DirectSound 开发指南

作者：智慧的鱼
编辑：中华视频网

中华视频网：http://www.chinavideo.org

佰锐科技：http://www.bairuitech.com

目 录

绪言

DirectSound 是微软多媒体技术 DirectX 的组成部分，封装了大量音频处理 API 函数，它可以提供快速的混音、硬件加速功能，并且可以直接访问相关设备，当然，最主要的是它提供的功能与现有的设备驱动程序保持兼容性。

DirectSound 允许进行波型声音的捕获、重放，也可以通过控制硬件和相应的驱动来获得更多的服务。

DirectSound 的优势当然和 DirectX 的其它组件一样——速度，它允许你最大效率的使用硬件，并拥有良好的兼容性。

中华视频网（www.chinavideo.org ）一直致力于语音视频技术的研究和推广，由于目前有关 DirectX 系列的编程资料比较少，最主要的参考资料还是 DirectX SDK 文档，特别是对于初学者来说，相对来说比较难以入门。早期也收编过多篇“智慧的鱼”的文章，鉴于本册《DirectSound 开发指南》是一篇比较实用的关于 DirectSound 编程手册，特收集整理成册，以供后来者学习。

非常感谢“智慧的鱼”前期的辛勤劳动和无私的分享精神！

中华视频网：www.chinavideo.org

Ffmpeg 工程组：www.ffmpeg.com.cn

佰锐科技：www.bairuitech.com

一、DirectSound 简介（Introduction to DirectSound）

曾经学习过 Directshow 的开发，对于 Dsound 一直没有仔细的莱学习，以前只是知道 Dsound 是做音频开发的，我一直以为它和 Dshow 的结构体系差不多，经过仔细学习后，发现，其实他们完全两码事。

闲话少说，下面我们看看 DirectSound 到底能帮我们做些什么。

1 播放 WAVE 格式的音频文件或者资源。
2 可以同时播放多个音频。
3Assign high-priority sounds to hardware-controlled buffers
4 播放 3D 立体声音
5 在声音中添加特技效果，比如回声，动态的改变特技的参数等
6 将麦克风或者其他音频输入设备的声音录制成 wave 格式的文件

呵呵，DirectSound 就能做这么多事情，读到这里，我都有点怀疑 DirectSound 是不是就是封装了 mmio 系列和 wav 系列的函数。因为这些底层的 API 也能够完成这些事情。

二、DirectSound 初体验（Getting Started with DirectSound）

在开始本节内容前，我会首先提醒一下，如果你想用Directsound开发，那么你首先要包含Dsound.h头文件，其实我可以实话告诉你，你仅仅包含dsound.h你的工程肯定调补通，其实下面的一些头文件也要包含，我第一次就搞了半天才搞好，

```
#include <windows.h>
#include <mmsystem.h>
#include <mmreg.h>
#include <dsound.h>
```

如果你还想使用 Dsound 的 API 的话，那么你就要在你的 vc 开发环境中添加 Dsound..lib 库，如果你的程序还提示有很多的外部链接找不到，那么我建议你可以将下面的库都添加到你的工程中 comctl32.lib dxerr9.lib winmm.lib dsound.lib dxguid.lib kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib，这些是我从 Dsound 提供的例子中得到的，肯定够你用的，ok，开发环境配置好了。

下面我们简单的来学习一下如果通过 Directsound 的 API 播放声音，既然是 breif overview，那么详细的内容你可以参考下面的一节内容，这里只是简单的介绍一下播放声音的步骤。

第一步，创建一个设备对象。

在你的代码中你可以通过调用 DirectSoundCreat8 函数来创建一个支持 IDirectSound8 接口的对象，这个对象通常代表缺省的播放设备。当然你可以枚举可用的设备，然后将设备的 GUID 传递给 DirectSoundCreat8 函数。

注意，Directsound 虽然基于 COM，但是你并不需要初始化 com 库，这些 Directsound 都帮你做好了，当然，如果你使用 DMOs 特技，你就要自己初始化 com 库了，切记。

第二步，创建一个辅助 Buffer，也叫后备缓冲区

你可以通过 **IDirectSound8::CreateSoundBuffer** 来创建 buffer 对象，这个对象主要用来获取处理数据，这种 buffer 称作辅助缓冲区，以和主缓冲区区别开来，Direcsound 通过把几个后备缓冲区的声音混合到主缓冲区中，然后输出到声音输出设备上，达到混音的效果。
第三步，获取 PCM 类型的数据
将 WAV 文件或者其他资源的数据读取到缓冲区中。
第四步，将数据读取到缓冲区
你可以通过 **IDirectSoundBuffer8::Lock**.方法来准备一个辅助缓冲区来进行写操作，通常这个方法返回一个内存地址，见数据从你的私人 buffer 中复制到这个地址中，然后调用 **IDirectSoundBuffer8::Unlock**.
第五步，播放缓冲区中的数据
你可以通过 **IDirectSoundBuffer8::Play** 方法来播放缓冲区中的音频数据，你可以通过 **IDirectSoundBuffer8::Stop** 来暂停播放数据，你可以反复的莱停止，播放，音频数据，如果你同时创建了几个 buffer，那么你就可以同时来播放这些数据，这些声音会自动进行混音的。

呵呵，简单介绍到这里的，如果想深入了解，请继续参考下一部分。

三、DirectSound 实用开发技巧 Using DirectSound

在进行这部分之前，我们首先学习一下 Directsound 中常用的几个对象，简单学习一下哦

对象	数量	作用	主要接口
设备对象	每个应用程序只有一个设备对象	用来管理设备，创建辅助缓冲区	IDirectSound8
辅助缓冲区对象	每一个声音对应一个辅助缓冲区，可以有多个辅助缓冲区	用来管理一个静态的或者动态的声音流，然后在主缓冲区中混音	IDirectSoundBuffer8, IDirectSound3DBuffer8, IDirectSoundNotify8
主缓冲区对象	一个应用程序只有一个主缓冲区	将辅助缓冲区的数据进行混音，并且控制 3D 参数.	IDirectSoundBuffer, IDirectSound3DListener8
特技对象	没有，或者	来辅助缓冲的声音数据进行处理	8 个特技接口 IDirectSoundFXChorus8

首先，要创建一个设备对象，然后通过设备对象创建 buffer 对象。辅助缓冲区由应用程序创建和管理，DirectSound 会自动地创建和管理主缓冲区，一般来说，应用程序即使没有获取这个对象的接口也可以播放音频数据，但是，如果应用程序要想得到 **IDirectSound3DListener8** 接口，就必须要自己创建一个主缓冲区。
我们可以将短小的声音文件全部读取到辅助缓冲区中，然后通过一个简单的命令来播放。如果声音文件很长，就必须采用数据流了。

3.1Dsound 设备对象(DirectSound Devices)

本节主要讲述下面的几个内容

1 如何枚举系统输出声音的设备

2 创建设备对象

3 设置声音设备的协作度

–

下面首先看看如何枚举系统中的声音输出设备

1 如何枚举系统输出声音的设备

如果你的应用程序使用用户首选的输出设备来输出声音，那么你就没有必要来枚举所有的输出设备，如果你通过 DirectSoundCreat8 函数来创建一个设备对象的同时，就给这个对象指定了一个缺省的设备，当然如果遇到下面的一些情形，你就要来枚举设备对象

例如，你的应用程序并不支持所有的输出设备，或者你的应用程需要两个或者多个设备，或者你希望用户自己来选择输出设备。

枚举设备，你首先要定义一个回调函数，这个回调函数可以被系统中的每个设备来调用，你可以在各函数做任何事情，这个函数的命名也没有任何的限制，但是函数应该以 **DSEnumCallback** 为原型，如果枚举没有结束，这个回调函数就返回 TRUE，如果枚举结束，例如你找到合适的设备，这个函数就要返回 FALSE。

下面是回调函数的一个例子，这个函数将枚举的每一个设备都添加到一个 combox 中，将设备的 GUID 保存到一个 item 中，这个函数的前三个参数由设备的驱动程序提供，第四个参数有 **DirectSoundEnumerate** 函数提供，这个参数可以是任意的 32 位值，这个例子里是 combox 的句柄，

```
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);
}
```

枚举设备通常都是在对话框初始化的时候才进行的，我们假设 hCombo 就是 combox 句柄，hDlg 就对话框的句柄，看看我们怎么来枚举设备的吧

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

在这个例子中，combox 的句柄作为参数传递到 DirectSoundEnumerate 函数中，然后又被传递到回调函数中，这个参数你可以是你想传递的任意的 32 位值。

注：第一个被枚举的设备通常称为 Primary sound driver，并且回调函数的 lpGUID 为 NULL，这个设备就是用户通过控制面板设置的缺省的输出声音设备，

2 创建设备对象

创建设备对象最简单的方法就是通过 **DirectSoundCreate8** 函数，这个函数的第一个参数指定了和这个对象邦定的设备的 GUID，你可以通过枚举设备来获取这个设备的 GUID，你可以传递一个下面的参数来指定一个缺省的设备

DSDEVID_DefaultPlayback 缺省的系统的声音输出设备，这个参数也可以为 NULL

SDEVID_DefaultVoicePlayback，缺省的声音输出设备，通常指第二缺省设备，例如 USB 耳机麦克风

如果没有声音输出设备，这个函数就返回 error，或者，在 VXD 驱动程序下，如果声音输出设备正被某个应用程序通过 waveform 格式的 api 函数所控制，该函数也返回 error，

下面是创建对象的代码，及其简单

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

如果你想通过表准的 COM 调用来创建设备对象，下面我就给出代码，你可以比较一下

```
HRESULT hr = CoInitializeEx(NULL, 0);
if (FAILED(hr))
{
    ErrorHandler(hr);    // Add error-handling here.
}
LPDIRECTSOUND8 lpds;
hr = CoCreateInstance(&CLSID_DirectSound8,
                      NULL,
                      CLSCTX_INPROC_SERVER,
                      IID_IDirectSound8,
                      (LPVOID*) &lpds);
if (FAILED(hr))
{
    ErrorHandler(hr);    // Add error-handling here.
}
hr = lpds->Initialize(NULL);
if (FAILED(hr))
```

