虚幻引擎编辑器拓展（二）

虚 幻 引 擎 程 序 开 发 工 程 师 班

游 戏 设 计 学 院

1

PART ONE

添加启动菜单

添加启动

一般创建的Slate控件，我们可以在启动游戏时启动，通过调用代码，也可以直接在编辑器中启动，以下我们介绍下如果通过向编辑器中添加按钮来完成启动。大致分为以下几种：

- 工具栏按钮
- 菜单按钮

1-1 菜单和工具栏中添加按钮

菜单工具栏

菜单和工具栏中添加的按钮本质是一样的，区别只是在注册添加位置时略有不同，总体可以按照以下步骤完成：

1. 创建**Commands**，用于添加指令信息，一个指令集中可以添加多个CommandInfo
2. 在Commands中创建CommandInfo，用于绑定执行事件，通过使用宏**UI_COMMAND**完成创建
3. 创建UICommandList，用于注册指令CommandInfo
4. 创建Extender，可以理解为是在菜单栏或是工具栏中额外添加的拓展器，用于添加CommandList
5. 将Extender添加到菜单或是任务栏管理器中

1-2 添加TCommands类

创建Commands类

添加原生C++类到模块中，并继承Tcommands，并在类中添加CommandInfo。注意，你需要多少个入口按钮就添加多少个CommandInfo。Commands类相当于是一个菜单集，使用时需要注意以下细节：

1. 父类是模板类，模板类需要提供模板参数，参数即为当前Command类
2. 必须显示调用父类带参构造函数
3. 必须实现RegisterCommands函数，否则此类为抽象类

头文件

```cpp
#pragma once

#include "CoreMinimal.h"
#include <Commands.h>

/**
*
*/
class WHITE_API FCustomCommands : public TCommands<FCustomCommands>
{
public:
    //指令集名称 指令集描述 父级名称 注册样式名称（无样式先随意填写，禁止留白）
    FCustomCommands()
        : TCommands<FCustomCommands>(TEXT("CustomCommands"), NSLOCTEXT("WhitePlugin", "CCommands", "White Plugin"), NAME_None, TEXT("ABC"))
    {}
public:
    //注意此函数需要添加到公有访问中
    virtual void RegisterCommands() override;

public:
    //添加的指令信息对象
    TSharedPtr<class FUICommandInfo> openCustomSlate;
};
```

源文件

注意已定要添加宏LOCTEXT_NAMESPACE

```
1   // Fill out your copyright notice in the Description page of Project Settings.
2
3   #include "CustomCommands.h"
4
5   #define LOCTEXT_NAMESPACE "WhitePluginCustomCommand"
6
7   void FCustomCommands::RegisterCommands()
8   {
9       //注册指令信息ID 指令名称 指令描述 指令样式 绑定打开快捷键
10      UI_COMMAND(openCustomSlate, "OpenCustomSlate", "this is my custom button", EUserInterfaceActionType::Button, FInputGesture());
11  }
12
13  #undef LOCTEXT_NAMESPACE
14
```

1-3 添加UICommandList类

添加UICommandList

CommandList主要用于管理指令列表，可以将多个指令添加到列表中。

找到模块入口函数，添加UICommandList，示例代码中将此步骤添放到插件StartModule函数中完成，代码如下。

```
void FWhiteModule::StartupModule()
{
    // This code will execute after your module is loaded into memory; the exact timing is specified in the .uplugin file per-module
    //注册指令集
    FCustomCommands::Register();
    //创建指令列表 头文件生命的变量 TSharedPtr<class FUICommandList> CustomCommandList;
    CustomCommandList = MakeShareable(new FUICommandList);
    //将指令信息添加到列表中，并绑定点击回调函数 OnOpenSlateButtonClicked（无返回类型无参数）
    CustomCommandList->MapAction(
        FCustomCommands::Get().openCustomSlate,
        FExecuteAction::CreateRaw(this, &FWhiteModule::OnOpenSlateButtonClicked),
        FCanExecuteAction());
}
```

