

从用户出发的交互类性能 流畅度

腾讯 victorhuang

有界面的才有的交互类性能

	最佳	好	可接受
响应延迟	≤ 100 ms	≤ 200 ms	≤ 500 ms
图形动画	≥ 120 fps	≥ 60 fps	≥ 30 fps
视频回放	≥ 60 fps	≥ 30 fps	≥ 20 fps

表 1：用户体验的行业经验值的示例（来源：英特尔公司，2011 年）

启动速度、界面切换速度、图片加载速度、视频首帧
流畅度

有界面的才有的交互类性能

	最佳	好	可接受
响应延迟	$\cong 100\text{ ms}$	$\cong 200\text{ ms}$	$\cong 500\text{ ms}$
图形动画	$\cong 120\text{ fps}$	$\cong 60\text{ fps}$	$\cong 30\text{ fps}$
视频回放	$\cong 60\text{ fps}$	$\cong 30\text{ fps}$	$\cong 20\text{ fps}$

表 1：用户体验的行业经验值的示例（来源：英特尔公司，2011 年）

启动速度、界面切换速度、图片加载速度、视频首帧

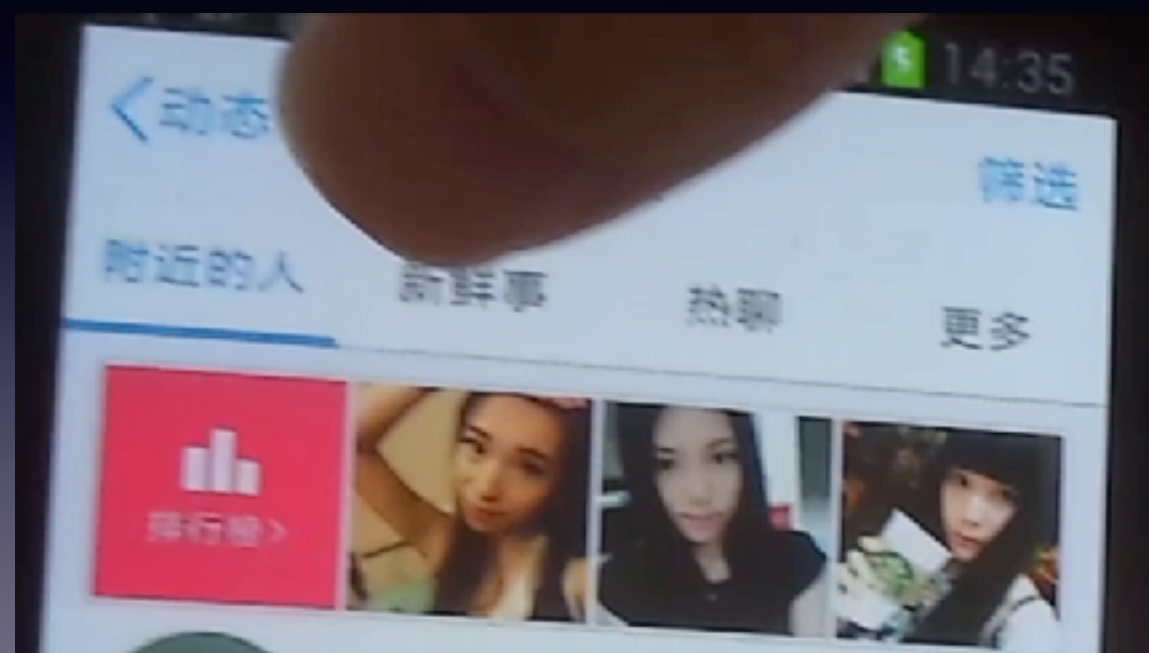
流畅度

来自用户的吐槽：流畅度

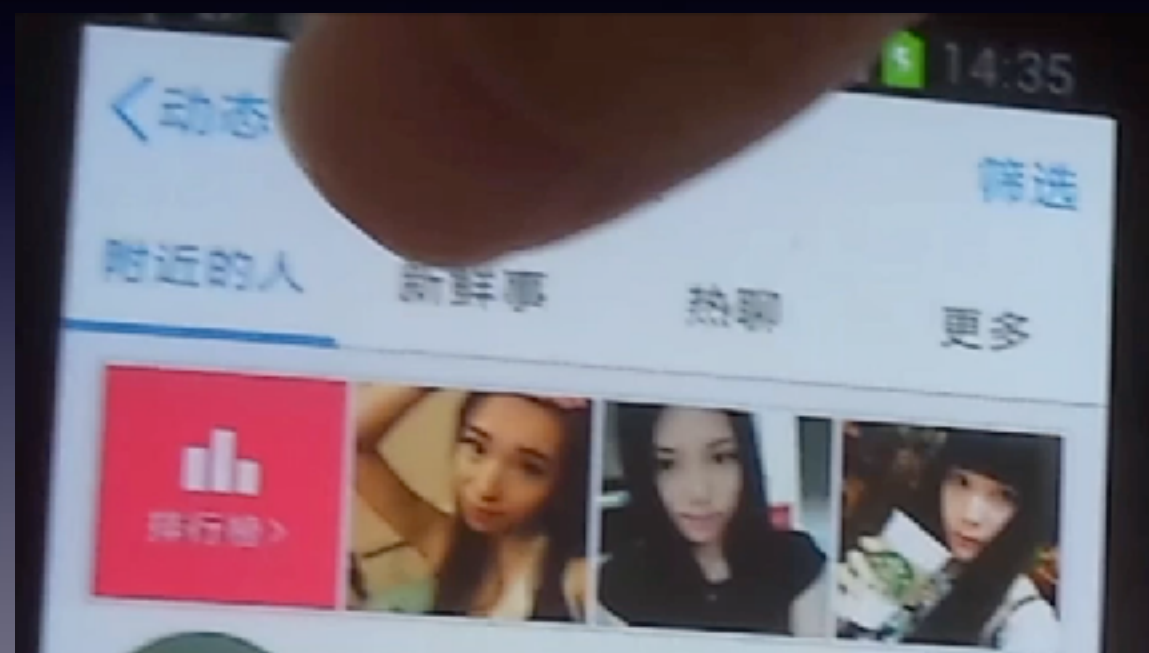
- 很卡
- 卡顿
- 卡死
- ANR

APP版本	OS	厂商	标题	内容
7.7.8.421	11_4_1	iphone6,1	"用户名片与资料"卡	进入"用户名片与资料"卡;这次qq更新后 无法在好友资料里复制对方qq号码了,以前可以在qq号的地方按住屏幕复制的 现在不行了 很不方便;首页描述:这次qq更新后 无法在好友资料里复制对方qq号码了,以前可以在qq号的地方按住屏幕复制的 现在不行了 很不方便 -Times2018年09月04日12:57
7.6.9.450	11_3	iphone8,4	问题与建议	没有银行卡怎么办
7.7.8.421	11_1	iphone8,1	"附近"相关功能卡	打开"热聊"页面卡 -times2018年09月04日12:50
7.7.8.421	11_4_1	-	问题与建议	反应慢 卡顿 卡卡卡
7.6.8.466	10_8_1	iphone7,2	问题与建议	我被骗了,希望尽快删除银行卡

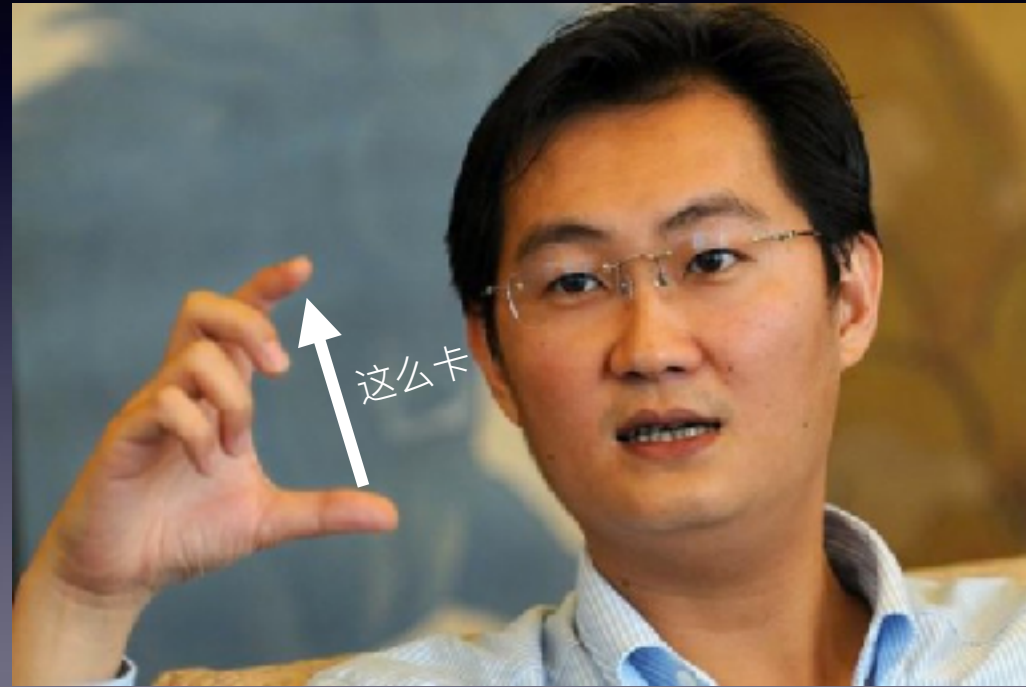
感受一下卡顿



感受一下卡顿



老板可以说



发现：必须要量化



“If You Can't Measure It, You Can't Manage It”

彼得·德鲁克



发现问题的手段

- 开发者选项(gfxinfo)
- SurfaceFlinger
- 日志里面的SkipFrame
- Looper
- DoFrame

gfxinfo

- 仅仅适用于硬件加速
- Draw + Process + Execute

gfxinfo

- 仅仅适用于硬件加速
- Draw + Process + Exec



SurfaceFlinger

- 4.1以上
- `dumpsys SurfaceFlinger--latency <layer-name/activity-name>`
- 4.1以下
- `service call SurfaceFlinger 1013`

动态刷新FrameRate

帧序	帧率版本	对比版本	最大值	变化	对比标准	说明
212	3.0.0	3.0.0	12.0	+ 5.0	5.0	正常
Task	3.0.0	3.0.0	5.0	+ 0.5	0.5	低于4个

```

shell@android:/ $ dumpsys SurfaceFlinger --latency com.tencent.mobileqq/com.tenc
ent.mobileqq.activity.SplashActivity
cy com.tencent.mobileqq/com.tencent.mobileqq.activity.SplashActivity      (
16383723
42999417419368 42999430969164 42999432586596
42999414814379 42999447589092 42999448618842
42999419896606 42999464019701 42999464721606
42999465301448 42999480255053 42999480956957
42999481536791 42999496612475 42999497497485
42999497974217 42999511122485 42999511422778

```

开始绘制

垂直同步

刷新该帧到屏幕

```

shell@android:/ # service call SurfaceFlinger 1013
service call SurfaceFlinger 1013
Result: Parcel(000013a8) '....'

```

Buffer交换
次数

接下来的每一行（最多128行）为每一帧的3个不同时间点的时间戳

- A) 应用程序开始绘制这一帧的时间戳
- B) SurfaceFlinger开始进行垂直同步刷新该帧到屏幕的时间戳
- C) SurfaceFlinger完成垂直同步刷新该帧到屏幕的时间戳

上图红框中数值是Surface buffer中两个buffer从系统启动到当前时刻的交换次数（每交换一次，屏幕刷新一帧。通过两次取样相减，即可以得到取样时间内刷新的帧数得到FPS值。

Perfbox: FPSTest



Perfbox: FPSTest



SkipFrame

- AndoridVersion \geq 4.1
- `debug.choreographer.skipwarning` $> 30 = 480\text{ms}$

10-16 17:34:15.632 I/Choreographer(12451): Skipped 52 frames! The application may be doing too much work on its main thread.

