delete

返回值：返回已改变的字符串的长度。

参数：

nIndex 要删除的第一个字符的索引。

nCount 要删除的字符数。

说明：

此成员函数用来从一个字符串中从 nIndex 开始的地方删除一个或多个字符。如果 nCount 比此字符串还要长，则字符串的其余部分都将被删除。

示例：

```
// 下面的例子说明了如何使用 CString::Delete。
str2 = "Hockey is best!";
printf( "Before: %s\n, (LPCTSTR) str2 );
int n = str2.Delete(6, 3);
printf( "After: %s\n*, (LPCTSTR) str2 );
ASSERT( n == str2.Getlength ( ) );
// 这些代码产生下面的删除行：
```

```
Before: Hockey is best!
After: Hockey best!
```

Empty

说明：

此成员函数用来使 CString 对象成为一个空字符串，并释放相应的内存。

更多的信息，参见“Visual C++程序员指南”中的“字符串：CString 异常清除”。

示例：下面的例子说明了如何使用 CString::Empty。

```
// CString::Empty 示例：
CString s( "abc" );
s.Empty();
ASSERT( s.GetLength( ) == 0 );
```

Find

返回值：

返回此 CString 对象中与需要的子字符串或字符匹配的第一个字符的从零开始的索引；如果没有找到子字符串或字符则返回-1。

参数：

ch　要搜索的单个字符。

lpszSub　要搜索的子字符串。

nStart　字符串中开始搜索的字符的索引，如果是 0，则是从头开始搜索。如果 nStart 不是 0，则位于 nStart 处的字符不包括在搜索之内。

pstr　指向要搜索的字符串的指针。

说明：

此成员函数用来在此字符串中搜索子字符串的第一个匹配的字符。函数的重载可以接收单个字符（类似于运行时函数 strchr）和字符串（类似于 strstr）。

示例：

```
//下面演示第一个例子
```

```
// CString::Find( TCHAR ch )
CString s( "abcdef" );
ASSERT( s.Find( 'c' ) == 2 );
ASSERT( s.Find( "de" ) == 3 );

// 下面演示第二个例子
// CString::Find(TCHAR ch,int nStart)
CString str("The stars are aligned");
int n = str.Find('e',5);
ASSERT(n == 12)
```

FindOneOf

返回值：

返回此字符串中第一个在 lpszCharSet 中也包括的字符的从零开始的索引。

参数：

lpszCharSet　包含用于匹配字符的字符串。

说明：

此成员函数在此字符串中搜索与 lpszCharSet 中任意字符匹配的第一个字符。

示例：下面的例子说明了如何使用 CString::FindOneOf。

```
// CString::FindOneOf 示例：
CString s( "abcdef" );
ASSERT( s.FindOneOf( "xd" ) == 3 ); // 'd' is first match.
```

Format

参数：

lpszFormat　一个格式控制字符串。

nFormatID　包含格式控制字符串的字符串资源标识符。

说明：

此成员函数用来将格式化数据写入一个 CString 中，其方法就像 sprintf 函数向一个 C-风格的字符数组中格式化输出数据一样。这个成员函数在 CString 中格式化并存储一系列字符和值。

根据 lpszFormat 中指定的格式或 nFormatID 标识的字符串资源，函数中的每一个可选参数（如果有）都被转换并输出。

如果此字符串对象本身是作为 Format 的一个参数，则调用将失败。例如象下面的代码：

CString str = "Some Data";

str.Format("%s%d",str, 123); //注意：在参数列表中也使用了 str 将导致不可预期的结果。

当你传递一个字符串作为一个可选择的参数时，你必须显式地将它转换为 LPCTSTR。这个格式与 printf 函数中的格式参数具有相同的形式和函数。（有关格式和参数的描述，参见“Microsofr Visual C++ 6.0 运行库参考”中的 printf。）在被写的字符串结尾将添加一个空字符。

更多的信息，参见“Microsofr Visual C++ 6.0 运行时库参考”中的 sprintf。

GetAt

返回值：返回一个包含字符串中的指定位置的字符串的 TCHAR。

参数：

nIndex　CString 对象中的某个字符的从零开始的索引。nIndex 参数必须大于或等于 0，并小于 GetLength 函数的返回值。Microsoft 基础类库的测试版验证 nIndex 的边界；但是发行版不验证。

说明：

你可以把一个 CString 对象看作是一个字符数组。GetAt 成员函数返回一个由一个索引号指定的单个字符。重载的下标（[]）是 GetAt 常用的代用符。

示例：下面的例子说明了如何使用 CString::GetAt。

```
// CString::GetAt 示例：
CString s( "abcdef" );
ASSERT( s.GetAt(2) == 'c' );
```

GetBuffer

返回值：一个指向对象的（以空字符结尾的）字符缓冲区的 LPTSTR 指针。

参数：

nMinBufLength　字符缓冲区的以字符数表示的最小容量。这个值不包括一个结尾的空字符的空间。

说明：

此成员函数返回一个指向 CString 对象的内部字符缓冲区的指针。返回的 LPTSTR 不是 const，因此可以允许直接修改 CString 的内容。

如果你使用由 GetBuffer 返回的指针来改变字符串的内容，你必须在使用其它的 CString 成员函数之前调用 ReleaseBuffer 函数。

在调用 ReleaseBuffer 之后，由 GetBuffer 返回的地址也许就无效了，因为其它的 CString 操作可能会导致 CString 缓冲区被重新分配。如果你没有改变此 CString 的长度，则缓冲区不会被重新分配。

当此 CString 对象被销毁时，其缓冲区内存将被自动释放。

注意，如果你自己知道字符串的长度，则你不应该添加结尾的空字符。但是，当你用 ReleaseBuffer 来释放该缓冲区时，你必须指定最后的字符串长度。如果你添加了结尾的空字符，你应该给 ReleaseBuffer 的长度参数传递-1，ReleaseBuffer 将对该缓冲区执行 strlen 来确定它的长度。

示例：下面的例子说明了如何用 CString::GetBuffer。

```
// CString::GetBuffer 例子
CString s( "abcd" );
#ifdef _DEBUG
   afxDump << "CString s" << s << "\n";
#endif
LPTSTR p = s.GetBuffer( 10 );
strcpy( p, "Hello" ); //  直接访问 CString 对象。
s.ReleaseBuffer( );
#ifdef _DEBUG
   afxDump << "CString s " << s << "\n";
#endif
```

GetBufferSetLength

返回值：

返回一个指向对象的（以空字符结尾的）字符缓冲区的 LPTSTR 指针。

参数：

nNewLength　此 CString 字符缓冲区的以字符数表示的精确容量。

说明：

此成员函数返回一个指向 CString 对象的内部字符缓冲区的指针，如果需要的话，切断或增长缓冲区的长度以精确匹配由 nNewLength 指定的长度。返回的 LPTSTR 不是 const，因此可以允许直接修改 CString 的内容。

如果你使用由 GetBuffer 返回的指针来改变字符串的内容，你必须在使用其它的 CString 成员函数之前调用 ReleaseBuffer 函数。

在调用 ReleaseBuffer 之后，由 GetBuffer 返回的地址也许就无效了，因为其它的 CString 操作可能会导致 CString 缓冲区被重新分配。如果你没有改变此 CString 的长度，则缓冲区不会被重新分配。

当此 CString 对象被销毁时，其缓冲区内存将被自动释放。

注意，如果你自己知道字符串的长度，则你不应该添加结尾的空字符。但是，当你用 ReleaseBuffer 来释放该缓冲区时，你必须指定最后的字符串长度。如果你添加了结尾的空字符，你应该给 ReleaseBuffer 的长度参数传递-1，ReleaseBuffer 将对该缓冲区执行 strlen 来确定它的长度。

有关引用计数的更多信息，参见下面的文章：

- “Win 32 SDK 程序员参考”中的“通过引用计数来管理对象的生命周期”。
- “Win 32 SDK 程序员参考”中的“实现引用计数”。
- “Win 32 SDK 程序员参考”中的“管理引用计数的规则”。

GetLength

返回值：返回字符串中的字节计数。

说明：

此成员函数用来获取这个 CString 对象中的字节计数。这个计数不包括结尾的空字符。

对于多字节字符集（MBCS），GetLength 按每一个 8 位字符计数；即，在一个多字节字符中的开始和结尾字节被算作两个字节。

示例：下面的例子说明了如何使用 CString::GetLength。

```
// CString::GetLength 示例：
CString s( "abcdef" );
ASSERT( s.GetLength() == 6 );
```

Insert

返回值：返回被改变的字符串的长度。

参数：

nIndex　某个字符的索引，在这个字符的前面将要进行插入操作。

ch　要插入的字符。

pstr　一个指向要被插入的子字符串的指针。

说明：

此成员函数用来在字符串中的给定索引处插入一个单个字符或一个子字符串。

nIndex 参数标识了将要被移走以给字符或子串空间的第一个字符。如果 nIndex 是零，则插入将方式在这个字符串之前。如果 nIndex 大于字符串的长度，则函数将把当前的字符串与由 ch 或 pstr 指定的新字符串连接起来。

示例：

```
// 下面的例子说明了如何使用 CString::Insert。
CString str("HockeyBest");
int n = str.Insert( 6, "is" );
ASSERT( n == str.GetLength( ) );
printf( "1: %s\n", ( LPCTSTR ) str );
n = str.Insert( 6, ' ' );
ASSERT( n == str.GetLength( ) );
printf ( *2: %s\n*, (LPCTSTR) STR );
n = str.Insert(555,'1');
ASSERT( n == str.GetLength ( ) );
printf ( "3: %s\n", ( LPCTSTR ) str );
// 上面的代码将产生下面这些输出行：
1.Hockeyis Best
2.Hockey is Best
3.Hockey is Best!
```

IsEmpty

返回值：如果 CString 对象的长度为 0，则返回非零值；否则返回 0。

说明：此成员函数用来测试一个 CString 对象是否是空的。

示例：下面的例子说明了如何使用 CString::IsEmpty。

```
// CString::IsEmpty 示例：
CString s;
ASSERT( s.IsEmpty() );
```

Left

返回值：

返回一个包含指定范围字符的拷贝的 CString 对象。注意，返回的 CString 对象可能是空的。

参数：

nCount　要从这个 CString 对象中提取的字符数。

说明：

此成员函数用来从此 CString 对象中提取最前面的 nCount 个字符，并返回被提取的子字符串的拷贝。如果 nCount 超过了字符串的长度，则整个字符串都被提取。Left 类似与 Basic LEFT$函数（除了索引是从零开始的）。

对于多字节字符集（MBCS），nCount 指的是每 8 位字符的个数，即，在一个多字节字符中开始和结尾字节被算作两个字符。

示例：下面的例子说明了如何使用 CString::Left。

```
// CString::Left 示例：
CString s( _T("abcdef") );
ASSERT( s.Left(2) == _T("ab") );
```

LoadString

返回值：如果加载资源成功则返回非零值；否则返回 0。

参数：

nID　一个 Windows 字符串资源 ID。

说明：

此成员函数用来读取一个由 nID 标识的 Windows 字符串资源，并放入一个已有的 CString 对象中。

示例：下面的例子说明了如何使用 CString::LoadString。

```
// CString::LoadString 示例：
#define IDS_FILENOTFOUND 1
CString s;
if (! s.LoadString( IDS_FILENOTFOUND ))
{
  AfxMessageBox("Error Loading String: IDS_FILENOTFOUND");
  ...
}
```

LockBuffer

返回值：返回一个指向 CString 对象的指针，或者是一个以 NULL 结尾的字符串。

说明：

此成员函数用来加锁缓冲区内的一个字符串。

通过调用 LockBuffer，可以创建一个字符串的拷贝，然后将引用计数设置为-1。当引用计数被设置为-1 时，缓冲区中的字符串被认为是处于“加锁”状态。当该字符串处于加锁状态时，字符串被从两个方面得到保护：

· 没有其它的字符串能够获得对此加锁字符串中的数据的引用，即使是该字符串被赋予了加锁字符串。

· 加锁字符串将不能引用其它的字符串，即使另一个字符串被拷贝到该加锁字符串中。

通过加锁缓冲区中的字符串，可以保证该字符串对缓冲区的独占保持完整。

在你完成了对 LockBuffer 的使用之后，调用 UnlockBuffer 来将该引用计数恢复到 1。

有关引用计数的更多信息，参见下面的文章：

· “Win 32 SDK 程序员参考”中的“通过引用计数来管理对象的生命周期”。

· “Win 32 SDK 程序员参考”中的“实现引用计数”。

· “Win 32 SDK 程序员参考”中的“管理引用计数的规则”。

Mid

返回值：

返回一个包含指定范围字符的拷贝的 CString 对象。注意，这个返回的 CString 对象可能是空的。

参数：

nFirst　此 CString 对象中的要被提取的子串的第一个字符的从零开始的索引。

nCount　要从此 CString 对象中提取的字符数。如果没有提供这个参数，则字符串的其余部分都被提取。

说明：

此成员函数从此 CString 对象中提取一个长度为 nCount 个字符的子串，从 nFirst（从零开始的索引）指定的位置开始。此函数返回一个对所提取的字符串的拷贝。Mid 类似于 Basic MID$函数（除了索引是从零开始的）。

对于多字节字符集（MBCS），nCount 指的是每 8 位字符的数目；也就是说，在一个多字节字符中开始和结尾字节被算作两个字符。

示例：下面的例子说明了如果如何使用 CString::Mid。

```
// CString::Mid 示例：
CString s( _T("abcdef") );
ASSERT( s.Mid( 2, 3 ) == _T("cde") );
```

ReleaseBuffer

参数：

nNewLength　此字符串的以字符数表示的新长度，不计算结尾的空字符。如果这个字符串是以空字符结尾的，则参数的缺省值－1 将把 CString 的大小设置为字符串的当前长度。

说明：

使用 ReleaseBuffer 来结束对由 GetBuffer 分配的缓冲区的使用。如果你知道缓冲区中的字符串是以空字符结尾的，则可以省略 nNewLength 参数。如果字符串不是以空字符结尾的，则可以使用 nNewLength 指定字符串的长度。在调用 ReleaseBuffer 或其它 CString 操作之后，由 GetBuffer 返回的地址是无效的。

示例：下面的例子说明了如何使用 CString::ReleaseBuffer。

```
// CString::ReleaseBuffer 示例：
```

```
CString s;s = "abc";
LPTSTR p = s.GetBuffer( 1024 );
strcpy(p, "abc"); // 直接使用该缓冲区
ASSERT( s.GetLength() == 3 ); // 字符串长度 = 3
s.ReleaseBuffer(); // 释放多余的内存，现在 p 无效。
ASSERT( s.GetLength() == 3 ); // 长度仍然是 3
```

Remove

返回值：返回从字符串中移走的字符数。如果字符串没有改变则返回零。

参数：

ch　要从一个字符串中移走的字符。

说明：

此成员函数用来将 ch 实例从字符串中移走。与这个字符的比较是区分大小写的。

示例：// 从一个句子中移走小写字母'c':

```
CString str ("This is a test.");
int n = str.Remove( 't' );
ASSERT( n == 2 );
ASSERT( str == "This is a es." );
```

Replace

返回值：返回被替换的字符数。如果这个字符串没有改变则返回零。

参数：

chOld　要被 chNew 替换的字符。

chNew　要用来替换 chOld 的字符。

lpszOld　一个指向字符串的指针，该字符串包含了要被 lpszNew 替换的字符。

lpszNew　一个指向字符串的指针，该字符串包含了要用来替换 lpszOld 的字符。

说明：

此成员函数用一个字符替换另一个字符。函数的第一个原形在字符串中用 chNew 现场替换 chOld。函数的第二个原形用 lpszNew 指定的字符串替换 lpszOld 指定的子串。

在替换之后，该字符串有可能增长或缩短；那是因为 lpszNew 和 lpszOld 的长度不需要是相等的。两种版本形式都进行区分大小写的匹配。

示例：

```
// 第一个例子，old 和 new 具有相同的长度。
CString strZap( "C--" );
int n = strZap.Replace('-', '+' );
ASSERT( n == 2 );
ASSERT(strZap == "C++" );
// 第二个例子，old 和 new 具有不同的长度。
CString strBang("Everybody likes ice hockey");
n = strBang.Replace( "hockey", "golf" );
ASSERT( n == 1 )
n = strBang.Replace ( "likes" , "plays" );
ASSERT( n == 1 )
n = strBang.Replace( "ice", NULL );
ASSERT( n == 1 )
ASSERT( strBang == "Everybody plays golf" )
// 注意，现在在你的句子中有了一个额外的空格。
// 要移走这个额外的空格，可以将它包括在要被替换的字符串中，例如，“ice”。
```

ReverseFind

返回值：

返回此 CString 对象中与要求的字符匹配的最后一个字符的索引；如果没有找到需要的字符则返回-1。

参数：

ch　要搜索的字符。

说明：

此成员函数在此 CString 对象中搜索与一个子串匹配的最后一个字符。此函数类似于运行时函数 strrchr。

示例：

```
// CString::ReverseFind 示例：
```

```
CString s( "abcabc" );
ASSERT( s.ReverseFind( 'b' ) == 4 );
```

Right

返回值：

返回一个包含指定字符拷贝的 CString 对象。注意，这个返回的 CString 对象可能是空的。

参数：

nCount　要从这个 CString 对象中提取的字符数目。

说明：

此成员函数用来从此 CString 对象中提取最后（最右边）的 nCount 个字符，并返回此提取字符串的一个拷贝。如果 nCount 超过了字符串的长度，则提取整个字符串。Right 类似于 Basic 的 RIGHT$函数（除了索引是从零开始的）。

对于多字节字符集（MBCS），nCount 指的是每 8 位字符的数目；也就是说，在一个多字节字符中开始和结尾字节被算作两个字符。

示例：下面的例子说明了如何使用 CString::Right。

```
// CString::Right 示例：
CString s( _T("abcdef") );
ASSERT( s.Right(2) == _T("ef") );
```

SetAt

参数：

nIndexCString　对象中的某个字符的从零开始的索引。nIndex 参数必须大于或等于 0，小于由 Getlength 的返回的值。Microsoft 基础类库的测试版本将检验 nIndex 的边界，而 Release 版本则不检验。

ch　要插入的字符。

说明：

你可以将一个 CString 对象看作是一个字符数组。SetAt 成员函数重写由一个索引值指定的单个字符。如果索引超出了已有字符串的边界，SetAt 不会扩大这个字符串。

TrimLeft

参数：

chTarget　　要被整理的目标字符。

lpszTargets　　指向一个字符串的指针，该字符串包含了要被整理的目标字符。

说明：

这个成员函数的没有参数的版本用来将字符串最前面的空格修整掉。当在没有参数的情况下调用时，TrimLeft 删除换行符，空格和 tab 字符。

这个成员函数的需要参数的版本用来将一个特定的字符或一群特定的字符从字符串的开始处删除。

参见 TrimRight 可以获得一个代码例子。更多的信息，参见“Visual C++程序员指南”中的“字符串主题”。

TrimRight

参数：

chTarget　　要被整理的目标字符。

lpszTargets　　一个指向字符串的指针，该字符串中包含了要被整理的目标字符。

说明：

这个成员函数的没有参数的版本用来将字符串最后面的空格修整掉。当在没有参数的情况下调用时，TrimRight 从字符串中删除换行符，空格和 tab 字符。

这个成员函数的需要参数的版本用来将一个特定的字符或一群特定的字符从字符串的结尾处删除。

示例：

```
CString strBefore;
CString strAfter;
  strBefore ="Hockey is Best!!!!" ;
  strAfter = strBefore;
  str.TrimRight('!' );
  printf ("Before: \"%s\"\n",(LPCTSTR) strBefore );
  printf ("After: \"%s\"\n",(LPCTSTR) strBefore );
```

```
strBefore ="Hockey is Best?!?!?!?!" ;
strAfter = strBefore;
str.TrimRight( *?!* );
printf ("Before: \"%s\"\n",(LPCTSTR) strBefore );
printf ( "After: \"%s\"\n",(LPCTSTR) strAfter );
```

在上面的第一个例子中，字符串“Hockey is Best!!!!”变成了“Hockey is Best”。

在上面的第二个例子中，字符串“Hockey is Best?!?!?!?!”变成了“Hockey isBest”。

更多的信息，参见“Visual C++程序员指南”中的“字符串主题”。

UnlockBuffer

说明：

此成员函数用来解锁先前通过调用 LockBuffer 加锁的的缓冲区。UnlockBuffer 将引用计数恢复为 1。

CString 析构函数使用 UnlockBuffer 来保证当析构函数被调用时缓冲区不会被加锁。

CStringArray

CStringArray 类支持 CString 对象数组。

CStringArray 的成员函数类似于 CObArray 类的成员函数。由于具有这些相似性，你可以参考关于 CObArray 的参考文件来获取 CStringArray 成员函数的详细说明。如果在说明中看到某一函数的返回值是一个指向 CObject 对象的指针，则可以用一个 CString（而不是一个 CString 指针）来代替它。如果看到某一函数的参数是一个指向 CObject 的指针，则可以用 LPCTSTR 来代替它。

例如，可以将

CObject* CObArray::GetAt(int <nIndex>) const;

转换为

CString CStringArray::GetAt(int <nIndex>) const;

和将

void SetAt(int <nIndex>, CObject* <newElement>)

