日志服务

最佳实践

最佳实践

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、ODPS 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）。

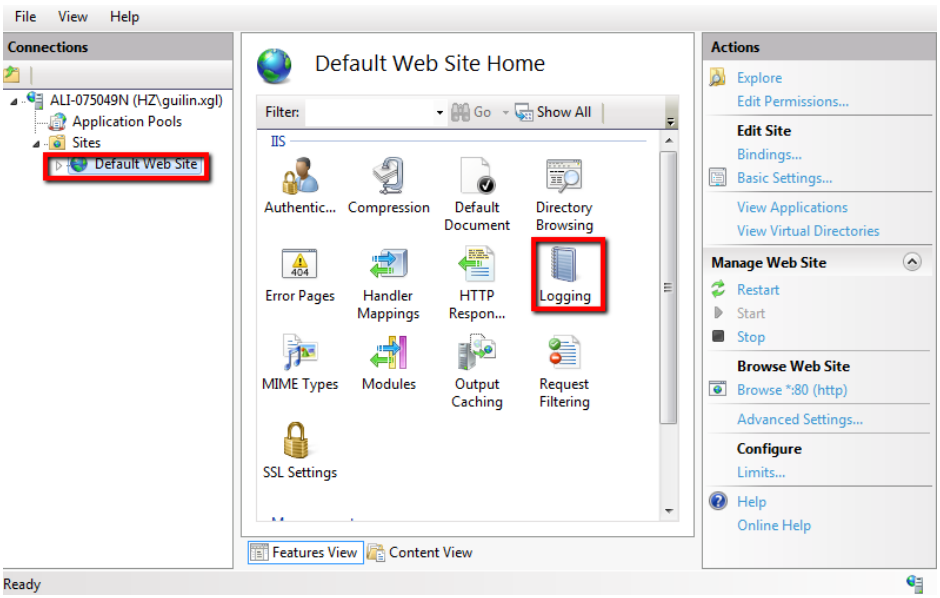
- 通过不同的日志库、机器组对应用程序进行集中式管理。
- 对不同日志进行处理。例如，访问日志进行流计算做实时监控；对操作日志进行索引、实时查询；对重要日志进行离线存档。
- 日志服务提供全套 API 进行配置管理和集成。
- 适用领域：有较多服务器需要管理的用户。

- 其它

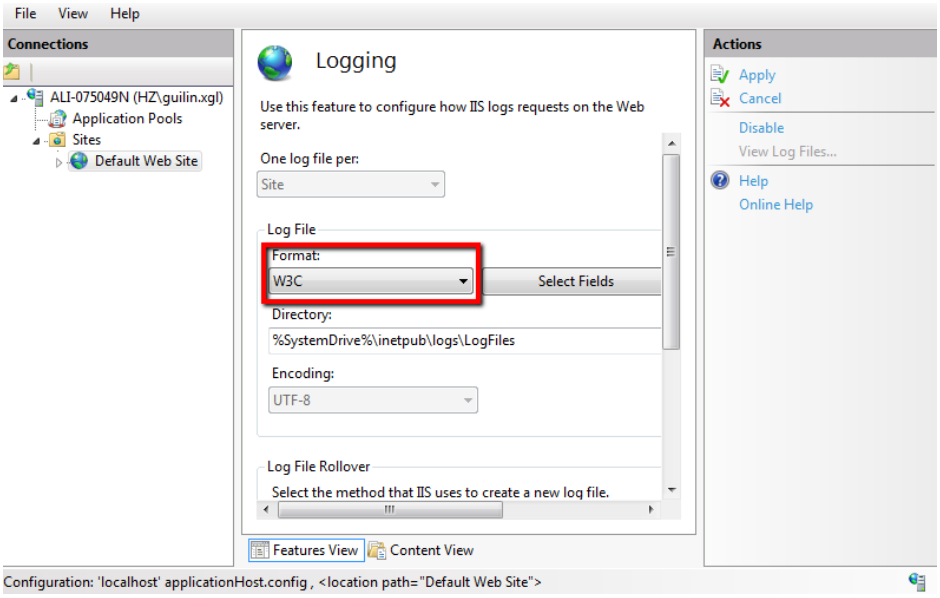
- 计量计费、业务系统监控、漏洞检测、运营分析、移动客户端分析等。在阿里云内部，日志服务无处不在，几乎所有云产品都在使用日志服务解决日志处理、分析等问题。

为收集 IIS 完整的访问日志（Access Log），您需要进行如下配置：

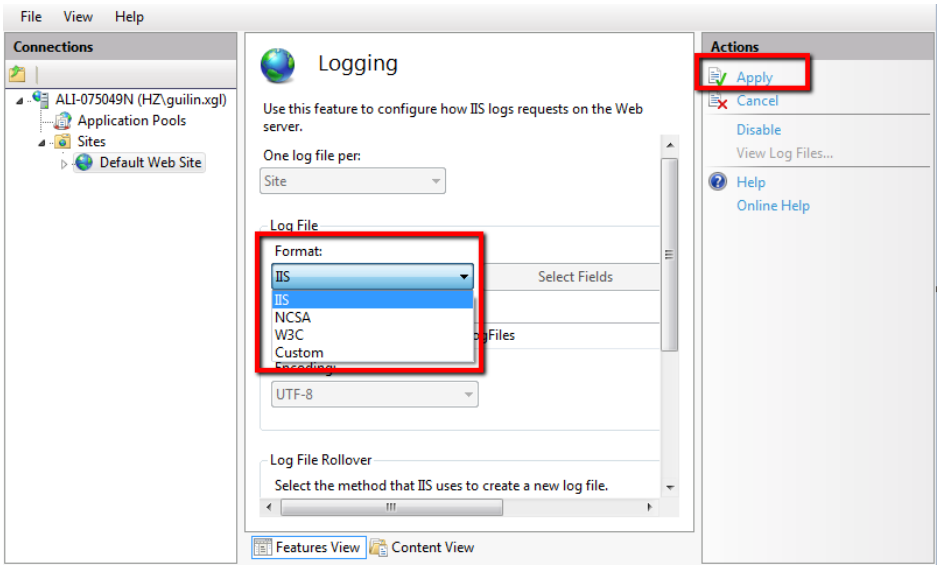
第一步：打开您的 IIS 管理器，并选中需要收集的网站



第二步：单击上图中的 Logging 进入日志配置页面



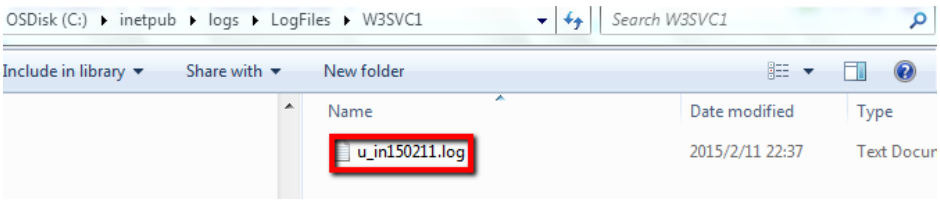
第三步：调整日志文件格式为 IIS 并单击 Apply



IIS 访问日志默认使用 W3C 格式。但是由于 W3C 格式产生的日志时间戳以格林威治时区（即零时区）计时，和系统的本地时间（阿里云国内数据中心为北京时间）不一致，导致日志服务日志收集客户端认为实时产生日志为历史数据，拒绝收集。我们会尽快支持 IIS 的 W3C 格式日期，请关注我们的服务更新。

第四步：主动访问上面设置的网站，让其产生日志数据

在访问完网站后，您可以到配置的日志文件生成目录（默认设置根目录为 C:\inetpub\logs\LogFiles，日志文件会生成在其子目录中）中查看生成的日志，如下图所示：



由于 IIS 日志生成后可能会缓存 1 分钟后才更新到磁盘，所以请在访问完网站后耐心等待一会。另外，如果在文件浏览器中没有看到文件，请反复刷新相应目录，直到出现日志文件。

第五步：使用 logstash 收集日志内容

您可以参考 通过 logstash 收集 Windows 平台日志 了解如何安装 logstash，具体配置样例可以参考 logstash 收集 IIS 日志。

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：

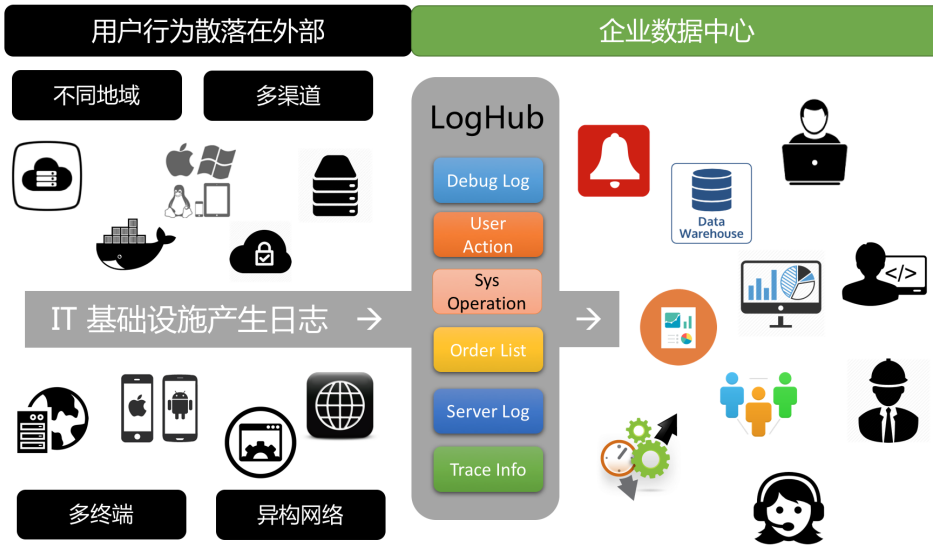
- 获取用户难，投放一笔不小的广告费到营销渠道（网页、微信推送），收获了一些用户，但无法评判各渠道的效果。
- 用户经常抱怨送货慢，但慢在什么环节，接单、配送还是加工，如何进行优化？
- 用户运营，经常搞一些优惠活动（发送优惠券），但无法获得效果。
- 调度问题，如何帮助商家在高峰时提前备货？如何调度更多的快递员到指定区域？
- 客服服务，用户反馈下单失败，用户背后的操作是什么？系统是否有错误？

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。

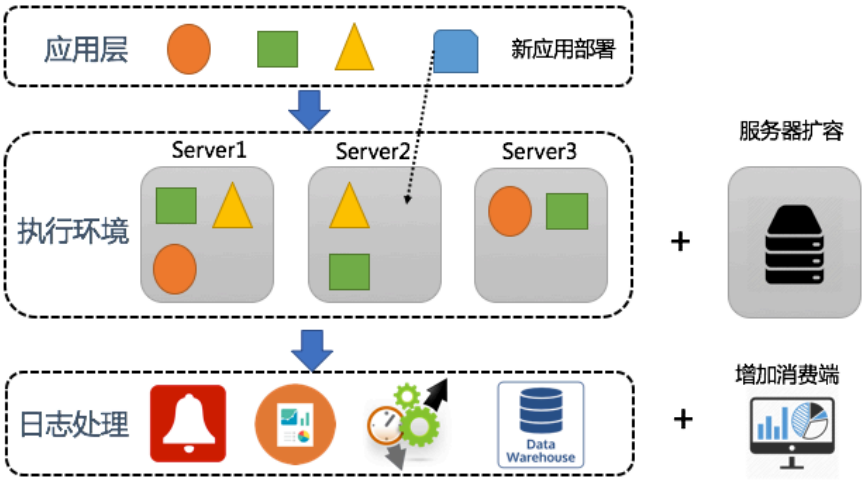
其他

- 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

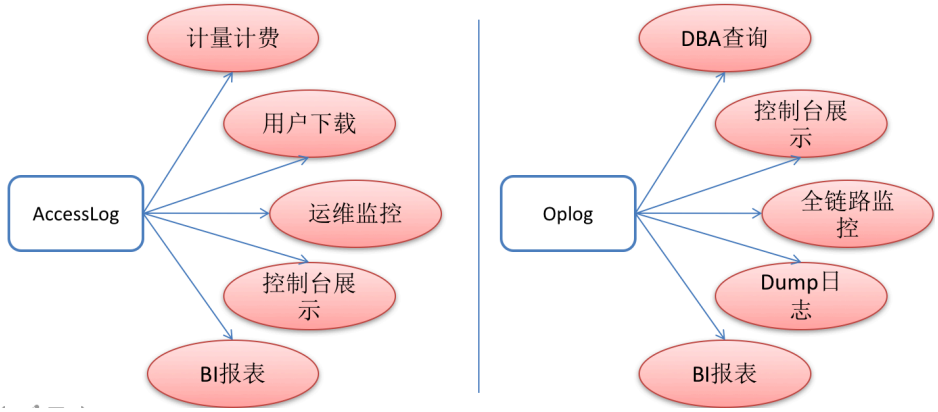
每个应用都包含访问日志（Access）、操作日志（OpLog）、业务逻辑（Logic）和错误（Error）等种类。当新增应用、应用与应用之间有依赖时，日志数目也会爆发。

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。

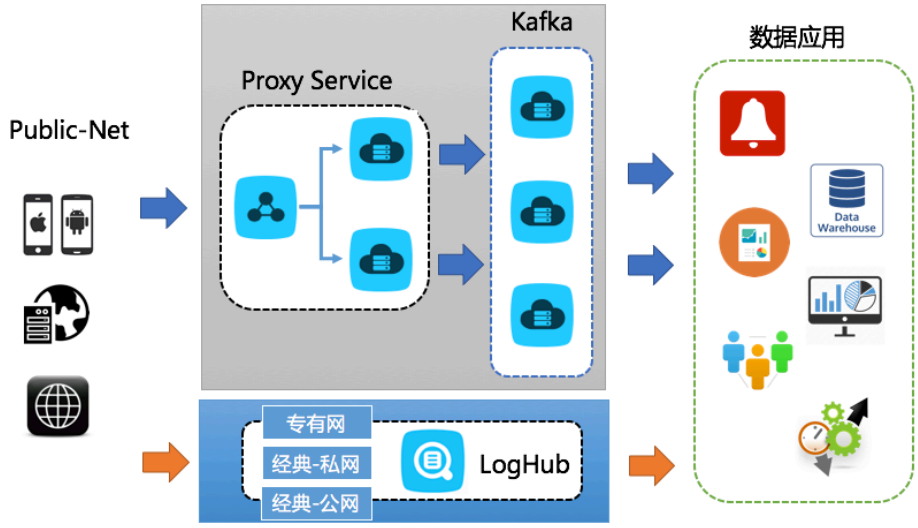
  日志查询（logsearch）：实时索引、查询数据。日志统一查询，不用关心线上服务器日志位置。