无法发现精细的

发现：完整且量化的流畅度指标

FPS

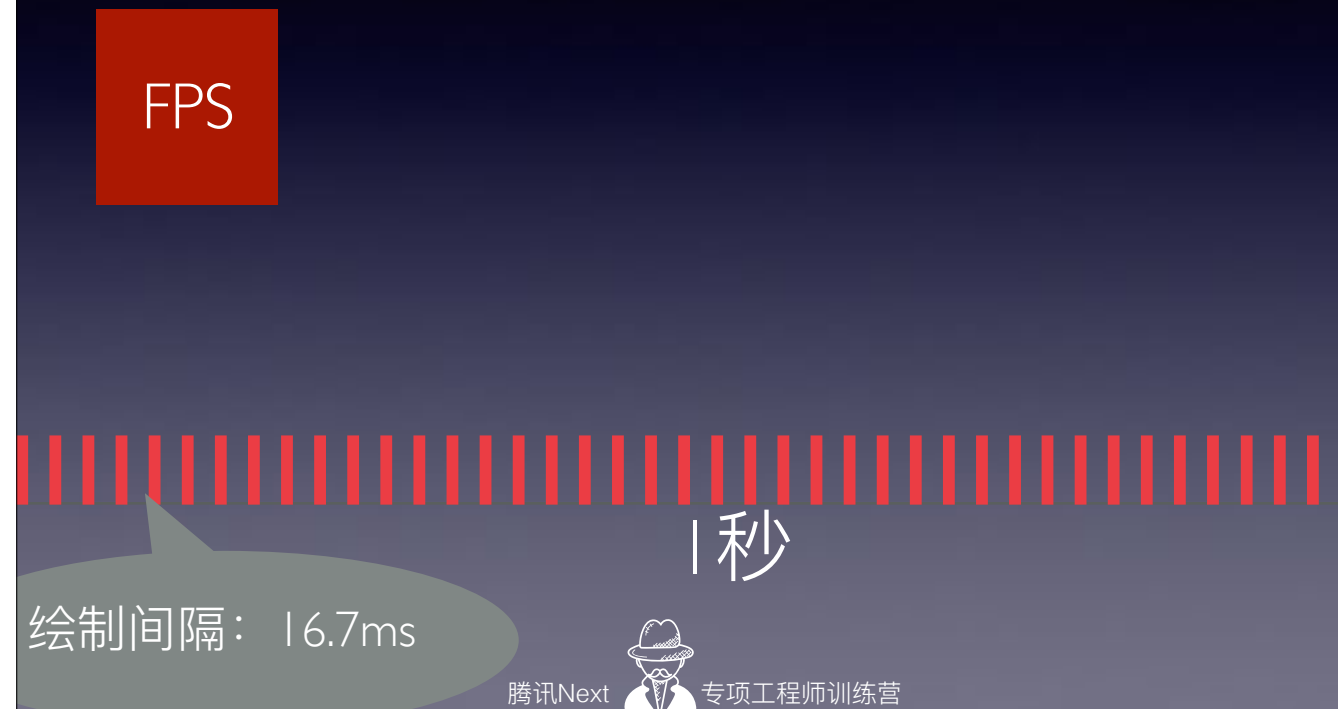
1 秒



Looper:

Android主线程更新UI。如果界面1秒钟刷新少于60次，即FPS小于60，用户就会产生卡顿感觉。简单来说，Android使用消息机制进行UI更新，UI线程有个Looper，在其loop方法中会不断取出message，调用其绑定的Handler在UI线程执行。如果在handler的dispatchMesaage方法里有耗时操作，就会发生卡顿。

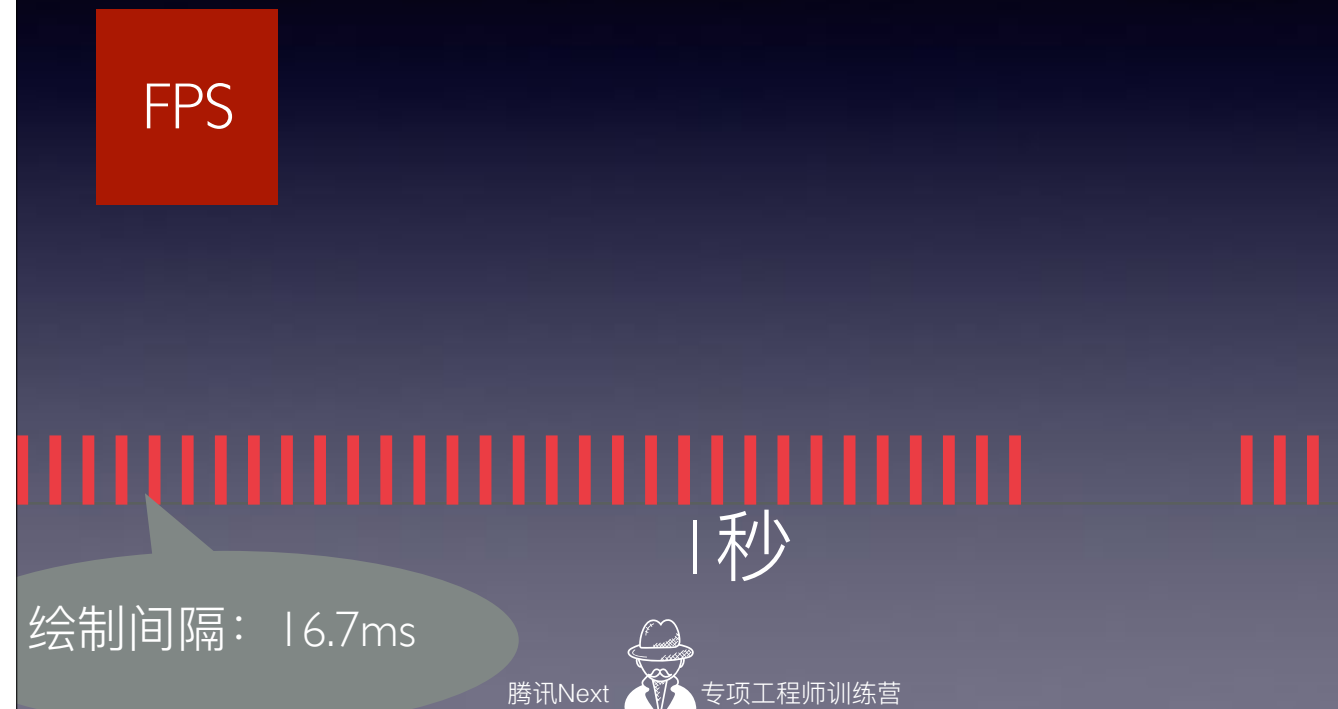
发现：完整且量化的流畅度指标



Looper:

Android主线程更新UI。如果界面1秒钟刷新少于60次，即FPS小于60，用户就会产生卡顿感觉。简单来说，Android使用消息机制进行UI更新，UI线程有个Looper，在其loop方法中会不断取出message，调用其绑定的Handler在UI线程执行。如果在handler的dispatchMesaage方法里有耗时操作，就会发生卡顿。

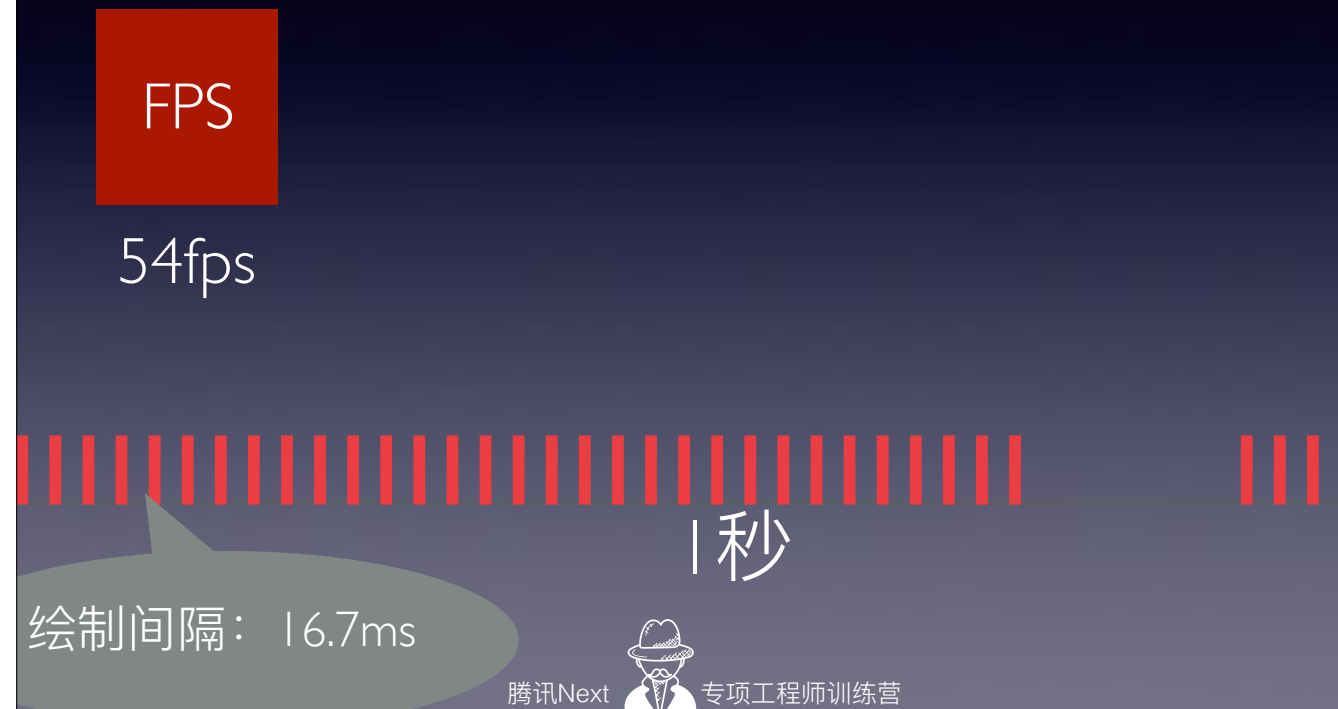
发现：完整且量化的流畅度指标



Looper:

Android主线程更新UI。如果界面1秒钟刷新少于60次，即FPS小于60，用户就会产生卡顿感觉。简单来说，Android使用消息机制进行UI更新，UI线程有个Looper，在其loop方法中会不断取出message，调用其绑定的Handler在UI线程执行。如果在handler的dispatchMesaage方法里有耗时操作，就会发生卡顿。

发现：完整且量化的流畅度指标



Looper:

Android主线程更新UI。如果界面1秒钟刷新少于60次，即FPS小于60，用户就会产生卡顿感觉。简单来说，Android使用消息机制进行UI更新，UI线程有个Looper，在其loop方法中会不断取出message，调用其绑定的Handler在UI线程执行。如果在handler的dispatchMesaage方法里有耗时操作，就会发生卡顿。

发现：完整且量化的流畅度指标



Looper:

Android主线程更新UI。如果界面1秒钟刷新少于60次，即FPS小于60，用户就会产生卡顿感觉。简单来说，Android使用消息机制进行UI更新，UI线程有个Looper，在其loop方法中会不断取出message，调用其绑定的Handler在UI线程执行。如果在handler的dispatchMesaage方法里有耗时操作，就会发生卡顿。

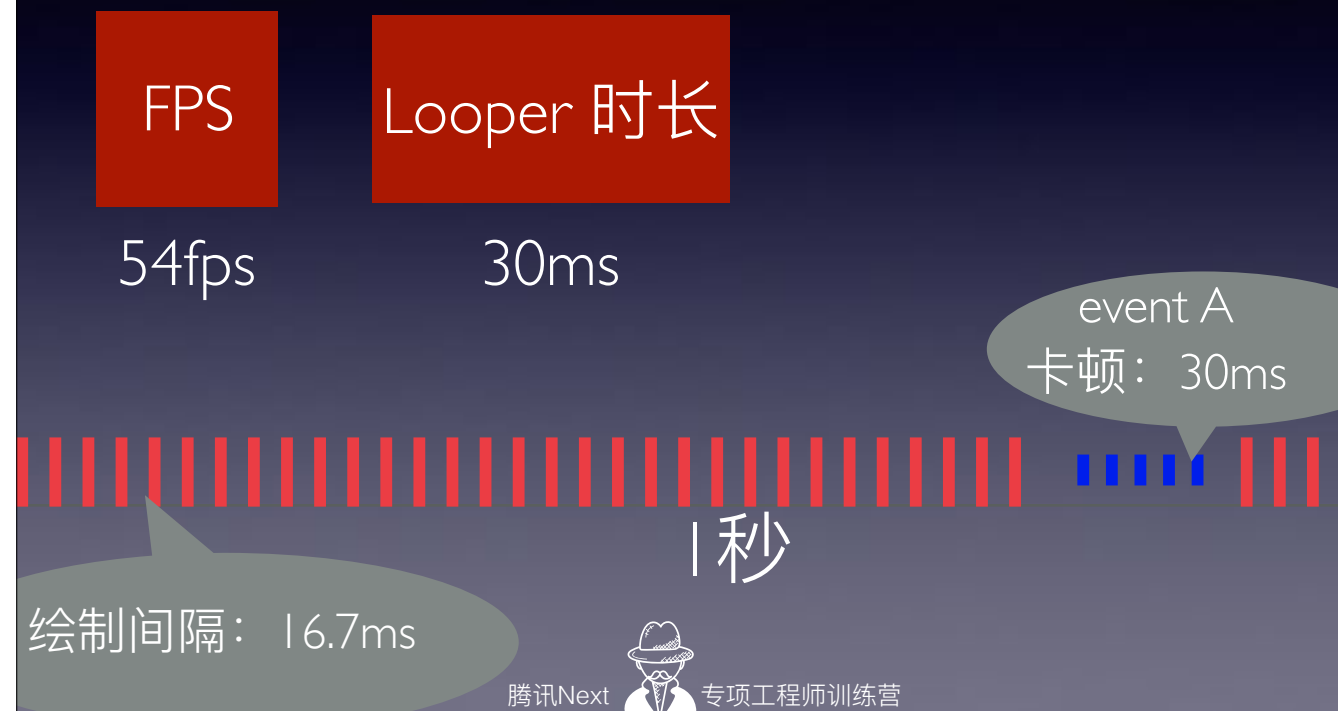
发现：完整且量化的流畅度指标



Looper:

Android主线程更新UI。如果界面1秒钟刷新少于60次，即FPS小于60，用户就会产生卡顿感觉。简单来说，Android使用消息机制进行UI更新，UI线程有个Looper，在其loop方法中会不断取出message，调用其绑定的Handler在UI线程执行。如果在handler的dispatchMesaage方法里有耗时操作，就会发生卡顿。

发现：完整且量化的流畅度指标



Looper:

Android主线程更新UI。如果界面1秒钟刷新少于60次，即FPS小于60，用户就会产生卡顿感觉。简单来说，Android使用消息机制进行UI更新，UI线程有个Looper，在其loop方法中会不断取出message，调用其绑定的Handler在UI线程执行。如果在handler的dispatchMesaage方法里有耗时操作，就会发生卡顿。

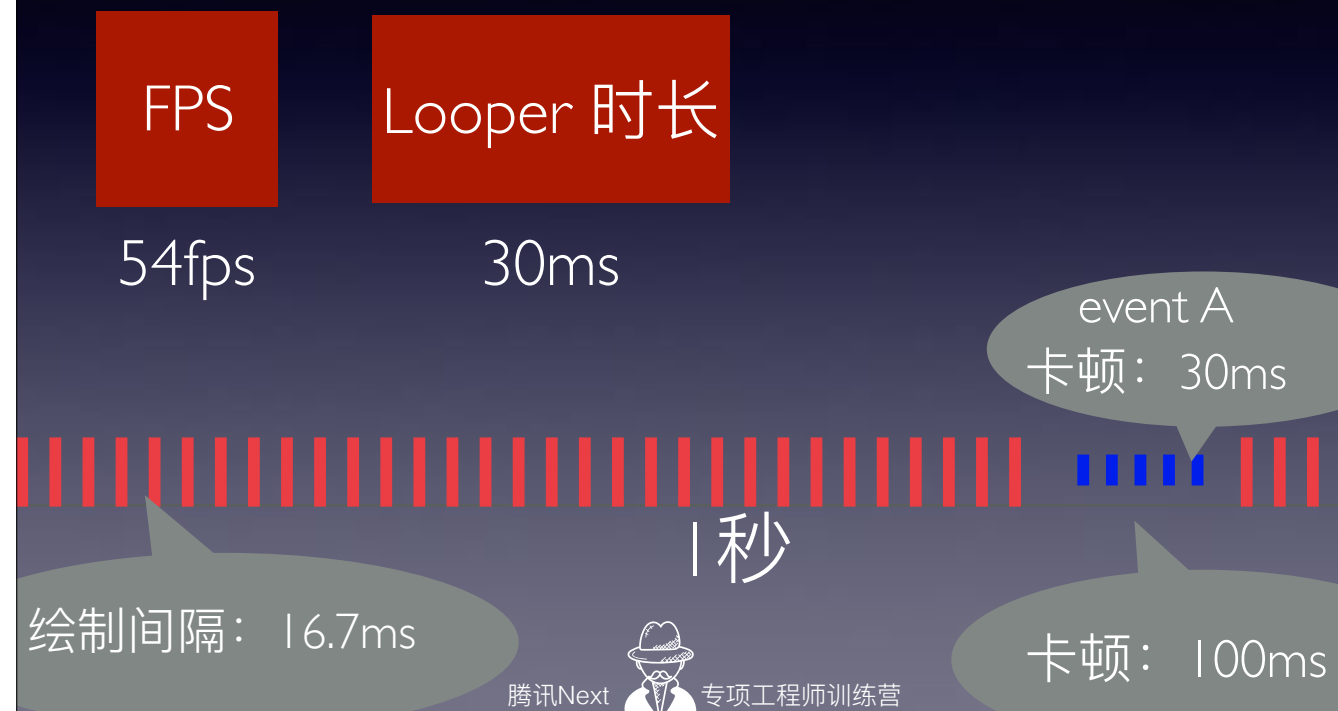
发现：完整且量化的流畅度指标



Looper:

Android主线程更新UI。如果界面1秒钟刷新少于60次，即FPS小于60，用户就会产生卡顿感觉。简单来说，Android使用消息机制进行UI更新，UI线程有个Looper，在其loop方法中会不断取出message，调用其绑定的Handler在UI线程执行。如果在handler的dispatchMesaage方法里有耗时操作，就会发生卡顿。

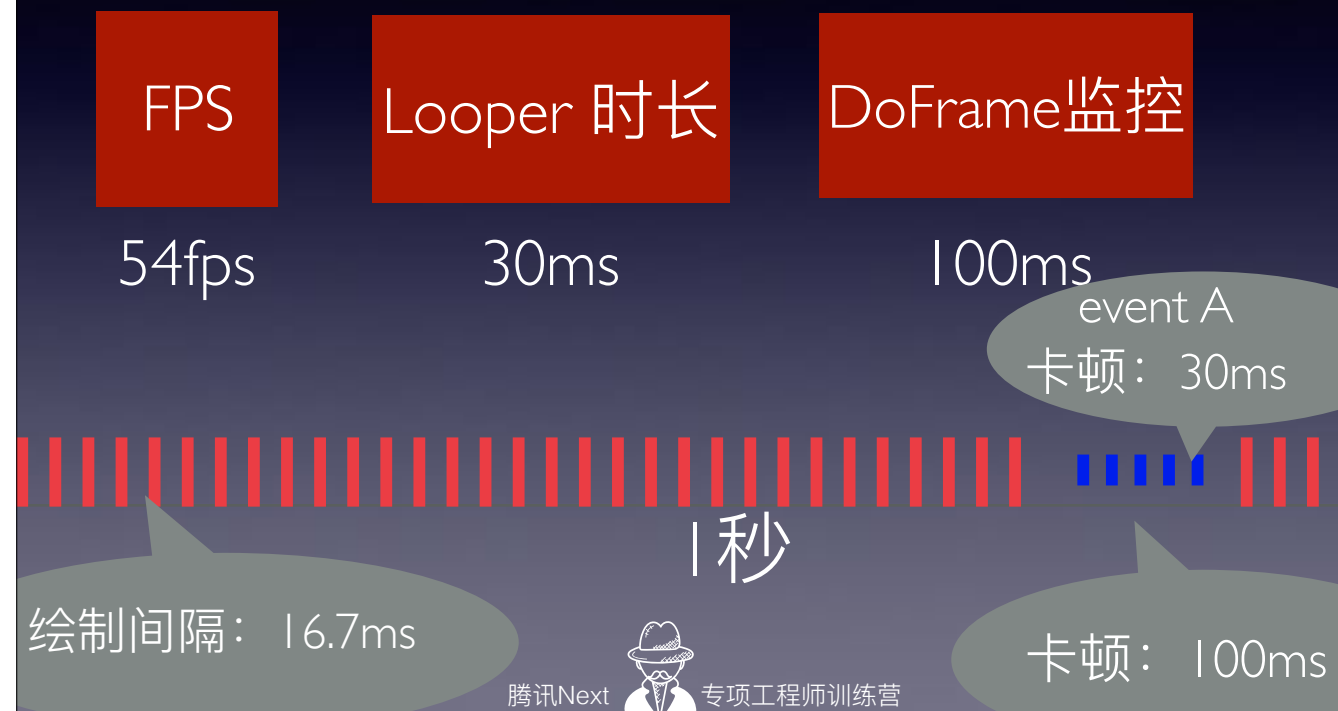
发现：完整且量化的流畅度指标



Looper:

Android主线程更新UI。如果界面1秒钟刷新少于60次，即FPS小于60，用户就会产生卡顿感觉。简单来说，Android使用消息机制进行UI更新，UI线程有个Looper，在其loop方法中会不断取出message，调用其绑定的Handler在UI线程执行。如果在handler的dispatchMesaage方法里有耗时操作，就会发生卡顿。

发现：完整且量化的流畅度指标



Looper:

Android主线程更新UI。如果界面1秒钟刷新少于60次，即FPS小于60，用户就会产生卡顿感觉。简单来说，Android使用消息机制进行UI更新，UI线程有个Looper，在其loop方法中会不断取出message，调用其绑定的Handler在UI线程执行。如果在handler的dispatchMesaage方法里有耗时操作，就会发生卡顿。

DoFrame监控具体怎么做?

- 注册Choreographer.FrameCallback
- 16.7ms回调doFrame
- mainlooper放入需要处理的message

小结

	SurfaceFlinger	gfxinfo	Looper.loop	Choreographer.FrameCallback
监控是否卡顿	√	√	√	√
支持静态页面 卡顿检测	×	×	√	√
支持计算帧率	√	√	×	√
支持获取App运行信息	×	×	√	√



“无法定位的性能问题等于不存在”

定界：利用资源类数据定界

- 磁盘：I/O Wait (TOP, hook)
- 内存：GC for Alloc (GC日志, alloc tracer)
- CPU：CPU jiffer (proc/pid/stat)



后面资源类性能的课程会详细介绍这里每个资源的问题

定位：利用分析型数据定位

- 与定界的区别，不定位到具体代码都不叫定位
- 至上而下
- 至下而上

至上而下



编舞者choreographer，有点像老板， 屏幕要”画“，信息告诉了画家， 画家就开始画， 也就是doframe， doframe的第一步是衡量， 然后就是布局， 布局完之后就开画了， 画的第一步是构思， 要画什么？ 那就是getdisplaylist， 然后开始动笔就是drawdislaylist， 然后交给老板， swapbuffers.



编舞者choreographer，有点像老板， 屏幕要”画“，信息告诉了画家， 画家就开始画， 也就是doframe， doframe的第一步是衡量， 然后就是布局， 布局完之后就开画了， 画的第一步是构思， 要画什么？ 那就是getdisplaylist， 然后开始动笔就是drawdislaylist， 然后交给老板， swapbuffers.



编舞者choreographer，有点像老板， 屏幕要”画“，信息告诉了画家， 画家就开始画， 也就是doframe， doframe的第一步是衡量， 然后就是布局， 布局完之后就开画了， 画的第一步是构思， 要画什么？ 那就是getdisplaylist， 然后开始动笔就是drawdislaylist， 然后交给老板， swapbuffers.



编舞者choreographer，有点像老板， 屏幕要”画“，信息告诉了画家， 画家就开始画， 也就是doframe， doframe的第一步是衡量， 然后就是布局， 布局完之后就开画了， 画的第一步是构思， 要画什么？ 那就是getdisplaylist， 然后开始动笔就是drawdislaylist， 然后交给老板， swapbuffers.



编舞者choreographer，有点像老板， 屏幕要”画“，信息告诉了画家， 画家就开始画， 也就是doframe， doframe的第一步是衡量， 然后就是布局， 布局完之后就开画了， 画的第一步是构思， 要画什么？ 那就是getdisplaylist， 然后开始动笔就是drawdislaylist， 然后交给老板， swapbuffers.



编舞者choreographer，有点像老板， 屏幕要”画“，信息告诉了画家， 画家就开始画， 也就是doFrame， doFrame的第一步是衡量， 然后就是布局， 布局完之后就开画了， 画的第一步是构思， 要画什么？ 那就是getDisplayList， 然后开始动笔就是drawDisplayList， 然后交给老板， swapBuffers.



