2024.9.17

项目准备开始。先看英雄设计，提前准备一些系统和机制。

命名规范：修改自虚幻项目命名规范

[AssetTypePrefix]\_[AssetName]\_[Descriptor]\_[OptionalVariantLetterOrNumber]

其中：

AssetTypePrefix 将表明资产的类型，详情请参阅下表。

AssetName 是资产的名称。

Descriptor 将提供资产的更多上下文，表明其用法。例如，纹理是正常贴图还是不透明度贴图。

OptionalVariantLetterOrNumber 是可选的，用于区分资产的多个版本或变体。通常使用时间和事件来描述

命名请使用中文

https://dev.epicgames.com/documentation/zh-cn/unreal-engine/recommended-asset-naming-conventions-in-unreal-engine-projects

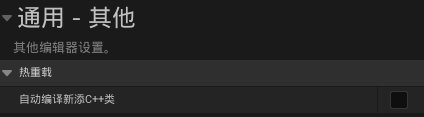
看英雄和装备的设计，在下面逐步增加机制设计

禁用实时编码

“编辑--编辑器偏好设置--通用--外观”



禁用自动编译新添C++类



资产编辑器打开路径设置为主窗口



接下来需要我们实现

角色死亡。

很简单，在角色总类中判断当前生命值是否小于等于0，为0，则播放死亡动画并DestroyActor

子类中为了继承该事件，不能删除ParentTick节点



创建死亡动画蒙太奇，之后创建动画蓝图，将角色的动画模式设置为使用动画蓝图。

若想播放死亡动画，一定会打断原有动画蓝图。所以要使用插槽

技能范围显性（最重要）、

全局游戏时停

特定单位时停

特定单位不时停

物品检索器（检索周围符合某类要求的物品的数量。比如检索英雄的数量）

从物品检索器引申出的物品影响器

地图天气机制。晴初、鬼等英雄。

英雄方向系统。比如破绽方向、长风等机制。

一切obj都继承自一个祖类，命名为“obj模板”。

思路修改：对于

伤害机制：

继承路径：模板--伤害。

计划每个东西都是蓝图类，伤害也是蓝图类。

物理伤害、魔法伤害、转化伤害、真实伤害、绝对伤害

攻击&技能

攻击有攻击间隔

技能有冷却

都有前摇后摇

将普通攻击设计成技能。

每个技能有如下属性：

前摇时间

后摇时间

冷却时间

施法范围

法力消耗

生命消耗

设计成Actor类。这样也能创建世界对象。

接下来

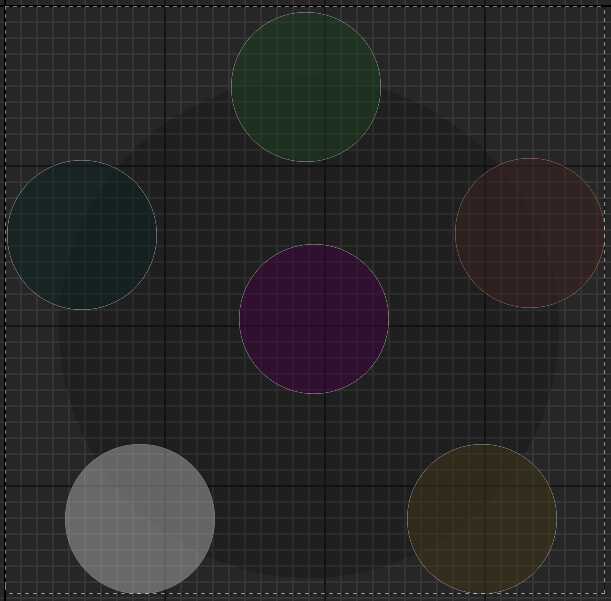
首先装备栏设计

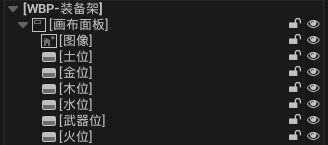
我们需要摁TAB键来查看和操作装备，同时在血条的右侧会有装备UI

UI需要构造一个圆形的UI

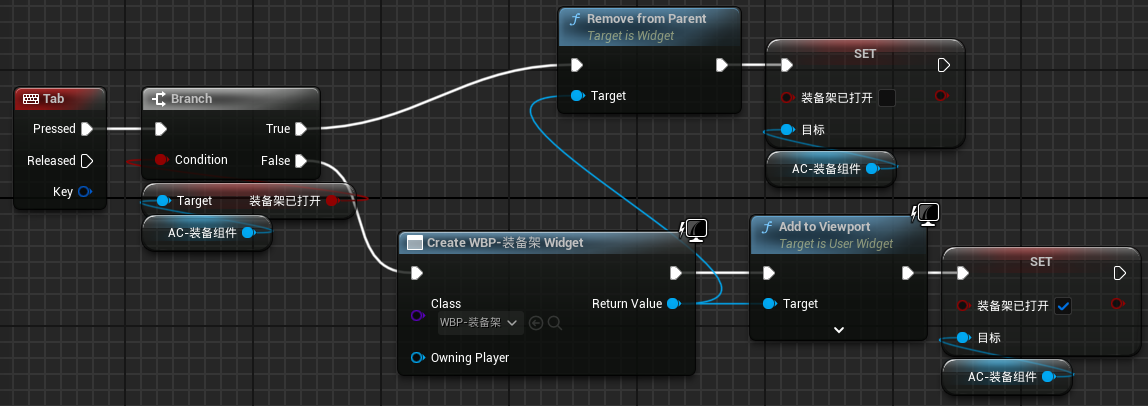
在WBP的按钮的细节中找到“外观--样式--普通--轮廓设置”设置好圆形盒体之后再设置圆角半径。

