虚幻引擎网络系统（二）

虚幻引擎高级程序开发工程师班

火星时代教育

PART 01

带宽管理

虚幻四高级程序开发工程师班

带宽管理

网络游戏是通过计算机硬件通信方案将多台终端链接，组建的玩家沟通环境，从而使得多个玩家链接到一起进行游戏。受限于网络传输环境的影响，软件设计本身必须要考虑网络资源的应用合理性。不能够无限制的使用网络带宽。软件设计的本质是优化资源分配，合理使用资源。

从引擎使用层面上，UE4设计了很多策略来解决网络带宽的节约，作为开发者我们需要了解这些策略，并且避免一些错误的操作，以达到高效的带宽管理目的。

1-1.相关性和优先级

相关性和优先级

相关性 用于决定是否需要在多人游戏期间复制Actor。复制期间将剔除被认为不相关的actor。此操作可节约带宽，以便相关Actor可更加高效地复制。若Actor未被玩家拥有，且不在玩家附近，将其被视为不相关，而不会进行复制。不相关Actor会存在于服务器上，且会影响授权游戏状态，但在玩家靠近前不会向客户端发送信息。覆盖 IsNetRelevantFor 函数以手动控制相关性，并可使用 NetCullDistanceSquared 属性决定成为相关Actor所需距离。

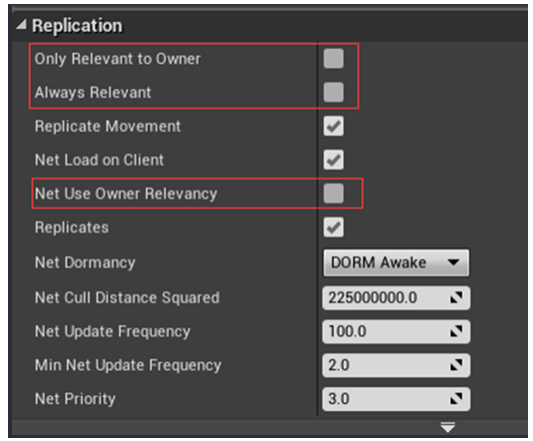
有时在游戏单帧内，没有足够带宽供复制所有相关Actor。因此，Actor拥有 **优先级**（Priority） 值，用于决定优先复制的Actor。Pawn和PlayerController的 NetPriority 默认为 3.0，从而使其成为游戏中最高优先级的Actor，而基础Actor的 NetPriority 为 1.0。Actor在被复制前经历的时间越久，每次成功通过时所处的优先级便越高。

相关性

场景的规模可能非常大，在特定时刻某个玩家只能看到关卡中的一小部分 Actor。场景中的其他大多数 Actor 都不会被看到和听到， 对玩家也不会产生显著的影响。被服务器认为可见或能够影响客户端的 Actor 组会被视为该客户端的相关 Actor 组。 虚幻引擎的网络代码中包含一处重要的带宽优化：服务器只会让客户端知道其相关组内的 Actor。

虚幻引擎（依次）参照以下规则确定玩家的相关 Actor 组。这些测试是在虚拟函数 AActor::IsNetRelevantFor() 中实施。

1. 如果 Actor 是 bAlwaysRelevant、归属于 Pawn 或 PlayerController、本身为 Pawn 或者 Pawn 是某些行为（如噪音或伤害）的发起者，则其具有相关性。
2. 如果 Actor 是 bNetUseOwnerRelevancy 且拥有一个所有者，则使用所有者的相关性。
3. 如果 Actor 是 bOnlyRelevantToOwner 且没有通过第一轮检查，则不具有相关性。
4. 如果 Actor 被附加到另一个 Actor 的骨架模型，它的相关性将取决于其所在基础的相关性。
5. 如果 Actor 是不可见的 (bHidden == true) 并且它的 Root Component 并没有碰撞，那么则不具有相关性，
6. 如果没有 Root Component 的话，AActor::IsNetRelevantFor() 会记录一条警告，提示是否要将它设置为 bAlwaysRelevant=true。
7. 如果 AGameNetworkManager 被设置为使用基于距离的相关性，则只要 Actor 低于净剔除距离，即被视为具有相关性。



