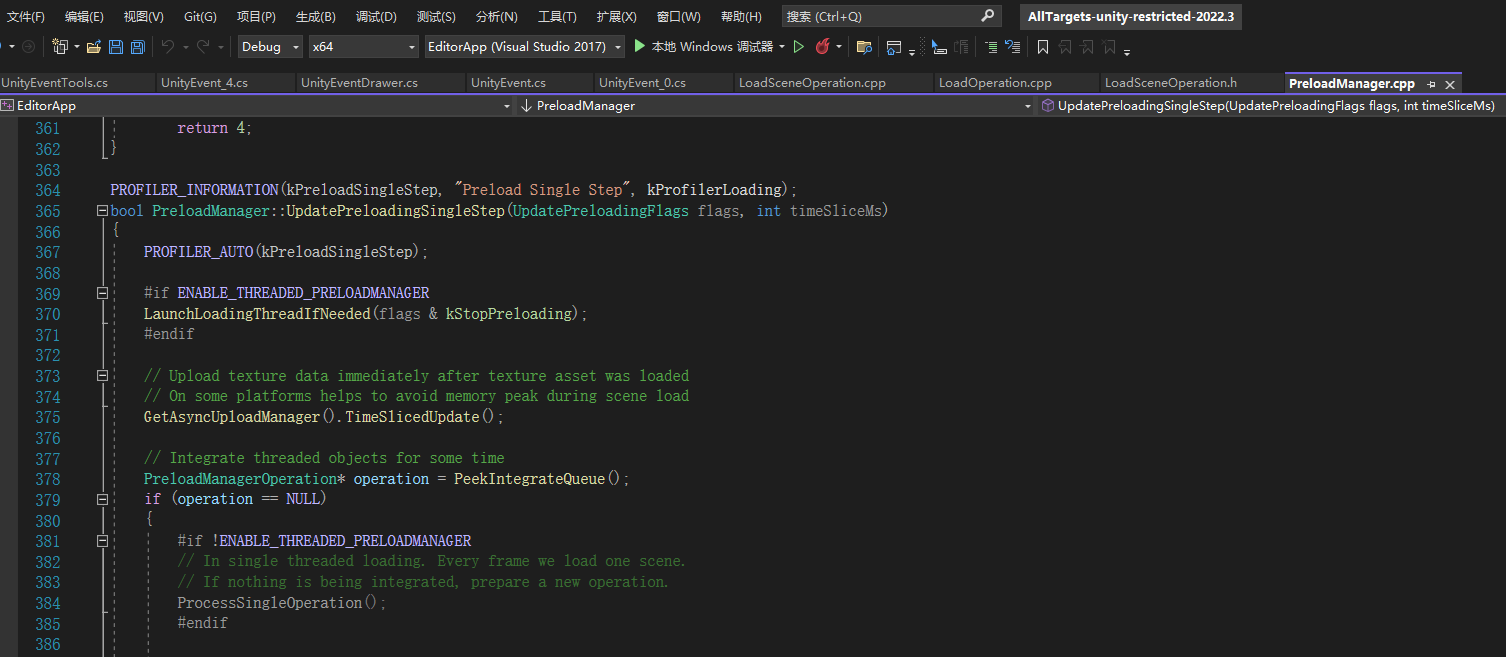
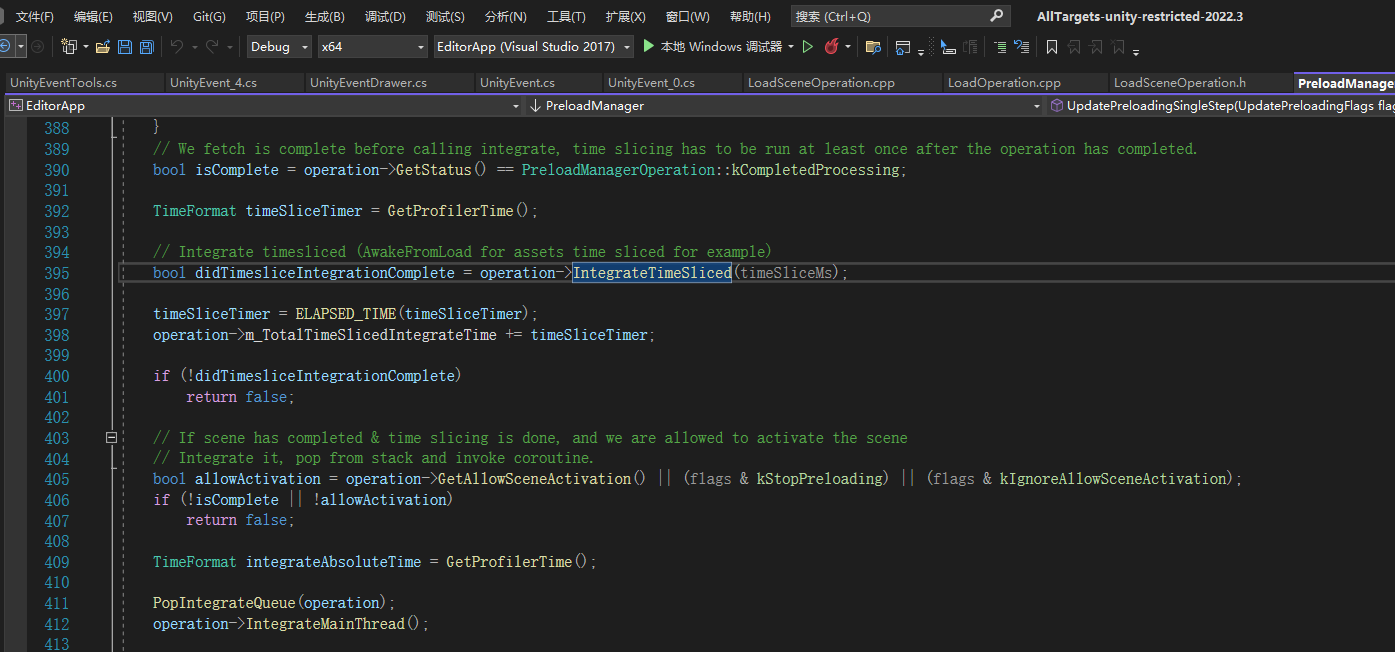