1-4 寻找EditorModule

FLevelEditorModule

首先需要通过**菜单扩展点**获取到按钮添加的位置，注意扩展点主要用于管理工具栏和菜单中的按钮的布局关系。

首先使用扩展点获取到LevelEditorModule。通过LevelEditorModule配合菜单扩展点完成按钮添加参照示例如下：

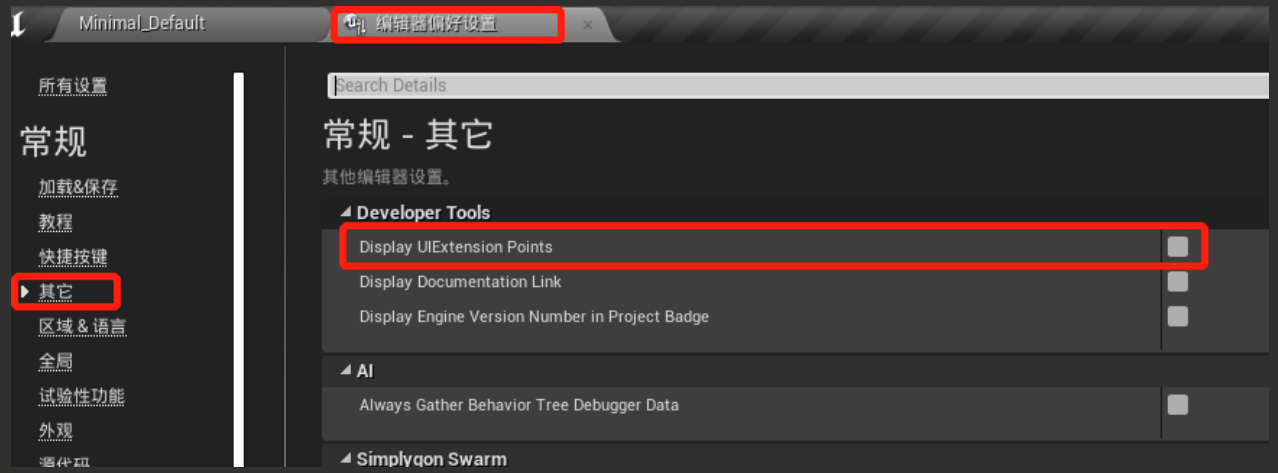
```
void FWhiteModule::StartupModule()
{
    // This code will execute after your module is loaded into memory; the exact timing is specified in the .uplugin file per-module
    //注册指令集
    FCustomCommands::Register();
    //创建指令列表 头文件生命的变量 TSharedPtr<class FUICommandList> CustomCommandList;
    CustomCommandList = MakeShareable(new FUICommandList);
    //将指令信息添加到列表中，并绑定点击回调函数 OnOpenSlateButtonClicked（无返回类型无参数）
    CustomCommandList->MapAction(
        FCustomCommands::Get().openCustomSlate,
        FExecuteAction::CreateRaw(this, &FWhiteModule::OnOpenSlateButtonClicked),
        FCanExecuteAction());

    FLevelEditorModule& LevelEditorModule = FModuleManager::LoadModuleChecked<FLevelEditorModule>("LevelEditor");
}
```

1-5 查看菜单扩展点

菜单扩展点

菜单扩展点主要是用来添加新的菜单到面板时，描述新菜单的位置关系，如需查看可以通过在编辑器设置中找到



1-6 FExtender

FExtender

Fextender是用来存放CommandList的，将Extender放置在菜单栏或是工具栏中，用于拓展面板按钮。

```cpp
void FWhiteModule::StartupModule()
{
    // This code will execute after your module is loaded into memory; the exact timing is specified in the .uplugin file per-module
    //注册指令集
    FCustomCommands::Register();
    //创建指令列表 头文件生命的变量 TSharedPtr<class FUICommandList> CustomCommandList;
    CustomCommandList = MakeShareable(new FUICommandList);
    //将指令信息添加到列表中，并绑定点击回调函数 OnOpenSlateButtonClicked（无返回类型无参数）
    CustomCommandList->MapAction(
        FCustomCommands::Get().openCustomSlate,
        FExecuteAction::CreateRaw(this, &FWhiteModule::OnOpenSlateButtonClicked),
        FCanExecuteAction());

    FLevelEditorModule& LevelEditorModule = FModuleManager::LoadModuleChecked<FLevelEditorModule>("LevelEditor");
    //创建拓展对象
    TSharedPtr<FExtender> MenuExtender = MakeShareable(new FExtender());
    //装载拓展命令列表
    //参数说明 1菜单扩展点名称 2跟随方式 3指令List 4绑定拓展添加Builder 函数原型void AddMenuExtension(FMenuBuilder& Builder);
    MenuExtender->AddMenuExtension("AboutUnrealED", EExtensionHook::After, CustomCommandList,
        FMenuExtensionDelegate::CreateRaw(this, &FWhiteModule::AddMenuExtension));
    //菜单中添加拓展器
    LevelEditorModule.GetMenuExtensibilityManager()->AddExtender(MenuExtender);
}
```

