**基本概念**

1. **项目：**

保存着构成游戏所需的所有内容和代码。

1. **蓝图：**

一种功能齐全的游戏脚本系统，可在虚幻编辑器中通过基于节点的界面来创建游戏元素。

个人理解：将程序中的类、对象、逻辑用类似流程图的方法实现。

1. **对象：**

在虚幻引擎中，最基本的类叫做Object，是最基本的构建单位，包含了资产的基本功能。虚幻引擎中大多数类都继承自Object。

C++中，UObject是所有Object的基类，包含垃圾回收、通过元数据将变量公开给编辑器，以及保存和加载时的序列化功能。

1. **Actor:**

所有可以放入关卡的对象都是 Actor，比如摄像机、静态网格体、玩家起始位置。Actor支持三维变换，例如平移、旋转和缩放。可以通过游戏逻辑代码（C++或蓝图）创建或销毁Actor。

在C++中，AActor是所有Actor的基类。

1.Actor的生成：

通过UWorld::SpawnActor()函数生成，函数返回Actor对应的指针。

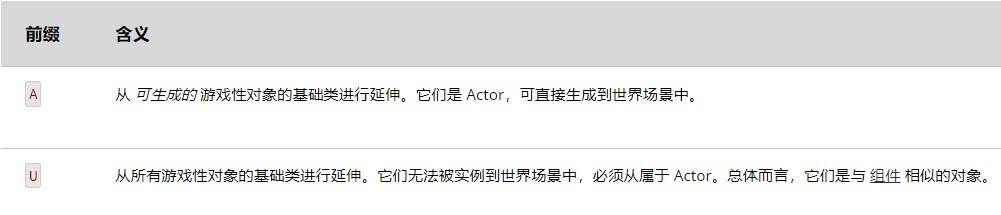
（[生成 Actors | 虚幻引擎文档 (unrealengine.com)](https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Actors/Spawning/)）

1. **类：**

用于定义虚幻引擎中Actor或对象的行为和属性。类可以被继承，这意味着某个类可以从其父类（衍生或派生出该类的类）获得信息，然后再将信息传递给子类。类可用C++代码或蓝图创建。

虚幻中的类拥有一个标准化的命名方案，通过首字母或前缀可立即明了其类别

**1. 游戏性类的前缀有：**



**2. 类头：**

虚幻引擎中的游戏性类通常拥有单独且唯一的类头文件。通常这些文件的命名与其中定义的类相匹配，减去A或U前缀，并使用.h文件扩展名。因此， AActor类的类头文件命名为Actor.h。虽然 Epic 代码遵循这些规则，但当前引擎中类名称和源文件名之间不存在正式关系。

在每个游戏性类头文件的顶端，需要包含生成的头文件（自动创建）。因此在 ClassName.h 的顶端必须出现以下行：

#include "ClassName.generated.h"

**3. 类声明形式：**

UCLASS（[specifier, specifier,…]，[meta(key=value, key=value,…)]）

Class Classname : public ParentName

{

GENERATED\_BODY()

}

解释：

（1）在标准声明之上，描述符（如类说明符和元数据）将传递到UCLASS宏。他们用于创建被声明类的UClass，它可被看作引擎对类的专有表达。

[相应的类说明符与元数据说明符见[游戏性类 | 虚幻引擎文档 (unrealengine.com)](https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/GameplayArchitecture/Classes/)]

（2）**GENERATED\_BODY()宏必须被放置在类体的最前方**。

所有的游戏性类必须使用GENERATED\_BODY宏进行正常实现。该执行在定义类和其所有变量和函数的类头（.h）文件中完成。最佳方法是使类源和头文件的命名与实现的类相匹配，减去A或U前缀。因此，AActor类的源文件命名为Actor.cpp，其头文件命名为Actor.h。对编辑器中"Add C++ Class"菜单选项创建的类而言，此操作将自动进行。

1. **组件**

组件是可以添加到Actor上的一项功能。当为Actor添加组件后，其便拥有了该组件需提供的功能。

（组件必须绑定在Actor上，他们无法单独存在）

1. **Pawn**

Pawn是Actor的子类，它可以充当游戏中的化身或人物（例如游戏中的角色）。Pawn可以由玩家控制，也可以由游戏AI控制并以非玩家角色（NPC）的形式存在于游戏中。

1. **角色**

角色（Character）是Pawn Actor的子类，代表垂直站立的玩家，旨在用作玩家角色。角色子类包括碰撞设置、双足运动的输入绑定，以及用于控制运动的附加代码。

1. **玩家控制器**

玩家控制器（Player Controller） 会获取游戏中玩家的输入信息，然后转换为交互效果，每个游戏中至少有一个玩家控制器。玩家控制器通常会控制一个Pawn或角色，将其作为玩家在游戏中的化身。

玩家控制器还是多人游戏中的主要网络交互节点。在多人游戏中，服务器会为游戏中的每个玩家生成一个玩家控制器实例，因为它必须对每个玩家进行网络函数调用。每个客户端只拥有与其玩家相对应的玩家控制器，并且只能使用其玩家控制器与服务器通信。

1. **AI控制器**

AI控制器通过控制Pawn来表示游戏中的非玩家角色（NPC）。默认情况下，Pawn和角色最终都会由基本的AI控制器控制，除非它们被指定通过玩家控制器控制，或被告知不需要为它们自己创建AI控制器。

1. **玩家状态**

玩家状态（Player State）表示某个游戏参与者的状态，可以是人类玩家，也可以是模拟玩家的机器人。作为游戏场景的一部分而存在的非玩家类AI不会有玩家状态。

1. **游戏模式**

游戏模式（GameMode）类负责设置当前游戏的规则。规则包括：

1.玩家如何加入游戏。

2.是否可以暂停游戏。

3.任何与游戏相关的行为，例如获胜条件。

1. **体积**

体积（Volumes）是一种存在边框的3D空间，会根据施加给它们的效果产生不同的用途。例如：

1.阻挡体积（Blocking Volumes），一种不可见的体积，防止Actor穿过它们。

2.伤害生成体积（Pain Causing Volumes），会对进入它们的Actor产生持续性的伤害。

3.触发体积（Trigger Volumes），可以通过编程，让Actor在进入或离开它们是触发事件。

参考网址：[虚幻引擎 4 术语 | 虚幻引擎文档 (unrealengine.com)](https://docs.unrealengine.com/4.27/zh-CN/Basics/UnrealEngineTerminology/)



**资产和包**

**一、资产**

1.资产是虚幻引擎项目中的一项内容，可将其看作序列化到文件中的UObject。

2.可在内容浏览器的源面板上点击“显示/隐藏源和包”按钮，显示“资产树”。

（1）资产树以列表形式显示内容目录中的文件内容；

（2）关卡文件格式 .umap；

（3）用户可在导入或创建资产时对其进行命名。资产路径将与它在硬盘上的位置直接关联。

（4）资产以.uasset文件格式存储，每个.uasset文件通常包含单个资产。每个资产引用都包含目录式的路径，每个路径可唯一标识游戏资产。

3.在内容浏览器中移动资产，则引用该资产的其他资产将进行相应更新，通过原资产文件所在位置保留的重定向器实现定位。可通过在内容浏览器中右击“整理文件夹中的重定向器”选项清除重定向器。