同步的执行顺序：

1、调用“TickComponent”函数

2、计算了帧的加速度和旋转变化

3、调用“PerformMovement”（针对本地控制的角色）或“ReplicateMoveToServer”（针对网络客户端）

“ReplicateMoveToServer”保存移动（到待定的移动列表中），在本地调用“PerformMovement”，然后通过调用复制的函数“ServerMove” 将移动复制到服务器。“ServerMove”接收移动参数，包括客户端的最终位置和时间戳。它在服务器上执行，在服务器上解码移动参数并导致适当的移动发生。然后它查看结果位置，并计算自最后一次客户端响应以来的时间，以及客户端声明的位置与服务器确定的位置之间的差异。如果差异足够大，服务器调用“ClientAdjustPosition”，它将复制到客户端并传递校正的位置。

4、当客户端再次调用“TickComponent”时，如果收到来自服务器的校正，客户端将在调用“PerformMovement”之前调用“ClientUpdatePosition” 。此过程将重演在服务器调整移动的时间戳之后发生的待定移动列表中的所有移动。

**核心函数**

1. Actor

/\*\* Returns the properties used for network replication \*/

virtual void AActor::GetLifetimeReplicatedProps( TArray< FLifetimeProperty > & OutLifetimeProps ) const

DOREPLIFETIME( AActor, bReplicateMovement );

DOREPLIFETIME\_CONDITION\_NOTIFY( AActor, AttachmentReplication, COND\_Custom, REPNOTIFY\_Always );

DOREPLIFETIME\_CONDITION\_NOTIFY( AActor, ReplicatedMovement, COND\_SimulatedOrPhysics, REPNOTIFY\_Always );

virtual void AActor::PreReplication( IRepChangedPropertyTracker & ChangedPropertyTracker ) // 在属性同步前调用的一个函数, 这里生成需要同步的位置信息, 并重写是否需要位置同步

/\*\* Fills ReplicatedMovement property \*/

virtual void GatherCurrentMovement();// 生成物体同步信息的函数

/\*\* Used for replication of our RootComponent's position and velocity \*/

UPROPERTY(EditDefaultsOnly, ReplicatedUsing=OnRep\_ReplicatedMovement, Category=Replication, AdvancedDisplay)

struct FRepMovement ReplicatedMovement; // 位置同步的结构体信息

UPROPERTY(Transient, ReplicatedUsing=OnRep\_AttachmentReplication)

struct FRepAttachment AttachmentReplication; // Attach相关的结构体信息

1. Character

void ACharacter::GetLifetimeReplicatedProps( TArray< FLifetimeProperty > & OutLifetimeProps ) const

{

Super::GetLifetimeReplicatedProps( OutLifetimeProps );

DOREPLIFETIME\_CONDITION( ACharacter, RepRootMotion, COND\_SimulatedOnly );

DOREPLIFETIME\_CONDITION( ACharacter, ReplicatedBasedMovement, COND\_SimulatedOnly );

DOREPLIFETIME\_CONDITION( ACharacter, ReplicatedServerLastTransformUpdateTimeStamp, COND\_SimulatedOnlyNoReplay );

DOREPLIFETIME\_CONDITION( ACharacter, ReplicatedMovementMode, COND\_SimulatedOnly );

DOREPLIFETIME\_CONDITION( ACharacter, bIsCrouched, COND\_SimulatedOnly );

DOREPLIFETIME\_CONDITION( ACharacter, bProxyIsJumpForceApplied, COND\_SimulatedOnly );

DOREPLIFETIME\_CONDITION( ACharacter, AnimRootMotionTranslationScale, COND\_SimulatedOnly );

}

/\*\* Replicated version of relative movement. Read-only on simulated proxies! \*/

UPROPERTY(ReplicatedUsing=OnRep\_ReplicatedBasedMovement)

struct FBasedMovementInfo ReplicatedBasedMovement;

/\*\* Replicated Root Motion montage \*/

UPROPERTY(ReplicatedUsing=OnRep\_RootMotion)

struct FRepRootMotionMontage RepRootMotion;

