DirectSound 的帮助文档（中文版）

（2006 年 4 月 28 日早）

weihua_zhang

What's New in DirectSound

DirectSound API 9.0 增强了许多功能。

下面是改进的方面：

·当操作系统支持时，缓冲区的最大频率从 100khz 增加到 200KHZ，（DirectShow 不支持高频率）

·频率和音效控制(DSBCAPS_CTRLFREQUENCY and DSBCAPS_CRTLFX)已经与缓冲区绑定，通过绑定标志可以对缓冲区做多普勒音效变换。

·音频文件是通过WAVEFORMATEXTENSIBLE 结构体来描述的。

·标准音效 DMOs 在模式 WAVE_FORMAT_IEEE_FLOAT 下能被处理。

·标准音效支持 parameter curves 。

DirectSound 介绍

这节提供了帮助你快速浏览的 DirectSound，并能开始学习 API。

DirectSound 的功能。

DirectSound 和 DirectMusic

DirectSound 的功能

DirectSound 播放声音是通过较低的的功能实现，提供给应用程序一个较高的控制级，来访问硬件资源。

使用 DirectSound 应用程序接口可以实现：

从 WAVE 文件中播放声音。

同时播放多个声音

赋予较高的权限去控制硬件缓冲区

在 3D 环境中定位声音

添加音效诸如回声合唱，可以动态改变音效

从麦克风或其他输入设备捕获声音

DirectSound 和 DirectMusic

DirectSound 和 DirectMusic 时 DirectX 的两个分离部分，但是功能有部分重叠，都是播放 wave 声音，DirectMusic 最终实现合成所有声音并已 WAVE 格式存放在 DirectSound 缓冲区，使其可以演奏。

在游戏和其他交互式应用程序中 DirectMusic 提供了丰富的 API 去实现完美的方案。DirectMusic 可以播放各种声音，包括非音乐的声音。DirectSound 是管理硬件资源的 API.

你可以使用 DirectSound API 直接去播放 WAVE 格式的声音，甚至可以使用 DirectMusic 去演奏其他内容，也可以用 DirectSound 去操纵由 DirectMusic 所管理的缓冲区。例如，你

可以将 DirectMusic 的缓冲区移到 3D 空间。

Functionality	DirectMusic	DirectSound
Play WAV sounds	Yes	Yes
Play MIDI	Yes	No
Play DirectMusic Producer segments	Yes	No
Load content files and manage objects	Yes	No, but some support in sample code
Control musical parameters at run time	Yes	No
Manage timeline for cuing sounds	Yes	No
Use downloadable sounds (DLS)	Yes	No
Set volume, pitch, and pan of individual sounds	Yes, through DirectSound API	Yes
Set volume on multiple sounds (audiopaths)	Yes	No
Implement 3-D sounds	Yes, through DirectSound API	Yes
Apply effects (DMOs)	Yes, through DirectMusic Producer content or DirectSound API	Yes
Chain buffers for mix-in (send) effects	Yes, through DirectMusic Producer content	No
Capture WAV sounds	No	Yes
Implement full duplex	No	Yes
Capture MIDI	Yes	No
Control allocation of hardware buffers	No	Yes

DirectSound 之始

这节提供了如何设置和调试 DirectSound 的工程文件，和编程的步骤：

·DirectSound 的工程文件

工程文件要包含 Dsound.h 头文件，当用 DirectSound 和 DirectMusic 联合编程时，应在添加 DMusic.h

如果你直接使用API或是拥有帮助性的函数如DirectSoundCreate8 应去连接Dsound.lib。

你必须确保 DirectSound 在使 GUIDs 在所有的预处理指示符前加上 Define INITGUID 或者是连接 Dxguid.lib

DirectSound Projects 的调试

你可以设置调试级别，在在控制面板的 DirectX 属性也上的调试输出级别对话框上有 0 到 5 级。0 为没有条是信息，5 为特别详细，许多情况下不需要设置高于 2。

First Steps in DirectSound Programming

演奏一小段的音乐要以下步骤：

1. 创建一个设备对象

调用 **DirectSoundCreate8** 去创建一个支持 **IDirectSound8** 接口的对象。这个对象通常代表默认的演奏设备，当然你可以枚举所有可用的设备，并将设备 GUID 传给 **DirectSoundCreate8。**

注：DirectSound是基于COM的，但是你不必初始化，除要用到DMOs.

