SButton 似乎是通过传递点击方式，尝试找到hover的事件

FSlateApplication进行事件分发

OnMouseMove OnMouseEnter等，都继承自SWidget类中的Mouse等事件，所以理论上都可以注册响应鼠标事件！！！

在Swidget中通过

virtual FReply OnMouseMove(const FGeometry& MyGeometry, const FPointerEvent& MouseEvent);

来重写父类事件，该事件应该是创建Slate的时候注册进FSlateApplication后(猜测)，由系统进行分发

TempReply = MouseCaptorWidget.Widget->OnMouseMove(MouseCaptorWidget.Geometry, Event);

FSlateApplication中的分发事件是这一句

Sborder重写一个TreeNode

ChildSlot，是SCompoundWidget中的一个FSimpleSlot对象

如果没有背景图绘制，则该控件不会主动响应点击等事件，但是可以接受子控件FReply::Unhandled()的冒泡,测试下好像线框也不行

用两个参数相减可以获得在widget中的正确点击位置，MyGeometry.GetAbsolutePosition()是该WIDGET在屏幕中的位置，而MouseEvent.GetScreenSpacePosition()是在屏幕中的点击位置

FVector2D CreatePos = MouseEvent.GetScreenSpacePosition();// -M\_Position;

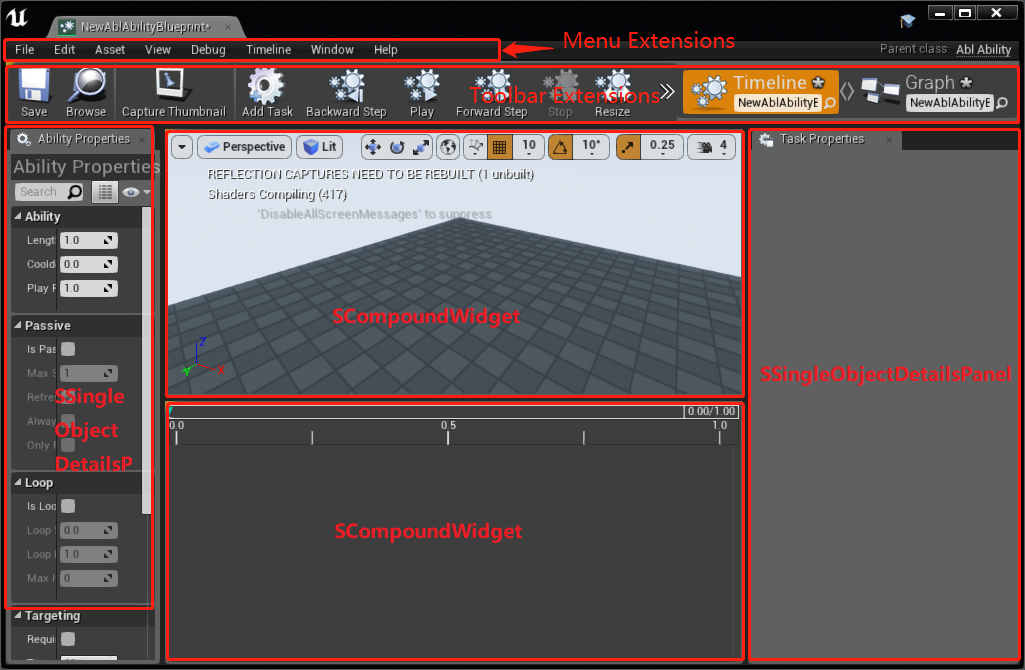
FVector2D absolutePos = MyGeometry.GetAbsolutePosition();

FVector2D ClickSizeInTree = CreatePos - absolutePos;

Paint流程，通过OnPaint开始绘制，在其中调用OnArrangeChildren的 ArrangedChildren.AddWidget 来设置ArrangedChildren的需要绘制的子数量，图形等通过FSlateDrawElement::MakeBox 来添加图形进绘制列表,那么可以尝试单独提出需要绘制成图形的部分！

<https://blog.csdn.net/u012999985/article/details/79244719> 什么是AssetTools

## 新编辑器相关

[](https://arenas0.com/2019/03/14/Extending_the_UE4_Editor_2/Extending_the_UE4_Editor_1-7f77cca4.png)

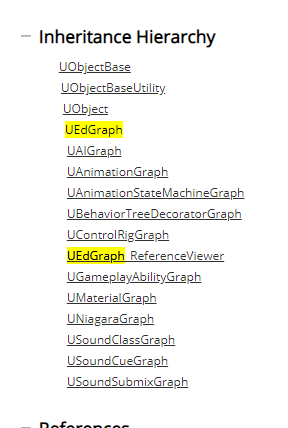
<https://arenas0.com/2019/03/14/Extending_the_UE4_Editor_2/> 图来源

FAssetEditorToolkit : 用于资产编辑的工具包的基类(摘要)

SGraphEditor

Interface and wrapper for GraphEditor widgets.用于GraphEditor小部件的接口和包装器。

图（UEdGraph）



<https://zhuanlan.zhihu.com/p/69067129>

TArray<class UEdGraphNode\*> Nodes：图中的所有节点。

// FDocumentTabFactory

class KISMET\_API FDocumentTabFactory : public FWorkflowTabFactory

{

public:

// Saves the state for this tab

//@TODO: DOCMANAGEMENT: This is very special-cased and isn't a general purpose solution

virtual void SaveState(TSharedPtr<SDockTab> Tab, TSharedPtr<FTabPayload> Payload) const {}

// Called when a tab created from this factory is activated

virtual void OnTabActivated(TSharedPtr<SDockTab> Tab) const {}

## GraphEditor

OnSelectedNodesChanged 注册后的该函数，在Graph中Node被选中时调用，包含selections（TSet<class UObject\*>&）Node列表，将对应的Instance OBJ设置给DetailsView就可以显示对应的属性。

