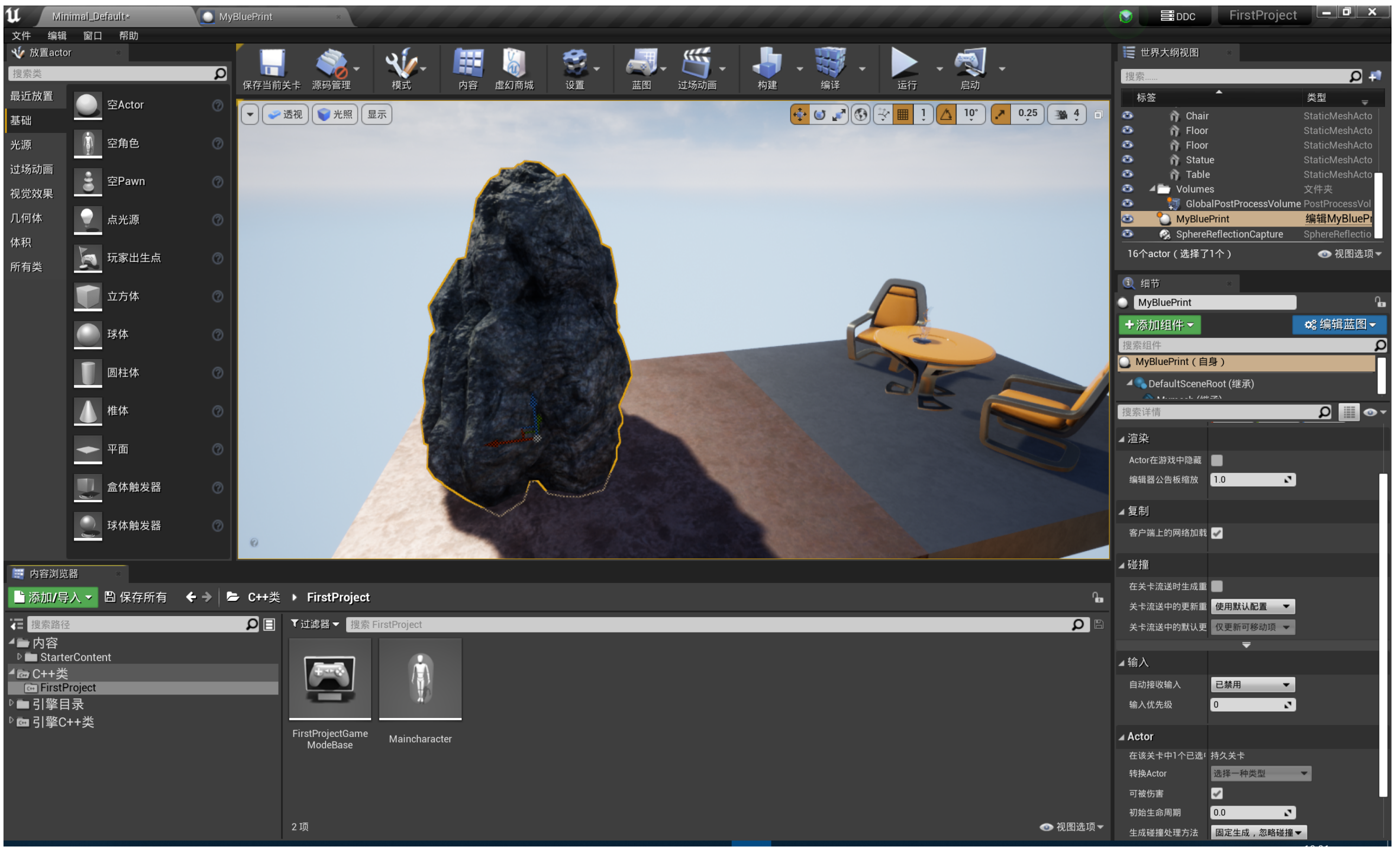