2. 创建副缓冲区

使用IDirectSound8::CreateSoundBuffer可以创建一个缓冲区对象，它可以用来存放声音数据，并且是对主缓冲区的区别，主缓冲区主要可以做混音的动作。

3. 提供 PCM 数据

从 WAVE 文件中读取数据放在私有缓冲区中。

4. 通过调用 IDirectSoundBuffer8::Lock 将数据写到副缓冲区中，然后再调用 IDirectSoundBuffer8::Unlock.

5. 播放缓冲区的数据

通过调用IDirectSoundBuffer8::Play.去播放声音，

下面的对象用来播放声音：

对象	数量	目的	主接口
Device	One in each application	Manages the device and creates sound buffers.	IDirectSound8
Secondary buffer	One for each sound	Manages a static or streaming sound and plays it into the primary buffer.	IDirectSoundBuffer8, IDirectSound3DBuffer8, IDirectSoundNotify8
Primary buffer	One in each	Mixes and plays	IDirectSoundBuffer,

	application	sounds from secondary buffers, and controls global 3-D parameters.	IDirectSound3DListener8
Effect	Zero or more for each secondary buffer	Transforms the sound in a secondary buffer.	Interface for the particular effect, such as IDirectSoundFXChorus8

在应用程序中第一步是创建一个 DirectSound 设备对象，用它来代表设备，用这个对象区创建缓冲区。
应用程序区创建和管理副缓冲区， DirectSound 自动创建和管理主缓冲区，并且应用程序能再没有得到对象接口的时候，播放声音。为了获得 IDirectSound3DListener8 接口，应用程序强制创建一个主缓冲区对象。当副缓冲区的声音播放以后，DirectSound 将其混进主缓冲区，然后发到设备上去，DirectSound 可以混音的缓冲区数量是受限制的。短小的声音可以被直接装载到副缓冲区，通过对方法的简单调用，就能实现人以播放。长声音必须被流式话以后才能播放，应用程序可以探知缓冲区数据流的位置，或者通过查询播放游标的位置，当播放游标到达某个确定位置时，通知应用程序。副缓冲区能实现回声等音效。

枚举声音设备

要去枚举设备，你首先必须定义一个回调函数，该函数可以被系统上的每个设备所调用一次。你可以在该函数中做你想做的事，函数名可以自己命名，但是必须将其声明为 DSEnumCallback 的原型。这个回调函数必须返回TRUE或FALSE。

下面的回调函数是完成将每个枚举到的设备名字添加到 Combo Box 中，并且存储设备的 GUID 作为一个数据项。 前面三个参数是设备驱动供应商提供，第四个参数通过 **DirectSoundEnumerate** 这个函数的参数传递一个 32 位的值，该值就是 Combo Box 的 窗口句柄。

```
BOOL CALLBACK DSEnumProc(LPGUID lpGUID,
         LPCTSTR lpszDesc,
         LPCTSTR lpszDrvName,
         LPVOID lpContext )
{
  HWND hCombo = (HWND)lpContext;
  LPGUID lpTemp = NULL;

  if (lpGUID != NULL)  //  NULL only for "Primary Sound Driver".
```

```
  {
    if ((lpTemp = (LPGUID)malloc(sizeof(GUID))) == NULL)
    {
        return(TRUE);
    }
    memcpy(lpTemp, lpGUID, sizeof(GUID));
  }

  ComboBox_AddString(hCombo, lpszDesc);
  ComboBox_SetItemData(hCombo,
      ComboBox_FindString(hCombo, 0, lpszDesc),
      lpTemp );
  free(lpTemp);
  return(TRUE);
}
```

这个枚举是初始化 combo box ，假定 *hCombo* 是 combo box 的句柄而且 *hDlg* 是对话框的句柄：.

```
if (FAILED(DirectSoundEnumerate((LPDSENUMCALLBACK)DSEnumProc,
    (VOID*)&hCombo)))
{
  EndDialog(hDlg, TRUE);
  return(TRUE);
}
```

I 在这种情况下，combo box 的句柄通过 **DirectSoundEnumerate** 传递给回调函数，这是一个 **32** 位的值，你可以再回调函数中去访问它。

注：第一个枚举的设备总是主声音设备，并且回调函数的 ***lpGUID*** 参数总是会为 ***NULL,***这个设备代表着用户在控制面板中设定的首选设备。

创建一个设备对象

用DirectSoundCreate8 函数可以很简单的区创建一个设备驱动程序，这个函数的第一个参数指出设备的GUID,你可以通过枚举设备来获得设备的GUID或你可以传递下列一个默认的 GUIDs来指定一个默认的设备。

GUID 默认值	详细信息
DSDEVID_DefaultPlayback	这时系统默认的声音设备，你可以传递一个 NULL 来指代这个设备。
DSDEVID_DefaultVoicePlayback	默认的声音通信设备，一般情况是第二设备。

