任务描述

UE5 编辑器扩展工具 例如在UE的菜单中添加一个窗口

操作

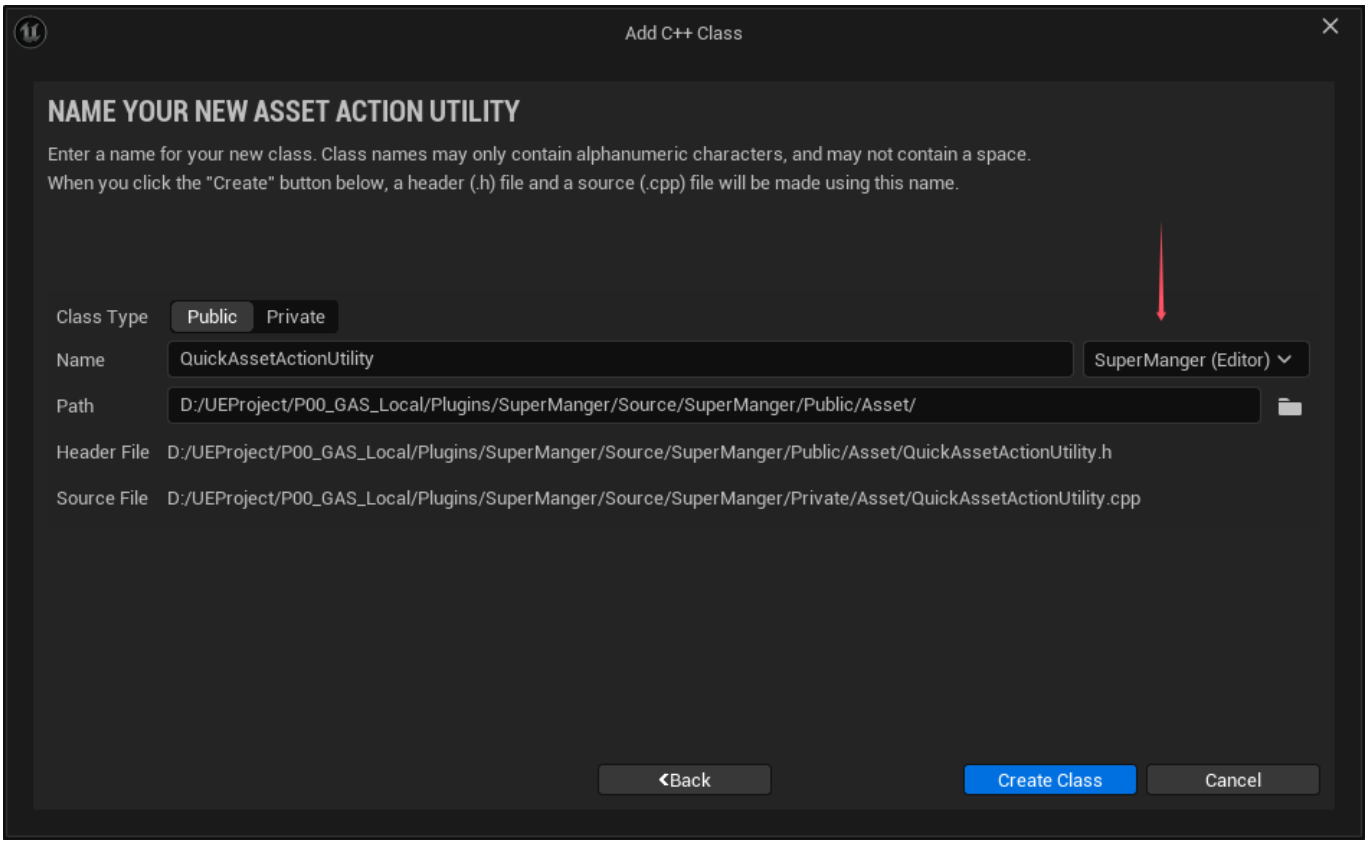
在UE浏览器中 -- Edit -- plugins -- Add --Blank(这个教程中使用的是空白) 点击创建 之后就会在程序类中创建Plugins这个文件，可以在这个文件下进行编辑 首先在.uplugin配置文件中设置CanContainContent为true

在这个配置文件中

```
    "Modules": [
        {
            "Name": "SuperManger",
            "Type": "Editor",    -- Runtime表示影响运行时 Editor表示更改为编辑器并用于加载阶段
            "LoadingPhase": "PreDefault"   -- 在游戏模块之前加载(何时加载插件)
        }
    ]
```

基类

在UE中针对操作对象不同分为对asset(资产)和actor的操作，基类分别是AsssetActionUtilit和ActorActionUtility



```
SuperManger.Build.cs
        PrivateIncludePaths.AddRange(
            new string[] {
                // ... add other private include paths required here ...
                System.IO.Path.GetFullPath(Target.RelativeEnginePath) +
"/Source/Editor/Blutility/Private"
            }
            );


        PublicDependencyModuleNames.AddRange(
            new string[]
            {
                "Core",
                "Blutility"
                // ... add other public dependencies that you statically link with
here ...
            }
            );
```

在上述例子中创建了这个class仍然会报错，是应为没有引入UAssetActionUtility，这里在build中导入AssetActionUtility的module 以及 PrivateInclude -- 参考视频1-p5(2Asset Action)

Asset

批量复制：

```
void UQuickAssetActionUtility::DuplicateAssets(int32 NumOfDuplicates)
{
    if (NumOfDuplicates <= 0)
    {
        Print(TEXT("Error Duplicate number of duplicates!"),FColor::Red);
        return;
    }

    //Select Assets
    TArray<FAssetData> SelectAssetData =
UEditorUtilityLibrary::GetSelectedAssetData();
    uint32 Counter = 0;
    for (const FAssetData& AssetData : SelectAssetData)
    {
        //复制的Asset(资产)名字、路径、复制后的路径名字
        for (int32 i=0;i<NumOfDuplicates;++i)
        {
            const FString SourceAssetPath = AssetData.ObjectPath.ToString();
            const FString NewDuplicateAssetName = AssetData.AssetName.ToString() +
FString::FromInt(i + 1);
            //PackPath -- /Game/Folder
            //PackName -- /Game/Folder/Asset
            const FString NewDuplicateAssetPath =
```

```
FPaths::Combine(AssetData.PackagePath.ToString(),NewDuplicateAssetName);

            if (UEditorAssetLibrary::DuplicateAsset(SourceAssetPath,
NewDuplicateAssetPath))
            {
                UEditorAssetLibrary::SaveAsset(NewDuplicateAssetPath,false);
                ++Counter;
            }
        }
    }
    if (Counter > 0 )
    {
        Print(TEXT("Successful Duplocated") + FString::FromInt(Counter) + "
Files!", FColor::Green);
    }
}
```

核心: UEditorUtilityLibrary::GetSelectedAssetData() / GetSelectedAsset 仅在编译的时候可以使用，返回在内容浏览器中选择的数据

修改前缀名

在.h文件中维护一个TMap 在函数中遍历GetSelectedAsset的变量的class,通过查表的形式判断是否有合适的前缀名

```
    #include "Materials/Material.h"
    #include "Materials/MaterialInstanceConstant.h"
    #include "Particles/ParticleSystem.h"
    #include "Sound/Soundcue.h"
    #include "Sound/SoundWave.h"
    #include "Engine/Texture.h"
    #include "Blueprint/UserWidget.h"
    #include "Components/SkeletalMeshComponent.h"
    #include "NiagaraSystem.h"
    #include "NiagaraEmitter.h"
    TMap<UClass*,FString>PrefixMap =
        {
        {UBlueprint::StaticClass(),TEXT("BP_")},
        {UStaticMesh::StaticClass(),TEXT("SM_")},
        {UMaterial::StaticClass(), TEXT("M_")},
        {UMaterialInstanceConstant::StaticClass(),TEXT("MI_")},
        {UMaterialFunctionInterface::StaticClass(), TEXT("MF_")},
        {UParticleSystem::StaticClass(), TEXT("PS_")},
        {USoundCue::StaticClass(), TEXT("SC_")},
        {USoundWave::StaticClass(), TEXT("SW_")},
        {UTexture::StaticClass(), TEXT("T_")},
        {UTexture2D::StaticClass(), TEXT("T_")},
        {UUserWidget::StaticClass(), TEXT("WBP_")},
        {USkeletalMeshComponent::StaticClass(), TEXT("SK_")},
        {UNiagaraSystem::StaticClass(), TEXT("NS_")},
        {UNiagaraEmitter::StaticClass(), TEXT("NE_")}
```

```
        };

void UQuickAssetActionUtility::AddPrefixes()
{
    TArray<UObject*>SelectedObjects =  UEditorUtilityLibrary::GetSelectedAssets();
    uint32 Counter = 0;
    for (UObject* SelectedObject : SelectedObjects)
    {
        if (!SelectedObject) continue;
        FString* PrefixFind =  PrefixMap.Find(SelectedObject->GetClass());
        if (!PrefixFind || PrefixFind->IsEmpty())
        {
            Print(TEXT("Fail to find Prefix for class")+SelectedObject-
>GetClass()->GetName(),FColor::Red);
            continue;
        }

        FString OldName = SelectedObject->GetName();
        if (OldName.StartsWith(*PrefixFind))
        {
            Print(OldName + TEXT("Already has prefix Added"), FColor::Green);
            continue;
        }
        //针对UMaterialInstanceConstant直接创建后需要删除原本的M_和后缀_Inst
        if (SelectedObject->IsA<UMaterialInstanceConstant>())
        {
            OldName.RemoveFromStart(TEXT("M_"));
            OldName.RemoveFromEnd(TEXT("_Inst"));
        }
        const FString NewNameWithPrefix = *PrefixFind + OldName;
        UEditorUtilityLibrary::RenameAsset(SelectedObject,NewNameWithPrefix);
        ++Counter;
    }
    if (Counter >= 0)
    {
        ShowNotifyInfo(TEXT("Successful Rename ") + FString::FromInt(Counter) + "
assets!");
    }
}
```

删除没有被引用的asset

```
void UQuickAssetActionUtility::RemoveUnusedAssets()
{
    TArray<FAssetData> SelectAssetsData =
UEditorUtilityLibrary::GetSelectedAssetData();
    TArray<FAssetData> UnusedAssetsData;
    FixUpReirectors();
    for (const FAssetData& AssetData : SelectAssetsData)
    {
      // 查看当前是否被引用
```

```
        TArray<FString> AssetRefrences =
UEditorAssetLibrary::FindPackageReferencersForAsset(AssetData.ObjectPath.ToString(
),false);
        if (AssetRefrences.Num() == 0)
        {
            UnusedAssetsData.AddUnique(AssetData);
        }
    }
    if (UnusedAssetsData.Num() == 0)
    {
        ShowMessageDialog(EAppMsgType::Type::Ok,TEXT("All Assets Used"),false);
        return;
    }
    const int32 DeleteRes = ObjectTools::DeleteAssets(UnusedAssetsData);
    if (DeleteRes == 0)
    {
        return;
    }
    ShowNotifyInfo(TEXT("Successful delete ") + FString::FromInt(DeleteRes) + "
unused assets!");
}
```

核心是UEditorAssetLibrary::FindPackageReferencersForAsset这里对AssetData的查询引用

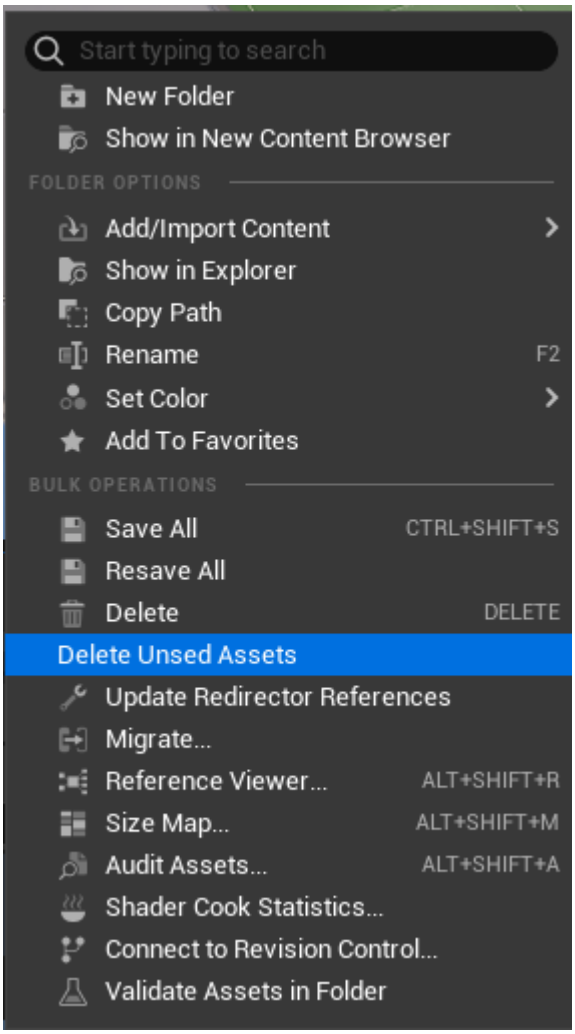
```
void UQuickAssetActionUtility::FixUpReirectors()
{
    // 需要修补的重定向器
    TArray<UObjectRedirector*> RedirectorsToFixArray;

    FAssetRegistryModule& AssetRegistryModule =
        FModuleManager::Get().LoadModuleChecked<FAssetRegistryModule>
(TEXT("AssetRegistry"));

    FARFilter Filter;
    // 允许进入子文件夹
    Filter.bRecursivePaths = true;
    // 决定哪一个文件夹要进入，以及鼠标是否悬停在上面
    Filter.PackagePaths.Emplace("/Game");
    // 过滤的Class是什么 这里只查找类型为ObjectRedirector的资产
    Filter.ClassPaths.Emplace("/Script/ObjectRedirector");


    TArray<FAssetData> OutRedirectorAssetData;
    // 获取所有的重定向器
    AssetRegistryModule.Get().GetAssets(Filter,OutRedirectorAssetData);
    for (const FAssetData& RedirectAssetData : OutRedirectorAssetData)
    {
        if (UObjectRedirector* RedirectorToFix = Cast<UObjectRedirector>
(RedirectAssetData.GetAsset()))
        {
            RedirectorsToFixArray.Add(RedirectorToFix);
        }
```

```
        }

        FAssetToolsModule& AssetToolsModule =
            FModuleManager::LoadModuleChecked<FAssetToolsModule>(TEXT("AssetTools"));
        //调用系统工具完成重定向操作
        AssetToolsModule.Get().FixupReferencers(RedirectorsToFixArray);
    }
```

UObjectRedirector:当资源在 Unreal 编辑器中被移动或重命名时，旧路径会生成一个 UObjectRedirector，作为对新路径的引用。这种重定向器可以确保对旧路径的引用不会立即失效。 FixupReferencers的作用是查询对象、资源仍旧使用的是就的路径的UObjectRedirector，让饭后引旧路径的对象更新为引用新资源的路径，之后会删除所有无用的定向器

添加自定义菜单

举一个例子，现在我们在BrowserContent右键需要添加一个新的功能栏--Delete Unused Assets



如何开始

首先在明确这里是在我们定义的插件中执行的，所以这里选择自定义的Plugin--FSuperMangerModule--.cpp/.h
在这里开启StartupModule的时候会启动一个Init

```
void FSuperMangerModule::StartupModule()
{
    // This code will execute after your module is loaded into memory; the exact
timing is specified in the .uplugin file per-module
    InitCBMenuExtention();

}
void FSuperMangerModule::InitCBMenuExtention()
{
    //加载BrowserModule目录
    FContentBrowserModule& ContentBrowserModule =
        FModuleManager::LoadModuleChecked<FContentBrowserModule>
(TEXT("ContentBrowser"));
    // 获取内容浏览器上下文菜单拓展器，在这个上面可以添加委托用来自定义菜单
    // 这里实际上是一个委托数组 类型和CustomCBMenuExtender一致
    TArray<FContentBrowserMenuExtender_SelectedPaths>&
ContentBrowserMoudleMenuExtenders =
        ContentBrowserModule.GetAllPathViewContextMenuExtenders();
    FContentBrowserMenuExtender_SelectedPaths CustomCBMenuDelegate;
    // 绑定委托函数
    CustomCBMenuDelegate.BindRaw(this,&FSuperMangerModule::CustomCBMenuExtender);
    ContentBrowserMoudleMenuExtenders.Add(CustomCBMenuDelegate);
}
```

PS:应为这里继承不是来自UObject 所以绑定直接使用BindRaw，也有其他方式 简单来说要创建这么一个按钮，

1. 首先需要获取到ContentBrowserModule
2. 获取到ContentBrowserModule的任务委托列表GetAllPathViewContextMenuExtenders
3. 在这个列表中添加我们自己的委托事件就会绑定到对应的事件列表ContentBrowserMoudleMenuExtenders上
4. 而在对ContentBrowserMoudleMenuExtenders的绑定中，在这个绑定过程中需要完成3个绑定事件

4.1 第一次绑定 首先需要告诉编译器在哪里创建、在什么位置 、用什么Icon

```
TSharedRef<FExtender> FSuperMangerModule::CustomCBMenuExtender(const
TArray<FString>& SelectedPath)
{
    TSharedRef<FExtender> MenuExtender(new FExtender());
    if (SelectedPath.Num() > 0)
    {
        // 第一个参数 来自菜单的锚点，这个通过编辑偏好-UIExtent可以查找，这里设置在删除
按钮之下
        // 第二个参数 来自EExtensionHook，描述实在第一个锚点的前面还是后面
        // 第三个参数是一个FUICommandList的智能指针 这个FUICommandList可以绑定一些不同
的关于button执行的操作或者快捷键
        // 第四个参数则是绑定菜单的详细信息
        MenuExtender->AddMenuExtension(FName("Delete"),
            EExtensionHook::After,
            TSharedPtr<FUICommandList>(),
```

```
FMenuExtensionDelegate::CreateRaw(this,&FSuperMangerModule::AddCBMenuEntry));
    }
    return MenuExtender;
}
```

例如:

```
#pragma region FUICommandList
FTestCommandsLineList::FTestCommandsLineList() : TCommands<FTestCommandsLineList>(
    "FTestCommandsLineList",
    NSLOCTEXT("Contexts", "TestCommandsLineList", "SuperManger Plugin"),
    NAME_None,
    FName(*FString("todo")))
{

}

void FTestCommandsLineList::RegisterCommands()
{
    /*
     * 命令的内部名称
     * 在用户界面（如工具栏或菜单）中显示的名称。
     * 命令的描述信息，用于工具提示（Tooltip）等。
     * 指定命令的操作类型
     * FInputChord 快捷键
     */
    UI_COMMAND(CommandA, "FTestCommandsLineList", "Execute TestYaksue CommandA",
EUserInterfaceActionType::Button,
            FInputChord(EModifierKey::Shift | EModifierKey::Alt, EKeys::Z));
}
#pragma endregion


void FSuperMangerModule::OnDeleteUnusedButtonClicked()
{
    CommandLineAAction();
}
void FSuperMangerModule::InitCBMenuExtention()
{
    ...
    //commandList exmp
    // 将命令绑定到动作
    PluginCommandList = MakeShareable(new FUICommandList);
    //为命令映射操作
    PluginCommandList->MapAction(
        FTestCommandsLineList::Get().CommandA,
        FExecuteAction::CreateRaw(this, &FSuperMangerModule::CommandLineAAction),
        FCanExecuteAction());
}
```

这里请参考 CommandLine任务举例： https://blog.csdn.net/u013412391/article/details/107891152

4.2 第二次绑定，绑定菜单详细信息

```
void FSuperMangerModule::AddCBMenuEntry(FMenuBuilder& MenuBuilder)
{
    /*
        InLabel – Label to show in the menu entry
        InToolTip – Tool tip used when hovering over the menu entry
        InIcon – The icon to use
        UIAction – Actions to execute on this menu item.
        InExtensionHook – The section hook. Can be NAME_None
        UserInterfaceActionType – Type of interface action
        InTutorialHighlightName – Optional name to identify this widget and
highlight during tutorials
    */
    MenuBuilder.AddMenuEntry(
        FText::FromString(TEXT("Delete Unsed Assets")),
        FText::FromString(TEXT("Safely delete all unused assets under folder")),
        FSlateIcon(),

FExecuteAction::CreateRaw(this,&FSuperMangerModule::OnDeleteUnusedButtonClicked)
        );
}
```

4.3 第三次绑定 绑定具体执行什么

这里以删除UnusedAsset距离

```
void FSuperMangerModule::OnDeleteUnusedButtonClicked()
{
    /*CommandLineAAction();*/
    // 只能选择一个文件夹
    if (FolderPathSelected.Num() == 0)
    {
        DebugHeader::ShowMessageDialog(EAppMsgType::Type::Ok,TEXT("You didn't
select any folder"));
    }
    else if (FolderPathSelected.Num() > 1)
    {
        DebugHeader::ShowMessageDialog(EAppMsgType::Type::Ok,TEXT("You can only do
this for one folder"));
        return;
    }
    DebugHeader::Print(TEXT("Currently Selected folder : ") +
FolderPathSelected[0],FColor::Green);
    // Return the list of all the assets found in the DirectoryPath.
    TArray<FString> AssetPathNames =
UEditorAssetLibrary::ListAssets(FolderPathSelected[0]);
    if (AssetPathNames.Num() == 0)
```

```
    {
        DebugHeader::ShowMessageDialog(EAppMsgType::Type::Ok,TEXT("No Asset find
in select folder"));
        return;
    }
    EAppReturnType::Type ConfirmType =
DebugHeader::ShowMessageDialog(EAppMsgType::YesNo,TEXT("A total of ") +
        FString::FromInt(AssetPathNames.Num()) + TEXT(" folders selected.\nWould
you like to proceed?"));
    if (ConfirmType == EAppReturnType::No)
    {
        return;
    }
    FixUpRedirectors();
    TArray<FAssetData> UnusedAssetDataArray;
    for (const FString&AssetPathName : AssetPathNames)
    {
        //Do not touch root folder(Collections/Developers)
        if (AssetPathName.Contains(TEXT("Collections")) ||
AssetPathName.Contains(TEXT("Developers")))
        {
            continue;
        }
        //Check Asset Exist
        if (!UEditorAssetLibrary::DoesAssetExist(AssetPathName))
        {
            continue;
        }
        // 查看当前是否被引用 查询UnusedAssetData
        TArray<FString> AssetReferencers =
UEditorAssetLibrary::FindPackageReferencersForAsset(AssetPathName);
        if (AssetReferencers.Num() == 0)
        {
            const FAssetData UnusedAssetData =
UEditorAssetLibrary::FindAssetData(AssetPathName);
            UnusedAssetDataArray.Add(UnusedAssetData);
        }
    }
    if (UnusedAssetDataArray.Num() > 0)
    {
        ObjectTools::DeleteAssets(UnusedAssetDataArray);
    }
    else
    {
        DebugHeader::ShowMessageDialog(EAppMsgType::Type::Ok,TEXT("No unused Asset
find in select folder"));
    }
}

void FSuperMangerModule::FixUpRedirectors()
{
    TArray<UObjectRedirector*>RedirectorsToFixArray;
    // 加载资产重定向工具
    FAssetRegistryModule& AssetRegistryModule =
```

```
        FModuleManager::Get().LoadModuleChecked<FAssetRegistryModule>
(TEXT("AssetRegistry"));

    FARFilter Filter;
    // 允许进入子文件夹
    Filter.bRecursivePaths = true;
    // 决定哪一个文件夹要进入，以及鼠标是否悬停在上面
    Filter.PackagePaths.Emplace("/Game");
    // 过滤的Class是什么 这里只查找类型为ObjectRedirector的资产
    Filter.ClassPaths.Emplace("/Script/ObjectRedirector");

    TArray<FAssetData> OutRedirectorAssetData;
    // 获取所有的重定向器
    AssetRegistryModule.Get().GetAssets(Filter,OutRedirectorAssetData);
    for (const FAssetData& RedirectAssetData : OutRedirectorAssetData)
    {
        if (UObjectRedirector* RedirectorToFix = Cast<UObjectRedirector>
(RedirectAssetData.GetAsset()))
        {
            RedirectorsToFixArray.Add(RedirectorToFix);
        }
    }
    FAssetToolsModule& AssetToolsModule =
        FModuleManager::LoadModuleChecked<FAssetToolsModule>(TEXT("AssetTools"));
    //调用系统工具完成重定向操作
    AssetToolsModule.Get().FixupReferencers(RedirectorsToFixArray);
}
```

