GamePlay框架

UHD Class这是一个有关于游戏的UI界面的管理，包括他的生命周期。

游戏状态类 这是一个游戏当前处于的状态，跟人没有关系，比如玩家的得分情况，游戏天气。

旁观者类 这个类就是像吃鸡中的观战模式

在设置中Maps选项下 有一个Game instance 你可以将数据存在这里，每换一次地图，gamemode会改变，但是你设置了这个，相当于他是一个全局。

基本类型

Byte uint8

FName 这是一个常量string 12字节 不区分大小写，比const string效率高，主要是表示一个物体名称

FText 一般作用于显示和本地化

FVector 这是一个三维向量，值得注意

FRotator 旋转变量

FTransform 这是每个actor都带着的变量，这里面有三个值，包括旋转，大小，我们每次编辑actor的形状时都是在这个选项框名称下面编辑的。

第四讲

UPROPERTY

EditAnyWhere 细节面板也可以看见。可修改

EditDefaultOnly 这个只能在类默认设置中可见，细节面板看不到

VisibleAnyWhere 与第一个的区别就是不能修改

BluePrintReadOnly 可以在蓝图搜索中找到

BluePrintRendWrite 可修改

EditInstanceOnly 只有在实例中才可见

元素说明符(meta)

UPROPERTY(EditAnywhere,meta=(ClampMin=“10”，ClampMax=“100”))这个代表最小的数不能够低于他

InstanceEditable 这个就可以实现在实例化中依旧可以修改

ExposeOnSpawn 不知道咋用

UFUNCTION（category=“AAAA”）可以把它们分类到AAA中

BluePrintCallable 这个是有两个引脚的函数

BluePrintPure 这个是带有返回值的函数，他们两个都可以在蓝图中找到

BlueprintImplementableEvent 这个是需要在蓝图中继承的

BlueprintNativeEvent 这个与上面的区别是，他必须在c++中有一个实现

第五讲枚举

UENUM(BlueprintType)//如果不写这个就没办法暴露给蓝图

enum class ECustomColorType :uint8

{

RED,

BLUE,

YELLOW,

};

USTRUCT(BlueprintType)

struct Fxiaomingge

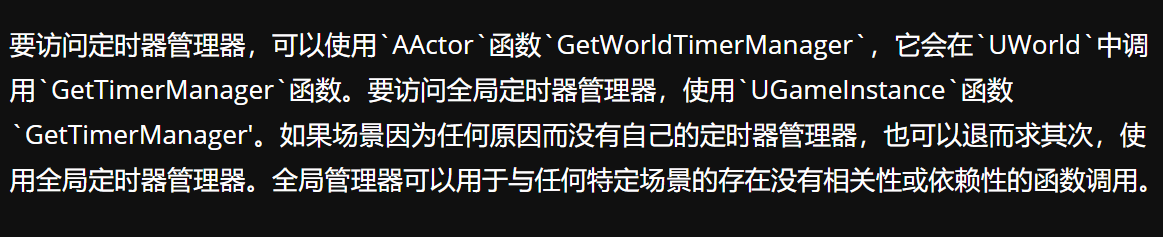
{

GENERATED\_BODY()

int32 vb;

};