相关性

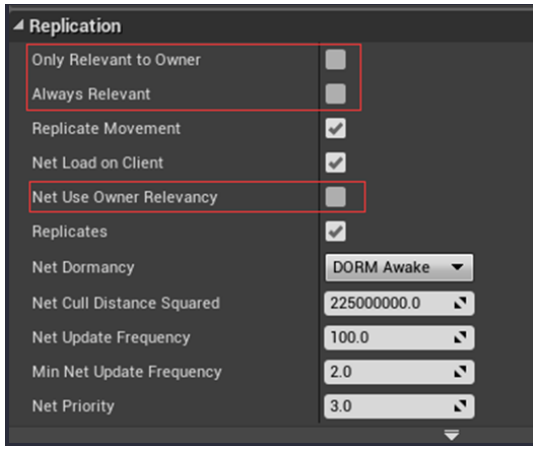
Engine\Source\Runtime\Engine\Private\ActorReplication.cpp

```
bool AActor::IsNetRelevantFor(const AActor* RealViewer, const AActor* ViewTarget, const FVector& SrcLocation) const
{
    if (bAlwaysRelevant || IsOwnedBy(ViewTarget) || IsOwnedBy(RealViewer) || this == ViewTarget || ViewTarget == Instigator)
    {
        return true;
    }
    else if ( bNetUseOwnerRelevancy && Owner)
    {
        return Owner->IsNetRelevantFor(RealViewer, ViewTarget, SrcLocation);
    }
    else if ( bOnlyRelevantToOwner )
    {
        return false;
    }
    else if ( RootComponent && RootComponent->GetAttachParent() && RootComponent->GetAttachParent()->GetOwner() &&
    (Cast<USkeletalMeshComponent>(RootComponent->GetAttachParent()) || (RootComponent->GetAttachParent()->GetOwner() == Owner)) )
    {
        return RootComponent->GetAttachParent()->GetOwner()->IsNetRelevantFor(RealViewer, ViewTarget, SrcLocation);
    }
    else if( bHidden && (!RootComponent || !RootComponent->IsCollisionEnabled()) )
    {
        return false;
    }

    if (!RootComponent)
    {
        UE_LOG(LogNet, Warning, TEXT("Actor %s / %s has no root component in AActor::IsNetRelevantFor. (Make bAlwaysRelevant=true?)"), *GetClass()->GetName(),
        *GetName() );
        return false;
    }

    return !GetDefault<AGameNetworkManager>()->bUseDistanceBasedRelevancy ||
            IsWithinNetRelevancyDistance(SrcLocation);
}
```

优先级

虚幻引擎采用了负载平衡技术来安排所有 Actor 的优先级，并根据它们对游戏的重要性为其分别提供一个公平的带宽份额。

每个 Actor 都有一个名为 NetPriority 的浮点变量。这个变量的数值越大，Actor 相对于其他“同伴”的带宽就越多。和优先级为 1.0 的 Actor 相比，优先级是 2.0 的 Actor 可以得到两倍的更新频度。唯一影响优先顺序的就是它们的比值；所以很显然，您无法通过提高所有优先级的数值来增加虚幻引擎的 网络性能。下面是我们在性能调整中分配的部分 NetPriority 值：



- Actor = 1.0
- Matinee = 2.7
- Pawn = 3.0
- PlayerController = 3.0

计算 Actor 的当前优先级时使用了虚拟函数 AActor::GetNetPriority()。为避免出现饥荒（starvation），AActor::GetNetPriority() 使用 Actor 上次复制后经过的时间 去乘以 NetPriority。同时，GetNetPriority 函数还考虑了 Actor 与观察者的相对位置以及两者之间的距离。

