阿里云ARMS小程序监控进阶之路

> 作者：付萌（慕扉）
> 创作日期：2019-05-31
> 注：本文为 FDCon2019第4届中国前端开发者千人峰会《阿里云ARMS小程序监控进阶之路》 分享内容，已经过数据脱敏处理，转载请注明出处。

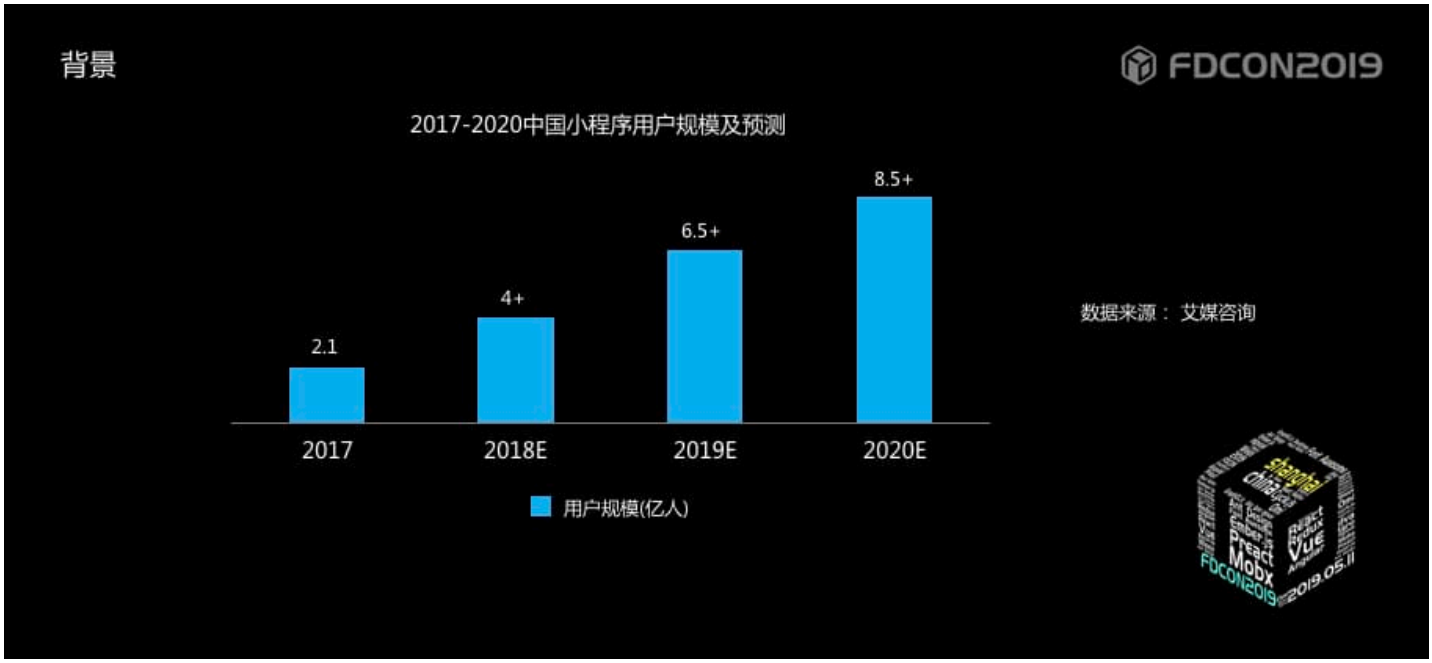
ARMS 是应用实时监控服务 (Application Real-Time Monitoring Service)的简称，是阿里云的一款 APM 类的监控产品。

小程序监控的现状及问题

小程序的发展及变化

从2017年到现在，腾讯、阿里、百度、头条等互联网公司相继推出了小程序，重点投入打造小程序发展生态圈。小程序已然从最初的基础元年开始走向大爆发，成为移动互联网的一个重要方向。

再来看一组数据，是艾媒咨询在2018年底的统计数据，其中2018年小程序用户迅猛增长，用户规模超4亿，预计2020年将超8.5亿人，基本实现手机网民的全覆盖。



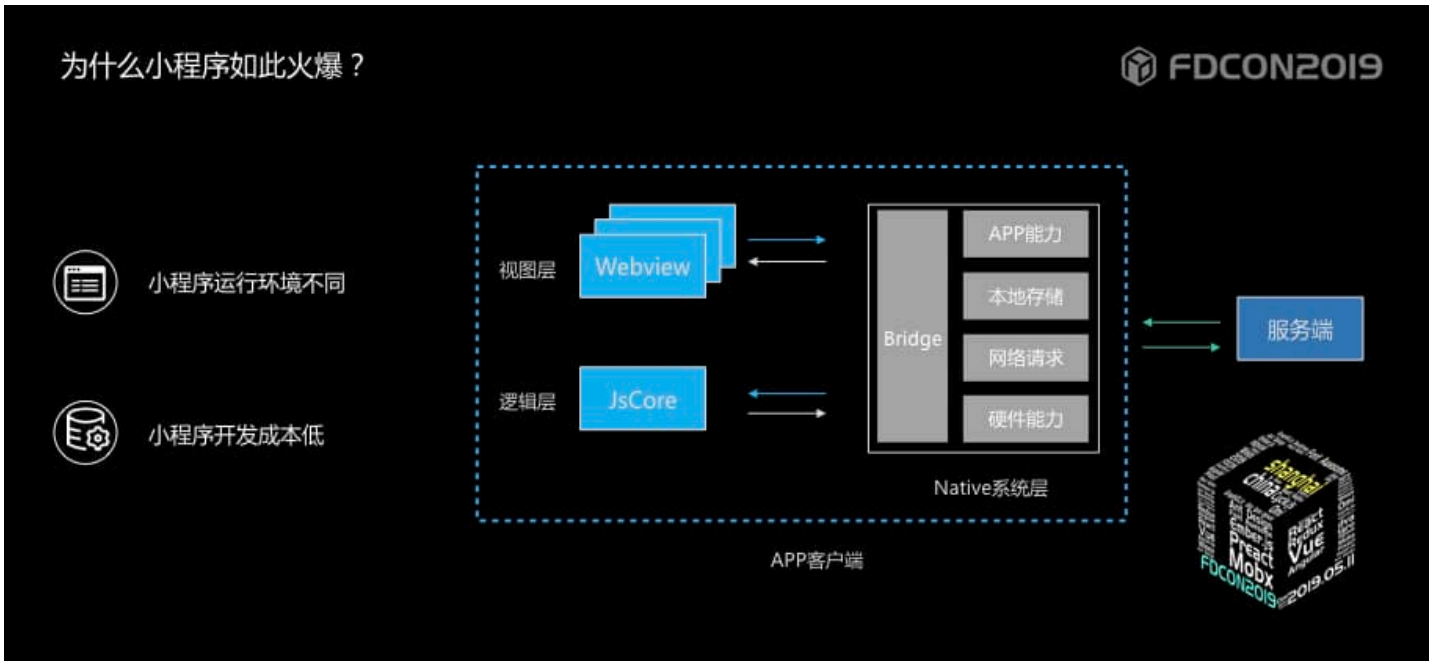
通过这两组内容，我们可以看到小程序发展是非常迅速的。那么小程序为什么会如此火爆呢?

