Asset资源

1：第三方产生

2：unity产生

3：脚本

最终导入到library文件夹里，不同平台导入结果不同，导致，切换平台时要hold on

1：运行时（跟着打包）

2：编辑器

1：meta是对源资源的设置更改，并不会改变你导入的源文件

2：而是生成一个副本在library文件夹中

推荐放原文件，最终用到的文件是经过unity一顿神操作之后的文件。

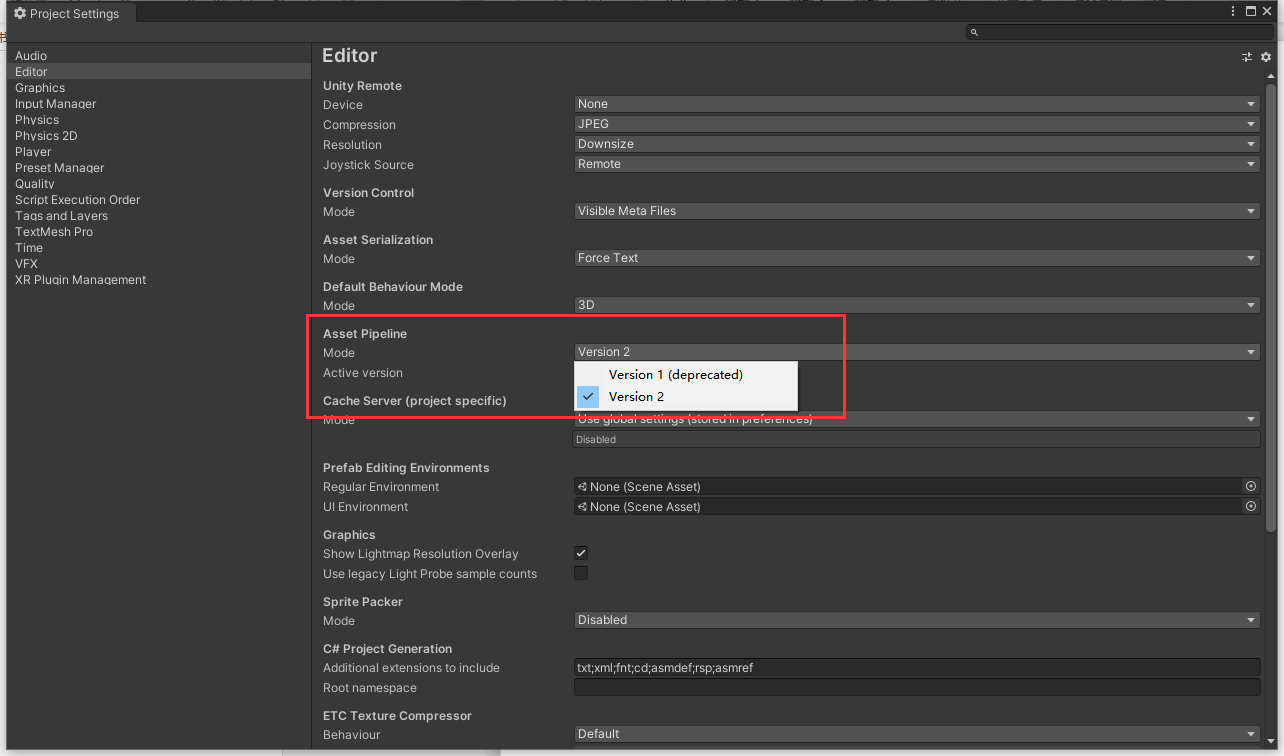
神奇的StreamingAssets文件夹

