虚幻引擎编辑器拓展（一）

虚 幻 引 擎 程 序 开 发 工 程 师 班

游 戏 设 计 学 院

火星时代教育

1

PART ONE

概述

概述

Slate是一套用户界面框架解决方案。虚幻引擎就是基于Slate框架进行编写，**其具备高效的硬件加速，跨平台，适配简单的特性**，采用了C++进行编写，遵守常规用户界面设计框架的特点。组件化模式为核心设计模式。

在过去的开发历程中，Slate框架被应用到引擎产品设计中，很多游戏公司使用Slate框架构建游戏产品界面。但是由于游戏交互界面数量庞大，设计精细度和调整频率极高，而Slate框架本身无法单纯通过美术设计人员使用，只能通过编码完成，从而导致生产成本提升。所以在UE4引擎中退居为编辑器及工具设计框架，而新的游戏内UI产品则推荐UMG进行设计。

UMG系统也是由Slate进行构建，他具备独立的用户设计器，通过图形方式，更简单的设计用户界面

Slate可以作为备选，用户拓展引擎界面功能。

Slate特点

1. Slate是一套高效且灵活的支持用户完全自定义的UI设计框架，它与平台无关。旨在构建工具和应用程序。
2. 使用布局和样式功能，使得UI可以更轻松的设计和迭代
3. 可以轻松的访问代码和数据
4. 支持完全程序生成UI
5. 添加样式和动画困难
6. 对于非编码类设计人员不友好

Slate框架简述

UMG框架设计源泉是Slate框架，所以UMG框架的一些概念，在Slate框架中依然通用。例如插槽，面板，控件概念，均存在。根据插槽数量，我们可以将控件划分三大类（方便我们学习）

1. 不具备插槽（Image，Text Block）
2. 只有一个插槽（Button，Border）
3. 多个插槽（一般是面板）

一般不具备插槽，或是只有一个插槽的控件具备绘制单元，可以进行绘制逻辑执行，和UMG中一样。

在控件进行生成时，**控件遵循递归方式，父控件根据子控件大小决定自身大小。然后根据控件大小，计算对应的位置。**

控件分类

控件大致被分为三类

- LeafWidget不带有插槽控件
- Panels带有多个插槽控件
- Compound Widget拥有一个插槽的控件

常见控件

控件大致被分为三类

- LeafWidget不带有插槽控件
- Panels带有多个插槽控件
- Compound Widget拥有一个插槽的控件

2

PART TWO

Hello Slate

2-2 小试牛刀

操作步骤

1. 引入Slate模块（InputCore，Slate，SlateCore）
2. 注册悬停页面窗口生成器
3. 编写生成函数
4. 构建窗口生成代码
5. 执行创建显示逻辑

引入模块

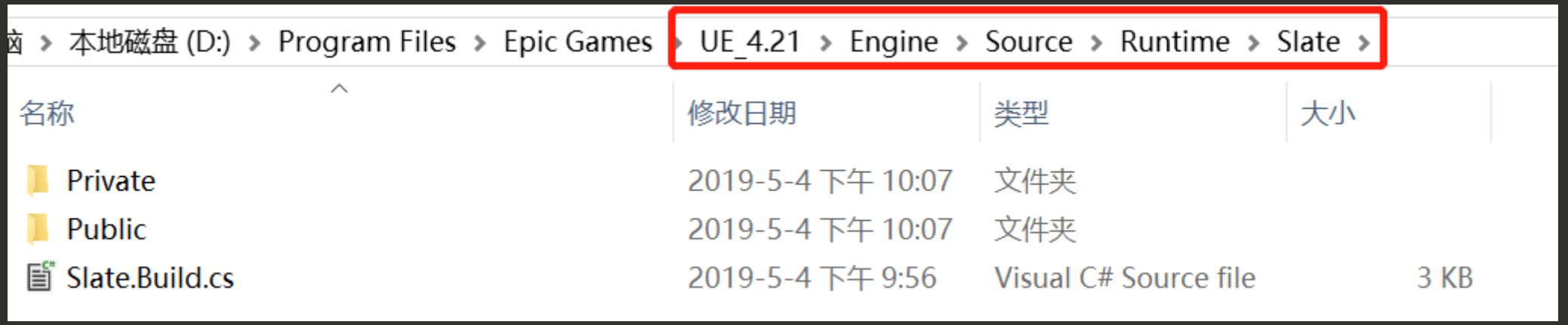
通过编写小的案例，让大家先熟悉Slate的基本操作结构，和逻辑入口。

首先，我们需要在工程中加入Slate模块，否则无法使用Slate框架相关内容。

```
// Uncomment if you are using Slate UI
PrivateDependencyModuleNames.AddRange(new string[] { "Slate", "SlateCore" });
```

