日志服务

用户指南

用户指南

日志服务中日志为日志服务中处理的最小数据单元，采用半结构化数据模式定义一条日志，具体数据模型包括主题（Topic）、时间（Time）、内容（Content）和来源（Source），详细描述请参考产品简介。

日志主题（Topic）为用户自定义字段，用以标记一批日志，一个日志库内的日志可以通过日志Topic来划分。用户可以在写入时指定日志Topic，并在查询时指定查询的日志Topic。例如，访问日志根据不同站点进行标记、一个平台用户可以使用用户编号作为日志Topic写入日志。这样在查询时可利用日志主题让不同用户仅看到自己的日志。如果不需要划分一个日志库内的日志，让所有日志使用相同的日志Topic即可。默认值为空字符串，空字符串也为一个有效的Topic。

您可以通过控制台设置或者修改Topic。

注意：syslog不支持配置Topic。

Topic生成方式

用户可以在Logtail收集日志时设置Topic，也可以使用API/SDK上传数据时设置Topic。目前支持通过控制台设置Topic生成方式为**空-不生成Topic**、**机器组Topic属性**和**文件路径正则**。

空-不生成Topic

通过控制台配置Logtail收集文本文件时，日志Topic生成方式默认为**空-不生成Topic**，即Topic为空字符串，在查询日志时不需要输入Topic即可查询。

机器组Topic属性

机器组Topic属性方式用于明确区分不同服务器产生的日志数据。如果您的不同服务器日志数据均保存在相同名称的所在的文件路径和文件名中，当您需要在收集日志时通过Topic区分不同服务器的日志数据，可以将机器分为不同的机器组，即在创建机器组时，为不同的机器组设置不同的**Topic属性**，并设置**Topic生成方式**为**机器组Topic属性**。将两个机器组应用之前创建的Logtail配置后，即完成对应配置。

如选择**机器组Topic属性**，Logtail上报数据时会将机器所在机器组的Topic属性作为主题名称上传至日志服务，在使用**日志索引分析**功能查询时需要指定Topic，即需要指定目标机器组Topic属性为查询条件。

文件路径正则

文件路径正则方式用于区分具体用户或实例产生的日志数据。如果服务日志根据不同的用户或者实例将日志记录在不同目录下面，但是只要下级目录下同、日志文件名称相同，日志服务在收集日志文件时就无法明确区分日志内容是由那个用户或实例产生的。此时可以设置**Topic生成方式**为**文件路径正则**，并且输入文件路径的正则表达式，配置Topic为实例名称。

当选择**文件路径正则**主题生成方式时，Logtail上报数据时会将实例名称作为主题名称上传至日志服务。根据您的目录结构和配置，会生成不同的Topic，在使用**日志索引分析**功能查询时需要指定主题名称为实例名称。

设置日志Topic

根据使用Logtail采集文本文件，通过控制台配置Logtail。

如您需要配置Topic生成方式为**机器组Topic属性**，请在创建机器组/修改机器组页面中配置**机器组Topic**。

在Logtail配置页面中，展开**高级选项**，在**Topic生成方式**中选择Topic的生成方式。

修改日志Topic

如您需要修改日志Topic的生成方式，请直接在Logtail配置界面修改**Topic生成方式**选项。

注意：修改后的配置仅对生效后采集的新数据有效。

项目

您可以通过日志服务管理控制台创建项目（Project）。

注意：目前，日志服务仅提供控制台方式创建Project。

前提条件

开始使用日志服务之前，您需要使用主账号开通日志服务，并且创建AccessKey。创建AccessKey的详细操作，请参考5分钟快速入门中**创建秘钥对**步骤。

使用说明

- Project的名字需要全局唯一（在所有阿里云Region内）。如果您选择的Project名称已经被别人使用，您会收到页面提醒信息“**Project XXX already exist**”，请您更换一个Project名称重试。
- Project 创建时需要指定所在的阿里云Region。您需要根据需要收集的日志来源和其他实际情况选择合适的阿里云Region。如果您需要收集来自阿里云ECS虚拟机的日志，建议在ECS虚拟机相同的Region创建Project。这样既可以加快日志收集速度，还可以使用阿里云内网（不占用ECS虚拟机公网带宽）收集日志。
- Project一旦创建完成则无法改变其所属Region，且日志服务目前也不支持 Project 的迁移，所以请谨慎选择Project的所属Region。
- 一个用户在所有阿里云Region总计最多可创建30个Project。

操作步骤

登录 日志服务管理控制台。

单击右上角的 **创建Project**。

填写 **Project名称** 和 **所属地域**，单击 **确认**。

Project名称：项目名称只能包含小写字母、数字和连字符（-），且必须以小写字母和数字开头和结尾，长度不能超过 3~63 个字节。

注意：项目名称创建后不能修改。

所属地域：您需要为每个项目指定阿里云的一个地域（Region），且创建后就不能修改地域，也不能在多个地域间迁移项目。

后续操作

创建项目后，系统会提示您创建日志库，您可以单击 **创建** 创建日志库或者单击 **取消** 进入项目列表页面。有关如何创建日志库，参见 **创建日志库**。

您还可以 **查看项目列表**，**访问项目**，**管理项目** 或 **删除项目**。

查看项目列表

您可以查看您的日志服务项目列表。

您可以通过项目列表操作相应的项目，具体如下：

- 管理项目：单击项目名称可以对该项目进行日志库管理和 **机器组管理**。
- 删除项目：单击项目列表右边的 **删除** 按钮可删除当前项目。
- 修改注释：单击项目列表右边的 **修改注释** 按钮可以修改当前项目的注释。

另外，您可以将鼠标移动到项目名称上显示该项目的更多细节，如下图所示。

Project是日志服务的资源管理单元，您可以通过Project来管理各种日志库和需要采集日志的机器。

> **注意**：目前，日志服务仅提供控制台方式帮助您管理Project。您可以通过控制台完成如上所述的所有管理操作。

具体来说，您可以通过如下操作管理日志服务的Project：

创建和删除Project内的日志库（Logstore）。

日志库是日志服务内的日志存储单元，它用于存储一类日志。而一个用户实际项目中可能需要采集的日志类型可能有多种，如前端Web服务器的访问日志（Access log），后端应用程序生成的应用日志（Application log）等。用户则可以在Project创建独立的日志库并把不同类型的日志写入不同的日志库。关于日志库的更多信息请参考修改日志库配置。

管理采集日志的机器。

您的日志会在各个不同的日志源产生，其中最为常见的是服务器。Project以机器组（Machine group）的方式帮助您管理需要采集日志的服务器。您可以在一个Project中创建和删除机器组，通过服务器IP地址或标识把需要采集的服务器归类到相应的机器组中。常见场景为：将需要写入一个Logstore的所有日志源（如所有前端Web服务器）加入到一个机器组进行管理。

删除项目

在某些情况下（如关闭日志服务，销毁 project 的所有日志等），您可能需要删除整个 project。日志服务允许您在控制台上方便地删除整个 project。

> **注意**：当您的 project被删除后，其管理的所有日志数据及配置信息都会永久释放，不可恢复。所以，在删除 project 前请仔细确认，避免数据丢失。

操作步骤

登录 日志服务管理控制台。

在 project 列表中，选择需要删除的项目。

单击右侧的 **删除**。

日志库

目前，日志服务支持在控制台或者通过API创建Logstore。使用API创建Logstore 请参考 CreateLogstore。

使用说明

- 任何一个Logstore必须在某一个Project下创建。
- 每个日志服务项目可创建最多100个日志库。
- Logstore名称在其所属项目内必须唯一。
- 数据保存时间创建后还可以进行修改。您可以在Logstore页面，单击**操作**列下的 **修改**，修改 **数据保存时间** 并单击 **修改**，然后关闭对话框即可。

操作步骤

在 **Project列表** 页面，单击项目的名称，然后单击 **创建** 创建日志库。

或者在创建完项目后，根据系统提示创建日志库，单击 **创建**。

填写日志库的配置信息并单击 **确认**。

Logstore名称：日志库名称只能包含小写字母、数字和连字符（-），且必须以小写字母和数字开头和结尾，长度为3~63个字节。Logstore名称在其所属项目内必须唯一。

注意：日志库名称创建后不能修改。

数据保存时间：数据在日志库中的保存时间，单位为天。可以设置为1~365天。超过该时间后，数据会被删除。

Shard数目：设置日志库的分区数量，每个Logstore可以创建1~10个分区。每个Project中可以创建最多200个分区。

后续操作

创建完日志库后，系统会提示您创建Logtail 配置，您可以单击 **创建Logtail配置** 创建一个Logtail 配置或者单击 **取消** 进入 **Logstore列表** 页面。

您可以通过日志服务管理控制台查看您的日志库列表。

操作步骤

登录 日志服务管理控制台。

选择所需的项目，单击项目名称或者选择所需的项目并单击右侧的 **管理**。

更多操作

具体来说，您可以通过日志库列表做如下操作：

- 单击列表中日志索引列下的 **查询** 可以进入日志查询页面，具体请参考 日志查询操作。
- 单击日志收集模式列下的Logtail配置 **管理** 可以创建Logtail配置、查看当前日志库下已经创建的Logtail配置、修改已有的Logtail配置。如果希望使用 Logtail收集日志并写入该日志库，您需要配置此项，具体请参考 Logtail 收集日志。
- 单击日志收集模式列下的 **更多**，可以在下拉菜单中单击 **Logstash**、**API** 或 **SDK**，查看日志服务的Logstash文档、API文档 和SDK文档，通过Logstash、API或者SDK向该日志库写入日志。
- 单击日志消费列下的 **预览** 查看日志信息或单击 **修改** 修改日志数据保存的时间以及日志库的Shard个数。您可以对分区进行 **分裂** 或 **合并** 操作。
- 单击日志投递列下的**OSS**，开启和设置OSS日志投递。更多信息请参考OSS日志投递。
- 单击操作列下的**修改**可以修改数据保存时间和Shard数目等。
- 单击日志库列表中最右边的 **删除** 删除对应的日志库。

创建日志库以后，您还可以在需要的时候修改日志库的配置。

操作步骤

登录 日志服务管理控制台。

选择所需的项目，单击项目名称。

在 **Logstore列表** 页面，选择所需的日志库并单击操作列下的 **修改**。

在弹出的对话框中修改日志库的配置并关闭对话框。

在某些情况下（如希望废弃某个Logstore），您可能需要删除指定的Logstore。日志服务允许您在控制台上删除Logstore。

注意事项

- 一旦Logstore删除，其存储的日志数据将会永久丢弃，不可恢复，请谨慎操作。
- 删除指定Logstore前必须删除其对应的所有Logtail配置。
- 如果该Logstore上还启用了日志投递的消费模式，则不保证删除前Logstore里的所有数据都会成功投递到MaxCompute中。如果您需要保证被删除的Logstore内所有数据都能投递到MaxCompute请按照如下流程操作：
    i. 删除前先停止向该Logstore写入新日志。
    ii. 确认Logstore里的所有日志数据都成功导入到MaxCompute。
    iii. 删除Logstore。

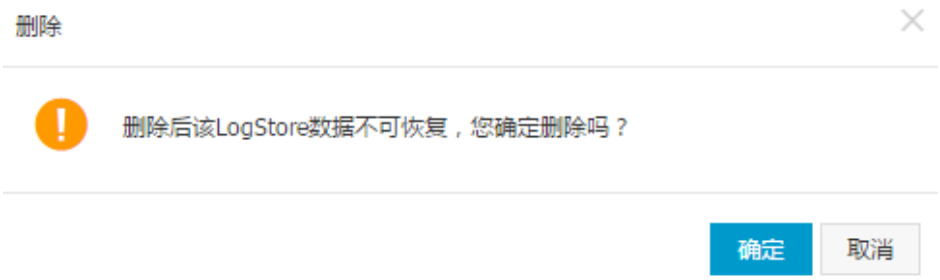
操作步骤

登录 日志服务管理控制台。

选择所需的项目，单击项目名称或者单击右侧的 **管理**。

在 **Logstore列表** 页面，选择要删除的日志库并单击右侧的 **删除**。

在弹出的确认对话框中，单击 **确定**。



分区

Logstore读写日志必定保存在某一个分区（Shard）上。每个日志库（Logstore）分若干个分区，每个分区由MD5左闭右开区间组成，每个区间范围不会相互覆盖，并且所有的区间的范围是MD5整个取值范围。

分区范围

创建Logstore时，指定分区个数，会自动平均划分整个MD5的范围。每个分区均有范围，可用MD5方式来表示，且必定包含于以下范围中：[00000000000000000000000000000000,ffffffffffffffffffffffffffffffff）。

分区的范围均为左闭右开区间，由以下Key组成：

- BeginKey：分区起始的Key值，分区范围中包含该Key值
- EndKey：分区结束的Key值，分区范围中不包含该Key值

分区的范围用于支持指定Hash Key的模式写入，以及分区的分裂和合并操作。在向分区读写数据过程中，读必须指定对应的分区，而写的过程中可以使用负载均衡模式或者指定Hash Key的模式。负载模式下，每个数据包随机写入某一个当前可用的分区中，在指定Hash Key模式下，数据写入分区范围包含指定Key值的分区。

例如，某Logstore共有4个分区，且该Logstore的MD5取值范围是[00,FF)。各个分区范围如下表所示。

分区号	范围
Shard0	[00,40)
Shard1	[40,80)
Shard2	[80,C0)
Shard3	[C0,FF)

当写入日志时，通过指定Hash Key模式指定一个MD5的Key值是5F，日志数据会写入包含5F的Shard1分区上；如果指定一个MD5的Key值是8C，日志数据会写入包含8C的Shard2分区上。

分区的读写能力

每个分区可提供一定的服务能力：

- 写入：5MB/s，2000次/s
- 读取：10MB/s，100次/s

建议您根据实际数据流量规划分区个数，流量超出读写能力时，及时分裂分区以增加分区个数，从而达到更大的读写能力；如您的流量远远达不到分区的最大读写能力时，建议您合并分区以减少分区个数，从而节约分区租赁费用。

例如，如果您的有两个readwrite状态的分区，最大可以提供10MB/s的数据写入服务，但如果您实时写入数据流量达到14MB，建议分裂其中一个分区，使readwrite分区数量达到3个。如果您实施写入数据流量仅为3MB/s，那么一个分区即可满足需要，建议您合并两个分区。

注意：

- 当写入的API持续报告403或者500错误时，通过 Logstore云监控查看流量和状态码 判断是否需要增加分区。
- 对超过分区服务能力的读写，系统会尽可能服务，但不保证服务质量。

分区的状态

分区的状态包括：

- readwrite：可以读写

  readonly：只读数据

  创建分区时，所有分区状态均为readwrite状态，**分裂**或**合并**操作会改变分区状态为readonly，并生成新的readwrite分区。分区状态不影响其数据读取的性能，同时，readwrite分区保持正常的数据写入性能，readonly状态分区不提供数据写入服务。

  在**分裂**分区时，需要指定一个处于readwrite状态的ShardId和一个MD5。MD5要求必须大于分区的BeginKey并且小于EndKey。分裂操作可以从一个分区中分裂出另外两个分区，即分裂后分区数量增加2。在分裂完成后，被指定分裂的原分区状态由readwrite变为readonly，数据仍然可以被消费，但不可写入新数据。两个新生成的分区状态为readwrite，排列在原有分区之后，且两个分区的MD5范围覆盖了原来分区的范围。

  在**合并**操作时，必须指定一个处于readwrite状态的分区，指定的分区不能是最后一个readwrite分区。服务端会自动找到所指定分区的右侧相邻分区，并将两个分区范围合并。在合并完成后，所指定的分区和其右侧相邻分区变成只读（readonly）状态，数据仍然可以被消费，但不能写入新数据。同时新生成一个 readwrite 状态的分区，新分区的MD5范围覆盖了原来两个分区的范围。

通过日志服务管理控制台您可以进行以下分区操作：

- 扩容分区
- 缩容分区
- 删除分区

每个分区能够处理5M/s的数据写入和10M/s的数据读取，当数据流量超过分区服务能力时，建议您及时增加分区。扩容分区通过分裂（split）操作完成。

使用指南

在分裂分区时，需要指定一个处于readwrite状态的ShardId和一个MD5。MD5要求必须大于分区的BeginKey并且小于EndKey。

分裂操作可以从一个分区中分裂出另外两个分区，即分裂后分区数量增加2。在分裂完成后，被指定分裂的原分区状态由readwrite变为readonly，数据仍然可以被消费，但不可写入新数据。两个新生成的分区状态为readwrite，排列在原有分区之后，且两个分区的MD5范围覆盖了原来分区的范围。

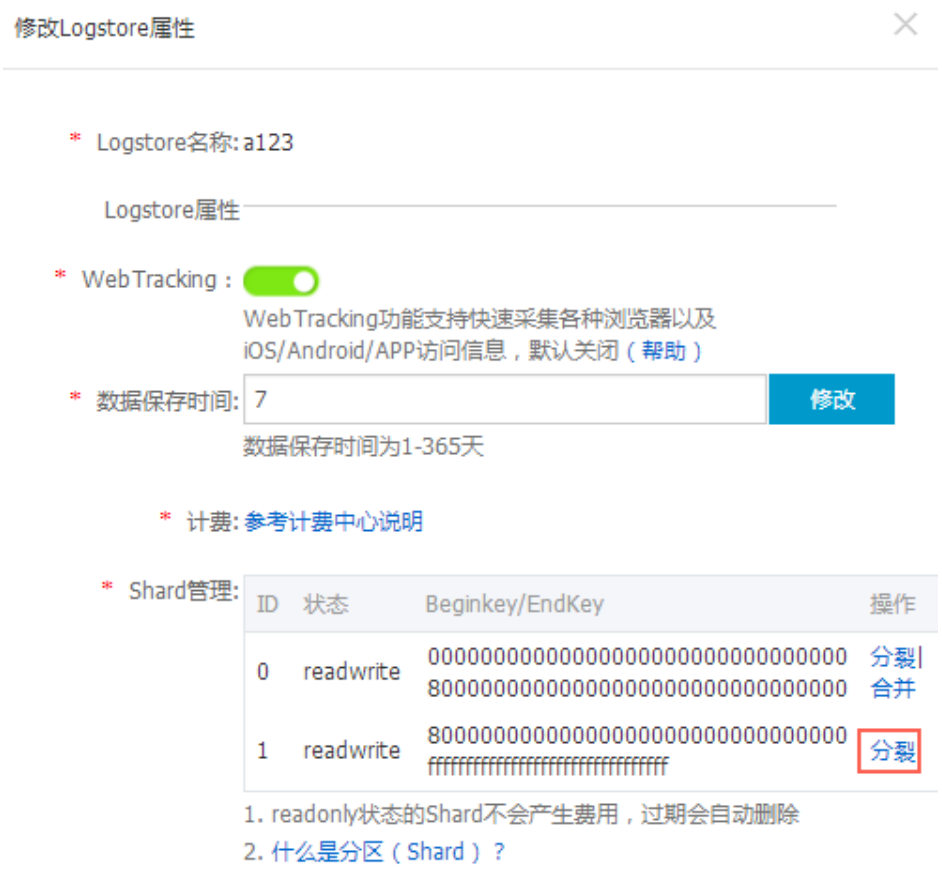
操作步骤

登录 日志服务管理控制台。

选择所需的项目，单击项目名称。

在 **Logstore列表** 页面，选择所需的日志库并单击日志消费列下的 **修改**。

选择要分裂的分区，单击右侧的 **分裂**。



单击 **确认** 并关闭对话框。

分裂操作完成后，原分区变为readonly状态，两个新生成的分区的MD5范围覆盖了原来分区的范围。

您可以通过合并（merge）操作缩容分区。合并操作可以将指定分区与其右侧相邻分区的范围合并，并将此范围赋予新生成的readwrite分区，两个被合并的原分区变为readonly状态。

使用指南

在合并操作时，必须指定一个处于readwrite状态的分区，指定的分区不能是最后一个readwrite分区。服务端会自动找到所指定分区的右侧相邻分区，并将两个分区范围合并。在合并完成后，所指定的分区和其右侧相邻分区变成只读（readonly）状态，数据仍然可以被消费，但不能写入新数据。同时新生成一个 readwrite 状态的分区，新分区的MD5范围覆盖了原来两个分区的范围。

操作步骤

登录 日志服务管理控制台。

选择所需的项目，单击项目名称或者单击右侧的 **管理**。

在 **Logstore列表** 页面，选择所需的日志库并单击操作列下的 **修改**。

选择要合并的分区，单击右侧的 **合并** 并关闭对话框即可。

在合并完成后，所指定的分区和其右侧相邻分区变成只读（readonly）状态，，新生成的readwrite分区的MD5范围覆盖了原来两个分区的范围。

Logstore的生命周期即数据保存时间支持1-365天，分区及分区中的日志数据在超出该时间后会自动删除。readonly 分区不参与计费。

实时采集

LogHub 支持客户端、网页、协议、SDK/API（移动、游戏）等多种日志采集方式，所有采集方式均基于Restful API实现，除此之外您也可以通过API/SDK实现新的采集方式。

通过客户端采集

- Logtail：LogHub自带Agent

- 完整支持Windows/Linux所有平台，中心化管理与Web端控制，高性能/低消耗。
- 支持多种解析模式：极简（单行）；Json 格式；Delimiter 格式；正则表达式提取。常见配置示例： Nginx， Apache， Log4J ，Wordpress ， Python， NodeJS， 分隔符（Delimiter, 如 CSV、TSV 等格式）、Logstash ， ThinkPHP。
- 支持Rsyslog协议。

- logstash

面向容器（Docker）

- 阿里云容器服务
- 其他容器
  - 通过Logtail进行采集：参见 使用Logtail采集日志/Docker 部分。
  - 通过Docker Driver进行采集：集成后Docker完整代码，细节参见Docker/Deamon/Driver下Alilog部分；Driver相关Commit。
  - FluentBit （客户提供）。

通过 SDK/API/移动端

- Java
- LogHub Producer Library（Java）: 客户端高并发写入
- Log4J Appender：LogHub Producer Library 包装 Log4J Appender
- Python
- PHP
- C#
- Go
- Native C
- NodeJs
- C++
- Android
- iOS
- JS/Web Tracking：通过匿名方式进行数据采集

阿里云云产品

- 云服务器（ECS）日志：通过Logtail采集
- 阿里云容器服务
- 消息服务（MNS）日志
- 函数服务(FC)：程序日志自动关联
- 即将打通：对象存储 （OSS）、内容分发（CDN）、API Gateway

网络选择

网络

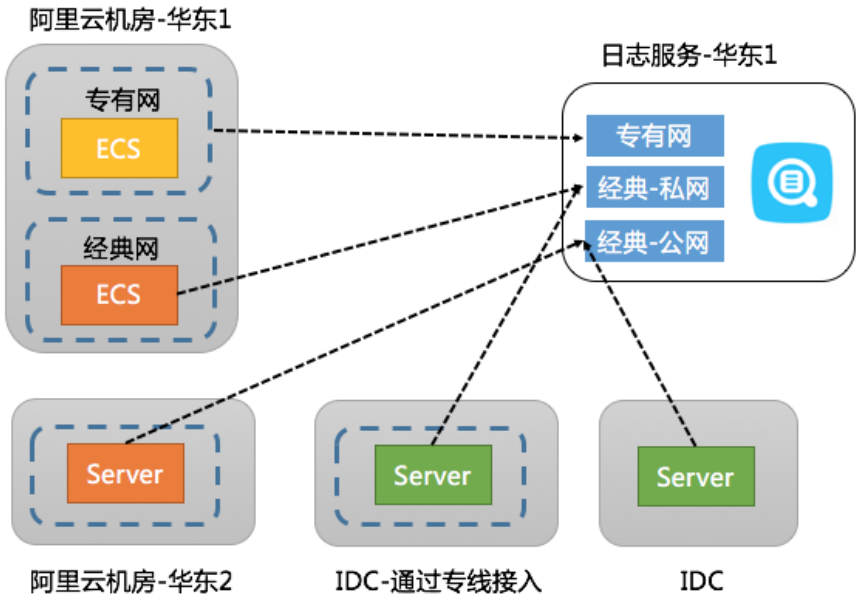
日志服务（LOG）提供各种内网/私网/公网等访问点，非常便捷地将混合网络环境中数据接入并融合。

日志服务（LOG）在各 Region 提供 **访问点**，每个 Region 提供三种方式接入点：

- 内网（经典网）：本 Region 内服务访问，带宽链路质量最好（**推荐**）
- 公网（经典网）：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

示例

假设日志服务创建在华东 1，则不同网络下接入图例如下：



详细方案如下：

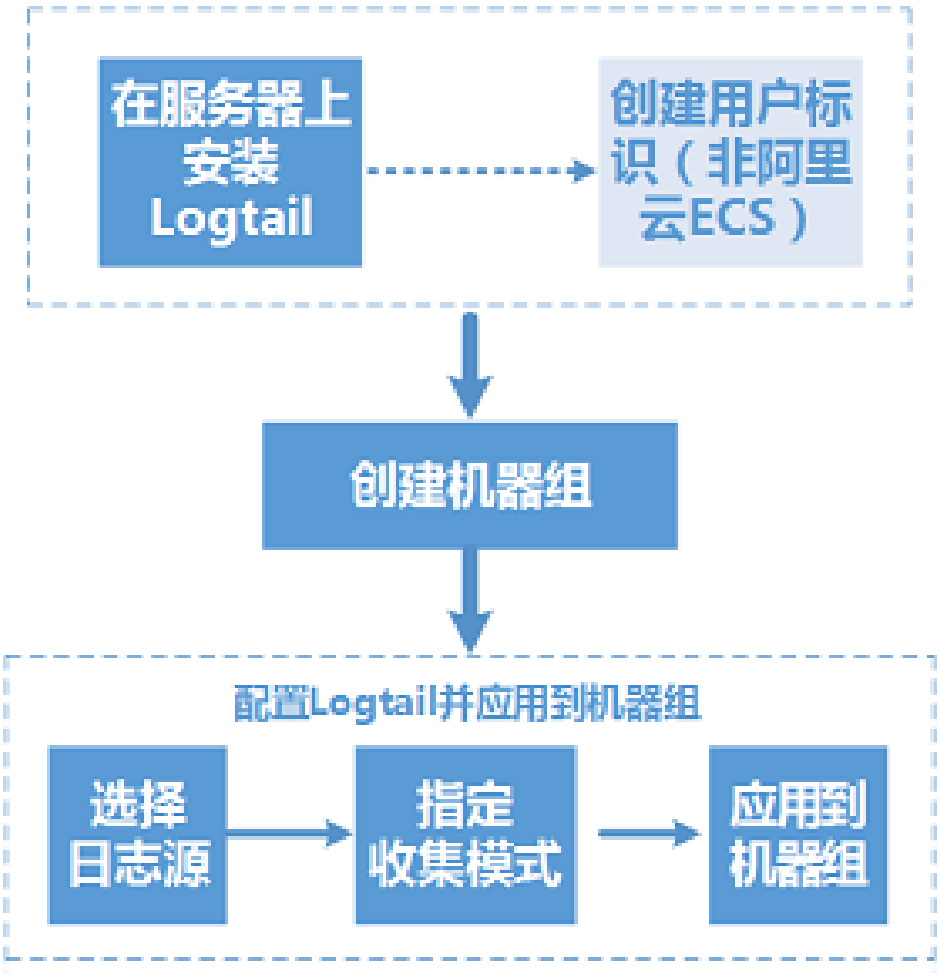
数据源	网络接入类型	网络方案
华东 1	ECS 经典网	内网
华东 1	ECS VPC	私网
华东 2	ECS 经典网/VPC	公网（HTTPS）
IDC	与华东 1 拉专线	内网
IDC		公网（HTTPS）

接入参考

- logtail: 在 Windows 上安装 logtail 和 在 Linux 上安装 logtail
    - 非 ECS 机器，需要在机器上 touch 用户标识，以表明这台机器属于对应的用户，可以收集该机器上日志，参见 非阿里云 ECS 配置用户标识。
    - 在 VPC（专有网）下，如果机器组通过 IP 无法标识，可以使用自定义标识来管理机器。标识的作用相当于把机器分为 web-server，app-server 等机器分组，可以自动扩容。
- API/SDK：配置接入点。
- Tracking Pixel/Android/IOS SDK：走公网。
- syslog：参见上边 logtail 的介绍。

Logtail接入服务是日志服务提供的日志采集Agent，通过控制台方式帮助您实时采集阿里云ECS等服务器上的日志。

配置流程



通过Logtail采集服务器日志可以通过以下步骤完成：

1. 安装Logtail。在需要采集日志的源服务器上安装Logtail操作请参见安装Logtail（Windows）和 安

装Logtail（Linux）。
2. 创建用户标识（非阿里云ECS）。从阿里云ECS采集日志不需要执行此步骤。
3. 创建机器组。日志服务通过机器组的方式管理所有需要通过Logtail客户端采集日志的服务器。日志服务支持通过IP或者自定义标识的方式定义机器组。您也可以在应用Logtail配置到机器组时，根据提示创建机器组。
4. 创建Logtail配置，并应用到机器组。您可以通过创建Logtail配置以 采集文本文件或 收集syslog日志，并将该Logtail配置应用到机器组。

您可以参考 样例 了解如何配置Logtail收集配置中的日志提取规则。

在完成如上流程后，您的ECS服务器上需要收集的新增日志会被主动收集、发送到对应Logstore中，历史数据不会被收集。您可以通过日志服务控制台或者SDK及API查询到这些日志。您还可以通过日志服务控制查询到所有ECS服务器上的Logtail收集日志状态，例如是否在正常收集，是否有错误等。

Logtail接入服务在日志服务控制台上的完整操作请参考 Logtail 收集日志。

Docker

- 阿里云容器服务：参见 集成日志服务
- ECS/IDC自建Docker（需要把容器中日志目录Mount到宿主机上）
    i. 安装Logtail（Windows） 和 安装Logtail（Linux）。
    ii. 将容器中日志目录Mount到宿主机目录。
        - 选择 1：使用命令（例如宿主机目录为 /log/webapp，容器中日志目录为 /opt/webapp/log）。

```
docker run -d -P --name web -v /src/webapp:/opt/webapp training/webapp python app.py
```

        - 选择 2：使用编排模板Mount。

注意：建议您修改Logtail启动参数，更改Logtail的checkpoint保存地址并mount到宿主机，防止容器释放时因丢失checkpoint信息而产生重复采集。

核心概念

机器组：一个机器组包含一或多台需要收集一类日志的机器。通过绑定一组Logtail配置到一个机器组，可以让日志服务根据同样的Logtail配置采集一个机器组内所有服务器上的日志。您也可以通过日志服务控制台方便地对机器组进行管理（包括创建、删除机器组，添加、移除机器等）。同一个机器组内不可同时包含 Windows和 Linux机器，但可以包含不同版本的Windows Server或者不同发行版本的Linux机器。

Logtail客户端：Logtail是运行在需要收集日志的服务器上上执行日志收集工作的Agent。请参照 安装Logtail（Windows） 和 安装Logtail（Linux）。 在服务器上安装Logtail后，需要配置Logtail并应用到机器组。

- **Linux** 下，Logtail安装在 /usr/local/ilogtail 目录下并启动两以 ilogtail 开头的个独立进程，一个为收集进程，另外一个为守护进程，程序运行日志为 /usr/local/ilogtail/ilogtail.LOG。
- **Windows** 下，Logtail安装在目录 C:\Program Files\Alibaba\Logtail（32 位系统）或 C:\Program Files (x86)\Alibaba\Logtail（64 位系统）。您可以通过Windows管理工具-服务查看到两个Windows Service，LogtailWorker负责收集日志，LogtailDaemon负责守护工作程序。程序运行日志为安装目录下的 logtail_*.log。

Logtail配置：是Logtail收集日志的策略集合。通过为Logtail配置数据源、收集模式等参数，来对机器组内所有服务器进行定制化的收集策略。描述如何在机器上收集一类日志并解析、发送到日志服务的指定日志库。您可以通过控制台对每个Logstore添加Logtail配置，表示该Logstore接收以此Logtail配置收集的日志。

基本功能

Logtail接入服务提供如下功能：

实时收集日志：动态监控日志文件，实时地读取、解析增量日志。日志从生成到发往服务端一般在3秒延迟内。

> **注意**：Logtail接入服务不支持对历史数据的收集。对于一条日志，读取该日志的时刻减去日志产生的时刻，差值超过5分钟的会被丢弃；

自动处理日志轮转：很多应用会按照文件大小或者日期对日志文件进行轮转（rotation），把原日志文件重命名，并新建一个空日志文件等待写入。例如：监控app.LOG，日志轮转会产生 app.LOG.1，app.LOG.2 等。您可以指定收集日志写入的文件，如 app.LOG，Logtail会自动检测到日志轮转过程，保证这个过程中不会出现日志数据丢失。

> **注意**：如果日志文件秒级别时间范围内多次发生轮转，可能会丢失数据。

自动处理收集异常：因为服务端错误、网络措施、Quota超限等各种异常导致数据发送失败，Logtail会按场景主动重试。如果重试失败则会将数据写入本地缓存，稍后自动重发。

> **注意**：本地缓存位于用户服务器的磁盘上，如果本地缓存的数据24小时内仍无法成功被服务端接收，缓存数据会被丢弃并从本地缓存删除。

灵活配置收集策略：可以通过Logtail配置来非常灵活地指定如何在一台ECS服务器上收集日志。具体来说，您可以根据实际场景选择日志目录，文件（即可精确匹配，也可通过通配符模糊匹配）。您可以自定义日志收集提取的方式和各个提取字段的名称（支持正则表达式方式的日志提取）。另外，由于日志服务日志数据模型要求每条日志必须有精确的时间戳信息，Logtail提供了自定义的日志时间格式，方便您从不同格式的日志数据中提取必须要的日志时间戳信息。

自动同步收集配置：您在日志服务控制台上新建或更新配置，Logtail一般在3分钟时间内即可自动接受并使之生效，更新配置过程中数据收集不丢失。

自动升级客户端：在您手动安装Logtail到服务器后，日志服务负责Logtail 自动运维升级，此过程无需您参与。在整个Logtail升级过程中日志数据不丢失。

自我监控状态：为避免Logtail客户端消耗您太多资源而影响您其他服务。Logtail客户端会实时监控自身CPU和内存消耗。如果Logtail客户端在运行过程中，资源使用超出限制将会自动重启，避免影响机器上的其它作业。同时，该客户端也会有主动的网络限流保护措施，防止过度消耗用户带宽。

注意：

- Logtail客户端在重启期间日志数据可能会丢失。
- 如果Logtail客户端自身处理逻辑出现异常导致退出，相应的保护机制会触发并重新启动该客户端继续收集日志。但在重新启动之前的日志数据可能丢失。

签名数据发送：为保证您的数据在发送过程中不会被篡改，Logtail客户端会主动获取用户的阿里云访问秘钥并对所有发送日志的数据包进行数据签名。

注意： Logtail客户端在获取您的阿里云访问秘钥时采用HTTPS通道，保障您的访问秘钥安全性。

功能优势

- 基于日志文件、无侵入式的收集日志。用户无需修改应用程序代码，且日志收集不会影响用户应用程序的运行逻辑。
- 能够稳定地处理日志收集过程中各种异常。当遇到网络异常、服务端异常，用户数据临时超预留写入带宽限制等问题时会主动重试、本地缓存等措施保障数据安全。
- 基于服务端的集中管理能力。用户在安装Logtail后（参见 安装Logtail（Windows） 和 安装Logtail（Linux）），只需要在服务端集中配置需要收集的机器、收集方式等信息即可，无需逐个登录服务器进行配置。
- 完善的自我保护机制。为保证运行在客户机器上的收集Agent不会明显影响用户自身服务的性能，Logtail客户端在CPU、内存及网络使用方面都做了严格的限制和保护机制。

处理能力与限制

Logtail接入服务在每台服务器上的处理能力及限制如下：

项目	能力与限制
文件编码	支持UTF8/GBK编码日志文件，如果日志文件是其它编码则会出现乱码、丢数据等未定义行为。建议使用UTF8编码以获得更好的处理性能。
日志处理吞吐能力	原始日志流量默认限制为1MB/s，通过阿里云内部网络发送数据。超过该日志流量则有可能丢失日志，可根据 文档 调整参数，最大支持约50MB/s。
网络错误处理	支持本地缓存，最多使用500MB本地存储空间。在出现网络异常或者临时服务端超限，会主动缓存数据到本地并在之后尽快重试。
配置更新	用户的配置更新生效的延时约30秒。
状态自检	支持异常情况下自动重启，例如程序异常退出及使用资源超限等。
监控目录数	主动限制可以监控的目录格式，避免出现过多消耗用户资源。如果监控上限已到，则放弃监控更多目录和日志文件。限制最多3000个目录（含子目录）。
软链接支持	支持监控目录为软链接。
日志文件大小	无限制。
单条日志大小	单条日志大小限制为512KB。超出512KB的日志将不被采集。多行日志按行首正则表达式划分后，每条日志大小限制仍为512KB。
正则表达式类型	使用Perl兼容正则表达式。

使用 Logtail 采集日志

使用 Logstash 采集日志

安装包说明

日志服务提供了一个基于Logstash-2.2.2版本且集成 JRE1.8、日志服务写出插件、NSSM 2.24的安装包，部署步骤相较于**自定义安装** 更加简洁，如有复杂需求可以选择自定义安装。

安装方法

下载**安装包**后解压缩到 C 盘。

确认Logstash的启动程序路径为 C:\logstash-2.2.2-win\bin\logstash.bat。

除了可以快速安装Logstash，您还可以根据需要进行自定义安装。

步骤 1 安装 Java

下载安装包。

请进入 Java 官网 下载 JDK 并双击进行安装。

设置环境变量。

打开高级系统设置，新增或修改环境变量。

- **PATH:** C:\Program Files\Java\jdk1.8.0_73\bin
- **CLASSPATH:** C:\Program Files\Java\jdk1.8.0_73\lib;C:\Program Files\Java\jdk1.8.0_73\lib\tools.jar
- **JAVA_HOME:** C:\Program Files\Java\jdk1.8.0_73

验证。

执行 PowerShell 或 cmd.exe 进行验证：

```
PS C:\Users\Administrator> java -version
java version "1.8.0_73"
Java(TM) SE Runtime Environment (build 1.8.0_73-b02)
Java HotSpot(TM) 64-Bit Server VM (build 25.73-b02, mixed mode)
PS C:\Users\Administrator> javac -version
javac 1.8.0_73
```

步骤 2 安装Logstash

下载安装包。

官网下载：Logstash主页 选择 2.2 或以上版本。

安装。

解压 logstash-2.2.2.zip 到 C:\logstash-2.2.2 目录。

确认Logstash的启动程序路径是否正确：C:\logstash-2.2.2\bin\logstash.bat。

步骤 3 安装Logstash写日志服务插件

请根据机器所处网络环境决定在线或离线安装模式：

在线安装

该插件托管于 RubyGems，更多信息请点击查看 。

执行 PowerShell 或 cmd.exe，进入Logstash安装目录：

```
PS C:\logstash-2.2.2> .\bin\plugin install logstash-output-logservice
```

离线安装

官网下载：进入 logstash-output-logservice 页面，单击右下角 **下载** 按钮。

如采集日志机器无法访问公网，请拷贝下载的 gem 包到采集日志机器的 C:\logstash-2.2.2 目录。执行 PowerShell 或 cmd.exe，进入 Logstash 安装目录：

```
PS C:\logstash-2.2.2> .\bin\plugin install C:\logstash-2.2.2\logstash-output-logservice-0.2.0.gem
```

验证

```
PS C:\logstash-2.2.2> .\bin\plugin list
```

在本机已安装的插件列表中可以找到 logstash-output-logservice。

步骤 4 安装 NSSM

官网下载：进入nssm 官网 下载。

下载安装包到本地后，解压缩文件到目录 C:\logstash-2.2.2\nssm-2.24。

在 PowerShell 下启动 logstash.bat，Logstash进程会在前台工作，一般用于配置测试和采集调试。建议调试通过后把Logstash设置为 Windows Service，可以保持后台运行以及开机自启动。

出了将Logstash设置为Windows Service之外，您还可以通过命令行启动、停止、修改和删除服务。更多 NSSM 使用方法请参考 官方文档。

添加服务

一般用于首次部署时执行，如已添加过服务，请跳过该步骤。

您可以执行以下命令添加服务。

32 位系统

```
C:\logstash-2.2.2-win\nssm-2.24\win32\nssm.exe install logstash "C:\logstash-2.2.2-win\bin\logstash.bat" "agent -f C:\logstash-2.2.2-win\conf"
```

64 位系统

```
C:\logstash-2.2.2-win\nssm-2.24\win64\nssm.exe install logstash "C:\logstash-2.2.2-win\bin\logstash.bat" "agent -f C:\logstash-2.2.2-win\conf"
```

启动服务

如Logstash conf 目录后有配置文件更新，请先停止服务，再启动服务。

您可以执行以下命令启动服务。

32 位系统

```
C:\logstash-2.2.2-win\nssm-2.24\win32\nssm.exe start logstash
```

64 位系统

```
C:\logstash-2.2.2-win\nssm-2.24\win64\nssm.exe start logstash
```

停止服务

您可以执行以下命令停止服务。

32 位系统

```
C:\logstash-2.2.2-win\nssm-2.24\win32\nssm.exe stop logstash
```

64 位系统

```
C:\logstash-2.2.2-win\nssm-2.24\win64\nssm.exe stop logstash
```

修改服务

您可以执行以下命令修改服务。

32 位系统

```
C:\logstash-2.2.2-win\nssm-2.24\win32\nssm.exe edit logstash
```

64 位系统

```
C:\logstash-2.2.2-win\nssm-2.24\win64\nssm.exe edit logstash
```

删除服务

您可以执行以下命令删除服务。

32 位系统

```
C:\logstash-2.2.2-win\nssm-2.24\win32\nssm.exe remove logstash
```

64 位系统

```
C:\logstash-2.2.2-win\nssm-2.24\win64\nssm.exe remove logstash
```

相关插件

logstash-input-file

通过该插件tail方式收集日志文件，详细信息请参考 logstash-input-file。

> **注意：**path 填写文件路径时请使用 UNIX 模式的分隔符，如：C:/test/multiline/*.log，否则无法支持模糊匹配。

logstash-output-logservice

通过该插件可以 input 插件采集的日志写出到日志服务。

参数	说明
endpoint	如 http://cn-shenzhen.log.aliyuncs.com， 参考：日志服务入口
project	日志服务项目名称
logstore	日志库名称
topic	日志主题名称，默认设置空即可
source	日志来源，如为空则自动取本机 IP，否则以设置值为准
access_key_id	阿里云云账号秘钥 ID
access_key_secret	阿里云云账号秘钥 secret
max_send_retry	数据包发送日志服务发生异常时最大重试次数，重试不成功的数据包丢弃，重试间隔为 200 毫秒

配置步骤

1 创建采集配置

新增配置文件到 C:\logstash-2.2.2-win\conf\ 目录后，重启Logstash生效。

可以为每种日志新建一个配置文件，格式为 *.conf。建议统一保存于 C:\logstash-2.2.2-win\conf\ 目录下，以方便管理。

> **注意：**配置文件格式必须以 UTF-8 无 BOM 格式编码，可以通过 notepad++ 修改文件编码格式。

IIS 日志

请参考 IIS 日志配置。

CSV 日志

使用采集日志的系统时间作为上传日志时间，请参考 CSV 日志配置。

自带时间日志

以CSV日志格式为例，以日志内容中的时间作为上传日志时间，请参考 CSV 日志配置。

通用日志

默认使用采集日志的系统时间作为上传的日志时间，日志不解析字段，支持单行、多行日志格式。请参考 通用日志配置。

2 验证配置语法

执行 PowerShell 或 cmd.exe，进入Logstash安装目录：

```
PS C:\logstash-2.2.2-win\bin> .\logstash.bat agent --configtest --config C:\logstash-2.2.2-win\conf\iis_log.conf
```

修改收集配置文件，在 output 阶段临时添加一行 rubydebug 配置以输出采集结果到 Console。配置中 type 字段请自行设置。

```
output {
if [type] == "***" {
stdout { codec => rubydebug }
logservice {
...
}
}
}
```

执行 PowerShell 或 cmd.exe，进入Logstash安装目录启动进程：

```
PS C:\logstash-2.2.2-win\bin> .\logstash.bat agent -f C:\logstash-2.2.2-win\conf
```

验证完成后，请结束 logstash.bat 进程并删除 rubydebug 临时配置项。

后续操作

在 PowerShell 下启动 logstash.bat，logstash 进程会在前台工作，一般用于配置测试和采集调试。建议调试通过后把Logstash设置为 Windows Service，可以保持后台运行以及开机自启动。有关如何将Logstash设置

为 Windows Service，参见 配置 Logstash 为 Windows Service。

Logstash 提供了 大量插件，可以满足个性化需求，例如：

grok：通过正则表达式结构解析日志内容成多个字段。

json_lines、json：提供结构化解析 JSON 类型日志功能。

date：提供日志内容中有关日期、时间字段相关的解析、转换功能。

multiline：可自定义更为复杂的多行日志类型。

kv：提供结构化解析 Key-Value 类型日志格式功能。

通过Logstash收集日志数据时，如遇到以下收集错误，请按照对应建议进行处理。

日志服务查看到数据乱码

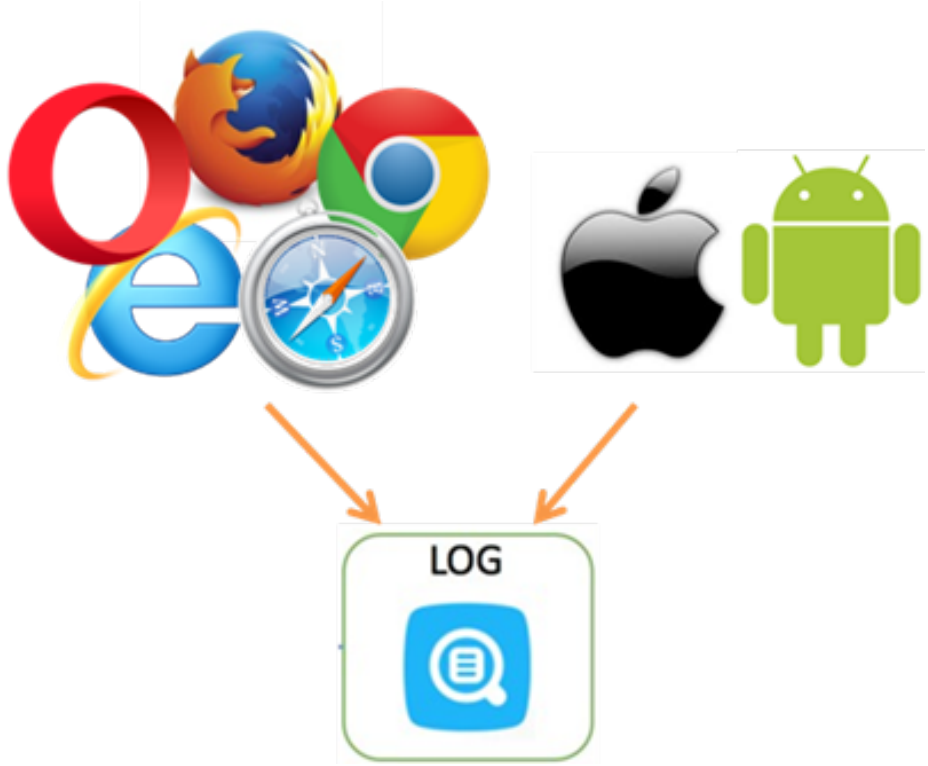
Logstash 默认支持 UTF8 格式文件编码，请确认输入文件编码是否正确。

