

逻辑代码的性能瓶颈 定位与优化方法

张强

侑虎科技（上海）有限公司

日期：2018.12.15

主要内容

堆内存

累积分析

泄漏分析

占用分析

CPU

瓶颈函数定位

高频函数优化

Lua耗时分析

LWA
MAKE IT SIMPLE

堆内存



堆内存 – 累积分析

堆内存-累积分析

影响：GC触发频率

UWA GOT：Mono 堆内存具体分配



堆内存-累积分析

影响：GC频率

原模式定位方式，逐层展开

高堆内存函数列表			正序调用分析 ▾
函数名 ▾	总堆内存分配 ▾	变量总数 ▾	
DemoFinder.Update	77.08 MB	1788916	
DemoCtrl.LateUpdate	74.89 MB	923344	
Minimap.Update	34.96 MB	954563	
Loader.Update	15.89 MB	318211	
TakeX.Update	12.30 MB	327850	
TakeY.Update	11.26 MB	203617	
Minimap.LateUpdate	9.22 MB	197439	
Player.doAudio	7.72 MB	85588	
EventSystem.Update	6.26 MB	223872	
LuaBehaviour.Start	6.17 MB	102431	

堆内存-累积分析

影响：GC频率

原模式定位方式，逐层展开

	函数名	总堆内存	自身堆内存	总变量数	自身变量数
<input checked="" type="checkbox"/>	▼ DemoFinder.Update	77.08 MB	0 B	1788916	0
<input type="checkbox"/>	▼ Hand.doCell	77.08 MB	339.09 KB	1788916	17352
<input type="checkbox"/>	▼ LuaManager.CallFunction	53.37 MB	0 B	1196139	0
<input type="checkbox"/>	▼ LuaFunction.Call	52.30 MB	0 B	1161019	0
<input type="checkbox"/>	▼ LuaFunction.PCall	14.82 MB	0 B	260338	0
<input type="checkbox"/>	▼ LuaState.PCall	14.82 MB	558 B	260338	5
<input type="checkbox"/>	▶ ToLua.Print	9.39 MB	0 B	90316	0
<input type="checkbox"/>	▶ EventDispatcherWrap.DispatchEvent	3.02 MB	0 B	98096	0
<input type="checkbox"/>	▶ EventDispatcher.DispatchEvent	10.08 MB	0 B	65665	0
<input type="checkbox"/>	▶ EventDispatcherWrap.DispatchEvent	4.92 MB	0 B	78049	0
<input type="checkbox"/>	▶ DelegateFactory-EventDispatcher_EventBack_Event.Call	3.10 MB	0 B	82449	0
<input type="checkbox"/>	▶ DelegateFactory-GridAndLoop_OnInitializeItem_Event.Call	2.97 MB	0 B	103878	0
<input type="checkbox"/>	▶ WalkHelper.moveTo	2.12 MB	0 B	104128	0
<input type="checkbox"/>	▶ WalkHelper.gotoPt	1.96 MB	0 B	95649	0
<input type="checkbox"/>	▶ GridAndLoop.UpdateItem	1.90 MB	0 B	16250	0
<input type="checkbox"/>	▶ PlayerWalkAI.start	1.70 MB	0 B	95906	0

堆内存-累积分析

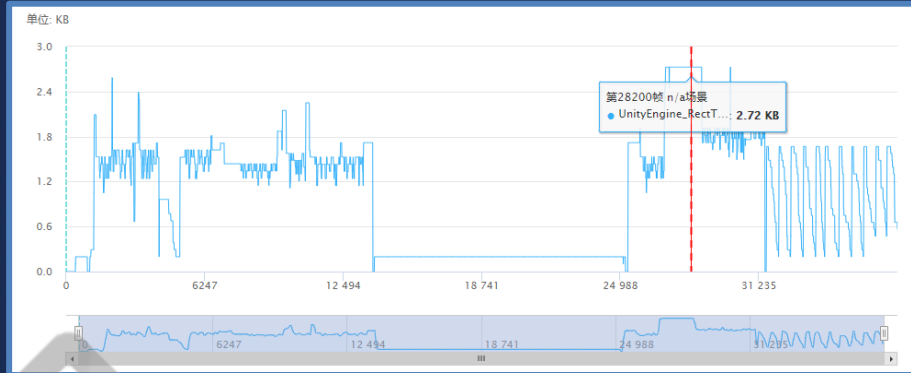
影响：GC频率

倒序模式：直接函数、调用路径、开销分布

高堆内存函数列表			倒序调用分析 ▾
函数名 ⇅	总堆内存分配 ⇅	变量总数 ⇅	
UnityEngine.RectTransformWrap.Lua_ToString	37.49 MB	405110	
UnityEngine.TransformWrap.Lua_ToString	35.32 MB	414633	
StringBuilderCache.Acquire	21.10 MB	80459	
Loader.GetDepends	13.45 MB	88876	
UnityEngine.GameObjectWrap.Lua_ToString	11.95 MB	142075	
string.Concat	10.66 MB	168884	
EventDispatcher.DispatchEvent	8.94 MB	371996	
LuaDLL.lua_ptrtostring	8.58 MB	217018	
TextGenerator.ctor	6.66 MB	7824	
SQLiteBase.PointerToString	6.08 MB	214251	

