https://github.com/BillEliot/GASDocumentation_Chinese#concepts-gc-batching-manualrpc

GAS分析：

https://zhuanlan.zhihu.com/p/417226776

GAS

一、基础知识

适用场景：

1. RPG、MOBA等类型游戏；
2. GAS支持网络同步，所以适用于DS服务器，而不适用于自研后端服务器；
3. C++项目，所以较为适用于端游？

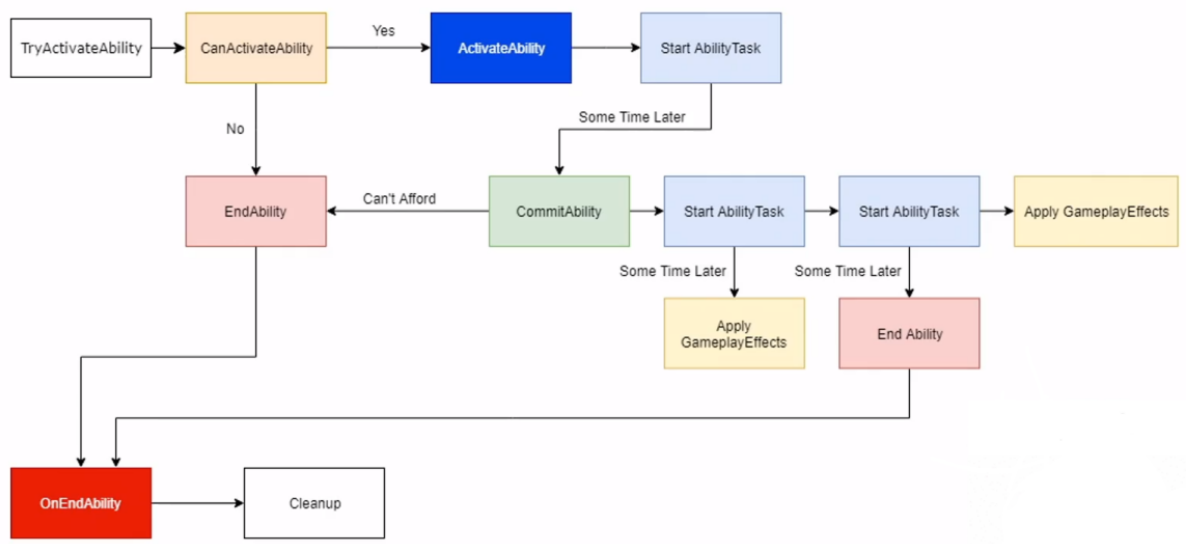
GAS主要包含三个模块（Module）：GameplayAbilities、GameplayTags、GameplayTasks

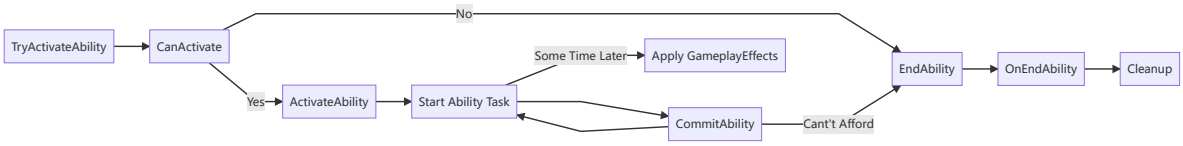
GameplayTag

1. Tag常用结构体类型：

   FGameplayTag：包含一个层级标签

   FGameplayTagContainer：包含一组层级标签





TryActivateAbility: 该结点是通过调用ASC(Ability System Component)的方法尝试使用一个技能。

CanActivate：判断是否可以使用技能，判断条件例如Tag是否互斥、角色是否拥有该技能、该技能是否正在使用等。

ActivateAbility: 如果允许使用一个技能，将调用技能的ActivateAbility方法，该方法我们会在GA蓝图中实现。

StartAbilityTask：一个技能通常是一个或多个的任务，例如播放Montage，播放粒子效果等。该节点表示执行技能任务。

CommitAbility：当技能使用完毕后确认技能结束，用于进行结束后的结算处理等，例如减血耗蓝等。

EndAbility：技能使用完毕后进行状态恢复和清理，如结束Montage等。

技能组件

- UAbilitySystemComponent

  技能组件，用于管理角色的技能系统。由Character包含该组件，包含该组件的Character应继承IAbilitySystemInterface，并实现GetAbilitySystemComponent方法（该方法返回技能组件）。

