**UE4 UMG**自定义控件开发

UE4下UMG提供了一些基本控件，满足简单的UI开发需求，对于一些复杂的界面，需要组合这些基本的控件，或者需要开发自定义控件。本文主要介绍UMG/Slate的开发自定义控件的基本方法。

1.UMG与Slate的关系

Slate是UE提供一套UI Framework，支持跨平台。它既可以用来做应用程序的UI(如UE4 Editor)、工具的UI，也可以做游戏中的UI。Slate通过宏和重载C++运算符实现声明式语法, 所有的布局和组件创建只能用C++实现。

// Add a new section for static meshes

ContextualEditingWidget->AddSlot()

.Padding( 2.0f )

[

SNew( SDetailSection )

.SectionName("StaticMeshSection")

.SectionTitle( LOCTEXT("StaticMeshSection", "Static Mesh").ToString() )

.Content()

[

SNew( SVerticalBox )

+ SVerticalBox::Slot()

.Padding( 3.0f, 1.0f )

[

SNew( SHorizontalBox )

+ SHorizontalBox::Slot()

.Padding( 2.0f )

[

SNew( SComboButton )

.ButtonContent()

[

SNew( STextBlock )

.Text( LOCTEXT("BlockingVolumeMenu", "Create Blocking Volume") )

.Font( FontInfo )

]

.MenuContent()

[

BlockingVolumeBuilder.MakeWidget()

]

]

]

]

];

而UMG基于UObject体系，对Slate进行封装而来的UI框架。UMG可以很好的和UE的GameFramework以及蓝图进行交互，并且提供一个可视化的编辑器，Slate缺少可视化编辑器。

在UMG的控件类中，我们可以找到一个Slate控件的智能指针，如：在UMG的UButton中可以找一个类型为SButton的智能指针成员变量（TSharedPtr<SButton> MyButton）。

下面将以按钮为例来分析Slate与UMG联系。

在UButton的源文件中，我们可以找到一个智能指针成员：TSharedPtr<SButton> MyButton。这个智能指针是实现UButton所有功能的关键。UMG控件在创建和重建时都会调用RebuildWidget()，在RebuildWidget()函数中，会对这个Slate类的实例进行初始化。

TSharedRef<SWidget> UButton::RebuildWidget()

{

MyButton = SNew(SButton)

.OnClicked(BIND\_UOBJECT\_DELEGATE(FOnClicked, SlateHandleClicked))

.OnPressed(BIND\_UOBJECT\_DELEGATE(FSimpleDelegate, SlateHandlePressed))

.OnReleased(BIND\_UOBJECT\_DELEGATE(FSimpleDelegate, SlateHandleReleased))

.OnHovered\_UObject( this, &ThisClass::SlateHandleHovered )

.OnUnhovered\_UObject( this, &ThisClass::SlateHandleUnhovered )

.ButtonStyle(&WidgetStyle)

.ClickMethod(ClickMethod)

.TouchMethod(TouchMethod)

.PressMethod(PressMethod)

.IsFocusable(IsFocusable)

;

if ( GetChildrenCount() > 0 )

{

Cast<UButtonSlot>(GetContentSlot())->BuildSlot(MyButton.ToSharedRef());

}

return MyButton.ToSharedRef();

}

在UButton的RebuildWidget()函数中，将UButton的样式设置和交互设置一股脑的传递给真正的按钮对象SButton。在其他的函数关键函数中，都是调用SButton的相关函数，进行真正的处理。另外一面，由于UMG控件都是继承UObject的对象，能直接和UE的game framework进行交互。

2.自定义控件

我们要实现自定义控件就有两种选择：利用现有控件组合成一个新控件和从零开始完全实现一种新控件。UMG的ListView、ScrollView、Table等复杂控件就是利用前一种方法实现的，因此前一种方法就不再这里赘述了。这里主要讲下第二种方法。

基于上述UMG和Slate的关系，我们要实现一个新控件，就需要实现两个类：一个是继承UWidget体系的UMG控件，另一个是继承SWidget体系的Slate控件类。如果我们实现的控件还能包含子控件，则还需要实现两个Slot类：继承UPanelSlot的Slot类和继承TSlotBase<T>模版的FSlot类。

下面我们以实现一个Toggle。这个Toggle主要功能有：

* 能发送状态切换事件给使用者
* 对于每个状态能设置复杂表现，而不是简单设置一张图片。

首先创建一个c++的UE工程，然后创建两个名为UMyToggle和SMyToggle的类。也可以在即有的UE工程中直接创建这两个类。UMyToggle是UPanelWidget的子类，SMyToggle是SPanel的子类。

2.1 UMyToggle

我们来看下UMyToggle类的声明。

#pragma once

#include "CoreMinimal.h"

#include "UObject/ObjectMacros.h"

#include "Components/PanelWidget.h"

#include "SlateCore/Public/Styling/SlateTypes.h"

#include "SMyToggle.h"

#include "MyToggle.generated.h"

class SMyToggle;

class UMyToggleSlot;

class SWidget;