1-2.可靠性

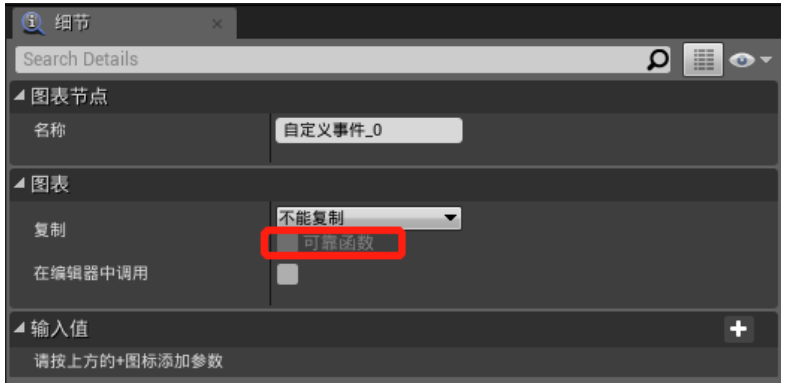
可靠性

UE4网络框架中RPC的操作需要我们提供可靠性标记。在蓝图中，函数和事件默认为不可靠。要将函数指定为可靠，将细节面板（Details Panel）中的 可靠（Reliable） 设置设为 true。在C++中，必须将 Reliable 或 Unreliable 说明符作为 Server、Client 或 NetMulticast 函数，添加到RPC的 UFUNCTION 宏及其状态。

不可靠RPC无法保证必会到达预定目的地，但其发送速度和频率高于可靠的RPC。其最适用于对gameplay而言不重要或经常调用的函数。例如，由于Actor移动每帧都可能变换，因此使用不可靠RPC复制该Actor移动。

可靠的RPC保证到达预定目的地，并在成功接收之前一直保留在队列中。其最适合用于对gameplay很关键或者不经常调用的函数。相关例子包括碰撞事件、武器发射的开始或结束，或生成Actor。

即：高频动作同步建议不可靠，非敏感同步建议不可靠，会影响玩法体验的同步动作设置可靠

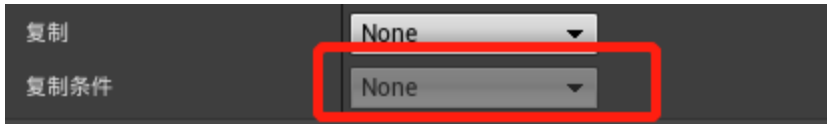


1-3.属性复制条件

复制条件

属性复制被注册后，无法进行取消。所有属性默认的复制条件是：**不发生变化不进行复制**。我们可以使用UE提供的条件进行修正，通过条件的约束设定可以有效的**节约网络带宽**

已经提供的条件



- COND_InitialOnly 该属性仅在初始数据组尝试发送
- COND_OwnerOnly 该属性仅发送至 actor 的所有者
- COND_SkipOwner 该属性将发送至除所有者之外的每个连接
- COND_SimulatedOnly 该属性仅发送至模拟 actor
- COND_AutonomousOnly 该属性仅发送给自主 actor
- COND_SimulatedOrPhysics 该属性将发送至模拟或 bRepPhysics actor
- COND_InitialOrOwner 该属性将发送初始数据包，或者发送至 actor 所有者
- COND_Custom 该属性没有特定条件，但需要通过 SetCustomIsActiveOverride 得到开启/关闭能力（如果没有特定需求尽量少用自定义）

