


1단계_05. 언리얼 오브젝트 리플렉션 시스템 #1

📅 수강일	@ 2023/03/22
➤ 이름	 전영재
🔍 멘토	Min-Kang Song, 현웅 최
👥 멘티	
⚙️ 작성 상태	In progress
📌 단계	
☑ 강의 시청 여부	<input checked="" type="checkbox"/>
☑ 이수 여부	<input type="checkbox"/>

Contents



- 언리얼 오브젝트 리플렉션 시스템 #1
 - [1] 언리얼 오브젝트의 리플렉션 시스템
 - 리플렉션 데이터 사용하기
 - [2] 언리얼 오브젝트의 구성
 - 클래스 기본 오브젝트 (CDO)
 - 언리얼 오브젝트의 처리
 - 자동 프로퍼티 초기화
 - 레퍼런스 자동 업데이트
 - Serialization
 - 프로퍼티 값 업데이트하기
 - 에디터 통합
 - 런타임 유형 정보 및 형변환
 - 가비지 컬렉션
 - 네트워크 리플리케이션
 - [3] 실습을 통해 위 기능을 다시 이해해보자.
 - 실습 개요
- 0n. 언리얼 오브젝트 리플렉션 시스템 #1 강의 과제

언리얼 오브젝트 리플렉션 시스템 #1



강의 목표

- 언리얼 오브젝트의 특징과 리플렉션 시스템의 이해
- 언리얼 오브젝트의 처리방식 이해

[1] 언리얼 오브젝트의 리플렉션 시스템

리플렉션 시스템은 프로그램이 실행 시간(런타임)에 자기 자신을 조사하는 기능이다.

이전 강의에서 언급한 대로, C++는 리플렉션을 지원하지 않으므로 언리얼에서 직접 구현하였으며, UObject의 큰 특징 중 하나이다.

UENUM(), UClass(), USTRUCT(), UFUNCTION(), UPROPERTY() 등의 매크로를 사용함으로써 언리얼 리플렉션 시스템에 등록되고, UHT가 이를 파싱하여 컴파일 전에 generated.h에 해당 소스코드를 작성해준다.

```
UPROPERTY(EditAnywhere, Category=Pawn) // ()에 들어가는 EditAnywhere, Category와 같은 정보들을 '메타데이터' 라고한다.
int32 ResourcesToGather;
//앞으론 UPROPERTY로 지정한 변수를 속성(Property)라 하겠다.
```

```
uint8 MyTeamNum;

UFUNCTION(BlueprintCallable, Category=Attachment)
void SetWeaponAttachment(class UStrategyAttachment* Weapon);
```

ResourcesToGaher 와 같이 `UPROPERTY()` 매크로를 지정해주면 리플렉션 시스템에 등록되어 에디터와 연동하여 사용이 가능해진다. 그러나 모든 멤버 변수가 `UPROPERTY()` 선언이 될 필요는 없다. `uint8 MyTeamNum`과 같이 선언하지 않고 사용해도 무방하다.

💡 단, `UPROPERTY()` 를 사용하지 않으면 리플렉션 시스템에 등록되지 않으므로, 메모리 관리를 직접 처리해야 한다. (예: 가비지 컬렉터가 자동으로 처리하지 못한다) ⇒ 그냥 설정하는게 더 편하겠는데?

리플렉션 데이터 사용하기

프로퍼티 시스템의 계층구조는 다음과 같다.

💡 UField → UStruct → UClass → UScriptStruct → UFunction → UEnum → UProperty → ...

UStruct부터 클래스의 리플렉션이 시작되고, UClass라는 클래스를 구성하는 구조체는 자손으로 함수나 프로퍼티를 포함하는 반면, UFunction과 UScriptStruct는 프로퍼티로만 제한된다.

리플렉션의 원리를 쉽게 설명하자면, **Unreal Build Tool (UBT)** 와 **Unreal Header Tool (UHT)** 이 함께 컴파일이 시작되기 전에 작동하여 해당 코드를 분석해 자동으로 시스템을 구축한다.

