UCLASS(Abstract)

*类标识符Abstract 说明符会将类声明为"抽象基类"，阻止用户向关卡中添加此类的Actor。对于单独存在时没有意义的类，此说明符非常有用*

增强输入：需启用增强输入模块，引入增强输入头文件

内容大纲：

1. 创建角色类和怪物类
2. 设置角色怪物动画
3. 添加增强输入，绑定增强输入
4. 添加gameplay中的playercontroller和gamemode
5. 添加怪物高亮接口
6. **开始使用GAS，添加GameplayAbilities插件，创建GameplayAbilityComponent和AttributeSet类：**
7. 给需要使用的类添加可存放AbilityComponent和AttributeSet的指针
8. 在所需类上创建UAuraAbilityComponent和AttributeSet的子组件，并在PlayerState也同样创建该组件，PlayerState数据存于服务器中，在playerstate创建可以复制到客户端上
9. 设置ReplicationMode

ReplicationMode类别：

Full：游戏效果被复制到所有客户端，适用于单人游戏

Mixed(玩家当前使用)：游戏效果只复制到自己的客户端，玩法提示和玩法标签复制到所有客户端，适用于多人游戏。（在该模式下，OwnerActor必须为controller，在Possession方法中设置PlayerState会被自动Controller，如果OwnerActor不是PlayerState必须在OwnerActor上调用SetOwner()）

Minimal(怪物当前使用)：玩法无法复制。玩法提示和玩法标签复制到所有客户端，适用于多人，AI方

1. 添加ASC：

一般只放在Character or PlayerState上，在武器上加ASC组件也不是不行，但是并没有很好的实践供参考，官方文档提到过这一点。

OwnerActor和AvartarActor是比较常见的概念，如果ASC在Character类身上，那么二者是相同的。

如果Character需要销毁再重新生成，如MOBA游戏角色死亡后泉水复活，那么ASC可以放在PlayerState上避免随着角色一同销毁。此时的OwnerActor是PlayerState，AvatarActor则是Character。

For the AuraEnemy class:

将InitAbilityActorInfo写在BeginPlay

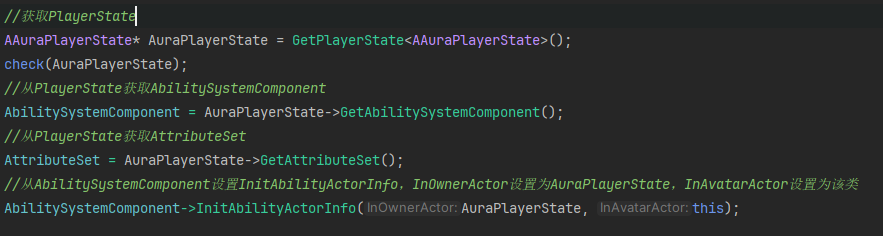
For AuraCharacter:

获取playerstate上的asc，设置AuraCharacter的Asc和AS的指针

**PossessedBy():被附身时回调**

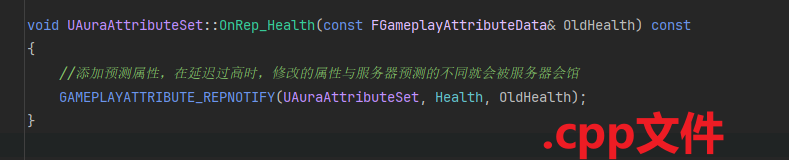
**OnRep\_PlayerState():playstate被复制时回调**

1. 在AuraCharacter和AuraEnemy调用InitAbilityActorInfo，初始化Actor信息：

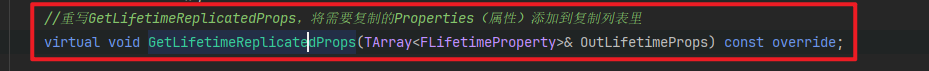


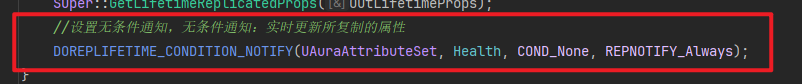
1. **Attribut，属性集**
2. 在AuraAttributeSet里添加属性（血量）
3. 添加预测属性及方法格式如下：