Gameplay计时器



用法

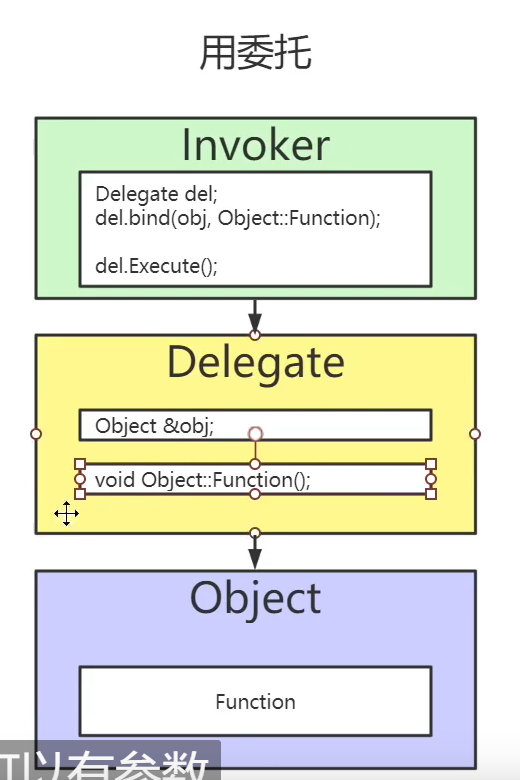


链接

<https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Timers/>

**代理**

代理的全部流程 第一步委托绑定一个对象和函数，第二步委托调用绑定对象的函数，第三步对象调用函数。



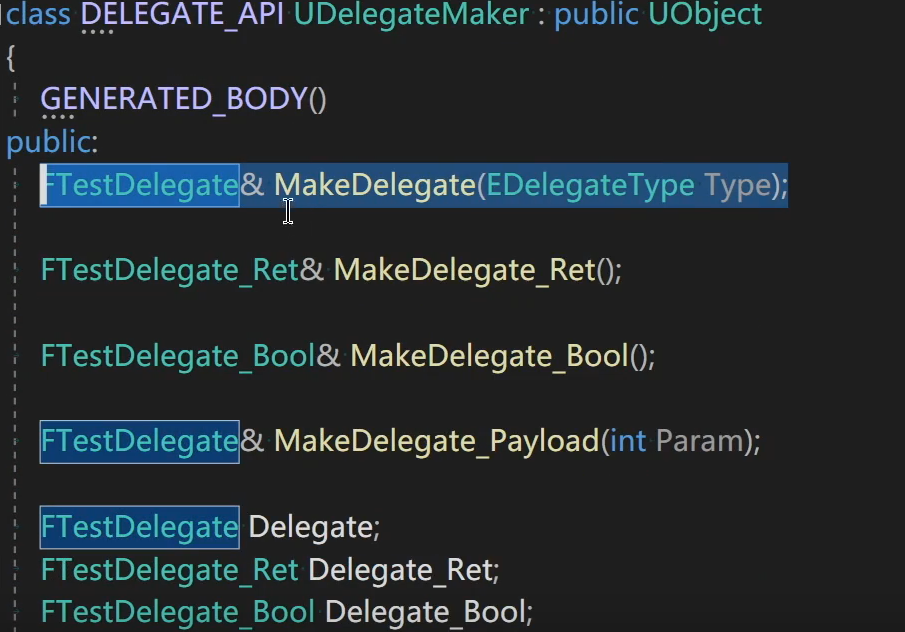
**代码**

委托声明



第一委托声明就是无返回值 ，具体类型从官网查询<https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Delegates/>