转换为

void SetAt(int <nIndex>, LPCTSTR <newElement>);

CStringArray 与 IMPLEMENT_SERIAL 宏联合起来支持其元素的连续和转储。如果一个 CString 对象数组被用一个重载的插入操作符或 Serialize 成员函数保存到一个存档中，则它的每一个元素都按顺序连续。

注意：

在使用一个数组之前，使用 SetSize 来建立它的大小并给它分配内存。如果你不使用 SetSize，则向数组中添加元素将导致数组被频繁地拷贝和分配内存。频繁分配内存和拷贝会导致效率低和内存零碎。

如果你需要数组中个别字符串元素的转储，则应该将转储环境的深度设置为 1 或更大。当一个 CString 数组被删除时，或当其中的个别元素被删除时，字符串内存被根据需要释放。

CStringList

CStringList 类支持 CString 对象的列表。所有的比较都是通过值比较来完成的，这意味着不是比较字符串的地址而是比较字符串中的字符。

CStringList 的成员函数类似于类 CObList 类的成员函数。由于具有这些相似性，你可以参考关于 CObList 的参考文件来获取 CStringList 成员函数的详细说明。如果在说明中看到某一函数的返回值是一个指向 CObject 对象的指针，则可以用一个 CString（而不是一个 CString 指针）来代替它。如果看到某一函数的参数是一个指向 CObject 的指针，则可以用 LPCTSTR 来代替它。

例如，可以将

CObject*& CObList::GetHead() const;

转换为

CString& CStringList::GetHead() const;

和将

POSITION AddHead(CObject* <newElement>);

转换为

POSITION AddHead(LPCTSTR <newElement>);

CStringList 与 IMPLEMENT_SERIAL 宏联合起来支持其元素的连续和转储。如果一个 CString 对象列表被用一个重载的插入操作符或 Serialize 成员函数保存到一个存档中，则它的每一个元素都按顺序连续。

如果你需要数组中个别字符串元素的转储，则应该将转储环境的深度设置为 1 或更大。

当一个 CStringList 对象被删除时，或当它的元素被删除时，则相应的 CString 对象被删除。

CTime

CTime 没有基类。

一个 CTime 对象代表一个绝对的时间和日期。CTime 类引入了 ANSI time_t 数据类型以及其相关的运行时函数，其中包括向或自一个 Gregorian 日期和 24-小时时间的转换功能。

CTime 值是基于世界标准时间（UCT）的，UCT 时间等于格林威治（Greenwich）时间（GMT）。本地时区是由 TZ 环境变量控制的。

当创建一个 CTime 时，将 nDST 参数设置为 0 表示有效的是标准时间，或将其设置为大于 0 表示有效的白天保留时间，将其设置为小于零的值表示由 C 运行时库代码来计算有效的是标准时间还是白天保留时间。如果没有设置这个参数，则它的值是不明确的，而从 mktime 返回的值是不可预知的。如果 timeptr 指向一个由先前调用 asctime，gmtime，或 localtime 返回的 tm 结构，则 tm_isdst 域包含了适当的值。

参见“Microsoft Visual C++ 6.0 参考库”的“Microsoft Visual C++ 6.0 运行时参考库”卷可以获得有关 time_t 数据类型和 CTime 使用的运行时函数的更多信息。与 CTime 类对应的类是 CTimeSpan 类，它代表了一段时间间隔——两个 CTime 对象之间的差值。

CTime 和 CTimeSpan 类都是不可派生的。因为没有虚函数，CTime 和 CTimeSpan 对象的大小都正好是四个字节。多数成员函数都是内联函数。

有关使用 CTime 的更多信息，参见“Visual C++程序员指南”中的文章“日期和时间”，以及“Microsoft Visual C++ 6.0 运行时库参考”中的“时间管理”。

CTime

参数：

timeSrc　表示一个已经存在的 CTime 对象。

time　　表示一个时间值。

nYear,nMonth, nDay,

nHour, nMin, nSec　　表示要拷贝到新的 CTime 对象中去的日期和时间值。

nDST　　表明有效的是否是 daylight saving time。可以是下列三个值中的某一个：

· nDST 被设置为 0 表示起作用的是标准时间。

· nDST 被设置为一个大于 0 的值　　表示起作用的是 daylight saving time。

· nDST 被设置为一个小于 0 的值　　表示要自动计算起作用的是标准时间还是 daylight saving time。

wDosDate, wDosTime　要被转换为一个日期/时间值并被拷贝到新的 CTime 对象中去的 MSDOS 日期和时间。

sysTime　要被转换为一个日期/时间值并被拷贝到新的 CTime 对象中去的 SYSTEMTIME 结构。

fileTime　要被转换为一个日期/时间值并被拷贝到新的 CTime 对象中去的 FILETIME 结构。

说明：

所有这些构造函数都创建一个新的 CTime 对象，并用指定的基于当前时区的绝对时间值来初始化这个对象。

下面是对每一个构造函数的描述：

· CTime( );　　构造一个没有初始化的 CTime 对象。这个构造函数允许你定义 CTime 对象数组。在使用此对象之前，你应该用有效的时间来初始化这个数组。

· CTime(const CTime&);　　从另一个 CTime 值构造一个 CTime 对象。

· CTime(time_t);　从一个 time_t 类型构造一个 CTime 对象。

· CTime(int, int, etc.);　从本地时间成分构造一个 CTime 对象，每一个时间成分被约束在下列范围：

构成　　范围

nYear　1970—3000

nMonth　1—12

nDay　　1—31

nHour　没有约束

nMin　　没有约束

nSec　　没有约束

*最大的日期限是 1/18/2038。更宽的日期范围，参见 COleDateTime。这个构造函数进行到 UTC 的相应转换。如果一个或多个年，月，或日成分超出了范围，则 Microsoft 基础类库的调试版会给出断言。在调用之前由你负责检验参数。

· CTime(WORD, WORD); 根据指定的 MSDOS 日期和时间构造一个 CTime 对象。

· CTime(const SYSTEMTIME&); 根据一个 SYSTEMTIME 结构构造一个 CTime 对象。

· CTime(const FILETIME&); 根据一个 FILETIME 结构构造一个 CTime 对象。你最好不要直接使用 CTime FILETIME 初始化。如果你使用一个 CTime 对象来操纵一个文件，则 CFile::GetStatus 通过使用一个用 FILETIME 结构初始化的 CTime 对象来为你获取该文件的时间标记。

有关 time_t 数据类型的更多信息，参见“Microsoft Visual C++ 6.0 运行库参考”中的时间函数。

更多的信息，参见“Win32 SDK 程序员参考”中的 SYSTEMTIME 和 FILETIME 结构。

更多的信息，参见 Win32 SDK 文档中的 MSDOS 日期和时间项。

示例：

```
// CTime::CTime 示例：
time_t osBinaryTime; // C 运行时时间(在<time.h>中定义)
time( &osBinaryTime ); // 从操作系统中获取当前时间
CTime time1; // 空的 CTime. (0 是不合法的时间值.)
CTime time2 = time1; // 拷贝构造函数
CTime time3( osBinaryTime ); // 根据 C 运行时时间构造
CTimeCTime time4( 1999, 3, 19, 22, 15, 0 ); // 10:15PM March 19, 1999
```

Format

返回值：返回一个包含了格式化时间的 CString。

参数：

pFormat 一个类似于 printf 格式化字符串的格式化字符串。前面有一个百分号（%）标记的格式化代码，被相应的 CTime 成分替换。格式化字符串中的其它字符被不作变动地拷贝到返回字符串中。参见运行时函数 strftime 可以获得详细的信息。Format 的格式化代码的值和意义如下所示：

· %D 此 CTime 中的总天数。

- %H 当前天的小时。
- %M 当前小时中的分钟。
- %S 当前分钟中的秒。
- %% 百分号。

nFormatID 用来表示这个格式的字符串的 ID。

说明：

此成员函数用来创建一个日期/时间值的格式化表达式。如果此 CTime 对象的状态是空，则返回值是一个空字符串。如果 CTime 对象的状态是无效，则返回值是一个空字符串。

示例：

```
// CTime::Format 和 CTime::FormatGmt 的示例：
CTime t( 1999, 3, 19, 22, 15, 0 ); // 10:15PM March 19, 1999
CString s = t.Format( "%A, %B %d, %Y" );
ASSERT( s == "Friday, March 19, 1999" );
```

FormatGmt

返回值：一个包含格式化时间的 CString。

参数：

pFormat 一个类似于 printf 格式化字符串的格式化字符串。细节参见运行时函数 strftime。

nFormatID 用来表示这个格式的字符串的 ID。

说明：

此成员函数用来生成一个对应于这个 CTime 对象的格式化字符串。这个时间值没有被转换，因此是反映 UTC 的。

GetAsSystemTime

返回值：如果成功则返回非零值；否则返回 0。

参数：

timeDest　　是一个对 SYSTEMTIME 结构的引用，该结构要被转换为一个日期/时间值并被拷贝到新的 CTime 对象中。

说明：

此成员函数用来将保存在 CTime 对象中的时间信息转换为一个 Win32 兼容的 SYSTEMTIME 结构。GetAsSystemTime 将结果时间保存在引用的 timeDest 对象中。由这个函数初始化的 SYSTEMTIME 数据结构把它的 wMilliseconds 成员设置为零。

GetCurrentTime

说明：此成员函数返回一个代表当前时间的 CTime 对象。

示例：

```
// CTime::GetCurrentTime 示例：
CTime t = CTime::GetCurrentTime();
```

GetDay

说明：

此成员函数根据本地时间返回范围在 1--31 之间的该月的天。这个函数调用 GetLocalTm，该函数使用了一个内部的、静态分配的缓冲区。调用其它的 CTime 成员函数会导致这个缓冲区中的数据被覆盖。

示例：

```
// CTime::GetDay, CTime::GetMonth, 和 CTime::GetYear 的示例：
CTime t( 1999, 3, 19, 22, 15, 0 ); // 10:15PM March 19, 1999
ASSERT( t.GetDay() == 19 );
ASSERT( t.GetMonth() == 3 );
ASSERT( t.GetYear() == 1999 );
```

GetDayOfWeek

说明：

此成员函数根据本地时间返回该星期的天；1 就是星期日，2 就是星期一，...，7 就是星期六。这个函数调用 GetLocalTm，该函数使用了一个内部的，静态分配的缓冲区。调用其它的 CTime 成员函数会导致这个缓冲区中的数据被覆盖。

GetGmtTm

返回值：

返回一个指向已填好的 struct tm 的指针，struct tm 是在包含文件 TIME.H 中定义的。该结构的成员和值如下所示：

- tm_sec 秒
- tm_min 分钟
- tm_hour 小时（0—23）
- tm_mday 月中的日（1—31）
- tm_mon 月（0－11；一月＝0）
- tm_year 年（实际的年减 1900 的差）
- tm_wday 星期几（1－7；星期日＝1）
- tm_yday 年中的日（0－365；一月一日＝0）
- tm_isdst 总是 0

注意：struct tm 中的年是在范围 70 至 138 之间的；在 CTime 中，年的范围是 1970 到 2038（包含 2038）。

参数：

ptm 指向一个将用来接收实际数据的缓冲区。如果这个指针是 NULL，则使用一个内部的，静态分配的缓冲区。调用其它的 CTime 成员函数将导致此缓冲区中的数据被改写。

说明：

此成员函数用来获取一个包含在此 CTime 对象中所含的时间分解 struct tm。GetGmtTm 返回 UTC。

这个函数调用 GetLocalTm，该函数使用了一个内部的、静态分配的缓冲区。调用其它的 CTime 成员函数会导致这个缓冲区中的数据被覆盖。示例参见 GetLocalTm 的示例。

GetHour

说明：

此成员函数根据本地时间返回范围在 0 至 23 之间的小时。这个函数调用 GetLocalTm，该函数使用了一个内部的、静态分配的缓冲区。调用其它的 CTime 成员函数会导致这个缓冲区中的数据被覆盖。

示例：

```
// CTime::GetHour, CTime::GetMinute, 和 CTime::GetSecond 的示例：
CTime t( 1999, 3, 19, 22, 15, 0 ); // 10:15PM March 19, 1999
ASSERT( t.GetSecond() == 0 );
ASSERT( t.GetMinute() == 15 );
ASSERT( t.GetHour() == 22 );
```

GetLocalTm

返回值：

返回一个指向已填好的 struct tm 的指针，struct tm 是在包含文件 TIME.H 中定义的。关于该结构的成员安排，参见 GetGmtTm。

参数：

ptm　　指向一个将用来接收实际数据的缓冲区。如果这个指针是 NULL，则使用一个内部的、静态分配的缓冲区。调用其它的 CTime 成员函数将导致此缓冲区中数据被改写。

说明：

此成员函数用来获取一个包含此 CTime 对象的分解后的各个成分的 struct tm。GetLocalTm 返回本地数据。

示例：

```
// CTime::GetLocalTm 示例：
CTime t( 1999, 3, 19, 22, 15, 0 ); // 10:15PM March 19, 1999
struct tm* osTime; // 指向一个包含数据元素的指针。
osTime = t.GetLocalTm( NULL );
ASSERT( osTime->tm_mon == 2 ); // Note zero-based month!
```

GetMinute

说明：

此成员函数根据本地时间返回范围在 0 至 59 之间的分钟。这个函数调用 GetLocalTm，该函数使用了一个内部的、静态分配的缓冲区。调用其它的 CTime 成员函数会导致这个缓冲区中的数据被覆盖。

GetMonth

说明：

此成员函数根据本地时间返回范围在 1 至 12 之间的月（1＝一月）。这个函数调用 GetLocalTm，该函数使用了一个内部的、静态分配的缓冲区。调用其它的 CTime 成员函数会导致这个缓冲区中的数据被覆盖。示例参见 GetDay 的示例。

GetSecond

说明：

此成员函数根据本地时间返回范围在 0 至 59 之间的秒。这个函数调用 GetLocalTm，该函数使用了一个内部的、静态分配的缓冲区。调用其它的 CTime 成员函数会导致这个缓冲区中的数据被覆盖。

GetTime

说明：

此成员函数返回一个给定 CTime 对象的 time_t 值。

示例：

```
// CTime::GetTime 示例：
CTime t( 1999, 3, 19, 22, 15, 0 ); // 10:15PM March 19, 1999
time_t osBinaryTime = t.GetTime(); // time_t 在<time.h>中定义
printf( "time_t = %ld\n", osBinaryTime );
```

GetYear

说明：

此成员函数根据本地时间返回范围在 1970 年 1 月 1 日至 2038 年 1 月 18 日之间的年。这个函数调用 GetLocalTm，该函数使用了一个内部的，静态分配的缓冲区。调用其它的 CTime 成员函数会导致这个缓冲区中的数据被覆盖。示例参见 GetDay 的示例。

CView

CView 类为用户定义的视图类提供了基本的功能。视图被连接到文档上，用作文档和用户之间的媒介：视图在屏幕或打印机上显示文档的图像，并将用户的输入解释为对文档的操作。

视图是框架窗口的子窗口。可能会有多个视图共用一个框架窗口，就像在分隔窗口中那样。视图类、框架窗口类和文档类之间的联系通过 CDocTemplate 类来建立。当用户打开一个新窗口或将现有窗口分隔为多个时，框架会创建一个新视图并将它连接到文档对象上。

一个视图只能被连接到一个文档，但是一个文档可以有多个视图与之相连接，

例如，在分隔窗口或多文档界面（MDI）应用程序的多重子窗口中显示的文档就是如此。对于给定的文档类型，应用程序可以支持不同类型的视图；例如，一个字处理程序可能既要提供文档的完整文本视图，又要提供只显示每节标题的大纲视图。这些不同的视图类型可以放在不同的框架窗口中，如果你使用的是分隔窗口，你也可以把它们放在同一框架窗口的不同板块中。

视图可以响应几种类型的输入，例如键盘输入，鼠标输入或拖放输入，还有菜单、工具条和滚动条产生的命令输入。视图接收框架窗口发送给它的命令，如果视图不接受一个给定的命令，它就将这个命令发送给相连接的文档。与所有的命令目标一样，视图类通过消息映射处理消息。

当文档的数据发生变化时，视图类响应这种变化，通常调用文档的 CDocument::UpdateAllViews 函数，通知所有其它的视图调用 OnUpdate 函数。OnUpdate 数的缺省实现使视图的整个用户区域无效。你可以重载这个函数，只使视图中与文档的变化部分相对应的区域无效。

如果要使用 CView，应当从它派生一个类，并实现它的 OnDraw 函数以在屏幕上显示。你还可以利用 OnDraw 函数来进行打印和打印预览。框架将处理打印循环以实现对文档的打印和打印预览。

通过 CWnd::OnHScroll 和 CWnd::OnVScroll 成员函数来处理滚动条消息。你可以在这些函数中实现对滚动消息的处理，你也可以利用 CView 的派生类 SCrollView 来处理滚动。

除了 CScrollView 以外，微软基础类库还提供了其它的 CView 派生类：

· CCtrlView 允许你在树，列表和带格式编辑控件中使用文档/视图结构。

· CDaoRecordView 在对话框控件中显示数据库记录的视图。

· CEditView 提供了一个简单的多行文本编辑器的视图。你可以将CEditView用作对话框中的一个控件，也可以将它用作文档的视图。

· CFormView 一种可以滚动的视图，其中包含了对话框控件，它建立在对话框模板资源的基础上。

· CListView 使你能够在列表控件中使用文档/视结构的视图。

· CRecordView 在对话框控件中显示数据库记录的视图。

· CRichEditView 使你能够在带格式编辑控件中使用文档/视图结构的视图。

· CScrollView 自动提供滚动支持的一种视图。

· CTreeView 使你能够在树控件中使用文档/视图结构的视图。

CView 类还有一种派生类，名为 CPreviewView，它被框架用于实现打印预览。这个类提供了对打印预览窗口特性的支持，例如工具条、单页或双页预览以及放大（被用来放大预览的图像）等。你没有必要调用或重载 CPreviewView 的任何成员函数，除非你想实现自己的打印预览界面（例如，如果你希望支持在预览模式下的编辑）。有关使用 CView 的更多信息参见“Visual C++程序员指南”中的“文档/视图结构主题”和“打印”。另外，有关自定义打印预览的更多细节可以参见“Visual C++文档”中的“技术注释 30”。

CView

说明：

构造一个 CView 对象。当生成一个新的框架窗口或者要分隔一个窗口时，框架将调用这个构造函数。重载 OnInitialUpdate 成员函数以在与文档连接之后初始化该视图。

GetDocument

返回值：

指向与视图相连接的文档的指针。如果这个视图没有与文档相连接，则返回 NULL。

说明：

调用这个函数以获得视图的文档指针。这使你能够调用文档的成员函数。

IsSelected

返回值：如果指定的文档项目被选中，则返回非零值；否则返回 0。

参数：

pDocItem　　指向要测试的文档项目的指针。

说明：

由框架调用，检查指定的文档项目是否被选中。这个函数的缺省实现返回 FALSE。如果你使用 CDocItem 对象实现选择，则重载这个函数。如果你的视图中包含 OLE 项，则必须重载这个函数。

OnActiveFrame

参数：

nstate 指定框架窗口是被激活还是反之。它可以是下列值之一：

· WA_INACTIVE　框架窗口将结束激活状态。

· WA_ACTIVE 框架窗口将通过不同于鼠标点击的方式而激活（例如，通过键盘接口来选择窗口）。

· WA_CLICKACTIVE　框架窗口将通过鼠标点击而激活。

pFrameWnd　指向要激活的框架窗口的指针。

说明：

当包含视图的框架窗口要被激活或结束激活状态时，框架就调用这个函数。如果你希望在与视图相关的框架窗口被激活或结束激活状态的时候进行特殊处理，那么应当重载这个成员函数。例如，当 CFormView 保存或恢复具有焦点的控件的时候，它就重载了这个函数。

OnActivateView

参数：

bActivate　　指明该视图是要被激活还是要结束激活状态。

pActivateView　　指向要激活的视图的指针。

pDeactiveView　　指向要结束激活状态的视图对象的指针。

说明：

当视图被激活或结束激活状态的时候，框架调用这个函数。这个函数的缺省实现将焦点设置到要激活的视图中。如果你希望在视图被激活或结束激活状态的时候进行特殊处理，那么应当重载这个函数。例如，如果你希望提供特别的视觉效果，使活动的视图与非活动的视图能有区别，你应当检查 bActivate 的值，并根据结果相应地更新视图的外观。

如果应用程序的主框架窗口被激活，而它的活动视图没有发生变化，那么 pActivateView 和 pDeactiveView 参数指向同一个视图——例如，焦点是从另一个应用程序传送到这个应用程序，而不是在应用程序内部从一个视图传送到另一个视图或是在 MDI 的子窗口之间传递。

当对一个视图调用 CFrameWnd::SetActiveView，而这个视图与 CFrameWnd::GetActive-View 返回的视图不一致的时候，这些参数也是不同的。这通常在分隔窗口中发生。