编舞者choreographer，有点像老板， 屏幕要”画“，信息告诉了画家， 画家就开始画， 也就是doframe， doframe的第一步是衡量， 然后就是布局， 布局完之后就开画了， 画的第一步是构思， 要画什么？ 那就是getdisplaylist， 然后开始动笔就是drawdislaylist， 然后交给老板， swapbuffers.

定位策略：至上而下



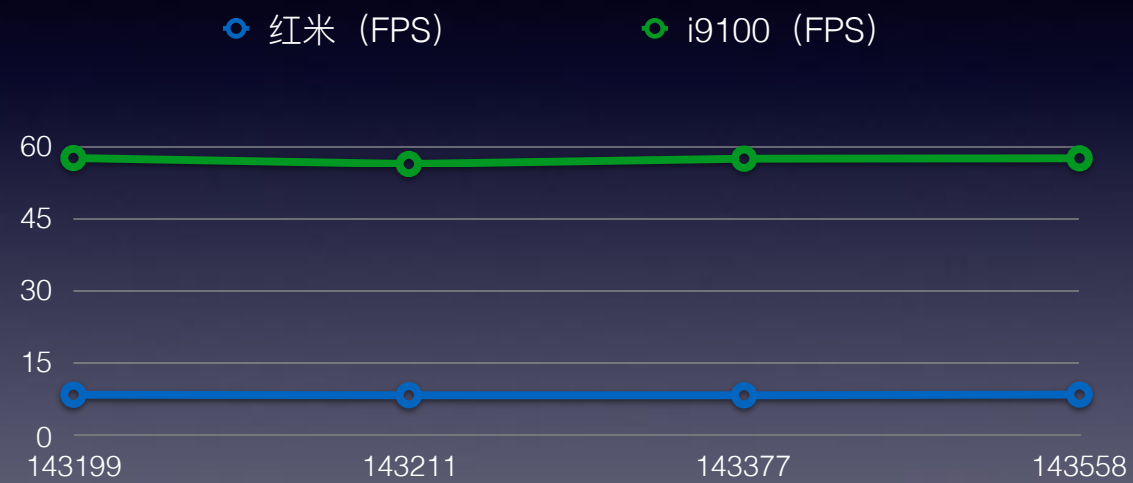
我们的整体结构，以ListView滑动的经典场景为例。我们可以提取出关键函数，已经对应的。

5.0有所不同，在DRAW的部分多了个RenderThread以及一系列关键函数。好，我们看下针对这个案例是否有效。

measure，获取宽和高，并设置给每个View

layout，设置子View的大小和位置。这时ViewGroup根本就不知道子View的大小，onLayout告诉ViewGroup，子View在它里面中的大小和应该放在哪里。注意两个的区别，我当时也被搞得一头雾水。

案例：红米会话列表卡顿



按照之前说的Measure， 我们已经有了上面的结果， 蓝色代表的红米， 和i9100相差N倍， 同一个版本， 卡顿到极致。一堆卡顿冲过来， 我们先说第一个案例。

红米会话列表卡顿

getView

measure

layout

draw

会到问题，我们针对会话列表的场景录制了trace, 发现getView, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿

getView

name	incl CPU Time %	incl CPU Time	Excl CPU Time %	Excl CPU Time	incl Real Time %	incl Real Time	Excl Real Time %	Excl Real Time	LAUNCHER
com.tencent.migot/ heads/ViewLrAdaptiveView	7.5%	297.907	0.0%	0.000	0.7%	277.921	0.6%	6.209	200+2
measure									
layout									
draw									

会到问题，我们针对会话列表的场景录制了trace, 发现getView, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿

getView

name	incl CPU Time %	incl CPU Time	Excl CPU Time %	Excl CPU Time	incl Real Time %	incl Real Time	Excl Real Time %	Excl Real Time	LAUNCHER
53 com.tencent.tmgts/ headsViewListAdaptiveView	7.5%	297.907	0.0%	0.000	0.7%	271.921	0.0%	0.209	200+2

measure

24 android.view.View.measure()V	22.2%	1215.169	0.0%	1.000	3.2%	3040.000	0.0%	1.100	3194072
---------------------------------	-------	----------	------	-------	------	----------	------	-------	---------

layout

draw

会到问题，我们针对会话列表的场景录制了trace, 发现getView, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿

getView

name	incl Call Time %	incl Call Time	Excl Call Time %	Excl Call Time	incl Real Time %	incl Real Time	Excl Real Time %	Excl Real Time	LAUNCHER
53 com.tencent.tmgts/ headerViewListAdapter.getView	7.5%	297.907	0.0%	0.000	0.7%	271.921	0.0%	0.000	200+2

measure

24 android.view.ViewGroup.measure (UI)	22.2%	1215.169	0.0%	0.000	3.2%	3040.000	0.0%	0.000	319+672
--	-------	----------	------	-------	------	----------	------	-------	---------

layout

80 android.view.ViewGroup.layout (UI)	3.1%	121.379	0.0%	0.000	0.3%	293.900	0.0%	0.000	110+22
---------------------------------------	------	---------	------	-------	------	---------	------	-------	--------

draw

会到问题，我们针对会话列表的场景录制了trace, 发现getView, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿

getView

name	incl CPU Time %	incl CPU Time	Excl CPU Time %	Excl CPU Time	incl Real Time %	incl Real Time	Excl Real Time %	Excl Real Time	LAUNCHER
53 com.tencent.tmgts/.../MediaViewListAdapter.getView	7.5%	297.907	0.0%	0.000	0.7%	271.921	0.0%	0.209	200+2

measure

24 android.view.View.measure()V	22.2%	1215.269	0.0%	0.000	3.2%	2040.000	0.0%	0.000	319+672
---------------------------------	-------	----------	------	-------	------	----------	------	-------	---------

layout

80 android.view.ViewGroup.layout()V	3.1%	121.379	0.0%	0.000	0.3%	283.900	0.0%	0.000	110+22
-------------------------------------	------	---------	------	-------	------	---------	------	-------	--------

draw

27 android.view.View.draw()V	34.4%	108.959	0.0%	0.000	2.7%	2931.844	0.0%	0.000	60+2
------------------------------	-------	---------	------	-------	------	----------	------	-------	------

会到问题，我们针对会话列表的场景录制了trace，发现getView, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿

getView

7.5%

measure

layout

draw

会到问题，我们针对会话列表的场景录制了trace，发现getView, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿

getView

7.5%

297

measure

layout

draw

会到问题，我们针对会话列表的场景录制了trace，发现getView, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿



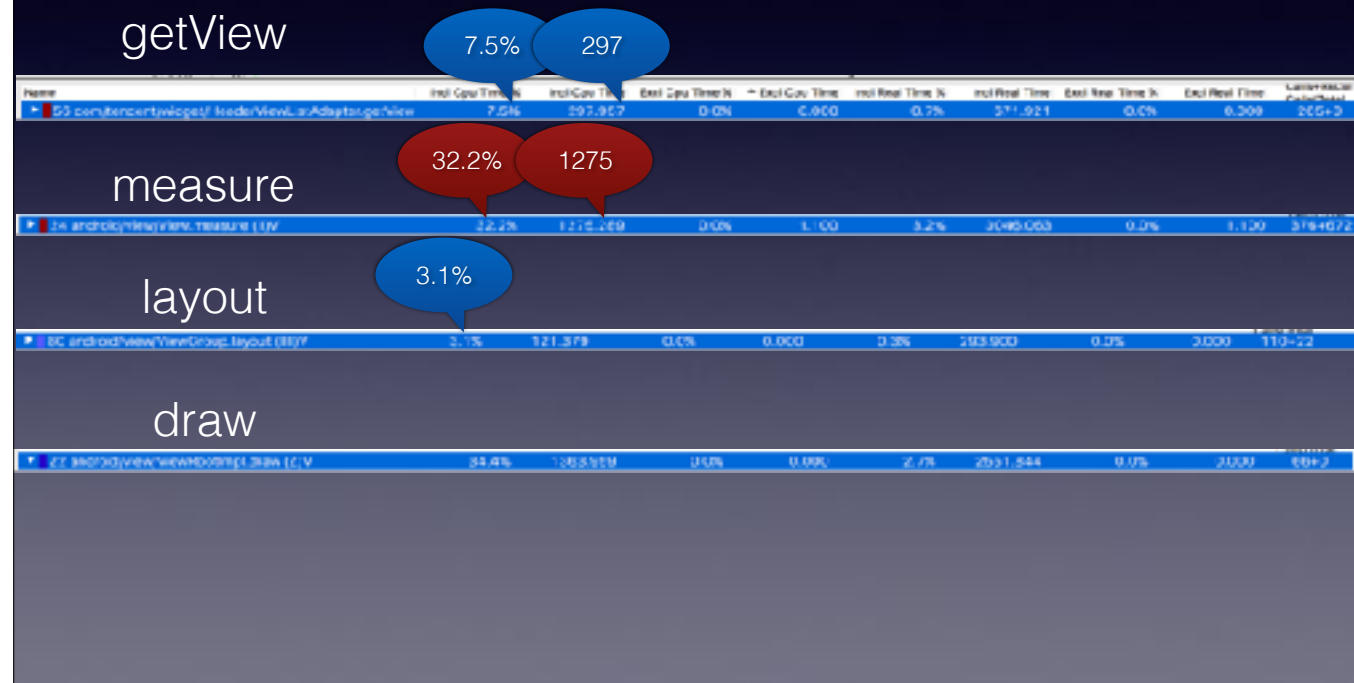
会到问题，我们针对会话列表的场景录制了trace，发现getView, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿



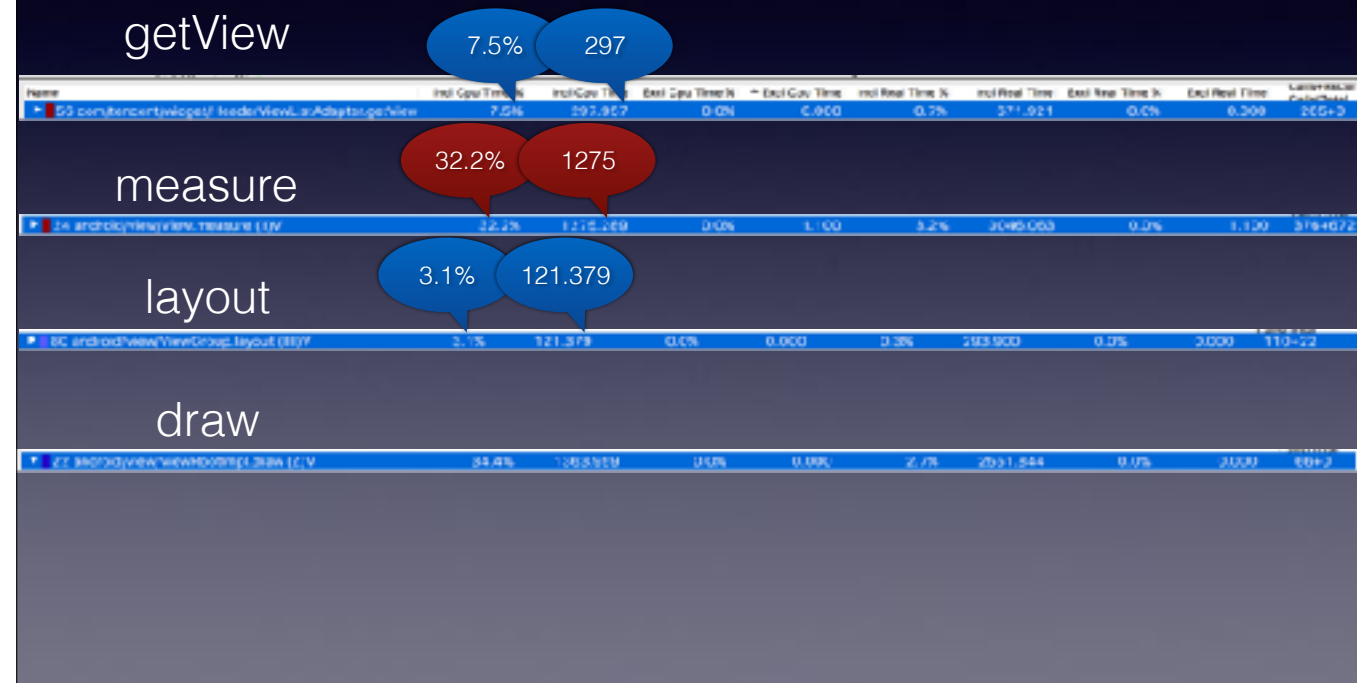
会到问题，我们针对会话列表的场景录制了trace，发现getView, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿



