オブジェクト指向と ゲームプログラミング

▶ DirectX Audio編 ― 第７回　Waveファイルの読み込み ◀

Waveファイルの構成

DirectSoundBufferオブジェクトが保持するサウンドイメージの形式は、通常のWaveファイルのイメージ(PCM形式)とまったく同じです。フォーマットが同じなので、Waveファイルのイメージをそのままサウンドバッファに転送するだけで再生することが可能です。しかし、DirectSoundには、Waveファイルを読み込むメソッドを提供していないので、Waveファイルを解析しながら読み込まなければなりません。そのためには、Waveファイルの構造を正しく理解しておく必要があります。

Waveファイルは、AVIファイルなどでも使われているRIFF(Resource Interchange File Format)形式になっています。一般的なWaveファイルは、以下のような構成になっています。

RIFFの４文字('RIFF'ファイルのヘッダ)
この部分以降のデータサイズ ※１
WAVEの４文字('WAVE'チャンクのヘッダ)
fmt□の４文字('fmt 'チャンクのヘッダ) DWORD　fmt チャンクのサイズ WORD　フォーマットID WORD　チャネル数 DWORD　サンプリング周波数 DWORD　平均データ速度 WORD　ブロックサイズ WORD　１サンプル当たりのビット数 WORD　ヘッダ拡張部サイズ(byte) ※２ ～　ヘッダ拡張部 ※２ ※３
factの４文字('fact'チャンクのヘッダ) DWORD　factチャンクのサイズ DWORD　全サンプル数
dataの４文字('data'チャンクのヘッダ) DWORD　dataチャンクのサイズ ～　WAVEファイルのイメージ

※１　このサイズはWAVEヘッダ(４バイト)、fmt領域、fact領域、data領域の合計です。
通常は、「ファイルサイズ－８」です。

※２　フォーマットIDが１(リニアPCM)のときは存在しません。

※３　ヘッダ拡張部サイズが０のときは存在しません

※４　fmt、fact、dataはWAVEチャンクのサブチャンクです。

※５　fmt、fact、dataは順不同です。どのような順番で存在しても構いません。

※６　DirectSoundがサポートしているのはフォーマットIDが１のみです。他の形式はACMでリニアPCM形式にデコードする必要があります。

※７　フォーマットIDが１のとき、fmt領域は、fmt領域のサイズを除けば、WAVEFORMATEX構造体からcbSizeメンバを除いたものとまったく同じ構造です。

○フォーマットID

0x0001　リニアPCM	0x0011　ADPCM(IMA/DVI)	0x0022　TrueSpeech
0x0002　MS ADPCM	0x0012　MediaSpace ADPCM	0x0030　DOLBY AC2
0x0005　IBM CSVD	0x0013　Sierra ADPCM	0x0031　GSM 6.10
0x0006　A-Law	0x0014　ADPCM(G.723)	0x0040　ADPCM(G.721)
0x0007　μ-Law	0x001E　MPEG1 Layer-3	0x0200　CREATIVE ADPCM
0x0010　OKI ADPCM	0x0020　YAMAHA ADPCM	0x0300　FM-TOWNS SND

(この他にも多数存在します)

○リニアPCMについて

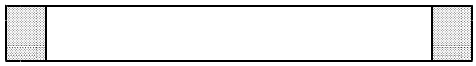
リニアPCM(圧縮されていないPCM)では、１サンプルあたりのビット数は、８と16が存在します。８ビットの場合は0～255でサウンドデータを表し、128(0x80)が無音状態です。16ビットの場合は-32,768～32,767でサウンドデータを表し、０が無音状態です。ステレオデータは、左チャネル、右チャネルの順で格納されています。また、ヘッダ拡張部やfact領域がないものが多く存在します。

サウンドバッファのロックとリングバッファ

Waveファイルのイメージをサウンドバッファに転送するには、サウンドバッファのメモリ領域のアドレス(ポインタ)を取得し、CopyMemory関数やReadFile関数、マルチメディア入出力関数などで直接転送します。

サウンドバッファへのポインタを取得するには、IDirectSoundBuffer8::Lockメソッドを使います。このメソッドは、バッファの一部または全体を安全に書き込めるようにロックし、その領域へのポインタを返します。このとき、２つの領域のポインタを返すことがあります。それは、サウンドバッファがリングバッファになっているためです。

