オブジェクト指向と
ゲームプログラミング

▶ DirectX編 － 第２回　COM ◀

COM

DirectXは、COM(Component Object Model)と呼ばれる仕様で実装されています。

アプリケーションに新しい機能を追加したりバグが見つかった場合、全体を見直して関連のない部分に影響がないように注意しながら改良を加え、全体を再コンパイルするという作業は、アプリケーションの肥大化に伴い、非常にコストが掛かるようになりました。そこで、アプリケーションをいくつかのコンポーネント(部品)に分解し、それらを融合した形でアプリケーションを構成しようという発想が生まれました。こうしておけば全体を再構築する必要はなく、改良を加えたコンポーネントを交換するだけで済みます。また、機能を追加するときも、それを１つのコンポーネントとして作って埋め込むことが可能です。
このようなコンポーネントスタイルのアプリケーションを構築するための設計仕様がCOMです。言語やコンポーネントそのものを指すわけではありません。
COMの設計思想には、以下の２つの特徴があります。

・ダイナミックリンクが行える
・カプセル化されている

COMは、ダイナミックリンクという機能を持っています。ダイナミックリンク(動的連結)とは、コンパイル時にコンポーネントをリンク(スタティックリンク：静的連結)するのではなく、実行時に、必要に応じてメモリに読み込んでリンクすることをいいます。この機能により、コンポーネントが置き換え可能になります。たとえば、3DCGの機能が必要になった部分でDirect3Dコンポーネントを読み込んで連結し、必要がなくなったらその連結を解除してメモリを解放します。ネットワーク機能が必要になったらDirectPlayコンポーネントを読み込んで連結、というようなことが可能です。また、インストールされているコンポーネントのバージョンを調べ、バージョンに合わせた処理を行わせることもできます。

次のカプセル化は、閉じたアプリケーションシステムの中では必要ありませんが、COMを堅牢にするために必要な機能です。コンポーネントが動的にリンクできても、バージョンアップによりコンポーネントとアプリケーションのインタフェースが変更されてしまっては、プログラムの書き直しが必要になってしまいます。そのため、一度公開したCOMのインタフェースは、変更されないことが保証されています。そして、公開されたコンポーネントの内部は完全なブラックボックスとなっており、どのような実装か意識する必要はありません。これがカプセル化の意味とその利点です。

インタフェース

コンポーネント同士またはアプリケーションとコンポーネントをつなぐインタフェースとは、関数のことです。これらの関数はコンポーネント内で実装され、アプリケーションで使用されます。
COMでは、このインタフェースは関数が格納された領域のアドレス(関数へのポインタ)の配列を表します。このインタフェースの部分は、純粋仮想関数のみで構成された抽象クラスを基底クラスとして使用しています。インタフェースは関数の部分に実体はなく、実体はコンポーネントの中にあります。インタフェースでは、その関数へのポインタしか含まれていません。
インタフェースは関数へのポインタの集合なので、型の実体は構造体です。ですから、IDirectDrawというインタフェースを取得するということは、DirectDrawコンポーネントが持つ関数の集合体の先頭アドレスを取得することになります。

インタフェースの取得

あるコンポーネントの機能を必要としているアプリケーション、あるいはコンポーネントのことをクライアントと呼びます。クライアントは、ある機能が必要になったら、その機能を持つコンポーネントのインスタンスを生成しなければなりません。

インスタンスの生成は、CoCreateInstance関数で行います。

CoCreateInstance関数

－説明－

CoCreateInstance関数は、COMオブジェクトのインスタンスを生成し、そのインタフェースを取得します。

－書式－

```
STDAPI CoCreateInstance(REFCLSID rclsid, LPUNKNOWN pUnkOuter, DWORD dwClsContext,
                        REFIID riid, LPVOID * ppv)
```

－パラメータ－

１つ目の引数(rclsid)は、生成するオブジェクトのクラス参照識別子(クラスID)です。

２つ目の引数(pUnkOuter)は、生成するオブジェクトがあるオブジェクトの集合体の一部である場合に、集合元のインタフェースを指定します。ほとんどの場合使用しないので、NULLにします。