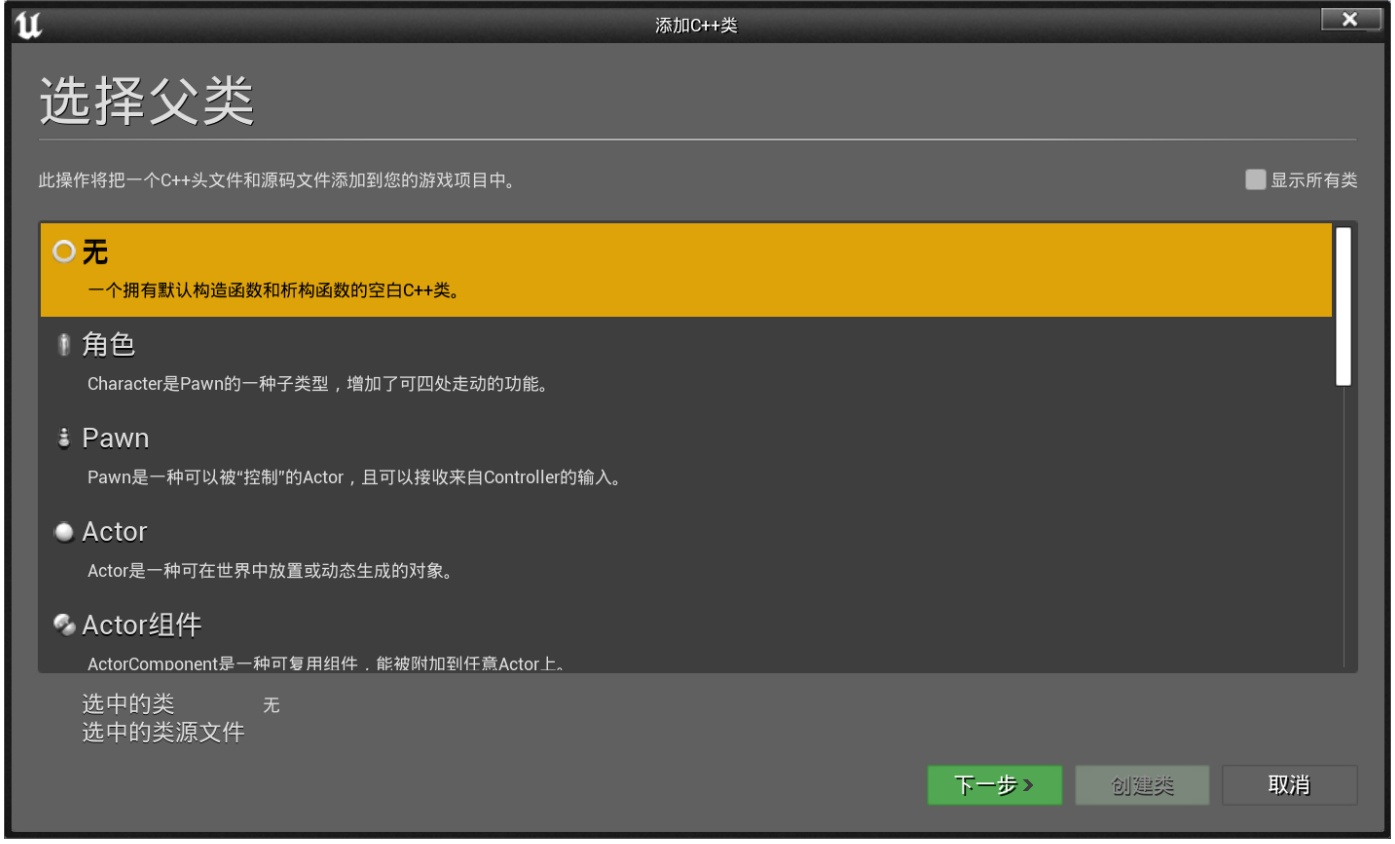
C++类的创建和使用