リングバッファ(循環バッファ)とは、概念的にバッファの最後と先頭がつながっているということで、バッファの最後尾の次は先頭に戻ります。たとえば、1000バイトの大きさを持つバッファの先頭から800バイト目を基点に400バイトをロックしたとします。通常のバッファなら、領域の大きさを超えるサイズをロックすることはできませんが、リングバッファである場合はロックすることができ、「800～999」と「0～199」の２つの領域がロックされます。



▲リングバッファが循環する例

Lockメソッドでロックされた領域は、書き込むための領域となります。この領域から読み取ることは許可されていません。バッファがサウンドカード上のメモリに配置されている場合、返されたポインタがシステムメモリにコピーされた一時バッファのアドレスとなる可能性があるからです。この領域からデータを読み込んでもまったく意味がありません。ロックを解除すると一時バッファのデータがサウンドカード上のメモリに転送されます。

書き込みが終わったら、ただちにロックを解除する必要があります。ロックを解除しないと、環境によっては、音声の分断や無音状態が発生する場合があります。ロックの解除は、IDirectSoundBuffer8::Unlockメソッドで行います。

IDirectSoundBuffer8::Lockメソッド

－説明－

Lockメソッドは、サウンドバッファの全部または一部をデータ書き込み用に準備し、データを書き込むためのポインタを返します。

－書式－

```
HRESULT Lock(DWORD dwOffset, DWORD dwBytes, LPVOID* ppvAudioPtr1, LPDWORD pdwAudioBytes1,
             LPVOID* ppvAudioPtr2, LPDWORD pdwAudioBytes2, DWORD dwFlags);
```

－パラメータ－

１つ目の引数(dwOffset)は、バッファの先頭からロック開始位置までのバイト数(オフセット)です。７つ目の引数がDSBLOCK_FROMWRITECURSORフラグの場合は無視されます。

２つ目の引数(dwBytes)は、ロックするバイト数です。７つ目の引数がDSBLOCK_ENTIREBUFFERフラグの場合は無視されます。

３つ目の引数(ppvAudioPtr1)は、バッファの最初にロックされた領域へのポインタを受け取る変数のアドレスです。

４つ目の引数(pdwAudioBytes1)は、最初にロックされた領域のバイト数を受け取る変数のアドレスです。この値が２つ目の引数より小さい場合、ロックは循環され、２番目にロックされた領域があることを表します。

５つ目の引数(ppvAudioPtr2)は、２番目にロックされた領域へのポインタを受け取る変数のアドレスです。循環しない場合、変数の内容がNULLになります。

６つ目の引数(pdwAudioBytes2)は、２番目にロックされた領域のバイト数を受け取る変数のアドレスです。循環しない場合、変数の値が０になります。

７つ目の引数(dwFlags)は、ロックの動作を指定するフラグです。以下のフラグが定義されています。

DSBLOCK_ENTIREBUFFER	バッファ全体をロックします
DSBLOCK_FROMWRITECURSOR	書き込みカーソルの位置からロックします

－戻り値－

成功した場合はDS_OK、それ以外はエラーコードを返します。

IDirectSoundBuffer8::Unlockメソッド

－説明－

Unlockメソッドは、サウンドバッファのロックを解除します。ロックを解除すると、書き込まれたデータが有効になり、書き込みカーソルの位置が更新されます。

－書式－

```
HRESULT Unlock(LPVOID pvAudioPtr1, DWORD dwAudioBytes1,
               LPVOID pvAudioPtr2, DWORD dwAudioBytes2);
```

－パラメーター－

１つ目の引数(pvAudioPtr1)は、Lockメソッドで取得した最初の領域のアドレスです。

２つ目の引数(dwAudioBytes1)は、最初の領域に実際に書き込んだバイト数です。

３つ目の引数(pvAudioPtr2)は、Lockメソッドで取得した２番目にロックされた領域のアドレスです。

４つ目の引数(dwAudioBytes2)は、２番目の領域に実際に書き込んだバイト数です。

－戻り値－

成功した場合はDS_OK、それ以外はエラーコードを返します。

マルチメディア入出力関数

RIFF形式のファイルは、マルチメディア入出力関数(mmio関数)を使うと簡単に扱うことができます。マルチメディア入出力関数では、RIFF形式のファイルを領域(チャンク)単位で扱います。Waveファイルの読み込みでは、以下の関数を使用します。

