ComptrとDirectXの話

マイクロソフト界隈のプログラミング環境で、度々見かける正体不明の言葉たち。
DirectX関連のコードを書いているとちょいちょい見かけて、知らんうちに使ってるけど正体がよくわかっていない怖いやつ**「COMオブジェクト」**について

COMオブジェクトって何ぞ？（ていがくねんむけせつめい）

☒まず「COM」ってなに？

COM（コム）は「**Component Object Model**（コンポーネント オブジェクト モデル）」の略です。
でもそんなカタい言葉、いったん忘れてください。

☒ COMオブジェクトは「お弁当のおかず」

たとえば「ゲームを作る」としましょう。

- 自分で全部作るのは大変なので、
- 「誰かが作ったキャラ描画システム」や「音を鳴らす仕組み」を借りたいとします。

それが、**「COMオブジェクト」**です。

☒ COMオブジェクトは「貸してくれる便利な道具」

- 「絵を描く道具」
- 「音を出す道具」
- 「データを保存する道具」

みたいな**「完成された便利な道具（部品）」**を、他の人（Microsoftなど）が作っていて、
それを**「COMオブジェクト」として使えるようにしてくれてる**んです。

☒「共通のルールで話せる友達」

COMオブジェクトにはルールがあります。

- 誰が使っても動くように、
- どんな言語（C++、C#、VB）でも使えるように、
- 「同じ約束（インターフェース）」でやりとりするんです。

だから**「知らない人が作ったプログラムの部品」**でも、安心して呼び出して使えます。

☒ まとめ

比喩	意味
おかず（道具）	COMオブジェクト
お弁当箱	自分のアプリ
おかずの受け渡しのルール	COMの決まり（インターフェース）
作ってくれた人	Microsoftや他の開発者

☒ 一言で言うと

> **「COMオブジェクト」とは、共通のルールで作られた便利な部品のこと。いろんなソフトから呼び出して使える。**

もう少し勉強した人向けの説明

☒ COMオブジェクトとは何か？

✳ ざっくり一言でいうと：

> **「異なるプログラミング言語やアプリケーション同士が、共通の仕組みでオブジェクトをやりとりできるようにした技術」**
> → それが **COM（Component Object Model）**
> → COMを通じて使えるインスタンス（機能の塊）が **COMオブジェクト**

☒ COMが目指すこと

プログラムの世界には「部品の再利用」や「モジュール化」が大事です。

でも、

- C++で書かれたモジュール
- VBで書かれたアプリ
- スクリプトで動かしたい自動処理

これらが**言語や仕組みを超えて、共通のやり方で連携できる**ようにしたのがCOMです。

☒ COMオブジェクトの仕組み（ざっくり）

1. **インターフェース（IUnknownやIDispatch）**
   - COMオブジェクトには最低1つ、**インターフェース**があります。
   - 利用者はインターフェース越しにしかCOMオブジェクトにアクセスできません。
   - 実装の中身は隠蔽されていて見えません（カプセル化）。
2. **GUID（Globally Unique Identifier）**
   - COMオブジェクトやインターフェースには一意のID（GUID）が振られています。
   - これにより、OSや他アプリから動的に識別できます。
3. **参照カウント（Reference Counting）**
   - COMは内部的にAddRef/Releaseを通じてオブジェクトの寿命を管理します（ガベージコレクションはしません）。

☒ よく使われる場面

- **DirectX**（Direct3DなどはCOMベース）
- **Office VBA**（Excel.Application などもCOMオブジェクト）
- **Windows APIの一部**
- **ActiveXコントロール**

☒ なぜ今でも使われる？

- Windows OSレベルで深く組み込まれている（レジストリ、OLE、自動化）
- いまだに **多くの既存ライブラリがCOMで提供されている**
- C++などで細かく制御したい場面に強い（低レイヤ）

☒☒ まとめ

項目	内容
目的	言語を超えて再利用できる「共通のオブジェクト化」技術
コア概念	インターフェースベース、参照カウント、GUID
主な用途	DirectX、Office自動化、ActiveX、WinAPI

メリット	再利用性、互換性、Windowsとの親和性
デメリット	実装がやや複雑（CoCreateInstance、IUnknown、HRESULT...）

ComPtrについて

☒ 前提まとめ：DirectXはCOMベース

まず前提として：

- **DirectX（特に Direct3D）** は **COMベースのAPI** です。
- つまり、Direct3Dの各機能（デバイス、バッファ、シェーダなど）は**COMオブジェクトとして提供されている**。
- 例：`ID3D11Device`, `ID3D11Texture2D`, `IDXGISwapChain` など

☒COMの注意点：メモリ管理が自分持ち

COMオブジェクトは自動で解放されません。
代わりに `AddRef()` と `Release()` で **参照カウント管理** を自前で行う必要があります。

```
ID3D11Device* device = nullptr;
CreateDevice(..., &device);
// 使い終わったら手動で解放
device->Release();
```

これをうっかり忘れると **メモリリーク** や **クラッシュ** の原因になります。

☒ ComPtr とは？（Microsoft::WRL::ComPtr）

> **ComPtrは、COMオブジェクトをスマートポインタで安全に扱うためのラッパークラス**です。

`ComPtr<T>` を使えば、参照カウントの管理を自動でやってくれます！

☒ ComPtrがやってくれること：

- `AddRef()` / `Release()` を自動で呼ぶ
- 所有権の移動（Move）やコピー（Copy）も安全に処理
- `nullptr` チェックや `QueryInterface()` も簡潔にできる

☒ 使用例（素のポインタ vs ComPtr）

【従来】生ポインタでの DirectX デバイス生成：

```
ID3D11Device* device = nullptr;
ID3D11DeviceContext* context = nullptr;

D3D11CreateDevice(..., &device, &context);
// ...使う
device->Release();
context->Release();
```

【今どき】ComPtrを使った安全なやり方：

```
#include <wrl/client.h>
using Microsoft::WRL::ComPtr;

ComPtr<ID3D11Device> device;
ComPtr<ID3D11DeviceContext> context;

D3D11CreateDevice(..., device.GetAddressOf(), context.GetAddressOf());
```

```
// 解放は自動（RAII）で行われる
```

> `GetAddressOf()` は `T**` が必要なAPIのために ComPtr の中身のアドレスを渡す関数です。

☒ DirectX × ComPtr：なぜ使うのが当たり前？

☒ メリットまとめ：

ComPtr の利点	解説
自動メモリ管理	`Release()` を書かなくていい。例外にも安全
スマートなコード	`if (device)` のようにそのままチェックできる
RAIIに沿う	C++のモダンな設計思想とマッチ
安心してAPI連携	`GetAddressOf()` で API に渡しやすい

☒ 最後に一言でまとめると…

> **DirectXはCOMでできている。COMの参照カウント管理は手間がかかる。ComPtrを使えばそれが自動で、ミスも減る。だからプロでも標準的に使われている。**

Revision #2
★Created 10 June 2025 00:27:32 by youe2
✎Updated 13 June 2026 03:33:29 by youe2