３つ目の引数(dwClsContext)は、生成したオブジェクトがどのような場所(コンテキスト)で実行されるのかを指定します。以下の値のいずれかを指定しますが、DirectXの場合は「CLSCTX_INPROC_SERVER」を指定します。

CLSCTX_INPROC_SERVER	オブジェクトは、同じプロセス空間で実行されます。
CLSCTX_INPROC_HANDLER	オブジェクトは、インプロセスハンドラです。
CLSCTX_LOCAL_SERVER	オブジェクトは、EXEとは別のプロセス空間(同じマシン上の別のプロセス)で実行されます。
CLSCTX_REMOTE_SERVER	オブジェクトは、リモートマシン上で実行されます。
CLSCTX_INPROC	(CLSCTX_INPROC_SERVER | CLSCTX_INPROC_HANDLER)
CLSCTX_SERVER	(CLSCTX_INPROC_SERVER | CLSCTX_LOCAL_SERVER | CLSCTX_REMOTE_SERVER)
CLSCTX_ALL	(CLSCTX_INPROC_SERVER | CLSCTX_INPROC_HANDLER | CLSCTX_LOCAL_SERVER | CLSCTX_REMOTE_SERVER)

４つ目の引数(riid)は､生成するオブジェクトのインタフェース参照識別子(インタフェースID)です。

５つ目の引数(ppv)は、生成したオブジェクトのインタフェースを格納する変数のアドレスです。

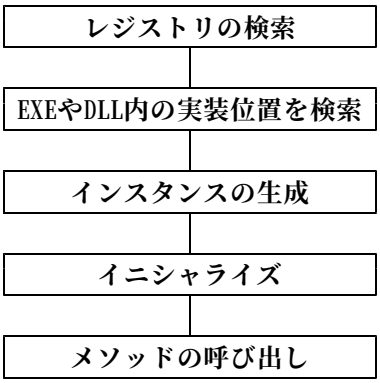
－戻り値－

関数が成功するとS_OK、それ以外はエラーの原因を返します。

```
// DirectDraw7インスタンスの生成とインタフェースの取得
IDirectDraw7*   pDDraw;
CoCreateInstance(CLSID_DirectDraw, NULL, CLSCTX_INPROC_SERVER,
                 IID_IDirectDraw7, (LPVOID*)&pDDraw);
```

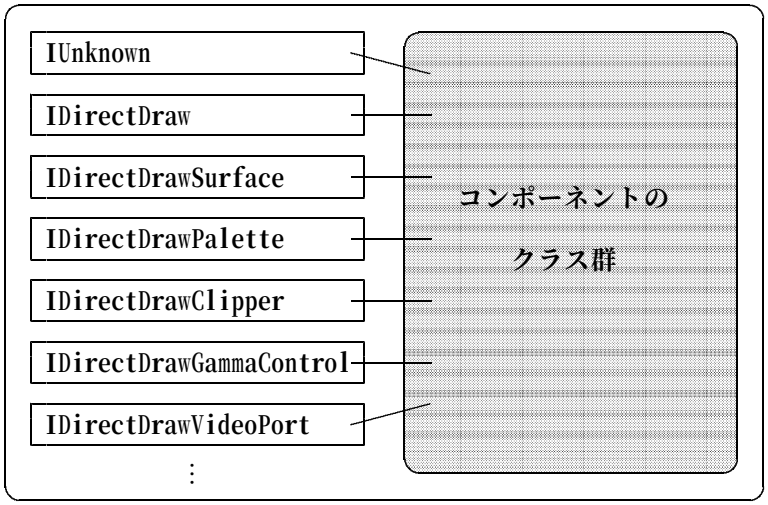
CoCreateInstance関数を呼び出すと、COMが含まれているEXEやDLLを探すために、レジストリからクラスIDの検索を行います。レジストリ内の<HKEY_CLASSES_ROOT\CLSID>内にさまざまなCOMのクラスIDが格納されています。この中からCoCreateInstance関数は、使用するEXEやDLL名を調べます。EXEまたはDLLが見つかると、それらを読み込み、そのコンポーネントのインスタンスをメモリ(プロセス空間)に生成します。

