1단계_09. C++설계 #3 델리게이트

수강일	@2023/04/02
이름	전영재
멘토	Min-Kang Song,현웅 최
작성 상태	In progress
단계	1단계
강의 시청 여부	☑

Contents

▼

- C++설계 #3 델리게이트
- [1] 강한 결합과 느슨한 결합
- [2] 델리게이트
  - 델리게이트 선언
  - 페이로드 데이터
- [3] 델리게이트 구현 실습
- 0n. C++설계 #3 델리게이트 강의 과제

C++설계 #3 델리게이트

💡 **강의 목표**

- 느슨한 결합을 이해하고 이를 위한 델리게이트의 이해
- 발행-구독 디자인 패턴의 이해
- 언리얼 델리게이트를 활용한 클래스 간의 느슨한 결합 설계와 구현

---

[1] 강한 결합과 느슨한 결합

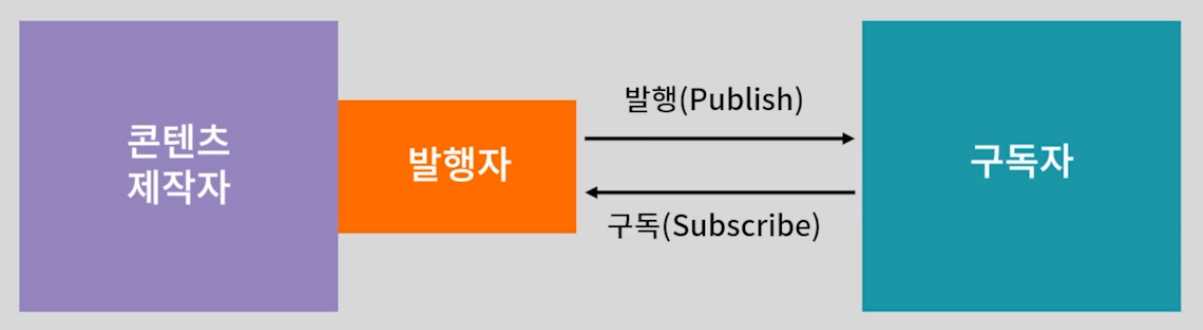
강한 결합	클래스들이 서로 의존성을 가지는 경우
느슨한 결합	실물에 의존하지 않고 추상적 설계에 의존하는 경우

언리얼 C++은 델리게이트를 지원하기에, 이를 통해 함수를 마치 오브젝트 처럼 다룰 수 있다.

---

[2] 델리게이트

발행-구독 디자인 패턴을 이용하여 콘텐츠 제작자는 온전히 제작에만, 구독자는 온전히 소비에만 집중하여 느슨한 결합 구조를 만들 수 있다.



델리게이트의 인자를 통해 값을 전달할 수 있는데, 보통 추천하지는 않는다. Heap에 메모리를 할당해야 하기 때문이며, 그럼에도 필요하다면 항상 참조를 통해 전달해야 한다.

델리게이트 선언

1대1 대응이냐, 1대다냐, 블루프린트랑 연동할것이냐 와 같은 추가옵션을 지정가능하다.

- `DECLARE_MULTICAST_DELEGATE…`
- `DECLARE_DYNAMIC_DELEGATE…`
- `DECLARE_DYNAMIC_MULTICAST_DELEGATE…`

> ⇒DECLARE_{델리게이트 유형}DELEGATE {함수정보}

```
C++에서만 사용할 것이고(Dynamic 사용x), 다수인원에게 발송하고(Multicast), 2개의 인자를가진다면(TwoParams)
DECLARE_MULTICAST_DELEGATE_TwoParams(FCourseInfoOnChangedSignature, const FString&, const FString&)
```

델리게이트 바인딩

델리게이트에 함수를 바인드해주는 방법에도 여러가지가 있다. 다음을 보고 어떤 차이점이 있는지 알아보자.

`Bind()`	일반적인 바인드
`BindStatic()`	raw C++ 포인터 글로벌 함수 델리게이트를 바인드
`BindRaw()`	raw C++포인터 델리게이트에 바인드. raw포인터는 어떠한 참조도 사용하지 않기에 오브젝트가 델리게이트 치하에서 삭제될 경우, 호출하기가 안전하지 않을 수 있다.
`BindSP()`	공유 포인터-기반멤버 함수 델리게이트에 바인드. 공유포인트 델리게이터는 객체로의 약한 레퍼런스를 유지한다. ExecuteIfBound()의 사용이 가능하다.
`BindUObject()`	UObject기반 멤버함수 델리게이트 바인드. 약한 레퍼런스를 유지하며 ExecuteIfBound()를 사용 가능하다.
`UnBind()`	델리게이트 바인드를 해제한다.

경험상, `Bind()`와 B`indUObject()`를 자주 사용했다.

페이로드 데이터

쉽게 말해 델리게이트를 바인드 할 때, 임의의 변수를 함께 저장한다는 뜻으로, 페이로드 데이터란 바인딩 된 함수를 불러낼 때 직접 전해지는 임의의 변수를 의미한다.

