日志服务

用户指南

用户指南

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项目。

前提条件

开始使用日志服务之前，您需要开通日志服务，并且创建Access Key。

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，系统会及时提醒。
- 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10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、数字和短横线（-），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区域（Region），且创建后就不能修改区域，也不能在多个区域间迁移项目。

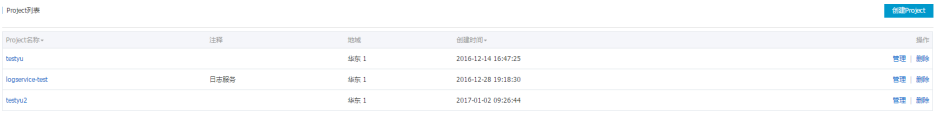
后续操作

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**。

您可以查看您的日志服务项目列表。

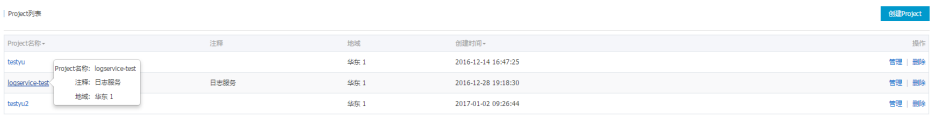
登录 **日志服务管理控制台**。**Project列表** 页面列出了当前阿里云账号下的所有项目，如下图所示。



您可以通过项目列表操作相应的项目，具体如下：

- 管理项目：点击项目名称或者项目列表右边的 **管理** 按钮都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。

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为日志服务的资源管理单元，您可以通过Project来管理自己的各种日志库，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，您可以如下管理日志服务的Project：

创建和删除Project内的日志库（Logstore）。日志库为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收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 管理日志库。

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，其中最为常见的即为服务器。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，同样机器IP把需要采集的服务器归类到相应的机器组中。一种常见的实践就是让需要写入一个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。

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**。

在弹出的对话框中，单击 **获取验证码**，输入您收到的校验码并单击 **确认**。

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，单击日志消费列下的 **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

创建完项目后，系统会提示您创建日志库，单击 **创建**。

您也可以在 **Project列表** 页面，单击项目的名称或者选择所需的项目并单击右侧的 **管理**，然后单击 **创建** 创建日志库，详细信息参见 创建日志库。

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、数字和短横线（-），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。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1~90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，可以设置为1~10。

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**。

您可以看到所选项目下的日志库。

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，您需要进行该项配置，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 ODPS **管理** 或 OSS **管理** 开启和设置 ODPS 或 OSS 日志投递。更多信息，请参考 **投递日志到 ODPS** 和 **投递日志到OSS**。
- 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，单击项目名称或者右侧的 **管理**。

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**。

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。

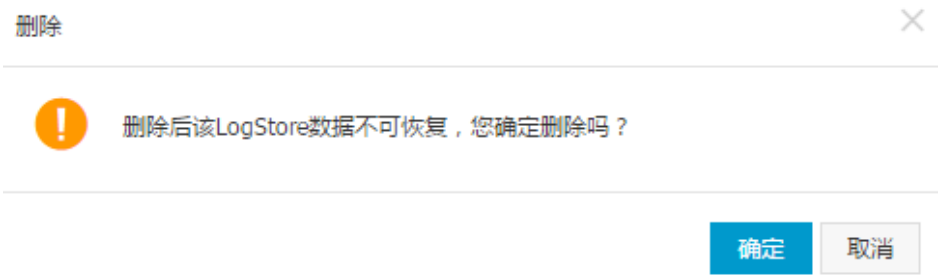
操作步骤

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。

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。

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

建议您根据数据流量合理规划分区个数。

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。

状态

分区的状态包括：

- readwrite：可以读写
- readonly：只读数据

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日志消费列下的 **修改**。

选择要分裂的分区，单击右侧的 **分裂**。


修改LogStore属性
* LogStore名称: test3
LogStore属性
* 数据保存时间: 1
修改
数据保存时间目前支持1～90天
* 计费: 单个Shard 0.04元/天（其它计费项）
* Shard管理:
ID 状态 Beginkey/EndKey 操作
0 readwrite 00000000000000000000000000000000 80000000000000000000000000000000 分裂| 合并
1 readwrite 80000000000000000000000000000000 ffffffffffffffffffffffffffffffff 分裂
1. readonly状态的shard不会产生费用，过期会自动删除
取消

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日志消费列下的 **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1-90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 支持通过Agent、Tracking功能和SDK/API等方式采集客户端、网页、syslog等多种来源的日志：

1. 使用 Logtail、Logstash等Agent采集日志，通过简单的配置就可以收集服务器上的日志，而且不用修改任何应用程序代码。
2. 通过 Tracking 功能采集日志。 支持 HTML、H5、iOS 和 Android 平台数据的采集。如需使用 Tracking 方式采集日志，您需要先开通 Logstore 上的 Tracking 功能。开通方法请参考 Web

Tracking 接入用户端日志。
3. 如果希望编程写入日志，Loghub 提供多种语言（Java/.NET/PHP/Python）的 SDK 方便您使用。
4. 如果需要通过其他语言代码写入日志，可以使用日志服务的 Rest 风格 API 来完成。

注意：无论通过何种方式写入日志，服务端可以接受的日志字段中时间为以当前基点 [-7 天~15 分钟] 范围内数据。否则，该日志数据将会被直接拒绝。具体请参考 PostLogStoreLogs 接口。

服务器日志采集方式

Agent（Logtail）

日志服务 logtail 接入服务的客户端使用正则表达式从用户原始数据提取信息，组织成为符合日志服务日志数据模型的结构。

- 单行日志
- Json 格式
- Delimiter 格式
- 通用日志格式：参见正则表达式

以下是常见配置示例： Nginx， Apache， Log4J ，Wordpress ， Python， NodeJS， 分隔符 (Delimiter, 如 CSV、TSV 等格式)、logstash ， ThinkPHP， IIS（Windows 平台，默认格式）/logstash 。有其他需求欢迎和我们反馈（support-log@list.alibaba-inc.com）

Agent（Logstash）

- logstash

面向容器（Docker）

- 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（Sample）
    - 这是集成后Docker完整代码，细节参见Docker/Deamon/Driver下Alilog部分
    - 这是Driver相关Commit

通过 SDK/API

- API
- Java
    - loghub Producer Library: 客户端高并发写入
    - Log4J Appender：基于 Producer Library 包装的 Log4J Appender
- Python
- PHP
- C#
- Go

- Native C
- NodeJs：联系我们
- C++: 联系我们

面向云产品

- 云主机（ECS）日志：使用 logtail 写入日志
- 容器服务（Container Service）日志：集成日志服务
- 对象存储（OSS）日志：使用 Python 工具导入 OSS 日志
- 消息服务（MNS）日志：消息服务日志管理功能

面向设备

- 移动端: IOS, Android
- 设备与协议: syslog, Native C
- 网页端: Web Tracking
- 游戏端: Unity3D SDK

网络

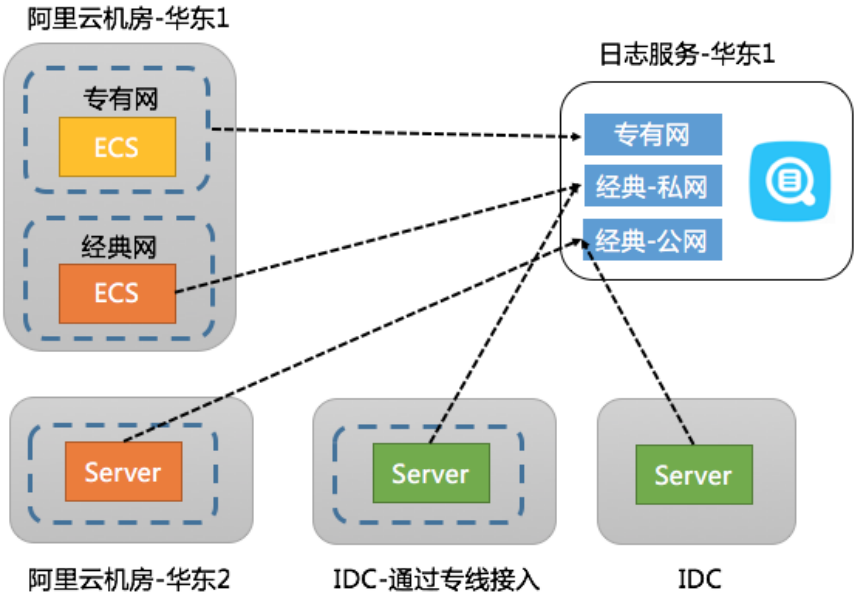
日志服务（LOG）提供各种内网/私网/公网等访问点，非常便捷地将混合网络环境中数据接入并融合。

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）

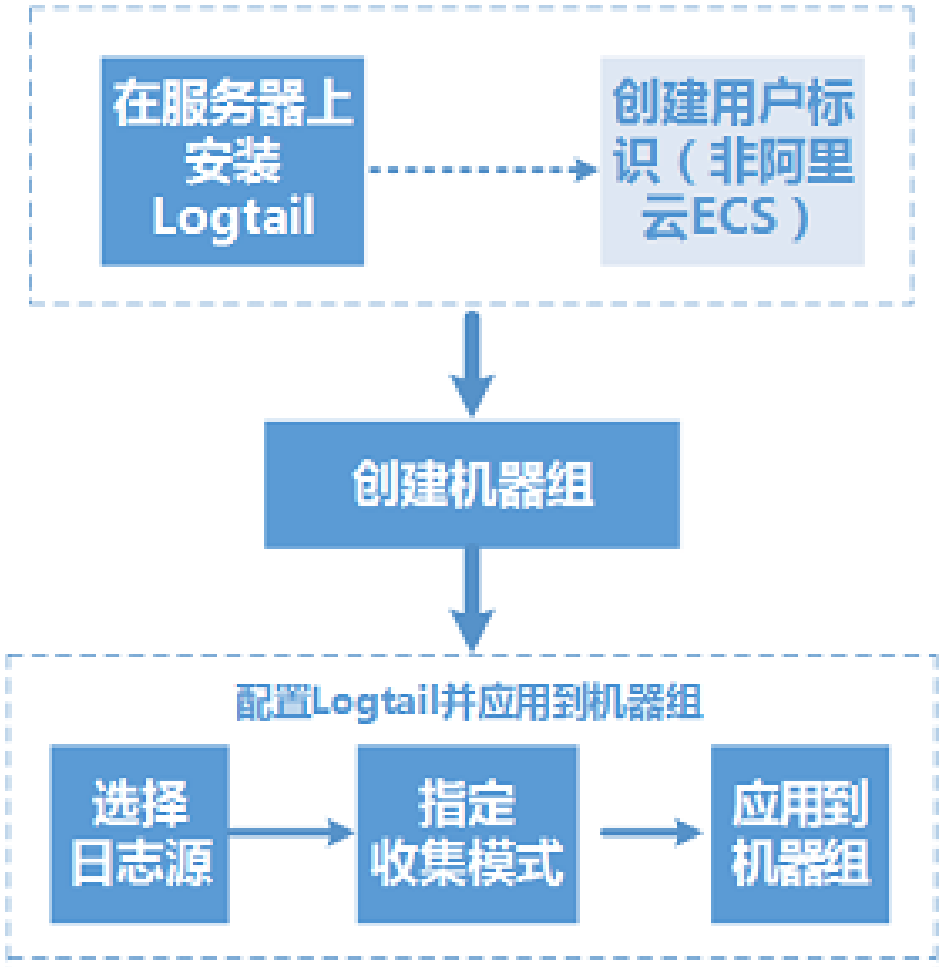
接入参考

- 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。

使用 logtail 采集日志

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。

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，既可精确匹配，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，日志服务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。

支持的系统

- 支持如下版本的Linux x86-64（64位）服务器
    - Aliyun Linux
    - Ubuntu
    - Debian
    - CentOS
    - OpenSUSE
- 阿里巴巴集团内部用户不要使用本文安装方式，请参考内部文档安装Logtail。

安装Logtail

覆盖安装模式，如之前已安装过Logtail，那么安装器会先执行卸载、删除 /usr/local/ilogtail 目录后再重新安装。安装后默认启动Logtail。

请根据机器所处的网络环境和日志服务所在Region下载安装器，选择不同参数进行安装。

ECS经典网络

数据通过阿里云内网写入日志服务，不消耗公网带宽。适用于阿里云ECS。

注：新开节点（海外、张北等）默认新开通ECS都是VPC网络，请查看VPC部分介绍。

- 华北 2（北京）

```
wget http://logtail-release-bj.oss-cn-beijing-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install cn_beijing
```

- 华北 1（青岛）

```
wget http://logtail-release-qd.oss-cn-qingdao-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install cn_qingdao
```

- 华东 1（杭州）

```
wget http://logtail-release.oss-cn-hangzhou-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install cn_hangzhou
```

- 华东 2（上海）

```
wget http://logtail-release-sh.oss-cn-shanghai-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install cn_shanghai
```

- 华南 1（深圳）

```
wget http://logtail-release-sz.oss-cn-shenzhen-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install cn_shenzhen
```

- 华北 3（张家口）

```
wget http://logtail-release-zjk.oss-cn-zhangjiakou-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install cn-zhangjiakou
```

- 香港

```
wget http://logtail-release-hk.oss-cn-hongkong-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install cn-hongkong
```

- 美国西部 1（硅谷）

```
wget http://logtail-release-us-west-1.oss-us-west-1-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install us-west-1
```

- 亚太东南 1（新加坡）

```
wget http://logtail-release-ap-southeast-1.oss-ap-southeast-1-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install ap-southeast-1
```

- 亚太东南 2（悉尼）

```
wget http://logtail-release-ap-southeast-2.oss-ap-southeast-2-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install ap-southeast-2
```

- 亚太东北 1 （日本）

```
wget http://logtail-release-ap-northeast-1.oss-ap-northeast-1-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install ap-northeast-1
```

- 欧洲中部 1 （法兰克福）

```
wget http://logtail-release-eu-central-1.oss-eu-central-1-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install eu-central-1
```

- 中东东部 1 （迪拜）

```
wget http://logtail-release-me-east-1.oss-me-east-1-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install me-east-1
```

ECS 专有网络 VPC

数据通过阿里云专有网络写入日志服务。适用于阿里云 ECS VPC。

- 华北 2（北京）

```
wget http://logtail-release-bj.vpc100-oss-cn-beijing.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install cn_beijing_vpc
```

- 华东 1（杭州）

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

- 华东 2（上海）

```
wget http://logtail-release-sh.vpc100-oss-cn-shanghai.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install cn_shanghai_vpc
```

