Unity几种动态加载Prefab方式的差异:

其实存在3种加载prefab的方式：

一是静态引用，建一个public的变量，在Inspector里把prefab拉上去，用的时候instantiate

二是Resource.Load，Load以后instantiate

三是AssetBundle.Load,Load以后instantiate

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perfab的全部assets都加载，instantiate时只是生成Clone。所以前两种方式，除非你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lag。官方论坛有人说Resources.Load和静态引用是会把所有资源都预先加载的，反复测试的结果，静态引用和Resources.Load也是OnDemand的，用到时才会加载。

几种AssetBundle创建方式的差异:

CreateFromFile:这种方式不会把整个硬盘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程序中使用。

CreateFromMemory和www.assetBundle:这两种方式AssetBundle文件会整个镜像于内存中，理论上文件多大就需要多大的内存，之后Load时还要占用额外内存去生成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

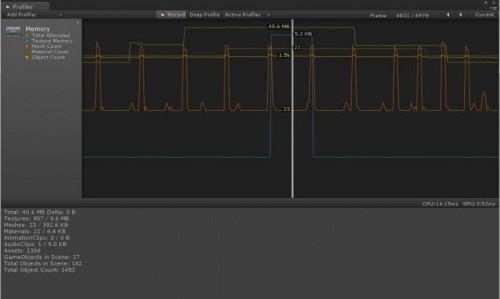
所以：UnusedAssets不但要没有被实际物体引用，也要没有被生命周期内的变量所引用，才可以理解为 Unused(引用计数为0)

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。

一个复杂的例子，代码很丑陋实际也不可能这样做，只是为了加深理解

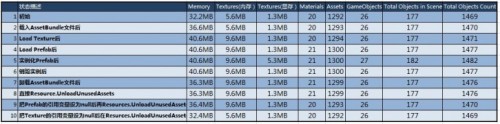
IEnumerator OnClick()  
{  
Resources.UnloadUnusedAssets();//清干净以免影响测试效果  
yield return new WaitForSeconds(3);  
float wait = 0.5f;  
//用www读取一个assetBundle,里面是一个Unity基本球体和带一张大贴图的材质，是一个Prefab  
WWW aa = new WWW(@"file://SpherePrefab.unity3d");  
yield return aa;  
AssetBundle asset = aa.assetBundle;  
yield return new WaitForSeconds(wait);//每步都等待0.5s以便于分析结果  
Texture tt = asset.Load("BallTexture") as  Texture;//加载贴图  
yield return new WaitForSeconds(wait);  
GameObject ba = asset.Load("SpherePrefab") as  GameObject;//加载Prefab  
yield return new WaitForSeconds(wait);  
GameObject obj1 = Instantiate(ba) as GameObject;//生成实例  
yield return new WaitForSeconds(wait);  
Destroy(obj1);//销毁实例  
yield return new WaitForSeconds(wait);  
asset.Unload(false);//卸载Assetbundle  
yield return new WaitForSeconds(wait);  
Resources.UnloadUnusedAssets();//卸载无用资源  
yield return new WaitForSeconds(wait);  
ba = null;//将prefab引用置为空以后卸无用载资源  
Resources.UnloadUnusedAssets();  
yield return new WaitForSeconds(wait);  
tt = null;//将texture引用置为空以后卸载无用资源  
Resources.UnloadUnusedAssets();  
}

这是测试结果的内存Profile曲线图

[](http://unity3d.9ria.com/?attachment_id=1863)

很经典的对称造型，用多少释放多少。

这是各阶段的内存和其他数据变化

[](http://unity3d.9ria.com/?attachment_id=1864)

说明:

1 初始状态

2 载入AssetBundle文件后，内存多了文件镜像，用量上升，Total Object和Assets增加1（AssetBundle也是object)

3 载入Texture后，内存继续上升，因为多了Texture Asset,Total Objects和Assets增加1

4 载入Prefab后，内存无明显变化，因为最占内存的Texture已经加载，Materials上升是因为多了Prefab的材质，Total Objects和Assets增加6，因为 Perfab 包含很多 Components

5 实例化Prefab以后，显存的Texture Memory、GameObjectTotal、Objects in Scene上升，都是因为实例化了一个可视的对象

6 销毁实例后，上一步的变化还原，很好理解

7 卸载AssetBundle文件后，AssetBundle文件镜像占用的内存被释放，相应的Assets和Total Objects Count也减1

8 直接Resources.UnloadUnusedAssets,没有任何变化，因为所有Assets引用并没有清空

9 把Prefab引用变量设为null以后，整个Prefab除了Texture外都没有任何引用了，所以被UnloadUnusedAssets销毁,Assets和Total Objects Count减6

10 再把Texture的引用变量设为null,之后也被UnloadUnusedAssets销毁，内存被释放，assets和Total Objects Count减1，基本还原到初始状态

从中也可以看出：

Texture加载以后是到内存，显示的时候才进入显存的Texture Memory。

所有的东西基础都是Object

Load的是Asset,Instantiate的是GameObject和Object in Scene

Load的Asset要Unload,new的或者Instantiate的object可以Destroy

http://unity3d.9ria.com/?p=1862