一、Graph

创建空白图表

1. UEdGraph

图表对象

2. UEdGraphSchema

负责管理节点等，例如创建节点菜单、可视化连接、节点右键菜单

```
//创建节点菜单，构建API菜单。创建节点在FEdGraphSchemaAction的实例对象中执行
virtual void GetGraphContextActions(FGraphContextMenuBuilder&
ContextMenuBuilder) const override;
//节点右键菜单
virtual void GetContextMenuActions(class UToolMenu* Menu, class
UGraphNodeContextMenuContext* Context) const override;
//设置节点连接规则
virtual const FPinConnectionResponse CanCreateConnection(const UEdGraphPin* A,
const UEdGraphPin* B) const override;
//自定义连线绘制规则。默认返回null，则使用赛贝尔曲线。
virtual class FConnectionDrawingPolicy* CreateConnectionDrawingPolicy(int32
InBackLayerID, int32 InFrontLayerID, float InZoomFactor, const FSlateRect&
InClippingRect, class FSlateWindowElementList& InDrawElements, class UEdGraph*
InGraphObj) const override;
```

3. SGraphEditor

图表Slate

示例代码

```
//初始化GraphObject
UEdGraph* GraphObject = NewObject<UEdGraph>();
GraphObject->AddToRoot();
GraphObject->Schema = UTestEdGraphSchema::StaticClass();

//初始化GraphEditor
SNew(SGraphEditor).GraphToEditor(GraphObject);
```

添加自定义节点

1. 自定义Schema

FEdGraphSchemaAction

每个实例对应节点菜单中的一个节点方法

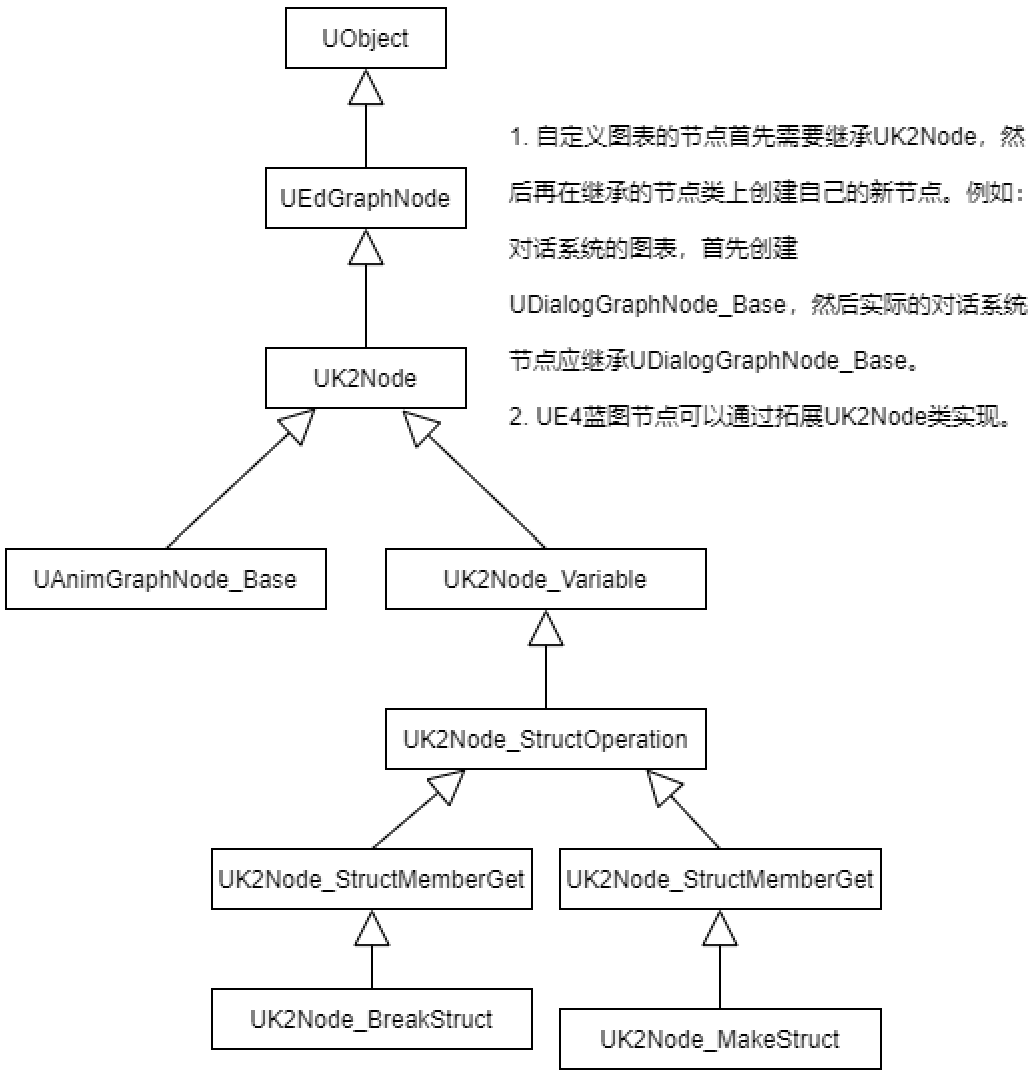
```
USTRUCT()
struct FTestGraphSchemaAction : public FEdGraphSchemaAction
{
    GENERATED_BODY()
 public:
    FTestGraphSchemaAction(){}
    FTestGraphSchemaAction(...){...}

    //创建节点：点击方法菜单中方法节点时调用该方法
    virtual UEdGraphNode* PerformAction(UEdGraph* ParentGraph, UEdGraphPin*
FromPin, const FVector2D Location, bool bSelectNewNode) override
    {
        UEdGraphNode* EdResultNode = nullptr;
        if(TestNode != nullptr)
        {
            const FScopedTransaction Transaction(LOCTEXT("Cancel",
"Cancel:TestNode"));     //支持撤销
            ParentGraph->Modify();
            if(FromPin != nullptr)
            {
                FromPin->Modify();
            }
            TestNode->Rename(nullptr, ParentGraph);
            /*
                图表添加节点
                @param 要添加的节点
                @param 是否由用户添加
                @param 是否是连线时要创建的节点
            */
            ParentGraph->AddNode(TestNode, true, bSelectNewNode);
            TestNode->CreateNewGuid();          //创建节点标识
            TestNode->PostPlaceNewNode();       //初始化节点
            TestNode->AllocateDefaultPins();    //创建针脚
            TestNode->AutowireNewNode(FromPin); //自动写入一个新节点？
            //设置节点位置
            TestNode->NodePosX = Location.X;
            TestNode->NodePosY = Location.Y;

            TestNode->SetFlags(RF_Transactional);   //和回收有关
            EdResultNode = TestNode;
        }

        return EdResultNode;
    }

public:
    class UMyNode_Test* TestNode;
}
```

