日志服务

最佳实践

最佳实践

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。

案例

- 数据采集：公网数据
- 数据清洗与 ETL
- 数仓对接

典型场景

- 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。

  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日志日志（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。

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：

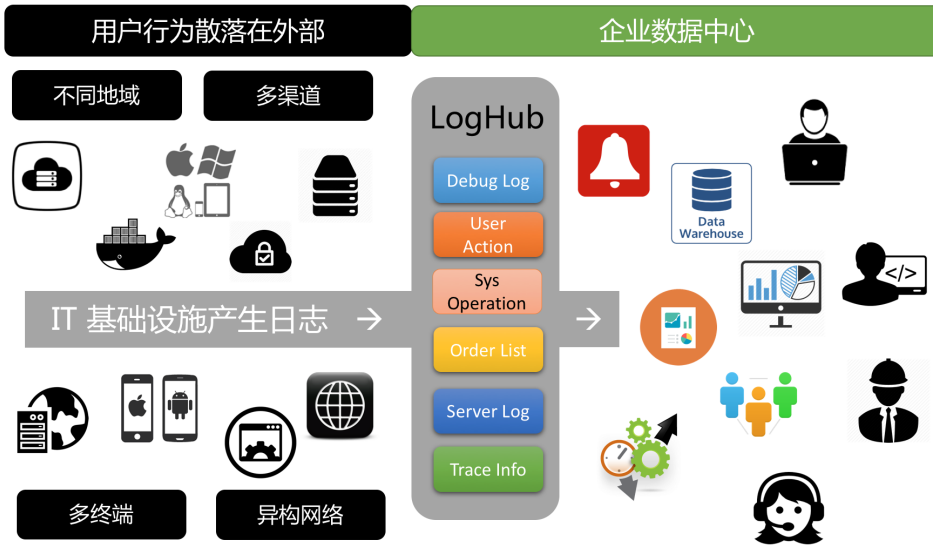
- 获取用户难，投放一笔不小的广告费到营销渠道（网页、微信推送），收获了一些用户，但无法评判各渠道的效果。
- 用户经常抱怨送货慢，但慢在什么环节，接单、配送还是加工，如何进行优化？
- 用户运营，经常搞一些优惠活动（发送优惠券），但无法获得效果。
- 调度问题，如何帮助商家在高峰时提前备货？如何调度更多的快递员到指定区域？
- 客服服务，用户反馈下单失败，用户背后的操作是什么？系统是否有错误？

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。
    - 参考 **数据清洗与 ETL**

更多操作可以参见 **快速开始/管理控制台**。

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?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:
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。
- Windows 服务器可以使用 logstash 采集。

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，参考 Tracking Web 接口。

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_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。

