#### 自定义事件与变量传参

###### 一、自定义事件

Add Custom Event节点



//一个拥有自定义名称和参数的事件

节点名称可以自命名，假设命名成“我自己的初始化”



然后直接空白处右键搜索自定义事件的名称，也就是“我自己的初始化”



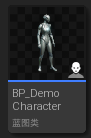
//我自己的初始化

//目标是<自定义事件在定义时所在的蓝图>

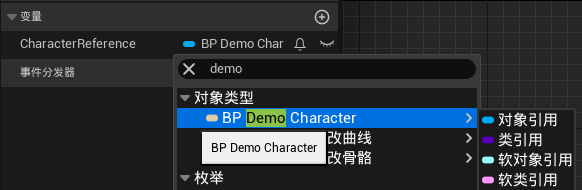
自定义事件的发生原理是：蓝色节点“我自己的初始化”是一个封装起来的函数，封装的内容是自定义事件红色节点连接的节点族

###### 二、变量类型

变量类型可以为某蓝图的（软）对象/类引用。如我们有如下资产



我们可以定义一个变量，变量类型为该类的对象引用，查找方式为蓝图类的名称



这样一种程度上可以跨蓝图类使用蓝图的reference。

定义好的变量，右侧有一个闭着的眼睛，会显示“变量并非公有，不能在此蓝图的实例上进行编辑”。如果把眼睛打开，细节中的“可编辑实例”也会打勾，并显示“变量为公有，可在此蓝图的每个实例上进行编辑”。这一重点在于细节中的“可编辑实例”，表示的是“该变量在此蓝图的实例上是否可公开编辑”。

在细节中同样有另外一个选项“生成时公开”，表示“要在生成此蓝图时让此变量公开为引脚”。打勾后，可以查看该蓝图类在其他类中的节点，比如该控件蓝图类在某角色Actor类中的Create Widget（该控件蓝图类）节点，节点下方会出现你自定义的那个变量名称的接口，接口类型为你自定义的变量的类型。因为该节点在角色蓝图中，直接在该蓝图中获得一个对自身的引用（Get a reference to self）即可，并将新节点给新接口进行赋值。本质是给这个空间函数增加一个输入变量，作用是将角色自己的数据传递给Widget，当角色自己发生变化的时候，Widget也能获取到变化的数据

###### 三、UI与Actor之间进行传参

假设某角色Actor有一个属性是得分，我们可以给这个角色设置一个变量命名为Score。

假设UMG中有一个文本（通用）要显示角色的Score

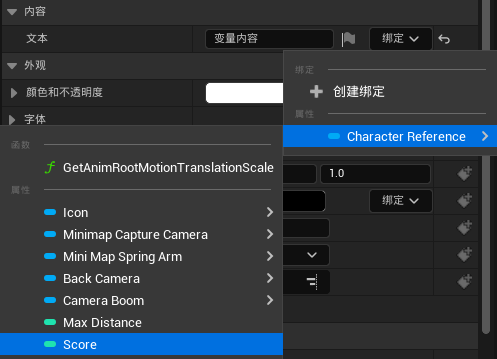
首先我们需要在UMG蓝图中创建一个角色的对象引用

然后回到UMG的设计器，点击要用来显示Score内容的文本（通用），在细节中的“内容--文本”，选择**绑定进行对Actor中变量的绑定**

接下来再假设：角色Actor可以发射一种子弹Actor，当子弹Actor触碰到另一触发区域Actor时，可以给Actor的Score进行增加，增加的值为子弹的ScoreTake变量，表示当子弹命中触发区域时子弹能够提供的分数

###### 四、Actor与Actor之间进行传参

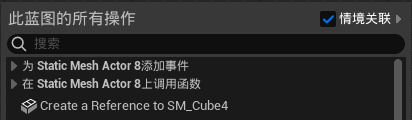
在触发区域的蓝图中，为了判断是否有球与BoxCollision发生碰撞，应当会将OnComponentBeginOverlap节点和CastTo<子弹Actor的名称>节点，CastTo节点拥有一个返回值，为As<子弹Actor的名称>，从该返回值处拉出引线，即可找到子弹Actor的变量的Get/Set。这样角色Actor就能够获得子弹Actor的变量，并将此变量用于给自己的Score变量传值。当然我们可以在触发区域Actor中将上述内容进行一个运算。在触发区域Actor中，**首先Get Player Pawn，将其返回值传递给CastTo<角色Actor的命名>，获得角色索引**，并从CastTo节点的返回值处拉出引线，获得角色Actor的Score变量。（注意，此时CastTo节点是需要连接白色串行线的，可以右键CastTo节点，点击“转换为纯类型转换”，这样就不需要连接串行线了）。若想实现Score=Score+ScoreTake的代码蓝图，可以先Get Score & ScoreTake，然后将计算结果传递给Set Score的参数