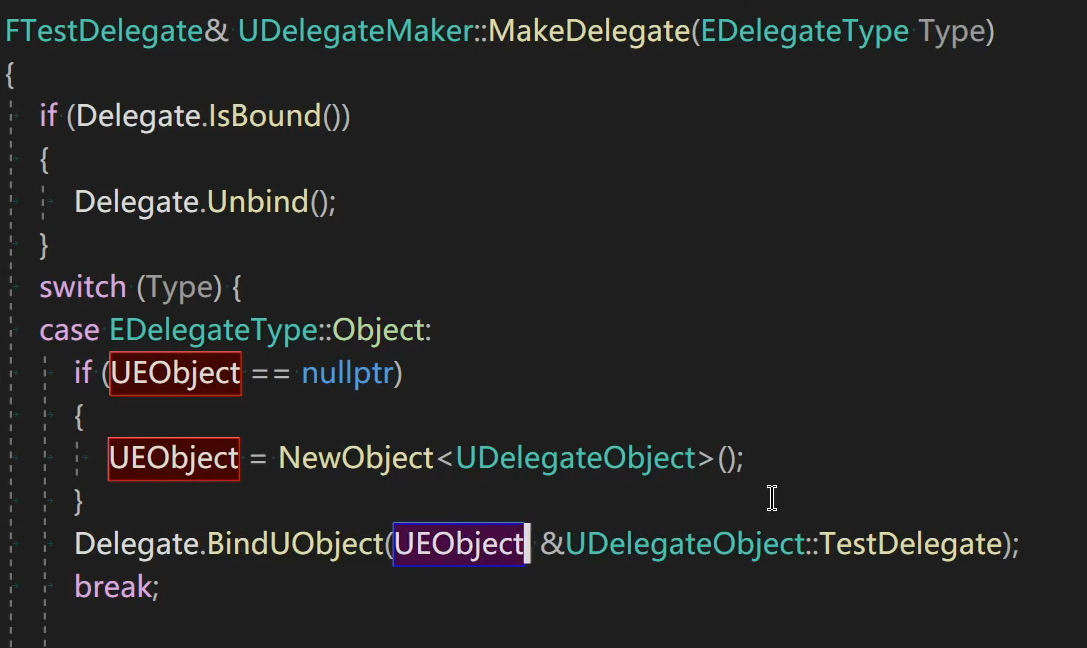
**UDelegteMaker（自定义类） 委托制作者用于委托绑定操作**

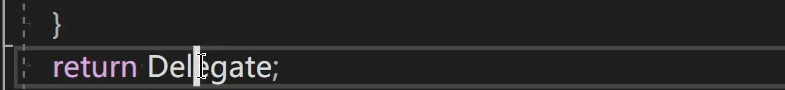


第一个函数 MakeDelgate 是一个委托制作者，他完成我们委托调用流程的第一步，看上面的图片。（后面几个函数都是代表不同的委托，有返回值和有参数）

第二个 我们可以看到类中有一个委托变量 Delegate；

**第一个函数委托制造者 MakeDelgate** 我们可以看出委托绑定的存在多种类型 根据不同类型进行绑定，但他们都是按照前面的一个流程，先绑定.





**第一种绑定 对象绑定（UEObject）**

**第一个参数绑定到对象 ，第二个参数对应的函数（上面图片中）**

**第二种绑定 Lamba函数绑定(匿名函数)**

**Lamba表达式基础用法**

[]（）{

// =是以值的传递方式、

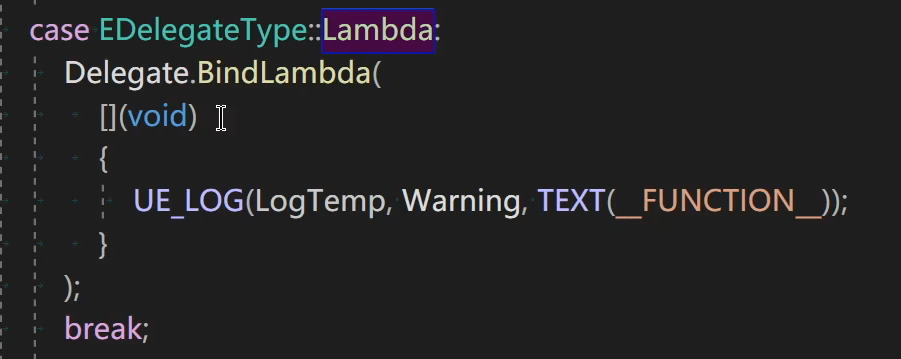
//& 是以引用的方式传递

}

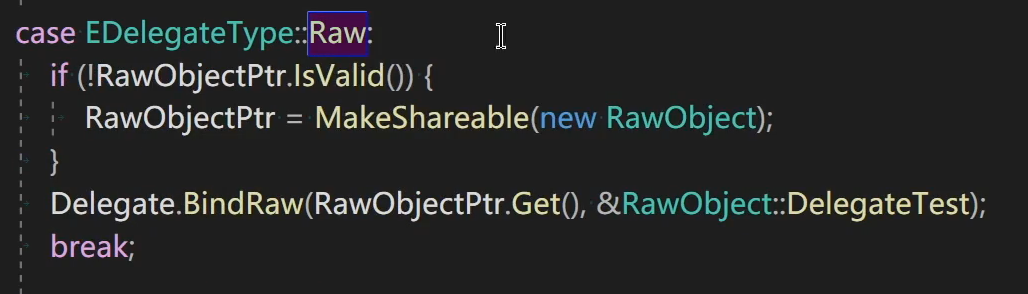
[a,b] (int32 c,float f) - mutable ->int32(这个代表着你要返回的值，不写这个直接在下面写return 会报错)