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

CoUninitialize();
```

3 设置声音设备的协作度

因为 Windows 是一个多任务操作环境，在同一个时刻有可能多个应用程序共用同一个设备，通过协作水平，DirectX 就可以保证这些应用程序在访问设备的时候不会冲突，每个 Directsound 应用程序都有一个协作度，用来确定来接近设备的程度，

当你创建完设备对象后，一定要调用 **IDirectSound8::SetCooperativeLevel** 来设置协作度，否则，你不会听到声音的，

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

Hwnd 参数代表了应用程序的窗口。

DirectSound 定义了三种水平，**DSSCL_NORMAL, DSSCL_PRIORITY, and DSSCL_WRITEPRIMARY**

在 NORMAL 水平的协作度下，应用程序不能设置主缓冲区的格式，也不能对主缓冲区进行写操作，所有在这个层次的应用程序使用的媒体格式都是 22kHZ，立体声，采样精度 8 位，所以，设备可以在各个应用程序中任意的切换。

在 Priority 层次的协作度下，应用程序可以有优先权使用硬件资源，比如使用硬件进行混音，当然也可以设置主缓冲区的媒体格式，游戏程序应该采用这个层次的协作度，这个层次的协作度在允许应用程序控制采用频率和位深度的同时，也给应用程序很大的权力，这个层次的协作度允许其他应用程序的声音和游戏的音频同时被听到，不影响。

最高层次的协作度就是 Write_primary，当采用这个层次的协作度来使用一个 dsound 设备时，在非 WDM 模式驱动下，你的应用程序可以直接操作主缓冲区，在这个模式下，应有程序必须直接的操作主缓冲区。如果你想直接把音频数据直接写入到主缓冲区，你就要将你的协作度模式设置为 writeprimary，如果你的应用程序没有设置到这个层次，那么所有的对主缓冲区的 **IDirectSoundBuffer::Lock 都会失败。**

当你的应用程序的协作度设置为 wirte_primary 水平时，并且你的应用程序 gains the foreground，那么其他应用程序的辅助缓冲区就会停止，并且标示为 lost，当你的应用程序转到 background，那么它的主缓冲区就会标示为 lost，当它再次回到 foreground 的时候会恢复到原来的状态，如果你的设备没有出现在你的系统中，那么你就没法设置 wirte primary 协作度了，所以，在设置协作度之前，可以通过 **IDirectSound8::GetCaps** 方法来查询一下设备是否可用。

4 设置声音设备（扬声器）

在 windowos98 或者 2000 系统中，用户可以通过控制面板来设置扬声器的属性，应用程序可以通过 **IDirectSound8::GetSpeakerConfig**.函数来获取这些属性，我们不建议应用程序通过 **IDirectSound8::SetSpeakerConfig** 函授来设置扬声器的属性，因为这些设置的改动会影响到其他用户或者应用程序。

5 查询输出设备的性能

你的应用程序可以通过 Directsound 来检查声音设备的性能，然而许多应用程序并不需要这么做，因为 Directsound 会自动地利用硬件提供的较好的性能，并不需要你手动地来选择。但是，However, high-performance applications can use the information to scale their sound requirements to the available hardware. For example, an application might choose to play more sounds if hardware mixing is available than if it is not.

当你调用 **DirectSoundCreate8** 创建一个设备对象后，你的应用程序就可以通过 **IDirectSound8::GetCaps** 函数来获取设备的性能属性，下面的代码演示了这个过程

```
DSCAPS dscaps;

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

通过 **DSCAPS** 结构，该函数就给我返回硬件设备的一些性能，但是在调用这个函数之前一定要初始化这个结构的 dwSize 成员。

如果你的应用程序能够操作硬件设备，比如你的协作度是 Write_primory 级别的，那么你在硬件设备上分配内存的时候，就要调用 IDirectSoudn8::GetCaps 来查看一下硬件的资源是否满足你的要求。

3.2Dsound 的 buffer 对象(DirectSound Buffers)

在存储和播放几个音频流的时候，你的应用程序要给每一个音频流都要创建一个辅助缓冲区（buffer）对象。

辅助缓冲区可以和应用程的生命期一样的长，也可以在不需要的时候销毁。辅助缓冲区可以是一个包含了整个声音数据的静态缓冲区，也是可以只包含声音数据的一部份，然后再播放时不断更新数据的流缓冲区。为了限制内存开销，在播放比较长的声音文件时要采用流缓冲区，这些缓冲区只包含几秒钟的数据量。

你可以通过同时播放几个辅助缓冲区中的声音来对他们进行混音，至于同时可以播放几个辅助缓冲区则有硬件设备的性能决定。

辅助缓冲区的格式并不完全一样，一般用来描述缓冲格式的参数如下

Format，缓冲区的 format 必须要所播放音频的 waveformat 一致。

Controls，不同的缓冲区的控制参数的值可以不一样，比如音量，频率，以及在不同方向的移动，当创建 buffer 时，你就要指定你需要的控制参数的值，例如，不要为一个不支持 3D 的音频创建一个 3D 缓冲区

Location，你创建的缓冲区可以在硬件管理的内存中，也可以在软件管理的内存中，当然，硬件缓冲区比软件缓冲区速度要快。

你可以调用 **IDirectSound8::CreateSoundBuffer** 函数来创建一个缓冲区（buffer）对象。这个函数返回一个指向 IDrectSoundBuffer 接口的指针，通过这个接口，应用程序可以获取 **IDirectSoundBuffer8** interface.

下面的一段代码演示了如何创建一个辅助缓冲区，并且返回一个 IDirectSoundBuffer8 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;
//   流 buffer
   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;
}
```

这个例子中创建了一个可以持续播放 3 秒钟的流缓冲区，如果你要创建静态的缓冲区，你就要创建的 buffersize 正好能够容下你的音频数据即可。

如果缓冲区的位置没有指定，DirectSound 在条件允许的情况下将你的缓冲区设置为硬件控制，因为硬件缓冲区的混音是通过声卡的加速器来进行的，受应用程序的影响较小。

如果你想自己控制你创建的缓冲区（buffer）的位置，那么你一定要将 **DSBUFFERDESC** 中的 dsbdesc.dwFlags 成员变量设置为 DSBCAPS_LOCHARDWARE 或者设置为

DSBCAPS_LOCSOFTWARE,如果设置为 DSBCAPS_LOCHARDWARE，此时硬件设备的资源不足时，缓冲区创建失败。

如果你想使用 DirectSound 的管理声音的特性，那么你创建缓冲区的时候一定要设置 DSBCAPS_LOCDEFER 标志，这个标志表示只有在播放的时候才分配内存，更多的细节，你可以参考动态的声音管理一节。

你可以通过 **IDirectSoundBuffer8::GetCaps** 方法来探明已经存在的缓冲区，并且可以检查该 buffer 的 dwFlags 设置情况。

缓冲区对象属于创建它的设备对象。当设备对象销毁时，它所创建的 buffer 对象也全部被销毁，没法被引用了。

你可以通过 **IDirectSound8::DuplicateSoundBuffer** 同时创建两个或者多个包含相同数据的辅助缓冲区，当然不允许复制主缓冲区。因为复制的缓冲区和原始的缓冲区共享内存，改变复制的缓冲区的数据的同时，也改变了原始缓冲区的内容。

下面我们看看 buffer 的控制属性

当你创建了缓冲区的时候，你的应用程序设置该缓冲区的控制属性，这是由 **DSBUFFERDESC** 结构的 dwFlags 成员来控制的。

下面我们来看看 dwFlags 的取值

DSBCAPS_CTRL3D 表示声音可以在 3 个方向上进行移动

DSBCAPS_CTRLFX ，表示可以在缓冲区中添加特技

DSBCAPS_CTRLFREQUENCY ,表示声音的频率可以被改动

DSBCAPS_CTRLPAN,表示声音可以从左声道被移动到右声道

DSBCAPS_CTRLPOSITIONNOTIFY，可以在 buffer 中设置通知的位置。

DSBCAPS_CTRLVOLUME，声音的大小可以被改变。

注意，有些标志位的组合是不允许的，具体的信息可以参见 DSBUFFERDESC 结构。

为了使得你的声卡更好的工作，你最好改变你应用程序用到的几个控制属性，其他的属性最好采用 DirectSound 来默认的设置。

DirectSound 通过控制属性来判断是否可以在硬件设备上分配缓冲区。例如，一个设备可能支持硬件缓冲区，但是不支持控制 pan，此时，当 DSBCAPS_CTRLVOLUME 标志没有设置的时候，DirectSound 就可以使用硬件加速。

如果你的应用程序想调用 buffer 不支持的控制项，那么调用就会失败，例如，如果你想通过 **IDirectSoundBuffer8::SetVolume** 方法来设置音量，只有在创建 buffer 时设置了 DSBCAPS_CTRLVOLUME 标志这个函数调用才会成功。

当你创建的辅助缓冲区的控制标志位被设置为 DSBCAPS_CTRL3D 时，如果该 buffer 创建的位置由软件控制，那么你就可以给你创建的缓冲区指定一个 3D 你的声音的算法，缺省的情况下，采用的是 HRTF（no head-related transfer function）算法来处理 3D 化声音。

下面我们看看如何两种 buffer 如何播放声音的，先看看静态的缓冲区吧。

包含全部音频数据的缓冲区我们称为静态的缓冲区，尽管，不同的声音可能会反复使用同一个内存 buffer，但严格来说，静态缓冲区的数据只写入一次。

静态缓冲区的创建和管理和流缓冲区很相似，唯一的区别就是它们使用的方式不一样，静态缓冲区只填充一次数据，然后就可以 play，然而，流缓冲区是一边 play，一边填充数据。

给静态缓冲区加载数据分下面几个步骤

1， 调用 **IDirectSoundBuffer8::Lock** 函数来锁定所有的内存，你要指定你锁定内存中你开始写入数据的偏移位置，并且取回该偏移位置的地址。

2， 采用标准的数据 copy 方法，将音频数据复制到返回的地址。

3， 调用 **IDirectSoundBuffer8::Unlock**.，解锁该地址。

下面的例子演示了上面提到的几个步骤，lpdsbStatic 是指向静态 buffer 的指针