- 华南 1（深圳）

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

- 华北 3 （张家口）

```
wget http://logtail-release-zjk.oss-cn-zhangjiakou-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install cn-zhangjiakou_vpc
```

- 香港

```
wget http://logtail-release-hk.vpc100-oss-cn-hongkong.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install cn-hongkong_vpc
```

- 美国西部 1 （硅谷）

```
wget http://logtail-release-us-west-1.vpc100-oss-us-west-1.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install us-west-1_vpc
```

- 亚太东南 1 （新加坡）

```
wget http://logtail-release-ap-southeast-1.vpc100-oss-ap-southeast-1.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install ap-southeast-1_vpc
```

- 亚太东南 2 （悉尼）

```
wget http://logtail-release-ap-southeast-2.oss-ap-southeast-2-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install ap-southeast-2_vpc
```

- 亚太东北 1 （日本）

```
wget http://logtail-release-ap-northeast-1.oss-ap-northeast-1-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install ap-northeast-1_vpc
```

- 欧洲中部 1 （法兰克福）

```
wget http://logtail-release-eu-central-1.oss-eu-central-1-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install eu-central-1_vpc
```

- 中东东部 1 （迪拜）

```
wget http://logtail-release-me-east-1.oss-me-east-1-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install me-east-1_vpc
```

公网（自建 IDC 或其它云主机）

数据通过公网写入日志服务，占用公网带宽。适用于非阿里云虚拟机或其它IDC。

日志服务无法获取非阿里云机器的属主信息，请在安装Logtail后手动配置用户标识，参见 **非阿里云ECS配置用户标识**，否则 Logtail心跳异常、无法收集日志。

- 华北 2（北京）

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

- 华北 1（青岛）

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

- 华东 1（杭州）

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

- 华东 2（上海）

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

- 华南 1（深圳）

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

- 华北 3 （张家口）

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

- 香港

```
wget http://logtail-release-hk.oss-cn-hongkong.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install cn-hongkong_internet
```

- 美国西部 1 （硅谷）

```
wget http://logtail-release.oss-cn-hangzhou.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install us-west-1_internet
```

- 亚太东南 1 （新加坡）

```
wget http://logtail-release.oss-cn-hangzhou.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install ap-southeast-1_internet
```

- 亚太东南 2 （悉尼）

```
wget http://logtail-release.oss-cn-hangzhou.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install ap-southeast-2_internet
```

- 亚太东北 1 （日本）

```
wget http://logtail-release.oss-cn-hangzhou.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install ap-northeast-1_internet
```

- 欧洲中部 1 （法兰克福）

```
wget http://logtail-release.oss-cn-hangzhou.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install eu-central-1_internet
```

- 中东东部 1 （迪拜）

```
wget http://logtail-release.oss-cn-hangzhou.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install me-east-1_internet
```

ECS 金融云

- 华东 1（杭州）

```
wget http://logtail-release-hz-finance.oss-cn-hzjbp-a-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install cn_hangzhou_finance
```

- 华南 1（深圳）

```
wget http://logtail-release-sz-finance.oss-cn-shenzhen-finance-1-internal.aliyuncs.com/linux64/logtail.sh; chmod 755 logtail.sh; sh logtail.sh install cn_shenzhen_finance
```

卸载Logtail

参考 **安装Logtail** 下载安装器 **logtail.sh**，shell下以管理员权限执行：

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

```
uninstall
```

支持系统

支持Windows Server2003（含）以后 32/64 位系统

- Windows 7 (Client) 32bit
- Windows 7 (Client) 64bit
- Windows Server 2003 32bit
- Windows Server 2003 64bit
- Windows Server 2008 32bit
- Windows Server 2008 64bit
- Windows Server 2012 32bit
- Windows Server 2012 64bit

安装Logtail

下载安装包

http://logtail-release.oss-cn-hangzhou.aliyuncs.com/win/logtail_installer.zip

按机器网络环境和日志服务所在Region进行安装

解压缩 logtail.zip 到当前目录，打开Windows Powershell或cmd进入 logtail_installer 目录。

日志服务 Region	机器网络环境	安装命令
华北 2（北京）	ECS 经典网络	.\logtail_installer.exe install cn_beijing
	ECS 专有网络 VPC	.\logtail_installer.exe install cn_beijing_vpc
	公网（自建 IDC 或其它云主机）	.\logtail_installer.exe install cn_beijing_internet
华北 1（青岛）	ECS 经典网络	.\logtail_installer.exe install cn_qingdao
	公网（自建 IDC 或其它云主机）	.\logtail_installer.exe install cn_qingdao_internet
华东 1（杭州）	ECS 经典网络	.\logtail_installer.exe install cn_hangzhou
	ECS 专有网络 VPC	.\logtail_installer.exe install cn_hangzhou_vpc
	公网（自建 IDC 或其它云主机）	.\logtail_installer.exe install cn_hangzhou_internet

	ECS 金融云	.\logtail_installer.exe install cn_hangzhou_finance
华东 2（上海）	ECS 经典网络	.\logtail_installer.exe install cn_shanghai
	ECS 专有网络 VPC	.\logtail_installer.exe install cn_shanghai_vpc
	公网（自建 IDC 或其它云主机）	.\logtail_installer.exe install cn_shanghai_internet
华南 1（深圳）	ECS 经典网络	.\logtail_installer.exe install cn_shenzhen
	ECS 专有网络 VPC	.\logtail_installer.exe install cn_shenzhen_vpc
	公网（自建 IDC 或其它云主机）	.\logtail_installer.exe install cn_shenzhen_internet
	ECS 金融云	.\logtail_installer.exe install cn_shenzhen_finance
华北 3（张家口）	ECS 经典网络	.\logtail_installer.exe install cn-zhangjiakou
	ECS 专有网络 VPC	.\logtail_installer.exe install cn-zhangjiakou_vpc
	公网（自建 IDC 或其它云主机）	.\logtail_installer.exe install cn-zhangjiakou_internet
香港	ECS 经典网络	.\logtail_installer.exe install cn-hongkong
	ECS 专有网络 VPC	.\logtail_installer.exe install cn-hongkong_vpc
	公网（自建 IDC 或其它云主机）	.\logtail_installer.exe install cn-hongkong_internet
美国西部 1（硅谷）	ECS 经典网络	.\logtail_installer.exe install us-west-1
	ECS 专有网络 VPC	.\logtail_installer.exe install us-west-1_vpc
	公网（自建 IDC 或其它云主机）	.\logtail_installer.exe install us-west-1_internet
亚太东南 1（新加坡）	ECS 经典网络	.\logtail_installer.exe install ap-southeast-1
	ECS 专有网络 VPC	.\logtail_installer.exe install ap-southeast-1_vpc
	公网（自建 IDC 或其它云主机）	.\logtail_installer.exe install ap-southeast-1_internet
亚太东南 2（悉尼）	ECS 经典网络	.\logtail_installer.exe install ap-southeast-2

	ECS 专有网络 VPC	.\logtail_installer.exe install ap-southeast-2_vpc
	公网（自建 IDC 或其它云主机）	.\logtail_installer.exe install ap-southeast-2_internet
亚太东北 1（日本）	ECS 经典网络	.\logtail_installer.exe install ap-northeast-1
	ECS 专有网络 VPC	.\logtail_installer.exe install ap-northeast-1_vpc
	公网（自建 IDC 或其它云主机）	.\logtail_installer.exe install ap-northeast-1_internet
欧洲中部 1（法兰克福）	ECS 经典网络	.\logtail_installer.exe install eu-central-1
	ECS 专有网络 VPC	.\logtail_installer.exe install eu-central-1_vpc
	公网（自建 IDC 或其它云主机）	.\logtail_installer.exe install eu-central-1_internet
中东东部 1（迪拜）	ECS 经典网络	.\logtail_installer.exe install me-east-1
	ECS 专有网络 VPC	.\logtail_installer.exe install me-east-1_vpc
	公网（自建 IDC 或其它云主机）	.\logtail_installer.exe install me-east-1_internet

注意：在自建 IDC 或其它云主机使用Logtail 时，由于日志服务无法获取非阿里云机器的属主信息，请在安装Logtail 后手动配置用户标识，参见 **非阿里云 ECS 配置用户标识**，否则Logtail心跳异常、无法收集日志。

卸载Logtail

打开 Windows Powershell或cmd进入 logtail_installer 目录，执行命令：

```
.\logtail_installer.exe uninstall
```

如果需要在非阿里云ECS上安装Logtail，请按如下步骤配置用户标识（账号 ID），否则心跳异常、无法收集数据到日志服务。

步骤 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。

步骤 2 登录机器配置账号ID标识文件

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：

Linux

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

当不需要Logtail收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

原理同Linux，创建、删除用户标识同名文件到目录 C:\LogtailData\users。

例如，C:\LogtailData\users\1559122535028493。

注意：

- 机器上配置账号ID标识后，标识该云账号有权限通过Logtail收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。

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。此外，您也可以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。

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，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、分字符（ - ）和下划线（ _ ），且必须以小写字母和数字开头和结尾，长度限制为 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陆云账号所有。
- 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**。

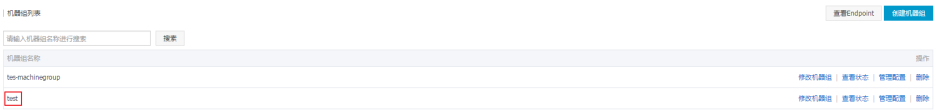
当模块需要扩容机器时，比如前端模块增加服务器，只需要在新增服务器上安装Logtail和创建自定义标识为http_module的文件即可自动同步不同机器组的配置，成功执行操作后可以在机器**查看状态**中看到新增机器。

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。

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作为标识，在自定义网络环境下（如VPC）可能出现不同机器IP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，一分钟后生效。

> **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，一分钟后生效。

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，找到LogtailWorker服务，右键重新启动生效。

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） 和 安装 logtail（Linux）。

使用限制

- 一个文件只能被一个配置收集。
- Logtail配置适合于所有支持的 Linux 64 位发行版本。具体可参考 使用 Logtail 写入日志。

操作步骤

在日志服务管理控制台首页点击目标日志库，进入**LogStore列表**，并点击**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、短横线（-）和下划线（_），且必须以小写字母和数字开头和结尾，长度限制为3~63字节。

注意：配置名称设置后不可修改。

指定日志的目录结构。

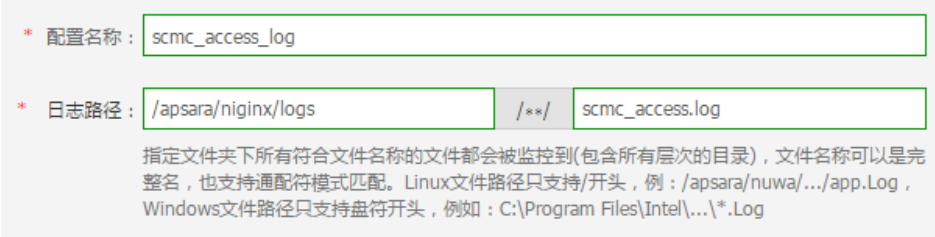
其中，文件夹支持完整路径和通配符两种模式，例如/apsara/*/log 则匹配apsara目录下所有二级目录下三级目录为log的目录。

注意： 目录通配符只支持单层目录匹配，例如/apsara/*/log可匹配/apsara/nuwa/log但不匹配/apsara/nuwa/data/log

文件名称可以是完整文件名，同时也支持通配符模式匹配。例如，/apsara/nuwa/.../*.Log可匹配/apsara/nuwa路径下（包括该目录下所有子目录）所有后缀名为.log的文件

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）。

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即会按照每行一条日志进行收集，并且不会对日志内容中字段进行提取，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。

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。

设置 **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（具体来说，就是正则表达式）。

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，告知需要提取的字段，日志服务控制台会自动生成正则表达式，如下图所示：

尽管第一种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最优化的，所以日志服务控制台仍然提供您手动直接输入正则的途径。您可以单击 **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，您需要给每个提取字段命名（设定对于字段的 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。

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：

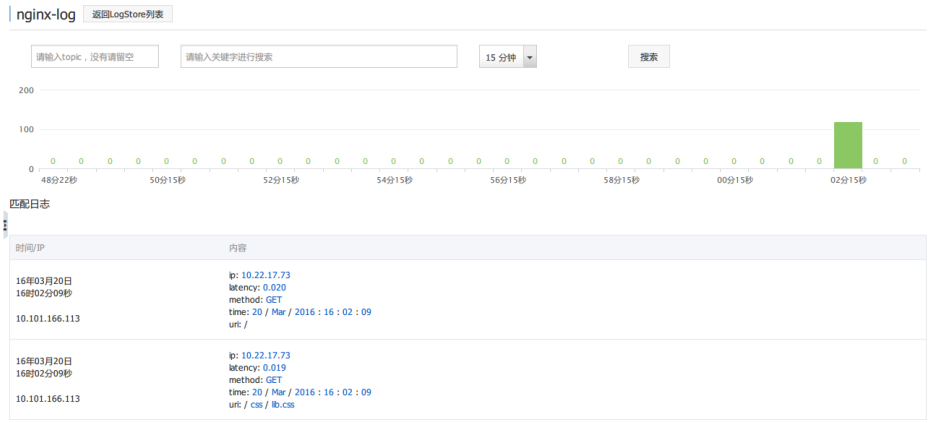
- Logtail配置推送生效时间最长需要 3 分钟，请耐心等待。
- 如果需要收集 IIS 的访问日志，请务必首先参考 **IIS 日志收集最佳实践** 配置 IIS。
- 创建Logtail配置后，您可以查看Logtail配置列表，修改Logtail配置或删除Logtail配置。详细信息，参见 **logtail 配置**。

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。



更多信息

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	指定收集日志文件所在的根目录，需要指定绝对路径，且不支持通配符
日志文件名	指定收集日志文件名称，大小写敏感，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	标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。具体可参考样例
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对 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