堆内存-累积分析

影响：GC频率

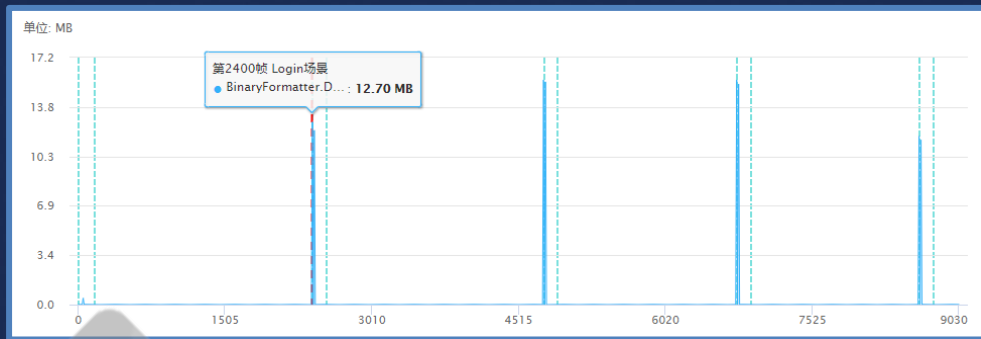


倒序模式：直接函数、调用路径、开销分布

	函数名	自身堆内存	自身变量数
<input checked="" type="checkbox"/>	▼ UnityEngine_RectTransformWrap.Lua_ToString	37.48 MB	405110
<input type="checkbox"/>	▼ LuaState.PCall	37.48 MB	405110
<input type="checkbox"/>	▼ LuaFunction.PCall	37.48 MB	405110
<input type="checkbox"/>	▼ DelegateFactory-EventDispatcher_EventBack_Event.Call	37.48 MB	405110
<input type="checkbox"/>	▼ EventDispatcher.DispatchEvent	37.48 MB	405110
<input type="checkbox"/>	▶ DemoCtrl.Comp	37.48 MB	405083
<input type="checkbox"/>	▶ layer.doReady	1.39 KB	17
<input type="checkbox"/>	▶ EventDispatcherWrap.DispatchEvent	756 B	9
<input type="checkbox"/>	▶ Controller.Reset	84 B	1

堆内存-累积分析

影响：GC频率



倒序模式：直接函数、调用路径、开销分布

	函数名	自身堆内存	自身变量数
<input checked="" type="checkbox"/>	▼ BinaryFormatter.Deserialize	114.64 MB	2448785
<input type="checkbox"/>	▼ Loader.ReadBytes	109.85 MB	2395470
<input type="checkbox"/>	▸ Loader.ReadFromStream	109.85 MB	2395470
<input type="checkbox"/>	▸ Map.GetInfo	2.87 MB	328
<input type="checkbox"/>	▸ Util.Read<object>	1.91 MB	52987

