**MFC的运行机制 以及 MFC中的DC、CDC、HDC、句柄、设备上下文 [转]**

在MFC程序中,我们并不经常直接调用Windows API,而是从MFC类创建对象并调用属于这些对象的成员函数.也就是说MFC封装了Windows API.你说你喜欢C++而MFC换一种说法就是一个用C++写的一个函数库 然后你来调用 只不过这个类不是你写的

MFC提供数百个类，最重要的、也是编写任何VC++应用程序都必不可少的两个类CWinApp和CFrameWnd,这两个类是编写复杂庞大应用程序的基石。

1>封装特性：构成MFC框架的是MFC类库而MFC类库又是C++的一个类库。这些类封装WIN32应用程序编程接口，OLE(Object Link Embed 对象链接嵌入)特性，ODBC和DAO数据访问的功能。

2>继承特性：MFC抽象出了众多类的共同特性，并设计出一些基类作为实现其他类的基础，这些类中最重要的类是CObject类和CCmdTarget类，程序员可以从适当的MFC类中派生出自己的类，实现特定的功能达到编程的目的。

3>虚拟和消息映射：MFC是以C++为基础，当然支持虚函数，但作为一个编程框架必须要解决的是效率问题：如果MFC仅仅通过虚函数来支持动态约束必然会产生大量的虚函数表这样编程框架过于臃肿而且消耗更多的内存。但是MFC建立了消息映射机制这样降低了内存的使用却大大提高了效率

消息映射是一个将消息和成员函数相互关联的表,当应用程序的框架窗口接收到一个消息时,MFC将搜索该窗口的消息映射,如果存在一个处理消息的处理程序,那么就调用该处理程序.

它通过宏来实现消息到成员函数的映射，而且这些函数不必是虚拟的成员函数，这样不需要为消息映射函数生成一个很大的虚拟函数表(V表)，节省内存。

MFC消息映射机制:

将消息与消息处理函数联系起来，形成一一对应的机制。

消息映射宏

声明：

1 DECLARE\_MESSAGE\_MAP

定义：

1 BEGIN\_MESSAGE\_MAP   
2   
3 ON\_COMMAND  
4   
5 ON\_CONTROL  
6   
7 ON\_MESSAGE   
8   
9 END\_MESSAGE\_MAP

MFC主要组成部分：类、宏和全局函数。

类是MFC中最主要的内容。MFC类是以层次结构方式组织起来的。MFC中的类分成两部分，除了一些辅助类，大多数的MFC类是直接或间接从根类CObject派生而来。

MFC宏主要功能：消息映射、运行时对象类型服务、诊断服务、异常处理。

MFC约定：全局函数以“Afx”为前缀，全局变量以“afx”为前缀

MFC类的层次关系

CObject项目类)->CCmdTarget(消息响应类)->

{

CWinThread(线程类)->CWinApp(Window应用程序类)

CDocument(文档类)

CWnd(窗体类)->[

              CFrameWnd(框架类)

              CView(视图类)

              ]

}

CObject类---由于MFC中大部分类是从CObject类继承而来的，CObject类描述了几乎所有的MFC类的一些公共特性，CObject类为程序员提供了对象诊断、运行时类型识别和序列化等功能。

CCmdTarget类---由CObject类直接派生而来，它负责将消息发送到能够响应这些消息的对象。它是所有能进行消息映射的MFC类的基类。

CWinApp类---在任何MFC应用程序中有且仅有一个CWinApp派生类的对象，它代表了程序中运行的主线程，也代表了应用程序本身。 CWinApp类取代了WinMain()主函数在SDK应用程序中的地位。传统SDK应用程序WinMain()函数完成的工作。现在由类CWinApp的InitApplication()，InitInstance()和Run()三个成员函数承担。

CWnd类---由CCmdTarget类直接派生而来，该类及其派生类的实例是一个窗口。CWnd类代表了MFC中最基本的GUI对象，它是一个功能最完善、成员函数最多的MFC类。

CFrameWnd类---是CWnd类的派生类，主要用来掌管一个窗口，它取代了SDK应用程序中窗口函数WndProc()的地位。CFrameWnd类的对象是一个框架窗口，包括边框、标题栏、菜单、最大化按钮、最小化按钮和一个激活的视图。

CDocument类---在应用程序中作为用户文档类的基类，它代表了用户存储或打开的一个文件。

CView类---是MFC中一个很基本的类，它作为其它MFC视图类和用户视图派生类的基类。

从API编程到MFC编程的过渡：

WinMain()

{  初始化WNDCLASS

    注册窗体结构

    创建窗口           ->>>>>>>>应用程序类CWinApp

    显示窗口

    消息循环

}

WndProc()