更多信息请参见 网络接入，总有一款适合您。

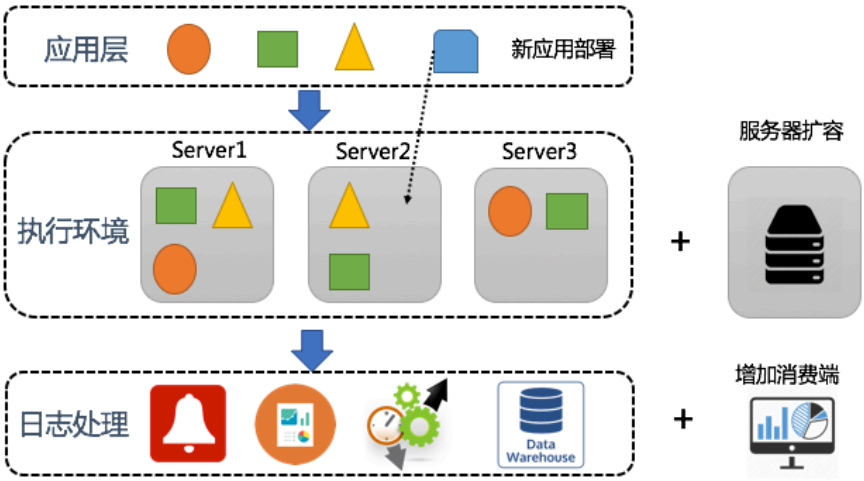
其他

- 参见 loghub完整采集方式。
- 参见 日志实时消费，涉及流计算、数据清洗、数据仓库和索引查询等功能。

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

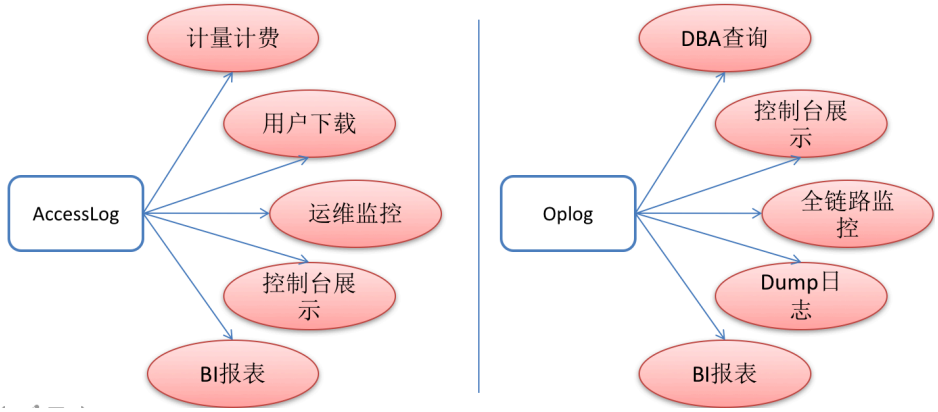
每个应用都包含访问日志（Access）、操作日志（OpLog）、业务逻辑（Logic）和错误（Error）等种类。当新增应用、应用与应用之间有依赖时，日志数目也会爆发。

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)
	nginx	alipay-nginx （支付宝服务端 nginx 日志)
	nginx	server-access（服务端 Access-Log）
Web-Error	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）
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，参见 服务简介

- 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。

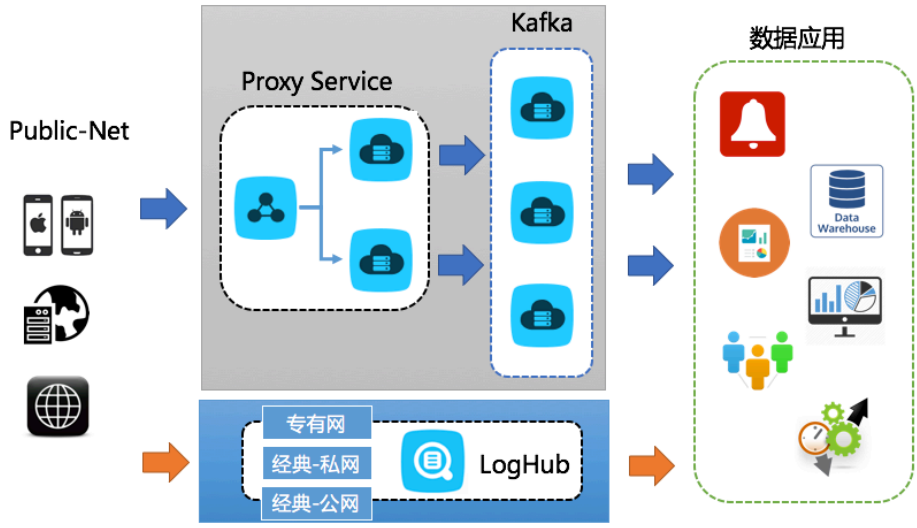
采集-公网数据

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

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

处理-数据清洗_ETL

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。

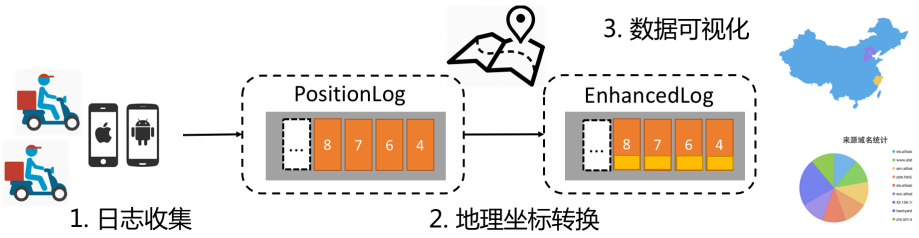
```
(GPS-X,GPS-Y) --> (GPS-X, GPS-Y, City, District, ZipCode)
```

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：

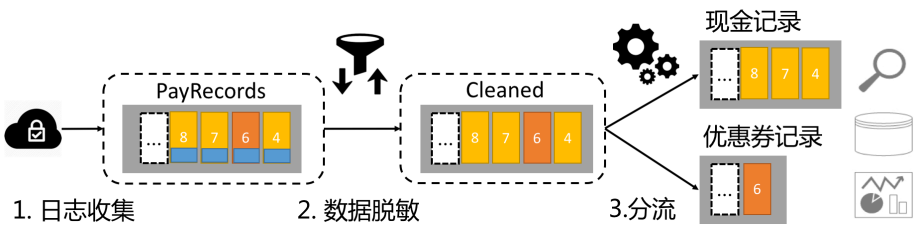
创建 4 个 logstore，原始数据（PayRecords），脱敏后数据（Cleaned），现金订单（Cash），代金券（Coupon）。应用程序通过 Log4J Appender 向原数据 logstore（PayRecords）写入订单数据。

