# DDB位图的显示

## 1、DDB与DIB

DDB（Device-dependent bitmap，与设备相关的位图）依赖于具体设备，它只能存在于内存中（视频内存或系统内存），其颜色模式必须与特定的输出设备相一致，使用系统调色板。一般只能载入色彩较简单的DDB位图，对于颜色较丰富的位图，需使用DIB才能长期保存。

DIB（Device-independent bitmap，与设备无关的位图） 不依赖于具体设备，可以用来永久性地保存图象。DIB一般是以\*.BMP文件的形式保存在磁盘中的，有时也会保存在\*.DIB文件中。 DIB位图的特点是将颜色信息储存在位图文件自身的颜色表中，应用程序要根据此颜色表为DIB创建逻辑调色板。因此，在输出一幅DIB位图之前，程序应该将其逻辑调色板选入到相关的设备上下文并实现到系统调色板中。

## 2、CBitmap、HBitmap、Bitmap区别和联系

### 2.1区别

CBitmap是MFC中封闭的bitmap的类，HBitmap是bitmap的指针，BITMAP是一个结构体，封装着bitmap的一些信息，定义了逻辑位图的高宽颜色格式和位值。

BITMAP结构具有如下形式：  
typedef struct tagBITMAP  
{

    int      bmType;  
     int      bmWidth;//宽  
     int      bmHeight;//高  
     int      bmWidthBytes;  
     BYTE     bmPlanes;  
     BYTE     bmBitsPixel;  
     LPVOID bmBits;  
}BITMAP;

### 2.2联系：

三者之间的关系转换：

HBITMAP hBitmap;

CBitmap bitmap;

BITMAP bm;

bitmap.Attach(hBitmap);//由HBITMAP 得到关联的CBitmap

bitmap.GetBitmap(&bm);//由CBitmap 得到关联的BITMAP   
hBitmap=(HBITMAP)bitmap.GetSafeHandle();//由CBitmap得到相关的HBITMAP

## 3、DDB的缺点

在VC中使用CBitmap类，必须将位图放入工程的资源中，并使用类 CBitmap的成员函数LoadBitmap加载之，再通过CDC类的成员函数BitBlt进行DC拷贝等操作达到显示的目的。CBitmap有显示的不足：

（1） 位图需要放入工程资源中，这将导致工程的可执行文件变大；

（2） 因为位图需放入工程资源中，而资源中不能无穷无尽地包含位图，应用程序无法自适应地选取其它位图，能使用的位图十分有限的；

（3） 类CBitmap只是DDB位图操作API的封装，不能独立于平台。

DIB位图则可以解决上述问题，其特点是以.BMP位图文件格式存储独立于平台的图像数据。

## 4、显示DDB位图

### 4.1、方法一：从资源中载入位图

定义变量：

CBitmap bitmap;

在OnInitialUpdate()函数中：

if (bitmap.GetSafeHandle()==NULL)

{

bitmap.LoadBitmap(IDB\_CHICAGO);

}

OnDraw函数中：

CDC dcMemory;

dcMemory.CreateCompatibleDC(pDC);

if (bitmap.GetSafeHandle()!=NULL)

{

HBITMAP hOldBitmap=(HBITMAP)dcMemory.SelectObject(bitmap.GetSafeHandle());

BITMAP bmp;

bitmap.GetBitmap(&bmp);

pDC->BitBlt(0,0,bmp.bmWidth,bmp.bmHeight,&dcMemory,0,0,SRCCOPY);

dcMemory.SelectObject(hOldBitmap);

}

DeleteObject(dcMemory.GetSafeHdc());

说明：程序定义了一个私有变量CBitmap bitmap，在之后的程序中，负责载入和保存位图。

在窗口更新的时候，bitmap载入位图，用到的函数是LoadBitmap，在载入之前，先检查其内部是否已载入位图，方法是检查bitmap.GetSafeHandle()是否返回NULL，若是，则说明内部还没有位图，可直接调用LoadBitmap载入，如果不为NULL，表明内部已有位图数据，此时如果想要载入新的位图数据，可调用Detach()函数，例如：

if (bitmap.GetSafeHandle()!=NULL)