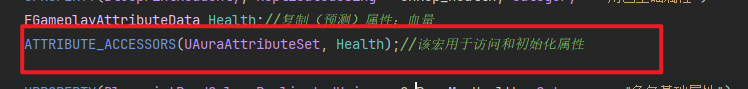


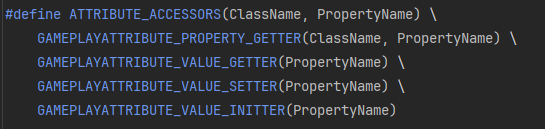
1. 将添加的复制（预测）属性添加到复制列表里，在重写的GetLifetimeReplicatedProps方法里设置通知条件：（.h,.cpp）



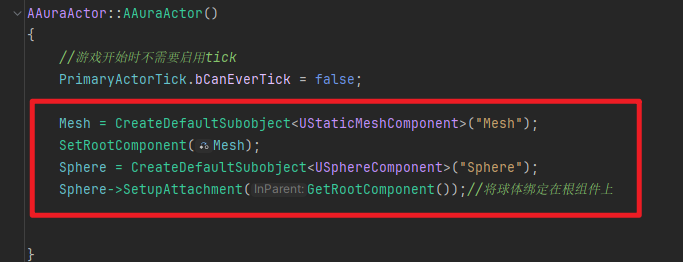


1. 调用GAMEPLAYATTRIBUTE\_PROPERTY\_GETTER宏来初始化和获取复制属性：



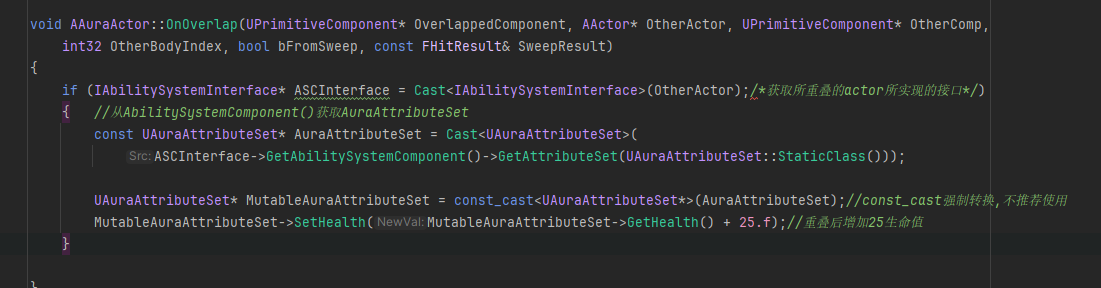


1. 新建AuraActor，该Actor作为最基础的Actor，上面添加StaticMesh，Sphere，为sphere绑定重叠事件



(6)通过重叠事件获取所碰撞到的Character，获取所实现的IAbilitySystemInterface接口进而获取Character所有的AbilitySystemComponent