如果没有相应的设备，**DirectSoundCreate8** 会调用失败！

如果没有声音设备，或是在VXD 驱动下都会返回错误！如果设备在应用程序却使用标准德WIN32 波形声音函数去控制设备，那么该函数会调用失败，那么应给用户适当的提示信息，并使程序能正常执行。下面的代码就是通过IDirectSound8 的接口去创建一个默认设备的对象。

```
LPDIRECTSOUND8 lpds;
HRESULT hr = DirectSoundCreate8(NULL, &lpds, NULL));
```

Note **DirectSoundCreate8** 这个不要调用**CoInitialize** 或 **CoInitializeEx** 是因为你的应用程序没有使用 DMOs,如果你的程序要捕获声音并且进行播放，你会很方便的去创建播放和捕获功能的对象，使用DirectSoundFullDuplexCreate8 并且可以获得在缓冲区。

你可以通过 COM 的标准函数去创建一个对象：

1. 用 **CoInitializeEx**. 在应用程序开始失去初始化 COM 库

   ```
   HRESULT hr = CoInitializeEx(NULL, 0);
   if (FAILED(hr))
   {
     ErrorHandler(hr);  // Add error-handling here.
   }
   ```

2. 通过使用 **CoCreateInstance** a和 IDirectSound8::Initialize方法去创建一个对象,　　至少要使用 DirectSoundCreate8 函数

   ```
   LPDIRECTSOUND8 lpds;
   hr = CoCreateInstance(&CLSID_DirectSound8,
             NULL,
             CLSCTX_INPROC_SERVER,
             IID_IDirectSound8,
             (LPVOID*) &lpds);
   if (FAILED(hr))
   {
     ErrorHandler(hr);  // Add error-handling here.
   }
   ```

   CLSID_DirectSound8 是 *DirectSound* 设备驱动类的类标识，*IID_IDirectSound8* 是接口标识。 *lpds* 接受接口的指针。

3. 调用 IdirectSound8::Initialize 这个方法 可以将设备和对象进行关联，这个方法与 DirectSoundCreate8 使用同一个GUID参数.。

   ```
   hr = lpds->Initialize(NULL);
   if (FAILED(hr))
   ```

```
{
  ErrorHandler(hr);  // Add error-handling here.
}
```

4. 在你退出应用程序之前应该调用 **CoUninitialize** 函数关闭 COM 库 。

与硬件的协作级

因为 Windows 是一个多任务环境，同一时间由多个应用程序去访问设备，通过使用协作级别， DirectX 可以确保应用程序不会在别的设备使用时去访问，每个 DirectSound 应用程序都有一个协作级别，这个级别决定着访问硬件的权限。

在创建一个设备对象以后，你必须通过用IDirectSound8::SetCooperativeLevel 来设置协作权限，否则你将听不到声音。

下面是通过 IdirectSound8 的对象指针，来设置写作权限， *hwnd* 参数是应用程序的窗口句柄；

```
HRESULT hr = lpDirectSound->SetCooperativeLevel(hwnd, DSSCL_PRIORITY);
if (FAILED(hr))
{
  ErrorHandler(hr);  // Add error-handling here.
}
```

DirectSound 定义了三个写作权限，分别是： DSSCL_NORMAL, DSSCL_PRIORITY, 和 DSSCL_WRITEPRIMARY.

```
////////////////////////////////////////////////////////////
/////////
```

一般协作级

一般协作级 (DSSCL_NORMAL), 应用程序不能设置primary buffer 的格式.，所有的应用程序使用此级别，都是用的是 22KHZ的主缓冲区。并且是 8 位的采样率，这样设备在应用程序之间可以平滑的转变。

优先权协作级

当以优先权协作级(DSSCL_PRIORITY),使用一个 DirectSound 设备时， 这个应用程序首先就有了硬件的资源，诸如硬件混音，设置主缓冲区的格式。

游戏程序应该使用该权限，这个级别在允许应用程序控制采样率，位深，更加鲁棒，允许声音来自其他的应用程序。

写主缓冲区级

这是最高的级别 (DSSCL_WRITEPRIMARY). 你的应用程序可以直接去访问主缓冲区,在这种模式里，应用程序必须直接写主缓冲区，副缓冲区不能再去播放。如果应用程序不是刺激别的话，那么在呼叫 **IDirectSoundBuffer::Lock** 时将失败。注：主缓冲区支持的是**IDirectSoundBuffer**而不是 IDirectSoundBuffer8.

