ALIBABA CLOUD

阿里云

日志服务

最佳实践

文档版本：20201105

阿里云

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
危险	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。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会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，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	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。	单击**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。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	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。	`switch {active|stand}`

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。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1万台服务器属于一家大公司了，但10万在线对于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：难以登录并调试，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 Producer

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，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、高性能、低资源消耗等特点，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20万日志写入只消耗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：不同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
	并发度	高	一般

类别	稳定性	C Producer	其他方案
	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（中国）+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：

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_producer_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20万，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，每秒可发送最多2万条日志。
  - 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

本文档为您介绍如何通过WebTracking采集日志数据到日志服务中，并对采集到的日志数据进行查询和分析。

背景信息

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等打通。
- 数据投递：LogShipper。与MaxCompute、E-MapReduce、OSS、Function Compute等打通。
- 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。

- 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();
```

详细步骤请参见使用Web Tracking采集日志。

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。

- 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

1. 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。
2. 生成Web Tracking标签。

i. 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）。

- 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。

ii. 将img标签放置在宣传内容中，并进行发布。

3. 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。

i. 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
```

ii. 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-搭建移动端日志直传服务

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

背景信息

普通模式下，日志写入日志服务需要开启主账号的访问密钥，用于鉴权以及防篡改。移动应用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，需要升级移动应用，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，如果移动应用数量较大，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、更便捷的移动应用日志数据采集方案，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，用户不需要在移动应用端保存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，例如限制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，并配置移动应用作为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

架构说明

节点	说明
Android/iOS 移动应用	最终用户手机上的应用，日志的来源。
LOG	即阿里云日志服务，负责存储应用上传的日志数据。
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手机应用后台服务，管理应用上传和下载的Token，以及用户在应用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，并配置移动应用作为RAM子用户扮演该角色，详细步骤请参见授权用户角色。

   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
   >
   > i. AccessKeyID：填写您的访问密钥ID。
   >
   > ii. AccessKeySecret：填写您的访问密钥Secret。
   >
   > iii. RoleArn：填写用户角色的RoleArn。
   >
   > iv. 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。
   >
   > v. 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/cjLYg80Aho1ek0Jm63XMhr9Oc5s·∂·∂3qaPer8p1YaX1NTDiCFZWFkvlHf1pQhuxfKBc+mRR9KAbHUefqH+rdjZqjTF7p2m1wJXP8S6k+G2MpHrUe6TYBkJ43GhhTVFMuM3BZajY3VjZWOXBIODRIR1FKZjIiEjMzMzE0MjY0NzM5MTE4NjkxMSoLY2xpZGSSDgSDGAGESGTETqOio6c2RrLWRlbW8vKgoUYWNzOm9zczoqOio6c2RrLWRlbW9KEDExNDg5MzAxMDcyNDY4MThSBTI2ODQyWg9Bc3N1bWVkUm9sZVVzZXJgAGoSMzMzMTQyNjQ3MzkxMTg2OTExcglzZGstZGVtbzI=",
  "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。

错误码	说明
ErrorCode	表示错误原因
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。

2.4. 采集公网数据

对于一些应用场景，如移动端、HTML 网页、PC、服务器、硬件设备、摄像头等，需要实时收集公网数据并进行实时处理。在传统架构中，一般通过前端服务器+Kafka来实现如上功能。现在日志服务LogHub功能可以代替这类架构，并提供稳定、低成本、弹性、安全的解决方案。

应用场景

公网数据采集包含移动端、外部服务器、网页和设备数据的采集，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Nginx服务器做公网代理，再使用Logstash或API通过Nginx将公网数据写入到Kafka等消息中间件。需要的设施如下：

设施	数目	配置	作用	价格
ECS 服务器	2 台	1 核 2GB	将 2 台 ECS 都作为前端机，同时提供服务	108 元/台*月
负载均衡	1 台	标准	按量计费实例	14.4 元/月（租赁）+ 0.8 元/GB（流量）
已安装Kafka和ZooKeeper的服务器	3 台	1 核 2GB	数据写入并处理	108 元/台*月

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	< 0.18 元/GB，具体请参见 计费规则。

方案对比

场景 1：一天采集10GB数据，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：0 元（上行流量免费，无公网下行流量）

ECS 费用：108 * 2 = 216 元（假设磁盘费用免费）

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 * 0.18 * 30 = 54 元

loghub 请求次数：0.12 * 30 = 3.6 元

共计：57.6 元/月

场景 2：一天采集 1TB 数据，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：0 元（上行流量免费，无公网下行流量）

负载均衡 规格费：0.63 * 24 * 30 = 453.6 元（标准型II (slb.s2.medium)）

ECS 费用：240 * 2 = 480 元（ecs.g6.large）

ECS 磁盘费用：4800 元（高峰是均值的2倍，单Replica需要确保50 MB/S写入，数据保存3天6 TB容量， SSD云盘月费用在4800元）

Kafka ECS: 免费，假设与其他服务公用

共计：6696 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 * 30 = 360 元

共计：4860 元/月

对比结果

从以上两个场景可以看出，使用方案2进行公网数据采集，比方案1的成本低很多。除此之外，和方案1相比方案2还具有以下优势。

- 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：不需要额外费用就能与数据仓库对接。
- 监控数据丰富：可以清楚的知道业务的具体情况。
- 基于SDK与上下游对接：LogHub支持多种SDK。您可以使用这些SDK将LogHub与上下游系统对接，以实现LogHub与开源产品及其他阿里云产品深度整合。

请参见 日志服务主页 体验该服务。

2.5. 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，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
  - 扫描网页二维码登录

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，具体请参见Logtail简介。
- 应用程序通过SDK写入，具体请参见概述 。

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

  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?AccessKeyId=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: Connecti
on 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-devcommo
n-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、登录、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 keyCod
e=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_1472555483
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。

3.查询分析

3.1. 关联Logstore与MySQL数据库进行查询分析

本文以游戏公司数据分析场景为例，介绍日志服务Logstore与MySQL数据库关联分析功能。

前提条件

- 已采集日志到日志服务，详情请参见数据采集。
- 已为日志字段创建索引，详情请参见开启并配置索引。
- 已有可用的MySQL数据库。

  RDS MySQL数据库详情请参见创建数据库和账号。

背景信息

某游戏公司，主要包括两大类数据：用户游戏日志数据和用户元数据。日志服务可实时采集用户游戏日志数据，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等信息。然而用户元数据，包括用户的性别、注册时间、地区等信息，不能打印到日志中，所以一般存储到数据库中。现在该公司希望将用户游戏日志与用户元数据进行联合分析，获得最佳的游戏运营方案。

针对上述需求，日志服务查询分析引擎，提供Logstore和外部数据源（ExternalStore，例如MySQL数据库、OSS等）联合查询分析功能。您可以使用SQL的JOIN语法把用户游戏日志和用户元数据关联起来，分析与用户属性相关的指标。除此之外，您还可以将计算结果直接写入外部数据源中，便于结果的进一步处理。

操作步骤

1. 在MySQL数据库中，创建用户属性表。

   创建一张名为chiji_user的数据表，用于保存用户ID、昵称、性别、年龄、注册时间、账户余额和注册省份。

   ```
   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_TIMEST
   AMP,
    `balance` float DEFAULT NULL,
    `province` text, PRIMARY KEY (`uid`)
   ) ENGINE=InnoDB DEFAULT CHARSET=utf8
   ```

2. 添加白名单。
   - 如果是RDS MySQL数据库，需添加白名单地址100.104.0.0/16、11.194.0.0/16和11.201.0.0/16，详情请参见设置白名单。

- 如果是自建的MySQL数据库，需添加白名单地址100.104.0.0/16、11.194.0.0/16和11.201.0.0/16到MySQL数据库的安全组。