堆内存-累积分析

影响：GC频率

1. 切换到倒序调用模式显示
2. 通过曲线（配合截图）判断优先级
3. 根据主要调用路径分析其可优化性
4. 找到直接函数（或其父函数）的实现进行优化

堆内存 — 泄漏分析

堆内存-泄漏分析

影响：堆内存峰值持续增大

UWA GOT：Mono 堆内存泄漏分析

UWA



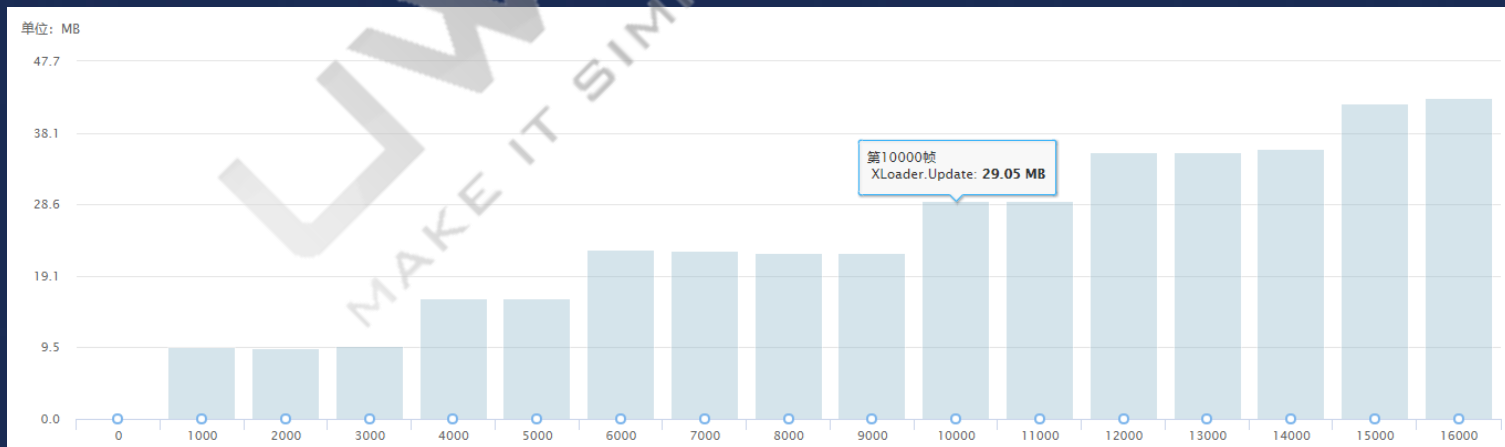
性能简报



堆内存具体分配



堆内存泄漏分析



堆内存-泄漏分析

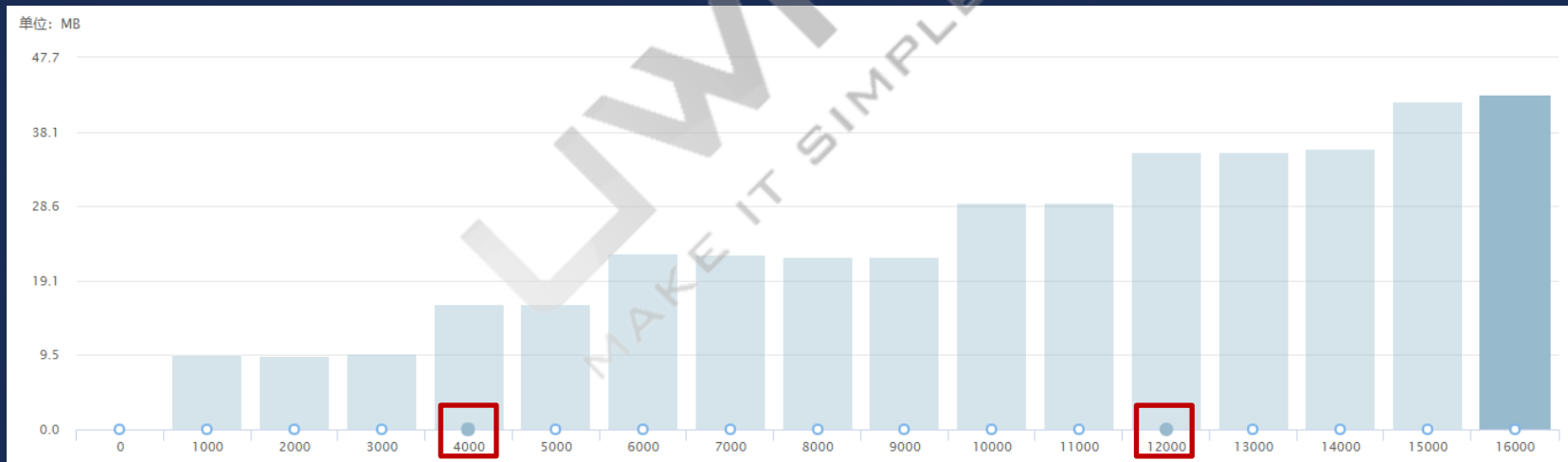
原因：

1. C#直接引用
2. Lua 间接引用



堆内存-泄漏分析

基本方法，快照对比
相同场景、游戏阶段



堆内存-泄漏分析

原因：

C#直接引用

容器、static变量

LWA
MAKE IT SIMPLE

堆内存-泄漏分析

Lua 间接引用 (Slua为例)