インスタンスを生成しただけでは、まだその機能を使うことができません。COMでは、さらにイニシャライズ(初期化)という作業が必要になります。イニシャライズをすることで、各コンポーネントが提供するメンバ関数(メソッド)を呼び出すことが可能になります。



DirectXのコンポーネントはDLLで与えられており、そのインスタンスを生成する関数がDLLからエクスポートされています(DirectDraw, Direct3D, DirectInput, DirectSoundのみ)。たとえば、DirectDrawコンポーネントには、DirectDrawCreateEx関数がエクスポートされており、DLLの外から呼び出すことができるようになっています。
このような関数は、インスタンスの生成だけでなくイニシャライズまで行うので非常に便利です。しかし、該当するバージョンのDLLが無いと、警告ダイアログボックスを表示した後、アプリケーションが強制終了されてしまうという欠点もあります。

コンポーネントが読み込まれ、イニシャライズされても、実はまだそのコンポーネントのすべての機能が使えるわけではありません。たとえば、DirectDrawコンポーネントは、ddraw.dllに複数のコンポーネントの集合として実装されています。CoCreateInstance関数やDirectDrawCreateEx関数でDirectDrawコンポーネントのインスタンスを生成しても、最も基本的な機能だけを提供するIDirectDrawインタフェイスしか取得できません。この段階では、DirectDrawSurfaceやDirectDrawClipperなどの機能を使いたくても使えません。これは、常に必要最低限のコンポーネントのみ用意するという設計思想によるものです。
では、どうやってほかのコンポーネントを取得するのかというと、ひとつはQueryInterfaceメソッドを使います。COMコンポーネントには、必ずQueryInterfaceメソッドが用意されています。QueryInterfaceメソッドは、同じコンポーネント内であれば、どのインタフェースからもすべてのインタフェースが取得できるようになっています。このメソッドのほかに、CreateSurfaceメソッドのような別のコンポーネントのインタフェースを取得できるメソッドが用意されている場合もあります。



COMインタフェースの概念

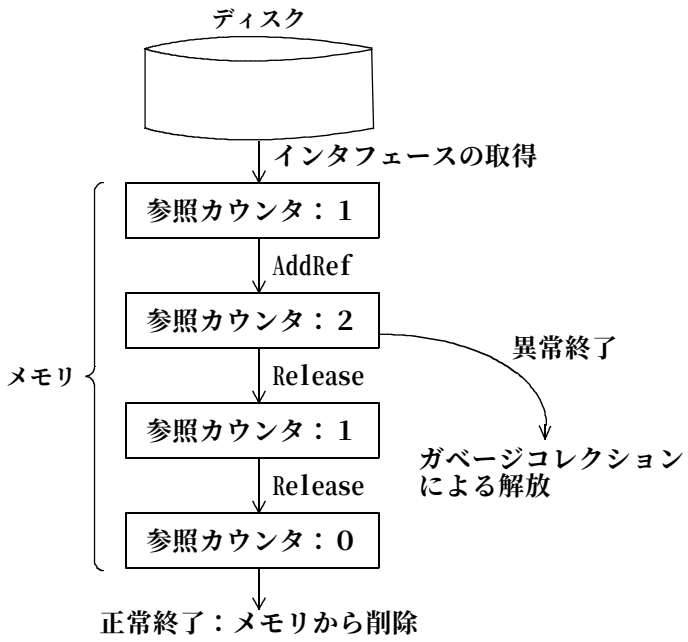
COMインタフェースで定義されている規格として、すべてのインタフェースはIUnknownインタフェースを継承しなければならないとされています。このIUnknownインタフェースは、次のように定義されています。

```
class IUnknown {
    virtual HRESULT QueryInterface(REFIID riid, LPVOID* ppv) = 0;
    virtual ULONG AddRef() = 0;
    virtual ULONG Release() = 0;
};
```

