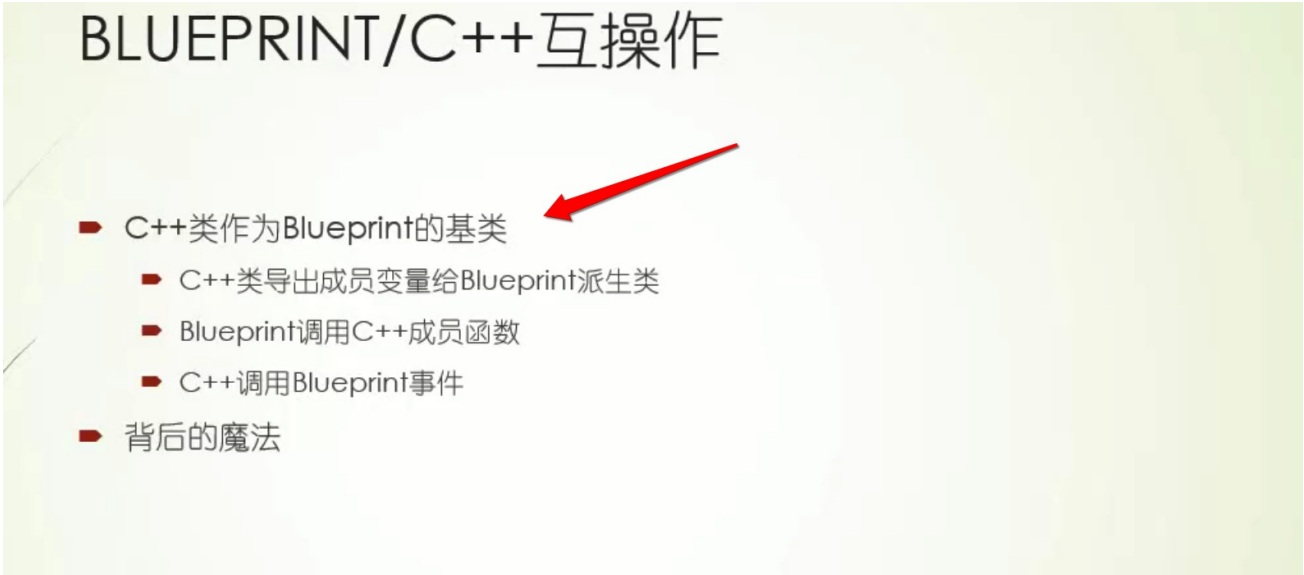
C++与蓝图的互操作

c++：项目框架 底层编写

蓝图：内容开发 快速搭建原型



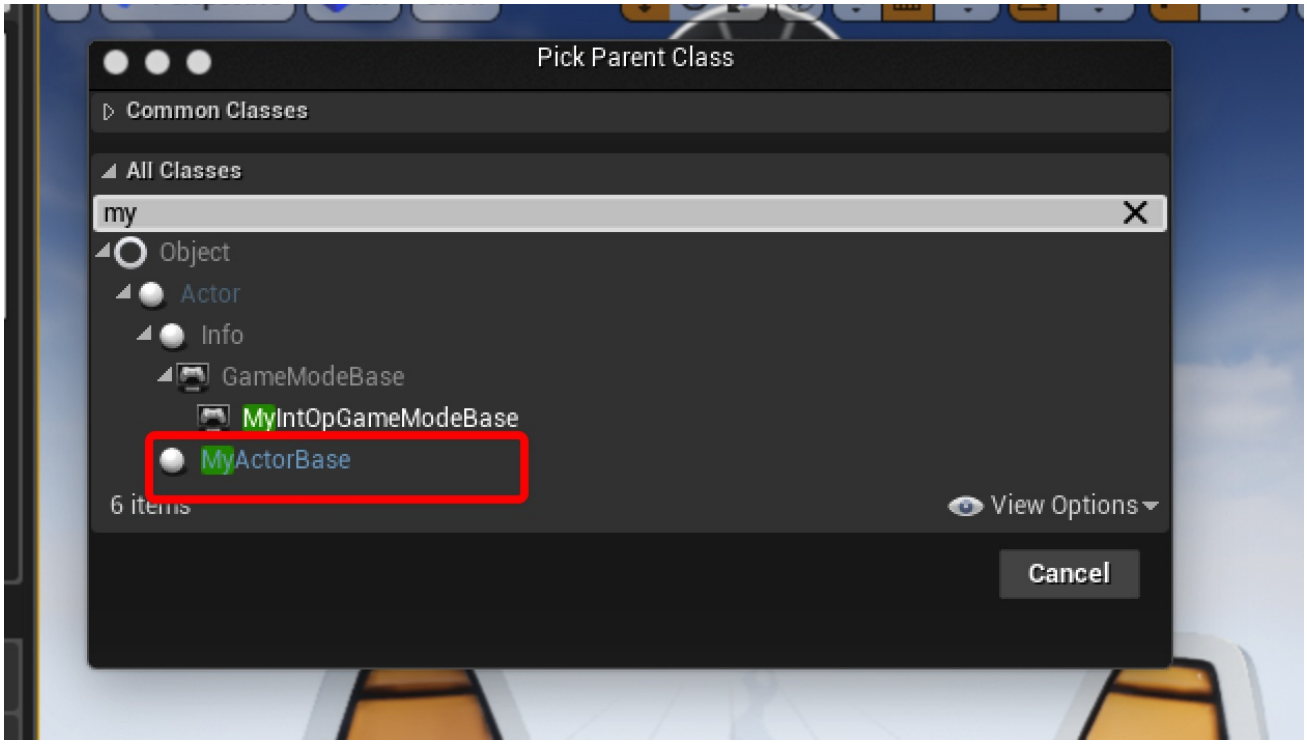
1 c++类作为buleprint的基类

---

c++类作为基类给蓝图使用，UCLASS(Blueprintable)

```
8 UCLASS(Blueprintable)
9 class MYINTOP_API AMyActorBase : public AActor
0 {
```