源码路径

在项目中，Slate框架源码路径在Source/Runtime/Slate中。创建的C++工程均会关联引擎源码（非全部），也可以在VS中查看。



编写生成器代码

我们将代码写在GameMode中，便于处理。

```
UFUNCTION(Exec)
void RegistTabWindows();
```

```
void AUECPPGameModeBase::RegistTabWindows()
{
    FGlobalTabmanager::Get()->RegisterNomadTabSpawner(TabWindowsDelName, FOnSpawnTab::CreateUObject(this, &AUECPPGameModeBase::OnSpawnTab))
        .SetDisplayName(NSLOCTEXT("HelloTab", "TabWindowsTitle", "Hi Windows"));
}
```

TabWindowsDelName：FName类型数据，用于在打开窗口时，调用窗口生成器，生成窗口

FOnSpawnTab：生成窗口代理

编写生成回调函数

注意函数返回类型，SDockTab是可以用于悬停的窗口类型

```
TSharedRef<class SDockTab> OnSpawnTab(const class FSpawnTabArgs& InArgs);
```

```
TSharedRef<class SDockTab> AUECPPGameModeBase::OnSpawnTab(const class FSpawnTabArgs& InArgs)
{
    return SNew(SDockTab)
        .TabRole(ETabRole::NomadTab)
        [
            SNew(STextBlock)
            .Text(NSLOCTEXT("HelloTab", "WindowsText", "this is the windows's content!"))
        ];
}
```

对于SDockTab对象构建，我们需要使用宏**SNew**完成，其中方括号是重载函数，旨在描述内部添加的其他控件

参数FSpawnTabArgs我们在后面会讲解，主要用来传递参数，使用代理的设计模式

执行显示逻辑

显示悬停窗口主要通过注册过的生成器完成，直接执行生成器代理绑定的生成函数（通过注册名称），即可完成构建加显示窗口

```
UFUNCTION(Exec)
void ShowTabWindows();
```

```
void AUECPPGameModeBase::ShowTabWindows()
{
    FGlobalTabmanager::Get()->InvokeTab(TabWindowsDelName);
}
```

注意：由于Slate框架并没有支持热更新，所以调整的编码逻辑无法直接被应用

移除注册

可以主动通过调用代码，移除注册的生成器

```
FGlobalTabmanager::Get()->UnregisterNomadTabSpawner(TabWindowsDelName);
```

切记，注册的Tab，在确定不再使用后已定要在管理器中移除。

头文件

```cpp
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/GameModeBase.h"
#include "UECPPGameModeBase.generated.h"

/**
 *
 */
UCLASS()
class UECPP_API AUECPPGameModeBase : public AGameModeBase
{
    GENERATED_BODY()
protected:
    const FName TabWindowsDelName = TEXT("TabWindows");
public:
    UFUNCTION(Exec)
    void RegistTabWindows();
    TSharedRef<class SDockTab> OnSpawnTab(const class FSpawnTabArgs& InArgs);
    UFUNCTION(Exec)
    void ShowTabWindows();

};
```

源文件

```cpp
#include "UECPPGameModeBase.h"
#include "Engine/Engine.h"
#include "TabManager.h"
#include "Widgets/Docking/SDockTab.h"
#include "Widgets/Layout/SBox.h"
#include "Widgets/Text/STextBlock.h"

void AUECPPGameModeBase::RegistTabWindows()
{
    FGlobalTabmanager::Get()->RegisterNomadTabSpawner(TabWindowsDelName, FOnSpawnTab::CreateUObject(this, &AUECPPGameModeBase::OnSpawnTab))
        .SetDisplayName(NSLOCTEXT("HelloTab", "TabWindowsTitle", "Hi Windows"));
}

TSharedRef<class SDockTab> AUECPPGameModeBase::OnSpawnTab(const class FSpawnTabArgs& InArgs)
{
    return SNew(SDockTab)
        .TabRole(ETabRole::NomadTab)
        [
            SNew(STextBlock)
            .Text(NSLOCTEXT("HelloTab", "WindowsText", "this is the windows's content!"))
        ];
}

void AUECPPGameModeBase::ShowTabWindows()
{
    FGlobalTabmanager::Get()->InvokeTab(TabWindowsDelName);
}
```

2-2 语法结构剖析

创建控件

在Slate中我们需要注意，所有创建的控件如需持有，一定要构建为共享指针，否则将会涉及到释放问题，一旦疏忽，将导致不可控制的内存泄露风险！

一般我们创建Slate控件，无需直接使用new关键，通过引擎提供的两个工厂生成宏即可完成生成。

- SNew 创建Slate控件，并反馈控件共享指针
- SAssignNew 创建Slate控件，可以通过第一个参数指定返回指针的存储对象。此宏非常方便用于链式语法中内部参数构建的控件存储。