다음 예제의 경우, 델리게이트를 불러내면 bool과 int32 데이터가 바인딩 된 함수에 전달된다.

```
MyDelegate.BindRaw( &MyFunction, true, 20 );
```

[3] 델리게이트 구현 실습

다음을 유의하며 델리게이트를 설계해보자.

- 하나의 클래스는 하나의 작업에만 집중하도록 설계
  - 학사정보는 델리게이트 선언과 알림에만 집중
  - 학생은 알림의 수신에만 집중
- 학사정보와 학생은 서로 헤더를 참조하지 않도록 신경쓸 것
- 이를 위해 발행-구독을 컨트롤하는 주체가 필요하다.
  - MyGameInstance에서 델리게이트의 구독과 알림을 컨트롤 할 것이다.

학사정보에서 델리게이트를 선언하는 것과 게임인스턴스에서 학사정보와 학생을 연결시켜주는 모습을 살펴보자.

(학생의 코드는 그리 중요하진 않다.)

```
CourseInfo.h

DECLARE_MULTICAST_DELEGATE_TwoParams(FCourseInfoOnChangedSignature, const FString&, const FString&)
UCLASS()
class ...API ...
{
Public:

  FCourseInfoOnChangedSignature OnChanged;

  void ChangeCourseInfo(const FString& InSchoolName, const FString& InNewContents);
}
```

```
CourseInfo.cpp

void UCourseInfo::ChangeCourseInfo(const FString& InSchoolName, const FString& InNewContents)
[
  UE_LOG(LogTemp, Log, TEXT("[CourseInfo] 학사정보가 변경되어 알림을 발송합니다."));
  Contents = InNewContents;
  OnChanged.BroadCast(InSChoolName, Contents); //OnChanged에 연결된 모든 함수들에게 방송
}
```

```
Student.h
UCLASS()
{
  ...
public:
  void GetNotification(const FString& School, const FString& NewCourseInfo)
}
```

```
Student.cpp

void UStudent::GetNotification(const FString& School, const FString& NewCourseInfo)
{
  UE_LOG(LogTemp, Log, TEXT("[Student] %s로부터 메시지 수신: %s"), *School, *NewCourseInfo);
}
```

```
MyGameInstance.h
UCLASS()
{
  ...
  private:
  UPROPERTY()
```

```
  TObjectPtr<class UCourseInfo> CourseInfo;
}
```

```
MyGameInstance.cpp
#include "CourseInfo.h"
...
//사실 CDO에 생성해도 되지만, 지금은 필요시에 호출함을 보일수 있게 Init에다 구현한다.
void UMyGameInstance::Init()
{
  Super::Init();
  CourseInfo = NewObject<UCourseInfo>(this); //CourseInfo의 아우터를 MyGameInstance로 지정해줌으로써, 컴포지션 관계를 만든다
                                             //특별한 일이 없으면, 메모리 내에 유지되야 하므로 아우터를 지정했다.

  UStudent* Student1 = NewObject<UStudent>(); // 실습용으로 한번 작동하고 사라져도 되는 오브젝트니 아우터를 안해줬다.
  UStudent* Student2 = NewObject<UStudent>();

  CourseInfo->OnChanged.AddUObject(Student1, &UStudent::GetNotification) //CourseInfo의 OnChanged라는 함수에 Student객체들을 연결
  CourseInfo->OnChanged.AddUObject(Student2, &UStudent::GetNotification)
  CourseInfo->ChangeCourseInfo(SchoolName, TEXT("변경된 학사 정보"));
}
```

위 코드 진행 시, 다음과 같은 결과가 나온다.

> [CourseInfo] 학사정보가 변경되어 알림을 발송합니다.
>
> [Student] 학교로부터 메시지 수신: 변경된 학사 정보 // 참고로 여긴 Student2
>
> [Student] 학교로부터 메시지 수신: 변경된 학사 정보 // 여긴 Student1에서 나온 텍스트다.
>
> // 바인드의 순서는 스택형식이라는걸 예상할 수 있다.

위 예제를 통해, CourseInfo와 Student는 서로를 include 하지 않았음에도 상호작용 할 수 있었다.
이것이 곧 느슨한 결합을 사용 예시이다.

📌 **Summary**

- 느슨한 결합으로 구현된 발행-구독 모델의 장점
  - 자신은 외부의 변경사항에 대해 영향받지 않는다.
  - 자신의 변경사항이 외부에 영향을 주지 않는다.
- 언리얼 C++ 델리게이트의 선언 방법과 활용
  - 몇개의 인자를 가지는지? (Param 갯수)
  - 어떤 방식으로 동작하는지? (MULTICAST 사용 유무)
  - 언리얼 에디터와 연동할 것인지? (DYNAMIC 사용 유무)

    ⇒위를 조합해 적합한 매크로를 선택하면 된다.

0n. C++설계 #3 델리게이트 강의 과제

❓ **Q1. 언리얼 델리게이트를 활용한 예제를 고안하고 직접 구현해보자.**