阿里云
日志服务

最佳实践

文档版本：20190903

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的”现状“、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含”阿里云”、Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告:** 重启操作将导致业务中断，恢复业务所需时间约10分钟。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明:** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	设置 > 网络 > 设置网络类型
粗体	表示按键、菜单、页面名称等UI元素。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	表示参数、变量。	`bae log list --instanceid` *`Instance_ID`*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig` *`[-all|-t]`*
{}或者{a|b}	表示必选项，至多选择一个。	`swich` *`{stand | slave}`*

目录

1 典型使用场景

基于日志服务的解决方案

日志服务与多个云产品、以及第三方开源生态进行对接，最大程度上降低用户的使用门槛。例如流计算、数据仓库、监控等。除此之外，日志服务在安全等领域引入ISV，可以通过安全云市场享受日志分析专家的服务。

方案

典型场景描述：

- 日志、大数据分析
  - 通过 Agent、API 实时收集系统产生的事件，例如访问、点击等。
  - 通过 Loghub 接口进行流计算，例如分析用户最喜爱的节目，当前观看最高的频道，各个省市点播率等，精确运营。
  - 对日志进行数仓离线归档，每天、每周出详细的运营数据、账单等。
  - 适用领域：流媒体、电子商务、移动分析、游戏运营等。例如网站运营 CNZZ 也是日志服务的用户。
- 日志审计
  - 通过 Agent 实时收集日志至日志服务，在此过程中无需担心误删、或被黑客删除。
  - 通过日志查询功能，快速分析访问行为，例如查询某个账户、某个对象、某个操作的操作记录。
  - 通过日志投递 OSS、MaxCompute 对日志进行长时间存储，满足合规审计需求。
  - 适用领域：电子商务、政府平台、网站等。 阿里云官网产品ActionTrail、日志审计等就是基于日志服务开发的。
- 问题诊断
  - 开发过程中，对客户端、移动设备、服务端、模块等加入日志，并通过 ID 进行关联。
  - 收集各个模块日志，通过云监控、流计算等实时获得访问情况。
  - 当请求或订单发生错误时，开发无需登录服务器，直接通过日志查询功能对错误关键词、次数、关联影响等进行查询，快速定位问题，减少影响覆盖面。
  - 适用领域：交易系统、订单系统、移动网络等。

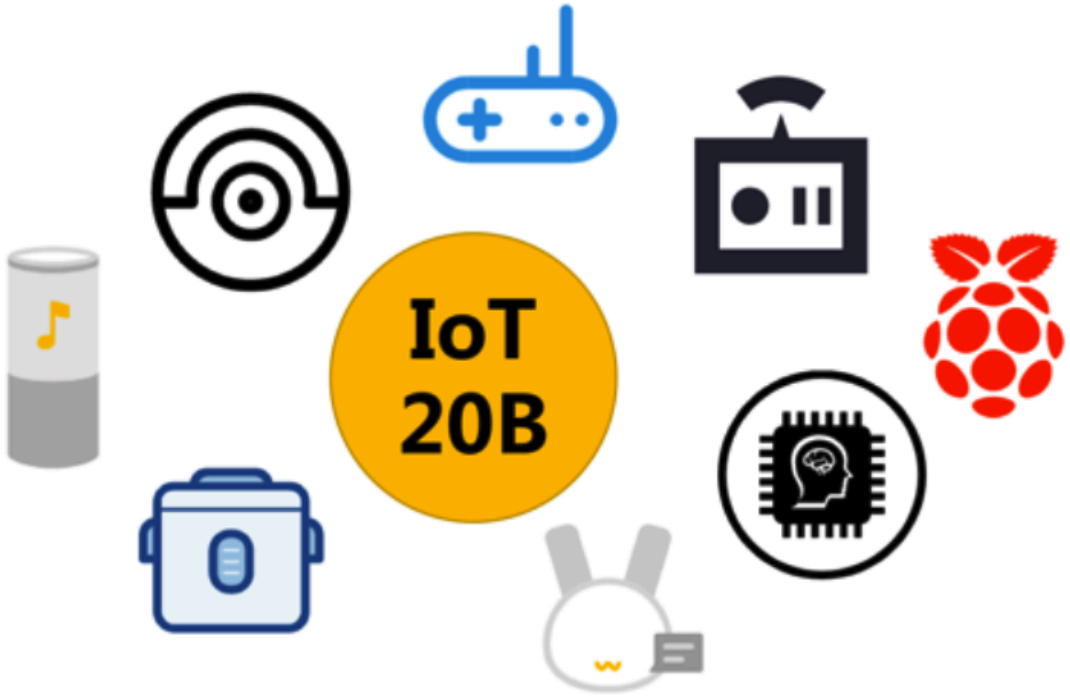
- 运维管理
  - 收集上百台、上千台机器上不同应用程序的日志（包括错误、访问日志、操作日志等）。
  - 通过不同的日志库、机器组对应用程序进行集中式管理。
  - 对不同日志进行处理。例如，访问日志进行流计算做实时监控；对操作日志进行索引、实时查询；对重要日志进行离线存档。
  - 日志服务提供全套 API 进行配置管理和集成。
  - 适用领域：有较多服务器需要管理的用户。
- 其它
  - 计量计费、业务系统监控、漏洞检测、运营分析、移动客户端分析等。在阿里云内部，日志服务无处不在，几乎所有云产品都在使用日志服务解决日志处理、分析等问题。

2 采集

2.1 采集-IoT/嵌入式日志

IoT（Internet of Things）正在高速增长，越来越多设备开始逐步走进日常生活，例如智能路由器、各种电视棒、天猫精灵、扫地机器人等，让我们体验到智能领域的便利。距Gartner预测，到2020年末预计会有200亿智能设备，可见该领域的巨大市场。传统软件领域的嵌入式开发模式在IoT设备领域的应用遇到了很多挑战，IoT设备数目多、分布广，难以调试且硬件受限，传统的设备日志解决方案无法完美满足需求。

日志服务团队根据多年Logtail的开发经验，结合IoT设备的特点，为IoT设备量身定制一套日志数据采集方案：C Producer。

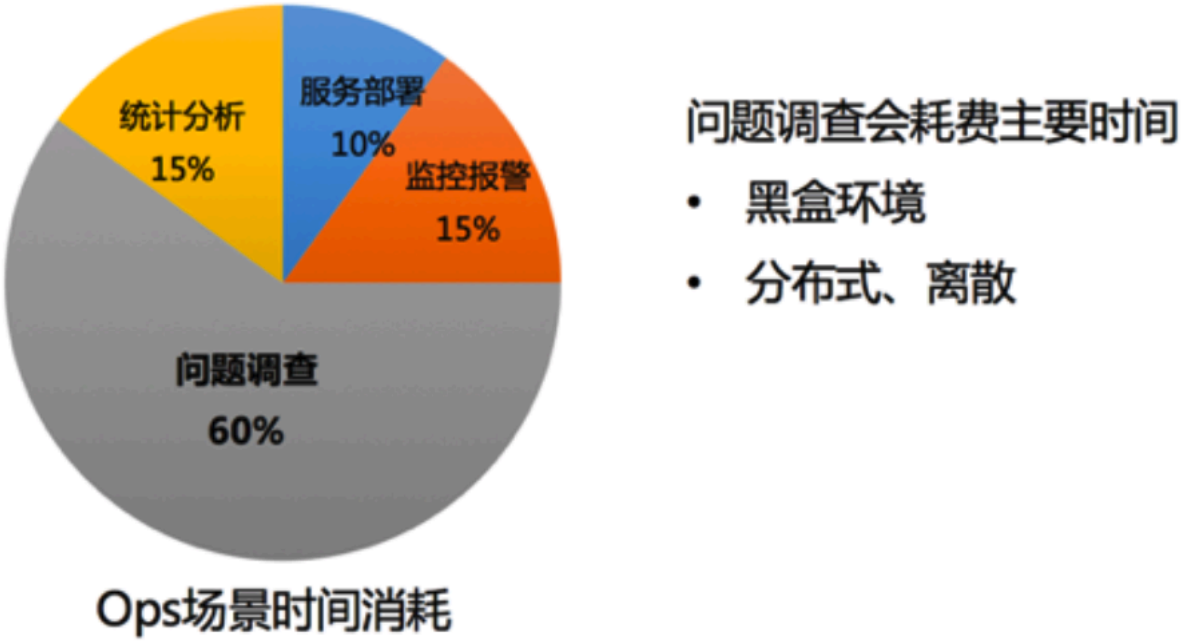


嵌入式开发需求

作为IoT/嵌入式工程师，除了需要深厚的开发功底外，面对海量的设备，如何有能力管理、监控、诊断这些“黑盒”设备至关重要。嵌入式开发需求主要有以下几点：

- 数据采集：如何实时采集分散在全球各地的百万/千万级设备上的数据?
- 调试：如何使用一套方案既满足线上数据采集又满足开发时的实时调试?
- 线上诊断：某个线上设备出现错误，如何快速定位设备，查看引起该设备出错的上下文是什么?

- 监控：当前有多少个设备在线？工作状态分布如何？地理位置分布如何？出错设备如何实时告警？
- 数据实时分析：设备产生数据如何与实时计算、大数据仓库对接，构建用户画像？



IoT领域面临的主要挑战

思考以上问题的解决方案，我们发现在传统软件领域那一套手段面临IoT领域基本全部失效，主要挑战来自于IoT设备这些特点：

- 设备数目多：在传统运维领域管理1W台服务器属于一家大公司了，但10W在线对于IoT设备而言只是一个小门槛。
- 分布广：硬件一旦部署后，往往会部署在全国、甚至全球各地。
- 黑盒：难以登陆并调试，大部分情况属于不可知状态。
- 资源受限：出于成本考虑，IoT设备硬件较为受限（例如总共只有32MB内存），传统PC领域手段往往失效。

C Prodecer:日志服务量身定制的日志数据采集解决方案

日志服务(原SLS)客户端Logtail在X86服务器上有百万级部署，可以参见文章：Logtail技术分享：多租户隔离技术+双十一实战效果，Polling + Inotify 组合下的日志保序采集方案。除此之外，日志服务提供多样化的采集方案：

- 移动端SDK：Android/IOS平台数据采集，一天已有千万级DAU。
- Web Tracking（JS）：类似百度统计，Google Analytics 轻量级采集方式，无需签名。

在IoT领域，我们从多年Logtail的开发经验中，汲取其中精华的部分，并结合IoT设备针对CPU、内存、磁盘、网络、应用方式等特点，开发出一套专为IoT定制的日志数据采集方案：C Producer。

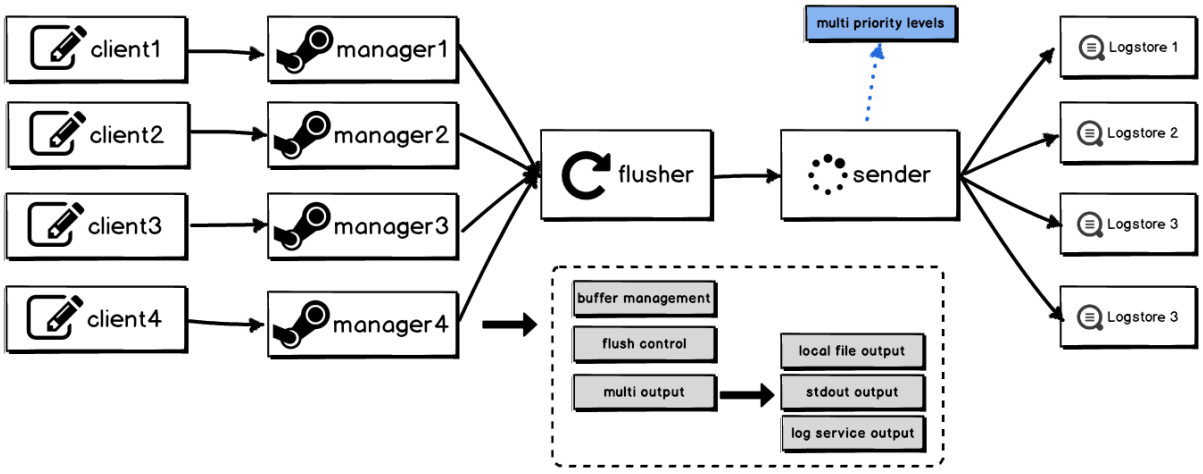
C Producer特点

C Producer Library 继承Logtail稳定、边界特点，可以定位是一个“轻量级Logtail”，虽没有Logtail实时配置管理机制，但具备除此之外70%功能，包括：

- 提供多租户概念：可以对多种日志（例如Metric，DebugLog，ErrorLog）进行优先级分级处理，同时配置多个客户端，每个客户端可独立配置采集优先级、目的project/logstore等。
- 支持上下文查询：同一个客户端产生的日志在同一上下文中，支持查看某条日志前后相关日志。
- 并发发送，断点续传：支持缓存上线可设置，超过上限后日志写入失败。

此外，C Producer还具备以下IoT设备专享功能，例如：

- 本地调试：支持将日志内容输出到本地，并支持轮转、日志数、轮转大小设置。
- 细粒度资源控制：支持针对不同类型数据/日志设置不同的缓存上线、聚合方式。
- 日志压缩缓存：支持将未发送成功的数据压缩缓存，减少设备内存占用。



功能优势

C-Producer作为IoT设备的量身定制方案，在以下方面具备明显优势：

- 客户端高并发写入：可配置的发送线程池，支持每秒数十万条日志写入，详情参见性能测试。
- 低资源消耗：每秒20W日志写入只消耗70% CPU；同时在低性能硬件（例如树莓派）上，每秒产生100条日志对资源基本无影响。
- 客户端日志不落盘：既数据产生后直接通过网络发往服务端。
- 客户端计算与 I/O 逻辑分离：日志异步输出，不阻塞工作线程。
- 支持多优先级：不通客户端可配置不同的优先级，保证高优先级日志最先发送。
- 本地调试：支持设置本地调试，便于您在网络不通的情况下本地测试应用程序。

在以上场景中，C Producer Library 会简化您程序开发的步骤，您无需关心日志采集细节实现、也不用担心日志采集会影响您的业务正常运行，大大降低数据采集门槛。

为了有一个感性认识，我们对C-Producer 方案与其他嵌入式采集方案做了一个对比，如下：

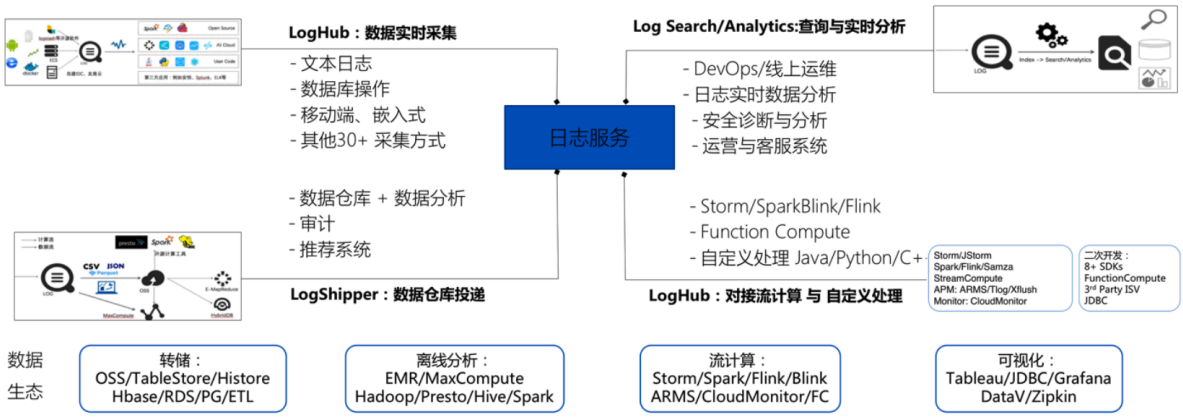
类别		C Producer	其他方案
编程	平台	移动端+嵌入式	移动端为主
	上下文	支持	不支持
	多日志	支持	不支持（一种日志）
	自定义格式	支持	不支持（提供若干个有限字段）
	优先级	支持	不支持
	环境参数	可配置	可配置
稳定性	并发度	高	一般
	压缩算法	LZ4（效率与性能平衡）+GZIP	优化
	低资源消耗	优化	一般
传输	断电续传	支持	默认不支持，需要二次开发
	接入点	8（国内）+8（全球）	杭州
调试	本地日志	支持	手动支持
	参数配置	支持	不支持
实时性	服务端可见	1秒（99.9%），3秒（Max）	1-2小时
自定义处理		15+对接方式	定制化实时+离线方案

C-Producer+日志服务解决方案

C-Producer结合阿里云日志服务产品配合使用，即可完成IoT设备日志全套解决方案。

- 规模大
  - 支持亿级别客户端实时写入。
  - 支持 PB/Day 数据量。
- 速度快
  - 采集快：0延迟：写入0延迟，写入即可消费。
    - 查询快：一秒内，复杂查询（5个条件）可处理10亿级数据。
    - 分析快：一秒内，复杂分析（5个维度聚合+GroupBy）可聚合亿级别数据。

- 对接广
  - 与阿里云各类产品无缝打通。
  - 各种开源格式存储、计算、可视化系统完美兼容。

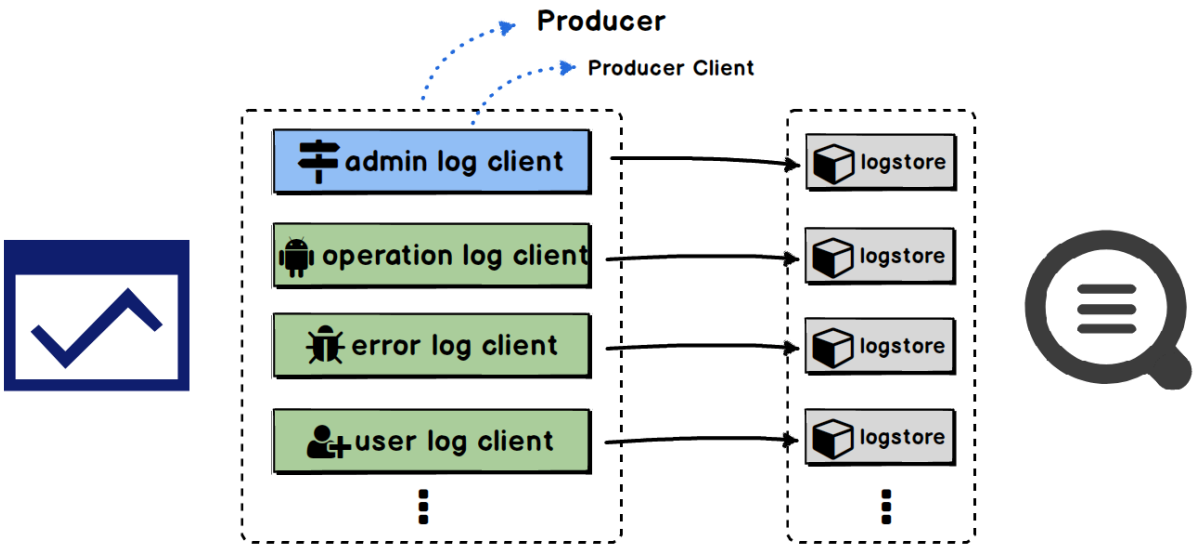


下载与使用

下载地址：Github

一个应用可创建多个producer，每个producer可包含多个client，每个client可单独配置目的地址、日志level、是否本地调试、缓存大小、自定义标识、topic等信息。

详细安装方式及操作步骤，请参考Github页面的README。



性能测试

环境配置

- 高性能场景：传统X86服务器。
- 低性能场景：树莓派（低功耗环境）。

配置分别如下：

高性能场景	低性能场景
• CPU : Intel(R) Xeon(R) CPU E5-2682 v4 @ 2.50GHz • MEM : 64GB • OS : Linux version 2.6.32-220.23.2.ali1113.el5.x86_64 • GCC : 4.1.2 • c-producer：动态库 162K、静态库140K (测试使用静态库，编译后的binary 157KB，所有都是strip后)	型号：树莓派3B CPU：Broadcom BCM2837 1.2GHz A53 64位(使用主机USB供电，被降频到600MHz) 内存：1GB DDR2 OS: Linux 4.9.41-v7+ #1023 SMP armv71 GNU/Linux GCC：6.3.0 (Raspbian 6.3.0-18+rpi1) c-producer：动态库 179K、静态库162K (测试使用静态库，编译后的binary 287KB，所有都是strip后)

C-Producer配置

· ARM（树莓派）

- 缓存：10MB
- 聚合时间：3秒 （聚合时间、聚合数据包大小、聚合日志数任一满足即打包发送）
- 聚合数据包大小：1MB
- 聚合日志数：1000
- 发送线程：1
- 自定义tag：5

· X86

- 缓存：10MB
- 聚合时间：3秒 （聚合时间、聚合数据包大小、聚合日志数任一满足即打包发送）
- 聚合数据包大小：3MB
- 聚合日志数：4096
- 发送线程：4
- 自定义tag：5

日志样例

1. 10个键值对，总数据量约为600字节

2. 9个键值对，数据量约为350字节

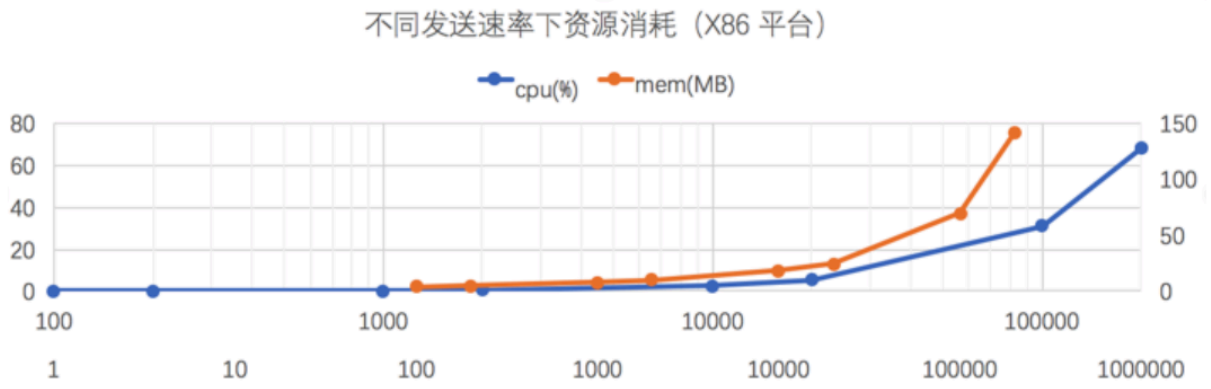
```
__source__: 11.164.233.187
__tag__:1: 2
__tag__:5: 6
__tag__:a: b
__tag__:c: d
__tag__:tag_key: tag_value
__topic__: topic_test
_file_: /disk1/workspace/tools/aliyun-log-c-sdk/sample/log_produc
er_sample.c
_function_: log_producer_post_logs
```

```
_level_: LOG_PRODUCER_LEVEL_WARN
_line_: 248
_thread_: 40978304
LogHub: Real-time log collection and consumption
Search/Analytics: Query and real-time analysis
Interconnection: Grafana and JDBC/SQL92
Visualized: dashboard and report functions
```

测试结果

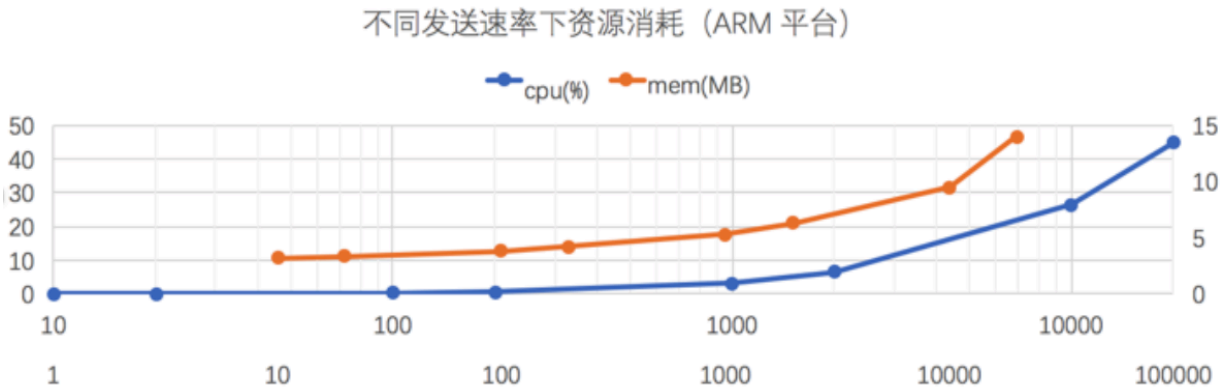
X86平台结果

- **C Producer可以轻松到达90M/s的发送速度，每秒上传日志20W，占用CPU只有70%，内存140M。**
- **服务器在200条/s，发送数据对于cpu基本无影响（降低到0.01%以内）。**
- **客户线程发送一条数据（输出一条log）的平均耗时为：1.2us。**



树莓派平台结果

- **在树莓派的测试中，由于CPU的频率只有600MHz，性能差不多是服务器的1/10左右，最高每秒可发送2W条日志。**
- **树莓派在20条/s的时候，发送数据对于cpu基本无影响（降低到0.01%以内）。**
- **客户线程发送一条数据（输出一条log）的平均耗时为：12us左右（树莓派通过USB连接到PC共享网络）。**



更多日志服务典型场景可以参见云栖论坛 和最佳实践。

2.2 采集-通过WebTracking采集日志

当发送重要邮件时为了确认对方已读，都会在邮件中设置读取回执标签，可以在对方已读时收到提醒信息。

读取回执这种模式用途很广，例如：

- 发送传单时，确保对方已读。
- 推广网页时，多少用户做了点击。
- 移动App运营活动页面，分析用户访问情况。

对这类个性化的采集与统计，针对网站与站长的传统方案都无法胜任，主要难点在于：

- 个性化需求难满足：用户产生行为并非移动端场景，其中会包括一些运营个性化需求字段，例如：来源、渠道、环境、行为等参数。
- 开发难度大/成本高：为完成一次数据采集、分析需求，首先需要购买云主机，公网IP，开发数据接收服务器，消息中间件等，并且通过互备保障服务高可用；接下来需要开发服务端并进行测试。
- 使用不易：数据达到服务端后，还需要工程师先清洗结果并导入数据库，生成运营需要的数据。
- 无法弹性：无法预估用户的使用量，因此需要预留很大的资源池。

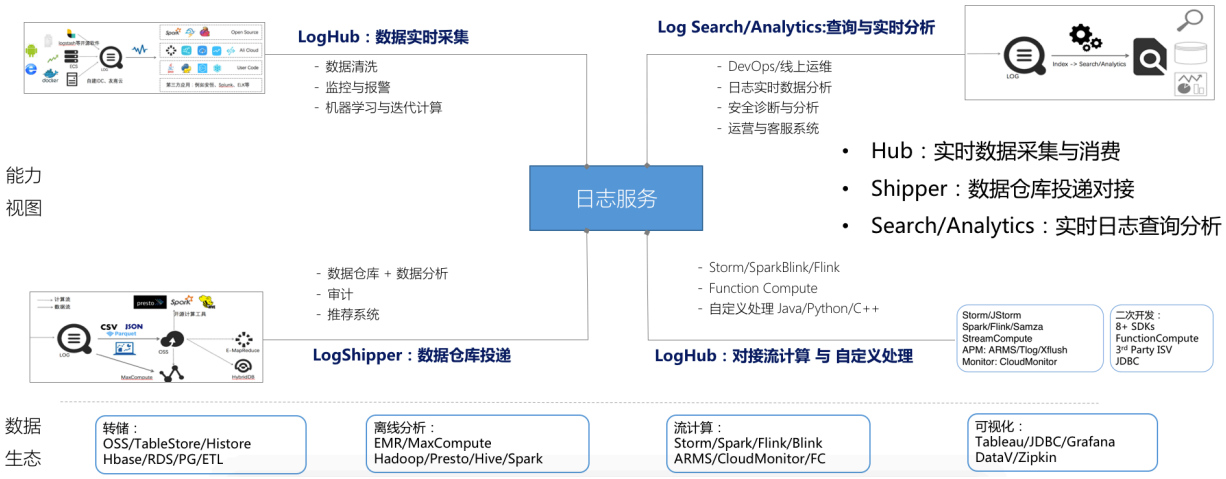
从以上几点看，当一个面向内容投放的运营需求来了后，如何能以快捷的手段满足这类用户行为采集、分析需求是一个很大的挑战。

日志服务(原SLS)提供Web Tracking/JS/Tracking Pixel SDK 就是为解决以上轻量级埋点采集场景而生，我们可以在1分钟时间内完成埋点和数据上报工作。此外日志服务每账号*每月提供500MB 免费额度，让您不花钱也能处理业务。

功能特点

这里引入采集 + 分析方案基于阿里云日志服务，该服务是针对日志类数据的一站式服务，无需开发就能快捷完成海量日志数据的采集、消费、投递以及查询分析等功能，提升运维、运营效率。服务功能包括：

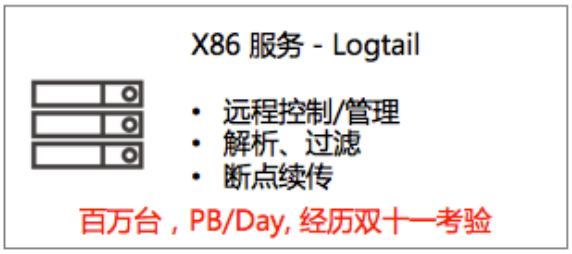
- LogHub：实时采集与消费。与Blink、Flink、Spark Streaming、Storm、Kepler打通。
- 数据投递：LogShipper。与MaxCompute、E-MapReduce、OSS、Function Compute打通。
- 查询与实时分析：LogSearch/Analytics。与DataV，Grafana，Zipkin，Tableua等打通。

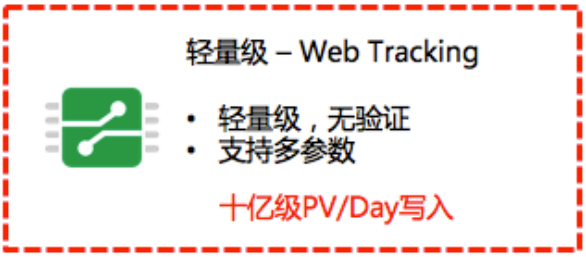


采集端优势

日志服务提供30+种数据采集方式，针对服务器、移动端、嵌入式设备及各种开发语言都提供完整的解决方案，例如：

- **Logtail：针对X86服务器设计Agent。**
- **Android/iOS：针对移动端SDK。**
- **Producer Library：面向受限CPU/内存、智能设备。**





本文档中介绍的轻量级采集方案（Web Tracking）只需一个http get请求即可将数据传输至日志服务Logstore端，适应各种无需任何验证的静态网页、广告投放、宣传资料和移动端数据采集。相比其他日志采集方案，特点如下：

Web Tracking接入流程

Web Tracking（也叫Tracking Pixel）术语来自于HTML语法中的图片标签：我们可以在页面上嵌入一个0 Pixel图片，该图片默认对用户不可见，当访问该页面显示加载图片时，会顺带发起一个Get请求到服务端，这个时候就会把参数传给服务端。

Web Tracking接入流程如下：

1. 为Logstore打开Web Tracking标签。

   Logstore默认不允许匿名写，在使用前需要先开通Logstore的Web Tracking开关。

2. 通过埋点方式向Logstore写入数据。

   您可以通过以下三种方式写入数据。

   - 直接通过HTTP Get方式上报数据。

     ```
     curl --request GET 'http://${project}.${sls-host}/logstores/${
     logstore}/track?APIVersion=0.6.0&key1=val1&key2=val2'
     ```

   - 通过嵌入HTML 下Image标签，当页面方式时自动上报数据。

     ```
     <img src='http://${project}.${sls-host}/logstores/${logstore}/
     track.gif?APIVersion=0.6.0&key1=val1&key2=val2'/>
     or
     <img src=‘http://${project}.${sls-host}/logstores/${logstore}/
     track_ua.gif?APIVersion=0.6.0&key1=val1&key2=val2’/>
     track_ua.gif
     ```

     除了将自定义的参数上传外，在服务端还会将http头中的UserAgent、referer也作为日志中的字段。

   - 通过Java Script SDK 上报数据。

     ```
     <script type="text/javascript" src="loghub-tracking.js" async></
     script>

     var logger = new window.Tracker('${sls-host}','${project}','${
     logstore}');
     logger.push('customer', 'zhangsan');
     ```

```
logger.push('product', 'iphone 6s');
logger.push('price', 5500);
logger.logger();
```

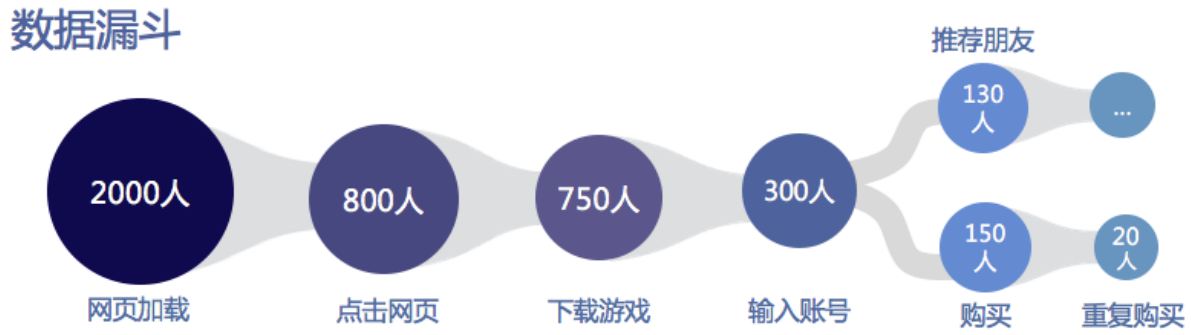
详细步骤请参考#unique_8。

内容多渠道推广实例

应用场景

当我们有一个新内容时（例如新功能、新活动、新游戏、新文章），作为运营人员总是迫不及待地希望能尽快传达到用户，因为这是获取用户的第一步、也是最重要的一步。

以游戏发行为例，市场很大一笔费用进行游戏推广，例如投放了1W次广告。广告成功加载的有2000人次，约占20%。其中点击的有800人次，最终下载并注册账号试玩的往往少之又少。



可见，能够准确、实时地获得内容推广有效性对于业务非常重要。为了达到整体推广目标，运营人员往往会挑选各个渠道来进行推广，例如：

- 用户站内信（Mail），官网博客（Blog），首页文案（Banner等）。
- 短信，用户Email，传单等。
- 新浪微博，钉钉用户群，微信公众账号，知乎论坛，今日头条等新媒体。

操作步骤

步骤一 开启Web Tracking功能

在日志服务中创建一个Logstore（例如叫：myclick），并开启WebTracking功能。

步骤二 生成Web Tracking标签

1. 为需要宣传的文档（article=1001）面对每个宣传渠道增加一个标识，并生成Web Tracking标签（以Img标签为例），如下：

   - 站内信渠道(mailDec)。

     ```
     <img src="http://example.cn-hangzhou.log.aliyuncs.com/logstores/myclick/track_ua.gif?APIVersion=0.6.0&from=mailDec&article=1001" alt="" title="">
     ```

   - 官网渠道(aliyunDoc)。

     ```
     <img src="http://example.cn-hangzhou.log.aliyuncs.com/logstores/myclick/track_ua.gif?APIVersion=0.6.0&from=aliyundoc&article=1001" alt="" title="">
     ```

   - 用户邮箱渠道(email)。

     ```
     <img src="http://example.cn-hangzhou.log.aliyuncs.com/logstores/myclick/track_ua.gif?APIVersion=0.6.0&from=email&article=1001" alt="" title="">
     ```

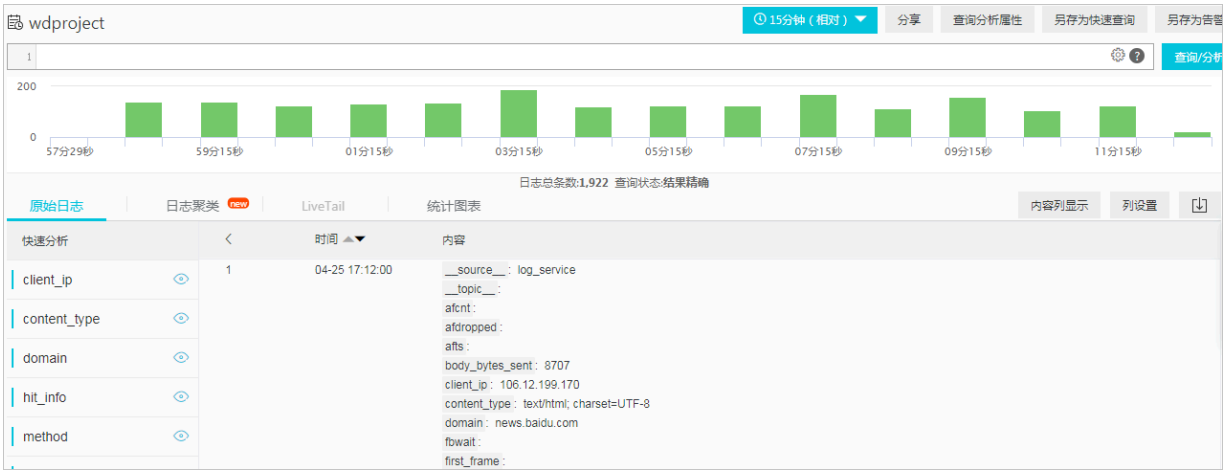
   其他更多渠道可以在from参数后加上，也可以在URL中加入更多需要采集的参数。

2. 将img标签放置在宣传内容中，就可以发布了。

步骤三 分析日志

在完成埋点采集后，我们使用日志服务LogSearch/Analytics功能可以对海量日志数据进行实时查询与分析。在结果分析可视化上，除自带Dashboard外，还支持DataV、Grafana、Tableua等对接方式。

以下是截止目前采集日志数据，您可以在搜索框中输入关键词进行查询。



也可以在查询后输入SQL进行秒级的实时分析并可视化：

1. 设计查询语句。

   以下是我们对用户点击/阅读日志的实时分析语句，更多字段和分析场景可以参见分析语法。

   - 当前投放总流量与阅读数。

     ```
     * | select count(1) as c
     ```

   - 每个小时阅读量的曲线。

     ```
     * | select count(1) as c, date_trunc('hour',from_unixtime(__time__
     )) as time group by time order by time desc limit 100000
     ```

   - 每种渠道阅读量的比例。

     ```
     * | select count(1) as c, f group by f desc
     ```

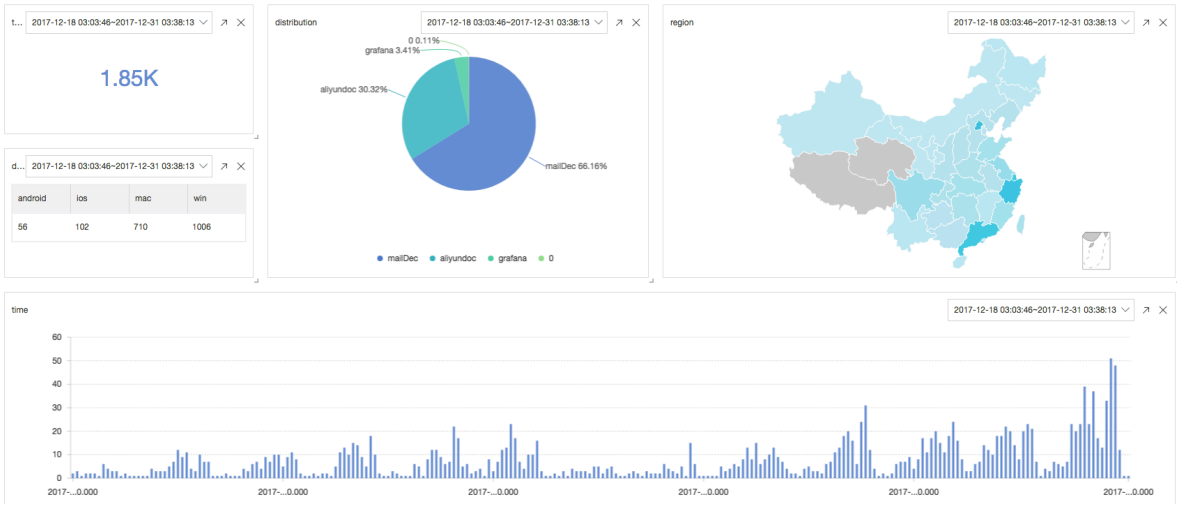
   - 阅读量来自哪些设备。

     ```
     * | select count_if(ua like '%Mac%')  as mac, count_if(ua like '%
     Windows%')  as win, count_if(ua like '%iPhone%')  as ios, count_if
     (ua like '%Android%')  as android
     ```

   - 阅读量来自哪些省市。

     ```
     * | select ip_to_province(__source__) as province, count(1) as c
     group by province order by c desc limit 100
     ```

2. 将这些实时数据配置到一个实时刷新Dashboard中，效果如下：



说明:

当您看完本文时，会有一个不可见的Img标签记录本次访问，您可以在本页面源代码中查看该标签。

2.3 采集-容器服务日志

日志服务支持通过Logtail采集Kubernetes集群日志，完成采集后，您还可以实时预览、检索和分析日志，也可以将检索分析的结果通过DataV在大屏上发布。

应用场景

通过日志服务收集您在杭州地域的Kubernetes集群日志，对收集到的访问日志进行查询分析、并将查询结果通过DataV以图表方式展示。

前提条件

1. 已经开启了Access Key。
2. 开通日志服务产品，并创建了Project和Logstore。
3. 开通DataV产品。

配置流程

1. 部署Logtail的DaemonSet配置Logtail机器组

   日志服务控制台创建自定义标识机器组，后续该Kubernetes集群伸缩无需额外运维。
2. 创建服务端采集配置

   日志服务控制台创建采集配置，所有采集均为服务端配置，无需本地配置。
3. 数据接入DataV

步骤一 部署DaemonSet

1. 连接到您的Kubernetes集群。
2. 配置参数。

   a. 单击下载日志服务 YAML 文件模板，用 vi 编辑器打开。

   b. 将env 环境变量一节中所有的 `${your_xxxx}` 替换为真实值。

参数	说明
`${your_region_name}`	region名，请替换为您创建的日志服务project所在region。region名称请参考Logtail安装参数中使用的region名。 说明: region名中-与_不要混淆。
`${your_aliyun_user_id}`	用户标识，请替换为您的阿里云主账号用户ID。主账号用户ID为字符串形式，如何查看ID请参考#unique_20。

参数	说明
`${your_machine_group_name}`	您集群的机器组自定义标识。如您尚未开启自定义标识，请参考自定义机器组，开启userdefined-id。

说明:

A. 您的主账号需要开启AccessKey，请参考快速入门中的创建密钥对。

B. 请您不要修改模板中的`volumeMounts`和`volumes`部分，否则会造成Logtail无法正常工作。

C. 您可以自定义配置Logtail容器的启动参数，只需保证如下几点即可：

A. 启动时，必须具备3个环境变量：`ALIYUN_LOGTAIL_USER_DEFINED_ID`、`ALIYUN_LOGTAIL_USER_ID`、`ALIYUN_LOGTAIL_CONFIG`。

B. 必须将Docker的Domain Socket挂载到`/var/run/docker.sock`。

C. 如果您需要采集其他容器或宿主机文件，需要将根目录挂载到Logtail容器的`/logtail_host`目录。

c. 部署Logtail的DaemonSet。

示例：

```
[root@iZu kubernetes]# curl http://logtail-release.oss-cn-hangzhou
.aliyuncs.com/docker/k8s/logtail-daemonset.yaml > logtail-
daemonset.yaml
[root@iZu kubernetes]# vi logtail-daemonset.yaml
...
     env:
          - name: "ALIYUN_LOGTAIL_CONFIG"
            value: "/etc/ilogtail/conf/cn_hangzhou/ilogtail_config.
json"
          - name: "ALIYUN_LOGTAIL_USER_ID"
            value: "16542189653****"
          - name: "ALIYUN_LOGTAIL_USER_DEFINED_ID"
            value: "k8s-logtail"
...
[root@iZu kubernetes]# kubectl apply -f logtail-daemonset.yaml
```

步骤二 配置Logtail机器组

1. 登录日志服务并单击目标Project。
2. 单击左侧导航栏的机器组图标展开机器组列表 。
3. 单击机器组后的图标，选择创建机器组。

4. 选择用户自定义标识，将上一步配置的`ALIYUN_LOGTAIL_USER_DEFINED_ID`填入用户自定义标识内容框中。

配置完成一分钟后，单击机器组名称查看已经部署Logtail DaemonSet 节点的心跳状态。具体参见#unique_23中的查看状态。

步骤三 创建服务端采集配置

1. 在概览页面单击接入数据，进入配置流程。
2. 选择数据类型。

   在接入数据页面中选择Docker标准输出。
3. 选择日志空间。

   如果您是通过日志库下的数据接入后的加号进入采集配置流程，系统会直接跳过该步骤。
4. 创建机器组。

   在创建机器组之前，您需要首先确认已经安装了Logtail。

   - 集团内部机器：默认自动安装，如果没有安装，请根据界面提示进行咨询。
   - ECS机器： 勾选实例后单击安装进行一键式安装。Windows系统不支持一键式安装，请参考#unique_24手动安装。
   - 自建机器：请根据界面提示进行安装。或者参考#unique_25或#unique_26文档进行安装。

   安装完Logtail后单击确认安装完毕创建机器组。如果您之前已经创建好机器组，请直接单击使用现有机器组。
5. 机器组配置。

   选择一个机器组，将该机器组从源机器组移动到应用机器组。

6. 数据源设置。

   在数据源设置页面，填写您的采集配置。示例如下，配置项说明请查看#unique_27。

```
{
 "inputs": [
     {
         "type": "service_docker_stdout",
         "detail": {
             "Stdout": true,
             "Stderr": true,
             "IncludeLabel": {
                 "app": "monitor"
             },
             "ExcludeLabel": {
                 "type": "pre"
             }
         }
     }
 ]
}
```

