HTML5 游戏性能分析和优化

by panda

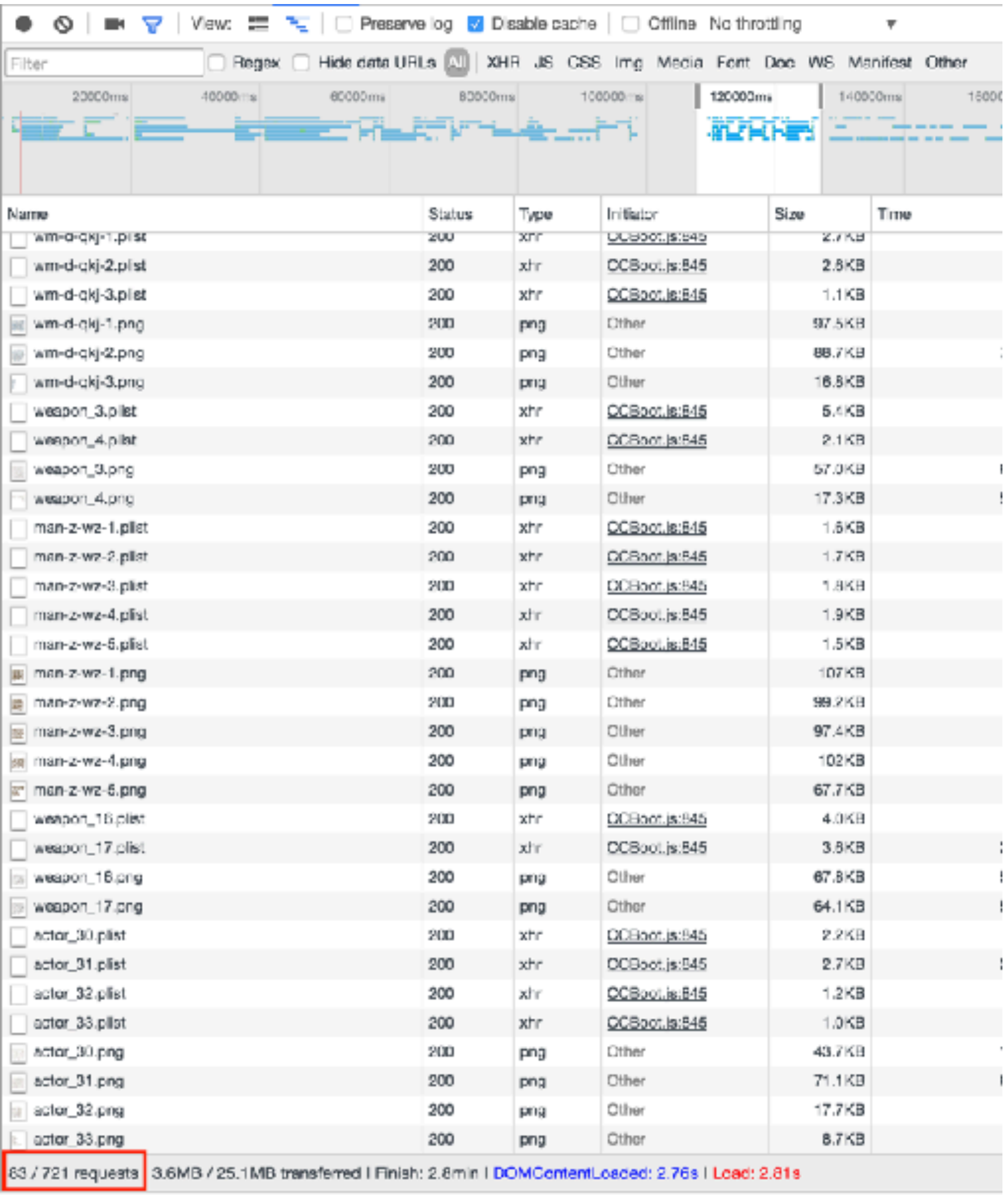
简介

- 使用 Chrome DevTools 分析调试
- 加载优化
- 渲染优化
- 内存优化
- CPU 占用优化

“

Less is more POWERFUL

性能热点分析调试



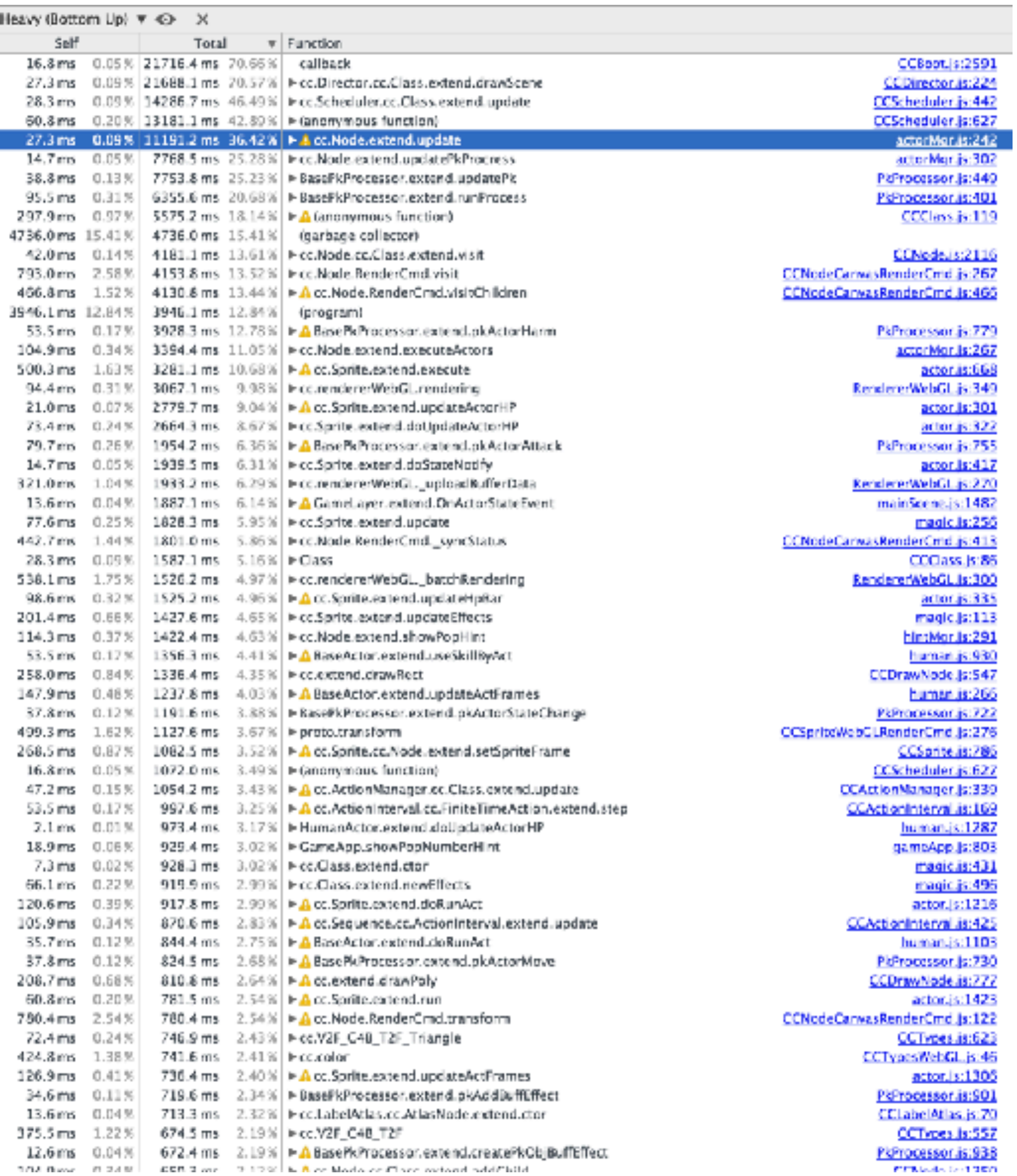
Name	Status	Type	Initiator	Size	Time
wm-d-qkj-1.plist	200	xhr	CCBoot.js:845	2.7KB	
wm-d-qkj-2.plist	200	xhr	CCBoot.js:845	2.8KB	
wm-d-qkj-3.plist	200	xhr	CCBoot.js:845	1.1KB	
wm-d-qkj-1.png	200	png	Other	97.5KB	
wm-d-qkj-2.png	200	png	Other	88.7KB	
wm-d-qkj-3.png	200	png	Other	16.8KB	
weapon_3.plist	200	xhr	CCBoot.js:845	5.4KB	
weapon_4.plist	200	xhr	CCBoot.js:845	2.1KB	
weapon_3.png	200	png	Other	57.0KB	
weapon_4.png	200	png	Other	17.3KB	
man-z-wz-1.plist	200	xhr	CCBoot.js:845	1.6KB	
man-z-wz-2.plist	200	xhr	CCBoot.js:845	1.7KB	
man-z-wz-3.plist	200	xhr	CCBoot.js:845	1.8KB	
man-z-wz-4.plist	200	xhr	CCBoot.js:845	1.9KB	
man-z-wz-5.plist	200	xhr	CCBoot.js:845	1.5KB	
man-z-wz-1.png	200	png	Other	107KB	
man-z-wz-2.png	200	png	Other	99.2KB	
man-z-wz-3.png	200	png	Other	97.4KB	
man-z-wz-4.png	200	png	Other	102KB	
man-z-wz-5.png	200	png	Other	67.7KB	
weapon_16.plist	200	xhr	CCBoot.js:845	4.0KB	
weapon_17.plist	200	xhr	CCBoot.js:845	3.8KB	
weapon_16.png	200	png	Other	67.8KB	
weapon_17.png	200	png	Other	64.1KB	
actor_30.plist	200	xhr	CCBoot.js:845	2.2KB	
actor_31.plist	200	xhr	CCBoot.js:845	2.7KB	
actor_32.plist	200	xhr	CCBoot.js:845	1.2KB	
actor_33.plist	200	xhr	CCBoot.js:845	1.0KB	
actor_30.png	200	png	Other	43.7KB	
actor_31.png	200	png	Other	71.1KB	
actor_32.png	200	png	Other	17.7KB	
actor_33.png	200	png	Other	8.7KB	