推荐使用 Log4J Appender 或 Producer Library，敏感数据不落盘。

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，参见 RAM。

- 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。

基于日志服务的监控系统

日志服务简介

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。

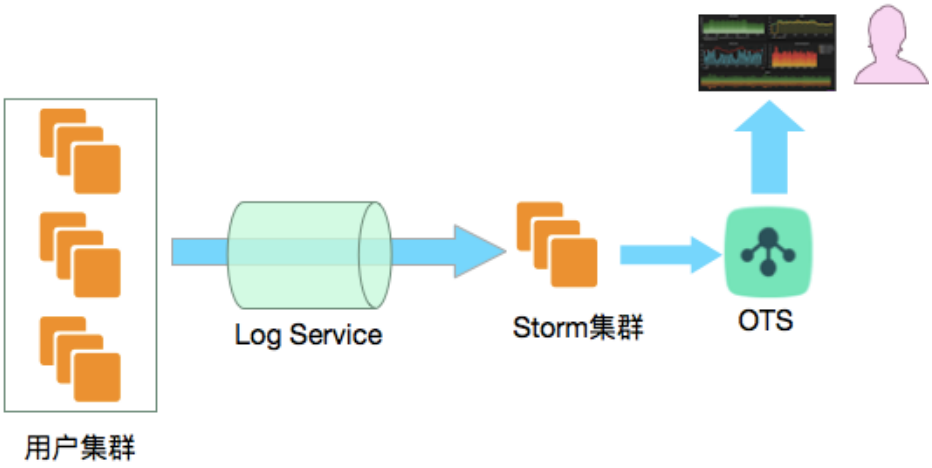
日志服务帮用户完成了日志收集，统一了日志格式，提供API用于下游消费。下游系统可以接入多重系统重复消费，比如导入Spark、Storm做实时计算，也可以导入elastic search做搜索，真正做到了一次收集，多次消费。在众多数据消费场景中，监控是最常见的一种场景。本文介绍阿里云基于日志服务的监控系统。

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：

1. 参考快速入门了解如何配置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

可以参考SDK使用方法，选择适合您的SDK版本。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。

处理-通过 ConsumerLib 实现不丢、保序、去重

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。

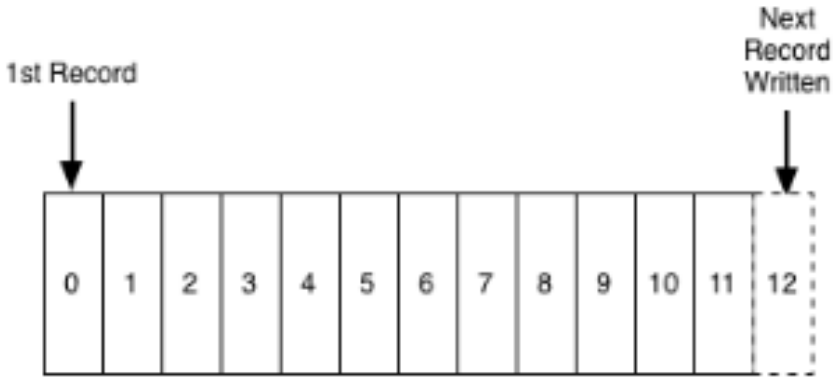
为方便理解，本文使用《银行的一天》作为例子将概念解释清楚。在文档末尾，介绍日志服务LogHub功能

，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：

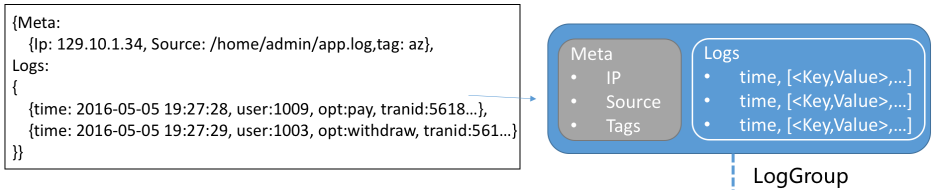
```
2016-06-28 08:00:00 张三 存款 1000元
2016-06-27 09:00:00 李四 取款 20000元
```

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。

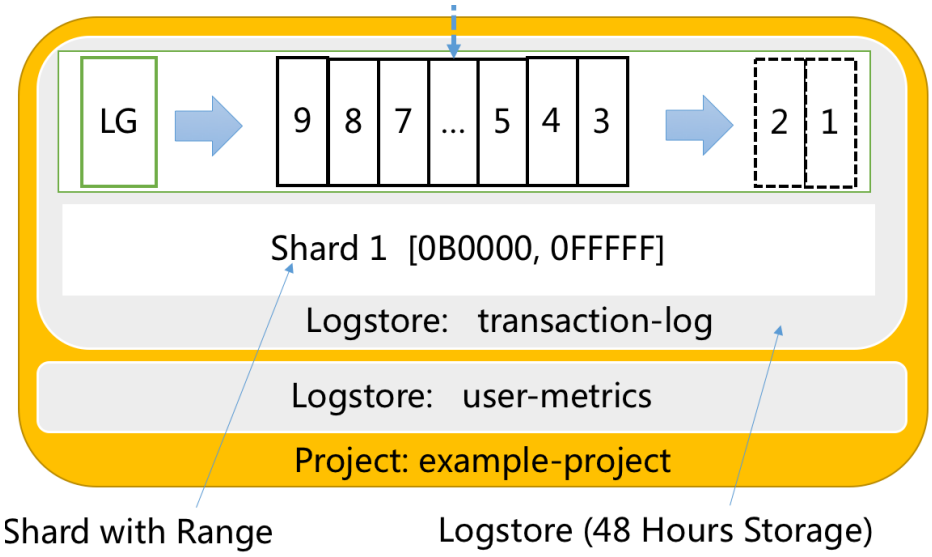
- Log: 由时间、及一组Key,Value对组成
- LogGroup: 一组日志的集合，包含相同Meta（IP，Source）等