OnBeginPrinting

参数：

pDC　　指向打印机设备环境。

pInfo　　指向一个 CPrintInfo 结构，该结构描述了当前的打印作业。

说明：

框架在开始打印或打印预览作业之前，而在 OnPreparePrinting 被调用之后调用这个函数。这个函数的缺省实现不做任何操作。重载这个函数以分配打印所需的 GDI 资源，如画笔或字体。在 OnPrint 成员函数内部将这些 GDI 资源选入设备环境。如果你使用同一个视图对象来执行打印和打印预览，那么对每种显示所需的 GDI 资源使用不同的变量；这样使你能够在打印的时候更新屏幕。

你还可以使用这个函数来实现依赖于打印机设备环境的属性的初始化工作。例如，打印文档所需的页数依赖于用户在 Print 对话框中指定的设置（例如页长度等）。在这种情况下，你不能在 OnPreparePrinting 成员函数中指定文档的长度，而在通常情况下你可以这么做；你必须等待，直到已经根据对话框设置创建了打印机设备环境。OnBeginPrinting 是使你能够访问代表了打印机设备环境的 CDC 对象的第一个重载函数，因此你可以在这个函数内部设置文档的长度。注意如果这时没有直到文档的长度，在预览的过程中将不会显示滚动条。

OnDraw

参数：

pDC　　指向设备环境的指针，该环境被用于画出文档的图像。

说明：

框架调用这个函数以画出文档的图像。框架调用这个函数来实现屏幕显示，打印以及打印预览，在各种情况下，它传递不同的参数。没有缺省的实现。

为了显示你的文档的视图，必须重载这个函数。你可以通过 pDC 指向的 CDC 对象来调用图形设备接口（GDI）。你可以在显示之前将 GDI 资源，如画笔和字体等选入设备环境，在显示完成之后，再把它们选出去。通常你的绘图代码是与设备无关的，这意味着，它不需要有关显示图形的设备类型的信息。

如果要优化绘图操作，调用设备环境的 RectVisible 成员函数以确定一个给定的矩形是否需要重画。如果你需要区别普通的屏幕显示和打印，那么应调用设备环境的 IsPrinting 函数。

OnDrop

返回值：

如果成功地下放，则返回非零值，否则返回 0。

参数：

pDataObject　指向将要被放入下放目标的 COleDataObject 对象。

dropEffect　　用户要求的下放效果。

- DROPEFFECT_COPY　创建被下放的数据对象的一个拷贝。
- DROPEFFECT_MOVE　　将数据对象移动到当前鼠标位置。
- DROPEFFECT_LINK　在数据对象和它的服务器之间创建连接。

point　　相对于视图的客户区域的当前鼠标位置。

说明：

当用户在有效的下放目标上方放开一个数据对象时，框架调用这个函数。缺省的实现不做任何操作，并且返回 FALSE。

重载这个函数以实现视图的客户区内的 OLE 下放效果。可以通过 pDataObject 来检查数据对象的剪贴板数据格式和指定点的下放数据。

注意 如果在这个视类中存在 OnDropEx 的重载函数，则框架不会调用这个函数。

OnEndPrinting

参数：

pDC　指向打印机设备环境的指针。

pInfo　指向一个 CPrintInfo 结构，描述了当前的打印作业。

说明：

框架在打印或预览文档之后调用这个函数。这个函数的缺省实现不做任何操作。重载这个函数以释放你在 OnBeginPrinting 成员函数中分配的 GDI 资源。

OnEndPrintPreview

参数：

pDC　指向打印机设备环境的指针。

pInfo　指向一个 CPrintInfo 结构，描述了当前的打印作业。

point　指向在预览模式下最后显示的页面中的位置。

pView　指向用于预览的视图对象的指针。

说明：

当用户退出打印预览模式时，框架调用这个函数。这个函数的缺省实现调用 OnEndPrinting 成员函数并将主框架窗口恢复到打印预览开始之前的状态。如果在预览模式结束的时候需要进行一些特殊的处理，则应重载这个函数。例如，如果你想要在从预览模式切换到普通显示模式的时候保持用户在文档中的位置，你可以滚动到 pInfo 参数指向的 CPrintInfo 结构中 m_nCurPage 所指向的页面，point 参数所描述的位置。

在你的重载函数中总是要调用基类的 OnEndPrintPreview 函数，通常是在函数的末尾。

OnPrint

参数：

pDC　指向打印机设备环境的指针。

pInfo　　指向 CPrintInfo 结构的指针，该结构描述了当前打印作业。

说明：

框架调用这个函数以打印或预览文档的一页。对于要被打印的每一页，框架在调用 OnPrepareDC 成员函数之后立即调用这个函数。要被打印的页是在 pInfo 指向的 CPrintInfo 结构的 m_nCurPage 成员中指定的。缺省的实现调用 OnDraw 函数并将打印机设备环境传递给它。

如果具有以下原因，则应重载这个函数：

· 要允许打印多页文档。仅画出与当前要打印的页相对应的文档内容。如果你要 OnDraw 函数来绘图，你可以调整视图口的原点，这样只有文档的适当的部分才会被打印。

· 要使打印出来的图像与屏幕显示的图像不同（如果你的应用程序不是所见即所得的）。不应将打印机设备环境传递给 OnDraw 函数，而是使用设备环境，用没有在屏幕上显示的属性来画出图像。如果你在打印时需要一些 GDI 资源，而在屏幕显示中没有使用它们，则应在绘图之前将它们选入设备环境，随后把它们选出。这些 GDI 资源必须在 OnBeginPrinting 函数中分配，而在 OnEndPrinting 函数中释放。

· 要实现页眉和页脚。只要你限制 OnDraw 可以打印的区域，你还可以使用 OnDraw 函数来绘图。注意 pInfo 参数的 m_rectDraw 成员以逻辑单位描述了页面中可以打印的区域。在你重载的 OnPrint 中不要调用 OnPrepareDC，框架在调用 OnPrint 之前自动调用了 OnPrepareDC。

示例：

下面是重载的 OnPrint 函数的基本结构：

```
void CMyView::OnPrint( CDC *pDC, CPrintInfo *pInfo )
{
// Print headers and/or footers, if desired.
// Find portion of document corresponding to pInfo->m_nCurPage
. OnDraw( pDC );
}
```

OnScroll

返回值：

如果 bDoScroll 为 TRUE，并且视图确实被滚动了，则返回非零值；否则返回 0。

如果 bDoScroll 为 FALSE，则返回当 bDoScroll 为 TRUE 时应当返回的值，即使你没有作实际的滚动。

参数：

nScrollCode　滚动条代码，指明用户的滚动请求。这个参数由两个部分组成：低字节确定了水平滚动的类型，高字节确定了垂直滚动的类型：

- SB_BOTTOM　　滚动到底部
- SB_LINEDOWN　往下滚动一行
- SB_LINEUP　往上滚动一行
- SB_PAGEDOWN　往下滚动一页
- SB_PAGEUP　　往上滚动一页
- SB_THUMBTRACK　将滚动块拖至指定位置。当前的位置由 nPos 指定
- SB_TOP　　滚动到顶部

nPos　　如果滚动条代码为 SB_THUMBTRACK，则包含了当前的滚动块位置；否则没有被使用。根据初始的滚动范围值，nPos 有可能为负值，并且在必要时应当被强制转换为整数。

bDoScroll　　确定你是否需要实际完成指定的滚动动作。如果该值为 TRUE，则必须执行滚动操作；如果为 FALSE，则不应执行滚动操作。

说明：

框架调用这个函数以确定是否能够滚动。有一种情况是，如果视图接受到一个滚动条消息，则框架调用这个函数并将 bDoScroll 设为 TRUE。在这种情况下，你必须执行实际的滚动。其它的情况是，当某个 OLE 项被拖入下放对象的自动滚动区域，在发生实际的滚动之前，框架调用这个函数并将 bDoScroll 设为 FALSE。在这种情况下，你不应当执行实际的滚动。

OnScrollBy

返回值：

如果该视图能被滚动，则返回非零值；否则返回 0。

参数：

sizeScroll　　水平及垂直滚动的像素的数目。

bDoScroll　　确定视图的滚动是否发生。如果该值为 TRUE，则发生了滚动；如果为 FALSE，则没有发生滚动。

说明：

当用户将 OLE 项拖到当前视图的边界之外，或者操作水平、垂直滚动条，观察超出文档已显示的部分的内容时，框架就会调用这个函数。缺省的实现不做任何操作。在派生类中，这个函数检查该视图在用户请求的方向上是否可以滚动，然后在必要时更新新的区域。为了执行实际的滚动请求，CWnd::OnHScroll 和 CWnd::OnVScroll 自动调用这个函数。

OnUpdate

参数：

pSender　指向修改了文档的视图，如果需要更新所有的视图，则为 NULL。

lHint　包含了与修改有关的信息。

pHint　指向保存了与修改有关的信息的对象。

说明：

框架在视图的文档被修改后调用这个函数；这个函数被 CDocument::UpdateAllViews 调用的，使视图能够更新它的显示以反映那些变化。它也被 OnInitialUpdate 的缺省实现所调用。缺省的实现使整个客户区域无效，使得下一次接收到 WM_PAINT 消息时重画这些区域。如果你只想更新与文档的修改部分对应的区域，则应重载这个函数。为此你必须利用提示参数传递有关修改的信息。

如果要使用 lHint，则应定义特殊的提示值，通常是位掩码或枚举值，并且使文档传递其中的一个值。要使用 pHint，则应从 CObject 集成一个提示类，并使文档传递提示对象的指针。当你重载 OnUpdate 函数的时候，应使用 CObject::IsKindOf 成员函数来确定提示对象的运行时类型。

通常你不用在 OnUpdate 中直接执行任何绘图操作。相反，确定以设备坐标表示的矩形，描述要更新的区域，将这个矩形传递给 CWnd::InvalidateRect。这会使下一次接收到 WM_PAINT 消息时产生绘图操作。

如果 lHint 为 0，pHint 为 NULL，文档将发送一个一般的更新通知。如果一个视图接收到了一般的更新通知，或者它没有解码出提示，它将会使整个客户区无效。

CWinApp

CWinApp 是一个基类，你通过它来继承 Windows 应用程序对象。应用程序对象为你提供了初始化应用程序（以及它的每一个实例）和运行应用程序所需的成员函数。

每个使用微软基础类库的应用程序都只能包含一个从 CWinApp 继承的对象。当 Windows 调用 WinMain 函数时，这个对象在其它 C++全局对象都已经生成并且可用之后才被创建，WinMain 函数是由微软基础类库提供的。将你的 CWinApp 对象定义为全局的。

当你从 CWinApp 继承应用程序类的时候，应重载 InitInstance 成员函数以创建应用程序的主窗口对象。

除了 CWinApp 的成员函数以外，微软基础类库还提供了以下全局函数，用于访问你的 CWinApp 对象以及其它全局信息：

- AfxGetApp　获得指向CWinApp对象的指针。
- AfxGetInstanceHandle　获得当前应用程序实例的句柄。
- AfxGetResourceHandle　　获得应用程序资源的句柄。
- AfxGetAppName　获得一个字符串指针，其中包含了应用程序的名字。

另外，如果你拥有一个指向 CWinApp 对象的指针，可以通过 m_pszExename 来获得应用程序的名字。

AddDocTemplate

参数：

pTemplate　　指向要增加的 CDocTemplate 的指针。

说明：

调用这个成员函数，将文档模板加入应用程序维护的可用文档模板列表中。你可以在调用 RegisterShellFileTypes 之前加入所有的文档模板。

示例：

```
BOOL CMyApp::InitInstance()
{
  // ...
  // 下面的代码是在你选择 MDI（多文档界面）选项时 AppWizard 为你生成的。
  CMultiDocTemplate* pDocTemplate;
  pDocTemplate = new CMultiDocTemplate(
     IDR_MYTYPE,
     RUNTIME_CLASS(CMyDoc),
     RUNTIME_CLASS(CMDIChildWnd), // 标准的 MDI 子框架
     RUNTIME_CLASS(CMyView)
    );

  AddDocTemplate(pDocTemplate);
  // ...
}
```

AddToRecentFileList

参数：

lpszPathName　　文件的路径。

说明：

调用这个成员函数以把 lpszPathName 加入 MRU 文件列表。你必须在使用这个成员函数之前调用 LoadStdProfileSetting 成员函数以载入当前的 MRU 文件列表。

当框架打开一个文件或者执行 Save As 命令用新名字保存文件时，它就调用这个成员函数。

示例：

```
// 这个例子将路径名 c:\temp\test.doc 加入// File 菜单中的最近使用（MRU）文件列表
AfxGetApp()->AddToRecentFileList("c:\\temp\\test.doc");
```

CloseAllDocuments

参数：

bEndSession　指定 Windows 会话是否要结束。如果为 TRUE，则会话将结束；否则为 FALSE。

说明：

调用这个函数以在退出之前关闭所有打开的文档。在调用 CloseAllDocuments 之前调用 HideApplication。

CreatePrinterDC

返回值：如果成功地创建了打印机设备环境，则返回非零值；否则返回 0。

参数：

dc　对打印机设备环境的引用。

说明：

调用该成员函数从选定的打印机中创建打印机设备上下文（DC）。CreatePrinterDC 初始化通过引用传递过来的设备上下文，因此，你可以使用该设备上下文进行打印。

如果该函数调用成功，在你打印完毕之后，必须销毁该设备上下文。你可以让 CDC 对象的析构器去做这件事，也可以显式地调用 CDC=::DeleteDC。

CWinApp

参数：

lpszAppName　一个以 null 结尾的字符串，其中包含了 Windows 使用的应用程序的名字。如果没有提供这个参数，或者其值为 NULL，CWinApp 使用资源字符串 AFX_IDS_APP_TITLE 或可执行文件的文件名。

说明：

构造一个 CWinApp 对象并将 lpszAppName 传递给它，当作应用程序的名字保存。你必须创建一个 CWinApp 派生类的全局对象。在你的应用程序中只能有一个 CWinApp 对象。构造函数保存了执行 CWinApp 对象的指针，因此 WinMain 可以调用对象的成员函数以初始化并运行应用程序。

DoWaitCursor

参数：

nCode　如果这个参数为 1，则出现等待光标。如果为 0，则恢复等待光标，但不增加引用计数。如果为-1，则结束等待光标。

说明：

这个成员函数被框架调用，用以实现 CWaitCursor，CCmdTarget::BeginWaitCursor，CCmdTarget::EndWaitCursor 和 CCmdTarget::RestoreWaitCursor。缺省的实现提供一个沙漏光标。DoWaitCursor 维护一个引用计数。当它为正时，将显示沙漏光标。尽管通常你不用直接调用 DoWaitCursor 函数，你也可以重载这个成员函数以改变等待光标或者在显示等待光标时加入附加的处理。如果需要更简单有效的实现等待光标的方式，使用 CwaitCursor

Enable3dControls

注意：在这个部分同时描述了 Enable3dControls 和 Enable3dControlsStatic。

返回值

如果成功地载入了 CTL3D32.DLL，则为 TRUE；否则为 FALSE。

如果操作系统支持控件的三维外观，则这个函数将返回 FALSE。

说明：

在你重载的 InitInstance 成员函数内调用这些成员函数以使对话框和窗口的控件能够具有三维外观。这些成员函数载入 CTL3D32.DLL 并向它注册应用程序。如果你调用了 Enable3dControls 或 Enable3dControlsStatic，你不需要调用 SetDialogBkColor 成员函数。

在与 MFC DLL 连接时，必须使用 Enable3dControls。当与 MFC 库进行静态连接时，必须使用 Enable3dControlsStatic。

仅在专业版和企业版中才具有的特征 只有 Visual C++的专业版和企业版才支持与 MFC 的静态连接。有关的更多信息参见“Visual C++”。

MFC 自动为下列的窗口类提供 3D 控件效果：

·CDialog

·CDialogBar

·CFormView

·CPropertyPage

·CPropertySheet

·CControlBar

·CToolBar

如果你希望具有 3D 外观的控件所在窗口属于上述类，那你只需调用 Enable3dControls 或 Enable3dControlsStatic。如果你希望为基于其它类的窗口中的控件提供 3D 外观，则必须直接调用 CTL3D32 的 API 函数。

示例：

```
#ifdef _AFXDLL
  Enable3dControls( ); // 调用 Enable3dControls
#else
  Enable3dControlsStatic( ); // 调用 Enable3dControlsStatic
#endif
```

EnableShellOpen

说明：

通常在你重载的 InitInstance 函数内调用这个函数，使你的应用程序的用户能够通过在 Windows 的文件管理器内双击文件的方式打开数据文件。与这个函数一起调用 RegisterShellFileTypes 成员函数，或者随应用程序提供一个.REG 文件，用于手动注册文档类型。

示例：

```
BOOL CMyApp::InitInstance()
{
  // ...
  CMultiDocTemplate* pDocTemplate;
  pDocTemplate = new CMultiDocTemplate(
    IDR_MYTYPE,
    RUNTIME_CLASS(CMyDoc),
    RUNTIME_CLASS(CMDIChildWnd), // 标准的 MDI 子框架
    RUNTIME_CLASS(CMyView));
  AddDocTemplate(pDocTemplate);
  // 创建 MDI 的主框架窗口
  CMainFrame* pMainFrame = new CMainFrame;
  if (!pMainFrame->LoadFrame(IDR_MAINFRAME)) return FALSE;
  // 保存指向主框架窗口的指针。
  // 这是框架获得主框架窗口类型的唯一方式。
```

```
    m_pMainWnd = pMainFrame;
    // 打开文件管理器的拖/放和 DDE 打开特性。
    EnableShellOpen();
    RegisterShellFileTypes();
    // ...
    // 根据应用程序启动时传递的 nCmdShow 参数显示主窗口
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();
    // ...
}
```

ExitInstance

返回值

应用程序的退出码；0 表示没有错误，大于 0 的值表示有错误。这个值被用作 WinMain 的返回值。

说明：

框架在 Run 成员函数的内部调用这个函数以退出应用程序的实例。不能在其它的任何地方调用这个成员函数，只能在 Run 成员函数内部调用。这个函数的缺省实现将框架的选项写入应用程序的.INI 文件。重载这个函数以在应用程序退出的时候执行一些清除操作。

GetFirstDocTemplatePosition

返回值

一个 POSITION 值，可以被用于反复或获取对象指针。如果这个列表为空则返回 NULL。

说明：

获得应用程序的第一个文档模板的位置。使用调用 GetNextDocTemplate 时返回的 POSITION 值来获得第一个 CDocTemplate 对象。

GetNextDocTemplate

返回值：指向 CDocTemplate 对象的指针。

参数：

pos 对一个 POSITION 值的引用，该值是上一次对 GetNextDocTemplate 或 GetFirstDocTemplate 的调用所返回的。这一次调用将这个值更新为下一个位置。

说明：

获得 pos 所标识的文档模板，然后将 pos 设置为 POSITION 值。如果你通过对 GetFirstDocTemplatePosition 的调用建立了初始的位置，你可以在一个向前的循环中使用 GetNextDocTemplate。你必须确保 POSITION 值是有效的。如果它无效，那么微软基础类库的调试版本将引起断言。如果获得的文档模板是最后一个有效模板，则新的 pos 值将被设为 NULL。

GetProfileString

返回值

返回值是应用程序的.INI 文件中的字符串，如果找不到该字符串，则为 lpszDefault。框架支持的字符串最大长度为_MAX_PATH。如果 lpszDefault 为 NULL，则返回值是一个空字符串。

参数：

lpszSection 指向一个以 null 结尾的字符串，指定了包含入口的部分。

lpszEntry 指向一个以 null 结尾的字符串，其中包含了要获取字符串的入口。这个值不能为 NULL。

lpszDefault 指向给定入口的缺省字符串值，当初始化文件中找不到入口时使用该值。

说明：

调用这个函数以获得与应用程序的注册表或.INI 文件中指定部分的入口相关的字符串。

这些入口按照如下方式保存：

·Windows NT 该值保存在注册表中

·Windows 3.X 该值保存在 WIN.INI 文件中

·Windows 95 该值保存在 WIN.INI 的缓冲版本中

示例：

```
CString strSection = "My Section";
CString strStringItem = "My String Item";
```

```
CString strIntItem = "My Int Item";
CWinApp* pApp = AfxGetApp();
pApp->WriteProfileString(strSection, strStringItem, "test");
CString strValue;
strValue = pApp->GetProfileString(strSection, strStringItem);
ASSERT(strValue == "test");
pApp->WriteProfileInt(strSection, strIntItem, 1234);
int nValue;
nValue = pApp->GetProfileInt(strSection, strIntItem, 0);
ASSERT(nValue == 1234);
```

HideApplication

说明：

调用这个成员函数以在关闭打开的文档之前隐藏应用程序。

InitInstance

返回值：如果初始化成功，则返回非零值；否则返回 0。

说明：

Windows 允许在同一时刻运行程序的几份拷贝。在概念上，应用程序的初始化可以被分为两个部分：一次性的应用程序初始化工作，这些在应用程序第一次运行时完成，以及示例的初始化工作，每次运行程序的一个拷贝时都会执行这些操作，包括第一次运行时。框架中 WinMain 的实现调用这个函数。

重载 InitInstance 以初始化在 Windows 下运行的应用程序的每个新实例。通常，你重载 InitInstance 以构造主窗口对象并设置 CWinThread::m_pMainWnd 数据成员，使其指向这个窗口。有关重载这个成员函数的更多信息参见“Visual C++ 程序员指南”中的“CWinApp：应用程序类”。