UEdGraphSchema

```
//1. 创建节点菜单：右键图表显示方法菜单时，将调用该方法构造菜单内容
virtual void GetGraphContextActions(FGraphContextMenuBuilder&
ContextMenuBuilder) const override
{
    TSharedPtr<FTestGraphSchemaAction> NewNodeAction(
        FTestGraphSchemaAction(
            LOCTEXT("NodeTag", "Test"),     //目录
            LOCTEXT("NodeName", "TestNode"),        //节点在选择列表中的名称
            LOCTEXT("NodeHint", "This is test node."),  //说明，提示Hint
            0)      //分组，其实就是显示的优先级
    );

    NewNodeAction->TestNode = NewObject<UMyNode_Test>
(ContextMenuBuilder.OwnerOfTemporaries);   //OwnerOfTemporaries储存临时的模板节点
    ContextMenuBuilder.AddAction(NewNodeAction);
}

//创建节点：创建节点是调用的FEdGraphSchemaAction中的PerformAction方法
```

2. 创建节点

```
UCLASS(MinimalAPI)
class UMyNode_Test : public UEdGraphNode
{
    GENERATED_BODY()
public:
    UMyNode_Test();

    //创建针脚
    virtual void AllocateDefaultPins() override
    {
        /*
        @param EEdGraphPinDirection,有三种类型：EGPD_Input, EGPD_Output, EGPD_MAX。
所以实际使用就是输出输出的枚举。
        @param 类型组
        @param 次级类型组
        @param 针脚名
        */
        CreatePin(EGPD_Input, "Test", FName(), TEXT("TestIn"));
    }
    //节点名称
    virtual FText GetNodeTitle(ENodeTitleType::Type TitleType) const override
    {
        return FText::FromString("TestNode");
    }

}
```