也可以通过图像的“外观--笔刷--绘制为”设置为圆形盒体

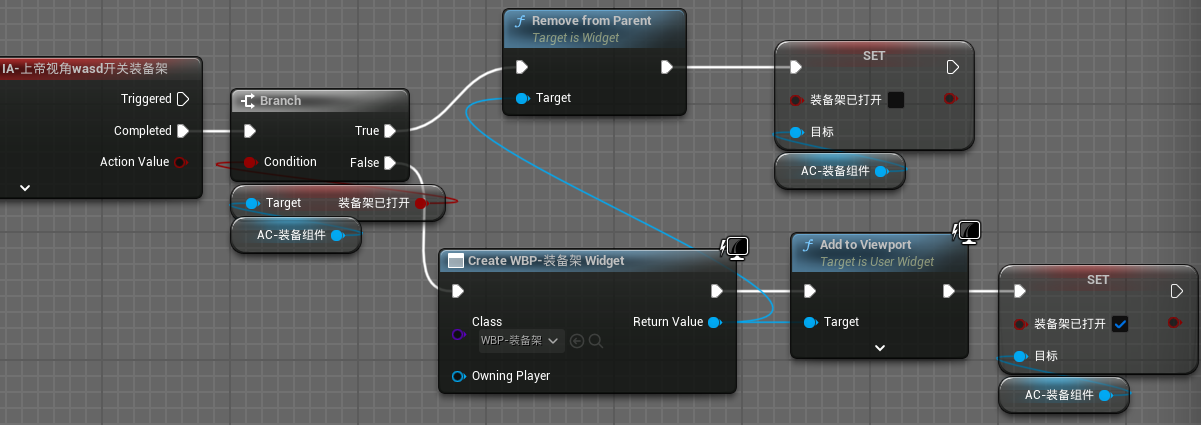




然后通过按键开关装备架



现在实现开关装备架的增强输入。因为增强输入时会持续输出，所以增强输入事件要用Completed



一般来说，装备栏只有在：接入装备、剔除装备、刷新层数（如果装备有层数，在BP装备中实现）、使用主动技能（目前没有涉及）时才会需要更新角色属性。而上述情况都是函数。所以我们需要创建一个变量，来表示应当刷新。这个变量在上述情况下进行更新。然后在“是否需要更新角色属性”里创建事件分发器，来表明需要更新角色属性。

换句话说，现在的情况下，我们在开着装备界面的时候是不能看到装备架刷新的，需要重新打开装备界面才能看到。

以函数名设置定时器

以句柄清除定时器并使之无效

## buff

直接写成组件，组件里面可以写函数，在角色里面调用就可以了。要几个都可以。Buff的数据还可以提升为变量，在角色里面自定义。

模块化好用好改。

就如，你在组件里面写了一个函数打印“哦耶”，装给角色装上，被攻击调用这个函数，就会打印哦耶，在组件里面提升为变量，你就可以在角色里面自定义打印的文本，不会影响组件。

首先你人物属性是一个总的结构体，然后结构体里其中一个变量也是结构体，是作为buff的

buff这个结构体包括buff类型，持续时间，发动效果

发动效果又是一个结构体

和buff有关的 都 继承一个 base抽象类

角色 与敌人 继承同一个 父类 这个父类 维护一个 buff抽象类的数组

Buff抽象类用UObject类

创建Object类蓝图，命名为“BP-buff模板”

创建buff的数据结构。创建结构，命名为“S-buff结构”

Buff的结构中应该有下列成员

ID--命名

名称--文本（文本类型适合用于翻译）

描述--文本

最大层数--int

持续时间--float（回合制游戏的持续时间为int）

图标--Texture2D

是否永久--bool

是否周期--bool（buff每几秒触发一次，如灼烧效果）

周期间隔时间--float

标签--Gameplay Tag Container（给buff加标签，属于什么样的buff）

时间规则--E-时间规则

（时间规则：添加buff时是刷新持续时间还是叠加持续时间）

（E-时间规则：叠加、刷新、保持）

层数规则--E-层数规则

（层数规则：多层buff中，当时间到达后，层数怎么变化）

（E-层数规则：清空、递减）

效果--BP-buff模板**类引用**



接下来进入BP-buff模板

Buff模板是实际buff运行时的一个基类。我们要在这里面实现一些功能。

首先实现所有buff的回调点

（回调点：例如周期触发）

创建函数，命名为“buff被回调”

创建函数，命名为“buff被创建”

创建函数，命名为“buff被移除”。添加形式参数，命名为“移除者”，类型为Actor对象引用。

创建函数，命名为“buff被改变层数”。添加形式参数，命名为“改变的层数”，类型为int。能实现一下改变多层buff的功能。

这只是简单声明了buff模板的功能。实际上，在不同的buff上会有不同的回调需求。比如“击杀时的buff”“被杀时的buff”“释放技能时buff”“命中角色时buff”“角色受伤时buff”等。这些可以在父类里实现，也可以在子类里实现或声明。

创建一些变量。这些变量都是buff运行期间的变量，或者说buff的状态

当前剩余时间--float

当前层数--int

当前周期剩余时间--float

创建--Actor对象引用

目标--Actor对象引用（buff的目标。一般来说都是buff的拥有者。如果是辅助有可能是别人）

存活之间--float（用于记录buff在目标身上存活了多长时间。有些技能会根据标记在目标上的持续时间造成额外伤害）



图片中有两个没提到的东西：buff配置和buff帧逻辑函数。我们在下面说明

创建变量Buff配置，变量类型为S-buff结构

然后，实现buff每帧处理的逻辑。创建函数，命名为“buff帧逻辑”，添加形式参数，命名为DeltaTime，类型为float。我们接下来用这个函数实现变量的逻辑实现。