○mmioOpen(LPSTR szFilename, LPMMIOINFO lpmmioinfo, DWORD dwOpenFlags)

入出力をバッファリングしてファイルを開きます。この関数が返すハンドルは、標準のファイルハンドル(HANDLE)ではないため、マルチメディアファイル入出力関数以外のファイル操作関数では使用できません。

－パラメーター－

LPSTR　　szFilename…ファイル名です。最後のNULLを含めて128バイトを越えることはできません。
LPMMIOINFO lpmmioinfo…追加情報が入ったMMIOINFO構造体のアドレスです。
DWORD　　dwOpenFlags…オープン操作のためのフラグを指定します。

－戻り値－

関数が成功すると、開いたファイルのハンドル(HMMIO型)が返ります。ファイルを開くことができない場合は、NULLが返ります。

○mmioFOURCC(CHAR ch0, CHAR ch1, CHAR ch2, CHAR ch3)

mmioFOURCC関数の実態はマクロです。４つの文字コードを４文字コードに変換します。処理内容は以下のようになっています。

```
((DWORD)(BYTE)(ch0)       | ((DWORD)(BYTE)(ch1) << 8) |
((DWORD)(BYTE)(ch2) << 16) | ((DWORD)(BYTE)(ch3) << 24 ))
```

－パラメーター－

CHAR ch0～ch3…４つの文字コードです。

－戻り値－

４つの文字コードから変換された、４文字コードを返します。

○mmioDescend(HMMIO hmmio, LPMMCKINFO lpck, LPMMCKINFO lpckParent, UINT wFlags)

mmioOpen関数で開いたファイルのチャンク(領域)に進入します。また、指定されたチャンクを検索することもできます。

－パラメーター－

HMMIO　　hmmio…RIFFファイルのハンドルを指定します。
LPMMCKINFO lpck…MMCKINFO構造体のアドレスを指定します。
LPMMCKINFO lpckParent…検索するチャンクの親を識別するMMCKINFO構造体のアドレスを指定します。このパラメータがNULLでない場合、mmioDescend関数が呼び出されて親チャンクに進入すると、mmioDescend関数は参照するMMCKINFO構造体に値が設定されていると想定して、親チャンク内でチャンクを検索します。親チャンクが指定されていない場合、このパラメータはNULLに設定します。

UINT　　wFlags…検索フラグを指定します。フラグが指定されていない場合、mmioDescend関数は現在のファイル位置のチャンクの先頭に進入します。
MMIO_FINDCHUNK　指定されたチャンク識別子のチャンクを検索します。
MMIO_FINDRIFF　親チャンクが"RIFF"で、指定されたサブチャンクを検索します。

－戻り値－
関数が成功するとMMSYSERR_NOERRORが返ります。失敗するとMMIOERR_CHUNKNOTFOUNDが返ります。

○mmioRead(HMMIO hmmio, HPSTR pch, LONG cch)
mmioOpen関数で開いたファイルから、指定されたバイト数を読み取ります。

－パラメーター－
HMMIO hmmio…読み取るファイルのハンドルを指定します。
HPSTR pch…ファイルから読み取られたデータを格納するアドレスを指定します。
LONG　cch…ファイルから読み取るバイト数を指定します。

－戻り値－
関数が成功すると、実際に読み取られたバイト数が返ります。ファイルの終わりに到達し、それ以上読み取ることができない場合は、０が返ります。ファイルの読み取りエラーが発生した場合は－１が返ります。

○mmioAscend(HMMIO hmmio, LPMMCKINFO lpck, UINT wFlags)
mmioDescend関数で進入したチャンクから退出します。

－パラメーター－
HMMIO　　hmmio…RIFFファイルのハンドルを指定します。
LPMMCKINFO lpck…mmioDescend関数で使用したMMCKINFO構造体のアドレスを指定します。
UINT　　wFlags…予約されています。０を指定してください。