属性复制条件可以很好的实现控制力与性能之间的平衡。它们可以使引擎以更快的速度针对多条连接检查并发送属性，同时让程序员对复制属性的方式和时机进行精细控制。

1-4.网络同步提示

复制条件

- 尽可能少用RPC或复制蓝图函数。在合适情况下改用RepNotify。
- 组播函数会导致会话中各连接客户端的额外网络流量，需尤其少用。
- 若能保证非复制函数仅在服务器上执行，则服务器RPC中无需包含纯服务器逻辑。
- 将可靠RPC绑定到玩家输入时需谨慎。玩家可能会快速反复点击按钮，导致可靠RPC队列溢出。应采取错失限制玩家激活此项的频率
- 若游戏频繁调用RPC或复制函数，如tick时，则应将其设为不可靠。
- 部分函数可重复使用。调用其响应游戏逻辑，然后调用其响应RepNotify，确保客户端和服务器拥有并列执行即可。
- 检查Actor的网络角色可查看其是否为 ROLE_Authority。此方法适用于过滤函数中的执行，该函数同时在服务器和客户端上激活。
- 使用C++中的 IsLocallyControlled 函数或蓝图中的Is Locally Controlled函数，可检查Pawn是否受本地控制。基于执行是否与拥有客户端相关来过滤函数时，此方法十分拥有。
- 构造期间Pawn可能未被指定控制器，因此避免在构造函数脚本中使用 IsLocallyControlled。

PART 02

C++中的同步

虚幻四高级程序开发工程师班

2-1.RPC

在CPP中我们使用RPC的复杂度要高于在蓝图中，在CPP中我们需要借助宏UFUNCTION进行**函数标记**，实现远程调用，标记方式分为以下三种

- UFUNCTION(Server) 声明一个在客户端调用，在服务器端执行的函数
- UFUNCTION(Client) 声明一个在服务端调用，在客户端执行的函数
- UFUNCTION(NetMulticast) 声明一个在服务端调用，在所有终端执行的函数

注意：不要使用具有返回值的函数进行RPC通信。

On Server

标记函数时必须要标记函数**RPC的可靠性（Reliable，Unreliable）**，并且需要实现对应函数名加_Implement**后缀**的函数，将逻辑放入后面的函数中。在客户端调用的函数需要加入**验证操作**，用来检测输入数据准确性。如果 RPC 的验证函数检测到任何 参数存在问题，就会通知系统将发起 RPC 调用的客户端/服务器断开。这样是为了鼓励使用安全的服务器 RPC 函数，同时尽可能方便其他人 添加代码以检查所有参数，确保其符合所有已知的输入限制。

```
public:
    UFUNCTION(Server, WithValidation, Reliable)
    void ChangeNameServer();
    void ChangeNameServer_Implementation();
    bool ChangeNameServer_Validate();
```

On Client

标记函数时必须要标记函数**RPC的可靠性（Reliable，Unreliable）**，并且需要实现对应函数名加**_Implement后缀**的函数，将逻辑放入后面的函数中。

```
UFUNCTION(Client, unreliable)
void ChangeName_Client();
void ChangeName_Client_Implementation();
```

Multicast

标记函数时必须要标记函数**RPC的可靠性（Reliable，Unreliable）**，并且需要实现对应函数名加**_Implement后缀**的函数，将逻辑放入后面的函数中。

```
UFUNCTION(NetMulticast, Reliable)
void ChangeName();
void ChangeName_Implementation();
```

2-2.属性同步

属性同步

参数同步需要将参数注册到复制参数列表，借助UPROPERTY宏进行标记，并且参数同步操作必须在服务器端进行修正，客户端直接修改无法达到同步目的。

UPROPERTY(Replicated)　标记参数为复制参数

UPROPERTY(ReplicatedUsing=函数名) 标记参数为复制参数，复制操作会回调函数，**函数需要使用UFUNCTION标记**（向除去服务器外所有终端进行通知，必须满足相关性）

```
UPROPERTY(Replicated, BlueprintReadOnly)
int32 TNum;
UPROPERTY(ReplicatedUsing = OnRep_TNum2)
int32 TNum2;
```

标记完成后需要绑定到复制参数列表

绑定参数到复制参数列表

所有需要同步的参数都需要绑定到复制参数列表，以便引擎可以方便进行管理，绑定需要实现函数GetLifetimeReplicatedProps，并使用宏DOREPLIFETIME进行注册

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

```
void ANetCharacter::GetLifetimeReplicatedProps(TArray< FLifetimeProperty > & OutLifetimeProps) const
{
    Super::GetLifetimeReplicatedProps(OutLifetimeProps);
    DOREPLIFETIME(ANetCharacter, TNum);
    DOREPLIFETIME(ANetCharacter, TNum2);
}
```