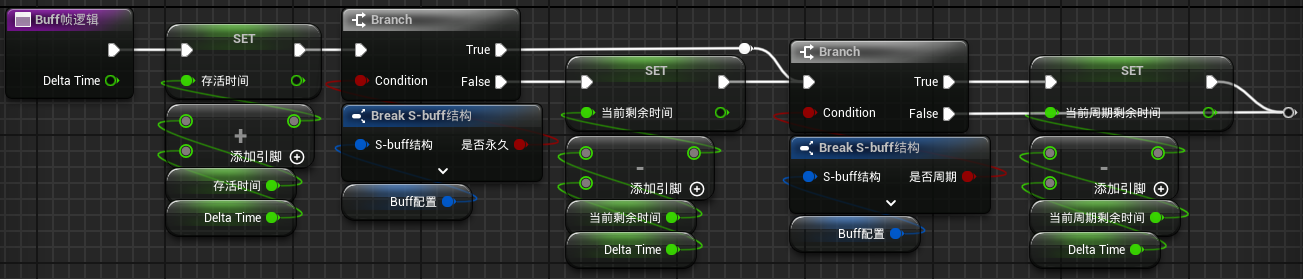
给函数一个返回值，命名为“时间结束”，类型为bool

首先是存活时间的设置。存活时间=存活时间+DeltaTime。

当前剩余时间=当前剩余时间-DeltaTime。在设置之前还要判断buff配置的“是否永久”是不是永久的，如果是永久的就没必要去处理。

当前周期剩余时间=当前周期剩余时间-DeltaTime：判断是不是周期，是的话才进行相关处理。

这样buff时间相关的就处理好了



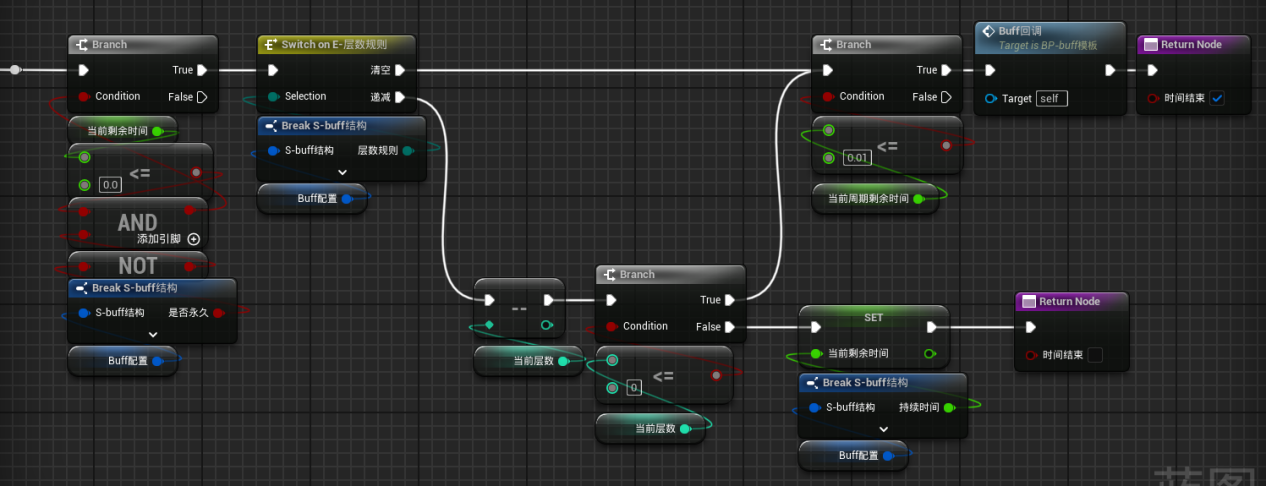
之后就是buff后续逻辑的处理。就是判断是不是时间已经结束了，还有周期是不是能够被触发。

判断是不是时间已经结束了：当前剩余时间<=0且buff配置的是否永久为false

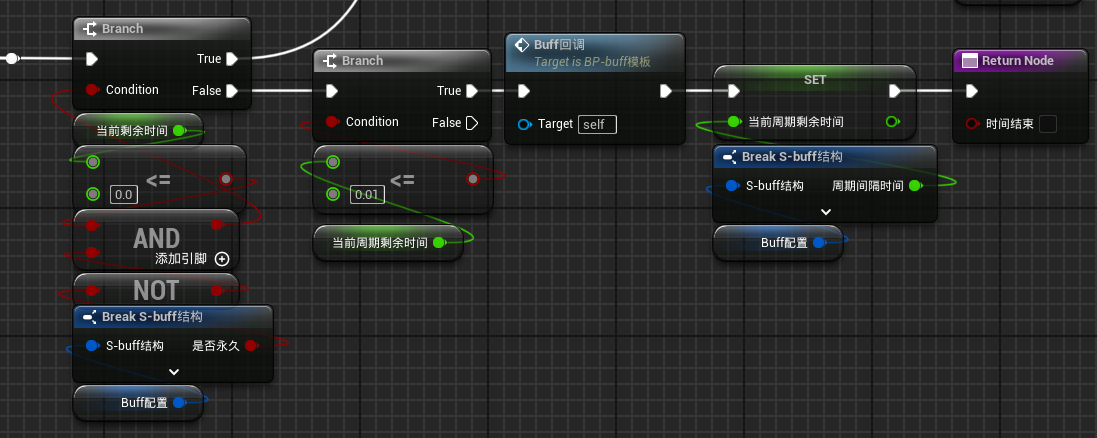
周期是不是能够被触发：当前周期剩余时间<=0。这个0是有误差的。一般给0.01.要自己去试。如果为真，则调用函数“buff被回调”

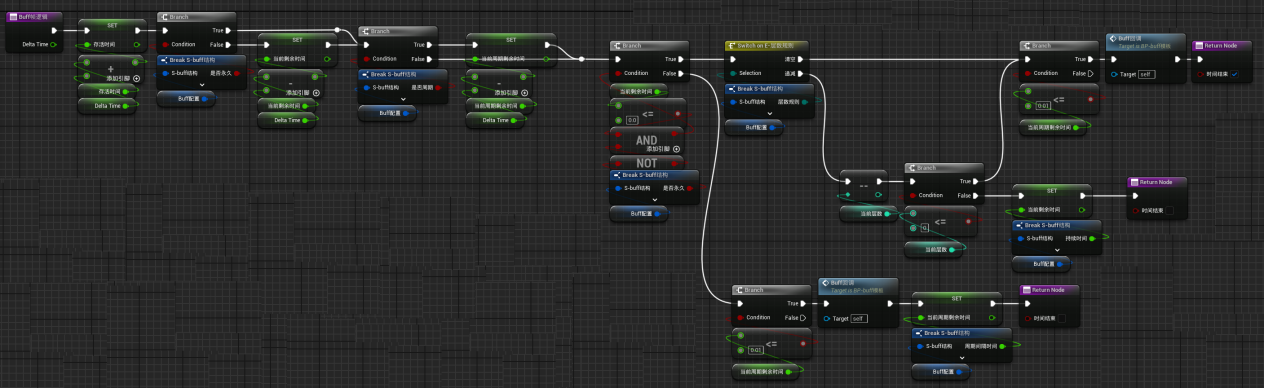
还有判断层数规则。描述起来较麻烦，根据判断逻辑自行实现。记得要刷新持续时间。

上述结束之后，给buff帧逻辑的返回值时间结束赋值为True



这只是buff结束的逻辑。还要去实现buff没结束的逻辑。





接下来创建一个具体的buff

我们可以创建一个打印buff。继承自buff模板，命名为“打印buff”。重写父类的4个函数。

要注意的是，移除buff的函数有一个参数移除者，我们要判断它IsValid。

具体打印什么随意。重点是接下来的：怎么由角色创建buff对象

创建buff的配置表。创建一个数据表格，行结构为S-buff配置，命名为DT-buff配置。

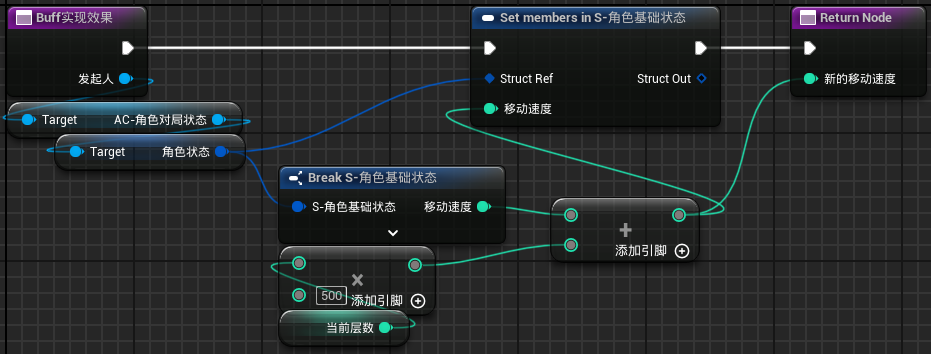
我们在buff配置表中新建一个buff，要注意的是要将它的“效果”变量赋值为“BP-buff模板”。其他的自行设置。

