**Unity3D占用内存太大怎么解决呢?**

Posted on 2013年01月10日 by U3d / [Unity3D脚本/插件](http://www.unitymanual.com/category/script)/被围观 1,555 次

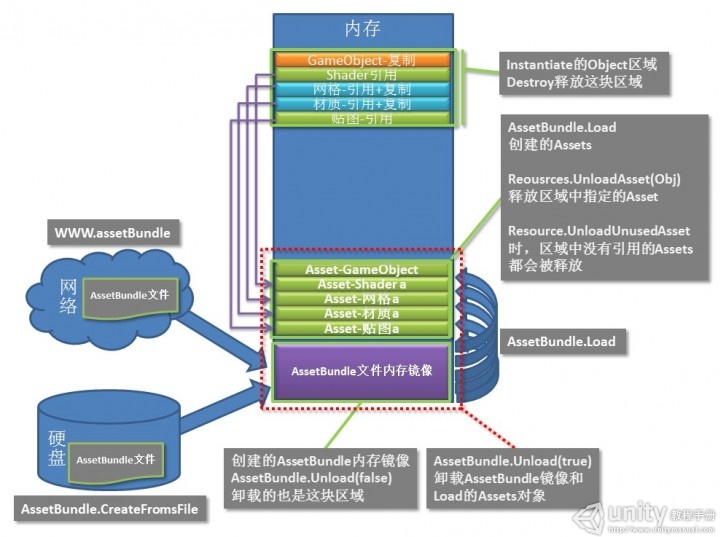
最近网友通过网站搜索Unity3D在手机及其他平台下占用内存太大. 这里写下关于Unity3D对于内存的管理与优化.

Unity3D 里有两种动态加载机制：一个是Resources.Load，另外一个通过AssetBundle,其实两者区别不大。 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Unity3D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 Instaniate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。把他挂到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，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Instaniate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：

[](http://www.unitymanual.com/wp-content/uploads/2013/01/2_6507_88c2ecf9476d602.jpg)

Unity3D占用内存太大怎么解决呢?

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

举两个例子帮助理解  
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 = Instaniate(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Instaniate 一个Prefab的时候都会卡一下，因为在你第一次Instaniate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。

顺便提一下几种加载方式的区别:  
其实存在3种加载方式：  
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assets的操作，导致第一次加载的lag。

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);  
}  
这样卸载比较快

Hog的评论引用：

感觉这是Unity内存管理暗黑和混乱的地方，特别是牵扯到Texture  
我最近也一直在测试这些用AssetBundle加载的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。**比较保险的做法是  
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  
用完后立即AssetBundle.Unload(false),关闭AssetBundle但不摧毁创建的对象和引用  
销毁时：  
对Instantiate的对象进行Destroy  
在合适的地方调用Resources.UnloadUnusedAssets,释放已经没有引用的Asset.  
如果需要立即释放加上GC.Collect()  
这样可以保证内存始终被及时释放  
只要你Unload过的AssetBundle,那些创建的对象和引用都会在LoadLevel时被自动释放。

**全面理解Unity加载和内存管理机制之二：进一步深入和细节**  
**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();

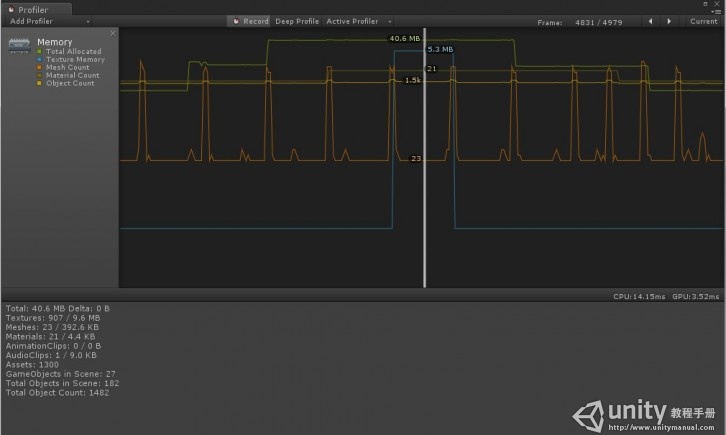
yield return new WaitForSeconds(wait);