7. 查询分析设置。

   默认已经设置好索引，您也可以手动修改。

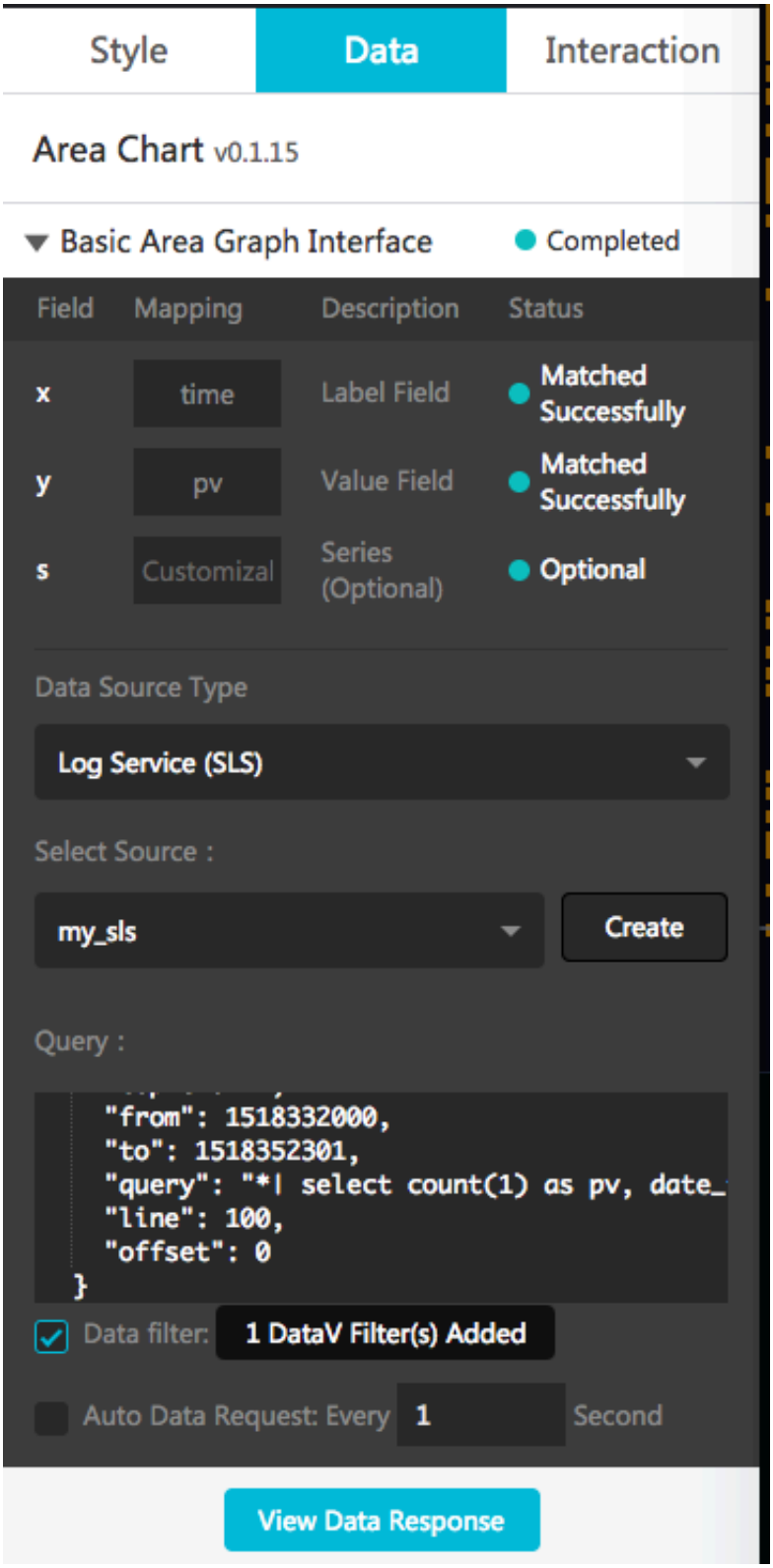
步骤四 数据接入DataV

1. 在我的数据页面中单击添加数据。 填写类型、您的AK等信息，单击完成。

2. 创建Area Chart，并填写相关配置。

其中，Query部分填写以下内容：

```
{
 "projectName": "sls-zc-test-hz-pub",
 "logStoreName": "nginx_access",
 "topic": "",
 "from": 1518332000,
 "to": 1518352301,
 "query": "*| select count(1) as pv, date_format(from_unixtime(
__time__ - __time__%3600) ,'%Y/%m/%d %H:%i:%s')   as time group by
time  order by time limit 1000" ,
 "line": 100,
 "offset": 0
}
```



注意事项

在测试时，示例中的from和to您可以先填写unix time，例如1509897600。发布之后换成：`from`和：`to`。您可以在url参数里控制这两个数值的具体时间范围。例如，预览时的url是http://

datav.aliyun.com/screen/86312，打开http://datav.aliyun.com/screen/86312?from=1510796077&to=1510798877后，会按照指定的时间进行计算。

完成图表配置后，单击预览和发布，一个大屏就创建成功了。

2.4 采集-搭建移动端日志直传服务

在移动互联的时代，直接通过手机APP上传数据越来越普遍，对于日志场景，我们希望将APP的日志直接上传到日志服务中，而不需要通过APP服务端中转，这样用户就能专注在自己的业务逻辑开发。

普通模式下，日志写入日志服务需要开启主账号的访问秘钥，用于鉴权以及防篡改。移动APP如果通过此模式接入日志服务，需要将您的AK信息保存在移动端，存在AK泄漏的数据安全风险。一旦发生AK泄漏，需要升级移动APP，并替换AK，代价太大。另一种移动端日志接入日志服务的方式为通过用户服务器中转，但是该种模式下，如果移动APP数量较大，用户的应用服务器需要承载所有的移动端数据，对服务器的规模有较高的要求。

为了避免以上问题，日志服务为您提供能更安全、更便捷的移动APP日志数据采集方案，即通过RAM搭建一个基于移动服务的移动应用数据直传服务。相较于直接使用AK访问日志服务，用户不需要在APP端保存AK，不存在AK泄露的风险。使用临时Token访问，更加安全，Token有生命周期，并且可以针对Token定制更加复杂的权限控制，比如限制IP段的访问权限等。成本低，用户不需要准备很多服务器，移动应用直联云平台，只有控制流走用户自己的应用服务器。

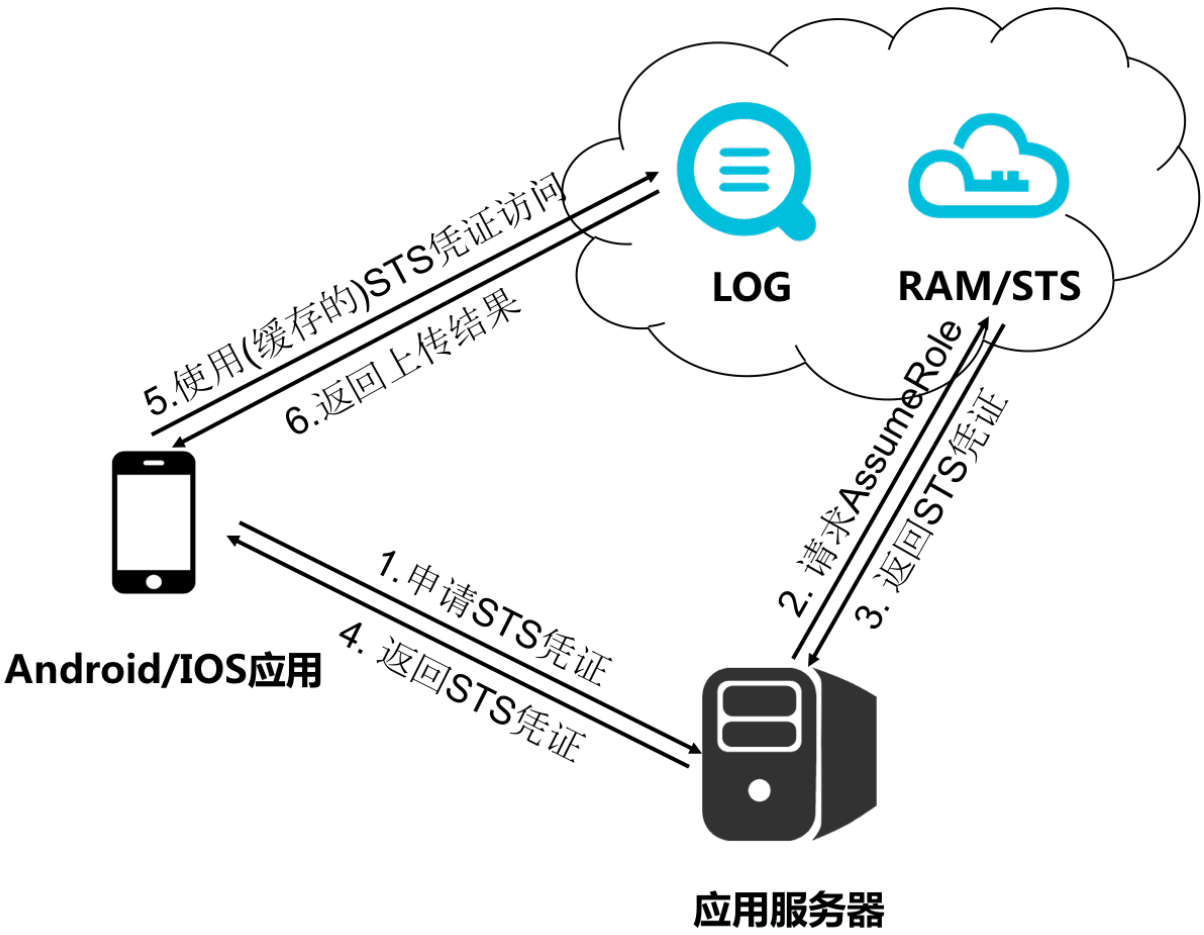
您可以创建日志服务的RAM用户角色，并配置APP作为RAM子用户扮演该角色，从而做到在30分钟内搭建一个基于日志服务的移动应用日志直传服务。所谓直传就是移动应用通过直连方式访问日志服务，只有控制流走用户自己的服务器。

优势

通过RAM搭建一个基于日志服务的移动应用数据直传服务，具有以下优势：

- 访问方式更加安全。临时、灵活的赋权鉴权。
- 成本低，用户不需要准备很多服务器。移动应用直联云平台，只有控制流走用户自己的应用服务器。
- 高并发，支持海量用户。日志服务有海量的上传和下载带宽。
- 弹性。日志服务有无限扩容的存储空间。

架构图如下所示。



说明：

节点	说明
Android/iOS 移动应用	最终用户手机上的APP，日志的来源。
LOG	即阿里云日志服务，负责存储APP上传的日志数据。
RAM/STS	阿里云访问控制云产品，提供用户身份管理和资源访问控制服务。负责生成临时上传凭证。
用户应用服务器	即提供该Android/iOS应用的开发者开发的APP后台服务，管理APP上传和下载的Token，以及用户在APP上传数据的元数据信息。

配置流程

1. 应用向用户的应用服务器申请一个临时访问凭证。

   Android/iOS应用不能直接存储AccessKeyID/AccessKeySecret，这样会存在泄密的风险。所以应用必须向用户的应用服务器申请一个临时上传凭证（下文将此临时上传凭证称为Token）。这个Token是有时效性的，如果这个Token的过期时间是30分钟（这个时间可以由应用服

务器指定），那么在这30分钟里面，该Android/iOS应用可以使用这个Token访问日志服务，30分钟后再重新获取。

2. 用户的应用服务器检测上述请求的合法性，然后返回Token给应用。
3. 手机拿到这个Token后就可以访问日志服务了。

本文档主要介绍应用服务器如何向RAM服务申请这个Token，和Android/iOS应用如何获取Token。

操作步骤

1. 授权用户角色操作日志服务。

创建日志服务的RAM用户角色，并配置APP作为RAM子用户扮演该角色，详细步骤请参考#unique_29。

配置完成后，您将获取以下参数。

- RAM子用户的accessKeyId、accessKey。
- 角色的资源路径RoleArn。

2. 搭建一个应用服务器。

为了方便开发，本教程提供了多个语言的版本示例程序供您下载，下载地址见文章最底部。

每个语言包下载下来后，都会有一个配置文件config.json如下所示：

```
{
 "AccessKeyID" : "",
 "AccessKeySecret" : "",
 "RoleArn" : "",
 "TokenExpireTime" : "900",
 "PolicyFile": "policy/write_policy.txt"
}
```

说明:

1. AccessKeyID：填写您的访问秘钥ID。
2. AccessKeySecret：填写您的访问秘钥Secret。
3. RoleArn：填写用户角色的RoleArn。
4. TokenExpireTime：指Android/iOS应用取到这个Token的失效时间。注意，最少是900s，默认值可以不修改。
5. PolicyFile：填写的是该Token所要拥有的权限列表的文件， 默认值可以不修改。

本文档提供了两种最常用Token权限文件，位于policy目录下面。

- write_policy.txt：指定了该Token拥有该账号下Project的写入权限。

· readonly_policy.txt：指定了该Token拥有该账号下Project的读取权限。

您也可以根据自己的需求设计自己的policy文件。

返回的数据格式：

```
//正确返回
{
    "StatusCode":200,
    "AccessKeyId":"STS.3p***dgagdasdg",
    "AccessKeySecret":"rpnwO9***tGdrddgsR2YrTtI",
   "SecurityToken":"CAES+wMIARKAAZhjH0EUOIhJMQBMjRywXq7MQ/cjLYg80Aho
1ek0Jm63XMhr9Oc5s˙ə˙ə3qaPer8p1YaX1NTDiCFZWFkvlHf1pQhuxfKBc+mRR9KAbHUe
fqH+rdjZqjTF7p2m1wJXP8S6k+G2MpHrUe6TYBkJ43GhhTVFMuM3BZajY3VjZWOXBI
ODRIR1FKZjIiEjMzMzE0MjY0NzM5MTE4NjkxMSoLY2xpZGSSDgSDGAGESGTETqOio6c2Rr
LWRlbW8vKgoUYWNzOm9zczoqOio6c2RrLWRlbW9KEDExNDg5MzAxMDcyNDY4MThSBTI2OD
QyWg9Bc3N1bWVkUm9sZVVzZXJgAGoSMzMzMTQyNjQ3MzkxMTg2OTExcglzZGstZGVtbzI
=",
   "Expiration":"2017-11-12T07:49:09Z",
}

//错误返回
{
    "StatusCode":500,
    "ErrorCode":"InvalidAccessKeyId.NotFound",
    "ErrorMessage":"Specified access key is not found."
}
```

正确返回说明：（下面五个变量将构成一个Token）

状态码	说明
StatusCode	表示获取Token的状态，获取成功时，返回值是200。
AccessKeyId	表示Android/iOS应用初始化LogClient获取的 AccessKeyId
AccessKeySecret	表示Android/iOS应用初始化LogClient获取AccessKeySecret。
SecurityToken	表示Android/iOS应用初始化的Token。
Expiration	表示该Token失效的时间。主要在Android SDK会自动判断是否失效，自动获取Token。

错误返回说明：

错误码	说明
StatusCode	表示获取Token的状态，获取失败时，返回值是500。
ErrorCode	表示错误原因

错误码	说明
ErrorMessage	表示错误的具体信息描述。

代码示例的运行方法：

对于JAVA版本 (依赖于java 1.7+)，将包下载解压后，新建一个java工程，将依赖和代码以及配置拷贝到工程里面，运行main函数即可，程序默认会监听7080端口，等待HTTP请求，其他语言类似。

3. 移动端构造HTTP请求，从应用服务器获取Token。

HTTP请求及回应格式如下。

```
Request URL: GET https://localhost:7080/

Response:
{
"StatusCode":"200",
"AccessKeyId":"STS.XXXXXXXXXXXXXXX",
"AccessKeySecret":"",
"SecurityToken":"",
"Expiration":"2017-11-20T08:23:15Z"
}
```

说明:

本文档中所有示例仅为演示服务器搭建流程，用户可以在此基础上进行自定义开发。

源码下载

应用服务器代码示例： PHP、JAVA、Ruby、Node.js

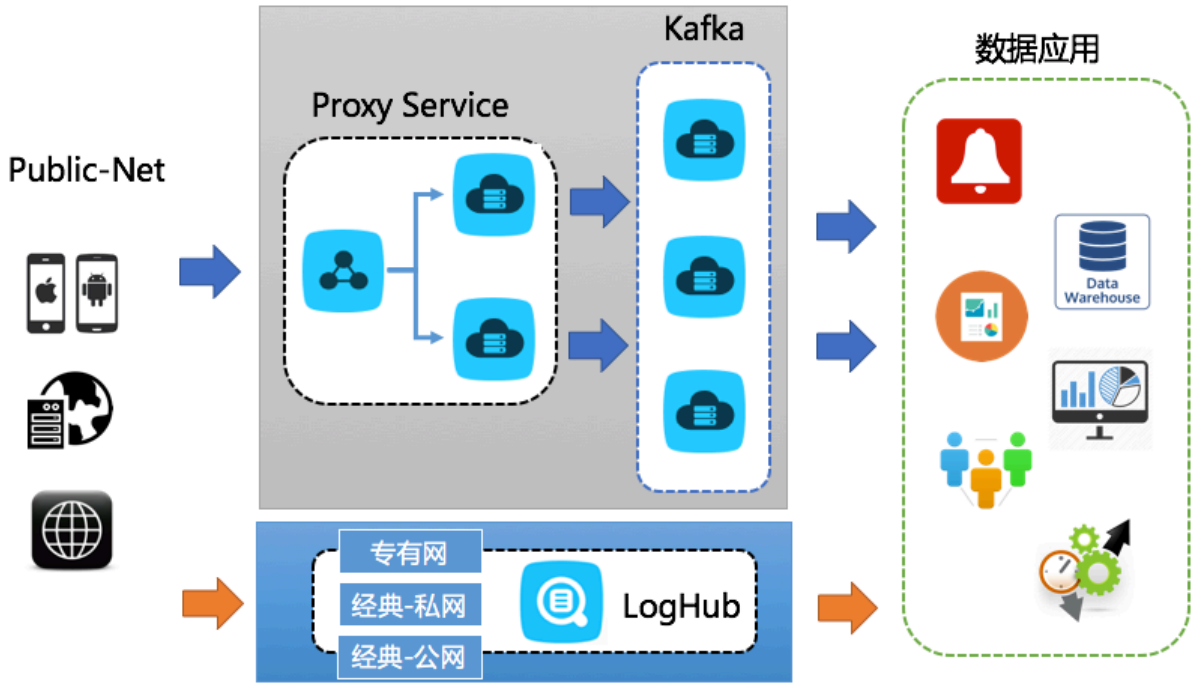
2.5 采集-公网数据

对一些应用场景而言，需要实时收集公网数据 （例如，移动端、HTML 网页、PC、服务器、硬件设备、摄像头等）实时进行处理。

在传统的架构中，一般通过前端服务器 + Kafka 这样的搭配来实现如上的功能。现在 日志服务 的 loghub 功能 能够代替这类架构，并提供更稳定、低成本、弹性、安全的解决方案。

场景

公网有移动端、外部服务器、网页和设备数据进行采集。采集完成后需要进行实时计算、数据仓库等数据应用。



方案 1：前端服务器 + Kafka

由于 Kafka 不提供 Resful 协议，更多是在集群内使用，因此一般需要架设 Nginx 服务器做公网代理，再通过 logstash 或 API 通过 Nginx 写 Kafka 等消息中间件。

需要的设施为：

设施	数目	配置	作用	价格
ECS 服务器	2 台	1 核 2GB	前端机、负载均衡，互备	108 元/台* Month
负载均衡	1 台	标准	按量计费实例	14.4 元/Month（租赁）+ 0.8 元/GB（流量）
Kafka/ZK	3 台	1 核 2GB	数据写入并处理	108 元/台* Month

方案 2：使用 loghub

通过 Mobile SDK、logtail、Web Tracking JS 直接写入 loghub EndPoint。

需要的设施为：

设施	作用	价格
loghub	实时数据采集	< 0.2 元/GB，参见 计费规则

场景对比

场景 1：一天 10GB 数据采集，大约一百万次写请求。（这里 10GB 是压缩后的大小，实际数据大小一般为 50GB~100GB 左右。）

```
方案 1:
--------------
负载均衡 租赁: 0.02 * 24 * 30 = 14.4 元
负载均衡 流量: 10*0.8*30  = 240 元
ECS 费用: 108 * 2 = 216 元
Kafka ECS: 免费，假设与其他服务公用
共计: 484.8 元/月

方案 2:
--------------
loghub 流量: 10 * 0.2 * 3- = 60 元
loghub 请求次数: 0.12  (假设一天 100W 请求) * 30 = 3.6 元
共计: 63.6 元/月
```

场景 2：一天 1TB 数据采集，大约一亿次写请求。

```
方案 1:
--------------
负载均衡 租赁: 0.02 * 24 * 30 = 14.4 元
负载均衡 流量: 1000 * 0.8* 30  = 24000 元
ECS 费用: 108 * 2 = 216 元
Kafka ECS: 免费，假设与其他服务公用
共计: 24230.4 元/月

方案 2:
--------------
loghub 流量: 1000 * 0.15 * 30 = 4500 元  (阶梯计价)
loghub 请求次数: 0.12 * 100 (假设一天 1 亿请求) * 30 = 360 元
共计: 4860 元/月
```

方案比较

从以上两个场景可以看到，使用 loghub 进行公网数据采集，成本是非常有竞争力的。除此之外，和方案 1 相比还有以下优势：

- 弹性伸缩：MB-PB/Day 间流量随意控制
- 丰富的权限控制：通过 ACL 控制读写权限
- 支持 HTTPS：传输加密
- 日志投递免费：不需要额外开发就能与数据仓库对接
- 详尽监控数据：让您清楚业务的情况
- 丰富的 SDK 与上下游对接：和 Kafka 一样拥有完整的下游对接能力，和阿里云及开源产品深度整合

有兴趣可以参见 日志服务主页 体验该服务。

2.6 采集-多渠道数据

日志服务loghub 功能提供数据实时采集与消费，其中实时采集功能支持 30+ 种手段。

数据采集一般有两种方式，区别如下。本文档主要讨论通过 loghub 流式导入（实时）采集数据。

方式	优势	劣势	例子
批量导入	吞吐率大，面向历史存量数据	实时性较差	FTP、OSS 上传、邮寄硬盘、SQL 数据导出
流式导入	实时，所见即所得，面向实时数据	收集端要求高	loghub、HTTP 上传、IOT，Queue

背景

“我要点外卖”是一个平台型电商网站，涉及用户、餐厅、配送员等。用户可以在网页、App、微信、支付宝等进行下单点菜；商家拿到订单后开始加工，并自动通知周围的快递员；快递员将外卖送到用户手中。

运营需求

在运营的过程中，发现了如下的问题：

- 获取用户难，投放一笔不小的广告费到营销渠道（网页、微信推送），收获了一些用户，但无法评判各渠道的效果。
- 用户经常抱怨送货慢，但慢在什么环节，接单、配送还是加工，如何进行优化？

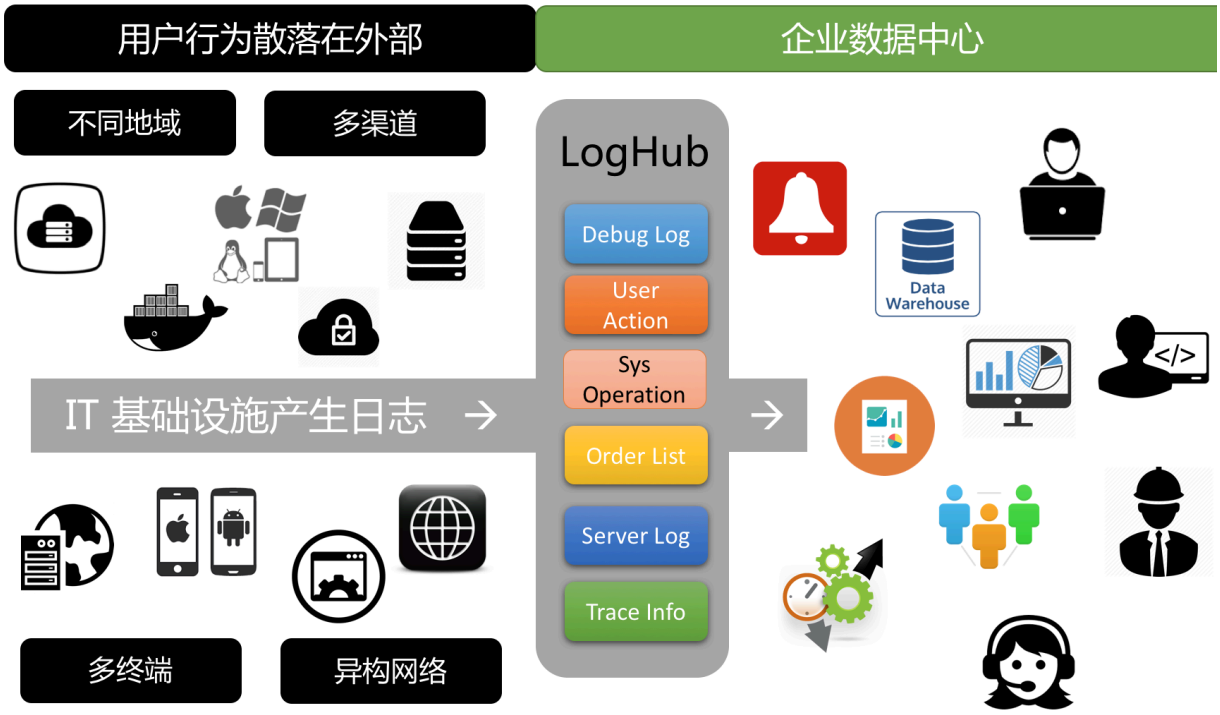
- 用户运营，经常搞一些优惠活动（发送优惠券），但无法获得效果。
- 调度问题，如何帮助商家在高峰时提前备货？如何调度更多的快递员到指定区域？
- 客服服务，用户反馈下单失败，用户背后的操作是什么？系统是否有错误？

数据采集难点

在数据化运营的过程中，第一步是如何将散落的日志数据集中收集起来，其中会遇到如下挑战：

- 多渠道：例如广告商、地推（传单）等
- 多终端：网页版、公众账号、手机、浏览器（web，m 站）等
- 异构网：VPC、用户自建 IDC，阿里云 ECS 等
- 多开发语言：核心系统 Java、前端 Nginx 服务器、后台支付系统 C++
- 设备：商家有不同平台（X86，ARM）设备

我们需要把散落在外部、内部的日志收集起来，统一进行管理。在过去这块需要大量的和不同种类的工作，现在可以通过 loghub 采集功能完成统一接入。



日志统一管理、配置

1. 创建管理日志项目，例如 myorder。

2. 为不同数据源产生的日志创建日志库，例如：

- wechat-server（存储微信服务器访问日志）
- wechat-app（存储微信服务器应用日志）
- wechat-error（错误日志）
- alipay-server
- alipay-app
- deliver-app（送货员 app 状态）
- deliver-error（错误日志）
- web-click（H5 页面点击）
- server-access（服务端 Access-Log）
- server-app (应用)
- coupon（应用优惠券日志）
- pay（支付日志）
- order（订单日志）

3. 如需要对原始数据进行清洗与 ETL，可以创建一些中间结果 logstore。

用户推广日志采集

为获取新用户，一般有两种方式：

- 网站注册时直接投放优惠券
- 其他渠道扫描二维码，投放优惠券
  - 传单二维码
  - 扫描网页二维码登陆

实施方法

定义如下注册服务器地址，生成二维码（传单、网页）供用户注册扫描。用户扫描该页面进行注册时，就可以得知用户是通过特定来源进入的，并记录日志。

```
http://examplewebsite/login?source=10012&ref=kd4b
```

当服务端接受请求时，服务器输出如下日志：

```
2016-06-20 19:00:00 e41234ab342ef034,102345,5k4d,467890
```

其中：

- time：注册时间。
- session：浏览器当前 session，用以跟踪行为。

- source：来源渠道。例如，活动 A 为 10001，传单为 10002，电梯广告为 10003。
- ref：推荐号，是否有人推荐注册，没有则为空。
- params：其他参数。

收集方式：

- 应用程序输出日志到硬盘，通过 logtail 采集。
- 应用程序通过 SDK 写入，参见 SDK。

服务端数据采集

支付宝/微信公众账号编程是典型的 Web 端模式，一般会有三种类型的日志：

- Nginx/Apache 访问日志：用以监控、实时统计

```
10.1.168.193 - - [01/Mar/2012:16:12:07 +0800] "GET /Send?AccessKeyI
d=8225105404 HTTP/1.1" 200 5 "-" "Mozilla/5.0 (X11; Linux i686 on
x86_64; rv:10.0.2) Gecko/20100101 Firefox/10.0.2"
```

- Nginx/Apache 错误日志

```
2016/04/18 18:59:01 [error] 26671#0: *20949999 connect() to unix:/
tmp/fastcgi.socket failed (111: Connection refused) while connecting
 to upstream, client: 10.101.1.1, server: , request: "POST /
logstores/test_log HTTP/1.1", upstream: "fastcgi://unix:/tmp/fastcgi
.socket:", host: "ali-tianchi-log.cn-hangzhou-devcommon-intranet.
sls.aliyuncs.com"
```

- 应用层日志：应用层日志要把事件产生的时间、地点、结果、延时、方法、参数等记录详细，扩展类字段一般放在最后

```
{
 "time":"2016-08-31 14:00:04",
 "localAddress":"10.178.93.88:0",
 "methodName":"load",
 "param":["31851502"],
 "result":....
 "serviceName":"com.example",
 "startTime":1472623203994,
 "success":true,
 "traceInfo":"88_1472621445126_1092"
}
```

- 应用层错误日志：错误发生的时间、代码行、错误码、原因等

```
2016/04/18 18:59:01 :/var/www/html/SCMC/routes/example.php:329
 [thread:1] errorcode:20045 message:extractFuncDetail failed:
account_hsf_service_log
```

实施方法

- 日志写到本地文件，通过 logtail 配置正则表达式写到指定 logstore 中。
- Docker 中产生的日志可以使用容器服务集成日志服务进行采集。

- Java 程序可以使用 Log4J Appender（日志不落盘），LogHub Producer Library（客户端高并发写入），Log4J Appender。
- C#、Python、Java、PHP、C 等可以使用SDK 写入。

终端用户日志接入

- 移动端：可以使用移动端 SDK IOS, Android 或 MAN（移动数据分析）接入。
- ARM 设备：ARM 平台可以使用 Native C 交叉编译。
- 商家平台设备：X86 平台设备可以用 SDK、ARM 平台可以使用 Native C 交叉编译。

Web/M 站页面用户行为

页面用户行为收集可以分为两类：

- 页面与后台服务器交互：例如下单，登陆、退出等。
- 页面无后台服务器交互：请求直接在前端处理，例如滚屏，关闭页面等。

实施方法

- 第一种可以参考服务端采集方法。
- 第二种可以使用 Tracking Pixel/JS Library 收集页面行为。

服务器日志运维

例如：

- Syslog 日志

```
Aug 31 11:07:24 zhouqi-mac WeChat[9676]: setupHotkeyListenning event
 NSEvent: type=KeyDown loc=(0,703) time=115959.8 flags=0 win=0x0
winNum=7041 ctxt=0x0 chars="u" unmodchars="u" repeat=0 keyCode=32
```

- 应用程序 Debug 日志

```
__FILE__:build/release64/sls/shennong_worker/ShardDataIndexManager.
cpp
__LEVEL__:WARNING
__LINE__:238
__THREAD__:31502
offset:816103453552
saved_cursor:1469780553885742676
seek count:62900
seek data redo
log:pangu://localcluster/redo_data/41/example/2016_08_30/250_147255
5483
```

```
user_cursor:1469780553885689973
```

- Trace 日志

```
[2013-07-13 10:28:12.772518]    [DEBUG] [26064]  __TRACE_ID__:
661353951201    __item__:[Class:Function]_end__  request_id:1734117
  user_id:124 context:.....
```

实施方法

参考服务端采集方法。

不同网络环境下的数据采集

loghub 在各 Region 提供访问点，每个 Region 提供三种方式接入：

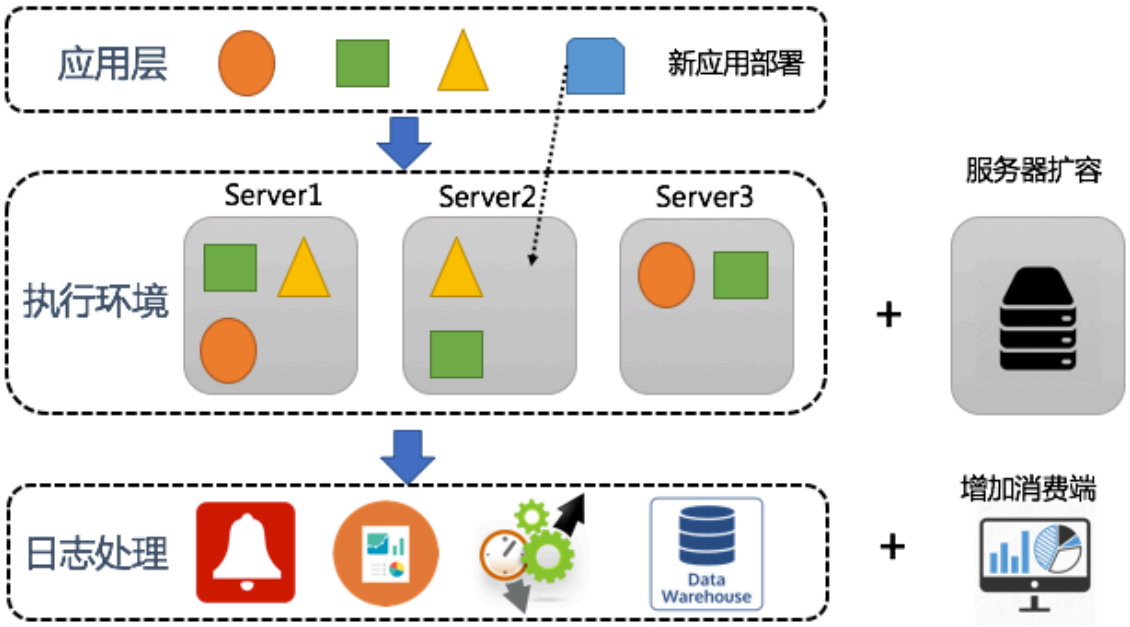
- 内网（经典网络）：本 Region 内服务访问，带宽链路质量最好（推荐）。
- 公网（经典网络）：可以被任意访问，访问速度取决于链路质量、传输安全保障建议使用 HTTPS。
- 私网（专有网络 VPC）：本 Region 内 VPC 网络访问。

2.7 采集-日志管理

日志管理

以下是一个典型的场景：服务器（容器）上有很多应用产生的日志数据，生成在不同的目录下。

- 开发会部署、下线新的应用。
- 服务器会根据需要水平扩展，例如在高峰时增加，低峰时减少。
- 根据不同需求，日志数据需要被查询、监控、仓库等，需求也在变化。



过程中的挑战

1. 应用部署上线速度快、日志种类越来越多

每个应用都包含访问日志（Access）、操作日志（OpLog）、业务逻辑（Logic）和错误（Error）等种类。当新增应用、应用与应用之间有依赖时，日志数目也会爆发。

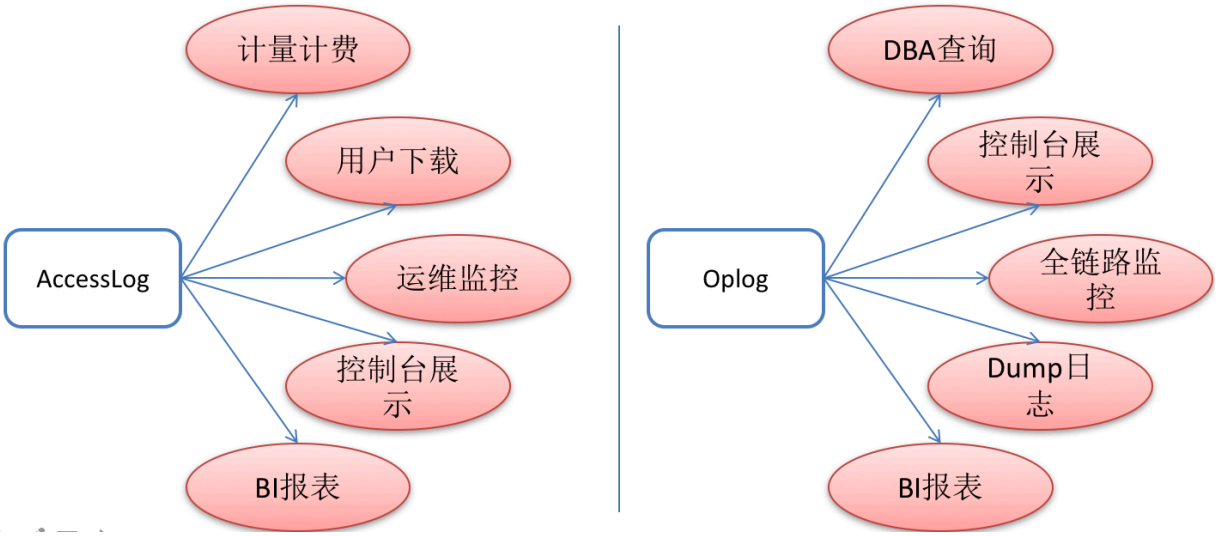
以下是一个外卖网站的例子：

分类	应用	日志名
Web	nginx	wechat-nginx (微信服务器 nginx 日志)
Web-Error	nginx	alipay-nginx （支付宝服务端 nginx 日志)
	nginx	server-access（服务端 Access-Log)
	nginx-error	alipay-nginx (nginx 错误日志)
	nginx-error	...
Web-App	tomcat	alipay-app (支付宝服务端应用逻辑)
	tomcat	...
App	Mobile App	deliver-app （送货员 app 状态)

分类	应用	日志名
App-Error	Mobile App	deliver-error （错误日志）
Web	H5	web-click （H5 页面点击）
server	server	服务端内部逻辑日志
Syslog	server	服务器系统日志

2. 日志被多个需求消费

例如，AccessLog 可以被用来计量计费、用户下载；OpLog 需要被 DBA 查询，需要做 BI、以及全链路监控等。



3. 环境与变化

互联网节奏非常快，在现实过程中，我们需要面对业务和环境的变化：

- 应用服务器扩容
- 服务器当机器
- 新增应用部署
- 新增日志消费者

一个理想的管理架构有如下需求

- 架构清晰，低成本
- 稳定高可靠、最好是无人值守（例如自动处理增减机器）
- 应用部署能够标准化，不需要复杂配置
- 日志处理需求能够被很容易地满足

日志服务解决方案

日志服务的loghub 功能在日志接入上定义如下概念，通过logtail方便地进行日志应用采集：

- 项目（Project）：管理容器
- 日志库（Logstore）：代表一类日志来源
- 机器组（MachineGroup）：代表日志产生目录、格式等
- 日志配置（Config）：标示日志产生的路径

这些概念关系如下：

- 一个项目包括多个 logstore，machineGroup 和 config，通过不同项目满足不同业务间需求。
- 一个应用可以有多种日志，每种日志一个 logstore，每种日志有一个固定的目录产生（config 相同）。

```
app --> logstore1, logstore2, logstore3
app --> config1, config2, config3
```

- 一个应用可以部署在多个机器组上，一个机器组可以部署多个应用。

```
app --> machineGroup1,  mahcineGroup2
machineGroup1 --> app1, app2, app3
```

- 将配置（config）定义的收集目录、应用到机器组上，收集到任意 logstore。

```
config1 * machineGroup1 --> Logstore1
config1 * machineGroup2 --> logstore1
config2 * machineGroup1 --> logstore2
```

优势

- 便捷：提供 WebConsole/SDK 等方式进行批量管理
- 大规模：可以管理百万级机器和百万级应用
- 实时：收集配置分钟内生效
- 弹性：
  - 通过机器标识功能支持服务器弹性扩容
  - loghub支持弹性伸缩
- 稳定可靠：无需人工干预

- 日志处理中实时计算、离线分析、索引等查询功能
  - 日志中枢（loghub）：实时采集与消费。通过 30+ 方式实时采集海量数据、下游实时消费。
  - 日志投递（logshipper）：稳定可靠的日志投递。将日志中枢数据投递至存储类服务（OSS/MaxCompute/Table Store)进行存储与大数据分析。
  - 日志查询（logsearch）：实时索引、查询数据。日志统一查询，不用关心线上服务器日志位置。

3 查询分析

3.1 查询分析-分页

查询日志时，查询结果内容过多会影响显示速度和查询体验。通过API接口进行分页查询可以指定查询结果分页显示，控制每次返回的数据量。

日志服务提供了分页的功能，不仅可以分页读取原始日志内容，也可以把SQL的计算结果分页读取到本地。开发者可以通过日志服务提供的SDK或者CLI，通过读数据接口分页读取日志。

分页方式

日志服务查询和分析功能支持在查询分析语句中同时实现关键字查询和查询结果的SQL分析。GetLogstoreLogs接口是日志服务提供的日志查询入口，您既可以根据关键字查询日志原始内容，也可以进行SQL计算，获取分析结果。查询分析语句中的查询部分和分析部分使用不同的分页方式。

- 查询语句：使用关键字查询，获取原始日志内容。通过API中的参数offset和lines来分页获取所有内容。
- 分析语句：使用SQL对查询结果进行分析，获取统计结果。通过SQL的limit语法来达到分页效果。

查询结果分页

在GetLogStoreLogs API中，参数offset和lines可以用来设置分页。

- offset：指定从第一行开始读取日志。
- lines：指定当前的请求读取多少行，该参数最大可设置为100，如果设置该参数大于100，则仍然返回100行。

在分页读取时，不停的增大offset，直到读取到某个offset后，获取的结果行数为0，并且结果的progress为complete状态，则认为读取到了全部数据。

查询分页代码示例

- 分页的伪代码：

```
    offset = 0// 从第0行开始读取
    lines = 100//每次读取100行
    query = "status:200"//查询status字段包含200的所有日志whileTrue:
    response = get_logstore_logs(query, offset, lines) // 执行读取请求
    process (response)                                  //调用自定义逻
辑，处理返回的结果
    如果 response.get_count() == 0 && response.is_complete()
         则读取结束，跳出当前循环
```

```
    否则
        offset += 100                                   //offset增加100
，读取下一个100行
```

- Python分页读取：

  详细案例请参考#unique_40。

```
    endpoint = ''# 选择与上面步骤创建Project所属区域匹配的Endpoint
    accessKeyId = ''# 使用您的阿里云访问密钥AccessKeyId
    accessKey = ''# 使用您的阿里云访问密钥AccessKeySecret
    project = ''# 上面步骤创建的项目名称
    logstore = ''# 上面步骤创建的日志库名称
    client = LogClient(endpoint, accessKeyId, accessKey)
    topic = ""
    query = "index"
    From = int(time.time()) - 600
    To = int(time.time())
    log_line = 100
    offset = 0
    while True:
        res4 = Nonefor retry_time in range(0, 3):
            req4 = GetLogsRequest(project, logstore, From, To, topic
, query, log_line, offset, False)
            res4 = client.get_logs(req4)
            if res4 isnotNoneand res4.is_completed():
                break
            time.sleep(1)
        offset += 100if res4.is_completed() && res4.get_count() == 0
:
              break;
        if res4 isnotNone:
            res4.log_print()  # 这里处理结果
```

- Java分页读取：

  更详细的案例参考#unique_41。

```
        int log_offset = 0;
        int log_line = 100;//log_line 最大值为100，每次获取100行数据。若
需要读取更多数据，请使用offset分页。offset和lines只对关键字查询有效，若使用SQL
查询，则无效。在SQL查询中返回更多数据，请使用limit语法。
        while (true) {
            GetLogsResponse res4 = null;
            // 对于每个 log offset,一次读取 10 行 log，如果读取失败，最多重
复读取 3 次。
            for (int retry_time = 0; retry_time < 3; retry_time++) {
                GetLogsRequest req4 = new GetLogsRequest(project,
logstore, from, to, topic, query, log_offset,
                        log_line, false);
                res4 = client.GetLogs(req4);

                if (res4 != null && res4.IsCompleted()) {
                    break;
                }
                Thread.sleep(200);
            }
            System.out.println("Read log count:" + String.valueOf(
res4.GetCount()));
            log_offset += log_line;
            if (res4.IsCompleted() && res4.GetCount() == 0) {
                        break;
```

```
            }
        }
```

分析结果分页

GetLogStoreLogs API参数中的offset和lines不支持用于SQL分析。如果按照上文分页读取原始内容的方式，遍历offset分页，那么每次SQL执行的结果都是一样的。如果在一次调用中获取全部的计算结果，可能会由于结果集太大而产生以下问题：

- 网络上传输大量数据延时比较高。
- 客户端的内存要保存大量的结果以供进一步处理，影响内存占用率。

为了解决SQL分页的问题，日志服务提供了标准SQL的limit分页语法：

```
limit Offset, Line
```

- Offset表示从第几行开始读取结果。
- Line表示读取多少行。Line没有大小限制，但是如果一次读取太多，会影响网络延时和客户端的处理速度。

例如，分析语句`* | selectcount(1) , url group by url` 返回2000条日志，可以通过分页指定每次读取500行，共4次读取完成：

```
* | selectcount(1) , url  group by url  limit 0, 500
* | selectcount(1) , url  group by url  limit 500, 500
* | selectcount(1) , url  group by url  limit 1000, 500
* | selectcount(1) , url  group by url  limit 1500, 500
```

分析分页代码示例

- SQL分页的伪代码：

```
    offset = 0// 从第0行开始读取
    lines = 500//每次读取500行
    query = "* | select count(1) , url  group by url  limit "
whileTrue:
    real_query = query + offset + "," +  lines
    response = get_logstore_logs(real_query) // 执行读取请求
    process (response)                       //调用自定义逻辑，处理返回的结果
    如果 response.get_count() == 0
         则读取结束，跳出当前循环
    否则
        offset += 500                        //offset增加100，读取下一个500行
```

