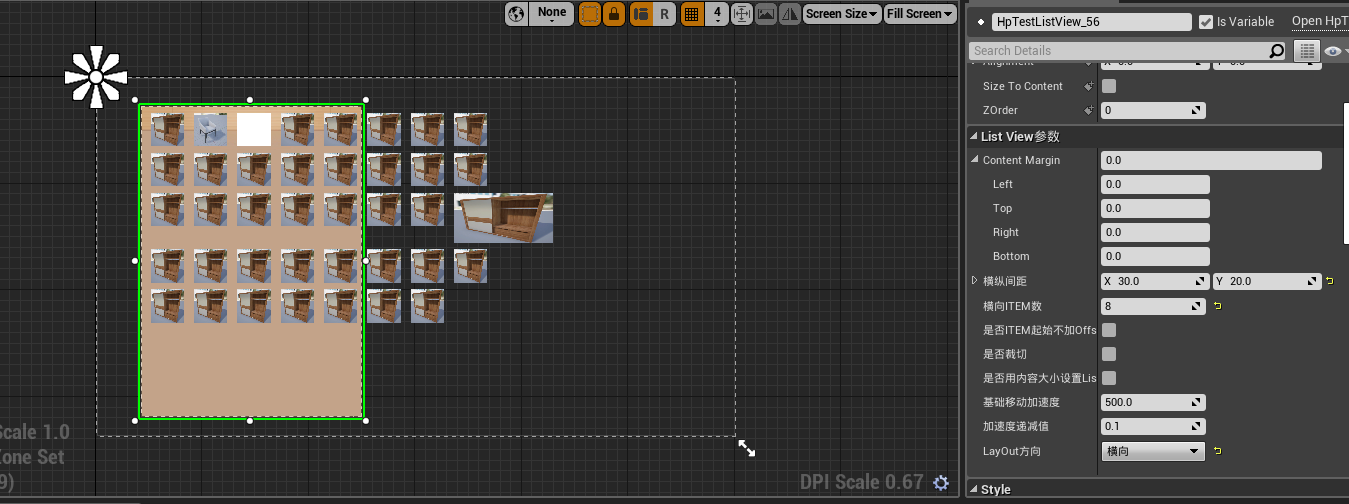
# SlateToUMG详细文档

该文档为使用Slate编写一个ListView的文档记录，说明了Slate编写UMG的详细流程以及注意事项以及一些接口的说明、各个文件的作用等。最后以简易的背包为例说明了自定义参数在UButton等控件中的用法。

本篇不会对所有的Slate写法等进行详细描述，只对运动到的部分进行说明。

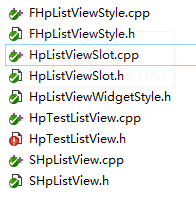
肯定会有不足以及错误的地方，请指正。

## SlateToUMG



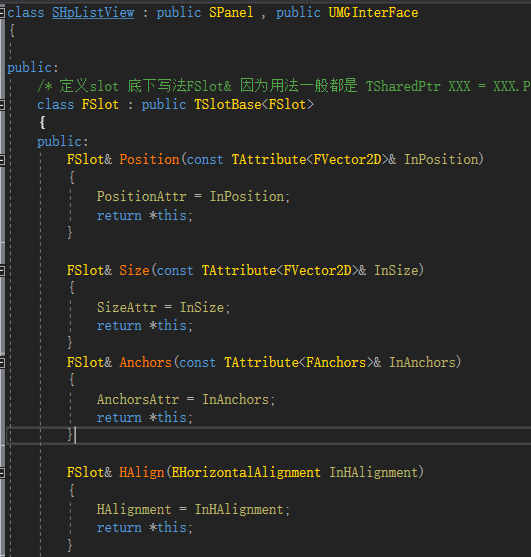
实现后的ListView如图所示，暂时实现的是正常滑动的List，对子项的大小可以自由修改，并且包含了行列数Offset移动速度是否裁剪等可调节参数。

一个转为UMG的listView会有以下几个文件，



### ShpListView

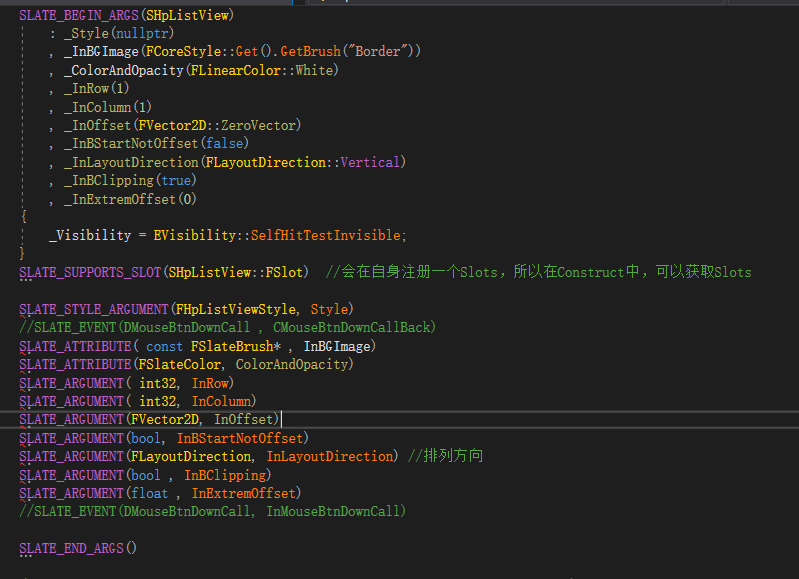
S开头的Slate文件，为该ListView进行显示的核心部分，该部分包括了ListView如何加载Child项并且对其进行排版与绘制；定义了该ListView的Slot；定义了可以由外部进行调改的参数等。这里因为我们是要制作一个List类型的控件，所以我们的SHpListView继承自SPanel

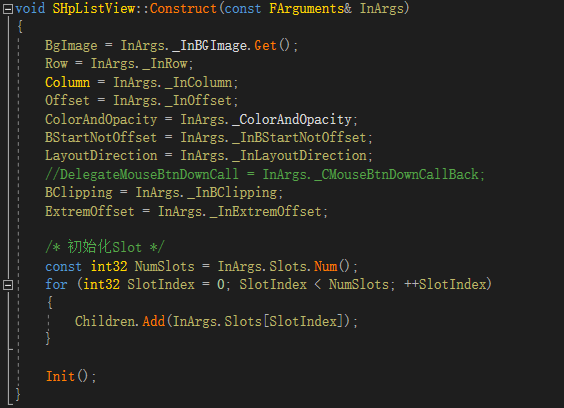


首先在SlistView中定义一个Slot，该Slot就是用来在包括UMG界面中拖入子控件、代码/蓝图生成子控件等情况时，用来存放该子控件的结构，可以自定义所需的任何参数。