示例：

```
// AppWizard 根据你选择的选项实现重载的 InitInstance 函数。
// 例如，对于下面由 AppWizard 创建的代码，选择了单文档界面（SDI）选项。
// 你可以在 AppWizard 创建的代码中加入其它的每个实例都执行的初始化代码。
BOOL CMyApp::InitInstance()
```

```
{
  // 标准的初始化工作
  // 如果你没有使用这些特性，并且希望减小最终可执行程序的大小，
  // 你应当从下面的初始化例程中移去不必要的代码。
  SetDialogBkColor(); // 将对话框的背景色设为灰色。
  LoadStdProfileSettings(); // 载入标准的 INI 文件选项（包括 MRU）
  // 注册应用程序的文档模板。文档模板
  //被用作文档、框架窗口和视图之间的联系。
  CSingleDocTemplate* pDocTemplate;
  pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CMyDoc),
    RUNTIME_CLASS(CMainFrame), // SDI 的主框架窗口
    RUNTIME_CLASS(CMyView)
   );
  AddDocTemplate(pDocTemplate);
  // 创建一个新（空的）文档
  OnFileNew();
  if (m_lpCmdLine[0] != '\0')
  {
    // 任务：在这里加入命令行处理代码
  }
  return TRUE;
}
```

LoadCursor

返回值：如果成功，则返回光标的句柄；否则返回 NULL。

参数：

lpszResourceName　　指向一个以 null 结尾的字符串，其中包含了光标资源的名字。你可以在这个参数中使用 CString 对象。

nIDResource　光标资源的 ID。

说明：

载入当前可执行文件中以 lpszResourceName 为名或 nIDResource 指定的光标资源。仅当光标以前没有被载入内存时，LoadCursor 才会将它载入；否则，它获得现存资源的句柄。

如果要访问预定义的 Windows 光标，则使用 LoadStandardCursor 或 LoadOEMCursor。

示例：

```
HCURSOR hCursor;
// 载入最初用图形编辑器创建并赋给 ID 值 IDC_MYCURSOR 的光标资源
hCursor = AfxGetApp()->LoadCursor(IDC_MYCURSOR);
HICON LoadIcon( LPCTSTR lpszResourceName ) const;
```

LoadIcon

返回值：如果成功，则返回图标的句柄；否则返回 NULL。

参数：

lpszResourceName　指向一个以 null 结尾的字符串，其中包含了图标资源的名字。你可以在这个参数中使用 CString 对象。

nIDResource　图标资源的 ID。

说明：

载入可执行文件中以 lpszResourceName 为名或是 nIDResource 指定的图标资源。仅当以前图标没有被载入内存时才将它载入；否则它获得现存资源的句柄。你可以使用 LoadStandardIcon 或 LoadOEMIcon 成员函数来访问预定义的 Windows 图标。

注意：这个成员函数调用 Win32 API 函数 LoadIcon，它仅能载入大小在 SM_CXICON 和 SM_CYICON 之内的图标。

LoadStandardCursor

返回值：如果成功，则返回光标的句柄；否则返回 NULL。

参数：

lpszCursorName　标识了预定义的 Windows 光标的 IDC_常量标识符。这些标识符在 WINDOWS.H 中定义。下面给出的是可能取值的列表和 lpszCursorName 的含义：

- IDC_ARROW　标准的箭头光标
- IDC_IBEAM　标准的插入文本光标

- IDC_WAIT 当 Windows 执行耗时操作时使用的沙漏光标
- IDC_CROSS 用于选择的十字光标
- IDC_UPARROW 向上指的箭头光标
- IDC_SIZE 已不被支持。应使用 IDC_SIZEALL
- IDC_SIZEALL 四向箭头。用于改变窗口大小。
- IDC_ICON 以不被支持。应使用 IDC_ARROW。
- IDC_SIZENWSE 双向箭头，尾部在左上角和右下角。
- IDC_SIZENESW 双向箭头，尾部在右上角和左下角。
- IDC_SIZEWE 双向水平箭头
- IDC_SIZENS 双向垂直箭头

说明：

载入 lpszCursorName 指定的 Windows 预定义光标。要访问 Windows 的预定义光标，应使用 LoadStandardCursor 或 LoadOEMCursor 成员函数。

示例：

```
HCURSOR hCursor;
// 载入 Windows 预定义的上箭头光标。
hCursor = AfxGetApp()->LoadStandardCursor(IDC_UPARROW);
```

LoadStandardIcon

返回值：如果成功，则返回图标的句柄；否则返回 NULL。

参数：

lpszIconName 指定了 Windows 的预定义图标的常量标识符。这些标识符在 WINDOWS.H 中定义。下面列出的是可能的取值和 lpszIconName 的含义：

- IDI_APPLICATION 缺省的应用程序图标
- IDI_HAND 手形图标，用于严重警告信息
- IDI_QUESTION 问号图标，用于提示信息
- IDI_EXCLAMATION 感叹号图标，用于警告信息

· IDI_ASTERISK 星号图标，用于通知消息

说明：

这个函数载入 lpszIconName 指定的 Windows 预定义图标。要访问 Windows 的预定义图标，应使用 LoadStandardIcon 或 LoadOEMIcon 成员函数。

OnContextHelp

说明：

你必须在 CWinApp 类的消息映射中加入

ON_COMMAND(ID_CONTEXT_HELP, OnContextHelp)语句，同时加入加速键表入口，通常是 SHIFT+F1，以允许这个成员函数。

OnContextHelp 使应用程序进入帮助模式。光标变为箭头和问号，用户可以移动鼠标指针，然后按下鼠标左键以选择对话框、窗口、菜单或命令按钮。这个成员函数获得光标下方对象的帮助上下文并据此调用 Windows 函数 WinHelp。

OnFileNew

说明：

你必须在 CWinApp 类的消息映射中加入以下语句：

ON_COMMAND(ID_FILE_NEW, OnFileNew)

以允许这个成员函数。

如果允许这个成员函数，则这个函数将处理 File New 命令。

有关此函数的缺省动作以及替换这个成员函数的方法参见技术注释 22。

示例：

```
        // 下面的消息映射是 AppWizard 生成的，将 File New，Open 和 Print Setup 菜单命令映射到框架中这些命令的缺省实现。
        BEGIN_MESSAGE_MAP(CMyApp, CWinApp)
        //{{AFX_MSG_MAP(CMyApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
          // NOTE - the ClassWizard will add and remove mapping macros here.
          //    DO NOT EDIT what you see in these blocks of generated code!
```

```
//}}AFX_MSG_MAP
// 标准的基于文件的文档命令
ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
// 标准的打印设置命令
ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
// 下面的消息映射演示了如何将
// File New，Open 和 Print Setup 菜单命令重定向到
// 你的 CWinApp 派生类中实现的处理函数。你可以
// 使用 ClassWizard 来绑定这些命令，如下面所演示的，
// 因为消息映射入口用//{{AFX_MSG_MAP 和//}}AFX_MSG_MAP
// 括起来。注意，如果需要，你可以将处理函数命名为
// CMyApp::OnFileNew，而不是 CMyApp::OnMyFileNew，其它处理函数类似。
BEGIN_MESSAGE_MAP(CMyApp, CWinApp)
//{{AFX_MSG_MAP(CMyApp)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
ON_COMMAND(ID_FILE_NEW, OnMyFileNew)
ON_COMMAND(ID_FILE_OPEN, OnMyFileOpen)
ON_COMMAND(ID_FILE_PRINT_SETUP, OnMyFilePrintSetup)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

OnFileOpen

说明：

你必须在 CWinApp 类的消息映射中加入以下语句：

ON_COMMAND(ID_FILE_OPEN, OnFileOpen)

以允许这个成员函数。

如果允许这个成员函数，它将处理 File Open 命令。

有关此函数的缺省动作以及替换这个成员函数的方法参见技术注释 22。

示例：

```
// 下面的消息映射是 AppwWizard 生成的，将
// File New，Open 和 Print Setup 菜单命令
// 映射到框架中这些命令的缺省实现。
BEGIN_MESSAGE_MAP(CMyApp, CWinApp)
//{{AFX_MSG_MAP(CMyApp)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
  // NOTE - the ClassWizard will add and remove mapping macros here.
  //    DO NOT EDIT what you see in these blocks of generated code!
//}}AFX_MSG_MAP
// 标准的基于文件的文档命令
ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
// 标准的打印设置命令
ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
// 下面的消息映射演示了如何将
// File New，Open 和 Print Setup 菜单命令重定向到
// 你的 CWinApp 派生类中实现的处理函数。你可以
// 使用 ClassWizard 来绑定这些命令，如下面所演示的，
// 因为消息映射入口用//{{AFX_MSG_MAP 和//}}AFX_MSG_MAP
// 括起来。注意，如果需要，你可以将处理函数命名为
// CMyApp::OnFileNew，而不是 CMyApp::OnMyFileNew，其它处理函数类似。
BEGIN_MESSAGE_MAP(CMyApp, CWinApp)
//{{AFX_MSG_MAP(CMyApp)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
ON_COMMAND(ID_FILE_NEW, OnMyFileNew)
ON_COMMAND(ID_FILE_OPEN, OnMyFileOpen)
ON_COMMAND(ID_FILE_PRINT_SETUP, OnMyFilePrintSetup)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

OnFilePrintSetup

说明：

你必须在 CWinApp 类的消息映射中加入以下语句：

ON_COMMAND(ID_FILE_PRINT_SETUP, OnFilePrintSetup)

以允许这个成员函数。

如果允许这个成员函数，它将处理 File Print 命令。

有关此函数的缺省动作以及替换这个成员函数的方法参见技术注释 22。

示例：

```
// 下面的消息映射是 AppWizard 生成的，将
// File New，Open 和 Print Setup 菜单命令
// 映射到框架中这些命令的缺省实现。
BEGIN_MESSAGE_MAP(CMyApp, CWinApp)
//{{AFX_MSG_MAP(CMyApp)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
// NOTE - the ClassWizard will add and remove mapping macros here.
//    DO NOT EDIT what you see in these blocks of generated code!
//}}AFX_MSG_MAP
// 标准的基于文件的文档命令
ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
// 标准的打印设置命令
ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
// 下面的消息映射演示了如何将
// File New，Open 和 Print Setup 菜单命令重定向到
// 你的 CWinApp 派生类中实现的处理函数。你可以
// 使用 ClassWizard 来绑定这些命令，如下面所演示的，
// 因为消息映射入口用//{{AFX_MSG_MAP 和//}}AFX_MSG_MAP
// 括起来。注意，如果需要，你可以将处理函数命名为
// CMyApp::OnFileNew，而不是 CMyApp::OnMyFileNew，其它处理函数类似。
BEGIN_MESSAGE_MAP(CMyApp, CWinApp)
//{{AFX_MSG_MAP(CMyApp)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
ON_COMMAND(ID_FILE_NEW, OnMyFileNew)
ON_COMMAND(ID_FILE_OPEN, OnMyFileOpen)
ON_COMMAND(ID_FILE_PRINT_SETUP, OnMyFilePrintSetup)
```

```
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

OnHelp

说明：

窗口、对话框、菜单项或工具条按钮的其它对象设置上下文，则应重载这个成员函数。使用你必须在 CWinApp 类的消息映射中加入以下语句：

ON_COMMAND(ID_HELP, OnHelp)

以允许这个成员函数。通常你还将为 F1 键加入一个加速键入口。允许 F1 键只是一个惯例，并不是必须的。如果允许这个成员函数，当用户按下 F1 键的时候，框架就调用这个函数。这个消息处理函数的缺省实现确定对应当前窗口、对话框或菜单项的帮助上下文，然后调用 WINHELP.EXE。如果当前没有可用的上下文，则使用缺省的上下文。

如果要为不是当前具有焦点的所需的帮助上下文 ID 来调用 WinHelp。

OpenDocumentFile

返回值：如果成功，则返回指向 CDocument 对象的指针；否则返回 NULL。

参数：

lpszFileName　　要打开的文件的名字。

说明：

框架调用这个成员函数为应用程序打开指定名字的 CDocument 文件。如果具有该名字的文档已经被打开了，则包含这个文档的第一个框架窗口将被激活。如果应用程序支持多文档模板，则框架使用文件扩展名查找适当的文档模板，试图载入此文档。如果成功，则文档模板为该文档创建一个框架窗口和视。

示例：

```
BOOL CMyApp::InitInstance()
{
  // ...
  if (m_lpCmdLine[0] == '\0')
  {
    // 创建一个新（空的）文档
    OnFileNew();
```

```
    }
    else
    {
      // 打开作为第一个命令行参数传递的文件
      OpenDocumentFile(m_lpCmdLine);
    }
    // ...
  }
```

ParseCommandLine

参数：

rCmdInfo　　对 CCommandLineInfo 对象的引用。

说明：

调用这个函数以解析命令行并将参数发送到 CCommandLineInfo::ParseParam，每次一个。

当你通过 AppWizard 开始一个新的 MFC 项目时，AppWizard 将会创建一个 Ccommand-LineInfo 的本地实例，然后在 InitInstance 成员函数中调用 ProcessShellCommand 和 ParseCommandLine。命令行的过程描述如下：

1.在 InitInstance 中被创建之后，CCommandLineInfo 对象被传递给 ParseCommandLine。

2.随后 ParseCommandLine 反复调用 CCommandLineInfo::ParseParam，每次解析出一个参数。

3.ParseParam 填充 CCommandLineInfo 对象，随后该对象被传递给 ProcessShellCommand。

4.ProcessShellCommand 处理命令行参数和标志。

注意：

如果需要的话，你可以直接调用 ParseCommandLine。

有关命令行标志的描述参见 CCommandLineInfo::m_nShellCommand。

PreTranslateMessage

返回值

如果消息在 PreTranslateMessage 中被完全处理并且不需要进一步处理，则返回非零值。如果消息还需要按照通常方式处理，则返回零。

参数：

pMsg　指向一个 MSG 结构的指针，其中包含了要处理的消息。

说明：

如果要在消息被分派到 Windows 函数 TranslateMessage 和 DispatchMessage 之前过滤窗口消息，则应重载这个函数。缺省的实现执行加速键的转换，因此你可以在重载函数中调用 CWinApp::PreTranslateMessage 成员函数。

请参阅：

::DispatchMessage, ::TranslateMessage

ProcessMessageFilter

返回值：如果消息被处理，则返回非零值；否则返回 0。

参数：

code　指定了钩子代码。这个成员函数使用这些代码以确定如何处理 lpMsg。

lpMsg　指向一个 Windows 的 MSG 结构的指针。

说明：

框架的钩子函数调用这个成员函数以过滤和响应特定的 Windows 消息。钩子函数在事件被发送到应用程序的正常消息处理过程之前处理这些事件。如果你重载了这个高级特性，确保你调用了基类版本以维护框架的钩子处理。

ProcessShellCommand

返回值：如果成功地处理了外壳命令，则返回非零值。如果 InitInstance 返回了 FALSE，则返回 0。

参数：

rCmdInfo　对 CCommandLineInfo 对象的引用。

说明：

这个成员函数被 InitInstance 调用，用以接收 rCmdInfo 所标识的 CCommandLineInfo 对象传递的参数，并执行指定的动作。

当你通过 AppWizard 开始一个 MFC 的新项目时，AppWizard 将创建 CCommandLineInfo 的一个本地实例，然后在 InitInstance 成员函数中调用 ProcessShellCommand 和 ParseCommandLine。命令行按照下面描述的路线传递：

1.在 InitInstance 中被创建以后，CCommandLineInfo 对象将它传递给 ParseCommand-Line。

2.随后 ParseCommandLine 反复调用 CCommandLineInfo::ParseParam，每次解析一个参数。

3.ParseParam 填充 CCommandLineInfo 对象，然后将之传递给 ProcessShellCommand。

4.ProcessShellCommand 处理命令行参数和标志。

CCommandLineInfo 对象中用 CCommandLineInfo::m_nShellCommand 标识的数据成员属于下面的枚举类型，它在 CCommandLineInfo 类中定义。

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

有关这些值的简要描述参见 CCommandLineInfo::m_nShellCommand。

ProcessWndProcException

返回值

这个返回值必须返回给 Windows。通常，对于 Windows 消息，返回值为 0L，对于命令消息，返回值为 1L（TRUE）。

参数：

e　指向没有被捕获的异常。 tr valign=top>　pMsg　一个 MSG 结构，其中包含了有关导致框架出现异常的 Windows 消息的信息。

说明：

每当应用程序不能捕获应用程序的消息处理函数或命令处理函数出现的异常时，框架就调用这个成员函数。不要直接调用这个函数。

这个成员函数的缺省实现创建一个消息框。如果这个未被捕获的异常源于一条菜单、工具条或加速键命令失败，则消息框显示一条“Command failed”消息；否则，显示一条“Internal application error”消息。

如果要为异常提供全局处理，则应重载这个成员函数。若你需要显示该消息框，则应调用基类的函数。

RegisterShellFileTypes

参数：

bCompat　　如果为 TRUE，则为外壳命令 Print 和 Print To 加入注册表入口，使用户能够从外壳直接打印文件，或者是将文件拖动到打印机对象上。同时它也加入一个 DefaultIcon 键。缺省情况下，为了向后的兼容性，这个参数为 FALSE。

说明：

调用这个函数在 Windows 的文件管理器中注册应用程序的所有文档类型。这就使用户能够通过在文件管理器内双击而打开应用程序创建的数据文件。对于应用程序中的每个文档模板，在调用 AddDocTemplate 之后调用 RegisterShellFileTypes。当你调用 RegisterShellFile-Types 的时候，同时也调用 EnableShellOpen 成员函数。

RegisterShellFileTypes 在应用程序维护的 CDocTemplate 对象列表中反复，并且，对于每个文档模板，在 Windows 维护的注册表数据库中加入入口。当用户双击文件的时候，文件管理器利用这个入口打开数据文件。这减小了随应用程序发放.REG 文件的必要性。

如果注册表数据库中已经将指定的文件扩展名与其它文件类型相关联了，则不会创建新的关联。有关注册信息时所用的字符串格式，请参见 CDocTemplate 类。

Run

返回值：WinMain 返回的整数值。

说明：

这个函数提供了缺省的消息循环。Run 函数接收并分派 Windows 消息，直到应用程序接收到一个 WM_QUIT 消息。如果应用程序的消息队列当前不包含任何消息，Run 就调用 OnIdle 以执行空闲时处理。接收到的消息先进入 PreTranslateMessage 成员函数以进行特殊处理，然后发送到 Windows 函数 TranslateMessage，进行标志的键盘转换，最后，调用 Windows 函数 DispatchMessage。

Run 很少被重载，但是你也可以重载它以提供特殊的功能。

RunAutomated

返回值：如果找到了该选项，则返回非零值；否则返回 0。

说明：

调用这个函数以确定是否出现了“/Automation”或“-Automation”选项，这表明服务器程序是否是由客户应用程序启动的。如果有，则从命令行中清除这个选项。有关 OLE 自动化的更多信息，参见“Visual C++ 程序员指南”中的“自动化服务器”。

RunEmbedded

返回值：如果发现了这个选项，则返回非零值；否则返回 0。

说明：

调用这个选项以确定是否出现了“/Embedding”或“-Embedding”选项，这表明服务器应用程序是否是由客户应用程序启动的。如果有，则从命令行中清除这个选项。有关嵌入的更多信息，参见“Visual C++ 程序员指南”中的“服务器：实现服务器”。

SaveAllModified

返回值：如果能够安全地结束应用程序，则返回非零值；如果不安全，则返回 0。

说明：

当应用程序的主框架窗口要被关闭时，或者接收到 WM_QUERYENDSESSION 消息时，框架调用这个函数以保存所有的文档。

这个成员函数的缺省实现为应用程序中所有被修改的文档调用 CDocument::SaveModified 成员函数。

SelectPrinter

参数：

hDevNames　指向 DEVNAMES 结构的句柄，标识了指定打印机的驱动程序、设备和输出端口名。

hDevMode　指向 DEVMODE 结构的句柄，指定了有关设备初始化和打印机环境的信息。

bFreeOld　释放以前选择的打印机。

说明：

调用这个成员函数以选择指定的打印机，并释放先前在 Print 对话框中选择的打印机。

如果 hDevMode 和 hDevNames 都是 NULL，则 SelectPrinter 使用当前的缺省打印机。

SetDialogBkColor

参数：

clrCtlBk 应用程序的对话框的背景颜色。

clrCtlText 应用程序的对话框控件的颜色。

说明：

在 InitInstance 成员函数内调用这个成员函数以设置应用程序中对话框和消息框的缺省背景色和文本颜色。

示例：