```
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.
}
```

将数据加载到缓冲区中就可以播放，调用 **IDirectSoundBuffer8::Play** 方法。如下：

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

因为这个例子中没有设置 DSBPLAY_LOOPING 标志，当 buffer 到达最后时就会自动停止，你也可以调用 **IDirectSoundBuffer8::Stop** 方法来停止播放。如果你提前停止播放，播放光标的位置就会被保存下来，因此，例子中 **IDirectSoundBuffer8::SetCurrentPosition** 方法就是为了保证从头开始播放。

下面我们看看流缓冲区的用法

流缓冲区用来播放那些比较长的声音，因为数据比较长，没法一次填充到缓冲区中，一边播放，一边将新的数据填充到 buffer 中。

可以通过 **IDirectSoundBuffer8::Play** 函授来播放缓冲区中的内容，注意在该函数的参数中一定要设置 DSBPLAY_LOOPING 标志。

通过 **IDirectSoundBuffer8::Stop** 方法中断播放，该方法会立即停止缓冲区播放，因此你要确保所有的数据都被播放，你可以通过拖动播放位置或者设置通知位置来实现。

将音频流倒入缓冲区需要下面三个步骤

1 确保你的缓冲区已经做好接收新数据的准备。你可以拖放播放的光标位置或者等待通知

2 调用 **IDirectSoundBuffer8::Lock**.函数锁住缓冲区的位置，这个函数返回一个或者两个可以写入数据的地址

3 使用标准的 copy 数据的方法将音频数据写入缓冲区中

4 **IDirectSoundBuffer8::Unlock**.，解锁

IDirectSoundBuffer8：：Lock 可能返回两个地址的原因在于你锁定内存的数量是随机的，有时你锁定的区域正好包含 buffer 的起始点，这时，就会给你返回两个地址，举个例子吧

假设你锁定了 30,000 字节，偏移位置为 20，000 字节，也就是开始位置，如果你的缓冲区的大小为 40，000 字节，此时就会给你返回四个数据

1 内存地址的偏移位置 20，000，

2 从偏移位置到 buffer 的最末端的字节数，也是 20，000，你要在第一个地址写入 20，000 个字节的内容

3 偏移量为 0 的地址

4 从起始点开始的字节数，也就是 10，000 字节，你要将这个字节数的内容写入第二个地址。如果不包含零点，最后两个数值为 NULL 和 0，

当然，你也有可能锁定 buffer 的全部内存，建议你在播放的时候不要这么做，通过你只是更新所有 buffer 中的一部份，例如，你可能在播放广标到达 1/2 位置前要将第一个 1/4 内存更新成新的数据，你一定不要更新 play 光标和 Write 光标间的内容。

```
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;
}
```

下面我们看看如何控制播放的属性

你可以通过 **IDirectSoundBuffer8::GetVolume** and **IDirectSoundBuffer8::SetVolume** 函数来获取或者设置正在播放的音频的音量的大小。

如果设置主缓冲区的音量就会改变声卡的音频的声量大小。音量的大小，用分贝来表示，一般没法来增强缺省的音量，这里要提示一下，分贝的增减不是线形的，减少 3 分贝相当于减少 1/2 的能量。最大值衰减 100 分贝几乎听不到了。

通过 **IDirectSoundBuffer8::GetFrequency** and **IDirectSoundBuffer8::SetFrequency** 方法可以获取设置音频播放的频率，主缓冲区的频率不允许改动，

通过 **IDirectSoundBuffer8::GetPan** and **IDirectSoundBuffer8::SetPan** 函数可以设置音频在左右声道播放的位置，具有 3D 特性的缓冲区没法调整声道。

下面要看看混音。

如果辅助缓冲区中的音频同时播放就会主缓冲区自动的混音，在 WDM 驱动模式下，混音的工作由核心混音器来完成，不同的辅助缓冲区可能具有不同的 WAV 格式（例如，不同的采样频率），在必要的时候，辅助缓冲区的格式要转换成主缓冲区，或者核心混音器的格式。

在 VXD 驱动模式下，如果你的辅助缓冲区都采用相同的音频格式，并且硬件的音频格式也和你的音频格式匹配，此时，混音器不用作任何的转换。你的应用程序可以创建一个主缓冲区，然后通过 **IDirectSoundBuffer8::SetFormat** 来设置硬件的输出格式。要注意，只有你的协作度一定要是 Priority Cooperative Level.，并且，一定要创建辅助缓冲区前设置主缓冲区，DirectSound 会将你的设置保存下来。

在 WDM 模式下，对主缓冲区的的设置没有作用，因为主缓冲区的格式是由内核混音器来决定的。

循环播放；

Directsound 并不支持在 buffer 内部的循环或者声音的一部份循环，DSBPLAY_LOOPING 标志是整个 buffer 到 end 处然后重新从头的播放，如果你在静态的缓冲区的播放到末端后然后将 play 光标设置到起始位置重新播放，会导致 audio glitches，因为 DirectSound 必须忽略所有的预处理的数据。所以，如果要循环播放，一定要在 stream buffer 中进行。

最后，我们来谈一下缓冲区管理。

通过 **IDirectSoundBuffer8::GetCaps** 方法我们可以获取 DirectSoundBuffer 对象的一些属性。应用程序可以通过 **IDirectSoundBuffer8::GetStatus** 方法还获取 buffer 是播放还是停止状态。

通过 **IDirectSoundBuffer8::GetFormat** 方法可以获取 buffer 中音频数据的格式，你也可以调用 **IDirectSoundBuffer8::SetFormat** 方法来主缓冲区的数据格式。

Sound Buffer 中的一些数据在某些条件下有可能丢失，例如，如果 buffer 位于声卡的内存中，此时其他的应用程序获取硬件的控制权并请求资源。当具有 Write_primary 权限的应用程序 moves to forground，此时，Directsound 就会使其他的 buffer 内容丢失才能够让 foreground 的应用程序直接向主缓冲区中写数据。

当 **IDirectSoundBuffer8::Lock** or **IDirectSoundBuffer8::Play** 方法向一个 lost buffer 进行操作时，就会返回一个错误码。当造成 buffer 丢失的应用程序降低协作度，低于 write_primary，或者 moves to background，其他的应用程序可以调用 **IDirectSoundBuffer8::Restore** 来重新分配内存，如果成功，这个方法就会恢复内存中的内容以及对该内存的设置，但是，恢复的缓冲区中不包含合法的数据，因此，应用程序要重新向该 buffer 中填写数据。

3.3Using WAV Data

在 WDM 驱动模式下，DirectSound 缓冲区支持如下 WAV 格式：多声道，多个扬声器配置，例如 5.1，在前左，前中，前右，后左，后右都有扬声器，超重低音。也支持多于 16 的采样精度。

这种格式可以用 **WAVEFORMATEXTENSIBLE** 结构来描述，这个结构是 **WAVEFORMATEX** 的扩展，

对于多声道，DirectSound 并不支持 3D。

下面我讲一下如何计算 wave 的播放时间

Wave 格式的播放长度由数据大小和格式决定，可以通过 CWaveFile：：GetSize 和 CWaveFile：：GetFormat 来获取数据的大小和格式

下面的代码告诉你如何计算 wave 的总播放时间以毫秒为时间单位

```
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;
}
```

3.43-D Sound

2005.11.23 补充了 3D

首先来讲一下 3D 空间

一个典型应用的实现只需要不多的代码，一般来说，首先必须启动 DirectSound，然后就是创建音效缓存，包括主缓存和次级缓存：主缓存是 DirectSound 必有的，主要用来表现听者（the listener）在 3D 听觉环境中的情况，如位置、速度、听者方向（向上、下、左、右看等）等等；次级缓存则可以有很多，主要用来表现音源（每一个次级缓存对应一个音源）的情况，如速度、音锥（sound cones）、与听者的最大/最小距离等等。音效缓存创建后，还要为每一个缓存创建三个界面：标准界面控制基本的缓存行为（回放音量、频率等），

在 3D 空间，声源和听者的位置，速度，以及方向可以通过笛卡儿坐标系来描述，分别用三个坐标轴，x，y，z 轴来表示。X 方向的值从左到右增加，y 轴方向表示从下到上，z 轴表示从进到远。

D3DVECTOR 结构包含了三个变量用来描述位置，速度，方向在三个轴上的大小，

对于位置来说，缺省的单位是 m，当然你也可以选择其他的长度单位。

对于速率来说，表示沿着轴方向每秒运动的距离，缺省的单位用 m/s

对于方向来说，方向的值是一个相对值，如果 3D 世界的基准视图是水平，朝向北方，如果听者的坐标为（-1，0，1），那么，听者就是面向西北方向的，因为，vector 中值不是一个绝对值，而是相对的，例如，(-5,0,-5) 和（-2.5, 0, 2.5）是相等的。

在平面坐标系中，如果我们从(0,0) 到（1，1）画直线和从（0，0）到(5,5)画直线的斜率都是一样的，只是后一条直线的距离要远一些，

在现实世界中，声音跟下面的几个因素有关，这些因素并不都是听觉上的，最重要的因素还是视觉。

下面看看都有哪些因素；

强度：当声源逐渐远离听者的时候，他所感觉到声音的强度按照一定的比例下降，这种现象称为 Rolloff。

耳间强度差异，当声音从听者的右边传来的时候，右边耳朵就会感觉声音比左边耳朵的声音大一些。

耳间时间差异，

DirectSound 3-D Buffers

在 3D 环境中，我们通过 **IDirectSound3DBuffer8** 接口来表述声源，这个接口只有创建时设置 DSBCAPS_CTRL3D 标志的 Directsound buffer 才支持这个接口，这个接口提供的一些函数用来设置和获取声源的一些属性。

应用程序在使用 Directsound 3D 音效时，一定要提供单声道的。如果你试图创建 3D buffer，并且设置多声道的 WVE 格式，就会出错。

要想获取 3D buffer 接口，首先要创建一个辅助缓冲区，然后给这个缓冲区设置 DSBCAPS_CTRL3D 标志，然后通过这个辅助缓冲区接口，

如果你不是采用的 DirectMusic，而是自己的程序中创建 和管理 Directsound 的辅助缓冲区，那么你通过下面的代码获取 IDirectSoundBuffer8 接口指针。