定义完Slot之后，就开始编写ListView本身。

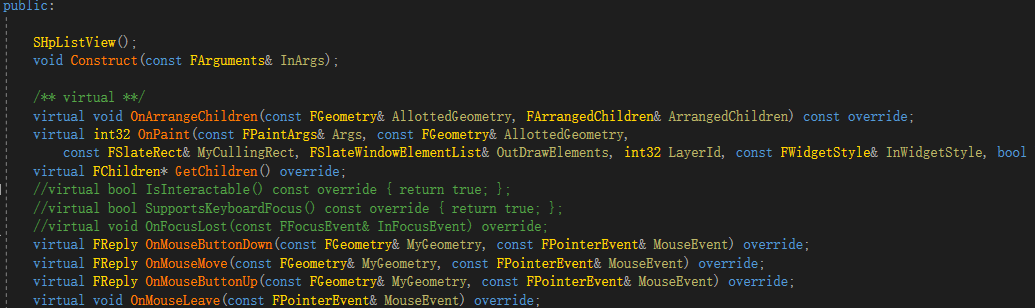
UE4中的Slate定义，是通过宏SLATE\_BEGIN\_ARGS(SHpListView) 与SLATE\_END\_ARGS()来完成的，SLATE\_BEGIN\_ARGS(SHpListView)相当于一个Slate的构造函数，可以在其中传递自己需要的参数，UE会通过调用Construct函数将所传入的参数已FArguments的方式来供我们获取：





添加参数我们使用的是SLATE\_ATTRIBUTE宏以及SLATE\_ARGUMENT宏，这里要注意一个问题就是SLATE\_ATTRIBUTE 属性可以是值，也可以是函数，SLATE\_ARGUMENT 就只能是值

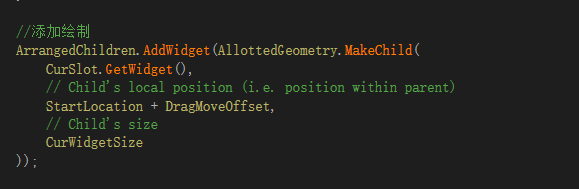
之后我们定义ListView本体的函数：



其中OnMouseButton相关的函数为响应鼠标操作的函数，其参数const FGeometry& MyGeometry, const FPointerEvent& MouseEvent 分别包含了当前的点击位置以及鼠标点击事件包括左键右键等，可以进行相对应的处理。

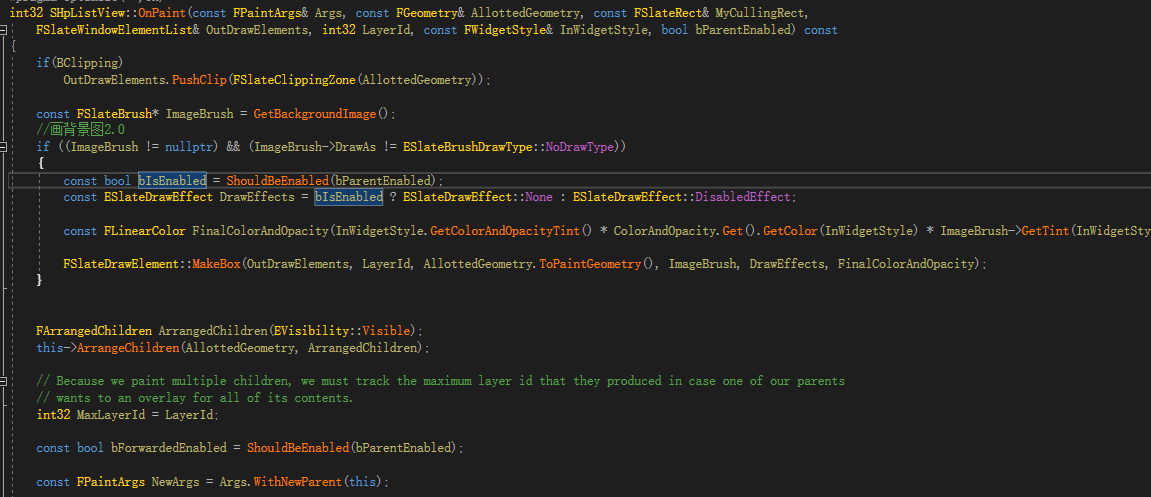
OnArrangeChildren函数以及OnPaint函数是Slate中的关键函数，

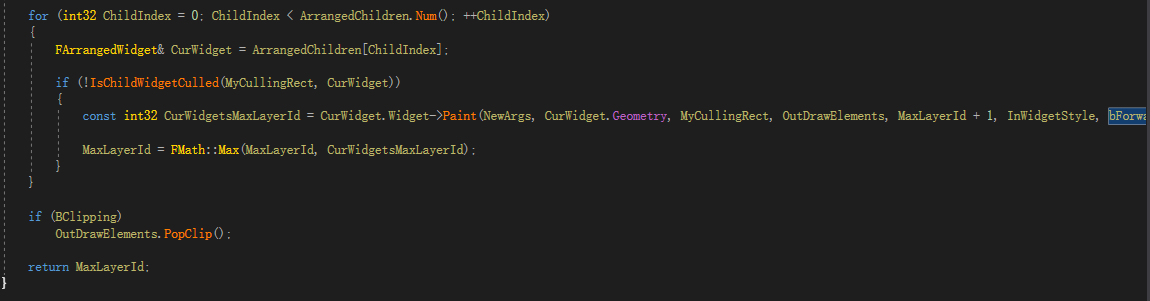




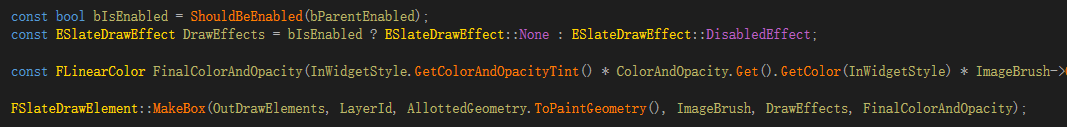
OnArrangeChildren函数中根据需求从该Slate的Slots中选择需要绘制的子控件，并计算其位置和大小后通过AddWidget函数添加进绘制列表中