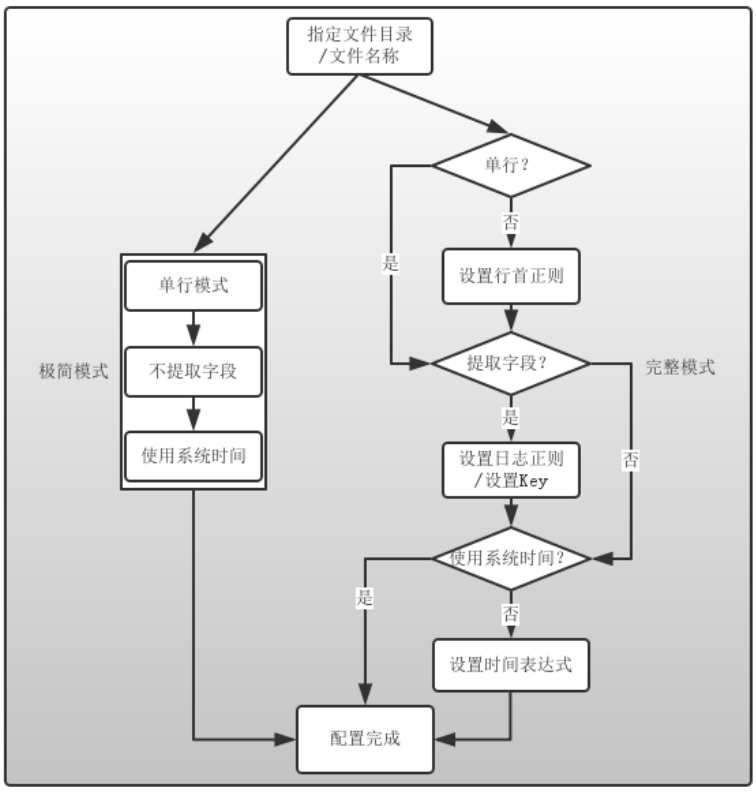
```
时间：13/Mar/2016:10:00:10
时间表达式：%d/%b/%Y:%H:%M:%S
```

抓取日志时的系统时间

时间：抓取日志时的时间戳

控制台配置文件日志收集模式

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：



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

日志库创建完成后，系统会提示您创建Logtail配置收集日志数据，在弹出的对话框中单击 **创建Logtail配置**。

您也可以通过 **Logstore列表** 页面的Logtail配置 **管理** 菜单创建 Logtail配置。

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、短横线（-）和下划线（_），且必须以小写字母和数字开头和结尾，长度限制为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。

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表示打开状态。

```
"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。详情参见xxxxxxx

```
"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
```

配置完之后重启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。

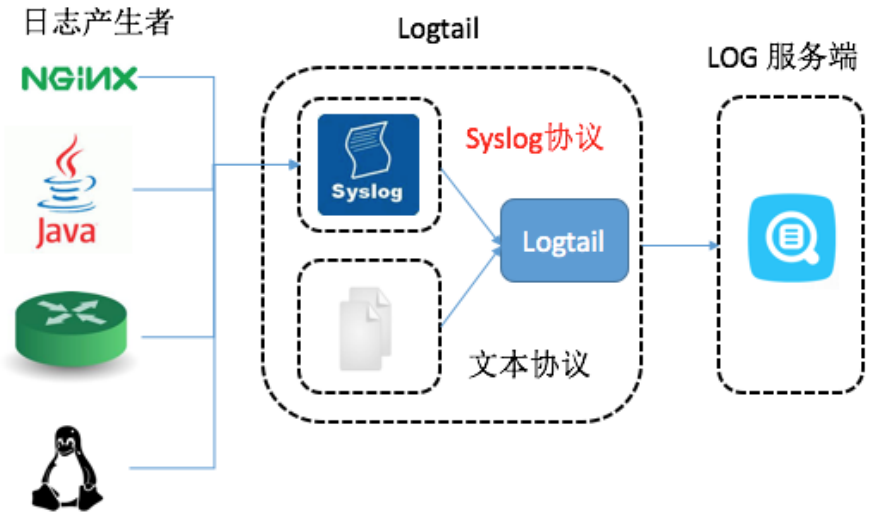
接着启动rsyslog（sudo /etc/init.d/rsyslog restart）。启动之前请先检查机器上是否安装了其他syslog的Agent，比如 syslogd、sysklogd、syslog-ng 等，如果有的话请关闭掉。

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目前支持的接入端主要有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。

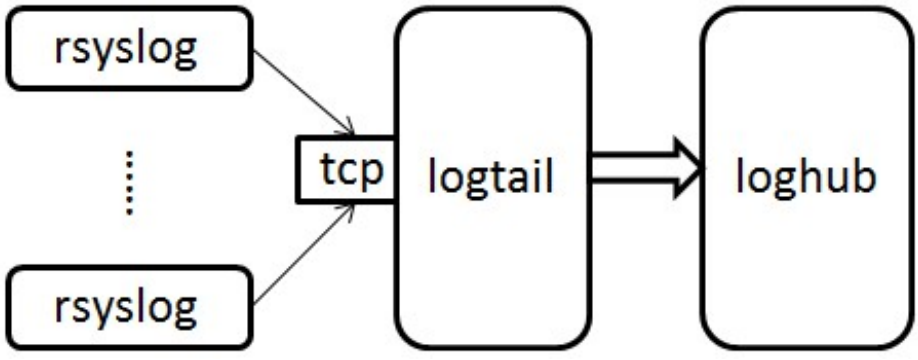
syslog优势

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。
- 日志内部除了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，具体解析方法后面会解释。
tag	数据标签，用于寻找Project/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上面的要求。
- 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：http://tengine.taobao.org/document_cn/http_log_cn.html

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：http://nginx.org/en/docs/syslog.html

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")
```

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。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**。

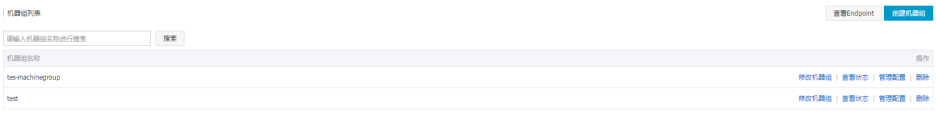
查看机器组列表

登录 **日志服务管理控制台**。

在 **Project列表** 页面，选择所需的项目，单击项目名称或右侧的 **管理**。

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就不可以更改。

登录 **日志服务管理控制台**。

在 **Project列表** 页面，选择所需的项目，单击项目名称或右侧的 **管理**。

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。

在 **Project列表** 页面，选择所需的项目，单击项目名称或右侧的 **管理**。

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。

在 **Project列表** 页面，选择所需的项目，单击项目名称或右侧的 **管理**。

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。

在 **Project列表** 页面，选择所需的项目，单击项目名称或右侧的 **管理**。

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**。

Logtail客户端可以帮助日志服务用户简单得通过控制台就能收集ECS云主机上的日志。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 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Logtail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。

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
%a	locale’s abbreviated weekday name	例如：Fri
%A	locale’s full weekday name	例如：Friday
%b	locale’s abbreviated month name	例如：Jan
%B	locale’s full month name	例如：January
%d	day of the month as a decimal number [1,31]	例如：7, 31
%h	same as %b	例如：Jan
%H	hour (24-hour clock) as a decimal number	例如：22
%I	hour (12-hour clock) as a decimal number	例如：11
%m	month as a decimal number	例如：08
%M	minute as a decimal number	例如：59

	[00,59]	
%n	newline character	换行符
%p	locale’ s equivalent of either a.m. or p.m	例如：AM/PM
%r	time in a.m. and p.m. notation (%I:%M:%S %p)	例如：11:59:59 AM
%R	time in 24 hour notation (%H:%M)	例如：23:59
%S	second as a decimal number[00,59]	例如：59
%t	tab character	TAB符
%y	year without century as a decimal number [00,99]	例如：04,98
%Y	year with century as a decimal number	例如：2004,1998
%z	timezone name or abbreviation, or by no bytes if no timezone information exists	例如：-07:00, +0800
%C	century number as a decimal number [00-99]	例如：16
%e	the day of the month as a decimal number [1,31]; a single digit is preceded by a space{color}	
%j	day of the year as a decimal number [001,366]	例如：365
%u	weekday as a decimal number [1,7], with 1 representing Monday	例如：2
%U	week number of the year (Sunday as the first day of the week) as a decimal number [00,53]	例如：23
%V	week number of the year (Monday as the first day of the week) as a decimal number [01,53]	例如：24
%w	weekday as a decimal number [0,6], with 0 representing Sunday{	例如：5
%W	week number of the year (Monday as the first day of the week) as a decimal number [00,53]	例如：23

%c	locale’s appropriate date and time representation	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	locale’s appropriate date representation	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	locale’s appropriate time representation	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

本文描述Logtail启动配置参数，如有特殊需求，可以参考本文进行设置。

应用场景

配置Logtail启动配置参数适用于以下场景：

- 因内存中需要维护每个文件的签名、采集位置、文件名等元信息，如需采集大量日志文件，可能会导致内存占用率高。
- 日志数据量大，发送到日志服务的流量也很大，导致CPU占用率高。
- 需要收集syslog/TCP数据流。

启动配置

文件路径

/usr/local/ilogtail/ilogtail_config.json

文件格式

JSON

文件示例（只展示部分配置项）

```
{
...
"cpu_usage_limit" : 0.4,
"mem_usage_limit" : 100,
"max_bytes_per_sec" : 2097152,
"process_thread_count" : 1,
"send_request_concurrency" : 4,
"streamlog_open" : false,
```

```
"streamlog_pool_size_in_mb" : 50,
"streamlog_rcv_size_each_call" : 1024,
"streamlog_formats":[],
"streamlog_tcp_port" : 11111,
"buffer_file_num" : 25,
"buffer_file_size" : 20971520,
"buffer_file_path" : "",
...
}
```

常用配置参数

参数名	参数值	参数说明
cpu_usage_limit	CPU使用阈值，double类型，以单核计算。	如0.4，则限制Logtail的CPU使用为CPU单核的40%，超出后Logtail自动重启。大部分场景下，极简模式单核处理能力约24MB/s，完整正则模式单核处理能力约12MB/s，参考 。
mem_usage_limit	常驻内存使用阈值，int类型，以MB计算。	如100，则限制Logtail的内存使用为100兆字节，超出后Logtail自动重启。如需要采集的 distinct文件数目超过1000，请酌情修改上调该阈值。
max_bytes_per_sec	Logtail发送原始数据的流量限制，int类型，以Byte/s 计算。	如 2097152，则限制Logtail 发送数据的速率为2MB/s。
process_thread_count	Logtail处理日志文件写入数据的线程数。	默认1个处理线程，一般可以处理极简模式24MB/s或完整正则模式12MB/s的写入。默认情况下无需调整该阈值，只在必要的时候适当上调。
send_request_concurrency	Logtail默认是异步发送数据包，如果写入TPS 很高可以配置更高的异步并发。	默认4个异步并发，可以按照一个并发支持0.5MB/s~1MB/s网络吞吐来计算，具体依据网络延时而定。
streamlog_open	是否打开接受syslog功能，bool类型。	false表示关闭，true 表示打开，详细说明。
streamlog_pool_size_in_mb	单位是MB，用于缓存接收到的syslog数据。	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。
streamlog_rcv_size_each_call	logtail 每次调用 linux socket rcv 接口使用的缓冲区大小，单位是 byte。	如果 syslog 流量很大，可以调高该值，建议取值范围 1024 到 8192。
streamlog_formats	定义接收到的 syslog 日志解析方式。	详细说明
streamlog_tcp_port	logtail 用于接收 syslog 日志的 TCP 端口。	默认是 11111。

buffer_file_num	网络异常，写入配额超限后，Logtail将实时解析后的日志写入本地文件（安装目录下）缓存起来，等待恢复后尝试重新发送服务端。该参数限制缓存文件的最大数目。	公有云用户默认25个。
buffer_file_size	该参数设置单个缓存文件允许的最大字节数，(buffer_file_num * buffer_file_size) 是缓存文件可以实际使用的最大磁盘空间。	默认20971520字节（20MB）。
buffer_file_path	该参数设置缓存文件存放目录，请在修改该参数后，手动将旧缓存目录下名称如 logtail_buffer_file_* 的文件移动到新缓存目录，以保证Logtail可以读取到该缓存文件并在发送后进行删除。	默认为空，缓存文件存放于程序安装目录（/usr/local/ilogtail）。
bind_interface	本机绑定的网卡名，例如eth1（只支持linux版本）	默认情况下自动绑定可用网卡，若配置该参数，logtail将强制使用该网卡进行日志上传

注意：

- 此表仅列出您需要关注的常用启动参数，如 ilogtail_config.json 内有表格中未列出的参数，会使用默认配置，属于正常情况。
- 请根据需要新增或修改指定配置参数所对应的值，用不到的配置项可以不用增加到 ilogtail_config.json，如采集 syslog 数据流相关设置等。

修改配置

按需配置 ilogtail_config.json

请确认修改配置后，配置内容为合法JSON。

重启Logtail使配置生效

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

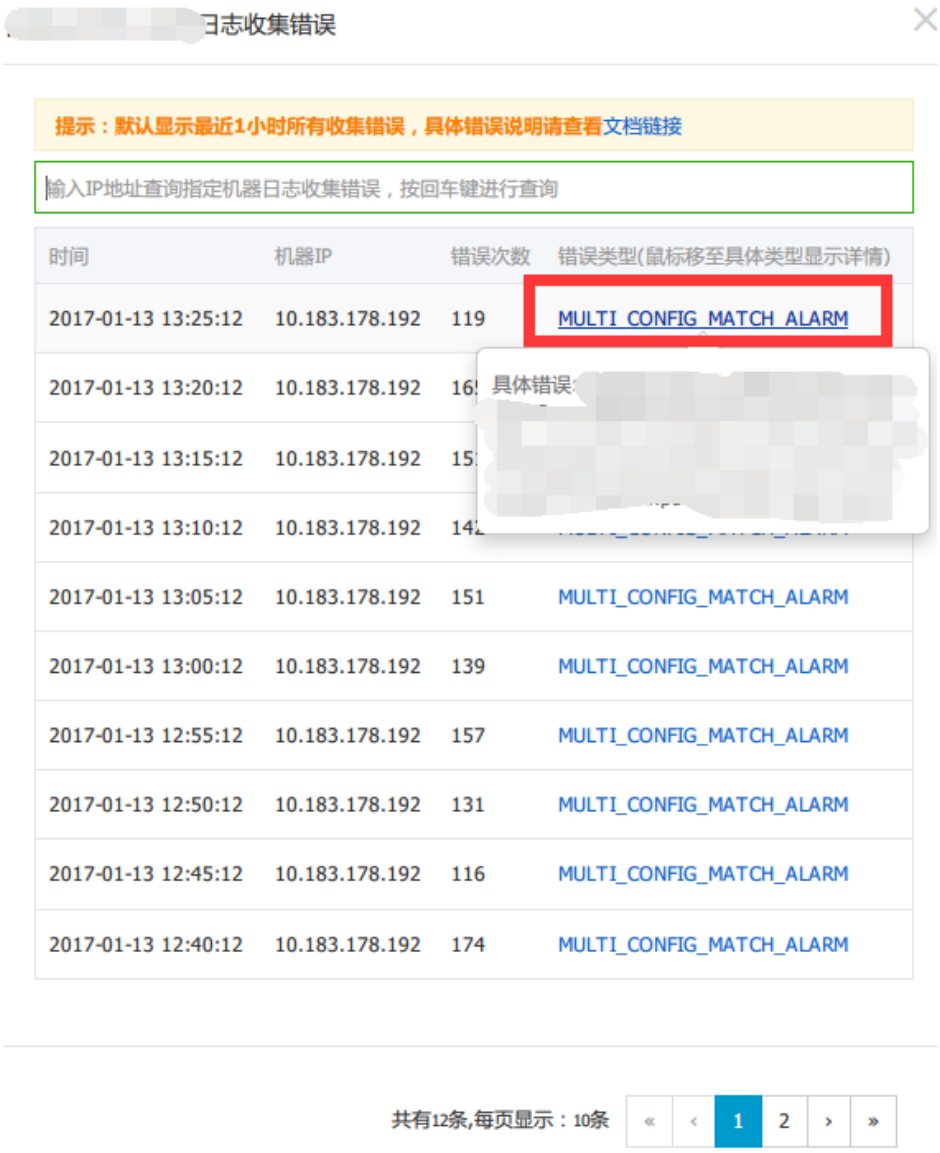
使用Logtail收集日志时，会遇到诸如正则解析失败，文件路径不正确，流量超过Shard服务能力等等错误，日志服务提供诊断功能，支持自主调试Logtail日志收集错误。