この３種類のインタフェースを用いてCOMを利用し、また作成していきます。AddRefメソッドとReleaseメソッドは、オブジェクト内部の参照カウンタを制御するために提供されています。
AddRefメソッドは、オブジェクトが参照されたことを通知し、参照カウンタを１つ増します。Releaseメソッドはオブジェクトが解放されたことを通知し、参照カウンタを１つ減らします。カウンタが０になったときは、コンポーネントがどこからも参照されていないことを示し、メモリから削除されます。

インタフェースの取得、参照のたびに参照カウンタは増やされます。不用になったインタフェースはReleaseメソッドで解放します。インタフェースの取得と解放を正しく行っていれば、正常終了時には特に問題ありませんが、異常終了した場合などは、参照カウンタを減らすことなくアプリケーションが終了してしまうので、オブジェクトはメモリから削除されないように思えます。COMでは、このような

場合、どのアプリケーションからも使用されていないコンポーネントを自動的に解放します。このような動作をガベージコレクションと呼びます。



COMライブラリの初期化と解放

CoCreateInstance関数のように、先頭に"Co"の文字がつくものはCOM関連の関数です。このような関数は、COMライブラリを初期化してから呼び出さないと、必ず失敗してしまいます。COMライブラリの初期化は、CoInitialize関数を呼び出すことで行います。

```
// COMライブラリ初期化
CoInitialize(NULL);
```

COMを使用する必要がなくなった場合は、CoUninitialize関数を呼び出し、COMが使用したリソースを解放します。

```
// COMライブラリ解放
CoUninitialize();
```

ここで問題となるのが、誰がCOMライブラリの初期化と解放を行うのかという点です。たとえば、描画のCDirectDrawクラスとサウンドのCDirectSoundクラスがあるとします。CDirectDrawクラスのコンストラクタでは、CoCreateInstance関数を呼び出し、COMライブラリの初期化を行います。当然、CDirectSoundクラスのコンストラクタでもCoCreateInstance関数を呼び出すように設計します。このような設計にすると、CoCreateInstance関数が複数回呼び出されてしまいます。CoCreateInstance関数を複数回呼び出すと、正しく初期化が行われません。CoCreateInstance関数を呼び出す前には、COMライブラリが解放されている状態でなければならないのです。だからといって、CDirectDrawクラスとCDirectSoundクラスのどちらかのみでCoCreateInstance関数を呼び出す、という設計にしてはいけません。COMライブラリの参照カウンタを設けるなどの工夫が必要となります。

SUCCEEDEDマクロとFAILEDマクロ

COMの関数やメソッドでは、戻り値として「成功を示す値」と「失敗を示す値」のほかに、「成功ではないが成功と見なしてもかまわない値」を返す場合があります。

メソッドの呼び出しが成功すると、D3D_OK, DI_OK, S_OKといった成功を示す値が返されます。メソッドの中には、これら以外の値でも成功とみなしてもかまわない値を返す場合があります。このような値を含めた成否を判定するために、SUCCEEDEDマクロとFAILEDマクロが提供されています。どちらのマクロも引数に関数やメソッドの戻り値を与えます。

SUCCEEDEDマクロは、関数やメソッドの戻り値が成功または成功とみなしてもかまわない値なら真を返します。FAILEDマクロはその逆で、メソッドの戻り値が成功を示す値でも成功とみなしてもかまわない値でもないとき(すなわち、完全に失敗のとき)に真を返します。

```
if(CoCreateInstance(省略) == S_OK) {
    // S_OKのみ成功
}

if(CoCreateInstance(省略) != S_OK) {
    // S_OK以外は失敗(成功と見なしてもかまわない値も失敗と見なす)
}

if(SUCCEEDED(CoCreateInstance(省略))) {
    // 関数呼び出しが成功
}

if(FAILED(CoCreateInstance(省略))) {
    // 関数呼び出しが失敗
}
```