在OnPaint函数中，通过遍历之前存放好的ArrangedChildren 中的绘制列表，调用对应Widget的Paint函数即可将控件绘制到屏幕中。





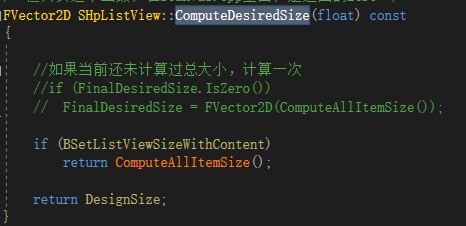
Paint函数中还可以添加自己想要绘制的各种图形，例如在本例中，



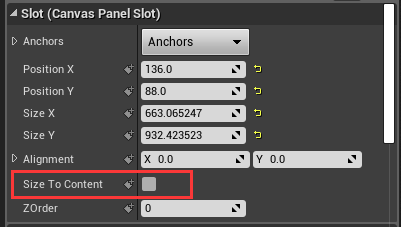
我通过调用MakeBox在ListView中绘制了一个背景。

类似的FSlateDrawElement中还有很多对应的绘制函数，可以根据需求进行自定义的绘制,有兴趣的同学可以去试试。

另外OutDrawElements.PushClip(FSlateClippingZone(AllottedGeometry));函数可以帮助我们进行确定区域的裁切，比如当我们要该List具有裁切功能等。



ComputeDesiredSize函数，用来计算该Slate的大小，由UMG的外部部分进行调用，对于一般的Canvas来说，可以直接返回Zero，因为该Slot的大小由内部的元素来决定，但是在本例中我们希望ListView有一个自身的大小，所以做了对应的计算。特别的，如果控件被加入到一个PanelSlot中，会有一个SizeToContent的选项，



如果该选项勾选上，则会自动将该控件的大小设置为ComputeSize中所计算的大小。



同样重要的还有对应Slot操作的函数，这些函数会由UMG对应的模块调用，将外部添加的子控件加入到新建的Slot中以及删除Slot等。

自此，一个基本的Slate的框架就已经搭建完毕，其中对应需要实现的功能，还有大量的逻辑性的代码以及变量函数等，在此不做累赘，根据需求自由发挥即可,类似于



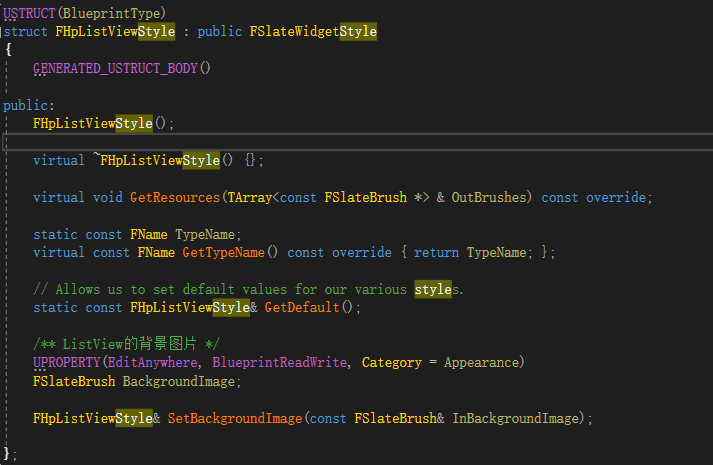
这样的函数，用来给外部的UMG代码进行调用，进而设置该Slate的显示、绘制等逻辑。

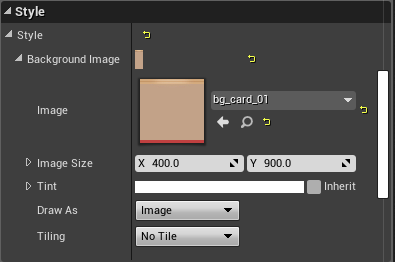
### FHpListViewStyle

Style是Slate的重要组成部分，自定义的Style继承自public FSlateWidgetStyle ，可以包含自己所需的例如图片笔刷，字体样式等的参数，用来设置我们的Slate的显示，之后通过UMG部分设置给对应的Slate控件显示。通过重载

virtual void GetResources( TArray<const FSlateBrush\*> & OutBrushes ) const 方法来传出对应的SlateBrush参数。

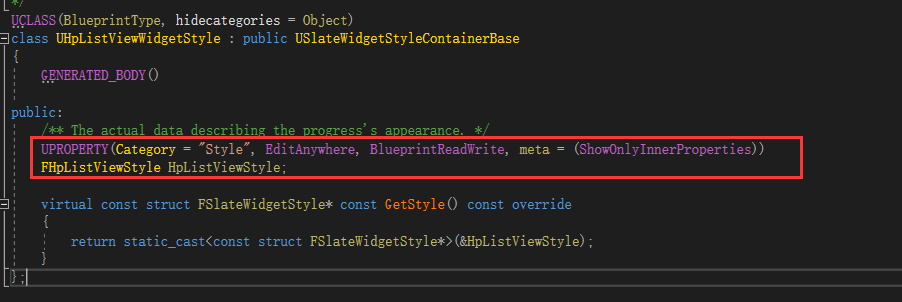
在本例中，只是简单添加了背景图片的笔刷参数，在UMG中显示效果如下，可以进行ListView背景图片的设置。





### HpListViewWidgetStyle.h

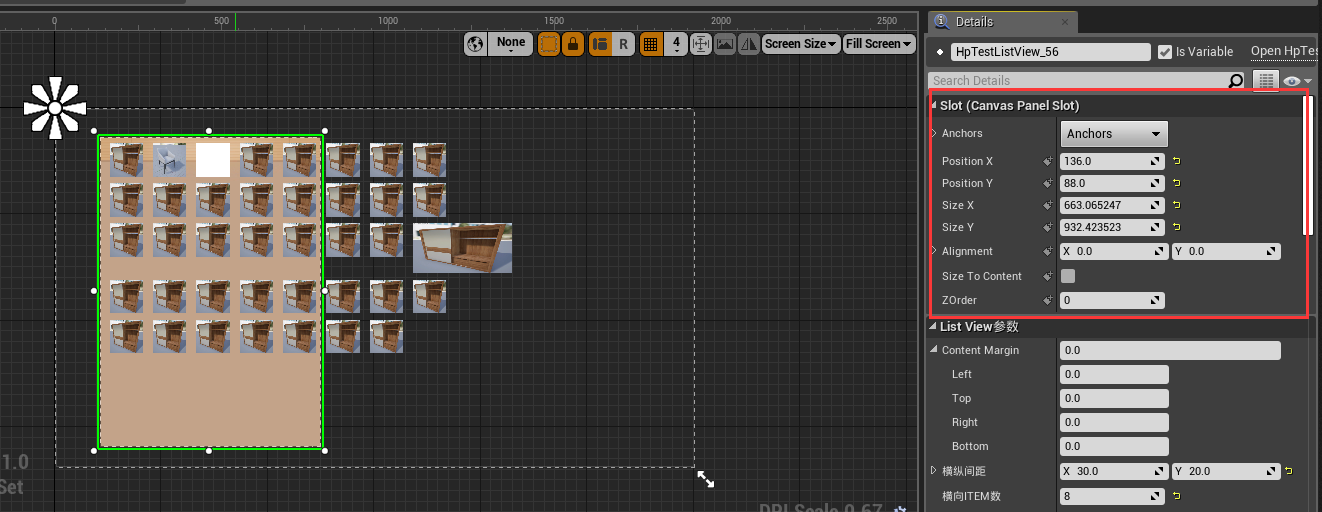
该文件非常的简单，主要是用来将自定义的Style进行一个系统的注册，方便后续进行取用。