63 / 721 requests | 3.6MB / 25.1MB transferred | Finish: 2.8min | DOMContentLoaded: 2.76s | Load: 2.81s

NETWORK 工具

- 关闭缓存
- Throttling 模拟网络环境
- 选取关注的部分
- TTFB 协助优化服务端

https://developers.google.com/web/tools/chrome-devtools/network-performance/resource-loading



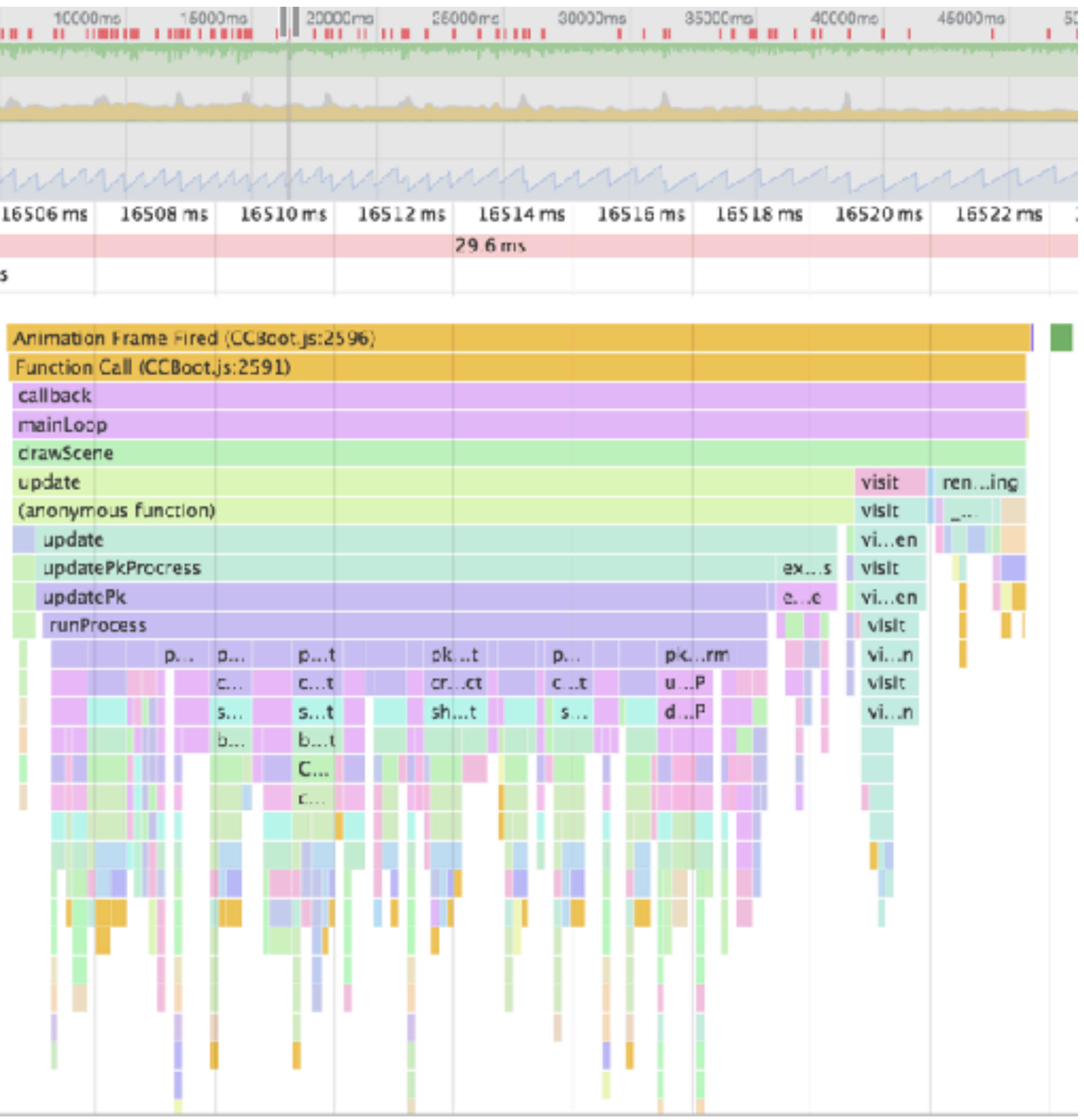
```
Heavy (Bottom Up) ▼ 👁 ✕
      Self              Total        ▼ Function
  16.8ms   0.05%  21716.4 ms  70.66%   callback                                            CCBoot.js:2591
  27.3ms   0.09%  21688.1 ms  70.57%  ▶cc.Director.cc.Class.extend.drawScene               CCDirector.js:224
  28.3ms   0.09%  14286.7 ms  46.49%  ▶cc.Scheduler.cc.Class.extend.update                 CCScheduler.js:442
  60.8ms   0.20%  13181.1 ms  42.89%  ▶(anonymous function)                                CCScheduler.js:627
  27.3ms   0.09%  11191.2 ms  36.42%  ▶⚠cc.Node.extend.update                              actorMgr.js:242
  14.7ms   0.05%   7768.5 ms  25.28%  ▶cc.Node.extend.updatePkProcess                      actorMgr.js:302
  38.8ms   0.13%   7753.8 ms  25.23%  ▶BasePkProcesser.extend.updatePk                     PkProcessor.js:440
  95.5ms   0.31%   6355.6 ms  20.68%  ▶BasePkProcesser.extend.runProcess                   PkProcessor.js:401
 297.9ms   0.97%   5575.2 ms  18.14%  ▶⚠(anonymous function)                               CCClass.js:119
4736.0ms  15.41%   4736.0 ms  15.41%   (garbage collector)
  42.0ms   0.14%   4181.1 ms  13.61%  ▶cc.Node.cc.Class.extend.visit                       CCNode.js:2116
 793.0ms   2.58%   4153.8 ms  13.52%  ▶cc.Node.RenderCmd.visit                             CCNodeCanvasRenderCmd.js:267
 466.8ms   1.52%   4130.8 ms  13.44%  ▶⚠cc.Node.RenderCmd.visitChildren                    CCNodeCanvasRenderCmd.js:465
3946.1ms  12.84%   3946.1 ms  12.84%   (program)
  53.5ms   0.17%   3928.3 ms  12.78%  ▶⚠BasePkProcessor.extend.pkActorHarm                 PkProcessor.js:779
 104.9ms   0.34%   3394.4 ms  11.05%  ▶cc.Node.extend.executeActors                        actorMgr.js:267
 500.3ms   1.63%   3281.1 ms  10.68%  ▶⚠cc.Sprite.extend.execute                           actor.js:668
  94.4ms   0.31%   3067.1 ms   9.98%  ▶cc.rendererWebGL.rendering                          RendererWebGL.js:349
  21.0ms   0.07%   2779.7 ms   9.04%  ▶⚠cc.Sprite.extend.updateActorHP                    actor.js:301
  73.4ms   0.24%   2664.3 ms   8.67%  ▶cc.Sprite.extend.doUpdateActorHP                    actor.js:322
  79.7ms   0.26%   1954.2 ms   6.36%  ▶⚠BasePkProcessor.extend.pkActorAttack               PkProcessor.js:755
  14.7ms   0.05%   1939.5 ms   6.31%  ▶cc.Sprite.extend.doStateNotify                      actor.js:417
 321.0ms   1.04%   1933.2 ms   6.29%  ▶cc.rendererWebGL._uploadBufferData                  RendererWebGL.js:220
  13.6ms   0.04%   1887.1 ms   6.14%  ▶⚠GameLayer.extend.OnActorStateEvent                 mainScene.js:1482
  77.6ms   0.25%   1828.3 ms   5.95%  ▶cc.Sprite.extend.update                             magic.js:256
 442.7ms   1.44%   1801.0 ms   5.86%  ▶cc.Node.RenderCmd._syncStatus                       CCNodeCanvasRenderCmd.js:413
  28.3ms   0.09%   1587.1 ms   5.16%  ▶Class                                               CCClass.js:86
 538.1ms   1.75%   1526.2 ms   4.97%  ▶cc.rendererWebGL._batchRendering                    RendererWebGL.js:300