Buff管理器组件

ActorComponent组件

里面有两个函数：添加buff和移除buff

首先实现加移动速度。继承自buff-buff模板创建子类命名为buff-加移动速度。打开蓝图。创建函数命名为buff实现效果。添加一个形式参数，一个返回值。



然后实现移速buff的使用。每次使用就增加一层层数。

首先创建ActorComponent来管理角色的buff。命名为AC-角色buff

创建两个函数：添加buff、删除buff

创建变量：

当前正在执行的buff--buff模板的对象引用的数组

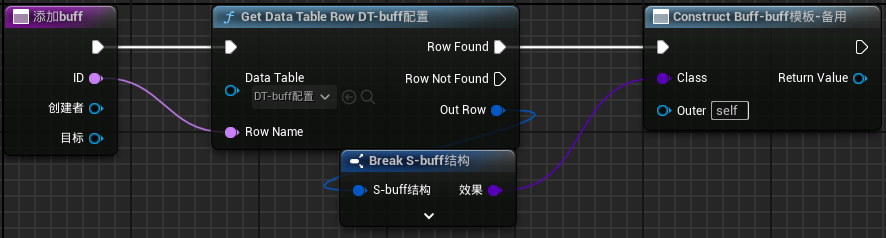
当前帧正在添加的buff--

等待删除的buff

先来实现添加buff的逻辑。添加三个参数：ID、创建者、目标。

第一步肯定是先判断当前添加的buff是否已经存在，以及根据ID判断当前buff是否合法（是不是在DT里面）。然后我们把他们添加到“当前帧正在添加的buff”

用到的节点：Construct Object from Class、Get data table row

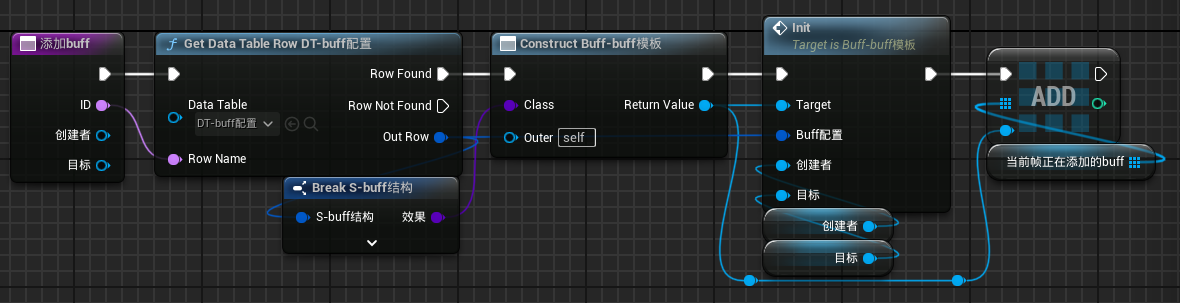


然后去写buff的初始化函数

进入buff-buff模板，创建函数，命名为Init。通过buff配置中的东西对其他变量进行初始化。通过函数提供参数来初始化buff配置。



然后回到AC，在Construct buff-buff模板之后，对Construct的buff对象进行初始化，并将Construct创建的buff对象添加到“当前帧正在添加的buff”

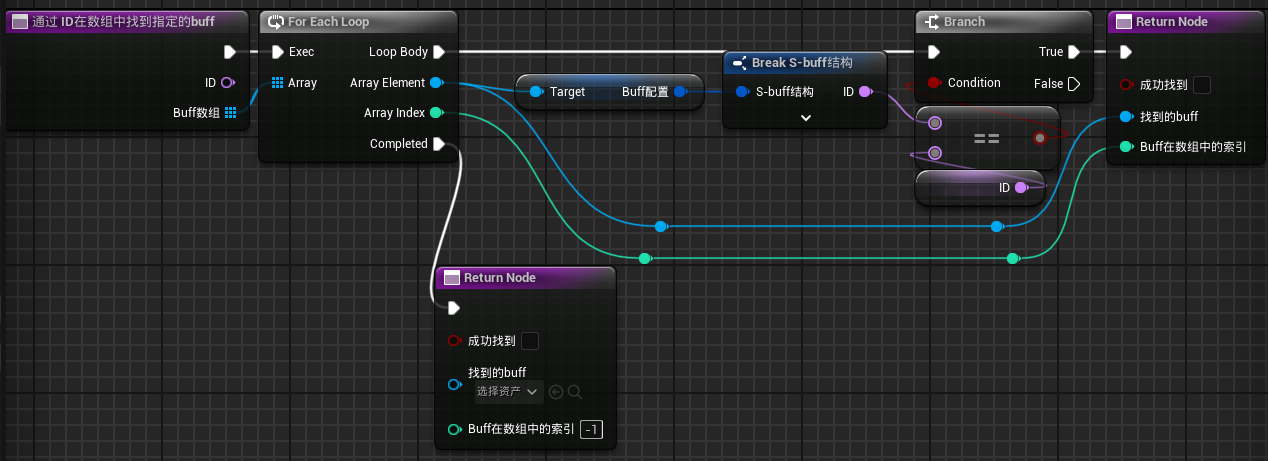


然后实现删除buff函数

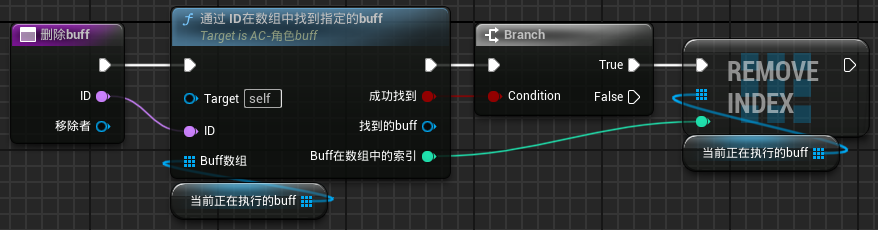
在实现删除操作时，需要先去判断在正在执行的buff数组中是否有我们要删除的这个buff。