  声明并初始化ASC

```
UPROPERTY()
URPGAbilitySystemComponent* AbilitySystemComponent;

AbilitySystemComponent = CreateDefaultSubobject<URPGAbilitySystemComponent>
(TEXT("AbilitySystemComponent"));
AbilitySystemComponent->SetIsReplicated(true);      //启用网络同步
```

技能

- UGameplayAbility

  通过继承该类实现一个技能。然后通过FGameplayAbilitySpec包装并初始化技能然后将处理后的GameplayAbilitySpec传给技能组件调用。

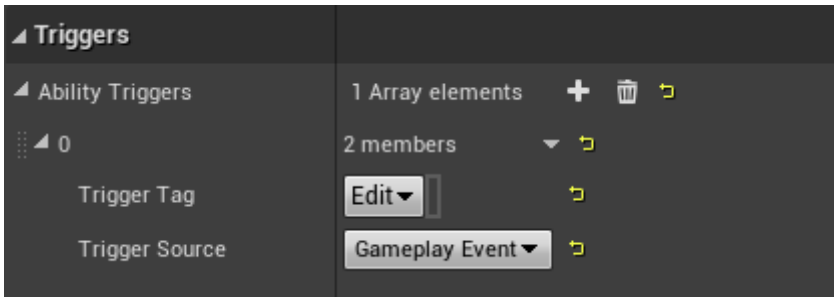
  授权技能：

```
AbilitySystemComponent->InitAbilityActorInfo(AActor* InOwnerActor, AActor*
InAvatarActor);

AbilitySystemComponent->GiveAbility(FGameplayAbilitySpec(StartupAbility,
GetCharacterLevel(), INDEX_NONE, this));
```

使用技能：

```
bool TryActivateAbilitiesByTag(const FGameplayTagContainer&
GameplayTagContainer, bool bAllowRemoteActivation = true);
bool TryActivateAbilityByClass(TSubclassOf<UGameplayAbility>
InAbilityToActivate, bool bAllowRemoteActivation = true);
bool TryActivateAbility(FGameplayAbilitySpecHandle AbilityToActivate, bool
bAllowRemoteActivation = true);
```

技能还可以通过触发器被Tag触发（利用GE为ASC添加Tag）：



- FGameplayAbilitySpec

  技能说明，其中包含技能等级、使用次数、使用后是否销毁等属性。简单理解就是用于包装一个Activatable技能，并在运行时由UAbilitySystemComponent持有并管理。一般是在Character技能初始化时，通过调用技能组件的GiveAbility方法传入一个FGameplayAbilitySpec对象来使用。

- FGameplayAbilitySpecHandle

  技能句柄，在技能组件中提供通过技能句柄使用技能的方法。

GA Details

Tags

- Ability Tags：通过该标签可以激活该技能
- Cancel Abilities with Tag：结束拥有该标签且正在执行的技能
- Block Abilities with Tag：技能执行期间，屏蔽执行拥有该标签技能
- Activation Owned Tags：当技能激活时，会给Owner添加该标签，不能复制。即时结束，Owner也会继续拥有该标签

Input

- Replicate Input Directly：开启此项将总是将输入的按下和释放事件传递到服务器。Epic官方建议不要使用此选项，取而代之的应该使用已存在的输入相关的 `AbilityTasks` 内置的 `Generic Replicated Events`：

```
/** Direct Input state replication. These will be called if
bReplicateInputDirectly is true on the ability and is generally not a good
thing to use. (Instead, prefer to use Generic Replicated Events). */
UAbilitySystemComponent::ServerSetInputPressed()
```

