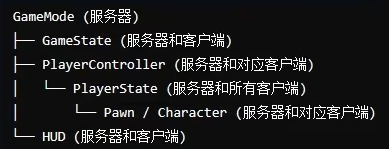
在游戏模式中，我们怎么知道别的玩家接入到游戏中呢？我作为服务器，我怎么知道别的玩家连接到服务器中呢？怎么才能得到玩家的数据呢？在游戏状态GameState中。

对于一个游戏实例，它有Gamemode、GameState、PlayerController、PlayerState、Pawn等。他们的生命周期和存在位置需要进行关注，以及他们能否复制。



GameInstance可以跨关卡存储数据。

列举出一场游戏的上述各个内容的层次关系。



一般来说，在创建GameMode等东西的时候，在新建蓝图类的时候，会有两种选项：GameModeBase类和GameMode类



两者的区别是：

GameMode是早期虚幻引擎还作为虚幻竞技场时就在使用的，直到UE4.14开始有GameModeBase类，用于大量的单机创作人。

GameState类是同理的。

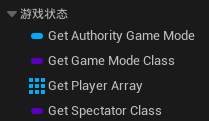
如果你前面创建的是Base类，那么后面的也要创建Base类的蓝图。

当我们想在GameMode 中获得玩家登陆进来的信息，一种方法是：通过“事件后台登陆时”，通过返回的PlayerController，创建一个数组，把这些PlayerController放到数组中，存储信息，来方便我们的后续使用。

实际上，这个思路是有误导性的。如果我创建了游戏模式和游戏状态，那么这里的所有东西都要我自己来实现。那我创建这个GM和GS的意义是什么？这属于脱裤子放屁。

解除误导，当我们创建完GameMode的时候，什么也不用做。那么作为服务器我们该怎么知道别的玩家连进来呢？在GameState中。

在GameState的事件图表空白处，右键找到“变量”，摁住Shift点开变量（展开全部），找到“游戏状态”，这些游戏状态，就是GameMode那边通过C++层面公开过来的。



这里的GetPlayerArray就已经涵盖了GameMode中接收到的玩家信息。

所以说自己创建数组是脱裤子放屁。

当然，我们也可以在PlayerState中获得分发下来的玩家数据。在PlayerState的变量--玩家状态中，有一个非常重要的数据：玩家ID



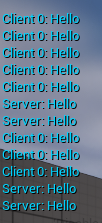
## 如何避免多人联机中的多重调用

在做多人联机的时候，遇到的第一个问题，就是重复触发的问题。

首先说一下重复触发的现象。

网络模式选择以监听服务器运行，玩家数量选择3。

在BP\_BaseCharacter的EventBeginPlay时，简单打印一个字符串。会发现有重复打印



按理说，只有三个PlayerStart，应该只输出3次Hello。但实际重复触发了很多次。

原因是这段代码没有做任何的保护。每当有一个客户端连入的时候，服务器也会跟着触发。

但是有个疑问：为什么EventBeginPlay默认链接的CastTo和后面那一坨，为什么他没有重复触发？原因就在于GameMode的默认Pawn类，当默认的类有的时候，后续生成的就不会覆盖掉。

那么有没有什么办法能避免这个问题？有。只需要加上“已本地控制IsLocallyControlled”节点



如果受本地控制器所控制，而不是网络控制器，则返回true

加上这个判断，就只生成一个了。

除此之外，如果是在玩家控制器里实现相同判断，使用的是“为本地玩家控制器IsLocalPlayerController”节点

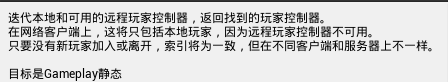


如果在PlayerController中不进行这个判断，服务器会重复触发，客户端不会重复触发。

现在看一些其他的函数

**GetPlayerController函数**





如果是服务器的话，作用非常大

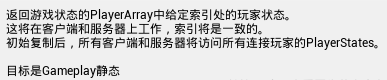
如果不是服务器，那么它将只包含本地玩家

如果这个函数再结合IsLocalPlayerController函数，就可以直接获取本地唯一的一个玩家控制器。

如果是在服务器上，就可以通过索引号来获取不同的玩家控制器。

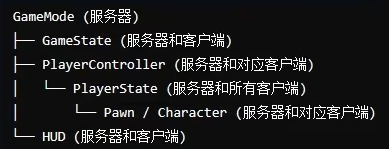
**GetPlayerState函数**





如果我们是客户端想获得其他的玩家状态，但我们不知道索引，怎么办？

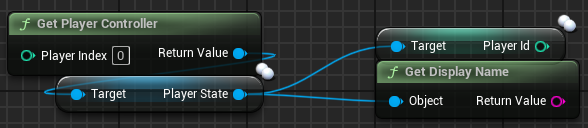
回顾层级关系



PlayerState是分配给PlayerController的。也就是说，PlayerController中有PlayerState这个变量“变量--控制器--GetPlayerState”我们可以通过GetPlayerController的返回值Get PlayerState变量，就绕过了GetPlayerState函数