- Python程序代码：

```
    endpoint = ''# 选择与上面步骤创建Project所属区域匹配的Endpoint
    accessKeyId = ''# 使用您的阿里云访问密钥AccessKeyId
    accessKey = ''# 使用您的阿里云访问密钥AccessKeySecret
```

```
    project = ''# 上面步骤创建的项目名称
    logstore = ''# 上面步骤创建的日志库名称
    client = LogClient(endpoint, accessKeyId, accessKey)
    topic = ""
    origin_query = "* | select count(1) , url  group by url  limit "
    From = int(time.time()) - 600
    To = int(time.time())
    log_line = 100
    offset = 0
    while True:
        res4 = None
        query = origin_query + str(offset) + " , " + str(log_line)
        for retry_time in range(0, 3):
            req4 = GetLogsRequest(project, logstore, From, To, topic
, query)
            res4 = client.get_logs(req4)
            if res4 isnotNoneand res4.is_completed():
                break
            time.sleep(1)
        offset += 100if res4.is_completed() && res4.get_count() == 0
:
              break;
        if res4 isnotNone:
            res4.log_print()  # 这里处理结果
```

- Java程序代码：

```
        int log_offset = 0;
        int log_line = 500;
        String origin_query = "* | select count(1) , url  group by
url  limit "while (true) {
            GetLogsResponse res4 = null;
            // 对于每个 log offset,一次读取 500 行 log，如果读取失败，最多
重复读取 3 次。
            query = origin_query + log_offset + "," + log_line;
            for (int retry_time = 0; retry_time < 3; retry_time++) {
                GetLogsRequest req4 = new GetLogsRequest(project,
logstore, from, to, topic, query);
                res4 = client.GetLogs(req4);

                if (res4 != null && res4.IsCompleted()) {
                    break;
                }
                Thread.sleep(200);
            }
            System.out.println("Read log count:" + String.valueOf(
res4.GetCount()));
            log_offset += log_line;
            if (res4.GetCount() == 0) {
                        break;
            }

        }
```

3.2 分析-销售系统日志

成交账单是电商公司的核心数据，是一系列营销和推广活动最终的转化成果。这些数据包含了很多有价值的信息：从这些数据出发，可以描绘出用户画像，为下一步的营销提供方向。账单数据还能提供货物的受欢迎程度，为下一步备货提供准备。

账单信息以日志的形式保存在阿里云日志服务上，日志服务能够提供快速的查询和SQL统计，在秒级别计算上亿条日志。本文将以几个例子来讲解如何挖掘有用信息。

一个完整的成交账单，包含了货物的信息(名称、价格)、成交的价格信息(成交的价格、支付手段、优惠信息)、交易对手的信息(会员信息)，账单日志样例：

```
__source__:  10.164.232.105       __topic__:  bonus_discount:  category
:  男装     commodity:  天天特价秋冬款青少年加绒加厚紧身牛仔裤男士冬季修身型小脚
裤子 commodity_id:  443 discount:  member_discount:  member_level:
nomember_point:  memberid:  mobile:  pay_transaction_id:  060f0e0d08
0e0b05060307010c0f0209010e0e010c0a0605000606050b0c0400 pay_with:
alipay real_price:  52.0 suggest_price:  52.0
```

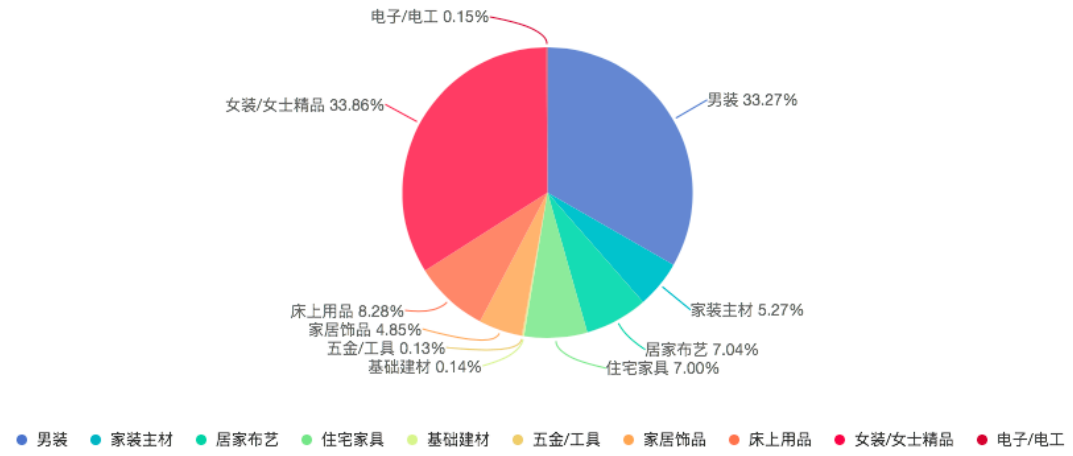
		时间	category	commodity	discount	member_discount	member_level	mobile	pay_with	real_price	suggest_price
1		08-31 20:12:02	住宅家具	小户型沙发宜家双人沙发两人咖啡厅卡座网吧沙发布艺单人沙发椅		3.19	gold	1398187088	cash	524.81	528.0
2		08-31 20:12:02	男装	天天特价秋季长袖衬衫韩版男士条纹衬衣潮流男装青年休闲格子寸衫		0.59	gold	1398185946	cash	42.41	43.0
3		08-31 20:12:02	男装	2016秋冬季新款男士横须贺棒球服夹克古着两面穿情侣修身刺绣外套		0.99	gold	1398187371	cash	129.01	130.0
4		08-31 20:12:02	女装/女士精品	S家原创女装秋冬季2016新款欧美加绒大衣中长款连帽加厚风衣外套		8.31	gold	1398181413	cash	323.79	332.1
5		08-31 20:12:02	女装/女士精品	冬季女装外套保暖棉衣宽松中长款连帽面包服黑色马甲女 收货返30%	0.1	1.47	gold	1398182131	cash	141.060477548	166.0
6		08-31 20:12:02	女装/女士精品	女装秋装2016新款潮中长款七分袖针织连衣裙秋季撞色条纹长裙春秋		0.89	gold	1398181717	cash	110.21	111.1
7		08-31 20:12:02	女装/女士精品	【双11特供款】九月陌墨 2016秋季新款纯色包边棉衣女宽松外套		3.4	gold	1398181935	cash	225.6	229.0
8		08-31 20:12:02	男装	马切达原创冬季新款夹棉夹克韩版纯色休闲连帽棉衣日系修身外套男	0.1	0.45	gold	1398180217	cash	150.75	168.0
9		08-31 20:12:02	女装/女士精品	短款女外套2016秋季韩版宽松鹿皮绒短上衣加厚羊羔毛休闲短外套冬		0.96	gold	1398184069	cash	91.5143800227	97.0

统计分析

要对数据进行查询和分析，请先#unique_43。

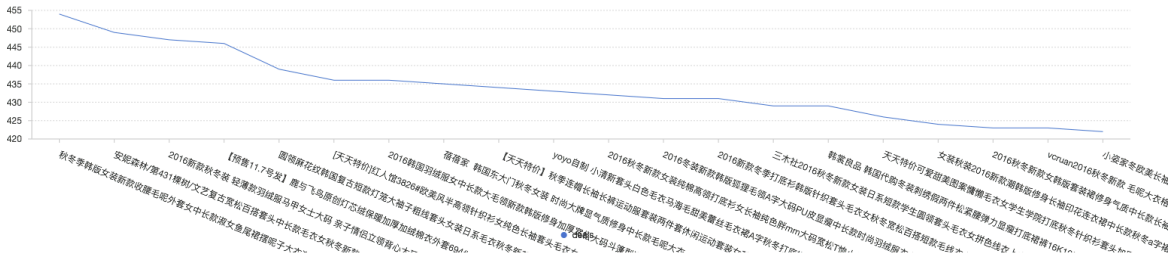
1. 查看产品的销售占比

```
*|select count(1) as pv ,category group by category limit 100
```



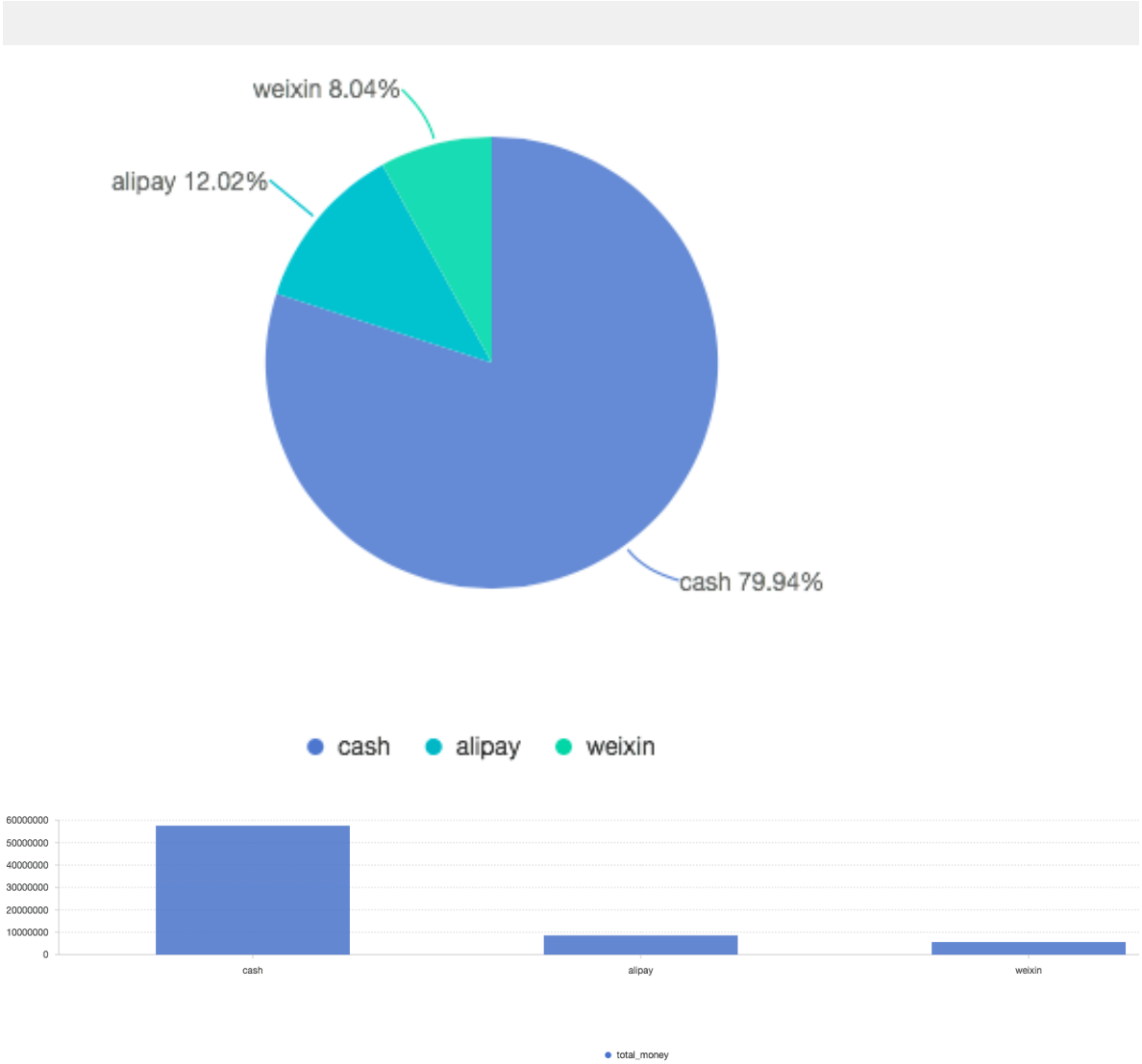
2. 查看女装销售

```
category:  女装/女士精品  | select count(1) as deals , commodity
 group by commodity order by deals  desc limit 20
```



3. 不同支付方式所占份额、成交额

```
*  | select count(1) as deals , pay_with group by pay_with  order by
 deals  desc limit 20

*  | select sum(real_price)  as total_money , pay_with
 group by pay_with  order by total_money  desc limit 20
```

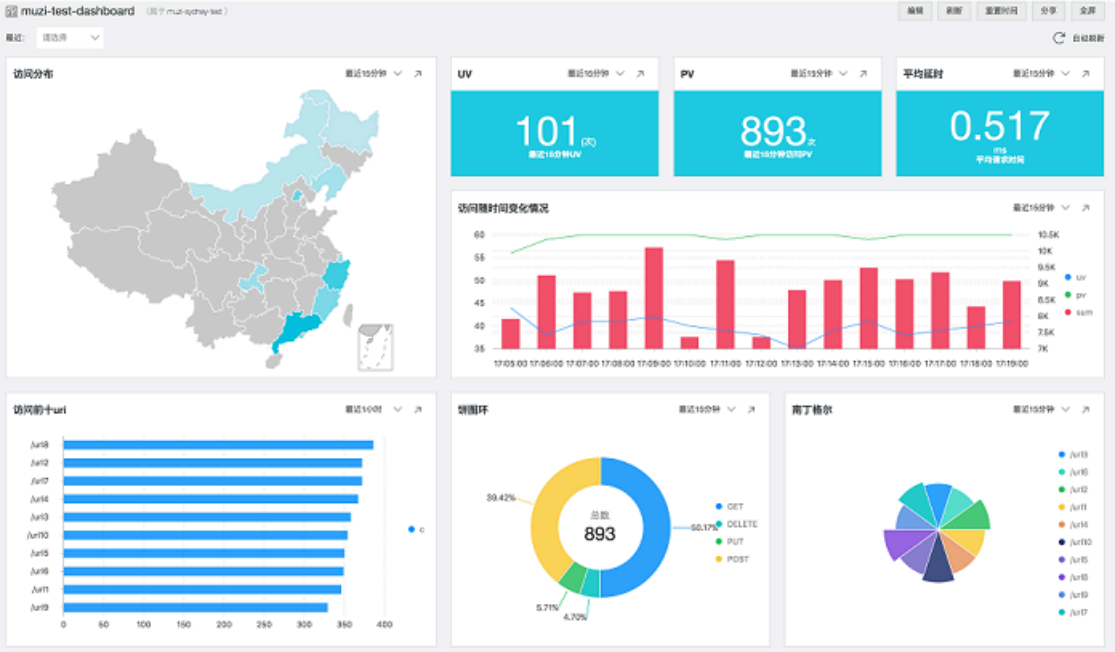


3.3 分析-网站日志

网站的访问日志信息对于个人站长来说至关重要，其中包含的PV、UV、访问地域分布以及访问前十页面等情况是网站访问情况的数据体现。应用的日志信息对于应用的开发者来说必不可少，针对Top方法的分析、优化可以直接提升应用质量。运维人员通过服务器日志可以监控数据、追溯异常，通过实时监控日志数据，可以获取最近1个小时的服务器响应时间变化、请求客户端负载均衡到某一台机器流量是否有异常情况等信息，并通过数据大屏展示日志监控数据，可以直观获取关键信息。

基于以上场景，日志服务提供多样化的日志数据采集、分析一站式解决方案，其中实时分析功能（LogSearch/Analytics），可以使用查询+SQL92语法对日志进行实时分析，并且可以在查询分析结果上支持自带Dashboard、DataV、Grafana、Tableua(通过JDBC)、QuickBI等可视化方式。

更为便捷的是，日志服务提供数据分析图表，即对日志的实时检索分析结果进行图表方式的直观展示，并通过仪表盘功能为您创建多种场景下的日志数据分析大盘。



点我试用

用户名：sls_reader1@**********

密码：pnX-32m-MHH-xbm

功能特点

- 无需事先定义：任何计算方法、任何过滤条件可以应用到任意时间段，秒级出图。
- 交互式分析：图表和原始日志无缝切换，双向打通。
- 场景化支持：通过数据接入向导直接生成分析大盘，无需复杂配置。

图标类型

目前，日志服务提供的可视化功能包含了如下图表类型：

流程架构

1. 数据采集。日志服务支持客户端、网页、协议、SDK/API（移动、游戏）等多种日志采集方式，所有采集方式均基于Restful API实现，除此之外您也可以通过API/SDK实现新的采集方式。
2. 设置数据索引并采用查询分析语法进行查询分析。
3. 可视化展示。日志服务提供基于Restful的开放式API，我们可以选择适合的方式对我们的日志数据进行可视化处理，本文档主要演示日志服务自带的可视化以及仪表盘（Dashboard）功能。

示例

本文档为您展示各种图形的典型应用场景。要对数据进行查询和分析，请先#unique_43。

1. 表格

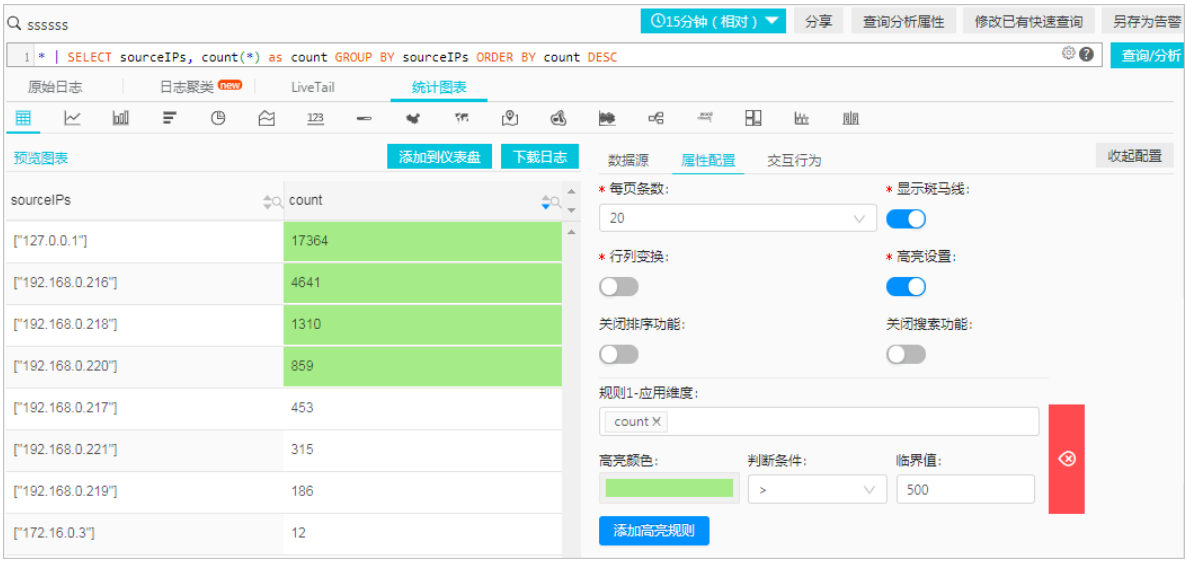
表格作为最常见的数据展示类型，由一组或多组单元格组成，用于显示数字和其他项以便快速引用和分析，表格中的项被组织为行和列，表格的第一行称为表头，指明表格每一列的内容和意义。

在日志服务中，我们通过查询分析语法得到的结果信息默认以表格方式进行展示。

例如，查看当前时间区间sourceIPs分布情况，并降序排列：

```
* | SELECT sourceIPs, count(*) as count GROUP BY sourceIPs ORDER BY
count DESC
```

表格结果如下所示，您可以利用表头上的排序按钮对某一列进行排序。



2. 折线图

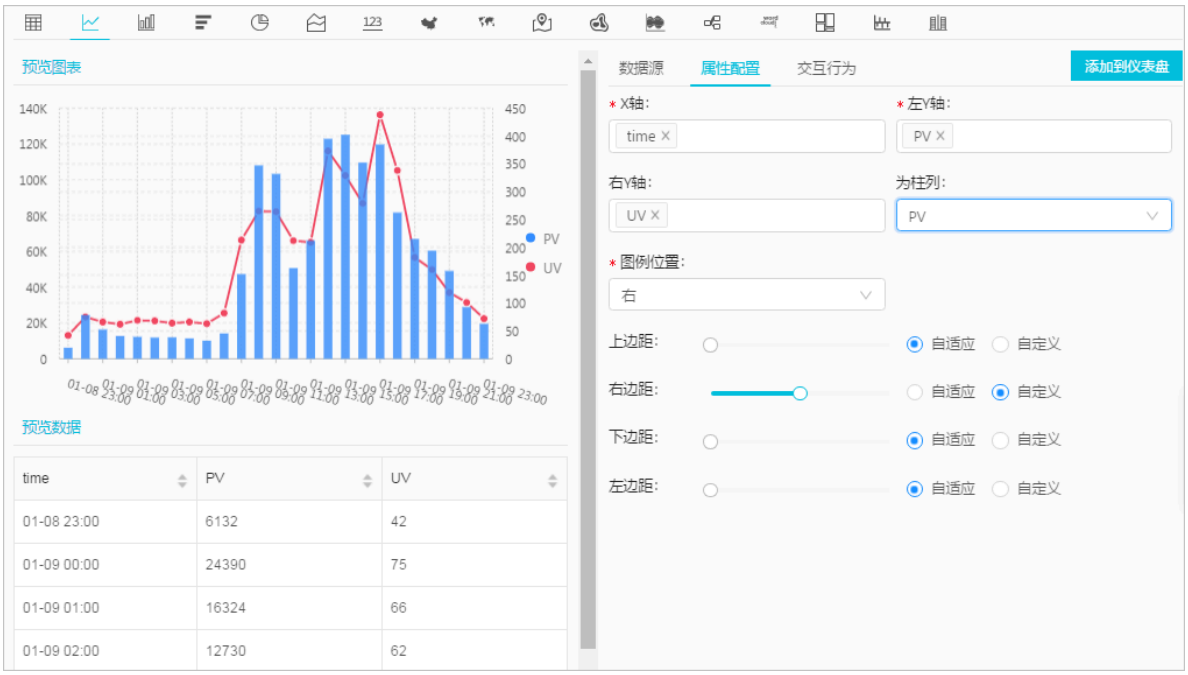
折线图属于趋势类分析图表，一般用于表示一组数据在一个有序数据类别（多为连续时间间隔）上的变化情况，用于直观分析数据变化趋势。

分析在最近15分钟内PV、UV以及平均响应时间的变化：

```
* | select date_format(from_unixtime(__time__ - __time__% 60), '%H:%
i:%S') as minutes, approx_distinct(remote_addr) as uv, count(1) as
```

```
pv, avg(request_time) as avg group by minutes order by minutes asc
limit 100000
```

选择minutes作为X轴，pv、uv放在左Y轴，avg为右Y轴并且设置uv为柱状，结果如下图所示：

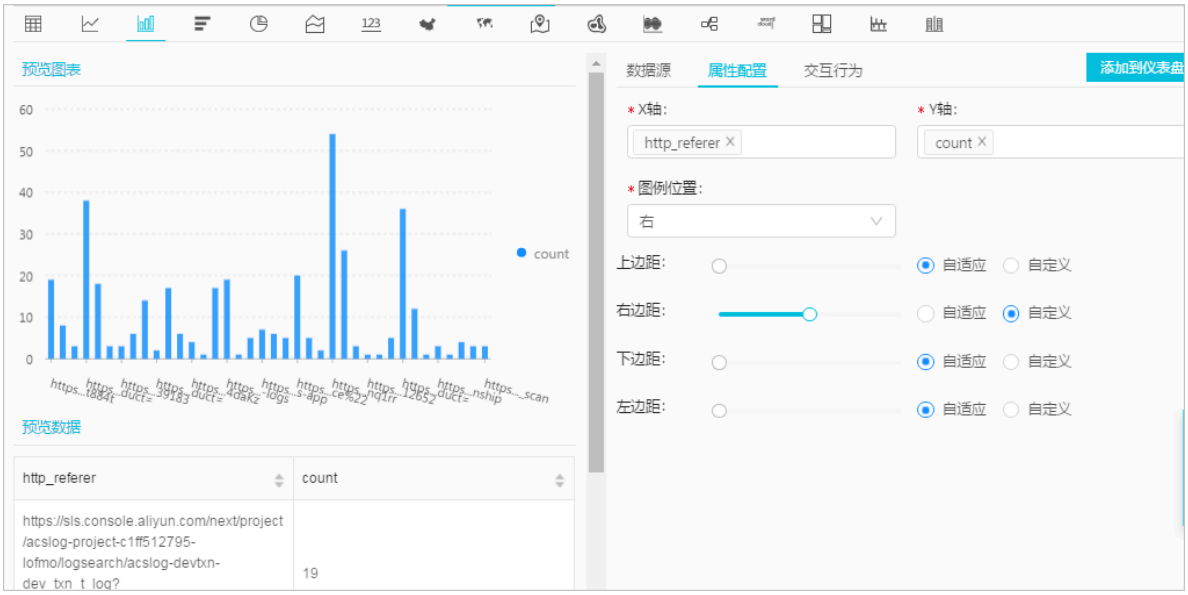


3. 柱状图

柱状图使用垂直或水平的柱子显示类别之间的数值比较，和#unique_46的不同之处在于，柱状图描述分类数据，并统计每一个分类中的数量，而折线图描述有序数据。

分析最近15分钟内不同`http_referer`的访问次数：

```
* | select http_referer, count(1) as count group by http_referer
```

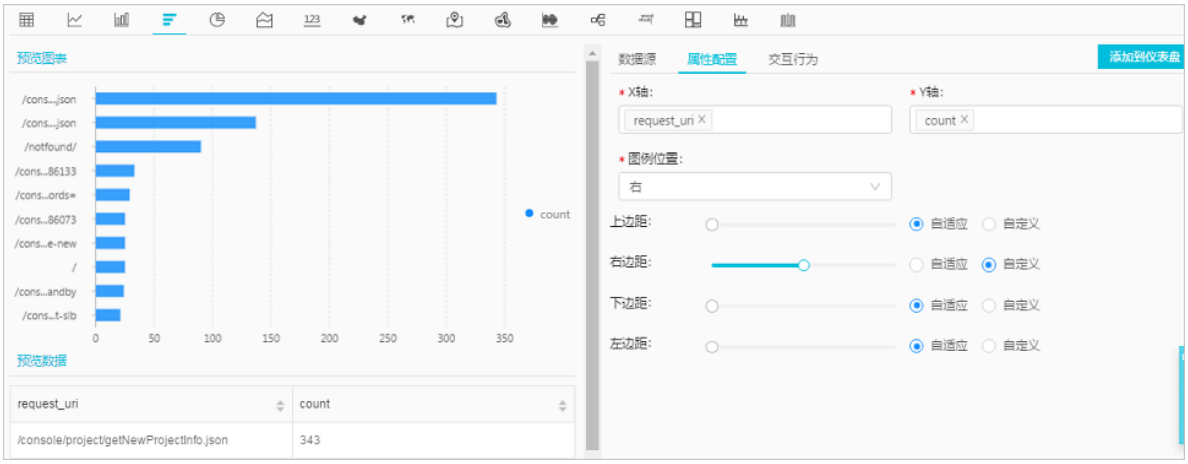


4. 条形图

条形图即为横向柱状图，适合分析分类数据的top情况。

分析最近15分钟内访问前十的`request_uri`：

```
* | select  request_uri, count(1) as count group by request_uri
order by count desc limit 10
```



5. 饼图

饼图用于表示不同分类的占比情况，通过弧度大小来对比各种分类。饼图通过将一个圆饼按照分类的占比划分成多个区块，整个圆饼代表数据的总量，每个区块（圆弧）表示该分类占总体的比例大小，所有区块（圆弧）的加和等于100%。

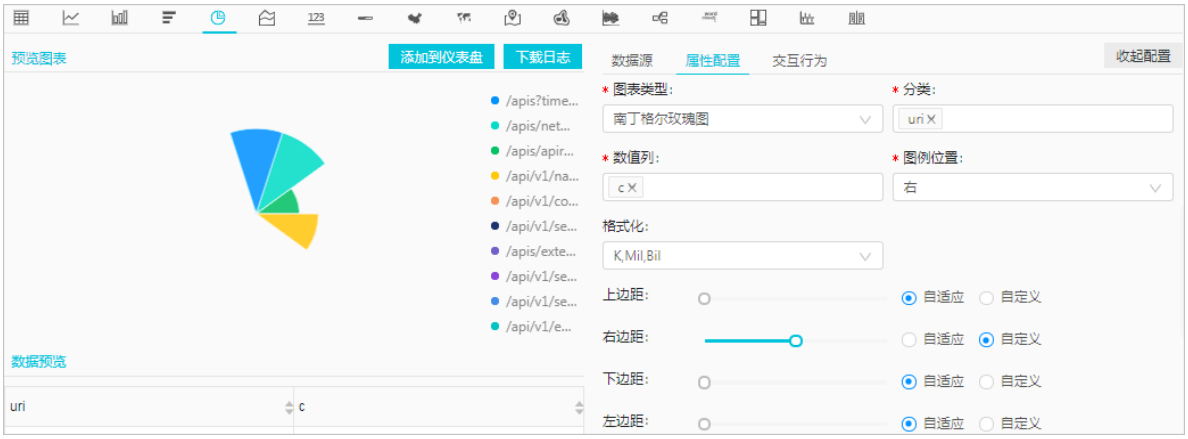
分析最近15分钟访问页面的分布：

```
* | select requestURI as uri , count(1) as c group by uri limit 10
```

饼图：

环图：

南丁格尔玫瑰图：

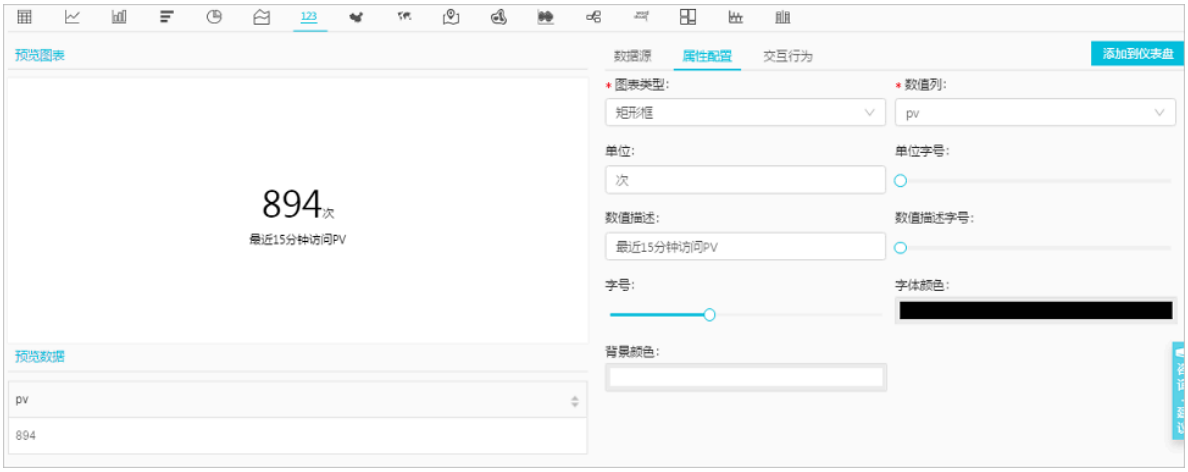


6. 单值图

单值图作为最简单直接的数据表现形式，直观清晰地将某一个点上的数据展示出来，一般用于表示某一个时间点上的关键信息。

统计最近15分钟的PV：

```
* | select count(1) as PV
```

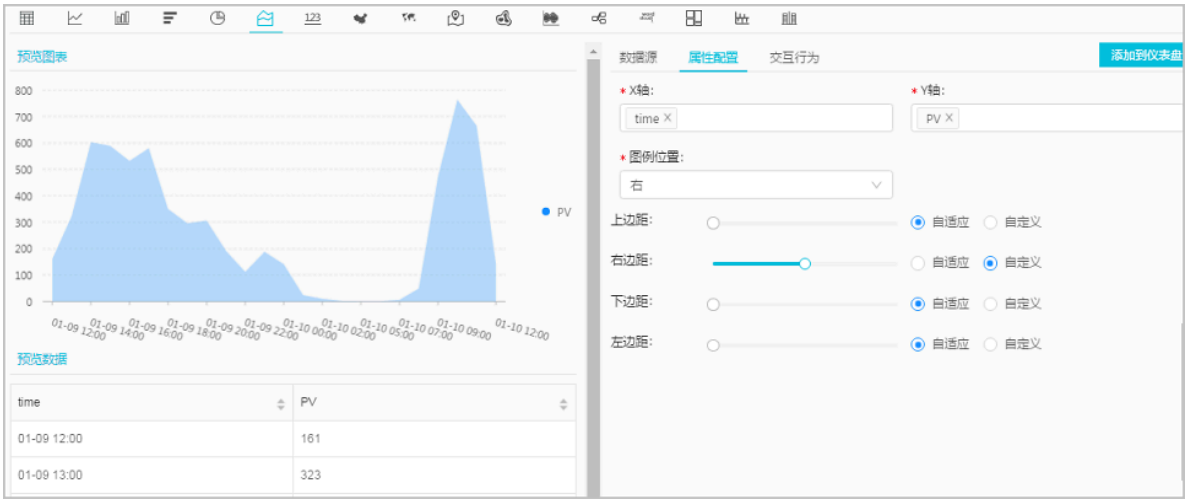


7. 面积图

面积图是在折线图的基础之上形成的，它将折线图中折线与坐标轴之间的区域使用颜色进行填充，这个填充即为面积，颜色的填充可以更好的突出趋势信息。

如统计`10.0.XX.XX`这个IP在最近1天内的访问情况：

```
remote_addr: 10.0.XX.XX | select date_format(date_trunc('hour',
__time__), '%m-%d %H:%i') as time, count(1) as PV group by time
order by time limit 1000
```



8. 地图

以地图作为背景，通过图形颜色、图像标记的方式展示地理数据信息。日志服务提供了三种地图方式，分别为：中国地图、世界地图以及高德地图（高德地图分为点图和热力图）。

通过remote_addr来绘制三种地图，统计前十的访问区域：

- 中国地图

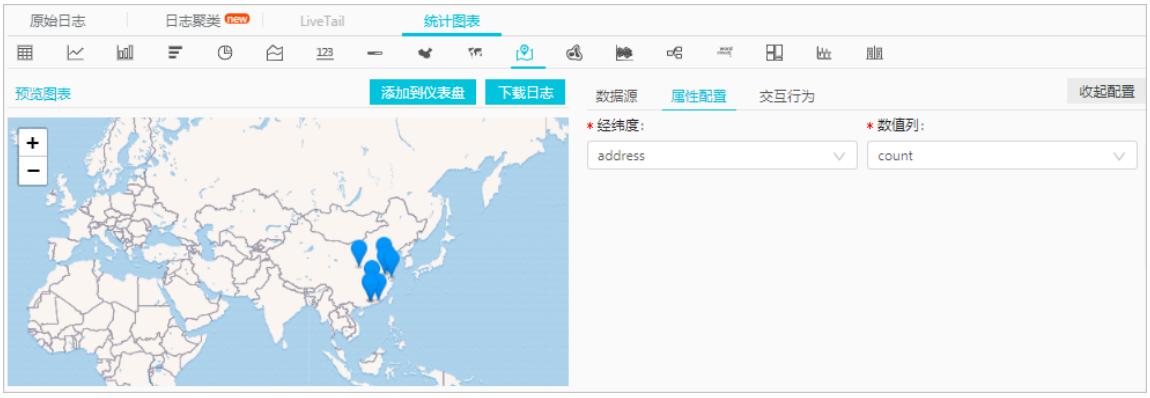
```
* | select ip_to_province(remote_addr) as address, count(1) as
count group by address order by count desc limit 10
```

- **世界地图**

```
* | select ip_to_country(remote_addr) as address, count(1) as
count group by address order by count desc limit 10
```

- **高德地图**

```
* | select ip_to_geo(remote_addr) as address, count(1) as count
group by address order by count desc limit 10
```


原始日志
日志聚类 new
LiveTail
统计图表
预览图表
添加到仪表盘
下载日志
数据源
属性配置
交互行为
收起配置
* 经纬度:
address
* 数值列:
count


原始日志
日志聚类 new
LiveTail
统计图表
预览图表
添加到仪表盘
下载日志
数据源
属性配置
交互行为
收起配置
* 经纬度:
address
* 数值列:
count

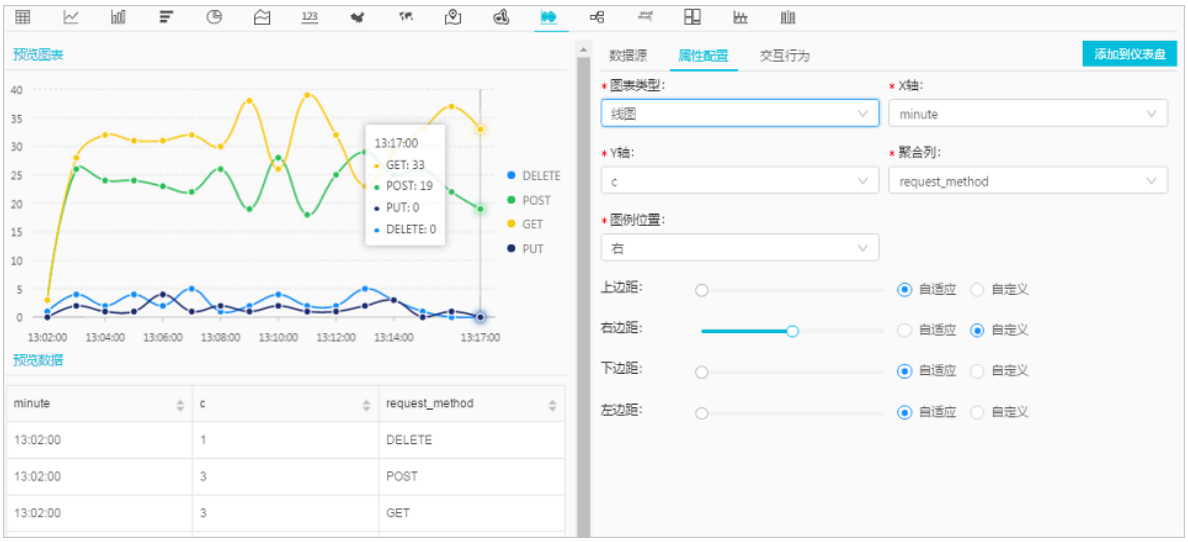
9. 流图

不同颜色的条带状分支代表了不同的分类信息，条状带的宽度映射了对应的数值大小。此外，原数据集中的时间属性，映射到X轴上，是一个三维关系的展现。

统计最近15分钟，不同method方法请求次数随时间变化趋势情况：

```
* | select date_format(from_unixtime(__time__ - __time__% 60), '%H
:%i:%S') as minute, count(1) as c, request_method group by minute,
request_method order by minute asc limit 100000
```

X轴选择minute，Y轴选择c，按照request_method聚合。

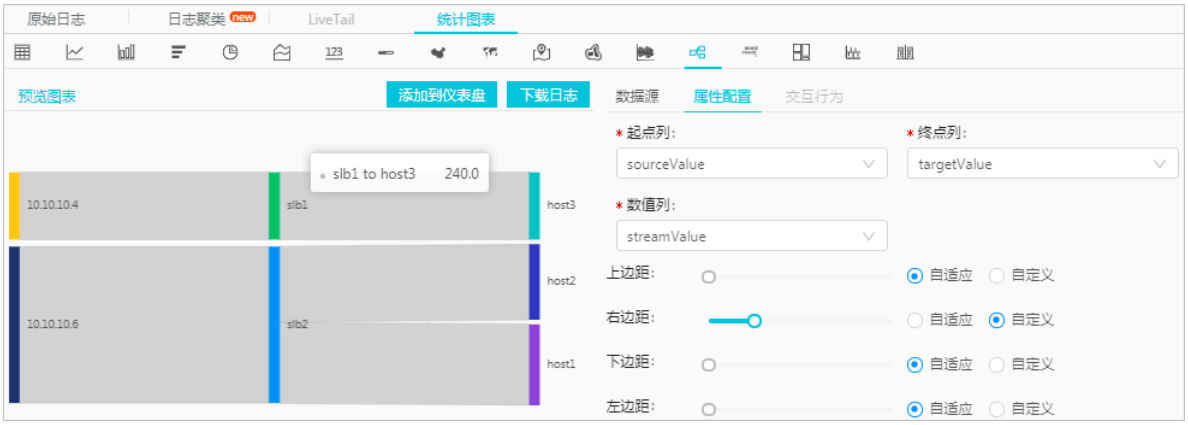


10.桑基图

桑基图 (Sankey Diagram)，是一种特定类型的流图，用于描述一组值到另一组值的流向。适合网络流量等场景，通常包含3组值source、target以及value。source和target描述了点的关系，而value描述了该source和target之间边的关系。

负载均衡场景示例：

```
* | select sourceValue, targetValue, streamValue group by sourceValu
e, targetValue, streamValue order by streamValue
```

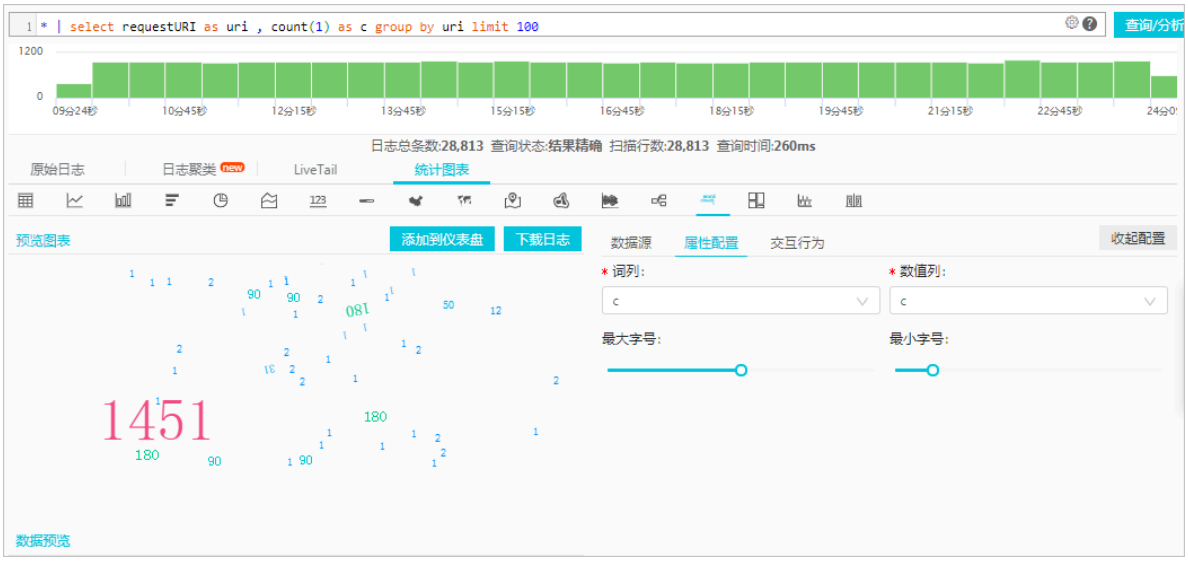


11.词云

词云，是文本数据的视觉表示，由词汇组成类似云的彩色图形，用于展示大量文本数据。每个词的重要性以字体大小或颜色显示，为您直观展示某一些关键词的权重大小。

统计最近15分钟访问requestURI的情况：

```
* | select requestURI as uri , count(1) as c group by uri limit 100
```



添加到仪表盘

所有通过查询分析语法获得的可视化图表都可以保存在一个仪表盘（Dashboard）中，再经过灵活的布局调整，就可以做出一张全面的仪表盘了。

您可以单击添加到仪表盘，建立一个仪表盘。建立仪表盘后可以通过标签快速打开仪表盘，实时查看数据。

演示视频：

3.4 分析-Nginx监控日志

Nginx和php-fpm、Docker、Apache等很多软件一样内建了一个状态页，对于Nginx的状态查看以及监控提供了很大帮助。本文档主要介绍通过日志服务Logtail采集Nginx status信息，并对采集的status信息进行查询、统计、搭建仪表盘、建立自定义报警，对您的Nginx集群进行全方位的监控。

环境准备

请按照以下步骤，开启Nginx status插件。

1. 确认Nginx是否具备status功能。

   执行以下命令查看Nginx是否具备status功能：

   ```
   nginx -V 2>&1 | grep -o with-http_stub_status_module
   with-http_stub_status_module
   ```

   如果回显信息为`with-http_stub_status_module`，表示支持status功能。

2. 配置Nginx status。

   在Nginx的配置文件（默认为/etc/nginx/nginx.conf）中开启status功能，样例配置如下：

   ```
       location /private/nginx_status {
         stub_status on;
         access_log   off;
         allow 10.10.XX.XX;
         deny all;
       }
   ```

    说明：

   该配置只允许ip为`10.10.XX.XX`的机器访问`nginx status`功能。

3. 验证Logtail安装的机器具有`nginx status`访问权限。

   可通过如下命令测试：

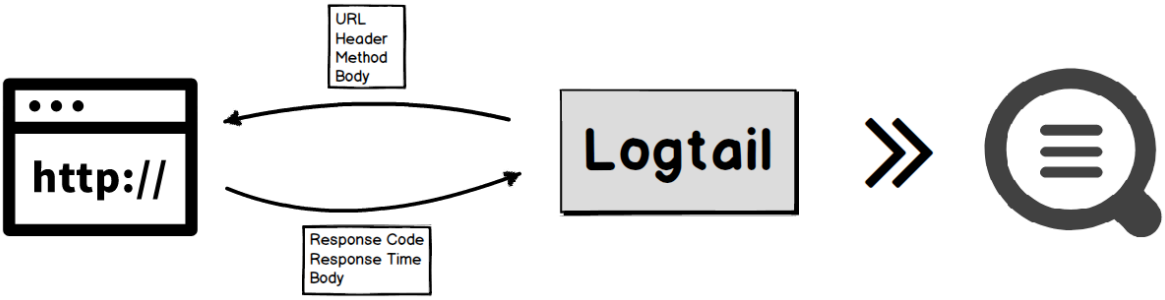
   ```
   $curl http://10.10.XX.XX/private/nginx_status
   Active connections: 1
   server accepts handled requests
   2507455 2507455 2512972
   Reading: 0 Writing: 1 Waiting: 0
   ```

