QCon 全球软件开发大会
INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

BEIJING 2017

# 电商广告计费系统容灾设计

美丽联合集团-子文

主办方 Geekbang. 极客邦科技 InfoQ

# 关于我

- 姓名:卢家雄，花名:子文
- 岗位:广告业务团队管理与架构
- 公司:蘑菇街到美丽联合

部门技术公众号

# 目录 content


- 一、计费系统概述
- 二、计费容灾设计
- 三、未来展望
- 四、Q&A

# 01

# 计费系统概述

# 概述-计费模式

- 怎么计费？

**CPC (Cost Per Click )**

**按点击计费的商业产品，对于电商, 常用于站内广告资源位，用于推广商品**

**CPS (Cost Per Sale )**

**按成交计费的商业产品，站外长尾流量，用于推广商品 / 店铺**

**CPM (Cost Per Mille，或者Cost Per Thousand;Cost Per Impressions )**

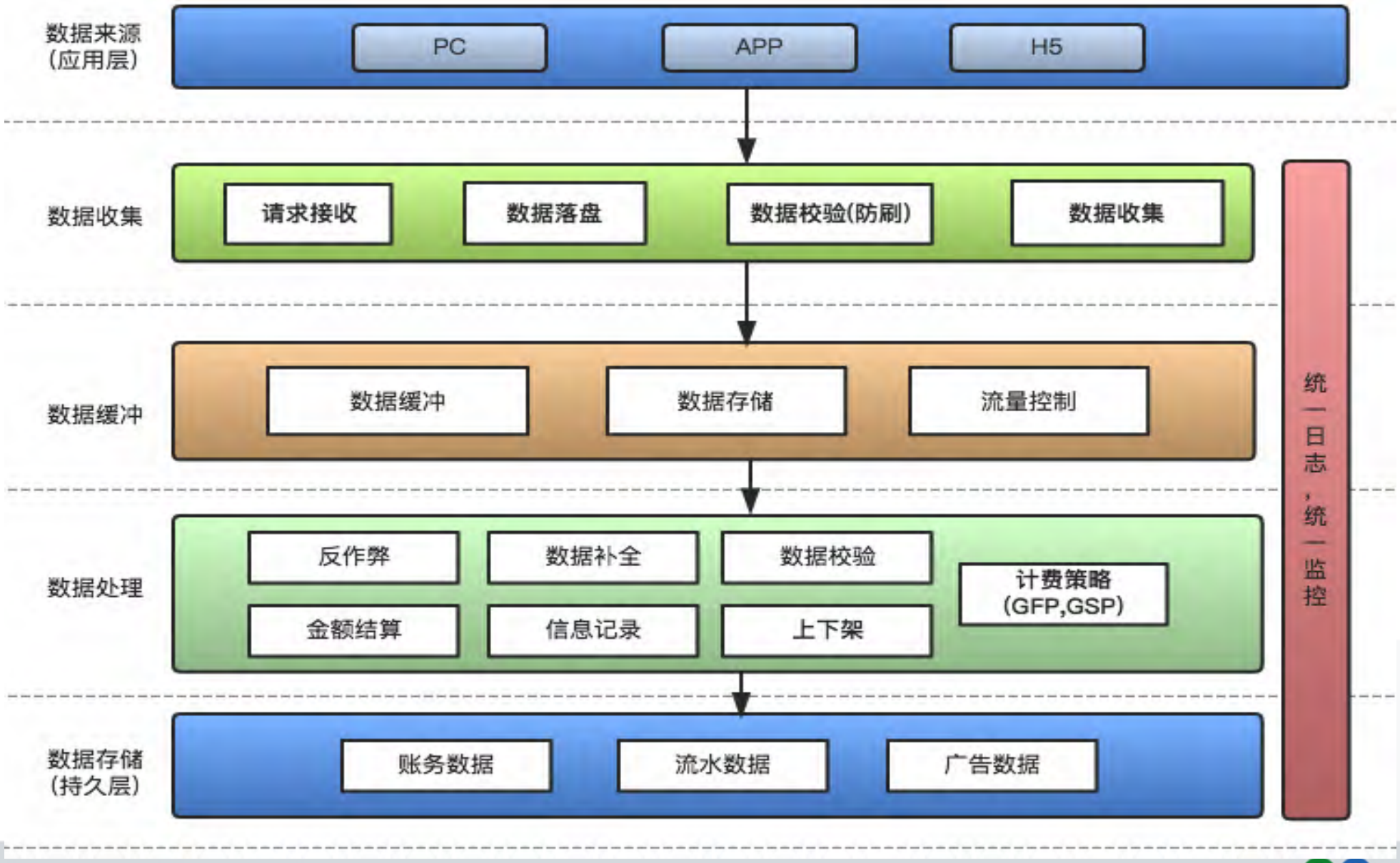
**按曝光计费的商业产品，站内banner位等资源，用于推广品牌店铺**

**其他常用模式：**

**CPD，CPT，**

**CPA….**

# 概述-计费系统框架




怎样的数据流？

# 概述-面临的核心问题

- 实时数据是否准确？
- 异常时数据怎么回溯？
- 整个链路是否出现数据丢失？

- 未来数据增长后如何实现快速扩容？
- 热点数据存在不均衡现象,怎么办？


- 计费系统的稳定性,可用性怎么保证？
- 系统抖动出现扣费问题,平台损失,商家账务问题怎么办？

- 数据是否安全,是否被篡改/被盗刷,遇到常用的DDOS等攻击怎么处理？
- 被攻击的情况下快速恢复数据？

- 如何做到不停服升级及AB？

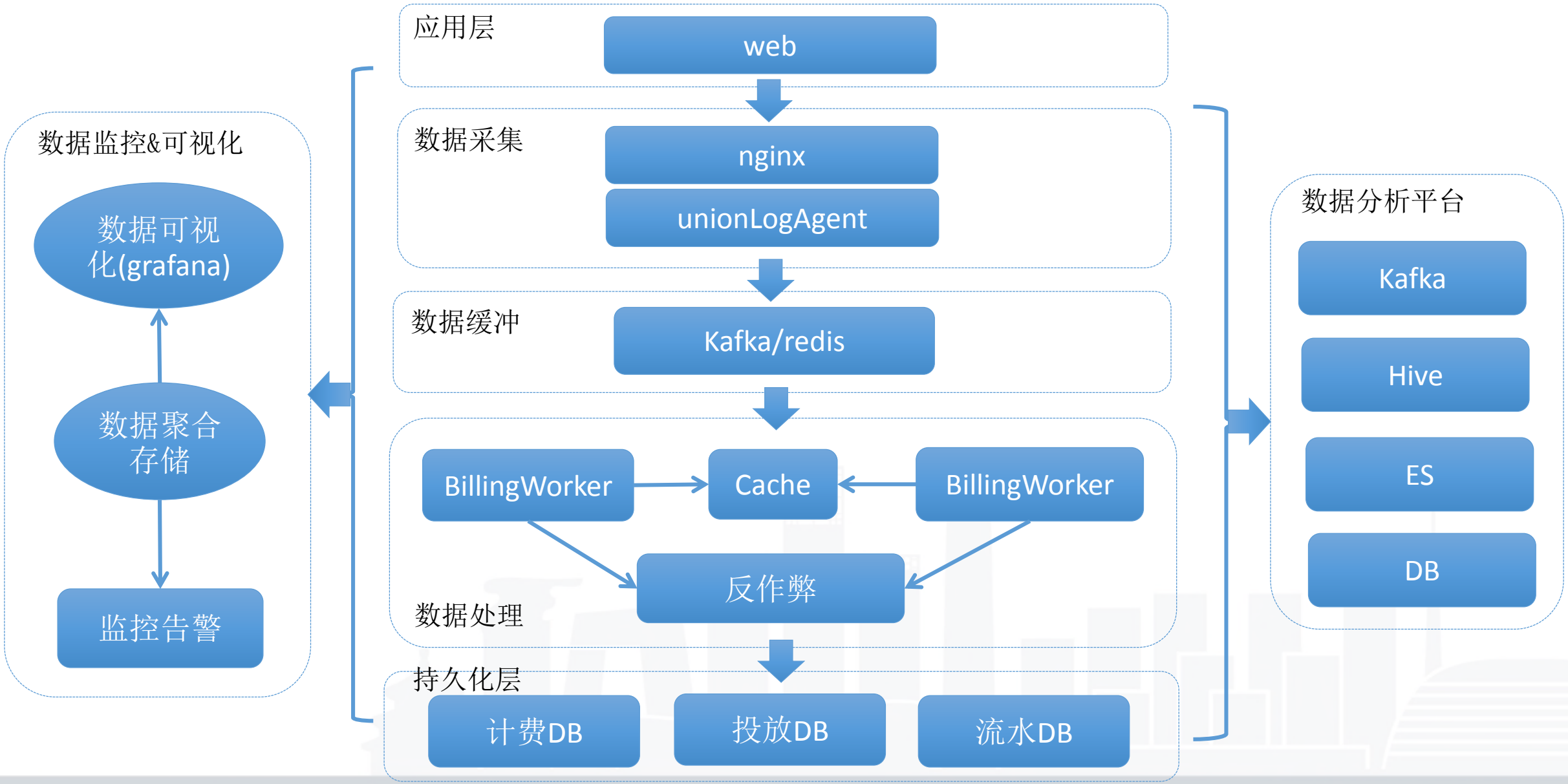
- 第三方依赖出问题，如何在保证不会发生雪崩及快速恢复？
- 如何避免强依赖，如何能快速监控发现问题？

