

SDCC 2017 | 上海

互联网应用架构实战峰会

CSDN

## 新瓶旧酒——换个角度提升 App 性能和质量的实践之路

高亮亮@饿了么移动技术部

2017.3.19

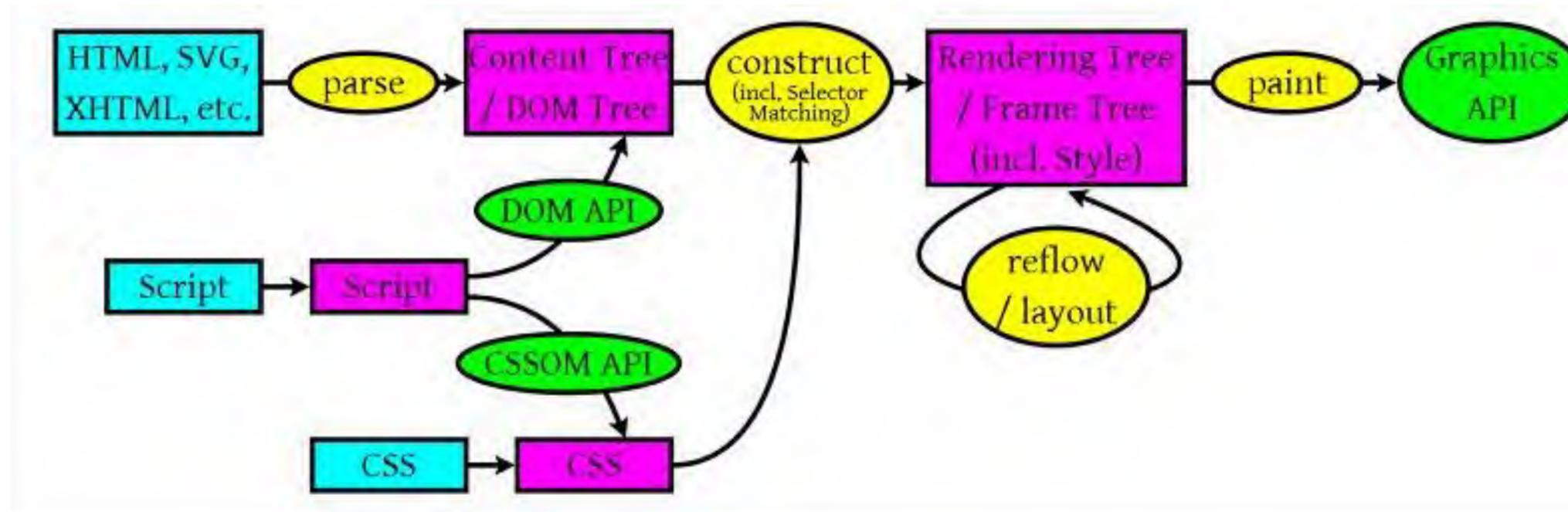
## 内容梗概

- 移动性能与质量概述
- “新”技术概念的介绍与实践
- 违反常规的“真理”