这里抛出2个因素，当然也和大会的分享主题相关：

1. 运行环境不同，有望解决信息孤岛问题 我们都知道在移动互联网时代，每个APP都是一个孤岛，APP之

间不能相互跳转，但小程序可以使用APP开放出来的能力，完成小程序之间的自由跳转。

2. 小程序开发相对于APP开发成本低且易于维护，对于中小企业也能承受起 小程序是运行在APP客户端上的，逻辑层和视图层是两个独立的线程，通过APP提供的Native系统层转发来完成相互通信。这样一方面在APP层面可以管控和安全，同时双线程可以不用担心抢占资源而导致页面卡顿等问题，从图中可以看到，小程序底层框架将APP相关native能力进行封装，对外透出的仍然是我们前端开发的三板斧：js、css、html，但需要符合相应平台的规范。



小程序监控的现状及问题

小程序有这么多好处及优势，但对于小程序开发同学依然绕不过一个问题，那就是针对线上问题如何及时发现、定位、解决，减少影响用户数。作为开发者，不可能24小时都在电脑前等待用户反馈解决问题，我们需要有一个系统能收集用户的使用信息，辅助问题的定位与修复，减少开发人员的负担，那么小程序监控就非常有必要了。

和web监控、weex监控一样，小程序监控也属于前端监控的一个场景，但由于运行环境的不同，不能直接复用web监控的能力。

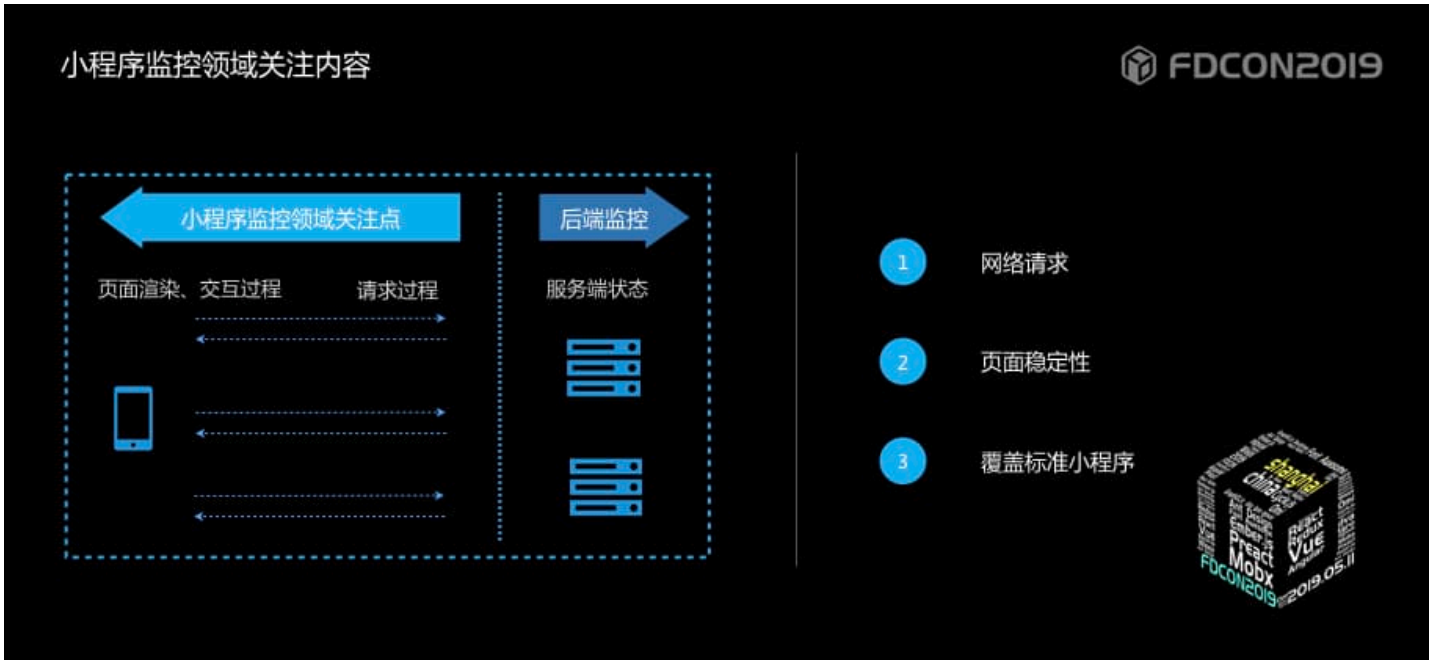
现在业界提供的小程序监控方案大概分为以下3类：

1. 用户数据监控 例如：新增用户数、访问次数、用户留存率等数据，主要用来助力产品运营，不能帮助开发人员定位解决线上问题。
2. 错误监控 这种情况下的监控数据较为单一，缺乏网络请求及性能相关的数据
3. 性能监控 该类监控大部分只支持指定类型的小程序监控，比如我新开发了一个某某小程序，却发现没有一个监控系统支持接入，那就非常尴尬了，处于一个无监控可用的状态。