这样，我们就可以通过PlayerState变量来获得相关信息，比如Player ID、显示命名等



我们可以打印一下这两个东西





此时会有报错



所以这里我们获得到了服务器的ID是345（不是定值，是根据上图打印结果得出的），而客户端获取到的是0。

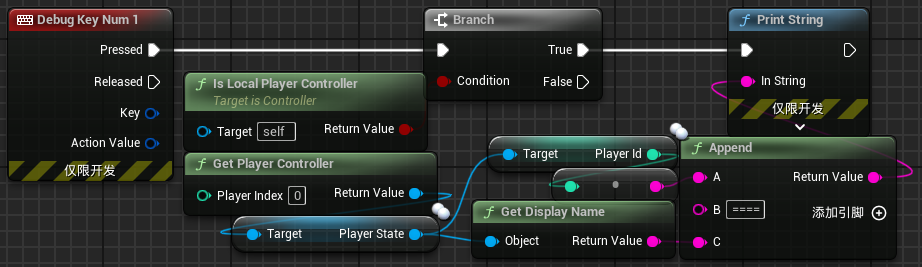
原因就在于这个EventBeginPlay的执行过程非常快，数据还没同步下来呢哈哈哈哈哈。所以加个延迟就行。

注意，多人游戏中加延迟，最好不要用Delay节点，会有意想不到的异步错误。需要延迟就用SetTimer节点。





在PlayerController中，也可以通过按键来调用这些功能



这样就只会在对应的游戏屏幕下进行打印（当然打印结果是都能看到的，只是在某一个窗口下只会打印行为一次）

**GetLocalRole函数**（获取本地角色）



因为前面介绍了“IsLocallyControlled”和“IsLocalPlayerController”，所以GetLocalRole函数就不是很重要了。他只需要在我们去搞一些准确确认Actor的所有权的时候才有可能用到。

## 实用调试技巧

首先看一下关于复制的问题

在启动游戏时，在不同的窗口下，你看到的世界是一样的，但只是看到的是一样的，实际是不一样的。实际上他们并不是在同一个世界上进行的相关活动。这是减小网络负担的一种优化。一般来说，所有的活动都是由服务器告诉客户端应该怎么执行。

一般来说，客户端只是展示客户端处理出来的结果。所以，如果有一个东西，在服务器上是存在且阻挡的，但在客户端上不存在这个实例，就会有类似网络延迟的丢包问题。

我们创建一个简单的Actor类对象，在细节中，



默认是这样的。如果这里我们取消勾选“客户端上的网络加载”，将这个Actor放到场景中后，再开始游戏，就会发现，在客户端上是看不到这个Actor的，但是当客户端的Character走过去的时候，会被阻挡。

原因就是，虽然不在一个世界，但是客户端的所有行为都是需要服务器提供数据的。



现在的问题是：我们调试的时候所有的行为几乎无延迟，但是在真实的网络环境中，是有延迟的

接下来我们来模拟一下真实的网络环境

在图示的高级设置中，有启用网络模拟





我们可以按照图示，固定设置延迟400。

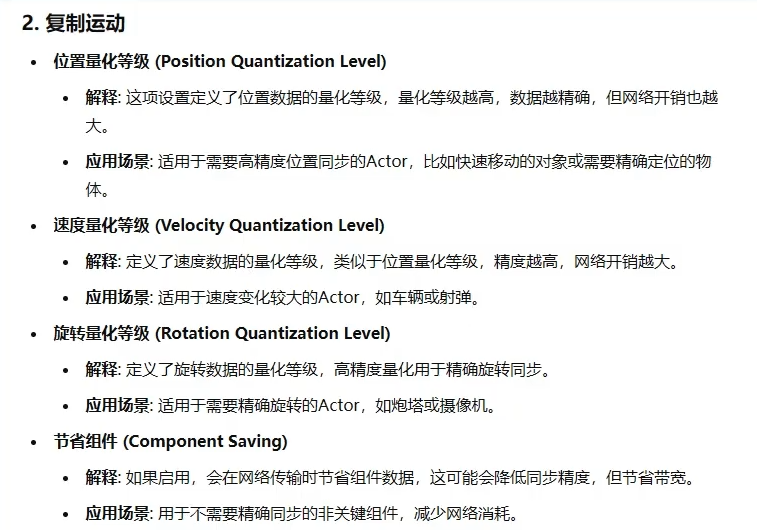
此时再进入游戏，在客户端上会发现，可以经过碰撞体，但会在延迟之后把自己拉回到碰撞体的位置。

在细节--赋值中有很多参数。这些参数的含义是：









## RPC网络复制

RPC（Remote Procedure Call，远程过程调用）是一种网络编程技术，用于实现不同计算机之间的通信和数据交互。在游戏开发中，RPC常用于实现客户端与服务器之间的同步和交互，例如角色动作、状态更新等。以下是RPC网络复制的简单解释：

**1.RPC的基本概念**

* 客户端（Client）：发起请求的一方，通常是玩家的游戏客户端。
* 服务器（Server）：接收请求并处理的一方，负责管理游戏的整体状态。
* 远程过程调用（RPC）：客户端调用服务器上的某个函数，就像调用本地函数一样，但实际上这个函数运行在服务器上。

**2.RPC的工作原理**

1. 客户端调用RPC函数：

* 客户端调用一个标记为RPC的函数（例如Server\_DoSomething）。这个函数通常在客户端的代码中定义，但实际执行在服务器上。
* 调用时，客户端将函数名和参数打包成一个网络消息，发送给服务器。