## 移动性能与质量概述

- 应用分级以及与性能质量的关系
- 根据设备特点设计提升方案
- 结合主要业务场景制定优先级

## 概念



- 回流：流式布局下，由于参照元素布局框发生变化而导致的布局重新计算。
- 重绘：元素布局不发生变化的情况下，重新渲染视图。

## 案例重现

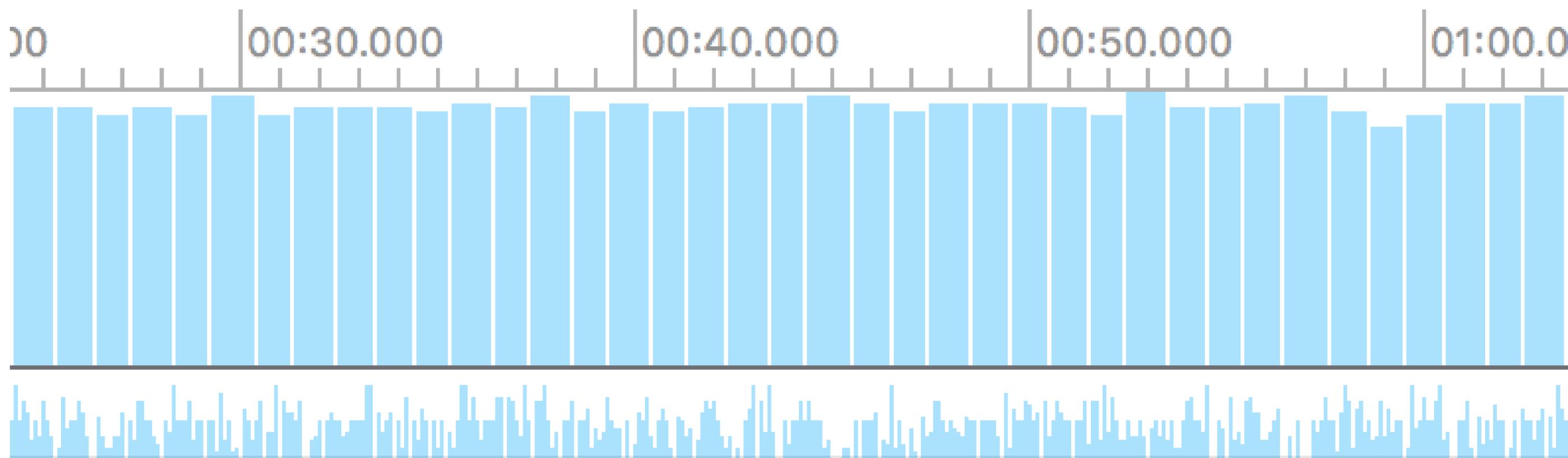
商品			收起 ^
1号篮子			
红烧肉[重辣]	x1	4.00	
2号篮子			
狮子头	x1	5.00	
3号篮子			
奶茶[去冰+半塘]	x2	6.00	