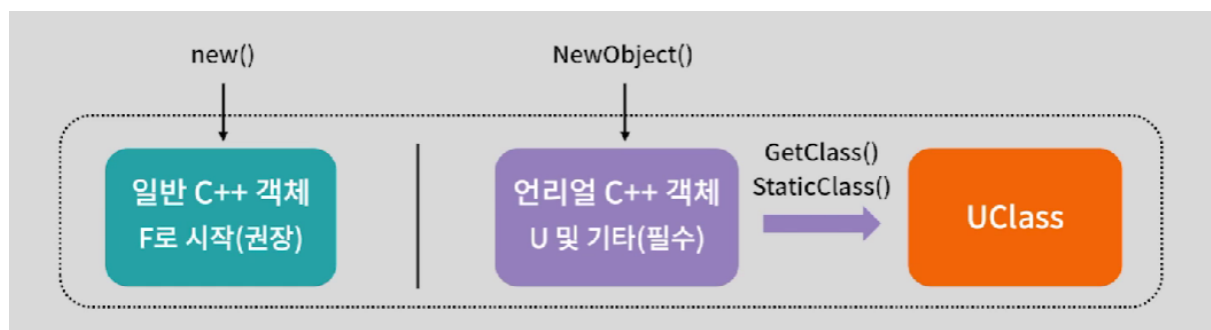
리플렉션된 클래스 정보를 보는 방법으로는 `StaticClass()`, `StaticStruct()` 와 같은 함수들이 존재한다. 이들 함수는 컴파일 단계에서 리플렉션 정보에 직접 접근하는 함수이다. 이러한 함수들은 UHT가 자동으로 생성한 `generated.h` 파일에 선언되어 있기에 곧바로 사용하면 된다.

[2] 언리얼 오브젝트의 구성

언리얼 오브젝트는 특별한 프로퍼티와 함수를 지정할 수 있다.

- `UPROPERTY()`
- `UFUNCTION()`

모든 언리얼 오브젝트는 '클래스 정보'를 포함하고 있고 해당 클래스 정보를 조회한다면 자신이 가진 프로퍼티와 함수 정보를 컴파일 타임과 런타임에서 조회할 수 있다. 이렇게 다양한 기능을 제공하는 언리얼 오브젝트는 일반적인 C++ 생성자인 `new` 키워드가 아니라 `NewObject()` 라는 생성자를 사용해야 한다.

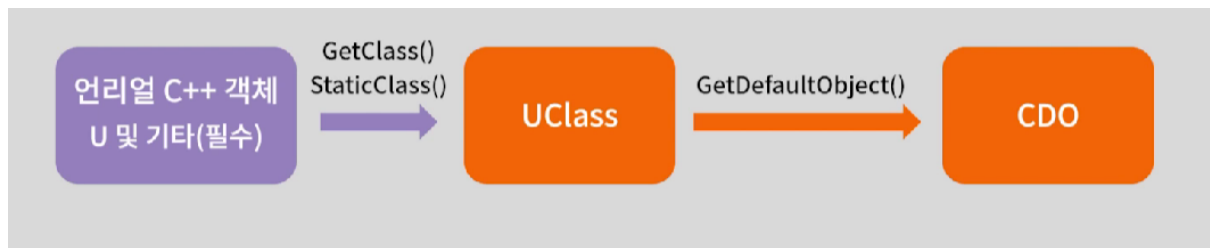


클래스 기본 오브젝트 (CDO)

UObject를 사용할 때는 **Class Default Object (CDO)** 라는 것을 알아야 한다.

CDO 엔리얼 객체의 기본값(Default)을 보관하는 템플릿 객체인데, 한 클래스에서 생성된 다수의 인스턴스의 기본값을 일관성 있게 조정하는데 사용할 수 있다.

해당 클래스의 디폴트 정보는 UClass의 `GetDefaultObject()` 를 통해서 얻어 올 수 있다.



엔리얼 오브젝트의 처리



앞선 '엔리얼 오브젝트 소개' 강의에서 알아본 내용으로, 리플렉션 시스템에 등록함으로써 얻을 수 있는 엔리얼 오브젝트의 기능을 설명한다.

자동 프로퍼티 초기화

C++ 프로퍼티의 경우, 생성자 혹은 `Initialize()` 를 통해 초기값을 설정해 주지 않으면 가비지가 생기지만 `UPROPERTY()` 로 선언된 프로퍼티의 경우 기본값을 0으로 자동으로 채운다.

레퍼런스 자동 업데이트

엔리얼 리플렉션 시스템에 프로퍼티를 등록함으로써 앞선 설명처럼 자동으로 레퍼런스 관련 업데이트가 진행된다.

Serialization

`UPROPERTY()` 를 지정함으로써 UObject 객체에 대한 정보를 지정된 포맷에 맞춰 디스크에 저장하고 일괄적으로 불러들일 수 있다.

프로퍼티 값 업데이트하기

CDO를 이용해, 여러 인스턴스에 대해 기본값을 일괄적으로 바꿀 수 있다.

에디터 통합

매크로 내의 메타데이터를 사용해 에디터에서 해당 프로퍼티나 함수를 인식하고 사용할 수 있다.