该级别上应用程序获得使用前景权后，所有的副缓冲区都将停止而且数据丢失。在返回背景时主缓冲区的数据丢失。.

//

设备性能

DirectSound 能使你的应用程序去考核声音硬件的性能，许多应用程序不需要去做这些，因为 DirectSound 会自动利用有效的硬件资源，高性能的应用程序，使用这些信息来满足他们的需要，例如：一个应用程序可以选择是否用硬件进行混音（前提是硬件支持混音）。

在调用了 DirectSoundCreate8 函数创建了一个设备对象之后，你的应用程序能通过调用,IDirectSound8::GetCaps 来返回设备的性能。

下面的例子是通过 Idirect Sound8 的接口指针lpDirectSound:来返回 设别的性能 。

```
DSCAPS dscaps;

dscaps.dwSize = sizeof(DSCAPS);
HRESULT hr = lpDirectSound->GetCaps(&dscaps);
if (FAILED(hr))
{
  ErrorHandler(hr);  // Add error-handling here.
}
```

DSCAPS 结构接收声音设备的操作性和资源信息。包括每种资源的最大数目，和当前有效的资源数。 **dwSize** 要初始化为该结构的大小。

扬音器的配置

DirectSound 扬音器的配置—实质上是如 3D 效果等。

在 98, 2000,操作系统在控制面板已经配置过，用户只要返回至来使用即可。IdirectSound8::GetSpeakerConfig.这个函数可以将值返回。

DirectSound 缓冲区

基本缓冲区

应用程序必须至少创建一个副缓冲区，用来存储要播放的声音数据文件。一个副缓冲区生命期可以比应用程序还长， 所以在不需要的时候，应将其释放，如果的静态的缓冲区那么它将保存完整的声音数据，或是流缓冲区重新刷新数据播放,较长的声音数据应通过流缓冲区进行播放。你可以将副缓冲区的不同声音数据进行同时播放，来达到混音的效果。不能创建相同的副缓冲区，缓冲区的特征如下：

- **Format**.这是缓冲区要播放的 WAVE Format 相匹配的格式
- **Controls**.不同的缓冲区可以有不同的配置，诸如音量，频率等。当创建一个缓冲区时，你应该指出你所需要的，例如：不要在非 3D 的环境中创建一个 3D 的缓冲区。
- **Location**. 缓冲区时硬件来管理的 ，或是由软件来管理的，硬件缓冲区通常很有效率，但它是有限制，硬件缓冲区不支持 64-bit 的操作系统。

创建副缓冲区

去创建一个缓冲区，只要调用 IDirectSound8::CreateSoundBuffer 方法，这个方法将返回一个指向 **IDirectSoundBuffer** 接口的指针，用这个指针程序可以获得 IDirectSoundBuffer8 的接口 。

```
HRESULT CreateBasicBuffer(LPDIRECTSOUND8 lpDirectSound,
LPDIRECTSOUNDBUFFER8* ppDsb8)
{
  WAVEFORMATEX wfx;
  DSBUFFERDESC dsbdesc;
  LPDIRECTSOUNDBUFFER pDsb = NULL;
  HRESULT hr;

  // Set up WAV format structure.

  memset(&wfx, 0, sizeof(WAVEFORMATEX));
  wfx.wFormatTag = WAVE_FORMAT_PCM;
  wfx.nChannels = 2;
  wfx.nSamplesPerSec = 22050;
  wfx.nBlockAlign = 4;
  wfx.nAvgBytesPerSec = wfx.nSamplesPerSec * wfx.nBlockAlign;
  wfx.wBitsPerSample = 16;

  // Set up DSBUFFERDESC structure.
```

```
  memset(&dsbdesc, 0, sizeof(DSBUFFERDESC));
  dsbdesc.dwSize = sizeof(DSBUFFERDESC);
  dsbdesc.dwFlags =
    DSBCAPS_CTRLPAN | DSBCAPS_CTRLVOLUME | DSBCAPS_CTRLFREQUENCY
    | DSBCAPS_GLOBALFOCUS;
  dsbdesc.dwBufferBytes = 3 * wfx.nAvgBytesPerSec;
  dsbdesc.lpwfxFormat = &wfx;

  // Create buffer.

  hr = lpDirectSound->CreateSoundBuffer(&dsbdesc, &pDsb, NULL);
  if (SUCCEEDED(hr))
  {
     hr = pDsb->QueryInterface(IID_IDirectSoundBuffer8, (LPVOID*)
ppDsb8);
     pDsb->Release();
  }
  return hr;
}
```