数据采集

1. 安装Logtail

   安装logtail，确认版本号在0.16.0及以上。若低于0.16.0版本请根据文档提示升级到最新版本。

2. 填写采集配置



a. 在日志服务控制台创建一个新的Logstore，采集向导中选择自建软件中的Nginx监控。

b. 根据提示配置Nginx监控的URL以及相关参数（基于http采集功能实现）。

> 说明:
>
> · 将样例配置中Addresses字段内容修改为您需要监控的url列表。
>
> · 如果您的Nginx status返回的信息和默认的不同，请修改processors用以支持http的body解析。

样例配置如下:

```
{
"inputs": [
 {
      "type": "metric_http",
      "detail": {
          "IntervalMs": 60000,
          "Addresses": [
              "http://10.10.XX.XX/private/nginx_status",
              "http://10.10.XX.XX/private/nginx_status",
              "http://10.10.XX.XX/private/nginx_status"
          ],
          "IncludeBody": true
      }
 }
],
"processors": [
 {
      "type": "processor_regex",
      "detail": {
          "SourceKey": "content",
          "Regex": "Active connections: (\\d+)\\s+server accepts
handled requests\\s+(\\d+)\\s+(\\d+)\\s+(\\d+)\\s+Reading: (\\d+)
Writing: (\\d+) Waiting: (\\d+)[\\s\\S]*",
          "Keys": [
              "connection",
              "accepts",
              "handled",
              "requests",
              "reading",
              "writing",
              "waiting"
          ],
          "FullMatch": true,
```

```
            "NoKeyError": true,
            "NoMatchError": true,
            "KeepSource": false
        }
    }
]
}
```

数据预览

应用配置1分钟后，点击预览可以看到状态数据已经成功采集。Logtail的http采集除了将body解析上传，还会将url、状态码、方法名、响应时间、是否请求成功一并上传。

说明:

若无数据，请先检查配置是否为合法json。

```
_address_:http://10.10.XX.XX/private/nginx_status
_http_response_code_:200
_method_:GET
_response_time_ms_:1.83716261897
_result_:success
accepts:33591200
connection:450
handled:33599550
reading:626
requests:39149290
waiting:68
writing:145
```

查询分析

要对数据进行查询和分析，请先#unique_43。

自定义查询

查询相关帮助文档参见#unique_9。

1. 查询某一ip的status信息： `_address_ : 10.168.0.0`
2. 查询响应时间超过100ms的请求: `_response_time_ms_ > 100`
3. 查看状态码非200的请求: `not _http_response_code_ : 200`

统计分析

统计分析语法参见#unique_13。

- 每5分钟统计 waiting reading writing connection 平均值：

```
*| select  avg(waiting) as waiting, avg(reading)  as reading,  avg
(writing)  as writing,  avg(connection)  as connection,  from_unixt
ime( __time__ - __time__ % 300) as time group by __time__ - __time__
 % 300 order by time limit 1440
```

```
```

- 统计top 10的 waiting：

```
*| select  max(waiting) as max_waiting, address, from_unixtime(max(
__time__)) as time group by address order by max_waiting desc limit
10
```

- 目前Nginx总数以及invalid数量：

```
* | select  count(distinct(address)) as total
```

```
not _result_ : success | select  count(distinct(address))
```

- 最近 top 10 失败的请求：

```
not _result_ : success | select _address_ as address, from_unixtime(
__time__) as time  order by __time__ desc limit 10
```

- 每5分钟统计统计请求处理总数：

```
*| select  avg(handled) * count(distinct(address)) as total_hand
led, avg(requests) * count(distinct(address)) as total_requests,
from_unixtime( __time__ - __time__ % 300) as time group by __time__
 - __time__ % 300 order by time limit 1440
```

- 每5分钟统计平均请求延迟：

```
*| select  avg(_response_time_ms_) as avg_delay,  from_unixtime(
__time__ - __time__ % 300) as time group by __time__ - __time__ %
300 order by time limit 1440
```

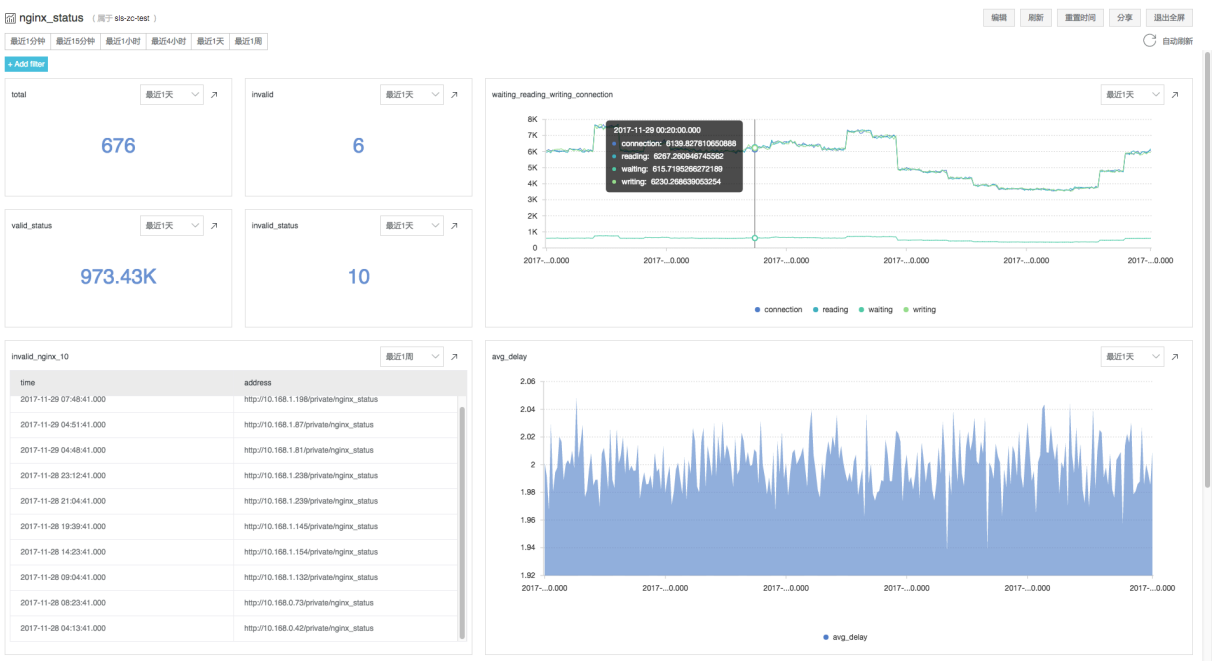
- 请求有效数/无效数：

```
not _http_response_code_ : 200  | select  count(1)
```

```
_http_response_code_ : 200  | select  count(1)
```

仪表盘

日志服务默认对于Nginx监控数据提供了仪表盘，您可以在Nginx status的仪表盘，仪表盘搭建参见#unique_10。



设置报警

1. 将以下查询另存为快速查询，名称为`invalid_nginx_status`：`not _http_response_code_ : 200 | select count(1) as invalid_count`。
2. 根据该快速查询#unique_49，样例如下：

配置	取值
告警名称	invalid_nginx_alarm
添加到仪表盘	nginx_status
图表名称	total
查询语句	* | select count(distinct(__source__)) as total
查询区间	15分钟
执行间隔	15分钟
触发条件	total>100
触发通知阈值	1
通知类型	通知中心
通知内容	nginx status 获取异常，请前往日志服务查看具体异常信息，project：xxxxxxxx，logstore：nginx_status

3.5 分析-行车轨迹日志

出租车公司记录了每一次载客交易发生的信息细节，包括上下客时间、经纬度、路程距离、支付方式、支付金额、缴税额等信息。详细的数据，为出租车公司的运营提供了极大的帮助，例如，了解哪些时间段比较热门，对应增加运行车次；哪些地区需求比较广泛，调度更多车辆前往。这些数据，使得乘客的需求得到了及时的响应，而驾驶员的收入也得到了提高，进而整个社会的效率得到了提高。

出租车公司把载客日志保存在阿里云日志服务上，利用日志服务可靠的存储，以及快速统计计算，挖掘日志中有用信息。本文将展示出租车公司如何使用阿里云日志服务来挖掘数据中的信息。

数据样例：

```
RatecodeID:  1VendorID:  2__source__:  11.164.232.105    __topic__:
  dropoff_latitude:  40.743995666503906    dropoff_longitude:  -73.
983505249023437extra:  0    fare_amount:  9    improvement_surcharg
e:  0.3    mta_tax:  0.5    passenger_count:  2    payment_type:
1    pickup_latitude:  40.761466979980469    pickup_longitude:  -73
.96246337890625    store_and_fwd_flag:  N    tip_amount:  1.96
tolls_amount:  0    total_amount:  11.76    tpep_dropoff_datetime:
2016-02-14 11:03:13    tpep_dropoff_time:  1455418993    tpep_picku
p_datetime:  2016-02-14 10:53:57    tpep_pickup_time:  1455418437
trip_distance:  2.02
```

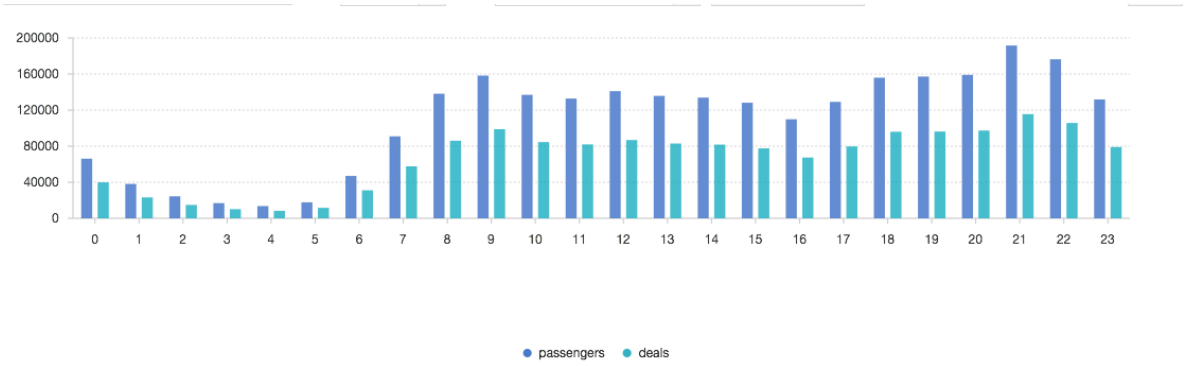
	时间 ↓	RatecodeID	VendorID	dropoff_latitude	dropoff_longitude	pickup_latitude	pickup_longitude	total_amount	tpep_dropoff_datetime	tpep_pickup_datetime	trip_distance
1	08-31 20:05:53	1	2	40.758163452148438	-73.991294860839844	40.704853057861328	-74.015922546386719	24.3	2016-02-14 14:49:31	2016-02-14 14:17:32	4.85
2	08-31 20:05:53	1	1	40.708518981933594	-74.017219543457031	40.718776702880859	-74.000679016113281	11.15	2016-02-14 14:27:32	2016-02-14 14:17:32	1.50
3	08-31 20:05:53	3	1	40.690460205078125	-74.177558898925781	40.741939544677734	-74.003875732421875	105.95	2016-02-14 14:47:29	2016-02-14 14:17:32	19.60
4	08-31 20:05:53	1	1	40.7266845703125	-73.990493774414063	40.7139892578125	-74.009140014648437	10.8	2016-02-14 14:29:52	2016-02-14 14:17:32	2.00
5	08-31 20:05:53	1	1	40.719020843505859	-73.999252319335938	40.711505889892578	-74.009956359863281	11.76	2016-02-14 14:29:43	2016-02-14 14:17:32	1.00
6	08-31 20:05:53	1	2	40.744297027587891	-73.985466003417969	40.764141082763672	-73.973602294921875	13.8	2016-02-14 14:37:21	2016-02-14 14:17:31	2.11
7	08-31 20:05:53	1	2	40.763916015625	-73.958244323730469	40.770534515380859	-73.948394775390625	7.56	2016-02-14 14:22:37	2016-02-14 14:17:31	.96
8	08-31 20:05:53	1	2	40.750503540039063	-73.989883422851562	40.763473510742188	-73.996414184570313	9.8	2016-02-14 14:29:07	2016-02-14 14:17:31	1.28
9	08-31 20:05:53	1	1	40.748451232910156	-73.988792419433594	40.717227935791016	-73.995231628417969	17.8	2016-02-14 14:43:07	2016-02-14 14:17:31	2.70
10	08-31 20:05:53	1	1	40.720909118652344	-74.000808715820313	40.742031097412109	-73.983070373535156	10.8	2016-02-14 14:30:50	2016-02-14 14:17:31	1.80

常见的统计

要对数据进行查询和分析，请先#unique_43。

1. 分时段乘车人次，查看哪些时段比较热门

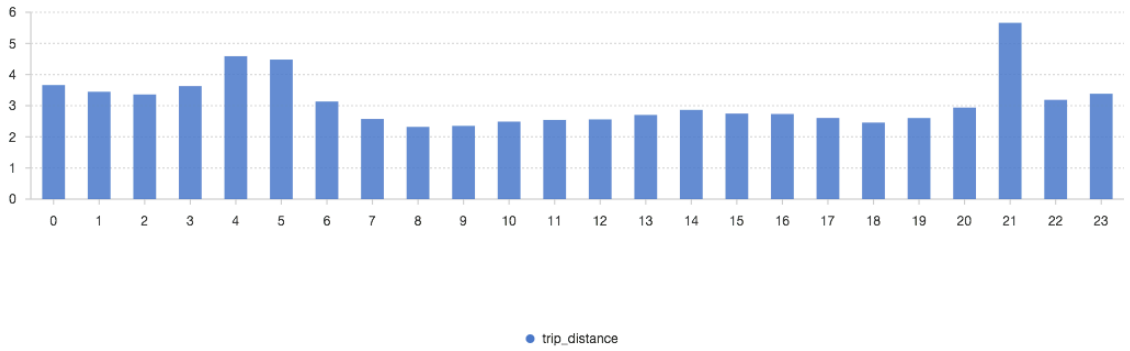
```
 *| select count(1) as deals, sum(passenger_count) as passengers,

 (tpep_pickup_time %(24*3600)/3600+8)%24 as time
 group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time
limit 24
```



从结果中可以看出，上午上班时间，以及晚上下班后，是乘车需求最旺盛的时候，出租车公司可以相应的调度更多的车辆。

2. **分时段平均乘车里程**

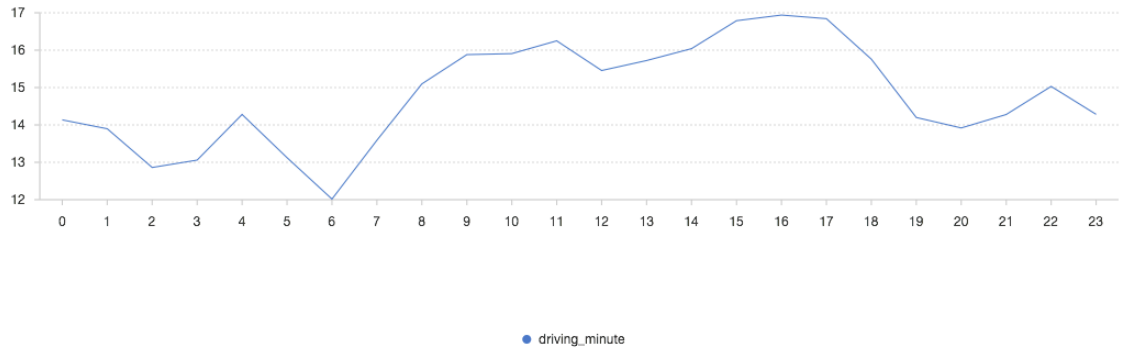
```
*| select  avg(trip_distance)  as trip_distance,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit
 24
```



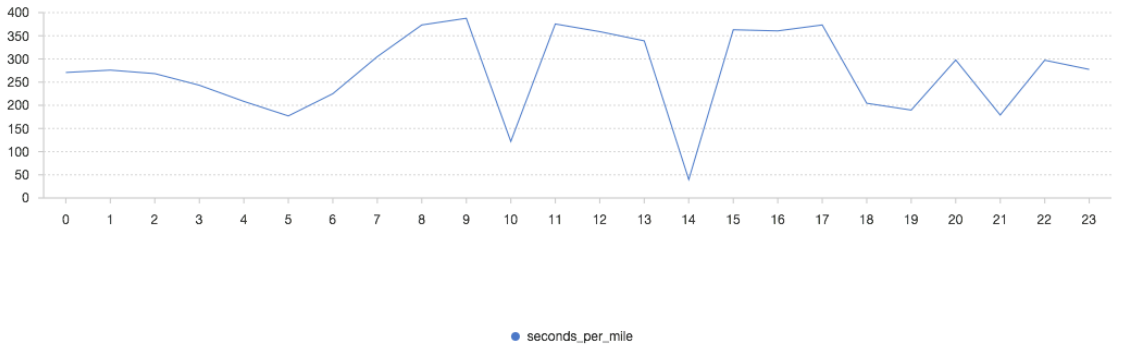
某些时刻，对乘车里程的需求也挺旺盛，出租车公司在对应的时候也需要准备更多的车辆。

3. **分时段平均乘车分钟数,单位里程需要的秒数，看看哪些时段比较堵**

```
*| select  avg(tpep_dropoff_time-tpep_pickup_time)/60  as driving_mi
nutes,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit
 24
```



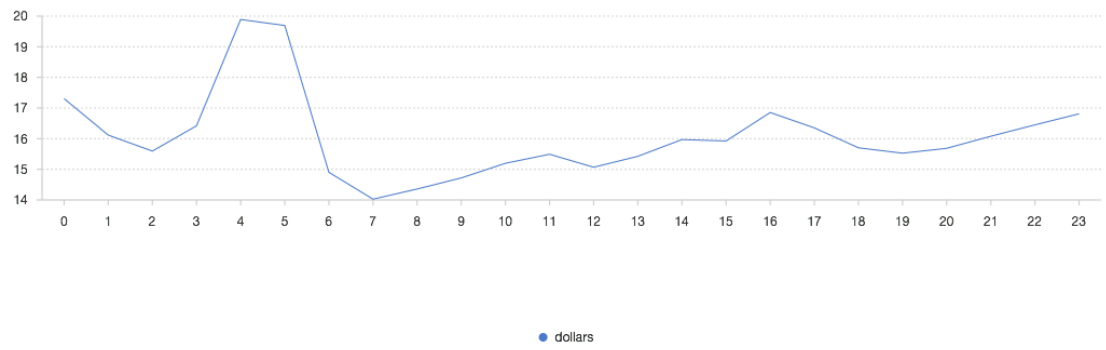

```
*| select  sum(tpep_dropoff_time-tpep_pickup_time)/sum(trip_distance
)  as driving_minutes,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit
 24
```



一些时刻特别堵，需要准备更多车辆来应对需求。

4. 分时段平均乘车费用，看看哪些时间赚的多

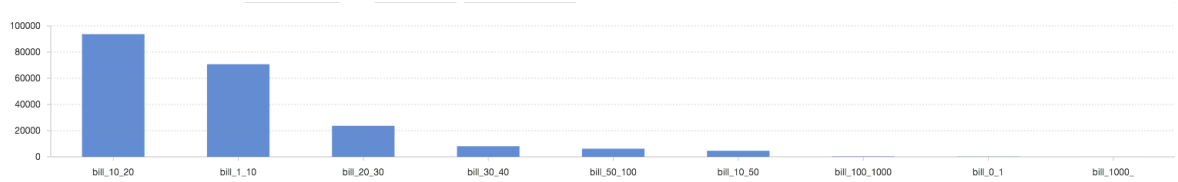
```
*| select  avg(total_amount)  as dollars,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit
 24
```



凌晨4点钟的客单价比较高，有经济压力的驾驶员可以选择在这个时候提供服务。

5. 看看账单范围分布情况

```
*| select case when total_amount < 1 then 'bill_0_1'
when total_amount < 10 then 'bill_1_10'
when total_amount < 20 then 'bill_10_20'
when total_amount < 30 then 'bill_20_30'
when total_amount < 40 then 'bill_30_40'
when total_amount < 50 then 'bill_10_50'
when total_amount < 100 then 'bill_50_100'
when total_amount < 1000 then 'bill_100_1000'
else 'bill_1000_'  end
as bill_level , count(1) as count group by
case when total_amount < 1 then 'bill_0_1'
when total_amount < 10 then 'bill_1_10'
when total_amount < 20 then 'bill_10_20'
when total_amount < 30 then 'bill_20_30'
when total_amount < 40 then 'bill_30_40'
when total_amount < 50 then 'bill_10_50'
when total_amount < 100 then 'bill_50_100'
when total_amount < 1000 then 'bill_100_1000'
else 'bill_1000_'  end
order by count desc
```



从成交金额的成交区间，可以看出大部分的成交金额在1到20美元之间。

3.6 分析-分析SLB七层访问日志

阿里云SLB是对多台云服务器进行流量分发的负载均衡服务，可以通过流量分发扩展应用系统对外的服务能力，通过消除单点故障提升应用系统的可用性。负载均衡对于大部分云上架构来说都是基础设施组件，因此，对SLB持续的监控、探测、诊断和报告是一个强需求。用户一般可以通过云厂商内置的监控报表来了解SLB实例运行状况。

SLB访问日志功能当前支持基于HTTP/HTTPS的七层负载均衡，访问日志内容丰富，提供近30个字段，例如：收到请求的时间、客户端的IP地址、处理Latency、请求URI、后端RealServer（阿里云ECS）地址、返回状态码等。完整字段及功能说明请参考负载均衡7层访问日志功能。

本文档基于阿里云日志服务的可视化和日志实时查询（OLTP+OLAP）能力，为您介绍SLB七层访问日志的实时采集、查询与分析最佳方案，并举例说明SLB实例的一些典型报表统计、日志查询分析的方法。该方案结合了可视化报表和查询分析引擎，可以实时、交互分析SLB实例状况。

目前SLB7层访问日志已在所有区域开放，欢迎使用。

前提条件

1. 已开通日志服务与SLB七层负载均衡。
2. 已成功采集到SLB七层负载均衡日志。详细步骤请参考采集步骤。

可视化分析

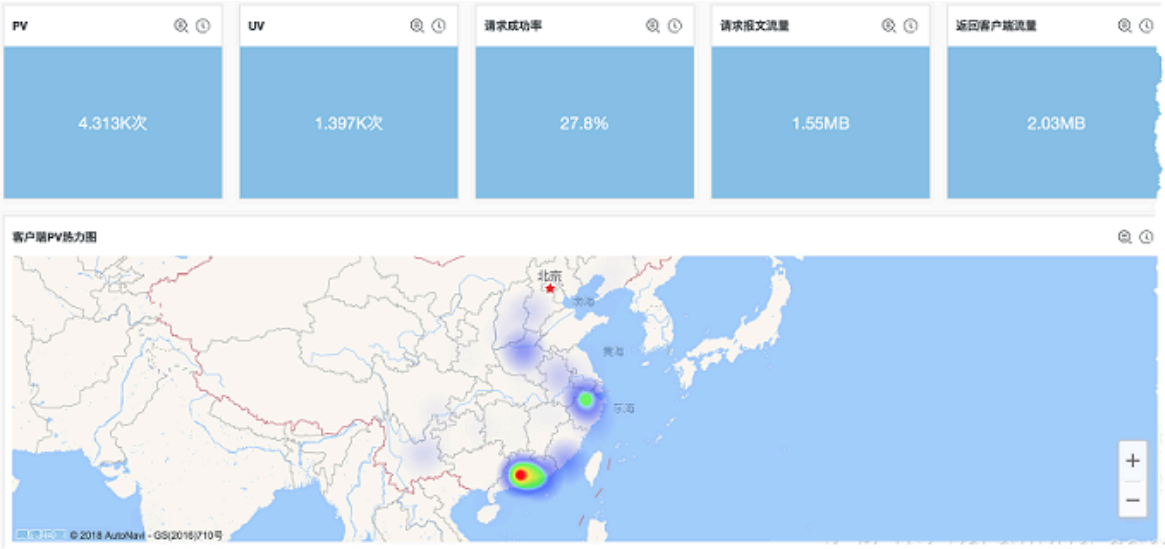
业务概览

负载均衡支持RealServer的水平扩展和故障冗余恢复，为应用提供大规模、高可靠的并发web访问服务支撑。

> 说明:
> 典型的概览指标包括：
>
> - PV：client（请求源IP）发起的HTTP(S)请求次数。
> - UV：对于相同client IP只计算一次，合计的总体请求次数。
> - 请求成功率：状态码为2XX的请求次数占总体PV的比例。
> - 请求报文流量：客户端请求报文长度（request_length字段）的总和。
> - 返回客户端流量：SLB返回给客户端的HTTP body的字节数（body_bytes_sent字段）总和。
> - 请求的热点分布：计算client IP的地理位置，按照每个地理位置来统计每个区域的PV情况。

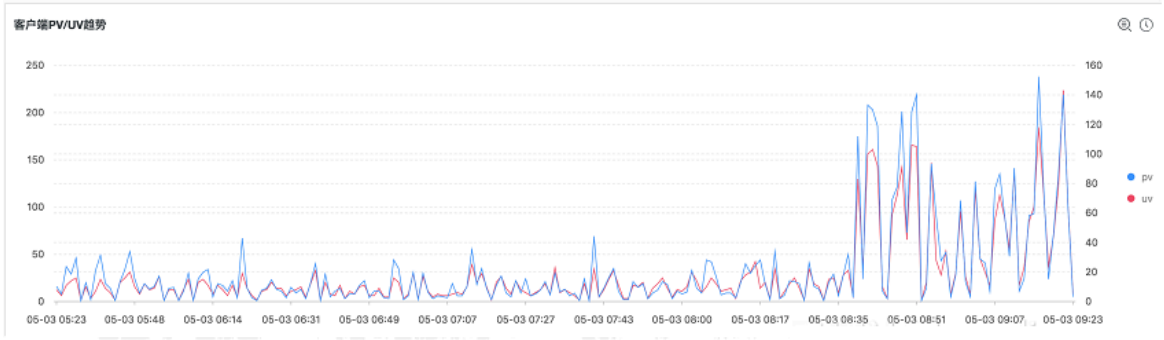
- 查看用户请求的来源地区。

  如下图所示，通过地图可以看到用户请求主要来自珠三角和长三角区域。



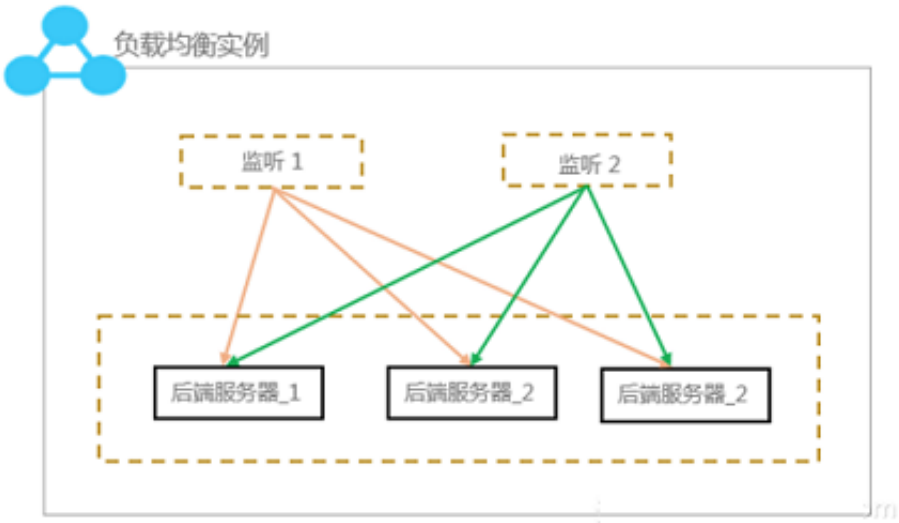
- 在日志服务的Dashboard中，通过添加过滤条件可以在当前图表中筛选符合条件的数据指标来展示，例如：client IP维度、SLB实例ID维度。

  例如，查询一个指定SLB实例ID的PV、UV随时间的变化趋势。



请求调度分析

从客户端过来的流量会先被SLB处理，并分发到多台RealServer的一台上做实际的业务逻辑处理。SLB可以检测不健康的机器并重新分配流量到其它正常服务的RealServer上，等异常机器恢复后再将流量重新加上去，这个过程是自动完成的。



对SLB实例添加一个监听，监听可以设置三种调度方法：轮询、加权轮询（WRR）和加权最小连接数（WLC）。

- 轮询：按照访问次数依次将外部请求依序分发到后端ECS上。
- 加权轮询：您可以对每台后端服务器设置权重值，权重值越高的服务器，被轮询到的次数（概率）也越高。
- 加权最小连接数：除了根据每台后端服务器设定的权重值来进行轮询，同时还考虑后端服务器的实际负载（即连接数）。当权重值相同时，当前连接数越小的后端服务器被轮询到的次数（概率）也越高。

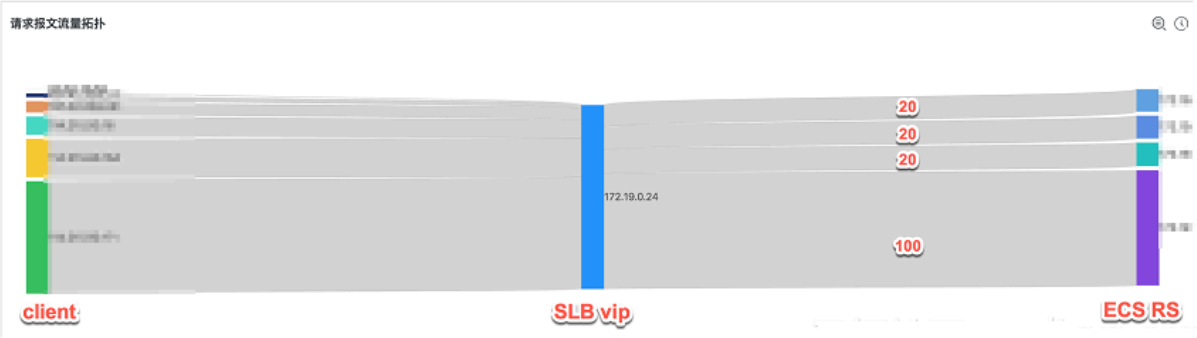
例如，`172.19.39.**` 机器同时兼有跳板机职能，其性能是其它三台机器的4倍，这里为它设置权重100，其余设置权重为20。

基于实例的访问日志，通过如下一条查询语句可以完成和两个维度的流量聚合：

```
* | select COALESCE(client_ip, vip_addr, upstream_addr) as source,
COALESCE(upstream_addr, vip_addr, client_ip) as dest, sum(request_le
```

```
ngth) as inflow group by grouping sets( (client_ip, vip_addr), (
vip_addr, upstream_addr))
```

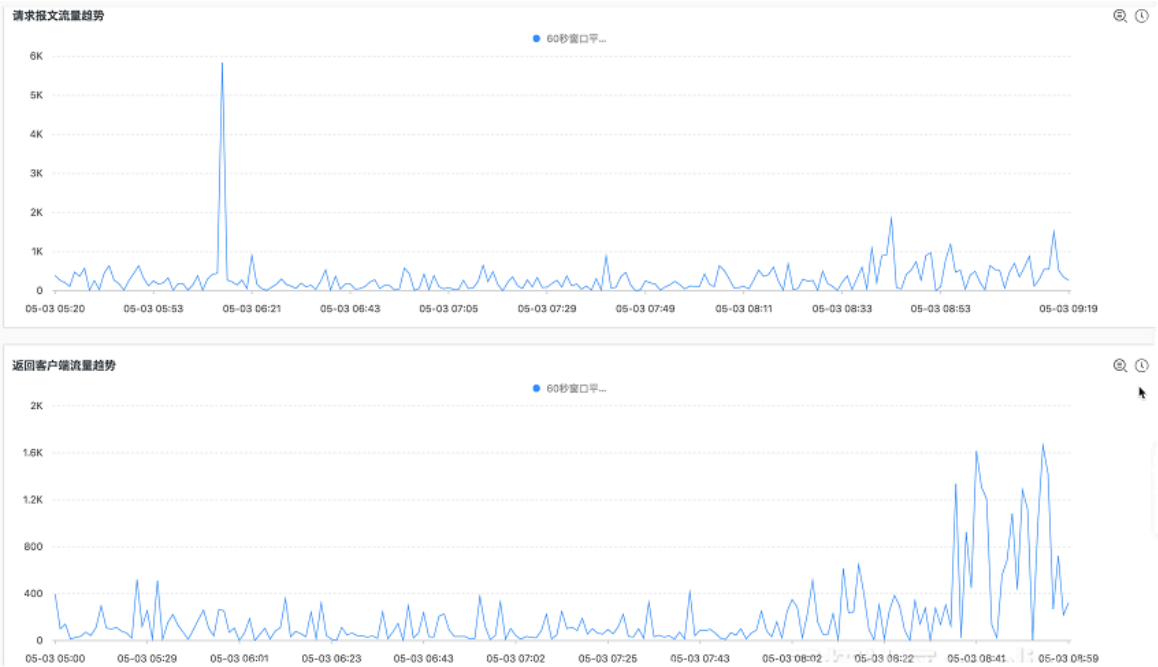
结合桑基图对SQL查询结果做vip维度的聚合可视化，最终得到请求报文流量拓扑图。多个client IP向SLB vip（`172.19.0.**`）发起请求，请求报文流量基本遵循20:20:20:100比例转发到后端的RealServer进行处理。桑基图清晰地表述了每台RealServer的负载情况。



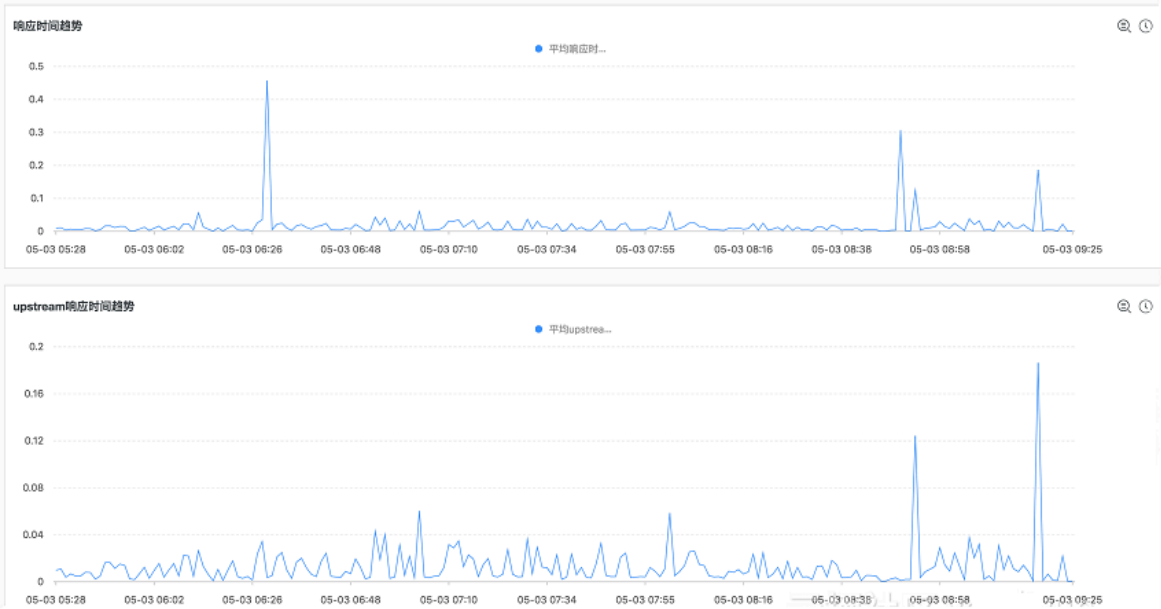
流量与Latency分析

按照1分钟时间维度对流量与latency指标做聚合计算：

- request_length、body_bytes_sent统计



- request_time、upstream_response_time统计



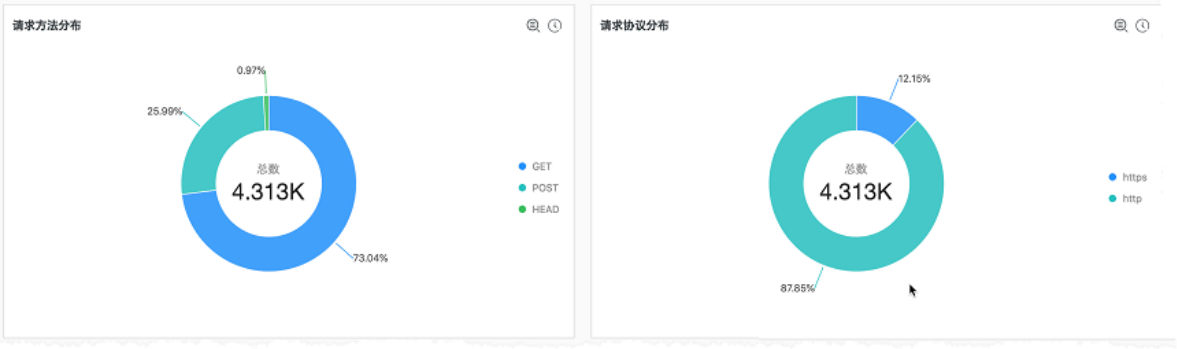
- 高延迟RealServer统计

用户请求概览

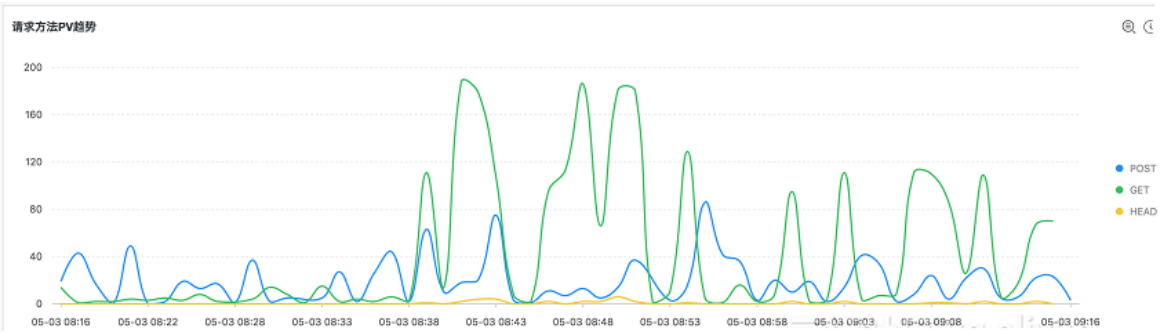
根据请求的方法、协议、状态码等维度分析访问日志中HTTP(S)请求。

指定时间段、状态码快速定位RealServer的需求，通过日志服务的查询分析功能可以迅速得到结果：

1. 在一个时间段内，请求方法维度上可以做PV分布统计。

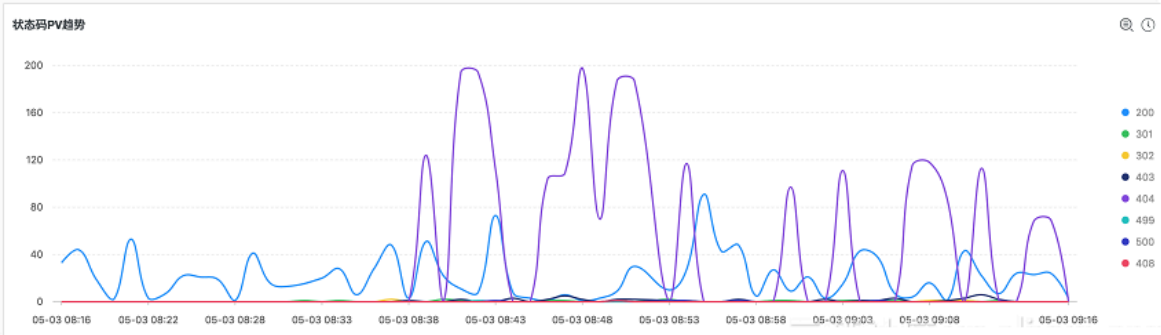


2. 加上时间属性，同时在时间、请求方法两个维度上可以统计出各方法的PV趋势。



3. 请求响应状态码分布可以展示服务的基本状况，如果大量的500状态码则意味着后端RealServer的应用程序在发生内部错误。

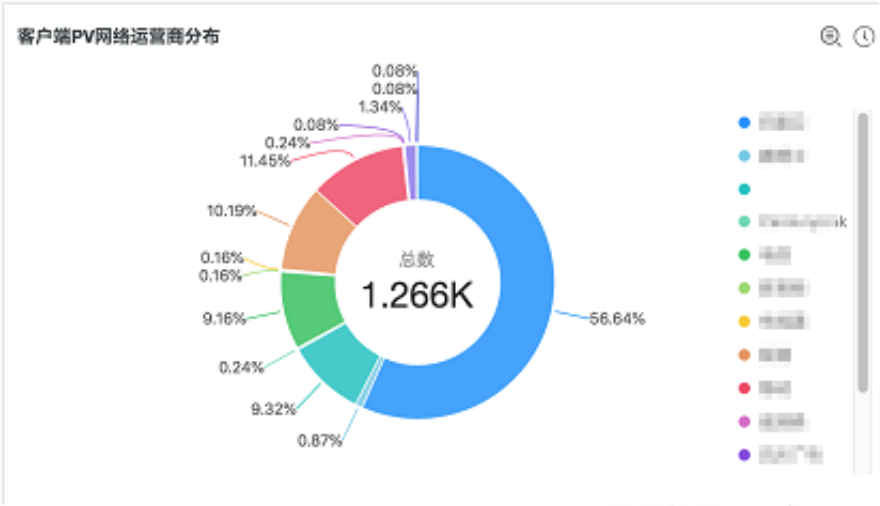
4. 围绕着每一个状态码，可以查看其随时间变化趋势。



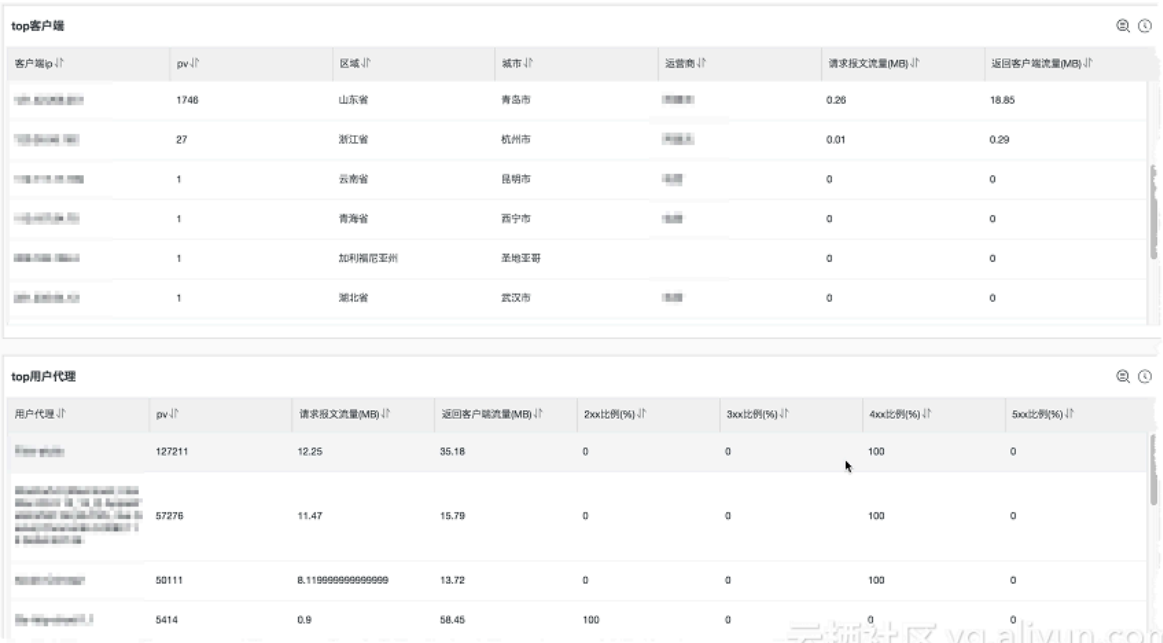
请求源分析

对client的IP做计算可以得到每条请求的发起地理位置（国家、省份、城市）、电信运营商信息。

1. 对用户请求IP的运营商做PV分布图。



2. 按照请求PV降序，对client IP做统计可以展示大用户请求的具体来源。

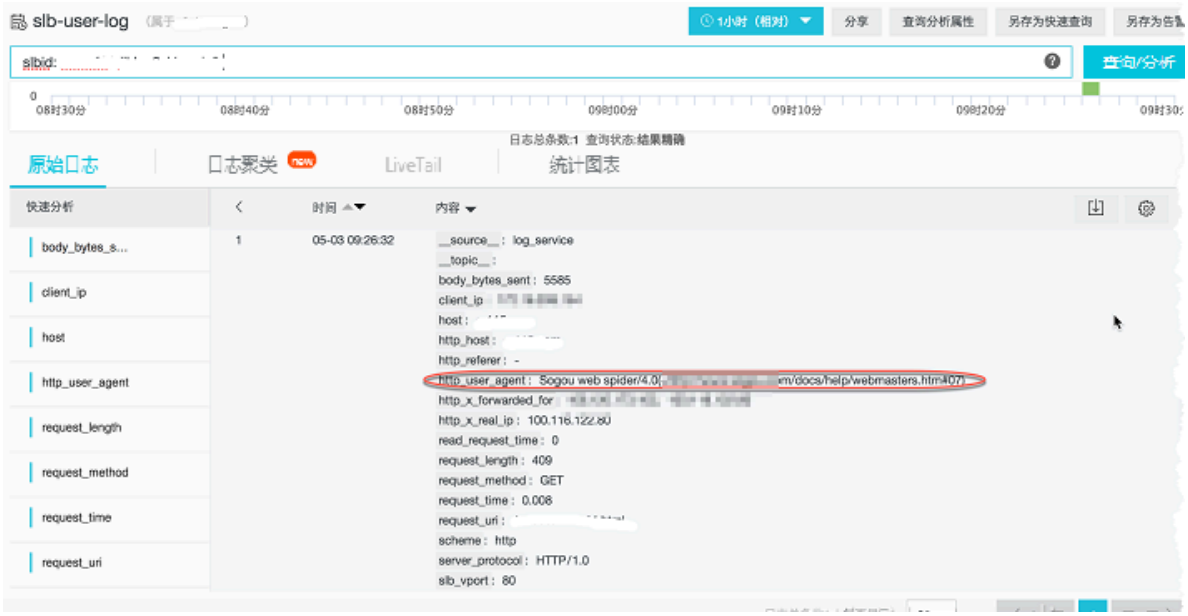


3. 查看http_user_agent。

用户代理（http_user_agent）也是常常需要关注的对象，可以据此区分出谁在访问我们的网站或服务。比如搜索引擎会使用爬虫机器人扫描或下载网站资源，一般情况下的低频爬虫访问可以让搜索引擎及时更新网站内容、有助于网站的推广和SEO。但如果高PV的请求都来自于爬