3. 创建ExternalStore。
   i. 安装日志服务CLI，详情请参见命令行工具CLI。
   ii. 创建配置文件*/root/config.json*。
   iii. 在*/root/config.json*文件中添加如下脚本，并根据实际情况替换参数配置。

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

参数	说明
region	RDS实例所在地域 ■ 如果是RDS MySQL，则必须配置region。 ■ 如果是自建的MySQL，则region配置为空字符串，即配置为**"region": ""**。
vpc-id	VPC ID ■ 如果是专有网络下的RDS MySQL，则必须配置vpc-id。 ■ 如果是经典网络下的RDS MySQL或者自建的MySQL，则vpc-id配置为空字符串，即配置为**"vpc-id": ""**。
instance-id	RDS实例ID ■ 如果是RDS MySQL，则必须配置instance-id。 ■ 如果是自建的MySQL，则instance-id配置为空字符串，即配置为**"instance-id": ""**。
host	数据库地址
port	网络端口

参数	说明
username	数据库用户名
password	数据库密码
db	数据库
table	数据表

iv. 创建ExternalStore。其中project_name为日志服务Project名称，请根据实际值替换。

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

4. 使用JOIN语法进行联合查询分析。
   i. 登录日志服务控制台。
   ii. 在**Project列表**区域，单击目标Project。
   iii. 在**日志存储 > 日志库**页签中，单击目标Logstore。
   iv. 执行查询分析语句。

   指定日志中的userid字段和数据库表中的uid字段关联Logstore和MySQL数据库。

   - 分析活跃用户的性别分布。

     ```
     * | 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
     ```

5. 保存查询分析结果到MySQL数据库中。
   i. 在MySQL数据库中，创建名为report的数据表，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
   ```

   ii. 参见步骤3为report表创建ExternalStore。

iii. 在日志服务Logstore的查询分析页面中，执行如下查询语句将分析结果保存到report表中。

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

保存成功后，您可以在MySQL数据库中查看保存结果。

3.2. 关联Logstore与OSS外表进行查询分析

在进行日志数据查询分析时，经常需要结合外部表格对日志数据进行分析。本文介绍如何在日志服务中联合OSS外表进行数据分析。

前提条件

- 已采集日志，详情请参见数据采集。
- 已开启并配置索引，详情请参见开启并配置索引。
- 已创建OSS Bucket，详情请参见创建存储空间。

背景信息

某支付公司，想要分析用户年龄、地域、性别等因素对支付习惯的影响。该公司已通过日志服务实时采集用户支付行为（支付方式、支付费用等）日志，并将用户属性（地域、年龄、性别等）信息保存在OSS中。针对该场景，日志服务查询分析引擎提供Logstore和外部数据源（ExternalStore，例如MySQL数据库、OSS等）联合查询分析功能。您可以使用SQL的JOIN语法把用户属性数据和行为数据进行联合，分析与用户属性相关的指标。

与OSS进行关联查询分析，具有如下优势：

- 节省费用：将更新频率低的数据保存在OSS上，只需要支付少量的存储费用，并且可以通过内网读数据，免去流量费用。
- 降低运维工作：在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。
- 节省时间：使用SQL分析数据，分析结果秒级可见，并可以将常用的分析结果定义为报表，打开即可看到结果。

操作步骤

1. 创建CSV文件并上传到OSS。
   i. 创建名为*user.csv*的文件。

   

   ii. 上传*user.csv*文件到OSS，详情请参见上传文件。
2. 登录日志服务控制台。
3. 在**Project列表**区域，单击目标Project。
4. 在**日志存储 > 日志库**页签中，单击目标Logstore。
5. 输入查询分析语句，单击**查询/分析**。通过SQL定义虚拟外部表（此处以user_meta1为例），映射到OSS文件，如果执行结果中的**result**为**true**，表示执行成功。

```
* | create table user_meta1 ( userid bigint, nick varchar, gender varchar, province varchar, gender varch
ar,age bigint) with ( endpoint='example.com',accessid='<youraccessid>',accesskey='<accesskey>',buck
et='testossconnector',objects=ARRAY['user.csv'],type='oss')
```

在查询分析语句中定义外部存储名称、表的Schema等信息，并通过WITH语法指定OSS访问信息及文件信息，详细信息如下表所示。

配置项	说明
外部存储名称	外部存储名称，即虚拟表的名称，例如user_meta1。
表的Schema	定义表的属性，包括表的列名及格式，例如(userid bigint, nick varchar, gender varchar, province varchar, gender varchar,age bigint)。
endpoint	OSS Bucket访问域名，详情请参见访问域名和数据中心。
accessid	您的AccessKey ID。
accesskey	您的AccessKey Secret。
bucket	CSV文件所在的OSS Bucket名称。
objects	CSV文件路径。 **说明** objects为array类型，可以包含多个OSS文件。
type	固定为oss，表示外部存储类型为OSS。

6. 验证是否成功定义外部表。 执行如下语句，如果返回结果为您之前定义的表内容，则表示定义外部表成功。

```
select * from user_meta1
```

7. 通过JOIN语法完成Logstore和OSS外表的联合查询。

   执行如下查询分析语句关联日志服务中日志的ID和OSS文件中的userid，补全日志信息。其中，test_accesslog为Logstore名称，l为Logstore别名，user_meta1为您定义的外部存储表，请根据实际情况替换。

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

   联合查询示例：

   - 统计不同性别用户的访问情况

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

   - 统计不同年龄用户的访问情况

```
* | select u.age, count(1) from test_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

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

3.3. 查询MNS日志

阿里云消息服务（MNS）日志推送到日志服务后，可进行实时查询，本文介绍实时查询的常用场景及操作步骤，您也可以通过多个关键字组合方式实现更加复杂的查询。

前提条件

- 已采集到MNS日志，详情请参见MNS日志。
- 已开启并配置索引，详情请参见开启并配置索引。

背景信息

MNS日志包括队列消息操作日志和主题消息操作日志，日志内容包含消息生命周期的所有信息，例如时间、客户端、操作等。您可以通过实时查询、实时计算和离线计算三种方式对日志进行分析计算。

- 实时查询：在日志服务控制台上进行实时查询，例如：查询消息轨迹、写入量、删除量等。
- 实时计算：使用Spark、Storm、StreamCompute、Consumer Library等方式对MNS日志进行实时计算。例如：计算某个队列中，Top 10消息的产生者和消费者；计算生产和消费的速度，确认是否均衡；计算某些消费者的处理延时，确认是否存在瓶颈等。
- 离线计算：使用MaxCompute、E-MapReduce、Hive进行长时间跨度的计算，例如：计算最近一周内消息从发布到被消费的平均延迟。

查询队列消息的消息轨迹

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 输入查询语句。本案例要查询队列消息的消息轨迹，即输入队列名称和消息ID，格式为**$QueueName and $MessageId**，例如**log and FF973C9C6572630D7F963C527CC5A82C**。
5. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

6. 单击**查询/分析**。

   查询结果如下所示，记录了某条消息从发送到接收的过程。

查询队列消息发送量

1. 在目标Logstore的查询分析页面，输入查询语句。本案例要查询队列消息发送量，即输入队列名称和发送操作，查询语句格式为**$QueueName and (SendMessage or BatchSendMessage)**，例如**log and (SendMessage or BatchSendMessage)**。
2. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

3. 单击**查询/分析**。查询结果如下所示，当前查询时段内，生产者向log队列发送了3条队列消息。

查询队列消息消费量

1. 在目标Logstore的查询分析页面，输入查询语句。本案例要查询队列消息消费量，即输入队列名称和消费操作，查询语句格式为**$QueueName and (ReceiveMessage or BatchReceiveMessage)**，例如**log and (ReceiveMessage or BatchReceiveMessage)**。
2. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