请注意每一次绑定的时候委托的参数声明

第二个菜单 --删除空文件夹

这里比较简单，新的菜单只需要通过FMenBuilder中额外增加一个菜单

```
void FSuperMangerModule::AddCBMenuEntry(FMenuBuilder& MenuBuilder){
    ...
    // 添加第二项菜单 删除空文件夹
    MenuBuilder.AddMenuEntry(
        FText::FromString(TEXT("Delete Empty Folders")),
        FText::FromString(TEXT("Safely Delete All Empty Folders")),
        FSlateIcon(),

FExecuteAction::CreateRaw(this,&FSuperMangerModule::OnDeleteEmptyFolderButtonClick
ed)
    );
}
```

而在OnDeleteEmptyFolderButtonClicked执行中需要找到对应的文件夹，而不是文件。这里就使用UEditorAssetLibrary::ListAssets(,,bIncludeFolder:True) 这里第三个参数定义为True之后输出的Tarray中始终将包含一个FolderPath 例如:

```
-- bIncludeFolder:True
Warning      LogTemp                    Folder :
/Game/_Game/SuperMangerBP/BP_Actor.BP_Actor
Warning      LogTemp                    Folder :
/Game/_Game/SuperMangerBP/BP_Actor1.BP_Actor1
Warning      LogTemp                    Folder :
/Game/_Game/SuperMangerBP/BP_Actor2.BP_Actor2
Warning      LogTemp                    Folder :
/Game/_Game/SuperMangerBP/BP_Actor3.BP_Actor3
Warning      LogTemp                    Folder :
/Game/_Game/SuperMangerBP/NewFolder1/
-- bIncludeFolder:False
Warning      LogTemp                    Folder :
/Game/_Game/SuperMangerBP/BP_Actor.BP_Actor
Warning      LogTemp                    Folder :
/Game/_Game/SuperMangerBP/BP_Actor1.BP_Actor1
Warning      LogTemp                    Folder :
/Game/_Game/SuperMangerBP/BP_Actor2.BP_Actor2
Warning      LogTemp                    Folder :
/Game/_Game/SuperMangerBP/BP_Actor3.BP_Actor3
```

在输出中多了一个文件夹

```
void FSuperMangerModule::OnDeleteEmptyFolderButtonClicked()
{
    /*DebugHeader::Print(TEXT("Working"),FColor::Green);*/
    if (FolderPathSelected.Num() == 0)
    {
        DebugHeader::ShowMessageDialog(EAppMsgType::Type::Ok,TEXT("You didn't
select any folder"));
    }
    else if (FolderPathSelected.Num() > 1)
    {
        DebugHeader::ShowMessageDialog(EAppMsgType::Type::Ok,TEXT("You can only do
this for one folder"));
        return;
    }
    DebugHeader::Print(TEXT("Currently Selected folder : ") +
FolderPathSelected[0],FColor::Green);
    // FolderPathArray include Folder Path
    FixUpRedirectors();
    TArray<FString> FolderPathArray =
UEditorAssetLibrary::ListAssets(FolderPathSelected[0],true,true);
    UINT32 Counter = 0;
    // 用来显示
    FString EmptyFolderPathNames;
    // 记录空文件夹
    TArray<FString>EmptyFoldersPathArray;
    for (const FString& FolderPath : FolderPathArray)
    {
```

```
        if (FolderPath.Contains(TEXT("Collections"))
            || FolderPath.Contains(TEXT("Developers"))||
            FolderPath.Contains(TEXT("__ExternalActors__")) ||
            FolderPath.Contains(TEXT("__ExternalObjects__")))
        {
            continue;
        }
        //去除不是Folder的
        if (!UEditorAssetLibrary::DoesDirectoryExist(FolderPath))
        {
            continue;
        }
        // Folder中是否有Assets
        if (!UEditorAssetLibrary::DoesDirectoryHaveAssets(FolderPath))
        {
            EmptyFolderPathNames.Append(FolderPath);
            EmptyFolderPathNames.Append(TEXT("\n"));

            EmptyFoldersPathArray.Add(FolderPath);
        }
    }

    if (EmptyFoldersPathArray.Num() == 0)
    {
        DebugHeader::ShowMessageDialog(EAppMsgType::Type::Ok,TEXT("No Empty
Folders found"),false);
        return;
    }
    EAppReturnType::Type ReturnType =
DebugHeader::ShowMessageDialog(EAppMsgType::Type::OkCancel,
        TEXT("Empty Folders found in:\n") + EmptyFolderPathNames + TEXT("\n"),
        false);
    if (ReturnType == EAppReturnType::Cancel)
    {
        return;
    }
    // delete empty folders
    for (const FString& EmptyFolderPath : EmptyFoldersPathArray)
    {
        if (UEditorAssetLibrary::DeleteDirectory(EmptyFolderPath))
        {
            Counter+=1;
        }
        else
        {
            DebugHeader::Print(TEXT("Failed to delete ") +
EmptyFolderPath,FColor::Red);
        }
    }
    if (Counter > 0)
    {
        DebugHeader::ShowMessageDialog(EAppMsgType::Type::OkCancel,
        TEXT("Successfully Delete ") + FString::FromInt(Counter) + TEXT("Empty
Folders\n"),
```

```
            false);
        }
    }
```

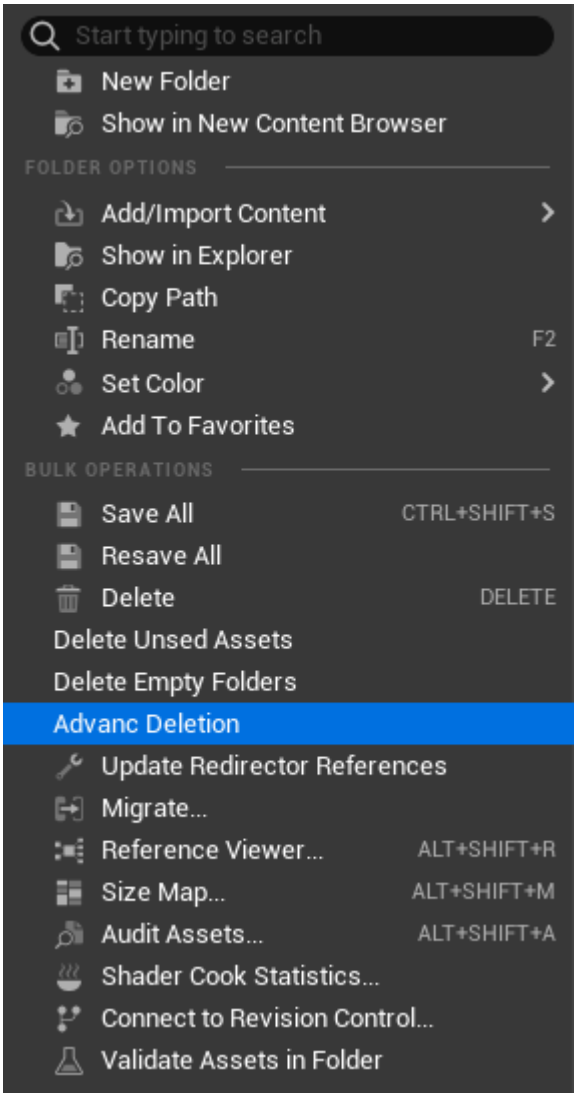
关键： UEditorAssetLibrary::ListAssets(FolderPathSelected[0],true,true); 输出包含文件夹
UEditorAssetLibrary::DoesDirectoryExist(FolderPath); 检查是否为文件夹
UEditorAssetLibrary::DoesDirectoryHaveAssets(FolderPath); 检查文件夹下是否有资产
UEditorAssetLibrary::DeleteDirectory(EmptyFolderPath); 删除空文件夹

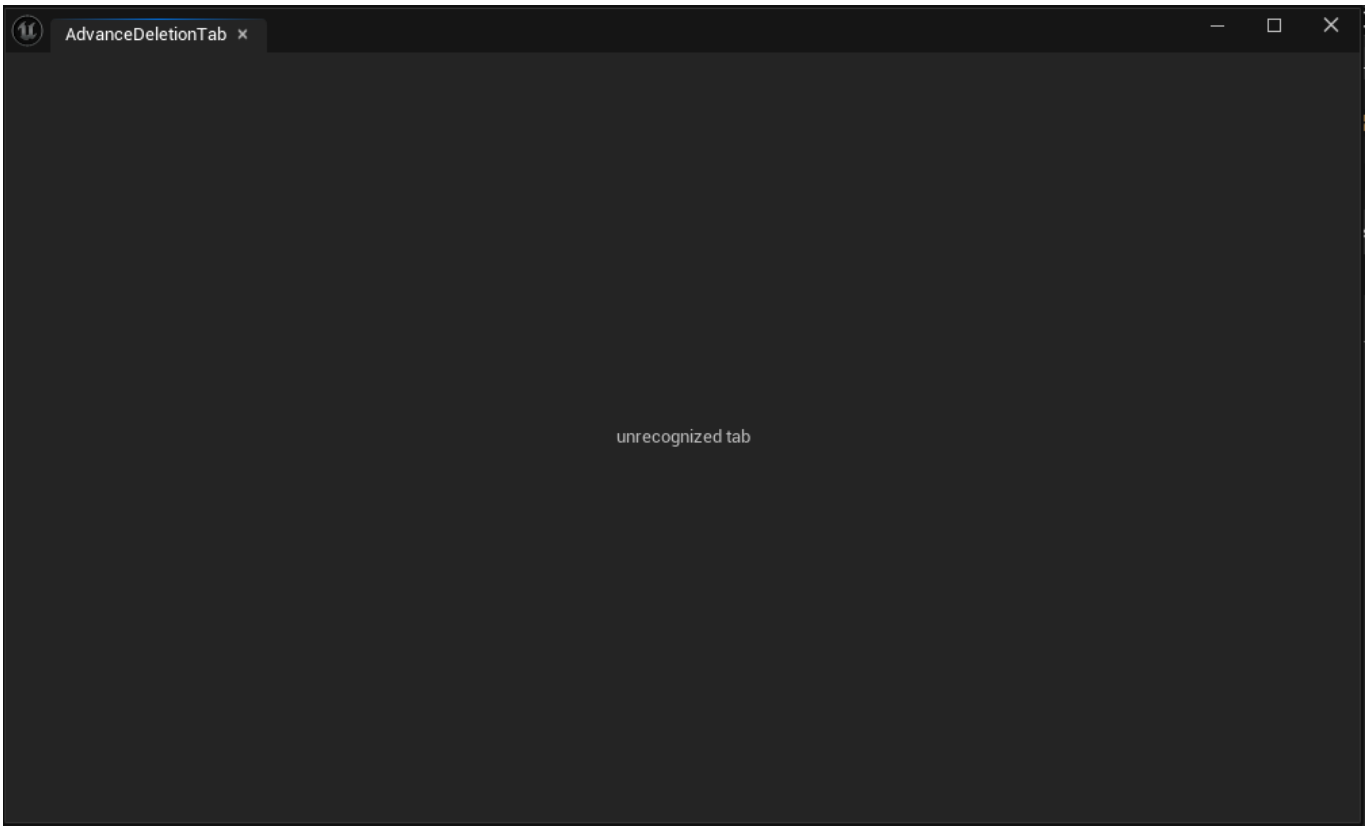
关于智能指针

1. MakeShared() -- 它通过模板推断自动构建共享指针，不需要手动 new，避免内存泄漏。 MakeShared(data)
2. MakeShareable() -- 适用于已经通过 new 创建的对象。 PluginCommandList = MakeShareable(new FUICommandList); 使用的时候需要用户先new对象

Slate Widget

Slate -- UE的主界面基本是由Slate构成的 通过slate可以自定义UE的小窗口 例如





创建一个菜单

创建一个菜单，提供资产阅读界面，通过这个界面可以选择资产、删除资产

首先这里需要在FGlobalTabmanager中注册，这里需要放在Super的StartupModule中执行

```
void FSuperMangerModule::RegisterAdvanceDeletionTab()
{
    //Register a new nomad tab spawner with the tab manage
    //触发Tab的委托
    FGlobalTabmanager::Get()->RegisterNomadTabSpawner(
        FName("AdvanceDeletionTab"),

FOnSpawnTab::CreateRaw(this,&FSuperMangerModule::OnSpawnAdvanceDeletionTab))
    .SetDisplayName(FText::FromString(TEXT("Advance Deletion")));
}
```

这里是向FGlobalTabmanager中注册了一个Tab name是AdvanceDeletionTab，这里要来InovkeTab,第二个参数则是绑定的函数，这里将放回一个SDockTab

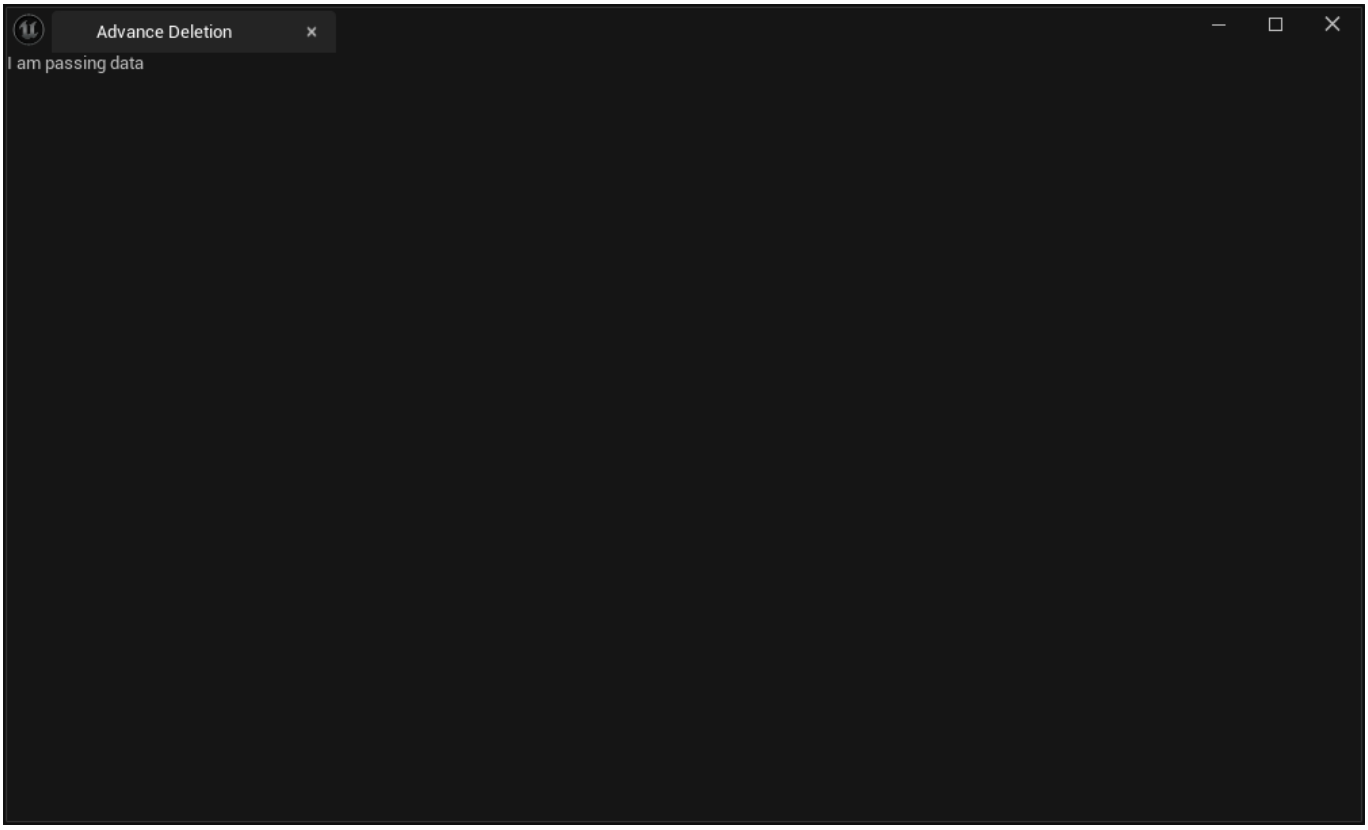
```
TSharedRef<SDockTab> FSuperMangerModule::OnSpawnAdvanceDeletionTab(const
FSpawnTabArgs& args)
{
    //create new Slate Weight
    return SNew(SDockTab).TabRole(NomadTab);
}
```

创建一个自己的Slate

1. 需要创建一个empty class -- AdvanceDeletionTab

```
#include "Widgets/SCompoundWidget.h"


class SAdvanceDeletionTab : public SCompoundWidget
{
    SLATE_BEGIN_ARGS(SAdvanceDeletionTab){}
    // 作为传递参数的宏定 要传递参数需要这么定义到FArguments中
    SLATE_ARGUMENT(FString,TestString)
    SLATE_END_ARGS()

public:
    void Construct(const FArguments& InArgs);
};
--.cpp files
void SAdvanceDeletionTab::Construct(const FArguments& InArgs)
{
    //Can the widget ever support keyboard focus
    bCanSupportFocus = true;
    // 接受Args中的参数
    //InArgs._TestString;
    ChildSlot
    [
        SNew(STextBlock)
        .Text(FText::FromString(InArgs._TestString))
    ];
}
```

关于slot的简单介绍 如下图所示输出一个I am passing data的Text(这里text就可以理解为一个slot)



在上述是通过SLATE_ARGUMENT作为传参的宏，在Construct的时候构造这个Slot并定义为Text 在外部调用的时候对TestString传值即可

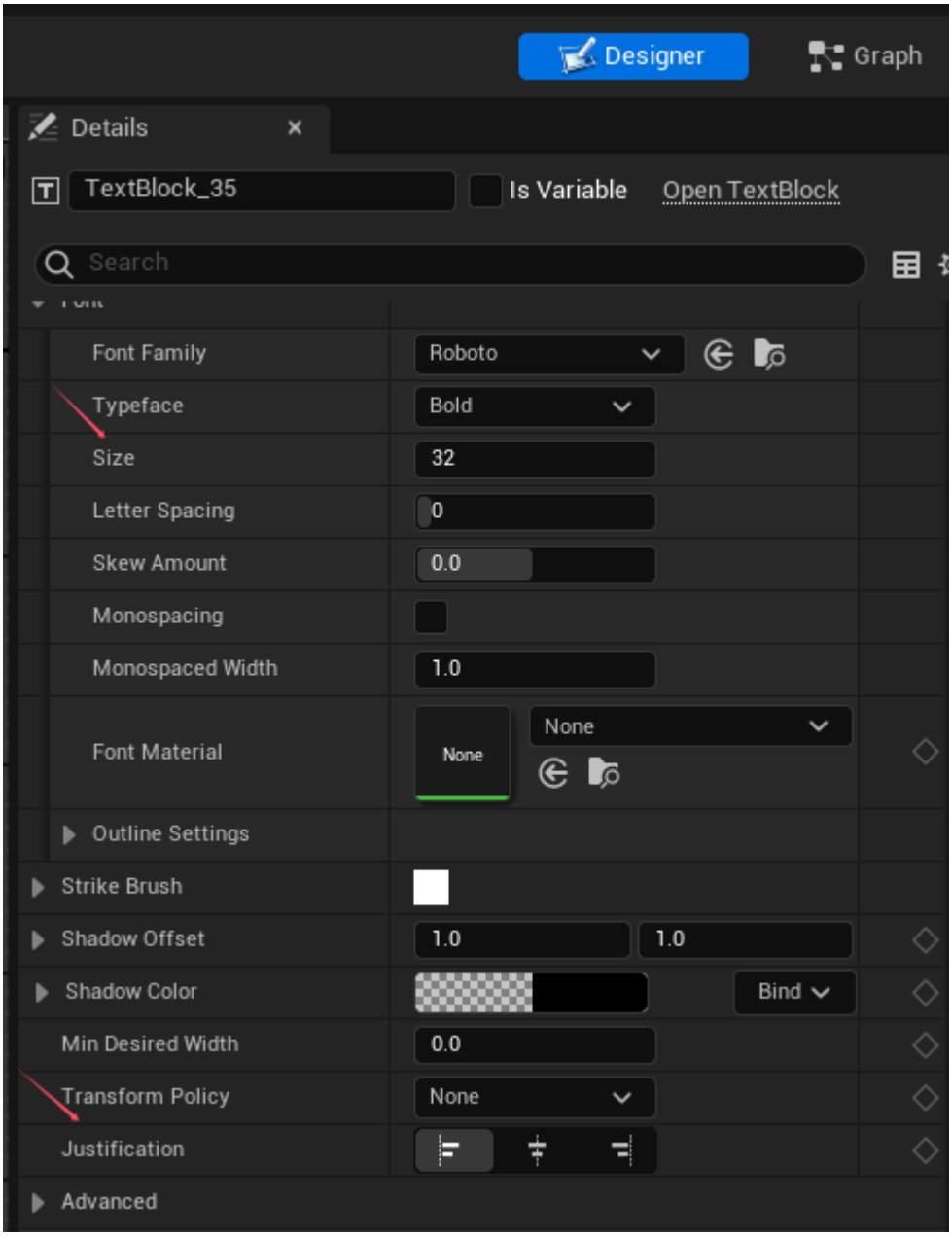
```
TSharedRef<SDockTab> FSuperMangerModule::OnSpawnAdvanceDeletionTab(const
FSpawnTabArgs& args)
{
    return SNew(SDockTab).TabRole(ETabRole::NomadTab)
    [
        SNew(SAdvanceDeletionTab)
        .TestString(TEXT("I am passing data"))
    ];
}
```

