静态加载

在一个模块开始运行之前，提前把该模块可能使用到的所有文件或资源加载到内存。
优势：使用简单，在运行过程中不用考虑因为加载资源而卡顿
缺点：模块启动时间很长，内存占用高（随着功能的复杂，启动时间与内存消耗是线性增长的）

总结：用户不关心实现起来是否简单，重点关心掉帧、启动时长、内存占用，所以对于稍微复杂一点的软件，都需要动态加载

动态加载

需要使用或条件允许的时候再加载资源

- 优势：

  由于启动时，只加载部分资源，启动时长自然不必担心

- 挑战：

  1.随着程序的运行，加载的模块越来越多，内存开销越来越大
  2.加载资源是从硬盘中加载的，速度很慢。机械硬盘读取速度不到200MB/s，读取2M文件就需要10ms，60帧的程序一帧只有16ms。导致每次读取大文件就掉帧
  3.使用复杂
  4.需要对资源何时加载的逻辑非常清晰，对生命周期了如指掌

- 解决方案：

  1. 对不需要使用的模块及时卸载或优化(动态销毁)
     1.1.直接关闭不使用的模块
     1.2.不适合关闭的模块，可以降低优先级，减小Tick频率(需要额外管理优先级系统)
     1.3.对于UMG，可以销毁对应的Slate页面
  2. 使用异步加载方案，不阻塞主线程
     2.1.懒加载：用到的时候直接请求异步或同步加载(即使使用异步加载也只能保证不卡死，但是依然存在加载缓慢的问题)
     面对加载缓慢时，对用户体验上的优化方案：在加载过程中播放加载动画、提示加载进度条、推出加载中可运行的小游戏
     2.2.预加载：在即将用到之前进行加载(对程序结构要求有点高，而且时机很难把握)
  3. 深入学习资产加载原理
  4. 合理设计程序结构

- 异步加载：

  在不阻塞主循环的前提条件下加载。分为在主线程或子线程异步加载，尽量推荐在子线程加载，但是请格外注意安全问题

- 同步加载：

  直接在需要用到的地方加载，加载完成之后再走下一步

总结：经过以上过程，我们对动态加载这种思路有了一个基本的认识。接下来就是把动态加载的思路应用在UE4中

挑战的1、3、4部分不是轻易就能解决的。我们只讨论加载资源的实现方式

UE4 中的资产加载方法

前置知识：加载资源的实现方法一般分为两类

- Load：从硬盘、网络等地方读取资源

  蓝图中看到的所有内容，均是UObject资产，我们所有资产都可以从硬盘或网络中读取到内存，然后解析成对应的UObject并使用

- New：创建UObject 或 new原生C++对象

  运行过程中，除了读取内存中保存好的资产，还有许多需要动态创建的资产，资产动态创建是通过UClass实现的

静态加载：

在当前资源加载时同步加载资源的方法。只有这里引用的资源加载成功时，当前才能加载成功

- 在蓝图或C++中直接引用对象，默认自动会加载对应资源
  - 由于简单方便，大量使用：UPROPERTY()标记、蓝图中硬引用、蓝图中类引用、蓝图中子widget、蓝图中使用组件等等
- 在构造函数中使用ConstructorHelpers::FObjectFinder读取资源
  - 在构造函数中加载资产不方便预览，很少使用，本项目仅1例
- 在构造函数中使用CreateDefaultSubobject创建对象，一般用于Actor组件创建
  - 在C++中创建静态组件时，使用这个方法最简单

动态加载：

- FStreamableManager 流加载，依托UE异步加载线程完成UObject资产加载
  - 官方推荐，最标准完善的资产加载方法
- NewObject/SpawnActor/CreateWidget 动态创建UObject，先同步加载资产，然后new一个对象
  - 在C++创建对象最简单的方法，给定希望创建的对象类型即可创建
- LoadObject / LoadClass 只同步加载资产对象，不new对象
  - 在C++加载资产最简单的方法，可加载任意资产
- LoadSynchronous
- 使用软指针加载资源，只是对LoadObject 的简单封装
- FFileHelper::LoadFileToArray 加载文件，手动解析成需要的类型
  - 原生的一套资源加载方法，主要用来加载一些本地文件
- 资产注册表 用于统一管理资产预加载
  - 对资产进行分类管理，在进入一个场景前，通过一个简单的方式加载下一个场景可能用到的资产

多媒体的页面动态加载

由于水平与精力有限，所有UI资源使用的都是静态加载，页面使用的是动态加载里面的懒加载(用到时再加载)，并且是同步加载的。并没有采用更好的资产注册表预加载方案。

接下来说一下我是怎么页面完成动态加载的(ADataAssetMananger)：

1. 我有一套页面管理框架，会告诉我页面启动的时机。当页面启动时，我调用FStreamableManager的方法进行资源加载。
2. 为了便于使用，我将资产加载方法进行了封装，调用ADataAssetMananger里面的4种函数均可完成资产加载。
3. 为了便于加载蓝图文件，并且通过引用关系而不是文件名称来管理资产，我使用UMediaDataAsset记录资产
4. 多媒体有许多重复利于的小控件，我提供了控件缓冲池
5. 多媒体有许多控件都是通过Ulistview与UWrapBox加载的，我对这两个控件的使用做了更多的封装