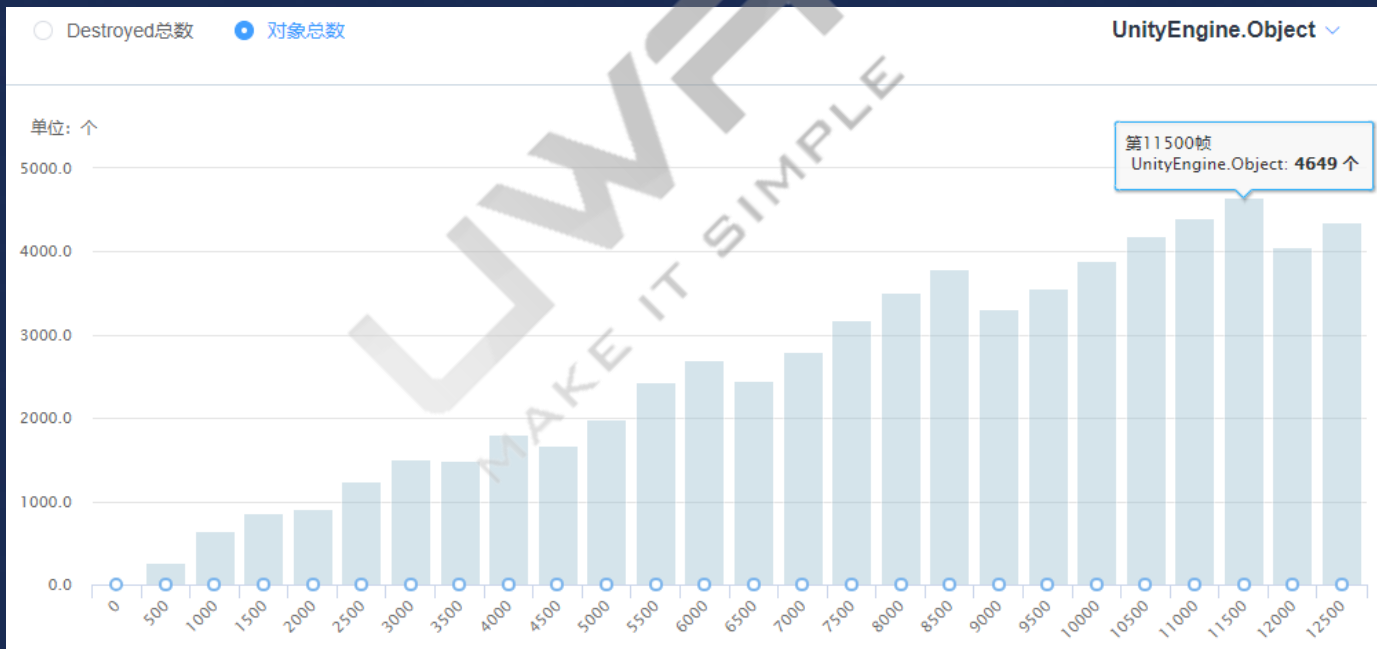
Dump ObjectCache 快照进行对比



堆内存-泄漏分析

Lua 间接引用 (Slua为例)

Dump ObjectCache 快照进行对比



堆内存-泄漏分析

影响：堆内存峰值持续增大

C#直接引用

1. 选择适当的两帧进行快照对比
2. 查找关键**泄漏路径** (Instantiate/Load/CreateMonster)
3. 查找标志性**泄漏类型** (Image/Monster)
4. 排查相关容器

堆内存-泄漏分析

影响：堆内存峰值持续增大

Lua间接引用

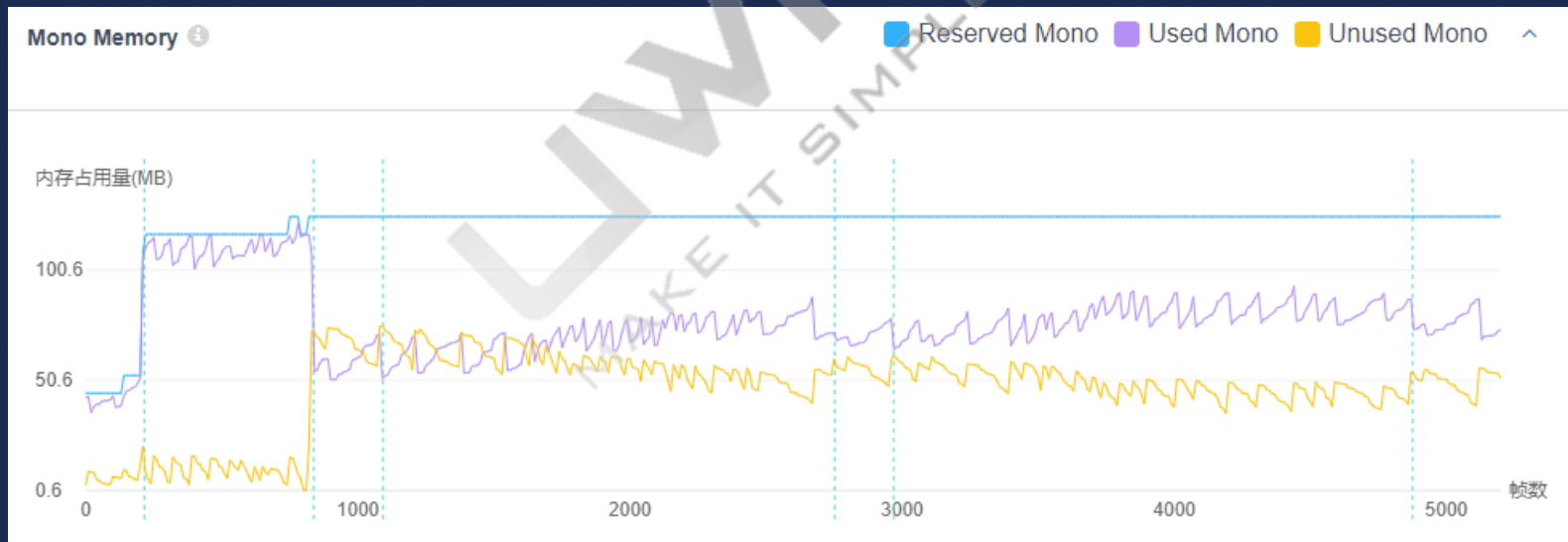
1. 对比ObjectCache(ObjectTranslator)对象快照
2. 查找标志性**泄漏类型** (Image/Monster)
3. 排查LuaGC的触发频率
4. 排查Lua代码中的相关容器