在编辑器中编译完后就可以直接创建蓝图类继承我们自己用c++创建的类了

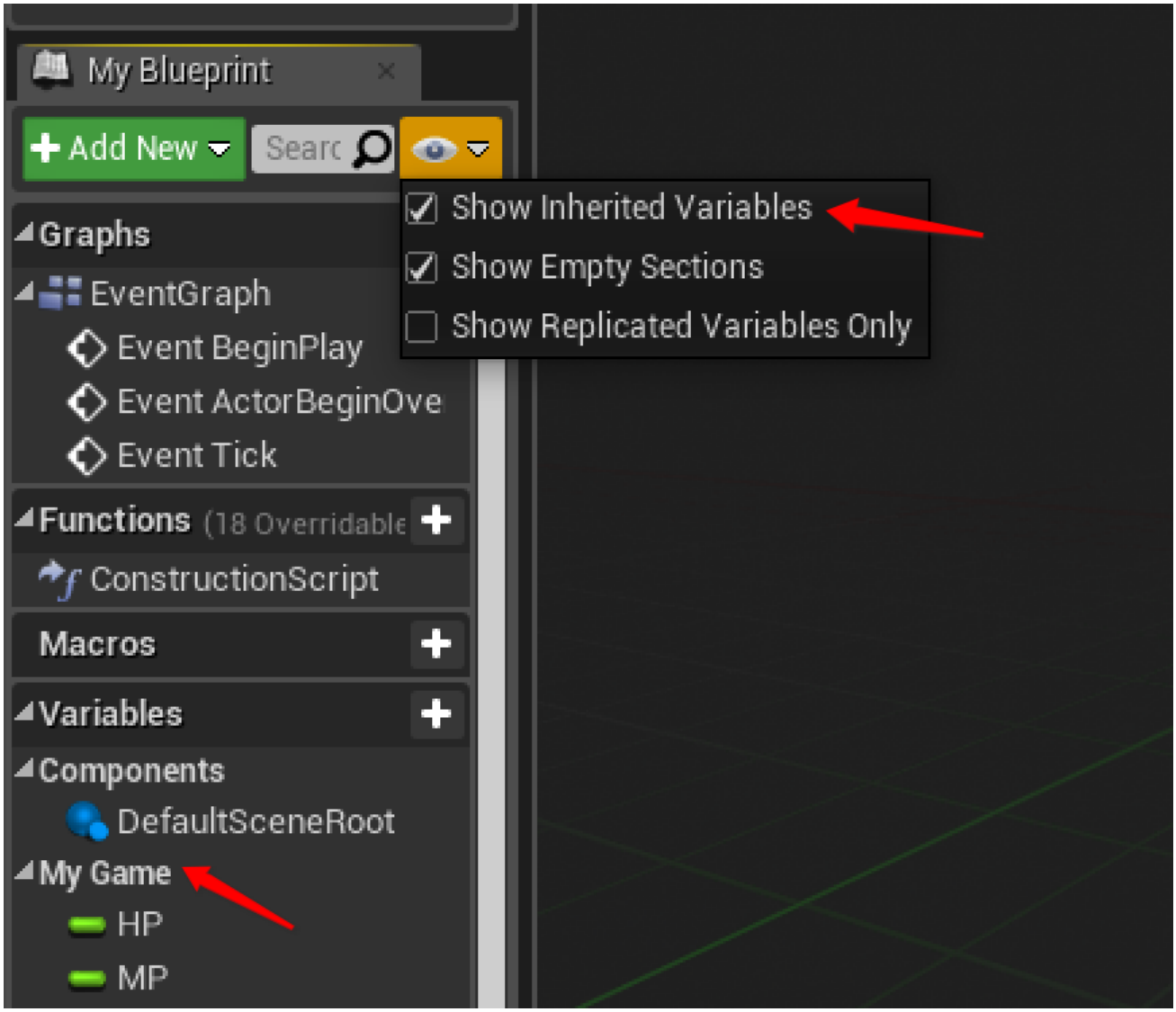


暴露给编辑器

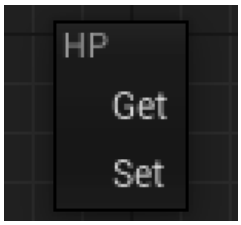
```
UPROPERTY(BlueprintReadWrite, Category="MyGame")
float HP;

UPROPERTY(BlueprintReadOnly, Category="MyGame")
float MP;
```