```
BOOL CMyApp::InitInstance()
{
  // 标准的初始化
  SetDialogBkColor(); // 将对话框背景颜色设为灰
  LoadStdProfileSettings(); // 载入标准的 INI 文件选项（包括 MRU）
  // ...
}
```

SetRegistryKey

参数：

lpszRegistryKey 字符串指针，包含了键的名字。

nIDRegistryKey 注册表中键的 ID/索引。

说明：

这个函数将应用程序的设置保存在注册表而不是 INI 文件中。这个函数设置 m_pszRegistry Key，它被 CWinApp 的成员函数 GetProfileInt，GetProfileString，WriteProfileInt 和 WriteProfileString 使用。如果调用了这个函数，最近使用（MRU）的文件也被保存到注册表中。通常注册表的键为公司的名字。它保存在如下形式的键中：

HKEY_CURRENT_USER\Software\\\\.

WriteProfileString

返回值：如果成功，则返回非零值；否则返回 0。

参数：

lpszSection　指向一个以 null 结尾的字符串，指定了包含入口的部分。如果这个部分不存在，就创建它。这个部分的名字对大小写不敏感，字符串可以是大写字符和小写字符的任意组合。

lpszEntry　指向一个以 null 结尾的字符串，指定了包含要写入值的入口的部分。如果在指定的部分中不存在这个入口，就创建它。

lpszValue　指向要写入的字符串。如果这个参数为 NULL，则将删除 lpszEntry 指定的键。

说明：

调用这个函数将指定的字符串写入应用程序的注册表或 INI 文件的指定部分。

这些入口按如下方式保存：

·在 Windows NT 中，该值被保存在注册表中。

·在 Windows 3.x 中，该值被保存在 WIN.INI 文件中。

·在 Windows 95 中，该值被保存在 WIN.INI 的缓存版本中。

示例：

```
CString strSection = "My Section";
CString strStringItem = "My String Item";
CString strIntItem = "My Int Item";
CWinApp* pApp = AfxGetApp();
pApp->WriteProfileString(strSection, strStringItem, "test");
CString strValue;
strValue = pApp->GetProfileString(strSection, strStringItem);
ASSERT(strValue == "test");
pApp->WriteProfileInt(strSection, strIntItem, 1234);
int nValue;
nValue = pApp->GetProfileInt(strSection, strIntItem, 0);
```

```
    ASSERT(nValue == 1234);
```

CWindowDC

CWindowDC 类是从 CDC 继承的。它在构造的时候调用 Windows 函数 GetWindowDC，在销毁的时候调用 ReleaseDC。这意味着 CWindowDC 对象可以访问 CWnd 的全部屏幕区域（包括客户区和非客户区）。

CWindowDC

参数：

pWnd　窗口指针，设备环境对象将访问其客户区域。

说明：

构造一个 CWindowDC 对象，它可以访问 pWnd 指向的 CWnd 对象的整个屏幕区域（包括客户区和非客户区）。构造函数调用 Windows 函数 GetWindowDC。

如果对 Windows 函数 GetWindowDC 的调用失败了，将出现异常（属于 Cresource-Exception 类型）。如果 Windows 已经分配了所有可用的设备环境，则可能不能访问设备环境。在 Windows 下，在任何给定时刻应用程序只能使用 5 个公共显示环境之一。

CWinThread

CWinThread 对象代表在一个应用程序内运行的线程。运行的主线程通常由 CWinApp 的派生类提供；CWinApp 由 CWinThread 派生。另外，CWinThread 对象允许一给定的应用程序拥有多个线程。

CWinThread 支持两种线程类型：工作者线程和用户界面线程。工作者线程没有收发消息的功能：例如，在电子表格应用程序中进行后台计算的线程。用户界面线程具有收发消息的功能，并处理从系统收到的消息。CWinApp 及其派生类是用户界面线程的例子。其它用户界面线程也可由 CWinThread 直接派生。

CWinThread 类的对象存在于线程的生存期。如果你希望改变这个特性，将 m_bAutoDelete 设为 FALSE。

要使你的代码和 MFC 是完全线程安全的，CWinThread 类是完全必要的。框架使用的用来维护与线程相关的信息的线程局部数据由 CWinThread 对象管理。

由于依赖 CWinThread 来处理线程局部数据，任何使用 MFC 的线程必须由 MFC 创建。例如，由运行时函数_beginthreadex 创建的线程不能使用任何 MFC API。

为了创建一个线程，调用AfxBeginThread函数。根据你需要工作者线程还是用户界面线程，有两种调用AfxBeginThread的格式。如果你需要用户界面线程，则将指向你的CWinThread派生类的CRuntimeClass的指针传递给AfxBeginThread。如果你需要创建工作者线程，则将指向控制函数的指针和控制函数的参数传递给AfxBeginThread。对于工作者线程和用户界面线程，你可以指定可选的参数来修改优先级，堆栈大小，创建标志和安全属性。

AfxBeginThread 线程将返回指向新的 CWinThread 对象的指针。

与调用 AfxBeginThread 相反，你可以构造一个 CWinThread 派生类的对象，然后调用 CreateThread。如果你需要在连续创建和终止线程的执行之间重复使用 CWinThread 对象，这种两步构造方法非常有用.

CreateThread

返回值：

若成功地创建了线程，返回值为非零；否则，返回值为 0。

参数：

dwCreateFlags　　指定控制线程的创建的附加标志。该标志可以是下列两个值之一：

· CREATE_SUSPENDED　　启动线程时将挂起计数置为一。直到调用 ResumeThread 线程才执行。

· 0　创建后立即启动线程

nStackSize　　指定以字节数计的新线程的堆栈大小。如果为 0，堆栈的大小缺省为与此过程的主线程的堆栈大小相同。

lpSecurityAttrs　　指向一个 SECURITY_ATTRIBUTES 结构，此结构指定线程的安全属性。

说明：

此成员函数创建一个在调用过程的地址空间中运行的线程。使用 AfxBeginThread 可以一步创建线程对象并运行之。如果你需要在连续创建和线程执行的终止之间重复使用线程对象，应使用 CreateThread。

CWinThread

说明：

此函数构造一个 CWinThread 对象。要执行线程，请调用 CreateThread 成员函数。你通常通过调用 AfxBeginThread 来创建线程，AfxBeginThread 将调用此构造函数和 CreateThread 函数。

ExitInstance

返回值：

是线程的退出码；值为 0 表示没有错误，值大于 0 表示有错误发生。通过调用::GetExitCodeThread.可以查询到该值。

说明：

框架通过很少被重载的 Run 成员函数调用此函数以退出线程的这个实例；或者当调用 InitInstance 失败时，调用此函数。

除了在 Run 成员函数内之外，不得在任何地方调用此成员函数。此成员函数仅被用户界面线程使用。

当 m_bAutoDelete 为真时，此函数的缺省实现删除 CWinThread 对象。如果你希望当线程终止时执行额外的清除工作，请重载此函数。当你的程序代码被执行之后，你的 ExitInstance 实现应调用基类的 ExitInstance 函数。

GetMainWnd

返回值：

此函数返回指向一个窗口的指针，这个窗口为两类窗口中的一种。如果你的线程是一个 OLE 服务器的一部分并且拥有一个位于活动容器中的现场激活的对象，此函数返回 CWinThread 对象的 CWinApp::m_pActiveWnd 数据成员。

如果没有位于容器中的现场激活的对象或者你的应用程序不是 OLE 服务器，此函数返回线程对象的 m_pMainWnd 数据成员。

说明：

如果你的应用程序为一个 OLE 服务器，调用此函数以得到应用程序的活动主窗口的指针，而不是直接引用应用程序对象的 m_pmainWnd 成员。对于用户界面线程，调用此函数等价于引用应用程序对象的 m_pActiveWnd 成员。

如果你的应用程序不是一个 OLE 服务器，则调用此函数等价于直接引用应用程序对象的 m_pMainWnd 成员。

重载此函数以修正缺省的行为。

GetThreadPriority

返回值：

此函数返回当前线程在其优先级类中的优先级。此返回值应是下列从高到低列出的优先级值中的一个：

·THREAD_PRIORITY_TIME_CRITICAL

·THREAD_PRIORITY_HIGHEST

·THREAD_PRIORITY_ABOVE_NORMAL

·THREAD_PRIORITY_NORMAL

·THREAD_PRIORITY_BELOW_NORMAL

·THREAD_PRIORITY_LOWEST

·THREAD_PRIORITY_IDLE

关于这些优先级的进一步信息，参阅“Win32 SDK 程序员参考大全”第 4 卷中的函数::SetThreadPriority。

说明：

此函数用于得到线程的当前线程优先级。

InitInstance

返回值：

若初始化成功，返回值为非零；不成功则返回 0。

说明：

InitInstance 必须被重载以初始化每个用户界面线程的新实例。统称，你重载 InitInstance 函数来执行当线程首次被创建时必须完成的任务。

此成员函数仅在用户界面线程中使用。工作者线程的初始化在传递给 AfxBeginThread 的控制函数中完成。

PostThreadMessage

返回值：

如果成功，则返回非零值；否则返回 0。

参数：

message 用户自定义消息的 ID。

wParam 第一个消息参数。

lParam 第二个消息参数。

说明：

调用这个函数以向其它 CWinThread 对象发送一个用户自定义消息。发送的消息通过消息映射宏 ON_THREAD_MESSAGE 被映射到适当的消息处理函数。

PreTranslateMessage

返回值：

如果消息在 PreTranslateMessage 中已经被完全处理，不需要再进一步处理，则返回非零值。如果消息需要按通常方式进一步处理，则返回零。

参数：

pMsg　指向包含了要处理的消息的 MSG 结构。

说明：

如果要在消息被分派到 Windows 函数::TranslateMessage 和::DispatchMessage 之前过滤 Windows 消息，则应重载这个函数。

这个函数仅在用户界面线程中使用。

ProcessMessageFilter

返回值：

如果消息被处理了，则返回非零值；否则返回 0。

参数：

code　指定了钩子代码。这个成员函数使用这些代码来确定如何处理 lpMsg。

lpMsg　指向 Windows 的 MSG 结构的指针。

说明：

框架的钩子函数调用这个函数以过滤并响应特定的 Windows 函数。钩子函数在事件被发送到应用程序的正常消息处理机制之前处理这些事件。

如果你改变了这种高级特性，确保你调用了基类的相应函数以维持框架的钩子处理。

ProcessWndProcException

返回值：

如果产生了一个 WM_CREATE 异常，则返回-1；否则返回 0。

参数：

e　指向没有被处理的异常。

pMsg　指向一个 MSG 结构，其中包含了与导致框架出现异常的 Windows 消息有关的信息。

说明：

每当处理函数没有捕获你的线程消息或命令处理函数所抛出的异常时，框架就调用这个成员函数。

不要直接调用这个成员函数。

这个成员函数的缺省实现仅处理下列消息产生的异常：

命令　动作

WM_CREATE　失败

WM_PAINT　使涉及的窗口有效，防止产生另一个 WM_PAINT 消息

重载这个成员函数以提供对异常的全局处理。仅当你希望执行缺省的动作时才调用基类的函数。

这个成员函数仅在具有消息收发功能的线程中使用。

ResumeThread

返回值：

如果成功，则返回线程的原挂起计数值；否则返回 0xFFFFFFFF。如果返回值为零，则表示当前线程没有被挂起。如果返回值为 1，线程被挂起，但是即将重新启动。任何大于 1 的返回值都表明线程将继续挂起。

说明：

调用这个函数以使被 SuspendThread 成员函数所挂起的线程恢复执行，或者使用 CREATE_SUSPENDED 标志创建的线程恢复执行。当前线程的挂起计数被减小 1。如果挂起计数被减小到 0，线程将恢复执行；否则线程继续被挂起。

Run

返回值：

线程返回的一个整数值。这个值可以通过调用::GetExitCodeThread 来获得。

说明：

这个函数为用户界面线程提供了缺省的消息循环。Run 接收并分派 Windows 消息，直到它接收到一个 WM_QUIT 消息。如果线程的当前消息队列中不包含消息，Run 就调用 OnIdle 以执行空闲处理。接收到的消息被送到 PreTranslateMessage 成员函数以进行特殊处理，然后被发送到 Windows 函数::TranslateMessage 以进行标志键盘转换。最后，调用 Windows 函数::DispatchMessage。

Run 函数很少被重载，但是你可以重载它以提供特殊的功能。

这个成员函数仅在用户界面线程中使用。

SetThreadPriority

返回值：

如果这个函数成功地执行，则返回非零值；否则返回 0。

参数：

nPriority 指　定了线程在其优先权类中的新优先权级。这个参数必须是下列值之一，从最高优先权到最低：

·THREAD_PRIORITY_TIME_CRITICAL

·THREAD_PRIORITY_HIGHEST

·THREAD_PRIORITY_ABOVE_NORMAL

·THREAD_PRIORITY_NORMAL

·THREAD_PRIORITY_BELOW_NORMAL

·THREAD_PRIORITY_LOWEST

·THREAD_PRIORITY_IDLE

有关这些优先权的更多信息参见“Win32 SDK 程序员参考”第四卷中的::SetThreadPriority。

说明：

这个函数设置当前线程在它所属的优先权类中的优先权级。它只能在 CreateThread 成功返回之后调用。

SuspendThread

返回值：

如果成功，则返回线程原来的挂起计数值；否则返回 0xFFFFFFFF。

说明：

增加当前线程的挂起计数。如果线程的挂起计数大于零，则该线程将不被执行。线程可以通过调用 ResumeThread 成员函数恢复执行。

CWnd

CWnd 类提供了微软基础类库中所有窗口类的基本功能。

CWnd 对象与 Windows 的窗口不同，但是两者有紧密联系。CWnd 对象是由 CWnd 的构造函数和析构函数创建或销毁的。另一方面，Windows 的窗口是 Windows 的一种内部数据结构，它是由 CWnd 的 Create 成员函数创建的，而由 CWnd 的虚拟析构函数销毁。DestroyWindow 函数销毁 Windows 的窗口，但是不销毁对象。

CWnd 类和消息映射机制隐藏了 WndProc 函数。接收到的 Windows 通知消息通过消息映射被自动发送到适当的 CWnd OnMessage 成员函数。你可以在派生类中重载 OnMessage 成员函数以处理成员的特定消息。

CWnd 类同时还使你能够为应用程序创建 Windows 的子窗口。先从 CWnd 继承一个类，然后在派生类中加入成员变量以保存与你的应用程序有关的数据。在派生类中实现消息处理成员函数和消息映射，以指定当消息被发送到窗口时应该如何动作。

你可以经过两个步骤来创建一个子窗口。首先，调用构造函数 CWnd 以创建一个 CWnd 对象，然后调用 Create 成员函数以创建子窗口并将它连接到 CWnd 对象。

当用户关闭你的子窗口时，应销毁 CWnd 对象，或者调用 DestroyWindow 成员函数以清除窗口并销毁它的数据结构。

在微软基础类库中，从 CWnd 派生了许多其它类以提供特定的窗口类型。这些类中有许多，包括 CFrameWnd,CMDIFrameWnd,CMDIChildWnd,CView 和 CDialog,被用来进一步派生。从 CWnd 派生的控件类，如 CButton，可以被直接使用，也可以被进一步派生出其它类来。

BeginPaint

返回值：标识了 CWnd 的设备环境。这个指针可能是临时的，不应在 EndPaint 之外保存。

参数：

lpPaint　执行 PAINTSTRUCT 结构，用于获取绘图信息。

说明：

为绘图准备 CWnd 并用与绘图有关的信息填充 PAINTSTRUCT 数据结构。

绘图结构中包含了一个 RECT 数据结构，它包含了完全封闭更新区域的最小矩形以及一个标志，指明背景是否需要擦除。

更新区域是由 Invalidate、InvalidateRect 或 InvalidateRgn 成员函数设置的，并且在更新区域改变大小、移动、创建、滚动或执行其它会影响客户区的操作后由系统设置。如果更新区域被标记为需要擦除，则 BeginPaint 发送一个 WM_ONERASEBKGND 消息。

除非是在响应 WM_PAINT 消息的时候，否则不要调用 BeginPaint 成员函数。每个对 BeginPaint 成员函数的调用都必须有对应的对 EndPaint 成员函数的调用。如果在这个区域中的插字符需要被重画，那么 BeginPaint 成员函数自动隐藏插字符以免被擦除。

BindDefaultProperty

参数：

dwDispID　　指定要与数据源控件绑定的数据绑定控件的属性的 DISPID。

vtProp　　指定要绑定的属性的类型--例如，VT_BSTR，VT_VARIANT 等等。

szFieldName　指定要与属性绑定的字段的名字，位于数据源控件提供的游标中。

pDSCWnd　　指向拥有数据源控件的窗口，属性将与该窗口绑定。利用 DCS 的宿主窗口的资源 ID 调用 GetDlgItem 以获取这个指针。

说明：

如类型库中标记的那样将调用对象缺省的简单绑定属性（比如编辑控件）与游标绑定起来，该游标是通过数据源控件的 DataSource、UserName、Password 和 SQL 属性定义的。你调用这个函数的 CWnd 对象必须是一个数据绑定对象。

BindDefaultProperty 必须在下面的上下文中使用：

```
BOOL CMyDlg::OnInitDialog()
{
    ...
    CWnd* pDSC = GetDlgItem(IDC_REMOTEDATACONTROL);
    CWnd* pList = GetDlgItem(IDC_DBLISTBOX);
    pList->BindDefaultProperty(0x2, VT_BSTR, _T("CourseID"), pDSC);
    CWnd* pEdit = GetDlgItem(IDC_MASKEDBOX);
    pEdit->BindDefaultProperty(0x16, VT_BSTR, _T("InstuctorID"), pDSC);
    ...
```

```
    return TRUE;
}
```

BindProperty

参数：

dwDispID　指定了要与数据源控件绑定的数据绑定控件的属性的 DISPID。

pWndDSC　指向拥有数据源控件的窗口，属性将与该窗口绑定。利用 DCS 的宿主窗口的资源 ID 调用 GetDlgItem 以获取这个指针。

说明：

将数据绑定控件（如网格控件）的游标绑定属性与数据源控件绑定起来，并且在 MFC 绑定管理器中注册这种联系。BindProperty 必须在下面的上下文中使用：