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

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

场景

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



方案 1：前端服务器 + Kafka

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

需要的设施为：

设施	数目	配置	作用	价格

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

方案 2：使用 loghub

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

需要的设施为：

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

场景对比

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

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

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

场景 2：一天 1TB 数据采集，大约一亿次写请求。

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

方案 2：
```

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

方案比较

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

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

有兴趣可以参见 日志服务主页 体验该服务。

第一步：创建 VPC 与 ECS 实例

参考 专有网络 VPC 用户文档 操作，创建 VPC 与 ECS 实例如下：

第二步：在 ECS 机器上安装 logtail

登录机器 192.168.1.2，安装 logtail，选择写数据到日志服务深圳 Region：

```
wget http://logtail-release.vpc100-oss-cn-hangzhou.aliyuncs.com/linux64/logtail.sh
chmod 755 logtail.sh
./logtail.sh install cn_shenzhen_vpc
```

详细信息请参考日志服务 安装 logtail。

第三步：到日志服务控制台进行配置

创建 logstore 及 logtail 收集配置：

填写文件目录、名称以及日志样例：

创建机器组，将 192.168.1.2 加入机器组：

将配置应用到机器组，等待约 3 分钟后开始验证。

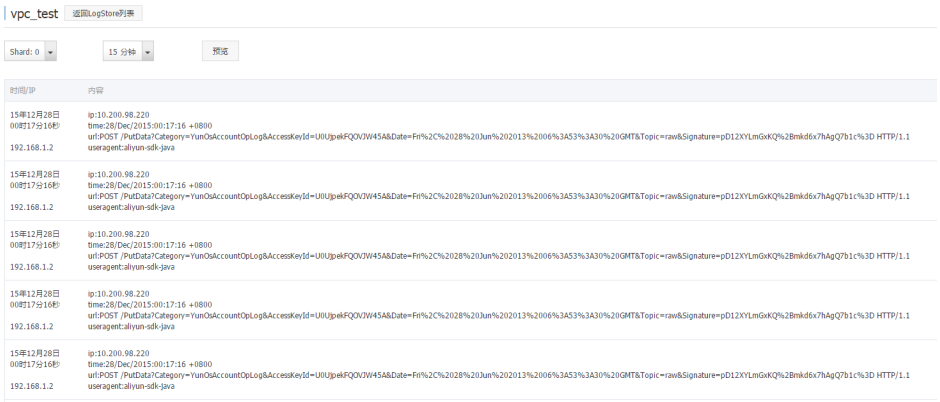
第四步：logtail 心跳状态验证

打开机器组，查看心跳状态：

如果心跳异常，请参考日志服务 为什么我的 logtail 心跳状态不正常 排查原因。

第五步：数据收集验证

当日志文件在持续追加产生时，查看 logstore，预览 logtail 收集到的日志数据。如下所示：



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。

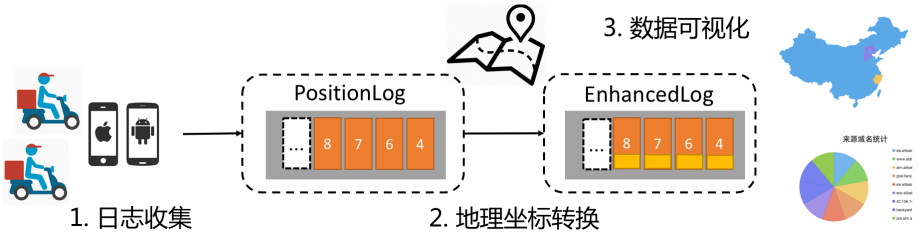
```
(GPS-X,GPS-Y) --> (GPS-X, GPS-Y, City, District, ZipCode)
```

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：

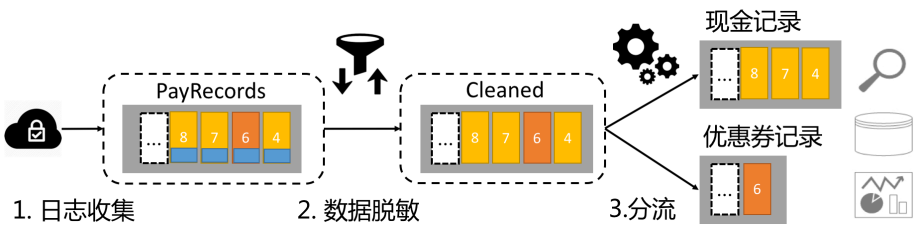
创建 4 个 logstore，原始数据（PayRecords），脱敏后数据（Cleaned），现金订单（Cash），代金券（Coupon）。应用程序通过 Log4J Appender 向原数据 logstore（PayRecords）写入订单数据。

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

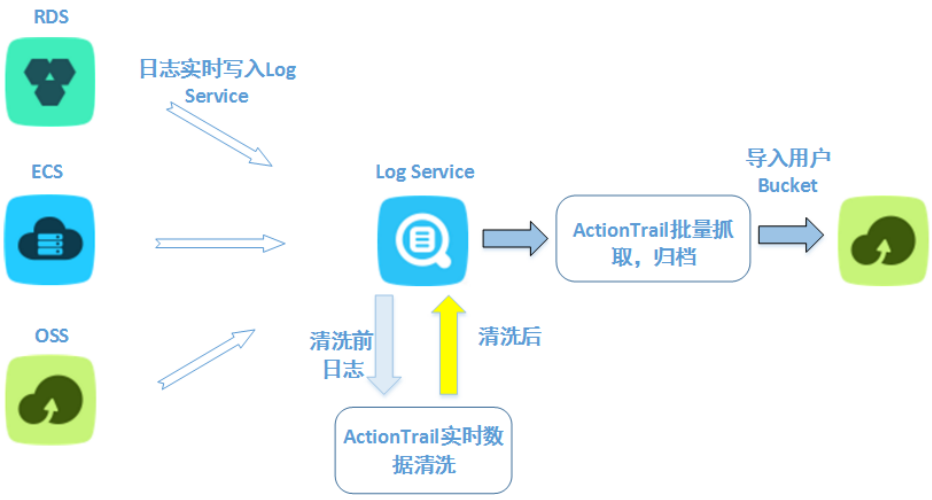
- loghub 功能下每个 logstore 可以通过 RAM 进行账号级权限控制，参见 RAM。

- 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。

Action Trail 是阿里云提供的一款云产品，用于记录用户账号各类 API 调用操作。Action Trail 记录了每次 API 调用的重要信息，如操作者、操作时间、对象、动作类型、来源等详尽的信息。这些调用记录，可以精确追踪、还原用户行为，对于安全分析，资源变更追查，合规审查有非常重要的作用。

日志服务数据可以通过Action Trail对采集到的多来源日志进行全方位的信息审计，达到日志审计的目的。

功能架构



1. Action Trail 接入日志服务

各类API调用的日志分散在各处，可以通过日志服务批量收集散落在各地的日志。各个应用通过Logtail，将每条日志解析为 key:value 式的半结构化形式，并上传至日志服务中应用各自的Project。然后通过 RAM 授权 Action Trail 读取这部分日志。只要约定 Action Trail 需要的key，Action Trail就可以直接从日志服务消费各应用的 API 调用日志了。

2. Action Trail 批量处理各类日志

各个应用所产生的日志类型和日志格式各不相同，但这些日志经由日志服务收集到服务端后，以半结构化格式储存。Action Trail 对于半结构化格式的日志可以做到批量处理、分析。Action Trail 实时地从日志服务中抓取各应用的日志，提取必要的字段，并将剩余的字段统一打包在一起。清洗完毕的数据，Action Trail 也记录在本地，通过 Logtail 实时上传至日志服务中 Action Trail 的 Project 中。

下图就是经过清洗处理后的一条日志，该日志记录了一个用户在 17:53 重启了一个虚拟机。

account: [redacted]
event_id [redacted]
event_name:RebootInstance
event_type:ApiCall
extra:eyJhcGlWZXJzaW9uIjoiMjAxNC0wNS0yNiIsImV2ZW50U291cmNlIjoiZWNzLWNuLWhhbmd6aG91
lkVDUyI6WyJBWTE0MDQyODA5MjcwNjEyNGI0OCJdfX0sInJlcXVlc3RQYXJhbWV0ZXJzIjp7IkZvcmNlU
2VjdXJlVHJhbnNwb3J0IjoidHJ1ZSJ9LCJzb3VyY2VJcEFkZHJlc3MiOiIxMTIuNS4yMzYuMTgiLCJ1c2VyQ
bzFUN0RoaHlYMlJqcyIsImFjY291bnRJZCI6IjE0ODM3Njg2OTY5NjAxOTciLCJwcmluY2lwYWxJZCI6IjE0
XRlcyI6eyJjcmVhdGlvbkRhdGUiOiIyMDE1LTExLTA0VDA5OjUzOjM5WiIsIm1mYUF1dGhlbnRpY2F0ZW
from:pop
model:Action
region: [redacted]
request_id [redacted]
resource_name [redacted]
resource_type:ECS
service_name:Ecs
stamp:2015-11-04T09:53:39Z
time:2015-11-04 17:53:59
to:cn-hangzhou
user: [redacted]
version:1

3. 数据存储

Action Trail 定期从日志服务中批量读取处理后的日志，根据用户账号和应用类型进行分类，上传至用户授权的 bucket 中，以供用户直接从自己的 OSS bucket 中读取这部分日志。

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。

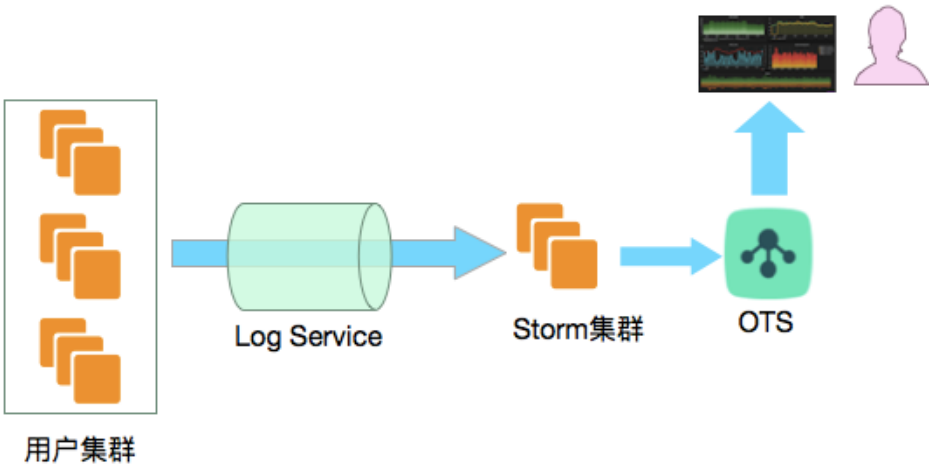
日志服务把所有集群的监控数据作为日志收集到服务端，解决了多集群管理和异构系统日志收集的问题，监控数据统一成格式化的日志发送到日志服务.

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。

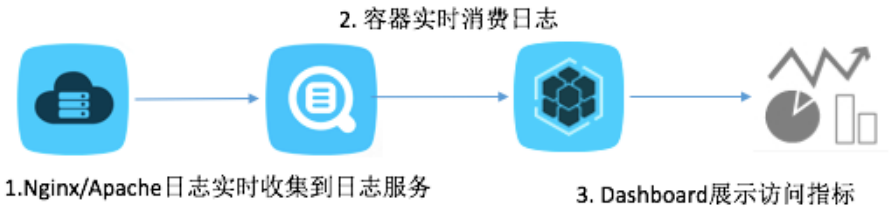
4. 展示监控信息

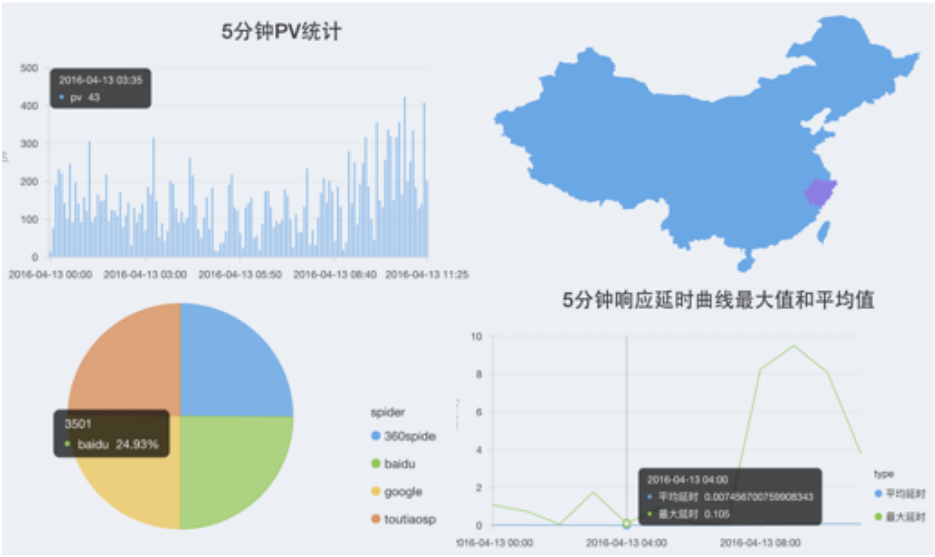
通过读取保存在OTS中的监控数据，在前端展示；或者读取数据，根据数据结果做报警。

网站访问日志统计分析介绍

场景

用户使用ECS搭建网站，网站的访问日志(Nginx,Apache访问日志)收集到阿里云日志服务中供查询。为了从访问日志中挖掘出更多价值，日志服务提供了一个docker镜像，用于实时统计和展示网站访问的一系列指标,例如PV、UV、延时、地理、状态码、爬虫、网络流量等指标。





日志字段

