ALIBABA CLOUD

阿里云

专有云企业版

日志服务
用户指南

产品版本：V3.13.0

文档版本：20240709

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
⚠ 警告	适用于可能会产生风险的场景。介绍用户在操作前就必须充分了解的信息、操作前必须要注意的事项或已具备的条件。	⚠ 警告 重启操作将导致业务短暂中断，建议您在业务低峰期执行重启操作，或确保已完成数据备份。如有必要，请联系阿里云技术支持提供协助。
ⓘ 重要	在操作前需要用户了解的提示信息、补充信息、注意事项、限制信息等。	ⓘ 重要 再次登录系统时，您需要修改登录账户的初始密码。
? 说明	用于额外的补充说明、最佳实践、窍门等，不是用户必须了解的信息。	? 说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.什么是日志服务

日志服务LOG（Log Service）是针对日志类数据的一站式服务，提供日志数据的采集、查询、分析、消费等多种功能。

日志服务在阿里巴巴集团经历大量大数据场景锤炼而成，您无需开发就能快捷完成日志数据采集、消费以及查询分析等操作，提升运维、运营效率，建立 DT 时代海量日志处理能力。

日志服务提供如下功能：

- 日志采集：支持通过Logtail客户端、JS等方式实时采集Event、Binlog、TextLog等多种格式的日志数据。
- 查询分析：对于采集到的日志数据，提供实时检索、日志分析等功能，并支持根据分析结果生成可视化图表、仪表盘。
- 状态告警：基于查询分析功能的告警服务，支持定期执行查询分析语句，当查询结果满足告警条件时，根据预先配置的告警任务发送实时告警。
- 实时消费：对于采集到服务端的日志数据提供实时消费接口。

2.快速入门

2.1. 流程

本章节为您介绍日志服务的基本操作流程，您可以参考本章节快速创建Project、Logstore、采集日志数据。

1. （可选）查看Accesskey。

   Access Key是您通过API/SDK操作日志服务的必要条件，请确保您当前使用的账号有一对AK。

2. 创建Project。

   在指定地域下创建Project并添加注释。

3. 创建Logstore。

   在已创建的Projec下创建Logstore，并指定Shard数目。

4. 采集文本日志。

   根据您的需求，选择合适的方式采集日志数据，快速入门以采集文本日志为例。

5. 开启并配置索引，查询分析日志。

   日志服务提供大规模的日志实时查询与分析能力，开启索引后您可以实时查询分析日志、并配置图表简介及仪表盘。

6. 设置告警。

   日志服务支持您基于日志查询结果配置告警，并根据您的配置以自定义webhook等方式发送告警。

7. 实时消费日志。

   日志服务支持Spark Streaming Client、Storm Sout和Flink Connector等多种方式的日志消费。

2.2. 登录日志服务控制台

本文档向您介绍如何登录日志服务控制台。

前提条件

- 登录Apsara Uni-manager运营控制台前，确认您已从部署人员处获取Apsara Uni-manager运营控制台的服务域名地址。
- 推荐使用Chrome浏览器。

操作步骤

1. 在浏览器地址栏中，输入Apsara Uni-manager运营控制台的服务域名地址，按回车键。

   > **说明**
   > - 单击右上角的当前语言可以切换其它语言。
   > - 由于浏览器证书过期或环境开启HTTPS认证等出现多次单击继续访问或无Apsara Uni-manager运营控制台地址入口等现象，此时需要安装证书才能正常访问。单击右上角的**浏览器证书安装**，可打开浏览器证书安装指南。

2. 输入正确的用户名及密码。

   请向运营管理员获取登录控制台的用户名和密码。

   > **说明**
   > 首次登录Apsara Uni-manager运营控制台时，需要修改登录用户名的密码，请按照提示完成密码修改。为提高安全性，密码长度必须为10~32位，且至少包含以下两种类型：
   > - 英文大写或小写字母（A~Z、a~z）
   > - 阿拉伯数字（0~9）
   > - 特殊符号（感叹号（!）、at（@）、井号（#）、美元符号（$）、百分号（%）等）

3. 单击**账号登录**。
4. 在顶部导航栏选择**产品 > 日志服务 SLS**。
5. 选择组织和地域后单击**SLS**，跳转到日志服务控制台。

2.3. 查看Accesskey

访问身份验证中通过使用AccessKey ID和AccessKey Secret对称加密的方法来验证某个请求的发送者身份。AccessKey ID用于标示用户，AccessKey Secret是用户用于加密签名字符串。本章节将为您描述如何获取AccessKey。

前提条件

只有运营管理员和一级组织管理员可以获取组织AccessKey。

背景信息

推荐使用个人账号AccessKey进行调用Apsara Uni-manager及云产品相关接口。如果使用个人账号AccessKey一般需要在Header中添加以下限制性参数：

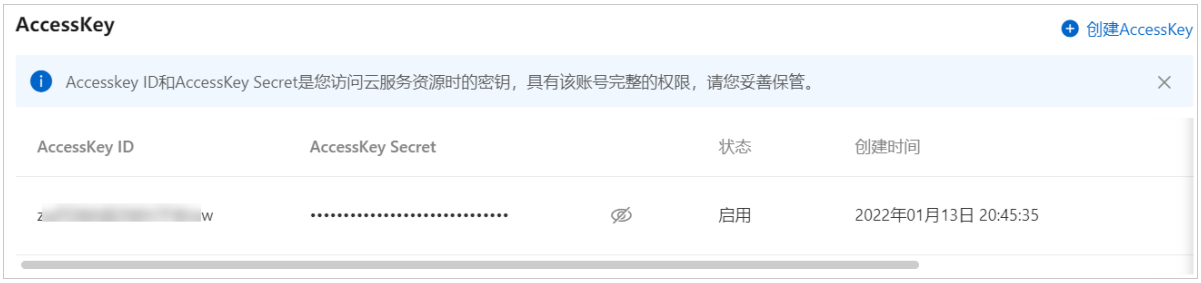
参数名称	描述
x-acs-regionid	环境Region如：cn-hangzhou-*。
x-acs-organizationid	Apsara Uni-manager体系中对应的组织ID。
x-acs-resourcegroupid	Apsara Uni-manager体系中对应的资源集ID。
x-acs-instanceid	进行操作的实例ID。

⚠ **警告** 个人账号AccessKey是由Apsara Uni-manager权限体系管控的受限AccessKey，组织AccessKey相对权限较大，需要管理员确认操作的安全性。

获取个人账号AccessKey

获取个人账号AccessKey的方法如下：

1. 登录Apsara Uni-manager运营控制台。
2. 在系统界面右上角，将鼠标悬停在当前登录用户的头像上，单击**个人信息**。
3. 在**阿里云AccessKey**区域，您可以查看个人账户的AccessKey信息。



说明

Accesskey ID和AccessKey Secret是您访问云资源时的密钥，具有该账号完整的权限，请您妥善保管。

获取组织AccessKey

获取组织AccessKey的方法如下：

1. 管理员登录Apsara Uni-manager运营控制台。
2. 在顶部菜单栏，单击**企业**。
3. 在左侧导航栏中，选择**资源管理 > 组织管理**。
4. 在组织结构中，单击目标一级组织名称。
5. 单击**管理Accesskey**。
6. 在弹出的对话框中，查看组织AccessKey信息。

2.4. 操作Project

您可以通过日志服务管理控制台对项目（Project）进行创建和删除操作。

背景信息

项目（Project）是日志服务中的资源管理单元，用于资源隔离和控制。您可以通过项目来管理某一个应用的所有日志及相关的日志源。它管理着用户的所有日志库（Logstore），采集日志的机器配置等信息，同时它也是用户访问日志服务资源的入口。具体来说，项目可以提供如下功能：

- 帮助您组织、管理不同的 Logstore。在实际使用中，您可能需要使用日志服务集中收集、存储不同项目、产品或者环境的日志。您可以把不同项目、产品或者环境的日志分类管理在不同的项目中，方便后续的日志消费、导出或者索引。同时，项目还是日志访问权限管理的载体。
- 提供您日志服务资源的访问入口。每创建一个项目，日志服务会为该项目分配一个独有的访问入口。该访问入口支持通过网络写入、读取及管理日志。

创建Project

> **说明**
> - 目前日志服务仅提供控制台方式创建Project。
> - 一个阿里云账户最多可创建50个Project。

1. 登录日志服务控制台。
2. 单击**Project列表**区域中的**创建Project**。
3. 设置Project配置项。

配置项	说明
Project名称	Project名称在所有阿里云Region内全局唯一。 其命名规则如下： 。 项目名称仅支持小写字母、数字和连字符（-）。 。 项目名称必须以小写字母或数字开头和结尾。 。 项目名称长度为3~63个字符。 **说明** 项目名称创建后不能修改。
注释	Project的简单注释，配置后将显示在**Project列表**页面。创建Project完成后可以在**Project列表**页面单击**编辑**来修改注释。 注释长度为0~64个字符，不支持 `<>'\"\\` 字符。
所属地域	请根据日志来源及其他实际情况选择合适的阿里云Region。 Project一旦创建完成则无法改变其所属Region，且日志服务不支持Project跨地域迁移，所以请谨慎选择Project的所属Region。

4. 单击**确定**。

修改Project

如果您需要修改Project的注释信息，可以通过修改Project来操作。

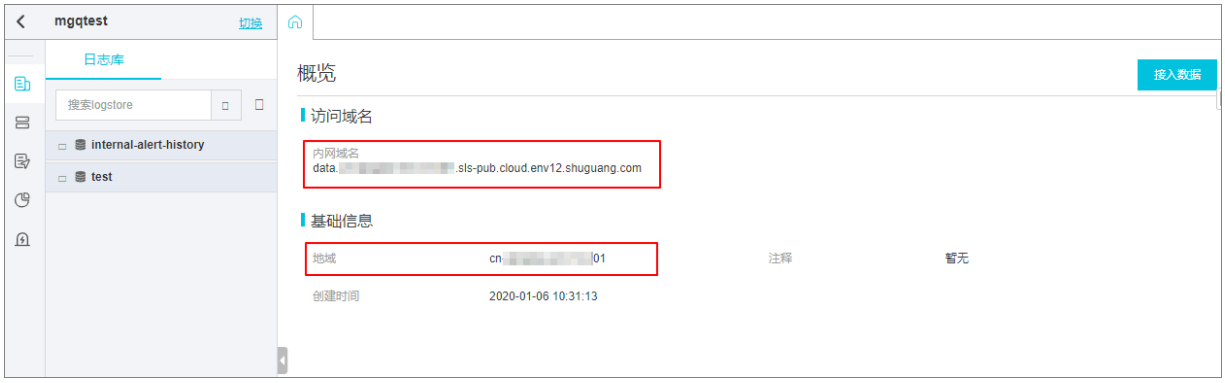
1. 在**Project列表**中，选择需要修改注释的Project。
2. 单击**操作**列的**编辑**按钮。
3. 在**修改Project**页面中修改注释信息。

> **说明** 不支持修改Project名称和所属地域。

4. 单击**确定**。

Project概览

Project概览页面展示了Project的基本信息，单击目标Project名称 ，您可以在这里查看Project的Endpoint和region。其中访问域名即为Endpoint，地域即为当前所属region。



删除Project

在某些情况下，您可能需要删除整个Project。 具体步骤如下。

> ⚠ **警告** Project被删除后，其管理的所有日志数据及配置信息都会被永久释放，不可恢复。所以在删除Project前请慎重确认，避免数据丢失。

1. 在**Project列表**中，选择需要删除的Project。
2. 单击**操作**列的**删除**按钮。
3. 在弹出的对话框中选择删除原因。
   如果选择**其他原因**，请在下方文本框中详细说明。

图 1. 删除Project

4. 单击**确定**。

2.5. 操作Logstore

日志库（Logstore）是创建在项目（Project）下的资源集合，Logstore中的所有数据都来自于同一个数据源。

背景信息

您可以根据实际需求为某一个项目生成多个日志库，其中常见的做法是为一个应用中的每类日志创建一个独立的日志库。
具体来说，Logstore 提供如下功能：

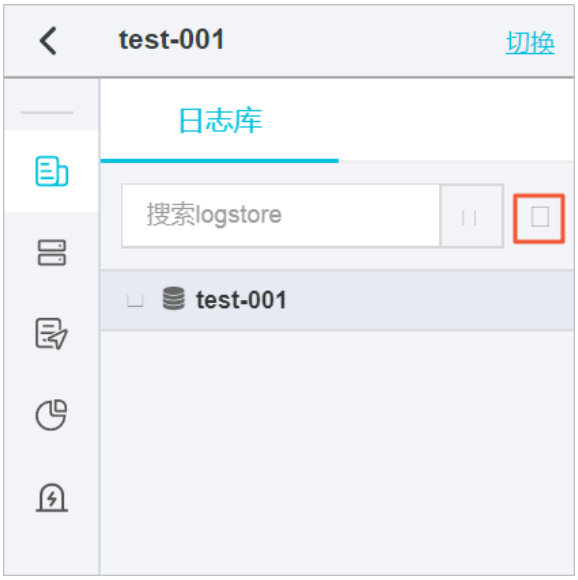
- 收集日志，支持实时日志写入。
- 存储日志，支持实时消费。
- 建立索引，支持日志实时查询。

创建Logstore

> **说明** 每个日志服务Project可创建最多100个Logstore。

1. 登录日志服务控制台。
2. 单击目标Project。

3. 单击日志库搜索框后的 创建Logstore。



4. 填写日志库的配置信息并单击**确定**。

配置项	说明
Logstore名称	Logstore名称在其所属项目内必须唯一。 。 日志库名称仅支持小写字母、数字、连字符（-）和下划线（_）。 。 必须以小写字母或者数字开头和结尾。 。 名称长度为3~63字符。 说明 日志库名称创建后不能修改。
WebTracking	确认是否开启WebTracking功能。WebTracking功能支持从HTML、H5、iOS或Android平台收集日志数据到日志服务。默认关闭。
永久保存	确认是否永久保存日志数据。 关闭后需要设置日志数据的保存时间。
数据保存时间	日志数据在日志库中的保存时间，单位为天。可以设置为1~3000天。超过该时间后，日志数据会被删除。
Shard数目	日志库的分区数量，每个Logstore可以创建1~10个分区。
自动分裂shard	确认是否开启自动分裂shard功能，默认为开启状态。 当数据量超过已有分区（Shard）的服务能力后，开启自动分裂功能可自动根据数据量增加分区数量。
最大分裂数	开启**自动分裂shard**后，Shard自动分裂的最大数目，最大为64。
记录外网IP	开启该功能后，日志服务接收到日志数据后，自动把以下信息添加到日志的Tag字段中。 。 `__client_ip__`：日志来源设备的公网IP地址。 。 `__receive_time__`：日志到达服务端的时间，格式为Unix时间戳。

修改Logstore配置

日志服务支持对已经创建的Logstore进行修改。

1. 单击目标Project名称。
2. 单击目标日志库名称后的 ，选择**修改**。

3. 单击**Logstore属性**右上角的**修改**。
   您可以选择修改**数据保存时间**、**WebTracking**、**自动分裂shard**、**最大分裂数**、**记录外网IP**，以及对当前Shard进行分裂合并操作。

4. 单击**保存**完成修改。

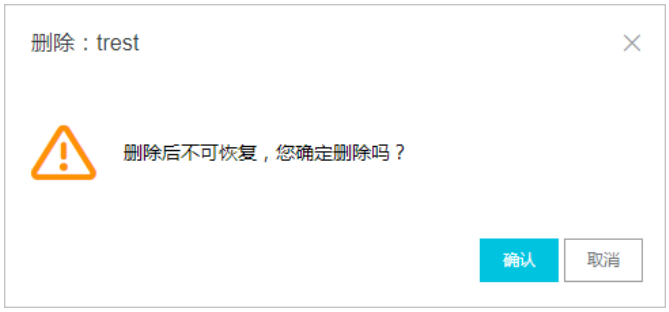
删除Logstore

假如您不再需要某个Logstore，您可以在控制台上删除该Logstore。

> **说明**
> - Logstore一旦删除，其存储的日志数据将会被永久删除，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。

1. 单击目标Project名称。
2. 单击目标日志库名称后的，选择**删除**。

3. 在弹出的确认对话框中，单击**确认**。



2.6. 操作Shard

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，您在创建Logstore时需要为该Logstore设置Shard的数量，创建完成后还可以分裂或合并Shard，以达到增加或减少Shard的目的。

Shard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。

如分区示例所示，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。 表 1. 分区

示例

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。

Shard读写能力

每个分区可提供一定的服务能力，建议您根据实际数据流量规划分区个数，流量超出读写能力时，及时分裂分区以增加分区个数，从而达到更大的读写能力；如您的流量远远达不到分区的最大读写能力时，建议您合并分区以减少分区个数，从而节约分区租赁费用。

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

> **说明**
> - 当写入的API持续报告403或者500错误时，通过数据流量判断是否需要增加分区。
> - 对超过分区服务能力的读写，系统会尽可能服务，但不保证服务质量。

分区的状态

分区的状态包括：

- readwrite：可以读写
- readonly：只读数据

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

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

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

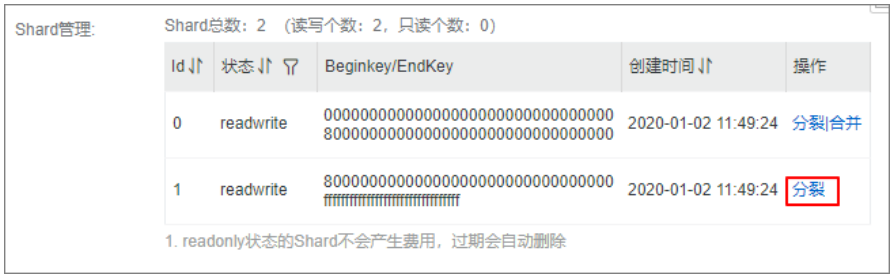
通过日志服务云控制台您可以进行以下分区操作：

- 扩容分区
- 缩容分区

分裂Shard

每个分区（Shard）能够处理5MB/s的数据写入和10MB/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。

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project名称。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore右侧的图标，选择**修改**。
4. 单击**Logstore属性**右上角的**修改**按钮。
5. 选择要分裂的分区，单击右侧的**分裂**。



6. 选择分区的分裂数量。
7. 单击**确定**进行分裂。
8. 单击**Logstore属性**右上角的**保存**按钮完成分裂操作。

自动分裂Shard

除手动分裂Shard之外，日志服务还支持自动分裂Shard功能。

开启自动分裂Shard功能后，当满足以下两个条件时Shard会自动分裂：

- 数据量超出当前已有Shard的服务能力且持续5分钟以上。
- Logstore中readwrite状态的Shard数目未超过设定的最大shard总数。

> **说明** 最近15分钟内分裂出来的新Shard不会自动分裂。

您可以在创建或修改Logstore时开启该功能，并设定Shard自动分裂后的最大数目。具体请参见操作Logstore。

- 自动分裂shard

  Shard自动分裂功能开关。开启该功能后，Shard会在数据量超出Shard服务能力时自动分裂。

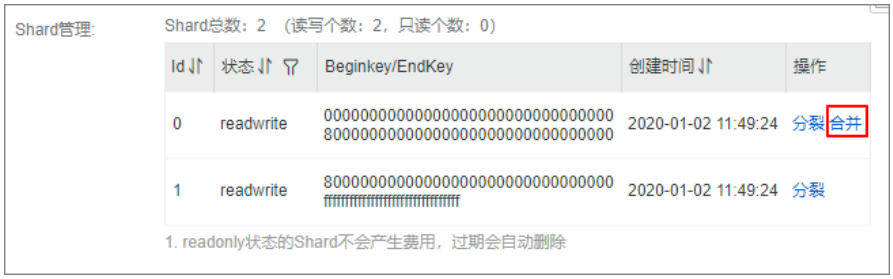
- 最大分裂数

  Shard自动分裂后的最大数目。开启自动分裂Shard功能后，最大可支持自动分裂至64个分区。

合并Shard

您可以通过合并（merge）操作缩容分区。

1. 单击**Logstore属性**右上角的**修改**按钮。
2. 选择要合并的分区，单击右侧的**合并**。



3. 单击**Logstore属性**右上角的**保存**按钮完成合并操作。

删除Shard

- 自动删除

  如果您在创建Logstore时设置了数据保存时间，那么分区及分区中的日志数据超出设置的时间后会被自动删除。

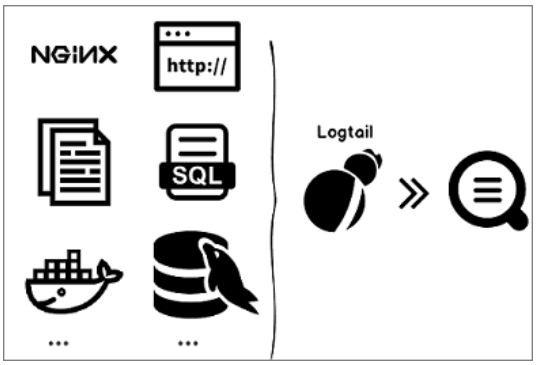
- 手动删除

  如果您在创建Logstore时开起了永久保存，那么这部分Shard和日志数将不会自动被删除。如果想要删除，建议您通过删除Logstore的方式删除整个日志库中的分区和数据。具体请参见删除Logstore。

3.数据采集
3.1. Logtail采集
3.1.1. 简介
3.1.1.1. Logtail简介

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、自建IDC、其他云厂商等服务器上的日志。

图 1. Logtail采集功能



功能优势

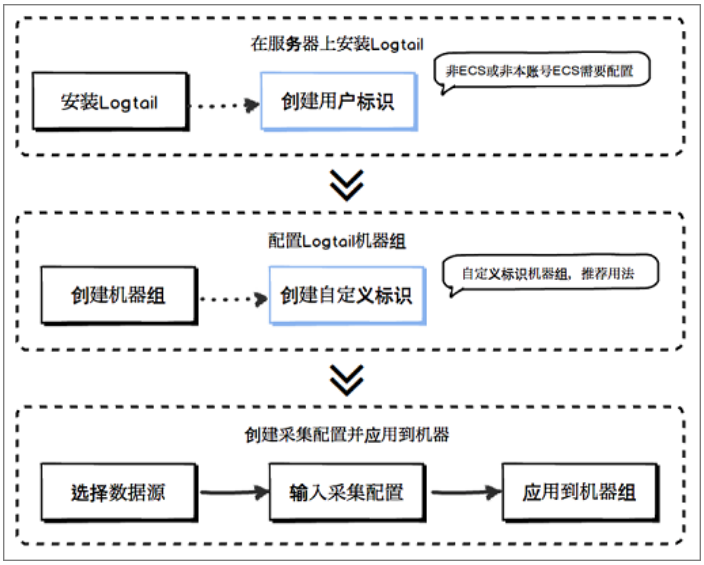
- 基于日志文件、无侵入式的收集日志。用户无需修改应用程序代码，且日志收集不会影响用户应用程序的运行逻辑。
- 除支持文本日志采集外，还支持binlog、http、容器stdout等采集方式。
- 对于容器支持友好，支持标准容器、swarm集群、Kubernetes集群等容器集群的数据采集。
- 能够稳定地处理日志收集过程中各种异常。当遇到网络异常、服务端异常等问题时会采用主动重试、本地缓存数据等措施保障数据安全。
- 基于服务端的集中管理能力。用户在安装Logtail后，只需要在服务端集中配置需要收集的机器、收集方式等信息即可，无需逐个登录服务器进行配置。
- 完善的自我保护机制。为了尽可能降低Logtail对于相关服务器性能的影响，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

请参见相关限制说明。

配置流程

图 2. 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（Linux系统）和安装Logtail（Windows系统）。
2. 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IP和自定义标识两种方式定义机器组。您也可以在应用Logtail配置到机器组时，根据提示创建机器组。
3. 创建Logtail采集配置，并应用到机器组。您可以通过创建Logtail采集配置以 文本日志采集流程等，并将该Logtail配置应用到机器组。

在完成如上流程后，您的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见Logtail采集日志 。

容器

- 阿里云容器服务Kubernetes、自建Kubernetes：请参见Kubernetes日志采集流程。
- 自建其他Docker集群：请参见标准Docker日志采集流程。

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Logtail配置到机器组，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执行日志收集工作的Agent。安装步骤请参见安装Logtail（Linux系统）和安装Logtail（Windows系统）。 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，并启动两个以 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）下。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。Logtail配置定义了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：

功能	说明

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的延迟一般在3秒内。 **说明** 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12小时的会被丢弃。
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。
多种采集输入源	Logtail除支持文本日志采集外，还支持syslog、http、MySQL binlog等输入源。
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。
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。您也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。 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。
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密钥并对所有发送日志的数据包进行数据签名。 **说明** Logtail客户端在获取您的阿里云访问密钥时采用HTTPS通道，保障您的访问密钥安全性。

数据采集可靠性

Logtail在采集数据时，会定期将采集的点位（CheckPoint）信息保存到本地，若遇到服务器意外关闭、进程崩溃等异常情况时，Logtail重启后会从上一次记录的位置处开始采集数据，尽可能保证数据不丢失。Logtail会根据配置文件中的资源限制进行工作，若资源占用超过限定值5分钟以上，则Logtail会强制重启。重启后可能会产生一定的数据重复。

Logtail内部采用了很多机制提升日志采集可靠性，但并不能保证日志一定不会丢失。以下情况可能造成日志丢失：

- Logtail未运行且日志轮转多次。
- 日志轮转速度极快，例如1秒轮转1次。
- 日志采集速度长期无法达到日志产生速度。

3.1.1.2. Logtail采集流程

使用Logtail客户端采集服务器日志时，Logtail采集日志的流程为监听文件、读取文件、处理日志、过滤日志、聚合日志和发送数据6个步骤。

说明 将Logtail采集配置应用到机器组之后，机器组中服务器上没有发生修改事件的日志文件会被判定为历史文件。Logtail在正常运行模式中不支持采集历史文件，若您需要采集历史日志，请参见导入历史日志文件。

监听文件

在服务器上安装Logtail客户端，并根据数据源添加Logtail采集配置之后，Logtail采集配置从服务端实时下发到Logtail。Logtail根据采集配置开始监听文件。

1. Logtail根据配置的日志路径和最大监控目录深度逐层扫描目录下符合指定文件名规则的日志目录和文件。

为保证日志采集时效性以及稳定性，Logtail会对采集目录注册事件监听（Linux下使用Inotify、Windows下使用ReadDirectoryChangesW）以及定期轮询。

2. Logtail如果监听到指定目录下符合规则的日志文件在应用配置之后没有修改过，则不会采集；如果有日志文件产生了修改事件，会触发采集流程，Logtail开始读取文件。

读取文件

确定日志文件有更新后，Logtail开始读取文件。

- 若该文件首次读取，会检查文件大小。
  - 如果文件小于1 MB，则从文件内容起始位置开启读取。
  - 如果文件大于1 MB，则从文件末尾1MB处开始读取。
- 如果该文件曾被Logtail读取过，则从上次读取的Checkpoint处继续读取。
- 读取文件时，每次最多可以读取512KB，因此每条日志请控制在512KB以内，否则无法正常读取。

> **说明** 如果您修改了服务器上的时间，请手动重启Logtail，否则会导致日志时间不正确、意外丢弃日志等现象。

处理日志

logtail读取日志后，对日志内容进行分行、解析，并确认日志时间字段。

- 分行

  如果Logtail采集配置中指定了**行首正则**，则根据行首配置对Logtail每次读取的日志数据块进行分行，切分成多条日志；如果没有指定，则将一个数据块作为一条日志处理。

- 解析
  根据Logtail采集配置，对每条日志内容执行对应的解析，例如正则、分隔符、JSON等。

  > **说明** 如果您的正则式较为复杂，可能会导致CPU占用率过高，请使用合理高效的正则表达式。

- 解析失败处理

  根据Logtail采集配置中是否开启丢弃解析失败日志功能，判断日志解析失败的处理方式。

  - 开启**丢弃解析失败日志**，则直接丢弃该日志，并上报解析失败的报错信息。
  - 关闭 **丢弃解析失败日志**，则上传解析失败的原始日志，其中Key为**raw_log**、Value为日志内容。

- 设置日志时间字段
  - 若未配置时间字段，则日志时间为当前解析时间。
  - 若配置了时间字段：
    - 日志记录的时间距离当前时间12小时以内，则从解析的日志字段中提取时间。
    - 日志记录的时间距离当前时间12小时以上，则丢弃该日志并上传错误信息。

过滤日志

处理日志后，根据Logtail采集配置中的过滤器配置过滤日志。

- 未设置**过滤器配置**：不过滤日志，执行下一个步骤。
- 已设置**过滤器配置**：对每条日志中的所有字段进行遍历并验证。
  - 符合过滤器配置的日志：如果日志中出现了过滤器中配置的所有字段且所有对应的字段全部符合配置，则采集该条日志。
  - 不符合过滤器配置的日志：不会被采集。

聚合日志

通过过滤器配置过滤日志后，符合配置的日志数据将发往日志服务。为降低网络请求次数，当日志处理、过滤完毕后，会在Logtail内部缓存一段时间，进行聚合打包，再发送到日志服务。

缓存过程中，如果满足以下条件之一，日志将即时打包发送到日志服务。

- 日志聚合时间超过3秒。
- 日志聚合条数超过4096条。
- 日志聚合总大小超过512 KB。

发送日志

Logtail将采集到的日志数据聚合发送到日志服务。设置配置启动参数中的参数 `max_bytes_per_sec` 和 `send_request_concurrency` 可以调整日志数据的发送速度和最大并发数，Logtail会保证发送速率以及并发不超过配置值。

若数据发送失败，Logtail自动根据错误信息决定重试或放弃发送。

错误信息	说明	Logtail处理方式
401错误	Logtail客户端没有权限采集数据。	直接丢弃日志包。
404错误	Logtail采集配置中指定的Project或Logstore不存在。	直接丢弃日志包。
403错误	Shard Quota超出限制。	等待3秒后重试。
500错误	服务端异常。	等待3秒后重试。
网络超时	网络连接错误。	等待3秒后重试。

3.1.1.3. Logtail配置和记录文件

Logtail运行时会依赖一系列的配置文件并产生部分信息记录文件，本文档介绍常见文件的基本信息及路径。

配置文件请参见：

- 启动配置文件（ilogtail_config.json）
- AliUid配置文件
- 用户自定义标识文件（user_defined_id）
- 采集配置文件（user_log_config.json）

记录文件请参见：

- AppInfo记录文件（app_info.json）
- Logtail运行日志（ilogtail.LOG）
- Logtail插件日志（logtail_plugin.LOG）
- 容器路径映射文件（docker_path_config.json）

启动配置文件（ilogtail_config.json）

启动配置文件（ilogtail_config.json）用来查看或配置Logtail的运行参数，文件类型为JSON。

安装Logtail后，您可以通过该文件：

- 修改Logtail的运行参数。

  可以通过修改启动配置文件来修改CPU使用阈值、常驻内存使用阈值等配置信息。

- 检验安装命令是否正确。

  该文件中的 `config_server_address` 和 `data_server_list` 取决于安装时选择的参数和安装命令，如果其中的区域和日志服务所在区域不一致或地址无法联通，说明安装时选择了错误的参数或命令。这时Logtail无法正常采集日志，需要重新安装。

> **说明**
> - 该文件必须为合法JSON，否则无法启动Logtail。
> - 修改该文件后需重启Logtail才能生效。

默认配置项如下，您也可以参见配置启动参数写入其他配置。

表 1. 启动配置文件默认配置项

配置项	说明
config_server_address	Logtail从服务端获取配置文件的地址，取决于安装时选择的参数和安装命令。 请保证该地址能够联通，且其中的区域和日志服务所在区域一致。
data_server_list	数据服务器地址，取决于安装时选择的参数和安装命令。 请保证该地址能够联通，且其中的区域和日志服务所在区域一致。
cluster	区域名称。
endpoint	服务入口。请在Project概览中查看Endpoint。

cpu_usage_limit	CPU使用阈值，以单核计算。
mem_usage_limit	常驻内存使用阈值。
max_bytes_per_sec	Logtail发送原始数据的流量限制，超过20 MB/s则不限流。
process_thread_count	Logtail处理日志文件写入数据的线程数。
send_request_concurrency	异步并发的个数。Logtail默认异步发送数据包，如果写入TPS很高，可以配置更高的异步并发。

- 文件地址
  - Linux：/usr/local/ilogtail/ilogtail_config.json。
  - 容器：该文件存储在Logtail容器中，文件地址配置在Logtail容器的环境变量 `ALIYUN_LOGTAIL_CONFIG` 中，可通过 `docker inspect ${logtail_container_name} | grep ALIYUN_LOGTAIL_CONFIG` 查看，例如/etc/ilogtail/conf/cn-hangzhou/ilogtail_config.json。
  - Windows：
    - x64：C:\Program Files (x86)\Alibaba\Logtail\ilogtail_config.json。
    - x32：C:\Program Files\Alibaba\Logtail\ilogtail_config.json。
- 文件示例

```
$cat /usr/local/ilogtail/ilogtail_config.json
{
    "config_server_address" : "http://logtail.cn-hangzhou-intranet.log.aliyuncs.com",
    "data_server_list" :
    [
        {
            "cluster" : "ap-southeast-2",
            "endpoint" : "cn-hangzhou-intranet.log.aliyuncs.com"
        }
    ],
    "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
}
```

AliUid配置文件

AliUid配置文件中包含阿里云账号的AliUid账号信息，主要用于标识这台服务器有权限被该账号访问、采集日志。采集非本账号ECS、自建IDC的日志时，需要手动创建AliUid配置文件。

> **说明**
> - 该文件为可选配置，仅在采集非本账号ECS、自建IDC日志时使用。
> - AliUid文件必须为主账号AliUid，不支持子账号。
> - AliUid文件只需配置文件名即可，文件不能有后缀。
> - 一个Logtail可配置多个AliUid文件，Logtail容器仅可配置一个AliUid文件。

- 文件地址
  - Linux：`/etc/ilogtail/users/`。
  - 容器：该文件存储在一个Logtail容器内。该文件地址在Logtail容器的环境变量 `ALIYUN_LOGTAIL_USER_ID` 中指定，您可以执行 `docker inspect ${logtail_container_name} | grep ALIYUN_LOGTAIL_USER_ID` 命令查看该文件路径。
  - Windows：`C:\LogtailData\users\`。
- 文件示例

```
$ls /etc/ilogtail/users/
*************** *****************
```

用户自定义标识文件（user_defined_id）

用户自定义标识文件（user_defined_id）用于配置自定义标识机器组，详细说明以及配置参见创建用户自定义标识机器组。

> **说明**
> - 该文件为可选配置，只有在配置自定义标识机器组时使用。
> - 若配置多个自定义标识，使用换行符分隔。

- 文件地址
  - Linux：/etc/ilogtail/user_defined_id。
  - 容器：该文件存储在一个Logtail容器内。文件地址在Logtail容器的环境变量 `ALIYUN_LOGTAIL_USER_DEFINED_ID` 中指定，您可执行docker inspect ${logtail_container_name} | grep ALIYUN_LOGTAIL_USER_DEFINED_ID命令查看该文件路径。
  - Windows：C:\LogtailData\user_defined_id。
- 文件示例

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

采集配置文件（user_log_config.json）

该文件记录Logtail从服务端获取的采集配置信息，文件类型为JSON，每次配置更新时会同步更新该文件。可通过该文件确认Logtail配置是否已经下发到该服务器。采集配置文件存在，且内容为最新，表示Logtail配置已下发。

> **说明**
> - 除手动配置密钥信息、数据库密码等敏感信息外，不建议修改该文件。
> - 提交工单时，请上传此文件。

- 文件地址
  - Linux：/usr/local/ilogtail/user_log_config.json。
  - 容器：/usr/local/ilogtail/user_log_config.json。
  - Windows
    - x64：C:\Program Files (x86)\Alibaba\Logtail\user_log_config.json。
    - x32：C:\Program Files\Alibaba\Logtail\user_log_config.json。
- 文件示例

```
$cat /usr/local/ilogtail/user_log_config.json
{
   "metrics" : {
      "##1.0##k8s-log-c12ba2028*****939f0b$app-java" : {
         "aliuid" : "16542189*****50",
         "category" : "app-java",
         "create_time" : 1534739165,
         "defaultEndpoint" : "cn-hangzhou-intranet.log.aliyuncs.com",
         "delay_alarm_bytes" : 0,
         "enable" : true,
         "enable_tag" : true,
         "filter_keys" : [],
         "filter_regs" : [],
         "group_topic" : "",
         "local_storage" : true,
         "log_type" : "plugin",
         "log_tz" : "",
         "max_send_rate" : -1,
         "merge_type" : "topic",
         "plugin" : {
            "inputs" : [
               {
                  "detail" : {
                     "IncludeEnv" : {
                        "aliyun_logs_app-java" : "stdout"
                     },
                     "IncludeLable" : {
                        "io.kubernetes.container.name" : "java-log-demo-2",
                        "io.kubernetes.pod.namespace" : "default"
                     },
                     "Stderr" : true,
                     "Stdout" : true
                  },
                  "type" : "service_docker_stdout"
               }
            ]
         },
         "priority" : 0,
         "project_name" : "k8s-log-c12ba2028c*****ac1286939f0b",
         "raw_log" : false,
         "region" : "cn-hangzhou",
         "send_rate_expire" : 0,
         "sensitive_keys" : [],
         "tz_adjust" : false,
         "version" : 1
      }
   }
}
```

AppInfo记录文件（app_info.json）

AppInfo记录文件（app_info.json）记录Logtail的启动时间、获取到的IP地址、hostname等信息。配置IP地址机器组时，需要在该文件中查看Logtail获取到的IP地址。

通常情况下，Logtail根据以下规则获取服务器IP地址：

- 如果已在服务器文件/etc/hosts中设置了主机名与IP地址绑定，则自动获取绑定的IP地址。
- 如果没有设置主机名绑定，会自动获取本机的第一块网卡的IP地址。

> **说明**
> - AppInfo记录文件仅用于记录Logtail内部信息，手动修改文件内容不能改变Logtail的基本信息。
> - 若修改了服务器的hostname等网络配置，请重新启动Logtail以获取新的IP地址。

表 2. 字段说明

字段	说明
UUID	服务器序列号。
hostname	主机名。
instance_id	随机生成的Logtail唯一标识。
ip	Logtail获取到的IP地址。该字段为空时表示Logtail没有获取到IP地址，Logtail无法正常运行。请为服务器设置IP地址并重启Logtail。 说明 如果机器组为IP地址机器组，请确保机器组中配置的IP与此处显示的IP地址一致。若服务端机器组填写了错误的IP地址，请修改机器组内IP地址并保存，等待1分钟再查看。
logtail_version	Logtail客户端版本。
os	操作系统版本。
update_time	Logtail最近一次启动时间。

- 文件地址
  - Linux：/usr/local/ilogtail/app_info.json。
  - 容器：/usr/local/ilogtail/app_info.json。
  - Windows
    - x64：C:\Program Files (x86)\Alibaba\Logtail\app_info.json。
    - x32：C:\Program Files\Alibaba\Logtail\app_info.json。
- 文件示例

```
$cat /usr/local/ilogtail/app_info.json
{
   "UUID" : "",
   "hostname" : "logtail-ds-slpn8",
   "instance_id" : "E5F93BC6-B024-11E8-8831-0A58AC14039E_1**.***.***.***_1536053315",
   "ip" : "1**.***.***.***",
   "logtail_version" : "0.16.13",
   "os" : "Linux; 3.10.0-693.2.2.el7.x86_64; #1 SMP Tue Sep 12 22:26:13 UTC 2017; x86_64",
   "update_time" : "2018-09-04 09:28:36"
}
```

Logtail运行日志（ilogtail.LOG）

Logtail运行日志（ilogtail.LOG）记录Logtail客户端的运行信息，日志级别从低到高分别为`INFO`、`WARN`和`ERROR`，其中`INFO`类型日志无需关注。

- 文件地址
  - Linux：/usr/local/ilogtail/ilogtail.LOG。
  - 容器：/usr/local/ilogtail/ilogtail.LOG。
  - Windows
    - x64：C:\Program Files (x86)\Alibaba\Logtail\logtail_*.log。
    - x32：C:\Program Files\Alibaba\Logtail\logtail_*.log。
- 文件示例

```
$tail /usr/local/ilogtail/ilogtail.LOG
[2018-09-13 01:13:59.024679]    [INFO]    [3155]    [build/release64/sls/ilogtail/elogtail.cpp:123]
change working dir:/usr/local/ilogtail/
[2018-09-13 01:13:59.025443]    [INFO]    [3155]    [build/release64/sls/ilogtail/AppConfig.cpp:175]
load logtail config file, path:/etc/ilogtail/conf/ap-southeast-2/ilogtail_config.json
[2018-09-13 01:13:59.025460]    [INFO]    [3155]    [build/release64/sls/ilogtail/AppConfig.cpp:176]
load logtail config file, detail:{
   "config_server_address" : "http://logtail.ap-southeast-2-intranet.log.aliyuncs.com",
   "data_server_list" : [
      {
         "cluster" : "ap-southeast-2",
         "endpoint" : "ap-southeast-2-intranet.log.aliyuncs.com"
      }
]
```

Logtail插件日志（logtail_plugin.LOG）

Logtail插件日志（logtail_plugin.LOG）记录容器标准输出、binlog、http等插件的运行信息，日志级别从低到高分别为 `INFO` 、 `WARN` 和 `ERROR` ，其中 `INFO` 类型日志无需关注。

- 文件地址
  - Linux：/usr/local/ilogtail/logtail_plugin.LOG。
  - 容器：/usr/local/ilogtail/logtail_plugin.LOG。
  - Windows：不支持插件功能。
- 文件示例

```
$tail /usr/local/ilogtail/logtail_plugin.LOG
2018-09-13 02:55:30 [INF] [docker_center.go:525] [func1] docker fetch all:start
2018-09-13 02:55:30 [INF] [docker_center.go:529] [func1] docker fetch all:stop
2018-09-13 03:00:30 [INF] [docker_center.go:525] [func1] docker fetch all:start
2018-09-13 03:00:30 [INF] [docker_center.go:529] [func1] docker fetch all:stop
2018-09-13 03:03:26 [INF] [log_file_reader.go:221] [ReadOpen] [##1.0##sls-zc-test-hz-pub$docker-stdout
-config,k8s-stdout]    open file for read,
file:/logtail_host/var/lib/docker/containers/7f46afec6a14de39b59ee9cdfbfa8a70c2fa26f1148b2e2f31bd3410f5b
24/7f46afec6a14de39b59ee9cdfbfa8a70c2fa26f1148b2e2f31bd3410f5b2d624-json.log    offset:40379573    sta
tus:794354-64769-40379963
2018-09-13 03:03:26 [INF] [log_file_reader.go:221] [ReadOpen] [##1.0##k8s-log-
c12ba2028cfb444238cd9ac1286939f0b$docker-stdout-config,k8s-stdout]    open file for read, file:/logtai
l_host/var/lib/docker/containers/7f46afec6a14de39b59ee9cdfbfa8a70c2fa26f1148b2e2f31bd3410f5b2d624/7f46af
a14de39b59ee9cdfbfa8a70c2fa26f1148b2e2f31bd3410f5b2d624-json.log    offset:40379573    status:794354-6
4769-40379963
2018-09-13 03:04:26 [INF] [log_file_reader.go:308] [CloseFile] [##1.0##sls-zc-test-hz-pub$docker-stdou
t-config,k8s-stdout]    close file, reason:no read timeout
file:/logtail_host/var/lib/docker/containers/7f46afec6a14de39b59ee9cdfbfa8a70c2fa26f1148b2e2f31bd3410f5b
24/7f46afec6a14de39b59ee9cdfbfa8a70c2fa26f1148b2e2f31bd3410f5b2d624-json.log    offset:40379963    sta
tus:794354-64769-40379963
2018-09-13 03:04:27 [INF] [log_file_reader.go:308] [CloseFile] [##1.0##k8s-log-
c12ba2028cfb444238cd9ac1286939f0b$docker-stdout-config,k8s-stdout]    close file, reason:no read timeo
ut
file:/logtail_host/var/lib/docker/containers/7f46afec6a14de39b59ee9cdfbfa8a70c2fa26f1148b2e2f31bd3410f5b
24/7f46afec6a14de39b59ee9cdfbfa8a70c2fa26f1148b2e2f31bd3410f5b2d624-json.log    offset:40379963    sta
tus:794354-64769-40379963
2018-09-13 03:05:30 [INF] [docker_center.go:525] [func1] docker fetch all:start
2018-09-13 03:05:30 [INF] [docker_center.go:529] [func1] docker fetch all:stop
```