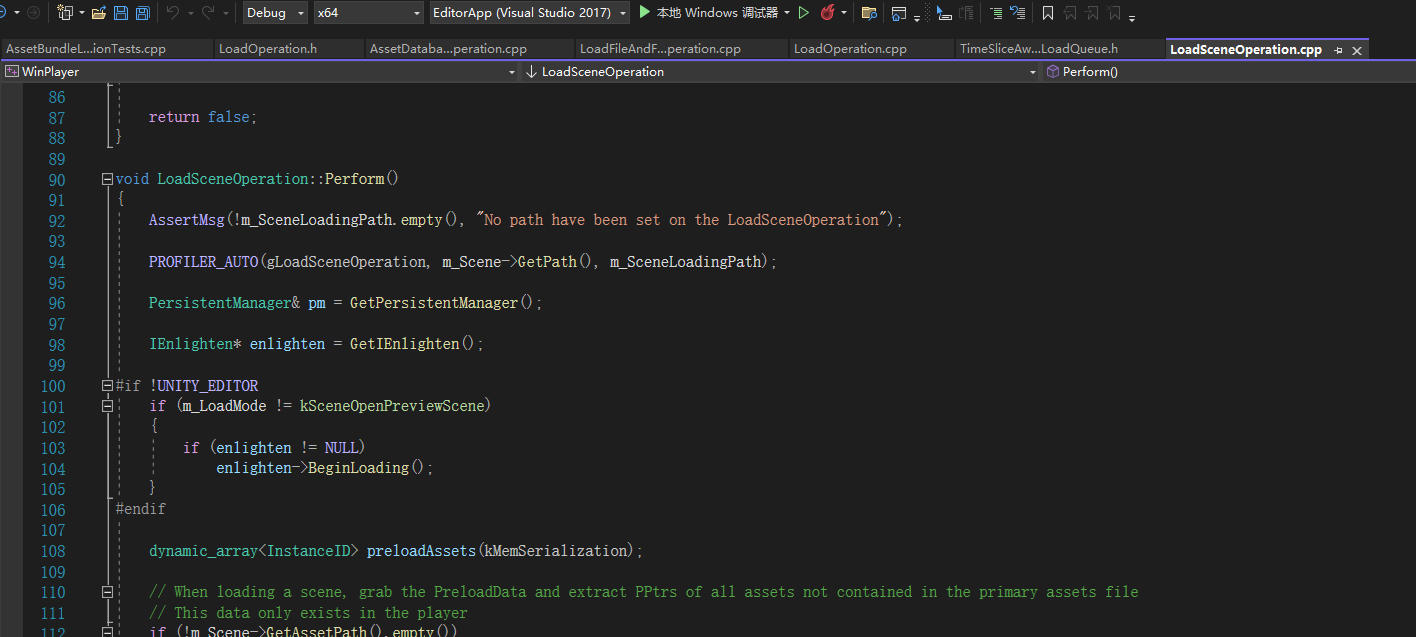
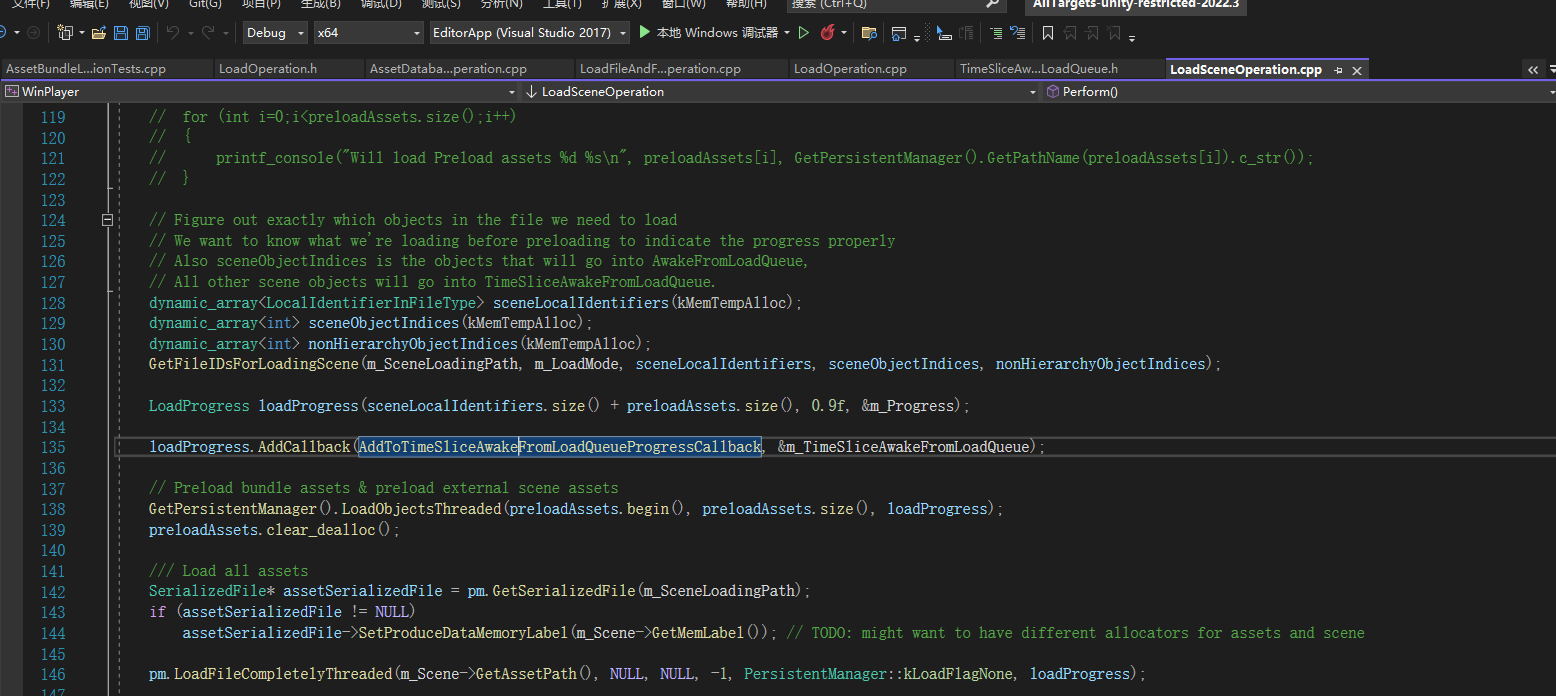
场景加载完成之后在这里做分帧的准备操作