参照下张PPT

创建控件

```
SAssignNew(SelfWindows, SWindow)
.Title(LOCTEXT("title", "Self Windows"))
.ClientSize(FVector2D( InX: 800, InY: 600))
.ScreenPosition(FVector2D( InX: 500, InY: 500))
.Style(&WindowStyle)
.IsTopmostWindow(true)
.SupportsMinimize(false) // SWindow::FArguments&
[
    SNew(SCanvas)
    +SCanvas::Slot().Position(FVector2D( InX: 10, InY: 20)).Size(FVector2D( InX: 100, InY: 100))
    [
        SNew(SBorder).HAlign(HAlign_Fill).VAlign(VAlign_Fill)
    ]
    +SCanvas::Slot().Position(FVector2D( InX: 120, InY: 20)).Size(FVector2D( InX: 80, InY: 40))
    [
        SNew(SButton).HAlign(HAlign_Center).VAlign(VAlign_Center).OnClicked(FOnClicked::CreateUObject(this, &UWindowsBuilder::OnButtonClicked))
        [
            SNew(STextBlock).Text(LOCTEXT("button_text", "OK")).ColorAndOpacity(FLinearColor::Blue)
        ]
    ]
];
```

重载运算符

所有的控件大致被分为功能控件和容器控件。容器控件用来装填其他控件。在Slate里面一样使用此规则。当控件被添加到容器时，需要使用到插槽。在Slate中有两个被特殊重载的运算符：[ ]和+。

运算符+：表明添加一个插槽。注意只有当父容器允许添加多个插槽才可以使用加号，如果父容器只允许添加一个控件，则禁止使用+号。

运算符[]：表明插槽内部装填的内容是什么控件。注意：**一个插槽内只能装填一个控件**。如果父容器允许装填多个插槽，则需要通过+运算符添加新的插槽。

参考下页PPT分析

重载运算符

```
SAssignNew(SelfWindows, SWindow)
.Title(LOCTEXT("title", "Self Windows"))
.ClientSize(FVector2D( InX: 800, InY: 600))
.ScreenPosition(FVector2D( InX: 500, InY: 500))
.Style(&WindowStyle)
.IsTopmostWindow(true)
.SupportsMinimize(false) // SWindow::FArguments&
[
    SNew(SCanvas)
    +SCanvas::Slot().Position(FVector2D( InX: 10, InY: 20)).Size(FVector2D( InX: 100, InY: 100))
    [
        SNew(SBorder).HAlign(HAlign_Fill).VAlign(VAlign_Fill)
    ]
    +SCanvas::Slot().Position(FVector2D( InX: 120, InY: 20)).Size(FVector2D( InX: 80, InY: 40))
    [
        SNew(SButton).HAlign(HAlign_Center).VAlign(VAlign_Center).OnClicked(FOnClicked::CreateUObject(this, &UWindowsBuilder::OnButtonClicked))
        [
            SNew(STextBlock).Text(LOCTEXT("button_text", "OK")).ColorAndOpacity(FLinearColor::Blue)
        ]
    ]
];
```

我们对照上面的代码，第一个箭头，父容器是SWindos只允许有一个子控件，所以添加控件直接使用[]运算符，无需使用+号添加插槽。而第二个箭头控件是SCanvas可以添加多个子控件，所以需要通过使用+添加插槽。而第三个箭头SButton又只能添加一个子控件，所以没有+。

3

PART THREE

Slate&UMG

如何将UMG添加到Slate窗口

UMG本身就是由Slate控件支撑创建的，所以当我们创建继承自**UUserwidget**控件的对象可以直接将控件内的Slate控件获取，主要通过以下代码完成。

```
TSubclassOf<UUserWidget> WidgetClass = LoadClass<UUserWidget>(nullptr, TEXT("WidgetBlueprint'/Game/NewWidgetBlueprint1.NewWidgetBlueprint1_C'"));

UUserWidget* Widget = CreateWidget<UUserWidget>(GetWorld()->GetFirstPlayerController(), WidgetClass);

return SNew(SDockTab).TabRole(NomadTab)
    [
        Widget->GetRootWidget()->TakeWidget()
    ];
```

需要我们注意的是：UMG对象一般是要依赖运行游戏的，也就是游戏关闭后，UMG对象会随之消亡。而Slate控件不与游戏世界相关，所以在上面的操作中，如果是在运行时创建的Slate控件，并将UMG对象控件添加到Slate中，关闭游戏后再打开Slate控件面板，有可能会出现空指针情况。