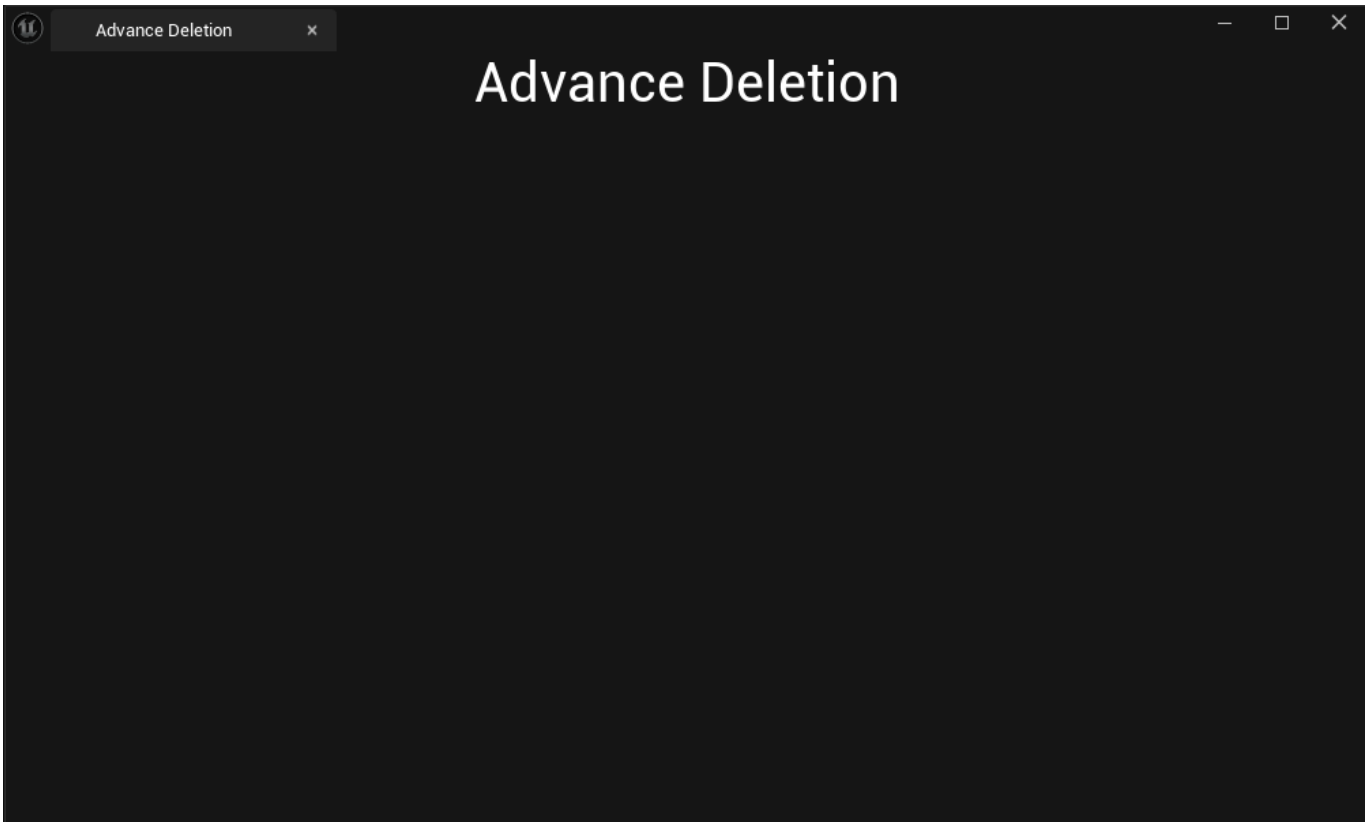
这里slot的格式语法和在蓝图中编辑窗口的时候很接近，所以可以参考蓝图是如何编辑widget中的内容的 例如在窗口中我们创建VerticalBox 并且设置box中的文本font、style、位置

```
    FSlateFontInfo TitleTexFont =
FCoreStyle::Get().GetFontStyle(FName("EmbossedText"));
    TitleTexFont.Size = 30;

    // 接受Args中的参数
    //InArgs._TestString;
    ChildSlot
    [
        //Main VerticalBox
        SNew(SVerticalBox)
```

```
        //First Vertcal slot title text
        +SVerticalBox::Slot()
        .AutoHeight()
        [
            SNew(STextBlock)
            .Text(FText::FromString(TEXT("Advance Deletion")))
            .Font(TitleTexFont)
            .Justification(ETextJustify::Center)
            .ColorAndOpacity(FColor::White)
        ]
    ];
```

Font\Justification等参数都可以在蓝图编辑widget中找到





向Slot中传递Assets

在了解完毕上面的内容(传值、Style)之后 这里就很简单了，首先这里我们需要对传过来的每一个Asset做对应的操作，所以这里肯定是一个数组，数组内部传递的必须是指针/引用变量，而不能是值变量，不然就变成拷贝了

这里定义

```
class SAdvanceDeletionTab : public SCompoundWidget{
    ...
// 作为传递参数的宏定义 要传递参数需要这么定义到FArguments中
    SLATE_ARGUMENT(TArray<TSharedPtr<FAssetData>>,AssetsDataArray)
private:
    // 方便从InArgs中读取
    TArray<TSharedPtr<FAssetData>> AssetsDataUnderSelectedFolderArray;
}
```

而在Manger中创建一个返回对应数据类型的接口

```
TSharedRef<SDockTab> FSuperMangerModule::OnSpawnAdvanceDeletionTab(const
FSpawnTabArgs& args)
{
    //create new Slate Weight
    /*
     *这里[]之间就是一个slot(槽)
     *可以立即为一个占据窗口的小部件
     */
    return SNew(SDockTab).TabRole(ETabRole::NomadTab)
    [
```

```
        SNew(SAdvanceDeletionTab)
        .AssetsDataArray(GetAllAssetDataUnderSelectedFolder())
    ];
}

TArray<TSharedPtr<FAssetData>>
FSuperMangerModule::GetAllAssetDataUnderSelectedFolder()
{
    TArray<TSharedPtr<FAssetData>> AvaiableAssetData;
    TArray<FString>  AssetPathNames =
UEditorAssetLibrary::ListAssets(FolderPathSelected[0]);
    for (const FString& AssetPathName : AssetPathNames)
    {
        if (AssetPathName.Contains(TEXT("Collections"))
            || AssetPathName.Contains(TEXT("Developers"))||
            AssetPathName.Contains(TEXT("__ExternalActors__")) ||
            AssetPathName.Contains(TEXT("__ExternalObjects__"))
            )
        {
            continue;
        }
        if (!UEditorAssetLibrary::DoesDirectoryExist(AssetPathName))
        {
            continue;
        }
        const FAssetData Data = UEditorAssetLibrary::FindAssetData(AssetPathName);
        AvaiableAssetData.Add(MakeShared<FAssetData>(Data));
    }
    return AvaiableAssetData;
}
```

使用SListView显示所有的Asset

```
//Third slot for the asset list
        +SVerticalBox::Slot()
        .AutoHeight()
        [
            // 带有左侧滚动条的box
            SNew(SScrollBox)
            +SScrollBox::Slot()
            [
                SNew(SListView<TSharedPtr<FAssetData>>)
                .ItemHeight(24.f)
                .ListItemsSource(&AssetsDataUnderSelectedFolderArray)
                /*
                    DECLARE_DELEGATE_RetVal_TwoParams (
                     return: The Widget visualization of the item
                    TSharedRef<class ITableRow>,
                        FOnGenerateRow,
                        -- param: An item to visualize
                        ArgumentType,
```

```
                        -- param: The owning widget
                        const TSharedRef< class STableViewBase >& );
                    */
                    .OnGenerateRow(this,&SAdvanceDeletionTab::OnGenerateRowForList)
                ]
            ]
```

OnGenerateRow是一个委托，首先第一个参数是使用object 第二个参数是当前每一行的信息的显示

```
// 决定这行中显示什么
TSharedRef<ITableRow>
SAdvanceDeletionTab::OnGenerateRowForList(TSharedPtr<FAssetData>
AssetDataToDisPlay,
    const TSharedRef<STableViewBase>& OwnerTbale)
{
    // 创建行
    const FString DisplayrAssetName = AssetDataToDisPlay->AssetName.ToString();
    TSharedRef<ITableRow> STbaleRowWidget =
SNew(STableRow<TSharedPtr<FAssetData>>,OwnerTbale)
    [
        SNew(STextBlock)
        .Text(FText::FromString(DisplayrAssetName))
    ];
    return STbaleRowWidget;
}
```

为每一行添加一个CheckBoxState -- 检查是否选中或者没选中

在创建行的内容中添加一个SHorizontalBox 并首先用函数创建ConstructCheckBox

```
TSharedRef<ITableRow>
SAdvanceDeletionTab::OnGenerateRowForList(TSharedPtr<FAssetData>
AssetDataToDisPlay,
    const TSharedRef<STableViewBase>& OwnerTable)
{
    // 传入指针检查
    if (!AssetDataToDisPlay.IsValid())
    {
        return SNew(STableRow < TSharedPtr <FAssetData> >,OwnerTable);
    }
    // 创建行
    const FString DisplayrAssetName = AssetDataToDisPlay->AssetName.ToString();
    TSharedRef< STableRow < TSharedPtr <FAssetData> > > ListViewRowWidget =
        SNew(STableRow < TSharedPtr <FAssetData> >,OwnerTable)
        [
            SNew(SHorizontalBox)
            //First slot for check box
            +SHorizontalBox::Slot()
            .HAlign(HAlign_Left) //左侧对其
```

```
            .VAlign(VAlign_Center) //水平剧中
            .FillWidth(0.5f)
            [
                ConstructCheckBox(AssetDataToDisPlay)
            ]
            ...
```

```
TSharedRef<SCheckBox> SAdvanceDeletionTab::ConstructCheckBox(const
TSharedPtr<FAssetData> AssetDataToDisPlay)
{
    TSharedRef<SCheckBox> ConstructedCheckBox = SNew(SCheckBox)
    .Type(ESlateCheckBoxType::CheckBox)
    .OnCheckStateChanged(this,&SAdvanceDeletionTab::OnCheckBoxStateChanged,AssetDa
taToDisPlay)
    .Visibility(EVisibility::Visible);
    CheckBoxesArray.Add(ConstructedCheckBox);
    return ConstructedCheckBox;
}

void SAdvanceDeletionTab::OnCheckBoxStateChanged(ECheckBoxState NewState,
TSharedPtr<FAssetData> AssetData)
{
    switch (NewState)
    {
    case ECheckBoxState::Unchecked:
            DebugHeader::Print(AssetData->AssetName.ToString() + TEXT(" is
unchecked"),FColor::Red);
            break;
        case ECheckBoxState::Checked:
            DebugHeader::Print(AssetData->AssetName.ToString() + TEXT(" is
checked"),FColor::Green);
            break;
        case ECheckBoxState::Undetermined:
            break;
        default:
            break;
    }
}
```

这里补充一个委托的点： 在查询的时候会发现OnCheckStateChanged是一个单参委托

```
DECLARE_DELEGATE_OneParam( FOnCheckStateChanged, ECheckBoxState );
```

但是我们这里却绑定了两个参数函数

```
SAdvanceDeletionTab::OnCheckBoxStateChanged(ECheckBoxState NewState,
TSharedPtr<FAssetData> AssetData)
```

这是应为bind支持参数拓展，我们绑定的时候额外传入第二个参数，这样就会实现参数的拓展 参考：https://www.cnblogs.com/kekec/p/10678905.html 例如

```
DECLARE_DELEGATE_OneParam(MyDelegate, int32);
MyDelegate MyDelegateFun2;
MyDelegateFun2.BindUObject(this,&ATestForDelegate::DelegateFun2,16);
MyDelegateFun2.Execute(18); // 1816
```

为每一行添加一个button 并实现触发事件

这里使用函数创建button并绑定触发事件

```
TSharedRef<SButton> SAdvanceDeletionTab::CreateButtonTextForRowWidget(const
FString& TextFContent,
    const TSharedPtr<FAssetData>& AssetDataToDisplay)
{
    TSharedRef<SButton> TextButtonRow = SNew(SButton)
        .Text(FText::FromString(TextFContent))

.OnClicked(this,&SAdvanceDeletionTab::OnDeleteButtonClicked,AssetDataToDisplay);

    return TextButtonRow;
}

FReply SAdvanceDeletionTab::OnDeleteButtonClicked(TSharedPtr<FAssetData>
ClickAssetData)
{
    DebugHeader::Print(ClickAssetData->AssetName.ToString() + TEXT(" is
deleted"),FColor::Green);
    return FReply::Handled();
}
```

在每一行的创建函数中继续添加第四个元素到HorizontalBox中

```
    //Fourth slot for a button
    +SHorizontalBox::Slot()
    .HAlign(HAlign_Right)
    .VAlign(VAlign_Fill)
    [
        CreateButtonTextForRowWidget(TEXT("Delete"),AssetDataToDisPlay)
    ]
```

添加删除功能并更新

对于删除操作，这里采用加载module的方式，将SuperManger的引用传进来。而删除Asset的部分在SuperManger中执行完毕 而对于删除操作，因为这里是对ViewList进行操作，所以这里可以使用一个变量存储ViewList，并构建一个可以生成ViewList的函数

```
TSharedRef<SListView<TSharedPtr<FAssetData>>>
SAdvanceDeletionTab::ConstructAssetListView()
{
        ConstructedAssetListView = SNew(SListView<TSharedPtr<FAssetData>>)
        .ItemHeight(24.f)
        .ListItemsSource(&AssetsDataUnderSelectedFolderArray)
        /*
            DECLARE_DELEGATE_RetVal_TwoParams (
             return: The Widget visualization of the item
            TSharedRef<class ITableRow>,
                FOnGenerateRow,
                -- param: An item to visualize
                ArgumentType,
                -- param: The owning widget
                const TSharedRef< class STableViewBase >& );
         */
        .OnGenerateRow(this,&SAdvanceDeletionTab::OnGenerateRowForList);

        return ConstructedAssetListView.ToSharedRef();
}

//Button的触发事件
FReply SAdvanceDeletionTab::OnDeleteButtonClicked(TSharedPtr<FAssetData>
ClickAssetData)
{
    // 加载一个模块并返回这个模块的引用 -- FSuperMangerModule ClassName -- Loads a
module by name -- 在build.cs中
    FSuperMangerModule& SuperMangerModule =
FModuleManager::LoadModuleChecked<FSuperMangerModule>(TEXT("SuperManger"));
    const bool bAssetDelete =
SuperMangerModule.DeleteSingleAssetDataForAssetList(*ClickAssetData.Get());
    //Refresh the list
    if (bAssetDelete)
    {
        // 更新AssetDataItem
        if (AssetsDataUnderSelectedFolderArray.Contains(ClickAssetData))
        {
            AssetsDataUnderSelectedFolderArray.Remove(ClickAssetData);
        }
        //Refresh
        RefrshAssetListView();
    }
    return FReply::Handled();
}
//更新view
void SAdvanceDeletionTab::RefrshAssetListView()
{
    //清空 选中的需要删除的数组
```

```
    CheckBoxesArray.Empty();
    AssetsDataToDeleteArray.Empty();
    if (ConstructedAssetListView.IsValid())
    {
        ConstructedAssetListView->RebuildList();
    }
}
```

更新其实调用的是RebuildList这个接口 当然可以将前面构建的部分替换为这个ConstructAssetListView函数

底部的三个按钮

在底部需要实现删除全部、全选、全不选的过程 创建Button的过程就不再赘述 -- 举例一个

```
TSharedRef<SButton> SAdvanceDeletionTab::ConstructDeSeleteAllButton()
{
    TSharedRef<SButton> ConstructButton = SNew(SButton)
    .ContentPadding(FMargin(5.f))
    .OnClicked(this,&SAdvanceDeletionTab::OntDeSeleteAllButtonClicked);
    ConstructButton->SetContent(ConstructTextForBottonButton(TEXT("DeSelect
All")));
    return ConstructButton;
}

FReply SAdvanceDeletionTab::OntDeSeleteAllButtonClicked()
{
    return FReply::Handled();
}

TSharedRef<STextBlock> SAdvanceDeletionTab::ConstructTextForBottonButton(const
FString& TextFContent)
{
    TSharedRef<STextBlock> ConstructTextBlock = SNew(STextBlock)
        .Text(FText::FromString(TextFContent))
        .Font(GetEmbossedTextFont(15))
        .Justification(ETextJustify::Center);
    return ConstructTextBlock;
}
```

操作流程是 全选/选择一部分 -- 删除全部 首先在h文件中需要一个数组存储等待删除的Asset 比较好操作的是因为这里操作是在对CheckBox处理，所以直接在CheckBox的状态change的函数中添加处理即可：AssetsDataToDeleteArray 是成员变量

```
void SAdvanceDeletionTab::OnCheckBoxStateChanged(ECheckBoxState NewState,
TSharedPtr<FAssetData> AssetData)
{
    switch (NewState)
    {
    case ECheckBoxState::Unchecked:
```

```
            if (AssetsDataToDeleteArray.Contains(AssetData))
            {
                AssetsDataToDeleteArray.Remove(AssetData);
            }
            break;
        case ECheckBoxState::Checked:
            AssetsDataToDeleteArray.AddUnique(AssetData);
            break;
        case ECheckBoxState::Undetermined:
            break;
        default:
            break;
    }
}
```

三个菜单栏的功能

删除所选项目 为了统计有多少内容是需要被删除的，这里需要一个额外的数组进行存储，所以这里设定一个数组

```
TArray<TSharedPtr<FAssetData>> AssetsDataToDeleteArray
```

删除全部其实就是通过module的方法创建一个链接捕获DeleteListAssetDataForAssetList方法

```
FReply SAdvanceDeletionTab::OntDeleteAllButtonClicked()
{
    if (AssetsDataToDeleteArray.Num() == 0)
    {
        DebugHeader::ShowMessageDialog(EAppMsgType::Ok,TEXT("No Assets Data
selected"));
        return FReply::Handled();
    }
    TArray<FAssetData>AssetDataToDelete;
    for (const TSharedPtr<FAssetData>& AssetData : AssetsDataToDeleteArray)
    {
        AssetDataToDelete.Add(*AssetData.Get());
        DebugHeader::Print(*AssetData.Get()->AssetName.ToString(),FColor::Green);
    }
    FSuperMangerModule& SuperMangerModule =
FModuleManager::LoadModuleChecked<FSuperMangerModule>(TEXT("SuperManger"));
    const bool bAssetsDeleted =
SuperMangerModule.DeleteListAssetDataForAssetList(AssetDataToDelete);
    if (bAssetsDeleted)
    {
        for (const TSharedPtr<FAssetData>& DeleteAssetData :
AssetsDataToDeleteArray)
        {
            // 更新拥有的数据
            if (AssetsDataUnderSelectedFolderArray.Contains(DeleteAssetData))
```

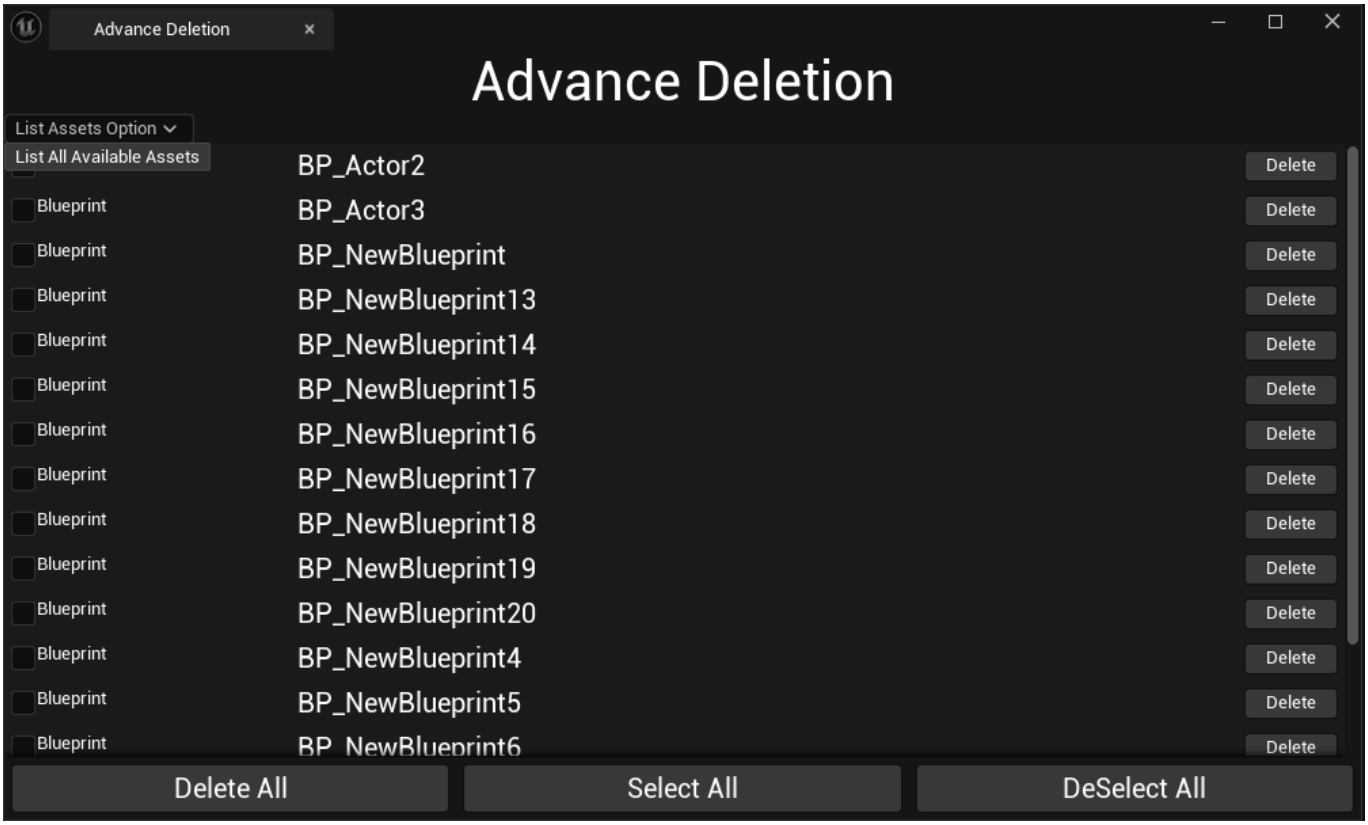
```
            {
                AssetsDataUnderSelectedFolderArray.Remove(DeleteAssetData);
            }
            // 更新DisplayedAssetData
            if (DisplayedAssetData.Contains(DeleteAssetData))
            {
                DisplayedAssetData.Remove(DeleteAssetData);
            }
        }
        RefrshAssetListView();
    }
    return FReply::Handled();
}
```

对于全选和取消全选的部分，首先需要一个checkbox的array 我们知道在ListView中存储其实都是checkbox，而全选和取消全选 就是遍历CheckBoxesArray修改其状态就可以，这里使用一个官方函数SCheckBox::ToggleCheckedState 实现状态的切换

```
FReply SAdvanceDeletionTab::OntDeSeleteAllButtonClicked()
{
    if (CheckBoxesArray.Num() == 0)
    {
        return FReply::Handled();
    }
    for (const TSharedRef<SCheckBox>& CheckBox : CheckBoxesArray){
        if (CheckBox->IsChecked())
        {
            // 如果Check被设置为check 就设置为Uncheck 反之 -- Toggle (切换)
            CheckBox->ToggleCheckedState();
        }
    }
    return FReply::Handled();
}
FReply SAdvanceDeletionTab::OntSeleteAllButtonClicked()
{
    if (CheckBoxesArray.Num() == 0)
    {
        return FReply::Handled();
    }
    for (const TSharedRef<SCheckBox>& CheckBox : CheckBoxesArray){
        if (!CheckBox->IsChecked())
        {
            // 如果Check被设置为check 就设置为Uncheck 反之 -- Toggle (切换)
            CheckBox->ToggleCheckedState();
        }
    }
    return FReply::Handled();
}
```