一、类的创建

首先我们在界面中先打开C++类的文件夹，



之后右键，创建一个类名称为Maincharacter


添加C++类

选择父类

此操作将把一个C++头文件和源码文件添加到您的游戏项目中。

无
一个拥有默认构造函数和析构函数的空白C++类。

角色
Character是Pawn的一种子类型，增加了可四处走动的功能。

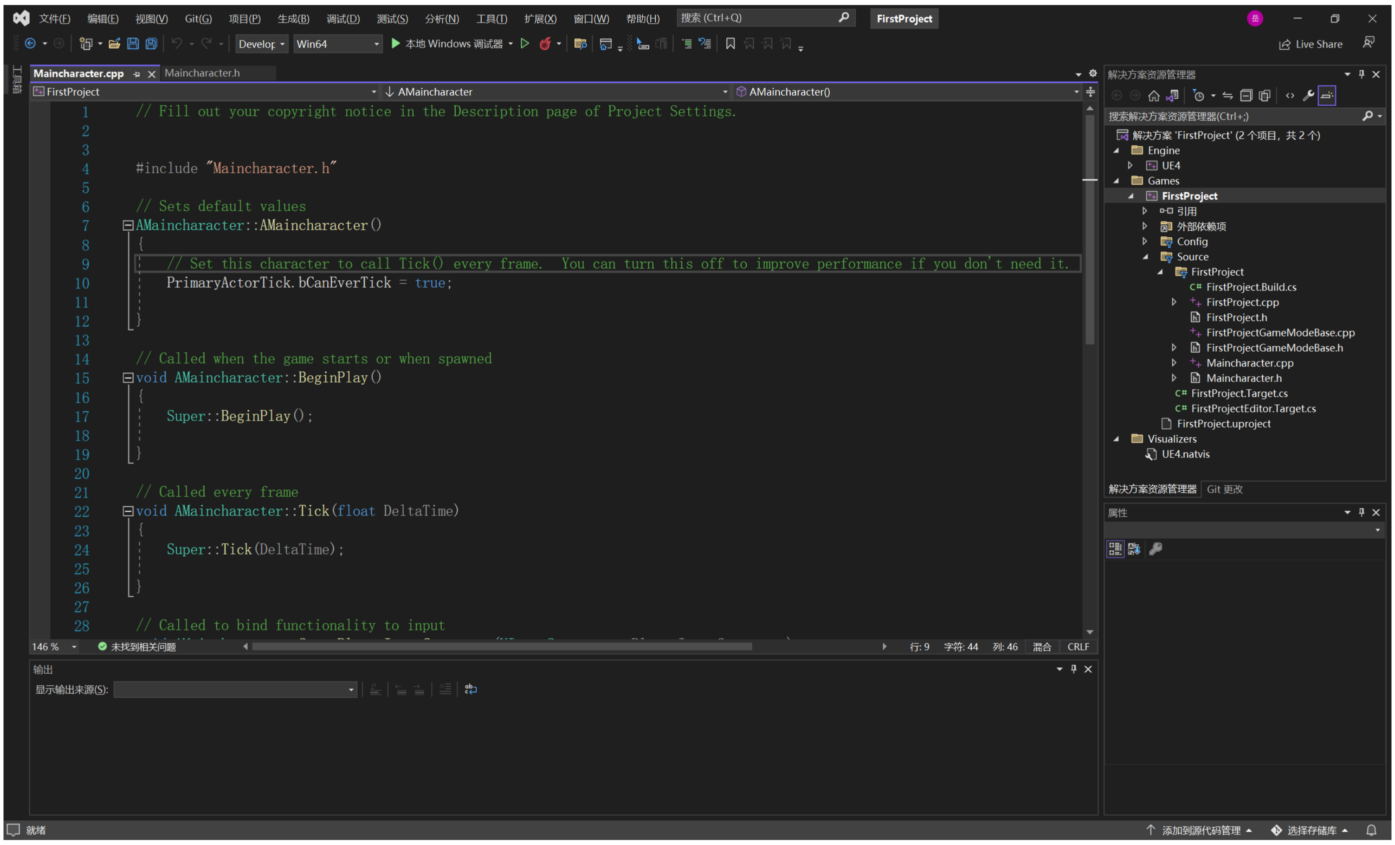
Pawn
Pawn是一种可以被“控制”的Actor，且可以接收来自Controller的输入。

