# 资源框架

资源框架分为ResourceManager，AssetBundleManager，ObjectManager三部分；游戏初始化在空对象挂载一个脚本RFramework（继承可挂载单例，负责管理类的初始化），以此初始化ResourceManager，AssetBundleManager，ObjectManager；

ResourceManager的初始化Init，根据优先级的数量，New出异步加载资源队列的数量（一个优先级一个List），方法传入挂载空对象的MonoBehaviour，用空对象的MonoBehaviour开携程，死循环，按照优先级，先下载当前最高优先级的资源的加载列表List，编辑器下直接根据路径用AssetDatabase下载（非编辑器模式，调用GetCacheResouceItem方法从字典缓存中获取，更新上次使用时间和引用计数，如果缓存中没有调用abManager的LoadResouceAssetBundle方法，根据crc值加载ab包，根据资源名字m\_AssetName，用ab包异步加载的方式加载出ab包，更新上次yeild的时间）；调用CacheResource方法缓存起来（调用WashOut方法，如果超出最大限制，清除最早未使用的资源，重新赋值刷新资源的引用次数和上次使用时间，存进字典）；调用自己异步加载结束的回调，重置回调方法，并且回调池回收这个回调，如果上次yeild时间超过最大加载时间，则进行yeild操作，进行其他操作，防止线程阻塞，等待下次调用继续下载。

AssetBundleManager的初始化LoadAssetBundleConfig，调用AES类AESFileByteDecrypt解密的方法，传入路径和秘钥，把该路径下的文件读取字节流，截取头标签和AES定义的头标签验证解密的是否是AES文件（防止浪费性能并且提高了安全性），是的话开始解密，返回解密成功的字节流数据。调用AssetBundle.LoadFromMemory（解密的字节流数据）获得清单列表的ab包，获取清单ab中的TextAsset，转为字节流内模式在序列化为自定义的 AssetBundleConfig类，遍历获得AssetBundleConfig类中ABBase的集合，把ABBase的值赋给ResouceItem类，并且以crc作为key，ResouceItem作为值存进字典中。

ObjectManager的初始化Init（回收池节点，场景节点），调用GetOrCreatClassPool<ResouceObj>(1000)方法，创建资源对象的对象池，如果类对象池字典中有，直接返回这个对象池，没有的话，New一个对象池类，存入类对象池字典并且返回这个对象池。

## 辅助类：

1. ResouceObj（单个对象类）
2. ResouceItem（单个对象的ab包类）
3. AssetBundleConfig清单文件类：可序列化，存储自定义ABBase的集合
4. ABBase（ab包基类）：可序列化（中间类，序列化后深拷贝赋值给ResouceObj类）
5. AsyncCallBack（回调类，存结束的委托和调用委托参数和重置方法（重置后可用对象池回收））
6. AsyncLoadResParam（异步加载对象类，存储加载完成后的回调集合）

## 辅助枚举：

LoadResPriority（加载资源优先级）：mid，low，hight，num

## ResourceManager

**内部字段：**

1. m\_Guid：ResouceObj唯一id
2. Dictionary<uint, AsyncLoadResParam> m\_LoadingAssetDic：存储资源对应异步回调队列（key为这种资源crc，value为异步回调的集合类AsyncLoadResParam）
3. List<AsyncLoadResParam>[] m\_LoadingAssetList存储所有优先级正在异步加载资源的回调列表的集合的数组：一个优先级对应数组中的一项（即一个异步回调队列集合），一个异步回调队列AsyncLoadResParam是一个资源的异步回调集合
4. m\_ClassObjectPool<AsyncLoadResParam> m\_AsyncLoadResParamPool异步队列类的对象池
5. ClassObjectPool<AsyncCallBack> m\_AsyncCallBackPool单个异步回调类的对象池

**公开字段：**

1.Dictionary<uint, ResouceItem> AssetDic存储该资源的字典：crc作为key

**公开接口：**

1.long CreatGuid()：给ResouceObj类赋值位置id

2.void ClearCache()：遍历缓存的字典，根据是否跳转场景删除这个字段，把跳转场景需要删除的资源调用内部DestoryResouceItme方法回收资源

