1단계_11. UCL #2 구조체와 Map

수강일	@2023/04/03
이름	전영재
멘토	Min-Kang Song,현웅 최
작성 상태	Done
단계	1단계
강의 시청 여부	☑

Contents

UCL #2 구조체와 Map

💡 **강의 목표**

- 언리얼 구조체의 선언과 특징 이해
- UCL 중 TMap의 내부구조 이해
- TArray, TSet, TMap의 장단점을 파악하고 알맞게 활용하는 방법 학습

---

[1] 언리얼 구조체

프로퍼티를 체계화 및 조작할 수 있는 **데이터 구조**를 말하며 데이터 저장/전송에 특화되어 있다.

- 대부분 `GENERATED_BODY` 매크로를 선언해 리플렉션. 직렬화 같은 언리얼 특유의 기능을 지원하도록 설정할 수 있다.

    - `GENERATED_BODY`매크로를 선언한 구조체는 엄밀히 말하면 `UScriptStruct`클래스로 구현된다.
    - 제한적으로 리플렉션을 지원해 `UPROPERTY`는 지원하지만 `UFUNCTION`은 지원하지 않는다.
- 가볍게 데이터를 다루기 위함이 목표이므로, 대부분 힙 메모리 할당(포인터 연산) 없이 스**택 내 데이터**로 사용된다.

구현방법

1. 구조체를 정의하려는 헤더를(.h)를 열고, C++구조체 앞에 `USTRUCT`매크로를 추가한다.
2. 구조체 상단에 `GENERATED_BODY`를 추가하고, 필요한 멤버변수에 `UPROPERTY`를 태그하면 된다.

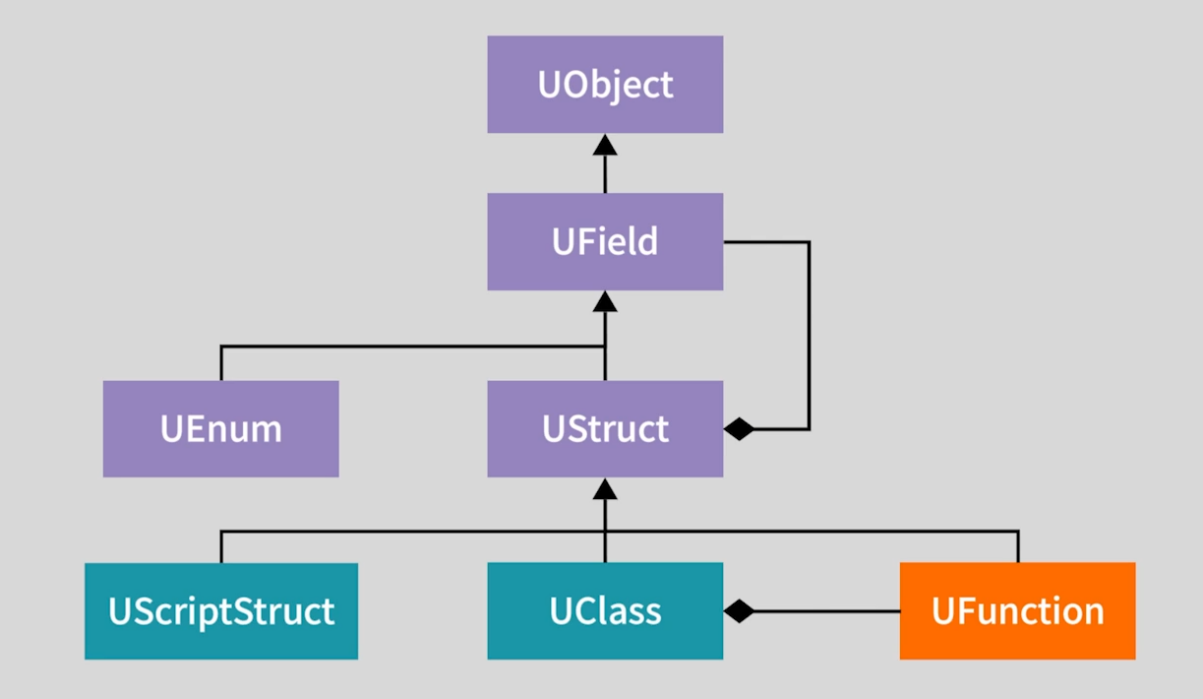
```
USTRUCT(BlueprintType) //에디터에서도 쓸수있게 해주는 메타데이터
struct FMyStruct //언리얼 오브젝트가 아니라 F접두사 추가
{
  GENERATED_BODY()

  UPROPERTY(EditAnywhere, BlueprintReadWrite) //리플랙션을 위해 UPROPERTY 태그
  int32 MyIntegerMeber;
}
```

사실 C++문법상 구조체와 클래스는 큰차이가 없는데(기본접근자가 public이냐 private냐 정돈데..) **언리얼에선 `UObject`와 `UStruct`는 사용 용도가 완전히 다르다.**

> `UStruct`는 단순한 데이터 타입에 적합하므로, **복잡한 인터랙션**을 위해선 차라리 `UObject`를 쓰는 걸 권장한다.