容器路径映射文件（docker_path_config.json）

容器路径映射文件（docker_path_config.json）只有在采集容器文件时才会自动创建，用于记录容器文件和实际文件的路径映射关系。文件类型为JSON。

> **说明** 该文件为信息记录文件，任何修改操作均不会生效；删除后会自动创建，不影响业务的正常运行。

- 文件地址

/usr/local/ilogtail/docker_path_config.json。

- 文件示例

```
$cat /usr/local/ilogtail/docker_path_config.json
{
   "detail" : [
      {
         "config_name" : "##1.0##k8s-log-c12ba2028cfb444238cd9ac1286939f0b$nginx",
         "container_id" : "df19c06e854a0725ea7fca7e0378b0450f7bd3122f94fe3e754d8483fd330d10",
         "params" : "{\n   \"ID\" :
\"df19c06e854a0725ea7fca7e0378b0450f7bd3122f94fe3e754d8483fd330d10\",\n   \"Path\" : \"/logtail_host/v
ar/lib/docker/overlay2/947db346695a1f65e63e582ecfd10ae1f57019a1b99260b6c83d00fcd1892874/diff/var/log\",\
\"Tags\" : [\n      \"nginx-type\",\n      \"access-log\",\n      \"_image_name_\",\n
\"registry.cn-hangzhou.aliyuncs.com/log-service/docker-log-test:latest\",\n
\"_container_name_\",\n      \"nginx-log-demo\",\n      \"_pod_name_\",\n      \"nginx-log-demo-h2lzc\
",\n      \"_namespace_\",\n      \"default\",\n      \"_pod_uid_\",\n      \"87e56ac3-b65b-11e8-b172-
00163f008685\",\n      \"_container_ip_\",\n      \"172.20.4.224\",\n      \"purpose\",\n
\"test\"\n   ]\n}\n"
      }
   ],
   "version" : "0.1.0"
}
```

3.1.2. 安装

3.1.2.1. 安装Logtail（Linux系统）

Logtail客户端是日志服务提供的日志采集客户端，本文介绍如何在Linux服务器上安装Logtail客户端。

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
- RedHat

安装步骤

> **说明** Logtail采用覆盖安装模式，若您之前已安装过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并注册开机启动。

1. 执行如下命令下载Logtail安装器。

```
wget http://${service:sls-backend-server:sls_data.endpoint}/logtail.sh -O logtail.sh; chmod 755
logtail.sh
```

> **说明** 命令中 `${service:sls-backend-server:sls_data.endpoint}` 请替换为真实的Endpoint，您可以在Project概览中查看Endpoint信息。

2. 执行安装命令。
   启动Shell终端，以管理员权限执行以下命令，安装Logtail。

```
./logtail.sh install
```

3. 配置用户标识。

查看Logtail版本

Logtail会将版本信息记录在 `/usr/local/ilogtail/app_info.json` 中的 `logtail_version` 字段，例如：

```
$cat /usr/local/ilogtail/app_info.json
{
   "UUID" : "0DF18E97-0F2D-486F-B77F-*********",
   "hostname" : "david*******",
   "instance_id" : "F4FAFADA-F1D7-11E7-846C-00163E30349E_*********_1515129548",
   "ip" : "**********",
   "logtail_version" : "0.16.0",
   "os" : "Linux; 2.6.32-220.23.2.ali1113.el5.x86_64; #1 SMP Thu Jul 4 20:09:15 CST 2013; x86_64",
   "update_time" : "2018-01-05 13:19:08"
}
```

升级Logtail

您可以通过Logtail安装器（logtail.sh）来进行Logtail的升级，安装器会根据已经安装的Logtail配置信息自动选择合适的方式进行升级。

> **说明** 升级过程中会短暂停止Logtail，但升级只会覆盖必要的文件，配置文件以及Checkpoint文件将会被保留。升级期间日志不会丢失。

执行以下命令升级Logtail：

```
# 下载安装器
wget http://${service:sls-backend-server:sls_data.endpoint}/logtail.sh -O logtail.sh; chmod 755
logtail.sh
# 执行升级命令
sudo ./logtail.sh upgrade
```

执行结果：

```
# 升级成功
Stop logtail successfully.
ilogtail is running
Upgrade logtail success
{
   "UUID" : "***",
   "hostname" : "***",
   "instance_id" : "***",
   "ip" : "***",
   "logtail_version" : "0.16.11",
   "os" : "Linux; 3.10.0-693.2.2.el7.x86_64; #1 SMP Tue Sep 12 22:26:13 UTC 2017; x86_64",
   "update_time" : "2018-08-29 15:01:36"
}
# 升级失败：已经是最新版本
[Error]:    Already up to date.
```

手动启动和停止Logtail

- 启动
  以管理员身份执行：

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

- 停止
  以管理员身份执行：

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

卸载Logtail

在Shell下以管理员身份执行以下命令，卸载Logtail。

```
wget http://${service:sls-backend-server:sls_data.endpoint}/logtail.sh -O logtail.sh
chmod 755 logtail.sh
./logtail.sh uninstall
```

3.1.2.2. 安装Logtail（Windows系统）

Logtail客户端是日志服务提供的日志采集Agent，请参考本文档，在Windows服务器上安装Logtail客户端。

支持的系统

Windows版Logtail客户端支持以下操作系统：

- Windows 7 (Client) 32bit
- Windows 7 (Client) 64bit
- Windows Server 2008 32bit
- Windows Server 2008 64bit
- Windows Server 2012 64bit
- Windows Server 2016 64bit

安装步骤

1. 下载安装包。
   执行如下命令下载安装包。

   ```
   wget http://${service:sls-backend-server:sls_data.endpoint}/windows/logtail_installer.zip
   ```

   > **说明** 命令中 `${service:sls-backend-server:sls_data.endpoint}` 请替换为真实的Endpoint，您可以在Project概览中查看Endpoint信息。

2. 解压缩 `logtail_installer.zip` 到当前目录。
3. 执行安装命令。
   以管理员身份运行Windows Powershell或cmd，进入 `logtail_installer` 目录，根据网络类型执行安装命令。

   ```
   ./logtail_installer.exe. install ${region}
   ```

   > **说明** 命令中 `${region}` 需要替换为实际信息，您可以在Project概览中查看当前region信息。

4. 配置用户标识。

安装路径

执行安装命令后，Logtail默认安装到指定路径下，不支持修改和变更。在该路径下可以通过文件app_info.json查看Logtail版本，或在安装路径下卸载Logtail。

安装路径如下：

- 32位Windows系统：C:\Program Files\Alibaba\Logtail
- 64位Windows系统：C:\Program Files (x86)\Alibaba\Logtail

> **说明** Windows 64位操作系统支持运行32/64位应用程序，但是出于兼容性考虑，在 64 位操作系统上，Windows会使用单独的x86目录来存放32位应用程序。
>
> Windows Logtail是32位程序，所以在64位操作系统上的安装目录为Program Files (x86)。后续日志服务如果推出64位Windows Logtail，会自动安装到Program Files目录下。

查看Logtail版本

Logtail会自动安装到默认目录中，您可以进入该目录，使用记事本或其他文本编辑器打开文件app_info.json，其中的 `logtail_version` 字段即为您当前安装的Logtail的版本号。

例如，以下内容表示Logtail的版本号为1.0.0.0：

```
{
    "logtail_version" : "1.0.0.0"
}
```

升级Logtail

- 自动升级

  通常情况下，Windows版Logtail支持自动升级。但将1.0.0.0之前的旧版升级为1.0.0.0及以上版本时，必须手动升级。

- 手动升级

  将1.0.0.0之前的旧版升级为1.0.0.0及以上版本时，必须手动升级。手动升级的步骤和安装步骤相同，您只需要下载并解压最新的安装包，然后按照步骤执行安装即可。

  > **说明** 手动升级相当于自动卸载并重新安装，所以会删除掉您原先安装目录中的内容，如有必要，请您在执行手动升级前做好备份工作。

手动启动和停止Logtail

打开**控制面板**中的**管理工具**，打开**服务**。

根据您所安装的版本找到对应的服务：

- 0.x.x.x版本：LogtailWorker服务。
- 1.0.0.0及以上版本：LogtailDaemon服务。

执行对应操作：

- 手动启动

  右键单击**启动**。

- 停止

  右键单击**停止**。

- 重启

  右键单击**重新启动**。

卸载Logtail

以管理员身份运行Windows Powershell或cmd进入 `logtail_installer` 目录，即您所下载安装包的解压目录，执行命令：

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

卸载成功后，您的Logtail安装目录会被完全删除，但是仍有部分配置会被保留在 C:\LogtailData目录中，您可以根据实际情况进行手动删除。遗留配置信息包括：

- checkpoint：保存所有插件（例如Windows event log插件）的checkpoint信息。
- logtail_check_point：保存Logtail主体部分的checkpoint信息。
- users：保存所配置的用户标识（Aliuid）信息。

3.1.2.3. 配置启动参数

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。

背景信息

配置Logtail启动配置参数适用于以下场景：

- 需要采集的日志文件数目大，占用大量内存。内存中需要维护每个文件的签名、采集位置、文件名等meta信息。
- 日志数据流量大导致CPU占用率高。
- 日志数据量大导致发送到日志服务的网络流量大。
- 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	说明	示例
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 。
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/Sec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20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。
streamlog_pool_size_in_mb	syslog用于接收日志的内存池大小，用于缓存syslog数据。单位是 MB。	程序启动时会申请内存，请根据机器内存以及实际需求填写。
streamlog_rcv_size_each_call	Logtail每次调用linux socket rcv 接口使用的缓冲区大小，单位为byte，取值范围为1024~8192。	如果syslog流量大，可以调高该值。
streamlog_formats	定义接收到的syslog日志解析方式。	--
streamlog_tcp_addr	Logtail用于接收syslog日志的绑定地址，默认为0.0.0.0。	--
streamlog_tcp_port	Logtail用于接收syslog日志的TCP端口。	默认为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。	默认为25。
buffer_file_size	该参数用于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。	默认为20971520 Byte（20MB） 。

buffer_file_path	该参数用于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。
check_point_filename	checkpoint文件保存的全路径，用于自定义Logtail的checkpoint保存位置。	默认情况下为/tmp/logtail_check_point。建议Docker用户修改此文件保存地址，并将checkpoint所在目录挂载到宿主机，否则容器释放时会因丢失checkpoint信息而产生重复采集。例如Docker中配置check_point_filename为 `/data/logtail/check_point.dat` ，Docker启动命令增加 `-v /data/docker1/logtail:/data/logtail` ，将宿主机/data/docker1/logtail目录挂载到Docker中的/data/logtail目录。

> **说明**
> - 这里只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。
> - 请根据需要新增或修改指定配置参数，不必要的配置项不需要增加到ilogtail_config.json中。

修改配置

1. 根据需要求配置 ilogtail_config.json。

   请确认修改配置后，配置内容为合法JSON。

2. 重启Logtail使配置生效。

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

3.1.3. Logtail机器组

3.1.3.1. 简介

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

机器组是包含多台服务器的虚拟分组。如果您有一台以上的服务器，并希望这些服务器使用同样的Logtail配置采集日志，可以将这些服务器加入同一个机器组，并将Logtail配置应用到该机器组。

您可以通过如下两种方法定义一个机器组：

- IP地址：在机器组中添加服务器的IP地址，通过IP地址识别服务器。
- 自定义标识：定义属于机器组的一个标识，在对应服务器上配置对应标识进行关联。

> **说明** 请勿将Windows服务器和Linux服务器添加到同一机器组中。

IP地址机器组

您可以通过添加服务器IP地址的方式，直接将多台服务器添加到一个机器组中，为其统一进行Logtail配置。

- 若您使用ECS服务器，且没有绑定过hostname、没有更换过网络类型，可以在机器组中配置ECS服务器的私网IP地址。
- 其他情况下，请在机器组中配置Logtail自动获取到的IP地址，该IP地址记录在服务器文件app_info.json的ip字段中。

  > **说明** app_info.json是记录Logtail内部信息的文件，其中包括Logtail自动获取到的IP地址，手动修改该文件的ip字段不能改变Logtail获取的IP地址。

Logtail自动获取服务器IP地址的方式：

- 如果已在服务器文件/etc/hosts中设置了主机名与IP地址绑定，则自动获取绑定的IP地址。
- 如果没有设置主机名绑定，会自动获取本机的第一块网卡的IP地址。

创建IP地址机器组，请参见创建IP地址机器组。

自定义标识机器组

除IP地址外，您还可以使用自定义标识（userdefined-id）来动态定义机器组。

通常情况下，系统由多个模块组成，每个模块都可以进行单独的水平扩展，即支持添加多台服务器。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机器组标识。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。

创建自定义标识机器组，请参见创建用户自定义标识机器组。

3.1.3.2. 创建IP地址机器组

日志服务支持创建IP地址机器组。将Logtail获取到的服务器IP地址添加进IP地址机器组中，则可以使用同样的Logtail配置采集这些服务器的日志。

前提条件

- 已创建Project和Logstore。
- 已有一台及以上的服务器并获取到服务器的IP地址。
- 已为服务器安装了Logtail。详细步骤请参见安装Logtail（Linux系统）和安装Logtail（Windows系统）。
- 已配置AliUid（用户标识），详细步骤请参见配置用户标识。

操作步骤

1. 登录日志服务控制台。
2. 单击目标Project。
3. 单击左侧导航栏的**机器组**展开机器组列表。
4. 单击**机器组**后的图标，选择**创建机器组**。

   您也可以在数据接入向导中创建机器组。
5. 创建机器组。
   i. 填写**机器组名称**。
   名称只能包含小写字母、数字、连字符（-）和下划线（_）且必须以小写字母或数字开头和结尾，长度为3~128字符。

   > **说明** 不支持修改机器组名称，请谨慎填写。

   ii. 选择**机器组标识**为**IP地址**。
   iii. 填写**机器组Topic**。
   机器组Topic的详细信息请参见生成主题。
   iv. 填写**IP地址**。
   请填写服务器IP地址。

   > **说明**
   > - 机器组中存在多台服务器时，IP地址之间请用换行符分割。
   > - 请勿将Windows服务器和Linux服务器添加到同一机器组中。
6. 单击**确定**。

执行结果

您可以在机器组列表中查看刚创建的机器组。

3.1.3.3. 创建用户自定义标识机器组

除IP地址外，您还可以使用用户自定义标识（Custom ID）来动态定义机器组。

背景信息

自定义标识机器组在以下场景中具有明显优势：

- VPC等自定义网络环境中，可能出现不同机器IP地址冲突的问题，导致服务端无法管理Logtail。使用自定义标识可以避免此类情况的发生。
- 多台服务器通过同一个自定义标识实现机器组弹性伸缩。您只需为新增的服务器配置相同的自定义标识，服务端可自动识别，并将其添加至机器组中。

操作步骤

1. 在服务器上设置用户自定义标识。
   - **Linux Logtail**

     通过文件 `/etc/ilogtail/user_defined_id` 来设置用户自定义标识。

     例如，设置用户自定义标识为 `userdefined` ，执行如下命令：

     ```
     # vim /etc/ilogtail/user_defined_id
     ```

     然后在该文件中输入 `userdefined` 。
   - **Windows Logtail**

     通过文件 `C:\LogtailData\user_defined_id` 来设置用户自定义标识。

     例如，设置用户自定义标识如下：

     ```
     C:\LogtailData>more user_defined_id
     userdefined_windows
     ```

     > **说明**
     > - 同一机器组中不允许同时存在Linux和Windows服务器，请勿在Linux和Windows服务器上配置同样的用户自定义标识。
     > - 一个服务器可配置多个用户自定义标识，标识之间以换行符分割。
     > - 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。
2. 创建机器组。
   i. 登录日志服务控制台。
   ii. 单击目标Project。
   iii. 单击左侧导航栏中的**机器组**。
   iv. 单击**机器组**后的图标，选择**创建机器组**。

v. 填写机器组配置。

- **名称**：填写机器组名称。

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

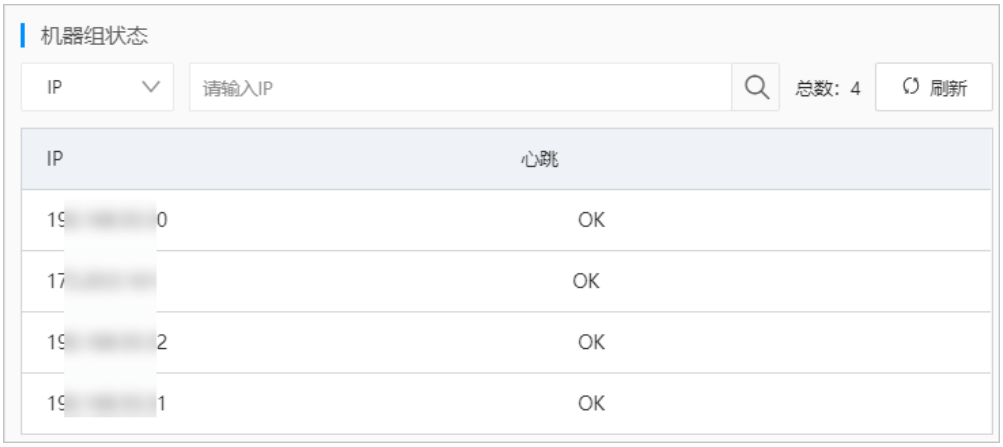
  > **说明** 不支持修改机器组名称，请谨慎填写。

- **机器组标识**：选择**用户自定义标识**。
- **机器组Topic**：填写机器组Topic，详细信息请参见生成主题。
- **用户自定义标识**：填写步骤一中创建的用户自定义标识。

vi. 单击**确定**完成配置。

> **说明** 需要扩容服务器时，只需要在新服务器上设置用户自定义标识，然后再机器组状态中就可以看到新增的服务器。

3. 查看机器组状态。
   在机器组列表页面，双击机器组名称，在右侧**机器组配置**页面查看机器组状态，可以看到使用相同用户自定义标识的服务器列表及其心跳状态。



> **说明**
>
> - 机器组状态中的IP列表，即为使用相同用户自定义标识的服务器的IP地址。例如：
>
>   假设当前为用户自定义标识机器组，用户自定义标识为userdefined，机器组状态中的IP分别为10.10.10.10、10.10.10.11、10.10.10.12。则表示您在这三个服务器上创建了相同的用户自定义标识userdefined。如果您需要新增10.10.10.13服务器，则只需要在该服务器上创建用户自定义标识userdefined，即可在机器组状态中看到该服务器10.10.10.13，日志服务会使用userdefined机器组的采集配置来采集新增服务器上的日志。
>
> - 心跳状态代表服务器与日志服务的连接是否正常，如果异常请参见Logtail 机器无心跳处理。

禁用用户自定义标识

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

- Linux系统

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

- Windows系统

```
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系统

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

- Windows系统

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

示例

系统通常由多个模块组成，每个模块可以包含多台服务器。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，每个模块可以进行单独的水平扩展，因此在新增服务器时需要能够对其进行实时日志采集。

1. 创建自定义标识。
   安装完成Logtail客户端后，为服务器开启用户自定义标识。示例场景中的模块可以分成4类机器标识：http_module、cache_module、logic_module和store_module。
2. 创建机器组。
   创建机器组时，**用户自定义标识**请填写机器组名称对应的用户自定义标识。

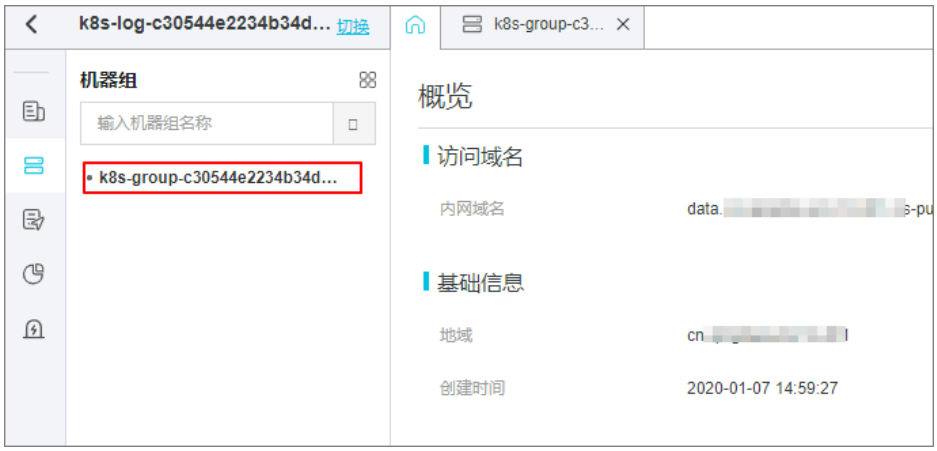
3. 查看机器组状态。
   可以在**机器组状态**中查看使用相同自定义标识的服务器列表及其心跳状态。
4. 若前端模块增加服务器10.1.1.3，只需在新服务器上开启用户自定义标识http_module。成功执行操作后可以在**机器组状态**中看到新增的服务器。

3.1.3.4. 查看机器组列表

如需查看当前项目下已创建的机器组，可以在**机器组列表**中查看。

操作步骤

1. 登录日志服务控制台。
2. 选择所需的项目，单击项目名称。
3. 单击左侧导航栏的**机器组**展开机器组列表。
   您可以查看该项目下的所有机器组。



3.1.3.5. 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。

操作步骤

1. 登录日志服务控制台。
2. 选择所需的项目，单击项目名称。
3. 单击左侧导航栏的**机器组**展开机器组列表 。
4. 找到需要修改的机器组，在**机器组配置**右上方单击**修改**。

> **说明** 机器组名称创建后不可更改。

5. 修改机器组的配置信息并单击**保存**。

3.1.3.6. 查看机器组状态

为验证 Logtail 客户端已经在机器组内的所有云主机安装成功，您可以查看 Logtail 客户端的心跳信息。

操作步骤

1. 登录日志服务控制台。
2. 选择所需的项目，单击项目名称。
3. 单击左侧导航栏的**机器组**展开机器组列表 。
4. 单击机器组名称，在**机器组配置**中查看机器组状态。
   - 心跳状态为**OK**表示所有云主机上的 Logtail 客户端都安装成功，Logtail与日志服务连接正常。
   - 心跳状态为**FAIL**表示Logtail连接异常。如果服务器心跳状态始终显示为**FAIL**，请参见页面提示及Logtail 机器无心跳进行排查。

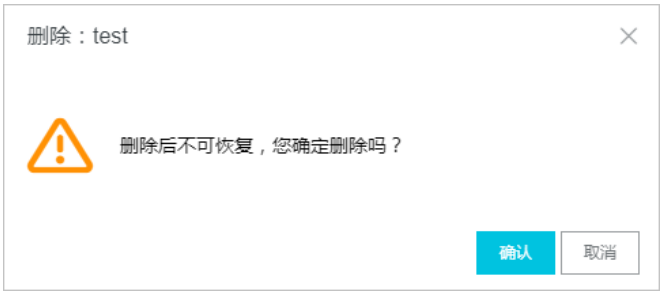
3.1.3.7. 删除机器组

当您不需要采集整个机器组的日志时，可以选择删除机器组。

操作步骤

1. 登录日志服务控制台。
2. 选择所需的项目，单击项目名称。
3. 单击左侧导航栏的**机器组**展开机器组列表。
4. 单击对应机器组后的图标，选择**删除**。

5. 在弹出的对话框中单击**确认**。



3.1.3.8. 管理机器组配置

日志服务通过机器组的方式管理所有需要通过Logtail客户端收集日志的ECS云服务器。您可以通过日志服务创建机器组，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。

背景信息

日志服务利用机器组管理所有需要收集日志的云主机，这其中的一个重要管理项目就是Logtail客户端的收集配置。您可以通过给一个机器组应用、删除Logtail配置来决定每台云主机上的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。

操作步骤

1. 登录日志服务控制台。
2. 选择所需的项目，单击项目名称。
3. 单击左侧导航栏的**机器组**展开机器组列表。
4. 选择需要修改的机器组，在**机器组配置**右上方单击 **修改**。
5. 在**管理配置区域**修改应用到机器组的Logtail配置，然后单击**保存**。
   当添加Logtail配置时，该Logtail配置就会下发到机器组内云主机的Logtail客户端。当移除Logtail配置时，该Logtail配置会从Logtail客户端移除。

3.1.3.9. 管理采集配置

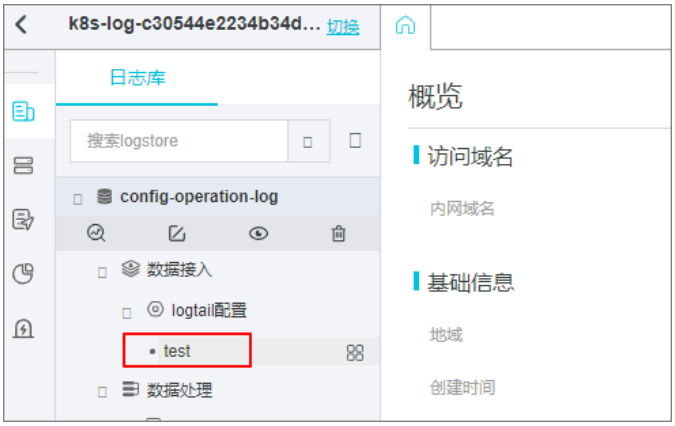
本文介绍如何在日志服务控制台上创建、查看、修改及删除Logtail采集配置等操作。

创建Logtail配置

有关如何通过日志服务控制台创建Logtail配置，参见使用文本日志采集流程。

查看Logtail配置列表

1. 登录日志服务控制台。
2. 单击目标Project。
3. 在日志库列表页面，单击日志库名称前的尖括号，依次展开**数据接入** > **Logtail配置**，每一项代表不同的配置列表项。



修改Logtail配置

单击需要修改的Logtail配置的名称，在**Logtail配置**页面右上方单击**修改**。

您也可以修改日志的收集模式并重新应用机器组。整个配置的修改流程与创建完全相同。

删除Logtail配置

选择需要删除的Logtail配置并单击右侧的图标，然后选择**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。

3.1.3.10. 配置用户标识

本文介绍如何在服务器上配置阿里云主账号ID为用户标识。

前提条件

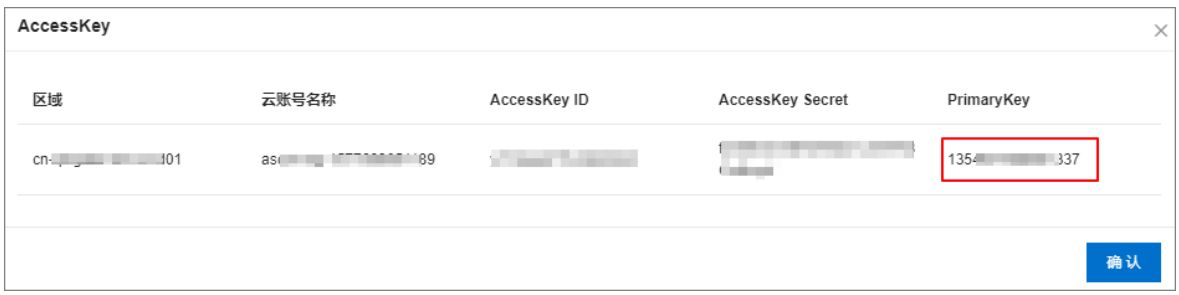
已在服务器上安装Logtail，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。

背景信息

如果您的服务器是与日志服务属于不同账号的ECS、其他云厂商的服务器和自建IDC时，您要通过Logtail采集该服务器日志，需要在服务器上安装Logtail后，配置阿里云主账号ID为用户标识，表示该账号有权限通过Logtail采集该服务器日志。否则在机器组中会显示服务器心跳失败，导致Logtail无法采集日志到日志服务。

操作步骤

1. 查看阿里云主账号ID。
   i. 登录Apsara Uni-manager控制台。
   登录方法请参见登录日志服务控制台。
   ii. 在顶部菜单栏中单击**企业**。
   iii. 在左侧导航栏中选择**组织管理**。
   iv. 选择目标账号，单击**获取accesskey**。
   v. 在**AccessKey**中查看主账号ID。



2. 登录服务器，配置主账号ID作为用户标识。
   - Linux系统

     创建主账号ID同名文件到/etc/ilogtail/users目录，如目录不存在请手动创建目录。一台服务器上可以配置多个主账号ID，例如：

```
touch /etc/ilogtail/users/1**************
touch /etc/ilogtail/users/1**************
```

当不需要Logtail收集数据到该用户的日志服务Project后，可使用如下命令删除该用户标识：

```
rm /etc/ilogtail/users/1**************
```

- Windows系统

  创建主账号ID同名文件到目录C:\LogtailData\users以配置主账号ID作为用户标识。如需删除该用户标识，请直接删除此文件。

  例如，C:\LogtailData\users\1**************。

  > **说明**
  > - 服务器配置主账号ID作为用户标识后，表示该账号有权限通过Logtail收集该服务器日志数据。请及时清理服务器上多余的主账号ID用户标识文件。
  > - 新增、删除主账号ID用户标识后，1分钟之内即可生效。

3.1.4. 文本日志

3.1.4.1. 文本日志采集流程

本文介绍创建Logtail采集配置的流程。

前提条件

已安装Logtail。Logtail支持Windows和Linux两大操作系统，安装方法参见安装Logtail（Linux系统）和安装Logtail（Windows系统）。

使用限制

- 一个文件只能被一个配置采集。如果需要采集多份，建议以软链接的形式实现。例如 `/home/log/nginx/log` 下需要采集两份，则其中一个配置原始路径，然后创建一个该文件夹的软链接 `ln -s /home/log/nginx/log /home/log/nginx/link_log` ，另一个配置软链接路径即可。
- Logtail客户端支持的操作系统请参见Logtail简介。

日志收集配置流程

通过控制台配置Logtail收集文本日志，可以通过极简模式、NGINX配置模式，APACHE配置模式，IIS配置模式，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。

图 1. 配置流程

采集模式

Logtail支持极简模式、完整正则模式、分隔符模式、JSON模式、NGINX配置模式 、IIS配置模式、APACHE配置模式等方式收集日志。

- 极简模式

  支持通过极简模式采集日志，详细采集步骤请参见极简模式采集日志。
- 完整正则模式

  支持通过完整正则模式采集日志，详细采集步骤请参见正则模式采集日志。
- 分隔符模式

  支持通过分隔符模式采集分隔符日志，请参见分隔符日志。
- JSON模式

  关于JSON日志的采集，请参见JSON 日志。
- NGINX配置模式

  关于Nginx日志的采集，请参见Nginx 日志。

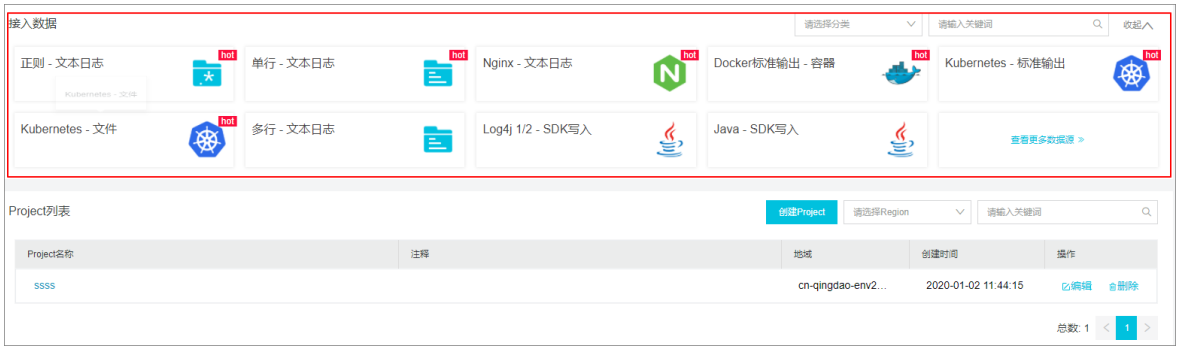
- IIS配置模式

  关于IIS日志的采集，请参见IIS 日志。
- APACHE配置模式

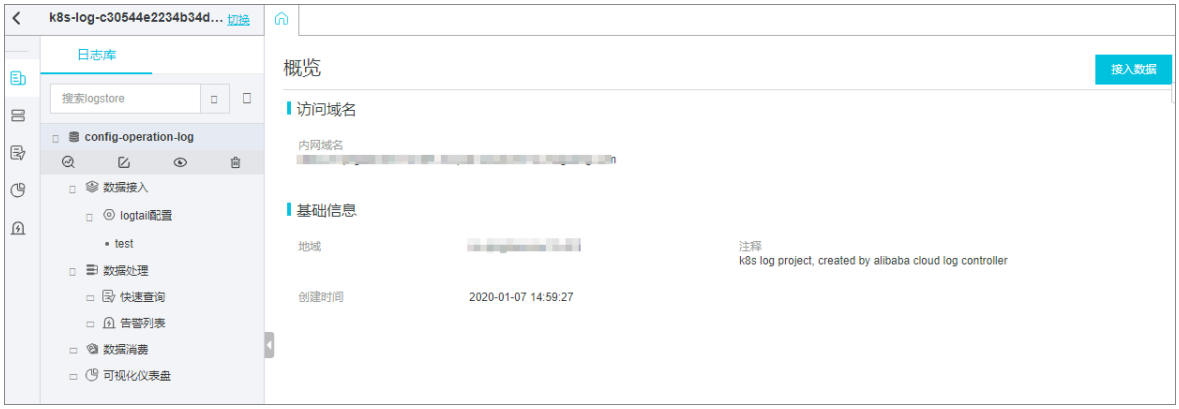
  关于Apache日志的采集，请参见Apache 日志。

采集步骤

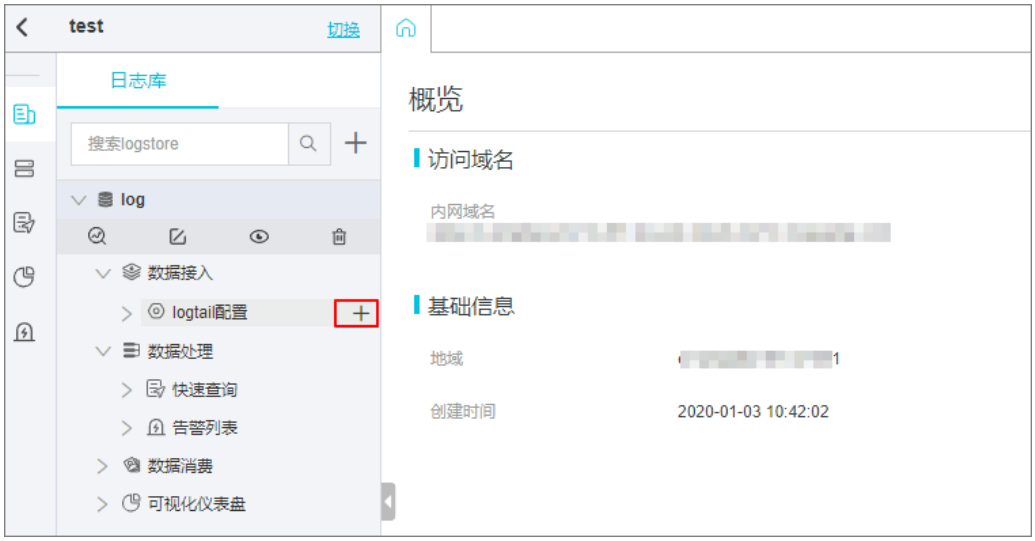
1. 登录日志服务控制台。
2. 选择数据源类型。
   您可以通过如下三种方式进入数据源界面。
   - 在日志服务首页的**接入数据**区域选择数据源类型。

   - 在**Project列表**中选择并单击Project名称，在Project概览页面单击**接入数据**进入数据源界面。

   - 选择一个日志库并展开其节点，单击**数据接入**后的加号进入数据源界面。

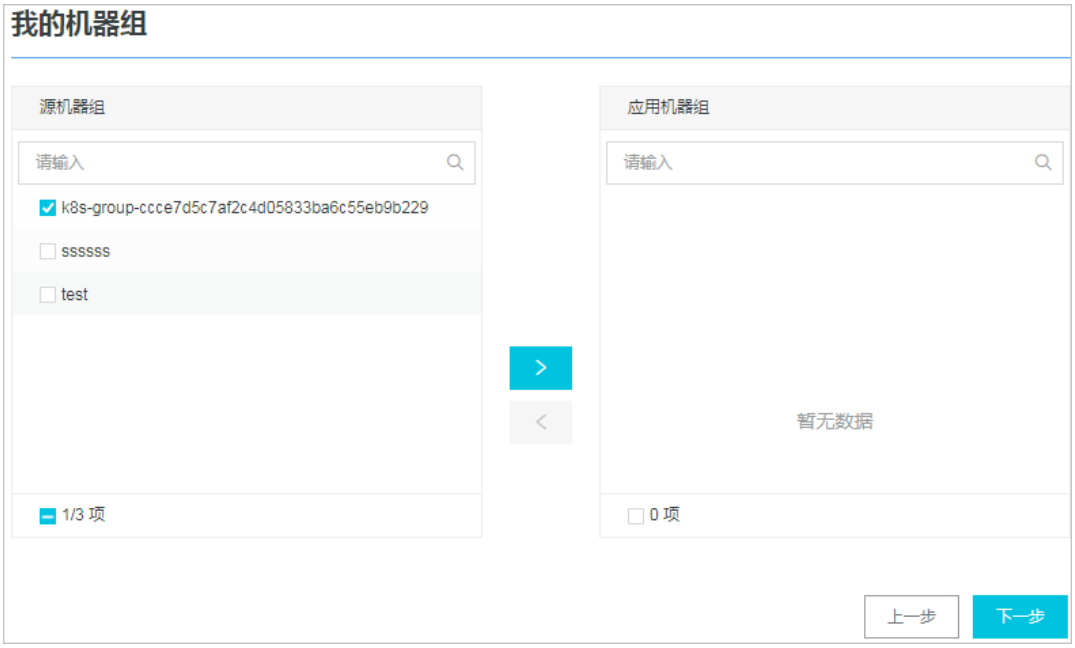
   请根据实际业务需求选择对应数据源类型。当前支持的文本数据源类型为：正则-文本日志、单行-文本日志、多行-文本日志、分隔符-文本日志、JSON-文本日志、Nginx-文本日志、IIS-文本日志、Apache-文本日志。

3. 选择日志空间，配置完成后，单击**下一步**。
   请选择Project和Logstore，您也可以直接单击**立即创建**，重新创建Project和Logstore。

   如果您是通过日志库下的**数据接入**后的加号进入采集配置流程，系统会直接跳过该步骤。
4. 创建机器组，单击**下一步**。
   在创建机器组之前，您需要首先确认已经安装了Logtail。

   请根据界面提示进行安装。或者请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）进行安装。

   安装完Logtail后单击**确认安装完毕**创建机器组，具体请参见简介。如果您之前已经创建好机器组，也可以直接单击**使用现有机器组**。
5. 配置机器组，配置完成后，单击**下一步**。
   选择一个机器组，将该机器组从**源机器组**移动到**应用机器组**。



6. 设置Logtail配置项。
   不同文本采集模式，Logtail配置项不同，请参见采集模式的配置项说明。
7. **可选：配置高级选项**，配置完成后，单击**下一步**。
   请根据您的需求选择高级配置。如没有特殊需求，建议保持默认配置。

配置项	详情
启用插件处理	开启后，支持通过插件实现对文本日志的多样处理。
上传原始日志	请选择是否需要上传原始日志。开启该功能后，原始日志内容会作为__raw__字段与解析过的日志一并上传。
Topic生成方式	◦ **空-不生成Topic**：默认选项，表示设置Topic为空字符串，在查询日志时不需要输入Topic即可查询。 ◦ **机器组Topic属性**：设置Topic生成方式为机器组Topic属性，可以用于明确区分不同前端服务器产生的日志数据。 ◦ **文件路径正则**：选择此项之后，您需要填写下方的**自定义正则**，用正则式从路径里提取一部分内容作为Topic。可以用于区分具体用户或实例产生的日志数据。
自定义正则	如您选择了**文件路径正则**方式生成Topic，需要在此处填写您的自定义正则式。
日志文件编码	◦ utf8：指定使用UTF-8编码。 ◦ gbk：指定使用GBK编码。
时区属性	设置采集日志时，日志时间的时区属性。 ◦ 机器时区：默认为机器所在时区。 ◦ 自定义时区：手动选择时区。

超时属性	如果一个日志文件在指定时间内没有任何更新，则认为该文件已超时。 ◦ 永不超时：指定持续监控所有日志文件，永不超时。 ◦ 30分钟超时：如日志文件在30分钟内没有更新，则认为已超时，并不再监控该文件。
过滤器配置	日志只有**完全符合**过滤器中的条件才会被收集。 例如： ◦ 过滤符合条件的数据：配置 `Key:level Regex:WARNING|ERROR` ，表示只收集level为WARNING或ERROR类型的日志。 ◦ 过滤不符合条件的数据： ▪ 配置 `Key:level Regex:^(?!.*(INFO|DEBUG)).*` ，表示不收集level为INFO或DEBUG类型的日志。 ▪ 配置 `Key:url Regex:.*^(?!.*(healthcheck)).*` ，表示不采集url中带有healthcheck的日志，例如key为url，value为 `/inner/healthcheck/jiankong.html` 的日志将不会被采集。

8. 配置索引，配置完成后，单击**下一步**。
   请根据业务需求设置索引。具体请参见开启并配置索引。

完成以上步骤日志服务就可以开始收集日志了。

3.1.4.2. 极简模式采集日志

日志服务支持通过控制台数据接入向导一站式配置极简模式采集日志与设置索引。

背景信息

极简模式可以分为两个模式：

- 单行模式

  单行模式表示一行内容为一条日志，即在日志文件中，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，可以通过修改采集配置进入完整模式逐项调整。有关如何修改Logtail配置请参见管理采集配置。

- 多行模式

  多行模式通过设置行首正则来划分一条日志。多行模式的采集设置和完整正则（关闭单行模式）类似，具体请参见正则模式采集日志，

采集步骤

1. 登录日志服务控制台。
2. 选择数据源类型。
   通过极简模式采集请选择**单行-文本文件**。
3. 选择目标Project和Logstore，配置完成后，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和Logstore。

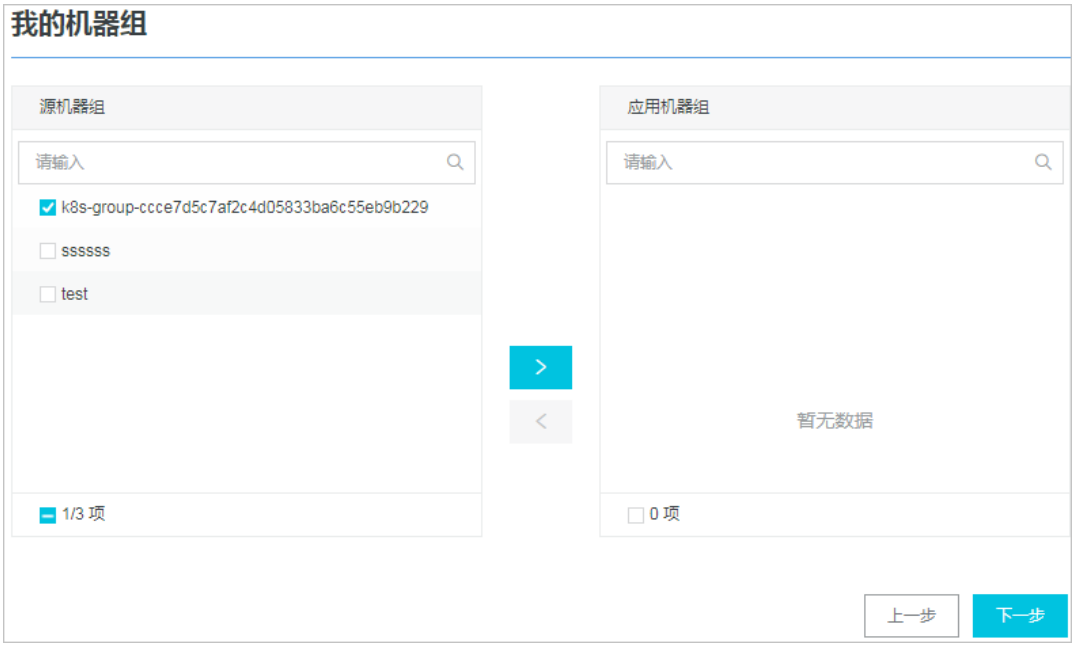
   如果您是通过日志库下的**数据接入**后的加号图标进入采集配置流程，系统会直接跳过该步骤。
4. 创建机器组，单击**下一步**。

   在创建机器组之前，您需要首先确认已经安装了Logtail。

   请根据界面提示进行安装。更多信息，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。

   安装完Logtail后，单击**确认安装完毕**创建机器组。如果您之前已经创建好机器组 ，也可以直接单击**使用现有机器组**。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **重要**
>
> 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为**FAIL**，您可单击**自动重试**。如果显示**FAIL**，请参见Logtail机器组无心跳处理。



6. Logtail配置。
   极简模式的Logtail配置项说明如下。

配置项	详情
配置名称	配置名称只能包含小写字母、数字、连字符（-）和下划线（_），且必须以小写字母和数字开头和结尾，长度为3~128字符。 说明 配置名称设置后不可修改。
日志路径	指定日志的目录和文件名。 ◦ 日志文件名支持完整文件名和通配符两种模式。 ◦ 日志文件查找模式为多层目录匹配，即指定文件夹下所有符合文件名模式的文件都会被监控到，包含所有层次的目录。 ▪ 例如 `/apsara/nuwa/ … /*.log` 表示 `/apsara/nuwa` 目录中（包含该目录的递归子目录）后缀名为 `.log` 的文件。 ▪ 例如 `/var/logs/app_* … /*.log*` 表示 `/var/logs` 目录下所有符合 `app_*` 模式的目录中（包含该目录的递归子目录）文件名包含 `.log` 的文件。 说明 ▪ 一个文件只能被一个配置收集。 ▪ 目录通配符只支持 `*` 和 `?` 两种。
是否为Docker文件	如果是Docker容器内部文件，可以直接配置内部路径与容器Tag，Logtail会自动监测容器创建和销毁，并根据Tag进行过滤采集指定容器的日志。
模式	前面步骤我们选择的数据源是**单行-文本文件**，所以这里默认是**极简模式**。
最大监控目录深度	指定从日志源采集日志时，监控目录的最大深度，即最多监控几层日志。最大目录监控深度范围0-1000，0代表只监控本层目录。

7. **可选：配置高级选项**，配置完成后，单击**下一步**。
   请根据您的需求选择高级配置。如没有特殊需求，建议保持默认配置。

配置项	详情

启用插件处理	开启后，支持通过插件实现对文本日志的多样处理。 **说明** 打开**启用插件处理**开关后，上传原始日志、时区属性、丢弃解析失败日志、过滤器配置、接受部分字段（分隔符模式）等功能不可用。
上传原始日志	打开**上传原始日志**开关后，原始日志将作为**__raw__**字段的值与解析过的日志一起上传到日志服务。
Topic生成方式	设置Topic生成方式。 ◦ **空-不生成Topic**：默认选项，表示设置Topic为空字符串，在查询日志时不需要输入Topic即可查询。 ◦ **机器组Topic属性**：设置为机器组Topic属性，用于明确区分不同服务器产生的日志数据。 ◦ **文件路径正则**：设置为文件路径正则，则需要设置**自定义正则**，用正则表达式从路径里提取一部分内容作为Topic。用于区分不同用户或实例产生的日志数据。
自定义正则	如您选择了**文件路径正则**方式生成Topic，需要在此处填写您的自定义正则式。
日志文件编码	设置日志文件编码格式，支持如下： ◦ utf8：指定使用UTF-8编码。 ◦ gbk：指定使用GBK编码。
时区属性	设置采集日志时，日志时间的时区属性。 ◦ 机器时区：默认为机器所在时区。 ◦ 自定义时区：手动选择时区。
超时属性	如果一个日志文件在指定时间内没有任何更新，则认为该文件已超时。 ◦ 永不超时：指定持续监控所有日志文件，永不超时。 ◦ 30分钟超时：如日志文件在30分钟内没有更新，则认为已超时，并不再监控该文件。 选择**30分钟超时**时，还需设置**最大超时目录深度**，范围为1~3。
过滤器配置	日志只有**完全符合**过滤器中的条件才会被收集。 例如： ◦ 满足条件即采集，例如设置**Key**为**level**，**Regex**为**WARNING|ERROR**，表示只采集level为WARNING或ERROR类型的日志。 ◦ 过滤不符合条件的日志。 ▪ 设置**Key**为**level**，**Regex**为**^(?!.*(INFO|DEBUG)).***，表示不采集level为INFO或DEBUG类型的日志。 ▪ 设置**Key**为**url**，**Regex**为**.*^(?!.*(healthcheck)).***，表示不采集URL中带有healthcheck的日志。例如Key为url，Value为/inner/healthcheck/jiankong.html的日志将不会被采集。

8. **查询和分析配置，包括配置索引。配置完成后，单击下一步。**

   日志服务默认开启全文索引。您也可以根据采集到的日志，手动或者自动设置字段索引。更多信息，请参见开启并配置索引。

   说明
   - 如果您要查询分析日志，那么全文索引和字段索引属性必须至少启用一种。同时启用时，以字段索引为准。
   - 索引类型为long、double时，大小写敏感和分词符属性无效。

配置完成后您就可以开始通过极简模式采集日志。

3.1.4.3. 正则模式采集日志

日志服务支持通过控制台数据接入向导一站式配置正则模式采集日志与设置索引。

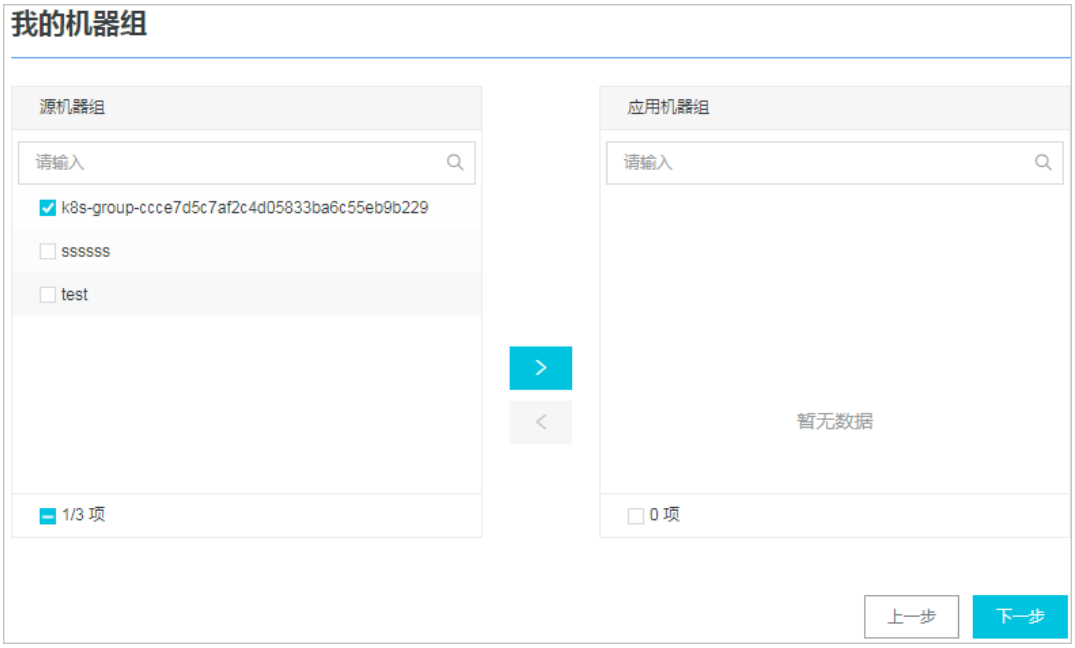
背景信息

如果需要对内容做更多个性化的字段提取设置（例如跨行日志，提取字段等），选择完整正则模式即可进行个性化定制。日志服务在数据采集向导中提供了基于日志样例生成正则表达式的功能，但鉴于日志样例的多样性，该表达式需要多次手动调试才能完全符合日志样例。如何调试正则表达式，请参见如何调试正则表达式。

采集步骤

1. 登录日志服务控制台。
2. 选择数据源类型。
   通过完整正则模式采集请选择**正则-文本文件**。
3. 选择目标Project和Logstore，配置完成后，单击**下一步**。
   您也可以单击**立即创建**，重新创建Project和Logstore。
   如果您是通过日志库下的**数据接入**后的加号图标进入采集配置流程，系统会直接跳过该步骤。
4. 创建机器组，单击**下一步**。
   在创建机器组之前，您需要首先确认已经安装了Logtail。
   请根据界面提示进行安装。更多信息，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。
   安装完Logtail后，单击**确认安装完毕**创建机器组。如果您之前已经创建好机器组，也可以直接单击**使用现有机器组**。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

   > **重要**
   > 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为**FAIL**，您可单击**自动重试**。如果显示**FAIL**，请参见Logtail机器组无心跳处理。



6. Logtail配置。
   正则模式的Logtail配置项说明如下。

配置项	详情
配置名称	配置名称只能包含小写字母、数字、连字符（-）和下划线（_），且必须以小写字母和数字开头和结尾，长度为3~128字符。 说明 配置名称设置后不可修改。

日志路径	指定日志的目录和文件名。 ◦ 日志文件名支持完整文件名和通配符两种模式。 ◦ 日志文件查找模式为多层目录匹配，即指定文件夹下所有符合文件名模式的文件都会被监控到，包含所有层次的目录。 ▪ 例如 `/apsara/nuwa/ … /*.log` 表示 `/apsara/nuwa` 目录中（包含该目录的递归子目录）后缀名为 `.log` 的文件。 ▪ 例如 `/var/logs/app_* … /*.log*` 表示 `/var/logs` 目录下所有符合 `app_*` 模式的目录中（包含该目录的递归子目录）文件名包含 `.log` 的文件。 **说明** ▪ 一个文件只能被一个配置收集。 ▪ 目录通配符只支持 `*` 和 `?` 两种。
是否为Docker文件	如果是Docker容器内部文件，可以直接配置内部路径与容器Tag，Logtail会自动监测容器创建和销毁，并根据Tag进行过滤采集指定容器的日志。
模式	之前步骤我们选择的数据源是**正则-文本文件**，所以这里默认是**完整正则模式**，您也可以手动修改。
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（例如Java 程序日志），需要关闭**单行模式**，然后设置 **行首正则表达式**。
日志样例	请务必使用实际场景的日志，方便日志服务控制台自动提取其中的正则匹配模式。
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。
提取字段	如果需要对日志内容中的字段单独分析处理，可以使用**提取字段**功能将指定字段变成 Key-Value对后发送到服务端，所以需要您指定解析一条日志内容的方式，即正则表达式。
正则表达式	当打开提取字段开关时，需要设置。 ◦ 自动生成正则表达式 填写完日志样例后，通过“划选”的方式操作日志样例，选中需要提取的字段，日志服务控制台会自动生成正则表达式。 ◦ 手动输入正则表达式 您也可以手动直接输入正则表达式。单击**手动输入正则表达式** 切换到手动输入模式。手动输入完成后，单击右侧的**验证** 即会验证您输入的正则表达式是否可以解析、提取日志样例。如何调试正则表达式，请参见如何调试正则表达式。
日志抽取内容	当打开提取字段开关时，需要设置。 无论使用自动生成还是手动输入方式，产生日志解析正则表达式后，您都需要给每个提取字段命名，设定对应字段的Key。
使用系统时间	当打开提取字段开关时，需要设置。 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`。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见配置时间格式。
丢弃解析失败日志	请选择解析失败的日志是否上传到日志服务。 ◦ 开启后，解析失败的日志不会上传到日志服务。 ◦ 关闭后，日志解析失败时上传原始日志。
最大监控目录深度	指定从日志源采集日志时，监控目录的最大深度，即最多监控几层日志。最大目录监控深度范围0-1000，0代表只监控本层目录。

7. **可选：配置高级选项，配置完成后，单击下一步。**

   请根据您的需求选择高级配置。如没有特殊需求，建议保持默认配置。

配置项	详情
启用插件处理	开启后，支持通过插件实现对文本日志的多样处理。 **说明** 打开**启用插件处理**开关后，上传原始日志、时区属性、丢弃解析失败日志、过滤器配置、接受部分字段（分隔符模式）等功能不可用。
上传原始日志	打开**上传原始日志**开关后，原始日志将作为**__raw__**字段的值与解析过的日志一起上传到日志服务。
Topic生成方式	设置Topic生成方式。 ◦ **空-不生成Topic**：默认选项，表示设置Topic为空字符串，在查询日志时不需要输入Topic即可查询。 ◦ **机器组Topic属性**：设置为机器组Topic属性，用于明确区分不同服务器产生的日志数据。 ◦ **文件路径正则**：设置为文件路径正则，则需要设置**自定义正则**，用正则表达式从路径里提取一部分内容作为Topic。用于区分不同用户或实例产生的日志数据。
自定义正则	如您选择了**文件路径正则**方式生成Topic，需要在此处填写您的自定义正则式。
日志文件编码	设置日志文件编码格式，支持如下： ◦ utf8：指定使用UTF-8编码。 ◦ gbk：指定使用GBK编码。
时区属性	设置采集日志时，日志时间的时区属性。 ◦ 机器时区：默认为机器所在时区。 ◦ 自定义时区：手动选择时区。
超时属性	如果一个日志文件在指定时间内没有任何更新，则认为该文件已超时。 ◦ 永不超时：指定持续监控所有日志文件，永不超时。 ◦ 30分钟超时：如日志文件在30分钟内没有更新，则认为已超时，并不再监控该文件。 选择**30分钟超时**时，还需设置**最大超时目录深度**，范围为1~3。
过滤器配置	日志只有**完全符合**过滤器中的条件才会被收集。 例如： ◦ 满足条件即采集，例如设置**Key**为**level**，**Regex**为**WARNING|ERROR**，表示只采集level为WARNING或ERROR类型的日志。 ◦ 过滤不符合条件的日志。 ▪ 设置**Key**为**level**，**Regex**为**^(?!.*(INFO|DEBUG)).***，表示不采集level为INFO或DEBUG类型的日志。 ▪ 设置**Key**为**url**，**Regex**为**.*^(?!.*(healthcheck)).***，表示不采集URL中带有healthcheck的日志。例如Key为url，Value为/inner/healthcheck/jiankong.html的日志将不会被采集。

8. **查询和分析配置，包括配置索引。配置完成后，单击下一步。**

   日志服务默认开启全文索引。您也可以根据采集到的日志，手动或者自动设置字段索引。更多信息，请参见开启并配置索引**。**

   说明
   - 如果您要查询分析日志，那么全文索引和字段索引属性必须至少启用一种。同时启用时，以字段索引为准。
   - 索引类型为long、double时，大小写敏感和分词符属性无效。

配置完成后您就可以开始通过正则模式采集日志。

3.1.4.4. 分隔符日志

日志服务支持通过控制台数据接入向导一站式配置分隔符模式采集日志与设置索引。

背景信息

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

日志介绍

常见的分隔符日志有：CSV、TSV 等。

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

单字符模式

单字符模式中，您需要指定分隔符，也可以同时指定引用符。

- **分隔符**：日志字段通过单字符的分隔符进行切分，例如制表符（ `\t` ）、竖线（ `|` ）、空格、逗号（ `,` ）、分号（ `;` ）和不可见字符等单字符。

  > **说明** 分隔符不支持设置为双引号（ `"` ）。

  如果双引号作为日志内容，而不是引用符（Quote）出现在日志中，则需要进行转义，日志中应处理为 `""` 。日志服务解析字段时会自动还原，即将 `""` 还原为 `"` 。双引号（ `"` ）可以作为Quote，在字段的边界单次出现；也可以作为字段内容成对出现（即转义为 `""` ），其它情况不符合分隔符日志的格式定义，请考虑极简模式、正则模式等其它方式进行字段解析。

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

- **引用符**：日志字段内容中包含分隔符时，为避免日志字段被误分割，需要指定引用符（Quote）对内部包含分隔符的日志字段进行包裹隔离。被引用符包裹的内容会被日志服务解析为一个完整字段，字段之间只能存在分隔符。

  引用符可以设置为制表符（ `\t` ）、竖线（ `|` ）、空格、逗号（ `,` ）、分号（ `;` ）和不可见字符等单字符。

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

> **说明** 日志服务支持将分隔符和引用符设置为不可见字符。不可见字符是ASCII码中编号为1~31及127的字符，指定分隔符和引用符为不可见字符时，您需要查找不可见字符在ASCII码中对应的十六进制数，输入的格式为 `0x不可见字符在ASCII码中对应的十六进制数` 。例如ASCII码中排行为1的不可见字符填写为 `0x01` 。

多字符模式

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

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

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

日志示例

- **单字符分隔符日志**