共 4 件商品		[已支付] 20.00	
		本单收入 ? 15.00	
下单时间: 03-08 18:39	取消订单	打印	
单号: 1200840947760561352			

29 [预] 今日 18:40 送达 已接单

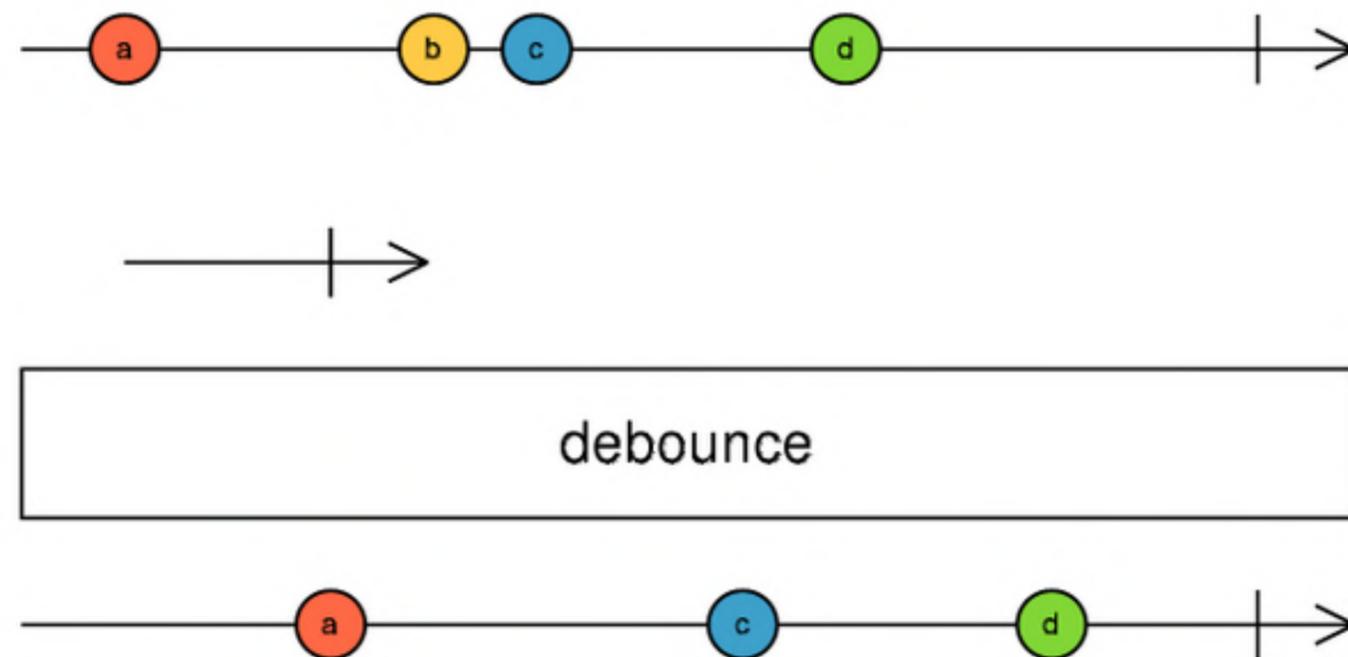
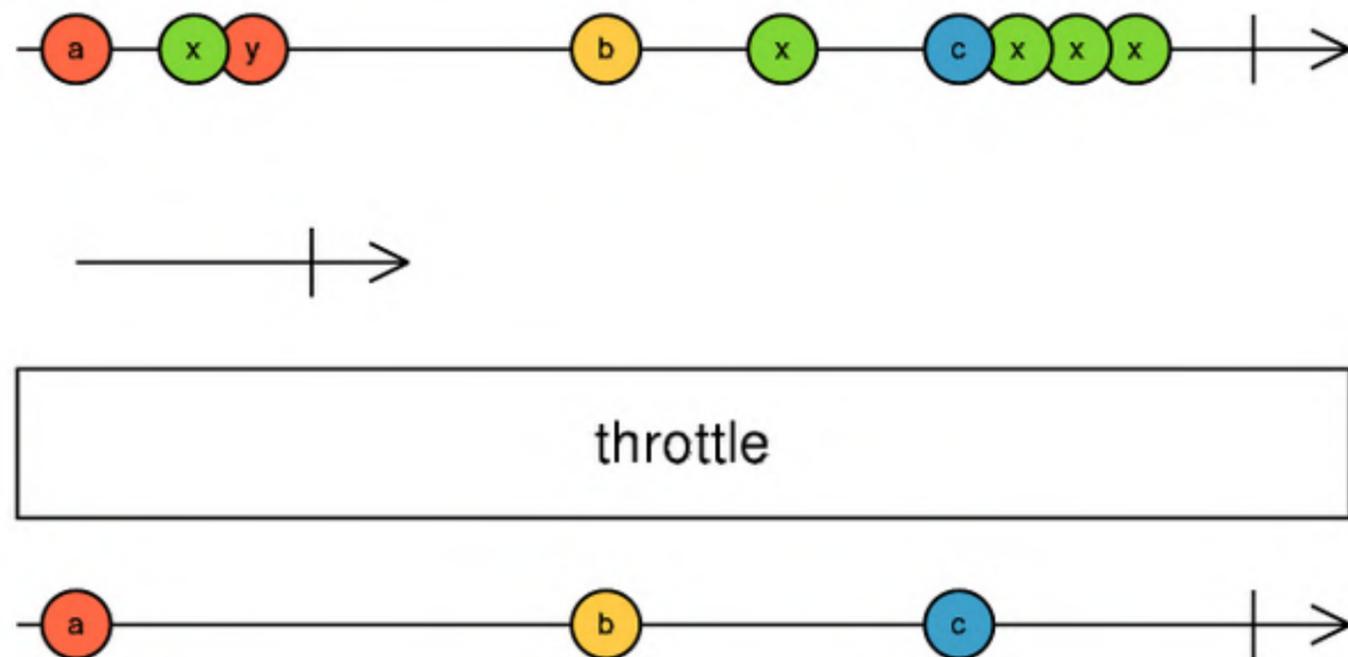
- 单张订单视图作为重用的基本单元
- 子视图层级复杂，且采用自动布局技术
- 子视图不固定，且存在强依赖关系
- 商品列表在滚动时产生严重回流