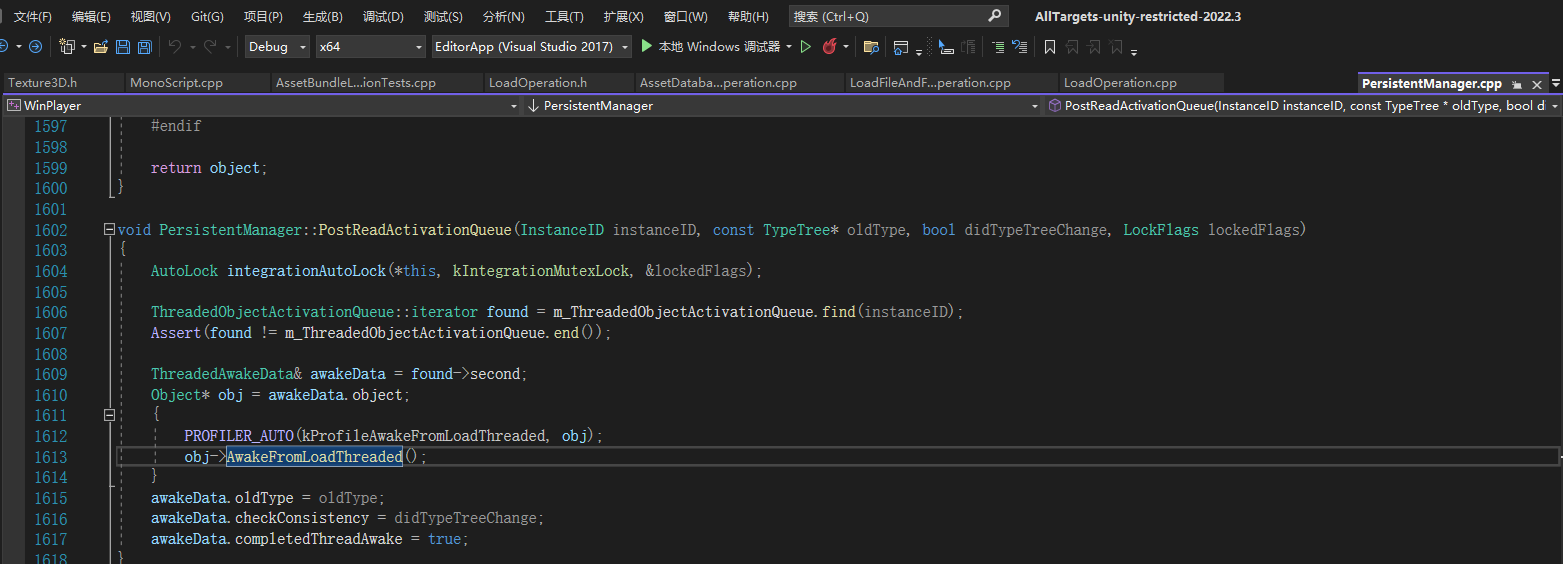


实际上就是调用operation->IntegrateTimeSliced

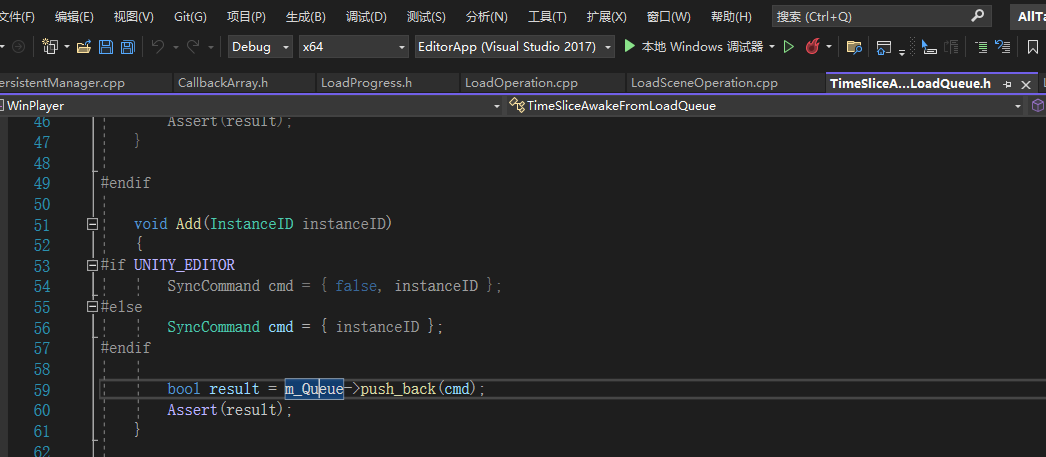


loadProgress.AddCallback(AddToTimeSliceAwakeFromLoadQueueProgressCallback, &m\_TimeSliceAwakeFromLoadQueue); 这里会将Scene加载结束之后所有的SceneObject放入到AwakeInThread的队