原封不动打进包里，不压缩，你怎么放到unity里，他就怎么储存在玩家机器里，unity不会压缩这个文件夹下的资源，所以apk也可以直接读取。

在安卓系统里可以直接被读出来（这可是apk啊）

这样使用方便，不过你也可以手动指定压缩

Meta 你可以用py写脚本遍历YAML格式的meta去修改meta，这跟你在编辑器里修改是一样的。

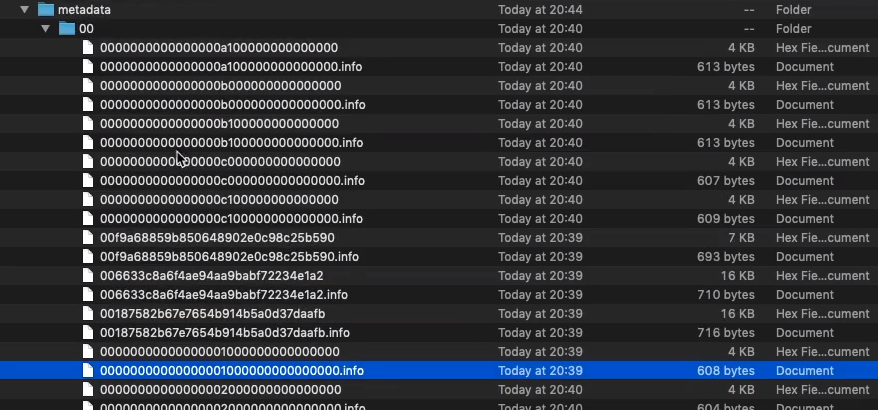


Version1 更直观但是已经要弃用了

Version2 封装了

当我们更改这个底层规则之后你的library文件夹会重新导入，我们看下有何蹊跷