Advanced

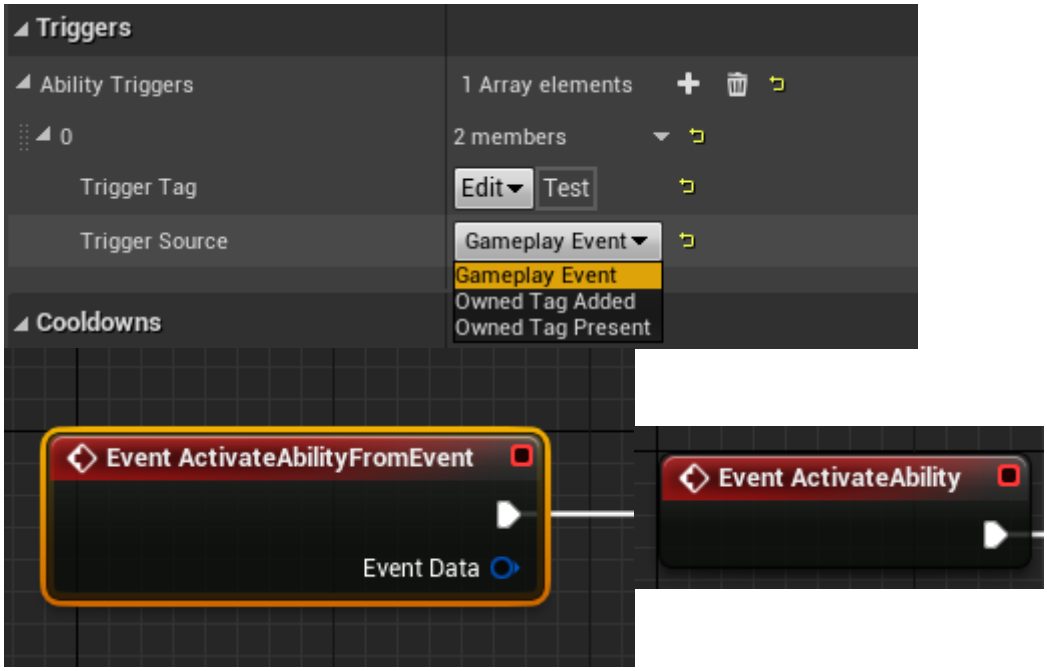
- Replication Policy：默认就是复制的，用不到（该设置可能会被移除）。
- Instancing Policy：
- Server Respects Remote Ability Cancellation：这个选项常常引发问题。这意味着如果客户端的`GameplayAbility`由于取消或自然完成而终止，将强制服务器的版本结束（无论其是否完成）。后一个问题很重要，尤其是对于高延迟玩家使用的本地预测的`GameplayAbilities`。 通常要禁用此选项。
- Retrigger Instanced Ability：
- Net Execution Policy：
- Net Security Policy：

Costs

- Cost Gameplay Effect Class：

Triggers

- Ability Triggers：通过事件激活该技能。就是当拥有该技能的Owner添加或移除触发Tag时，将自动激活该技能。被激活的技能必须删除Event ActivateAbility的实现，而去实现ActivateAbilityFromEvent。示例：比如在某个技能的Activation Owned Tags中添加标签，通过该标签激活其它Trigger技能。



Cooldowns

```
- Cooldown Gameplay Effect Class:
```

属性

- UAttributeSet

  负责定义和持有属性，并且管理属性的变化，包含网络同步。一个ASC可以拥有一个或多个（不同的）AttributeSet，因此可以角色共享一个很大的AttributeSet，也可以按需添加AttributeSet。但不支持动态添加属性。

  可以在属性变化前（PreAttributeChange）后（PostGameplayEffectExecute）处理相关逻辑，也可以通过委托的方式绑定属性变化。

该对象一般在Character声明，被ASC调用。因为不需要显式注册AttributeSet到ASC中，所以需要在声明处添加UPROPERTY宏使其可以在ASC初始化时通过反射查找到AttributeSet将其注册。