{

}



**第三种绑定 原始资源绑定**

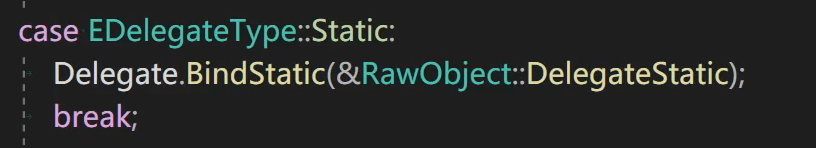


核心是通过获取原始对象（不是经过ue4封装的对象）指针来获取对象，进行调用

**第四种绑定 智能指针的绑定 （SP=shared\_Ptr）**

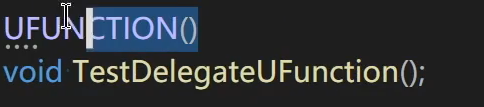


**第五种绑定 静态函数绑定**

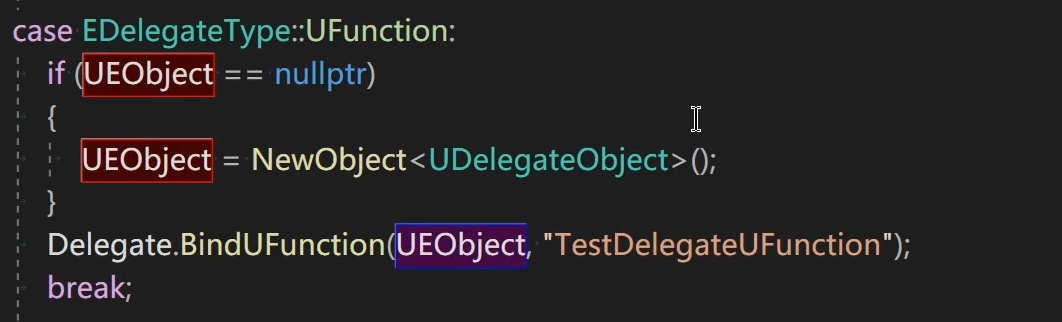


**第六种绑定 ue4函数绑定**

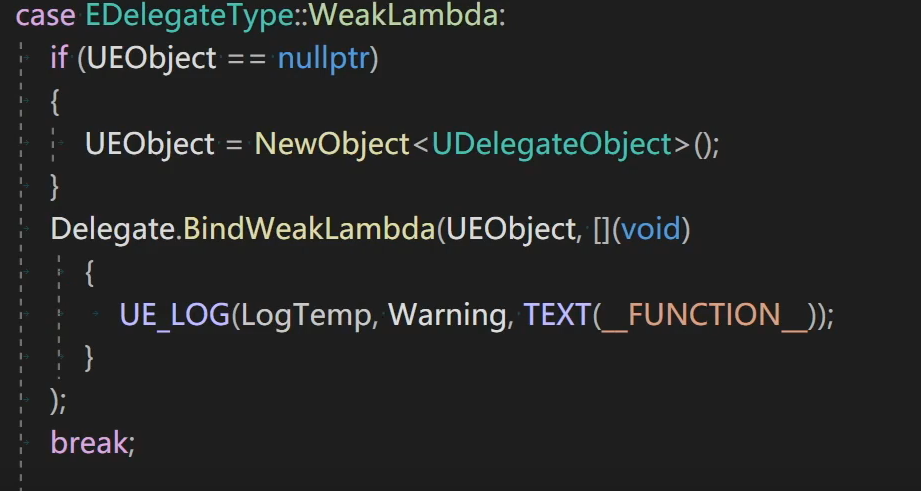
**当我们定义了ue4的函数的时候。**



我们可以直接通过字符串找到函数，与前面的通过类名找到函数不同。（注意绑定的第二个参数差别）



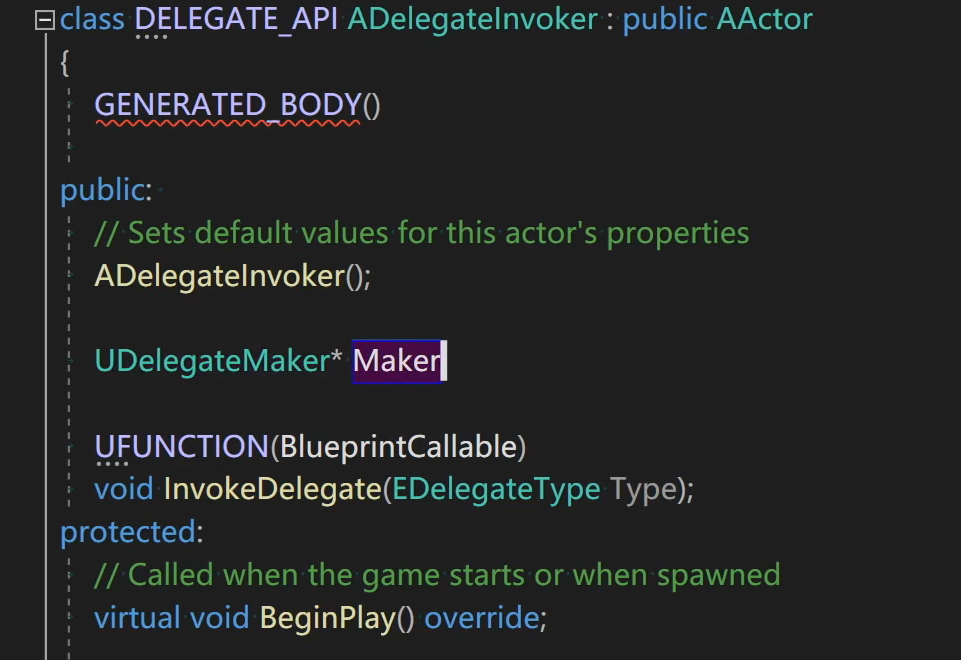
