# UE4中组件的概念与基本继承关系

以A开头的都是可以放置在场景中的，以U开头的组件只能依附于其他组件存在，不能单独放置在场景中。

## 组件

USceneComponent 场景组件（变换、位置等信息皆在该组件内，无显示信息， 一般为根组件）

UStaticMeshComponent 静态网格体组件

UParticleSystemComponent 粒子组件

UAudioComponent 声音组件

UBoxComponent 盒体组件

UCapsuleComponent 胶囊体组件

USphereComponent 球体组件

# 引擎内建类

## 模式相关

（1）编辑器模式：启动编辑器后的游戏状态。

（2）Runtime模式：游戏打包后的运行状态。

（3）PIE模式：（Play in editor）在编辑器中点击Play后运行状态。

## UEngine\* GEngine

代表引擎，数量为1； Editor或Runtime模式都是全局唯一，从进程启动开始创建，进程退出时销毁。

## UEditorEngine\* GEditor

代表编辑器，数量1；只在编辑器下存在且全局唯一，从编辑器启动开始创建，到编辑器退出时销毁。

## UGameInstance\* GameInstance

代表一场游戏，数量1。 从游戏的启动开始创建，游戏退出时销毁。这里的一场游戏指的是Runtime或PIE模式的运行的都算，一场游戏里可能会创建多个World切换。

## UWorld\* World

代表一个世界，数量可能>1。World和GameMode是关联的，可以包含多个Level，默认情况下OpenLevel常常会切换World。因此其生命周期，其实跟GameMode是一起的。编辑器模式下视口里的场景其实也是个World，因此EWorldType其实有多个类型：Game，Editor，PIE，EditorPreview，GamePreview等。

## ULocalPlayer\* LocalPlayer

代表本地玩家，数量可能>1。 UE支持本地分屏多玩家类型的游戏，但往往最常见的是就只有一个。LocalPlayer虽然往往跟PlayerController一起访问，但是其生命周期其实是跟UGameInstance一起的(默认一开始的时候就创建好一定数量的本地玩家)，或者更准确的说是跟LocalPlayer的具体数量挂钩（当然你也可以运行时动态调用AddLocalPlayer)。

# 代码声明相关

[UE4 c++编程，最近常用的头文件\_实习生小黄的博客-CSDN博客](https://blog.csdn.net/J_Wayne/article/details/105026889)

## [UCLASS（）](https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Objects/)

UCLASS（）宏可使虚幻引擎识别新的类；使用 C++ 类向导创建的类声明自动通过 UCLASS() 宏进行处理。 UCLASS() 宏使得引擎意识到这个类的存在，并且还可以同键盘修饰符结合使用来在引擎中设置该类的行为。

UCLASS 包含定义类的一套属性和函数。这些是本地代码可用的普通 C++ 函数和变量，但被虚幻引擎特有的元数据所标记，它们在对象系统中的行为也因此受到控制。

## [GENERATED\_BODY（）](https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Objects/)

GENERATED\_BODY宏不获取参数，但会对类进行设置，以支持引擎要求的基础结构。所有UCLASS均有此要求。

## [UPROPERTY（）](https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/GameplayArchitecture/Properties/)

属性声明，通过UPROPERTY宏来定义属性元数据和变量说明符，说明函数、结构体等与引擎和编辑器各方面的相处方式；以UPROPERTY（）宏作为声明序列的变量可被引擎执行垃圾回收，也可在虚幻编辑器中显示和编辑。

UPROPERTY([specifier, specifier, ...], [meta(key=value, key=value, ...)])

Type VariableName;

（类声明包含很多变量和函数声明。变量与函数可以通过UPROPERTY和 [UFUNCTION](https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/ProgrammingWithCPP/IntroductionToCPP/)宏分别进行处理，这些宏的功能和UCLASS() 宏类似。组件也可与UPROPERTY宏一起设置）

## PrimaryActorTick.bCanEverTick

在构造函数中出现，声明是否每帧都调用Tick()函数，若是则将其设置为True。

## SetupAttachment()

A->SetupAttachment(B)：将A附加到组件B上。

# [资源引用](https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/GameplayArchitecture/Classes/)

理想状态下，类中的资源引用并不存在。硬编码资源引用很脆弱，优选方法是使用蓝图配置资源属性。然而，仍然完全支持硬编码引用。不需要在每次构造对象时搜索资源，因此这些搜索只执行一次。一个静态结构体可确保只执行一次资源搜索。

引用方式：

ConstructorHelpers::FObjectFinder<T> namestring(TEXT(“资源文件路径”)

若未找到资源，会报告失败。

# 位置、缩放相关API

## 获取Actor位置

GetActorLocation()：获取组件位置，返回类型为Fvector，分量为x、y、z。

GetActorRotation()：获取组件角度，返回类型为FRotator，分量为Pitch、yaw、roll，分别表示绕y、z、x轴旋转。

## 获取游戏运行时间

GetGameTimeSinceCreation ()：返回类型为float。

## 设置Actor位置

SetActorLocationAndRotation(Fvector, FRotator)