1. 服务器接收并执行：

* 服务器收到客户端发送的RPC请求后，解析消息，找到对应的函数和参数。
* 服务器执行该函数，并处理相关逻辑。

1. 返回结果（可选）：

* 如果RPC函数有返回值，服务器会将结果打包成一个响应消息，发送回客户端。
* 客户端收到响应后，继续后续的处理。

**3. RPC在网络复制中的应用**

同步玩家操作： 玩家在本地客户端执行某个操作（例如射击、移动等），客户端通过RPC将操作发送到服务器。 服务器接收操作后，验证并同步到所有其他玩家的客户端上。

服务器广播状态： 服务器可以通过RPC向所有客户端广播某个事件或状态更新（例如怪物生成、其他玩家的位置变化等）。

**4. RPC的优缺点**

优点： 简单直观：RPC的调用方式类似于本地函数调用，易于理解和实现。 高效同步：可以快速实现客户端与服务器之间的同步操作。

缺点： 网络延迟：RPC调用依赖网络，可能会受到延迟和丢包的影响。 安全性问题：如果RPC函数没有正确验证，可能会被恶意利用。

**5. 虚幻引擎中的RPC**

在虚幻引擎（Unreal Engine）中，RPC是一种常用的网络同步机制。开发者可以在C++或蓝图中定义RPC函数。例如：

C++中的RPC：

UFUNCTION(BlueprintCallable, Server, Reliable, WithValidation)

void Server\_DoSomething(int32 Param);

Server：表示这是一个服务器端的RPC函数。

Reliable：表示这个RPC调用是可靠的（保证送达）。

WithValidation：表示服务器端有验证逻辑。

蓝图中的RPC： 在蓝图中，可以使用“调用RPC”节点，选择目标（服务器或客户端），并指定要调用的函数。

RPC网络复制是一种强大的工具，用于实现客户端与服务器之间的交互和同步。它通过网络消息传递，让客户端可以调用服务器上的函数，从而实现复杂的网络功能。在游戏开发中，RPC广泛应用于玩家操作同步、状态更新等场景。

## 角色创建及死亡重生（前置，可靠函数）

如果我们的角色死亡了，在我们不解除控制状态的情况下，重新生成一个新的角色。

如果考虑到死亡重生的情况，所以GameMode中的默认Pawn类应当被设置成None。这样的话就不会生成默认角色，生成角色的行为由玩家控制器端进行请求。

首先我们需要关注自定义事件的复制行为：

创建一个自定义事件，在细节中“图表--复制”中，有四个选项



未复制



组播



在服务器上运行



在拥有的客户端上运行



另外还有一个可靠函数的选项



Reliable：表示这个RPC调用是可靠的（保证送达）。

## 可靠函数

在虚幻引擎中，Reliable 函数是用于在网络环境中确保函数调用能够可靠到达目标端的特性。具体来说，它与 Server 或 Client 函数配合使用，确保在网络传输过程中，即使出现带宽限制或网络错误，函数调用也能被正确传递。

**主要作用**

1.保证调用的可靠性：标记为 Reliable 的函数在网络传输时会被确保送达，不会因为网络问题而丢失。这对于一些关键操作（如玩家的攻击动作、重要事件的触发等）非常重要。

2.与网络角色配合：

* Server：当函数标记为 Server, Reliable 时，该函数会在服务器上执行，且调用会被可靠地发送到服务器。
* Client：当函数标记为 Client, Reliable 时，该函数会在客户端上执行，且调用会被可靠地发送到指定的客户端。

3.适用于关键操作：通常用于需要确保执行的操作，例如玩家的交互、游戏逻辑的关键步骤等。

**使用场景**

* 服务器端可靠调用：例如，玩家的攻击动作需要在服务器上验证，因此可以使用 UFUNCTION(Server, Reliable) 来确保该动作的调用能够到达服务器。
* 客户端可靠调用：例如，服务器需要通知客户端某个重要的状态变化，可以使用 UFUNCTION(Client, Reliable)。

**注意事项**

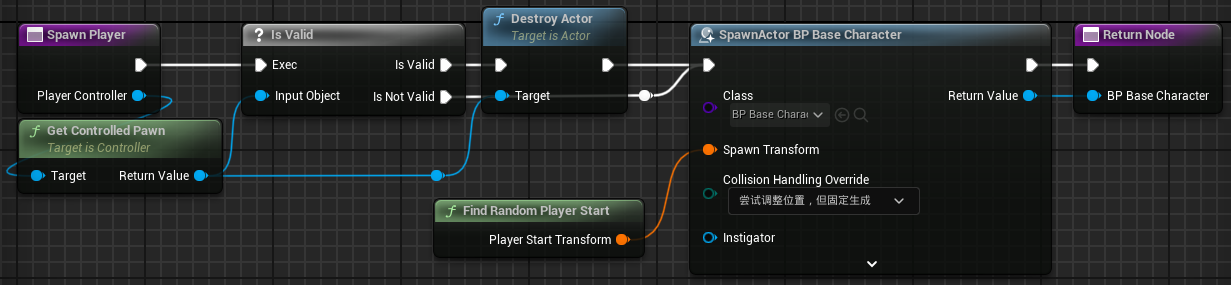
Reliable 函数在网络传输中会占用更多的带宽和资源，因为它需要进行额外的确认机制来确保送达。因此，建议仅在必要时使用。

