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	≦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

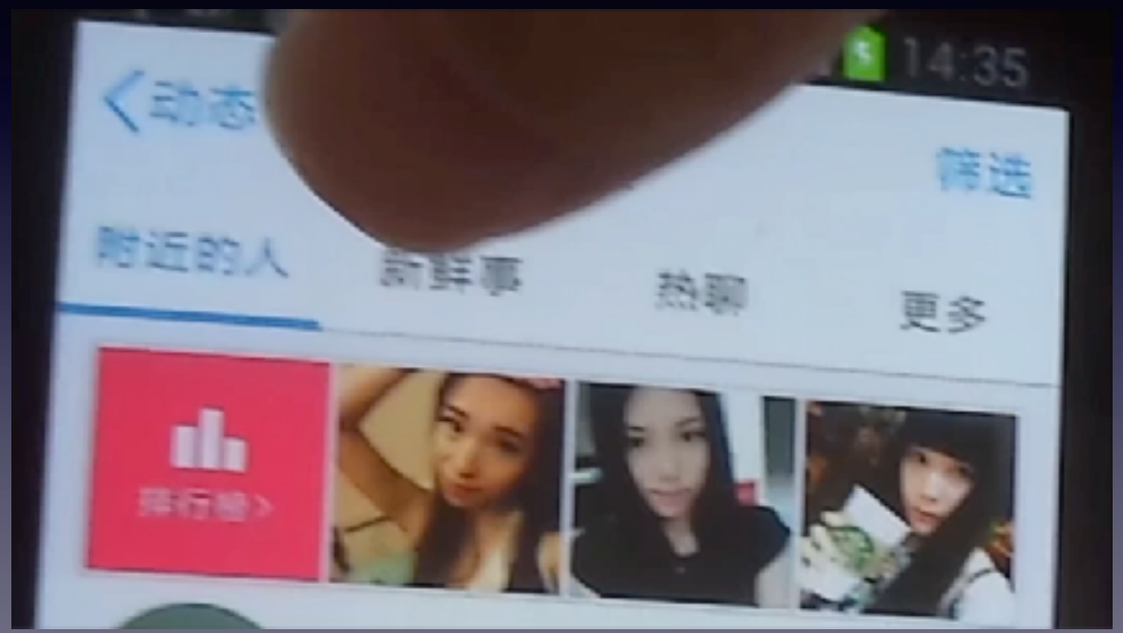
流畅度

来自用户的吐槽：流畅度

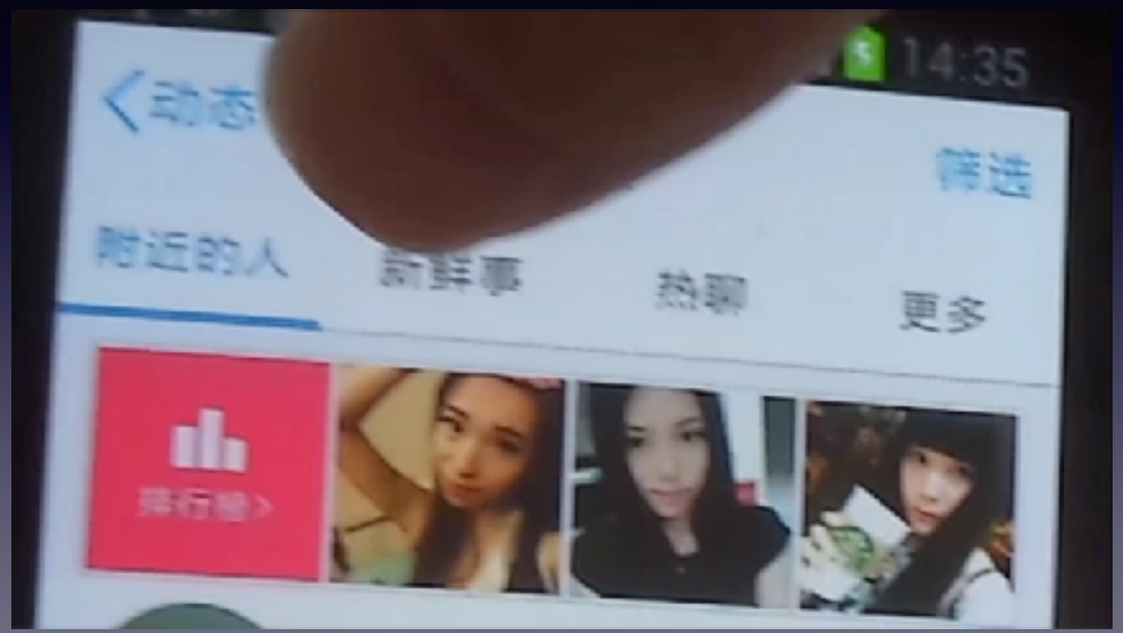
- 很卡
- 卡顿
- 卡死
- ANR

APP版本	OS	厂商	标题	内容	
7.7.8.421	11_4_1	iphone8,1	"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-time:2018年09月04日12:57	
7.8.9.450	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"附近"页面卡-time:2018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	
7.8.3.466	10_3_1	iphone7,2	问题与建议	我被骗了，希望尽快把银行卡	

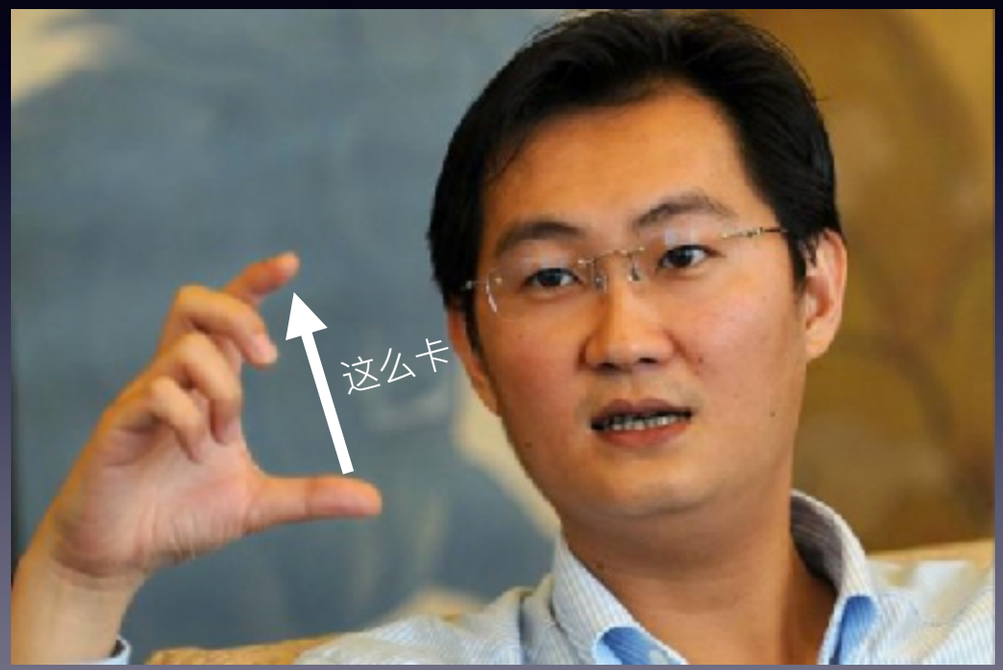
感受一下卡顿



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

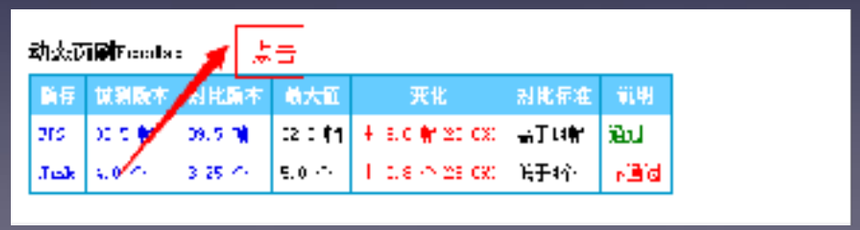
- 仅仅适用于硬件加速
- Draw + Process + Execute

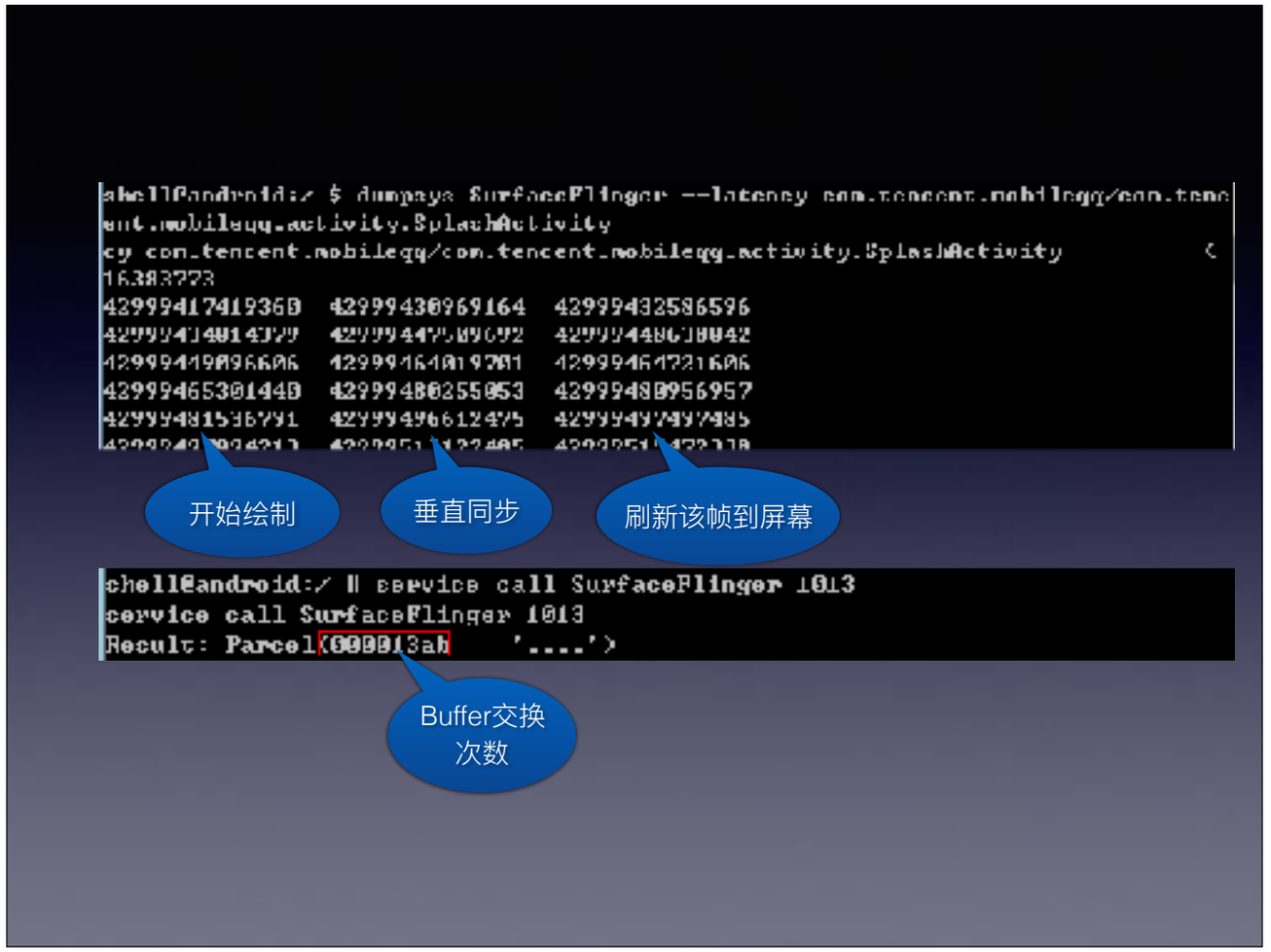
gfxinfo

- 仅仅适用于硬件加速
- Draw + Process + Exe

SurfaceFlinger

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





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 >= 4.1
- debug.choreographer.skipwarning > 30 = 480ms

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

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。

Looper：

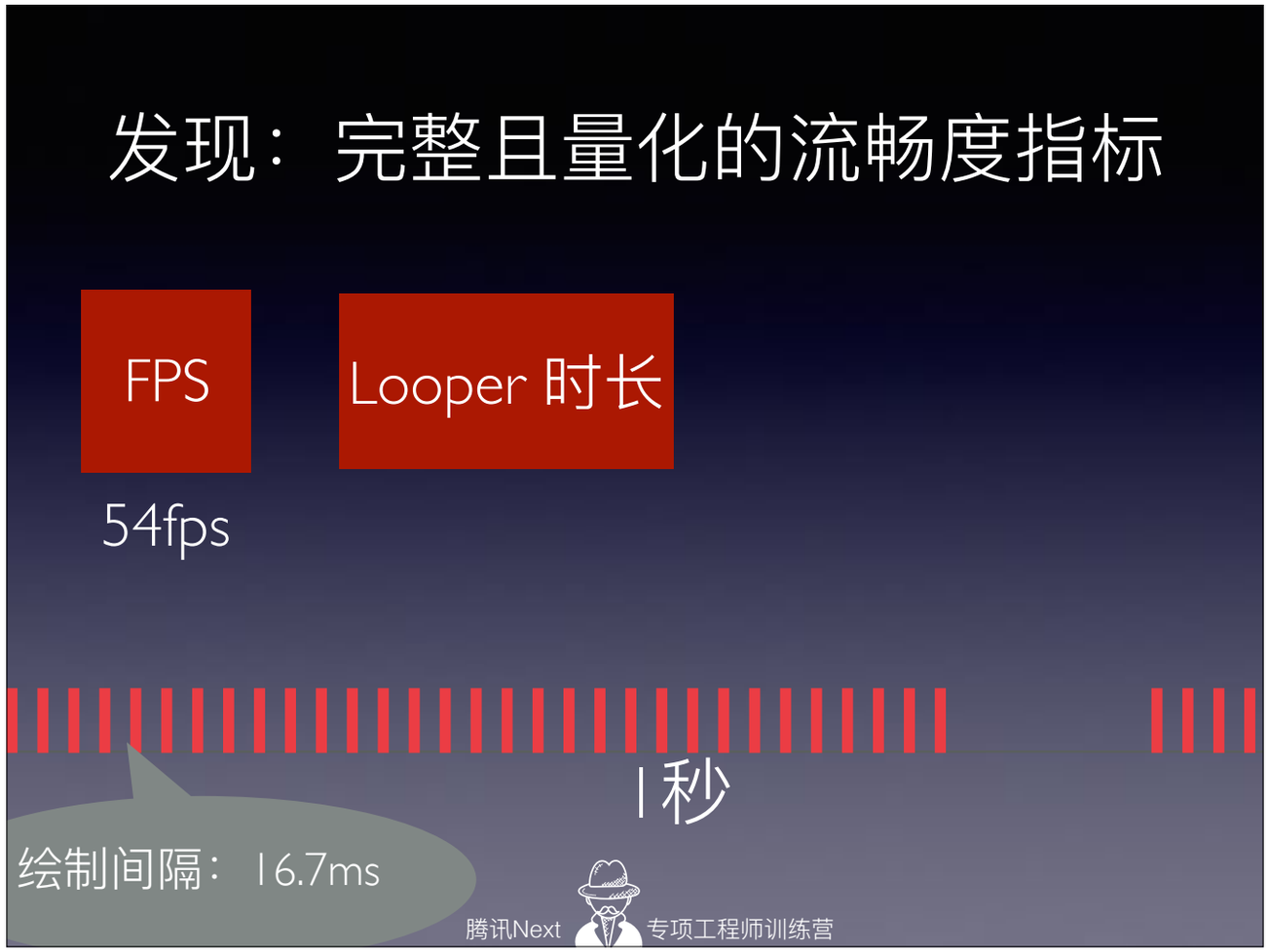
Android主线程更新UI。如果界面1秒钟刷新少于60次，即FPS小于60，用户就会产生卡顿感觉。简单来说，Android使用消息机制进行UI更新，UI线程有个Looper，在其loop方法中会不断取出message，调用其绑定的Handler在UI线程执行。如果在handler的dispatchMesaage方法里有耗时操作，就会发生卡顿。