{ switch(…)

           ->>>>>>>>>框架窗口类CFrameWnd

}

MFC Object和Windows Object的对应关系：

|  |  |  |
| --- | --- | --- |
| 描述 | Windows句柄 | MFC Object |
| 窗口 | HWND | CWnd |
| 设备上下文 | HDC | CDC |
| 菜单 | HMENU | CMenu |
| 笔 | HPEN | CPen |
| 刷子 | HBRUSH | CBrush |
| 字体 | HFONT | CFont |
| 位图 | HBITMAP | CBitmap |
| 套接字 | SOCKET | CSocket |

显示简略信息

三、手工创建一个MFC应用程序：

注意：创建MFC程序,要创建一个Win32空项目,并要选择项目属性中的”在共享DLL文件中使用MFC,然后新建我们的文件

例子：在”hello.h”头文件中添写如下代码：

1 class CMyApp:public CWinApp  
 2   
 3 {  
 4   
 5 public:  
 6   
 7 virtual BOOL InitInstance(); //虚函数  
 8   
 9 };  
10   
11 class CMainWindow:public CFrameWnd  
12   
13 {  
14   
15 public:  
16   
17 CMainWindow();  
18   
19 protected:  
20   
21 afx\_msg void OnPaint();  
22   
23 DECLARE\_MESSAGE\_MAP(); //声明消息映射  
24   
25 };

在”hello.cpp”源文件中添写如下代码：

1 #include <afxwin.h>  
 2   
 3 #include “hello.h"  
 4   
 5 CMyApp myApp;  
 6   
 7 BOOL CMyApp::InitInstance()  
 8   
 9 {  
10   
11 m\_pMainWnd = new CMainWindow;  
12   
13   
14   
15 m\_pMainWnd->ShowWindow(m\_nCmdShow);  
16   
17 m\_pMainWnd->UpdateWindow();  
18   
19 return TRUE;  
20   
21 }  
22   
23 BEGIN\_MESSAGE\_MAP(CMainWindow,CFrameWnd)  
24   
25 ON\_WM\_PAINT()  
26   
27 END\_MESSAGE\_MAP() //消息映射  
28   
29 CMainWindow::CMainWindow() //构造函数初始化  
30   
31 {  
32   
33 Create(NULL,“我的第一个MFC应用程序”); //创建窗体  
34   
35 }  
36   
37 void CMainWindow::OnPaint()  
38   
39 { CPaintDC dc(this);  
40   
41 CRect rect;  
42   
43 GetClientRect(&rect);  
44   
45 dc.DrawText("Hello MFC",-1,&rect, DT\_SINGLELINE|DT\_CENTER|DT\_VCENTER);  
46   
47 }

CWinApp是应用程序类，在MFC应用程序中必须从这个类派生出一个类，该派生类是MFC应用程序的入口

必须定义这个派生类的对象,并且只能有一个这个对象代表整个应用程序。

成员函数：InitInstance()

 功能：初始化应用程序实例和窗口实例，

 虚函数CWinApp::InitInstance必须在派生类中重写。在InitInstance函数中，编写初始化代码，如：

 创建一个窗口

 显示一个窗口

CFrameWnd类

作用：为应用程序提供一个窗口，同时实现消息处理功能。

成员函数： Create()

功能:创建窗体,将之赋于CFrameWnd对象上。

BOOL Create(窗口类型, 窗口标题，显示风格，显示区域，父窗口句柄，菜单，扩展显示风格，上下文对象)共有8个参数，前两个必须给出，后6个可以默认。

MFC应用程序的核心就是基于CWinApp类的应用程序对象,CWinApp提供了消息循环来检索消息并将消息调度给应用程序的窗口.我们在编写MFC应用程序时,要包含afxwin.h,

一个MFC应用程序可以有且仅有一个应用程序对象,对象必须声明为在全局范围内有效(也就是全局对象),以便它在程序开始时即在内存中被实例化

我们的Hello MFC的应用程序类被命名为CMyApp,它在hello.cpp中用如下语句进行了实例化：

CMyApp myApp;

CMyApp的类声明在hello.h中代码如下：

1 class CMyApp:public CWinApp  
2   
3 {  
4   
5 public:  
6   
7 virtual BOOL InitInstance();  
8   
9 };

CMyApp没有声明任何的数据成员,只是重写了一个从CWinApp类中继承来的函数,在应用程序的生存期内InitInstance的调用比较早,是在应用程序开始运行以后而窗口创建之前,除非InitIstance创建一个窗口,否则应用程序是不会有窗口,这正是为什么即使最小的MFC应用程序也必须从CWinApp派生出一个类并重写CWinApp::InitIstance的原因