언리얼 리플렉션 관련 계층 구조



언리얼은 리플렉션 기능을 사용하기 위해서 위와 같은 계층 구조를 설계했다.

UField를 상속받은 USturct는 또다른 UField객체를 소유(Composition)하여 사용하는 구조로 설계되었고, 이러한 USturct를 상속받은 UScriptStruct, UClass가 존재하는데, 이 중 UClass는 USturct를 상속받아 만들어진 UFunction을 소유하기에 결과적으론 UField를 소유한다고 볼 수 있다.

⇒결국, UClass와 USturct는 구조와 사용법이 다르다는 거다.

실습

구조체를 별도의 헤더파일을 만들지않고, GameInstance헤더에 넣어서 만들수도 있다.

```
MyGameInstance.h
USTRUCT()
struct FStudentData
{
  GENERATED_BODY()

  //기본적으로 Struct는 public접근자를 갖기에 따로 안써줘도되더라.
  FStudentData()
  {
    ...
  }

  //구조체는 UObject가 아니기에, New api를 통해 생성되지 않으므로, 인자를 가진 생성자를 만들어 자유롭게 사용하겠다.
  FStudentData(FString InName, int32 InOrder) : Name(InName), Order(InOrder) {}

  UPROPERTY() //안써도됨
  FString Name;
  UPROPERTY()
  int32 Order
}

UCLASS()
class UNREALCONTAINER_API UMyGameInstance : public UGameInstance
{
  GENERATED_BODY()

public:

  virtual void Init() override;

private:
  //구조체 FStudentData를 담는 TArray
  TArray<FStudentData> StudentsData;
  //UObject UStudent를 담는 TArray, UObject를 관리하기에 반드시 UPROPERTY를 선언해준다.
  UPROPERTY()
  TArray<TObjectPtr<class UStudent>> Students;
};
```

```
void UMyGameInstance::Init()
{
  ...
  const int32 StudentNum = 300;
  for (int32 ix = 1; ix <= StudentNum; ++ix)
  {
    StudentsData.Emplace(FStudentData(MakeRandomName(), ix)); //구조체를 배열에 넣을것이기에 메모리사용이 많이 생길 것이므로 Emplace사용
  }

  TArray<FString> AllStudentsNames;
  Algo::Transform(StudentsData, AllStudentsNames,
    [](const FStudentData& Val)
    {
      return Val.Name;
    }
  );

  UE_LOG(LogTemp, Log, TEXT("모든 학생 이름의 수 : %d"), AllStudentsNames.Num());
  ...
}
```