  98.6ms   0.32%   1525.2 ms   4.96%  ▶⚠cc.Sprite.extend.updateHpBar                       actor.js:335
 201.4ms   0.66%   1427.6 ms   4.65%  ▶cc.Sprite.extend.updateEffects                      magic.js:113
 114.3ms   0.37%   1422.4 ms   4.63%  ▶cc.Node.extend.showPopHint                          HintMgr.js:291
  53.5ms   0.17%   1356.3 ms   4.41%  ▶⚠BaseActor.extend.useSkillByAct                    human.js:930
 258.0ms   0.84%   1336.4 ms   4.35%  ▶cc.extend.drawRect                                  CCDrawNode.js:547
 147.9ms   0.48%   1237.8 ms   4.03%  ▶⚠BaseActor.extend.updateActFrames                  human.js:266
  37.8ms   0.12%   1191.6 ms   3.88%  ▶BasePkProcessor.extend.pkActorStateChange           PkProcessor.js:722
 499.3ms   1.62%   1127.6 ms   3.67%  ▶proto.transform                                     CCSpriteWebGLRenderCmd.js:276
 268.5ms   0.87%   1082.5 ms   3.52%  ▶⚠cc.Sprite.cc.Node.extend.setSpriteFrame            CCSprite.js:786
  16.8ms   0.05%   1072.0 ms   3.49%  ▶(anonymous function)                                CCScheduler.js:627
  47.2ms   0.15%   1054.2 ms   3.43%  ▶⚠cc.ActionManager.cc.Class.extend.update            CCActionManager.js:339
  53.5ms   0.17%    997.6 ms   3.25%  ▶⚠cc.ActionInterval.cc.FiniteTimeAction.extend.step  CCActionInterval.js:169
   2.1ms   0.01%    973.4 ms   3.17%  ▶HumanActor.extend.doUpdateActorHP                  human.js:1787
  18.9ms   0.06%    929.4 ms   3.02%  ▶GameApp.showPopNumberHint                           gameApp.js:803
   7.3ms   0.02%    928.3 ms   3.02%  ▶cc.Class.extend.ctor                                magic.js:431
  66.1ms   0.22%    919.9 ms   2.99%  ▶cc.Class.extend.newEffects                          magic.js:495
 120.6ms   0.39%    917.8 ms   2.99%  ▶⚠cc.Sprite.extend.doRunAct                          actor.js:1216
 105.9ms   0.34%    870.6 ms   2.83%  ▶⚠cc.Sequence.cc.ActionInterval.extend.update        CCActionInterval.js:425
  35.7ms   0.12%    844.4 ms   2.75%  ▶⚠BaseActor.extend.doRunAct                          human.js:1103
  37.8ms   0.12%    824.5 ms   2.68%  ▶⚠BasePkProcessor.extend.pkActorMove                 PkProcessor.js:730
 208.7ms   0.68%    810.6 ms   2.64%  ▶⚠cc.extend.drawPoly                                 CCDrawNode.js:777
  60.8ms   0.20%    781.5 ms   2.54%  ▶⚠cc.Sprite.extend.run                               actor.js:1423
 780.4ms   2.54%    780.4 ms   2.54%  ▶⚠cc.Node.RenderCmd.transform                        CCNodeCanvasRenderCmd.js:122
  72.4ms   0.24%    746.9 ms   2.43%  ▶cc.V2F_C4B_T2F_Triangle                             CCTypes.js:623
 424.8ms   1.38%    741.6 ms   2.41%  ▶cc.color                                            CCTypesWebGL.js:46
 126.9ms   0.41%    736.4 ms   2.40%  ▶⚠cc.Sprite.extend.updateActFrames                   actor.js:1306
  34.6ms   0.11%    719.6 ms   2.34%  ▶BasePkProcesser.extend.pkAddBuffEffect              PkProcessor.js:501
  13.6ms   0.04%    713.3 ms   2.32%  ▶cc.LabelAtlas.cc.AtlasNode.extend.ctor              CCLabelAtlas.js:70
 375.5ms   1.22%    674.5 ms   2.19%  ▶cc.V2F_C4B_T2F                                      CCTypes.js:557
  12.6ms   0.04%    672.4 ms   2.19%  ▶⚠BasePkProcessor.extend.createPkObjBuffEffect       PkProcessor.js:938
```