```
05/May/2016:13:30:28,10.10.*.*,"POST /PutData?
Category=YunOsAccountOpLog&AccessKeyId=****************&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20C
Topic=raw&Signature=****************************** HTTP/1.1",200,18204,aliyun-sdk-java
05/May/2016:13:31:23,10.10.*.*,"POST /PutData?
Category=YunOsAccountOpLog&AccessKeyId=****************&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20C
Topic=raw&Signature=****************************** HTTP/1.1",401,23472,aliyun-sdk-java
```

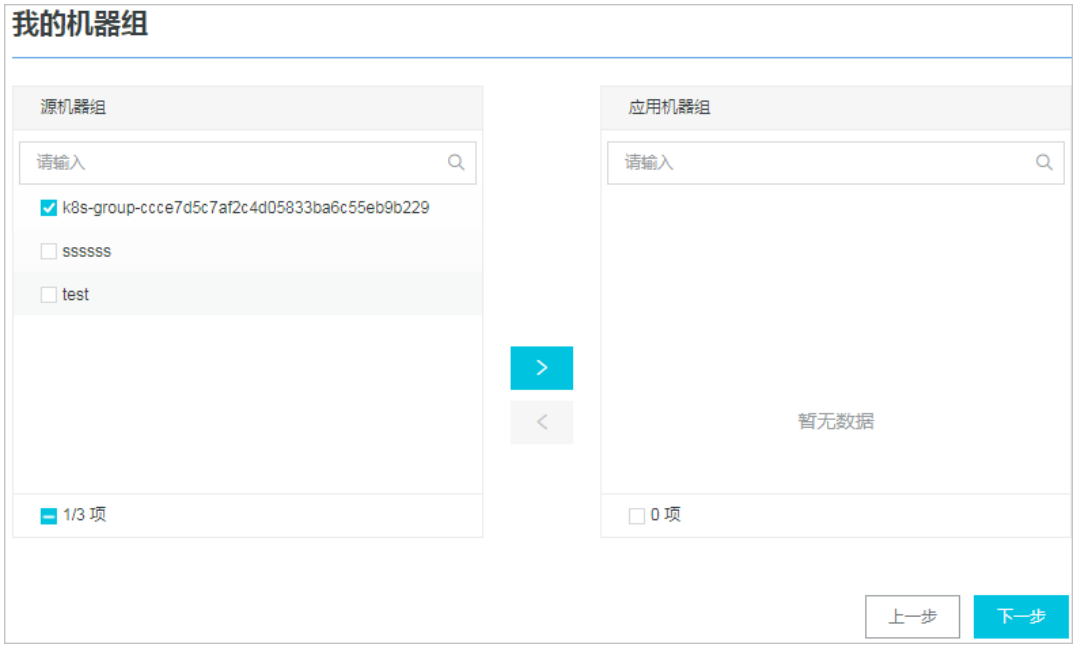
- **多字符分隔符日志**

```
05/May/2016:13:30:28&&10.200.**.**&&POST /PutData?
Category=YunOsAccountOpLog&AccessKeyId=****************&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20(
Topic=raw&Signature=pD12XYLmGxKQ%2Bmkd6x7hAgQ7b1c%3D HTTP/1.1&&200&&18204&&aliyun-sdk-java
05/May/2016:13:31:23&&10.200.**.**&&POST /PutData?
Category=YunOsAccountOpLog&AccessKeyId=****************&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20(
Topic=raw&Signature=****************************** HTTP/1.1&&401&&23472&&aliyun-sdk-java
```

采集步骤

1. 登录日志服务控制台。
2. 选择数据源类型。
   采集分隔符日志请选择**分隔符-文本日志**。
3. 选择目标Project和Logstore，配置完成后，单击**下一步**。
   您也可以单击**立即创建**，重新创建Project和Logstore。
   如果您是通过日志库下的**数据接入**后的加号图标进入采集配置流程，系统会直接跳过该步骤。
4. 创建机器组，单击**下一步**。
   在创建机器组之前，您需要首先确认已经安装了Logtail。
   请根据界面提示进行安装。更多信息，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。
   安装完Logtail后，单击**确认安装完毕**创建机器组。如果您之前已经创建好机器组，也可以直接单击**使用现有机器组**。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **重要**
> 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为**FAIL**，您可单击**自动重试**。如果显示**FAIL**，请参见Logtail机器组无心跳处理。



6. Logtail配置。
   分隔符日志采集配置的Logtail配置项说明如下。

配置项	详情
配置名称	配置名称只能包含小写字母、数字、连字符（-）和下划线（_），且必须以小写字母和数字开头和结尾，长度为3~128字符。 **说明** 配置名称设置后不可修改。

日志路径	指定日志的目录和文件名。 ◦ 日志文件名支持完整文件名和通配符两种模式。 ◦ 日志文件查找模式为多层目录匹配，即指定文件夹下所有符合文件名模式的文件都会被监控到，包含所有层次的目录。 ▪ 例如 `/apsara/nuwa/ … /*.log` 表示 `/apsara/nuwa` 目录中（包含该目录的递归子目录）后缀名为 `.log` 的文件。 ▪ 例如 `/var/logs/app_* … /*.log*` 表示 `/var/logs` 目录下所有符合 `app_*` 模式的目录中（包含该目录的递归子目录）文件名包含 `.log` 的文件。 **说明** ▪ 一个文件只能被一个配置收集。 ▪ 目录通配符只支持 `*` 和 `?` 两种。
是否为Docker文件	如果是Docker容器内部文件，可以直接配置内部路径与容器Tag，Logtail会自动监测容器创建和销毁，并根据Tag进行过滤采集指定容器的日志。
模式	之前步骤我们选择的数据源是**分隔符-文本日志**，所以这里默认是**分隔符模式**，您也可以手动修改。
日志样例	请务必使用实际场景的日志，方便日志服务控制台自动提取其中的正则匹配模式。
分隔符	选择分隔符。 请根据您的日志格式选择正确的分隔符否则日志数据会解析失败。 **说明** 指定分隔符为不可见字符时，您需要查找不可见字符在ASCII码中对应的十六进制数，输入的格式为 `0x`不可见字符在ASCII码中对应的十六进制数 。例如ASCII码中排行为1的不可见字符填写为 `0x01` 。
引用符	选择引用符。 请根据您的日志格式选择正确的引用符，否则日志数据会解析失败。 **说明** 指定引用符为不可见字符时，您需要查找不可见字符在ASCII码中对应的十六进制数，输入的格式为 `0x`不可见字符在ASCII码中对应的十六进制数 。例如ASCII码中排行为1的不可见字符填写为 `0x01` 。
日志抽取内容	填写日志样例并选择分隔符后，日志服务会按照您选择的分隔符提取日志字段，并将其定义为Value，您需要分别为Value指定对应的Key。
是否接受部分字段	配置采集后，若日志中分割出的字段数少于配置的Key数量，是否上传已解析的字段。开启表示上传，关闭表示丢弃本条日志。 例如，在配置采集分隔符日志时，示例日志为 `11|22|33|44|55` ，设置分隔符为 `|` ，日志字段会被解析为 `11` 、 `22` 、 `33` 、 `44` 和 `55` ，为其分别设置Key为 `A` 、 `B` 、 `C` 、 `D` 和 `E` 。如果配置采集时开启了**接受部分字段**，采集日志 `11|22|33|55` 时， `55` 字段会作为Key `D` 的Value被上传到日志服务。如果配置采集时关闭了**接受部分字段**，该条日志会因字段与Key不匹配而被丢弃。
使用系统时间	当打开提取字段开关时，需要设置。 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` 。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见配置时间格式。
丢弃解析失败日志	请选择解析失败的日志是否上传到日志服务。 ◦ 开启后，解析失败的日志不会上传到日志服务。 ◦ 关闭后，日志解析失败时上传原始日志。
最大监控目录深度	指定从日志源采集日志时，监控目录的最大深度，即最多监控几层日志。最大目录监控深度范围0-1000，0代表只监控本层目录。

7. **可选：配置高级选项**，配置完成后，单击**下一步**。

   请根据您的需求选择高级配置。如没有特殊需求，建议保持默认配置。

配置项	详情
启用插件处理	开启后，支持通过插件实现对文本日志的多样处理。 **说明** 打开**启用插件处理**开关后，上传原始日志、时区属性、丢弃解析失败日志、过滤器配置、接受部分字段（分隔符模式）等功能不可用。
上传原始日志	打开**上传原始日志**开关后，原始日志将作为**__raw__**字段的值与解析过的日志一起上传到日志服务。
Topic生成方式	设置Topic生成方式。 ◦ **空-不生成Topic**：默认选项，表示设置Topic为空字符串，在查询日志时不需要输入Topic即可查询。 ◦ **机器组Topic属性**：设置为机器组Topic属性，用于明确区分不同服务器产生的日志数据。 ◦ **文件路径正则**：设置为文件路径正则，则需要设置**自定义正则**，用正则表达式从路径里提取一部分内容作为Topic。用于区分不同用户或实例产生的日志数据。
自定义正则	如您选择了**文件路径正则**方式生成Topic，需要在此处填写您的自定义正则式。
日志文件编码	设置日志文件编码格式，支持如下： ◦ utf8：指定使用UTF-8编码。 ◦ gbk：指定使用GBK编码。
时区属性	设置采集日志时，日志时间的时区属性。 ◦ 机器时区：默认为机器所在时区。 ◦ 自定义时区：手动选择时区。
超时属性	如果一个日志文件在指定时间内没有任何更新，则认为该文件已超时。 ◦ 永不超时：指定持续监控所有日志文件，永不超时。 ◦ 30分钟超时：如日志文件在30分钟内没有更新，则认为已超时，并不再监控该文件。 选择**30分钟超时**时，还需设置**最大超时目录深度**，范围为1~3。
过滤器配置	日志只有**完全符合**过滤器中的条件才会被收集。 例如： ◦ 满足条件即采集，例如设置**Key**为**level**，**Regex**为**WARNING|ERROR**，表示只采集level为WARNING或ERROR类型的日志。 ◦ 过滤不符合条件的日志。 ▪ 设置**Key**为**level**，**Regex**为**^(?!.*(INFO|DEBUG)).***，表示不采集level为INFO或DEBUG类型的日志。 ▪ 设置**Key**为**url**，**Regex**为**.*^(?!.*(healthcheck)).***，表示不采集URL中带有healthcheck的日志。例如Key为url，Value为/inner/healthcheck/jiankong.html的日志将不会被采集。

8. **查询和分析配置**，包括配置索引。配置完成后，单击**下一步**。

   日志服务默认开启全文索引。您也可以根据采集到的日志，手动或者自动设置字段索引。更多信息，请参见开启并配置索引。

> **说明**
> - 如果您要查询分析日志，那么全文索引和字段索引属性必须至少启用一种。同时启用时，以字段索引为准。
> - 索引类型为long、double时，大小写敏感和分词符属性无效。

配置完成后您就可以开始采集分隔符日志。

3.1.4.5. JSON 日志

日志服务支持通过控制台数据接入向导一站式配置JSON模式采集日志与设置索引。

背景信息

Logtail支持的JSON日志是Object类型，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日志建构于两种结构：

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

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

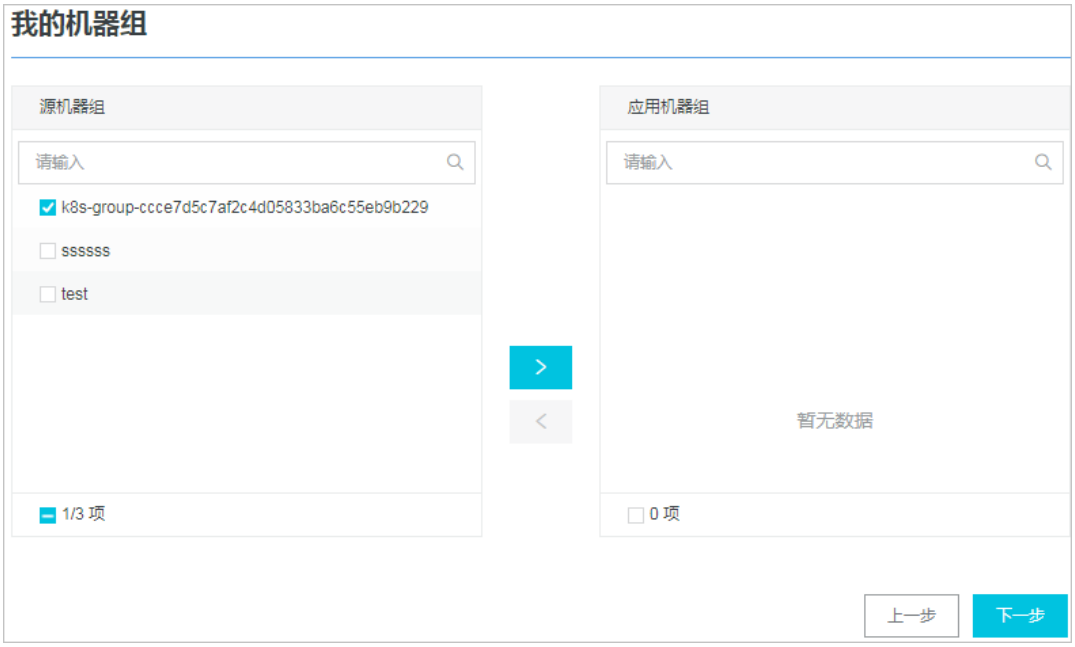
JSON格式日志样例如下：

```
{"url": "POST /PutData?
Category=YunOsAccountOpLog&AccessKeyId=U0Ujpek********&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20GMT&
ic=raw&Signature=pD12XYLmGxKQ%2Bmkd6x7hAgQ7b1c%3D HTTP/1.1", "ip": "10.200.98.220", "user-agent": "aliyu
n-sdk-java", "request": {"status": "200", "latency": "18204"}, "time": "05/May/2016:13:30:28"}
{"url": "POST /PutData?
Category=YunOsAccountOpLog&AccessKeyId=U0Ujpek********&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20GMT&
ic=raw&Signature=pD12XYLmGxKQ%2Bmkd6x7hAgQ7b1c%3D HTTP/1.1", "ip": "10.200.98.210", "user-agent": "aliyu
n-sdk-java", "request": {"status": "200", "latency": "10204"}, "time": "05/May/2016:13:30:29"}
```

采集步骤

1. 登录日志服务控制台。
2. 选择数据源类型。
   通过JSON模式采集请选择**JSON-文本日志**。
3. 选择目标Project和Logstore，配置完成后，单击**下一步**。
   您也可以单击**立即创建**，重新创建Project和Logstore。
   如果您是通过日志库下的**数据接入**后的加号图标进入采集配置流程，系统会直接跳过该步骤。
4. 创建机器组，单击**下一步**。
   在创建机器组之前，您需要首先确认已经安装了Logtail。
   请根据界面提示进行安装。更多信息，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。
   安装完Logtail后，单击**确认安装完毕**创建机器组。如果您之前已经创建好机器组，也可以直接单击**使用现有机器组**。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **重要**
> 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为**FAIL**，您可单击**自动重试**。如果显示**FAIL**，请参见Logtail机器组无心跳处理。



6. Logtail配置。
   JSON模式的Logtail配置项说明如下。

配置项	详情
配置名称	配置名称只能包含小写字母、数字、连字符（-）和下划线（_），且必须以小写字母和数字开头和结尾，长度为3~128字符。 说明 配置名称设置后不可修改。
日志路径	指定日志的目录和文件名。 ◦ 日志文件名支持完整文件名和通配符两种模式。 ◦ 日志文件查找模式为多层目录匹配，即指定文件夹下所有符合文件名模式的文件都会被监控到，包含所有层次的目录。 ▪ 例如 `/apsara/nuwa/ … /*.log` 表示 `/apsara/nuwa` 目录中（包含该目录的递归子目录）后缀名为 `.log` 的文件。 ▪ 例如 `/var/logs/app_* … /*.log*` 表示 `/var/logs` 目录下所有符合 `app_*` 模式的目录中（包含该目录的递归子目录）文件名包含 `.log` 的文件。 说明 ▪ 一个文件只能被一个配置收集。 ▪ 目录通配符只支持 `*` 和 `?` 两种。
是否为Docker文件	如果是Docker容器内部文件，可以直接配置内部路径与容器Tag，Logtail会自动监测容器创建和销毁，并根据Tag进行过滤采集指定容器的日志。
模式	之前步骤我们选择的数据源是**JSON-文本日志**，所以这里默认是**JSON模式**，您也可以手动修改。
使用系统时间	当打开提取字段开关时，需要设置。 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`。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见配置时间格式。

丢弃解析失败日志	请选择解析失败的日志是否上传到日志服务。 ◦ 开启后，解析失败的日志不会上传到日志服务。 ◦ 关闭后，日志解析失败时上传原始日志。
最大监控目录深度	指定从日志源采集日志时，监控目录的最大深度，即最多监控几层日志。最大目录监控深度范围0-1000，0代表只监控本层目录。

7. **可选：配置高级选项，配置完成后，单击下一步。**

   请根据您的需求选择高级配置。如没有特殊需求，建议保持默认配置。

配置项	详情
启用插件处理	开启后，支持通过插件实现对文本日志的多样处理。 **说明** 打开**启用插件处理**开关后，上传原始日志、时区属性、丢弃解析失败日志、过滤器配置、接受部分字段（分隔符模式）等功能不可用。
上传原始日志	打开**上传原始日志**开关后，原始日志将作为**__raw__**字段的值与解析过的日志一起上传到日志服务。
Topic生成方式	设置Topic生成方式。 ◦ **空-不生成Topic**：默认选项，表示设置Topic为空字符串，在查询日志时不需要输入Topic即可查询。 ◦ **机器组Topic属性**：设置为机器组Topic属性，用于明确区分不同服务器产生的日志数据。 ◦ **文件路径正则**：设置为文件路径正则，则需要设置**自定义正则**，用正则表达式从路径里提取一部分内容作为Topic。用于区分不同用户或实例产生的日志数据。
自定义正则	如您选择了**文件路径正则**方式生成Topic，需要在此处填写您的自定义正则式。
日志文件编码	设置日志文件编码格式，支持如下： ◦ utf8：指定使用UTF-8编码。 ◦ gbk：指定使用GBK编码。
时区属性	设置采集日志时，日志时间的时区属性。 ◦ 机器时区：默认为机器所在时区。 ◦ 自定义时区：手动选择时区。
超时属性	如果一个日志文件在指定时间内没有任何更新，则认为该文件已超时。 ◦ 永不超时：指定持续监控所有日志文件，永不超时。 ◦ 30分钟超时：如日志文件在30分钟内没有更新，则认为已超时，并不再监控该文件。 选择**30分钟超时**时，还需设置**最大超时目录深度**，范围为1~3。

过滤器配置	日志只有**完全符合**过滤器中的条件才会被收集。 例如： ◦ 满足条件即采集，例如设置**Key**为**level**，**Regex**为**WARNING|ERROR**，表示只采集level为WARNING或ERROR类型的日志。 ◦ 过滤不符合条件的日志。 ▪ 设置**Key**为**level**，**Regex**为**^(?!.*(INFO|DEBUG)).***，表示不采集level为INFO或DEBUG类型的日志。 ▪ 设置**Key**为**url**，**Regex**为**.*^(?!.*(healthcheck)).***，表示不采集URL中带有healthcheck的日志。例如Key为url，Value为/inner/healthcheck/jiankong.html的日志将不会被采集。

8. **查询和分析配置，包括配置索引。配置完成后，单击下一步。**

   日志服务默认开启全文索引。您也可以根据采集到的日志，手动或者自动设置字段索引。更多信息，请参见开启并配置索引。

> **说明**
> ◦ 如果您要查询分析日志，那么全文索引和字段索引属性必须至少启用一种。同时启用时，以字段索引为准。
> ◦ 索引类型为long、double时，大小写敏感和分词符属性无效。

配置完成后您就可以开始通过JSON模式采集日志。

3.1.4.6. Nginx 日志

日志服务支持通过控制台数据接入向导一站式配置Nginx配置模式采集日志与设置索引。

背景信息

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

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这种日志格式和写入的文件名。

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

日志样例

Nginx日志样例如下：