，再通过AbilitySystemComponent获取AttributeSet, 通过所获取到的AttributeSet设置生命值



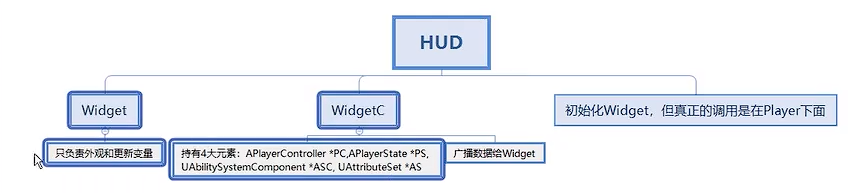
1. **游戏UI**

**步骤：**

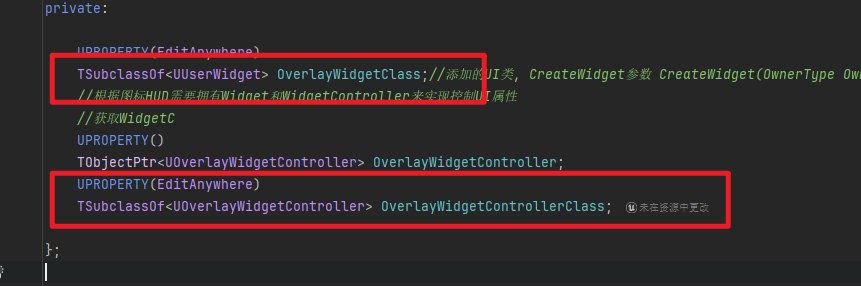
1. 创建UAuraUserWidget，UAuraWidgetController，AAuraHUD