InitInstance函数:CWinApp::InitInstance是一个虚函数,其默认操作仅包含一条语句：return TRUE;

InitInstance是用来执行程序每次开始时都需要进行的初始化工作最好的地方

在hello.cpp中,CMyApp的InitInstance通过实例化hello的CMainWindow类来创建hello窗口,语句：

m\_pMainWnd = new CMainWindow;

构造了一个CMainWindow对象指针,并将其地址复制到了应用程序对象的m\_pMainWnd数据成员中,窗口创建以后,InitInstance就会通过CMainWindow指针调用ShowWindow和UpdateWindow函数显示它:

m\_pMainWnd->ShowWindow(m\_nCmdShow);

m\_pMainWnd->UpdateWindow();

ShowWindow和UpdateWindow是所有窗口对象共用的CWnd成员函数其中包括CFrameWnd类的对象,CMainWindow就是从CFrameWnd派生出来的.

要从MFC程序调用一个常规的Windows API函数,需要在函数名称前添加一个全局运算符:: 例如：::UpdateWindow();

通过生成窗口对象并调用其Create函数,MFC应用程序可以创建一个窗口,在CMyApp::InitInstance中,hello创建了一个CMainWindow对象,CMainWindow的构造函数生成在屏幕上看到的窗口：

Create(NULL,”我的第一个MFC应用程序”);

CPaintDC dc(this);

MFC的CPaintDC类是从MFC的CDC类派生的,CDC类封装了Windows设备环境,以及包含了绘制到屏幕、打印机和其他设备的几十个成员函数

在MFC中如何处理消息呢?

在SDK中我们利用的是消息循环和窗口过程函数对消息进行消息处理.

在MFC中我们用的是消息映射机制.

下面是将消息映射添加到一个类中需要做的全部工作.

1>通过将DECLARE\_MESSAGE\_MAP语句添加到类声明中,声明消息映射.

2>通过放置标识消息的宏来执行消息映射,相应的类将在对BEGIN\_MESSAGE\_MAP和END\_MESSAGE\_MAP的调用之间处理消息

3>添加成员函数来处理消息

1、构造CWinApp派生类的对象

2、系统调用WinMain()

3、WinMain调用InitInstance,在该函数中创建CFrameWnd派生类对象，调用Create函数创建窗口、调用ShowWindow函数显示窗口。

4、之后内部机制调用Run，接受用户的消息，并将消息导向默认的处理函数。当接收到WM\_QUIT消息时，Run内部调用ExitInstance,退出程序。

MFC采用消息映射（Message Map）机制取代C/C++语言中的switch-case结构来处理消息。

消息映射：在MFC中把消息处理函数和它所要处理的特定的消息连接起来的一种机制。

它通过宏来实现消息到成员函数的映射，而且这些函数不必是虚拟的成员函数，这样不需要为消息映射函数生成一个很大的虚拟函数表(V表)，节省内存。

MFC消息映射机制包括一组消息映射宏。一条消息映射宏把一个Windows消息和其消息处理函数联结起来。

MFC应用程序框架提供了消息映射功能。

在类的实现源文件中用BEGIN\_MESSAGE\_MAP()和END\_MESSAGE\_MAP()宏来定义消息映射。

在类定义的结尾用DECLARE\_MESSAGE\_MAP()宏来声明使用消息映射。

Hello的CmainWindow类只处理一种消息类型—WM\_PAINT,因此其消息映射的实现如下所示：

1 BEGIN\_MESSAGE\_MAP(CMainWindow,CFrameWnd);  
2   
3 ON\_WM\_PAINT()  
4   
5 END\_MESSAGE\_MAP()

BEGIN\_MESSAGE\_MAP开始了消息映射,并标识了消息映射所属的类和该类的基类

END\_MESSAGE\_MAP()结束消息映射.

