**项目优化方案和性能记录**

熊健

1. **CPU占用**

**----------------------------------------------------------------------------**

红米Note2：

>33ms：占总帧数58%

>50ms：18%

华为6Plus：

>33ms：占总帧数56%

>50ms：27%

1. **内存占用**

**----------------------------------------------------------------------------**

红米：

总体：450M（建议低端手机150m）

Mono: 50M

建议：

检查配置文件大小

检查较大Container（Array，List，Dic），缓冲池，控制缓冲池开辟的size

检测函数的堆内存分配情况

1. **加载模块**

**----------------------------------------------------------------------------**

1. Loading.UpdatePreloading

3个部分的CPU开销

1.Resources.UnloadUnusedAsset：占据41%

2.资源加载：45%

3.GC调用：5%

2. 逻辑加载代码

1.GameMgr.Update

GameMgr.GCCollect

AssetManager.Update

3. 加载方式

AssetBundle.LoadAsync, 调用了3603次

4. AssetBundle加载

加载：New WWW

建议：

对加载次数过多的AssetBundle尝试将其常驻于内存，

避免频繁加载和卸载带来性能开销

对场景加载效率的优化，建议：

1.简化资源

2.尝试将部分UI资源等常驻内存

3.避免频繁的Resources.UnloadUnusedAssets调用

4.尝试将异步加载换成同步加载

5.严格检测资源的加载策略

6.对AssetBundle文件加载进行进一步规划

7.避免不必要的Log输出

8.对GameObject上Actor组件的序列化信息进行清理

5. Instantiate实例化

建议：

将实例化频率较高的GameObject放入缓存池

统计：

游戏在19000帧中调用GC 86次，平均每219帧调用触发一次GC，每次GC耗时57ms

建议：通过缓冲池Active和Deactive来代替Instantiate和Destroy

6. Resources.UnloadUnusedAssets

建议：

1。通过Resources.UnloadAsset去除确定不再使用的资源

降低Resource.UnloadUnusedAssets统一处理时的压力

2.对调用频率进行控制，目前在切场景时调用次数过多

3.严格控制场景中粒子系统的使用数量

4.对AssetBundle进行优化打包，尽可能去除冗余资源

7. Shader解析

通过AssetBundle依赖关系打包将Shader进行抽离，编程单独含有Shader的AssetBundle文件

8. 资源比较

9. GameObject.Active 和 Deactive

按照剧情运行的16700帧中，游戏总共Active GameObject 32860ci

Deactive GameObject 69864次

建议：

查看代码中是否存在重复操作的情况

**五．资源使用**

**----------------------------------------------------------------------------**

1.纹理

峰值达到45M，处于合理范围，但仍有提升空间

1.1格式：

建议：

1.对项目使用RGBA32，ARGB32和RGBA16格式的纹理资源进一步检测

极可能转换为2个ETC1纹理（RGB ETC1+Alpha ETC1）

2.对使用ETC2的纹理，慎重考虑

3.对DXT格式，进行修正

1.2.纹理Mipmap

占用了170k

1.3.纹理冗余

原因：

1.重名资源

2AssetBundle打包时将同种资源打到不同AssetBundle中国

3.加载后并没有完全卸载

2.Mesh

内存峰值：10.9M

建议：

1.不少mesh数据中大量含有Color数据，Normal，Tangent

应根据情况去除这些属性

2.将多余数据的网格去除，建议开启“Optimize MeshData”

3.内存中存在少量冗余资源

3. Shader

内存：0.4M

4. AnimationClip

内存：19.2M

建议：

1.精简资源，占用200k的Clip较多，尽量精简

2.存在疑似冗余资源

**六．代码CPU**

**----------------------------------------------------------------------------**

1.Camera.Render

CPU耗时：均值15.3；5.2-31（5%-95%）

1.1.MeshSkinning.Render: 3ms

建议：对SkinnedMesh简化，查看所有相关模型

1.2.粒子系统渲染

建议：根据移动设备对ps进行管理，对中低端降低ps负责程度和使用数量

1.3.场景物体的不透明和半透明渲染

半透明：3ms（3-50ms）

不透明：2ms（2-10ms）

建议：

1.首次渲染将从CPU传入GPU，CPU较好耗时

2.中低端手机，对场景半透明模型简化，对UI优化，降低DC

1.4.多线程渲染

多线程耗时：8.4ms

非多线程：15.3

建议：

1.开启多线程渲染

2.大量测试多线程渲染的设备兼容性

2. GameMgr.Update

平均耗时：12ms（2.5-100ms）

建议：

1.对AssetBundle，资源加载/卸载进行合理控制

2.对Instantiate实例化操作（特别是战斗）进行完善

3.对BundleReference.CheckUnloadBundles进行检测，查看CPU占用是否合理

4.对自身Native代码的CPU开销进一步检测

5.对GCCollect检测

3. Animators.Update

耗时：均值10ms

建议：

1.对角色开启“OptimizeGameObject”，降低Animator.Update

和MeshSkinning.Update

2.场景中GameObject挂在的Animator没有Disable

a。对Animator Disable

b。GameObject处于Active，使用CullingGroup将其身上的Animator或其组件禁用

3.对于动画事件的CPU占用，CPU开销较高因为资源动态加载和实例化导致，

建议场景（特别是战斗）资源加载进一步规划和缓存

4．UIPanel.LateUpdate&UICamera.Update&Canvas.SendWillRenderCavas

UIPanel.LateUpdate: 4ms(3-40ms)

UICamera.Update: (0-20)

Canvas.SendWillRenderCavases:0-20

5． Loading.UpdatePreloading

耗时：0-50

6． EffectManager.Update

耗时：0-20

建议：

优化SetParticleBinding函数

优化ForceUpdateAllAnim递归函数

优化m\_AnimParticle数组的缓存数量，降低调用频率

7.Destroy

在副本中有大量调用操作

建议：

尽可能通过缓冲池减少频繁Instantiate/Destroy操作

**七．代码堆内存**

**----------------------------------------------------------------------------**

堆内存分配过大或过快，导致GC频繁，GC又会导致暂停所有线程，出现卡顿

堆内存Top3函数：

GameMgr.Update 471M

UIPanel.LateUpdate 103

EffectManager.Update 35

1．Camera.Update

建议：

1.在Release版本中控制Log的输出，关闭非关键Log的输出

2．UIPanel.LateUpdate

3．EffectManager.Update

内存分配：5-40k

4．Animators.Update

堆内存分配因素：

1.字符串操作

2.资源加载

5. AudioLoader.Update

堆内存分配主要来来自于ObjectReference.ReleaseObject

6. UICamera.Update