1. 自定义图表的节点首先需要继承UK2Node，然后再在继承的节点类上创建自己的新节点。例如：对话系统的图表，首先创建UDialogGraphNode_Base，然后实际的对话系统节点应继承UDialogGraphNode_Base。
2. UE4蓝图节点可以通过拓展UK2Node类实现。

蓝图节点

如果希望继承UK2Node的自定义节点在普通蓝图中显示和调用，则需要实现GetMenuActions方法。该方法实现节点的创建。

```
//该方法用于替代EdGraphNode中的GetMenuEntries方法，可以将自身节点添加到节点上下文菜单中
virtual void GetMenuActions(FBlueprintActionDatabaseRegistrar& ActionRegistrar)
const override;
```

```
//自定义节点样式，如果没有实现该方法，则从节点工厂中查找可用节点UI
virtual TSharedPtr<SGraphNode> CreateVisualWidget() override;
```

自定义连接样式

UEdGraphSchema

```
/*
    @param 连接线的ID
    @param 连接线箭头的ID
    @param 图表缩放比例
    @param 图表视口数据
    @param 要绘制的元素列表
    @param 图表对象
*/
virtual class FConnectionDrawingPolicy* CreateConnectionDrawingPolicy(int32
InBackLayerID, int32 InFrontLayerID, float InZoomFactor, const FSlateRect&
InClippingRect, class FSlateWindowElementList& InDrawElements, class UEdGraph*
InGraphObj) const override
{
    return new FTestConnectionDrawingPolicy(InBackLayerID, InFrontLayerID,
InZoomFactor, InClippingRect, InDrawElements, InGraphObj);
}
```

FConnectionDrawingPolicy

```
class FTestConnectionDrawingPolicy : public FConnectionDrawingPolicy
{
public:
     FConnectionDrawingPolicy(int32 InBackLayerID, int32 InFrontLayerID, float
InZoomFactor, const FSlateRect& InClippingRect, FSlateWindowElementList&
InDrawElements, UEdGraph* InGraphObj)
         : FConnectionDrawingPolicy(InBackLayerID, InFrontLayerID, InZoomFactor,
InClippingRect, InDrawElements)
         , EdGraphObj(InGraphObj)
     {}

    //定义连接线样式
    virtual void DetermineWiringStyle(UEdGraphPin* OutPin, UEdGraphPin*
InputPin, FConnectionParams& Params) override
    {
        Params.WireThickness = 5.5f;            //连接线粗细
        Params.WireColor = DefaultWiringColor;  //连接线颜色

        if(HoveredPins.Num() > 0)
        {
            //拖拽时的样式
            ApplyHoverDeemphasis(OutputPin, InputPin, Params.WireThickness,
Params.WireColor);
        }
    }

    //如果不复写该方法，将默认使用赛贝尔曲线的连接线，该方法在线条存在期间一直执行
    virtual void DrawConnection(int32 LayerId, const FVector2D& Start, const
FVector2D& End, const FConnectionParams& Params)
    {
        //改成直线连接
        const FVector2D Delta = End - Start;
        const FVector2D DirDelta = Delta.GetSafeNormal();
        FSlateDrawElement::MakeDrawSpaceSpline(
            DrawElementsList,
            LayerId,                //层级
            Start, DirDelta,        //开始位置和方向
            End, DirDelta,          //结束位置和方向
            Params.WireThickness,   //粗细
            ESlateDrawEffect::None, //
            Params.WireColor        //颜色
        );
    }

protected:
    UEdGraph* EdGraphObj;

    TMap<UEdGraphNode*, int32> EdNodeWidgetMap;
}
```

自定义节点样式

FGraphPanelNodeFactory

节点工厂可以对指定节点创建并返回节点的SlateUI：

1. 继承FGraphPanelNodeFactory并实现节点工厂类，主要就是Create方法
2. 通过FEdGraphUtilities::RegisterVisualNodeFactory方法将工厂类注册