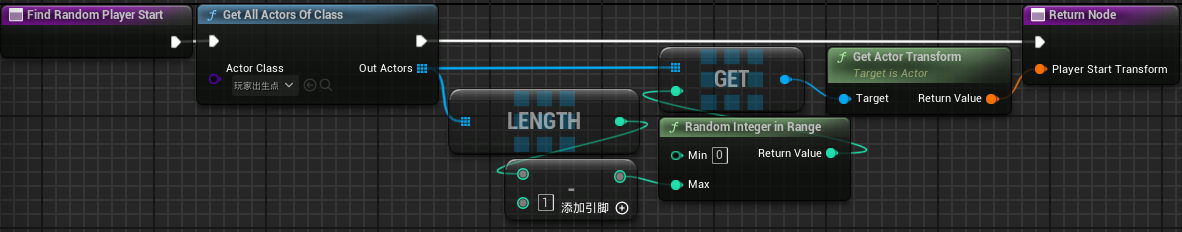
它必须与 Server 或 Client 一起使用，不能单独使用。

## 角色创建及死亡重生（后置）

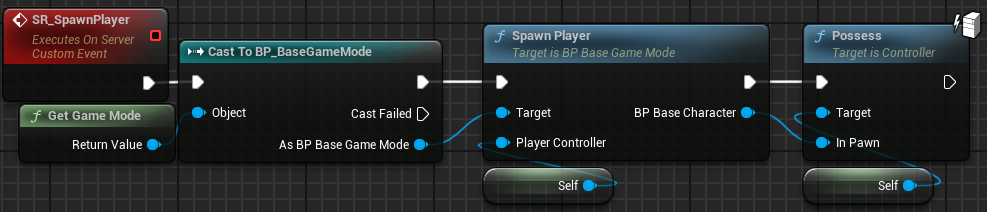
我们已经将GameMode中的默认Pawn类应当被设置成None。

首先在GameMode中实现函数SpawnPlayer





然后在PlayerController中创建自定义事件。



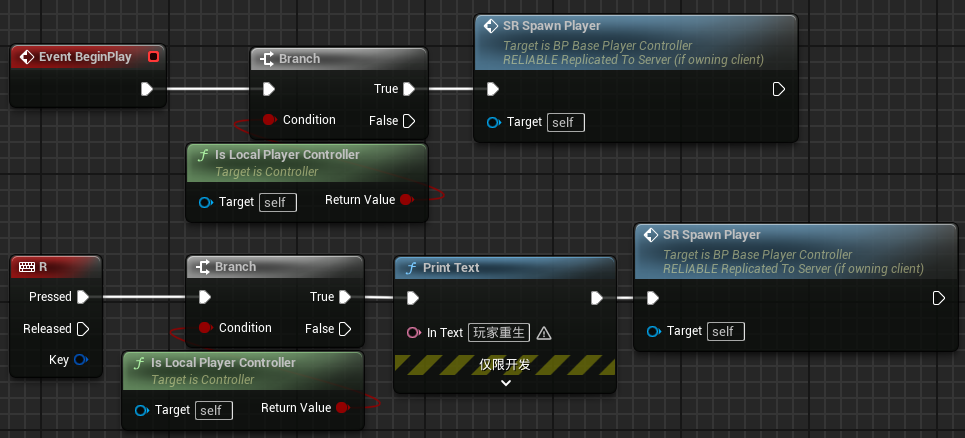
（Possess是PlayerController自带的函数，翻译过来是“控制”）

（Possess函数，这个函数有两个写法，一个是写在SpawnPlayer函数里面，也就是写在GameMode里，还有一个写法就是上图所示的写在PlayerController中。Possess的Target是PlayerController的对象，在哪里写无所谓）

