# 多播代理

多播代理无法使用带有返回值的函数进行构建

##### 构建宏

DECLARE\_MULTICAST\_DELEGATE(DMNotify)

注意DMNotify名字是自定义的。用来当做类型使用。

使用此类型进行代理对象的构建

**注意构建宏，决定了你能够绑定的函数的类型**

**是否带参，是否带返回值**

##### 声明代理格式：

DMNotify done;

##### 常用函数

Add函数

##### 广播

调用函数Broadcast进行广播，但是注意，在广播之前必须要进行检查是否存在有效的绑定，使用函数IsBound

**广播顺序不能保证先加入的被先通知**

##### 使用步骤

1. 使用宏构建代理类型
2. 在类内部构建成员变量，数据类型是多播代理类型
3. 进行通知绑定
4. 执行通知

# 动态代理

允许被序列化的数据结构，这将使得代理可以被数据化提供给蓝图进行使用。达到C++中调用代理广播，事件通知到蓝图中。

**动态代理分为动态单播代理和动态多播代理。其中动态单播无法被绑定到蓝图中，而动态多播可以被绑定到蓝图中**

**动态代理绑定的目标函数都必须使用UFUNCTION进行标记，否则绑定将失败**

**注意：构建动态代理时，类型的声明必须用F开头，声明结束后加分号 ；**

**动态代理只能绑定通知UObject类及子类**

**因为要序列化啊**

##### 构建动态单播代理

DECLARE\_DYNAMIC\_DELEGATE(FDDNotify);//构建动态单播类型

声明

FDDNotify DDn;//构建动态单播对象

绑定通知函数

//创建的一个局部变量

auto Target = GetWorld()->SpawnActor<AHeroActor>();//动态创建一个对象，直接加入到场景中

DDn.BindDynamic(Target, &AHeroActor::SayHello);//绑定的函数必须使用宏UFUNCTION进行标记，否则无法绑定

通知回调

DDn.ExecuteIfBound();

##### 构建动态多播

动态多播没有返回值

声明动态多播类型

DECLARE\_DYNAMIC\_MULTICAST\_DELEGATE(FDDMNotify);//构建动态的多播

声明动态多播类型数据

FDDMNotify MNotify;

绑定通知

//创建的一个局部变量

auto Target = GetWorld()->SpawnActor<AHeroActor>();//动态创建一个对象，直接加入到场景中

MNotify.AddDynamic(Target, &AHeroActor::SayHello);//将对象函数（必须使用UFUNCTION标记）绑定到动态多播代理上

进行通知回调

if (MNotify.IsBound())//进行检查是否存在有效的绑定

{

MNotify.Broadcast();//进行广播

}

**构建带参动态多播需要注意，必须提供参数类型和名称**

DECLARE\_DYNAMIC\_MULTICAST\_DELEGATE\_OneParam(FDDMNotifyOne, int32, Num);//构建动态多播带参类型

##### 构建动态代理用于通知到蓝图

只有动态多播才可以绑定到蓝图，像是派发器

构建动态多播类型，然后声明动态多播代理对象，使用UPROPERTY宏加指令进行标记

UPROPERTY(BlueprintAssignable)

FDDMNotify MNotify;

# UE4中的资源操作

UE4中的资源操作分为直接属性引用和间接引用

### 直接属性引用

通过UPROPERTY(EditDefaultsOnly)宏，将资源属性引用暴露在蓝图编辑器窗口，直接通过编辑器选择资源，暴露的属性为资源指针强引用，操作中需要注意资产的不安全状态。

###### TsubClassOf

**编辑器中拾取UClass用于创建操作对象**

TSubclassOf是提供UClass的安全模板类，通过此模板类我们可以快速在编辑器中进行类型选择（只拾取给定的类型类），帮助我们快速构建某种类型对象数据。

**TS对类型有约束，只能选取模版类型或是继承自模版类型的类或是蓝图**

使用语法

TSubclassOf<T> type;

type.Get();//获取UClass

构建指定类的对象

UPROPERTY(EditAnywhere)

UClass\* classT;

UPROPERTY(EditAnywhere)

TSubclassOf<AMyActor> ActorType;

借助TSubClass生成对象

GetWorld()->SpawnActor(CppActorClass);

### 构造加载

在构造函数中可以借助构造函数资产加载类进行资源引用，更加方便便捷。

静态资源引用类ConstructorHelpers可以进行类引用，源资源引用，注意ConstructorHelpers只能在构造函数中使用

常用资源加入分类

FClassFinder 常用来加载创建后的蓝图对象

FObjectFinder 用来加载各种资源，如音频，图片，材质，静态网格

FClassFinder语法

ConstructorHelpers::FClassFinder<APawn>(TEXT("/Game/Flappybird/Blueprints/BP\_Bird")).Class;//返回数据类型是TSubClassof

也可以使用如下路径拾取蓝图对象

**static** ConstructorHelpers::FClassFinder<AActor> UnitSelector(TEXT("Blueprint'/Game/Blueprints/MyBlueprint.MyBlueprint\_C'"));

FObjectFinder语法

ConstructorHelpers::FObjectFinder<UTexture2D> BarFillObj(TEXT("/Game/UI/HUD/BarFill"));

BarFillTexture = BarFillObj.Object;

注意：

1. 操作路径前加入/Game/前缀
2. ConstructorHelpers 类将尝试在内存中查找该资产，如果找不到，则进行加载
3. ConstructorHelpers只能在构造函数中使用
4. 如果加载失败或是未找到资源，对象内的资产属性为null