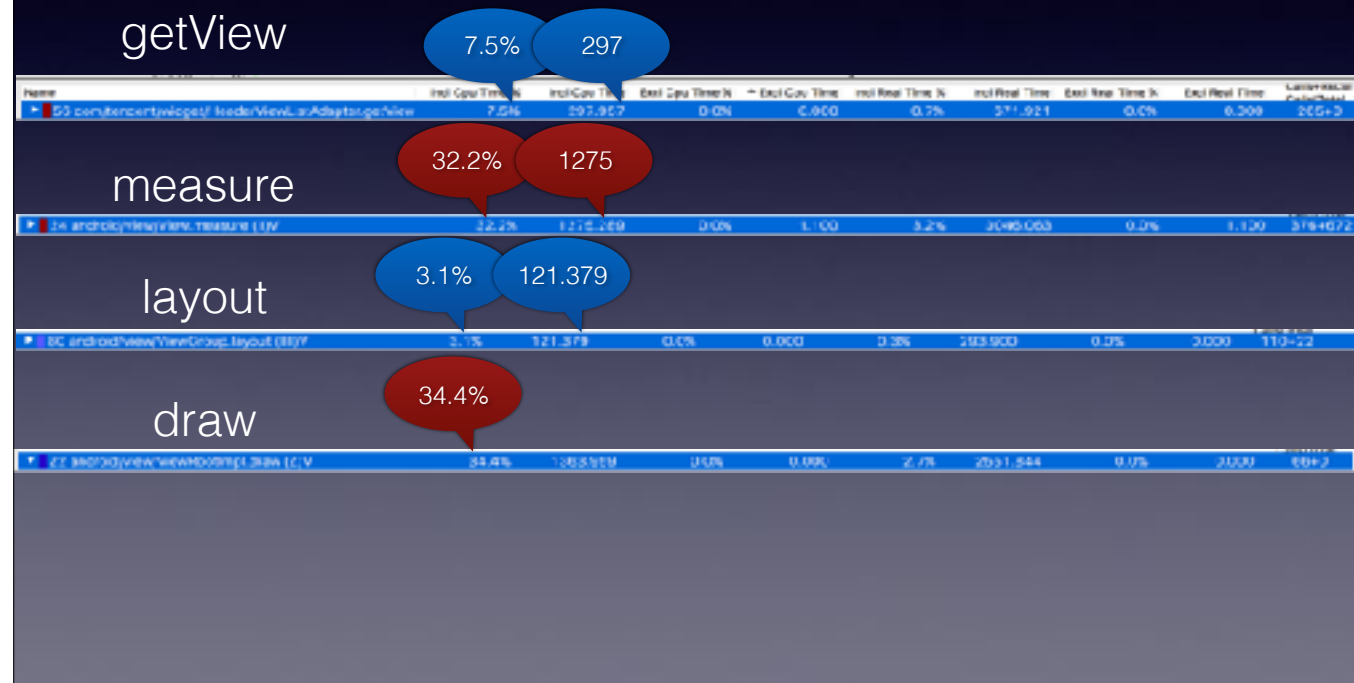
会到问题，我们针对会话列表的场景录制了trace, 发现getView, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿



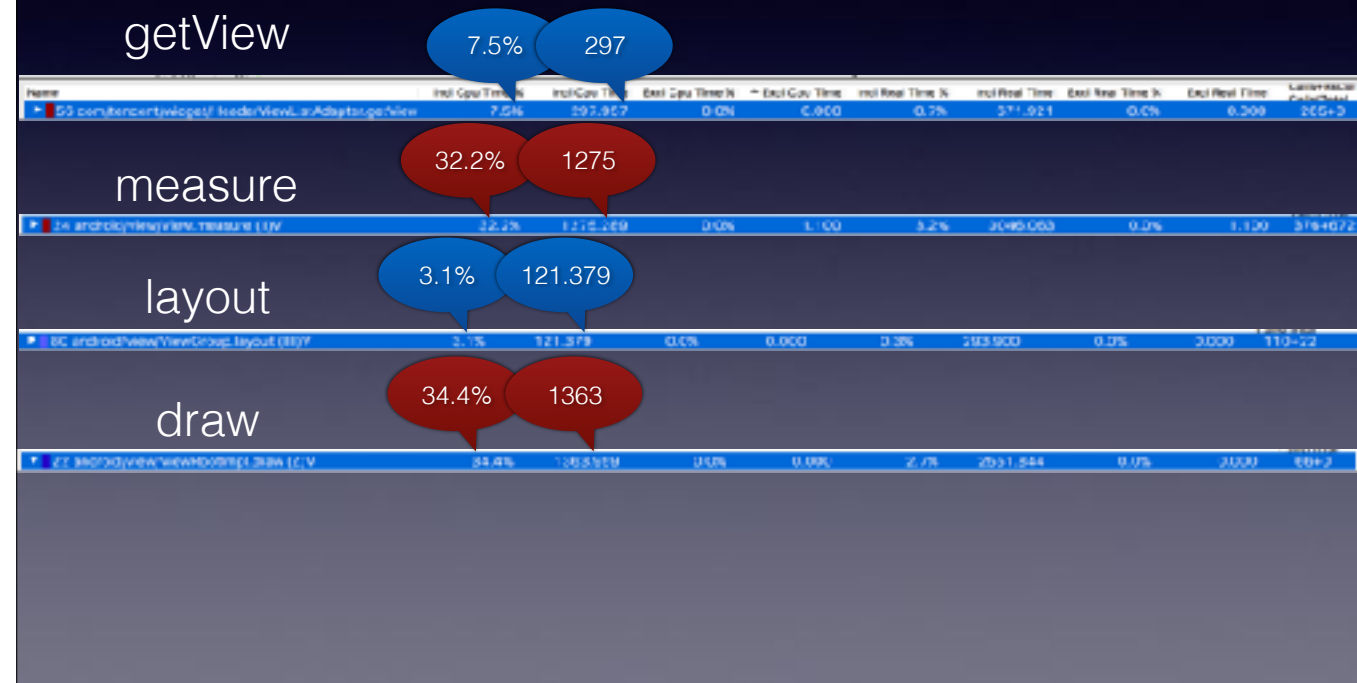
会到问题，我们针对会话列表的场景录制了trace, 发现getView, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿



会到问题，我们针对会话列表的场景录制了trace，发现getView, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿



会到问题，我们针对会话列表的场景录制了trace, 发现getView, measure, layout, draw耗时，和百分比如上。

▼ 24 android/view/View.measure (I V)	376+612	32.2%	1275.289
▶ Parents			
▼ Children			
self		0.1%	1.100
25 android/widget/RelativeLayout.onMeasure (I V)	376/376	99.2%	1265.295
535 android/view/ViewGroup.resolveRtlPropertiesIfNeeded ()Z	3/5	0.3%	3.285
304 android/util/LongSparseLongArray.put (I,J)V	2/18	0.2%	2.306
176 java/lang/System.currentTimeMillis ()J	2/40	0.2%	2.204
178 android/view/View.isLayoutModeOptical (Ljava/lang/Object)Z	1/42	0.1%	1.099
(context switch)	1/3404	0.0%	0.000
▶ Parents while recursive			
▼ Children while recursive			
26 android/widget/TextView.onMeasure (I V)	632/632	79.1%	1006.326
171 android/widget/LinearLayout.onMeasure (I V)	44/44	3.6%	45.477

进一步看measure. 我们发现两点。第一个SB, 56.4, 另外, textview measure特别多。

最后怎么解决的呢? 我先不说具体怎样解决。大家有没有发现这个思路很顺畅, 并非试一下, 是至上而下的, 借助traceview能做到。但是这里有个问题。

24	android/view/View.measure (IIV)	376+612	32.2%	1275.289
Parents				
Children				
self			0.1%	1.100
25	android/widget/RelativeLayout.onMeasure (IIV)	376/376	99.2%	1265.295
535	android/view/ViewGroup.resolveRtlPropertiesIfNeeded (IZ)	3/5	0.3%	3.285
304	android/util/LongSparseLongArray.put (JJ)V	2/18	0.2%	2.306
176	java/lang/System.currentTimeMillis ()J	2/40	0.2%	2.204
178	android/view/View.isLayoutModeOptical (Ljava/lang/Object;)Z	1/42	0.1%	1.099
	(context switch)	1/3404	0.0%	0.000
Parents while recursive				
Children while recursive				
26	android/widget/TextView.onMeasure (IIV)	632/632	79.1%	1006.326
171	android/widget/LinearLayout.onMeasure (IIV)	44/44	3.6%	45.477

79.1%

进一步看measure. 我们发现两点。第一个SB, 56.4, 另外, textview measure特别多。
最后怎么解决的呢? 我先不说具体怎样解决。大家有没有发现这个思路很顺畅, 并非试一下, 是至上而下的, 借助traceview能做到。但是这里有个问题。

24	android/view/View.measure (IIV)	376+612	32.2%	1275.289
Parents				
Children				
	self		0.1%	1.100
25	android/widget/RelativeLayout.onMeasure (IIV)	376/376	99.2%	1265.295
535	android/view/ViewGroup.resolveRtlPropertiesIfNeeded (IZ)	3/5	0.3%	3.285
304	android/util/LongSparseLongArray.put (JJ)V	2/18	0.2%	2.306
176	java/lang/System.currentTimeMillis (J)	2/40	0.2%	2.204
178	android/view/View.isLayoutModeOptical (Ljava/lang/Object)Z	1/42	0.1%	1.099
	(context switch)	1/3404	0.0%	0.000
Parents while recursive				
Children while recursive				
26	android/widget/TextView.onMeasure (IIV)	632/632	79.1%	1006.326
171	android/widget/LinearLayout.onMeasure (IIV)	44/44	3.6%	45.477

79.1%

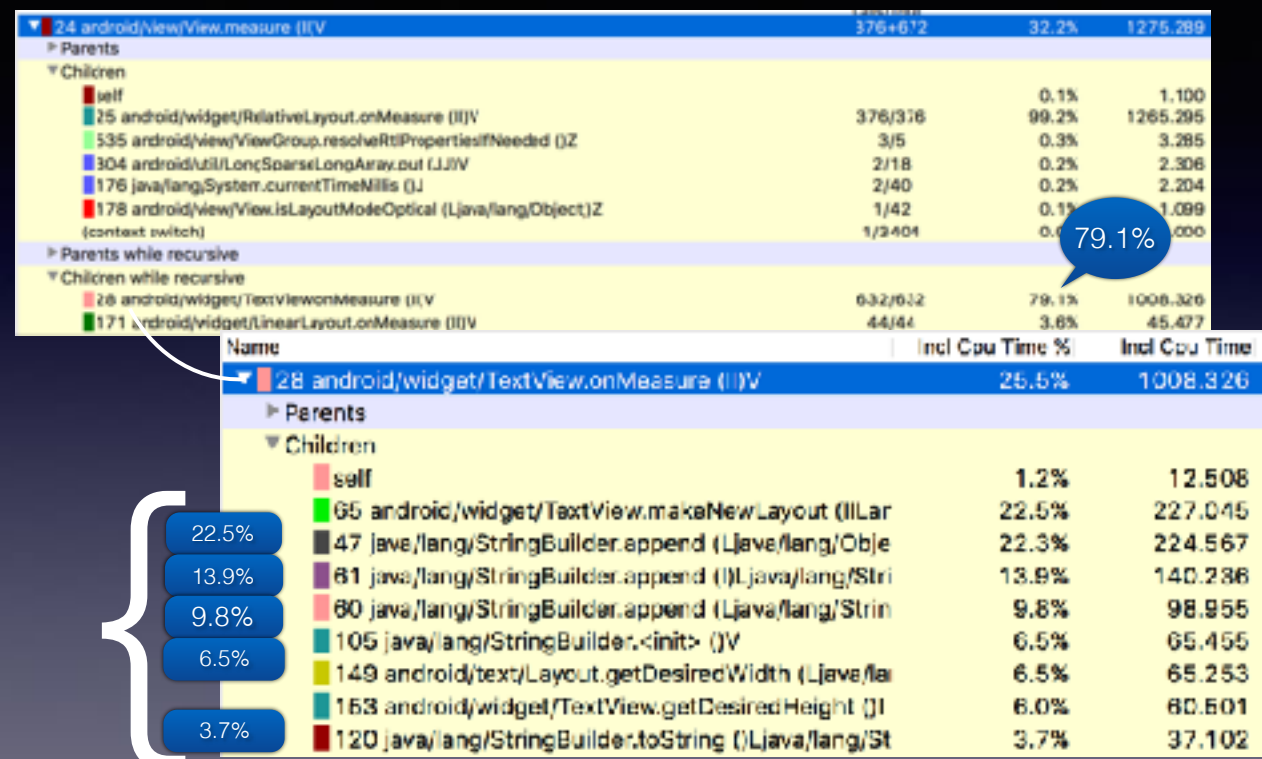
进一步看measure. 我们发现两点。第一个SB, 56.4, 另外, textview measure特别多。
最后怎么解决的呢? 我先不说具体怎样解决。大家有没有发现这个思路很顺畅, 并非试一下, 是至上而下的, 借助traceview能做到。但是这里有个问题。