Widget类显示UI效果，Widgetcontroller类控制UI属性数值，HUD绘制UI

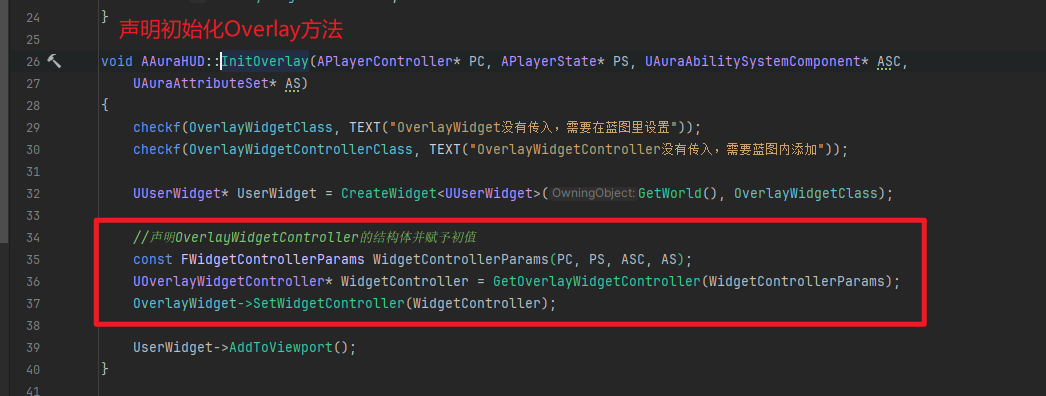
结构如下图：



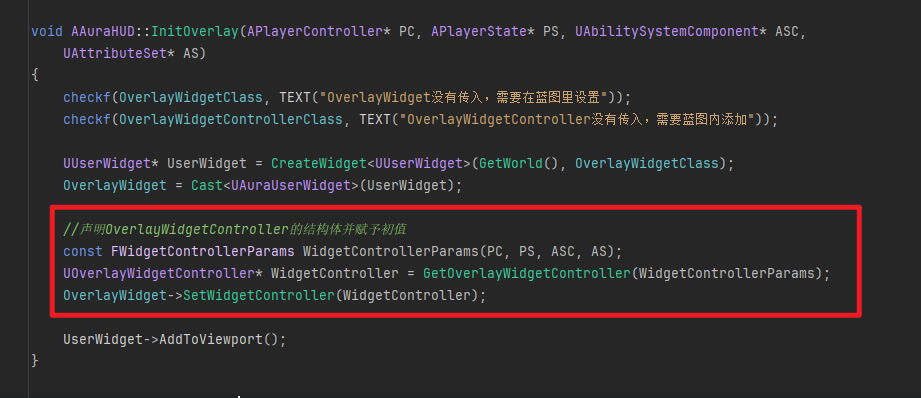
在HUD中获取Widget和WidgetController



将WidgetController所拥有的四大元素初始化

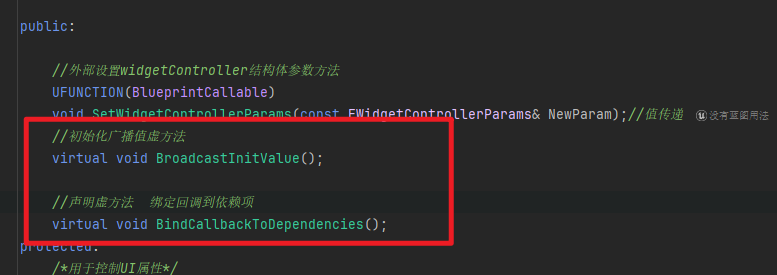


在HUD中完成初始化方法后，在Player（AuraCharacter）中调用初始化函数InitOverlay



1. 同步设置广播UI值

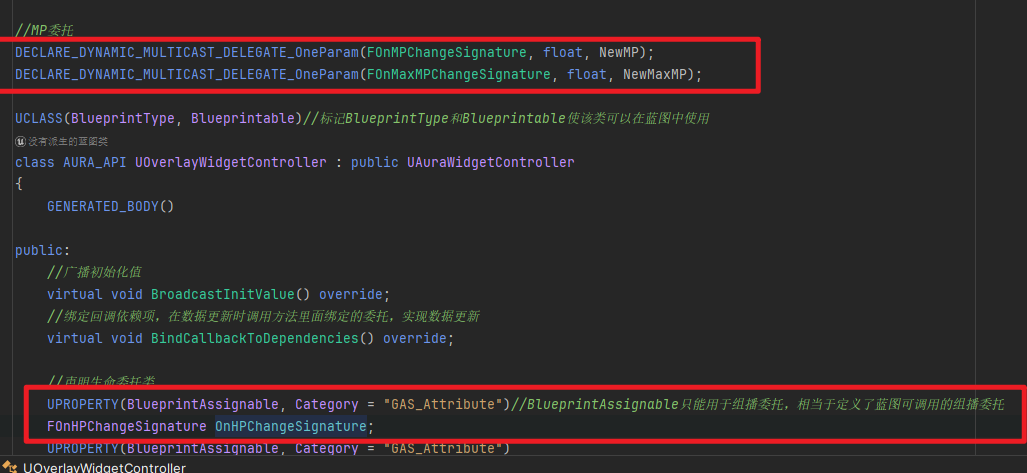
在Widget基类声明方法BroadcastInitValue()，BindCallbackToDependencies(),这俩方法需要在子类重新实现