ON\_WM\_PAINT()在BEGIN\_MESSAGE\_MAP和END\_MESSAGE\_MAP()之间,称做消息条目,在MFC为100多种Window消息提供了宏.

afx\_msg void OnPaint();

DECLARE\_MESSAGE\_MAP()

afx\_msg 醒目地暗示OnPaint是一个消息处理程序,

DECLARE\_MESSAGE\_MAP()声明消息映射

MFC把消息主要分为三大类：

(1)、标准Windows消息（WM\_XXX）

使用宏：ON\_WM\_XXX()          特点：有默认的消息处理函数

(2)、命令消息：（WM\_COMMAND）

来自于菜单、工具条、按钮等的消息

使用宏： ON\_COMMAND(命令按钮标识符ID，消息处理函数)

特点：由用户指定消息处理函数

3、”Notification消息” （通知消息） 由控件产生：

BOOL    布尔值，取值为TRUE或者FALSE

BSTR     32为字符指针

BYTE      8位整数无符号的

COLORREF     32位数值代表一个颜色值

DWORD    32位整数无符号的

LONG     32位整数带符号的

LPCTSTR   32位指针，指向一个常字符串

LPVOID    32位指针，指向一个为指定类型的数据

MFC特有的数据类型：

1>POSITION :一个数值，代表数组或者链表中元素的位置，在MFC中常用于数据处理类

2>LPRECT：32位指针，指向一个不变的矩形区域结构

**关于CDC(设备上下文) HDC**

可以简单的如此理解：

DC(DEVICE CONTEXT)，就是设备描述表（或者叫做设备环境），就像你作画，需要用到画笔，画刷，颜料等等，这些东西就是作画的环境，属于设备DC的属性所有，想作图就先把作图环境准备一下，这跟我们作画是一样的道理。

CDC和HDC简单使用：

CDC是设备上下文类;

HDC是设备上下文句柄;

1 //CDC－－>HDC与HDC－>CDC  
 2   
 3 CDC dc;  
 4   
 5 HDC hdc;  
 6   
 7 hdc=dc.m\_hDC;  
 8   
 9 dc=CDC::FromHandle(hdc);  
10   
11 // or hdc=dc.GetSafeHdc();

挂一段正规文档（来源MSDN)：

设备上下文

设备上下文是一种包含有关某个设备（如显示器或打印机）的绘制属性信息的 Windows 数据结构。所有绘制调用都通过设备上下文对象进行，这些对象封装了用于绘制线条、形状和文本的 Windows API。设备上下文允许在 Windows 中进行与设备无关的绘制。设备上下文可用于绘制到屏幕、打印机或者图元文件。

CPaintDC 对象---将 Windows 的常见固定用语进行封装，调用 BeginPaint 函数，然后在设备上下文中绘制，最后调用 EndPaint 函数。CPaintDC 构造函数为您调用 BeginPaint，析构函数则调用 EndPaint。该简化过程将创建 CDC 对象、绘制和销毁 CDC 对象。在框架中，甚至连这个过程的大部分也是自动的。具体说来，框架给 OnDraw 函数传递（通过 OnPrepareDC）准备好的 CPaintDC，您只需绘制到 CPaintDC 中。根据调用 OnDraw 函数的返回，CPaintDC 被框架销毁并且将基础设备上下文释放给 Windows。

CClientDC 对象---封装对一个只表示窗口工作区的设备上下文的处理。CClientDC 构造函数调用 GetDC 函数，析构函数调用 ReleaseDC 函数。CWindowDC 对象封装表示整个窗口（包括其框架）的设备上下文。

CMetaFileDC 对象---将绘制封装到 Windows 图元文件中。与传递给 OnDraw 的 CPaintDC 相反，在这种情况下您必须自己调用 OnPrepareDC。

**鼠标绘图**

框架程序中的大多数绘图（由此，大部分设备上下文参与）都在视图的 OnDraw 成员函数中完成。但是，您仍然可以将设备上下文对象作其他用途使用。例如，若要在视图中提供鼠标运动的跟踪回馈，只需直接绘制到视图中而无需等待调用 OnDraw。

在这种情况中，可以使用 CClientDC 设备上下文对象直接绘制到视图中。

再来一段比较清晰容易理解的。

一、区别与联系  
HDC是句柄；

CDC是MFC封装的Windows设备相关的一个类；

CClientDC是CDC的衍生类，产生对应于Windows客户区的对象

HDC是WINDOWS的一种数据类型，是设备描述句柄。  
而CDC是MFC里的一个类，它封装了几乎所有的关于HDC的操作。

也可以这样说，HDC定义的变量指向一块内存，这块 内存用来描述一个设备的相关的内容，所以也可以认为HDC定义的是一个指针；而CDC类定义一个对象，这个对象拥有HDC定义的一个设备描述表，同时也包含与HDC相关的操作的函数。这与HPEN和CPen，POINT与CPoint之间的差别是一样的。  
  
二、获得CDC \*  
CDC\* pDC  
pDC=GetDC();//原型CDC\* CWnd::GetDC()   
  
三、获得hdc  
HDC hDC;  
1,hDC=GetDC(pCpp->hWnd);  
2,pDC->m\_hDC;