堆内存 - 占用分析

堆内存-占用分析

影响：堆内存峰值过大

UWA GOT：Mono 堆内存泄漏分析 + Direct 模式



堆内存-占用分析

影响：堆内存峰值过大

1. Direct 模式测试，查看堆内存泄漏分析
2. 排查根节点、多线程部分的堆内存占用
3. 针对短时间出现的峰值，进行手动采样

CPU



CPU – 瓶颈函数定位

CPU - 瓶颈函数定位

影响：帧率 & 流畅度

UWA GOT : Overview / CPU

UWA
MAKE IT SIMPLE

CPU - 瓶颈函数定位

用 UWA API 统计 Unity API 耗时

```
gameObject.AddComponent<Rigidbody>();
```

```
gameObject.AddComponentX<Rigidbody>();
```

```
namespace UwaApiExtensions
{
    public static class UwaUnityApiWrap
    {
        public static T AddComponentX<T>(this GameObject go) where T : Component
        {
            UWAEngine.PushSample("GameObject.AddComponent");
            T obj = go.AddComponent<T>();
            UWAEngine.PopSample();
            return obj;
        }
    }
}
```

```
[Conditional("ENABLE_PROFILER")]
public static void PushSample(string sampleName)
{
    if (UWAEngine._instance == null)
        return;
    UWAEngine._instance.PushSample(sampleName);
}
```

CPU - 瓶颈函数定位

定位 Hot Path!

iOS: Instrument

Android: Profiler / UWA GOT
人为采样

Deep Profile

<https://answer.uwa4d.com/question/5bdaaa3286a0b505003c12ab>

CPU - 瓶颈函数定位

Profiler-DeepProfile

堆栈精确
统计误差大

Profiler-Simple
UWA GOT

统计精确
堆栈简略



CPU - 瓶颈函数定位

Profiler-DeepProfile

堆栈精确
统计误差大

Profiler-Simple
UWA GOT

统计相对精确
堆栈简略

Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms	△
▼ Update.ScriptRunBehaviourUpdate	63.5%	0.0%	1	344.7 KB	144.86	0.00	
▼ BehaviourUpdate	63.5%	0.0%	1	344.7 KB	144.86	0.03	
▼ TypewriterEffect.Update()	63.3%	0.0%	1	339.1 KB	144.24	0.00	
▼ TypewriterEffect.Update()	63.3%	0.0%	1	339.1 KB	144.23	0.18	
▼ UILabel.set_text()	62.9%	0.0%	214	289.7 KB	143.39	0.15	
▼ UILabel.ProcessAndRequest()	62.3%	0.0%	213	289.7 KB	141.94	0.09	
▼ UILabel.ProcessText()	62.1%	0.0%	213	289.7 KB	141.69	0.04	
▼ UILabel.ProcessText()	62.1%	0.1%	213	289.7 KB	141.65	0.32	
▼ NGUIText.WrapText()	33.4%	3.2%	213	289.7 KB	76.14	7.51	
▼ NGUIText.GetGlyphWidth()	22.2%	3.0%	21944	0 B	50.58	6.93	
▼ BMFont.GetGlyph()	9.0%	1.1%	21944	0 B	20.55	2.66	
▼ BMFont.GetGlyph()	7.8%	1.8%	21944	0 B	17.89	4.13	
Dictionary`2.TryGetValue()	5.4%	2.0%	21944	0 B	12.40	4.63	
Dictionary`2.get_Count()	0.5%	0.5%	21944	0 B	1.35	1.35	
Object.op_Inequality()	5.3%	1.1%	21944	0 B	12.08	2.56	
UIFont.get_bmFont()	2.9%	1.1%	21944	0 B	6.81	2.71	
BMGlyph.GetKerning()	1.8%	1.2%	21132	0 B	4.16	2.84	
▼ NGUIText.ParseSymbol()	2.0%	1.2%	21944	0 B	4.61	2.95	
NGUIText.ParseSymbol()	0.7%	0.7%	21944	0 B	1.65	1.65	
▼ Mathf.RoundToInt()	2.0%	1.2%	21944	0 B	4.60	2.86	
Mathf.Round()	0.7%	0.7%	21944	0 B	1.73	1.73	
▼ String.Substring()	1.8%	0.2%	4089	147.8 KB	4.10	0.56	
String.SubstringUnchecked()	1.5%	0.3%	4086	147.8 KB	3.54	0.87	
▼ StringBuilder.Append()	1.2%	0.3%	4330	131.4 KB	2.80	0.69	
String.CharCopy()	0.6%	0.2%	3876	0 B	1.40	0.51	
StringBuilder.InternalEnsureCapacity()	0.2%	0.0%	625	131.4 KB	0.68	0.17	
String.InternalSetChar()	0.0%	0.0%	241	0 B	0.02	0.02	
NGUIText.IsSpace()	0.6%	0.6%	21944	0 B	1.52	1.52	

Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms	△
▼ Update.ScriptRunBehaviourUpdate	64.3%	0.0%	1	339.4 KB	18.86	0.00	
▼ BehaviourUpdate	64.3%	0.2%	1	339.4 KB	18.86	0.06	
TypewriterEffect.Update()	63.6%	63.6%	1	339.1 KB	18.63	18.63	
UIRect.Update()	0.2%	0.2%	44	0 B	0.08	0.08	
ActiveAnimation.Update()	0.1%	0.0%	2	284 B	0.04	0.02	