1. 错误诊断入口

选择指定Project后，进入LogStore列表，在列表的“日志收集模式”列中找到“调试”链接，点击进入。

2. 查看日志收集错误

进入查询页面后，即可查看指定LogStore对应的Logtail收集日志错误。



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。

附录：Logtail错误类型及处理方式

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	检查行首正则正确性，如果是单

	，无法对日志做分行	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	请点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，如有疑问请提工单解决。
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	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；4)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，如有疑问请提工单解决
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	首先查看是否存在READ_LOG_DELAY_ALARM，

	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	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；如有疑问请提工单解决

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。

请按以下步骤依次进行排查：

1. 确认机器组内Logtail心跳正常

参考Logtail机器组，进入控制台点击查看机器组状态。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。

导致Logtail心跳失败的原因一般有：

机器上未安装Logtail

Linux查看客户端状态:

```
sudo /etc/init.d/ilogtaild status
```

Windows查看客户端状态：

```
控制面板 -> 管理工具 -> 服务
查看LogtailDaemon、LogtailWorker两个Windows Service运行状态。
```

如未安装Logtail客户端，请参考Logtail安装，务必按照您日志服务Project所属Region以及网络类型进行安装。

安装时所选参数错误

日志服务是Region化的，需要在安装时为客户端指定正确的服务端访问入口，请查看您已安装的客户端使用的配置：

- Linux：/usr/local/ilogtail/ilogtail_config.json
- Windows x64 : C:\Program Files (x86)\Alibaba\Logtail\ilogtail_config.json
- Windows x32 : C:\Program Files\Alibaba\Logtail\ilogtail_config.json

确认以下两点：

- 客户端连接的网络入口所属Region是否与您Project所在Region一致。网络入口列表参考服务入口。
- 是否根据您的机器所属网络环境选择了正确的域名。如VPC环境如果选择了内部域名，是无法联通的。可以Telnet测试ilogtail_config.json中配置的域名，如：telnet logtail.cn-hangzhou-intranet.log.aliyuncs.com 80

服务端配置了错误的IP或用户标识

一般来说，Logtail在机器上获取IP的方式为：

- 如果本机在文件/etc/hosts中设置了主机名绑定，需要确认绑定的IP。执行命令hostname可以查看主机名。
- 如果没有设置主机名绑定，会取本机的第一块网卡IP。

在服务器上查看IP地址：

- Linux：/usr/local/ilogtail/app_info.json
- Windows x64 : C:\Program Files (x86)\Alibaba\Logtail\app_info.json
- Windows x32 : C:\Program Files\Alibaba\Logtail\app_info.json

如果服务端机器组内填写的IP与客户端获取的IP不一致，则根据情况进行修改：

- 若服务端机器组填写了错误IP，请修改机器组内IP并保存，等待1分钟再查看。

  若修改了机器上的网络配置（如/etc/hosts修改），请重新启动Logtail以获取新的IP。

  如有需要，可以执行以下命令重启Logtail。

  - Linux：sudo /etc/init.d/ilogtaild stop; sudo /etc/init.d/ilogtaild start
  - Windows : 控制面板 -> 管理工具 -> 服务 -> 重启LogtailWorker

2. 确认是否创建了采集配置并应用到机器组

在客户端状态正常以后，需要确认两件事情：

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。

Logtail配置已应用到机器组

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。

3. 检查采集错误

以上两项正常后，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，您可以执行以下步骤诊断。

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。

> 最后，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，并附上排查过程中的发现的关键信息。

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 变得简洁，如有复杂需求可以参考后者自行 DIY。

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。

您除了可以快速安装 logstash，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。

请在 PowerShell 下执行以下命令，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，请略过。

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。

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

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

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

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
```

插件参数说明

logstash-input-file

通过该插件 tail 方式收集日志文件，细节了解请参考 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

创建采集配置

可以为每种日志新建一个配置文件，命名格式如 xxx.conf，建议统一到 C:\logstash-2.2.2-win\conf\ 目录下方便管理。

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

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。

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。

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
```

验证数据收集

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。

控制台提示

如右提示请忽略，不影响功能使用：io/console not supported; tty will not be manipulated。

其它错误可以去 Google 或 logstash 论坛查找帮助信息。

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。

日志服务通过Web Tracking功能支持HTML、H5、iOS和 Android平台数据的采集，允许您自定义维度和指标。

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），比如：

- 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打开互联网匿名写的权限，可能会产生脏数据，敬请留意。

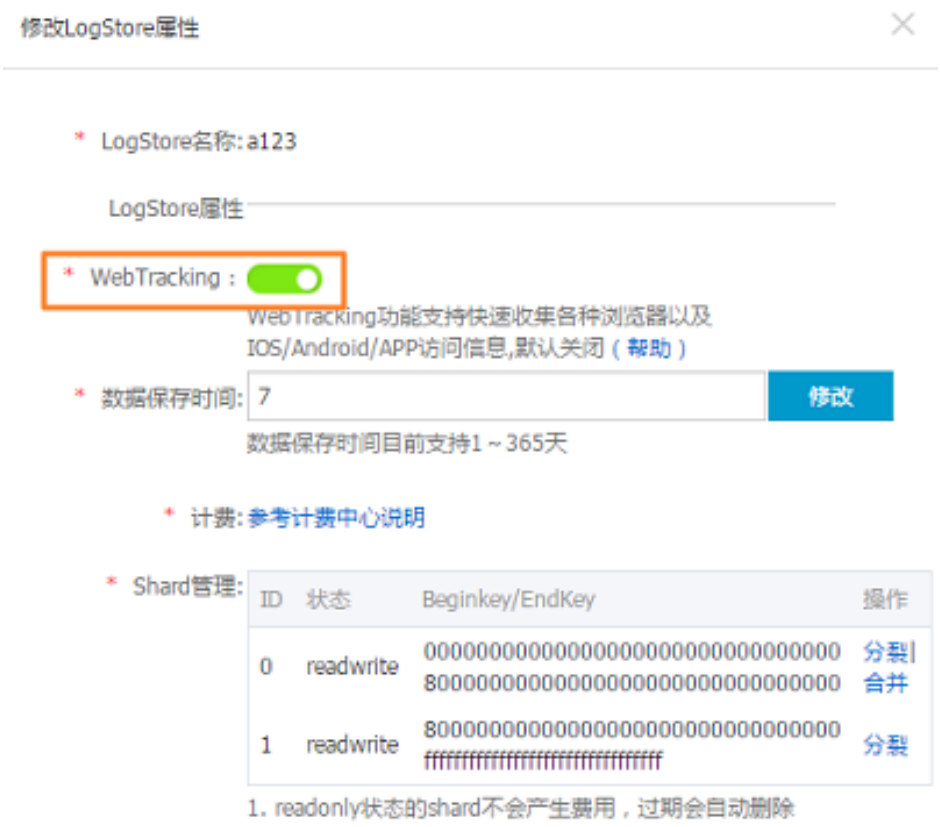
开通Web Tracking

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

选中需要开通Web Tracking功能的Logstore。

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();
}
}
}
```

上传数据

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请求发送数据到日志服务中

```
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();
```

上面语句执行完后，在日志服务中就会看到如下两条日志：

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


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

消费数据

数据上传到日志服务之后，可以使用日志服务将 数据导入MaxCompute或者 EMR 进行数据分析，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有 ERROR、WARN、INFO、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。

使用 Loghub Log4j Appender，您可以控制日志的输出目的地为阿里云日志服务，有一点需要特别注意，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。

Loghub Log4j Appender 的优势

- 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 IO 效率。

使用方法

Step 1： maven 工程中引入依赖。

```
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>log-loghub-log4j-appender</artifactId>
<version>0.1.3</version>
</dependency>
```

Step 2: 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。

使用 loghub 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。

配置参数

Loghub Log4j Appender 可供配置的参数如下，其中注释为必选参数的是必须填写的，可选参数在不填写的情况下，使用默认值。

```
#日志服务的 project 名，必选参数
log4j.appender.loghub.projectName = [you project]
#日志服务的 logstore 名，必选参数
log4j.appender.loghub.logstore = [you logstore]
#日志服务的 HTTP 地址，必选参数
log4j.appender.loghub.endpoint = [your project endpoint]
#用户身份标识，必选参数
log4j.appender.loghub.accessKeyId = [your accesskey id]
log4j.appender.loghub.accessKey = [your accesskey]
#当使用临时身份时必须填写，非临时身份则删掉这行配置
log4j.appender.loghub.stsToken=[your ststoken]
#被缓存起来的日志的发送超时时间，如果缓存超时，则会被立即发送，单位是毫秒，可选参数
log4j.appender.loghub.packageTimeoutInMS=3000
```

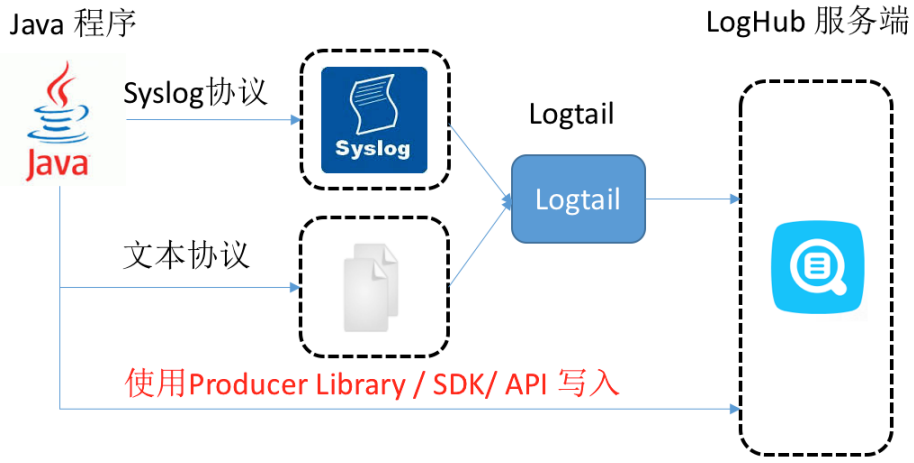
```
#每个缓存的日志包中包含日志数量的最大值，不能超过 4096，可选参数
log4j.appender.loghub.logsCountPerPackage=4096
#每个缓存的日志包的大小的上限，不能超过 5MB，单位是字节，可选参数
log4j.appender.loghub.logsBytesPerPackage = 5242880
#Appender 实例可以使用的内存的上限，单位是字节，默认是 100MB，可选参数
log4j.appender.loghub.memPoolSizeInByte=1048576000
#后台用于发送日志包的 IO 线程的数量，默认值是 1，可选参数
log4j.appender.loghub.ioThreadsCount=1
# 输出到日志服务的时间格式，使用 Java 中 SimpleDateFormat 格式化时间，默认是 ISO8601，可选参数
log4j.appender.loghub.timeFormat=yyyy-MM-dd'T'HH:mmZ
log4j.appender.loghub.timeZone=UTC
```

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 读写包装，降低数据收集与消费的门槛。

LogHub Producer Library 解决的问题：

- 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 IO 逻辑分离：打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代价，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，其他语言待开发。

LogHub Producer Library 功能

- 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 IO 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。

使用方法

producer 使用分为以下几个步骤：

step 1: maven 工程中添加依赖

```
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>log-loghub-producer</artifactId>
<version>0.1.4</version>
</dependency>
```

step 2：程序中配置 ProducerConfig

其中，各个参数说明如下：

```
public class ProducerConfig
{
//被缓存起来的日志的发送超时时间，如果缓存超时，则会被立即发送，单位是毫秒
public int packageTimeoutInMS = 3000;
//每个缓存的日志包中包含日志数量的最大值，不能超过 4096
public int logsCountPerPackage = 4096;
//每个缓存的日志包的大小的上限，不能超过 5MB，单位是字节
public int logsBytesPerPackage = 5 * 1024 * 1024;
//单个 producer 实例可以使用的内存的上限，单位是字节
public int memPoolSizeInByte = 1000 * 1024 * 1024;
//IO 线程池最大线程数量，主要用于发送数据到日志服务
public int maxIOThreadSizeInPool = 50;
//当使用指定 shardhash 的方式发送日志时，这个参数需要被设置，否则不需要关心。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 会定时从从 loghub 拉取，该参数的含义是每隔 shardHashUpdateIntervalInMS 毫秒，更新一次 shard 的 hash 区间。
public int shardHashUpdateIntervalInMS = 10 * 60 * 1000;
//如果发送失败，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。
public int retryTimes = 3;
}
```

step 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。

step 4：创建 producer 实例，调用 send 接口发数据

示例

main：

```
public class ProducerSample {
private final static int ThreadsCount = 25;

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
Thread[] threads = new Thread[ThreadsCount];
```

