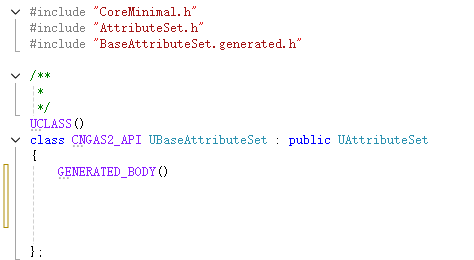
用C++来实现

在内容文件夹中，在“All--C++类--<项目名>--Public”中创建新的C++类，父类是所有类中的AttributeSet类，Public类型，命名为BaseAttributeSet。

在BaseAttributeSet.h中



接下来在类中定义，并添加头文件

#include "AbilitySystemComponent.h"



‘’’

#pragma once

#include "CoreMinimal.h"

#include "AttributeSet.h"

#include "AbilitySystemComponent.h"///需要放在generated.h之前

#include "BaseAttributeSet.generated.h"

/\*\*

\*

\*/

#define ATTRIBUTE\_ACCESSORS(ClassName, PropertyName) \

GAMEPLAYATTRIBUTE\_PROPERTY\_GETTER(ClassName, PropertyName) \

GAMEPLAYATTRIBUTE\_VALUE\_GETTER(PropertyName) \

GAMEPLAYATTRIBUTE\_VALUE\_SETTER(PropertyName) \

GAMEPLAYATTRIBUTE\_VALUE\_INITTER(PropertyName)

UCLASS()

class CNGAS2\_API UBaseAttributeSet : public UAttributeSet

{

GENERATED\_BODY()

public:

//生命值

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "BaseAttributeSet")

FGameplayAttributeData HP;

ATTRIBUTE\_ACCESSORS(UBaseAttributeSet, HP);

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "BaseAttributeSet")

FGameplayAttributeData MaxHP;

ATTRIBUTE\_ACCESSORS(UBaseAttributeSet, MaxHP);

//法力

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "BaseAttributeSet")

FGameplayAttributeData MP;

ATTRIBUTE\_ACCESSORS(UBaseAttributeSet, MP);

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "BaseAttributeSet")

FGameplayAttributeData MaxMP;

ATTRIBUTE\_ACCESSORS(UBaseAttributeSet, MaxMP);

//能量

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "BaseAttributeSet")

FGameplayAttributeData SP;

ATTRIBUTE\_ACCESSORS(UBaseAttributeSet, SP);

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "BaseAttributeSet")

FGameplayAttributeData MaxSP;

ATTRIBUTE\_ACCESSORS(UBaseAttributeSet, MaxSP);

};

‘’’

首先公有的才能有BlueprintReadWrite，来暴露给蓝图去可读可写。

Category是分类

然后回到虚幻编辑器，在右下角点击重新编译



此时编译错误。我们回到VisualStudio，在VisualStudio里生成。

有时并不需要关注报错，只需要能通过VisualStudio打开虚幻编辑器即可。

## 初始化属性值，并给普攻附加伤害

初始化属性值。首先我们要做一个表格。

创建DataTable，行结构为AttributeMetaData，命名为DT\_CharacterAttribute

打开数据表格，数据表格自带相当的数据



然后添加行。注意，添加行之后，行名称是很有讲究的。是根据我在C++中怎么定义的，然后再去怎么命名。

命名应当是 BaseAttributeSet.HP 这样的格式。

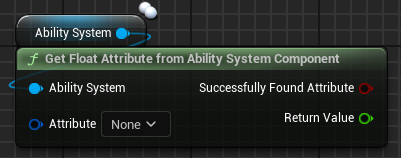
这里有最大最小值，最小值是0.这里Epic并没有写好，所以我们要自己写伤害溢出的处理。



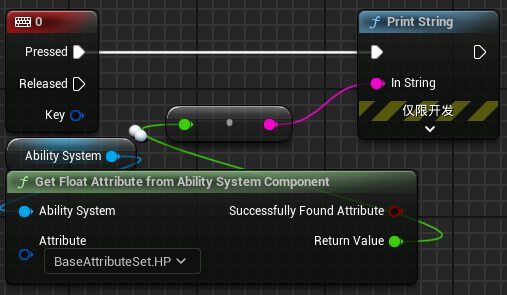
然后进入BP\_BaseCharacter，在AbilitySystem组件的细节中“属性测试--默认起始数据”添加一个索引，索引0的默认起始表为刚创建的表格，属性为BaseAttributeSet。



为了获取表格中的数据，通过Ability System节点的Get Float Attribute From Ability System Component函数即可



