# Windowsプログラミング

みんな大好きWindowsプログラミングについてのいろいろ

# 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。**

# Windowsプログラミングについて

## Windowクラス

### 種類

- WNDCLASSEXA
- WNDCLASSEXW
- WNDCLASSA
- WNDCLASSW

### 構造体メンバ

| | 列-A | 列-B |
|---|---|---|
| 行-1 | A-1 | B-1 |
| 行-2 | A-2 | B-2 |
| 行-3 | A-3 | B-3 |
| 行-3 | A-3 | B-3 |
| 行-3 | A-3 | B-3 |
| 行-3 | A-3 | B-3 |
| 行-3 | A-3 | B-3 |

### このパラメタを変えるとこうなる！

## WinMainの引数の意味

- hInstance
  - aaa
- hPrevInstace
  - aaa
- lpCmdLine
  - aaa
- nCmdShow
  - aaa

## CreateWindowExの引数と戻り値

- dwExStyle
- lpClassName
- lpWindowName
- dwStyle
- X
- Y
- nWidth
- nHeight
- hWndParent
- hMenu
- hInstance
- lpParam

## メッセージループとコールバック関数の関係と仕組み

### メッセージループ

### コールバック

# XMVECTORとXMFLOATどっち？問題

## XMVECTOR

`XMVECTOR`は、DirectXMathライブラリの中核をなすデータ型です。その特徴は以下の通りです。

- **SIMD (Single Instruction, Multiple Data) 最適化**: 複数の浮動小数点数を同時に処理できるCPU命令（SSE/AVXなど）を利用するために設計されています。これにより、ベクトル演算や行列演算が非常に高速に行われます。
- **アラインメント**: SIMD命令が効率的に動作するために、通常16バイト境界にアラインメントされています。
- **レジスタでの操作**: 多くの場合、CPUのレジスタ上で直接操作されるため、メモリアクセスによるオーバーヘッドが少ないです。
- **直接的なアクセスが難しい**: `XMVECTOR`の個々の要素に直接アクセス（例: `vec.x`）することは通常できません。要素にアクセスするには、`XMVectorGetX()`, `XMVectorSetX()`などの関数を使用する必要があります。

**`XMVECTOR`を使うべき場面**:

- **ベクトルや行列の計算**: 加算、減算、乗算、内積、外積、変換など、あらゆる数学的な演算を行う場合。
- **シェーダーへの定数バッファのアップロード**: シェーダーで利用するベクトルや行列データは、通常`XMVECTOR`で計算し、`XMFLOAT4`などで構成された定数バッファに格納してからGPUへ送ります。
- **一時的な計算結果**: 関数内で中間的な計算を行う場合。

**例**:

C++

```
#include <DirectXMath.h>

// XMVECTOR同士の加算
XMVECTOR v1 = XMVectorSet(1.0f, 2.0f, 3.0f, 0.0f);
XMVECTOR v2 = XMVectorSet(4.0f, 5.0f, 6.0f, 0.0f);
XMVECTOR vResult = XMVectorAdd(v1, v2);

// 行列とベクトルの乗算
XMMATRIX m = XMMatrixRotationY(XM_PIDIV4);
XMVECTOR transformedVec = XMVector3TransformCoord(v1, m);
```

## XMFLOAT (例: XMFLOAT3, XMFLOAT4)

`XMFLOAT`シリーズのデータ型（`XMFLOAT2`, `XMFLOAT3`, `XMFLOAT4`など）は、C++の通常の構造体として定義されています。

- **メモリ上でのデータ格納**: メモリ上にデータをシンプルに格納するために使用されます。
- **アクセスしやすい**: `float x, y, z, w;`のようにメンバが直接公開されているため、個々の要素にアクセスしやすいです。
- **アラインメントの制約が少ない**: 特別なアラインメントを考慮する必要がない場合が多いです（ただし、定数バッファなどGPUに送るデータの場合はアラインメントが重要になります）。
- **SIMD最適化はされない**: `XMFLOAT`自体はSIMD最適化の恩恵を受けません。

**`XMFLOAT`を使うべき場面**:

- **構造体やクラスのメンバ変数**: オブジェクトの位置、回転、スケールなど、永続的に保持したいデータ。
- **頂点バッファのレイアウト**: 頂点データ（位置、法線、UVなど）を定義する場合。
- **ファイルへの保存や読み込み**: データをファイルに書き込んだり、ファイルから読み込んだりする場合。
- **GUIでの表示など、個々の要素へのアクセスが必要な場合**。
- **`XMVECTOR`での計算結果の格納**: `XMVECTOR`で計算した結果をメモリに保存する際、`XMStoreFloat3()`や`XMStoreFloat4()`関数を使用して`XMFLOAT`型に変換します。

**例:**

C++

```
#include <DirectXMath.h>

// 構造体のメンバとして位置を保持
struct GameObject {
    XMFLOAT3 position;
    XMFLOAT4 color;
    // ...
};

GameObject player;
player.position = XMFLOAT3(0.0f, 1.0f, 0.0f);
player.color = XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f);

// XMVECTORの計算結果をXMFLOAT3に格納
XMVECTOR vec = XMVectorSet(10.0f, 20.0f, 30.0f, 0.0f);
XMFLOAT3 resultFloat3;
XMStoreFloat3(&resultFloat3, vec); // ここで変換
```