```
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"
```

字段说明

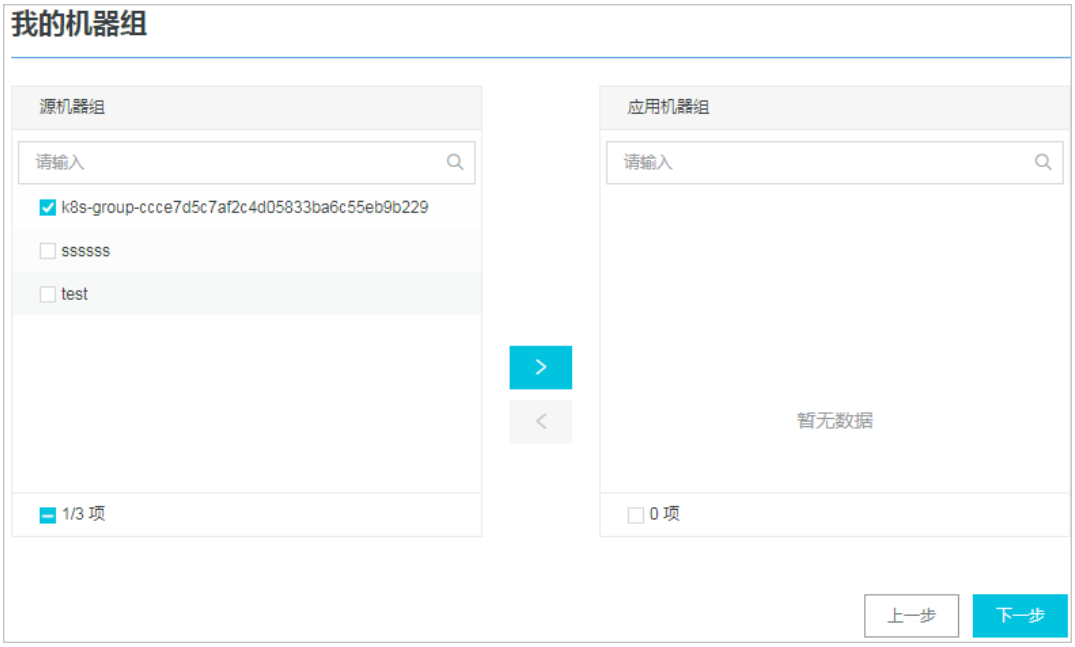
字段名称	含义
remote_addr	表示客户端IP地址。
remote_user	表示客户端用户名称。
request	表示请求的URL和HTTP协议。
status	表示请求状态。

body_bytes_sent	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 `.` 。
http_referer	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。
request_length	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。

采集步骤

1. 登录日志服务控制台。
2. 选择数据源类型。
   通过Nginx配置模式采集请选择**NGINX-文本日志**。
3. 选择目标Project和Logstore，配置完成后，单击**下一步**。
   您也可以单击**立即创建**，重新创建Project和Logstore。
   如果您是通过日志库下的**数据接入**后的加号图标进入采集配置流程，系统会直接跳过该步骤。
4. 创建机器组，单击**下一步**。
   在创建机器组之前，您需要首先确认已经安装了Logtail。
   请根据界面提示进行安装。更多信息，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。
   安装完Logtail后，单击**确认安装完毕**创建机器组。如果您之前已经创建好机器组，也可以直接单击**使用现有机器组**。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **重要**
> 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为**FAIL**，您可单击**自动重试**。如果显示**FAIL**，请参见Logtail机器组无心跳处理。



6. Logtail配置。
Nginx配置模式的Logtail配置项说明如下。

配置项	详情
配置名称	配置名称只能包含小写字母、数字、连字符（-）和下划线（_），且必须以小写字母和数字开头和结尾，长度为3~128字符。 说明 配置名称设置后不可修改。
日志路径	指定日志的目录和文件名。 ◦ 日志文件名支持完整文件名和通配符两种模式。 ◦ 日志文件查找模式为多层目录匹配，即指定文件夹下所有符合文件名模式的文件都会被监控到，包含所有层次的目录。 ▪ 例如 `/apsara/nuwa/ … /*.log` 表示 `/apsara/nuwa` 目录中（包含该目录的递归子目录）后缀名为 `.log` 的文件。 ▪ 例如 `/var/logs/app_* … /*.log*` 表示 `/var/logs` 目录下所有符合 `app_*` 模式的目录中（包含该目录的递归子目录）文件名包含 `.log` 的文件。 说明 ▪ 一个文件只能被一个配置收集。 ▪ 目录通配符只支持 `*` 和 `?` 两种。
是否为Docker文件	如果是Docker容器内部文件，可以直接配置内部路径与容器Tag，Logtail会自动监测容器创建和销毁，并根据Tag进行过滤采集指定容器的日志。
模式	默认是**NGINX配置模式**，您也可以手动修改。
NGINX日志配置	请填写标准Nginx配置文件日志配置部分，通常以 `log_format` 开头。
NGINX键名称	日志服务会自动读取您的Nginx键。
丢弃解析失败日志	请选择解析失败的日志是否上传到日志服务。 ◦ 开启后，解析失败的日志不会上传到日志服务。 ◦ 关闭后，日志解析失败时上传原始日志。

最大监控目录深度	指定从日志源采集日志时，监控目录的最大深度，即最多监控几层日志。最大目录监控深度范围0-1000，0代表只监控本层目录。

7. **可选：配置高级选项，配置完成后，单击下一步。**

   请根据您的需求选择高级配置。如没有特殊需求，建议保持默认配置。

配置项	详情
启用插件处理	开启后，支持通过插件实现对文本日志的多样处理。 说明 打开**启用插件处理**开关后，上传原始日志、时区属性、丢弃解析失败日志、过滤器配置、接受部分字段（分隔符模式）等功能不可用。
上传原始日志	打开**上传原始日志**开关后，原始日志将作为**__raw__**字段的值与解析过的日志一起上传到日志服务。
Topic生成方式	设置Topic生成方式。 ◦ **空-不生成Topic**：默认选项，表示设置Topic为空字符串，在查询日志时不需要输入Topic即可查询。 ◦ **机器组Topic属性**：设置为机器组Topic属性，用于明确区分不同服务器产生的日志数据。 ◦ **文件路径正则**：设置为文件路径正则，则需要设置**自定义正则**，用正则表达式从路径里提取一部分内容作为Topic。用于区分不同用户或实例产生的日志数据。
自定义正则	如您选择了**文件路径正则**方式生成Topic，需要在此处填写您的自定义正则式。
日志文件编码	设置日志文件编码格式，支持如下： ◦ utf8：指定使用UTF-8编码。 ◦ gbk：指定使用GBK编码。
时区属性	设置采集日志时，日志时间的时区属性。 ◦ 机器时区：默认为机器所在时区。 ◦ 自定义时区：手动选择时区。
超时属性	如果一个日志文件在指定时间内没有任何更新，则认为该文件已超时。 ◦ 永不超时：指定持续监控所有日志文件，永不超时。 ◦ 30分钟超时：如日志文件在30分钟内没有更新，则认为已超时，并不再监控该文件。 选择**30分钟超时**时，还需设置**最大超时目录深度**，范围为1~3。
过滤器配置	日志只有**完全符合**过滤器中的条件才会被收集。 例如： ◦ 满足条件即采集，例如设置**Key**为**level**，**Regex**为**WARNING|ERROR**，表示只采集level为WARNING或ERROR类型的日志。 ◦ 过滤不符合条件的日志。 ▪ 设置**Key**为**level**，**Regex**为**^(?!.*(INFO|DEBUG)).***，表示不采集level为INFO或DEBUG类型的日志。 ▪ 设置**Key**为**url**，**Regex**为**.*^(?!.*(healthcheck)).***，表示不采集URL中带有healthcheck的日志。例如Key为url，Value为/inner/healthcheck/jiankong.html的日志将不会被采集。

8. **查询和分析配置，包括配置索引。配置完成后，单击下一步。**

   日志服务默认开启全文索引。您也可以根据采集到的日志，手动或者自动设置字段索引。更多信息，请参见开启并配置索引。

> 说明
> - 如果您要查询分析日志，那么全文索引和字段索引属性必须至少启用一种。同时启用时，以字段索引为准。
> - 索引类型为long、double时，大小写敏感和分词符属性无效。

配置完成后您就可以开始通过Nginx配置模式采集日志。

3.1.4.7. IIS 日志

日志服务支持通过控制台数据接入向导一站式配置采集IIS日志与设置索引。

背景信息

为了更好满足分析场景，推荐选用W3C日志格式，在IIS管理器中单击选择字段按钮，勾选发送的字节数和和接收的字节数。

日志格式

W3C配置格式如下：

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

- 字段前缀说明

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

- 各个字段说明

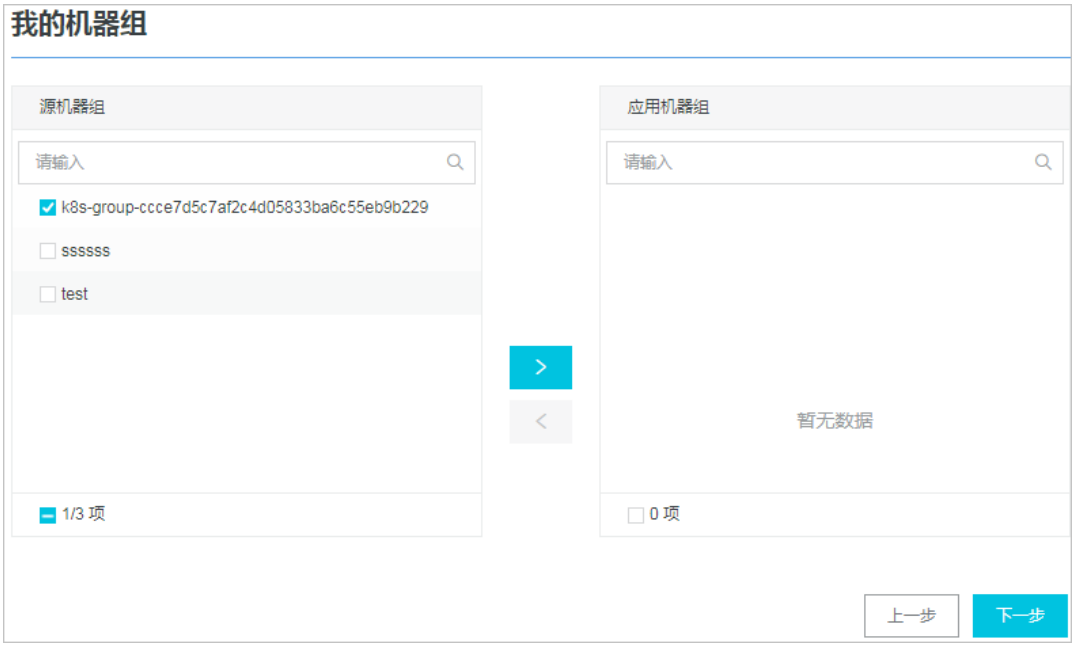
字段	说明
date	日期，表示活动发生的日期。
time	时间，表示活动发生的时间。
s-sitename	服务名，表示客户端所访问的该站点的 Internet 服务和实例的号码。
s-computername	服务器名，表示生成日志项的服务器名称。
s-ip	服务器IP，表示生成日志项的服务器的IP地址。
cs-method	方法，例如GET或POST。
cs-uri-stem	URI资源，表示请求访问的地址。
cs-uri-query	URI查询，表示查询HTTP请求中问号（?）后的信息。
s-port	服务器端口，表示客户端连接的服务器端口号。

cs-username	通过验证的域或用户名，对于通过身份验证的用户，格式是 `域\用户名` ；对于匿名用户，是一个连字符 (-)。
c-ip	客户端IP，表示访问服务器的客户端真实IP 地址。
cs-version	协议版本，例如 HTTP 1.0 或 HTTP 1.1。
user-agent	用户代理，表示在客户端使用的浏览器。
Cookie	Cookie，表示发送或接受的Cookie内容，如果没有Cookie，则显示连字符（-）。
referer	引用站点，表示用户访问的前一个站点。此站点提供到当前站点到链接。
cs-host	主机，表示主机头内容。
sc-status	协议返回状态，表示HTTP或FTP的操作状态。
sc-substatus	HTTP子协议的状态。
sc-win32-status	win32状态，即用 Windows使用的术语表示的操作的状态。
sc-bytes	服务器发送字节。
cs-bytes	服务器接收字节。
time-taken	所用时间，即操作所花时间长短，单位为毫秒。

采集步骤

1. 登录日志服务控制台。
2. 选择数据源类型。
   采集IIS文本日志请选择**IIS-文本日志**。
3. 选择目标Project和Logstore，配置完成后，单击**下一步**。
   您也可以单击**立即创建**，重新创建Project和Logstore。
   如果您是通过日志库下的**数据接入**后的加号图标进入采集配置流程，系统会直接跳过该步骤。
4. 创建机器组，单击**下一步**。
   在创建机器组之前，您需要首先确认已经安装了Logtail。
   请根据界面提示进行安装。更多信息，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。
   安装完Logtail后，单击**确认安装完毕**创建机器组。如果您之前已经创建好机器组，也可以直接单击**使用现有机器组**。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **重要**
> 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为**FAIL**，您可单击**自动重试**。如果显示**FAIL**，请参见Logtail机器组无心跳处理。



6. Logtail配置。
IIS日志采集配置的Logtail配置项说明如下。

配置项	详情
配置名称	配置名称只能包含小写字母、数字、连字符（-）和下划线（_），且必须以小写字母和数字开头和结尾，长度为3~128字符。 说明 配置名称设置后不可修改。
日志路径	指定日志的目录和文件名。 ◦ 日志文件名支持完整文件名和通配符两种模式。 ◦ 日志文件查找模式为多层目录匹配，即指定文件夹下所有符合文件名模式的文件都会被监控到，包含所有层次的目录。 ▪ 例如 `/apsara/nuwa/ … /*.log` 表示 `/apsara/nuwa` 目录中（包含该目录的递归子目录）后缀名为 `.log` 的文件。 ▪ 例如 `/var/logs/app_* … /*.log*` 表示 `/var/logs` 目录下所有符合 `app_*` 模式的目录中（包含该目录的递归子目录）文件名包含 `.log` 的文件。 说明 ▪ 一个文件只能被一个配置收集。 ▪ 目录通配符只支持 `*` 和 `?` 两种。
是否为Docker文件	如果是Docker容器内部文件，可以直接配置内部路径与容器Tag，Logtail会自动监测容器创建和销毁，并根据Tag进行过滤采集指定容器的日志。关于容器文本日志采集请参见容器文本日志。
模式	前面步骤我们选择的数据源是**IIS-文本日志**，所以这里默认是**IIS配置模式**。
日志格式	选择您的IIS服务器日志采用的日志格式。 ◦ IIS：Microsoft IIS日志文件格式。 ◦ NCSA：NCSA公用日志文件格式。 ◦ W3C：W3C扩展日志文件格式。

IIS配置字段	IIS配置字段。 ◦ IIS或NCSA格式：配置字段已预设。 ◦ W3C日志：请按照配置IIS配置字段配置IIS配置字段。
IIS键名称	IIS日志服务会自动提取出相应的键名称。
丢弃解析失败日志	请选择解析失败的日志是否上传到日志服务。 ◦ 开启后，解析失败的日志不会上传到日志服务。 ◦ 关闭后，日志解析失败时上传原始日志。
最大监控目录深度	指定从日志源采集日志时，监控目录的最大深度，即最多监控几层日志。最大目录监控深度范围0-1000，0代表只监控本层目录。

7. **配置IIS配置字段。**
   i. **打开IIS配置文件。**
      - IIS5配置文件默认路径：C:\WINNT\system32\inetsrv\MetaBase.bin
      - IIS6配置文件默认路径：C:\WINDOWS\system32\inetsrv\MetaBase.xml
      - IIS7配置文件默认路径：C:\Windows\System32\inetsrv\config\applicationHost.config
   ii. **找到** `logFile logExtFileFlags` **字段，并复制引号内的字段内容。**
   iii. **粘贴字段内容到IIS配置字段输入框中的引号内。**
8. **可选：配置高级选项，配置完成后，单击下一步。**
   请根据您的需求选择高级配置。如没有特殊需求，建议保持默认配置。

配置项	详情
启用插件处理	开启后，支持通过插件实现对文本日志的多样处理。 **说明** 打开**启用插件处理**开关后，上传原始日志、时区属性、丢弃解析失败日志、过滤器配置、接受部分字段（分隔符模式）等功能不可用。
上传原始日志	打开**上传原始日志**开关后，原始日志将作为**__raw__**字段的值与解析过的日志一起上传到日志服务。
Topic生成方式	设置Topic生成方式。 ◦ **空-不生成Topic**：默认选项，表示设置Topic为空字符串，在查询日志时不需要输入Topic即可查询。 ◦ **机器组Topic属性**：设置为机器组Topic属性，用于明确区分不同服务器产生的日志数据。 ◦ **文件路径正则**：设置为文件路径正则，则需要设置**自定义正则**，用正则表达式从路径里提取一部分内容作为Topic。用于区分不同用户或实例产生的日志数据。
自定义正则	如您选择了**文件路径正则**方式生成Topic，需要在此处填写您的自定义正则式。
日志文件编码	设置日志文件编码格式，支持如下： ◦ utf8：指定使用UTF-8编码。 ◦ gbk：指定使用GBK编码。
时区属性	设置采集日志时，日志时间的时区属性。 ◦ 机器时区：默认为机器所在时区。 ◦ 自定义时区：手动选择时区。

超时属性	如果一个日志文件在指定时间内没有任何更新，则认为该文件已超时。 ◦ 永不超时：指定持续监控所有日志文件，永不超时。 ◦ 30分钟超时：如日志文件在30分钟内没有更新，则认为已超时，并不再监控该文件。 选择**30分钟超时**时，还需设置**最大超时目录深度**，范围为1~3。
过滤器配置	日志只有**完全符合**过滤器中的条件才会被收集。 例如： ◦ 满足条件即采集，例如设置**Key**为**level**，**Regex**为**WARNING|ERROR**，表示只采集level为WARNING或ERROR类型的日志。 ◦ 过滤不符合条件的日志。 ▪ 设置**Key**为**level**，**Regex**为**^(?!.*(INFO|DEBUG)).***，表示不采集level为INFO或DEBUG类型的日志。 ▪ 设置**Key**为**url**，**Regex**为**.*^(?!.*(healthcheck)).***，表示不采集URL中带有healthcheck的日志。例如Key为url，Value为/inner/healthcheck/jiankong.html的日志将不会被采集。

9. **查询和分析配置，包括配置索引。配置完成后，单击下一步。**

   日志服务默认开启全文索引。您也可以根据采集到的日志，手动或者自动设置字段索引。更多信息，请参见开启并配置索引。

   > **说明**
   > ◦ 如果您要查询分析日志，那么全文索引和字段索引属性必须至少启用一种。同时启用时，以字段索引为准。
   > ◦ 索引类型为long、double时，大小写敏感和分词符属性无效。

配置完成后您就可以开始采集IIS日志。

3.1.4.8. Apache 日志

日志服务支持通过控制台数据接入向导一站式配置采集Apache日志与设置索引。

日志格式

Apache日志配置文件中默认定义了两种打印格式，分别为combined格式和common格式。您也可以添加自定义配置，按需求配置您的日志的打印格式。

- 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"
```

- 自定义格式

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %D %f %k %p %q %R %T %I %O" customized
```

Apache日志配置文件中需要同时指定当前日志的打印格式、日志文件路径及名称。例如以下声明表示日志打印时使用配置文件中定义的combined格式，且日志路径和名称为/var/log/apache2/access_log。

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

字段说明

字段格式	键名称	含义
%a	client_addr	请求报文中的客户端IP地址。
%A	local_addr	本地私有IP地址。
%b	response_size_bytes	响应字节大小，空值时可能为“-”。

%B	response_bytes	响应字节大小，空值时为0。
%D	request_time_msec	请求时间，单位为毫秒。
%h	remote_addr	远端主机名。
%H	request_protocol_supple	请求协议。
%l	remote_ident	客户端日志名称，来自identd。
%m	request_method_supple	请求方法。
%p	remote_port	服务器端口号。
%P	child_process	子进程ID。
%q	request_query	查询字符串，如果不存在则为空字符串。
“%r”	request	请求内容，包括方法名、地址和http协议。
%s	status	响应的http状态码。
%>s	status	响应的http状态码的最终结果。
%f	filename	请求的文件名。
%k	keep_alive	keep-alive请求数。
%R	response_handler	服务端的处理程序类型。
%t	time_local	服务器时间。
%T	request_time_sec	请求时间，单位为秒。
%u	remote_user	客户端用户名。
%U	request_uri_supple	请求的URI路径，不带query。
%v	server_name	服务器名称。
%V	server_name_canonical	服务器权威规范名称。
%I	bytes_received	服务器接收的字节数，需要启用mod_logio模块。
%O	bytes_sent	服务器发送的字节数，需要启用mod_logio模块。
“%{User-Agent}i”	http_user_agent	客户端信息。
“%{Rererer}i”	http_referer	来源页。

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, l
ike Gecko) Chrome/48.0.2564.97 Safari/537.36"
```

采集步骤

1. 登录日志服务控制台。
2. 选择数据源类型。
   通过APACHE配置模式采集请选择**Apache-文本日志**。
3. 选择目标Project和Logstore，配置完成后，单击**下一步**。
   您也可以单击**立即创建**，重新创建Project和Logstore。
   如果您是通过日志库下的**数据接入**后的加号图标进入采集配置流程，系统会直接跳过该步骤。
4. 创建机器组，单击**下一步**。

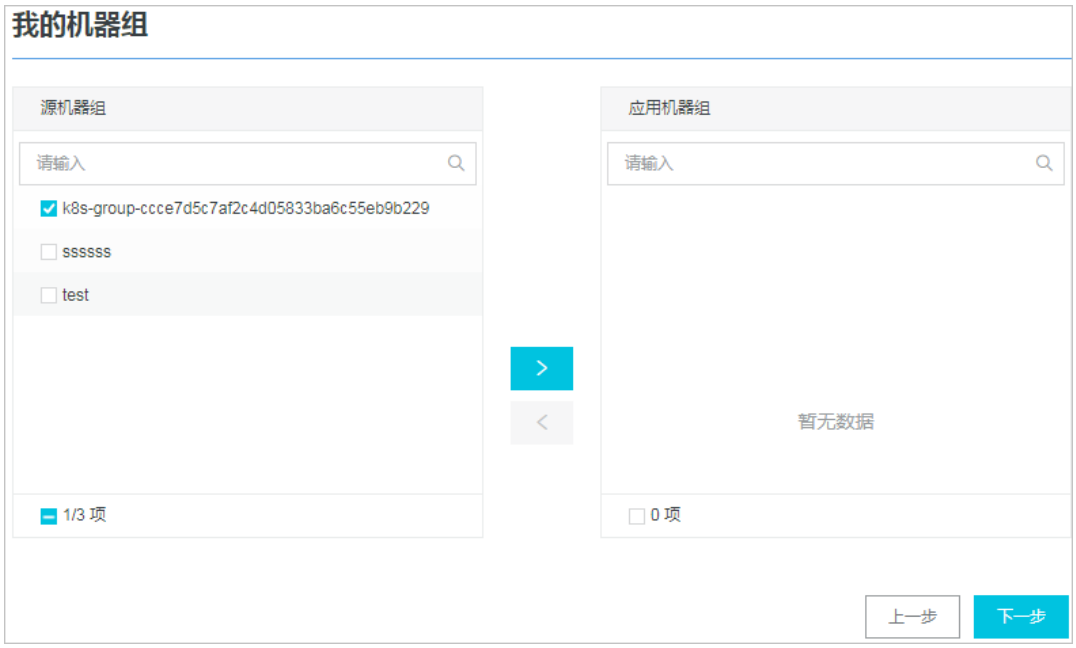
在创建机器组之前，您需要首先确认已经安装了Logtail。

请根据界面提示进行安装。更多信息，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。

安装完Logtail后，单击**确认安装完毕**创建机器组。如果您之前已经创建好机器组，也可以直接单击**使用现有机器组**。

5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **重要**
> 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为**FAIL**，您可单击**自动重试**。如果显示**FAIL**，请参见Logtail机器组无心跳处理。



6. Logtail配置。
Apache日志采集配置的Logtail配置项说明如下。

配置项	详情
配置名称	配置名称只能包含小写字母、数字、连字符（-）和下划线（_），且必须以小写字母和数字开头和结尾，长度为3~128字符。 **说明** 配置名称设置后不可修改。
日志路径	指定日志的目录和文件名。 ◦ 日志文件名支持完整文件名和通配符两种模式。 ◦ 日志文件查找模式为多层目录匹配，即指定文件夹下所有符合文件名模式的文件都会被监控到，包含所有层次的目录。 ▪ 例如 `/apsara/nuwa/ … /*.log` 表示 `/apsara/nuwa` 目录中（包含该目录的递归子目录）后缀名为 `.log` 的文件。 ▪ 例如 `/var/logs/app_* … /*.log*` 表示 `/var/logs` 目录下所有符合 `app_*` 模式的目录中（包含该目录的递归子目录）文件名包含 `.log` 的文件。 **说明** ▪ 一个文件只能被一个配置收集。 ▪ 目录通配符只支持 `*` 和 `?` 两种。
是否为Docker文件	如果是Docker容器内部文件，可以直接配置内部路径与容器Tag，Logtail会自动监测容器创建和销毁，并根据Tag进行过滤采集指定容器的日志。关于容器文本日志采集请参见容器文本日志。

模式	前面步骤我们选择的数据源是**Apache-文本日志**，所以这里默认是**APACHE配置模式**。
日志格式	请按照您的Apache日志配置文件中声明的格式选择日志格式。为了便于日志数据的查询分析，日志服务推荐您使用自定义的Apache日志格式。
APACHE配置字段	请填写标准Apache配置文件日志配置部分，通常以LogFormat开头。 **说明** 如您的**日志格式**为**common**或**combined**格式，此处会自动匹配对应格式的配置字段，请确认是否和本地Apache配置文件中定义的格式一致。
APACHE键名称	日志服务会自动读取您的Apache键。请在当前页面确认Apache键名称。
丢弃解析失败日志	请选择解析失败的日志是否上传到日志服务。 ◦ 开启后，解析失败的日志不会上传到日志服务。 ◦ 关闭后，日志解析失败时上传原始日志。
最大监控目录深度	指定从日志源采集日志时，监控目录的最大深度，即最多监控几层日志。最大目录监控深度范围0-1000，0代表只监控本层目录。

7. **可选：配置高级选项，配置完成后，单击下一步。**

   请根据您的需求选择高级配置。如没有特殊需求，建议保持默认配置。

配置项	详情
启用插件处理	开启后，支持通过插件实现对文本日志的多样处理。 **说明** 打开**启用插件处理**开关后，上传原始日志、时区属性、丢弃解析失败日志、过滤器配置、接受部分字段（分隔符模式）等功能不可用。
上传原始日志	打开**上传原始日志**开关后，原始日志将作为**__raw__**字段的值与解析过的日志一起上传到日志服务。
Topic生成方式	设置Topic生成方式。 ◦ **空-不生成Topic**：默认选项，表示设置Topic为空字符串，在查询日志时不需要输入Topic即可查询。 ◦ **机器组Topic属性**：设置为机器组Topic属性，用于明确区分不同服务器产生的日志数据。 ◦ **文件路径正则**：设置为文件路径正则，则需要设置**自定义正则**，用正则表达式从路径里提取一部分内容作为Topic。用于区分不同用户或实例产生的日志数据。
自定义正则	如您选择了**文件路径正则**方式生成Topic，需要在此处填写您的自定义正则式。
日志文件编码	设置日志文件编码格式，支持如下： ◦ utf8：指定使用UTF-8编码。 ◦ gbk：指定使用GBK编码。
时区属性	设置采集日志时，日志时间的时区属性。 ◦ 机器时区：默认为机器所在时区。 ◦ 自定义时区：手动选择时区。
超时属性	如果一个日志文件在指定时间内没有任何更新，则认为该文件已超时。 ◦ 永不超时：指定持续监控所有日志文件，永不超时。 ◦ 30分钟超时：如日志文件在30分钟内没有更新，则认为已超时，并不再监控该文件。 选择**30分钟超时**时，还需设置**最大超时目录深度**，范围为1~3。

过滤器配置	日志只有**完全符合**过滤器中的条件才会被收集。 例如： ◦ 满足条件即采集，例如设置**Key**为**level**，**Regex**为**WARNING|ERROR**，表示只采集level为WARNING或ERROR类型的日志。 ◦ 过滤不符合条件的日志。 ▪ 设置**Key**为**level**，**Regex**为**^(?!.*(INFO|DEBUG)).***，表示不采集level为INFO或DEBUG类型的日志。 ▪ 设置**Key**为**url**，**Regex**为**.*^(?!.*(healthcheck)).***，表示不采集URL中带有healthcheck的日志。例如Key为url，Value为/inner/healthcheck/jiankong.html的日志将不会被采集。

8. **查询和分析配置，包括配置索引。配置完成后，单击下一步。**

   日志服务默认开启全文索引。您也可以根据采集到的日志，手动或者自动设置字段索引。更多信息，请参见开启并配置索引。

   > **说明**
   > ◦ 如果您要查询分析日志，那么全文索引和字段索引属性必须至少启用一种。同时启用时，以字段索引为准。
   > ◦ 索引类型为long、double时，大小写敏感和分词符属性无效。

配置完成后您就可以开始采集Apache日志。

3.1.4.9. 配置解析

在采集接入过程中，需要对日志内容适当进行一些配置。

指定日志行分割方式

一条完整的访问日志一般为一行一条，例如Nginx的访问日志，每条日志以换行符分割。例如以下两条访问日志：

```
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，即行首正则表达式为 `[\d+-\d+-\w+:\d+:\d+,\d+]\s.*` **。在控制台可按照如下格式填写：**

图 1. 完整模式解析正则表达式

提取日志字段内容

根据日志服务数据模型，一条日志的内容包含一个或者多个Key-Value对，如果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。

对于上例中的访问日志，您可以选择提取字段或不提取字段。

- 提取字段

  正则表达式为 `(\S+)\s-\s-\s\[(\S+)\s[^]]+]\s"(\w+).*` ，提取内容为： `10.1.1.1` 、 `13/Mar/2016:10:00` 和 `GET` 。

- 不提取字段

  正则表达式为 `(.*)` ，提取内容为： `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做为日志的时间。

对于上例中的访问日志：

- 如果提取日志内容中的时间字段，时间为 `13/Mar/2016:10:00:10` ，时间表达式为 `%d/%b/%Y:%H:%M:%S` 。
- 抓取日志时的系统时间，时间为抓取日志时的时间戳。

3.1.4.10. 配置时间格式

每条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时，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。

> **说明**
> - 日志的时间戳精确到秒，所以时间格式只需配置到秒，无需配置毫秒、微秒等信息。
> - 只需配置time字段中的时间部分即可，其他内容无需配置。

Logtail支持的常见日志时间格式

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	说明	示例
%a	星期的缩写。	Fri

%A	星期的全称。	Friday
%b	月份的缩写。	Jan
%B	月份的全称。	January
%d	每月第几天，十进制格式，范围为01~31。	07, 31
%h	月份的缩写，与 `%b` 相同。	Jan
%H	小时，24小时制。	22
%I	小时，12小时制。	11
%m	月份，十进制格式。	08
%M	分钟，十时制格式，范围为00~59。	59
%n	换行符。	换行符
%p	本地的AM（上午）或PM（下午）。	AM/PM
%r	12小时制的时间组合，与 `%I:%M:%S %p` 相同。	11:59:59 AM
%R	小时和分钟组合，与 `%H:%M` 相同。	23:59
%S	秒数，十进制，范围为00~59。	59
%t	TAB符。	TAB符
%y	年份，十进制，不带世纪，范围为00~99。	04；98
%Y	年份，十进制。	2004；1998
%C	十进制世纪，范围为00~99。	16
%e	每月第几天，十进制格式，范围为1~31。如果是个位数字，前面需要加空格。	7，31
%j	一年天数的十进制表示，范围为001~366。	365
%u	星期的十进制表示，范围为1~7，1 表示周一。	2
%U	每年的第几周，星期天是一周的开始。范围为00~53。	23
%V	每年的第几周，星期一是一周的开始。如果一月份刚开始的一周>=4天，则认为是第1周，否则认为下一个星期一是第1周。范围为01~53。	24
%w	星期几，十进制格式 ，范围为0~6，0代表周日。	5
%W	每年的第几周，星期一是一周的开始。范围为00~53。	23
%c	标准的日期、时间。	需要指定长日期、短日期等更多信息，可以考虑用上面支持的格式更精确表达。
%x	标准的日期。	需要指定长日期、短日期等更多信息，可以考虑用上面支持的格式更精确表达
%X	标准的时间。	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。	1476187251

示例

常见的日志时间格式、示例及对应的时间表达式如下：

日志时间格式	示例	时间表达式
自定义	2017-12-11 15:05:07	%Y-%m-%d %H:%M:%S
自定义	[2017-12-11 15:05:07.012]	[%Y-%m-%d %H:%M:%S
RFC822	02 Jan 06 15:04 MST	%d %b %y %H:%M
RFC822Z	02 Jan 06 15:04 -0700	%d %b %y %H:%M
RFC850	Monday, 02-Jan-06 15:04:05 MST	%A, %d-%b-%y %H:%M:%S
RFC1123	Mon, 02 Jan 2006 15:04:05 MST	%A, %d-%b-%y %H:%M:%S
RFC3339	2006-01-02T15:04:05Z07:00	%Y-%m-%dT%H:%M:%S
RFC3339Nano	2006-01-02T15:04:05.999999999Z07:00	%Y-%m-%dT%H:%M:%S

3.1.4.11. 导入历史日志文件

Logtail默认只采集增量的日志文件，如果您需要导入历史文件，可使用Logtail自带的导入历史文件功能。

前提条件

Logtail版本需在0.16.15（Linux）或1.0.0.1（Windows）及以上，若低于此版本，请先升级至最新版本。

背景信息

Logtail基于事件进行文件采集，事件通常由监听或定期轮询文件修改产生。除以上方式外，Logtail还支持从本地文件中加载事件，以此驱动日志采集。历史文件采集就是基于本地事件加载实现的功能。

导入历史日志的步骤需要在Logtail的安装目录下操作，该目录在不同操作系统中位于不同位置。

- Linux系统：/usr/local/ilogtail
- Windows系统：
  - 32位：C:\Program Files\Alibaba\Logtail
  - 64位：C:\Program Files (x86)\Alibaba\Logtail

> **说明**
> - 本地事件导入最长触发延迟为1分钟。
> - 由于加载本地配置属于特殊行为，Logtail会向服务器发送 `LOAD_LOCAL_EVENT_ALARM` 以提醒用户。
> - 若您导入的文件量较大，建议参见配置启动参数修改Logtail启动配置，提升CPU、内存上限，建议CPU调整至2.0或以上，内存调整至512 MB或以上。

操作步骤

1. 创建采集配置。
   请根据文本日志采集流程创建采集配置并应用到机器组，若该配置只用来导入历史文件，可以设置一个不存在的采集目录。
2. 获取配置唯一标识。
   采集配置的唯一标识可在Logtail安装目录下的文件user_log_config.json中获取。Linux系统可在该目录下执行命令**grep**查看；Windows系统可以通过记事本等工具查看。
   以Linux系统为例，查看标识的操作如下：

   ```
   grep "##" /usr/local/ilogtail/user_log_config.json | awk '{print $1}'
                       "##1.0##log-config-test$multi"
                       "##1.0##log-config-test$ecs-test"
                       "##1.0##log-config-test$metric_system_test"
                       "##1.0##log-config-test$redis-status"
   ```

3. 添加本地事件。
   本地事件保存在Logtail安装目录下的文件local_event.json中，类型为标准JSON，格式为：

```
[
  {
    "config" : "${your_config_unique_id}",
    "dir" : "${your_log_dir}",
    "name" : "${your_log_file_name}"
   },
  {
   ...
   }
   ...
]
```

- 配置项

配置项	说明	示例
config	步骤2获取的配置唯一标识。	##1.0##log-config-test$ecs-test
dir	日志所在文件夹。 说明 文件夹不能以 / 结尾。	/data/logs
name	日志文件名，支持通配符。	access.log.2018-08-08 、 access.log*

> 说明 为防止Logtail加载无效的JSON，建议您将本地事件配置保存在临时文件中，编辑完成后拷贝到local_event.json中。

- 配置示例

  以Linux系统为例，添加本地事件的步骤如下，Windows系统可直接通过记事本等工具修改local_event.json文件。

```
$ cat /usr/local/ilogtail/local_event.json
[
  {
    "config": "##1.0##log-config-test$ecs-test",
    "dir": "/data/log",
    "name": "access.log*"
  },
  {
    "config": "##1.0##log-config-test$tmp-test",
    "dir": "/tmp",
    "name": "access.log.2017-08-09"
  }
]
```

后续步骤

- 检查Logtail是否加载配置

  通常情况下，本地 local_event.json 保存后，Logtail会在1分钟内将本地配置文件加载到内存中，并将 local_event.json 内容清空。

  您可以通过以下方式检查Logtail是否已经读取事件：

  - 若 local_event.json 文件被清空，说明Logtail已经读取到事件信息。
  - 检查Logtail安装目录中的文件ilogtail.LOG中是否包含 process local event 关键字。若 local_event.json 被清空但未查询到该组关键字，可能因为您的本地配置文件内容不合法而被过滤。

- 配置被加载但未采集到数据

  若Logtail已经加载配置但数据未被采集到，可能由以下几种原因造成：

  - 配置不合法。

- 本地配置 `config` 不存在。
- 日志文件不在Logtail采集配置已设定的路径下。
- 该日志文件被Logtail采集过。

3.1.4.12. 生成主题

日志可以通过日志主题（Topic）来划分。您可以在写入时指定日志主题，并在查询时指定查询的日志主题。

Topic生成方式

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

- **空-不生成Topic**

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

- **机器组Topic属性**

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

  如选择**机器组Topic属性**，Logtail上报数据时会将机器所在机器组的Topic属性作为主题名称上传至日志服务，在查询日志时需要指定Topic，即需要指定目标机器组Topic属性为查询条件。

- **文件路径正则**
  - 该方式用于区分具体用户或实例产生的日志数据。如果服务日志根据不同的用户或者实例将日志记录在不同目录下面，但是只要下级目录、日志文件名称相同，日志服务在收集日志文件时就无法明确区分日志内容是由那个用户哪个产生的。此时可以设置**Topic生成方式**为**文件路径正则**，并且输入文件路径的正则表达式（此处的正则表达式需要完整匹配文件路径），配置Topic为实例名称。
  - 当选择**文件路径正则**主题生成方式时，Logtail上报数据时会将实例名称作为主题名称上传至日志服务。根据您的目录结构和配置，会生成不同的Topic，在查询日志时需要指定主题名称为实例名称。例如，在以下目录结构中，根据不同的用户或者实例将日志记录在不同目录下面。

    ```
    /logs
      | - /userA/serviceA
        | - service.log
      | - /userB/serviceA
        | - service.log
      | - /userC/serviceA
        | - service.log
    ```

  - 如果文件路径中有多个字段需要单独提取，可以使用多层提取方式，多层提取使用 `?P<key>` 的方式 （key只支持小写字母和数字的组合）。例如：

    ```
    /home/admin/serviceA/userB/access.log
    \/home\/admin\/(?P<service>[^\/]+)/(?P<user>[^/]+)/.*
    ```

    则日志会带上自定义的tag：

    ```
    "__tag__ : service : serviceA"
    "__tag__ : user : userB"
    ```

    > **说明** 支持的Logtail版本为0.16.19及以上版本。

  - 如果仅配置文件路径为/logs，文件名称为service.log，将三个service目录下的日志内容实时收集至服务端，但是无法明确区分日志内容具体由哪个用户或者实例产生。此时可以另外设置**Topic生成方式**为**文件路径正则**，并且输入正则表达式 `\/(.*)\/serviceA\/.*` 提取实例名称。设置后会为不同目录下的日志生成不同的Topic，分别为“userA”、“userB”和“userC”，查询日志时可以指定Topic查询。

  > **说明** 文件路径的正则表达式中，需要对字符 `/` 进行转义。

- **静态主题生成**

  设置**Topic生成方式**为**文件路径正则**，在**自定义正则**中输入 `customized:// + 自定义主题名` 即可。

> **说明** 支持的Logtail版本为0.16.21及以上版本。

设置日志Topic

1. 根据文本日志采集流程，通过控制台配置Logtail。
   如您需要配置Topic生成方式为**机器组Topic属性**，请在创建机器组/修改机器组页面中配置**机器组Topic**。
2. 在接入数据时的Logtail配置步骤中，展开**高级选项**，在**Topic生成方式**中选择Topic的生成方式。

3.1.5. 自定义插件

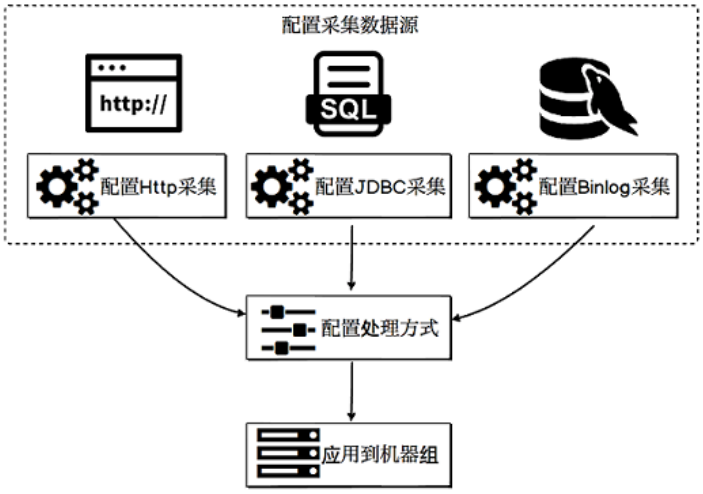
背景信息

日志服务Logtail除了支持采集文本日志及syslog之外，也通过Logtail插件支持配置若干数据源，例如Http、MySQL查询结果和MySQL Binlog。

通过配置采集HTTP数据，并将处理结果上传到日志服务，可以进行实时的服务可用性检测和持续的可用性监控；配置MySQL查询结果为数据源，可以根据自增ID或时间等标志进行增量数据同步；配置SQL数据源以同步MySQL Binlog，可以增量订阅数据库改动，并进行实时查询与分析。

> **说明** 此功能目前仅支持Linux，依赖Logtail 0.16.0及以上版本，版本查看与升级参见在 Linux 上安装 Logtail。

配置流程



1. 配置输入源采集方式。

   每种不同的数据源具有不同的采集配置格式，请按照您的数据源类型选择对应的采集配置格式。

   - 采集 MySQLBinlog数据
   - 采集 MySQL查询结果
   - 采集 HTTP数据
   - 采集 容器标准输出
2. 配置处理方式。

   Logtail对于Binlog、MySQL查询结果、Nginx监控和HTTP输入源提供了统一的数据处理配置。用户可对一个输入源配置多个处理方式，各类输入源均支持所有类型的处理方式。Logtail会根据配置顺序逐一执行各个处理方式。

   详细说明请参考通用处理配置。
3. 应用到机器组。

   配置采集方式和处理方式之后，保存并应用到指定机器组，Logtail会自动拉取配置并开始采集。

3.1.5.1. MySQL Binlog方式

MySQL Binlog同步类似Canal功能，通过MySQL slave来同步Binlog，同步效率较高。

功能特点

- 通过Binlog增量采集数据库更新操作数据，性能优越。支持MySQL协议的数据库（例如阿里云RDS）。
- 支持多种数据库过滤方式，如正则表达式。
- 支持Binlog位点设置。
- 支持通过Checkpoint机制同步保存状态。

使用限制

- 此功能目前仅支持Linux，依赖Logtail 0.16.0及以上版本，版本查看与升级请参见安装Logtail（Linux系统）。
- MySQL必须开启Binlog，且Binlog必须为row模式（默认RDS已经开启）。

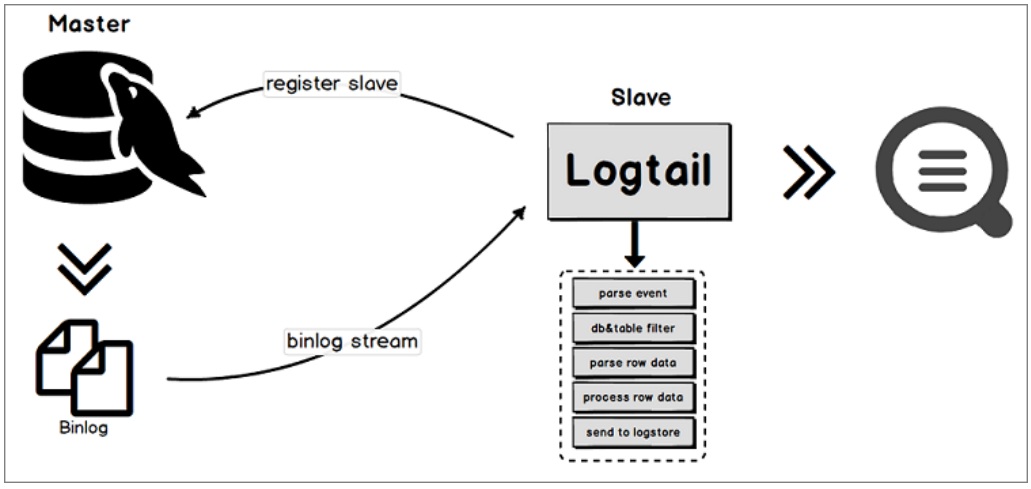
```
# 查看是否开启Binlog
mysql> show variables like "log_bin";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | ON    |
+---------------+-------+
1 row in set (0.02 sec)
# 查看Binlog类型
mysql> show variables like "binlog_format";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | ROW   |
+---------------+-------+
1 row in set (0.03 sec)
```

- ServerID必须唯一。确保需要同步的MySQL所有Slave的ID不重复。
- RDS限制
  - 无法直接在RDS服务器上安装Logtail，您需要将Logtail安装在能连通RDS实例的任意一台ECS上。
  - RDS备库不支持Binlog采集，您需要配置RDS主库进行采集。

实现原理

Logtail内部实现了MySQL Slave的交互协议，具体流程如下所示。

1. Logtail将自己伪装为MySQL的Slave节点并向MySQL的master发送dump协议。
2. MySQL的master收到dump请求后，会将自身的Binlog实时发送给Logtail。
3. Logtail对Binlog进行事件解析、过滤、数据解析等，并将解析好的数据上传到日志服务。



应用场景

适用于数据量较大且性能要求较高的数据同步场景。

- 增量订阅数据库改动进行实时查询与分析。
- 数据库操作审计。
- 使用日志服务对数据库更新信息进行自定义查询分析、可视化、对接下游流计算、导入MaxCompute离线计算、导入OSS长期存储等。

数据可靠性

建议您启用MySQL服务器的全局事务ID（GTID）功能，并将Logtail升级到0.16.15及以上版本以保证数据可靠性，避免因主备切换造成的数据重复采集。

- 数据漏采集：Logtail与MySQL服务器之间的网络长时间中断时，可能会产生数据漏采集情况。

  MySQL Binlog插件通过伪装成MySQL的slave节点持续从master节点获取binlog数据。因此，Logtail会和master节点之间建立连接以获取数据。如果Logtail和master之间的网络发生中断，master节点依然会不断地产生新的binlog数据并且回收旧的binlog数据。当网络恢复且Logtail重连成功后，Logtail会使用自身的checkpoint去向master请求更多的binlog数据。而由于长时间的网络中断，它所需要的数据很可能已经被回收了，这时就会触发Logtail的异常恢复机制。在异常恢复机制中，Logtail会从master获取最近的binlog位置，以它为起点继续采集，这样就会跳过checkpoint和最近的binlog位置之间的数据，导致数据漏采集。

- 数据重复采集：如果master和slave之间的binlog序号不同步时，发生了主备切换事件，可能会产生数据重复采集情况。

  在MySQL主备同步的设置下，master会将产生的binlog同步给slave，slave收到后会存储到本地的binlog文件中。如果master和slave之间的binlog序号不同步时，发生了主备切换事件，以binlog文件名和偏移作为checkpoint的机制将导致数据重复采集。

  例如，有一段数据在master上位于 `(binlog.100, 4)` 到 `(binlog.105, 4)` 之间，而在slave上是 `(binlog.1000, 4)` 到 `(binlog.1005, 4)` ，并且Logtail已经从master获取了这部分数据，将本地checkpoint更新到了 `(binlog.105, 4)` 。如果此时发生了主备切换且无任何异常发生，Logtail将会继续使用本地checkpoint `(binlog.105, 4)` 去向新master获取binlog信息。但是因为新master上的 `(binlog.1000, 4)` 到 `(binlog.1005, 4)` 这部分数据的序号都大于Logtail所请求的序号，它会把它们返回给Logtail，这就引起了binlog数据的重复采集。

配置项

MySQL Binlog方式输入源类型为： `service_canal` 。

配置项	类型	是否必须	说明
Host	string	否	数据库主机，默认为127.0.0.1。
Port	int	否	数据库端口，默认为3306。

User	string	否	数据库用户名，默认为root。
			需保证配置的用户具有数据库读权限以及MySQL REPLICATION权限，示例如下。
			`CREATE USER canal IDENTIFIED BY 'canal';`
			`GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'canal'@'%';`
			`-- GRANT ALL PRIVILEGES ON *.* TO 'canal'@'%' ;`
			`FLUSH PRIVILEGES;`

Password	string	否	数据库密码，默认为空。 如果安全需求较高，建议将SQL访问用户名和密码配置为xxx，待配置同步至本地机器后，在/usr/local/ilogtail/user_log_config.json文件找到对应配置进行修改。 **说明** • 修改完毕后请执行**sudo /etc/init.d/ilogtaild stop; sudo /etc/init.d/ilogtaild start**命令重启Logtail。 • 如果您在Web端修改了此配置，配置同步至本地后会覆盖当前本地的配置，请根据需要手动修改本地配置。
ServerID	int	否	Logtail伪装成的Mysql Slave ID，默认为125。 **说明** ServerID对于MySQL数据库必须唯一，否则会同步失败。
IncludeTables	string 数组	是	包含的表名称（包括db，例如 `test_db.test_table` ），为正则表达式。若某个表不符合IncludeTables的任一条件则该表不会被采集。如果您希望采集所有表，请将此参数指定为 `.*\\..*` 。 **说明** 若需要完全匹配，请在前后分别加上 `^` `$` ，例如 `^test_db\\.test_table$` 。

ExcludeTables	string 数组	否	忽略的表名称（包括db，例如 `test_db.test_table` ），为正则表达式。若某个表符合ExcludeTables的任一条件则该表不会被采集。不设置时默认收集所有表。 **说明** 若需要完全匹配，请在前后分别加上 `^` `$` ，例如 `^test_db\\.test_table$` 。

StartBinName	string	否	首次采集的Binlog文件名，不设置时默认从当前时间点开始采集。
			如果想从指定位置采集，可以查看当前的Binlog以及offset，并将 `StartBinName` 、 `StartBinLogPos` 设置成对应的值，示例如下。
			`# StartBinName 设置成 "mysql-bin.000063", StartBinLogPos 设置成 0`
			`mysql> show binary logs;`
			`+------------------+-----------+`
			`| Log_name | File_size |`
			`+------------------+-----------+`
			`| mysql-bin.000063 | 241 |`
			`| mysql-bin.000064 | 241 |`
			`| mysql-bin.000065 | 241 |`
			`| mysql-bin.000066 | 10778 |`
			`+------------------+-----------+`
			`4 rows in set (0.02 sec)`
			说明
			当指定 `StartBinName` 时，第一次采集会产生较大流量。

StartBinLogPos	int	否	首次采集的Binlog文件名的offset，默认为0。
EnableGTID	bool	否	是否附加全局事务ID。默认为true，为false时上传的数据将不附加全局事务ID。
EnableInsert	bool	否	是否收集insert事件的数据。默认为true，为false时将不采集insert事件数据。
EnableUpdate	bool	否	是否收集update事件的数据。默认为true，为false时将不采集update事件数据。
EnableDelete	bool	否	是否收集delete事件的数据。默认为true，为false时将不采集delete事件数据。
EnableDDL	bool	否	是否收集DDL（data definition language）事件数据。 默认为false表示不收集DDL事件数据。 **说明** 该选项不支持 `IncludeTables` 和 `ExcludeTables` 过滤。
Charset	string	否	编码方式，默认为 `utf-8` 。
TextToString	bool	否	是否将text类型数据转换成string，默认为false。
PackValues	bool	否	是否将事件的数据打包成JSON格式。默认为false表示不打包。如果开启此功能，Logtail会将事件数据以JSON格式集中打包到data和old_data两个字段中，其中old_data仅在row_update事件中有意义。 示例：假设数据表有三列数据c1，c2，c3，在不开启此功能的情况下，row_insert事件数据中会有c1，c2，c3三个字段，而开启此功能后，c1，c2，c3会被统一打包为data字段，值为 `{"c1":"...", "c2": "...", "c3": "..."}` 。 **说明** 该参数仅支持Logtail 0.16.19及以上版本。

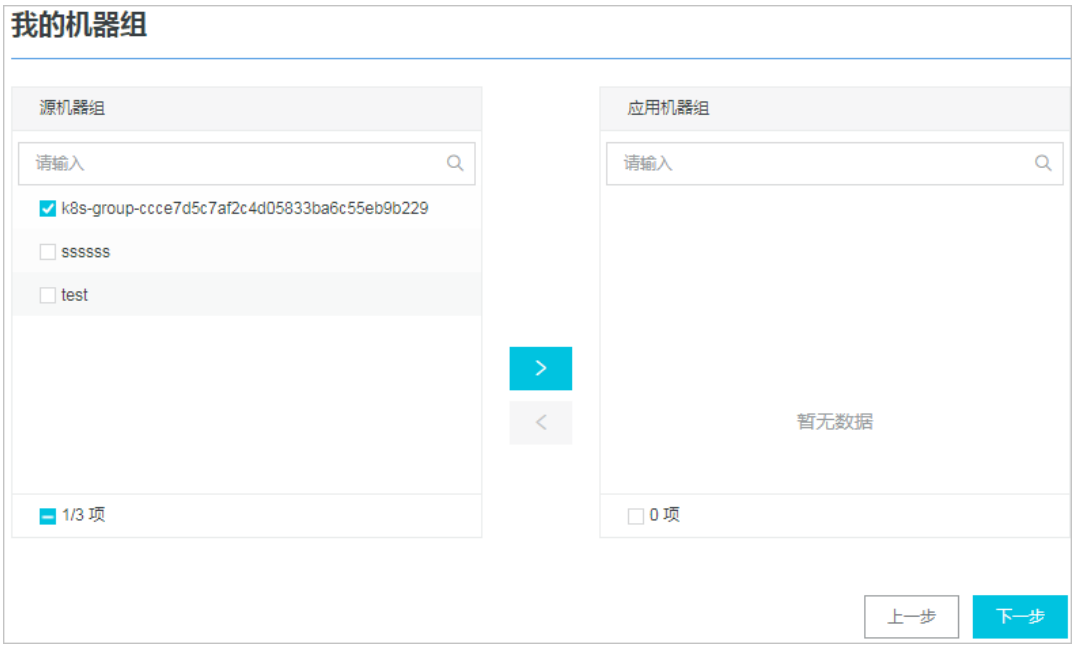
EnableEventMeta	bool	否	是否采集事件的元数据，默认为false表示不采集。 Binlog事件的元数据包括 `event_time` 、 `event_log_position` 、 `event_size` 以及 `event_server_id` 。 **说明** 该参数仅支持Logtail 0.16.21及以上版本。

操作步骤

从RDS中同步 `user_info` 库中不以 `_inner` 结尾的表，配置如下。

1. 登录日志服务控制台。
2. 选择输入源。
   单击**接入数据**按钮，并在**接入数据**页面中选择**MySQL BinLog**。
3. 选择目标Project和Logstore，配置完成后，单击**下一步**。
   您也可以单击**立即创建**，重新创建Project和Logstore。
   如果您是通过日志库下的**数据接入**后的加号图标进入采集配置流程，系统会直接跳过该步骤。
4. 创建机器组，单击**下一步**。
   在创建机器组之前，您需要首先确认已经安装了Logtail。
   请根据界面提示进行安装。更多信息，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。
   安装完Logtail后，单击**确认安装完毕**创建机器组。如果您之前已经创建好机器组 ，也可以直接单击**使用现有机器组**。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

   重要
   如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为**FAIL**，您可单击**自动重试**。如果显示**FAIL**，请参见Logtail机器组无心跳处理。



6. 设置数据源。
   填写配置名称和插件配置。
   插件配置输入框中已为您提供配置模板，请根据您的需求替换配置参数。