[3] TMap 구조와 활용

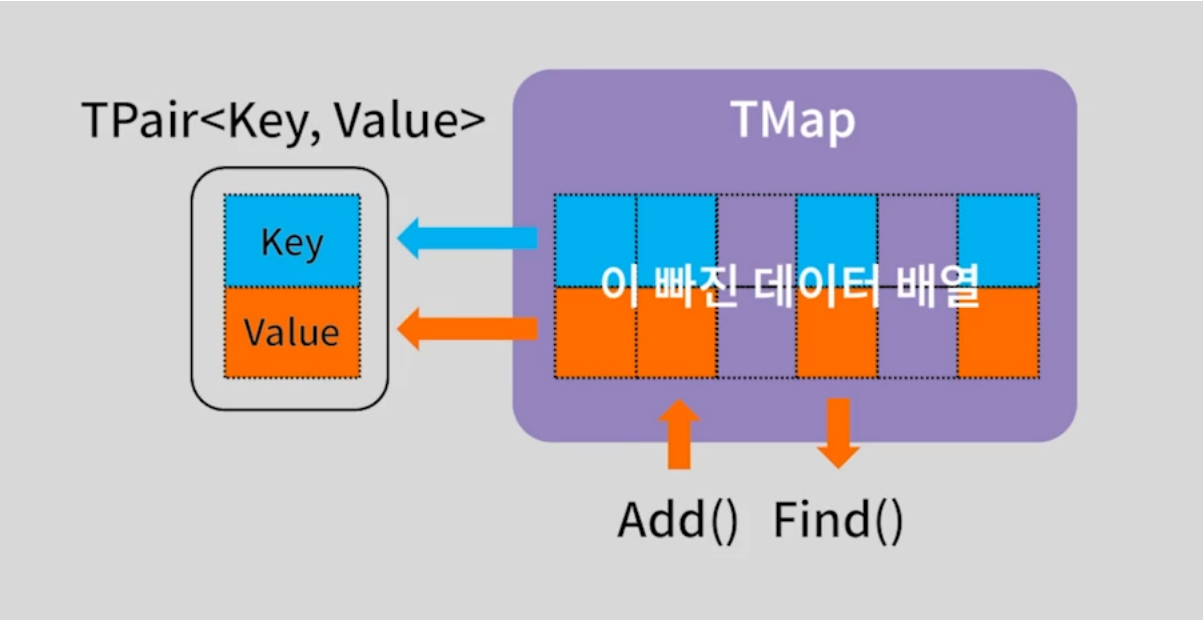
STL Map과 TMap의 비교

<STL Map>

- 이진트리 구성
- 정렬은 지원하지만 메모리 구성이 효율적이지 않음. 데이터 삭제시 정렬 재구축
- 자료 순회에 적합 X

<TMap>

- 키-밸류 구성의 Tuple데이터의 **TSet구조**임
  - **다시말해, 튜플쓰는 TSet이다**
- 그래서 내부구조랑 메카니즘이 TSet랑 똑같음
- 해시테이블 - 동적배열 - 빠른순회 - 빠른검색- 삭제후 재구축안함 - 비어있는데이터
- TMultiMap을 사용하면 중복데이터도 관리할 수 있다.
- 오히려 STL unordered_map과 유사함



TSet과 달리 TMap은 TPair라는 튜플을 채택하는 것이 다르다.

TMap의 특징

- `TArray`다음으로 많이 쓰인다.
- `TMap`에 중복된 키저장을 넣으면 기존값이 대체되고, `TMultiMap`에 넣으면 새로 추가한다.
- 키-값 쌍을 마치 **오브젝트**처럼 `Map`의 엘리먼트 유형으로 정의한다.
- 해시 컨테이너라 GetTypeHash 함수를 통해 키유형에서 해시를 뽑을 **해시표**를 만들어야한다.

생성방법

```
TMap<int32, FString> FruitMap;
```

쿼리, 제거, 소팅, 연산자, 슬랙

> 그냥 키-밸류 인 TSet인지라 전반적인 함수 구성이 거의 똑같다.

KeyFuncs

이게 좀 중요하다. 저번에 TSet에서 안다뤘던 그것임.

해시테이블 구조이기에 커스텀 구조체의 '`operator ==`'과 '`GetTypeHash`'를 만들어줘야 한다.

TMap

TMap, 맵은 크게 키 유형과 값 유형, 두 가지로 정의되며, 맵에 하나의 짝으로 저장됩니다.

https://docs.unrealengine.com/5.0/ko/map-containers-in-unreal-engine/

```
struct FMyStruct
{
    // String which identifies our key
    FString UniqueID;

    // Some state which doesn't affect struct identity
    float SomeFloat;

    explicit FMyStruct(float InFloat)
        : UniqueID (FGuid::NewGuid().ToString())
        , SomeFloat(InFloat)
    {
    }
};
template <typename ValueType>
struct TMyStructMapKeyFuncs :
    BaseKeyFuncs<
        TPair<FMyStruct, ValueType>,
        FString
    >
{
private:
    typedef BaseKeyFuncs<
        TPair<FMyStruct, ValueType>,
        FString
    > Super;

public:
    typedef typename Super::ElementInitType ElementInitType;
    typedef typename Super::KeyInitType     KeyInitType;

    static KeyInitType GetSetKey(ElementInitType Element)
    {
        return Element.Key.UniqueID;
    }

    static bool Matches(KeyInitType A, KeyInitType B)
    {
        return A.Compare(B, ESearchCase::CaseSensitive) == 0;
    }

    static uint32 GetKeyHash(KeyInitType Key)
    {
        return FCrc::StrCrc32(*Key);
    }
};
```

특이한 경우?

구조체 생성시 키값을 uniqueID로 만들게 설정한 경우, **==는 어떻게 만들것이냐?**

아무리 밸류가 같아도 키값이 다르면 ==를 사용해 정의하기 어렵기에, 이것을 위해 더욱 구체적으로 명시하거나 특수조치가 필요하다는 것.

이를 위해서는 `MapKeyFuncs`를 구현해야함. 이걸 해결하려면 문서를 참고해라.