```
192.168.1.101 - - [17/Mar/2016:10:28:30 +0800] "GET /fonts/fontawesome-webfont.woff?v=4.2.0 HTTP/1.1" 0.021 1207 304 0 "https://sls.console.aliyun.com/css/lib.css" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.87 Safari/537.36"
```

以上述日志为例，分别提取的字段如下表所示。

字段名	字段样例
ip	192.168.1.101
method	GET

path	/fonts/fontawesome-webfont.woff?v=4.2.0
latency	0.021
request_length	1207
status	304
response_length	0
referer	https://sls.console.aliyun.com/css/lib.css
user_agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.87 Safari/537.36

指标

PV

分别以5分钟、1小时、1天为统计周期，统计每个周期内的日志总数。

UV

分别以1小时、1天为统计周期，统计每个周期内的IP总数。

页面

以天为统计周期，统计访问最多的页面路径，以及访问最多的参数。例如请求/fonts/fontawesome-webfont.woff?v=4.2.0,提取出页面/fonts/fontawesome-webfont.woff 和参数v = 4.2.0

方法

方法指的是HTTP方法，包括GET，POST，DELETE，PUT等。以小时和天为统计周期，统计每个周期内每个方法的日志条数。

地理

统计每个IP所属的省份，展示所选时间段内每个省份的分布图。

状态码

状态码指的是HTTP状态码，包括200，401，403，500等常见状态码。以小时和天为统计周期，统计每个周期内的状态码次数。

浏览器

浏览器分为多个子指标，分别统计每一个子指标出现的PV、UV。包括以下内容：

- 终端类型
  - 移动终端
  - 非移动终端

  浏览器类型

  - chrome
  - safari
  - IE
  - firefox

  操作系统

  - mac
  - window
  - linux

  浏览器内核

  - webkit
  - gecko

爬虫

统计常见的爬虫访问量。常见爬虫包括百度、Google、360、今日头条。

来源页

根据referer统计的来源域名，统计来源最高的20个域名。

延时

- 统计每5分钟内的网络请求的延时的平均值和最大值。
- 统计每天分布最多的延时的分布情况。出现次数较少的延时区间不会加入统计，比如一天内只有一次延时为8s，大部分的延时都在0.3s 到 0.5s之间，那么只会统计0.3->0.4, 0.4->0.5的延时分布。
- 统计每个小时延时最大的日志。

流量

以小时为单位，根据request_length字段和response_length字段，统计访问的入网流量和出网流量的大小。

和Google Analytics的比较

	Google Analytics	基于日志服务的访问统计

实现方式	在浏览器端加JS，用户访问时触发统计	根据服务端访问日志统计
是否能看到爬虫信息	否	是
能否看到请求延时	否	是

开始使用

请参考使用文档。

网站访问日志统计分析

使用视频

如何使用网站访问日志分析请参照视频。

Demo地址

必要条件

Nginx/Apache访问日志必须接入到阿里云日志服务

开通日志服务

开通容器服务

有阿里云AccessKey可以访问日志服务(可以是子帐号)

子帐号权限

如果您使用子帐号授权，请参考下边的权限配置，将${your_project}替换成你的Project名称，${your_logstore}替换成您的访问日志所在的Logstore。

```
{
"Version": "1",
"Statement": [
```

```
{
"Action": [
"log:Get*",
"log:List*"
],
"Resource": "acs:log:*:*:project/${your_project}/logstore/${your_logstore}",
"Effect": "Allow"
},
{
"Action": [
"log:CreateConsumerGroup",
"log:ListConsumerGroup",
"log:ConsumerGroupUpdateCheckPoint",
"log:ConsumerGroupHeartBeat",
"log:GetConsumerGroupCheckPoint"
],
"Resource": "acs:log:*:*:project/${your_project}/logstore/${your_logstore}/consumergroup/*",
"Effect": "Allow"
}
]
}
```

非必要条件

- 自建 MySQL 或者 RDS（默认使用docker镜像内的MySQL）

实施步骤

实施步骤以Ubuntu系统为例，其他系统请以本文档做参考

访问日志接入到阿里云日志服务，具体接入方法请参考阿里云日志服务文档。

开通**容器服务**。

在容器服务中**创建集群**，操作系统选择Ubuntu。

在容器服务中**创建应用**。

i. 应用名称输入dashboard（或自定义）。
ii. 部署集群选择刚刚创建的集群。
iii. 单击 **使用镜像创建**。
iv. 单击**选择镜像**，选择阿里云镜像，access_log_stat_dashboard镜像。选中出现的aliyunlog/access_log_stat_dashboard镜像。
v. 在**web路由规则**中，容器端口输入80，域名输入dashboard，单击 **添加**。
vi. 单击**确定**。

在【服务】中，找到刚刚创建的服务（dashboard，或自定义服务名称），点击服务名称，在出现

的基本信息中找到访问端点，例如访问端点。

6. 在浏览器中打开上述URL，开始使用。
7. 首次打开dashboard，需要使用日志服务的帐号信息登录，包括region,project ,AccessId, AccessKey。
8. 登录完成后，首次使用dashboard要求配置：
    i. 日志信息。包括region,project ,AccessId, AccessKey，LogStore。比登录信息多了一个LogStore。
    ii. 日志内容字段映射，docker镜像为使用一些默认的名称来描述访问日志的一些字段，如果您在接入日志服务时使用不同的字段名称，请在这里做字段映射，保证您的字段的含义能够被分析程序识别。例如latency字段，假如您接入日志服务时配置该字段的名称为request_time，那么需要在这里填写request_time。一段日志样例：

```
192.168.1.101 - - [17/Mar/2016:10:28:30 +0800] "GET /fonts/fontawesome-webfont.woff?v=4.2.0 HTTP/1.1" 0.021 1207 304 0 "https://sls.console.aliyun.com/css/lib.css" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.87 Safari/537.36"
```

后台处理key	字段样例
ip	192.168.1.101
method	GET
path	/fonts/fontawesome-webfont.woff?v=4.2.0
latency	0.021
request_length	1207
status	304
response_length	0
referer	https://sls.console.aliyun.com/css/lib.css
user_agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.87 Safari/537.36

高级选项

镜像计算结果的数据保存在镜像的 MySQL 中,如果您释放您的容器，那么历史计算结果会丢失，为了保证所有的历史结果，请您使用自己的 MySQL：

创建一个MySQL用户，允许这个用户从docker中访问这个MySQL。

```
 grant all privileges on *.* to loguser@'%' identified by '123456789';
flush privileges;
```

上述用户名和密码根据自己的需求自定义

- 修改/etc/mysql/my.conf ,注释掉bind-address 0.0.0.0这一行

重启mysql

```
sudo service mysql restart
```

点击dashboard页面上方的 **计算结果临时保存在docker容器中，若需永久保存，请更改数据库**，进入配置MySQL。填写MySQL的地址和账户信息，单击 **迁移。**

使用ECS启动docker

上文讲述了如何使用容器服务来启动Docker，对于购买了ECS的用户而言，可以使用自己的虚拟机来启动Docker。在ECS上启动镜像，请执行下边的命令：