声明并初始化AttributeSet

```
UPROPERTY()
URPGAttributeSet* AttributeSet;

AttributeSet = CreateDefaultSubobject<URPGAttributeSet>
(TEXT("AttributeSet"));
```

- FGameplayAttributeData

  属性变量，例如血量、体力等，一般被添加到AttributeSet中。

  声明一个属性：

  1. 包含ATTRIBUTE_ACCESSORS宏
  2. 包含FGameAttributeData类型变量作为属性
  3. 给这些变量设置宏以拥有Getter和Setter等方法
  4. 实现GetLifetimeReplicatedProps、PreAttributeChange、PostGameplayEffectExecute等方法

```
UPROPERTY(BlueprintReadOnly, ReplicatedUsing = OnRep_Health)
FGameplayAttributeData Health;
ATTRIBUTE_ACCESSORS(URPGAttributeSet, Health)
UFUNCTION()
void OnRep_Health(const FGameplayAttributeData& OldValue);
--------------------------------------------------------------------------
--------------------
void OnRep_Health(const FGameplayAttributeData& OldValue)
{
    GAMEPLAYATTRIBUTE_REPNOTIFY(URPGAttributeSet, Health/*当前值*/,
OldValue/*旧值*/)
}
```

  属性修改的回调：

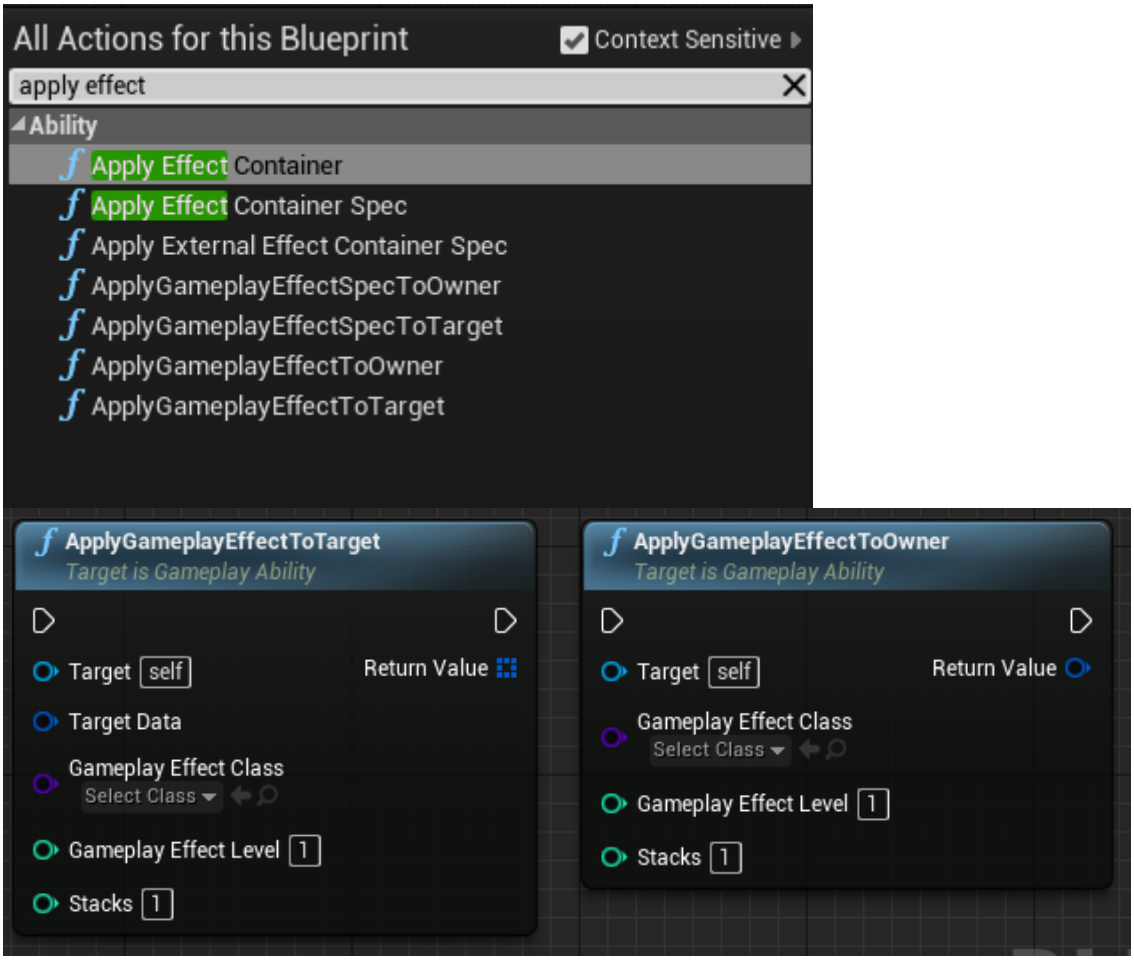
```
//属性修改前回调
virtual viod PreAttributeChange(const FGameplayAttribute& Attribute, float&
NewValue) override;

//属性修改后回调
virtual void PostGameplayEffectExecute(const FGameplayEffectModCallbackData&
Data) override;
```