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。

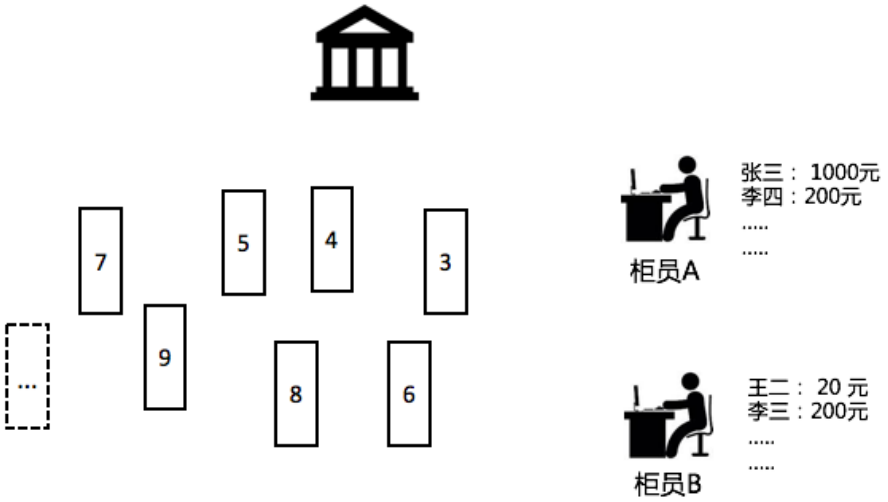
在分布式世界里，我们可以把柜员认为是固定内存和计算能力单机。用户是来自各个数据源的请求，Bank大厅是处理用户存取数据的日志库（Logstore）。

- 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。



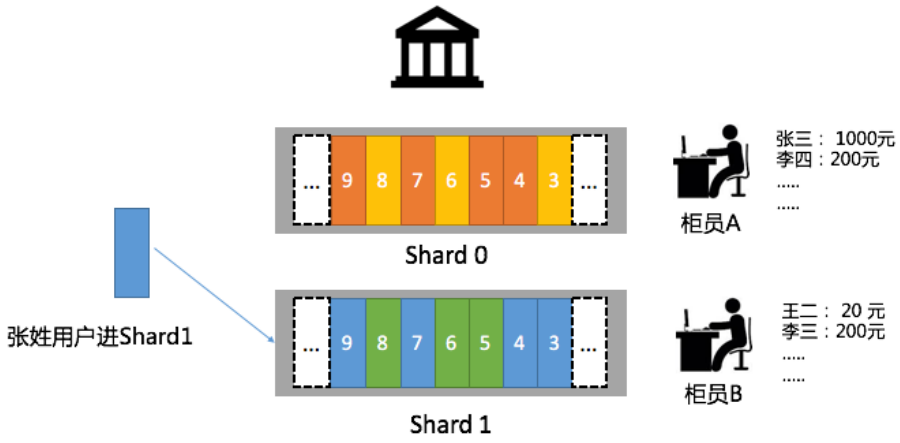
实现保序的方法很简单：排队，创建一个Shard,终端只有一个柜员A来处理。用户请求先进先出，一点问题都没有。但带来的问题是效率低下，假设有1000个用户来进行操作，即使有10个柜员也无济于事。

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。



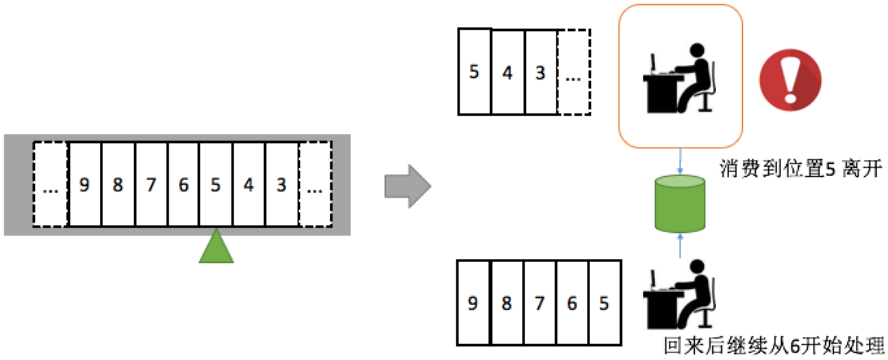
问题2：不丢失（At-Least Once）

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。但难免也会遇到当机、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）