而关于CheckBoxesArray什么时候添加呢 则是在创建CheckBox的时候将创建的CheckBox存入数组中去

下拉菜单栏的可选设计



基础的设计如下：

```
    //创建ComboxBox在 主界面的Title下面
    TSharedRef<SComboBox<TSharedPtr<FString>>> ConstructComboBox();
    // 下拉菜单栏的可选项
    TArray<TSharedPtr<FString>> ComboBoxSourceItems;
    // ComboBox中每一行的元素创建函数
    TSharedRef<SWidget> OnGenerateComboContent(TSharedPtr<FString> SourceItem);
    // ComboBox中如果选择更改时触发函数
    void OnSelectionChangedComboBox(TSharedPtr<FString>
SelectedOption,ESelectInfo::Type InSelectInfo);
    // 当前菜单栏的选项
    TSharedPtr<STextBlock> ComboDisplayTextBlock;
```

```
TSharedRef<SComboBox<TSharedPtr<FString>>>
SAdvanceDeletionTab::ConstructComboBox()
{
    TSharedRef<SComboBox<TSharedPtr<FString>>> ComboBox =
        SNew(SComboBox<TSharedPtr<FString>>)
        .OptionsSource(&ComboBoxSourceItems)
        .OnGenerateWidget(this,&SAdvanceDeletionTab::OnGenerateComboContent)
        .OnSelectionChanged(this,&SAdvanceDeletionTab::OnSelectionChangedComboBox)
        [
            // base settings
            SAssignNew(ComboDisplayTextBlock,STextBlock)
            .Text(FText::FromString(TEXT("List Assets Option")))
```

```
        ];
    return ComboBox;
}

TSharedRef<SWidget>
SAdvanceDeletionTab::OnGenerateComboContent(TSharedPtr<FString> SourceItem)
{
    TSharedRef<SWidget> ConstructComboText = SNew(STextBlock)
        .Text(FText::FromString(*SourceItem.Get()));
    return ConstructComboText;
}
void SAdvanceDeletionTab::OnSelectionChangedComboBox(TSharedPtr<FString>
SelectedOption,
    ESelectInfo::Type InSelectInfo)
{
    ComboDisplayTextBlock->SetText(FText::FromString(*SelectedOption.Get()));
}
```

ComboBoxSourceItems需要在构造的时候就进行传递值

下拉菜单的选项

如何对于下拉菜单，如何添加下拉的菜单的内容，对于Combobox来讲，每一个页就是一个状态(state) 在AdvanceDeletionWidget.cpp中会有宏定义：

```
#define ListAll TEXT("List All Available Assets")  // 列出所有Asset
#define ListUnused TEXT("List unused Assets")      // 列出Unused的
#define ListSameName TEXT("List Same Name Assets")  // 列出同名的Asset
```

初始化Construct的时候将这些写进一个TArray< FString>中，直接传入到ComboBoxSourceItems中去

对于OnGenerateWidget生成当前行的的信息 对于OnSelectionChanged 表示如果当前选择的状态修改就会触发什么函数。 这里再次重申一下，如果不知道这里传什么函数进去，就点击这个元素如OnSelectionChanged进去看一下

```
void SAdvanceDeletionTab::OnSelectionChangedComboBox(TSharedPtr<FString>
SelectedOption,
    ESelectInfo::Type InSelectInfo)
{
    ComboDisplayTextBlock->SetText(FText::FromString(*SelectedOption.Get()));
    FSuperMangerModule& SuperMangerModule =
FModuleManager::LoadModuleChecked<FSuperMangerModule>(TEXT("SuperManger"));
    // Pass Data for Our Modile to filter
    if (*SelectedOption.Get() == ListAll)
    {
        //List all Asset
        SuperMangerModule.ListAllAssetsForAssetList(DisplayedAssetData);
        RefrshAssetListView();
    }
```

```
    else if (*SelectedOption.Get() == ListUnused)
    {
        //List unused Asset

SuperMangerModule.ListUnusedAssetsForAssetList(AssetsDataUnderSelectedFolderArray,
DisplayedAssetData);
        RefrshAssetListView();
    }
    else if (*SelectedOption.Get() == ListSameName)
    {
        //List all SameName Asset

SuperMangerModule.ListSameNameAssetsForAssetList(AssetsDataUnderSelectedFolderArra
y,DisplayedAssetData);
        RefrshAssetListView();
    }
}
```

筛选unsed的Asset和SameName的Asset

首先对于Unsed的部分筛选部分，我们使用SuperMode中的操作，因为可以通过这个传递Filter之后的内容 AssetDataToFilterArray是传入的数据 OutUnusedAssetData是修正后的内容

```
void FSuperMangerModule::ListUnusedAssetsForAssetList(const
TArray<TSharedPtr<FAssetData>>& AssetDataToFilterArray,
    TArray<TSharedPtr<FAssetData>>& OutUnusedAssetData)
{
    // 先清空之前的可见数据
    OutUnusedAssetData.Empty();
    for (const TSharedPtr<FAssetData>& AssetDataToFilter : AssetDataToFilterArray)
{
        TArray<FString>AssetRef =

UEditorAssetLibrary::FindPackageReferencersForAsset(AssetDataToFilter.Get()-
>ObjectPath.ToString());
        if (AssetRef.Num() == 0){
            OutUnusedAssetData.Add(AssetDataToFilter);
        }
    }
}
```

对于捕获同名，其实可以通过TMultiMap通过AssetName进行捕获，然后进行一个滤波

```
void FSuperMangerModule::ListSameNameAssetsForAssetList(
    const TArray<TSharedPtr<FAssetData>>& AssetDataToFilterArray,
TArray<TSharedPtr<FAssetData>>& OutUnusedAssetData)
{
    OutUnusedAssetData.Empty();
    // multimap for supporting
```

```
    TMultiMap<FString,TSharedPtr<FAssetData>> AssetDataInfo;
    for (const TSharedPtr<FAssetData>& AssetDataToFilter : AssetDataToFilterArray)
    {
        AssetDataInfo.Add(AssetDataToFilter.Get()-
>AssetName.ToString(),AssetDataToFilter);
    }

    for (const TSharedPtr<FAssetData>& DataSharedPtr : AssetDataToFilterArray)
    {
        TArray<TSharedPtr<FAssetData>> AssetDataFromMutil;
        AssetDataInfo.MultiFind(DataSharedPtr.Get()-
>AssetName.ToString(),AssetDataFromMutil);
        if (AssetDataFromMutil.Num() <= 1)
        {
            continue;
        }
        for (const TSharedPtr<FAssetData>& SameAssetData : AssetDataFromMutil)
        {
            if (SameAssetData.IsValid())
            {
                OutUnusedAssetData.AddUnique(SameAssetData);
            }
        }
    }
}
```

这里可能注意到了，这里使用的是DisplayedAssetData而不是一开始在构造AdvanceDeletionWidget中的 存储数据的AssetsDataUnderSelectedFolderArray

这是因为由于滤波的存在，实际上我们需要一个显示在列表中的数组，而并不是直接在原始数组上修改。所以AssetsDataUnderSelectedFolderArray起到 一个存储所有文件的作用，而DisplayedAssetData则是显示列表，也就是我们不同选择下滤波的结果

可用的学习资源

UE5.4 Test Suite : 点击编辑器最下面那一栏中的Trace,选择Unreal Insights。 在Unreal Insights页面中点击"Open Trace"(蓝色按钮) 的右边的“向下”按钮,选择Starship Test Suite即可

更多模组搜索SStarshipGallery就可以看到了

Icon

首先需要将自己下载的图片存储在项目/Plugins/Resource 下 然后创建一个空的C++Class 这里命名为SuperManger/SuperMangerStyle 例如

```
//SuperManger/SuperMangerStyle.h
#include "Styling/SlateStyle.h"
class FSuperMangerStyle
{

};
```

Static

Static 类外 例如DebugHeder中面对多个Cpp文件的重复引用，这里将所有函数变为Static函数避免命名冲突 因为Static函数尽在他所申明的cpp文件中可见，这样就避免了不可见的行为。DebugHeder中面对多个Cpp文件的重复引用 Static 类内 只会初始化一次 并且类内静态成员变量只能在类外初始化，因为不属于类成员，静态成员函数可以不使用实例化就是用 并且类静态成员函数不允许使用类的成员函数 例如：

```
class FSuperMangerStyle
{
public:
    // 注册Icon
    static void InitializeIcons();
    static void ShutDown();
private:
    // 注册使用的name
    static FName StyleSetName;
}
cpp
FName FSuperMangerStyle::StyleSetName =FName("SuperMangerStyle");
```

如何创建一个ICON

1. 创建一个FSlateStyleSet，这这里设置ICON的位置、大小、路径

```
TSharedRef<FSlateStyleSet> FSuperMangerStyle::CreateStyleSet()
{
    TSharedRef<FSlateStyleSet>CustomStyleSet =
        MakeShareable(new FSlateStyleSet(StyleSetName));
    // Icon的存放位置 注意/ 是一个重载 用来合并两个FString
    const FString IconDir =
        IPluginManager::Get().FindPlugin(TEXT("SuperManger"))->GetBaseDir() /
"Resources";
    // 设置Content的目录
    CustomStyleSet->SetContentRoot(IconDir);
    // IconSize
    const FVector2D Icon16x16(16.f,16.f);
    // 注册Icon 通过这个name作为FSlateIcon的获取管控
    CustomStyleSet->Set("ContentBrowser.DeletUnsedAsseets",
        new FSlateImageBrush(IconDir / "IconQ.jpg",Icon16x16));
    CustomStyleSet->Set("ContentBrowser.DeletEmptyAsseets",
        new FSlateImageBrush(IconDir / "IconFX.jpg",Icon16x16));
    CustomStyleSet->Set("ContentBrowser.AdvancDeletion",
        new FSlateImageBrush(IconDir / "IconDD.png",Icon16x16));
    return CustomStyleSet;
}
```

2. 将创建的FSlateStyleSet注册到FSlateStyleRegistry

```
FName FSuperMangerStyle::StyleSetName =FName("SuperMangerStyle");
TSharedPtr<FSlateStyleSet> FSuperMangerStyle::CreatedSlateStyleSet = nullptr;
void FSuperMangerStyle::InitializeIcons()
{
    if (CreatedSlateStyleSet.IsValid() == false)
    {
        CreatedSlateStyleSet = CreateStyleSet();
        FSlateStyleRegistry::RegisterSlateStyle(*CreatedSlateStyleSet.Get());
    }
}
```

3. 在通过注册名字和Instylename捕获ICON

```
void FSuperMangerModule::AddCBMenuEntry(FMenuBuilder& MenuBuilder)--
    MenuBuilder.AddMenuEntry(
        FText::FromString(TEXT("Delete Unsed Assets")),
        FText::FromString(TEXT("Safely delete all unused assets under folder")),

FSlateIcon(FSuperMangerStyle::GetStyleSetName(),"ContentBrowser.DeletUnsedAsseets"
),

FExecuteAction::CreateRaw(this,&FSuperMangerModule::OnDeleteUnusedButtonClicked)
        );
    // 添加第二项菜单 删除空文件夹
    MenuBuilder.AddMenuEntry(
        FText::FromString(TEXT("Delete Empty Folders")),
        FText::FromString(TEXT("Safely Delete All Empty Folders")),

FSlateIcon(FSuperMangerStyle::GetStyleSetName(),"ContentBrowser.DeletEmptyAsseets"
),

FExecuteAction::CreateRaw(this,&FSuperMangerModule::OnDeleteEmptyFolderButtonClick
ed)
    );
    // 添加第三项菜单 资产阅读界面，通过这个界面可以选择资产、删除资产
    MenuBuilder.AddMenuEntry(
        FText::FromString(TEXT("Advanc Deletion")),
        FText::FromString(TEXT("List Asset by Specific in a tab for delete")),

FSlateIcon(FSuperMangerStyle::GetStyleSetName(),"ContentBrowser.AdvancDeletion"),

FExecuteAction::CreateRaw(this,&FSuperMangerModule::OnAdvanceDeletionButtonClicked
)
    );
```

可以看到在绑定name的时候，实际上第一项是FSuperMangerStyle的name 而第二个则是注册ICon的name 最后的最后别忘了更新注册取消的内容

```
void FSuperMangerStyle::ShutDown()
{
    if(CreatedSlateStyleSet.IsValid())
    {
        FSlateStyleRegistry::UnRegisterSlateStyle(*CreatedSlateStyleSet.Get());
        CreatedSlateStyleSet.Reset();
    }
}
```

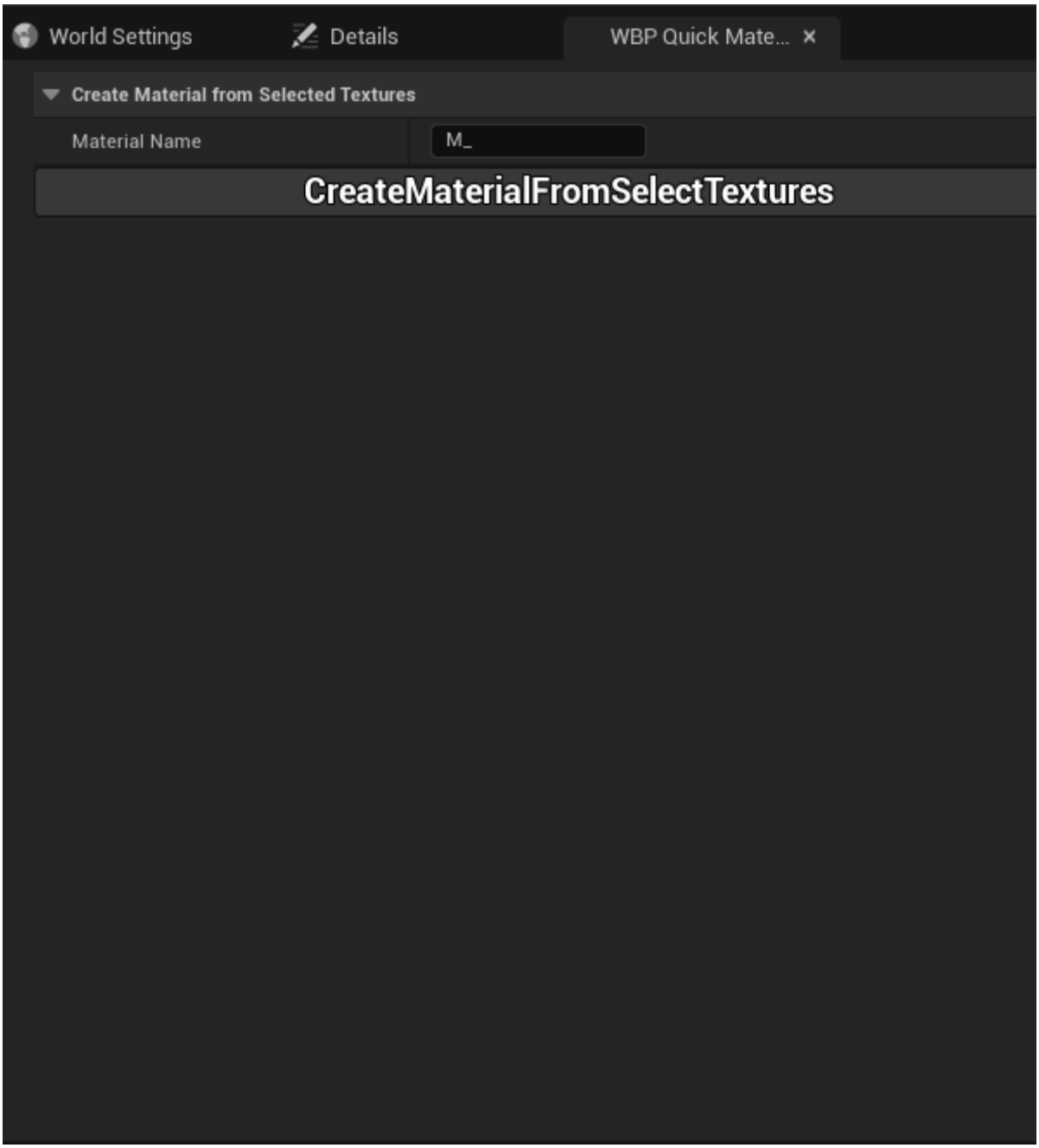
在FSuperMangerModule::ShutdownModule调用

```
void FSuperMangerModule::StartupModule()
{
    // This code will execute after your module is loaded into memory; the exact
timing is specified in the .uplugin file per-module

    //注册命令:
    FTestCommandsLineList::Register();
    FSuperMangerStyle::InitializeIcons();
    InitCBMenuExtention();
    RegisterAdvanceDeletionTab();
}
```

材质创建

基础widget的创建

首先创建基于EditorUtilityWidget的C++ Class 这里命名为QuickMaterialCreationWidget 并创建对应的蓝图类WBP_QuickMaterialCreation 注意这里是创建EditorWidget的蓝图

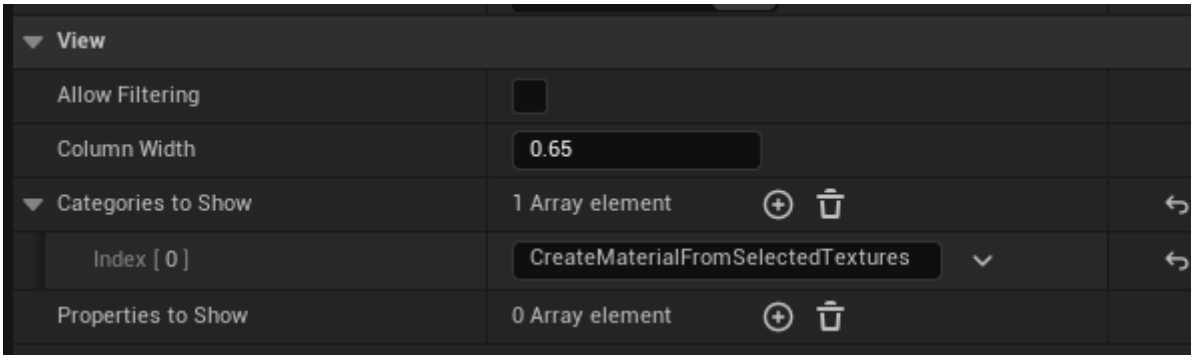


如上图所示

我们需要设计一个如图所示的窗口 关于这个窗口的设计参考课程链接的p58 这里不再赘述

但是这里有一个细节的点，是我们需要QuickMaterial这个DetialsView显示我们的class的数据。在蓝图中设置他的categories to show，名字为在class中设置的catergory的名字

```
QuickMaterialCreationWidget.h
UPROPERTY(EditAnywhere,
BlueprintReadWrite,Category="CreateMaterialFromSelectedTextures",meta =
(EditCondition="bCustomMaterialName"))
    FString MaterialName = TEXT("M_");
```



如何按下按钮就创建材质

bCustomMaterialName -- 是否使用了自己的定义的名字 MaterialName -- 定义的材质的名称

首先定义一个蓝图中可调用函数CreateMaterialFromSelectedTextures，在蓝图中对button创建click事件，每次点击就会触发这个函数

在创建材质之前首先要检查：

1. 选择的路径下是否有Texture文件 参数1 : 获取点击文件到的全部Asset 参数2 : 过滤后得到的UTexture2D* 参数3 UTexture2D*的路径 返回是否当前TArray< FAssetData>中找到了材质资产

```
bool UQuickMaterialCreationWidget::ProcessSelectedData(const TArray<FAssetData>&
SelectedDataToProcess,
    TArray<UTexture2D*>& SelectedTextureArray, FString&
OutSelectedTextureFolderPath)
{
    if (SelectedDataToProcess.Num() == 0)
    {
        DebugHeader::ShowMessageDialog(EAppMsgType::Type::Ok,TEXT("No Texture
Selected"));
        return false;
    }
    // OutSelectedTextureFolderPath是否被设置
    bool bMaterialNameSets = false;
    for (const FAssetData& SelectedAssetData : SelectedDataToProcess)
    {
        UObject* SelectedAsset = SelectedAssetData.GetAsset();
        if (!SelectedAsset) continue;
        UTexture2D* SelectedTexture = Cast<UTexture2D>(SelectedAsset);
        if(!SelectedTexture)
        {
            DebugHeader::ShowMessageDialog(EAppMsgType::Type::Ok,
                TEXT("Please Selected only textures") + SelectedAsset->GetName() +
```

```
TEXT(" is not a texture"));
            return false;
        }
        SelectedTextureArray.Add(SelectedTexture);
        if (OutSelectedTextureFolderPath.IsEmpty())
        {
            OutSelectedTextureFolderPath =
SelectedAssetData.PackagePath.ToString();
        }
        // 如果不是自定义名字 就会默认做这个工作 更换MaterialName为原始的名字
        if (!bCustomMaterialName && !bMaterialNameSets)
        {
            MaterialName = SelectedTexture->GetName();
            // 更换前缀
            MaterialName.RemoveFromStart(TEXT("T_"));
            MaterialName.InsertAt(0, TEXT("M_"));
            bMaterialNameSets = true;
        }
    }
    return true;
}
```

2. 选择的名字是否已经存在

```
bool UQuickMaterialCreationWidget::CheckIsNameUsed(const FString&
FolderPathToCheck, const FString& MaterialNameToCheck)
{
    // 这将是否递归检查取消 因为我们只希望当前文件下不存在重复名称
    // ListAssets 返回值是Folder+AssetName 例如
/Game/_Game/SuperMangerBP/BP_Actor.BP_Actor
    TArray<FString> AssetsPathInFolder =
UEditorAssetLibrary::ListAssets(FolderPathToCheck,false);
    for (const FString& AssetPath : AssetsPathInFolder)
    {
        // 去除前面的FolderName 仅仅返回基础名称
        FString ExitingAssetName = FPaths::GetBaseFilename(AssetPath);
        if (ExitingAssetName == MaterialNameToCheck)
        {
            DebugHeader::ShowMessageDialog(EAppMsgType::Type::Ok,TEXT("Material
already exists"));
            return true;
        }
    }
    return false;
}
```

3. 创建材质 这里使用FAssetToolsModule创建材质

```
UMaterial* UQuickMaterialCreationWidget::CreateMaterial(const FString&
NameOfMaterial, const FString& PathToPutMaterial)
{
    FAssetToolsModule& AssetrToolsModule =
FModuleManager::LoadModuleChecked<FAssetToolsModule>("AssetTools");
    UMaterialFactoryNew* MaterialFactory =  NewObject<UMaterialFactoryNew>();
    UObject* CreatedNewObject =
AssetrToolsModule.Get().CreateAsset(NameOfMaterial,PathToPutMaterial,
        UMaterial::StaticClass(),MaterialFactory);
    return Cast<UMaterial>(CreatedNewObject);
}
```