增益/减益

- UGameplayEffect

  技能效果，在GAS中为广义buff，可以是短期或长期的技能加成或属性消耗。例如击晕效果、体力、血量消耗等。一般是通过GE添加Tags实现。

  可以通过ApplyGameplayEffect等方法使用GE。



- FGameplayEffectContextHandle
- FGameplayEffectSpec
- FGameplayEffectSpecHandle

  FGameplayEffectSpec包含一个GameplayEffect对象，然后GameplayEffectSpec初始化一个FGameplayEffectSpecHandle。

  通过技能组件的MakeEffectContext方法获取一个FGameplayEffectContextHandle类型的Effect上下文句柄。然后通过技能组件的MakeOutgoingSpec方法，获取EffectSpec句柄。然后通过技能组件的ApplyGameplayEffectSpecToTarget方法激活。示例：

```
FGameplayEffectContextHandle EffectContext = AbilitySystemComponent-
>MakeEffectContext();
EffectContext.AddSourceObject(this);

FGameplayEffectSpecHandle NewHandle = AbilitySystemComponent-
>MakeOutgoingSpec(GameplayEffect, GetCharacterLevel(), EffectContext);
if(NewHandle.IsValid())
{
    FActiveGameplayEffectHandle ActiveGEHandle = AbilitySystemComponent-
>ApplyGameplayEffectSpecToTarget(*NewHandle.Data.Get(),
AbilitySystemComponent);
}
```

表现

- UGameplayCues

  效果（声效、 美术特效等）

- UGameplayTask

  用于更新UI等

标签

- FGameplayTags

  层级标签，可以用于替代bool或enum的状态。当技能或效果使用某个二级或三级子标签时，可以用他们的父级标签表示或判断受限。

- FGameplayTagContainer

  Tag

二、技能系统流程

1. 启用插件，包含三个模块：GameplayAbilities、GameplayTags、GameplayTasks
2. 初始化PlayerCharacter
    1. PlayerCharacter应包含UAbilitySystemComponent组件并在构造器中初始化，还可以通过SetIsReplicated启用网络复制

    ```
    //创建并初始化技能组件，启用网络复制
    AbilitySystemComponent =
    CreateDefaultSubobject<URPGAbilitySystemComponent>
    (TEXT("AbilitySystemComponent"));
    AbilitySystemComponent->SetIsReplicated(true);

    //初始化属性集，默认情况下网络复制
    AttributeSet = CreateDefaultSubobject<URPGAttributeSet>
    (TEXT("AttributeSet"));
    ```

    2. PlayerCharacter需要继承IAbilitySystemInterface接口并实现其GetAbilitySystemComponent方法（返回技能组件）
    3. 初始化技能

    一般会在BeginPlay或PossessBy Controller时初始化技能。

    ```
    //首先调用以下方法初始化技能相关信息（如过去Movement组件、Skeletal组件等）
    //InOwnerActor是逻辑上的技能控制Actor，InAvatarActor是实体施用技能Actor，两者可以一样
    AbilitySystemComponent->InitAbilityActorInfo(AActor* InOwnerActor,
    AActor* InAvatarActor);