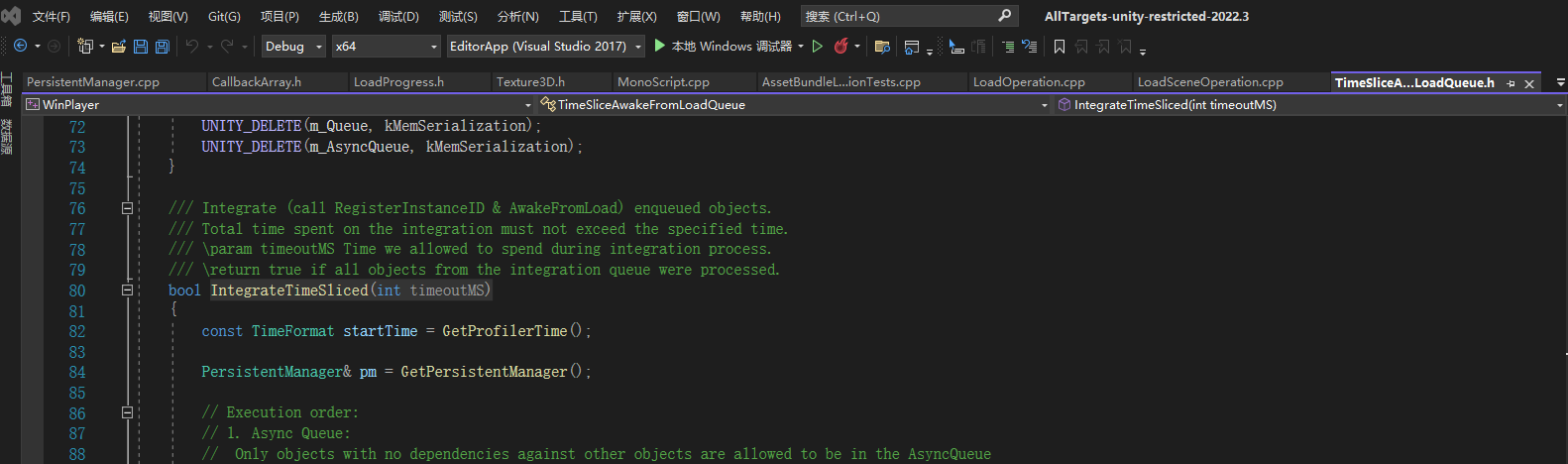
在场景加载的时候，会读取Mono，随后直接在线程做ThreadOnlyAwake



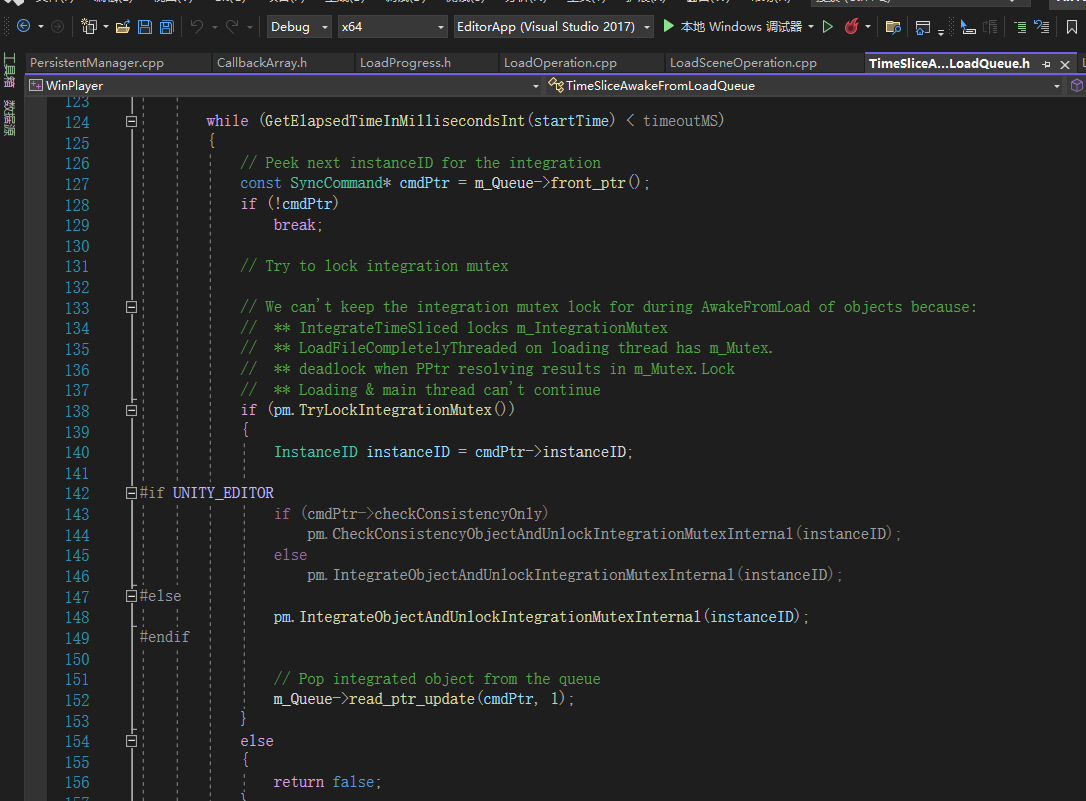
pm.LoadFileCompletelyThreaded(m\_Scene->GetAssetPath(), NULL, NULL, -1, PersistentManager::kLoadFlagNone, loadProgress);的时候会将所有的Object加入到m\_TimeSliceAwakeFromLoadQueue，在这句话完成（也就是场景Object都加载完成之后）将m\_ThreadedObjectActivationQueue列表加入到m\_TimeSliceAwakeFromLoadQueue