```
pull registry.aliyuncs.com/aliyunlog/access_log_stat_dashboard

docker run --dns 223.5.5.5 -p 80:80 -d registry.aliyuncs.com/aliyunlog/access_log_stat_dashboard
```

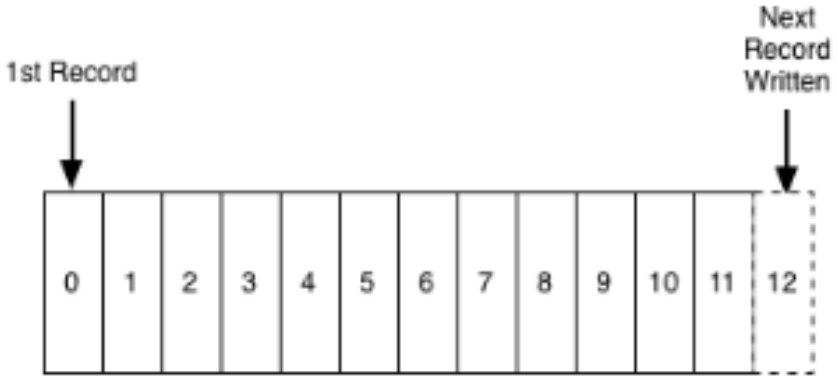
日志处理是一个很大范畴，其中包括实时计算、数据仓库、离线计算等众多点。这篇文章主要介绍在实时计算场景中，如何能做到日志处理保序、不丢失、不重复，并且在上下游业务系统不可靠（存在故障）、业务流量剧烈波动情况下，如何保持这三点。

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：

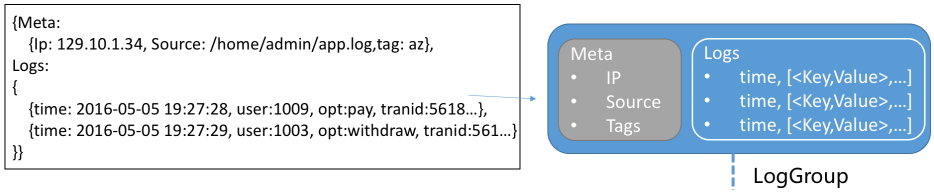
```
2016-06-28 08:00:00 张三 存款 1000元
2016-06-27 09:00:00 李四 取款 20000元
```

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。

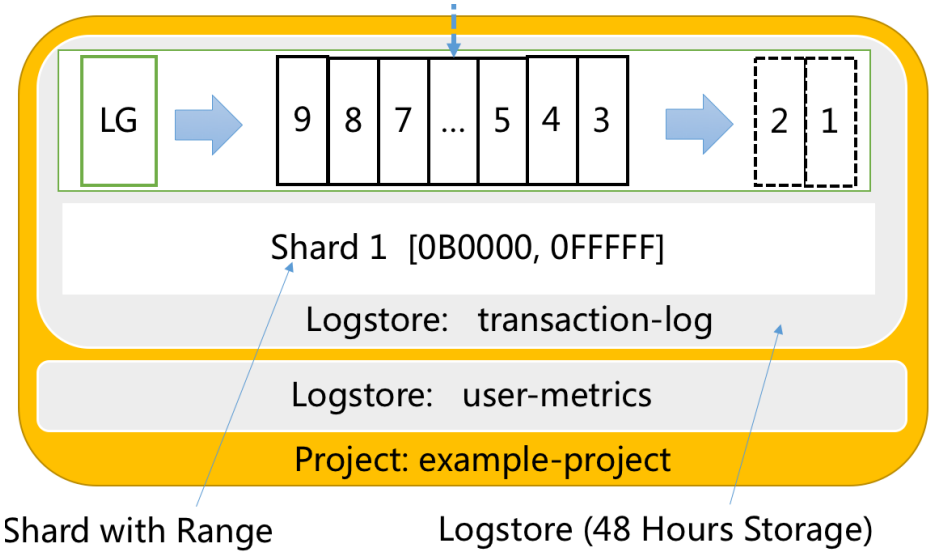
- Log: 由时间、及一组Key,Value对组成
- LogGroup: 一组日志的集合，包含相同Meta（IP，Source）等

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）。

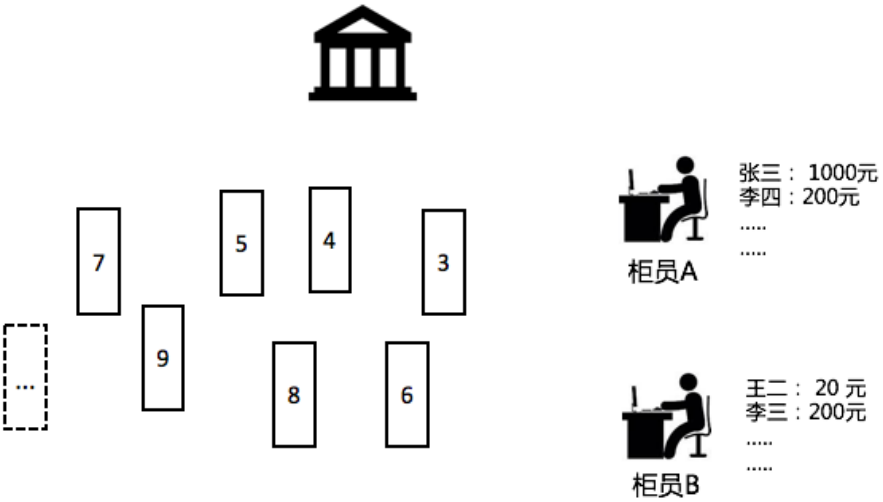
- Log/LogGroup：用户发出的存取款等操作。

- 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。

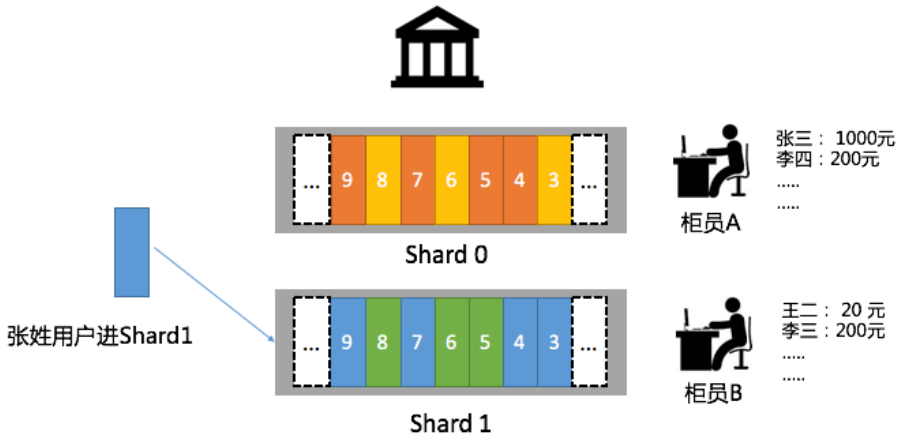


实现保序的方法很简单：排队，创建一个Shard,终端只有一个柜员A来处理。用户请求先进先出，一点问题都没有。但带来的问题是效率低下，假设有1000个用户来进行操作，即使有10个柜员也无济于事。

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。



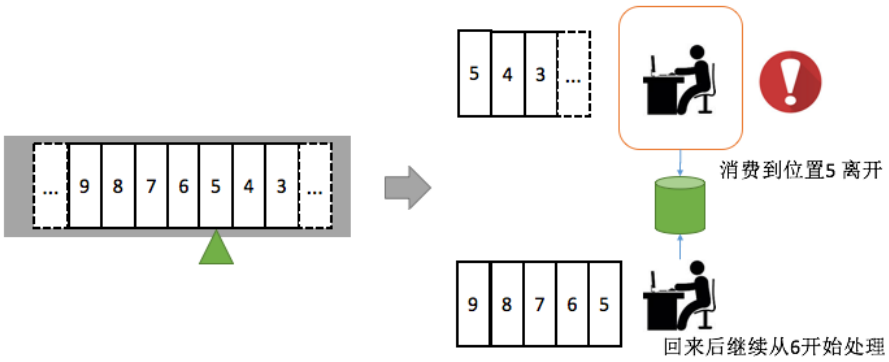
问题2：不丢失（At-Least Once）

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。

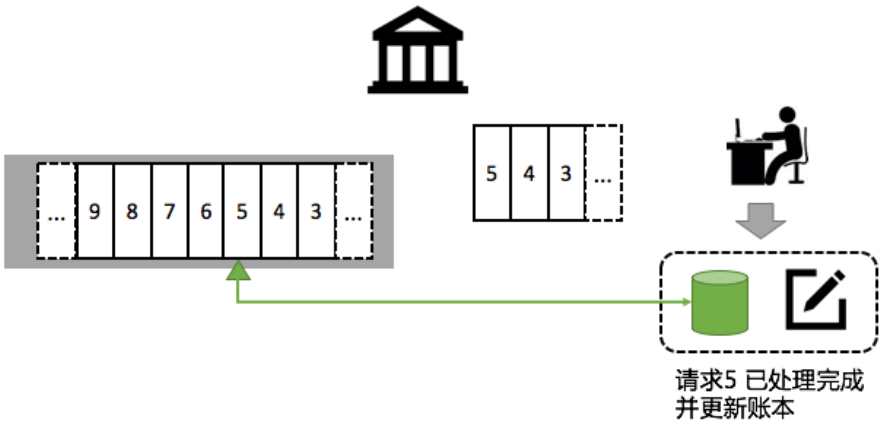
问题3：不重复（Exactly Once）

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

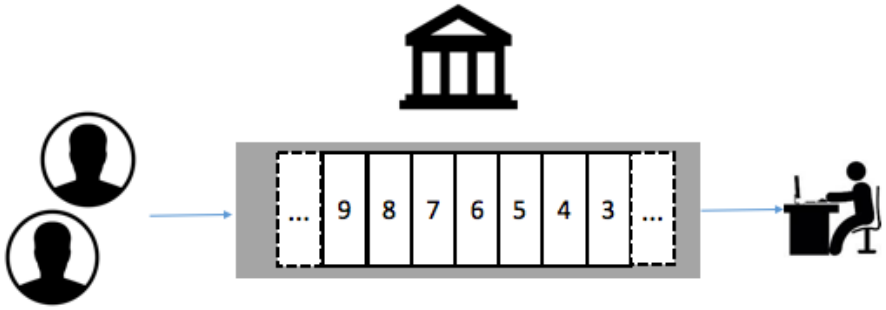
以上三个概念解释完成后，原理并不复杂。但在现实世界中，规模的变化与不确定性会使得以上三个问题便得更复杂。例如：

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

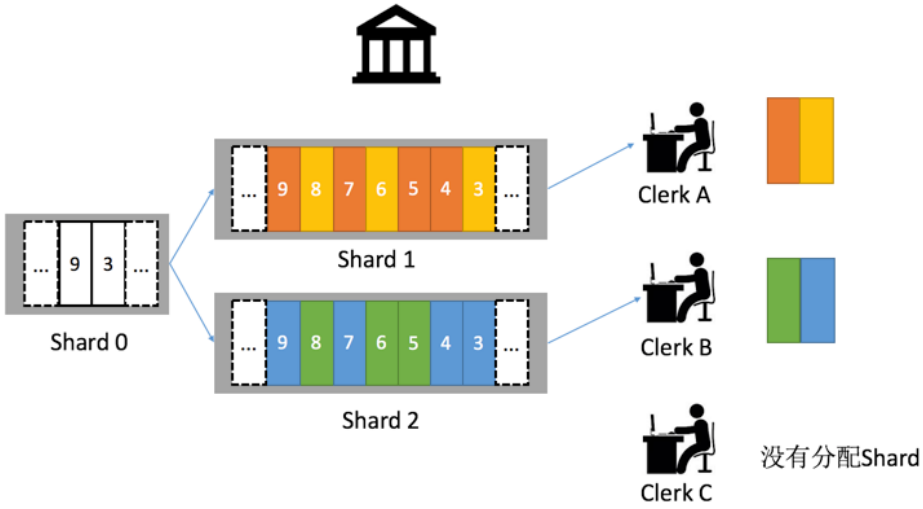
8点银行开门

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。

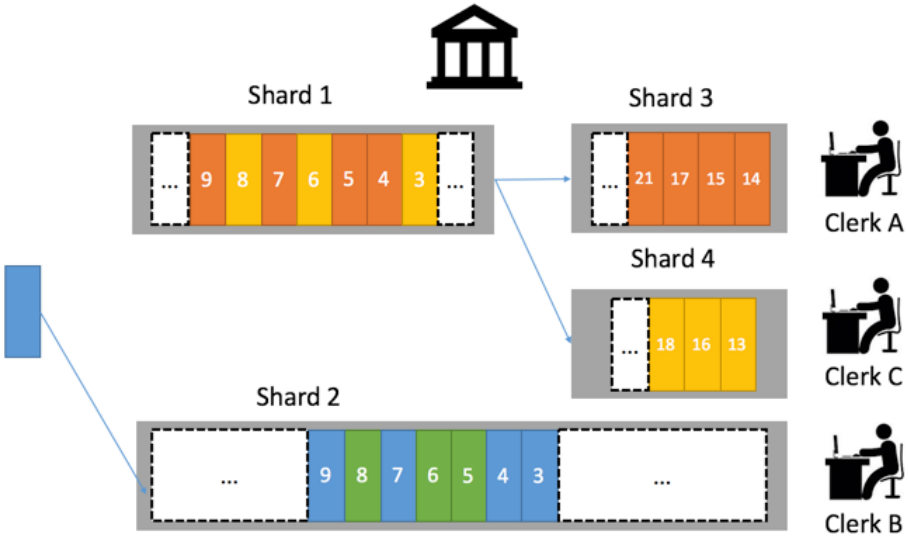


10-12点请求消费状态：

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，参考LogHub弹性伸缩(Merge/Split)。
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，参考LogHub Consumer Library-协同消费组自动负载均衡。
3. 过程中支持保序，参考LogHub支持保序写入和消费。
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，参考通过控制台查看协同消费组进度。
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），参考LogHub多种数据接入方式。

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。

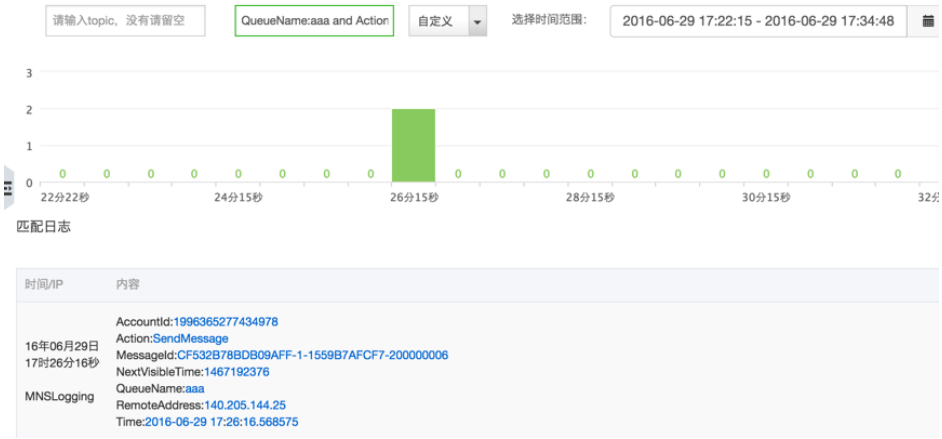
另外，Storm、Spark Streaming已经通过Consumer Library实现了对应的接口，欢迎试用。有兴趣的读者可以参考下日志服务的主页，以及日志处理圈子，里面有不少干货哦。

阿里云消息服务(MNS)开通将日志推送日志服务功能，这里我们介绍下如何利用这部分日志。

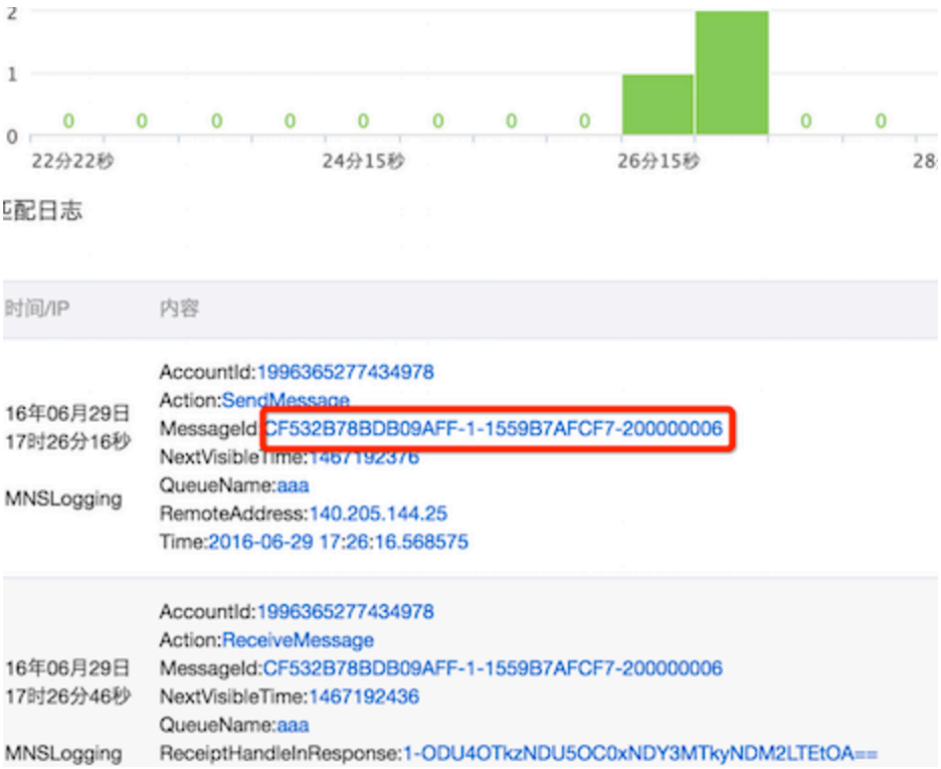
消息服务日志格式队列消息操作日志、以及主题消息操作日志两个章节，其中日志包含了消息生命周期的所有内容，时间、地点、操作和上下文等。您可以通过三种方法对日志进行分析：

实时查询

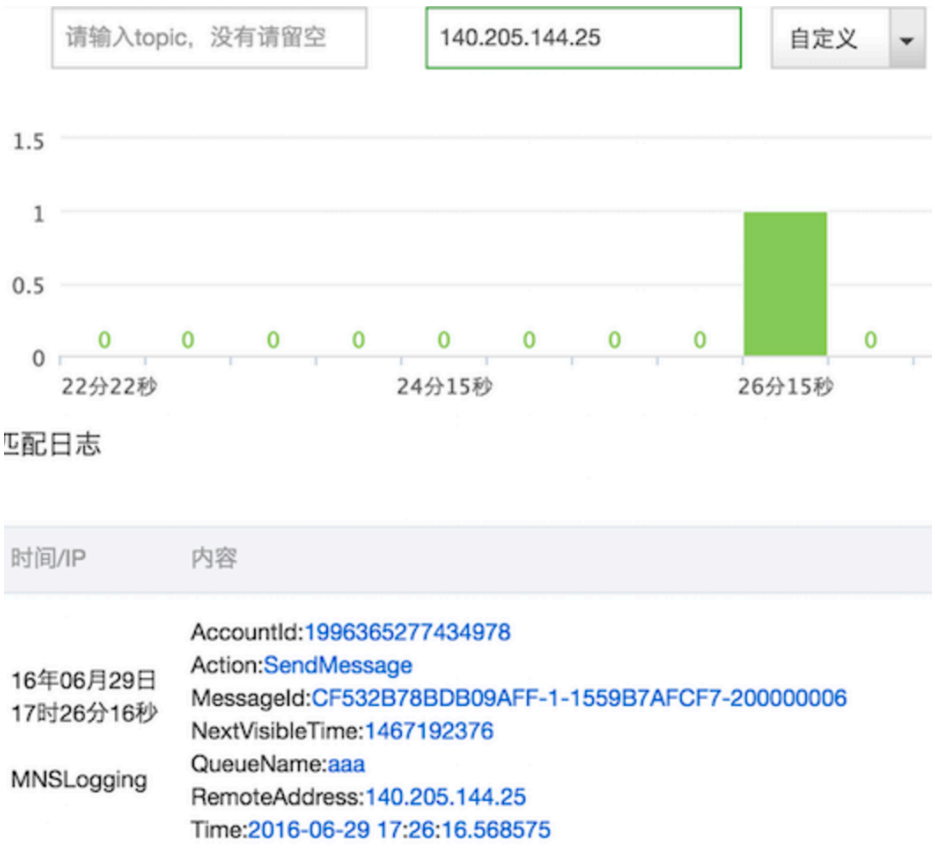
选定时间内，发送的消息数量，或指定 Queue，以及 Action:SendMessage 既可以看到该时间段内有2条消息被发出。



某一条消息的生命周期，通过在Query中输入MessageId既可以快速检索到。



要查询某个服务器向消息队列发布的消息数量，输入该服务器IP即可，也可以通过IP + DeleteMessage等组合查询该时间段行为。



实时计算 & 离线计算

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

在几百台机器、十几个应用程序、面向万级用户量的场景下定位问题是非常大的挑战，往往需要管理员在多维度及条件变量下进行实时排查。尤其是在网络攻击中，攻击者会不断地变化来源IP、目标等，让我们无法实时做出反应。这类场景的解决方案不仅需要海量处理能力，还需需要保证实时业务不中断，LS+LogHub可以确保日志从服务器产生到被查询在3秒以内（99.9%情况下），让您永远快人一步。

日志服务(原SLS)是针对大规模日志实时采集与查询服务，支持文本、数值、模糊、上下文等查询方式。在2017年5月版本中开始支持 **SQL实时统计分析功能** ，能够在日志数据秒级查询的基础上支持实时统计分析。

操作方法

在控制台或API上对日志数据进行检索时，在SQL查询语句后增加管道操作符“|”与统计子句，可以在查询日

志的同时，对查询结果进行实时的统计或计算处理。其中SQL统计子句不需要定义from和where字段，默认从当前Logstore查询，统计范围为管道操作符前的检索结果。

支持的SQL子句包括：聚合、Group By（包括Cube、Rollup）、Having、排序、字符串、日期、数值操作，以及统计和科学计算等。详细使用方法及统计语法请参见分析语法。

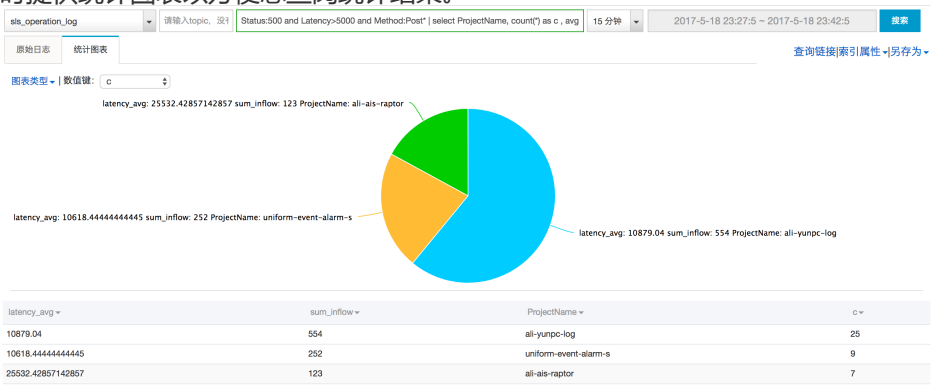
例如，在所有日志中查询状态码为500、延时大于5000微秒、请求方法为POST开头的所有日志，查询语句为：