Actor
Actor是一种可在世界中放置或动态生成的对象。

Actor组件
ActorComponent是一种可复用组件，能被附加到任意Actor上。

选中的类 无
选中的类源文件

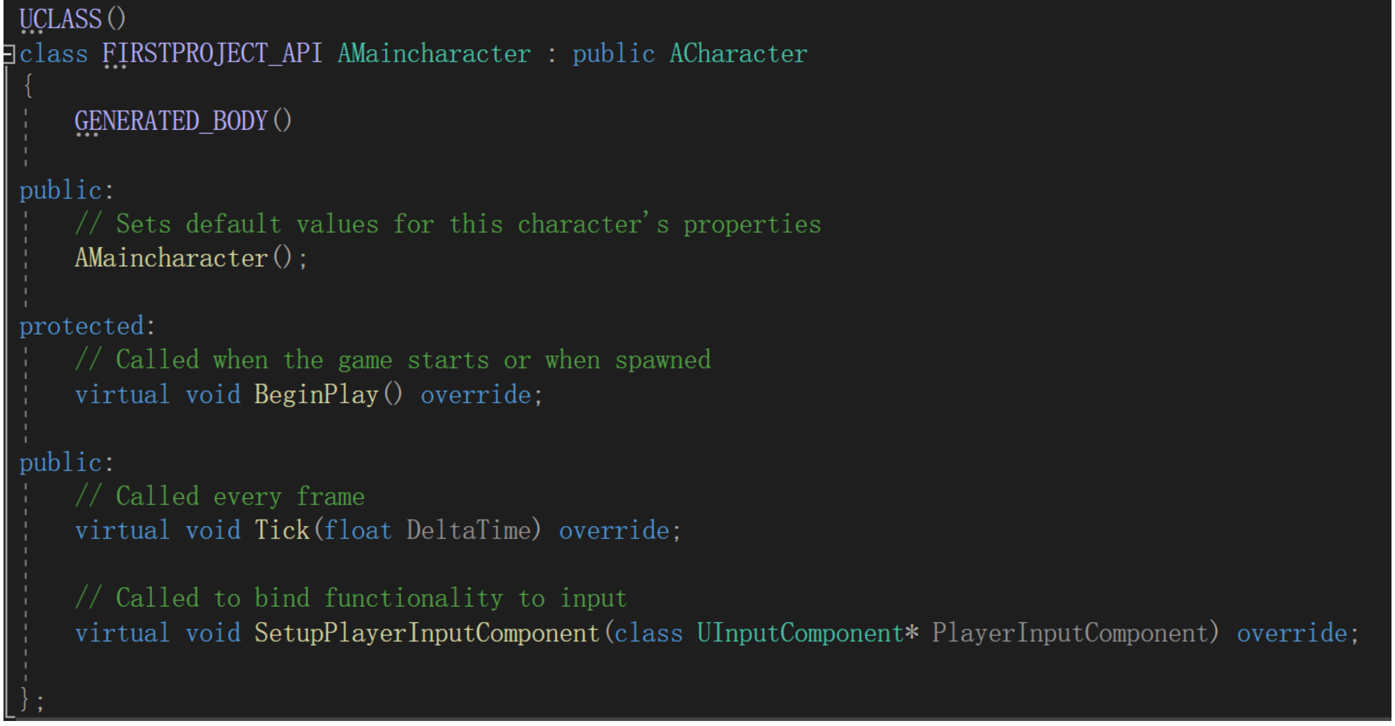
这里我们创建一个角色，之后会在visual studio中生成两个文件，分别为当前类对应的头文件以及.cpp文件，然后得到了如下的东西



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


#include "Maincharacter.h"

// Sets default values
AMaincharacter::AMaincharacter()
{
	// Set this character to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
	PrimaryActorTick.bCanEverTick = true;
}

// Called when the game starts or when spawned
void AMaincharacter::BeginPlay()
{
	Super::BeginPlay();
	
}

// Called every frame
void AMaincharacter::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

}

// Called to bind functionality to input
```

二、对自动创建的各个类的分析

首先是对我们创建的这个类的定义的分析：



```
UCLASS()
class FIRSTPROJECT_API AMaincharacter : public ACharacter
{
	GENERATED_BODY()

public:
	// Sets default values for this character's properties
	AMaincharacter();

protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;

	// Called to bind functionality to input
	virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override;

