###### 背景介绍

WINDOWS下音频的采集，播放有三种模式：

1）通过高级音频函数、媒体控制接口MCI[1、2]设备驱动程序；

2）低级音频函数MIDI Mapper、低级音频设备驱动（WaveX API）；

3）利用DirectX中的DirectSound；

使用MCI的方法极其简便，灵活性较差；使用低级音频函数的方法相对来说难一点，但是能够对音频数据进行灵活的操控；而采用DirectSound的方法，控制声音数据灵活，效果比前二者都好，但实现起来是三者中最难的。

程序设计采用方法2 ，使用低级音频函数和相对应的音频设备驱动。

###### 相关介绍

1. 基本过程

低层音频服务及重要的数据结构低级音频服务控制着不同的音频设备，这些设备包括 WAVE,MIDI和辅助音频设备。低级音频服务包括如下内容：

　　(1)查询音频设备；

　　(2)打开和关闭设备驱动程序；

　　(3)分配和准备音频数据块；

　　(4)管理音频数据块；

　　(5)应用MMTIME结构；

　　(6)处理错误。

1. 相关库

WaveX低级音频函数的相关声明和定义在mmsystem.h头文件和Winmm.lib库中。所以如果程序中用到这些函数，必须包含mmsystem.h这个头文件，同时导进Winmm.lib库。如下：

1. #include "mmsystem.h"
2. #pragma comment(lib,"Winmm.lib")

双/多缓冲技术可以很好的实现声音的快速连续采集和实时顺畅播放。当程序在响应某块缓冲数据已满或播放完毕消息时，声卡可以继续往下一块缓冲添加数据或播放下一块缓冲的数据，如此循环保障声音的连续性。

1. 数据结构

声音在采集（录音）和播放的时需要有一些统一的格式，包括音频格式类型，声道，采样率等信息。下面的数据结构具体描述了该格式：

typedef struct tWAVEFORMATEX

{

    WORD        wFormatTag;

    WORD        nChannels;

    DWORD       nSamplesPerSec;

    DWORD       nAvgBytesPerSec;

    WORD        nBlockAlign;

    WORD        wBitsPerSample;

    WORD        cbSize;

} WAVEFORMATEX, \*PWAVEFORMATEX,

其中，wFormatTag是音频格式类型，nChannels是声道数，nSamplesPerSec是采样频率，nAvgBytesPerSec是每秒钟的字节数，nBlockAlign是每个样本的字节数，wBitsPerSample是每个样本的量化位数，cbSize是附加信息的字节大小。

1. 参数设置

录音使用window函数，采用的是PCM方式。

声卡输入和输出的音频属性可定义如下：（考虑音质和存MP3两点）

m\_waveformt.wFormatTag        =  WAVE\_FORMAT\_PCM;

m\_waveformt.nChannels         =  2;双声道

m\_waveformt.nSamplesPerSec    =  44100;

m\_waveformt.wBitsPerSample    =  16;

m\_waveformt.cbSize            =  0;

m\_waveformt.nBlockAlign       =  2;

m\_waveformt.nAvgBytesPerSec   = 16000;

　　其中，存在着下面的关系：

nBlockAlign       =  nChannels \* wBitsPerSample / 8 ;

nAvgBytesPerSec   =  nSamplesPerSec \* nBlockAlign ;

1. 详细流程（输入和输出）

* 调用WaveX 低级音频函数API启动声卡录音的基本操作步骤如下图所示：

               打开录音设备： waveInOpen

                           ↓

        为录音设备准备缓存： waveInPrepareHeader

                           ↓

          为输入设备增加缓存： waveInAddBuffer

                           ↓

                 启动录音： waveInStart

                           ↓

            清除缓存： waveInUnprepareHeader

                           ↓

                 停止录音： waveInReset

                           ↓

               关闭录音设备： waveInClose

在这个过程中，会产生很多WM\_WIM\_\*\*\*格式的WINDOWS消息。程序通过捕获这些消息对缓存，数据和设备进行处理，具体可见后面章节。录音设备打开时，可以指定消息的响应方式：回掉函数，线程ID，WINDOWS窗口句柄或事件句柄等。

WM\_WIM\_OPEN        音频输入 设备打开消息

            ↓

         WM\_WIM\_DATA缓冲录满或停止录音消息

            ↓

        WM\_WIM\_CLOSE  音频输入设备关闭消息

MMRESULT waveInOpen(

    LPHWAVEIN phwi,         //输入设备句柄

    UINT uDeviceID,         //输入设备ID

    LPWAVEFORMATEX pwfx,    //录音格式指针

    DWORD dwCallback,       //处理消息的回调函数或窗口句柄，线程ID等

    DWORD dwCallbackInstance,   //通常为0

    DWORD fdwOpen           //处理消息方式的符号位

);

* 调用WaveX 低级音频函数API启动声卡输出的基本操作步骤如下图所示：

                 打开输出设备： waveOutOpen

                              ↓

          为输出设备准备缓存： waveOutPrepareHeader

                              ↓

             写数据导输出设备缓存： waveOutWrite

                              ↓

            清除输出缓存： waveOutUnprepareHeader

                              ↓

                   停止输出： waveOutReset

                              ↓

                 关闭输出设备： waveOutClose

　　同录音一样，在音频输出过程中也有一系列的消息，程序通过捕获这些消息对缓存，数据和设备进行处理。

WM\_WOM\_OPEN    音频输出 设备打开消息

            ↓

        WM\_WOM\_DONE缓冲播放完或停止输出消息

            ↓

        WM\_WOM\_CLOSE音频输出设备关闭消息

1. 程序结果



通过按键实现录音和播放功能，使用暂停（继续）和停止进行相应的控制。

文件保存，在程序当前目录下存有source.wav文件。

设定16位、44K的采样，经测试，一分钟录音生成的文件大小大概在10M左右。考虑到数据量过大会导致相应的存储问题，在此引入MP3机制。通过对wav数据流进行相应的压缩，转化为MP3文件（1分钟文件大小0.9M左右），压缩了10倍多。因此可以节约可观的数据存储空间。

形成的MP3文件存储在D:\Record\目录下，并根据当天日期建立文件夹。MP3文件通过录音时刻（时分）来命名，实现文件自动存储。

 