虫，则可能对服务的性能和机器资源造成浪费，需要实时监控高PV请求状况并采取手段来控制影响。

访问日志中根据查询SLB ID或应用host、http_user_agent关键词，可以很快检索出相关记录。下图是一条搜狗爬虫程序的GET请求日志，请求很稀疏，对于应用无影响。

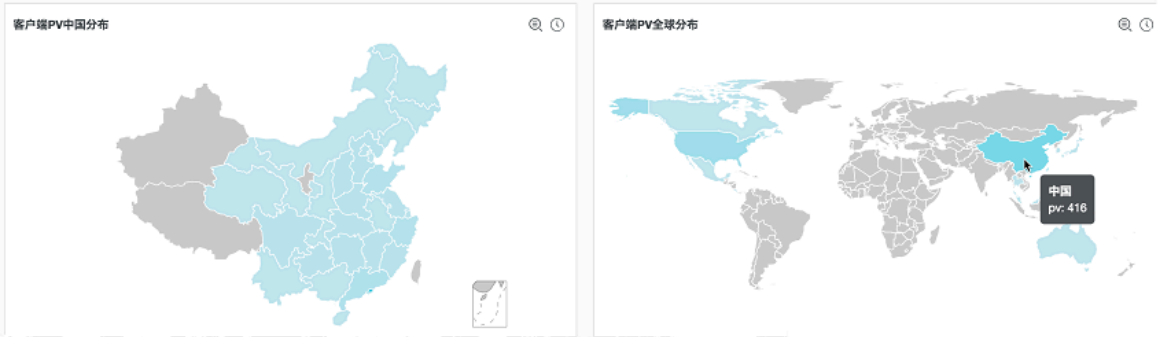


运营概览

SLB访问日志在运营同学手里同样发挥着重要作用，可以基于日志分析出来流量模式，进而辅助业务决策。

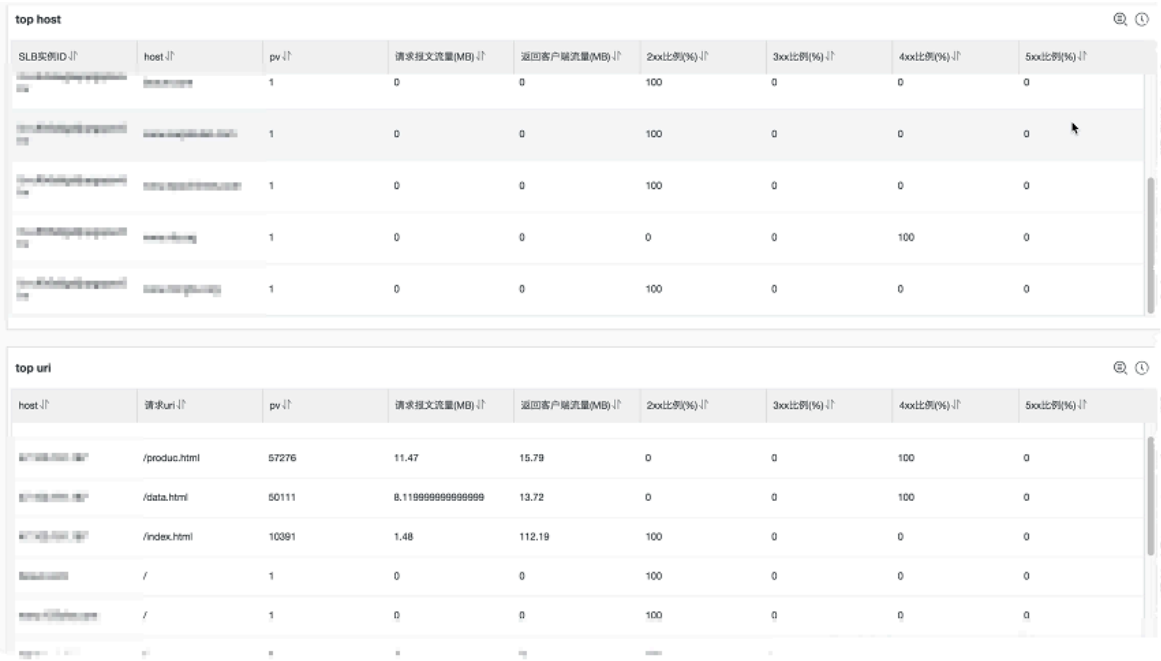
1. 查看PV的热点分布。

   在地理维度上，通过PV的热点分布，可以清晰了解到我们服务的重点客户在哪里，PV低的区域可能需要再推广加强。



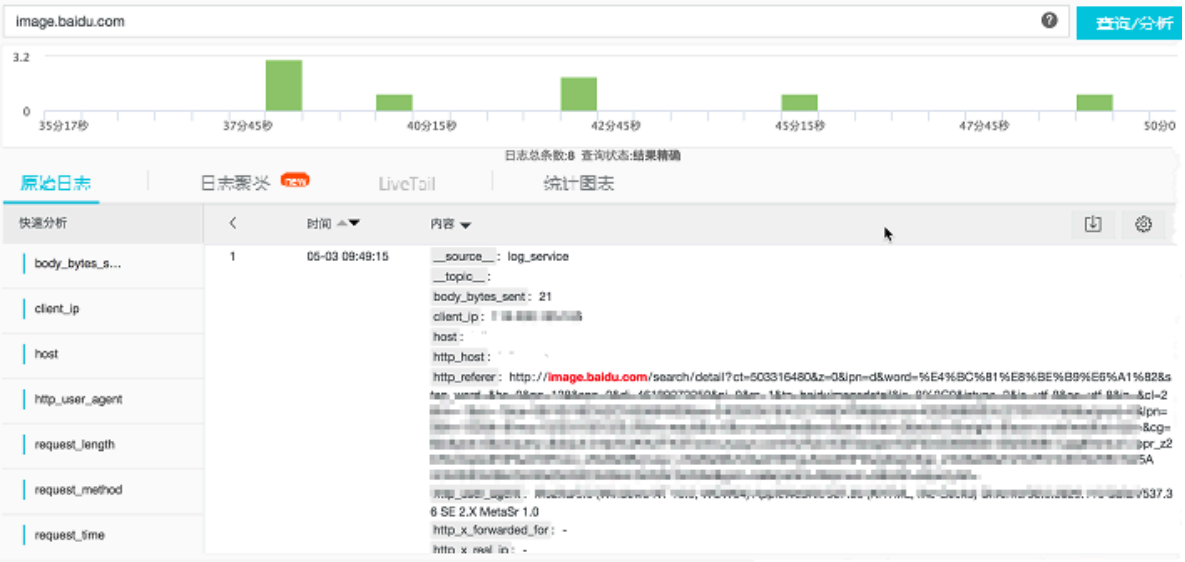
2. 查看host/URI。

对于一个网站而言，通过分析访客的行为可以为网站内容建设提供有力的参考。哪些内容好，哪些内容不好？这个问题可以用头部、尾部PV的host/URI来回答。



3. 查看request_uri。

对于热门资源（访问日志的request_uri字段），可以关注一下日志详情的http_referer字段，查看网站的请求来源。好的导流入口需要持续加强，对于盗链行为则需要想办法克制。下图是在日志查询页面中找到一条来自百度图片的跳转请求。



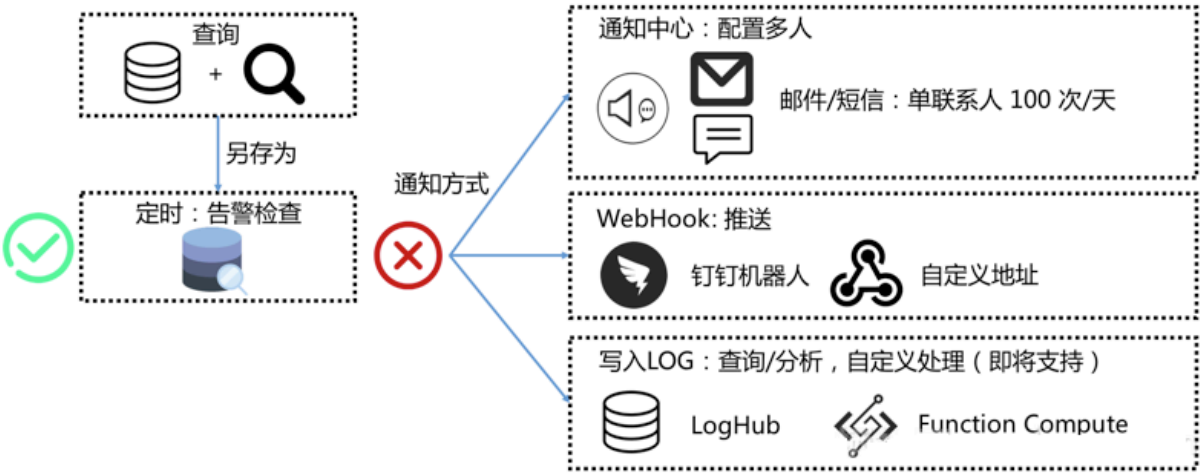
3.7 分析-Nginx访问日志

目前，日志服务支持保存查询语句为快速查询，对查询设置触发周期（间隔），并对执行结果设定判断条件并且告警。您还可以设置告警动作，即定期运行的快速查询结果一旦触发告警条件，将以何种方式通知您。

目前支持通知方式有以下3种：

- 通知中心：在阿里云通知中心可以设置多个联系人，通知会通过邮件和短信方式发送。
- WebHook：包括钉钉机器人，及自定义WebHook等。
- （即将支持）写回日志服务（logstore）：可以通过流计算，函数服务进行事件订阅；也可以对告警生成视图和报表。

告警功能配置与操作请参考#unique_54。 除通过日志服务监控并告警外，您还可以通过云监控产品对日志服务各项指标进行监控，并在触发告警条件时为您发送提醒消息。



实践场景

本文以Nginx日志为例，为您示范如何使用日志服务对采集到的日志信息定时查询分析，并通过日志查询结果判断以下业务问题：

- 是否有错误。
- 是否有性能问题。
- 是否有流量急跌或暴涨。

准备工作（Nginx日志接入）

1. 采集日志数据。

   a. 在概览页面单击接入数据，并选择NGINX-文本日志。

   b. 选择日志空间。

      如果您是通过日志库下的数据接入后的加号进入采集配置流程，系统会直接跳过该步骤。

   c. 创建机器组。

      在创建机器组之前，您需要首先确认已经安装了Logtail。

      - 集团内部机器：默认自动安装，如果没有安装，请根据界面提示进行咨询。
      - ECS机器： 勾选实例后单击安装进行一键式安装。Windows系统不支持一键式安装，请参考#unique_24手动安装。
      - 自建机器：请根据界面提示进行安装。或者参考#unique_25或#unique_26文档进行安装。

      安装完Logtail后单击确认安装完毕创建机器组。如果您之前已经创建好机器组，请直接单击使用现有机器组。

   d. 机器组配置。

      选择一个机器组，将该机器组从源机器组移动到应用机器组。

   e. 填写配置名称，日志路径，Nginx日志配置，NGINX键名称，并酌情配置高级选项。

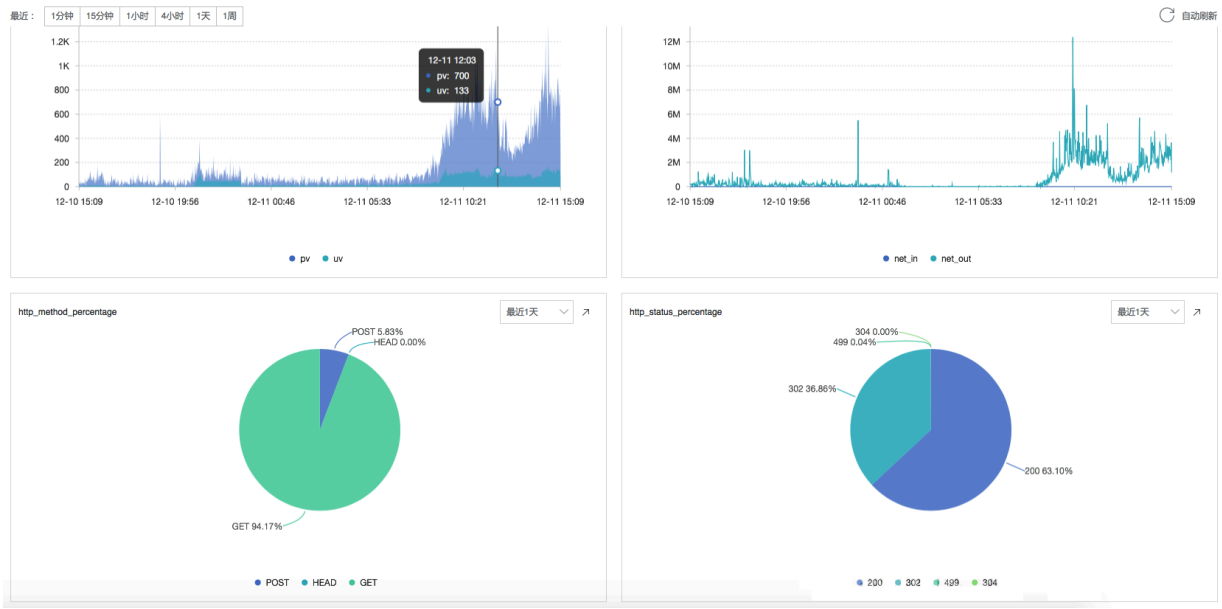
   f. 单击下一步，进入配置索引步骤。

2. 查询分析设置。

   详细内容请参考#unique_43与可视化或最佳实践网站日志分析案例。

3. 对关键指标设置视图和告警。

Sample视图：



操作步骤

1. 判断是否有错误

   错误一般有这样几类：404（请求无法找到地址）/502/500（服务端错误），我们一般只需关心500（服务端错误）。

   判断是否有500错误，您可以使用以下query统计单位时间内错误数c，并将报警规则设置为c > 0 则发送告警。

   ```
   status:500 | select count(1) as c
   ```

   这种方式比较简单，但往往过于敏感，对于一些业务压力较大的服务而言有零星几个500是正常的。为了应对这种情况，您可以在告警条件中设置触发次数为2次，即只有连续2次检查都符合条件后再发告警。

2. 判断是否有性能问题

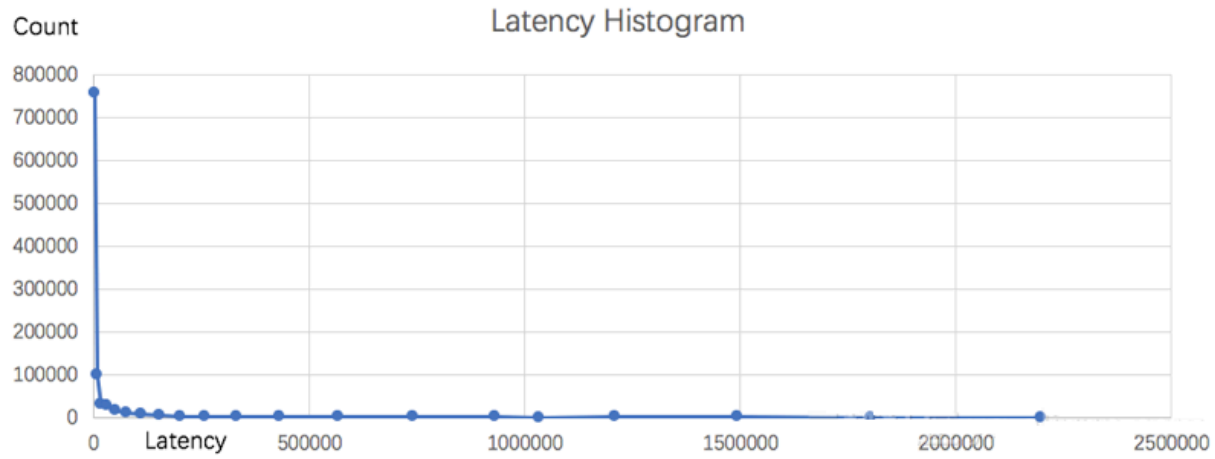
   服务器运行过程中虽然没有错误，但有可能会出现延迟（Latency）增大情况，您可以针对延迟进行告警。

例如您可以通过以下方式计算某个接口（“/adduser"）所有写请求（”Post “）延时。告警规则设置为 l > 300000 即当平均值超过300ms后告警。

```
Method:Post and URL:"/adduser" | select avg(Latency) as l
```

利用平均值来报警简单而直接，但这种方法往往会使得一些个体请求延时被平均掉，无法反馈问题。例如，对该时间段的Latency计算一个数学上的分布，即划分20个区间，计算每个区间内的数目，从分布图上可以看到大部分请求延时非常低（<20ms），但最高的延时有2.5s。

```
Method:Post and URL:"/adduser" | select numeric_histogram(20, Latency)
```



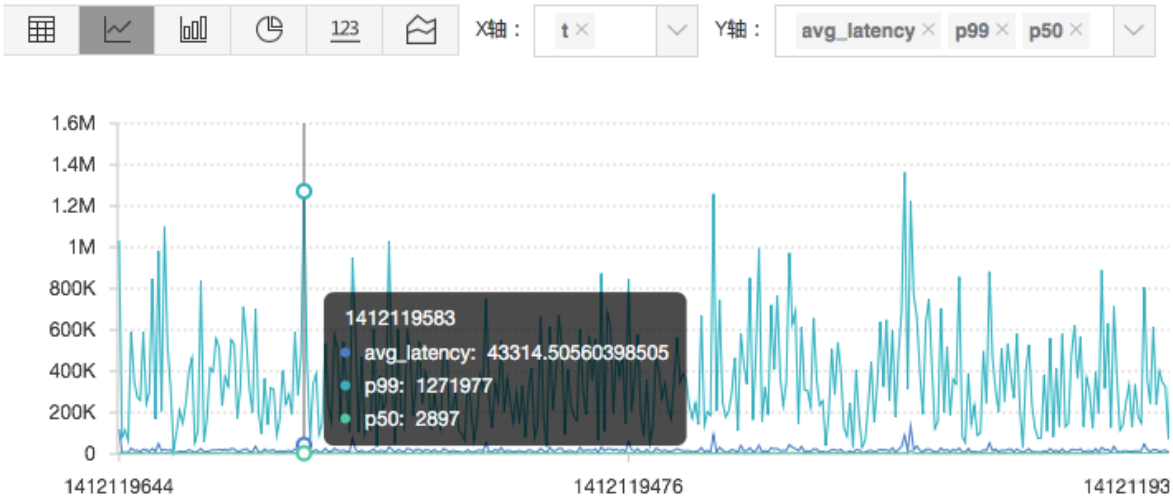
为应对这种情况，您可以用数学统计中的百分数（99%最大延时）来作为报警条件，这样既可以排除偶发的延时高引起误报，也能对整体的演示更有代表性。以下的语句计算了99%分位的延时大小

approx_percentile(Latency, 0.99)，同样您也可以修改第二个参数进行其他分位的划分，例如中位数的请求延时 approx_percentile(Latency, 0.5)。

```
Method:Post and URL:"/adduser" | select approx_percentile(Latency, 0.
99) as p99
```

在监控的场景中，您也可以在一个图上绘出平均延时，50%分位延时，以及90%分位延时。以下是按一天的窗口（1440分钟）统计各分钟内延时的图：

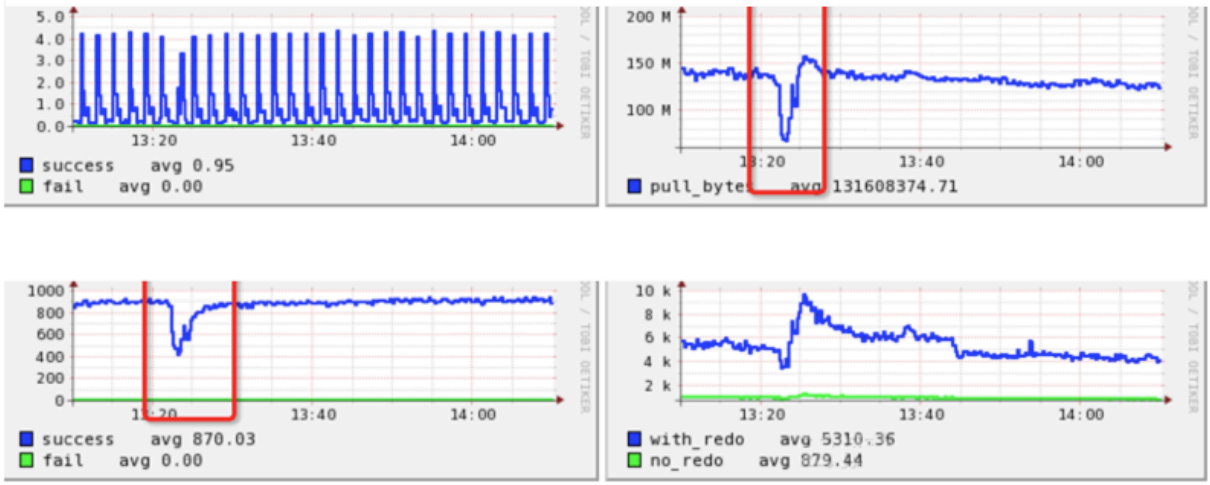
```
* | select avg(Latency) as l, approx_percentile(Latency, 0.5) as p50,
approx_percentile(Latency, 0.99) as p99, date_trunc('minute', time) as
 t group by t order by t desc limit 1440
```



3. 判断是否有流量急跌或暴涨

服务器端自然流量一般符合概率上的分布，会有一个缓慢上涨或下降过程。流量急跌或暴涨表示短时间内流量变化非常大，一般都是不正常的现象，需要留意。

如以下监控图所示，在2分钟时间内流量大小下跌30%以上，在2分钟内后又迅速恢复。



急跌和暴涨一般会有如下参考系：

- 上一个时间窗口：环比上一个时间段。
- 上一天该时间段的窗口：环比昨天。
- 上一周该时间段的窗口：环比上周。

本文以第一种情况为例，计算流量infow数据的变动率，也可以换成QPS等流量。

3.1 首先定义一个计算窗口

定一个1分钟的窗口，统计该分钟内的流量大小，以下是一个5分钟区间统计：

```
* | select sum(inflow)/(max(__time__)-min(__time__)) as inflow ,
__time__-__time__%60  as window_time from log group by window_time
order by window_time limit 15
```

从结果分布上看，每个窗口内的平均流量 `sum(inflow)/(max(__time__)-min(__time__))` 应该是均匀的。

window_time	inflow
1513045740	315574947
1513045800	333233937
1513045860	335821584
1513045920	330556452
1513045980	316785257

3.2 计算窗口内的差异值（最大值变化率）

这里我们会用到子查询，我们写一个查询，从上述结果中计算最大值或最小值与平均值的变化率（这里的max_ratio），例如如下计算结果max_ratio为 1.02。我们可以定义一个告警规则，如果max_ratio > 1.5（变化率超过50%）就告警。

```
 * | select max(inflow)/avg(inflow) as max_ratio from (select sum(
inflow)/(max(__time__)-min(__time__)) as inflow , __time__-__time__%
```

```
60  as window_time from log group by window_time order by window_time
limit 15)
```

window_time	inflow
1513045740	315574947
1513045800	333233937
1513045860	335821584
1513045920	330556452
1513045980	316785257

3.3 计算窗口内的差异值（最近值变化率）

在一些场景中我们更关注最新的数值是否有波动（是否已经恢复），那可以通过max_by方法获取最大windows_time中的流量来进行判断，这里计算的最近值为lastest_ratio=0.97。

```
 * | select max_by(inflow, window_time)/1.0/avg(inflow) as lastest_ra
tio from (select sum(inflow)/(max(__time__)-min(__time__)) as inflow
 , __time__-__time__%60  as window_time from log group by window_time
order by window_time limit 15)
```

说明:

这里的max_by函数计算结果为字符类型，我们需要强转成数字类型。 如果要计算变化相对率，可以用（1.0-max_by(inflow, window_time)/1.0/avg(inflow)) as lastest_ratio代替。

window_time	inflow
1513045740	315574947
1513045800	333233937
1513045860	335821584
1513045920	330556452
1513045980	316785257

3.4 计算窗口内的差异值（定义波动率，上一个值与下一个变化率）

波动率另外一种计算方法是数学上一阶导数，即当前窗口值与上个窗口值的变化值。

window_time	inflow
1513308660	8138522256
1513308720	8584340710
1513308780	9210706832
1513308840	9684619494

Diff

使用窗口函数(lag)进行计算，窗口函数中提取当前inflow与上一个周期inflow "lag(inflow, 1, inflow)over() " 进行差值，并除以当前值作为一个变化比率：

```
 * | select (inflow- lag(inflow, 1, inflow)over() )*1.0/inflow as diff
, from_unixtime(window_time) from (select sum(inflow)/(max(__time__)-
min(__time__)) as inflow , __time__-__time__%60  as window_time from
log group by window_time order by window_time limit 15)
```

在示例中，11点39分流量有一个较大的降低（窗口之间变化率为40%以上）：

如果要定义一个绝对变化率，可以使用abs函数（绝对值）对计算结果进行统一。

总结

日志服务查询分析能力是完整SQL92，支持各种数理统计与计算等，只要会用SQL都能进行快速分析，欢迎尝试！

3.8 查询-消息服务(MNS)日志

阿里云消息服务(MNS)支持将日志推送到日志服务，本文为您介绍日志成功推送后，如何通过日志查询特定信息。

消息服务支持队列消息操作日志以及主题消息操作日志，其中日志包含了消息生命周期的所有内容，时间、地点、操作和上下文等。您可以通过实时查询、实时计算和离线计算三种方法对日志进行分析。

实时查询

本文档仅介绍几种常用场景的查询，用户可以通过组合多个关键字来实现更加复杂的查询。

查看队列消息的消息轨迹

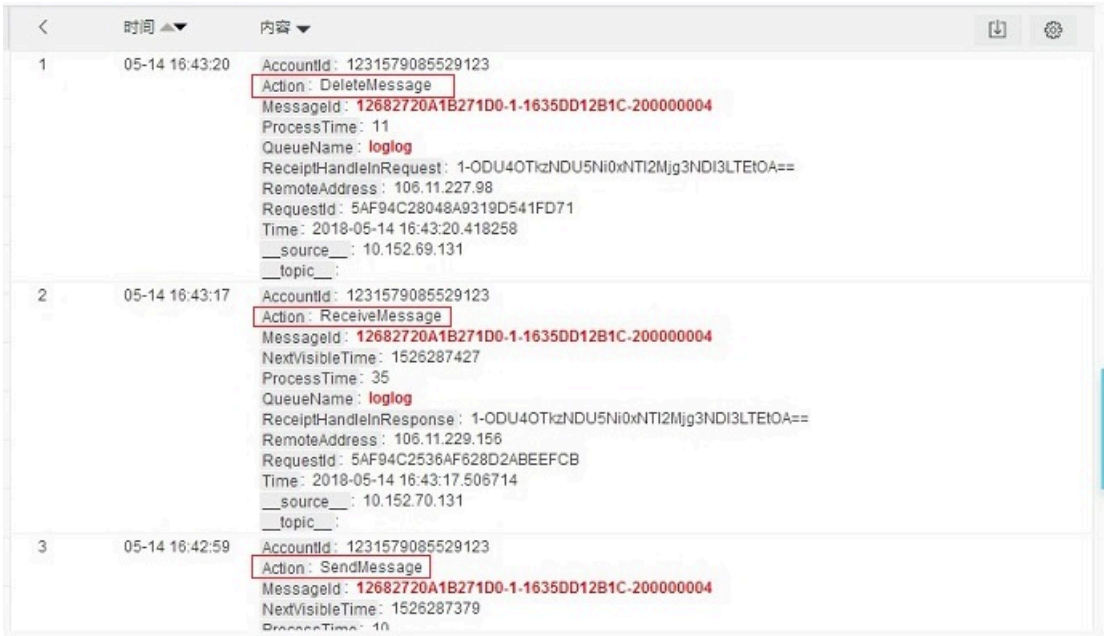
- 步骤

  1. 在搜索框中输入队列名称和MessageId。格式为`$queuename and $messageid`。
  2. 选择合适的时间范围后，单击查询/分析即可查看该消息的详细操作日志。

- 示例

  查看loglog队列中MessageId为`12682720A1B271D0-1-1635DD12B1C-200000004`的消息轨迹。

  - 查询语句：`loglog and 12682720A1B271D0-1-1635DD12B1C-200000004`
  - 查询结果：如下图所示，查询结果中展示了该消息从发送到删除的过程。



查看队列消息写入量

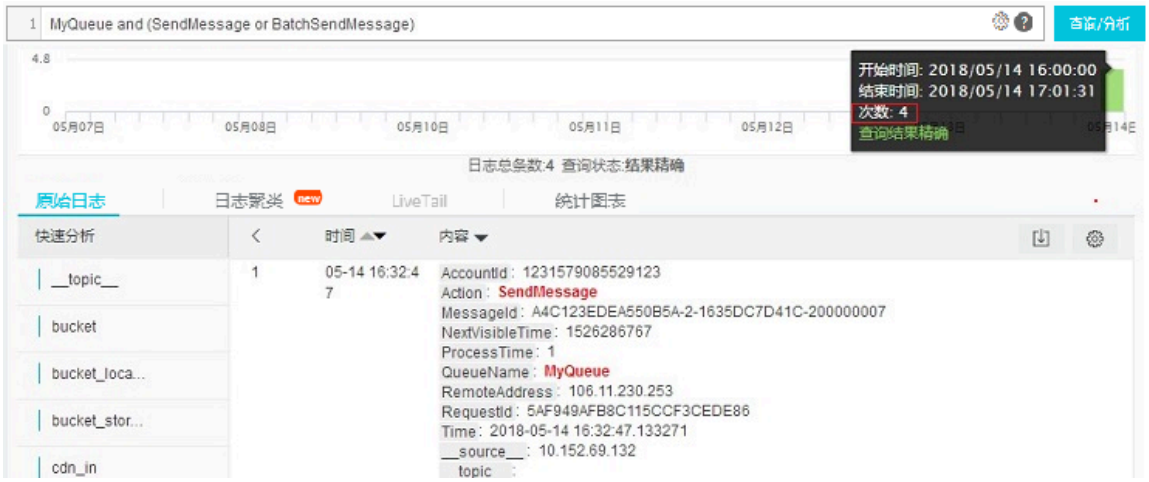
- 步骤

  1. 在搜索框中输入队列名称和写入操作。格式为`$queuename and (SendMessage or BatchSendMessage)`。
  2. 选择合适的时间范围后，单击查询/分析即可查看该队列的所有写入消息。鼠标移至绿色柱状图上，可以查看当前时间段内的具体消息数量。

- 示例

  查看MyQueue队列中的消息写入量。

  - 查询语句：`MyQueue and (SendMessage or BatchSendMessage)`
  - 查询结果：如下图所示，当前查询时段内，有4条写入操作。



查看队列消息消费量

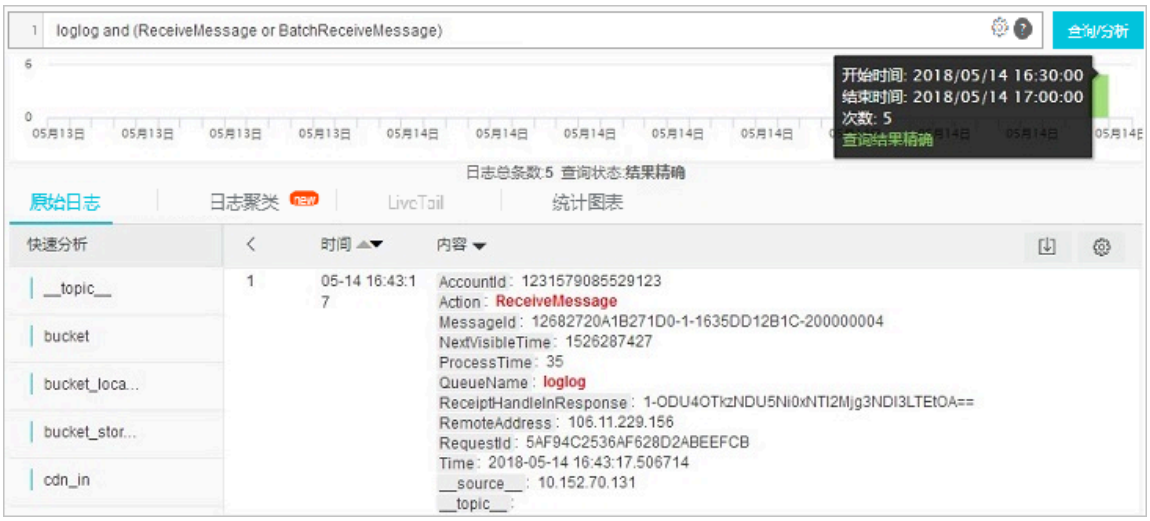
- 步骤

  1. 在搜索框中输入队列名称和消费操作。格式为`$queuename and (ReceiveMessage or BatchReceiveMessage)`。
  2. 选择合适的时间范围后，单击查询/分析即可查看该队列消息消费量。

- 示例

  查看loglog队列中的消息消费量。

  - 查询语句：`loglog and (ReceiveMessage or BatchReceiveMessage)`
  - 查询结果：如下图所示，当前查询时段内，有5条消费记录。



查看队列消息删除量

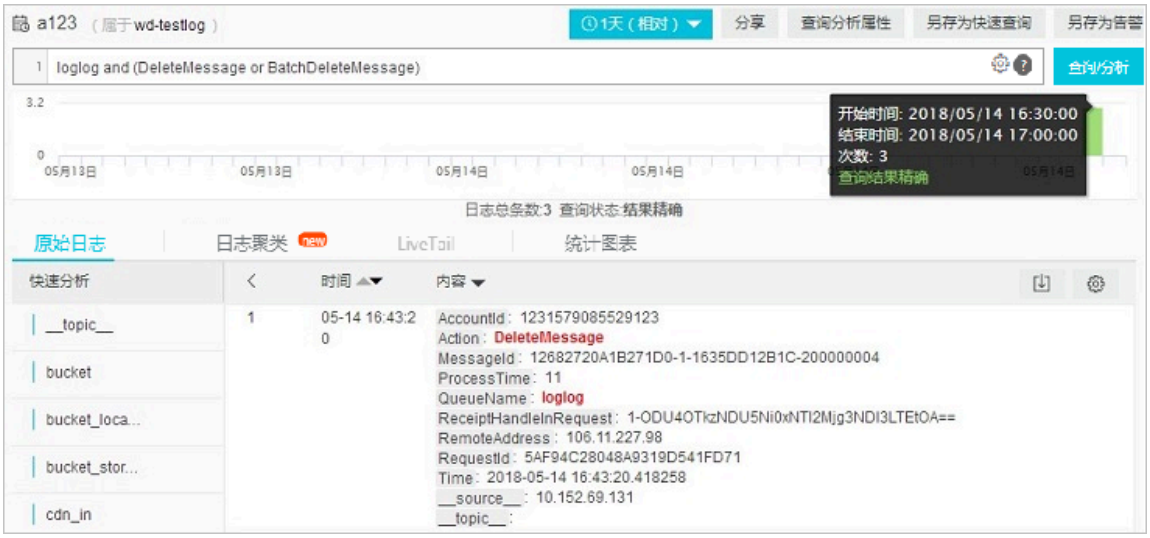
- 步骤

  1. 在搜索框中输入队列名称和删除操作。格式为：`$queuename and (DeleteMessage or BatchDeleteMessage)`。
  2. 选择合适的时间范围后，单击查询/分析即可查看该队列消息删除量。

- 示例

  查看名为loglog的队列消息删除量。

  - 查询语句：`loglog and (DeleteMessage or BatchDeleteMessage)`
  - 查询结果：如下图所示，搜索结果中展示了loglog队列消息的删除日志，您可以查看删除量。



查看主题消息的消息轨迹

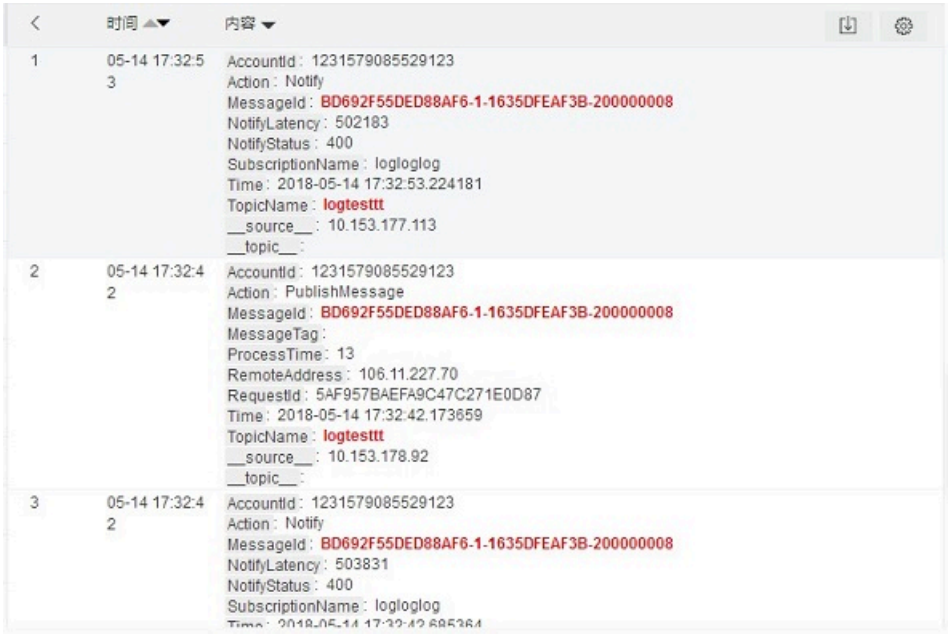
- 步骤

  1. 在搜索框中输入主题名称和messageid。格式为`$topicname and $messageid`。
  2. 选择合适的时间范围后，单击查询/分析即可查看该主题消息的消息轨迹。

- 示例

  查看名为logtesttt主题中MessageId为`BD692F55DED88AF6-1-1635DFEAF3B-200000008`的消息轨迹。

  - 查询语句：`logtesttt and BD692F55DED88AF6-1-1635DFEAF3B-200000008`
  - 查询结果：如下图所示，搜索结果中展示了logtesttt主题中消息从发送到通知的过程。

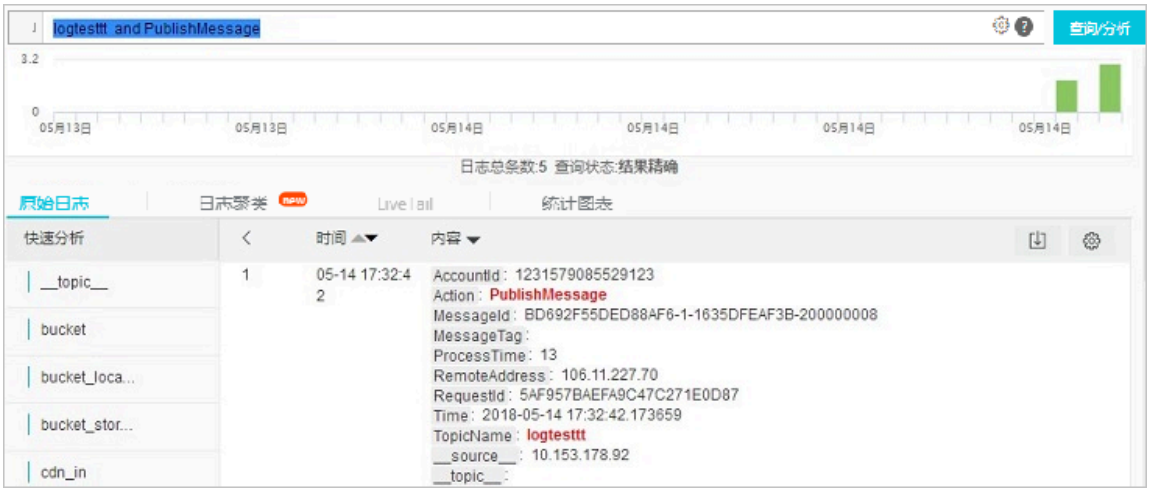


查看主题消息发布量

- 步骤

  1. 在搜索框中输入主题名称和发布操作。格式：`$topicname and PublishMessage`。
  2. 选择合适的时间范围后，单击查询/分析即可查看该主题消息发布量。

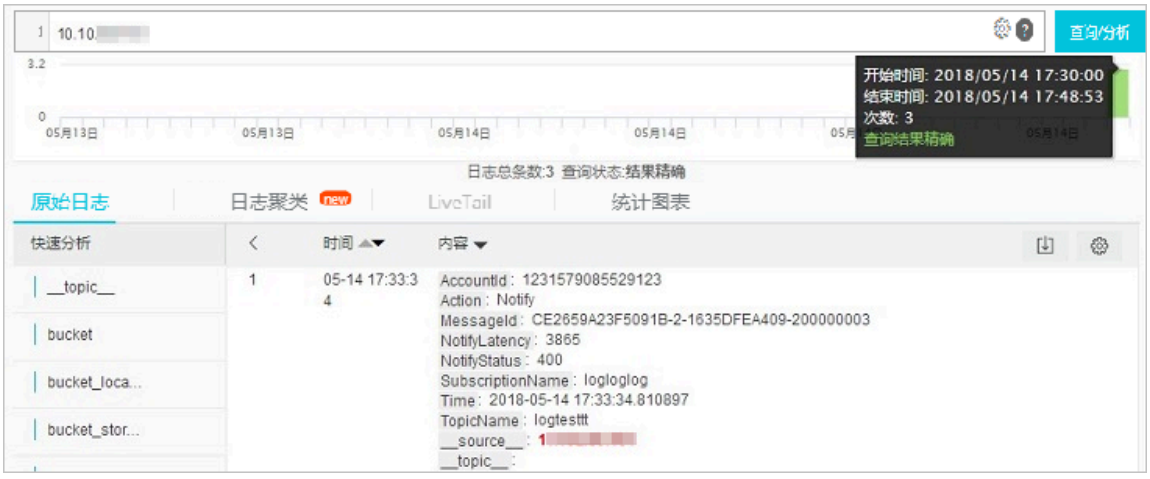
- 示例：查询名为logtesttt的主题消息发布量。
  - 查询语句：`logtesttt and PublishMessage`
  - 查询结果：如下图所示，当前查询时段内，有5条消息发布记录。



查看某个客户端消息处理量

- 步骤
  1. 索框中输入客户端IP。格式：`$ClientIP`，如果希望查询某个客户端的某类操作日志，搜索框中增加具体操作即可，例如：`$ClientIP and (SendMessage or BatchSendMessage)`。
  2. 选择合适的时间范围后，单击搜索按钮即可查看该客户端所有的消息操作日志。
- 示例：

  查询IP为10.10.XX.XX的客户端消息处理量。
  - 查询语句：`10.10.XX.XX`,
  - 查询结果：如下图所示，当前查询时段内，有3条消息处理记录。



实时计算 & 离线计算

- 实时计算：使用Spark、Storm或StreamCompute，Consumer Library等方式可以实时对消息服务日志进行分析。例如：
  - 对一个队列而言，Top 10 消息的产生者、消费者分别是谁哪些IP?
  - 生产和消费的速度是否均衡？某些消费者在处理延时上是否有瓶颈?
- 离线：使用MaxCompute 或 E-MapReduce/Hive进行大时间跨度的计算。
  - 最近一周内，消息从发布到被消费平均延迟是什么?
  - 对比升级前和升级后两个时间段内性能变化如何?

3.9 查询分析-程序日志

程序日志（AppLog）有什么特点?

- 内容最全：程序日志是由程序员给出，在重要的地点、变量数值以及异常都会有记录，可以说线上90%以上Bug都是依靠程序日志输出定位到的。
- 格式比较随意：代码往往经过不同人开发，每个程序员都有自己爱好的格式，一般非常难进行统一，并且引入的一些第三方库的日志风格也不太一样。
- 有一定共性：虽然格式不统一，但一般都会有一些共性的地方。例如对Log4J日志而言，会有如下几个固定字段：
  - 时间
  - 级别（Level）
  - 所在文件或类（file or class）
  - 行数（Line Number）
  - 线程号（ThreadId）

处理程序日志会有哪些挑战?