CPU PROFILE

- 记录热点时段
- Self Time & Total Time
- 排除引擎正常损耗
- 定位性能热点
- Warning 符号： Not Optimized in JIT

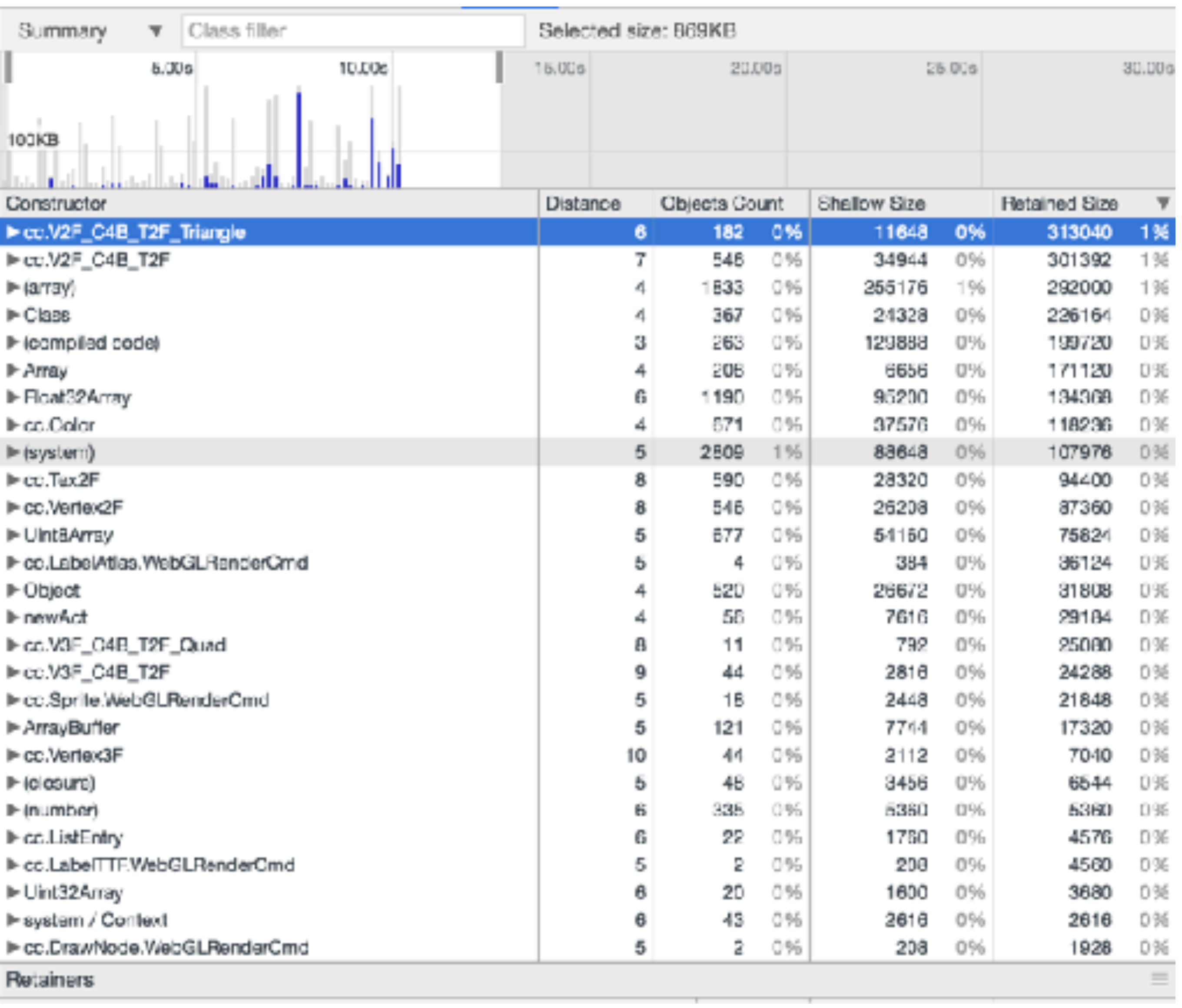
PERFORMANCE 工具

- Profile 工具用于总体分析
- Timeline 工具提供时间维度的分析
- 整体观察性能热点和内存
- 局部分析热点帧
- 调用栈



https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/timeline-tool

检测内存使用

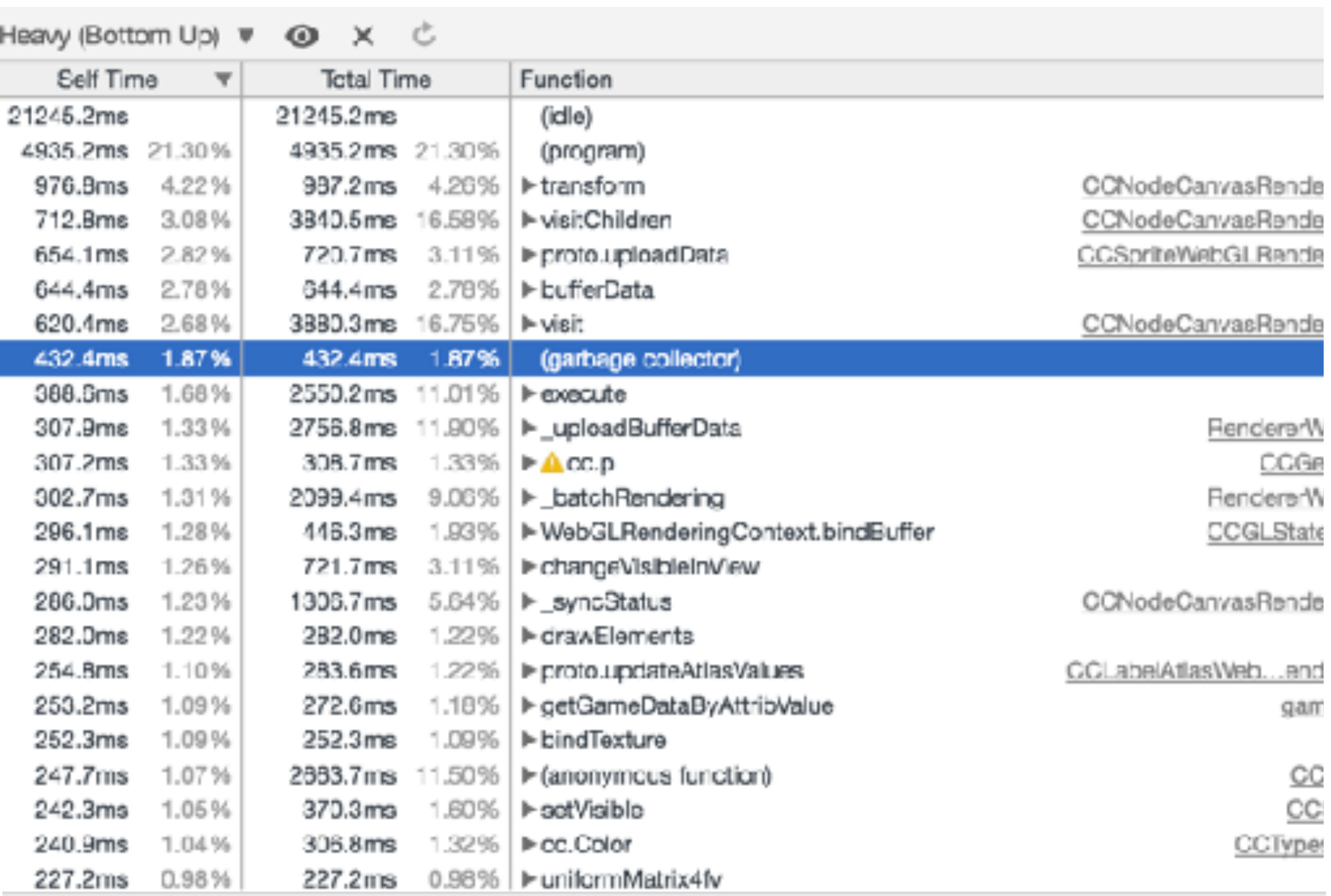


Constructor	Distance	Objects Count		Shallow Size		Retained Size	
►cc.V2F_C4B_T2F_Triangle	6	182	0%	11648	0%	313040	1%
►cc.V2F_C4B_T2F	7	546	0%	34944	0%	301392	1%
►(array)	4	1633	0%	255176	1%	292000	1%
►Class	4	367	0%	24328	0%	226164	0%
►(compiled code)	3	263	0%	129888	0%	199720	0%
►Array	4	206	0%	6656	0%	171120	0%
►Float32Array	6	1190	0%	95200	0%	134368	0%
►cc.Color	4	571	0%	37576	0%	118236	0%
►(system)	5	2509	1%	83648	0%	107976	0%
►cc.Tex2F	8	590	0%	28320	0%	94400	0%
►cc.Vertex2F	8	546	0%	26208	0%	87360	0%