1代版本这些资源都在library的这个文件夹下

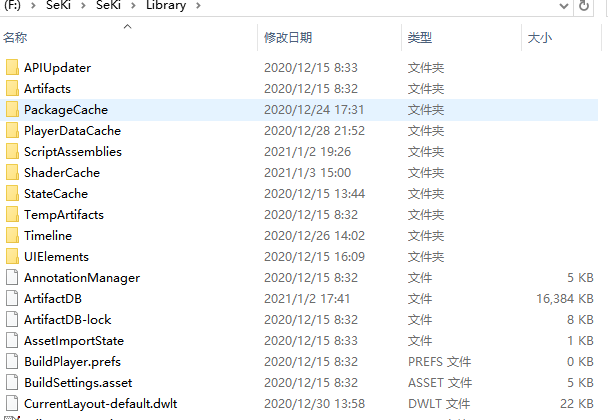


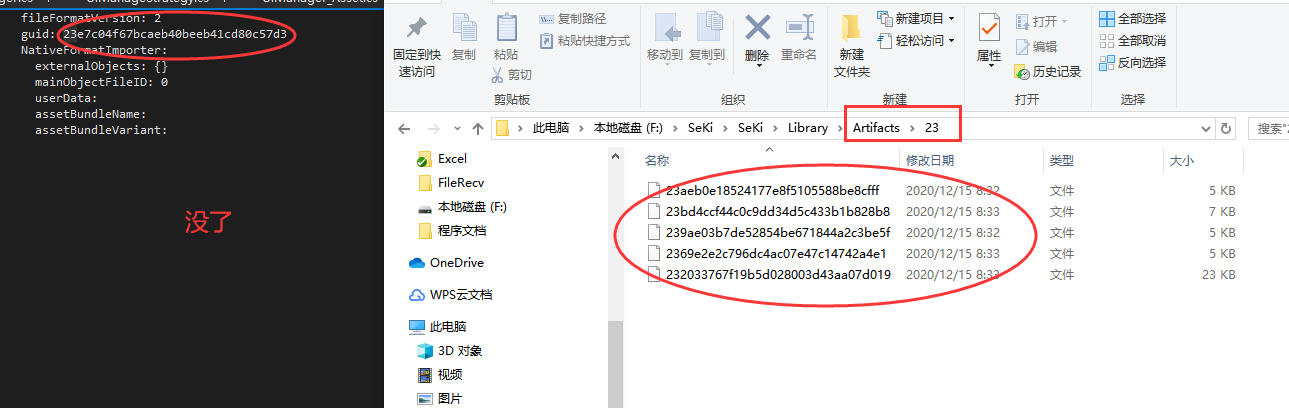
00开头湿系统的别动了

资源都在你手中，你可以做一些机智的操作

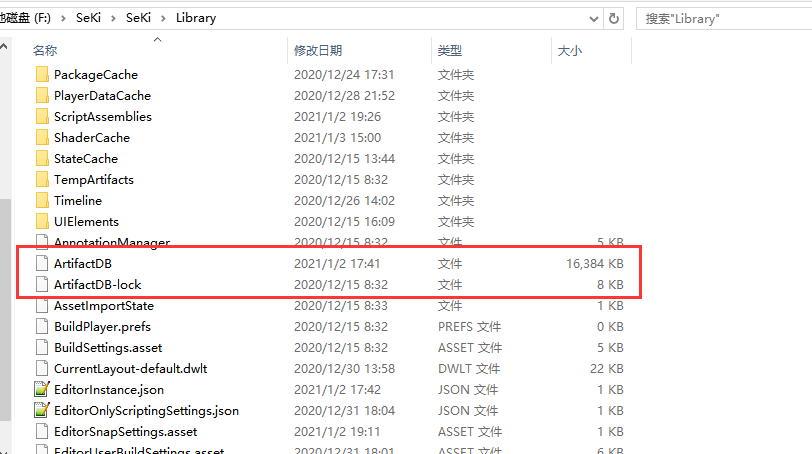
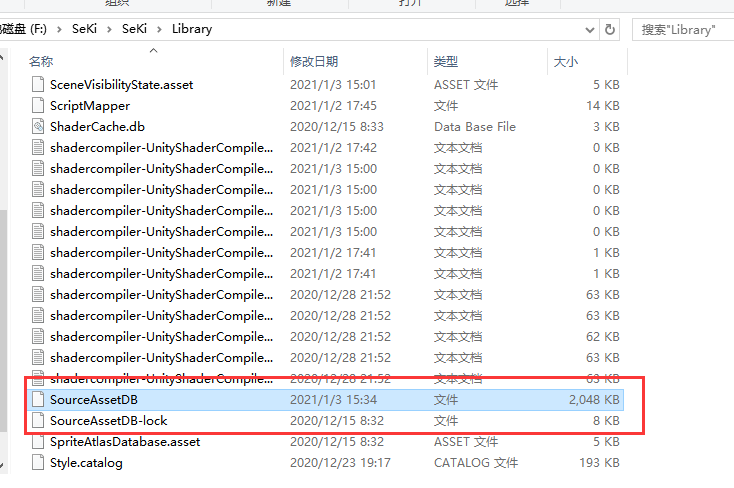
比如根据时间去决定哪些该重新打包