- 数据量大

  程序日志一般会比访问日志大1个数量级：假设一个网站一天有100W次独立访问，每个访问大约有20逻辑模块，在每个逻辑模块中有10个主要逻辑点需要记录日志。

  则日志总数为：

  ```
  100W * 20 * 10 = 2 * 10^8 条
  ```

  每条长度为200字节，则存储大小为：

  ```
  2 * 10^8 * 200 = 4 * 10^10 = 40 GB
  ```

  这个数据会随着业务系统复杂变得更大，一天100-200GB日志对一个中型网站而言是很常见的。

- 分布服务器多

  大部分应用都是无状态模式，跑在不同框架中，例如:

  - 服务器
  - Docker（容器）
  - 函数计算（容器服务）

  对应实例数目会从几个到几千，需要有一种跨服务器的日志采集方案。

- 运行环境复杂

  程序会落到不同的环境上，例如：

  - 应用相关的会在容器中
  - API相关日志会在FunctionCompute中
  - 旧系统日志在传统IDC中
  - 移动端相关日志在用户处
  - 网页端（M站）在浏览器里

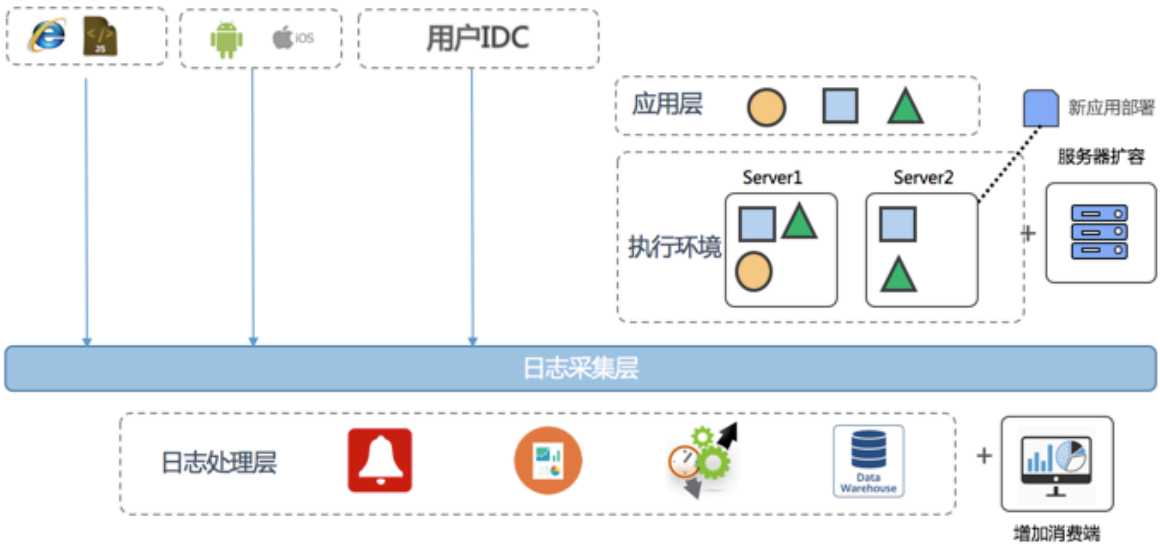
  为了能够获得全貌，我们必须把所有数据统一并存储起来。

如何解决程序日志需求

统一存储

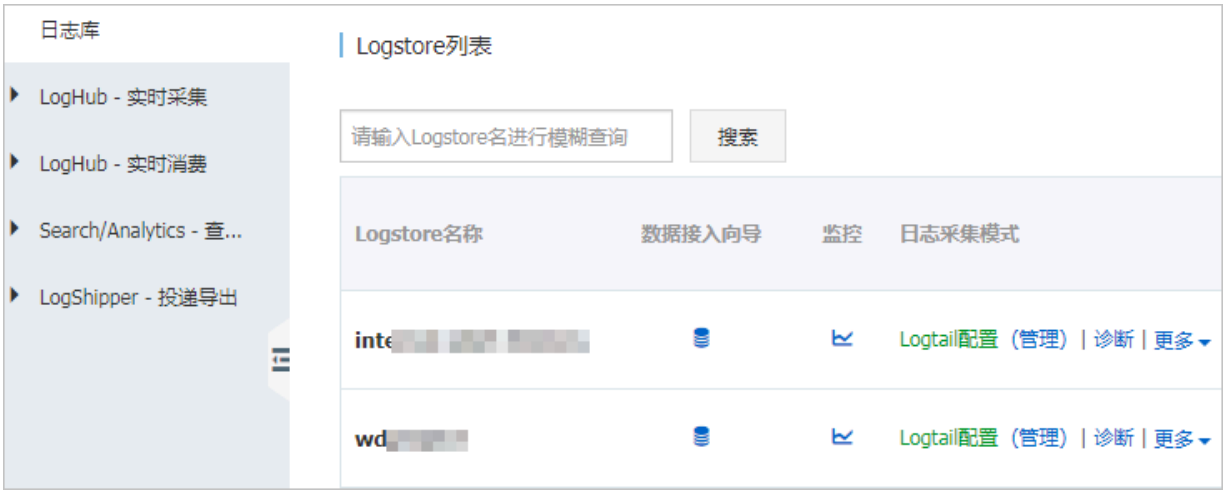
目标：要把各渠道数据采集到一个集中化中心，打通才可以做后续事情。

我们可以在日志服务中创建一个项目来存储应用日志，日志服务提供30+种日志采集手段：无论是在硬件服务器中埋点，还是网页端JS，或是服务器上输出日志都支持采集。



在服务器日志上，除了使用SDK等直接写入外，日志服务提供便捷、稳定、高性能Agent-Logtail。Logtail提供Windows、Linux两个版本，在控制台定义好机器组，日志采集配置后，就能够实时将服务日志进行采集。

在创建完成一个日志采集配置后，我们就可以在项目中操作各种日志了。



可能有人要问到，日志采集Agent非常多，有Logstash，Flume，FluentD，以及Beats等，Logtail和这些相比有什么特点吗?

- 使用便捷：提供API、远程管理与监控功能，融入阿里集团百万级服务器日志采集管理经验，配置一个采集点到几十万设备只需要几秒钟。
- 适应各种环境：无论是公网、VPC、用户自定义IDC等都可以支持，https以及断点续传功能使得接入公网数据也不再话下。
- 性能强，对资源消耗非常小：经过多年磨练，在性能和资源消耗方面比开源要好。

快速查找定位

目标：无论数据量如何增长、服务器如何部署，都可以保证定位问题时间是恒定的。

例如有一个订单错误，一个延时很长，我们如何能够在一周几TB数据量日志中快速定位到问题。其中还会涉及到各种条件过滤和排查等。

1. 例如我们对于程序中记录延时的日志，调查延时大于1秒，并且方法以Post开头的请求数据：

```
Latency > 1000000 and Method=Post*
```

2. 对于日志中查找包含error关键词，不包含merge关键词的日志：

   - 一天的结果

   - 一周的结果

   - 更长时间结果

   这些查询都是在不到1秒时间内可以返回。

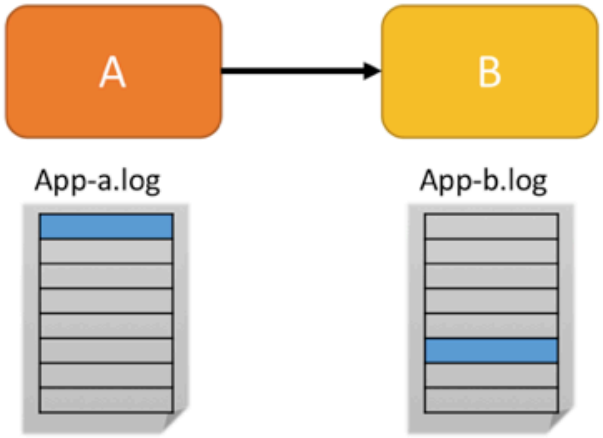
关联分析

关联有两种类型，进程内关联与跨进程关联。我们先来看看两者有什么区别：

- 进程内关联：一般比较简单，因为同一个函数前后日志都在一个文件中。在多线程环节中，我们只要根据线程Id进行过滤即可。
- 跨进程关联：跨进程的请求一般没有明确线索，一般会通过RPC中传入TracerId来进行关联。

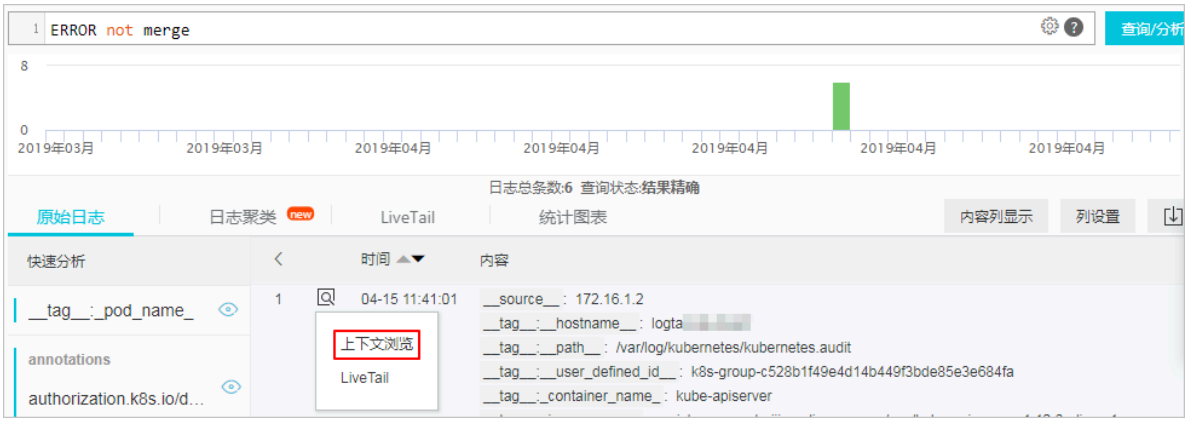
同进程，同一个日志

不同进程，不同日文件不同位置



- 上下文关联

  还是以使用日志服务控制台举例，线上通过关键词查询定位到一个异常日志：



  点击上下文浏览后，即跳转到前后N条上下文。

  - 显示框可以通过更早、更新等按钮加载更多上下文。
  - 也可以输入关键词进行高亮显示。

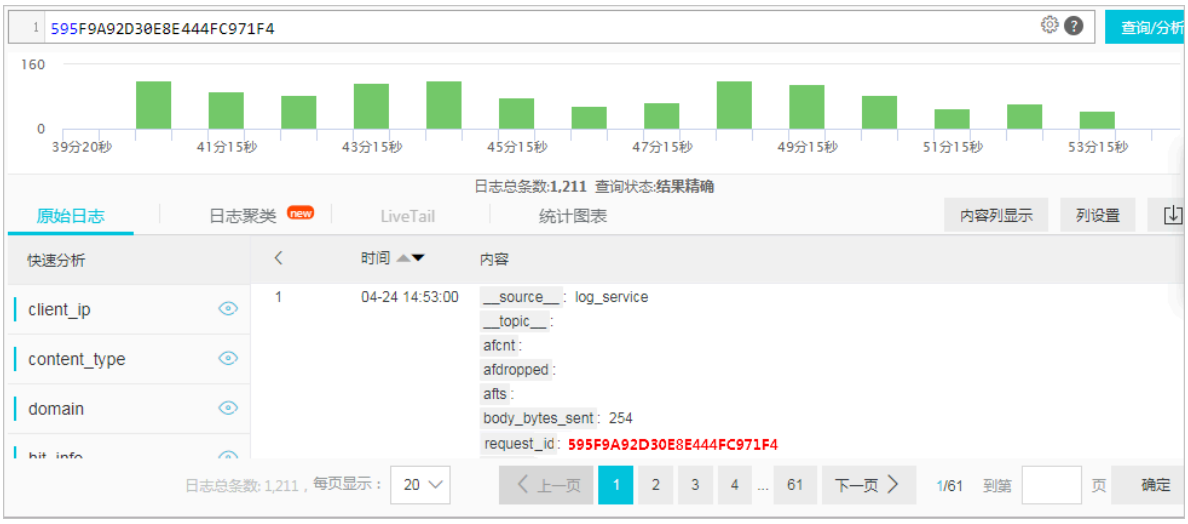
- 跨进程关联

跨进程关联也叫Tracing，最早的工作是Google在2010年的那篇著名“Dapper, a Large-Scale Distributed Systems Tracing Infrastructure”，之后开源界借鉴了Google思想做成过了平民化的各种Tracer版本。目前比较有名的有：

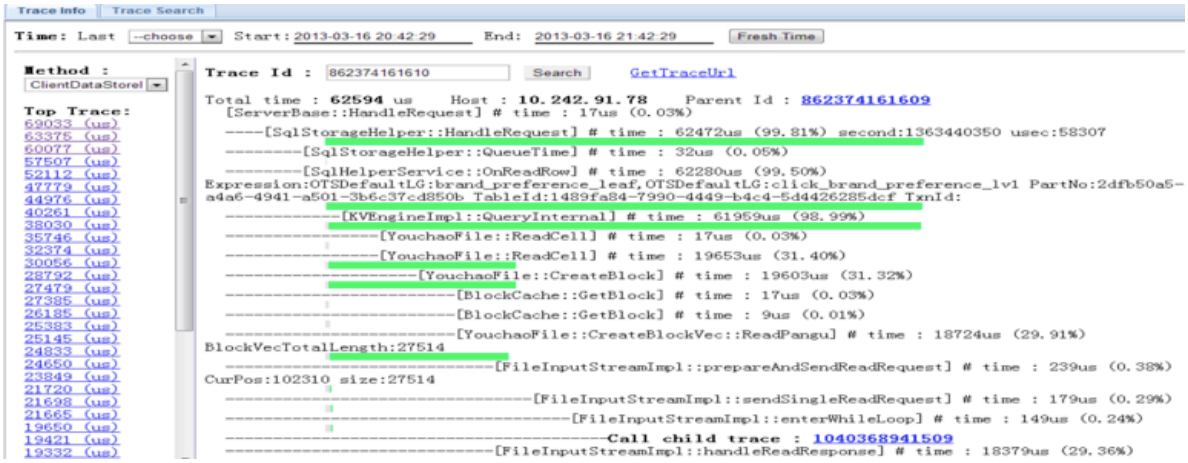
- Dapper(Google): 各 tracer基础
- StackDriver Trace (Google)，现在兼容了ZipKin
- Zipkin：twitter开源的Tracing系统
- Appdash：golang版本
- 鹰眼：阿里巴巴集团中间件技术技术部研发
- X-ray：AWS在2016年Re：Invent上推出技术

如果从0开始使用Tracer会相对容易一些，但在现有系统中去使用，则会有改造的代价和挑战。

今天我们可以基于日志服务，实现一个基本Tracing功能：在各模块日志中输出Request_id，OrderId等可以关联的标示字段，通过在不同的日志库中查找，即可拿到所有相关的日志。



例如我们可以通过SDK查询 前端机，后端机，支付系统，订单系统等日志，拿到结果后做一个前端的页面将跨进程调用关联起来，以下就是基于日志服务快速搭建的Tracing系统。



统计分析

查找到特点日志后，我们有时希望能做一些分析，例如线上有多少种不同类型的错误日志。

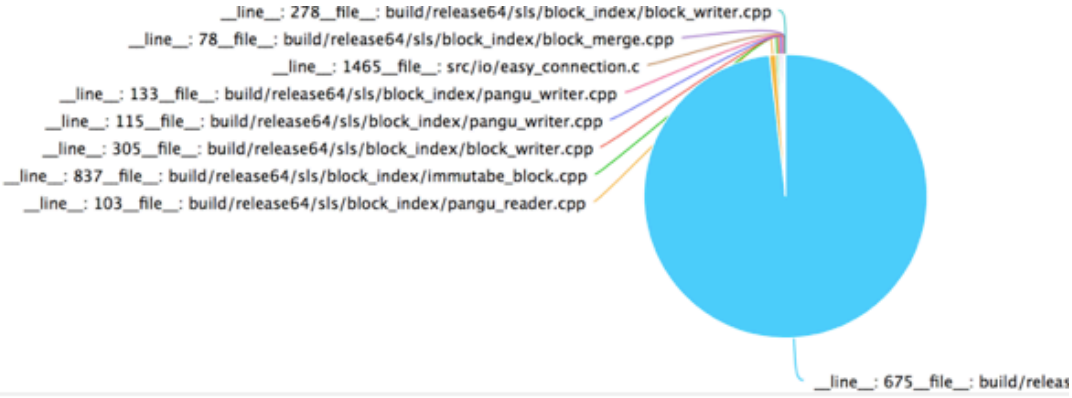
1. 我们先通过对`__level__`这个日志级别字段进行查询，得知一天内有20个错误：

```
__level__:error
```

2. 接下来，我们可以根据file，line这两个字段（确定唯一的日志类型）来进行统计聚合。

```
__level__:error | select __file__, __line__, count(*) as c group by
__file__, __line__ order by c desc
```

即可得出所有错误发生的类型和位置的分布。



__line__	c	__file__
675	2670	build/release64/sls/shennong_worker/PackageDispatcher.cpp
103	21	build/release64/sls/block_index/pangu_reader.cpp
837	7	build/release64/sls/block_index/immutabe_block.cpp
305	6	build/release64/sls/block_index/block_writer.cpp
115	6	build/release64/sls/block_index/pangu_writer.cpp
133	5	build/release64/sls/block_index/pangu_writer.cpp
1465	3	src/io/easy_connection.c
78	1	build/release64/sls/block_index/block_merge.cpp
278	1	build/release64/sls/block_index/block_writer.cpp

其他还有诸如根据错误码、高延时等条件进行IP定位与分析等。

其他

- 备份日志审计

  可以将日志备份至OSS或存储成本更低的IA，或直接到MaxCompute。

- 关键词报警

  目前有如下方式可以进行报警。

  - 日志服务告警
  - 通过云监控报警

- 日志查询权限分配管理。

  可以通过子账号+授权组方法隔离开发、PE等权限。

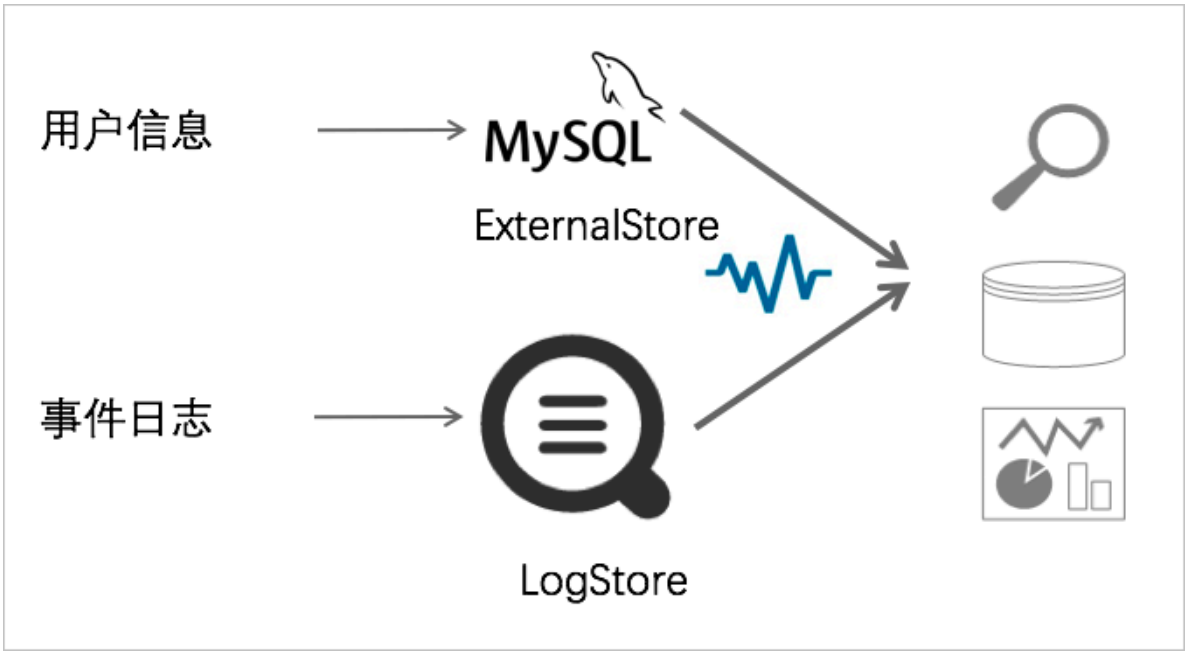
  最后说一下价格与成本，程序日志主要会用到日志服务LogHub + LogSearch功能，和开源方案对比，在查询成本上是开源方案25%，使用非常便捷，令您的开发工作事半功倍。

3.10 查询分析-数据库与日志关联分析

在日志分析场景中，我们经常会遇到数据分散在各地的场景。而我们需要用日志和数据库中的数据对用户进行分层统计，将最后的计算结果写入数据库中供报表系统查询，因此，我们要在日志服务Logstore和其他数据源中进行关联查询。

背景信息

- 用户日志数据：以游戏日志为例，一条经典的游戏日志，包括操作、目标、血、魔法值、网络、支付手段、点击位置、状态码、用户id等。
- 用户元数据：日志表示的是增量的事件，一些静态的用户信息，例如用户的性别、注册时间、地区等是固定不变的，在客户端很难获取，不能够打印到日志里。我们把这些信息称为用户元信息。
- 日志服务和MySQL关联分析：日志服务查询分析引擎，提供跨Logstore和ExternalStore的查询分析功能，使用SQL的join语法把日志和用户元信息关联起来，用来分析跟用户属性相关的指标。除在查询过程中引用ExternalStore，日志服务还支持将计算结果直接写入ExternalStore中（例如MySQL），方便结果的进一步处理。
  - 日志服务Logstore：提供日志的收集、存储、查询分析。
  - 日志服务ExternalStore：映射到RDS表，开发者把用户信息放到RDS表中。



操作步骤

1. 采集日志到日志服务。
   - 移动端日志采集：Android，iOS。
   - 服务器日志采集：ilogtail
2. 创建用户属性表。

   创建一张chiji_user表，保存用户的id、昵称、性别、年龄、注册时间、账户余额、注册省份。

```
CREATE TABLE `chiji_user` (
  `uid` int(11) NOT NULL DEFAULT '0',
  `user_nick` text,
  `gender` tinyint(1) DEFAULT NULL,
  `age` int(11) DEFAULT NULL,
  `register_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
  `balance` float DEFAULT NULL,
  `province` text, PRIMARY KEY (`uid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

3. 创建ExternalStore。

   a. 创建ExternalStore需要使用日志服务CLI，因此需要安装CLI。

```
pip install -U aliyun-log-cli
```

   b. 创建ExternalStore，指定所属的Project，以及ExternalStore的配置文件/root/config.json。

```
aliyunlog log create_external_store --project_name="log-rds-demo"
 --config="file:///root/config.json"
```

   c. 在配置文件中，指定外部存储的名称，参数。RDS VPC需要指定的参数有：vpc-id，RDS实例id，域名、端口、用户名、密码、数据库和表名、RDS所属region。

```
{
     "externalStoreName": "chiji_user",
     "storeType": "rds-vpc",
     "parameter": {
     "vpc-id": "vpc-m5eq4irc1pucpk85f****",
     "instance-id": "rm-m5ep2z57814qs****",
     "host": "example.com",
     "port": "3306",
     "username": "testroot",
     "password": "123456789",
     "db": "chiji",
     "table": "chiji_user",
     "region": "cn-qingdao"
  }
}
```

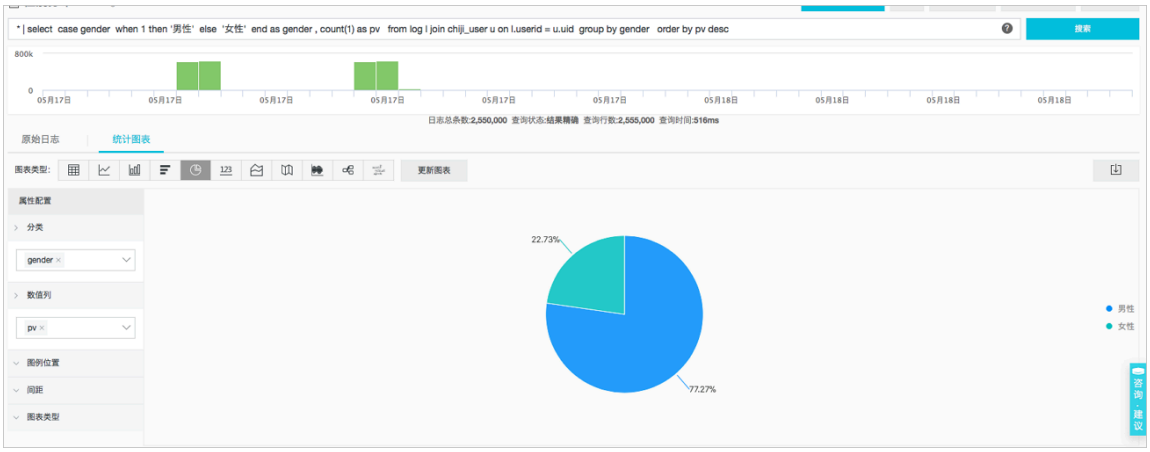
4. 添加白名单。
   - 在RDS中，添加白名单地址`100.104.0.0/16`。
   - 如果是MySQL，请添加该地址到安全组。

5. 关联分析。

   · 分析活跃用户的性别分布。

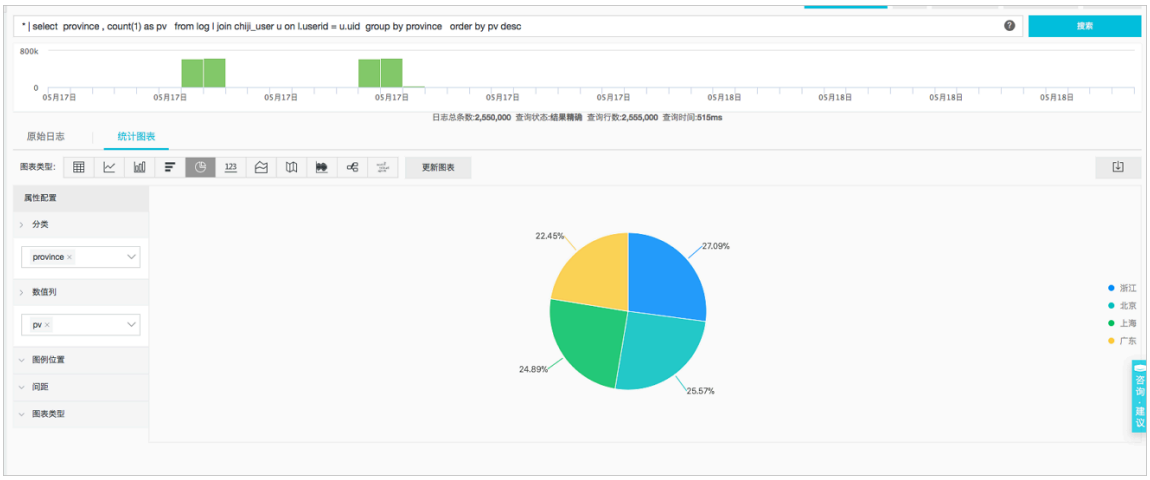
   使用join语法，通过指定日志中的userid和RDS中的uid相等来关联日志和用户属性。

```
* | select case gender when 1 then '男性' else '女性' end as gender
 , count(1) as pv from log l join chiji_user u on l.userid = u.uid
 group by gender order by pv desc
```



   · 分析不同省份活跃度。

```
* | select province , count(1) as pv from log l join chiji_user u
on l.userid = u.uid group by province order by pv desc
```



   · 分析不同性别的消费情况。

```
* | select case gender when 1 then '男性' else '女性' end as gender
 , sum(money) as money from log l join chiji_user u on l.userid =
u.uid group by gender order by money desc
```

6. 保存查询分析结果。

   a. 创建结果表，该表存储每分钟的PV值。

```
CREATE TABLE `report` (
```

```
  `minute` bigint(20) DEFAULT NULL,
  `pv` bigint(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

b. 参考上述创建ExternalStore的方法给report表创建ExternalStore，然后将结果保存到report。

```
* | insert into report select __time__- __time__ % 300 as min,
count(1) as pv group by min
```

SQL返回的结果是最终输出到RDS中的行数。

```
mysql> select * from report;
+------------+-------+
| minute     | pv    |
+------------+-------+
| 1526448600 |  3000 |
| 1526448540 |  9900 |
| 1526448780 |  3100 |
| 1526448480 |  5400 |
| 1526448720 |  3000 |
| 1526448960 |  3000 |
| 1526448900 |  3000 |
| 1526449080 |  3000 |
| 1526449140 |  3000 |
| 1526448660 |  2900 |
| 1526449260 |  3000 |
```

3.11 查询分析-日志服务与OSS外表关联分析

在日志分析场景中，我们经常遇到日志中的信息不完善。例如，日志中包含了用户的点击行为，但是却缺少用户的属性，例如注册信息、资金等信息。 而分析日志的时候，往往需要联合分析用户的属性和行为，例如分析用户地域对付费习惯的影响等。

背景信息

日志服务提供的这种跨数据源（OSS）的分析能力，可以帮助用户解决以下问题：

- 节省费用
  - 异构数据，根据数据的特性选择合适的存储系统，最大限度的节省成本。 对于更新少的数据，选择存放在OSS上，只需要支付少量的存储费用。如果存放在MySQL上，还要支付计算实例的费用。
  - OSS是阿里云的存储系统，可以走内网读取数据，免去了流量费用。

- 节省精力

  在我们轻量级的联合分析平台中，不需要搬迁数据到同一个存储系统中，节省了用户精力。

- 节省时间
  - 当用户需要分析数据时，使用一条SQL，秒级别即可获得结果。
  - 把常用的视图，定义成报表，打开即可看到结果。

操作步骤

1. 上传CSV文件到OSS。

   a. 定义一份属性文件，包含用户ID、用户昵称、性别、省份、年龄。

   ```
   userid,nick,gender,province,age
   1,阳光男孩,male,上海,18
   2,么么茶,female,浙江,19
   3,刀锋1937,male,广东,18
   ```

   b. 保存该文件为`user.csv`，使用osscmd上传到OSS。

   ```
   osscmd   put  ~/user.csv   oss:/testossconnector/user.csv
   ```

2. 定义外部存储。

   通过SQL定义一个存储名为user_meta虚拟外部表。

   ```
   * | create table user_meta ( userid bigint, nick varchar, gender
    varchar, province varchar, gender varchar,age bigint) with (
   endpoint='example.com',accessid='<youraccessid>',accesskey='<
   accesskey>',bucket='testossconnector',objects=ARRAY['user.csv'],type
   ='oss')
   ```

   说明：

   - 在SQL中，需要指定三部分信息：
     - 表的schema：包含的列，以及每一列的属性。
     - OSS访问信息：OSS的域名，accessid 和accesskey。
     - OSS文件信息：文件所属bucket，以及文件的object路径。

- objects是一个数组，可以同时包含多个文件。

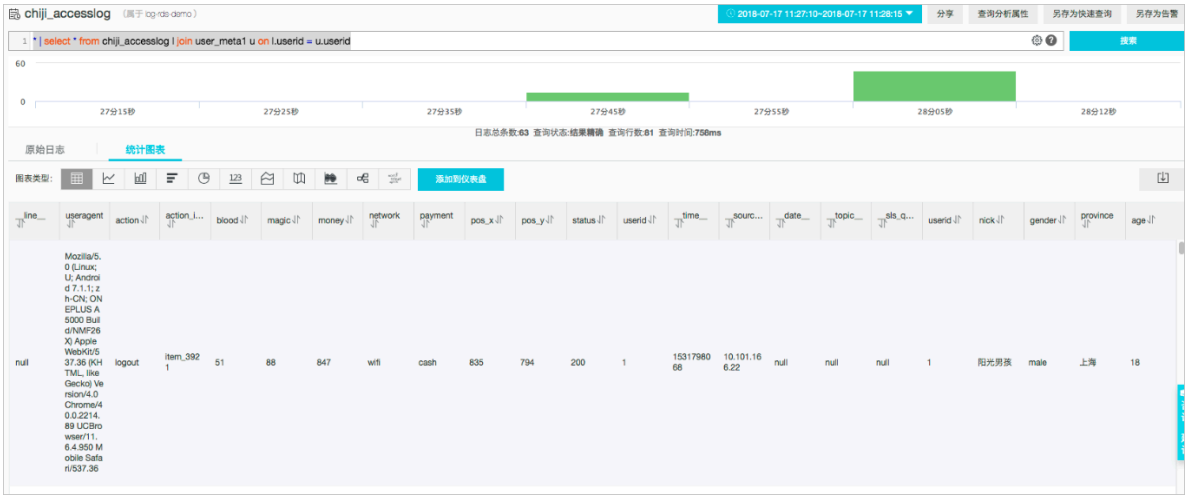
执行结果为true，表示执行成功。

执行SQL查看结果：`select * from user_meta`。

3. 联合分析。

在原始日志中，包含了用户的ID信息，我们可以通过SQL关联日志中的ID和OSS文件中的userid，补全日志的信息。

```
* | select * from chiji_accesslog l join user_meta1 u on l.userid = u.userid
```

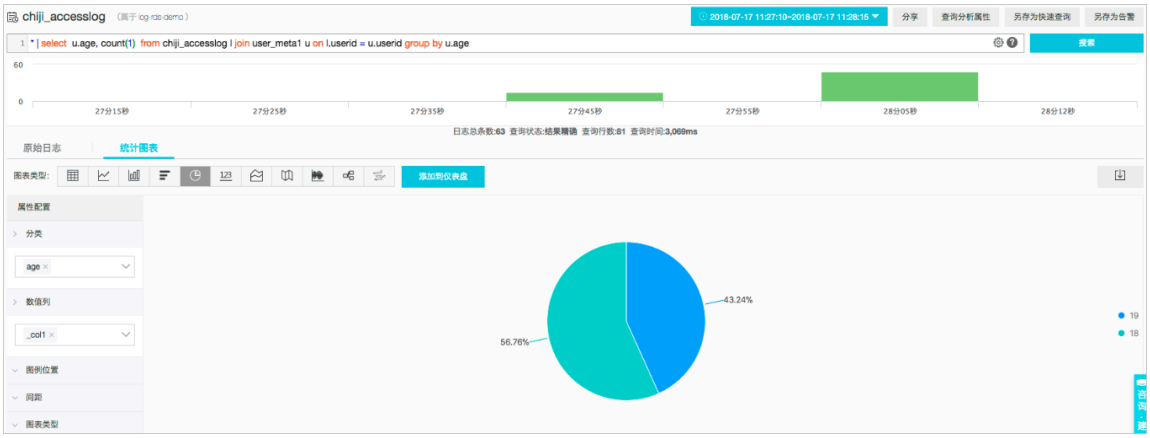


· 统计用户性别的访问情况：

```
* | select u.gender, count(1) from chiji_accesslog l join user_meta1 u on l.userid = u.userid group by u.gender
```

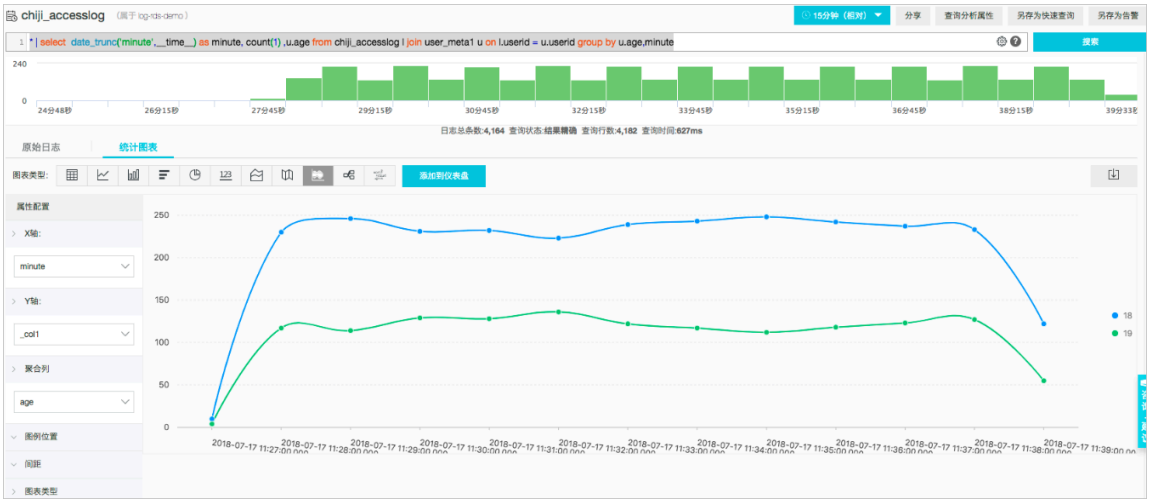
- **统计用户年龄的访问情况：**

```
* | select u.age, count(1) from chiji_accesslog l join user_meta1
u on l.userid = u.userid group by u.age
```



- **统计不同年龄段在时间维度上的访问趋势：**

```
* | selectdate_trunc('minute',__time__) as minute, count(1) ,u.age
 from chiji_accesslog l join user_meta1 u on l.userid = u.userid
group by u.age,minute
```



4 消费

4.1 消费-通过函数计算清洗数据

日志服务的LogHub是流式的数据中心，日志写入后可实时消费。日志服务ETL面向的正是这些流式写入的数据，提供秒级实时的ETL作业。

概述

ETL（Extract-Transform-Load）用来描述将数据从来源端经过抽取(Extract)、转换(Transform)、加载(Load)至目的端的过程。传统ETL是构建数据仓库的重要一环，用户从数据源抽取出所需的数据，经过数据清洗，最终按照预先定义好的数据仓库模型，将数据加载到数据仓库中去。随着业务需求的日益增加，不同系统的相互大批量数据交互也已成为常态。数据在不同系统中流动起来，有助于充分发掘日志大数据的价值。

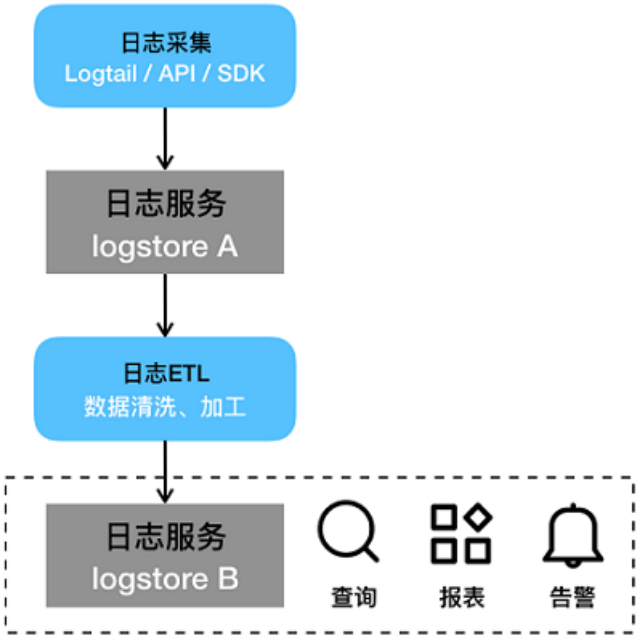
“日志服务+函数计算”ETL的优势

- 一站式采集、存储、加工、分析
- 全托管加工任务，按时间触发，自动重试
- 资源按shard水平扩展，满足大数据需求
- 基于函数计算提供数据加工，弹性资源，按需付费
- ETL对用户透明，提供日志、报警功能
- 持续增加内置函数模板，降低主流需求下的函数开发代价

应用场景

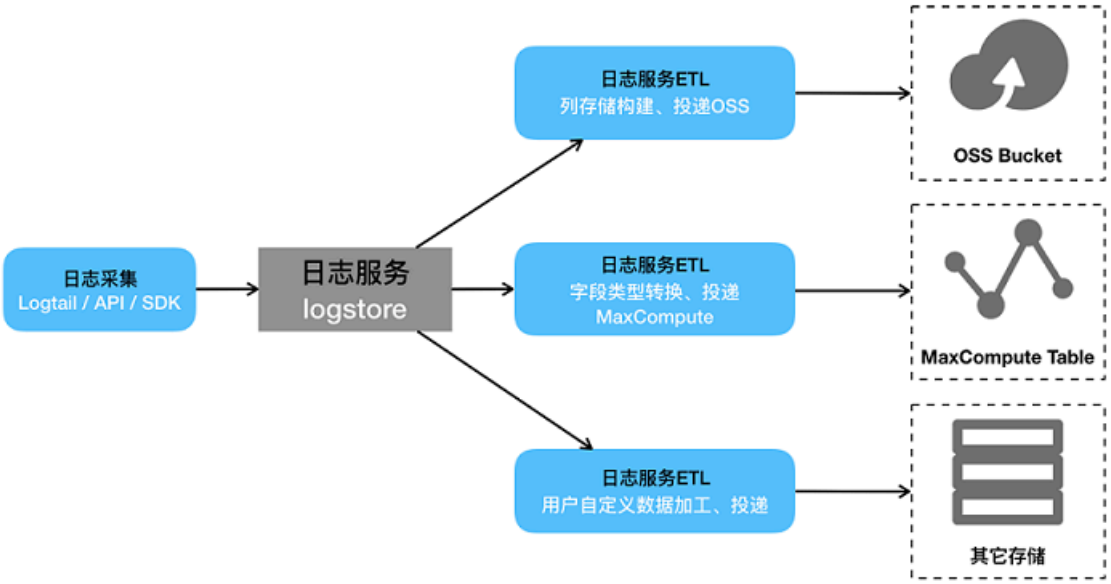
数据清洗、加工场景

通过日志服务，快速完成日志采集、加工、查询、分析。



数据投递场景

为数据的目的端落地提供支撑，构建云上大数据产品间的数据管道。

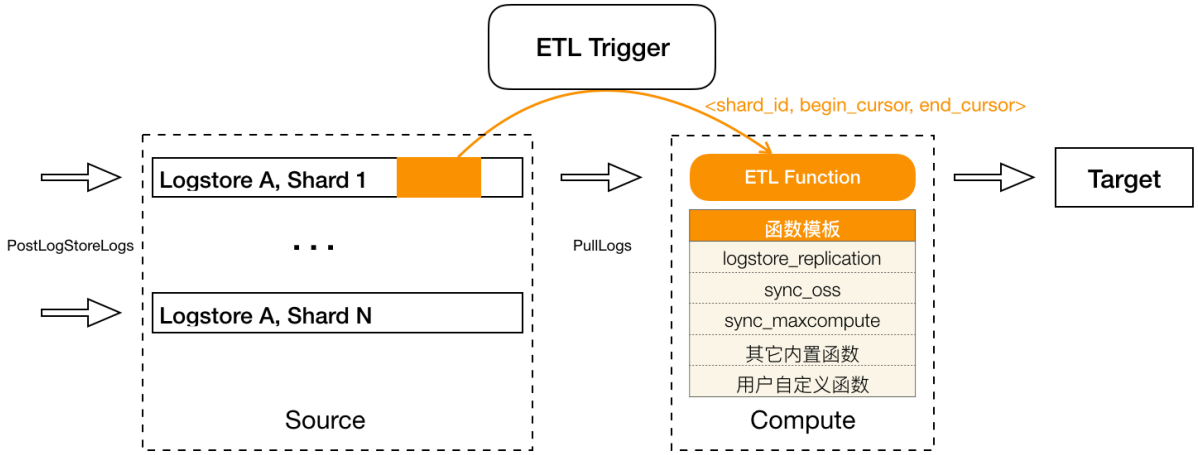


ETL模型

实时数据流处理，基于流的模型。ETL Trigger轮询源logstore下各shard的写入位置，并定时生成三元组信息触发函数执行，该三元组用于标识本次ETL任务所应该处理的数据范围。

通过shard的并发做到水平扩展，shard弹性伸缩保证了ETL的动态伸缩，通过定时器触发作业完成持续的数据加载。

在ETL任务执行层面，考虑UDF的灵活性，加工逻辑会跑在函数计算的函数上，而函数计算提供了按需付费、弹性伸缩能力以及自定义代码执行功能，正是很多云上用户所需要的。另一方面，从用户数据端到端延时、大数据吞吐、SQL易用性角度，日志服务未来也考虑把ETL的runtime扩展到流计算引擎（例如阿里云流计算）上，去服务更多的用户场景。



ETL日志

- ETL过程日志

  这是一类是执行过程日志，这一部分日志是在ETL执行过程中每执行一步的记录关键点和错误，包括某一步骤的开始、结束时间、初始化动作完成情况，模块出错信息等。记录日志的目的是随时可以知道ETL运行情况，如果出错了，可以知道哪里出错。函数运行产生的日志记录了数据加工过程中关键点、异常。

- ETL调度日志

  调度日志只记录ETL任务开始的时间、结束时间，任务是否成功以及成功返回的信息。如果ETL任务出错了，不仅要形成ETL出错日志，而且要向系统管理员发送报警邮件或短信。在调度日志的基础上，可以构建出报表统计ETL的总体运行状况。

应用实例

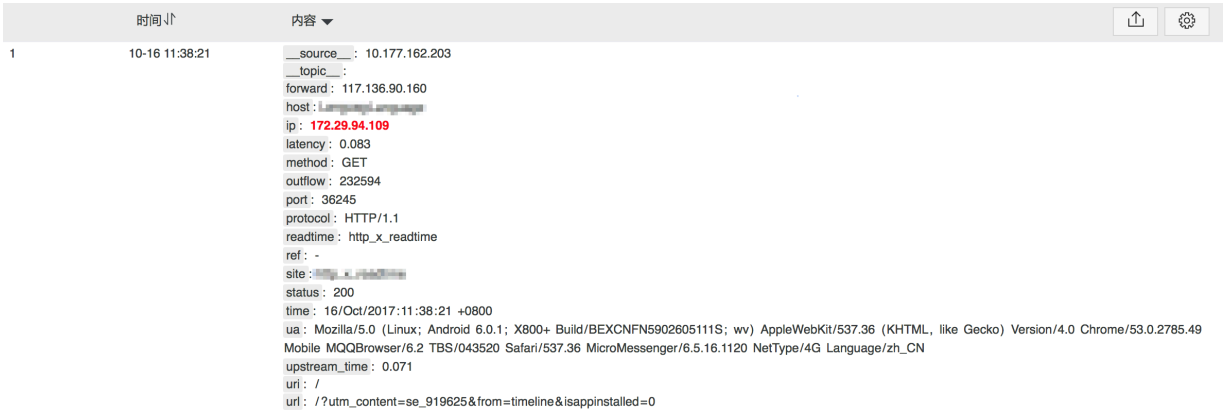
通过Nginx、Apache等HTTP服务器构建的软件，可以记录每一个用户访问日志。通过日志服务+函数计算ETL，可以分析您服务了哪些地区的用户，这些用户通过什么链路访问您的服务。