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。

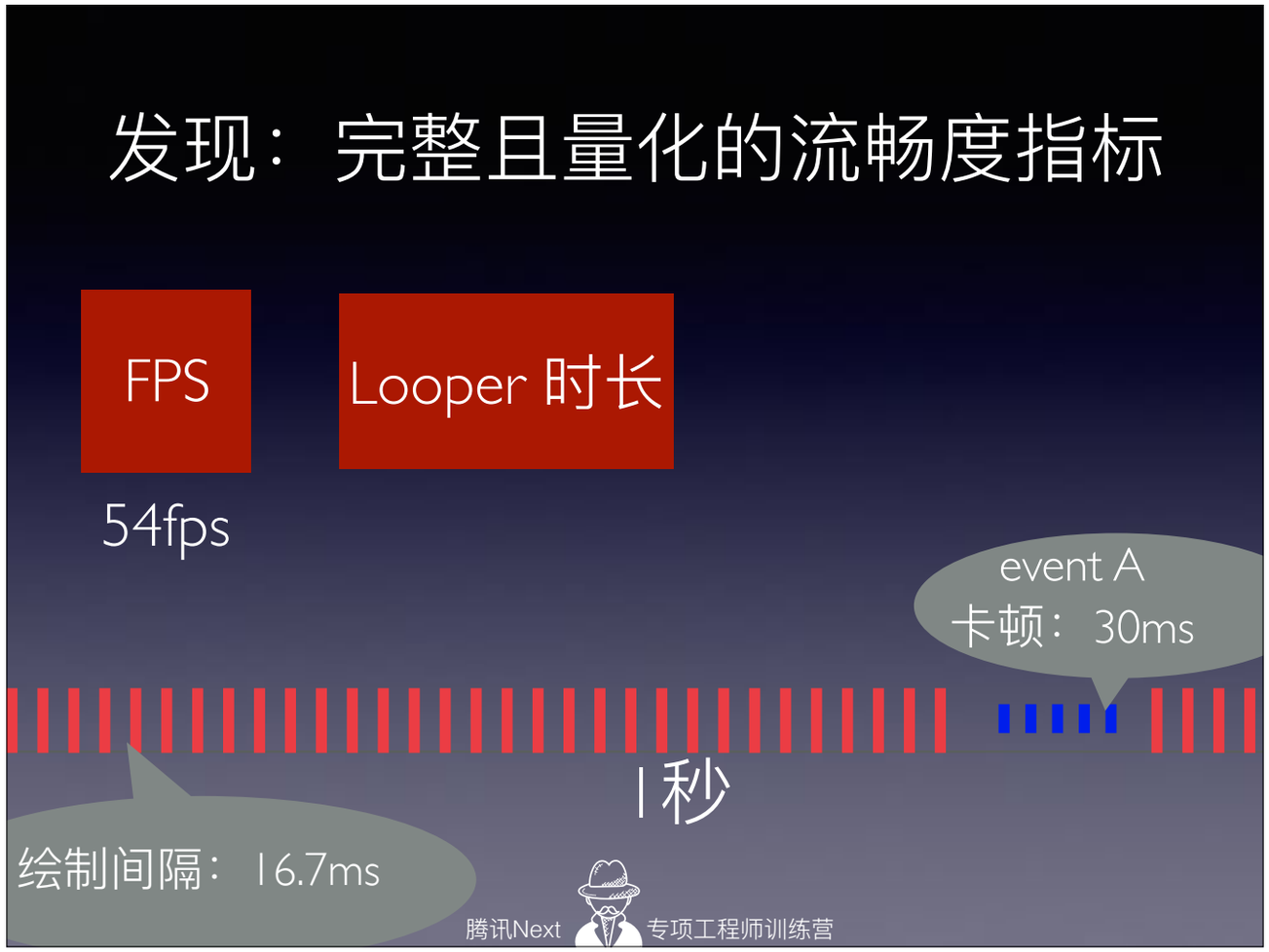
Looper：

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。



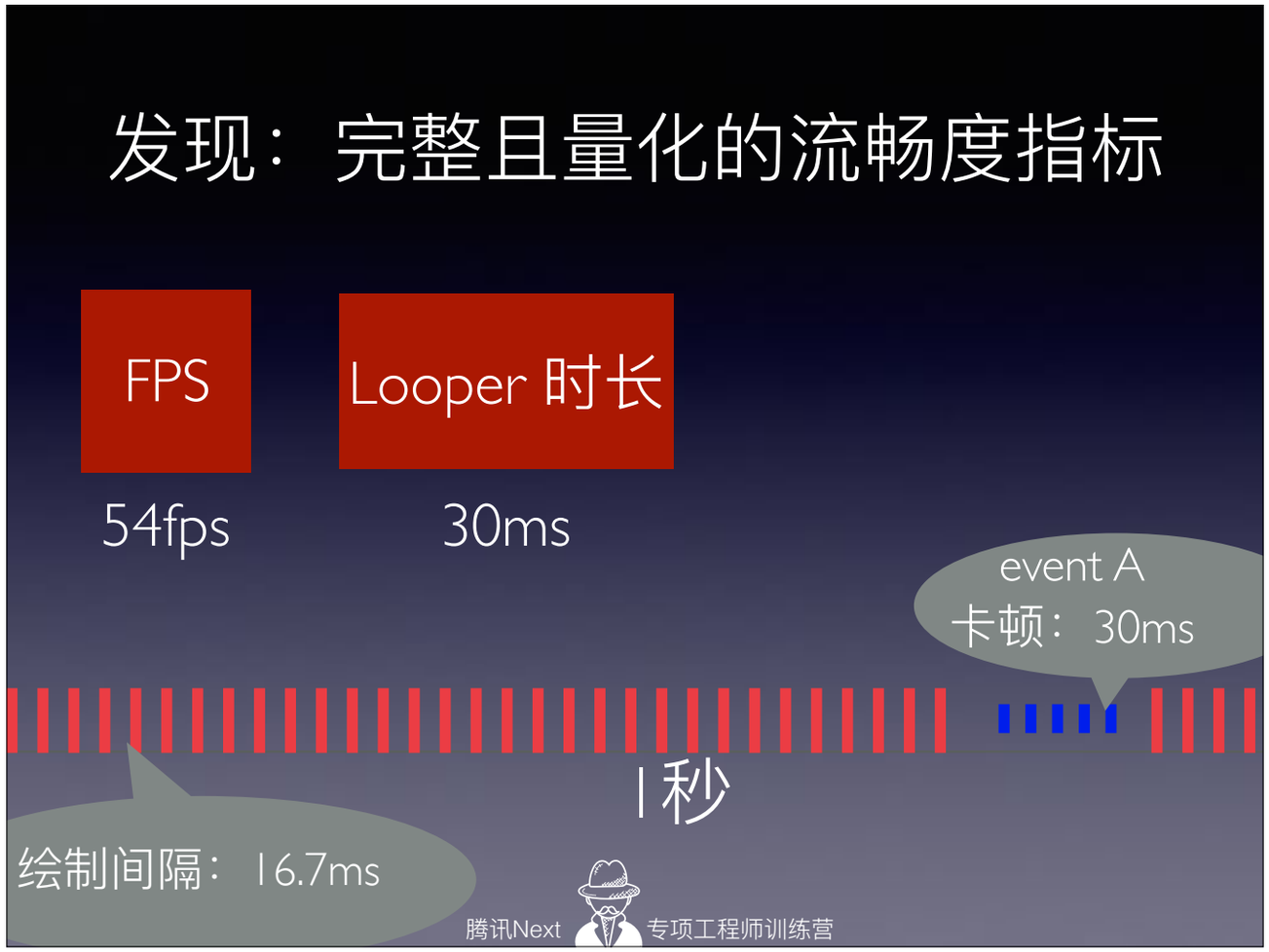
Looper：

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。



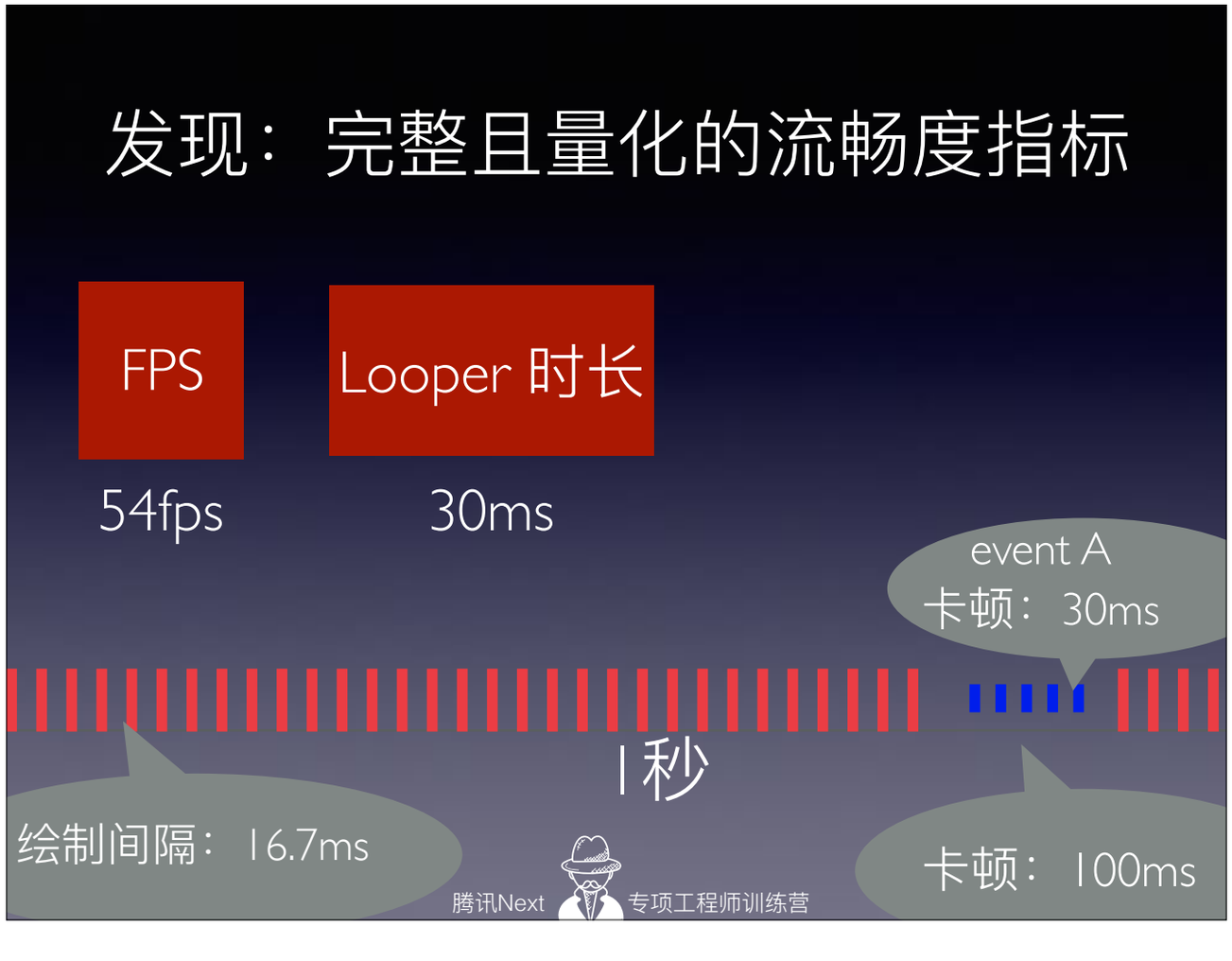
Looper：

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。



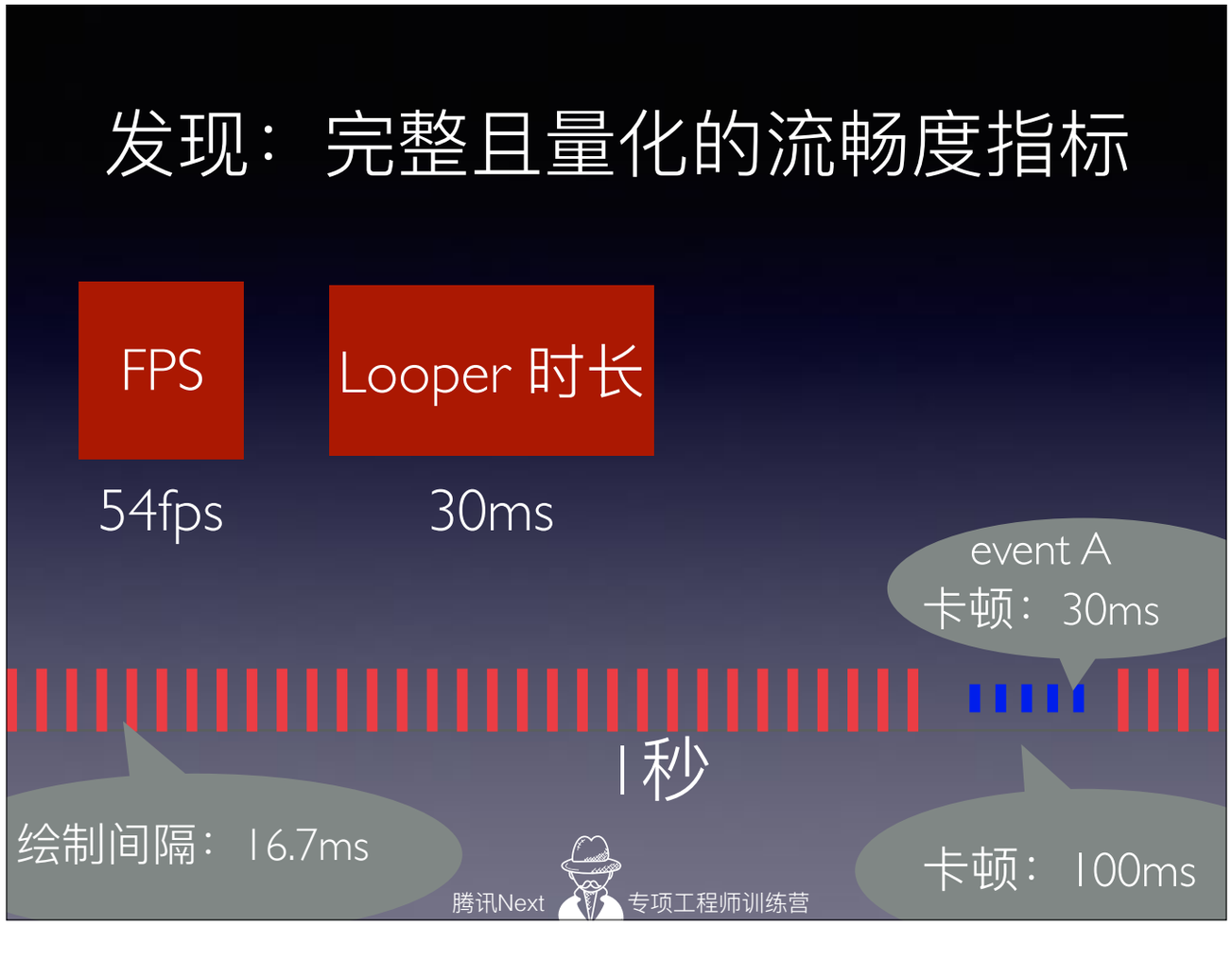
Looper：

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。



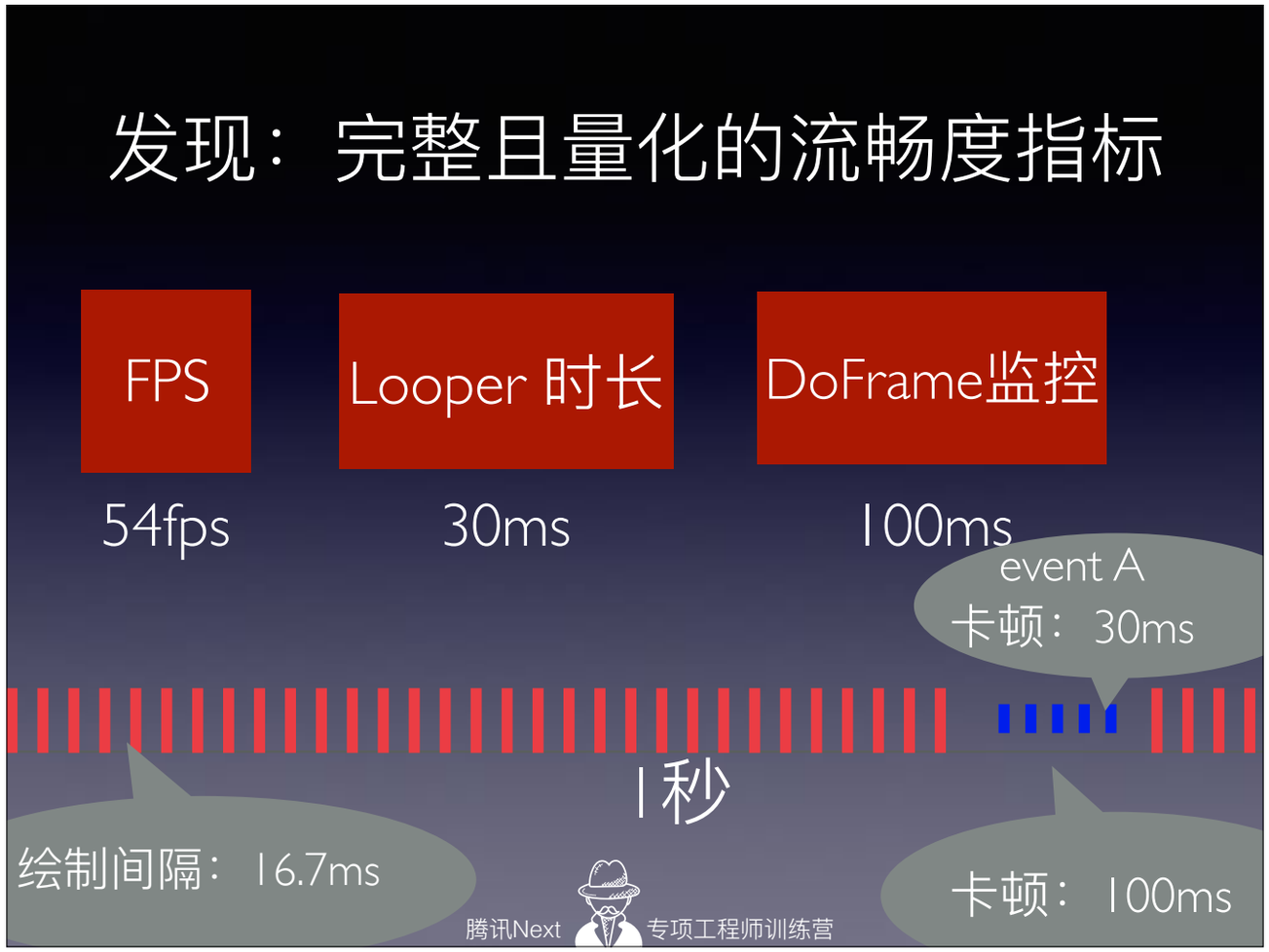
Looper：

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。



