序列化

一、什么是UE4的序列化

对游戏中某些核心逻辑的代码做一个快照，可以保存到硬盘，可以上传到服务器。

二、UE4如何实现的序列化

1，总览

UE4本身就写一套序列化，UObject身上带有一个Serialize()方法， 负责对整个类里面的某些信息做序列化，被UPROPERTY()宏标记的属性，一般都是会被序列化的。

序列化到磁盘之后，UE4将序列化的二进制数据以.uasset后缀的文件保存起来，使用LoadObject可以重新将uasset文件反序列化成UObject

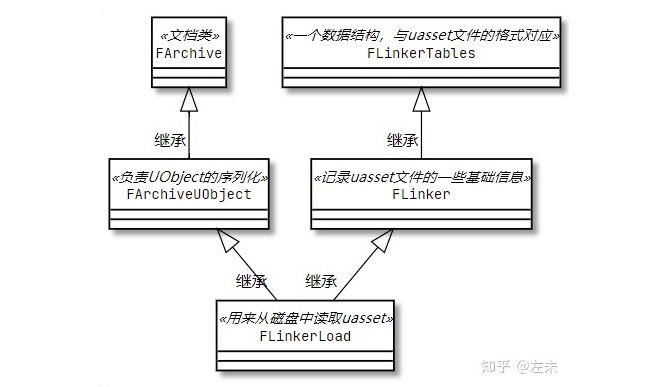
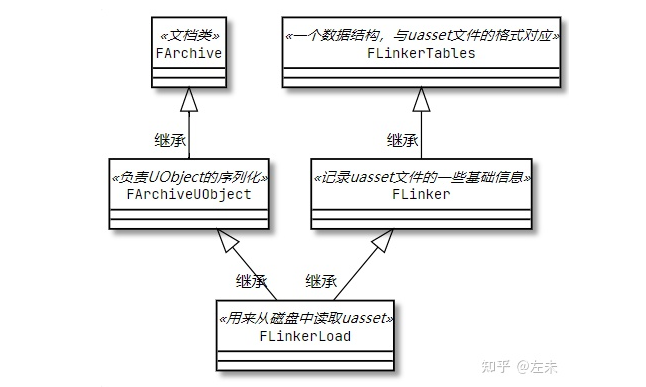
UE4中UObject的序列化和反序列化都对应函数Serialize，通过传递进来的FArchive的类型不同而作不同的操作

UE4中的属性分为普通C++变量和uproperty两个部分，普通的C++变量只能在Runtime的时候进行读写，所以不需要进行序列化，而Property可以在通过UE4Editor的蓝图进行修改操作，一般来说会进行序列化（被标记了Transient的变量因为不需要存储在硬盘里所以不会进行序列化）

2，UE4的序列化流程详解

一些基本概念：

UPackage:可以当做是UAsset文件在内存中的表示



FArchive

即文档类，序列化和反序列化都是依靠他完成，FArchive有各式各样数量繁多的派生类，其中复制则把UObject序列化到磁盘和从磁盘反序列化生成UObject实例的分别是FLinkerSave和FLinkerLoad

Linker家族

FLinkerTable的结构与uasset文件的内容是一一对应的，也就是说，当uasset被加载到内存时，查看LinkerTable的内容就知道uasset里面究竟是什么内容。

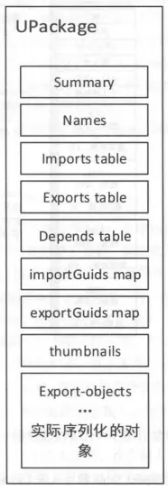
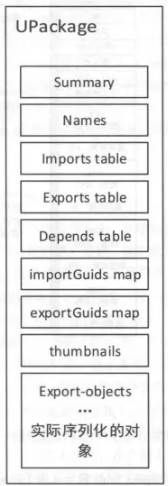
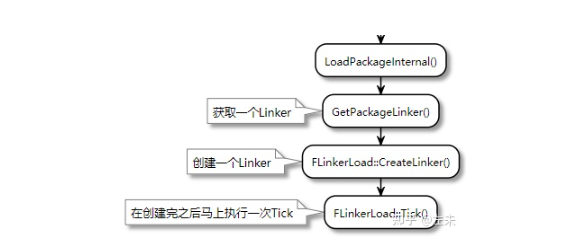
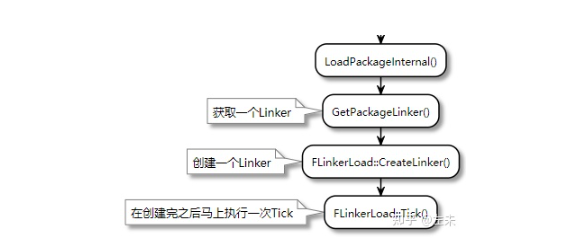
FLinker这个类是FLinkerLoad和FLinkerSave的基类，然后就是加载uasset的主力军FLinkerLoad

工作流程：

FLinkerLoad的加载流程：

UMyClass**\*** LoadedInstance **=** LoadObject**<**UMYClass**>**(**nullptr**, TEXT("/Game/xxx.xxx"));