## 获取组件缩放比例

GetComponentScale()：

返回FVector，可通过加“.X”、“.Y”、“.Z”获取各个方向比例。

SetRelativeScale3D()

设置组件相对于其父组件的非均匀比例

## 设置缩放比例

SetWorldScale3D()：

输入FVector，设置三个方向的缩放比例

## 获取Actor向前的方向

Rotator1.Vector()

根据物体的旋转信息得到其向前的方向向量（FVector），Rotator类型数据

# 控制器相关

## APlayerController 控制器类

## 获取控制器

UGameplayStatics::GetPlayerController(const UObject\* WorldContextObject, int32 Playerindex)：

第一个参数：世界中物体的引用

第二个参数：玩家控制器的索引，0表示本地玩家

## 获取控制器对应相机

GetViewTarget()

## 设置摄像机

SetViewTarget()

SetViewTargetWithBlend ()：设置摄像机平滑过渡

## [控制器输入](https://docs.unrealengine.com/4.26/zh-CN/ProgrammingAndScripting/ProgrammingWithCPP/CPPTutorials/PlayerInput/)

（1）生成的Pawn类中带有“SetupPlayerInputComponent()”函数，在函数体中可通过“InputComponent->BindAction()”函数绑定操作映射，函数参数列表中可输入动作名称、函数指针等内容；同理，通过“InputComponent->BindAxis()”函数绑定轴映射。

（2）可通过“AutoPossessPlayer = EAutoReceiveInput::Player0”将pawn设置为由最小编号的玩家控制器控制（一般最小编号指本地玩家控制器）。

## [角色移动](https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/ProgrammingWithCPP/CPPTutorials/FirstPersonShooter/2/)（character）

（1）获取角色X方向向量（返回类型为FVector）：

FRotationMatrix(Controller->GetControlRotation()).GetScaledAxis(EAxis::X)

（2）在某一方向移动：

AddMovementInput(方向向量,value)

第一个系数为移动方向，第二个为系数

（3）对角色添加视角转动

AddControllerYawInput函数，Pawn类的内置函数，视角转动；

UE4使用左手坐标系，roll、pitch、yaw分别表示绕X、Y、Z旋转。

（4）角色跳跃

Acharacter中内置了跳跃相关的布尔变量“bPressedJump”，按下跳跃键为True，松开为False；通过改变其值可实现跳跃。

## SetOnlyOwnerSee(布尔型数值)

A->SetOnlyOwnerSee(true)：设置A网格体仅对有此角色的PlayerController可见。

# [Actor](https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/ProgrammingWithCPP/IntroductionToCPP/)

[Actors | 虚幻引擎文档 (unrealengine.com)](https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Actors/)

## [生成Actor](https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Actors/Spawning/)

SpawnActor()：

UWorld的成员之一，成功生成Actor后，引擎调用其BeginPlay()

**函数参数：**

AActor\* UWorld::SpawnActor

(

UClass\* Class,

FName InName,

FVector **const**\* Location,

FRotator **const**\* Rotation,

AActor\* Template,

**bool** bNoCollisionFail,

**bool** bRemoteOwned,

AActor\* Owner,

APawn\* Instigator,

**bool** bNoFail,

ULevel\* OverrideLevel,

**bool** bDeferConstruction

)

## 销毁Actor

（1）通过Destroy()销毁Actor，在该过程中将调用EndPlay()，能在Actor进入回收站之前执行自定义逻辑。

（2）另一个控制Actor生命周期时长的方法是使用 Lifespan 成员。可以在对象的构造函数中设置Actor的时间跨度，也可以在运行时使用其他代码进行设置。当这段时间到期后，会自动对该Actor调用 Destroy。

**设置生命周期：InitialLifeSpan=3.0f;**

## [获取Actor视角](https://docs.unrealengine.com/5.0/en-US/API/Runtime/Engine/GameFramework/AActor/GetActorEyesViewPoint/https:/docs.unrealengine.com/5.0/en-US/API/Runtime/Engine/GameFramework/AActor/GetActorEyesViewPoint/)

AActorLLGetActorEyesViewPoint(Location, Rotation)

# 坐标相关

## 坐标变换

将坐标从物体空间坐标系转换至世界空间坐标系

FTransform(Rotation1).TransformVector(FVector1)

其中，Rotation1为物体的旋转变换，FVector1为物体空间坐标系下的坐标，返回类型为FVector。

# [设置材质](https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/ProgrammingWithCPP/CPPTutorials/FirstPersonShooter/3/)

## 动态材质实例：UMaterialInstanceDynamic（是一种类型）

此材质是动态的，可发生改变

## 材质类型：UMaterial

## 设置材质

//获取材质资源

**static** ConstructorHelpers::FObjectFinder<UMaterial> Material(TEXT(""));

**if** (Material.Succeeded())

{

//为声明的动态材质实例赋值

MaterialInstance = UMaterialInstanceDynamic::Create(Material.Object, MeshComp onent);

}

MeshComponent->**SetMaterial**(0, MaterialInstance);