控制台提示错误

控制台提示错误io/console not supported; tty will not be manipulated时， 不影响产品功能，请忽略。

其它错误类型建议建议参考Google或Logstash论坛，查询帮助信息。

日志服务支持通过Web Tracking功能进行HTML、H5、iOS和 Android平台日志数据的采集，支持自定义维度和指标。



如上图所示，使用Web Tracking功能可以采集各种浏览器以及iOS、Android APP的用户信息（除iOS/Android SDK 外），例如：

- 用户使用的浏览器、操作系统、分辨率等。
- 用户浏览行为记录，比如用户网站上的点击行为、购买行为等。
- 用户在APP中停留时间、是否活跃等。

注意：使用Web Tracking意味着该Logstore打开互联网匿名写入的权限，可能会产生脏数据，请留意。

配置步骤

1 开通Web Tracking

使用前，需要先开通Logstore的Web Tracking开关，目前在控制台上暂不支持可视化设置Logstore支持Web Tracking，如果要使用该功能，请先通过控制台或Java SDK开通。

通过控制台开通Web Tracking

在Logstore列表页面，选中需要开通Web Tracking功能的Logstore，单击右侧的**修改**。

打开 Web Tracking 开关。

通过 Java SDK 开通Web Tracking

使用 Java SDK：

```
import com.aliyun.openservices.log.Client;
import com.aliyun.openservices.log.common.LogStore;
import com.aliyun.openservices.log.exception.LogException;
public class WebTracking {
static private String accessId = "your accesskey id";
static private String accessKey = "your accesskey";
static private String project = "your project";
static private String host = "log service data address";
static private String logStore = "your logstore";
static private Client client = new Client(host, accessId, accessKey);
public static void main(String[] args) {
try {
//在已经创建的Logstore 上开通Web Tracking功能。
LogStore logSt = client.GetLogStore(project, logStore).GetLogStore();
client.UpdateLogStore(project, new LogStore(logStore, logSt.GetTtl(), logSt.GetShardCount(), true));
//关闭Web Tracking功能。
//client.UpdateLogStore(project, new LogStore(logStore, logSt.GetTtl(), logSt.GetShardCount(), false));
//新建支持Web Tracking功能的Logstore。
//client.UpdateLogStore(project, new LogStore(logStore, 1, 1, true));
}
catch (LogException e){
```

```
e.printStackTrace();
}
}
}
```

2 收集日志数据

Logstore开通Web Tracking功能后，可以使用以下三种方法上传数据到Logstore中。

使用HTTP GET请求

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

其中各个参数的含义如下：

字段	含义
${project}	您在日志服务中开通的Project名称。
${sls-host}	您日志服务所在地区的域名。
${logstore}	${project} 下面开通Web Tracking功能的某一个Logstore的名称。
APIVersion=0.6.0	保留字段，必选。
key1=val1、key2=val2	您要上传到日志服务的Key-Value对，可以有多个，但是要保证URL的长度小于16KB。

使用 HTML img 标签

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

各个参数的含义同上。

使用 js SDK

将 loghub-tracking.js 复制到 web 目录，并在页面中引入如下脚本：

点击下载

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

注意：为了不阻塞页面加载，脚本会异步发送 HTTP 请求，如果页面加载过程中需要多次发送数据，后面的请求会覆盖前面的 HTTP 请求，看到的现象是浏览器中会显示Tracking 请求退出。使用同

步发送可以避免该问题，同步发送请在脚本中执行如下语句替换：

原始语句：

```
this.httpRequest_.open("GET", url, true)
```

替换最后一个参数变成同步发送：

```
this.httpRequest_.open("GET", url, false)
```

创建Tracker对象。

第一个参数是endpoint、第二个是Project、第三个是 Logstore。

```
var logger = new window.Tracker('cn-hangzhou-staging-intranet.sls.aliyuncs.com','ali-test-
tracking','web-tracking');
logger.push('customer', 'zhangsan');
logger.push('product', 'iphone 6s');
logger.push('price', 5500);
logger.logger();
logger.push('customer', 'lisi');
logger.push('product', 'ipod');
logger.push('price', 3000);
logger.logger();
```

执行以上命令后，可以在日志服务看到如下两条日志：

```
customer:zhangsan
product:iphone 6s
price:5500


customer:lisi
product:ipod
price:3000
```

后续步骤

数据上传到日志服务之后，可以使用日志服务将 数据导入OSS，也可以使用日志服务提供的 Loghub client library 消费数据。

Loghub Log4j Appender（源码）

Log4j 是 Apache 的一个开放源代码项目，通过使用 Log4j，您可以控制日志信息输送的目的地是控制台、文件、GUI 组件、甚至是套接口服务器、NT 的事件记录器、UNIX Syslog 守护进程等；您也可以控制每一条日志的输出格式；通过定义每一条日志信息的级别，您能够更加细致地控制日志的生成过程。最令人感兴趣的就是，这些可以通过一个配置文件来灵活地进行配置，而不需要修改应用的代码。

Log4j 由三个重要的组件构成：日志信息的优先级，日志信息的输出目的地，日志信息的输出格式。日志信息的优先级从高到低分别为 ERROR、WARN、INFO和DEBUG，分别用来指定这条日志信息的重要程度；日志信息的输出目的地指定了日志将打印到控制台还是文件中；而输出格式则控制了日志信息的显示内容。

通过Loghub Log4j Appender，您可以控制日志的输出目的地为阿里云日志服务。需要注意的是，Loghub Log4j Appender不支持设置日志的输出格式，写到日志服务中的日志的样式如下：

```
level:ERROR
location:test.TestLog4jAppender.main(TestLog4jAppender.java:18)
message:test log4j appender
thread:main
time:2016-05-27T03:15+0000
```

其中：

- level 是日志级别。
- location 是日志打印语句的代码位置。
- message 是日志内容。
- thread 是线程名称。
- time 是日志打印时间。

注意：目前只支持log4j1.x。

功能优势

- 客户端日志不落盘，即数据生产后直接通过网络发往服务端。
- 对于已经使用Log4j记录日志的应用，只需要简单修改配置文件就可以将日志传输到日志服务。
- 异步高吞吐，Loghub Log4j Appender 会将用户的日志merge之后异步发送，提高网络I/O效率。

配置步骤

maven 工程中引入依赖。

```
<dependency>
<groupId>com.google.protobuf</groupId>
<artifactId>protobuf-java</artifactId>
<version>2.5.0</version>
</dependency>
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>log-loghub-log4j-appender</artifactId>
```

```
<version>0.1.3</version>
</dependency>
```

修改 log4j.properties 文件

修改 log4j.properties 文件，不存在则在项目根目录创建。

配置根Logger，其语法为：

```
log4j.rootLogger = [level] , appenderName1, appenderName2, ...
```

其中：

- level 是日志记录的优先级，优先级从高到低分别是 ERROR、WARN、INFO、DEBUG。通过在这里定义的级别，您可以控制应用程序中相应级别的日志信息的开关。比如在这里定义了 INFO 级别，则应用程序中所有 DEBUG 级别的日志信息将不被打印出来。
- appenderName 指定日志信息输出到哪个地方。您可以同时指定多个输出目的地，这里的每个 appender 会对应到具体某一种Appender类型，每种Appender都会提供一些配置参数。

Loghub Log4j Appender的配置如下：

```
log4j.rootLogger=WARN,loghub
log4j.appender.loghub = com.aliyun.openservices.log.log4j.LoghubAppender
log4j.appender.loghub.projectName = [you project]
log4j.appender.loghub.logstore = [you logstore]
log4j.appender.loghub.endpoint = [your project endpoint]
log4j.appender.loghub.accessKeyId = [your accesskey id]
log4j.appender.loghub.accessKey = [your accesskey]
```

配置中中括号内的部分是需要填写的，具体含义见下面的说明。

配置格式

```
log4j.appender.loghub.projectName = [you project]
log4j.appender.loghub.logstore = [you logstore]
log4j.appender.loghub.endpoint = [your project endpoint]
log4j.appender.loghub.accessKeyId = [your accesskey id]
log4j.appender.loghub.accessKey = [your accesskey]
log4j.appender.loghub.stsToken=[your ststoken]
log4j.appender.loghub.packageTimeoutInMS=3000
log4j.appender.loghub.logsCountPerPackage=4096
log4j.appender.loghub.logsBytesPerPackage = 5242880
log4j.appender.loghub.memPoolSizeInByte=1048576000
log4j.appender.loghub.ioThreadsCount=1
log4j.appender.loghub.timeFormat=yyyy-MM-dd'T'HH:mmZ
log4j.appender.loghub.timeZone=UTC
```

参数说明

Loghub Log4j Appender 可供配置的参数如下。必选参数需要您自定义配置，可选参数如未配置，则使用默认值。

参数	含义	取值
projectName	日志服务的Project名。必选参数 。	只能包含小写字母，数字，连字符（-）；必须以小写字母和数字开头和结尾；长度为3~63字节。必须是已经存在的项目名称。
logstore	日志服务的Logstore名。必选参数。	只能包含小写字母，数字，连字符（-）；必须以小写字母和数字开头和结尾；长度为3~63字节。必须是已经存在的Logstore。
endpoint	日志服务的HTTP地址。必选参数。	URL格式，示例：myproject.cn-shanghai.log.aliyuncs.com。
accessKeyId	用户身份标识accesskey ID。必选参数。	-
accessKey	用户身份标识accesskey secret。必选参数。	-
stsToken	您的STS Token。当使用临时身份时必须填写，非临时身份无需配置此行	-
packageTimeoutInMS	指定被缓存的日志的发送超时时间，如果缓存超时，则会被立即发送。可选参数。	整数形式，单位是毫秒。
logsCountPerPackage	指定每个缓存的日志包中包含日志数量的最大值。可选参数。	整数形式，取值范围是1~4096。
logsBytesPerPackage	指定每个缓存的日志包的大小的上限，可选参数。	整数形式，取值范围是1~5242880，单位是字节。
memPoolSizeInByte	指定Appender实例可以使用的内存的上限。可选参数。	整数形式，单位是字节。默认为104857600。
ioThreadsCount	指定后台用于发送日志包的I/O线程的数量。可选参数。	整数形式，默认值为1。
timeFormat	指定输出到日志服务的时间格式。可选参数。	使用Java中SimpleDateFormat格式化时间，默认为ISO8601。
timeZone	指定时区	-

LogHub Producer Library 是针对 Java 应用程序高并发写LogHub类库，Producer Library 和 Consumer

Library 是对LogHub的读写包装，降低数据收集与消费的门槛。

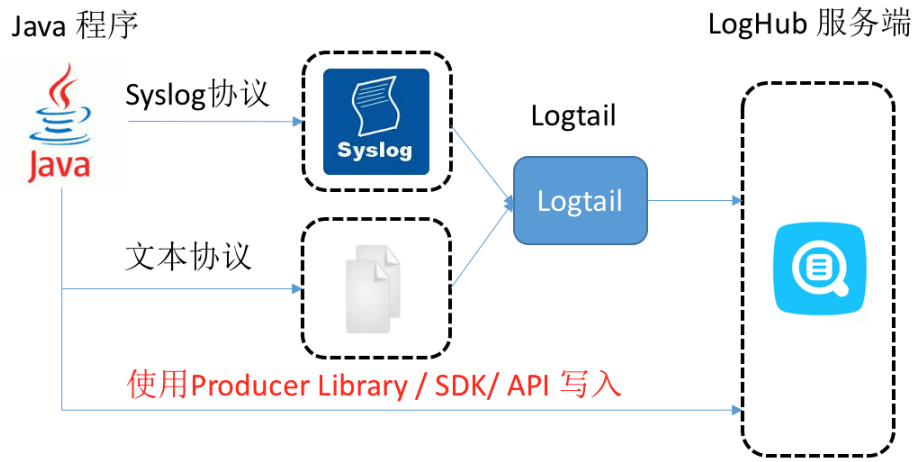
功能特点

- 提供异步的发送接口，线程安全。
- 可以添加多个Project的配置。
- 用于发送的网络 I/O 线程数量可以配置。
- merge成的包的日志数量以及大小都可以配置。
- 内存使用可控，当内存使用达到用户配置的阈值时，Producer 的 send 接口会阻塞，直到有空闲的内存可用。

功能优势

- 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。
- 客户端高并发写入：例如一秒钟会有百次以上写操作。
- 客户端计算与 I/O 逻辑分离：打印日志不影响计算耗时。

在以上场景中，Producer Library 会简化您程序开发的步骤，帮助您批量聚合写请求，通过异步的方式发往LogHub服务端。在整个过程中，您可以配置批量聚合的参数、服务端异常处理的逻辑等。



以上各种接入方式的对比：

接入方式	优点/缺点	针对场景
日志落盘 + Logtail	日志收集与打日志解耦，无需修改代码	常用场景
syslog + Logtail	性能较好（80MB/S），日志不落盘，需支持 syslog 协议	syslog 场景
SDK 直发	不落盘，直接发往服务端，需要处理好网络 IO 与程序 IO 之间的切换	日志不落盘
Producer Library	不落盘，异步合并发送服务端，吞吐量较好	日志不落盘，客户端 QPS 高

注意：目前 Producer Library 只支持 Java 版本。

配置步骤

Producer Library配置分为以下几个步骤：

maven 工程中添加依赖

```
<dependency>
<groupId>com.google.protobuf</groupId>
<artifactId>protobuf-java</artifactId>
<version>2.5.0</version>
</dependency>
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>aliyun-log</artifactId>
<version>0.6.8</version>
</dependency>
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>log-loghub-producer</artifactId>
<version>0.1.4</version>
</dependency>
```

程序中配置 ProducerConfig

配置格式如下，参数取值见本文档中**参数取值**部分。

```
public class ProducerConfig
{
public int packageTimeoutInMS = 3000;
public int logsCountPerPackage = 4096;
public int logsBytesPerPackage = 5 * 1024 * 1024;
public int memPoolSizeInByte = 1000 * 1024 * 1024;
public int maxIOThreadSizeInPool = 50;
public int shardHashUpdateIntervalInMS = 10 * 60 * 1000;
public int retryTimes = 3;
}
```

继承 ILogCallback

callback 主要用于日志发送结果的处理，结果包括发送成功和发生异常。您也可以选择不处理，这样就不需要继承 ILogCallback。

创建 producer 实例，调用 send 接口发数据

参数取值

参数	参数说明	取值
packageTimeoutInMS	指定被缓存日志的发送超时时间，如果缓存超时，则会被立即发送。	整数形式，单位为毫秒。
logsCountPerPackage	指定每个缓存的日志包中包含日志数量的最大值。	整数形式，取值为1~4096。
logsBytesPerPackage	指定每个缓存的日志包的大小上限。	整数形式，取值为1~5242880，单位为字节。
memPoolSizeInByte	指定单个Producer实例可以使用的内存的上限。	整数形式，单位为字节。
maxIOThreadSizeInPool	指定I/O线程池最大线程数量，主要用于发送数据到日志服务。	整数形式。
shardHashUpdateIntervalInMS	指定更新Shard的Hash区间的时间间隔，当指定shardhash的方式发送日志时，需要设置此参数。 后端merge线程会将映射到同一个Shard的数据merge在一起，而Shard关联的是一个Hash区间，Producer在处理时会将用户传入的Hash映射成Shard关联Hash区间的最小值。每一个Shard关联的Hash区间，Producer会定时从LogHub拉取。	整数形式。
retryTimes	指定发送失败时重试的次数，如果超过该值，就会将异常作为callback的参数，交由用户处理。	整数形式。

使用实例

main：

```
public class ProducerSample {

public static String RandomString(int length) {
String str = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
Random random = new Random();
StringBuffer buf = new StringBuffer();
for (int i = 0; i < length; i++) {
int num = random.nextInt(62);
buf.append(str.charAt(num));
}
return buf.toString();
```

```
}

public static void main(String args[]) throws InterruptedException {
ProducerConfig producerConfig = new ProducerConfig();
//使用默认配置创建 producer 实例
final LogProducer producer = new LogProducer(producerConfig);
// 添加多个 project 配置
producer.setProjectConfig(new ProjectConfig("your project 1",
"endpoint", "your accesskey id", "your accesskey"));
producer.setProjectConfig(new ProjectConfig("your project 2",
"endpoint", "your accesskey id", "your accesskey",
"your sts token"));
// 更新 project 1 的配置
producer.setProjectConfig(new ProjectConfig("your project 1",
"endpoint", "your new accesskey id", "your new accesskey"));
// 删除 project 2 的配置
producer.removeProjectConfig("your project 2");
// 生成日志集合，用于测试
final Vector<Vector<LogItem>> logGroups = new Vector<Vector<LogItem>>();
for (int i = 0; i < 100000; ++i) {
Vector<LogItem> tmpLogGroup = new Vector<LogItem>();
LogItem logItem = new LogItem((int) (new Date().getTime() / 1000));
logItem.PushBack("level", "info" + System.currentTimeMillis());
logItem.PushBack("message", "test producer send perf "
+ RandomString(50));
logItem.PushBack("method", "SenderToServer " + RandomString(10));
tmpLogGroup.add(logItem);
logGroups.add(tmpLogGroup);
}
// 并发调用 send 发送日志
Random random = new Random();
for (int j = 0; j < 100000; ++j) {
int rand = random.nextInt(99999);
producer.send("project 1", "logstore 1", "topic", "source ip", logGroups.get(rand), new CallbackSample("project 1",
"logstore 1", "topic", "source ip", null, logGroups.get(rand), producer));
}
//主动刷新缓存起来的还没有被发送的日志
producer.flush();
// 等待数据发送完毕
Thread.sleep(2 * producerConfig.packageTimeoutInMS);
//关闭后台 io 线程，close 会将调用时刻内存中缓存的数据发送出去
producer.close();
}
}
```

callback：

```
public class CallbackSample extends ILogCallback {
//保存要发送的数据，当时发生异常时，进行重试
public String project;
public String logstore;
public String topic;
public String shardHash;
public String source;
public Vector<LogItem> items;
```

```
public LogProducer producer;
public int retryTimes = 0;
public CallbackSample(String project, String logstore, String topic,
String shardHash, String source, Vector<LogItem> items, LogProducer producer) {
super();
this.project = project;
this.logstore = logstore;
this.topic = topic;
this.shardHash = shardHash;
this.source = source;
this.items = items;
this.producer = producer;
}

public void onCompletion(PutLogsResponse response, LogException e) {
if (e != null) {
// 打印异常
System.out.println(e.GetErrorCode() + ", " + e.GetErrorMessage() + ", " + e.GetRequestId());
//最多重试三次
if(retryTimes++ < 3)
{
producer.send(project, logstore, topic, source, shardHash, items, this);
}
}
else{
System.out.println("send success, request id: " + response.GetRequestId());
}
}

}
```

常见日志格式

Apache日志格式和目录通常在配置文件 /etc/apache2/httpd.conf 中。

Apache日志格式

日志格式

Apache日志配置文件中定义了两种打印格式，分别为combined格式和common格式。

- combined格式：

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

common格式：

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
```

声明使用了combined日志格式和写入的文件名。

```
CustomLog "/var/log/apache2/access_log" combined
```

字段说明

字段格式	含义
%a	remote_ip
%A	local_ip
%B	size
%b	size
%D	time_taken_ms
%h	remote_host
%H	protocol
%l	ident
%m	method
%p	port
%P	pid
“%q”	url_query
“%r”	request
%s	status
%>s	status
%t	time
%T	time_taken
%u	remote_user
%U	url_stem
%v	server_name
%V	canonical_name
%I	bytes_received
%O	bytes_sent
“%{User-Agent}i”	user_agent

“%{Referer}i”	referer

日志样例

```
192.168.1.2 - - [02/Feb/2016:17:44:13 +0800] "GET /favicon.ico HTTP/1.1" 404 209 "http://localhost/x1.html" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.97 Safari/537.36"
```

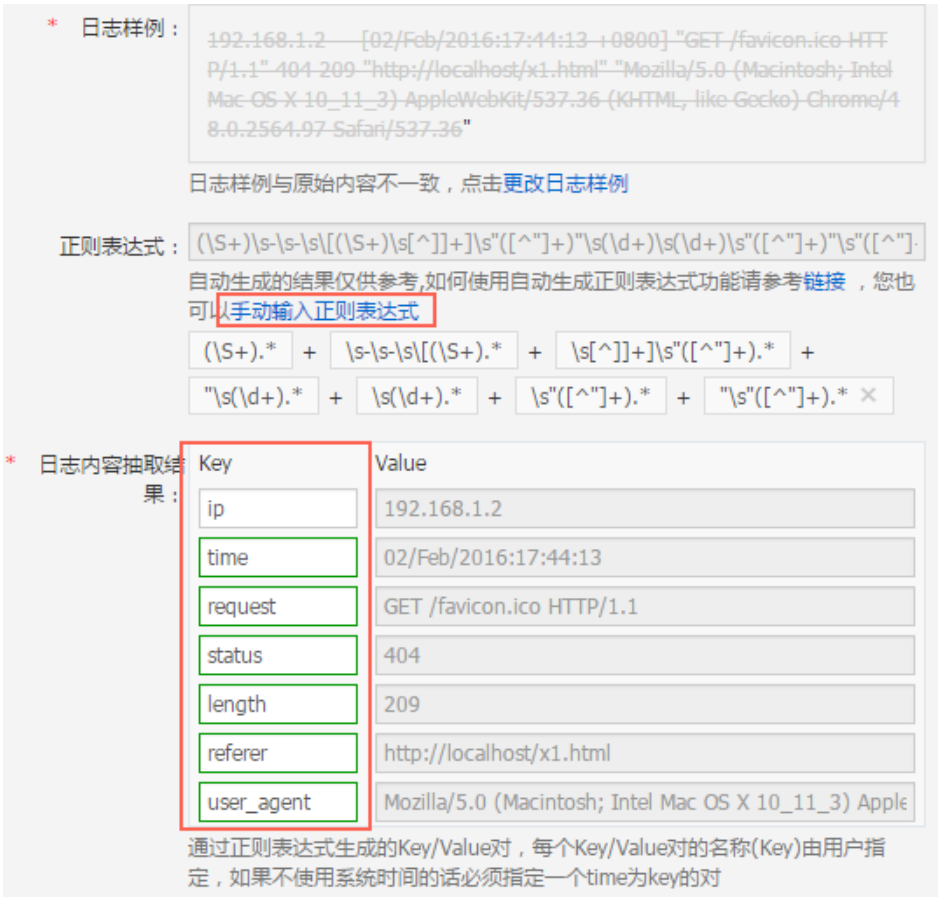
配置Logtail收集Apache日志

通过Logtail收集Apache日志完整流程请参考**快速入门**，根据您的网络部署和实际情况选择对应配置。本文档仅展示配置Logtail的第二步**指定收集模式**步骤中的详细配置。

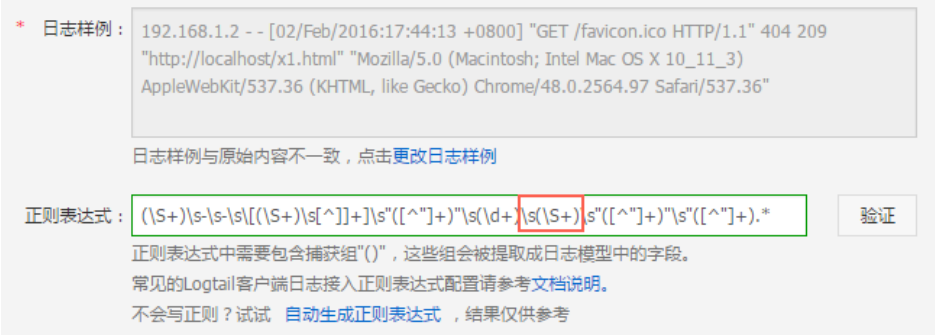
填写配置名称、日志路径，并选择日志收集模式为**完整正则模式**。

输入日志样例并开启**自动提取字段**。

单击 **手动输入正则表达式**，并调整正则表达式。



日志服务支持对日志样例划词自动解析，即对您在划词时选取的字段自动生成正则表达式。但鉴于实际的日志数据格式可能会有细微变动，您需要在根据实际情况对自动生成的正则表达式做出调整，使其符合收集过程中所有可能出现的日志格式。



> 由于 length 这个字段在这里是数字类型，但有些情况下这里不是数字而是“-” ，所以匹配结果(\d+)需要替换成(\S+)。如果您还有其它字段存在这种情况，请按照同样的规则完成替换。

正则表达式修改完成后，单击 **验证**。如果正则式没有错误，会出现提取的结果，如果有错误请再次调整正则式。

为日志内容抽取结果填写对应的Key。

分别为提取结果取一个有意义的字段名称，比如时间字段的命名为 time。开启 **使用系统时间**，然后单击 **下一步**。

Logtail配置完成后，将此配置应用到机器组即可开始规范收集Apache日志。

Nginx日志格式和目录通常在配置文件 /etc/nginx/nginx.conf 中。

Nginx日志格式

日志格式

配置文件中定义了Nginx日志的打印格式，即main格式：

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

声明使用了main日志格式和写入的文件名。

```
access_log /var/logs/nginx/access.log main
```

- **字段说明**

字段名称	含义
remoteaddr	表示客户端IP地址。
remote_user	表示客户端用户名称。
request	表示请求的URL和HTTP协议。
status	表示请求状态。
bodybytessent	表示发送给客户端的字节数，不包括响应头的大小；该变量与Apache模块modlogconfig里的bytes_sent发送给客户端的总字节数相同。
connection	表示连接的序列号。
connection_requests	表示当前通过一个连接获得的请求数量。
msec	表示日志写入的时间。单位为秒，精度是毫秒。
pipe	表示请求是否通过HTTP流水线（pipelined）发送。通过HTTP流水线发送则pipe值为“p”，否则为“.”。
httpreferer	表示从哪个页面链接访问过来的。
“http_user_agent”	表示客户端浏览器相关信息，前后必须加上双引号。
requestlength	表示请求的长度。包括请求行，请求头和请求正文。
request_time	表示请求处理时间，单位为秒，精度为毫秒。从读入客户端的第一个字节开始，直到把最后一个字符发送给客户端后进行日志写入为止。
[$time_local]	表示通用日志格式下的本地时间，前后必须加上中括号。

- **日志样例**

```
192.168.1.2 - - [10/Jul/2015:15:51:09 +0800] "GET /ubuntu.iso HTTP/1.0" 0.000 129 404 168 "-" "Wget/1.11.4 Red Hat modified"
```

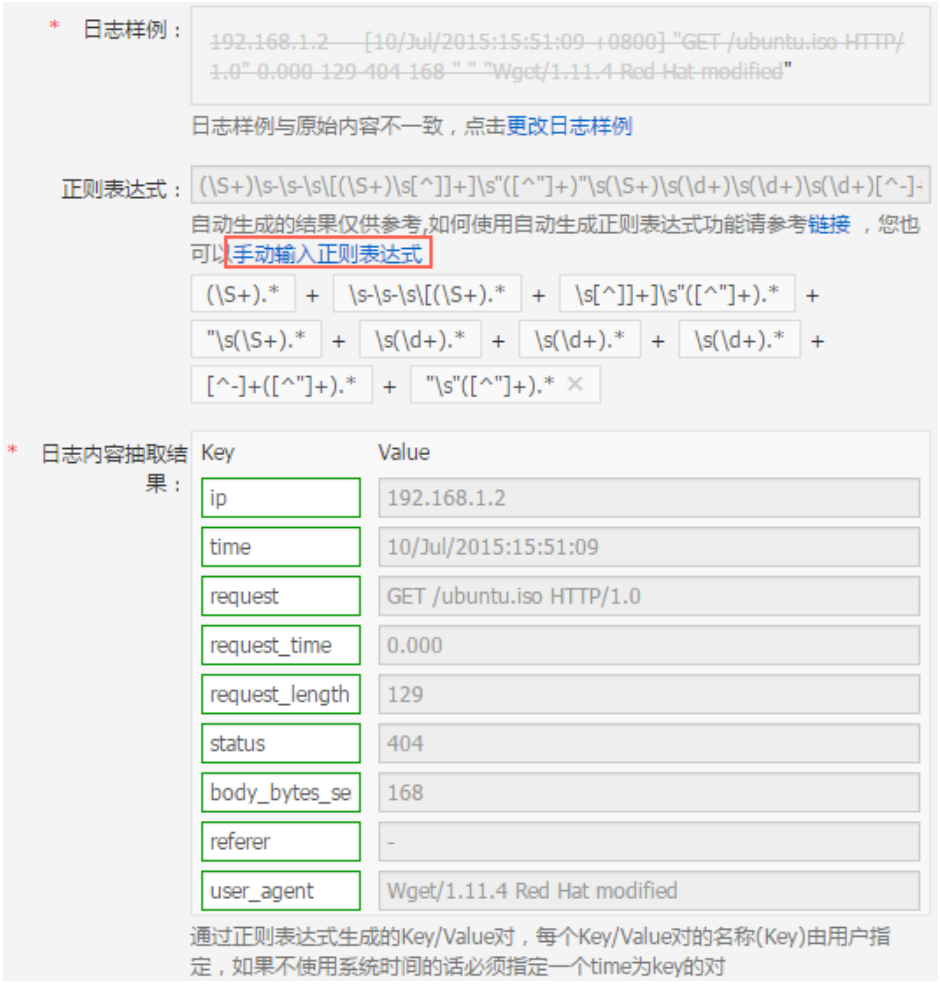
配置Logtail收集Nginx日志

通过Logtail收集Nginx日志完整流程请参考**快速入门**，根据您的网络部署和实际情况选择对应配置。本文档仅展示配置Logtail的第二步**指定收集模式**步骤中的详细配置。

填写配置名称、日志路径，并选择日志收集模式为**完整正则模式**。

输入日志样例并开启**自动提取字段**。

单击 手动输入正则表达式，并调整正则表达式。



日志服务支持对日志样例划词自动解析，即对您在划词时选取的字段自动生成正则表达式。但鉴于实际的日志数据格式可能会有细微变动，您需要在根据实际情况对自动生成的正则表达式做出调整，使其符合收集过程中所有可能出现的日志格式。

由于 request_length 和 body_bytes_sent 这两个字段在这里是数字类型，但有些情况下这里不是数字而是 “-” ，所以匹配结果(\d+)需要替换成(\S+)。如果您还有其它字段存在这种情况，请按照同样的规则完成替换。

符合日志格式的正则表达式：

```
(\S+)\s-\s-\s\[(\S+)\s[^\]]+\]\s"(\S+)\s(\S+)\s(\S+)"\s(\S+)\s(\S+)\s(\S+)\s(\S+)\s"([^"]+)"\s"([^"]+)"
```

正则表达式修改完成后，单击 **验证**。如果正则式没有错误，会出现提取的结果，如果有错误请再次调整正则式。

为日志内容抽取结果填写对应的Key。

分别为提取结果取一个有意义的字段名称，比如时间字段的命名为time。开启 **使用系统时间**，然后单击 **下一步**。

Logtail配置完成后，将此配置应用到机器组即可开始规范收集Nginx日志。

Python 的 logging 模块提供了通用的日志系统，可以方便第三方模块或者是应用使用。这个模块提供不同的日志级别，并可以采用不同的方式记录日志，比如：文件、HTTP GET/POST、SMTP、Socket等，甚至可以自己实现具体的日志记录方式。logging 模块与 log4j 的机制是一样的，只是具体的实现细节不同。模块提供 logger、handler、filter、formatter。

Python日志格式

日志格式

日志的格式在formatter中指定日志记录输出的具体格式。formatter的构造方法需要两个参数：消息的格式字符串和日期字符串，这两个参数都是可选的。

Python日志格式：

```
import logging
```

```
import logging.handlers

LOG_FILE = 'tst.log'

handler = logging.handlers.RotatingFileHandler(LOG_FILE, maxBytes = 1024*1024, backupCount = 5) # 实例化 handler
fmt = '%(asctime)s - %(filename)s:%(lineno)s - %(name)s - %(message)s'

formatter = logging.Formatter(fmt) # 实例化 formatter
handler.setFormatter(formatter) # 为 handler 添加 formatter

logger = logging.getLogger('tst') # 获取名为 tst 的 logger
logger.addHandler(handler) # 为 logger 添加 handler
logger.setLevel(logging.DEBUG)

logger.info('first info message')
logger.debug('first debug message')
```

字段含义

关于 formatter 的配置，采用的是 %(key)s 的形式，就是字典的关键字替换。提供的关键字包括：

格式	含义
%(name)s	生成日志的Logger名称。
%(levelno)s	数字形式的日志级别，包括DEBUG, INFO, WARNING, ERROR和CRITICAL。
%(levelname)s	文本形式的日志级别，包括’ DEBUG’ 、 ‘INFO’ 、 ‘WARNING’ 、 ‘ERROR’ 和 ’ CRITICAL’ 。
%(pathname)s	输出该日志的语句所在源文件的完整路径（如果可用）。
%(filename)s	文件名。
%(module)s	输出该日志的语句所在的模块名。
%(funcName)s	调用日志输出函数的函数名。
%(lineno)d	调用日志输出函数的语句所在的代码行（如果可用）。
%(created)f	日志被创建的时间，UNIX标准时间格式，表示从1970-1-1 00:00:00 UTC计算起的秒数。
%(relativeCreated)d	日志被创建时间与日志模块被加载时间的时间差，单位为毫秒。
%(asctime)s	日志创建时间。默认格式是 “2003-07-08 16:49:45,896” ，逗号后为毫秒数。
%(msecs)d	毫秒级别的日志创建时间。
%(thread)d	线程ID（如果可用）。
%(threadName)s	线程名称（如果可用）。
%(process)d	进程ID（如果可用）。

%(message)s	日志信息。

日志样例

```
2015-03-04 23:21:59,682 - log_test.py:16 - tst - first info message
2015-03-04 23:21:59,682 - log_test.py:17 - tst - first debug message
```

配置Logtail收集Python日志

配置Logtail收集Python日志的详细操作步骤请参考快速入门和apache日志，根据您的网络部署和实际情况选择对应配置。

在生成正则式的部分，由于自动生成的正则式只参考了日志样例，无法覆盖所有的日志情况，所以需要用户在自动生成之后做一些微调。

常见的Python日志及其正则表达式：

- 日志样例：

```
2016-02-19 11:03:13,410 - test.py:19 - tst - first debug message
```

正则表达式：

```
(\d+-\d+-\d+\s\S+)\s+-\s+([^:]+):(\d+)\s+-\s+(\w+)\s+-\s+(.*)
```

日志格式：

```
%(asctime)s - %(filename)s:%(lineno)s - %(levelno)s %(levelname)s %(pathname)s %(module)s
%(funcName)s %(created)f %(thread)d %(threadName)s %(process)d %(name)s - %(message)s
```

日志样例：

```
2016-02-19 11:06:52,514 - test.py:19 - 10 DEBUG test.py test <module> 1455851212.514271
139865996687072 MainThread 20193 tst - first debug message
```

正则表达式：

```
(\d+-\d+-\d+\s\S+)\s-\s([^:]+):(\d+)\s+-
\s+(\d+)\s+(\w+)\s+(\S+)\s+(\w+)\s+(\S+)\s+(\S+)\s+(\d+)\s+(\w+)\s+(\d+)\s+(\w+)\s+-\s+(.*)
```

日志服务支持通过以下方式采集Log4j日志：

- Loghub Log4j Appender
- Logtail

通过Loghub Log4j Appender采集Log4j日志

详细内容及采集步骤请参考Log4j Appender。

通过Logtail采集Log4j日志

配置Logtail收集Python日志的详细操作步骤请参考快速入门和Apache日志，根据您的网络部署和实际情况选择对应配置。

在生成正则式的部分，由于自动生成的正则式只参考了日志样例，无法覆盖所有的日志情况，所以需要用户在自动生成之后做一些微调。

Log4j 默认日志格式打印到文件中的日志样例如下：

```
2013-12-25 19:57:06,954 [10.207.37.161] WARN impl.PermanentTairDaoImpl - Fail to Read Permanent Tair,key:e:470217319319741_1,result:com.example.tair.Result@172e3ebc[rc=code=-1, msg=connection error or timeout,value=,flag=0]
```

多行日志起始匹配（使用IP信息表示一行开头）：

```
\d+-\d+-\d+\s.*
```

提取日志信息的正则表达式：

```
(\d+-\d+-\d+\s\d+:\d+:\d+,\d+)\s\[([^\]]*)\]\s(\S+)\s+(\S+)\s-\s(.*)
```

时间转换格式：

```
%Y-%m-%d %H:%M:%S
```

样例日志提取结果：

Key	Value
time	2013-12-25 19:57:06,954
ip	10.207.37.161
level	WARN
class	impl.PermanentTairDaoImpl
message	Fail to Read Permanent Tair,key:e:470217319319741_1,result:com.exa

	mple.tair.Result@172e3ebc[rc=code=-1, msg=connection error or timeout,value=,flag=0]

Node.js的日志默认打印到控制台，为数据收集和问题调查带来不便。通过log4js可以实现把日志打印到文件、自定义日志格式等功能，便于数据收集和整理。

```
var log4js = require('log4js');
log4js.configure({
appenders: [
{
type: 'file', //文件输出
filename: 'logs/access.log',
maxLogSize: 1024,
backups:3,
category: 'normal'
}
]
});
var logger = log4js.getLogger('normal');
logger.setLevel('INFO');
logger.info("this is a info msg");
logger.error("this is a err msg");
```

日志格式

通过log4js实现日志数据存储为文本文件格式后，日志在文件中显示为以下格式：

```
[2016-02-24 17:42:38.946] [INFO] normal - this is a info msg
[2016-02-24 17:42:38.951] [ERROR] normal - this is a err msg
```

log4js分为6个输出级别，从低到高分别为trace、debug、info、warn、error、fatal。

通过Logtail收集Node.js日志

配置Logtail收集Python日志的详细操作步骤请参考快速入门和apache日志，根据您的网络部署和实际情况选择对应配置。

在生成正则式的部分，由于自动生成的正则式只参考了日志样例，无法覆盖所有的日志情况，所以需要用户在自动生成之后做一些微调。您可以参考以下Node.js日志示例，为您的日志撰写正确、全面的正则表达式。

常见的Node.js日志及其正则表达式：

Node.js日志示例1

日志示例：

```
[2016-02-24 17:42:38.946] [INFO] normal - this is a info msg
```

正则表达式：

```
\[([^]]+)]\s\[([^\]]+)]\s(\w+)\s-(.*)
```

提取字段：

time、level、loggerName和message。

Node.js日志示例2：

日志示例：

```
[2016-01-31 12:02:25.844] [INFO] access - 42.120.73.203 - - "GET
/user/projects/ali_sls_log?ignoreError=true HTTP/1.1" 304 - "http://
aliyun.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_3) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/48.0.2564.97 Safari/537.36"
```

正则表达式：

```
\[([^]]+)]\s\[(\w+)]\s(\w+)\s-\s(\S+)\s-\s-\s"([^"]+)"\s(\d+)[^"]+("[^"]+)"\s"([^"]+).*
```

提取字段：

time、level、loggerName、ip、 request、status、referer和 user_agent。

wordpress 日志

WordPress 默认日志格式

原始日志样例：

```
172.64.0.2 - - [07/Jan/2016:21:06:39 +0800] "GET /wp-admin/js/password-strength-meter.min.js?ver=4.4 HTTP/1.0"
200 776 "http://wordpress.c4a1a0aecdb1943169555231dcc4adfb7.cn-hangzhou.alicontainer.com/wp-
```

```
admin/install.php" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36"
```

多行日志起始匹配（使用 IP 信息表示一行开头）：

```
\d+\.\d+\.\d+\.\d+\s-\s.*
```

提取日志信息的正则表达式：

```
(\S+) - - \[([^\]]*)] "(\S+) ([^"]+)" (\S+) (\S+) "([^"]+)" "([^"]+)"
```

时间转换格式：

```
%d/%b/%Y:%H:%M:%S
```

样例日志提取结果：

Key	Value
ip	127.64.0.2
time	07/Jan/2016:21:06:39 +0800
method	GET
url	/wp-admin/js/password-strength-meter.min.js?ver=4.4 HTTP/1.0
status	200
length	776
ref	http://wordpress.c4a1a0aecdb1943169555231dcc4adfb7.cn-hangzhou.alicontainer.com/wp-admin/install.php
user-agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36

分隔符日志以换行符作为边界，每一个自然行都是一条日志。每一条日志以固定分隔符连接日志的多个字段，分隔符（Separator）包括制表符、空格、竖线、逗号、分号等单字符。如果字段内部包含分隔符，使用双引号（Quote）对字段进行包裹。

常见的分隔符日志有CSV日志和TSV日志 等。

日志格式

分隔符日志使用分隔符（Separator）将一条日志切分成多个字段，支持**单字符**和**多字符**两种模式。

单字符模式

单字符模式通过匹配单字符进行日志切分，例如制表符（\t）、空格、竖线（|）、逗号（,）、分号（;）等单字符。

注意： 分隔符不允许设置为双引号（"），双引号被作为默认的单字符分隔符的Quote。

单字符分隔符容易出现日志字段中包含分隔符的场景，为避免日志字段被误分割，需要使用双引号（"）作为Quote，对日志字段进行包裹隔离。如果内容中在非Quote情况下出现双引号，则需要进行转义，处理成""。双引号（"）要么作为Quote使用，在字段的边界单次出现，要么作为字段内数据成对出现（""），其它情况不符合分隔符日志的格式定义，请考虑极简模式、正则模式等其它方式进行字段解析。

双引号作为Quote

双引号（"）作为Quote时，内部包含分隔符的字段需要被一对Quote包裹。Quote必须紧邻分隔符，如有两者之间包含空格、制表符等字符，请修改格式。

例如，逗号（,）作为分隔符，双引号作为Quote时，日志格式为：1997,Ford,E350,"ac, abs, moon",3000.00。该日志可以被解析为5个字段： 1997、Ford、E350、ac, abs, moon和3000.00。其中被Quote包裹的ac, abs, moon被看做是一个完整字段。

双引号作为日志字段中的一部分

双引号作为日志字段中的一部分时，双引号不作为Quote出现，需要进行转义，处理成""。解析字段时进行还原，即将""还原为"。

例如，逗号作为分隔符，双引号和逗号作为日志字段中的一部分，需要将包含分逗号的日志字段用Quote包裹，同时将日志字段中的双引号转义为成对的双引号""。处理后的日志格式为：1999,Chevy,"Venture ""Extended Edition, Very Large""","",5000.00。该日志可以被解析为五个字段： 1999、Chevy、Venture "Extended Edition, Very Large"、空字段和5000.00。

多字符模式

多字符模式中，分隔符可以包括2~3个字符，如（||）、(&&&)、（^_^）等多字符。多字符分隔符模式下，日志解析完全根据分隔符进行匹配，您无需使用Quote对日志进行包裹。

注意： 需确保日志字段内容中不会出现分隔符的完整匹配，否则会产生字段误分割。

例如，分隔符设置为&&的情况下，日志：1997&&Ford&&E350&&ac&abs&moon&&3000.00会被解析为5个字段：1997、Ford、E350、ac&abs&moon和3000.00。

日志示例

单字符分隔符日志

```
05/May/2016:13:30:28,10.200.98.220,"POST
/PutData?Category=YunOsAccountOpLog&AccessKeyId=U0UjpekFQOVJW45A&Date=Fri%2C%2028%20Jun%2020
13%2006%3A53%3A30%20GMT&Topic=raw&Signature=pD12XYLmGxKQ%2Bmkd6x7hAgQ7b1c%3D
HTTP/1.1",200,18204,aliyun-sdk-java
05/May/2016:13:31:23,10.200.98.221,"POST
/PutData?Category=YunOsAccountOpLog&AccessKeyId=U0UjpekFQOVJW45A&Date=Fri%2C%2028%20Jun%2020
13%2006%3A53%3A30%20GMT&Topic=raw&Signature=pD12XYLmGxKQ%2Bmkd6x7hAgQ7b1c%3D
HTTP/1.1",401,23472,aliyun-sdk-java
```

多字符分隔符日志

```
05/May/2016:13:30:28&&10.200.98.220&&POST
/PutData?Category=YunOsAccountOpLog&AccessKeyId=U0UjpekFQOVJW45A&Date=Fri%2C%2028%20Jun%2020
13%2006%3A53%3A30%20GMT&Topic=raw&Signature=pD12XYLmGxKQ%2Bmkd6x7hAgQ7b1c%3D
HTTP/1.1&&200&&18204&&aliyun-sdk-java
05/May/2016:13:31:23&&10.200.98.221&&POST
/PutData?Category=YunOsAccountOpLog&AccessKeyId=U0UjpekFQOVJW45A&Date=Fri%2C%2028%20Jun%2020
13%2006%3A53%3A30%20GMT&Topic=raw&Signature=pD12XYLmGxKQ%2Bmkd6x7hAgQ7b1c%3D
HTTP/1.1&&401&&23472&&aliyun-sdk-java
```

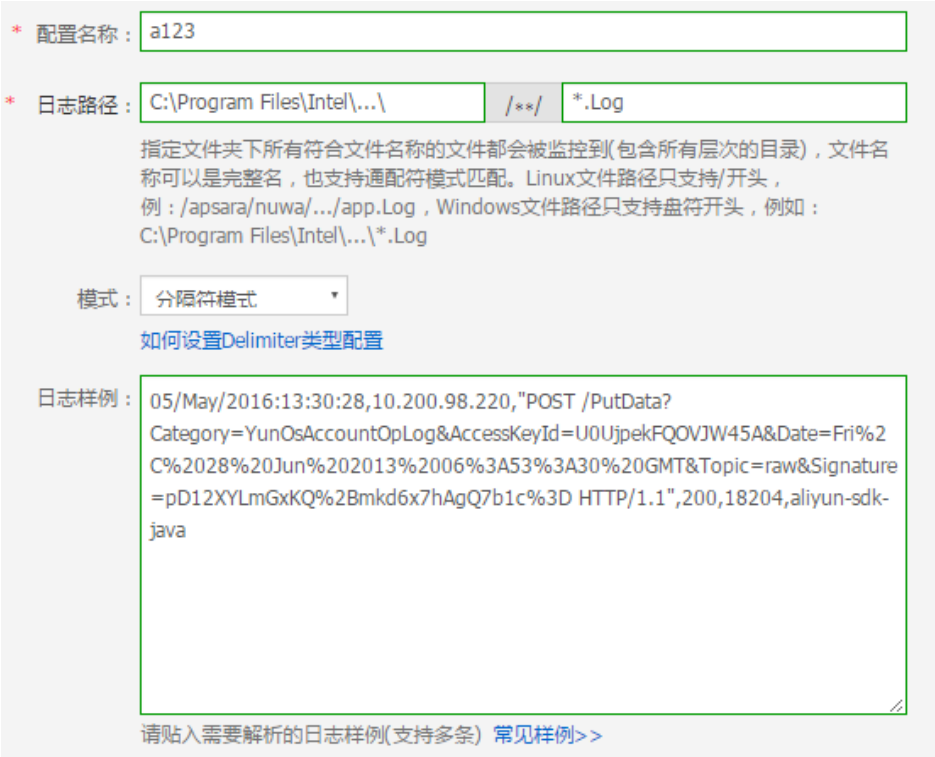
配置Logtail收集分隔符日志

通过Logtail收集分隔符日志完整流程请参考快速入门，根据您的网络部署和实际情况选择对应配置。本文档仅展示配置Logtail的第二步**指定收集模式**步骤中的详细配置。

填写配置名称、日志路径，并选择日志收集模式为**分隔符模式**。

填写日志样例并选择分隔符。

请根据您的日志格式选择正确的分隔符，否则日志数据会解析失败。



指定日志抽取结果中的Key。

填写日志样例并选择分隔符后，日志服务会按照您选择的分隔符提取日志字段，并将其定义为Value，您需要分别为Value指定对应的Key。

如上日志样例，使用逗号（,）进行分割，一共包含6个字段，设置Key值分别为：time，ip，url，status，latency，user-agent。

指定日志时间。

可以使用选择系统时间作为一条日志的时间，也可以使用日志的一列作为时间，比如选择 time 字段（05/May/2016:13:30:29）作为时间，配置日期格式请参考Logtail 日期格式。

在控制台上预览日志，确认是否成功收集。

JSON日志建构于两种结构：

- Object： “键/值”对的集合（A collection of name/value pairs）。
- Array：值的有序列表（An ordered list of values）。

Logtail支持Object类型的JSON日志，可以自动提取Object首层的键作为字段名称、Object首层的值作为字段值。字段值可以是Object、Array或基本类型，如String、Number等。JSON行与行之间用\n进行分割，每一行作为一条单独日志进行提取。

如果是JSON Array等非Object类型数据，Logtail不支持自动解析，请使用正则表达式提取字段，或者使用极简模式整行采集日志。

日志样例

