2단계_07. 캐릭터 스탯과 위젯

수강일	@2023/07/10
이름	전영재
멘토	Min-Kang Song,현웅 최
작성 상태	Done
강의 시청 여부	☑

Contents

캐릭터 스탯과 위젯

💡 **강의 목표**

- 액터 컴포넌트를 활용한 액터 기능의 확장 방법
- 언리얼 델리게이트를 활용한 발행-구독 모델의 학습
- 액터의 초기화 단계와 위젯 초기화 과정의 이해

---

[1] 캐릭터 스탯의 설정

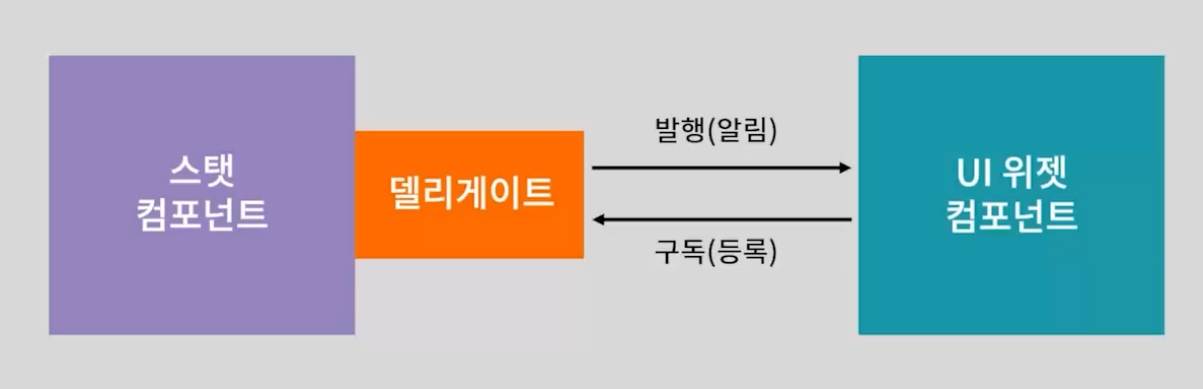
캐릭터의 HP와 같은 스탯은 액터 내에 구현할 수도 있지만, 별도로 액터 컴포넌트를 활용해 해당 기능을 분리해서 구현할 수 있다. 이렇게 기능을 **모듈화 해 분리**하면, 후에 기능의 확장이 필요할 때 편리하게 사용할 수 있다.

델리게이트를 활용한 발행 구독 모델의 구현

스탯 컴포넌트와 UI위젯 컴포넌트는 **델리게이트**를 사용한다면 서로를 참조할 필요가 없어진다.

델리게이트를 사용해 약간의 결합고리만 만들어주고, 묵묵히 자기 내부의 기능을 수행하게 만들어 **느슨한 결합**을 만드는 것이 좋다.

본 실습에서는 스탯 컴포넌트에서 델리게이트를 생성하고, UI위젯 컴포넌트에서는 이 델리게이트를 구독하여 스탯 컴포넌트 내의 이벤트 발생을 감지하고 UI의 수치가 변경되는 로직을 작성할 것이다.



다음과 같이 Delegate를 선언해준다.

```
#include "CoreMinimal.h"
#include "Components/ActorComponent.h"
#include "ABCharacterStatComponent.generated.h"

DECLARE_MULTICAST_DELEGATE(FOnHpZeroDelegate);
DECLARE_MULTICAST_DELEGATE_OneParam(FOnHpChangedDelegate, float /*CurrentHp*/);

UCLASS( ClassGroup=(Custom), meta=(BlueprintSpawnableComponent) )
class ARENABATTLE_API UABCharacterStatComponent : public UActorComponent
{
  GENERATED_BODY()

public:
  UABCharacterStatComponent();
  ...
public:
  FOnHpZeroDelegate OnHpZero;
  FOnHpChangedDelegate OnHpChanged;
  ...
};
```

두 컴포넌트를 관리하는 액터에서 (지금의 경우 이들을 소지하는 ABCharacter가 될것이다.) 구독 관계를 설정해준다.