3. 单击**查询/分析**。

   查询结果如下所示，当前查询时段内，log队列中有12条消息被消费。

查询队列消息删除量

1. 在目标Logstore的查询分析页面，输入查询语句。本案例要查询队列消息删除量，即输入队列名称和删除操作，查询语句格式为**$QueueName and (DeleteMessage or BatchDeleteMessage)**，例如**log and (DeleteMessage or BatchDeleteMessage)**。
2. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

3. 单击**查询/分析**。

   查询结果如下所示，当前查询时段内，61条log队列消息被删除。

查询主题消息的消息轨迹

1. 在目标Logstore的查询分析页面，输入查询语句。本案例要查询主题消息的消息轨迹，即输入主题名称和MessageId，查询语句格式为**$TopicName and $MessageId**，例如**logtest and 979628CD657261357FCB3C8A68BFA0E3**。
2. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

3. 单击**查询/分析**。

   查询结果如下图所示，记录了某条消息从发送到通知的过程。

   

查询主题消息发布量

1. 在目标Logstore的查询分析页面，输入查询语句。本案例要查询主题消息发布量，即输入主题名称和发布操作，查询语句格式为**$TopicName and PublishMessage**，例如**logtest and PublishMessage**。
2. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

3. 单击**查询/分析**。

   查询结果如下图所示，当前查询时段内，生产者向logtest主题发布了3条消息。

   

查询某个客户端消息处理量

1. 在目标Logstore的查询分析页面，输入查询语句。本案例要查询某个客户端消息处理量，即输入客户端IP地址，查询语句格式为**$ClientIP**，例如**10.10.10.0**。

   如果您要查询某个客户端的某类操作日志，可使用多个关键字组合方式，例如**$ClientIP and (SendMessage or BatchSendMessage)**。
2. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

3. 单击**查询/分析**。

   查询结果如下图所示，当前查询时段内，该客户端处理了66条消息。

   

3.4. 采集及分析Nginx监控日志

Nginx中的自建状态页，可帮忙您监控Nginx状态。日志服务支持通过Logtail插件采集Nginx状态信息，并对监控日志进行查询分析、告警等操作，全方位监控您的Nginx集群。

前提条件

已在服务器上安装Logtail，详情请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。

> **说明** 目前支持Linux Logtail 0.16.0及以上版本，Window Logtail 1.0.0.8及以上版本。

步骤1 准备环境

请按照以下步骤，开启Nginx status插件。

1. 执行以下命令确认Nginx已具备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配置文件（默认为/etc/nginx/nginx.conf）中开启status功能，配置示例如下所示，详情请参见Nginx status。

> **说明** allow 10.10.XX.XX表示只允许IP地址为10.10.XX.XX的服务器访问nginx status功能。

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

3. 执行如下命令验证安装Logtail的服务器具备nginx status访问权限。

```
$curl http://10.10.XX.XX/private/nginx_status
```

如果回显信息如下所示，则表示已完成Nginx status配置。

```
Active connections: 1
server accepts handled requests
2507455 2507455 2512972
Reading: 0 Writing: 1 Waiting: 0
```

步骤2 采集Nginx监控日志

1. 登录日志服务控制台。
2. 在**接入数据**区域，选择**自定义数据插件**。
3. 在**选择日志空间**页签中，选择目标Project和Logstore，单击**下一步**。 您也可以单击**立即创建**，重新创建Project和Logstore，详情请参见步骤1：创建Project和Logstore。
4. 在**创建机器组**页签中，创建机器组。
   - 如果您已有可用的机器组，请单击**使用现有机器组**。
   - 如果您还没有可用的机器组，请执行以下操作（以ECS为例）。
     a. 选择ECS实例安装Logtail，详情请参见安装Logtail（ECS实例）。

        如果已在ECS上安装Logtail，可直接单击**确认安装完毕**。

        > **说明** 如果是自建集群、其他云厂商服务器，需要手动安装Logtail，详情请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。

     b. 安装完成后，单击**确认安装完毕**。

c. 创建机器组，详情请参见创建IP地址机器组或创建用户自定义标识机器组。

5. 在**机器组配置**页签中，应用机器组。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. 在**数据源设置**页签中，配置**配置名称**和**插件配置**。
   - inputs为Logtail采集配置，必选项，请根据您的数据源配置。

     > **说明** 一个inputs中只允许配置一个类型的数据源。

   - processors为Logtail处理配置，可选项。您可以配置一种或多种处理方式，详情请参见处理数据。

```
{
"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
   }
}
]
}
```

重要参数说明如下表所示：

参数	类型	是否必须	说明
type	string	是	数据源类型，固定为metric_http。
IntervalMs	int	是	每次请求的间隔，单位：ms。
Addresses	string[]数组	是	配置为您需要监控的URL列表。
IncludeBody	bool	否	是否采集请求的body，默认值：false。如果为true，则将请求body内容存放在名为content的字段中。

7. 在**查询分析配置**页签中，设置索引。默认已设置索引，您也可以根据业务需求，重新设置索引，具体请参见开启并配置索引。

   > **说明**
   > - 全文索引和字段索引属性必须至少启用一种。同时启用时，以字段索引属性为准。
   > - 索引类型为long、double时，大小写敏感和分词符属性无效。

完成采集配置1分钟后，即可查看日志数据，样例如下所示。并且日志服务默认生成nginx_status仪表盘，展示查询分析结果。

```
_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
```

步骤3 查询分析日志

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

5. 在搜索框中输入查询分析语句，单击**查询/分析**。查询分析语句详情请参见实时分析简介，您还可以为查询结果设置告警，详情请参见设置告警。
   - 查询日志
     - 查询某IP地址的请求状态。

       ```
       _address_ : 10.10.0.0
       ```

     - 查询响应时间超过100ms的请求。

       ```
       _response_time_ms_ > 100
       ```

     - 查询状态码非200的请求。

       ```
       not _http_response_code_ : 200
       ```

   - 分析日志
     - 每5分钟统计waiting、reading、writing、connection的平均值。

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

     - 统计最大等待连接数排名前十的服务器。

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

     - 统计请求IP的数量以及请求失败的IP数量。

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

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

     - 统计最近十次访问失败的IP地址。

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

     - 每5分钟统计请求处理总数。

       ```
       *| select avg(handled) * count(distinct(address)) as total_handled, 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
       ```

- 统计请求成功的数量和失败的数量。

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

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

3.5. 分析Nginx访问日志

日志服务支持采集Nginx日志，并进行多维度分析。本文介绍分析网站访问情况、诊断及调优网站和重要场景告警的分析案例。

前提条件

已采集Nginx访问日志，详情请参见Nginx模式。

在日志采集配置向导中，已根据日志字段自动生成索引，如果您要修改索引，详情请参见开启并配置索引。

背景信息

Nginx是一款主流的网站服务器，当您选用Nginx搭建网站时，Nginx日志是运维网站的重要信息。传统模式下，需使用CNZZ等方式，在前端页面插入JS，记录访问请求。或者利用流计算、离线计算分析Nginx访问日志，此方式还需要搭建环境，在实时性以及分析灵活性上难以平衡。

日志服务支持通过数据接入向导一站式采集Nginx日志，并为Nginx日志创建索引和仪表盘。nginx_Nginx访问日志仪表盘包括来源IP分布、请求状态占比、请求方法占比、访问PV/UV统计、流入流出流量统计、请求UA占比、前十访问来源、访问前十地址和请求时间前十地址等信息，全方位展示网站访问情况。您还可以使用日志服务的查询分析语句，分析网站的延时情况，及时调优网站。针对性能问题、服务器错误、流量变化等重要场景，您还可以设置告警，当满足告警条件时给您发送告警信息。