```
{"url": "POST
/PutData?Category=YunOsAccountOpLog&AccessKeyId=U0UjpekFQOVJW45A&Date=Fri%2C%2028%20Jun%2020
13%2006%3A53%3A30%20GMT&Topic=raw&Signature=pD12XYLmGxKQ%2Bmkd6x7hAgQ7b1c%3D HTTP/1.1",
"ip": "10.200.98.220", "user-agent": "aliyun-sdk-java", "request": {"status": "200", "latency": "18204"}, "time":
"05/May/2016:13:30:28"}
{"url": "POST
/PutData?Category=YunOsAccountOpLog&AccessKeyId=U0UjpekFQOVJW45A&Date=Fri%2C%2028%20Jun%2020
```

13%2006%3A53%3A30%20GMT&Topic=raw&Signature=pD12XYLmGxKQ%2Bmkd6x7hAgQ7b1c%3D HTTP/1.1", "ip": "10.200.98.210", "user-agent": "aliyun-sdk-java", "request": {"status": "200", "latency": "10204"}, "time": "05/May/2016:13:30:29"}

配置Logtail收集JSON日志

通过Logtail收集JSON日志完整流程请参考**快速入门**，根据您的网络部署和实际情况选择对应配置。本文档仅展示配置Logtail的第二步**指定收集模式**步骤中的详细配置。

填写配置名称、日志路径，并选择日志收集模式为**JSON模式**。

确认是否使用系统时间。

请根据您的需求确认是否使用系统时间作为日志时间。您可以选择开启或者关闭**使用系统时间**功能。

开启 **使用系统时间**

使用系统时间表示不提取日志中的时间字段，将日志服务采集该日志的时间作为日志时间。

关闭 **使用系统时间**

不使用系统时间表示从日志数据中提取时间字段，将其作为日志时间。

如果选择关闭 **使用系统时间**，您需要定义被提取作为时间字段的Key名称，同时定义时间转换格式。例如JSON Object中的 time 字段（05/May/2016:13:30:29）可以提取为日志时间。配置日期格式请参考 Logtail日期格式。

ThinkPHP 是一个PHP语言的Web应用开发框架。

日志格式

在ThinkPHP中打印日志按照以下格式：

```
<?php
Think\Log::record('D 方法实例化没找到模型类' );
?>
```

日志示例

```
[ 2016-05-11T21:03:05+08:00 ] 30.9.181.163 /index.php
INFO: [ app_init ] --START--
INFO: Run Behavior\BuildLiteBehavior [ RunTime:0.000014s ]
INFO: [ app_init ] --END-- [ RunTime:0.000091s ]
INFO: [ app_begin ] --START--
INFO: Run Behavior\ReadHtmlCacheBehavior [ RunTime:0.000038s ]
INFO: [ app_begin ] --END-- [ RunTime:0.000076s ]
INFO: [ view_parse ] --START--
INFO: Run Behavior\ParseTemplateBehavior [ RunTime:0.000068s ]
INFO: [ view_parse ] --END-- [ RunTime:0.000104s ]
INFO: [ view_filter ] --START--
INFO: Run Behavior\WriteHtmlCacheBehavior [ RunTime:0.000032s ]
INFO: [ view_filter ] --END-- [ RunTime:0.000062s ]
INFO: [ app_end ] --START--
INFO: Run Behavior\ShowPageTraceBehavior [ RunTime:0.000032s ]
INFO: [ app_end ] --END-- [ RunTime:0.000070s ]
ERR: D 方法实例化没找到模型类
```

配置Logtail收集ThinkPHP日志

通过Logtail收集ThinkPHP日志完整流程请参考快速入门和apache日志，根据您的网络部署和实际情况选择对应配置。

在生成正则式的部分，由于自动生成的正则式只参考了日志样例，无法覆盖所有的日志情况，所以需要用户在自动生成之后做一些微调。

由于ThinkPHP是多行日志，而且模式并非固定，可以从日志中提取的字段包括时间、访问IP、访问的URL、以及打印的 Message。其中Message字段包含了多行信息，由于其模式不固定，只能打包到一个字段之中。

ThinkPHP日志的Logtail收集配置参数：

行首正则式：

```
\[\s\d+-\d+-\w+:\d+:\d+\+\d+:\d+\s.*
```

正则表达式：

```
```

```
\[\s(\d+-\d+-\w+:\d+:\d+)[^:]+:\d+\s]\s+(\S+)\s(\S+)\s+(.*)
```

时间表达式：

```
%Y-%m-%dT%H:%M:%S
```

日志样例

查看 IIS 日志配置，选择格式为 W3C（默认字段设置）保存生效。

```
2016-02-25 01:27:04 112.74.74.124 GET /goods/list/0/1.html - 80 - 66.249.65.102
Mozilla/5.0+(compatible;+Googlebot/2.1;++http://www.google.com/bot.html) 404 0 2 703
```

采集配置

```
input {
file {
type => "iis_log_1"
path => ["C:/inetpub/logs/LogFiles/W3SVC1/*.log"]
start_position => "beginning"
}
}

filter {
if [type] == "iis_log_1" {
#ignore log comments
if [message] =~ "^#" {
drop {}
}

grok {
# check that fields match your IIS log settings
match => ["message", "%{TIMESTAMP_ISO8601:log_timestamp} %{IPORHOST:site} %{WORD:method}
%{URIPATH:page} %{NOTSPACE:querystring} %{NUMBER:port} %{NOTSPACE:username} %{IPORHOST:clienthost}
%{NOTSPACE:useragent} %{NUMBER:response} %{NUMBER:subresponse} %{NUMBER:scstatus}
%{NUMBER:time_taken}"]
}

date {
match => [ "log_timestamp", "YYYY-MM-dd HH:mm:ss" ]
timezone => "Etc/UTC"
}

useragent {
source=> "useragent"
prefix=> "browser"
}
```

```
mutate {
remove_field => [ "log_timestamp"]
}
}
}

output {
if [type] == "iis_log_1" {
logservice {
codec => "json"
endpoint => "***"
project => "***"
logstore => "***"
topic => ""
source => ""
access_key_id => "***"
access_key_secret => "***"
max_send_retry => 10
}
}
}
```

注意：

- 配置文件格式必须以 UTF-8 无 BOM 格式编码，可以通过notepad++修改文件编码格式。
- path 填写文件路径时请使用UNIX模式的分隔符，如：C:/test/multiline/*.log，否则无法支持模糊匹配。
- type 字段需要统一修改并在该文件内保持一致，如果单台机器存在多个 Logstash 配置文件，需要保证各配置 type 字段唯一，否则会导致数据处理的错乱。

相关插件：file、grok。

重启 Logstash 生效

创建配置文件到 conf 目录，参考 配置Logstash 重启 Logstash 生效。

使用系统时间作为日志时间上传

日志样例

```
10.116.14.201,-,2/25/2016,11:53:17,W3SVC7,2132,200,0,GET,project/shenzhen-
test/logstore/logstash/detail,C:\test\csv\test_csv.log
```

采集配置

```
input {
file {
type => "csv_log_1"
path => ["C:/test/csv/*.log"]
start_position => "beginning"
}
}

filter {
if [type] == "csv_log_1" {
csv {
separator => ","
columns => ["ip", "a", "date", "time", "b", "latency", "status", "size", "method", "url", "file"]
}
}
}

output {
if [type] == "csv_log_1" {
logservice {
codec => "json"
endpoint => "***"
project => "***"
logstore => "***"
topic => ""
source => ""
access_key_id => "***"
access_key_secret => "***"
max_send_retry => 10
}
}
}
```

注意：

- 配置文件格式必须以 UTF-8 无 BOM 格式编码，可以下载 notepad++ 修改文件编码格式。
- path 填写文件路径时请使用 UNIX 模式的分隔符，如：C:/test/multiline/*.log，否则无法支持模糊匹配。
- type 字段需要统一修改并在该文件内保持一致，如果单台机器存在多个Logstash配置文件，需要保证各配置 type 字段唯一，否则会导致数据处理的错乱。

相关插件：file、csv。

重启Logstash生效

创建配置文件到 conf 目录，参考配置Logstash重启Logstash生效。

使用日志字段内容作为日志时间上传

日志样例

```
10.116.14.201,-,Feb 25 2016 14:03:44,W3SVC7,1332,200,0,GET,project/shenzhen-test/logstore/logstash/detail,C:\test\csv\test_csv_withtime.log
```

采集配置

```
input {
file {
type => "csv_log_2"
path => ["C:/test/csv_withtime/*.log"]
start_position => "beginning"
}
}

filter {
if [type] == "csv_log_2" {
csv {
separator => ","
columns => ["ip", "a", "datetime", "b", "latency", "status", "size", "method", "url", "file"]
}
date {
match => [ "datetime" , "MMM dd YYYY HH:mm:ss" ]
}
}
}

output {
if [type] == "csv_log_2" {
logservice {
codec => "json"
endpoint => "***"
project => "***"
logstore => "***"
topic => ""
source => ""
access_key_id => "***"
access_key_secret => "***"
max_send_retry => 10
}
}
}
```

注意：

- 配置文件格式必须以 UTF-8 无 BOM 格式编码，可以下载 notepad++ 修改文件编码格式。
- path 填写文件路径时请使用 UNIX 模式的分隔符，如：C:/test/multiline/*.log，否则无法支持模糊匹配。
- type 字段需要统一修改并在该文件内保持一致，如果单台机器存在多个Logstash配置文件，需要保证各配置 type 字段唯一，否则会导致数据处理的错乱。

相关插件：file、csv、date。

重启Logstash生效

创建配置文件到 conf 目录，参考配置Logstash重启Logstash生效。

日志样例

```
2016-02-25 15:37:01 [main] INFO com.aliyun.sls.test_log4j - single line log
2016-02-25 15:37:11 [main] ERROR com.aliyun.sls.test_log4j - catch exception !
java.lang.ArithmeticException: / by zero
at com.aliyun.sls.test_log4j.divide(test_log4j.java:23) ~[bin/:?]
at com.aliyun.sls.test_log4j.main(test_log4j.java:13) [bin/:?]
2016-02-25 15:38:02 [main] INFO com.aliyun.sls.test_log4j - normal log
```

采集配置

```
input {
file {
type => "common_log_1"
path => ["C:/test/multiline/*.log"]
start_position => "beginning"
codec => multiline {
pattern => "^\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}"
negate => true
auto_flush_interval => 3
what => previous
}
}
}

output {
if [type] == "common_log_1" {
logservice {
codec => "json"
endpoint => "***"
project => "***"
logstore => "***"
topic => ""
source => ""
access_key_id => "***"
access_key_secret => "***"
max_send_retry => 10
}
}
}
```

注意：

- 配置文件格式必须以 UTF-8 无 BOM 格式编码，可以下载 notepad++ 修改文件编码格式。
- path 填写文件路径时请使用 UNIX 模式的分隔符，如：C:/test/multiline/*.log，否则无法支持模糊匹配。
- type 字段需要统一修改并在该文件内保持一致，如果单台机器存在多个Logstash配置文件，需要保证各配置 type 字段唯一，否则会导致数据处理的错乱。

相关插件：file、multiline（若日志文件是单行日志，可以去掉 codec => multiline 配置）。

重启Logstash生效

创建配置文件到 conf 目录，参考配置Logstash重启Logstash生效。

Unity3D是由Unity Technologies开发的一个让玩家轻松创建诸如三维视频游戏、建筑可视化、实时三维动画等类型互动内容的多平台的综合型游戏开发工具，是一个全面整合的专业游戏引擎。

日志服务支持Web Tracking功能，您可以通过Web Tracking功能非常方便的收集Unity 3D的日志，本文档以收集Unity Debug.Log 为例，说明如何通过Web Tracking功能将Unity日志收集到日志服务中。

1 开通 Web Tracking 功能

开通方法请参考日志服务 Tracking 功能。

2 注册 Unity3D LogHandler

在Unity editor中创建C#文件 LogOutputHandler.cs，输入以下代码，并修改其中的三个成员变量，分别为：

project，表示日志项目名称

logstore，表示日志库名称

serviceAddr，表示日志项目的地址

serviceAddr请参考服务入口。

```
using UnityEngine;
using System.Collections;

public class LogOutputHandler : MonoBehaviour
{
```

```
//Register the HandleLog function on scene start to fire on debug.log events
public void OnEnable()
{
Application.logMessageReceived += HandleLog;
}

//Remove callback when object goes out of scope
public void OnDisable()
{
Application.logMessageReceived -= HandleLog;
}

string project = "your project name";
string logstore = "your logstore name";
string serviceAddr = "http address of your log service project";

//Capture debug.log output, send logs to Loggly
public void HandleLog(string logString, string stackTrace, LogType type)
{
string parameters = "";
parameters += "Level=" + WWW.EscapeURL(type.ToString());
parameters += "&";
parameters += "Message=" + WWW.EscapeURL(logString);
parameters += "&";
parameters += "Stack_Trace=" + WWW.EscapeURL(stackTrace);
parameters += "&";
//Add any User, Game, or Device MetaData that would be useful to finding issues later
parameters += "Device_Model=" + WWW.EscapeURL(SystemInfo.deviceModel);

string url = "http://" + project + "." + serviceAddr + "/logstores/" + logstore + "/track?APIVersion=0.6.0&" +
parameters;
StartCoroutine(SendData(url));
}

public IEnumerator SendData(string url)
{
WWW sendLog = new WWW(url);
yield return sendLog;
}
}
```

以上代码可以异步的将日志发送到阿里云日志服务中，在示例中您可以添加更多想要收集的字段。

3 产生Unity日志

在工程中创建 LogglyTest.cs 文件，并加入下面的代码：

```
using UnityEngine;
using System.Collections;

public class LogglyTest : MonoBehaviour {
```

```
void Start () {
Debug.Log ("Hello world");
}
}
```

4 在控制台预览日志

上述步骤做完之后，运行Unity程序，就可以在 日志服务控制台 预览您发送的日志了。如何预览日志请参考日志预览。

以上示例提供了 Debug.Log 或者Debug.LogError、Debug.LogException 等类似日志的收集方法。Unity的组件对象模型及其提供的程序崩溃API、其他各种LOG API使您可以非常方便的收集客户端的设备信息。

客户端 Logtail

数据源

Logtail客户端可以帮助日志服务用户简单地通过控制台收集ECS云服务器上的日志。

创建完日志库后，系统会提示您创建Logtail配置，您可以在弹出的对话框中单击 **创建 Logtail 配置** 创建一个Logtail配置。此外，您也可以通过 **Logstore列表** 页面创建Logtail配置。

前提条件

设置使用Logtail收集日志前，您需要安装Logtail。Logtail支持Windows和 Linux两大操作系统，安装方法参见 安装Logtail（Windows系统） 和 安装 Logtail（Linux系统）。

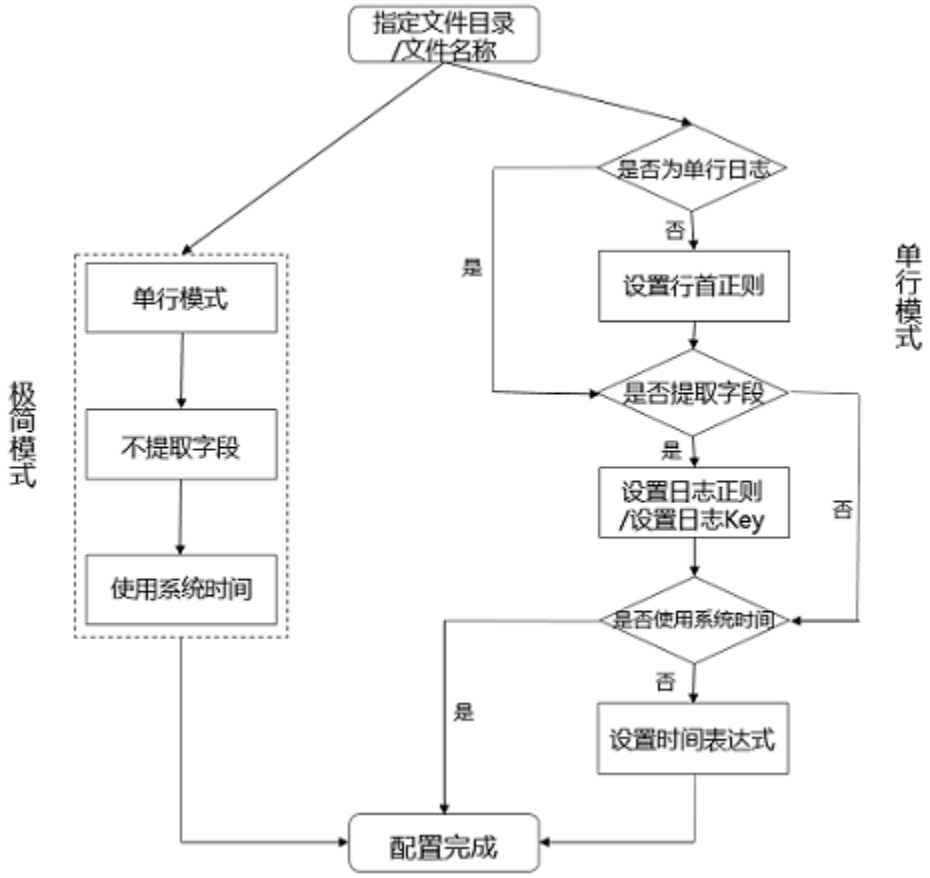
使用限制

- 一个文件只能被一个配置收集（如果需要采集多份，建议以软链接形式实现。例如 /home/log/nginx/log下需要采集两份，则其中一个配置原始路径，另外创建一个该文件夹的软链接 ln -s /home/log/nginx/log /home/log/nginx/link_log，另一个配置软链接路径即可)。
- Logtail客户端支持的操作系统可参考 使用 Logtail 写入日志。

日志收集配置流程

通过控制台配置Logtail收集文本日志，可以通过极简模式、分隔符模式、JSON 模式、完整正则模式等方式收

集日志进行设置。以极简模式和完整正则模式为例，配置流程如下。



操作步骤

在日志服务管理控制台单击目标项目，进入**Logstore列表**。

选择目标Logstore，并单击**管理**进入**Logtail配置列表**。

在**Logtail配置列表**页面右上角点击**创建**。进入Logtail配置流程。

选择数据源类型 **文本文件** 并单击 **下一步**。

指定 **配置名称**。

配置名称只能包含小写字母、数字、连字符（-）和下划线（_），且必须以小写字母和数字开头和结尾，长度为3~63字节。

注意：配置名称设置后不可修改。

指定日志的目录和文件名。

目录结构支持完整路径和通配符两种模式。

注意： 目录通配符只支持 *和? 两种。

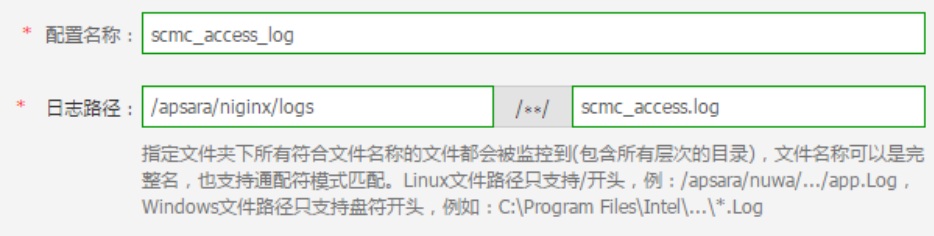
日志文件名支持完整文件名和通配符两种模式，文件名规则请参考Wildcard matching。

日志文件查找模式为多层目录匹配，即指定文件夹下所有符合文件名模式的文件都会被监控到，包含所有层次的目录。

- 例如/apsara/nuwa/ ... /*.log表示/apsara/nuwa目录中（包含该目录的递归子目录）后缀名为.log的文件。

  例如/var/logs/app_* ... /*.log*表示/var/logs目录下所有符合app_*模式的目录中（包含该目录的递归子目录）文件名包含.log的文件。

  注意： 一个文件只能被一个配置收集。



设置收集模式。

Logtail支持极简模式、分隔符模式、JSON 模式、完整正则模式等方式收集日志，具体说明请参考**采集模式**。本示例以极简模式和完整正则模式为例介绍收集模式的设置。

极简模式

目前极简模式即单行模式。单行模式下默认一行日志内容为一条日志，即日志文件中，以换行符分隔两条日志。单行模式下，不提取日志字段，即默认正则表达式为(.*)，同时记录当前服务器的系统时间作为日志产生的时间。如果后续您需要对极简模式进行更详细的设置，可以通过修改配置进入完整模式逐项调整。有关如何修改Logtail配置，参见 Logtail配置。

极简模式下，您只需要指定文件目录和文件名称，Logtail会按照每行一条日志进行收集，同时将日志时间设定为抓取该条日志时服务器的系统时间，不会提取日志内容中的字段。

完整正则模式

如果需要对内容做更多个性化的字段提取设置（比如跨行日志，提取字段等），选择 **完整正则模式** 即可进行个性化定制。您可以参考 使用Logtail写入日志 了解这些参数的具体含义和设置方式。

输入 **日志样例**。

让您提供日志样例的目的是方便日志服务控制台自动提取其中的正则匹配模式，请务必使用实际场景的日志。

关闭 **单行模式**。

默认为使用单行模式，即按照一行为一条日志进行分割，如果需要收集跨行日志（比如 Java 程序日志），需要关闭 **单行模式**，然后设置 **行首正则表达式**。

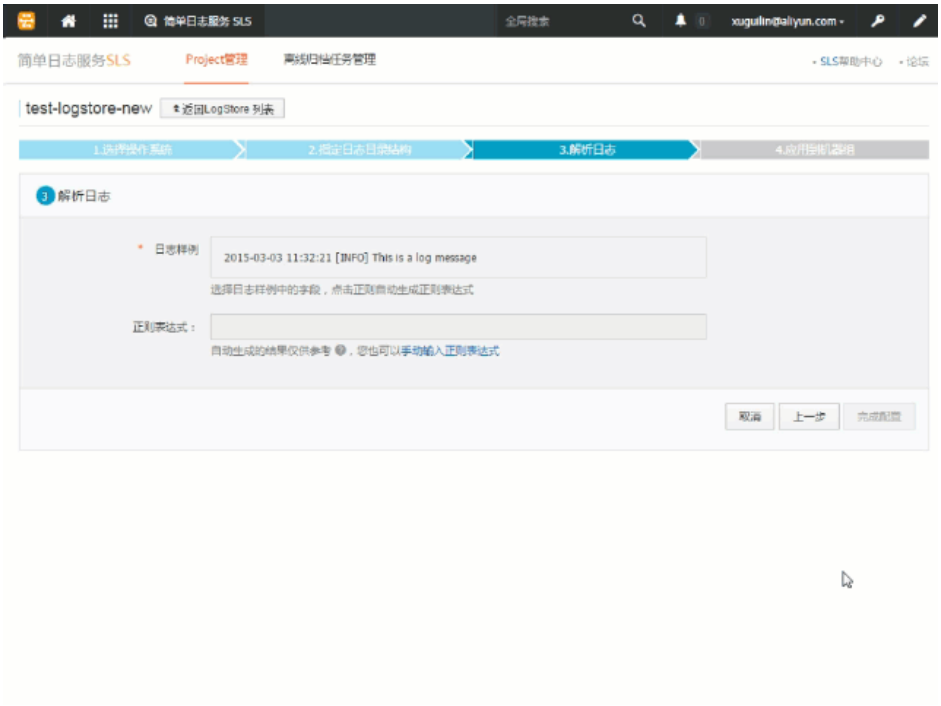
设置 **行首正则表达式**。

提供自动生成和手动输入两种功能。填写完日志样例后，单击 **自动生成** 即会生成正则；如果无法自动生成，可以切换为手动模式输入进行验证。

设置 **提取字段**。

如果需要对日志内容中的字段单独分析处理，可以使用 **提取字段** 功能将指定字段变成 Key-Value 对后发送到服务端，所以需要您指定解析一条日志内容的方式，即正则表达式。

日志服务控制台提供两种方式让您指定解析正则表达式。第一种方式是通过简单交互自动生成正则表达式。您通过“划选”的方式操作日志样例，选中需要提取的字段，日志服务控制台会自动生成正则表达式，如下图所示：



尽管自动生成方式避免了您自己写正则表达式的困扰，但是自动生成的正则表达式很多时候并不是完美的，您可以手动直接输入正则表达式。单击 **手动输入正则表达式** 切换到手动输入模式。手动输入完成后，单击右侧的 **验证** 即会验证您输入的正则表达式是否可以解析、提取日志样例。

无论使用自动生成还是手动输入方式，产生日志解析正则表达式后，您都需要给每个提取字段命名，设定对应字段的 Key，如下图所示：

设置 **使用系统时间**。

默认设置 **使用系统时间**。如果关闭，您需要在提取字段时指定某一字段（Value）为时间字段，并命名为 **time**（如上图）。在选取 time 字段后，您可以单击 **时间转换格式** 中的 **自动生成** 生成解析该时间字段的方式。关于日志时间格式的更多信息请参考 Logtail 日期格式。

酌情配置 **高级选项**。

请根据您的需求配置**本地缓存**、**配置日志Topic生成方式**、**日志文件编码**、**最大监控目录深度**、**超时属性**和**过滤器配置**。如没有特殊需求，可以保持默认配置。

配置项	详情
本地缓存	请选择是否打开**本地缓存**。当日志服务不可用时，日志可以缓存在机器本地目录，服务恢复后进行续传，默认最大缓存值1GB。
Topic生成方式	• **空-不生成Topic**：默认选项，表示设置Topic为空字符串，在查询日志时不需要输入Topic即可查询。 • **机器组Topic属性**：设置Topic生成方式为机器组Topic属性，可以用于明确区分不同前端服务器产生的日志数据

	。 • **文件路径正则**：选择此项之后，您需要填写下方的**自定义正则**，用正则式从路径里提取一部分内容作为Topic。可以用于区分具体用户或实例产生的日志数据。
自定义正则	如您选择了**文件路径正则**方式生成Topic，需要在此处填写您的自定义正则式。
日志文件编码	• utf8：指定使用UTF-8编码。 • gbk：指定使用GBK编码。
最大监控目录深度	指定从日志源采集日志时，监控目录的最大深度，即最多监控几层日志。最大目录监控深度范围0-1000，0代表只监控本层目录。
超时属性	如果一个日志文件在指定时间内没有任何更新，则认为该文件已超时。您可以对**超时属性**指定以下配置。 • 永不超时：指定持续监控所有日志文件，永不超时。 • 30分钟超时：如日志文件在30分钟内没有更新，则认为已超时，并不再监控该文件。
过滤器配置	日志只有**完全符合**过滤器中的条件才会被收集。 例如：配置Key:level Regex:WARNING|ERROR 代表只收集level为WARNING或ERROR类型的日志；若需要**过滤不符合某条件的数据**，也可用该方式实现，例如：Key:level Regex:^(?!.*(INFO|DEBUG)) 代表不收集level为INFO或DEBUG类型的日志，类似示例可参考regex-exclude-word、regex-exclude-pattern。

设置完成后，单击 **下一步**。

勾选所需的机器组并单击 **应用到机器组** 将配置应用到机器组。

如果您还未创建机器组，需要先创建一个机器组。有关如何创建机器组，参见 **创建机器组**。

注意：

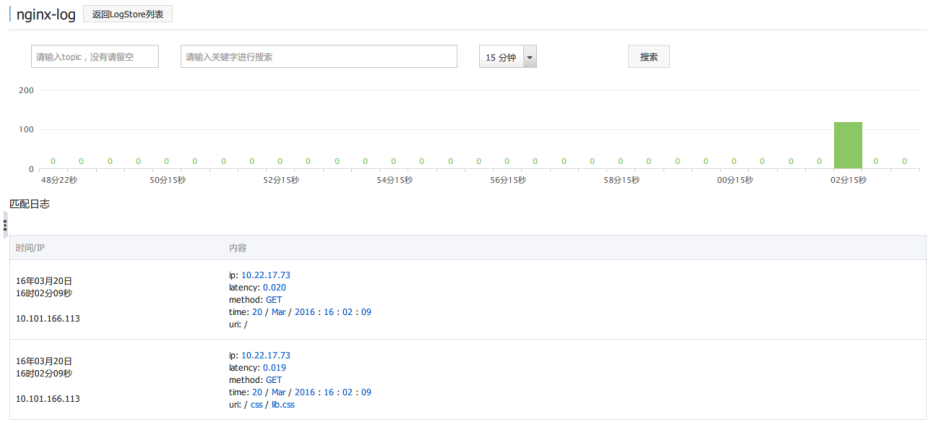
- Logtail配置推送生效时间最长需要 3 分钟，请耐心等待。
- 如果需要收集 IIS 的访问日志，请务必首先参考 **IIS 日志收集最佳实践** 配置 IIS。
- 创建Logtail配置后，您可以查看Logtail配置列表，修改Logtail配置或删除Logtail配置。详细信息，参见 **Logtail 配置**。

后续操作

完成配置后，日志服务开始收集日志。您可以查看并检索收集到的日志。有关如何查询日志，参见 **查询日志**。

极简模式下收集到服务端的日志如下所示。每条日志的所有内容都在名为 **content** 的KEY下面。

完整正则模式下，收集到服务端的日志内容如下所示。每条日志的内容都按照设定的 Key-Value收集到了服务端。



Logtail配置项

配置Logtail时需要填写配置项，常用配置项具体描述与限制如下：

配置项	描述
日志路径	目录结构支持完整路径和通配符两种模式。通配符模式为多层目录匹配，即指定文件夹下所有符合文件名称的文件都会被监控到，包含所有层次的目录。
日志文件名	指定收集日志文件名称，区分大小写，可以使用通配符。例如*.log。Linux下的文件名通配符包括“*”，“?”和“[...]”。
本地存储	表示是否启用本地缓存临时存储因网络短暂中断而无法发送的日志。
日志首行头	指定多行日志的起始头，需指定正则表达式。在多行日志收集场景下（如应用程序日志中的堆栈信息），无法使用行来分割每条日志。这时需要指定一个多行日志的起始头，当发现该起始头则表示上条日志已经结束，新的一条已经开始。由于每条日志的起始头可能并不一样（如时间戳），故需要指定一个起始头的匹配规则，即这里的正则表达式。
日志解析表达式	定义如何提取一条日志信息，并转化成为日志服务日志的格式。用户需要指定一个正则表达式提取需要的日志字段信息，并且定义每个提取的字段名称。
日志时间格式	定义如何解析日志数据中的时间戳字符串的时间格式，具体请参见Logtail日志时间格式。

日志写入方式

除了使用Logtail收集日志外，日志服务还提供API和SDK的方式，以方便您写入日志。

使用 API 写入日志

日志服务提供REST风格的API帮助您写入日志。您可以通过API中的 PostLogStoreLogs 接口写入数据。关于

API的完整参考请见 API 参考。

使用 SDK 写入日志

除了API，日志服务还提供了多种语言（Java、.NET、PHP 和 Python）的SDK方便您写入日志。关于SDK的完整参考请见 SDK 参考。

配置Logtail收集文本文件流程

Logtail按照如下步骤收集日志内容：

指定文件路径名称 > 指定日志行分割方式 > 提取日志字段内容 > 指定日志时间

指定日志行分割方式

一般完整的一条访问日志为一行一条，比如nginx的访问日志，每条日志以换行符分割。例如，两条单行的访问日志如下：

```
10.1.1.1 - - [13/Mar/2016:10:00:10 +0800] "GET / HTTP/1.1" 0.011 180 404 570 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; 360se)"
10.1.1.1 - - [13/Mar/2016:10:00:11 +0800] "GET / HTTP/1.1" 0.011 180 404 570 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; 360se)"
```

Java应用中的程序日志，一条日志通常会跨越多行，因此只能通过日志开头的特征区分每条日志行首，例如以下 Java 程序日志：

```
[2016-03-18T14:16:16,000] [INFO] [SessionTracker] [SessionTrackerImpl.java:148] Expiring sessions
0x152436b9a12aecf, 50000
0x152436b9a12aed2, 50000
0x152436b9a12aed1, 50000
0x152436b9a12aed0, 50000
```

以上Java日志起始字段均为时间格式，可以用“行首正则表达式”描述为 \[\d+-\d+-\w+:\d+:\d+,\d+]\s.*。在控制台可按照如下格式填写：

提取日志字段内容

根据日志服务数据模型要求，一条日志的内容包含一个或者多个 Key-Value 对，如果需要提取指定字段进行分析处理，需要设置正则表达式提取指定内容，如果不需要对日志内容进行处理，可以将整条日志做为一个 Key-Value 对。对于如上访问日志：

提取字段

```
正则表达式：(\S+)\s-\s-\s\[(\S+)\s[^]]+]\s"(\w+).*
提取内容：1）10.1.1.1；2）13/Mar/2016:10:00 ；3）GET
```

不提取字段

```
正则表达式：(.*)
提取内容：1）10.1.1.1 - - [13/Mar/2016:10:00:10 +0800] "GET / HTTP/1.1" 0.011 180 404 570 "-"
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; 360se)"
```

指定日志时间

根据日志服务数据模型要求，一条日志必须要有时间（time）字段，并且格式为unix时间戳。目前提供使用系统时间（即Logtail抓取该条日志的时间）或者日志内容中的时间字段做为日志的时间。

对于上例中的访问日志：

提取日志内容中的时间字段

```
时间：13/Mar/2016:10:00:10
时间表达式：%d/%b/%Y:%H:%M:%S
```

抓取日志时的系统时间

```
时间：抓取日志时的时间戳
```

正如日志服务 核心概念 中所描述，每条日志服务日志都必须包括该日志发生的时间戳信息。Logtail接入服务在采集用户日志文件中的日志数据，必须提取该条日志中时间戳字符串并把它解析为时间戳。因此，Logtail需要您指定其日志的时间戳格式帮助解析。

Linux平台下的Logtail支持 strftime 函数 提供的所有时间格式。只需要您的日志时间戳字符串能够被该函数定义的日志格式所表达，即可以被Logtail解析并使用。

现实环境中的日志时间戳字符串格式非常多样化，为方便用户配置，Logtail支持的常见日志时间格式如下：

支持格式	格式意义	示例(说明)
%a	星期缩写	例如：Fri
%A	星期全称	例如：Friday
%b	月份缩写	例如：Jan
%B	月份全称	例如：January
%d	十进制表示的每月第几天 [01,31]	例如：07, 31
%h	月份缩写，同 (%b)	例如：Jan
%H	24小时制的小时	例如：22
%I	12小时制的小时	例如：11
%m	十进制表示的月份	例如：08
%M	十时制表示的分钟数 [00,59]	例如：59
%n	换行符	换行符
%p	本地的AM（上午）或PM（下午）	例如：AM/PM
%r	12小时制的时间组合，同 (%I:%M:%S %p)	例如：11:59:59 AM
%R	小时和分钟组合，同 (%H:%M)	例如：23:59
%S	十进制的秒数 [00,59]	例如：59
%t	TAB符	TAB符
%y	不带世纪的十进制年份 [00,99]	例如：04,98
%Y	十进制年份	例如：2004,1998
%z	时区或者缩写	例如：-07:00, +0800
%C	十进制世纪 [00-99]	例如：16

%e	十进制表示的每月第几天 [1,31]; 单独的数字前面需要有空格	例如：7, 31
%j	一年天数的十进制表示 [001,366]	例如：365
%u	星期的十进制表示 [1,7], 1 代表周一	例如：2
%U	每年的第几周（星期天认为是一周的开始） [00,53]	例如：23
%V	每年的第几周（星期一认为是一周的开始），如果一月份刚开始的一周>=4天，则认为是第1周，否则认为下一个星期一是第1周 [01,53]	例如：24
%w	星期的十进制表示 [0,6], 0 代表周日	例如：5
%W	每年的第几周（星期一认为是一周的开始） [00,53]	例如：23
%c	标准的日期、时间表示	需要指定长日期、短日期等更多信息，可以考虑用上面支持的格式更精确表达
%x	标准的日期表示	需要指定长日期、短日期等更多信息，可以考虑用上面支持的格式更精确表达
%X	标准的时间表示	需要指定长日期、短日期等更多信息，可以考虑用上面支持的格式更精确表达
%s	unix 时间戳	例如：1476187251

Logtail支持在本地配置TCP端口，接收syslog Agent通过TCP协议转发过来的syslog数据，Logtail解析接收到的数据并转发至LogHub中。

前提条件

设置使用Logtail收集日志前，您需要安装Logtail。Logtail支持Windows和Linux两大操作系统，安装方法参见 安装Logtail（Windows） 和 安装Logtail（Linux）。

操作步骤

收集syslog数据，您需要进行如下设置。

步骤 1 在日志服务管理控制台创建Logtail syslog配置

在日志服务管理控制台单击目标项目，进入**Logstore列表**。

选择目标Logstore，并单击**管理**进入**Logtail配置列表**。

在**Logtail配置列表**页面右上角点击**创建**。进入Logtail配置流程。

选择数据源类型 **syslog** 并单击 **下一步**。

设置 **配置名称** 和 **tag设置** 并单击 **下一步**。

配置名称只能包含小写字母、数字、连字符（-）和下划线（_），且必须以小写字母和数字开头和结尾，长度为3~63字节。

注意：配置名称设置后不可修改。

勾选所需的机器组并单击 **应用到机器组** 将配置应用到机器组。

如果您未创建机器组，请参考**创建机器组**创建一个机器组，再将配置应用到改机器组。

酌情配置 **高级选项**。

请选择是否打开**本地缓存**。当日志服务不可用时，日志可以缓存在机器本地目录，服务恢复后进行续传。默认开启缓存，最大缓存值1GB。

步骤 2 配置Logtail使协议生效

从机器Logtail安装目录下找到 ilogtail_config.json，一般在 /usr/local/ilogtail/ 目录下面。根据需求修改和syslog相关的配置。

确认syslog功能已开启。

true表示syslog功能处于打开状态，false表示关闭状态。

```
"streamlog_open" : true
```

2. 配置syslog用于接收日志的内存池大小。程序启动时会一次性申请指定大小的内存，请根据机器内存大小以及实际需求填写，单位是MB。

```
"streamlog_pool_size_in_mb" : 50
```

3. 配置缓冲区大小。需要配置Logtail每次调用socket io rcv 接口使用的缓冲区大小，单位是byte。

```
"streamlog_rcv_size_each_call" : 1024
```

4. 配置日志syslog格式。

```
"streamlog_formats":[]
```

5. 配置TCP端口。需要配置Logtail用于接收syslog日志的TCP端口，默认是11111。

```
"streamlog_tcp_port" : 11111
```

配置完成后重启Logtail。重启Logtail要执行以下命令关闭Logtail客户端，并再次打开。

```
sudo /etc/init.d/ilogtaild stop
sudo /etc/init.d/ilogtaild start
```

步骤 3 安装rsyslog并修改配置

如果机器已经安装rsyslog，忽略这一步。

安装方法请参见：

- Ubuntu 安装方法
- Debian 安装方法
- RHEL/CENTOS 安装方法

在 /etc/rsyslog.conf 中根据需要修改配置，例如：

```
$WorkDirectory /var/spool/rsyslog # where to place spool files
$ActionQueueFileName fwdRule1 # unique name prefix for spool files
$ActionQueueMaxDiskSpace 1g # 1gb space limit (use as much as possible)
$ActionQueueSaveOnShutdown on # save messages to disk on shutdown
$ActionQueueType LinkedList # run asynchronously
$ActionResumeRetryCount -1 # infinite retries if host is down
# 定义日志数据的字段
$template ALI_LOG_FMT,"0.1 sys_tag %timegenerated:::date-unixtimestamp% %fromhost-ip% %hostname% %pri-text% %protocol-version% %app-name% %procid% %msgid% %msg:::drop-last-lf%\n"
*.* @@10.101.166.173:11111;ALI_LOG_FMT
```

注意：模板 ALI_LOG_FMT 中第二个域的值是 sys_tag，这个取值必须和步骤 1 中创建的一致，这个配置的含义是将本机接收到的所有（*.*）syslog 日志按照 ALI_LOG_FMT 格式化，使用 TCP 协议转发到 10.101.166.173:11111。机器 10.101.166.173 必须在步骤 1 中的机器组中并且按照步骤 2 配置。

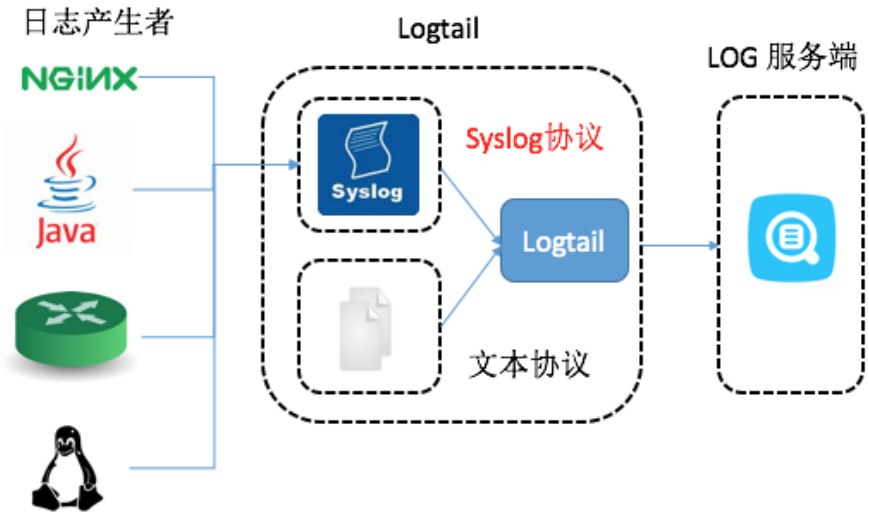
启动rsyslog（sudo /etc/init.d/rsyslog restart）。启动之前请先检查机器上是否安装了其他syslog的Agent，比如 syslogd、sysklogd、syslog-ng 等，如果有的话请关闭掉。

上面三步完成之后就可以将机器上的syslog收集到日志服务了。

更多信息

有关syslog日志采集的更多信息以及如何格式化syslog数据，参见 收集 syslog 数据详解。

Logtail目前支持的接入端为syslog和文本文件，如下图所示：



Logtail通过TCP协议支持syslog。配置Logtail采集syslog日志详细步骤请参见通过Logtail采集syslog日志。

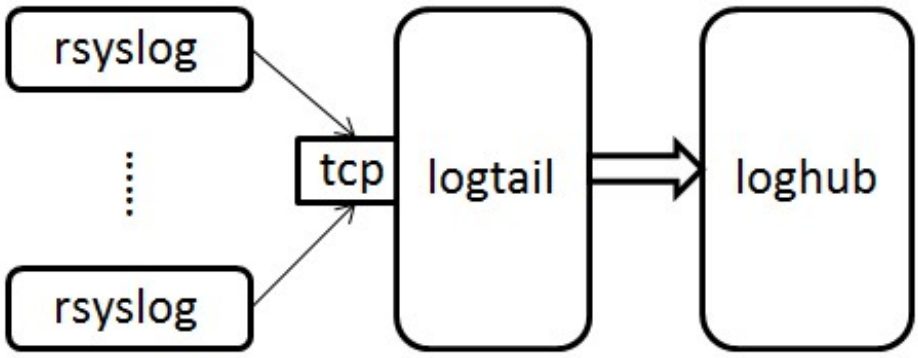
syslog优势

syslog概念请参考 鸟哥的 Linux 私房菜。

和利用文本文件相比，使用syslog时日志数据直接收集到LogHub，不会保存到磁盘，保密性好。免去了文件缓存和解析的代价，单机可达80MB/s吞吐率。

基本原理

Logtail 支持在本地配置TCP端口，接收syslog Agent转发的日志。Logtail开启TCP端口，接收rsyslog或者其他syslog Agent通过TCP协议转发过来的syslog数据，Logtail解析接收到的数据并转发到LogHub中。配置Logtail采集syslog日志过程请参见通过Logtail采集syslog日志。Logtail、syslog、LogHub三者之间的关系如下图所示。



syslog日志格式

Logtail通过TCP端口接收到的数据是流式的，如果要从流式的数据中解析出一条条的日志，日志格式必须满足

以下条件：

- 每条日志之间使用换行符（\n）分隔，一条日志内部不可以出现换行符。
- 日志内部消息正文可以包含空格，其他字段不可以包含空格。

syslog日志格式如下：

```
$version $tag $unixtimestamp $ip [$user-defined-field-1 $user-defined-field-2 $user-defined-field-n] $msg\n"
```

各个字段含义：

日志字段	含义
version	该日志格式的版本号，Logtail使用该版本号解析user-defined-field 字段。
tag	数据标签，用于寻找Project或Logstore，不可以包含空格和换行符。
unixtimestamp：	该条日志的时间戳。
ip	该条日志的对应的机器IP，如果日志中的该字段是127.0.0.1，最终发往服务端的日志数据中该字段会被替换成TCP socket的对端地址。
user-defined-field	用户自定义字段，中括号表示是可选字段，可以有0 个或多个，不可以包含空格和换行符。
msg	日志消息正文，不可以包含换行符，末尾的 \n 表示换行符。

以下示例日志即为满足格式要求的日志：

```
2.1 streamlog_tag 1455776661 10.101.166.127 ERROR com.alibaba.streamlog.App.main(App.java:17) connection refused, retry
```

另外，不仅 syslog 日志可以接入Logtail，任何日志工具只要能满足以下条件，都可以接入:

- 可以将日志格式化，格式化之后的日志格式满足以上要求。
- 可以通过TCP协议将日志Append到远端。

Logtail解析syslog日志规则

Logtail 需要增加配置以解析syslog日志。例如：

```
"streamlog_formats":
[
{"version": "2.1", "fields": ["level", "method"]},
{"version": "2.2", "fields": []},
{"version": "2.3", "fields": ["pri-text", "app-name", "syslogtag"]}
]
```

Logtail通过读取到的version字段到streamlog_formats中找到对应的user-defined字段的格式，应用该配置，上面的日志样例version字段为 2.1，包含两个自定义字段level和method，因此日志样例将被解析为如下格式：

```
{
"source": "10.101.166.127",
"time": 1455776661,
"level": "ERROR",
"method": "com.alibaba.streamlog.App.main(App.java:17)",
"msg": "connection refused, retry"
}
```

version用于解析user-defined字段，tag用于寻找数据将要被发送到的 Project/Logstore，这两个字段不会作为日志内容发送到阿里云日志服务。另外，Logtail预定义了一些日志格式，这些内置的格式都使用 0.1、1.1 这样以“0.”、“1.”开头的version值，所以用户自定义version不可以以“0.”、“1.”开头。

常见日志工具接入Logtail syslog log

log4j

引入 log4j 库

```
 <dependency>
<groupId>org.apache.logging.log4j</groupId>
<artifactId>log4j-api</artifactId>
<version>2.5</version>
</dependency>
<dependency>
<groupId>org.apache.logging.log4j</groupId>
<artifactId>log4j-core</artifactId>
<version>2.5</version>
</dependency>
```

程序中引入 log4j 配置文件 log4j_aliyun.xml

```
 <?xml version="1.0" encoding="UTF-8"?>
<configuration status="OFF">
<appenders>
<Socket name="StreamLog" protocol="TCP" host="10.101.166.173" port="11111">
<PatternLayout pattern="%X{version} %X{tag} %d{UNIX} %X{ip} %-5p %l %enc{%m}%n" />
</Socket>
</appenders>
<loggers>
<root level="trace">
<appender-ref ref="StreamLog" />
</root>
</loggers>
```

```
</configuration>
```

其中 10.101.166.173:11111是安装了Logtail客户端的服务器地址。

程序中设置ThreadContext