```
LPDIRECTSOUND3DBUFFER8 lpDs3dBuffer;
HRESULT hr = lpDsbSecondary->QueryInterface(IID_IDirectSound3DBuffer8,
                    (LPVOID *)&lpDs3dBuffer);
```

最大和最小的距离

当听者越接近声源，那么听到的声音就越大，距离减少一半，音量会增加一倍。但是，当你继续接近到声源，当距离缩短到一定距离后，音量就不会持续增加。这就是声源的最小距离。

声源的最小距离，就是声音的音量开始随着距离大幅度衰减的起始点。例如，对于飞机，这个最小距离也许是 100m，但是对于蜜蜂，这个最小距离是 2 cm，根据这个最小距离，当听者距离飞机 400m 时，声音的音量就要衰减一半，对于蜜蜂来说，当超过 4cm 的时候，音量衰减一半。下面这个图表现了，最小和最大距离对飞机和蜜蜂音量的影响。

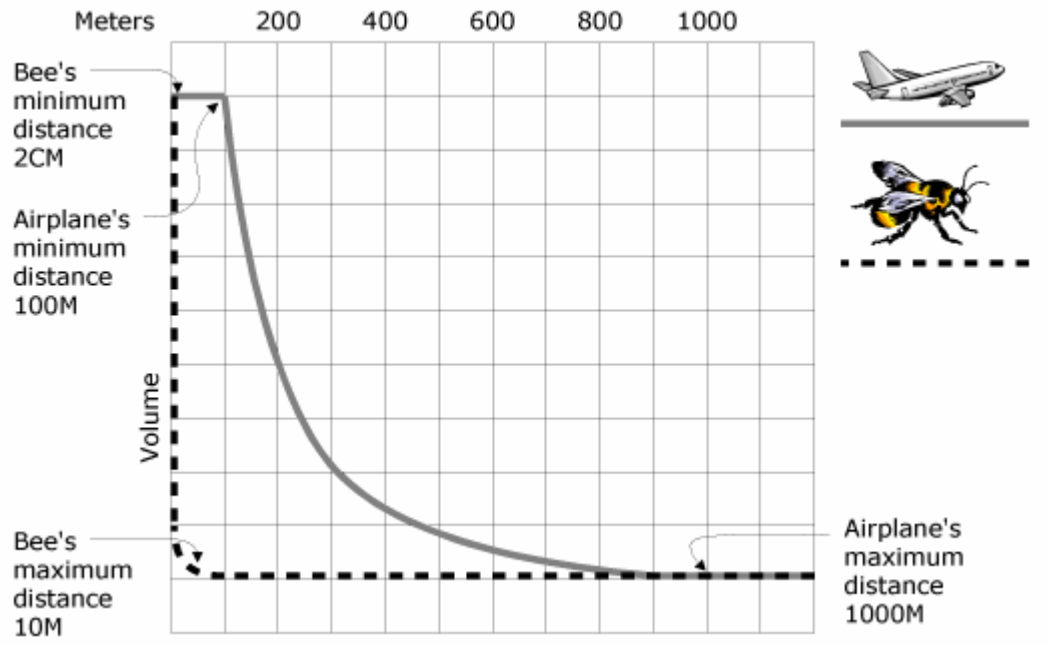
Directsound 的缺省的最小距离 DS3D_DEFAULTMINDISTANCE 定义为 1 个单位，或者是 1 米。我们规定，声音在 1 米处的音量是 full volume，在 2 米处衰减一半，4 米处衰减为 1/4，一次类推。对于大多数声音来说，我们要设置一个比较大的最小距离，这样，当声音运动的时候，不至于衰减的这么快。

最大距离，就是就是声源的音量不再衰减的距离，我们称为声源的最大距离。对于 Directsound 3D buffer 缺省的最大的距离 DS3D_DEFAULTMAXDISTANCE 是 1 billion。 也就是说，当

声音超出我们的听觉范围以外的时候，衰减还是在继续。在 VXD 驱动下，为了避免不必要的计算处理，我们在创建 buffer 的时候就要设置一个合理的最大距离。

最大距离同时也用来避免某种声音听不到。例如，如果你将某种声音的最小距离设置为 100m，那么声音可能在 1000m 的处衰减的可能就听不到了，你可以将最大的距离设置为 800m，这样你就可以保证声音在无论多远处都为原音量的 1/10。

记得，缺省的单位是 m。



处理模式

Sound buffers 有三种处理模式，normal, head-relative, and disabled.

在正常模式下，声源的位置和方向是真实世界中的绝对值，这种模式适合声源相对于听者不动的情形。

在 head reative 模式下，声源的所有 3D 特性都跟听者的当前的位置，速度，以及方向有关，当听者移动，或者转动方向，3D buffer 就会自动的重新调整 world space 。这种模式可以应用实现一种在听者头部不停的嗡嗡叫的声音，那些一直跟随着听者的声音，根本没有必要用 3D 声音来实现。

在 disable 模式下，3 D 声音失效，所有的声音听起来好像发自听者的头部。

Buffer 的位置和速度

当一个声源发生移动时，应用程序就要检查该 buffer 的位置和速度。

位置可以根据现实世界和听者的相对值以及处理模式，通过三个轴方向的值大小来度量。

速度就是在三个轴方向的每秒移动的距离。

声音的锥效应

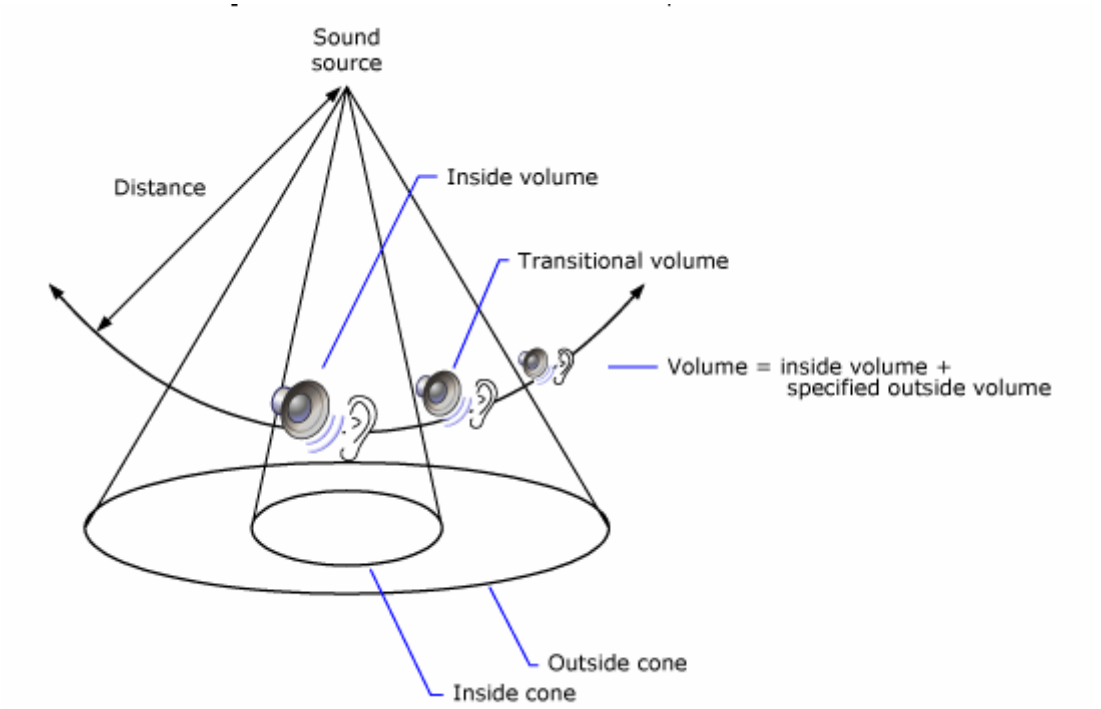
没有方向的声音在各个方向上的振幅都相同，有方向的声音在该方向上的振幅最大，声音的锥效应分为内部的锥效应，和外部的锥效应，锥的外部的角度应该大于等于锥的内部角度。

在锥的内部，在考虑到 buffer 中的基本的音量，以及距离听者的距离远近，何听者的方向，声音的音量就跟没有锥效应一样。

在锥的外部，正常的音量被削弱了，从 0 到负的百分之几分贝。

在锥体的内部和外部之间，是一个过渡带，从内部的 volume 到外部的 volume，这个音量

的逐渐的降低。下面的图表显示了声音的锥效应。



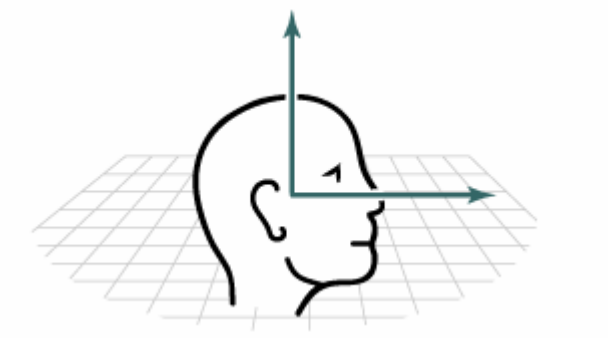
每一个的 3D buffer 都有一个声音锥，但是缺省的情况下，每一个 3D 声音 buffer 都像一个没有衰减的 声源，因为锥体内外都没有声音的音量的降低，锥的角度内外的角度是 360 度，除非我们的应用程序改变这个设置，否则的话，3D 声音没有方向感。

利用 3D 声音，可以给你的程序添加比较奇异的特技，例如，你可以将声音源放到房间的中间，将声源的方向朝向门的方向，然后将声音的锥体角度包含门的宽度，将锥体的外部设置的更宽一些，然后将锥体的外部音量设置为 0。这样，当听者只有在门的附近才能听到声音，在正对门的位置，声音才更响。

DirectSound 3-D Listeners

在一个虚拟的 3D 环境中，声音只和听到声音的位置有关，
在 Directx 应用程序中，影响 3D 特技的因素不只有声源的位置，方向，速度，并且和虚拟的听者的位置，速度，方向有关系。

缺省的情况下，听者固定在各种向量的零点，鼻子朝向 z 轴的正方向，头的顶部朝向 Y 轴的正方向，应用程序可以改变这些参数来改变虚拟世界中听者的位置和运动，因为虚拟的听者控制着声学环境中的多数参数，比如多普勒变换的数量，音量衰减的比率。



象声源一样，3D 世界中的听者也有位置，速度和方向。

听者的方向由通过听者头部的两个方向向量决定，如上图，向上的向量指向头的顶部，前向的向量指向听者的面部正前方，

缺省时，front 向量是(0.0, 0.0, 1.0),，顶部的向量是(0.0, 1.0, 0.0).，如果需要的话，Directsound 可以调整 front 向量，因为它在 top 向量的右向。

如何来获取 3D 听者呢

我们可以通过主缓冲区来获取 **IDirectSound3DListener8** 接口，我们的应用程序应该创建一个主缓冲区对象，然后获取 listerner 指针 通过 **IDirectSound8::CreateSoundBuffer** 创建一个主缓冲区，

```
GetListener(LPDIRECTSOUND8 lpds, LPDIRECTSOUND3DLISTENER8* ppListener)
{
   DSBUFFERDESC                dsbd;
   LPDIRECTSOUNDBUFFER         lpdsbPrimary;    // Cannot be IDirectSoundBuffer8.
   LPDIRECTSOUND3DLISTENER8 lp3DListener = NULL;
   HRESULT hr;