런타임 유형 정보 및 형변환

리플렉션 시스템에 등록함으로써 런타임 중에도 해당 데이터에 대한 접근 혹은 형변환을 안전하게 진행할 수 있다.

가비지 컬렉션

더이상 사용되지 않는(참조하지 않는) 데이터를 자동으로 감지하여 소멸시키는 기능이다.

네트워크 리플리케이션

Serialization이 등록된 엔리얼 프로퍼티를 자동으로 디스크에 저장하고 읽을 수 있는 것처럼, `UPROPERTY()` 를 통해 시스템에 등록함으로써 네트워크 통신에서도 자동으로 해당 프로퍼티를 전송하고 받을 수 있다.

[3] 실습을 통해 위 기능을 다시 이해해보자.

실습 개요

어느 학교에서 교수와 학생의 수업 시간

(학교 = GamelInstance / 교수,학생 = 인물 Class)

```
UMyGameInstance::UMyGameInstance()
{
    SchoolName = TEXT("기분학교");
}
```

```
void UMyGameInstance::Init()
{
    Super::Init();

    UE_LOG(LogTemp, Warning, TEXT("====="));

    UClass* ClassRuntime = GetClass(); //런타임에서 해당 클래스의 정보를 얻을 수 있다.
    UClass* ClassCompile = UMyGameInstance::StaticClass(); //컴파일에서 클래스의 정보를 얻을 수 있다.

    check(ClassRuntime == ClassCompile); // Assertion
    ensure(ClassRuntime == ClassCompile);
    ensureMsgf(ClassRuntime != ClassCompile, TEXT("ClassRuntime과 ClassCompile이 같으면 나을 ensure문구!"));

    UE_LOG(LogTemp, Warning, TEXT("학교를 담당하는 클래스 이름 : %s"), *ClassRuntime->GetName());
    SchoolName = TEXT("연신 학교");
    UE_LOG(LogTemp, Warning, TEXT("학교이름 : %s"), *SchoolName);
    UE_LOG(LogTemp, Warning, TEXT("학교이름 기본값 : %s"), *GetClass()->GetDefaultObject<UMyGameInstance>()->SchoolName); // 디폴트오브젝트를 U

    UE_LOG(LogTemp, Warning, TEXT("====="));
}
```

GetClass() ⇒ 런타임 도중의 객체에 대한 Class 정보를 얻는다.

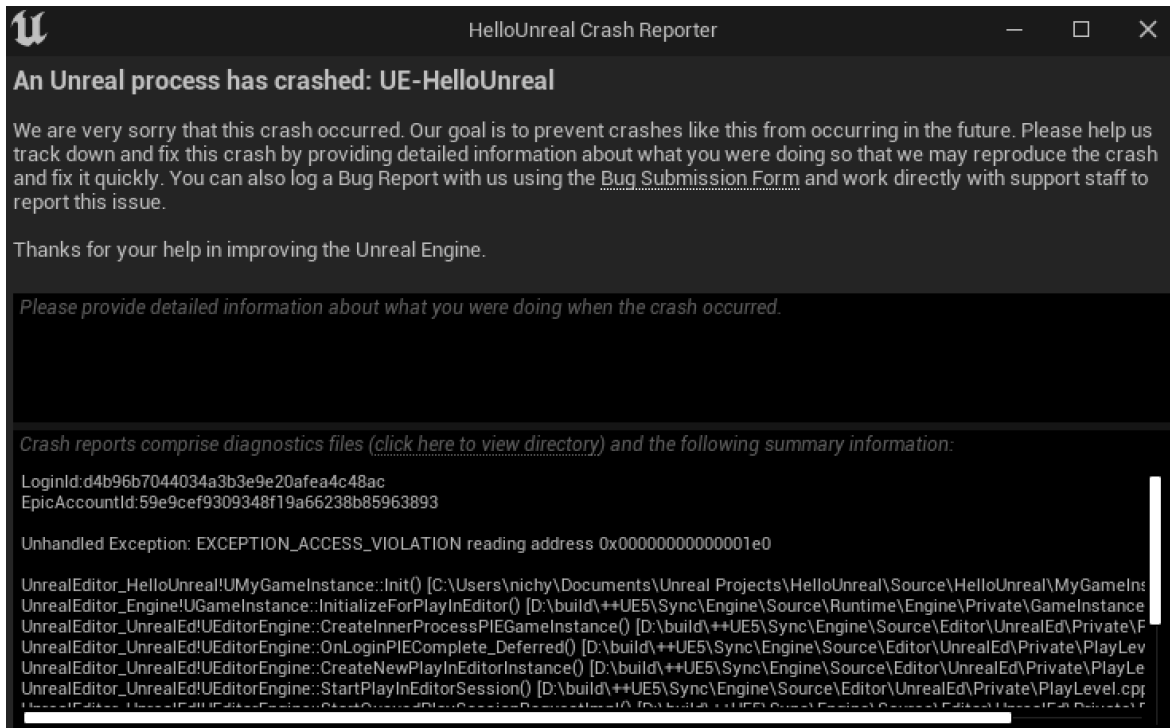
StaticClass() ⇒ 컴파일 타임의 해당 객체의 Class 정보를 얻는다.