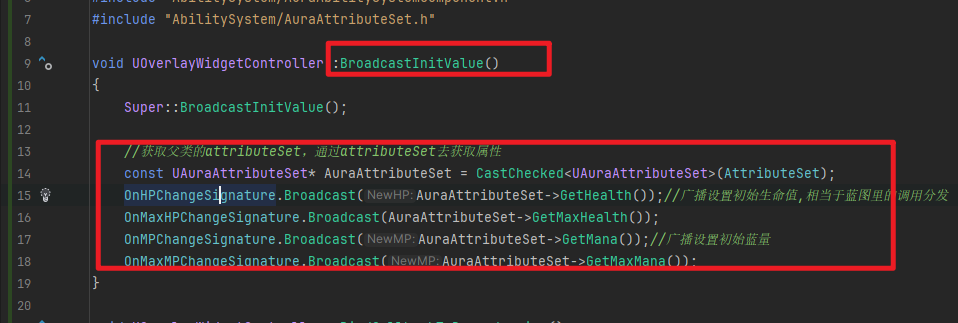
BroadcastInitValue()用于广播初始化参数

BindCallbackToDependencies()用于绑定属性回调委托方法

在overlayWidgetController声明广播



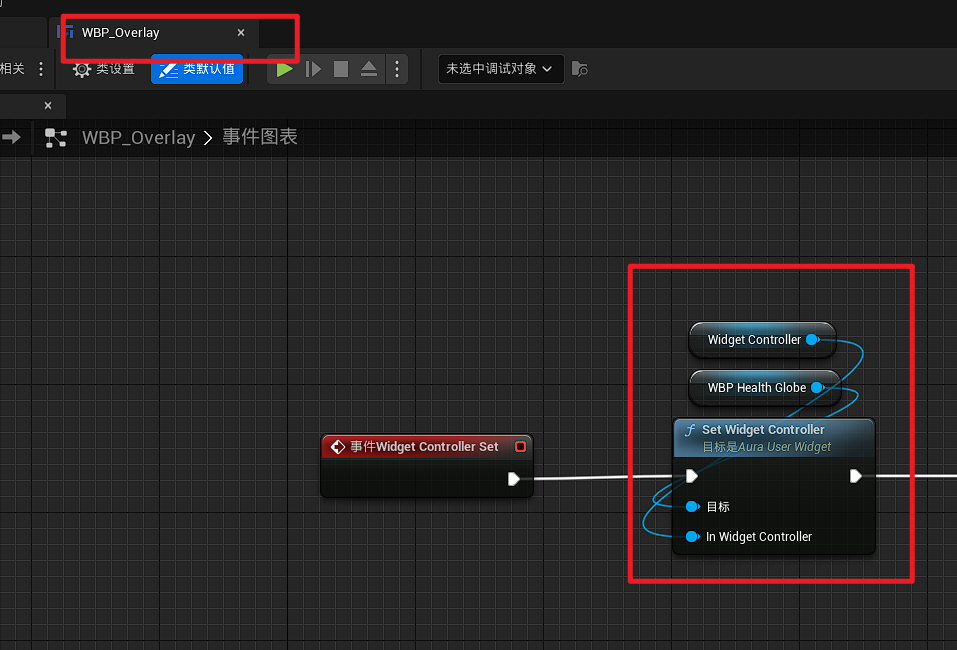
在overlayWidgetController实现的BroadcastInitValue()添加更新的属性值



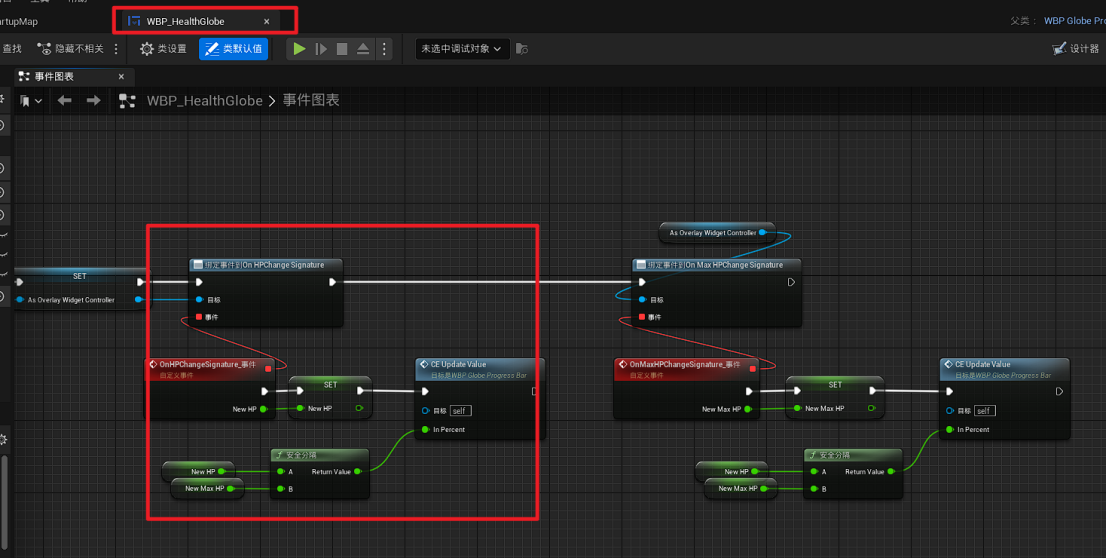
在HUD.cpp中调用初始化方法BroadcastInitValue()