编译完成后打开蓝图面板 显示父类变量就可以看到暴露的变量了



HP可读可写

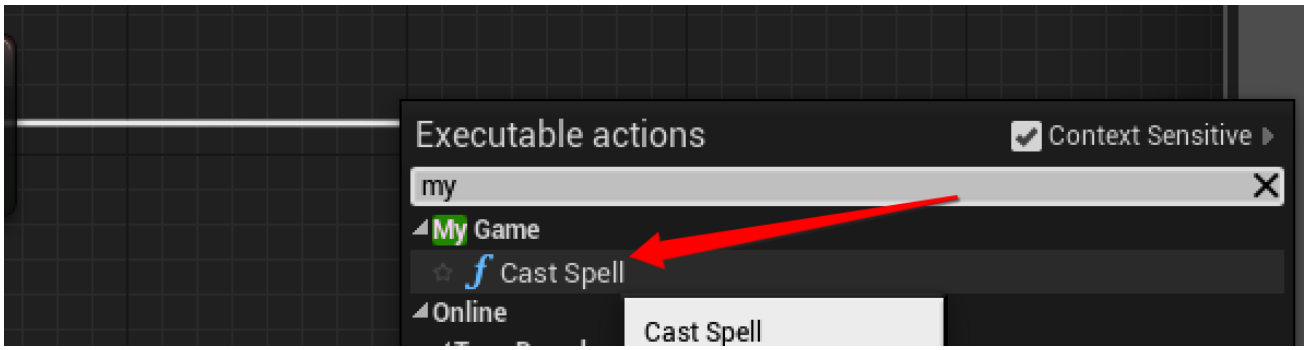


MP只能读不能写

暴露函数给蓝图

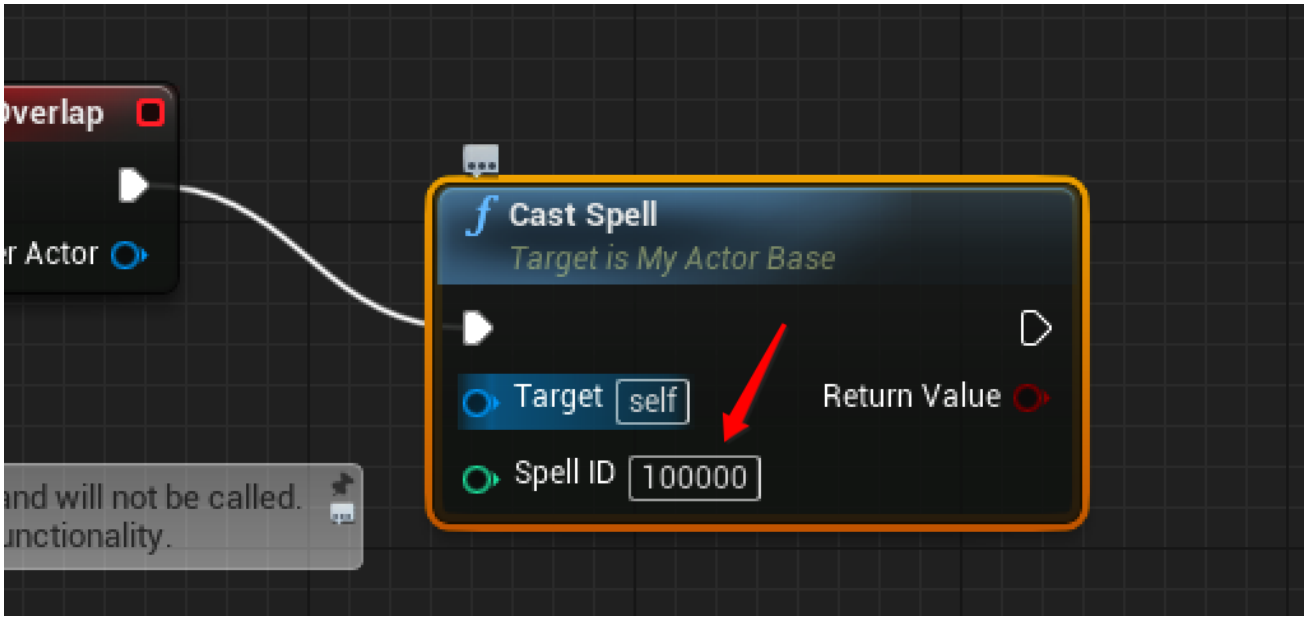
UFUNCTION(BlueprintCallable, Category="MyGame")

```
UFUNCTION(BlueprintCallable, Category="MyGame")
bool castSpell(int32 spellID);
```



给函数参数设置默认值 编译后就会直接显示出

```
UFUNCTION(BlueprintCallable, Category="MyGame")
bool castSpell(int32 spellID = 100000);
```