而2代



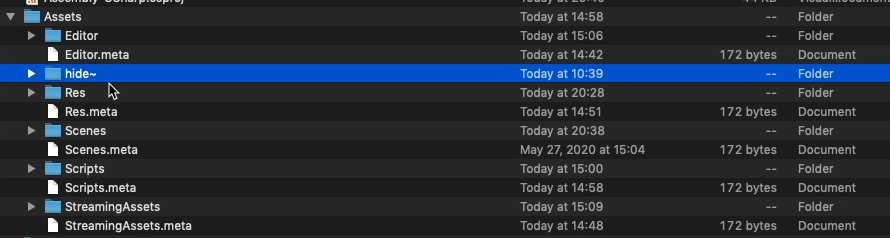


其实是放在了

这种文件里，这种文件是Lightning Memory-Mapped Database闪电内存映射数据库简称LMDB

跟Cache server有一点关系，题外话了

害羞的波浪线



这种文件unity会屏蔽它，不止文件夹哦

Asset 兄弟姐妹 assetbundle

Assetbundle 的原理：

用文件可以但是依赖关系不好搞，这是优势之一

还有快速索引、跨平台（大小端啥的）的优势

说白了，是一套虚拟文件系统

使得build代码一致性，可以一套代码到处打包，棒

一部分是压缩的内容，我们叫它内容体，体

还有就是头，摘要信息。

（实际更复杂，最里面是CAD开头的cave，rcavs）

Assetbundle和thingbundle不能打在一起，要分开（我他吗都听不清）

Assetbundle加载的时候头会立即加载

Bundle里面的asset是按需加载的，不需要不加载，但是有一个例外（我他吗都听不清）

AssetBundle的参数：

BuildAssetBundleOptions资源包构建选项。

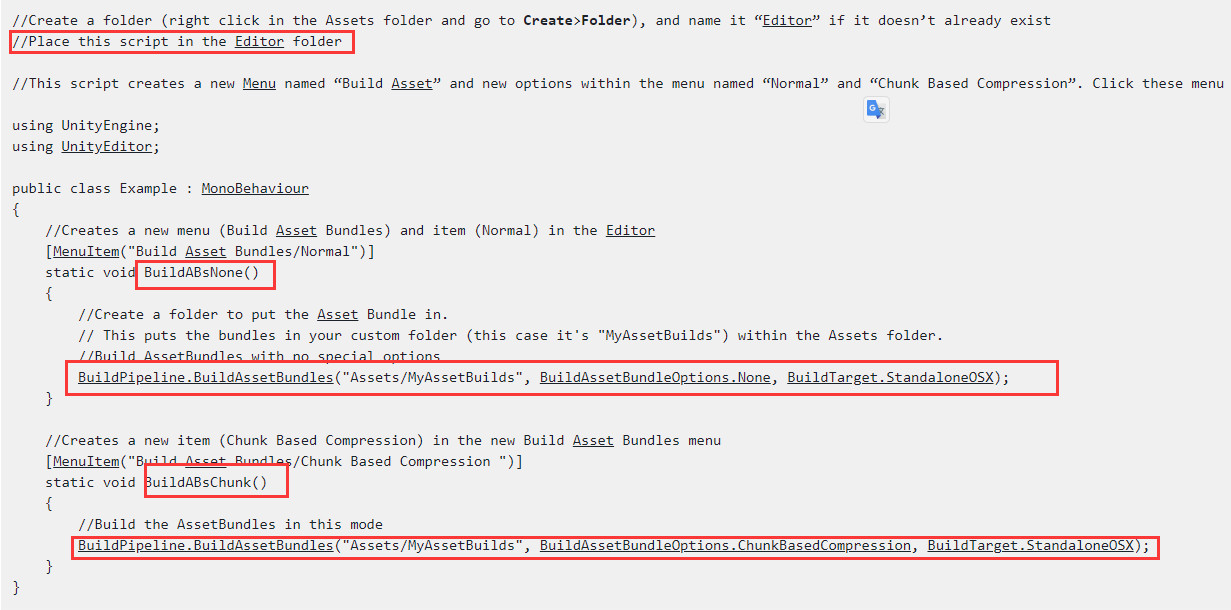
ChunkBasedCompression参数

压缩你的AssetBundle

比如安卓的streamingasset不压缩的，那你要是想要减少包体大小怎么办？我们可以自己去做ChunkBasedCompression。

实际上是改良过得LZ4

