<https://blog.csdn.net/u010377179/article/details/52922710>

<https://blog.csdn.net/Daniel_love/article/details/51699128>

<http://www.cnblogs.com/ybgame/p/3973177.html>

<https://blog.csdn.net/langresser_king/article/details/44208585/>

<https://blog.csdn.net/huutu/article/details/46477471>

<https://blog.csdn.net/hanghangaidoudou/article/details/78170591>

<https://www.cnblogs.com/lan-yt/p/7787290.html>

<https://www.cnblogs.com/WongSiuming/p/7574312.html>

[https://blog.csdn.net/lodypig/article/category/6315960](https://blog.csdn.net/lodypig/article/category/6315960   )

assetbundle五篇详细讲解

<https://blog.csdn.net/ldghd/article/details/8069489>

Unity里面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，其实场景中所有静态的对象也有这么一个加载过程，只是Unity后台自动完成了。

**采用Resources加载资源**

GameObject prefab = (GameObject)Resources.Load("prefab2", typeof(GameObject));

GameObject t = Instantiate(prefab) as GameObject;

t.name = "clone\_prefab1";

Debug.Log("name = " + prefab.name);

在上面的例子中尝试过建立多个Resources文件，实际上都可以找到prefab，注意Resources.Load不需要添加资源后缀名

**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就是内部数据读取万后自动创建了一个assetBundle而已

Create完成以后，等于把硬盘或者网络的一个文件读到内存的 一个区域，这时候只是一个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的内存镜像中读取并创建一个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(false)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AssetBundle文件的内存镜像并销毁所有用 Load创建的Asset内存对象

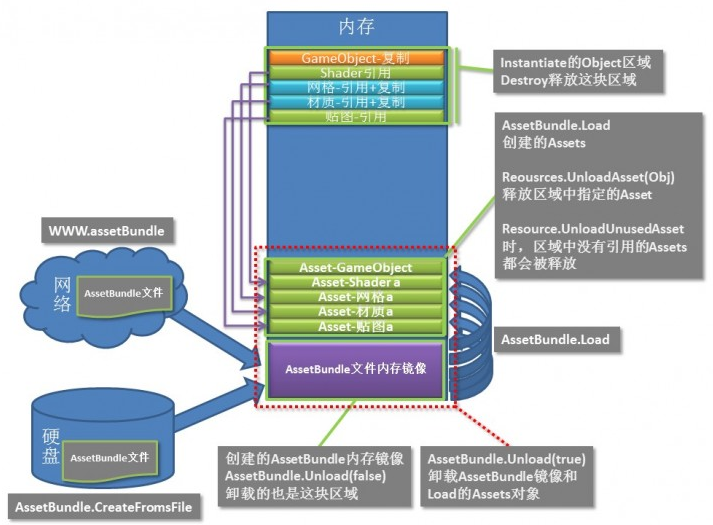
一个prefab从assetbundle里load出来，里面可能包括：GameObject Transform Mesh Texture Material Shader Script和各种其他的Assets，你Instaniate一个Prefab，是一个对Asset进行Clone（复制）+引用结合的过程，GameObject transform是clone新生成的，其他mesh texture material shader等，这些是纯引用关系，包括：Texture和TerrainData，还有应用和复制同时存在的，包括：Mesh material phycisMaterial。引用的Asset对象不会被复制，只是简单的指针指向已经load的asset对象。

专门要提的一个特殊的东西：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，并没有调用代码，光Class的定义脚本是不会工作的。其实Unity引用就是那个调用代码，Clone一个script asset等于new一个class实例，实例才会完成工作。把它挂到Unity主线程的调用链里去，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。在new Class这个剁成中，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

你可以再Instaniate一个同样的Prefab，还是这套mesh texture material shader，这时候会有新的GameObject，但是不会创建新的应用对象比如Texture

**所以，load出来的Assets其实就是一个数据源，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，只是释放了那些clone对象，并不会释放应用对象和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就成了没有应用的游离数据块了，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



内存管理

创建时：

先建立一个AssetBundle，无论是从www还是memory

用assetBundle.Load加载需要的Asset

加载完之后立即调用AssetBUndle.Unload(flase)，释放AssetBundle文件本身的内存镜像，但不销毁加载的Asset对象（这样就不用保存AssetBundle的应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，释放已经没有应用的Asset

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

**用AssetBundle加载asset一样可以用Resources.UnloadUnusedAssets卸载，但必须先AssetBundle.Unload，才会被识别为无用的asset**

Unity动态加载prefab方式的差异：

其实在unity中存在三种加载prefab的方式

1. 静态引用，建立一个public的变量，在inspector里把prefab拉上去，用的时候instantiate
2. Resource.Load，Load以后instaniate
3. AssetBundle.Load，Load以后instantiate

这三种方式有细节差异，前两种方式，引用对象texture是在instantiate时加载，而assetbundle.load会把prefab的全部assets都加载，instantiate时只是生成clone，多有前两种方式，除非提前加载相关引用对象，否则第一次instantiate时会包含加载引用类assets的操作，导致第一次加载的时间有延时。具体看<https://blog.csdn.net/u011866450/article/details/51220764>

几种AssetBundle创建方式的差异：

CreateFromFile：这种方式不会把整个硬盘的AssetBundle文件都加载到内存来，而是类似建立一个文件操作句柄和缓冲区，需要时才实时Load，，所以这种加载方式是最节省资源的，基本上AssetBundle本身不占什么内存，只需要Asset对象的内存，可惜只能在PC/Mac Standalone程序重视会用