在蓝图中设置生命值UI的UIController

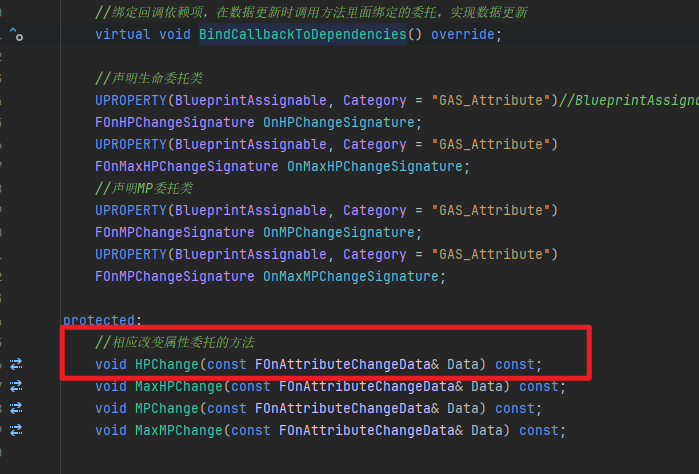


绑定在C++内部声明的委托，完成ui初始化属性值操作



1. 同步更新UI属性值

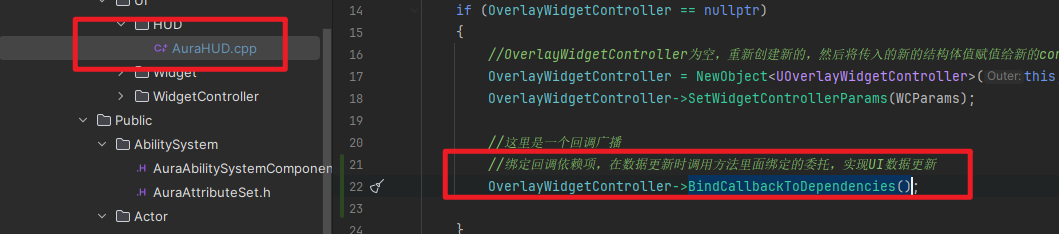
实现overlayWidgetController基类的方法，并声明委托所调用的方法





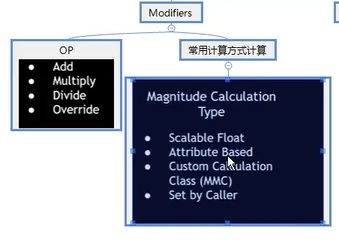
GetGameplayAttributeValueChangeDelegate是AbilitySystemComponent内置的方法，其作用是更新Attribute内部属性的时候会调用设置的参数委托

在HUD中调用设置好的BindCallbackToDependencies()



1. **GamePlayEffect**





1. Modifiers计算方式

Scalable Float：硬编码（基础伤害）

Attribute Based：基于属性的伤害

Custom Calculation Class(MMC)：自实现的伤害计算类

Set by Caller：通过标记调用修改

1. 技能作用时间分类：

Instant：一次性作用

Has Duration：持续作用

Infinite：被动作用（一致回蓝回血等）

1. 处理Tag

Stacking：可重叠的（血瓶）

Add Gameplay Tags：标记

Grant Abilities：获取该标签获取能力



1. 代码部分，删除了AuraEffectActor的StaticMesh和碰撞胶囊（这类使用蓝图能更方便实现）

创建Effect作用到目标的方法（提供蓝图调用）

void ApplyEffectToTarget(AActor\* Target, TSubclassOf<UGameplayEffect> GamePlayEffectClass)