MenuBuilder

在拓展器中绑定了代理函数AddMenuExtension，通过代理函数告知编辑器系统添加的按钮是谁。

```
void FWhiteModule::AddMenuExtension(FMenuBuilder& Builder)
{
    //添加入口按钮
    Builder.AddMenuEntry(FCustomCommands::Get().openCustomSlate);
}
```

编译后启动编辑器将看到添加的按钮已经在菜单中

1-7 添加到工具栏

FExtender

添加到工具栏如同菜单栏一致，需要添加拓展器，然后将指令列表装载到拓展器中

```
void FWhiteModule::StartupModule()
{
    // This code will execute after your module is loaded into memory; the exact timing is specified in the .uplugin file per-module
    //注册指令集
    FCustomCommands::Register();
    //创建指令列表 头文件生命的变量 TSharedPtr<class FUICommandList> CustomCommandList;
    CustomCommandList = MakeShareable(new FUICommandList);
    //将指令信息添加到列表中，并绑定点击回调函数 OnOpenSlateButtonClicked（无返回类型无参数）
    CustomCommandList->MapAction(
        FCustomCommands::Get().openCustomSlate,
        FExecuteAction::CreateRaw(this, &FWhiteModule::OnOpenSlateButtonClicked),
        FCanExecuteAction());

    FLevelEditorModule& LevelEditorModule = FModuleManager::LoadModuleChecked<FLevelEditorModule>("LevelEditor");
    //创建拓展对象
    TSharedPtr<FExtender> MenuExtender = MakeShareable(new FExtender());
    //装载拓展命令列表
    //参数说明 1菜单扩展点名称 2跟随方式 3指令List 4绑定拓展添加Builder 函数原型void AddMenuExtension(FMenuBuilder& Builder);
    MenuExtender->AddMenuExtension("AboutUnrealED", EExtensionHook::After, CustomCommandList,
        FMenuExtensionDelegate::CreateRaw(this, &FWhiteModule::AddMenuExtension));
    //菜单中添加拓展器
    LevelEditorModule.GetMenuExtensibilityManager()->AddExtender(MenuExtender);
    //创建拓展器
    TSharedPtr<FExtender> ToolbarExtender = MakeShareable(new FExtender);
    //装载拓展器命令列表
    //注意绑定拓展器代理函数原型void AddToolbarExtension(FToolBarBuilder& Builder)
    ToolbarExtender->AddToolBarExtension("Settings", EExtensionHook::After, CustomCommandList,
        FToolBarExtensionDelegate::CreateRaw(this, &FWhiteModule::AddToolbarExtension));

    LevelEditorModule.GetToolBarExtensibilityManager()->AddExtender(ToolbarExtender);
}
```

FToolBarBuilder

FToolBarBuilder是用来添加指令到工具栏列表的构建器

```
void FWhiteModule::AddToolbarExtension(FToolBarBuilder& Builder)
{
    Builder.AddToolBarButton(FCustomCommands::Get().openCustomSlate);
}
```

编译执行结果如下

1-8 卸载Commands

卸载动作

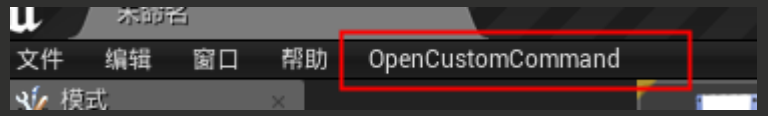
所有的操作都需要关注卸载，记得卸载是必须的，它可以让你的逻辑更加的稳固，数据更加的安全。我们可以在模块卸载函数中编写卸载指令动作。

```
void FWhiteModule::ShutdownModule()
{
    // This function may be called during shutdown to clean up your module.  For modules that support dynamic reloading,
    // we call this function before unloading the module.
    FCustomCommands::Unregister();
}
```