## 解决方案

- 调整视图关系，合理利用重用机制，避免滚动时回流
- ADK 方案，异步计算布局并缓存，细腻的线程控制



# 概念



## 案例重现

失败重试导致的 Self-DDoS

- 在保证服务前提下的自动重试，且固定重试频率
- 忽略错误类型，“一刀切”式的 DFF 设计
- 重试周期同步，从而导致恶性循环

## 解决方案

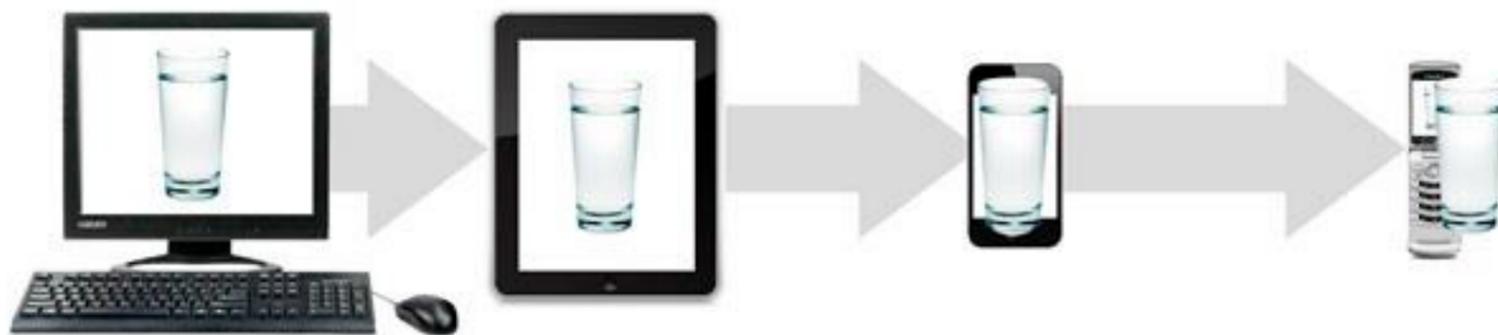
- 指数回退 —— 固定重试间隔加倍
- 添加抖动 —— 随机抖动间隔，避免锁定同步周期
- 标记重试 —— 优先处理高重试请求
- “黄金”重试节流策略