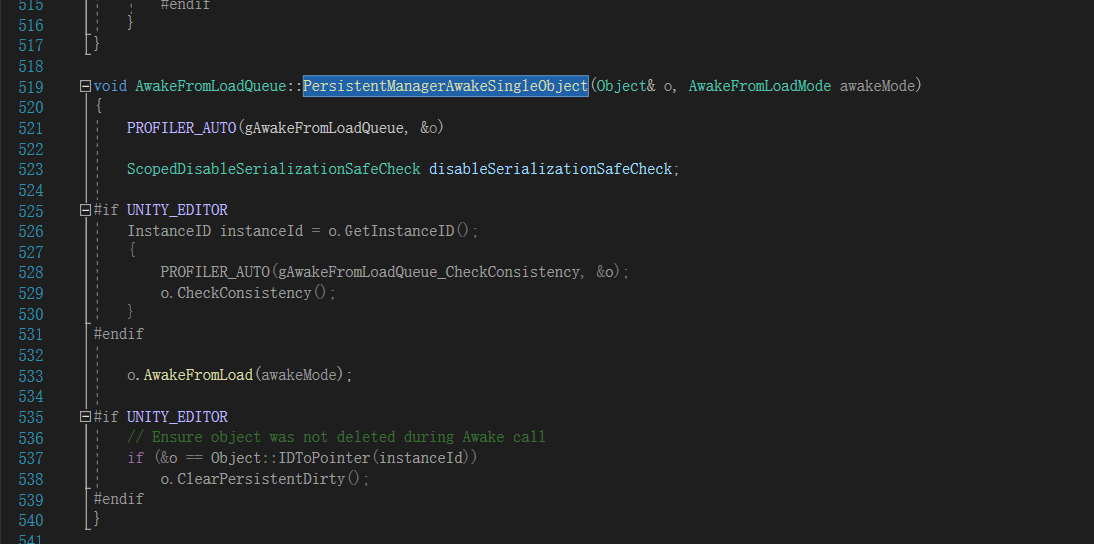
可以看到加入到了m\_Queue里面



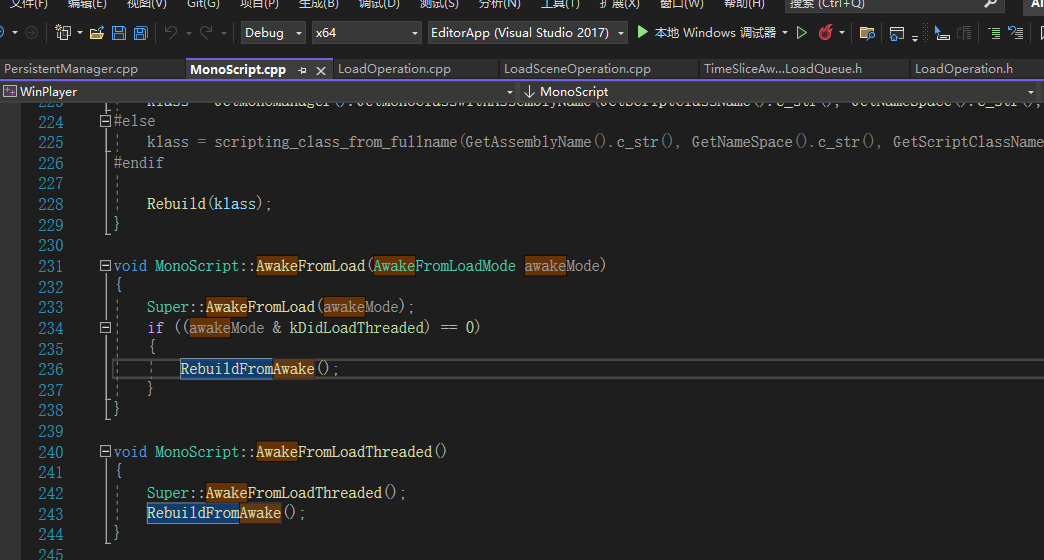
随后这个m\_TimeSliceAwakeFromLoadQueue可以在IntegrateTimeSliced(int timeoutMS)来做分帧操作