};
```

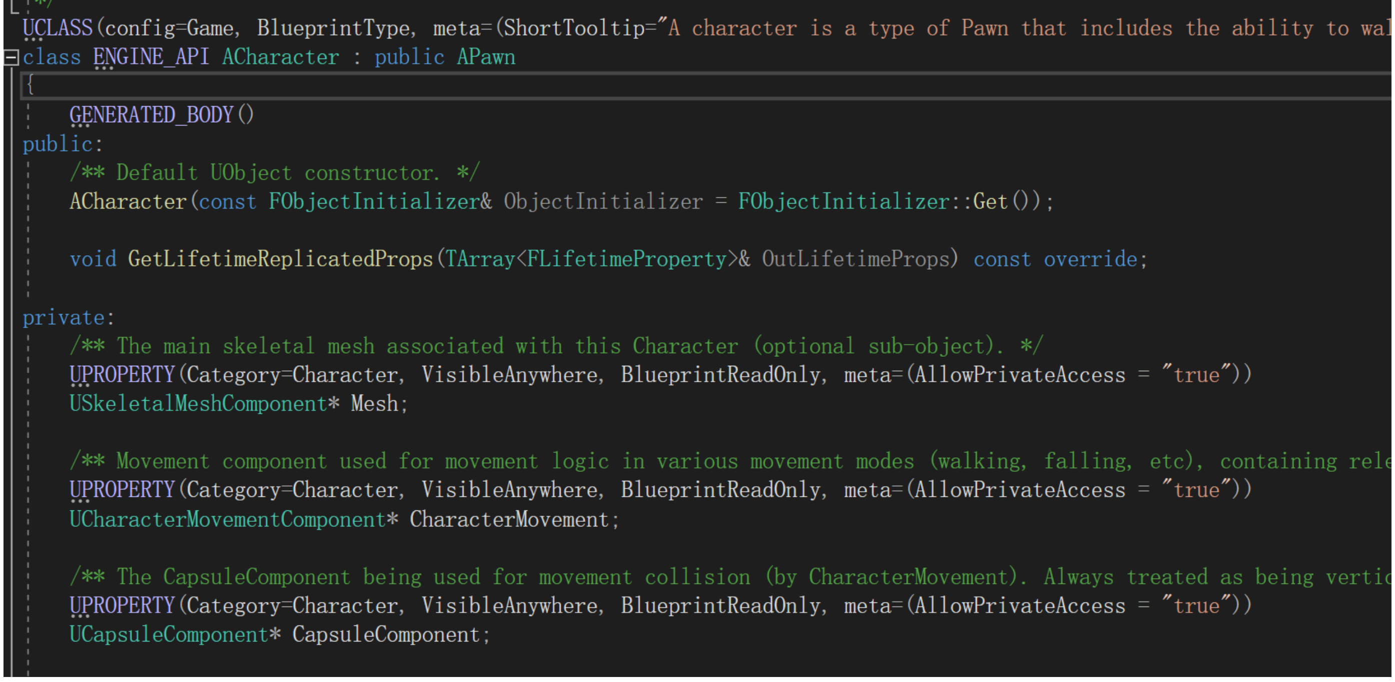
这里可以看到有一个叫做UCLASS()，这个是UE特有的宏声明，证明下面的类是可以用于UE编辑器的类；

然后我们看到这里的类名叫做Axxxx，这里的“A”代表这里的类继承自Actor，因为character是Actor的子类，也是Actor的一种，后面的那个继承自ACharacter也是一样的意思，这里就是UE内置的类了，所以说在UE创建C++类的过程中，只要是源自Actor类的对象，都会自动在前面加上“A”进行标识。

接下来我们发现，这里面创建的函数，都是虚函数，它的子类甚至可以进行重写对这些函数，这样可以极大的增加C++类的可拓展性

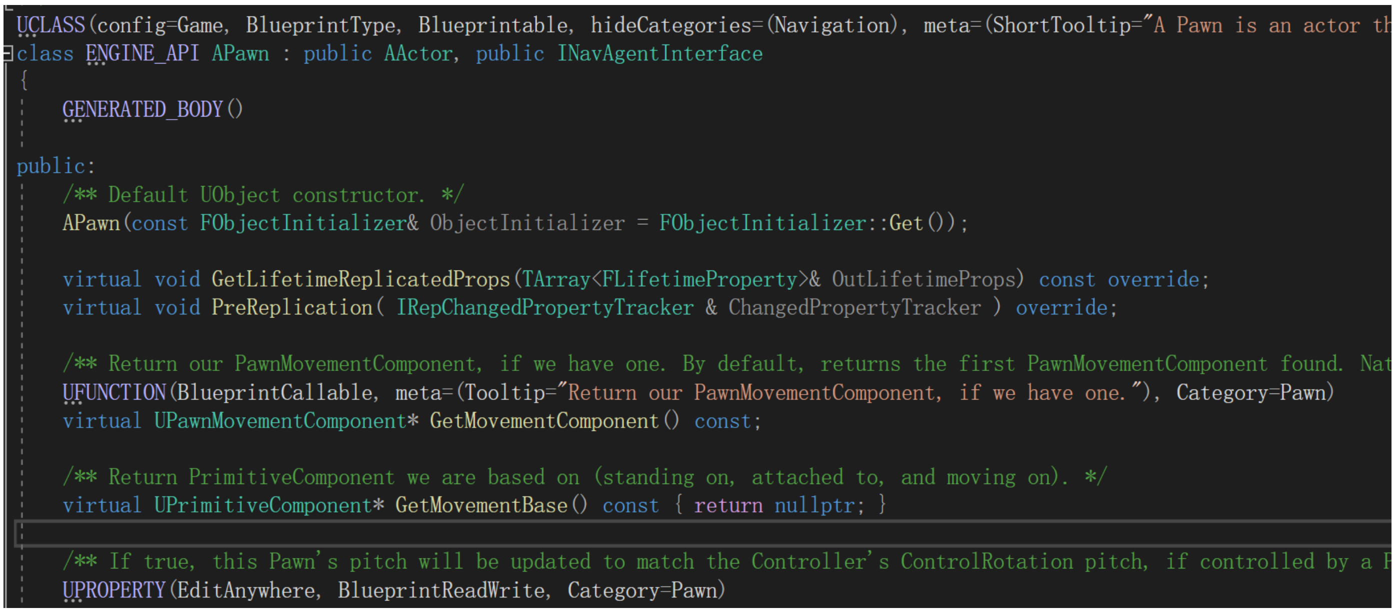
我们上一节的内容是UE4的层级关系，我们这里也可以看到UE4的层级关系，