```
{
 "inputs": [
     {
         "type": "service_canal",
         "detail": {
             "Host": "************.mysql.rds.aliyuncs.com",
             "Port": 3306,
             "User" : "root",
             "ServerID" : 56321,
             "Password": "********",
             "IncludeTables": [
                 "user_info\\..*"
             ],
             "ExcludeTables": [
                 ".*\\.\\S+_inner"
             ],
             "TextToString" : true,
             "EnableDDL" : true
         }
     }
 ]
}
```

- inputs部分为采集配置，必选项。请根据您的数据源配置对应的采集语句。
- processors部分为处理配置，可选项。请参见处理采集数据配置一种或多种采集方式。

7. **查询和分析配置，包括配置索引。配置完成后，单击下一步。**

   日志服务默认开启全文索引。您也可以根据采集到的日志，手动或者自动设置字段索引。更多信息，请参见开启并配置索引。

   > **说明**
   > - 如果您要查询分析日志，那么全文索引和字段索引属性必须至少启用一种。同时启用时，以字段索引为准。
   > - 索引类型为long、double时，大小写敏感和分词符属性无效。

8. **可选：修改本地配置。**
   如果您没有在插件配置中输入真实的URL、账号、密码等信息，需要在采集配置下发到本地后手动修改其中的内容。

   > **说明** 如果您服务端输入的是真实信息，则无需此步骤。

   i. **登录Logtail所在服务器，查找** `/usr/local/ilogtail/user_log_config.json` **文件中** `service_canal` **关键词，修改下述对应的** `Host` **、** `User` **、** `Password` **等字段。**

   ii. **执行以下命令重启Logtail。**

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

Binlog采集配置已经完成，如果您的数据库存在对应的修改操作，则Logtail会立即将变化的数据采集到日志服务。

> **说明** Logtail默认采集Binlog的增量数据，如查看不到数据请确认在配置更新后数据库的表存在修改操作。

元数据字段

Binlog采集会将一些元数据和日志一起上传，具体上传的元数据列表如下。

字段	说明	示例
`_host_`	数据库host名称。	`*********.mysql.rds.aliyuncs.com`
`_db_`	数据库名称。	`my-database`
`_table_`	表的名称。	`my-table`

event	事件类型。	row_update、row_insert、row_delete等
_id_	本次采集的自增ID，从0开始，每次采集一个binlog事件后加1。	1
gtid	GTID。	7d2ea78d-b631-11e7-8afb-00163e0eef52:536
filename	Binlog文件名。	binlog.001
offset	Binlog文件偏移量，该值只会在每次commit后更新。	12876

示例

按照以上操作步骤配置处理方式后，对 user_info 下的 SpecialAlarm 表分别执行 INSERT、UPDATE、DELETE 操作。数据库表结构、数据库操作及Logtail采集的样例如下。

- 表结构

```
CREATE TABLE `SpecialAlarm` (
`id` int(11) unsigned NOT NULL AUTO_INCREMENT,
`time` datetime NOT NULL,
`alarmtype` varchar(64) NOT NULL,
`ip` varchar(16) NOT NULL,
`count` int(11) unsigned NOT NULL,
PRIMARY KEY (`id`),
KEY `time` (`time`) USING BTREE,
KEY `alarmtype` (`alarmtype`) USING BTREE
) ENGINE=MyISAM AUTO_INCREMENT=1;
```

- 数据库操作
  执行INSERT、DELETE和UPDATE三种操作。

```
insert into specialalarm (`time`, `alarmType`, `ip`, `count`) values(now(), "NO_ALARM", "10.10.**.***"
, 55);
delete from specialalarm where id = 4829235  ;
update specialalarm set ip = "10.11.***.**" where id = "4829234";
```

  为 zc.specialalarm 创建一个索引。

```
ALTER TABLE `zc`.`specialalarm`
ADD INDEX `time_index` (`time` ASC);
```

- 样例日志
  在数据预览或查询分析页面中，可以看到对应每种操作的样例日志如下。
  - INSERT语句

```
__source__:  10.30.**.**
__tag__:__hostname__:  iZbp145dd9fccu*****
__topic__:
_db_:  zc
_event_:  row_insert
_gtid_:  7d2ea78d-b631-11e7-8afb-00163e0eef52:536
_host_:  *********.mysql.rds.aliyuncs.com
_id_:  113
_table_:  specialalarm
alarmtype:  NO_ALARM
count:  55
id:  4829235
ip:  10.10.***.***
time:  2017-11-01 12:31:41
```

- DELETE语句

```
__source__:  10.30.**.**
__tag__:__hostname__:  iZbp145dd9fccu****
__topic__:
_db_:  zc
_event_:  row_delete
_gtid_:  7d2ea78d-b631-11e7-8afb-00163e0eef52:537
_host_:  *********.mysql.rds.aliyuncs.com
_id_:  114
_table_:  specialalarm
alarmtype:  NO_ALARM
count:  55
id:  4829235
ip:  10.10.**.***
time:  2017-11-01 12:31:41
```

- UPDATE语句

```
__source__:  10.30.**.**
__tag__:__hostname__:  iZbp145dd9fccu****
__topic__:
_db_:  zc
_event_:  row_update
_gtid_:  7d2ea78d-b631-11e7-8afb-00163e0eef52:538
_host_:  *********.mysql.rds.aliyuncs.com
_id_:  115
_old_alarmtype:  NO_ALARM
_old_count:  55
_old_id:  4829234
_old_ip:  10.10.22.133
_old_time:  2017-10-31 12:04:54
_table_:  specialalarm
alarmtype:  NO_ALARM
count:  55
id:  4829234
ip:  10.11.***.***
time:  2017-10-31 12:04:54
```

- DDL（data definition language）语句

```
__source__:  10.30.**.**
__tag__:__hostname__:  iZbp145dd9fccu****
__topic__:
_db_:  zc
_event_:  row_update
_gtid_:  7d2ea78d-b631-11e7-8afb-00163e0eef52:539
_host_:  *********.mysql.rds.aliyuncs.com
ErrorCode:  0
ExecutionTime:  0
Query:  ALTER TABLE `zc`.`specialalarm`
ADD INDEX `time_index` (`time` ASC)
StatusVars:
```

注意事项

建议您适当放开对Logtail的资源限制以应对流量突增，避免Logtail因为资源超限被强制重启，对您的数据造成不必要的风险。

资源限制可通过修改/usr/local/ilogtail/ilogtail_config.json文件实现，修改完成后执行命令**sudo /etc/init.d/ilogtaild stop;sudo /etc/init.d/ilogtaild start**重启Logtail。

如下示例表示将CPU的资源限制放宽到双核，将内存资源的限制放宽到2048MB。

```
{
    ...
    "cpu_usage_limit":2,
    "mem_usage_limit":2048,

    ...
}
```

3.1.5.2. MySQL查询结果

本文介绍如何通过日志服务控制台创建Logtail采集配置来采集MySQL查询结果。

背景信息

以SQL形式定期采集数据库中的数据，根据SQL可实现各种自定义采集方式。

> **说明** 此功能目前仅支持Linux，依赖Logtail 0.16.0及以上版本，版本查看与升级参见安装Logtail（Linux系统）。

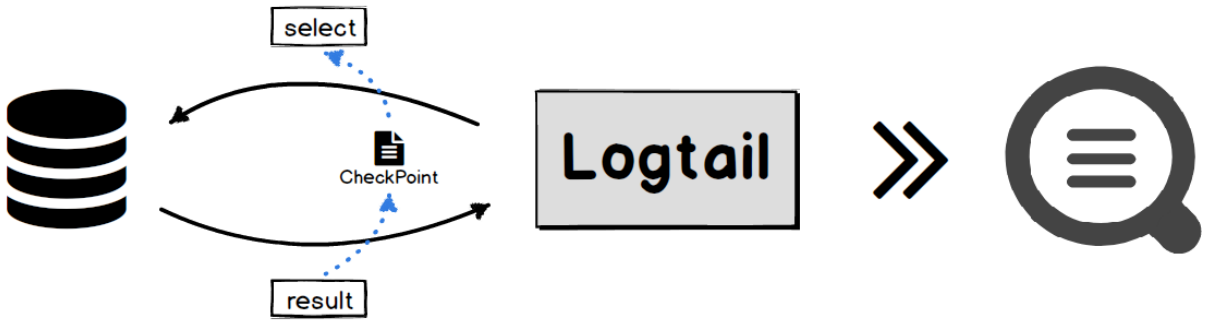
功能

支持提供MySQL接口的数据库，包括RDS：

- 支持分页设置
- 支持时区设置
- 支持超时设置
- 支持checkpoint状态保存
- 支持SSL
- 支持每次最大采集数量限制

实现原理

如下图所示，Logtail内部根据用户配置定期执行指定的SELECT语句，将SELECT返回的结果作为数据上传到日志服务。



Logtail在获取到执行结果时，会将结果中配置的CheckPoint字段保存在本地，当下次执行SQL时，会将上一次保存的CheckPoint带入到SELECT语句中，以此实现增量数据采集。

应用场景

- 根据自增ID或时间等标志进行增量数据同步。
- 自定义筛选同步。

参数说明

该输入源类型为：`service_mysql`。

参数	类型	必选或可选	参数说明
Address	string	可选	MySQL地址，默认为”127.0.0.1:3306”。
User	string	可选	数据库用户名，默认为”root”。

Password	string	可选	数据库密码，默认为空。
DialTimeOutMs	int	可选	数据库连接超时时间，单位为ms，默认为5000ms。
ReadTimeOutMs	int	可选	数据库连接超时时间，单位为ms，默认为5000ms。
StateMent	string	必选	SQL语句。
Limit	bool	可选	是否使用Limit分页，默认为false。
PageSize	int	可选	分页大小，Limit为true时必须配置。
MaxSyncSize	int	可选	每次同步最大记录数，为0表示无限制，默认为0。
CheckPoint	bool	可选	是否使用checkpoint，默认为false。
CheckPointColumn	string	可选	checkpoint列名称，CheckPoint为true时必须配置。
CheckPointColumnType	string	可选	checkpoint列类型，支持 `int` 和 `time` 两种类型。
CheckPointStart	string	可选	checkpoint初始值。
CheckPointSavePerPage	bool	可选	为true时每次分页时保存一次checkpoint，为false时每次同步完后保存checkpoint。
IntervalMs	int	必选	同步间隔，单位为ms。

使用限制

- 建议使用 `Limit` 分页，使用 `Limit` 分页时，SQL查询会自动在 `StateMent` 后追加LIMIT语句。
- 使用 `CheckPoint` 时， `StateMent` 中SELECT出的数据中必须包含checkpoint列，且where条件中必须包含checkpoint列，该列的值填 `?`

  例如checkpoint为”id”， `StateMent` 为 `SELECT * from ... where id > ?` 。
- `CheckPoint` 为true时必须配置 `CheckPointColumn` 、 `CheckPointColumnType` 、 `CheckPointStart` 。
- `CheckPointColumnType` 只支持 `int` 和 `time` 类型， `int` 类型内部存储为int64， `time` 支持MySQL的date、datetime、time。

操作步骤

从MySQL中增量同步logtail.VersionOs中count > 0的数据，同步间隔为10s，checkpoint起始时间为2017-09-25 11:00:00，请求方式为分页请求，每页100，每次分页后保存checkpoint。具体配置步骤如下。

1. 登录日志服务控制台。
2. 选择数据源。
   单击**接入数据**按钮，并在**接入数据**页面中选择**MYSQL查询结果**。
3. 选择目标Project和Logstore，配置完成后，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和Logstore。

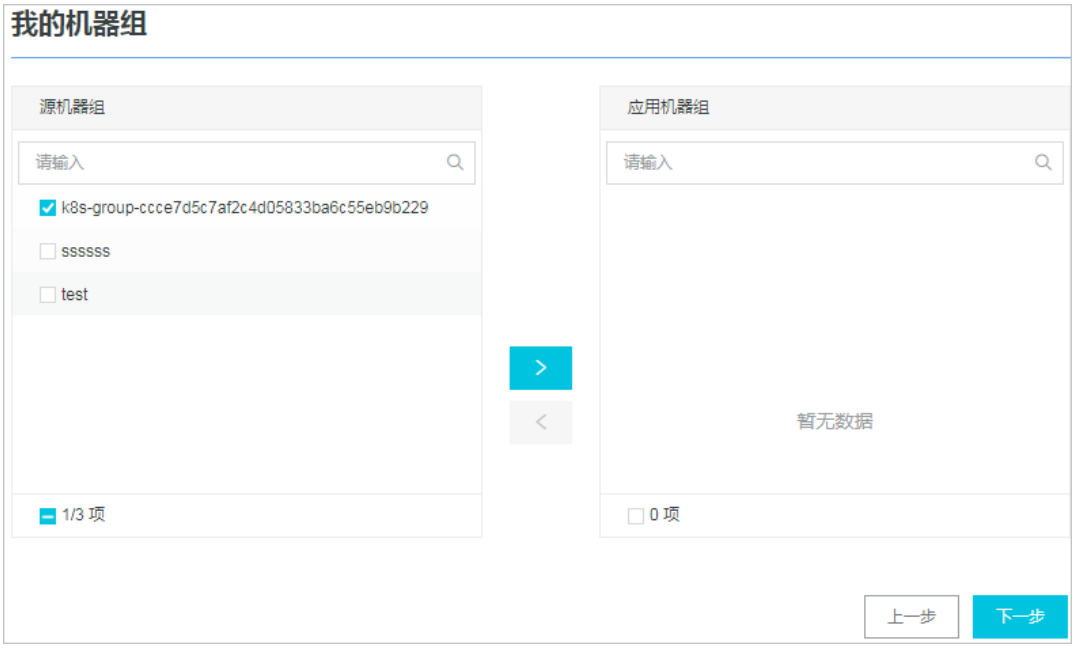
   如果您是通过日志库下的**数据接入**后的加号图标进入采集配置流程，系统会直接跳过该步骤。
4. 创建机器组，单击**下一步**。

   在创建机器组之前，您需要首先确认已经安装了Logtail。

   请根据界面提示进行安装。更多信息，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。

   安装完Logtail后，单击**确认安装完毕**创建机器组。如果您之前已经创建好机器组，也可以直接单击**使用现有机器组**。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **重要**
> 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为**FAIL**，您可单击**自动重试**。如果显示**FAIL**，请参见Logtail机器组无心跳处理。



6. 设置数据源。
   - **插件配置**输入框中已为您提供配置模板，请根据您的需求替换配置参数信息。
   - `inputs` 部分为采集配置，是必选项；`processors` 部分为处理配置，是可选项。采集配置部分需要按照您的数据源配置对应的采集语句，处理配置部分请参见处理采集数据配置一种或多种采集方式。

   > **说明** 若安全需求较高，建议将SQL访问用户名/密码配置为xxx，待配置同步至本地机器后，在/usr/local/ilogtail/user_log_config.json文件找到对应配置进行修改。

   示例配置如下：

```
{
 "inputs": [
     {
         "type": "service_mysql",
         "detail": {
             "Address": "********:3306",
             "User": "logtail",
             "Password": "*******",
             "DataBase": "logtail",
             "Limit": true,
             "PageSize": 100,
             "StateMent": "SELECT * from logtail.VersionOs where time > ?",
             "CheckPoint": true,
             "CheckPointColumn": "time",
             "CheckPointStart": "2017-09-25 11:00:00",
             "CheckPointSavePerPage": true,
             "CheckPointColumnType": "time",
             "IntervalMs": 10000
         }
     }
 ]
}
```

7. 查询和分析配置，包括配置索引。配置完成后，单击**下一步**。

日志服务默认开启全文索引。您也可以根据采集到的日志，手动或者自动设置字段索引。更多信息，请参见开启并配置索引。

> **说明**
> - 如果您要查询分析日志，那么全文索引和字段索引属性必须至少启用一种。同时启用时，以字段索引为准。
> - 索引类型为long、double时，大小写敏感和分词符属性无效。

8. **可选：**修改本地配置。
   如果您没有在**数据源设置**页面输入真实的URL、账号、密码等信息，需要在采集配置下发到本地后手动修改其中的内容。

   > **说明** 如果您服务端输入的是真实信息，则无需此步骤。

   i. 登录Logtail所在服务器，查找 `/usr/local/ilogtail/user_log_config.json` 文件中 `service_mysql` 关键词，修改下述对应的 `Address` 、 `User` 、 `Password` 等字段。

   ii. 执行以下命令重启Logtail。

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

示例

例如，按照以上操作步骤配置处理方式后，即可在日志服务控制台查看采集处理过的日志数据。表结构和Logtail采集的日志样例如下。

- 表结构

```
CREATE TABLE `VersionOs` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT COMMENT 'id',
  `time` datetime NOT NULL,
  `version` varchar(10) NOT NULL DEFAULT '',
  `os` varchar(10) NOT NULL,
  `count` int(11) unsigned NOT NULL,
  PRIMARY KEY (`id`),
  KEY `timeindex` (`time`)
)
```

- 样例输出

```
"count":  "4"
"id":  "721097"
"os":  "Windows"
"time":  "2017-08-25 13:00:00"
"version":  "1.3.0"
```

3.1.5.3. Syslog输入源

Logtail支持通过自定义插件采集syslog。

前提条件

已经在服务器上安装0.16.13及以上版本的Logtail。

简介

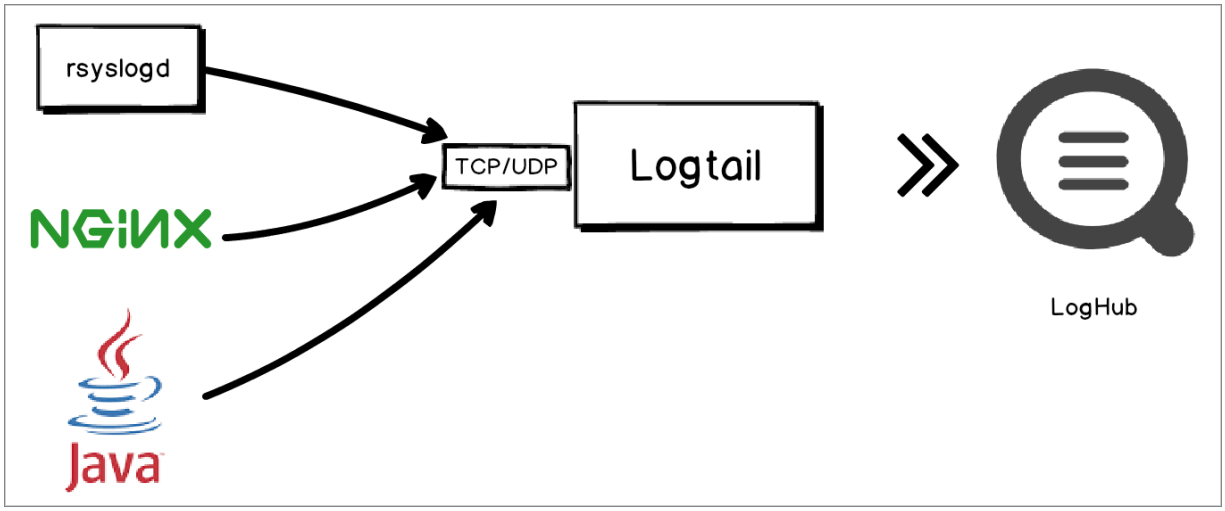
在Linux上，本地的syslog数据可以通过rsyslog等syslog agent转发到指定服务器IP地址和端口。为指定服务器添加Logtail配置之后，Logtail插件会以TCP或UDP协议接收转发过来的syslog数据。并且syslog插件能够将接收到的数据按照指定的syslog协议进行解析，提取日志中的facility、tag（program）、severity、content等字段。syslog协议支持RFC3164和RFC5424。

> **说明**
> - Windows Logtail不支持syslog插件。
> - Logtail可同时配置多个syslog插件，例如同时使用TCP和UDP监听127.0.0.1:9999。

实现原理

通过插件对指定的地址和端口进行监听后，Logtail能够作为syslog服务器采集来自各个数据源的日志，包括通过rsyslog采集

的系统日志、 Nginx转发的访问日志或错误日志，以及JAVA等语言的syslog客户端转发的日志。



Logtail配置项

该插件的输入类型为：`service_syslog`。

配置项	类型	是否必须	说明
Address	string	否	指定插件监听的协议、地址和端口，Logtail插件会根据配置进行监听并获取日志数据。格式为 `[tcp/udp]://[ip]:[port]`，默认为 `tcp://127.0.0.1:9999`。 **说明** • Logtail插件配置监听的协议、地址和端口号必须与rsyslog配置文件设置的转发规则相同。 • 如果安装Logtail的服务器有多个IP地址可接收日志，可以将地址配置为0.0.0.0，表示监听服务器的所有IP地址。
ParseProtocol	string	否	指定解析所使用的协议，默认为空，表示不解析。其中： • rfc3164：指定使用RFC3164协议解析日志。 • rfc5424：指定使用RFC5424协议解析日志。 • auto：指定插件根据日志内容自动选择合适的解析协议。
IgnoreParseFailure	boolean	否	指定解析失败后的行为，默认为true。其中： • true：放弃解析直接填充所返回的content字段。 • false：会丢弃日志。

默认字段

字段名	字段类型	字段含义
hostname	string	主机名，如果日志中未提供则获取当前主机名。
program	string	对应协议中的tag字段。
priority	string	对应协议中的priority字段。
facility	string	对应协议中的facility字段。
severity	string	对应协议中的severity字段。
unixtimestamp	string	日志对应的时间戳。
content	string	日志内容，如果解析失败的话，此字段包含未解析日志的所有内容。

_ip_	string	当前主机的IP地址。

配置Logtail插件采集syslog

1. 为rsyslog添加一条转发规则。
   在需要采集syslog的服务器上修改rsyslog的配置文件/etc/rsyslog.conf，在配置文件的最后添加一行转发规则。添加转发规则后，rsyslog会将syslog转发至指定地址端口。
   - 通过当前服务器采集本机syslog：配置转发地址为127.0.0.1，端口为任意非知名的空闲端口。
   - 通过其他服务器采集本机syslog：配置转发地址为其他服务器的公网IP，端口为任意非知名的空闲端口。

   例如以下配置表示将所有的日志都通过TCP转发至127.0.0.1:9000。

   ```
   *.* @@127.0.0.1:9000
   ```

2. 执行以下命令重启rsyslog，使日志转发规则生效。

   ```
   sudo service rsyslog restart
   ```

3. 登录日志服务控制台。
4. 选择数据源类型为**自定义数据插件**。
   您可以通过如下三种方式进入数据源界面。
   - 在日志服务首页的**接入数据**区域选择数据源类型。
   - 在**Project列表**中选择并单击Project名称，在Project概览页面单击**接入数据**进入数据源界面。
   - 选择一个日志库并展开其节点，单击**接入数据**后的加号进入数据源界面。
5. 选择目标Project和Logstore，配置完成后，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和Logstore。

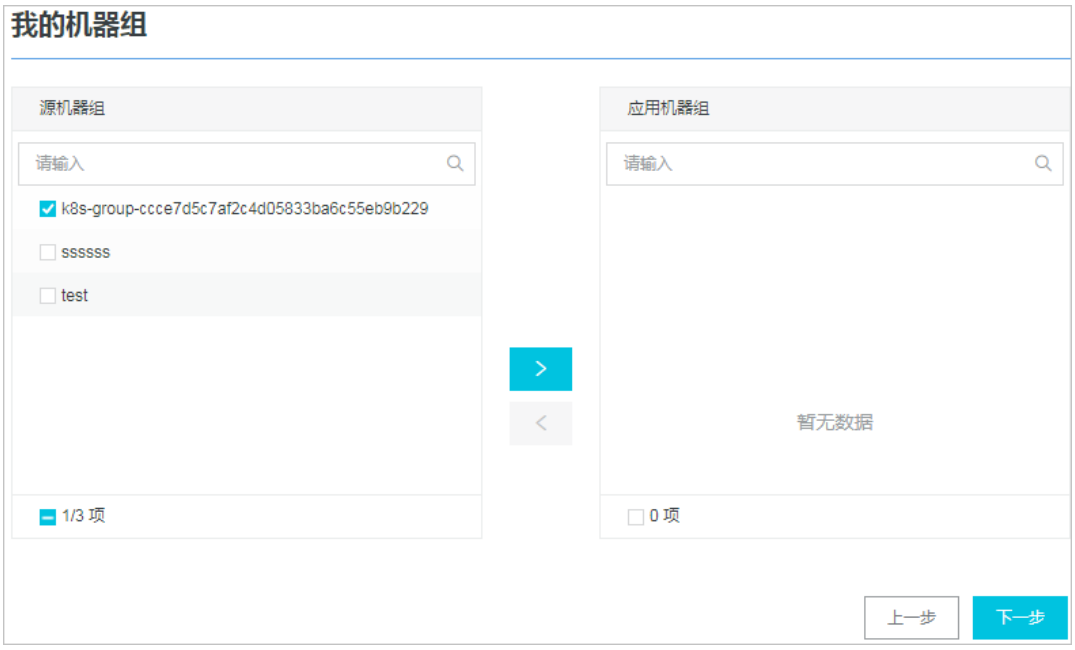
   如果您是通过日志库下的**数据接入**后的加号图标进入采集配置流程，系统会直接跳过该步骤。
6. 创建机器组，单击**下一步**。

   在创建机器组之前，您需要首先确认已经安装了Logtail。

   请根据界面提示进行安装。更多信息，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。

   安装完Logtail后，单击**确认安装完毕**创建机器组。如果您之前已经创建好机器组，也可以直接单击**使用现有机器组**。
7. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **重要**
> 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为**FAIL**，您可单击**自动重试**。如果显示**FAIL**，请参见Logtail机器组无心跳处理。



8. 数据源设置。
   请填写**配置名称**和**插件配置**。

   `inputs` 部分为采集配置，是必选项；`processors` 部分为处理配置，是可选项。采集配置部分需要按照您的数据源配置对应的采集语句，处理配置部分请参见处理采集数据配置一种或多种采集方式。

   同时监听UDP和TCP的示例配置如下：

```
{
     "inputs": [
         {
             "type": "service_syslog",
             "detail": {
                 "Address": "tcp://127.0.0.1:9000",
                 "ParseProtocol": "rfc3164"
             }
         },
         {
             "type": "service_syslog",
             "detail": {
                 "Address": "udp://127.0.0.1:9001",
                 "ParseProtocol": "rfc3164"
             }
         }
     ]
 }
```

9. 查询和分析配置，包括配置索引。配置完成后，单击**下一步**。

   日志服务默认开启全文索引。您也可以根据采集到的日志，手动或者自动设置字段索引。更多信息，请参见开启并配置索引。

> **说明**
> - 如果您要查询分析日志，那么全文索引和字段索引属性必须至少启用一种。同时启用时，以字段索引为准。
> - 索引类型为long、double时，大小写敏感和分词符属性无效。

配置Logtail插件采集Nginx日志

Nginx支持直接把访问日志以syslog协议转发到指定地址和端口。如果您希望把服务器上包括Nginx访问日志在内的所有数据都以syslog的形式集中投递到日志服务，可以创建Logtail配置并应用到该服务器所在的机器组。

1. 在Nginx服务器的 nginx.conf文件中增加转发规则。
   例如，在配置文件中增加如下内容。

```
http {
    ...

    # Add this line.
    access_log syslog:server=127.0.0.1:9000,facility=local7,tag=nginx,severity=info combined;

    ...
}
```

2. 执行以下命令重启Nginx服务，使配置生效。

```
sudo service nginx restart
```

3. 创建Logtail配置并应用到该服务器所在的机器组。
   配置过程请参见配置Logtail插件采集syslog。
4. 检验Logtail配置是否生效。
   在shell中执行命令 `curl http://127.0.0.1/test.html` 生成一条访问日志。如果采集配置已生效，可以在日志服务控制台的查询页面查看到日志信息。

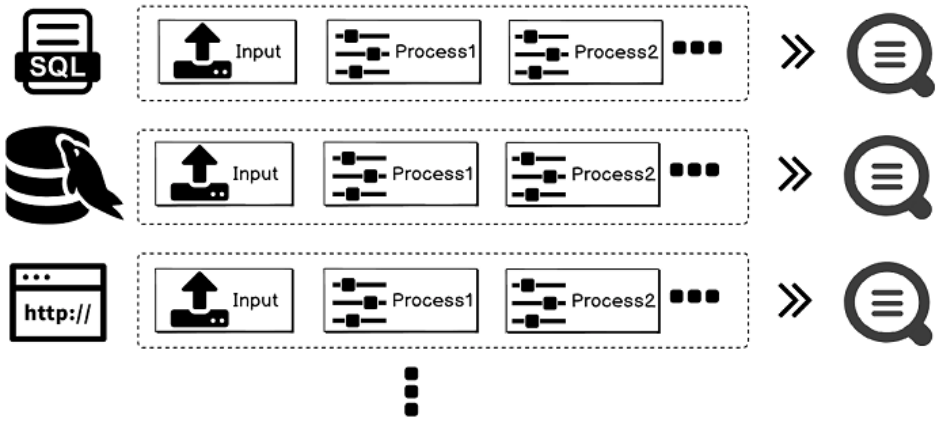
3.1.5.4. 处理采集数据

Logtail提供数据处理插件，您可对数据源添加一个或多个处理配置，Logtail会根据处理配置顺序逐一执行。本文列举了Logtail处理配置及相关说明。

实现原理

数据处理配置的实现原理如下图所示。

图 1. 实现原理



使用说明

为采集数据配置处理方式，处理配置的key为 `processors` ，value为json object的数组，数组内每个object代表一个处理方式。

单一处理方式包含两个字段： `type` 、 `detail` ，其中 `type` 为该处理方式的类型， `detail` 为该处理方式的详细配置。

```
"processors" : [
        {
            "type" : "processor_anchor",
            "detail" : {
                ...
            }
        },
        {
            "type" : "processor_regex",
            "detail" : {
                ...
            }
        }
    ]
```

处理方式

目前支持的处理方式如下：

- 正则提取
- 标定提取
- 单字分隔符
- 多字符分隔符
- GeoIP转换
- 正则过滤
- 字段添加
- 字段丢弃
- 日志时间提取（Golang）
- JSON 处理
- 字段打包（JSON）
- 字段重命名
- 日志时间提取（Strptime）

您也可以根据以上处理方式，为您的输入源定制组合配置。

正则提取

该方式通过对指定字段进行正则表达式匹配来提取其中匹配的字段。

该插件的类型为 `processor_regex` 。 表 1. 参数说明

参数	类型	必选或可选	参数说明
SourceKey	string	必选	原始字段名，即需要进行正则提取的字段。
Regex	string	必选	用于匹配的正则表达式，需要提取的字段使用 `()` 标注。
Keys	string 数组	必选	需要提取的字段名，例如[“key1”, “key2” ...]。
NoKeyError	bool	可选	无匹配时是否报错，默认为false表示否。
NoMatchError	bool	可选	正则不匹配时是否报错，默认为false表示否。
KeepSource	bool	可选	是否保留原始字段，默认为false表示否。
FullMatch	bool	必选	默认为true表示只有字段完全匹配 `Regex` 时才会进行提取，为false时表示部分字段匹配也会进行提取。

提取Access日志，详细配置示例如下：

- 输入

```
"content" : "10.200.**.** - - [10/Aug/2017:14:57:51 +0800] \"POST /PutData?
Category=YunOsAccountOpLog&AccessKeyId=
<yourAccessKeyId>&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20GMT&Topic=raw&Signature=<yourSignatur
e> HTTP/1.1\" 0.024 18204 200 37 \"-\" \"aliyun-sdk-java"
```

- 配置详情

```
{
    "type" : "processor_regex",
    "detail" : {"SourceKey" : "content",
         "Regex" : "([\\d\\.]+) \\S+ \\S+ \\[(\\S+) \\S+\\] \"(\\w+) ([^\\\"]*)\" ([\\d\\.]+) (\\d+) (
\\d+) (\\d+|-) \"([^\\\"]*)\" \"([^\\\"]*)\" (\\d+)",
         "Keys"    : ["ip", "time", "method", "url", "request_time", "request_length", "status", "lengt
h", "ref_url", "browser"],
         "NoKeyError" : true,
         "NoMatchError" : true,
         "KeepSource" : false
    }
}
```

- 处理后结果

```
"ip" : "10.200.**.**"
"time" : "10/Aug/2017:14:57:51"
"method" : "POST"
"url" : "/PutData?Category=YunOsAccountOpLog&AccessKeyId=
<yourAccessKeyId>&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20GMT&Topic=raw&Signature=<yourSignatur
e>"
"request_time" : "0.024"
"request_length" : "18204"
"status" : "200"
"length" : "27"
"ref_url" : "-"
"browser" : "aliyun-sdk-java"
```

标定提取

该插件通过标定指定字段 `起始` 和 `结束` 关键字进行提取，标定后的字符串支持直接提取和JSON展开。

该插件的类型为 `processor_anchor` 。 表 2. 参数说明

参数	类型	必选或可选	参数说明
SourceKey	string	必选	原始字段名，即需要进行提取的字段。
Anchors	Anchor 数组	必选	标定项列表，具体参见下述表格。
NoAnchorError	bool	可选	查找不到关键字是否报错，默认为false表示否。
NoKeyError	bool	可选	无匹配时是否报错，默认为false表示否。
KeepSource	bool	可选	是否保留原始字段，默认为false表示否。

表 3. Anchor类型说明

参数	类型	必选或可选	参数说明
Start	string	必选	起始关键字，若为空则代表匹配字符串开头。
Stop	string	必选	结束关键字，若为空则代表匹配字符串结尾。
FieldName	string	必选	提取的字段名。
FieldType	string	必选	提取字段类型，支持 `string` 、 `json` 两种类型。
ExpondJson	bool	可选	是否进行JSON展开，默认为false表示否。 当 `FieldType` 为 `json` 时生效。
ExpondConnecter	string	可选	JSON展开的连接符，默认为 `_` 。
MaxExpondDepth	int	可选	JSON展开最大深度，默认为 `0` 表示无限制。

如下配置示例对某混合类型输入的处理结果如下：

- 输入

```
"content" : "time:2017.09.12 20:55:36\tjson:{\"key1\" : \"xx\", \"key2\": false, \"key3\":123.456,
\"key4\" : { \"inner1\" : 1, \"inner2\" : false}}"
```

- 配置详情

```
{
   "type" : "processor_anchor",
   "detail" : {"SourceKey" : "content",
      "Anchors" : [
          {
              "Start" : "time",
              "Stop" : "\t",
              "FieldName" : "time",
              "FieldType" : "string",
              "ExpondJson" : false
          },
          {
              "Start" : "json:",
              "Stop" : "",
              "FieldName" : "val",
              "FieldType" : "json",
              "ExpondJson" : true
          }
      ]
  }
}
```

- 处理后结果

```
"time" : "2017.09.12 20:55:36"
"val_key1" : "xx"
"val_key2" : "false"
"val_key3" : "123.456"
"value_key4_inner1" : "1"
"value_key4_inner2" : "false"
```

单字符分隔符

该插件通过指定 `分隔符` 对字段进行分割，可指定Quote进行分隔符屏蔽。

该插件的类型为 `processor_split_char` 。 表 4. 参数说明

参数	类型	必选或可选	参数说明
SourceKey	string	必选	原始字段名，即需要进行提取的字段。
SplitSep	string	必选	分隔符，必须为单字符，可以设置不可见字符，例如 `\u0001` 。
SplitKeys	string 数组	必选	切分后的字段名，例如[“key1”, “key2”...]。
QuoteFlag	bool	可选	是否使用 `Quote` ，默认为false表示否。
Quote	string	可选	必须为单字符， `QuoteFlag` 为true时生效，可以设置为不可见字符，例如 `\u0001` 。
NoKeyError	bool	可选	无匹配时是否报错，默认为false表示否。
NoMatchError	bool	可选	单字符不匹配时是否则报错，默认为false表示否。
KeepSource	bool	可选	是否保留原始字段，默认为false表示否。

对分隔符数据采用单字符分隔符处理方式，配置示例详情及处理结果如下：

- 输入

```
"content" : "10.**.**.**|10/Aug/2017:14:57:51 +0800|POST|PutData?
Category=YunOsAccountOpLog&AccessKeyId=
<yourAccessKeyId>&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20GMT&Topic=raw&Signature=<yourSignatur
e>|0.024|18204|200|37|-|
aliyun-sdk-java"
```

- 配置详情

```
{
   "type" : "processor_split_char",
   "detail" : {"SourceKey" : "content",
      "SplitSep" : "|",
      "SplitKeys" : ["ip", "time", "method", "url", "request_time", "request_length", "status", "lengt
h", "ref_url", "browser"]
   }
}
```

- 处理后结果

```
"ip" : "10.**.**.**"
"time" : "10/Aug/2017:14:57:51 +0800"
"method" : "POST"
"url" : "/PutData?Category=YunOsAccountOpLog&AccessKeyId=
<yourAccessKeyId>&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20GMT&Topic=raw&Signature=<yourSignatur
e>"
"request_time" : "0.024"
"request_length" : "18204"
"status" : "200"
"length" : "27"
"ref_url" : "-"
"browser" : "aliyun-sdk-java"
```

多字符分隔符

和单字符分隔符类似，多字符分隔符不支持Quote，完全按照分隔符拆分日志。

该插件的类型为 `processor_split_string` 。 表 5. 参数说明

参数	类型	必选或可选	参数说明
SourceKey	string	必选	原始字段名，即需要进行提取的字段。
SplitSep	string	必选	分隔符，可以设置不可见字符，例如 `\u0001\u0002` 。
SplitKeys	string 数组	必选	切分后的字段名，例如[“key1”, “key2”...]。
PreserveOthers	bool	可选	若分割的字段大于 `SplitKeys` 长度时是否保留超出部分，默认为false表示否。
ExpandOthers	bool	可选	是否会继续解析超出部分，默认为false表示否。
ExpandKeyPrefix	string	可选	超出部分的命名前缀，例如配置 `expand_` ，则key为 `expand_1` 、 `expand_2` 。
NoKeyError	bool	可选	无匹配时是否报错，默认为false表示否。
NoMatchError	bool	可选	多字符不匹配时是否报错，默认为false表示否。
KeepSource	bool	可选	是否保留原始字段，默认为false表示否。

对分隔符数据输入采用多字符分隔符方式处理，配置示例详情及处理结果如下：

- 输入

```
"content" : "10.**.**.**|#|10/Aug/2017:14:57:51 +0800|#|POST|#|PutData?
Category=YunOsAccountOpLog&AccessKeyId=
<yourAccessKeyId>&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20GMT&Topic=raw&Signature=<yourSignatur
e>|#|0.024|#|18204|#|200|#|37|#|-|#|
aliyun-sdk-java"
```

- 配置详情

```
{
   "type" : "processor_split_string",
   "detail" : {"SourceKey" : "content",
      "SplitSep" : "|#|",
      "SplitKeys" : ["ip", "time", "method", "url", "request_time", "request_length", "status"],
      "PreserveOthers" : true,
      "ExpandOthers" : true,
      "ExpandKeyPrefix" : "expand_"
  }
}
```

- 处理后结果

```
"ip" : "10.**.**.**"
"time" : "10/Aug/2017:14:57:51 +0800"
"method" : "POST"
"url" : "/PutData?Category=YunOsAccountOpLog&AccessKeyId=
<yourAccessKeyId>&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20GMT&Topic=raw&Signature=<yourSignatur
e>"
"request_time" : "0.024"
"request_length" : "18204"
"status" : "200"
"expand_1" : "27"
"expand_2" : "-"
"expand_3" : "aliyun-sdk-java"
```

GeoIP转换

GeoIP转换对数据中的IP进行地理位置转换，能够将IP转换成：国家、省份、城市、经纬度。

> **说明** Logtail安装包本身并不带有GeoIP的数据库，需要您手动下载到本地并配置，建议下载精确到 `City` 的数据库。

该插件的类型为 `processor_geoip` 。 表 6. 参数说明

参数	类型	必选或可选	参数说明
SourceKey	string	必选	原始字段名，即需要进行IP转换的字段。
DBPath	string	必选	GeoIP数据库的全路径，数据库格式为 `mmdb` ，例如 `/user/data/GeoLite2-City_20180102/GeoLite2-City.mmdb` 。
NoKeyError	bool	可选	无匹配的key时是否报错。默认为false表示不报错。
NoMatchError	bool	可选	若IP地址无效或数据库中未匹配到该IP时是否报错，默认为false表示不会上报错误。
KeepSource	bool	可选	是否保留原始字段，默认为true表示保留。
Language	string	可选	语言属性，默认为 `zh-CN` ，需确保您的GeoIP数据库中包含相应的语言。

对输入的IP采用GeoIP转换成地理位置信息，配置示例详情及处理结果如下：

- 输入