InitAssetEditor（） 之后，就会调用RegisterTabSpawners ，此时就正式将SGraphEditor与我们注册的Tab进行关联并注册！

FAssetEditorToolkit::RegisterTabSpawners 调用该父类函数可以生成一个带有Save的默认Toolbar，如果想要用自己的，自己注册RegisterTabSpawner创建UI就可以了

删除节点是通过FBlueprintEditorUtils::RemoveNode(nullptr, Node, true);

与创建的时候对应

### 模式（UEdGraphSchema）

TSubclassOf<class UEdGraphSchema> Schema：这个图要遵循的schema。

模式是与Graph对应的，约定了当前Graph能创建什么样的节点、两个引脚能否产生连接等等。

EdGraphSchema定义了graph大部分操作，可以继承来修改表现形式，如可修改CreateConnectionDrawingPolicy改变连接线条的样式曲度等

GetGraphContextActions（FGraphContextMenuBuilder& ContextMenuBuilder）点击右键，或者是拖出线来的时候，会调用该函数。

// This context is used when building a list of actions that can be done in the current context

struct FGraphContextMenuBuilder : public FGraphActionMenuBuilder

##

ECanCreateConnectionResponse 两个引脚是否可以链接的标志

CreateAutomaticConversionNodeAndConnections() 通过重写该方法，可以绘制在两个节点连接时的中间的可点击节点，

## FEdGraphSchemaAction

/\*\* This structure represents a context dependent action, with sufficient information for the schema to perform it. \*/

USTRUCT()

struct ENGINE\_API FEdGraphSchemaAction

将 自己继承的类Add进ContextMenuBuilder 中，就可以在每次点击菜单中的一项时，调用

/\*\* Execute this action, given the graph and schema, and possibly a pin that we were dragged from. Returns a node that was created by this action (if any). \*/

virtual UEdGraphNode\* PerformAction(class UEdGraph\* ParentGraph, UEdGraphPin\* FromPin, const FVector2D Location, bool bSelectNewNode = true) { return nullptr; }

一旦点击菜单中的一项，就会调用FEdGraphSchemaAction的函数PerformAction，

因此要实现这个函数，也就是把创建新Node的逻辑写在这里。

这里会用到一个模板类FGraphNodeCreator，所以要重写PerformAction就可以创建自己的Node

/\*\* Struct containing information about what actions occurred on the graph \*/

struct FEdGraphEditAction

**自定义一个自己的Action,暂时只实现PerformAction接口即可：**

const FScopedTransaction Transaction(LOCTEXT("AddNode", "Add Node"));

给编辑器记录一次修改操作！应该是对应的control+z？

FScopedTransaction灵活性比GEditor->BeginTransaction, GEditor->EndTransaction低得多，也就是FScopedTransaction变量声明到退出所在的局部函数的时候被视为“一次修改”。

 DuplicateObject（）复制对象的便利模板

##

FPinConnectionResponse：

这是来自CanCreateConnection的响应，指示连接操作是否合法以及结果如何

在ScenarioEditor中的RestoreStoryGraph中，调用的是

FBlueprintEditorUtils::CreateNewGraph

而该方法内部可以看到，其实也就是NewOjbect了一个UEdGraph并传入响应的参数，所以其实，自己写也是完全可以的！所以有些插件并没有使用

只是返回一个EdGraph ,需要再自己转一遍

* 为UObject创建一个新的有效负载包装

FTabPayload\_UObject::Make()

在BehaviorTree的创建中，与

DocumentManager->OpenDocument(Payload, bNewGraph ? FDocumentTracker::OpenNewDocument : FDocumentTracker::RestorePreviousDocument);

配合，创建了一个SDockTab！TSharedPtr<SDockTab> DocumentTab

/ \* \*

\*处理打开文档和处理标签导航或强制打开新标签

\*

\* @param inpayload payload to use for the tab

\* @param表示文件被打开的原因

\*

\* @返回修改后的标签

\* /

TSharedPtr<SDockTab> OpenDocument(TSharedPtr<FTabPayload> InPayload, EOpenDocumentCause InOpenCause);

/ \* \*

\* GraphEditor小部件的接口和包装。

优雅地处理卸载的GraphEditorModule。

\* /

SGraphEditor : public SCompoundWidget

## FapplicationMode

从行为树的设计上来看，

/\*\* Application mode for main behavior tree editing mode \*/

class FBehaviorTreeEditorApplicationMode : public FApplicationMode

有一个这样的类，用来调用RestoreScenario进行创建Graph

## GraphNode：

帮助对象，以确保正确构造了一个图节点

\*

\*典型的使用模式是:

\* FNodeGraphNodeCreate < NodeType > NodeCreator(图);

\* Node = Node . createnode ();

\* //调用生成节点

\* Node->MemberVar =…

\* NodeCreator.Finalize

FGraphNodeCreator

CreateNode()后还需要调用 Finalize()来确定节点构造完成

UEdGraphNode\* UEdGraph::CreateNode( TSubclassOf<UEdGraphNode> NewNodeClass, bool bFromUI, bool bSelectNewNode )

{

UEdGraphNode\* NewNode = NewObject<UEdGraphNode>(this, NewNodeClass, NAME\_None, RF\_Transactional);

if (HasAnyFlags(RF\_Transient))

{

NewNode->SetFlags(RF\_Transient);

}

AddNode(NewNode, bFromUI, bSelectNewNode );

return NewNode;

}

最终是调用了UEdGraph的CreateNode，时机也就是New了一个Object并且将其加入自身

**Node应该是有两块，一块是SGraphNode用来做Node的Slate显示，一块是**

**UEdGraphNode 应该是用来进行数据存储等(UObject)**

**SGraphNode继承自SNodePanel：**

TArray<UEdGraphPin\*> Pins; 引脚的保存

这个类被设计为包含交互式小部件的面板/画布的基类

\*可由用户选择和移动。它还管理缩放和平移，允许更大

\*用于放置小部件的虚拟空间。

\*

\*用户负责创建小部件(必须从SNode派生)和任何自定义绘图

\*所需的代码。另一个主要限制是每个SNode实例必须有一个惟一的关联UObject\*

\*。

根据关联的UObject来进行绑定

/\*\*

\* Implements an overlay widget.

