前言

序列化测试一定要注意保存的场景中的Actor，如果修改了序列化的加载/保存函数，则重新打开时会加载数据错误。

## 尝试进行自定义的序列化保存

<https://blog.csdn.net/mohuak/article/details/83027211>

<https://wiki.unrealengine.com/Save_System,_Read_%26_Write_Any_Data_to_Compressed_Binary_Files>

<http://runedegroot.com/saving-and-loading-actor-data-in-unreal-engine-4/#comment-93>

主要是参照第三个

首先定义一个接口，用来进行基础的ActorSaveDataLoaded以及ActorSaveDataSaved();

创建一个Struct用来存储Actor的基本序列化数据

再创建一个整个游戏的用来存储序列化数据的数据结构。

最后，最重要的一步，创建一个继承自FObjectAndNameAsStringProxyArchive的struct用来保存Actor的数据！只有继承自该结构的Ar才可以正确进行Actor的序列化！

UGameplayStatics::GetAllActorsWithInterface(GetWorld(), USaveableActorInterface::StaticClass(), Actors);

可以获取到所有继承了某个接口的Actor

SerializeScriptProperties,跟进查看对于标记UPROPERTY的参数是如何进行反序列化的:

调用SerializeTaggedProperties，序列化对象属性，并且加入tag;

最终跟进是通过 PropertyTag的

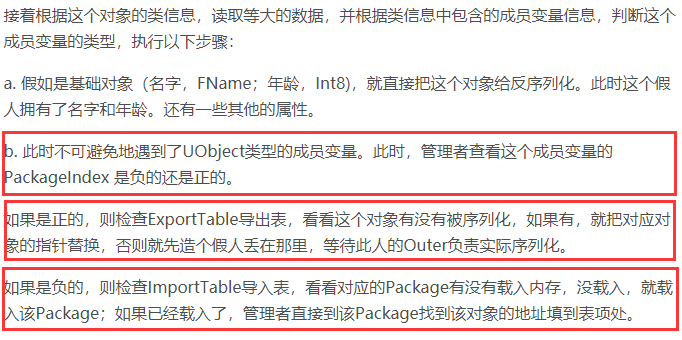
FPropertyTag::SerializeTaggedProperty(FStructuredArchive::FSlot Slot, UProperty\* Property, uint8\* Value, uint8\* Defaults) const

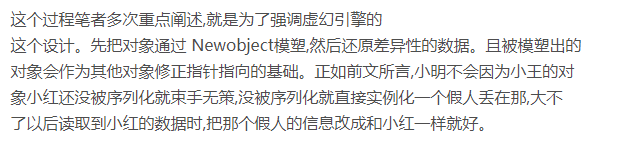
函数，将UProperty（测试时是看的UStaticMeshComponent\* GroundMeshComponet）调用

Property->SerializeItem(Slot, Value, Defaults); 进行初始化。

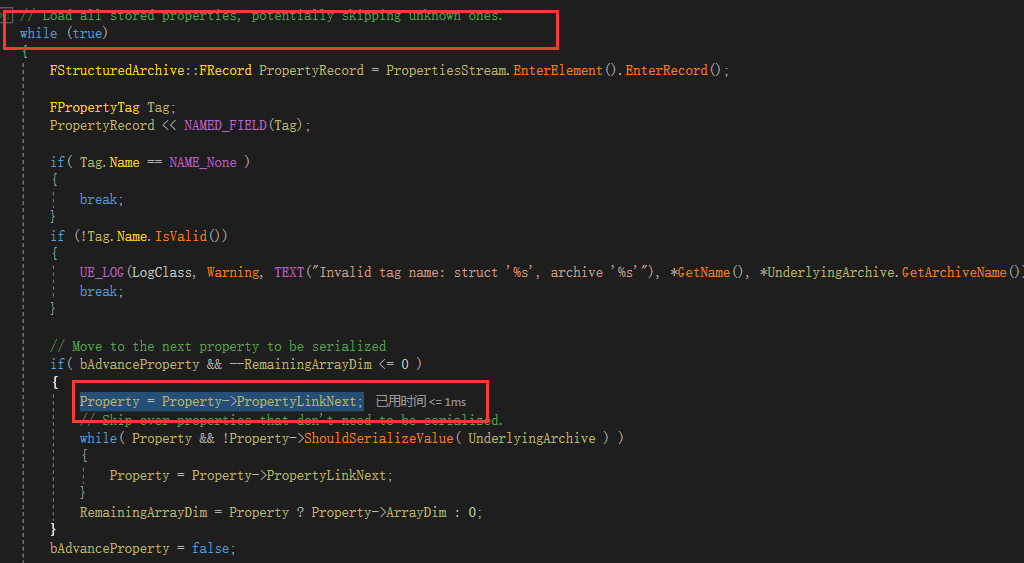
void UObjectProperty::SerializeItem( FStructuredArchive::FSlot Slot, void\* Value, void const\* Defaults ) const

