オブジェクト指向と
ゲームプログラミング

▶ DirectX Audio編 ― 第４回　セグメントの再生と停止 ◀

セグメントの再生

セグメントの再生は、IDirectMusicPerformance8::PlaySegmentExメソッドで行います。

IDirectMusicPerformance8::PlaySegmentExメソッド

―説明―

PlaySegmentExメソッドは、セグメントの再生を開始します。

―書式―

```
HRESULT PlaySegmentEx(IUnknown* pSource, WCHAR* pwzSegmentName, IUnknown* pTransition,
                      DWORD dwFlags, __int64 i64StartTime,
                      IDirectMusicSegmentState** ppSegmentState,
                      IUnknown* pFrom, IUnknown* pAudioPath);
```

―パラメータ―

１つ目の引数(pSource)は、再生を開始するセグメントのインタフェースを指定します

２つ目の引数(pwzSegmentName)は、予約されておりNULLにします。

３つ目の引数(pTransition)は、テンプレートセグメントを指定します。通常は使用しないのでNULLにします。

４つ目の引数(dwFlags)は、動作フラグです。おもに以下のフラグを指定します。

0	プライマリセグメントで再生します。再生優先度がもっとも高いので、演奏時間が長いデータは必ずこのモードを使用します。ただし、プライマリとして指定できるのは、１つのパフォーマンスにつき１つのセグメントだけです。
DMUS_SEGF_SECONDARY	セカンダリセグメントで再生します。複数のセグメントを同時に再生することができます。ただし、演奏時間が長いデータは、テンポずれなどの不具合が起こる場合があります。
DMUS_SEGF_QUEUE	現在再生されているプライマリセグメントのデータが終了次第、即座に演奏を開始します。

BGMなどの大きなデータはプライマリ、効果音などの小さなデータはセカンダリで演奏します(データの組み合わせによっては、同時に演奏できない場合があります)。

５つ目の引数(i64StartTime)は、再生が開始されるまでのパフォーマンス時間です。０を指定するとできる限り早く開始されます。

６つ目の引数(ppSegmentState)は、セグメントの再生状態を取得する場合に使用します。使用しない場合はNULLにします。

７つ目の引数(pFrom)は、再生開始時に再生を停止するセグメントまたはオーディオパスのインタフェースを指定します。使用しない場合はNULLにします。

８つ目の引数(pAudioPath)は、再生に使うオーディオパスのインタフェースを指定します。デフォルトのオーディオパスで再生する場合はNULLにします。

―戻り値―

成功した場合はS_OK、それ以外はエラーコードを返します。

```
// セグメントの再生
IDirectMusicPerformance8*   pDMPerformance;      // パフォーマンス
IDirectMusicSegment8*       pDMSegment;         // セグメント
pDMPerformance->PlaySegmentEx(pDMSegment, NULL, NULL, DMUS_SEGF_SECONDARY, 0, NULL,
                              NULL, NULL);      // セカンダリセグメントとして再生
```

セグメントごとにループ回数を設定することができます。設定は、IDirectMusicSegment8::SetRepeatsメソッドで行います。引数はループ回数で、DMUS_SEG_REPEAT_INFINITEで無限ループ、０はループなし、１以上はその回数だけループします。指定回数ループすると再生は停止します。

```
// ループ回数の設定
pIDMSegment8->SetRepeats(DMUS_SEG_REPEAT_INFINITE);      // 無限ループ
```

セグメントの停止

セグメントの停止は、IDirectMusicPerformance8::StopExメソッドで行います。

IDirectMusicPerformance8::StopExメソッド

－説明－

StopExメソッドは、セグメントの再生を停止します。

－書式－

HRESULT StopEx(IUnknown* *pObjectToStop*, __int64 *i64StopTime*, DWORD *dwFlags*);

－パラメータ－

１つ目の引数(pObjectToStop)は、再生を停止するセグメントのインタフェースを指定します。NULLを指定すると、ダウンロードされているすべてのセグメントの再生が停止されます。

２つ目の引数(i64StopTime)は、停止するまでのパフォーマンス時間です。０を指定すると即座に停止されます。

３つ目の引数(dwFlags)は、停止フラグです。通常は０を指定します。

－戻り値－

成功した場合はS_OK、失敗した場合はE_POINTERを返します。

```
// 再生の停止
pDMPerformance->StopEx(pDMSegment, 0, 0);
```