DECLARE\_DYNAMIC\_MULTICAST\_DELEGATE\_TwoParams(FOnToggleStateChanged, ECheckBoxState, LastState, ECheckBoxState, **NewState**);

/\*\*

\*

\*/

UCLASS()

class UMGEXTENTIONSAMPLE\_API UMyToggle : public UPanelWidget

{

GENERATED\_UCLASS\_BODY()

public:

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category="Appearance")

ECheckBoxState CheckedState;

UPROPERTY()

FGetCheckBoxState CheckedStateDelegate;

UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Interaction")

bool IsFocusable;

UPROPERTY(BlueprintAssignable, Category = "Toggle|Event")

FOnToggleStateChanged OnToggleCheckStateChanged;

public:

// Begin UVisual Interface

virtual void ReleaseSlateResources(bool **bReleaseChildren**) override;

// End UVisual Interface

// Begin UWidget

virtual void SynchronizeProperties() override;

#if WITH\_EDITOR

virtual const FText GetPaletteCategory() override;

#endif

// End UWidget

TSharedPtr<SMyToggle> GetToggleWidget()const;

bool GetGeometryForSlot(UMyToggleSlot\* **InSlot**, FGeometry& **ArrangedGeometry**) const;

protected:

// Begin UWidget

virtual TSharedRef<SWidget> RebuildWidget() override;

virtual void OnWidgetRebuilt() override;

// End UWidget

// Begin UPanelWidget

virtual UClass\* GetSlotClass() const override;

virtual void OnSlotAdded(UPanelSlot\* **InSlot**) override;

virtual void OnSlotRemoved(UPanelSlot\* **InSlot**) override;

// End UPanelWidget

void SlateOnToggleCheckeStateChanged(ECheckBoxState **NewState**);

protected:

TSharedPtr<SMyToggle> MyToggle;

PROPERTY\_BINDING\_IMPLEMENTATION(ECheckBoxState, CheckedState)

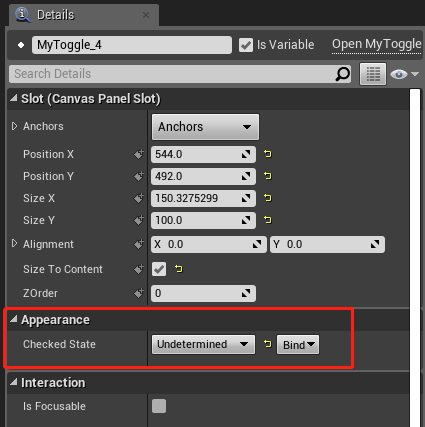
};

2.1.1 成员变量

* CheckState表示toggle的三种状态：未选中态，选中态，悬而未决态。

UPROPERTY是UObject体系中重要的标记，用于UE自动生成代码，可以参照UE文档。它有很多参数，这里针对我们的主题，主要讲下Category。

Category：用于标记这个属性所属类别，同时也定义改属性在Detail面板中显示在哪个分栏中。例如我们的toggle里面的CheckState这个属性，将在显示在MyToggle控件的Detail面板中的Appearance分栏中。



* CheckedStateDelegate这个delegate用于实现在蓝图里面绑定逻辑到这个变量上；在编辑器Detail面板中，我们可以在CheckedState后面看到一个Bind按钮。



点击这个按钮，在蓝图里面可以生成一个函数，在这个函数实现里面可以实现我们自己的需求。

* OnToggleCheckStateChanged这个是Toggle的状态变化的回调delegate，可以将我们处理Toggle状态变化的函数绑定到这个delegate上。

2.1.2 重要成员函数

UMG是对Slate的封装的， Slate控件是UMG的核心。UMG对Slate控件的创建、更新以及销毁主要通过以下成员函数实现：

* 创建：TSharedRef<SWidget> RebuildWidget()

在RebuildWidget()中，创建Slate控件，并进行初始化，控件交互回调的绑定，子Slot的构建（子控件的挂接）。在我们的toggle控件中，我们进行了CheckedState属性和状态切换事件的绑定，以及子Slot的构建。

TSharedRef<SWidget> UMyToggle::RebuildWidget()

{

MyToggle = SNew(SMyToggle)

.IsToggleChecked(CheckedState)

.IsFocusable(IsFocusable)

.OnToggleCheckStateChanged(BIND\_UOBJECT\_DELEGATE(FOnToggleCheckStateChanged, SlateOnToggleCheckeStateChanged));

for (UPanelSlot\* **slot** : Slots)

{

if (UMyToggleSlot\* **ToggleSlot** = Cast<UMyToggleSlot>(**slot**))

{

**ToggleSlot**->Parent = this;

**ToggleSlot**->BuildSlot(MyToggle.ToSharedRef());

}

}

return MyToggle.ToSharedRef();

}

当界面蓝图UserWidget对象通过AddToScreen()添加到屏幕上时， UWidget::TakeWidget\_Private()成员函数将调用UserWidget对象的RebuildWidget()，UserWidge对象的WidgetTree通过RootWidget递归调用TakeWidget()构建整个控件树。