24	android/view/View.measure (IIV)	376/612	32.2%	1275.289
Parents				
Children				
self			0.1%	1.100
25	android/widget/RelativeLayout.onMeasure (IIV)	376/376	99.2%	1265.295
535	android/view/ViewGroup.resolveRtlPropertiesIfNeeded (IZ)	3/5	0.3%	3.285
304	android/util/LongSparseLongArray.put (JJIV)	2/18	0.2%	2.306
176	java/lang/System.currentTimeMillis (J)	2/40	0.2%	2.204
178	android/view/View.isLayoutModeOptical (Ljava/lang/Object)Z	1/42	0.1%	1.099
	(context switch)	1/3404	0.0%	0.000
Parents while recursive				
Children while recursive				
28	android/widget/TextView.onMeasure (IIV)	632/632	79.1%	1008.326
171	android/widget/LinearLayout.onMeasure (IIV)	44/44	3.6%	45.477

Name	Incl Cpu Time %	Incl Cpu Time
28 android/widget/TextView.onMeasure (IIV)	25.5%	1008.326
Parents		
Children		
self	1.2%	12.508
65 android/widget/TextView.makeNewLayout (ILar	22.5%	227.045
47 java/lang/StringBuilder.append (Ljava/lang/Obje	22.3%	224.567
61 java/lang/StringBuilder.append (Ljava/lang/Stri	13.9%	140.238
60 java/lang/StringBuilder.append (Ljava/lang/Stri	9.8%	98.955
105 java/lang/StringBuilder.<init> (JV	6.5%	65.455
149 android/text/Layout.getDesiredWidth (Ljava/la	6.5%	65.253
163 android/widget/TextView.getDesiredHeight (J)	6.0%	60.601
120 java/lang/StringBuilder.toString ()Ljava/lang/St	3.7%	37.102

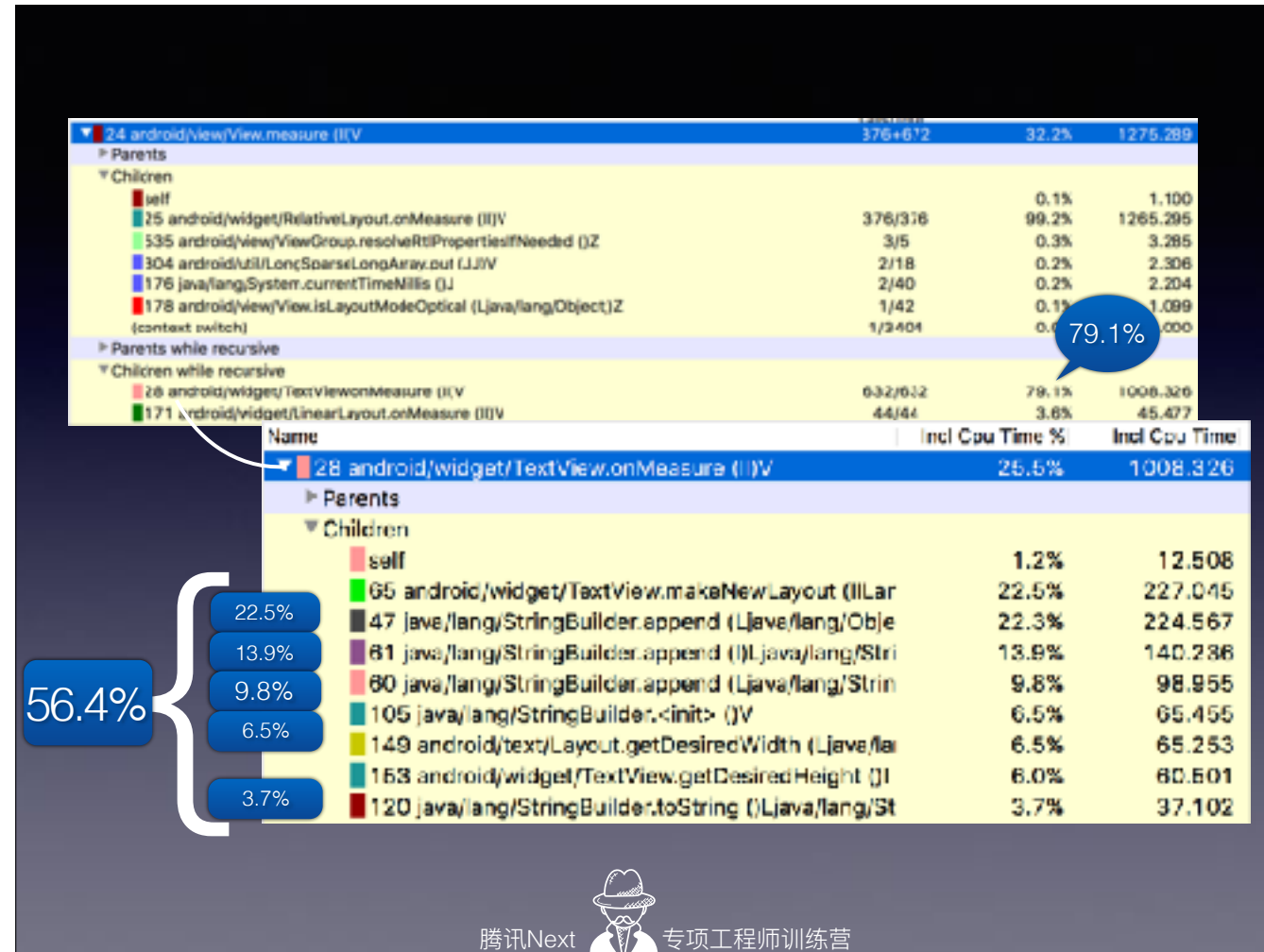
进一步看measure. 我们发现两点。第一个SB, 56.4, 另外, textview measure特别多。

最后怎么解决的呢? 我先不说具体怎样解决。大家有没有发现这个思路很顺畅, 并非试一下, 是至上而下的, 借助traceview能做到。但是这里有个问题。

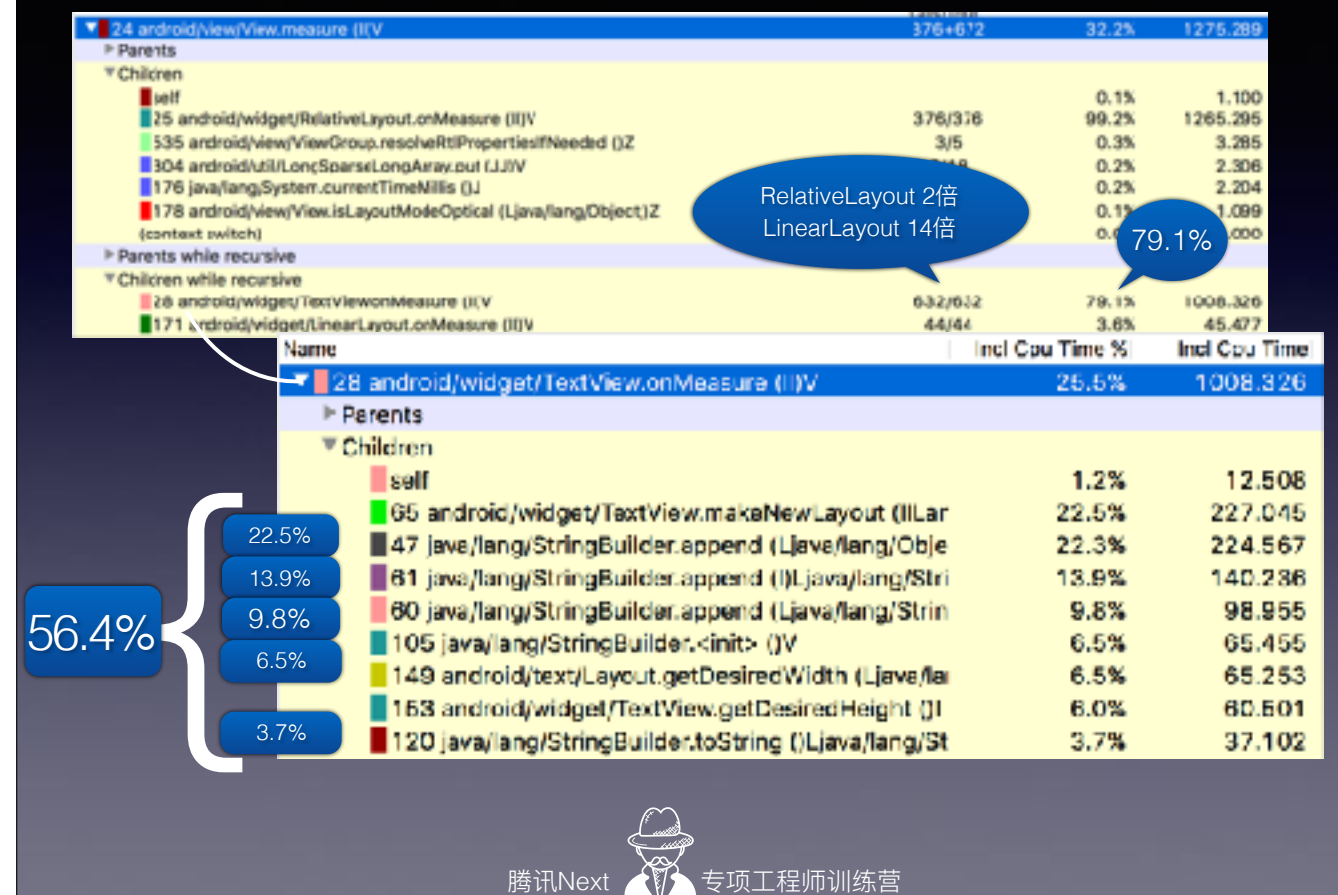


进一步看measure. 我们发现两点。第一个SB, 56.4, 另外, textview measure特别多。

最后怎么解决的呢? 我先不说具体怎样解决。大家有没有发现这个思路很顺畅, 并非试一下, 是至上而下的, 借助traceview能做到。但是这里有个问题。



进一步看measure. 我们发现两点。第一个SB, 56.4, 另外, textview measure特别多。
最后怎么解决的呢? 我先不说具体怎样解决。大家有没有发现这个思路很顺畅, 并非试一下, 是至上而下的, 借助traceview能做到。但是这里有个问题。



进一步看measure. 我们发现两点。第一个SB, 56.4, 另外, textview measure特别多。
最后怎么解决的呢? 我先不说具体怎样解决。大家有没有发现这个思路很顺畅, 并非试一下, 是至上而下的, 借助traceview能做到。但是这里有个问题。

traceview是否万能的？



这里引入另外一个案例。

长文本卡顿问题



15FPS!