测试时看到，因为之前序列化保存的GroundGridMgrComponent还没有反序列化初始化出来，所以GroundObj中的GroundGridMgrComponent并没有进行完整的赋值，只是创建了一个默认的GroundGridMgrComponent并将指针指向，在正式的GroundGridMgrComponent反序列化之后，才会将该指针指向的结构初始化（还是替换为正确的？）生动的说明在此：





当一个UPROPERTY属性初始化完成之后，



通过While循环会将指针指向Next进行初始化。

如果是UStruct，则会通过

UStructProperty::SerializeItem(FStructuredArchive::FSlot Slot, void\* Value, void const\* Defaults) const

进行初始化

保存时序列化也是走类似的流程。

如果该对象从来没有被加载过，则包内不会出现该对象，此时应该手动加载。

### 关于组件好像并没有跟着Actor序列化的问题：

看一下Level.cpp可以看到，是Level在serialize中对各个Actor做了自己对应的处理，才能让各个Actor保存的那么好。

主题医院的测试中，list类型的序列化，严重依赖于序列化的先后顺序，如果要仿造UE4的结构进行序列化的处理，可以预计会十分复杂，因此在此放弃全部使用序列化存储的方式，改为进行序列化+动态生成动态链接的方式进行处理。

### 猜测：

UE自身的序列化，应该也是多次调用的！先序列化出主Actor，将指针所对应的子Object生成假人放着，当真正的子Object生成了之后，再调用自身的Serialize进行重新序列化！跟源代码看看。

场景从Serialize保存过程中可以看到，使用了FReplaceReferenceHelper的

FindAndReplaceReferences方法，会对场景中的具有引用的关系做一个保存，也就是后期反序列化时的替换依据了。FArchiveHasReferences::GetAllReferencers 在该函数中，会调用引用的对象的Serialize进行序列化.有互相引用，就会不断互相调序列化。

### 初步试验模块成功，可以正常保存以及加载继承自自定义SaveableInterface的Object了！

通过查阅文档发现，ArrayProperty，EnumProperty之类的都是存在的！！那么，就可以对Obj中的TArray等结构进行遍历处理，找到其对应的UObject并且将对应的按照一定方法保存生成refurrence！

在遍历Object的过程中发现，会有UObject的那些Property也被算进来，所以，如果想要序列化那些自己声明的引擎自身的UPROPERTY组件等，

一：自定义一套UPROPERTY的声明规则，或是用自己的Meta或是自定义Category等，在遍历的时候对应单独处理。

二：还是按照原来的处理方式，只对我们自己定义了Saveable的进行存储，可以将原生的类包一层继承我们自己的Object。这样其实更方便更实用。原生的类虽然不继承我们的Saveableinterface，但是还是可以作为正常指针附加到我们的需要引用的指针上。

**类中还是要自定义自己的Serialize后的处理方案，例如反序列化后，重新创建某些组件之类的，因为反序列化，是在OnConstruct以及构造函数之后才调用的！**

设置Property的值，必须使用SetObjectPropertyValue等系列函数，不能直接赋值指针。

## 那么问题来了，为什么不直接使用配置存储进入游戏后生成而要使用序列化？

个人想法是，用配置存储，需要根据Actor/Component的参数不同而定义不同的存储类型，存储结构，存储table等，而使用序列化的方式，则可以进行一个统一的存储结构的定义以及一个统一的初始化、替换指针引用的目的。