02

# 计费容灾设计

数据/系统/链路完整性/监控

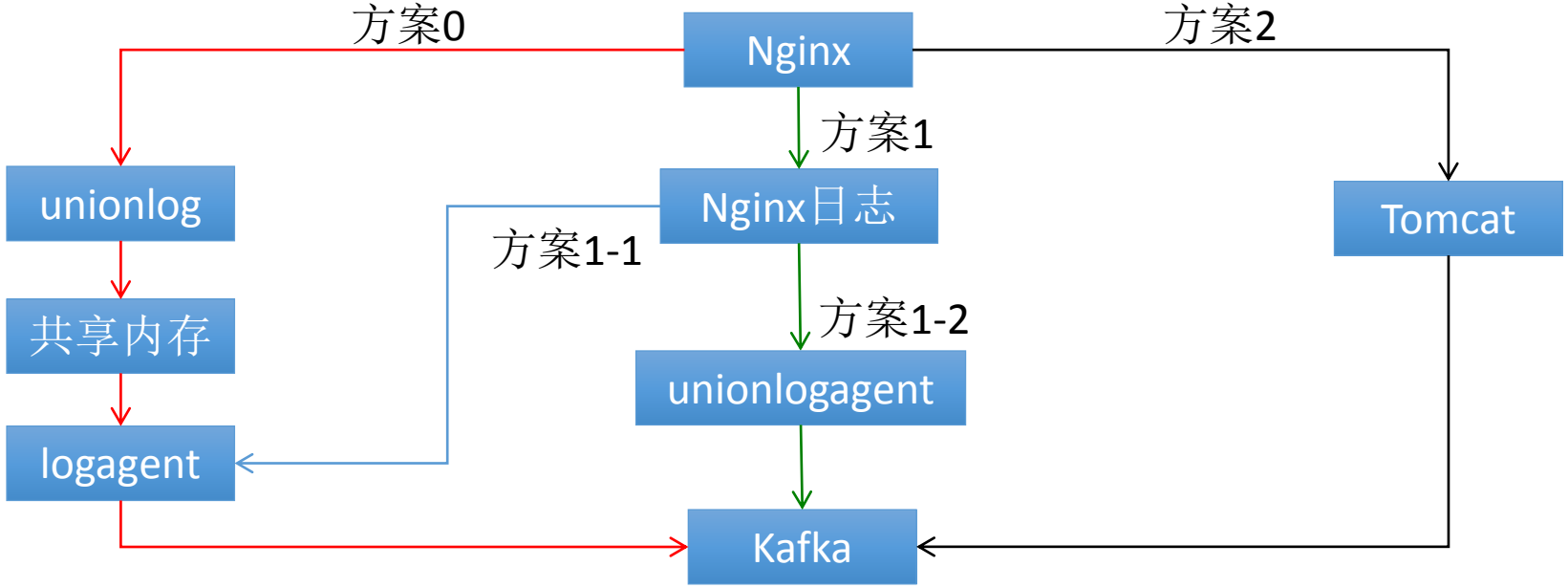
# 一、整体架构


应用层
web
数据采集
nginx
unionLogAgent
数据缓冲
Kafka/redis
BillingWorker
Cache
BillingWorker
反作弊
数据处理
持久化层
计费DB
投放DB
流水DB
数据监控&可视化
数据可视化(grafana)
数据聚合存储
监控告警
数据分析平台
Kafka
Hive
ES
DB