4

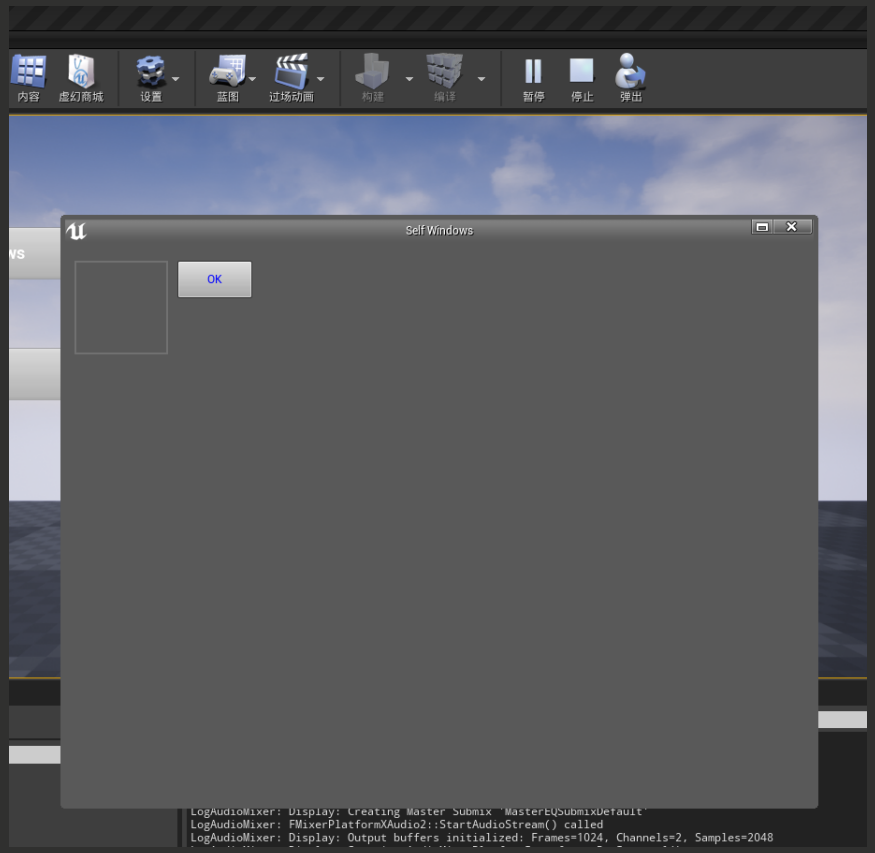
PART FOUR

SWindow

SWindow

SWindow允许你在虚幻引擎中独立创建一个窗口，窗口与原编辑器无关，独立存在。如下图所示，如在开发软件过程中，我们可以通过创建独立窗口编写编辑器界面。

SWindow本质依旧是创建容器，容器中添加的控件，控件构建方式，依旧遵守前面章节中的内容。



创建SWindow

创建代码如下，注意我们可以通过使用FWindowStyle数据类型（WindowStyle变量）来设置window样式。

```
void UWindowsBuilder::Build()
{
    SAssignNew(SelfWindows, SWindow)
    .Title(LOCTEXT("title", "Self Windows"))//窗体名称
    .ClientSize(FVector2D( InX: 800, InY: 600))//窗体大小
    .ScreenPosition(FVector2D( InX: 500, InY: 500))//窗体位置
    .Style(&WindowStyle)//窗体样式
    .IsTopmostWindow(true)//窗体是否置顶
    .SupportsMinimize(false)//是否支持窗体最小化
    [
        SNew(SCanvas)
        +SCanvas::Slot().Position(FVector2D( InX: 10, InY: 20)).Size(FVector2D( InX: 100, InY: 100))
        [
            SNew(SBorder).HAlign(HAlign_Fill).VAlign(VAlign_Fill)
        ]
        +SCanvas::Slot().Position(FVector2D( InX: 120, InY: 20)).Size(FVector2D( InX: 80, InY: 40))
        [
            SNew(SButton).HAlign(HAlign_Center).VAlign(VAlign_Center).OnClicked(FOnClicked::CreateUObject(this, &UWindowsBuilder::OnButtonClicked))
            [
                SNew(STextBlock).Text(LOCTEXT("button_text", "OK")).ColorAndOpacity(FLinearColor::Blue)
            ]
        ]
    ];
}
```

打开SWindow

打开需要借助FSlateAppliocation管理器来完成，依旧是添加窗体到窗体列表，完成渲染工作。代码参照如下。

```
void UWindowsBuilder::Show()
{
    if (SelfWindows.IsValid())
    {
        FSlateApplication::Get().AddWindow(SelfWindows.ToSharedRef());
    }
}
```

THANK