首先我们看到了当前这个类继承自ACharacter，我们按住ctrl之后点击当前这个类，进入到这个类的定义，发现其继承自APawn



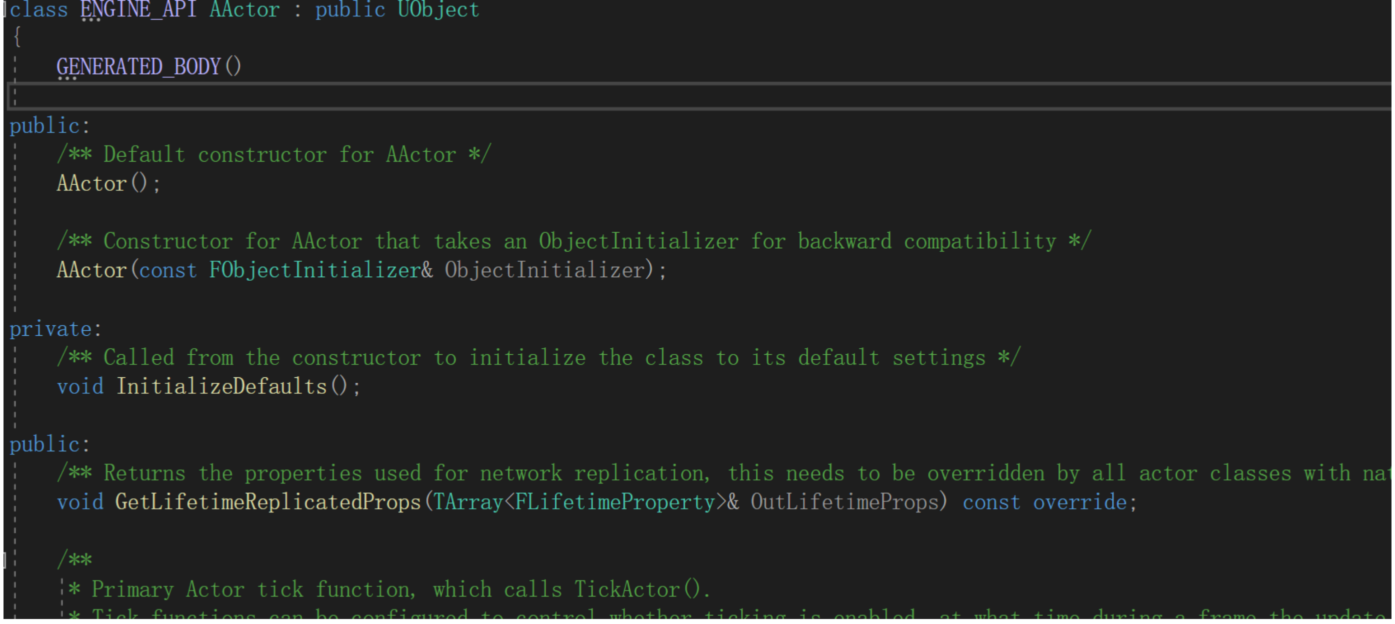
```
UCLASS(config=Game, BlueprintType, meta=(ShortTooltip="A character is a type of Pawn that includes the ability to wal
class ENGINE_API ACharacter : public APawn
{
	GENERATED_BODY()
public:
	/** Default UObject constructor. */
	ACharacter(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