# 数据流面临问题：

- 恶意流量攻击, 异常数据对系统的冲击
- 数据回溯, 快速定位失败数据
- 链路数据的一致性, 数据丢失或者重复

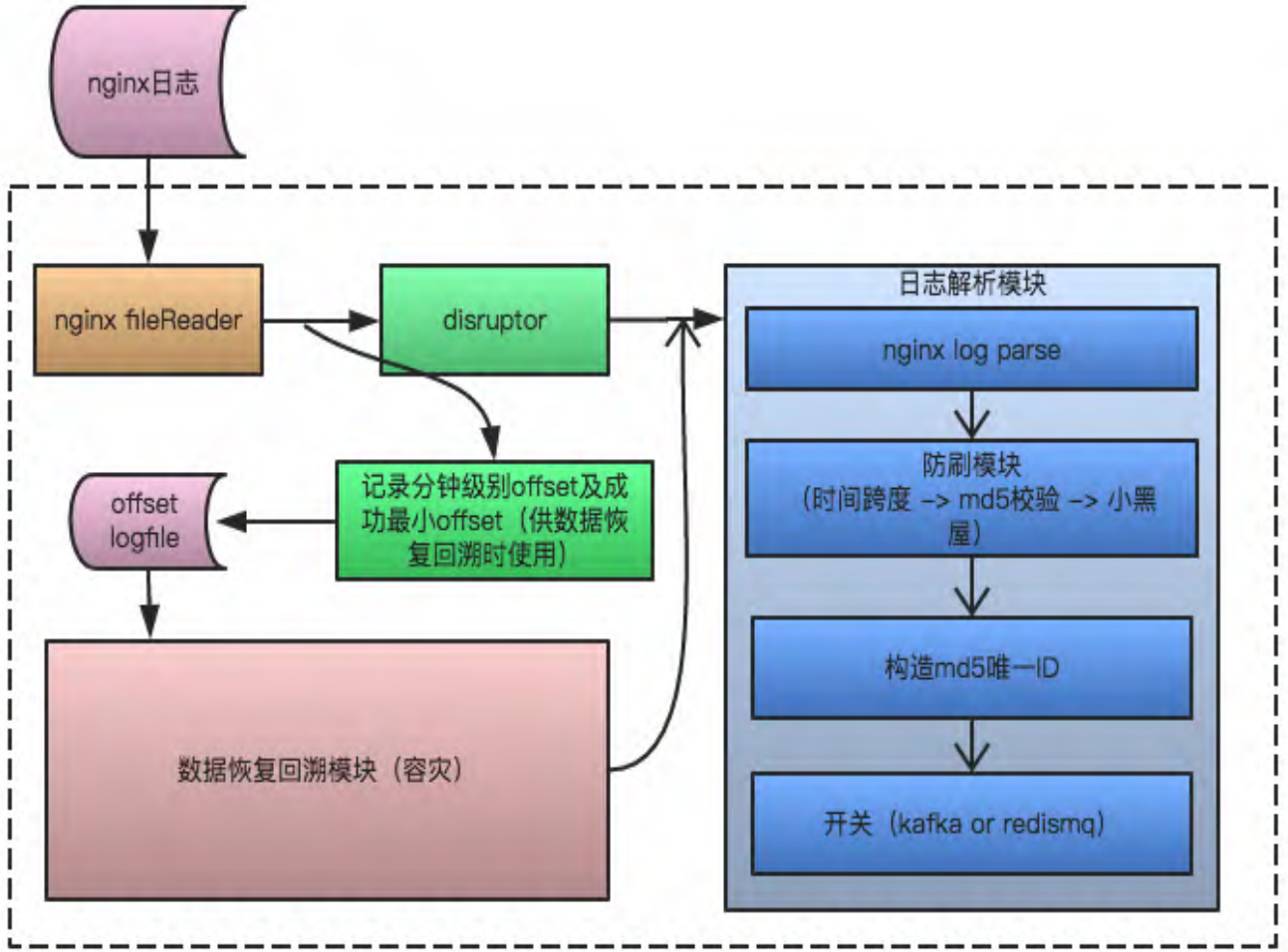
# 1、数据流-数据采集



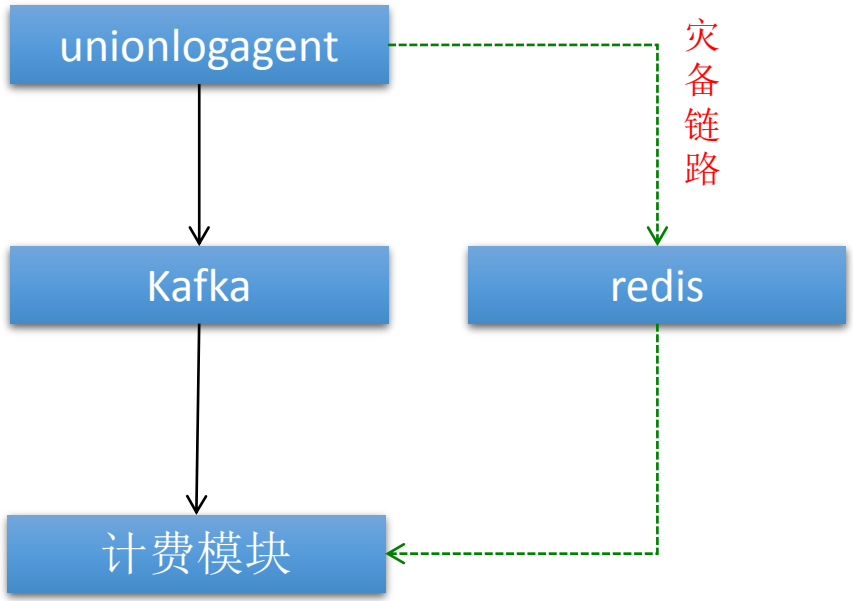

| 方案0<br>原方案 | 方案1 | | 方案2 |
|---|---|---|---|
| | 1—1 | 1—2 | |
| ☐ 无kafka客户端，导致链路长<br>☐ nginx+php，单机性能差 | ☐ logagent无法定制 | ☐ unionlogagent可以定制，如防刷、按userId分区、分流 | ☐ 标准化部署、发布<br>☐ 环境重<br>☐ 耦合 |
| | ☐ 解耦<br>☐ 单机性能高<br>☐ 环境轻 | | |

# 1、数据流-数据采集

- 日志落盘与收集解耦，减少依赖影响
- 可定制化和快速扩展的防刷模块
  直接将异常数据上层过滤,避免无用数据对下游冲击
- 分钟级别保存消费日志offset,快速定点及日志回溯
- 下游链路灾备切换,快速系统恢复