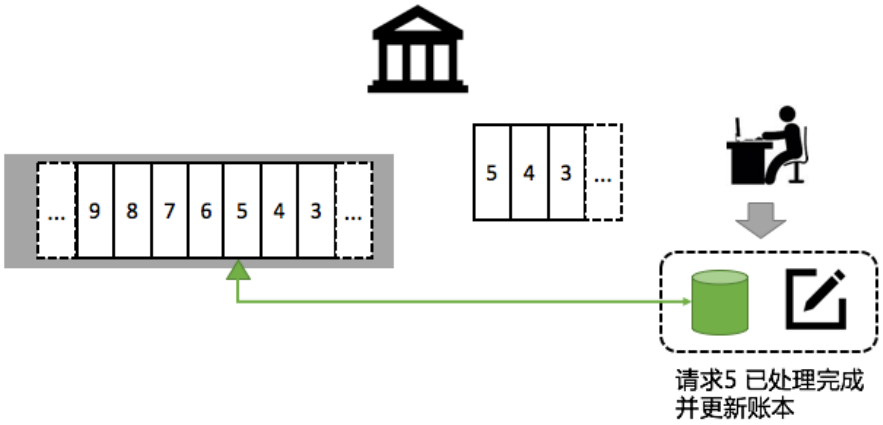
重复一定会带来问题吗？不一定。

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

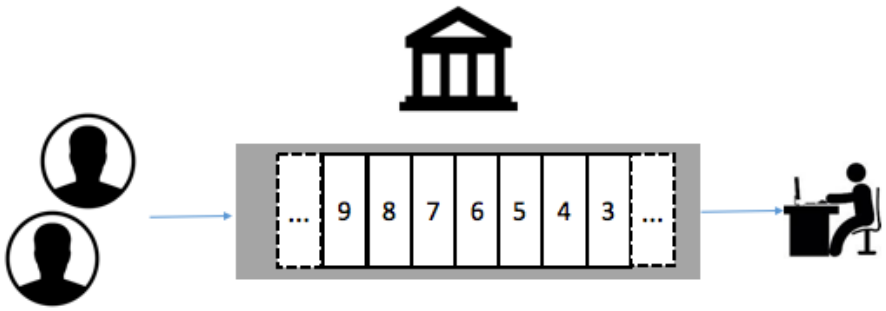
以上三个概念解释完成后，原理并不复杂。但在现实世界中，规模的变化与不确定性会使得以上三个问题便得更复杂。例如：