－戻り値－
関数が成功するとMMSYSERR_NOERRORが返ります。それ以外はエラーコードが返ります。

○mmioClose(HMMIO hmmio, UINT wFlags)
mmioOpen関数で開いたファイルを閉じます。

－パラメーター－
HMMIO hmmio…閉じるファイルのハンドルを指定します。
UINT　wFlags…クローズ操作のためのフラグを指定します。HMMIO型以外のファイルハンドルを指定して開かれたファイルの場合のみ使用します。

－戻り値－
関数が成功すると、０が返ります。

マルチメディア入出力関数を使用する場合は、ヘッダファイル<mmsystem.h>(または<dmusici.h>)と静的リンクライブラリ"winmm.lib"が必要になります。

課　題

CDXAudio8クラスに、サウンドバッファの生成と解放を行う機能を追加しましょう。

(１)CDXAudio8クラスの適切な場所に、Waveファイルをもとにバッファオブジェクトを生成し、そのインタフェースを返す以下のCreateBufferFromFile関数の足りない部分を補い、追加しましょう。

```
/*****************************************************************************************/
/*                                    バッファ生成                                       */
/*****************************************************************************************/
ISoundBuffer* CDXAudio8::CreateBufferFromFile(LPSTR inFileName)
{
    HMMIO                hFile     = NULL;
    ISoundBuffer*        pBuffer   = NULL;
    IDirectSoundBuffer8* pDSBuffer = NULL;

    try {
        if(m_pDSound == NULL)
            throw "*** Error - DirectX Audio未初期化(CDXAudio8_CreateBufferFromFile)\n";

        // ファイルオープン
```

```
    hFile = ::????????(inFileName, NULL, MMIO_READ | MMIO_ALLOCBUF);
    if(hFile == NULL)
        throw "*** Error - ファイルオープン失敗(CDXAudio8_CreateBufferFromFile)\n";

    MMCKINFO   parent_chunk, child_chunk;
    ::ZeroMemory(&parent_chunk, sizeof(parent_chunk));
    ::ZeroMemory(&child_chunk,  sizeof(child_chunk ));

    // 'WAVE'チャンクへ進入
    parent_chunk.fccType = mmioFOURCC('W', 'A', 'V', 'E');
    if(::mmioDescend(hFile, &parent_chunk, NULL, MMIO_FINDRIFF) != MMSYSERR_NOERROR)
        throw "*** Error - 'WAVE'チャンクが見つかりません(CDXAudio8_CreateBufferFromFile)\n";

    // 'WAVE'チャンクの'fmt 'チャンクへ進入
    child_chunk.ckid = mmioFOURCC('f', 'm', 't', ' ');
    if(::mmioDescend(hFile, &child_chunk, &parent_chunk, MMIO_FINDCHUNK) != MMSYSERR_NOERROR)
        throw "*** Error - 'fmt 'チャンクが見つかりません(CDXAudio8_CreateBufferFromFile)\n";

    // フォーマット情報取得
    WAVEFORMATEX   format;
    if((DWORD)::mmioRead(hFile, (HPSTR)&format, sizeof(format)) != sizeof(format))
        throw "*** Error - フォーマット情報取得失敗(CDXAudio8_CreateBufferFromFile)\n";

    ::mmioAscend(hFile, &child_chunk, 0);   // 'fmt 'チャンク退出

    // 'WAVE'チャンクの'data'チャンクへ進入
    child_chunk.ckid = ??????????('d', 'a', 't', 'a');
    if(::???????????(hFile, &child_chunk, &parent_chunk, MMIO_FINDCHUNK) != MMSYSERR_NOERROR)
        throw "*** Error - 'data'チャンクが見つかりません(CDXAudio8_CreateBufferFromFile)\n";

    // セカンダリバッファ生成
    ???????????   dsbd;
    ::ZeroMemory(&dsbd, sizeof(dsbd));
    dsbd.dwSize          = sizeof(dsbd);
    dsbd.dwBufferBytes   = child_chunk.cksize;
    dsbd.dwReserved      = 0;
    dsbd.lpwfxFormat     = &format;
    dsbd.dwFlags         = DSBCAPS_LOCDEFER | DSBCAPS_CTRLVOLUME | DSBCAPS_CTRLPAN |
                           DSBCAPS_CTRLFREQUENCY | DSBCAPS_GETCURRENTPOSITION2;
    dsbd.guid3DAlgorithm = GUID_NULL;

    // DirectSoundBuffer生成
    IDirectSoundBuffer*   pdsb;
    if(FAILED(※ここは各自考えましょう))
        throw "*** Error - DirectSoundBuffer生成失敗(CDXAudio8_CreateBuffer)\n";

    // IDirectSoundBuffer8インタフェース取得
    const HRESULT   hr = pdsb->????????????????(IID_IDirectSoundBuffer8, (LPVOID*)&pDSBuffer);
    pdsb->Release();
    if(hr != S_OK)
        throw "*** Error - IDirectSoundBuffer8インタフェース取得失敗(CDXAudio8_CreateBuffer)\n";

    // バッファのロック
    LPVOID    pBuf[2];
    DWORD     BufBytes[2];
    if(pDSBuffer->????(0, 0, &pBuf[0], &BufBytes[0], &pBuf[1], &BufBytes[1], DSBLOCK_ENTIREBUFFER)
       != DS_OK)
        throw "*** Error - バッファロック失敗(CDXAudio8_CreateBuffer)\n";

    // イメージ読み込み
    const LONG   READ_BYTES = ::????????(hFile, (HPSTR)pBuf[0], BufBytes[0]);
    pDSBuffer->Unlock(pBuf[0], READ_BYTES != -1 ? READ_BYTES : 0, pBuf[1], 0);
    if(READ_BYTES == -1)
        throw "*** Error - イメージ読み込み失敗(CDXAudio8_CreateBuffer)\n";

    pBuffer = new CSoundBuffer(pDSBuffer);
} catch(LPCSTR ErrorString) {
    ::OutputDebugString(ErrorString);
    // 例外が発生した場合は、NULLオブジェクトを生成
    pBuffer = new CNullSoundBuffer();
}
SafeRelease(pDSBuffer);
if(hFile != NULL)
    ::mmioClose(hFile, 0);
```

```
    // バッファリストへ追加
    m_BufferList.?????????(pBuffer);

    return pBuffer;
}
```

