自定义蓝图方法

K2Node

参考文章：

https://zhuanlan.zhihu.com/p/84958215 （通过派生class UK2Node创建自定义蓝图节点）

https://zhuanlan.zhihu.com/p/87489168 （自定义蓝图节点的编译过程）

Thunk函数

参考文章：https://zhuanlan.zhihu.com/p/84956153

实例1. 接受任意UStruct类型参数

BlueprintWildcardLibrary.h

```
USTRUCT(BlueprintInternalUseOnly)
struct FDummyStruct {
    GENERATED_USTRUCT_BODY()

};

UCLASS()
class UNREALCOOKBOOK_API UBlueprintWildcardLibrary : public
UBlueprintFunctionLibrary {
    GENERATED_BODY()

public:
    UFUNCTION(BlueprintCallable, CustomThunk, Category = "MyDemo", meta =
(CustomStructureParam = "CustomStruct"))
        static void ShowStructFields(const FDummyStruct& CustomStruct);
    static void Generic_ShowStructFields(const void* StructAddr, const
UStructProperty* StructProperty);

    DECLARE_FUNCTION(execShowStructFields) {

        Stack.MostRecentPropertyAddress = nullptr;
        Stack.MostRecentProperty = nullptr;

        Stack.StepCompiledIn<UStructProperty>(NULL);
        void* StructAddr = Stack.MostRecentPropertyAddress;
        UStructProperty* StructProperty = Cast<UStructProperty>
(Stack.MostRecentProperty);


        P_FINISH;

        P_NATIVE_BEGIN;
        Generic_ShowStructFields(StructAddr, StructProperty);
        P_NATIVE_END;
```

```
    }
};
```

BlueprintWildcardLibrary.cpp

```
#include "BlueprintWildcardLibrary.h"
#include "Engine/Engine.h"

void UBlueprintWildcardLibrary::Generic_ShowStructFields(const void* StructAddr,
const UStructProperty* StructProperty) {
    UScriptStruct* Struct = StructProperty->Struct;
    for (TFieldIterator<UProperty> iter(Struct); iter; ++iter) {

        FScreenMessageString NewMessage;
        NewMessage.CurrentTimeDisplayed = 0.0f;
        NewMessage.Key = INDEX_NONE;
        NewMessage.DisplayColor = FColor::Blue;
        NewMessage.TimeToDisplay = 5;
        NewMessage.ScreenMessage = FString::Printf(TEXT("Property: [%s].[%s]"),
            *(Struct->GetName()),
            *(iter->GetName())
        );
        NewMessage.TextScale = FVector2D::UnitVector;
        GEngine->PriorityScreenMessages.Insert(NewMessage, 0);
    }
}
```

解释一下这段代码：

1. 首先声明了一个UFunction：`static void ShowStructFields(const FDummyStruct& CustomStruct);`，其参数类型是“FDummyStruct”，这只是一个占位符；
2. 在UFUNCTION宏里面指定“CustomThunk”和“CustomStructureParam”；
3. 实现一个`execShowStructFields`函数。这个函数很简单，主要是处理蓝图的Stack，从中取出需要的参数，然后对用C++的实现；
4. 具体功能实现在：`static void Generic_ShowStructFields(const void* StructAddr, const UStructProperty* StructProperty)`这个函数中。

实例2. 对数组中的Struct的数值型求平均

参照引擎源代码，我定义了这样一个宏，用来从栈上取出泛型数组参数，并正确的移动栈指针：

```
#define P_GET_GENERIC_ARRAY(ArrayAddr, ArrayProperty) Stack.MostRecentProperty =
nullptr;\
        Stack.StepCompiledIn<UArrayProperty>(NULL);\
        void* ArrayAddr = Stack.MostRecentPropertyAddress;\
        UArrayProperty* ArrayProperty = Cast<UArrayProperty>
(Stack.MostRecentProperty);\
        if (!ArrayProperty) {   Stack.bArrayContextFailed = true;   return; }
```

通过这个宏，可以得到两个局部变量：

- `void\* ArrayAddr`：数组的起始内存地址；

- `UArrayProperty* ArrayProperty`：数组的反射信息，`ArrayProperty->Inner`就是数组成员对应的类型了；

有了这个宏，我们就可以很方便的写出thunk函数了：

```
DECLARE_FUNCTION(execArray_NumericPropertyAverage) {

        // get TargetArray
        P_GET_GENERIC_ARRAY(ArrayAddr, ArrayProperty);

        // get PropertyName
        P_GET_PROPERTY(UNameProperty, PropertyName);

        P_FINISH;

        P_NATIVE_BEGIN;
        *(float*)RESULT_PARAM = GenericArray_NumericPropertyAverage(ArrayAddr,
ArrayProperty, PropertyName);
        P_NATIVE_END;
    }
```