```
"source_ip" : "**.**.**.**"
```

下载GeoIP数据库到安装Logtail的主机，可使用MaxMind GeoLite2中的City数据库。

> **说明** 请检查数据库格式为mmdb类型。

- 配置详情

```
{
   "type": "processor_geoip",
    "detail": {
         "SourceKey": "ip",
         "NoKeyError": true,
         "NoMatchError": true,
         "KeepSource": true,
         "DBPath" : "/user/local/data/GeoLite2-City_20180102/GeoLite2-City.mmdb"
    }
}
```

- 配置后结果

```
"source_ip_city_" : "**.**.**.**"
"source_ip_province_" : "浙江省"
"source_ip_city_" : "杭州"
"source_ip_province_code_" : "ZJ"
"source_ip_country_code_" : "CN"
"source_ip_longitude_" : "120.********"
"source_ip_latitude_" : "30.********"
```

正则过滤

该插件通过对字段进行正则表达式匹配对日志进行过滤，可组合使用 `Include` 和 `Exclude` 两种方式。

该插件的类型为 `processor_filter_regex` 。

表 7. 参数说明

参数	类型	必选或可选	参数说明
Include	key:string value:string 的map	可选	key为日志字段，value为该字段匹配的正则表达式，若指定字段符合该表达式，则该条日志被收集。
Exclude	key:string value:string 的map	可选	key为日志字段，value为该字段匹配的正则表达式，若指定字段符合该表达式，则该条日志不被收集。

> **说明** 一条日志只有完全被 `Include` 中的参数匹配，且不被 `Exclude` 中的任一参数匹配时才会被采集，否则直接丢弃。

对输入日志采正则过滤方式处理，配置示例详情及处理结果如下：

- 输入
  - 日志1

    ```
    "ip" : "10.**.**.**"
    "method" : "POST"
    ...
    "browser" : "aliyun-sdk-java"
    ```

  - 日志2

    ```
    "ip" : "10.**.**.**"
    "method" : "POST"
    ...
    "browser" : "chrome"
    ```

  - 日志3

    ```
    "ip" : "192.168.*.*"
    "method" : "POST"
    ...
    "browser" : "ali-sls-ilogtail"
    ```

- 配置详情

```
{
   "type" : "processor_filter_regex",
    "detail" : {
         "Include" : {
            "ip" : "10\\..*",
            "method" : "POST"
         },
         "Exclude" : {
            "browser" : "aliyun.*"
         }
    }
}
```

- 处理后结果

日志	是否匹配	原因
日志1	不匹配	browser匹配上了Exclude。
日志2	匹配	-
日志3	不匹配	`Include` 中 `ip` 字段不以 `10` 开头，不匹配。

字段添加

通过该插件可以为日志数据添加指定的字段（支持添加多个字段）。

该插件的类型为 `processor_add_fields` 。

> **说明** 该插件支持Logtail 0.16.28及以上版本。

表 8. 参数说明

参数	类型	必选或可选	参数说明
Fields	map	可选	键值对，指定要添加的一个或者多个键值。
IgnoreIfExist	bool	可选	当key相同时是否忽略。默认为false表示不忽略。

为日志数据增加指定字段，配置示例及处理结果如下：

- 输入

```
"aaa1":"value1"
```

- 配置详情

```
{
  "processors":[
    {
      "type":"processor_add_fields",
      "detail": {
        "Fields": {
          "aaa2": "value2",
          "aaa3": "value3"
        }
      }
    }
  ]
}
```

- 处理结果

```
"aaa1":"value1"
"aaa2":"value2"
"aaa3":"value3"
```

字段丢弃

通过该插件对日志数据中指定的字段进行丢弃。

该插件的类型为 `processor_drop` 。

> **说明** 该插件支持Logtail 0.16.28及以上版本。

表 9. 参数说明

参数	类型	必选或可选	参数说明
DropKeys	array	可选	字符串数组，指定要从日志中删除的一个或者多个字段。

从日志中删除 `aaa1` 和 `aaa2` 字段，配置详情及处理结果如下：

- 输入

```
"aaa1":"value1"
"aaa2":"value2"
"aaa3":"value3"
```

- 配置详情

```
{
  "processors":[
    {
      "type":"processor_drop",
      "detail": {
        "DropKeys": ["aaa1","aaa2"]
      }
    }
  ]
}
```

- 处理结果

```
"aaa3":"value3"
```

日志时间提取（Golang）

该插件可以从指定字段中提取日志时间，并将其转换为其他字段。

该插件的类型为 `processor_gotime` 。

> **说明** 该插件支持Logtail 0.16.28及以上版本。

表 10. 参数说明

参数	类型	必选或可选	参数说明
SourceKey	string	必选	提取日志时间的源Key，为空不生效。
SourceFormat	string	必选	将源key解析为日志时间的go时间格式。
SourceLocation	int	必选	源时区，为空表示本机时区。
DestKey	string	必选	目标Key，为空不生效。
DestFormat	string	必选	目标key解析的go时间格式。
DestLocation	int	可选	目标时区，为空表示本机时区。
SetTime	bool	可选	是否设置到时间字段，默认为true。
KeepSource	bool	可选	是否保留源字段，默认为true表示保留。
NoKeyError	bool	可选	无匹配的key是否报错，默认为true表示报错。
AlarmIfFail	bool	可选	提取失败是否告警，默认为true表示告警。

以 `2006-01-02 15:04:05` （东八区）解析字段 `s_key` 的值作为日志时间，并以格式 `2006/01/02 15:04:05` （东九区）将日志时间转换到新字段 `d_key` 中。配置详情及处理结果如下所示：

- 输入

```
"s_key":"2019-07-05 19:28:01"
```

- 配置详情

```
{
  "processors":[
    {
      "type":"processor_gotime",
      "detail": {
        "SourceKey": "s_key",
        "SourceFormat":"2006-01-02 15:04:05",
        "SourceLocation":8,
        "DestKey":"d_key",
        "DestFormat":"2006/01/02 15:04:05",
        "DestLocation":9,
        "SetTime": true,
        "KeepSource": true,
        "NoKeyError": true,
        "AlarmIfFail": true
      }
    }
  ]
}
```

- **处理结果**

```
"s_key":"2019-07-05 19:28:01"
"d_key":"2019/07/05 20:28:01"
```

JSON 处理

通过该插件可以对指定字段进行JSON展开。

该插件的类型为 `processor_json` **。**

> **说明** 该插件支持Logtail 0.16.28及以上版本。

表 11. 参数说明

参数	类型	必选或可选	参数说明
SourceKey	string	必选	要进行JSON展开的源key。
NoKeyError	bool	可选	无匹配key时是否报错，默认为true表示报错。
ExpandDepth	int	可选	JSON展开的深度。非负整数，默认为0表示不限制，1表示当前层级，以此类推。
ExpandConnector	string	可选	JSON展开时的连接符，默认为 `_` ，也可以为空。
Prefix	string	可选	JSON展开时对Key附加的前缀，默认为空。
KeepSource	bool	可选	是否保留源字段，默认为true表示保留。
UseSourceKeyAsPrefix	bool	可选	是否将源key作为所有JSON展开key的前缀，默认为false表示否。

对字段 `s_key` **进行JSON展开，配置详情及处理结果如下：**

- **输入**

```
"s_key":"{\"k1\":{\"k2\":{\"k3\":{\"k4\":{\"k51\":\"51\",\"k52\":\"52\"},\"k41\":\"41\"}}})"
```

- **配置详情**

```
{
  "processors":[
    {
      "type":"processor_json",
      "detail": {
        "SourceKey": "s_key",
        "NoKeyError":true,
        "ExpandDepth":0,
        "ExpandConnector":"-",
        "Prefix":"j",
        "KeepSource": false,
        "UseSourceKeyAsPrefix": true
      }
    }
  ]
}
```

- 处理结果

```
"s_key":"{\"k1\":{\"k2\":{\"k3\":{\"k4\":{\"k51\":\"51\",\"k52\":\"52\"},\"k41\":\"41\"}}})"
"js_key-k1-k2-k3-k4-k51":"51"
"js_key-k1-k2-k3-k4-k52":"52"
"js_key-k1-k2-k3-k41":"41"
```

字段打包（JSON）

通过该插件可以将指定字段（支持多个）打包成JSON格式的字段。

该插件的类型为 `processor_packjson` 。

> **说明** 该插件支持Logtail 0.16.28及以上版本。

表 12. 参数说明

参数	类型	必选或可选	参数说明
SourceKeys	array	必选	字符串数组，需要打包的key。
DestKey	string	可选	打包后的JSON格式的key。
KeepSource	bool	可选	是否保留源字段，默认为true表示保留。
AlarmIfIncomplete	bool	可选	是否在不存在任何源字段时告警，默认为true表示告警。

将指定的 `a` 、 `b` 两个字段打包成JSON字段 `d_key` ，配置详情及处理结果如下：

- 输入

```
"a":"1"
"b":"2"
```

- 配置详情

```
{
  "processors":[
    {
      "type":"processor_packjson",
      "detail": {
        "SourceKeys": ["a","b"],
        "DestKey":"d_key",
        "KeepSource":true,
        "AlarmIfEmpty":true
      }
    }
  ]
}
```

- 处理结果

```
"a":"1"
"b":"2"
"d_key":"{\"a\":\"1\",\"b\":\"2\"}"
```

字段重命名

通过该插件可以对指定字段（支持多个）进行重命名。

该插件的类型为 `processor_rename` **。**

> **说明** 该插件支持Logtail 0.16.28及以上版本。

表 13. 参数说明

参数	类型	必选或可选	参数说明
NoKeyError	bool	可选	无匹配的key是否报错，默认false表示不报错。
SourceKeys	array	必选	字符串数组，指定要进行重命名的源key。
DestKeys	array	必选	字符串数组，对源key进行重命名后的目标key。

将字段 `aaa1` **和** `aaa2` **重命名为** `bbb1` **和** `bbb2` **，配置详情及处理结果如下：**

- 输入

```
"aaa1":"value1"
"aaa2":"value2"
"aaa3":"value3"
```

- 配置详情

```
{
  "processors":[
    {
      "type":"processor_rename",
      "detail": {
        "SourceKeys": ["aaa1","aaa2"],
        "DestKeys": ["bbb1","bbb2"],
        "NoKeyError": true
      }
    }
  ]
}
```

- 处理结果

```
"bbb1":"value1"
"bbb2":"value2"
"aaa3":"value3"
```

日志时间提取（Strptime）

通过该插件可以从指定字段中提取日志时间，时间格式为Linux strptime。

该插件的类型为 `processor_strptime` 。

> **说明** 该插件支持Logtail 0.16.28及以上版本。

表 14. 参数说明

参数	类型	必选或可选	参数说明
SourceKey	string	必选	提取日志时间的源Key，为空则不生效。
Format	string	必选	解析源Key所使用的时间格式。
AdjustUTCOffset	bool	可选	是否对时间时区进行调整，默认为false表示不调整。
UTCOffset	int	可选	用于调整的时区偏移秒数，如14400表示东八区。
AlarmIfFail	bool	可选	提取失败时是否告警，默认为true表示进行告警。
KeepSource	bool	可选	是否保留源字段，默认为true表示保留。

将 `log_time` 的值解析为 `%Y/%m/%d %H:%M:%S` 格式的日志时间，时区使用机器时区。配置详情及处理结果如下：

- 示例1：假设时区为东八区。
  - 输入

    ```
    "log_time":"2016/01/02 12:59:59"
    ```

  - 配置详情

    ```
    {
      "processors":[
        {
          "type":"processor_strptime",
          "detail": {
            "SourceKey": "log_time",
            "Format": "%Y/%m/%d %H:%M:%S"
          }
        }
      ]
    }
    ```

  - 处理结果

    ```
    "log_time":"2016/01/02 12:59:59"
    Log.Time = 1451710799
    ```

- 示例2：假设时区为东七区。
  - 输入

    ```
    "log_time":"2016/01/02 12:59:59"
    ```

- 配置详情

```
{
  "processors":[
    {
      "type":"processor_strptime",
      "detail": {
        "SourceKey": "log_time",
        "Format": "%Y/%m/%d %H:%M:%S",
        "AdjustUTCOffset": true,
        "UTCOffset": 25200
      }
    }
  ]
}
```

- 处理结果

```
"log_time":"2016/01/02 12:59:59"
Log.Time = 1451714399
```

组合配置

各个处理配置可以组合搭配使用。您可以参考下述配置对日志先进行分隔符切分，再对切分后的 `detail` 进行标定提取。

- 输入

```
"content" :
"ACCESS|QAS|11.**.**.**|1508729889935|52460dbed4d540b88a973cf5452b1447|1238|appKey=ba,env=pub,requestTim
508729889913,latency=22ms,
request={appKey:ba,optional:{\\domains\\:\\daily\\,\\version\\:\\v2\\},rawQuery:{\\query\\:\\去乐山的路
线\\,\\domain\\:\\导航\\,\\intent\\:\\navigate\\,\\slots\\:\\to_geo:level3=乐山\\,\\location\\:\\北京\\}
,
requestId:52460dbed4d540b88a973cf5452b1447},
response={answers:[],status:SUCCESS}|"
```

- 配置详情

```
"processors" : [
      {
          "type" : "processor_split_char",
          "detail" : {"SourceKey" : "content",
              "SplitSep" : "|",
              "SplitKeys" : ["method", "type", "ip", "time", "req_id", "size", "detail"]
          }
      },
      {
          "type" : "processor_anchor",
          "detail" : "SourceKey" : "detail",
              "Anchors" : [
                  {
                          "Start" : "appKey=",
                      "Stop" : ",env=",
                      "FieldName" : "appKey",
                      "FieldType" : "string"
                  },
                  {
                      "Start" : ",env",
                      "Stop" : ",requestTime=",
                      "FieldName" : "env",
                      "FieldType" : "string"
                  },
                  {
                      "Start" : ",requestTime=",
                      "Stop" : ",latency",
                      "FieldName" : "requestTime",
                      "FieldType" : "string"
                  },
                  {
                      "Start" : ",latency=",
                      "Stop" : ",request=",
                      "FieldName" : "latency",
                      "FieldType" : "string"
                  },
                  {
                      "Start" : ",request=",
                      "Stop" : ",response=",
                      "FieldName" : "request",
                      "FieldType" : "string"
                  },
                  {
                      "Start" : ",response=",
                      "Stop" : "",
                      "FieldName" : "response",
                      "FieldType" : "json"
                  }
              ]
          }
      }
  ]
```

- 处理后结果

```
"method" : "ACCESS"
"type" : "QAS"
"ip" : "**.**.**.**"
"time" : "1508729889935"
"req_id" : "52460dbed4d540b88a973cf5452b1447"
"size" : "1238"
"appKey" : "ba"
"env" : "pub"
"requestTime" : "1508729889913"
"latency" : "22ms"
"request" : "{appKey:nui-banma,optional:{\\domains\\:\\daily-faq\\,\\version\\:\\v2\\},rawQuery:
{\\query\\:\\\345\216\273\344\271\220\345\261\261\347\232\204\350\267\257\347\272\277\\,\\domain\\:\\\34
57\274\350\210\252\\,\\intent\\:\\navigate\\,\\slots\\:\\to_geo:level3=\344\271\220\345\261\261\\,\\loca
n\\:\\\345\214\227\344\272\254\\},requestId:52460dbed4d540b88a973cf5452b1447}"
"response_answers" : "[]"
"response_status" : "SUCCESS"
```

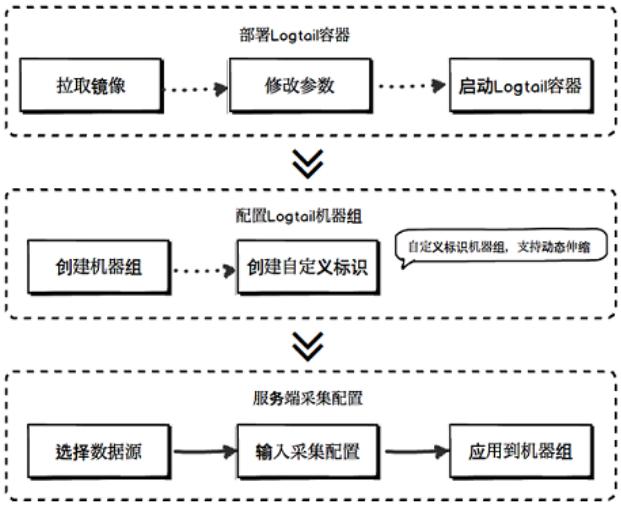

3.1.6. 容器日志采集

3.1.6.1. 标准Docker日志采集流程

Logtail支持采集标准Docker日志，并附加容器的相关元数据信息一起上传到日志服务。

配置流程

图 1. 配置流程



1. 部署Logtail容器。
2. 配置机器组。

   日志服务控制台创建自定义标识机器组，后续该容器集群伸缩无需额外运维。
3. 创建采集配置。

   在日志服务控制台创建采集配置，所有采集均为服务端配置，无需本地配置。

部署Logtail容器

1. 拉取Logtail镜像。

   ```
   docker pull registry.cn-hangzhou.aliyuncs.com/log-service/logtail
   ```
2. 启动Logtail容器。
   替换启动模板中的3个参数：`${your_region_name}`、`${your_aliyun_user_id}`和`${your_machine_group_user_defined_id}`。

```
docker run -d -v /:/logtail_host:ro -v /var/run:/var/run --env
ALIYUN_LOGTAIL_CONFIG=/etc/ilogtail/conf/${your_region_name}/ilogtail_config.json
--env ALIYUN_LOGTAIL_USER_ID=${your_aliyun_user_id} --env
 ALIYUN_LOGTAIL_USER_DEFINED_ID=${your_machine_group_user_defined_id} registry.cn-
hangzhou.aliyuncs.com/log-service/logtail
```

> **说明** 请在配置参数前执行以下任意一种配置，否则删除其他container时可能出现错误 `container text file busy` 。
>
> - Centos 7.4及以上版本设置fs.may_detach_mounts=1，相关说明请参见Bug 1468249、Bug 1441737和issue 34538。
> - 为Logtail授予 `privileged` 权限，启动参数中添加 `--privileged` 。详细内容请参见docker run命令。

参数	参数说明
`${your_region_name}`	日志服务Project所在Region。请参见Project概览查看。
`${your_aliyun_user_id}`	用户标识，请替换为您的阿里云主账号用户ID。主账号用户ID为字符串形式，如何查看ID请参见配置用户标识中的步骤一。
`${your_machine_group_user_defined_id}`	您集群的机器组自定义标识。如您尚未开启自定义标识，请参见创建用户自定义标识机器组的步骤一，开启userdefined-id。

替换参数后的样例如下：

```
docker run -d -v /:/logtail_host:ro -v /var/run:/var/run
--env ALIYUN_LOGTAIL_CONFIG=/etc/ilogtail/conf/cn_hangzhou/ilogtail_config.json --env
 ALIYUN_LOGTAIL_USER_ID=1654218******--env ALIYUN_LOGTAIL_USER_DEFINED_ID=log-docker-demo registry.cn-
hangzhou.aliyuncs.com/log-service/logtail
```

> **说明**
> 您可以自定义配置Logtail容器的启动参数，只需保证以下前提条件。
>
> - 启动时，必须具备3个环境变量： `ALIYUN_LOGTAIL_USER_DEFINED_ID` 、 `ALIYUN_LOGTAIL_USER_ID` 、 `ALIYUN_LOGTAIL_CONFIG` 。
> - 必须将/var/run挂载到Logtail容器的/var/run目录。
> - 如果您需要采集容器标准输出、容器或宿主机文件，需要将根目录挂载到Logtail容器的 `/logtail_host` 目录。
> - 如果Logtail日志/usr/local/ilogtail/ilogtail.LOG中出现 `The parameter is invalid : uuid=none` 的错误日志，请在宿主机上创建一个product_uuid文件，在其中输入任意合法UUID（例如 `169E98C9-ABC0-4A92-B1D2-AA6239C0D261` ），并把该文件挂载到Logtail容器的/sys/class/dmi/id/product_uuid路径上。

配置机器组

1. 登录日志服务控制台。
2. 单击目标Project名称。
3. 单击左侧导航栏的**机器组**图标展开机器组列表。
4. 单击机器组后的图标，选择**创建机器组**。
   您也可以在数据接入向导中创建机器组。
5. 选择**用户自定义标识**，将您上一步配置的 `ALIYUN_LOGTAIL_USER_DEFINED_ID` 填入**用户自定义标识**内容框中。

配置完成一分钟后，在**机器组**列表页面单击机器组名称，即可在**机器组配置**页面看到已经部署Logtail容器的心跳状态。具体请参见查看机器组状态。

创建采集配置

请根据您的需求在控制台创建Logtail采集配置。

- Docker文件请参见容器文本日志。
- Docker标准输出请参见容器标准输出。
- 宿主机文本文件。

默认宿主机根目录挂载到Logtail容器的 `/logtail_host` 目录，配置路径时，您需要加上此前缀。例如需要采集宿主机上 `/home/logs/app_log/` 目录下的数据，配置页面中日志路径设置为 `/logtail_host/home/logs/app_log/` 。

其他操作

- 查看Logtail容器运行状态。

  您可以执行命令 `docker exec ${logtail_container_id} /etc/init.d/ilogtaild status` 查看Logtail运行状态。

- 查看Logtail的版本号信息、IP、启动时间等。

  您可以执行命令 `docker exec ${logtail_container_id} cat /usr/local/ilogtail/app_info.json` 查看Logtail相关信息。

- 查看Logtail的运行日志。

  Logtail运行日志保存在 `/usr/local/ilogtail/` 目录下，文件名为 `ilogtail.LOG` ，轮转文件会压缩存储为 `ilogtail.LOG.x.gz` 。

  示例如下：

```
[root@iZbp17enxc2us3624wexh2Z ilogtail]# docker exec a287de895e40 tail -n 5
/usr/local/ilogtail/ilogtail.LOG
[2018-02-06 08:13:35.721864]    [INFO]    [8]    [build/release64/sls/ilogtail/LogtailPlugin.cpp:104]
logtail plugin Resume:start
[2018-02-06 08:13:35.722135]    [INFO]    [8]    [build/release64/sls/ilogtail/LogtailPlugin.cpp:106]
logtail plugin Resume:success
[2018-02-06 08:13:35.722149]    [INFO]    [8]
[build/release64/sls/ilogtail/EventDispatcher.cpp:369]    start add existed check point events, size:0
[2018-02-06 08:13:35.722155]    [INFO]    [8]
[build/release64/sls/ilogtail/EventDispatcher.cpp:511]    add existed check point events, size:0    ca
che size:0    event size:0    success count:0
[2018-02-06 08:13:39.725417]    [INFO]    [8]    [build/release64/sls/ilogtail/ConfigManager.cpp:3776]
check container path update flag:0    size:1
```

  容器stdout并不具备参考意义，请忽略以下stdout输出。

```
start umount useless mount points, /shm$|/merged$|/mqueue$
umount:
/logtail_host/var/lib/docker/overlay2/3fd0043af174cb0273c3c7869500fbe2bdb95d13b1e110172ef57fe840c82155/m
ed: must be superuser to unmount
umount:
/logtail_host/var/lib/docker/overlay2/d5b10aa19399992755de1f85d25009528daa749c1bf8c16edff44beab6e69718/m
ed: must be superuser to unmount
umount:
/logtail_host/var/lib/docker/overlay2/5c3125daddacedec29df72ad0c52fac800cd56c6e880dc4e8a640b1e16c22dbe/m
ed: must be superuser to unmount
......
xargs: umount: exited with status 255; aborting
umount done
start logtail
ilogtail is running
logtail status:
ilogtail is running
```

- 重启Logtail。

  请参考以下示例重启Logtail。

```
[root@iZbp17enxc2us3624wexh2Z ilogtail]# docker exec a287de895e40 /etc/init.d/ilogtaild stop
kill process Name: ilogtail pid: 7
kill process Name: ilogtail pid: 8
stop success
[root@iZbp17enxc2us3624wexh2Z ilogtail]# docker exec a287de895e40 /etc/init.d/ilogtaild start
ilogtail is running
```

3.1.6.2. Kubernetes日志采集流程

本文主要介绍如何安装并使用Logtail采集Kubernetes集群日志。

配置流程

Kubernetes集群日志采集配置流程如下所示。

1. 安装alibaba-log-controller Helm包。
2. 通过控制台进行采集配置管理。

步骤1 安装Logtail

- 安装阿里云容器服务Kubernetes集群
  若您未安装日志服务组件，请先手动安装Logtail日志服务组件。
  i. 通过cloudshell连接Kubernetes集群。
  ii. 执行如下命令在cloudshell中获取您的主账号aliuid。

```
echo $ALIBABA_CLOUD_ACCOUNT_ID
```

  iii. 替换 `${your_k8s_cluster_id}` 、 `${your_ali_uid}` 、 `${your_k8s_cluster_region_id}` 参数后执行以下安装命令。

```
wget https://acs-logging.oss-cn-hangzhou.aliyuncs.com/alicloud-k8s-log-installer.sh -O alicloud-k8s-log-installer.sh; chmod 744 ./alicloud-k8s-log-installer.sh; ./alicloud-k8s-log-installer.sh --cluster-id ${your_k8s_cluster_id} --ali-uid ${your_ali_uid} --region-id ${your_k8s_cluster_region_id}
```

- 安装自建Kubernetes集群

> **说明**
> - Kubernetes1.8及以上版本。
> - 已经安装2.6.4及以上版本的Helm命令行工具。

  i. 在日志服务中创建一个名称以 `k8s-log-custom-` 开头的Project。
  ii. 替换下述命令中的参数并执行该命令。

```
wget http://logtail-release-cn-hangzhou.oss-cn-hangzhou.aliyuncs.com/kubernetes/alicloud-log-k8s-custom-install.sh; chmod 744 ./alicloud-log-k8s-custom-install.sh; sh ./alicloud-log-k8s-custom-install.sh {your-project-suffix} {region-id} {aliuid} {access-key-id} {access-key-secret}
```

  命令中各参数及其说明如下：

参数	说明
{your-project-suffix}	您创建的Project名称 `k8s-log-custom-` 之后部分。例如创建的Project为 `k8s-log-custom-xxxx` ，则此处填写 `xxxx` 。
{regionId}	您的Project所在区域的Region Id。请在Project概览中查看。
{aliuid}	用户标识（AliUid），请替换为您的阿里云主账号ID。 **说明** 阿里云主账号ID为字符串形式，如何查看主账号ID请参见配置用户标识。
{access-key-id}	您的账号AccessKey id。
{access-key-secret}	您的账号AccessKey secret。

  安装好之后，日志服务会自动在该Project下创建机器组，机器组名为 `k8s-group-${your_k8s_cluster_id}` 。

> **说明**
> - Project下会自动创建名为 `config-operation-log` 的Logstore，请勿删除该Logstore。
> - 自建Kubernetes安装时，默认为Logtail授予 `privileged` 权限，主要为避免删除其他Pod时可能出现错误 `container text file busy` 。相关说明请参见Bug 1468249、Bug 1441737和 issue 34538。

  执行成功后会输出以下内容：

```
[root@iZbp1dsxxxxxqfbiaZ ~]#  wget http://logtail-release-cn-hangzhou.oss-cn-
hangzhou.aliyuncs.com/kubernetes/alicloud-log-k8s-custom-install.sh; chmod 744 ./alicloud-log-k8s-
custom-install.sh; sh ./alicloud-log-k8s-custom-install.sh xxxx cn-hangzhou 165xxxxxxxx050
LTAxxxxxxxxxx AIxxxxxxxxxxxxxxxxxxxxxxxxxxxxxe
....
....
....
NAME:   alibaba-log-controller
LAST DEPLOYED: Fri May 18 16:52:38 2018
NAMESPACE: default
STATUS: DEPLOYED
RESOURCES:
==> v1beta1/ClusterRoleBinding
NAME                    AGE
alibaba-log-controller  0s
==> v1beta1/DaemonSet
NAME        DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE SELECTOR  AGE
logtail-ds  2        2        0      2           0          <none>         0s
==> v1beta1/Deployment
NAME                    DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
alibaba-log-controller  1        1        1           0          0s
==> v1/Pod(related)
NAME                                    READY  STATUS             RESTARTS  AGE
logtail-ds-7xf2d                        0/1    ContainerCreating  0         0s
logtail-ds-9j4bx                        0/1    ContainerCreating  0         0s
alibaba-log-controller-796f8496b6-6jxb2  0/1    ContainerCreating  0         0s
==> v1/ServiceAccount
NAME                    SECRETS  AGE
alibaba-log-controller  1        0s
==> v1beta1/CustomResourceDefinition
NAME                                  AGE
aliyunlogconfigs.log.alibabacloud.com  0s
==> v1beta1/ClusterRole
alibaba-log-controller  0s
[INFO] your k8s is using project : k8s-log-custom-xxx, region : cn-hangzhou, aliuid :
***************, accessKeyId : LTA**********
[SUCCESS] install helm package : alibaba-log-controller success.
```

您可以使用 `helm status alibaba-log-controller` 查看日志服务组件状态，状态全部成功表示安装成功。

安装成功后登录日志服务控制台，即可看到已经自动创建出的日志服务Project（若您的Project数量过多，可以通过搜索 `k8s-log` 关键字进行过滤）。

步骤2 设置采集配置

请根据您的需求在控制台创建Logtail采集配置，具体步骤请参见：

- Kubernetes文本文件请参见容器文本日志。
- Kubernetes标准输出请参见容器标准输出。
- 宿主机文本文件。

  默认宿主机根目录挂载到Logtail容器的 `/logtail_host` 目录下，配置路径时您需要加上此前缀。例如需要采集宿主机上 `/home/logs/app_log/` 目录下的数据，则配置页面中日志路径设置为 `/logtail_host/home/logs/app_log/` 。

其他操作

- 多集群使用同一个日志服务Project

  如果您希望将多个集群的日志采集到同一个日志服务Project中，您可以在安装其他集群日志服务组件时，将上述安装参数中的 `${your_k8s_cluster_id}` 替换为您第一次安装的集群ID。

  例如您现在有3个集群，ID分别为abc001、abc002、abc003，三个集群安装组件的参数 `${your_k8s_cluster_id}` 都填写为 `abc001` 。

  > **说明** 此方式不支持跨region的Kubernetes多集群共享。

- Logtail容器日志

Logtail日志存储在Logtail容器中的 `/usr/local/ilogtail/` 目录中，文件名为 `ilogtail.LOG` 以及 `logtail_plugin.LOG` ，容器stdout并不具备参考意义，请忽略以下stdout输出。

```
start umount useless mount points, /shm$|/merged$|/mqueue$
umount:
/logtail_host/var/lib/docker/overlay2/3fd0043af174cb0273c3c7869500fbe2bdb95d13b1e110172ef57fe840c82155/m
ed: must be superuser to unmount
umount:
/logtail_host/var/lib/docker/overlay2/d5b10aa19399992755de1f85d25009528daa749c1bf8c16edff44beab6e69718/m
ed: must be superuser to unmount
umount:
/logtail_host/var/lib/docker/overlay2/5c3125daddacedec29df72ad0c52fac800cd56c6e880dc4e8a640b1e16c22dbe/m
ed: must be superuser to unmount
......
xargs: umount: exited with status 255; aborting
umount done
start logtail
ilogtail is running
logtail status:
ilogtail is running
```

- 查看Kubernetes集群中日志相关组件的状态

```
helm status alibaba-log-controller
```

- alibaba-log-controller启动失败
  请确认您是否按照以下方式进行安装：
  - 安装命令在Kubernetes集群的master节点执行。
  - 安装命令参数输入的是您的集群ID。

  若由于以上问题安装失败，请使用 `helm del --purge alibaba-log-controller` 删除安装包并重新执行安装命令。
- 查看Kubernetes集群中Logtail DaemonSet状态
  执行命令 `kubectl get ds -n kube-system` 查看Logtail运行状态。

  > **说明** Logtail默认的namespace为 `kube-system` 。

- 查看Logtail的版本号、IP、启动时间等信息
  示例如下：

```
[root@iZbp1dsu6v77zfb40qfbiaZ ~]# kubectl get po -n kube-system | grep logtail
NAME            READY     STATUS    RESTARTS   AGE
logtail-ds-gb92k   1/1       Running   0          2h
logtail-ds-wm7lw   1/1       Running   0          4d
[root@iZbp1dsu6v77zfb40qfbiaZ ~]# kubectl exec logtail-ds-gb92k -n kube-system cat
/usr/local/ilogtail/app_info.json
{
   "UUID" : "",
   "hostname" : "logtail-ds-gb92k",
   "instance_id" : "0EBB2B0E-0A3B-11E8-B0CE-0A58AC140402_172.20.4.2_1517810940",
   "ip" : "172.20.4.2",
   "logtail_version" : "0.16.2",
   "os" : "Linux; 3.10.0-693.2.2.el7.x86_64; #1 SMP Tue Sep 12 22:26:13 UTC 2017; x86_64",
   "update_time" : "2018-02-05 06:09:01"
}
```

- 查看Logtail的运行日志

  Logtail运行日志保存在 `/usr/local/ilogtail/` 目录下，文件名为 `ilogtail.LOG` ，轮转文件会压缩存储为 `ilogtail.LOG.x.gz` 。

  示例如下：

```
[root@iZbp1dsu6v77zfb40qfbiaZ ~]# kubectl exec logtail-ds-gb92k -n kube-system tail
/usr/local/ilogtail/ilogtail.LOG
[2018-02-05 06:09:02.168693] [INFO] [9] [build/release64/sls/ilogtail/LogtailPlugin.cpp:104] logtail
plugin Resume:start
[2018-02-05 06:09:02.168807] [INFO] [9] [build/release64/sls/ilogtail/LogtailPlugin.cpp:106] logtail
plugin Resume:success
[2018-02-05 06:09:02.168822] [INFO] [9] [build/release64/sls/ilogtail/EventDispatcher.cpp:369] start a
dd existed check point events, size:0
[2018-02-05 06:09:02.168827] [INFO] [9] [build/release64/sls/ilogtail/EventDispatcher.cpp:511] add
existed check point events, size:0 cache size:0 event size:0 success count:0
```

- 重启某个Pod的Logtail
  示例如下：

```
[root@iZbp1dsu6v77zfb40qfbiaZ ~]# kubectl exec logtail-ds-gb92k -n kube-system /etc/init.d/ilogtaild s
top
kill process Name: ilogtail pid: 7
kill process Name: ilogtail pid: 9
stop success
[root@iZbp1dsu6v77zfb40qfbiaZ ~]# kubectl exec logtail-ds-gb92k -n kube-system /etc/init.d/ilogtaild s
tart
ilogtail is running
```

3.1.6.3. 容器文本日志

Logtail支持采集容器内产生的文本日志，并附加容器的相关元数据信息一起上传到日志服务。

功能特点

相对基础的日志文件采集，Docker文件采集还具备以下功能特点：

- 只需配置容器内的日志路径，无需关心该路径到宿主机的映射。
- 支持label指定采集的容器。
- 支持label排除特定容器。
- 支持environment指定采集的容器。
- 支持environment指定排除的容器。
- 支持多行日志（例如java stack日志等）。
- 支持容器数据自动打标。
- 支持Kubernetes容器自动打标。

限制说明

- 采集停止策略：当container被停止后，Logtail监听到容器 `die` 的事件后会停止该容器日志的采集（延迟1~3秒），若此时采集出现延迟，则可能丢失停止前的部分日志。
- Docker存储驱动限制：目前只支持overlay、overlay2，其他存储驱动需将日志所在目录mount到本地。
- Logtail运行方式：必须以容器方式运行Logtail，且遵循Logtail部署方式进行部署。
- Label：此处的label为docker inspect中的label信息，并不是Kubernetes配置中的label。
- Environment：此处的environment为容器启动中配置的environment信息。

配置流程

1. 部署并配置Logtail容器。
2. 设置服务端采集配置。

Logtail部署和配置

- Kubernetes

  Kubernetes日志采集参见Kubernetes日志采集部署方案。

- 其他容器管理方式

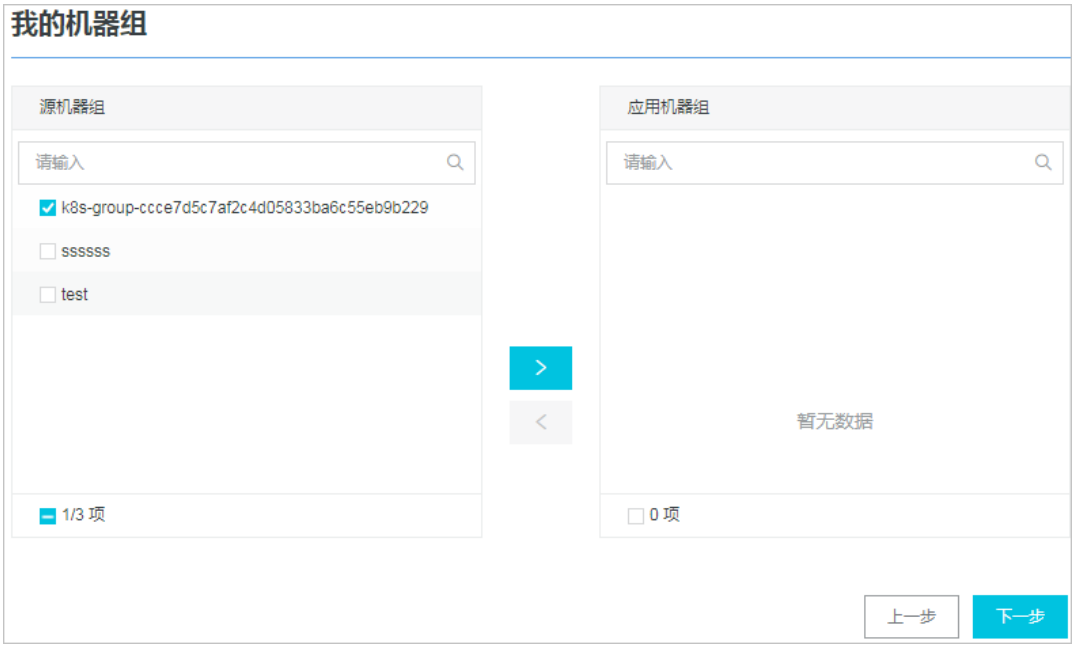
  Swarm、Mesos等其他容器管理方式，请参见Docker日志采集通用部署方案。

采集配置步骤

1. 登录日志服务控制台。

2. 单击**接入数据**按钮，并在**接入数据**页面中选择**Docker文件**。
3. 选择目标Project和Logstore，配置完成后，单击**下一步**。
   您也可以单击**立即创建**，重新创建Project和Logstore。
   如果您是通过日志库下的**数据接入**后的加号图标进入采集配置流程，系统会直接跳过该步骤。
4. 创建机器组，单击**下一步**。
   在创建机器组之前，您需要首先确认已经安装了Logtail。
   请根据界面提示进行安装。更多信息，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。
   安装完Logtail后，单击**确认安装完毕**创建机器组。如果您之前已经创建好机器组，也可以直接单击**使用现有机器组**。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **重要**
> 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为**FAIL**，您可单击**自动重试**。如果显示**FAIL**，请参见Logtail机器组无心跳处理。



6. 设置Logtail。
   此处列举该数据源的特有配置项，通用配置项说明请参见文本日志采集流程。

配置项	说明
是否为Docker文件	确认采集的目标文件是否为Docker文件。
Label白名单	每项中LabelKey必填，若LabelValue不为空，则只采集容器label中包含LabelKey=LabelValue的容器；若LabelValue为空，则采集所有label中包含LabelKey的容器。 **说明** ◦ 多个键值对间为或关系，即只要容器的label满足任一键值对即可被采集。 ◦ 此处的label为docker inspect中的label信息。
Label黑名单	每项中LabelKey必填，若LabelValue不为空，则只排除容器label中包含LabelKey=LabelValue的容器；若LabelValue为空，则排除所有label中包含LabelKey的容器。 **说明** ◦ 多个键值对间为或关系，即只要容器的label满足任一键值对即可被排除。 ◦ 此处的label为docker inspect中的label信息。

环境变量白名单	每项中EnvKey必填，若EnvValue不为空，则只采集容器环境变量中包含EnvKey=EnvValue的容器；若EnvValue为空，则采集所有环境变量中包含EnvKey的容器。 **说明** ◦ 多个键值对间为或关系，即只要容器的环境变量满足任一键值对即可被采集。 ◦ 此处的environment为容器启动中配置的environment信息。
环境变量黑名单	每项中EnvKey必填，若EnvValue不为空，则只排除容器环境变量中包含EnvKey=EnvValue的容器；若EnvValue为空，则排除所有环境变量中包含EnvKey的容器。 **说明** ◦ 多个键值对间为或关系，即只要容器的环境变量满足任一键值对即可被排除。 ◦ 此处的environment为容器启动中配置的environment信息。

说明

- 本文中label白名单、黑名单与Kubernetes中定义的label不是同一概念，本文档中的label为Docker inspect中的label信息。
- Kubernetes中的namespace和容器名会映射到docker的label中，分别为 `io.kubernetes.pod.namespace` 和 `io.kubernetes.container.name` 。例如您创建的pod所属namespace为backend-prod，容器名为worker-server，则仅需将其中一个配置为label白名单以指定只采集该容器的日志，分别为 `io.kubernetes.pod.namespace : backend-prod` 或者 `io.kubernetes.container.name : worker-server` 。
- Kubernetes中除 `io.kubernetes.pod.namespace` 和 `io.kubernetes.container.name` 外不建议使用其他label。其他情况请使用环境变量白名单或黑名单。

7. **查询和分析配置，包括配置索引。配置完成后，单击下一步。**

   日志服务默认开启全文索引。您也可以根据采集到的日志，手动或者自动设置字段索引。更多信息，请参见开启并配置索引。

说明

- 如果您要查询分析日志，那么全文索引和字段索引属性必须至少启用一种。同时启用时，以字段索引为准。
- 索引类型为long、double时，大小写敏感和分词符属性无效。

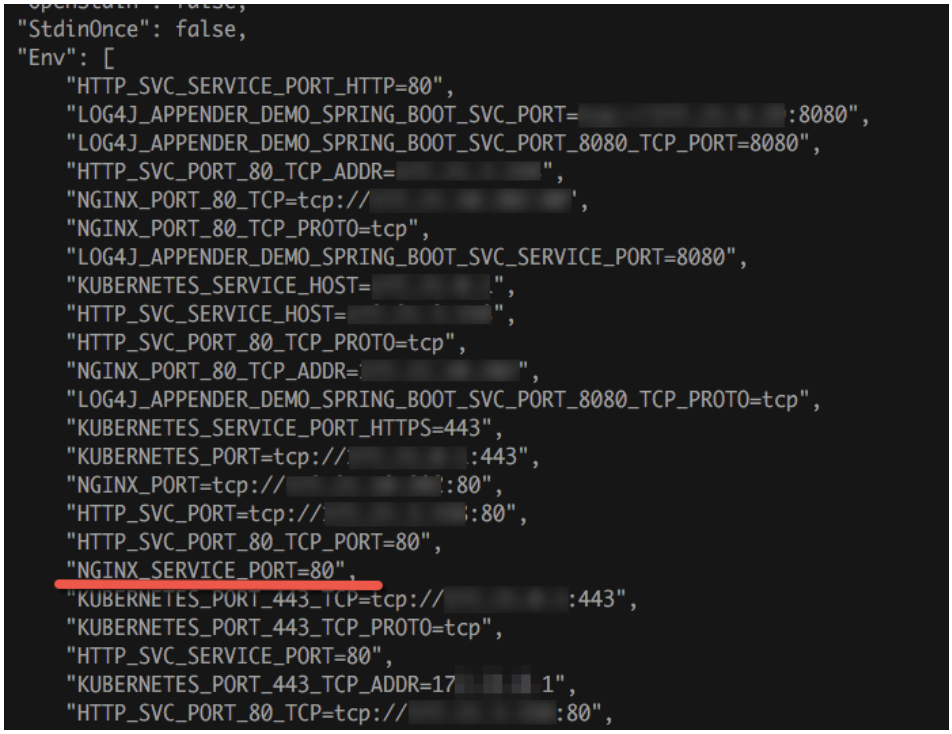
配置示例

- environment 配置方式

  采集environment为 `NGINX_PORT_80_TCP_PORT=80` 且environment不为 `POD_NAMESPACE=kube-system` 的容器日志，日志文件路径为 `/var/log/nginx/access.log` ，日志解析方式为极简类型。

说明 此处的environment为容器启动中配置的environment信息。

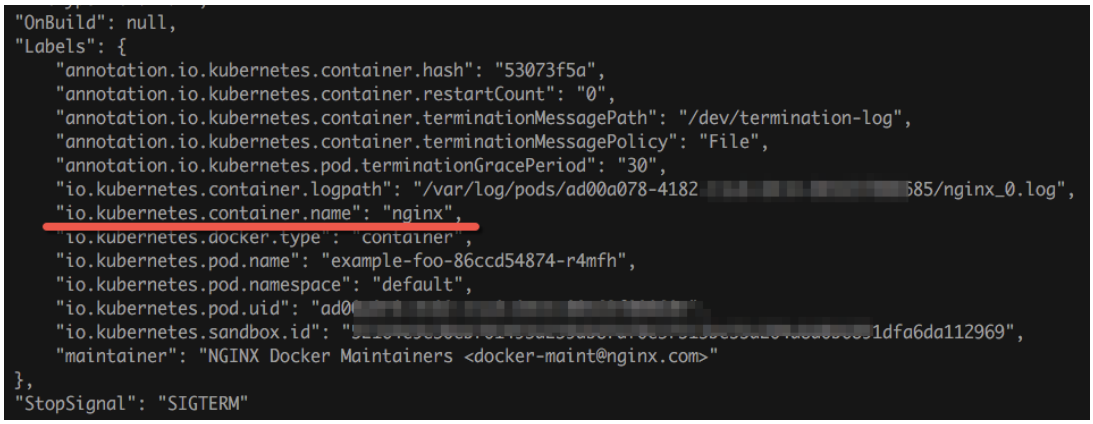
图 1. environment 配置方式示例



- Label 配置方式
  采集label为 `io.kubernetes.container.name=nginx` 的容器日志，日志文件路径为 `/var/log/nginx/access.log` ，日志解析方式为极简类型。

  > **说明** 此处的label为docker inspect中的label信息，并不是Kubernetes配置中的label。

  图 2. label方式示例



默认字段

普通Docker默认每条日志上传以下字段：

字段名	说明
_image_name_	镜像名
_container_name_	容器名
_container_ip_	容器IP地址

若该集群为Kubernetes，则默认每条日志上传以下字段：

字段名	说明
_image_name_	镜像名
_container_name_	容器名
_pod_name_	pod名
namespace	pod所在命名空间
_pod_uid_	pod的唯一标识
_container_ip_	pod的IP地址

3.1.6.4. 容器标准输出

Logtail支持将容器的标准输出流作为输入源，并附加容器的相关元数据信息一起上传到日志服务。

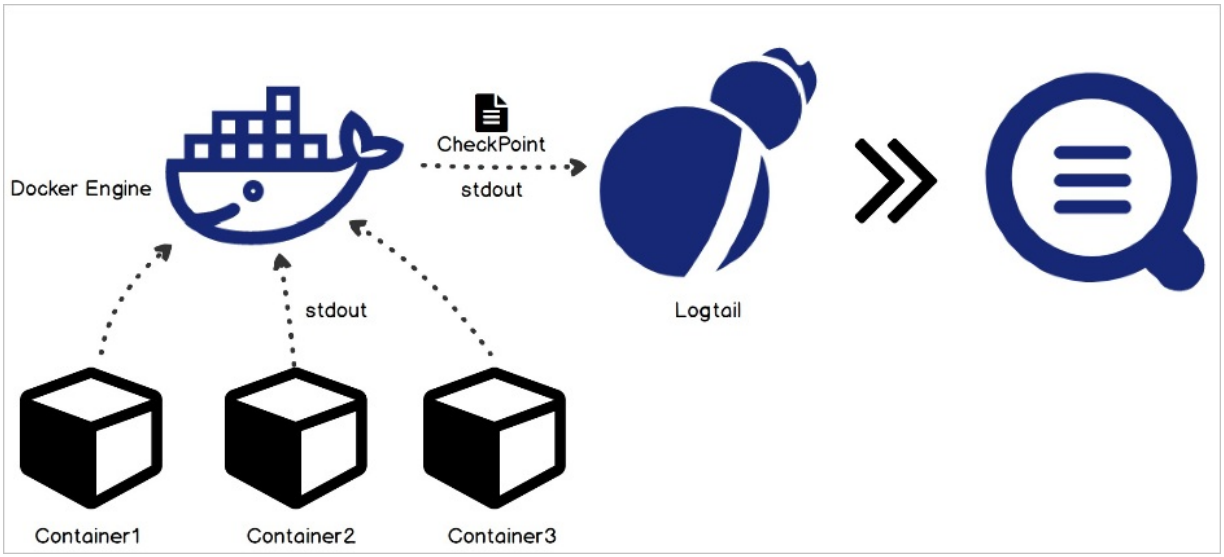
功能特点

- 支持采集stdout、stderr。
- 支持label指定采集的容器。
- 支持label排除特定容器。
- 支持environment指定采集的容器。
- 支持environment指定排除的容器。
- 支持多行日志（例如java stack日志等）。
- 支持container数据自动打标。
- 支持Kubernetes容器自动打标。

实现原理

如下图所示，Logtail会与Docker的Domain Socket进行通信，查询该Docker上运行的所有container，并根据container配置的label和environment信息定位需要被采集的container。随后Logtail会通过docker logs命令获取指定container的日志。

Logtail在采集容器的标准输出时，会定期将采集的点位信息保存到checkpoint文件中，若Logtail停止后再次启动，会从上一次保存的点位开始采集日志。



使用限制

- 此功能目前仅支持Linux，依赖Logtail 0.16.0及以上版本，版本查看与升级参见安装Logtail（Linux系统）。
- Logtail默认通过 `/var/run/docker.sock` 访问docker engine，请确保该Domain Socket存在且具备访问权限。
- 多行日志限制：为保证多行组成的一条日志不因为输出延迟而被分割成多条，多行日志情况下，采集的最后一条日志默认都会

缓存一段时间。默认缓存时间为3秒，可通过 `BeginLineTimeoutMs` 设置，但此值不能低于1000，否则容易出现误判。

- 采集停止策略：当container被停止后，Logtail监听到容器 `die` 的事件后会停止采集该container的标准输出，若此时采集出现延迟，则可能丢失停止前的部分输出。
- Docker日志驱动类型限制：目前标准输出采集仅支持JSON类型的日志驱动。
- 上下文限制：默认一个采集配置在同一上下文中，若需要每种类型的container配置在不同上下文中，请单独配置。
- 数据处理：采集到的数据默认字段为 `content` ，支持通用的处理配置。请参见处理采集数据配置一种或多种处理方式。
- Label：此处的label为docker inspect中的label信息，并不是Kubernetes配置中的label。
- Environment：此处的environment为容器启动文件中配置的environment信息。

配置流程

1. 部署并配置Logtail容器。
2. 设置服务端采集配置。

Logtail部署和配置

- Kubernetes

  Kubernetes日志采集请参见Kubernetes日志采集部署方案。

- 其他容器管理方式

  Swarm、Mesos等其他容器管理方式，请参见Docker日志采集通用部署方案。

设置数据源

1. 登录日志服务控制台。
2. 单击**接入数据**按钮，并在**接入数据**页面中选择**Docker标准输出**。
3. 选择目标Project和Logstore，配置完成后，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和Logstore。

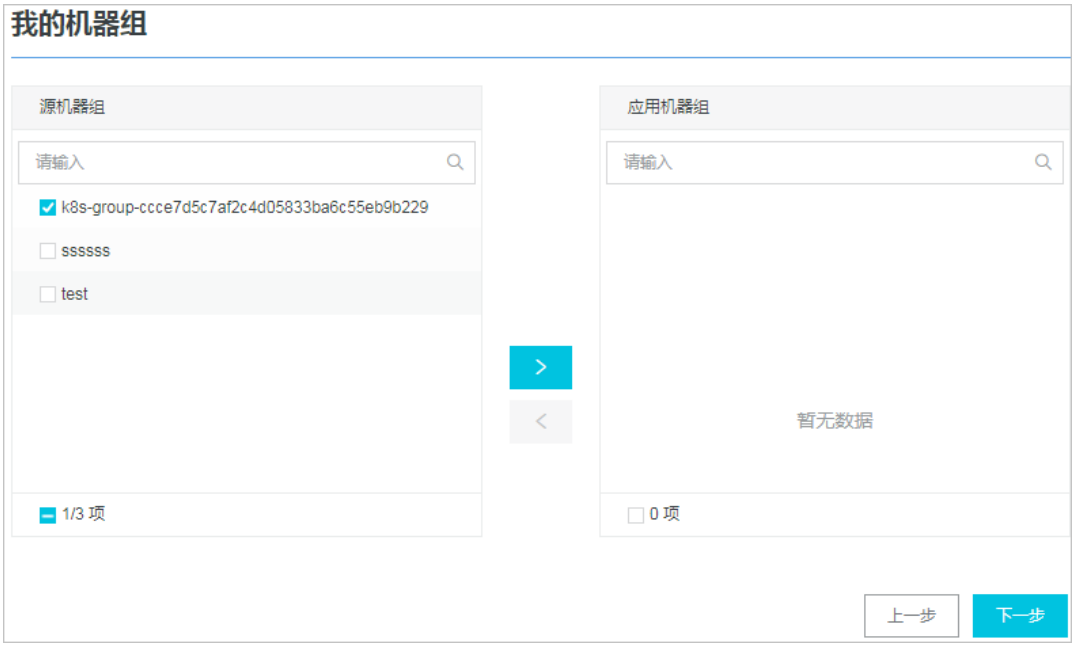
   如果您是通过日志库下的**数据接入**后的加号图标进入采集配置流程，系统会直接跳过该步骤。
4. 创建机器组，单击**下一步**。

   在创建机器组之前，您需要首先确认已经安装了Logtail。

   请根据界面提示进行安装。更多信息，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。

   安装完Logtail后，单击**确认安装完毕**创建机器组。如果您之前已经创建好机器组，也可以直接单击**使用现有机器组**。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **重要**
>
> 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为**FAIL**，您可单击**自动重试**。如果显示**FAIL**，请参见Logtail机器组无心跳处理。



6. **设置数据源。**

   在插件配置中填写您的采集配置。示例如下，配置项说明请查看本文档中配置项说明部分。