Name	Incl Cpu Time %	Incl Cpu Time	Incl Real Time %	Incl Real Time
1.1 android.view.ViewRootImpl.draw (Z)V	93.3%	2384.892	14.2%	2661.165
Parent				
10 android.view.ViewRootImpl.performDraw ()V	100.0%	2384.892	100.0%	2661.165
Children				
self	0.1%	3.137	0.1%	3.147
12 android.view HardwareRenderer\$GLESRenderer.draw (Landroid	99.7%	2378.071	99.7%	2644.348

draw:93.3%, Incl Cpu Time:2384

getDisplayList -> DrawDisplayList -> SwapBuffers

Name	Incl Cpu Time %	Incl Cpu Time	Incl Real Time %	Incl Real Time
11 android.view.ViewRootImpl.draw (Z)V	100.0%	2378.071	100.0%	2644.348
Children				
self	0.3%	7.382	0.3%	7.249
13 android.view.GLES20Canvas.drawDisplayList (Landroid/view	89.5%	2128.087	87.1%	2304.192
15 android.view.View.getDisplayList ()(Landroid/view/DisplayList;	6.9%	163.854	7.0%	185.156
100 android.view HardwareRenderer\$GLES20Renderer.onPreDraw	0.2%	5.012	0.8%	21.243

DrawDisplayList: 89.5% Incl Cpu Time:2128.087

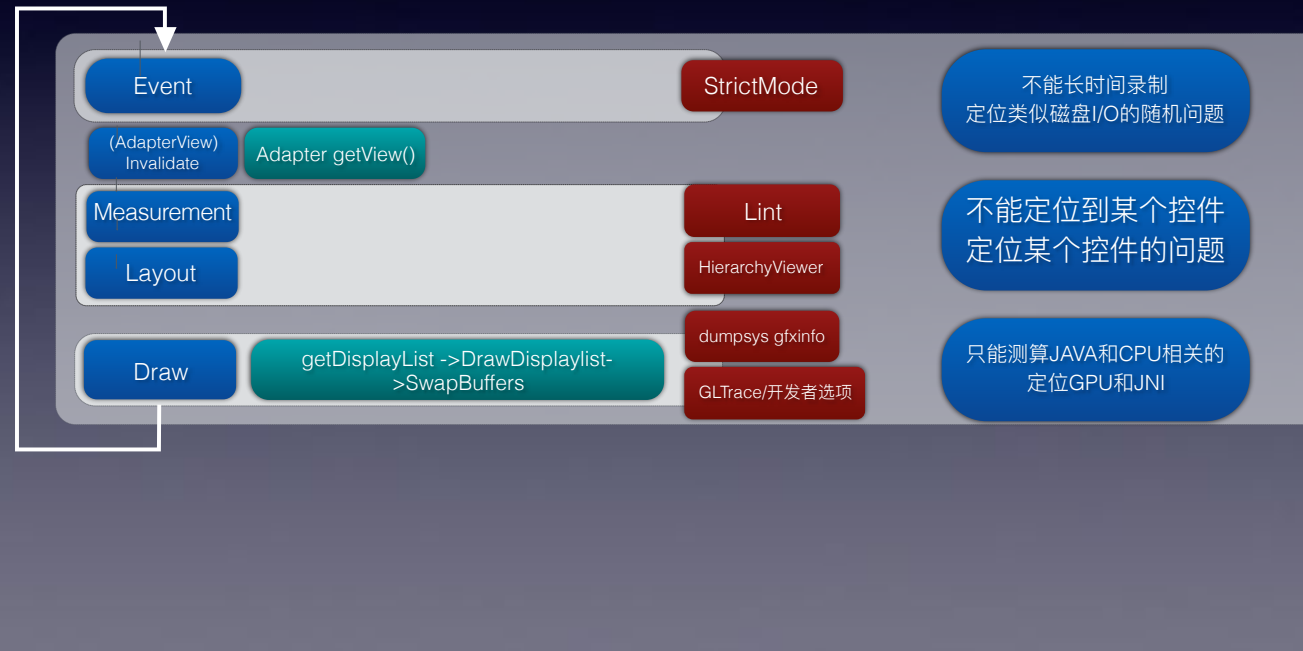
getDisplayList: 6.9% Incl Cpu Time: 163.854

93.3%的Draw, 其他东西就完全不需要看了。

Name	Incl Cpu Time %
▼ 14 android/view/GLES20Canvas.nDrawDisplayList (lll android/graphics)	83.2%
▼ Parents	
13 android/view/GLES20Canvas.drawDisplayList (Landroid/view/	100.0%
▼ Children	
self	100.0%
222 android/graphics/Rect.set (lll)V	0.0%

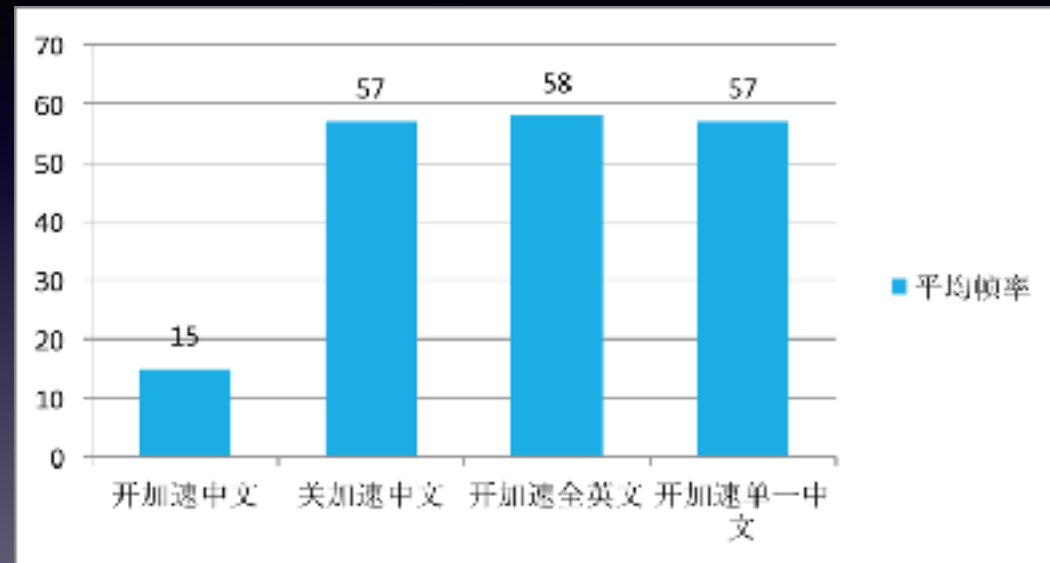
跟不下去了，悲剧
起码定位了Draw

TraceView并非万能



不能解决的三个问题

1. 长时间： traceview不能录取太长时间
2. 定位某个控件： 某个通用控件，measure或者layout的耗时长
3. 仅限CPU,JAVA，NDK（C，C++）与GPU渲染相关的，表示无力



因为4.1.2的缘故，opengl trace不能用，gfxinfo在traceview已经定位到draw的基础上也是没卵用。那怎么办，在4.1.2上面。幸亏我们有经验，我们做了测试验证了一下，特别是第四点，只有一个中文就很快，因为每个中文字都需要绘制，所以基本可以确认是中文给的缓存太少。除了这个还有一个绝招，堆栈上报

卡顿堆栈上报

- 当主线程耗时超过50ms，开始截取堆栈并上报

节点	辅助功能	版本占%	基准版本占%	版本占比变化	耗时(秒)
com.tencent.wic	展开红色	14.606	11.032	32.4 %	147
com.tencent	展开红色	9.637	7.150	34.8 %	97
com.tenc	展开红色	9.637	7.140	35.0 %	97
com.tencent	展开红色	2.422	1.384	75.0 %	24
com.tencent	展开红色	0.780	0.215	262.1 %	8
com.tencent.mobil	展开红色	0.436	0.128	239.8 %	4
com.tencent	展开红色	0.298	0.109	174.4 %	3
com.tencent	展开红色	0.177	0.105	69.2 %	2
com.tencent	展开红色	0.108	0.058	86.7 %	1
com.tencent	展开红色	0.070	0.000	19645.1 %	1
mqq.app.AppAc	展开红色	13.609	9.663	40.8 %	137

至下而上

判断文件是否存在

反向聚合后的卡顿堆栈中的排行榜

从这个排行版中，我们得到了量化的数据外，对于当时的我们来说，是得到了一个反直觉的判断

判断文件是否存在

反向聚合后的卡顿堆栈中的排行榜

	求和次	求和	平均耗时
java.lang.ClassLoader.loadCla	681487	1090866913	1600.7
java.io.File.exists	19158	31499380	1644.2
java.lang.StringBuilder.<init>	60561	66418071	1096.7
java.lang.StringBuilder.append	58918	60714907	
java.lang.System.gc	45849	54573433	
java.lang.Object.wait	35202	48079552	
java.lang.Class.getDeclaredFie	28764	47433660	
java.lang.Throwable.printStackTrace	50113	39630191	

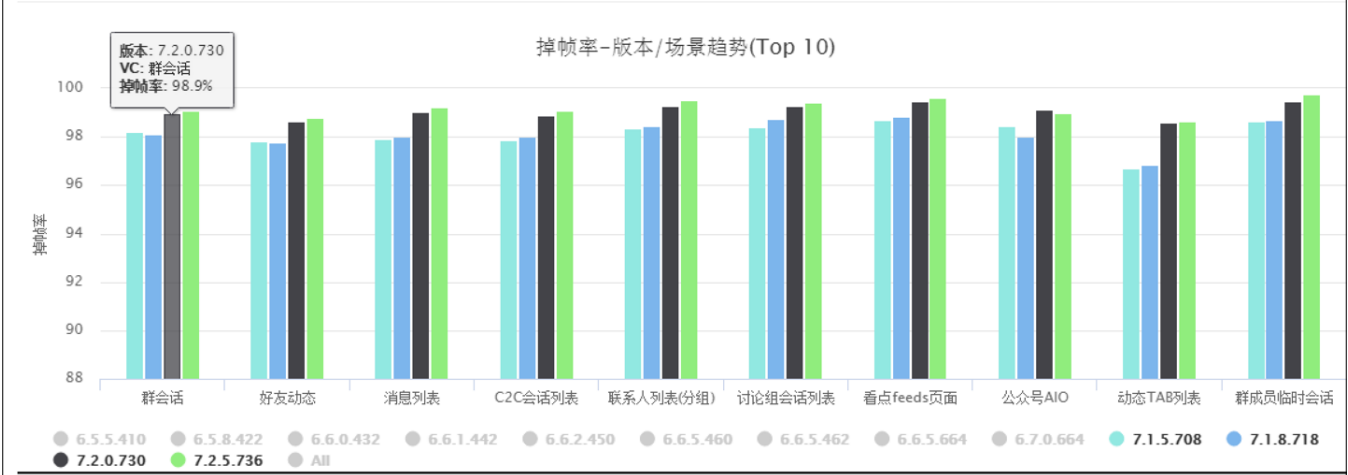
从这个排行版中，我们得到了量化的数据外，对于当时的我们来说，是得到了一个反直觉的判断

至下而上:FileExist影响范围

[illegible]

成效: up1%

2. 数据来自 SNGAPM 掉帧流畅度上报：从大盘卡顿流畅度数据来看，720 比 718 流畅度数据指标提升了近 1%。



小结

- TRACEVIEW, SYSTRACE, 堆栈上报等等
- 至上而下, 使用合适的工具做合适的事情
- 至下而上, 发现以点打面的优化点 (长尾问题)



“一个小小的错误可以预防重重的坠落。”



腾讯Next

专项工程师训练营

到了解决。也是给大家一句名言。为什么这样说呢？

解决

会话列表的卡顿问题

减少measure次数
& measure的耗时

长文本卡顿问题

```
setLayerType(View.LAYER  
_TYPE_SOFTWARE, null);
```

问题

解决思路



1. 重写TEXTVIEW

2. 回顾下，是中文缓存问题，如果这个变成一张图，是不是就回避了问题。

这样的解决够吗？

更进一步的思考

会话列表的卡顿问题

减少measure次数
& measure的耗时

问题

解决思路



输出了组件，让能力可以快速复制外。我们还发现StringBuilder这个货，可能带来的性能风险，包括它的GC和扩容，关联到了日志，并开始进行优化，虽然最后是开发同学也想到并落地修改了，但至少证明方向没有错。

更进一步的思考

会话列表的卡顿问题

减少measure次数
& measure的耗时

输出组件：
SingleLineTextView

问题

解决思路



输出了组件，让能力可以快速复制外。我们还发现StringBuilder这个货，可能带来的性能风险，包括它的GC和扩容，关联到了日志，并开始进行优化，虽然最后是开发同学也想到并落地修改了，但至少证明方向没有错。

更进一步的思考

会话列表的卡顿问题

减少measure次数
& measure的耗时

输出组件：
SingleLineTextView

StringBuilder优化策略

问题

解决思路



输出了组件，让能力可以快速复制外。我们还发现StringBuilder这个货，可能带来的性能风险，包括它的GC和扩容，关联到了日志，并开始进行优化，虽然最后是开发同学也想到并落地修改了，但至少证明方向没有错。