分析网站访问情况

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标Project。
3. 在**日志存储 > 日志库**中，单击目标Logstore左侧的>。
4. 在**可视化仪表盘**中，单击**nginx_Nginx访问日志**。nginx_Nginx访问日志仪表盘中的重要图表说明如下所示：
   - **来源IP分布**图展示最近一天访问IP地址的来源情况，所关联的查询分析语句如下所示：

     ```
     * | select count(1) as c, ip_to_province(remote_addr) as address group by address limit 100
     ```

     Nginx访问日志-访问地域分析
   - **请求状态占比**图展示最近一天各HTTP状态码的占比情况，所关联的查询分析语句如下所示：

     ```
     * | select count(1) as pv,
         status
         group by status
     ```
   - **请求方法占比**图展示最近一天各请求方法的占比情况，所关联的查询分析语句如下所示：

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

- **请求UA占比**图展示最近一天各种浏览器的占比情况，所关联的查询分析语句如下所示：

```
* | select count(1) as pv, case when http_user_agent like '%Chrome%' then 'Chrome' when http_user_agent like '%Firefox%' then 'Firefox' when http_user_agent like '%Safari%' then 'Safari' else 'unKnown' end as http_user_agent  group by case when http_user_agent like '%Chrome%' then 'Chrome' when http_user_agent like '%Firefox%' then 'Firefox' when http_user_agent like '%Safari%' then 'Safari' else 'unKnown' end   order by pv desc limit 10
```

- **前十访问来源**图展示最近一天PV数最多的前十个访问来源页面，所关联的查询分析语句如下所示：

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

Nginx访问日志-前十访问来源

- **流入流出流量统计**图展示最近一天流量的流入和流出情况，所关联的查询分析语句如下所示：

```
* | select sum(body_bytes_sent) as net_out, sum(request_length) as net_in ,date_format(date_trunc('hour', __time__), '%m-%d %H:%i')  as time group by date_format(date_trunc('hour', __time__), '%m-%d %H:%i') order by time limit 10000
```

流量统计

- **访问PV/UV统计**图展示最近一天内的PV数和UV数，所关联的查询分析语句如下所示：

```
*| select approx_distinct(remote_addr) as uv ,count(1) as pv , date_format(date_trunc('hour', __time__), '%m-%d %H:%i')  as time group by date_format(date_trunc('hour', __time__), '%m-%d %H:%i') order by time limit 1000
```

Nginx访问日志的PV/UV统计

- **PV预测**图预测未来4小时的PV数，所关联的查询分析语句如下所示：

```
* | select ts_predicate_simple(stamp, value, 6, 1, 'sum') from (select __time__ - __time__ % 60 as stamp, COUNT(1) as value from log GROUP BY stamp order by stamp) LIMIT 1000
```

PV预测

- **访问前十地址**图展示最近一天PV数最多的前十个访问地址，所关联的查询分析语句如下所示：

```
* | select count(1) as pv, split_part(request_uri,'?',1) as path  group by path order by pv desc limit 10
```

Nginx访问日志-统计访问数据

诊断及调优网站

在网站运行过程中，还需关注请求延时问题，例如处理请求延时情况如何、哪些页面的延时较大等。您可以自定义查询分析语句分析延迟情况，相关案例如下所示，操作步骤可参见查询分析日志。

- 计算每5分钟请求的平均延时和最大延时，从整体了解延时情况。

```
* | select from_unixtime(__time__ -__time__% 300) as time,
    avg(request_time) as avg_latency ,
    max(request_time) as max_latency
    group by __time__ -__time__% 300
```

- 统计最大延时对应的请求页面，进一步优化页面响应。

```
* | select from_unixtime(__time__ - __time__% 60) ,
    max_by(request_uri,request_time)
    group by __time__ - __time__%60
```

- 统计分析网站所有请求的延时分布，将延时分布分成10个组，分析每个延时区间的请求个数。

```
* |select numeric_histogram(10,request_time)
```

- 计算最大的十个延时及其对应值。

```
* | select max(request_time,10)
```

- 对延时最大的页面进行调优。

  例如/url2页面的访问延时最大，需要对/url2页面进行调优，则需计算/url2页面的访问PV、UV、各种请求方法次数、各种请求状态次数、各种浏览器次数、平均延时和最大延时。

```
request_uri:"/url2" | select count(1) as pv,
    approx_distinct(remote_addr) as uv,
    histogram(method) as method_pv,
    histogram(status) as status_pv,
    histogram(user_agent) as user_agent_pv,
    avg(request_time) as avg_latency,
    max(request_time) as max_latency
```

告警

针对性能问题、网站错误、流量急跌或暴涨等情况，您可以设置查询分析语句，并设置告警，操作步骤请参见设置告警。

- 错误告警

  在网站运行过程中，一般需关注500错误，即服务器错误。您可以使用如下查询分析语句计算单位时间内的错误数c，并将告警触发条件设置为c > 0。

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

> **说明** 对于一些业务压力较大的服务，偶尔出现几个500错误是正常现象。针对此情况，您可以在创建告警时，设置**触发通知阈值**为2，即只有连续2次检查都符合条件才产生告警。

- 性能告警

  如果在服务器运行过程出现延迟增大情况，您可以针对延迟创建告警。例如您可以使用如下查询分析语句分析 /adduser 接口所有请求方法为 Post 的写请求延时，并将告警触发条件设置为 l > 300000，即当延时平均值超过300ms则产生告警。

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

  使用平均值创建告警的方式比较简单，但会造成一些个体请求延时被平均，无法反映真实情况。针对此问题，您可以使用数学统计中的百分数（例如99%最大延时）来作为告警触发条件，例如使用如下查询分析语句计算99%分位的延时大小。

  ```
  Method:Post and URL:"/adduser" | select approx_percentile(Latency, 0.99) as p99
  ```

  在监控场景中，您可以使用如下查询分析语句计算一天窗口（1440分钟）内各分钟的平均延时大小、50%分位的延时大小和90%分位的延时大小。

  ```
  * | 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
  ```

  判断是否有性能问题-分布图2

- 流量急跌或暴涨告警

  如果在网站运行过程中出现流量急跌或暴涨情况，一般属于不正常现象。针对此问题，您可以计算流程大小，并设置告警。一般根据如下参考信息反映流量的急跌或暴涨情况：

  - 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。

  本案例以第一种情况为例，计算流量大小的变动率，日志查询范围为5分钟。

  i. 定义一个计算窗口。

     定义一个1分钟的窗口，计算该分钟内的流量大小。

     ```
     * | 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
     ```

     从分析结果中看，每个窗口内的平均流量是均匀的。

     首先定义一个计算窗口

  ii. 计算窗口内的差异值。

     - 计算最大值或最小值与平均值的变化率，此处以最大值max_ratio为例。

       本示例中计算结果max_ratio为1.02，您可以定义告警条件为max_ratio > 1.5（变化率超过50%）则告警。

       ```
       * | 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)
       ```

       计算窗口内的差异值

- 计算最近值变化率，查看最新的数值是否有波动或已恢复。

  通过max_by方法获取窗口中的最大流量进行判断，本案例中的计算结果lastest_ratio为0.97。

  ```
  * | select max_by(inflow, window_time)/1.0/avg(inflow) as lastest_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)
  ```

  > **说明** max_by函数计算结果为字符类型，需强制转换成数字类型。 如果您要计算变化相对率，可以用（1.0-max_by(inflow, window_time)/1.0/avg(inflow)) as lastest_ratio。

  计算窗口内的差异值

- 计算波动率，即计算当前窗口值与上个窗口值的变化值。

  计算窗口内的差异值

  使用窗口函数lag提取当前流量inflow与上一个周期流量inflow "lag(inflow, 1, inflow)over() "进行差值计算，并除以当前流量inflow获取变化率。例如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。