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

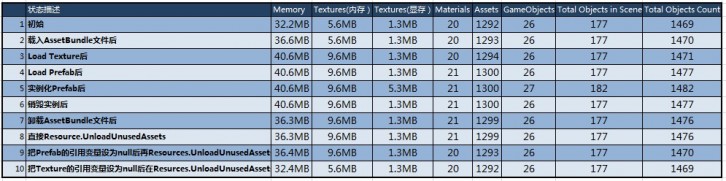
[](http://www.unitymanual.com/wp-content/uploads/2013/01/29.jpg)

Unity3D占用内存太大怎么解决呢?

图片:p12.jpg

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://www.unitymanual.com/wp-content/uploads/2013/01/35.jpg)

Unity3D占用内存太大怎么解决呢?

图片:p13.jpg

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**

**Unity 3D中的内存管理**

Unity3D在内存占用上一直被人诟病，特别是对于面向移动设备的游戏开发，动辄内存占用飙上一两百兆，导致内存资源耗尽，从而被系统强退造成极 差的体验。类似这种情况并不少见，但是绝大部分都是可以避免的。虽然理论上Unity的内存管理系统应当为开发者分忧解难，让大家投身到更有意义的事情中 去，但是对于Unity对内存的管理方式，官方文档中并没有太多的说明，基本需要依靠自己摸索。最近在接手的项目中存在严重的内存问题，在参照文档和 Unity Answer众多猜测和证实之后，稍微总结了下Unity中的内存的分配和管理的基本方式，在此共享。

虽然Unity标榜自己的内存使用全都是“Managed Memory”，但是事实上你必须正确地使用内存，以保证回收机制正确运行。如果没有做应当做的事情，那么场景和代码很有可能造成很多非必要内存的占用， 这也是很多Unity开发者抱怨内存占用太大的原因。接下来我会介绍Unity使用内存的种类，以及相应每个种类的优化和使用的技巧。遵循使用原则，可以 让非必要资源尽快得到释放，从而降低内存占用。

**Unity中的内存种类**

实际上Unity游戏使用的内存一共有三种：程序代码、托管堆（Managed Heap）以及本机堆（Native Heap）。

程序代码包括了所有的Unity引擎，使用的库，以及你所写的所有的游戏代码。在编译后，得到的运行文件将会被加载到设备中执行，并占用一定内存。

这部分内存实际上是没有办法去“管理”的，它们将在内存中从一开始到最后一直存在。一个空的Unity默认场景，什么代码都不放，在iOS设备上占 用内存应该在17MB左右，而加上一些自己的代码很容易就飙到20MB左右。想要减少这部分内存的使用，能做的就是减少使用的库，稍后再说。

托管堆是被Mono使用的一部分内存。Mono项目一个开源的.net框架的一种实现，对于Unity开发，其实充当了基本类库的角色。

托管堆用来存放类的实例（比如用new生成的列表，实例中的各种声明的变量等）。“托管”的意思是Mono“应该”自动地改变堆的大小来适应你所需要的内存，

并且定时地使用垃圾回收（Garbage Collect）来释放已经不需要的内存。关键在于，有时候你会忘记清除对已经不需要再使用的内存的引用，

从而导致Mono认为这块内存一直有用，而无法回收。

最后，本机堆是Unity引擎进行申请和操作的地方，比如贴图，音效，关卡数据等。Unity使用了自己的一套内存管理机制来使这块内存具有和托管堆类似的功能。

基本理念是，如果在这个关卡里需要某个资源，那么在需要时就加载，之后在没有任何引用时进行卸载。听起来很美好也和托管堆一样，

但是由于Unity有一套自动加载和卸载资源的机制，让两者变得差别很大。自动加载资源可以为开发者省不少事儿，

但是同时也意味着开发者失去了手动管理所有加载资源的权力，这非常容易导致大量的内存占用（贴图什么的你懂的），

也是Unity给人留下“吃内存”印象的罪魁祸首。

**优化程序代码的内存占用**

这部分的优化相对简单，因为能做的事情并不多：主要就是减少打包时的引用库，改一改build设置即可。

对于一个新项目来说不会有太大问题，但是如果是已经存在的项目，可能改变会导致原来所需要的库的缺失（虽说一般来说这种可能性不大），

因此有可能无法做到最优。

当使用Unity开发时，默认的Mono包含库可以说大部分用不上，在Player Setting（Edit->Project Setting->Player或者Shift+Ctrl(Command)+B里的Player Setting按钮）