# 2、数据流-数据缓冲




- 消息队列冗余
- kafka的ack等级设置为all,保证异常情况不会出现消息丢失

**高吞吐量/低延迟**：
kafka每秒可以处理几十万条消息，它的延迟最低只有几毫秒

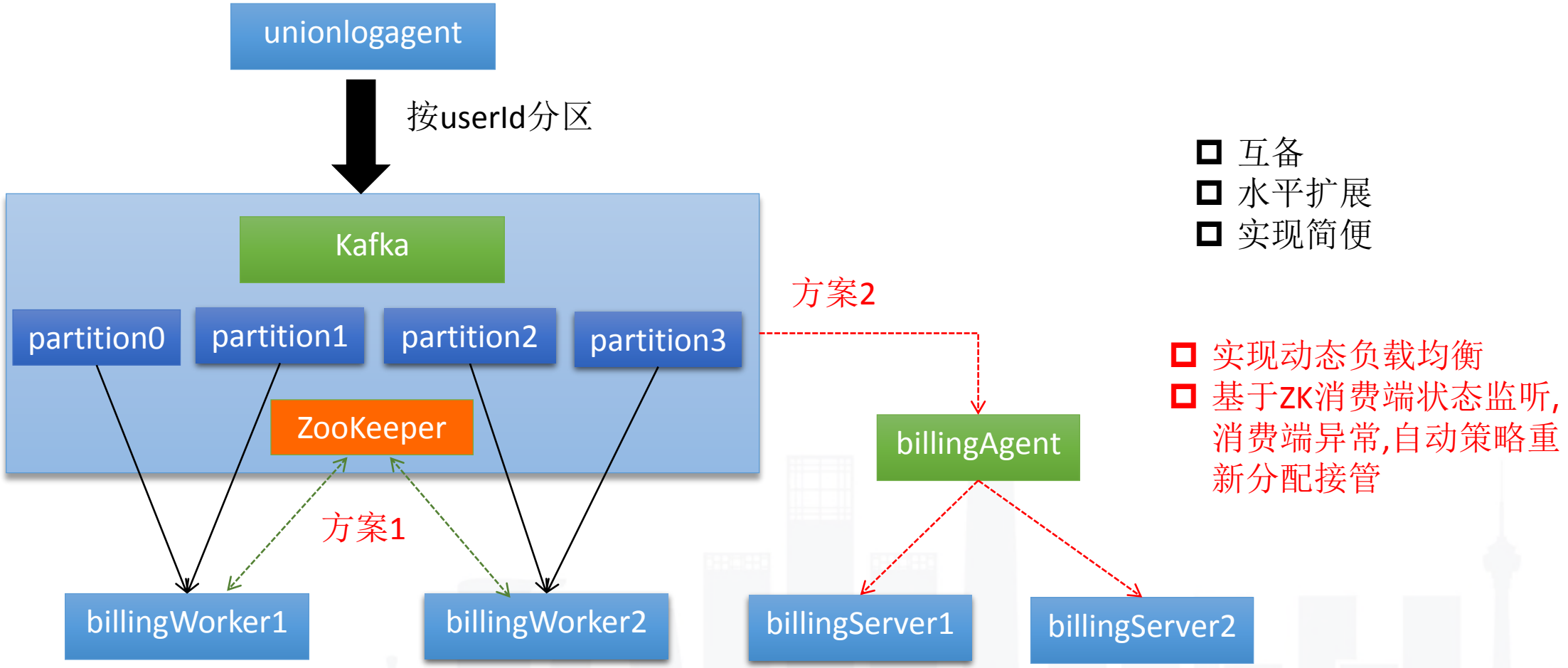
**可扩展性**：
kafka集群支持热扩展

**持久性/可靠性**：
消息被持久化到本地磁盘，并且支持数据备份防止数据丢失

**容错性**：
允许集群中节点失败（若副本数量为n,则允许n-1个节点失败）

**高并发**：
支持数千个客户端同时读写

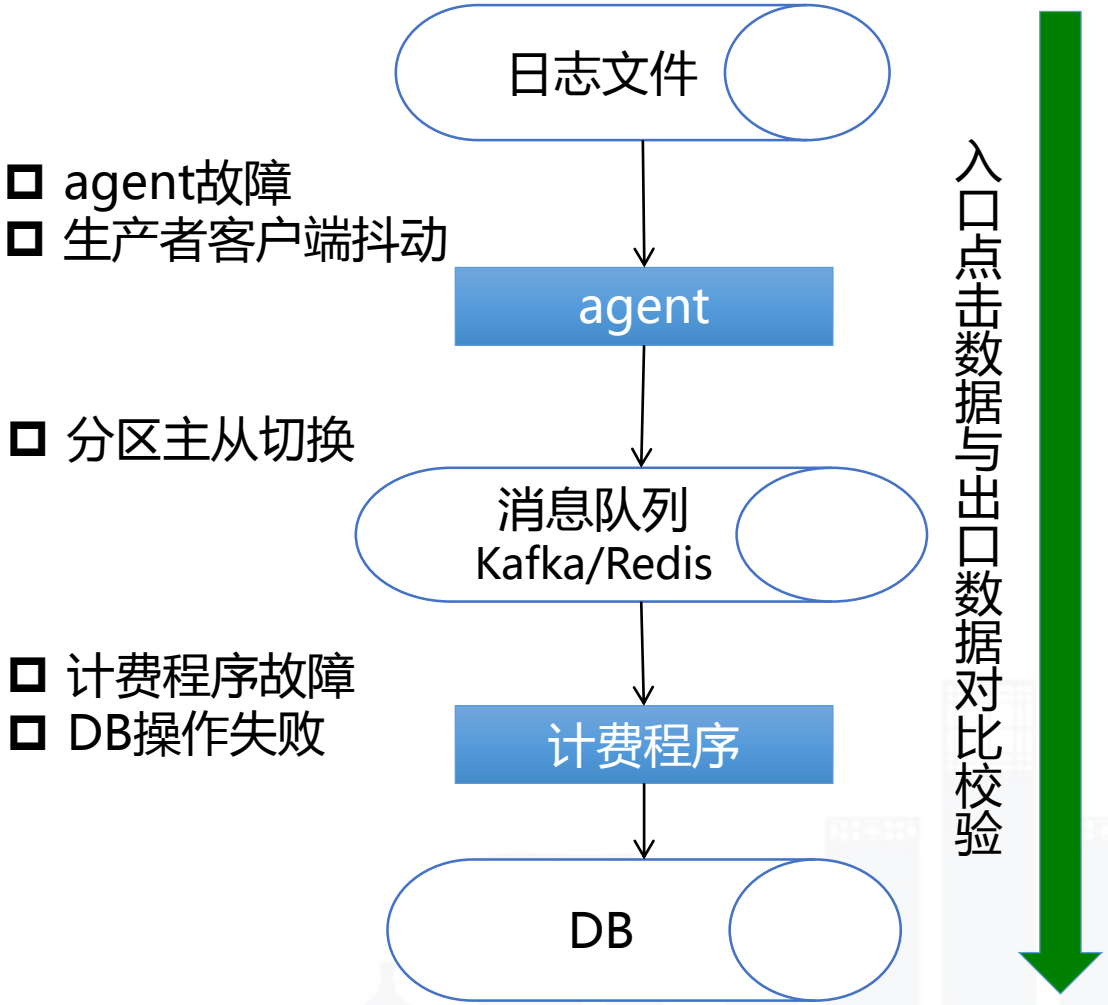
# 3、数据流-数据处理




- 互备
- 水平扩展
- 实现简便

- 实现动态负载均衡
- 基于ZK消费端状态监听，消费端异常，自动策略重新分配接管

# 4、数据一致性



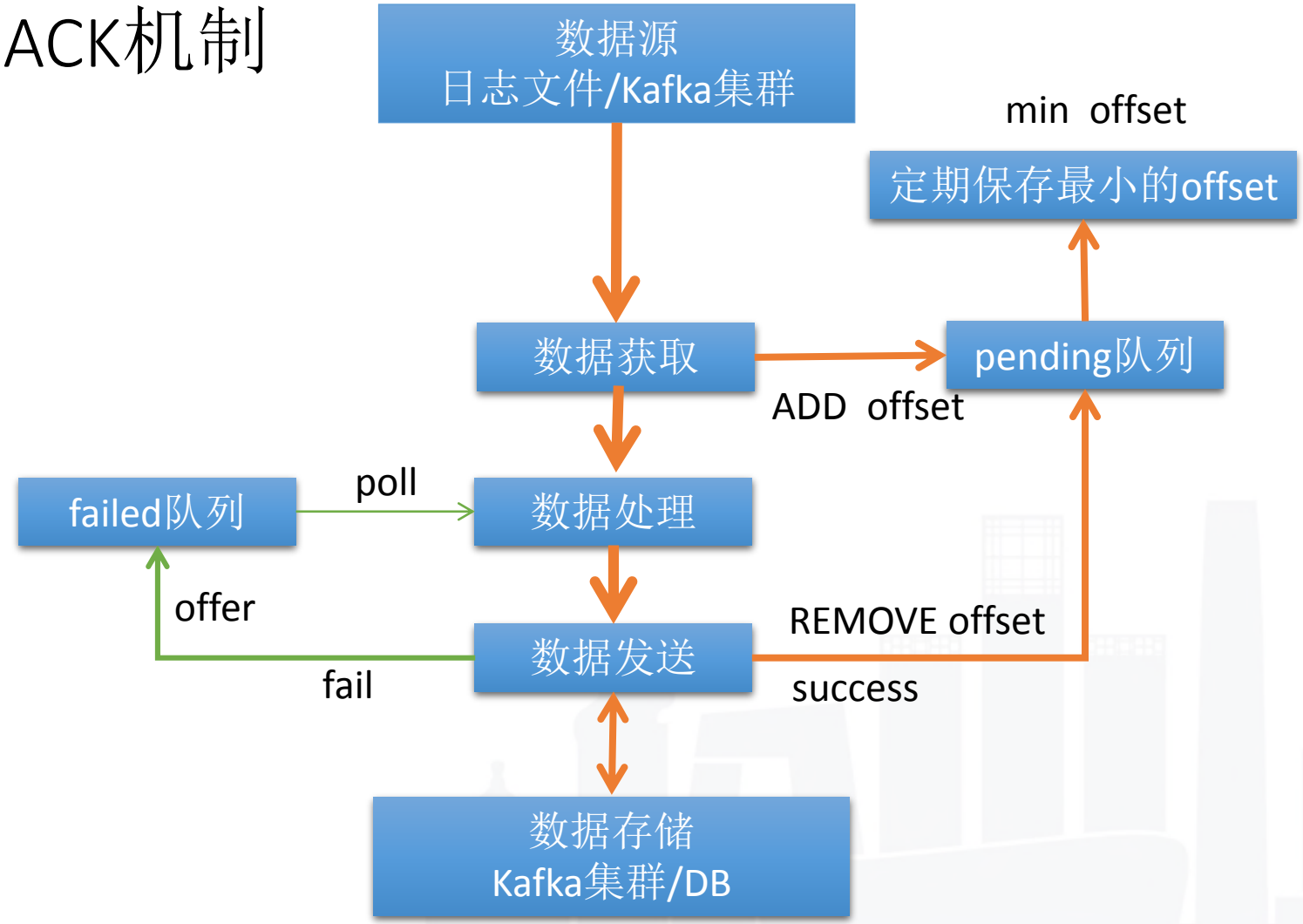

- agent故障
- 生产者客户端抖动
- 分区主从切换
- 计费程序故障
- DB操作失败

- 区部一致性→全局一致性
- unionlogagent、billingWorker中通过ack机制实现每个模块的数据一致性

- 依靠ACK机制自动恢复失败数据,无需人工参与

# 4、数据一致性(ACK机制)

ACK机制




- 最小成功offset持久化保存
- Fail队列重试
- 数据收集及数据处理阶段ACK机制,数据一致性保证

# 系统稳定性面临问题：

- 扣费时效性及热点数据处理
- 系统大流量的冲击，系统的雪崩
- 自身AB及不停服切换，平滑升级

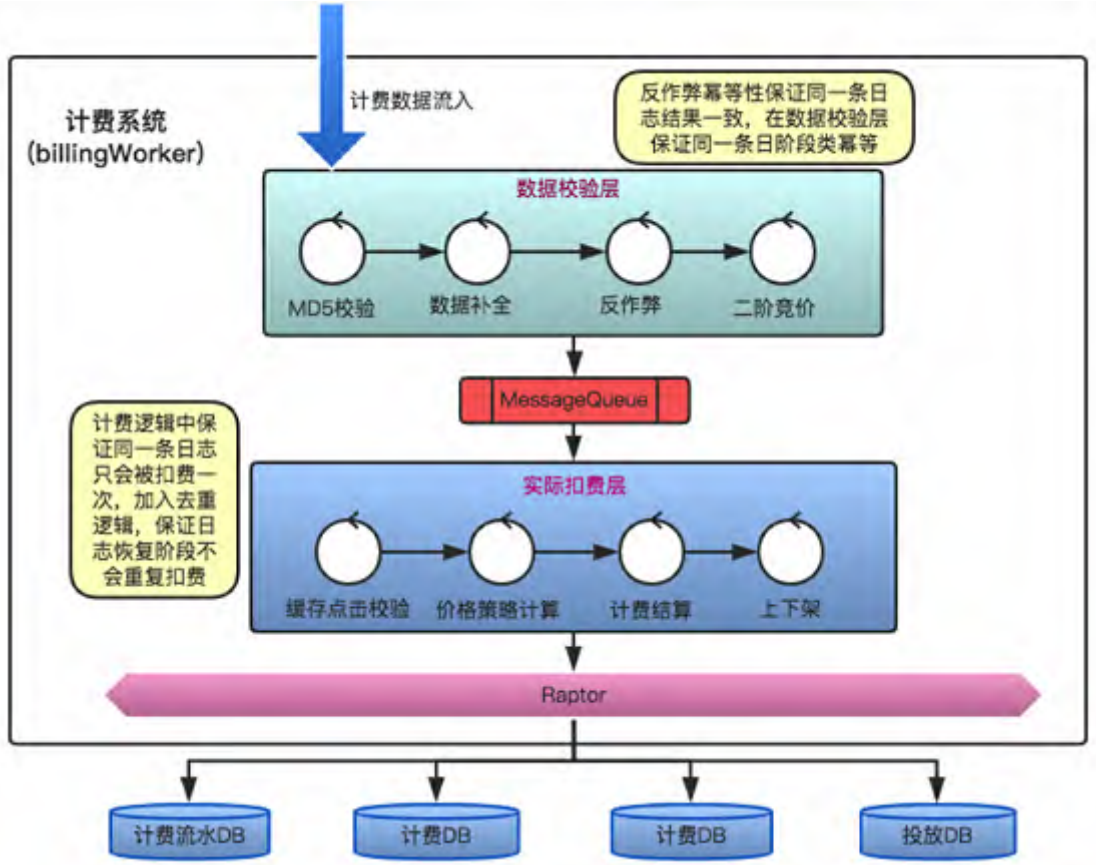
# 1、系统容灾-系统幂等性

扣费系统“幂等”功能，对于有状态环节，同一数据结果必须幂等，实现计费链路去重逻辑

- **日志生产端**：
  构造单条扣费日志唯一性主键（GUID）,保证每条日志都有自己的唯一标识

- **反作弊**：
  保证同一条日志结果一致,校验层同一条日志结果幂等，基于缓存保存每条反作弊的校验结果

- **计费**：
  保证同一条日志只会被扣费一次,加入去重逻辑，保证日志恢复阶段也不会重复扣费，DB唯一索引



# 2、系统容灾-扣费时效性

## 导致系统低效原因分析

1. 第三方依赖(DB,redis)链接及数据传输网络开销
2. 第三方逻辑耦合在主流程中,占用主链路的开销
3. 线程的频繁创建与销毁,频繁的上下文切换
4. 热点数据的分布不均匀,不能高效的利用CPU

