# 虚幻4高级程序-标记宏

UNREAL 4 HIGH INTEGRATION PROGRAM DEVELOPMENT ENGINEER

# 01 UObject

# UObject

UE4对于C++进行调整后，靠拢了其他虚拟机语言，为了方便对象内存管理和映射数据操作，UE4对于UECPP进行了更加严谨的语法定义，所有UE中使用的对象大部分继承自UObject，这样系统级的管理更加轻松，不必涉及更多不定性内容，UOB具备如下优点

- 垃圾回收
- 引用更新（更新受限于GC回收机制，需使用UPROPERTY宏标记）
- 序列化（场景中的Actor被保存时发生）
- 默认属性变化自动更新
- 源码调整了对象属性时，编译后会自动更新到资源实例上，前提场景实例资源没有修改过属性值，修改过后将使用场景中的修改内容做填充参考
- 自动属性初始化
- 自动编辑器整合
- 运行时类型信息可用
- 运行时使用Cast可以进行类型信息投射检查，
- 网络复制

# UCLASS宏

用于标记从**UObject派生的类**，使得UObject处理系统识别到它们。

该宏为UOB提供一个对UCLASS的引用，描述其是基于虚幻引擎的类型。对于标记UCLASS的类，引擎可以进行识别。

# 继承关系中的头文件描述

#include "MyObject.generated.h“

MyObject：是你的头文件名称

必须是最后一个被引入的头文件，默认由UE模版生成，此头文件需要了解其他类，所以需要放到最后

UCLASS()

UCLASS 宏使虚幻引擎 4 能识别 UMyObject

class MYPROJECT_API UMyObject : public UObject

如 MyProject 希望将 UMyObject 类公开到其他模块，则需要指定 MYPROJECT_API

GENERATED_BODY()

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，GENERATED_BODY 宏将成员访问等级设为“public”，而非设置“private”的缺省语言。

# 创建Uobject类对象

NewObject()是最为简单的UObject工厂模式。它需要可选的外部对象和类，并会创建拥有自动生成的名称的新实例。

```
template< class T >
T* NewObject
(
    UObject* Outer=(UObject*)GetTransientPackage(),
    UClass* Class=T::StaticClass()
)
```

| 参数 | 描述 |
| --- | --- |
| 外部 | 可选。UObject作为被创建的 **对象** 的外部参数。 |
| 类 | 可选。UClass定义待创建的 **对象** 类。 |

**返回值**

指向指定类生成实例的指针。

# 02 UCLASS宏标记

结构

UCLASS(描述指令，描述指令，...)

# 常用标记

**BlueprintType NotBlueprintType**

此类可以作为蓝图中的一种变量类型使用，Actor**类默认均可被当作蓝图变量**，我们常用此标记描述结构体，提供给蓝图作为参数类型使用。

**Blueprintable NotBlueprintable**

用来标明当前类是否可被蓝图继承。标记关系向子类传递，子类可覆盖描述关系，如果父类标记可被继承，子类具备相同的特性。

**Abstract**

Abstract 类修饰符将类声明为“抽象基类”，这样会阻止用户在虚幻编辑器中向这个世界中添加这个类的Actor，或者在游戏过程中创建这个类的实例。

**Const**

本类中的所有属性及函数均为常量，并应作为常量导出。该标识由子类继承。

**Config**

讲类内的成员变量数据信息保存到本地配置文件中，需要显式调用函数SaveConfig使用，并配合UPROPERTY宏操作

UCLASS(Config=Game)

**ClassGroup**

用来配置**组件**在添加时分组情况

UCLASS(ClassGroup = (ZQSELF))

# 03 UFUNCTION宏标记

结构

UFUNCTION(指令，指令..，meta(key=value))

注意：在UFUNCTION修饰的函数中，如果参数类型是引用型参数，则在蓝图中将当做返回参数使用，无法查到输入针脚。如果参数类型是const修饰的引用型参数，则参数被当做输入针脚使用