이때 OnHpZero 델리게이트는 `PostInitializeComponents()`에서 바인드 해주었는데, OnHpChanged 델리게이트는 `PostInitializeComponents()`나 `BeginPlay()`가 아닌 UABHpBarWidget의 `NativeConstruct()` 단계에서 호출하도록 설정해줬다. 그 이유는 액터와 위젯의 **생명주기** 때문인데, 다음 **[2]챕터**에서 알아보자.

```
void AABCharacterBase::PostInitializeComponents()
{
  Super::PostInitializeComponents();

  Stat->OnHpZero.AddUObject(this, &AABCharacterBase::SetDead);
}
```

```
void UABHpBarWidget::NativeConstruct()
{
  Super::NativeConstruct();

  HpProgressBar = Cast<UProgressBar>(GetWidgetFromName(TEXT("PbHpBar")));
  ensure(HpProgressBar);
  // ABCharacter의 참조를 없애, 의존성을 낮추기 위해 인터페이스 사용
  IABCharacterWidgetInterface* CharacterWidget = Cast<IABCharacterWidgetInterface>(OwningActor);
  if (CharacterWidget)
  {
```

```
    CharacterWidget->SetupCharacterWidget(this); //OnHpChanged델리게이트와 UpdateHpBar를 바인드해주는 함수
  }
}
```

```
void AABCharacterBase::SetupCharacterWidget(UABUserWidget* InUserWidget)
{
  UABHpBarWidget* HpBarWidget = Cast<UABHpBarWidget>(InUserWidget);
  if (HpBarWidget)
  {
    HpBarWidget->SetMaxHp(Stat->GetMaxHp());
    HpBarWidget->UpdateHpBar(Stat->GetCurrentHp());
    Stat->OnHpChanged.AddUObject(HpBarWidget, &UABHpBarWidget::UpdateHpBar);
  }
}
```

💡 `PostInitializeComponents()`와 `BeginPlay()`의 차이점은 Tick이 시작하고 안하고의 한끝 차이라는 것이다.

💡 UserWidget의 생성자는 특이하게 **const FObjectInitializer라는 인자를 가져야만 한다.**

`UABHpBarWidget(const FObjectInitializer& ObjectInitializer);`

또한 생성자 구현부에서도 다음과 같이 구현해야 한다.

```
UABHpBarWidget::UABHpBarWidget(const FObjectInitializer& ObjectInitializer) : Super(ObjectInitializer)
{
 ...
}
```