CreateFromMemory和[www.assetBundle](http://www.assetBundle)：这两种方式AssetBundle文件会整个镜像于内存中，理论文件多大就需要多大的内存，之后load时还要占用额外的内存去生成Asset对象

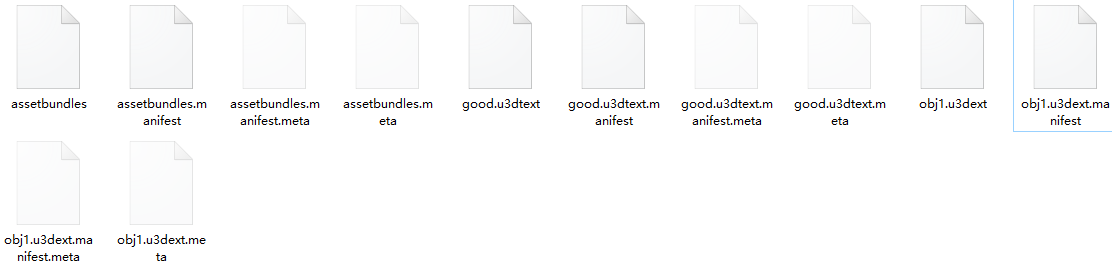
**什么时候才是UnusedAssets?**  
看一个例子：  
Object obj = Resources.Load("MyPrefab");  
GameObject instance = Instantiate(obj) as GameObject;  
.........  
Destroy(instance);  
创建随后销毁了一个Prefab实例，这时候 MyPrefab已经没有被实际的物体引用了，但如果这时：  
Resources.UnloadUnusedAssets();  
内存并没有被释放，原因：MyPrefab还被这个变量obj所引用  
这时候：  
obj  = null;  
Resources.UnloadUnusedAssets();  
这样才能真正释放Assets对象  
所以：UnusedAssets不但要没有被实际物体引用，也要没有被生命周期内的变量所引用，才可以理解为 Unused(引用计数为0)  
所以所以：如果你用个全局变量保存你Load的Assets，又没有显式的设为null，那在这个变量失效前你无论如何UnloadUnusedAssets也释放不了那些Assets的。如果你这些Assets又不是从磁盘加载的，那除了UnloadUnusedAssets或者加载新场景以外没有其他方式可以卸载之。

**Unity5的打包**

资源下方有两个下拉框，左边用于指定AssetBundle的名字，后边的用于指定AssetBundle Variants的名字

BuildPipeline.BuildAssetBundles(Application.dataPath + "/assetbundles", BuildAssetBundleOptions.None, EditorUserBuildSettings.activeBuildTarget);

这样会在assetbundles文件夹中看到打包的文件



其中会产生一个与文件夹名字相同的manifest文件，其他每个资源都产生一个manifest文件

获取所有assetBundle

[MenuItem("Assets/GetAllAssetBundle")]

public static void GetAllAssetBundle()

{

string[] names = AssetDatabase.GetAllAssetBundleNames();

for (int i = 0, size = names.Length; i < size; i++)

{

Debug.Log(string.Format("i = {0}, AssetBundle name = {1}", i, names[i]));

string[] paths = AssetDatabase.GetAssetPathsFromAssetBundle(names[i]);

foreach (var path in paths)

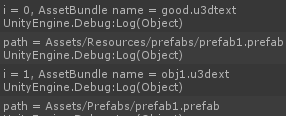
{

Debug.Log("path = " + path);

}

}

}



**AssetBundle的压缩**

你可以不对AssetBundles进行压缩，这样虽然文件很大，但是一旦下载好之后访问是最快的，通常都会进行压缩，并且Unity的AssetBundle支持LZMA和LZ4两种格式的压缩

LZMA格式：这是unity打包成AssetBundle时的默认格式，会将序列化数据压缩成一个LZMA流，使用时需要整体解包。优点是打包后体积小，缺点是解包时间长导致加载时间长。

LZ4格式：相对于LZMA会生成更大的压缩文件，但优点是使用时不需要整体解压。LZ4是一种基于chunk的算法，加载对象时只有相应的chunk会被解压，该格式需要unity5.3以上的版本，以前的版本并不支持，BuildAssetBundleOptions选择ChunkBasedCompression

**什么是AssetBundle？**

AssetBundle只是一种使用LZMA压缩方式压缩的资源文件，至于它的后缀名是什么，一点关系都没有，可以自己定。

AssetBundle打包的时，可以指定一个mainAsset，那么加载完之后就可以通过AssetBundle.mainAsset来获取到了，也可以不指定mainAsset，直接打包一堆内容进去，然后加载通过AssetBundle.LoadAsset指定名字读取出来。在资源之间，存在着依赖关系，你可以把资源拆分得很细，比如一个模型，由网格模型、材质、贴图构成，可以将他们分开，各自打包成压缩文件。当Unity需要加载使用时，把该模型的所有依赖的小资源都加载起来，然后根据依赖关系组装就变回我们看到的资源了。

**打包AssetBundle**

Unity5.x大幅度简化了AssetBundle的打包过程，理论上只需要调用一个API既可以完成打包, BuildPipeline.BuildAssetBundles

但是资源之间的依赖关系，避免资源重复打包的问题还是要自己解决

Unity5r.x中会将设置AssetBundleName相同的资源打包到一起，所以我们的打包过程其实就是把需要打包到一起的资源设置成相同的AssetBundleName，为每个资源设置AssetBundleName，最后调用打包API即可完成打包

**AssetBundle的依赖结构**

要说明依赖关系，还需要使用一个例子，一个模型，分为了网络模型、材质、贴图。那么它们是怎样依赖的呢？然后在unity5的打包里面，它们是怎样表现出依赖关系的呢？