3,（SDK中找不到相关类的支持）

    MEMDCXP Mdcxp;  
    GetMemDCXP(&Mdcxp);  
    hDC = Mdcxp.hMemDC;  
4,hDC=::GetDC(HWND handle)

四、转换(?，未确定用法的正确性)

CDC\* pDC  
HDC hDC;  
pDC=Attach(hDC);  
hDC=GetSafeHDC(pDC);  
pDC->m\_hDC=hDC；

下面一段翻译自MSDN SDK文档

    CDC类定义的是设备上下文对象的类。CDC对象提供处理显示器或打印机等设备上下文的成员函数，以及处理与窗口客户区对应的显示上下文的成员。      
   通过CDC对象的成员函数进行所有的绘图。类对设备上下文操作提供了成员函数，处理绘图工具。安全型图形设备接口（GDI）对象收集，以及处理颜色和调色板。它还为获取和设置绘图属性、映射，处理视点、窗口扩展、转换坐标，处理区域、剪贴、绘制直线及绘制简单椭圆和多边形等形状提供了成员函数。另外还为绘制文本、处理字体，使用打印机跳转，滚动和播放元文件提供成员函数。      
   使用CDC对象时要构造它，然后调用与它平等的、使用设备上下文的Windows函数的成员函数。

  为了特定用途，Microsoft基本类库提供了几个CDC派生类。CPaintDC包括BeginPaint和EndPaint调用。CClientDC管理窗口用户区对应的显示上下文。CWindowDC管理与整个窗口对应的显示上下文，包括它的结构和控件。CMetaFileDC与带元文件的设备上下文对应。      
   CDC包含m\_hDC和m\_hAttribDC两个设备上下文，它们在CDC对象创建时参考同一个设备。CDC指导所有对m\_hDC的输出GDI调用以及对m\_hAttribDC的大部分属性GDI调用（GetTextColor是属性调用的例子，而SetTextColor是一个输出调用）。      
   例如框架使用这两个设备上下文实现CMetaFileDC对象，在从物理设备读取属性期间向元文件发送输出。打印预览以相同风格在框架中实现。还可以以相似方法在特殊应用代码中使用这两个设备上下文。

<http://zoudaokou2006.blog.163.com/blog/static/66650867200812910281360/>

**CDC、HDC、pDC (zz)**

1.CDC \*pDC和HDC hdc有什么不同,类似的有CWnd \*pWnd和HWnd?  
  
pDC是类指针  
HDC是windows句柄  
通过pDC获得hdc:  
HDC hdc=pDC->GetSafeHdc();  
通过hdc获得pDC:  
CDC \*pDC=new CDC;  
pDC->Attach(hdc);  
  
2.hDC和CDC有本质区别  
  
HDC是WINDOWS的一种数据类型，是设备描述句柄。而CDC是MFC里的一个类，它封装了几乎所有的关于HDC的操作。也可以这样说，HDC定义的变量指向一块内存，这内存用来描述一个设备的相关的内容，所以也可以认为HDC定义的是一个指针；而CDC类定义一个对象，这个对象拥有HDC定义的一个设备描述表，同时也包含与HDC相关的操作的函数。这与HPEN和CPen，POINT与CPoint之间的差别是一样的。  
  
CDC是对hDC的相关操作进行封装，例如CDC的一个TextOut函数隐去其错误检测，完全可以简化到这样程度

1 CDC:TextOut( int x, int y, const CString& str )  
2 {  
3 TextOut( m\_hDC, x, y, (LPCTSTR)str, str.GetLength() );  
4 }

m\_hDC就是CDC的成员变量HDC m\_hDC;  
CDC有一个operator HDC() const { return m\_hDC; }     
你可以把它当成一个HDC使用  
  
3.this是dc输出目标窗口的指针，通过它可以得到窗口句柄，对象带参构造这有什么奇怪的呢？     
      
    CPaintDC        无效区dc，相当于BeginPaint,    EndPaint     
    CClientDC       客户区dc，相当于GetDC,         ReleaseDC     
    CWindowDC       整窗口dc, 相当于GetWindowDC,   ReleaseDC     
    CDC             任何dc,   相当于CreateDC,      DeleteDC

**HDC,CDC,CClientDC的区别和联系是什么？**

简而言之，HDC是句柄；CDC是MFC封装的Windows 设备相关的一个类；CClientDC是CDC的衍生类，产生对应于Windows客户区的对象

pDC 是 类指针   
HDC 是 windows句柄   
通过pDC获得hdc:   
HDC hdc=pDC->GetSafeHdc();   
通过hdc获得pDC:   
CDC \*pDC=new CDC;   
pDC->Attach(hdc);