创建材质之前如和预设节点

对于材质，会有许多节点例如颜色、粗糙度、不透明度、金属材质程度等。所以对材质的命名需要进行一些统一,方便进行搜索 之后根据检测的到的材质的名称和之前默认设置的内容(SupportTextureNames) 检索，判断UTexture2D应当属于哪一个PinsConnectedCounter

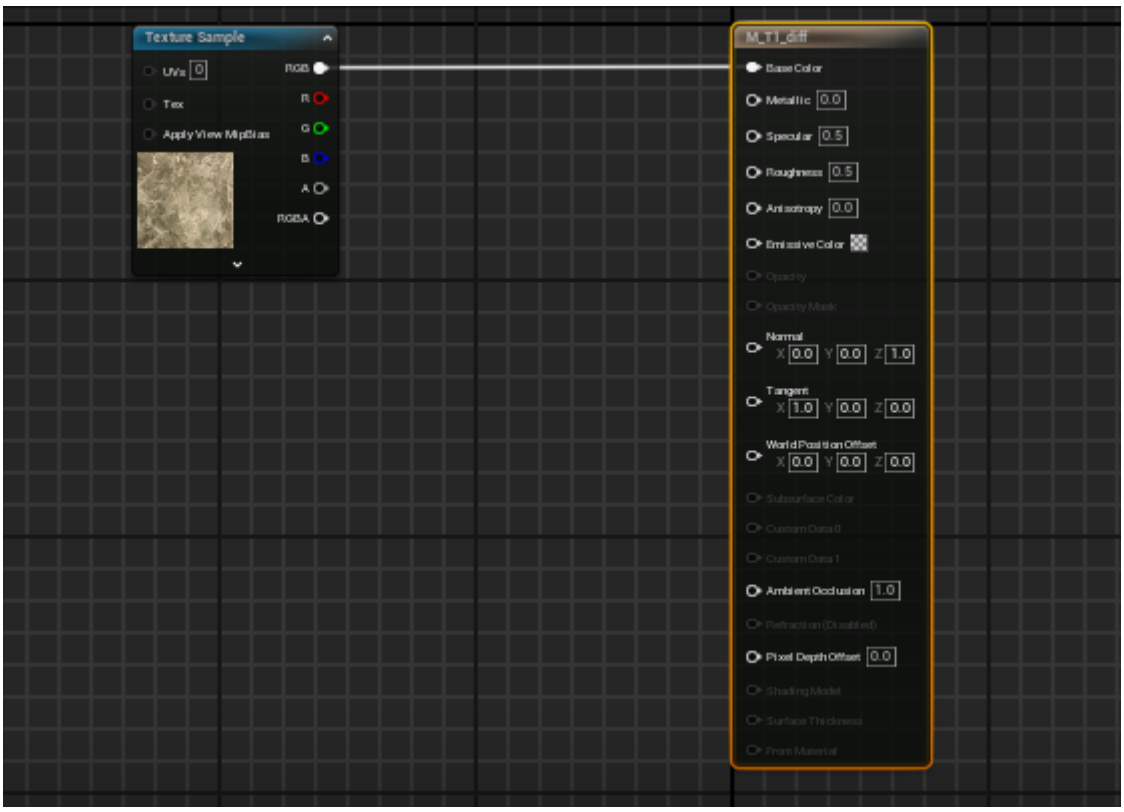
在材质class中，每一个参数都是一个UMaterialExpression的class

```
Material.h
    /** Used to detect duplicate parameters.  Does not contain parameters in
referenced functions! */
    TMap<FName, TArray<UMaterialExpression*> > EditorParameters;
```

对于不同名称的Texture，我们使用的方法是逐一将其和SupportTextureNames中的数组遍历比较，判断属于哪一个数组名称 之后就将这个Texture链接到创建的材质中 例如:

```
void UQuickMaterialCreationWidget::Defaule_CreateMaterialNodes(UMaterial*
CreatedMaterial, UTexture2D* SelelectTexture,
    uint32& PinsConnectedCounter)
{
    // UMaterialExpressionTextureSample 每一个添加的节点都是一个UMaterialExpression
对象
    // 这里创建一个纹理采样材质
    UMaterialExpressionTextureSample* TextureSampleNode =
                    NewObject<UMaterialExpressionTextureSample>(CreatedMaterial);
    if (!TextureSampleNode)
    {
        return;
    }
    // 检测创建的材质是不是已经有color材质链接了
    if (!CreatedMaterial->HasBaseColorConnected())
    {
        if
(TryConnectBaseColor(TextureSampleNode,SelelectTexture,CreatedMaterial))
        {
            PinsConnectedCounter++;
```

```
            return;
        }
    }
}
bool
UQuickMaterialCreationWidget::TryConnectBaseColor(UMaterialExpressionTextureSample
* TextureSampleNode,
    UTexture2D* SelelectTexture, UMaterial* CreatedMaterial)
{
    //和SupportTextureNames中的数组遍历
    for (const FString& BaseColorName : BaseColorArray)
    {
        // Connect pins to base color
        if (SelelectTexture->GetName().Contains(BaseColorName))
        {
            TextureSampleNode->Texture = SelelectTexture;
            CreatedMaterial-
>GetExpressionCollection().AddExpression(TextureSampleNode);
            CreatedMaterial->GetExpressionInputForProperty(MP_BaseColor)-
>Connect(0,TextureSampleNode);
            // 更新依赖关系
            CreatedMaterial->PostEditChange();
            // 调整这个node在蓝图中的位置
            TextureSampleNode->MaterialExpressionEditorX = -600;
            return true;
        }
    }
    return false;
}
```



同样的方式我们可以创建和Metail等不同的材质链接的code 例如

```
Defaule_CreateMaterialNodes--
    if(!CreatedMaterial->HasMetallicConnected())
    {
        if (TryConnectMetallic(TextureSampleNode,SelelectTexture,CreatedMaterial))
        {
            PinsConnectedCounter++;
            return;
        }
    }

bool
UQuickMaterialCreationWidget::TryConnectMetallic(UMaterialExpressionTextureSample*
TextureSampleNode,
    UTexture2D* SelelectTexture, UMaterial* CreatedMaterial)
{
    // 和SupportTextureNames中的数组BaseColorArray遍历 判断是否在BaseColorArray中 如
果是就链接BaseColor
    for (const FString& MetallicName : MetallicArray)
    {
        // Connect pins to base color
        if (SelelectTexture->GetName().Contains(MetallicName))
        {
            SelelectTexture->CompressionSettings = TC_Default;
            SelelectTexture->SRGB = false;
            SelelectTexture->PostEditChange();
            //材质的欸外设置
            TextureSampleNode->Texture = SelelectTexture;
            TextureSampleNode->SamplerType = SAMPLERTYPE_LinearColor;
            //创建对应的node并链接
            CreatedMaterial-
>GetExpressionCollection().AddExpression(TextureSampleNode);
            CreatedMaterial->GetExpressionInputForProperty(MP_Metallic)-
>Connect(0,TextureSampleNode);
            CreatedMaterial->PostEditChange();
            // 调整这个node在蓝图中的位置
            TextureSampleNode->MaterialExpressionEditorX -= 600;
            TextureSampleNode->MaterialExpressionEditorY += 240;
            return true;
        }
    }
    return false;
}
```

此外还有有个比较特殊的arm 其rgb对应不同的pin 如r对应ambient g对应Metaillic b对应Roughness 此时的需要一些欸外的设置

```
bool UQuickMaterialCreationWidget::TryConnectORM(UMaterialExpressionTextureSample*
TextureSampleNode,
```

```
    UTexture2D* SelelectTexture, UMaterial* CreatedMaterial)
{
    for (const FString& ORMName : ORMArray)
    {
        if (SelelectTexture->GetName().Contains(ORMName))
        {
            // 这里设置为masks
            SelelectTexture->CompressionSettings = TC_Masks;
            SelelectTexture->SRGB = false;
            SelelectTexture->PostEditChange();

            TextureSampleNode->Texture = SelelectTexture;
            TextureSampleNode->SamplerType = SAMPLERTYPE_Masks;
            // 这里通过index不同 创建不同pins
            CreatedMaterial-
>GetExpressionCollection().AddExpression(TextureSampleNode);
            // 注意第一个参数不同 表示在蓝图中可见的那个id
            CreatedMaterial->GetExpressionInputForProperty(MP_AmbientOcclusion)-
>Connect(1,TextureSampleNode);  //r
            CreatedMaterial->GetExpressionInputForProperty(MP_Roughness)-
>Connect(2,TextureSampleNode);   //g
            CreatedMaterial->GetExpressionInputForProperty(MP_Metallic)-
>Connect(3,TextureSampleNode);   //b

            TextureSampleNode->MaterialExpressionEditorX -= 600;
            TextureSampleNode->MaterialExpressionEditorY += 960;
            return true;
        }
    }
    return false;
}
```

注意这里通过下标设置RGB的位置 此外我们使用一个enum作为是否使用arm的标记

```
void UQuickMaterialCreationWidget::CreateMaterialFromSelectedTextures()
...
switch (ChannelPackingType)
        {
            case E_ChannelPackingType::ECPT_NoChannelPacking:

Defaule_CreateMaterialNodes(CreatedMaterial,SelectedTexture,PinsConnectedCounter);
                break;
            case E_ChannelPackingType::ECPT_ORM:

ORM_CreateMaterialNodes(CreatedMaterial,SelectedTexture,PinsConnectedCounter);
                break;
            case E_ChannelPackingType::ECPT_Max:
                break;
            default:
                break;
        }
void UQuickMaterialCreationWidget::ORM_CreateMaterialNodes(UMaterial*
```

```
CreatedMaterial, UTexture2D* SelelectTexture,
    uint32& PinsConnectedCounter)
{
    // UMaterialExpressionTextureSample 每一个添加的节点都是一个UMaterialExpression
对象
    // 这里创建一个纹理采样材质
    UMaterialExpressionTextureSample* TextureSampleNode =
                   NewObject<UMaterialExpressionTextureSample>(CreatedMaterial);
    if (!TextureSampleNode)
    {
        return;
    }
    // 检测创建的材质是不是已经有color材质链接了
    if (!CreatedMaterial->HasBaseColorConnected())
    {
        if
(TryConnectBaseColor(TextureSampleNode,SelelectTexture,CreatedMaterial))
        {
            PinsConnectedCounter++;
            return;
        }
    }
    if (!CreatedMaterial->HasNormalConnected())
    {
        if (TryConnectNormal(TextureSampleNode,SelelectTexture,CreatedMaterial))
        {
            PinsConnectedCounter++;
            return;
        }
    }
    // 此处就不一样了 应为其他三个pin我们将其链接成一个rgb的arm_Texture
    if (!CreatedMaterial->HasRoughnessConnected())
    {
        if (TryConnectORM(TextureSampleNode,SelelectTexture,CreatedMaterial))
        {
            PinsConnectedCounter += 3;
            return;
        }
    }
    DebugHeader::ShowMessageDialog(EAppMsgType::Type::Ok,
            TEXT("Fail to connect the texture: ") + SelelectTexture->GetName());
}
```

创建Material的同时创建material的实例

在CreateMaterialFromSelectedTextures中创建一个蓝图可见的bool值判断是否需要创建material的同时创建材质实例 如果是就调用

```
UMaterialInstanceConstant*
UQuickMaterialCreationWidget::CreateMaterialInstance(UMaterial* CreatedMaterial,
    const FString& NameOfMaterial,const FString& PathToPutMaterial)
```

```
{
    MaterialName.RemoveFromStart("M_");
    MaterialName.InsertAt(0, "MI_");

    FAssetToolsModule& AssetToolsModule =
FModuleManager::LoadModuleChecked<FAssetToolsModule>("AssetTools");
    UMaterialInstanceConstantFactoryNew* MaterialInstanceFactory =
NewObject<UMaterialInstanceConstantFactoryNew>();
    MaterialInstanceFactory->InitialParent = CreatedMaterial;
    UObject* MaterialInstance =
AssetToolsModule.Get().CreateAsset(NameOfMaterial,PathToPutMaterial,
        UMaterialInstanceConstant::StaticClass(),MaterialInstanceFactory);
    if (UMaterialInstanceConstant* CreateMaterialInstance =
Cast<UMaterialInstanceConstant>(MaterialInstance))
    {
        // 更新依赖关系
        CreatedMaterial->PostEditChange();
        CreateMaterialInstance->PostEditChange();
        return CreateMaterialInstance;
    }
    return nullptr;
}
```

注意创建材质实例需要更新引用PostEditChange

基类分别是AsssetActionUtilit和ActorActionUtility

Subsystem

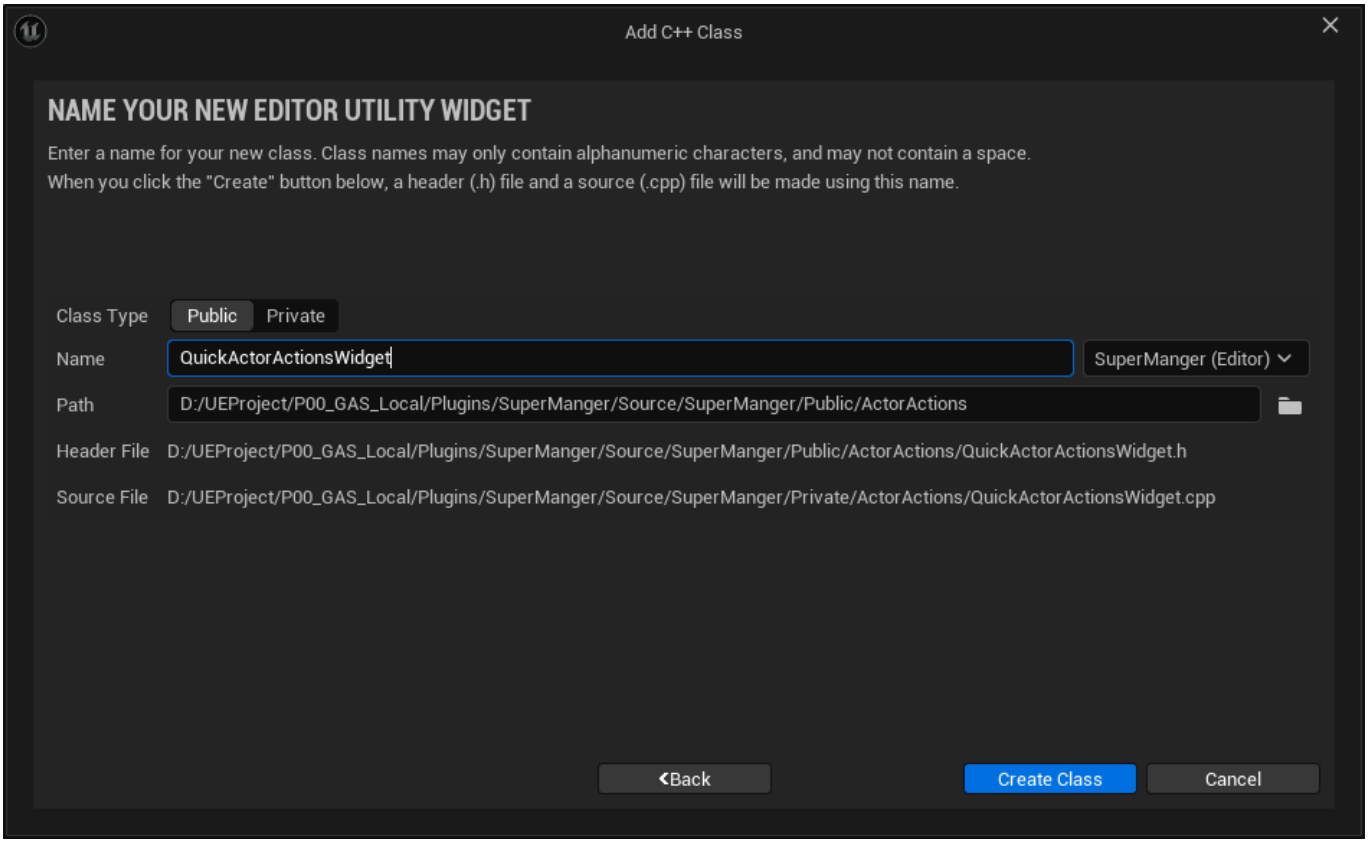
这里首先会介绍4中subsystem

1. UEigenSubsystem -- 和整个引擎声明周期相关的子系统，在整个引擎运行期间是全局的，日志、全局设计等
2. UEditorSubsystem -- 专门为编辑器（Editor）功能服务的子系统。生命周期与编辑器相关联。用于扩展 Unreal Editor 的功能，例如工具创建、自定义编辑器行为、编辑器插件。
3. UGameInstanceSubsystem -- 与游戏实例（UGameInstance）绑定的子系统。生命周期与 UGameInstance 相同。 适合用于游戏范围内的功能，跨关卡有效。管理跨关卡的逻辑，例如成就系统、存档系统、在线服务等。
4. ULocalPlayerSubsystem -- 生命周期与 ULocalPlayer（本地玩家）绑定。玩家特定的设置或状态管理

在这里主要使用的是UEditorSubsystem 针对Actor 则是UEditorActorSubsystem

基础widget创建

创建C++ class



注意如果要在项目中使用UEditorActorSubsystem 需要在build.cs中引入ModuleName:UnrealEd

widget的创建和前面材质窗口创建类似

ActorName ? ActorLabel

首先和前面提到的AssetName不同。在Level中使用的Actor的name是定义在ActorLabel中的

查询相似label的材质

```
void UQuickActorActionsWidget::SeletctAllActorsWithSimilarName()
{
    if(!GetEditorActorSubsystem())
    {
        return;
    }
    //Find all loaded Actors that are selected in the world editor
    TArray<AActor*>SelectedActors = EditorActorSubsystem-
>GetSelectedLevelActors();
    uint32 SelectedCounter = 0;
    if (SelectedActors.Num() == 0)
    {
        DebugHeader::ShowNotifyInfo(TEXT("No Actors Selected"));
        return;
    }
    if(SelectedActors.Num() > 1)
    {
        DebugHeader::ShowNotifyInfo(TEXT("You can only Selected one Actors"));
        return;
```

```
    }
    FString SelectedActorName = SelectedActors[0]->GetActorLabel();   // SM_Chair
    const FString NameToSearch = SelectedActorName.LeftChop(4);       // SM_C

    TArray<AActor*>AllLevelActors = EditorActorSubsystem->GetAllLevelActors();
    for (AActor* ActorInLevel : AllLevelActors)
    {
        if (!ActorInLevel)
        {
            continue;
        }
        if (ActorInLevel->GetActorLabel().Contains(NameToSearch,SearchCase))
        {
            EditorActorSubsystem->SetActorSelectionState(ActorInLevel,true);
            SelectedCounter+=1;
        }
    }
    if (SelectedCounter > 0)
    {
        DebugHeader::ShowNotifyInfo(TEXT("Successfully Selected ") +
FString::FromInt(SelectedCounter) + TEXT(" Actors"));
    }
    else
    {
        DebugHeader::ShowNotifyInfo(TEXT("No Actors With Same name found"));
    }
}
```

简单的一个逻辑就是从选中的actor的lable中捕获前4个字母，之后和GetAllLevelActors进行比对

复制Actor

```
void UQuickActorActionsWidget::DuplicateActors()
{
    if (!GetEditorActorSubsystem())
    {
        return;
    }
    // 获取选择的SelectedActors
    TArray<AActor*> SelectedActors = EditorActorSubsystem-
>GetSelectedLevelActors();
    uint32 DuplicateCounter = 0;
    if (SelectedActors.Num() == 0)
    {
        DebugHeader::ShowNotifyInfo(TEXT("No Actors Selected"));
        return;
    }
    if (NumberOfDuplicate <= 0 || OffsetDist == 0)
    {
        DebugHeader::ShowNotifyInfo(TEXT("Did not correct NumberOfDuplicate or
OffsetDist"));
```

```
        return;
    }
    for (AActor* SelectedActor : SelectedActors)
    {
        if (!SelectedActor)
        {
            continue;
        }
        for ( int32 i=0;i<NumberOfDuplicate;++i)
        {
            AActor* DuplicateActor = EditorActorSubsystem-
>DuplicateActor(SelectedActor,SelectedActor->GetWorld());
            if (!DuplicateActor)
            {
                continue;
            }
            const float DuplicateOffSetDist = (i+1) * OffsetDist;
            switch (AixForDuoDuplicationAixs)
            {
                case E_DuplicationAixs::EDA_XAixs:
                    DuplicateActor->AddActorWorldOffset(
                        FVector(DuplicateOffSetDist,0.f,0.f));
                    break;
                case E_DuplicationAixs::EDA_YAixs:
                    DuplicateActor->AddActorWorldOffset(
                        FVector(0.f,DuplicateOffSetDist,0.f));
                    break;
                case E_DuplicationAixs::EDA_ZAixs:
                    DuplicateActor->AddActorWorldOffset(
                        FVector(0.f,0.f,DuplicateOffSetDist));
                    break;
                default:
                    break;
            }
            EditorActorSubsystem->SetActorSelectionState(DuplicateActor,false);
            DuplicateCounter += 1;
        }
        if (DuplicateCounter > 0)
        {
            DebugHeader::ShowNotifyInfo(TEXT("Successfully Duplicated " +
FString::FromInt(DuplicateCounter)+" Actors"));
        }
    }
}
```

随机旋转的设置

其实就是设置选中Actor的WorldRotation 这里使用一个结构体存储设置旋转的最大最小值，通过最大最小这设置旋转角度

```
USTRUCT(BlueprintType)
struct FRandomActionRotation
{
    GENERATED_BODY()
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    bool bRandomRotationYaw = false;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,meta =(EditCondition =
"bRandomRotationYaw"))
    float fRYawMax = -45.f;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,meta =(EditCondition =
"bRandomRotationYaw"))
    float fRYawMin = 45.f;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    bool bRandomRotationPitch = false;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,meta =(EditCondition =
"bRandomRotationPitch"))
    float fRPitchMax = -45.f;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,meta =(EditCondition =
"bRandomRotationPitch"))
    float fRPitchMin = 45.f;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    bool bRandomRotationRoll = false;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,meta =(EditCondition =
"bRandomRotationRoll"))
    float fRRollMax = -45.f;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,meta =(EditCondition =
"bRandomRotationRoll"))
    float fRRollMin = 45.f;
};

void UQuickActorActionsWidget::RandomActionsTransform()
{
    // 判断是否设置
    const bool ConditionNoSet = RandomActionRotation.bRandomRotationPitch
                                && RandomActionRotation.bRandomRotationYaw
                                && RandomActionRotation.bRandomRotationRoll;
    if (!GetEditorActorSubsystem())
    {
        return;
    }
    if(!ConditionNoSet)
    {
        DebugHeader::ShowNotifyInfo(TEXT("Random Actions NoSet"));
        return;
    }
    TArray<AActor*> SelectedActors = EditorActorSubsystem-
>GetSelectedLevelActors();
    if (SelectedActors.Num() == 0)
    {
        DebugHeader::ShowNotifyInfo(TEXT("No Actors Selected"));
    }
```

```
    for (AActor* SeletctActor : SelectedActors)
    {
        if (!SeletctActor)
        {
            continue;
        }
        FRotator RandomRotator =  FRotator(0,0,0);
        if (RandomActionRotation.bRandomRotationYaw)
        {
            RandomRotator.Yaw =
FMath::RandRange(RandomActionRotation.fRYawMin,RandomActionRotation.fRYawMax);
        }
        if (RandomActionRotation.bRandomRotationPitch)
        {
            RandomRotator.Pitch =
FMath::RandRange(RandomActionRotation.fRPitchMin,RandomActionRotation.fRPitchMax);
        }
        if (RandomActionRotation.bRandomRotationRoll)
        {
            RandomRotator.Roll =
FMath::RandRange(RandomActionRotation.fRRollMin,RandomActionRotation.fRRollMax);
        }
        SeletctActor->AddActorWorldRotation(RandomRotator);
    }
    DebugHeader::ShowNotifyInfo(TEXT("Successfully Set ") +
FString::FromInt(SelectedActors.Num()) + TEXT(" Actors"));
    return;
}
```