CastTo节点是很耗费性能的，所以我们要用蓝图接口替换实现。

###### 五、Actor与Level之间的传参

打开关卡蓝图，然后回到关卡设计页面，选中即将进行传参的obj，然后在关卡蓝图中右键，即可创建一个对关卡中某Actor的引用。



###### 六、事件分发器

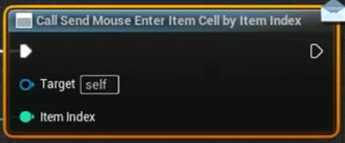
事件分发器表示一个关注。

当A对象里执行一个函数F的时候，如果想让B对象知晓，那么就用事件分发器。

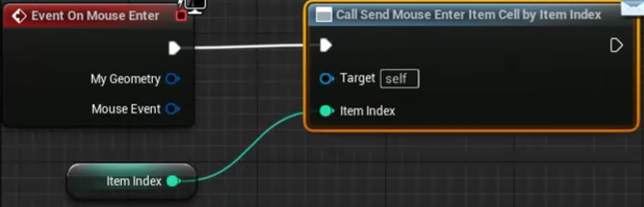
A的执行函数F1里调用事件分发器

B里面绑定事件分发器到函数F2

需要注意点：事件分发器的参数必须与F2一致。

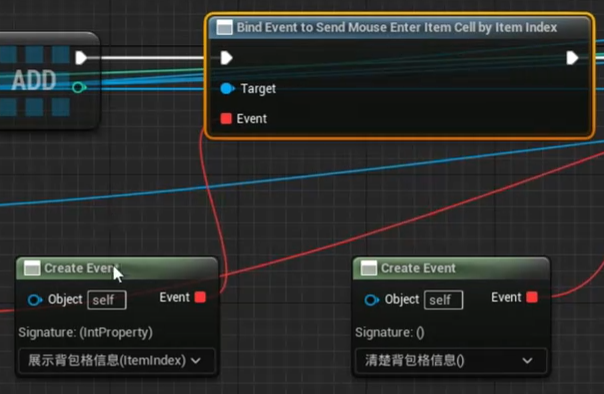
1. 调用  
   

调用方法：call <事件分发器名称>



1. 绑定

在B里面，Bind。绑定的F2就是这个Event下面对应的“展示背包格信息”



1. 注意

绑定的函数的参数和事件分发器的参数是一致的。

蓝图与蓝图之间进行信息传递的11种方式

1. Get All Actors Of Class  
   性能消耗是很大的，不建议使用
2. Get All Actors With Tag  
   通过给场景的Actor打上Tag，在这里获取。
3. Get All Actors With Interface  
   获取场景中所有包含这个接口的所有Actor类的对象的引用。
4. GetGameInstance
5. GameINstanceSubsystem
6. 对象引用变量
7. 关卡蓝图中直接获取实例化对象的引用
8. Execute Console Command
9. MultiLineTraceByChannel
10. SpawnActorFromClass

总结一下： 其实两个蓝图之间通信的主要难点就在于如何获取对方的引用。

本质上其实只有三种方法，

第一种是最消耗性能的办法但一定最可靠，就是遍历世界所有的对象，直到找到自己所需的对象。（如果你场景中有几万个对象这种方法不建议使用）

第二中相当于两个蓝图提前把自己的引用存到一个公共区域，然后再在公共区域获取对方的引用（好比人群当中两人找不到对方，于是约定好在超市见面），“公共区域”也不一定只有游戏实例。玩家控制器，游戏状态，玩家状态，游戏模式都可以是“公共区域”。 优点是完全不消耗性能，但是使用步骤相对来说略微麻烦。

第三个通过碰撞来获得引用，此方法使用physx系统遍历碰撞对象来实现的，但本质依旧是一种遍历，只不过有一定的优化，相对来说没有像直接遍历场景对象那么卡，但代价是不一定精准，有些重叠或是已经发生碰撞的物体，不一定会通知。