1-8 添加菜单条按钮

菜单条按钮

如果希望添加菜单按钮，可以使用以下代码完成，基本与添加工具，添加菜单项相同。注意回调函数原型结构 void OnMenuBarExtenderBuild(FMenuBarBuilder& MenuBarBuilder);




```
TSharedPtr<FExtender> MenuBarExtender = MakeShareable(new FExtender);
MenuBarExtender->AddMenuBarExtension("Help", EExtensionHook::After, CommandList,
    FMenuBarExtensionDelegate::CreateRaw(this, &FWhitePluginModule::OnMenuBarExtenderBuild));

//添加拓展器到关卡Module中
LevelModule.GetMenuExtensibilityManager()->AddExtender(MenuExtender);
LevelModule.GetToolBarExtensibilityManager()->AddExtender(ToolBarExtender);
LevelModule.GetMenuExtensibilityManager()->AddExtender(MenuBarExtender);
```

1-9 设置菜单层级

2

PART TWO

添加模式按钮

模式按钮

游戏模式按钮也是我们经常会使用到的操作入口，添加模式按钮在之前的学习基础上进行，更加的简单。

大致步骤分可以拆分为以下内容：

1. **引入模块UnrealEd和LevelEditor，切记需要引入模块**
2. 添加FEdMode类，用于管理模块注册动作
3. 通过编辑器模块注册器（FEditorModeRegistry）进行注册
4. 设置模式面板信息

2-1 添加FEdmode

FEdMode

FEdMode是用来承载模块按钮的入口类，此类会协助完成模块分类项注册。头文件如下

```
3   #pragma once
4
5   #include "CoreMinimal.h"
6   #include <EdMode.h>
7
8   /**
9    *
10   */
11  class WHITE_API FCustomMode : public FEdMode
12  {
13  public:
14      //创建模块ID
15      const static FEditorModeID CustomModeId;
16
17  public:
18      //重写进入函数 每次打开模式都会调用
19      virtual void Enter() override;
20      //卸载退出函数 每次离开都会调用
21      virtual void Exit() override;
22  };
```

源文件

```cpp
// Fill out your copyright notice in the Description page of Project Settings.

#include "CustomMode.h"

//为模块ID赋值
const FEditorModeID FCustomMode::CustomModeId = TEXT("WhiteCustomModeId");

void FCustomMode::Enter()
{
    //显示调用父类函数
    FEdMode::Enter();
}

void FCustomMode::Exit()
{
    //显示调用父类函数
    FEdMode::Exit();
}
```

2-2 注册FEdmode

注册

注册Mode需要通过获取**编辑器模块注册器**（FEditorModeRegistry）来完成注册，可以将注册代码放到插件启动函数中。参照下图。

```
void FWhiteModule::StartupModule()
{
    // ...

    // ...

    //LevelEditorModule.GetToolBarExtensibilityManager()->AddExtender(ToolbarExtender);

    //注册模块
    //参数 1.模式ID 2.模式名称 3.模式图标 4.是否显示
    FEditorModeRegistry::Get().RegisterMode<FCustomMode>(FCustomMode::CustomModeId, LOCTEXT("CustomModeName", "CustomMode"),
        FSlateIcon(), true);
}
```

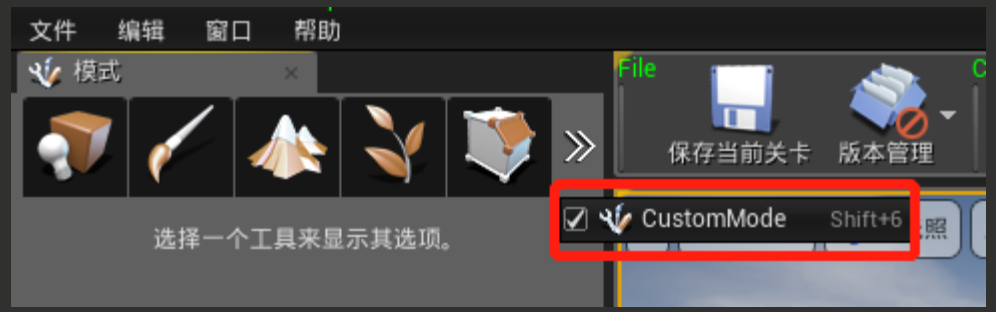
卸载

同样切记所有的模块在使用时都需要注意卸载问题。以下代码在插件卸载函数中调用。

```
void FWhiteModule::ShutdownModule()
{
    // This function may be called during shutdown to clean up your module.  For modules that support dynamic reloading,
    // we call this function before unloading the module.
    FCustomCommands::Unregister();
    //通过ID完成卸载
    FEditorModeRegistry::Get().UnregisterMode(FCustomMode::CustomModeId);
}
```

