**Unity3D教程：Unity加载和内存管理**

Posted on 2013年02月10日 by U3d / [Unity3D 基础教程](http://www.unitymanual.com/category/manual/unity3d-%e5%9f%ba%e7%a1%80%e6%95%99%e7%a8%8b)/被围观 416 次

Unity里有两种动态加载机制：一是Resources.Load，一是通过AssetBundle,其实两者本质上我理解没有什么区别。Resources.Load就是从一个缺省打进程序包里的AssetBundle里加载资源，而一般AssetBundle文件需要你自己创建，运行时动态加载，可以指定路径和来源的。

其实场景里所有静态的对象也有这么一个加载过程，只是Unity后台替你自动完成了。

详细说一下细节概念：

**AssetBundle运行时加载:**

来自文件就用CreateFromFile(注意这种方法只能用于standalone程序）这是最快的加载方法

也可以来自Memory,用CreateFromMemory(byte[]),这个byte[]可以来自文件读取的缓冲，www的下载或者其他可能的方式。

其实WWW的assetBundle就是内部数据读取完后自动创建了一个assetBundle而已

Create完以后，等于把硬盘或者网络的一个文件读到内存一个区域，这时候只是个AssetBundle内存镜像数据块，还没有Assets的概念。

**Assets加载:**

用AssetBundle.Load(同Resources.Load) 这才会从AssetBundle的内存镜像里读取并创建一个Asset对象，创建Asset对象同时也会分配相应内存用于存放(反序列化)

异步读取用AssetBundle.LoadAsync

也可以一次读取多个用AssetBundle.LoadAll

**AssetBundle的释放：**

AssetBundle.Unload(flase)是释放AssetBundle文件的内存镜像，不包含Load创建的Asset内存对象。

AssetBundle.Unload(true)是释放那个AssetBundle文件内存镜像和并销毁所有用Load创建的Asset内存对象。

一个Prefab从assetBundle里Load出来 里面可能包括：Gameobject transform mesh texture material shader script和各种其他Assets。

你Instantiate一个Prefab，是一个对Assets进行Clone(复制)+引用结合的过程，GameObject transform 是Clone是新生成的。其他mesh / texture / material / shader 等，这其中有些是纯引用的关系的，包括：Texture和TerrainData，还有引用和复制同时存在的，包括：Mesh/material/PhysicMaterial。引用的Asset对象不会被复制，只是一个简单的指针指向已经Load的Asset对象。这种含糊的引用加克隆的混合，大概是搞糊涂大多数人的主要原因。  
专门要提一下的是一个特殊的东西：Script Asset，看起来很奇怪，Unity里每个Script都是一个封闭的Class定义而已,并没有写调用代码，光Class的定义脚本是不会工作的。其实Unity引擎就是那个调用代码，Clone一个script asset等于new一个class实例，实例才会完成工作。通过AddComponent给物体添加一个Script Assets，就完成了把脚本类实例挂到Unity主线程的调用链里去的工作，Class实例里的OnUpdate OnStart等才会被执行。多个物体挂同一个脚本，其实就是在多个物体上挂了那个脚本类的多个实例而已，这样就好理解了。在new class这个过程中，数据区是复制的，代码区是共享的，算是一种特殊的复制+引用关系。

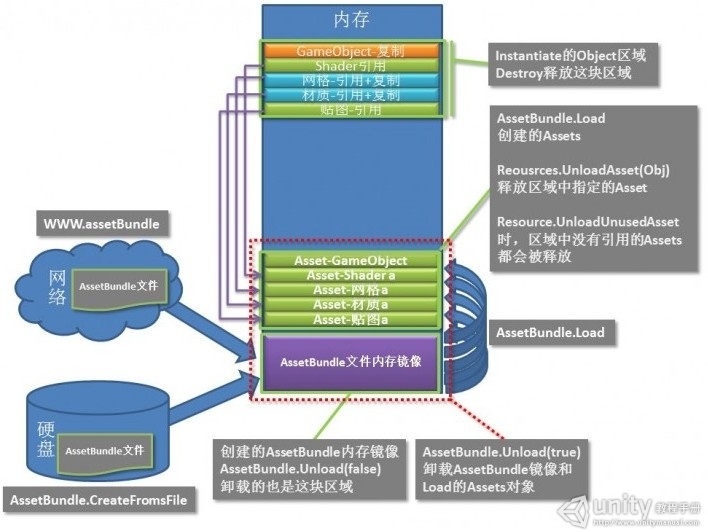
你可以再Instantiate一个同样的Prefab,还是这套mesh/texture/material/shader...，这时候会有新的GameObject等，但是不会创建新的引用对象比如Texture.