为复制参数增加条件

在进行绑定参数过程中，我们可以通过宏DOREPLIFETIME_CONDITION进行条件添加，可以方便优化，增加对属性复制的控制

```
DOREPLIFETIME_CONDITION(ANetCharacter, TNum2, COND_InitialOnly);
```

可供使用条件包括：

- COND_InitialOnly 该属性仅在初始数据组尝试发送
- COND_OwnerOnly 该属性仅发送至 actor 的所有者
- COND_SkipOwner 该属性将发送至除所有者之外的每个连接
- COND_SimulatedOnly 该属性仅发送至模拟 actor
- COND_AutonomousOnly 该属性仅发送给自治 actor
- COND_SimulatedOrPhysics 该属性将发送至模拟或 bRepPhysics actor
- COND_InitialOrOwner 该属性将发送初始数据包，或者发送至 actor 所有者
- COND_Custom 该属性没有特定条件，但需要通过 SetCustomIsActiveOverride 得到开启/关闭能力

PART 03

关卡切换

虚幻四高级程序开发工程师班

关卡切换

UE4网络框架中支持关卡切换，基于框架特点，UE4的网络切换动作会传递到每一个终端中，即当主机发起切换动作，连接到主机的所有用户均会准备开始切换地图。

UE4 中主要有两种转移方式：**无缝和非无缝方式。**两者的主要区别在于，**无缝转移是一种非阻塞（non-blocking）操作，而非无缝转移则是一种阻塞（blocking）操作。**

当客户端执行非无缝转移时，客户端将与服务器断开连接，然后重新连接到同一服务器，而服务器将准备新的地图以供加载。

我们建议 UE4 多人模式游戏尽量采用无缝转移。这样做通常可以提供更流畅的体验，同时避免重新连接过程中可能出现的问题。**同时无缝模式可以更方便我们携带数据过关。**

有三种情形中必然产生非无缝转移：

- 初次加载地图时
- 初次作为客户端连接服务器时
- 想要终止一个多人模式游戏并启动新游戏时

关卡切换后，将启用新关卡的GameMode，旧关卡的GameMode将被释放。

非无缝切换

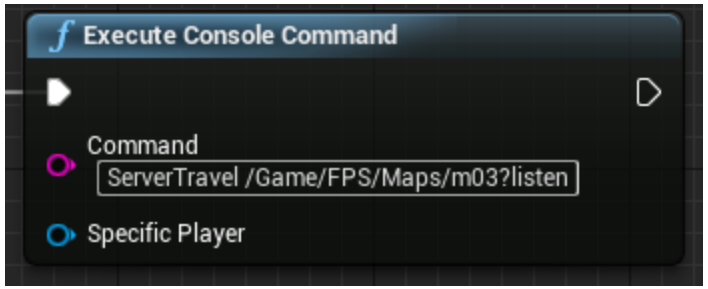
首先测试非无缝切换，**需要将客户端进行打包测试**，由于编辑器模式下，所有启动终端均在一个进程中，并且使用相同的连接，所以测试无法进行。