```
BOOL CMyDlg::OnInitDialog()
{
    ...
    CWnd* pDSC = GetDlgItem(IDC_REMOTEDATACONTROL);
    CWnd* pList= GetDlgItem(IDC_DBLISTBOX);
    pList.BindProperty(0x9, pDSC);
    ...
    return TRUE;
}
```

CancelToolTips

参数：

bKeys　如果为 TRUE，当按下键时取消工具提示，并将状态条文本设为缺省值；否则为 FALSE。

说明：

如果当前显示了工具提示，则调用这个函数以从屏幕上清除工具提示。

注意:这个成员函数对你的代码管理的工具提示不起作用。它只影响 CWnd::Enable ToolTips 管理的工具提示控件。

CenterWindow

参数：

pAlternateOwner　指向一个窗口的指针，本窗口将被定位到该窗口（而不是其它的父窗口）的中央。

说明：

这个函数将一个窗口定位到它的父窗口的中央。通常在 CDialog::OnInitDialog 中调用，用于将对话框定位到应用程序主窗口的中央。在缺省情况下，这个函数将子窗口定位到它们的父窗口的中央，而将弹出窗口定位到拥有者的中央。如果弹出窗口没有拥有者，它将被定位到屏幕中央。如果要使窗口根据不是父窗口也不是拥有者的窗口来定位，则可以将 pAlternateOwner 参数可以被设为一个有效的窗口。如果要强迫相对于屏幕定位，则应在 pAlternateOwner 参数中传递 CWnd::GetDesktopWindow 返回的值。

Create

返回值：如果成功，则返回非零值；否则返回 0。

参数：

lpszClassName　指向一个以 null 结尾的字符串，它命名了一个 Windows 的窗口类（一个 WNDCLASS 结构）。类名可以是用全局函数 AfxRegisterWndClass 注册的任何名字，也可以是任何预定义的控制类名。如果该参数为 NULL，则使用缺省的 CWnd 属性。

lpszWindowName　指向一个 null 结尾的字符串，其中包含了窗口名。

dwStyle　指定了窗口风格属性。不能使用 WS_POPUP。如果你想要创建一个弹出窗口，则应使用 CWnd::CreateEx。

rect　窗口的位置和大小，使用 pParentWnd 的客户区坐标。

pParentWnd　父窗口。

nID　子窗口的 ID。

pContext　窗口的创建上下文。

说明：

创建一个 Windows 的子窗口，并将它连接到 CWnd 对象上。

你可以经过两步构造一个子窗口。首先，调用构造函数，创建一个 CWnd 对象。然后调用 Create，创建一个 Windows 的子窗口，并将它连接到 CWnd。Create 函数初始化窗口的类名、窗口名，并为它的风格、父窗口和 ID 注册值。

CreateCaret

参数：

pBitmap 标识了一个位图，定义了插入字符的形状。

说明：

为系统插入字符创建一个新的形状，并声明对插入字符的所有权。

这个位图必须是先前用 CBitmap::CreateBitmap 成员函数、Windows 的 CreateDIBitmap 函数或 CBitmap::LoadBitmap 成员函数创建的。

CreateCaret 自动销毁原来的插字符形状，并不考虑哪个窗口拥有这个插字符。在被创建之后，插字符最初是隐藏的。要显示插字符，必须调用 ShowCaret 成员函数。

系统插字符是一种共享资源。CWnd 只应在它具有输入焦点或者活动的时候才创建插字符。在它失去输入焦点或变为非活动之前，它应该销毁插字符。

CreateControl

返回值：如果成功，则返回非零值；否则返回 0。

参数：

lpszClass 这个字符串可能包含了该类的 OLE 的“短名”（ProgID），例如，“CIRC3.Circ3Ctrl.1”。这个名字应该与控件注册的名字相匹配。或者，这个字符串可能包含了 CLSID 的字符串形式，包括在大括号内，例如，“{9DBAFCCF-592F-101B-85CE-00608CEC297B}”。在其它情况下，CreateControl 将该字符串转换为对应的类 ID。

lpszWindowName 指向要显示在控件中的文本的指针。设置了控件的标题或文本属性（如果有）的值。如果该指针为 NULL，则不改变控件的标题或文本属性。

dwStyleWindows 风格。可能的取值在说明部分列出。

rect 指定了控件的大小和位置。它可以是一个 CRect 对象，也可以是一个 RECT 结构。

pParentWnd 指定了控件的父窗口。它不能为 NULL。

nID　　指定了控件的 ID。

pPersist　指向一个 CFile 对象的指针，其中包含了控件的永久状态。缺省值为 NULL，表明控件在初始化自己的时候并不读任何永久性的存储。如果该参数不是 NULL，它必须是一个 CFile 派生类对象的指针，其中包含了控件的永久数据，可以是流的形式，也可以是存储的形式。这些数据必须是在客户以前的活动中保存的。CFile 对象中还可以包含其它数据，但是当调用 CreateControl 的时候，它的读写指针必须定位在永久数据的第一个字节。

bStorage　　指明 pPersist 中的数据是被解释为 IStorage 数据还是 IStream 数据。如果 pPersist 中的数据是一种存储，则 bStorage 应该为 TRUE。如果 pPersist 中的数据是一个流，则 bStorage 应该是 FALSE。缺省值为 FALSE。

bstrLicKey　可选的许可键数据。这个数据仅在创建需要运行时许可的控件时才需要。如果该控件支持许可，要成功地创建控件，你必须提供许可键。缺省的值为 NULL。

clsid　　控件的唯一的类 ID。

说明：

使用这个成员函数来创建一个 OLE 控件，在 MFC 程序中，它用一个 CWnd 对象来代表。CreateControl 与 CWnd::Create 函数类似，CWnd::Create 为 CWnd 创建一个窗口。CreateControl 创建一个 OLE 控件，而不是其它的普通窗口。

CreateControl 仅支持 Windows 的 dwStyle 风格的一个子集：

· WS_VISIBLE　　创建一个最初可见的窗口。如果你希望该控件立即可见，向普通窗口一样，则需要这个风格。

· WS_DISABLED　创建一个最初被禁止的窗口。被禁止的窗口不能接收用户的输入。如果控件具有 Enable 属性，则可以设置这个风格。

· WS_BORDER　　创建一个带有细边框的窗口。如果控件具有 BorderStyle 属性，则可以设置这个风格。

· WS_GROUP　指定了一组控件中的第一个控件。用户可以在组中使用方向键来把键盘焦点从一个控件转移到另一个控件。在第一个控件之后所有用 WS_GROUP 风格定义的控件都属于同一组。下一个具有 WS_GROUP 风格的控件将结束这个组并开始一个新组。

· WS_TABSTOP　指明当用户按下 TAB 键时，控件可以接收键盘焦点。按下 TAB 键时键盘焦点转移到具有 WS_TABSTOP 风格的下一个控件。

CWnd

说明：

构造一个 CWnd 对象。必须在 CreateEx 或 Create 处于函数被调用以后才会创建 Windows 的窗口并与之连接。

Default

返回值：依赖于发出的消息。

说明：

调用缺省的窗口过程。缺省的窗口过程提供了对应用程序没有处理的任何窗口消息的缺省处理。这个处理函数确保每个消息都被处理。

DefWindowProc

返回值：依赖于发送的消息。

参数：

message 指定了要处理的 Windows 消息。

wParam 指定了与消息有关的附加信息。

lParam 指定了与消息有关的附加信息。

说明：

这个函数调用缺省的窗口过程，提供了对应用程序没有处理的任何窗口消息的缺省处理。这个成员函数确保每个消息都被处理。它必须用与窗口过程接收到的参数相同的参数来调用。

DeleteTempMap

说明：

这个函数被 CWinApp 对象的空闲时间处理函数自动调用。删除 FromHandle 成员函数所产生的任何临时 CWnd 对象。

DestroyWindow

返回值：如果销毁了窗口，则返回非零值；否则返回 0。

说明：

这个函数销毁一个与 CWnd 对象相连接的 Windows 窗口。DestroyWindow 成员函数向窗口发送一个适当的消息，以使该窗口变为非激活的并移去输入焦点。它还销毁窗口的菜单，清除应用程序的队列，销毁定时器，清除剪贴板拥有权，并且如果 CWnd 对象位于剪贴板观察器链的顶部，还打断剪贴板观察器链。它向窗口发送 WM_DESTROY 消息和 WM_NCDESTROY 消息。它不销毁 CWnd 对象。

DestroyWindow 是一个用于执行清除工作的占位符。因为 DestroyWindow 是一个虚拟函数，在 ClassWizard 中，它显示在任何 CWnd 的派生类中。但是即使你在自己的 CWnd 派生类中重载了这个函数，也不必调用 DestroyWindow。如果在 MFC 代码中没有调用 DestroyWindow，并且你希望调用它的话，必须在自己的代码中调用它。

例如，假定你在 CView 的派生类中重载了 DestroyWindow。由于 MFC 的源代码在任何 CFrameWnd 的派生类中都没有调用 DestroyWindow，因此除非你调用了它，否则你重载的 DestroyWindow 不会被调用。

如果该窗口是其它窗口的父窗口，当父窗口被销毁时，这些子窗口将被自动销毁。DestroyWindow 成员函数首先销毁子窗口，然后销毁本窗口。

DestroyWindow 成员函数也销毁 CDialog::Create 创建的无模式对话框。如果要被销毁的 CWnd 是一个子窗口并且没有设置 WS_EX_NOPARENTNOTIFY 风格，则 WM_PARENTNOTIFY 消息将被发送到父窗口。

EndPaint

参数：

lpPaint　指向一个 PAINTSTRUCT 结构，其中包含了 BeginPaint 成员函数接收到的绘图信息。

说明：

标记了给定窗口的绘图过程的结束。每个被调用的 BeginPaint 成员函数都需要有一个对对应的 EndPaint 成员函数，仅在完成绘图操作以后。

如果 BeginPaint 成员函数隐藏了插字符，则 EndPaint 在屏幕上恢复插字符。

FindWindow

返回值：

标识了具有指定的类名或窗口名的窗口。如果没有找到这样的窗口，则返回 NULL。

返回的 CWnd*值可能是临时的，不能被保存以供将来使用。

参数：

lpszClassName　指向一个以 null 结尾的字符串，指定了窗口类（一个 WNDCLASS 结构）的名字。如果 lpClassName 为 NULL，则所有的类名都匹配。

lpszWindowName 指向一个以 null 结尾的字符串，指定了窗口的名字（窗口的标题）。如果 lpWindowName 为 NULL，所有的窗口名都匹配。

说明：

返回顶层的 CWnd，其窗口类是由 lpszClassName，其窗口名或标题是 lpszWindowName 给定的。这个函数不搜索子窗口。

FlashWindow

返回值：如果在调用 FlashWindow 成员函数之前窗口是激活的，则返回非零值；否则返回 0。

参数：

bInvert　指定 CWnd 是要闪烁还是返回它的原始状态。如果 bInvert 为 TRUE，则 CWnd 从一种状态闪烁到另一种状态。如果 bInvert 为 FALSE，则窗口返回它的原始状态（可以是活动或非活动的）。

说明：

使给定窗口闪烁一次。要实现连续的闪烁，则应该创建一个系统定时器，并反复调用 FlashWindow。使 CWnd 闪烁意味着改变它的标题条的外观，就像 CWnd 从非活动状态改变到活动状态，或反之（非活动状态的标题条改变到活动标题条；活动标题条改变到非活动标题条）。

通常，窗口被闪烁以通知用户它需要被注意，但是它当前不具有输入焦点。

仅当窗口得到了输入焦点并且不再需要闪烁时，bInvert 才应被设为 FALSE，在等待输入焦点的时候，应当在对这个函数的连续调用中将它设为 TRUE。

对于最小化窗口，这个函数总是返回非零值。如果窗口是最小化的，FlashWindow 只是简单地闪烁窗口的图标，对于最小化窗口 bInvert 会被忽略。

GetActiveWindow

返回值：

返回活动窗口，如果在被调用时没有活动窗口，则返回 NULL。这个指针可能是临时的，不能被保存以供将来使用。

说明：

这个函数获得活动窗口的指针。活动窗口或者是拥有当前输入焦点的窗口，或者是用 SetActiveWindow 成员函数激活的窗口。

GetCurrentMessage

返回值：

返回一个指向 MSG 结构的指针，该结构中包含了窗口当前处理的消息。只应在 OnMessage 消息处理函数内部调用

GetDC

返回值：

如果调用成功，则返回 CWnd 客户区的设备环境；否则，返回 NULL。这个指针可能是临时的，不能被保存以供将来使用。

说明：

这个函数获得一个指针，指向一个客户区的公用的、属于类的或者私有的设备环境，依赖于为 CWnd 指定的类风格。对于公用的设备环境，GetDC 每次获得设备环境时都给它赋予缺省值。对于属于类的或者私有的设备环境，GetDC 保持原来的属性不变。在随后的图形设备接口（GDI）函数中可以使用设备环境以在客户区中绘图。

除非设备环境属于一个窗口类，否则在绘图之后必须调用 ReleaseDC 成员函数以释放设备环境。由于在同一时刻只有五个公用设备环境可供使用，因此如果释放设备环境时失败，可能导致其它应用程序不能访问设备环境。

如果在注册窗口类的时候，在 WNDCLASS 的风格中指定了 CS_CLASSDC，CS_OWNDC 或 CS_PARENTDC，则 GetDC 成员函数将返回属于 CWnd 类的设备环境。

GetDesktopWindow

返回值：标识了 Windows 的桌面窗口。这个指针可能是临时的，因此不能被保存以供将来使用。

说明：

这个函数返回 Windows 的桌面窗口。桌面窗口覆盖整个屏幕，并且所有的图标和其它窗口都画在它上面。

GetFocus

返回值：

指向拥有当前焦点的窗口的指针，如果没有焦点窗口，则返回 NULL。这个指针可能是临时的，不能被保存以供将来使用。

说明：这个函数获得指向当前拥有输入焦点的 CWnd 的指针。

GetIcon

返回值：指向一个图标的句柄。如果不成功，则返回 NULL。

参数：

bBigIcon　　如果为 TRUE，则指定了 32×32 像素的图标；如果为 FALSE，则指定了 16×16 像素的图标。

说明：

调用这个函数以获得大图标（32×32）或小图标（16×16）的句柄，由 bBigIcon 指定。

GetNextWindow

返回值：

如果这个成员函数执行成功，则返回窗口管理器中的下一个（或前一个）窗口。返回的指针可能是临时的，不应保存以供将来使用。

参数：

nFlag　指定了该函数是返回下一个窗口还是前一个窗口的指针。它可以是GW_HWNDNEXT，表明返回窗口管理器列表中 CWnd 对象之后的窗口，或者是GW_HWNDPREV，返回窗口管理器列表中的前一个窗口。

说明：

这个函数在窗口管理器列表中搜索下一个（或前一个）窗口。窗口管理器的列表中包含了所有顶层窗口、它们的相关子窗口和任意子窗口的子窗口。

如果 CWnd 是一个顶层窗口，则函数搜索下一个（或前一个）顶层窗口；如果 CWnd 是一个子窗口，则函数搜索下一个（或前一个）子窗口。

GetParent

返回值：

如果这个成员函数执行成功，则返回父窗口指针；否则返回值为 NULL，表明发生了错误或没有父窗口。

返回的指针可能是临时的，不应保存以供将来使用。

说明：

调用这个函数以获得子窗口的父窗口（如果有）的指针。GetParent 函数返回直接父窗口的指针。与此不同，GetParentOwner 函数返回不是子窗口（不具有 WS_CHILD 风格）的最直接父窗口或拥有者窗口的指针。如果你在子窗口中还有子窗口，则 GetParent 和GetParentOwner 返回不同的结果。

DWORD GetStyle( ) const;

返回值：窗口的风格。

GetSystemMenu

返回值：

如果 bRevert 为 FALSE，则标识了控制菜单的一个拷贝；如果为 TRUE，则返回值没有定义。

返回的指针可能是临时的，不能被保存以供将来使用。

参数：

bRevert 指定要采取的动作。如果 bRevert 为 FALSE，则 GetSystemMenu 返回当前使用的控制菜单的一个拷贝的句柄。这个拷贝最初与控制菜单一样，但是可以被修改。如果 bRevert 为 TRUE，GetSystemMenu 将控制菜单复位到原来的状态。以前控制菜单可能发生的变化都被销毁。这时返回值没有定义。

说明：

这个函数允许应用程序访问控制菜单，用于拷贝和修改。

任何不使用 GetSystemMenu 以生成它自己的控制菜单拷贝的窗口将接收标准的控制菜单。

GetSystemMenu 成员函数返回的指针可以在 CMenu::AppendMenu，CMenu::InsertMenu 或 CMenu::ModifyMenu 中使用，用于改变控制菜单。

控制菜单中最初包含了用不同的 ID 标识的项，如 SC_CLOSE，SC_MOVE 和 SC_SIZE。控制菜单中的项产生 WM_SYSCOMMAND 消息。所有的预定义控制菜单项都具有大于 0xF000 的 ID 值。如果应用程序在控制菜单中加入了项，则必须使用小于 F000 的 ID 值。

Windows 可能自动使标准控制菜单上的项变灰。CWnd 可以通过响应 WM_INITMENU 消息来执行它自己的选中或变灰操作，这个消息是在任何菜单要被显示之前发送的。

GetTopWindow

返回值：

标识了 CWnd 的子窗口链表中的顶层子窗口。如果没有子窗口，则返回值为 NULL。

返回值可能是临时的，不应保存以供将来使用。

说明：

这个函数搜索属于 CWnd 的顶层子窗口。如果 CWnd 没有子窗口，这个函数返回 NULL。

GetWindow

返回值：

返回要求的窗口指针；如果没有，则返回 NULL。

返回的指针可能是临时的，不应保存以供将来使用。

参数：

nCmd　指定了 CWnd 和返回的窗口之间的关系。可以取下列值之一：

· GW_CHILD　标识了 CWnd 的第一个子窗口。

· GW_HWNDFIRST　如果 CWnd 是一个子窗口，则返回它的第一个兄弟窗口；否则返回列表中的第一个顶层窗口。

· GW_HWNDLAST　如果 CWnd 是一个子窗口，则返回最后一个兄弟窗口；否则返回列表中的最后一个顶层窗口。

· GW_HWNDNEXT　返回窗口管理器中的下一个窗口。

· GW_HWNDPREV　返回窗口管理器中的前一个窗口。

· GW_OWNER　标识了 CWnd 的拥有者。

GetWindowDC

返回值：

如果这个函数成功，则返回给定窗口的显示环境；否则返回 NULL。

返回的指针可能是临时的，不应保存以供将来使用。在每次成功地调用了 GetWindowDC 之后，必须调用 ReleaseDC。

说明：

这个函数获得整个窗口的显示环境，包括标题条、菜单和滚动条。窗口的显示环境允许程序在 CWnd 的任何地方绘图，因为该环境的原点是在 CWnd 的左上角，而不是客户区的左上角。

每次获得环境的时候都给它赋以缺省的属性。以前的设置将会丢失。

GetWindowDC 用于在 CWnd 的非客户区实现特殊的绘图效果。不推荐在任何窗口的非客户区绘图。

可以利用 Windows 的 GetSystemMetrics 函数来获得非客户区的不同部分的大小，如标题条、菜单和滚动条。

在绘图结束以后，必须调用 ReleaseDC 成员函数以释放显示环境。如果没有成功地释放显示环境，则可能会严重影响应用程序要求的绘图，因为在同一时刻能打开的显示设备环境的数目是有限的。

GetWindowText

返回值：

指定了要拷贝的字符串的长度，以字节为单位，不包括结尾的空字符。如果 CWnd 没有标题或标题为空，则为 0。

参数：

lpszStringBuf　　指向要接收窗口标题的复制字符串的缓冲区。

nMaxCount　指定了要拷贝的缓冲区的最大字符数目。如果字符串比 nMaxCount 指定的数目还要长，则被截断。

rString　用于接收窗口标题的复制字符串的 CString 对象。

说明：

这个函数将 CWnd 的标题（如果有）拷贝到 lpszStringBuf 指向的缓冲区或者目的字符串 rString。如果 CWnd 对象是一个控件，则 GetWindowText 成员函数将拷贝控件内的文本（而不是控件的标题）。这个成员函数会向 CWnd 对象发送一个 WM_GETTEXT 消息。

IsChild

返回值：描述函数的结果。若 pWnd 标识的窗口是 CWnd 的子窗口，则返回非零值；否则返回 0。

参数：

pWnd　标识了要测试的窗口。

说明：

这个函数指明 pWnd 标识的窗口是否是 CWnd 的子窗口或直接后代窗口。如果 CWnd 原始弹出窗口到该子窗口的父窗口链中，则该子窗口是 CWnd 的直接后代窗口。

KillTimer

返回值：

指定了函数的结果。如果事件已经被销毁，则返回值为非零值。如果 KillTimer 成员函数不能找到指定的定时器事件，则返回 0。

参数：

nIDEvent　　传递给 SetTimer 的定时器事件值。

说明：

销毁以前调用 SetTimer 创建的用 nIDEvent 标识的定时器事件。任何与此定时器有关的未处理的 WM_TIMER 消息都从消息队列中清除。

void MoveWindow(int x, int y, int nWidth, int nHeight, BOOL bRepaint = TRUE);

MoveWindow

参数：

x　指定了 CWnd 的左边的新位置。

y　指定了 CWnd 的顶部的新位置。

nWidth　指定了 CWnd 的新宽度。

nHeight　指定了 CWnd 的新高度。

bRepaint　　指定了是否要重画 CWnd。如果为 TRUE，则 CWnd 象通常那样在 OnPaint 消息处理函数中接收到一条 WM_PAINT 消息。如果这个参数为 FALSE，则不会发生任何类型的重画操作。这应用于客户区、非客户区（包括标题条和滚动条）和由于 CWnd 移动而露出的父窗口的任何部分。当这个参数为 FALSE 的时候，应用程序必须明确地使 CWnd 和父窗口中必须重画的部分无效或重画。

lpRect　CRect 对象或 RECT 结构，指定了新的大小和位置。

说明：

这个函数改变窗口的位置和大小。

对于顶层的 CWnd 对象，x 和 y 参数是相对于屏幕的左上角的。对于子对象，它们是相对于父窗口客户区的左上角的。

MoveWindow 函数发送一条 WM_GETMINMAXINFO 消息。处理这个消息时，CWnd 得到一个改变最大和最小的窗口缺省值的机会。如果传递给 MoveWindow 成员函数的参数超过了这些值，则在 WM_GETMINMAXINFO 处理函数中可以用最小或最大值来代替这些值。

OnActivate

参数：

nState　指定 CWnd 是要被激活还是取消活动状态。它可以是下列值之一：

· WA_INACTIVE　窗口将被取消活动状态。

· WA_ACTIVE　窗口将通过不同于鼠标点击的某些方法激活（例如，用键盘接口选择窗口）。

· WA_CLICKACTIVE　窗口经鼠标点击而激活。

pWndOther　指向要激活或取消活动状态的 CWnd 对象的指针。这个指针可以为 NULL，也有可能是临时的。

bMinimized　指定了要激活或取消活动状态的 CWnd 的最小化状态。如果值为 TRUE，表明窗口是最小化的。如果该值为 TRUE，则 CWnd 将被激活，否则将取消活动状态。

说明：

当 CWnd 对象被激活或取消活动状态时，框架调用这个成员函数。首先调用要取消活动状态的主窗口的 OnActivate 函数，然后调用要被激活的主窗口的 OnActivate 函数。

如果 CWnd 对象是被鼠标点击激活的，则它还将接收到对 OnMouseActivate 的调用。

注意：

框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnActivateApp

参数：

bActive　指定了 CWnd 是要被激活还是被取消活动状态。TRUE 意味着 CWnd 要被激活。FALSE 意味着 CWnd 将失去活动状态。

hTask　指定了任务句柄。如果 bActive 为 TRUE，则该句柄标识了拥有被取消活动状态的 CWnd 对象的任务。如果 bActive 为 FALSE，则该句柄标识了拥有被激活的 CWnd 对象的任务。

说明：

框架为被激活的任务的所有顶层窗口或被取消活动状态的任务的所有顶层窗口调用这个成员函数。

注意：

框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnCancelMode

说明：

框架调用这个成员函数以通知 CWnd 取消内部模式。如果 CWnd 拥有焦点，则当对话框或消息框被显示时，它的 OnCancelMode 成员函数将被调用。这使 CWnd 拥有一个取消模式的机会，如鼠标捕获等。

缺省的实现通过调用 Windows 的 ReleaseCapture 来作响应。在你的派生类中重载这个函数以处理其它模式。

OnChar

参数：

nChar　包含了键的字符代码值。

nRepCnt　包含了重复计数，当用户按下键时重复的击键数目。

nFlags　包含了扫描码，键暂态码，以前的键状态以及上下文代码，如下面的列表所示：

值　含义

0--15　指定了重复计数。其值是用户按下键时重复的击键数目

16--23　指定了扫描码。其值依赖于原始设备制造商（OEM）

24　指明该键是否是扩展键，如增强的 101 或 102 键盘上右边的 ALT 或 CTRL 键如果它是个扩展键，则该值为 1；否则，值为 0

25--28　Windows 内部使用

29　指定了上下文代码。如果按键时 ALT 键是按下的，则该值为 1；否则，值为 0

30　指定了以前的键状态。如果在发送消息前键是按下的，则值为 1；如果键是弹起的，则值为 0

31　指定了键的暂态。如果该键正被放开，则值为 1，如果键正被按下，则该值为 0

说明：

当击键被转换为非系统字符时，框架调用这个成员函数。这个函数是在 OnKeyUp 成员函数之前，OnKeyDown 成员之后调用的。OnChar 包含了被按下或放开的键值。

由于按键和产生的 OnChar 调用不必是一一对应的，因此 nFlags 中的信息对应用程序通常是没有用的。NFlags 中的信息仅对最近在 OnChar 之前调用的 OnKeyUp 成员函数或 OnKeyDown 成员函数有用。

对于 IBM 增强型 101 和 102 键键盘，键盘的主体部分的增强键是右边的 ALT 和 CTRL 键；还有数字键盘左侧的 INS，DEL，HOME，END，PAGE UP，PAGE DOWN 以及箭头键等；以及数字键盘上的斜杠（/）和 ENTER 键。其它键盘可能会支持 nFlags 中的扩展键位。

注意：

框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnChildNotify

返回值：如果窗口负责处理发送给它的父窗口的消息，则返回非零值；否则返回 0。

参数：

message 发送给父窗口的 Windows 消息。

wParam 与消息相关的 wparam。

lParam 与消息相关的 lparam。

pLResult 指向父窗口过程返回值的指针。如果没有返回值，则这个指针可以是 NULL。

说明：

当窗口的父窗口接收到这个窗口有关的通知消息时，就调用这个成员函数。

永远不要直接调用这个成员函数。

这个成员函数的缺省实现返回 0，这意味着父窗口将处理消息。重载这个成员函数以扩展响应通知消息的方式。

OnClose

说明：

框架调用这个函数，作为 CWnd 或应用程序要被关闭的信号。缺省的实现调用 DestroyWindow。

OnCommand

返回值：如果应用程序要处理这个消息，则返回非零值；否则返回 0。

参数：

wParam　wParam 的低位字标识了菜单项或控件的命令 ID。如果消息是控件发出的，则 wParam 的高位字标识了通知消息。如果消息是加速键发出的，则高位字为 1。如果消息是菜单发出的，则高位字为 0。

lParam　如果消息是控件发出的，则标识了发出消息的控件；否则 lParam 为 0。

说明：

当用户从菜单中选择了一项，或者子控件发出一个通知消息，或者转换了一个加速键的击键事件，框架就调用这个成员函数。

OnCommand 处理控件通知和 ON_COMMAND 入口的消息映射，并调用相应的成员函数。

在你的派生类中重载这个函数以处理 WM_COMMAND 消息。除非调用了基类的 OnCommand，否则不应处理消息映射。

注意：

框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnCreate

返回值：OnCreate 必须返回 0 以继续 CWnd 对象的创建过程。如果应用程序返回-1，窗口将被销毁。

参数：

lpCreateStruct　　指向一个 CREATESTRUCT 结构，其中包含了与要创建的 CWnd 对象有关的信息。

说明：

当应用程序通过调用成员函数 Create 或 CreateEx 请求创建 Windows 的窗口时，框架调用这个成员函数。CWnd 对象在窗口被创建以后，但是在它变为可见之前接收到对这个函数的调用。OnCreate 是在 Create 或 CreateEx 成员函数返回之前被调用的。

重载这个成员函数以执行派生类所需的初始化工作。

CREATESTRUCT 结构中包含了用于创建窗口的参数的拷贝。

注意：

框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnDestroy

返回值：

框架调用这个成员函数以通知 CWnd 对象它将被销毁。OnDestroy 是在 CWnd 对象已经从屏幕上清除以后被调用的。

首先为被销毁的 CWnd 调用 OnDestroy，然后当 CWnd 的子窗口被销毁时为它们调用 OnDestroy。可以假定当 OnDestroy 运行的时候，所有的子窗口依然存在。

如果被销毁的 CWnd 对象是剪贴板观察器链（通过调用 SetClipboardViewer 成员函数设置）的一部分，CWnd 必须在从 OnDestroy 函数返回之前调用 ChangeClipboardChain 成员函数，将自己从剪贴板观察器链中清除。

OnEnable

参数：

bEnable　指定了 CWnd 对象是被允许的还是被禁止的。如果 CWnd 是允许的，则这个参数为 TRUE；如果 CWnd 是被禁止的，则为 FALSE。

说明：

当应用程序改变 CWnd 对象的允许状态时，框架调用这个成员函数。OnEnable 在 EnableWindow 成员函数返回之前被调用，但是是在窗口的允许状态（WS_DISABLE 风格位）改变之后。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnKeyDown

参数：

nChar　指定了给定键的虚拟键码。

nRepCnt　重复计数（用户按住键引起的重复击键数目）。

nFlags　指定了扫描码、暂态键码、原来的键状态和上下文代码，如下面的列表所示：

值　描述

0-7　扫描码（依赖于 OEM 的值）

8　扩展键，比如功能键或数字键盘上的键（如果它是扩展键，则为 1）

9-10　未使用

11-12　Windows 内部使用

13　上下文代码（如果按下键时 ALT 键时被按下的，则为 1；否则为 0）

14　原来的键状态（如果在调用之前键时按下的，则为 1；如果键是弹起的，则为 0）

15　暂态（如果键正在被释放，则为 1；如果键正被按下，则为 0）

对于 WM_KEYDOWN 消息，键暂态位（15 位）为 0，并且上下文代码位（13 位）为 0。

说明：

当用户按下了一个非系统键时，框架调用这个成员函数。非系统键是指当 ALT 键为被按下时按下的键盘键或者当 CWnd 拥有输入焦点时按下的键盘键。