UPROPERTY(BlueprintReadOnly, replicatedUsing=OnRep\_IsCrouched, Category=Character)

uint32 bIsCrouched:1;

/\*\* Scale to apply to root motion translation on this Character \*/

UPROPERTY(Replicated)

float AnimRootMotionTranslationScale;

/\*\* CharacterMovement MovementMode (and custom mode) replicated for simulated proxies. Use CharacterMovementComponent::UnpackNetworkMovementMode() to translate it. \*/

UPROPERTY(Replicated)

uint8 ReplicatedMovementMode;

/\*\* CharacterMovement ServerLastTransformUpdateTimeStamp value, replicated to simulated proxies. \*/

UPROPERTY(Replicated)

float ReplicatedServerLastTransformUpdateTimeStamp;

/\*\* Set to indicate that this Character is currently under the force of a jump (if JumpMaxHoldTime is non-zero). IsJumpProvidingForce() handles this as well. \*/

UPROPERTY(Transient, Replicated)

uint32 bProxyIsJumpForceApplied : 1;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Replicated, Category=Character, Meta=(ClampMin=0.0, UIMin=0.0))

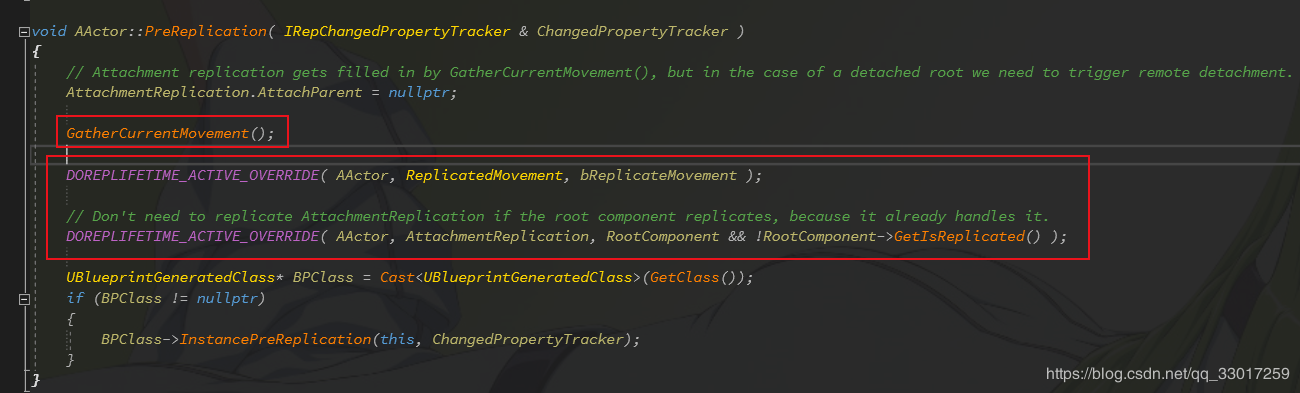
float JumpMaxHoldTime;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Replicated, Category=Character)

int32 JumpMaxCount;

整体流程

1. UE4中物体位置同步的实现服务器向客户端是靠属性同步。在同步前, 需要产生需要同步的位置信息,并重写是否需要位置同步ReplicatedMovement只有**bReplicateMovement为True时**，才进行同步**AttachmentReplication在**根组件存在并且根组件不同步**的时候同步。如果需要同步, 那么然后物体ReplicatedMovement 或 AttachmentReplication会被同步到客户端, 并调用相应的OnRep函数执行对应的逻辑**



代码细节

一、void AActor::GatherCurrentMovement()

1、该函数在每次同步前都需要调用, 产生相应的位置信息。这里的产生条件是bReplicateMovement 或者 RootComponent->GetAttachParent()。前者为了生成 ReplicatedMovement, 后者为了生成 AttachmentReplication。