**当标记函数时，函数中如果带有参数，则参数名字必须明确指出，如果函数中的参数名字相同（不敏感大小写），则编译不过（序列化函数参数名称使用的是FName）**

# 常用标记

**BlueprintCallable(修饰函数访问域随意)**

表明此函数（如果是成员函数）可在蓝图中被调用

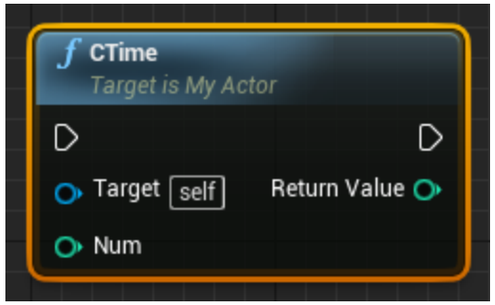
如果是**静态成员函数**，则在任意蓝图中均可调用

注意：如果函数参数是引用类型，则在蓝图中调用被当做输出针脚，如果传入参数是const修饰的引用类型，则在蓝图中被当做输入针脚

UFUNCTION(BlueprintCallable)

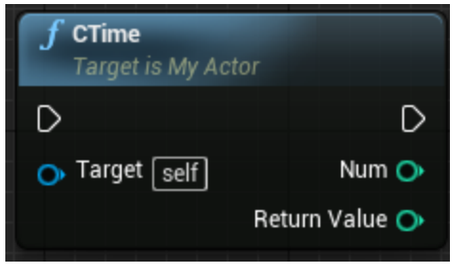
int32 CTime(const int32& num);

编译后




UFUNCTION(BlueprintCallable)

int32 CTime(int32& num);//被当做函数输出针脚使用



# 常用标记

**Category**

标明此函数在蓝图中的分类

用法 Category=”UE4Test|MyActor” // |符号用来划分分类级别

**BlueprintImplementableEvent(函数不能私有)**

标记的函数可以在c++中调用，也可以蓝图中调用（需要标记BlueprintCallable ）

用此标记可以在C++中构建函数声明，但是定义由蓝图完成，从而达到C++向蓝图进行调用的目的，在CPP无需定义

UFUNCTION(BlueprintImplementableEvent, Category = "UECPP|ACPPActory")

类似纯虚函数，但是继承关系中的蓝图并不用必须重写此函数

注意：

如果函数**没有返回类型**，则在蓝图中当作**事件Event**使用

如果函数**存在返回类型**，则在蓝图中当作**函数**使用（需要在函数的overlap中寻找）

如果带有参数

- 没有返回值，参数是基本数据类型，则当作普通事件输入参数使用
- 没有返回值，参数是基本数据类型引用，则当作函数看待，在函数表中寻找
- 没有返回值，参数是自定义数据类型引用，则当作普通函数看待，在函数表中寻找
- 没有返回值，参数是自定义数据对象类型FString（非引用标记），则编译不过（FString只能传递引用或是常引用）
- 没有返回值，参数是U类指针，则被当做事件看待
- 如果有返回值，则当作函数使用
- 带有返回值并带有参数，则参数规则参照BlueprintCall的规则

# 常用标记

**BlueprintNativeEvent**

标记的函数可以在c++中调用，也可以蓝图中调用（需要标记BlueprintCallable ）

此标记可以标注函数可在蓝图中被重写，并且具备在C++中有另一个标记函数

函数定义格式为：

返回类型 函数名_ Implementation(参数列表)

达到效果，如果蓝图重写此函数，则函数实现在蓝图，如果蓝图没有重写此函数，则函数实现在（函数名_ Implementation）上。

**如果希望调用函数时将父类的行为也调用执行，则可以在蓝图中右键函数节点，选择add call to parent function可以调用父类的函数逻辑（类似类中的虚函数，在继承关系中子类可以调用父类的虚函数）**

注意：

如果函数没有返回类型，则在蓝图中当作事件Event使用

如果函数存在返回类型，则在蓝图中当作函数使用（需要在函数的overlap中寻找）

基本效果同上

传递引用操作如BlueprintCallable

# 常用标记

**BlueprintPure**

特殊标记，构建一个蓝图中的纯函数，用来获取对象数据。所以此标记的函数必须有有效返回值（**无返回值编译报错**），并且在蓝图中无法加入到执行队列，只能以输出值的操作方式被使用，定义实现均放在cpp中

UFUNCTION(BlueprintCallable, BlueprintPure)