创建函数，命名为“通过ID在数组中找到指定的buff”

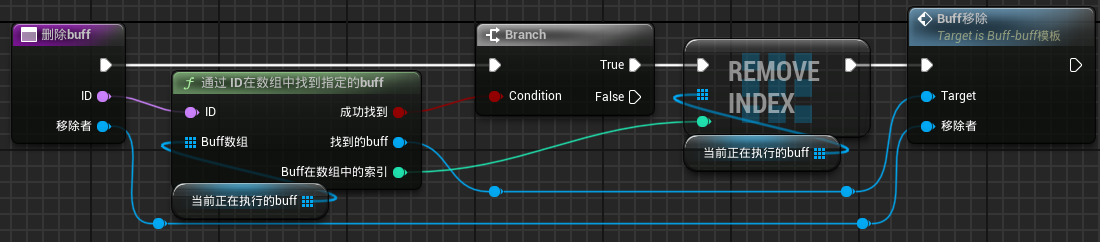
输入值为ID和buff数组，输出值为“成功找到”（bool）、找到的buff、buff在数组中的索引。



然后使用这个函数实现删除buff函数



删除之后还要触发一下被删除的回调点。



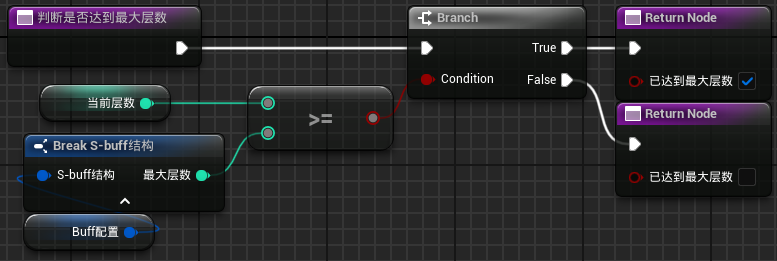
接下来要做整个AC的buff的Tick处理。创建函数命名为“buffs帧处理”。具有一个形参DeltaTime。

首先判断“当前正在添加的buff”在“正在执行的buff”中是否存在，如果不存在，就把这个正在添加的buff添加到正在执行的buff，并且回调。如果存在，就判断是否已经达到最大层数，达到了则刷新持续时间，没达到则加一层并刷新持续时间。

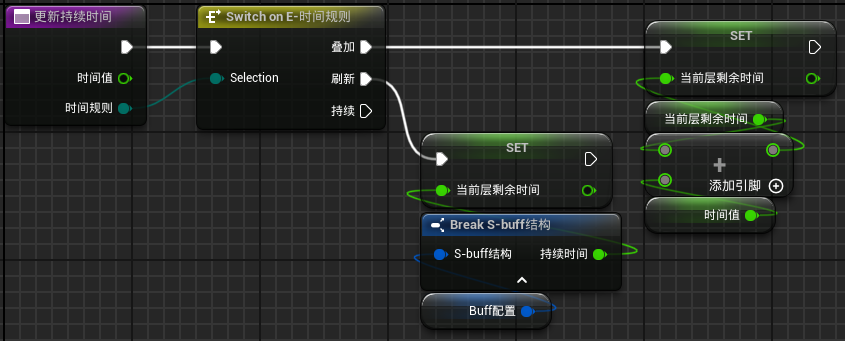
实际上并不一定是刷新持续时间，还要看时间规则的赋值。

根据上述流程，实现一些前置函数

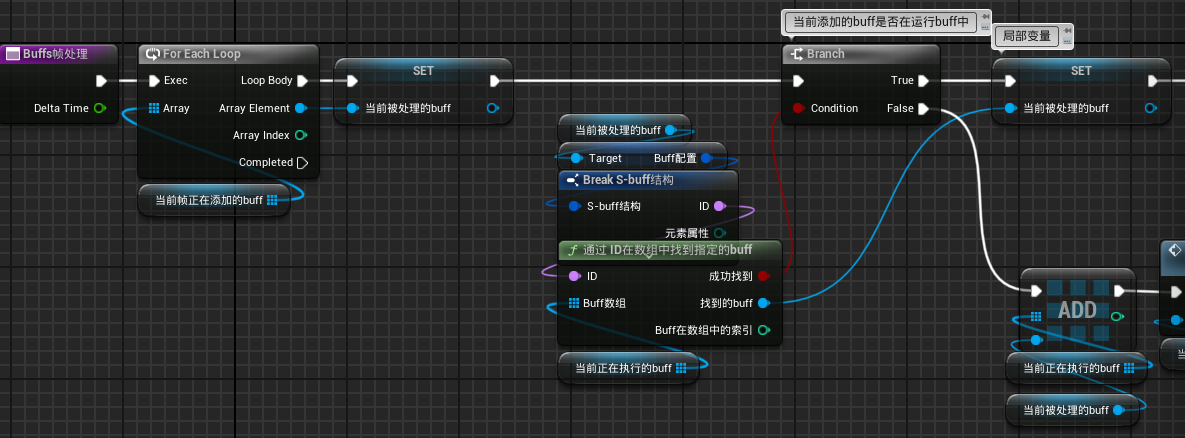
在buff模板中实现函数“判断是否达到最大层数”