更近一步 对transform的随机和复位

其实就是对struct中增加关于offset和scale的设计

```
struct FRandomActionRotation
{
    GENERATED_BODY()
#pragma region RandomRotation
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    bool bRandomRotationYaw = false;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,meta =(EditCondition =
"bRandomRotationYaw"))
    float fRYawMax = -45.f;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,meta =(EditCondition =
"bRandomRotationYaw"))
    float fRYawMin = 45.f;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    bool bRandomRotationPitch = false;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,meta =(EditCondition =
"bRandomRotationPitch"))
```

```
    float fRPitchMax = -45.f;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,meta =(EditCondition =
"bRandomRotationPitch"))
    float fRPitchMin = 45.f;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    bool bRandomRotationRoll = false;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,meta =(EditCondition =
"bRandomRotationRoll"))
    float fRRollMax = -45.f;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,meta =(EditCondition =
"bRandomRotationRoll"))
    float fRRollMin = 45.f;
# pragma endregion


# pragma region RandomScale
    UPROPERTY(EditAnywhere, BlueprintReadWrite,Category="RandomActionTransform")
    bool bRandomScale = false;
    UPROPERTY(EditAnywhere,
BlueprintReadWrite,Category="RandomActionTransform",meta =(EditCondition =
"bRandomScale"))
    float ScaleMin = 0.8f;
    UPROPERTY(EditAnywhere,
BlueprintReadWrite,Category="RandomActionTransform",meta = (EditCondition =
"bRandomScale"))
    float ScaleMax = 2.8f;
# pragma endregion
# pragma region RandomOffset
    UPROPERTY(EditAnywhere, BlueprintReadWrite,Category="RandomActionTransform")
    bool bRandomOffest = false;
    UPROPERTY(EditAnywhere,
BlueprintReadWrite,Category="RandomActionTransform",meta = (EditCondition =
"bRandomOffest"))
    float OffestMin = -50.f;
    UPROPERTY(EditAnywhere,
BlueprintReadWrite,Category="RandomActionTransform",meta =(EditCondition =
"bRandomOffest"))
    float OffestMax = 50.f;
# pragma endregion RandomOffset
};
# pragma endregion RandomActionTransform
void UQuickActorActionsWidget::RandomActionsTransform()
{
    // 判断是否设置
    const bool bConditionNoSet = RandomActionTransform.bRandomRotationPitch
                                || RandomActionTransform.bRandomRotationYaw
                                || RandomActionTransform.bRandomRotationRoll
                                || RandomActionTransform.bRandomScale
                                || RandomActionTransform.bRandomOffest;
    if (!GetEditorActorSubsystem())
    {
        return;
    }
```

```
    if(!bConditionNoSet)
    {
        DebugHeader::ShowNotifyInfo(TEXT("Random Actions NoSet"));
        return;
    }
    TArray<AActor*> SelectedActors = EditorActorSubsystem-
>GetSelectedLevelActors();
    if (SelectedActors.Num() == 0)
    {
        DebugHeader::ShowNotifyInfo(TEXT("No Actors Selected"));
    }

    for (AActor* SeletctActor : SelectedActors)
    {
        if (!SeletctActor)
        {
            continue;
        }
        // for rotation
        FRotator RandomRotator =  FRotator(0,0,0);
        if (RandomActionTransform.bRandomRotationYaw)
        {
            RandomRotator.Yaw =
FMath::RandRange(RandomActionTransform.fRYawMin,RandomActionTransform.fRYawMax);
        }
        if (RandomActionTransform.bRandomRotationPitch)
        {
            RandomRotator.Pitch =
FMath::RandRange(RandomActionTransform.fRPitchMin,RandomActionTransform.fRPitchMax
);
        }
        if (RandomActionTransform.bRandomRotationRoll)
        {
            RandomRotator.Roll =
FMath::RandRange(RandomActionTransform.fRRollMin,RandomActionTransform.fRRollMax);
        }
        SeletctActor->AddActorWorldRotation(RandomRotator);

        // for scale
        if (RandomActionTransform.bRandomScale)
        {
            const float RandomScale =
FMath::RandRange(RandomActionTransform.ScaleMin,RandomActionTransform.ScaleMax);
            SeletctActor-
>SetActorScale3D(FVector(RandomScale,RandomScale,RandomScale));
        }

        // for offset
        if (RandomActionTransform.bRandomOffest)
        {
            const float RandomOffset =
FMath::RandRange(RandomActionTransform.OffestMin,RandomActionTransform.OffestMax);
            SeletctActor-
>AddActorWorldOffset(FVector(RandomOffset,RandomOffset,RandomOffset));
```

```
                RandomActionOffsetMap.Add({SeletctActor,RandomOffset});
            }
        }
        DebugHeader::ShowNotifyInfo(TEXT("Successfully Set ") +
FString::FromInt(SelectedActors.Num()) + TEXT(" Actors"));
        return;
}

void UQuickActorActionsWidget::ResetRandomActionsTransform()
{
        if (!GetEditorActorSubsystem())
        {
            return;
        }
        TArray<AActor*> SelectedActors = EditorActorSubsystem-
>GetSelectedLevelActors();
        if (SelectedActors.Num() == 0)
        {
            DebugHeader::ShowNotifyInfo(TEXT("No Actors Selected"));
        }
        FRotator ResetRandomRotator =  FRotator(0,0,0);
        FVector  ResetRandomScale = FVector(1,1,1);
        FVector  ResetRandomOffset = FVector(1,1,1);
        for (AActor* SeletctActor : SelectedActors)
        {
            if (!SeletctActor)
            {
                continue;
            }
            if (!SeletctActor->SetActorRotation(ResetRandomRotator))
            {
                DebugHeader::ShowNotifyInfo(TEXT("error to set rotation"));
            }
            SeletctActor->SetActorScale3D(ResetRandomScale);
            if (RandomActionOffsetMap.Find(SeletctActor)!= nullptr)
            {
                const float RandomOffset = RandomActionOffsetMap[SeletctActor];
                ResetRandomOffset.X = -RandomOffset;
                ResetRandomOffset.Y = -RandomOffset;
                ResetRandomOffset.Z = -RandomOffset;
                SeletctActor->AddActorWorldOffset(ResetRandomOffset);
            }
        }
}
```

关卡编辑器

注册

这里和之前在InitCBMenuExtention中注册asset操作界面一样 需要注册三个委托绑定

1. 获取LevelEditor的管理，添加新的委托操作

```
    void FSuperMangerModule::InitLevelEditorMenuExtension()
{
    // 获取LevelEditorManger
    FLevelEditorModule& LevelEditorModule =
        FModuleManager::LoadModuleChecked<FLevelEditorModule>
(TEXT("LevelEditor"));

    TSharedRef<FUICommandList> ExitingUICommands =
LevelEditorModule.GetGlobalLevelEditorActions();
    ExitingUICommands->Append(CustomUICommands.ToSharedRef());

    // 获取当前的LevelMenu的菜单列表
    TArray<FLevelEditorModule::FLevelViewportMenuExtender_SelectedActors>&
LevelEditorExtenders =
        LevelEditorModule.GetAllLevelViewportContextMenuExtenders();
    // 以委托的方式添加新菜单
    FLevelEditorModule::FLevelViewportMenuExtender_SelectedActors NewRawMenu =

FLevelEditorModule::FLevelViewportMenuExtender_SelectedActors::CreateRaw(this,&FSu
perMangerModule::OnCustomLevelEditorMenuExtender);
    LevelEditorExtenders.Add(NewRawMenu);
}
```

2. 在列表中设置事件的位置以及触发的函数

```
TSharedRef<FExtender> FSuperMangerModule::OnCustomLevelEditorMenuExtender(
    const TSharedRef<FUICommandList> UICommandList, const TArray<AActor*>
SelectedActors)
{
    TSharedRef<FExtender> MenuExtender =  MakeShareable(new FExtender());
    if (SelectedActors.Num() > 0)
    {
        MenuExtender->AddMenuExtension(
            FName("ActorOptions"),
            EExtensionHook::Before,
            UICommandList,

FMenuExtensionDelegate::CreateRaw(this,&FSuperMangerModule::AddLevelEditorMenuEntr
y)
        );
    }
    return MenuExtender;
}
```

3. 通过FMenuBuilder创建对应的事件操作

```
void FSuperMangerModule::AddLevelEditorMenuEntry(FMenuBuilder& MenuBuilder)
{
```

```
    MenuBuilder.AddMenuEntry(
        FText::FromString(TEXT("Lock Actor Selection")),
        FText::FromString(TEXT("Prevent Actor from being Selected")),

FSlateIcon(FSuperMangerStyle::GetStyleSetName(),"ActorBrowser.LockActors"),

FExecuteAction::CreateRaw(this,&FSuperMangerModule::OnLockActorSelectionButtonClic
ked)
    );
    MenuBuilder.AddMenuEntry(
        FText::FromString(TEXT("UnLock All Actor Selection")),
        FText::FromString(TEXT("Remove Selected Locked Actor")),

FSlateIcon(FSuperMangerStyle::GetStyleSetName(),"ActorBrowser.UnLockSelectedActors
"),

FExecuteAction::CreateRaw(this,&FSuperMangerModule::OnUnLockActorSelectionButtonCl
icked)
    );
    MenuBuilder.AddMenuEntry(
        FText::FromString(TEXT("UnLock All Actor IN Level")),
        FText::FromString(TEXT("Remove All Locked Actor")),

FSlateIcon(FSuperMangerStyle::GetStyleSetName(),"ActorBrowser.UnLockAllActors"),

FExecuteAction::CreateRaw(this,&FSuperMangerModule::OnUnLockAllActorsInLevelButton
Clicked)
    );
}
```

lock和unlock

首先关于上锁和解锁的设定:被锁定的actor不允许在被鼠标选中

捕获点击Actor的name -- 触发函数

这里首先要注册一个函数，使用USelection获取选择actor,当选择到某一个actor的时候，触发OnActorSelected函数，这个函数会检查是否被锁定，如果锁定就接触选择状态，保证事务不可被选取

```
void FSuperMangerModule::InitCustomSelectEvent()
{
    // Returns the set of selected actors.
    USelection* UserSelection = GEditor->GetSelectedActors();
    UserSelection-
>SelectObjectEvent.AddRaw(this,&FSuperMangerModule::OnActorSelected);
}

void FSuperMangerModule::OnActorSelected(UObject* SelectedObject)
{
    if (!GetEditorActorSubsystem())
    {
```

```
        return;
    }
    if (!SelectedObject)
    {
        return;
    }
    if (AActor* SelectedActor = Cast<AActor>(SelectedObject))
    {
        if (CheckIsActorSelectedLocked(SelectedActor))
        {
            //Deselect 解除选择
            WeakEditorActorSubsystem->SetActorSelectionState(SelectedActor,false);
        }
    }
}
```

上锁和解锁的实现

要实现上锁和解锁 首先要实现一个点击事件，也就是当点击level中的actor的时候，可以将点击到的actor的信息回传 这里需要使用前面actor开发中使用的UEditorActorSubsystem中的GetSelectedLevelActors方法，为了管理方便这个使用WeakObjectPtr管理 每次调用之前需要先验证

```
bool FSuperMangerModule::GetEditorActorSubsystem()
{
    if (!WeakEditorActorSubsystem.IsValid())
    {
        WeakEditorActorSubsystem = GEditor-
>GetEditorSubsystem<UEditorActorSubsystem>();
    }
    return WeakEditorActorSubsystem.IsValid();
}
```

WeakObjectPtr 管理 UEditorActorSubsystem

为什么使用WeakObjectPtr

示例代码如下所示，这里引用一个AActor*指针并且使用UPROPERTY将其纳入到UE的GC机制，这种写法相当对该AActor就添加了一次引用，意味着该Actor将不会被GC。当然如果使用TWeakObjectPtr来包装对应AActor，这样可以使用该AActor，但是并不会阻止其GC。

```
UPROPERTY()
AActor* A;

TWeakObjectPtr<AActor> A;
A = GetXXXActor();
```

当然既然TWeakObjectPtr不会影响对应UObject的GC流程，那么使用时也需要注意其有效性。使用前当然需要检验当前UObjecct的可用性，TWeakObjectPtr提供了IsValid方法用于判断引用的UObject是否可用，Get方法则会返回一个UObject指针

```
AMyActor* Actor
TWeakObjectPtr<AMyActor> ActorReference = Actor;
.......

if (ActorReference.IsValid())
{
    // 使用Get()方法获取对象指针
    AMyActor* ValidActor = ActorReference.Get();
    // 在有效的Actor上执行操作
    ValidActor->SomeMethod();
}
else
{
    // TWeakObjectPtr无效，可能是因为对象已被销毁
    UE_LOG(LogTemp, Warning, TEXT("Actor reference is invalid!"));
}
```

总之： TWeakObjectPtr是一个平时在开发过程中出现频率很高的工具，通常对于一些不确定其生命周期的UObject对象都可以通过TWeakObjectPtr包装一层以避免循环引用问题，并且可以避免对其UObject生命周期有所影响，确保引用的对象在其生命周期内被正确管理。当然在搞清楚TWeakObjectPtr的实现中也学到了很多其他知识，比如UObject在Unreal中是怎么被管理，当然还有一些模板元编程的技巧。

参考： https://blog.uwa4d.com/archives/USparkle_TWeakObjectPtr.html

如何区分lock和unlock的actor 如何上锁、解锁

首先关于上锁和解锁的设定:被锁定的actor不允许在被鼠标选中

区分上锁和解锁的状态，这里使用AActor类中的tag数组

```
void FSuperMangerModule::LockActorSelection(AActor* ActorToProcess)
{
    if (!ActorToProcess){return;}
    if (!ActorToProcess->ActorHasTag(TEXT("Locked")))
    {
        ActorToProcess->Tags.Add(FName("Locked"));
    }
}
void FSuperMangerModule::UnLockActorSelection(AActor* ActorToProcess)
{
    if (!ActorToProcess){return;}
    if (ActorToProcess->ActorHasTag(TEXT("Locked")))
    {
        ActorToProcess->Tags.Remove(TEXT("Locked"));
    }
}
```

```
bool FSuperMangerModule::CheckIsActorSelectedLocked(AActor* ActorToProcess)
{
    if (!ActorToProcess){return false;}
    return ActorToProcess->ActorHasTag(TEXT("Locked"));
}
```

在实现完成上锁、解锁的操作后就是将其调用到触发函数中

```
void FSuperMangerModule::OnLockActorSelectionButtonClicked()
{
    if (!GetEditorActorSubsystem())
    {
        return;
    }
    // 通过EditorActorSubsystem获取选择的Actors
    TArray<AActor*> SelectedActors = WeakEditorActorSubsystem-
>GetSelectedLevelActors();
    if (SelectedActors.Num() == 0)
    {
        DebugHeader::ShowNotifyInfo(TEXT("No Actor Selection"));
        return;
    }
    FString CurrentLockActorName = TEXT("Locked Actor for :\n");
    for (AActor* SelectedActor : SelectedActors)
    {
        if (!SelectedActor)
        {
            continue;
        }
        LockActorSelection(SelectedActor);
        WeakEditorActorSubsystem->SetActorSelectionState(SelectedActor,false);
        CurrentLockActorName.Append(SelectedActor->GetActorLabel() + "\n");
    }
    DebugHeader::ShowNotifyInfo(CurrentLockActorName);
}

void FSuperMangerModule::OnUnLockActorSelectionButtonClicked()
{
    if (!GetEditorActorSubsystem())
    {
        return;
    }
    // 通过EditorActorSubsystem获取选择的Actors
    TArray<AActor*> SelectedActors = WeakEditorActorSubsystem-
>GetSelectedLevelActors();
    if (SelectedActors.Num() == 0)
    {
        DebugHeader::ShowNotifyInfo(TEXT("No Actor Selection"));
        return;
    }
    FString CurrentUnLockActorName = TEXT("Locked Actor for :\n");
    for (AActor* SelectedActor : SelectedActors)
```

```
    {
        if (!SelectedActor){
            continue;
        }
        if (CheckIsActorSelectedLocked(SelectedActor))
        {
            UnLockActorSelection(SelectedActor);
            CurrentUnLockActorName.Append(SelectedActor->GetActorLabel() + "\n");
        }
    }
    DebugHeader::ShowNotifyInfo(CurrentUnLockActorName);
}

void FSuperMangerModule::OnUnLockAllActorsInLevelButtonClicked()
{
    if (!GetEditorActorSubsystem())
    {
        return;
    }
    // 这里用tEditorActorSubsystem遍历全部的actor
    TArray<AActor*> AllLevelActors = WeakEditorActorSubsystem-
>GetAllLevelActors();
    if (AllLevelActors.Num() == 0)
    {
        DebugHeader::ShowNotifyInfo(TEXT("No Actor In Level"));
        return;
    }
    uint32 UnLockNum = 0;
    for (AActor* LevelActor : AllLevelActors)
    {
        if (!LevelActor)
        {
            continue;
        }
        if (CheckIsActorSelectedLocked(LevelActor))
        {
            UnLockActorSelection(LevelActor);
            UnLockNum+=1;
        }
    }
    DebugHeader::ShowNotifyInfo(TEXT("UnLock " + FString::FromInt(UnLockNum)+  "
Actors in Level"));
}
```

关于log的设置 就不再赘述了

HotKeys 设置

一个hot的设置无外乎就是:

1. 设置一个继承自TCommand< T>的class 并重写RegisterCommands方法
2. 设置一个UI_Command 并用一个 TSharedPtr< FUICommandInfo>作为commandID向外调用

3. 在调用的插件中StartupModule时注册，ShutdownModule时注销
4. 使用TSharedPtr< FUICommandList>添加绑定commandID，并将其绑定对应的操作函数
5. 在第一次页面委托注册的时候就将TSharedPtr< FUICommandList>添加到command管理中

TCommands继承的设置

构造必须进行的传参 virtual void RegisterCommands() = 0; 纯函数

HotKey设置 -- levelEditor

levelEditor中设置快捷键实现之前面的上锁等操作 首先对TCommand类

```
class FSuperMangerUICommand : public TCommands<FSuperMangerUICommand>{
public:
    // const FName InContextName(ContexName), const FText& InContextDesc(Contex描
述), const FName InContextParent, const FName InStyleSetName
    FSuperMangerUICommand() : TCommands<FSuperMangerUICommand>(
         TEXT("SuperManger"),
         FText::FromString((TEXT("Super Manger UI Commands"))),
         NAME_None,
         TEXT("SuperManger")
    ){}

    virtual void RegisterCommands() override;


    // 向外暴露的实现UICommand热键的接口
    TSharedPtr<FUICommandInfo> LockActorSelection;
    TSharedPtr<FUICommandInfo> UnLockActorSelection;
    TSharedPtr<FUICommandInfo> UnLockAllActor;
};
// UI_Command需要一个LOCTEXT_NAMESPACE 这里直接使用FSuperMangerModule
#define LOCTEXT_NAMESPACE "FSuperMangerModule"
void FSuperMangerUICommand::RegisterCommands()
{
    // 注册一个UI_COMMAND 快捷键 w+alt type为按下
    // 第一个参数 CommandID--TSharedPtr<FUICommandInfo>
    // 第二个参数和第三个参数是name和描述  第四个参数是触发类型 第五个参数为触发按键
    UI_COMMAND(
        LockActorSelection,
        "Lock Actor Selection",
        "Lock actor selection in level, once triggered, actor can no longer be
selected",
        EUserInterfaceActionType::Button,
        FInputChord(EKeys::W,EModifierKey::Alt)
    );

    UI_COMMAND(
        UnLockActorSelection,
        "UnLock Actor Selected",
        "UnLock actor selection in level",
        EUserInterfaceActionType::Button,
```

```
        FInputChord(EKeys::E,EModifierKey::Alt)
    );

    UI_COMMAND(
        UnLockAllActor,
        "UnLock ALL Actor ",
        "UnLock All actor selection in level",
        EUserInterfaceActionType::Button,
        FInputChord(EKeys::E,EModifierKey::Alt | EModifierKey::Shift)
    );
}
#undef LOCTEXT_NAMESPACE
```

在插件管控类中使用MapAction的方式绑定触发函数

```
void FSuperMangerModule::InitCustomUICommands()
{
    CustomUICommands = MakeShareable(new FUICommandList());
    CustomUICommands->MapAction(
        FSuperMangerUICommand::Get().LockActorSelection,

FExecuteAction::CreateRaw(this,&FSuperMangerModule::OnSelectionLockHotKeyPressed))
;

    CustomUICommands->MapAction(
        FSuperMangerUICommand::Get().UnLockActorSelection,

FExecuteAction::CreateRaw(this,&FSuperMangerModule::OnSelectionUnLockHotKeyPressed
));

    CustomUICommands->MapAction(
        FSuperMangerUICommand::Get().UnLockAllActor,

FExecuteAction::CreateRaw(this,&FSuperMangerModule::OnUnLockAllActorsInLevelButton
Clicked));
}
//触发函数实际上就是调用之前实现的接口
void FSuperMangerModule::OnSelectionLockHotKeyPressed()
{
    OnLockActorSelectionButtonClicked();
}

void FSuperMangerModule::OnSelectionUnLockHotKeyPressed()
{
    OnUnLockActorSelectionButtonClicked();
}

void FSuperMangerModule::OnSelectionUnLockAllHotKeyPressed()
{
    OnUnLockAllActorsInLevelButtonClicked();
}
```

在前面绑定的时候将CustomUICommands传入LevelEditorModule的GetGlobalLevelEditorActions中

```
void FSuperMangerModule::InitLevelEditorMenuExtension()
{
    // 获取LevelEditorManger
    FLevelEditorModule& LevelEditorModule =
        FModuleManager::LoadModuleChecked<FLevelEditorModule>
(TEXT("LevelEditor"));

    TSharedRef<FUICommandList> ExitingUICommands =
LevelEditorModule.GetGlobalLevelEditorActions();
    ExitingUICommands->Append(CustomUICommands.ToSharedRef());
```