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。



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

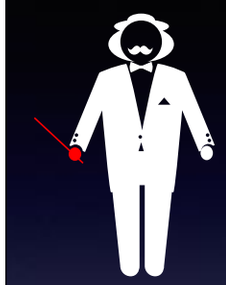
- 磁盘：I/O Wait　（TOP，hook）
- 内存：GC for Alloc　（GC日志，alloc tracer）
- CPU：CPU jiffer　（proc/pid/stat）

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



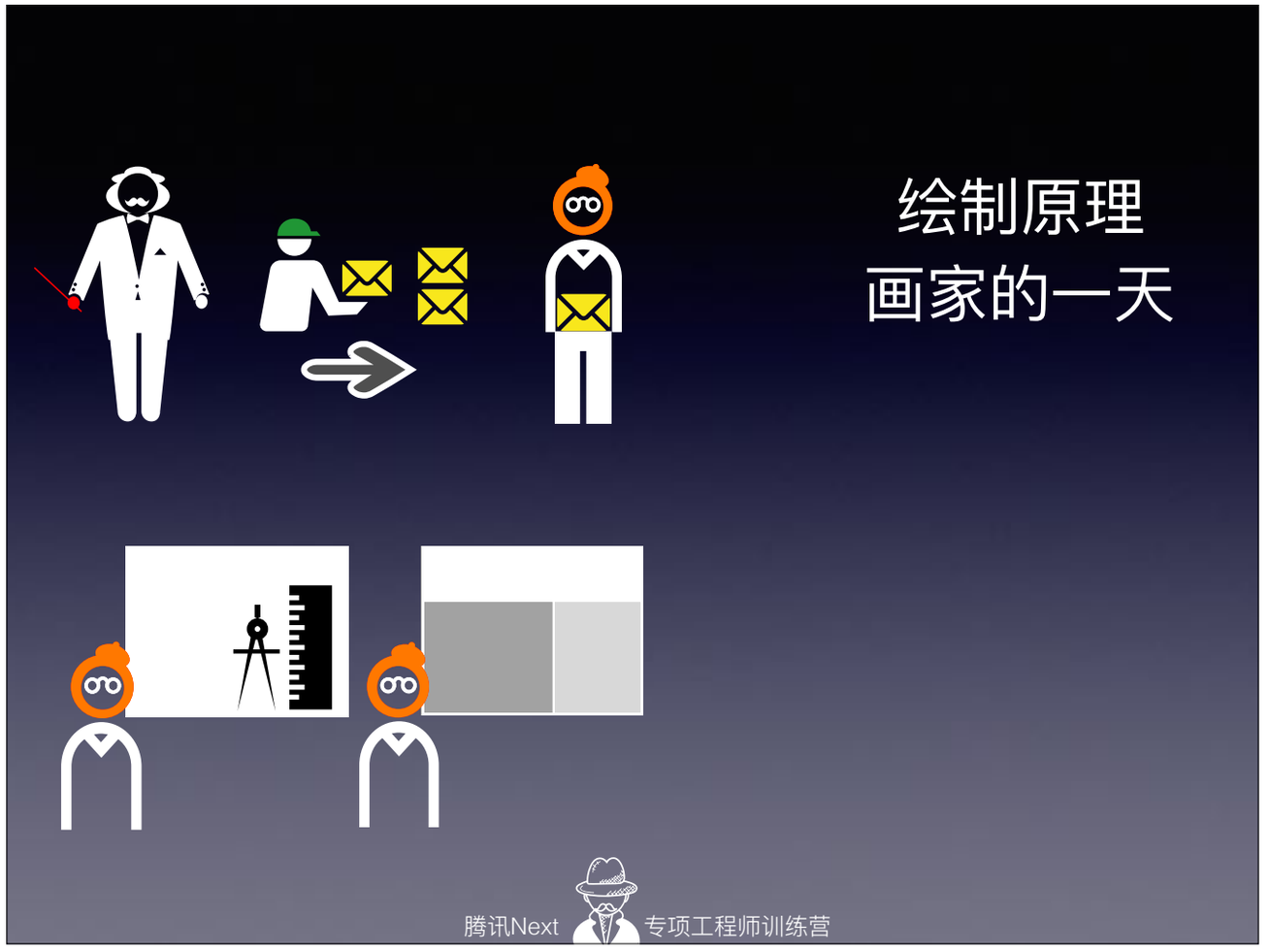
绘制原理
画家的一天

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.

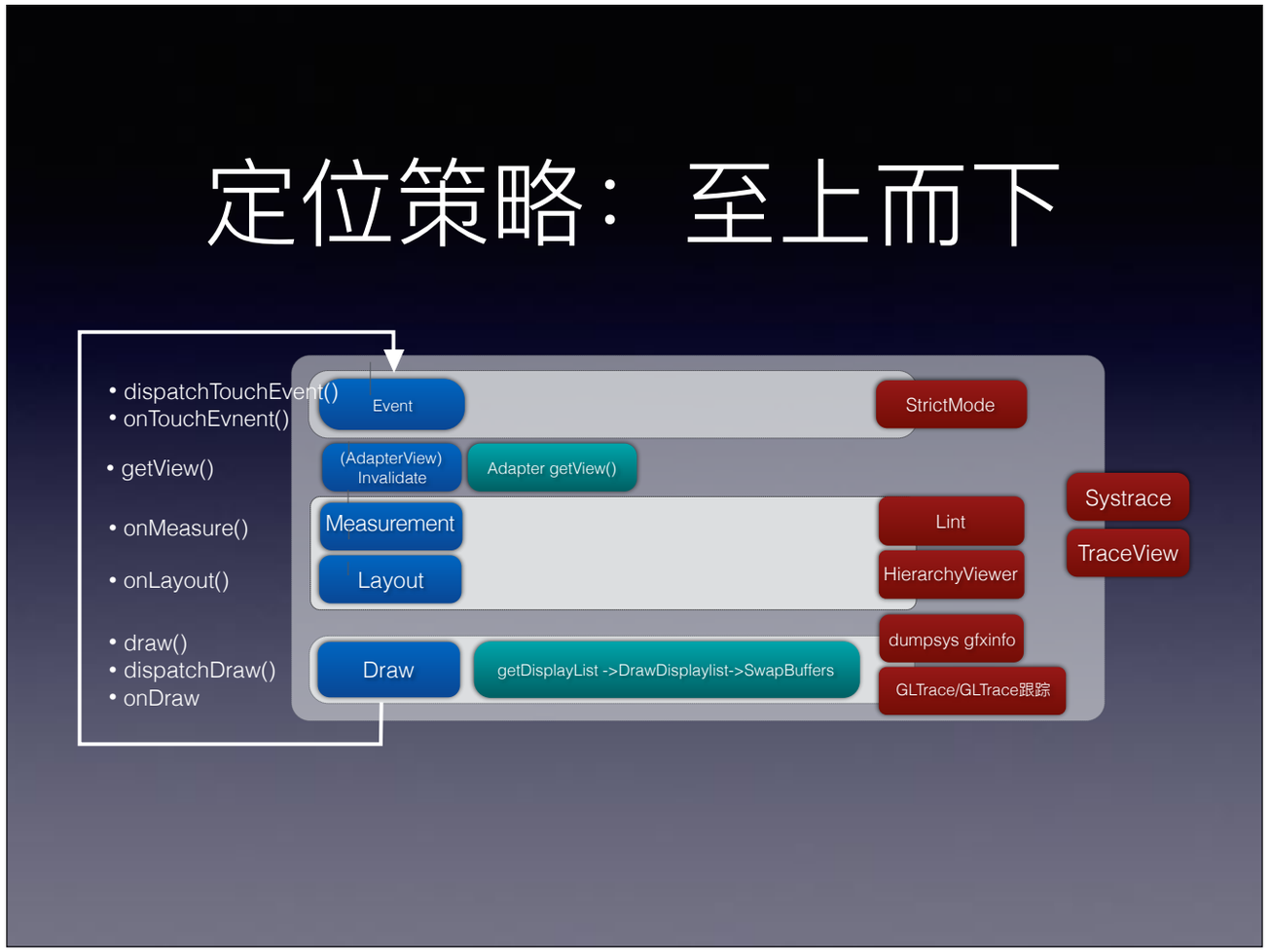
编舞者choreographer，有点像老板， 屏幕要”画“，信息告诉了画家， 画家就开始画， 也就是doframe，doframe的第一步是衡量， 然后就是布局， 布局完之后就开画了，画的第一步是构思，要画什么？ 那就是getdisplaylist， 然后开始动笔就是drawdislaylist，然后交给老板，swapbuffers.