CPU - 瓶颈函数定位

UWA GOT [PC版] – Auto profile

进一步降低Profile本身耗时

排除调用次数对统计造成的误差

减小数据量

UWA
MAKE IT SIMPLE

CPU - 瓶颈函数定位

UWA GOT [PC版] – Auto profile

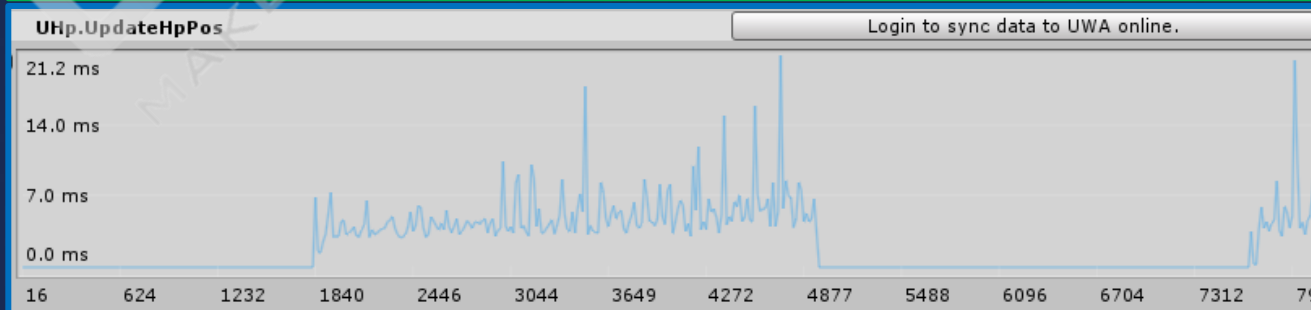
Name	percent	selfPercent	totalTime	calls	selfTime	selfCalls
▼ TypewriterEffect:Update	100.00 %	4.13 %	0.4 ms	643	0.0 ms	1
▼ UILabel:set_text	95.86 %	0.00 %	0.4 ms	640	0.0 ms	214
▼ UILabel:ProcessAndRequest	95.86 %	0.00 %	0.4 ms	639	0.0 ms	213
▼ UILabel:ProcessText	95.86 %	0.00 %	0.4 ms	426	0.0 ms	213
▼ UILabel:ProcessText	95.86 %	0.00 %	0.4 ms	2982	0.0 ms	213
▼ NGUIText:WrapText	62.04 %	62.04 %	0.3 ms	1093	0.3 ms	213
▶ System.Text.StringBuilder:ctor	0.00 %	0.00 %	0.0 ms	426	0.0 ms	213
▶ System.Text.StringBuilder:Append	0.00 %	0.00 %	0.0 ms	482	0.0 ms	241
NGUIText:Prepare	0.00 %	0.00 %	0.0 ms	213	0.0 ms	213
UnityEngine.Mathf:FloorToInt	0.00 %	0.00 %	0.0 ms	213	0.0 ms	213
▼ NGUIText:CalculatePrintedSize	33.81 %	33.81 %	0.1 ms	639	0.1 ms	213
NGUIText:Prepare	0.00 %	0.00 %	0.0 ms	213	0.0 ms	213
NGUIText:StripSymbols	0.00 %	0.00 %	0.0 ms	213	0.0 ms	213
▶ UIFont:get_defaultSize	0.00 %	0.00 %	0.0 ms	639	0.0 ms	213
UIWidget:get_width	0.00 %	0.00 %	0.0 ms	213	0.0 ms	213
▶ UILabel:get_defaultFontSize	0.00 %	0.00 %	0.0 ms	639	0.0 ms	213
UILabel:set_shouldBeProcessed	0.00 %	0.00 %	0.0 ms	213	0.0 ms	213
▶ UILabel:get_trueTypeFont	0.00 %	0.00 %	0.0 ms	426	0.0 ms	213
UIWidget:get_height	0.00 %	0.00 %	0.0 ms	213	0.0 ms	213
UILabel:get_isValid	0.00 %	0.00 %	0.0 ms	213	0.0 ms	213
▶ UILabel:UpdateNGUIText	0.00 %	0.00 %	0.0 ms	2130	0.0 ms	213
▶ UnityEngine.Mathf:Abs	0.00 %	0.00 %	0.0 ms	426	0.0 ms	213

CPU - 瓶颈函数定位

UWA GOT - 瓶颈函数

Name	percent	selfPerce	totalTime	calls	selfTime	selfCalls
▼ UMain:LateUpdate	100.00 %	0.06 %	109292.1 ms	7772	67.0 ms	3886
▶ UHp.UpdateHpPos	10.19 %	10.19 %	11147.6 ms	36252	11147.6 ms	36252
▶ UPlayerState.ChangeTo	4.94 %	4.94 %	5401.4 ms	119343	5401.4 ms	119343
▶ UPlayer.Update	4.07 %	4.07 %	4459.8 ms	122155	4459.8 ms	122155
▶ UWapon.Do	2.93 %	2.93 %	3207.7 ms	10459	3207.7 ms	10459
▶ UTopUi.ChangeTo	2.83 %	2.83 %	3095.7 ms	34800	3095.7 ms	34800
▶ UNpcCtrl.UpdateRotation	2.46 %	2.46 %	2693.8 ms	36203	2693.8 ms	36203
▶ USkillCtrl.ChangeTo	2.34 %	2.34 %	2567.6 ms	174037	2567.6 ms	174037
▶ UHp.UpdateHp	2.34 %	2.34 %	2565.4 ms	36252	2565.4 ms	36252
▶ UXptCtrl.Lerp	2.06 %	2.06 %	2256.8 ms	122155	2256.8 ms	122155
▶ UPlayer.GoTo	2.03 %	2.03 %	2226.2 ms	10394	2226.2 ms	10394