  > **说明** 如果要定义一个绝对变化率，可以使用abs函数对计算结果进行统一。

  ```
  * | 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)
  ```

  计算窗口内的差异值2

3.6. 分析Apache日志

日志服务支持采集Apache日志，并进行多维度分析。本文通过PV、UV、访问地域分布、错误请求、客户端类型等维度分析Apache日志，以评估网站访问情况。

前提条件

已采集Apache日志，详情请参见Apache模式。

在数据接入向导中，已根据日志字段自动生成索引，如果您要修改索引，详情请参见开启并配置索引。

背景信息

Apache是一款主流的网站服务器，当您选用Apache搭建网站时，Apache日志是运维网站的重要信息。

日志服务支持通过数据接入向导一站式采集Apache日志，并为Apache日志创建索引和仪表盘。apache_Apache访问日志仪表盘包括来源IP分布、请求状态占比、请求方法占比、访问PV/UV统计、流入流出流量统计、请求UA占比、前十访问来源、访问前十地址和请求时间前十地址等信息，全方位展示网站访问情况。

操作步骤

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标Project。

3. 在**日志存储 > 日志库**中，单击目标Logstore左侧的>。
4. 在**可视化仪表盘**中，单击**apache_Apache访问日志**。

   apache_Apache访问日志仪表盘包括如下图表：

   - **来源IP分布**图展示访问IP地址的来源情况，所关联的查询分析语句如下所示：

     ```
     * | select ip_to_province(remote_addr) as address, count(1) as c group by ip_to_province(remote_addr) limit 100
     ```

     

   - **请求状态占比**图展示最近一天各HTTP状态码的占比情况，所关联的查询分析语句如下所示：

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

     

   - **请求方法占比**图展示最近一天各请求方法的占比情况，所关联的查询分析语句如下所示：

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

     

   - **访问PV/UV统计**图展示最近一天内的PV数和UV数，所关联的查询分析语句如下所示：

     ```
     * | select date_format(date_trunc('hour', __time__), '%m-%d %H:%i')  as time, count(1) as pv, approx_distinct(remote_addr) as uv group by date_format(date_trunc('hour', __time__), '%m-%d %H:%i') order by time limit 1000
     ```

     

   - **流入流出流量统计**图展示流量的流入和流出情况，所关联的查询分析语句如下所示：

     ```
     * | select date_format(date_trunc('hour', __time__), '%m-%d %H:%i') as time, sum(bytes_sent) as net_out, sum(bytes_received) as net_in group by time order by time limit 10000
     ```

     

   - **请求UA占比**图展示最近一天各种浏览器的占比情况，所关联的查询分析语句如下所示：

     ```
     * | select case when http_user_agent like '%Chrome%' then 'Chrome' when http_user_agent like '%Firefox%' then 'Firefox' when http_user_agent like '%Safari%' then 'Safari' else 'unKnown' end as http_user_agent, count(1) as pv group by case when http_user_agent like '%Chrome%' then 'Chrome' when http_user_agent like '%Firefox%' then 'Firefox' when http_user_agent like '%Safari%' then 'Safari' else 'unKnown' end   order by pv desc limit 10
     ```

     

   - **前十访问来源**图展示最近一天PV数最多的前十个访问来源页面，所关联的查询分析语句如下所示：

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

     

- **访问前十地址**展示最近一天PV数最多的前十个访问地址，所关联的查询分析语句如下所示：

```
* | select split_part(request_uri,'?',1) as path, count(1) as pv group by split_part(request_uri,'?',1) order by pv desc limit 10
```

访问前十地址

- **请求时间前十地址**图展示最近一天请求响应延时最长的前十个地址，所关联的查询分析语句如下所示：

```
* | select request_uri as top_latency_request_uri,
      request_time_sec
      order by request_time_sec desc limit 10 10
```

请求时间前十地址

3.7. 分析IIS日志

日志服务支持采集IIS日志，并进行多维度分析。本文通过PV、UV、访问地域分布、错误请求、请求方法等维度分析II日志，以评估网站访问情况。

前提条件

已采集IIS日志，详情请参见IIS模式。

在日志采集配置向导中，已根据日志字段自动生成索引，如果您要修改索引，详情请参见开启并配置索引。

背景信息

IIS是一款主流的网站服务器，具备简单易用、安全性能高等优势。当您选用IIS搭建网站时，IIS日志是运维网站的重要信息。

日志服务推荐选用W3C日志格式，W3C配置格式如下所示：

```
logExtFileFlags="Date, Time, ClientIP, UserName, SiteName, ComputerName, ServerIP, Method, UriStem, UriQuery, HttpStatus, Win32Status, BytesSent, BytesRecv, TimeTaken, ServerPort, UserAgent, Cookie, Referer, ProtocolVersion, Host, HttpSubStatus"
```

IIS日志样例如下所示：

```
#Software: Microsoft Internet Information Services 7.5
#Version: 1.0
#Date: 2020-09-08 09:30:26
#Fields: date time s-sitename s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username c-ip cs(User-Agent) sc-status sc-substatus sc-win32-status sc-bytes cs-bytes time-taken
2009-11-26 06:14:21 W3SVC692644773 125.67.67.* GET /index.html - 80 - 10.10.10.10 Baiduspider+(+http://www.baidu.com)200 0 64 185173 296 0
```

- 字段前缀说明

前缀	说明
s-	服务器操作
c-	客户端操作
cs-	客户端到服务器的操作
sc-	服务器到客户端的操作

- 各个字段说明

字段	说明
date	客户端发送请求的日期.。
time	客户端发送请求的时间。
s-sitename	服务名，表示客户端所访问的站点的Internet服务和实例的号码。
s-computername	服务器名，表示生成日志的服务器名称。
s-ip	服务器IP地址，表示生成日志的服务器的IP地址。
cs-method	请求方法，例如：GET、POST。
cs-uri-stem	URI资源，表示请求访问的地址。
cs-uri-query	URI查询，表示查询HTTP请求中问号（?）后的信息。
s-port	服务器端口，表示连接客户端的服务器端口号。
cs-username	通过验证的域或用户名。 对于通过身份验证的用户，格式为 `域\用户名` ；对于匿名用户，显示短划线（-）。
c-ip	客户端IP地址，表示访问服务器的客户端真实IP地址。
cs-version	协议版本，例如：HTTP 1.0、HTTP 1.1。
cs(User-Agent)	用户代理，表示在客户端使用的浏览器。
Cookie	Cookie，表示发送或接受的Cookie内容，如果没有Cookie，则显示短划线（-）。
referer	引用站点，表示用户访问的前一个站点。
cs-host	主机信息。
sc-status	协议返回状态，表示HTTP或FTP的操作状态。
sc-substatus	HTTP子协议的状态。

字段	说明
sc-win32-status	win32状态，表示操作状态。
sc-bytes	服务器发送的字节数。
cs-bytes	服务器接收的字节数。
time-taken	操作所花费的时间，单位：毫秒。

操作步骤

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

5. 在搜索框中输入查询分析语句，单击**查询/分析**。查询分析语句详情请参见查询分析语句格式。
   - 通过IP地址来源分析访问地域，查询分析语句如下所示：

     ```
     | select ip_to_geo("c-ip") as country, count(1) as c group by ip_to_geo("c-ip") limit 100
     ```

   - 统计最近的PV数和UV数，查询分析语句如下所示：

     ```
     *| select approx_distinct("c-ip") as uv ,count(1) as pv , date_format(date_trunc('hour', __time__), '%m-%d %H:%i') as time group by date_format(date_trunc('hour', __time__), '%m-%d %H:%i') order by time limit 1000
     ```