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

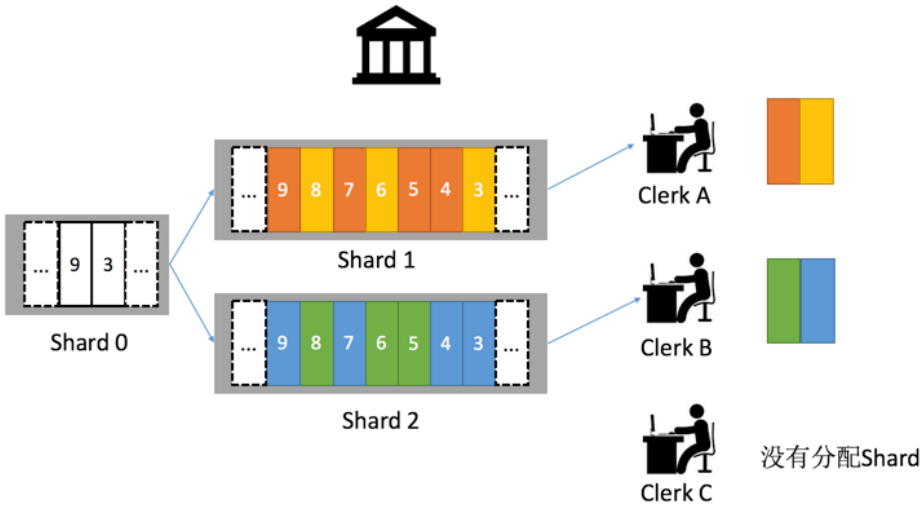
8点银行开门

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。

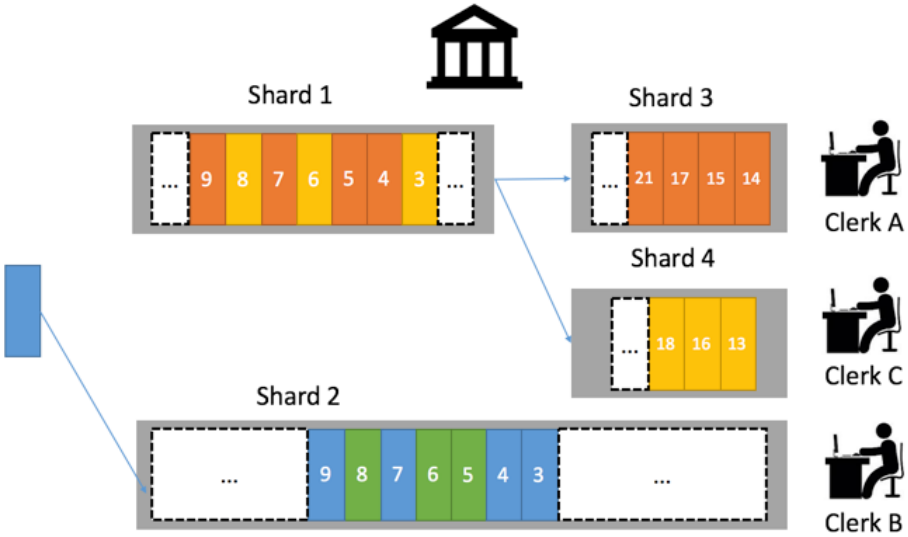


10-12点请求消费状态：

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。
3. 过程中支持保序。
4. 过程中不重复（需要消费者配合）。
5. 观察到消费进度，以便合理调配计算资源。
6. 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，欢迎试用。

许多个人站长选取了Nginx作为服务器搭建网站，在对网站访问情况进行分析时，需要对Nginx访问日志统计分析，从中获取网站的访问量、访问时段等访问情况。传统模式下利用CNZZ等方式，在前端页面插入js，用户访问的时候触发js，但仅能记录访问请求。或者利用流计算、离线统计分析Nginx的访问日志，但需要搭建一套环境，并且在实时性以及分析灵活性上难以平衡。

日志服务在查询功能的基础上支持SQL实时日志分析功能，极大的降低了Nginx访问日志的分析复杂度，可以

用于便捷统计网站的访问数据。本文档以分析Nginx访问日志为例，介绍SQL实时日志分析功能在分析Nginx访问日志场景下的详细步骤。

应用场景

个人站长选取Nginx作为服务器搭建了个人网站，需要通过分析Nginx访问日志来获取网站的PV、UV、热点页面、热点方法、错误请求、客户端类型和来源页面制表，以评估网站的访问情况。

日志格式

```
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" $http_host '
'$status $request_length $body_bytes_sent "$http_referer" '
'"$http_user_agent" $request_time';

access_log access.log main;
```

各字段含义如下。

字段	含义
remote_addr	客户端地址
remote_user	客户端用户名
time_local	服务器时间
request	请求内容，包括方法名，地址，和http协议
http_host	用户请求是使用的http地址
status	返回的http状态码
request_length	请求大小
body_bytes_sent	返回的大小
http_referer	来源页
http_user_agent	客户端名称
request_time	整体请求延时

配置步骤

1 采集Nginx服务器日志

将Nginx访问日志采集到日志服务。详细步骤请参考5分钟快速入门和Nginx日志。

2 建立索引查询分析

1. 在日志服务管理控制台，单击目标项目右侧的**管理**。
2. 选择目标日志库并单击日志索引列下的**查询**。
3. 单击右上角的**查询分析设置** > **开启索引**。

按照实际情况填写索引配置信息。Nginx日志请参考下图。

注意：其中request拆分为method和URL两列。

日志样例：

3 分析访问日志

开启索引后，在关键词索引框中填写统计语句，即可按照语句中定义的统计方式，对符合条件的检索结果进行统计分析。具体语句如下。

PV统计

PV统计不仅可以一段时间总的PV，还可以按照小的时间段，查看每个时间段的PV，比如每5分钟PV。

统计语句：

```
*|select from_unixtime( __time__- __time__% 300) as t,
count(1) as pv
group by t
order by t limit 60
```

UV统计

统计一小时内每5分钟的UV。

统计语句：

```
*|select from_unixtime( __time__- __time__% 300) as t,
approx_distinct(remote_addr) as uv
group by t
order by t limit 60
```

热点访问页面统计

统计最近一小时访问最多的10个页面。

统计语句：

```
*|select count(1) as pv, split_part(url,'?',1) as path  group by path order by pv desc limit 20
```