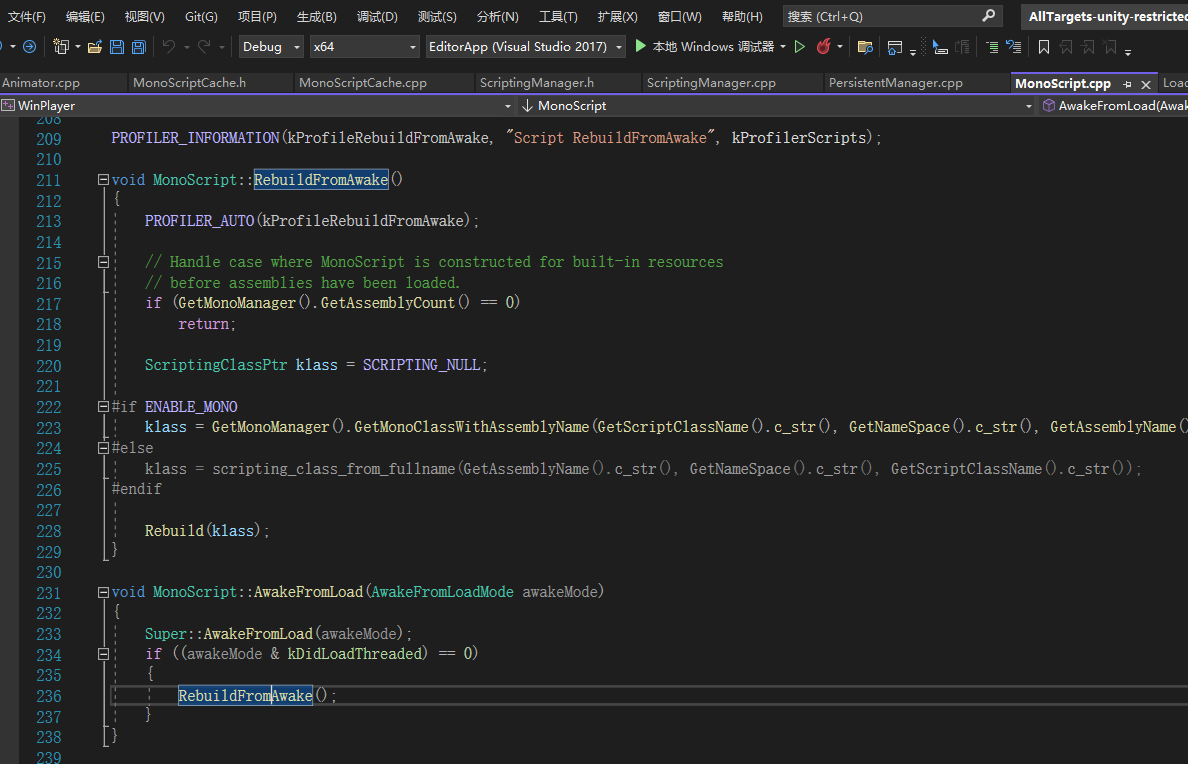


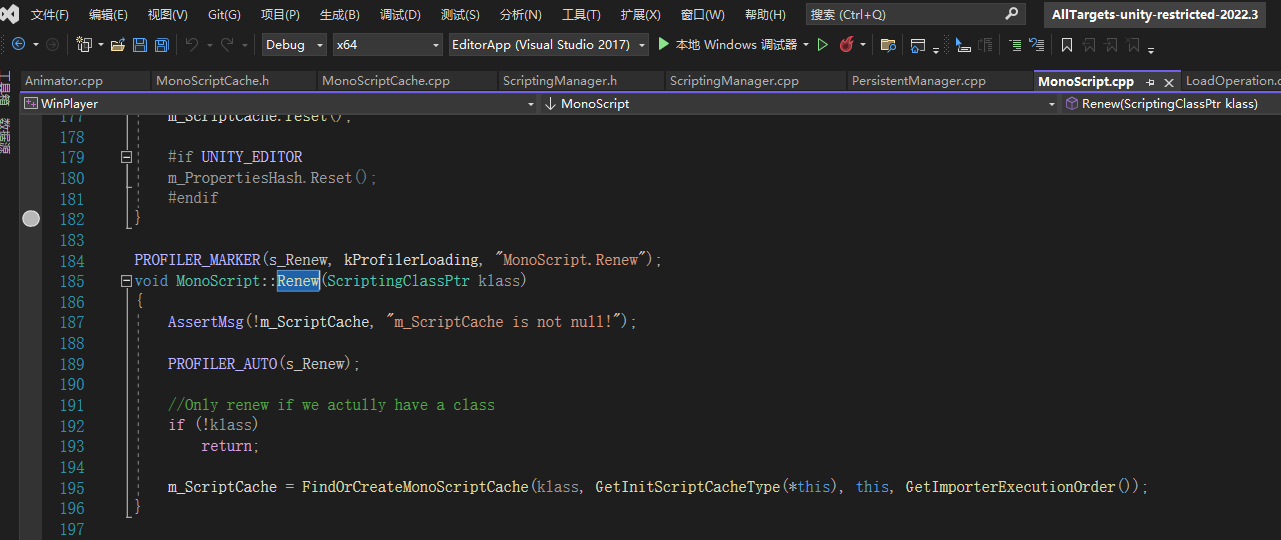
pm.IntegrateObjectAndUnlockIntegrationMutexInternal(instanceID);里面调用AwakeFromLoad