```
{
 "inputs": [
     {
         "type": "service_docker_stdout",
         "detail": {
             "Stdout": true,
             "Stderr": true,
             "IncludeLabel": {
                 "io.kubernetes.container.name": "nginx"
             },
             "ExcludeLabel": {
                 "io.kubernetes.container.name": "nginx-ingress-controller"
             },
             "IncludeEnv": {
                 "NGINX_SERVICE_PORT": "80"
             },
             "ExcludeEnv": {
                 "POD_NAMESPACE": "kube-system"
             }
         }
     }
 ]
}
```

7. **查询和分析配置，包括配置索引。配置完成后，单击下一步。**

   日志服务默认开启全文索引。您也可以根据采集到的日志，手动或者自动设置字段索引。更多信息，请参见开启并配置索引。

> **说明**
> - 如果您要查询分析日志，那么全文索引和字段索引属性必须至少启用一种。同时启用时，以字段索引为准。
> - 索引类型为long、double时，大小写敏感和分词符属性无效。

配置项说明

该输入源类型为： `service_docker_stdout` **。**

> **说明** Logtail支持对采集的数据进行处理加工后上传，处理方式参见处理采集数据。

配置项	类型	是否必选	说明
IncludeLabel	map类型，其中key为string，value为string	必选	默认为空，为空时代表采集所有Container数据；当key非空，value为空时，代表包含label中所有包含此key的container。 **说明** • 多个键值对间为或关系，即只要容器的label满足任一键值对即可被采集。 • 此处的label为docker inspect中的label信息。
ExcludeLabel	map类型，其中key为string，value为string	可选	默认为空，为空时不排除任何Container；当key非空，value为空时，代表排除label中所有包含此key的container。 **说明** • 多个键值对间为或关系，即只要容器的label满足任一键值对即被排除。 • 此处的label为docker inspect中的label信息。
IncludeEnv	map类型，其中key为string，value为string	可选	默认为空，为空时代表采集所有Container数据；当key非空，value为空时，代表包含environment中所有包含此key的container。 **说明** • 多个键值对间为或关系，即只要容器的environment满足任一键值对即被采集。 • 此处的environment为容器启动中配置的environment信息。
ExcludeEnv	map类型，其中key为string，value为string	可选	默认为空，为空时不排除任何Container；当key非空，value为空时，代表排除Environment中所有包含此key的container。 **说明** • 多个键值对间为或关系，即只要容器的environment满足任一键值对即被排除。 • 此处的environment为容器启动中配置的environment信息。
Stdout	bool	可选	默认为true，为false时不采集stdout数据。
Stderr	bool	可选	默认为true，为false时不采集stderr数据。
BeginLineRegex	string	可选	默认为空，非空时为行首匹配的正则表达式。若该表达式匹配某行，则将该行作为新的一条日志；否则将此行数据连接到上一条日志。
BeginLineTimeoutMs	int	可选	行首匹配超时时间，默认为3000，单位为毫秒。若3秒内没有新日志出现，则将最后一条日志输出。
BeginLineCheckLength	int	可选	行首匹配的长度，默认为10×1024，单位为字节。若行首规则在前N个字节即可体现，可设置此参数，以此提升行首匹配效率。
MaxLogSize	int	可选	日志最大长度，默认为512×1024，单位为字节。若日志超过该项配置，则不继续查找行首，直接上传。

> **说明**
> - 本文档中IncludeLabel、ExcludeLabel和Kubernetes中定义的label不是同一概念，本文档中的label为docker inspect中的label信息。
> - Kubernetes中的namespace和容器名会映射到docker的label中，分别为 `io.kubernetes.pod.namespace` 和 `io.kubernetes.container.name` 。例如您创建的pod所属namespace为backend-prod，容器名为worker-server，则仅需将其中一个配置为label白名单以指定只采集该容器的日志，分别为 `io.kubernetes.pod.namespace : backend-prod` 或者 `io.kubernetes.container.name : worker-server` 。
> - Kubernetes不建议使用除 `io.kubernetes.pod.namespace` 和 `io.kubernetes.container.name` 之外的其他label。其他情况请使用IncludeEnv/ExcludeEnv。

默认字段

- 普通Docker
  每条日志默认上传以下字段：

字段名	说明
`_time_`	数据上传时间，样例为 `2018-02-02T02:18:41.979147844Z`
`_source_`	输入源类型，stdout 或 stderr
`_image_name_`	镜像名
`_container_name_`	容器名
`_container_ip_`	容器IP

- Kubernetes
  每条日志默认上传以下字段：

字段名	说明
`_time_`	数据上传时间，样例为 `2018-02-02T02:18:41.979147844Z`
`_source_`	输入源类型，stdout 或 stderr
`_image_name_`	镜像名
`_container_name_`	容器名
`_pod_name_`	pod名
`_namespace_`	pod所在命名空间
`_pod_uid_`	pod的唯一标识
`_container_id_`	pod的IP地址

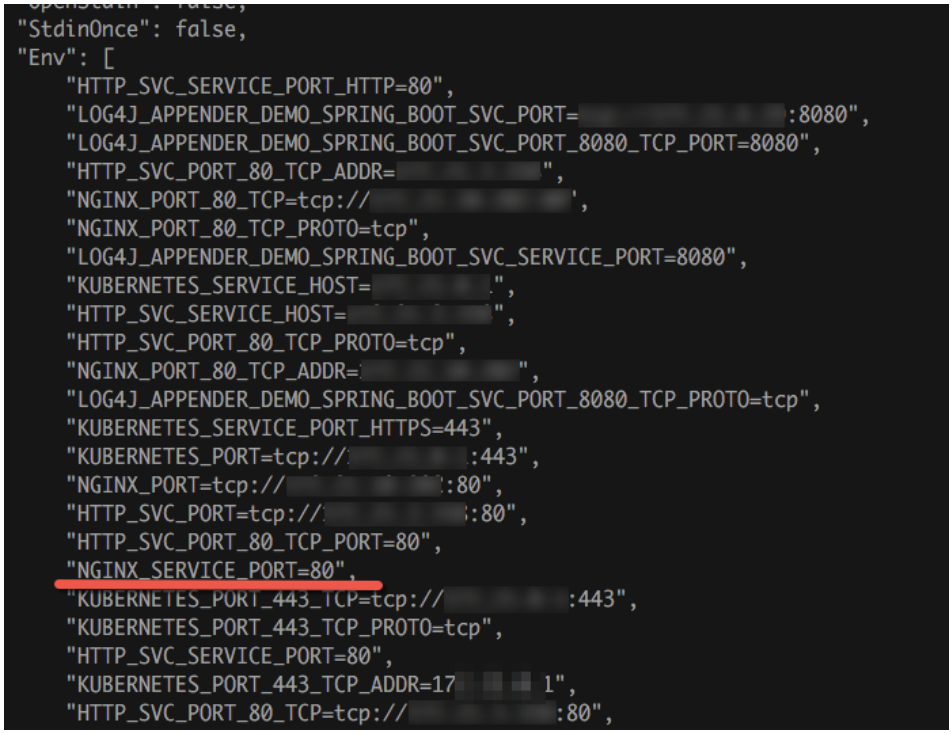
常规配置示例

- environment 配置方式

  采集environment为 `NGINX_PORT_80_TCP_PORT=80` 且environment不为 `POD_NAMESPACE=kube-system` 的stdout以及stderr日志：

> **说明** 此处的environment为容器启动中配置的environment信息。

图 1. environment 配置方式示例



采集配置：

```
{
    "inputs": [
        {
            "type": "service_docker_stdout",
            "detail": {
                "Stdout": true,
                "Stderr": true,
                "IncludeEnv": {
                    "NGINX_PORT_80_TCP_PORT": "80"
                },
                "ExcludeEnv": {
                    "POD_NAMESPACE": "kube-system"
                }
            }
        }
    ]
}
```

- Label 配置方式

  采集label为 `io.kubernetes.container.name=nginx` 且label不为 `type=pre` 的stdout以及stderr日志：

  > **说明** 此处的label为docker inspect中的label信息，并不是Kubernetes配置中的label。

  图 2. label配置方式示例


```
{
    "inputs": [
        {
            "type": "service_docker_stdout",
            "detail": {
                "Stdout": true,
                "Stderr": true,
                "IncludeLabel": {
                    "io.kubernetes.container.name": "nginx"
                },
                "ExcludeLabel": {
                    "type": "pre"
                }
            }
        }
    ]
}
```

多行日志采集配置示例

多行日志采集对于Java异常堆栈输出的采集尤为重要，这里介绍一种标准的Java标准输出日志的采集配置。

- 日志示例

```
2018-02-03 14:18:41.968  INFO [spring-cloud-monitor] [nio-8080-exec-4]
c.g.s.web.controller.DemoController : service start
2018-02-03 14:18:41.969 ERROR [spring-cloud-monitor] [nio-8080-exec-4]
c.g.s.web.controller.DemoController : java.lang.NullPointerException
at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:193)
at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:166)
at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:199)
at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:96)
...
2018-02-03 14:18:41.968  INFO [spring-cloud-monitor] [nio-8080-exec-4]
c.g.s.web.controller.DemoController : service start done
```

- 采集配置

  采集label为 `app=monitor` 输入的日志，行首为日期类型（为提高匹配效率，这里只判断行首的10个字节）。

```
{
"inputs": [
  {
    "detail": {
      "BeginLineCheckLength": 10,
      "BeginLineRegex": "\\d+-\\d+-\\d+.*",
      "IncludeLabel": {
        "app": "monitor"
      }
    },
    "type": "service_docker_stdout"
  }
]
}
```

采集数据处理

Logtail对于采集到的Docker标准输出，支持通用数据处理方式。使用正则表达式将日志解析成time、module、thread、class、info。

- 采集配置：
  采集label为 `app=monitor` 输入的日志，行首为日期类型（为提高匹配效率，这里只判断行首的10个字节）。

```
{
"inputs": [
  {
    "detail": {
      "BeginLineCheckLength": 10,
      "BeginLineRegex": "\\d+-\\d+-\\d+.*",
      "IncludeLabel": {
        "app": "monitor"
      }
    },
    "type": "service_docker_stdout"
  }
],
"processors": [
    {
        "type": "processor_regex",
        "detail": {
            "SourceKey": "content",
            "Regex": "(\\d+-\\d+-\\d+ \\d+:\\d+:\\d+\\.\\d+)\\s+(\\w+)\\s+\\[([^]]+)]\\s+\\[([^]]+)]\\
s+:\\s+([\\s\\S]*)",
            "Keys": [
                "time",
                "module",
                "thread",
                "class",
                "info"
            ],
            "NoKeyError": true,
            "NoMatchError": true,
            "KeepSource": false
        }
    }
]
}
```

- 样例输出

  对于日志 `2018-02-03 14:18:41.968 INFO [spring-cloud-monitor] [nio-8080-exec-4] c.g.s.web.controller.DemoController : service start done` 处理后的输出为：

```
__tag__:__hostname__:logtail-dfgef
_container_name_:monitor
_image_name_:registry.cn-hangzhou.aliyuncs.xxxxxxxxxxxxxx
_namespace_:default
_pod_name_:monitor-6f54bd5d74-rtzc7
_pod_uid_:7f012b72-04c7-11e8-84aa-00163f00c369
_source_:stdout
_time_:2018-02-02T14:18:41.979147844Z
time:2018-02-02 02:18:41.968
level:INFO
module:spring-cloud-monitor
thread:nio-8080-exec-4
class:c.g.s.web.controller.DemoController
message:service start done
```

3.1.7. 相关限制说明

本文档主要介绍与Logtail相关的限制信息，包括文件采集、资源、错误处理等限制。

文件采集限制

分类	限制说明
文件编码	支持UTF8/GBK编码日志文件，建议使用UTF8编码以获得更好的处理性能。如果日志文件为其它编码格式则会出现乱码、数据丢失等错误。
日志文件大小	无限制。
日志文件轮转	支持，流转文件名支持配置为 `.log*` 或者 `.log` 。
日志解析阻塞时采集行为	日志解析阻塞时，Logtail会将该日志文件FD保持打开状态；若解析阻塞期间出现多次日志文件轮转，Logtail会尽可能保持各个轮转日志解析顺序。若未解析的日志轮转超过20个，则后续文件不被处理。
软链接	支持监控目录为软链接。
单条日志大小	单条日志大小限制为512KB。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。若日志超过512KB后，会强制拆分多块进行采集。例如：日志单条1025KB，则第一次处理前512KB，第二次处理512KB，第三次处理1KB。
正则表达式	正则表达式类型支持Perl兼容正则表达式。
同一文件对应多个采集配置	不支持，建议文件采集到一个Logstore，可以配置多份订阅。若有相关需求，可通过为文件配置软连接的方式绕过该限制。
文件打开行为	Logtail会保持被采集文件处于打开状态，若该文件超过5分钟未修改，则会关闭该文件（未发生轮转情况下）。
首次日志采集行为	Logtail只采集增量的日志文件，首次发现文件修改后，若文件大小超过1M，则从最后1M处开始采集，否则从开始位置采集；若配置下发后日志文件一直无修改，则不采集该文件。
非标准文本日志	对于日志中包含'\0'的行，该条日志会被截断到第一个'\0'处。

Checkpoint管理

项目	能力与限制
Checkpoint超时时间	若文件超过30天未修改，则会删除该Checkpoint。
Checkpoint保存策略	定期保存（15分钟），程序退出时会自动保存。
Checkpoint保存位置	保存路径默认为 `/tmp/logtail_checkpoint` ，可根据配置启动参数调整参数。

配置限制

项目	能力与限制
配置更新	用户的配置更新生效的延时约30秒。
配置动态加载	支持，且其中某一配置更新不影响其他采集。
配置数	理论无限制，建议一台服务器采集配置数不超过100。
多租户隔离	各个采集配置间隔离。

资源、性能限制

项目	能力与限制
日志处理吞吐能力	原始日志流量默认限制为2MB/s（数据会编码压缩后上传，一般压缩率为5-10倍）。超过该日志流量则有可能丢失日志，可根据配置启动参数调整参数。
最大性能	单核能力：极简模式日志最大处理能力为100MB/s，正则默认最大处理能力为20MB/s（和正则复杂度有关），分隔符日志最大处理能力为40MB/s，JSON日志最大处理能力为30MB/s；开启多个处理线程性能可提高1.5-3倍左右
监控目录数	主动限制监控的目录层深，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）。
监控文件数	每台服务器上的每个Logtail采集配置监控的最大文件数量为10,000个，每台服务器上的Logtail客户端最多可监控100,000个文件。超出限制的文件不监控。 达到限制时，您可以： • 在Logtail配置中提高监控目录的精度。 • 参见配置启动参数修改mem_usage_limit，提高Logtail内存。 Logtail内存最大可调整至2 GB，表示每个Logtail采集配置可监控100,000个文件，每个Logtail客户端可监控的文件数对应提高至1,000,000个。
默认资源限制	默认Logtail最多会占用40%CPU、256MB内存，如日志产生速率较高，可根据 配置启动参数调整参数。
资源超限处理策略	若3分钟内Logtail占用的相关资源超过最大限制，则Logtail会强制重启，此时数据可能会丢失或重复。

错误处理限制

项目	能力与限制
网络错误处理	在出现网络异常时会主动重试并自动调整重试间隔。
资源配额超限处理	若数据发送速率超出Logstore最大配额，Logtail会阻塞采集并自动重试。
超时最大尝试时间	若数据持续发送失败超过6小时，则丢弃该数据。
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。

其他限制

项目	能力与限制
日志采集延迟	正常情况下从日志写入磁盘到Logtail采集改日志延迟不超过1秒（阻塞状态下除外）。
日志上传策略	Logtail会将同一文件的日志自动聚合上传，聚合条件为：日志超过2000条、日志总大小超过2M或者日志采集时间超过3秒，任一条件满足则触发上传行为。

3.2. 其他采集方式

3.2.1. WebTracking

本文档向您介绍如何通过WebTracking功能采集HTML、H5、iOS和Android平台日志数据。

背景信息

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

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

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

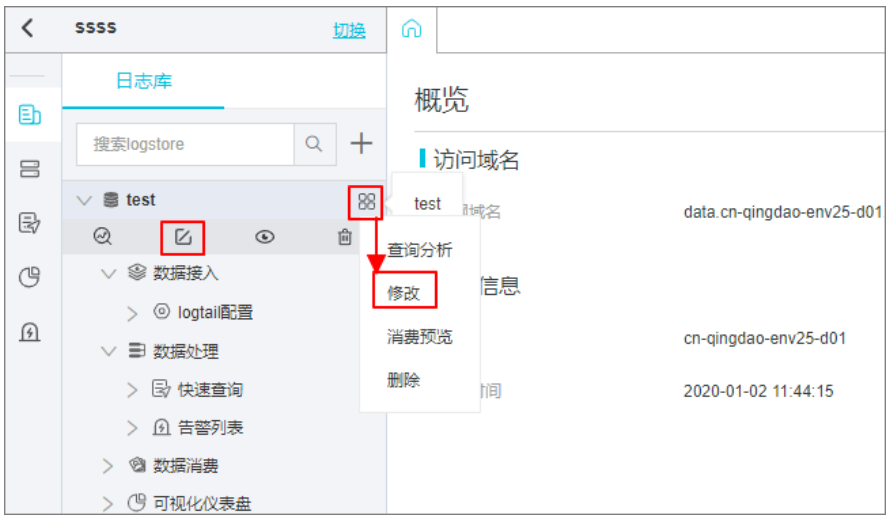
注意事项

- 使用WebTracking意味着该Logstore打开互联网匿名写入的权限，没有经过有效鉴权，可能会产生脏数据。
- 支持GET请求，但不支持上传16KB以上的body。
- POST请求限制与PutLogs一致，大小不超过3MB，条数不超过4096条。

步骤1 开通WebTracking

通过控制台开通WebTracking。

1. 登录日志服务控制台。
2. 选择目标Logstore，进入**Logstore属性**页面。
   - 方式1：单击Logstore下面的修改图标。
   - 方式2：单击Logstore后面的选择**修改**。



3. 单击**Logstore属性**页面右上方的**修改**。
4. 打开WebTracking开关。

通过Java SDK开通WebTracking

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WebTracking功能。
          LogStore logSt = client.GetLogStore(project, logStore).GetLogStore();
          client.UpdateLogStore(project, new LogStore(logStore, logSt.GetTtl(), logSt.GetShardCount(), t
rue));
          //关闭WebTracking功能。
          //client.UpdateLogStore(project, new LogStore(logStore, logSt.GetTtl(), logSt.GetShardCount(),
false));
          //新建Logstore并开通WebTracking功能。
          //client.UpdateLogStore(project, new LogStore(logStore, 1, 1, true));
      }
      catch (LogException e){
          e.printStackTrace();
      }
  }
}
```

步骤2 收集日志数据

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

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

     单击这里对内容进行复制。

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

> **说明** 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WebTracking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)
> ```

ii. 创建Tracker对象。

```
var logger = new window.Tracker('${host}','${project}','${logstore}');
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();
```

其中各个参数的含义如下：

字段	含义
`${host}`	您日志服务所在Region的endpoint。请参见日志服务开发指南中的**《获取Endpoint》**。
`${project}`	您在日志服务中开通的Project名称。
`${logstore}`	`${project}` 中的Logstore的名称。

执行以上命令后，可以在日志服务看到如下两条日志：

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

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

- 使用HTTP GET请求

```
curl --request GET 'http://${project}.${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。
${host}	您日志服务所在地区的域名。
${logstore}	`${project}` 下面开通WebTracking功能的某一个Logstore的名称。
APIVersion=0.6.0	保留字段，必选。
__topic__=yourtopic	指定日志的topic，保留字段，可选
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://${project}.${host}/logstores/${logstore}/track.gif?
APIVersion=0.6.0&key1=val1&key2=val2'/>
<img src='http://${project}.${host}/logstores/${logstore}/track_ua.gif?
APIVersion=0.6.0&key1=val1&key2=val2'/>
```

各个参数的含义同上，track_ua.gif除了将自定义的参数上传外，在服务端还会将http头中的UserAgent、referer也作为日志中的字段。

3.2.2. SDK采集

3.2.2.1. Producer Library

Aliyun LOG Java Producer 是一个易于使用且高度可配置的 Java 类库，专门为运行在大数据、高并发场景下的 Java 应用量身打造。

Github 项目地址以及更多详细说明请参见Aliyun LOG Java Producer。

3.2.2.2. Log4j Appender

本文介绍Log4j日志的概念及Alibaba Cloud Log Log4j Appender的操作方法。

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及输出格式。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。Log4j由三个重要的组件构成，如下所示。

- 日志记录器（Loggers）

  日志信息的优先级从高到低分别为ERROR、WARN、INFO和DEBUG，分别用来指定这条日志信息的重要程度。

- 日志输出端（Appenders）

  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。

- 日志格式化器（Layout）

  日志信息的输出格式控制了日志信息的显示内容。

通过Alibaba Cloud Log Log4j Appender，您可以控制日志的输出目的地为阿里云日志服务，下载地址及使用说明请参见Log4j Appender。

3.2.2.3. Logback Appender

Logback是由Log4j创始人设计的又一个开源日志组件。通过使用Logback，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。

通过Aliyun Log Logback Appender，您可以控制日志的输出目的地为阿里云日志服务，写到日志服务中的日志的样式如下。

```
level: ERROR
location:
com.aliyun.openservices.log.logback.example.LogbackAppenderExample.main(LogbackAppenderExample.java:18)
message: error log
throwable: java.lang.RuntimeException: xxx
thread: main
time: 2018-01-02T03:15+0000
log: 2018-01-02 11:15:29,682 ERROR [main]
com.aliyun.openservices.log.logback.example.LogbackAppenderExample: error log
__source__: xxx
__topic__: yyy
```

下载地址及使用说明请参见Logback Appender。

3.2.2.4. Golang Producer Library

Aliyun LOG Go Producer Library 是一个易于使用且高度可配置的 Golang 类库，专门为运行在大数据、高并发场景下的 golang 应用量身打造，使用producer 会自动重试发送失败的日志，并且将要发送的数据进行压缩，以提高数据写入效率。

Github 项目地址以及更多详细说明请参见：Aliyun Log Golang Producer 。

3.2.2.5. Python logging

配置

关于Python logging模块的配置，请参见Python logging。

Python logging模块允许通过编程或者文件的形式配置日志，如下我们通过文件 `logging.conf` 配置。

```
[loggers]
keys=root,sls

[handlers]
keys=consoleHandler, slsHandler

[formatters]
keys=simpleFormatter, rawFormatter

[logger_root]
level=DEBUG
handlers=consoleHandler

[logger_sls]
level=INFO
handlers=consoleHandler, slsHandler
qualname=sls
propagate=0

[handler_consoleHandler]
class=StreamHandler
level=DEBUG
formatter=simpleFormatter
args=(sys.stdout,)

[handler_slsHandler]
class=aliyun.log.QueuedLogHandler
level=INFO
formatter=rawFormatter
args=(os.environ.get('ALIYUN_LOG_SAMPLE_ENDPOINT', ''), os.environ.get('ALIYUN_LOG_SAMPLE_ACCESSID',
''), os.environ.get('ALIYUN_LOG_SAMPLE_ACCESSKEY', ''), os.environ.get('ALIYUN_LOG_SAMPLE_TMP_PROJECT',
''), "logstore")

[formatter_simpleFormatter]
format=%(asctime)s - %(name)s - %(levelname)s - %(message)s


[formatter_rawFormatter]
format=%(message)s
```

此处我们配置了一个 `root` 和一个 `sls` 的Log Handler，其中 `sls` 是实例化类 `aliyun.log.QueuedLogHandler` ，并传入如下参数。详细参数请参见详细参数列表。

```
args=(os.environ.get('ALIYUN_LOG_SAMPLE_ENDPOINT', ''), os.environ.get('ALIYUN_LOG_SAMPLE_ACCESSID',
''), os.environ.get('ALIYUN_LOG_SAMPLE_ACCESSKEY', ''), os.environ.get('ALIYUN_LOG_SAMPLE_TMP_PROJECT',
''), "logstore")
```

> **说明** 这里使用了 `os.environ` 从环境变量中获取相关配置。您也可以直接填写实际的值。

上传日志

使用logging配置文件并输出日志即可。

```
import logging
import logging.config

# 配置
logging.config.fileConfig('logging.conf')
logger = logging.getLogger('sls')

# 使用logger
logger.info("test1")

try:
    1/0
except ZeroDivisionError as ex:
    logger.exception(ex)
```

之后日志即可自动上传到日志服务。如果要使用统计查询功能，请打开索引。

配置日志服务Logstore的索引

将接收日志的Logstore的索引打开，将特定域进行索引。推荐使用阿里云命令行工具CLI进行如下配置。

```
aliyunlog log update_index --project_name="project1" --logstore_name="logstore1" --
index_detail="file:///Users/user1/loghandler_index.json"
```

配置文件请参见python_logging_handler_index.json。

调整收集日志域

目前支持如下的日志信息，默认会收集所有相关域。

域	说明
message	消息内容。
record_name	logging handler的名字，上面例子是 `sls` 。
level	级别，INFO、ERROR等。
file_path	代码文件全路径。
func_name	所在函数名。
line_no	行号。
module	所在模块。
thread_id	当前线程Id。
thread_name	当前线程名。
process_id	当前进程Id。
process_name	当前进程名。

参考QueuedLogHandler的参数 `fields` 接收一个列表来调整想要配置的域。具体请参见日志域列表。

下面例子中，我们修改之前的日志配置文件，只收集个别域如module、func_name等。

```
[handler_slsHandler]
class=aliyun.log.QueuedLogHandler
level=INFO
formatter=rawFormatter
args=('cn-beijing.log.aliyuncs.com', 'ak_id', 'ak_key', 'project1', "logstore1", 'mytopic', ['level', 'f
unc_name', 'module', 'line_no']  )
```

> 说明
> - message是一定会被收集的。
> - 您也可以通过参数 `buildin_fields_prefix` 和 `buildin_fields_suffix` 给这些内置域增加前缀和后缀。例如 `__level__` 等。

使用JSON配置

如果期望更加灵活的配置，也可以使用代码配置。

```
#encoding: utf8
import logging, logging.config, os

# 配置
conf = {'version': 1,
        'formatters': {'rawformatter': {'class': 'logging.Formatter',
                                        'format': '%(message)s'}
                       },
        'handlers': {'sls_handler': {'()':
                                     'aliyun.log.QueuedLogHandler',
                                     'level': 'INFO',
                                     'formatter': 'rawformatter',

                                     # custom args:
                                     'end_point': os.environ.get('ALIYUN_LOG_SAMPLE_ENDPOINT', ''),
                                     'access_key_id': os.environ.get('ALIYUN_LOG_SAMPLE_ACCESSID', ''),
                                     'access_key': os.environ.get('ALIYUN_LOG_SAMPLE_ACCESSKEY', ''),
                                     'project': 'project1',
                                     'log_store': "logstore1"
                                     }
                     },
        'loggers': {'sls': {'handlers': ['sls_handler', ],
                                    'level': 'INFO',
                                    'propagate': False}
                    }
        }
logging.config.dictConfig(conf)

# 使用
logger = logging.getLogger('sls')
logger.info("Hello world")
```

> 说明 `QueuedLogHandler` 的初始化方式，用的是传入命名参数的方式。具体参数列表可以参见这里。

3.2.3. 常见日志格式

3.2.3.1. Log4j日志

日志服务支持采集Log4j日志。

通过Loghub Log4j Appender采集Log4j日志

详细内容及采集步骤请参见Log4j Appender。

通过Logtail采集Log4j日志

Log4j日志分第一代和第二代，本文档以第一代的默认配置为例，讲述如何配置正则式，如果采用Log4j 2，需要修改默认配置，把日期完整打印出来。

```
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss:SSS zzz} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Logger name="com.foo.Bar" level="trace">
      <AppenderRef ref="Console"/>
    </Logger>
    <Root level="error">
      <AppenderRef ref="Console"/>
    </Root>
  </Loggers>
</Configuration>
```

配置Logtail收集Log4j日志的详细操作步骤请参见Python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]

3.2.3.2. Python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。

Python的 logging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，例如文件、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 的机制相同，只是具体的实现细节不同。模块提供 logger、handler、filter、formatter。

采集Python日志，推荐您直接使用Logging Handler：

- 使用Log Handler自动上传Python日志
- Log Handler自动解析KV格式的日志

- Log Handler自动解析JSON格式的日志

Python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

日志输出样例：

```
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
```

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 13986599668707
2 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+(.*)
```

配置Logtail收集Python日志

1. 登录日志服务控制台。
2. 单击**接入数据**按钮，并在**接入数据**页面中选择**正则-文本文件**。
3. 选择目标Project和Logstore，配置完成后，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和Logstore。

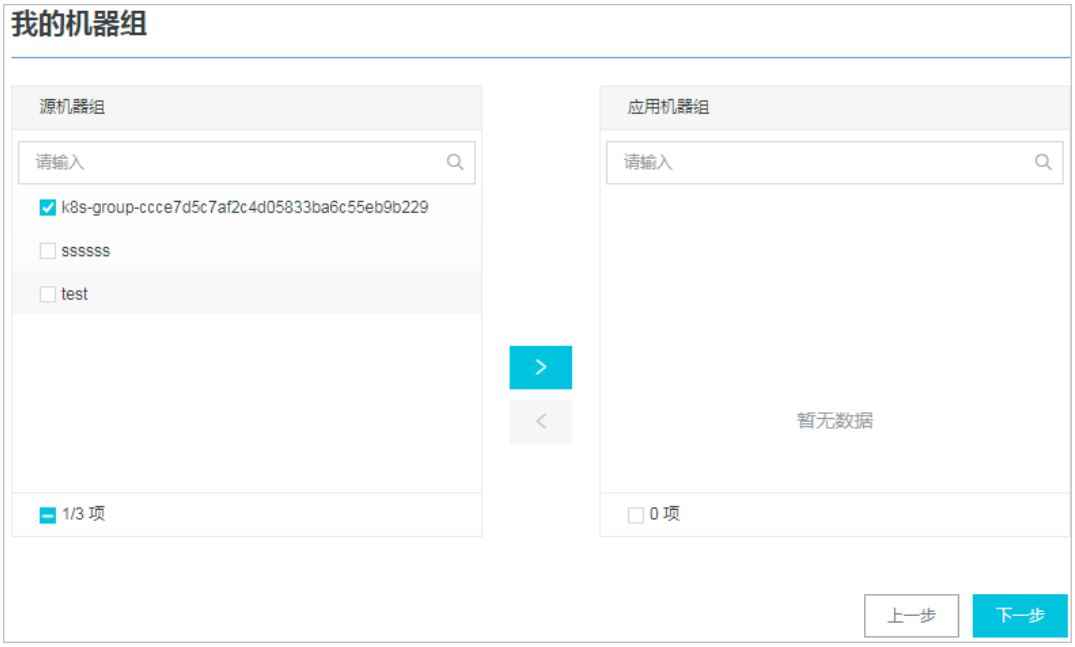
   如果您是通过日志库下的**数据接入**后的加号图标进入采集配置流程，系统会直接跳过该步骤。
4. 创建机器组，单击**下一步**。

   在创建机器组之前，您需要首先确认已经安装了Logtail。

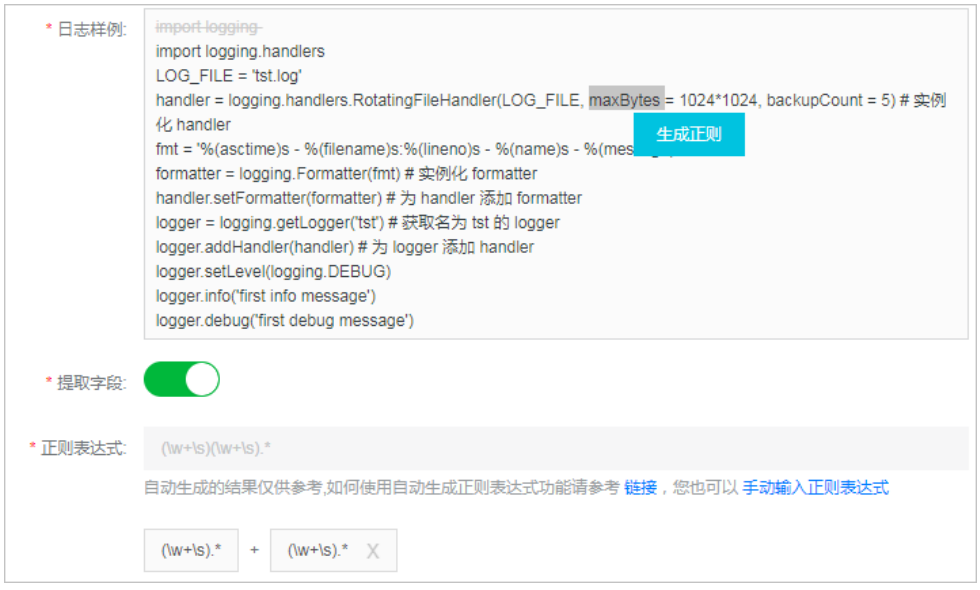
   请根据界面提示进行安装。更多信息，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。

   安装完Logtail后，单击**确认安装完毕**创建机器组。如果您之前已经创建好机器组，也可以直接单击**使用现有机器组**。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **重要**
>
> 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为**FAIL**，您可单击**自动重试**。如果显示**FAIL**，请参见Logtail机器组无心跳处理。



6. 设置Logtail配置。
   i. **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**。
   ii. **开启单行模式**。
   iii. **输入日志样例**。
   iv. **开启提取字段**。
   v. **配置正则表达式**。
      a. 通过划选生成正则表达式
      如果自动生成的正则表达式不匹配您的日志样例，可以通过划选的方式生成正则表达式。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。请在**日志样例**中划选日志字段内容，并单击**正则**。**正则表达式**一栏中会出现划选内容的正则式。您可以通过多次划选生成日志样例的完整正则表达式。



      b. 调整正则表达式

      鉴于实际的日志数据格式可能会有细微变动，单击**手动输入正则表达式**，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。

      c. 验证正则表达式

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

vi. **确认日志内容抽取结果。**
查看日志字段的解析结果，并为日志内容抽取结果填写对应的Key。
分别为日志字段提取结果取一个有意义的字段名称，例如时间字段的命名为time。如果不使用系统时间，您必须指定Value为时间的字段，并将其Key命名为time。

7. **可选：配置高级选项，配置完成后，单击下一步。**
请根据您的需求选择高级配置。如没有特殊需求，建议保持默认配置。

配置项	详情
启用插件处理	开启后，支持通过插件实现对文本日志的多样处理。 **说明** 打开**启用插件处理**开关后，上传原始日志、时区属性、丢弃解析失败日志、过滤器配置、接受部分字段（分隔符模式）等功能不可用。
上传原始日志	打开**上传原始日志**开关后，原始日志将作为**__raw__**字段的值与解析过的日志一起上传到日志服务。
Topic生成方式	设置Topic生成方式。 ◦ **空-不生成Topic**：默认选项，表示设置Topic为空字符串，在查询日志时不需要输入Topic即可查询。 ◦ **机器组Topic属性**：设置为机器组Topic属性，用于明确区分不同服务器产生的日志数据。 ◦ **文件路径正则**：设置为文件路径正则，则需要设置**自定义正则**，用正则表达式从路径里提取一部分内容作为Topic。用于区分不同用户或实例产生的日志数据。
自定义正则	如您选择了**文件路径正则**方式生成Topic，需要在此处填写您的自定义正则式。
日志文件编码	设置日志文件编码格式，支持如下： ◦ utf8：指定使用UTF-8编码。 ◦ gbk：指定使用GBK编码。
时区属性	设置采集日志时，日志时间的时区属性。 ◦ 机器时区：默认为机器所在时区。 ◦ 自定义时区：手动选择时区。

超时属性	如果一个日志文件在指定时间内没有任何更新，则认为该文件已超时。 ◦ 永不超时：指定持续监控所有日志文件，永不超时。 ◦ 30分钟超时：如日志文件在30分钟内没有更新，则认为已超时，并不再监控该文件。 选择**30分钟超时**时，还需设置**最大超时目录深度**，范围为1~3。
过滤器配置	日志只有**完全符合**过滤器中的条件才会被收集。 例如： ◦ 满足条件即采集，例如设置**Key**为**level**，**Regex**为**WARNING|ERROR**，表示只采集level为WARNING或ERROR类型的日志。 ◦ 过滤不符合条件的日志。 ▪ 设置**Key**为**level**，**Regex**为**^(?!.*(INFO|DEBUG)).***，表示不采集level为INFO或DEBUG类型的日志。 ▪ 设置**Key**为**url**，**Regex**为**.*^(?!.*(healthcheck)).***，表示不采集URL中带有healthcheck的日志。例如Key为url，Value为/inner/healthcheck/jiankong.html的日志将不会被采集。

8. **查询和分析配置，包括配置索引。配置完成后，单击下一步。**

   日志服务默认开启全文索引。您也可以根据采集到的日志，手动或者自动设置字段索引。更多信息，请参见开启并配置索引。

   > **说明**
   > ◦ 如果您要查询分析日志，那么全文索引和字段索引属性必须至少启用一种。同时启用时，以字段索引为准。
   > ◦ 索引类型为long、double时，大小写敏感和分词符属性无效。

配置完成后即可开始规范收集Python日志。

3.2.3.3. Node.js日志

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

配置示例如下：

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

日志格式

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

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

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

通过Logtail收集Node.js日志

配置Logtail收集Node.js日志的详细操作步骤请参见Python日志，根据您的网络部署和实际情况选择对应配置。

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

常见的Node.js日志及其正则表达式：

- Node.js日志示例1
  - 日志示例

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

  - 正则表达式

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

  - 提取字段

    `time`、`level`、`loggerName`和`message`。
- Node.js日志示例2
  - 日志示例

    ```
    [2016-01-31 12:02:25.844] [INFO] access - 42.120.73.203 - - "GET /user/projects/ali_sls_log?ignoreEr
    ror=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`。

3.2.3.4. WordPress日志

本文档为您展示WordPress的日志格式和提取结果。

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 Geck
o) Chrome/47.0.2526.106 Safari/537.36"
```