▼ UHp.UpdateHpPos	10.19 %	10.19 %	11147.6 ms	36252	11147.6 ms	36252
▼ UHp.SetHp	10.19 %	10.19 %	11147.6 ms	36252	11147.6 ms	36252
▼ UHp.UpdateHp	10.19 %	10.19 %	11147.6 ms	36252	11147.6 ms	36252
▼ UDemoCtrl.Update	10.19 %	10.19 %	11147.6 ms	36252	11147.6 ms	36252
▶ UPlayer.Update	10.19 %	10.19 %	11147.6 ms	36252	11147.6 ms	36252



CPU - 瓶颈函数定位

UWA GOT - 瓶颈函数

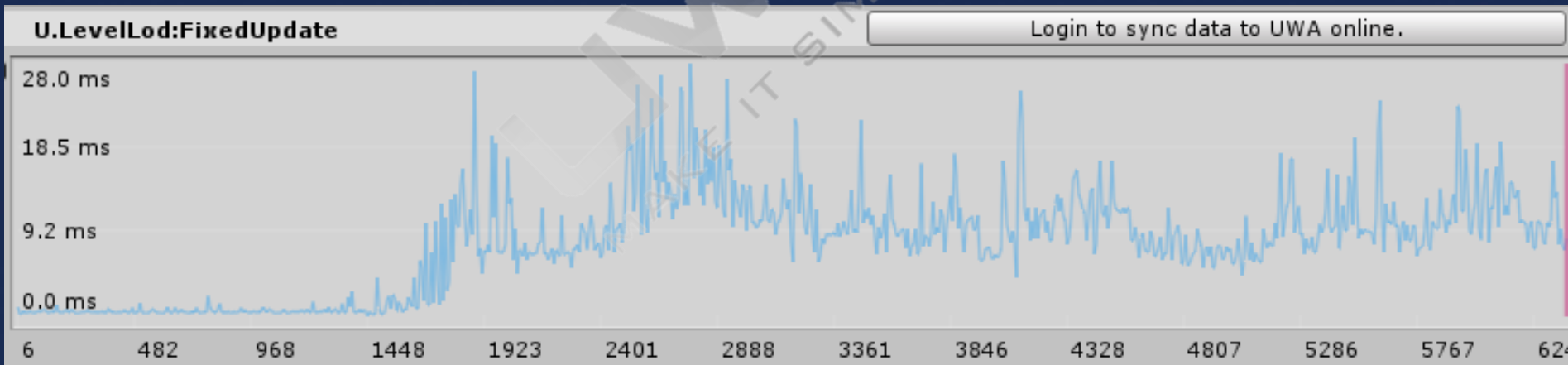
1. 通过UWA API统计耗时的Unity或自身API；
2. 通过Auto Profile排查**非高频**的瓶颈函数。步骤与累积堆内存部分相似（曲线、堆栈、查函数）。
3. 通过Deep Profile排查瓶颈函数中哪些是由于**高频子函数**调用导致。

CPU – 高频函数优化

CPU - 高频函数优化

UWA GOT - 高频函数

Name	percent	selfPercent	totalTime	calls	selfTime	selfCalls
▼ U.LevelLod:FixedUpdate	9.01 %	8.94 %	38059.1 ms	13312198	37735.9 ms	13225586
U.LevelLod.UpdateLod	0.07 %	0.07 %	323.2 ms	86612	323.2 ms	86612
► MainPlayer:FixedUpdate	5.95 %	0.26 %	25115.1 ms	63084	1116.2 ms	31542
► BillBoardItem:Update	5.73 %	2.76 %	24182.3 ms	3824358	11673.8 ms	1912179



CPU - 高频函数优化

高频 Update

One might ask why should anyone care about an empty method? The thing is that these are the calls from native C++ land to managed C# land, they have a cost. Let's see what this cost is.

	iPhone 6	iPhone 4s
Update	5.4ms	10.91ms
Manager (List)	1ms	2.52ms
Manager (array)	0.22ms	1.15ms

<https://blogs.unity3d.com/2015/12/23/1k-update-calls/>

CPU - 高频函数优化

针对高频 Update

1. 通过Deep Profile统计调用次数是否合理
2. 不使用Coroutine
3. 不使用MonoBehaviour.Update/
LateUpdate/**FixedUpdate**
4. 尽可能用Array遍历DoUpdate
5. 针对**低频逻辑**，轮循式->事件队列