```
Status:500 and Latency>5000 and Method:POST*
```

如果需要在查询结果中，根据项目名称（ProjectName）挑选出发生次数最大的10个项目，并且计算出平均延时，需要在SQL查询语句后增加管道操作符“|”，以及相关SQL统计子句。增加统计子句后的SQL语句为：

```
Status:500 and Latency>5000 and Method:Post* | select count(*) as c , avg(Latency) as latency_avg, sum(Inflow) as sum_inflow, ProjectName Group by ProjectName order by c Desc limit 10
```

在日志检索界面的查询框中输入以上SQL语句并点击**搜索**，可以在控制台上获得统计结果（见下图）。我们同时提供统计图表以方便您查阅统计结果。

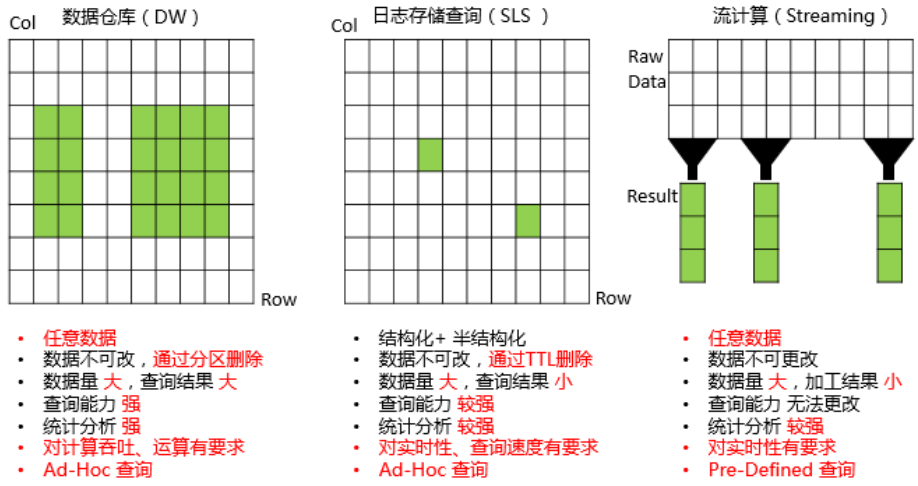


> 为了更快速地获得统计结果，我们暂时对SQL执行数据设置了限制条件，详见SQL分析语法中“使用限制”部分。在机房扩容后和下一步优化后，我们会取消该限制，敬请关注。

功能优势

日志服务产品的检索统计功能与数据仓库、流计算等分析引擎相比，有如下特点：

- 支持结构化、半结构化数据
- 实时性强、查询延时短
- 支持数据量大，查询结果集合相对较小



除此之外日志服务支持对接MaxCompute、OSS（E-MapReduce、Hive、Presto）、TableStore、流计算（Spark Streaming、Storm、Stream Compute）、Cloud Monitor、ARMS（TLog）、Powerlog，可以方便地将日志数据进行多样化处理。

使用实例

管理员通过日志服务对服务器上的访问日志进行实时采集和分析，通过监控数据发现有HTTP状态码非200的访问错误产生，可以通过以下步骤进行排查。

统计报告该错误的用户数量。查询产生该错误的用户数量可以确定此问题是全体用户均受影响的普遍问题，还是仅部分用户受到影响的个别问题。

```
Status in (200 500] | Select count(*) as c, ProjectName group by ProjectName
```

使用以上语句查询统计后，发现大部分错误产生于名为“abc”的Project下。

统计名为“abc”的Project中，访问错误的日志的请求方式。

```
Status in (200 500] and ProjectName:"abc"| Select count(*) as c, Method Group by Method
```

使用以上语句查询统计后，发现此错误均发生于POST请求中。

统计写POST请求的延时分布。

```
Status in (200 500] and ProjectName:"abc" and Method:POST* | select numeric_histogram(10,latency)
```

使用以上语句查询统计后，发现POST请求延时高。

确定高延时的产生原因。

通过时序分析，对这部分请求进行一个时间维度的划分，以此确定问题的产生是间歇性的还是持续性的。

```
Status in (200 500] and ProjectName:"abc" and Method:Post* |select  from_unixtime( __time__ - __time__ % 60) as t,
truncate (avg(latency) ) ,
current_date
group by __time__ - __time__ % 60
order by t desc
limit 60
```

对产生问题的POST请求按服务端IP进行统计，以此定位产生问题的服务端IP，指向该IP地址的访问请求均会发生非200访问错误。

```
Status in (200 500] and ProjectName:"abc" and Method:Post* and Latency>150000 | select count(*) as c, Ip Group by Ip order by c desc
```

数次排查后可大致定位到高延时的问题机器。

根据问题机器的RequestId继续在日志数据中进行查询与搜索。

以上操作都可以通过控制台或API完成。

访问日志实时分析案例

很多个人站长在搭建网站时使用nginx作为服务器，为了了解网站的访问情况，一般有两种手段：

- 使用CNZZ之类的方式，在前端页面插入js，用户访问的时候触发js，记录访问请求。
- 利用流计算、或离线统计分析nginx的access log，从日志中挖掘有用信息。

两种方式各有优缺点：

- CNZZ、百度、谷歌统计等使用比较简单，各种指标定义清楚。但这种方式只能记录页面的访问请求，像ajax之类的请求是无法记录的，还有爬虫信息也不会记录。
- 利用流计算、离线计算引擎可以支持个性化需求，但需要搭建一套环境，并且在实时性以及分析灵活性上比较难平衡。

日志服务在查询基础上新推出SQL支持实时日志分析功能，极大的降低了站长们分析访问日志（access log）门槛，本文将详细介绍如何使用日志服务分析access log中的各种指标。

Nginx访问日志格式

一个典型的nginx访问日志配置：

```
    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;
```

字段解释：

1. remote_addr : 客户端地址
2. remote_user : 客户端用户名
3. time_local : 服务器时间
4. request : 请求内容，包括方法名，地址，和http协议
5. http_host : 用户请求是使用的http地址
6. status : 返回的http 状态码
7. request_length : 请求大小
8. body_bytes_sent : 返回的大小
9. http_referer : 来源页
10. http_user_agent : 客户端名称
11. request_time : 整体请求延时

收集访问日志到日志服务

首先把日志收集到日志服务

请参考：5分钟快速文档

把日志收集到日志服务后，设置每一列的类型：

注：其中request拆分城method 和uri两列

日志样例：

分析访问日志

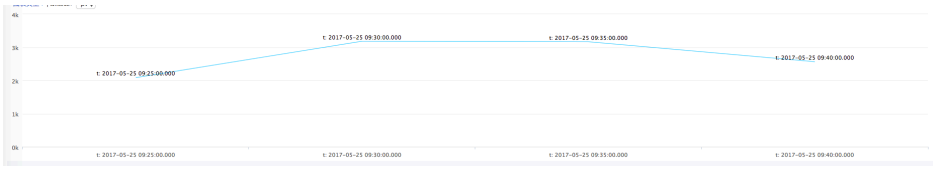
通常，对access log的访问需求有，查看网站的pv，uv，热点页面，热点方法，错误请求，客户端类型，来源页面等等。下文将逐个介绍各个指标的计算方法。

PV统计不仅可以一段时间总的PV，还可以按照小的时间段，查看每段时间的，比如每5分钟pv

统计代码

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

统计结果



统计一小时内每5分钟的UV

统计代码：

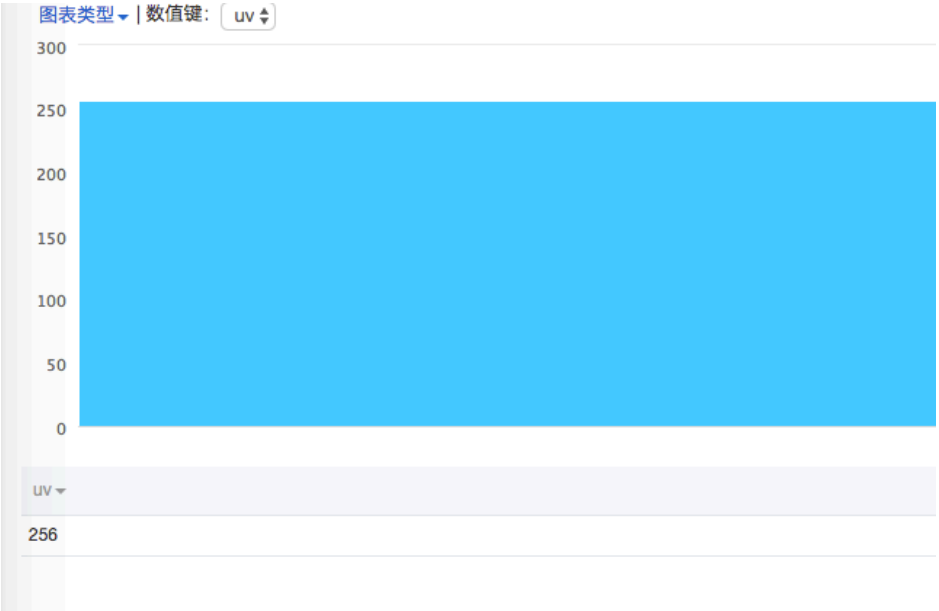
```
 *|select from_unixtime( __time__- __time__% 300) as t,
approx_distinct(remote_addr) as uv
group by __time__- __time__% 300
order by t limit 60
```

统计一小时内总的UV

统计代码：

```
 *|select approx_distinct(remote_addr)
```

统计结果：



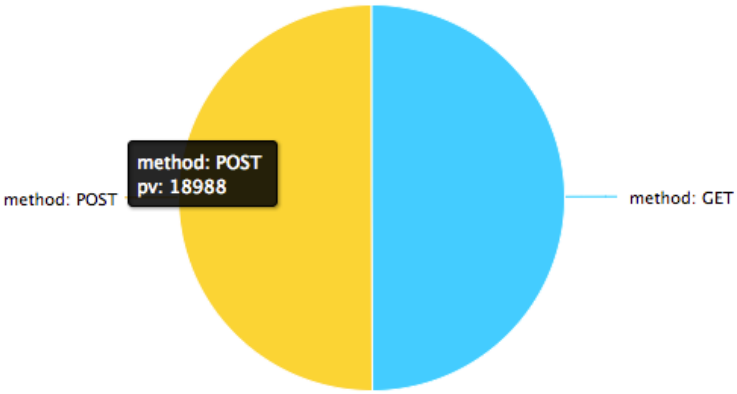
最近一小时访问最多的10个页面

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

pv	url
3881	/9
3857	/4
3843	/1
3803	/5
3790	/6
3776	/8
3773	/0
3769	/7
3742	/3
3697	/2

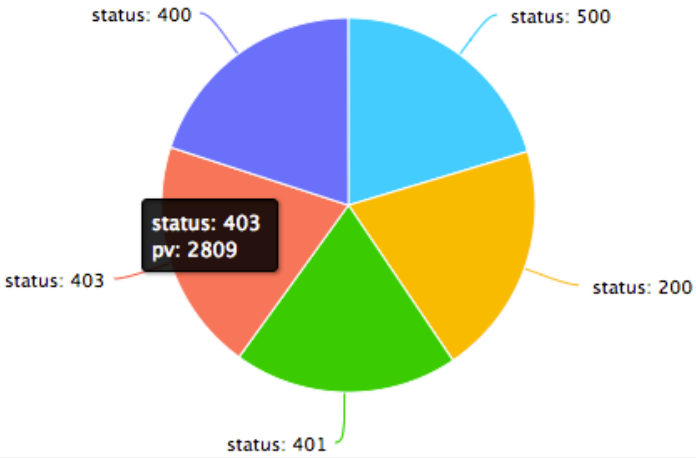
最近一小时各种请求方法的占比

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



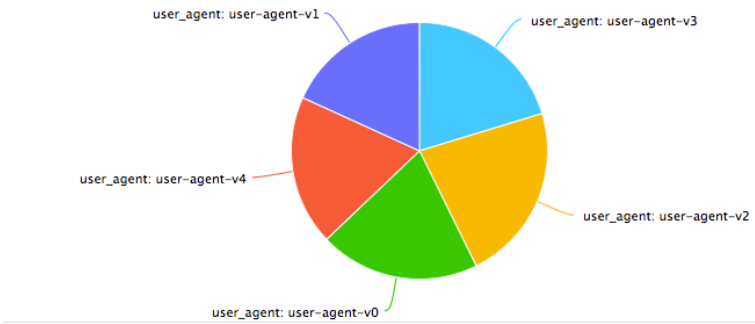
最近一小时各种http状态码的占比

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



最近一小时各种浏览器的占比

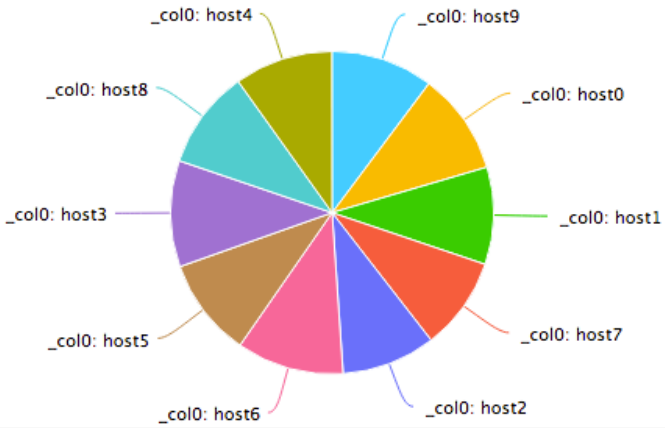
```
*| select user_agent, count(1) as pv group by user_agent
```



最近一小时referer来源于不同域名的占比

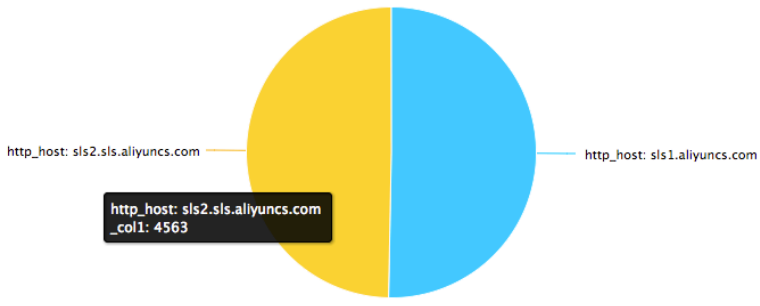
```
*|select url_extract_host(http_referer) ,count(1) group by url_extract_host(http_referer)
```

注：url_extract_host为从url中提取域名



最近一小时用户访问不同域名的占比