ボリュームの設定

パフォーマンス全体のボリュームは、IDirectMusicPerformance8::SetGlobalParamメソッドで設定することができます。１つ目の引数に「GUID_PerfMasterVolume」、２つ目の引数にボリューム値を格納した変数のアドレス、３つ目の引数に２つ目の引数で指定した変数のサイズ(バイト数)を指定します。ボリューム値は+2000～-20000の範囲で指定でき、０が元のボリューム値です。しかし、実際に使用できる範囲は+1000～-10000です。このメソッドは、ダウンロードされているすべてのセグメントに効果があります。セグメントごとのボリュームを設定することはできません。

```
// ボリュームの設定
int    Volume = -1000;        // ボリュームを小さくする
pDMPerformance->SetGlobalParam(GUID_PerfMasterVolume, (LPVOID)&Volume, sizeof(Volume));
```

課　題

CMusicSegmentクラスに、セグメントの再生と停止機能を追加しましょう。

(１)再生はPlay関数、停止はStop関数として作成します。Play関数には、動作フラグとループ回数を渡します。ループ回数が正の場合はその回数だけループし、負の場合は無限ループするようにします。
これらをふまえると、各関数のプロトタイプは以下のようになります。IMusicSegmentクラスの適切な場所に追加しましょう。

```
virtual bool Play(const DWORD inFlag = 0, const int inLoop = -1) = 0;
virtual void Stop() = 0;
```

(２)CMusicSegmentクラスに、Play関数とStop関数を実装します。以下のプログラムを適切な場所に追加しましょう。

```
virtual bool Play(const DWORD inFlag, const int inLoop);
virtual void Stop() { m_pDMPerformance->??????(m_pDMSegment, 0, 0); }
```

(３)CMusicSegment::Play関数の足りない部分を補い、適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                              再　生                                               */
/*****************************************************************************************************/
bool CMusicSegment::Play(const DWORD inFlag, const int inLoop)
{
    Stop();

    // ループ設定
    if(inLoop >= 0)
        ※ここは各自考えましょう;
    else
        ※ここは各自考えましょう;

    // 再生
    if(※ここは各自考えましょう[inFlagを忘れずに使いましょう] != S_OK) {
        ::OutputDebugString("*** Error - 再生失敗(CMusicSegment_Play)\n");
        return false;
    }

    return true;
}
```

CMusicSegment::Play関数、Stop関数ともに、m_pDMPerformanceやm_pDMSegmentがNULLかどうかを調べるコードがありません。これが、Null Objectパターンの効果です。

CMusicSegment::Stop関数では、自分で処理するのではなく、そのままm_pDMPerformanceのメソッドに任せています。このような動作を委譲(delegation)と呼びます。

(４)CNullMusicSegmentクラスに、「何もしない」Play関数とStop関数を実装します。以下のプログラムを適切な場所に追加しましょう。

```
    virtual bool Play(const DWORD inFlag, const int inLoop) { return true; }
    virtual void Stop() {}
```

(５)Play関数に渡す動作フラグは、「DMUS_SEGF_SECONDARY」と長く、覚えにくいので、以下のように短縮名を作成しておくと便利です。左辺は扱いやすい名前に変更し、MusicSegment.hppの適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                    ミュージックセグメント再生フラグ                                 */
/*****************************************************************************************************/
enum DXAMUSICSEGMENT_PLAYFLAGS {
    DXAMS_PRIMARY   = 0,                        // プライマリ
    DXAMS_SECONDARY = DMUS_SEGF_SECONDARY,      // セカンダリ
    DXAMS_QUEUE     = DMUS_SEGF_QUEUE           // 再生キューに登録
};
```

(６)セグメントが正しく動作するか確認します。以下のようにプログラムを追加、変更しましょう。

・CTestSceneクラスのメンバに追加

```
IMusicSegment*   m_pBGM;
```

・CTestScene::CTestScene関数に追加

```
m_pBGM = DXAudio().CreateSegmentFromFile("Wav\\BGM.wav");
m_pBGM->Play();        // m_pBGM->Play(DXAMS_PRIMARY, -1)と同じ意味
```

・CTestScene::~CTestScene関数に追加

```
DXAudio().ReleaseAllSegments();
```

複数のセグメントが同時に再生できるかどうかもテストしましょう。