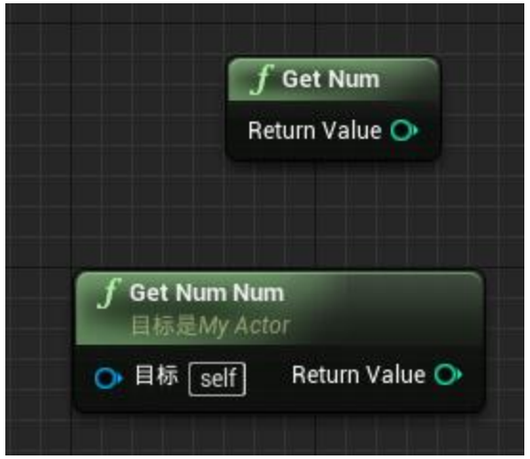
**注意：成员函数添加const关键字标记为常函数，被序列化后则失去执行引脚输入。（const关键字只能标记成员函数）**

语法

UFUNCTION(BlueprintCallable, BlueprintPure)
int32 GetNum();
UFUNCTION(BlueprintCallable)
int32 GetNumNum() const;

蓝图节点变化

注意：const修饰的函数，丢掉了执行白色引脚，但是依旧是成员函数，带有输入目标



# 04 UPROPERTY宏标记

# 语法结构

注意：用于将对象属性暴露到蓝图中操作，整形中只有int32能暴露到蓝图中。**切记，变量如果希望在蓝图中被操作，则应该放到公有域或是受保护域，不能放在私有域。**

语法

UPROPERTY（标记，标记，...，meta（key=value，key=value，...））

类型 参数名称

# 常用标记

**BlueprintReadOnly**

标记属性可以在蓝图中获取，但是只能读取无法进行操作

**BlueprintReadWrite**

在蓝图中即可读取也可以操作设置

**Category**

标明此变量在蓝图中的分类

用法 Category=”UE4Test|MyActor” // |符号用来划分分类级别

**Config**

标记此属性可被存储到指定的配置文件中，启动时属性内容将从配置文件中获取

UPROPERTY(Config)

**EditAnywhere**

表示此属性可以在编辑器窗口中进行编辑也可在场景细节面板中编辑

**EditDefaultsOnly**

可以在蓝图编辑器中编辑原型数据，但无法在场景细节面板中编辑场景中的具体对象

**EditInstanceOnly**

表明该属性的修改权限在实例，不能在蓝图编辑器原型中修改

# 常用标记

**VisibleAnywhere**

表明属性可以在属性窗口可见（原型实例中均可看到），但无法编辑。

注意如果标记组件指针，则表示组件内容在细节面板中显示所有编辑项。

**VisibleDefaultsOnly**

属性仅能在原型蓝图编辑器属性窗口中可见，无法编辑

注意如果标记组件指针，则表示组件内容在细节面板中显示所有编辑项。

**VisibleInstanceOnly**

属性仅能在实例属性窗口中可见，无法编辑

注意如果标记组件指针，则表示组件内容在细节面板中显示所有编辑项。

**EditFixedSize**

限定动态数组长度禁止在蓝图属性面板中修改（单一添加无法显式，需要配合使用上面两个标记）

**AdvancedDisplay**

将属性信息在细节面板中隐藏到高级显式项内容中

# meta描述

**别名标记指令**

meta=（DisplayName=”别名”）

使用此标记可以帮助我们对函数或是属性进行别名操作，可以使用在UF和UP中，可以在蓝图中或是细节面板中显示别名

**成员属性值域约束**

meta = (UIMin="10", UIMax = "20") UI上约束，如直接填入数据则不约束

meta = (ClampMin = "0", ClampMax = "10") UI上约束，填入数据也被约束

**成员属性修改约束**

EditCondition

可以借助一个布尔变量用来控制另一个变量是否可以在面板中被修改

```
UPROPERTY(EditAnywhere)
bool bShow;
UPROPERTY(EditAnywhere, AdvancedDisplay, meta = (ClampMin = "0", ClampMax = "10", EditCondition="bShow"))
int32 wock;
```

火星时代教育

# 感谢聆听

THANK YOU

帮助更多的人实现梦想