由于自动重复，在调用 OnKeyUp 成员函数之前可能会产生多个 OnKeyDown 调用。指明原来的键状态的位可以被用来确定 OnKeyDown 调用时是第一次被按下还是重复的按下状态。

对于 IBM 增强 101 和 102 键键盘，增强键包括键盘主体部分的右 ALT 键和右 CTRL 键；数字键盘左侧的 INS，DEL，HOME，END，PAGE UP，PAGE DOWN 和箭头键；以及数字键盘上的斜杠（/）和 ENTER 键。一些其它的键盘可能支持 nFlags 中的扩展键位。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnKeyUp

参数：

nChar 指定了给定键的虚拟键码。

nRepCnt 重复计数（用户按住键引起的重复击键数目）。

nFlags 指定了扫描码、暂态键码、原来的键状态和上下文代码，如下面的列表所示：

值 描述

0-7 扫描码（依赖于 OEM 的值）。高位字的低字节

8 扩展键，比如功能键或数字键盘上的键（如果它时扩展键则为 1）

9-10 未使用

11-12 Windows 内部使用

13 上下文代码（如果按下键时 ALT 键时被按下的，则为 1；否则为 0）

14 原来的键状态（如果在调用之前键时按下的，则为 1；如果键是弹起的，则为 0）

15 暂态（如果键正在被释放，则为 1；如果键正被按下，则为 0）

对于 WM_KEYDOWN 消息，键暂态位（15 位）为 1，并且上下文代码位（13 位）为 0。

说明：

当一个非系统键被释放的时候，框架调用这个成员函数。非系统键是指当 ALT 键未按下时按下的键盘键，或者是当 CWnd 拥有输入焦点时按下的键盘键。

对于 IBM 增强 101 和 102 键键盘，增强键包括键盘主体部分的右 ALT 键和右 CTRL 键；数字键盘左侧的 INS，DEL，HOME，END，PAGE UP，PAGE DOWN 和箭头键；以及数字键盘上的斜杠（/）和 ENTER 键。一些其它的键盘可能支持 nFlags 中的扩展键位。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnKillFocus

参数：

pNewWnd 指定了获得输入焦点的窗口指针（可能为 NULL，或可能是临时的）。

说明：

框架在失去输入焦点之后立刻调用这个成员函数。

如果 CWnd 显示了一个插字符，则此时必须销毁插字符。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnLButtonDblClk

参数：

nFlags 指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

· MK_CONTROL 如果 CTRL 键被按下，则设置此位。

· MK_LBUTTON 如果鼠标左键被按下，则设置此位。

· MK_MBUTTON 如果鼠标中键被按下，则设置此位。

· MK_RBUTTON 如果鼠标右键被按下，则设置此位。

· MK_SHIFT 如果 SHIFT 键被按下，则设置此位。point 指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。说明当用户双击鼠标左键时框架调用这个成员函数。只有具有 CS_DBLCLKS WNDCLASS 风格的窗口才接收 OnLButtonDlbClk 调

point 指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明：

当用户双击鼠标左键时框架调用这个成员函数。

只有具有 CS_DBLCLKS WNDCLASS 风格的窗口才接收 OnLButtonDlbClk 调用。这是微软基础类窗口的缺省状态。当用户按下、释放，然后在系统规定的双击时间限制之内再次按下鼠标左键时，Windows 就调用 OnLButtonDblClk。双击鼠标左键实际产生四个事件：WM_LBUTTONDOWN，WM_LBUTTONUP 消息，WM_LBUTTONDBLCLK 调用，以及释放按钮时的另一个 WM_LBUTTONUP 消息。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnLButtonDown

参数：

nFlags　指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL　如果 CTRL 键被按下，则设置此位。
- MK_LBUTTON　如果鼠标左键被按下，则设置此位。
- MK_MBUTTON　如果鼠标中键被按下，则设置此位。
- MK_RBUTTON　如果鼠标右键被按下，则设置此位。
- MK_SHIFT　如果 SHIFT 键被按下，则设置此位。

point　指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明：当用户按下鼠标左键时，框架调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

afx_msg void OnLButtonUp(UINT nFlags, CPoint point);

nFlags　指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL　如果 CTRL 键被按下，则设置此位。
- MK_LBUTTON　如果鼠标左键被按下，则设置此位。
- MK_MBUTTON　如果鼠标中键被按下，则设置此位。
- MK_RBUTTON　如果鼠标右键被按下，则设置此位。
- MK_SHIFT　如果 SHIFT 键被按下，则设置此位。

point　　指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明：

当用户放开鼠标左键时，框架调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnMButtonDblClk

参数：

nFlags　指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL　如果 CTRL 键被按下，则设置此位。
- MK_LBUTTON　如果鼠标左键被按下，则设置此位。
- MK_MBUTTON　如果鼠标中键被按下，则设置此位。
- MK_RBUTTON　如果鼠标右键被按下，则设置此位。
- MK_SHIFT　如果 SHIFT 键被按下，则设置此位。

point　　指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明：

当用户双击鼠标中键时框架调用这个成员函数。

只有具有 CS_DBLCLKS WNDCLASS 风格的窗口才接收 OnMButtonDlbClk 调用。这是微软基础类窗口的缺省状态。当用户按下、释放，然后在系统规定的双击时间限制之内再次按下鼠标中键时，Windows 就调用 OnMButtonDblClk。双击鼠标中键实际产生四个事件：WM_MBUTTONDOWN，WM_MBUTTONUP 消息，WM_MBUTTONDBLCLK 调用，以及释放按钮时的另一个 WM_MBUTTONUP 消息。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnMButtonDown

参数：

nFlags　指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL　如果 CTRL 键被按下，则设置此位。
- MK_LBUTTON　如果鼠标左键被按下，则设置此位。
- MK_MBUTTON　如果鼠标中键被按下，则设置此位。
- MK_RBUTTON　如果鼠标右键被按下，则设置此位。
- MK_SHIFT　如果 SHIFT 键被按下，则设置此位。

point　指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明：

当用户按下鼠标中键时，框架调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnMButtonUp

nFlags　指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL　如果 CTRL 键被按下，则设置此位。
- MK_LBUTTON　如果鼠标左键被按下，则设置此位。
- MK_MBUTTON　如果鼠标中键被按下，则设置此位。
- MK_RBUTTON　如果鼠标右键被按下，则设置此位。
- MK_SHIFT　如果 SHIFT 键被按下，则设置此位。

point　指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明：

当用户放开鼠标中键时，框架调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnMouseActivate;

返回值：

指定了是否要激活 CWnd 以及是否放弃鼠标事件。它必须是下列值之一：

- MA_ACTIVATE 激活 CWnd 对象。
- MA_NOACTIVATE 不激活 CWnd 对象 。
- MA_ACTIVATEANDEAT 激活 CWnd 对象并放弃鼠标事件。
- MA_NOACTIVATEANDEAT 不激活 CWnd 对象并放弃鼠标事件。

参数：

pDesktopWnd 指定了要激活的窗口的顶层父窗口的指针。这个指针可能是临时的，不能被保存。

nHitTest 指定了击中测试区域代码。击中测试是用来确定光标的位置的。

message 指定了鼠标消息。

说明：

当光标位于非激活窗口内并且用户按下了鼠标按钮时，框架就调用这个成员函数。

缺省的实现在进行任何处理之前把这个消息传递给父窗口。如果父窗口返回 TRUE，则处理过程中止。

有关不同的击中测试区域代码的描述参见 OnNcHitTest 成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnMouseMove

参数：

nFlags　指明是否按下了不同的虚拟键。这个参数可以是下列值的组合：

MK_CONTROL　如果 CTRL 键被按下，则设置此位。

MK_LBUTTON　如果鼠标左键被按下，则设置此位。

MK_MBUTTON　如果鼠标中键被按下，则设置此位。

MK_RBUTTON　如果鼠标右键被按下，则设置此位。

MK_SHIFT　如果 SHIFT 键被按下，则设置此位。

point　指定了光标的 x 和 y 轴坐标。这些坐标常是相对于窗口的左上角的。说明当鼠标光标移动时，框架调用这个成员函数。如果鼠标没有被捕获，则 WM_MOUSEMOVE 由鼠标下方的窗口所接收；否则消息被发往捕获了鼠标的窗口。

注意 框架调用这个成员函数以允许你的应用程序处通理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）

OnMouseWheel

返回值：如果允许鼠标轮滚动，则返回非零值；否则返回 0。

参数：

nFlags　指明是否按下了虚拟键。这个参数可以是下列值的组合：

- MK_CONTROL　如果 CTRL 键被按下，则设置此位。
- MK_LBUTTON　如果鼠标左键被按下，则设置此位。
- MK_MBUTTON　如果鼠标中键被按下，则设置此位。
- MK_RBUTTON　如果鼠标右键被按下，则设置此位。
- MK_SHIFT　如果 SHIFT 键被按下，则设置此位。

zDelta　指明了旋转的距离。zDelta 值以 WHEEL_DELTA，即 120 的倍数或部分的形式表达。小于零的数表明往回滚动（向着用户），而大于零的数表明滚向远处（离开用户）。用户可以在鼠标软件中改变滚轮设置以反转这种响应。有关这个参数的更多信息参见说明部分。

pt　指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明：

当用户旋转鼠标滚轮并达到滚轮的下一个刻度时，框架就调用这个成员函数。除非被重载，否则 OnMouseWheel 调用 WM_MOUSEWHEEL 的缺省处理。Windows 自动将该消息转

发到具有输入焦点的控件或子窗口。Win32 函数 DefWindowProc 将该消息上传到拥有它的窗口。

zDelta 参数是 WHEEL_DELTA 的倍数，它被设为 120。这个值是要采取的动作的开端，这一类动作（比如向前滚动到下一个刻度）必须为每一个 delta 产生。

delta 被设为 120，以允许将来使用更高精度的滚轮，例如没有刻度的自由旋转滚轮。这种设备在每次旋转是可能会发送多个消息，但是每次消息中的值更小。要支持这个可能性，或者可以累计输入的 delta 值，直到达到一个 WHEEL_DELTA（因此你达到与给定 delta 的旋转相同的响应），或者滚动部分行以响应更频繁的消息。你可以选择你的滚动精度并累计 delta 值直到达到 WHEEL_DELTA。

重载这个成员函数以提供你自己的鼠标滚轮滚动特性。

注意 OnMouseWheel 为 Windows NT 4.0 处理消息。对于 Windows 95 或 Windows NT 3.51 的消息处理，应使用 OnRegisteredMouseWheel。

OnMove

参数：

x 指定了客户区左上角的新 x 轴坐标。对于重叠式和弹出式窗口，坐标是用屏幕坐标给出的，对于子窗口，是用父窗口的客户区坐标给出的。

y 指定了客户区左上角的新 y 轴坐标。对于重叠式和弹出式窗口，坐标是用屏幕坐标给出的，对于子窗口，是用父窗口的客户区坐标给出的。

说明：

框架在 CWnd 对象被移动之后调用这个成员函数。

注意：

框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnMoving

参数：

nSide 被移动的窗口的边界。

lpRect　CRect 或 RECT 的地址，其中包含了项的坐标。

说明：

当用户移动 CWnd 对象时框架调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnNotify

返回值：如果应用程序处理这个消息，则返回非零值；否则返回 0。

参数：

wParam　如果消息是从控件发出的，则标识了发出消息的控件；否则 wParam 为 0。

lParam　指向通知消息（NMHDR）结构的指针，其中包含了通知消息代码和附加信息。对于某些通知消息，这个指针指向一个更大的结构，NMHDR 是其第一个参数。

pResult　指向 LRESULT 变量的指针，如果消息被处理，则用它来保存返回代码。

说明：

框架调用这个函数以通知控件的父窗口，在控件中发生了一个事件，或者该控件需要某些类型的信息。

OnNotify 处理控件通知的消息映射。在你的派生类中重载这个成员函数以处理 WM_NOTIFY 消息。重载函数不会处理消息映射，除非调用了基类的 OnNotify。

有关 WM_NOTIFY 消息的更多信息参见“TN061：ON_NOTIFY 和 WM_NOTIFY 消息”。可能你会对“TN060：新的 Windows 公共控件”和“TN062：Windows 控件的消息反射”中描述的相关主题感兴趣。所有这些都在 Visual C++联机文档中。

OnPaint

说明：

当 Windows 或应用程序请求重画应用程序窗口的一部分时，框架调用这个成员函数。WM_PAINT 在调用 UpdateWindow 或 RedrawWindow 成员函数时发出。当设置了 RDW_INTERNALPAINT 标志并调用 RedrawWindow 成员函数时，窗口可能会接收到内部重画消息。在这种情况下，窗口可能没有更新区域。应用程序必须调用 GetUpdateRect 成员函

数以确定窗口是否具有更新区域。如果 GetUpdateRect 返回 0，则应用程序不应调用 BeginPaint 和 EndPaint 成员函数。

应用程序负责检查是否需要内部重画或更新，这可通过查看每条 WM_PAINT 消息的内部数据结构来完成，因为一条 WM_PAINT 可能是由于一个无效区域或由于使用 RDW_INTERNALPAINT 标志调用了 RedrawWindow 成员函数而引起的。

Windows 只发送一次内部 WM_PAINT 消息。在通过 UpdateWindow 成员函数向窗口发送了内部 WM_PAINT 消息以后，将不会再向窗口发送其它 WM_PAINT 消息，直到再次使用 RDW_INTERNALPAINT 标志调用了 RedrawWindow 成员函数。

有关在文档/视应用程序中描绘图象的信息参见 CView::OnDraw。

有关使用 WM_PAINT 的更多信息参见《Win32 SDK 程序员参考》中的下列主题：

·“WM_PAINT 消息”

·“使用 WM_PAINT 消息”

OnRButtonDblClk

参数：

nFlags　指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL　如果 CTRL 键被按下，则设置此位。
- MK_LBUTTON　如果鼠标左键被按下，则设置此位。
- MK_MBUTTON　如果鼠标中键被按下，则设置此位。
- MK_RBUTTON　如果鼠标右键被按下，则设置此位。
- MK_SHIFT　如果 SHIFT 键被按下，则设置此位。

point　指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明：

当用户双击鼠标右键时框架调用这个成员函数。

只有具有 CS_DBLCLKS WNDCLASS 风格的窗口才接收 OnRButtonDlbClk 调用。这是微软基础类窗口的缺省状态。当用户按下、释放，然后在系统规定的双击时间限制之内再次按下鼠标右键时，Windows 就调用 OnRButtonDblClk。双击鼠标右键实际产生四个事件：WM_RBUTTONDOWN，WM_RBUTTONUP 消息，WM_RBUTTONDBLCLK 调用，以及释放按钮时的另一个 WM_RBUTTONUP 消息。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnRButtonDown

参数：

nFlags 指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL 如果 CTRL 键被按下，则设置此位。
- MK_LBUTTON 如果鼠标左键被按下，则设置此位。
- MK_MBUTTON 如果鼠标中键被按下，则设置此位。
- MK_RBUTTON 如果鼠标右键被按下，则设置此位。
- MK_SHIFT 如果 SHIFT 键被按下，则设置此位。

point 指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明：

当用户按下鼠标右键时框架调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnRButtonUp

参数：

nFlags 指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL 如果 CTRL 键被按下，则设置此位。
- MK_LBUTTON 如果鼠标左键被按下，则设置此位。
- MK_MBUTTON 如果鼠标中键被按下，则设置此位。
- MK_RBUTTON 如果鼠标右键被按下，则设置此位。
- MK_SHIFT 如果 SHIFT 键被按下，则设置此位。

point 指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明：

当用户放开鼠标右键时框架调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnSetCursor

返回值：如果要停止进一步处理，则返回非零值；如果要继续，则返回 0。

参数：

pWnd 指定了包含光标的窗口指针。这个指针可能是临时的，不能被保存以供将来使用。

nHitTest 指定了击中测试区域代码。击中测试确定了光标的位置。

message 指定了鼠标消息。

说明：

如果鼠标输入没有被捕获并且鼠标使光标在 CWnd 对象内移动，则框架调用这个成员函数。

缺省的实现在处理之前调用父窗口的 OnSetCursor。如果父窗口返回 TRUE，则将停止进一步处理。调用父窗口使父窗口能够控制子窗口中光标的设置。

如果光标不在客户区内，缺省的实现将光标设为箭头；如果是在客户区内，则将光标设为注册的类光标。

如果 nHitTest 为 HTERROR 并且该消息是一个鼠标键按下消息，则将调用 MessageBeep 成员函数。

当 CWnd 进入菜单模式时，消息参数为 0。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnSetFocus

参数：

pOldWnd　包含了失去输入焦点的 CWnd 对象（可能为 NULL）。这个指针可能是临时的，不能被保存以供将来使用。

说明：

框架在获得输入焦点以后调用这个成员函数。如果要显示插字符，CWnd 必须在此时调用适当的插字符函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnShowWindow

参数：

bShow　指定窗口是否要被显示。如果窗口要被显示，则为 TRUE；如果窗口要被隐藏，则为 FALSE。

nStatus　指定了要显示的窗口的状态。如果是因为调用 ShowWindow 成员函数而发出的消息，则为 0；否则 nStatus 为下列值之一：

- SW_PARENTCLOSING　父窗口正被关闭（变为图标）或弹出式窗口正被隐藏。
- SW_PARENTOPENING　父窗口正被打开（被显示）或弹出式窗口正被显示。 f

说明：

当 CWnd 对象要被显示或隐藏时，框架调用这个成员函数。当调用 ShowWindow 成员函数时，或者重叠窗口被最大化或复原，或者重叠式或弹出式窗口被关闭（变为图标）或打开（被显示）时，窗口被显示或隐藏。当重叠窗口被关闭时，所有的与此窗口相关的所有弹出窗口都被隐藏。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnSize

参数：

nType　指定了要求的调整大小的类型。这个参数可以是下列值之一：

- SIZE_MAXIMIZED　窗口已经被最大化。

· SIZE_MINIMIZED 窗口已经被最小化。

· SIZE_RESTORED 窗口被改变了大小，但 SIZE_MINIMIZED 和 SIZE_MAXIMIZED 都不适用。

· SIZE_MAXHIDE 当其它窗口被最大化时，消息被发送到所有的弹出窗口。

· SIZE_MAXSHOW 当其它窗口被恢复到原来的大小时，消息被发送到所有的弹出窗口。

cx 指定了客户区域的新宽度。

cy 指定了客户区域的新高度。

说明：

框架在窗口的大小被改变以后调用这个成员函数。

如果在 OnSize 中为子窗口调用了 SetScrollPos 或 MoveWindow 成员函数，则 SetScrollPos 或 MoveWindow 的 bRedraw 参数必须为非零值，以使 CWnd 能被重画。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

OnTimer

参数：

nIDEvent 指定定时器的标识符。

说明：

当在 SetTimer 成员函数中指定的每一个时间间隔都被用来安装一个定时器之后，框架调用这个成员函数。

当在应用程序的消息队列中没有其它的消息时，Windows 函数 DispatchMessage 发送一个 WM_TIMER 消息。

注意 框架调用这个成员函数来使你的应用程序可以处理一个 Windows 消息。传递给你的函数的参数反映了在消息收到时框架收到的参数。如果你调用了这个函数的基类实现，这个实现将使用最初随着消息传递过来的参数，而不是使用你提供给函数的参数。

PostMessage

返回值：如果公布了消息，则返回非零值；否则返回 0。

参数：

message 指定了要公布的消息。

wParam 指定了附加的消息信息。这个参数的内容依赖于要公布的消息。

lParam 指定了附加的消息信息。这个参数的内容依赖于要公布的消息。

说明：

这个函数将一个消息放入窗口的消息队列，然后直接返回，并不等待对应的窗口处理消息。消息队列中的消息是通过调用 Windows 的 GetMessage 或 PeekMessage 函数来获得的。

可以通过 Windows 的 PostMessage 函数来访问其它应用程序。

RedrawWindow

返回值：如果窗口被成功地重画，则返回非零值；否则返回 0。

参数：

lpRectUpdate 指向一个 RECT 结构，其中包含了更新区域的坐标。如果 prgnUpdate 中包含了有效的区域句柄，则这个参数将被忽略。

prgnUpdate 表示了更新区域。如果 prgnUpdate 和 lpRectUpdate 都为 NULL，则整个客户区将被加入更新区域。

flags 下面的标志被用于使窗口无效：

RDW_ERASE 使窗口在重画时接收到一个 WM_ERASEBKGND 消息。必须同时指定 RDW_INVALIDATE 标志；否则 RDW_ERASE 标志将没有效果。

RDW_FRAME 使窗口非客户区中与更新区域重叠的任何部分接收到一条 WM_NCPAINT 消息。必须同时指定 RDW_INVALIDATE 标志，否则 RDW_FRAME 标志将没有效果。

RDW_INTERNALPAINT 使一条 WM_PAINT 消息被传递到窗口，而不管窗口是否包含一个无效区域。

RDW_INVALIDATE 使 lpRectUpdate 或 prgnUpdate（仅有一个可能为 NULL）无效。如果这个两个参数都为 NULL，则整个窗口都无效。

下面的标志被用于使窗口有效：

RDW_NOERASE 禁止任何未处理的 WM_ERASEBKGND 消息。

RDW_NOFRAME 禁止任何未处理的 WM_NCPAINT 消息。这个标志必须与 RDW_VALIDATE 一起使用，通常也与 RDW_NOCHILDREN 一起使用。这个选项必须小心使用，因为它可能会使窗口的某些部分不能正确地画出。

RDW_NOINTERNALPAINT 禁止任何未处理的内部 WM_PAINT 消息。这个标志不影响从无效区域产生的 WM_PAINT 消息。

RDW_VALIDATE 使 lpRectUpdate 或 prgnUpdate（仅有一个可能为 NULL）有效。如果这个两个参数都为 NULL，则整个窗口都有效。这个标志不影响内部 WM_PAINT 消息。

下面的标志控制着何时产生重画动作。除非指定了这些位，否则 RedrawWindow 函数不会执行绘图动作。

RDW_ERASENOW 如果有必要，则在函数返回前使涉及的窗口（如 RDW_ALLCHILDREN 和 RDW_NOCHILDREN 标志所指定的）接收到 WM_NCPAINT 和 WM_ERASEBKGND 消息。WM_PAINT 消息将被延缓。

RDW_UPDATENOW 如果有必要，则在函数返回前使涉及的窗口（如 RDW_ALLCHILDREN 和 RDW_NOCHILDREN 标志所指定的）接收到 WM_NCPAINT，WM_ERASEBKGND 和 WM_PAINT 消息。

在缺省情况下，RedrawWindow 函数影响的窗口依赖于指定的窗口是否具有 WS_CLIPCHILDREN 风格。WS_CLIPCHILDREN 窗口的子窗口不会被影响。但是，那些不具有 WS_CLIPCHILDREN 风格的窗口将被递归地有效或无效，直到遇见具有 WS_CLIPCHILDREN 风格的窗口。下面的标志控制着 RedrawWindow 函数将影响哪些窗口：

RDW_ALLCHILDREN 在重画操作中包含子窗口，如果有的话。

RDW_NOCHILDREN 在重画操作中不包括子窗口，如果有的化。

说明：

这个函数更新给定窗口的客户区中指定的矩形或区域。

当 RedrawWindow 成员函数被用于使捉摸窗口的部分无效的时候，该窗口不接收 WM_PAINT 消息。如果要重画桌面，应用程序必须使用 CWnd::ValidateRgn，CWnd::InvalidateRgn，CWnd::UpdateWindow 或::RedrawWindow

ReleaseDC

返回值：如果成功，则返回非零值；否则返回 0。

参数：

pDC　　标识了要释放的设备环境。

说明：

释放设备环境，以供其它应用程序使用。ReleaseDC 成员函数的效果依赖于设备环境的类型。

对于 GetWindowDC 和 GetDC 成员函数的每一次调用，应用程序都应当调用 ReleaseDC 成员函数。

SendMessage

返回值：消息处理的结果；它的值依赖于发送的消息。

参数：

message　指定了要发送的消息。

wParam　指定了与消息有关的附加信息。

lParam　指定了与消息有关的附加信息。

说明：

这个函数向窗口发送指定的消息。SendMessage 成员函数直接调用窗口过程并在窗口过程处理了消息以后才返回。这与 PostMessage 成员函数形成对比，该函数将消息放入窗口的消息队列并立即返回。

SendNotifyMessage

返回值：如果函数成功，则返回非零值；否则返回 0。

参数：

message　指定了要发送的消息。

wParam　指定了与消息有关的附加信息。

lParam　指定了与消息有关的附加信息。

说明：

这个函数向窗口发送指定的消息。如果窗口是由调用线程创建的，则 SendNotifyMessage 调用窗口的窗口过程，并在窗口处理了消息之后返回。如果窗口是由其它线程创建的，则 SendNotifyMessage 将消息传递给窗口过程并立即返回；它并不等待窗口过程结束处理消息。

SetActiveWindow

返回值：原来活动的窗口。返回的指针可能是临时的，不能被保存以供将来使用。

说明：

这个函数使 CWnd 成为活动窗口。

SetActiveWindow 成员函数必须小心使用，因为它允许应用程序任意地接管活动窗口和输入焦点。通常，Windows 管理着所有的活动。

SetCapture

返回值：

原来接收所有鼠标输入的窗口的指针。如果没有这样的窗口，则返回值为 NULL。返回的指针可能是临时的，不能被保存以供将来使用。

说明：

这个函数使随后的所有鼠标输入都被发送到当前的 CWnd 对象，并不考虑光标的位置。

当 CWnd 不再需要所有的鼠标输入时，应用程序应当调用 ReleaseCapture 函数以使其它窗口能够接收鼠标输入。

SetFocus

返回值：

原来拥有输入焦点的窗口对象的指针。如果没有这样的窗口，则返回值为 NULL。返回的指针可能是临时的，不应被保存。

说明：

这个函数要求得到输入焦点。输入焦点将随后的所有键盘输入定向到这个窗口。原来拥有输入焦点的任何窗口都将失去它。

SetFocus 成员函数项失去输入焦点的窗口发送一条 WM_KILLFOCUS 消息，并向接收输入焦点的窗口发送一条 WM_SETFOCUS 消息。它还激活该窗口或它的父窗口。

如果当前窗口是激活的，但是不具有输入焦点（这意味着，没有窗口具有输入焦点），则任何按下的键都将产生 WM_SYSCHAR，WM_SYSKEYDOWN 或 WM_SYSKEYUP 消息。

SetTimer

返回值：

如果函数成功，则返回新定时器的标识符。应用程序可以将这个值传递给 KillTimer 成员函数以销毁定时器。如果成功，则返回非零值；否则返回 0。

参数：

nIDEvent　　指定了不为零的定时器标识符。

nElapse　指定了定时值；以毫秒为单位。

lpfnTimer　　指定了应用程序提供的 TimerProc 回调函数的地址，该函数被用于处理 WM_TIMER 消息。如果这个参数为 NULL，则 WM_TIMER 消息被放入应用程序的消息队列并由 CWnd 对象来处理。

说明：

这个函数设置一个系统定时器。指定了一个定时值，每当发生超时，则系统就向设置定时器的应用程序的消息队列发送一个 WM_TIMER 消息，或者将消息传递给应用程序定义的 TimerProc 回调函数。

lpfnTimer 回调函数不需要被命名为 TimerProc，但是它必须按照如下方式定义：