\*

\* Overlay widgets allow for layering several widgets on top of each other.

\* Each slot of an overlay represents a layer that can contain one widget.

\* The slots will be rendered on top of each other in the order they are declared in code.

\*

\* Usage:

\* SNew(SOverlay)

\* + SOverlay::Slot(SNew(SMyWidget1))

\* + SOverlay::Slot(SNew(SMyWidget2))

\* + SOverlay::Slot(SNew(SMyWidget3))

\*

\* Note that SWidget3 will be drawn on top of SWidget2 and SWidget1.

\*/

class SLATECORE\_API SOverlay 可以实现小部件层层叠加的控件

/\*\* The area where output pins reside \*/

TSharedPtr<SVerticalBox> RightNodeBox;

输出引脚的区域

EVisibility::SelfHitTestInvisible 只是自身不会接受鼠标等事件，但不影响子物体接受（应该是自身FReply::handler了）

CreatePinWidgets()

父类中是通过判断所有的PINS，判断可以链接的Node才会调用CreateStandardPinWidget（CurPin）进行真正的Pins创建，自己写的话其实可以跳过这个步奏。。直接创建

调用的FNodeFactory的CreatePinWidget，其实最终也是通过SNew去创建一个Slate，所以功能不多的时候可以自己写。

父类中的AddPin ，就是将Pin添加到LeftNodeBox或者RightNodeBox中，通过

EEdGraphPinDirection类型。

## GraphPin

/\*\*

\* Represents a pin on a node in the GraphEditor

\*/

class GRAPHEDITOR\_API SGraphPin : public SBorder

Pin 是一个Sborder

/\*\* The GraphPin that this widget represents. \*/

class UEdGraphPin\* GraphPinObj;

Pin的Slate实体，也是一个UEdGraphPin Object的。

Pin是有OwnerNode的，可以获取Schema

FReply SGraphPin::OnPinMouseDown( const FGeometry& SenderGeometry, const FPointerEvent& MouseEvent )

{

bIsMovingLinks = false;

if (MouseEvent.GetEffectingButton() == EKeys::LeftMouseButton)

{

if (IsEditingEnabled())

{

if (MouseEvent.IsAltDown())

{

// Alt-Left clicking will break all existing connections to a pin

const UEdGraphSchema\* Schema = GraphPinObj->GetSchema();

Schema->BreakPinLinks(\*GraphPinObj, true);

return FReply::Handled();

}

从代码上可以看出，默认的鼠标点击事件，是当AltDown的时候，会断开Pin的链接。

## FGraphPanelNodeFactory , FGraphPanelPinFactory , FGraphPanelPinConnectionFactory

要在模块开始的时候，去注册三个对应的Factory，在FGraphNodeCreator CreateNode的时候，应该就会调用对应的！

也可以。。直接使用Node的CreateVisualWidget方法。，。不过测试过后好像不行,调不到SNew

## FConnectionDrawingPolicy

如果自定义了Pin，原始的链接方式应该会找不到对应的连线点，所以也就链接不上了，那么这个时候！就通过注册我们自己的连线就好

//其实只是设置了一下颜色以及Pin等参数并且通过ApplyHoverDeemphasis设置到成员函数中

void FScenarioConnectionDrawingFactory::DetermineWiringStyle(UEdGraphPin\* OutputPin, UEdGraphPin\* InputPin, FConnectionParams& Params)

DrawSplineWithArrow 该方法中，调用的就是FSlateDrawElement::MakeBox进行的连线，也可以用其他方式进行连线，比如自定义形状啥的

## SGraphPanel

在这个类里面，才是调用SNode啊Pin和DrawLine的factory创建显示Slate的地方，绘制的方式当然！是在OnPaint里面进行的调用绘制！！

## 问题：

目前总是在这一步抛出异常

/\*\*

\* Find a pair with the specified key, removes it from the map, and returns the value part of the pair.

\*

\* If no pair was found, an exception is thrown.

\*

\* @param Key the key to search for

\* @return whether or not the key was found

\*/

FORCEINLINE ValueType FindAndRemoveChecked(KeyConstPointerType Key)

这个类提供了垃圾收集的通用注册

\* non-UObject类。它是一个需要您实现的抽象基类

AddReferencedObjects()方法

class COREUOBJECT\_API FGCObject

// Create the layout of our custom asset editor

TSharedRef<FTabManager::FLayout> StandaloneDefaultLayout = FTabManager::NewLayout("Standalone\_CustomAssetEditor\_Layout\_v1")

->AddArea(

FTabManager::NewPrimaryArea()->SetOrientation(Orient\_Vertical)

->Split(

FTabManager::NewStack()

->SetSizeCoefficient(0.1f)

->SetHideTabWell(true)

->AddTab(GetToolbarTabId(),ETabState::OpenedTab)

)

// Split the tab and pass the tab id to the tab spawner

->Split(

FTabManager::NewSplitter()

->Split

(

FTabManager::NewStack()

->AddTab(PropertiesTabId, ETabState::OpenedTab)

)

)

);