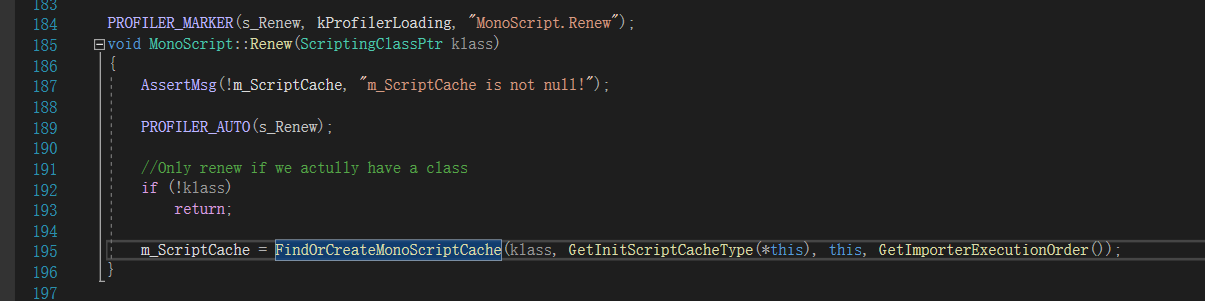


对于Mono来说调用效果如下



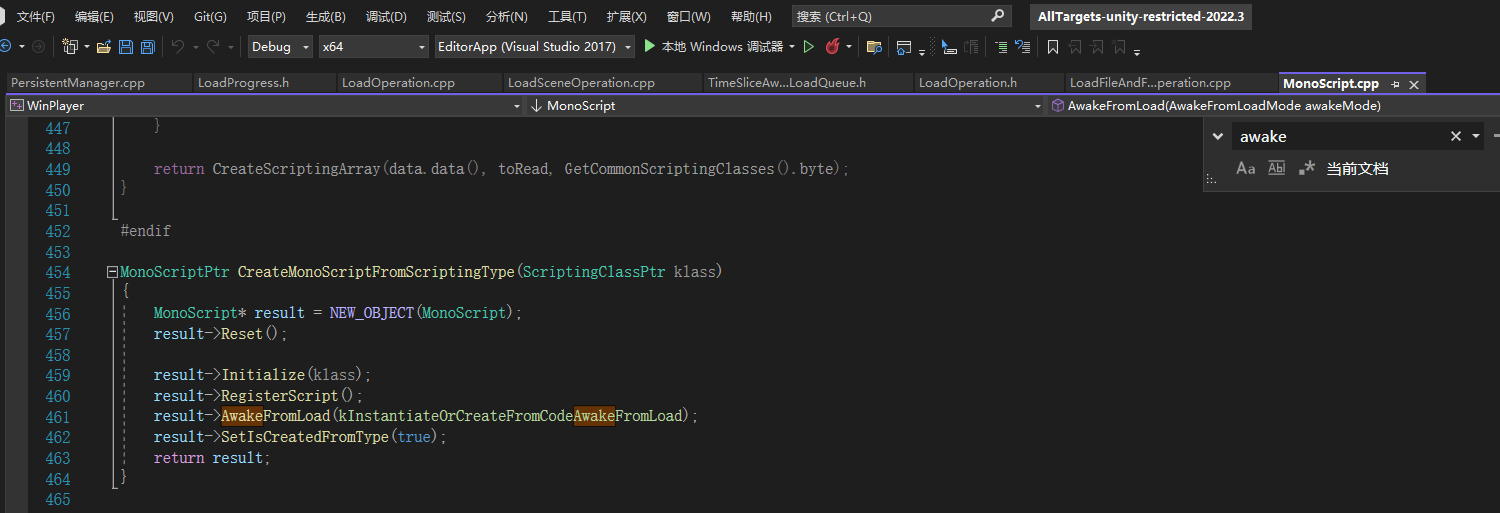




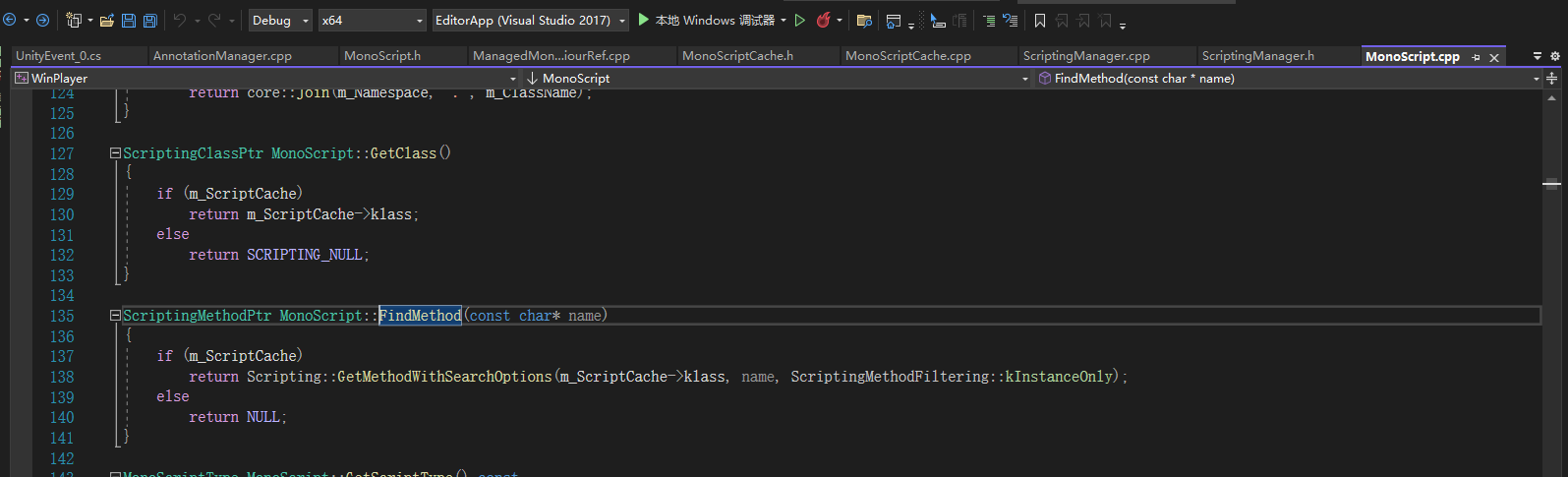


这里的m\_ScriptCache就是调用OnEnable Update之类生命周期方法的操作类

对于MonoBehaviour创建的时候可以参考这个代码



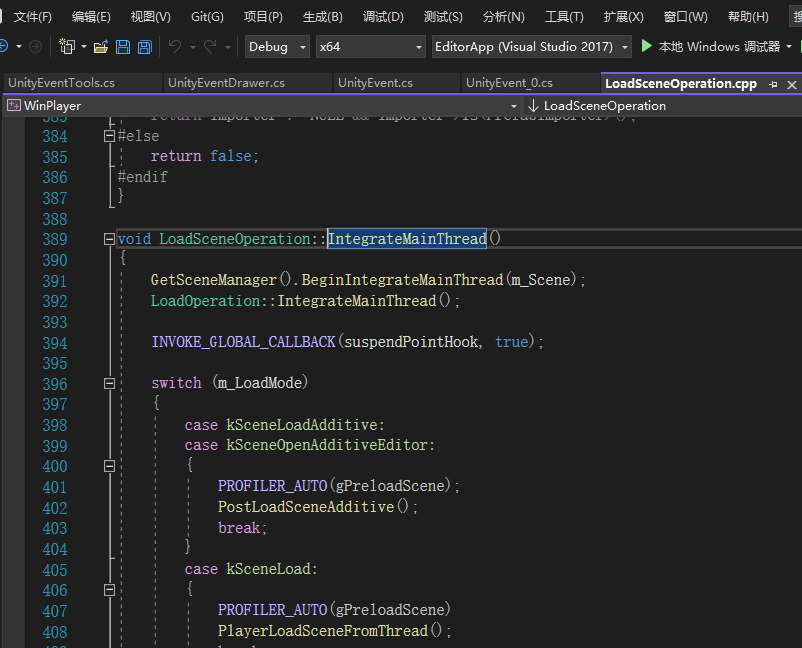
但这里只是注册了m\_ScriptCache，并收集了各种事件方法，后面会Unity用FindMethod的套路寻找方法来调用，这就是为什么空方法删除掉可以提高些许性能的原因