	void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>& OutLifetimeProps) const override;

private:
	/** The main skeletal mesh associated with this Character (optional sub-object). */
	UPROPERTY(Category=Character, VisibleAnywhere, BlueprintReadOnly, meta=(AllowPrivateAccess = "true"))
	USkeletalMeshComponent* Mesh;

	/** Movement component used for movement logic in various movement modes (walking, falling, etc), containing rele
	UPROPERTY(Category=Character, VisibleAnywhere, BlueprintReadOnly, meta=(AllowPrivateAccess = "true"))
	UCharacterMovementComponent* CharacterMovement;

	/** The CapsuleComponent being used for movement collision (by CharacterMovement). Always treated as being vertic
	UPROPERTY(Category=Character, VisibleAnywhere, BlueprintReadOnly, meta=(AllowPrivateAccess = "true"))
	UCapsuleComponent* CapsuleComponent;
```

再同样的操作，我们可以看到APawn是一个多重继承的结构



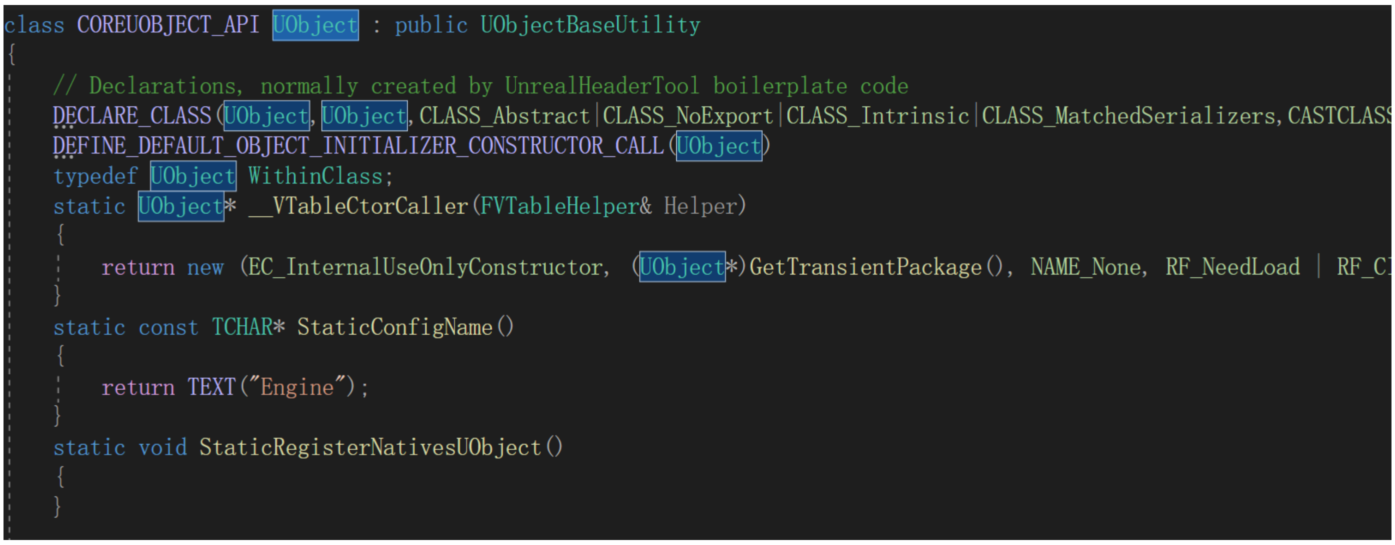
```
UCLASS(config=Game, BlueprintType, Blueprintable, hideCategories=(Navigation), meta=(ShortTooltip="A Pawn is an actor th
class ENGINE_API APawn : public AActor, public INavAgentInterface
{
	GENERATED_BODY()

public:
	/** Default UObject constructor. */
	APawn(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

	virtual void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>& OutLifetimeProps) const override;
	virtual void PreReplication( IRepChangedPropertyTracker & ChangedPropertyTracker ) override;

	/** Return our PawnMovementComponent, if we have one. By default, returns the first PawnMovementComponent found. Nat
	UFUNCTION(BlueprintCallable, meta=(Tooltip="Return our PawnMovementComponent, if we have one."), Category=Pawn)
	virtual UPawnMovementComponent* GetMovementComponent() const;

	/** Return PrimitiveComponent we are based on (standing on, attached to, and moving on). */
	virtual UPrimitiveComponent* GetMovementBase() const { return nullptr; }

	/** If true, this Pawn's pitch will be updated to match the Controller's ControlRotation pitch, if controlled by a F
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category=Pawn)
```

我们再向上查找，AActor，这里还是没有到头，但是跟我们上节课学习的结构完全一致，现在可以看到其继承自UObject



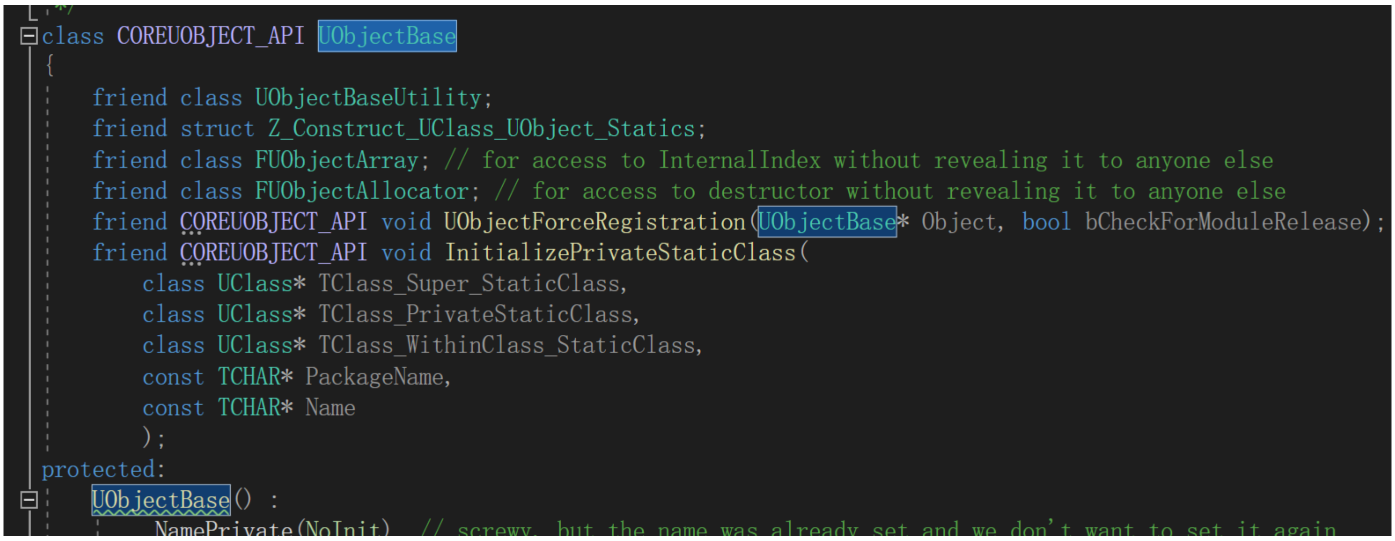
```
class ENGINE_API AActor : public UObject
{
    GENERATED_BODY()