## 拓展运用

- 实时查询防抖 —— 合并网络请求
- 事件响应节流 —— 避免冗余资源消耗
- 界面渲染节流 —— 避免大量 CPU 消耗

# 概念

## Graceful Degradation



- != 容错设计 ( 替代品避免消极影响 )
- 做减法, 直到系统完全不可用

## Progressive Enhancement



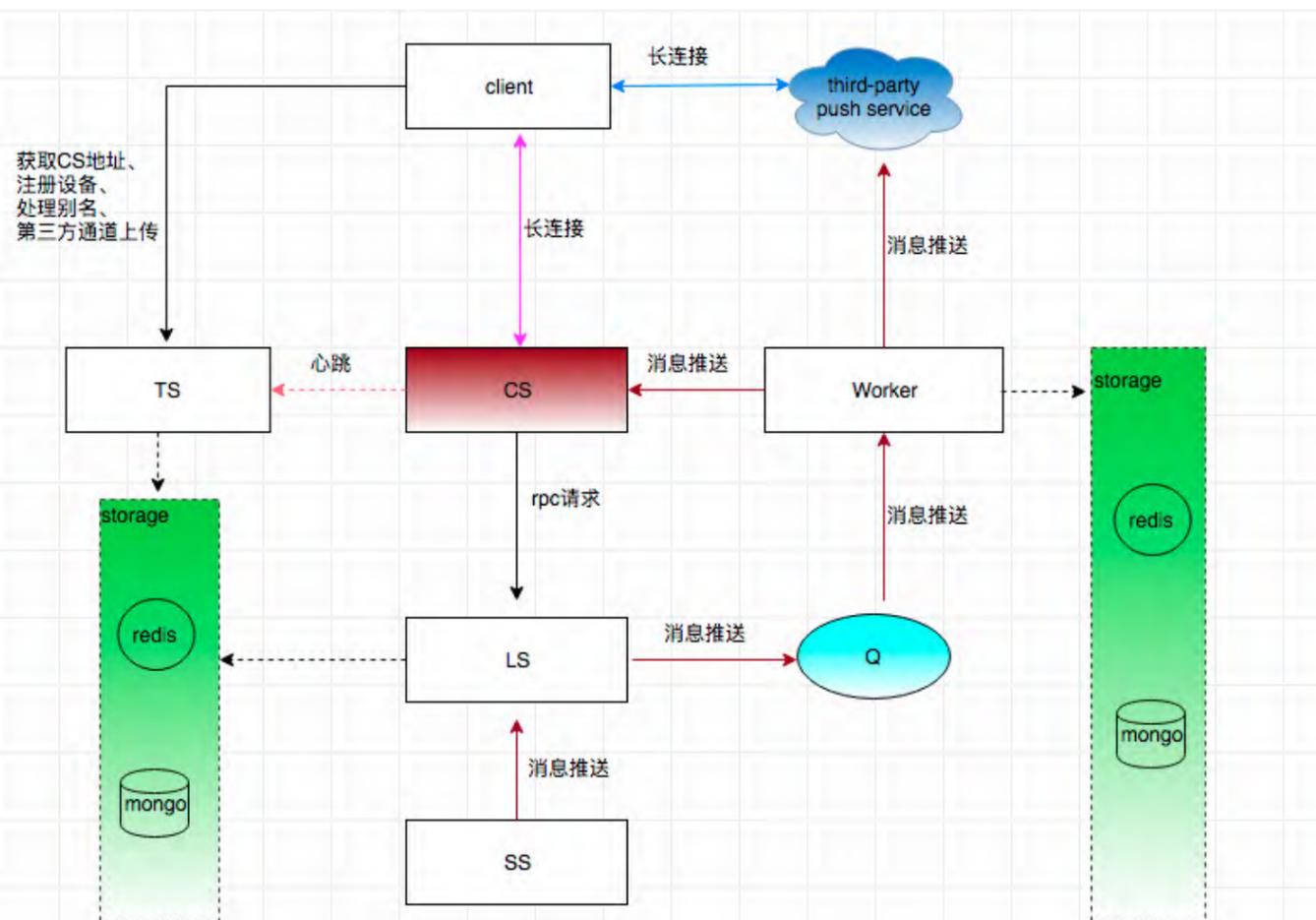
- 保证最基本的功能可用
- 做加法, 逐步提升系统

## 案例重现

### 基于三方服务的推送系统

- 不同业务对推送的实时性、可靠性要求高且存在差异
  - 利用更优的组件作为首选，三方作为备选
- 三方服务不可控，且在实时性、可靠性上都存在不足
  - 操作的效率和速度随着失效部件的增加逐渐下降