## 解决方案

➢ 减少不必要的依赖：强依赖转弱依赖

1、第三方依赖异步化（kafka / sentry / 数据统计），保证主流程安全，同时提高主流程效率。
2、弱依赖埋好开关，在出现性能瓶颈时可以及时关闭。
3、连接池应用及网络部署优化

➢ 找到相对合适的上下文切换

CPU密集型：线程数＝2 * 核数
IO密集型：线程数＝核数 / （1-阻塞系数） 阻塞系数＝链路中IO占用的时间占比

# 3、系统容灾-扣费时效性(热点数据)

## 问题

实际投放过程中,存在热点商家或者热点流量，一种情况很多大商家全部被分到同一个线程队列中处理，则会导致某些线程阻塞而某些线程空闲，未能最大化并发效果，处理不好同时还有可能造成系统雪崩；

1. 过程中如果只是以特定的线程来处理一个队列，由于点击在商家维度的不均衡性，一定会出现某些队列堵塞。

2. 大队列堵塞会导致大量的缓存点击，会影响收入。

## 解决方案

- 分而治之

1. 按照userId的方式进行hash队列拆分，分拆合适粒度的队列，多队列并行，消费程序(billingWorker)分布式部署分配接管队列消费

2. 线程和队列分离,按队列数据长度，动态分配线程，提高热点数据消费效率.

# 4、系统容灾-限流降级方案

## 三把利器

**缓存：** 提升系统访问速度和增大系统能处理的容量，可谓是抗高并发流量的银弹
**降级：** 暂时屏蔽掉非核心流程,避免对主流程造成影响，待高峰或者问题解决后再打开
**限流：** 一个时间窗口内的的请求进行限速来保护系统
常见的限流算法有：令牌桶、漏桶、计数器也可以进行粗暴限流实现

## 三级限流保护

### 一级：接入层限流

数据总入口，根据入口流量及系统流量瓶颈，进行限流。

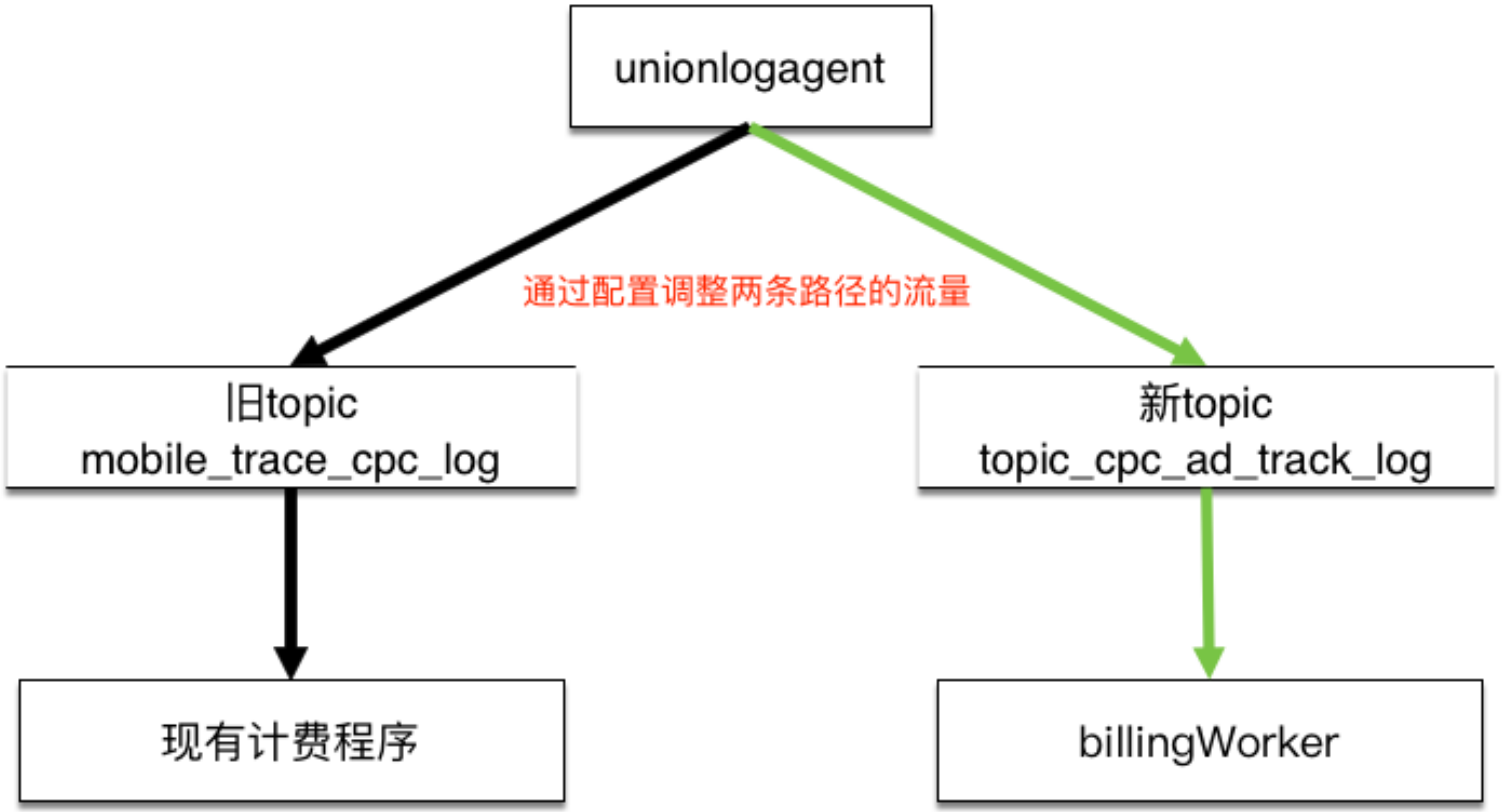
### 二级：系统级限流

程序本身计费逻辑限流，保证队列不会出现堵塞：在程序数据拉取过程中，判断计费队列中的消息数量，比如当数量大于某个阈值（10000）时，限制流量进入（如：当前队列数据10240个，在QPS 4500时计费队列平均10000个），同时控制总量，避免内存队列堆积大量数据。
当出现断电式故障时，可以快速确定丢失的数量，进行数据恢复。

### 三级：重试机制限流

程序拉取上游数据限流：当程序下游雪崩时，程序会出现大量异常,异常数据会存储入fail队列，计费程序将根据异常的出现次数,fail队列的长度，决定是否继续从上游获取数据、是否进行不进行重试，走fail队列消费逻辑进行数据恢复。