**第七种绑定 弱的Lambda表达式绑定**

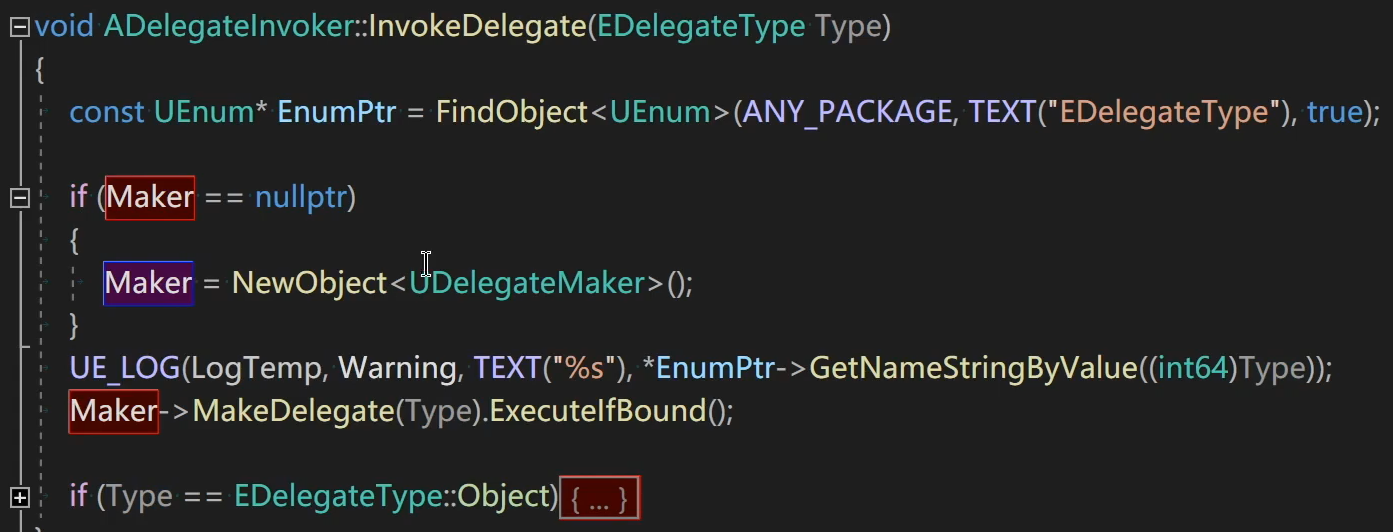


相对于前面的lamba表达式，多了一个ueobject对象，在绑定的时候有可能这个对象挂掉了，那么这个弱的Lambda表达式会保留一个对象的指针，用于检查也就是UEobject，如果指针无效，那么就会执行这个绑定的下面一句话报错。（多了一个保险）

**委托调用**

**（根据前面的委托制造者的绑定，我们这边书写了委托制作者的指针，用于调用）**



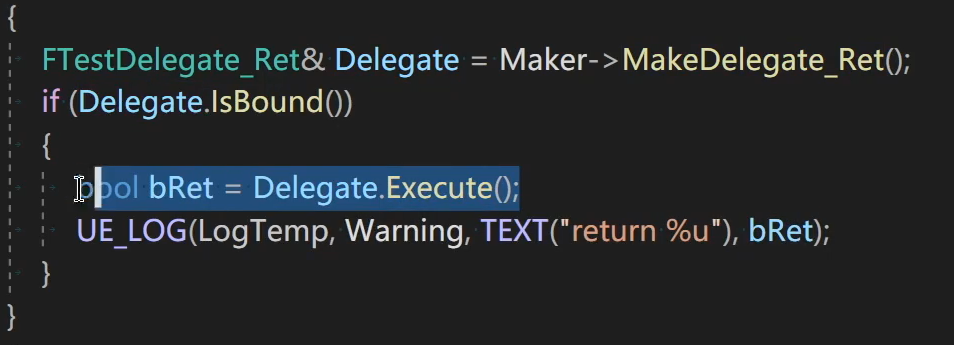


这个函数中，我们首先判断委托是否有效，其次进行最关键的一步，根据前面的委托制作者调用委托制造并且执行，我们需要注意，我们这个委托调用函数传入了Type类型，表明是那个委托，这个类型传入到MakeDelegate(Type)中，所以我们流程是，需要向委托调用者传入一个类型，而不是委托制作者。