# 解决方案



TS: ticket server, 主要负责设备注册、别名处理、通道信息上传、分配CS(网关)地址  
 CS: comet server, 主要负责tcp连接的维持和消息的转发  
 LS: logic server, 主要负责业务逻辑处理, 设备上下线、消息发送、应用配置、消息ack等  
 SS: soa server, 对内提供soa api  
 Worker: async server, 处理一些需要异步的操作, 消息发送、消息ack、超时重发等.

## 符合“优雅降级”的推送系统设计

优先建立长连，可控可靠的通道

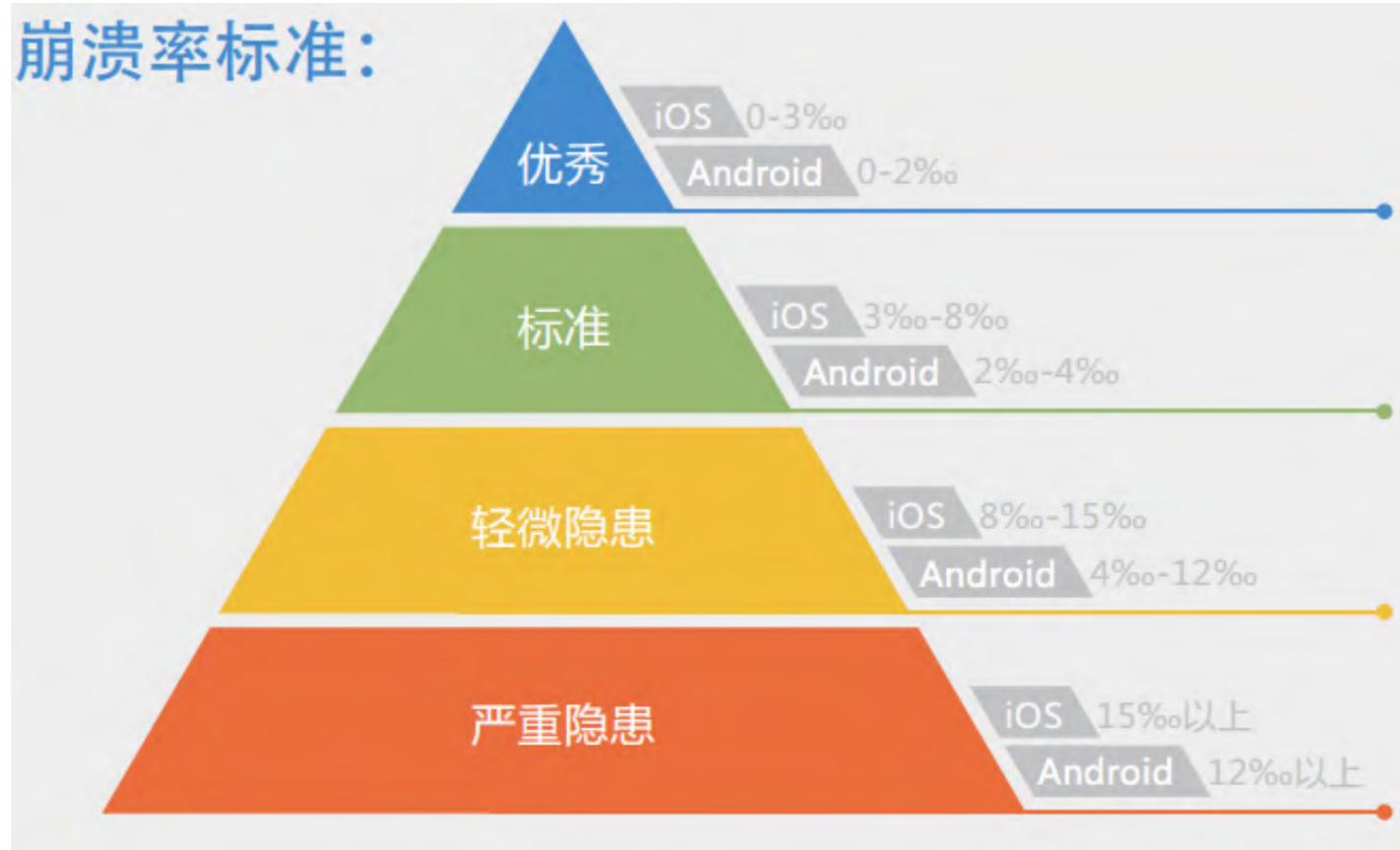
长连通道异常，降级到三方推送

单个三方通道异常，多通道自动

稳 => 快 => 省，普适法则

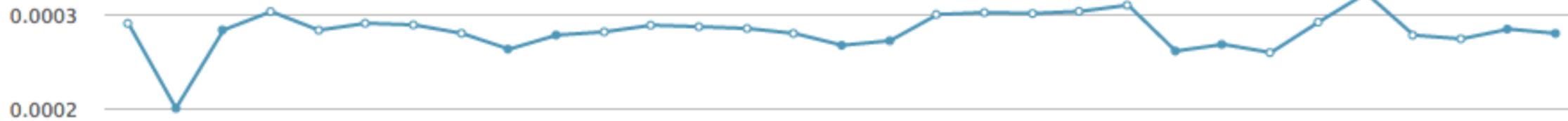
- 0 崩溃 & 0 错误 != 好用
- 启动时间：main() 后比 main() 前重要
- 包体积优化：二进制 > 资源
- 耗电优化：硬件 > 软件