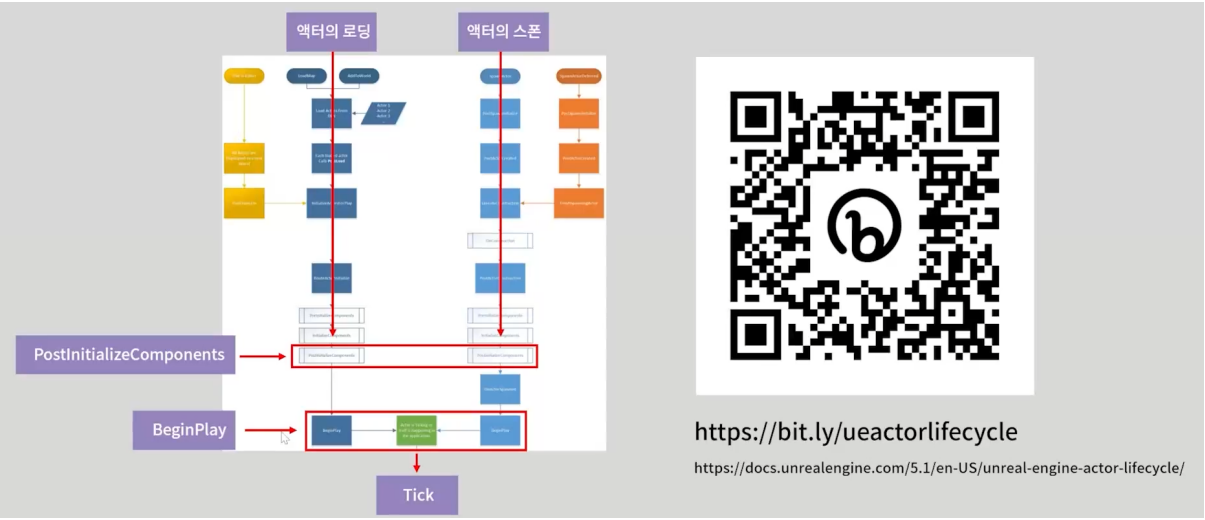
---

[2] 액터의 초기화 과정

두 델리게이트의 **차이점**은 다음과 같다. OnHpZero 는 ABCharacter의 함수와 바인드 되어야 하고 OnHpChanged는 ABHpBarWidget과 바인드 되어야 한다는 것이다.

결론부터 말하자면, OnHpChanged를 똑같이 `PostInitializeComponents`단계에서 바인드 해준다면 크래쉬가 난다. 해당 단계에서 아직 ABHpBarWidget이 생성되지 않았기 때문이다.

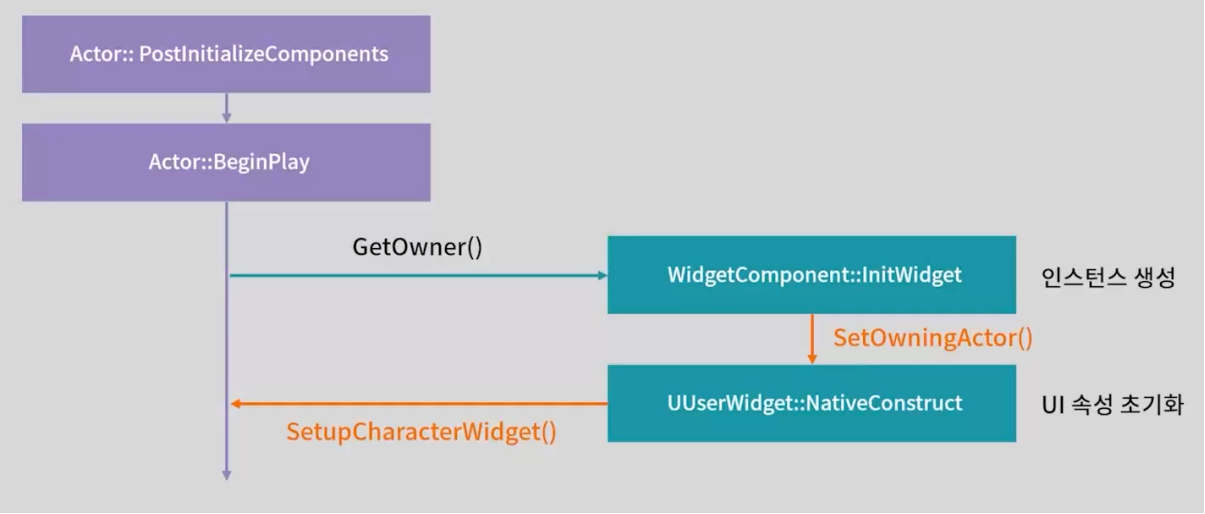
즉, 생명주기로 인해 일어난 문제인데, 이것을 이해하기 위해서는 다음 그림의 이해가 필요하다.



쉽게말하면 스탯 컴포넌트의 경우에는 `PostInitializeComponents()` 단계에서 모든 **초기화**가 완료된다. 그렇기에 이 시기에 액터 내 함수에 바인드를 걸어주는 것이 아무 문제가 없다.

하지만 유저 위젯은 이후 `BeginPlay()` 단계 이후에 **생성**이 되기 때문에 아직 위젯 내 함수와 스탯 컴포넌트를 바인드 할 수 없는 것이다.

두 컴포넌트를 연결해주기 위한 적당한 타이밍이 필요하며, 이것은 UserWidget의 초기화가 완료되는 `NativeConstruct()` 단계이다.