```
for (int i = 0; i < ThreadsCount; ++i) {
threads[i] = new Thread(null, new Runnable() {
Random random = new Random();

public void run() {
int j = 0, rand = random.nextInt(99999);
while (++j < Integer.MAX_VALUE) {
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));
}
}
}, i + "");
threads[i].start();
}
//等待发送线程退出
Thread.sleep(1 * 60 * 60 * 1000);
//主动刷新缓存起来的还没有被发送的日志
producer.flush();
//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"
```

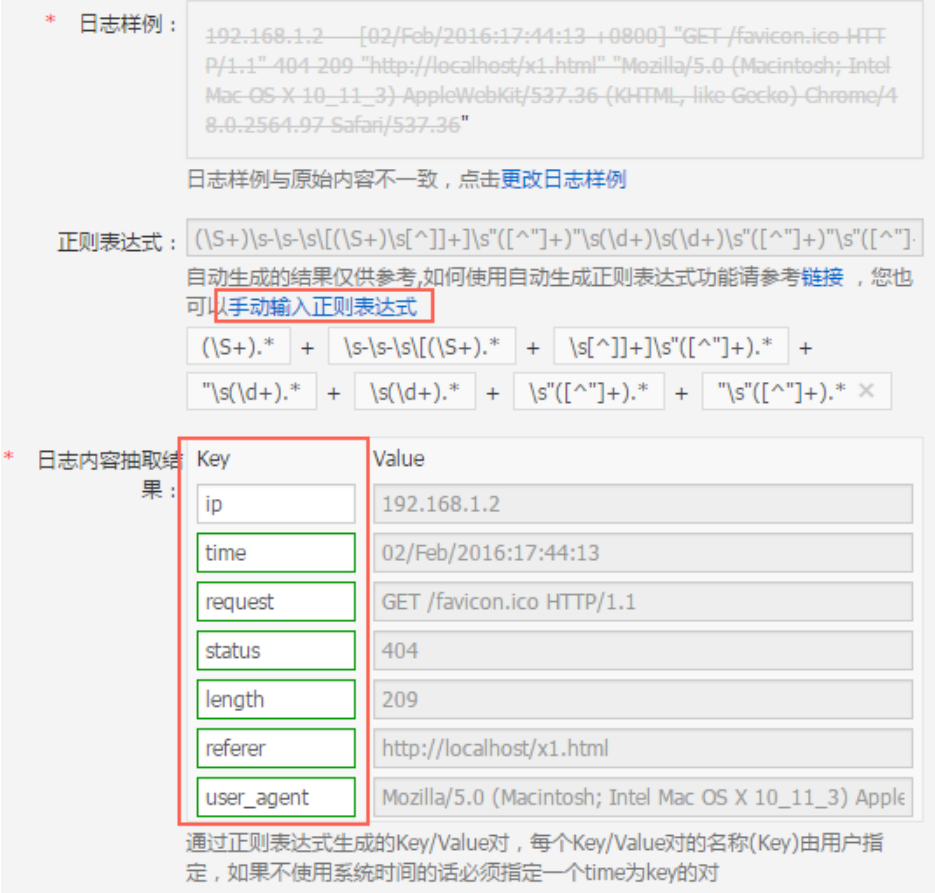
配置Logtail收集apache日志

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。

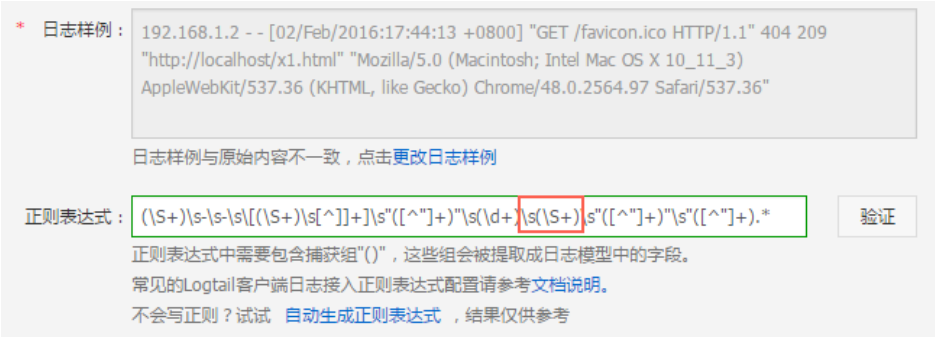
填写配置名称、日志路径，并选择日志收集模式为**完整正则模式**。

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

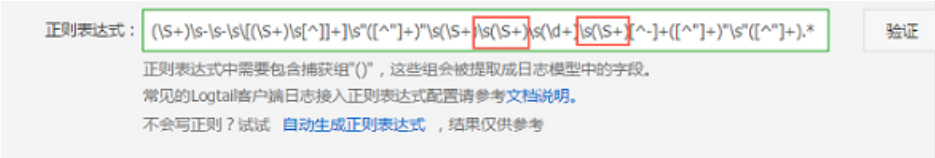
通过Logtail收集nginx日志完整流程请参考**快速入门**，根据您的网络部署和实际情况选择对应配置。本文档仅展示配置Logtail的第二步**指定收集模式**步骤中的详细配置。

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

Nodejs 的日志默认打印到控制台，这对于收集数据和调查问题非常不方便。通过 log4js 这个包，可以实现把日志打印到文件、自定义日志格式等功能。

```
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 的输出级别 6 个: trace、debug、info、warn、error、fatal

通过Logtail收集Nodejs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Nodejs日志示例，为您的日志撰写正确、全面的正则表达式。

常见的Nodejs日志及其正则表达式：

Nodejs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

Nodejs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 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

日志介绍

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（Separator，如制表符、空格、竖线、逗号、分号等单字符）连接日志的多个字段。如果字段内部包含分隔符，使用 Quote（双引号）对字段进行包裹。

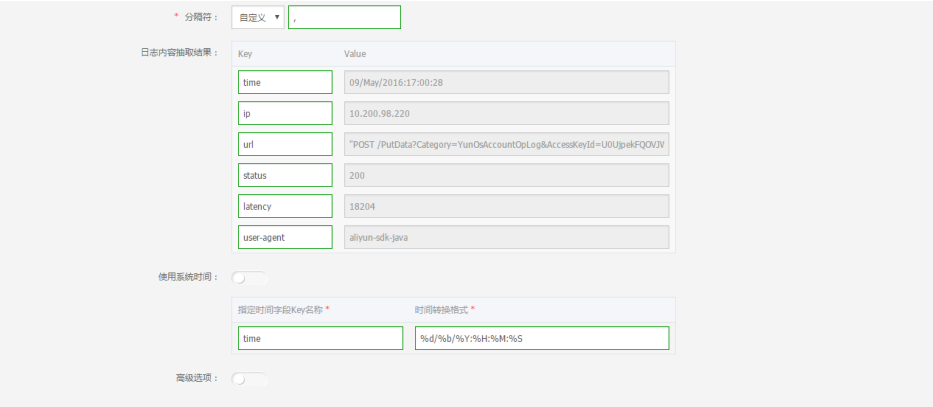
常见的分隔符日志有：CSV、TSV 等。

日志示例

```
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
```

采集配置

如上日志样例，使用逗号（,）进行分割，一共包含 6 列，设置列名分别为：time，ip，url，status，latency，user-agent。



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。

采集结果

更多日志格式问题

Separator

分隔符日志使用分隔符（separator）将一条日志切分成多个字段，有以下格式要求：

- separator 必须是单个字符，例如制表符（\t）、空格、竖线（|）、逗号（,）、分号（;）等单字符。
- spearator 不允许设置为多字符，例如 ||，&&& 等形式。
- spearator 不允许设置为双引号（” ），双引号被作为默认的 quote。

Quote

为了防止日志的字段内部包含分隔符（separator），使用 quote（双引号）对字段进行包裹。如果内容中在非 quote 情况下出现双引号，需要进行转义，处理成”” 。

- 使用双引号作为 quote

使用逗号（,）作为 separator，quote 必须紧邻 separator（如有两者之间包含空格、制表符等字符，请修改格式），日志样例如下：

```
1997,Ford,E350,"ac, abs, moon",3000.00
```

解析成 5 个字段：

字段 1	字段 2	字段 3	字段 4	字段 5

1997	Ford	E350	ac, abs, moon	3000.00

- 日志字段内通过转义处理双引号

第 3 个日志字段内有双引号，但并不作为 quote 使用，日志样例如下：

```
1999,Chevy,"Venture ""Extended Edition, Very Large""","",5000.00
```

解析成 5 个字段：

字段 1	字段 2	字段 3	字段 4	字段 5
1999	Chevy	Venture “Extended Edition, Very Large”		5000.00

也就是说，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（” ”），其它情况不符合分割符日志的格式定义，请考虑其它方式（极简模式，正则模式等）进行字段解析。

更多细节请参考：维奇百科，百度百科。

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%202013%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%202013%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。

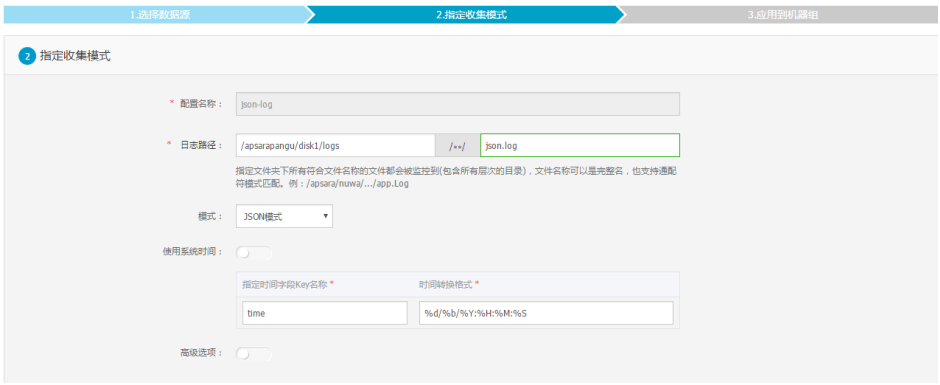
开启 **使用系统时间**

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、grok。

重启 Logstash 生效

创建配置文件到 conf 目录，参考 通过 logstash 收集 Windows 平台日志 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 通过 logstash 收集 Windows 平台日志 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 通过 logstash 收集 Windows 平台日志 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 通过 logstash 收集 Windows 平台日志 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。

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：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，单击项目名称或右侧的 **管理**。

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。

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。

使用场景

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。

举一个例子，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 read only 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：

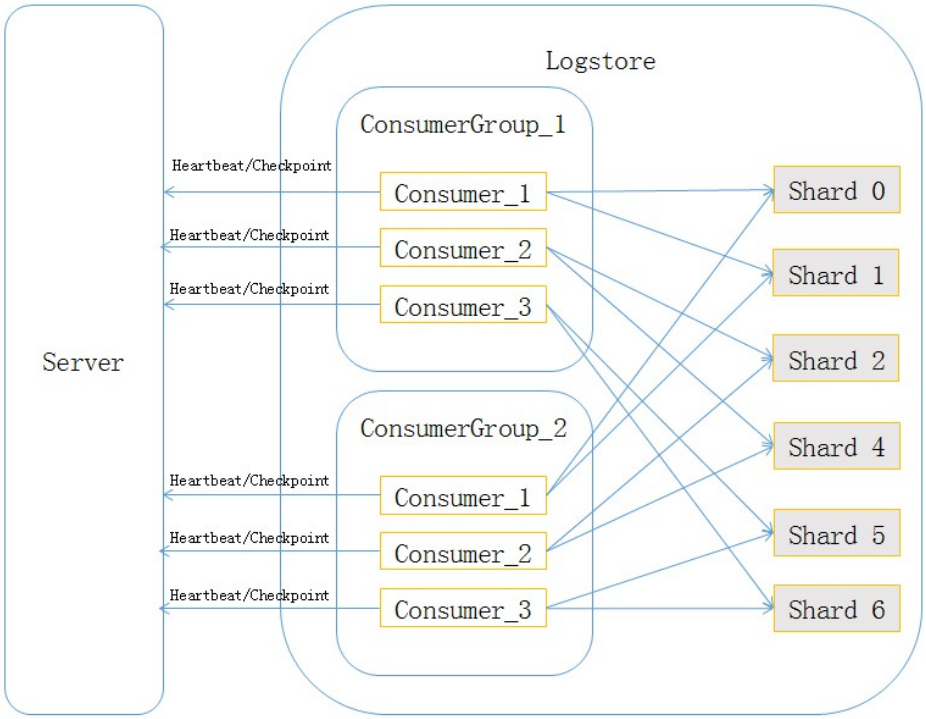
1. 填写配置参数。
2. 写处理日志的代码。
3. 启动消费实例。

强烈建议使用 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。

术语简介

loghub consumer library 中主要有 4 个概念，分别是 consumer group、consumer、heartbeat 和

checkpoint，它们之间的关系如下：



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 5 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。

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.aliyun.openservices</groupId>
<artifactId>loghub-client-lib</artifactId>
<version>0.6.5</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)
{
LogGroup lg = logGroup.GetLogGroup();
System.out.println("source ip:" + lg.getSource());
System.out.println("topic: " + lg.getTopic());
for(Log log: lg.getLogsList())
{
StringBuilder content = new StringBuilder();
content.append(log.getTime() + "\t");
for(Content cont: log.getContentsList())
{
content.append(cont.getKey() + "=" + cont.getValue()+ "\t");
}
System.out.println(content.toString());
}
}
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'_','-',长度在 [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 shard 0~shard 3 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}**

背景

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

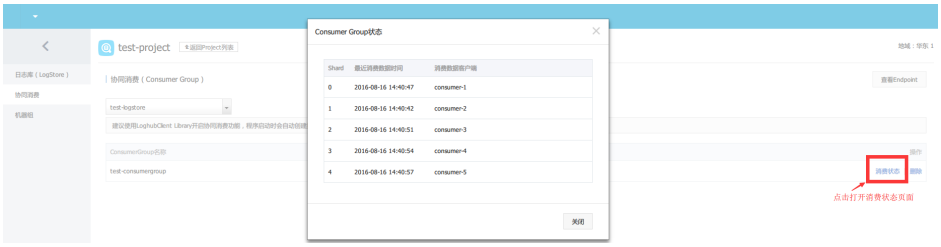
登录 日志服务管理控制台。

选择所需的项目，单击项目名称或右侧的 **管理**。

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 Java SDK 作为例子，演示如何通过 API 获得消费状态。

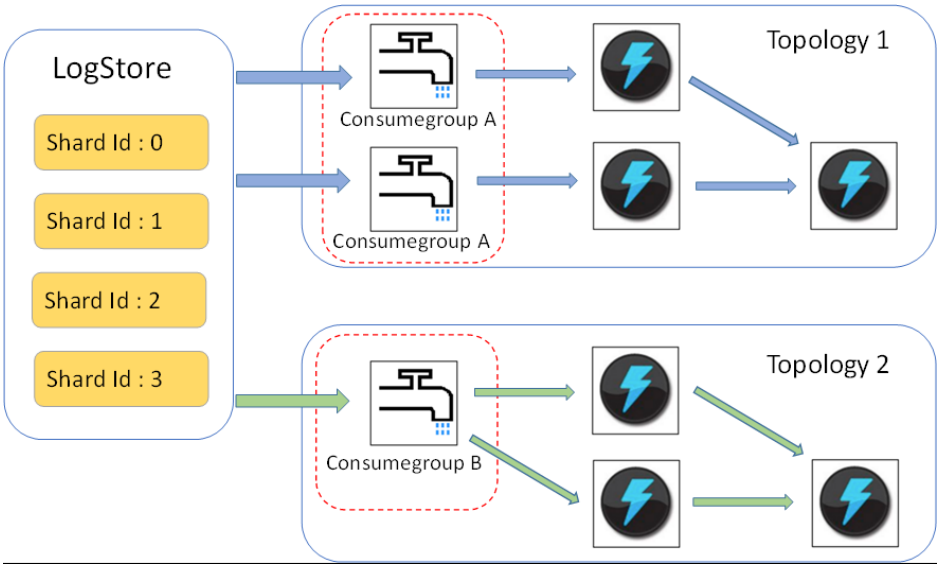
```
package test;

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);
}
}
}

}
```