（关于为什么在这里获取游戏模式，因为客户端无法获取GameMode，它只存在于服务器）

这里自定义事件的复制为“在服务器上运行”且为可靠函数。

另外在两个地方调用这个自定义事件



这一系列的操作重点是在玩家不丢失控制权的情况下去生成新的Actor

## 聊天系统

首先创建UI

第一个UI：WBP\_ChatMessage



两个都勾选是变量



两个文本类变量可编辑实例生成时公开。

第二个UI：WBP\_ChatPanel



边界的笔刷颜色为自定义设置。

滚动框的固定显示滚动条勾选。

滚动框与文本框勾选是变量。

在事件图表中创建文本型变量ChatMessage

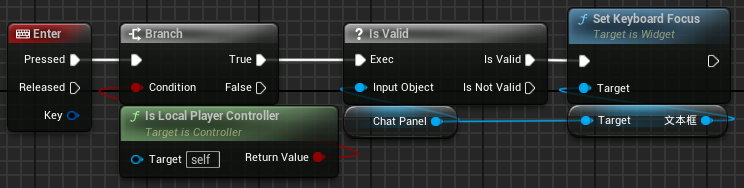
我们所有的入口都是从文本框开始。

在文本框细节的事件中有“文本提交时”，这就是发消息的入口

回到。在发消息之前首先进入PlayerController。首先创建聊天界面



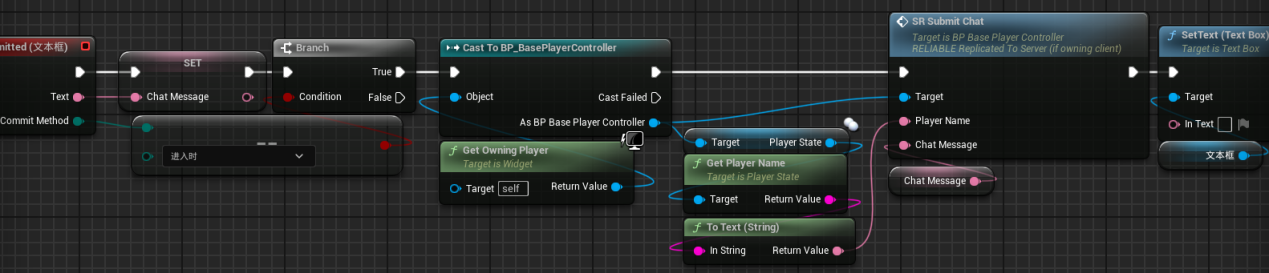
下面这是多人联机聊天的功能



（最后一个节点的函数名翻译过来是“设置键盘聚焦”）

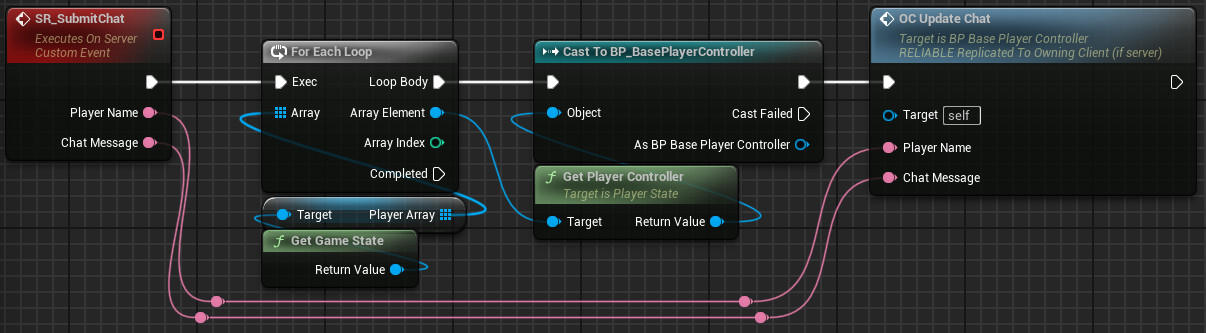
当我们摁下回车键的时候，我们需要激活文本框，变为可输入状态。

然后回到WBP\_ChatPanel，当我们再次摁回车，也就是文本提交时。

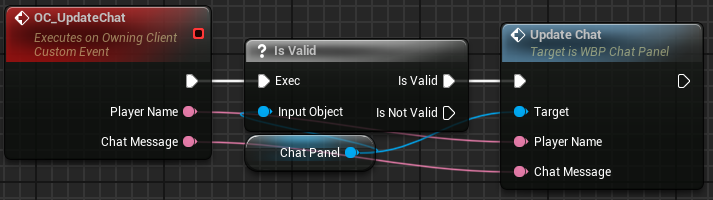