```
*|select  http_host ,count(1) group by http_host
```

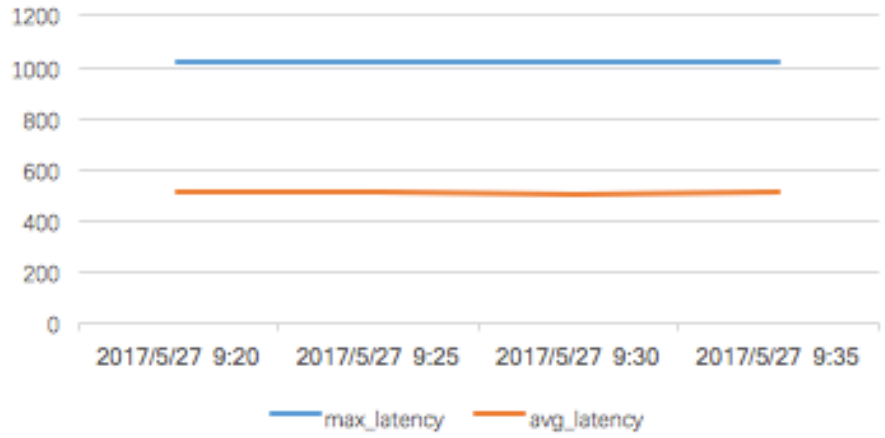


其他高级功能

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。

通过每5分钟的平均延时和最大延时，对延时的情况有个总体的把握

```
*|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
```



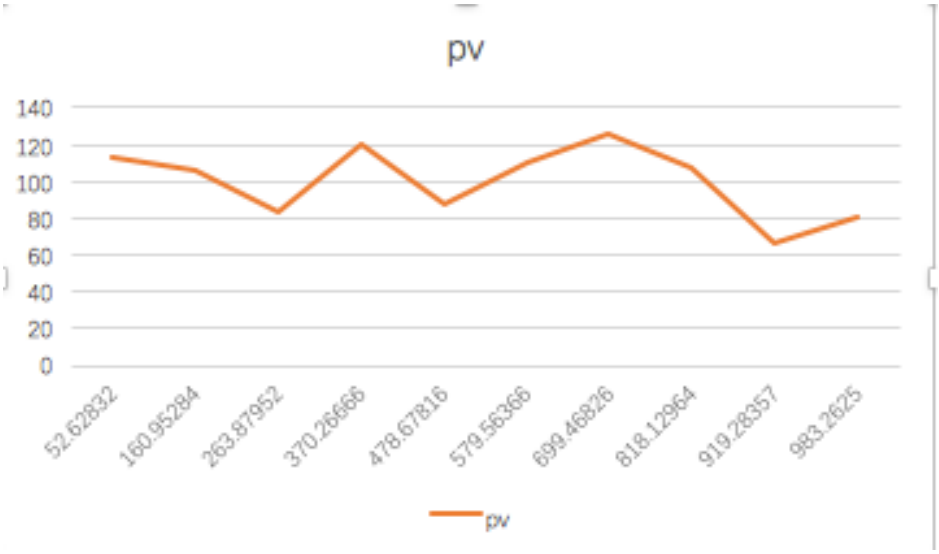
知道了最大延时之后，我们需要知道最大延时对应的请求页面是哪个，方便进一步优化页面响应。

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

/2	2017-05-27 09:33:00.000
/0	2017-05-27 09:36:00.000
/8	2017-05-27 09:22:00.000
/6	2017-05-27 09:28:00.000
/6	2017-05-27 09:35:00.000
/9	2017-05-27 09:34:00.000
/5	2017-05-27 09:31:00.000
/7	2017-05-27 09:24:00.000
/8	2017-05-27 09:21:00.000
/3	2017-05-27 09:27:00.000

从总体把握，我们需要知道网站的所有请求的延时的分布，把延时分布在十个桶里边，看每个延时区间的请求个数

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



除了最大的延时，我们还需要知道最大的十个延时，对应的值是多少

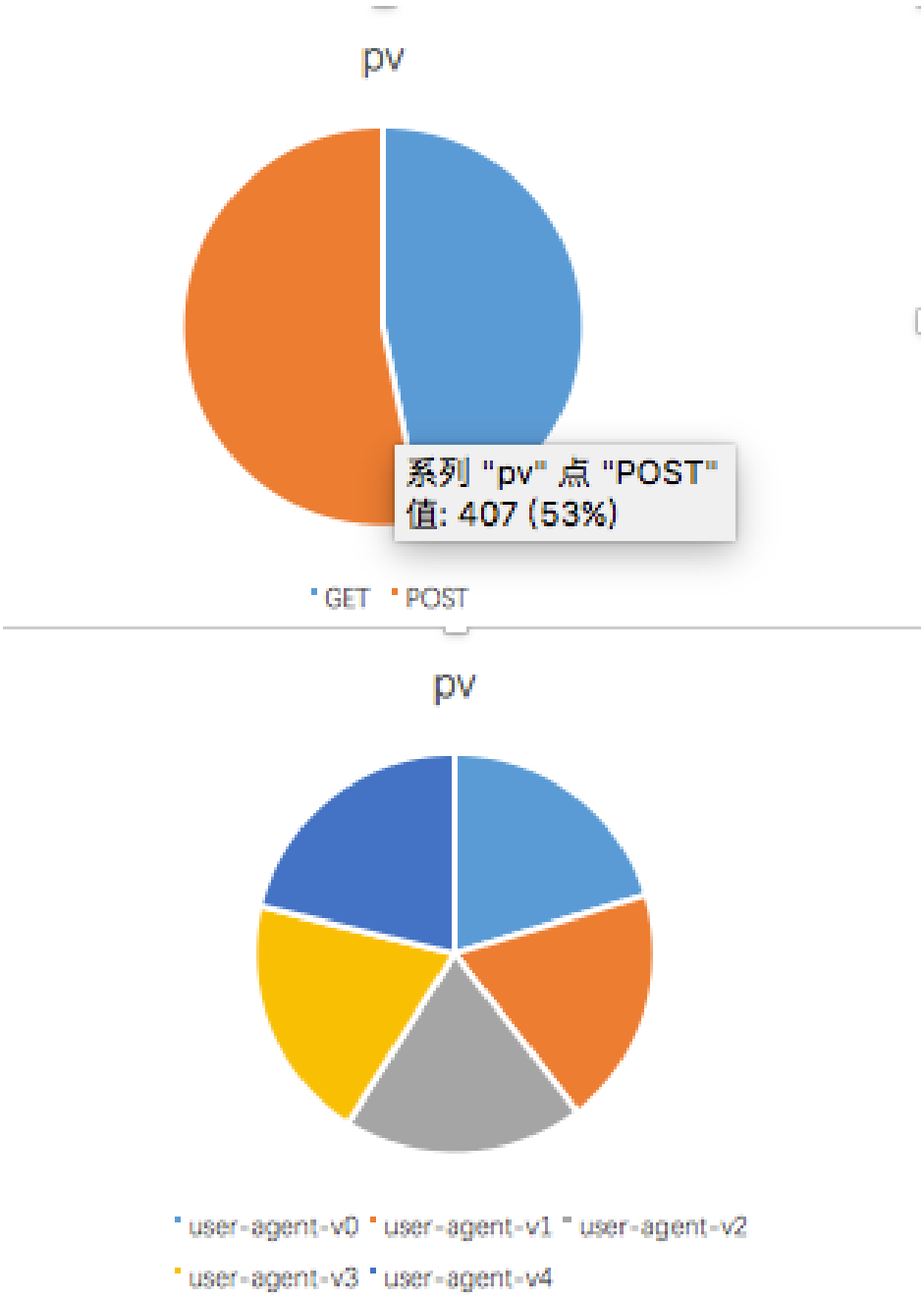
```
*|select max(request_time,10)
```

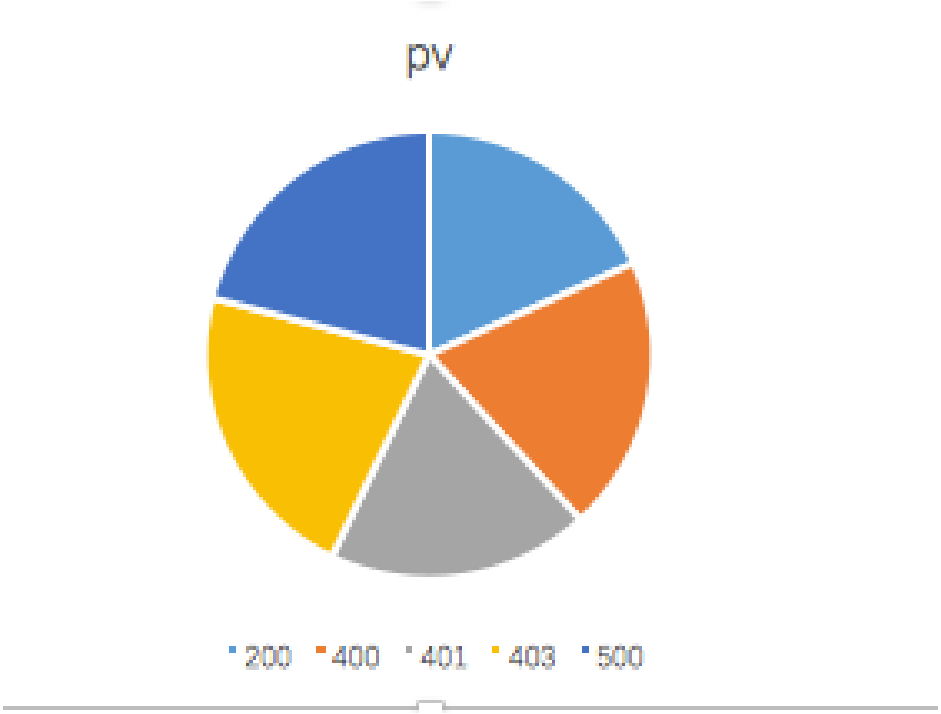
_col0
[1023, 1023, 1023, 1023, 1022, 1022, 1022, 1022, 1022, 1022]

当我们知道了/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,UV,各种method次数，各种status次数,各种浏览器次数，平均延时，最大延时

```
 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
```

avg_latency	method_pv	uv	max_latency	pv	user_agent_pv	status_pv
518.84032287817	{ "GET" : 358, "POST" : 407 }	236	1023	765	{ "user-agent-v0" : 155, "user-agent-v1" : 147, "user-agent-v2" : 149, "user-agent-v3" : 152, "user-agent-v4" : 162 }	{ "200" : 140, "400" : 152, "401" : 145, "403" : 168, "500" : 160 }

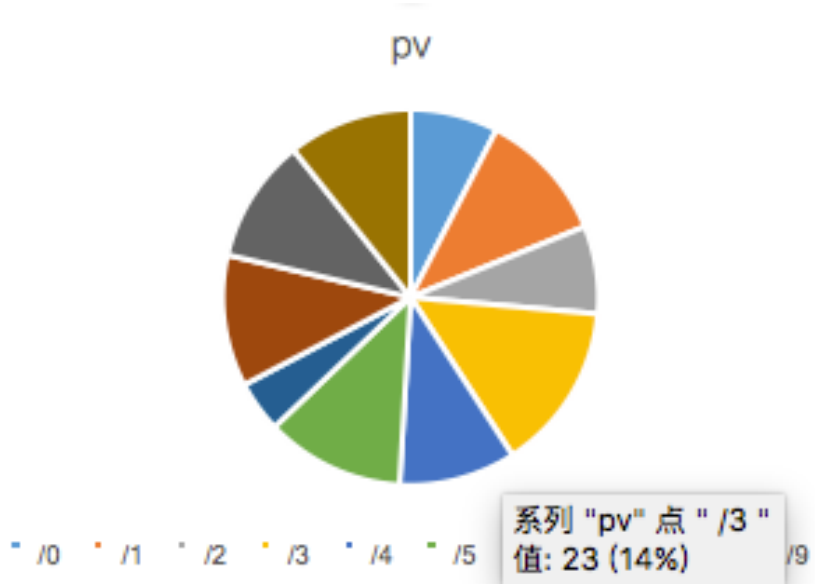

pv
系列 "pv" 点 "POST"
值: 407 (53%)
GET
POST
pv
user-agent-v0
user-agent-v1
user-agent-v2
user-agent-v3
user-agent-v4



同时，我们也可以限定只查看request_time 大于1000的请求的pv，uv，以及各个url的请求次数