整体来看，现在的小程序的监控体系并不成熟，所以我们希望能完善小程序的监控体系。

阿里云ARMS小程序监控SDK进阶之路

下图是用户打开一个小程序后的整个过程的示意图。从页面请求、到页面渲染、交互的过程。



作为开发同学，会更关注： 1. 网络请求，因为这些直接影响小程序页面上数据及交互情况。 2. 页面报错，相信也是每个小程序开发同学都会特别关注的内容。 3. 解决部分小程序无监控可用的现状，覆盖所有标准小程序。

当然还有很多其他关注的内容，在本次分享中会重点针对这三部分来看我们的解决方案及衍进过程。

网络请求

所有的线上故障最直接的反应是在页面上，所以大家的第一反应都是前端有问题，但很多情况下并不是前端的问题，相信大家都有感触。所以我们需要有监控来看是哪里出了问题。

发现问题：API成功率

第一步，我们要提供API的请求成功率，来确认是前端的问题还是后端的问题。通过hack的方式获取到请求发起的状态及请求返回后的状态，从而计算到API的成功率及耗时情况。

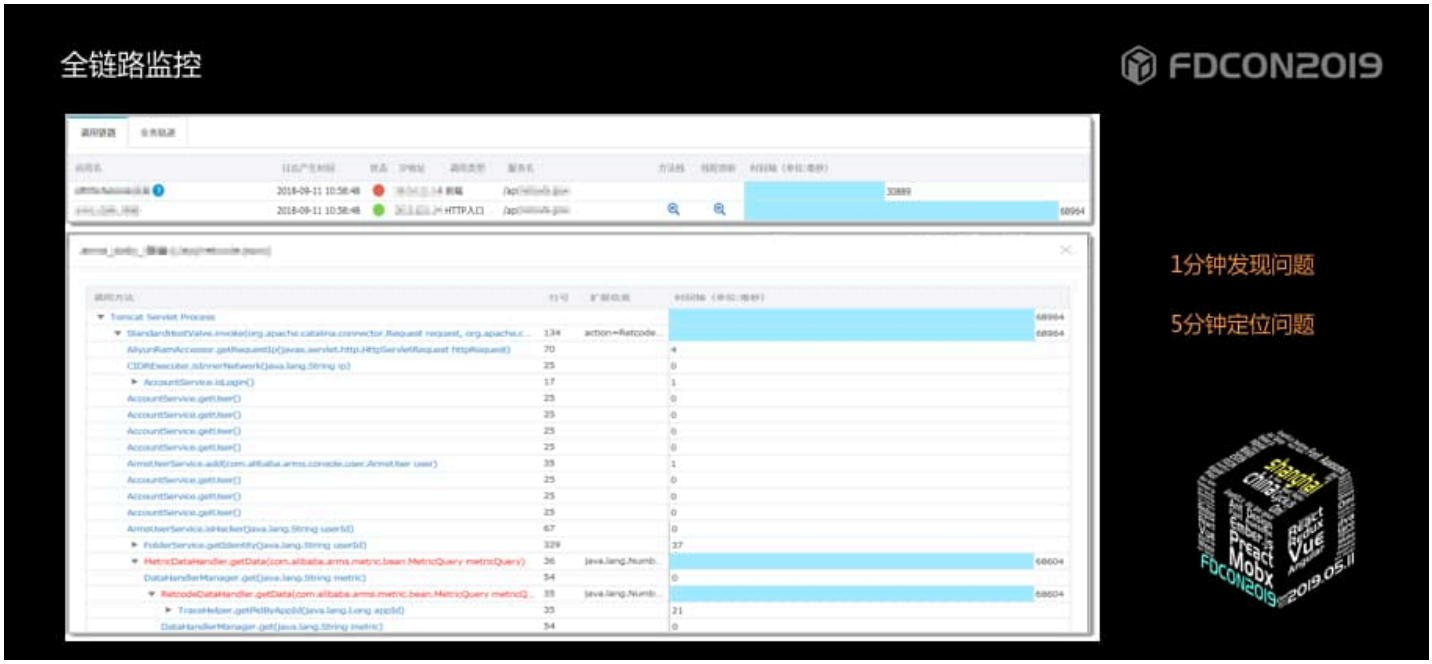
通过这部分数据，可以确认问题了，但不能帮助定位和解决这个问题。比如某API请求的成功率从最初的99%左右直接降到了60%左右，这肯定是有问题的，我们会让后端同学去排查，但后端同学排查也是非常困难的，因为问题无法直接复现。到这里是不是遇到了瓶颈？必须要根据请求涉及到的所有链路一路排查下去？那要花费的时间就无法预估了，线上问题分秒必争。