对于数据分析工程师而言，ETL过程往往占据整个项目工作60%~70%的工作量。日志服务的目标是使用内置的函数模板的前提下，将构建ETL的时间缩短到15分钟内。

步骤1 日志集中化存储

您可以使用日志服务的Logtail客户端快速接入机器上的日志文件。

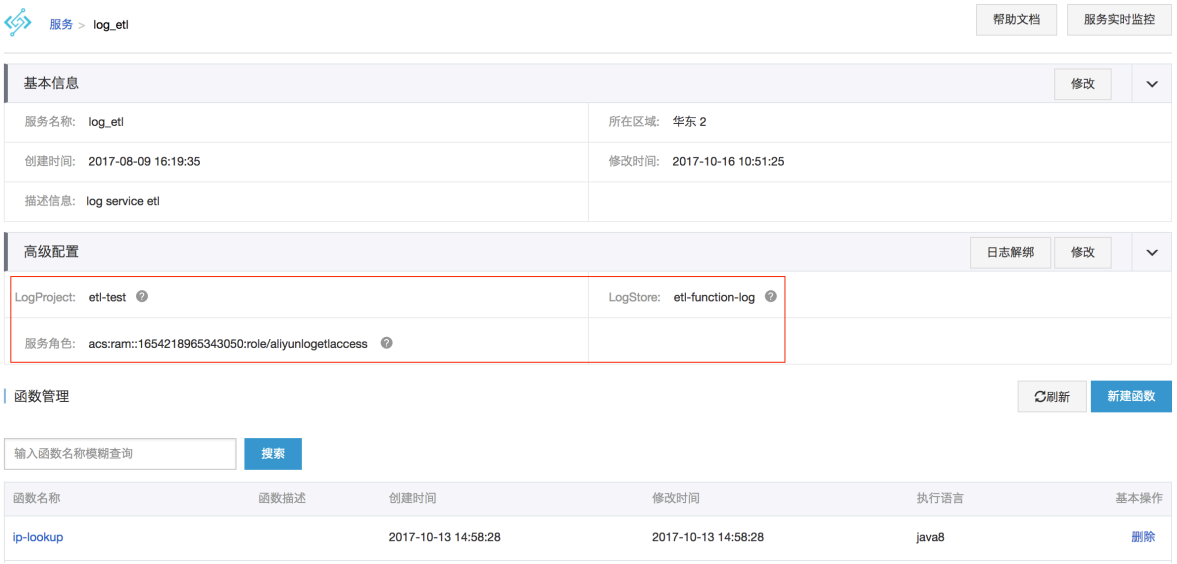
客户端采集Nginx访问日志将会集中存储到日志服务的一个Logstore中，如下图，forward字段的IP记录了用户请求的来源。



步骤2 云端数据加工

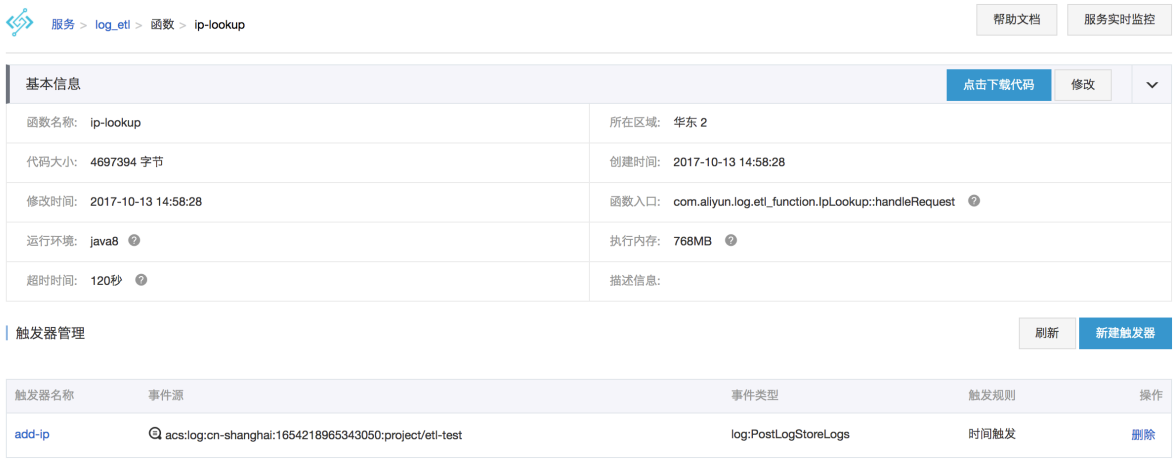
1. 登录函数计算控制台创建Service。

   在高级配置中，建议为ETL Function配置加工过程中的日志记录的存储Logstore，方便通过日志来定位加工过程中的异常行为。为函数授予日志服务AliyunLogFullAccess权限，函数在运行过程中会读源Logstore数据，数据处理后再写到目标Logstore。



2. 通过内置模板创建函数。

默认的函数配置如下：



3. 在函数上新建日志服务触发器。

日志服务触发器配置如下。

指定数据源为第一步中采集到中心化Nginx日志Logstore，例如本例子的Project`etl-test`/`logstore:nginx_access_log`。

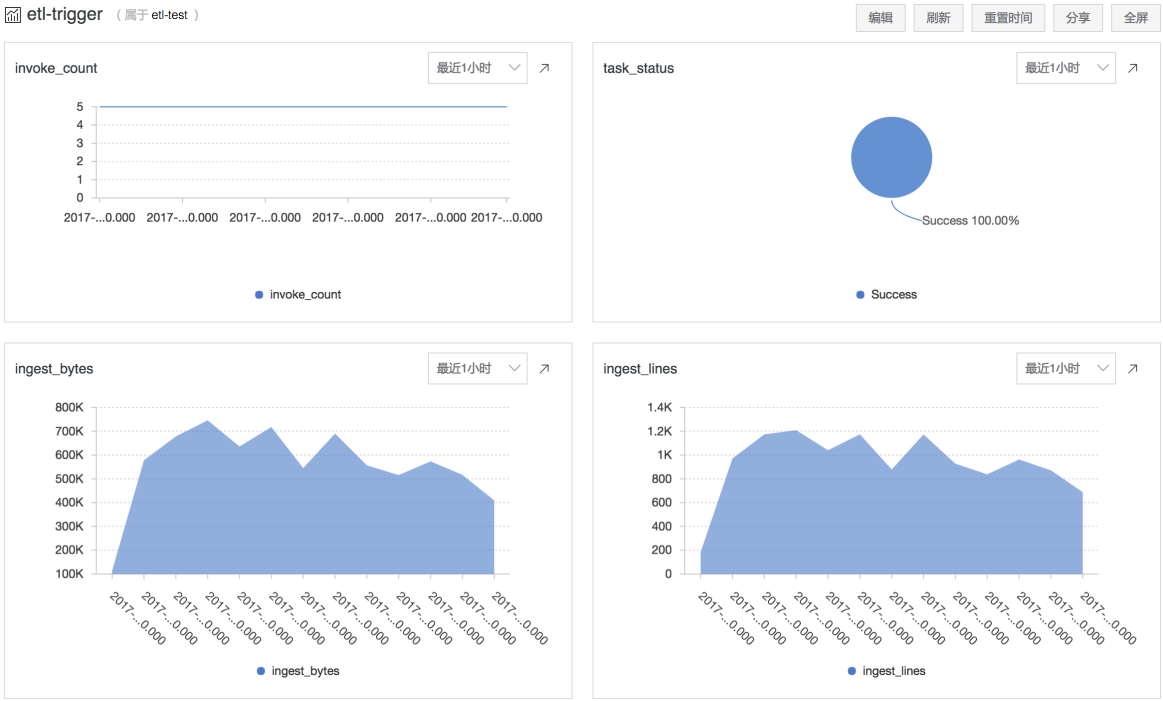
日志服务将轮询Logstore的数据，当数据持续产生时，每60秒创建一次ETL任务，并调用函数执行。触发函数执行以及函数执行结果将会记录到触发器日志logstore：etl-trigger-log中。

不同函数的实现和功能不同，其配置也不相同，ip-lookup的详细配置项说明请参考README。

4. 保存配置，等待1分钟后ETL任务开始执行。

   可以关注一下ETL过程日志、调度日志，按如上配置，分别在logstore：etl-function-log、etl-trigger-log。

   可以通过查询语句构建出如本文日志部分所示的报表：



   - 截图左上角是每分钟调度函数执行的触发次数，构建自查询语句。

     ```
     project_name : etl-test and job_name : ceff019ca3d077f85aca
     ad35bb6b9bba65da6717 | select from_unixtime(__time__ - time % 60)
     as t, count(1) as invoke_count group by from_unixtime(__time__ -
     time % 60) order by t asc limit 1000
     ```

   - 截图右上角是ETL任务成功、失败的比例，构建自查询语句。

     ```
     project_name : etl-test and job_name : ceff019ca3d077f85aca
     ad35bb6b9bba65da6717 | select task_status, count(1) group by
     task_status
     ```

   - 截图左下角是每5分钟的摄入的日志字节数，构建自查询语句。

     ```
     project_name : etl-test and job_name : ceff019ca3d077f85aca
     ad35bb6b9bba65da6717 and task_status : Success | select from_unixt
     ime(__time__ - time % 300) as t, sum(ingest_bytes) as ingest_byt
     es group by from_unixtime(__time__ - time % 300) order by t asc
     limit 1000
     ```

- 截图右下角则是每5分钟摄入处理的日志行数，构建自查询语句。

```
project_name : etl-test and job_name : ceff019ca3d077f85aca
ad35bb6b9bba65da6717 and task_status : Success | select from_unixt
ime(__time__ - time % 300) as t, sum(ingest_lines) as ingest_lin
es group by from_unixtime(__time__ - time % 300) order by t asc
limit 1000
```

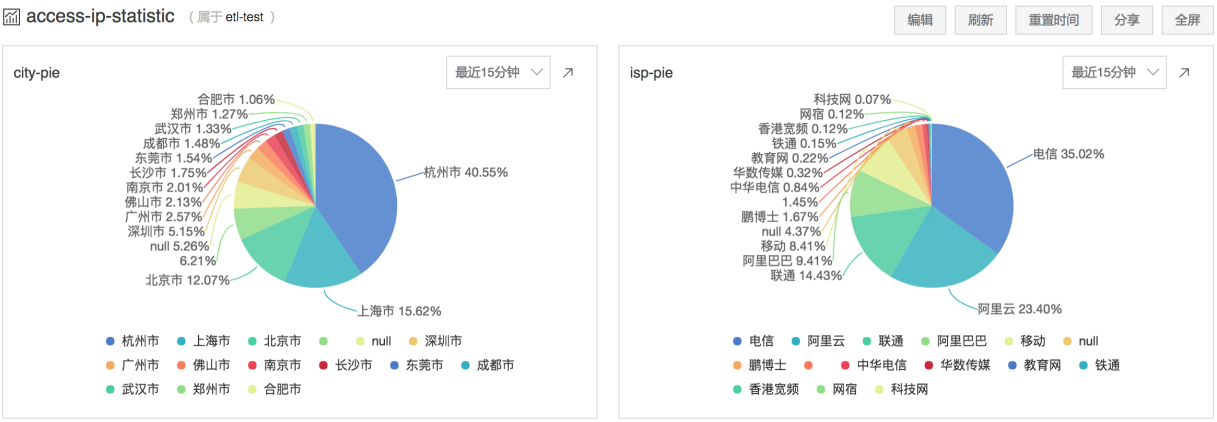
步骤3 加工后数据建模

机器上的Nginx日志经由Logtail实时采集到源logstore，再由ETL准实时加工后写出到目标Logstore。经函数处理后带IP信息数据如下：

对比加工前后，可以发现，新的数据增加了四个字段（country、province、city、isp），可以知道：IP源117.136.90.160的请求来自中国山西太原，运营商是中国移动。

接下来，使用日志服务的日志分析功能查询一个时间段内请求IP的城市和ISP分布。通过如下两个查询语句构建报表：

- `* | select city, count(1) as c group by city order by c desc limit 15`
- `* | select isp, count(1) as c group by isp order by c desc limit 15`



4.2 消费-搭建监控系统

日志服务是阿里云一个重要的基础设施，支撑着阿里云所有集群日志数据的收集和分发。众多应用比如OTS、ODPS、CNZZ等利用日志服务logtail收集日志数据，利用API消费数据，导入下游实时统计系统或者离线系统做分析统计。作为一个基础设施，日志服务具备：

- 可靠性：经过多年阿里集团内部用户的检验，经历多年双十一考验，保证数据的可靠、不丢失。
- 可扩展性：数据流量上升，通过增加shard个数快速动态扩容处理能力。
- 便捷性：一键式管理包括上万台机器的日志收集。

日志服务帮用户完成了日志收集，统一了日志格式，提供API用于下游消费。下游系统可以接入多重系统重复消费，比如导入Spark、Storm做实时计算，也可以导入elastic search做搜索，真正做到了一次收集，多次消费。在众多数据消费场景中，监控是最常见的一种场景。本文介绍阿里云基于日志服务的监控系统。

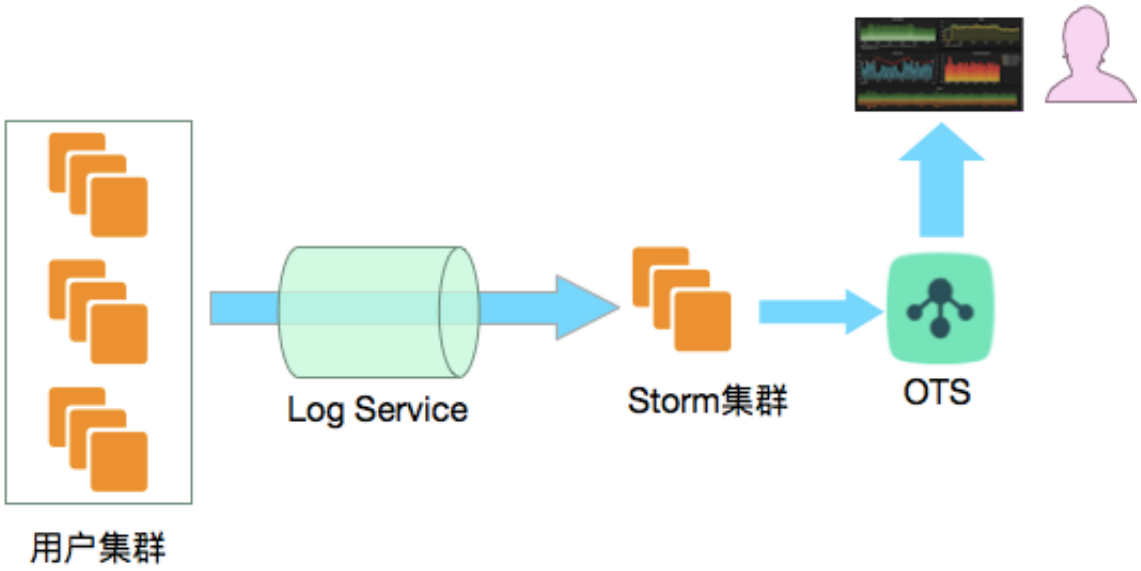
日志服务把所有集群的监控数据作为日志收集到服务端，解决了多集群管理和异构系统日志收集的问题，监控数据统一成格式化的日志发送到日志服务。

日志服务为监控系统提供了：

- 统一的机器管理：安装一次logtail，所有的后续操作在日志服务端进行。
- 统一的配置管理：需要收集哪些日志文件，只要在服务端配置一次，配置会自动下发到所有机器。
- 结构化的数据：所有数据格式化成日志服务的数据模型，方便下游消费。

· 弹性的服务能力：处理大规模数据写入和读取的能力。

图 4-1: 监控系统架构



如何搭建监控系统

1. 收集监控数据

   配置SLS的日志收集，确保日志收集到了日志服务。

2. 中间件使用API消费数据

   通过SDK的PullLog接口从日志服务批量消费日志数据，并且把数据同步到下游实时计算系统。

3. 搭建storm实时计算系统

   选择storm或者其他的类型的实时计算系统，配置计算规则，选择要计算的监控指标，计算结果写入到OTS中。

4. 展示监控信息

   通过读取保存在OTS中的监控数据，在前端展示；或者读取数据，根据数据结果做报警。

4.3 消费-计量计费日志

使用云服务最大好处是按量付费，无需预留资源，因此各云产品都有计量计费需求。这里我们介绍一种基于日志服务计量计费方案，该方案每天处理千亿级计量日志，被众多云产品使用。

计量日志生成计费结果过程

计量日志记录了用户涉及计费的项目，后台计费模块根据计费项和规则进行运算，产生最后账单。例如如下原始访问日志记录了项目（Project）使用情况：

```
microtime:1457517269818107 Method:PostLogStoreLogs Status:200 Source:
10.145.6.81 ClientIP:112.124.143.241 Latency:1968 InFlow:1409 NetFlow
:474 OutFlow:0 UserId:44 AliUid:1264425845****** ProjectName:app-
myapplication ProjectId:573 LogStore:perf UserAgent:ali-sls-logtail
APIVersion:0.5.0 RequestId:56DFF2D58B3D939D691323C7
```

计量计费程序读取原始日志，根据规则生成用户在各维度使用数据（包括流量、使用次数、出流量等）：

C	D	E	F	G	H	I	J	K	L
uid	project	region	inflowsize	writecount	readcount	outflowsize	network_out	shard_size	index_size
1.47543E+15	aquilapreproductionenviron	cn-hangzhou	437	0	0	0	0	48	790
1.76991E+15	ali-tbosstest-log	cn-hangzhou-corp	0	0	0	0	0	92	0
1.72535E+15	ali-icbu-janus-log	cn-shanghai-corp	229879259	0	0	0	0	96	########
1.62572E+15	corp-scmg-admin	cn-hangzhou-stg	15709	0	0	0	0	16	82344
1.19214E+15	dtboost	cn-hangzhou	0	0	0	0	0	48	0
1.63404E+15	md-oa	cn-beijing	0	0	0	0	0	240	0
1.85233E+15	ots_e2e_test_2nd	cn-hangzhou-stg	0	0	0	0	0	8	0
1.2358E+15	wxb-log	cn-hangzhou	466323	0	0	0	0	240	1394118
1.26443E+15	portal-b8568f751df75214dc	cn-hangzhou-stg	0	73811	0	500	73811	600	0
1.26854E+15	ali-pangumaster-log-hangz	cn-hangzhou-stg	98041	0	0	0	0	4	633933
1.85386E+15	daily	cn-hangzhou	205159	0	0	0	0	96	0
1.59853E+15	ali-alipay-siteprobe-test	cn-hangzhou-corp	0	0	0	0	0	184	0
34762362	1111111	cn-qingdao	0	0	0	0	0	96	0

典型计量日志计费场景

- 电力公司：每10秒会收到一条日志，记录该10秒内每个用户ID下该周期内功耗、峰值、均值等，每天、每小时和每月给用户提供账单。
- 运营商：每隔10秒从基站收到时间段内某个手机号码的动作（上网、电话、短信、VoIP），使用量（流量），时长等信息，后台计费服务统计出该区间内消耗资费。
- 天气预测API服务：根据用户调用接口类型、城市、查询类型、结果大小等对用户请求进行收费。

要求与挑战

既要算对，又要算准是一件要求很高的事情，系统要求如下：

- 准确可靠：既不可多算，也不能少算。
- 灵活：支持补数据等场景，例如一部分数据没有推送过来，当需要修正时可以重新计算。
- 实时性强：能够做到秒级计费，对于欠费场景快速切断。

其他需求（Plus）：

- 账单修正功能：在实时计费失败时，我们可以通过理想计费进行对账。
- 查询明细：支持用户查看自己消费明细。

现实中还有两类挑战：

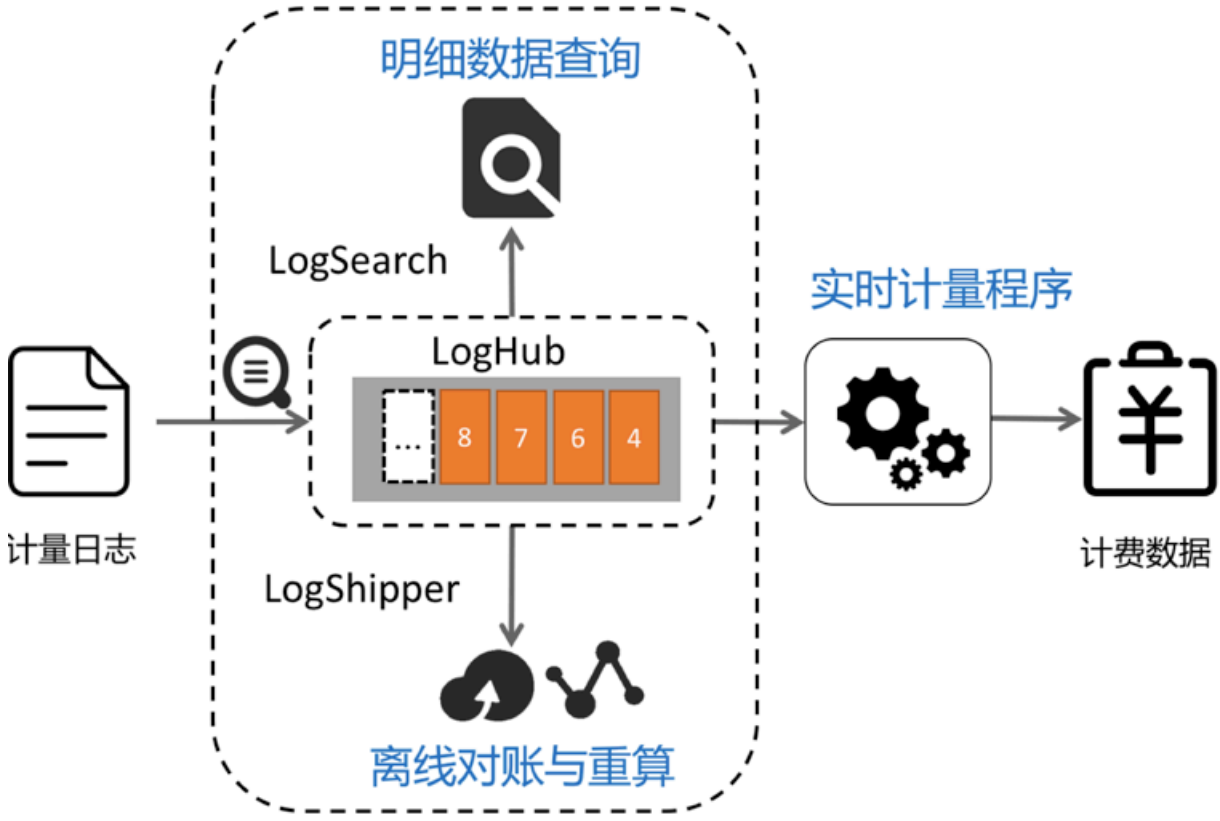
- 不断增长的数据量：随着用户以及调用上升，数据规模会越来越大，如何保持架构的弹性伸缩。
- 容错处理：计费程序可能有Bug，如何确保计量数据与计费程序独立。

本文档主要介绍一种阿里云基于日志服务开发计量计费方案，该方案已在线上稳定运行多年，从未出现过一例算错、延迟等情况，供单价参考。

系统架构

以阿里云日志服务的LogHub功能为例：

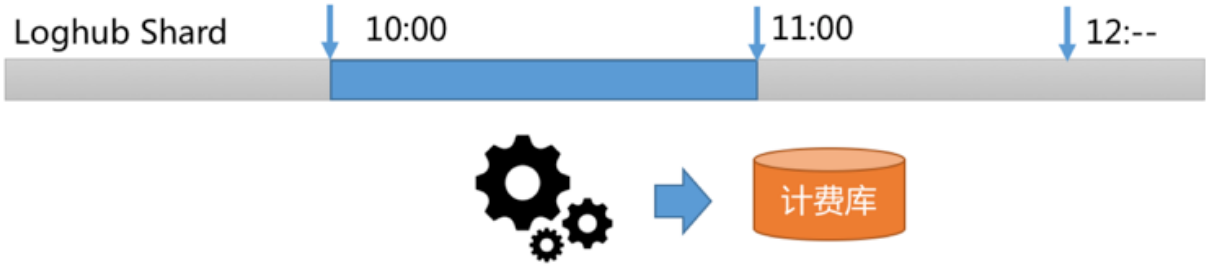
1. 使用LogHub进行计量日志实时采集与计量程序对接：LogHub 支持的30+种API和接入手段，接入计量日志非常容易。
2. 计量程序每隔固定时间消费LogHub中步长数据，在内存中计算结果生成计费数据。
3. （附加）对明细数据查询需求，可以将计量日志配置索引查询。
4. （附加）将计量日志推送至OSS、MaxCompute进行离线存储，进行T+1等对账与统计。



实时计量程序内部结构：

1. 根据LogHub读取接口GetCursor功能，选定某个时间段日志（例如10：00-11：00）Cursor。
2. 通过PullLogs接口消费该时间段内数据。
3. 在内存中进行数据统计与计算，拿到结果，生成计费数据。

   我们可以以此类推，把选择时间计算逻辑修改为1分钟，10秒钟等。



性能分析：

- 假设有10亿条/天计量日志，每条长度为200字节，数据量为200GB。
- LogHub 默认SDK或Agent都带压缩功能，实际存储数据量为40GB（一般至少有5倍压缩率），一个小时数据量为40/24 = 1.6GB。
- LogHub读取接口一次最大支持读1000个包（每个包最大为5MB），在千兆网条件下2秒内即可读完。
- 加上内存中数据累计与计算时间，对1小时计量日志进行汇总，不超过5秒。

数据量大应如何解决

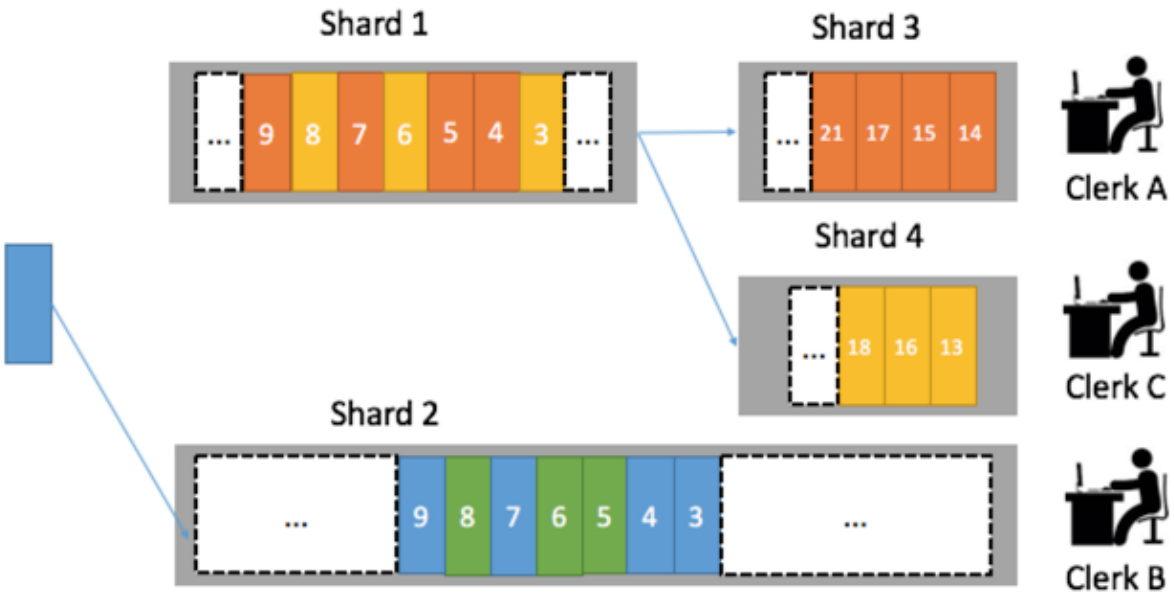
在一些计费场景下（例如运营商、IoT等）计量日志量会很大（例如十万亿，数据量为2PB/Day），折算压缩数据后一小时有16 TB，以万兆网络读取需要1600秒，已不能满足快速出账单需求。

1. 控制产生的计费数据量

我们对于产生计量日志程序进行改造（例如Nginx），先在内存中做了聚合，每隔1分钟Dump一次该时间段聚合的汇总计量日志结果。这样数据量就和总体的用户数相关了：假设Nginx该时间段内有1000个用户，一个小时数据点也才1000 * 200 * 60 = 12 GB（压缩后为 240 MB）。

2. 将计量日志处理并行化

LogHub下每个日志库可以分配不同Shard（分区），我们可以分配3个分区，3个计量消费程序。为了保证一个用户计量数据总是由一个消费程序处理，我们可以根据用户ID Hash到固定Shard中。例如杭州市西湖区用户写在1号Shard，杭州上城区用户数据写在2号Shard，这样后台计量程序就可水平扩展。



其他问题

- 补数据怎么办?

  LogHub 下每个Logstore可以设置生命周期（1-365天），如果计费程序需要重新消费数据，在生命周期内可以任意根据时间段进行计算。

- 计量日志散落在很多服务器（前端机）怎么办

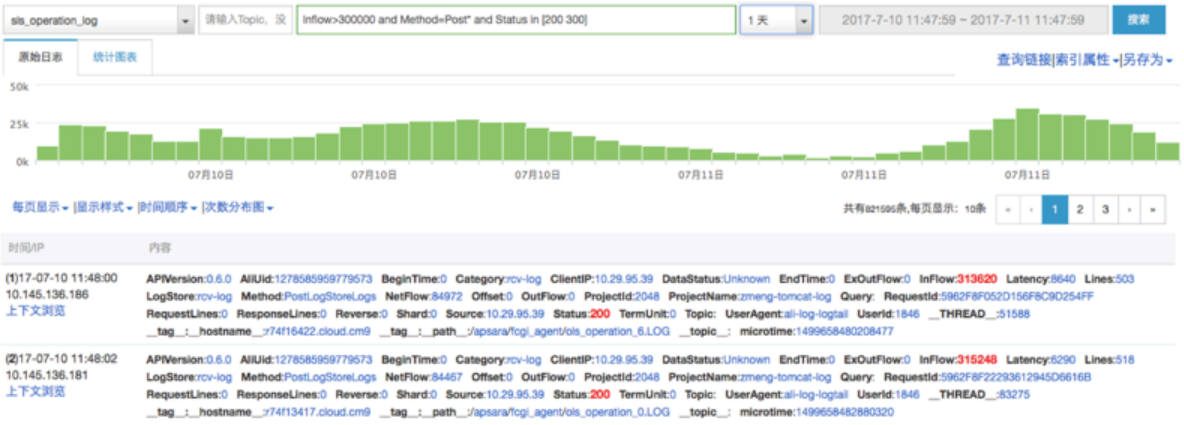
  1. 使用Logtail Agent实时采集
  2. 使用机器标示定义一套动态机器组弹性伸缩

- 查询明细需求如何满足

  对LogHub中数据可以创建索引，支持实时查询与统计分析，例如我们想调查有一些特别大的计量日志：

```
Inflow>300000 and Method=Post* and Status in [200 300]
```

  在对Loghub中数据打开索引后，即可实时查询与分析。



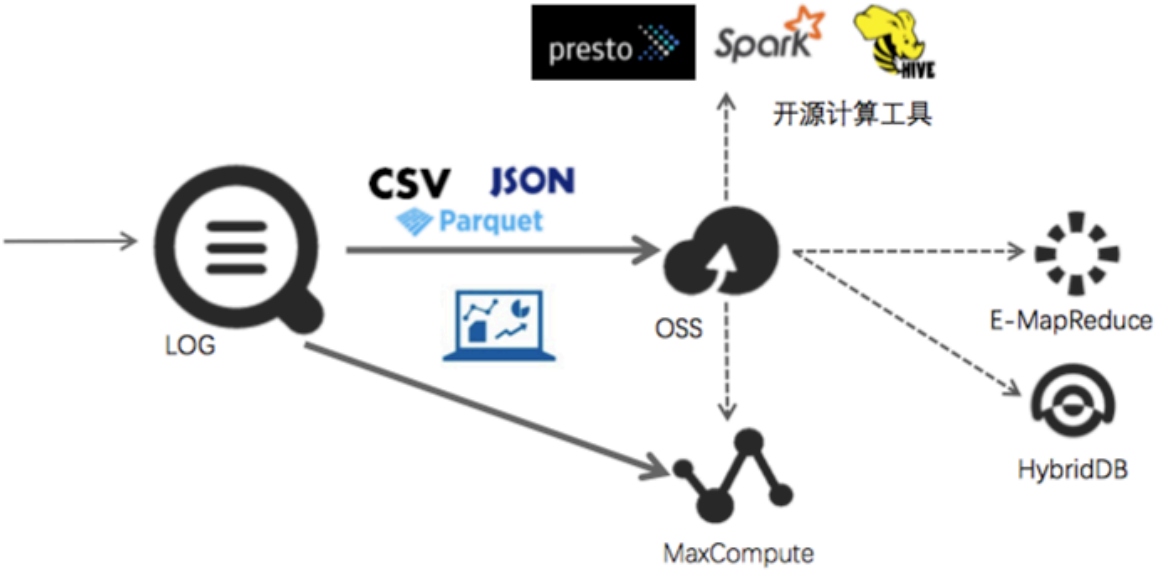
  也可以在查询后加上统计分析：

```
Inflow>300000 and Method=Post* and Status in [200 300] | select max(
Inflow) as s, ProjectName group by ProjectName order by s desc
```

ProjectName	s
rel-stat-staff-benefits	1207132
rel-acticle-user-action	816968
tteduhaproxy	688385
noc	583006
xiaobinggd	539798
ludashi-stat	534489
sls-weloop	526956
ykf-testalipay	524523
fc-monitor-ol	524515
syt-accesslog	486647

- 存储日志并进行T+1对账

  日志服务提供LogHub中数据投递功能，支持自定义分区、自定义存储格式等将日志存储在OSS/MaxCompute上，利用E-MapReduce、MaxCompute、HybridDB、Hadoop、Hive、Presto、Spark等进行计算。



4.4 消费-通过Consumer Library实现高可靠消费

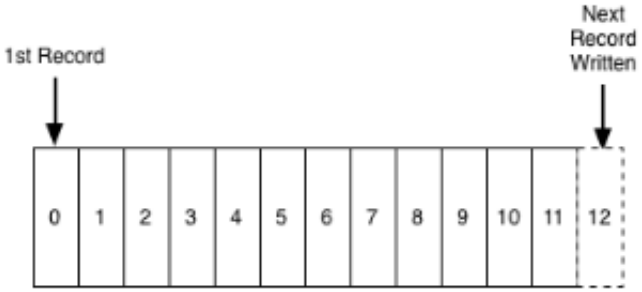
日志处理是一个很大范畴，其中包括实时计算、数据仓库、离线计算等众多点。这篇文章主要介绍在实时计算场景中，如何能做到日志处理保序、不丢失、不重复，并且在上下游业务系统不可靠（存在故障）、业务流量剧烈波动情况下，如何保持这三点。

为方便理解，本文使用《银行的一天》作为例子将概念解释清楚。在文档末尾，介绍日志服务LogHub功能，如何与Spark Streaming、Storm Spout等配合，完成日志数据的处理过程。

问题定义

什么是日志数据?

原LinkedIn员工Jay Kreps在《The Log: What every software engineer should know about real-time data's unifying abstraction》描述中提到：“append-only, totally-ordered sequence of records ordered by time”。



- Append Only：日志是一种追加模式，一旦产生过后就无法修改。
- Totally Ordered By Time：严格有序，每条日志有一个确定时间点。不同日志在秒级时间维度上可能有重复，比如有2个操作GET、SET发生在同一秒钟，但对于计算机而言这两个操作也是有顺序的。

什么样的数据可以抽象成日志?

半世纪前说起日志，想到的是船长、操作员手里厚厚的笔记。如今计算机诞生使得日志产生与消费无处不在：服务器、路由器、传感器、GPS、订单、及各种设备通过不同角度描述着我们生活的世界。从船长日志中我们可以发现，日志除了带一个记录的时间戳外，可以包含几乎任意的内容，例如：一段记录文字、一张图片、天气状况、船行方向等。半个世纪过去了，“船长日志”的方式已经扩展到一笔订单、一项付款记录、一次用户访问、一次数据库操作等多样的领域。

在计算机世界中，常用的日志有：Metric，Binlog（Database、NoSQL），Event，Auditing，Access Log 等。

在我们今天的演示例子中，我们把用户到银行的一次操作作为一条日志数据。其中包括用户、账号名、操作时间、操作类型、操作金额等。

例如：

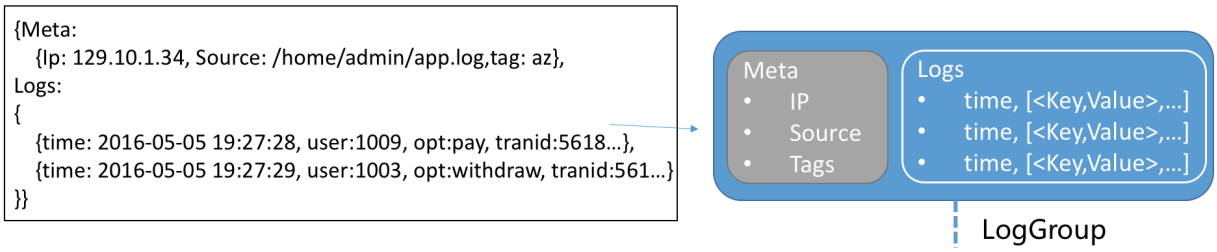
```
2016-06-28 08:00:00 张三 存款 1000元
2016-06-27 09:00:00 李四 取款 20000元
```

LogHub数据模型

为了能抽象问题，这里以阿里云日志服务下LogHub作为演示模型，详细可以参见日志服务下基本概念 。

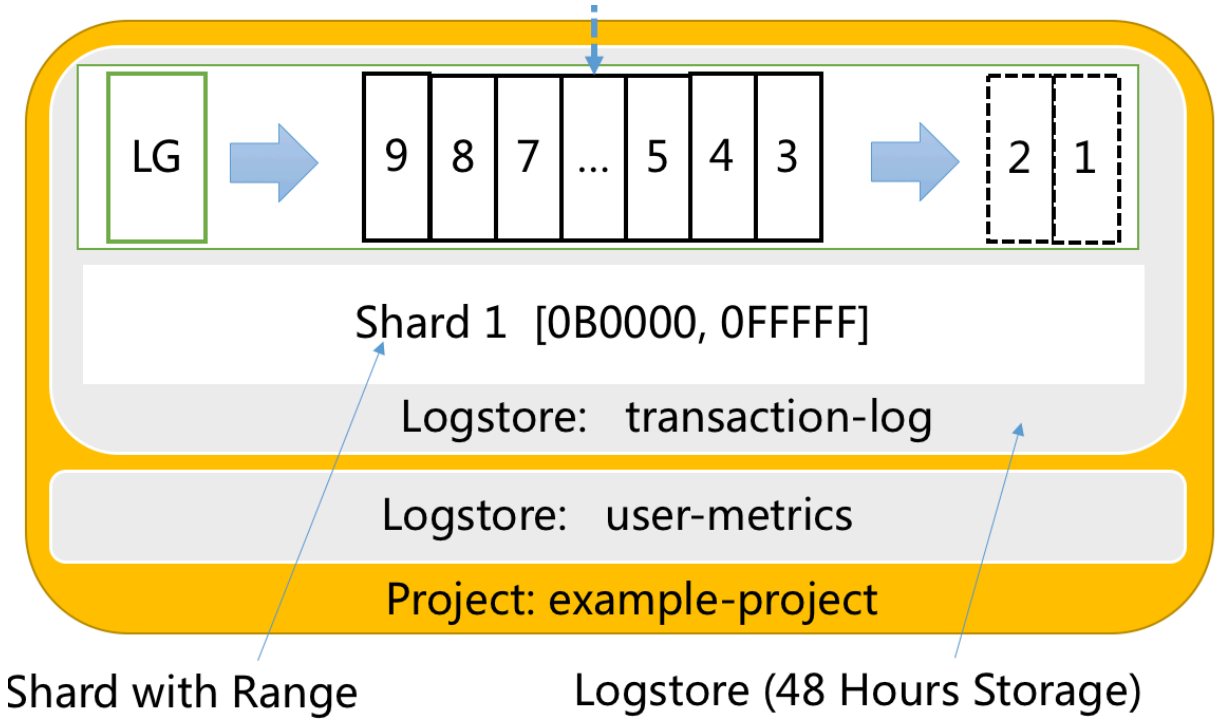
- Log：由时间、及一组Key-Value对组成。
- LogGroup：一组日志的集合，包含相同Meta（IP，Source）等。

两者关系如下：



- Shard：分区，LogGroup读写基本单元，可以理解为48小时为周期的FIFO队列。每个Shard提供 5 MB/S Write, 10 MB/S Read能力。Shard 有逻辑区间（BeginKey，EndKey）用以归纳不同类型数据。
- Logstore：日志库，用以存放同一类日志数据。Logstore是一个载体，通过由[0000, FFFF ..)区间Shard组合构建而成，Logstore会包含1个或多个Shard。
- Project：Logstore存放容器。

这些概念相互关系如下：



银行的一天

以19世纪银行为例。某个城市有若干用户（Producer），到银行去存取钱（User Operation），银行有若干个柜员（Consumer）。因为19世纪还没有电脑可以实时同步，因此每个柜员都有一个小账本能够记录对应信息，每天晚上把钱和账本拿到公司去对账。

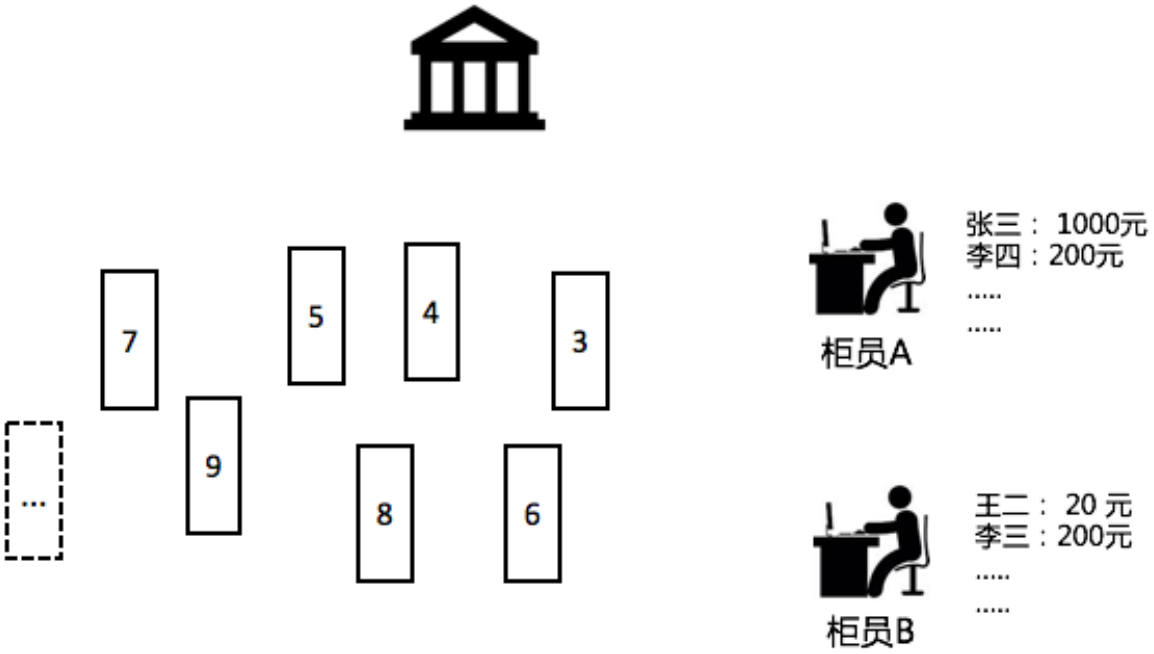
在分布式世界里，我们可以把柜员认为是固定内存和计算能力单机。用户是来自各个数据源的请求，Bank大厅是处理用户存取数据的日志库（Logstore）。

- Log/LogGroup：用户发出的存取款等操作。
- 用户（User）：Log/LogGroup生产者。
- 柜员（Clerk）：银行处理用户请求的员工。
- 银行大厅（Logstore）：用户产生的操作请求先进入银行大厅，再交给柜员处理。
- 分区（Shard）：银行大厅用以安排用户请求的组织方式。

问题1：保序（Ordering）

银行有2个柜员（A，B），张三进了银行，在柜台A上存了1000元，A把张三1000元存在自己的账本上。张三到了下午觉得手头紧到B柜台取钱，B柜员一看账本，发现不对，张三并没有在这里存钱。

从这个例子可以看到，存取款是一个严格有序的操作，需要同一个柜员（处理器）来处理同一个用户的操作，这样才能保持状态一致性。

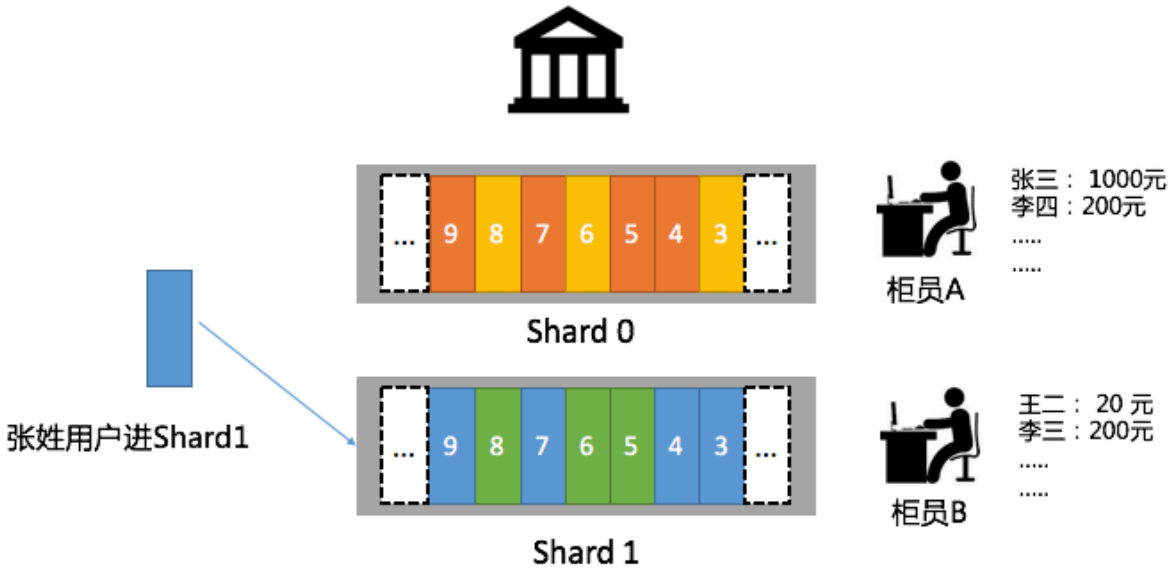


实现保序的方法很简单：排队，创建一个Shard，终端只有一个柜员A来处理。用户请求先进先出，一点问题都没有。但带来的问题是效率低下，假设有1000个用户来进行操作，即使有10个柜员也无济于事。

这种场景怎么办?