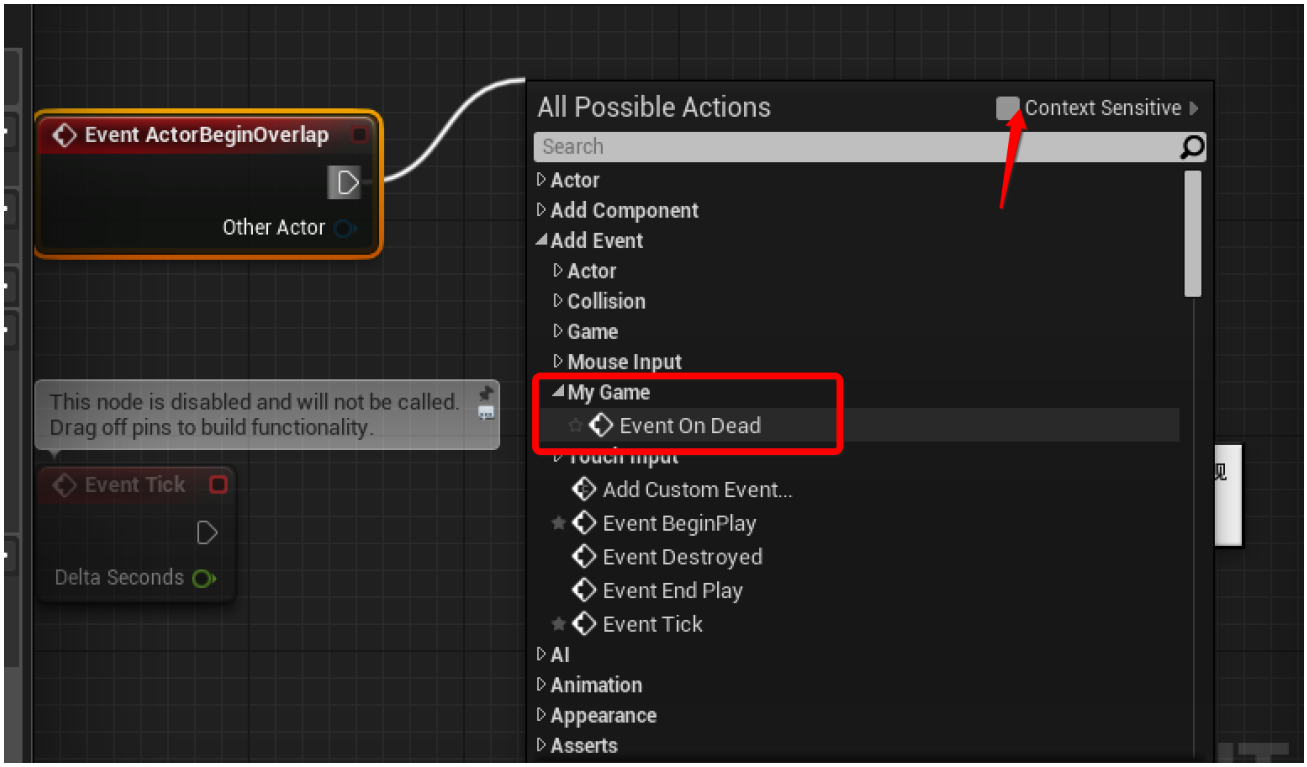
C++调用蓝图的事件

暴露接口给蓝图

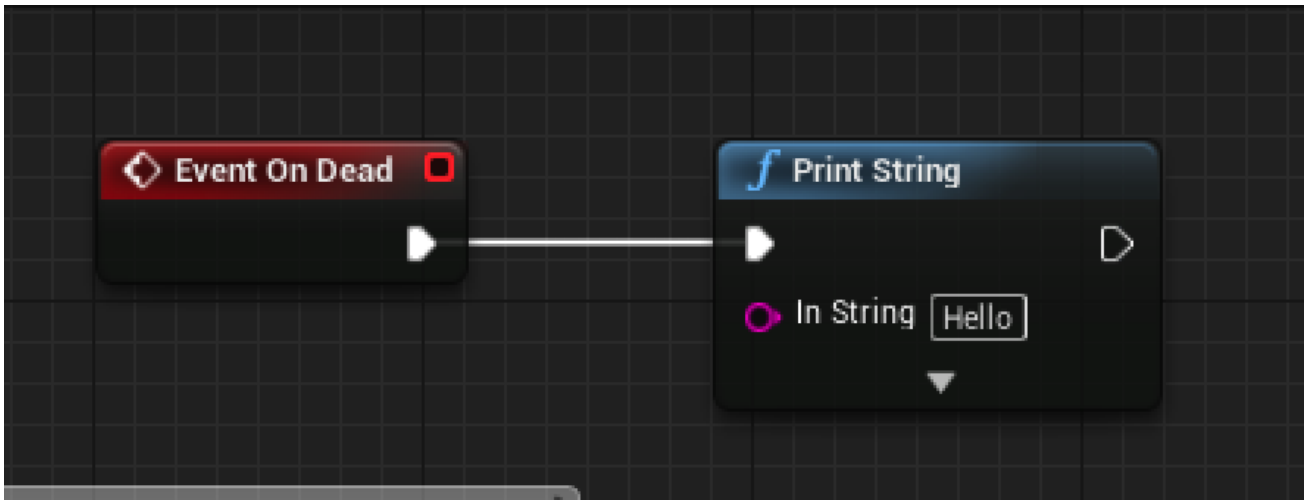
BlueprintImplementableEvent

相当于c++的纯虚函数 留给蓝图去实现

```
//BlueprintImplementableEvent: 在蓝图中实现, c++不用实现
UFUNCTION(BlueprintImplementableEvent, Category="MyGame")
void onDead();
```