创建界面需要的layout

不使用BehaviorTree的创建系统而直接自己写界面比较方便！！成功

## 流程

AssetTypeAction打开AssetEditorModule并且调用其中的CreateXXXEditor,

AssetEditorModule调用类似

TSharedRef< FScenarioEditor > NewCustomAssetEditor(new FScenarioEditor());

NewCustomAssetEditor->InitScenarioEditor(Mode, InitToolkitHost, CustomAsset);

return NewCustomAssetEditor;

这样的方式来创建一个Editor；

Editor中通过使用DocumentManager（也有不同方式）创建一个DocumentTabFactory并且注册：

FScenarioGraphEditorSummoner 通过这个创建器（自定名）来创建激活这个TAB­，并且在生成创建时调用

/\* 创建GraphEditor! \*/

TSharedRef<SGraphEditor> FScenarioEditor::CreateGraphEditorWidget(UEdGraph\* InGraph)

{

check(InGraph != NULL);

return SNew(SGraphEditor);

}

return SNew(SGraphEditor)

.AdditionalCommands(GraphEditorCommands)

.IsEditable(this, &FBehaviorTreeEditor::InEditingMode, bGraphIsEditable)

.Appearance(this, &FBehaviorTreeEditor::GetGraphAppearance)

.TitleBar(TitleBarWidget)

.GraphToEdit(InGraph)

.GraphEvents(InEvents);

应该是在return里面去设置Graph之类的东西

之后调用

InitAssetEditor(Mode, InitToolkitHost, TEXT("ScenarioEditorTreeEditorApp"), DummyLayout,true,true, CustomAsset);

AddApplicationMode(ScenarioMode, MakeShareable(newFScenarioApplicationMode(SharedThis(this))));

进行注册！

FScenarioApplicationMode中，会通过构造函数，以及RegisterTabFactories PostActivateMode等函数，调用FScenarioEditor的RegisterTabSpawners 以及RestoreStoryGraph（）等函数，

同事FApplicationMode::RegisterTabFactories（）进行注册！ 这里与自定义简单资源就对上了。

之后调用Schema中的CreateDefaultNodesForGraph（）去创建默认节点，使用

GetGraphContextActions函数去进行右键点击的菜单设置以及拉出线后出现的菜单设置。

FGraphNodeCreator<UScenarioNodeBase> NodeCreater(Graph);

UScenarioNodeBase\* Nodebase = NodeCreater.CreateNode();

NodeCreater.Finalize();

Finalize()后会调用Node的AllocateDefaultPins函数去创建引脚。

自定义一个SchemaAction，去进行Node的创建。

<https://zhuanlan.zhihu.com/p/83588300>这个总结的不错

（以上为Behaviortree等官方很多使用，但其实并不是一个特别好用的流程， 推荐使用自己通过TabManager进行的整个Slate创建）。

自己创建的话，ScenarioApplicationMode与ScenarioGraphEditorSummoner 都不需要了

<https://neil3d.gitee.io/unreal/bp_dynamic_pin.html> 自定义蓝图节点文章

## PIN

在EDGraphPin中有字段

/\*\* Set of pins that we are linked to \*/

TArray<UEdGraphPin\*> LinkedTo;

/\*\* The pins created when a pin is split and hidden \*/

TArray<UEdGraphPin\*> SubPins; 应该是中间有穿插sub节点时的pin

/\*\* The pin that was split and generated this pin \*/

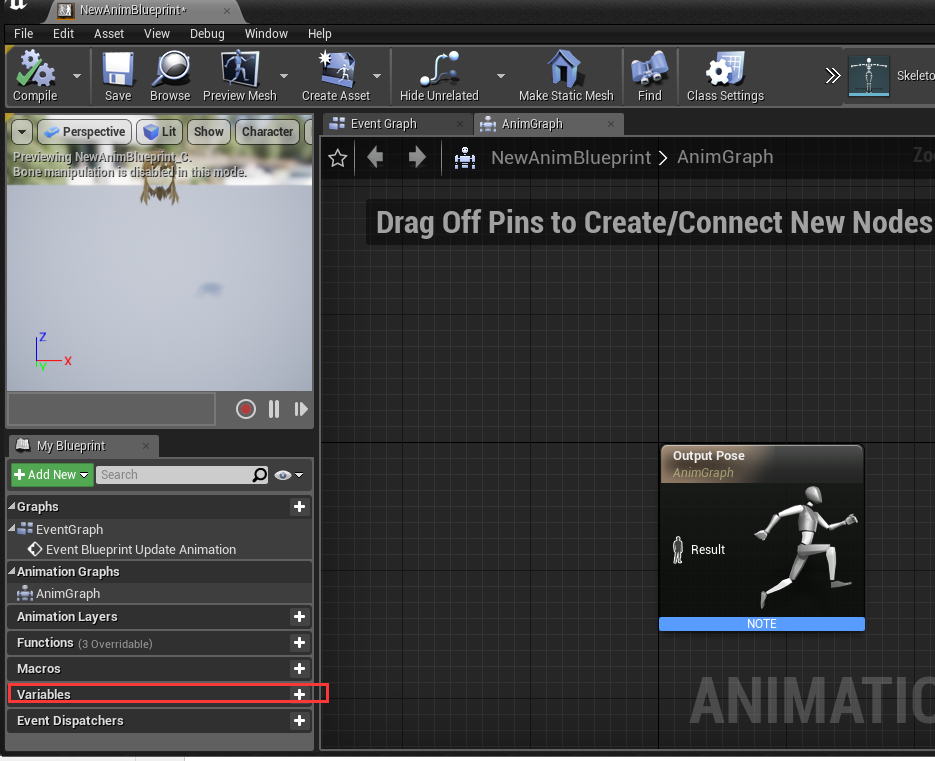
UEdGraphPin\* ParentPin;