**到此为止我们就可以清楚的看到委托的整个的流程，UP主 阿棍儿是真的牛逼**

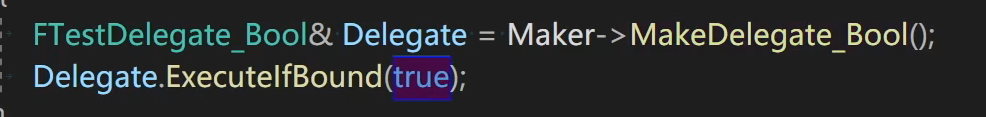
**有返回值的委托**

（绑定的过程前面的过程一样，委托的调用我们可以看到委托变量的调用返回了一个值。）



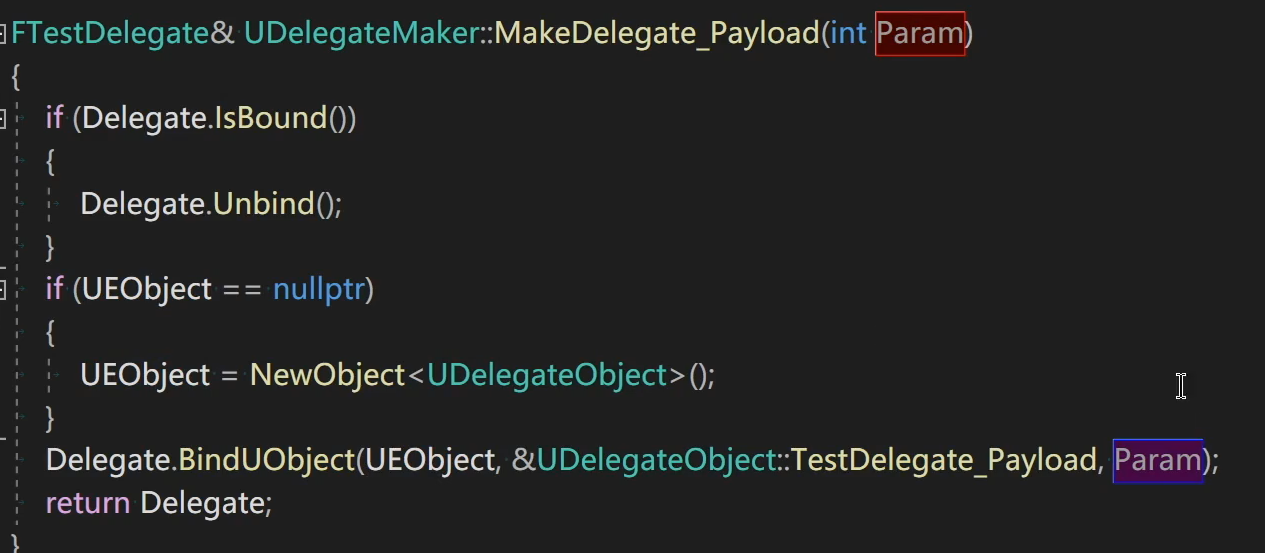
**有参数的委托**

（绑定的过程前面的过程一样，委托的调用我们可以看到委托变量的调用可以传入一个参数）

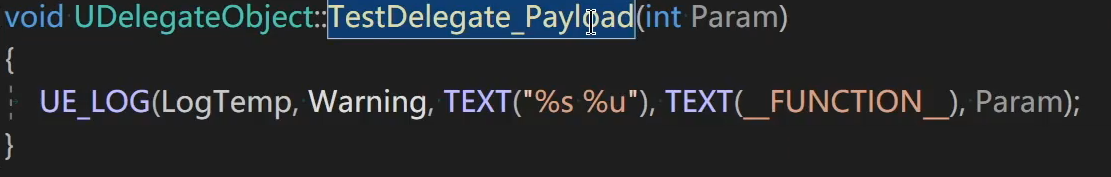


**有负载的委托**

注意他的绑定过程，除了对象，函数，还有一个负载值，这个负载值可以绑定几个取决于你定义的函数（TestDelegate\_Payload），最多可以存在4个负载值 ，负载是绑定的时候传入参数，参数是调用的时候传入，这是两者区别。



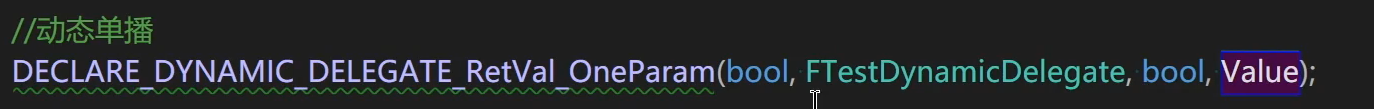
函数定义了一个变量，前面的绑定就可以把负载传个这个参数（负载和这个参数名字都一样）