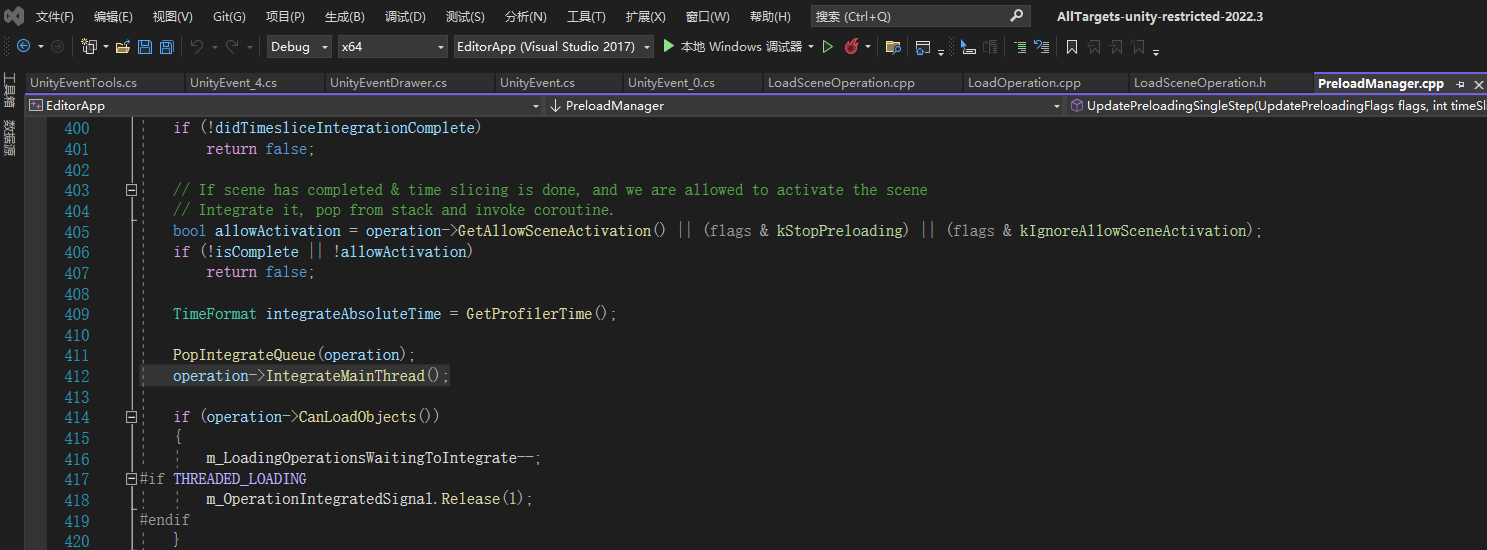


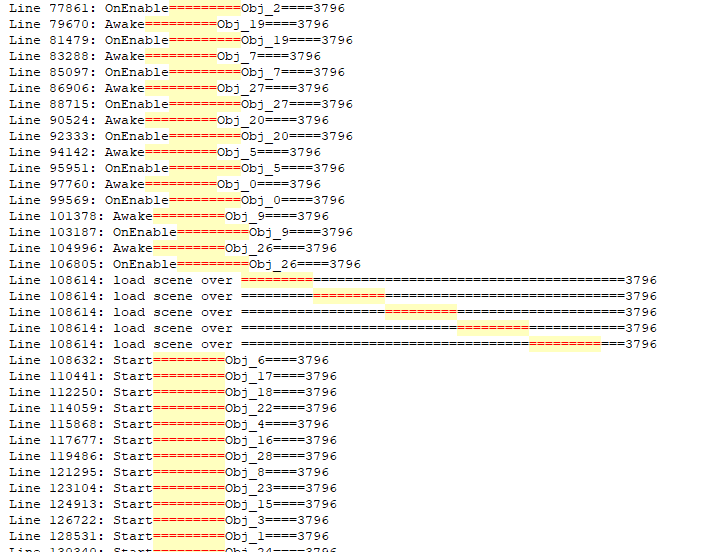
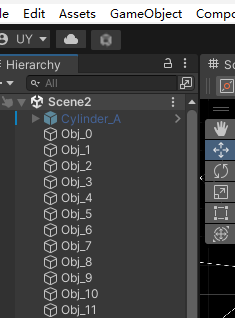
分帧操作全都完成之后，再在主线程来一做一次

operation->IntegrateMainThread();

对于场景加载的Operation来说即调用







按照测试结果来看，从AB异步加载场景确实是没有做GameObject以及组件的分帧，只有AddCache的注册分帧，顺序为 Awake -> OnEnable -> SceneLoadOver -> Start