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.

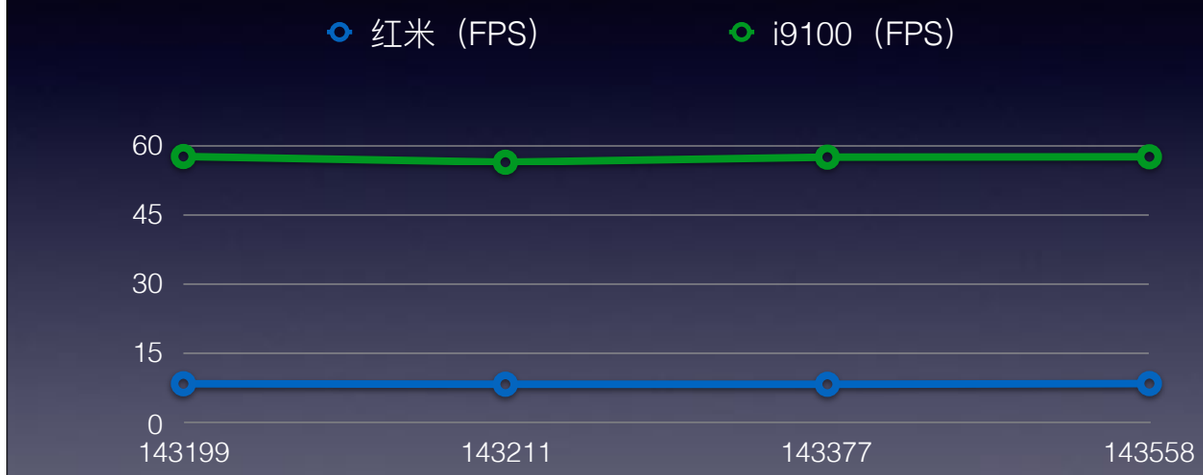
编舞者choreographer，有点像老板， 屏幕要”画“，信息告诉了画家， 画家就开始画， 也就是doframe，doframe的第一步是衡量， 然后就是布局， 布局完之后就开画了，画的第一步是构思，要画什么？ 那就是getdisplaylist， 然后开始动笔就是drawdislaylist，然后交给老板，swapbuffers.



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

measure

layout

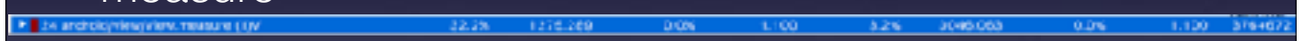
draw

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

measure



layout

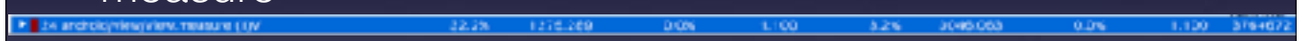
draw

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。

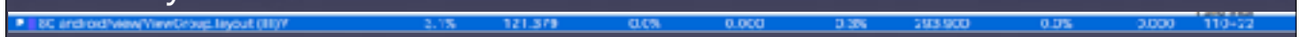
红米会话列表卡顿

getView

measure



layout

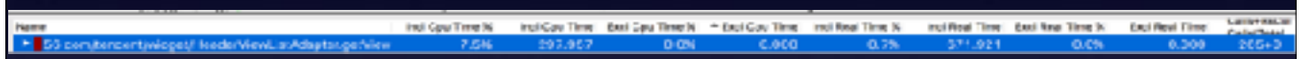


draw

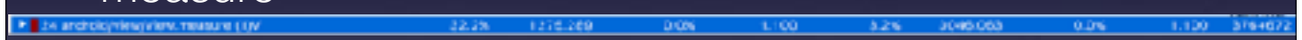
会到问题，我们针对会话列表的场景录制了trace, 发现getview, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿

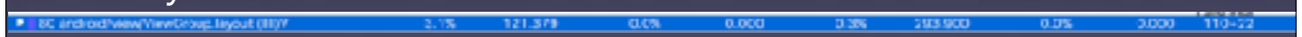
getView



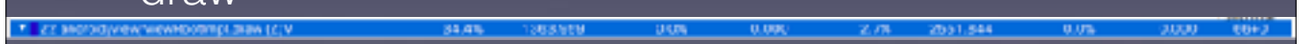
measure



layout



draw

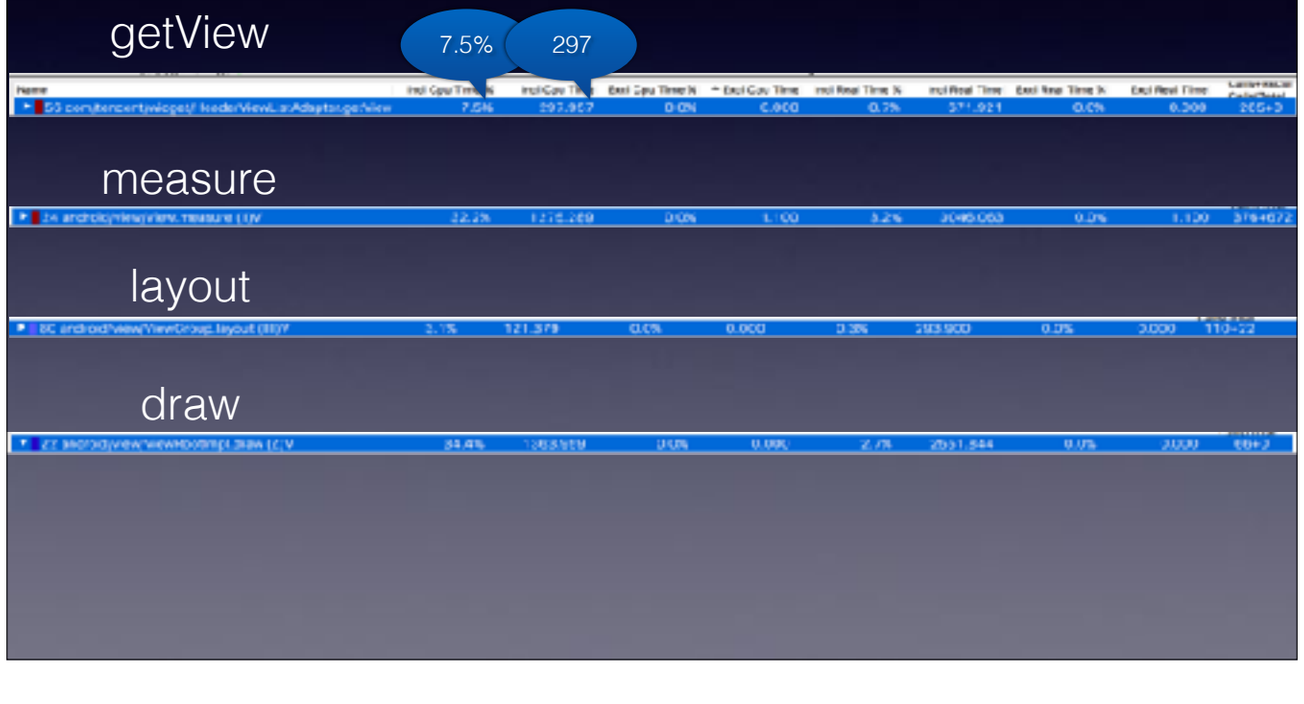


会到问题，我们针对会话列表的场景录制了trace, 发现getview, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿

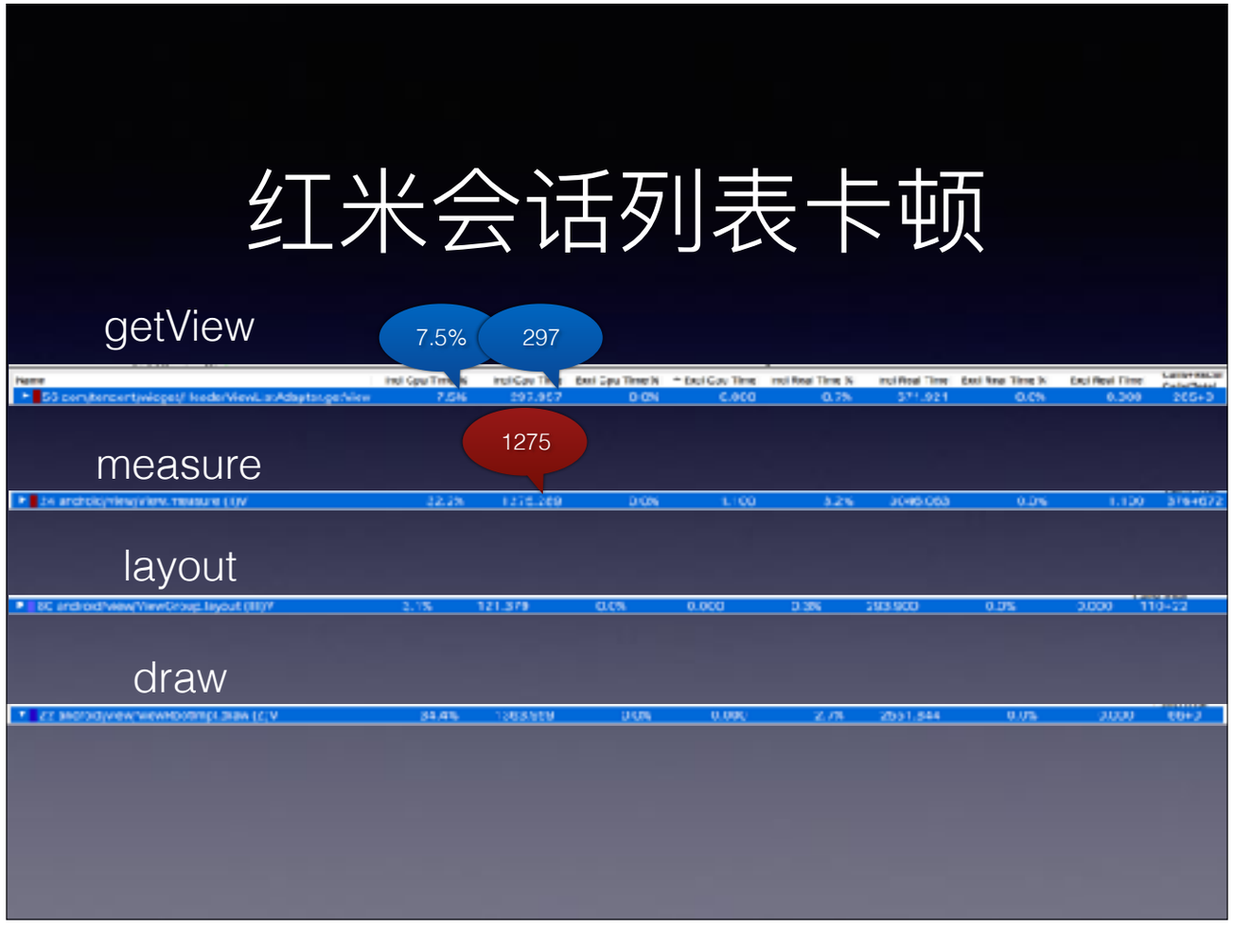
会到问题，我们针对会话列表的场景录制了trace, 发现getview, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿



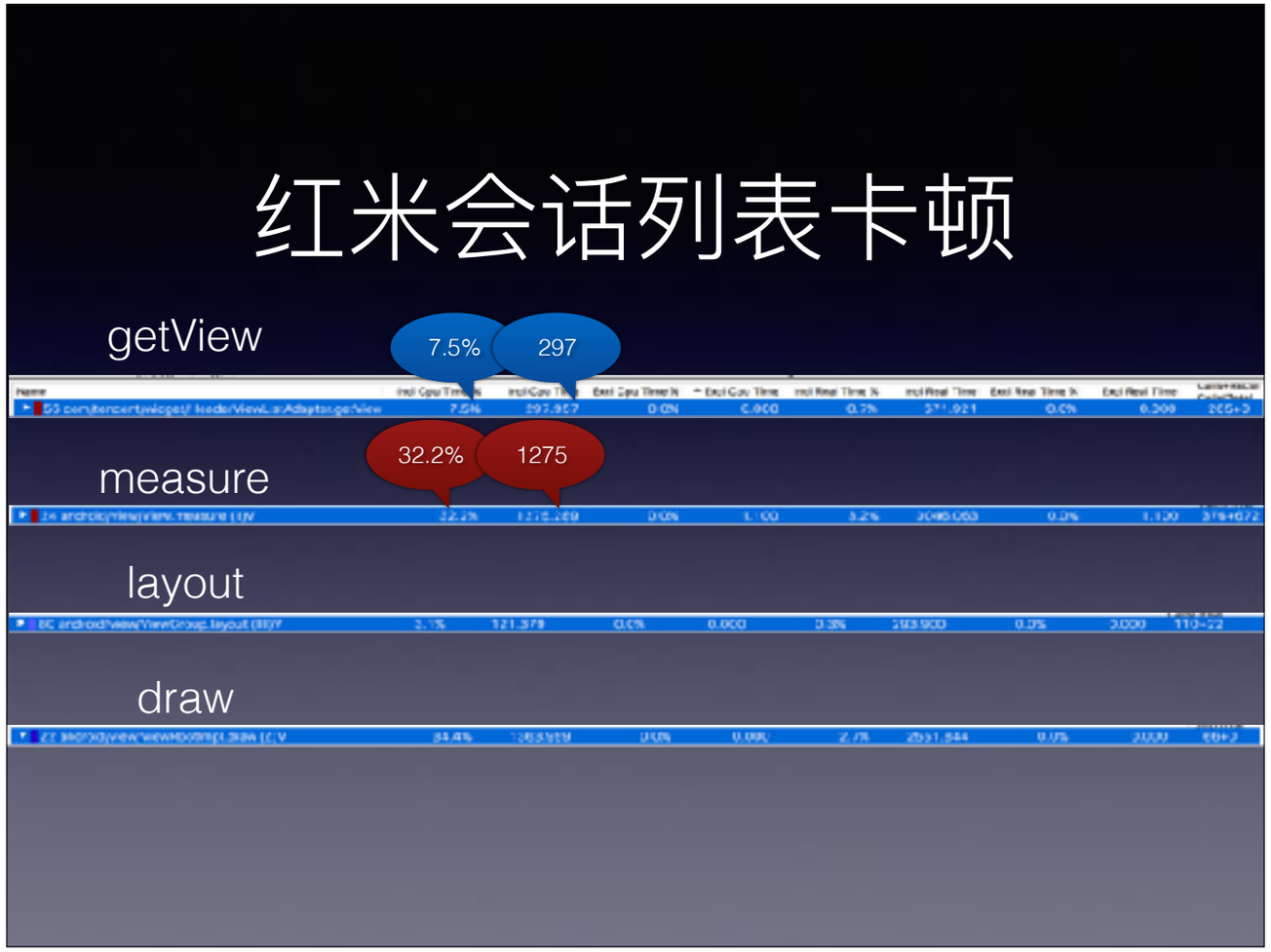
会到问题，我们针对会话列表的场景录制了trace, 发现getview, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿



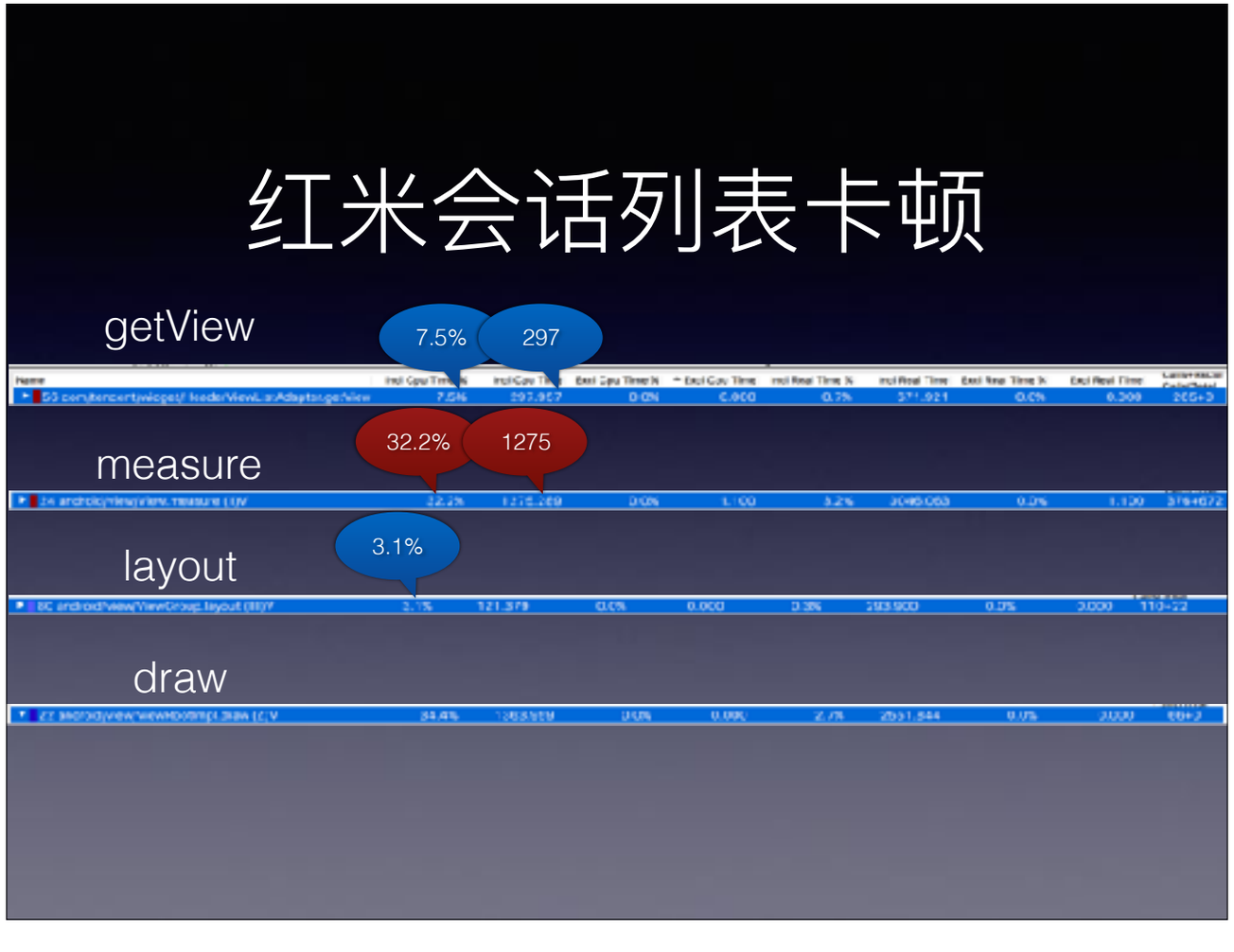
会到问题，我们针对会话列表的场景录制了trace, 发现getview, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿



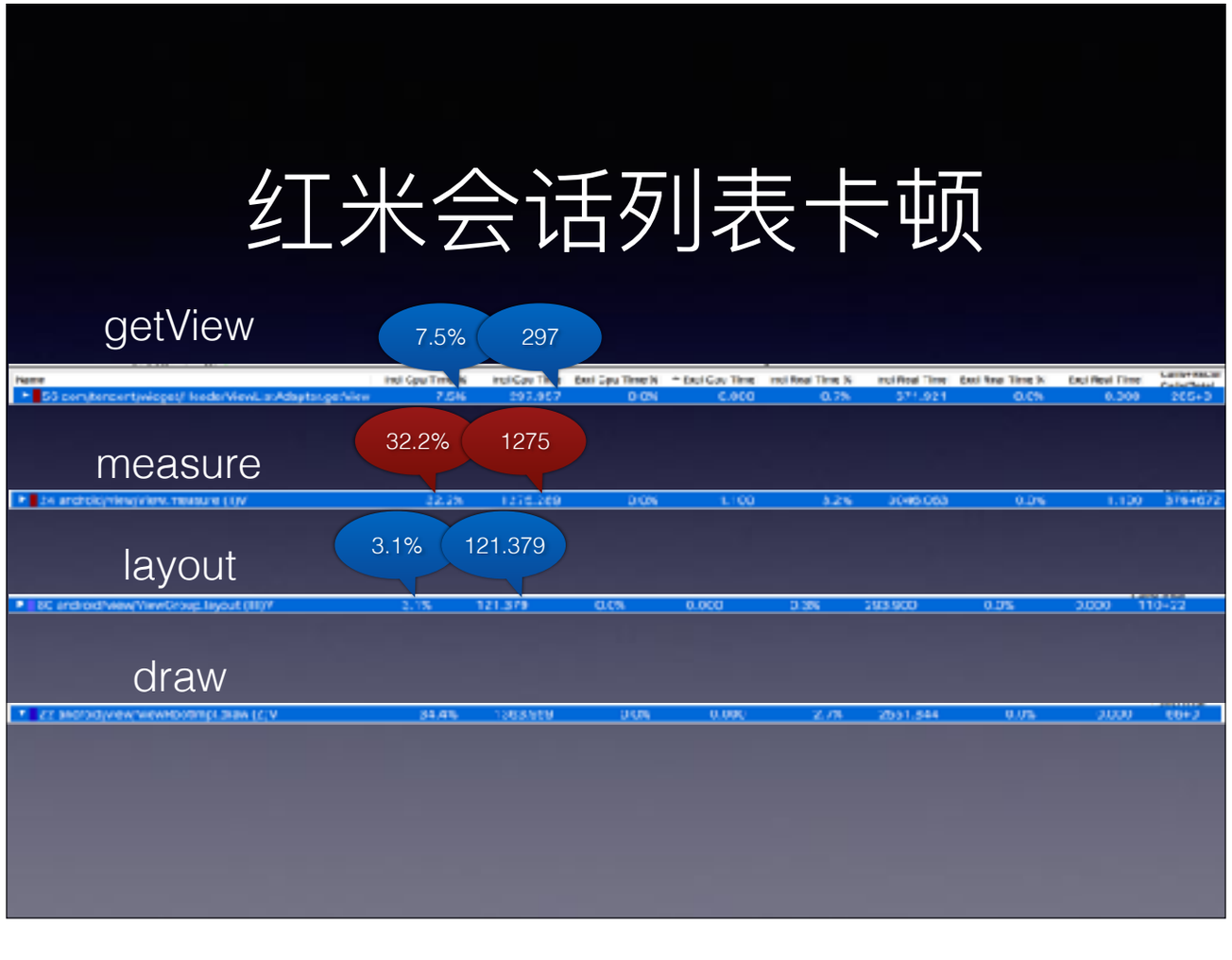
会到问题，我们针对会话列表的场景录制了trace, 发现getview, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿



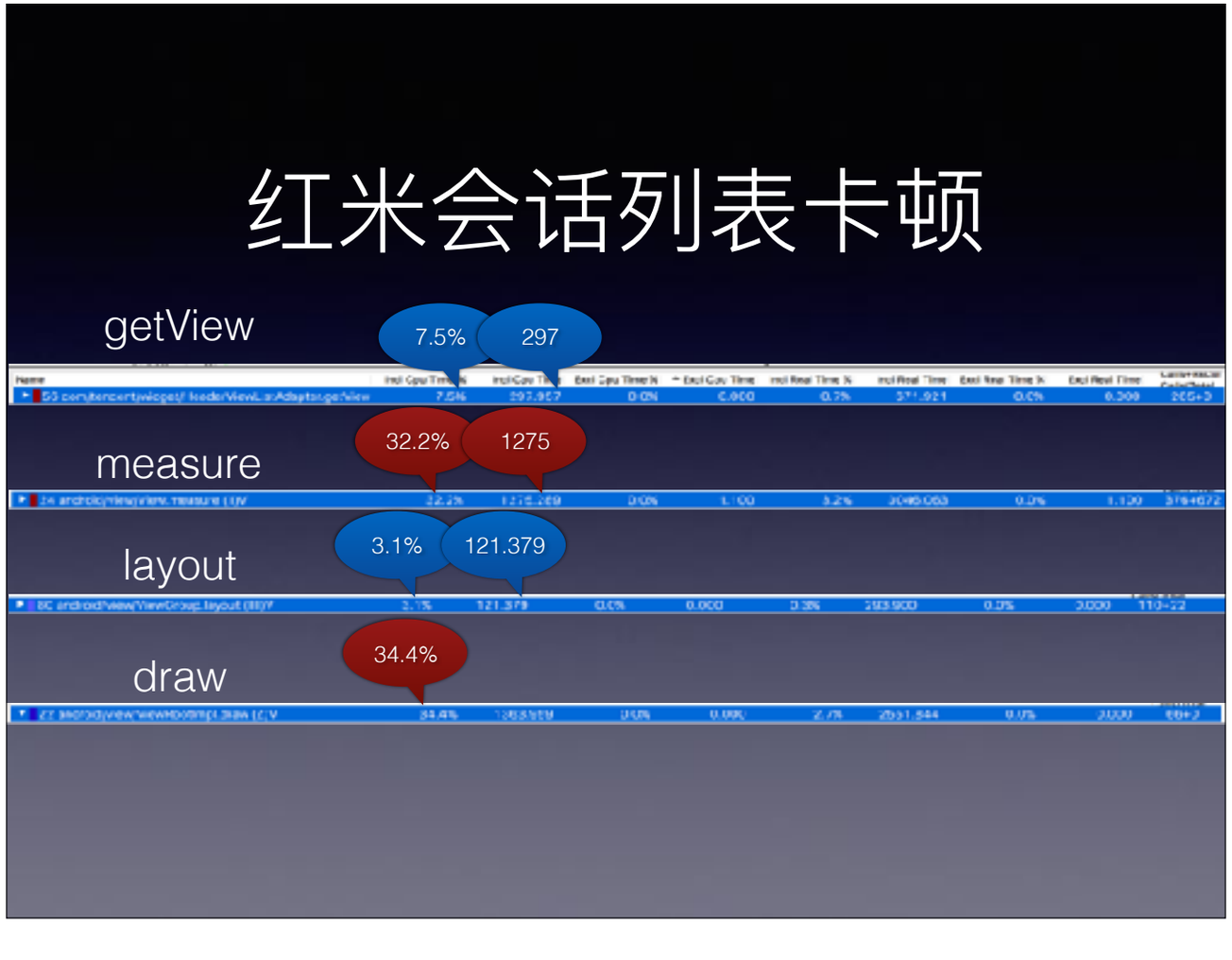
会到问题，我们针对会话列表的场景录制了trace, 发现getview, measure, layout, draw耗时，和百分比如上。

红米会话列表卡顿



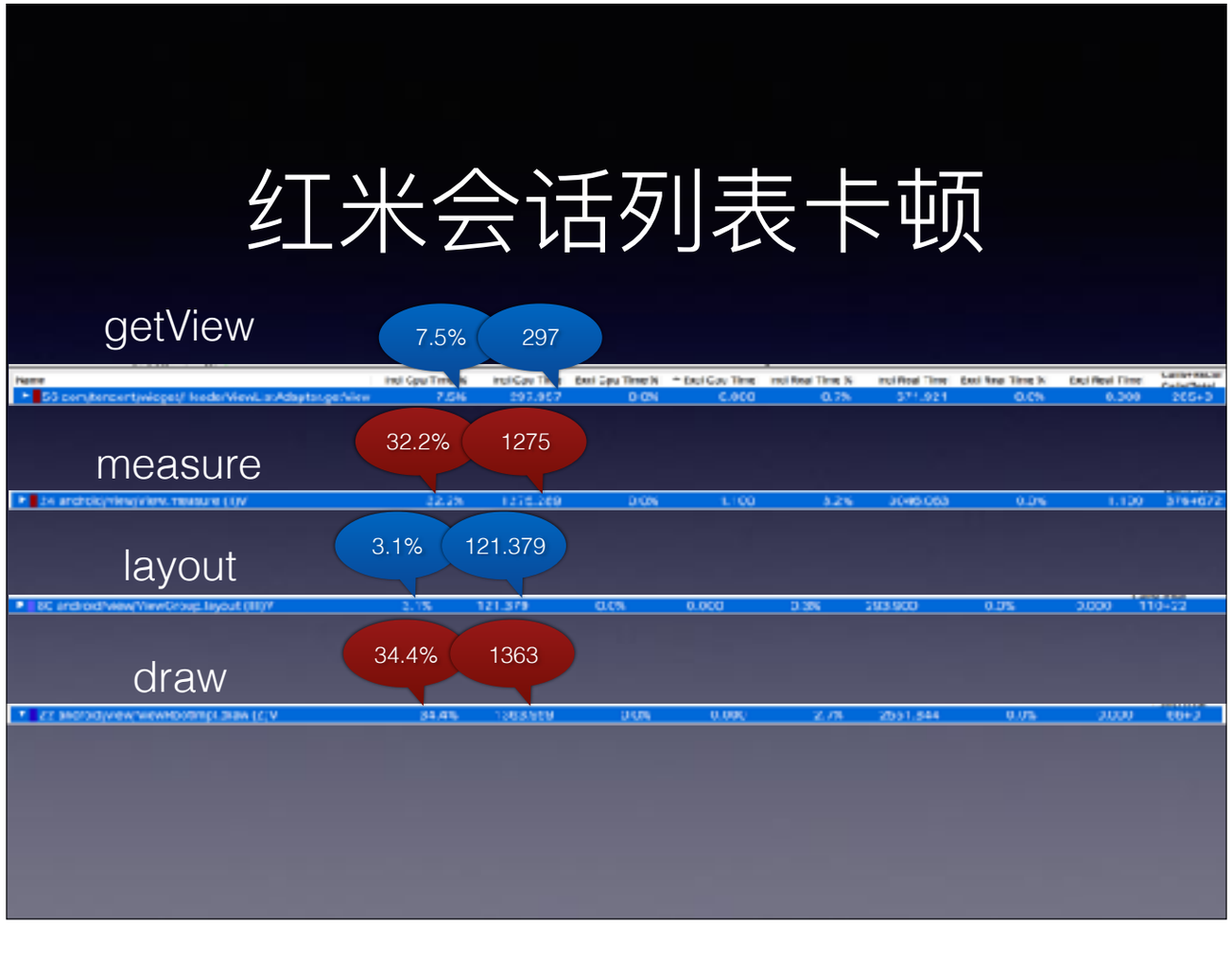
会到问题，我们针对会话列表的场景录制了trace, 发现getview, measure, layout, draw耗时，和百分比如上。