这个例子创建了一个流缓冲区，该缓冲区可以保存 3 秒的流数据。非流式缓冲区要能完全包容声音数据。 DSBCAPS_GLOBALFOCUS 标志表明当应用程序窗口在失去焦点是缓冲区依然能够继续播放。没有这个标志，当时去焦点时就会不发音。如果缓冲区的位置没有指定，DirectSound 将其放入硬件内存中。因为通过声卡处理，硬件可以发音，几乎不影响应用程序性能。

如果你希望指定缓冲区位置，比让DirectSound 来决定他在哪里好，在结构 DSBUFFERDESC 中设置DSBCAPS_LOCHARDWARE 或 DSBCAPS_LOCSOFTWARE 标志。. 如果设置DSBCAPS_LOCHARDWARE 标志，没有硬件资源的情况下，缓冲区的创建就会失败。要利用 DirectSound对声音的管理，创建缓冲区就要设置 DSBCAPS_LOCDEFER 标志。这个标志提供了为缓冲区分配资源的能力。你能通过使用IDirectSoundBuffer8::GetCaps确定缓冲区的位置 ，可以去检测 DSBCAPS 结构成员**dwFlags**是否是 DSBCAPS_LOCHARDWARE or DSBCAPS_LOCSOFTWARE 。缓冲区对象是属于设备对象的。当设备对象被释放，哪么所用缓冲区对象都将被释放。

复制缓冲区

你可以创建两个或更多的饱含同样声音数据的副缓冲区，通过使用 IDirectSound8::DuplicateSoundBuffer 这个方法能达到这个效果。 当然不能复制主缓冲区了，因为复制缓冲区时是共享了原始的内存。如果改变将影响其他声音的改变。

缓冲区的控制配置

当创建一个声音缓冲区时，应用程序必须指定控制配置， DSBUFFERDESC 结构的dwFlags 成员， 它可以设置成下列标志的组合：

Flag	Description
DSBCAPS_CTRL3D	The sound source can be moved in 3—D space.
DSBCAPS_CTRLFX	Effects can be added to the buffer.
DSBCAPS_CTRLFREQUENCY	The frequency of the sound can be changed.
DSBCAPS_CTRLPAN	The sound source can be moved from left to right.
DSBCAPS_CTRLPOSITIONNOTIFY	Notification positions can be set on the buffer.
DSBCAPS_CTRLVOLUME	The volume of the sound can be changed.

为了获得较好的操作性能，应用程序应该指定他所用到的配置。

DirectSound 通过控制配置来决定是否分配硬件资源来给应用程序，例如 一个硬件支持缓冲区，但不提供音调控制，这种情况如果没有指明DSBCAPS_CTRLPAN标志的话DirectSound将使用硬件作为加速设备。如果应用程序试图通过IDirectSoundBuffer8::SetVolume 使用控制资源很少的缓冲区，如果设置了DSBCAPS_CTRLVOLUME 标志，那么这样的操作将会成功返回。失败时返回DSERR_CONTROLUNAVAIL 错误代码！

Buffers 的 3-D 算法

当创建副缓冲区时设置 DSBCAPS_CTRL3D 控制标志时，可以给声音指定一定的算法使其能够变化，默认 HRTF 是主要算法。

填充静态缓冲区

副缓冲区包含完整的声音数据就叫做静态缓冲区。虽然，和其他缓冲区没有什么不同，但是数据却是只写进去一次。静态缓冲区的创建和管理很想流式缓冲区，他们所不同的是在使用上，静态缓冲区只填充一次然后去使用，但是流式缓冲区是需要不断用待演奏的数据刷新。

注：静态缓冲区是不需要在创建时设置 DSBCAPS_STATIC 标志去刻画缓冲区的，这个标志要求声卡分配内存，在大多数硬件上是可以的。静态缓冲区可以存在系统内存内，创建时加上 DSBCAPS_LOCHARDWARE 或 DSBCAPS_LOCSOFTWARE 标志。

给静态缓冲区装载数据需要三个过程：

1. 通过调用IDirectSoundBuffer8::Lock.将缓冲区锁住，指出你要写入的偏移地址，然后得到内存的地址。

2. 通过标准的内存复制机制将声音写声音数据到返回的地址。
3. 调用 IDirectSoundBuffer8::Unlock.