►Uint8Array	5	677	0%	51150	0%	75824	0%
►cc.LabelAtlas.WebGLRenderCmd	5	4	0%	384	0%	36124	0%
►Object	4	520	0%	26672	0%	31808	0%
►newAct	4	56	0%	7616	0%	29184	0%
►cc.V3F_C4B_T2F_Quad	8	11	0%	792	0%	25080	0%
►cc.V3F_C4B_T2F	9	44	0%	2816	0%	24288	0%
►cc.Sprite.WebGLRenderCmd	5	16	0%	2448	0%	21848	0%
►ArrayBuffer	5	121	0%	7744	0%	17320	0%
►cc.Vertex3F	10	44	0%	2112	0%	7040	0%
►(closure)	5	48	0%	3456	0%	6544	0%
►(number)	6	335	0%	5360	0%	5360	0%
►cc.ListEntry	6	22	0%	1780	0%	4576	0%
►cc.LabelTTF.WebGLRenderCmd	5	2	0%	208	0%	4560	0%
►Uint32Array	6	20	0%	1600	0%	3680	0%
►system / Context	6	43	0%	2616	0%	2616	0%
►cc.DrawNode.WebGLRenderCmd	5	2	0%	208	0%	1928	0%

Retainers

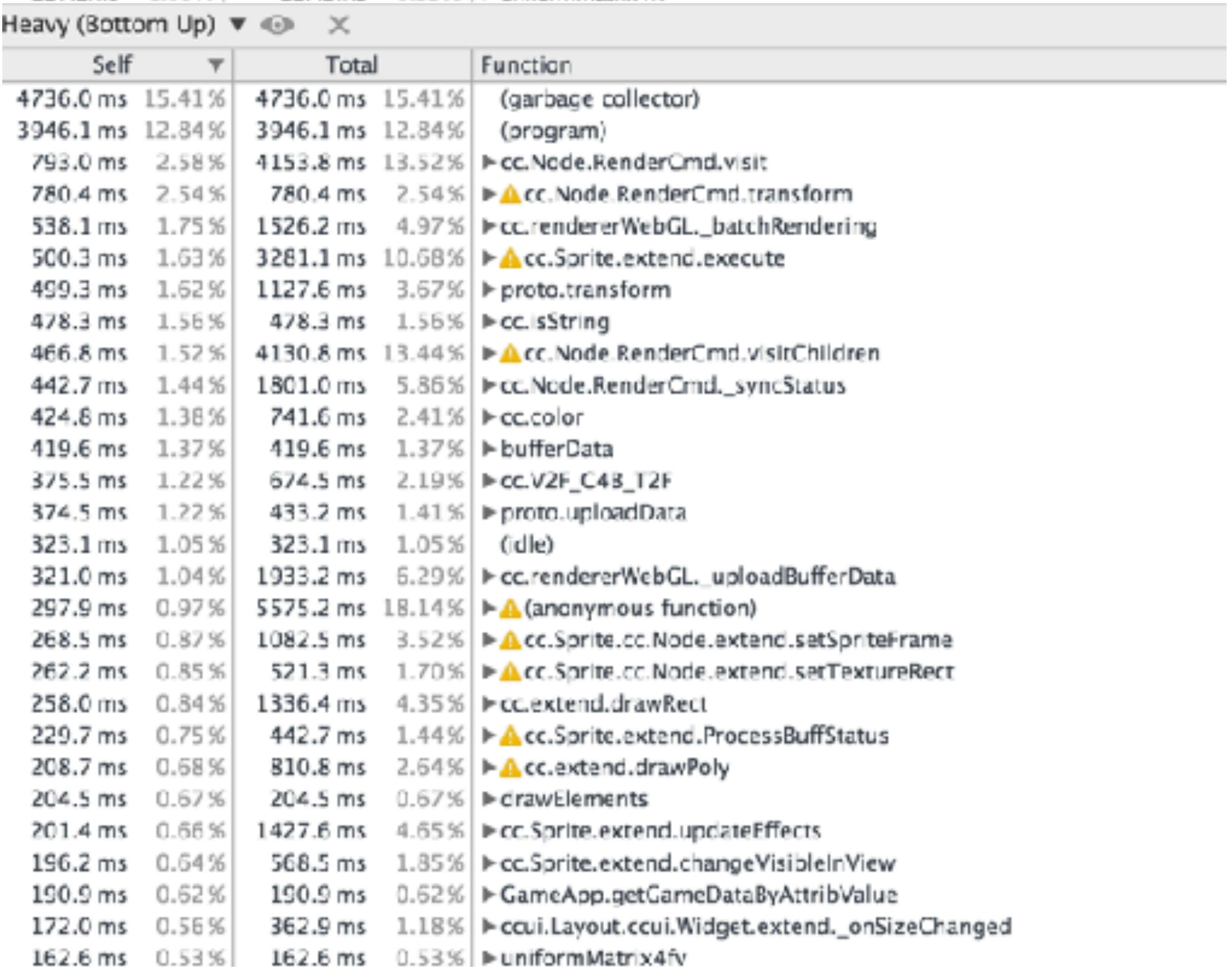
- ➤ Timeline 观察 GC 调用频率
- ➤ Allocation Timeline 中选取录制阶段的内存分配
- ➤ 蓝色代表新分配内存，灰色代表已回收内存，寻找长期不释放的蓝色内存
- ➤ 使用 Allocation Profile 观察产生内存的热点函数