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。

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

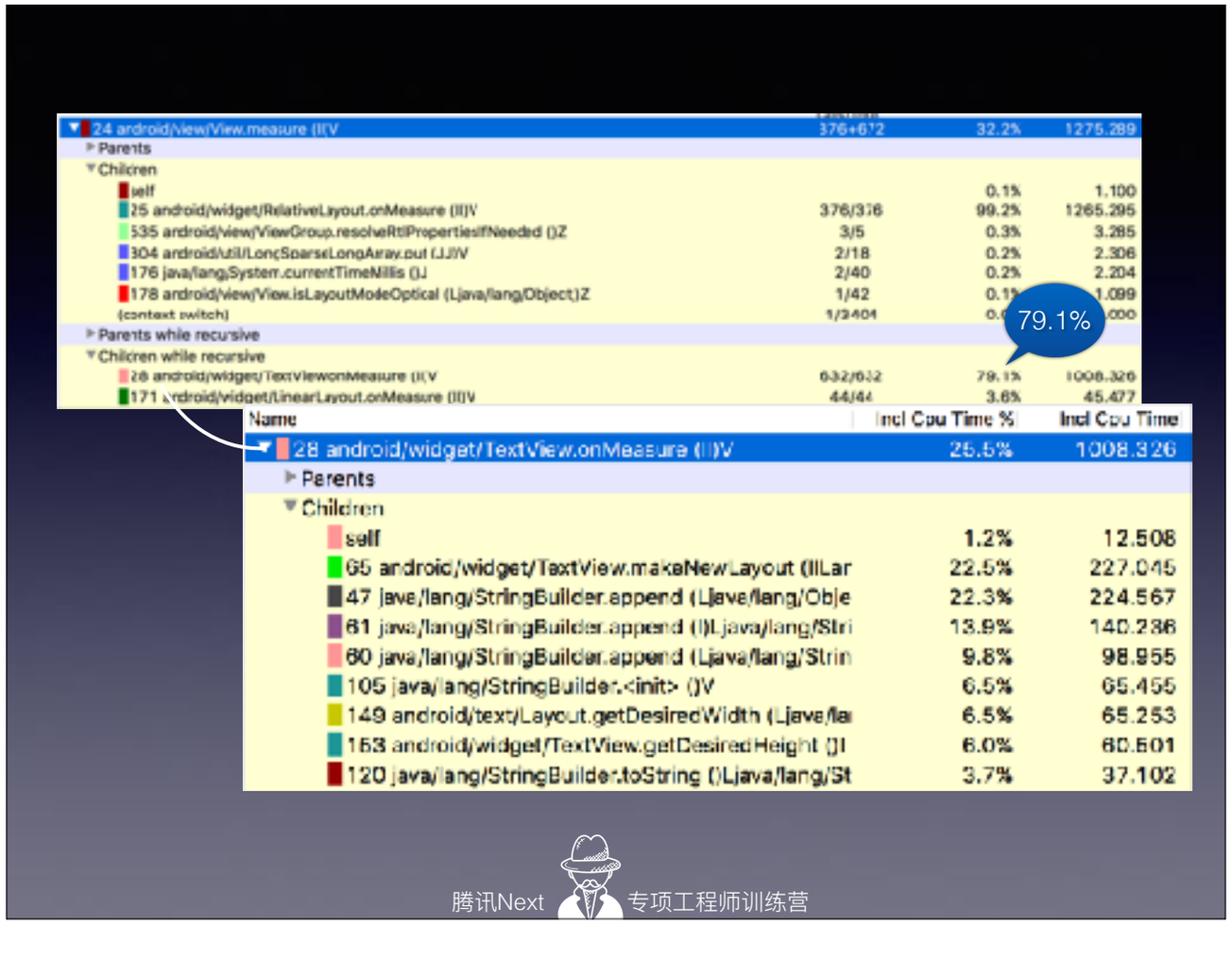
最后怎么解决的呢？ 我先不说具体怎样解决。大家有没有发现这个思路很顺畅，并非试一下，是至上而下的，借助traceview能做到。但是这里有个问题。

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。

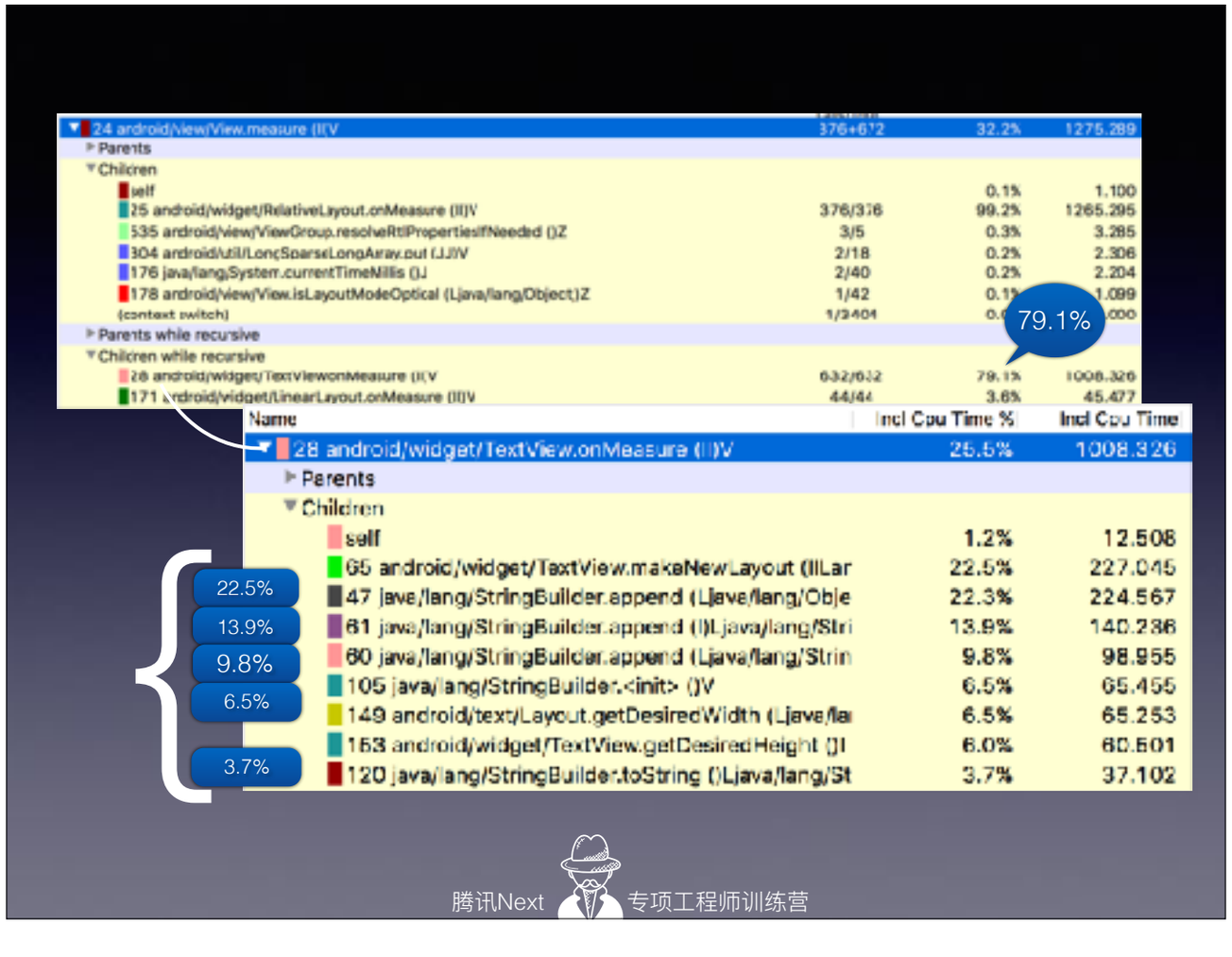
进一步看measure. 我们发现两点。第一个SB, 56.4, 另外，textview measure特别多。

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。



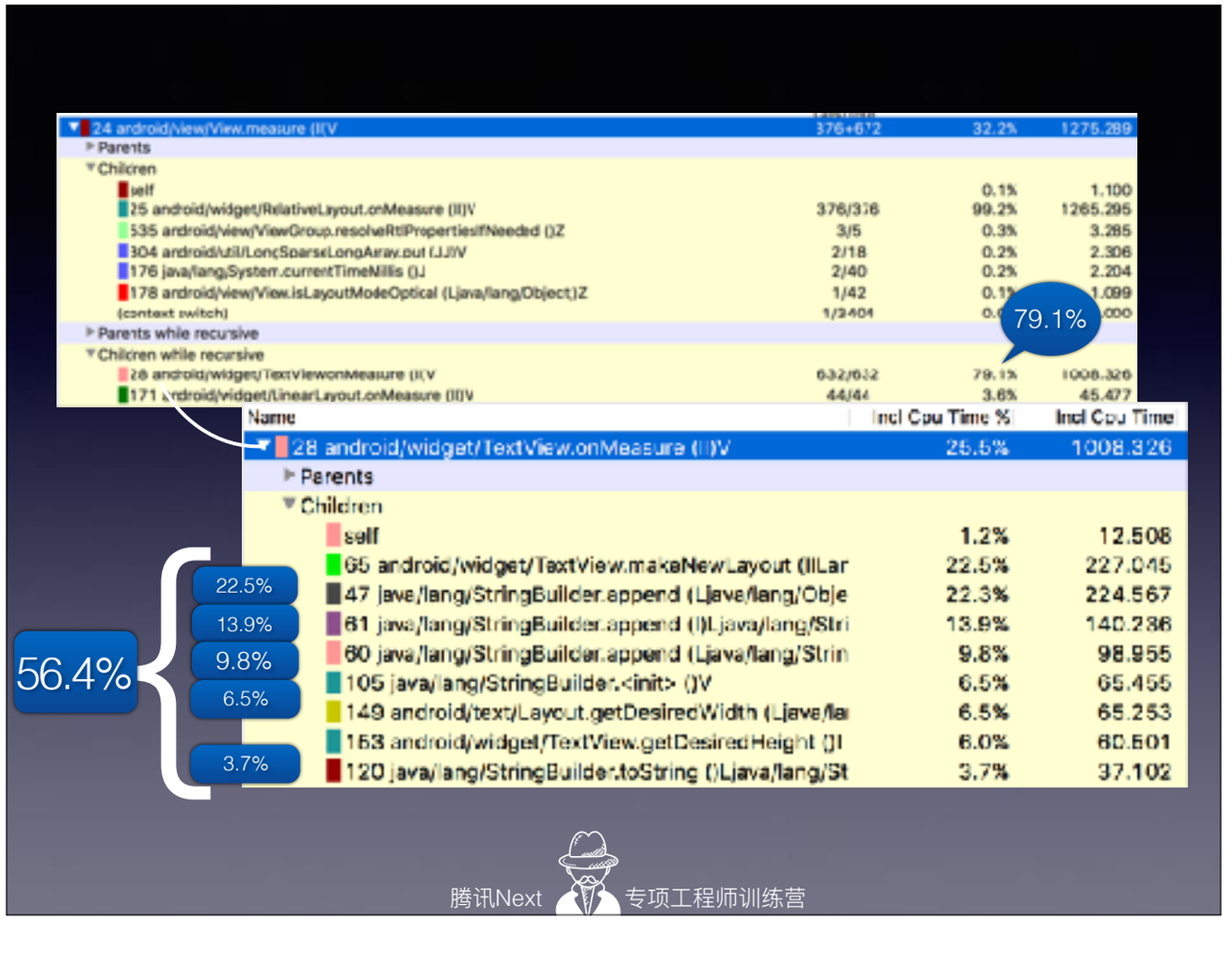
进一步看measure. 我们发现两点。第一个SB, 56.4, 另外，textview measure特别多。