还有上面讲的另外一种更为直接的创建方法，就是直接在Node类中实现CreateVisualWidget方法，返回一个指定的SlateUI，如果未指定才会通过默认工厂类或自定义节点工厂类来创建一个自定义节点UI。

```
class FMyGraphPanelNodeFactory : public FGraphPanelNodeFactory
{
public:
    FMyGraphPanelNodeFactory() {}
    virtual TSharedPtr<class SGraphNode> Create(class UEdGraphNode* Node) const
    {
        //我们新创建的节点工厂类只处理我们创建的自定义节点，所以检查是否能转换为自定义节点基类
        if (UMyNode_Test* MarkerNode = Cast<UMyNode_Test>(Node))
        {
            return SNew(SMyGraphNode_Test, MarkerNode);
        }
        return NULL;
    }
}
```

在模块启动时，将节点工厂类注册

```
FEdGraphUtilities::RegisterVisualNodeFactory(MakeShareable(new
FMyGraphPanelNodeFactory));
```

SGraphNode

如果图表也是自定义的，我们可以直接继承SGrahNode，如果是在K2蓝图基础上拓展节点，我们可以继承SGraphNodeK2Base。

```
class SMyGraphNode_Test : public SGraphNode
{
public:
    SLATE_BEGIN_ARGS(SMyGraphNode_Test)
    {}

    SLATE_END_ARGS()

    void Construct(const FArguments& InArgs, UMyNode_Test* MarkerNode)
    {
        GraphNode = MarkerNode;
        this->SetCursor(EMouseCursor:Hand);
        this->UpdateGraphNode();
    }

    virtual void UpdateGraphNode() override
    {
        //输入输出数据置空
        InputPins:Empty();
        OutputPins:Empty();
        //UI清空
        RightNodeBox.Reset();
        LeftNodeBox.Reset();

        const FSlateBrush* MyNodeIcon =
FEditorStyle::GetBrush(TEXT("Graph.StateNode.Icon"));

        this->GetOrAddSlot(ENodeZone::Center)
            .HAlign(HAlign_Center)
            .VAlign(VAlign_Center)
            [
                SAssignNew(PinBox, SBox)
                .HAlign(HAlign_Fill)
                .VAlign(VAlign_Fill)
                [
                    SNew(SBorder)
                    .BorderBackgroundColor_Lambda([&]()
                    {
                        FSlateColor SlateColor(FLinearColor(1.f, 1.f, 1.f));
                        return SlateColor;
                    })
                    [
                        SNew(SHorizontalBox)
                        + SHorizontalBox::Slot()
                        .HAlign(HAlign_Right)
                        .VAlign(VAlign_Fill)
                        .AutoWidth()
                        [
                        .....
                    ]
                ]
            ]
        ];
        CreatePinWidgets();
    }
    //创建Pin，自定义pin在下一节
    virtual void CreatePinWidgets() override
    {
        UMyNode_Test* TestNode = CastChecked<UMyNode_Test>(GraphNode);
        if(TestNode)
        {
            for(UEdGraphPin* CurPin : TestNode->Pins)
            {
                TSharedPtr<SGraphPin> NewPin = SNew(SMyGraphPin_Test, CurPin);
                NewPin->SetIsEditable(IsEditable);

                this->AddPin(NewPin.ToSharedRef());

                if(CurPin->Direction == EEdGraphPinDirection::EGPD_Input)
                {
                    InputPins.Add(NewPin.ToSharedPtr());
                }
                if(CurPin->Direction == EEdGraphPinDirection::EGPD_Output)
                {
                    OutputPins.Add(NewPin.ToSharedPtr());
                }
            }
        }
    }

protected:
    TSharedPtr<class SBox> PinBox;
}
```

自定义Pin

SGraphPin

```
class SMyGraphPin_Test : public SGraphPin
{
public:
    ...
    void Construct(const FArguments& InArgs, UEdGraphPin* InPin)
    {
        this->SetCursor(EMouseCursor::Hand);
        GraphPinObj = InPin;
        check(GraphPinObj);

        const UEdGraphSchema* Schema = GraphPinObj->GetSchema();
        check(Schema);
        SBorder::Construct(SBorder::FArguments()
        .BorderBackgroundColor_Lambda()...
        );
    }
}
```

工厂解耦

NodeFactory

FNodeFactory::CreatePin...

NodePinFactory

GraphPanelPinConnectionPolicyFactory