{

bitmap.Detach();

bitmap.LoadBitmap(IDB\_BITMAP);

}

在OnDraw中显示位图有几个步骤：

1）创建与当前DC兼容的兼容DC

CDC dcMemory;

dcMemory.CreateCompatibleDC(pDC);

2）将位图选入兼容DC

HBITMAP hOldBitmap=(HBITMAP)dcMemory.SelectObject(bitmap.GetSafeHandle());

3）为得到位图的尺寸数据，可定义BITMAP结构体，调用GetBitmap可返回位图的相关信息：

BITMAP bmp;

bitmap.GetBitmap(&bmp);

4）将兼容DC上的位图复制到当前DC上

pDC->BitBlt(0,0,bmp.bmWidth,bmp.bmHeight,&dcMemory,0,0,SRCCOPY);

5）为兼容DC重新选入旧的位图

dcMemory.SelectObject(hOldBitmap);

6）销毁兼容DC

DeleteObject(dcMemory.GetSafeHdc());

或：

DeleteDC(dcMemory.GetSafeHdc());

（不知两都有何差别，但实验证明，两者都能用）

注意：在销毁DC的时候两个常见的函数DeleteDC()和ReleaseDC()，两者是不同的：

由CreateDC、CreateCompatibleDC函数创建的设备上下文环境，只能用DeleteDC来释放。如果一个设备上下文的句柄是通过调用GetDC得到的，那么应用程序不能删除设备上下文，而应该用ReleaseDC来释放。

### 4.2 方法二：从磁盘中载入位图

定义变量：

HBITMAP HBM;

在OnInitialUpdate()函数中：

if(HBM==NULL)

{

HBM=(HBITMAP)LoadImage(NULL,\_T("E:\\TESTBMP.bmp"),IMAGE\_BITMAP,

0,0,LR\_CREATEDIBSECTION|LR\_LOADFROMFILE);

}

注意，这里用的是LoadImage而不是LoadBitmap.

在OnDraw函数中：

if (HBM!=NULL)

{

BITMAP bmp;

GetObject(HBM,sizeof(bmp),&bmp);

CDC dcMemory;

dcMemory.CreateCompatibleDC(pDC);

dcMemory.SelectObject(HBM);

BitBlt(pDC->GetSafeHdc(),0,0,bmp.bmWidth,bmp.bmHeight,dcMemory,0,0,SRCCOPY);

DeleteObject(dcMemory);

}

说明：从磁盘中载入位图用LoadImage，用法如下：

HBM=(HBITMAP)LoadImage(NULL,\_T("E:\\TESTBMP.bmp"),IMAGE\_BITMAP,

0,0,LR\_CREATEDIBSECTION|LR\_LOADFROMFILE);

该函数返回的是HBITMAP句柄，所以用一个HBITMAP来接收它，而不是一个CBitmap变量，另外，从磁盘中载入，LR\_CREATEDIBSECTION|LR\_LOADFROMFILE是必须的。在载入位图前，先要检查HBM是否为NULL，如果为NULL再载入，若不是NULL，先要调用DeleteObject释放句柄，例如：

if (HBM!=NULL)

{

DeleteObject(HBM);

HBM=(HBITMAP)LoadImage(NULL,\_T("E:\\Chicago.bmp"),IMAGE\_BITMAP,0,0,

LR\_LOADFROMFILE|LR\_CREATEDIBSECTION);

}

在获取位图信息的时候，用到的函数是GetObect。

以上两种方法的主要区别：

|  |  |  |
| --- | --- | --- |
| 保存位图变量 | CBitmap | HBITMAP |
| 载入位图 | LoadBitmap | LoadImage |
| 检测变量是否为空 | if (bitmap.GetSafeHandle()==NULL) | if(HBM==NULL) |
| 获取位图信息 | BITMAP bmp;  bitmap.GetBitmap(&bmp); | BITMAP bmp;  GetObject(HBM,sizeof(bmp),&bmp); |
| 释放变量 | bitmap.Detach(); | DeleteObject(HBM); |