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。



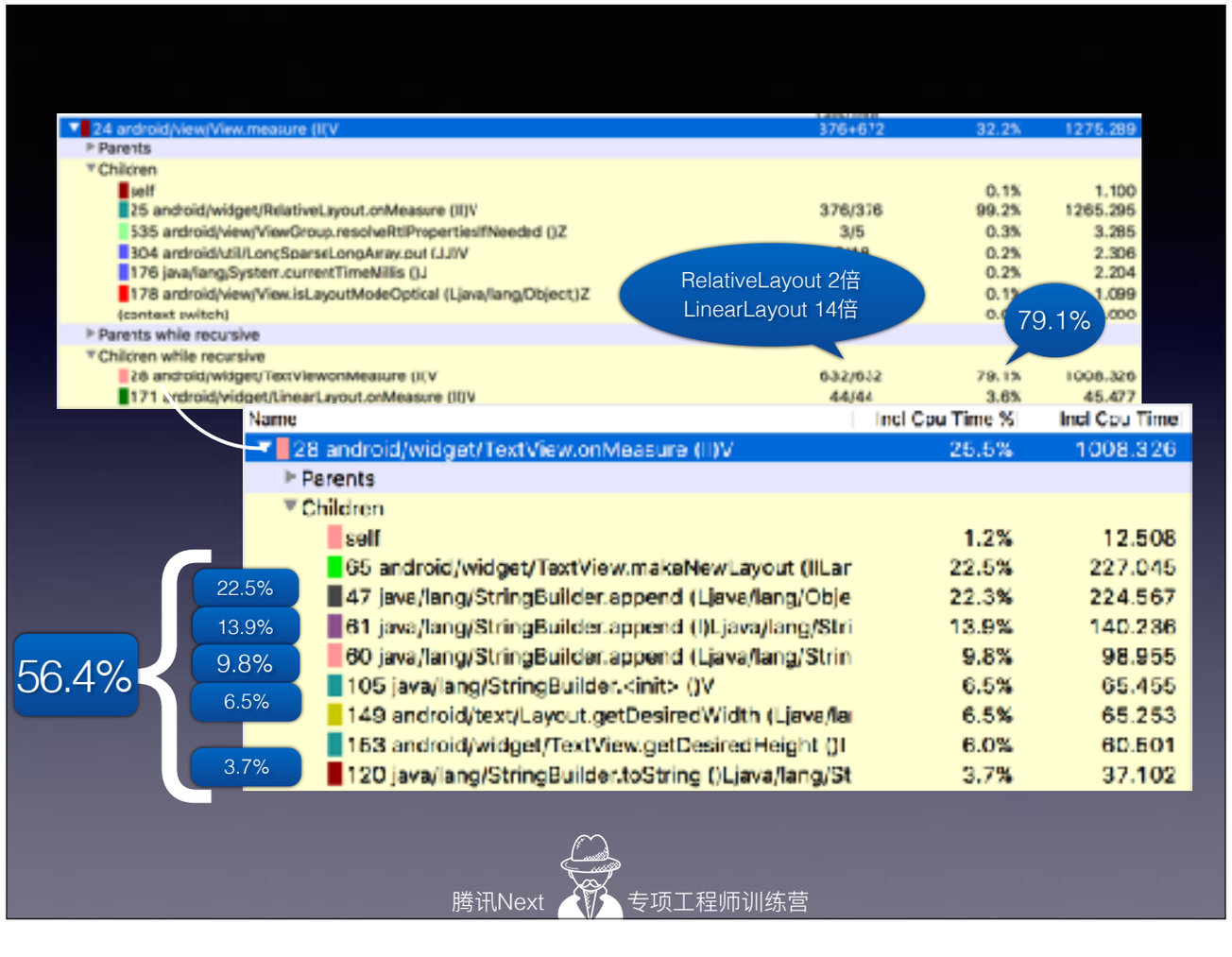
进一步看measure. 我们发现两点。第一个SB, 56.4, 另外，textview measure特别多。

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。

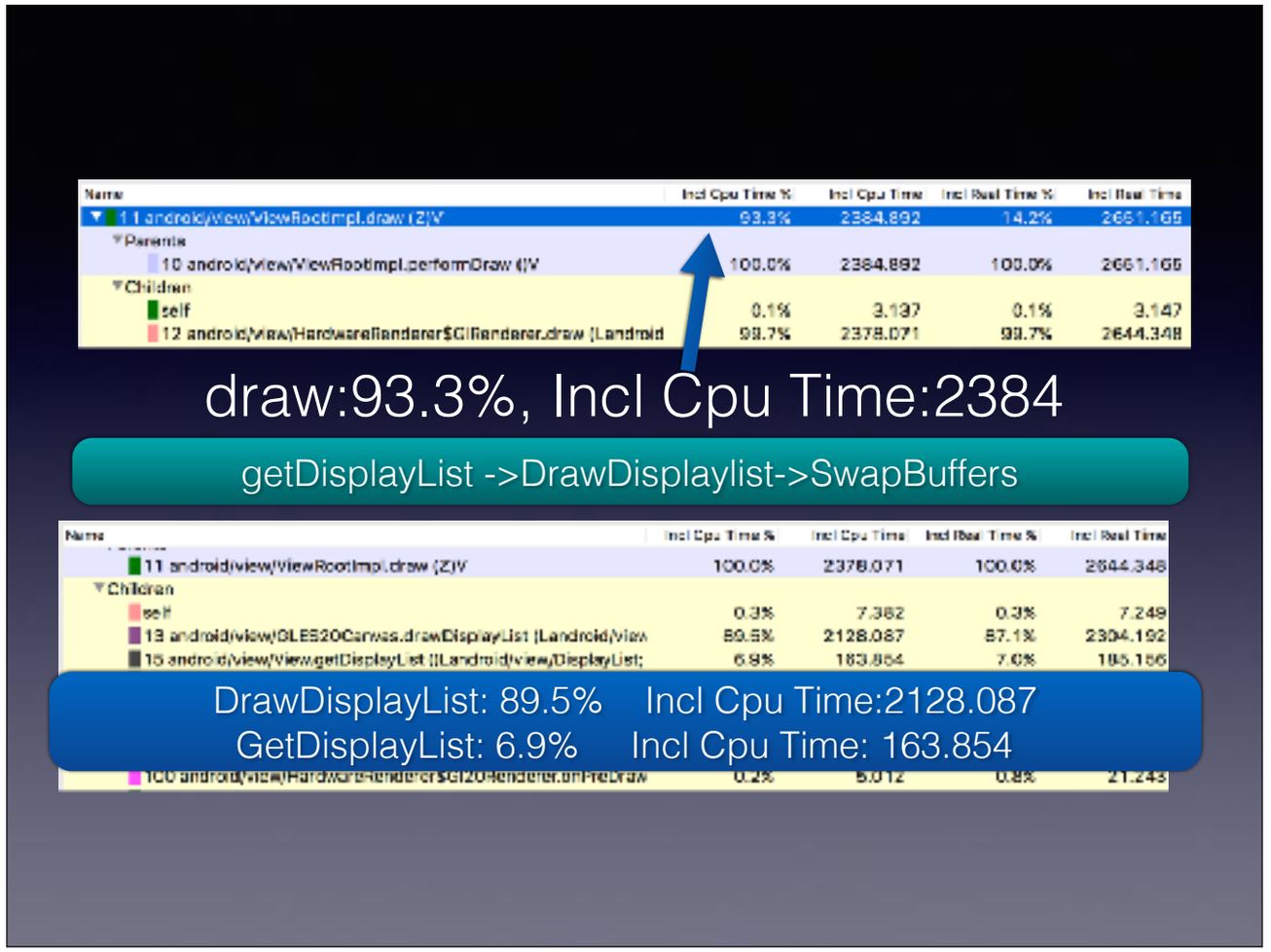
最后怎么解决的呢？ 我先不说具体怎样解决。大家有没有发现这个思路很顺畅，并非试一下，是至上而下的，借助traceview能做到。但是这里有个问题。

traceview是否万能的?

这里引入另外一个案例。

长文本卡顿问题

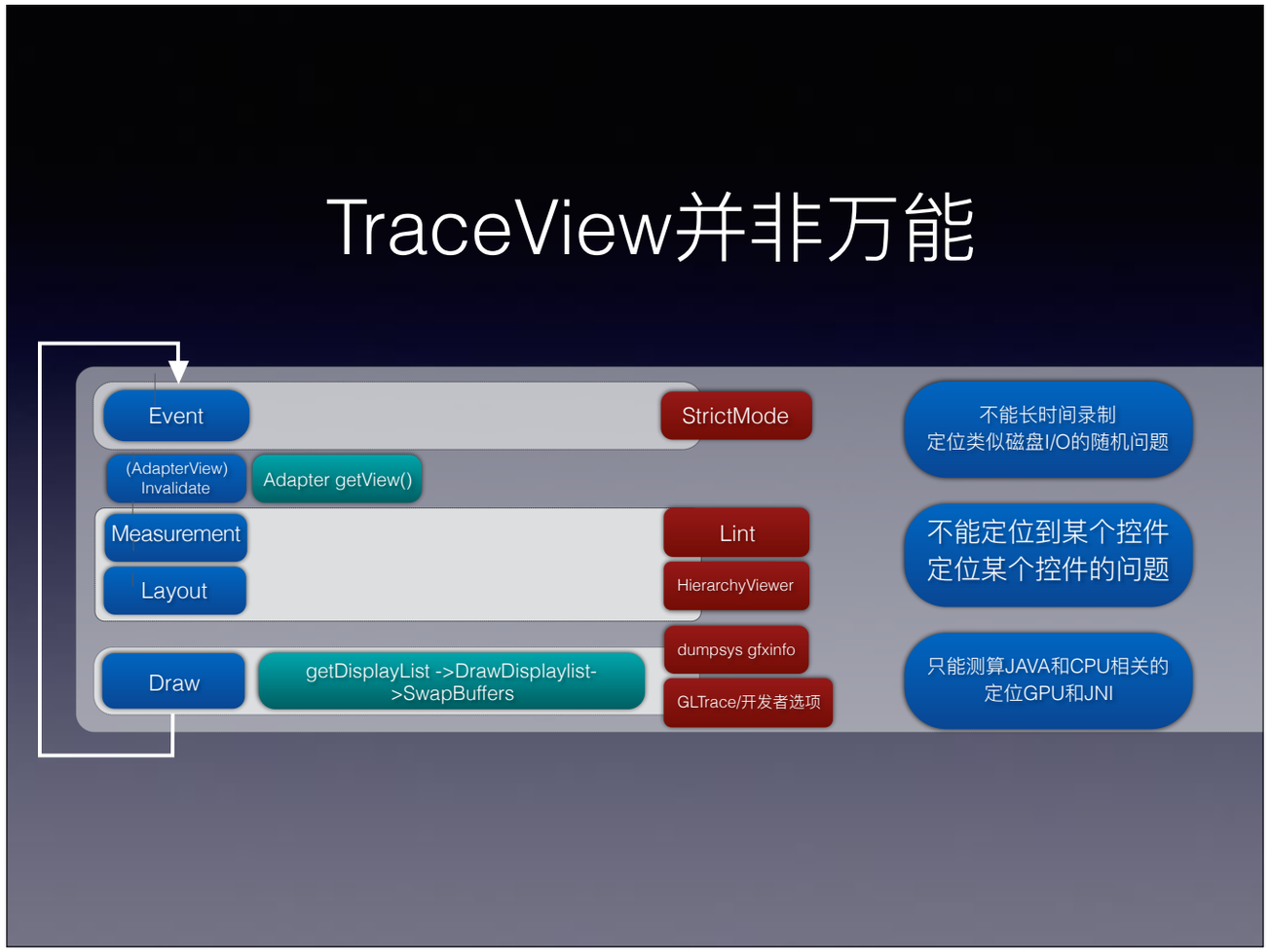
15FPS!



93.3%的Draw, 其他东西就完全不需要看了。

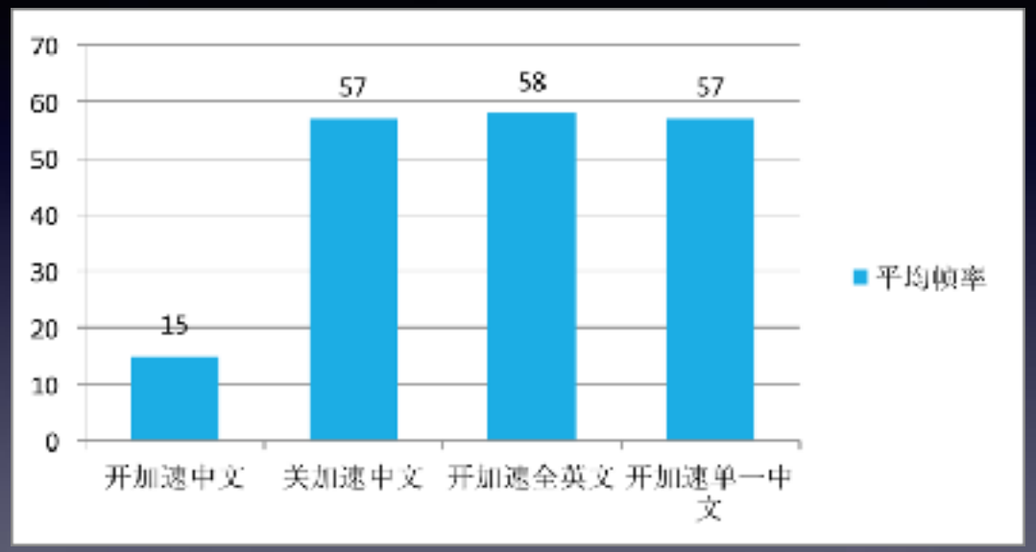
跟不下去了，悲剧
起码定位了Draw

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.wi	展开红色	14.606	11.032	32.4 %	147
com.tencent .getView	展开红色	9.637	7.150	34.8 %	97
com.tenc dapter1.getView	展开红色	9.637	7.140	35.0 %	97
com.tencent ItemLayout.onMeasure	展开红色	2.422	1.384	75.0 %	24
com.tencent motionAdapter.getView	展开红色	0.780	0.215	262.1 %	8
com.tencent.mobil elViewPagerAdapter.instantiateItem	展开红色	0.436	0.128	239.8 %	4
com.tencent el.AudioPanel.onMeasure	展开红色	0.298	0.109	174.4 %	3
com.tencent .getItemViewType	展开红色	0.177	0.105	69.2 %	2
com.tencent ayout	展开红色	0.108	0.058	86.7 %	1
com.tencent .onLayout	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.printStack	50113	39630191	

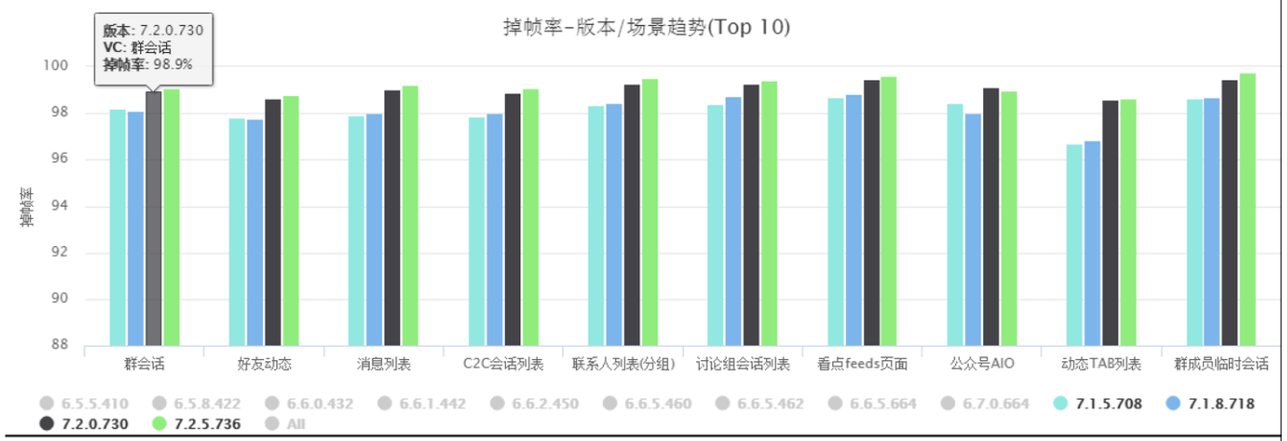
从这个排行版中，我们得到了量化的数据外，对于当时的我们来说，是得到了一个反直觉的判断

