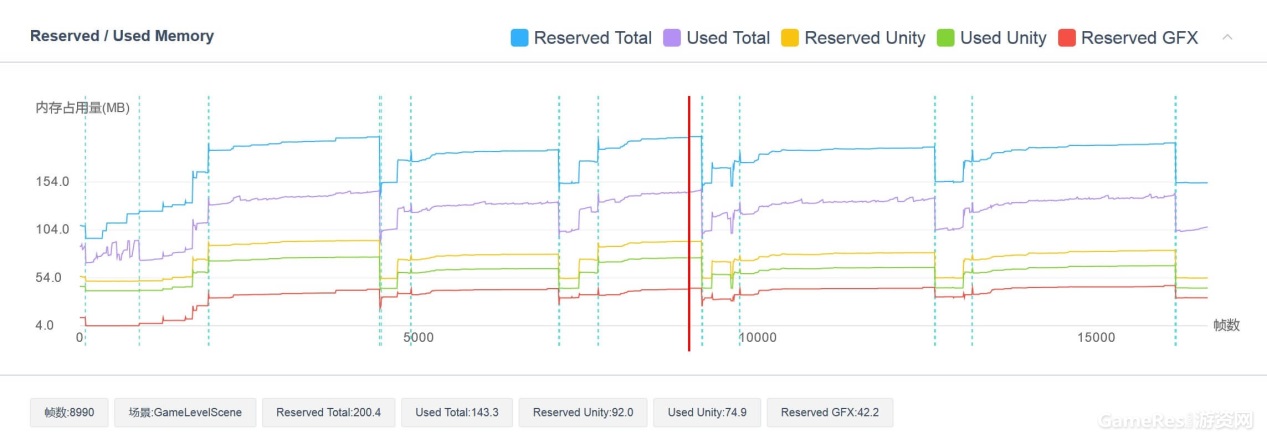
<http://www.58kaifa.com/article/537>

Android PSS/iOS Instrument

其中，蓝线和紫线的分离情况，反映了无效堆内存的分配大小。比如，图中所选中时刻，蓝线的Reserved Total为当前项目所占据的总物理内存，而紫线的Used Total为当前项目所使用的总物理内存，这说明当前项目中的空闲内存为57.1MB（200.4-143.3），而这其中主要由两部分组成，空闲的Unity引擎内存和无效的Mono堆内存。其中，空闲的Unity内存为17.1MB（92.0-74.9），所以当前所选帧的无效Mono堆内存为40.0MB。

内存的开销无外乎以下三大部分：

1. 资源内存占用；2.引擎模块自身内存占用；3.托管堆内存占用。

一、资源内存占用：

在一个较为复杂的大中型项目中，资源的内存占用往往占据了总体内存的70%以上。

纹理（Texture）

网格（Mesh）

动画片段（AnimationClip）

音频片段（AudioClip）

材质（Material）

着色器（Shader）

字体资源（Font）

文本资源（Text Asset）等等。

其中，**纹理、网格、动画片段和音频片段**则是最容易造成较大内存开销的资源。

纹理：

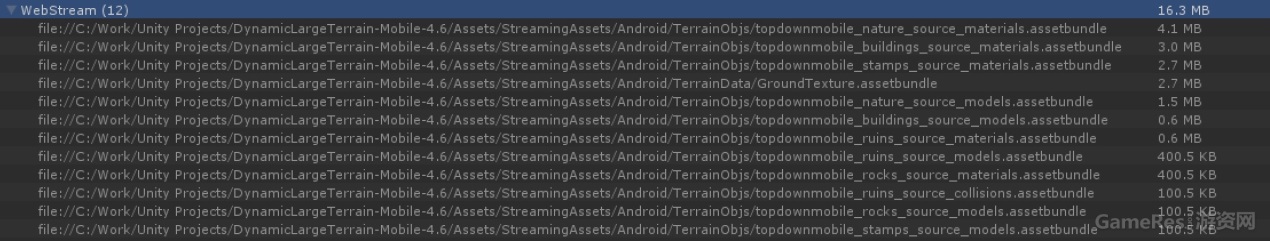
1. 尽量减少纹理的色差范围，使其尽可能使用硬件支持的压缩格式进行储存。
2. 尺寸越小越好

Mesh：

Mesh进行Draw Call Batching操作的话，那么将很有可能进一步增大总体内存的占用。比如，100个Mesh进行拼合，其中99个Mesh均没有Color、Tangent等属性，剩下一个则包含有Color、Normal和Tangent属性，那么Mesh拼合后，CombinedMesh中将为每个Mesh来添加上此三个顶点属性，进而造成很大的内存开销。正因如此，我们在UWA测评报告中为每个Mesh展示了其Normal、Color和Tangent属性的具体使用情况，研发团队可以直接针对每种属性进行排序查看，直接定位出现冗余数据的资源。