```
package com.alibaba.streamlog;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.ThreadContext;

public class App
{
private static Logger logger = LogManager.getLogger(App.class);
public static void main( String[] args ) throws InterruptedException
{
ThreadContext.put("version", "2.1");
ThreadContext.put("tag", "streamlog_tag");
ThreadContext.put("ip", "127.0.0.1");
while(true)
{
logger.error("hello world");
Thread.sleep(1000);
}
//ThreadContext.clearAll();
}
}
```

tengine

tengine可以通过syslog接入ilogtail.

tengine使用ngx_http_log_module模块将日志打入本地syslog Agent，在本地 syslog Agent中forward到rsyslog。

tengine配置syslog请参考：tengine配置syslog

示例：

以user类型和info级别将access log发送给本机Unix dgram(/dev/log)，并设置应用标记为nginx。

```
access_log syslog:user:info:/var/log/nginx.sock:nginx
```

rsyslog配置：

```
module(load="imuxsock") # needs to be done just once
input(type="imuxsock" Socket="/var/log/nginx.sock" CreatePath="on")
$template ALI_LOG_FMT,"2.3 streamlog_tag %timegenerated:::date-unixtimestamp% %fromhost-ip% %pri-text%
%app-name% %syslogtag% %msg:::drop-last-lf%\n"
```

```
if $syslogtag == 'nginx' then @@10.101.166.173:11111;ALI_LOG_FMT
```

nginx

以收集 nginx accesslog 为例。

access log 配置：

```
access_log syslog:server=unix:/var/log/nginx.sock,nohostname,tag=nginx;
```

rsyslog 配置：

```
module(load="imuxsock") # needs to be done just once
input(type="imuxsock" Socket="/var/log/nginx.sock" CreatePath="on")
$template ALI_LOG_FMT,"2.3 streamlog_tag %timegenerated:::date-unixtimestamp% %fromhost-ip% %pri-text%
%app-name% %syslogtag% %msg:::drop-last-lf%\n"
if $syslogtag == 'nginx' then @@10.101.166.173:11111;ALI_LOG_FMT
```

参考：nginx

Python Syslog

示例：

```
import logging
import logging.handlers

logger = logging.getLogger('myLogger')
logger.setLevel(logging.INFO)

#add handler to the logger using unix domain socket '/dev/log'
handler = logging.handlers.SysLogHandler('/dev/log')

#add formatter to the handler
formatter = logging.Formatter('Python: { "loggerName":"%(name)s", "asciTime":"%(asctime)s",
"pathName":"%(pathname)s", "logRecordCreationTime":"%(created)f", "functionName":"%(funcName)s",
"levelNo":"%(levelno)s", "lineNo":"%(lineno)d", "time":"%(msecs)d", "levelName":"%(levelname)s",
"message":"%(message)s"}')

handler.formatter = formatter
logger.addHandler(handler)

logger.info("Test Message")
```

机器组

Logtail 配置

Logtail客户端可以帮助日志服务用户简单的通过控制台就能收集ECS云服务器上的日志。除了安装Logtail客户端外（参见 安装Logtail（Windows） 和 安装Logtail（Linux）），为Logtail客户端创建日志收集配置也非常关键。您可以通过日志库列表给相应日志库创建、修改Logtail配置。

如果需要收集IIS的访问日志，请参考 IIS 日志收集最佳实践 来进行配置。

创建Logtail配置

有关如何通过日志服务管理控制台创建Logtail配置，参见使用Logtail收集文本文件 和 使用Logtail收集 syslog 数据。

查看Logtail配置列表

登录 日志服务管理控制台。

选择所需的项目，单击项目名称或者单击右侧的 **管理**。

在 **Logstore列表** 页面，单击日志收集模式列下的Logtail配置 **管理**，进入 **Logtail配置列表** 页面。

该页面列出了指定Logstore对应的所有配置，其中包含三部分内容：配置名称、数据来源和配置详情，其中当数据来源为 **文本文件** 时，配置详情展示了文件路径和文件名称，如下图所示。

注意：一个文件只能被一个配置收集。

修改Logtail配置

登录 日志服务管理控制台。

选择所需的项目，单击项目名称或者单击右侧的 **管理**。

在 **Logstore列表** 页面，单击日志收集模式列下的Logtail配置 **管理**，进入 **Logtail配置列表** 页面。

单击需要修改的Logtail配置的名称。

您可以修改日志的收集模式并重新指定应用到的机器组。整个配置修改的流程和创建完全相同。

删除Logtail配置

登录 日志服务管理控制台。

选择所需的项目，单击项目名称或者单击右侧的 **管理**。

在 **Logstore列表** 页面，单击日志收集模式列下的Logtail配置 **管理**，进入 **Logtail配置列表** 页面。

选择需要删除的Logtail配置并单击右侧的 **删除**。

删除成功后即会将该配置与之前应用机器组解除绑定，Logtail也会停止收集该配置对应的日志文件内容。

注意：删除指定Logstore前必须删除其对应的所有Logtail配置。

Logtail启动后会汇报机器标识到服务端，当客户端汇报的标识与机器组里的标识保持一致时，Logtail才能正常工作。

应用场景

- Logtail默认使用机器IP地址作为标识，在自定义网络环境下（如VPC）可能出现不同机器IP地址冲突的问题，导致服务端无法管理Logtail。
- 多台机器使用相同自定义机器标识以支持扩容时自动收集日志（如何使用？）。

开启 userdefined-id

Linux Logtail

通过文件 /etc/ilogtail/user_defined_id 来设置userdefined-id。

例如，设置自定义机器标识如下：

```
#cat /etc/ilogtail/user_defined_id
aliyun-ecs-rs1e16355
```

Windows Logtail

通过文件 C:\LogtailData\user_defined_id 来设置userdefined-id。

例如，设置自定义机器标识如下：

```
C:\LogtailData>more user_defined_id
aliyun-ecs-rs1e16355
```

添加 aliyun-ecs-rs1e16355 到机器组，1分钟之内即可生效。

> **注意**：若目录 /etc/ilogtail/、C:\LogtailData或文件 /etc/ilogtail/user_defined_id、C:\LogtailData\user_defined_id不存在，请手动创建。

禁用 userdefined-id

如果想恢复使用机器IP作为标识，请删除user_defined_id文件，1分钟之内即可生效。

Linux Logtail

```
rm -f /etc/ilogtail/user_defined_id
```

Windows Logtail

```
del C:\LogtailData\user_defined_id
```

生效时间

新增、删除、修改user_defined_id文件后，默认情况下，1分钟之内即可生效。

如需立即生效，请执行以下命令重启Logtail：

Linux Logtail

```
/etc/init.d/ilogtaild stop
/etc/init.d/ilogtaild start
```

Windows Logtail

Windows控制面板 -> **管理工具** -> **服务**，在服务列表中右键点击LogtailWorker服务，选择**重新启动**以使配置生效。

日志服务通过机器组的方式管理所有需要通过Logtail客户端收集日志的ECS云服务器。您可以通过日志服务项目列表进入该项目的 **机器组列表** 页面。您可以通过日志服务创建机器组，查看机器组列表，修改机器组，查看机器组状态，管理配置，删除机器组，使用机器组标识。

创建机器组

您可以通过如下两种方法定义一个机器组：

- IP：定义机器组名称并添加一组机器的内网IP。
- 标识：定义属于机器组一个标识，在对应机器上配置对应标识进行关联。

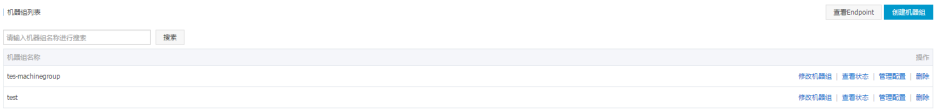
有关如何创建机器组，参见 创建机器组。

查看机器组列表

登录 日志服务管理控制台。

单击左侧导航栏的 **LogHub - 实时采集** > **Logtail机器组** 进入该项目的 **机器组列表** 页面。

您可以查看该项目下的所有机器组。



修改机器组

在创建完机器组后，您可以随时调整该机器组内的服务器列表。

注意：机器组名称在创建时选定后不可更改。

登录 日志服务管理控制台。

单击左侧导航栏的 **LogHub - 实时采集** > **Logtail机器组** 进入该项目的 **机器组列表** 页面。

选择机器组，单击 **修改机器组**。

修改机器组的配置信息并单击 **确认**。

查看状态

为验证Logtail客户端已经在机器组内的所有服务器安装成功，您可以查看Logtail 客户端的心跳信息。

登录 日志服务管理控制台。

单击左侧导航栏的 **LogHub - 实时采集** > **Logtail机器组** 进入该项目的 **机器组列表** 页面。

选择机器组，单击 **查看状态**。

如果所有服务器上的Logtail客户端都安装成功，则心跳信息应该都为**OK**。如果心跳信息为**FAIL**，建议首先按照页面提示进行自查。如自查仍无法解决问题，可通过工单寻求帮助。

注意：

- 心跳“OK”指Logtail客户端与日志服务连接正常。机器加入机器组后可能会有几分钟左右的延时才能看到心跳“OK”状态，请耐心等待。
- 如服务器心跳长期处于“FAIL”状态，请按照 安装Logtail（Windows）和 安装Logtail（Linux）进行操作。

管理配置

日志服务利用机器组管理所有需要收集日志的服务器，这其中的一个重要管理项目就是Logtail客户端的收集配置（请参考 使用Logtail收集文本文件 和 使用Logtail收集syslog数据）。您可以通过给一个机器组应用、删除Logtail配置来决定每台服务器上的Logtail收集哪些日志，如何解析这些日志，发送日志到哪个日志库等。

登录 日志服务管理控制台。

单击左侧导航栏的 **LogHub - 实时采集** > **Logtail机器组** 进入该项目的 **机器组列表** 页面。

选择机器组，单击 **管理配置**。

您可以选择所需的配置并单击 **添加** 或 **删除** 来修改应用到机器组的 Logtail配置。

当添加Logtail配置时，该Logtail配置就会下发到机器组内服务器的Logtail客户端。当移除Logtail配置时，该Logtail配置会从Logtail客户端移除。

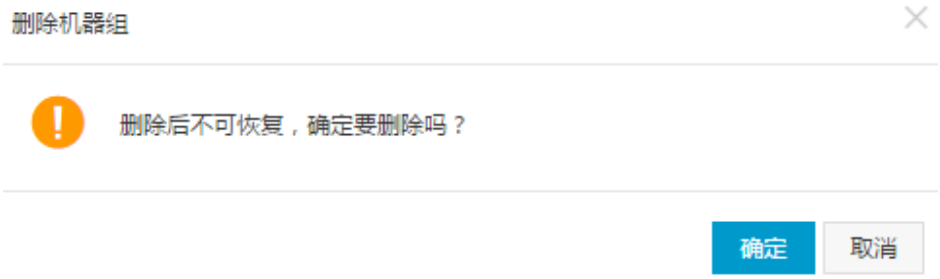
删除机器组

登录 日志服务管理控制台。

单击左侧导航栏的 **LogHub - 实时采集** > **Logtail机器组** 进入该项目的 **机器组列表** 页面。

选择机器组，单击 **删除**。

在确认对话框里，单击 **确定**。



日志服务通过机器组的方式管理所有需要通过Logtail客户端收集日志的ECS云机器。

创建Logtail配置后，您可以在日志服务项目列表进入该项目的 **机器组列表** 页面进行创建。此外，您也可以根据页面提示，在 **应用到机器组** 页面，单击 **创建机器组** 来创建机器组。

您可以通过如下两种方法定义一个机器组：

IP：定义机器组名称并添加一组机器的内网IP。

您可以通过添加阿里云ECS内网IP的方式，直接将多台ECS添加到一个机器组中，为其统一Logtail配置。非ECS服务器创建机器组可以参考非阿里云ECS配置用户标识。

标识：定义属于机器组的一个标识，在对应机器上配置对应标识进行关联。

系统由多个模块组成，每个模块每部分都可以进行单独的水平扩展、可以包含多台机器，为每个模块分别创建机器组，可以达到分类采集日志的目的。因此需要为每个模块分别创建自定义标识，并在各个模块的服务器上配置各自所属的标识。例如常见网站分为前端 HTTP 请求处理模块、缓存模块、逻辑处理模块和存储模块，其自定义标识可以分别定义为http_module、cache_module、logic_module和store_module。

操作步骤

在日志服务控制台 **Project列表** 页面单击项目名称，进入该项目的Logstore列表。

单击左侧导航栏的 **LogHub - 实时采集** > **Logtail机器组** 进入该项目的 **机器组管理** 页面 > 单击 **创建机器组**。

或者在创建Logtail配置后，于 **应用到机器组** 页面，单击 **创建机器组**。

填写 **机器组名称**。

名称只能包含小写字母、数字、连字符（-）和下划线（_），且必须以小写字母和数字开头和结尾，长度为3~128字节。

选择 **机器组标识**。

IP 地址

选择此选项后您需要在 **IP地址** 处填写云服务器的内网 IP。

注意：

- 请确保您填写的云服务器为此登录云账号所有。
- 请确保您填写的云服务器和当前日志服务Project在同一阿里云Region。
- 请确保使用ECS云服务器的内网IP。添加多个IP时，请换行后再输入。
- 请不要把Windows云服务器和Linux云服务器添加到同一机器组。
- 目前日志服务已经关闭云盾远程安装Logtail客户端功能，请您按照 自助安装Logtail文档进行操作。

用户自定义标识

选择此选项后您需要在 **用户自定义标识** 处填写您自定义的标识。

执行此步骤前，请确认您已经在需要采集日志的服务器上创建了用户自定义标识。有关如何使用用户自定义标识，请参见**用户自定义标识**。

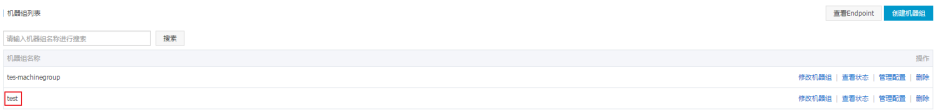
当模块需要扩容机器时，比如前端模块增加服务器，只需要在新增服务器上安装Logtail和创建自定义标识为http_module的文件即可自动同步不同机器组的配置，成功执行操作后可以在机器**查看状态**中看到新增机器。

填写 **机器组Topic**。

有关如何使用机器组 Topic，参见 如何使用机器组 Topic。

单击 **确定**。

您可以在机器组列表里看到已创建的机器组。



更多操作

创建完机器组之后，您可以查看机器组列表，修改机器组，查看机器组状态，管理配置或删除机器组。更多信息，参见 Logtail 机器组。

如果您需要通过Logtail采集日志的服务器并非阿里云ECS，或非属于本账号下创建ECS时，则需要在服务器上安装Logtail之后，配置用户标识（账号 ID），证明这台机器有选项被该账号访问联通。否则会显示心跳失败、无法采集数据到日志服务。请按照以下步骤配置。

1 安装Logtail

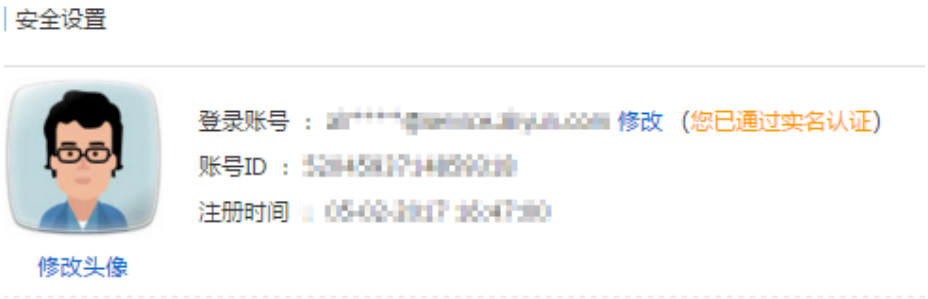
在您需要采集日志的服务器上安装Logtail客户端，Window系统请参见Logtail安装—Windows系统，Linux系

统请参见Logtail安装—Linux系统。

2 配置用户标识

2.1 查看阿里云账号 ID

登陆 阿里云账号管理页面，查看日志服务Project所属账号的ID。



2.2 在服务器上配置账号ID标识文件

Linux系统

创建账号ID同名文件到 /etc/ilogtail/users 目录，如目录不存在请手动创建，一台机器上可以配置多个账号ID，例如：

```
touch /etc/ilogtail/users/1559122535028493
touch /etc/ilogtail/users/1329232535020452
```

当不需要Logtail采集数据到该用户的日志服务Project后，可删除用户标识：

```
rm /etc/ilogtail/users/1559122535028493
```

Windows系统

创建账号ID同名文件到目录 C:\LogtailData\users以配置用户标识。如需删除用户标识，请直接删除此文件。

例如，C:\LogtailData\users\1559122535028493。

注意：

- 机器上配置账号ID标识后，表示该云账号有权限通过Logtail采集该机器上的日志数据。机器上不必要的账号标识文件请及时清理。

- 新增、删除用户标识后，1分钟之内即可生效。

异常排查

1. 简介

Logtail具备自身健康度以及日志采集进度查询的功能，便于您对于日志采集问题进行自检，同时您可基于该功能定制日志采集的状态监控。

2. 使用指南

确认已安装支持状态查询功能的Logtail客户端之后，在客户端输入相对命令即可查询本地采集状态。安装Logtail参见Logtail安装方法。

在客户端输入命令 /etc/init.d/ilogtaild -h，确认当前客户端是否支持本地采集状态查询功能。若输出logtail insight, version关键字则表示已安装支持此功能的Logtail。

```
/etc/init.d/ilogtaild -h
Usage: ./ilogtaild { start | stop (graceful, flush data and save checkpoints) | force-stop | status | -h for help}$
logtail insight, version : 0.1.0

commond list :
status all [index]
get logtail running status
status active [--logstore | --logfile] index [project] [logstore]
list all active logstore | logfile. if use --logfile, please add project and logstore. default --logstore
status logstore [--format=line | json] index project logstore
get logstore status with line or json style. default --format=line
status logfile [--format=line | json] index project logstore fileFullPath
get log file status with line or json style. default --format=line
status history beginIndex endIndex project logstore [fileFullPath]
query logstore | logfile history status.

index : from 1 to 60. in all, it means last $(index) minutes; in active/logstore/logfile/history, it means last $(index)*10 minutes
```

Logtail目前支持的查询命令、命令功能、可查询的时间区间以及结果统计的时间窗口如下：

命令	功能	可查询时间区间	统计窗口
all	查询Logtail的运行状态	最近60分钟	1分钟
active	查询当前活跃（有数据采集）的Logstore或日志文件	最近600分钟	10分钟
logstore	查询Logstore的采集状态	最近600分钟	10分钟
logfile	查询日志文件的采集状态	最近600分钟	10分钟
history	查询Logstore或日志文件一段时间内的采集状态	最近600分钟	10分钟

注意：

- 命令中的index参数代表查询的时间窗口索引值，有效值为1~60，从当前时间开始计算。若统计窗口是1分钟，则查询距当前(index, index-1]分钟内的窗口；若统计窗口是10分钟，则查询距当前(10*index, 10*(index-1)]分钟内的统计窗口
- 所有查询命令均属于status子命令，因此主命令为status

2.1. all命令

命令格式

```
/etc/init.d/ilogtaild status all [ index ]
```

说明：

all命令用来查看Logtail的运行状态。index为可选参数，不输入时默认代表1。

示例

```
/etc/init.d/ilogtaild status all 1
ok
/etc/init.d/ilogtaild status all 10
busy
```

输出信息描述

项目	描述	紧急度	解决方法
ok	当前状态正常。	无	无需处理。
busy	当前采集速度较高，Logtail状态正常。	无	无需处理。
many_log_files	当前正在采集的日志数较多。	低	检查配置中是否包含无需采集的文件。
process_block	当前日志解析出现阻塞。	低	检查日志产生速度是否过高，若一直出现，按需调整Logtail启动参数配置修改CPU使用上限或网络发送并发限制。
send_block	当前发送出现阻塞。	较高	检查日志产生速度是否过高以及网络状态是否正常，若一直出现，按需调整Logtail启动参数配置修改CPU使用上限或网络发送并发限制。
send_error	日志数据上传失败。	高	参考收集错误查询:SEND_DATA_FAIL_ALARM解决。

2.2. active命令

命令格式

```
/etc/init.d/ilogtaild status active [--logstore] index
```

```
/etc/init.d/ilogtaild status active --logfile index project-name logstore-name
```

说明：

- 命令active [--logstore] index用来查看当前活跃的Logstore，--logstore参数可以省略，命令含义不变。
- 命令active --logfile index project-name logstore-name用来查看某Project中Logstore下的所有活跃日志文件。
- active命令用来逐级查看活跃的日志文件。建议您先定位当前活跃的Logstore，再定向查询该Logstore下的活跃日志文件。

示例

```
/etc/init.d/ilogtaild status active 1
sls-zc-test : release-test
sls-zc-test : release-test-ant-rpc-3
sls-zc-test : release-test-same-regex-3

/etc/init.d/ilogtaild status active --logfile 1 sls-zc-test release-test
/disk2/test/normal/access.log
```

输出信息描述

- 执行命令active --logstore index，则以project-name : logstore-name形式输出当前所有活跃Logstore；若执行命令active --logfile index project-name logstore-name，则输出活跃日志文件的完整路径。
- 若Logstore或日志文件在查询窗口期内没有日志采集活动，则不会出现在active命令的输出信息中。

2.3. logstore命令

命令格式

```
/etc/init.d/ilogtaild status logstore [--format={line|json}] index project-name logstore-name
```

说明：

- 命令logstore指定以line或json形式输出指定Project和Logstore的采集状态。
- 如果不配置--format=参数，则默认选择--format=line，按照line形式输出回显信息。*注意*：--format参数需位于logstore之后。
- 若无该Logstore或该Logstore在当前查询窗口期没有日志采集活动，则line形式输出为空，json下为null。

示例

```
/etc/init.d/ilogtaild status logstore 1 sls-zc-test release-test-same
time_begin_readable : 17-08-29 10:56:11
time_end_readable : 17-08-29 11:06:11
time_begin : 1503975371
time_end : 1503975971
project : sls-zc-test
logstore : release-test-same
status : ok
config : ##1.0##sls-zc-test$same
read_bytes : 65033430
parse_success_lines : 230615
parse_fail_lines : 0
last_read_time : 1503975970
read_count : 687
avg_delay_bytes : 0
max_unsend_time : 0
min_unsend_time : 0
max_send_success_time : 1503975968
send_queue_size : 0
send_network_error_count : 0
send_network_quota_count : 0
send_network_discard_count : 0
send_success_count : 302
send_block_flag : false
sender_valid_flag : true


/etc/init.d/ilogtaild status logstore --format=json 1 sls-zc-test release-test-same
{
"avg_delay_bytes" : 0,
"config" : "##1.0##sls-zc-test$same",
"last_read_time" : 1503975970,
"logstore" : "release-test-same",
"max_send_success_time" : 1503975968,
"max_unsend_time" : 0,
"min_unsend_time" : 0,
"parse_fail_lines" : 0,
"parse_success_lines" : 230615,
"project" : "sls-zc-test",
"read_bytes" : 65033430,
"read_count" : 687,
"send_block_flag" : false,
"send_network_discard_count" : 0,
"send_network_error_count" : 0,
"send_network_quota_count" : 0,
"send_queue_size" : 0,
"send_success_count" : 302,
"sender_valid_flag" : true,
"status" : "ok",
"time_begin" : 1503975371,
"time_begin_readable" : "17-08-29 10:56:11",
"time_end" : 1503975971,
"time_end_readable" : "17-08-29 11:06:11"
}
```

输出信息描述

关键字	含义	单位
status	该Logstore整体状态。具体状态、含义以及修改方式见下表。	无
time_begin_readable	可读的开始时间。	无
time_end_readable	可读的截止时间。	无
time_begin	统计开始时间。	Unix时间戳，秒
time_end	统计结束时间。	Unix时间戳，秒
project	Project名。	无
logstore	Logstore名。	无
config	采集配置名（由##1.0## + project + $ + config组成的全局唯一配置名）。	无
read_bytes	窗口内读取日志数。	字节
parse_success_lines	窗口内日志解析成功的行数。	行
parse_fail_lines	窗口日志解析失败的行数。	行
last_read_time	窗口内最近的读取时间。	Unix时间戳，秒
read_count	窗口内日志读取次数。	次数
avg_delay_bytes	窗口内平均每次读取时当前偏移量与文件大小差值的平均值。	字节
max_unsend_time	窗口结束时发送队列中未发送数据包的最大时间，队列空时为0。	Unix时间戳，秒
min_unsend_time	窗口结束时发送队列中未发送数据包的最小时间，队列空时为0。	Unix时间戳，秒
max_send_success_time	窗口内发送成功数据的最大时间。	Unix时间戳，秒
send_queue_size	窗口结束时当前发送队列中未发送数据包数。	个
send_network_error_count	窗口内因网络错误导致发送失败的数据包个数。	个
send_network_quota_count	窗口内因quota超限导致发送失败的数据包个数。	个
send_network_discard_count	窗口内因数据异常或无权限导致丢弃数据包的个数。	个
send_success_count	窗口内发送成功的数据包个数。	个
send_block_flag	窗口结束时发送队列是否阻塞。	无

sender_valid_flag	窗口结束时该Logstore的发送标志位是否有效，true代表正常，false代表可能因为网络错误或quota错误而被禁用。	无

Logstore状态列表

状态	含义	处理方式
ok	状态正常	无需处理。
process_block	日志解析阻塞	检查日志产生速度是否过高，若一直出现，按需调整Logtail启动参数配置修改CPU使用上限或网络发送并发限制。
parse_fail	日志解析失败	检查日志格式与日志采集配置是否一致。
send_block	当前发送出现阻塞	检查日志产生速度是否过高以及网络状态是否正常，若一直出现，按需调整Logtail启动参数配置修改CPU使用上限或网络发送并发限制。
sender_invalid	日志数据发送异常	检查网络状态，若网络正常，参考收集错误查询:SEND_DATA_FAIL_ALARM解决。

2.4. logfile命令

命令格式

```
/etc/init.d/ilogtaild status logfile [--format={line|json}] index project-name logstore-name fileFullPath
```

说明：

- logfile命令指定以line或json形式输出指定日志文件的采集状态。
- 如果不配置--format=参数，则默认选择--format=line，按照line形式输出回显信息。
- 若无该logfile或该logfile在当前查询窗口期没有日志采集活动，则line形式输出为空，json下为null。

注意：

- --format参数需位于logfile之后。
- filefullpath必须是全路径名。

示例

```
/etc/init.d/ilogtaild status logfile 1 sls-zc-test release-test-same /disk2/test/normal/access.log
time_begin_readable : 17-08-29 11:16:11
time_end_readable : 17-08-29 11:26:11
time_begin : 1503976571
time_end : 1503977171
project : sls-zc-test
logstore : release-test-same
status : ok
config : ##1.0##sls-zc-test$same
file_path : /disk2/test/normal/access.log
file_dev : 64800
file_inode : 22544456
file_size_bytes : 17154060
file_offset_bytes : 17154060
read_bytes : 65033430
parse_success_lines : 230615
parse_fail_lines : 0
last_read_time : 1503977170
read_count : 667
avg_delay_bytes : 0

/etc/init.d/ilogtaild status logfile --format=json 1 sls-zc-test release-test-same /disk2/test/normal/access.log
{
"avg_delay_bytes" : 0,
"config" : "##1.0##sls-zc-test$same",
"file_dev" : 64800,
"file_inode" : 22544456,
"file_path" : "/disk2/test/normal/access.log",
"file_size_bytes" : 17154060,
"last_read_time" : 1503977170,
"logstore" : "release-test-same",
"parse_fail_lines" : 0,
"parse_success_lines" : 230615,
"project" : "sls-zc-test",
"read_bytes" : 65033430,
"read_count" : 667,
"read_offset_bytes" : 17154060,
"status" : "ok",
"time_begin" : 1503976571,
"time_begin_readable" : "17-08-29 11:16:11",
"time_end" : 1503977171,
"time_end_readable" : "17-08-29 11:26:11"
}
```

输出信息描述

关键字	含义	单位
status	该日志文件当前窗口期的采集状态，参见logstore命令的status。	无
time_begin_readable	可读的开始时间。	无
time_end_readable	可读的截止时间。	无

time_begin	统计开始时间。	Unix时间戳，秒
time_end	统计结束时间。	Unix时间戳，秒
project	Project名。	无
logstore	Logstore名。	无
file_path	该日志文件路径。	无
file_dev	该日志文件的device id。	无
file_inode	该日志文件的inode。	无
file_size_bytes	窗口内最近一次扫描到的该文件大小。	字节
read_offset_bytes	当前该文件解析偏移量。	字节
config	采集配置名（由##1.0## + project + $ + config组成的全局唯一配置名）。	无
read_bytes	窗口内读取日志数。	字节
parse_success_lines	窗口内日志解析成功的行数。	行
parse_fail_lines	窗口内日志解析失败的行数。	行
last_read_time	窗口内最近的读取时间。	Unix时间戳，秒
read_count	窗口内日志读取次数。	次数
avg_delay_bytes	窗口内平均每次读取时当前偏移量与文件大小差值的平均值。	字节

2.5. history命令

命令格式

```
/etc/init.d/ilogtaild status history beginIndex endIndex project-name logstore-name [ fileFullPath]
```

说明：

- history命令用来查询Logstore或日志文件一段时间内的采集状态。
- beginIndex、endIndex分别为代码查询窗口索引的起始值和终止值，需确保beginIndex <= endIndex。
- 若参数中不输入fileFullPath，则代码查询Logstore的采集信息；否则查询日志文件的采集信息。

示例

```
/etc/init.d/ilogtaild status history 1 3 sls-zc-test release-test-same /disk2/test/normal/access.log
```

```
begin_time status read parse_success parse_fail last_read_time read_count avg_delay device inode file_size
read_offset
17-08-29 11:26:11 ok 62.12MB 231000 0 17-08-29 11:36:11 671 0B 64800 22544459 18.22MB 18.22MB
17-08-29 11:16:11 ok 62.02MB 230615 0 17-08-29 11:26:10 667 0B 64800 22544456 16.36MB 16.36MB
17-08-29 11:06:11 ok 62.12MB 231000 0 17-08-29 11:16:11 687 0B 64800 22544452 14.46MB 14.46MB

$/etc/init.d/ilogtaild status history 2 5 sls-zc-test release-test-same
begin_time status read parse_success parse_fail last_read_time read_count avg_delay send_queue network_error
quota_error discard_error send_success send_block send_valid max_unsend min_unsend max_send_success
17-08-29 11:16:11 ok 62.02MB 230615 0 17-08-29 11:26:10 667 0B 0 0 0 0 300 false true 70-01-01 08:00:00 70-01-
01 08:00:00 17-08-29 11:26:08
17-08-29 11:06:11 ok 62.12MB 231000 0 17-08-29 11:16:11 687 0B 0 0 0 0 303 false true 70-01-01 08:00:00 70-01-
01 08:00:00 17-08-29 11:16:10
17-08-29 10:56:11 ok 62.02MB 230615 0 17-08-29 11:06:10 687 0B 0 0 0 0 302 false true 70-01-01 08:00:00 70-01-
01 08:00:00 17-08-29 11:06:08
17-08-29 10:46:11 ok 62.12MB 231000 0 17-08-29 10:56:11 692 0B 0 0 0 0 302 false true 70-01-01 08:00:00 70-01-
01 08:00:00 17-08-29 10:56:10
```

输出信息描述

- 该命令以列表形式输出Logstore或日志文件的历史采集信息，每个窗口期一行。
- 输出字段含义请参见logstore和logfile命令。

3. 命令返回值

正常返回值

所有命令输入有效情况下返回值为0（**包括无法查询到Logstore或日志文件**），例如：

```
/etc/init.d/ilogtaild status logfile --format=json 1 error-project error-logstore /no/this/file
null

echo $?
0

/etc/init.d/ilogtaild status all
ok
echo $?
0
```

异常返回值

返回值非0时，说明发生异常，请参考以下信息。

返回值	类型	输出	问题排查
10	无效命令或缺少参数	invalid param, use -h for help.	输入-h查看帮助。
1	查询超过1-60的时间窗口	invalid query interval	输出-h查看帮助。

1	无法查询到指定时间窗口	query fail, error: $(error)，具体参见errno释义	可能原因是logtail启动时间小于查询时间跨度，其他情况请提交工单处理。
1	查询窗口时间不匹配	no match time interval, please check logtail status	检查Logtail是否在运行，其他情况请提交工单处理。
1	查询窗口内没有数据	invalid profile, maybe logtail restart	检查Logtail是否在运行，其他情况请提交工单处理。

示例

```
/etc/init.d/ilogtaild status nothiscmd
invalid param, use -h for help.

echo $?
10

/etc/init.d/ilogtaild status/all 99
invalid query interval

echo $?
1
```

4. 功能使用场景示例

通过Logtail健康度查询可以获取Logtail当前整体状态；通过采集进度查询可以获取采集过程中的相关指标信息。用户可根据获取的这些信息实现自定义的日志采集监控。

4.1 监控Logtail运行状态

通过all命令实现Logtail运行状态监控。

实现方式：每隔一分钟定期查询Logtail当前状态，若连续5分钟状态处于process_block、send_block、send_error则触发报警。

具体报警持续时间以及监控的状态范围可根据具体场景中日志采集重要程度调整。

4.2. 监控日志采集进度

通过logstore命令实现具体日志库采集进度监控。

实现方式：定期每隔10分钟调用logstore命令获取该logstore的状态信息，若avg_delay_bytes超过1MB（1024*1024）或status不为ok则触发报警。

具体avg_delay_bytes报警阈值可根据日志采集流量调整。

4.3. 判断日志文件是否采集完毕

通过logfile命令判断日志文件是否采集完毕。

实现方式：日志文件已经停止写入后，定期每隔10分钟调用logfile命令获取该文件的状态信息，若该文件read_offset_bytes与file_size_bytes一致，则该日志文件已经采集完毕。

4.4. 日志采集问题排查

若发现某台服务器日志采集进度延迟，可用history命令查询该服务器上相关的采集信息。

send_block_flag为true，则说明日志采集进度延迟block在网络部分。

- 若send_network_quota_count大于0时，需要对Logstore的Shard进行分裂。
- 若send_network_error_count大于0时，需要检查网络连通性。
- 若无相关network error，则需要调整Logtail的发送并发以及流量限制。

发送部分相关参数正常但avg_delay_bytes较高。

- 可根据read_bytes计算出日志平均解析速度，判断日志产生流量是否异常。
- 可适当调整logtail的资源使用限制。

parse_fail_lines大于0。

检查日志采集解析配置是否能够匹配所有日志。

使用Logtail收集日志时，会遇到诸如正则解析失败，文件路径不正确，流量超过Shard服务能力等错误，日志服务提供诊断功能，支持自主诊断Logtail日志收集错误。

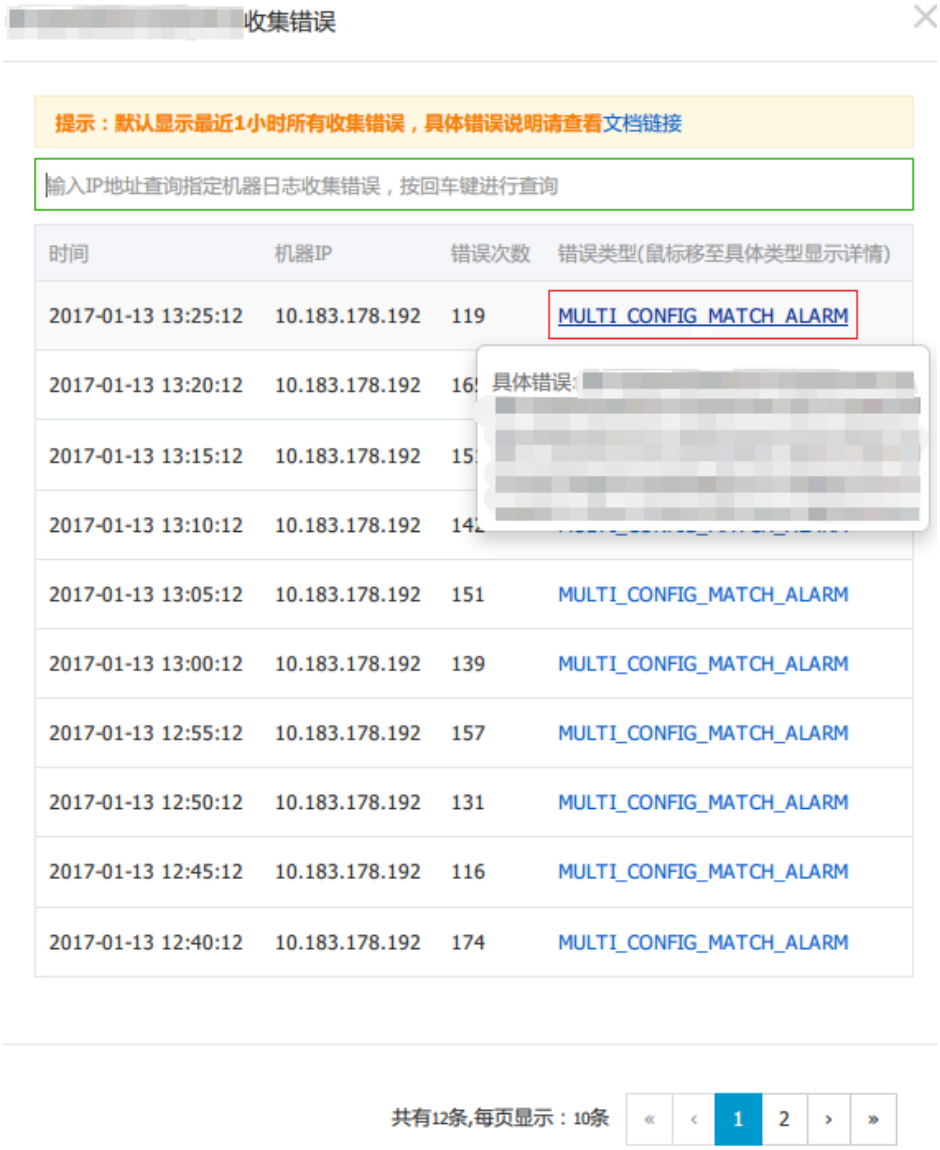
诊断步骤

1. 进入错误诊断页面

选择指定Project后，进入Logstore列表，在列表的**日志收集模式**列中单击**诊断**。

2. 查看日志收集错误

进入诊断页面后，即可查看指定Logstore对应的所有Logtail收集日志错误列表。



3. 查询指定机器收集错误

在错误查询页面中，可以通过在输入框输入指定机器IP地址，显示指定机器的所有收集错误。其中Logtail上报错误信息的时间间隔为5分钟。

处理问题完毕后，根据错误统计时间可以查看业务恢复正常后是否仍有报错。历史报错在过期前仍可显示，您可以忽略这部分报错，仅确认在问题处理完毕的时间点之后是否有新的错误即可。

诊断参考

以下为Logtail错误类型及处理方式，您可以在诊断收集错误时根据报错来查询处理。如有其它问题，请提工单。

错误类型	错误说明	处理方式
LOGFILE_PERMINSSION_ALARM	Logtail无权限读取指定文件。	检查服务器Logtail的启动账户，建议以root方式启动。
SPLIT_LOG_FAIL_ALARM	行首正则与日志行首匹配失败，无法对日志做分行。	检查行首正则正确性，如果是单行日志可以配置为.*。
MULTI_CONFIG_MATCH_ALARM	同一个文件，只能被一个Logtail的配置收集，不支持同时被多个logtail配置收集。	检查一个文件是否在多个配置中被收集，删除多余的配置。
REGEX_MATCH_ALARM	正则表达式解析模式下，日志内容和正表达式不匹配。	复制错误内容中的日志样例重新尝试匹配，并生成新的新的解析正则式。
PARSE_LOG_FAIL_ALARM	JSON、分隔符等解析模式下，由于日志格式不符合定义而解析失败。	请单击错误查看匹配失败的详细报错。
CATEGORY_CONFIG_ALARM	Logtail采集配置不合法。	常见的错误为正则表达式提取文件路径作为topic失败，其它错误请提工单解决。
LOGTAIL_CRASH_ALARM	Logtail因超过服务器资源使用上限而崩溃。	请参考Logtail启动参数配置修改CPU、内存使用上限，如有疑问请提工单。
INOTIFY_DIR_NUM_LIMIT_ALARM	Linux默认的inotify watcher上限是8192，Logtail限制使用最多3000个inotify watcher，每一个日志目录占用一个watcher。	检查是否有收集配置目录子目录较多，合理设置监控的根目录和目录最大监控深度。
DISCARD_DATA_ALARM	配置Logtail使用的CPU资源不够或网络发送流控导致。	请参考Logtail启动参数配置修改CPU使用上限或网络发送并发限制，如有疑问请提工单解决。
SEND_DATA_FAIL_ALARM	（1）主账号未创建任何AcessKey； （2）Logtail客户端机器与日志服务服务端无法连通或者网络链路质量较差； （3）服务端写入配额不足。	（1）主账号登录AcessKey控制台 创建AcessKey； （2）检查本地配置文件/usr/local/ilogtail/ilogtail_config.json，执行curl <服务地址>，查看是否有内容返回； （3）为Logstore增加Shard数目，以支持更大数据量的写入。

PARSE_TIME_FAIL_ALARM	Logtail根据时间解析表达式解析time字段失败。	请根据日志时间配置正确的时间解析表达式。
REGISTER_INOTIFY_FAIL_ALARM	Logtail为日志目录注册inotify watcher失败。	请检查目录是否存在以及目录权限设置。
SEND_QUOTA_EXCEED_ALARM	日志写入流量超出限制。	在控制台进行分区（Shard）扩容。
READ_LOG_DELAY_ALARM	日志采集进度落后于日志产生进度，一般是由于配置Logtail使用的CPU资源不够或是网络发送流控导致。	请参考Logtail启动参数配置修改CPU使用上限或网络发送并发限制，如有疑问请提工单。
LOGDIR_PERMINSSION_ALARM	没有日志监控目录读取权限。	请检查日志监控目录是否存在，若存在请检查目录权限设置。
ENCODING_CONVERT_ALARM	编码转换失败。	请检查日志编码格式配置是否与日志编码格式一致。
OUTDATED_LOG_ALAR	过期的日志，日志时间落后超过12小时。可能原因：日志解析进度落后超过12小时、用户自定义时间字段配置错误或日志记录程序时间输出异常。	首先查看是否存在READ_LOG_DELAY_ALARM，如存在按照READ_LOG_DELAY_ALARM处理方式解决；若不存在请检查时间字段配置；若时间字段配置正常请检查日志记录程序时间输出是否正常；如有疑问请提工单。

注：如需查看所有解析失败而丢弃的完整日志行，请登录机器查看/usr/local/ilogtail/ilogtail.LOG。

当您使用Logtail采集日志时，如果采集状态异常，请按以下步骤进行排查。

1 确认主账号是否已配置AccessKey

请在**主账号**的控制台右上角账户名称下拉列表中单击**accesskeys**，进入AK管理页面，查看是否已创建并启用了AK。

如果您的主账号没有配置AccessKey，Logtail不能正常运行。请参考5分钟快速入门中配置AccessKey步骤，正确配置AccessKey。

2 确认机器组内Logtail心跳是否正常

参考Logtail机器组，进入控制台单击查看机器组状态。如果心跳OK则直接进入下一步；如果心跳FAIL则需要进一步排查。

详细排查步骤请参考Logtail机器组心跳失败检查。

3 确认是否创建了采集配置并应用到机器组

确认客户端状态正常后，则需要确认以下配置。

3.1 是否已创建Logtail配置

参考Logtail配置。务必确认日志监控目录和日志文件名与机器上文件相匹配。其中，目录不支持模糊匹配，需填写绝对路径；日志文件名支持模糊匹配。

3.2 Logtail配置是否已应用到机器组

查看Logtail机器组，选择**管理配置**，确认目标配置是否已经应用到机器组。

4 检查采集错误

确认Logtail配置正常后，您需要确认日志文件是否实时产生新数据。Logtail只采集增量数据，存量文件如果没有更新则不会被读取。如日志有更新但未在日志服务中查询到，您可以通过以下方式诊断。

采集错误诊断

进行Logtail 收集错误查询，根据Logtail上报的错误类型处理。

查看Logtail日志

客户端日志会记录关键INFO以及所有WARNING、ERROR日志。如果想了解更为实时完整的错误，可以在以下路径查看客户端日志：

- Linux：/usr/local/ilogtail/ilogtail.LOG
- Windows x64 : C:\Program Files (x86)\Alibaba\Logtail\logtail_*.log
- Windows x32 : C:\Program Files\Alibaba\Logtail\logtail_*.log

用量超限

如果有大日志量或者大文件数据量的采集需求，可能需要修改Logtail的启动参数，以达到更高的日志采集吞吐量。参考Logtail 启动配置参数。

在完成以上三步检查后如问题仍未解决，请工单联系日志服务，并附上排查过程中发现的关键信息。

实时消费

数据收集至日志服务LogHub后，有三种方法可以消费日志：

方式	场景	实时性	存储时间
实时消费（LogHub）	流计算、实时计算等	实时(<10ms)	365天（如需更长时间请联系我们)
索引查询（LogSearch）	适合最近热数据的在线查询	实时（99.9% 1秒，最大3秒)	365天（如需更长时间请联系我们)
投递存储（LogShipper）	适合全量存储日志，进行离线分析	5~30分钟	依赖于存储系统

实时消费

消费过程

在写入日志后，最基本功能就是如何消费日志。消费日志与查询日志都意味着“读取”日志，两者区别见 **消费日志与查询日志的区别**。对于一个Shard 中日志，消费过程如下：

1. 根据时间、Begin、End等条件获得游标。
2. 通过游标、步长参数读取日志，同时返回下一个位置游标。
3. 不断移动游标进行日志消费。

消费方式

除最基本的API外，日志服务提供SDK、Storm Spout、Spark Client、Web Console等方式进行日志消费：

- 使用Spark Client消费日志: 参考E-MapReduce实现的Spark Streaming Client。
- 使用Storm Spout消费日志:日志服务提供一个LogHub Storm Spout来完成Storm和LogHub对接。
- 使用LogHub Consumer Library消费日志。Loghub Consumer Library是对LogHub消费者提供的高级模式。它提供了一个轻量级计算框架，解决多个消费者同时消费Logstore时自动分配Shard以及保序的问题，详情请参考Consumer Library。
- 使用SDK消费日志：日志服务提供多种语言（Java 和 Python）的 SDK，且这些语言的SDK都支持日志消费接口。关于SDK的更多信息请参考 **日志服务 SDK**。
- **访问日志统计镜像**：对常用日志进行实时分析 Docker 镜像，免费使用。
- 使用云产品消费日志：
    - **使用 CloudMonitor 云监控消费**：监控场景。
    - **使用 ARMS消费**：业务实时监控场景。
    - **使用 StreamCompute 消费日志**：自定义监控场景。
    - 使用 E-MapReduce 消费日志：参见Storm，Spark Streaming。

索引查询

- 使用日志服务控制台查询日志：参见 **查询日志**。

- 使用日志服务 SDK/API 查询日志：日志服务提供 REST 风格的 API，基于 HTTP 协议实现。日志服务的 API 同样提供全功能的日志查询接口。具体参考请见 日志服务 API。

投递存储

- 投递至OSS：长时间存储或使用 E-MapReduce 分析日志。
- 投递至表格存储（Table Store）：使用表格存储（NoSQL）存储日志。
- 投递至MaxCompute（即将上线）：使用MaxCompute分析日志。

其他

安全日志服务: 日志服务与安全云产品对接，可通过ISV消费云产品日志。

日志服务控制台提供专门的预览页面帮助您在浏览器内直接预览日志库中部分日志。

操作步骤

登录 日志服务管理控制台。

选择所需的项目，单击项目名称。

在 **LogStore列表** 页面，选择要查看的日志库并单击日志消费列下的 **预览**。

在日志查询页面，指定查询日志库的 shard 并限定日志时间区域，然后单击 **预览**。

日志预览页面原始数据列表两种方式向您展示指定时间区间开始的 10 个数据包的数据。

术语简介

LogHub Consumer Library中主要有4个概念，分别是Consumer Group、Consumer、Heartbeat 和 Checkpoint。

Consumer Group

是Logstore的子资源，拥有相同Consumer Group名字的消费者共同消费同一个Logstore的所有数据，这些消费者之间不会重复消费数据，一个 Logstore下面可以最多创建10个Consumer Group，不可以重名，同一个Logstore下面的Consumer Group之间消费数据不会互相影响。Consumer Group有两个很重要的属性：