请求方法统计

统计最近一小时各种请求方法的占比。

统计语句：

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

http状态码统计

统计最近一小时各种http状态码的占比。统计语句：

统计语句：

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

客户端类型统计

统计最近一小时各种浏览器的占比。

统计语句：

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

来源页面统计

统计最近一小时referer来源于不同域名的占比。

统计语句：

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

各省份统计

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

4 访问诊断及优化

除了一些访问指标外，站长常常还需要对一些访问请求进行诊断，查看一下处理请求的延时如何，有哪些比较大的延时，哪些页面的延时比较大。

统计平均延时和最大延时

通过每5分钟的平均延时和最大延时，从整体上了解延时情况。

统计语句：

```
 *|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
limit 60
```

统计最大延时对应的请求页面

知道了最大延时之后，需要明确最大延时对应的请求页面是，以方便进一步优化页面响应。

统计语句：

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

统计请求延时的分布

统计网站的所有请求的延时的分布，把延时分布在十个桶里面，看每个延时区间的请求个数。

统计语句：

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

统计最大的十个延时

除最大的延时之外，还需要统计最大的十个延时及其对应值。

统计语句：

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

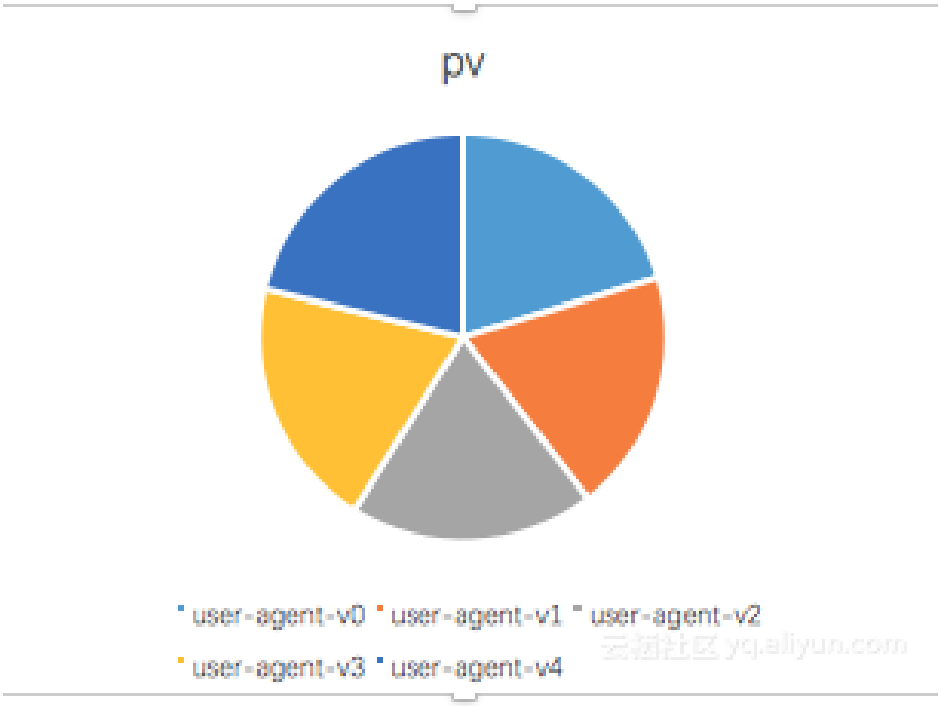
对延时最大的页面调优

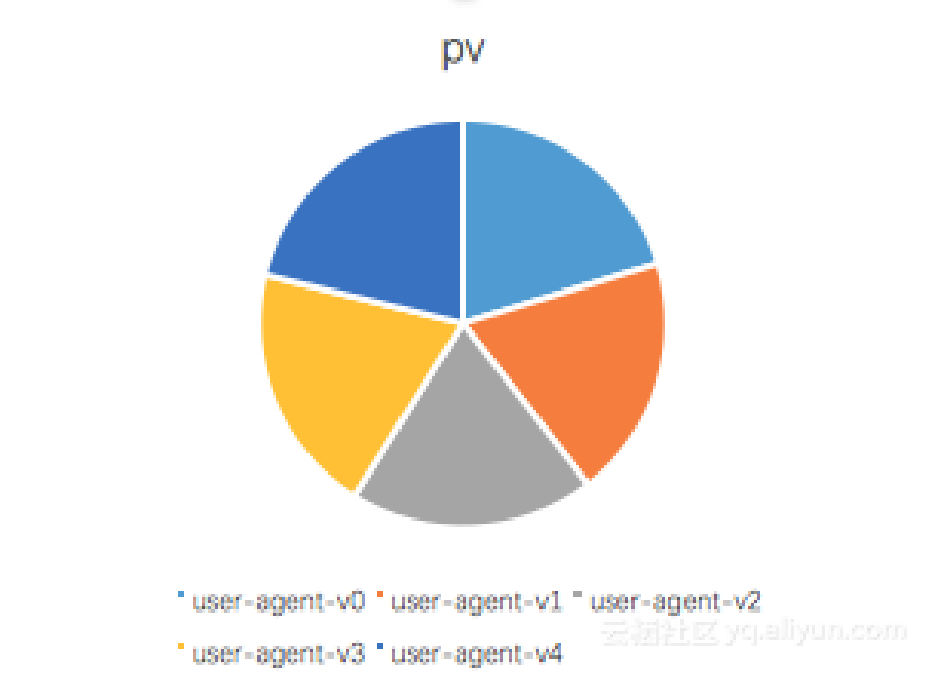
由统计结果可知，/0这个页面的访问延时最大，为了对/0页面进行调优，接下来需要统计/0这个页面的访问PV、U、各种method次数、各种status次数、各种浏览器次数、平均延时和最大延时。