DirectXでは、成功とみなしてもかまわない値を返すメソッドが多くあるので、すべてのメソッド呼び出しでSUCCEEDEDマクロかFAILEDマクロで成否を判定しても構いません。

課　　題

COMライブラリの初期化と解放を参照カウンタで管理するクラスCComLibraryを作成しましょう。

(１)CComLibraryクラスのヘッダファイル(ComLibrary.hpp)を以下のように作成しましょう。

－ComLibrary.hpp－

```
/*
===========================================================================================
                                オブジェクト指向ゲームプログラミング
    Programmed by Hibikino software. Copyright (c) 2005 Hibikino software. All rights reserved.
===========================================================================================

  【対象ＯＳ】
    Microsoft Windows2000/XP

  【コンパイラ】
    Microsoft Visual C++ 2005

  【プログラム】
    ComLibrary.hpp
                COMライブラリクラスヘッダ

  【履歴】
    * Version    1.00       2005/03/dd  hh:mm:ss

===========================================================================================
*/

#pragma once

/*******************************************************************************************/
/*                                      インクルードファイル                               */
/*******************************************************************************************/
#include <objbase.h>

/*******************************************************************************************/
/*                                    COMライブラリクラス定義                              */
/*******************************************************************************************/
class CComLibrary {
public:
    static bool Initialize();
    static void Uninitialize();

private:
```

```
    static int    m_Ref;      // 参照カウンタ
};
```

(２)CComLibraryクラスのソースファイル(ComLibrary.cpp)の足りない部分を補い、完成させましょう。

－ComLibrary.cpp－

```
/*
=========================================================================================
                                オブジェクト指向ゲームプログラミング
     Programmed by Hibikino software. Copyright (c) 2005 Hibikino software. All rights reserved.
=========================================================================================

  【対象ＯＳ】
     Microsoft Windows2000/XP

  【コンパイラ】
     Microsoft Visual C++ 2005

  【プログラム】
     ComLibrary.cpp
                COMライブラリクラス

  【履歴】
     * Version    1.00       2005/03/dd  hh:mm:ss

=========================================================================================
*/

/*****************************************************************************************/
/*                                    インクルードファイル                                 */
/*****************************************************************************************/
#include "ComLibrary.hpp"

/*****************************************************************************************/
/*                                      クラス変数                                        */
/*****************************************************************************************/
※ここは各自考えましょう

/*****************************************************************************************/
/*                                   COMライブラリ初期化                                   */
/*****************************************************************************************/
bool CComLibrary::Initialize()
{
    m_Ref++;
    if(m_Ref == 1) {
        if(::※ここは各自考えましょう != S_OK) {
            ::OutputDebugString("*** Error - COMライブラリ初期化失敗(CComLibrary_Initialize)\n");
            m_Ref = 0;
            return false;
        }
    }

    return true;
}

/*****************************************************************************************/
/*                                   COMライブラリ解放                                     */
/*****************************************************************************************/
void CComLibrary::Uninitialize()
{
    if(m_Ref <= 0)
        return;

    m_Ref--;
    if(m_Ref == 0)
        ::※ここは各自考えましょう;
}
```

(３)CComLibraryクラスのメンバは、以下のとおりです。

Initialize 初期化

参照カウンタを１つ増やし、参照カウンタが１の場合にはCOMライブラリの初期化を行います。

Uninitialize 解放

参照カウンタを１つ減らし、参照カウンタが０の場合にはCOMライブラリの解放を行います。

m_Ref クラス変数

COMライブラリの参照カウンタです。

```
static int m_Ref;
```

(４)CComLibraryクラスは、Monostateパターンで設計されています。Monostateパターンが適切であるかどうかを検討し、必要ならSingletonパターンに変更しましょう。