

システム全体像

`ALevelManager` (*Singleton Actor*)



主要な役割

`ALevelManager`

- 役割: ゲーム全体のマネージャークラスの統括・シングルトン管理
- 主な機能:
 - サウンドマネージャーの一元管理
 - UIマネージャーの一元管理
 - ポストプロセスエフェクトマネージャーの管理
 - シーン遷移(`LoadScene`)
 - シングルトンインスタンスの提供

`FManagerAccessKey` (アクセスキー)

- 役割: 特定のマネージャーへのアクセス制限
- 仕組み:
 - `private`コンストラクタ → 外部から生成不可
 - `friend class UPostProcessEffectHandle` → Handleクラスのみが生成可能
 - キーを持つクラスだけが`GetPostProcessEffectManager()`を呼び出せる

データの流れ

初期化フロー (`BeginPlay`)

1. `instance = this` (シングルトン登録)
2. `SoundManager`を生成
 - `Init()`で初期化
 - インターフェイス化して`mSoundManager`に設定
3. `UIManager`を生成
 - `Init()`で初期化
 - インターフェイス化して`mUIManager`に設定
4. `PostProcessEffectManager`は既にコンストラクタで生成済み

アクセスフロー

どこからでも:

```
ALevelManager::GetInstance(WorldContext)  
→ instance取得/生成  
→ 各マネージャーへアクセス
```

サウンド/UI:

```
GetSoundManager() → インターフェイス経由で自由にアクセス可能  
GetUIManager() → インターフェイス経由で自由にアクセス可能
```

ポストプロセス:

```
GetPostProcessEffectManager(Key) → キー必須  
→ UPostProcessEffectHandleのみがアクセス可能
```

設計上の特徴

1. シングルトンパターン

cpp

```
static TWeakObjectPtr<ALevelManager> instance;
```

- ゲーム内で唯一のインスタンスを保証
- `GetInstance()`で遅延初期化もサポート

2. インターフェイス分離

cpp

```
TScriptInterface<ISoundManagerProvider> mSoundManager;  
TScriptInterface<IUIManagerProvider> mUIManager;
```

- 利点: 実装の詳細を隠蔽、依存を最小化
- 用途: サウンド/UIは多くのクラスから使われるため、インターフェイス経由で公開

3. キーベースアクセス制御

cpp

```
UPostProcessEffectManager* GetPostProcessEffectManager(const FManagerAccessKey&  
Key)
```

- 目的: PostProcessEffectManagerへの直接アクセスを制限
- 理由: エフェクト管理は特定のハンドルクラス経由でのみ操作させたい
- 仕組み: `FManagerAccessKey`のコンストラクタがprivate →
`UPostProcessEffectHandle`のみが`friend`として生成可能

各マネージャーとの連携

マネージャー	アクセス方法	制限
SoundManager	<code>GetSoundManager()</code>	なし(インターフェイス)
UIManager	<code>GetUIManager()</code>	なし(インターフェイス)
PostProcessEffectManager	<code>GetPostProcessEffectManager(Key)</code>	キー必須

使用例

cpp

```
// どこからでもシングルトンを取得
ALevelManager* LevelMgr = ALevelManager::GetInstance(this);

// サウンド再生(自由にアクセス可)
LevelMgr->GetSoundManager()->PlaySound(ESoundKinds::BGM, "Default");

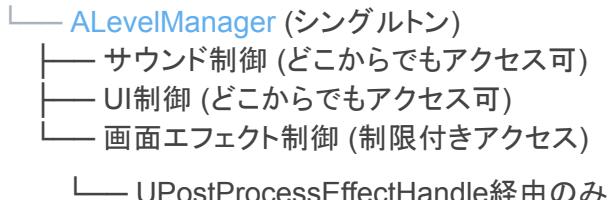
// UI表示(自由にアクセス可)
LevelMgr->GetUIManager()->ShowWidget(EUIType::MainMenu);

// ポストプロセス(キーが必要 - UPostProcessEffectHandleのみ可能)
// LevelMgr->GetPostProcessEffectManager(Key)->ActivateEffect(...);
...
```

システムアーキテクチャ上の位置付け

...

ゲーム全体



設計意図:

- 頻繁に使われる機能(音・UI)は簡単にアクセス可能
- 慎重に扱うべき機能(エフェクト)は専用ハンドルで制御