/\*\* Pin that this pin uses for passing through reference connection \*/

UEdGraphPin\* ReferencePassThroughConnection;

## 尝试，自定义Param相关

FReply SMyBlueprint::OnAddButtonClickedOnSection(int32 InSectionID) 是



这里调用的函数

void FBlueprintEditor::OnAddNewVariable() 最终分发到这里来，BlueprintEditor.cpp

OnActionSelected 选择一个参数时调用 SMyBlueprint.cpp

FBPVariableDescription

指示生成的类中的变量的结构

## 自定义detail相关

Details继承IPropertyTypeCustomization，

const TSharedRef<IPropertyHandle> PropertyHandle

handle可以通过GetOuterObjects来获取外部的Object？

EventTypePropertyHandle = StructPropertyHandle->GetChildHandle(GET\_MEMBER\_NAME\_CHECKED(FDlgEvent, EventType));

EventTypePropertyHandle->SetOnPropertyValueChanged(

FSimpleDelegate::CreateSP(this, &Self::OnEventTypeChanged, true));

可以设置属性变化时的回调

PropertyHandle是可以获取GetPropertyDisplayName 的,类似还有很多方法。

这是定制属性类型，但是需求如果是要定制Detail面板，则需要使用的是IDetailCustomization

## SLATE转UMG

UWidget继承的UMG\_API UVisual 中，有ReleaseSlateResources方法

The base class for elements in UMG: slots and widgets.

分为4部：

先制作Slate，再为Slate添加SlateStyle

在通过USlateWidgetStyleContainerBase 为样式加上一个基础容器，并且反射给UMG编辑器。

最后需要一个继承自UWidget等基类的UMG类

自定义style

SLATE\_ATTRIBUTE 属性可以是值，也可以是函数

SLATE\_ARGUMENT 就只能是值

Slot相关 这三个函数就是用来编辑器模式下拖动小控件等时的回调了

/\* 用键盘左右移动时的回调 \*/

virtual bool NudgeByDesigner(const FVector2D& NudgeDirection, const TOptional<int32>& GridSnapSize) override;

/\* 将小部件拖动到Canvas上时的回调 \*/

virtual bool DragDropPreviewByDesigner(const FVector2D& LocalCursorPosition, const TOptional<int32>& XGridSnapSize, const TOptional<int32>& YGridSnapSize) override;

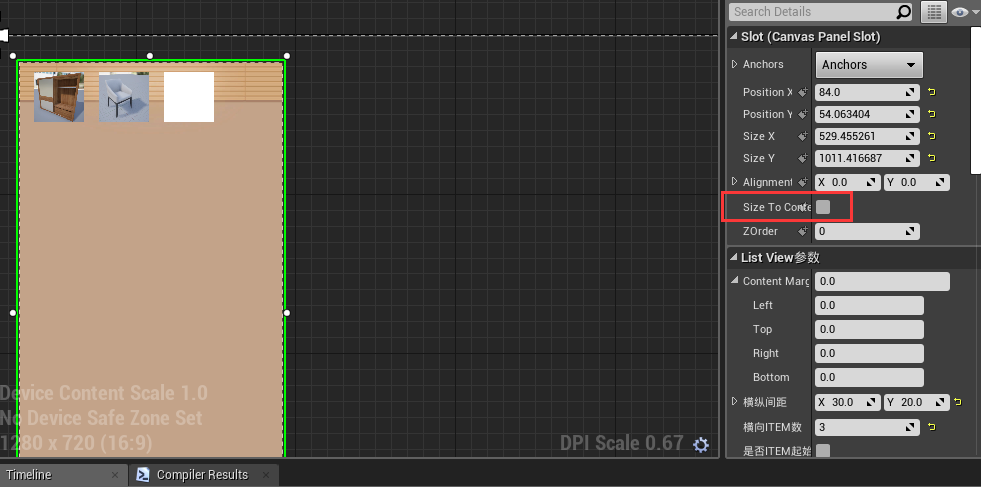
/\*\*

\* Called by the designer when a design-time widget needs to have changes to its associated template synchronized.

\*\*/

virtual void SynchronizeFromTemplate(const UPanelSlot\* const TemplateSlot) {};

在Slate的ComputeDesiredSize 中返回的大小，会使用在一些地方，目前已知的有



如果这里勾选了父节点Slot的SizeToContent，则会取ComputeDisiredSize来计算当前的绘制大小。

在所有的Widget中，有一个变量

/\*\*

\* The parent slot of the UWidget. Allows us to easily inline edit the layout controlling this widget.

\*/

UPROPERTY(Instanced, TextExportTransient, EditAnywhere, BlueprintReadOnly, Category=Layout, meta=(ShowOnlyInnerProperties))

UPanelSlot\* Slot;

该变量是自身所处的父类的Slot，所以编辑这个就可以获取大小等了。

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RebuildWidget函数。

PanelSlot中的这个函数，应该是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设置进去了，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。

void UHpListViewSlot::BuildSlot(TSharedRef<SHpListView> InHpListView)

{

Slot = &InHpListView->AddSlot()

[

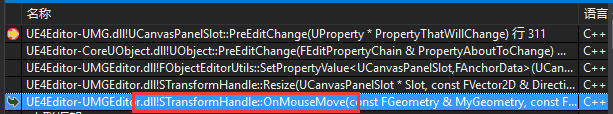
Content == NULL?SNullWidget::NullWidget:Content->TakeWidget()

];

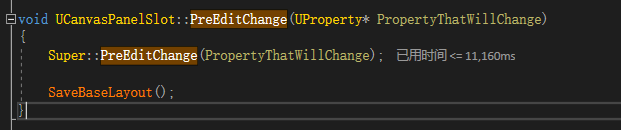
SynchronizeProperties();