# 0 崩溃 & 0 错误 != 好用

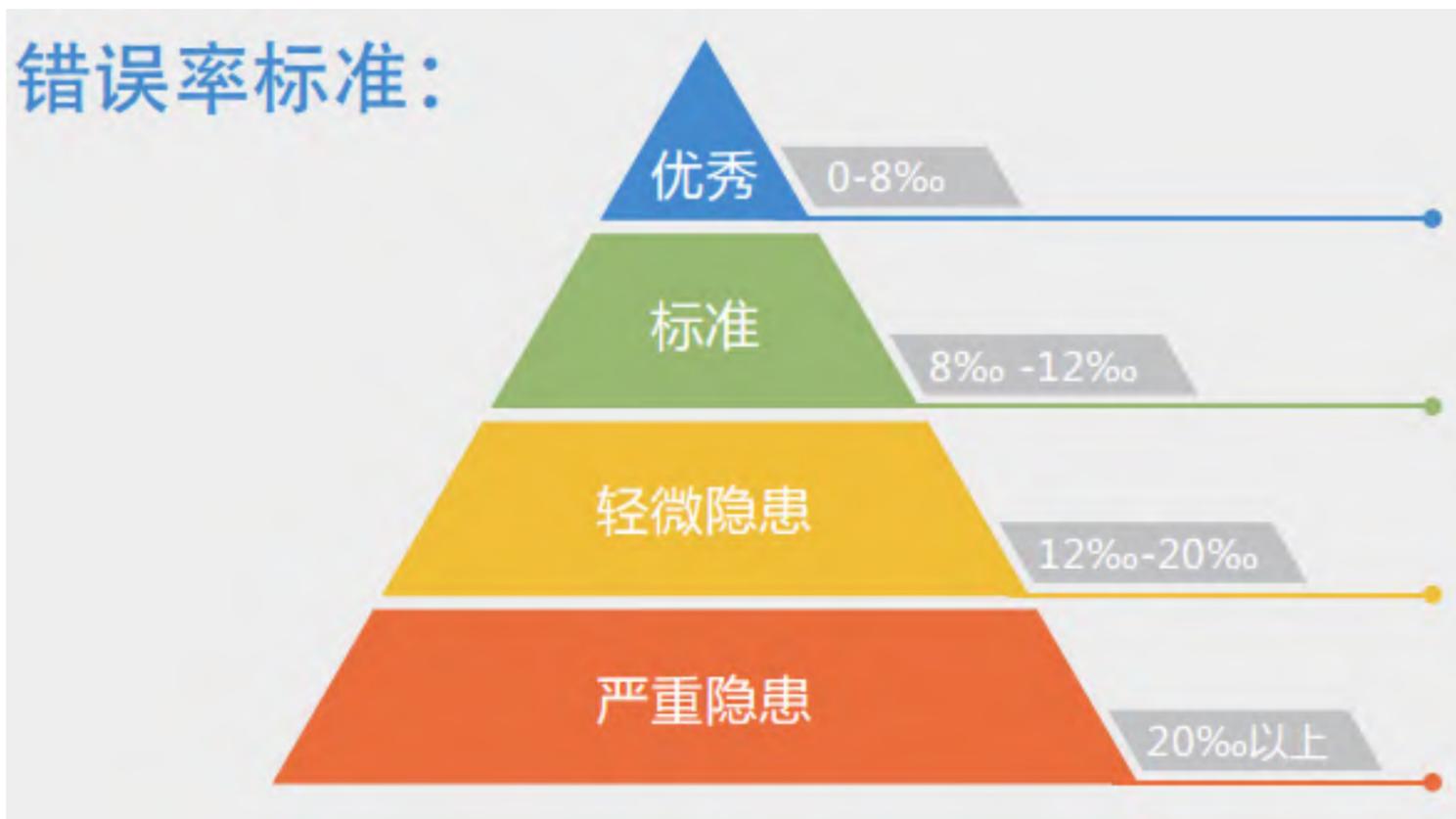


## 追求 0 崩溃带来的消极问题

- DP 的中庸问题
- 是否有效的 DFF



# 0 崩溃 & 0 错误 != 好用

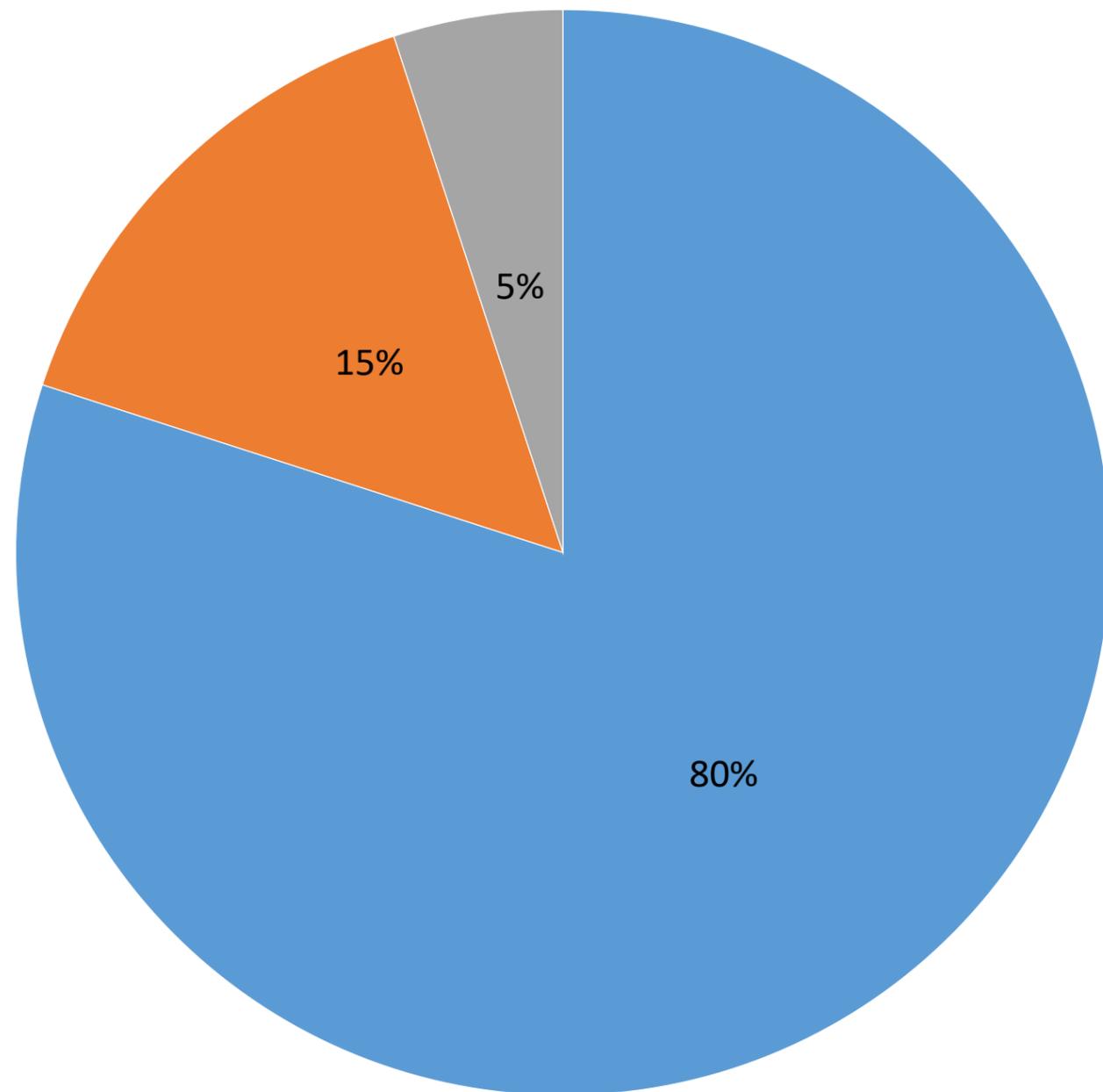


2017-03-08 10:52:09	2017-03-08 10:52:10	502		2/65449350/<0.01%
2017-03-07 18:38:19	2017-03-10 14:38:49	504		26437/34285221/0.07%
2017-03-07 19:17:05	2017-03-10 10:59:28	404		51/296913/0.01%