这些步骤在下面的例子中有体现， *lpdsbStatic* 是一个 IDirectSoundBuffer8 接口指针 ，*pbData* 是数据源的地址。

```
LPVOID lpvWrite;
DWORD  dwLength;

if (DS_OK == lpdsbStatic->Lock(
      0,          // Offset at which to start lock.
      0,          // Size of lock; ignored because of flag.
      &lpvWrite,  // Gets address of first part of lock.
      &dwLength,  // Gets size of first part of lock.
      NULL,       // Address of wraparound not needed.
      NULL,       // Size of wraparound not needed.
      DSBLOCK_ENTIREBUFFER))  // Flag.
{
  memcpy(lpvWrite, pbData, dwLength);
  lpdsbStatic->Unlock(
      lpvWrite,   // Address of lock start.
      dwLength,   // Size of lock.
      NULL,       // No wraparound portion.
      0);         // No wraparound size.
}
else
(
  ErrorHandler();  // Add error-handling here.
}
```

去播放缓冲区要调用 IDirectSoundBuffer8::Play,

```
lpdsbStatic->SetCurrentPosition(0);
HRESULT hr = lpdsbStatic->Play(
    0,  // Unused.
    0,  // Priority for voice management.
    0); // Flags.
if (FAILED(hr))
(
  ErrorHandler();  // Add error-handling here.
}
```

因为 DSBPLAY_LOOPING 标志没有在这个例子中设定，缓冲区在到达末尾的时候将自动停止，当然你可以去停止播放，通过调用 IDirectSoundBuffer8::Stop. 来实现。当你停止播放，播放光标的位置将保持不变， 因此调用 IDirectSoundBuffer8::SetCurrentPosition 在这个例子中，是为了保证播放从缓冲

区的开始进行。

使用流式缓冲区

流式缓冲区播放较长的声音时，不能一次将所有的声音数据都填充到缓冲区。因此在缓冲区播放以后，新的数据将覆盖旧的数据。

播放流式缓冲区要调用 IDirectSoundBuffer8::Play 这个方法,指出dwFlags参数中的 DSBPLAY_LOOPING 标志，要停止演奏通过调用 IDirectSoundBuffer8::Stop 这个方法将立即停止缓冲区的播放，因此你要确定所有的数据已经播放，这就要借助于查询播放光标的位置，或是设置通知消息标志。

流式缓冲区的操作步骤：

1. 探知缓冲区是否准备好接受新的数据，这就要借助于查询播放光标的位置，或是设置通知消息标志。
2. 通过调用IDirectSoundBuffer8::Lock 锁住缓冲区的一部分，这个方法将返回一个或两个可以去写数据的地址. 。
3. 将声音数据写到返回的地址中，同样是使用标准的内存拷贝函数。
4. 通过调用 IdirectSoundBuffer8::Unlock解锁 。

   这个 **IDirectSoundBuffer8::Lock** 可以返回两个地址，可以在缓冲区锁住任意数量的内存，与开始地址无关。如果你想从开始的位置锁住缓冲区，那么你可能就要对两块分开的内存块进行操作。

   例如：你想在一个有 40000 字节的缓冲区，锁住 30000 字节的内存，从开始偏移地址 20000 处开始，那么你调用 LCOK 后可能是：

- 内存的偏移地址是 20,000.
- 指针到缓冲区结尾只有 20000 个字节。
- 内存的偏移地址是 0.
- 从这个位置开始还有 10,000 bytes. .

虽然可以完全的锁住缓冲区，在播放的时候禁止那样去做！通常你每次只刷新一小部分，例如：你可以在缓冲区读其他 3/4 时，先锁住 1/4 的缓冲区。记住你不能在播放光标和写光标之间的部分进行访问。

下面这个函数是通过传递 *dwOffset* 来指定开始位置。

```
BOOL AppWriteDataToBuffer(
    LPDIRECTSOUNDBUFFER8 lpDsb,  // The buffer.
    DWORD dwOffset,              // Our own write cursor.
    LPBYTE lpbSoundData,         // Start of our data.
    DWORD dwSoundBytes)          // Size of block to copy.
{
  LPVOID  lpvPtr1;
```

```
DWORD dwBytes1;
LPVOID  lpvPtr2;
DWORD dwBytes2;
HRESULT hr;

// Obtain memory address of write block. This will be in two parts
// if the block wraps around.

hr = lpDsb->Lock(dwOffset, dwSoundBytes, &lpvPtr1,
    &dwBytes1, &lpvPtr2, &dwBytes2, 0);

// If the buffer was lost, restore and retry lock.

if (DSERR_BUFFERLOST == hr)
{
  lpDsb->Restore();
  hr = lpDsb->Lock(dwOffset, dwSoundBytes,
      &lpvPtr1, &dwBytes1,
      &lpvPtr2, &dwBytes2, 0);
}
if (SUCCEEDED(hr))
{
  // Write to pointers.

  CopyMemory(lpvPtr1, lpbSoundData, dwBytes1);
  if (NULL != lpvPtr2)
  {
    CopyMemory(lpvPtr2, lpbSoundData+dwBytes1, dwBytes2);
  }

  // Release the data back to DirectSound.

  hr = lpDsb->Unlock(lpvPtr1, dwBytes1, lpvPtr2,
  dwBytes2);
  if (SUCCEEDED(hr))
  {
    // Success.
    return TRUE;
  }
}

// Lock, Unlock, or Restore failed.

return FALSE;
```