2、判断是否开启物理, 物理的同步和没有物理的同步方式不一样。

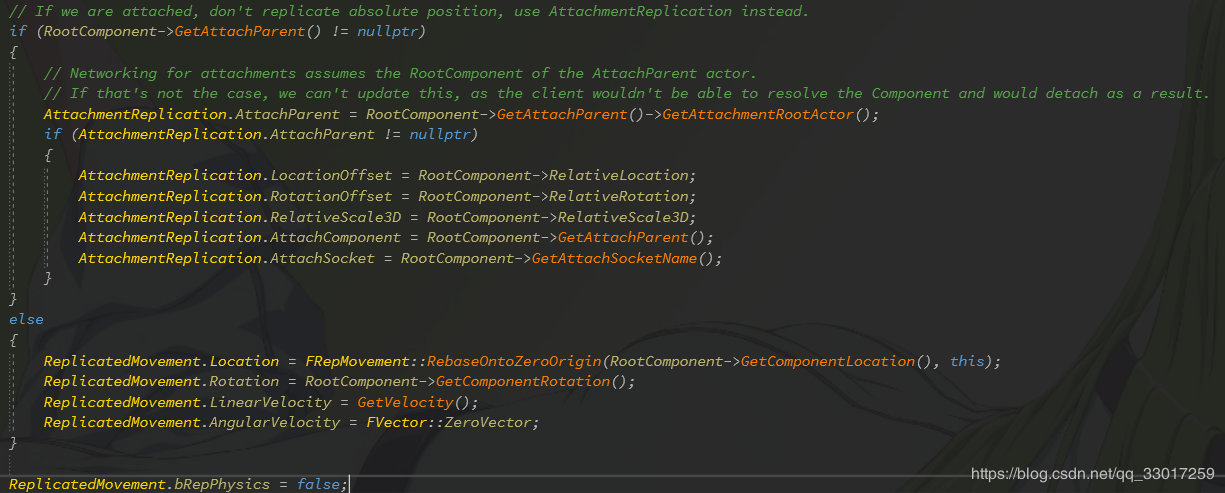
1）：物理同步相关：可以看到物理同步会同步位置旋转速度和是否休眠。



2）：没有物理的同步：两种情况, 有没有父组件

有, 生成AttachementReplication, 这里同步位置, 旋转, 缩放, AttachComponent和SocketName。

没有, 生成ReplicatedMovement, 这里同步位置, 旋转, 速度。



调用堆栈：

> UE4Editor-Engine.dll!ACharacter::PreReplication(IRepChangedPropertyTracker & ChangedPropertyTracker) 行 1405 C++

UE4Editor-Engine.dll!AActor::CallPreReplication(UNetDriver \* NetDriver) 行 1059 C++

UE4Editor-ReplicationGraph.dll!UReplicationGraph::ReplicateSingleActor(AActor \* Actor, FConnectionReplicationActorInfo & ActorInfo, FGlobalActorReplicationInfo & GlobalActorInfo, FPerConnectionActorInfoMap & ConnectionActorInfoMap, UNetConnection \* NetConnection, const unsigned int FrameNum) 行 1469 C++

UE4Editor-ReplicationGraph.dll!UReplicationGraph::ReplicateActorListsForConnection\_Default(UNetReplicationGraphConnection \* ConnectionManager, FGatheredReplicationActorLists & GatheredReplicationListsForConnection, FNetViewer & Viewer) 行 1150 C++

UE4Editor-ReplicationGraph.dll!UReplicationGraph::ServerReplicateActors(float DeltaSeconds) 行 827 C++

UE4Editor-Engine.dll!UNetDriver::ServerReplicateActors(float DeltaSeconds) 行 4239 C++

UE4Editor-Engine.dll!UNetDriver::TickFlush(float DeltaSeconds) 行 506 C++

二、virtual void OnRep\_ReplicatedMovement();处理比较简单了, 将这些信息设置到本地。

（ OnRep\_ReplicatedMovement()--》PostNetReceiveVelocity(ReplicatedMovement.LinearVelocity);

**{**

PostNetReceiveLocationAndRotation();--》CharacterMovement->SmoothCorrection(OldLocation, OldRotation, NewLocation, ReplicatedMovement.Rotation.Quaternion());

OnUpdateSimulatedPosition(OldLocation, OldRotation);

**}**

）

客户端向服务器同步：

一、函数：ServerMove（）、ServerMoveNoBase（）、ServerMoveDual（）、ServerMoveDualNoBase（）、ServerMoveDualHybridRootMotion（）、ServerMoveOld（）

二、实现：

1、调用客户端：UCharacterMovementComponent::ServerMove(...) => Calls CharacterOwner->ServerMove(...)

