3[**Unity教程之记一次基于Unity的Profiler性能分析**](http://www.unity.5helpyou.com/2791.html)

本篇unity教程我们来学习下Unity的Profiler性能分析

A. WaitForTargetFPS:  
      Vsync(垂直同步)功能所，即显示当前帧的CPU等待时间  
   B. Overhead：  
      Profiler总体时间-所有单项的记录时间总和。用于记录尚不明确的时间消耗，以帮助进一步完善Profiler的统计。  
        C. Physics.Simulate：  
      当前帧物理模拟的CPU占用时间。  
   D. Camera.Render：  
      相机渲染准备工作的CPU占用量  
   E. RenderTexture.SetActive：  
      设置RenderTexture操作.  
      底层实现：1.比对当前帧与前一帧的ColorSurface和DepthSurface.  
               2.如果这两个Buffer一致则不生成新的RT，否则则生成新的RT，并设置与之相对应的Viewport和空间转换矩阵.  
   F. Monobehaviour.OnMouse\_ ：  
      用于检测鼠标的输入消息接收和反馈，主要包括：SendMouseEvents和DoSendMouseEvents。（只要Edtor开起来，这个就会存在）  
   G. HandleUtility.SetViewInfo：  
      仅用于Editor中，作用是将GUI和Editor中的显示看起来与发布版本的显示一致。  
H. GUI.Repaint：  
      GUI的重绘(说明在有使用原生的OnGUI)  
   I. Event.Internal\_MakeMasterEventCurrent：  
      负责GUI的消息传送  
   J. Cleanup Unused Cached Data：  
      清空无用的缓存数据，主要包括RenderBuffer的垃圾回收和TextRendering的垃圾回收。  
         1.RenderTexture.GarbageCollectTemporary:存在于RenderBuffer的垃圾回收中，清除临时的FreeTexture.  
         2.TextRendering.Cleanup:TextMesh的垃圾回收操作  
   K. Application.Integrate Assets in Background：  
      遍历预加载的线程队列并完成加载，同时，完成纹理的加载、Substance的Update等.  
   L. Application.LoadLevelAsync Integrate：  
      加载场景的CPU占用，通常如果此项时间长的话70%的可能是Texture过长导致.  
   M. UnloadScene：  
      卸载场景中的GameObjects、Component和GameManager，一般用在切换场景时.  
   N. CollectGameObjectObjects：  
      执行上面M项的同时，会将场景中的GameObject和Component聚集到一个Array中.然后执行下面的Destroy.  
   O. Destroy：  
      删除GameObject和Component的CPU占用.  
   P. AssetBundle.LoadAsync Integrate：  
      多线程加载AwakeQueue中的内容，即多线程执行资源的AwakeFromLoad函数.  
   Q. Loading.AwakeFromLoad：  
      在资源被加载后调用，对每种资源进行与其对应用处理.  
  
2. CPU Usage  
   A. Device.Present:  
      device.PresentFrame的耗时显示，该选项出现在发布版本中.  
   B. Graphics.PresentAndSync：  
      GPU上的显示和垂直同步耗时.该选项出现在发布版本中.  
   C. Mesh.DrawVBO：  
      GPU中关于Mesh的Vertex Buffer Object的渲染耗时.  
   D. Shader.Parse：  
      资源加入后引擎对Shader的解析过程.  
   E. Shader.CreateGPUProgram：  
      根据当前设备支持的图形库来建立GPU工程.

**3. Memory Profiler**

**A. Used Total:  
      当前帧的Unity内存、Mono内存、GfxDriver内存、Profiler内存的总和.  
   B. Reserved Total:  
      系统在当前帧的申请内存.  
   C. Total System Memory Usage:  
      当前帧的虚拟内存使用量.（通常是我们当前使用内存的1.5~3倍)  
   D. GameObjects in Scene:  
      当前帧场景中的GameObject数量.  
   E. Total Objects in Scene:  
      当前帧场景中的Object数量(除GameObject外，还有Component等).  
   F. Total Object Count:  
      Object数据 + Asset数量.**