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>0.6.4</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>0.1.1</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

StreamCompute 在创建Loghub类型数据源后，可以直接消费Loghub中数据，配置如下：

```
CREATE STREAM TABLE source_test_galaxy ( $schema ) WITH ( type='loghub', endpoint=$endpoint,
accessId=$loghub_access_id, accessKey=$loghub_access_key, projectName=$project, logstore=$logstore );
```

参数列表：

参数	参数说明
$schema	将日志中哪些Key映射成StreamCompute表中的

	Column，如：name STRING, age STRING, id STRING。
$endpoint	数据访问接入点，各Region接入点。
$loghub_access_id	有读权限账号（或子账号）对应AccessId。
$loghub_access_key	有读权限账号（或子账号）对应AccessKey。
$project	数据所在Project。
$logstore	数据所在Logstore。

示例：

```
CREATE STREAM TABLE source_test_galaxy ( name STRING, age STRING, id STRING ) WITH ( type='loghub',
endpoint='http://cn-hangzhou-intranet.log.aliyuncs.com', accessId='mock_access_id',
accessKey='mock_access_key', projectName='ali-cloud-streamtest', logstore='stream-test' );
```

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

业务实时监控服务 ARMS 是一款端到端一体化实时监控解决方案的PaaS级阿里云产品，如果用户已经开通了阿里云日志服务，ARMS可与LogHub无缝对接。

操作步骤：

- 准备数据源 > LogHub数据源

投递日志

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

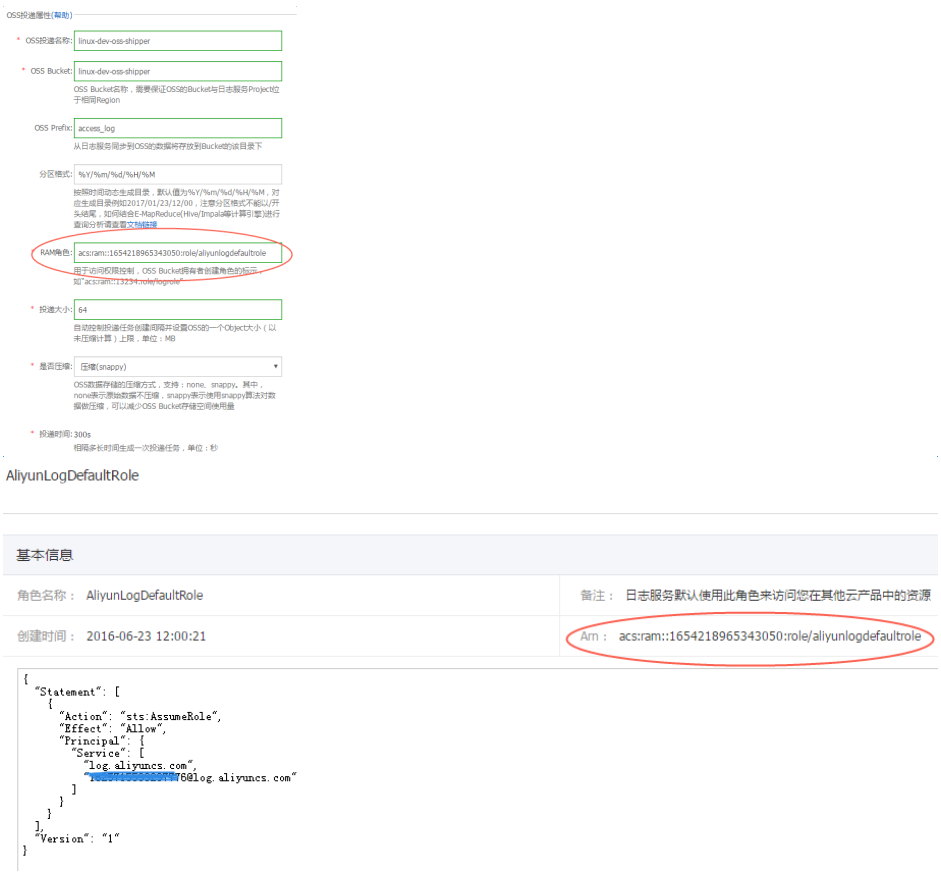
登录 日志服务管理控制台。

选择所需的项目，单击项目名称或者右侧的 **管理**。

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**。

- **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，需要保证 OSS 的 Bucket 与日志服务 Project 位于相同 Region。
- **OSS Prefix**：从日志服务同步到 OSS 的数据将存放到 Bucket 的该目录下。
- **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（格式化参考strptime API），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。
- **RAM角色**：用于访问权限控制，OSS Bucket 拥有者创建角色的标识，如 acs:ram::45643:role/aliyunlogdefaultrole，如何获得ARN请参见步骤1/查看、修改角色。
- **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，单位为 MB。
- **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，支持：none、snappy。其中，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**：相隔多长时间生成一次投递任务，单位为秒，默认值 300。



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_285008.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_example	%Y%m%d%H	oss://test-bucket/log_ship_oss_example/2017012019_14849130433515 25351_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（如MaxCompute表结构不符合日志服务规范、无授权等），需要您参与解决问题。您可以参考 投递日志到MaxCompute 的 **投递任务管理** 部分了解更多细节。

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。

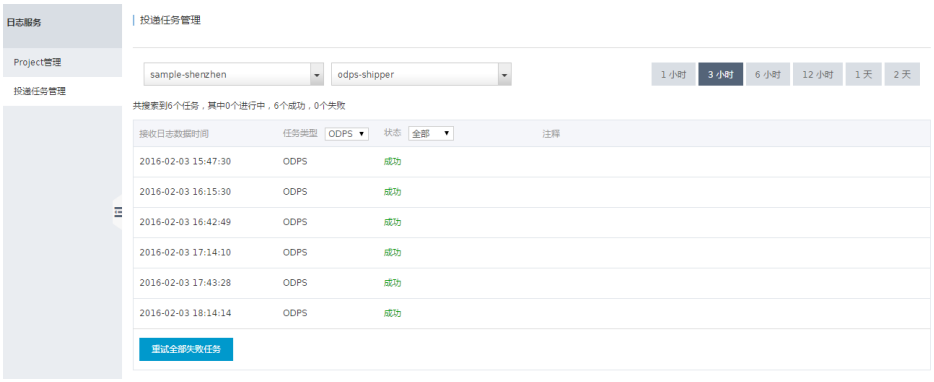
操作步骤

登录 日志服务管理控制台。

选择所需的项目，单击项目名称或者右侧的 **管理**。

选择所需的日志库，单击日志投递列下的 OSS **管理**。

您可以查看投递任务的状态。



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，你也可以手动重试。

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	处理方法
UnAuthorized	没有权限，请确认： - 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： - 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。

理解目录的时间含义

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。

本文介绍日志服务投递OSS使用JSON存储的相关细节，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日志，JSON 格式，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，我们提供了工具进行解压 snappy 文件，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
```

本文介绍日志服务投递OSS使用Parquet存储的相关细节，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"
}
```

功能优势

日志服务收集的日志除了可以被实时查询外，还可以把日志数据投递到大数据计算服务MaxCompute（原ODPS），进一步进行个性化BI分析及数据挖掘。通过日志服务投递日志数据到MaxCompute具有如下优势：

使用便捷

您只需要完成2步配置即可以把日志服务Logstore的日志数据迁移到MaxCompute中。

避免重复收集工作

由于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一遍日志数据后再导入到MaxCompute。

充分复用日志服务内的日志分类管理工作

用户可让日志服务中不同类型的日志（存在不同Logstore中）、不同Project的日志自动投递到不同的MaxCompute表格，方便管理及分析MaxCompute内的日志数据。

> 在大部分情况下日志数据在写入Logstore后的0.5~1个小时导入到MaxCompute，用户可以在控制台“投递任务管理”查看导入状态。导入成功后用户即可在MaxCompute内查看到相关日志数据。

结合日志服务的实时消费，投递日志数据到MaxCompute的数据通道以及日志索引功能，可以让用户按照不同的场景和需求、以不同的方式复用数据，充分发挥日志数据的价值。

配置流程

举例日志服务的一条日志如下：

```
16年01月27日20时50分13秒
10.170.148.237
ip:10.200.98.220
status:200
thread:414579208
time:27/Jan/2016:20:50:13 +0800
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
user-agent:aliyun-sdk-java
```

日志左侧的ip、status、thread、time、url、user-agent等是日志服务数据的字段名称，需要在下方配置中应用到。

1 初始化数加平台

1.1 在日志服务的控制台Logstore列表单击日志投递列的MaxCompute。

自动跳转到初始化数加平台的页面。MaxCompute默认为按量付费模式，具体参见MaxCompute计量计费说明

1.2 查看服务协议和条款后单击确定，初始化数加平台。

初始化开通需10~20秒左右，请耐心等待。如果已经开通数加及大数据计算服务MaxCompute（原ODPS），将直接跳过该步骤。

目前暂不支持子用户来投递大数据计算服务MaxCompute（原ODPS）。

2.1 数据模型映射

在日志服务和大数据计算服务MaxCompute（原ODPS）之间同步数据，涉及两个服务的数据模型映射问题。您可以参考日志服务日志数据结构和MaxCompute表了解两种数据结构。

将样例日志导入MaxCompute，分别定义MaxCompute数据列、分区列与日志服务字段的映射关系：

MaxCompute列类型	MaxCompute列名（可自定义）	MaxCompute列类型（可自定义）	日志服务字段名（投递配置里填写）	日志服务字段类型	日志服务字段语义
数据列	log_source	string	__source__	系统保留字段	日志来源的机器IP。
数据列	log_time	bigint	__time__	系统保留字段。	日志的Unix时间戳，从1970年1月1日开始所经过的秒数，由用户日志的time字段计算得到。
数据列	log_topic	string	__topic__	系统保留字段。	日志主题。
数据列	time	string	time	日志内容字	解析自日志

				段。	。
数据列	ip	string	ip	日志内容字段。	解析自日志。
数据列	thread	string	thread	日志内容字段。	解析自日志。
数据列	log_extract_others	string	__extract_others__	系统保留字段。	未在配置中进行映射的其它日志内字段会通过key-value序列化到json，该json是一层结构，不支持字段内部json嵌套。
分区列	log_partition_time	string	__partition_time__	系统保留字段。	由日志的time字段对齐计算而得，分区粒度可配置，在配置项部分详述。
分区列	status	string	status	日志内容字段。	解析自日志，该字段取值应该是可以枚举的，保证分区数目不会超出上限。

- MaxCompute表至少包含一个数据列、一个分区列。
- 系统保留字段中建议使用__partition_time__，__source__，__topic__。
- MaxCompute单表有分区数目6万的限制，分区数超出后无法再写入数据，所以日志服务导入MaxCompute表至多支持3个分区列。请谨慎选择自定义字段作为分区列，保证其值是可枚举的。
- 系统保留字段__extract_others__历史上曾用名_extract_others_，填写后者也是兼容的。
- MaxCompute分区列的值不支持”/“等特殊字符，这些是MaxCompute的保留字段。
- MaxCompute分区列取值不支持空，所以映射到分区列的字段必须要在日志里存在，空分区列的日志会在投递中被丢弃。

3 投递配置

3.1 开启投递

初始化数加平台之后，根据页面提示进入**LogHub —— 数据投递**页面，选择需要投递的Logstore，并单击**开启投递**。

您也可以在**MaxCompute(原ODPS)投递管理**页面选择需要投递的Logstore，并单击**开启投递**以进入**LogHub —— 数据投递**页面。

3.2 配置投递规则

在**LogHub —— 数据投递**页面配置**字段关联**等相关内容。

选项含义：

参数	语义
投递名称	自定义一个投递的名称，方便后续管理。
MaxCompute Project	MaxCompute项目名称，该项默认为新创建的Project，如果已经是MaxCompute老客户，可以下拉选择已创建其他Project。
MaxCompute Table	MaxCompute表名称，请输入自定义的新建的

	MaxCompute表名称或者选择已有的MaxCompute表。
MaxCompute 普通列	按序，左边填写与MaxCompute表数据列相映射的日志服务字段名称，右边填写或选择MaxCompute表的普通字段名称及字段类型。
MaxCompute 分区列	按序，左边填写与MaxCompute表分区列相映射的日志服务字段名称，右边填写或选择MaxCompute表的普通字段名称及字段类型。
分区时间格式	__partition_time__输出的日期格式，参考Java SimpleDateFormat。
导入MaxCompute间隔	MaxCompute数据投递间隔，默认1800，单位：秒。

- 该步会默认为客户创建好新的MaxCompute Project和Table，其中如果已经是MaxCompute老客户，可以下拉选择其他已创建Project。
- 日志服务投递MaxCompute功能按照字段与列的顺序进行映射，修改MaxCompute表列名不影响数据导入，如更改MaxCompute表schema，请重新配置字段与列映射关系。
- 日志服务数据的一个字段最多允许映射到一个MaxCompute表的列（数据列或分区列），不支持字段冗余。

参考信息

__partition_time__格式

将日志时间作为分区字段，通过日期来筛选数据是MaxCompute常见的过滤数据方法。

日志服务根据日志Time字段和分区时间格式计算出日期作为分区列。为满足MaxCompute单表分区数目的限制，日期分区列的值会按照导入MaxCompute间隔对齐。

举例来说，日志提取的time字段是”27/Jan/2016:20:50:13 +0800”，日志服务据此计算出保留字段__time__为1453899013（Unix时间戳），不同配置下的时间分区列取值如下：

导入MaxCompute间隔	分区时间格式	__partition_time__
1800	yyyy_MM_dd_HH_mm_00	2016_01_27_20_30_00
1800	yyyy-MM-dd HH:mm	2016-01-27 20:30
1800	yyyyMMdd	20160127
3600	yyyyMMddHHmm	201601272000
3600	yyyy_MM_dd_HH	2016_01_27_20

1. 请勿使用精确到秒的日期格式：1. 很容易导致单表的分区数目超过限制（6万）；2. 单次投递任务的数据分区数目必须在512以内。
2. 以上分区时间格式是测试通过的样例，您也可以参考Java SimpleDateFormat自己定义日期格式，但是该格式不得包含斜线字符”/“（这是MaxCompute的保留字段）。