然后我们就可以在Attribute中获取。



这样就完成了属性的初始化和获取。

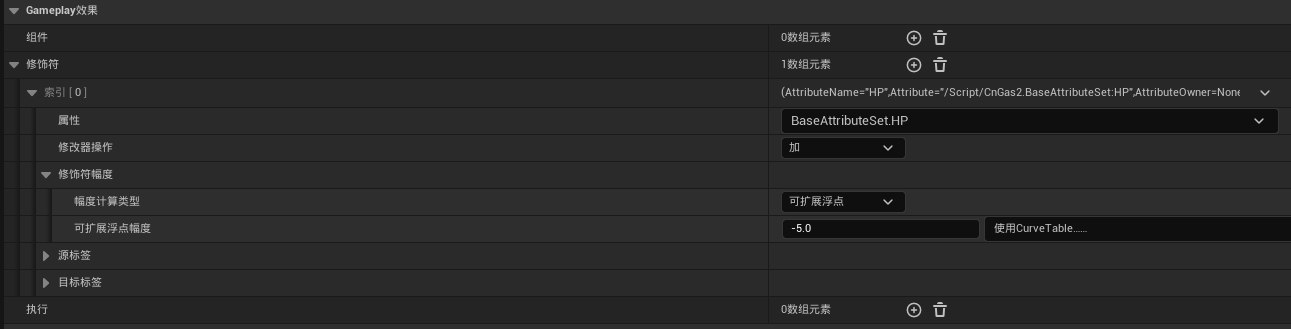
接下来实现普攻伤害。

普攻伤害属于GE，创建蓝图，类型为GameplayEffect，命名为GE\_Melee\_Damage

它是一次性伤害，持续时间策略为实时。

Gameplay效果中的修饰符（Modify）添加一个数组元素

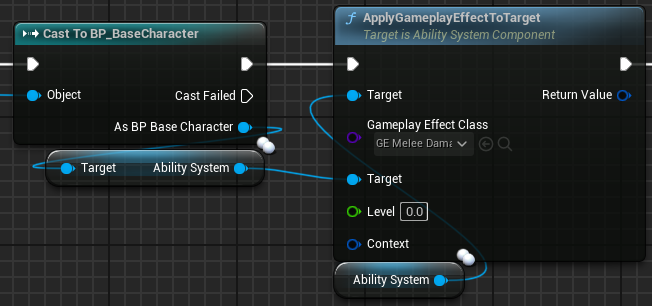
属性为BaseAttributeSet.HP，因为伤害是给HP进行计算，伤害值为5，所以可扩展浮点幅度为-5（因为修改器操作是加）

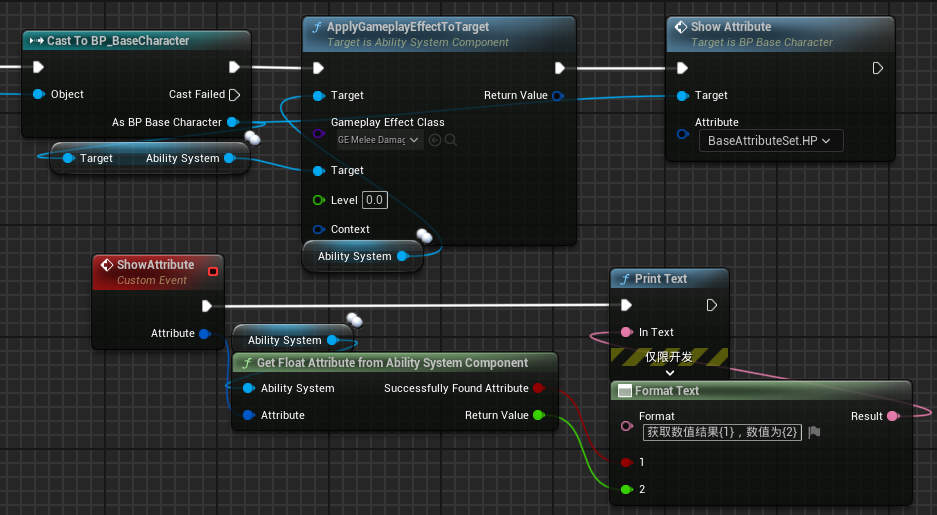


应用伤害就是对AbilitySystem进行ApplyGameplayEffectToTarget即可。

这些一般是要写在GAB\_Melee中的，但是因为普攻太简单了，我们直接写在BaseCharacter中。

当OnComponentBeginOverlap时，调用Apply函数





这样就可以触发伤害。

然后我们要实现伤害溢出的处理。在C++中处理。

在基类中有个后处理函数，函数原型是

virtual void PostGameplayEffectExecute(const struct FGameplayEffectModCallbackData &Data) { }

这是个额外的计算，所有的伤害计算完后，会执行这个函数。

我们在BaseAttributeSet.cpp中重写该函数，在.h中声明该函数。

void PostGameplayEffectExecute(const struct FGameplayEffectModCallbackData& Data) override;

（不需要标记UFUNCTION，因为不在蓝图中进行处理）

在cpp中实现时，需要加上头文件（如果不加，Data不能 . 出东西）

#include "GameplayEffectExtension.h"

'''

#include "BaseAttributeSet.h"

#include "GameplayEffectExtension.h"

void UBaseAttributeSet::PostGameplayEffectExecute(const FGameplayEffectModCallbackData& Data)

{

Super::PostGameplayEffectExecute(Data);

//对父类的继承处理

if (Data.EvaluatedData.Attribute == GetHPAttribute())

{

SetHP(FMath::Clamp(GetHP(), 0.0, GetMaxHP()));

//set HP，做一个数学上的夹值

//对GetHP的下限设置为0，上限设置为GetMaxHP

//GetMaxHP就是表格里给的那个100

}

if (Data.EvaluatedData.Attribute == GetMPAttribute())

{

SetMP(FMath::Clamp(GetMP(), 0.0, GetMaxMP()));

}

if (Data.EvaluatedData.Attribute == GetSPAttribute())

{

SetSP(FMath::Clamp(GetSP(), 0.0, GetMaxSP()));

}

}

'''

**保存**后回到引擎里编译

编译成功后，再进入游戏，攻击敌人时，就可以发现，敌人的生命值不会再低于0.

（如果在sln里编译生成，那么一定会报错，报错为MSB307，报错原因是和sln中的live coding实时编译冲突了。所以要在引擎里编译）

要想得到能控制Data的头文件，可以用这样的方法：

右键Data的类型，点击Quick Action And Refactorings（VA），选择Add Include