针对这个问题怎么来解呢？看一下针对网络请求提供的第二个能力：全链路追踪。

定位问题：全链路追踪

我们会给到错误请求整个链路，即前端及后端的数据，然后提供后端的整个调用链路，红色的部分是有问题

的链路，这样后端同学就可以直接排查有问题的链路解决问题了。1分钟内发现问题，5分钟内能帮你快速定位到具体问题。

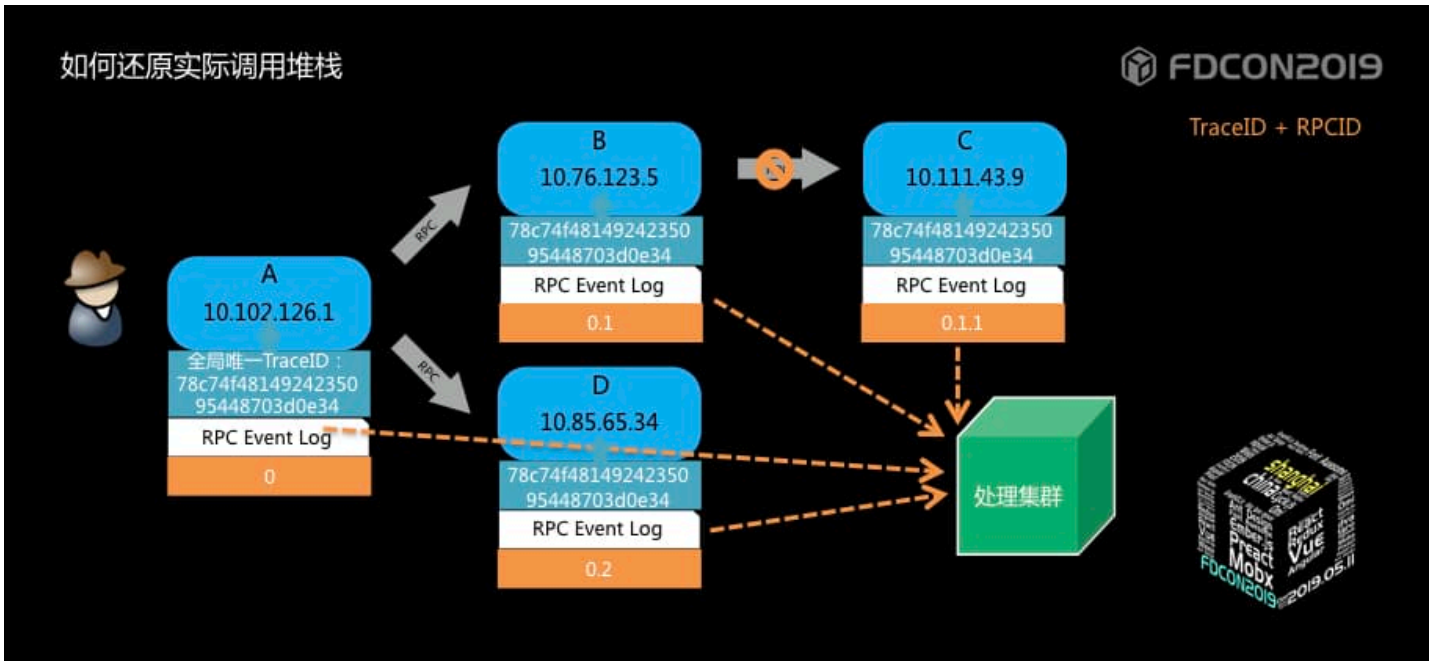


链路监控的原理

链路追踪的原理是基于基于谷歌开源的《Dapper, a Large-Scale Distributed Systems Tracing Infrastructure》论文，也是业界做链路追踪的理论基础。

在一次分布式调用链路中，如果C链路断掉了，我们要如何来发现呢？首先是在调用的源头生成一个全局唯一的TraceId，透传到后续的RPC链路，记录相应的事件日志，收集所有日志，处理，可以发现哪个RPC调用出现了问题。

分布式的调用链路，都有依赖关系，如何能够还原整个堆栈呢？那机制就是每个RPC根据链路的深度生成一个唯一的标识，通过标识的关系可以快速还原一次调用的堆栈。



根据这个原理，我们需要将一次请求的前端链路与后端链路串联起来，其中做全链路监控的挑战点如下：

1. TraceId如何透传
2. TraceId在整个链路的唯一性如何保证
3. 如何还原用户请求上下文

TraceId如何透传

方案一：通过后端将traceId透传到客户端

由于最初TraceId是后端RPC调用生成的，为保证TraceId的含义，可以将将TraceId透传到前端，从而串联起来。但这个方案有一个问题，那就是如果网络原因，接口超时了，那么TraceId就无法透传了，整个链路还是断开的，所以我们舍弃了这个方案。

方案二：从请求的源头生成TraceId透传到后端的整个链路

透传的方式可以有以下三种方式： - cookie机制：由于小程序运行环境原因，不提供cookie机制，所以这个方案行不通。 - param：在请求的URL参数中加入TraceId会破坏业务的请求原始URL，太突兀也不是最好的方式。 - request header：在请求头中加入TraceId的值，同时小程序请求不存在同源限制，所以最终选择了该方式。