# 5、系统容灾-自身AB&平滑上线

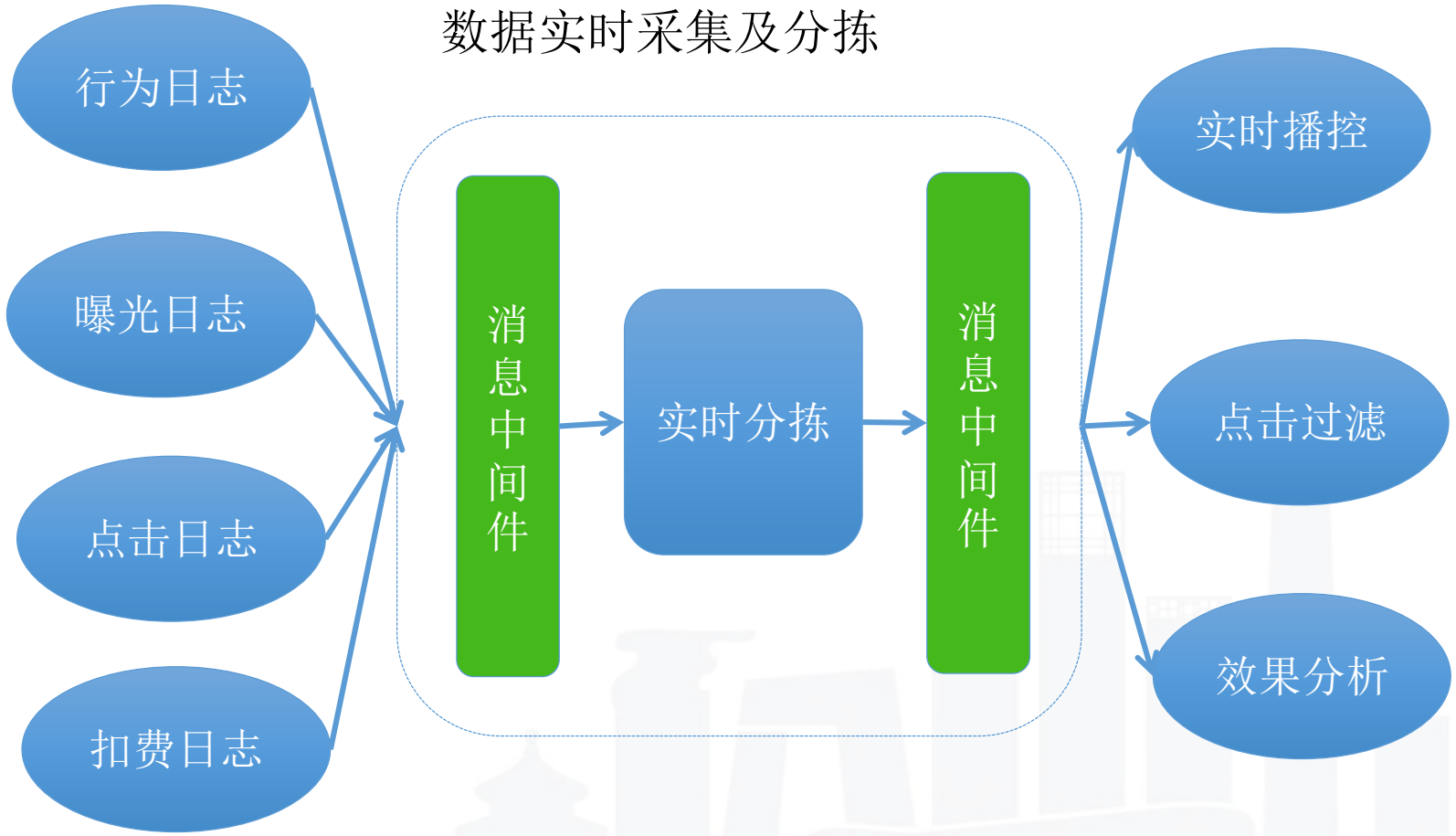



- 小流量平滑上线
- 自身AB及新业务计费模式测试验证
- 故障时可以快速切回

# 链路完整性面临问题：

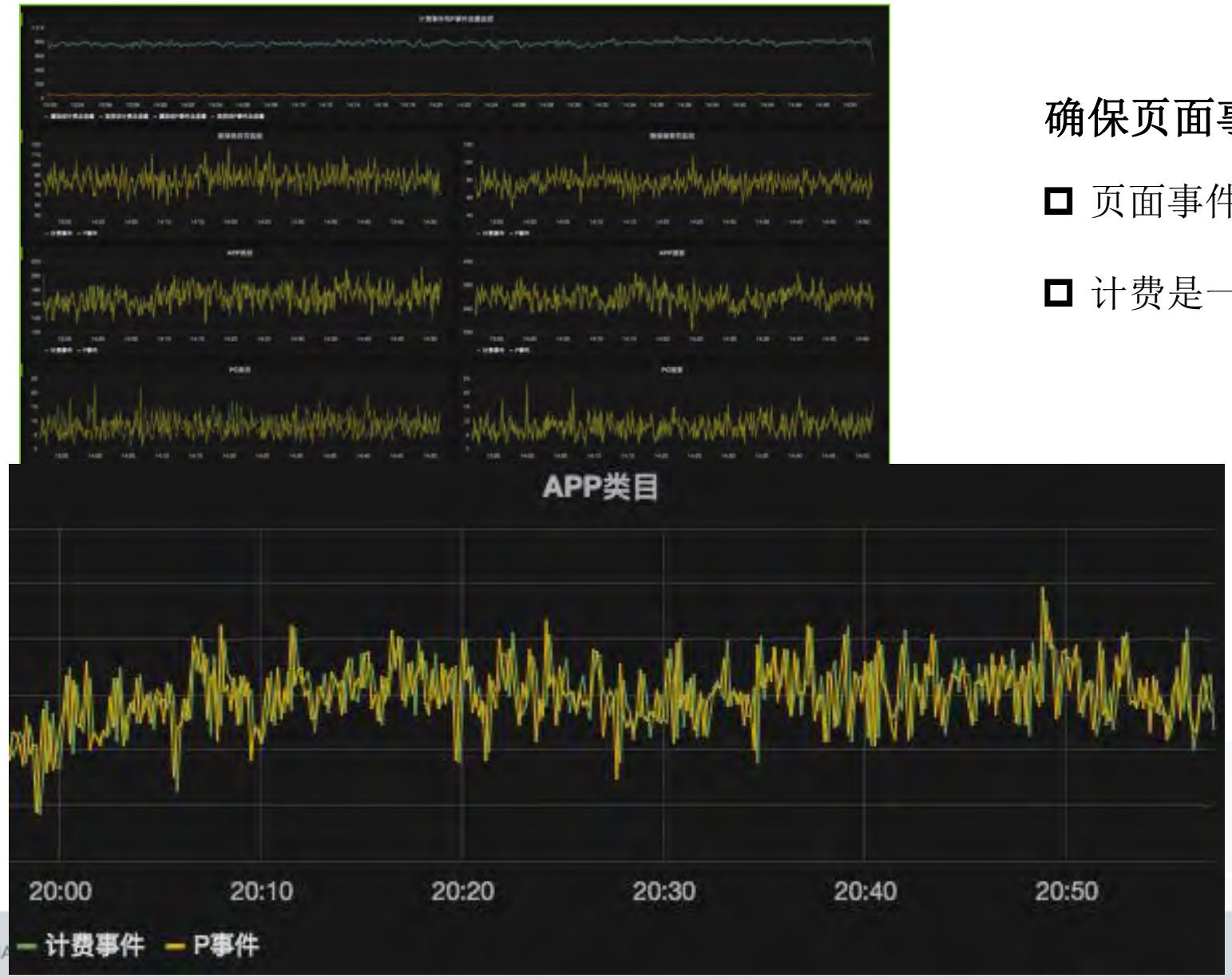
- 前端链路是否丢失
  （流量劫持, 第三方拦截）
- 后端链路数据处理是否异常