3.bool CancleLoad(ResouceObj)取消异步加载：根据ResObj的crc，在LoadingAssetDic字典中找到这个资源的异步队列，遍历他的异步队列，重置这个回调，m\_m\_AsyncCallBackPool回收这个回调，准备复用，从这个资源的回调队列中都移除。遍历完后，如果该资源的回调队列已经清空，重置这个回调队列，m\_LoadingAssetList移除这个回调队列，异步队列回收池回收这个异步队列

4. int IncreaseResouceRef(ResouceObj resObj, int count = 1)增加该资源的引用次数，并且返回次数：调用自己重载函数，根据resobj的crc值，增加引用计数

5.int IncreaseResouceRef(uint crc = 0, int count = 1)：根据crc值增加引用计数，使用Time.realtimeSinceStartup更新该资源的 上次使用时间

6.int DecreaseResoucerRef(ResouceObj resObj, int count = 1)

7.int DecreaseResoucerRef(uint crc, int count = 1)根据crc减少引用次数，并且返回当前引用次数

8.void PreloadRes(string path,bool isSprite = false)预加载的方法，根据路径，调用Crc32类获取crc值。1.已经加载过的根据crc值调用GetCacheResouceItem（crc，0增加引用次数）方法从缓存的字典AssetDic中获取ResouceItem；2.没有加载过，缓存中没有，调用ABManager的LoadResouceAssetBundle(crc)方法，加载出ResouceItem，调用CacheResource方法添加到缓存里，设置这个ResItem为跳转场景不删除（因为可能场景跳转过去才加载出来）

9.ResouceObj LoadResource(path,resObj)同步加载资源，针对给ObjectManager的接口：根据路径计算crc值。1.已经加载过的根据crc值调用GetCacheResouceItem（crc，0增加引用次数）方法从缓存的字典AssetDic中获取ResouceItem；2.没有加载过，缓存中没有，调用ABManager的LoadResouceAssetBundle(crc)方法，加载出ResouceItem，调用CacheResource方法添加到缓存里，给resobj赋值ResItem（ab包）并且返回这个resobj

10.bool ReleaseResouce(ResouceObj resObj, bool destoryObj = false)根据ResObj卸载资源，摧毁这个resobj的预制体，引用次数--，调用DestoryResouceItme方法

11.AsyncLoadResource(path, OnAsyncResFinish , LoadResPriority , bool isSprite = false,

object param1 = null, object param2 = null, object param3 = null, uint crc = 0)异步加载资源的方法：根据路径计算crc值，如果缓存中有，调用GetCacheResouceItem方法获取ResItem，如果获取到的不为空，调用传过来的结束的回调委托；如果缓存中没有，m\_LoadingAssetDic中根据crc取出正在加载这个资源的异步队列（如果没有加载，从异步队列回收池取出一个，把这个异步队列存入m\_LoadingAssetDic和m\_LoadingAssetList中），New一个异步回调，把这个异步回调存进取出来的异步队列中。

1. Sprite LoadSpriteBySpriteAtlas(传入完整路径)通过图集加载图片：裁剪完整路径，获取到图集路径和图片名字，调用LoadResource方法加载图集，根据图片名字返回图集中获取到的图片

**内部方法：**

1. DestoryResouceItme（ResouceItem，是否清除缓存）：如果不清除字典中缓存，存入最大堆中，准备复用；如果清清除缓存，把这个资源在缓存中取出，也从最大堆中取出，调用ABManager的ReleaseAsset（ResouceItem）方法，释放ab包资源；调用ObjManager的ClearPoolObject（crc）方法，清除这个资源资源在对象池中的所有对象，m\_LoadingAssetDic字典中也移除
2. ResouceItem GetCacheResouceItem(uint crc, int addrefcount = 1)从字典AssetDic中获取ResouceItem，增加引用次数，更新上次使用时间
3. void CacheResource(string path, ref ResouceItem item, uint crc, Object obj, int addrefcount = 1)：调用WashOut方法，如果超出最大限制，清除最早未使用的资源，重新赋值刷新资源的引用次数和上次使用时间，存进字典或者刷新字典中的值