打包完成后，启动工程，在指令窗口输入：open 地图名（/Game/路径）?listen

注意：英文符号，指令？listen的目的是将给定的地图作为服务器地图，并启动服务器

连接指令：open id　注意：IP地址需要是可访问的域内有效地址，如果是本机可以输入127.0.0.1

切换地图动作应该发生在服务器，在蓝图中由指令完成



最后的？listen必须添加，此动作会携带所有玩家完成切换地图

客户端也可以使用控制台指令启动，具体参照PPT（一）P12中介绍

非无缝切换模式中，当前关卡的所有对象均将被释放，切勿使用任何手段保存当前关卡的对象数据。

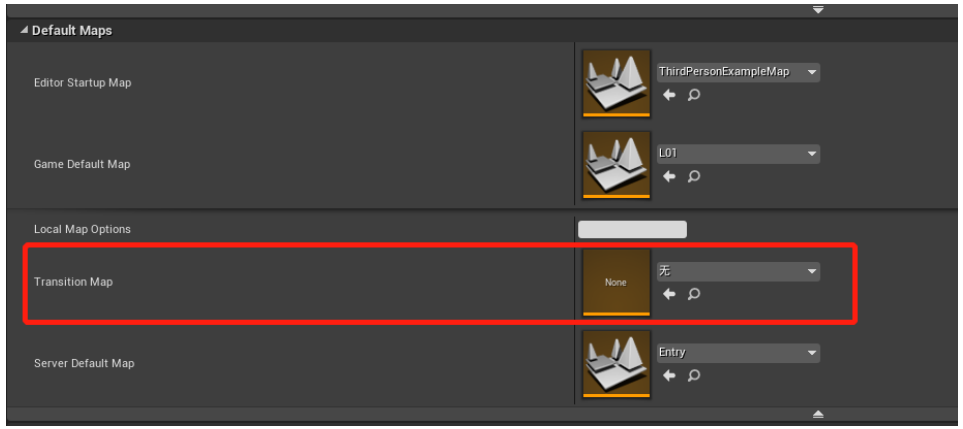
无缝切换

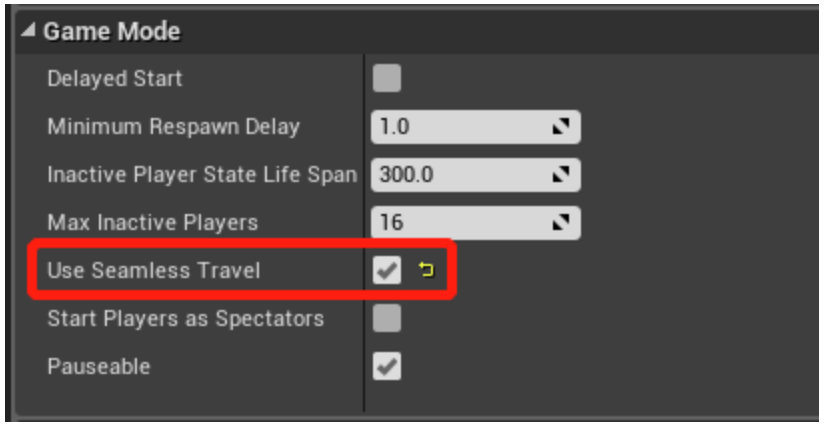
要启用无缝切换，您需要设置一个过渡地图。这需要通过 UGameMapsSettings::TransitionMap 属性进行配置。该属性默认为空，如果您的游戏保持这一默认状态，就会为过渡地图创建一个空地图。

之所以存在过渡地图，是因为必须始终有一个被加载的世界（用于存放地图），所以在加载新地图之前，我们不能释放原有的地图。由于地图可能会非常大，因此让新旧地图同时存放在存储器内绝对是个坏主意，这时就需要过渡地图来帮忙了。

现在，我们可以从当前地图转移到过渡地图，然后可以从那里转移到最终的地图。由于过渡地图非常小，因此在“中转”当前地图和最终地图时不会造成太大的资源消耗。

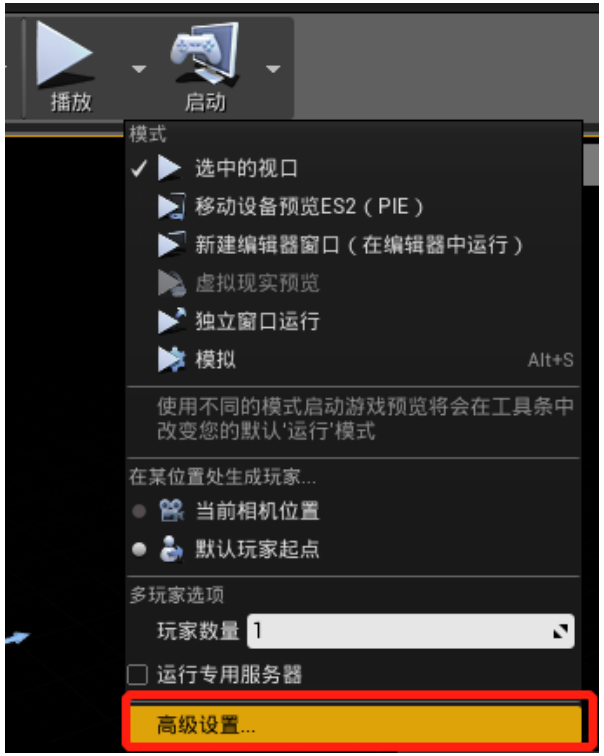
设置好过渡地图后，您需要将 AGameModeBase::bUseSeamlessTravel 设置为 true，这样就可以实现无缝切换了！