# 二、链路完整性-前端链路数据完整性校验




- 日志端: 集中式接入、保持业务无关性
- 应用端:轻量级数据订阅
- 实时分拣:定制化的业务规则

# 二、链路完整性-前端链路数据完整性校验

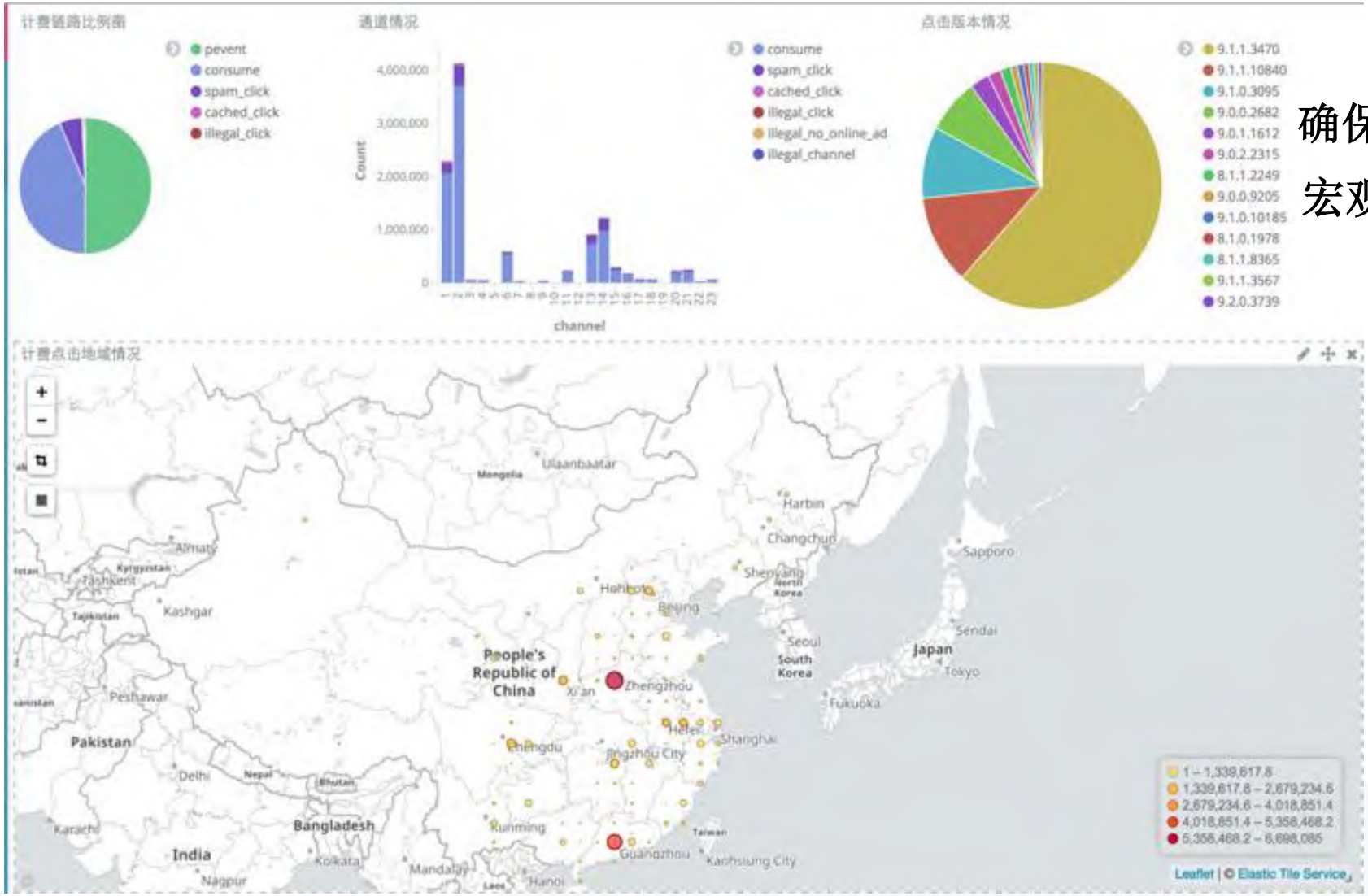



## 确保页面事件触发及打点完整性

- 页面事件和扣费事件监控
- 计费是一个实时过程，实时的监控校验

计费事件和P事件某日流量监控图

# 二、链路完整性-后端链路数据完整性校验




链路实时数据可视化图表

**确保后端接收及处理数据完整性**

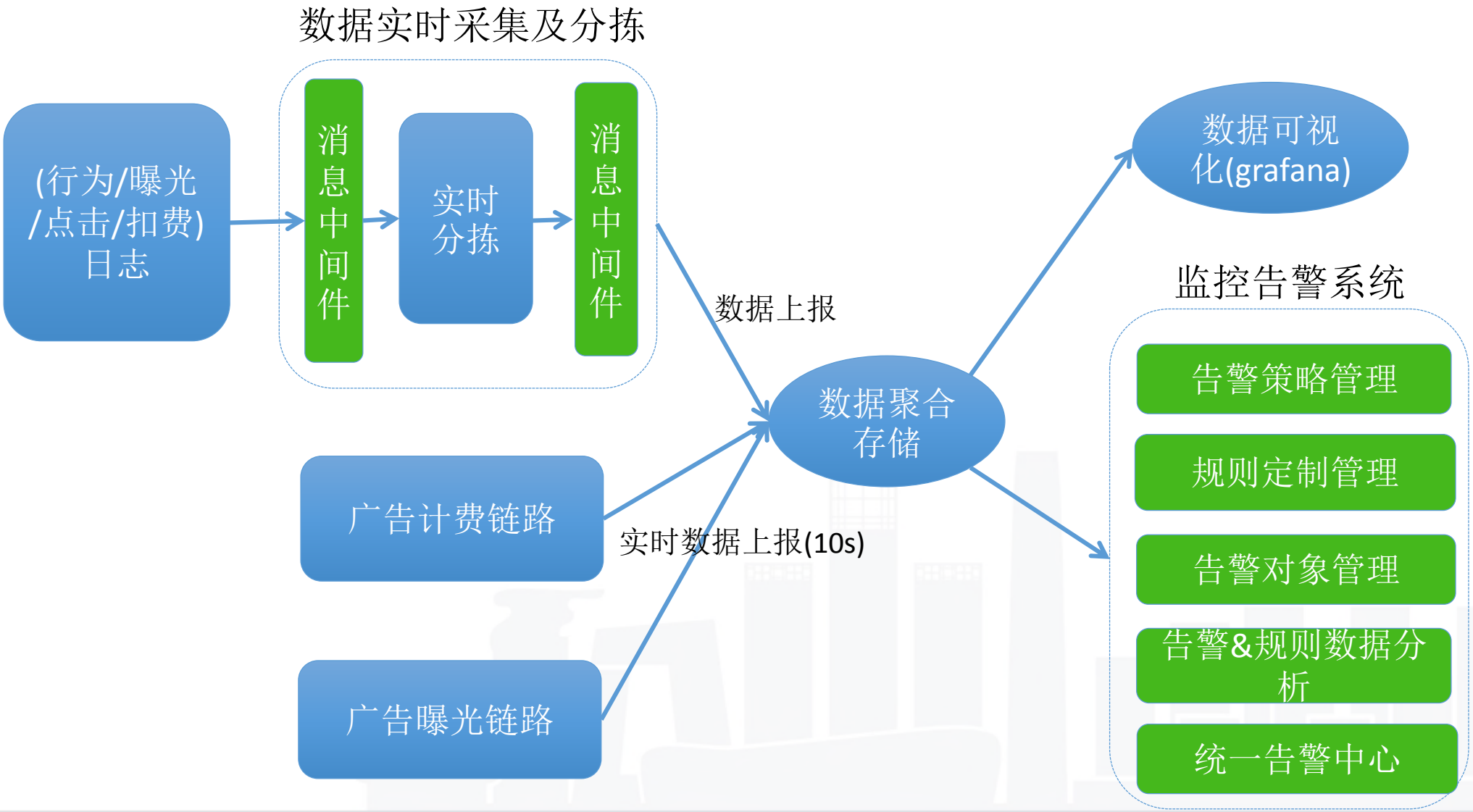
**宏观层面监控业务逻辑的正确性**

- 业务层面的实时分析对比
- 地域分布，渠道分布，场景分布，APP版本分布等

# 监控

- 减少误报
- 快速发现问题
- 能很好的定位问题

# 三、监控-数据流


数据实时采集及分拣
(行为/曝光/点击/扣费)日志
消息中间件
实时分拣
消息中间件
数据上报
广告计费链路
广告曝光链路
实时数据上报(10s)
数据聚合存储
数据可视化(grafana)
监控告警系统
告警策略管理
规则定制管理
告警对象管理
告警&规则数据分析
统一告警中心

