# 间接属性引用

可以用来加载普通的资源也可以用来加载蓝图类，分为同步加载和异步加载，如果加载类，则使用UClass指针进行资源引用

间接属性引用区别于直接属性引用，间接属性引用不会在游戏启动时将资产加载到内存，而是可以根据使用者的需求，进行加载。使用较直接属性引用稍复杂，但是可控性更强。

### FStringAssetReferences （资源拾取对象）

资源拾取器（可以拾取类资产，也可以拾取数据资产），可以直接通过属性引用的方式暴露到编辑器中，也可以通过使用设置路径进行资源拾取，此操作类加载资源区别于“直接属性引用”加载方式，此方式对于资源的引用不会主动加载，需要结合资源管理器进行加载。

**注意：使用FAR进行资源引用无法直接使用，需要进行加载操作才可以使用**

# FStreamableManager

资源加载器，提供包括同步加载和异步加载方案，使用可靠性更强，但是操作繁琐度高，理解较难。是UE中比较基础的资源管理器，操作方式是动态操作

同步加载

在加载运行线程中，阻塞线程的流程执行，将线程停止在当前加载逻辑中，加载完成后继续线程的执行逻辑操作，对于加载小资源可以保证资源的不为空，但是加载大资源将导致调用线程卡顿

异步加载

在加载线程中，不阻塞当前线程逻辑加载资源，加载器本身具备线程进行资源加载。较之同步加载更加的灵活，但是相对维护成本较高，资源加载成功后需要进行回调通知，以完成整个加载流程

构建FStreamableManager



**如果声明在头文件中，记得引入FStreamableManager头文件**

### FStringAssetReferences 加载资源

**测试启动独立窗口否则无法展现资源未加载情况**

资源引用方式有两种可选

·直接设置资源路径，借助SetPath函数（可以设置数据资产路径也可以设置类资产路径）。



·通过宏UPROPETY(EditAnyWhere)直接向编辑器暴露属性

同步加载

**注意FSA结构体本身带有TryLoad函数，此函数可以直接借助LoadObject加载资源，如果资源较大会比较慢（同步阻塞操作）**

借助StreamableManager同步加载语法如下



加载完成后可以使用如下语法进行资源转换到目标资源类型

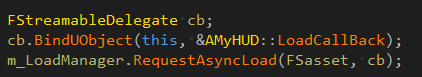


**注意函数ResolveObject可能返回空指针，操作需要额外注意**

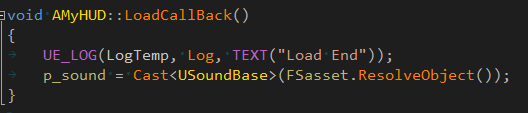
异步加载

异步加载与同步加载区别在于，异步加载会通过回调方式（**无参无返回成员函数**）进行资源加载完毕通知，所以需要构建通知多播代理

语法



加载完毕后可以进行继续的逻辑编写



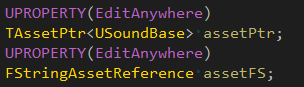
# TAssetPtr（数据资产拾取）

对FAR资源拾取器的包装对象，进行类型约束，可以使得资源拾取有范围限定，可以通过调用结构体的函数ToStringAssetReferences（）获取FAR的操作对象，调用Get（）获取模板资源对象引用

语法



暴露到编辑器语法



### TAssetPtr加载资源

借助FSreamableManager进行加载，借助函数IsPending判定是否完成加载

资源引用

·可以通过宏UPROPETY(EditAnyWhere)暴露到编辑器方式进行资源引用

·通过使用FAR进行赋值（**重载了赋值运算符**）



同步加载

和FAR方式一样，语法如下



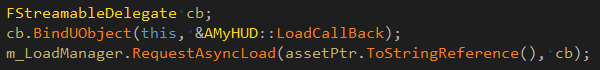
加载完成后使用

直接调用Get函数即可



异步加载

由于StreamableManage没有重载异步加载TAssetPtr的函数，所以需要借助使用函数ToStringReference转换到FAR然后再进行异步加载，语法



加载完成后使用方式

直接调用Get函数即可



注意Get函数可能返回空

# TAssetSubclassOf（类资产拾取）

同TAssetPtr操作方式相同，但是只用于拾取类资产，需要使用FStreamableMange进行加载，对FAR进行的额外封装，通过类模版的设定约束加载类内容

构建



资源引用

·可以通过宏UPROPETY(EditAnyWhere)暴露到编辑器方式进行资源引用

·通过使用FAR进行赋值（重载了**赋值运算符**）

同步加载



异步加载

加载方式同TAssetPtr相同