所以你Load出来的Assets其实就是个数据源，用于生成新对象或者被引用，生成的过程可能是复制（clone)也可能是引用（指针）

当你Destroy一个实例时，只是释放那些Clone对象，并不会释放引用对象和Clone的数据源对象，Destroy并不知道是否还有别的object在引用那些对象。

等到没有任何游戏场景物体在用这些Assets以后，这些assets就成了没有引用的游离数据块了，是UnusedAssets了，这时候就可以通过Resources.UnloadUnusedAssets来释放,Destroy不能完成这个任务，AssetBundle.Unload(false)也不行，AssetBundle.Unload(true)可以但不安全，除非你很清楚没有任何对象在用这些Assets了。

配个图加深理解：



Unity3D脚本：Unity加载和内存管理

**虽然都叫Asset，但复制的和引用的是不一样的，这点被Unity的暗黑技术细节掩盖了，需要自己去理解。**

**关于内存管理**

按照传统的编程思维，最好的方法是：自己维护所有对象，用一个Queue来保存所有object,不用时该Destory的，该Unload的自己处理。

但这样在C# .net框架底下有点没必要，而且很麻烦。

稳妥起见你可以这样管理

**创建时：**

先建立一个AssetBundle,无论是从www还是文件还是memory

用AssetBundle.load加载需要的asset

加载完后立即AssetBundle.Unload(false),释放AssetBundle文件本身的内存镜像，但不销毁加载的Asset对象。（这样你不用保存AssetBundle的引用并且可以立即释放一部分内存）

**释放时：**

如果有Instantiate的对象，用Destroy进行销毁

在合适的地方调用Resources.UnloadUnusedAssets,释放已经没有引用的Asset.

如果需要立即释放内存加上GC.Collect()，否则内存未必会立即被释放，有时候可能导致内存占用过多而引发异常。

这样可以保证内存始终被及时释放，占用量最少。也不需要对每个加载的对象进行引用。

当然这并不是唯一的方法，只要遵循加载和释放的原理，任何做法都是可以的。

系统在加载新场景时，所有的内存对象都会被自动销毁，包括你用AssetBundle.Load加载的对象和Instaniate克隆的。但是不包括AssetBundle文件自身的内存镜像，那个必须要用Unload来释放，用.net的术语，这种数据缓存是非托管的。

**总结一下各种加载和初始化的用法:**

AssetBundle.CreateFrom.....：创建一个AssetBundle内存镜像，注意同一个assetBundle文件在没有Unload之前不能再次被使用

WWW.AssetBundle：同上，当然要先new一个再 yield return 然后才能使用

AssetBundle.Load(name)：从AssetBundle读取一个指定名称的Asset并生成Asset内存对象，如果多次Load同名对象，除第一次外都只会返回已经生成的Asset对象，也就是说多次Load一个Asset并不会生成多个副本（singleton）。

Resources.Load(path&name)：同上,只是从默认的位置加载。

Instantiate（object)：Clone一个object的完整结构，包括其所有Component和子物体（详见官方文档）,浅Copy，并不复制所有引用类型。有个特别用法，虽然很少这样用，其实可以用Instantiate来完整的拷贝一个引用类型的Asset,比如Texture等，要拷贝的Texture必须类型设置为Read/Write able。

**总结一下各种释放**

Destroy:主要用于销毁克隆对象，也可以用于场景内的静态物体，不会自动释放该对象的所有引用。虽然也可以用于Asset,但是概念不一样要小心，如果用于销毁从文件加载的Asset对象会销毁相应的资源文件！但是如果销毁的Asset是Copy的或者用脚本动态生成的，只会销毁内存对象。