```
request_time > 1000 |select count(1) as pv, approx_distinct(remote_addr) as uv, histogram(url) as url_pv
```

uv	pv	url_pv
108	159	{ "/0" : 12, "/1" : 18, "/2" : 12, "/3" : 23, "/4" : 16, "/5" : 19, "/6" : 7, "/7" : 18, "/8" : 17, "/9" : 17 }



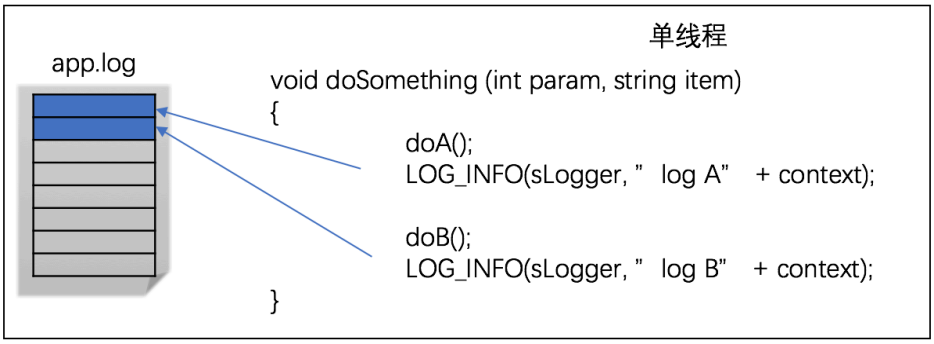
随着技术的发展，越来越多的系统从单机转向到分布式，以及无状态计算（例如FunctionCompute等），同时

对应的日志查看方式除了直接查看文件外，也发展出了各种集中日志管理和查询的方式。

具体如下：

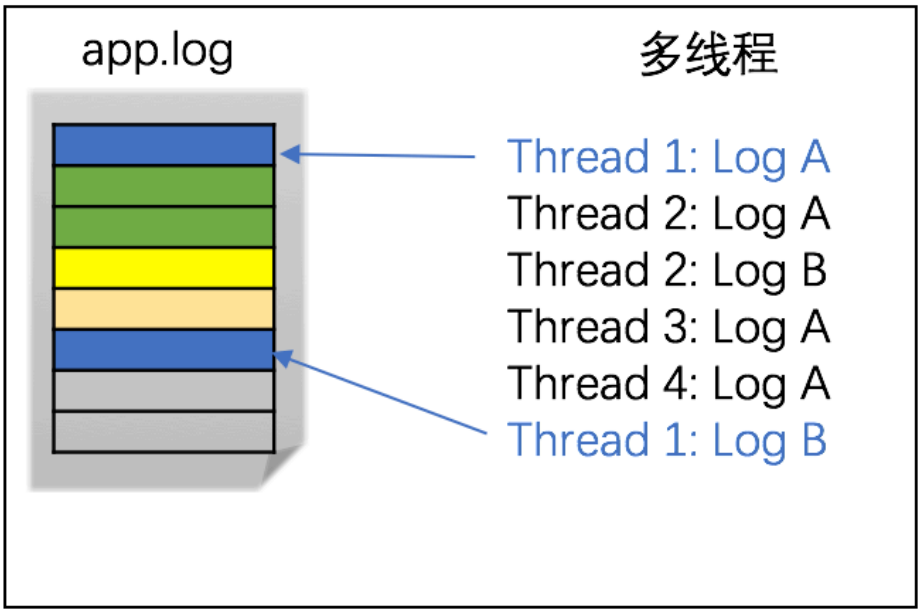
1. 单线程程序（三十年前）

开发单机程序时诊断非常简单：设置断点，检查断点上下变量状态即可。当程序部署线上后，可以把断点中内容以日志方式记录下来。出现错误后，还原日志上下文（Context）不难，只要把日志文件中前后N行拿到后都能容易找到问题



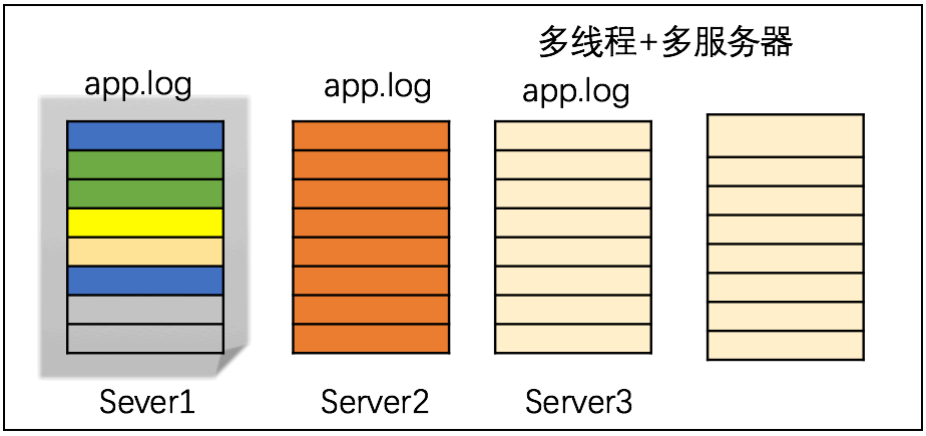
2. 多线程程序（十年前）

当多线程出现后，还原上下文变得复杂一些，因为不同线程打日志顺序会造成一定干扰。我们可以在日志中记录、筛选线程（ThreadID）来排除其他线程干扰，拿到上下文



3. 分布式系统（现在）

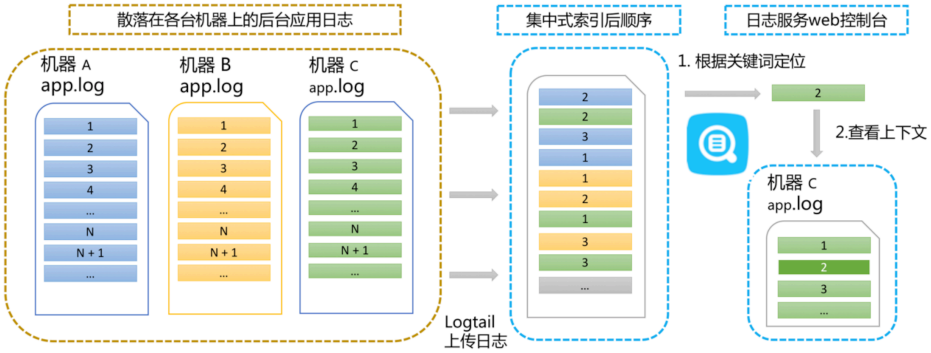
在分布式场景下（比如Docker、LXC、多服务器等），进程会同时跑在不同服务器上，一般通过集中式日志管理（例如Splunk、ELK、日志服务）等查找定位日志。



这种场景下查找日志非常方便（例如通过”Level:Error”，”Latency>100000 & Method:HandleRequest”等能对上亿日志进行快速查找），但定位到关键日志后还原上下文却变得很难。因为日志采集、索引都是无序的，为了还原上下文，必须通过时间、机器、线程号来筛选和查找（这种方法一般不够精确），业界并没有好的方法。

日志上下文查看

日志服务（原SLS）LogSearch在查询分析基础上，提供了查看上下文的按钮，既可以在控制台上 **完全精确** 还原任意日志上下文（上下N条日志），并支持对指定内容（比如ThreadId）进行筛选，帮助你快速定位问题。整个过程如图所示：



使用依赖

- 只支持使用Logtail客户端收集的日志（后续我们会和LogAgent合作支持该功能）
- 指定日志库（Logstore）必须开启索引功能

具体参见文档索引查询下**上下文查询**。

实际场景

- Dev在线上通过关键词查询，定位到一个异常日志

- 点击上下文查询后，既跳转到前后N条上下文显示框
  - 可以通过“更早”、“更新”等按钮加载更多上下文，也可以点击“返回普通搜索模式”进一步排查
  - 通过筛选框筛选ThreadID，进行精准上下文的过滤

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：

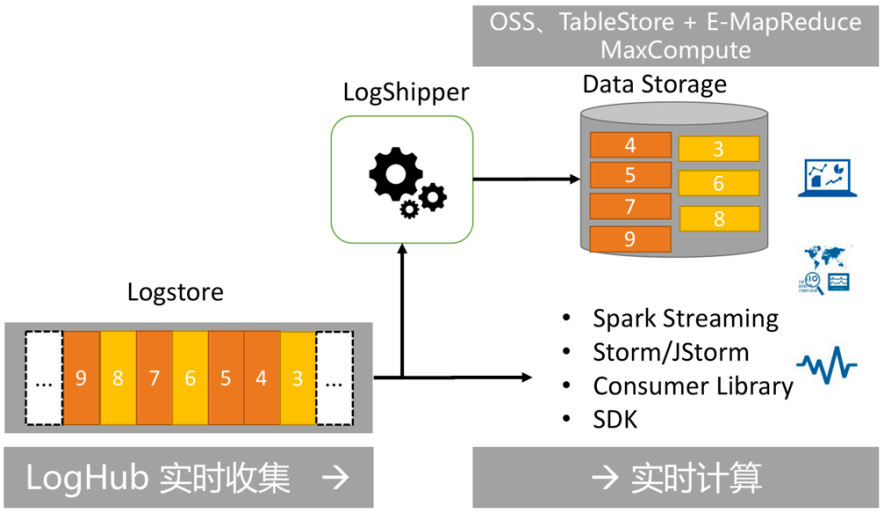
- Lamdba Architecture: 数据进来后，既支持流式处理、同时存入数仓。但用户发起查询时，会根据查询需求和复杂度从实时计算、离线计算拿结果返回。
- Kappa Architecture: kafka based Architecture。弱化离线计算部分，数据存储都在Kafka中，实时计算解决所有问题。

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
- TableStore(NoSQL数据存储服务):
  - 操作步骤
- MaxCompute(大数据计算服务):
  - 说明文档：



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。

Kafka是分布式消息系统，由于其高吞吐和水平扩展，被广泛使用于消息的发布和订阅。以开源软件的方式提供，各用户可以根据需要搭建Kafka集群。

日志服务(Log Service)是基于飞天Pangu构建的针对日志平台化服务。服务提供各种类型日志的实时采集，存储，分发及实时查询能力。通过标准话的Restful API对外提供服务。

Log Service Loghub提供公共的日志采集、分发通道，用户如果不想自己搭建、运维kafka集群，可以使用Log Service LogHub功能。

Log Service Loghub & Kafka 概念映射

概念	Kafka	Loghub
存储对象	topic	logstore
水平分区	partition	shard
数据消费位置	offset	cursor

Loghub & Kafka 功能比较

功能	Kafka	LogHub

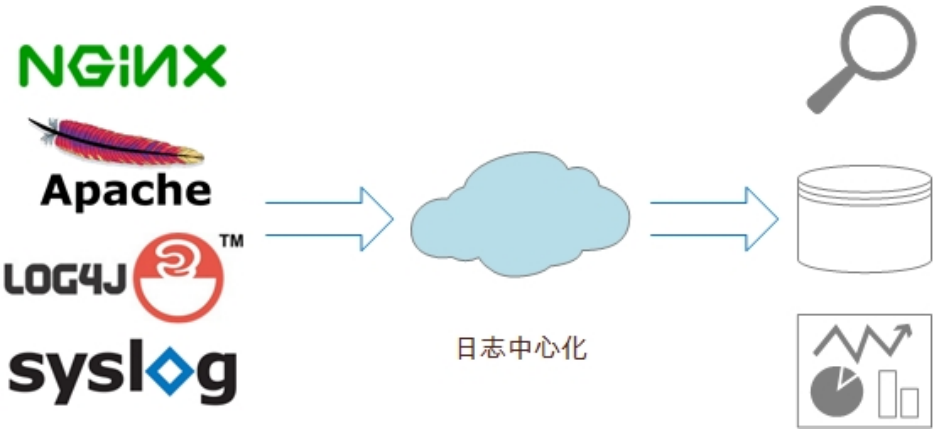
使用依赖	自建或共享Kafka集群	Log Service服务
通信协议	TCP 打通网络	Http (restful API)，80端口
访问控制	无	基于云账号的签名认证+访问控制
动态扩容	暂无	支持动态shard个数弹性伸缩(Merge/Split)，对用户无影响
多租户Qos	暂无	基于shard的标准化流控
数据拷贝数	用户自定义	暂不开放，默认3份拷贝
failover/replication	调用工具完成	自动，用户无感知
扩容/升级	调用工具完成，影响服务	用户无感知
写入模式	round robin/key hash	暂只支持round robin/key hash
当前消费位置	保存在kafka集群的zookeeper	服务端维护、无需关心
保存时间	配置确定	根据需求动态调整

成本对比

参见成本对比（其他方案）下LogHub部分

日志采集场景下客户端测评

DT时代，数以亿万计的服务器、移动终端、网络设备每天产生海量的日志。中心化的日志处理方案有效地解决了在完整生命周期内对日志的消费需求，而日志从设备采集上云是始于足下的第一步。



三款日志采集工具

Logstash

- 开源界鼎鼎大名ELK stack中的”L”，社区活跃，生态圈提供大量插件支持
- Logstash基于JRuby实现，可以跨平台运行在JVM上
- 模块化设计，有很强的扩展性和互操作性。

Fluentd

- 开源社区中流行的日志采集工具，td-agent是其商业化版本，由Treasure Data公司维护，是本文选用的评测版本。
- Fluentd基于CRuby实现，并对性能表现关键的一些组件用C语言重新实现，整体性能不错。
- Fluentd设计简洁，pipeline内数据传递可靠性高
- 相较于Logstash，其插件支持相对少一些。

- Logtail
  - 阿里云日志服务的生产者，经过3年多阿里集团大数据场景考验
  - 采用C++语言实现，对稳定性、资源控制、管理等下过很大的功夫，性能良好
  - 相比于Logstash、Fluentd的社区支持，Logtail功能较为单一，专注日志采集功能。

功能对比

功能项	Logstash	Fluentd	Logtail
日志读取	轮询	轮询	事件触发
文件轮转	支持	支持	支持
Failover处理 (本地checkpoint)	支持	支持	支持
通用日志解析	支持grok（基于正则表达式）解析	支持正则表达式解析	支持正则表达式解析
特定日志类型	支持delimiter、key-value、json等主流格式	支持delimiter、key-value、json等主流格式	支持delimiter、key-value、json等主流格式
数据发送压缩	插件支持	插件支持	LZ4
数据过滤	支持	支持	支持
数据buffer发送	插件支持	插件支持	支持
发送异常处理	插件支持	插件支持	支持
运行环境	JRuby实现，依赖JVM环境	CRuby、C实现，依赖Ruby环境	C++实现，无特殊要求
线程支持	支持多线程	多线程受GIL限制	支持多线程
热升级	不支持	不支持	支持
中心化配置管理	不支持	不支持	支持
运行状态自检	不支持	不支持	支持cpu/内存阈值保

			护

日志文件采集场景 - 性能对比

日志样例:以Nginx的access log为样例，如下一条日志365字节，结构化成14个字段：

```
42.120.74.166 370261 - [14/Nov/2015:17:50:05 +0800] "POST http://www.xxx.com/auction/order/
ip rt time url
unity_order_confirm.htm" 200 1152 "http://www.xxx.com/test_now.jhtml" "Mozilla/5.0 (Windows NT 6.1)
status size ref
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/28.0.1500.72 Safari/537.36" "316312088"
agent cookie_unb
"78c97666dbee0bc3dc5558e4f5a28e55" "ac1539981387814767045178 4e" center test_local 29374
cookie_cookie2 monitor_traceid cell ups remote_port
```

在接下来的测试中，将模拟不同的压力将该日志重复写入文件，每条日志的time字段取当前系统时间，其它13个字段相同。

相比于实际场景，模拟场景在日志解析上并无差异，有一点区别是：较高的数据压缩率会减少网络写出流量。

Logstash

logstash-2.0.0版本，通过grok解析日志并写出到kafka（内置插件，开启gzip压缩）。

日志解析配置：