}

播放控制

- 要得到正在播放的缓冲区的音量和设置音量时，你的应用程序可以使用 IDirectSoundBuffer8::GetVolume和IDirectSoundBuffer8::SetVolume 。

调用IDirectSoundBuffer8::GetFrequency 和 IDirectSoundBuffer8::SetFrequency 你可以设置正在播放的声音频率。但是主缓冲区的频率不能设定。

写光标与播放光标

DirectSound 保存了缓冲区的两个指针，即写光标和播放光标，并且它们的位置都是相对缓冲区的开始的偏移地址。

调用 IDirectSoundBuffer8::Play 这个方法总是从播放光标的位置开始播放。当缓冲区被创建时，播放光标就会自动初始化为零，缓冲区开始播放时，播放光标开始移动，并且总是指向播放位置。当播放停止，播放光标将保持不变。

写光标总是指向可以安全写入数据的位置，但是播放光标和写光标之间的部分， 不能去访问。

写光标随着播放光标的移动而移动，不是随着数据的写入而移动。如你的是流数据，你应该是自己的指针能表明应该写入的位置。

应用程序可以通过调用 IDirectSoundBuffer8::GetCurrentPosition 方法来返回播放光标和写光标， IDirectSoundBuffer8::SetCurrentPosition 它可以移动播放光标，但是不能移动写光标。

确保播放光标尽可能的被指出，总是在创建副缓冲区时指定 DSBCAPS_GETCURRENTPOSITION2 标志。

播放缓冲区通知消息

当播放光标到达缓冲区指定位置时或者停止，DirectSound 能通知应用程序。

- 应用程序完成一些操作，诸如当一段声音播放完之后，播放其他缓冲区。
- 应用程序是数据流缓冲区，需要知道什么时候到达预定的播放点，可以将后继数据覆盖上去。

注意：如果你利用声音管理，缓冲区通知消息可能不能到达。参看标志 DSBPOSITIONNOTIFY.

使用 IDirectSoundNotify8::SetNotificationPositions 方法，你能设置任意数量的

通知点，当缓冲区播放时，不能这样操作。

首先你应该提供一个指向 IDirectSoundNotify8 的接口。你可以利用缓冲区对象的**QueryInterface** 方法。在要设置通知的缓冲区必须提供一个分离的接口。

为每一个要通知的位置通过调用 Win32 **CreateEvent** 函数创建一个事件，然后将句柄赋值给 **DSBPOSITIONNOTIFY** 结构成员 **hEventNotify** ，结构的 **dwOffset** 成员是你要指定偏移缓冲的的相对位置。

下面是一个例子，创建缓冲区要用 DSBCAPS_CTRLPOSITIONNOTIFY 标志。

```
HRESULT SetStopNotification(HANDLE hMyEvent,
        LPDIRECTSOUNDBUFFER8 lpDsbSecondary)
{
  LPDIRECTSOUNDNOTIFY8 lpDsNotify;
  DSBPOSITIONNOTIFY PositionNotify;
  HRESULT hr;

  if (SUCCEEDED(
      hr = lpDsbSecondary->QueryInterface(IID_IDirectSoundNotify8,
            (LPVOID*)&lpDsNotify)))
  {
    PositionNotify.dwOffset = DSBPN_OFFSETSTOP;
    PositionNotify.hEventNotify = hMyEvent;
    hr = lpDsNotify->SetNotificationPositions(1, &PositionNotify);
    lpDsNotify->Release();
  }
  return hr;
}
```

通知在硬件缓冲区上可能是假的，因为驱动可能不做这样的事情，为了保证调用正确，你必须验证这个位置，通过调用 IDirectSoundBuffer8::GetCurrentPosition 方法。最好将缓冲区创建在软件中，这样就可以不用验证了。

混音

副缓冲区自动模拟播放混音效果，将处理后的数据放在主缓冲区，在 windows 驱动模式下混音是内核来完成的，不同的副缓冲区可以有不同的 wave 格式，在需要的情况下内核自动完成混音。

循环播放

DirectSound 在同一个缓冲区内不能直接支持循环播放， 或是重复一部分的工

作， (The DSBPLAY_LOOPING flag causes the entire buffer to play again when the play cursor reaches the end.) 在静态缓冲区可以在播放到结束时，将播放光标重置，来完成播放。

缓冲区管理

这个 IDirectSoundBuffer8::GetCaps 方法可以返回DirectSoundBuffer object.