TraceId在整个链路的唯一性

全链路追踪还需要保证一个点就是TraceId在整个链路的唯一性，否则就会导致链路内容不准确。

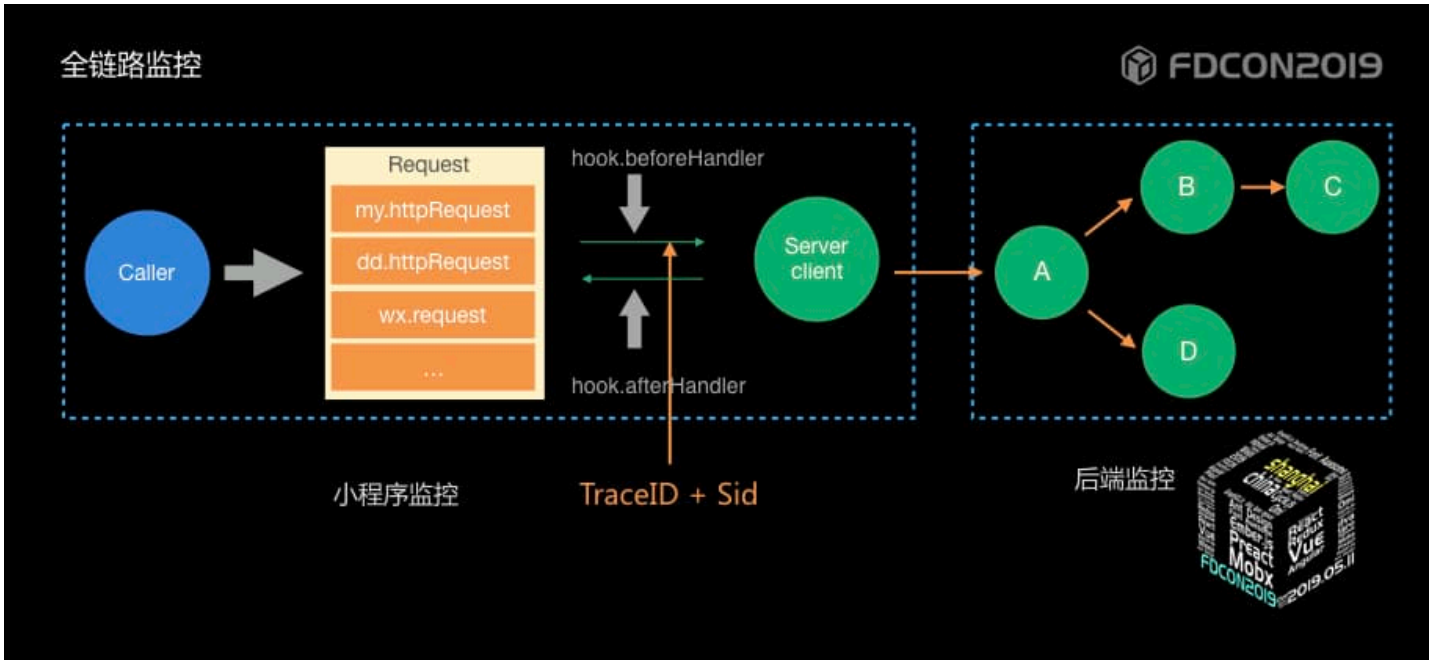
保证机制有2个： - 在TraceId生成源头利用生成规则保证唯一性； - 透传过程中，下游TraceId要与上游保持一致。

还原用户请求上下文

我们都知道用户在操作页面时，不同的操作顺序也有可能会触发不同的逻辑，从而导致的问题也会不一样。所以如何还原某次失败的链路的上下文场景呢?

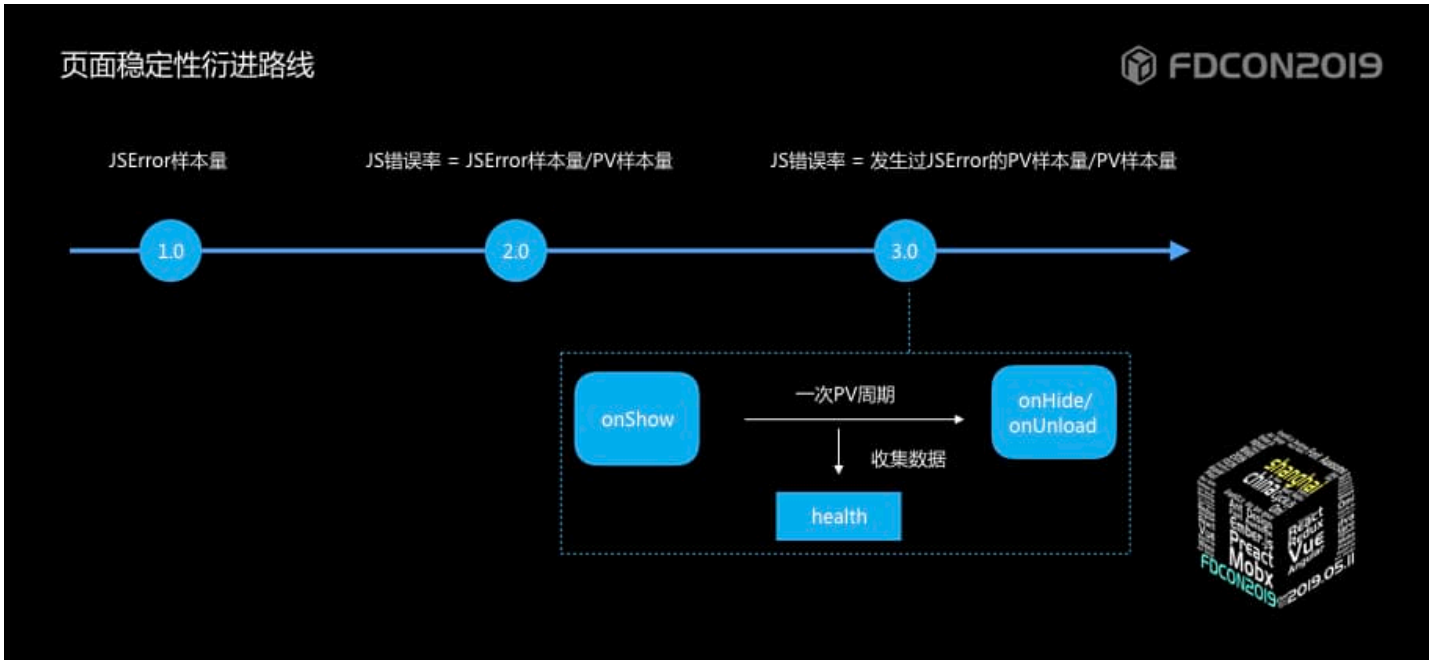
还原上下文肯定是通过时间先后顺序还原，但时间区间如何选择，时间区间长或者短都有可能无法帮助用户定位到问题。所以我们提出了一个方法，就是PV周期的概念，所有的有效操作基本上是在一次PV周期中的记做SID，通过获取一次PV周期内所有的traceId链路，根据时间顺序可以还原用户操作的一个上下文场景。