   ZeroMemory(&dsbd, sizeof(DSBUFFERDESC));
   dsbd.dwSize = sizeof(DSBUFFERDESC);
   dsbd.dwFlags = DSBCAPS_CTRL3D | DSBCAPS_PRIMARYBUFFER;

   if (SUCCEEDED(hr = lpds->CreateSoundBuffer(&dsbd, &lpdsbPrimary, NULL)))
   {
      hr = lpdsbPrimary->QueryInterface(IID_IDirectSound3DListener8,
                                        (LPVOID *)ppListener);
      lpdsbPrimary->Release();
   }
   return hr;
}
```

3.5 增加声音特技 Using Effects

DirectX 通过 DirectX Media Objects（DMOs）来支持在声音中添加特技。为了使用特技，DirectSound 应用程序必须首先 CoInitialize 来初始化 Com 库，当然也不排除通过 DirectSoundCreate8 来创建设备对象。

对于在 DSBUFFERDESC 结构中设置 DSBCAPS_CTRLFX 标志的辅助缓冲区，你可以在该缓冲区添加任意的特技，但是，buffer 一定要处于停止状态并且没有被锁定，在 DiectX 9.0 中，特技不依靠硬件来加速。当然也可以在硬件缓冲区中设置特技，但这样做没有任何的好处。

对于很小的缓冲区，特技也许并不能好好工作，DirectSound 不提倡在小于 150 毫秒的数据缓冲区中使用特技，这个值被定义为 DSBSIZ_FX_MIN。

下面的代码在缓冲设置了回声特技。

```
HRESULT SetEcho(LPDIRECTSOUNDBUFFER8 pDSBuffer)
{
   HRESULT hr;
   DWORD dwResults[1];   // One element for each effect.

   // Describe the effect.
   DSEFFECTDESC dsEffect;
   memset(&dsEffect, 0, sizeof(DSEFFECTDESC));
   dsEffect.dwSize = sizeof(DSEFFECTDESC);
   dsEffect.dwFlags = 0;
   dsEffect.guidDSFXClass = GUID_DSFX_STANDARD_ECHO; //设置特技

   // Set the effect
   if (SUCCEEDED(hr = pDSBuffer->SetFX(1, &dsEffect, dwResults)))
   {
      switch (dwResults[0])
      {
         case DSFXR_LOCHARDWARE:
            OutputDebugString("Effect was placed in hardware.");
            break;
         case DSFXR_LOCSOFTWARE:
            OutputDebugString("Effect was placed in software.");
            break;
         case DSFXR_UNALLOCATED:
            OutputDebugString("Effect is not yet allocated to hardware or software.");
            break;
      }
   }
   return hr;
}
```

为了获取或者设置特技的参数，首先你要从包含特技的缓冲区对象中获取相应的特技接口

指针。下面列出 DirectSound 支持的特技接口。

- IDirectSoundFXChorus8
- IDirectSoundFXCompressor8
- IDirectSoundFXDistortion8
- IDirectSoundFXEcho8
- IDirectSoundFXFlanger8
- IDirectSoundFXGargle8
- IDirectSoundFXI3DL2Reverb8
- IDirectSoundFXParamEq8
- IDirectSoundFXWavesReverb8

下面我简单介绍一下标准的特技
Chorus，
Compression
Distortion
Echo
Environmental Reverberation
Flange
Gargle
Parametric Equalizer
Waves Reverberation

3.6 录制 Capturing Waveforms

1 枚举录音的设备
如果你的程序只是想从用户缺省的设备上进行声音的录制，那么就没有必要来枚举出系统中的所有录音的设备，当你调用 **DirectSoundCaptureCreate8** 或者另外一个函数 **DirectSoundFullDuplexCreate8** 的时候，其实就默认指定了一个缺省的录音设备。
当然，在下面的情况下，你就必须要枚举系统中所有的设备，

```
DWORD pv;    // Can be any 32-bit type.

HRESULT hr = DirectSoundCaptureEnumerate(
      (LPDSENUMCALLBACK)DSEnumProc, (VOID*)&pv);
```

2 创建设备对象
你可以通过 **DirectSoundCaptureCreate8** 或者 **DirectSoundFullDuplexCreate8** 函数直接创建设备对象，该函数返回一个指向 **IDirectSoundCapture8** 接口的指针

3 录音设备的性能
你可以通过 **IDirectSoundCapture8::GetCaps** 方法来获取录音的性能，这个函数的参数是一个 **DSCCAPS** 类型的结构，在传递这个参数之前，一定要初始化该结构的 dwSize 成员变量。同时，你可以通过这个结构返回设备支持的声道数，以及类似 **WAVEINCAPS 结构的其他设备属性**
4 创建录音的 buffer

我们可以通过 **IDirectSoundCapture8::CreateCaptureBuffer** 来创建一个录音的 buffer 对象，这个函数的一个参数采用 **DSCBUFFERDESC** 类型的结构来说明 buffer 的一些特性，这个结构的最后一个成员变量是一个 **WAVEFORMATEX** 结构，这个结构一定要初始化成泥需要的 wav 格式。

说明一下，如果你的应用程序一边播放的同时进行录制，如果你录制的 buffer 格式和你的主缓冲 buffer 不一样，那么你创建录制 buffer 对象就会失败，原因在于，有些声卡只支持一种时钟，不能同时支持录音和播放同时以两种不同的格式进行。

下面的函数，演示了如何创建一个录音的 buffer 对象，这个 buffer 对象能够处理 1 秒的数据，注意，这里传递的录音设备对象参数一定要通过 **DirectSoundCaptureCreate8** 来创建，而不是早期的 **DirectSoundCaptureCreate** 接口，否则，buffer 对象不支持 **IDirectSoundCaptureBuffer8** 接口，

```
HRESULT CreateCaptureBuffer(LPDIRECTSOUNDCAPTURE8 pDSC,
                            LPDIRECTSOUNDCAPTUREBUFFER8* ppDSCB8)
{
   HRESULT hr;
   DSCBUFFERDESC                dscbd;
   LPDIRECTSOUNDCAPTUREBUFFER   pDSCB;
   WAVEFORMATEX                 wfx =
      {WAVE_FORMAT_PCM, 2, 44100, 176400, 4, 16, 0};
      // wFormatTag, nChannels, nSamplesPerSec, mAvgBytesPerSec,
      // nBlockAlign, wBitsPerSample, cbSize

   if ((NULL == pDSC) || (NULL == ppDSCB8)) return E_INVALIDARG;
   dscbd.dwSize = sizeof(DSCBUFFERDESC);
   dscbd.dwFlags = 0;
   dscbd.dwBufferBytes = wfx.nAvgBytesPerSec;
   dscbd.dwReserved = 0;
   dscbd.lpwfxFormat = &wfx;
   dscbd.dwFXCount = 0;
   dscbd.lpDSCFXDesc = NULL;

   if (SUCCEEDED(hr = pDSC->CreateCaptureBuffer(&dscbd, &pDSCB, NULL)))
   {
      hr = pDSCB->QueryInterface(IID_IDirectSoundCaptureBuffer8, (LPVOID*)ppDSCB8);
      pDSCB->Release();
   }
   return hr;
}
```

5 通过录音 buffer 对象获取信息

你可以通过 **IDirectSoundCaptureBuffer8::GetCaps** 方法来获取录音 buffer 的大小，但一定要记得初始化 **DSCBCAPS** 结构类型参数的 dwSize 成员变量。

为了获取 buffer 中数据的格式，你可以通过 **IDirectSoundCaptureBuffer8::GetFormat**.方法来获取 buffer 中的数据格式，这个函数通过 **WAVEFORMATEX** 结构返回音频数据的信息，如果我们想知道一个录音 buffer 目前的状态如何，可以通过

IDirectSoundCaptureBuffer8::GetStatus 来获取，这个函数通过一个 DWORD 类型的参数来表示该 buffer 是否正在录音，
IDirectSoundCaptureBuffer8::GetCurrentPosition 方法可以获取 buffer 中 read 指针和录制指针的偏差。Read 指针指向填充到该 buffer 中的数据的最末端，capture 指针则指向复制到硬件的数据的末端，你 read 指针指向的前段数据都是安全数据，你都可以安全的复制。

6 录音 buffer 对象通知机制

为了安全的定期的从录音 buffer 中 copy 数据，你的应用程序就要知道，什么时候 read 指针指向了特定的位置，一个方法是通过 **IDirectSoundCaptureBuffer8::GetCurrentPosition**. 方法来获取 read 指针的位置，另外一个更有效的方法采用通知机制，通过 **IDirectSoundNotify8::SetNotificationPositions** 方法，你可以设置任何一个小于 buffer 的位置来触发一个事件，切记，当 buffer 正在 running 的时候，不要设置。

如何来设置一个触发事件呢，首先要得到 **IDirectSoundNotify8** 接口指针，你可以通过 buffer 对象的 QuerInterface 来获取这个指针接口，对于你指定的任何一个 position，你都要通过 CreateEvent 方法，创建一个 win32 内核对象，然后将内核对象的句柄赋给 **DSBPOSITIONNOTIFY** 结构的 hEventNotify 成员，通过该结构的 dwOffset 来设置需要通知的位置在 buffer 中的偏移量。

最后将这个结构或者结构数组，传递给 SetNotificationPositions 函数，下面的例子设置了三个通知，当 position 达到 buffer 的一半时会触发一个通知，当达到 buffer 的末端时，触发第二个通知，当录音停止时触发第三个通知消息。

```
HRESULT SetCaptureNotifications(LPDIRECTSOUNDCAPTUREBUFFER8 pDSCB)
{
  #define cEvents   3

  LPDIRECTSOUNDNOTIFY8 pDSNotify;
  WAVEFORMATEX              wfx;
  HANDLE       rghEvent[cEvents] = {0};
  DSBPOSITIONNOTIFY   rgdsbpn[cEvents];
  HRESULT      hr;

  if (NULL == pDSCB) return E_INVALIDARG;
  if          (FAILED(hr          =          pDSCB->QueryInterface(IID_IDirectSoundNotify,
(LPVOID*)&pDSNotify)))
  {
    return hr;
  }
  if (FAILED(hr = pDSCB->GetFormat(&wfx, sizeof(WAVEFORMATEX), NULL)))
  {
    return hr;
  }

  // Create events.
  for (int i = 0; i < cEvents; ++i)
  {
    rghEvent[i] = CreateEvent(NULL, TRUE, FALSE, NULL);
```