这允许在从 AssetBundle 读取数据时进行实时解压缩。 使用此选项创建的 AssetBundle 在下载后会以压缩形式存储（位于磁盘缓存或内存中）



DisableWriteTypeTree参数

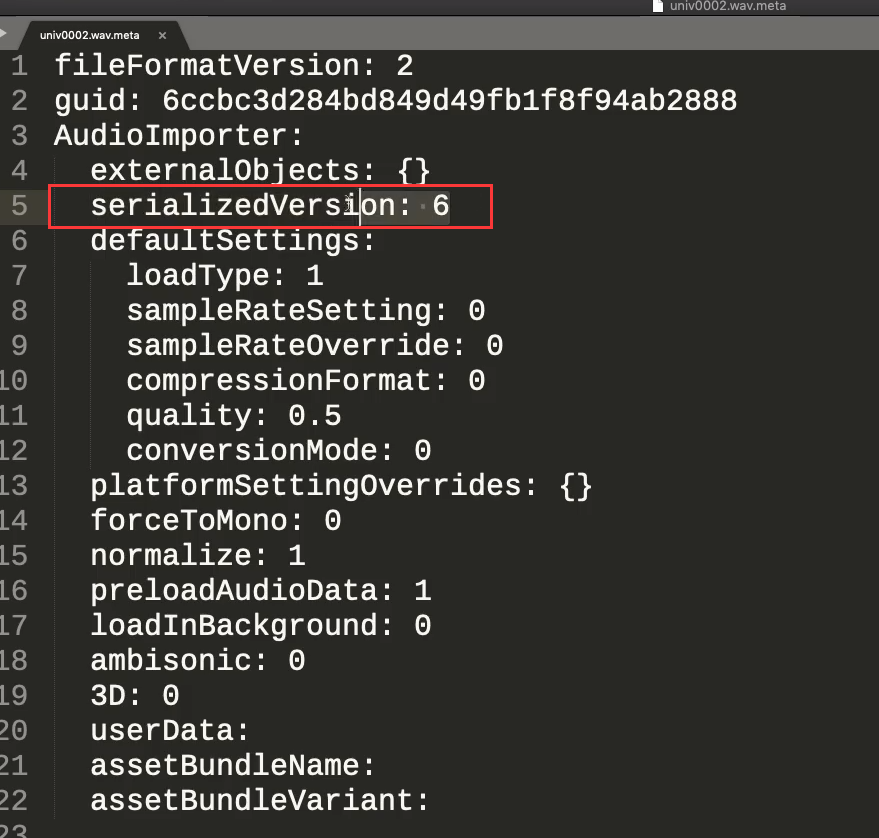
减小你的AssetBundle包体的大小

减小你的内存大小

节省你去加载这个AssetBundle的时候的cpu时间

（既然关掉它有这么多好处，为甚么unity还要写入TypeTree）

因为可以跨版本做兼容性



这条说明unity在使用更改了6次的版本属性

在typetree打开时，解析这个文件时，就会比对版本号，图中是6，它会去找对应的6号版本号的属性列表，一一解析若是5那就找5的，所以这就意味着，一份文件，他要去存储各个版本的解析方式，并且在运行时，动态比对。

这样跨版本的事情做出来了，但是很浪费，所以，如果你确认，你的assetbundle和你的apk版本一致，那你就可以不用typetree.

比如都用2019.3.7f1去打assetbundle和apk那你就可以~~~

DisableLoadAssetByFileName参数

DisableLoadAssetByFileNameWithExtension参数



Unity在加载一个assetbundle时一般写法就这样，很简单，这是纯路径，还有一种是只写名字，也是可以找到的，还有一种是你加上他的扩展名，也是可以找到的，但是，应用这些是有代价的，你这样能找到是因为unity把他们写到hash里，unity自己去生成原路径去对比找到的，所以，会有性能消耗，而如果你确定你都是用原路径去加载的话，，你完全可以应用这两个参数去把他们关闭掉，来节省时间、空间。