通过这三步衍进，形成最终的阿里云ARMS全链路追踪方案。



页面稳定性

页面稳定性的整个衍进路线如图:



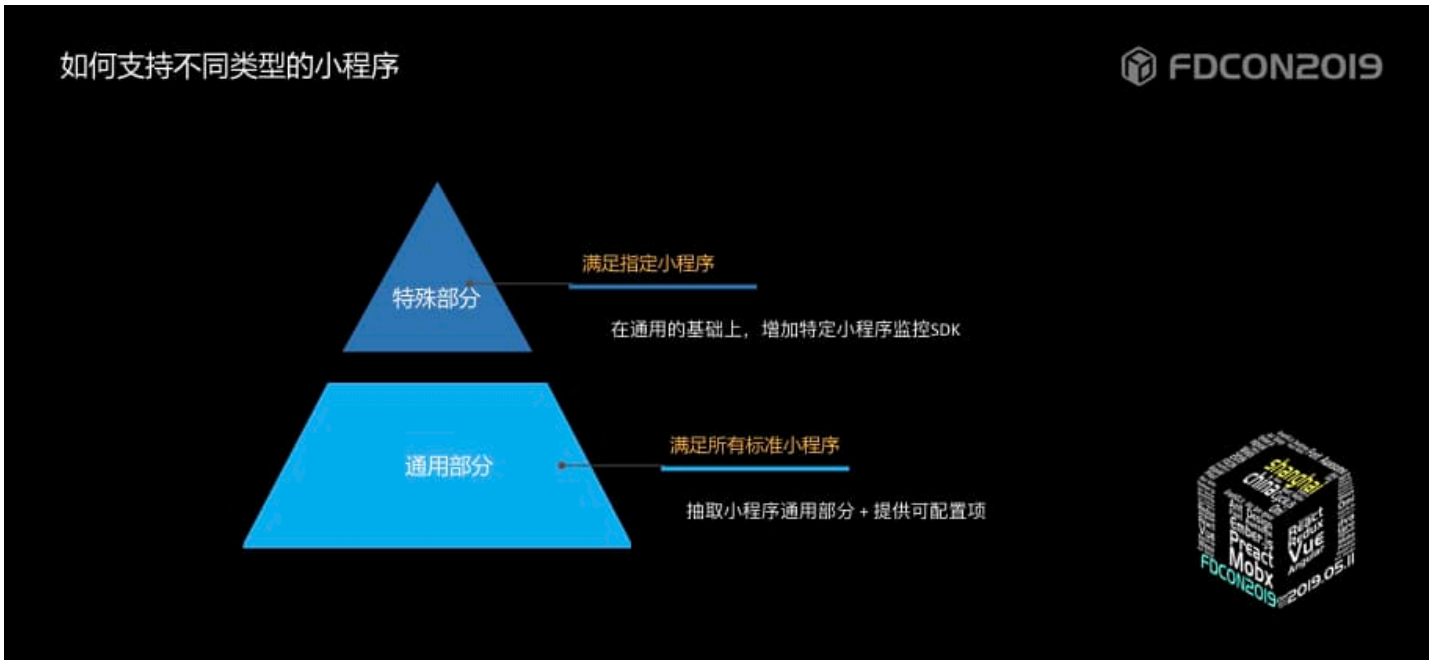
1. jserror的错误数 该方案存在一个问题，举一个栗子：场景一用户在一次访问时频繁触发js错误，场景二用户每次访问必触发js错误，两个场景上报的js错误数量一样，但影响程度却不同，所以直接的错误数是无法衡量影响程度的。
2. jserror错误率 = js错误样本 / PV样本 该方案可以解决1.0版本中单纯的统计jserror样本量存在的问题，但仍存在问题，就是该值会超出100%，值的大小无法和严重程度划等号。
3. jserror错误率 = 发生过js错误的PV样本 / 总的PV样本 通过该方案计算得到的jserror错误率值一定会<=100%，从而值的大小可衡量错误的影响程度。

在方案的衍进过程中涉及到了PV的采集原理，在传统的web监控中通过document的onload及unload等机制可以确定一次PV周期，但小程序运行在JScore中，没有该方案。

那么我们就要看一下小程序的一个生命周期了，我们发现Page页面无论是初次启动还是切换到前台，都会触发onshow事件，通过onUnload和onHide时间会切出该页面，所以我们的PV统计会根据该原理进行统计。