▼ 여기는 쓸모 있을듯 없을듯한 'UCLASS - CDO 관계'에 대한 궁금증

그렇다면 컴파일 타임의 Class정보를 담은 ClassCompile의 SchoolName을 확인 한다면 기본값이 나올까 변경된 값이 나올까? 확인을 위해 다음과 같은 코드를 추가해 보았다.

```
UMyGameInstance* CastToGameInstance = Cast<UMyGameInstance>(ClassCompile);
UE_LOG(LogTemp, Warning, TEXT("ClassCompile의 SchoolName : %s"), *CastToGameInstance->SchoolName);
```

그 결과는 다음과 같다.



크래쉬가 났다..

클래스에 대한 정보를 담은 ClassCompile을 객체 클래스인 UMyGameInstance 로 형변환 한다는 것이 잘못된 것 아닐까하는 의구심이 들었다.

그렇다면 또 한 가지 궁금증이 생겼다.

```
*GetClass()->GetDefaultObject<UMyGameInstance>()->SchoolName
```

해당 코드는 분명 클래스 정보를 통해 해당 객체를 찾는 것일텐데, 내부 구조가 어떻게든 단순 Cast 형변환은 크래시가 나고 GetDefaultObject는 성공적으로 작동되는 것일까?

```
UObject* UClass::CreateDefaultObject()
{
    if ( ClassDefaultObject == NULL )
    {
        ensureMsgf(!bLayoutChanging, TEXT("Class named %s creating its CDO while changing its layout"), *GetName());

        UClass* ParentClass = GetSuperClass();
        UObject* ParentDefaultObject = NULL;
        if ( ParentClass != NULL )
        {
            UObjectForceRegistration(ParentClass);
            ParentDefaultObject = ParentClass->GetDefaultObject(); // Force the default object to be constructed if it isn't already
            check(GConfig);
            if (GEventDrivenLoaderEnabled && EVENT_DRIVEN_ASYNC_LOAD_ACTIVE_AT_RUNTIME)
            {
                check(ParentDefaultObject && !ParentDefaultObject->HasAnyFlags(RF_NeedLoad));
            }
        }

        if ( (ParentDefaultObject != NULL) || (this == UObject::StaticClass()) )
        {
            // If this is a class that can be regenerated, it is potentially not completely loaded. Preload and Link here to ensure we
            if( HasAnyClassFlags(CLASS_CompiledFromBlueprint) && (PropertyLink == NULL) && !GIsDuplicatingClassForReinstancing)
            {
                auto ClassLinker = GetLinker();
                if (ClassLinker)
                {
                    if (!GEventDrivenLoaderEnabled)
                    {
                        UField* FieldIt = Children;
                        while (FieldIt && (FieldIt->GetOuter() == this))
                        {
                            // If we've had cyclic dependencies between classes here, we might need to preload to ensure that we load the rest of
                            if (FieldIt->HasAnyFlags(RF_NeedLoad))
                            {
                                {
                                    ClassLinker->Preload(FieldIt);
                                }
                                FieldIt = FieldIt->Next;
                            }
                        }
                    }

                    StaticLink(true);
                }
            }

            // in the case of cyclic dependencies, the above Preload() calls could end up
            // invoking this method themselves... that means that once we're done with
            // all the Preload() calls we have to make sure ClassDefaultObject is still
            // NULL (so we don't invalidate one that has already been setup)
            if (ClassDefaultObject == NULL)
            {
                // RF_ArchetypeObject flag is often redundant to RF_ClassDefaultObject, but we need to tag
                // the CDO as RF_ArchetypeObject in order to propagate that flag to any default sub objects.
                ClassDefaultObject = StaticAllocateObject(this, GetOuter(), NAME_None, EObjectFlags(RF_Public|RF_ClassDefaultObject|RF_ArchetypeObject));
                check(ClassDefaultObject);
                // Register the offsets of any sparse delegates this class introduces with the sparse delegate storage
                for (TFieldIterator<FMulticastSparseDelegateProperty> SparseDelegateIt(this, EFieldIteratorFlags::ExcludeSuper, EFieldIteratorFlags::ExcludeNative); SparseDelegateIt.IsValid(); ++SparseDelegateIt)
                {
                    const FSparseDelegate& SparseDelegate = SparseDelegateIt->GetProperty->InContainer(ClassDefaultObject);
                    USparseDelegateFunction* SparseDelegateFunction = CastChecked<USparseDelegateFunction>(SparseDelegateIt->SignatureFunction);
                    FSparseDelegateStorage::RegisterDelegateOffset(ClassDefaultObject, SparseDelegateFunction->DelegateName, (size_t)&SparseDelegate);
                }
                EObjectInitializerOptions InitOptions = EObjectInitializerOptions::None;
                if (!HasAnyClassFlags(CLASS_Native | CLASS_Intrinsic))
                {
                    // Blueprint CDOs have their properties always initialized.
                    InitOptions |= EObjectInitializerOptions::InitializeProperties;
                }
                (*ClassConstructor)(FObjectInitializer(ClassDefaultObject, ParentDefaultObject, InitOptions));
                if (GetOutermost()->HasAnyPackageFlags(PKG_CompiledIn) && !GetOutermost()->HasAnyPackageFlags(PKG_RuntimeGenerated))
                {
                    // ...
                }
            }
        }
    }
}
```

```

    TCHAR  PackageName[FName::StringBufferSize];
    TCHAR  CDOName[FName::StringBufferSize];
    GetOutermost()->GetFName().ToString(PackageName);
    GetDefaultObjectName().ToString(CDOName);
    NotifyRegistrationEvent(PackageName, CDOName, ENotifyRegistrationType::NRT_ClassCDO, ENotifyRegistrationPhase::NRP_Finis
    }
    ClassDefaultObject->PostCDOConstruct();
    }
}
return ClassDefaultObject;
}