https://developers.google.com/web/tools/chrome-devtools/memory-problems/



Heavy (Bottom Up)

Self Time		Total Time		Function	
21245.2ms		21245.2ms		(idle)	
4935.2ms	21.30%	4935.2ms	21.30%	(program)	
976.8ms	4.22%	987.2ms	4.26%	transform	CCNodeCanvasRende
712.8ms	3.08%	3840.5ms	16.58%	visitChildren	CCNodeCanvasRende
654.1ms	2.82%	720.7ms	3.11%	proto.uploadData	CCSpriteWebGLRende
644.4ms	2.78%	644.4ms	2.78%	bufferData	
620.4ms	2.68%	3880.3ms	16.75%	visit	CCNodeCanvasRende
432.4ms	1.87%	432.4ms	1.87%	(garbage collector)	
388.8ms	1.68%	2550.2ms	11.01%	execute	
307.9ms	1.33%	2756.8ms	11.90%	_uploadBufferData	RendererW
307.2ms	1.33%	308.7ms	1.33%	⚠ cc.p	CCGe
302.7ms	1.31%	2099.4ms	9.06%	_batchRendering	RendererW
296.1ms	1.28%	446.3ms	1.93%	WebGLRenderingContext.bindBuffer	CCGLState
291.1ms	1.26%	721.7ms	3.11%	changeVisibleInView	
286.0ms	1.23%	1306.7ms	5.64%	_syncStatus	CCNodeCanvasRende
282.0ms	1.22%	282.0ms	1.22%	drawElements	
254.8ms	1.10%	283.6ms	1.22%	proto.updateAtlasValues	CCLabelAtlasWeb...end
253.2ms	1.09%	272.6ms	1.18%	getGameDataByAttribValue	gam
252.3ms	1.09%	252.3ms	1.09%	bindTexture	
247.7ms	1.07%	2663.7ms	11.50%	(anonymous function)	CC
242.3ms	1.05%	370.3ms	1.60%	setVisible	CC
240.9ms	1.04%	306.8ms	1.32%	cc.Color	CCType
227.2ms	0.98%	227.2ms	0.98%	uniformMatrix4fv	



Heavy (Bottom Up)

Self		Total		Function
4736.0 ms	15.41%	4736.0 ms	15.41%	(garbage collector)
3946.1 ms	12.84%	3946.1 ms	12.84%	(program)
793.0 ms	2.58%	4153.8 ms	13.52%	cc.Node.RenderCmd.visit
780.4 ms	2.54%	780.4 ms	2.54%	⚠ cc.Node.RenderCmd.transform
538.1 ms	1.75%	1526.2 ms	4.97%	cc.rendererWebGL._batchRendering
500.3 ms	1.63%	3281.1 ms	10.68%	⚠ cc.Sprite.extend.execute
499.3 ms	1.62%	1127.6 ms	3.67%	proto.transform
478.3 ms	1.56%	478.3 ms	1.56%	cc.isString
466.8 ms	1.52%	4130.8 ms	13.44%	⚠ cc.Node.RenderCmd.visitChildren
442.7 ms	1.44%	1801.0 ms	5.86%	cc.Node.RenderCmd._syncStatus
424.8 ms	1.38%	741.6 ms	2.41%	cc.color
419.6 ms	1.37%	419.6 ms	1.37%	bufferData
375.5 ms	1.22%	674.5 ms	2.19%	cc.V2F_C4B_T2F
374.5 ms	1.22%	433.2 ms	1.41%	proto.uploadData
323.1 ms	1.05%	323.1 ms	1.05%	(idle)
321.0 ms	1.04%	1933.2 ms	6.29%	cc.rendererWebGL._uploadBufferData
297.9 ms	0.97%	5575.2 ms	18.14%	⚠ (anonymous function)
268.5 ms	0.87%	1082.5 ms	3.52%	⚠ cc.Sprite.cc.Node.extend.setSpriteFrame
262.2 ms	0.85%	521.3 ms	1.70%	⚠ cc.Sprite.cc.Node.extend.setTextureRect
258.0 ms	0.84%	1336.4 ms	4.35%	cc.extend.drawRect
229.7 ms	0.75%	442.7 ms	1.44%	⚠ cc.Sprite.extend.ProcessBuffStatus
208.7 ms	0.68%	810.8 ms	2.64%	⚠ cc.extend.drawPoly
204.5 ms	0.67%	204.5 ms	0.67%	drawElements
201.4 ms	0.66%	1427.6 ms	4.65%	cc.Sprite.extend.updateEffects
196.2 ms	0.64%	568.5 ms	1.85%	cc.Sprite.extend.changeVisibleInView
190.9 ms	0.62%	190.9 ms	0.62%	GameApp.getGameDataByAttribValue
172.0 ms	0.56%	362.9 ms	1.18%	ccui.Layout.ccui.Widget.extend._onSizeChanged
162.6 ms	0.53%	162.6 ms	0.53%	uniformMatrix4fv