覆盖标准小程序

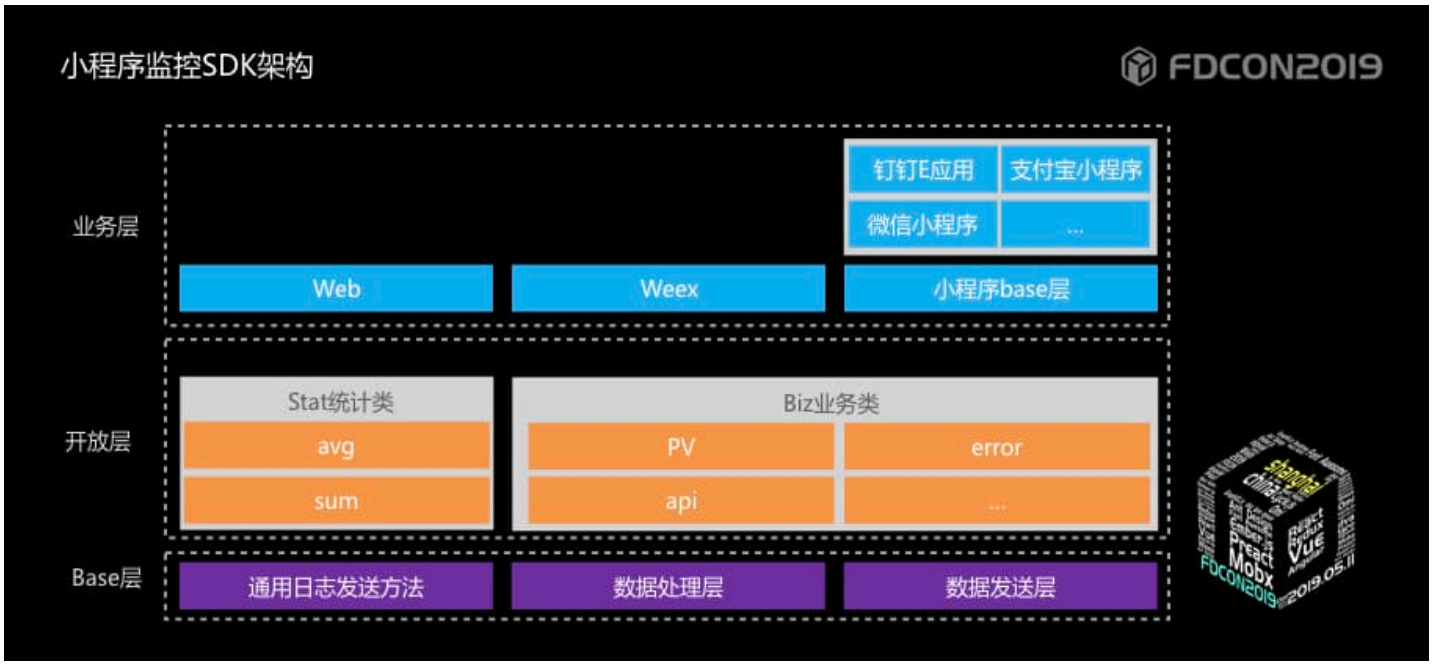
我们好不容易熬过了IE时代，webView时代，现在又进入了更加多元化的小程序时代，如何让所有小程序都有监控可用也是我们需要考虑的一个问题。



本着一个最基本的分层原则，将小程序通用的部分抽取出来作为小程序基础的base层，不同小程序的监控特殊内容基于base层做扩展，形成针对指定小程序的SDK。

也期待后续小程序发展可以走向标准及规范。

下图是阿里云ARMS小程序监控SDK的整体架构，可方便场景的快速扩展。

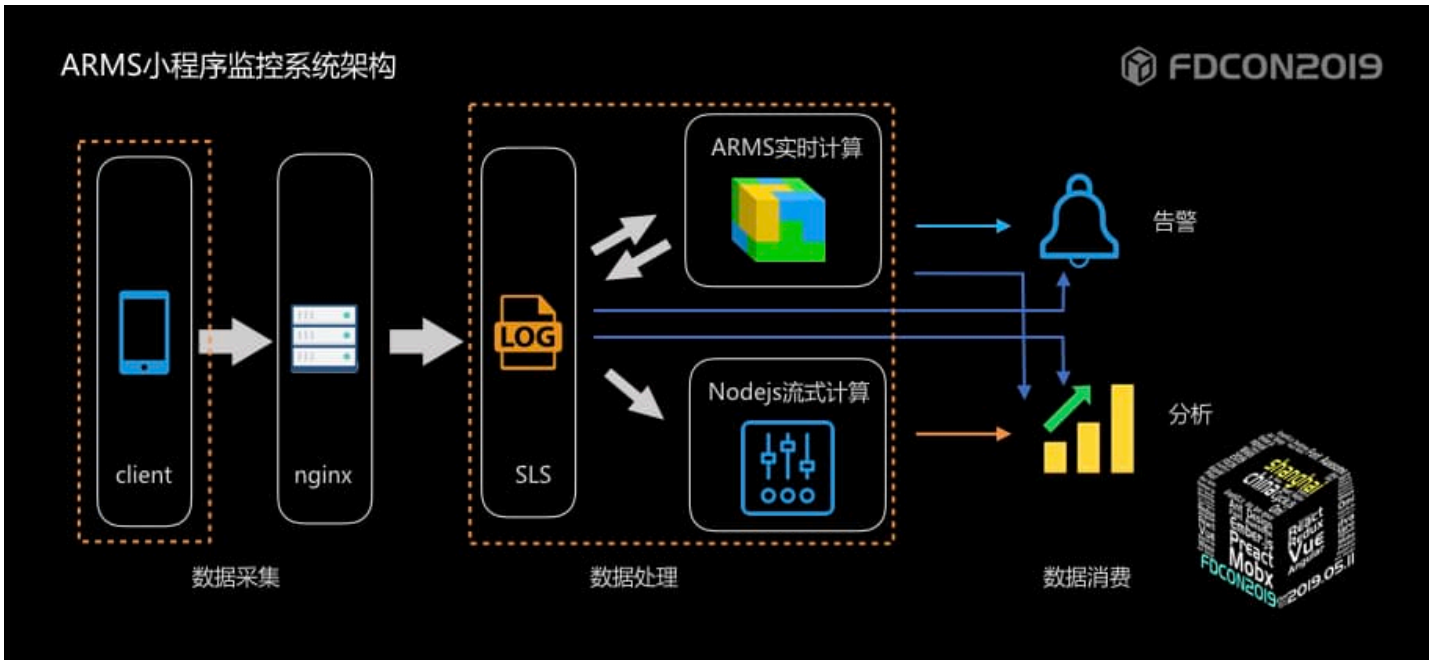


阿里云ARMS小程序监控系统设计

站在一个全局的角度来看，一个监控系统需要有：

1. 数据采集
2. 数据处理
3. 数据消费

一个好的监控系统，这三部分缺一不可。看起来比较简单，但当真正做一个监控系统时，会遇到很多问题。



针对数据处理部分，来看一下阿里云ARMS小程序监控的衍进路线。

1.ARMS实时计算

作为一个监控系统，数据的实时性是硬性要求，否则就难以达到监控的目的。所以在计算部分，我们有ARMS实时计算引擎帮助用户快速发现问题。实时计算的前提是要知道计算的规则，所以我们会设置通用的计算规则进行数据的统计。