# 三、监控-告警规则与量化

## 监控对象

- 业务规则监控
- 系统可靠性监控

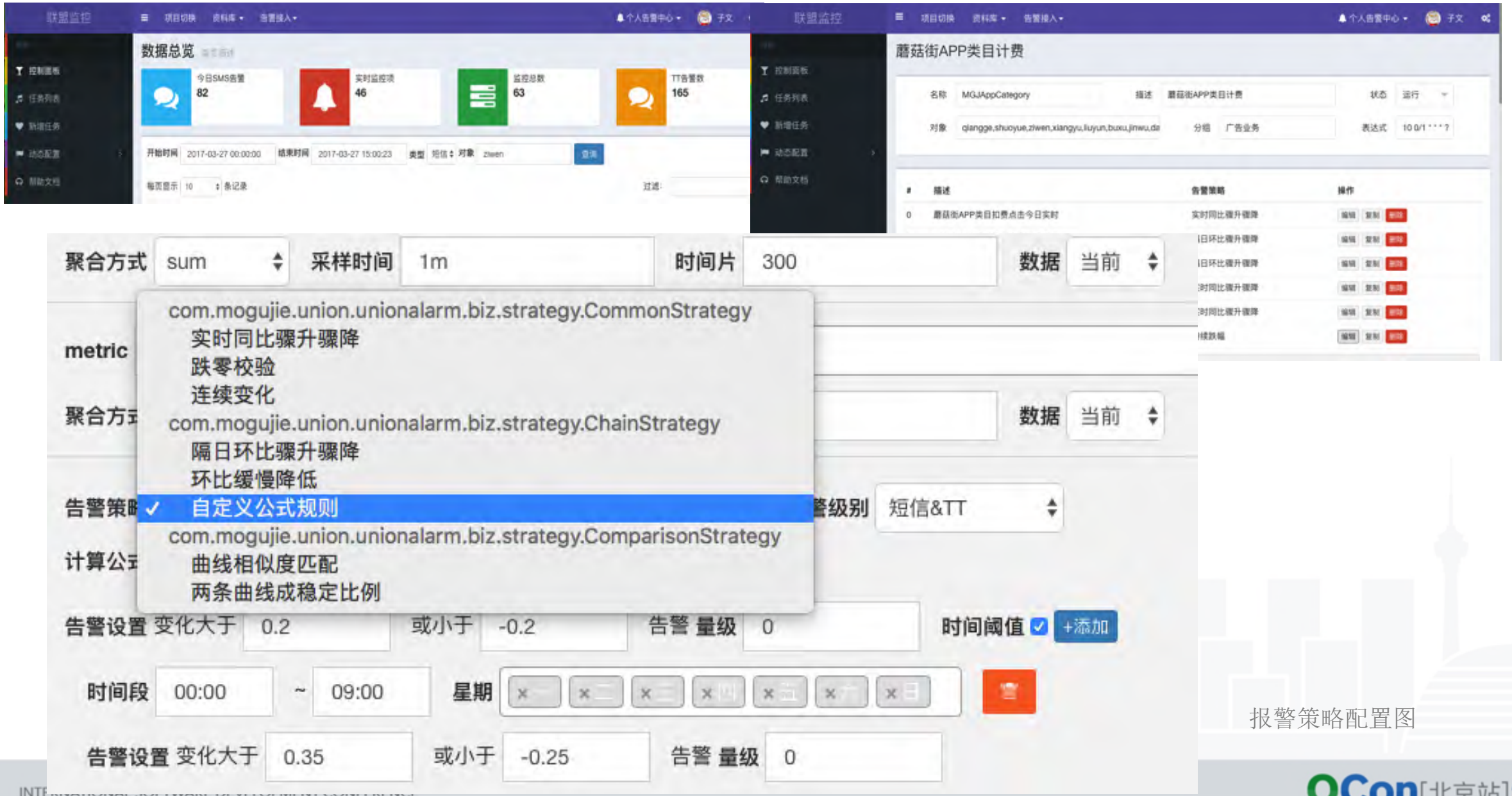
## 量化指标

- **误警率**：在所有产生的告警中，有多少代表了真正的故障
- **敏感度**：在所有真正的故障中，有多少产生了告警

一般来说，误警率可以放宽一些。虽然可能会产生狼来了的结果，但是稍微多一两条告警并不会产生什么实质的危害。需要卡控得比较严格的是漏警的情况。

一旦监控系统漏警了，往往意味着一起责任事故和不小的损失。所以我们需要在这2个之间做一个平衡

# 三、监控-告警系统&策略

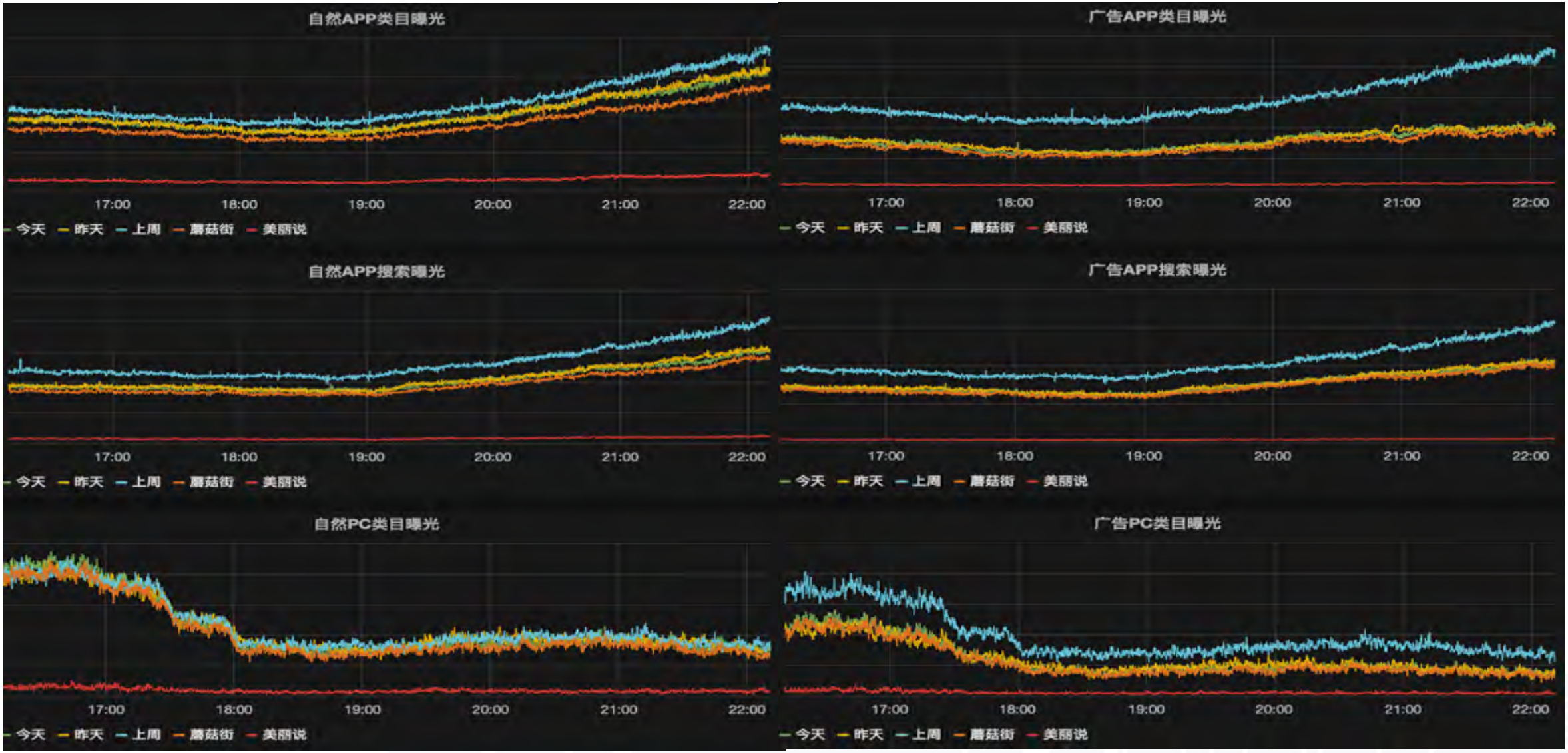



报警策略配置图

# 三、监控-计费全链路监控


计费全链路某日监控图

# 三、监控-分场景环节监控




分场景环节某日监控图

# 四、总结

- **计费系统互备与扩展**

  采用多机房部署,利用ZooKeeper连接同步状态信息,30秒节点失联,自动切换,每次切换告警通知,存活节点自动接管,系统各个模块支持平行扩展

- **网络切割多链路数据保证**

  任何中间链路依赖服务失联,灾备链路保证日志到最终计费的正常,整个链路数据一致性的保证

- **系统健康报告**

  心跳机制, 机器健康指标， 系统资源状况， 服务运行情况, 全链路全方位监控, **系统异常秒级别告警**

- **全链路数据监控**

  多维度,多场景,全链路数据监控,从日志打点到最终数据处理,链路数据出错或者上层业务异常变更， 10s内快速发现

# 容灾设计的核心点


## 05 监控

- 链路监控
- 分场景环节监控
- 报警策略

## 01 计费效率

- 计费实效性保证
- 热点数据不均衡避免
- 资源效率的提升

## 04 链路完整性

- 链路数据完整性校验
- 链路数据一致性保障
- 资金账户流水一致性
- 数据安全性的保障

## 02 分区容错性

与依赖系统出现局部网络分割时，需保证数据不丢失

## 03 可用性

- 容灾
- 幂等性
- 限流降级方案

# 03

# 未来展望

# 未来展望

电商商业化规则的复杂，各种新的商业化模式的接入，减少系统成本和复杂性的同时，怎样更加稳定快速的支持各种商业模式的计费

- 日志统一，数据流处理归一化，提高效率
- 业务逻辑抽离,高效率扩展,底层平台化
- 系统稳定性保障自动化、产品化

- 结合投放链路流量分配规则，支持AB实验数据查询；
- 依赖实时数据（补全商品、地域信息）进行多维度交叉分析；
- 依赖实时多维度数据产生一系列商家工具，帮商家了解运营状况

# Q&A

部门技术公众号