至下而上:FileExist影响范围

堆栈	个数	总量	占百分比	所属业务
#com.tencent.theme.SkinEngine.a#com.tencent.theme.SkinEngine.setSkin	403	2370	17.0%	皮肤
#com.tencent.mobileqq.emoticon.EmojiGridAdapter.instantiateItem#com.tencent.mobileqq.emoticon.EmojiListenerManager.isEmojiFileExist#java.io.File.exists	76	2370	3.2%	表情
#com.tencent.mobileqq.activity.QQLSActivity$SingleItemAdapter.getView#com.tencent.mobileqq.app.message.QQMessageFacade.decodeMsg#com.tencent.mobileqq.app.message.BaseMessage.parse#com.tencent.qqnt.data.MessageForPic.doParse#java.io.File.exists	61	2370	2.6%	图文混排
#com.tencent.mobileqq.activity.aio.ChatAdapter1.getView#com.tencent.mobileqq.activity.aio.BaseBubbleBuilder.getBubbleView#com.tencent.mobileqq.utils.FileUtils#com.tencent.mobileqq.transfile.AbsDownloader.getFilePath#java.io.File.exists	53	2370	2.2%	图片消息
#com.tencent.util.WeakReferenceHandler.handleMessage#com.tencent.mobileqq.activity.recent.DrawerFrameLayout$3.onDrawerOpened#com.tencent.mobileqq.activity.QQSettingMe.doOnCreateView#com.tencent.mobileqq.task.ApolloActionHelper.isResValid#com.tencent.mobileqq.apollo.utils.ApolloResUtil	39	2370	1.6%	apollo

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）

到了解决。 也是给大家一句名言。为什么这样说呢?

解决

问题	解决思路
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);

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?

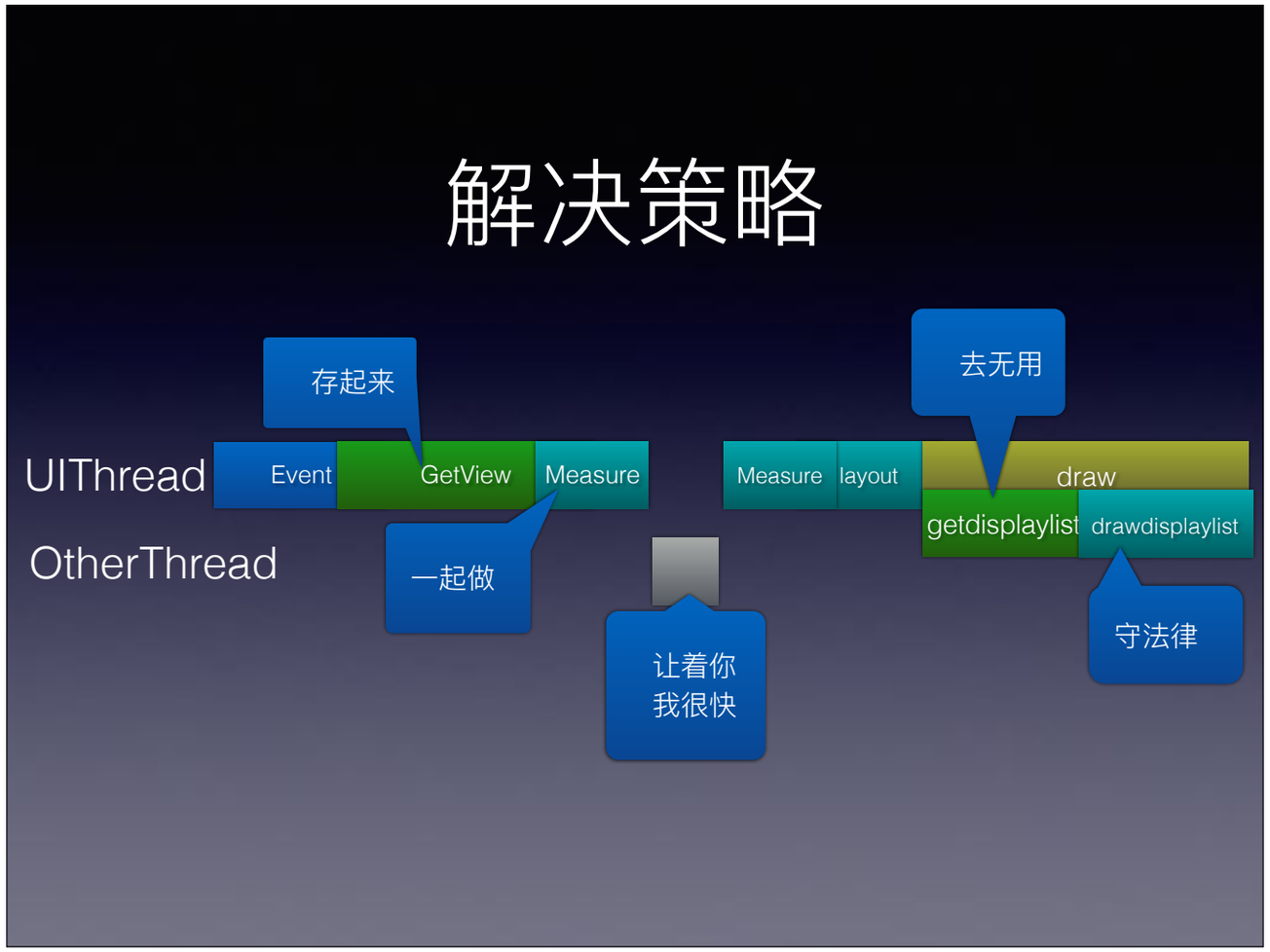
输出了组件，让能力可以快速复制外。我们还发现StringBuilder这个货，可能带来的性能风险，包括它的GC和扩容，关联到了日志，并开始进行优化，虽然最后是开发同学也想到并落地修改了，但至少证明方向没有错。

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。

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。

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

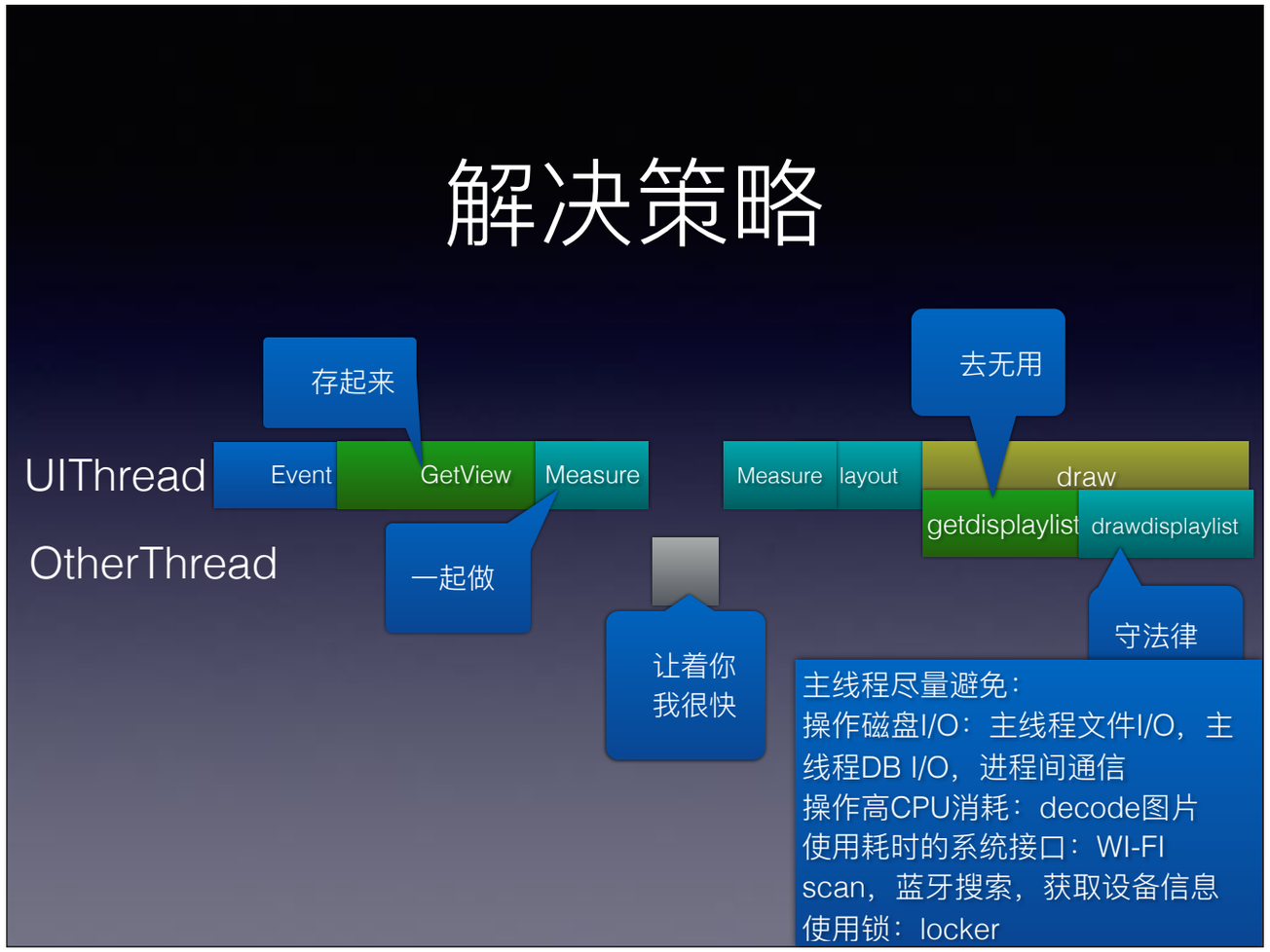
去无用: 去掉无用的，例如overdraw的，除了去掉多余的background, 针对自定义的复杂的view还可以canvas.cliprect，canvas.quickreject

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



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%(2358K)	98.4%(1037K)	98.2%(1062K)	98.6%(338K)	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%

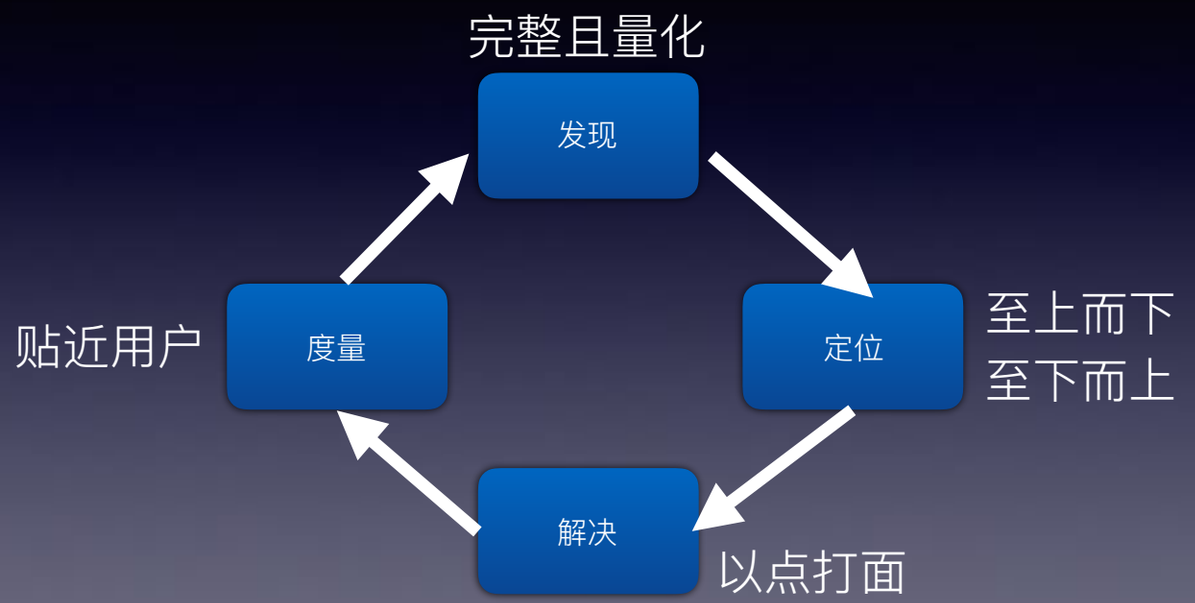
节点	辅助功能	版本占%	基准版本占%	版本占比变化	耗时(秒)
com.tencent.wi	展开红色	14.606	11.032	32.4 %	147
com.tencent .getView	展开红色	9.637	7.150	34.8 %	97
com.tenc dapter1.getView	展开红色	9.637	7.140	35.0 %	97
com.tencent ItemLayout.onMeasure	展开红色	2.422	1.384	75.0 %	24
com.tencent motionAdapter.getView	展开红色	0.780	0.215	262.1 %	8
com.tencent.mobil elViewPagerAdapter.instantiateItem	展开红色	0.436	0.128	239.8 %	4
com.tencent el.AudioPanel.onMeasure	展开红色	0.298	0.109	174.4 %	3
com.tencent .getItemViewType	展开红色	0.177	0.105	69.2 %	2
com.tencent ayout	展开红色	0.108	0.058	86.7 %	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%(2358K)	98.4%(1037K)	98.2%(1062K)	98.6%(338K)	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.wi	展开红色	14.606	11.032	32.4 %	147
com.tencent .getView	展开红色	9.637	7.150	34.8 %	97
com.tenc dapter1.getView	展开红色	9.637	7.140	35.0 %	97
com.tencent ItemLayout.onMeasure	展开红色	2.422	1.384	75.0 %	24
com.tencent motionAdapter.getView	展开红色	0.780	0.215	262.1 %	8
com.tencent.mobil elViewPagerAdapter.instantiateItem	展开红色	0.436	0.128	239.8 %	4
com.tencent el.AudioPanel.onMeasure	展开红色	0.298	0.109	174.4 %	3
com.tencent .getItemViewType	展开红色	0.177	0.105	69.2 %	2
com.tencent ayout	展开红色	0.108	0.058	86.7 %	1

小结



Thank you

腾讯Next 专项工程师训练营

相关链接

渲染机制