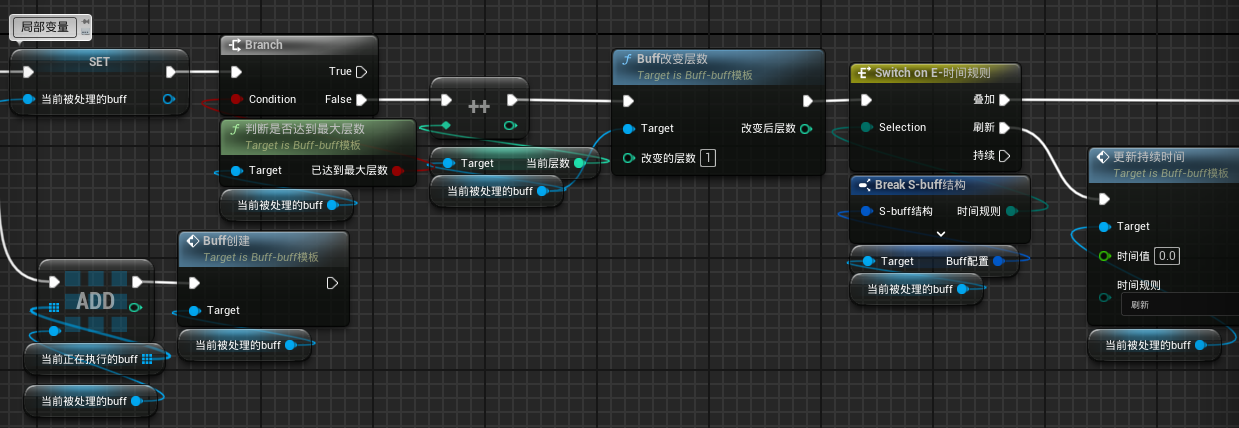


实现函数“刷新持续时间”和“更新持续时间”。因为更新持续时间可以实现刷新持续时间，所以用一个函数来实现



AC中buffs帧处理

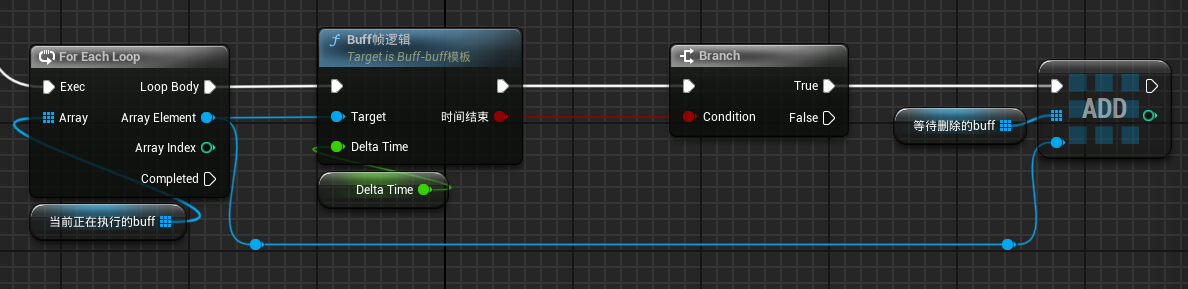




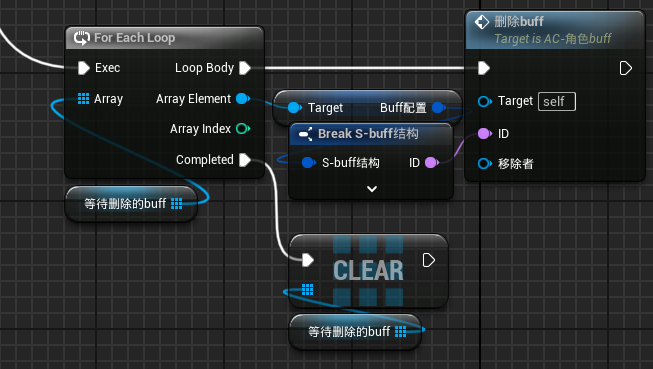


这是buffsTick的一部分。我们用Sequence去执行另一部分。

遍历当前执行的buff，并执行当前buff自己的Tick。之后判断是否已经结束了，如果结束了就删除。

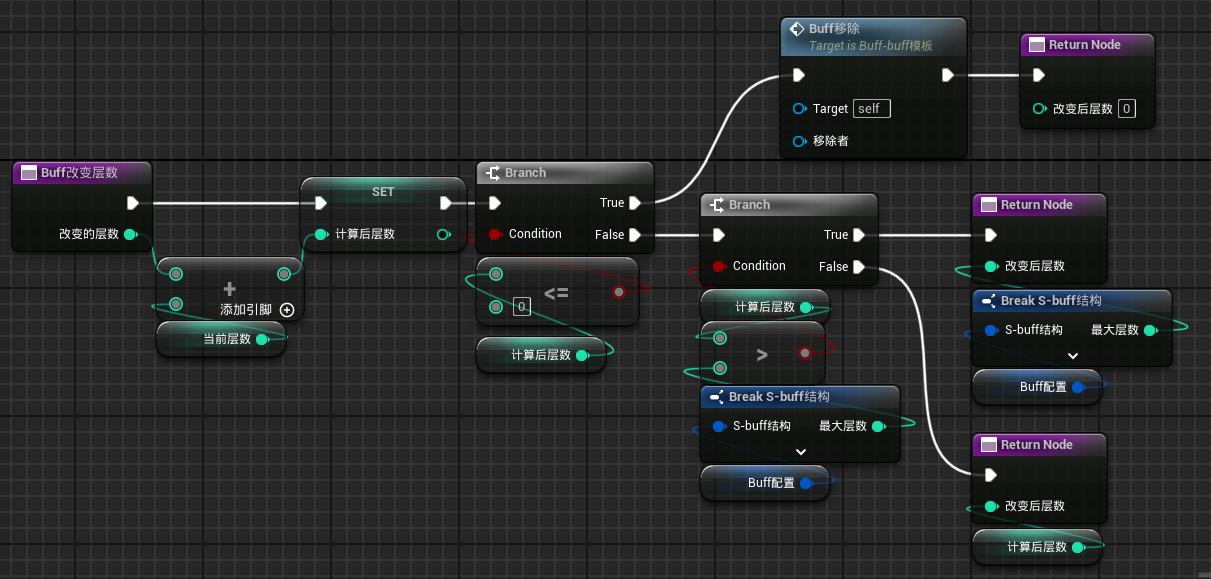


BuffTick第三个要处理的，遍历等待删除的buff数组，并删除。然后清空除了执行的buff以外的数组空间。



在buff模板中简单实现一下函数

Buff改变层数

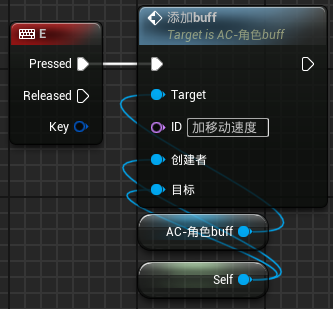


（buff移除函数是buff模板中自定义的另一个函数）

写好之后去AC的事件图表中用EventTick调用Buffs帧处理函数



在角色模板中添加组件，调用组件的添加buff函数

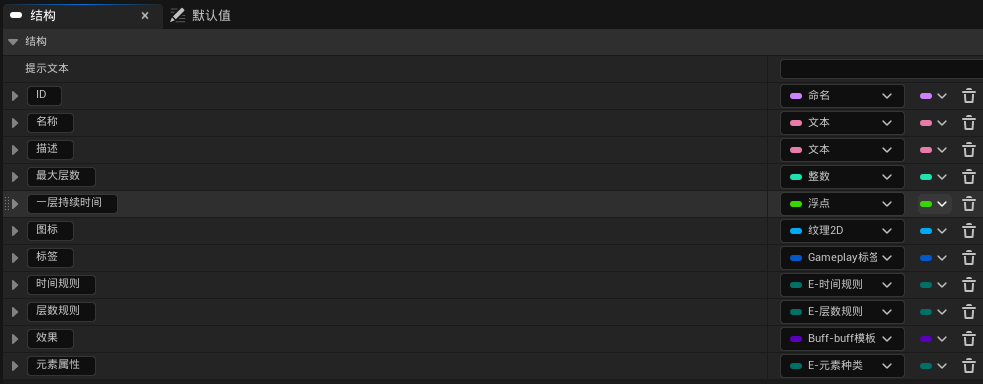


Buff结构

创建一个结构来表示buff的属性，命名为“S-buff结构”。

这里我们不考虑buff持续时间内周期触发和永久buff。

Buff应当有如下属性



时间规则有三种：叠加刷新持续

层数规则有两种：递减清零

创建数据表，命名为DT-buff配置。