```
void CALLBACK EXPORT TimerProc(
  HWND hWnd,      // 调用 SetTimer 的 CWnd 的句柄
  UINT nMsg,      // WM_TIMER
  UINT nIDEvent    // 定时器标识
  DWORD dwTime    // 系统时间
  );
```

定时器是有限的全局资源；因此对于应用程序来说，检查 SetTimer 返回的值以确定定时器是否可用是很重要的。

SetWindowPos

返回值：如果函数成功，则返回非零值；否则返回 0。

参数：

pWndInsertAfter　标识了在 Z 轴次序上位于这个 CWnd 对象之前的 CWnd 对象。这个参数可以是指向 CWnd 对象的指针，也可以是指向下列值的指针：

wndBottom　将窗口放在 Z 轴次序的底部。如果这个 CWnd 是一个顶层窗口，则窗口将失去它的顶层状态；系统将这个窗口放在其它所有窗口的底部。

wndTop　将窗口放在 Z 轴次序的顶部。

wndTopMost　将窗口放在所有非顶层窗口的上面。这个窗口将保持它的顶层位置，即使它失去了活动状态。

wndNoTopMost　将窗口重新定位到所有非顶层窗口的顶部（这意味着在所有的顶层窗口之下）。这个标志对那些已经是非顶层窗口的窗口没有作用。

有关这个函数以及这些参数的使用规则参见说明部分。

x　指定了窗口左边的新位置。

y　指定了窗口顶部的新位置。

cx　指定了窗口的新宽度。

cy　指定了窗口的新高度。

nFlags　指定了大小和位置选项。这个参数可以是下列值的组合：

SWP_DRAWFRAME　围绕窗口画出边框（在创建窗口的时候定义）。

SWP_FRAMECHANGED　向窗口发送一条 WM_NCCALCSIZE 消息，即使窗口的大小不会改变。如果没有指定这个标志，则仅当窗口的大小发生变化时才发送 WM_NCCALCSIZE 消息。

SWP_HIDEWINDOW　隐藏窗口。

SWP_NOACTIVATE　不激活窗口。如果没有设置这个标志，则窗口将被激活并移动到顶层或非顶层窗口组（依赖于 pWndInsertAfter 参数的设置）的顶部。

SWP_NOCOPYBITS　废弃这个客户区的内容。如果没有指定这个参数，则客户区的有效内容将被保存，并在窗口的大小或位置改变以后被拷贝回客户区。

SWP_NOMOVE　保持当前的位置（忽略 x 和 y 参数）。

SWP_NOOWNERZORDER　不改变拥有者窗口在 Z 轴次序上的位置。

SWP_NOREDRAW　不重画变化。如果设置了这个标志，则不发生任何种类的变化。这适用于客户区、非客户区（包括标题和滚动条）以及被移动窗口覆盖的父窗口的任何部分。当这个标志被设置的时候，应用程序必须明确地无效或重画要重画的窗口和父窗口的任何部分。

SWP_NOREPOSITION　　与 SWP_NOOWNERZORDER 相同。

SWP_NOSENDCHANGING　　防止窗口接收 WM_WINDOWPOSCHANGING 消息。

SWP_NOSIZE　　保持当前的大小（忽略 cx 和 cy 参数）。

SWP_NOZORDER　　保持当前的次序（忽略 pWndInsertAfter）。

SWP_SHOWWINDOW　　显示窗口。

说明：

调用这个成员函数以改变子窗口、弹出窗口和顶层窗口的大小、位置和 Z 轴次序。

窗口在屏幕上按照它们的 Z 轴次序排序。在 Z 轴次序上处于顶端的窗口将程序在所有其它窗口的顶部。

子窗口的所有坐标都是客户坐标（相对于父窗口客户区的左上角）。

窗口可以被移动到 Z 轴次序的顶部，既可以通过将 pWndInsertAfter 参数设为 &wndTopMost，并确保没有设置 SWP_NOZORDER 标志，也可以通过设置窗口的 Z 轴次序使它位于所有现存的顶层窗口上方。当一个非顶层窗口被设为顶层窗口时，它拥有的窗口也被设为顶层的。它的拥有者不发生变化。

如果顶层窗口被重新定位到 Z 轴次序的底部（&wndBottom）或任何非顶层窗口之后，则它将不再是顶层窗口。当顶层窗口被变为非顶层窗口时，它所有的拥有者和它拥有的所有窗口都被变为非顶层窗口。

如果既没有指定 SWP_NOACTIVE 标志也没有指定 SWP_NOZORDER 标志（这意味着应用程序要求窗口被同时激活并放入指定的 Z 轴次序），则 pWndInsertAfter 参数中指定的值将只在下列环境下适用：

- 在 pWndInsertAfter 参数中既没有指定&wndTopMost 也没有指定&wndNoTopMost。
- 这个窗口不是活动窗口。

应用程序不能激活一个非活动窗口但同时又不把它带到 Z 轴次序的顶部。应用程序可以没有任何限制地改变活动窗口的 Z 轴次序。

非顶层窗口可能拥有一个顶层窗口，但是反之则不成立。任何被顶层窗口拥有的窗口（例如对话框）都将自己变为顶层窗口，以确保所有被拥有的窗口位于它们的拥有者上方。

在 Windows 3.1 或更新的版本中，可以将窗口移动到 Z 轴次序的顶部，并通过设置它们的 WS_EX_TOPMOST 风格而将之锁定在那里。这种顶层窗口即使在失去活动状态以后也会保持顶层位置。例如，选择 WinHelp 的 Always On Top 命令会使帮助窗口变为顶层，并且在你返回应用程序之后它还保持可见。

要创建一个顶层窗口，应在调用 SetWindowPos 的时候将 pWndInsertAfter 参数设为 &wndTopMost，或者在创建窗口的时候设置 WS_EX_TOPMOST 风格。

如果 Z 轴次序中包含了任何具有 WS_EX_TOPMOST 风格的窗口，则用&wndTopMost 移动的窗口将被放到所有非顶层窗口的顶部，但是位于任何顶层窗口的下面。当应用程序激活一个不具有 WS_EX_TOPMOST 风格的非活动窗口时，该窗口将被移动到所有非顶层窗口的上方，但是位于所有顶层窗口的下方。

如果在调用 SetWindowPos 的时候 pWndInsertAfter 参数被设为&wndBottom，并且 CWnd 是一个顶层窗口，则该窗口失去顶层状态（WS_EX_BOTTOM 风格被清除），并且系统将窗口放在 Z 轴次序的底部。

SetWindowText

参数：

lpszString　　指向一个 CString 对象或以 null 结尾的字符串，将被用作新的标题或控件文本。

说明：

这个函数将窗口的标题设为指定的文本。如果窗口为一个控件，则将设置控件内的文本。

这个函数使一条 WM_SETTEXT 消息被发送到这个窗口。

ShowWindow

返回值：如果窗口原来可见，则返回非零值；如果 CWnd 原来是隐藏的，则返回 0。

参数：

nCmdShow　指定了 CWnd 应如何被显示。它必须是下列值之一：

SW_HIDE　隐藏窗口并将活动状态传递给其它窗口。

SW_MINIMIZE　最小化窗口并激活系统列表中的顶层窗口。

SW_RESTORE　激活并显示窗口。如果窗口是最小化或最大化的，Windows 恢复其原来的大小和位置。

SW_SHOW　激活窗口并以其当前的大小和位置显示。

SW_SHOWMAXIMIZED　激活窗口并显示为最大化窗口。

SW_SHOWMINIMIZED　激活窗口并显示为图标。

SW_SHOWMINNOACTIVE　将窗口显示为图标。当前活动的窗口将保持活动状态。

SW_SHOWNA　按照当前状态显示窗口。当前活动的窗口将保持活动状态。

SW_SHOWNOACTIVATE 按窗口最近的大小和位置显示。当前活动的窗口将保持活动状态。

SW_SHOWNORMAL 激活并显示窗口。如果窗口是最小化或最大化的，则 Windows 恢复它原来的大小和位置。

说明：

这个函数设置窗口的可视状态。

每个应用程序只应用CWinApp::m_nCmdShow为主窗口调用一次ShowWindow。以后调用ShowWindow应该用上面列出的值来代替CWinApp::m_nCmdShow指定的值。

UpdateData

返回值：

如果操作成功，则返回非零值；否则返回 0。如果 bSaveAndValidate 为 TRUE，则返回非零值意味着已成功地使数据有效。

参数：

bSaveAndValidate　指明是要初始化对话框（FALSE）还是获取数据（TRUE）的标志。

说明：

调用这个成员函数以初始化对话框中的数据，或者获得并检验对话框数据。

当一个模式对话框被创建时，框架自动在CDialog::OnInitDialog的缺省实现中调用UpdateData，bSaveAndValidate被设为FALSE。这个函数在对话框可见之前被调用。CDialog::OnOK的缺省实现令bSaveAndValidate为TRUE并调用这个成员函数以获得对话框中的数据，如果成功，将关闭对话框（如果在对话框中点击了Cancel按钮，则对话框将被关闭，并不获取数据）。

WindowProc

返回值：返回值依赖于消息。

参数：

message 指定了要处理的 Windows 消息。

wParam　提供了可用于消息处理的附加信息。这个参数的值与消息有关。

lParam　提供了可用于消息处理的附加信息。这个参数的值与消息有关。

说明：

这个函数为 CWnd 对象提供了 Windows 过程（WindowProc）。它通过窗口的消息映射分派消息。