Hotkey设置 -- ContentBrowser

回到InitCBMenuExtention中，在FContentBrowserModule中并没有有直接返回TSharedRef< FUICommandList>的方法。只有GetAllContentBrowserCommandExtenders返回一个委托管理的command的数组，而这个委托传入的参数还有一个委托用来处理传入的asset的数据。

```
/** Called when registering a custom command/keybinding for the content browser */
DECLARE_DELEGATE_TwoParams(FOnContentBrowserGetSelection, TArray<FAssetData>&
/*SelectedAssets*/, TArray<FString>& /*SelectedPaths*/);
DECLARE_DELEGATE_TwoParams(FContentBrowserCommandExtender,
TSharedRef<FUICommandList> /*CommandList*/, FOnContentBrowserGetSelection
/*GetSelectionDelegate*/);
```

所以这里就得这么写 首先关于TCommand

```
class FTestCommandsLineList : public TCommands<FTestCommandsLineList>
{
public:
    FTestCommandsLineList();

    // TCommands<>的接口：注册命令
    virtual void RegisterCommands() override;

public:
    //命令A
    TSharedPtr<FUICommandInfo> CommandA;
};
FTestCommandsLineList::FTestCommandsLineList() : TCommands<FTestCommandsLineList>(
    "FTestCommandsLineList",
    NSLOCTEXT("Contexts", "TestCommandsLineList", "SuperManger Plugin"),
    NAME_None,
    FName(*FString("todo")))
{
```

```
}

void FTestCommandsLineList::RegisterCommands()
{
    /*
     * 命令的内部名称
     * 在用户界面（如工具栏或菜单）中显示的名称。
     * 命令的描述信息，用于工具提示（Tooltip）等。
     * 指定命令的操作类型
     * FInputChord 快捷键
     */
    UI_COMMAND(CommandA,
        "FTestCommandsLineList",
        "Execute TestYaksue CommandA",
        EUserInterfaceActionType::Button,
        FInputChord(EModifierKey::Shift | EModifierKey::Alt, EKeys::X));
}
```

在注册的时候 -- 这里使用绑定lamda表达式的方式减少函数数量

```
void FSuperMangerModule::InitCBMenuExtention()
{
    //加载BrowserModule目录
    FContentBrowserModule& ContentBrowserModule =
        FModuleManager::LoadModuleChecked<FContentBrowserModule>
(TEXT("ContentBrowser"));
    TArray<FContentBrowserCommandExtender>& ContentBrowserCommandList =
        ContentBrowserModule.GetAllContentBrowserCommandExtenders();

    // 加载Command命令
    ContentBrowserCommandList.Add(FContentBrowserCommandExtender::CreateLambda(
        [this](TSharedRef<FUICommandList> CommandList,
                        FOnContentBrowserGetSelection GetSelectionDelegate)
        {
            //为命令映射操作
            GetSelectionDelegate = FOnContentBrowserGetSelection::CreateLambda(
                [this]( TArray<FAssetData>& SelectedAssets, TArray<FString>&
SelectedPaths)
                {

DebugHeader::PrintLog("FOnContentBrowserGetSelection::Fun()\n");
                    if (SelectedAssets.Num() > 0 && SelectedPaths.Num() > 0 &&
SelectedAssets.Num() == SelectedPaths.Num())
                    {
                        for (int32 i=0;i<SelectedAssets.Num() ;++i)
                        {
                            DebugHeader::PrintLog("SelectedAssets Name is " +
SelectedAssets[i].AssetName.ToString()
                                + "SelectedAssets Name is " + SelectedPaths[i] +
"\n");
                        }
```

```
                    }
            });
            CommandList->MapAction(
                FTestCommandsLineList::Get().CommandA,
                FExecuteAction::CreateLambda([this,GetSelectionDelegate]()
                {
                    CommandLineAAction(GetSelectionDelegate);
                })
            );
        }
    ));
    // 获取内容浏览器上下文菜单拓展器，在这个上面可以添加委托用来自定义菜单
    // 这里实际上是一个委托数组 类型和CustomCBMenuExtender一致
    TArray<FContentBrowserMenuExtender_SelectedPaths>&
ContentBrowserMoudleMenuExtenders =
        ContentBrowserModule.GetAllPathViewContextMenuExtenders();
    FContentBrowserMenuExtender_SelectedPaths CustomCBMenuDelegate;
    // 绑定委托函数
    CustomCBMenuDelegate.BindRaw(this,&FSuperMangerModule::CustomCBMenuExtender);
    ContentBrowserMoudleMenuExtenders.Add(CustomCBMenuDelegate);
}
```

而实际上的触发函数并不是那么重要，因为没有实现什么功能，主要是上面两个委托的绑定

```
void FSuperMangerModule::CommandLineAAction(FOnContentBrowserGetSelection
InContentBrowserGetSelection)
{
    DebugHeader::PrintLog("FSuperMangerModule::CommandLineAAction()\n");
    DebugHeader::ShowNotifyInfo("FSuperMangerModule::CommandLineAAction()\n");
    if (InContentBrowserGetSelection.IsBound())
    {
        TArray<FAssetData> SelectedAssets =
UEditorUtilityLibrary::GetSelectedAssetData();
        TArray<FString> SelectedPath;
        if (SelectedAssets.Num() > 0)
        {
            for (int32 i=0;i<SelectedAssets.Num();++i)
            {
                SelectedPath.Add(SelectedAssets[i].PackagePath.ToString());
            }
        }
        InContentBrowserGetSelection.Execute(SelectedAssets,SelectedPath);
    }
    else
    {
        DebugHeader::Print(TEXT("delegate not bind"),FColor::Red);
    }
}
```

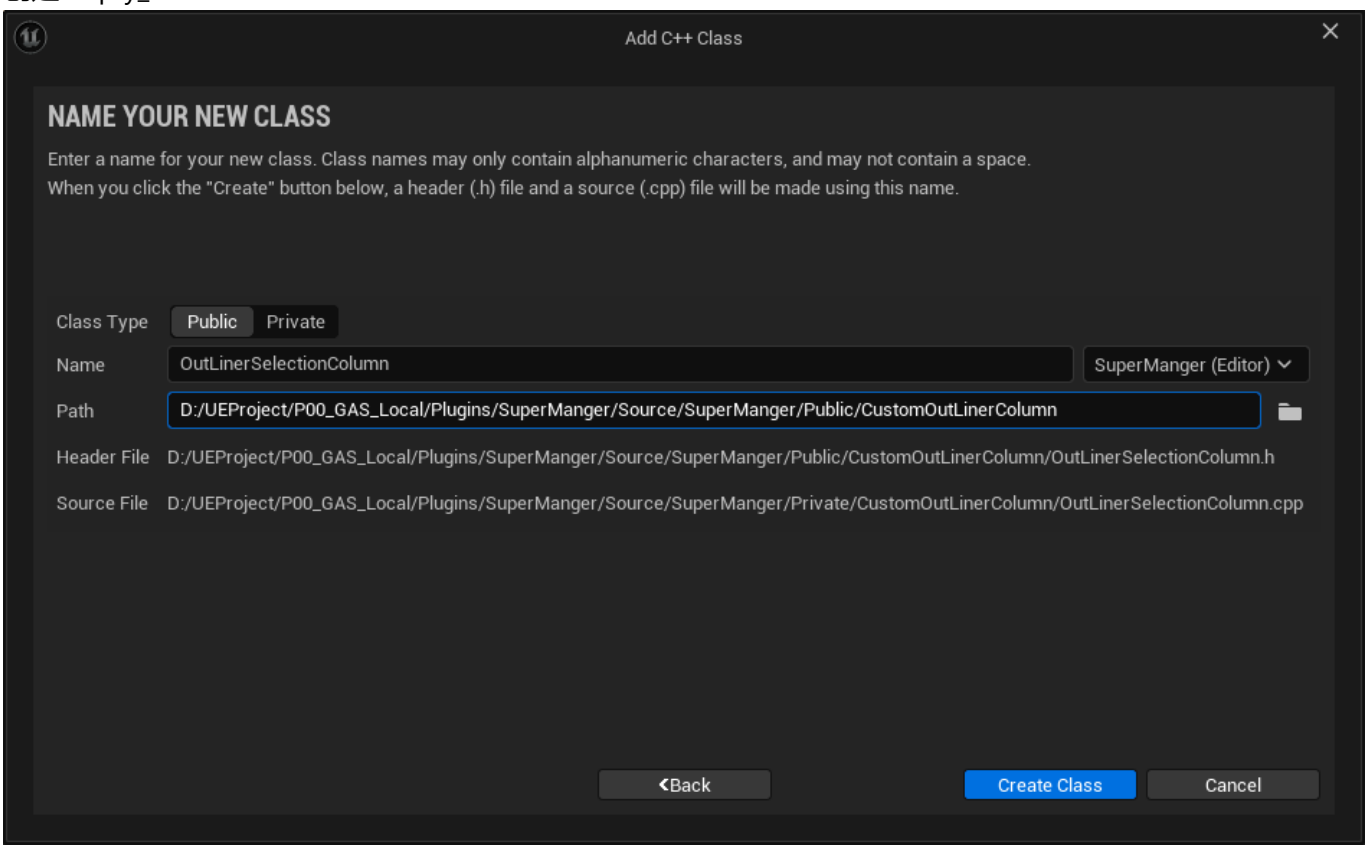
最关键的 StartupModule

前面实现了很多功能 都需要在StartupModule中注册才能使用

```
void FSuperMangerModule::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Style -- Icon
    FSuperMangerStyle::InitializeIcons();
    // dyj 注册命令:
    FTestCommandsLineList::Register();
    // 注册 ContentBrowser
    InitCBMenuExtention();
    // 注册绑定事件
    InitCustomSelectEvent();
    // 注册HotKey
    FSuperMangerUICommand::Register();
    // 注册Hotkey绑定函数
    InitCustomUICommands();
    // 由于在这里使用hotkey中CustomUICommands 所以先注册hotkey再执行注册levelBrowser
    InitLevelEditorMenuExtension();
    // 注册slot窗口
    RegisterAdvanceDeletionTab();
}
```

对lableOntliner的自定义设置（右侧显示level中actor的列表）

创建empty_class



首先需要一个ISceneOutLinerInterface 这要在build中添加一个"SceneOutliner" 才能引入头文件#include "ISceneOutlinerColumn.h"

重写ISceneoutliner 的虚函数

这里创建头部(top顶部)的slot

```
SHeaderRow::FColumn::FArguments
FOutLinerSelectionLockColumn::ConstructHeaderRowColumn()
{
    SHeaderRow::FColumn::FArguments ConstructHeadRow =
SHeaderRow::Column(GetColumnID())
    .FixedWidth(24.f)
    .HAlignHeader(HAlign_Center)
    .VAlignHeader(VAlign_Center)
    .HAlignCell(HAlign_Center)
    .VAlignCell(VAlign_Center)
    .DefaultTooltip(FText::FromString("Actor Selection Lock - Press Icon to lock
Actor Selection"))
    [
        SNew(SImage)
        .ColorAndOpacity(FSlateColor::UseForeground())
        .Image(FSuperMangerStyle::GetCreatedSlateStyleSet()-
>GetBrush("ActorBrowser.LockActors")) // 第二个参数是FSuperMangerStyle绑定lock事件的
icon
    ];
    return ConstructHeadRow;
}
```

在supermanger中通过

```
void FSuperMangerModule::InitSceneOutLinerExtension()
{
    //创建module
    FSceneOutlinerModule& SceneOutlinerModule =
        FModuleManager::LoadModuleChecked<FSceneOutlinerModule>("SceneOutliner");

    //注册事件
    // 第一个参数是否可见 第二个参数权重 默认情况下icon是0 这里紧跟在icon后面设置为1
    FSceneOutlinerColumnInfo SelectionColumnInfo(
        ESceneOutlinerColumnVisibility::Visible,
        1,

FCreateSceneOutlinerColumn::CreateRaw(this,&FSuperMangerModule::OnCreateSelectionL
ockColumns));
    SceneOutlinerModule.RegisterDefaultColumnType<FOutLinerSelectionLockColumn>
(SelectionColumnInfo);
}

TSharedRef<ISceneOutlinerColumn>
```

```
FSuperMangerModule::OnCreateSelectionLockColumns(ISceneOutliner& SceneOutliner)
{
    return MakeShared<FOutLinerSelectionLockColumn>(SceneOutliner);
}
```

完成调用

TreeItem

在outline中是树状结构的显示。所以在构建每一行信息中，ISceneOutLiner提供的方法的参数是FSceneOutlinerTreeItemRef TreeItem, 、const STableRow< FSceneOutlinerTreeItemPtr>& Row这两个参数

首先就是要将TreeItem转换为actorItem

```
const TSharedRef<SWidget>
FOutLinerSelectionLockColumn::ConstructRowWidget(FSceneOutlinerTreeItemRef
TreeItem,
    const STableRow<FSceneOutlinerTreeItemPtr>& Row)
{
    // 获取World中的TreeItem
    FActorTreeItem* ActorTreeItem = TreeItem->CastTo<FActorTreeItem>();
    if (!ActorTreeItem || !ActorTreeItem->IsValid())
    {
        return SNullWidget::NullWidget;
    }

    // 由于CheckBox的状态和当前lable是否被lock相关 所以这里获取一下SuperManger的方法
    FSuperMangerModule& SuperMangerModule =
            FModuleManager::LoadModuleChecked<FSuperMangerModule>("SuperManger");
    // 检查当前是否被锁定
    const bool bIsActorSelectionLocked =
        SuperMangerModule.OutLinerCheckActorSelectionLocked(ActorTreeItem-
>Actor.Get());
    const FCheckBoxStyle& ToggleButtonStyle =
        FSuperMangerStyle::GetCreatedSlateStyleSet()-
>GetWidgetStyle<FCheckBoxStyle>(FName("SceneOutliner.SelectionLock"));

    TSharedRef<SWidget> RowWidgetCheckBox = SNew(SCheckBox)
    .HAlign(HAlign_Center)
    .Type(ESlateCheckBoxType::ToggleButton)   // 将其切换为ToggleButton
    .Style(&ToggleButtonStyle)
    .Visibility(EVisibility::Visible)
    .IsChecked(bIsActorSelectionLocked ? ECheckBoxState::Checked :
ECheckBoxState::Unchecked)
    .OnCheckStateChanged(this,
&FOutLinerSelectionLockColumn::OnRowCheckStateChanged,ActorTreeItem->Actor);
    return RowWidgetCheckBox;
}
void FOutLinerSelectionLockColumn::OnRowCheckStateChanged(ECheckBoxState
NewCheckState,
    TWeakObjectPtr<AActor> CorrespondingActor)
```

```
{
    FSuperMangerModule& SuperMangerModule =
                    FModuleManager::LoadModuleChecked<FSuperMangerModule>
("SuperManger");
    switch (NewCheckState)
    {
        case ECheckBoxState::Unchecked:
            SuperMangerModule.ProcessLocingForOutliner(CorrespondingActor.Get(),
false);
            break;
        case ECheckBoxState::Checked:
            SuperMangerModule.ProcessLocingForOutliner(CorrespondingActor.Get(),
true);
            break;
        case ECheckBoxState::Undetermined:
            break;
        default:
            break;
    }
}
```

当修改之后的内容更新

这里实际上就是向SuperMangerModule调用接口获取是否被锁定和状态修改的结果触发函数 实际上是根据Check状态更新每一个CheckBox

```
void FSuperMangerModule::RefreshSceneOutliner()
{
    FLevelEditorModule& LevelEditorModule =
        FModuleManager::LoadModuleChecked<FLevelEditorModule>
(TEXT("LevelEditor"));
    TSharedPtr<ISceneOutliner> SceneOutliner =
        LevelEditorModule.GetFirstLevelEditor()->GetSceneOutliner();
    if (SceneOutliner.IsValid())
    {
        SceneOutliner->FullRefresh();
    }
}
```

一些Debug

1. 在deletionTab 需要设置当传入folder数组为空的时候的设置
2. 在打开deletionTab中 还可以使用前面的功能如删除不用的数组等

修改策略将DeletionTab设置为类内变量

```
TSharedRef<SDockTab> FSuperMangerModule::OnSpawnAdvanceDeletionTab(const
FSpawnTabArgs& args)
{
```

```
    //create new Slate Weight
    /*
     *这里[]之间就是一个slot(槽)
     *可以立即为一个占据窗口的小部件
     */
    if (FolderPathSelected.Num() == 0)
    {
        return SNew(SDockTab).TabRole(NomadTab);
    }
    ConstructedDockTab = SNew(SDockTab).TabRole(ETabRole::NomadTab)
    [
        SNew(SAdvanceDeletionTab)
        .AssetsDataArray(GetAllAssetDataUnderSelectedFolder())
        .CurrentSelectFodler(FolderPathSelected[0])
    ];
    // 设置关闭是触发的函数 -- 将ConstructedDockTab设置为空
    ConstructedDockTab->SetOnTabClosed(

SDockTab::FOnTabClosedCallback::CreateRaw(this,&FSuperMangerModule::OnAdvanceDelet
ionTabClosed));
    return ConstructedDockTab.ToSharedRef();
}
```

而其他功能使用通过前需要检查ConstructedDockTab是否为空，如果vaild，就直接return

```
void FSuperMangerModule::OnDeleteUnusedButtonClicked()
{
    // 如果ConstructedDockTab(DeletionTab)存在(打开) 就不能执行这个操作
    if (ConstructedDockTab.IsValid())
    {
        DebugHeader::ShowMessageDialog(EAppMsgType::Type::Ok,TEXT("Wait! Close
advance deletion tab"));
        return;
    }
void FSuperMangerModule::OnDeleteEmptyFolderButtonClicked()
{
    // 如果ConstructedDockTab(DeletionTab)存在(打开) 就不能执行这个操作
    if (ConstructedDockTab.IsValid())
    {
        DebugHeader::ShowMessageDialog(EAppMsgType::Type::Ok,TEXT("Wait! Close
advance deletion tab"));
        return;
    }
```

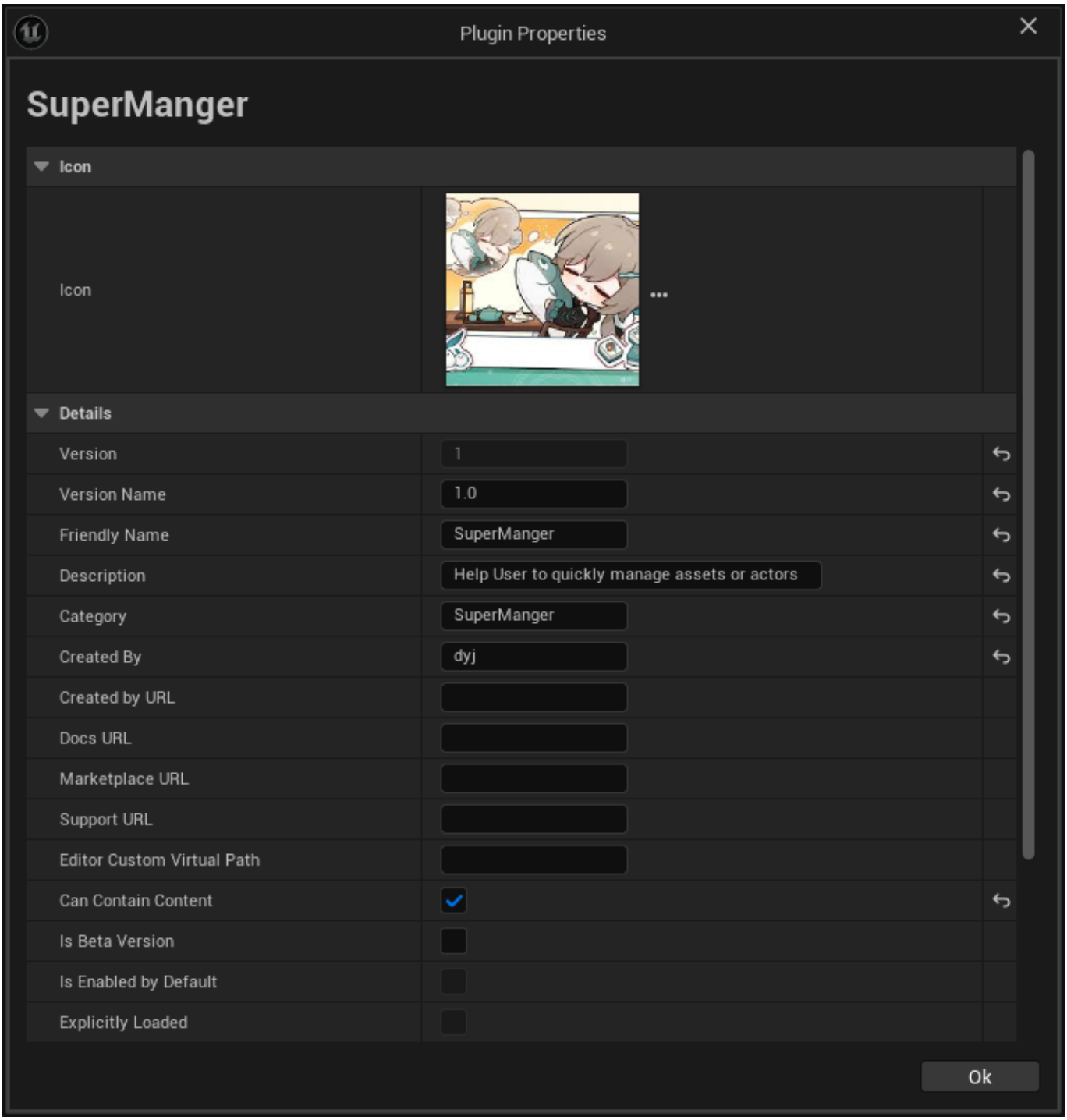
并且设置当Tab关闭的时候设置触发函数SetOnTabClosed-OnAdvanceDeletionTabClosed 在这个函数中将ConstructedDockTab复位

```
void FSuperMangerModule::OnAdvanceDeletionTabClosed(TSharedRef<SDockTab>
TabToClose)
```

```
{
    if (ConstructedDockTab.IsValid())
    {
        ConstructedDockTab.Reset();
        FolderPathSelected.Empty();
    }
}
```

打包插件

edit - plugins - other 可以找到我们自己的plugins



点击package就可以选择一个文件夹进行打包

当需要在别的项目中使用这个插件的时候 删除选中的两个文件内容 -- 保证其可以重新被编译

打包的指令参考 这里拷贝了执行的log 使用的时候重新进行RunUAT打包

```
LogMonitoredProcess: Running Serialized UAT: [ cmd.exe /c
""D:/UE/UE_5.4/Engine/Build/BatchFiles/RunUAT.bat" BuildPlugin -
Plugin="D:/UEProject/P00_GAS_Local/Plugins/SuperManger/SuperManger.uplugin" -
Package="D:/UEProject/P00_GAS_Local/Plugins/SuperManger/Package/SuperManger" -
CreateSubFolder" -nocompile -nocompileuat ]
UATHelper: Package Plugin Task (Windows): Running AutomationTool...
```

当然copy新的plugins之后 可以再将被拷贝新项目中的上图中选中的文件删除，之后重新编译即可

参考链接 -- ！ 顺序存在优先级！

课程 https://www.bilibili.com/video/BV1M84y1K7m4/?p=3&spm_id_from=333.1007.top_right_bar_window_history.content.click&vd_source=eeea416256f7a4069ece466b8b539e84