远程调试

- 不同设备 / 浏览器的表现差异巨大
- Android Chrome / X5 远程调试
- iOS Safari 远程调试比较弱

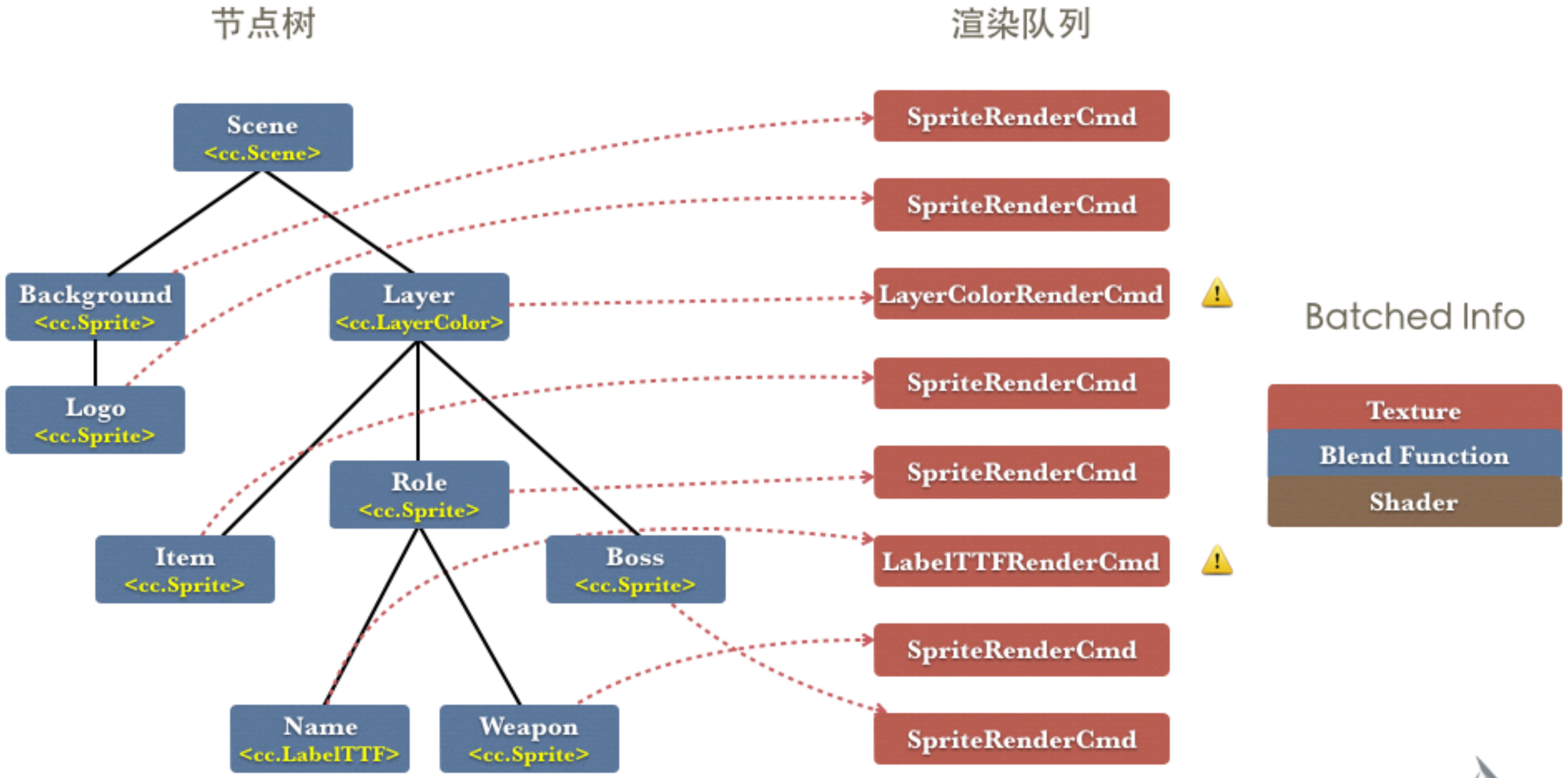
https://developers.google.com/web/tools/chrome-devtools/remote-debugging/

加载优化

- project.json 选择模块
- moduleConfig.json 定制模块
- Google Closure Compiler 高级压缩
- 优化资源加载（减少并发，优化缓存策略，缩短 TTFB）

渲染优化

自动批处理渲染


节点树
渲染队列
Scene <cc.Scene>
Background <cc.Sprite>
Layer <cc.LayerColor>
Logo <cc.Sprite>
Role <cc.Sprite>
Item <cc.Sprite>
Boss <cc.Sprite>
Name <cc.LabelTTF>
Weapon <cc.Sprite>
SpriteRenderCmd
SpriteRenderCmd
LayerColorRenderCmd
SpriteRenderCmd
SpriteRenderCmd
LabelTTFRenderCmd
SpriteRenderCmd
SpriteRenderCmd
Batched Info
Texture
Blend Function
Shader

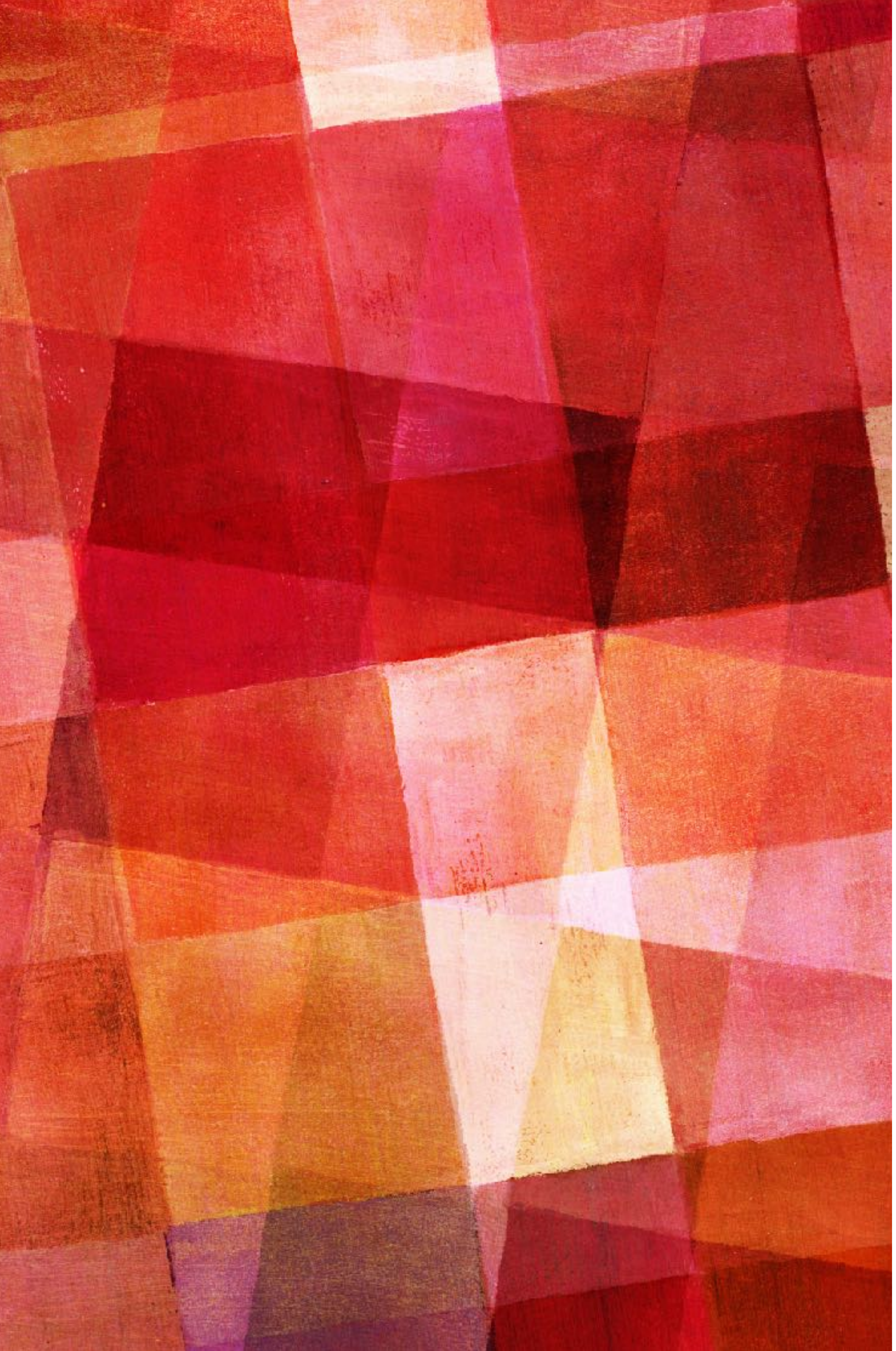
自动批处理渲染相关细节

- 目前对 Sprite, Tilemap, BMFont Label, Atlas Label, DragonBones 等有效
- LabelTTF, Spine 会打断批处理