效果

编译后即可看到模式入口按钮，但是我们并没有设置内容，所以模式中是空白的



2-3 添加面板信息

FModeToolkit

在面板中添加内容，需要依赖FModeToolkit来完成，EdMode中完成Toolkit初始化，用于创建模式面板。

```
2
3     #pragma once
4
5   #include "CoreMinimal.h"
6   #include <BaseToolkit.h>
7
8   /**
9    *
10   */
11  class WHITE_API FCustomModeToolkit : public FModeToolkit
12  {
13  public:
14      FCustomModeToolkit();
15
16      //用于初始化面板控件
17      virtual void Init(const TSharedPtr<IToolkitHost>& InitToolkitHost) override;
18
19      //获取面板名称
20      virtual FName GetToolkitFName() const override;
21      //获取基础面板名称
22      virtual FText GetBaseToolkitName() const override;
23      //获取隶属模式对象
24      virtual class FEdMode* GetEditorMode() const override;
25      //获取面板控件
26      virtual TSharedPtr<class SWidget> GetInlineContent() const override { return ToolkitWidget; }
27
28  private:
29      //面板控件
30      TSharedPtr<SWidget> ToolkitWidget;
31  };
32
```

源文件

```cpp
#include "CustomModeToolkit.h"
#include <SBorder.h>
#include <STextBlock.h>
#include <Editor.h>
#include "CustomMode.h"
#include <EditorModeManager.h>

FCustomModeToolkit::FCustomModeToolkit()
{

}

void FCustomModeToolkit::Init(const TSharedPtr<IToolkitHost>& InitToolkitHost)
{
    //创建Widget控件
    SAssignNew(ToolkitWidget, SBorder)
        .HAlign(HAlign_Center)
        [
            SNew(STextBlock).Text(NSLOCTEXT("CustomModeToolkit", "t1", "CustomMode Toolkit Text"))
        ];
    //显示调用父类函数 需要放到最后
    FModeToolkit::Init(InitToolkitHost);
}

FName FCustomModeToolkit::GetToolkitFName() const
{
    return FName("CustomModeToolkit");
}

FText FCustomModeToolkit::GetBaseToolkitName() const
{
    return NSLOCTEXT("CustomModeToolkit", "DisplayName", "CustomMode Tool Kit");
}

class FEdMode* FCustomModeToolkit::GetEditorMode() const
{
    return GLevelEditorModeTools().GetActiveMode(FCustomMode::CustomModeId);
}
```

2-4 添加Toolkit

添加Toolkit

Toolkit添加需要到EdMode中完成，通过Enter函数完成创建，通过Exit函数完成卸载。

```
3     #pragma once
4
5   #include "CoreMinimal.h"
6   #include <EdMode.h>
7
8   /**
9    *
10   */
11  class WHITE_API FCustomMode : public FEdMode
12  {
13  public:
14      //创建模块ID
15      const static FEditorModeID CustomModeId;
16
17  public:
18      //重写进入函数 每次打开模式都会调用
19      virtual void Enter() override;
20      //卸载退出函数 每次离开都会调用
21      virtual void Exit() override;
22  };
```

源文件

```cpp
#include "CustomMode.h"
#include "CustomModeToolkit.h"
#include <EditorModeManager.h>
#include <ToolkitManager.h>

//为模块ID赋值
const FEditorModeID FCustomMode::CustomModeId = TEXT("WhiteCustomModeId");

void FCustomMode::Enter()
{
    //显示调用父类函数
    FEdMode::Enter();
    if (!Toolkit.IsValid())//父类中的工具箱对象
    {
        Toolkit = MakeShareable(new FCustomModeToolkit);
        Toolkit->Init(Owner->GetToolkitHost());
    }
}

void FCustomMode::Exit()
{
    //先卸载工具箱，再调用父类函数
    if (Toolkit.IsValid())
    {
        FToolkitManager::Get().CloseToolkit(Toolkit.ToSharedRef());
        Toolkit.Reset();
    }
    //显示调用父类函数
    FEdMode::Exit();
}
```

编译后效果

THANKS