}

直接在编辑器中拖动控件大小，是通过设置参数的改变，调用

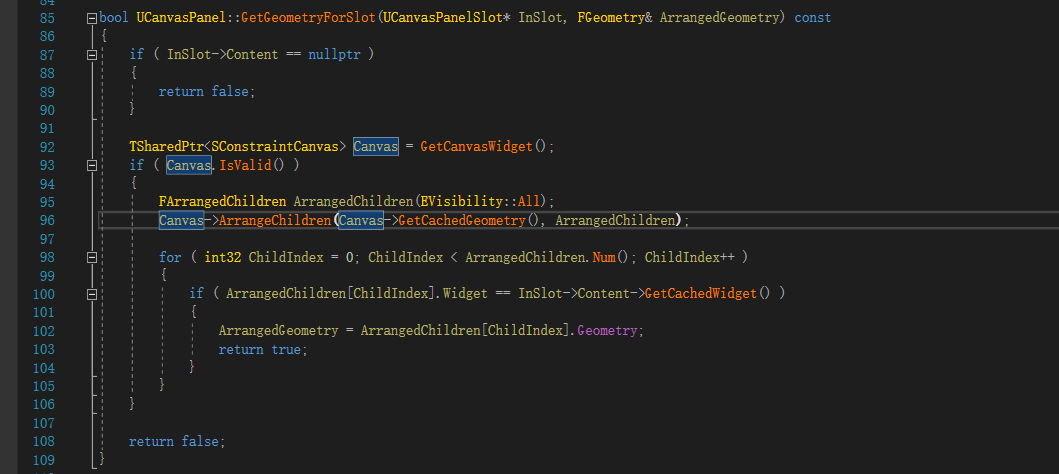


CanvasPanelSlot中继承自UObject的PreEditChange函数，在函数中调用



SaveBaseLayout

进而调用Slate中的ArrangeChildren并且使用新的Geometry来设置大小



Swidget的OnMouseleave的坑，当调用OnMouseButtonDown时，会直接调用OnMouseLeave！

// If the cursor is not currently over the widget on which the drag

// operation started (which should only be the case due to cursor

// movement), the remainder of events are handled in

// RoutePointerMoveEvent(), which will immediately call OnDragLeave()

// on the widget that started the drag followed by OnDragEnter() on

// the current widget. Thus, using the letters above, and assuming

// all of the widgets are different, the sequence should end up being:

//

// 1. B - OnDragDetected (processing reply with drag content)

// 1.1. A - OnMouseLeave

// 1.2. B - OnDragEnter

// 2. B - OnDragLeave

// 3. C - OnDragEnter

// 4. C - OnDragOver

在SlateApplication.cpp中的FSlateApplication::ProcessReply有说明

但是如果OnMouseButtonDown，Handled本次事件，则不会触发Leave了

Swidget的Tick可以用来处理拖动后鼠标移开问题？

OnFocusReceived 和 OnFocusLost 应该是只有在Widget正式被代码设置Focus的时候才会调用，因为不会响应鼠标的调用

### 设置Slot的一些移动拖动拉动等记录:

/\*\*

\* Called by the designer to "nudge" a widget in a direction. Returns true if the nudge had any effect, false otherwise.

\*\*/

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

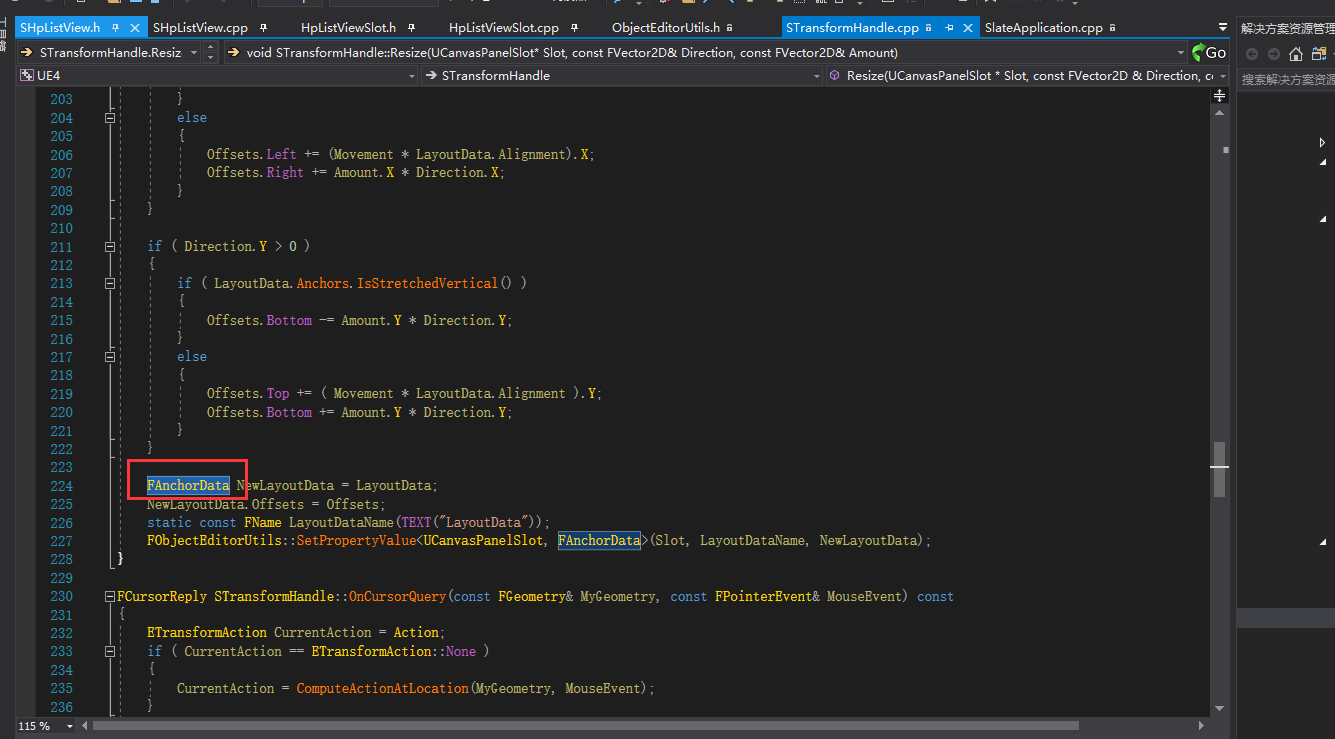
/\*\*

\* Called by the designer when a design-time widget is dragged. Returns true if the drag had any effect, false otherwise.