```
        if (NULL == rghEvent[i])
        {
            hr = GetLastError();
            return hr;
        }
    }

    // Describe notifications.

    rgdsbpn[0].dwOffset = (wfx.nAvgBytesPerSec/2) -1;
    rgdsbpn[0].hEventNotify = rghEvent[0];

    rgdsbpn[1].dwOffset = wfx.nAvgBytesPerSec - 1;
    rgdsbpn[1].hEventNotify = rghEvent[1];

    rgdsbpn[2].dwOffset = DSBPN_OFFSETSTOP;
    rgdsbpn[2].hEventNotify = rghEvent[2];

    // Create notifications.

    hr = pDSNotify->SetNotificationPositions(cEvents, rgdsbpn);
    pDSNotify->Release();
    return hr;
}
```

7 录音的步骤

录音主要包括下面几个步骤

1 调用 **IDirectSoundCaptureBuffer8::Start** 使 buffer 对象开始工作，通过你要给这个函数 dwFlags 传递一个 DSCBSTART_LOOPING 参数，注意，buffer 就会不停工作，而不是当 buffer 填充满了就停止工作，当缓冲区满了后，就会从头重新填充

2 等待你期望的事件通知，当 buffer 被填充到某个你期望的位置时，会触发通知。

3 当你收到通知的时，你就要调用 **IDirectSoundCaptureBuffer8::Lock** 来锁住 bufer 的一部份，切记，不要将 capture 指针指向的内存锁住，你可以调用 **IDirectSoundCaptureBuffer8::GetCurrentPosition** 方法来获取 read 指针的位置。在传递给 Lock 函数的参数中，你一定要指定内存的大小和偏移量，这个函数会返回你锁住的内存的起始地址，以及 block 的大小，

4 从你锁住的内存中复制 data，

5 复制完成后要记得 **IDirectSoundCaptureBuffer8::Unlock** 方法来解锁内存

6 在你停止录音之前，你可以反复的重复 2~5 步骤，如果你想停止录音了，你可以调用 **IDirectSoundCaptureBuffer8::Stop** 方法。

9 将录音写入 wav 文件

WAV 文件是采用 RIFF 格式的文件，在文件中包含一系列的 chunks，来描述头信息和数据信息，win32API 提供一套 mmio 系列函数用来操作 RIFF 格式的文件，但是 Directsound 并没有提供读写 wav 格式文件的函数，但是，Directsound 里封装了一个 CWaveFile 类用来操

作 wav 文件，可以通过 open 来写入文件的头信息，write 来写入文件的数据，close 函数写入文件的长度，关闭文件。
下面是代码，如何创建一个 wav 格式的文件

```
CWaveFile      g_pWaveFile;
WAVEFORMATEX    wfxInput;

ZeroMemory( &wfxInput, sizeof(wfxInput));
wfxInput.wFormatTag = WAVE_FORMAT_PCM;
wfxInput.nSamplesPerSec = 22050
wfxInput.wBitsPerSample =    8;
wfxInput.nChannels = 1;
wfxInput.nBlockAlign =
      wfxInput.nChannels * (wfxInput.wBitsPerSample / 8);
wfxInput.nAvgBytesPerSec =
      wfxInput.nBlockAlign * wfxInput.nSamplesPerSec;

g_pWaveFile = new CWaveFile;
if (FAILED(g_pWaveFile->Open("mywave.wav", &wfxInput,
      WAVEFILE_WRITE)))
{
   g_pWaveFile->Close();
}
```

下面的代码就演示了如何从 buffer 对象中将数据写入文件中

```
HRESULT RecordCapturedData()
{
   HRESULT hr;
   VOID* pbCaptureData    = NULL;
   DWORD dwCaptureLength;
   VOID* pbCaptureData2 = NULL;
   DWORD dwCaptureLength2;
   VOID* pbPlayData      = NULL;
   UINT    dwDataWrote;
   DWORD dwReadPos;
   LONG lLockSize;

   if (NULL == g_pDSBCapture)
         return S_FALSE;
   if (NULL == g_pWaveFile)
         return S_FALSE;

   if (FAILED (hr = g_pDSBCapture->GetCurrentPosition(
      NULL, &dwReadPos)))
         return hr;
```

```
    // Lock everything between the private cursor
    // and the read cursor, allowing for wraparound.

    lLockSize = dwReadPos - g_dwNextCaptureOffset;
    if( lLockSize < 0 ) lLockSize += g_dwCaptureBufferSize;

    if( lLockSize == 0 ) return S_FALSE;

    if (FAILED(hr = g_pDSBCapture->Lock(
                g_dwNextCaptureOffset, lLockSize,
                &pbCaptureData, &dwCaptureLength,
                &pbCaptureData2, &dwCaptureLength2, 0L)))
        return hr;

    // Write the data. This is done in two steps
    // to account for wraparound.

    if (FAILED( hr = g_pWaveFile->Write( dwCaptureLength,
            (BYTE*)pbCaptureData, &dwDataWrote)))
        return hr;

    if (pbCaptureData2 != NULL)
    {
        if (FAILED(hr = g_pWaveFile->Write(
                dwCaptureLength2, (BYTE*)pbCaptureData2,
                &dwDataWrote)))
            return hr;
    }

    // Unlock the capture buffer.

   g_pDSBCapture->Unlock( pbCaptureData, dwCaptureLength,
        pbCaptureData2, dwCaptureLength2    );

    // Move the capture offset forward.

    g_dwNextCaptureOffset += dwCaptureLength;
    g_dwNextCaptureOffset %= g_dwCaptureBufferSize;
    g_dwNextCaptureOffset += dwCaptureLength2;
    g_dwNextCaptureOffset %= g_dwCaptureBufferSize;