其中的GetStyle方法，即是获取对应的Style。

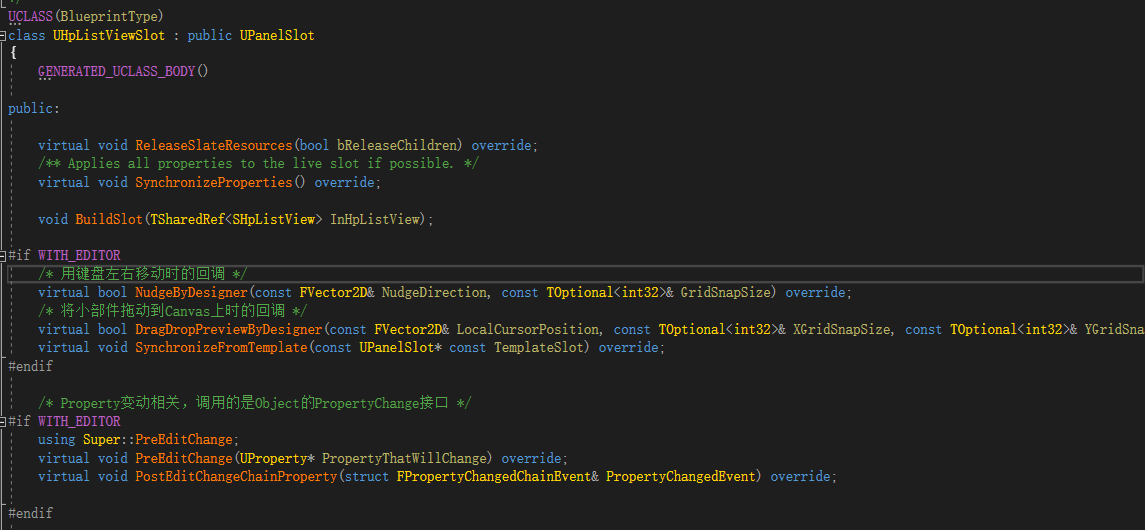
### HpListViewSlot

Slot为UMG的重要组成部分，该Slot与Slate中的Slot不同，是继承自UPanelSlot 的供UMG层使用的插槽，该插槽为UMG控件在父类中放置的位置，当我们在UMG编辑界面中点击任意的控件时，



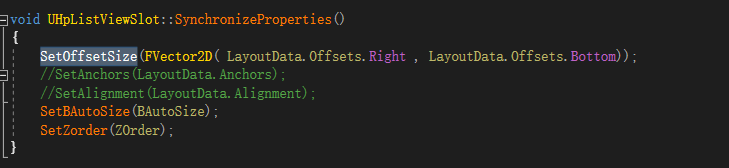
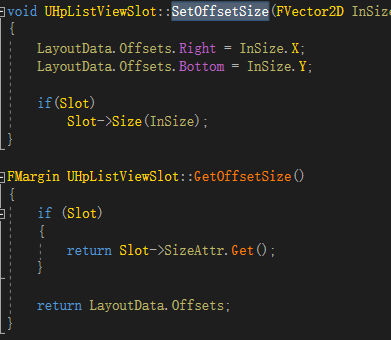
右侧可调节参数，以及左侧编辑部分的拖动，拉伸等，都是通过对父类的Slot进行设置进而更改了参数以及绘制的效果，**USlot也是一个中间的桥梁作用，用来将UMG参数部分传递给Slate中的Slot。**

首先从继承自父类的函数开始，



PreEditChange，PoseEditChangeChainProperty两个函数，是继承自Uobject的函数，他们的作用主要是当标记为UPROPERTY的参数进行变动时调用，来通知进行更新。

SynchronizeProperties 该函数是所有widget上的数据变动时都会调用的参数，不仅仅是在Detail面板中更新数值，也是在Editor中拉大控件时会调用的函数。移动控件，会调用UMG部分的RebuildWidget函数，进而更新整个控件。

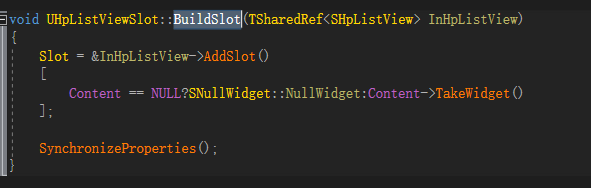
 

virtual bool NudgeByDesigner(const FVector2D& NudgeDirection, const TOptional<int32>& GridSnapSize) { return false; } 当用键盘方向键点击移动时调用

virtual bool DragDropPreviewByDesigner(const FVector2D& LocalCursorPosition, const TOptional<int32>& XGridSnapSize, const TOptional<int32>& YGridSnapSize) { return false; }