HDC是WINDOWS的一种数据类型，是设备描述句柄。   
而CDC是MFC里的一个类，它封装了几乎所有的关于   
HDC的操作。   
也可以这样说，HDC定义的变量指向一块内存，这块   
内存用来描述一个设备的相关的内容，所以也可以   
认为HDC定义的是一个指针；而CDC类定义一个对象，   
这个对象拥有HDC定义的一个设备描述表，同时也包   
含与HDC相关的操作的函数。   
这与HPEN和CPen，POINT与CPoint之间的差别是一样   
的

显然     
  CDC是类     
  HDC是句柄     
  CDC是包了HDC的类

**CDC和HDC的区别与转换**

源自: <http://www.thatsky.cn/article.asp?id=81>

一、区别与联系  
HDC是句柄；CDC是MFC封装的Windows   设备相关的一个类；CClientDC是CDC的衍生类，产生对应于Windows客户区的对象  
HDC是WINDOWS的一种数据类型，是设备描述句柄。  
而CDC是MFC里的一个类，它封装了几乎所有的关于HDC的操作。     
也可以这样说，HDC定义的变量指向一块内存，这块 内存用来描述一个设备的相关的内容，所以也可以   认为HDC定义的是一个指针；而CDC类定义一个对象，   这个对象拥有HDC定义的一个设备描述表，同时也包   含与HDC相关的操作的函数。     
这与HPEN和CPen，POINT与CPoint之间的差别是一样的。  
  
二、获得CDC \*  
CDC\* pDC  
pDC=GetDC();  
  
三、获得hdc  
HDC hDC;  
1,hDC=GetDC(pCxp->hWnd);  
2,pDC->m\_hDC;  
3,  
MEMDCXP Mdcxp;  
GetMemDCXP(&Mdcxp);  
hDC = Mdcxp.hMemDC;  
4,hDC=::GetDC(HWND handle)  
  
四、转换  
CDC\* pDC  
HDC hDC;  
pDC=Attach(hDC);  
hDC=GetSafeHDC(pDC);  
pDC->m\_hDC==hDC

设备描述表

**设备描述表**

　　DC（Device Context）设备描述表

　　设备描述表是一个定义一组图形对象及其属性、影响输出的图形方式(数据)结构。windows提供设备描述表，用于应用程序和物理设备之间进行交互，从而提供了应用程序设计的平台无关性。设备描述表又称为设备上下文，或者设备环境。

　　设备描述表是一种数据结构，它包括了一个设备（如显示器和打印机）的绘制属性相关的信息。所有的绘制操作通过设备描述表进行。设备描述表与大多 WIN32结构不同，应用程序不能直接访问设备描述表，只能由各种相关API函数通过设备描述表的[句柄](http://baike.baidu.com/view/194921.htm)间接访问该结构。

　　设备描述表总是与某种系统硬件设备相关。比如屏幕设备描述表与显示设备相关，打印机设备描述表与打印设备相关等等。

　　屏幕设备描述表，一般我们简单地称其为设备描述表。它与显示设备具有一定的对应关系，在windows GDI界面下，它总是相关与某个窗口或这窗口上的某个显示区域。通常意义上窗口的设备描述表，一般指的是窗口的客户区，不包括标题栏、菜单栏所占有的区域，而对于整个窗口来说，其设备描述表严格意义上来讲应该称为窗口设备描述表，它包含窗口的全部显示区域。二者的操作方法完全一致，所不同的仅仅是可操作的范围不同而已。

　　windows 窗口一旦创建，它就自动地产生了与之相对应的设备描述表数据结构，用户可运用该结构，实现对窗口显示区域的GDI操作，如划线、写文本、绘制位图、填充等，并且所有这些操作均要通过设备描述表句柄了进行。