面板里，将最下方的Optimization栏目中“Api Compatibility Level”选为.NET 2.0 Subset，表示你只会使用到部分的.NET 2.0 Subset，不需要Unity将全部.NET的Api包含进去。接下来的“Stripping Level”表示从build的库中剥离的力度，每一个剥离选项都将从打包好的库中去掉一部分内容。你需要保证你的代码没有用到这部分被剥离的功能，

选为“Use micro mscorlib”的话将使用最小的库（一般来说也没啥问题，不行的话可以试试之前的两个）。库剥离可以极大地降低打包后的程序的尺寸以及程序代码的内存占用，唯一的缺点是这个功能只支持Pro版的Unity。

这部分优化的力度需要根据代码所用到的.NET的功能来进行调整，有可能不能使用Subset或者最大的剥离力度。

如果超出了限度，很可能会在需要该功能时因为找不到相应的库而crash掉（iOS的话很可能在Xcode编译时就报错了）。

比较好地解决方案是仍然用最强的剥离，并辅以较小的第三方的类库来完成所需功能。

一个最常见问题是最大剥离时Sysytem.Xml是不被Subset和micro支持的，如果只是为了xml，完全可以导入一个轻量级的xml库来解决依赖（Unity官方推荐这个）。

关于每个设定对应支持的库的详细列表，可以在这里找到。关于每个剥离级别到底做了什么，Unity的文档也有说明。

实际上，在游戏开发中绝大多数被剥离的功能使用不上的，因此不管如何，库剥离的优化方法都值得一试。

**托管堆优化**

Unity有一篇不错的关于托管堆代码如何写比较好的说明，在此基础上我个人有一些补充。

首先需要明确，托管堆中存储的是你在你的代码中申请的内存（不论是用js，C#还是Boo写的）。

一般来说，无非是new或者Instantiate两种生成object的方法（事实上Instantiate中也是调用了new）。

在接收到alloc请求后，托管堆在其上为要新生成的对象实例以及其实例变量分配内存，如果可用空间不足，则向系统申请更多空间。

当你使用完一个实例对象之后，通常来说在脚本中就不会再有对该对象的引用了（这包括将变量设置为null或其他引用，超出了变量的作用域，

或者对Unity对象发送Destory()）。在每隔一段时间，Mono的垃圾回收机制将检测内存，将没有再被引用的内存释放回收。总的来说，

你要做的就是在尽可能早的时间将不需要的引用去除掉，这样回收机制才能正确地把不需要的内存清理出来。但是需要注意在内存清理时有可能造成游戏的短时间卡顿，

这将会很影响游戏体验，因此如果有大量的内存回收工作要进行的话，需要尽量选择合适的时间。

如果在你的游戏里，有特别多的类似实例，并需要对它们经常发送Destroy()的话，游戏性能上会相当难看。比如小熊推金币中的金币实例，按理说每枚金币落下台子后

都需要对其Destory()，然后新的金币进入台子时又需要Instantiate，这对性能是极大的浪费。一种通常的做法是在不需要时，不摧毁这个GameObject，而只是隐藏它，

并将其放入一个重用数组中。之后需要时，再从重用数组中找到可用的实例并显示。这将极大地改善游戏的性能，相应的代价是消耗部分内存，一般来说这是可以接受的。

关于对象重用，可以参考Unity关于内存方面的文档中Reusable Object Pools部分，或者Prime31有一个是用Linq来建立重用池的视频教程（Youtube，需要翻墙，上，下）。

如果不是必要，应该在游戏进行的过程中尽量减少对GameObject的Instantiate()和Destroy()调用，因为对计算资源会有很大消耗。在便携设备上短时间大量生成和摧毁物体的