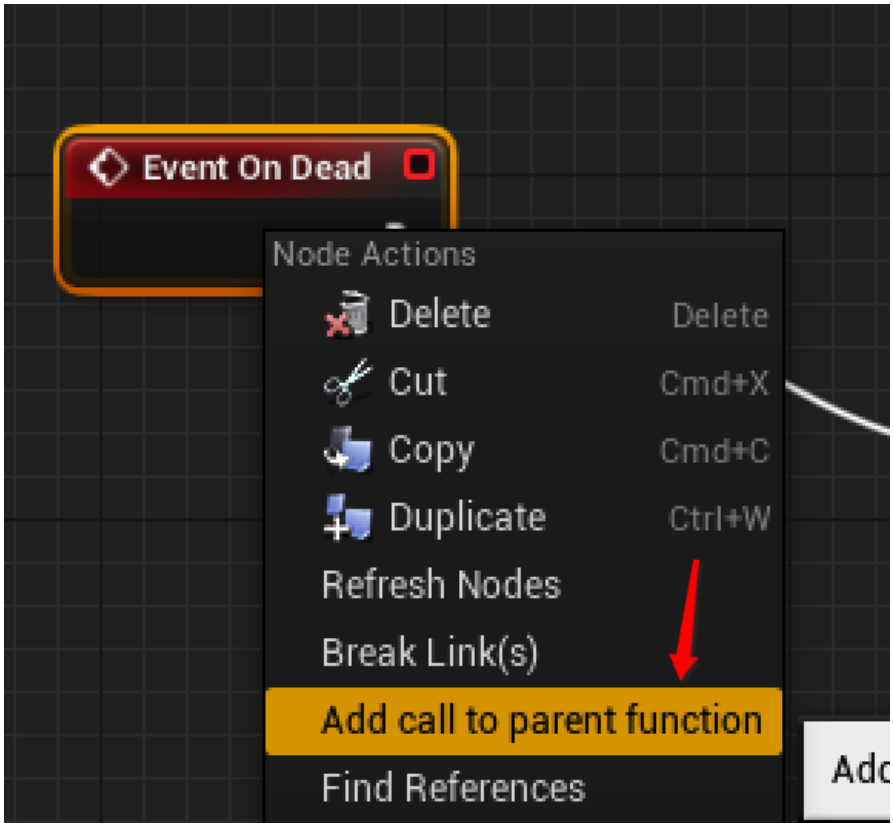
当C++执行到onDead方法是就会执行蓝图的逻辑

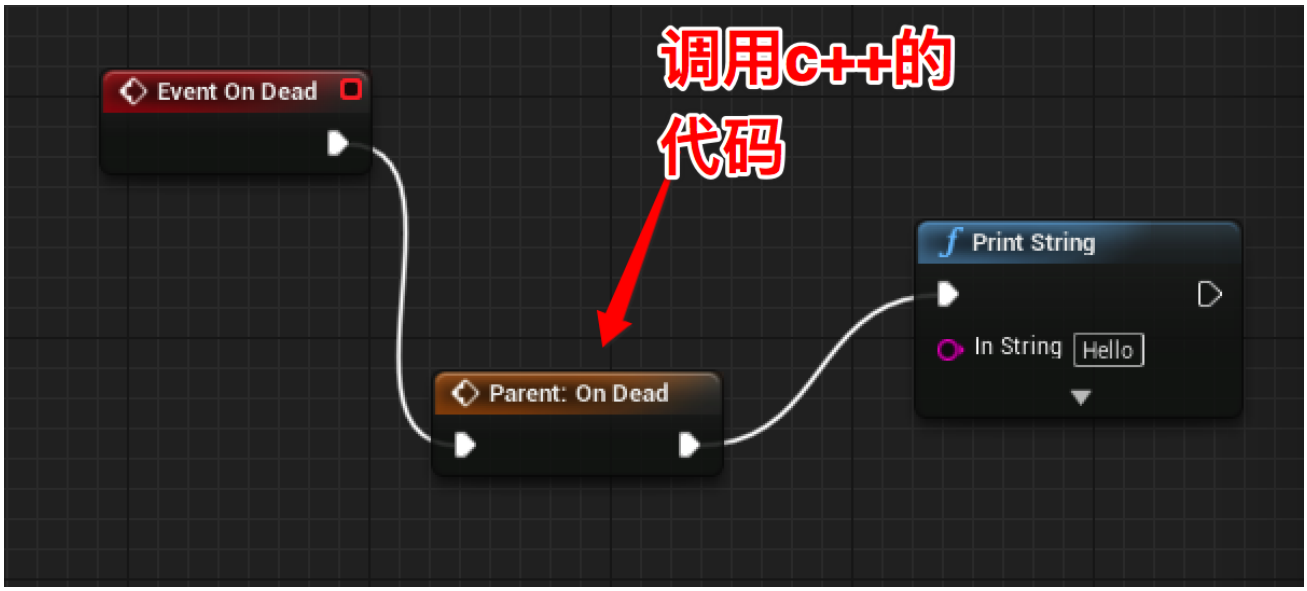


可以在c++实现也可以在蓝图实现，蓝图会重载c++的实现

c++的实现方法必须在后面加_Implementaion，在一个新的方法里实现

```
//可以在c++实现也可以在蓝图实现，蓝图会重载c++的实现
//c++的实现方法必须在后面加_Implementaion
UFUNCTION(BlueprintNativeEvent, Category="MyGame")
void onDead();