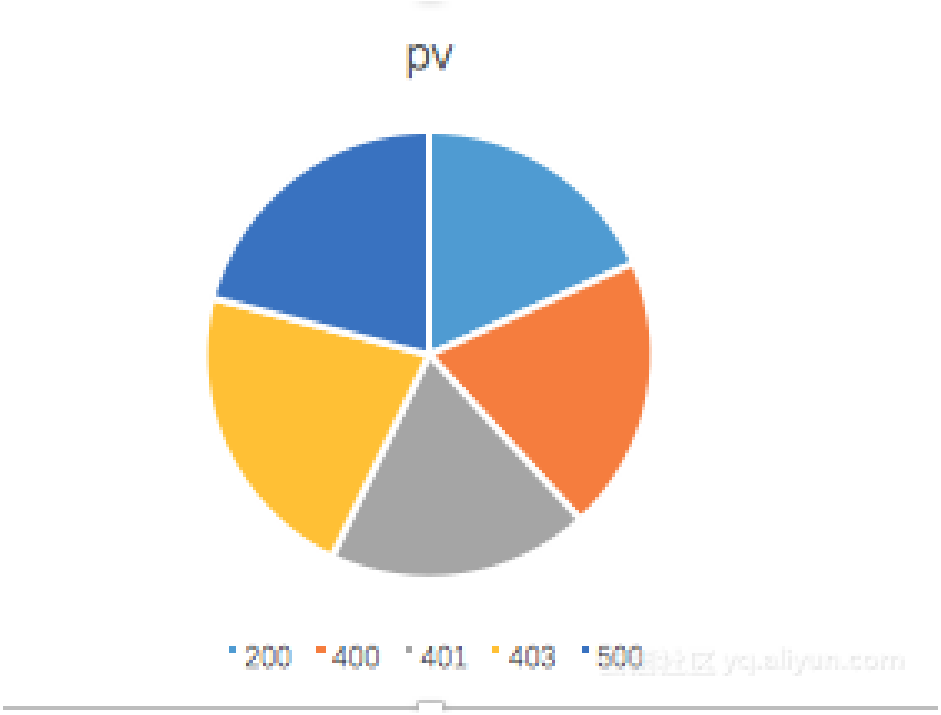
统计语句：

```
  url:"/0"|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
```

统计结果：


pv
user-agent-v0
user-agent-v1
user-agent-v2
user-agent-v3
user-agent-v4


pv
user-agent-v0
user-agent-v1
user-agent-v2
user-agent-v3
user-agent-v4



得到以上数据后，就可以对网站的访问情况进行有针对性的详细评估。

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：

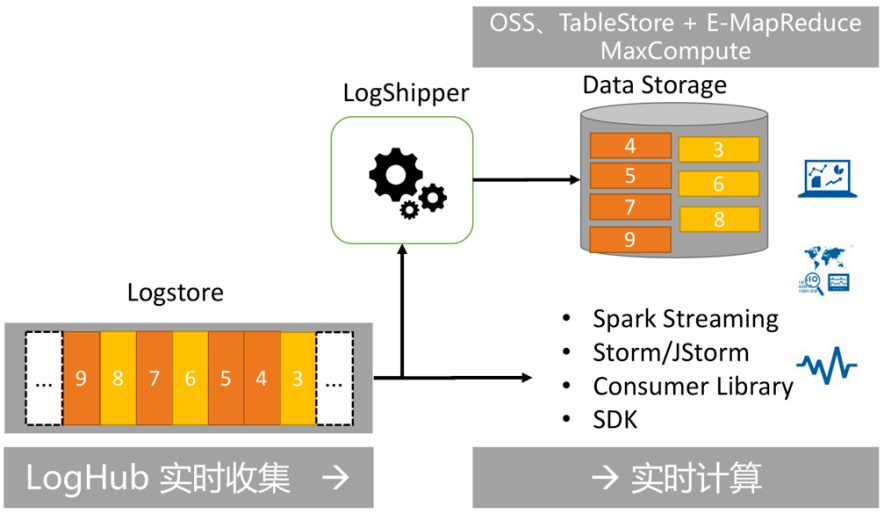
- Lamdba Architecture: 数据进来后，既支持流式处理、同时存入数仓。但用户发起查询时，会根据查询需求和复杂度从实时计算、离线计算拿结果返回。
- Kappa Architecture: kafka based Architecture。弱化离线计算部分，数据存储都在Kafka中，实时计算解决所有问题。

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）：
  - 说明文档
  - 操作步骤
  - OSS上格式可以通过Hive处理，推荐E-MapReduce
- Table Store（NoSQL数据存储服务）：
  - 操作步骤



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。

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。他的需求如下：

- 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+RAM组合，可轻松应对这类需求。