二、引擎模块自身占用

真正占据较大内存开销的是这两处：WebStream 和 SerializedFile。其**绝大部分的内存分配则是由AssetBundle加载资源所致**。其绝大部分的内存分配则是由AssetBundle加载资源所致。简单言之，当您使用new WWW或CreateFromMemory来加载AssetBundle时，Unity引擎会加载原始数据到内存中并对其进行解压，而WebStream的大小则是AssetBundle原始文件大小 + 解压后的数据大小 + DecompressionBuffer(0.5MB)。同时，由于Unity 5.3版本之前的AssetBundle文件为LZMA压缩，其压缩比类似于Zip（20%-25%）,所以对于一个1MB的原始AssetBundle文件，其加载后WebStream的大小则可能是5~6MB，因此，当项目中存在通过new WWW加载多个AssetBundle文件，且AssetBundle又无法及时释放时，WebStream的内存可能会很大，这是研发团队需要时刻关注的。



对于SerializedFile，则是当你使用LoadFromCacheOrDownload、CreateFromFile或new WWW本地AssetBundle文件时产生的序列化文件。

对于WebStream和SerializedFile，你需要关注以下两点：  
  
　　是否存在AssetBundle没有被清理干净的情况。开发团队可以通过Unity Profiler直接查看其使用具体的使用情况，并确定Take Sample时AssetBundle的存在是否合理；  
  
　　对于占用WebStream较大的AssetBundle文件（如UI Atlas相关的AssetBundle文件等），建议使用LoadFromCacheOrDownLoad或CreateFromFile来进行替换，即将解压后的AssetBundle数据存储于本地Cache中进行使用。这种做法非常适合于内存特别吃紧的项目，即通过本地的磁盘空间来换取内存空间。  
  
　　注意：关于AssetBundle的详细管理机制，建议查看我们之前的AssetBundle技术文章。

三、托管堆内存占用

对于目前绝大多数基于Unity引擎开发的项目而言，其托管堆内存是由Mono分配和管理的。

“托管” 的本意是Mono可以自动地改变堆的大小来适应你所需要的内存，并且适时地调用垃圾回收（Garbage Collection）操作来释放已经不需要的内存，从而降低开发人员在代码内存管理方面的门槛。

因为目前Unity所使用的Mono版本存在一个很严重的问题，即：Mono的堆内存一旦分配，就不会返还给系统。

UIPanel.LateUpdate。这是NGUI中CPU和堆内存开销最大的函数。它本身只是一个函数，但NGUI的大量使用使它逐渐成为了一个不可忽视规则。该函数的堆内存分配和自身CPU开销，其根源上是一致的，即是由UI网格的重建造成。因此，**其对应的优化方法是直接查看CPU篇中的UI模块讲解。**

**主要标准：**

1、150M的总体内存标准

将Reserved Total设定在150MB，这是Unity引擎的自身内存分配，以保证App在使用到的系统库后，其OS中的整体内存也在200MB以下。

2、内存分配原则

　纹理资源： 50 MB 动画片段：15 MB 音频片段：15 MB  
　　 网格资源： 20 MB Mono堆内存：40 MB 其他：10 MB

内存泄漏：

内存泄露是开发人员在项目研发过程中最常见也最不愿遇到的问题。就目前来看，大家对于判断项目是否存在内存泄露仍然存在一些误区：  
　　**误区一**  
　　我的项目进出场景前后内存回落不一致，比如进入场景后，内存增加40MB，出来后下降30MB，仍有10MB内存没有返回给系统，即说明内存存在泄露情况。

**因为造成内存不能完全回落的情况有很多，比如资源加载后常驻内存以备后续使用、Mono堆内存的只升不降等等**

**误区二**

我的项目在进出场景前后，Unity Profiler中内存回落正常，但Android的PSS数值并没有完全回落（出场景后的PSS值高于进场景前的PSS值），即说明内存存在泄露情况。

一般来说，我们推荐的判断内存是否泄漏的方法如下：

1. 检查资源的使用周期
2. 通过Profiler来检测WebStream或SerializedFile的使用情况AssetBundle的管理不当也会造成一定的内存泄露，即上一场景中使用的AssetBundle在场景切换时没有被卸载掉，而被带入到了下一场场景中。对于这种情况，建议直接通过Profiler Memory中的Take Sample来对其进行检测，通过直接查看WebStream或SerializedFile中的AssetBundle名称，即可判断是否存在“泄露”情况。
3. 通过Android PSS/iOS Instrument。

理论上来说，多次切换同样的场景，如果Profiler中显示的Unity内存回落正常，那么其PSS/Instrument的内存数值波动范围也是趋于稳定的，但如果出现了PSS/Instrument内存持续增长的情况，则需要大家注意了。这可能有两种可能：  
  
　　Unity引擎自身的内存泄露问题。这种概率很小，之前仅在少数版本中出现过。  
  
　　第三方插件在使用时出现了内存泄露。这种概率较大，因为Profiler仅能对Unity自身的内存进行监控，而无法检测到第三方库的内存分配情况。因此，在出现上述内存问题时，建议大家先对自身使用的第三方库进行排查。