```

나는 위 코드에서 다음 부분에 집중했다.

```

UClass* ParentClass = GetSuperClass();
UObject* ParentDefaultObject = NULL;
if ( ParentClass != NULL )
{
    UObjectForceRegistration(ParentClass);
    ParentDefaultObject = ParentClass->GetDefaultObject(); // Force the default object to be constructed if it isn't already
    check(GConfig);
    ...
}

```

자세히 분석하지 못했지만, 대략적으로 `GetDefaultObject()` 함수는 해당 오브젝트의 부모클래스를 찾음과 동시에 그의 DefaultObject를 담을 수 있는 빈 `UObject* ParentDefaultObject` 생성한다. 이것을 재귀적으로 반복하여 적합한 Class (`<UMyGameInstance>`)를 찾고 해당 오브젝트를 ParentDefaultObject에 담아 반환 하는 것이다.



즉, 내부적으로 여러 과정을 거쳐서 조상을 찾아가고, 맞는 조상을 찾으면 그 조상의 오브젝트를 반환하는 것이다. 그렇기에 당연히 Class자체를 Object로 형변환하는 `Cast()` 와 내부적으로 생성한 Object를 반환하는 `GetDefaultObject()` 는 차이가 날 수 밖에 없다!



Assertion(역설)Error : 개발자가 반드시 참이어야 한다고 생각하는 조건문

- `check()` : true를 반환하지 않는다면 크래시가 난다.
- `ensure()` : 크래시가 나지 않고 OutputLog에 Error 로그가 남는다
- `ensureMsgf()` : `ensure()` 와 비슷하지만, 원하는 로그메시지를 남길 수 있다.



Summary

- 언리얼 오브젝트는 항상 클래스 정보를 담은 UClass 객체와 매칭되어 있다.
- UClass는 CDO와 연결되어, 컴파일 단계 이전에 클래스의 정보를 저장한다.
- 이렇게 엔진 초기화 과정에서 생성된 데이터들은 런타임 중에도 변하지 않고 확인할 수 있기에 안전하게 사용 가능하다.

0n. 언리얼 오브젝트 리플렉션 시스템 #1 강의 과제



Q1. 언리얼 오브젝트를 선언하고 구현할 때, 언리얼 헤더들로부터 지켜야하는 코딩 규칙을 정리해보세요.



Q2. NewObject로 생성된 언리얼 오브젝트의 인스턴스와 클래스 정보 UClass, CDO의 관계를 그림으로 정리해보세요. 각 오브젝트의 생성 순서도 유추해보면서 정리하면 좋습니다.

? Q3. 컴파일 타임과 런타임에 대해 잘 모르고 있다면 추가로 정리하세요. (선택)

 Reference