把外部的Widget控件拖入时调用。

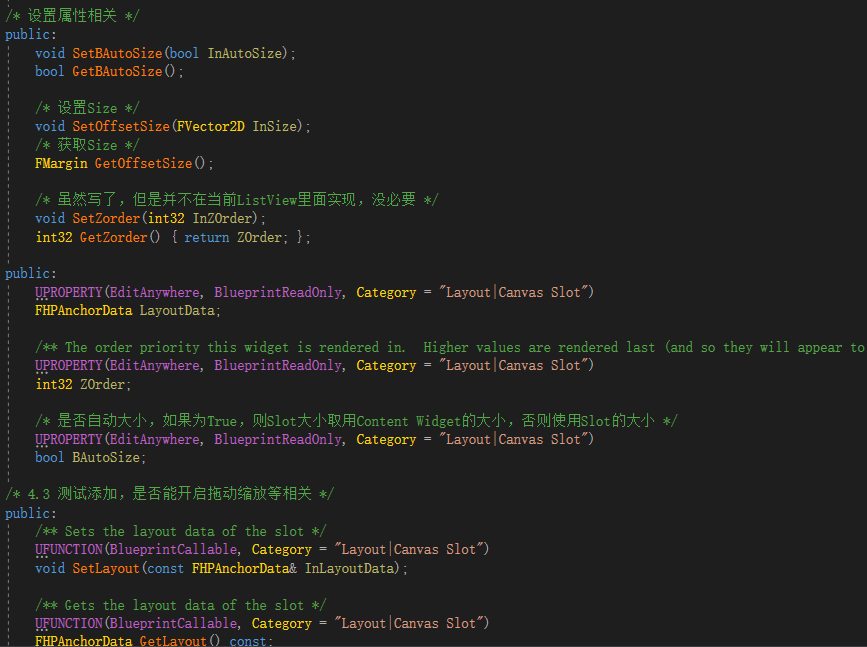
之后最重要的一个函数，即是BuildSlot了，



该函数用来在Slate控件中生成一个Slot，并且将对应的控件设置上去。

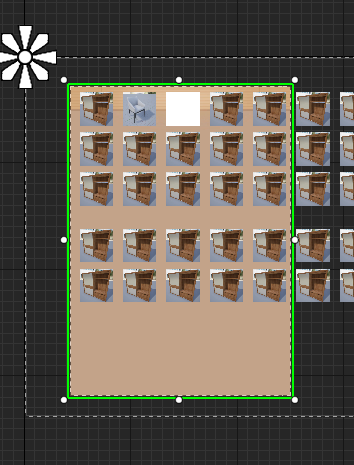
在编辑器中将子控件移入（添加入）的时候，就已经将Content对应的Widget设置了，所以这一步生成一个Slot并且判断Content是否为Null，如果为Null设置一个空Widget进去，防止设置失败。