```
{
"order":boolean,
"timeout": integer
}
```

- order 属性表示是否按照写入时间顺序消费Key相同的数据。
- timeout 表示Consumer Group中消费者的超时时间，单位是秒，当一个消费者汇报心跳的时间间隔超过Timeout，会被认为已经超时，服务端认为这个Consumer此时已经下线了。

Consumer

消费者，每个Consumer上会被分配若干个Shard，Consumer的职责就是要消费这些Shard上的数据，同一个Consumer Group中的Consumer必须不重名。

Heartbeat

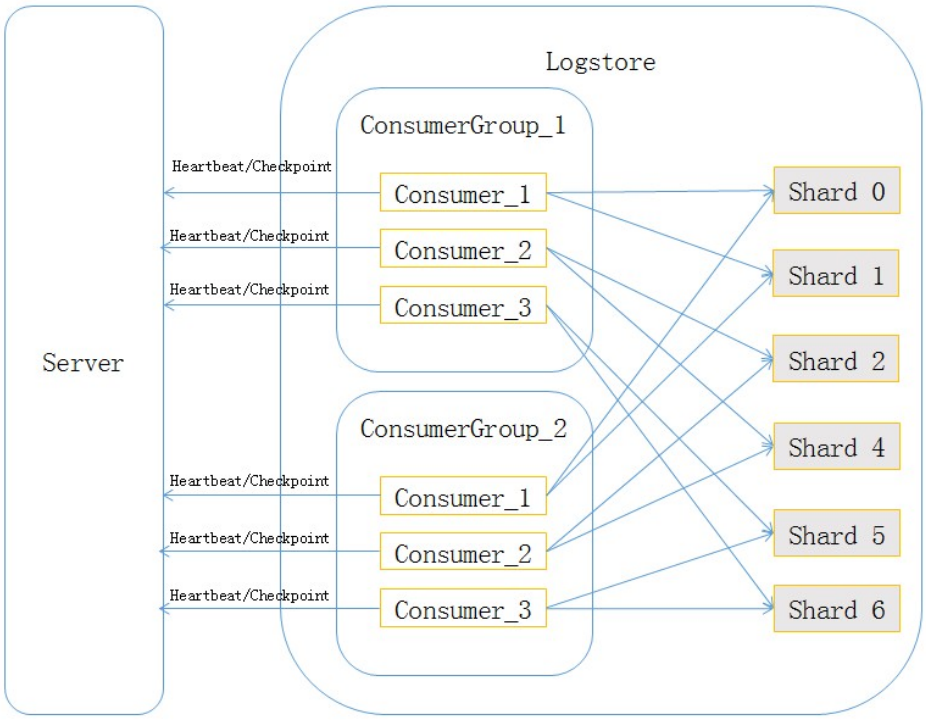
消费者心跳，consumer 需要定期向服务端汇报一个心跳包，用于表明自己还处于存活状态。

Checkpoint

消费者定期将分配给自己的Shard消费到的位置保存到服务端，这样当这个Shard被分配给其它消费者时，从服务端可以获取Shard的消费断点，接着从断点继续消费数据。

功能架构

Consumer Group、Consumer、Heartbeat 和Checkpoint之间的关系如下：



应用场景

LogHub Consumer Library是对LogHub消费者提供的高级模式，解决多个消费者同时消费Logstore时自动分配Shard的问题。

例如在Storm、Spark场景中多个消费者情况下，自动处理Shard的负载均衡、消费者Failover等逻辑。您只需专注在自己业务逻辑上，而无需关心Shard分配、CheckPoint、Failover等事宜。

例如，您需要通过Storm进行流计算，启动了A、B、C 3 个消费实例。在有10个Shard情况下，系统会自动为A、B、C分配3、3、4 个Shard进行消费。

- 当消费实例A宕机情况下，系统会把A未消费的3个Shard中数据自动均衡B、C上，当A恢复后，会重新均衡。
- 当添加实例D、E情况下，系统会自动进行均衡，每个实例消费2个Shard。
- 当Shard有Merge/Split等情况下，会根据最新的Shard信息，重新均衡。
- 当readonly状态的Shard消费完之后，剩余的Shard会重新做负载均衡。

以上整个过程不会产生数据丢失以及重复，您只需在代码中做三件事情：

1. 填写配置参数。
2. 写处理日志的代码。
3. 启动消费实例。

建议您使用LogHub Consumer Library进行数据消费，这样您只需要关心怎么处理数据，而不需要关注复杂的负载均衡、消费断点保存、按序消费、消费异常处理等问题。

在控制台查看消费进度

更多详细信息，参见 文档说明。

Spark Streaming，Storm Spout 默认已经使用Consumer Library消费LogHub数据。

如何使用LogHub Consumer Library

实现LogHub Consumer Library中的两个接口类：

- ILogHubProcessor：每个Shard对应一个实例，每个实例只消费特定Shard的数据。
- ILogHubProcessorFactory：负责生产实现ILogHubProcessor接口实例。

填写参数配置。

启动一个或多个Client Worker实例。

Maven 地址

```
<dependency>
<groupId>com.google.protobuf</groupId>
<artifactId>protobuf-java</artifactId>
<version>2.5.0</version>
</dependency>
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>loghub-client-lib</artifactId>
<version>0.6.8</version>
</dependency>
```

使用示例

main 函数

```
public static void main(String args[])
{
LogHubConfig config = new LogHubConfig(...);

ClientWorker worker = new ClientWorker(new SampleLogHubProcessorFactory(), config);

Thread thread = new Thread(worker);
//Thread运行之后，Client Worker会自动运行，ClientWorker扩展了Runnable接口。
thread.start();
```

```
Thread.sleep(60 * 60 * 1000);
//调用worker的Shutdown函数，退出消费实例，关联的线程也会自动停止。
worker.shutdown();
//ClientWorker运行过程中会生成多个异步的Task，Shutdown之后最好等待还在执行的Task安全退出，建议30s。
Thread.sleep(30 * 1000);
}
```

ILogHubProcessor、ILogHubProcessorFactory 实现 sample

各个Shard对应的消费实例类，实际开发过程中您主要需要关注数据消费逻辑，同一个ClientWorker实例是串行消费数据的，只会产生一个 ILogHubProcessor 实例， ClientWorker 退出的时候会调用ILogHubProcessor的shutdown函数。

```
public class SampleLogHubProcessor implements ILogHubProcessor
{
private int mShardId;
// 记录上次持久化 check point 的时间
private long mLastCheckTime = 0;

public void initialize(int shardId)
{
mShardId = shardId;
}

// 消费数据的主逻辑
public String process(List<LogGroupData> logGroups,
ILogHubCheckPointTracker checkPointTracker)
{
for(LogGroupData logGroup: logGroups){
FastLogGroup flg = logGroup.GetFastLogGroup();
System.out.println(String.format("\tcategory\t:\t%s\n\tsource\t:\t%s\n\ttopic\t:\t%s\n\tmachineUUID\t:\t%
s",
flg.getCategory(), flg.getSource(), flg.getTopic(), flg.getMachineUUID()));
System.out.println("Tags");
for (int tagIdx = 0; tagIdx < flg.getLogTagsCount(); ++tagIdx) {
FastLogTag logtag = flg.getLogTags(tagIdx);
System.out.println(String.format("\t%s\t:\t%s", logtag.getKey(), logtag.getValue()));
}
for (int lIdx = 0; lIdx < flg.getLogsCount(); ++lIdx) {
FastLog log = flg.getLogs(lIdx);
System.out.println("--------\nLog: " + lIdx + ", time: " + log.getTime() + ", GetContentCount: " +
log.getContentsCount());
for (int cIdx = 0; cIdx < log.getContentsCount(); ++cIdx) {
FastLogContent content = log.getContents(cIdx);
System.out.println(content.getKey() + "\t:\t" + content.getValue());
}
}
}
long curTime = System.currentTimeMillis();
// 每隔 60 秒，写一次 check point 到服务端，如果 60 秒内，worker crash，
// 新启动的 worker 会从上一个 checkpoint 其消费数据，有可能有重复数据
if (curTime - mLastCheckTime > 60 * 1000)
{
```

```
try
{
//参数 true 表示立即将 checkpoint 更新到服务端，为 false 会将 checkpoint 缓存在本地，默认隔 60s
//后台会将 checkpoint 刷新到服务端。
checkPointTracker.saveCheckPoint(true);
}
catch (LogHubCheckPointException e)
{
e.printStackTrace();
}
mLastCheckTime = curTime;
}
else
{
try
{
checkPointTracker.saveCheckPoint(false);
}
catch (LogHubCheckPointException e)
{
e.printStackTrace();
}
}
// 返回空表示正常处理数据， 如果需要回滚到上个 check point 的点进行重试的话，可以 return
checkPointTracker.getCheckpoint()
return null;
}
// 当 worker 退出的时候，会调用该函数，用户可以在此处做些清理工作。
public void shutdown(ILogHubCheckPointTracker checkPointTracker)
{
//将消费断点保存到服务端。
try {
checkPointTracker.saveCheckPoint(true);
} catch (LogHubCheckPointException e) {
e.printStackTrace();
}
}
}
```

- 生成 ILogHubProcessor 的工厂类 ：

```
public class SampleLogHubProcessorFactory implements ILogHubProcessorFactory
{
public ILogHubProcessor generatorProcessor()
{
// 生成一个消费实例
return new SampleLogHubProcessor();
}
}
```

配置说明

```
public class LogHubConfig
{
//worker 默认的拉取数据的时间间隔
public static final long DEFAULT_DATA_FETCH_INTERVAL_MS = 200;
//consumer group 的名字，不能为空，支持 [a-z][0-9] 和'_'，'-'，长度为3~63字符，只能以小写字母和数字开头结尾
private String mConsumerGroupName;
//consumer 的名字，必须确保同一个 consumer group 下面的各个 consumer 不重名
private String mWorkerInstanceName;
//loghub 数据接口地址
private String mLogHubEndPoint;
//项目名称
private String mProject;
//日志库名称
private String mLogStore;
//云账号的 access key id
private String mAccessId;
//云账号的 access key
private String mAccessKey;
//用于指出在服务端没有记录Shard的 checkpoint 的情况下应该从什么位置消费Shard，如果服务端保存了有效的
checkpoint 信息，那么这些取值不起任何作用， mCursorPosition 取值可以是 [BEGIN_CURSOR, END_CURSOR,
SPECIAL_TIMER_CURSOR]中的一个，BEGIN_CURSOR 表示从Shard中的第一条数据开始消费，END_CURSOR 表示从
Shard中的当前时刻的最后一条数据开始消费，SPECIAL_TIMER_CURSOR 和下面的 mLoghubCursorStartTime 配对使用
，表示从特定的时刻开始消费数据。
private LogHubCursorPosition mCursorPosition;
//当 mCursorPosition 取值为 SPECIAL_TIMER_CURSOR 时，指定消费时间，单位是秒。
private int mLoghubCursorStartTime = 0;
// 轮询获取 LogHub 数据的时间间隔，间隔越小，抓取越快，单位是毫秒，默认是
DEFAULT_DATA_FETCH_INTERVAL_MS，建议时间间隔 200ms 以上。
private long mDataFetchIntervalMillis;
// worker 向服务端汇报心跳的时间间隔，单位是毫秒，建议取值 10000ms。
private long mHeartBeatIntervalMillis;
//是否按序消费
private boolean mConsumeInOrder;
}
```

常见问题&注意事项

LogHubConfig 中 consumerGroupName 表一个消费组，consumerGroupName 相同的 consumer 分摊消费 logstore 中的Shard，同一个 consumerGroupName 中的 consumer，通过 workerInstance name 进行区分。

```
假设 logstore 中有Shard0~Shard3 这 4 个Shard。
有 3 个 worker，其 consumerGroupName 和 workerinstance name 分别是：
<consumer_group_name_1 , worker_A>，
<consumer_group_name_1 , worker_B>，
<consumer_group_name_2 , worker_C>
则，这些 worker 和 shard 的分配关系是：
<consumer_group_name_1 , worker_A>: shard_0, shard_1
<consumer_group_name_1 , worker_B>: shard_2, shard_3
<consumer_group_name_2 , worker_C>: shard_0, shard_1, shard_2, shard_3 # group name 不同的 worker 相
互不影响
```

确保实现的 ILogHubProcessor process() 接口每次都能顺利执行，并退出，这点很重要。

ILogHubCheckPointTracker 的 saveCheckPoint() 接口，无论传递的参数是 true，还是 false，都表示当前处理的数据已经完成，参数为 true，则立刻持久化至服务端，false 则每隔 60 秒同步一次到服务端。

如果 LogHubConfig 中配置的是子用户的 accessKeyId、accessKey，需要在 RAM 中进行以下授权，详细内容请参考 API 文档。

Action	Resource
log:GetCursorOrData	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/logstore/**${logstoreName}**
log:CreateConsumerGroup	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/logstore/**${logstoreName}**/consumergroup/*
log:ListConsumerGroup	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/logstore/**${logstoreName}**/consumergroup/*
log:ConsumerGroupUpdateCheckPoint	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/logstore/**${logstoreName}**/consumergroup/**${consumerGroupName}**
log:ConsumerGroupHeartBeat	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/logstore/**${logstoreName}**/consumergroup/**${consumerGroupName}**
log:GetConsumerGroupCheckPoint	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/logstore/**${logstoreName}**/consumergroup/**${consumerGroupName}**

协同消费组（ConsumerGroup）是实时消费数据高级模式，能够提供多个消费实例对日志库消费自动负载均衡。Spark Streaming、Storm 都以 ConsumerGroup 作为基础模式。

通过控制台查看消费进度

登录 日志服务管理控制台。

选择所需的项目，单击项目名称。

单击左侧导航栏中的 **LogHub - 实时消费** > **协同消费**。

在 **协同消费** 功能页面，选择日志库（Logstore）后即可查看目前是否启用协同消费功能。

选择指定的 ConsumerGroup 之后，单击 **消费状态**，即可查看当前每个 Shard 消费数据的进度。

如上图所示，页面上展示该日志库包含 5 个 Shard，对应 5 个消费者，其中每个消费者最近消费的数据时间如第二列显示。通过消费数据时间可以判断出目前数据处理是否能满足数据产生速度，如果已经严重落后于当前时间（即数据消费速率小于数据产生速率），可以考虑增加消费者数目。

通过 API/SDK 查看消费进度

以下代码行以Java SDK为例，向您演示如何通过 API 获得消费状态。

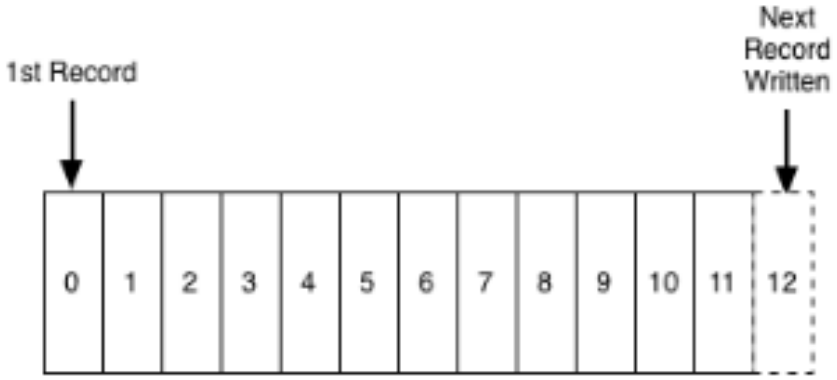
```
package test;

import java.util.ArrayList;
```

```
import com.aliyun.openservices.log.Client;
import com.aliyun.openservices.log.common.Consts.CursorMode;
import com.aliyun.openservices.log.common.ConsumerGroup;
import com.aliyun.openservices.log.common.ConsumerGroupShardCheckPoint;
import com.aliyun.openservices.log.exception.LogException;

public class ConsumerGroupTest {
static String endpoint = "";
static String project = "";
static String logstore = "";
static String accesskeyId = "";
static String accesskey = "";
public static void main(String[] args) throws LogException {
Client client = new Client(endpoint, accesskeyId, accesskey);
//获取这个 logstore 下的所有 consumer group，可能不存在，此时 consumerGroups 的长度是 0
ArrayList<ConsumerGroup> consumerGroups;
try{
consumerGroups = client.ListConsumerGroup(project, logstore).GetConsumerGroups();
}
catch(LogException e){
if(e.GetErrorCode() == "LogStoreNotExist")
System.out.println("this logstore does not have any consumer group");
else{
//internal server error branch
}
return;
}
for(ConsumerGroup c: consumerGroups){
//打印 consumer group 的属性，包括名称、心跳超时时间、是否按序消费
System.out.println("名称: " + c.getConsumerGroupName());
System.out.println("心跳超时时间: " + c.getTimeout());
System.out.println("按序消费: " + c.isInOrder());
for(ConsumerGroupShardCheckPoint cp: client.GetCheckPoint(project, logstore,
c.getConsumerGroupName()).GetCheckPoints()){
System.out.println("shard: " + cp.getShard());
//请格式化下，这个时间返回精确到毫秒的时间，长整型
System.out.println("最后一次消费数据的时间: " + cp.getUpdateTime());
System.out.println("消费者名称: " + cp.getConsumer());

String consumerPrg = "";
if(cp.getCheckPoint().isEmpty())
consumerPrg = "尚未开始消费";
else{
//unix 时间戳，单位是秒，输出的时候请注意格式化
try{
int prg = client.GetPrevCursorTime(project, logstore, cp.getShard(), cp.getCheckPoint()).GetCursorTime();
consumerPrg = "" + prg;
}
catch(LogException e){
if(e.GetErrorCode() == "InvalidCursor")
consumerPrg = "非法，前一次消费时刻已经超出了logstore中数据的生命周期";
else{
//internal server error
throw e;
}
}
```

```
}
System.out.println("消费进度: " + consumerPrg);

String endCursor = client.GetCursor(project, logstore, cp.getShard(), CursorMode.END).GetCursor();
int endPrg = 0;
try{
endPrg = client.GetPrevCursorTime(project, logstore, cp.getShard(), endCursor).GetCursorTime();
}
catch(LogException e){
//do nothing
}
//unix 时间戳，单位是秒，输出的时候请注意格式化
System.out.println("最后一条数据到达时刻: " + endPrg);
}
}
}

}
```

ConsumerGroup 是一个消费者组，包含多个Consumer，每个Consumer消费Logstore中的一部分Shard。

Shard的数据模型可以简单理解成一个队列，新写入的数据被加到队尾，队列中的每条数据都会对应一个数据写入时间，下图是Shard的数据模型。



协同消费延迟报警中的基本概念：

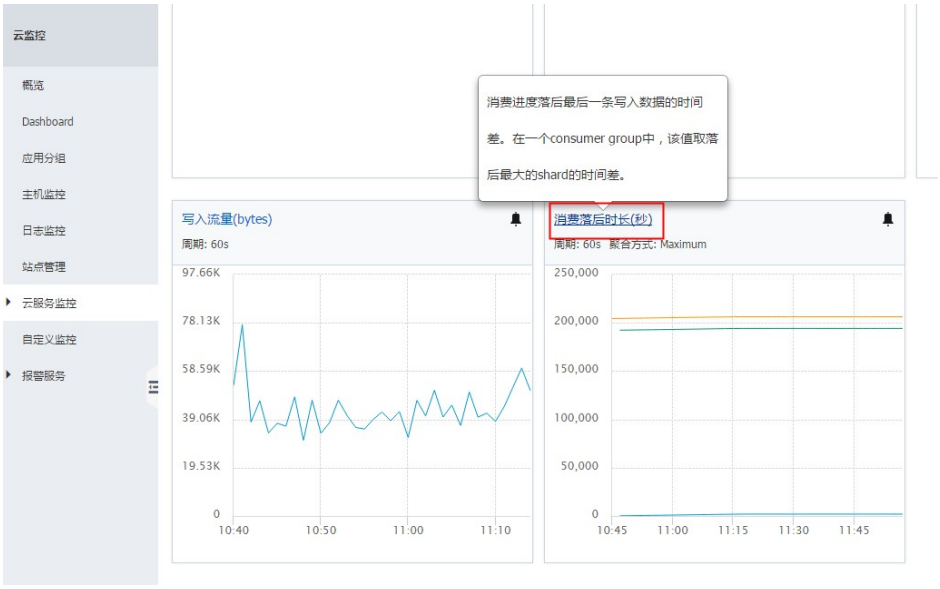
- **消费过程**：消费者从队头开始顺序读取数据的过程。
- **消费进度**：消费者当前读取的数据对应的写入时间。
- **消费落后时长**：当前消费进度和队列中最新的数据写入时间的差值，单位为秒。

ConsumerGroup的消费落后时长取其包含的所有Shard的消费落后时长的最大值，当超过用户预设阈值时，就认为消费落后太多，需要报警。

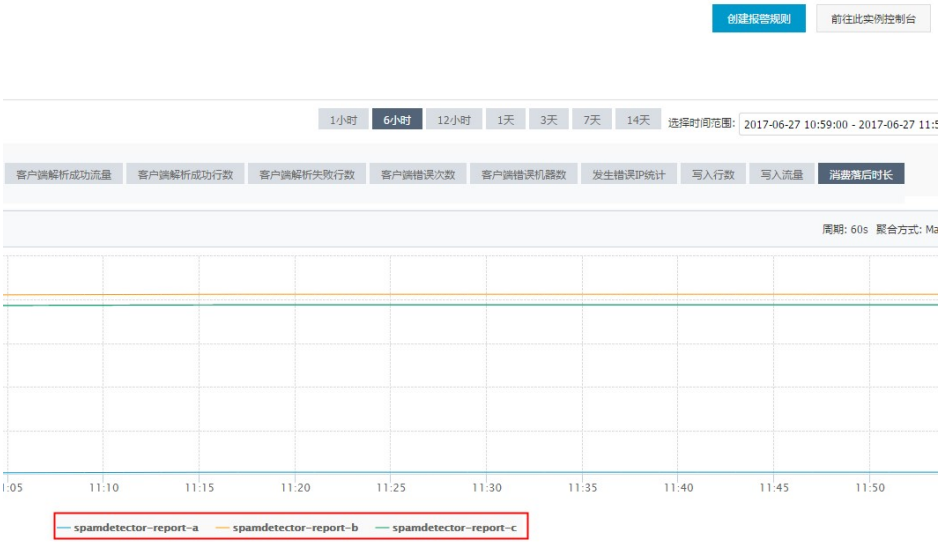
配置方法

登录 日志服务管理控制台，单击需要监控的 Logstore 的监控图标。

找到消费落后时长图表，单击进入云监控控制台。



该图展示了 Logstore 下所有 ConsumerGroup 的消费落后时长，单位为秒。红框中图例是所有的 ConsumerGroup，单击右上角 **创建报警规则** 进入规则创建页面。



创建针对 ConsumerGroup spamdetector-report-c 的报警规则，5min 内只要有一次延迟大于等于 600 秒就会报警。设置生效时间和报警通知联系人，保存规则。

上面的操作完成后便成功创建了报警规则。有关报警规则配置的任何问题，请提工单到云监控。

Flink log connector

介绍

Flink log connector是阿里云日志服务提供的，用于对接Flink的工具，包括两部分，消费者(Consumer)和生产者(Producer)。

消费者用于从日志服务中读取数据，支持exactly once语义，支持shard负载均衡。生产者用于将数据写入日志服务，使用connector时，需要在项目中添加maven依赖：

```
<dependency>
<groupId>org.apache.flink</groupId>
<artifactId>flink-streaming-java_2.11</artifactId>
<version>1.3.2</version>
</dependency>
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>flink-log-connector</artifactId>
<version>0.1.2</version>
```

```
</dependency>
<dependency>
<groupId>com.google.protobuf</groupId>
<artifactId>protobuf-java</artifactId>
<version>2.5.0</version>
</dependency>
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>aliyun-log</artifactId>
<version>0.6.10</version>
</dependency>
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>log-loghub-producer</artifactId>
<version>0.1.8</version>
</dependency>
```

用法

1. 请参考日志服务文档，正确创建Logstore。
2. 如果使用子账号访问，请确认正确设置了Logstore的RAM策略。参考授权RAM子用户访问日志服务资源。

1. Log Consumer

在Connector中， 类FlinkLogConsumer提供了订阅日志服务中某一个Logstore的能力，实现了exactly once语义，在使用时，用户无需关心Logstore中shard数量的变化，consumer会自动感知。

Flink中每一个子任务负责消费Logstore中部分shard，如果Logstore中shard发生split或者merge，子任务消费的shard也会随之改变。

1.1 配置启动参数

```
Properties configProps = new Properties();
// 设置访问日志服务的域名
configProps.put(ConfigConstants.LOG_ENDPOINT， "cn-hangzhou.log.aliyuncs.com");
// 设置访问ak
configProps.put(ConfigConstants.LOG_ACCESSSKEYID， "");
configProps.put(ConfigConstants.LOG_ACCESSKEY， "");
// 设置日志服务的project
configProps.put(ConfigConstants.LOG_PROJECT， "ali-cn-hangzhou-sls-admin");
// 设置日志服务的Logstore
configProps.put(ConfigConstants.LOG_LOGSTORE， "sls_consumergroup_log");
// 设置消费日志服务起始位置
configProps.put(ConfigConstants.LOG_CONSUMER_BEGIN_POSITION， Consts.LOG_END_CURSOR);
// 设置日志服务的消息反序列化方法
RawLogGroupListDeserializer deserializer = new RawLogGroupListDeserializer();
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
DataStream<RawLogGroupList> logTestStream = env.addSource(
new FlinkLogConsumer<RawLogGroupList>(deserializer， configProps));
```

上面是一个简单的消费示例，我们使用java.util.Properties作为配置工具，所有Consumer的配置都可以在ConfigConstants中找到。

> 注意，Flink stream的子任务数量和日志服务Logstore中的shard数量是独立的，如果shard数量多于子任务数量，每个子任务不重复的消费多个shard，如果少于子任务数量，那么部分子任务就会空闲，等到新的shard产生。

1.2 设置消费起始位置

Flink log consumer支持设置shard的消费起始位置，通过设置属性ConfigConstants.LOG_CONSUMER_BEGIN_POSITION，就可以定制消费从shard的头尾或者某个特定时间开始消费，另外，connector也支持从某个具体的ConsumerGroup中恢复消费。具体取值如下：

- Consts.LOG_BEGIN_CURSOR： 表示从shard的头开始消费，也就是从shard中最旧的数据开始消费。
- Consts.LOG_END_CURSOR： 表示从shard的尾开始，也就是从shard中最新的数据开始消费。
- Consts.LOG_FROM_CHECKPOINT：表示从某个特定的ConsumerGroup中保存的Checkpoint开始消费，通过ConfigConstants.LOG_CONSUMERGROUP指定具体的ConsumerGroup。
- UnixTimestamp： 一个整型数值的字符串，用1970-01-01到现在的秒数表示， 含义是消费shard中这个时间点之后的数据。

三种取值举例如下：

```
configProps.put(ConfigConstants.LOG_CONSUMER_BEGIN_POSITION， Consts.LOG_BEGIN_CURSOR);
configProps.put(ConfigConstants.LOG_CONSUMER_BEGIN_POSITION， Consts.LOG_END_CURSOR);
configProps.put(ConfigConstants.LOG_CONSUMER_BEGIN_POSITION， "1512439000");
configProps.put(ConfigConstants.LOG_CONSUMER_BEGIN_POSITION, Consts.LOG_FROM_CHECKPOINT);
```

> 注意，如果在启动Flink任务时，设置了从Flink自身的StateBackend中恢复，那么connector会忽略上面的配置，使用StateBackend中保存的Checkpoint。

1.3 监控：消费进度(可选)

Flink log consumer支持设置消费进度监控，所谓消费进度就是获取每一个shard实时的消费位置，这个位置使用时间戳表示，详细概念可以参考文档消费组-查看状态，消费组-监控报警。

```
configProps.put(ConfigConstants.LOG_CONSUMERGROUP，"your consumer group name”);
```

> 注意上面代码是可选的，如果设置了，consumer会首先创建consumerGroup，如果已经存在，则什么都不做，consumer中的snapshot会自动同步到日志服务的consumerGroup中，用户可以在日志服务的控制台查看consumer的消费进度。

1.4 容灾和exactly once语义支持

当打开Flink的checkpointing功能时，Flink log consumer会周期性的将每个shard的消费进度保存起来，当作业失败时，flink会恢复log consumer，并从保存的最新的checkpoint开始消费。

写checkpoint的周期定义了当发生失败时，最多多少的数据会被回溯，也就是重新消费，使用代码如下：

```
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
// 开启flink exactly once语义
env.getCheckpointConfig().setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE);
// 每5s保存一次checkpoint
env.enableCheckpointing(5000);
```

更多Flink checkpoint的细节请参考Flink官方文档Checkpoints。

1.5 补充材料：关联 API与权限设置

Flink log consumer 会用到的阿里云日志服务接口如下：

GetCursorOrData

用于从shard中拉数据， 注意频繁的调用该接口可能会导致数据超过日志服务的shard quota， 可以通过ConfigConstants.LOG_FETCH_DATA_INTERVAL_MILLIS和ConfigConstants.LOG_MAX_NUMBER_PER_FETCH 控制接口调用的时间间隔和每次调用拉取的日志数量，shard的quota参考文章shard简介.

```
  configProps.put(ConfigConstants.LOG_FETCH_DATA_INTERVAL_MILLIS， "100");
configProps.put(ConfigConstants.LOG_MAX_NUMBER_PER_FETCH， "100");
```

ListShards

用于获取Logstore中所有的shard列表，获取shard状态等.如果您的shard经常发生分裂合并，可以通过调整接口的调用周期来及时发现shard的变化。

```
   // 设置每30s调用一次ListShards
configProps.put(ConfigConstants.LOG_SHARDS_DISCOVERY_INTERVAL_MILLIS， "30000");
```

CreateConsumerGroup

该接口调用只有当设置消费进度监控时才会发生，功能是创建consumerGroup，用于同步checkpoint。

ConsumerGroupUpdateCheckPoint

该接口用户将flink的snapshot同步到日志服务的consumerGroup中。

子用户使用Flink log consumer需要授权如下几个RAM Policy：

接口	资源
log:GetCursorOrData	acs:log:${regionName}:${projectOwnerAliUid}:project/${projectName}/logstore/${logstoreName}
log:ListShards	acs:log:${regionName}:${projectOwnerAliUid}:project/${projectName}/logstore/${logstoreName}
log:CreateConsumerGroup	acs:log:${regionName}:${projectOwnerAliUid}:project/${projectName}/logstore/${logstoreName}/consumergroup/*
log:ConsumerGroupUpdateCheckPoint	acs:log:${regionName}:${projectOwnerAliUid}:project/${projectName}/logstore/${logstoreName}/consumergroup/${consumerGroupName}

2. Log Producer

FlinkLogProducer 用于将数据写到阿里云日志服务中。

> 注意producer只支持Flink at-least-once语义，这就意味着在发生作业失败的情况下，写入日志服务中的数据有可能会重复，但是绝对不会丢失。

用法示例如下，我们将模拟产生的字符串写入日志服务：

```
// 将数据序列化成日志服务的数据格式
class SimpleLogSerializer implements LogSerializationSchema<String> {

public RawLogGroup serialize(String element) {
RawLogGroup rlg = new RawLogGroup();
RawLog rl = new RawLog();
rl.setTime((int)(System.currentTimeMillis() / 1000));
rl.addContent("message" ,  element);
rlg.addLog(rl);
return rlg;
}
}
public class ProducerSample {
public static String sEndpoint = "cn-hangzhou.log.aliyuncs.com";
public static String sAccessKeyId = "";
public static String sAccessKey = "";
public static String sProject = "ali-cn-hangzhou-sls-admin";
public static String sLogstore = "test-flink-producer";
private static final Logger LOG = LoggerFactory.getLogger(ConsumerSample.class);


public static void main(String[] args) throws Exception {
```

```
final ParameterTool params = ParameterTool.fromArgs(args);
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.getConfig().setGlobalJobParameters(params);
env.setParallelism(3);

DataStream<String> simpleStringStream = env.addSource(new EventsGenerator());

Properties configProps = new Properties();
// 设置访问日志服务的域名
configProps.put(ConfigConstants.LOG_ENDPOINT， sEndpoint);
// 设置访问日志服务的ak
configProps.put(ConfigConstants.LOG_ACCESSSKEYID， sAccessKeyId);
configProps.put(ConfigConstants.LOG_ACCESSKEY， sAccessKey);
// 设置日志写入的日志服务project
configProps.put(ConfigConstants.LOG_PROJECT， sProject);
// 设置日志写入的日志服务Logstore
configProps.put(ConfigConstants.LOG_LOGSTORE， sLogstore);

FlinkLogProducer<String> logProducer = new FlinkLogProducer<String>(new SimpleLogSerializer()， configProps);

simpleStringStream.addSink(logProducer);

env.execute("flink log producer");
}
// 模拟产生日志
public static class EventsGenerator implements SourceFunction<String> {
private boolean running = true;

@Override
public void run(SourceContext<String> ctx) throws Exception {
long seq = 0;
while (running) {
Thread.sleep(10);
ctx.collect((seq++) + "-" + RandomStringUtils.randomAlphabetic(12));
}
}

@Override
public void cancel() {
running = false;
}
}
}
```

2.1 初始化

Producer初始化主要需要做两件事情：

- 初始化配置参数Properties， 这一步和Consumer类似， Producer有一些定制的参数，一般情况下使用默认值即可，特殊场景可以考虑定制：

```
 // 用于发送数据的io线程的数量，默认是8
ConfigConstants.LOG_SENDER_IO_THREAD_COUNT
// 该值定义日志数据被缓存发送的时间，默认是3000
```

```
ConfigConstants.LOG_PACKAGE_TIMEOUT_MILLIS
// 缓存发送的包中日志的数量，默认是4096
ConfigConstants.LOG_LOGS_COUNT_PER_PACKAGE
// 缓存发送的包的大小，默认是3Mb
ConfigConstants.LOG_LOGS_BYTES_PER_PACKAGE
// 作业可以使用的内存总的大小，默认是100Mb
ConfigConstants.LOG_MEM_POOL_BYTES
```

上述参数不是必选参数，用户可以不设置，直接使用默认值。

重载LogSerializationSchema，定义将数据序列化成RawLogGroup的方法。

RawLogGroup是log的集合，每个字段的含义可以参考文档日志数据模型。

如果用户需要使用日志服务的shardHashKey功能，指定数据写到某一个shard中，可以使用LogPartitioner产生数据的hashKey，用法例子如下：

```
FlinkLogProducer<String> logProducer = new FlinkLogProducer<String>(new SimpleLogSerializer()， configProps);
logProducer.setCustomPartitioner(new LogPartitioner<String>() {
// 生成32位hash值
public String getHashKey(String element) {
try {
MessageDigest md = MessageDigest.getInstance("MD5");
md.update(element.getBytes());
String hash = new BigInteger(1， md.digest()).toString(16);
while(hash.length() < 32) hash = "0" + hash;
return hash;
} catch (NoSuchAlgorithmException e) {
}
return "00000000000000000000000000000000000000000000000000000000000000000";
}
});
```

注意LogPartitioner是可选的，不设置情况下， 数据会随机写入某一个shard。

2.2 权限设置：RAM Policy

Producer依赖日志服务的API写数据，如下：

- log:PostLogStoreLogs
- log:ListShards

当RAM子用户使用Producer时，需要对上述两个API进行授权：

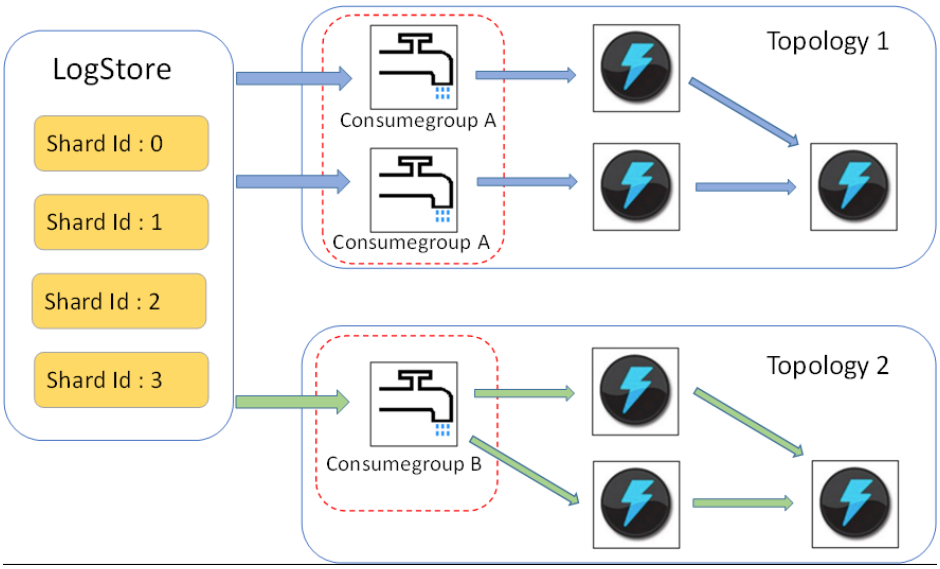
接口	资源
log:PostLogStoreLogs	acs:log:${regionName}:${projectOwnerAliUid}:project/${projectName}/logstore/${logstoreName}
log:ListShards	acs:log:${regionName}:${projectOwnerAliUid}:project/${projectName}/logstore/${logstoreN

	ame}

日志服务的LogHub提供了高效、可靠的日志通道功能，您可以通过Logtail、SDK等多种方式来实时收集日志数据。收集日志之后，可以通过Spark Stream、Storm 等各实时系统来消费写入到LogHub中的数据。

为了降低Storm用户消费LogHub的代价，日志服务提供了LogHub Storm Spout来实时读取LogHub的数据。

基本结构和流程



- 上图中红色虚线框中就是LogHub Storm Spout，每个Storm Topology会有一组Spout，同组内的Spout共同负责读取Logstore中全部数据。不同Topology中的Spout相互不干扰。
- 每个Topology需要选择唯一的LogHub Consume Group名字来相互标识，同一 Topology内的Spout通过 LogHub client lib 来完成负载均衡和自动failover。
- Spout从LogHub中实时读取数据之后，发送至Topology中的Bolt节点，定期保存消费完成位置作为checkpoint到LogHub服务端。

使用限制

- 为了防止滥用，每个Logstore最多支持 5 个Consumer Group，对于不再使用的 Consumer Group，可以使用Java SDK中的DeleteConsumerGroup接口进行删除。
- Spout的个数最好和Shard个数相同，否则可能会导致单个Spout处理数据量过多而处理不过来。
- 如果单个Shard 的数据量太大，超过一个Spout处理极限，则可以使用Shard split接口分裂Shard，来降低每个Shard的数据量。
- 在Loghub Spout中，强制依赖Storm的ACK机制，用于确认Spout将消息正确发送至Bolt，所以在Bolt中一定要调用ACK进行确认。

使用样例

Spout 使用示例（用于构建 Topology）

```
public static void main( String[] args )
{
String mode = "Local"; // 使用本地测试模式

String conumser_group_name = ""; // 每个Topology 需要设定唯一的 consumer group 名字，不能为空，支持 [a-z][0-9]
和 '_'，'-'，长度在 [3-63] 字符，只能以小写字母和数字开头结尾
String project = ""; // 日志服务的Project
String logstore = ""; // 日志服务的Logstore
String endpoint = ""; // 日志服务访问域名
String access_id = ""; // 用户 ak 信息
String access_key = "";

// 构建一个 Loghub Storm Spout 需要使用的配置
LogHubSpoutConfig config = new LogHubSpoutConfig(conumser_group_name,
endpoint, project, logstore, access_id,
access_key, LogHubCursorPosition.END_CURSOR);

TopologyBuilder builder = new TopologyBuilder();

// 构建 loghub storm spout
LogHubSpout spout = new LogHubSpout(config);

// 在实际场景中，Spout的个数可以和Logstore Shard 个数相同
builder.setSpout("spout", spout, 1);
builder.setBolt("exclaim", new SampleBolt()).shuffleGrouping("spout");


Config conf = new Config();
conf.setDebug(false);
conf.setMaxSpoutPending(1);

// 如果使用Kryo进行数据的序列化和反序列化，则需要显示设置 LogGroupData 的序列化方法
LogGroupDataSerializSerializer
Config.registerSerialization(conf, LogGroupData.class, LogGroupDataSerializSerializer.class);

if (mode.equals("Local")) {
logger.info("Local mode...");

LocalCluster cluster = new LocalCluster();

cluster.submitTopology("test-jstorm-spout", conf, builder.createTopology());

try {
Thread.sleep(6000 * 1000); //waiting for several minutes
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
```

```
cluster.killTopology("test-jstorm-spout");
cluster.shutdown();

} else if (mode.equals("Remote")) {

logger.info("Remote mode...");

conf.setNumWorkers(2);
try {
StormSubmitter.submitTopology("stt-jstorm-spout-4", conf, builder.createTopology());
} catch (AlreadyAliveException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (InvalidTopologyException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
} else {
logger.error("invalid mode: " + mode);
}

}

}
```

Bolt 代码样例

以下为消费数据的Bolt代码样例，只打印每条日志的内容。

```
public class SampleBolt extends BaseRichBolt {
private static final long serialVersionUID = 4752656887774402264L;

private static final Logger logger = Logger.getLogger(BaseBasicBolt.class);

private OutputCollector mCollector;

@Override
public void prepare(@SuppressWarnings("rawtypes") Map stormConf, TopologyContext context,
OutputCollector collector) {

mCollector = collector;
}

@Override
public void execute(Tuple tuple) {

String shardId = (String) tuple
.getValueByField(LogHubSpout.FIELD_SHARD_ID);

@SuppressWarnings("unchecked")
List<LogGroupData> logGroupDatas = (ArrayList<LogGroupData>)
tuple.getValueByField(LogHubSpout.FIELD_LOGGROUPS);
```

```
for (LogGroupData groupData : logGroupDatas) {
// 每个 LogGroup 由一条或多条日志组成
LogGroup logGroup = groupData.GetLogGroup();
for (Log log : logGroup.getLogsList()) {
StringBuilder sb = new StringBuilder();
// 每条日志，有一个时间字段， 以及多个 Key:Value 对，
int log_time = log.getTime();
sb.append("LogTime:").append(log_time);
for (Content content : log.getContentsList()) {
sb.append("\t").append(content.getKey()).append(":")
.append(content.getValue());
}
logger.info(sb.toString());
}
}
// 在LogHub spout中，强制依赖Storm的ACK机制，用于确认Spout将消息正确
// 发送至Bolt，所以在Bolt中一定要调用ACK
mCollector.ack(tuple);
}


@Override
public void declareOutputFields(OutputFieldsDeclarer declarer) {
//do nothing
}
}
```

Maven

storm 1.0 之前版本（如 0.9.6），请使用：

```
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>loghub-storm-spout</artifactId>
<version>0.6.5</version>
</dependency>
```

storm 1.0 版本及以后，请使用：