实时计算如此强大，但针对监控系统也不是万能的，也会有他受限的地方。例如： - 实时计算后的数据均为统计后的数据了，没有原始日志无法对问题场景还原； - 计算规则提前设置； - 实时计算的维度统计不能无限制增加，会加大计算的压力；

2.阿里云日志服务SLS

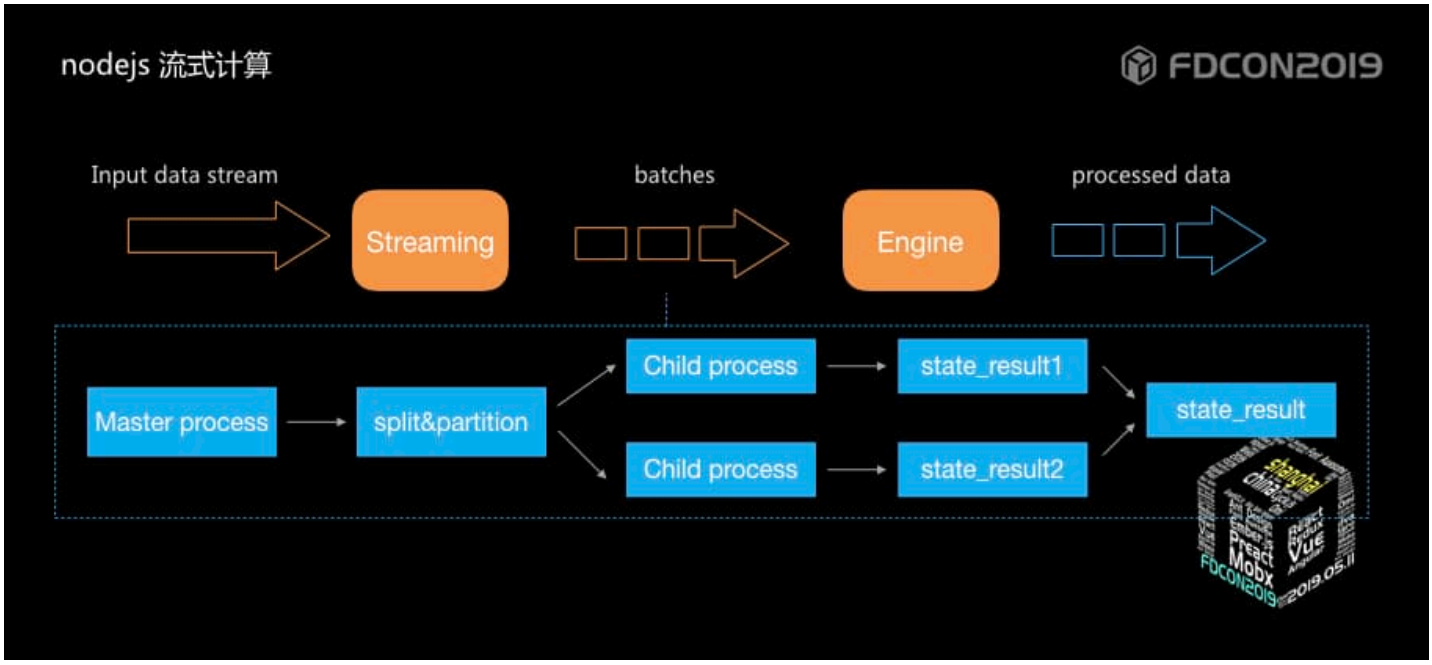
针对ARMS实时计算受限的情况，加入了阿里云日志服务，一方面可以做源日志的存储，方便问题的定位；一方面可以提供针对索引的快速查询及聚类分析。比如地理、设备、版本号、分辨率等等维度可以建立索引，通过日志服务完成聚类分析。减轻实时计算压力的同时也保证了用户多维诉求。

实时计算可帮助快速发现问题，阿里云日志服务可以帮助定位问题，这样看，实时计算+阿里云日志服务是完美的配合，但用户的诉求还远不止于此。

3.Nodejs流式计算

比如：根据某业务想针对某个特殊指标进行数据统计，业务定制诉求对于实时性要求并不高，主要帮助做业务决策。针对这种情况，在实时计算中增加计算规则并不合算，因为并不是所有业务都要这个逻辑，同时在日志服务中存储的原始日志量级较大，存储的时间也不宜太长等问题，我们需要有一个能定制化处理用户诉求的引擎，所以加入了Nodejs流式计算引擎。

主要过程如下：



读取数据流，将数据流拆分成多个小的数据流，其中每个小的数据流是一个独立的计算单元，利用计算引擎的多进程处理，每次处理都会维护一个状态表，最终将结果合并处理作为该数据流的一个统计结果。由于在定制化处理，维护的是用户提交的计算逻辑，所以整个的处理流程不涉及多张表之间的jion、sort等复杂的计算，计算的速度也是非常快的。

在数据处理部分，我们通过实时计算、日志服务、nodejs流式计算不断的加入，满足不同用户对于数据的诉求，也意在挖掘出数据的更大价值，帮助用户更好的发现、定位、解决问题。

由于篇幅限制，部分内容不能展开来分享，有兴趣的同学可以加入我们的钉钉技术交流群。

附录

- 阿里云业务实时监控服务ARMS
- 阿里云业务实时监控服务ARMS前端监控
- 小程序监控接入文档

加入我们

【稳定大于一切】打造国内稳定性领域知识库，**让无法解决的问题少一点点，让世界的确定性多一点点。**

- GitHub 地址
- 钉钉群号：23179349
- 如果阅读本文有所收获，欢迎分享给身边的朋友，期待更多同学的加入！