2、调服务器：ACharacter::ServerMove\_Implementation(...) => Calls CharacterMovement->ServerMove\_Implementation

举例：ACharacter ServerMove的定义如下：

UFUNCTION(Server, Reliable, WithValidation)

**void ServerMove(**float TimeStamp, FVector\_NetQuantize10 InAccel, FVector\_NetQuantize100 ClientLoc, uint8 CompressedMoveFlags, uint8 ClientRoll, uint32 View, UPrimitiveComponent\* ClientMovementBase, FName ClientBaseBoneName, uint8 ClientMovementMode);

函数定义为UFUNCTION(Server, Reliable, WithValidation)，需要同时为其定义\_Validate和\_Implementation两个函数

1）：这个函数就比较重要了，他有Server修饰，表示这个函数即使被Client调用，也会在Server上处理，同理也有Client修饰符号

2）： WithValidation表示这个函数在调用远端相应函数时会进行有效性检查，有一种防止作弊的味道

3）：而reliable表示这个函数在本地调用之后，无论网络传输是否丢包，一定会在远端得到调用，是一个可靠传输

virtual bool ServerMove\_Validate(float TimeStamp, FVector\_NetQuantize10 InAccel, FVector\_NetQuantize100 ClientLoc, uint8 CompressedMoveFlags, uint8 ClientRoll, uint32 View, UPrimitiveComponent\* ClientMovementBase, FName ClientBaseBoneName, uint8 ClientMovementMode);//这个函数会在Server上调用，是判断Client发来的参数是否正确可信，就是上边WithValidation对应的检查函数 调用UCharacterMovementComponent:: ServerMove\_Validate（）一般是返回True

virtual void ServerMove\_Implementation(float TimeStamp, FVector\_NetQuantize10 InAccel, FVector\_NetQuantize100 ClientLoc, uint8 CompressedMoveFlags, uint8 ClientRoll, uint32 View, UPrimitiveComponent\* ClientMovementBase, FName ClientBaseBoneName, uint8 ClientMovementMode); //这个函数是ServerMove真正的实现代码，ServerMove本身没有实现！

CPP文件：原函数名不用实现，但**\_Validate**和**\_Implementation**必须实现

自动的Replicate都是Server向Client进行同步的，Client不会实时向Server同步数据。 这一点很重要，因为Replicate都是Server向Client进行同步，所以我们大部分的运算和判定都应发生在Server上，然后Server把运算好的游戏世界同步到Client。

函数标识为UFUNCTION(unreliable, client)：ClientAckGoodMove（）、ClientAdjustPosition（）、ClientVeryShortAdjustPosition（）、ClientAdjustRootMotionPosition（）、ClientAdjustRootMotionSourcePositio（）

服务器向客户端位置调整的调用：

UReplicationGraph::ServerReplicateActors(float DeltaSeconds)—》APlayerController::SendClientAdjustment()—》UCharacterMovementComponent::SendClientAdjustment()—》分为5种情况UCharacterMovementComponent:: ClientAckGoodMove()/UCharacterMovementComponent:: ClientAdjustRootMotionSourcePosition()/UCharacterMovementComponent:: ClientAdjustRootMotionPosition()/UCharacterMovementComponent:: ClientVeryShortAdjustPosition()/UCharacterMovementComponent:: ClientAdjustPosition ()—》通知对应CharacterOwner::XX()方法—》通知客户端对应的UCharacterMovementComponent::XX\_Implementation ()