그러나 현재 언리얼 엔진에서는 UUserWidget에서 자신을 소유하고 있는 액터의 정보를 가져오는 기능이 없다. 그래서 **우리가 클래스를 확장** 해야한다.

WidgetComponent에서 `InitWidget()`이 호출될 때는 액터의 컴포넌트가 생성되어, 해당하는 위젯 인스턴스가 생성된 직후이다. 그렇기에 이 순간부터 소유자액터의 정보를 얻어올 수 있게 된다.

그러나 소유자 정보를 얻어와도 기존 UUserWidget에는 소유자 정보를 담는 프로퍼티가 없기에 우리가 해당 프로퍼티를 만들어줘야 한다.(애초에 소유자에 관심이 없다. 그래서 우리가 관심가지라고 직접 넣어주는 거다)

> 💡 유사한 이름으로 헷갈리지만 다음을 짚어가며 천천히 읽어보자.
> WidgetComponent = 액터에 장착되는 컴포넌트
> UserWidget = UI 위젯

```
#include "CoreMinimal.h"
#include "Components/WidgetComponent.h"
#include "ABWidgetComponent.generated.h"

UCLASS()
class ARENABATTLE_API UABWidgetComponent : public UWidgetComponent
{
  GENERATED_BODY()

protected:
  virtual void InitWidget() override;
};
```

```
#include "UI/ABWidgetComponent.h"
#include "ABUserWidget.h"

void UABWidgetComponent::InitWidget()
{
  Super::InitWidget();

  UABUserWidget* ABUserWidget = Cast<UABUserWidget>(GetWidget());
  if (ABUserWidget)
  {
    ABUserWidget->SetOwningActor(GetOwner());
  }
}
```

```
#include "CoreMinimal.h"
#include "Blueprint/UserWidget.h"
#include "ABUserWidget.generated.h"

UCLASS()
class ARENABATTLE_API UABUserWidget : public UUserWidget
{
  GENERATED_BODY()

public:
  FORCEINLINE void SetOwningActor(AActor* NewOwner) { OwningActor = NewOwner; }

protected:
  UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Actor")
  TObjectPtr<AActor> OwningActor;
}
```

이렇게하여, 앞선 UABWidgetComponent의 `InitWidget()` 단계에서 소유자의 정보를 불러왔으니 UABUserWidget은 `NativeConstruct()` 단계에서 소유자 정보를 사용할 수 있게 된다.

📌 **Summary**

- 액터 컴포넌트를 사용해 기능을 분산
- 델리게이트와 인터페이스를 사용한 느슨한 결합
- 컴포넌트 생명주기의 이해

0n. 캐릭터 스탯과 위젯 강의 과제

❓ Q1. 현재 프로젝트에서 느슨한 결합 구현을 위해 델리게이트를 활용하는(아니면 앞으로 활용할) 사례를 정리해보시오.

플레이어 캐릭터의 BuildComponent는 건물의 생성을 담당한다.

그러나 컴포넌트 내에서 빌드할 건물에 대한 헤더를 소유하는 것은 상호의존성을 높이게 될 것이다.

→ 그렇다면 BuildManager라는 별도의 액터 클래스를 생성하고, BuildComponent에서 BuildStart라는 이벤트가 트리거 될 때, BuildManager의 생성로직에 바인드하여 해당 건물 액터를 생성하게 설계한다면 느슨한 결합 구조를 구현할 수 있을 것이다.

❓ Q2. 충돌 채널 설정에서 오브젝트 타입에 ABPawn이라는 타입을 새롭게 추가해 다른 폰의 반응은 무시하고 오로지 ABCharacter의 캡슐에만 반응하도록 설정하고자 한다. 이를 위해 추가해야 할 충돌 프로필을 생각해보시오.

ABPawn오브젝트 타입을 만들때 Default 반응을 Ignore로 해주고, 프로필을 생성할 때 ABCharacter란만 overlap 또는 block으로 설정해준다.