## 変換について

`XMVECTOR`と`XMFLOAT`の間は、以下の関数を使って相互に変換できます。

- **`XMFLOAT`→`XMVECTOR`:**
  - `XMLoadFloat2()`
  - `XMLoadFloat3()`
  - `XMLoadFloat4()`
  - `XMLoadFloat3x3()` (行列用)
  - `XMLoadFloat4x4()` (行列用) これらの関数は、`XMFLOAT`型のデータを`XMVECTOR`型（または`XMMATRIX`型）にロードし、SIMD演算の準備をします。
- **`XMVECTOR`→`XMFLOAT`:**
  - `XMStoreFloat2()`
  - `XMStoreFloat3()`
  - `XMStoreFloat4()`
  - `XMStoreFloat3x3()`
  - `XMStoreFloat4x4()` これらの関数は、`XMVECTOR`型の計算結果を`XMFLOAT`型のメモリ領域にストアします。

## まとめと推奨されるフロー

DirectXプログラミングにおける理想的なデータフローは以下のようになります。

1. **データ格納:** オブジェクトの状態（位置、回転、色など）は、クラスや構造体のメンバとして**`XMFLOAT`**型で保持します。
2. **計算開始:** 計算が必要になったら、**`XMLoadFloat`**関数を使用して**`XMFLOAT`**型のデータを**`XMVECTOR`**型にロードします。
3. **計算実行:** **`XMVECTOR`**型を使って、DirectXMathの関数（`XMVectorAdd`, `XMMatrixMultiply`など）で高速なSIMD演算を行います。
4. **計算結果の格納:** 計算が終わったら、**`XMStoreFloat`**関数を使用して**`XMVECTOR`**型の結果を再び**`XMFLOAT`**型にストアし、永続化します。

# ダイアログ

## ☒ 1. ダイアログってなに？

**ダイアログ（ダイアログボックス）は、**
**ユーザーに何かを聞いたり、設定を入力させたりするための小さなウィンドウ**です。

### 例：

- ファイルを保存するときの「名前を付けて保存」画面
- 「OK」「キャンセル」などを選ばせる小さなウィンドウ
- 設定画面（オプション、プロパティ）

---

## ☒ 2. モーダル vs モードレス

| 種類 | 特徴 | 例 |
|---|---|---|
| **モーダル** | 他のウィンドウを操作できなくなる（完了するまで） | 「保存しますか？」などの確認ダイアログ |
| **モードレス** | 他のウィンドウも同時に操作できる | 検索ウィンドウ、ツールパレット |

### ☒ イメージ

- モーダル：**呼び出した人に返事を返すまで、他は触らせない！**
- モードレス：**聞くけど、待たなくていいよ。他のことしててOK！**

---

## ☒ 3. よく使われる「コントロール（部品）」

### ☒ ボタン類

| 名前 | 説明 |
|---|---|
| **Button（ボタン）** | 「OK」「キャンセル」など。押すと動作を実行。 |
| **CheckBox（チェックボックス）** | ON/OFF の選択に使う。複数選択も可能。 |
| **RadioButton（ラジオボタン）** | 選択肢を1つだけ選ぶ（複数ある中で1つ） |

### ☒ 入力系

| 名前 | 説明 |
|---|---|
| **Edit（エディット）** | 文字を入力するテキストボックス。 |
| **ComboBox（コンボボックス）** | リストから選ぶ＋入力もできる。 |
| **ListBox（リストボックス）** | 複数行の選択肢から選ぶ。複数選択可もあり。 |

### ☒ 表示系・他

| 名前 | 説明 |
|---|---|
| **Static（スタティック）** | 文字や画像を表示するだけのラベル。 |
| **GroupBox** | コントロールをグループ化して囲む枠。 |
| **ProgressBar（プログレスバー）** | 処理の進行度を表示するバー。 |

---

## ☒☒ 4. 実際に使うとどうなる？

### モーダルダイアログを表示するコード例（Win32 API）

```
DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), hWnd, DialogProc);
```

### モードレスダイアログを表示するコード例

```
CreateDialog(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), hWnd, DialogProc);
ShowWindow(hDlg, SW_SHOW);
```

---

## ☒ 5. まとめ

| 項目 | ポイント |
|---|---|
| ダイアログ | ユーザーに質問や操作を促す小ウィンドウ |
| モーダル | 他の操作をブロックして返事を待つ |
| モードレス | 他の操作もOK、自由に使える |
| コントロール | ボタン、入力欄、リストなど部品のこと |

---

### ☒ もっとわかりやすく例えると…

> ダイアログは**コンビニのレジ**。
> モーダルは「レジの人が“お会計終わるまで商品戻してね”って止める」状態。
> モードレスは「セルフレジ。会計しながら他の人も動ける」状態です。