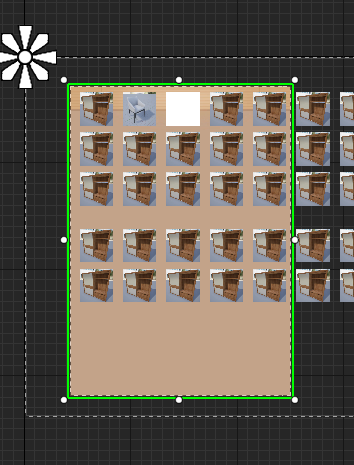
\*\*/

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。



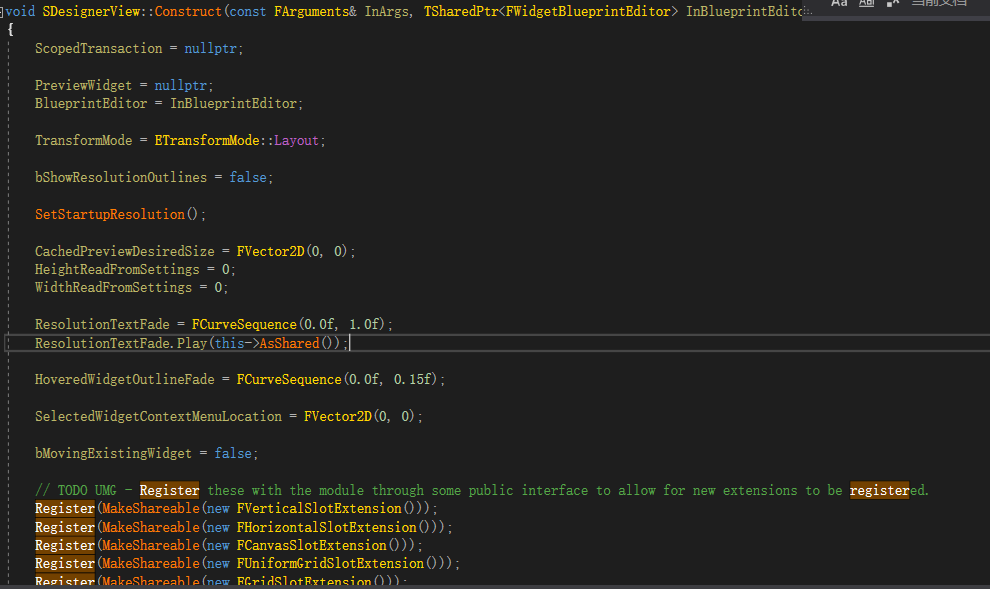
点击，某些UMG控件可以看到会有

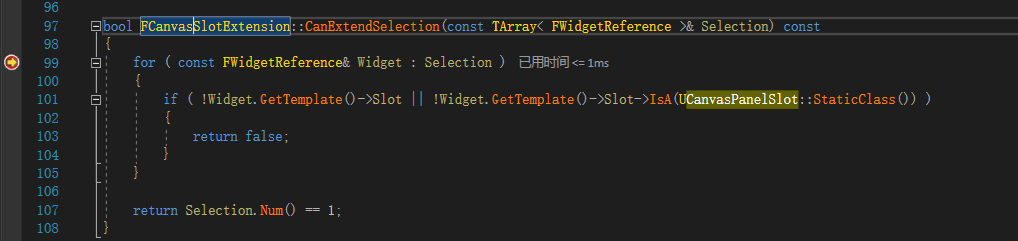


类似于这样的可拖动缩放设置锚点等的标识，是调用的SDesignerView,

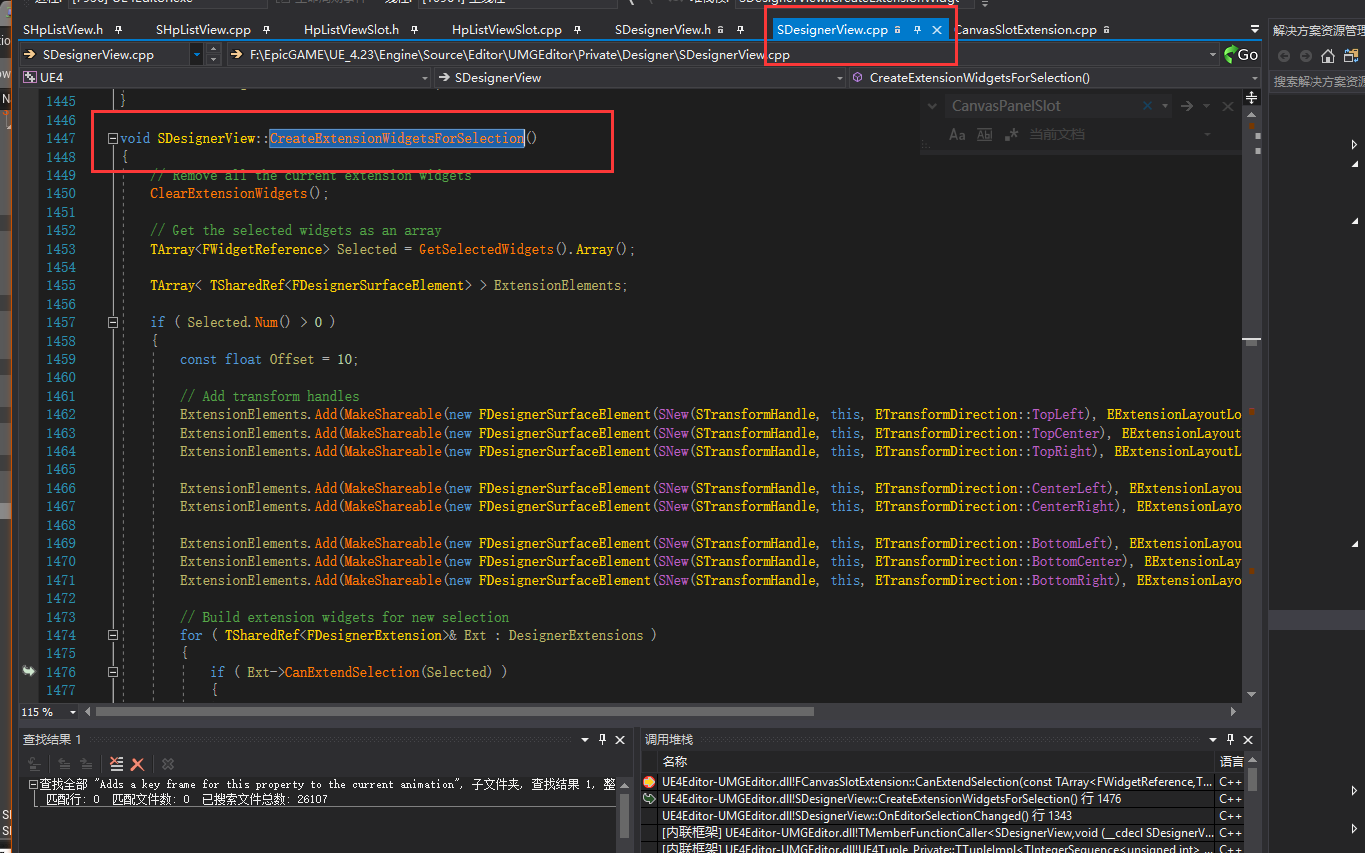
SDesignerView 通过代码可以看到，在里面注册了一些可以扩展的UMG控件，由该函数可以看到，循环注册的DesignerExtensions来寻找可以扩展的UMG,

在构造函数中可以看到只注册了这几种，所以如果！把自己写的Slot注册进去就行了，但是得该地层，算了，还可以直接继承这几个Slot！