当界面蓝图UserWidget对象通过UPanelWidget::AddChild()添加到各个布局控件（如CanvasPanel、GridPanel、HorizontalBox等）时，布局控件都实现了OnSlotAdded()这个函数，在OnSlotAdded函数中构建子Slot，同时调用添加的UserWidget对象的TakeWidget()进行构建子控件树。

void UMyToggle::OnSlotAdded(UPanelSlot\* **InSlot**)

{

if (MyToggle.IsValid())

{

CastChecked<UMyToggleSlot>(**InSlot**)->BuildSlot(MyToggle.ToSharedRef());

}

}

void UMyToggleSlot::BuildSlot(TSharedRef<SMyToggle> **Toggle**)

{

Slot = &**Toggle**->AddSlot()

[

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

];

SynchronizeProperties();

}

* 更新：void SynchronizeProperties()以及一些成员变量的Setter。

SynchronizeProperties同步UMG与Slate控件的属性，在RebuildWidget()后会执行一次。

成员变量的Setter会直接调用Slate控件相应的接口，进行更新数据以及状态。

* 销毁/释放：void ReleaseSlateResources(bool **bReleaseChildren**)

在ReleaseSlateResources函数中释放当前控件对应Slate控件的引用，并递归调用子控件的ReleaseSlateResources()。

其他的成员函数：

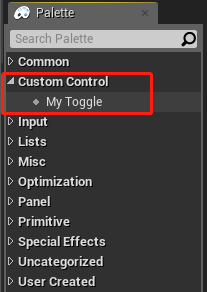
* GetPaletteCategory():编辑器中将调用这个函数来获取控件在UMG编辑器Palette中所属类别。

const FText UMyToggle::GetPaletteCategory()

{

return LOCTEXT("Custom Control", "Custom Control");

}



* GetSlotClass():控件支持的Slot类，在UPanelWidget:AddChild()中用于构建子Slot。

2.2 **SMyToggle**

由于SMyToggle继承自SPanel，而SPanel继承自SWidget。SWidget类中有以下几个纯虚成员函数：

* virtual FVector2D ComputeDesiredSize(float **LayoutScaleMultiplier**) const = 0;

这个函数用于计算控件当前状态最理想的尺寸。通常情况下，我们只需要计算当前状态包含所有可见子控件的最大边界框大小就可以了。如果有特殊需求，可以根据需求在这里进行计算即可。

* virtual FChildren\* GetChildren() = 0;

函数名已表明了它的作用，如果控件是一个布局控件，那么这个是获取子控件的接口。

* virtual int32 OnPaint(const FPaintArgs& **Args**, const FGeometry& **AllottedGeometry**, const FSlateRect& **MyCullingRect**, FSlateWindowElementList& **OutDrawElements**, int32 **LayerId**, const FWidgetStyle& **InWidgetStyle**, bool **bParentEnabled**) const = 0;

这个函数递归调用进行拼装控件需要绘制的Geometry信息。组装时，可以根据SWidget::ArrangeChildren()得到的列表或者自定义的列表进行可见性和裁切状态判断子控件是否需要绘制。

* virtual void OnArrangeChildren(const FGeometry& **AllottedGeometry**, FArrangedChildren& **ArrangedChildren**) const = 0;

当我们需要对子控件进行调整布局、动态过滤某些子控件时，就可以在这个函数里面实现这些功能。在我们的togg中，需要根据CheckedState来过滤掉一些子控件，所以在OnArrangeChildren()中进行了状态判断。

在SPanel中并没有全部实现，在SMyToggle必须要实现这几个函数，才能实现我们的需求。

另外，我们的toggle需要跟玩家交互，所以我们还必须实现我们的交互事件处理函数：

virtual bool SupportsKeyboardFocus() const override;

virtual FReply OnKeyUp(const FGeometry& **MyGeometry**, const FKeyEvent& **InKeyEvent**) override;

virtual FReply OnMouseButtonDown(const FGeometry& **MyGeometry**, const FPointerEvent& **MouseEvent**) override;

virtual FReply OnMouseButtonDoubleClick(const FGeometry& **InMyGeometry**, const FPointerEvent& **InMouseEvent**) override;

virtual FReply OnMouseButtonUp(const FGeometry& **MyGeometry**, const FPointerEvent& **MouseEvent**) override;

virtual void OnMouseEnter(const FGeometry& **MyGeometry**, const FPointerEvent& **MouseEvent**) override;

virtual void OnMouseLeave(const FPointerEvent& **MouseEvent**) override;

virtual bool IsInteractable() const override;

这里要注意一下，UE将移动设备上的点击/移动操作和鼠标的操作封装为了PointerEvent，处理的回调函数都是使用MouseXXX，而SWidget里面的TouchXXX是针对TouchPad的。

至此，我们的toggle已实现，虽然没有实现checkbox那种hover和pressed的表现效果，但是在现有的基础上去扩展实现也不是难事。