__partition_time__ 使用方法

使用MaxCompute的字符串比较筛选数据，可以避免全表扫描。比如查询2016年1月26日一天内日志数据：

```
select * from {ODPS_TABLE_NAME} where log_partition_time >= "2015_01_26" and log_partition_time <
"2016_01_27";
```

__extract_others__ 使用方法

log_extract_others为一个json字符串，如果想获取该字段的user-agent内容，可以进行如下查询：

```
select get_json_object(sls_extract_others, "$.user-agent") from {ODPS_TABLE_NAME} limit 10;
```

1. get_json_object是MaxCompute提供的标准UDF。请联系MaxCompute团队开通使用该标准UDF的权限。
2. 示例供参考，请以MaxCompute产品建议为最终标准。

其他操作

编辑投递配置

在Logstore列表投递项，单击“修改”即可针对之前的配置信息进行编辑。其中如果想新增列，可以在大数据计算服务MaxCompute（原ODPS）修改投递的数据表列信息，则点击“修改”后会加载最新的数据表信息。

投递任务管理

在启动“投递功能”后，日志服务后台会定期启动离线投递任务。用户可以在控制台上看到这些投递任务的状态和错误信息。具体请参考日志投递任务管理。

如果投递任务出现错误，控制台上会显示相应的错误信息：

错误信息	建议方案
MaxCompute项目空间不存在	在MaxCompute控制台中确认配置的MaxCompute项目是否存在，如果不存在则需要重新创建或配置。
MaxCompute表不存在	在MaxCompute控制台中确认配置的MaxCompute表是否存在，如果不存在则需要重新创建或配置。
MaxCompute项目空间或表没有向日志服务授权	在MaxCompute控制台中确认授权给日志服务账号的权限是否还存在，如果不存在则需要重新添加上相应权限。
MaxCompute错误	显示投递任务收到的MaxCompute错误，请参考MaxCompute相关文档或联系MaxCompute团队解决。日志服务会自动重试最近两天时间的失败任

	务。
日志服务导入字段配置无法匹配MaxCompute表的列	重新配置MaxCompute表格的列与日志服务数据字段的映射配置。

当投递任务发生错误时，请查看错误信息，问题解决后可以通过管理控制台中“日志投递任务管理”或SDK来重试失败任务。

MaxCompute中消费日志

MaxCompute用户表中示例数据如下：

```
| log_source | log_time | log_topic | time | ip | thread | log_extract_others | log_partition_time | status |
+------------+------------+-----------+-----------+-----------+-----------+-----------------+--------------------+--------
----+
| 10.170.148.237 | 1453899013 | | 27/Jan/2016:20:50:13 +0800 | 10.200.98.220 | 414579208 | {"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","user-agent":"aliyun-sdk-java"} | 2016_01_27_20_50 | 200 |
+------------+------------+-----------+-----------+-----------+-----------+-----------------+--------------------+--------
----+
```

同时，我们推荐客户直接使用已经与MaxCompute绑定的大数据开发Data IDE来进行可视化的BI分析及数据挖掘，这将提高数据加工的效率。

授予MaxCompute数据投递权限

如果在数加平台执行表删除重建动作，会导致默认授权失效。请手动重新为日志服务投递数据授权。

在MaxCompute项目空间下添加用户：

```
ADD USER aliyun$shennong_open@aliyun.com;
```

> shennong_open@aliyun.com 是日志服务系统账号（请不要用自己的账号），授权目的是为了能将数据写入到MaxCompute

MaxCompute项目空间Read/List权限授予：

```
GRANT Read, List ON PROJECT {ODPS_PROJECT_NAME} TO USER aliyun$shennong_open@aliyun.com;
```

MaxCompute项目空间的表Describe/Alter/Update权限授予：

```
GRANT Describe, Alter, Update ON TABLE {ODPS_TABLE_NAME} TO USER aliyun$shennong_open@aliyun.com;
```

确认MaxCompute授权是否成功：

```
SHOW GRANTS FOR aliyun$shennong_open@aliyun.com;

A projects/{ODPS_PROJECT_NAME}: List | Read
A projects/{ODPS_PROJECT_NAME}/tables/{ODPS_TABLE_NAME}: Describe | Alter | Update
```

可以参考MaxCompute用户授权管理指南了解更多细节。

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	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**

	符合日志服务规范、无授权等。	**理** 部分。

索引查询

日志服务提供日志索引功能，支持通过关键词、上下文等进行日志查询。索引功能默认关闭，如果需要使用该功能，请在Logstore上创建索引。

创建索引后，输入关键词即可查询日志。日志服务支持在搜索关键词中使用查询语法，通过定制化索引完成条件检索。

例如：

索引内容	含义
KEY_1 OR KEY_?2 AND KEY_N > 300	日志中出现 KEY_1 或满足 KEY_?2 (例如 KEY_12、KEY_22) 并且 KEY_N 数值大于 300
KEY_1 AND KEY_*2	日志中同时出现 KEY_1 和 KEY_*2 (例如 KEY_222、KEY_123452)
KEY_1 AND KEY_*2	日志中出现 KEY_1 但不出现 KEY_2
(KEY_1 OR KEY_2) AND KEY_3 NOT KEY_4	日志中出现 KEY_1 或 KEY_2，并且出现 KEY_3 但不出现 KEY_4

完整查询语法请参考 **查询语法**。

配置说明

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，也能节省使用费用。

- 所有查询不需要指定键名称（Key）
  - 可以只设置 **全文索引属性**
  - 无需设置 **键值索引属性**
- 部分查询需要指定键名称（Key）
  - 根据需求，对特定键（Key）创建键值索引

- 查询需要指定数值范围
- 可以设置特定键（Key）索引类型为long或者double

操作步骤

登录 日志服务管理控制台。

选择目标项目，单击项目名称或者单击右侧的 **管理**。

选择目标日志库并单击日志索引列下的 **查询**。

单击右上角的 **索引属性** > **开启索引**。

填写索引配置信息。

例如日志库中日志的日志内容为：

时间/IP	内容
16年03月30日 16时05分04秒 10.101.166.22	agent:test&agent code:internalError error:internalError message:a,b;c;D-F version:20160301

全文索引属性

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**，则只能通过关键字“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”。

键值索引属性

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，需要设置键值索引，如下图示：

其中，**键名称** 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

全文索引属性 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
```

日志服务的索引模式能让您在Logstore上快速查询海量的日志数据，检索出您真正感兴趣的日志数据。日志写入Logstore后，您可以通过控制台为Logstore内的日志内容 创建索引。

索引和查询特点如下：

- 大规模：支持 PB/Day 数据量
- 低延时：数据写入 3 秒内既可以查询
- 秒级返回：一秒可以过 10 亿级日志

注意：

- 消费日志与查询日志都意味着“读取”日志。
- 在查询特定日志后，使用Logtail的用户可以使用 上下文查询功能。

使用控制台查询日志

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。

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添加查询关键字。

- 提供 上下文查询功能 帮助定位。

操作步骤

登录 日志服务管理控制台。

选择需要的项目，单击项目名称或者单击右侧的 **管理**。

选择所需的日志库并单击日志索引列下的 **查询**。

您可以设置查询条件（日志库，topic，关键字，时间间隔，时间范围等）单击右上角的 **查询链接** 查询日志。

查询语法

参见 日志查询语法 章节。索引参见 索引设置 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：

修改索引属性。

详情请参见修改索引设置。

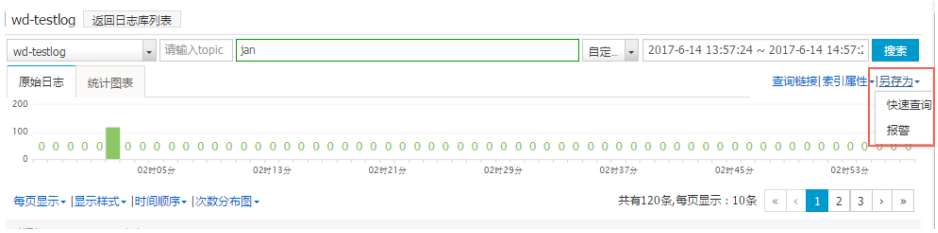
查询条件另存为**快速查询**或**报警**。

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，在Logtail导航栏左侧**LogSearch-索引查询**下的**快速查询**页面下，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。

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。



原始日志和**日志分布**形式的查询结果显示。

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，你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。

创建日志索引后，您可以通过日志服务管理控制台修改索引的设置。

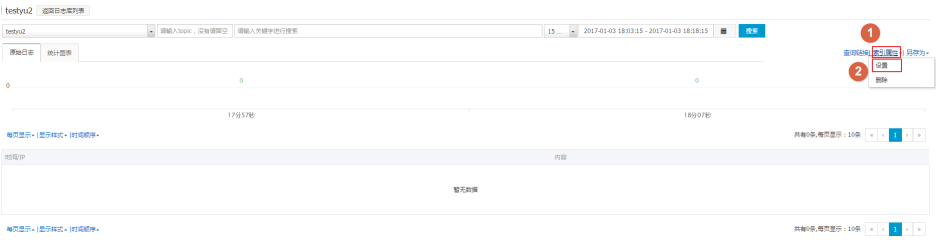
操作步骤

登录 日志服务管理控制台。

选择需要的项目，单击项目名称或者单击右侧的 **管理**。

选择所需的日志库并单击日志索引列下的 **查询**。

单击右上角的 **索引属性** > **设置**。



修改索引设置并单击 **确认**。

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	支持
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%

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

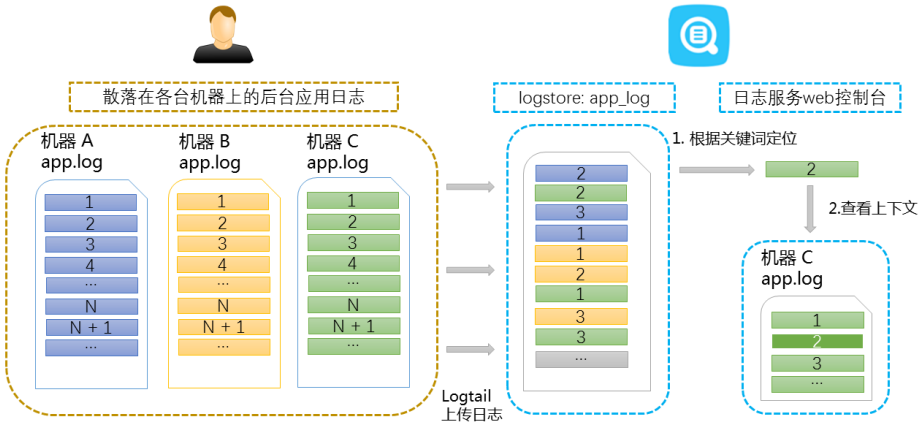
例如，O2O 外卖网站在服务器上的程序日志里会记录一次订单成交的轨迹：

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。

操作步骤

登录 日志服务管理控制台。

选择所需的项目，单击项目名称或右侧的 **管理**。

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。

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

Log Service 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，需要用引号把整个 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。
?	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城文本类型，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。

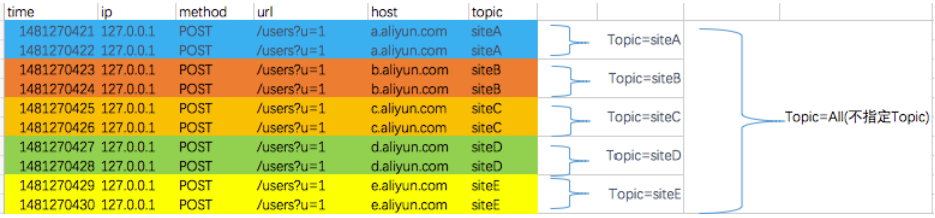
其他语法

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。

Histogram 查询

日志服务新增语法提供了自定义区间的功能。查询语法为:

```
where_condition | timeslice 1[hms] |count
h 为一小时
m 为一分钟
s 为一秒钟
```

通过变更Timeslice的参数，可以灵活的调整区间大小。例如，查询 2 个小时数据，不同的Timeslice参数对应的结果为：

Timeslice 参数	Histogram 区间数	每个区间大小
1h	2	1 小时
30m	4	30 分钟
2m	60	2 分钟
30s	240	30 秒

统计语法简介

日志服务新版本提供类似于sql的聚合计算功能，该功能结合了索引查询功能和sql的计算功能，对查询结果进行计算。

目前该同能处于试用期，免费使用。

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，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，参见语法文档 。

sql语法结构：

- 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。
- 各个子句支持的语法详见下文介绍。

SELECT子句

select 支持多个operator，每个operator之间以”,”分割。支持的operator包括：

1. 通用方法

随机返回x列中的一个值。

```
arbitrary(x)
```

样例

```
latency > 100|select arbitrary(method)
```

计算x列的算数平均值。

```
avg(x)
```

样例：

```
latency > 100 |select avg(latency)
```

计算某一列的checksum,返回hase64编码。

```
checksum(x)
```

样例：

```
latency > 100 |select checksum(method)
```

计算某一列的个数

```
count(*)
```

计算某一列非空的个数

```
count(x)
```

样例：

```
latency > 100 | count(method)
```

计算某一列的集合平均数

```
geometric_mean(x)
```

样例：

```
latency > 100 |select geometric_mean(latency)
```

返回当y取最大值时，x当前的值

```
max_by(x,y)
```

样例：查询延时最高的时候，对应的method

```
latency>100|select max_by(method,latency)
```

返回y最高的n行，对应的x的值

```
max_by(x,y,n)
```

样例：查询延时最高的3行，对应的method

```
latency > 100 |select max_by(method,latency,3)
```

返回当y取最小值时，x当前的值

```
min_by(x,y)
```

样例：查询延时最低的请求，对应的method

```
 * |select min_by(x,y)
```

返回y最小的n行，对应的x的值

```
 min_by(x,y,n)
```

样例：查询延时最小的3行，对应的method

```
* |select max_by(method,latency,3)
```

返回最大值

```
 max(x)
```

样例：

```
latency > 100|select max(inflow)
```

返回最小值

```
 min(x)
```

样例：

```
latency > 100|select min(inflow)
```

返回x列的和

```
 sum(x)
```

样例：

```
latency > 10 |select   sum(inflow)
```

对某一列的所有数值做and计算

```
bitwise_and_agg(x)
```

对某一列的数值做or计算

```
bitwise_or_agg(x)
```

2 映射

按照x的每个值group by,计算count

```
histogram(x)
```

语法相当于select count group by x。

使用样例：

```
latency > 10 | histogram(status)
```

上述语法等同于

```
latency > 10 |select count(1) group by status
```

返回key，value组成的map

```
map_agg(key, value)
```

使用样例：展示每个method的随机的latency

```
latency > 100 |select map_agg(method,latency)
```

返回key, value 组成的多value map