```
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>loghub-storm-1.0-spout</artifactId>
<version>0.1.2</version>
</dependency>
```

E-MapReduce 实现了一套通用的Spark Streaming实时消费LogHub的接口，参见GitHub。

云监控(Cloud Monitor)可以直接消费LogHub下LogStore数据提供监控功能，例如：

- 对日志中的关键字报警
- 统计单位时间内的QPS、RT
- 统计单位时间内的PV、 UV等

操作步骤参见：

- 日志监控概览
- 云监控管理日志监控

日志投递与ETL

将日志源接入日志服务后，日志服务开始实时采集日志，并提供控制台或SDK/API方式的日志消费和日志投递功能。日志服务采集到LogHub的日志数据，可以实时投递至OSS、Table Store、MaxCompute（即将上线）等存储类阿里云产品，您只需要在控制台配置即可完成，同时，LogShipper提供完整状态API与自动重试功能。

应用场景

- 对接数据仓库

日志数据来源

日志服务的日志投递功能所投递的日志数据来源于日志服务采集到LogHub的日志。这部分日志生成之后，被日志服务实时采集并投递至其他云产品中进行存储与分析。

日志投递目标

- OSS（大规模对象存储）：
  - 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
- MaxCompute (大数据计算服务)：
  - 操作步骤

日志服务可以把Logstore中的数据自动归档到OSS，以发挥日志更多的效用。

- OSS 数据支持自由设置生命周期，可以对日志进行长期存储。

- 可以通过自建程序和更多系统（如E-MapReduce）消费OSS数据。

功能优势

通过日志服务投递日志数据到OSS具有如下优势：

- 操作简单。仅需在控制台上做简单配置即可将日志服务Logstore的数据同步到OSS。
- 效率提升。日志服务的日志收集过程已经完成不同机器上的日志集中化，无需重复在不同机器上收集日志导入OSS。
- 便于管理。投递日志到OSS可以充分复用日志服务内的日志分类管理工作。用户可让日志服务不同项目（Project）、不同类型（Logstore）的日志自动投递到不同的OSS Bucket目录，方便管理OSS数据。

应用场景

例如两个阿里云用户主账号A、B：

主账号A在日志服务深圳Region开通Project-a，并创建了一个Logstore存放nginx访问日志。

主账号B在OSS深圳Region创建了Bucket-b。

注意：

- A 和 B可以是相同账号，这种情况下RAM授权最为便捷。
- 日志服务Project和OSS的Bucket必须位于相同Region，不支持跨Region投递数据。

用户希望将日志服务Project-a的nginx访问日志归档到Bucket-b的 prefix-b 目录下。

使用日志服务投递OSS功能，需要完成RAM授权和投递规则配置两个步骤。

操作步骤

步骤 1 访问控制（RAM）授权

快捷授权

主账号 B 登录阿里云控制台，免费开通 访问控制 RAM。

单击 权限控制 RAM 快捷授权页，确认授予写 OSS 所有 Bucket 权限，日志服务将替 A 扮演角色写 B 的 OSS Bucket。

查看、修改角色

主账号 B 登录阿里云控制台 访问控制 RAM，进入 **角色管理** 并查看角色（快捷授权默认创建的角色是

AliyunLogDefaultRole）。

若 A、B不相同，请参考RAM进阶（授权Role管理）修改Role。若 A、B相同，则快捷授权创建的角色直接可用。

1. 记录角色的 Arn（如acs:ram::45643:role/aliyunlogdefaultrole），该 Arn 需要提供给主账号 A 创建 OSS 投递规则时使用。
2. 角色 Arn 可以在角色管理/管理下，ARN 栏获得。

快捷授权默认会授予写 B 的所有 OSS Bucket 权限，如需更精细化的权限控制，请参考RAM进阶（授权Policy管理）修改 Policy。

使用子账号创建投递规则授权

在 B 创建角色、授权完成之后，主账号 A 有权限使用主账号 B 创建的角色写数据到 OSS Bucket，但这仅限于主账号 A 本身创建投递规则。

如果主账号 A 的子账号 a_1 希望使用该角色创建投递规则，请参考RAM进阶（主子账号Role授权）进行PassRole授权。

步骤 2 日志服务用户配置OSS投递规则

登录 日志服务管理控制台。

选择所需的项目，单击项目名称。

选择所需的日志库，单击日志投递列下的**OSS**。

单击 **开启投递**，设置 OSS 投递配置并单击 **确认**。

请参考下表设置OSS投递配置。

配置项	说明	取值范围
OSS投递名称	您所创建的投递的名称。	只能包含小写字母，数字，连字符（-）和下划线（_），必须以小写字母和数字开头和结尾，且名称长度为3~63字节。
OSS Bucket	OSS Bucket 名称。	必须是已存在的Bucket名称，且需要保证 OSS 的 Bucket 与日志服务 Project 位于相同 Region。
OSS Prefix	OSS前缀，从日志服务同步到 OSS 的数据将存放到 Bucket 的该目录下。	必须是已存在的OSS Prefix名称。

分区格式	将投递任务创建时间使用%Y，%m，%d，%H，%M等格式化生成分区字符串，以此来定义写到OSS的Object文件所在的目录层次结构，其中斜线/表示一级OSS目录。如下表举例说明OSS Prefix和分区格式如何定义OSS目标文件路径。	格式化参考strptime API。
RAM角色	RAM角色ARN及角色名称，用于访问权限控制，OSS Bucket 拥有者创建角色的标识，如何获得ARN请参见步骤1中的查看、修改角色。	如acs:ram::45643:role/aliyunlogdefaultrole。
投递大小	自动控制投递任务创建间隔并设置 OSS 的一个 Object 大小（以未压缩计算）上限。	取值范围为5~256，单位为MB。
存储格式	日志数据投递OSS的存储格式。	支持JSON/Parquet/CSV三种格式，配置细节请点击查看：JSON、Parquet、CSV。
是否压缩	OSS 数据存储的压缩方式。	- none：表示原始数据不压缩。 - snappy：表示使用 snappy 算法对数据做压缩，可以减少 OSS Bucket 存储空间使用量。
投递时间	投递任务的间隔时长。	取值范围为300~900，默认值300。单位为秒。

注意：日志服务在后端并发执行数据投递，如果您的数据量较大，可能会多个投递线程进行服务。每一个投递线程都会根据大小、时间共同决定任务生成的频率，当任一条件满足时，投递线程即会创建任务。

分区格式

每个投递任务会写入OSS一个文件，路径格式是oss:// OSS-BUCKET/OSS-PREFIX/PARTITION-FROMAT_RANDOM-ID。PARTITION-FROMAT根据投递任务创建时间格式化得到，以创建时间为2017/01/20 19:50:43的投递任务为例，说明分区格式的用法：

OSS Bucket	OSS Prefix	分区格式	OSS文件路径
test-bucket	test-table	%Y/%m/%d/%H/%M	oss://test-bucket/test-table/2017/01/20/19/50/43_1484913043351525351_2850008
test-bucket	log_ship_oss_example	year=%Y/mon=%m/day=%d/log_%H%M%s	oss://test-bucket/log_ship_oss_example/year=2017/mon=01/day=20/log_195043_1484913043351525351_2850008.parquet
test-bucket	log_ship_oss_example	ds=%Y%m%d/%H	oss://test-bucket/log_ship_oss_example/ds=20170120/19_1484913043351525351_2850008.snappy
test-bucket	log_ship_oss_example	%Y%m%d/	oss://test-bucket/log_ship_oss_example/20170120/_1484913043351525351_2850008
test-bucket	log_ship_oss_exa	%Y%m%d%H	oss://test-

	mple		bucket/log_ship_oss_example/20170712019_1484913043351525351_2850008

使用Hive或MaxCompute等大数据平台分析OSS数据时，如果希望使用Partition信息，可以设置每一层目录上为key=value格式（Hive-style partition）。

例如：oss://test-bucket/log_ship_oss_example/year=2017/mon=01/day=20/log_195043_1484913043351525351_2850008.parquet

可以设置三层分区列，分别为：year、mon、day。

日志投递任务管理

在启动 OSS 投递功能后，日志服务后台会定期启动投递任务。您可以在控制台上看到这些投递任务的状态。

通过 **日志投递任务管理**，您可以：

查看过去两天内的所有日志投递任务，了解其状态。投递任务状态可以是“成功”、“进行中”和“失败”。“失败”状态则表示您的投递任务出现了因外部原因而无法重试的错误，需要您参与解决问题。

对于创建两天内的投递失败任务，您可在任务列表中查看导致失败的外部原因。修复好这些外部原因后，您可以逐一或者整体重试所有失败任务。

操作步骤

登录 日志服务管理控制台。

选择所需的项目，单击项目名称。

选择所需的日志库，单击日志投递列下的**OSS**。

您可以查看任务开始时间、任务结束的时间、接受日之的时间、数据行数、投递任务的状态等信息。

如果投递任务执行失败，控制台上会显示相应的错误信息，系统会按照策略默认为你重试，您也可以手动重试。

任务重试

一般情况下，日志数据在写入Logstore后的 30 分钟内同步到 OSS。

日志服务默认会按照退火策略重试最近两天之内的任务，重试等待的最小间隔是 15 分钟。当任务执行失败时，第一次失败需要等待 15 分钟再试，第二次失败需要等待 30 分钟（2 乘以 15）再试，第三次失败需要等待 60 分钟（2 乘以 30）再试，以此类推。

如需立即重试失败任务，可以通过控制台单击上图中的 **重试全部失败任务** 或通过 API/SDK 方式指定任务进行重试。

失败任务错误

常见失败任务的错误信息如下：

错误信息	错误原因	处理方法
UnAuthorized	没有权限。	请确认以下配置： - OSS 用户是否已创建角色。 - 角色描述的账号 ID 是否正确。 - 角色是否授予 OSS Bucket 写权限。 - role-arn 是否配置正确。
ConfigNotExist	配置不存在	一般是由于删除投递规则导致，如又重新创建了规则，可以通过重试来解决。
InvalidOssBucket	OSS Bucket 不存在。	请确认以下配置： - OSS Bucket 所在 Region 是否与日志服务Project一致。 - Bucket 名称是否配置正确。
InternalServerError	日志服务内部错误。	通过重试来解决。

OSS 数据存储

可以通过控制台、API/SDK 或其它方式访问到 OSS 数据。

如使用 Web 管理控制台访问，进入 OSS 服务，选择 Bucket，单击 **Object管理** 即可看到有日志服务投递过来的数据。

更多 OSS 使用细节请参考 OSS 文档。

Object 地址

oss:// OSS-BUCKET/OSS-PREFIX/PARTITION-FROMAT_RANDOM-ID

路径字段说明

- OSS-BUCKET、OSS-PREFIX 表示 OSS 的 Bucket 名称和目录前缀，由用户配置，INCREMENTID 是系统添加的随机数。
- PARTITION-FORMAT定义为%Y/%m/%d/%H/%M，其中%Y，%m，%d，%H，%M分别表示年、月、日、小时、分钟，由本次投递任务的服务端创建时间通过strptime API计算得到。
- RANDOM-ID是一个投递任务的唯一标识。

目录的时间含义

OSS数据目录是按照投递任务创建时间设置的，假设 5 分钟数据投递一次 OSS，2016-06-23 00:00:00 创建的投递任务，它投递的数据是 2016-06-22 23:55 后写入日志服务的数据。如需分析完整的 2016-06-22 全天日志，除了 2016/06/22 目录下的全部 object 以外，还需要检查 2016/06/23/00/ 目录下前十分钟的 Object 是否有包含 2016-06-22 时间的日志。

Object存储格式

JSON

请参考OSS投递JSON存储。

Parquet

请参考OSS投递Parquet存储。

CSV

请参考OSS投递CSV存储。

本文档主要介绍日志服务投递OSS使用JSON存储的相关配置，关于投递日志到OSS的其它内容请参考投递日志到 OSS。

OSS文件压缩类型及文件地址见下表。

压缩类型	文件后缀	OSS文件地址举例
不压缩	无	oss://oss-shipper-shenzhen/ecs_test/2016/01/26/20/54_1453812893059571256_937
snappy	.snappy	oss://oss-shipper-shenzhen/ecs_test/2016/01/26/20/54_1453812893059571256_937.snappy

不压缩

Object由多条日志拼接而成，文件的每一行是一条JSON格式的日志，样例如下：

```
{"__time__":1453809242,"__topic__":"","__source__":"10.170.148.237","ip":"10.200.98.220","time":"26/Jan/2016:19:54:02 +0800","url":"POST /PutData?Category=YunOsAccountOpLog&AccessKeyId=U0UjpekFQOVJW45A&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20GMT&Topic=raw&Signature=pD12XYLmGxKQ%2Bmkd6x7hAgQ7b1c%3D HTTP/1.1","status":"200","user-agent":"aliyun-sdk-java"}
```

snappy压缩

采用 snappy 官网 的 C++ 实现（Snappy.Compress 方法），对 none 格式数据做文件级别的压缩得到。对 .snappy 文件解压缩后即可得到对应的 none 格式文件。

使用 C++ Lib 解压缩

snappy 官网 右侧下载 Lib，执行 Snappy.Uncompress 方法解压。

使用 Java Lib解压缩

xerial snappy-java，可以使用 Snappy.Uncompress 或 Snappy.SnappyInputStream（不支持 SnappyFramedInputStream）。

```
<dependency>
<groupId>org.xerial.snappy</groupId>
<artifactId>snappy-java</artifactId>
<version>1.0.4.1</version>
<type>jar</type>
<scope>compile</scope>
</dependency>
```

注意：1.1.2.1 版本存在 bug 可能无法解压部分压缩文件，至 1.1.2.6 版本修复。

Snappy.Uncompress

```
String fileName = "C:\\我的下载\\36_1474212963188600684_4451886.snappy";
RandomAccessFile randomFile = new RandomAccessFile(fileName, "r");
int fileLength = (int) randomFile.length();
randomFile.seek(0);
byte[] bytes = new byte[fileLength];
int byteread = randomFile.read(bytes);
System.out.println(fileLength);
System.out.println(byteread);
byte[] uncompressed = Snappy.uncompress(bytes);
String result = new String(uncompressed, "UTF-8");
```

```
System.out.println(result);
```

Snappy.SnappyInputStream

```
String fileName = "C:\\我的下载\\36_1474212963188600684_4451886.snappy";
SnappyInputStream sis = new SnappyInputStream(new FileInputStream(fileName));
byte[] buffer = new byte[4096];
int len = 0;
while ((len = sis.read(buffer)) != -1) {
System.out.println(new String(buffer, 0, len));
}
```

Linux 环境解压工具

针对 Linux 环境，我们提供了可以解压 snappy 文件的工具，点击下载 snappy_tool。

```
./snappy_tool 03_1453457006548078722_44148.snappy 03_1453457006548078722_44148
compressed.size: 2217186
snappy::Uncompress return: 1
uncompressed.size: 25223660
```

本文档主要介绍日志服务投递OSS使用Parquet存储的相关配置，关于投递日志到OSS的其它内容请参考投递日志到 OSS。

Parquet存储字段配置

数据类型

Parquet存储支持6种类型：string、boolean、int32、int64、float、double。

日志投递过程中，会将日志服务数据由字符串转换为Parquet目标类型。如果转换到非String类型失败，则该列数据为null。

列配置

请依次填写Parquet中需要的日志服务数据字段名和目标数据类型，在投递时将按照该字段顺序组织Parquet数据，并使用日志服务的字段名称作为Parquet数据列名，以下两种情况发生时将置数据列值为null：

- 该字段名在日志服务数据中不存在。

  改字段由string转换非string（如double、int64等）失败。

  字段配置页面：

可配置的保留字段

在投递OSS过程中，除了使用日志本身的Key-Value外，日志服务保留同时提供以下几个保留字段可供选择：

保留字段	语义
__time__	日志的 Unix 时间戳（是从 1970 年 1 月 1 日开始所经过的秒数），由用户日志字段的 time 计算得到。
__topic__	日志的 topic。
__source__	日志来源的客户端 IP。

JSON格式存储会默认带上以上字段内容。

Parquet、CSV存储可以根据您的需求自行选择。例如您需要日志的topic，那么可以填写字段名：__topic__，字段类型string。

OSS存储地址

压缩类型	文件后缀	OSS文件地址举例
无外部压缩	.parquet	oss://oss-shipper-shenzhen/ecs_test/2016/01/26/20/54_1453812893059571256_937.parquet
snappy	.snappy.parquet	oss://oss-shipper-shenzhen/ecs_test/2016/01/26/20/54_1453812893059571256_937.snappy.parquet

数据消费

E-MapReduce / Spark / Hive

参考社区文档。

单机校验工具

开源社区提供的parquet-tools可以用来文件级别验证Parquet格式、查看schema、读取数据内容。

您可以自行编译该工具或者点击下载日志服务提供的版本。

- 查看Parquet文件schema

```
$ java -jar parquet-tools-1.6.0rc3-SNAPSHOT.jar schema -d 00_1490803532136470439_124353.snappy.parquet |
head -n 30
message schema {
optional int32 __time__;
optional binary ip;
optional binary __source__;
optional binary method;
optional binary __topic__;
optional double seq;
optional int64 status;
optional binary time;
optional binary url;
optional boolean ua;
}

creator: parquet-cpp version 1.0.0

file schema: schema
-------------------------------------------------------------------------------
__time__: OPTIONAL INT32 R:0 D:1
ip: OPTIONAL BINARY R:0 D:1
.......
```

- 查看Parquet文件全部内容

```
$ java -jar parquet-tools-1.6.0rc3-SNAPSHOT.jar head -n 2 00_1490803532136470439_124353.snappy.parquet
__time__ = 1490803230
ip = 10.200.98.220
__source__ = 11.164.232.106
method = POST
__topic__ =
seq = 1667821.0
status = 200
time = 30/Mar/2017:00:00:30 +0800
url =
/PutData?Category=YunOsAccountOpLog&AccessKeyId=U0UjpekFQOVJW45A&Date=Fri%2C%2028%20Jun%2020
13%2006%3A53%3A30%20GMT&Topic=raw&Signature=pD12XYLmGxKQ%2Bmkd6x7hAgQ7b1c%3D HTTP/1.1
```

```
__time__ = 1490803230
ip = 10.200.98.220
__source__ = 11.164.232.106
method = POST
__topic__ =
seq = 1667822.0
status = 200
time = 30/Mar/2017:00:00:30 +0800
url =
/PutData?Category=YunOsAccountOpLog&AccessKeyId=U0UjpekFQOVJW45A&Date=Fri%2C%2028%20Jun%2020
13%2006%3A53%3A30%20GMT&Topic=raw&Signature=pD12XYLmGxKQ%2Bmkd6x7hAgQ7b1c%3D HTTP/1.1
```

更多用法请执行：java -jar parquet-tools-1.6.0rc3-SNAPSHOT.jar -h，参考帮助。

本文介绍日志服务投递OSS使用CSV存储的相关细节，其它内容请参考投递日志到 OSS。

CSV存储字段配置

配置页面

可以在日志服务数据预览或索引查询页面查看一条日志的多个Key-Value，将你需要投递到OSS的字段名（Key）有序填入。

如您配置的Key名称在日志中找不到，CSV行中这里一列值将设置为空值字符串（null）。

配置项

配置项	取值	备注
分隔符 delimiter	字符	长度为1的字符串，用于分割不同字段
转义符 quote	字符	长度为1的字符串，字段内出现分隔符（delimiter）或换行符等情况时，需要用quote前后包裹这个字段，避免读数据时造成字段错误切分
跳出符 escape	字符	长度为1的字符串，默认设置与quote相同，暂不支持修改。字段内部出现quote（当成正常字符而不是转义符）时需要在quote前面加上escape做转义
无效字段内容 null	字符串	当指定Key值不存在时，字段填写该字符串表示该字段无值
投递字段名称 header	布尔	是否在csv文件的首行加上字段名的描述

细节请参考CSV标准、postgresql CSV说明。

可配置的保留字段

在投递OSS过程中，除了使用日志本身的Key-Value外，日志服务保留同时提供以下几个保留字段可供选择：

保留字段	语义
__time__	日志的 Unix 时间戳（是从 1970 年 1 月 1 日开始所经过的秒数），由用户日志字段的 time 计算得到。
__topic__	日志的 topic。
__source__	日志来源的客户端 IP。

JSON格式存储会默认带上以上字段内容。

CSV存储可以根据您的需求自行选择。例如您需要日志的topic，那么可以填写字段名：__topic__。

OSS存储地址

压缩类型	文件后缀	OSS文件地址举例
无压缩	.csv	oss://oss-shipper-shenzhen/ecs_test/2016/01/26/20/54_1453812893059571256_937.csv
snappy	.snappy.csv	oss://oss-shipper-shenzhen/ecs_test/2016/01/26/20/54_1453812893059571256_937.snappy.csv

数据消费

HybridDB

建议配置如下：

- 分隔符 delimiter：逗号(,)
- 转义符 quote：双引号(“)
- 无效字段内容 null：不填写（空）
- 投递字段名称 header：不勾选（HybirdDB默认csv文件行首无字段说明）

更多细节请参考HybidDB相关使用说明。

其它

CSV是可读格式，可以直接从OSS下载以文本形式打开查看。

如果使用了snappy压缩，可以参考OSS投递JSON存储的snappy解压缩说明。

本文是作为快捷授权的补充，介绍细粒度Policy设置、父子账号Role授权等功能。

授权 Role 管理

通过快捷授权，OSS 用户主账号 B 默认创建的 AliyunLogDefaultRole 角色描述如下：

```
{
"Statement": [
{
"Action": "sts:AssumeRole",
"Effect": "Allow",
"Principal": {
"Service": [
"log.aliyuncs.com"
]
}
}
],
"Version": "1"
}
```

如日志服务用户主账号 A 与 OSS 用户主账号 B 相同，保持默认角色描述即可，log.aliyuncs.com 等价于 B_ALIYUN_ID@log.aliyuncs.com。

否则，请登录 账号管理->安全设置 查看 A 账号 ID 并修改角色描述，添加 A_ALIYUN_ID@log.aliyuncs.com（支持添加多个），假设 A 的主账号 ID 为1654218965343050：

```
{
"Statement": [
{
"Action": "sts:AssumeRole",
"Effect": "Allow",
"Principal": {
"Service": [
"1654218965343050@log.aliyuncs.com",
"log.aliyuncs.com"
]
}
}
],
"Version": "1"
}
```

该角色描述 A 有权限通过日志服务获取临时 Token 来操作 B 的资源（细节请参考 权限控制 STS）。

授权 Policy 管理

通过快捷授权，角色 AliyunLogDefaultRole 默认被授予 AliyunLogRolePolicy，具有写用户 B 所有 OSS Bucket 的权限。

AliyunLogRolePolicy 的授权描述如下：

```
{
"Version": "1",
"Statement": [
{
"Action": [
"oss:PutObject"
],
"Resource": "*",
"Effect": "Allow"
}
]
}
```

若希望更精细的访问控制粒度，请解除角色 AliyunLogDefaultRole 的 AliyunLogRolePolicy 授权，并参考 OSS 授权 创建更细粒度的 Policy，授权给角色 AliyunLogDefaultRole。

主子账号 Role 授权

默认情况下，OSS 用户主账号 B 为日志服务主账号 A 创建角色、授权后，由 A 使用角色在日志服务创建 OSS 投递配置。

如果 A 的子账号 a_1 需要使用该角色创建日志投递数据到 OSS 规则，那么，主账号 A 需要为子账号 a_1 授予 PassRole 权限。

进入阿里云“访问控制”控制台，主账号 A 可以为子账号 a_1 授予 AliyunRAMFullAccess 权限。如此，a1 将具备 RAM 所有权限，可以使用 B 为 A 创建的角色来配置日志服务投递 OSS 规则。

AliyunRAMFullAccess 权限较大，如果 A 需要控制 a_1 的权限范围，只授予投递 OSS 的必须权限，可以修改授权策略的 Action、Resource，如下示例（其中45643 只是示例uid，请以实际uid为准）：

```
{
"Statement": [
{
"Action": "ram:PassRole",
"Effect": "Allow",
"Resource": "acs:ram::45643:role/aliyunlogdefaultrole"
}
],
"Version": "1"
}
```

投递日志是日志服务的一个功能，能够帮助您最大化数据价值。您可以选择将收集到的日志数据通过控制台方式投递至MaxCompute或者OSS，做数据长期存储或联合其它系统（如 E-MapReduce）消费数据。一旦启用日志投递功能，日志服务后台会定时把写入到该日志库内的日志投递到对应云产品中。为方便您及时了解投递进度，处理线上问题，日志服务控制台提供了 **日志投递任务管理** 页面，您可以查询指定时间内的数据投递状态。

在**Logstore列表**界面，单击**日志投递**列下的**MaxCompute**或**OSS**，进入**日志投递任务管理**页面。您可以执行以下操作管理您的投递任务：

开启/关闭投递任务

日志投递任务管理页面左上角菜单中选择目标Logstore。

单击Logstore菜单旁的**开启投递**或**关闭投递**。

关闭投递任务后再次开启任务，需要重新配置投递规则。

配置投递规则

成功开启投递任务之后，可以单击**投递配置**修改投递规则。

查看投递任务详情

根据Logstore、时间段和任务投递状态筛选出需要查看的投递任务后，可以在当前页面中查看指定投递任务状态、开始时间、结束时间、接受日志数据时间、任务类型等详细信息。

状态

任务的投递状态有以下三种：

状态	含义	操作
成功	投递任务正常运行状态。	无须关注。
进行中	投递任务进行中。	请稍后查看是否投递成功。
失败	日志数据投递失败。投递任务因外部原因出现了错误，且无法重试。如MaxCompute表结构不符合日志服务规范、无授权等。	请排查投递问题。详细信息请参考 投递日志到MaxCompute或投递日志到 OSS的 **投递任务管理** 部分。

查询分析

除LogHub、LogShipper外，日志服务提供大规模、低成本的实时分析能力（LogSearch/Analytics）。

和其他方案相比，实时日志分析体现在：

- 大规模：支持 PB/Day 数据量
- 采集快：通过API/SDK写入0延迟，Agent提供秒级采集上传能力
- 一致性强：写入后立马可以被分析
- 查询速度快：一秒内，复杂查询（5个条件）可处理10亿级数据
- 分析速度快：一秒内，复杂分析（5个维度聚合+GroupBy）可聚合亿级别数据
- 方式灵活：可以改变任意查询和分析条件，实时拿到结果

基本概念

实时分析查询（Query）：由（Search、Analytics）两个部分组成，中间通过|进行分割：

```
$Search | $Analytics
```

- 查询（Search）：查询条件，可以由关键词、模糊、数值等、区间范围和组合条件等产生。如果为空或”*”，则代表所有数据
- 分析（Analytics）：对查询结果或全量数据进行计算和统计

两部分均为可选，当Search部分为空时代表针对该时间段所有数据不过滤任何条件，直接对结果进行统计。当Analysis部分为空时，代表只返回查询结果，不需要做统计

例如：

1. 查询部分，根据区间条件+模糊匹配定位日志：

```
status in (200 500] and method:Get*
```

2. 分析部分：对访问日志每个URL访问次数，写入与读取流量进行统计，获得Top 100结果

```
select count(1) as c, sum(inflow) as sum_in, sum(outflow) as sum(out), url group by url order by c desc limit 100
```

3. 也可以对这两者进行组合

```
status in (200 500) and method:Get* | select count(1) as c, sum(inflow) as sum_in, sum(outflow) as sum(out), url group by url order by c desc limit 100
```

使用指南

开启查询分析

1. 登录 日志服务管理控制台。
2. 选择目标项目，单击项目名称或者单击右侧的 **管理**。
3. 选择目标日志库并单击日志索引列下的 **查询**。
4. 单击右上角的 **设置查询分析** > **设置**
   - 开启查询、统计后意味着数据将会在后台被索引，会产生索引的流量，以及索引对应存储的空间
   - 如果不需要查询分析功能，可以点击删除

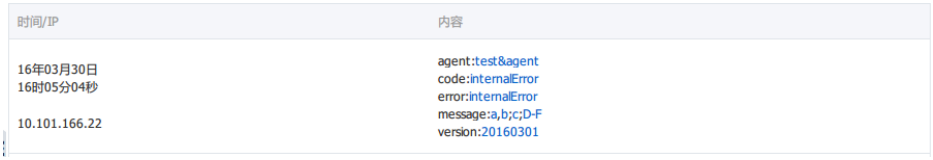
进入设置菜单

查询设置

根据具体的查询业务需求，可以选择合适的索引方法，既能达到高效查询的需求，也能节省使用索引及存储费用。

- 所有查询不需要指定键名称（Key）
  - 可以只设置 **全文查询**
  - 无需设置 **字段查询分析**
- 部分查询需要指定字段名称（Key）
  - 根据需求，对特定字段（Key）创建键值索引
- 查询需要指定数值范围
  - 可以设置特定字段（Key）索引类型为long或者double

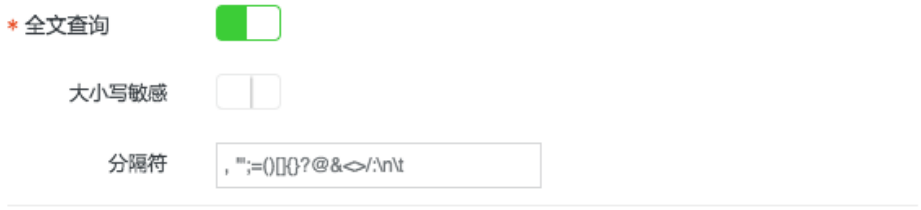
例如日志内容为：



全文查询属性

- 大小写敏感：选择 **false** 表示不区分大小写，即查询关键字“INTERNALERROR”和“internalerror”都能查询到样例日志。如果选择 **true**，表示区分大小写，只能通过关键字“internalError”查询到样例日志。

- 分词符：根据指定单字符，将日志内容切分成多个关键词。例如一条日志内容为a,b;c;D-F。设置分隔符为逗号“，”、分号“；”和连字符“-”，可以将日志内容切分为5个关键词”a” “b” “c” “D” “F”。



字段查询属性

- 全文查询与字段查询区别：默认的索引会查询日志中所有Key对应的内容，只要有一个命中，就会被查询到。例如，日志样例中，如果查询“internalError”，在error和code两个Key中都满足该查询条件

  如果只需要查询error为“internalError”的日志内容，需要设置字段索引，如下图示：

  其中，**字段名称** 即为您指定日志内容特定字段Key，其它两项属性 **大小写敏感** 和 **分词符** 与 **全文查询** 中的功能一致。创建完成如上图的索引属性后，可根据如下关键字查询获取error字段为“internalError” 的日志内容。

```
error:internalError
```

- 如果键对应内容为数值，并且需要按照数值范围进行查询，需要将类型属性设置为“**long**”（整数）或者“**double**”（小数），当设置为数值类型后，**大小写敏感**和**分词符**属性将失效，对于该键的查询只能通过数值范围。例如，您可以通过以下关键字查询键值范围为1000~2000的longkey。

```
longKey > 1000 and longKey < 2000
```

统计设置

在查询之外，我们需要对某些字段进行统计，例如：对所有流量字段求和 sum（inflow），计算平均延时 avg（latency），或计算uv distinct_count(ip)，对字符串处理 substring(text, 0, 10) 等。 可以对需要进行处理字段开启统计。

使用

1. 查询分析日志，查询分析演示视频
2. 语法：查询语法， 分析语法
3. 对查询：设置报警
4. 对查询：配置仪表盘
5. 案例：Nginx访问日志统计，行车轨迹日志分析, 销售系统日志分析

价格

- 计费模式
- 对比：与自建ELK全访问对比
- 对比：与数据仓库+ES综合对比

日志服务控制台提供和 API/SDK 同样能力的日志查询功能。与此同时，控制台还提供了非常直观的交互界面帮助您查询和理解查询结果。除此之外，您还可以指定查询的日志主题、返回的最大日志条数及返回日志的Offset等信息。

> **注意**：在查询日志时您必须指定查询的Logstore和查询时间区域。Logstore必须属于所指定的Project，时间区域不可以超过Logstore的日志存储周期。

使用控制台查询日志

功能特点

- 提供统计图表界面来展示查询结果的分布情况。并用不同颜色的柱状图标示日志查询结果是否完整。
- 提供灵活的翻页、排序机制帮助用户浏览返回的日志原始数据。
- 提供直观的交互方式让您修正查询条件。例如，通过在柱状图上拖拽的方式修改查询时间范围，或者点击日志原始数据上的关键字来添加查询关键字。
- 提供 上下文查询功能 帮助定位。

操作步骤

1. 登录 日志服务管理控制台。
2. 选择需要的项目，单击项目名称。
3. 选择所需的日志库并单击**查询**。
4. 您可以输入查询条件（日志库，Topic，关键字，时间间隔，时间范围等），并单击右上角的 **搜索** 查询日志。

查询语法

查询语法请参见 日志查询语法 章节，对查询结果进行分析参见分析语法章节。

其他功能

为满足您在日志检索过程中的多方面需求，日志服务为您提供检索功能以外的多样化实用功能。您可以在检索日志的同时，进行以下相关操作：

查询条件另存为**快速查询**。

将当前查询条件另存为**快速查询**后，您可以在报警规则中使用该快速查询条件，或者按照该查询条件一键检索。设置快速查询后，单击指定查询条件一行右侧的**查看**，日志服务会在弹窗中为您返回快速查询结果。

查询条件另存为**报警**。

将当前查询条件另存为**报警**后，可以为该报警建立报警规则。详情请参考设置报警规则。

原始日志的查询结果和**日志的时间分布**。

原始日志方式提供**次数分布图**，可以查看符合查询条件的日志数目及这些日志在整个查询时间区域上的分布情况；同时在图表下方为您展示符合查询条件的日志内容，您可以根据需求调整查询结果的显示方式与排序方式。

统计图表方式提供柱状图、折线图、曲线图、线面图和饼图五种统计图表显示方式。您可以根据统计分析的需要选择统计图表类型。

查询延时

日志服务内部对于写入Logstore的日志按照其日志时间戳划分为实时数据和历史数据。具体定义及其对查询的结果影响如下。

- 实时数据：日志中时间点为服务器当前时间点 (-180秒，900秒]。例如，日志时间为 UTC 2014-09-25 12:03:00，服务器收到时为 UTC 2014-09-25 12:05:00，则该日志在写入Logstore时实时建立索引，且从写入Logstore到可以查询到的延时在 3 秒内。
- 历史数据：日志中时间点为服务器当前时间点 [-7 x 86400秒, -900秒)。例如，日志时间为 UTC 2014-09-25 12:00:00，服务器收到时为 UTC 2014-09-25 12:05:00，则该日志被作为历史数据处理。这类日志从写入Logstore到可以查询到的延时最多为 5 分钟。

如果您查询需要检索的日志数据量很大时（如查询时间跨度非常长，您的日志数据量很大等），则一次查询请求无法检索完所有数据。在这种情况下，日志服务会把已有的数据返回给您，并在返回结果中告知您该查询结果并不完整。如此同时，服务端会缓存 15 分钟内的查询结果。当查询请求的结果有部分被缓存命中，则服务端会在这次请求中继续扫描未被缓存命中的日志数据。为了减少您合并多次查询结果的工作量，日志服务会把缓存命中的查询结果与本次查询新命中的结果合并返回给您。因此日志服务可以让您通过以相同参数反复调用该接口来获取最终完整结果。

使用 SDK 查询日志

日志服务提供多种语言（Java、.NET、PHP 和 Python）的 SDK，且这些语言的 SDK 都支持全功能的日志查询接口。关于 SDK 的更多信息请参考 日志服务 SDK。

使用 API 查询日志

日志服务提供 REST 风格的 API，基于 HTTP 协议实现。日志服务的 API 同样提供全功能的日志查询接口。具体参考请见 日志服务 API。

为了能够帮助您更有效地查询日志，Log Service 提供一套查询语法用以表达查询条件。您可以通过 Log Service API 中的 GetLogs 和 GetHistograms 接口或者在 Log Service 控制台的查询页面指定查询条件。本文档详细说明该查询条件的语法。

索引类型

日志服务支持通过两种模式建立对日志库索引：

- 全文索引：将整行日志作为整体进行查询，既不区分键与数值（Key，Value）。
- 键值索引：指定键（Key）情况下进行查询，例如 FILE:app、Type:action。在该键下被包含字符串都会命中。

语法关键词

LogSearch 查询条件支持如下关键字：

名称	语义
and	双目运算符。形式为 query1 and query2，表示 query1 和 query2 查询结果的交集。如果多个单词间没有语法关键词，默认是 and 的关系。
or	双目运算符。形式为 query1 or query2，表示 query1 和 query2 查询结果的并集。
not	双目运算符。形式为 query1 not query2，表示符合 query1 并且不符合 query2 的结果，相当于 query1–query2。如果只有 not query1，那么表示从全部日志中选取不包含 query1 的结果。
(,)	左右括号用于把一个或多个子 query 合并成一个 query，用于提高括号内 query 的优先级。
:	用于 key-value 对的查询。term1:term2 构成一个 key-value 对。如果 key 或者 value 内有空格、冒号:等保留字符，需要用双引号""把整个 key 或者 value 包括起来。
“	把一个关键词转换成普通的查询字符。左右引号内部的任何一个 term 都会被查询，而不会当成语法关键词。或者在 key-value 查询中把左右引号内的所有 term 当成一个整体。
\	转义符。用于转义引号，转义后的引号表示符号本身，不会当成转义字符，例如 "\""。
|	管道运算符，表示前一个计算的基础上进行更多计算，例如 query1 | timeslice 1h | count。
timeslice	时间分片运算符，表示多长时间的数据作为一个整体进行计算，使用方式有 timeslice 1h，timeslice 1m，timeslice 1s 分别表示以 1 小时，1 分钟，1s 作为一个整体。例如 query1 | timeslice 1h | count 表示查询 query 这个条件，并且返回以 1 小时为时间分片的总次数。
count	计数运算符，表示日志条数。
*	模糊查询关键字，用于替代 0 个或多个字符，例如：que*，会返回 que 开头的所有命中词。注意：该查询最多返回100个符合关键词下结果
?	模糊查询关键字，用于替代一个字符，比如 qu?ry，会返回以 qu 开头，以 ry 结尾，并且中间还有一个字符的所有命中词。
__topic__	查询某个 topic 下数据，新的语法下，可以在 query 中查询 0 个或多个 topic 的数据，例如 __topic__:mytopicname。
__tag__	查询某个 tag key 下某个 tag value，例如 __tag__:tagkey:tagvalue。

source	查询某个 IP 的数据，例如 source:127.0.0.1。
>	查询某个字段下大于某个数值的日志，例如 latency > 100。
>=	查询某个字段下大于或等于某个数值的日志，例如 latency >= 100。
<	查询某个字段下小于某个数值的日志，例如 latency < 100。
<=	查询某个字段下小于或等于某个数值的日志，例如 latency <= 100。
=	查询某个字段下等于某个数值的日志，例如 latency = 100。
in	查询某个字段处于某个范围内的日志，使用中括号表示闭区间，使用小括号表示开区间，括号中间使用两个数字，数字中间为若干个空格。例如 latency in [100 200]或 latency in (100 200]。

注意：

- 语法关键词不区分大小写。
- 语法关键字的优先级由高到底排序为：> " > ( ) > and not > or。
- Log Service 还保留以下关键字的使用权，如果您需要使用以下关键字，请使用双引号包含起来：sort asc desc group by avg sum min max limit。
- 同时配置全文索引和键值索引时，如果两者的分词字符不一样，那么使用全文查询方式时数据无法查出。
- 使用数值查询的前提条件是给该列设置类型为double或long，如果您没有设置类型，或者数值范围查询的语法不正确，日志服务会将该查询条件解释成全文索引，可能与您的期望的结果不同。
- 如果您之前把某列配置为文本类型，现在改成数值类型，那么之前的数据只支持=查询。

查询示例

- 同时包含 a 和 b 的日志： a and b 或者 a b。
- 包含 a 或者包含 b 的日志：a or b。
- 包含 a 但是不包含 b 的日志：a not b。
- 所有日志中不包含 a 的日志：not a。
- 查询包含 a 而且包含 b，但是不包括 c 的日志：a and b not c。
- 包含 a 或者包含 b，而且一定包含 c 的日志：(a or b) and c。
- 包含 a 或者包含 b，但不包括 c 的日志：(a or b) not c。
- 包含 a 而且包含 b，可能包含 c 的日志：a and b or c。
- FILE 字段包含 apsara的日志： FILE:apsara。
- FILE 字段包含 apsara 和 shennong 的日志：FILE:"apsara shennong" 或者 FILE:apsara FILE:shennong 或者 FILE:apsara and FILE:shennong。

- 包含 and 的日志：and。
- FILE 字段包含 apsara 或者 shennong 的日志: FILE:apsara or FILE:shennong。
- file info 字段包含 apsara 的日志："file info":apsara。
- 包括引号的日志：\"。
- 查询以 shen 开头的所有日志：shen*。
- 查询 FILE 字段下，以 shen 开头的所有日志：FILE:shen*。
- 查询以 shen 开头，以 ong 结尾，中间还有一个字符的日志：shen?ong。
- 查询包括以 shen 开头，并且包括以 aps 开头的日志：shen* and aps*。
- 查询以 shen 开头的日志的分布，时间片为 20 分钟: shen*| timeslice 20m | count。
- 查询 topic1 和 topic2 下的所有数据： __topic__:topic1 or __topic__ : topic2。
- 查询 tagkey1 下 tagvalue2 的所有数据：__tag__ : tagkey1 : tagvalue2。
- 查询latency大于等于100，并且小于200的所有数据，有两种写法: latency >=100 and latency < 200或latency in [100 200)。
- 查询latency 大于100的所有请求，只有一种写法： latency > 100。
- 查询不包含爬虫的日志，并且http_referer中不包含opx的日志： not spider not bot not http_referer:opx。

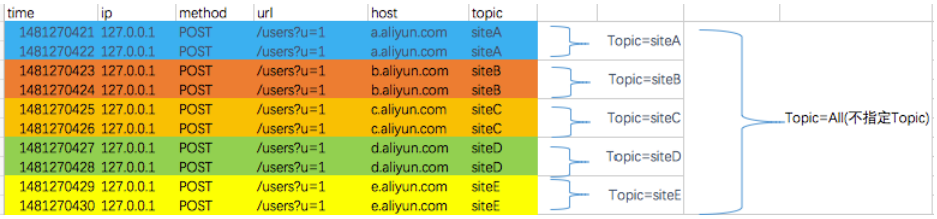
其他语法

指定或跨 Topic 查询

每个Logstore根据Topic可以划分成一个或多个子空间，当进行查询时，指定Topic可以限定查询范围，达到更快速度。因此我们推荐对Logstore有二级分类需求的用户使用Topic进行划分。

当指定一个或多个Topic进行查询时，仅从符合条件的Topic中进行查询。但不输入Topic，默认查询所有Topic下的数据。

例如，使用Topic来划分不同域名下日志：



Topic 查询语法：

- 支持查询所有Topic下的数据，在查询语法和参数中都不指定Topic表示查询所有Topic的数据。
- 支持在Query中查询Topic，查询语法为 __topic__:topicName。同时仍然支持旧的模式，即在URL参数中指定Topic。
- 支持查询多个Topic，例如 __topic__:topic1 or __topic__:topic2 表示查询Topic1和Topic2下的数据的并集。

分析语法

日志服务提供类似于SQL的聚合计算功能，该功能结合了**查询功能**和SQL的计算功能，对查询结果进行计算。

语法示例：

```
status>200 |select avg(latency),max(latency) ,count(1) as c GROUP BY  method  ORDER BY c DESC  LIMIT 20
```

基本语法：

```
[search query] | [sql query]
```

search条件和计算条件以|分割，表示以search query从日志中过滤出需要的日志，并对这些日志进行SQL query计算。search query的语法为日志服务专有语法，参见**查询语法文档**。

前提条件

要使用分析功能，必须在**查询分析设置**中点击SQL涉及的字段下**开启分析**开关，详情请参考**简介**中设置步骤。

- 如果不开启统计，默认只提供每个shard最多1万行数据的计算功能，而且延时比较高。
- 开启后可以提供秒级别快速分析。
- 开启后只对新数据生效。
- 开启统计后不会产生额外费用。

限制说明

1. 每个Projectde 最高并发为5。
2. 单列varchar，最大长度位512，超过后会截断。

支持的SQL语法

日志服务支持以下SQL语法，详细内容请点击链接查看。

- SELECT聚合计算函数：
  - **通用函数**
  - **映射**
  - **估算**
  - **数学统计函数**
  - **数学计算**
  - **字符串函数**

    - 日期函数
    - URL转化函数
    - 正则式函数
    - JSON函数
    - 类型和类型转换
    - IP识别函数
- GROUP BY 语法
- 窗口函数
- HAVING语法
- ORDER BY 语法
- LIMIT 语法
- CASE WHEN语法
- 列的别名
- 嵌套子查询

语法结构

SQL语法结构如下：

- SQL语句中不需要填写from子句和where子句， 默认from表示从当前Logstore的数据中查询，where条件为search query。
- 支持的子句包括SELECT、GROUP BY、ORDER BY [ASC,DESC]、LIMIT、HAVING。
- 默认情况下只返回前10个结果，如要返回更多请加上limit n，例如* | select count(1) as c, ip group by ip order by c desc limit 100。

内置字段

日志服务内置了一些字段供统计，当用户配置了任何一个有效列后，就会自动加上这些内置字段。

字段名	类型	含义
__time__	bigint	日志的时间。
__source__	varchar	日志来源IP。注意，在搜索时，该字段是source，在SQL中才会带上前后各两个下划线。
__topic__	varchar	日志的Topic。

限制说明

1. 每个Project的最高并发为5。
2. 单列varchar，最大长度为512，超过后会截断。

使用实例

统计每小时的PV、UV和最高延时对应的用户请求，延时最高的10个延时：

```
*|select date_trunc('hour',from_unixtime(__time__)) as time,
count(1) as pv,
approx_distinct(userid) as uv,
max_by(url,latency) as top_latency_url,
max(latency,10) as top_10_latency
group by 1
order by time
```

日志服务查询分析功能支持通过通用聚合函数进行日志分析，详细语句及含义如下：

语句	含义	示例
arbitrary(x)	随机返回x列中的一个值	latency > 100 | select arbitrary(method)
avg(x)	计算x列的算数平均值	latency > 100 | select avg(latency)
checksum(x)	计算某一列的checksum，返回base64编码	latency > 100 | select checksum(method)
count(*)	计算某一列的个数	-
count(x)	计算某一列非空的个数	latency > 100 | count(method)
geometric_mean(x)	计算某一列的几何平均数	latency > 100 | select geometric_mean(latency)
max_by(x,y)	返回当y取最大值时，x当前的值	查询延时最高的时候，对应的method： latency>100 | select max_by(method,latency)
max_by(x,y,n)	返回y最高的n行，对应的x的值	查询延时最高的3行，对应的method： latency > 100 | select max_by(method,latency,3)
min_by(x,y)	返回当y取最小值时，x当前的值	查询延时最低的请求，对应的method： * | select min_by(x,y)
min_by(x,y,n)	返回y最小的n行，对应的x的值	查询延时最小的3行，对应的method： * | select max_by(method,latency,3)
max(x)	返回最大值	latency > 100| select max(inflow)
min(x)	返回最小值	latency > 100| select min(inflow)

sum(x)	返回x列的和	latency > 10 | select sum(inflow)
bitwise_and_agg(x)	对某一列的所有数值做and计算	-
bitwise_or_agg(x)	对某一列的数值做or计算	-

日志服务查询分析功能支持通过映射函数进行日志分析，详细语句及含义如下：

语句	含义	示例
histogram(x)	按照x的每个值GROUP BY，计算count。语法相当于select count group by x	latency > 10 | histogram(status)。上述语法等同于latency > 10 | select count(1) group by status。
map_agg(Key,Value)	返回Key、Value组成的map	展示每个method的随机的latency：latency > 100 | select map_agg(method,latency) 。
multimap_agg(Key,Value)	返回Key、Value组成的多Value map	返回每个method的所有的latency： latency > 100 | select multimap_agg(method,latency)。

日志服务查询分析功能支持通过估算进行日志分析，详细语句及含义如下：

语句	含义	示例
approx_distinct(x)	估算x列的唯一值的个数	-
approx_percentile(x,percentage)	对于x列排序，找出大约处于percentage位置的数值	找出位于一半位置的数值：approx_percentile(x,0.5)
approx_percentile(x, percentages)	与上述用法类似，但可以指定多个percentage，找出每个percentage对应的数值	approx_percentile(x,array(0.1, 0.2))
numeric_histogram(buckets, Value)	对于数值列，分多个桶进行统计。把Value这一列，分到buckets个桶中，返回每个桶的Key及对应的count数，相当于针对数值的select count group by	对于POST请求，把延时分成10个桶，看每个桶的大小：method:POST | select numeric_histogram(10,latency)

日志服务查询分析功能支持通过数学统计函数进行日志分析，详细语句及含义如下：

语句	含义	示例
corr(y, x)	给出两列的相关度，结果从0到1	latency>100| select corr(latency,request_size)

covar_pop(y, x)	计算总体协方差	latency>100| select covar_pop(request_size,latency)
covar_samp(y, x)	计算样本协方差	latency>100| select covar_samp(request_size,latency)
regr_intercept(y, x)	返回输入值的线性回归截距。y是依赖值。 x是独立值	latency>100| select regr_intercept(request_size,latency)
regr_slope(y,x)	返回输入值的线性回归斜率。y是依赖值。 x是独立值	latency>100| select regr_slope(request_size,latency)
stddev(x) 或stddev_samp(x)	返回x列的样本标准差	latency>100| select stddev(latency)
stddev_pop(x)	返回x列的总体标准差	latency>100| select stddev_pop(latency)
variance(x) 或 var_samp(x)	计算x列的样本方差	latency>100| select variance(latency)
var_pop(x)	计算x列的总体方差	latency>100| select variance(latency)

日志服务查询分析功能支持通过数学计算函数进行日志分析，您可以结合查询语句和数学计算函数，对日志查询结果进行数学计算。

数学运算符

数学运算符支持 + - * / % 。可以用在SELECT子句中。

样例：