AssetBundle的识别：

AssetBundle的策略：

不大不小就好

官方推荐1m-2m，一般2m（指下载的，进包的5M，不要超10M，最大2G吧，断线续传需要服务器支持）

5G的话可能5m-10m

加载和管理

Editor和runtime加载机制不同，也就是一定要以真机去性能分析

序列化和反序列化

两个场景，一个场景物体都是非预制体5个cube，另一个都是prefab物体5个cube

在解析、加载的时候，使用prefab的场景会更快

反应在实际方面：.unity文件前一个文本量更多，后一个更少

同步异步用啥？

都有优缺点

同步快，但是主线程卡顿

异步慢，至少比同步慢一帧，我这一帧开始异步加载，最快也要下一帧执行。

异步和同步要是同时使用，可能就会有问题，是这两个大佬的问题

Preload 和 presistent

Unity引擎内部，是由这两个东西主要进行加载的

Preload 负责调度任务，当上层有任务下来，回形成一个option=>，给到preloadmanager，其内部有一个队列，每一帧，会从里面取出一个opt去执行，如果是并行的，会再赛一个再塞一个，但是，在并行执行的过程中，会去调用presistentmanager，这个presistentmanager任务：把文件从硬盘读取到内存，然后给他一个ID。

所以同步异步有问题：

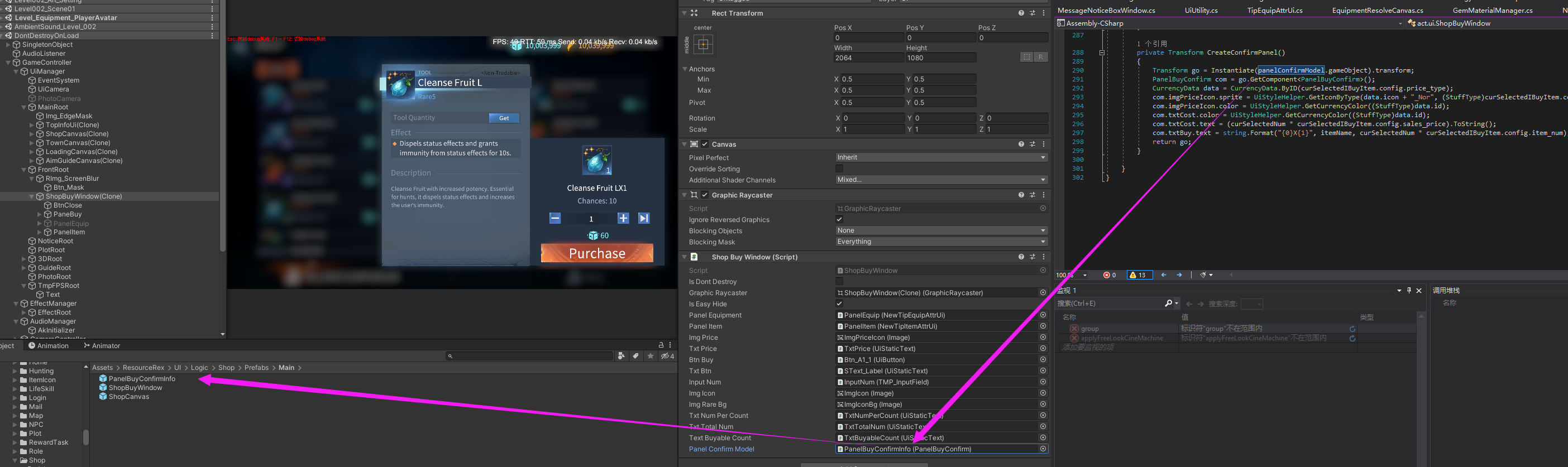
此时一个opt异步正在跑，下一帧有一个同步的opt来了，同步的和异步的会去抢这个presistentmanager，因为presistentmanager分配ID会去阻断线程，block所以你的异步工作可能会被同步工作阻断，反过来也有可能。

两个纯同步也会有问题

Asset的卸载

Unloadunusedasset 必须独自完成不能并行 场景方面unity已经做好了 不归场景管理的资源需要手动前去调用

Assetbundle.unload 自主管理，主要分为两种：一种遍历把bundle和asset都卸载了，或者只卸载bundle

预制体资源居然可以这样用