void onDead_Implementation();
```

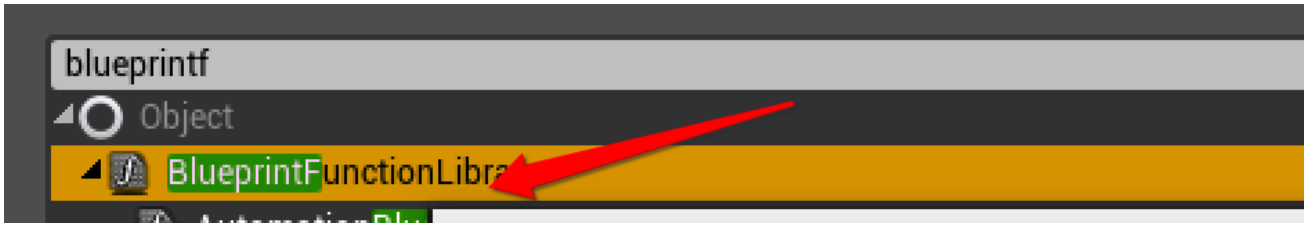




2 c++提供给蓝图一些函数库

项目中的一些工具函数 全局函数之类的
c++的函数更简便的让蓝图来使用

新建一个类，派生于blueprintFunctionLibrary



在c++类你创建方法 必须是静态方法 并暴露给蓝图

```
public:

    UFUNCTION(BlueprintCallable, Category="MyBPLB")
    static void sayHello();
```

编译完成后打开蓝图可以发现我们暴露的方法