   - 统计HTTP请求状态码的占比，查询分析语句如下所示：

     ```
     *| select count(1) as pv ,"sc-status" group by "sc-status"
     ```

   - 统计流量的流入和流出情况，查询分析语句如下所示：

     ```
     *| select sum("sc-bytes") as net_out, sum("cs-bytes") as net_in ,date_format(date_trunc('hour', time), '%m-%d %H:%i') as time group by date_format(date_trunc('hour', time), '%m-%d %H:%i') order by time limit 10000
     ```

   - 统计各种请求方法的占比，查询分析语句如下所示：

```
*| select count(1) as pv ,"cs-method" group by "cs-method"
```

- 统计各种浏览器的占比，查询分析语句如下所示：

```
*| select count(1) as pv, case when "user-agent" like '%Chrome%' then 'Chrome' when "user-agent" li
ke '%Firefox%' then 'Firefox' when "user-agent" like '%Safari%' then 'Safari' else 'unKnown' end as "
user-agent" group by case when "user-agent" like '%Chrome%' then 'Chrome' when "user-agent" lik
e '%Firefox%' then 'Firefox' when "user-agent" like '%Safari%' then 'Safari' else 'unKnown' end order
by pv desc limit 10
```

- 统计访问数量前十的地址，查询分析语句如下所示：

```
*| select count(1) as pv, split_part("cs-uri-stem",'?',1) as path group by split_part("cs-uri-stem",'?',1)
order by pv desc limit 10
```

3.8. 分析Log4j日志

本文以电商平台的日志为例，为您介绍Log4j日志的分析操作。

前提条件

- 已采集Log4j日志，详情请参见采集Log4j日志。
- 已开启并配置索引，详情请参见开启并配置索引。

  本案例的索引如下图所示。

  指定字段查询

背景信息

Log4j是Apache的一个开放源代码项目，通过Log4j可以控制日志的优先级、输出目的地和输出格式。日志级别从高到低为ERROR、WARN、INFO、DEBUG，日志的输出目的地指定了将日志打印到控制台还是文件中，输出格式控制了输出的日志内容格式。

例如某电商公司，希望通过分析用户行为习惯数据（例如用户登录方式、上线的时间点及时长、浏览页面、页面停留时间、平均下单时间、消费水平等）、平台稳定性、系统报错、数据安全性等信息获取平台的最佳运营方案。针对此需求，日志服务提供一站式数据采集与分析功能，帮忙客户存储并分析日志。日志服务采集到的日志样例如下所示：

- 记录用户登录行为的日志：

```
level: INFO
location: com.aliyun.log4jappendertest.Log4jAppenderBizDemo.login(Log4jAppenderBizDemo.java:38)
message: User login successfully. requestID=id4 userID=user8
thread: main
time: 2018-01-26T15:31+0000
```

- 记录用户购买行为的日志：

```
level: INFO
location: com.aliyun.log4jappendertest.Log4jAppenderBizDemo.order(Log4jAppenderBizDemo.java:46)
message: Place an order successfully. requestID=id44 userID=user8 itemID=item3 amount=9
thread: main
time: 2018-01-26T15:31+0000
```

操作步骤

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1 min以内的误差。

5. 在搜索框中输入查询分析语句，单击**查询/分析**。查询分析语句详情请参见查询分析语句格式。
   - 统计最近1小时发生错误最多的3个位置。

     ```
     level: ERROR | select location ,count(*) as count GROUP BY  location  ORDER BY count DESC LIMIT 3
     ```

   - 统计最近15分钟各种日志级别产生的日志条数。

     ```
     | select level ,count(*) as count GROUP BY level ORDER BY count DESC
     ```

   - 统计最近1小时登录次数最多的三个用户。

     ```
     login | SELECT regexp_extract(message, 'userID=(?<userID>[a-zA-Z\d]+)', 1) AS userID, count(*) as count GROUP BY userID ORDER BY count DESC LIMIT 3
     ```

   - 统计最近15分钟每个用户的付款总额。

     ```
     order | SELECT regexp_extract(message, 'userID=(?<userID>[a-zA-Z\d]+)', 1) AS userID, sum(cast(regexp_extract(message, 'amount=(?<amount>[a-zA-Z\d]+)', 1) AS double)) AS amount GROUP BY userID
     ```

3.9. 查询分析程序日志

本文通过查询、关联分析、统计分析等场景介绍如何使用日志服务对程序日志进行查询和分析。

背景信息

程序日志内容全、存在一定共性，它是运维程序的重要信息，但程序日志具有如下不便于存储与分析的特性：

- 格式随意：不同开发者的代码风格不同，对应的日志风格也不同，难以统一。
- 数据量大：程序日志一般比访问日志大1个数量级。
- 分布的服务器多：大部分应用为无状态模式，运行在不同框架中，例如云服务器、容器服务等，对应的实

例数从几个到数千个，需要有一种跨服务器的日志采集方案。

- 运行环境复杂：程序运行在不同的环境中，产生的日志也保存在不同的环境中，例如应用相关的日志在容器中、API相关日志在FunctionCompute中、旧系统日志在本地IDC中、移动端相关日志在用户处、网页端日志在浏览器中等。

为了能够获得全量日志，必须把所有日志统一存储。针对该场景，日志服务提供多样化的日志采集方式及一站式分析功能，您可通过查询+SQL92语法对日志进行实时分析，并以图表形式直观展示分析结果。和开源方案对比，日志服务提供的解决方案在查询分析成本上仅是开源方案的25%。

查询程序日志

例如某APP出现订单错误或请求延时等问题，您可以通过查询语句在TB级数据量的日志中快速（1s内）定位问题。还可以根据业务需求，设置时间范围、查询关键字等信息，更精准地返回查询结果。

- 查询延时大于1s，并且请求方法是以Post开头的请求数据。

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

- 查找包含error关键词但不包含merge关键词的日志。

```
error not merge
```

关联分析程序日志

关联分析包括进程内关联与跨进程关联，区别如下：

- 进程内关联：一般比较简单，因为同一个进程前后日志都在一个文件中。在多线程环节中，只需根据线程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，一般通过RPC中传入的TracerId来进行关联。

- 进程内关联

  通过上下文查询查看关联日志。例如通过关键词查询定位到一个异常日志，然后单击**上下文浏览**，查看该日志前后N条日志，操作步骤请参见上下文查询。

  

  上下文查询结果如下所示：

  

- 跨进程关联

  跨进程关联也叫Tracing，比较常见的工具有鹰眼、Dapper、StackDriver Trace、Zipkin、Appdash、X-ray等。

  此处基于日志服务，实现基本的Tracing功能。您可以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，获取所有相关日志。

  

  例如通过SDK查询前端机、后端机、支付系统、订单系统等日志。获得结果后，制作一个前端页面将跨进程分析关联起来，如下图所示。

  

统计分析程序日志

查询到日志后，您还可以做进一步统计分析。

执行如下查询分析语句，统计所有错误发生的类型和位置的分布。

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

相关操作

- 备份日志

  将日志备份至OSS、MaxCompute等产品中。

- 关键词告警
  - 通过日志服务告警
  - 通过云监控告警

- 日志查询权限分配管理

  可以通过RAM用户授权方法隔离开发、PE等权限。

3.10. 分析网站日志

日志服务支持通过SQL92语法分析日志，并提供丰富的统计图表（例如表格、线图、柱状图、饼图、流图、地图等）展示分析结果。本文介绍如何在日志服务控制台上分析网站日志，并通过合适的统计图表可视化展示分析结果。