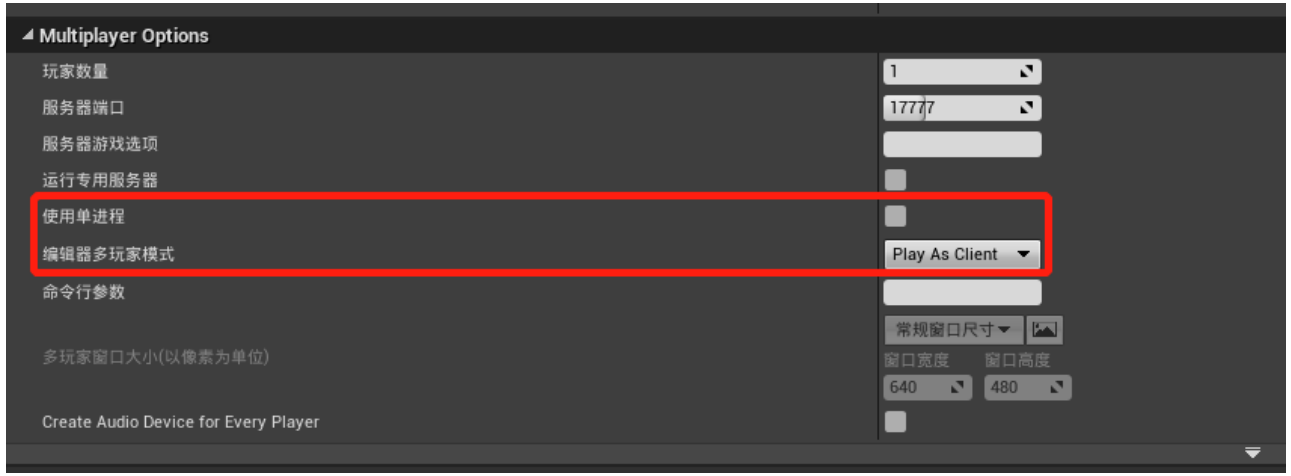




无缝切换

在蓝图中切换指令参照非无缝方式完成，并且禁止在编辑器中使用非无缝方式切换地图。你也可以修改编辑器的启动模式来适应无缝切换操作。





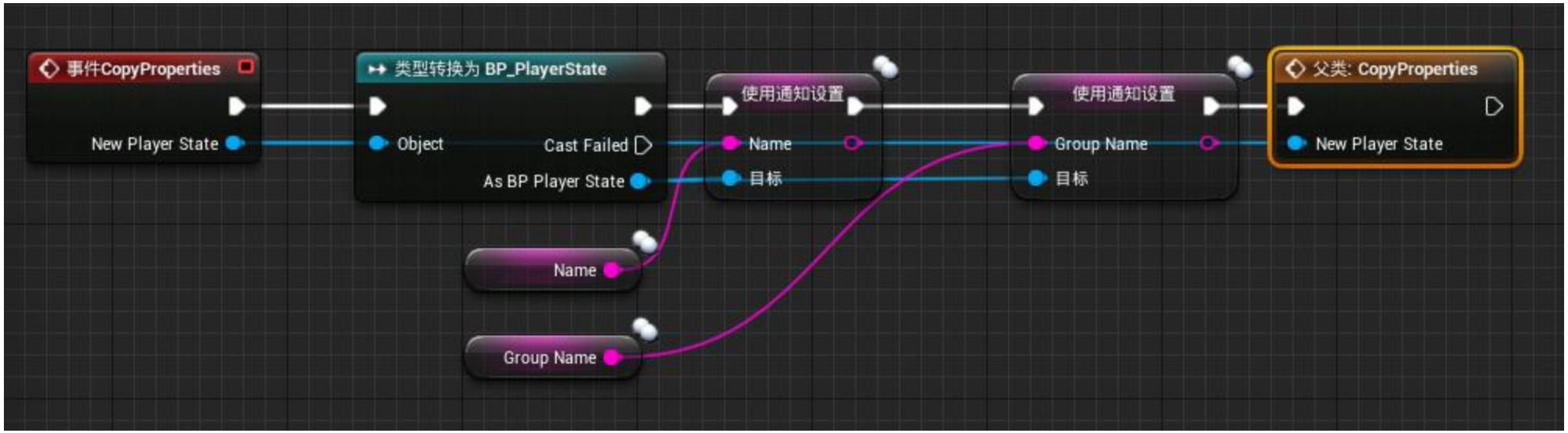
无缝切换流程

下面是执行无缝切换时的一般流程：

1. 标记出要在过渡关卡中存留的actor
2. 转移到过渡关卡
3. 标记出要在最终关卡中存留的actor
4. 转移到最终关卡

通过PlayerState携带数据

在PlayerState中，我们可以将玩家数据在无缝切换模式下带到另外一个关卡中，操作需要在蓝图中或是C++中重写函数CopyProperties。在切换中，会将就有关卡中的PlayerState和新关卡中的PlayerState完成一次见面握手，与此同时，你可以通过此函数完成数据的交换。（切勿携带为添加到漫游列表中的Actor对象数据）



注意：一定要调用父类函数节点。交换时，新的PlayerState可以和当前PlayerState类型不同。

PART 04

关卡切换（C++）

虚幻四高级程序开发工程师班

无缝切换（C++）

在C++中有三个用来驱动转移的主要函数：UEngine::Browse、UWorld::ServerTravel 和 APlayerController::ClientTravel。在确定使用哪个函数时，你可能会感到有些困惑，所以请遵循下面的准则：

`UEngine::Browse`

- 就像是加载新地图时的硬重置。
- 将始终导致非无缝切换。
- 将导致服务器在切换到目标地图前与当前客户端断开连接。
- 客户端将与当前服务器断开连接。
- 专用服务器无法切换至其他服务器，因此地图必须存储在本地（不能是 URL）。