经过以上的准备，我们就已经可以正确的处理"泛型数组"了。下一步就是对这个数组中指定的数"值类型成员变量"求均值了，这主要依靠Unreal的反射信息，一步步抽丝剥茧，找到数组中的每个变量即可。反射系统的使用不是本文的重点，先看完整代码吧。

BlueprintWildcardLibrary.h

```
#pragma once

#include "CoreMinimal.h"
#include "Kismet/BlueprintFunctionLibrary.h"
#include "BlueprintWildcardLibrary.generated.h"

#define P_GET_GENERIC_ARRAY(ArrayAddr, ArrayProperty) Stack.MostRecentProperty =
nullptr;\
        Stack.StepCompiledIn<UArrayProperty>(NULL);\
        void* ArrayAddr = Stack.MostRecentPropertyAddress;\
        UArrayProperty* ArrayProperty = Cast<UArrayProperty>
(Stack.MostRecentProperty);\
        if (!ArrayProperty) {   Stack.bArrayContextFailed = true;   return; }

UCLASS()
class UNREALCOOKBOOK_API UBlueprintWildcardLibrary : public
UBlueprintFunctionLibrary {
    GENERATED_BODY()

public:

    UFUNCTION(BlueprintPure, CustomThunk, meta = (DisplayName = "Array Numeric
Property Average", ArrayParm = "TargetArray", ArrayTypeDependentParams =
"TargetArray"), Category = "MyDemo")
        static float Array_NumericPropertyAverage(const TArray<int32>&
TargetArray, FName PropertyName);
    static float GenericArray_NumericPropertyAverage(const void* TargetArray,
const UArrayProperty* ArrayProperty, FName ArrayPropertyName);
```

```
public:
    DECLARE_FUNCTION(execArray_NumericPropertyAverage) {

        // get TargetArray
        P_GET_GENERIC_ARRAY(ArrayAddr, ArrayProperty);

        // get PropertyName
        P_GET_PROPERTY(UNameProperty, PropertyName);

        P_FINISH;

        P_NATIVE_BEGIN;
        *(float*)RESULT_PARAM = GenericArray_NumericPropertyAverage(ArrayAddr,
ArrayProperty, PropertyName);
        P_NATIVE_END;
    }
};
```

BlueprintWildcardLibrary.cpp

```
#include "BlueprintWildcardLibrary.h"
#include "Engine/Engine.h"

float UBlueprintWildcardLibrary::Array_NumericPropertyAverage(const
TArray<int32>& TargetArray, FName PropertyName) {
    // We should never hit these!  They're stubs to avoid NoExport on the class.
Call the Generic* equivalent instead
    check(0);
    return 0.f;
}

float UBlueprintWildcardLibrary::GenericArray_NumericPropertyAverage(const void*
TargetArray, const UArrayProperty* ArrayProperty, FName PropertyName) {

    UStructProperty* InnerProperty = Cast<UStructProperty>(ArrayProperty-
>Inner);
    if (!InnerProperty) {
        UE_LOG(LogTemp, Error, TEXT("Array inner property is NOT a UStruct!"));
        return 0.f;
    }

    UScriptStruct* Struct = InnerProperty->Struct;
    FString PropertyNameStr = PropertyName.ToString();
    UNumericProperty* NumProperty = nullptr;
    for (TFieldIterator<UNumericProperty> iter(Struct); iter; ++iter) {
        if (Struct->PropertyNameToDisplayName(iter->GetFName()) ==
PropertyNameStr) {
            NumProperty = *iter;
            break;
        }
    }
    if (!NumProperty) {
        UE_LOG(LogTemp, Log, TEXT("Struct property NOT numeric = [%s]"),
            *(PropertyName.ToString())
```

```
        );
    }


    FScriptArrayHelper ArrayHelper(ArrayProperty, TargetArray);
    int Count = ArrayHelper.Num();
    float Sum = 0.f;

    if(Count <= 0)
        return 0.f;

    if (NumProperty->IsFloatingPoint())
        for (int i = 0; i < Count; i++) {
            void* ElemPtr = ArrayHelper.GetRawPtr(i);
            const uint8* ValuePtr = NumProperty->ContainerPtrToValuePtr<uint8>
(ElemPtr);
            Sum += NumProperty->GetFloatingPointPropertyValue(ValuePtr);

        }
    else if (NumProperty->IsInteger()) {
        for (int i = 0; i < Count; i++) {
            void* ElemPtr = ArrayHelper.GetRawPtr(i);
            const uint8* ValuePtr = NumProperty->ContainerPtrToValuePtr<uint8>
(ElemPtr);
            Sum += NumProperty->GetSignedIntPropertyValue(ValuePtr);
        }
    }
    // TODO: else if(enum类型)

    return Sum / Count;
}
```

蓝图编译

参考文章：https://zhuanlan.zhihu.com/p/92268112