（这里的--的搜索方法是Equal，是Enum的计算类型）

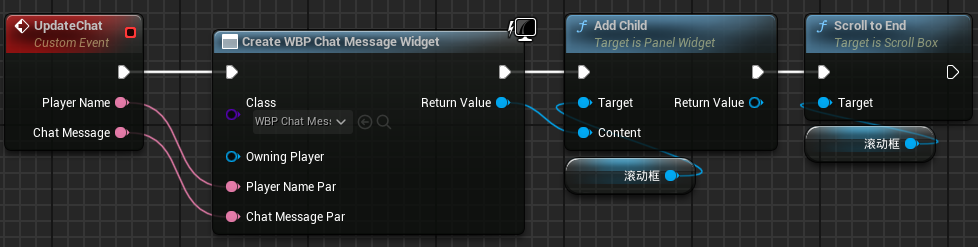
这里，SR\_SubmitChat函数在PlayerController中，



OC\_UpdateChat是在PlayerController中



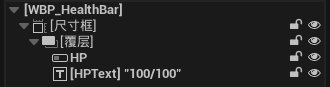
UpdateChat是在WBP\_ChatPanel中



## 玩家名称与血条

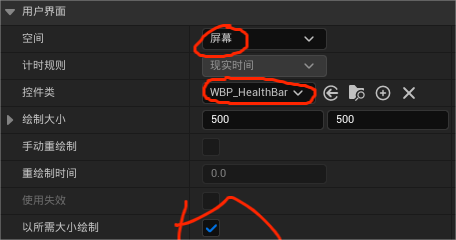
掉血同步等

创建WBP\_HealthBar，两个都勾选是变量。



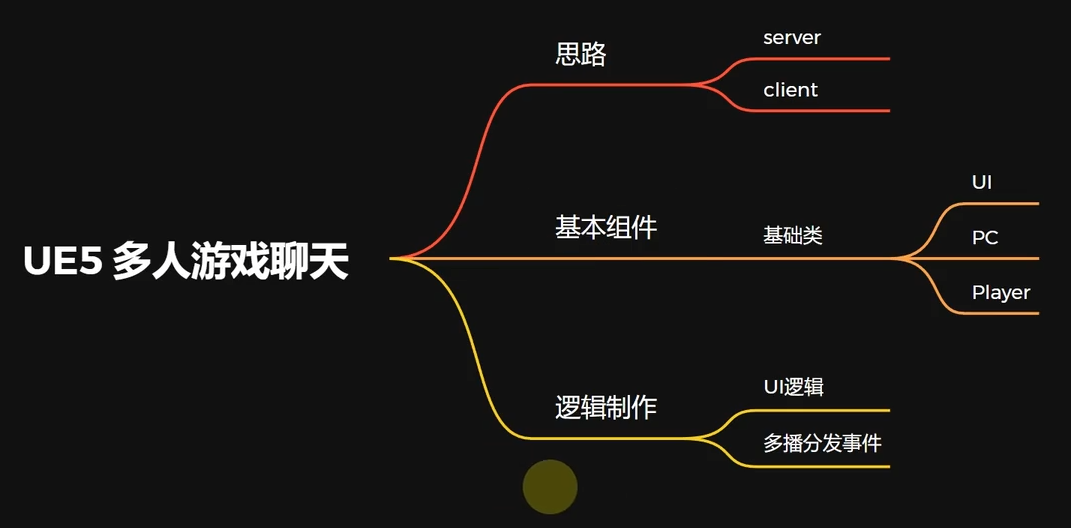
在图表中创建BP\_BaseCharacter类变量。

然后进入BP\_BaseCharacter添加控件组件



后面懒得弄了。

## 多人聊天



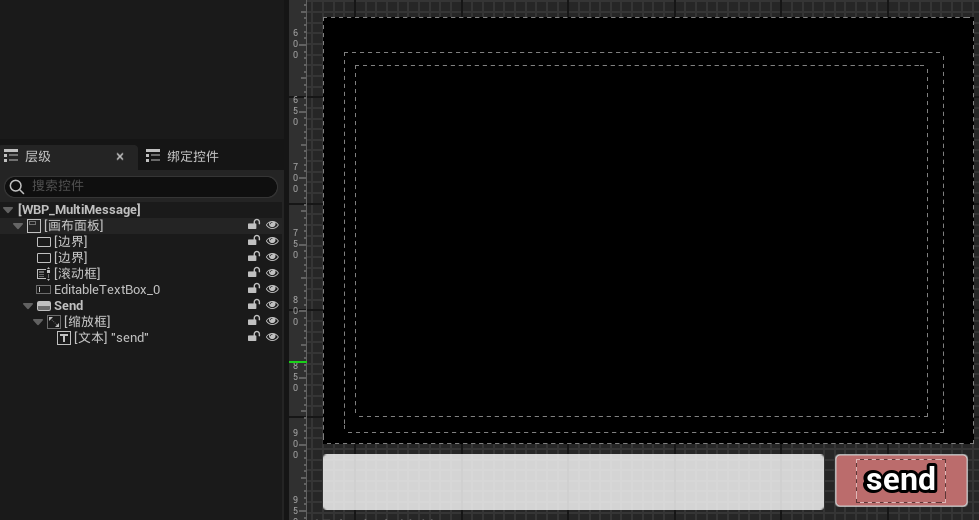
思路：

我们知道，客户端与客户端之间是不能进行通讯的，必须要经过服务器，也就是RPC远程过程调用。

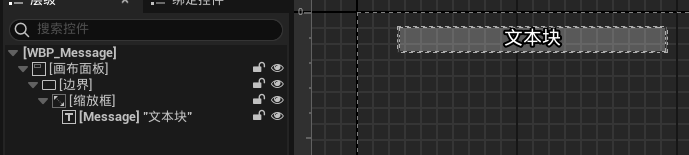
这个方法的理论依据是，客户端、服务器，整个模式我们在客户端里，需要让所有的客户端都收到消息，主要让用到服务器，发一条消息给服务器，然后由服务器将消息广播出去。