    return S_OK;
}
```

四、DirectSound 开发高级技巧

4.1Dsound 驱动模型（DirectSound Driver Models）

在 VXD 驱动模型下，所有的 DirectSound 的混音工作都是由 Dsound.vxd 来完成的，一个虚拟的设备驱动程序。Dsound.vxd 也提供操作声卡从 Cpu 接收数据的缓冲区的方法，这其实和 DirectSound 的主缓冲区是类似的。DirectSound 应用程序可以给主缓冲区设置特定的属性，例如，采样频率，或者采样精度，也就改变了硬件设备。

在 WDM 驱动模式下，DirectSound 并不直接操作硬件，当然，操作硬件加速缓冲区除外。相应的，DirectSound 将数据送往内核混音器，内核混音器的工作就是将多个格式的音频流调整为一个统一的格式，将它们进行混音，然后将结果送到硬件上进行播放。其实，它的工作和 Dsound.vxd 的工作类似。一个重要的区别在于 Dsound.vxd 仅仅对 DirectSound 的内存缓冲区进行混音，内核混音器会对所有的 windows 音频数据进行混音，包括那些使用 waveOut win32 函数输出数据的应用，DirectSound 和 Wave 格式的音频输出设备不能同时打开在 WDM 驱动模式下是不成立的。

最重要的是内核混音器和音频硬件的关系，内核混音器就是一个系统中的软件，可以用来指定硬件的 DMA 缓冲区。它根据它要进行混音的音频数据格式，来选择硬件的格式，它会将它混音的音频数据以一种高质量的形式进行输出，或者根据硬件的情况选择近似的音频质量。

这里有一个很重要的暗示，就是 DirectSound 不能设置硬件的 DMA 缓冲区格式。对于你的应用程序而言，硬件格式其实就是根据你实际播放的音频的格式来定的。如果你 play 的是 44kHZ，内核混音器就会将所有的数据都混音成 44kHZ，同时也保证硬件以 44Khz 进行输出。

作为应用程序的开发者，你没法来选择系统驱动模式，因为驱动模式的选择有声卡类型，windows 版本，以及安装的驱动程序来控制。由于这个原因，你在测试你的程序时要注意所有的情况，DirectSound 或许用的是 Dsound.vxd 或者用的是内核混音器，你要确保你的应用程序对两种方式都支持。

4.2 设置硬件的扩展属性（System Property Sets）

4.3Property Sets for DirectSound Buffers

DirectSound 设备的属性可以通过下面的类对象来获取，该类的 ID 为 CLSID_DirectSoundPrivate (11AB3EC0-25EC-11d1-A4D8-00C04FC28ACA). 该类支持 **IKsPropertySet** 接口，CLSID 和属性的定义在 Dsconf.h 文件中可以找到。

1 要想创建该类的对象，首先要通过 LoadLibrary 加载 Dsound.dll
2 调用 GetProcAddress 函数来获取 DllGetClassObject 函数接口。
3 通过 DllGetClassObject 函数来获取 IClassFactory 接口，CLSID_DirectSoudnPrivate 类厂对象接口。
4 调用 IClassFactory::CreateInstance 来创建 CLSID_DirectSoundPrivate 对象，并获取 IKsPropertySet 接口（IID_IKsPropertySet）。

CLSID_DirectSoundPrivate 对象只支持一个属性设置接口 DSPROPSETID_DirectSoundDevice(84624f82-25ec-11d1-a4d8-00c04fc28aca)这个属性设置接口暴露了下面三个只读的属性

DSPROPERTY_DIRECTSOUNDDEVICE_WAVEDEVICEMAPPING
DSPROPERTY_DIRECTSOUNDDEVICE_DESCRIPTION
DSPROPERTY_DIRECTSOUNDDEVICE_ENUMERATE

DSPROPERTY_DIRECTSOUNDDEVICE_WAVEDEVICEMAPPING 这个属性根据指定的设备的名称来获取该设备的 GUID。
获取的属性数据包含在
DSPROPERTY_DIRECTSOUNDDEVICE_WAVEDEVICEMAPPING_DATA 结构里，这个结构包含下面的成员

Member	Type	Description
DeviceName	String	[in] Name of device
DataFlow	DIRECTSOUNDDEVICE_DATAFLOW	[in] Direction of data flow, either DIRECTSOUNDDEVICE_DATAFLOW_RENDER or DIRECTSOUNDDEVICE_DATAFLOW_CAPTURE
DeviceID	GUID	[out] DirectSound device ID

DSPROPERTY_DIRECTSOUNDDEVICE_DESCRIPTION 属性
该属性可以返回指定 GUID 设备的全部的属性描述，可以通过下面的结构返回数据
DSPROPERTY_DIRECTSOUNDDEVICE_DESCRIPTION_DATA 这个结构的成员如下

Member	Type	Description
Type	DIRECTSOUNDDEVICE_TYPE	[out] Device type: DIRECTSOUNDDEVICE_TYPE_EMULATED, DIRECTSOUNDDEVICE_TYPE_VXD or DIRECTSOUNDDEVICE_TYPE_WDM
DataFlow	DIRECTSOUNDDEVICE_DATAFLOW	[in, out] Direction of data flow, either DIRECTSOUNDDEVICE_DATAFLOW_RENDER or DIRECTSOUNDDEVICE_DATAFLOW_CAPTURE

DeviceID	GUID	[in] DirectSound device GUID, or NULL for the default device of the type specified by DataFlow
Description	String	[out] Description of DirectSound device
Module	String	[out] Module name of the DirectSound driver
Interface	String	[out] PnP device interface name
WaveDeviceID	ULONG	[out] Identifier of the corresponding Windows Multimedia device

DSPROPERTY_DIRECTSOUNDDEVICE_ENUMERATE 属性
这个属性用来枚举所有的 DirectSound 的播放或者录音设备。可以通过 **DSPROPERTY_DIRECTSOUNDDEVICE_ENUMERATE_DATA** 结构将属性值返回，这个结构的成员如下

Member	Type	Description
Callback	LPFNDIRECTSOUNDDEVICEENUMERATECALLBACK	[in] Application-defined callback function. When IKsPropertySet::Get is called, this function is called once for each device enumerated. It takes as parameters a DSPROPERTY_DIRECTSOUNDDEVICE_DESCRIPTION_DATA structure describing the enumerated device, and the value in Context.
Context	LPVOID	[in] User-defined value to be passed to the callback function for each device enumerated.

4.4 如何优化 Directsound（Optimizing DirectSound Performance）

主要讲三个方面的内容，如何用硬件进行混音，动态声音管理，有效地使用缓冲区

许多声卡都拥有自己的辅助缓冲区，可以处理 3-D 特技和混音特技。这些缓冲区，就像它们的名字一样，其实是存在于系统的内存中，而不是在声卡上，但是这些缓冲区是由声卡

来直接操作的，所以比需要处理器来控制软件缓冲区，速度要快许多。所以，在硬件条件允许的条件下要多申请硬件的缓冲区，特别是 3-D 缓冲区。

缺省的情况下，DirectSound 会优先申请硬件缓冲区，但是能够申请多少硬件的缓冲区是由硬件设备的性能决定的。硬件在同一时刻能播放的声音数量越多，可申请的硬件缓冲区的数量就越多，当创建一个缓冲区时，DirectSound 就分配一个 hardware voice，缓冲区销毁时就释放 hardware voice。如果一个应用程序创建了很多的缓冲区，但是，很多缓冲区是由软件来销毁的，也就是说，这些缓冲区是由 CPU 而不是声卡来控制和混音的。

注意：DirectSound 声音管理器分配的是硬件混音资源，并不是缓冲区，在一个 PCI 板卡上，缓冲区在分配前后都占用相同大小的内存，不管是将该缓冲区分配给硬件还是软件混音器。

动态的声音管理，通过在缓冲区播放才进行 voice allocation，获取提前结束那么权限比较低的音频数据播放，释放他们的资源，从而可以减轻硬件设备的压力。

当缓冲区正在 play 的时候，为了延迟硬件资源用来分配给混音，3-D 特技，可以在创建缓冲区对象时，将 **DSBUFFERDESC** 结构的 dwFlages 设置为 DSBCAPS_LOCDEFER 标志，传递给 **IDirectSound8::CreateSoundBuffer**. 函数，当你对这样的缓冲区进行 **IDirectSoundBuffer8::Play** or **IDirectSoundBuffer8::AcquireResources** 操作时，DrectSound 会尽可能的在硬件上 play。

当调用 Play 的时候，你可以试图通过传递下面表格中的参数将其他正在使用的硬件 voice 资源释放掉。从而供你的 buffer 使用

DSBPLAY_TERMINATEBY_TIME	Select the buffer that has been playing longer than any other candidate buffers.
DSBPLAY_TERMINATEBY_DISTANCE	Select the 3-D candidate buffer farthest from the listener.
DSBPLAY_TERMINATEBY_PRIORITY	Select the buffer that has the lowest priority of candidate buffers, as set in the call to **Play**. If this is combined with one of the other two flags, the other flag is used only to resolve ties.

DirectSound 会根据你设置的标志，对所有正在 play 的 buffer 进行搜索，如果找到符合条件的缓冲区，DirectSound 就会停掉该资源，然后将该资源分配给你的心的缓冲区使用。如果没有找到符合条件的缓冲区，那么你的新创建的缓冲区只好在软件缓冲区播放了，当然，如果你设置了 DSBPLAY_LOCHARDWARE 标志，此时，play 调用就会失败。

下面我们看看如何更有效地使用缓冲区。

当使用流缓冲区的时候，要限制通知的次数和数据读写的次数。不要创建有很多通知 positions 的缓冲区，或者太小的缓冲区。流缓冲区在小于 3 个通知位置时工作的效率最高。当你改变一个辅助缓冲区的控制项时，此时效率就会受影响，所以，要尽可能少的调用 **IDirectSoundBuffer8::SetVolume**, **IDirectSoundBuffer8::SetFrequency**.

IDirectSoundBuffer8::SetPan，函数，例如，你有个习惯总是喜欢在控制面板上来回的拖动左右声道的位置。

一定要记住，3D 缓冲区是需要占用的更多的 CPU 的。所以，要注意下面的事情

1 将经常播放的 sounds 放到硬件缓冲区中，

2 Don't create 3-D buffers for sounds that won't benefit from the effect.

3 通过 **IDirectSound3DBuffer8::SetMode** 函数，设置 DS3DMODE_DISABLE 标志来停止对 3d 缓冲区的 3d 处理，

4 最好批量的参数进行调整。

4.5 向主缓冲区写数据（Writing to the Primary Buffer）

当应用程序需要一些特殊的混音或者特技，而辅助缓冲区不支持这些功能，那么 DirectSOund 允许直接曹操主缓冲区，

当你获得操作主缓冲区的权限时，其他的 DirectSound 特性就变得不可用了，辅助缓冲区没法混音，硬件加速混音器也无法工作。

大多数的应用程序应该使用辅助缓冲区避免直接操作主缓冲区，因为可以申请大块的辅助缓冲区，可以提高足够长的写入时间，从而避免了音频数据产生缝隙的危险。

只有当主缓冲区硬件时你才能操作它，所以你可以通过 **IDirectSoundBuffer8::GetCaps** 函数来查询，该函数的参数结构 dwFlages 成员设置为 DSBCAPS_LOCHARDWARE，如果你想锁定一个正在被软件枚举的主缓冲区，会失败的。

你通过 **IDirectSound8::CreateSoundBuffer** 函数来创建主缓冲区，只要设置 DSBCAPS_PRIMARYBUFFER 标志即可。同时要保证你的协作度为 DSSCL_WRITEPRIMARY。

下面的代码演示了如何获取向主缓冲区写数据的权限

```
BOOL AppCreateWritePrimaryBuffer(
    LPDIRECTSOUND8 lpDirectSound,
    LPDIRECTSOUNDBUFFER *lplpDsb,
    LPDWORD lpdwBufferSize,
    HWND hwnd)
{
    DSBUFFERDESC dsbdesc;
    DSBCAPS dsbcaps;
    HRESULT hr;
    WAVEFORMATEX wf;

    // Set up wave format structure.
    memset(&wf, 0, sizeof(WAVEFORMATEX));
    wf.wFormatTag = WAVE_FORMAT_PCM;
    wf.nChannels = 2;
    wf.nSamplesPerSec = 22050;
    wf.nBlockAlign = 4;
    wf.nAvgBytesPerSec =
        wf.nSamplesPerSec * wf.nBlockAlign;
```

```
    wf.wBitsPerSample = 16;

    // Set up DSBUFFERDESC structure.
    memset(&dsbdesc, 0, sizeof(DSBUFFERDESC));
    dsbdesc.dwSize = sizeof(DSBUFFERDESC);
    dsbdesc.dwFlags = DSBCAPS_PRIMARYBUFFER;
    // Buffer size is determined by sound hardware.
    dsbdesc.dwBufferBytes = 0;
    dsbdesc.lpwfxFormat = NULL; // Must be NULL for primary buffers.

    // Obtain write-primary cooperative level.
    hr = lpDirectSound->SetCooperativeLevel(hwnd, DSSCL_WRITEPRIMARY);
    if SUCCEEDED(hr)
    {
        // Try to create buffer.
        hr = lpDirectSound->CreateSoundBuffer(&dsbdesc,
            lplpDsb, NULL);
        if SUCCEEDED(hr)
        {
            // Set primary buffer to desired format.
            hr = (*lplpDsb)->SetFormat(&wf);
            if SUCCEEDED(hr)
            {
                // If you want to know the buffer size, call GetCaps.
                dsbcaps.dwSize = sizeof(DSBCAPS);
                (*lplpDsb)->GetCaps(&dsbcaps);
                *lpdwBufferSize = dsbcaps.dwBufferBytes;
                return TRUE;
            }
        }
    }
    // Failure.
    *lplpDsb = NULL;
    *lpdwBufferSize = 0;
    return FALSE;
}
```

下面的代码演示了应用程序如何混音的，其中 CustomMixer 函数是用来将几个音频流混音的函数 。AppMixIntoPrimaryBuffer 应该定时的被调用。

```
BOOL AppMixIntoPrimaryBuffer(
        APPSTREAMINFO* lpAppStreamInfo,
        LPDIRECTSOUNDBUFFER lpDsbPrimary,
        DWORD dwDataBytes,
        DWORD dwOldPos,
```

```
    LPDWORD lpdwNewPos)
{
  LPVOID lpvPtr1;
  DWORD dwBytes1;
  LPVOID lpvPtr2;
  DWORD dwBytes2;
  HRESULT hr;
  // Obtain write pointer.
  hr = lpDsbPrimary->Lock(dwOldPos, dwDataBytes,
            &lpvPtr1, &dwBytes1,
            &lpvPtr2, &dwBytes2, 0);

  // If DSERR_BUFFERLOST is returned, restore and retry lock.