其余的Slot函数与变量等，就是用来同步参数数据了。



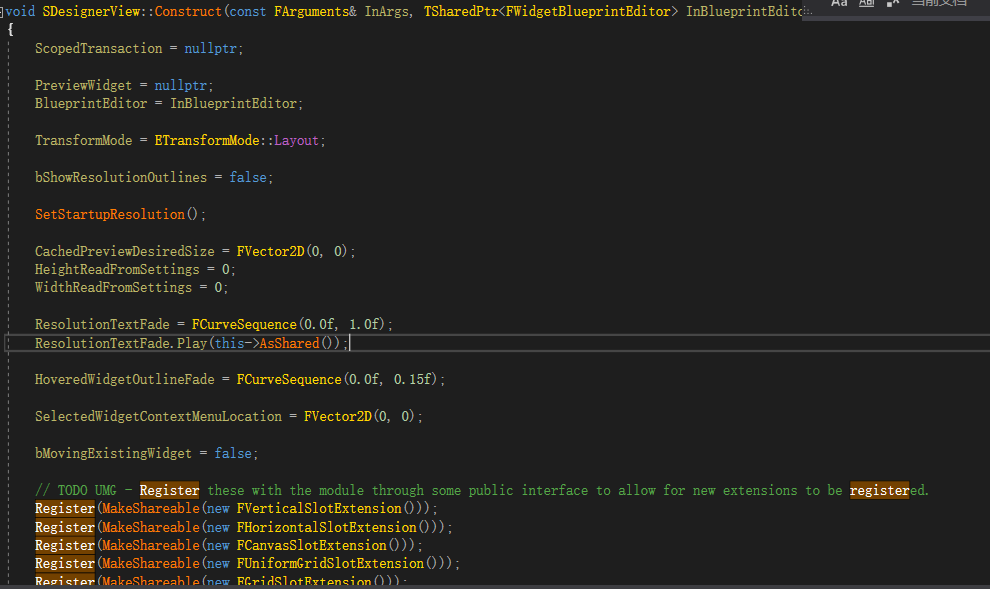
#### 特别说明

在我们点击一个控件的时候，经常会看见

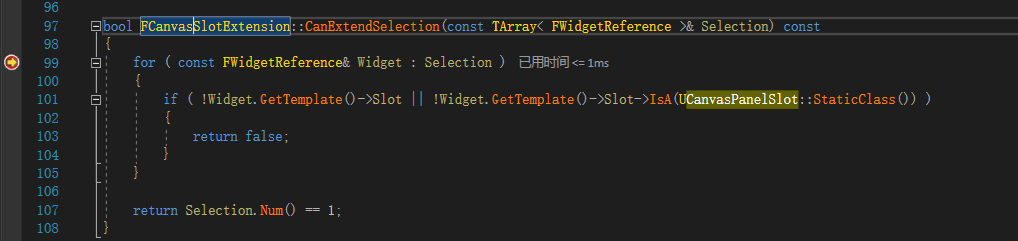


类似于这样的可拖动缩放设置锚点等的标识，是调用的SDesignerView,

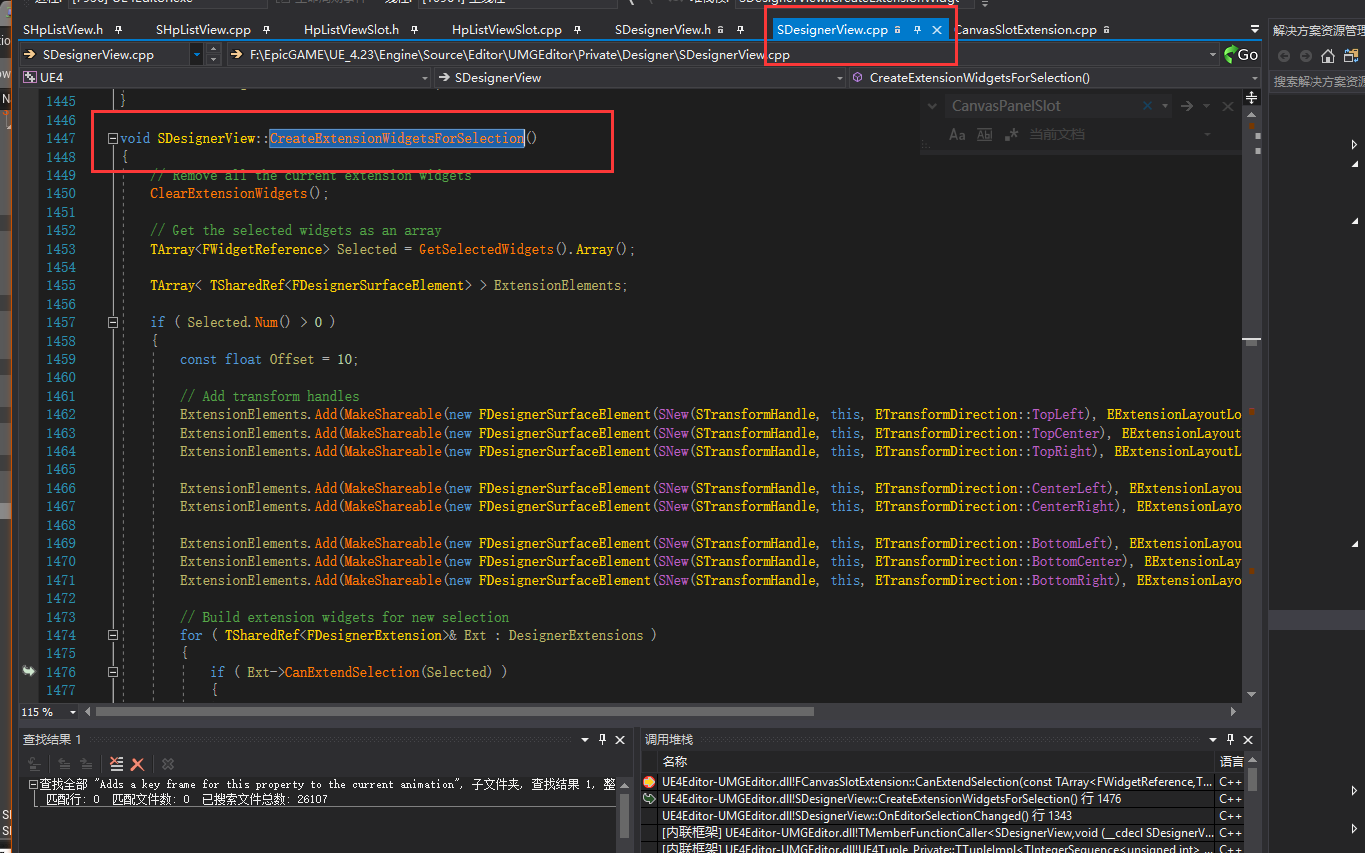
SDesignerView 通过代码可以看到，在里面注册了一些可以扩展的UMG控件，



其中就有CanvasSlot，如果点击UMG，会调用

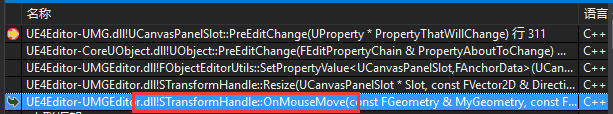


去判断是否为UCanvasPanelSlot，如果是，则会显示扩展

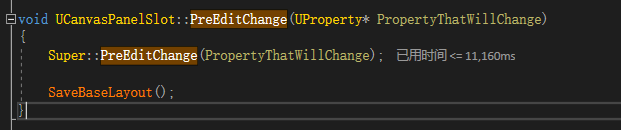


所以，如果我们想要我们自定义的Slot拥有这样的可拖动编辑控件，只需要继承自这几个Slot即可，也可以将自身定义的Slot添加到注册函数中，但是需要修改底层。

拖动控件的修改，会通过调用

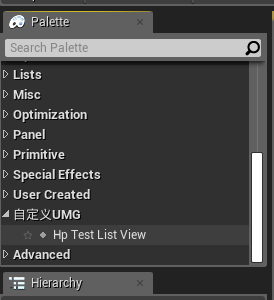


进而调用继承自UObject的PreEditChange函数，来通知拖动的更改，进而可以通过自定义设置来达到更改显示的目的。



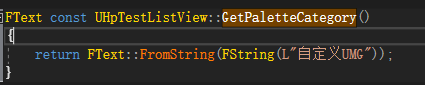
### UHpTestListView

终于到了最后一步，即在UMG编辑器中可显示的UMG本体

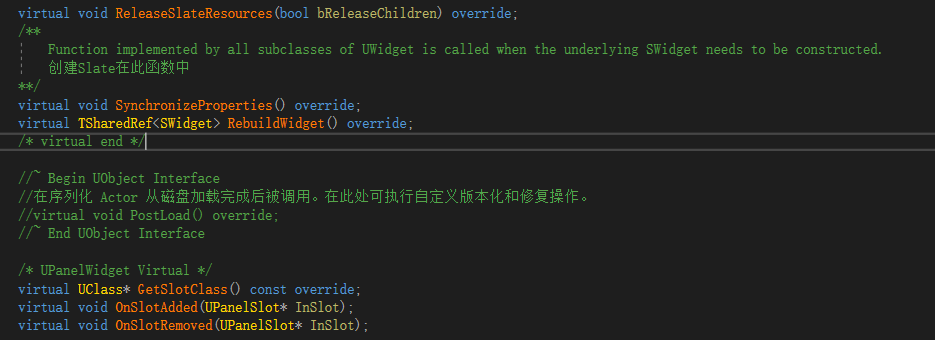


UMG本体部分就相对简单多了，该Widget继承自UPanelWidget（一个所有PanelWidget的基类，拥有Slot属性，可以在其中放入子控件），还是先需要编写继承自父类的方法：

 GetPaletteCategory()为获取分类的方法，系统会根据返回的值来将该UMG放到对应的分类中。



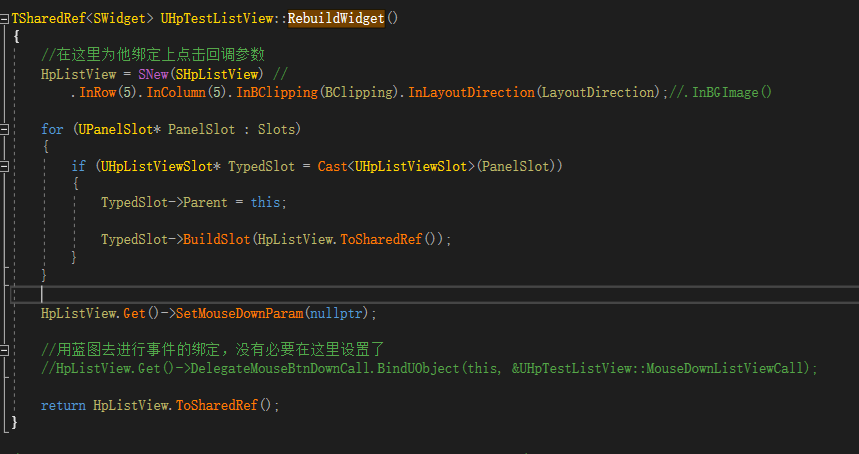
此外重要的几个方法有



ReleaseSlateResources 当清除UMG的时候调用，可以在里面添加一些清除的逻辑

SynchronizeProperties 方法与Slot中的方法类似，当一个UMG创建之后会进行调用，用来同步参数到Slot以及Slate界面中，该函数是所有widget上的数据变动时都会调用的参数，不仅仅是在Detail面板中更新数值，也是在Editor中拉大控件时会调用的函数。也可以手动调用来进行参数的更新。