`UWorld::ServerTravel`

- 仅适用于服务器。
- 会将服务器跳转到新的世界/场景。
- 所有连接的客户端都会跟随。
- 这就是多人游戏在地图之间转移时所用的方法，而服务器将负责调用此函数。
- 服务器将为所有已连接的客户端玩家调用 APlayerController::ClientTravel。

`APlayerController::Client Travel`（联网模式切勿主动调用）

- 如果从客户端调用，则转移到新的服务器
- 如果从服务器调用，则要求特定客户端转移到新地图（但仍然连接到当前服务器）

无缝切换（C++）

在C++中，三个函数均可以完成地图切换，而我们需要使用的是UWorld::ServerTravel，来完成地图之间的切换。

无缝切换中的存留 Actor（C++）

在使用无缝切换时，可以将（存留） actor 从当前关卡带到新的关卡。这适用于一些特定的 actor，如道具栏物品和玩家等。

默认情况下，这些 actor 将自动存留：

- GameMode actor（仅限服务器）
    - 通过 AGameModeBase::GetSeamlessTravelActorList 额外添加的任何 actor
- 拥有一个有效的 PlayerState （仅限服务器）的所有控制器
- 所有 PlayerControllers （仅限服务器）
- 所有本地 PlayerControllers （服务器和客户端）
    - 通过 APlayerController::GetSeamlessTravelActorList （在本地`PlayerControllers`上调用）额外添加的任何 actor

PART 05

框架特殊函数

虚幻四高级程序开发工程师班

PART 06

创建连接

虚幻四高级程序开发工程师班

感谢观看