因为后续用到buff图标甚至于用到许多图标，所以从epic中添加资源Craft Resources Icons。选择几个图标。打开图标后，将纹理组设置为UI。

这里我们选择两个图标，分别表示加攻击力和加移动速度，并如此命名。创建两个文件夹，如此命名。将相应的图标放到对应文件夹中。

在DT-buff配置中添加这两个buff。



进行如下的一些设置，以区分他们的使用。

加移动速度的最大层数为10，加攻击力的最大层数为1。

BuffUI

绘制一个正方形的buff框，导入到ue中

打开导入后的纹理，将“层次细节--纹理组”的内容赋值为UI

创建控件蓝图，命名为“buff控件”

添加一个画布面板

添加一个尺寸框，来限定buff的大小，对高度和宽度进行重载，大小设置为46

把画布面板设置为所需。然后再调节尺寸框的基础值和重载值一样

然后添加一个覆层，方便让组件重叠组合。这个覆层主要放置技能框和buff覆层。

还需要添加一个尺寸框来限制buff图标的大小。尺寸框水平垂直居中对齐，大小比上一个略小，设置为41

在尺寸框里面再添加一个覆层。因为这个存放buff图标，所以水平垂直对齐选择全覆盖。

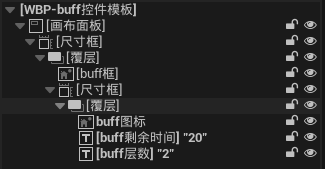
框架已经设置好，接下来就是在里面放置需要的东西。

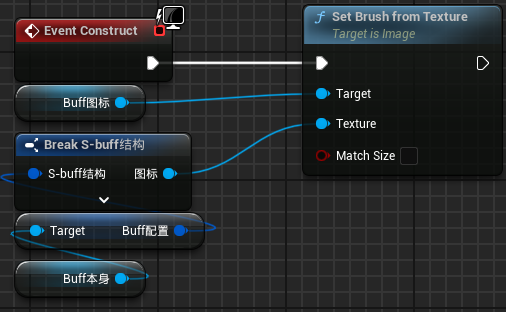
放置一个图像，勾选是变量，命名为buff图标。对齐选择全覆盖。

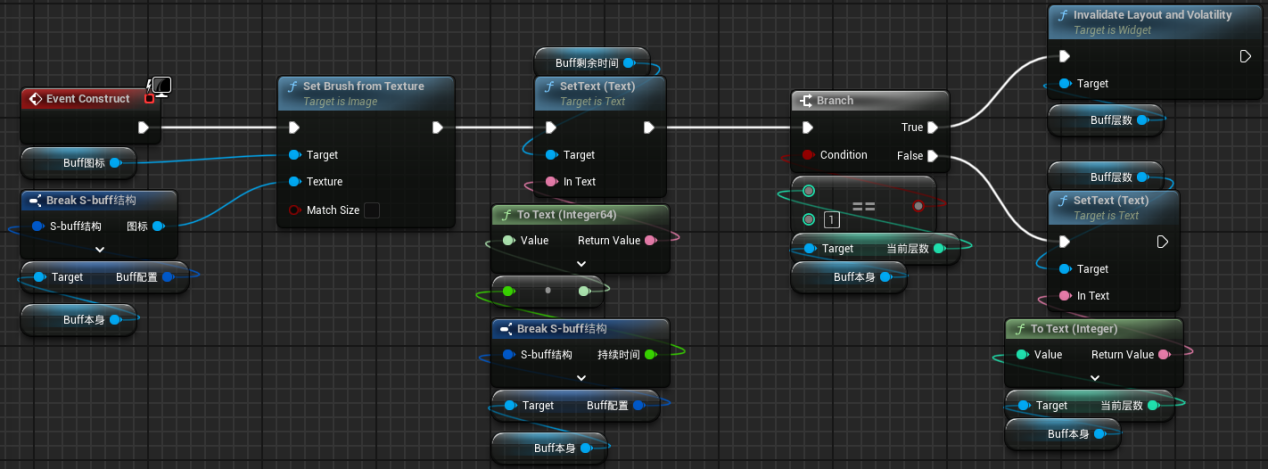
放置一个文本，勾选是变量，命名为buff剩余时间。对齐为居中。字体大小为20。“行为--可视性”设置为隐藏。