基于原有的GameMode，PlayerController，Character

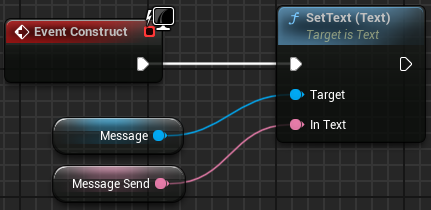
创建UI，命名为WBP\_MultiMessage



创建WBP\_Message



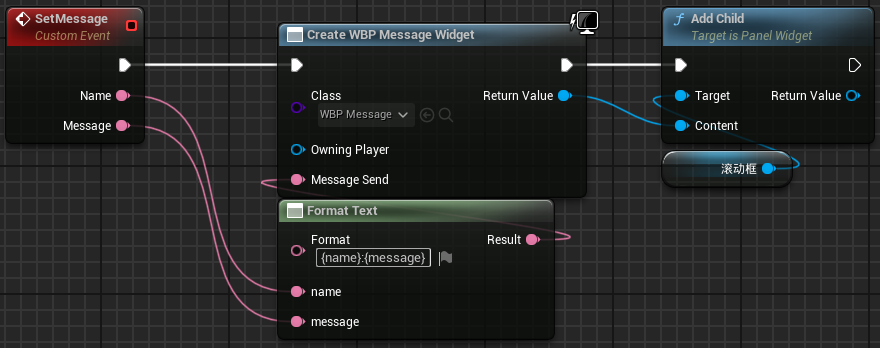
文本为是变量



MessageSend勾选可编辑实例生成是公开

回到WBP\_MultiMessage

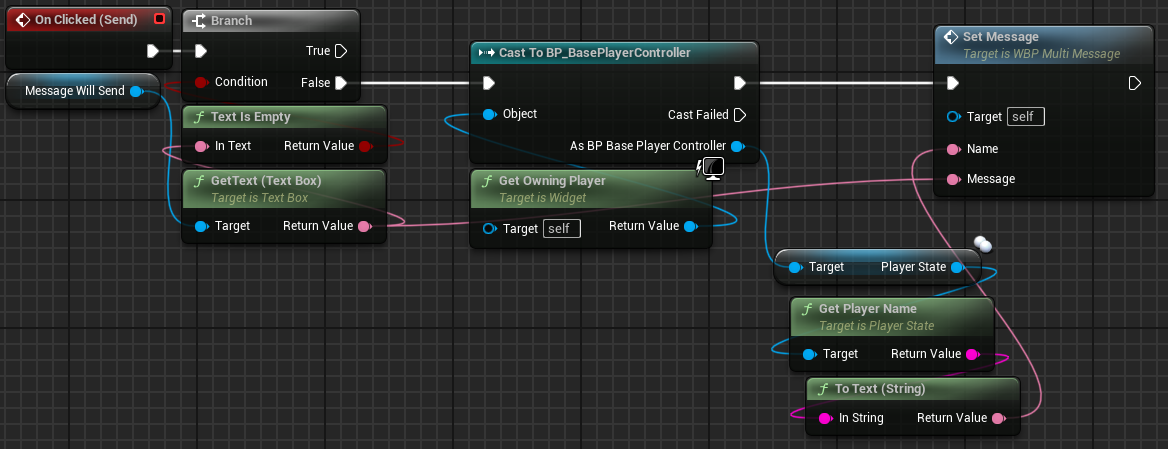
将滚动框勾选是变量，命名为滚动框



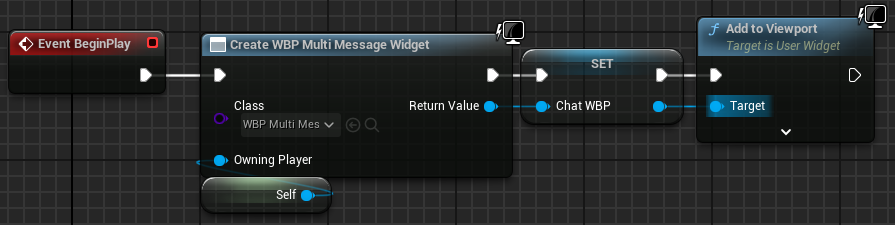
这个事件的目的就是填充滚动框

将“输入--文本框”这个WBP的组件设置为变量，命名为MessageWillSend

在点击按钮时发送信息



然后进入PlayerController，



运行，已经可以在本地输入本地看到了，还不能把信息发送到其他客户端。

另外还有报错



这里要注意一下，我们有两个玩家，也就是两个PlayerController，但是在一个游戏视口里只显示了一份，说明有一份没有显示。Zhegeplayercontroller没有生效，所以会报错。

解决只需要判断一下是不是本地的玩家控制器

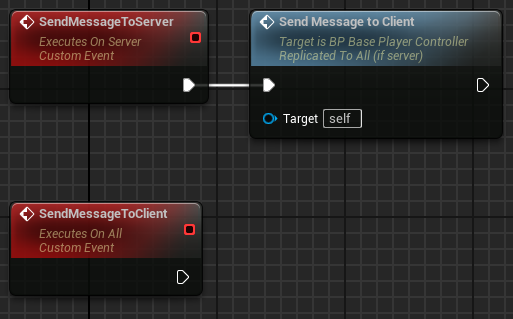


接下来，我们需要点击按钮的时候进行发送，也就是说要调用PlayerController的一个事件，让他去执行服务器的事情。

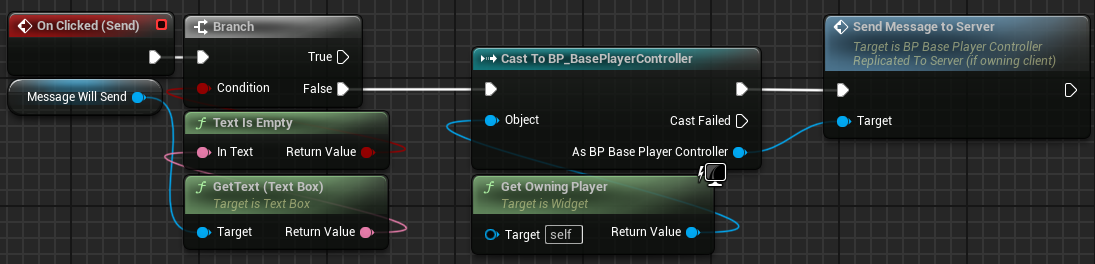
在PlayerController中创建自定义事件，命名为SendMessageToServer。复制选项为“在服务器上运行”