```
grok {
patterns_dir=>"/home/admin/workspace/survey/logstash/patterns"
match=>{ "message"=>"%{IPORHOST:ip} %{USERNAME:rt} - \[%{HTTPDATE:time}\] \"%{WORD:method}
%{DATA:url}\" %{NUMBER:status} %{NUMBER:size} \"%{DATA:ref}\" \"%{DATA:agent}\" \"%{DATA:cookie_unb}\"
\"%{DATA:cookie_cookie2}\" \"%{DATA:monitor_traceid}\" %{WORD:cell} %{WORD:ups}
%{BASE10NUM:remote_port}" }
remove_field=>["message"]
}
```

测试结果：

写入TPS	写入流量 (KB/s)	CPU使用率 (%)	内存使用 (MB)
500	178.22	22.4	427
1000	356.45	46.6	431
5000	1782.23	221.1	440
10000	3564.45	483.7	450

Fluentd

td-agent-2.2.1版本，通过正则表达式解析日志并写入kafka（第三方插件fluent-plugin-kafka，开启gzip压缩）。

日志解析配置：

```
<source>
type tail
format /^(?<ip>\S+)\s(?<rt>\d+)\s-
\s\[(?<time>[^\]]*)\]\s"(?<url>[^\"]+)"\s(?<status>\d+)\s(?<size>\d+)\s"(?<ref>[^\"]+)"\s"(?<agent>[^\"]+)"\s"(?<
cookie_unb>\d+)"\s"(?<cookie_cookie2>\w+)"\s"(?
<monitor_traceid>\w+)"\s(?<cell>\w+)\s(?<ups>\w+)\s(?<remote_port>\d+).*$/
time_format %d/%b/%Y:%H:%M:%S %z
path /home/admin/workspace/temp/mock_log/access.log
pos_file /home/admin/workspace/temp/mock_log/nginx_access.pos
tag nginx.access
</source>
```

测试结果：

写入TPS	写入流量 (KB/s)	CPU使用率 (%)	内存使用 (MB)
500	178.22	13.5	61
1000	356.45	23.4	61
5000	1782.23	94.3	103

注：受GIL限制，Fluentd单进程最多使用1个cpu核心，可以使用插件multiprocess以多进程的形式支持更大的日志吞吐。

Logtail

logtail 0.9.4版本，设置正则表达式进行日志结构化，数据LZ4压缩后以HTTP协议写到阿里云日志服务，设置batch_size为4000条。

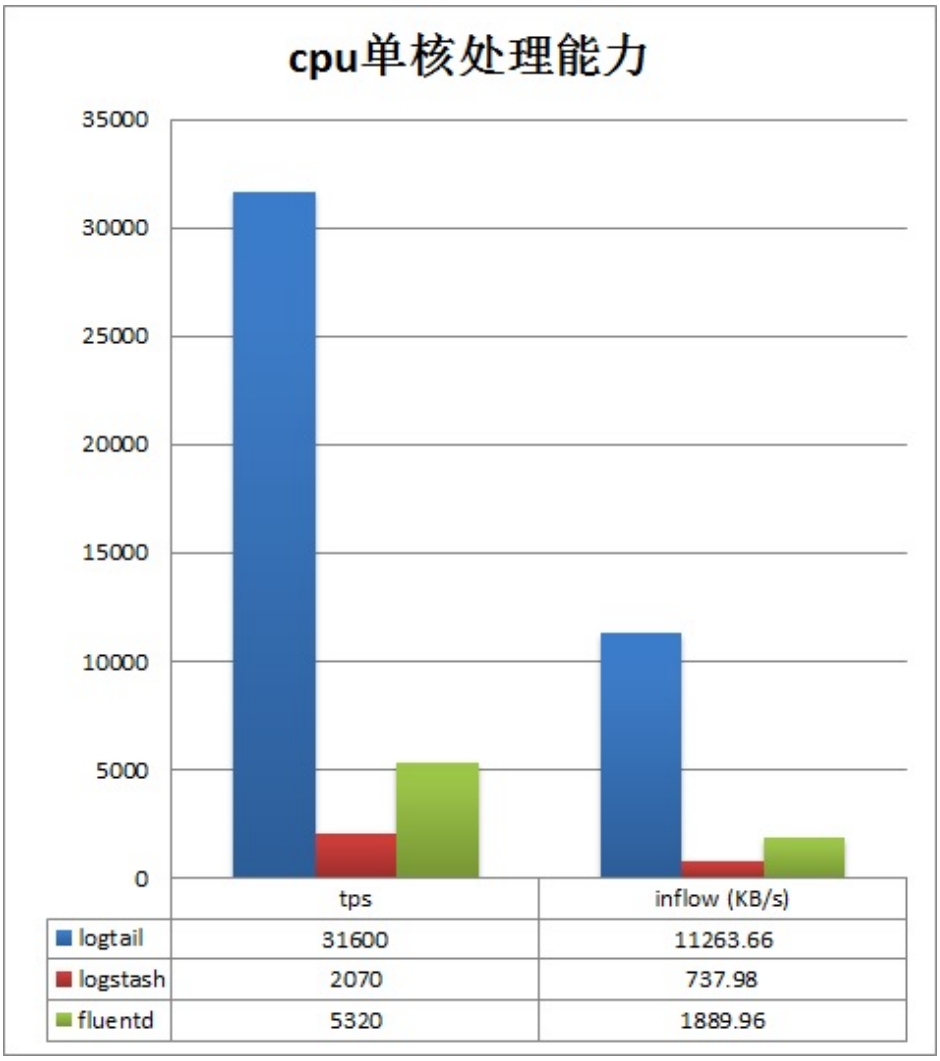
日志解析配置：

```
logRegex : (\S+)\s(\d+)\s-
\s\[([^]]+)]\s"([^"]+)"\s(\d+)\s(\d+)\s"([^"]+)"\s"([^"]+)"\s"(\d+)"\s"(\w+)"\s"(\w+)"\s(\w+)\s(\w+)\s(\d+).*
keys : ip,rt,time,url,status,size,ref,agent,cookie_unb,cookie_cookie2,monitor_traceid,cell,ups,remote_port
timeformat : %d/%b/%Y:%H:%M:%S
```

测试结果：

写入TPS	写入流量 (KB/s)	CPU使用率 (%)	内存使用 (MB)
500	178.22	1.7	13
1000	356.45	3	15
5000	1782.23	15.3	23
10000	3564.45	31.6	25

单核处理能力对比



	tps	inflow (KB/s)
logtail	31600	11263.66
logstash	2070	737.98
fluentd	5320	1889.96

总结

可以看到三款日志工具各有特点：

- Logstash支持所有主流日志类型，插件支持最丰富，可以灵活DIY，但性能较差，JVM容易导致内存使用量高。
- Fluentd支持所有主流日志类型，插件支持较多，性能表现较好。
- Logtail占用机器CPU/内存资源最少，性能吞吐量较好，针对常用日志场景支持全面，但缺少插件等机制，灵活性和可扩展性不如以上两个客户端。

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

在日志服务管理控制台，单击目标项目右侧的**管理**。

选择目标日志库并单击日志索引列下的**查询**。

单击右上角的**索引属性** > **开启索引**。

填写索引配置信息。

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mehtod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 __time__- __time__% 300
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 __time__- __time__% 300
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 url,count(1) as pv group by url order by pv desc limit 10
```

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 url_extract_host(http_referer) ,count(1) group by url_extract_host(http_referer)
```

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)
```

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

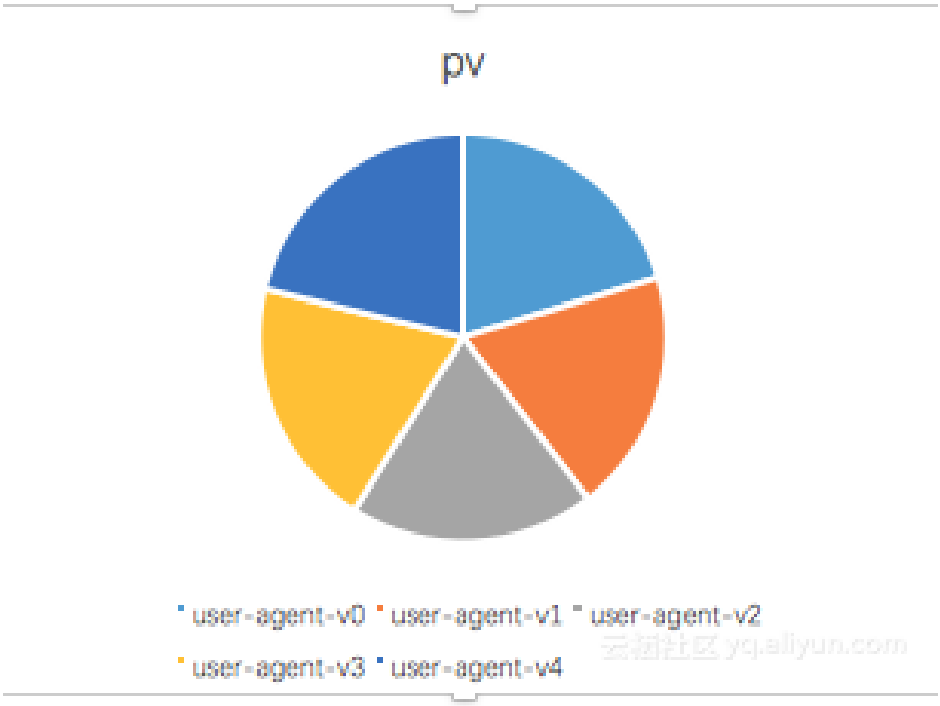
由统计结果可知，/0这个页面的访问延时最大，为了对/0页面进行调优，接下来需要统计/0这个页面的访问PV、U、各种method次数、各种status次数、各种浏览器次数、平均延时和最大延时。

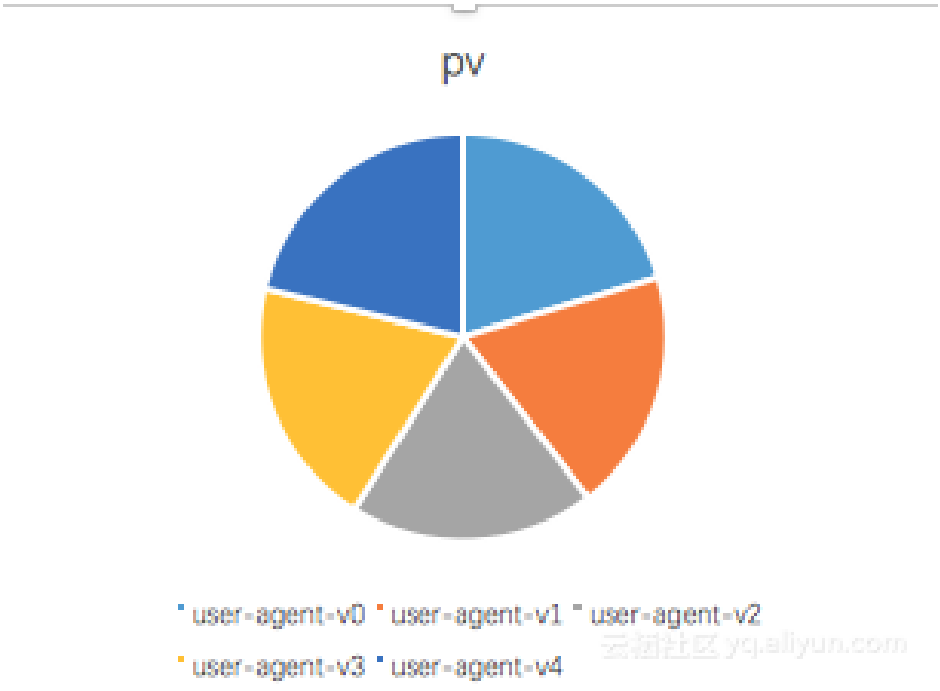
统计语句：

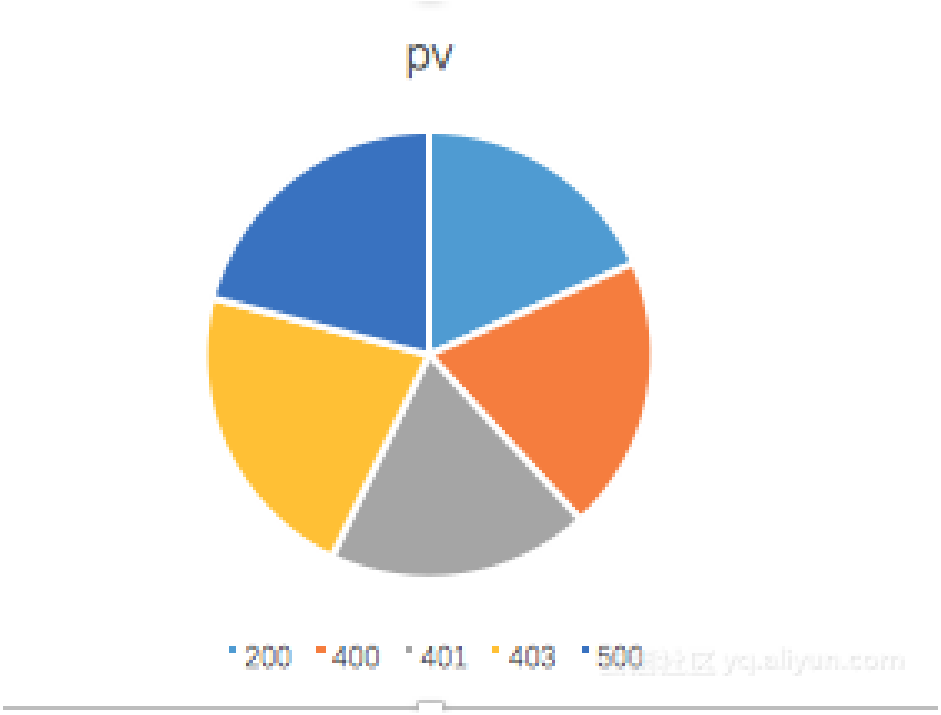
```
  url:"/0"|select count(1) as pv, approx_distinct(remote_addr) as uv, histogram(method) as
method_pv,histogram(status) as status_pv, histogram(user_agent) as user_agent_pv, avg(request_time)
```

```
as avg_latency,  max(request_time) as max_latency
```

统计结果：







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。