放置一个文本，勾选是变量，命名为buff层数。对齐为右下。字体大小为10。“行为--可视性”设置为隐藏。

最后添加一个图像，放在最外层的覆层中，用作buff框。







Buff本体

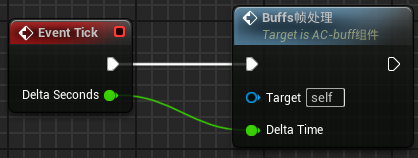
创建Object类对象，命名为buff-buff模板

AC-buff组件

后面我们先假设每个时间只有一个buff，进行实验，先不用数组

需要知道的是，buff只能使用EventTick进行更新。

创建AC-buff组件，创建函数命名为“buffs帧处理”。我们要在EventTick时对buff进行每帧处理。所以要给函数一个形式参数DeltaTime。



创建变量命名为“正在执行的buff”、待添加的buff、待删除的buff。

假设E键可以在角色自己身上加一个加移动速度的buff，Q键加一个加攻击力的buff。

当我们有一个buff要应用在角色身上时

AC在每帧应当进行下面的操作：Sequence

①判断待添加的buff是不是正在执行的buff。

如果不是，将待添加的buff添加到正在执行的buff中。

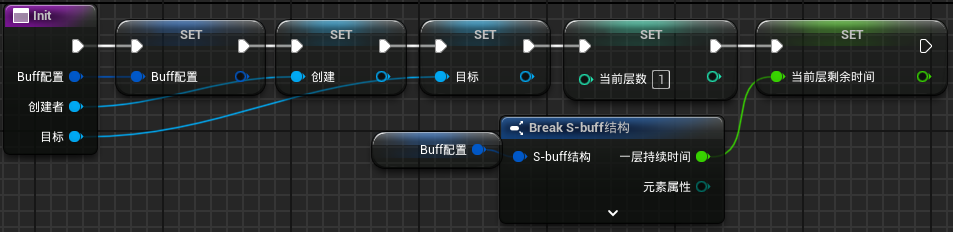
如果是，判断相加之后是否达到最大层。无论是否达到最大层，都应当按照层数规则更新buff的持续时间。

②遍历当前执行的buff，并执行当前buff自己的Tick。之后判断是否已经结束了，如果结束了就放到待删除的buff中。

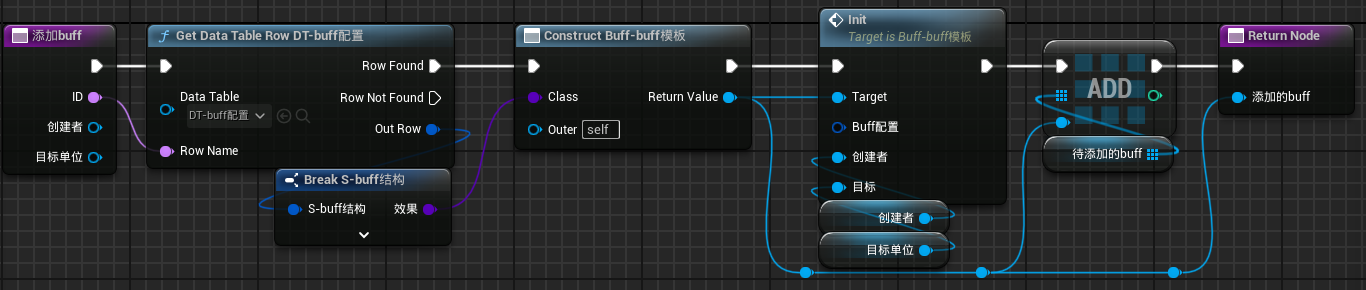
③删除待删除的buff中的内容。然后清空（Clear）除了执行的buff以外的数组空间。

Buffs帧处理函数只是更新buff相关数组的内容。当我们想主动使用某个buff时，应当使用一个创建buff的函数。在AC中我们命名为“添加buff”。函数的形式参数有：ID、创建者、目标单位。

在添加buff时，需要我们对buff进行初始化。在buff模板中实现Init函数。



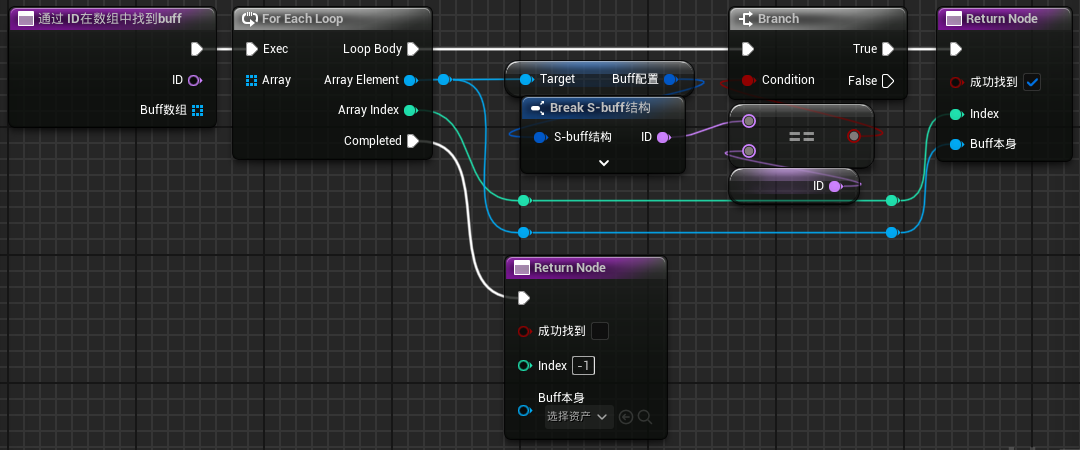
根据ID get data table row，在找到之后Construct Object from Class，Construct的是通过ID得到的那一行buff的效果（buff-buff模板的类引用）。然后对buff进行init，然后将Construct创建的对象添加到“待添加的buff”数组中。同时给函数一个返回值



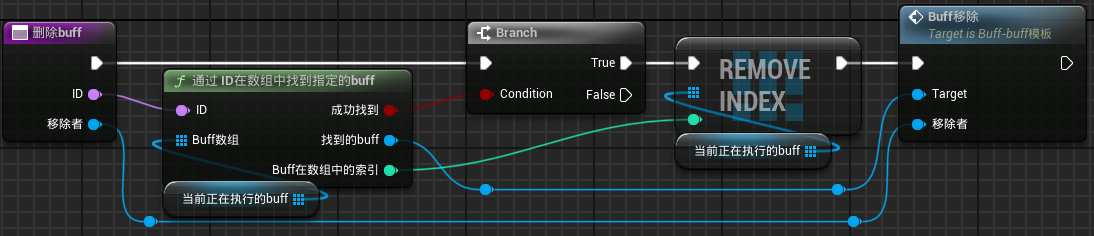
注意，添加buff函数会不断向“待添加的buff”数组中添加成员，需要在tick帧处理中对这个数组进行处理

在删除buff时，会将指定的buff从“待删除的buff”中Remove。指定的buff应当是在“正在执行的buff”中的成员。

所以我们要实现一个函数，在数组中找到指定的buff（声明为纯函数）



然后实现“删除buff”函数



10-22

对象池模式

当角色现需要一次释放许多火球，并在之后销毁他们时，如果真的生成并销毁，那对cpu的开销将是巨大的。

所以我们使用对象池模式，在游戏一开始的时候就创建这些对象，只是是否使用他们的时候对他们的“是正在使用”属性进行修改。这样可以极大地减少cpu的开销，不过会增大内存的开销。

现在我们对角色的