前提条件

- 已采集网站日志，详情请参见数据采集。
- 已配置索引，详情请参见开启并配置索引。

背景信息

网站日志是运维网站的重要信息，包含PV、UV、访问地域分布以及访问前十页面等信息。日志服务提供多样化的日志采集方式及一站式分析功能，您可通过查询+SQL92语法对日志进行实时分析，并以图表形式直观展示分析结果。日志服务还支持通过自带的仪表盘、DataV、Grafana、Tableua（通过JDBC链接）、QuickBI等可视化方式创建多种场景下的日志数据分析大盘。

功能体验

- 通过测试仪表盘体验查询分析功能。
  - 试用链接：仪表盘
  - 用户名：sls_reader1@aliyunlog.onaliyun.com
  - 密码：pnX-32m-MHH-xbm

  

- 通过数据实验室功能体验查询分析功能，详情请参见使用数据实验室。

操作步骤

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

5. 在搜索框中输入查询分析语句，单击**查询/分析**。日志服务提供丰富的统计图表，用于展示查询分析结果，详情请参见统计图表。
   - 通过表格展示最近1天客户端（remote_addr）访问情况，并降序排列。

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

     
   - 通过线图展示最近15分钟PV、UV以及平均响应时间的变化情况。

     ```
     * | 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
     ```

     配置**X轴**为*minutes*，**左Y轴**为*pv*和*uv*，**右Y轴**为*avg*，**为柱列**为*uv*，统计图表如下所示。

     
   - 通过柱状图展示最近15分钟不同来源地址（referer）的访问次数。

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

     配置**X轴**为*referer*，**Y轴**为*count*，统计图表如下所示。

     
   - 通过条形图展示最近15分钟访问前十的页面（request_uri）。

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

     配置**X轴**为*request_uri*，**Y轴**为*count*，统计图表如下所示。

     
   - 通过饼图展示最近15分钟页面（request_uri）访问情况。

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

     配置**分类**为*uri*，**数值列**为*c*，统计图表如下所示。

     
   - 通过单值图展示最近15分钟的PV数。

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

     配置**数值列**为*PV*，统计图表如下所示。

- 通过面积图展示最近1天某IP地址的访问情况。

  ```
  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
  ```

  配置**X轴**为*time*，**Y轴**为*PV*，统计图表如下所示。

  

- 通过地图展示访问前十的地域。
  - 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
    ```

    

- 通过流图展示最近15分钟不同方法（request_method）的请求次数随时间的变化趋势。

  ```
  * | 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
  ```

  配置**X轴**为*minute*，**Y轴**为*c*，**聚合列**为*request_method*，统计图表如下所示。

  

- 通过词云展示最近15分钟页面（request_uri）访问情况。

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

  配置**词列**为*c*，**数值列**为*c*，统计图表如下所示。

  

6. 添加统计图表到仪表盘。您可以单击**添加到仪表盘**，完成操作，详情请参见添加统计图表到仪表盘。

3.11. 分析负载均衡7层访问日志

本文档基于日志服务的可视化和实时查询分析能力，为您介绍负载均衡（SLB）7层访问日志查询分析的典型案例。

前提条件

已采集到SLB 7层负载均衡日志，详情请参见开通访问日志功能。

背景信息

对于大部分云上架构而言，负载均衡是基础设施组件，对SLB持续的监控、探测、诊断和报告是一个强需求。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，为应用提供大规模、高可靠的并发Web访问支撑。

SLB访问日志功能当前支持基于HTTP/HTTPS的7层负载均衡，访问日志内容丰富，完整字段说明请参见日志字段详情。SLB典型指标如下所示：

- PV：客户端发起HTTP、HTTPS请求的次数。
- UV：对于相同客户端只计算一次，合计总体请求次数。
- 请求成功率：状态码为2XX的请求次数占总PV的比例。
- 请求报文流量：客户端请求报文长度总和。
- 返回客户端流量：SLB返回给客户端的HTTP Body字节数总和。
- 请求的热点分布：统计客户端地理位置，按照地理位置统计每个地域的PV情况。

常用分析

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标Project。
3. 在**日志存储 > 日志库**中，单击目标Logstore左侧的>。
4. 在**可视化仪表盘**中，单击目标仪表盘。目标仪表盘包括slb-user-log-slb_layer7_operation_center_cn和slb-user-log-slb_layer7_access_center_cn。
   - 业务概览
     - 统计一定时间范围内用户请求的来源分布情况。
     - 通过过滤器筛选某指定SLB实例的PV、UV随时间的变化趋势。过滤器操作请参见添加过滤器。
   - 分析流量与延迟情况
     - 统计一定时间范围内请求报文的流量和返回客户端的流量。
     - 统计一定时间范围内请求的响应时间变化趋势和upstream响应时间变化趋势。
     - 统计一定时间范围内高延迟的请求。
   - 分析用户请求情况
     - 统计一定时间范围内的请求方法、请求协议分布情况。

- 统计一定时间范围内各种请求方法的PV趋势。

  

- 统计一定时间范围内服务运行情况。

  如果出现大量的500状态码则表示后端RealServer的应用程序发生内部错误。

  

- 统计一定时间范围内各种状态码的PV变化趋势。

  

- 分析请求源
  - 统计客户端所属的网络运营商分布情况。

    

  - 统计客户端所在的地理位置（国家、省份、城市）。

    

  - 查看用户代理信息。

    通过用户代理（http_user_agent）可得知哪些用户在访问网站或服务。例如搜索引擎会使用爬虫机器人扫描或下载网站资源，一般情况下低频爬虫访问可以帮助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，则可能影响服务性能及浪费机器资源。

    

- 运营概览

  运营人员可基于SLB访问日志分析流量情况，进而辅助业务决策。
  - 查看PV的热点分布，了解重点客户的分布及低PV地域分布，从而制定最优的推广方案。

    

  - 通过分析Host和URI信息获知访客最关注的内容，为网站内容建设提供有力的参考。

    

请求调度分析

客户端流量会先被SLB处理，分发到其中一台RealServer中进行实际的业务逻辑处理。SLB可自动检测到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重新分配流量。

为SLB实例添加一个监听，例如服务器（192.168.0.0）同时兼有跳板机职能，其性能是其它三台服务器的4倍，为该服务器设置监听权重为100，其余服务器监听权重为20。执行如下查询分析语句分析请求流量分布情况。

```
* | select COALESCE(client_ip, vip_addr, upstream_addr) as source, COALESCE(upstream_addr, vip_addr, client_ip) as dest, sum(request_length) as inflow group by grouping sets( (client_ip, vip_addr), (vip_addr, upstream_addr))
```

桑基图展示每台RealServer的负载情况，多个客户端向SLB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RealServer中。

3.12. 分页显示查询分析结果

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
while True:
   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：

  详细案例请参见Log Service Python SDK。

```
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：

  更详细的案例参见Log Service Java SDK。

```
    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),urlgroup 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;
  }

}
```

3.13. 分析-行车轨迹日志

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。

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。

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_surcharge: 0.3  mta_tax
: 0.5  passenger_count: 2  payment_type: 1  pickup_latitude: 40.761466979980469  pickup_longitude: -7
3.96246337890625  store_and_fwd_flag: N  tip_amount: 1.96  tolls_amount: 0  total_amount: 11.76  tp
ep_dropoff_datetime: 2016-02-14 11:03:13  tpep_dropoff_time: 1455418993  tpep_pickup_datetime: 2016
-02-14 10:53:57  tpep_pickup_time: 1455418437  trip_distance: 2.02
```

常见的统计

要对数据进行查询和分析，请先开启并配置索引。

- 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。

- 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。

- 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_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
```

```
*| 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。

- 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。

- 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。

3.14. 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），账单日志样例如下。

```
__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: 060f0e0d080e0b05060307010c0f0209010e0e010c0a0605000606050b0c0400 pay_with: alipay real_price: 52.0 suggest_price: 52.0
```