BEST PRACTICE

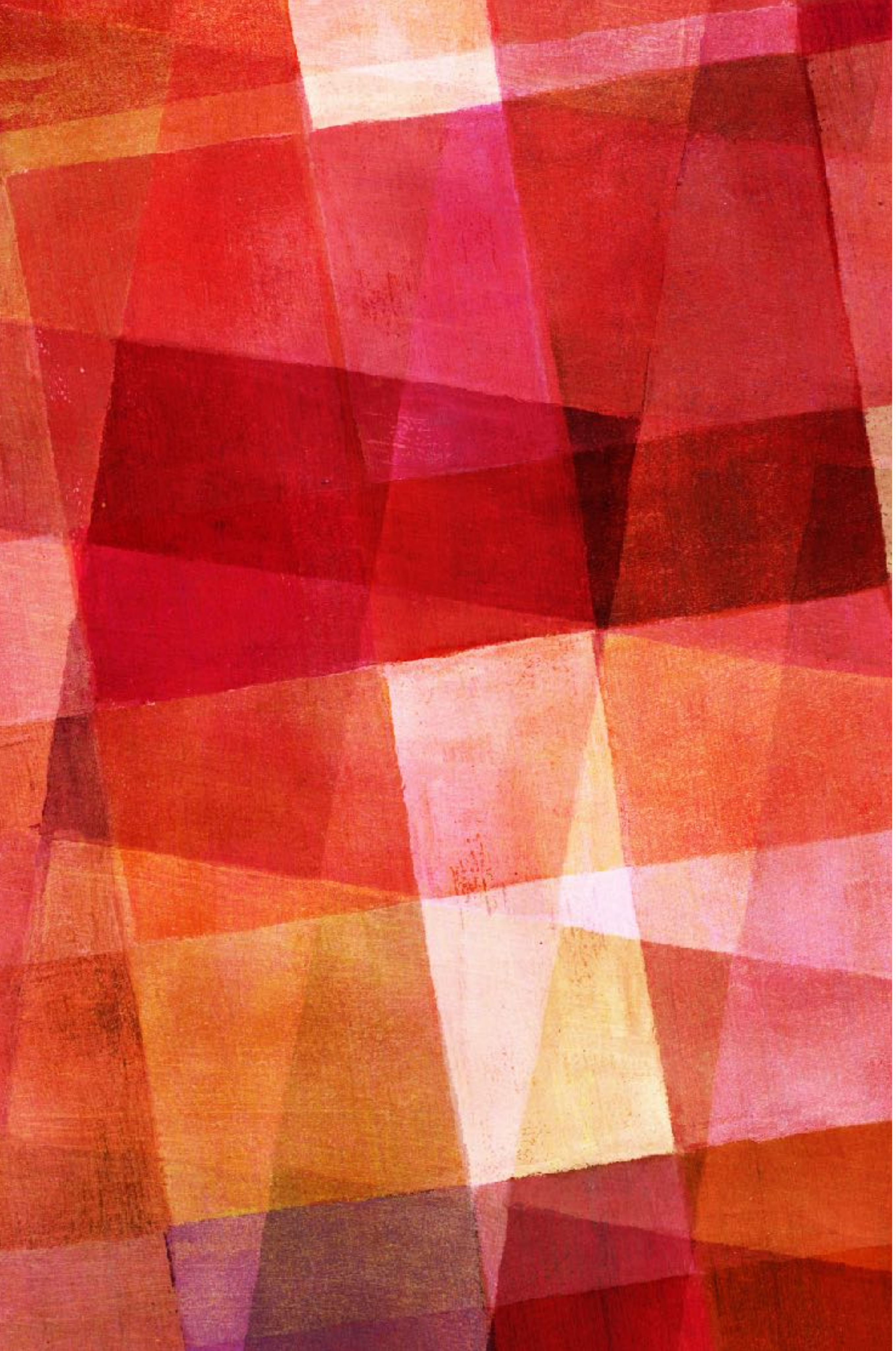
- 图集按照场景结构来合并：比如场景包含 地图，角色，UI 三大层，地图和 UI 应该分别使用一个图集
- 图文分层：比较复杂的图层比如角色层，能够合并图集的 Sprite 尽量排在一起，BMFont Label 排在一起，无法批处理的节点排在一起。比如 RPG 中可以是：
    - 角色 / 敌人
    - 特效
    - DrawNode
    - 姓名 + 其他 LabelTTF 节点
    - BMFont Label 提示文字
- 优化图集来提高批处理效率
- 动态合并图集，在运行时将可以批处理的图集合并在一起

内存优化



找到垃圾

- 用 Allocation Profile 定位
- 注意简单的对象的创建，比如数组，{}
- 匿名函数很容易被忽略
- 匿名函数使用的外部变量将被匿名函数持有



复用一切可复用的对象

- 不止是为了节省创建时的开销，更重要的是避免 GC 的开销
- 可复用的对象有：
  - 同类型节点：比如战斗的提示文字，比如敌人和血槽
  - cc.v2，cc.color 等基础对象
  - 数组和对象 {}
- 复杂类型用对象缓冲池，基础对象直接复用
- 复用时避免泄漏为全局变量

CPU 占用优化

CPU 占用优化

- 使用 JSHint
- 60 fps -> 30 fps
- 减少音频使用，一方面降低内存占用和 CPU 消耗，另一方面，现在 H5 的音频支持本来就很差，就算做的再好最后也可能播不了
- 降低调用栈深度，调用栈越深，终端 JS 执行效率越差
- 解决 JIT Not Optimized issue
- 注意不同的浏览器和 JS 引擎的特性不同，需要大量测试优化效果

JS 优化文章

- https://github.com/v8/v8/wiki/Design%20Elements
- https://github.com/petkaantonov/bluebird/wiki/Optimization-killers
- https://github.com/GoogleChrome/devtools-docs/issues/53
- https://github.com/vhf/v8-bailout-reasons/blob/master/README.md
- https://github.com/thlorenz/v8-perf/blob/master/compiler.md
- Google 搜索 JS tricks

JS Optimization Hell

- for-of 循环
- try-catch or try-finally
- __proto__, or get or set
- eval
- with

JS Optimization Hell - arguments

```
function defaultArgsReassign(a, b) {
    if (arguments.length < 2) b = 5;
}

function leaksArguments1() {
   return arguments;
}
```

JS Optimization Hell - for in

```
var key;
function nonLocalKey2() {
    var obj = {};
    for(key in obj);
}

function nonLocalKey1() {
    var obj = {};
    for(var key in obj);
    return function() {
        return key;
    };
}
```

```
function iteratesOverArray() {
    var arr = [1, 2, 3];
    for (var index in arr) {
    }
}
```