AssetBundle.Unload(false):释放AssetBundle文件内存镜像

AssetBundle.Unload(true):释放AssetBundle文件内存镜像同时销毁所有已经Load的Assets内存对象

Reources.UnloadAsset(Object):显式的释放已加载的Asset对象，只能卸载磁盘文件加载的Asset对象

Resources.UnloadUnusedAssets:用于释放所有没有引用的Asset对象

GC.Collect()强制垃圾收集器立即释放内存 Unity的GC功能不算好，没把握的时候就强制调用一下

在3.5.2之前好像Unity不能显式的释放Asset

**例子1：**

一个常见的错误：你从某个AssetBundle里Load了一个prefab并克隆之：obj = Instantiate(AssetBundle1.Load('MyPrefab”);

这个prefab比如是个npc，然后你不需要他的时候你用了：Destroy(obj);你以为就释放干净了，其实这时候只是释放了Clone对象，通过Load加载的所有引用、非引用Assets对象全都静静静的躺在内存里。

这种情况应该在Destroy以后用：AssetBundle1.Unload(true)，彻底释放干净。

如果这个AssetBundle1是要反复读取的 不方便Unload，那可以在Destroy以后用：Resources.UnloadUnusedAssets()把所有和这个npc有关的Asset都销毁。

当然如果这个NPC也是要频繁创建 销毁的 那就应该让那些Assets呆在内存里以加速游戏体验。

由此可以解释另一个之前有人提过的话题：为什么第一次Instantiate一个Prefab的时候都会卡一下，因为在你第一次Instantiate之前，相应的Asset对象还没有被创建，要加载系统内置的AssetBundle并创建Assets,第一次以后你虽然Destroy了，但Prefab的Assets对象都还在内存里，所以就很快了。

**例子2：**

从磁盘读取一个1.unity3d文件到内存并建立一个AssetBundle1对象，AssetBundle AssetBundle1 = AssetBundle.CreateFromFile("1.unity3d");

从AssetBundle1里读取并创建一个Texture Asset,把obj1的主贴图指向它obj1.renderer.material.mainTexture = AssetBundle1.Load("wall") as Texture;

把obj2的主贴图也指向同一个Texture Asset，obj2.renderer.material.mainTexture =obj1.renderer.material.mainTexture;

Texture是引用对象，永远不会有自动复制的情况出现(除非你真需要，用代码自己实现copy)，只会是创建和添加引用

如果继续：AssetBundle1.Unload(true) 那obj1和obj2都变成黑的了，因为指向的Texture Asset没了。

如果：AssetBundle1.Unload(false) 那obj1和obj2不变，只是AssetBundle1的内存镜像释放了

继续：Destroy(obj1),//obj1被释放，但并不会释放刚才Load的Texture

如果这时候：Resources.UnloadUnusedAssets();

不会有任何内存释放 因为Texture asset还被obj2用着

如果Destroy(obj2)，obj2被释放，但也不会释放刚才Load的Texture

继续Resources.UnloadUnusedAssets();

这时候刚才load的Texture Asset释放了，因为没有任何引用了

最后CG.Collect();强制立即释放内存，由此可以引申出论坛里另一个被提了几次的问题，如何加载一堆大图片轮流显示又不爆掉，不考虑AssetBundle，直接用www读图片文件的话等于是直接创建了一个Texture Asset

假设文件保存在一个List里

TLlist<string> fileList;  
int n=0;  
IEnumerator OnClick()  
{  
WWW image = new www(fileList[n++])；  
yield return image;  
obj.mainTexture = image.texture;  
  
n = (n>=fileList.Length-1)?0:n;  
Resources.UnloadUnusedAssets();  
}

这样可以保证内存里始终只有一个巨型Texture Asset资源，也不用代码追踪上一个加载的Texture Asset,但是速度比较慢

或者：

IEnumerator OnClick()  
{  
WWW image = new www(fileList[n++])；  
yield return image;  
Texture tex = obj.mainTexture;  
obj.mainTexture = image.texture;  
  
n = (n>=fileList.Length-1)?0:n;  
Resources.UnloadAsset(tex);  
}