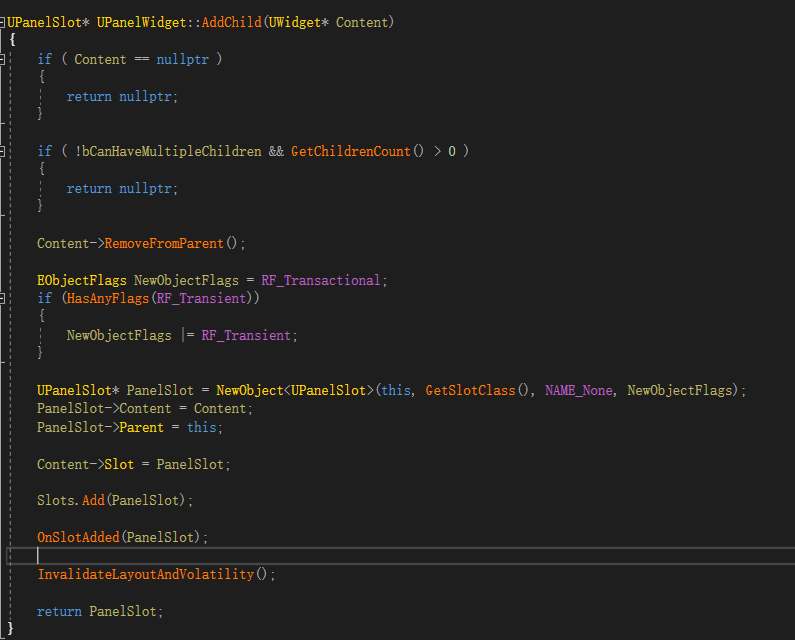
其中就有CanvasSlot，如果点击UMG，会调用



去判断是否为UCanvasPanelSlot，如果是，则会显示扩展



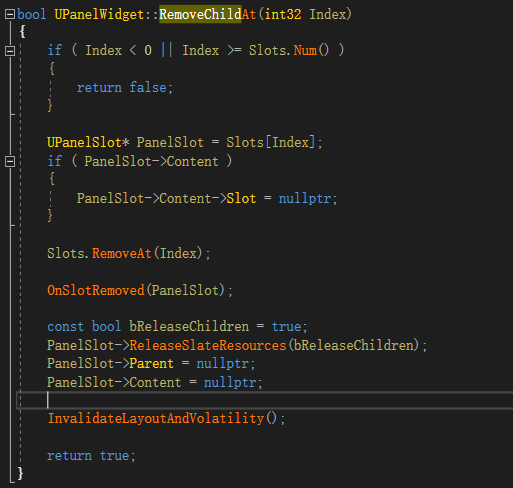
将Widget添加至Slot



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。



删除部分。

/\*\*

\* Invalidates the widget from the view of a layout caching widget that may own this widget.

\* will force the owning widget to redraw and cache children on the next paint pass.

\*/

void Invalidate(EInvalidateWidget InvalidateReason);