应用程序可以使用 IDirectSoundBuffer8::GetStatus 方法来决定是否播放缓冲区，或是停止播放。使用 IDirectSoundBuffer8::GetFormat 方法可以返回缓冲区声音数据的相关信息，你也可以使用 IDirectSoundBuffer8::SetFormat 去设置主缓冲区的数据格式。参看 Mixing Sounds.

注： **SetFormat** 方法不能再副缓冲区上使用，在副缓冲区创建以后，他的格式已经固定，如果你想使用另一种格式去播放缓冲区，那么你必须从新创建一种新的缓冲区。

当缓冲区定位到声卡上，其他应用程序获得对硬件的控制权以后，容易造成内存的丢失, 在写主存写作级时，有可能丢失播放光标。当在执行 IDirectSoundBuffer8::Lock or IdirectSoundBuffer8::Play方法时，造成缓冲区丢失，方法将返回 DSERR_BUFFERLOST错误代码！当应用程序丢失自己写协作权限时，或是去焦点，其他程序试图再分配缓冲区通过调用 IDirectSoundBuffer8::Restore 方法就可以实现。如果调用成功，仍然要去写那些要播放的声音数据倒在分配的缓冲区。

使用 WAVE 格式的数据

在 Windows 驱动模式下 (WDM), DirectSound 硬件缓冲区能够播放任何无压缩或压缩个使的WAVEFORMATEX 或 WAVEFORMATEXTENSIBLE结构数据。软件缓冲区支持 8 位或 16 位无压缩的格式。Wave格式的数据通常存储在文件或RIFF里。这些数据包括一个 WAV格式的描述表，包括采样速率，输出通道等。

多通道和 WAV 格式

在 WDM 驱动上, DirectSound缓冲区支持有多个输出通道的WAV格式。这种个是可以被描述为WAVEFORMATEXTENSIBLE结构体，这个结构体是WAVEFORMATEX结构体的扩展，同时必须配置WAVEFORMATEX.cbSize.成员是扩展风格。

在程序中不能直接用 WAVEFORMATEXTENSIBLE。

DirectSound 再多通道上不支持音效，或 3-D 效果，试图去用 DSBCAPS_CTRL3D 和 DSBCAPS_CTRLFX 标志创建一个缓冲区，支持多通道将会失败！

Reading WAV Data、

Dsutil.cpp 这个文件中有例子。

5. 创建一个CSoundManager Sample Class的对象。
6. 调用 CSoundManager::Initialize 去创建一个设备对象。
7. 将文件名或资源传给 CSoundManager::Create 或 SoundManager::CreateFromMemory. 这些方法将返回 CSound Sample Class 的对象，代表一个或多个静态缓冲区；另一种是传递文件名或资源给 CSoundManager::CreateStreaming. 这个方法将返回一个 CStreamingSound Sample Class对象，代表着一个流式缓冲区。
8. 调用 FillBufferWithSound 方法在前述返回的对象上，这将开始读从文件中读数据到缓冲区；在流式缓冲区中，用 CStreamingSound::HandleWaveStreamNotification. 填充。
9. 注：实际读取数据的是 CWaveFile 对象，它是 CSound or CStreamingSound 保护成员。

Calculating the Duration of a WAV Sound

The length of time a waveform will play is determined by the data size and the format. The data size and format can be retrieved by using the CWaveFile::GetSize and CWaveFile::GetFormat methods in the DirectSound sample framework.

The following example function returns the duration of a WAV file, in milliseconds:

```
DWORD GetSoundLength(LPSTR strFileName)
{
  CWaveFile* pWav;
  DWORD dwLen = 0;
  DWORD dwSize;
  WAVEFORMATEX* wfx;

  pWav = new CWaveFile();
  if (SUCCEEDED(pWav->Open(strFileName, NULL, WAVEFILE_READ)))
  {
    wfx = pWav->GetFormat();
    dwSize = pWav->GetSize();
    dwLen = (DWORD) (1000 * dwSize / wfx->nAvgBytesPerSec);
    pWav->Close();
  }
  if (pWav) delete pWav;
  return dwLen;
}
```