1. 假设有10个柜员，我们可以创建10个Shard。
2. 如何保证对于同一个账户的操作是有序的？可以根据一致性Hash方式将用户进行映射。例如我们开10个队伍（Shard），每个柜员处理一个Shard，把不同银行账号或用户姓名，映射到特定Shard中。在这种情况下张三 Hash（Zhang）= Z 永远落在一个特定Shard中（区间包含Z），处理端面对的永远是柜员A。

当然如果张姓用户比较多，也可以换其他策略。例如根据用户AccountID、ZipCode进行Hash，这样就可以使得每个Shard中操作请求更均匀。



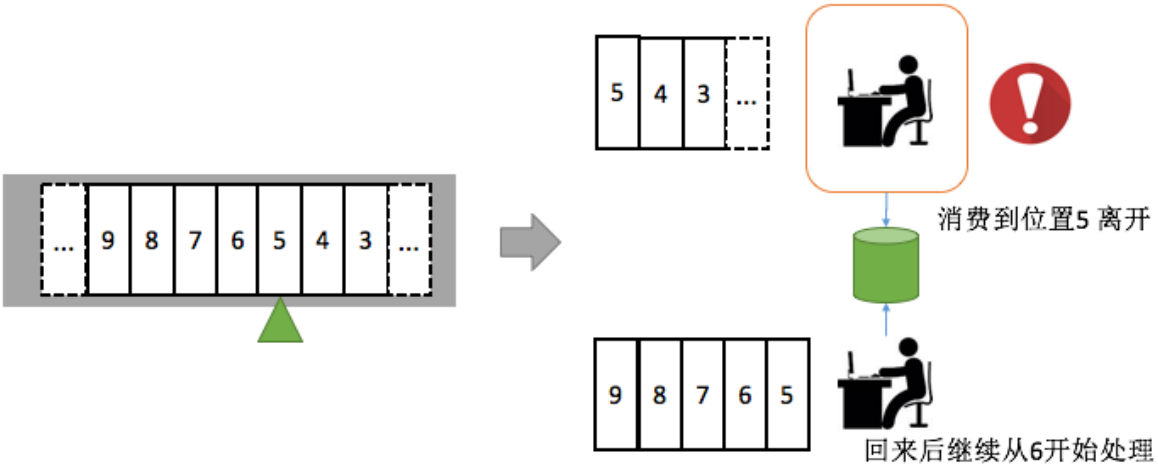
问题2：不丢失（At-Least Once）

张三拿着存款在柜台A处理，柜员A处理到一半去接了个电话，等回来后以为业务已经办理好了，于是开始处理下一个用户的请求，张三的存款请求因此被丢失。

虽然机器不会人为犯错，在线时间和可靠性要比柜员高。但难免也会遇到电脑故障、或因负载高导致的处理中断，因为这样的场景丢失用户的存款，这是万万不行的。

这种情况怎么办呢?

A可以在自己日记本上（非账本）记录一个项目：当前已处理到Shard哪个位置，只有当张三的这个存款请求被完全确认后，柜员A才能叫下一个。



带来问题是什么？可能会重复。比如A已经处理完张三请求（更新账本），准备在日记本上记录处理到哪个位置之时，突然被叫开了，当他回来后，发现张三请求没有记录下来，他会把张三请求再次处理一遍，这就会造成重复。

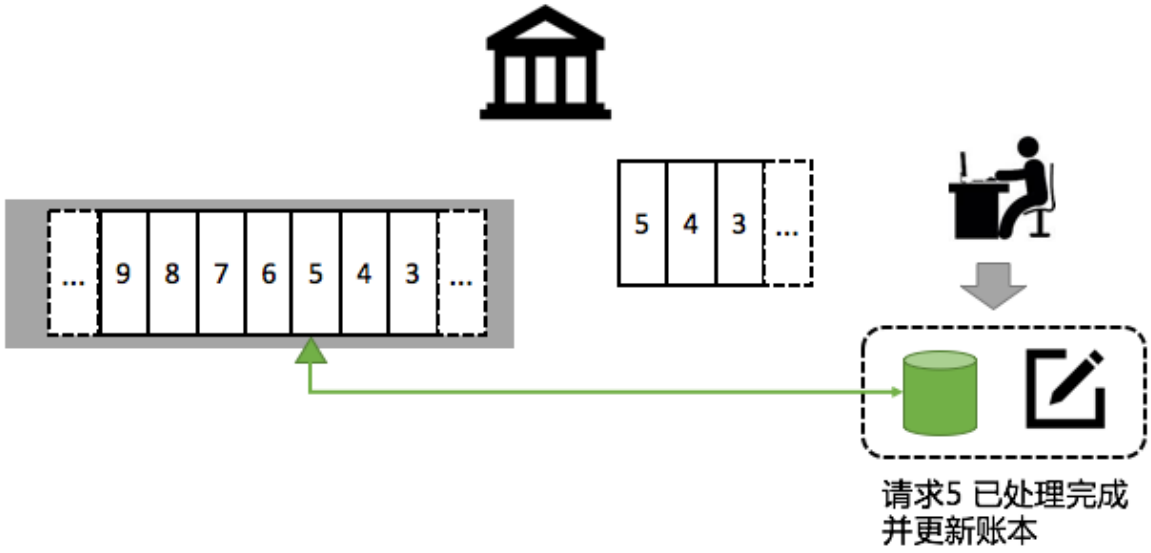
问题3：不重复（Exactly Once）

重复一定会带来问题吗？不一定。

在幂等情况下，重复虽然会有浪费，但对结果没有影响。什么叫幂等：重复消费不对结果产生影响的操作叫做幂等。例如用户有一个操作 “查询余额”，该操作是一个只读操作，重复做不影响结果。对于非只读操作，例如注销用户这类操作，可以连续做两次。

但现实生活中大部分操作不是幂等的，例如存款、取款等，重复进行计算会对结果带来致命的影响。解决的方式是什么呢？柜员（A）需要把账本完成 + 日记本标记Shard中处理完成作为一个事物合并操作，并记录下来（CheckPoint）。

如果A暂时离开或永久离开，其他柜员只要使用相同的规范：记录中已操作则处理下一个即可，如果没有则重复做，过程中需要保证原子性。



CheckPoint可以将Shard 中的元素位置（或时间）作为Key，放入一个可以持久化的对象中。代表当前元素已经被处理完成。

业务挑战

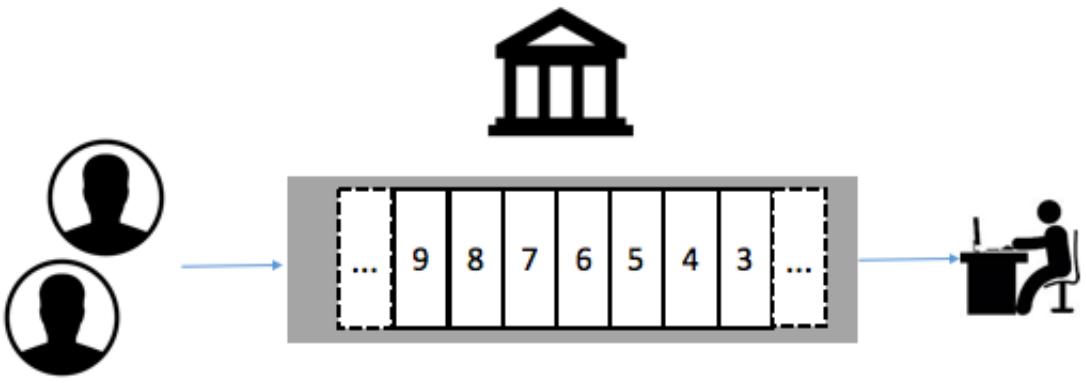
以上三个概念解释完成后，原理并不复杂。但在现实世界中，规模的变化与不确定性会使得以上三个问题便得更复杂。例如：

1. 遇到发工资日子，用户数会大涨。
2. 柜员（Clerk）毕竟不是机器人，他们需要休假，需要吃午饭。
3. 银行经理为了整体服务体验，需要加快柜员，以什么作为判断标准？Shard中处理速度？
4. 柜员在交接过程中，能否非常容易地传递账本与记录？

现实中的一天

8点银行开门

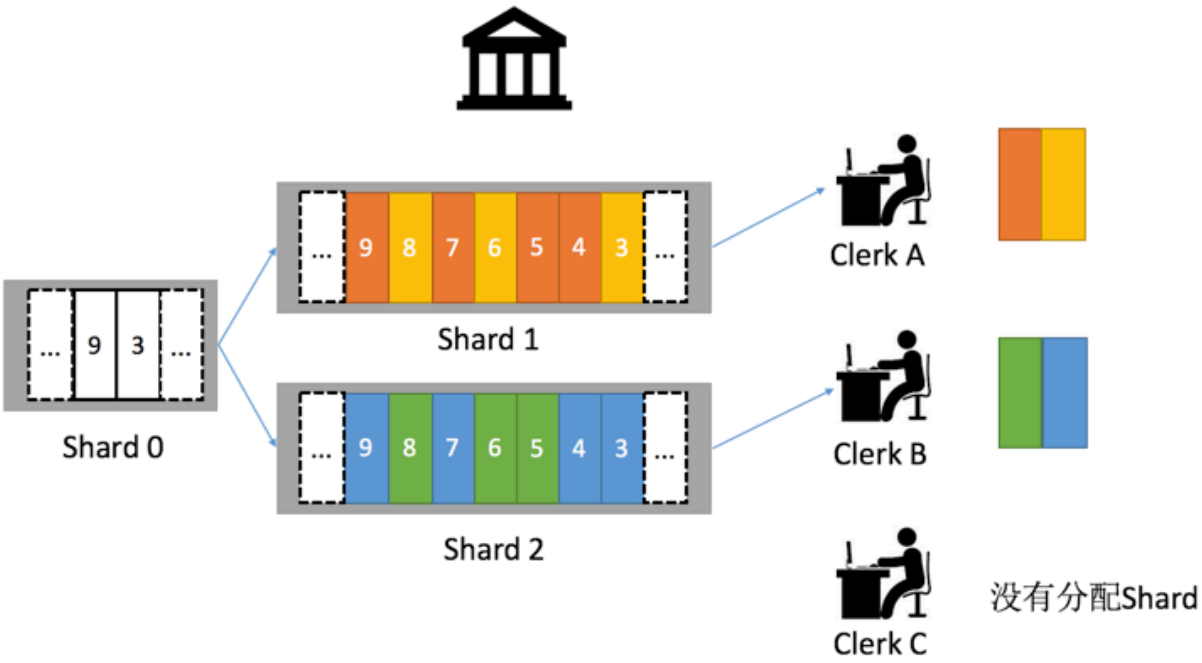
只有一个Shard0，用户请求全部排在Shard0下，柜员A也正好可以处理。



10点进入高峰期间

银行经理决定把10点后Shard0分裂成2个新Shard（Shard1，Shard2），并且给了如下规定，姓名是[A-W]用户到Shard1中排队，姓名是[X, Y, Z] 到Shard 2 中排队等待处理，为什么这两个Shard区间不均匀？因为用户的姓氏本身就是不均匀的，通过这种映射方式可以保证柜员处理的均衡。

10-12点请求消费状态：

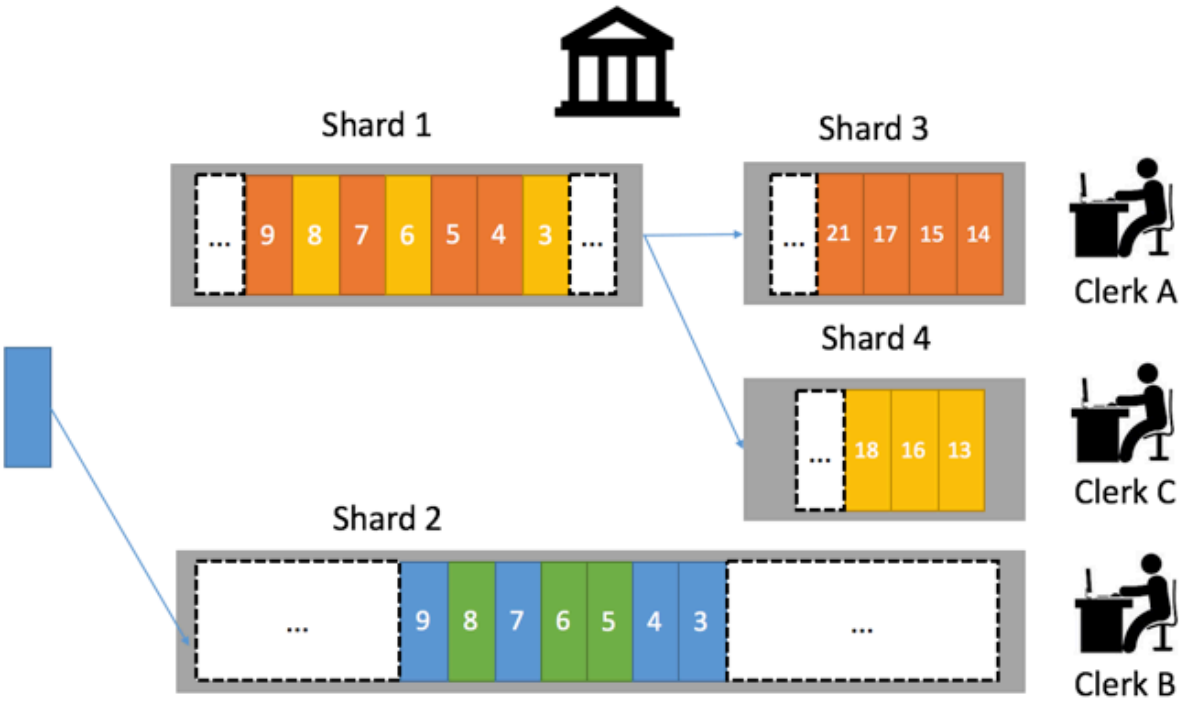


柜员A处理2个Shard非常吃力，于是经理派出柜员B、C出厂。因为只有2个Shard，B开始接管A负责一个Shard，C处于闲置状态。

中午12点人越来越多

银行经理觉得Shard1下柜员A压力太大，因此从Shard1中拆分出（Shard3，Shard4）两个新的Shard，Shard3由柜员A处理、Shard4由柜员C处理。在12点后原来排在Shard 1中的请求，分别到Shard3，Shard4中。

12点后请求消费状态：



流量持续到下午4点后，开始逐渐减少

因此银行经理让柜员A、B休息，让C同事处理Shard2，Shard3，Shard4中的请求。并逐步将Shard2与Shard3合并成Shard5，最后将Shard5和Shard4合并成一个Shard，当处理完成Shard中所有请求后银行关门。

现实中的日志处理

上述过程可以抽象成日志处理的经典场景，如果要解决银行的业务需求，我们要提供弹性伸缩、并且灵活适配的日志基础框架，包括：

1. 对Shard进行弹性伸缩。
2. 消费者上线与下线能够对Shard自动适配，过程中数据不丢失。过程中支持保序。
3. 过程中不重复（需要消费者配合）。
4. 观察到消费进度，以便合理调配计算资源。
5. 支持更多渠道日志接入（对银行而言开通网上银行、手机银行、支票等渠道，可以接入更多的用户请求）。

通过LogHub + LogHub Consumer Library 能够帮助您解决日志实时处理中的这些经典问题，只需把精力放在业务逻辑上，而不用去担心流量扩容、Failover等琐事。

另外，Storm、Spark Streaming已经通过Consumer Library实现了对应的接口，欢迎使用。日志服务的更多技术资讯和讨论，请查看日志服务的主页和日志处理圈子。

4.5 消费-通过ETL清洗数据

日志处理过程中的一个假设是：数据并不是完美的。在原始数据与最终结果之间有 Gap，需要通过 ETL（Extract Transformation Load）等手段进行清洗、转换与整理。

案例

“我要点外卖”是一个平台型电商网站，涉及用户、餐厅、配送员等。用户可以在网页、App、微信、支付宝等进行下单点菜；商家拿到订单后开始加工，并自动通知周围的外卖送货员；快递员将外卖送到用户手中。

运营小组有两个任务：

- 掌握外卖送货员的位置，定点调度。
- 掌握优惠券、现金的使用情况，定点投放优惠券进行互动运营。

送货员位置信息（GPS）数据加工

GPS 数据（X，Y）通过送货员 App 每分钟汇报一次，格式如下：

```
2016-06-26 19:00:15 ID:10015 DeviceID:EXX12345678 Network:4G GPS-X:10.
30.339 GPS-Y:17.38.224.5 Status:Delivering
```

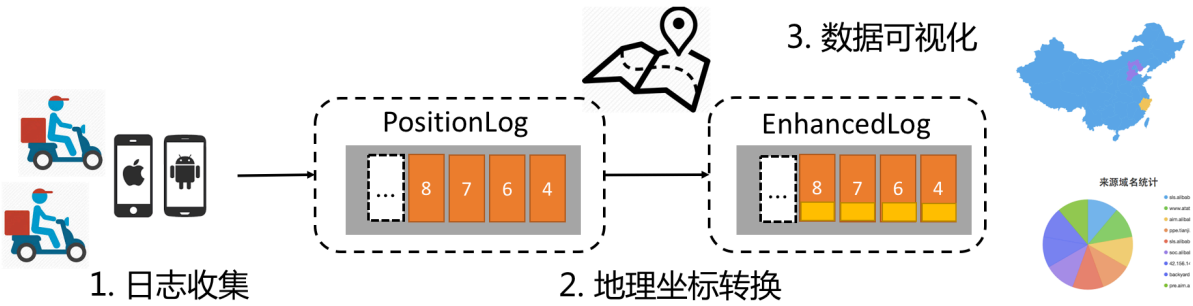
其中记录了上报时间，送货员 ID，使用网络，设备号，坐标位置 GPS-X，GPS-Y。GPS 给出的经纬度非常准确，但运营小组需要统计每个区域当前的状态，并不需要这么细的数据。因此，需要对原始数据做一次转换（Transformation），将坐标转换为可读的城市、区域、邮政编码等字段。

```
(GPS-X,GPS-Y) --> (GPS-X, GPS-Y, City, District, ZipCode)
```

这就是一个典型的 ETL 需求。使用 loghub 功能创建两个 logstore（PositionLog），以及转换后 logstore（EnhancedLog）。通过运行 ETL 程序（例如 Spark Streaming、Storm、或容器中启动 Consumer Library），实时订阅 PositionLog，对坐标进行转化，写入 EnhancedLog。可以对 EnhancedLog 数据机型仓库，连接实时计算进行可视化，或建立索引进行查询。

整个过程推荐架构如下：

1. 快递员 App 上埋点，每分钟汇报当前 GPS 位置一次，写入第一个 logstore（PositionLog）
   - 推荐使用 LogHub Android/IOS SDK、MAN（移动数据分析）将移动设备日志接入。
2. 通过实时程序实时订阅 PositionLog 数据，处理后实时写入 EnhancedLog Logstore。
   - 推荐使用 Spark Streaming、Storm 或 Consumer Lib（一种自动负载均衡的编程模式）或 SDK 订阅。
3. 对 Enahanced Log 进行处理，例如计算后进行数据可视化。推荐：
   - LogHub 对接流计算
   - LogShipper 投递（OSS，E-MapReduce，Table Store、MaxCompute）
   - LogSearch：订单查询等



支付订单脱敏与分析

支付服务接受支付请求，包括支付的账号、方式、金额、优惠券等。

- 其中包含部分敏感信息，需要进行脱敏。
- 需要对支付信息中优惠券、现金两个部分进行剥离。

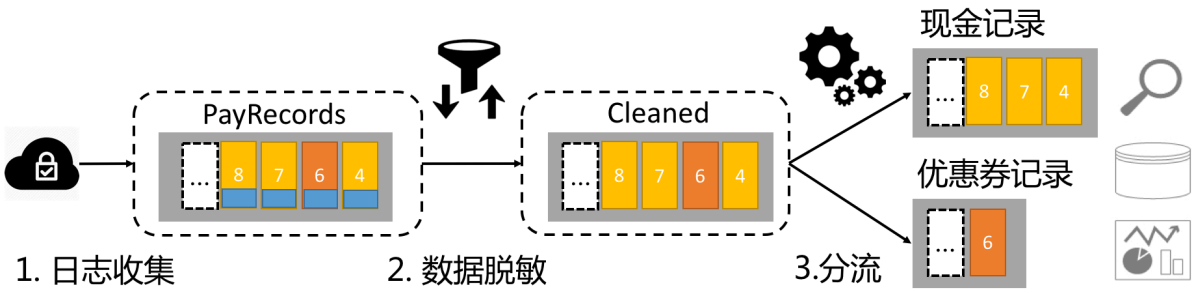
整个过程如下：

1. 创建 4 个 logstore，原始数据（PayRecords），脱敏后数据（Cleaned），现金订单（Cash），代金券（Coupon）。应用程序通过 Log4J Appender 向原数据 logstore（PayRecords）写入订单数据。

   推荐使用 Log4J Appender，敏感数据不落盘。
2. 脱敏程序实时消费 PayRecords，将账号相关信息剥离后，写入 Cleaned Logstore。

分流程序实时消费 Cleaned Logstore，通过业务逻辑把优惠券、现金两个部分分别存入对账相关的 logstore（Cash、Coupon）进行后续处理。

实时脱敏、分流程序推荐使用 Spark Streaming、Storm 或 Consumer Lib（一种自动负载均衡的编程模式） 或 SDK 订阅。



其他

- loghub 功能下每个 logstore 可以通过 RAM 进行账号级权限控制。
- loghub 当前读写可以通过监控获得，消费进度可以通过控制台查看。

5 投递

5.1 投递-对接数据仓库

日志服务LogShipper功能可以便捷地将日志数据投递到 OSS、Table Store、MaxCompute 等存储类服务，配合 E-MapReduce（Spark、Hive）、MaxCompute 进行离线计算。

数仓（离线计算）

数据仓库+离线计算是实时计算的补充，两者针对目标不同：

模式	优势	劣势	使用领域
实时计算	快速	计算较为简单	增量为主，监控、实时分析
离线计算（数据仓库）	精准、计算能力强	较慢	全量为主，BI、数据统计、比较

目前对于数据分析类需求，同一份数据会同时做实时计算+数据仓库（离线计算）。例如对访问日志：

- 通过流计算实时显示大盘数据：当前PV、UV、各运营商信息。
- 每天晚上对全量数据进行细节分析，比较增长量、同步/环比，Top数据等。

互联网领域有两种经典的模式讨论：

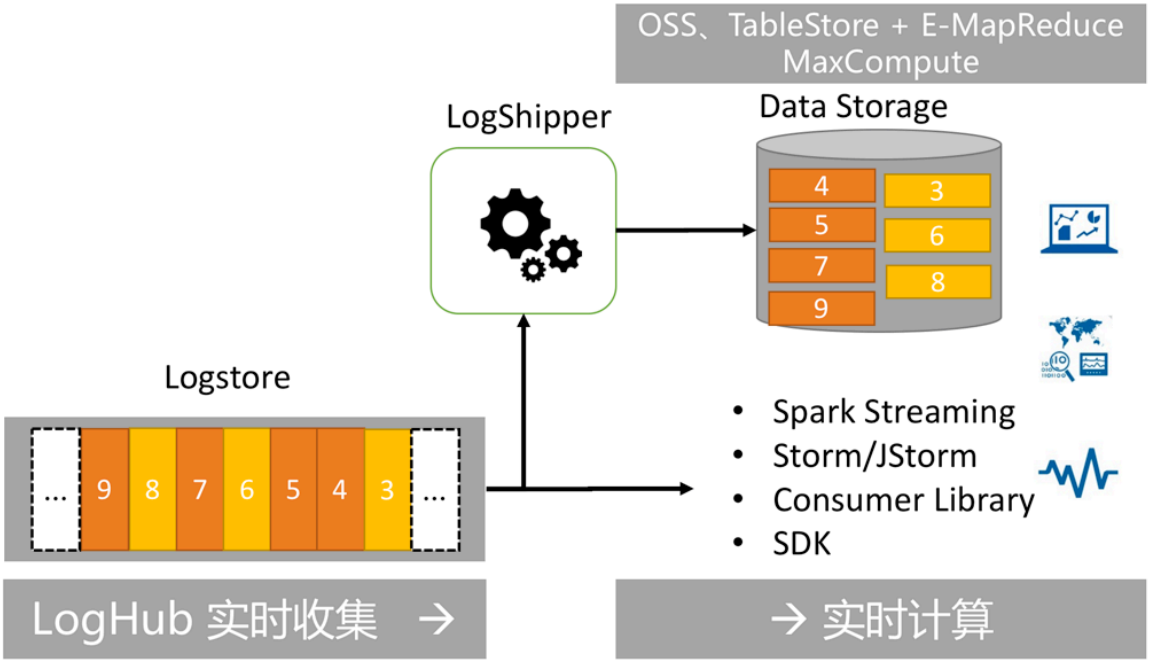
- Lamdba Architecture：数据进来后，既支持流式处理、同时存入数仓。但用户发起查询时，会根据查询需求和复杂度从实时计算、离线计算拿结果返回。
- Kappa Architecture：kafka based Architecture。弱化离线计算部分，数据存储都在Kafka中，实时计算解决所有问题。

日志服务提供模式比较偏向Lamdba Architecture。

LogHub/LogShipper一站式解决实时+离线场景

在创建Logstore后，可以在控制台配置LogShipper支持数据仓库对接。当前支持如下：

- OSS（大规模对象存储）：
- TableStore（NoSQL数据存储服务）：
- MaxCompute（大数据计算服务）：



LogShipper提供如下功能：

- 准实时：分钟级进入数据仓库
- 数据量大：无需担心并发量
- 自动重试：遇到故障自动重试、也可以通过API手动重试
- 任务API：通过API可以获得时间段日志投递状态
- 自动压缩：支持数据压缩、节省存储带宽

典型场景

场景 1：日志审计

小A维护了一个论坛，需要对论坛所有访问日志进行审计和离线分析。

- G部门需要小A配合记录最近180天内用户访问情况，在有需求时，提供某个时间段的访问日志。
- 运营同学在每个季度需要对日志出一份访问报表。

小A使用日志服务（LOG）收集服务器上日志数据，并且打开了日志投递（LogShipper）功能，日志服务就会自动完成日志收集、投递、以及压缩。有审查需要时，可以将该时间段日志授权给第三方。需要离线分析时，利用E-MapReduce跑一个30分钟离线任务，用最少的成本办了两件事情。也可以使用阿里云DLA对投递到OSS中的日志数据进行数据分析。

场景 2：日志实时+离线分析

小B是一个开源软件爱好者，喜欢利用Spark进行数据分析。需求如下：

- 移动端通过API收集日志。
- 通过Spark Streaming对日志进行实时分析，统计线上用户访问。
- 通过Hive进行T+1离线分析。
- 将日志数据开放给下游代理商，进行其他维度分析。

通过今天LOG+OSS+EMR/DLA+RAM组合，可轻松应对这类需求。

6 服务日志

6.1 开通、监控和消费服务日志

日志服务提供服务日志功能，帮助您实时掌握日志服务的使用状况、提高运维效率。

开通服务日志

服务日志分为详细日志和重要日志（包括Logtail相关日志，消费组延时日志和计量日志等），分别保存在internal-operation_log和internal-diagnostic_log中。internal-diagnostic_log不计费，internal-operation_log和普通Logstore一样计费。详细日志对应用户的每次操作或者API请求，当读写请求较多时会产生较多的操作日志。

用户可以根据需要开通服务日志：

日志存储位置建议使用推荐配置，将同一个地域的服务日志保存到相同的Project中，便于管理和统计。

监控Logtail心跳

Logtail 安装好之后，可以通过服务日志Logtail的状态日志，来检查Logtail的工作状态。

Logtail状态日志保存在internal-diagnostic_log 中，进入internal-diagnostic_log搜索：`__topic__: logtail_status`，即可搜索Logtail状态日志。我们可以统计最近一段时间内的机器个数，用于和Logtail应用机器组的总机器数比较，并配置告警规则，如果低于总机器数则触发告警。

- **查询语句**

```
__topic__: logtail_status | SELECT COUNT(DISTINCT ip) as ip_count
```

- **查询截图**

- 配置告警规则（此处假设总机器数为100）：

如果产生告警，则用户可以前往控制台查看机器组的状态，检查哪些机器心跳异常。

查看存储空间等计量数据

用户将日志写入日志服务后，通过日志流量，读写次数，存储空间等计量信息，可以看到日志服务的使用状况和各个计费项的详细内容。

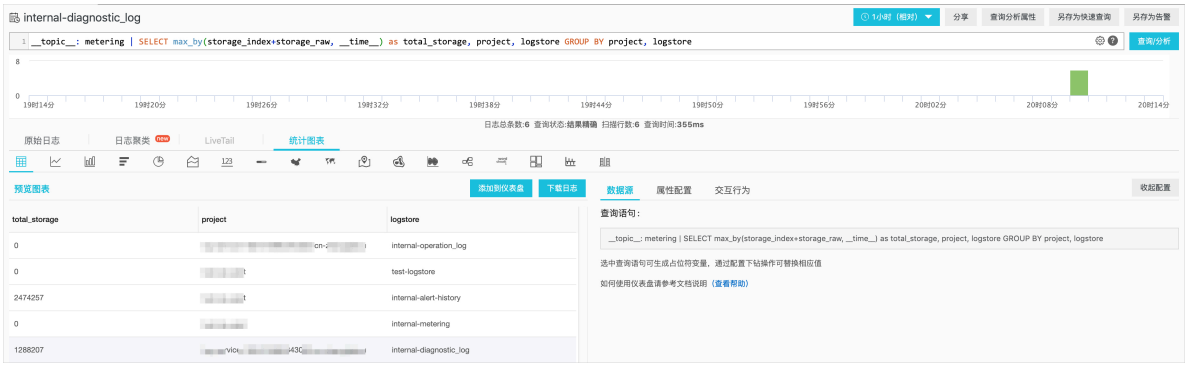
每个Logstore每小时产生一条计量日志，包含统计时间段内的读写流量，读写次数以及当前时间点的原始日志和索引的存储空间大小，各个字段解释可以参考计量日志字段解释。日志服务的计量日志保存在服务日志的internal-diagnostic_log中，通过搜索：`__topic__: metering`，即可搜索计量日志。

使用max_by函数计算各个Logstore的存储空间：

- 查询语句

```
__topic__: metering | SELECT max_by(storage_index+storage_raw,
__time__) as total_storage, project, logstore GROUP BY project,
logstore
```

- 查询截图



服务日志的默认仪表盘提供了丰富的计量日志相关的图表，参考计量日志报表。

查看消费组延时

日志写入日志服务后，除了普通的查询分析之外，可能还需要对日志进行消费。日志服务提供了多种语言实现的消费组协同消费库。

使用消费组消费时，日志的消费延时是用户比较关心的问题。通过观察消费延时，我们可以知道当前的消费进度，当延时较大时我们可以通过调整消费者个数等方式来改进消费速度。

消费组延时日志作为服务日志的一种，同样保存在internal-diagnostic_log中，每2分钟生成一次。通过搜索：`__topic__: consumergroup_log`，就可以搜索所有的消费组延时日志。

查询test-consumer-group这个消费组的消费延时：

- 查询语句

```
__topic__: consumergroup_log and consumer_group:  test-consumer-
group | SELECT max_by(fallbehind, __time__) as fallbehind
```

- 查询截图：

Logtail异常监控

Logtail是否正常运行关系到日志的完整性，如果用户能够及时发现Logtail的异常状况，就能尽快调整Logtail配置，使日志不会丢失。

Logtail的异常日志可以通过如下方式搜索：`__topic__: logtail_alarm`。

查询15分钟各种异常类型的次数：

- 查询语句

```
__topic__: logtail_alarm | select sum(alarm_count)as errorCount,
alarm_type  GROUP BY alarm_type
```

- 查询截图

Logstore写入流量监控

通过计量日志，我们可以统计一个小时内的读写流量，但是如果需要统计更精确的时间范围，如15分钟内的读写流量，则需要使用操作日志。用户的每次操作都对应一次请求，每次请求都会产生一条操作日志。以15分钟为例，我们只需要统计所有写入操作的流量总和即可。

1. 统计15分钟内Logstore写入的原始日志流量和压缩流量：

- 查询语句

```
Method: PostLogStoreLogs AND Project: my-project and LogStore: my
-logstore | SELECT sum(InFlow) as raw_bytes, sum(NetInflow) as
network_bytes
```

- 查询截图

2. 查询15分钟内写入网络流量的下降百分比：

- 查询语句

```
Method: PostLogStoreLogs AND Project: my-project and LogStore:
my-logstore | select round((diff[1]-diff[2])/diff[1],2) as rate
from (select compare(network_bytes, 900) as diff from (select sum(
NetInflow) as network_bytes from log))
```

- 查询截图

3. 创建告警：

设置大于50%时告警

操作审计

Project下所有资源的操作日志保存在internal-operation_log中，每行日志不仅会记录操作相关的信息，如创建机器组会记录机器组名称，操作Logstore会记录Logstore名称等，还会记录对应操作的用户信息。目前，用户信息可以分为如下类型：

类型	字段
主账户	· InvokerUid：主账户Aliuid。 · CallerType：Parent。
子账户	· InvokerUid：子账户Aliuid。 · CallerType：Subuser。

类型	字段
Sts	· InvokerUid：主账户Aliuid。 · CallerType：Sts。 · RoleSessionName：<session名称>。

通过上述信息就能知道不同操作日志对应的用户信息。

7 告警

7.1 告警设置

日志服务支持根据仪表盘中的查询图表设置告警，实现实时的服务状态监控。

告警的查询区间和执行间隔

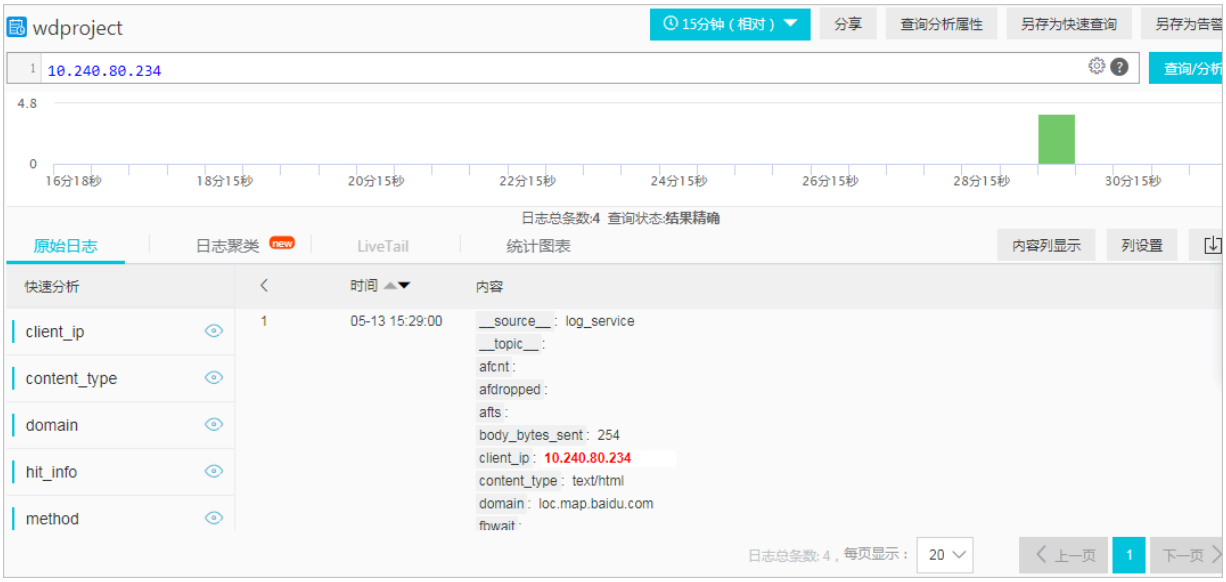
告警的实现原理是基于告警的查询范围，根据执行间隔定时执行配置的查询语句，并将查询结果作为告警条件的参数进行计算，如果计算结果为true，则告警触发。

不要将查询范围设置成和执行间隔一致的相对时间，如查询范围为相对1分钟，执行间隔为1分钟。原因如下（以执行间隔为1分钟为例）：

- 数据写入日志服务到能够被查询到中间存在延时，即便延时很低，也存在数据漏查的风险。如告警执行时间为12:03:30， 查询范围为相对一分钟则为[12:02:30，12:03:30)，对于12:03:29秒写入的日志，不能保证12:03:30这次时间点能够查询到。
  - 如果对告警的准确性要求高（不重复报警，不漏报），查询范围起止时间可以往前推移，如70秒前—10秒前。如告警执行时间为12:03:30，则查询范围为 [12:02:20, 12:03:20)，通过设置10秒的缓冲时间来避免因为索引速度导致的漏查。
  - 如果对实时性要求高（第一时间收到告警，能够容忍重复报警），查询范围开始时间可以往前推移，如 70秒—现在。如告警执行时间为12:03:30， 查询范围设置为相对70秒， [12:02:20, 12:03:30)。
- 对于写入包含同一分钟不同时间的日志时，由于日志服务的索引构建方式，可能会存在较晚的日志的索引落入较早的日志的时间点的可能。 如告警执行时间为12:03:30， 查询范围为相对一分钟则为[12:02:30，12:03:30)，如果在12:02:50秒写入多条日志，这些日志的时间有12:02:20，12:02:50等，那么这一批日志的索引可能会落入12:02:20 这个时间点，导致使用时间范围 [12:02:30，12:03:30) 查询不到。
  - 如果对告警的准确性要求高（不重复报警，不漏报），查询范围使用整点分钟，如整点1分钟，整点5分钟，整点1小时等，并且将执行间隔设置成一致的时间，如1分钟，5分钟，1小时等。
  - 如果对实时性要求高（第一时间收到告警，能够容忍重复报警），查询时间范围至少需要包含前一分钟。如告警执行时间为12:03:30， 查询范围可以设置为相对90秒，那么实际的查询范围为 12:02:00 - 12:03:30，同时设置执行间隔为1分钟。

基于查询结果告警

如果针对某个查询，只要查询结果不为空就认为满足告警条件，可以设置告警条件为判断任意字段存在即告警。如搜索包含IP 10.240.80.234 的日志：



只要查询到包含 10.240.80.234 的日志就告警，则可以通过任意字段设置一个始终为true的告警条件。假设 client_ip 这个字段在每条日志都存在且不可能为空字符串，则只要 client_ip 这个字段不为空就触发告警：

基于分析结果告警

基于分析结果设置告警是最常见的场景，比如针对特定的字段聚合之后告警。以最常见的包含ERROR 关键字的日志条数达到阈值即触发告警为例，查询语句可以按照如下的方式设置：

```
ERROR | select count(1) as errorCount
```

告警条件则为 errorCount 大于某个阈值，如 errorCount > 0。

关联查询告警

当从仪表盘入口创建告警时，可以选择多个图表作为告警查询的输入。

- 对不同时间范围的查询结果进行组合告警。

  如 15分钟内的PV 大于100000 且一小时内的UV 小于1000时触发告警：

说明：

在选择多个图表时，查询区间相互独立。在触发条件中需要使用 ${编号}.{字段} 的方式引用查询结果中的字段。如：$0.pv > 100000 && $1.uv < 1000 。

- 基于部分图表告警，其他图表的查询结果作为辅助信息。

  基于日志级别为ERROR的日志条数告警，查询语句：

  ```
  level: ERROR | select count(1) as errorCount
  ```

  告警条件：

  ```
  errorCount > 10
  ```

  与此同时，也希望能够在告警通知中看到实际的日志级别为ERROR的日志，则可以再配置第2个查询：

  ```
  level: ERROR
  ```

  在告警通知中只需要设置：

  ```
  ${results[1].RawResultsAsKv}
  ```

  即可看到实际的日志级别为ERROR的日志。

告警抑制

当告警触发时，可能会在一段时间内多次收到通知。为了防止因为数据抖动导致的误报和重复告警，可以通过如下两种方式对告警进行抑制：

- 设置连续触发通知阈值。

  只有告警在连续多次检查中都满足告警条件才会触发告警。

  如告警执行间隔为1分钟，触发阈值为5，则表示在连续5次即5分钟内每次告警检查结果都满足告警条件才会发送通知。只要有一次没有满足触发条件，计数将会重置。

- 设置通知间隔。

  当告警设置的执行间隔较小时，防止频繁收到通知，可以设置两次通知之间的最小间隔。如告警执行间隔为1分钟，通知间隔为30分钟，即使30分钟内有告警触发，也不会收到任何通知。

关闭告警通知

收到告警通知之后，如果希望临时关闭通知。可以通过告警概览页面关闭通知，如下图所示：

选择关闭的时长，如30分钟：

则在30分钟内，不会再发送任何通知，即便告警触发。在30分钟之后，通知自动恢复。

钉钉群成员查看告警

钉钉群是最常见的告警通知渠道，在配置钉钉通知时，我们可以@钉钉群的成员处理告警。如下图所示：

说明:

需要在被@人列表和发送内容中同时指定对应成员的手机号。被@人列表是用于识别发送内容中的@是提醒还是普通的@字符。

使用模版变量丰富通知内容

在配置通知方式时，可以使用模版变量来丰富通知内容。邮件标题，钉钉标题，消息内容都支持使用模版变量。 每次告警执行的时候，都会生成一个告警的上下文，其中的每个变量都可以作为模版变量，完整的变量可以参考#unique_81。

- **对于顶层的变量如Project，AlertName，Dashboard，可以直接使用${project} 这种方式引用，不区分大小写。**
- **对于每个查询的上下文，包含在Results 这个数组中，数组中的每个元素对应告警关联的一个图表（对于大多数场景，可能只有一个元素），包含的变量如下所示：**

```
{
  "EndTime": "2006-01-02 15:04:05",
```

```
  "EndTimeTs": 1542507580,
  "FireResult": {
    "__time__": "1542453580",
    "field": "value1",
    "count": "100"
  },
  "FireResultAsKv": "[field:value1,count:100]",
  "Truncated": false,
  "LogStore": "test-logstore",
  "Query": "* | SELECT field, count(1) group by field",
  "QueryUrl": "http://xxxx",
  "RawResultCount": 2,
  "RawResults": [
    {
      "__time__": "1542453580",
      "field": "value1",
      "count": "100"
    },
    {
      "__time__": "1542453580",
      "field": "value2",
      "count": "20"
    }
  ],
  "RawResultsAsKv": "[field:value1,count:100],[field:value2,count:20
]",
  "StartTime": "2006-01-02 15:04:05",
  "StartTimeTs": 1542453580
}
```

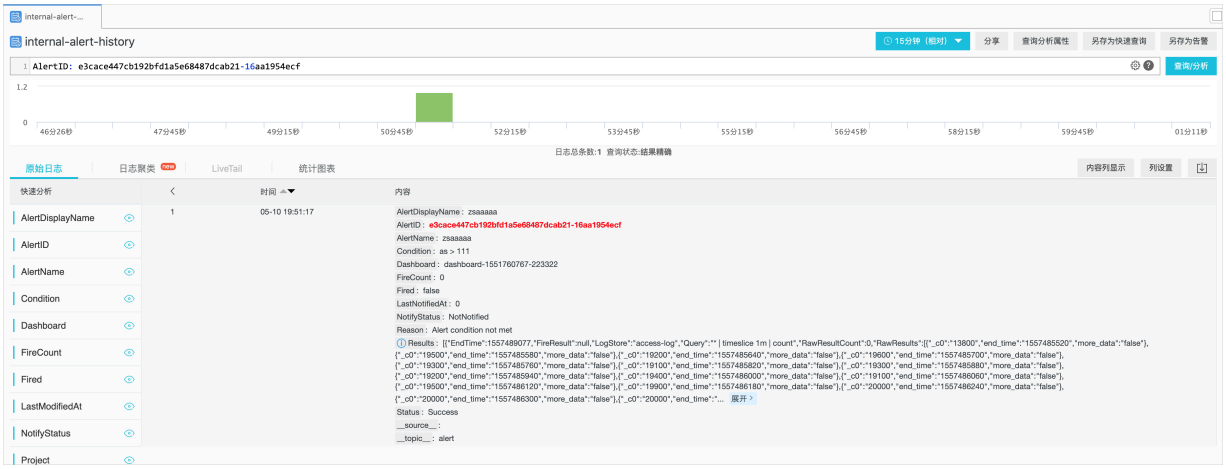

字段解释可以参考#unique_82。Results中的字段时可以通过如下方式引用：

- 数组类型通过"${fieldName[{index}]}"方式引用， 下标从0开始。如 ${results[0]} 表示引用Results的第1个元素。
- 对象类型通过"${object.key}"引用，如 ${results[0].StartTimeTs}的结果为 1542453580。

> **说明:**
> 只有RawResults 和FireResult 内的字段为查询结果，区分大小写，其他字段均不区分大小写。

排查告警未触发原因

配置告警之后，可以通过#unique_83查看告警统计。对于单次告警的上下文，可以直接在internal-alert-history这个Logstore中查看，如下图所示。



日志字段解释参考#unique_82。

每次执行都会生成一个唯一的告警ID和一条对应的日志，日志中包含了告警执行的状态和查询的结果（如果查询结果超过2KB，会被截断），通过日志可以排查告警没有触发的原因。