**如何理解设备描述表？？？**

设备描述表是一个定义一组图形对象及其属性、影响输出的图形方式(数据)结构。windows提供设备描述表，用于应用程序和物理设备之间进行交互，从而提供了应用程序设计的平台无关性。设备描述表又称为设备上下文，或者设备环境。       
  　　设备描述表是一种数据结构，它包括了一个设备（如显示器和打印机）的绘制属性相关的信息。所有的绘制操作通过设备描述表进行。设备描述表与大多WIN32结构不同，应用程序不能直接访问设备描述表，只能由各种相关API函数通过设备描述表的句柄间接访问该结构。     
  　　设备描述表总是与某种系统硬件设备相关。比如屏幕设备描述表与显示设备相关，打印机设备描述表与打印设备相关等等。     
  　　屏幕设备描述表，一般我们简单地称其为设备描述表。它与显示设备具有一定的对应关系，在windows   GDI界面下，它总是相关与某个窗口或这窗口上的某个显示区域。通常意义上窗口的设备描述表，一般指的是窗口的客户区，不包括标题栏、菜单栏所占有的区域，而对于整个窗口来说，其设备描述表严格意义上来讲应该称为窗口设备描述表，它包含窗口的全部显示区域。二者的操作方法完全一致，所不同的仅仅是可操作的范围不同而已。     
  　　windows   窗口一旦创建，它就自动地产生了与之相对应的设备描述表数据结构，用户可运用该结构，实现对窗口显示区域的GDI操作，如划线、写文本、绘制位图、填充等，并且所有这些操作均要通过设备描述表句柄了进行。     
  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/     
  书中是这么说的，但是如何理解啊？     
  如果将设备描述表理解成为设备的一种抽象地描述（或者说是硬件在数据结构上一种映射），那么显示设备描述表对应显示器，在编写程序的时候现在假设以GetDC获得设备描述标的句柄。     
  hdc   =   GetDC(hwnd);//获得窗口设备描述表句柄     
  [使用GDI函数]     
  ReleaseDC(hwnd,   hdc);     
  我想知道这个过程和显存的关系。也就是如何通过显存绘制到显示器对应的窗口上的。     
  再确切地说就是GDI绘制既然针对的是设备描述表，那么设备描述表是否在内存中?     
  他和显存之间的关系是什么？

通过设备描述表调用显示卡驱动，由显示卡驱动程序访问显存实现绘图输出，因此在GDI下面是无法直接访问到显存的。这种方式主要是提供了统一的编程界面，使得编程人员不用考虑具体的设备的特性，一切和设备打交道的工作交由系统完成，编程者只要调用统一的GDI函数即可。设备描述表在内存中是存在的。

                                                                                                                                                 转自CSDN

GDI是Graphics Device Interface的缩写，含义是图形设备接口，它的主要任务是负责系统与绘图程序之间的信息交换，处理所有Windows程序的图形输出。  
　　在Windows操作系统下，绝大多数具备图形界面的应用程序都离不开GDI，我们利用GDI所提供的众多函数就可以方便的在屏幕、打印机及其它输出设备上输出图形，文本等操作。GDI的出现使程序员无需要关心硬件设备及设备驱动，就可以将应用程序的输出转化为硬件设备上的输出，实现了程序开发者与硬件设备的隔离，大大方便了开发工作。   
　　GDI是如何实现输出的？　  
　　要想在屏幕或者其它输出设备上输出图形或者文字，那么我们就必须先获得一个称为设备描述表( DC:Device Context）的对象的句柄，以它为参数，调用各种GDI函数实现各种文字或图形的输出。设备描述表是GDI内部保存数据的一种数据结构，此结构中的属性内容与特定的输出设备（显示器，打印机等）相关，属性定义了GDI函数的工作细节，在这里属性确定了文字的颜色，x坐标和y坐标映射到窗口显示区域的方式等。  
　　设备描述表句柄一旦获得，那么系统将使用默认的属性值填充设备描述表结构。  
　　如果有必要，我们可以使用一些GDI函数获取和改变设备描述表中的属性值。

如何使用设备描述表 2007-02-22 19:46:18  www.hackbase.com  来源：互联网

本文示例源代码下载

　　Windows 程序在屏幕、打印机或其它设备上画图时，它并不是将像素直接输出到设备上，而是将图绘制到由设备描述表表示的

本文示例源代码下载

　　Windows 程序在屏幕、打印机或其它设备上画图时，它并不是将像素直接输出到设备上，而是将图绘制到由设备描述表表示的逻辑意义上的"显示平面"上去。设备描述表（DC）是Windows中的一种数据结构，它包含GDI需要的所有关于显示界面情况的描述字段，包括相连的物理设备和各种各样的状态信息。在Windows画图之前，Windows程序从GDI获取设备描述表句柄（HDC）,并在每次调用完GDI输出函数后将句柄返回给GDI。本文将对如何获得和释放 HDC以及 CDC及其派生类的使用方法进行演示说明。

　　下载示例工程　DCDemo 具体说明：

　　1、按钮 IDC\_GETDC\_APINULL的函数

void CDCDemoDlg::OnGetdcApinull()

{

　　HDC hDC=::GetDC(NULL);

　　::MoveToEx(hDC,0,0,NULL);

　　LineTo(hDC,200,20);

　　::ReleaseDC(NULL,hDC);