```
multimap_agg(key,value)
```

使用样例：返回每个method的所有的latency

```
latency > 100 |select multimap_agg(method,latency)
```

3 估算

估算x列的唯一值的个数

```
approx_distinct(x)
```

对于x列排序，找出大约处于percentage位置的数

```
approx_percentile(x,percentage)
```

比如approx_percentile(x,0.5)，找出位于一半位置的数

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, percentages)
```

用法：approx_percentile(x,array(0.1,0.2))

对于数值列，分多个桶进行统计

```
numeric_histogram(buckets, value)
```

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

样例：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)
```

4 统计方法

方法	含义	样例
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)

5 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)
```

数学函数

函数名	含义
abc(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)	把弧度转化成度
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

6 字符串函数

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)	把字符串分裂城array，最多取limit个值
strpos(string, substring)	查找字符串中的子串的开始位置
substr(string, start)	返回字符串的子串
substr(string, start, length)	返回字符串的子串
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

7 日期函数

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 (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”)

此外还支持Mysql时间格式，包括%a,%b,%v 等等... 具体请参考mysql的文档。

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 (“2017-05-17 09:45:00.848 Asia/Shanghai”,”%y-%m-%d %H:%i:%S”)
date_parse(string, format)	把string以format格式解析，转化成timestamp	latency>100| select (“2017-05-17 09:45:00”,”%y-%m-%d %H:%i:%S”)

使用时间格式的一个综合样例：

```
*|select  from_unixtime( __time__ - __time__ % 60) as t,
truncate (avg(latency) ) ,
current_date
group by __time__ - __time__ % 60
order by t desc
limit 60
```

8 select多列计算

select多个子句之间以逗号(,)分割。

样例

```
*| select min(latency),max(latency),avg(latency)
```

GROUP BY 语法

group by 支持多列

样例

```
method:PostLogstoreLogs |select avg(latency)   group by projectName,logstore
```

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)
```

HAVING

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

样例：

```
method :PostLogstoreLogs |select avg(latency),projectName group by projectName  HAVING avg(latency) > 100
```

ORDER BY

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：

```
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

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
```

一个例子

统计每5分钟的pv,uv,最高延时对应的用户请求，延时最高的10个延时

```
*|select from_unixtime(__time__ - __time__ % 300) 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 __time__ - __time__ % 300
order by time
```

使用限制

1. 为了快速返回结果，最多为每个shard计算(**search后的结果**)20万行数据或最多计算5s。

2. 如果您search结果过多，会导致计算结果不是完整的，这种情况下，在API的返回结果中，Progress为InComplete。
3. 为了您能够看到完整结果，建议优化search条件，保证命中的日志尽可能少，日志越少，响应越快。
4. 在后续的版本中，我们会逐步放开限制单次计算的数据量, 最终满足单次计算完整结果,敬请期待。

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。

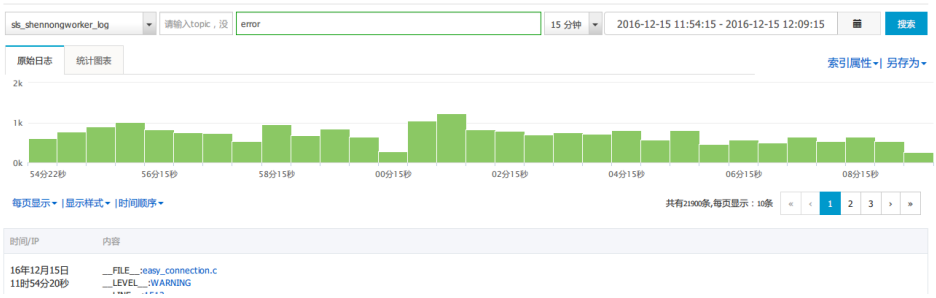
步骤 1 配置快速查询

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。

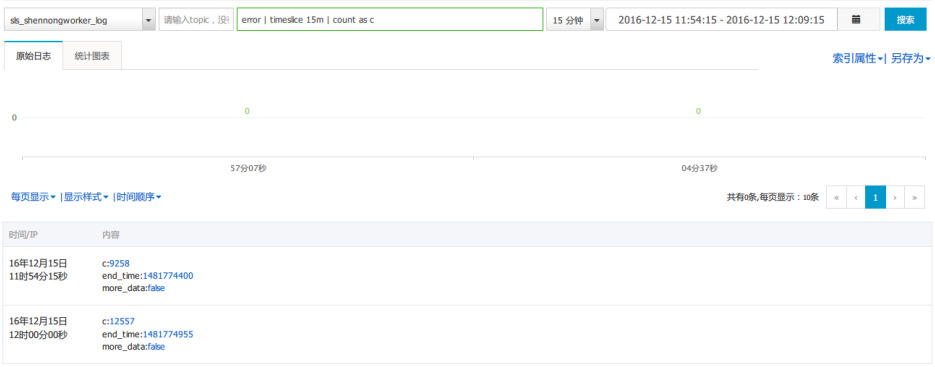
从这个例子可以看到，报警只会判断 477 条中前 10 条，存在误差。因此推荐对统计结果进行报警。

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

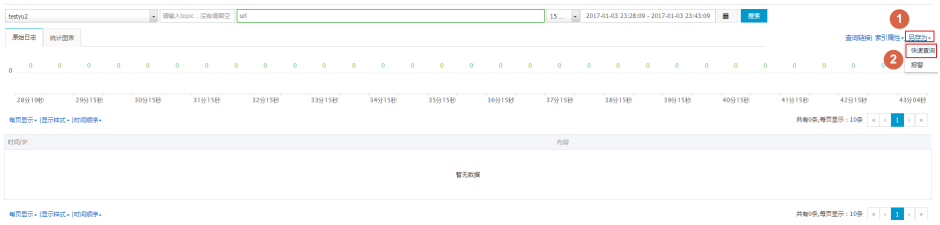
登录 日志服务管理控制台。

选择所需的项目，单击项目名称或右侧的 **管理**。

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**。

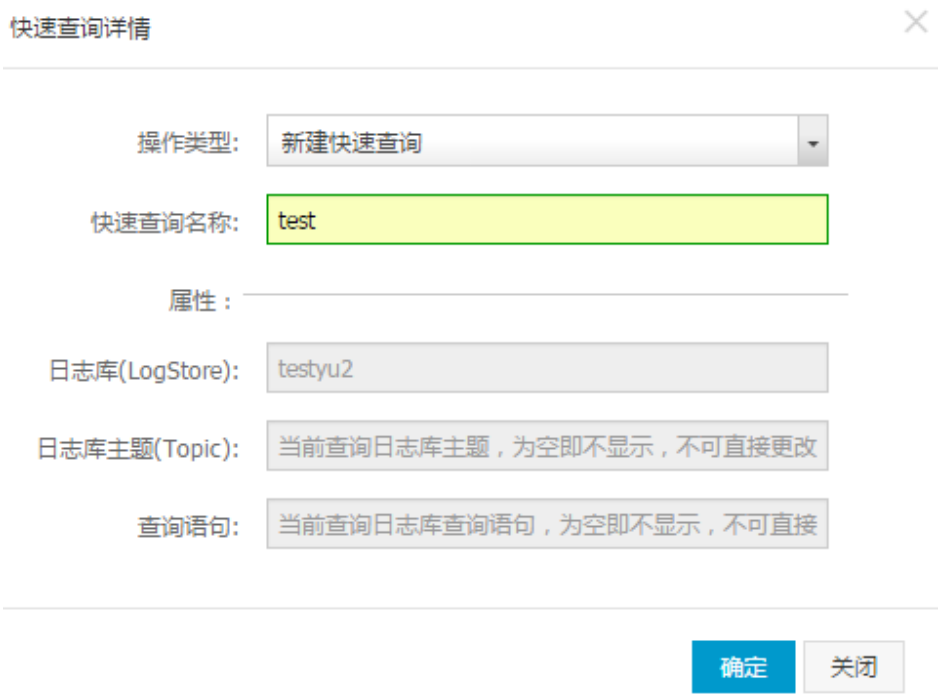
根据需求指定日志库（Logstore）、主题（Topic）和查询语句后，搜索指定日志。

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。

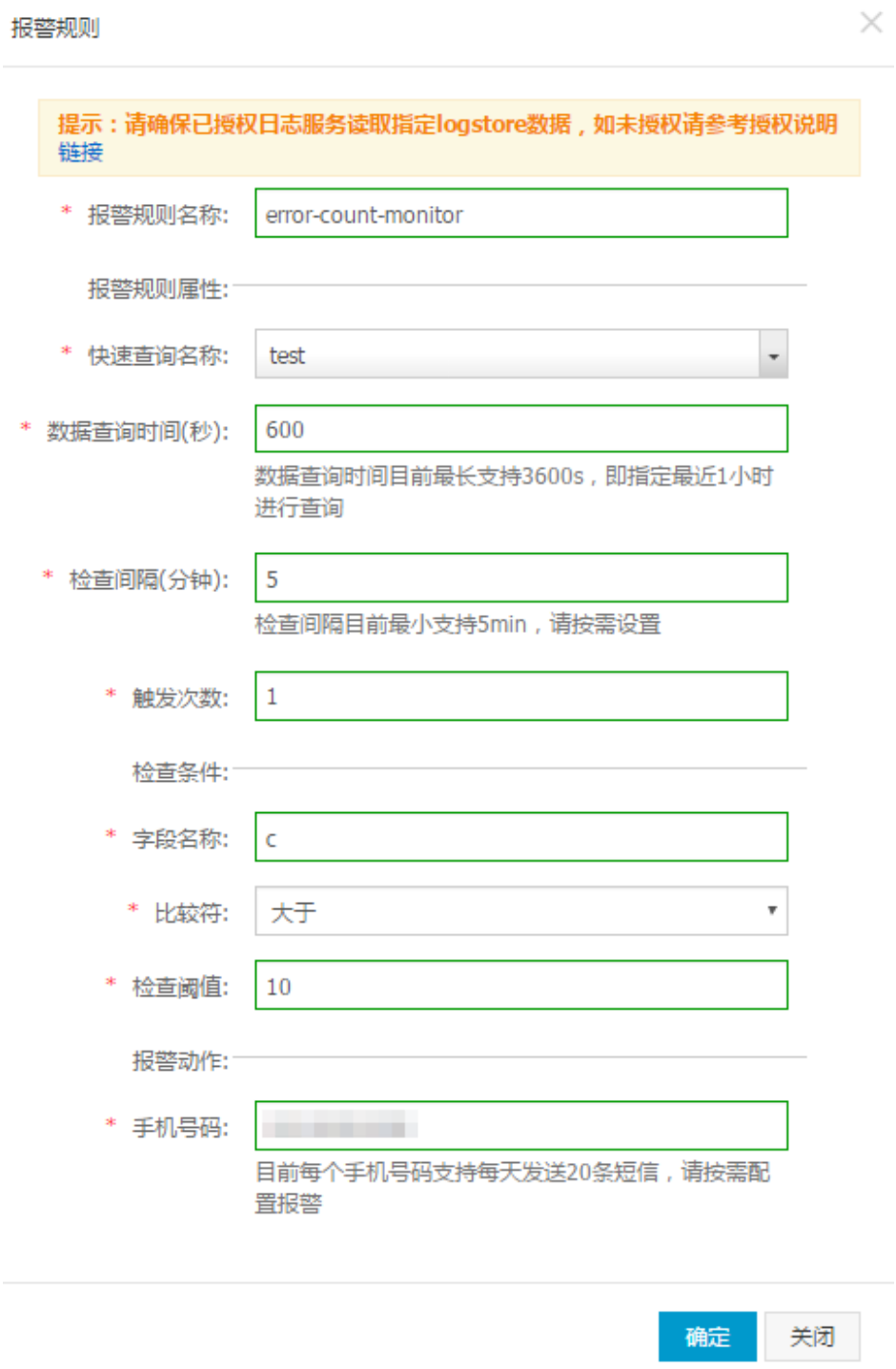


步骤 2 创建报警规则

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**。



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），如下：

操作	描述	例子
>	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，单击左侧导航栏中的 **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。

我们在二期（计划）会支持根据 MNS（消息通知服务），支持 HTTP 推送其他通知系统。

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
  - 投递 OSS/MaxCompute 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。

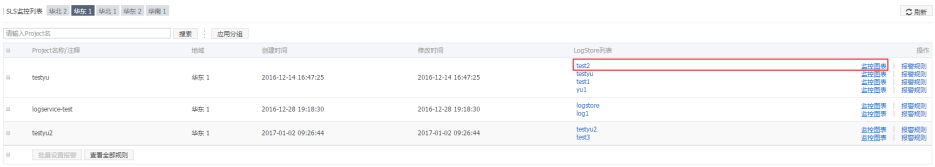
登录 日志服务管理控制台。

选择所需的项目，单击项目名称或右侧的 **管理**。

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。

选择所需的项目，选择所需的日志库并单击 **监控图表**。



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。

子用户访问日志服务控制台需要以下三步，详细说明文档参考 RAM 子帐号访问控制台。

步骤 1 创建子用户

登录 访问控制服务控制台。

单击左侧导航栏中的 **用户管理**。

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

以给子用户授予只读访问父帐号资源为例。

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。

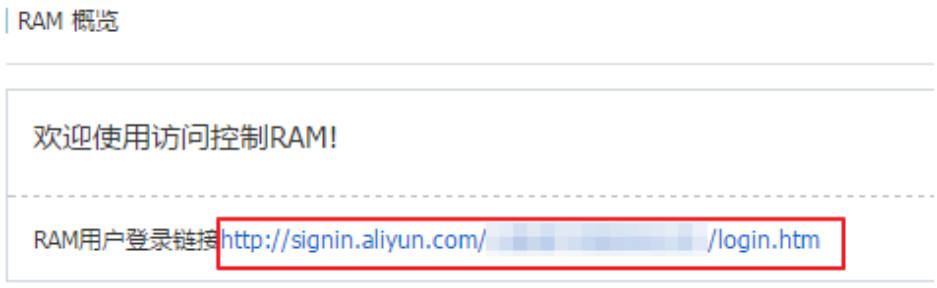
在弹出的验证对话框中，单击 **点击获取** 并输入手机验证码，然后单击 **确定**。

步骤 3 子用户登录控制台

上面两步做完之后，子用户就有权限访问日志服务控制台了。

登录方式一：

回到访问控制服务控制台概览页面，单击子用户登录链接，使用前面创建的子用户用户名和密码登录。



登录方式二：

您可以访问https://signin.aliyun.com 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。

子用户登录常见问题