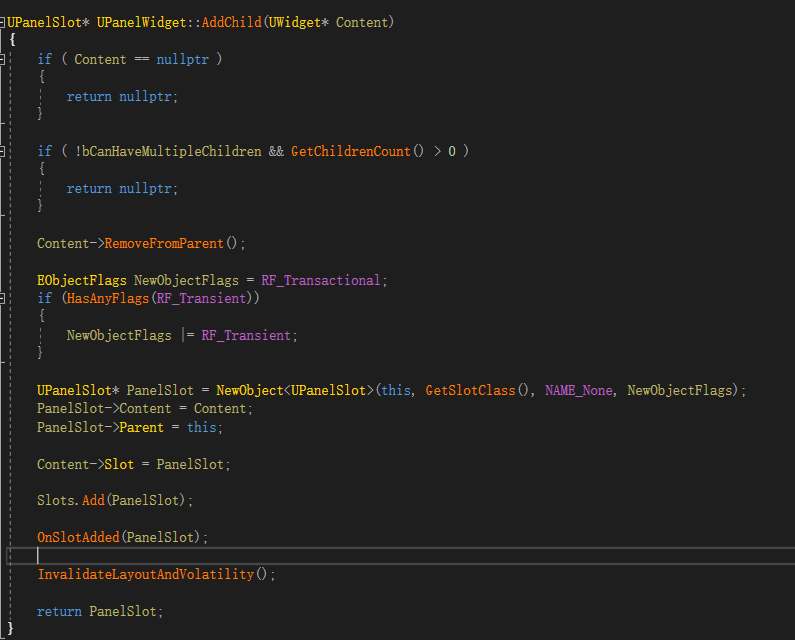
virtual TSharedRef<SWidget> RebuildWidget() override; 是一个比较重要的方法，当移动控件，新添加控件等时候，都会调用RebuildWidget来重新创建UMG，也是在此函数中，与Slate建立连接创建对应的Slate控件以及创建对应的Slot等，当然还可以在其中添加一些自己的逻辑，更新参数等。



virtual void OnSlotAdded(UPanelSlot\* InSlot);

virtual void OnSlotRemoved(UPanelSlot\* InSlot);

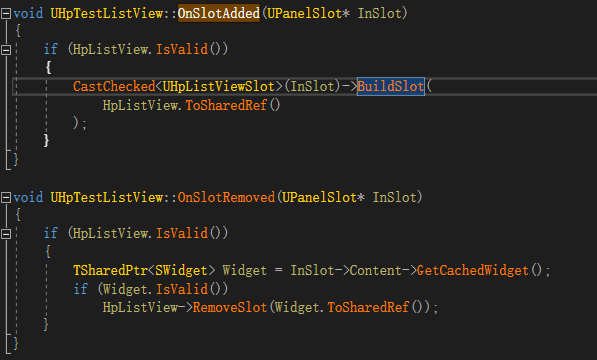
这两个函数，分别为新添加Slot以及移除Slot时调用，添加Slot以及移除Slot时会先调用Slot的AddChild进而调用UMG的这两个函数。



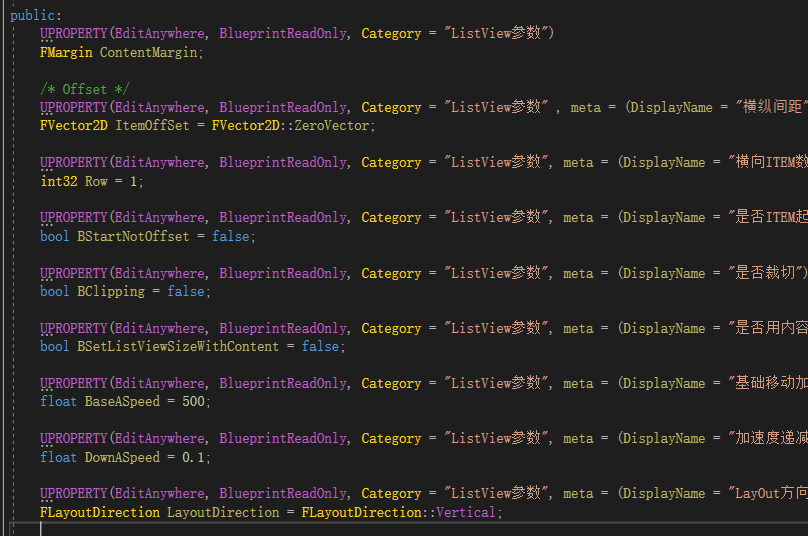
可以看到里面调用了OnSlotAdded，该函数根据自身重载的OnSlotAdded函数去SWidget中New一个Slot并且将本次创建的Slot添加进去。

InvalidateLayoutAndVolatility

该函数使得可能拥有此小Widget的布局缓存中的该Widget无效话，并且会在下一次的绘制中强制刷新该Widget以及其所拥有的的Children。



以上父类必备的方法完成之后，我们就可以根据需求添加自己对应的界面需求参数了



之后在变量更新时，对应去刷新我们的参数同步给Slot再同步给Slate控件即可。

## UMG传递自定义参数/变参相关

### 以蓝图的方式进行调用

虽然C++委托可以使用例如

DECLARE\_DELEGATE\_OneParam( FDelegateMouseButtonDown , FString )

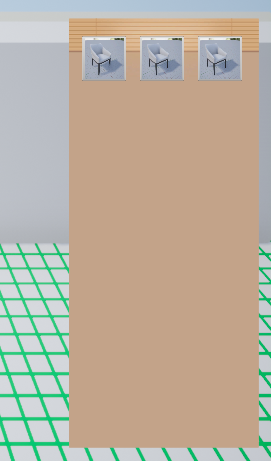
FDelegateMouseButtonDown DelegateMouseButtonDown;