public:
    /** Default constructor for AActor */
    AActor();

    /** Constructor for AActor that takes an ObjectInitializer for backward compatibility */
    AActor(const FObjectInitializer& ObjectInitializer);

private:
    /** Called from the constructor to initialize the class to its default settings */
    void InitializeDefaults();

public:
    /** Returns the properties used for network replication, this needs to be overridden by all actor classes with na
    void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>& OutLifetimeProps) const override;

    /**
     * Primary Actor tick function, which calls TickActor().
     * Tick functions can be configured to control whether ticking is enabled, at what time during a frame the update
```

这里按照我们上节课的学习，这里应该是最上层的类了，但是我们此时再进行点击



```
class COREUOBJECT_API UObject : public UObjectBaseUtility
{
    // Declarations, normally created by UnrealHeaderTool boilerplate code
    DECLARE_CLASS(UObject,UObject,CLASS_Abstract|CLASS_NoExport|CLASS_Intrinsic|CLASS_MatchedSerializers,CASTCLASS
    DEFINE_DEFAULT_OBJECT_INITIALIZER_CONSTRUCTOR_CALL(UObject)
    typedef UObject WithinClass;
    static UObject* __VTableCtorCaller(FVTableHelper& Helper)
    {
        return new (EC_InternalUseOnlyConstructor, (UObject*)GetTransientPackage(), NAME_None, RF_NeedLoad | RF_C
    }
    static const TCHAR* StaticConfigName()
    {
        return TEXT("Engine");
    }
    static void StaticRegisterNativesUObject()
    {
    }
```

可以看到其上层还有类，然后我们最终发现最上层的类为UObjectBase



```
 */
class COREUOBJECT_API UObjectBase
{
    friend class UObjectBaseUtility;
    friend struct Z_Construct_UClass_UObject_Statics;
    friend class FUObjectArray; // for access to InternalIndex without revealing it to anyone else
    friend class FUObjectAllocator; // for access to destructor without revealing it to anyone else
    friend COREUOBJECT_API void UObjectForceRegistration(UObjectBase* Object, bool bCheckForModuleRelease);
    friend COREUOBJECT_API void InitializePrivateStaticClass(
        class UClass* TClass_Super_StaticClass,
        class UClass* TClass_PrivateStaticClass,
        class UClass* TClass_WithinClass_StaticClass,
        const TCHAR* PackageName,
        const TCHAR* Name
        );
protected:
    UObjectBase() :
        NamePrivate(NoInit)  // screwy, but the name was already set and we don't want to set it again
```

这样我们可以发现一个复杂的类的继承关系。

之后我们回到.cpp文件，这里的代码如下

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


#include "Maincharacter.h"

// Sets default values
AMaincharacter::AMaincharacter()
{
 	// Set this character to call Tick() every frame.  You can turn this off to
improve performance if you don't need it.
	PrimaryActorTick.bCanEverTick = true;

}

// Called when the game starts or when spawned
void AMaincharacter::BeginPlay()
{
	Super::BeginPlay();

}

// Called every frame
void AMaincharacter::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

}

// Called to bind functionality to input
void AMaincharacter::SetupPlayerInputComponent(UInputComponent*
PlayerInputComponent)
{
	Super::SetupPlayerInputComponent(PlayerInputComponent);

}
```

这里全部都是使用Super进行调用，也就是调用了其父类的同名函数。

三、总结

在这个看UE源码的过程中，我们逐步理解了UE4中的类的继承结构，并且明白了创建一个新的C++类时我们创建的类的结构，当然我们创建不同类的时候会有不同的结构，但是我们掌握了分析的方法，后面的内容也就大同小异了。