    //仅在服务器上授权技能
    if (GetLocalRole() == ROLE_Authority && !bAbilitiesInitialized)
    {
        /*  初始化GameplayAbility。
            这里GameplayAbilities为空，所以实际并没有执行。
    ```

```
        分析：这里的GameplayAbilites类型实际为TSubclassOf<UGameplayAbility>数组，实际作用是为Character授权技能。但是不同武器的技能表现不一        样，如果直接把所有Character技能一次性授权，当通过武器使用技能时，将不能确定执行哪个技能。所以，此处能授权的技能应与武器或道具无关。而与        道具（武器）有关的技能类则保存在道具数据中，当通过道具数据初始化道具时，再授权技能并保存其映射关系。使用道具技能时，则只需要传入道具，然        后通过其查找应使用的技能spec来激活技能。如下例代码。
    */
    for (TSubclassOf<URPGGameplayAbility>& StartupAbility :
GameplayAbilities)
    {
        AbilitySystemComponent-
>GiveAbility(FGameplayAbilitySpec(StartupAbility, GetCharacterLevel(),
INDEX_NONE, this));
    }

    //初始化GameplayEffect，暂略，后面分析
}
```

通过道具数据授权技能：

```
//通过道具数据，初始化包含映射关系的技能Spec（SlottedAbilitySpecs）。道具数据主要是包含技能等级、技能类型等
TMap<FRPGItemSlot, FGameplayAbilitySpec> SlottedAbilitySpecs;
FillSlottedAbilitySpecs(SlottedAbilitySpecs);

// 通过技能Spec授权技能
for (const TPair<FRPGItemSlot, FGameplayAbilitySpec>& SpecPair :
SlottedAbilitySpecs)
{
    //SlottedAbilities类型为TMap<FRPGItemSlot,
FGameplayAbilitySpecHandle>.实际为添加并获取一个技能句柄，用于保存映射关系。
    FGameplayAbilitySpecHandle& SpecHandle =
SlottedAbilities.FindOrAdd(SpecPair.Key);
    //授权技能
    if (!SpecHandle.IsValid())
    {
        SpecHandle = AbilitySystemComponent-
>GiveAbility(SpecPair.Value);
    }
}
```

4. 施用技能

主要有三个方法去激活使用技能，分别为：TryActivateAbilitiesByTag、TryActivateAbilityByClass、TryActivateAbility

```
bool TryActivateAbilitiesByTag(const FGameplayTagContainer&
GameplayTagContainer, bool bAllowRemoteActivation = true);
bool TryActivateAbilityByClass(TSubclassOf<UGameplayAbility>
InAbilityToActivate, bool bAllowRemoteActivation = true);
bool TryActivateAbility(FGameplayAbilitySpecHandle AbilityToActivate,
bool bAllowRemoteActivation = true);
```

5. 在Character Runtime下初始化时，初始化GameplayEffect，待完善。

   FGameplayEffectContextHandle

   FGameplayEffectSpecHandle

常用方法（其它）

通过Tag获取当前正在使用的包含该tag的技能。

```
void ACharacterBase::GetActiveAbilitiesWithTags(FGameplayTagContainer
AbilityTags, TArray<URPGGameplayAbility*>& ActiveAbilities)
{
    if (AbilitySystemComponent)
    {
        AbilitySystemComponent->GetActiveAbilitiesWithTags(AbilityTags,
ActiveAbilities);
    }
}
```

参考文档

GAS Document

原文：https://github.com/tranek/GASDocumentation

中文：https://blog.csdn.net/pirate310/article/details/106311256

官方文档和教学Blog

官方文档：https://docs.unrealengine.com/en-US/InteractiveExperiences/GameplayAbilitySystem/index.html

教学Blog：https://www.cnblogs.com/JackSamuel/p/7155500.html