更进一步的思考

会话列表的卡顿问题

减少measure次数
& measure的耗时

输出组件：
SingleLineTextView

StringBuilder优化策略

问题

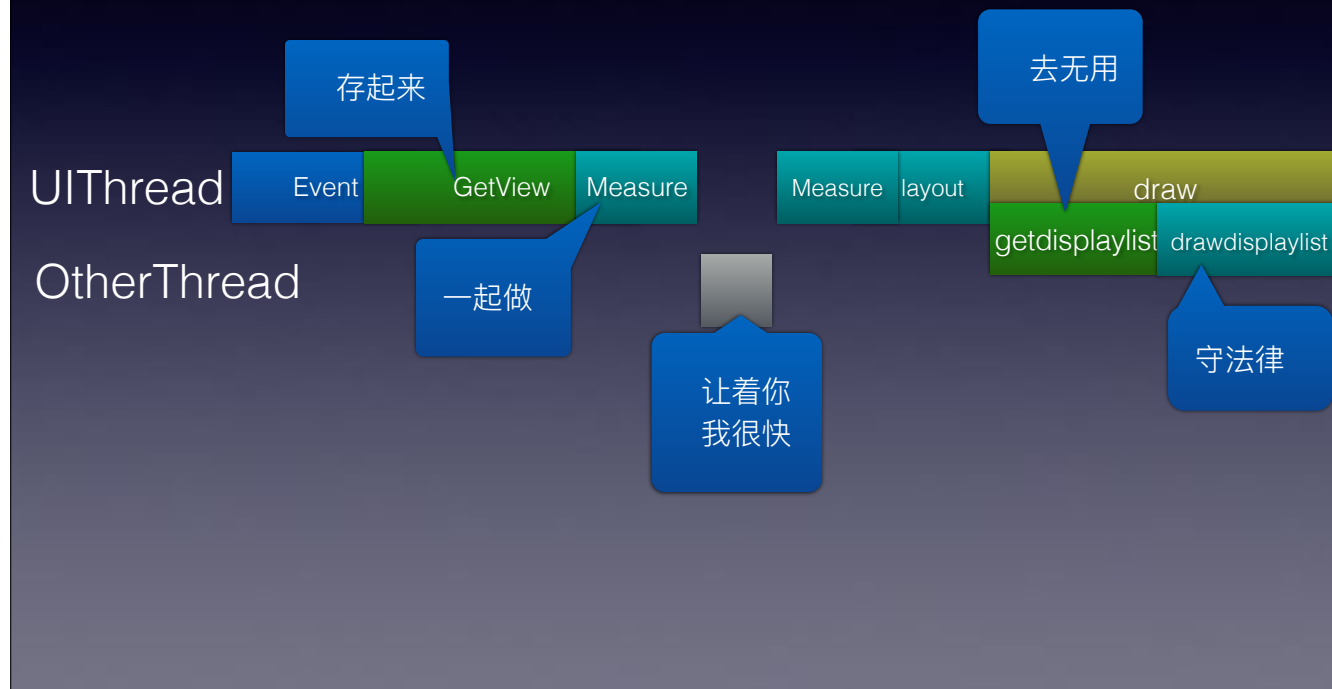
解决思路

以点打面



输出了组件，让能力可以快速复制外。我们还发现StringBuilder这个货，可能带来的性能风险，包括它的GC和扩容，关联到了日志，并开始进行优化，虽然最后是开发同学也想到并落地修改了，但至少证明方向没有错。

解决策略



5点:

存起来: ViewHolder

去无用: 去掉无用的, 例如overdraw的, 除了去掉多余的background, 针对自定义的复杂的view还可以canvas.cliprect, canvas.quickreject

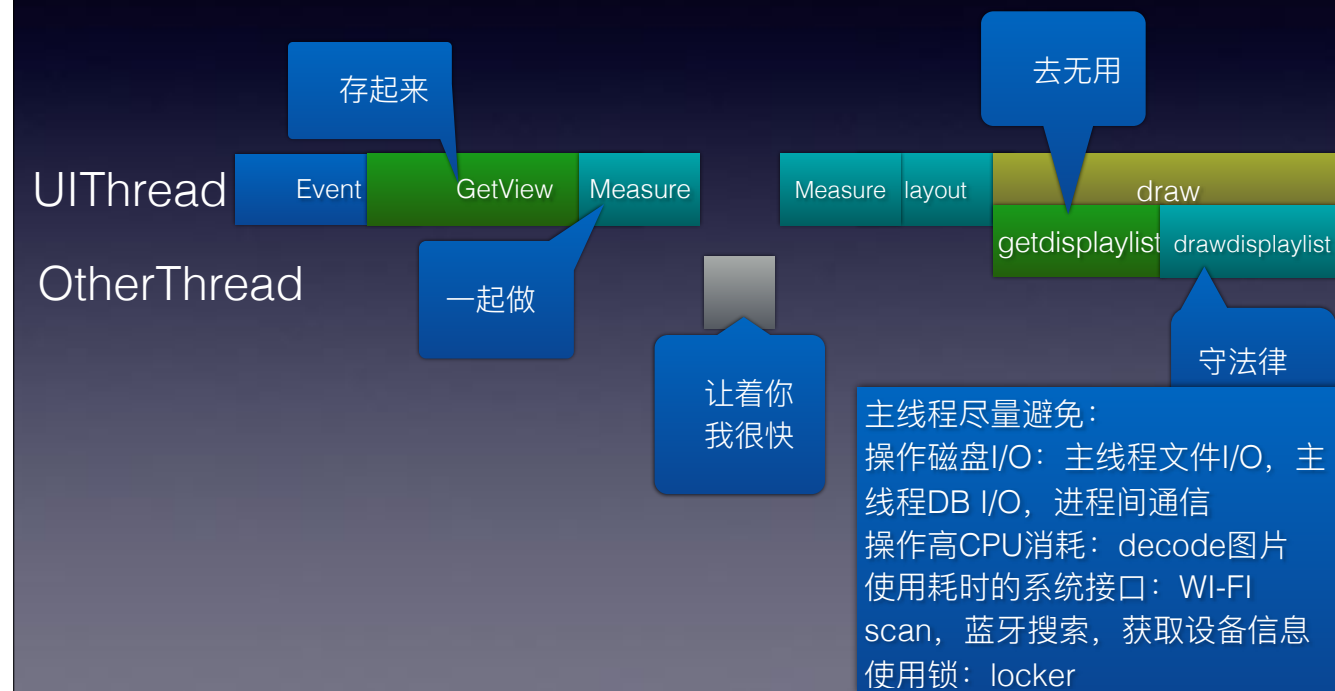
守法律: 尽量不要用 View alpha, 禁止调用saveLayer()

一起做: 利用singleTextView

我很快: 如StringBuilder, 使用delete代替new, 如主线程I/O避免, 使用ZipFile方法

让着你: CPU的优化策略中, 如降低线程优先级, Handler, 异步线程

解决策略



5点：

存起来： ViewHolder

去无用: 去掉无用的，例如overdraw的，除了去掉多余的background, 针对自定义的复杂的view还可以`canvas.cliprect`， `canvas.quickreject`

守法律： 尽量不要用 View alpha， 禁止调用`saveLayer()`

一起做： 利用`singeTextView`

我很快： 如`StringBuilder`， 使用`delete`代替`new`， 如主线程I/O避免， 使用`ZipFile`方法

让着你： CPU的优化策略中， 如降低线程优先级， Handler， 异步线程

度量：度量价值



“If You Can't Measure It, You Can't Manage It”

彼得·德鲁克



刚才发现手段中哪个方案适合度量

- 开发者选项(gfxinfo)
- SurfaceFlinger
- 日志里面的SkipFrame
- Looper监控
- DoFrame监控
- 用户反馈Keyword统计



刚才发现手段中哪个方案适合度量

- 开发者选项(gfxinfo)
- SurfaceFlinger
- 日志里面的SkipFrame
- Looper监控
- DoFrame监控
- 用户反馈Keyword统计

基于DoFrame的流畅度度量

版本	标准值	所有	群会话	好友动态	消息列表	C2C会话列表	联系人列表(分组)	讨论组会话列表	合计
7.1.0.692	98.0%	98.4%(7817K)	98.4%(2414K)	98.3%(2359K)	98.4%(1037K)	98.2%(1062K)	98.8%(339K)	98.6%(103K)	98.3%
7.0.0.676	98.0%	97.8%(814K)	97.8%(208K)	97.7%(183K)	97.2%(91K)	97.3%(77K)	97.8%(28K)	98.0%(7K)	97.3%

节点	辅助功能	版本占%	基准版本占%	版本占比变化	耗时(秒)
com.tencent.wic	展开红色	14.606	11.032	32.4 %	147
com.tencent	展开红色	9.637	7.150	34.8 %	97
com.tenc	展开红色	9.637	7.140	35.0 %	97
com.tencent	展开红色	2.422	1.384	75.0 %	24
com.tencent	展开红色	0.780	0.215	262.1 %	8
com.tencent.mobil	展开红色	0.436	0.128	239.8 %	4
com.tencent	展开红色	0.298	0.109	174.4 %	3
com.tencent	展开红色	0.177	0.105	69.2 %	2
com.tencent	展开红色	0.108	0.058	86.7 %	1
com.tencent	展开红色	0.070	0.000	19645.1 %	1

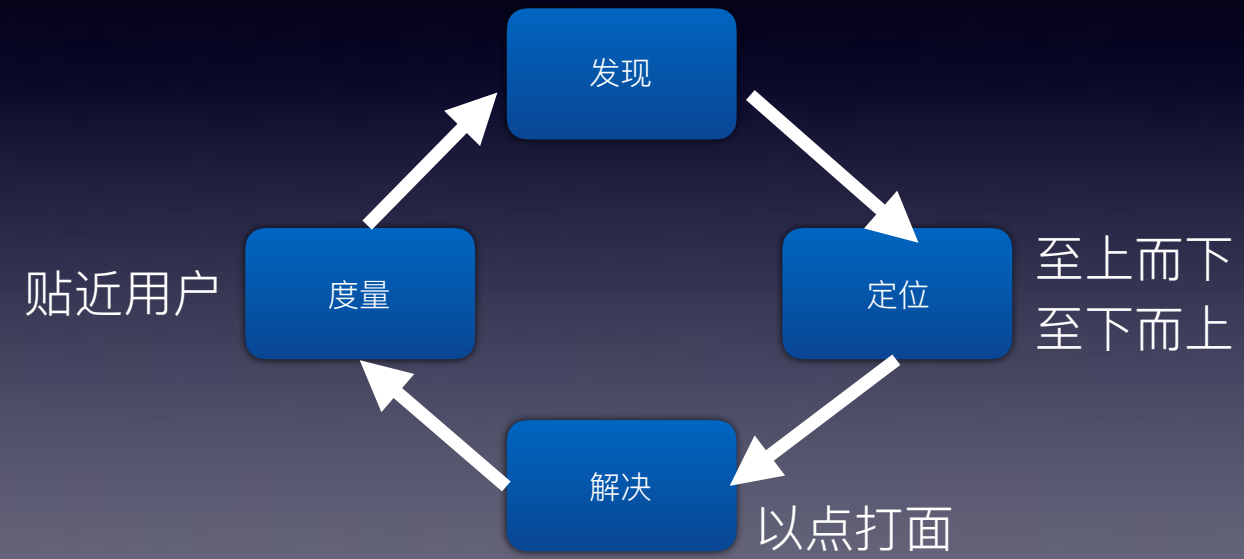
基于DoFrame的流畅度度量

版本	标准值	所有	群会话	好友动态	消息列表	C2C会话列表	联系人列表(分组)	讨论组会话列表	合计
7.1.0.692	98.0%	98.4%(7817K)	98.4%(2414K)	98.3%(2359K)	98.4%(1037K)	98.2%(1062K)	98.8%(339K)	98.6%(103K)	98.3%
7.0.0.676	98.0%	97.8%(814K)	97.8%(208K)	97.7%(183K)	97.2%(91K)	97.3%(77K)	97.8%(28K)	98.0%(7K)	97.3%

节点	辅助功能	版本占%	基准版本占%	版本占比变化	耗时(秒)
com.tencent.wic	展开红色	14.606	11.032	32.4 %	147
com.tencent	展开红色	9.637	7.150	34.8 %	97
com.tenc	展开红色	9.637	7.140	35.0 %	97
com.tencent	展开红色	2.422	1.384	75.0 %	24
com.tencent	展开红色	0.780	0.215	262.1 %	8
com.tencent.mobil	展开红色	0.436	0.128	239.8 %	4
com.tencent	展开红色	0.298	0.109	174.4 %	3
com.tencent	展开红色	0.177	0.105	69.2 %	2
com.tencent	展开红色	0.108	0.058	86.7 %	1
com.tencent	展开红色	0.070	0.000	10645.1 %	1

小结

完整且量化



Thank you

相关链接

[渲染机制](#)