```
*|select avg(latency)/100 , sum(latency)/count(1)
```

数学计算函数说明

日志服务支持以下运算函数：

函数名	含义
abs(x)	返回x列的绝对值
cbrt(x)	返回x列的立方根
ceiling (x)	返回x列向上最接近的整数
cosine_similarity(x,y)	返回稀疏向量x和y之间的余弦相似度

degrees	把弧度转化为度
e()	返回自然常数
exp(x)	返回自然常数的指数
floor(x)	返回x向下最接近的整数
from_base(string,radix)	以radix进制解释string
ln(x)	返回自然对数
log2(x)	返回以2为底，x的对数
log10(x)	返回以10为底，x的对数
log(x,b)	返回以b为底，x的对数
pi()	返回π
pow(x,b)	返回x的b次幂
radians(x)	把度转化成弧度
rand()	返回随机数
random(0,n)	返回[0，n)随机数
round(x)	x四舍五入
sqrt(x)	返回x的平方根
to_base(x, radix)	把x以radix进制表示
truncate(x)	丢弃掉x的小数部分
acos(x)	反余弦
asin(x)	反正弦
atan(x)	反正切
atan2(y,x)	y/x的反正切
cos(x)	余弦
sin(x)	正弦
cosh(x)	双曲余弦
tan(x)	正切
tanh(x)	双曲正切
infinity()	double最大值
is_infinity(x)	判断是否是最大值
is_finity(x)	判断是否是最大值
is_nan(x)	判断是否是数值

日志服务查询分析功能支持通过字符串函数进行日志分析，详细语句及含义如下：

函数名	含义
length(x)	字段长度
levenshtein_distance(string1, string2)	返回两个字符串的最小编辑距离
lower(string)	转化成小写
ltrim(string)	删掉左侧的空白字符
replace(string, search)	把字符串中string中的search删掉
replace(string, search,rep)	把字符串中string中的search替换为rep
reverse(string)	翻转string
rtrim(string)	删掉字符串结尾的空白字符
split(string,delimeter,limit)	把字符串分裂成array，最多取limit个值。生成的结果为数组，下标从1开始
split_part(string,delimeter,offset)	把字符串分裂成array，取第offset个字符串。生成的结果为数组，下标从1开始
strpos(string, substring)	查找字符串中的子串的开始位置。返回结果从1开始，如果不存在则返回0
substr(string, start)	返回字符串的子串，start下标从1开始
substr(string, start, length)	返回字符串的子串，start下标从1开始
trim(string)	删掉字符串开头和结尾的空白字符
upper(string)	转化为大写字符
concat(string,string.....)	把两个或多个字符串拼接成一个字符串

注意：字符串需要用单引号包含，双引号表示列名。例如：a=’ abc’ 表示列a=字符串abc；a=” abc” 表示a列=abc列。

日志服务支持时间函数和日期函数，您可以在分析语句中使用本文档中介绍的日期和时间函数。

日期时间类型

1. unixtime：以int类型表示从1970年1月1日开始的秒数，例如1512374067 表示的时间是Mon Dec 4 15:54:27 CST 2017。日志服务每条日志中内置的时间__time__即这种类型。
2. timestamp类型：以字符串形式表示时间，例如2017-11-01 13:30:00。

日期函数

日志服务支持的常见日期函数如下：

函数名	含义	样例

current_date	当天日期	latency>100| select current_date
current_time	时:分；秒，毫秒 时区	latency>100| select current_time
current_timestamp	结合current_date + current_time的结果	latency>100| select current_timestamp
current_timezone()	返回时区	latency>100| select current_timezone()
from_iso8601_timestamp(string)	把iso8601时间转化成带时区的时间	latency>100| select from_iso8601_timestamp(iso8601)
from_iso8601_date(string)	把iso8601转化成天	latency>100| select from_iso8601_date(iso8601)
from_unixtime(unixtime)	把unix时间转化为时间戳	latency>100| select from_unixtime(1494985275)
from_unixtime(unixtime,string)	以string为时区，把unixtime转化成时间戳	latency>100| select from_unixtime (1494985275,'Asia/Shanghai')
localtime	当前时间	latency>100| select localtime
localtimestamp	当前时间戳	latency>100| select localtimestamp
now()	等同于current_timestamp	-
to_unixtime(timestamp)	timestamp转化成unixtime	*| select to_unixtime('2017-05-17 09:45:00.848 Asia/Shanghai')

时间函数

MySQL时间格式

日志服务还支持MySQL时间格式，包括%a、%b、%y 等。

函数名	含义	样例
date_format(timestamp, format)	把timestamp转化成以format形式表示	latency>100| select date_format (date_parse('2017-05-17 09:45:00','%Y-%m-%d %H:%i:%S'), '%Y-%m-%d') group by method
date_parse(string, format)	把string以format格式解析，转化成timestamp	latency>100|select date_parse('2017-05-17 09:45:00','%Y-%m-%d %H:%i:%S') group by method

格式	描述
%a	星期缩写 (Sun .. Sat)
%b	月份缩写 (Jan .. Dec)
%c	月份，数值类型 (1 .. 12) [4]
%D	每月的第几天，带上后缀 (0th, 1st, 2nd, 3rd, …)
%d	每月的第几天 (01 .. 31) [4]
%e	每月的第几天 (1 .. 31) [4]
%H	小时 (00 .. 23)
%h	小时 (01 .. 12)
%I	小时，12时制 (01 .. 12)
%i	分钟 (00 .. 59)
%j	每年的第几天 (001 .. 366)
%k	小时 (0 .. 23)
%l	小时 (1 .. 12)
%M	月份，英文 (January .. December)
%m	月份，数值 (01 .. 12) [4]
%p	AM 或 PM
%r	时间 12小时制，格式为(hh:mm:ss 后跟AM 或 PM)
%S	秒 (00 .. 59)
%s	秒 (00 .. 59)
%T	时间，24时制 (hh:mm:ss)
%U	每年的第几周(00 .. 53)
%u	每年的第几周 (00 .. 53)
%V	每年的第几周 (01 .. 53)
%v	每年的第几周Week (01 .. 53), where Monday is the first day of the week; used with %x
%W	星期几的名称 (Sunday .. Saturday)
%w	一周第几天 (0 .. 6), 星期日为第0天
%Y	年份
%y	年份，两位数
%%	%转义字符

时间段对齐函数

日志服务支持时间段对齐函数，可以按照秒、分钟，小时、日、月、年等对齐。这个函数常用于一些按照时间进行统计的场景。

函数语法：

```
date_trunc(unit, x)
```

参数：

Unit可选的值有(x取2001-08-22 03:04:05.000)：

Unit	转化后结果
second	2001-08-22 03:04:05.000
minute	2001-08-22 03:04:00.000
hour	2001-08-22 03:00:00.000
day	2001-08-22 00:00:00.000
week	2001-08-20 00:00:00.000
month	2001-08-01 00:00:00.000
quarter	2001-07-01 00:00:00.000
year	2001-01-01 00:00:00.000

x 可以是一个timestamp类型，也可以是unix time。

date_trunc只能在按照一些固定时间间隔统计，如果我们需要按照灵活的时间维度进行统计，例如统计每5分钟的，这种场景下，我们需要按照数学取模方法进行GROUP BY。

```
* | SELECT count(1) as pv,  __time__ - __time__% 300 as minute5 group by minute5 limit 100
```

上述公式中的%300表示按照5分钟进行取模对齐。

日期函数样例

以下为使用时间格式的一个综合样例。

```
*|select  date_trunc('minute' ,  __time__)  as t,
truncate (avg(latency) ) ,
current_date
group by t
order by t desc
limit 60
```

URL函数支持从标准URL路径中提取字段，一个标准的URL如下：

```
[protocol:][//host[:port]][path][?query][#fragment]
```

常见URL函数

函数名	含义	样例
url_extract_fragment(url)	提取出URL中的fragment，结果为varchar类型	*| select url_extract_fragment(url)
url_extract_host(url)	提取出URL中的host，结果为varchar类型	*|select url_extract_host(url)
url_extract_parameter(url, name)	提取出URL中的query中name对应的参数值，结果为varchar类型	*|select url_extract_parameter(url)
url_extract_path(url)	提取出URL中的path，结果为varchar类型	*|select url_extract_path(url)
url_extract_port(url)	提取出URL中的端口，结果为bigint类型	*|select url_extract_port(url)
url_extract_protocol(url)	提取出URL中的协议，结果为varchar类型	*|select url_extract_protocol(url)
url_extract_query(url)	提取出URL中的query，结果为varchar类型	*|select url_extract_query(url)
url_encode(value)	对URL进行转义编码	*|select url_encode(url)
url_decode(value)	对URL进行解码	*|select url_decode(url)

正则式函数解析一串字符串，并且返回需要的一部分子串。

常见的正则式函数及含义如下：

函数名	含义	样例
regexp_extract_all(string, pattern)	返回字符串中命中正则式的所有子串,返回结果是一个字符串数组	*|SELECT regexp_extract_all('5a 67b 890m', '\d+') 结果为['5','67','890']
regexp_extract(string, pattern)	返回字符串命中的正则式的第一个子串	*|SELECT regexp_extract_all('5a 67b 890m', '\d+') 结果为'5'
regexp_like(string, pattern)	判断字符串是否命中正则式，返回bool类型，正则式可以只命中字符串的一部分	*|SELECT regexp_like('5a 67b 890m', '\d+m') 结果为true
regexp_replace(string,	把字符串中命中正则式的部分替	*|SELECT regexp_replace('5a

pattern, replacement)	换成replacement	67b 890m', '\d+','a') 结果为 'aa ab am'
regexp_replace(string, pattern)	把字符串中命中正则式的部分删除，相当于 regexp_replace(string,patterm,'')	*|SELECT regexp_replace('5a 67b 890m', '\d+') 结果为'a b m'
regexp_split(string, pattern)	使用正则式把字符串切分成数组	*|SELECT regexp_split('5a 67b 890m', '\d+') 结果为 ['a','b','m']

JSON函数，可以解析一段字符串为JSON类型，并且提取JSON中的字段。JSON主要有两种结构：map和array。如果一个字符串解析成JSON失败，那么返回的是null。

日志服务支持以下常见的JSON函数：

函数名	含义	样例
json_parse(string)	把字符串转化成JSON类型	SELECT json_parse('[1, 2, 3]') 结果为JSON类型数组
json_format(json)	把JSON类型转化成字符串	SELECT json_format(json_parse('[1, 2, 3]')) 结果为字符串
json_array_contains(json, value)	判断一个JSON类型数值，或者一个字符串（内容是一个JSON数组）是否包含某个值	SELECT json_array_contains(json_parse('[1, 2, 3]'), 2)或 SELECT json_array_contains('[1, 2, 3]', 2)
json_array_get(json_array, index)	同json_array_contains，是获取一个JSON数组的某个下标对应的元素	SELECT json_array_get('["a", "b", "c"]', 0)结果为'a'
json_array_length(json)	返回JSON数组的大小	SELECT json_array_length('[1, 2, 3]') 返回结果3
json_extract(json, json_path)	从一个JSON对象中提取值，JSON路径的语法类似 $.store.book[0].title，返回结果是一个JSON对象	SELECT json_extract(json, '$.store.book');
json_extract_scalar(json, json_path)	类似json_extract，但是返回结果是字符串类型	-
json_size(json,json_path)	获取JSON对象或数组的大小	SELECT json_size('[1, 2, 3]') 返回结果3

日志服务配置中支持的类型包括long、double、text类型。在查询中支持的类型包括bigint、double、varchar、timestamp、int等。

类型转换函数，强制把某一列转换成指定类型：

```
cast(value AS type) → type
try_cast(value AS type) → type
```

IP 识别函数，可以识别一个IP是内网IP还是外网IP，也可以判断IP所属的国家、省份、城市。

函数名	含义	样例
ip_to_domain(ip)	判断IP所在的域，是内网还是外网。返回intranet或internet。	SELECT ip_do_domain(ip)
ip_to_country(ip)	判断IP所在的国家。	SELECT ip_to_country(ip)
ip_to_province(ip)	判断IP所在的省份，如果是外国，则返回国家名称。	SELECT ip_to_province(ip)
ip_to_city(ip)	判断IP所在的城市，如果是外国，则返回国家名称。	SELECT ip_to_city(ip)
ip_to_provider(ip)	获取IP对应的网络运营商。	SELECT ip_to_provider(ip)

使用样例

1. 在查询中过滤掉内网访问请求,看请求总数

```
* | select count(1)  where ip_to_domain(ip) !='intranet'
```

2. 查看top10的访问省份

```
* | SELECT count(1) as pv, ip_to_province(ip) as province GROUP BY province order by pv desc limit 10
```

响应结果样例

```
[
{
"__source__": "",
"__time__": "1512353137",
"province": "浙江省",
"pv": "4045"
}, {
"__source__": "",
"__time__": "1512353137",
"province": "上海市",
"pv": "3727"
}, {
"__source__": "",
"__time__": "1512353137",
"province": "北京市",
"pv": "954"
```

```
}, {
"__source__": "",
"__time__": "1512353137",
"province": "内网IP",
"pv": "698"
}, {
"__source__": "",
"__time__": "1512353137",
"province": "广东省",
"pv": "472"
}, {
"__source__": "",
"__time__": "1512353137",
"province": "福建省",
"pv": "71"
}, {
"__source__": "",
"__time__": "1512353137",
"province": "阿联酋",
"pv": "52"
}, {
"__source__": "",
"__time__": "1512353137",
"province": "美国",
"pv": "43"
}, {
"__source__": "",
"__time__": "1512353137",
"province": "德国",
"pv": "26"
}, {
"__source__": "",
"__time__": "1512353137",
"province": "吉隆坡",
"pv": "26"
}
]
```


在上述结果中包含了内网IP，有时候，我们开发自己的测试是从内网发出的，为了过滤掉这部分访问请求，我们可以使用下边的分析语句：

3. 过滤掉内网请求，查看top 10的网络访问省份

```
* | SELECT count(1) as pv, ip_to_province(ip) as province WHERE ip_to_domain(ip) != 'intranet'  GROUP BY province ORDER BY pv desc limit 10
```

4. 查看不同国家的平均响应延时，最大响应延时，最大延时对应的request

```
* | SELECT AVG(latency),MAX(latency),MAX_BY(requestId, latency) ,ip_to_country(ip) as country group by country
```

```
limit 100
```

5. 查看不同网络运营商的平均延时

```
* | SELECT AVG(latency) , ip_to_provider(ip) as provider group by provider limit 100
```

GROUP BY 支持多列。GROUP BY支持通过SELECT的列的别名来表示对应的KEY。

样例：

```
method:PostLogstoreLogs |select avg(latency),projectName,date_trunc('hour',__time__) as hour   group by
projectName,hour
```

别名hour代表第三个SELECT列date_trunc('hour',__time__)。这类用法对于一些非常复杂的query非常有帮助
。

GROUP BY 支持GROUPING SETS、CUBE、ROLLUP。

样例：

```
method:PostLogstoreLogs |select avg(latency)   group by cube(projectName,logstore)
method:PostLogstoreLogs |select avg(latency) group by GROUPING SETS ( ( projectName,logstore),
(projectName,method))
method:PostLogstoreLogs |select avg(latency) group by rollup(projectName,logstore)
```

实践样例

按照时间进行GROUP BY

每条日志都内置了一个时间列__time__，当打开任意一列的统计功能后，会自动给时间列打开统计。

使用date_trunc函数，可以把时间列对齐到小时(hour)、分钟(minute)、天(day)、月(month)、年(year)。
date_trunc接受一个对齐单位，和一个unix time或者timestamp类型的列，例如__time__。

- 按照每小时、每分钟统计计算PV

```
* | SELECT count(1) as pv , date_trunc('hour',__time__) as hour group by hour order by hour limit 100
* | SELECT count(1) as pv , date_trunc('minute',__time__) as minute group by minute order by minute limit 100
```

注: limit 100表示最多获取100行，如果不加LIMIT语句，默认最多获取10行数据。

- 按照灵活的时间维度进行统计，例如统计每5分钟的，date_trunc只能在按照一些固定时间间隔统计

，这种场景下，我们需要按照数学取模方法进行GROUP BY。

```
* | SELECT count(1) as pv,  __time__ - __time__% 300 as minute5 group by minute5 limit 100
```

上述公式中的%300表示按照5分钟进行取模对齐。

在GROUP BY 中提取非agg列

在标准SQL中，如果使用了GROUP BY语法，那么在SELECT时，只能选择SELECT GROUP BY的列原始内容，或者对任意列进行聚合计算，不允许获取非GROUP BY列的内容。

例如，以下语法是非法的，因为b是非GROUP BY的列，在按照a进行GROUP BY时，有多行b可供选择，系统不知道该选择哪一行输出。

```
* | select a, b , count(c) gropu by a
```

为了达到以上目的，可以使用arbitrary函数输出b：

```
* | select a, arbitrary(b) ，count(c) gropu by a
```

窗口函数简介

窗口函数用来跨行计算。普通的SQL聚合函数只能用来计算一行内的结果，或者把所有行聚合成一行结果。窗口函数，可以跨行计算，并且把结果填到到每一行中。

窗口函数语法:

```
SELECT key1, key2, value,
rank() OVER (PARTITION BY key2
ORDER BY value DESC) AS rnk
FROM orders
ORDER BY key1,rnk
```

核心部分是:

```
rank() OVER (PARTITION BY  KEY1 ORDER BY KEY2 DESC)
```

其中rank()是一个聚合函数，可以使用分析语法中的任何函数，也可以使用本文档列出的函数。PARTITION BY是值按照哪些桶进行计算。

窗口中使用的特殊聚合函数

函数	含义

rank()	在窗口内，按照某一列排序，返回在窗口内的序号
row_number()	返回在窗口内的行号
first_value(x)	返回窗口内的第一个value，一般用法是窗口内数值排序，获取最大值
last_value(x)	含义和first value相反
nth_value(x, offset)	窗口内的第offset个数
lead(x,offset,defaut_value)	窗口内x列某行之后offset行的值，如果不存在该行，则取default_value
lag(x,offset,defaut_value)	窗口内x列某行之前offset行的值，如果不存在该行，则取default_value

使用样例

1. 在整个公司的人员中，获取每个人的薪水在部门内排名

```
* | select department, persionId, sallary , rank() over(PARTITION BY department order by sallary desc) as sallary_rank order by department,sallary_rank
```

响应结果:

department	persionId	sallary	sallary_rank
dev	john	9000	1
dev	Smith	8000	2
dev	Snow	7000	3
dev	Achilles	6000	4
Marketing	Blan Stark	9000	1
Marketing	Rob Stark	8000	2
Marketing	Sansa Stark	7000	3

2. 在整个公司的人员中，获取每个人的薪水在部门内的占比

```
* | select department, persionId, sallary *1.0 / sum(sallary) over(PARTITION BY department ) as sallary_percentage
```

响应结果:

department	persionId	sallary	sallary_percentage
dev	john	9000	0.3
dev	Smith	8000	0.26

dev	Snow	7000	0.23
dev	Achilles	6000	0.2
Marketing	Blan Stark	9000	0.375
Marketing	Rob Stark	8000	0.333
Marketing	Sansa Stark	7000	0.29

3. 按天统计，获取每天UV相对前一天的增长情况

```
* | select day ,uv, uv *1.0 /(lag(uv,1,0) over() ) as diff_percentage from
(
select approx_distinct(ip) as uv, date_trunc('day',__time__) as day from log group by day order by day asc
)
```

响应结果:

day	uv	diff_percentage
2017-12-01 00:00:00	100	null
2017-12-02 00:00:00	125	1.25
2017-12-03 00:00:00	150	1.2
2017-12-04 00:00:00	175	1.16
2017-12-05 00:00:00	200	1.14
2017-12-06 00:00:00	225	1.125
2017-12-07 00:00:00	250	1.11

日志服务查询分析功能支持标准SQL的HAVING语法，和GROUP BY配合使用，用于过滤GROUP BY的结果。

格式：

```
method :PostLogstoreLogs |select avg(latency),projectName group by projectName  HAVING avg(latency) > 100
```

HAVING和WHERE的区别

HAVING 用于过滤GROUP BY之后的聚合计算的结果，WHERE在聚合计算之间过滤原始数据。

示例

对于气温大于10℃的省份，计算每个省份的平均降雨量，并在最终结果中只显示平均降雨量大于100mL的省份

：

```
* | select avg(rain) ,province where teporature > 10 group by province having avg(rain) > 100
```

ORDER BY 用于对输出结果进行排序，目前只支持按照一列进行排序。

语法格式：

```
order by  列名 [desc|asc]
```

样例：

```
method :PostLogstoreLogs |select avg(latency) as avg_latency,projectName group by projectName
HAVING avg(latency) > 5700000
order by avg_latency desc
```

LIMIT后跟一个数字，表示最多输出的行数。如果不添加LIMIT语句，默认只输出10行。

注意：不支持Limit offset、lines语法。

样例：

```
*| select avg(latency) as avg_latency , method group by method order by  avg_latency desc limit 100
```

支持CASE WHEN语法，对连续数据进行归类。例如，从http_user_agent中提取信息，归类成Android和iOS两种类型：

```
SELECT
CASE
WHEN http_user_agent like '%android%' then 'android'
WHEN http_user_agent like '%ios%' then 'ios'
ELSE 'unknown' END
as http_user_agent,
count(1) as pv
group by http_user_agent
```

样例：

- 计算状态码为200的请求占总体请求的比例：

```
* | SELECT
sum(
```

```
CASE
WHEN status =200 then 1
ELSE 0 end
) *1.0 / count(1) as status_200_percentage
```

- 统计不同延时区间的分布

```
* | SELECT `
CASE
WHEN latency < 10 then 's10'
WHEN latency < 100 then 's100'
WHEN latency < 1000 then 's1000'
WHEN latency < 10000 then 's10000'
else 's_large' end
as latency_slot,
count(1) as pv
group by latency_slot
```

在SQL标准中，列名必须由字母、数字、下划线组成，且以字母开头。

如果在日志收集配置中，用户如果配置了不符合SQL标准的列名（例如User-Agent），那么需要在配置统计属性的页面，给列取一个别名，用于查询。别名仅仅用于SQL统计，在底层存储时，仍然是原始名称，搜索时需要使用原始名称。

此外，当用户原始的列名特别长时，也可以取一个别名来代替原始列名查询。

别名样例：

原始列名	别名
User-Agent	ua
User.Agent	ua
123	col
abceefghijklmnopqrstuvw	a

不同的query在分析时的效率是不同的，在此为您提供部分常见的优化query方式，供您参考。

1. 尽量避免对字符串列进行GROUP BY计算
2. GROUP BY多列时，把字典大的字段放在前面
3. 使用估算函数
4. 在SQL中获取需要的列，尽量不要读取所有列

1. 尽量避免对字符串列进行GROUP BY计算

对字符串进行GROUP BY，会导致大量的hash计算，这部分计算量往往会占据整体计算的50%以上。

例如以下两个query:

```
* | select count(1) as pv , date_trunc('hour',__time__) as time group by time
* | select count(1) as pv , from_unixtime(__time__-__time__%3600) as time group by __time__-__time__%3600
```

Query 1 和2达到的效果是相同的，都是计算每个小时的日志count数，但是Query 1 首先把时间转化成字符串，例如2017-12-12 00:00:00， 然后对这个字符串进行GROUP BY。 Query 2是先对时间整点值进行计算，GROUP BY计算后才会转化成字符串类型。Query 1需要对字符串进行hash操作，所以在执行效率上，Query 2更佳。

2. GROUP BY多列时，把字典大的字段放在前面

例如，province有13个，用户有1亿。

```
快：  * | select province,uid,count(1) group by province,uid
慢: * | select province,uid,count(1) group by uid,province
```

3. 使用估算函数

估算函数的性能要比精确计算好很多。估算会损失一些可接受的精确度，来达到快速计算的效果。

```
快：  * | select approx_distinct(ip)
慢: * | select count(distinct(ip))
```

4. 在SQL中获取需要的列，尽量不要读取所有列

获取所有列，请使用**查询语法**。在SQL计算时，尽量只读取需要参与计算的列，这会加快计算。

```
快 : * |select a,b c
慢  : * |select *
```

案例列表

1. 500错误率极速增加时，触发报警
2. 当流量暴跌时，触发报警
3. 按照数据区间分桶，在每个桶内计算平均延时
4. 在group by的结果中，返回百分比

案例内容

1. 尽量避免对字符串列进行GROUP BY计算

统计每分钟的500错误率，当最近5分钟错误率超过40%时触发报警。

```
status:500 | select __topic__, max_by(error_count,window_time)/1.0/sum(error_count) as error_ratio,
sum(error_count) as total_error  from (
select __topic__, count(*) as error_count , __time__ - __time__ % 300 as window_time from log group by __topic__,
window_time
)
group by __topic__ having max_by(error_count,window_time)/1.0/sum(error_count) > 0.4 and sum(error_count) >
500 order by total_error desc limit 100
```

2. 当流量暴跌时，触发报警

统计每分钟的流量，挡最近的流量出现暴跌时，触发报警。 由于在最近的一分钟内，统计的数据不是一个完整分钟的，所以，需要除以(max(**time**) - min(**time**)) 进行归一化，统计每个分钟内的流量均值。

```
* | SELECT SUM(inflow) / (max(__time__) - min(__time__))  as inflow_per_minute, date_trunc('minute',__time__)  as
minute group by minute
```

3. 按照数据区间分桶，在每个桶内计算平均延时

```
* | select avg(latency) as latency , case when originSize < 5000 then 's1' when originSize < 20000 then 's2' when
originSize < 500000 then 's3' when originSize < 100000000 then 's4' else 's5' end as os group by os
```

4. 在group by的结果中，返回百分比

不同部门的count结果，及其所占百分比。该query结合了子查询、窗口函数。其中sum(c) over() 表示计算所有行的和。

```
* |  select   department, c*1.0/ sum(c) over ()  from(select  count(1) as c, department   from log  group by
department)
```

针对一些复杂的查询场景，一层SQL无法满足需求，通过SQL嵌套查询可以满足复杂的需求。

嵌套子查询和无嵌套查询的区别在于，要在SQL中指定from 条件。在查询中要指定from log这个关键字，表示从日志中读取原始数据。

样例:

```
* | select sum(pv) from
(
select count(1) as pv from log group by method
)
```

当您展开一份日志文件，每一条日志都记录一个事件，并且往往不是孤立存在的，连续的若干条日志可以回放整个事件序列的发生过程。

日志上下文查询是指定日志来源（机器 + 文件）和其中一条日志，将该日志在原始文件中的前若干条（上文）或后若干条日志（下文）也查找出来，尤其是在 DevOps 场景下对于理清问题来龙去脉来说可谓是一把利器。

日志服务控制台提供专门的查询页面，您可以在控制台查看指定日志在原始文件中的上下文信息，体验类似于在原始日志文件中向上或向下翻页功能。通过查看指定日志的上下文信息，您可以在业务故障排查中快速查找相关故障信息，方便定位问题。

应用场景

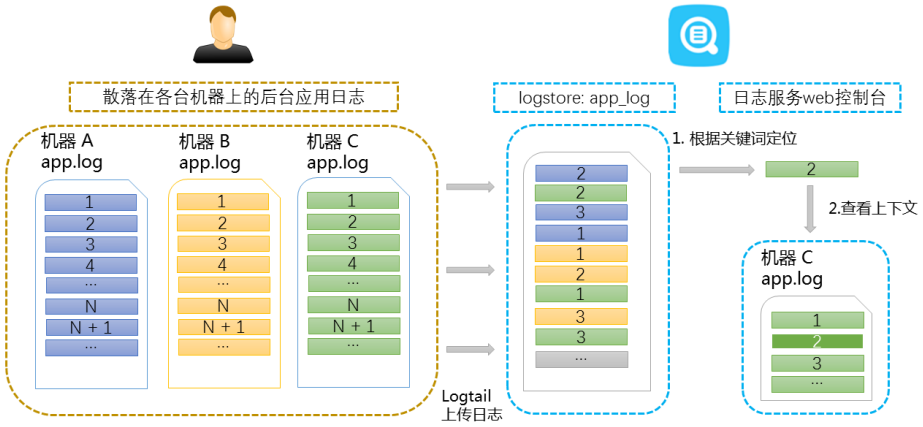
例如，O2O 外卖网站在服务器上的程序日志里会记录一次订单成交的轨迹：

用户登录 > **浏览商品** > **点击物品** > **加入购物车** > **下单** > **订单支付** > **支付扣款** > **生成订单**

如果用户下单失败了，运维人员需要快速定位问题原因。传统的上下文查询中，需要管理员相关人员添加机器登录权限，然后调查者依次登录应用所部署的每一台机器，以订单 ID 为关键词搜索应用程序日志文件，帮助判断下单失败原因。

在日志服务中，可以按照以下步骤排查：

1. 到服务器上安装日志采集客户端 Logtail，并到控制台上添加机器组、日志采集配置，然后 Logtail 开始上传增量日志。
2. 到日志服务供控制台日志查询页面，指定时间段根据订单 ID 找到订单失败日志。
3. 以查到的错误日志为基准，向上翻页直到发现与之相关的其它日志信息（例如：信用卡扣款失败）。



功能优势

- 不侵入应用程序，日志文件格式无需改动。
- 在日志服务控制台上可以查看任意机器、文件的指定日志上下文信息，无需登录每台机器查看日志文件。
- 结合事件发生的时间线索，在日志服务控制台指定时间段快速定位可疑日志后再进行上下文查询，往往可以事半功倍。
- 不用担心服务器存储空间不足或日志文件轮转（rotate）造成的数据丢失，到日志服务控制台上随时可以查看过往的数据。

前提条件

- 使用 Logtail 采集日志 上传数据到日志库，除创建机器组、采集配置以外无需其它配置。
- 开通 日志索引查询 功能。

注意：

- 上下文查询功能目前仅支持 使用 Logtail 采集日志 上传的数据，且需要开通 日志索引查询 功能。
- 上下文查询功能暂不支持syslog日志。
- Java SDK/Producer Library/Log4J Appender等SDK写入数据暂不支持。

操作步骤

登录 日志服务管理控制台。

选择所需的项目，单击项目名称。

在 **Logstore列表** 页面，选择所需的日志库并单击日志索引列下的 **查询**。

查询结果页中任一条日志的左侧有 **上下文浏览** 链接，表明该日志支持上下文查看功能。

选中一条日志，单击 **上下文浏览**。

控制台从普通搜索模式跳转到上下文查询模式。

使用鼠标在当前页面上下滚动查看选中日志周边的日志信息。如需要继续查看上文和下文，单击 **更早** 或 **更新** 进行翻页浏览。

除Restful API外，您还可以使用JDBC + 标准SQL 92进行日志查询与分析。

连接参数

连接参数	示例	说明
host	cn-shanghai-intranet.log.aliyuncs.com	访问点，目前仅支持经典网络内网访问和VPC网络访问
port	10005	默认使用10005作为端口号
user	bq2sjzesjmo86kq	访问秘钥 AccesskeyId

password	4fdO1fTDDuZP	访问秘钥 Accesskey
database	sample-project	用户账号下的 项目（Project）
table	sample-logstore	项目下的 日志库（Logstore）

例如通过MySQL命令连接示例如下：

```
mysql -hcn-shanghai-intranet.log.aliyuncs.com -ubq2sjzesjmo86kq -p4fdO1fTDDuZP -P10005

use sample-project; // 使用某个Project
```

前提条件

访问JDBC接口，必须使用主账号的AK或者子帐号的AK。子帐号必须是Project owner的子帐号，同时子帐号具有Project级别的读权限。

语法说明

注意事项

在where条件中必须包含__date__或__time__来限制查询的时间范围。__date__是timestamp类型__time__是bigint类型。

例如：

- __date__ > '2017-08-07 00:00:00' and __date__ < '2017-08-08 00:00:00'
- __time__ > 1502691923 and __time__ < 1502692923

上述两种条件必须出现一个。

过滤语法

关于where下过滤（filter）语法如下：

语义	示例	说明
字符串搜索	key = "value"	查询的是分词之后的结果
字符串模糊搜索	key = "valu*"	查询的是分词之后模糊匹配的结果
数值比较	num_field > 1	支持的比较运算符包括>、>=、 =、 <和<=
逻辑运算	and or not	例如a = "x" and b ="y"或a = "x" and not b ="y"
全文搜索	__line__ ="abc"	如果使用全文索引搜索，需要使用特殊的key（__line__）

计算语法

支持计算操作符参见分析语法。

SQL92语法

过滤 + 计算组合为SQL92语法。

例如对于如下查询：

```
status>200 |select avg(latency),max(latency) ,count(1) as c GROUP BY  method  ORDER BY c DESC  LIMIT 20
```

我们可以将查询中过滤部分+ 时间条件组合成为查询的条件，变成标准SQL92语法：

```
select avg(latency),max(latency) ,count(1) as c from sample-logstore where status>200 and __time__>=1500975424
and __time__ < 1501035044 GROUP BY  method  ORDER BY c DESC  LIMIT 20
```

通过JDBC协议访问

程序调用

开发者可以在任何一个支持MySQL connector的程序中使用MySQL语法连接日志服务。例如使用JDBC或者Python MySQLdb。

使用样例:

```
import com.mysql.jdbc.*;

import java.sql.*;
import java.sql.Connection;
import java.sql.ResultSetMetaData;
import java.sql.Statement;

public class testjdbc {
public static void main(String args[]){
Connection conn = null;
Statement stmt = null;
try {
//STEP 2: Register JDBC driver
Class.forName("com.mysql.jdbc.Driver");

//STEP 3: Open a connection
System.out.println("Connecting to a selected database...");

conn = DriverManager.getConnection("jdbc:mysql://cn-shanghai-intranet.log.aliyuncs.com:10005/sample-
project","accessid","accesskey");
System.out.println("Connected database successfully...");
```

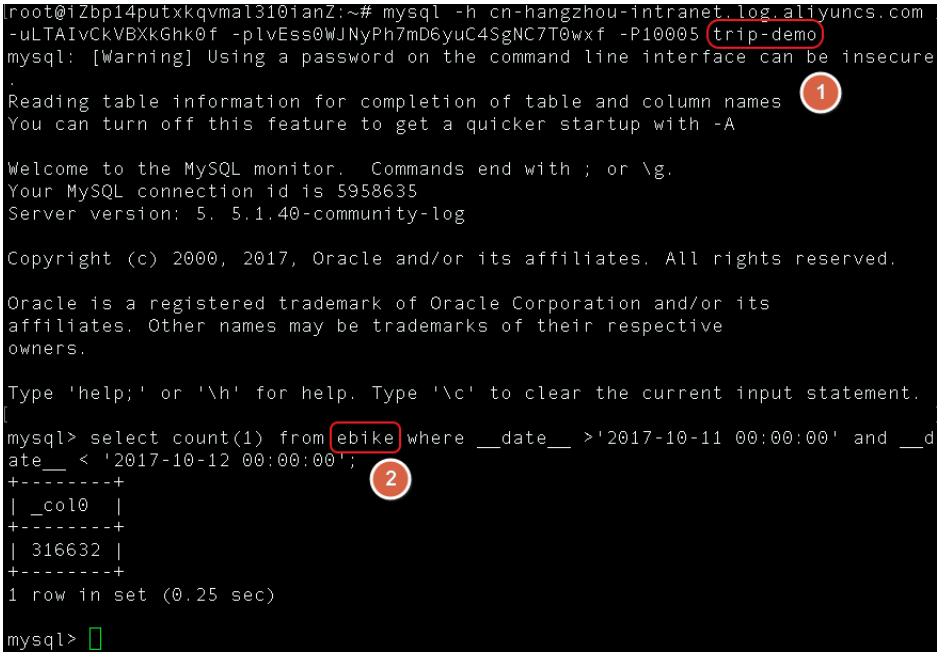
```
//STEP 4: Execute a query
System.out.println("Creating statement...");
stmt = conn.createStatement();
String sql = "SELECT method,min(latency,10) as c,max(latency,10) from sample-logstore where
__time__>=1500975424 and __time__ < 1501035044 and latency > 0 and latency < 6142629 and not
(method='Postlogstorelogs' or method='GetLogtailConfig') group by method " ;

String sql-example2 = "select count(1) ,max(latency),avg(latency),
histogram(method),histogram(source),histogram(status),histogram(clientip),histogram(__source__) from test10
where __date__ >'2017-07-20 00:00:00' and __date__ <'2017-08-02 00:00:00' and __line__='abc#def' and latency <
100000 and (method = 'getlogstorelogS' or method='Get**' and method <> 'GetCursorOrData' )";

String sql-example3 = "select count(1) from sample-logstore where __date__ > '2017-08-07 00:00:00' and __date__
< '2017-08-08 00:00:00' limit 100";

ResultSet rs = stmt.executeQuery(sql);

//STEP 5: Extract data from result set
while(rs.next()){
//Retrieve by column name
ResultSetMetaData data = rs.getMetaData();
System.out.println(data.getColumnCount());
for(int i = 0;i < data.getColumnCount();++i) {
String name = data.getColumnName(i+1);
System.out.print(name+":");
System.out.print(rs.getObject(name));
}
System.out.println();
}
rs.close();
} catch (ClassNotFoundException e) {
e.printStackTrace();
} catch (SQLException e) {
e.printStackTrace();
} catch (Exception e) {
e.printStackTrace();
} finally {
if (stmt != null) {
try {
stmt.close();
} catch (SQLException e) {
e.printStackTrace();
}
}
if (conn != null) {
try {
conn.close();
} catch (SQLException e) {
e.printStackTrace();
}
}
}
}
}
```

工具类调用

在经典网内网/VPC环境通过MySQL Client进行连接。

注意：

1. ①处填写您的Project。
2. ②处填写您的Logstore。



日志服务支持基于您的 **日志查询结果** 进行报警，您可以通过配置报警规则将具体报警内容以短信方式发送到指定手机。

基本流程为：

1. 配置快速查询。
2. 系统定时运行快速查询，判断是否触发报警条件。
3. 发送短信/查看报警结果。

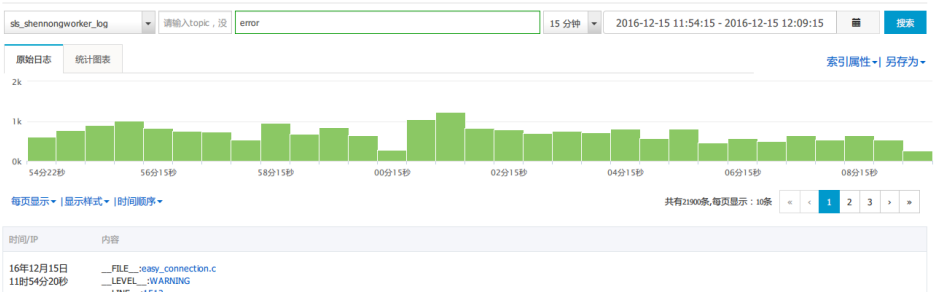
步骤 1 配置快速查询

结果返回方式

日志查询结果可以通过以下两种方式表示：**结果直接返回**和**结果统计**。**结果直接返回**方式可以直接返回命中日志数目；**结果统计**方式提供时间段内命中数目分布情况，是推荐方式。

结果直接返回

例如，您查询最近 15 分钟内包含 error 的数据，条件为 error，一共有 477 条，分布如下：



每条内容都是 Key、Value 组合，您可以对某个 Key 下的 Value 设置报警条件。

> 对于查询结果一次超过 10 条的情况，报警规则只判断 **前10条** ，其中有任何一条符合条件，都会触发报警。

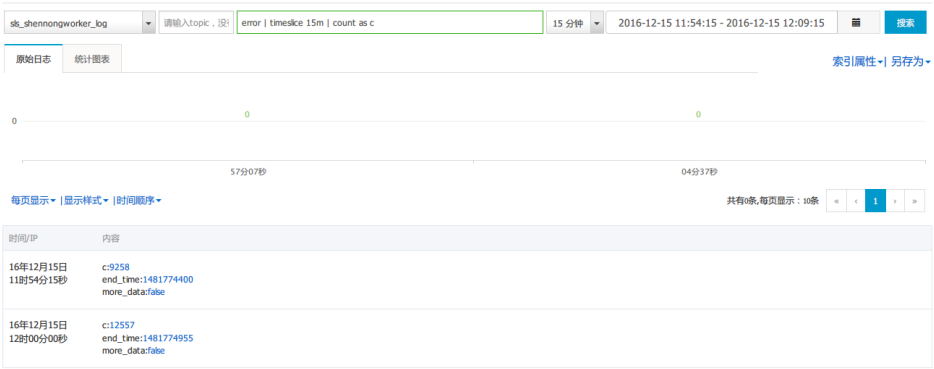
从这个例子可以看到，报警只会判断 477 条中前 10 条，存在误差。因此推荐对统计结果进行报警。

结果统计（Histogram 查询）

结果统计增加 timeslice 语法对查询结果定义统计区间，如下（**详细查询语法**）：

```
where_condition | timeslice 1[hms] |count
h为一小时，m为一分钟，s为一秒钟
```

例如，将上述查询做一个区间汇总，条件为 error | timeslice 15m | count as c，结果如下：



因此，可以设一个条件 c > 10 来判断该每个 15 分钟区间内是否包含 error 关键词是否超过 10 条来进行报警。

通过变更 timeslice 的参数，可以灵活的调整区间大小。例如，汇总 2 个小时窗口内是否出现等。

操作步骤

登录 日志服务管理控制台。

选择所需的项目，单击项目名称。

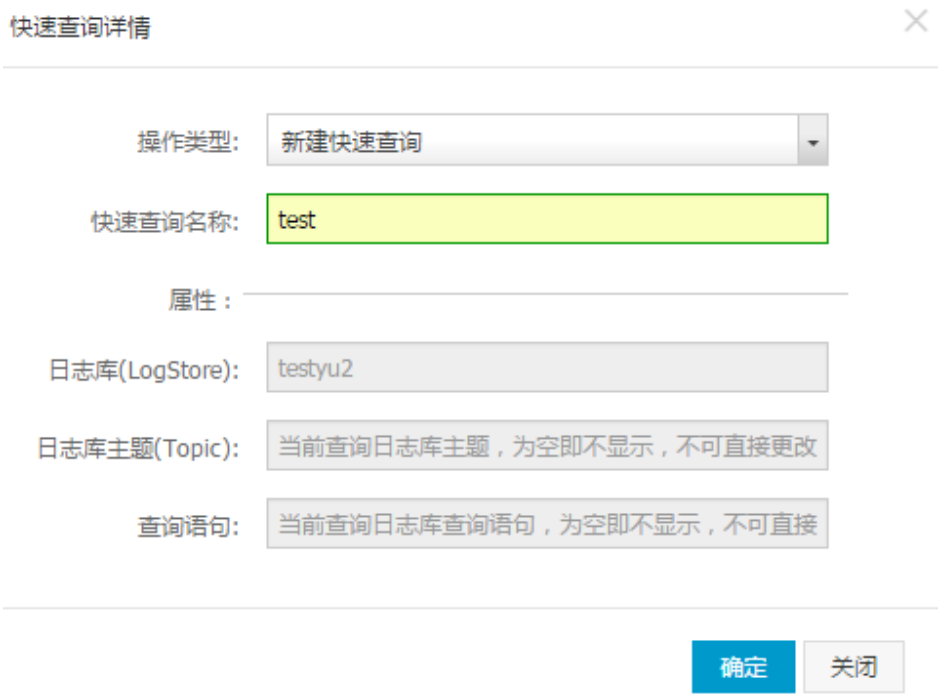
在 **Logstore列表** 页面，选择所需的日志库并单击日志查询列下的 **查询**。

根据需求指定日志库（Logstore）、主题（Topic）和查询语句后，搜索指定日志。

单击页面右上角的 **另存为** > **快速查询**。将查询参数保存为快速查询。

设置快速查询详情并单击 **确定**。

- **操作类型**：选择 **新建快速查询**。
- **快速查询名称**：快速查询的名称。



步骤 2 创建报警规则

将查询参数保存为快速查询后，您可以创建报警规则。

单击 **另存为** > **报警**。规则配置页面如下所示。

设置报警规则并单击 **确定**。

报警通知类型目前支持手机短信 和 消息服务（MNS）两种。

规则说明

快速查询名称：目前支持选择已经创建的快速查询。

数据查询时间：服务端每次执行报警检查读取的数据时间范围，600 即对最近 600 秒到执行检查的当前时间进行查询。

注意：目前服务端每次报警规则检查只会采样处理时间区间开始的前 10 条数据。

检查间隔：服务端每次执行报警检查的时间间隔（目前，最小间隔支持 5min）。

触发次数：报警检查连续触发的次数，比如间隔为 5 分钟，2 代表连续 2 次检查满足报警条件即发送报警（报警最快间隔为 10 分钟）。

字段名称：日志内容中用于报警的 Key 名称。

比较符：支持数值类（大于/大于等于/小于/小于等于）和字符类（包含类/正则匹配），说明如下：

操作	描述	示例
>	该列是否大于数值	$count > 0
<	该列是否小于数值	$count<200
>=	大于等于某个数值	$count>=0
<=	小于等于某个数值	$count<=0
like	匹配子串	$project like “admin”
regex	正则匹配字符串	$project regex match “^/S+$”

步骤 3 查看报警结果

创建完成报警规则后，您可以查看具体的报警结果。

登录 日志服务管理控制台。

选择所需的项目，单击项目名称或右侧的 **管理**。

在 **Logstore列表** 页面，选择左侧导航栏中的 **LogSearch - 索引查询** > **报警**。

选择要查看的报警规则并单击右侧的 **查看报警记录** 即可查看具体报警结果。

报警状态

- **成功**：规则被成功执行，在报警触发详情显示触发标准。
- **失败**：查询、报警规则、或通知阶段失败，可以查看 **报警触发详情** 获得详细信息。
  - 查询失败，一般为语法不正确。
  - 查询调用失败，请提交工单。
  - 规则调用失败，请检查规则参数和返回数据格式是否一致。

报警短信通知

当前，一个报警配置只支持一个手机号码，同一个手机号码一天短信通知上限为 20 条。

消息服务MNS通知

日志服务产生的报警已和消息服务（MNS）打通，支持自定义事件订阅与推送。相比直接短信报警有如下特点：

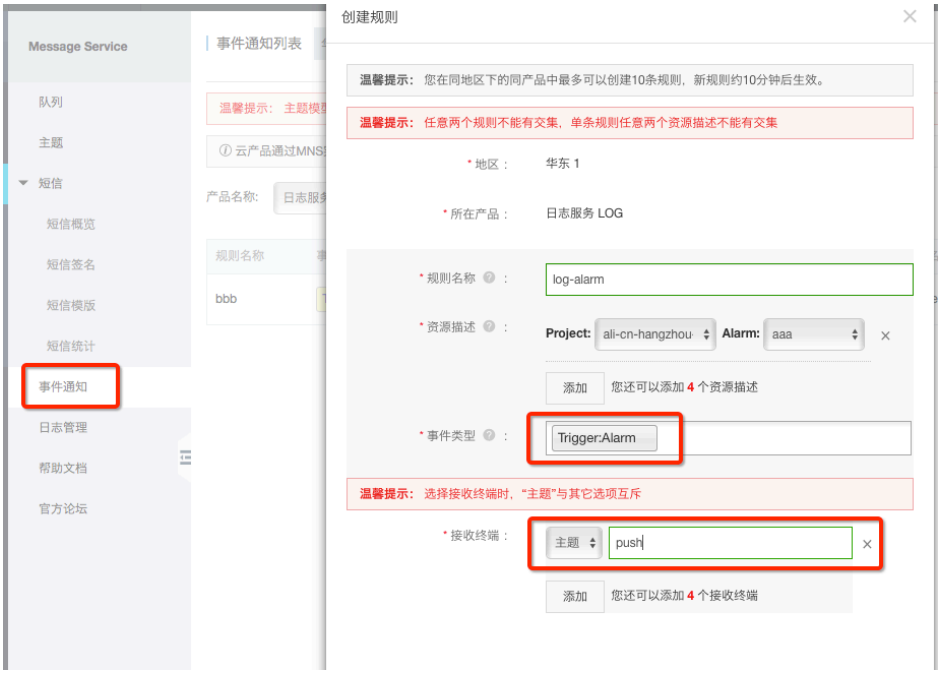
- 多种通知方式支持：短信、邮件、http等
- 可定制短信模板，批量进行发送

配置步骤

日志服务控制台选择通知类型**消息服务**。

打开消息服务控制台

i. 选择事件通知。
ii. 选择同一地域（Region），在产品名称中选择**日志服务-LOG**。
iii. 为事件定义一个名称。
iv. 选择日志服务对应的Project，以及报警列表。
v. 事件类型选择Trigger:Alarm。
vi. 定义事件通知终端，例如主题、队列、或HTTP等。



推送对象为队列时，可以通过预览查看报警事件内容。

i. 内容包含对应的项目，报警规则的名称，以及填写的参数（param）和报警的内容（message）。