CDXAudio8::CreateBufferFromFile関数の仕様は、以下のとおりです。

CreateBufferFromFile　生成

指定されたWaveファイルをもとに、サウンドバッファを生成します。

書式 ISoundBuffer* CreateBufferFromFile(LPSTR inFileName);

Return	生成されたバッファオブジェクトへのポインタ
inFileName	Waveファイルの名前

(２)以下のプログラムは、CDXAudio8::CreateBufferFromFile関数が生成したバッファオブジェクトを解放するReleaseBuffer関数です。足りない部分を補い、適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                          バッファ解放                                             */
/*****************************************************************************************************/
void CDXAudio8::ReleaseBuffer(ISoundBuffer*& pBuffer)
{
    m_BufferList.??????(pBuffer);   // バッファリストから削除
    delete pBuffer;                 // バッファ解放
    pBuffer = NULL;
}
```

この関数にCDXAudio8::CreateBufferFromFile関数が返したバッファオブジェクトへのポインタを渡すと、バッファリストから検索し、削除されます。

(３)以下のプログラムは、CDXAudio8クラスが管理しているすべてのバッファオブジェクトを解放するReleaseAllBuffers関数です。足りない部分を補い、適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                          全バッファ解放                                           */
/*****************************************************************************************************/
void CDXAudio8::ReleaseAllBuffers()
{
    for(std::list<ISoundBuffer*>::iterator it = m_BufferList.?????(); it != m_BufferList.???(); it++)
        delete *it;
    m_BufferList.clear();
}
```

(４)CDXAudio8クラスの解放時に、すべてのバッファが解放されるように、以下のプログラムを適切な場所に追加しましょう。

```
    // 全バッファ解放
    ReleaseAllBuffers();
```

応用問題１　CDXAudio8クラスに、IDirectSound8::DuplicateSoundBufferメソッドでメモリを共有するサウンドバッファを生成するDuplicateBuffer関数を作成しましょう。

応用問題２　バッファオブジェクトにニックネームを付けられるようにし、ニックネームからバッファオブジェクトを検索できる機能を追加しましょう。