统计分析

要对数据进行查询和分析，请先开启并配置索引。

- 查看产品的销售占比。

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

- 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
```

- 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
```

4.消费

4.1. 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。

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。

4.2. 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）：

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

  i. 使用Logtail Agent实时采集

  ii. 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
```

- 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。

4.3. 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。

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，例如有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？

假设有10个柜员，我们可以创建10个Shard。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 落在一个特定Shard中（区间包含Z），处理端面对的一直是柜员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。例如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A回来后，发现张三请求没有记录下来，A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变得更复杂。

- 遇到发工资日子，用户数会大涨。
- 柜员（Clerk）毕竟不是机器人，也需要休假，需要吃午饭。
- 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？
- 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：

- 对Shard进行弹性伸缩。
- 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。
- 过程中不重复（需要消费者配合）。
- 观察到消费进度，以便合理调配计算资源。
- 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，请参见日志服务。

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）：由表格存储提供服务，从日志服务拉取数据，详情请参见日志数据传送。
- 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SLS+OSS+EMR/DLA+RAM组合，可轻松应对这类需求。

6.服务日志

6.1. 监控日志服务

日志服务提供服务日志功能，用于记录操作日志、消费组延迟日志、Logtail告警日志、Logtail采集日志和Logtail状态日志，帮助您实时掌握日志服务的使用状况，提高运维效率。

前提条件

已开通服务日志，详情请参见开通服务日志。

监控Logtail心跳

安装Logtail后，日志服务将Logtail状态日志保存在名为internal-diagnostic_log的Logstore中，您可以通过Logtail状态日志，检查Logtail的工作状态。

在internal-diagnostic_log Logstore查询分析页面，执行 `__topic__: logtail_status` 查询语句，即可查询Logtail状态日志，具体操作步骤请参见查询分析日志。例如执行如下语句查询Logtail状态日志及统计最近一段时间内心跳正常的机器个数，并配置告警。如果查询分析结果低于Logtail所绑定的机器组中的总机器数则触发告警。

- 查询语句

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

- 配置告警规则（此处假设总机器数为100）

  如果产生告警，您可以在日志服务控制台上查看机器组状态，排查心跳异常的机器。

查看消费组延时

在日志服务中，除了查询分析日志外，还可以通过消费组对日志进行消费，详情请参见通过消费组消费日志数据。

在使用消费组消费时，日志服务将消费延时日志保存在名为internal-diagnostic_log的Logstore中。您可以通过消费延时日志了解当前的消费进度，当延时较大时，可以及时调整消费者个数等方式来改进消费速度。

在internal-diagnostic_log Logstore查询分析页面，执行 `__topic__: consumergroup_log` 查询语句，即可查询消费组延时日志，具体操作步骤请参见查询分析日志。例如执行如下语句，查询test-consumer-group这个消费组的消费延时情况。

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

监控Logtail异常情况

安装Logtail后，日志服务将Logtail告警日志保存在名为internal-diagnostic_log的Logstore中，您可以通过Logtail告警日志，及时发现Logtail异常状况，调整Logtail配置，确保日志不丢失。

在internal-diagnostic_log Logstore查询分析页面，执行 `__topic__: logtail_alarm` 查询语句，即可查询Logtail告警日志，具体操作步骤请参见查询分析日志。例如执行如下语句，查询分析15分钟内各种异常类型发生的次数。

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

操作审计

日志服务Project下所有资源的操作日志保存在internal-operation_log Logstore中，可用于操作审计。每条操作日志不仅记录了操作相关的信息，例如创建机器组会记录机器组名称，操作Logstore会记录Logstore名称等，还记录了操作对应的用户信息。用户信息分类如下表所示：

类型	字段
主账户	• **InvokerUid**：阿里云主账户ID • **CallerType**：Parent
RAM用户	• **InvokerUid**：RAM用户的UID • **CallerType**：Subuser
Sts	• **InvokerUid**：阿里云主账户ID • **CallerType**：Sts • **RoleSessionName**：session名称

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通知方式。

- 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
}
```


字段解释可以参考告警日志字段。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，可以通过查看告警记录查看告警统计。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。

日志字段解释参考告警日志字段。

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。

8.企业上云实践

8.1. SLS多云日志采集、处理及分析

本文介绍如何采集、处理及分析多云日志。

概述

从第三方云平台或线下IDC服务器上采集日志写入到阿里云日志服务，通过日志服务的logsearch和Log Analytics进行数据分析，帮助提升运维、运营效率，建立DT时代海量日志处理能力。针对未使用其他日志采集服务的用户，推荐在其他云或线下服务器安装logtail采集并使用Https安全传输；针对已使用其他日志采集工具并且已有日志服务需要继续服务的用户，可以通过Log producer SDK写入日志服务。

适用场景

- 第三方云平台或线下IDC服务器需要使用阿里云日志服务。
- 第三方云平台或线下IDC服务器已有完整日志采集、处理及分析工具，但仍需要使用阿里云日志服务。

方案架构

推荐方案：采用第三方云平台或线下IDC服务器安装logtail，进行logtail配置，采用公网/专线+Https（DCDN）安全传输日志到日志服务。

针对在第三方云平台或线下IDC服务器已部署其他日志采集工具（如filebeat）的场景：filebeat采集日志写入kafka，spark streaming实时消费kafka消息，在作业中集成Log producer SDK通过公网/专线+Https安全传输（DCDN）写入日志服务。

- spark对数据ETL后可以写入SLS\RDS\ES\HBASE\ADB等产品，本例以SLS为例。
- 本示例使用ERM的kafka和spark集群模拟在第三方云或线下IDC使用开源生态自建kafka和spark集群。
- 本示例使用SNAT IP池让日志机器组集群具备主动访问公网能力，适合海量日志机器以及分布式多机房场景。

方案优势

- 以SLS的核心能力为切入点，从第三方云平台或线下IDC搬迁客户日志上阿里云。
- 融合阿里云的日志服务生态，增强用户粘性（如无缝对接Blink、Elasticsearch、 RDS、ADB、EMR、dataV等产品）。
- 与第三方开源生态无缝对接，在不侵入用户应用的情况下传输日志到SLS，降低用户使用门槛。
- 使用DCDN的安全加速保障用户数据安全、降低消费延时。
- 使用SANT IP池让日志采集机器组具备主动访问公网能力。

操作步骤

具体操作步骤请参见SLS多云日志采集、处理及分析。

8.2. 微服务架构日志采集运维管理

本文介绍部署了微服务应用的用户如何实现日志的采集、运维和管理。

概述

利用阿里云容器服务Kubernetes版、容器镜像服务、文件存储NAS、日志服务SLS、云数据库RDS、云服务器ECS等产品展示用户在应用微服务改造落地过程中对容器化应用/非容器化应用日志采集、运维分析的产品最佳实践能力。

场景描述

容器/Kubernetes技术在微服务落地过程中的部署、交付等环节给用户带来了很多便捷，也同时可能会存在容器化应用/非容器化应用混合部署的形态；在日志采集处理分析领域，会有一些主要的业务痛点。

方案架构

方案优势

- 基于阿里云容器服务Kubernetes托管版集群与文件存储NAS构建互联网微服务应用的高可用及高弹性架构。
- 容器镜像服务（ACR）提供自动化部署和更新能力，维护应用的最新状态。
- 数据通过RDS存储，提供读写分离容灾能力。
- SLS与阿里云产品深度集成，提供便捷一站式的日志采集、消费、分析和查询的日志运维能力。
- 容器化应用和非容器化应用混合部署，技术架构平滑演进。
- 阿里云智能DNS提供稳定易用的域名DNS解析能力。

操作步骤

具体操作步骤请参见微服务架构日志采集运维管理。