## AssetBundleManager

**字段：**

m\_ResouceItemDic（字典存储ab清单文件中所有的ab包，key=crc，value=单个游戏对象的item类）

**公开接口：**

1. LoadAssetBundleConfig（初始化方法）：如果是编辑器下，直接返回；把热更新下载的清单列表中所有ab包存入字典
2. ReleaseAsset(ResouceItem)（释放资源方法）：遍历自己所有依赖项，调用UnLoadAssetBundle（name）方法，卸载所有的依赖项，然后在卸载自己。

内部方法：

1. void UnLoadAssetBundle(name)卸载AB包方法：根据ab包name获取crc值，在字典中找到这个资源，引用次数--，如果引用次数<0，UnLoad（true）卸载这个ab包，调用rest方法重置这个AssetBundleItem，字典中移除这个ab包，调用对象池中Recycle方法回收这个AssetBundleItem

## 三、ObjectManager

**内部字段：**

1.Dictionary<uint, List<ResouceObj>> m\_ObjectPoolDic：crc作为key（一种对象一个key，值为该游戏对象的集合），存储某些游戏对象的Resobj集合（比如小怪，小草）

2.Dictionary<int, ResouceObj> m\_ResouceObjDic：存所有游戏对象的ResouceObj，key为游戏对象的GetInstanceID（key为游戏对象唯一id，值为对应的ResouceObj）

3.ClassObjectPool<ResouceObj> m\_ResourceObjClassPool：ResouceObj这个类的对象池，负责这个ResouceObj的回收复用

4.Dictionary<long, ResouceObj> m\_AsyncResObjs：根据guid存储当前正在异步加载的Resobj

**公开字段：**

回收节点，场景节点

**公开接口：**

1. void ClearPoolObject(uint crc)清除这个资源在集合字典中的所有对象：根据crc值，在ObjectPoolDic字典中找到这个ResouceObj集合遍历，如果跳转场景删除：1.从找到的集合中删除，2.获取到ResouceObj的预制体的GetInstanceID，根据这个id，从m\_ResouceObjDic中移除这个ResObj，这个摧毁这个对象，3.然后ResouceObj这个就可以调用reset重置，ResourceObjClassPool回收池回收这个实例，准备再次复用，4.如果从ObjectPoolDic字典中找到的集合已经全部清空，从这个字典中把这个集合移除
2. Void ClearCache()清空m\_ObjectPoolDic字典中存储的所有游戏对象的集合：声明一个临时uint集合存储所有需要清除资源的crc，遍历对象集合的字典中的每个集合，（遍历这个集合，如果这个对象不为空并且是跳转场景需要删除的，摧毁这个对象，从缓存对象的字典中移除这个对象，ResObj重置，ResObj回收池回收这个ResObj，从这个集合中删除。如果这个集合的长度为0，存进生命的临时集合里面）。遍历临时集合crc，如果对象集合字典中有这类对象，从这个对象集合字典中删除
3. OfflineData FindOfflineData(GameObject obj)根据实例化对象直接获取离线数据：通过这个对象的唯一ID在m\_ResouceObjDic字典中获取到这个ResObj中的离线数据
4. void CancleLoad(long guid)取消异步加载：根据guid在m\_AsyncResObjs正在异步加载的字典中获取这个resobj，调用ResMgr的CancleLoad（resobj）取消异步方法，如果成功取消异步加载，把这个resobj从正在异步加载的字典中移除，把这个resobj重置，resobj回收池回收这个resobj
5. bool IsingAsyncLoad(long guid)判断那这个对象是否正在异步加载：判断正在异步加载的字典中有没有这个guid，即可知道
6. bool IsObjectManagerCreat(GameObject obj)判断这个对象是否是这个类创建的：根据obj的唯一id，查看存储所有resobj的字典中有没有，有的话是

**内部方法：**

1.ResouceObj GetObjectFromPool(uint crc)从对象池中取对象并且把数据设为初始状态：根据crc值从m\_ObjectPoolDic对象集合字典中取出这类对象的集合，从集合中取出一个，调用ResMgr的IncreaseResouceRef方法增加引用计数，调用方法重置resobj的离线数据