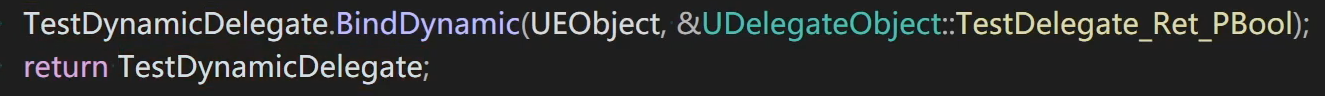
**动态委托**

clipboard.png

动态委托不能使用负载，否则会奔溃，中间加了一个dynamic



其余的过程都和单播委托相似，他的绑定过程有点不一样，BindDynamic这个不是一个函数，而是一个宏，其函数按名查找，前面我们也做到了按名查找，不过传入的是一个字符串，这里没有传入一个字符串，宏BindDynamic作用就是可以将函数名转换成一个字符串。



**多播代理**

和单播代理使用相同，声明有所区别。

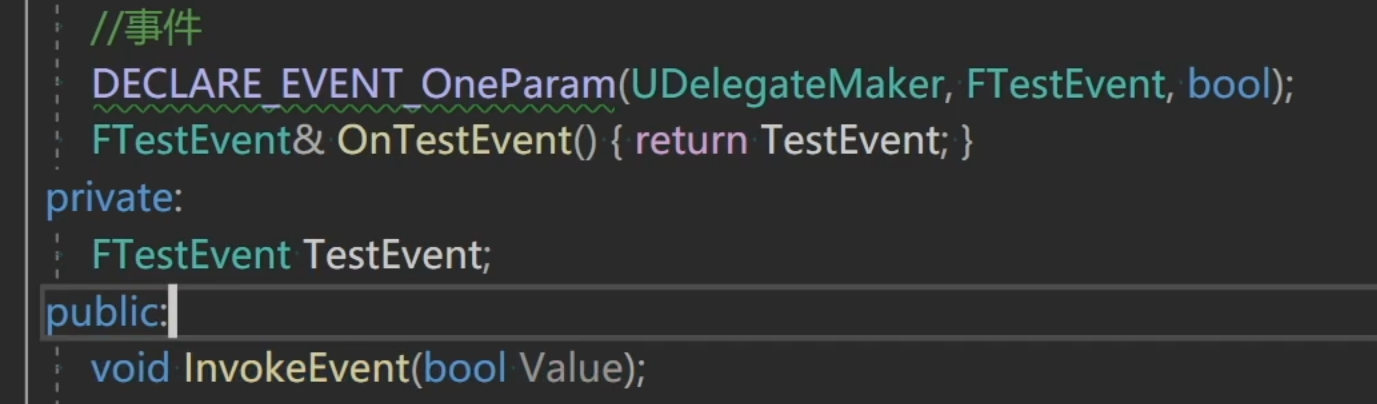
**事件**

可以理解为特殊的多播代理。 声明 第一个是UDelegateMaker，代表类的名字。所以他的使用限制在类中。第二个参数是类型名称。

事件的书写是有套路的。

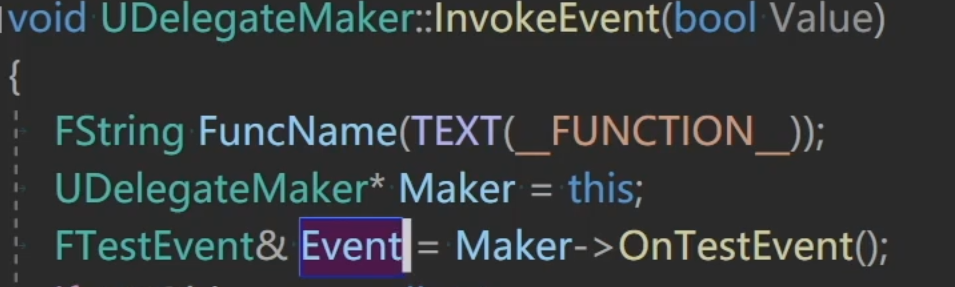
第一先声明。

第二 按照声明下面的函数格式书写一个函数（函数名的开头 都是On 后面的TestEvent 随便起名称）



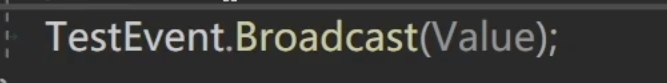
**创造事件**

跟代理制造机一样。不过代理制造机是早制造代理的过程中，进行绑定，这里我们是，先创造出事件，再进行绑定，他的绑定和多播一模一样



**事件调用**

多播也是使用broadcast，



//单播代理的具体可以看

下面链接 https://zhuanlan.zhihu.com/p/126630820 这个是代理的大概流程。

DECLARE\_DELEGATE(FTestDelegateNoparam);//我们这个定义的类型一般都加上F，因为在动态代理中必须要加上F，所以我们统一标准。