话，很容易造成瞬时卡顿。如果内存没有问题的话，尽量选择先将他们收集起来，然后在合适的时候（比如按暂停键或者是关卡切换），将它们批量地销毁并 且回收内存。Mono的内存回收会在后台自动进行，系统会选择合适的时间进行垃圾回收。在合适的时候，也可以手动地调用 System.GC.Collect()来建议系统进行一次垃圾回收。

要注意的是这里的调用真的仅仅只是建议，可能系统会在一段时间后在进行回收，也可能完全不理会这条请求，不过在大部分时间里，这个调用还是靠谱的。

**本机堆的优化**

当你加载完成一个Unity的scene的时候，scene中的所有用到的asset（包括Hierarchy中所有GameObject上以及脚本中赋值了的的材质，贴图，动画，声音等素材），

都会被自动加载（这正是Unity的智能之处）。也就是说，当关卡呈现在用户面前的时候，所有Unity编辑器能认识的本关卡的资源都已经被预先加 入内存了，这样在本关卡中，用户将有良好的体验，不论是更换贴图，声音，还是播放动画时，都不会有额外的加载，这样的代价是内存占用将变多。Unity最 初的设计目的还是面向台式机，

几乎无限的内存和虚拟内存使得这样的占用似乎不是问题，但是这样的内存策略在之后移动平台的兴起和大量移动设备游戏的制作中出现了弊端，因为移动设 备能使用的资源始终非常有限。因此在面向移动设备游戏的制作时，尽量减少在Hierarchy对资源的直接引用，而是使用Resource.Load的方 法，在需要的时候从硬盘中读取资源，

在使用后用Resource.UnloadAsset()和Resources.UnloadUnusedAssets()尽快将其卸载掉。总之，这里是一个处理时间和占用内存空间的trade off，

如何达到最好的效果没有标准答案，需要自己权衡。

在关卡结束的时候，这个关卡中所使用的所有资源将会被卸载掉（除非被标记了DontDestroyOnLoad）的资源。注意不仅是DontDestroyOnLoad的资源本身，

其相关的所有资源在关卡切换时都不会被卸载。DontDestroyOnLoad一般被用来在关卡之间保存一些玩家的状态，比如分数，级别等偏向文 本的信息。如果DontDestroyOnLoad了一个包含很多资源（比如大量贴图或者声音等大内存占用的东西）的话，这部分资源在场景切换时无法卸 载，将一直占用内存，

这种情况应该尽量避免。

另外一种需要注意的情况是脚本中对资源的引用。大部分脚本将在场景转换时随之失效并被回收，但是，在场景之间被保持的脚本不在此列（通常情况是被附 着在DontDestroyOnLoad的GameObject上了）。而这些脚本很可能含有对其他物体的Component或者资源的引用，这样相关的 资源就都得不到释放，

这绝对是不想要的情况。另外，static的单例（singleton）在场景切换时也不会被摧毁，同样地，如果这种单例含有大量的对资源的引用，也会成为大问题。

因此，尽量减少代码的耦合和对其他脚本的依赖是十分有必要的。如果确实无法避免这种情况，那应当手动地对这些不再使用的引用对象调用Destroy()

或者将其设置为null。这样在垃圾回收的时候，这些内存将被认为已经无用而被回收。

需要注意的是，Unity在一个场景开始时，根据场景构成和引用关系所自动读取的资源，只有在读取一个新的场景或者reset当前场景时，才会得到清理。

因此这部分内存占用是不可避免的。在小内存环境中，这部分初始内存的占用十分重要，因为它决定了你的关卡是否能够被正常加载。因此在计算资源充足

或是关卡开始之后还有机会进行加载时，尽量减少Hierarchy中的引用，变为手动用Resource.Load，将大大减少内存占用。在 Resource.UnloadAsset()和Resources.UnloadUnusedAssets()时，只有那些真正没有任何引用指向的资源 会被回收，因此请确保在资源不再使用时，将所有对该资源的引用设置为null或者Destroy。

同样需要注意，这两个Unload方法仅仅对Resource.Load拿到的资源有效，而不能回收任何场景开始时自动加载的资源。与此类似的还有 AssetBundle的Load和Unload方法，灵活使用这些手动自愿加载和卸载的方法，是优化Unity内存占用的不二法则～

总之这些就是关于Unity3d优化细节,具体还是查看Unity3D的技术手册,以便实现最大的优化.