  if (DSERR_BUFFERLOST == hr)
  {
    lpDsbPrimary->Restore();
    hr = lpDsbPrimary->Lock(dwOldPos, dwDataBytes,
              &lpvPtr1, &dwBytes1,
              &lpvPtr2, &dwBytes2, 0);
  }
  if SUCCEEDED(hr)
  {
    // Mix data into the returned pointers.
    CustomMixer(lpAppStreamInfo, lpvPtr1, dwBytes1);
    *lpdwNewPos = dwOldPos + dwBytes1;
    if (NULL != lpvPtr2)
    {
      CustomMixer(lpAppStreamInfo, lpvPtr2, dwBytes2);
      *lpdwNewPos = dwBytes2; // Because it wrapped around.
    }
    // Release the data back to DirectSound.
    hr = lpDsbPrimary->Unlock(lpvPtr1, dwBytes1,
              lpvPtr2, dwBytes2);
    if SUCCEEDED(hr)
    {
      return TRUE;
    }
  }
  // Lock or Unlock failed.
  return FALSE;
}
```

五、DirectSound 接口函数和指针简介

5.1DSound 常用的接口指针

其实 DirectSound 的接口很少，很简单，真正有用的有三个，设备对象，缓冲区对象，通知对象接口，然后就是一写特技接口。

1 IDirectSound8

2 IDirectSoundBuffer8

3 IDirectSoundCapture8

4 IDirectSoundCaptureBuffer8

5 IDirectSoundNotify8

6 IKsPropertySet

5.2Dsound 函数

函数更少了，不足 10 个

1 DirectSoundCreate8

```
HRESULT WINAPI DirectSoundCreate8(
  LPCGUID  lpcGuidDevice,
  LPDIRECTSOUND8 * ppDS8,
  LPUNKNOWN  pUnkOuter
);
```

用来创建设备对象。

2 DirectSoundEnumerate 用来枚举播放设备的函数

```
HRESULT WINAPI DirectSoundEnumerate(
  LPDSENUMCALLBACK  lpDSEnumCallback,
  LPVOID  lpContext
);
```

参数为一个回调函数，一个指针

3 DirectSoundCaptureCreate8 创建录音设备对象

```
HRESULT WINAPI DirectSoundCaptureCreate8(
  LPCGUID lpcGUID,
  LPDIRECTSOUNDCAPTURE8 *lplpDSC,
  LPUNKNOWN pUnkOuter
);
```

4 DirectSoundCaptureEnumerate 枚举录音设备对象

```
HRESULT WINAPI DirectSoundCaptureEnumerate(
  LPDSENUMCALLBACK  lpDSEnumCallback,
  LPVOID  lpContext
);
```

5 GetDeviceID

```
HRESULT GetDeviceID(
```

```
  LPCGUID  pGuidSrc,
  LPGUID  pGuidDest
);
```

6 DirectSoundFullDuplexCreate8

```
HRESULT WINAPI DirectSoundFullDuplexCreate8(
  LPCGUID  pcGuidCaptureDevice,
  LPCGUID  pcGuidRenderDevice,
  LPCDSCBUFFERDESC  pcDSCBufferDesc,
  LPCDSBUFFERDESC  pcDSBufferDesc,
  HWND  hWnd,
  DWORD  dwLevel,
  LPDIRECTSOUNDFULLDUPLEX*  ppDSFD,
  LPDIRECTSOUNDCAPTUREBUFFER8  *ppDSCBuffer8,
  LPDIRECTSOUNDBUFFER8  *ppDSBuffer8,
  LPUNKNOWN  pUnkOuter
);
```

5.3Dsound 常用的结构

六、Wave 文件格式以及底层操作函数 API 使用技巧

6.1RIFF 文件结构

RIFF（Resource Interchange File Format，资源互换文件格式）是微软公司定义的一种用于管理 windows 环境中多媒体数据的文件格式，波形音频 wave，MIDI 和数字视频 AVI 都采用这种格式存储。

构造 RIFF 文件的基本单元叫做数据块（Chunk），每个数据块包含 3 个部分，

1 4 字节的数据块标记（或者叫做数据块的 ID）

2 数据块的大小

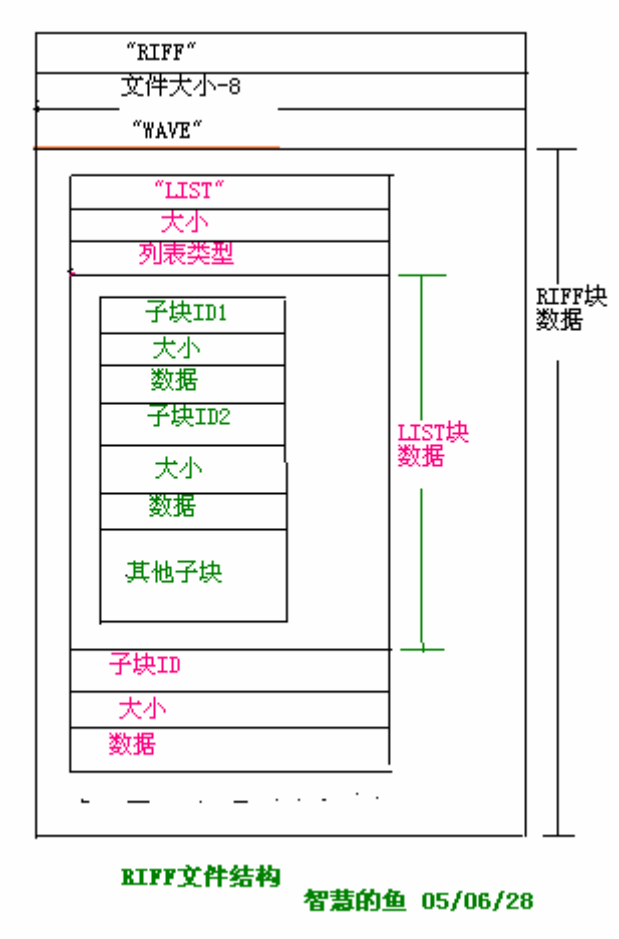
3 数据

整个 RIFF 文件可以看成一个数据块，其数据块 ID 为 RIFF，称为 RIFF 块。一个 RIFF 文件中只允许存在一个 RIFF 块。RIFF 块中包含一系列的子块，其中有一种字块的 ID 为“LIST”，称为 LIST，LIST 块中可以再包含一系列的子块，但除了 LIST 块外的其他所有的子块都不能再包含子块。

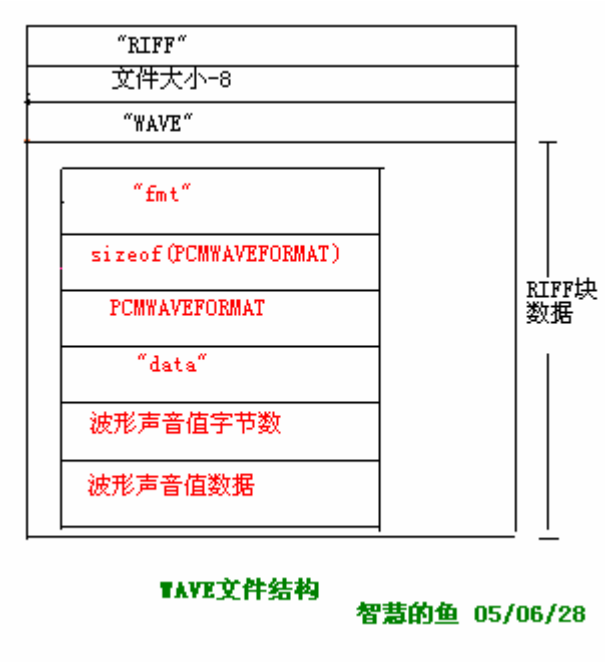
RIFF 和 LIST 块分别比普通的数据块多一个被称为形式类型（Form Type）和列表类型（List Type）的数据域，其组成如下：

1 4 字节的数据块标记（Chunk ID）

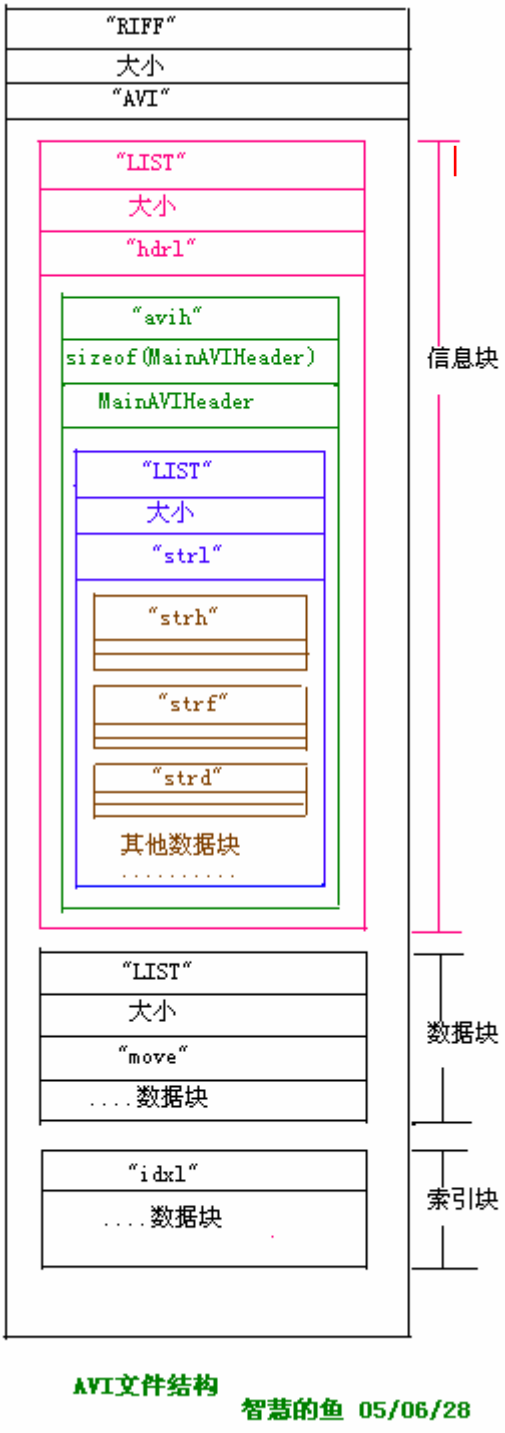
2 数据块的大小
3 4 字节的形式类型或者列表类型
4 数据
下面的结构显示了一个 RIFF 文件的结构



6.2WAVE 文件结构


"RIFF"
文件大小-8
"WAVE"
"fmt"
sizeof(PCMWAVEFORMAT)
PCMWAVEFORMAT
"data"
波形声音值字节数
波形声音值数据
RIFF块
数据
WAVE文件结构
智慧的鱼 05/06/28

6.3avi 文件结构



6.4 多媒体文件输入输出

6.5 波形音频的编程（wave 系列函数）

6.6AVI 编程