```
{"project":"ali-cn-hangzhou-sls-admin", "trigger":"aaa","param":"message service",
"message":"2138 > 1"}
```

确定队列中预览的效果。

在开启查询分析后，除了能在查询框中输入查询（Query）外，我们可以将常用的查询保存到：

- 仪表盘（Dashboard）
- 快速查询（SavedSearch）
- 报警（Alarm）：通过快速查询创建

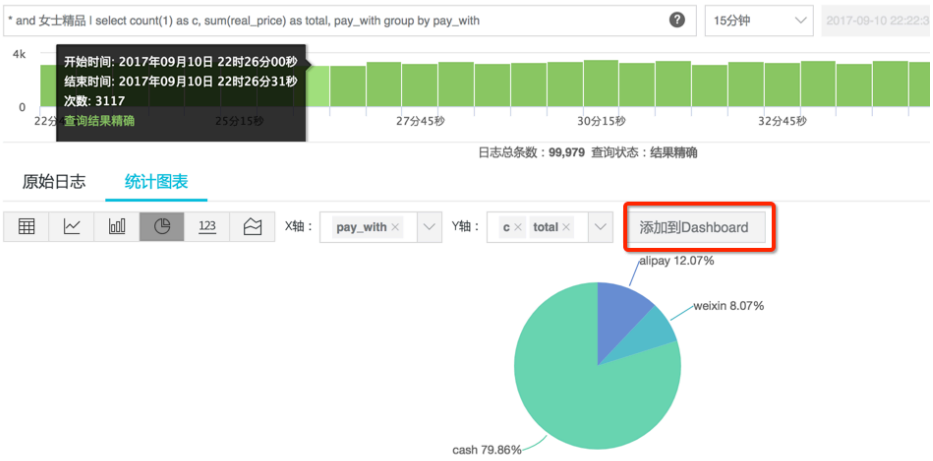
仪表盘演示视频

操作步骤

在搜索框中输入搜索和分析语句，并点击**搜索**。

对查询分析视图添加仪表盘，在查询分析页面选择对应视图，选择**添加到Dashboard**。

添加：在弹出对话框中选择**新建**，或**添加到已有**。



其他操作

查看：对于已存储的Dashboard可以进行以下操作：

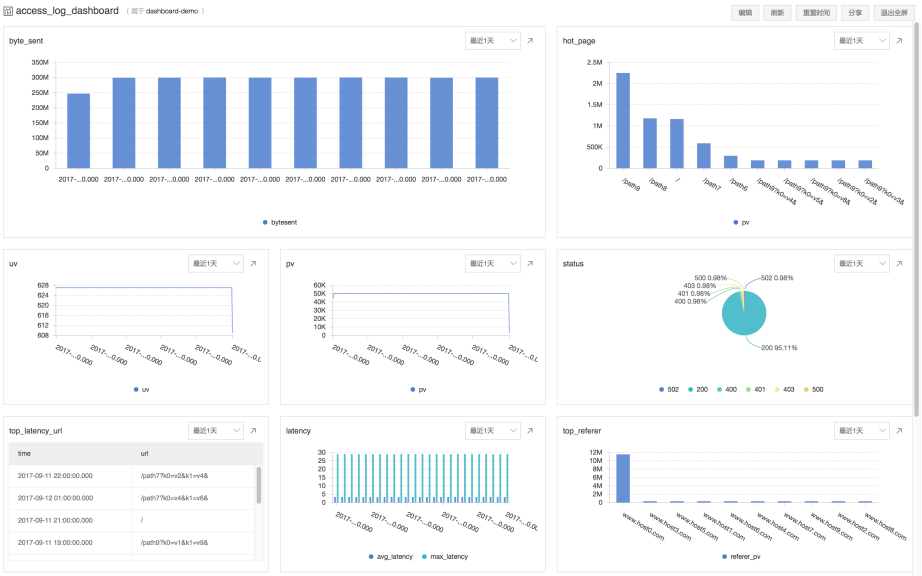
- 项目左侧菜单栏中**查询分析**下**仪表盘**中查看或删除。
- 在查询分析页面，左侧**新建标签**，在弹出菜单中点击**仪表盘**标签。

修改：在仪表盘页面中选择编辑，即可调整图标属性、大小、位置等信息，或点击全屏、刷新等按钮以达到好的效果。

使用限制

- 每个Project下创建最多5个仪表盘，每个仪表盘创建最多10个分析Query同时进行展示。
- 支持线图、柱状图、饼图、数值、面积图显示方式（其它显示方式陆续增加中）。
- 支持单个图表自定义位置、大小缩放等设置保存。

仪表盘视图：可通过链接（用户名：sls_reader1@1654218965343050 密码：pnX-32m-MHH-xbm）试用：



查询方案（ELK/Hive）对比

DevOps 场景日志查询方案对比：ELK（搜索类）、Hadoop/Hive

在互联网大潮中，为应对不断加速的软件、服务交付需求，无论是在创业团队还是大型互联网公司，都已经转

向或逐步转向 DevOps 模式，通过开发（Dev）和运维支持（Ops）之间有效协作，解决跨部门协作、快速响应客户需求、进行持续交付。

日志在 DevOps 下显得越发重要，无论是在问题调查、安全审计、运营支撑等各方面，日志都起到了重要的支撑作用。一个合适的日志解决方案，对于 DevOps 显得尤为重要。

我们从如下方面考察 LogSearch 与 ELK、Hadoop/Hive 类方案的对比：

- 延时：日志产生后，多久可查询
- 查询能力：单位时间扫描数据量
- 查询功能：关键词查询、条件组合查询、模糊查询、数值比较、上下文查询
- 弹性：快速应对百倍流量上涨
- 成本：每 GB 费用
- 可靠性：日志数据安全不丢失

常用方案以及对比

- 自建 ELK：通过 Elastic、Logstash、Kibana 进行对比
- 离线 Hadoop + Hive：将数据存储在 Hadoop，利用 Hive 或 Presto 进行查询（非分析）
- 使用日志服务（LogSearch）

以应用程序日志和 Nginx 访问日志为例（每天 10GB），对比几种方案。

功能项	ELK 类系统	Hadoop + Hive	日志服务
可查延时	1~60 秒(由 refresh_interval 控制)	几分钟~数小时	1~3 秒
查询延时	小于 1 秒	分钟级	小于 1 秒
超大查询	几十秒~数分钟	分钟级	秒级（查询 10 亿日志）
关键词查询	支持	支持	支持
模糊查询	支持	支持	支持
上下文查询	不支持	不支持	支持
数值比较	支持	支持	目前不支持（ETA:2017/2)
连续字符串查询	支持	支持	不支持
弹性	提前预备机器	提前预备机器	秒级 10 倍扩容
写入成本	写入5 元/GB，查询免费	写入免费，查询一次 0.3 元/GB	写入 0.5 元/GB，查询免费
存储成本	≤ 3.36 元/GB*天	≤ 0.035 元/GB*天	≤ 0.016 元/GB*天
可靠性	设置拷贝数	设置拷贝数	SLA > 99.9%，数据 > 99.99999999%

简介

提到日志实时分析，很多人都会想到很火的ELK Stack（Elastic/Logstash/Kibana）来搭建。ELK方案开源，在社区中有大量的内容和使用案例。

阿里云日志服务产品在新版中增强查询分析功能（LogSearch/Analytics），支持对日志数据实时索引与查询分析，并且对查询性能和计算数据量做了大量优化。在这里我们做一个全方位的比较，对于用户关心的点，我们依次展开分析：

- 易用：上手及使用过程中的代价
- 功能（重点）：主要针对查询与分析两块
- 性能（重点）：对于单位大小数据量查询与分析需求，延时如何
- 规模（重点）：能够承担的数据量，扩展性等
- 成本（重点）：同样功能和性能，使用分别花多少钱

易用

对日志分析系统而言，有如下使用过程：

1. 采集：将数据稳定写入
2. 配置：如何配置数据源
3. 扩容：接入更多数据源，更多机器，对存储空间，机器进行扩容
4. 使用：这部分在功能这一节介绍
5. 导出：数据能否方便导出到其他系统，例如做流计算、放到对象存储中进行备份
6. 多租户：如何将数据能否分享给他人使用，使用是否安全等

以下是比较结果：

项目	小项	自建ELK	Log Analytics
采集	协议	Restful API	Restful API
	Agent	Logstash/Beats/FluentD，生态十分丰富	Logtail（为主）+其他(例如Logstash)
配置	单元	提供Index概念用以区分不同日志	Project + Logstore。提供两层概念，Project相当于命名空间，可以在Project下建立多个Logstore
	属性	API + Kibana	API + SDK + 控制台
扩容	存储	增加机器，购买云盘	无需操作
	机器	新增机器	无需操作
	配置	配置Logstash并通过配管系统应用机器	控制台/API 操作，无需配管系统
	采集点	通过配管系统控制，将配置和Logstash安装	控制台/API 操作，无需配管系统

		到机器组	
导出	方式	API/SDK	API/SDK + 各流计算引擎（Spark，Storm，Flink，StreamCompute，CloudMonitor，ARMS） + 存储（OSS + MaxCompute）
多租户	安全	无（非商业版）	https + 传输签名 + 多租户隔离 + 访问控制
	使用	同一账号	子账号，角色，产品，临时授权等

总体而言：

ELK 有非常多的生态和写入工具，使用中的安装、配置等都有较多工具可以参考。LogSearch/Anlaytics是一个托管服务，从接入、配置、使用上集成度非常高，普通用户5分钟就可以接入。并且过程中不需要担心容量、并发等问题，按量计费，弹性伸缩。

功能（查询+分析）

查询主要将符合条件的日志快速命中，分析功能是对数据进行统计与计算。

例如我们有如下需求：所有状态码大于200读请求，根据Ip统计次数和流量，这样的分析请求就可以转化为两个操作：查询到指定结果，对结果进行统计分析。在一些情况下我们也可以不进行查询，直接对所有日志进行分析。

```
1. Status in (200,500] and Method:Get*
2. select count(1) as c, sum(inflow) as sum_inflow, ip group by Ip
```

查询能力对比

类型	小项	自建ELK	Log-Search/Analytics
文本	索引查询	支持	支持
	分词	支持	支持
	中文分词	支持	（计划支持）
	前缀	支持	支持
	后缀	支持	
	模糊	支持	支持
	Wildcast	支持	（计划支持）
数值	long	支持	支持

	double	支持	支持
Nested	Json查询	支持	
Geo	Geo查询	支持	不直接支持，但可以通过区间查询代替
Ip	Ip查询	支持	不直接支持，通过字符串支持
上下文	上下文查询		支持
	上下文过滤		支持

ElasticSearch 支持的数据类型，以及查询方式上要更强。Log-Search/Analytics 支持大部分常用查询，也有一些特色（例如程序日志上下文查询与展开）。

分析能力对比

- ES 5.5 Aggregation
- Log-Search/Analytics分析语法

类型	小项	自建ELK	Log-Search/Analytics
接口	方式	API/SDK	API/SDK + SQL92
	其他协议		JDBC
Agg	Bucketing	支持	支持
	Metric	支持	支持
	Matrix	支持	支持
	Pipeline	有限支持	完整支持
运算	数值		支持
	字符串		支持
	估算		支持
	数理统计		支持
	日期转换		支持
GroupBy	Agg	支持	支持
	Having条件		支持
排序	排序		支持
Join	多表Join		支持

从结果看Log-Search/Analytics 提供的功能是ES 超集，完整支持SQL 92 语义，只要会写SQL 都能直接使用。

性能

接下来我们测试针对相同数据集，分别对比写入数据及查询，和聚合计算能力。

实验环境

1. 测试配置

	自建ELK	LogSearch/LogAnalytics
环境	ECS 4核16GB * 4台 + 高效云盘 SSD	
Shard	10	10
拷贝数	2	3（默认配置，对用户不可见）

1. 测试数据
   - 5列double
   - 5列long
   - 5列text，字典大小分别是256，512，768，1024，1280

以上字段完全随机，如下为一条测试日志样例：

```
timestamp:August 27th 2017, 21:50:19.000
long_1:756,444 double_1:0 text_1:value_136
long_2:-3,839,872,295 double_2:-11.13 text_2:value_475
long_3:-73,775,372,011,896 double_3:-70,220.163 text_3:value_3
long_4:173,468,492,344,196 double_4:35,123.978 text_4:value_124
long_5:389,467,512,234,496 double_5:-20,10.312 text_5:value_1125
```

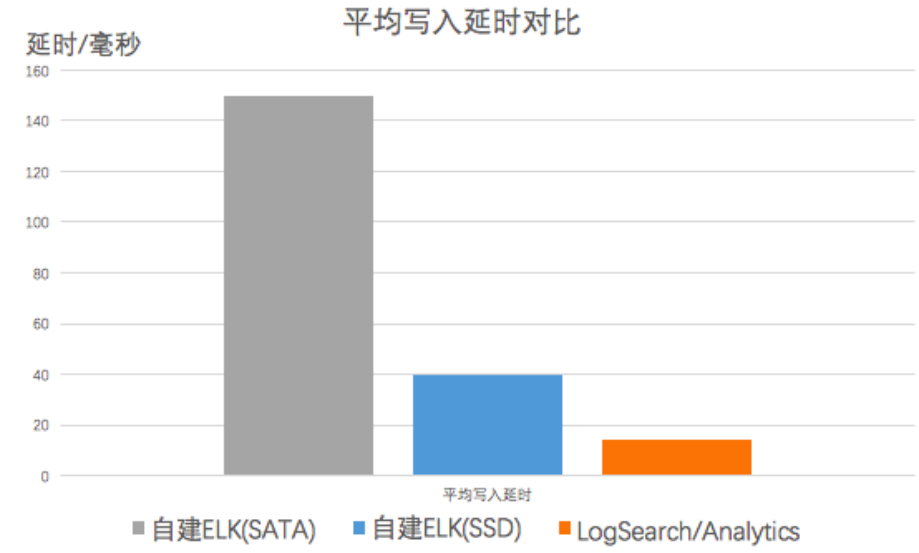
1. 数据集大小
   - 原始数据大小：50 GB
   - 原始数据除去Key后内容大小：27 GB（LogSearch/Analytics 以该大小作为存储计费单元）
   - 日志行数：162,640,232 (约为1.6亿条)

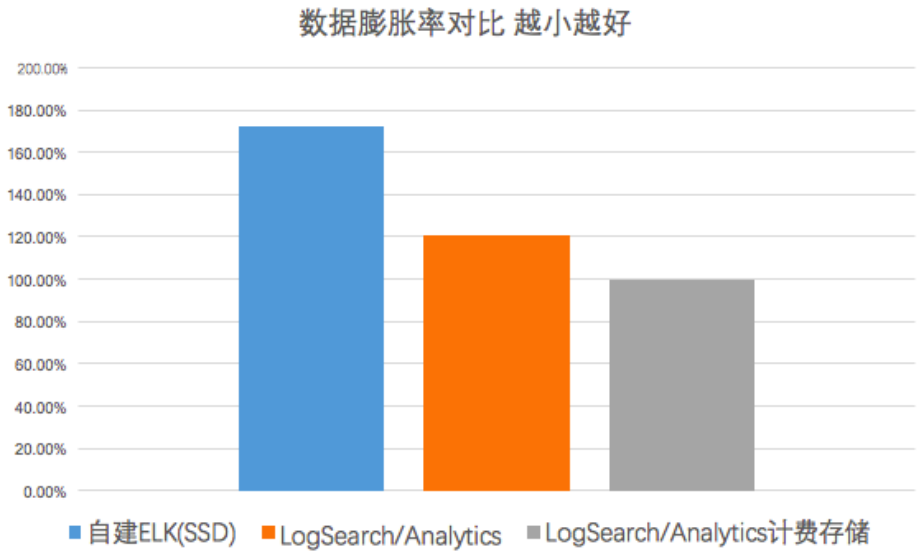
写入测试结果

ES采用bulk api批量写入，LogSearch/Analytics 用PostLogstoreLogs API批量写入，结果如下：

类型	项目	自建ELK	LogSearch/Analytics
延时	平均写入延时	40 ms	14 ms
存储	单拷贝数据量	86G	58G
	膨胀率：数据量/原始	172%	121%

	数据大小		





备注：LogSearch/Analytics 产生计费的存储量包括压缩的原始数据写入量(23G)+索引流量(27G)，共50G存储费用。

从测试结果来看

- LogSearch/Analytics写入延时好于ES，40ms vs 14 ms
- 空间：原始数据50G，因测试数据比较随机所以存储空间会有膨胀（大部分真实场景下，存储会因压缩后会比原始数据小）。ES胀到86G，膨胀率为172%，在存储空间超出LogSearch/Analytics 58%

读取（查询+分析）测试

测试场景

我们在此选取两种比较常见的场景：日志查询和聚合计算。分别统计并发度为1，5，10时，两种case的平均延时。

针对全量数据，对任意text列计算group by，计算5列数值的avg/min/max/sum/count，并按照count排序，取前1000个结果，例如：

```
select count(long_1) as pv,sum(long_2),min(long_3),max(long_4),sum(long_5)
group by text_1 order by pv desc limit 1000
```

针对全量数据，随机查询日志中的关键词，例如查询 “value_126” ，获取命中的日志数目与前100行，例如：

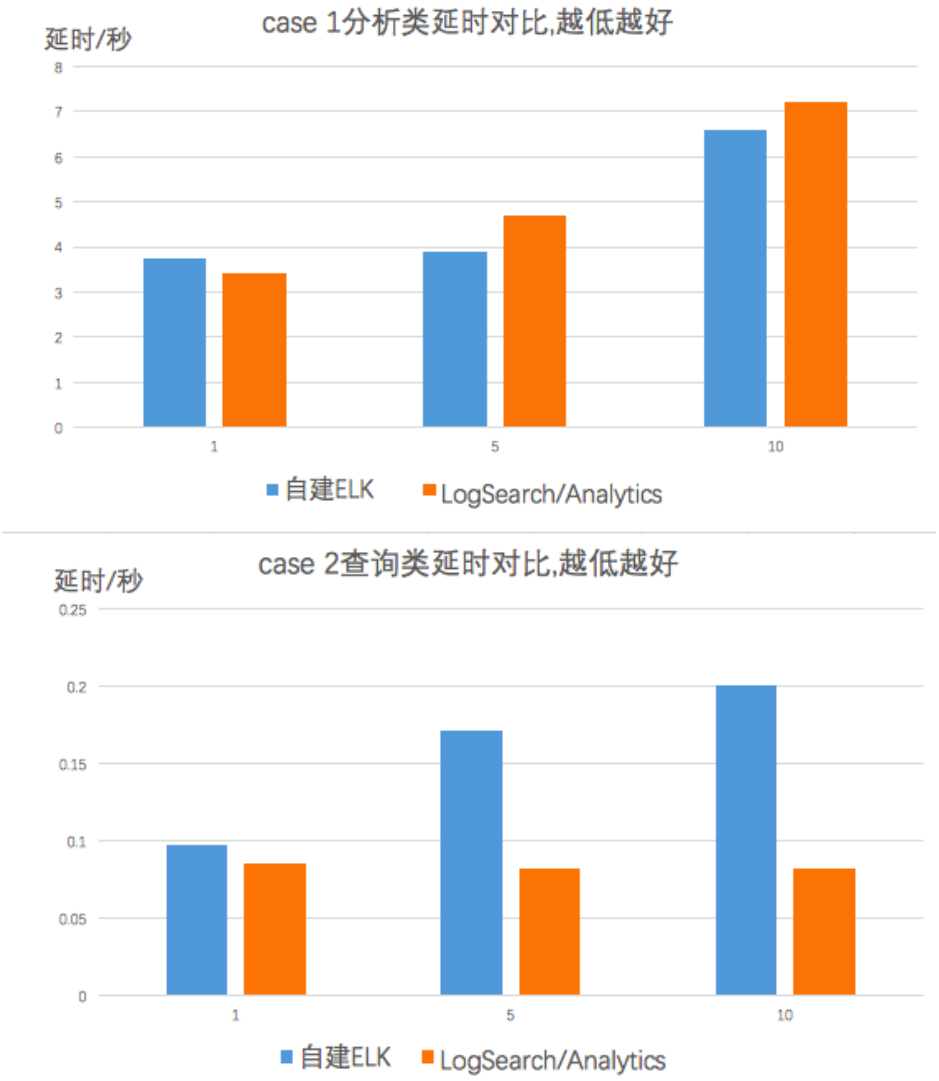
```
value_126
```

测试结果

类型	并发数	ES延时(单位s)	LogSearch/Analytics延时(单位s)
case1：分析类	1	3.76	3.4
	5	3.9	4.7
	10	6.6	7.2
case2：查询类	1	0.097	0.086
	5	0.171	0.083
	10	0.2	0.082

结果分析

- 从结果看，对于1.5亿数据量这个规模，两者都达到了秒级查询与分析能力
- 针对统计类场景（case 1）， ES和日志服务延时处同一量级。ES采用SSD硬盘，在读取大量数据时IO优势比较高
- 针对查询类场景（case 2）， LogAnalytics在延时明显优于ES。随着并发的增加，ELK延时对应增加，而LogAnalytics延时保持稳定甚至略有下降



关于LogSearch/Analytics 查询性能和规模可以参见查询分析Demo视频。

规模

LogSearch/Analytics 一天可以索引PB级数据，一次查询可以在秒级过几十TB规模数据，在数据规模上可以做到弹性伸缩与水平扩展

ES比较适合服务场景为：写入GB-TB/Day、存储在TB级。主要受限于2个原因：

- 单集群规模：比较理想为20台左右，业界比较大的为100节点一个集群
- 写入扩容：shard创建后便不可再修改，当吞吐率增加时，需要动态扩容节点，最多可使用的节点数便是shard的个数

  存储扩容：主shard达到磁盘的上线时，要么迁移到更大的一块磁盘上，要么只能分配更多的shard。一般做法是创建一个新的索引，指定更多shard，并且rebuild旧的数据

LogSearch/Analytics 则没有扩展性方面的问题，每个shard都是分布式存储。并且当吞吐率增加时，可以动态分裂shard，达到处理能力水平扩展。

成本

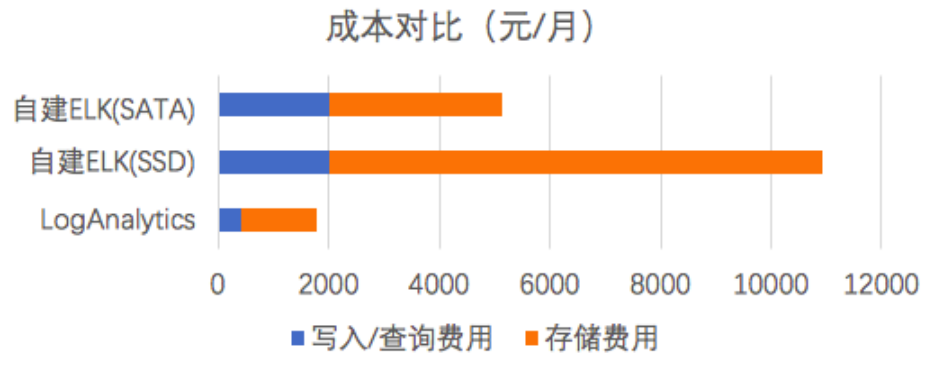
以上述测试数据为例，一天写入50GB数据（其中27GB 为实际的内容），保存90天，平均一个月的耗费。

1. 日志服务（LogSearch/LogAnalytics）计费规则参考文档，包括读写流量、索引流量、存储空间等计费项，查询功能免费。

计费项目	值	单价	费用(元)
读写流量	23G * 30	0.2 元/GB	138
存储空间（保存90天）	50G * 90	0.3 元/GB*Month	1350
索引流量	27G * 30	0.35 元/GB	283
总计			1771

1. ES费用包括机器费用，及存储数据SSD云盘费用
   - 云盘一般可以提供高可靠性，因此我们这里不计费副本存储量
   - 存储盘一般需要预留15%剩余空间，以防空间写满，因此乘以一个1.15系数

计费项目	值	单价	费用(元)
服务器	4台4核16G(三个月)(ecs.mn4.xlarge)	包年包月费用：675元/Month	2021
存储	86 *1.15* 90（这里只计算一个副本）	SSD：1 元/GB*M	8901
		SATA：0.35 元/GB*M	3115
总计			12943 (SSD)
			5135 (SATA)



同样性能，使用LogSearch/Analytics与ELK（SSD）费用比为 13.6%。在测试过程中，我们也尝试把SSD换成SATA以节省费用（LogAnalytics与SATA版费用比为 34%），但测试发现延时会从40ms上升至150ms，在长时间读写下，查询和读写延时变得很高，无法正常工作了。

总结

LogSearch/Analytics在提供同样查询速度、更高吞吐量、更强分析能力背后，只需要开源方案13%成本，并且按量付费免运维，让你把精力放在业务分析上。

日志服务除了LogSearch/Analytics外，还提供LogHub、LogShipper功能，支持实时采集数据、与流计算系统（Spark、Storm、Flink、StreamCompute）、离线分析系统（MaxCompute、E-MapReduce、Presto、Hive等）打通，提供一站式实时数据解决方案。

日志监控

日志服务支持通过以下方式实现日志监控：

- 通过云监控
    - 日志库（Logstore）级读写
    - 分区（Shard）级监控数据（即将推出）
    - Agent（Logtail）日志收集
- 控制台
    - 实时订阅消费（Spark Streaming、Storm、Consumer Library）当前点位
    - 日志投递状态

其中，云监控方式即通过阿里云云产品**云监控**完成，控制台方式监控日志请参见控制台 协同消费组进度查看、LogShipper 状态查看和设置报警规则。

云监控配置步骤

注意：使用子账号请参考 云监控授权文档。

登录 日志服务管理控制台。

选择所需的项目，单击项目名称。

选择所需的日志库并单击监控列下的图标打开云监控。

您也可以通过云监控管理控制台左侧导航栏中的**云服务监控** > **日志服务**进入监控配置页面。

在云监控中对日志数据进行监控，详细信息请参考云监控中的日志监控。

监控项含义

参考 日志服务监控项含义。

设置报警规则

参考 日志服务监控项含义。

指标含义

监控数据入口请参考LogHub监控章节。

1. 写入/读取流量
    - 含义：每个日志库（Logstore）写入、以及读取实时情况
    - 单位：Bytes/min
    - 含义：统计该Logstore通过ilogtail和SDK、API等读写实时流量，大小为传输大小（压缩情况下为压缩后），每分钟统计一个点，单位为字节/分钟。
2. 原始数据大小
    - 含义：每个Logstore写入数据原始大小（压缩前）
    - 单位：Byte/min
3. 总体QPS
    - 含义：所有操作QPS，每分钟统计一个点
    - 单位：Count/Min
4. 操作次数
    - 含义：统计用户的各种操作对应的QPS，每分钟统计一个点
    - 单位：次/分钟（Count/Min)
    - 所有的操作包括：
        - 写入操作：
            - PostLogStoreLogs ：0.5API以后版本接口。
            - PutData ： 0.4 API以前版本接口。
        - 根据关键字查询：
            - GetLogStoreHistogram: 查询关键字分布情况，0.5API以后版本接口。
            - GetLogStoreLogs: 查询关键字命中日志，0.5API以后版本接口。
            - GetDataMeta ： 同GetLogStoreHistogram，为0.4API以前版本接口。
            - GetData : 同GetLogStoreLogs，为0.4API以前版本接口。
        - 批量获取数据：
            - GetCursorOrData：该操作包含了获取Cursor和批量获取数据两种方

法。
- ListShards：获取一个Logstore下所有的Shard。
- List操作：
  - ListLogStoreLogs:遍历一个project下所有的Logstore。
  - ListCategory：同ListLogStoreLogs，为0.4API以前版本接口。
  - ListLogStoreTopics：遍历一个Logstore下所有的Topic。

5. 服务状态
   - 含义：该视图统计用户的各种操作返回的HTTP 状态码对应的QPS，方便用户根据错误的返回码来判断操作异常，及时调整程序。
   - 各状态码:
     - 200：为正常的返回码，表示操作成功。
     - 400：错误的参数，包括Host,Content-length,APIVersion,RequestTimeExpired,查询时间范围，Reverse，AcceptEncoding,AcceptContentType,Shard,Cursor，PostBody,Paramter,ContentType等方面的错误。
     - 401：鉴权失败，包括AccessKeyId不存在、签名不匹配、或者签名账户没有操作权限，请到SLSweb上查看project权限列表，是否包含了该AK。
     - 403: 超过预定Quota，包括能够创建的Logstore个数、Shard总数、以及读写操作的每分钟限额，请根据返回的Message判断发生了哪种错误。
     - 404：请求的资源不存在，包括Project、 Logstore、Topic 、User等资源。
     - 405：错误的操作方法，请检查请求的URL路径。
     - 500：服务端错误，请重试。
     - 502：服务端错误，请重试。
6. 客户端解析成功流量
   - 含义：Logtail收集成功的日志大小，为原始数据大小
   - 单位：字节
7. 客户端（Logtail）解析成功行数
   - 含义：Logtail收集成功的日志的行数
   - 单位: 行
8. 客户端解析失败行数
   - 含义：Logtail收集日志过程中，采集出错的行数大小，如果该视图有数据，表示有错误发生
   - 单位：行
9. 客户端错误次数
   - 含义：Logtail收集日志过程中，出现所有收集错误的IP总数
   - 单位：次
10. 发生客户端错误机器数
    - 含义：Logtail收集日志过程中，出现收集错误的告警次数
    - 单位：个
11. 错误IP统计（Count/5min)
    - 含义：分类别展示各种采集错误发生的IP数，各种错误包括：
      - LOGFILE_PERMINSSION_ALARM:没有权限打开日志文件。
      - SENDER_BUFFER_FULL_ALARM：数据采集速度超过了网络发送速度，数据被

丢弃。

- INOTIFY_DIR_NUM_LIMIT_ALARM（INOTIFY_DIR_QUOTA_ALARM）：监控的目录个数超过了3000个，请把监控的根目录设置成更低层目录。
- DISCARD_DATA_ALARM：数据丢失，因为数据时间在系统时间之前15分钟，请保证新写入日志文件的数据是在15分钟之内的。
- MULTI_CONFIG_MATCH_ALARM：有多个配置在收集同一个文件，logtail会随机选择一个配置进行收集，另一个配置则收集不到数据。
- REGISTER_INOTIFY_FAIL_ALARM：注册inotify事件失败，具体原因请查看logtail日志。
- LOGDIR_PERMINSSION_ALARM：没有权限打开监控目录。
- REGEX_MATCH_ALARM：正则式匹配错误，请调整正则式。
- ENCODING_CONVERT_ALARM：转换日志编码格式时出现错误，具体原因请查看logtail日志。
- PARSE_LOG_FAIL_ALARM：解析日志错误，一般是行首正则表达式错误或单条日志超过512KB导致的日志分行错误，请查看Logtail日志确定原因，如行首正则表达式错误请调整配置。
- DISCARD_DATA_ALARM：丢弃数据，Logtail发送数据到日志服务失败且写本地缓存文件失败导致，可能的原因是日志文件产生较快但写磁盘缓存文件较慢。
- SEND_DATA_FAIL_ALARM：解析完成的日志数据发送日志服务失败，请查看Logtail日志发送数据失败相关ErrorCode和ErrorMessage，常见的错误有服务端Quota超限、客户端网络异常等。
- PARSE_TIME_FAIL_ALARM：解析日志time字段出错，Logtail根据正则表达式解析出来的time字段按照时间格式配置无法解析成功，请修改配置。
- OUTDATED_LOG_ALARM：Logtail丢弃历史数据，请保证当前写入日志数据的时间与系统时间相差在5分钟以内。

- 请根据具体错误请找到出错IP，登录机器查看/usr/logtail/ilogtail.LOG查看错误原因

使用监控+报警

Logtail日志是否完整收集？

客户端（Logtail)在运行过程中，可能会因设置不正确产生错误：例如某些日志格式不匹配，一个日志文件被重复收集等（Logtail场景问题）。为了及时发现这种情况，我们可以对客户端解析失败行数、客户端错误次数等指标进行监控，以及时发现这类问题。步骤如下：

i. 打开LogHub云监控页面，参考LogHub监控章节。
ii. 选择关心的Logstore，新建报警规则：
    i. 选择”客户端解析错误”
    ii. 设置规则，当出错误后报警
    iii. 设置报警短信接收人进行报警

iii. 除此之外，还可以根据Logtail其他错误项进行报警，第一时间发现各类日志收集过程中发现的问题

Logstore下Shard资源是否足够

Logstore下每个Shard提供5MB/S (1000次/S) 写入能力，这个数值对于大部分用户而言都是足够的，在超过时日志服务会尽可能去服务（非拒绝）你的请求，但在高峰期间不保证超出部分可用性。如果你的日志量非常大，需要添加更多Shard，可以在控制台（日志消费/修改）中进行调整。在之前可以设置logstore出入流量报警以检测该情况。可以设置Logstore流量报警以检测该情况，步骤如下：

- 方案1：对流量预警
  a. 打开LogHub云监控页面，参考LogHub监控章节。
  b. 选择关心的Logstore，新建报警规则：
  - 选择”原始数据大小”
  - 设置规则，超过25GB/Min后进行报警
  - 设置报警短信接收人进行报警
- 方案2：设置状态码报警
  a. 打开LogHub云监控页面，参考LogHub监控章节。
  b. 选择关心的Logstore，新建报警规则：
  - 选择”服务状态”，status填写403（流量超过阈值）
  - 设置规则，超过25GB/Min后进行报警
  - 设置报警短信接收人进行报警

3. **Project Quota是否足够** 每个Project默认写入限制为30GB/分钟（原始数据大小），这个数值主要目的是为了保护用户因程序错误产生大量日志设计，在一般场景中对于大部分用户都是足够的。如果

你的日志量非常大，可能会超过限制，可以通过工单联系我们调整大这个数值。Logstore流量报警与预警步骤如上。

日志服务支持通过云监控设置报警，当服务状态符合设置的报警规则时发送报警短信或邮件。您可以通过配置云监控中的日志监控，对日志收集状态、Shard资源使用状态等异常状态进行监控。本文以监控日志收集状态和Shard资源使用状态为例，介绍使用云日志设置的报警规则。更多信息请参考云监控中的日志监控。

监控Logtail日志收集状态

Logtail客户端在运行过程中，可能会因设置不正确产生错误，例如某些日志格式不匹配、一个日志文件被重复收集等（Logtail场景问题）。为了及时发现这种情况，您可以对客户端解析失败行数、客户端错误次数等指标进行监控，以便及时发现这类问题。配置步骤如下。

打开LogHub云监控页面。详细步骤请参考LogHub监控章节。

选择目标Logstore，右上角单击**创建报警规则**。

关联资源。

根据您的实际情况填写**关联资源**。

设置报警规则。

设置**规则名称**，并选择**规则描述**。您可以根据需要选择**客户端解析失败行数**或**客户端错误次数**选项，并配置统计周期、统计方法等规则项。除此之外，还可以根据Logtail其他错误项进行报警，第一时间发现各类日志收集过程中发现的问题。

配置**通知方式**。

选择**通知对象**和**通知方式**。

监控Shard资源使用状态

Logstore下每个Shard提供5MB/s (1000次/s) 写入能力，这个数值对于大部分用户而言都是足够的，在超过时日志服务会尽可能去服务（非拒绝）您的请求，但在高峰期间不保证超出部分的可用性。您可以设置Logstore出入流量报警以检测该情况。如果您的日志量非常大，需要添加更多Shard，请及时在控制台中进行调整。设置Logstore流量报警步骤如下。

方案1：设置流量预警

1. 打开LogHub云监控页面。详细步骤请参考LogHub监控章节。

选择目标Logstore，单击**创建报警规则**。

关联资源。

根据您的实际情况填写**关联资源**。

设置报警规则。

设置**规则名称**，并配置**规则描述**为**原始数据大小**。您可以在此处设置统计周期和统计方法，如需超过25GB/Min后进行报警，请设置**5分钟**、**总计**、**>=**、**134217728000**，表示5分钟内总计流量超出134217728000 bytes 时进行报警。

配置**通知方式。**

选择**通知对象**和**通知方式。**

方案2：设置服务状态报警

1. 打开LogHub云监控页面。详细步骤请参考LogHub监控章节。

选择目标Logstore，单击**创建报警规则**。

关联资源。

根据您的实际情况填写**关联资源**。

设置报警规则。

设置**规则名称**，并配置**规则描述**为**服务状态**，**status**自定义设置为403。您可以在此处设置统计周期和统计方法，如您需要在5分钟内出现5个以上403服务状态时收到报警，请设置为**五分钟**、**采样计数值**、**>=**、和**5**。

配置**通知方式。**

选择**通知对象**和**通知方式。**

访问控制 RAM

RAM (Resource Access Management) 是阿里云为客户提供的 **用户身份管理** 与 **资源访问控制** 服务。使用

RAM，您可以创建、管理用户账号（比如员工、系统或应用程序），并可以控制这些用户账号对您名下资源具有的操作权限。当您的企业存在多用户协同操作资源时，使用 RAM 可以让您避免与其他用户共享云账号密钥，按需为用户分配最小权限，从而降低您的企业信息安全风险。

为了更精细地管理和操作日志服务资源，您可以通过阿里云RAM产品为您名下的子账号、日志服务的RAM服务角色和用户角色赋予相应的访问权限。

身份管理

您可以通过RAM进行用户身份管理。例如在您的账号下创建并管理用户账号/用户组、创建服务角色以代表日志服务、创建用户角色以进行跨账号的资源操作与授权管理。

资源访问控制

您可以为名下的用户账号/用户组以及角色授予对应的授权策略。

您也可以创建自定义授权策略，或者以自定义授权策略和系统授权策略为模板，编辑更细粒度的授权策略。

日志服务支持以下既定授权策略。

授权策略	类型	说明
AliyunLogFullAccess	系统策略	日志服务的全部管理权限。
AliyunLogReadOnlyAccess	系统策略	只读访问日志服务的权限。
AliyunLogRolePolicy	快捷授权策略	日志服务默认使用此角色来访问您在其他云产品中的资源，用于日志服务默认角色的授权策略，包含OSS的写入权限。快速授权请单击快速授权页。
AliyunLogETLRolePolicy	快捷授权策略	日志服务默认使用此角色来访问您在其他云产品中的资源，用于日志服务ETL功能角色的授权策略。快速授权请单击快速授权页。

应用场景

授权RAM子用户访问日志服务

在实际的应用场景中，主账号可能需要将日志服务的运营维护工作交予其名下的RAM子用户，由RAM子用户对日志服务进行日常维护工作；或者主账号名下的RAM子用户可能有访问日志服务资源的需求。此时，主账号需要对其名下的RAM子用户进行授权，授予其访问或者操作日志服务的权限。出于安全性的考虑，日志服务建议您将RAM子用户的权限设置为需求范围内的最小权限。

配置详情请参考授权RAM子用户访问日志服务。

授权服务角色读日志

日志服务目前提供基于用户日志内容报警功能，为了读取日志数据，需要用户显式授权日志服务服务账号访问用户数据。

配置详情请参考授权服务角色读日志。

授权用户角色操作日志服务

RAM 用户角色是一种虚拟用户，它没有确定的身份认证密钥，且需要被一个受信的实体用户（比如云账号、RAM-User 账号、云服务账号）扮演才能正常使用。扮演成功后实体用户将获得 RAM 用户角色的临时安全令牌，使用这个临时安全令牌就能以RAM服务角色身份访问被授权的资源。

> 将日志服务的操作权限授予一个受信实体用户，允许该实体用户下的RAM角色操作日志服务。配置详情请参考授权用户角色操作日志服务。
>
> 授权移动应用客户端通过直连方式访问日志服务，将APP的日志直接上传到日志服务中。配置详情请参考授权移动应用直连日志服务。

在实际的应用场景中，主账号可能需要将日志服务的运营维护工作交予其名下的RAM子用户，由RAM子用户对日志服务进行日常维护工作；或者主账号名下的RAM子用户可能有访问日志服务资源的需求。此时，主账号需要对其名下的RAM子用户进行授权，授予其访问或者操作日志服务的权限。出于安全性的考虑，日志服务建议您将RAM子用户的权限设置为需求范围内的最小权限。

主账号授权RAM子用户访问日志服务资源，需要按照以下步骤完成。关于RAM子用户的详细信息，请参考RAM 子帐号访问控制台。

步骤 1 创建RAM子用户

登录 访问控制服务控制台。

在**用户管理**页面单击页面右上角的**新建用户**。

填写用户信息，勾选 **为该用户自动生成AccessKey** 并单击 **确定**。

在弹出的验证对话框中，单击 **点击获取** 并输入手机验证码，然后单击 **确定**。您可以在用户列表中看到创建的用户。

单击创建的用户，跳转到用户详情，单击页面中的 **启用控制台登录**，配置用户登录密码并单击 **确定**。

步骤 2 授权子用户访问日志服务资源

日志服务提供两种系统授权策略，即AliyunLogFullAccess和AliyunLogReadOnlyAccess，分别表示管理权限和只读权限。您还可以在访问控制服务控制台可以自定义授权策略，创建方法参考**创建自定义授权策略**，权限策略示例请参考**日志服务RAM授权策略**。本文档以给子用户授予只读权限为例。

在 **用户管理** 页面找到对应的子用户，单击 **授权**。

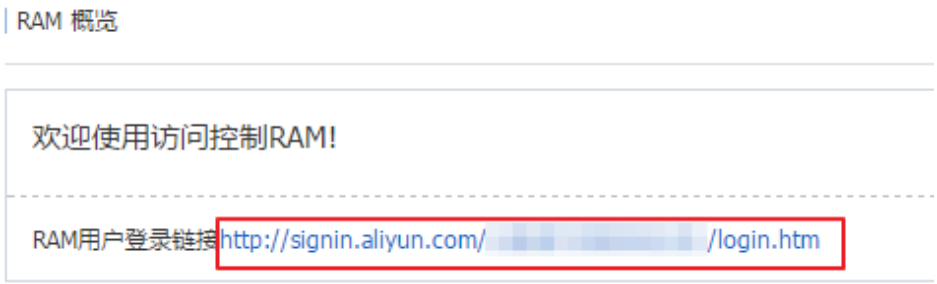
在弹出的对话框中，选择 **AliyunLogReadOnlyAccess**。

在弹出的验证对话框中，单击 **点击获取** 并输入手机验证码，然后单击 **确定**。

步骤 3 子用户登录控制台

完成创建子用户和子用户授权之后，子用户就有权限访问日志服务控制台了。您可以通过以下两种方式以RAM子用户身份登录控制台。

在访问控制服务控制台概览页面，单击**子用户登录**，使用步骤1中创建的子用户用户名和密码登录。



访问**子用户登录页**登录，也可以使用RAM控制台概览页的RAM用户登录链接登录。

其中的 **[企业号别名]** 默认为主账号的账号id（ali uid），可以在RAM控制台的**设置->账户别名设置**中查看并设置您的云帐号别名。

自定义授权示例

例如，需要赋予子账号以下权限：

1. 子账号可以看到主账号有哪些日志服务Project。
2. 子账号对主账号的某个日志服务Project有只读的访问权限。

同时满足1、2的授权策略如下：

```
{
"Version": "1",
"Statement": [
{
"Action": ["log:ListProject"],
"Resource": ["acs:log:*:*:project/*"],
"Effect": "Allow"
} ,
{
"Action": [
"log:Get*",
"log:List*"
],
"Resource": "acs:log:*:*:project/<指定的Proejct名称>/*",
"Effect": "Allow"
}
]
}
```

日志服务目前提供基于用户日志内容报警功能，为了读取日志数据，需要用户显式授权日志服务服务账号访问用户数据。如果已经阅读过该文档完成授权，可以略过以下内容直接创建报警规则，具体授权步骤如下具体说明。

创建访问控制（RAM）角色

1. 用户登录**访问控制控制台**，打开**角色管理**功能，单击右上角**新建角色**按钮，第一步角色类型选择**服务角色**。

2. 类型信息选择**LOG日志服务**。

3. 填写角色名称为**aliyunlogreadrole**（默认会扮演该角色访问数据，请不要修改名称），单击**创建**。

授权角色访问日志数据权限

角色创建完成后，在**角色管理**列表中，找到**aliyunlogreadrole**角色名称，点击**授权**，搜索**AliyunLogReadOnlyAccess**权限，直接应用。

完成如上步骤后，日志服务即有权限定期读取指定日志库数据进行报警检查。

角色，与 **用户**一样，都是 RAM 中使用的身份。与 RAM 用户相比，RAM 用户角色是一种虚拟用户，它没有确定的身份认证密钥，且需要被一个受信的实体用户（比如云账号、RAM-User 账号、云服务账号）扮演才能正常使用。扮演成功后实体用户将获得 RAM 用户角色的临时安全令牌，使用这个临时安全令牌就能以RAM服务角色身份访问被授权的资源。

如果您需要将日志服务的操作权限授予一个受信实体用户，允许该实体用户下的RAM角色操作日志服务，您需要创建RAM用户角色并指定受信云账号、为RAM用户角色授权、为受信账号下的RAM用户授予AssumeRole权限、获取RAM用户角色的临时安全令牌。

更多内容请参考RAM文档。

步骤1 创建用户角色并指定受信云账号

登录到 RAM 控制台，并在左侧导航栏单击 **角色管理**。

单击右上角 **新建角色**。

在**选择角色类型**子页，选择 **用户角色**。

在**填写类型信息**子页，选择 **受信云账号**。

注意：

- 若创建的角色是给您自己名下的 RAM 用户使用（比如授权移动 App 客户端直接操作LOG资源），请选择 **当前云账号** 为受信云账号。

- 若创建的角色是给其他云账号名下的 RAM 用户使用（比如跨账号的资源授权），请选择**其他云账号**，并在受信云账号 ID 中填写其他云账号的 ID。

在**配置角色基本信息**子页，输入 **角色名称** 和 **备注** 后，单击 **创建**。

步骤2 为RAM用户角色授权

成功创建用户角色后，该用户角色没有任何权限，您需要为RAM用户角色授予操作日志服务的权限。您上一步中指定的受信云账号将有权限扮演该RAM用户角色操作日志服务。

注意：您可以赋予RAM用户角色一个或多个授权策略，包括系统授权策略和自定义授权策略。本文档以授予RAM用户角色管理日志服务的权限为例。

在RAM控制台左侧导航栏单击 **角色管理**。

单击目标RAM用户角色名称右侧的**授权**。

选择**AliyunLogFullAccess**权限，并单击**确定**。

单击**点击获取**以获取验证码，输入收到的验证码，单击**确认**。

更多内容请参考 授权。

步骤3 为受信云账号的RAM用户授权

RAM用户角色需要被一个受信的实体用户扮演才能正常使用，但是受信实体用户不能以自己的身份扮演RAM用户角色，必须以RAM用户的身份和形式扮演。即**RAM 用户角色只能通过 RAM 用户身份来扮演使用。**

另外，受信云账号必须为其名下的RAM用户进行**AssumeRole**授权，授予该RAM用户调用STS服务

AssumeRole接口的权限，此RAM用户才能代表受信云账号扮演步骤1中创建的RAM用户角色。

登录受信云账号的RAM控制台。

在**用户管理**页面中，选定用于授权的RAM用户，并单击右侧的**授权**。

如果您之前没有创建过RAM用户，请参考RAM文档创建一个RAM用户。

选择系统授权策略**AliyunSTSAssumeRoleAccess**并单击**确定**。

单击**点击获取**以获取验证码，输入收到的验证码，单击**确认**。

步骤4 获取RAM用户角色的临时安全令牌

当 RAM 用户被授予 AssumeRole 权限之后，可以使用其 AccessKey 调用安全令牌服务(STS) 的 AssumeRole 接口，以获取某个角色的临时安全令牌。

关于 AssumeRole API 的调用方法，请参考 STS API文档。

使用STS SDK拿到AccessKeyId、AccessKeySecret 、SecurityToken 之后就可以使用日志服务的SDK访问日志服务了。

下面是使用AccessKeyId、AccessKeySecret 、SecurityToken初始化LogClient的示例，Java SDK使用请参考文档。

```
package sdksample;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;
import java.util.Date;
import com.aliyun.openservices.log.Client;
import com.aliyun.openservices.log.common.*;
import com.aliyun.openservices.log.exception.*;
import com.aliyun.openservices.log.request.*;
import com.aliyun.openservices.log.response.*;
import com.aliyun.openservices.log.common.LogGroupData;
import com.aliyun.openservices.log.common.LogItem;
import com.aliyun.openservices.log.common.Logs.Log;
import com.aliyun.openservices.log.common.Logs.Log.Content;
import com.aliyun.openservices.log.common.Logs.LogGroup;
import com.aliyun.openservices.log.common.Consts.CursorMode;
public class sdksample {
public static void main(String args[]) throws LogException, InterruptedException {
String endpoint = "<log_service_endpoint>"; // 选择与上面步骤创建 Project 所属区域匹配的Endpoint
String accessKeyId = "<your_access_key_id>"; // 使用您的阿里云访问密钥 AccessKeyId
String accessKeySecret = "<your_access_key_secret>"; // 使用您的阿里云访问密钥AccessKeySecret
String securityToken = "<your_security_token>"; //角色的SecurityToken
String project = "<project_name>"; // 上面步骤创建的项目名称
```

```
String logstore = "<logstore_name>"; // 上面步骤创建的日志库名称
// 构建一个客户端实例
Client client = new Client(endpoint, accessKeyId, accessKeySecret);
// 设置SecurityToken
client.SetSecurityToken(securityToken);
// 写入日志
String topic = "";
String source = "";
// 连续发送 10 个数据包，每个数据包有 10 条日志
for (int i = 0; i < 10; i++) {
Vector<LogItem> logGroup = new Vector<LogItem>();
for (int j = 0; j < 10; j++) {
LogItem logItem = new LogItem((int) (new Date().getTime() / 1000));
logItem.PushBack("index"+String.valueOf(j), String.valueOf(i * 10 + j));
logGroup.add(logItem);
}
PutLogsRequest req2 = new PutLogsRequest(project, logstore, topic, source, logGroup);
client.PutLogs(req2);
}
}
}
```