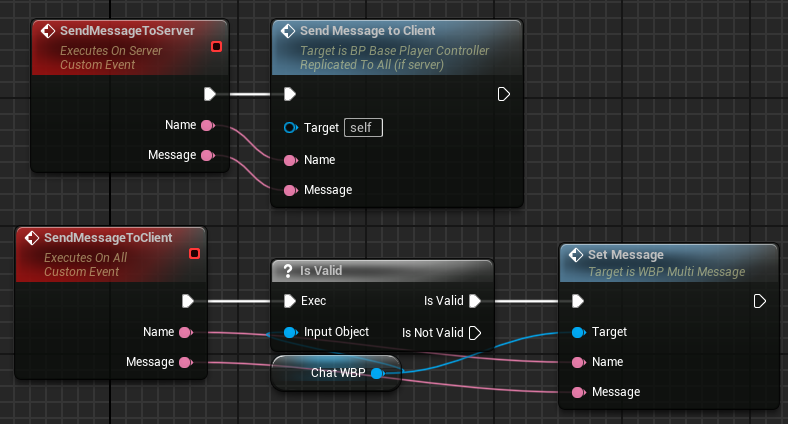
再创建自定义事件，命名为SengMessageToClient。复制选项为“组播”，我们要把他分发给所有的Client。

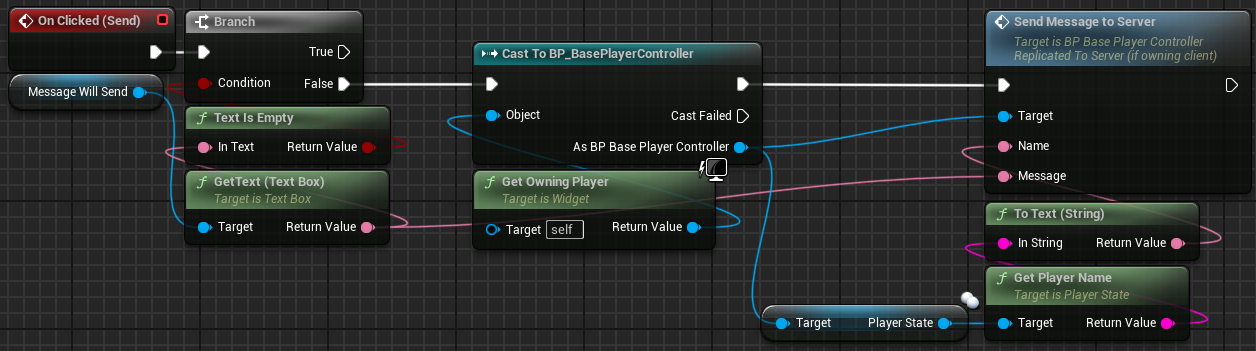
说白了就是在Server上做，做完调用客户端上的事件。



回到WBP\_MultiMessage，我们要在点击按钮之后执行PlayerController的SendMessageToServer事件。

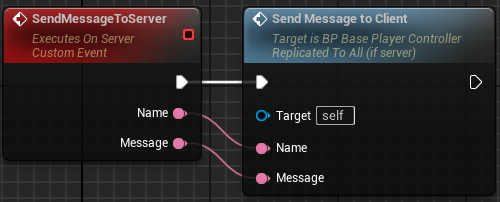






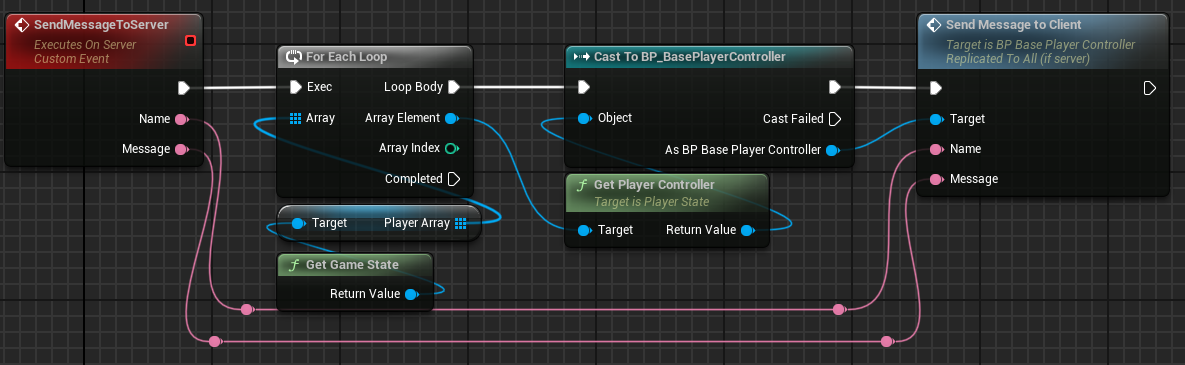
此时进入游戏会发现还是不行，说明多播有点问题。

原因在这里



我们只说发送到客户端，但是我们并不知道哪个是客户端。所以我们需要获取所有的客户端。

通过GetGameState获取客户端



完成

下一篇文章：Steam在线多人联机