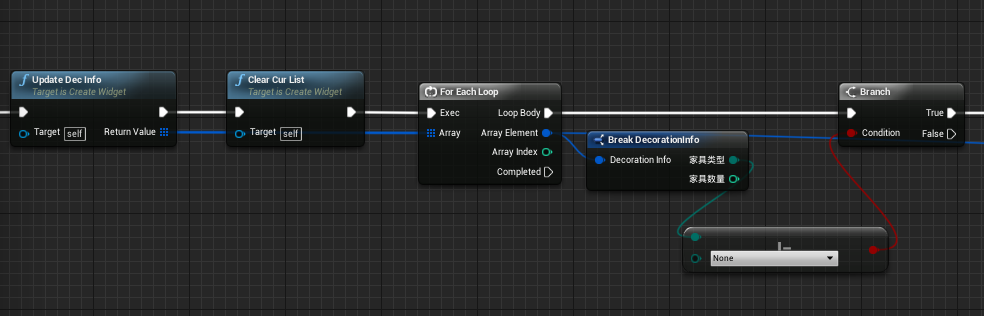
DelegateMouseButtonDown.Execute("1" ， UObject ， MyStruct);

的方式来传递除了生命的变量之外的任意结构体、指针，但是并没有比较好且方便的方式来在绑定委托时将任意类型传入保存，因此参考其他项目的方式，直接使用蓝图绑定进行自定义参数/变参的传递是最方便的。

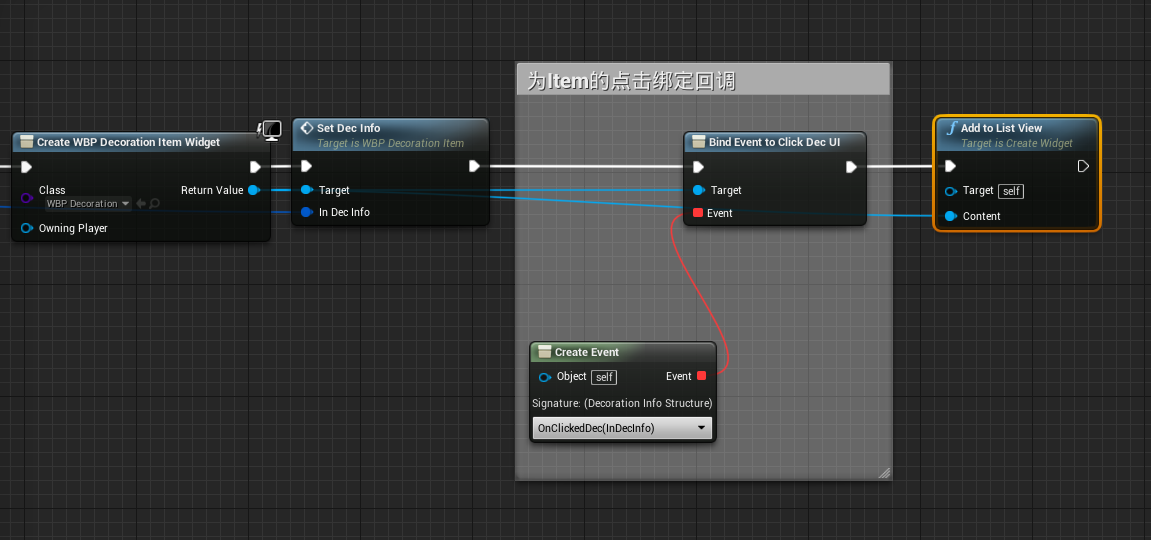
**同时，所有的UI逻辑，建议都在蓝图中完成，C++层只负责提供对应的数据参数方法结构体等暴露给蓝图使用。**

我们以一个简单的类背包功能作为例子，

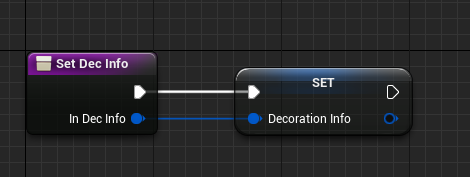
这是一个加载了背包ITEM的UI，我们需要在创建Item的时候，就对每一个Item设置好点击参数，当点击后调用对应的函数并传递不同的参数。



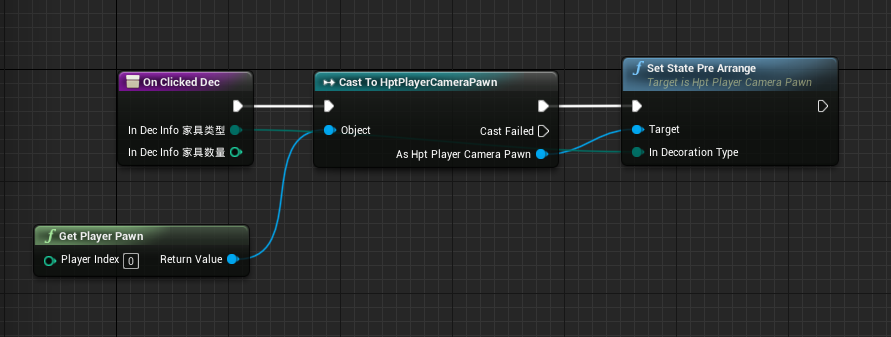
首先我们根据C++提供的数据结构，获取Item的数量及参数，再使用for循环去创建并且设置ITEM



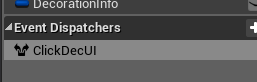
其中SetDecInfo为为Item设置对应的点击参数



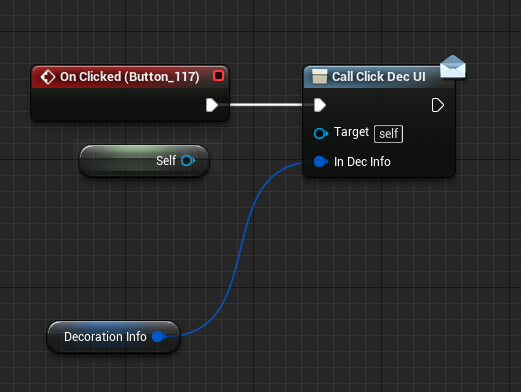
其中的OnClickedDec为点击Item后的回调函数，根据参数不同做不同的处理



而BindToClickDecUI则是在Item中声明的一个委托事件，



在点击Item之后调用绑定的函数并且将Item中的点击参数传出即可



这样便可以用最轻松的方式将自定义参数传出，也大大减少C++代码的复杂度。