}　　该段代码具体演示了如何使用API 函数GetDC(NULL),取得屏幕的HDC.并进行画图。 HDC GetDC(HWND hWnd); //功能：取得hWnd窗口的设备描述表句柄。当hWnd参数为NULL时，取得整个屏幕的设备描述表句柄。 MoveToEx和LineTo演示在DC上画一条直线记住，最后调用ReleaseDC释放HDC资源。int　ReleaseDC (

HWND hWnd,//你要控制的那个窗口的句柄，如果你在GetDC函数传递的是NULL，现在还要传递NULL。

HDC hDC //DC的句柄

);编译运行程序，按下按钮，发现屏幕的左上角处画了一条直线。

　　2、按钮 IDC\_GETDC\_API的函数

void CDCDemoDlg::OnGetdcApi()

{

　　HDC hDC=::GetDC(m\_hWnd);

　　::MoveToEx(hDC,0,0,NULL);

　　LineTo(hDC,200,50);

　　::ReleaseDC(m\_hWnd,hDC);

}　　这段代码与前一段代码唯一的区别是GetDC的参数不再是NULL，而改为CWnd的成员变量m\_hWnd，即对话框窗口的句柄。对比两段代码运行结果，深入体会API函数 GetDC。

　　3、按钮IDC\_GETDC\_CWND的函数

void CDCDemoDlg::OnGetdcCwnd()

{

　　CDC \*pDC=GetDC();

　　pDC->MoveTo(0,0);

　　pDC->LineTo(200,100);

　　ReleaseDC(pDC);

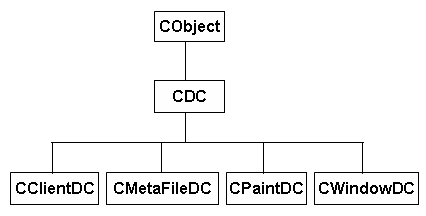
}　　该段代码演示使用MFC CWnd类的 GetDC函数和ReleaseDC的方法,使用该方法在程序窗口中画图非常方便。为了避免获取和释放设备描述表所带来的麻烦，MFC提供了一些CDC派生类，如CPaintDC,CClientDC,CWindowDC,这些类被设计为可直接进行实例化。各个类的构造函数和析构函数调用相应的函数捕获和释放设备描述表，从而使得更加方便简捷。

　　4、按钮IDC\_CCLIENTDC的函数

<http://www.hackbase.com/lib/2007-02-22/13807.html>

windows 程序设计中的 设备描述表

从CDC 派生出四个功能更具体的设备描述表类。层次如图所示。



下面，分别讨论派生出的四种设备描述表。

CCientDC   
代表窗口客户区的设备描述表。其构造函数CClientDC(CWnd \*pWin)通过::GetDC获取指定窗口的客户区的设备描述表HDC，并且使用成员函数Attach把它和CClientDC对象捆绑在一起；其析构函数使用成员函数Detach把设备描述表句柄HDC分离出来，并调用::ReleaseDC释放设备描述表HDC。

CPaintDC   
仅仅用于响应WM\_PAINT消息时绘制窗口，因为它的构造函数调用了::BeginPaint获取设备描述表HDC，并且使用成员函数Attach把它和CPaintDC对象捆绑在一起；析构函数使用成员函数Detach把设备描述表句柄HDC分离出来，并调用::EndPaint释放设备描述表HDC，而::BeginPaint和::EndPaint仅仅在响应WM\_PAINT时使用。

CMetaFileDC   
用于生成元文件。

CWindowDC   
代表整个窗口区(包括非客户区)的设备描述表。其构造函数CWindowDC(CWnd \*pWin)通过::GetWindowDC获取指定窗口的客户区的设备描述表HDC，并使用Attach把它和CWindowDC对象捆绑在一起；其析构函数使用Detach把设备描述表HDC分离出来，调用::ReleaseDC释放设备描述表HDC。

要使用设备描述表，一般有如下步骤：

获取或者创建设备描述表；

必要的话，改变设备描述表的属性；

使用设备描述表完成绘制操作；

释放或删除设备描述表。   
Common设备描述表通过::GetDC，::GetDCEx，::BeginPaint来获得一个设备描述表，用毕，用::ReleaseDC或::EndPaint释放设备描述表；

Printer设备描述表通过::CreateDC创建设备描述表，用::DeleteDC删除设备描述表。

Memory设备描述表通过::CreateCompatibleDC创建设备描述表，用::DeleteDC

Information设备描述表通过::CreateIC创建设备描述表，用::DeleteDC删除

本文来自CSDN博客，转载请标明出处：<http://blog.csdn.net/anthit/archive/2007/05/15/1610037.aspx>

转自：<http://www.cnblogs.com/iwanc/archive/2012/02/28/2371220.html>