跟踪调用链会发现最后其实是会Load一个Package



LoadPackageIntenal()中会根据路径创建一个对应的FLinkerLoad，它被创建完后会马上执行自身的Tick()， tick其实就是在一点点地加载uasset的内容进来，就是将上图里面的每个部分都读取到Linker当中

三、UE4序列化的作用

四、UE4序列化的使用

FString PackageName = "FirstPackage";

FString PackagePath = "/Game/blueprints/" + PackageName;

UPackage\* MeshPackage = CreatePackage(nullptr, \*PackagePath);

ARenderLeft\* StaticMesh = NewObject< ARenderLeft >(MeshPackage, FName(\*PackageName), RF\_Public | RF\_Standalone);

FString PackageFileName = FPackageName::LongPackageNameToFilename(PackagePath, FPackageName::GetAssetPackageExtension());

bool bSuccess = UPackage::SavePackage(MeshPackage, StaticMesh, RF\_Public | RF\_Standalone, \*PackageFileName, GError, nullptr, true, true, SAVE\_NoError);

上述代码可以将UObject序列化到磁盘。

Uobject::Serialize()方法

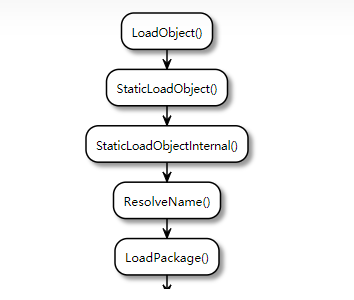
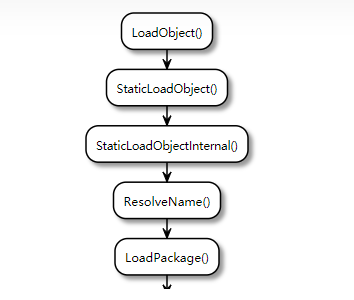
**virtual** **void** **Serialize**(FArchive**&** Ar);

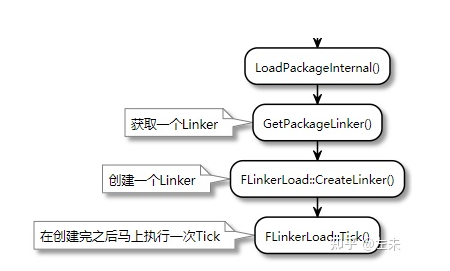
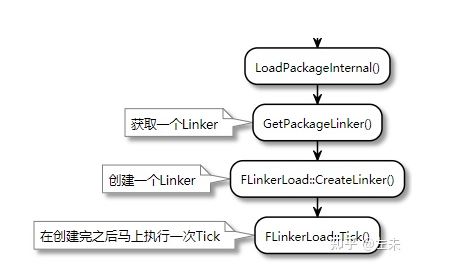
**virtual** **void** **Serialize**(FStructuredArchive**::**FRecord Record);

第二个版本才是主角

LinkerLoad的大致工作流程

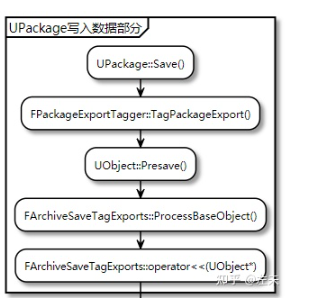
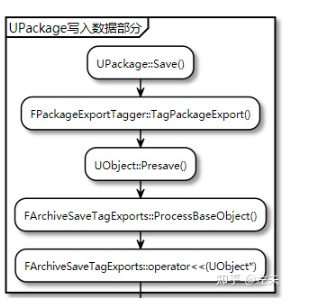
LoadObject()的源码的调用流程会去调用一个LoadPackage()，然后LoadPackageInternal()内部会创建一个Linker（FLinkerLoad::），Linker会马上执行一次Tick()，





Tick里面其实就是一点点地加载uasset的内容进来。

关于序列化的基础知识



void FArchiveSaveTagExports::ProcessBaseObject(UObject\* BaseObject)

{

    (\*this) << BaseObject;  //调用Operator<< 将BaseObject放入TaggedObjects数据

    ProcessTaggedObjects(); //处理TaggedObjects，执行序列化！

}

FArchive& FArchiveSaveTagExports::operator<<(UObject\*& Obj){

if (!Obj || Obj->HasAnyMarks(OBJECTMARK\_TagExp) || Obj->HasAnyFlags(RF\_Transient) || !Obj->IsInPackage(Outer))

{

    return \*this;

}}

//上述函数可以看出，如果在序列化时Obj没有满足以上条件，将无法正常序列化。

Obj->IsInPackage(Outer)  //Obj所在的包是否是我们要保存的那个包，如果我们要保存的UObject所在的包不是我们要保存的那个包，就不能进行序列化！

UPackage::SavePackage()可以进行序列化，最终会进行到上述流程，一般来说负责序列化的FArchive类型为 FArchiveSaveTagExports