DECLARE\_DELEGATE\_TwoParams(FTestDelegateTwoparams, float, const FString &); 注意第二个和第一个的区别在于多了一个TwoParams代表的是有两个参数所以，也可以这样ThereParams

DECLARE\_DELEGATE\_RetVal\_TwoParams(int32, FTestDelegateTwparamsRetval, float, const FString &);注意第三个和第二个的区别在于多了一个Retval代表他有一个返回值。

//多播代理

上绑定的函数执行的顺序是不确定的

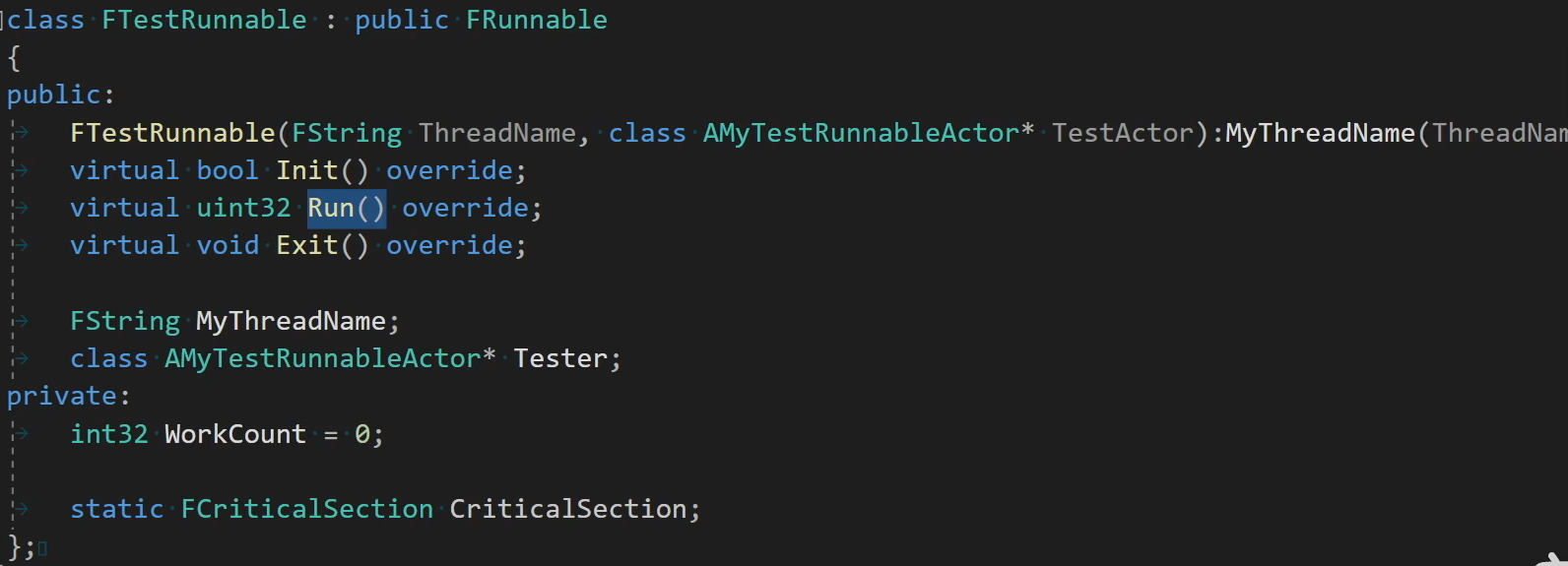
多播代理没有返回值类型，所以他不能添加一个有返回值的函数，他定义代理类的类型和单播一样。

//动态单播代理（需要暴露给蓝图使用的时候就用这根。）

DECLARE\_DELEGATE\_RetVal\_TwoParams(int32, FTestDelegateTwparamsRetval,int32，a）这个需要带一个变量名，这个是非常耗性能的。

**多线程**

先继承FRunnable 然后重新实现三个函数。



创建线程调用