## 启动时间：main() 后比 main() 前重要

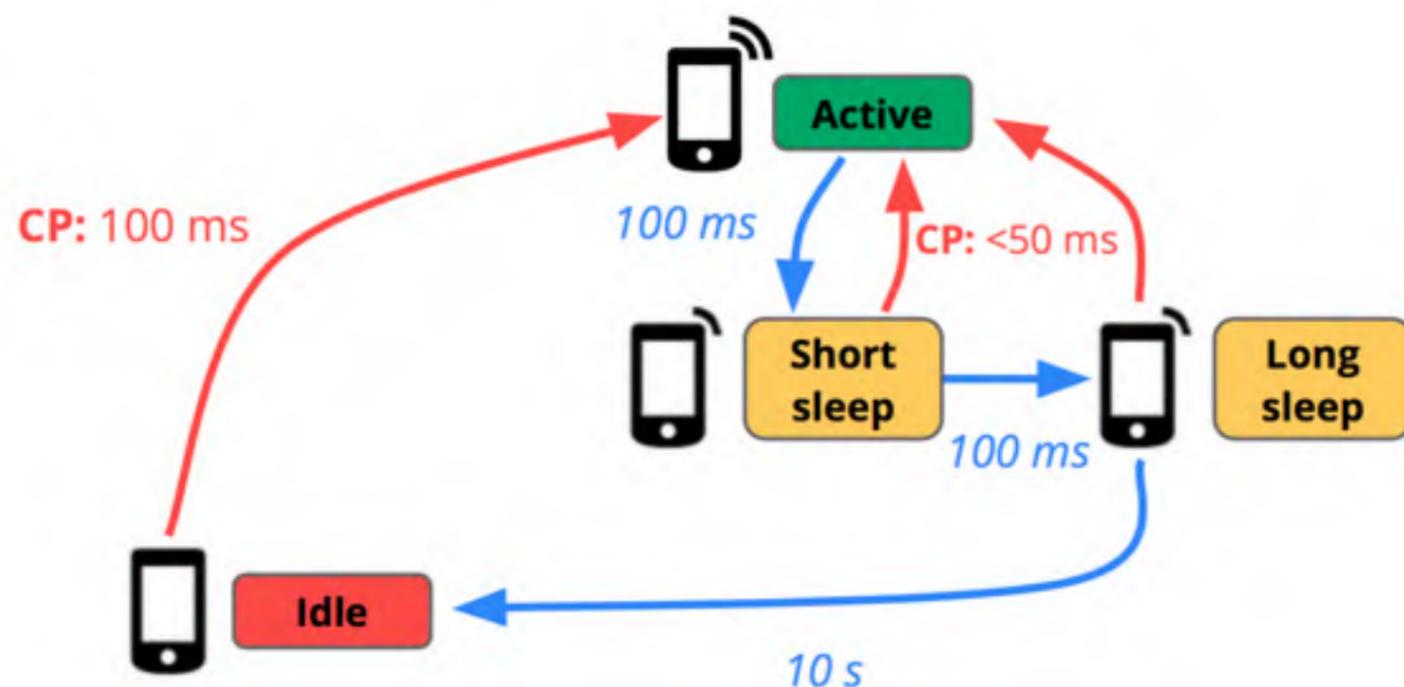
- main() 前优化点
  - dylib loading —— 大多为系统动态库，普遍使用静态库
  - rebase / binding —— 占比低，减少 Class 等行为违反软件工程原则
  - Objc Setup —— 受工程量影响，盲目优化违反工程原则
  - Initializer —— + load 优化，影响工程设计
- main() 后优化点
  - 首屏渲染优化
  - 避免主线程阻塞
  - 关键路径线程优化

## 包体积优化：二进制 > 资源



- 各类文件比重
  - 二进制 70% ~ 90%
  - 资源 5% ~ 20%
  - 其它 5% ~ 10%
- 平台特殊性
  - OC 运行时
  - “胖”二进制文件
  - bitcode 优化

## 耗电优化：硬件 &gt; 软件



耗能模式转化频繁，降低耗电周期

- 弱网下的“节流”传输
- 数据压缩，安全快速
- 合理缓存，批量传输
- 多用“节流”，避免无意义的消耗

# 耗电优化：硬件 > 软件



**SDCC 2017 | 上海**

**互联网应用架构实战峰会**

**CSDN**

**Q & A**

SDCC 2017 | 上海

互联网应用架构实战峰会

CSDN