知乎总结 https://zhuanlan.zhihu.com/p/605181368

课程路人总结 https://blog.csdn.net/weixin_42150569/article/details/129556296

课程项目 https://github.com/vinceright3/ExtendingEditorCourseSourceCode

CommandLine任务举例： https://blog.csdn.net/u013412391/article/details/107891152

weakobjectptr https://blog.uwa4d.com/archives/USparkle_TWeakObjectPtr.html

菜单栏拓展 https://blog.csdn.net/j756915370/article/details/121674509

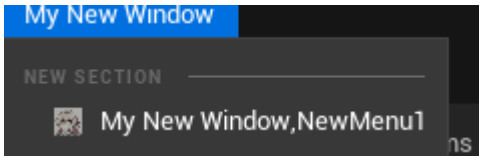
编辑器内容扩展 https://mytechplayer.com/archives/ue45%E4%B8%AD%E7%BC%96%E8%BE%91%E5%99%A8%E5%92%8C%E5%91%BD%E4%BB%A4%E7%9A%84%E6%89%A9%E5%B1%95%E6%96%B0

edit 第二期 -- FToolMenuSection

打开编辑模式

参考知乎 文档 需要创建一个default配置 否则UE5.4无法找到这个

之后在窗口就可以找到了

使用FToolMenuSection管理 

使用FToolMenuSection创建一个新的菜单窗口

```
void FMyEditorToolBarButtonModule::ExtendNewWindow()
{
    UToolMenu* Menu = UToolMenus::Get()->ExtendMenu("LevelEditor.MainMenu");
    // 传入Name_None表示一定进行添加
    FToolMenuSection& Section = Menu->FindOrAddSection(NAME_None);
    FToolMenuEntry& MakeEntry = Section.AddSubMenu(
        "NewMenuInMainWindow",     //name
        LOCTEXT("NewMenuLable","My New Window"),    // lable
        LOCTEXT("NewMenuToolTip","My New Window top list"),
        FNewToolMenuChoice()
    );
    MakeEntry.InsertPosition = FToolMenuInsert("Help",EToolMenuInsertType::After);

    // 注册为新菜单  注意这个name必须和你前面创建的name相同
    static const FName BaseMenuName = "LevelEditor.MainMenu.NewMenuInMainWindow";
    Menu = UToolMenus::Get()->RegisterMenu(BaseMenuName);
    // 在新菜单下添加Section和Entry
    FToolMenuSection& NewSection = Menu->AddSection("New Section",
FText::FromString("New Section"));
    NewSection.AddMenuEntry(
        FName(TEXT("NewMenu1")),
        FText::FromString("My New Window,NewMenu1"),
        FText::FromString("My New Window,NewMenu1,Top Message"),

FSlateIcon(FMyEditorToolBarButtonStyle::GetStyleSetName(),"MyEditorToolBarButton.I
conQQ"),
        FToolUIActionChoice(FExecuteAction::CreateRaw(this,

&FMyEditorToolBarButtonModule::PluginMenuFolderButtonClick))
    );
}
```

```
void FMyEditorToolBarButtonModule::RegisterMenus()
{
    // Owner will be used for cleanup in call to UToolMenus::UnregisterOwner
    // 确保在菜单修改时，当前对象（this）被标记为菜单的拥有者。
    FToolMenuOwnerScoped OwnerScoped(this);
    // 拓展主菜单
    {
```

```
        // 创建一个新的主菜单
        UToolMenu* Menu = UToolMenus::Get()->ExtendMenu("LevelEditor.MainMenu");
        FToolMenuSection& Section = Menu->FindOrAddSection("MyExtendTools");
        FToolMenuEntry& MakeEntry = Section.AddSubMenu(
            "MyToolsBar",     //name
            FText::FromString(FString("MyToolWindow")),    // lable
            FText::FromString(FString("MyToolWindow Top List")),
            FNewToolMenuChoice()
        );
        MakeEntry.InsertPosition =
FToolMenuInsert("Help",EToolMenuInsertType::After);
        static const FName BaseMenuName = "LevelEditor.MainMenu.MyToolsBar";
        Menu = UToolMenus::Get()->RegisterMenu(BaseMenuName);


        // 在新菜单下添加Section和Entry
        FToolMenuSection& PartMenuSection1 = Menu-
>AddSection("SectionPart1",FText::FromString("SectionPart1"));
        PartMenuSection1.AddMenuEntry(
            "Selection1",     //name
            FText::FromString(FString("MyToolWindow.Selection1")),    // lable
            FText::FromString(FString("MyToolWindow.Selection1 Top List")),
            FSlateIcon(),
            FToolUIActionChoice()
        );

        // 给新创建的New Section添加一个子类bar
        PartMenuSection1.AddSubMenu(
            "NewSubSelection",
            FText::FromString("Label: SectionBar"),
            FText::FromString("This is a SectionBar by MyToolsBar"),
             FNewToolMenuChoice(FNewToolMenuDelegate::CreateLambda([](UToolMenu*
SubMenu)
                {
                    FToolMenuSection& SubMenuSection = SubMenu-
>AddSection("SubMenuSection1",
                        FText::FromString("SubMenu Section"));
                    // 添加一个普通的 MenuEntry
                    SubMenuSection.AddMenuEntry(
                        "SubMenu1",
                        FText::FromString(FString("MyToolWindow.SubMenu1")),
                        FText::FromString(FString("MyToolWindow.SubMenu1,Top
tip")),
                        FSlateIcon(),
                        FToolUIActionChoice(FExecuteAction::CreateLambda([]()
                        {

DebugHeader::Print("MyToolWindow.SubMenu1",FColor::Green);
                        }))
                    );
                    SubMenuSection.AddMenuEntry(
                        "SubMenu2",
                        FText::FromString(FString("MyToolWindow.SubMenu2")),
                        FText::FromString(FString("MyToolWindow.SubMenu2,Top
```

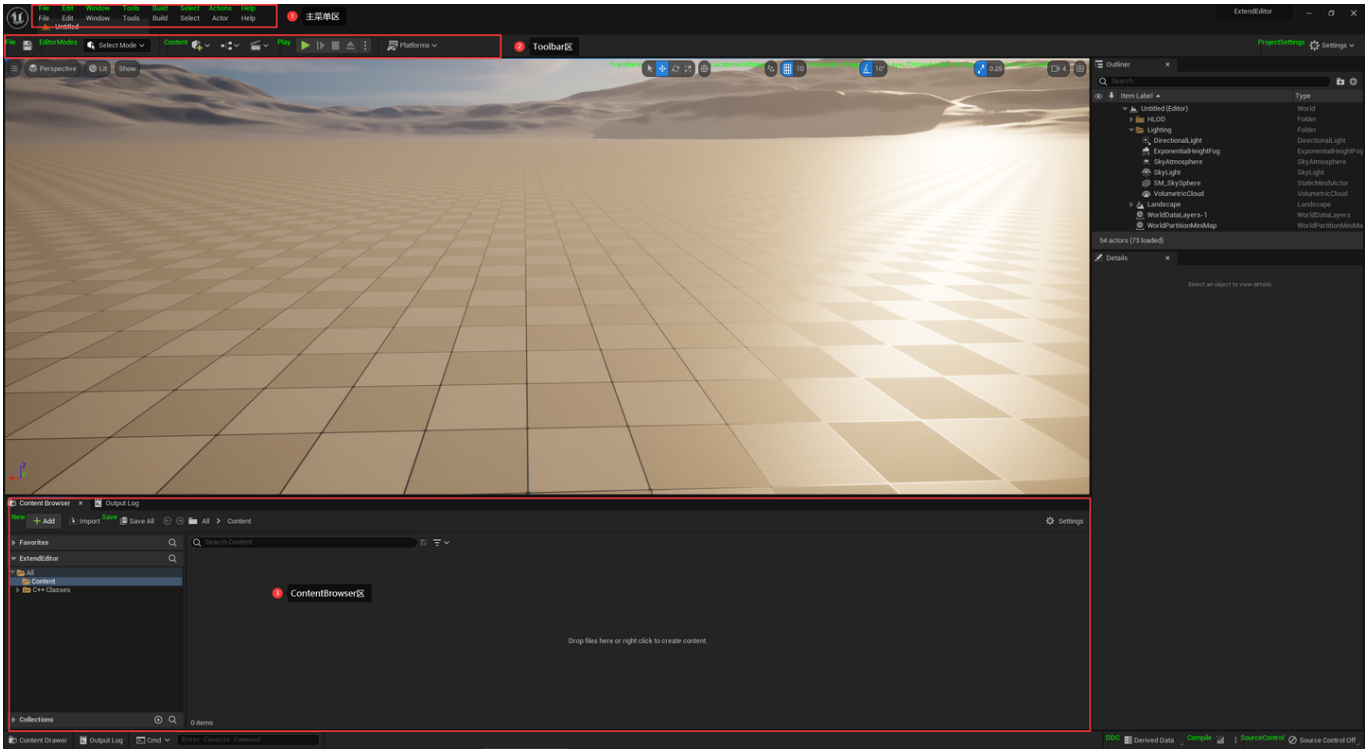
```
tip")),
                        FSlateIcon(),
                        FToolUIActionChoice(FExecuteAction::CreateLambda([]()
                        {

DebugHeader::Print("MyToolWindow.SubMenu2",FColor::Green);
                        }))
                    );
                }
            ))
        );

        // 添加一个新的LablePart 并绑定快捷键
        FToolMenuSection& PartMenuSection2 = Menu-
>AddSection("SectionPart2",FText::FromString("SectionPart2"));
        PartMenuSection2.AddMenuEntry(
            "SubSection1",
            FText::FromString("Label: SubSection"),
            FText::FromString("This is a SubSection by MyToolsBar"),
            FSlateIcon(),

FToolUIActionChoice(FToolUIActionChoice(FMyEditorToolBarButtonCommands::Get().Plug
inMenuFolder,
                                        *PluginCommands.Get()))
        );
    }
    // 拓展主菜单.Window
    {
        UToolMenu* Menu = UToolMenus::Get()-
>ExtendMenu("LevelEditor.MainMenu.Window");
        {
            // 查找或添加子类部分
            FToolMenuSection& Section = Menu->FindOrAddSection("WindowLayout");
            Section.AddMenuEntryWithCommandList(
                FMyEditorToolBarButtonCommands::Get().PluginOpenWindows,
PluginCommands);
        }
    }
    // 扩展工具栏
    {
        UToolMenu* ToolbarMenu = UToolMenus::Get()-
>ExtendMenu("LevelEditor.LevelEditorToolBar.PlayToolBar");
        {
            FToolMenuSection& Section = ToolbarMenu-
>FindOrAddSection("PluginTools");
            {
                FToolMenuEntry& Entry =
Section.AddEntry(FToolMenuEntry::InitToolBarButton(FMyEditorToolBarButtonCommands:
:Get().PluginShowMessages));
                Entry.SetCommandList(PluginCommands);
            }
        }
    }
}
```

```
void FMyEditorToolBarButtonModule::StartupModule()
    PluginCommands = MakeShareable(new FUICommandList);
    // 绑定命令
    PluginCommands->MapAction(
        FMyEditorToolBarButtonCommands::Get().PluginMenuFolder,
        FExecuteAction::CreateRaw(this,
&FMyEditorToolBarButtonModule::PluginMenuFolderButtonClick));
    PluginCommands->MapAction(
        FMyEditorToolBarButtonCommands::Get().PluginOpenWindows,
        FExecuteAction::CreateRaw(this,
&FMyEditorToolBarButtonModule::PluginOpenWindowsButtonClicked));
    PluginCommands->MapAction(
        FMyEditorToolBarButtonCommands::Get().PluginShowMessages,
        FExecuteAction::CreateRaw(this,
&FMyEditorToolBarButtonModule::PluginShowMessagesButtonClicked));

    FLevelEditorModule& LevelEditorModule =
FModuleManager::LoadModuleChecked<FLevelEditorModule>("LevelEditor");
    LevelEditorModule.GetGlobalLevelEditorActions()-
>Append(PluginCommands.ToSharedRef());
```

使用FExtender的总结

对各个地方的开发

FExtender将界面分成



FExtender的基础扩展和前面SuperManger中一致，都是经过三次绑定实现

下面工程创建的过程 -- plugins - blank 首先要完成对FExtender的注册

```
void FExtendMenuPluginModule::ExtendMenuByFExtend()
{
    const TSharedPtr<FExtender> MainMenuExtend = MakeShareable(new FExtender());

    ...

    // 获取LevelEditor的ModuleManager
    FLevelEditorModule& LevelEditorModule =

FModuleManager::Get().LoadModuleChecked<FLevelEditorModule>("LevelEditor");
    // 用于为模块（如工具栏或菜单）提供扩展性
    const TSharedPtr<FExtensibilityManager> ExtendMenuExtend =
                            LevelEditorModule.GetMenuExtensibilityManager();
    // 将 MainMenuExtend 注册到菜单扩展管理器中。
    ExtendMenuExtend->AddExtender(MainMenuExtend);
}
```

FExtender对主菜单区的扩展

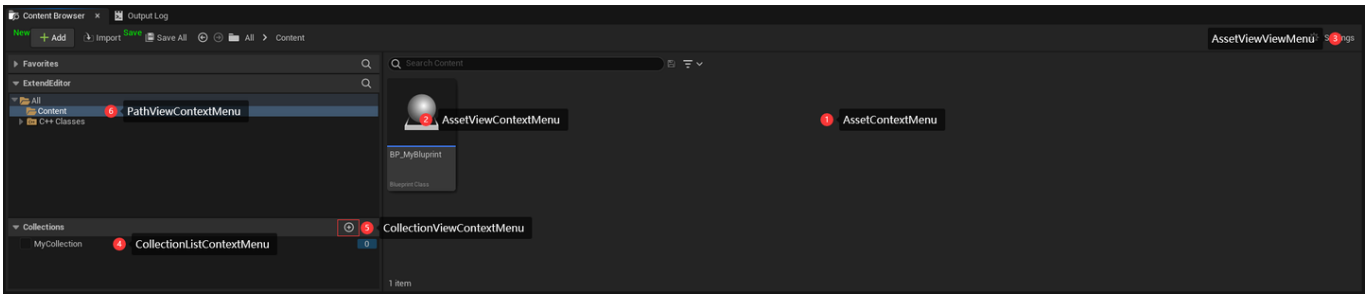
在ExtendMenuByFExtend通过MainMenuExtend进行注册，在第一次注册的时候需要绑定一个锚点

```
void FExtendMenuPluginModule::ExtendMenuByFExtend() --
    // 在这里添加创建的窗口
    MainMenuExtend->AddMenuExtension(
        "EpicGamesHelp",    // 查定位可以看到的hook
        EExtensionHook::After,    // 在hook的位置
        nullptr,

FMenuExtensionDelegate::CreateRaw(this,&FExtendMenuPluginModule::MakeExistingMainM
enuEntry)
    );
    // 主窗口扩展 添加下拉菜单
    MainMenuExtend->AddMenuBarExtension(
        "Help",
        EExtensionHook::After,
        nullptr,

FMenuBarExtensionDelegate::CreateRaw(this,&FExtendMenuPluginModule::MakeExistingMe
nuBarEntry)
    );

// 添加button的函数
void FExtendMenuPluginModule::MakeExistingMainMenuEntry(FMenuBuilder& MenuBuilder)
{
    MenuBuilder.AddMenuEntry(
        FText::FromString("Extend Menu Lable"),
        FText::FromString("Extend Menu Lable Top Tips"),
        FSlateIcon(),

FUIAction(FExecuteAction::CreateRaw(this,&FExtendMenuPluginModule::ExistingMainMen
```

```
uEntryAction))
    );
}
// 添加button具体执行的函数
void FExtendMenuPluginModule::ExistingMainMenuEntryAction()
{
    DebugHeader::Print(FString("ExistingMainMenuEntryAction"),FColor::Green);
}
```

FExtender对ContentBrowser拓展

内容浏览器分为以下几个模块



可以通过FContentBrowserModule的Getxxx方法获取 这里参考三个地方

1. 对AssetContextMenu -- 标号1

```
void FExtendMenuPluginModule::ExtendContentBrowserByFExtend()
{
    FContentBrowserModule& ContentBrowserModule =
        FModuleManager::Get().LoadModuleChecked<FContentBrowserModule>
("ContentBrowser");
    /* 对AssetContext 添加功能 */
    TArray<FContentBrowserMenuExtender_SelectedPaths>& MenuExtendersArray1 =
        ContentBrowserModule.GetAllAssetContextMenuExtenders();
    MenuExtendersArray1.Add(FContentBrowserMenuExtender_SelectedPaths::CreateRaw(
        this,&FExtendMenuPluginModule::ExtendAssetContextMenu));
```

2. 对AssetViewContext -- 标号2

```
void FExtendMenuPluginModule::ExtendContentBrowserByFExtend()
{
    FContentBrowserModule& ContentBrowserModule =
        FModuleManager::Get().LoadModuleChecked<FContentBrowserModule>
("ContentBrowser");

    /* 对AssetContext 添加功能 */
    TArray<FContentBrowserMenuExtender_SelectedAssets>& MenuExtendersArray2 =
        ContentBrowserModule.GetAllAssetViewContextMenuExtenders();
    MenuExtendersArray2.Add(FContentBrowserMenuExtender_SelectedAssets::CreateRaw(
this,
        &FExtendMenuPluginModule::ExtendAssetViewMenu));
```

注意这里都是引用 直接通过添加委托的方式实现对这个功能的拓展

3. 在supermager中我们拓展的是6 PathViewContext

```
void FSuperMangerModule::InitCBMenuExtention()
    TArray<FContentBrowserMenuExtender_SelectedPaths>&
ContentBrowserMoudleMenuExtenders =
        ContentBrowserModule.GetAllPathViewContextMenuExtenders();
    FContentBrowserMenuExtender_SelectedPaths CustomCBMenuDelegate;
    // 绑定委托函数
    CustomCBMenuDelegate.BindRaw(this,&FSuperMangerModule::CustomCBMenuExtender);
    ContentBrowserMoudleMenuExtenders.Add(CustomCBMenuDelegate);
```

GetAllLevelViewportContextMenuExtenders：Level 场景中选中 Actor 的右键菜单。
GetAllLevelViewportOptionsMenuExtenders：视口选项菜单，点击视口右上角小齿轮时弹出的菜单。
GetAllLevelViewportShowMenuExtenders：视口视图菜单。点击视口右上角小眼睛时弹出的菜单。
GetAllLevelViewportDragDropContextMenuExtenders：按住右键并拖拽 Object 到视口中松开时弹出的菜单。

对Outline的拓展

这里参考SuperManger中的过程

```
void FSuperMangerModule::InitSceneOutLinerExtension()
{
    //创建module
    FSceneOutlinerModule& SceneOutlinerModule =
        FModuleManager::LoadModuleChecked<FSceneOutlinerModule>("SceneOutliner");

    //注册事件
    // 第一个参数是否可见 第二个参数权重 默认情况下icon是0 这里紧跟在icon后面设置为1
    FSceneOutlinerColumnInfo SelectionColumnInfo(
        ESceneOutlinerColumnVisibility::Visible,
        1,

FCreateSceneOutlinerColumn::CreateRaw(this,&FSuperMangerModule::OnCreateSelectionL
ockColumns));
    SceneOutlinerModule.RegisterDefaultColumnType<FOutLinerSelectionLockColumn>
(SelectionColumnInfo);

    FSceneOutlinerColumnInfo ResetingColumnInfo(
        ESceneOutlinerColumnVisibility::Visible,
        2,

FCreateSceneOutlinerColumn::CreateRaw(this,&FSuperMangerModule::OnCreateSelectionR
estingColumns));

SceneOutlinerModule.RegisterDefaultColumnType<FOutLinerSelectionResetTransformsCol
```

```
umn>(ResetingColumnInfo);
}
```

```
class FOutLinerSelectionLockColumn : public ISceneOutlinerColumn
{
public:
    FOutLinerSelectionLockColumn(ISceneOutliner& SceneOutliner){}

//ISceneOutlinerColumn需要实现的虚函数
#pragma region ISceneOutlinerColumn
    FORCEINLINE virtual FName GetColumnID() override{return
FName("SelectionLock");};
    // Outliner中添加列的顶部构建
    virtual SHeaderRow::FColumn::FArguments ConstructHeaderRowColumn() override;
    // Outliner中添加列每一行的构造
    virtual const TSharedRef< SWidget >
ConstructRowWidget(FSceneOutlinerTreeItemRef TreeItem,
                            const STableRow<FSceneOutlinerTreeItemPtr>& Row)
override;
    static FName GetID(){return FName("SelectionLock");}
#pragma endregion
private:
    void OnRowCheckStateChanged(ECheckBoxState
NewCheckState,TWeakObjectPtr<AActor> CorrespondingActor);
};
```

如何用Menu打开自己定义的一个slot

这个在supermanger中也有体现 首先需要注册

```
void FSuperMangerModule::RegisterAdvanceDeletionTab()
{
    //Register a new nomad tab spawner with the tab manage
    //触发Tab的委托
    FGlobalTabmanager::Get()->RegisterNomadTabSpawner(
        FName("AdvanceDeletionTab"),

FOnSpawnTab::CreateRaw(this,&FSuperMangerModule::OnSpawnAdvanceDeletionTab))
    .SetDisplayName(FText::FromString(TEXT("Advance Deletion")))
    .SetIcon(FSlateIcon(FSuperMangerStyle::GetStyleSetName(),"ContentBrowser.Advan
cDeletion"));
}

TSharedRef<SDockTab> FSuperMangerModule::OnSpawnAdvanceDeletionTab(const
FSpawnTabArgs& args)
{
    //create new Slate Weight
    /*
     *这里[]之间就是一个slot(槽)
```

```
        *可以立即为一个占据窗口的小部件
        */
        if (FolderPathSelected.Num() == 0)
        {
            return SNew(SDockTab).TabRole(NomadTab);
        }
        ConstructedDockTab = SNew(SDockTab).TabRole(ETabRole::NomadTab)
        [
            SNew(SAdvanceDeletionTab)
            .AssetsDataArray(GetAllAssetDataUnderSelectedFolder())
            .CurrentSelectFodler(FolderPathSelected[0])
        ];
        // 设置关闭是触发的函数 -- 将ConstructedDockTab设置为空
        ConstructedDockTab->SetOnTabClosed(

    SDockTab::FOnTabClosedCallback::CreateRaw(this,&FSuperMangerModule::OnAdvanceDelet
    ionTabClosed));
        return ConstructedDockTab.ToSharedRef();
    }
```

注册完毕后就使用tryinvoke方法调用这个FName

```
void FSuperMangerModule::OnAdvanceDeletionButtonClicked()
{
    /*DebugHeader::Print(TEXT("Working!"),FColor::Green);*/
    FixUpRedirectors();
    // !这里name要和slate中注册的一致(RegisterNomadTabSpawner)
    FGlobalTabmanager::Get()->TryInvokeTab(FName("AdvanceDeletionTab"));
}
```

当然使用完后要接触注册

参考资料

总集篇 https://zhuanlan.zhihu.com/p/604684703 如何为每一个专栏添加内容 https://zhuanlan.zhihu.com/p/605181368