CPU - 高频函数优化

高频遍历Unity组件？

Entity Component System(ECS)
C# Job System

Unite Austin 2017 - Writing High Performance C#
Scripts

CPU - Lua 开销分析

CPU - Lua 开销分析

Lua插件开销分类

Lua到C#的调用本身的开销

Lua端本身的函数造成的开销

C#端Wrap函数造成的开销

通过反射调用C#函数的开销

UWA
MAKE IT SIMPLE

CPU - Lua 开销分析

Lua到C#的调用本身的开销

```
gameobj.transform.position = pos
```

https://blog.uwa4d.com/archives/USparkle_Lua.html

CPU - Lua 开销分析

通过 UWA API 记录 Lua 到 C# 的调用次数

```
void WriteUwaMarker(StreamWriter file, Type t, MethodInfo m)
{
    Write(file, "UWAEngine.AddMarker(\"\" + t.Name + "." + m.Name + "\");");
}
```

```
void WriteFunctionImpl(StreamWriter file, MethodInfo m, Type t, BindingFlags bf)
{
    WriteUwaMarker(file, t.Name, m.Name);
    WriteTry(file);
}
```

WriteFunctionImpl (1处)

WriteField (4处)

```
[SLua.MonoPInvokeCallbackAttribute(typeof(LuaCSFunction))]
[UnityEngine.Scripting.Preserve]
static public int Find s(IntPtr l) {
    UWAEngine.AddMarker("Shader.Find");
    try {
        System.String a1;
        checkType(l,1,out a1);
        var ret=UnityEngine.Shader.Find(a1);
        pushValue(l,true);
        pushValue(l,ret);
        return 2;
    }
    catch(Exception e) {
        return error(l,e);
    }
}
```

CPU - Lua 开销分析

通过 UWA API 记录 Lua 到 C# 的调用次数




CPU - Lua 开销分析


Lua端本身的函数造成的开销


UWA GOT Lua 模式


UWA

 性能简报

 总体堆内存

 代码效率

 CPU时间占用

 堆内存具体分配

 Mono对象引用

函数总体堆栈信息						总体堆栈信息 ▾
	函数名	总耗时(ms)	自身耗时(ms)	调用次数	显著调用帧数	
<input checked="" type="checkbox"/>	▼ unknown@Main:75	3524.95	155.32	12544	0	
<input type="checkbox"/>	▼ Update@Base/YwMonoBehaviour:113	3369.63	241.61	12544	0	
<input checked="" type="checkbox"/>	▶ Update@Logic/LgBackgroundParallax:103	2035.89	583.01	12422	2	
<input type="checkbox"/>	▶ Update@Logic/LgPlayer:117	338.73	53.92	11937	0	
<input type="checkbox"/>	▶ Update@Logic/LgGun:82	322.25	60.67	12059	0	
<input type="checkbox"/>	▶ Update@Logic/LgFollowPlayer:57	156.94	98.32	11937	1	
<input type="checkbox"/>	Update@Logic/LgScoreShadow:52	123.96	123.96	12422	0	
<input type="checkbox"/>	▶ Update@Logic/LgScore:72	107.85	59.39	12544	0	
<input type="checkbox"/>	Update@Logic/LgPauser:47	42.39	42.39	12422	0	

CPU - Lua 开销分析

Lua端本身的函数造成的开销

UWA GOT Lua

Assets > 2DPlatformer-SLua-master-2017 > Assets > StreamingAssets > Lua > Logic

Name	Date modified	Type	Size
<input type="checkbox"/> LgPickupSpawner.lua.meta	4/10/2016 4:09 PM	METADATA FILE	1 KB
<input checked="" type="checkbox"/> LgPlayer.lua	4/10/2016 4:09 PM	LUA File	4 KB
<input type="checkbox"/> LgPlayer.lua.meta	4/10/2016 4:09 PM	METADATA FILE	1 KB

▶ Update@Logic/LgBackgroundParallax

▶ Update@Logic/LgPlayer:117

▶ Update@Logic/LgGun:82

```
111 -- Update method.  
112 function LgPlayer:Update()  
113     --print("LgPlayer:Update")  
114  
115     self.m_cPlayerCtrl:Update()  
116     self.m_cLayBombs:Update()  
117 end
```

<https://www.jianshu.com/p/111111111111>

CPU - Lua 开销分析

Lua端本身的函数造成的开销

LuaJIT ->

https://blog.uwa4d.com/archives/usparkle_luajit.html

Lua -> <https://blog.uwa4d.com/archives/2037.html>

UWA
MAKE IT SIMPLE

CPU - Lua 开销分析

针对Lua开销

1. 通过UWA API记录Lua调用C#的次数
2. 通过合并函数等方式降低高频函数
3. 定位耗时的Lua函数，针对瓶颈优化

主要内容

堆内存

累积分析

泄漏分析

占用分析

Mono 模式

具体分配/倒序调用

泄漏分析

Direct+泄漏分析

CPU

瓶颈函数定位

高频函数优化

Lua耗时分析

Overview 模式

Auto/Deep Profile模式

Lua 模式

主要内容

Todo:

Mono

多线程统计

Overview

子线程GC

Deep/AutoProfile

Thank you