사실 이런경우가 얼마나 있을진 모르겠다.

실습

- TMap 만들고, 기존 TArray로 만들었던 데이터를 TMap에 넣는다.
  - 넣을때는 Algo::Transform을 사용해 람다 만들어줘서 넣었다.
    (튜플로 넣어야 되니까 람다 만들때 return값을 2개로 만들었다.)

```
void UMyGameInstance::Init()
{
  ...
  Algo::Transform(StudentsData, StudentsMap,
    [](const FStudentData& Val)
    {
      return TPair<int32, FString>(Val.Order, Val.Name);
    }
  );
  UE_LOG(LogTemp, Log, TEXT("순번에 따른 학생 맵의 레코드 수 : %d"), StudentsMap.Num());

  TMultiMap<FString, int32> StudentMapByName;
  Algo::Transform(StudentsData, StudentMapByName,
    [](const FStudentData& Val)
    {
      return TPair<FString, int32>(Val.Name, Val.Order);
    }
  );
  UE_LOG(LogTemp, Log, TEXT("이름에 따른 학생 멀티맵의 레코드 수 : %d"), StudentMapByName.Num());
  ...
};
```

좀 중요한 실습!

`FStudentData`를 `TSet`에서 쓰려면 어떤 작업을 거쳐야 하는지 살펴보자. (커스텀 구조체 사용하기)

```
TSet<FStudentData> StudentSet;
/** 그냥 하면 GetTypeHash가 없다고 에러가 엄청 나온다. 해결하려면 operator와 GetTypeHash를 만들어야함.*/
for (int32 ix=1; ix<=StudentNum; ++ix)
{
  StiudentSet.Emplace(FStudentData(MakeRandomName(),ix));
}
```

해결하려면, `FStudentData` 구조체에 `operator==`와 `GetTypeHash()`를 추가한다.

```
struct FStudentData
{
  GENERATED_BODY()
  ...
  bool operator==(const FStudentData* InOther) const
  {
    return Order == InOther.Order; //여길 내맘대로 커스텀
  }

 //전역함수로 만들던가 friend로 만들어도 깔끔하다.
  friend FORCEINLINE uint32 GetTypeHash(const FStudentData& InStudentData)
  {
    return GetTypeHash(InStudentData.Order); //order를 곧 해쉬값으로 쓰겠다.
  }
}
```

---

[4] 자료구조 간 비교

	TArray	TSet	TMap	TMultiMap
접근	O(1)	O(1)	O(1)	O(1)
검색	O(N)	O(1)	O(1)	O(1)
삽입	O(N)	O(1)	O(1)	O(1)
삭제	O(N)	O(1)	O(1)	O(1)
	빈틈없는 메모리 가장 높은 접근성능 가장 높은 순회성능	빠른 중복 감지	중복 불허 키 밸류 관리	중복 허용 키 밸류 관리

TArray - TMap - TSet 순으로 많이 쓰인다.

📌 **Summary**

- TArray, TSet, TMap의 내부구조와 활용 방법 학습
- 언리얼 구조체의 선언 방법
- TSet과 TMap에서 언리얼 구조체를 사용하기 위해 필요한 함수의 선언과 구현 방법 학습

0n. UCL #2 구조체와 Map 강의 과제

❓ **Q1. 현재 프로젝트에서 구조체, Array, Set, Map, MultiMap이 유용하게 활용될 수 있는 사례를 생각해보시오.**

타워디펜스형 게임으로 몰려오는 적들 중 누구를 먼저 공격할지 판단해야 한다.

⇒ 자신이 인식한 적 포인터를 자료구조 내에 저장하고 **우선순위**를 결정해야 한다. 가장 먼저 발견했다는 것이 핵심이므로 **순서를 자동으로 정리**해주는 `TArray` 자료구조를 사용하는 것이 유용할 것으로 예상된다. 배열 내 첫번째 인덱스에 해당하는 객체를 공격하는 식으로 설계한다.

범위 공격을 가할때, 대상에 대해 어떻게 인식하고 다룰것인지?

⇒ 범위무기 앞으로 `SweepMulti` 를 사용하여, 일정 범위 내에 어떤 액터들이 있는지 파악한다.

그러나 라인트레이스의 결과로 하나의 객체가 **여러번 검출**되는 문제가 있었다. 그렇기에 이들을 **중복을 허용하지 않는** `TSet` 자료구조에 넣어 한번 **정제**해준다. 배열 내에 삽입되는 순서는 중요치 않기에 속도를 중요시하고 중복을 불허하는 TSet이 가장 우수할 것으로 예상된다.