**4. Detail Memory Profiler  
   A. Assets:  
      Texture2d:记录当前帧内存中所使用的纹理资源情况，包括各种GameObject的纹理、天空盒纹理以及场景中所用的Lightmap资源.  
   B. Scene Memory:  
      记录当前场景中各个方面的内存占用情况，包括GameObject、所用资源、各种组件以及GameManager等（天般情况通过AssetBundle加载的不会显示在这里).  
   A. Other:  
      ManagedHeap.UseSize:代码在运行时造成的堆内存分配，表示上次GC到目前为止所分配的堆内存量.  
      SerializedFile(3):  
      WebStream:这个是由WWW来进行加载的内存占用.  
      System.ExecutableAndDlls:不同平台和不同硬件得到的值会不一样。**

5. 优化重点  
   A. CPU-GC Allow:  
      关注原则：1.检测任何一次性内存分配大于2KB的选项 2.检测每帧都具有20B以上内存分配的选项.  
   B. Time ms:  
      记录游戏运行时每帧CPU占用（特别注意占用5ms以上的）.  
  **C. Memory Profiler-Other:  
      1.ManagedHeap.UsedSize: 移动游戏建议不要超过20MB.  
      2.SerializedFile: 通过异步加载(LoadFromCache、WWW等)的时候留下的序列化文件,可监视是否被卸载.  
      3.WebStream: 通过异步WWW下载的资源文件在内存中的解压版本,比SerializedFile大几倍或几十倍,重点监视.\*\*\*\*  
   D. Memory Profiler-Assets:  
      1.Texture2D: 重点检查是否有重复资源和超大Memory是否需要压缩等.  
      2.AnimationClip: 重点检查是否有重复资源.  
      3.Mesh： 重点检查是否有重复资源.**  
  
6. 项目中可能遇到的问题

   A. Device.Present:  
      1.GPU的presentdevice确实非常耗时，一般出现在使用了非常复杂的shader.  
      2.GPU运行的非常快，而由于Vsync的原因，使得它需要等待较长的时间.  
      3.同样是Vsync的原因，但其他线程非常耗时，所以导致该等待时间很长，比如：过量AssetBundle加载时容易出现该问题.  
      4.Shader.CreateGPUProgram:Shader在runtime阶段（非预加载）会出现卡顿(华为K3V2芯片).  
   B. StackTraceUtility.PostprocessStacktrace()和StackTraceUtility.ExtractStackTrace():  
      1.一般是由Debug.Log或类似API造成.  
      2.游戏发布后需将Debug API进行屏蔽.

   C. Overhead:  
      1.一般情况为Vsync所致.  
      2.通常出现在Android设备上.  
   D. GC.Collect:  
      原因: 1.代码分配内存过量(恶性的) 2.一定时间间隔由系统调用(良性的).  
      占用时间：1.与现有Garbage size相关 2.与剩余内存使用颗粒相关（比如场景物件过多，利用率低的情况下，GC释放后需要做内存重排)  
   E. GarbageCollectAssetsProfile:  
      1.引擎在执行UnloadUnusedAssets操作(该操作是比较耗时的,建议在切场景的时候进行).  
      2.尽可能地避免使用Unity内建GUI，避免GUI.Repaint过渡GC Allow.  
      3.if(other.tag == GearParent.MogoPlayerTag)改为other.CompareTag(GearParent.MogoPlayerTag).因为other.tag为产生180B的GC Allow.  
   F. 少用foreach，因为每次foreach为产生一个enumerator(约16B的内存分配)，尽量改为for.  
   G. Lambda表达式，使用不当会产生内存泄漏.  
   H. 尽量少用LINQ：  
      1.部分功能无法在某些平台使用.  
      2.会分配大量GC Allow.  
   I. 控制StartCoroutine的次数：  
      1.开启一个Coroutine(协程)，至少分配37B的内存.  
      2.Coroutine类的实例 — 21B.  
      3.Enumerator — 16B.  
   J. 使用StringBuilder替代字符串直接连接.  
   K. 缓存组件:  
      1.每次GetComponent均会分配一定的GC Allow.  
      2.每次Object.name都会分配39B的堆内存.