配置Logtail收集WordPress日志

WordPress日志的Logtail收集配置参数：

- 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	10.10.10.1
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

3.2.3.5. Unity3D日志

日志服务支持 通过web tracking功能非常方便的收集Unity3D的日志，下面以收集Unity Debug.Log为例，介绍如何将Unity日志收集到日志服务中。

背景信息

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。

操作步骤

1. 开通web tracking功能。
   开通方法请参见WebTracking。
2. 创建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请参见Project概览。

```
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;
    }
}
```

上面的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. 在控制台查看日志。

   上述步骤做完之后，运行 Unity 程序，就可以在日志服务的控制台看到您发送的日志了。

   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异常处理API、其他各种日志服务API使您可以非常方便的收集客户端的设备信息。

4.查询与分析

4.1. 简介

日志服务提供大规模日志实时查询与分析能力（LogSearch/Analytics），未开启索引时，原始数据可以根据Shard进行类似Kafka的顺序消费；开启索引后，除了支持顺序消费外，还可以对日志数据进行统计与查询。

功能优势

- 实时：写入后可以立即被分析。
- 快速：
  - 查询：一秒内查询（5个条件）可处理10亿级数据。
  - 分析：一秒内分析（5个维度聚合+GroupBy）可聚合亿级别数据。
- 灵活：可以改变任意查询和分析条件，实时获取结果。
- 生态丰富：除控制台提供的报表、仪表盘、快速分析等功能外，还可以与Grafana、DataV、Jaeger等产品无缝对接，并支持Restful API，JDBC等协议。

索引

日志服务中的索引是对日志数据一列或多列的值进行排序的一种结构，使用索引可快速访问日志服务采集到的日志数据。使用日志服务查询与分析功能之前，必须采集到日志数据，并对日志数据开启并配置索引。

日志服务索引包括**全文索引**和**指定字段查询**。

- **全文索引**：对日志全文内容开启索引，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。
- **指定字段查询**：为特定的Key配置索引，配置字段索引后，可以通过查询特定Key的内容，缩小查询范围。

其中，在**指定字段查询**中，需要指定字段的数据类型。当前，日志服务支持的字段类型包括：text、json、long和double。关于索引数据类型的更多信息请参见简介。

查询方式

- 控制台查询

  在日志服务控制台查询页面指定查询时间段和查询语句进行查询。详细说明和查询语法请参见查询日志和查询语法。

- API查询

  通过日志服务API中的接口GetLogs和GetHistograms接口可以查询日志数据。

> **说明** 查询日志前，请确认您已采集到日志数据且已开启并配置索引。

查询分析语句格式

对采集到的日志数据进行实时查询分析时，需要输入查询分析语句（Query）。由查询语句（Search）和分析语句（Analytics）两个部分组成，查询和分析语句之间通过 | 进行分割。

```
$Search |$Analytics
```

语句类型	是否可选	说明
查询语句（Search）	可选	查询语句，可以包括关键词、模糊查询关键字、数值、区间范围和组合条件。 如果为空或"*"，表示针对当前时间段所有数据不设置任何过滤条件，即返回所有数据。详细说明请参见查询语法。
分析语句（Analytics）	可选	对查询结果或全量数据进行计算和统计。 如果为空，表示只返回查询结果，不需要做统计分析。详细说明请参见实时分析简介。

注意事项

如果您需要检索的日志数据量很大，例如查询时间跨度非常长，其中数据量在百亿以上时，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。

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。

4.2. 实时分析简介

日志服务提供类似于SQL的聚合计算功能，该功能结合了查询功能和SQL计算功能，对查询结果进行计算。

语法示例：

```
status>200 |select avg(latency),max(latency) ,count(1) as c GROUP BY  method  ORDER BY c DESC  LIMIT 20
```

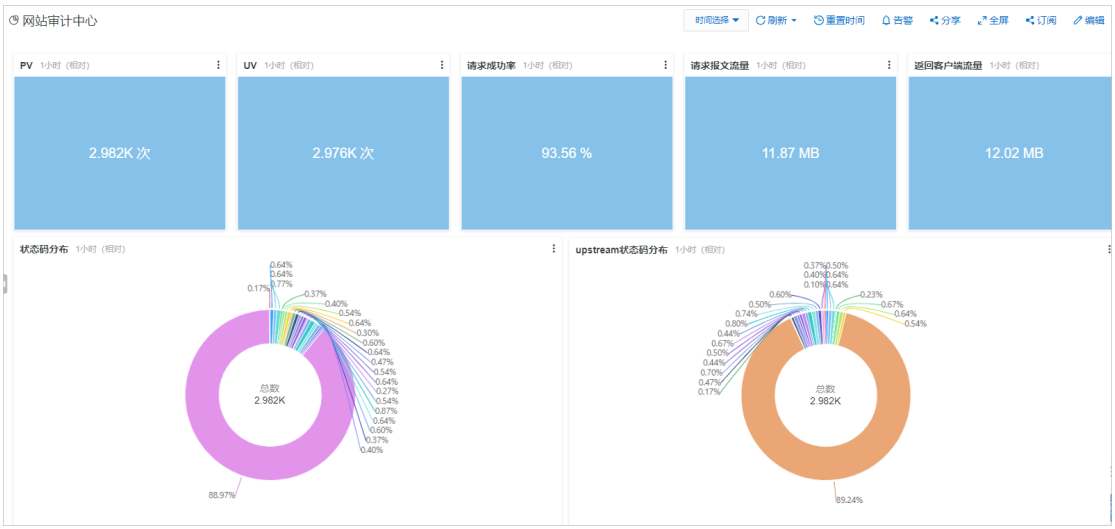
基本语法：

```
[search query] | [sql query]
```

search条件和计算条件以 `|` 分割，表示以search query从日志中过滤出需要的日志，并对这些日志进行SQL query计算。search query的语法为日志服务专有语法，参见查询语法。

效果展示

图 1. 效果展示



交互分析、仪表盘、Grafana、Datav等更多Demo请单击DEMO查看。

前提条件

要使用分析统计功能，必须在**查询分析属性设置**中打开对应字段的**开启统计**开关，详情请参见开启并配置索引。

- 如果不开启统计，默认只提供每个shard最多1万行数据的计算功能，而且计算延时比较高。
- 开启后可以提供秒级别快速分析。
- 开启后只对新数据生效。
- 开启统计后不会产生额外费用。

支持的SQL语法

日志服务支持以下SQL语法，详细内容请单击链接查看。

- SELECT聚合计算函数：
  - 通用聚合函数
  - 安全检测函数
  - Map映射函数
  - 估算函数
  - 数学统计函数
  - 数学计算函数

- 字符串函数
- 日期和时间函数
- URL函数
- 正则式函数
- JSON函数
- 类型转换函数
- IP地理函数
- 数组
- 二进制字符串函数
- 位运算
- 同比和环比函数
- 比较函数和运算符
- Lambda函数
- 逻辑函数
- 空间几何函数
- 地理函数
- 机器学习函数
- 电话号码函数

- GROUP BY 语法
- 窗口函数
- HAVING语法
- ORDER BY语法
- LIMIT语法
- CASE WHEN和IF分支语法
- unnest语法
- 列的别名
- 嵌套子查询

语法结构

SQL语法结构如下：

- SQL语句中不需要填写from子句和where子句， 默认from表示从当前Logstore的数据中查询，where条件为search query。
- 支持的子句包括SELECT、GROUP BY、ORDER BY [ASC,DESC]、LIMIT、HAVING。

> **说明** 默认情况下返回前100个结果，如要返回更多请加上limit n，例如 `* | select count(1) as c, ip group by ip order by c desc limit 100` 。

内置字段

日志服务内置了一些字段供统计，当用户配置了任何一个有效列后，就会自动加上这些内置字段。

字段名	类型	含义
`__time__`	bigint	日志的时间。
`__source__`	varchar	日志来源IP。在搜索时，该字段是source，在SQL中才会带上前后各两个下划线。
`__topic__`	varchar	日志的Topic。

限制说明

- 每个Project中，最大并发数为15个。
- 字段的值的最大长度为2048，超过后会截断。
- 默认最多返回100行数据，不支持翻页。若需要返回更多数据，请使用LIMIT语法。

示例

统计每小时的PV、UV和最高延时对应的用户请求。

```
*|select date_trunc('hour',from_unixtime(__time__)) as time,
     count(1) as pv,
     approx_distinct(userid)  as uv,
     max_by(url,latency) as top_latency_url,
     max(latency,10) as top_10_latency
     group by  1
     order by time
```

4.3. 开启并配置索引

使用日志服务查询分析功能之前，请先开启并配置索引。

背景信息

开启并配置索引后，您才可以查询配置索引后采集到的日志数据。请根据您的日志字段内容和查询需求，合理配置索引。

> **说明** 开启索引后会产生索引流量以及索引对应存储的空间。

采集日志时日志服务会自动将日志来源、时间等信息以Key-Value对的形式添加到日志中，这些字段是日志服务的保留字段。当开启并配置索引时，自动开启这些字段的索引和统计功能。 表 1. 日志服务保留字段

保留字段名称	说明
`__topic__`	日志主题（Topic）。如果您设置了日志主题，日志服务会自动为您的日志添加日志主题字段，Key为 `__topic__` ，Value为您的主题内容。
`__source__`	日志来源。该字段表示这条日志的来源设备。
`__time__`	使用SDK写入日志数据时指定的日志时间。

> **说明** `__topic__` 和 `__source__` 的索引分词为空，表示在查询这两个字段时，查询关键字必须完全匹配。

操作步骤

1. 登录日志服务控制台。
2. 单击目标Project。
3. 选择目标日志库名称右侧的 > **查询分析**。
4. 单击右上角的**开启索引**。

   > **说明** 若您之前已创建过索引，可以单击**查询分析属性** > **设置**修改索引。

5. 配置索引。

   > **说明** 若同时配置了全文索引和指定字段查询，以指定字段查询设置为准。

   表 2. 索引类型

索引类型	说明
全文索引	以文本形式为所有的字段创建索引，Key和Value都是普通文本，都可以进行查询。long类型字段在查询具体Value时需要指定对应的Key名称，其他类型字段在查询时不必指定Key的名称。
指定字段查询	配置字段索引后，在查询时需要指定Key的名称。当某个字段配置了指定字段查询时，则该字段的全文索引不生效。 字段可以配置为多种数据类型，包括： ◦ 文本类型 ◦ JSON类型 ◦ 数值类型（long和double）

- 配置全文索引。
  配置针对全文内容的索引，查询日志时默认查询所有Key对应的内容。

配置项	说明	示例
全文索引	对日志全文内容开启索引，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。	-
大小写敏感	查询时英文字母是否区分大小写。 ■ 关闭后，表示日志查询不区分大小写。例如某个日志含有internalError，那么使用关键字"INTERNALERROR"和"internalerror"都能查到该日志。 ■ 开启后，表示日志查询区分大小写。例如某个日志含有internalError，那么只能使用internalError才能查询到该日志。	-
包含中文	设置是否区分中英文。 ■ 开启后，如果日志中包含中文，则对中文按照中文语法进行分词，对英文按照分词符进行分词。 ■ 关闭后，对所有内容按照分词符进行分词。	-
分词符	根据指定分词符，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` 。设置分词符为逗号"，"、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"。	`, '";=()[]{}?@&<>/:\n\t`

- 配置字段索引。
  为特定的字段配置索引，配置字段索引后您可以通过查询特定字段的内容，缩小查询范围。

配置项	说明	示例
字段名称	指定日志字段名称。 **说明** ■ 如果配置公网IP、Unix时间戳等tag字段的索引，请将**字段名称**配置为 `__tag__:key` 形式，例如 `__tag__:__receive_time__` 。 ■ tag字段不支持数值类型索引，请将所有tag字段的索引**类型**配置为text（文本类型）。	`_address_`
类型	日志字段内容的数据类型，包括： ■ **text**：指定日志字段内容为文本类型。 ■ **long**：指定日志字段内容为整数，需要按照数值范围进行查询。 ■ **double**：指定日志字段内容为小数，需要按照数值范围进行查询。 ■ **json**：指定日志字段内容为JSON类型。 **说明** 数值类型（long和double类型）不支持设置**大小写敏感**、**包含中文**和**分词符**。	-
别名	列的别名。 别名仅用于SQL统计，在底层存储和搜索时仍需要使用原始名称。详细说明请参见列的别名。	`address`
大小写敏感	查询时英文字母是否区分大小写。 ■ 关闭后，表示日志查询不区分大小写。例如某个日志含有internalError，那么使用关键字"INTERNALERROR"和"internalerror"都能查到该日志。 ■ 开启后，表示日志查询区分大小写。例如某个日志含有internalError，那么只能使用"internalError"才能查询到该日志。	-

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` 。设置分词符为逗号“，”、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”。	`, '";=()[]{}?@&<>/:\n\t`
包含中文	设置是否区分中英文。 ■ 开启后，如果日志中包含中文，则对中文按照中文语法进行分词，对英文按照分词符进行分词。 ■ 关闭后，对所有内容按照分词符进行分词。	-
开启统计	是否开启统计分析功能。该功能默认开启。 开启之后，您可以结合查询语句和分析语句，对日志查询结果进行统计分析。	-

6. 单击**确定**完成配置。

> **说明**
> - 索引配置在1分钟之内生效。
> - 开启或修改索引后，新的索引配置只对新写入的数据生效。

4.4. 查询日志

开启并配置索引后，可以在查询页面对采集到的日志进行实时查询与分析。

前提条件

- 已采集到日志数据。
- 已开启并配置索引。

操作步骤

1. 登录日志服务控制台。
2. 单击日志库名称后的图标，选择**查询分析**。
3. 在搜索框中输入查询分析语句。
   查询分析语句由查询语句和分析语句构成，格式为 `查询语句|分析语句` 。详细说明请参见查询分析语句格式。
4. 在页面右上角单击**15分钟（相对）**，设置查询的时间范围。
   您可以选择相对时间、整点时间和自定义时间范围。

> **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

5. **单击查询/分析，查看搜索结果。**
   日志服务为您提供日志分布直方图、原始日志和统计图表形式的查询分析结果。

   > **说明** 默认返回100个结果，如果您需要返回更多，请使用LIMIT语法。

   - 日志分布直方图
     日志分布直方图主要展示查询到的日志在时间上分布。
     - 鼠标指向绿色的数据块，可以查看该数据块代表的时间范围和日志命中次数。
     - 单击数据块，可以查看更细时间粒度的日志分布，同时**原始日志**页签中也会同步展示指定时间范围内的日志查询结果。

   - 原始日志
     原始日志页签展示当前查询结果，也就是当前查询条件命中的日志。
     - 快速分析：快速分析功能用于快速分析某一字段在一段时间内的分布情况，详细说明请查看快速分析。
     - 下载日志：单击页签右上角的下载图标，选择下载的范围，并单击**确定**。
     - 设置列：单击页签右上角的**列设置**，勾选字段并单击**添加**，页签中会新增选中字段的列，其中列名称为字段名，内容为每条日志的字段值。

       > **说明** **内容**列需要被选中，页签中才会出现日志内容一列。

- 设置内容列显示：字段内容如果超出3000字符，会默认折叠处理，并在Key值前显示提醒信息“该字段过长，已做折叠处理”。单击页签右上角的**内容列显示**，设置**Key-Value对排列**和**长字符折叠**。

  > **说明** 展开至10000个字符以上时，第10000以外的字符的将做降级处理，降级处理的字符不提供分词的功能。

配置		说明
Key-Value对排列		您可以设置Key-Value对之间为**换行**显示或**整行**显示。
长字符折叠	**Key**	当某一Value值超过3000字符时，默认为Value值设置折叠显示，如果日志中不存在过长的Value值，则此处为空。 Key为过长而被折叠的Value对应的Key。
	状态	是否开启Value值折叠。默认为开启状态。 ■ **开启**：表示Key-Value对里的Value值长度超出**折叠步长**时，会自动对字符进行折叠。单击字符末尾展开按钮可以进行增量展开，增量为折叠步长。 ■ **关闭**：表示超出**折叠步长**时不折叠。
	折叠步长	Value正常显示的最大长度，也是每次增量展开的长度。 单位为字符，取值范围为500~10000，默认为3000。

- 统计图表：
  如果在索引设置中开启了统计功能，且在搜索中使用查询分析语句，则可以在**统计图表**页签中查看分析结果。
  - 查看分析结果：日志服务为您提供表格、折线图、柱状图等多种类型的统计图表，您可以根据分析需求选用合适的图表类型展示分析结果。
  - 添加图表到仪表盘：仪表盘是日志服务提供的实时数据分析大盘。单击**添加到仪表盘**，将常用的查询语句以图表形式保存到仪表盘中。
  - 设置下钻配置：下钻分析是在分析时加深维度，对数据进行层层深入的查看。设置下钻配置并将图表添加到仪表盘，在仪表盘中单击图表值可以获取更深维度的分析结果。详细说明请查看下钻分析。

另外，在查询页面右上角单击**另存为快速查询**和**另存为告警**，可以使用日志服务的快速查询和告警功能。

4.5. 导出日志

日志服务支持通过下载的方式将日志数据导出到本地，您可以导出本页日志（CSV格式）到本地。

操作步骤

1. 登录日志服务控制台。
2. 单击目标Project名称。
3. 单击日志库名称后的图标，选择**查询分析**。
4. 单击**原始日志**页签右侧的下载图标打开日志下载对话框。
5. 在**日志下载**对话框中选择**下载本页日志**。
6. 单击**确定**将本页面的日志以CSV格式保存到本地。

4.6. 索引数据类型

4.6.1. 简介

日志服务支持对采集到的日志设置全文或字段索引。设置全文索引后，Value为整条日志；设置字段索引后，每个Key都可以设置数据类型。

数据类型

目前支持的索引数据类型如下：

查询类别	索引数据类型	说明	查询示例

基础查询	TEXT	文本类型，可以进行关键词+模糊匹配。	`uri:"login*" method:"post"`
	Long	数值类型，支持区间查询。	`status>200, status in [200, 500]`
	Double	带浮点数数值类型。	`price>28.95, t in [20.0, 37]`
组合查询	JSON	内容为JSON字段，默认为Text类型，支持嵌套模式。可以通过 a.b等路径格式给a层下b元素设置（Text、Long、Double）类型索引，设置后的字段类型以设置为主。	`level0.key>29.95 level0.key2:"action"`
	文本	整条日志当做文本进行查询。	`error and "login fail"`

查询示例

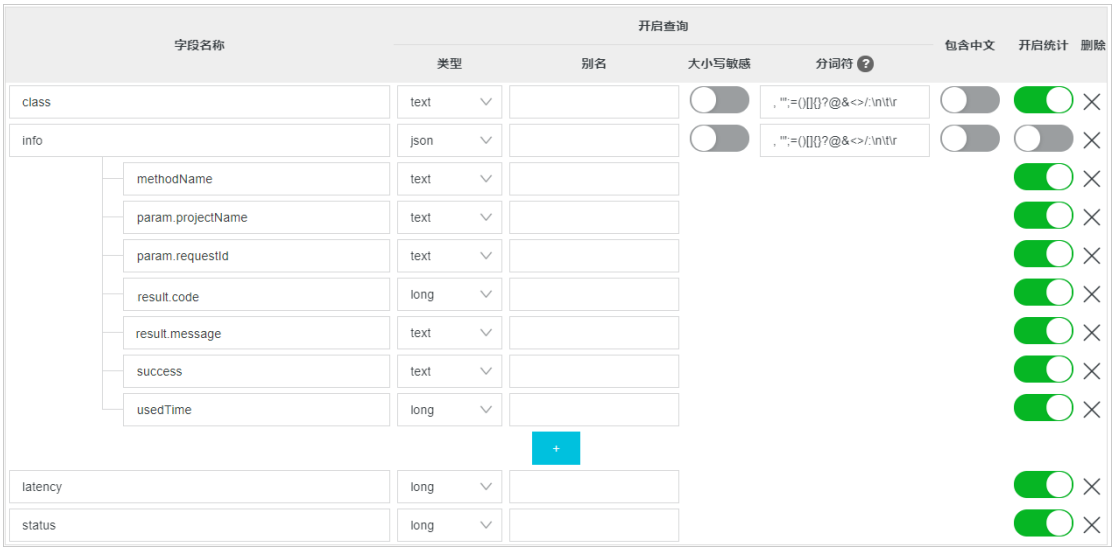
以下一条日志除时间外，还包含4个键值：

序号	Key	类型
0	time	-
1	class	text
2	status	long
3	latency	double
4	message	json

```
0. time:2018-01-01 12:00:00
  1. class:central-log
  2. status:200
  3. latency:68.75
  4. message:
  {
      "methodName": "getProjectInfo",
      "success": true,
      "remoteAddress": "1.1.1.1:11111",
      "usedTime": 48,
      "param": {
              "projectName": "ali-log-test-project",
              "requestId": "d3f0c96a-51b0-4166-a850-f4175dde7323"
      },
      "result": {
          "message": "successful",
          "code": "200",
          "data": {
              "clusterRegion": "ap-southeast-1",
              "ProjectName": "ali-log-test-project",
              "CreateTime": "2017-06-08 20:22:41"
          },
          "success": true
      }
  }
```

索引设置如下：

图 1. 索引设置



其中：

- ①表示可查询json字段中所有string和bool数据。
- ②表示可查询long类型数据。
- ③表示配置的字段可进行SQL分析。

示例

1. 查询string、bool类型
   - json内字段无需配置。
   - json map、array自动展开，支持多层嵌套，每一层以 `.` 进行分割。

```
class : cental*
message.traceInfo.requestId : 92.137_1518139699935_5599
message.param.projectName : ali-log-test-project
message.success : true
```

2. 查询Double、Long类型

   需要对json内字段独立配置，字段必须不在array。

```
latency>40
message.usedTime > 40
```

3. 组合查询

```
class : cental* and message.usedTime > 40 not message.param.projectName:ali-log-test-project
```

4.6.2. 文本类型

本文档向您介绍文本类型数据查询时的配置说明。

和搜索引擎类似，文本类（Text）数据查询基于词（Term）的命中，因此需要配置分词符、大小写敏感，包含中文（中文分词）选项。

配置说明

- 大小写敏感
  原始日志查询时是否区分大小写。例如原始日志为 `internalError` ：
  - false（不区分），即查询关键字 `INTERNALERROR` 和 `internalerror` 都能查询到样例日志。
  - true（区分），只能通过关键字 `internalError` 查询到样例日志。
- 分词符

  原始日志内容根据分词符可以将日志内容切分成多个关键词。

  例如我们要查询如下日志内容：

```
/url/pic/abc.gif
```

- 不设置任何分词符，整个字符串会作为一个独立单词 `/url/pic/abc.gif` ，只有通过该完整字符串，或通过模糊查询 `/url/pic/*` 才能找到。
- 如果设置分词符为 `/` ，则原始日志被切分为 `url` 、 `pic` 和 `abc.gif` 三个单词，可以使用任意一个单词或单词模糊查询都可以找到该日志，例如 `url` 、 `abc.gif` 或 `pi*` ，也可以使用 `/url/pic/abc.gif` 进行查询（查询时会被拆分为 `url and pic and abc.gif` 三个条件）。
- 如果设置分词符为 `/.` ，则原始日志被切分为 `url` 、 `pic` 、 `abc` 和 `gif` 四个单词。

> **说明** 通过设置合理的分词符，可以放宽查询的范围。

- 包含中文
  如果日志中包含中文，需要打开中文分词。例如如下日志内容：

```
buyer:用户小李飞刀lee
```

  默认分词符为":"，则原始日志会被拆分为 `buyer` 、 `用户小李飞刀lee` 这两个单词，如果搜索 `用户` ，则不会返回 `lee` ，如果开启包含中文选项后，日志服务后台分词器会智能去理解中文含义，并将日志拆分为 `buyer` 、 `用户` 、 `小李` 、 `飞刀` 和 `lee` 五个单词，无论使用 `飞刀` 或 `小李飞刀` （会被解析为： `小李 and 飞刀` ）都可以查找到日志。

> **说明** 中文分词对写入速度会有一定影响，请根据需求谨慎设置。

- 全文索引
  全文查询（索引）默认会将整条日志（除Time以外所有字段、包括Key）作为文本类型，全文查询默认不需要指定key。例如对以下由4个字段组成的日志（time/status/level/message）。

```
[20180102 12:00:00] 200,error,some thing is error in this field
```

  - `time:2018-01-02 12:00:00`
  - `level:"error"`
  - `status:200`
  - `message:"some thing is error in this field"`

> **说明**
> - 全文检索时不需要输入前缀，在检索过程中搜索error时（level和message两个字段中error都会被命中）。
> - 全文检索需要设置分词符，例如当设置分词符为 时，可以 `status:200` 作为一个短语；如果分词符为 `:` 时， `status` 和 `200` 分别会作为2个独立短语。
> - 数值类会被作为文本处理，例如200可以检索到该日志，时间字段（time）不会被作为文本处理。
> - 当输入Key时整条日志也会被命中，例如 `status` 。

4.6.3. 数值类型

在配置索引时，您可以将字段配置为数值类型，并通过数值范围查询键值。

配置说明

支持类型： `long` （长整数）或者 `double` （小数），当设置为数值类型后对于该键的查询只能通过数值范围。

查询示例

查询键值范围为(1000 2000]的longkey，可以使用以下查询方式：

- 数值类查询语法，例如：

```
longKey > 1000 and longKey <= 2000
```

- 也可以使用区间查询语法，例如：

```
longKey in (1000 2000]
```

更多语法请参见查询语法。

4.6.4. JSON类型

索引数据类型可设置为JSON类型，支持JSON格式日志的查询和分析功能。

JSON是由文本、布尔、数值、数组（Array）和图（Map）构成的组合类型数据。JSON数据作为一种通用类型的数据类型，其自解析、灵活的特性，使其能够很好满足复杂场景下数据的记录需求，在很多日志内容中格式不固定的部分往往都是以JSON的形式进行记录，例如将一次http请求的request参数和response内容以JSON的形式记录在一条日志中。

日志服务支持在索引中将字段设置为JSON类型，支持JSON格式日志的查询和分析。

配置说明

- 支持json格式解析，所有text、bool类型自动索引

```
json_string.key_map.key_text : test_value
json_string.key_map.key_bool : true
```

- 非json array中的double、long类型数据，可通过配置指定json路径后进行查询

```
配置key_map.key_long这个字段的类型为long
查询 : json_string.key_map.key_long > 50
```

- 非json array中的text、double、long类型字段，可开启“统计分析”功能，进行sql分析

```
json_string.key_map.key_long > 10 | select count(*) as c ,
    "json_string.key_map.key_text" group by
    "json_string.key_map.key_text"
```

> **说明**
> - 不支持json object、json array类型。
> - 字段不能在json array中。
> - bool类型字段可以转成text类型。
> - 查询分析时，JSON类型字段需要用双引号括起来。

- 支持非完全合法json数据解析

  日志服务会尽可能解析有效内容，直到遇到非法部分结束。

  例如以下示例在key_3之后的数据被截断丢失，对于这种缺失的日志，日志服务可正确解析到 json_string.key_map.key_2 这个字段。

```
"json_string":
{
     "key_1" :  "value_1",
     "key_map" :
      {
             "key_2" : "value_2",
             "key_3" : "valu
```

查询语法

指定Key查询需要加上JSON中父路径的前缀，文本、数值类查询语法与其他类型相同，详情请参见查询语法。

查询示例

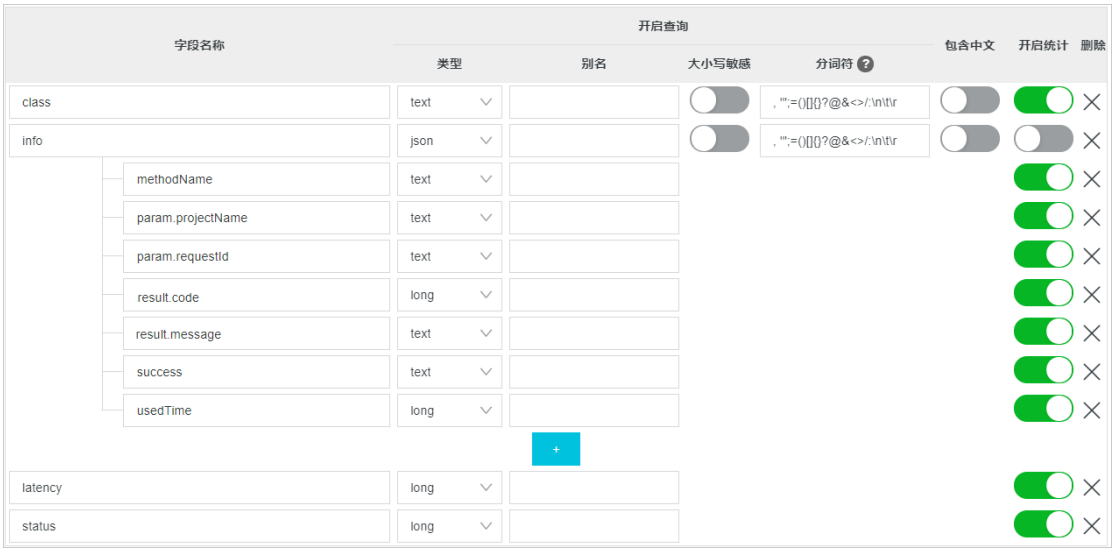
以下一条日志除时间外，还包含4个键值，其中 `message` 字段是json格式。

序号	Key	类型
0	time	-
1	class	text
2	status	long
3	latency	double
4	message	json

```
0. time:2018-01-01 12:00:00
  1. class:central-log
  2. status:200
  3. latency:68.75
  4. message:
  {
      "methodName": "getProjectInfo",
      "success": true,
      "remoteAddress": "1.1.1.1:11111",
      "usedTime": 48,
      "param": {
              "projectName": "ali-log-test-project",
              "requestId": "d3f0c96a-51b0-4166-a850-f4175dde7323"
      },
      "result": {
          "message": "successful",
          "code": "200",
          "data": {
              "clusterRegion": "ap-southeast-1",
              "ProjectName": "ali-log-test-project",
              "CreateTime": "2017-06-08 20:22:41"
          },
          "success": true
      }
  }
```

索引设置如下：

图 1. 索引设置



其中：

- ①表示可查询json字段中所有string和bool数据。
- ②表示可查询long类型数据。
- ③表示配置的字段可进行SQL分析。

示例

1. 查询string、bool类型

> **说明**
> - json内字段无需配置。
> - json map、array自动展开，支持多层嵌套，每一层以 `.` 进行分割。

```
message.traceInfo.requestId : 92.137_1518139699935_5599
message.param.projectName : ali-log-test-project
message.success : true
message.result.data.ProjectStatus : Normal
```

2. 查询Double、Long类型

> **说明** 需要对json内字段独立配置，字段必须不在array。

```
message.usedTime > 40
```

3. Sql 统计分析

> **说明**
> - 需要对json内字段独立配置，字段必须不在array。
> - 查询字段需要使用引号，或者设置别名。

```
* | select avg("message.usedTime") as avg_time ,
"message.methodName"  group by "message.methodName"
```

4.7. 查询语法与功能

4.7.1. 查询语法

为了能够帮助您更有效地查询日志，日志服务提供一套查询语法用于设置查询条件。

查询方式

开启并配置索引之后，在日志查询界面输入查询分析语句即可查询日志。

日志查询语句是查询分析语句的前半部分，用来指定日志查询时的过滤规则，返回符合条件的日志数据。查询语句支持全文查询和字段查询。

- 全文查询

  全文查询时，将整条日志作为一个特殊的 Key-Value 对，Value 为全部的日志内容。全文查询表示在日志内容中查询关键字，即指定查询条件为包含或不包含某个关键字，满足查询条件的日志会作为结果返回。

  全文查询可以分为普通全文查询、短语查询、模糊查询。

  - 普通全文查询：指定关键字和规则，包含该关键字并符合规则的日志会作为结果返回。

    例如 `a and b` 表示查询同时包含关键字 `a` 和 `b` 的日志。

  - 短语查询：如果需要查询的短语中包含空格，可以将短语用双引号（“”）包裹，表示将双引号中的内容作为一个完整的关键字查询。

    例如 `"http error"` 表示查询包含关键字 `http error` 的日志。

  - 模糊查询：指定一个64个字符以内的词，在词的中间或者末尾加上模糊查询关键字，即 `*` 和 `?` ，日志服务会在所有日志中为您查询到符合条件的100个词，返回包含这100个词并满足查询条件的所有日志。

    例如 `addr?` 表示在所有日志中查找以 `addr` 开头的100个词，并返回包含这些词的日志。

    限制说明：

    - 查询时必须指定前缀，即 `*` 和 `?` 不能出现在词的开头。
    - 指定的词越精确，查询结果越精确。
    - 查询的词超过64个字符，无法使用模糊查询。建议您把查询的词长度缩小到64个字符以内。
    - 模糊查询最多返回100个符合查询条件的日志。

- 字段查询

  为字段配置字段索引之后，可以指定字段名称和字段内容进行查询。对于 double 和 long 类型的字段，可以指定数值范围进行查询。例如设置字段查询语句为 `Latency>5000 and Method:Get* and not Status:200` ，表示查询 `Latency` 字段值大于5000、 `Method` 字段值以 `Get` 开头，且 `Status` 字段值不是200的日志。

  根据字段索引中设置的数据类型，您可以进行多种类型的基础查询和组合查询。字段查询示例请参见简介。

注意事项

- 同时配置全文查询和字段查询时，如果索引设置中两者的分词符不同，以字段查询设置为准。

- 只有将某些字段的数据类型设置为 double 或 long 后，才能通过数值范围查询这些字段的数据。若字段设置的数据类型不是 double 或 long，或者查询数值范围的语法错误，那么日志服务会将该查询条件解释成全文索引，这样查询到的结果可能与您期望的结果不同。
- 如果将某字段由文本类型改成数值类型，则修改索引之前采集到的数据只支持 `=` 查询。

运算符

查询语句支持如下运算符：

运算符	说明
and	双目运算符。格式为 `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。格式为 `query1 or query2` ，表示 `query1` 和 `query2` 查询结果的并集。
not	双目运算符。格式为 `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。
(,)	括号用于把一个或多个子查询合并成一个查询条件，用于提高括号内查询条件的优先级。
:	用于 key-value 对的查询。 `term1:term2` 构成一个 key-value 对。如果 key 或者 value 内有空格（）、冒号（:）等保留字符，需要用双引号 `""` 把整个 key 或者 value 包裹起来。
“	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，表示在前一个计算的基础上进行更多计算，例如 `query1 | select count(1)` 。
count	计数运算符，表示统计日志条数。
*	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。 说明 模糊查询最多返回100个符合关键词的日志。
?	模糊查询关键字，用于替代一个字符，例如 `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的词。
`__topic__`	查询某个 topic 下数据，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。仅支持小写字符in。例如 `latency in [100 200]` 或 `latency in (100 200]` 。

> **说明**
> - 除in运算符外的其他运算符不区分大小写。
> - 运算符的优先级由高到低排序为 `:` > `"` > `( )` > `and` > `not` > `or` 。
> - 日志服务保留以下运算符的使用权，如果您需要使用以下运算符作为查询关键字，请使用双引号包裹起来： `sort` 、 `asc` 、 `desc` 、 `group by` 、 `avg` `sum` 、 `min` 、 `max` 和 `limit` 。

查询示例

查询需求	例句
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 FILE 字段下，值为shen*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 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	`latency >=100 and latency < 200` 或 `latency in [100 200)`
查询 latency 大于100的所有请求	`latency > 100`
查询不包含爬虫的日志，并且 http_referer 中不包含 opx 的日志	`not spider not bot not http_referer:opx`
查询 cdnIP 字段不为空的日志	`not cdnIP:""`

查询 cdnIP 字段不存在的日志	`not cdnIP:*`
查询存在 cdnIP 字段的日志	`cdnIP:*`
查询指定 url	`* | select * where url = 'www.xxxxx.com'`

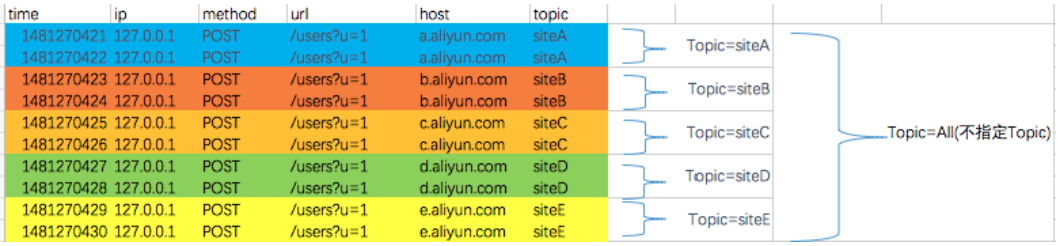
指定或跨 Topic（日志主题） 查询

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：

图 1. 日志Topic



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 意味着查询所有 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，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。

4.7.2. LiveTail

LiveTail是日志服务在控制台提供了日志数据实时监控的交互功能，帮助您实时监控日志内容、提取关键日志信息。

前提条件

- 已成功采集到日志数据后，才能使用LiveTail功能。
- LiveTail功能仅支持Logtail采集到的日志数据。

背景信息

在线上运维的场景中，往往需要对日志队列中进入的数据进行实时监控，从最新的日志数据中提取出关键的信息进而快速地分析出异常原因。在传统的运维方式中，如果需要对日志文件进行实时监控，需要到服务器上对日志文件执行命令 `tail -f` ，如果实时监控的日志信息不够直观，可以加上 `grep` 或者 `grep -v` 进行关键词过滤。日志服务在控制台提供了日志数据实时监控的交互功能LiveTail，针对线上日志进行实时监控分析，减轻运维压力。

功能优势

- 监控日志的实时信息，按关键词过滤日志数据。
- 结合采集配置，对采集的日志进行索引区分。
- 日志字段做分词处理，以便查询包含分词的上下文日志。
- 根据单条日志信息追踪到对应日志文件进行实时监控，无需连接线上机器。

使用LiveTail实时监控日志

1. 登录日志服务控制台。
2. 在Project中单击目标Project名称。
3. 单击日志库名称后的图标，选择**查询分析**。
4. **可选：**快捷开启LiveTail。

i. 在**原始日志**页签中，单击指定原始日志的序号右侧图标，并选择**LiveTail**。

系统为您自动开启LiveTail，并开始计时。其中**来源类型**、**机器名称**和**文件名称**已预设为指定原始日志的信息。
开启LiveTail后，Logtail采集到的日志数据会实时显示排列在页面中。最新的日志数据始终在页面底部，且滚动条默认在最下方，即显示最新数据。页面最多显示1000条数据，满1000条后页面自动刷新并重新填充日志数据。

ii. **可选：**您也可以在搜索框中输入关键词，包含关键词的日志才会显示在监控列表中。筛选包含关键词的日志，便于对特定日志进行实时内容监控。

5. 自定义设置LiveTail。

i. 在查询分析页面单击**LiveTail**页签。

ii. 配置LiveTail。

配置	是否必选	说明
来源类型	必选	日志的来源，包括： ■ 普通日志 ■ Kubernetes ■ Docker
机器名称	必选	日志来源服务器的名称。
文件名称	必选	日志文件的完整路径及文件名。
过滤条件	可选	关键词，设置关键词后，只有包含关键词的日志才会显示在实时监控窗口中。

iii. 单击**开启LiveTail**。
开启LiveTail后，Logtail采集到的日志数据会实时显示排列在页面中。最新的日志数据始终在页面底部，且滚动条默认在最下方，即显示最新数据。页面最多显示1000条数据，满1000条后页面自动刷新并重新填充日志数据。

6. 在实时监控日志的时候，如果某些日志数据可能存在异常需要分析的时候，单击**停止LiveTail**。

停止LiveTail后，LiveTail计时结束，不再实时更新日志数据。

针对监控日志时发现的异常，日志服务提供多种分析方式，详细信息请参见使用LiveTail分析日志。

使用LiveTail分析日志

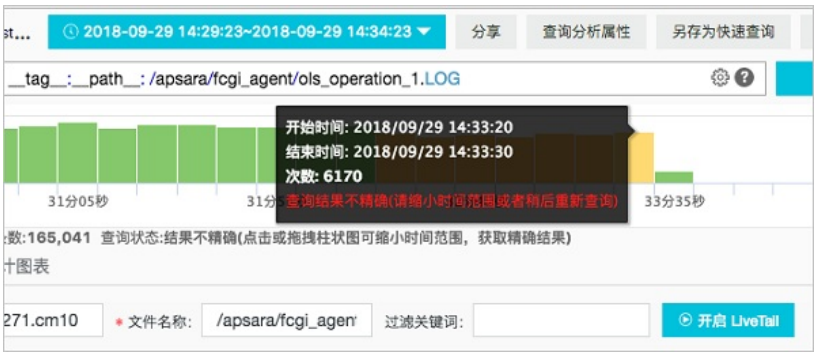
停止LiveTail之后，实时监控窗口不再更新显示日志内容，可以对监控过程中发现的问题进行进一步分析排查。

- 查看包含指定字段的日志内容。

  所有的字段都进行过分词处理，您可以在**LiveTail**页签中单击指定异常字段的value，页面会自动跳转到**原始日志**页签中，并会将异常字段的value作为关键词筛选出与该字段相关的所有日志内容。另外也可以对包含该关键字的日志进行上下文查询、查看统计图表等方式进行分析。

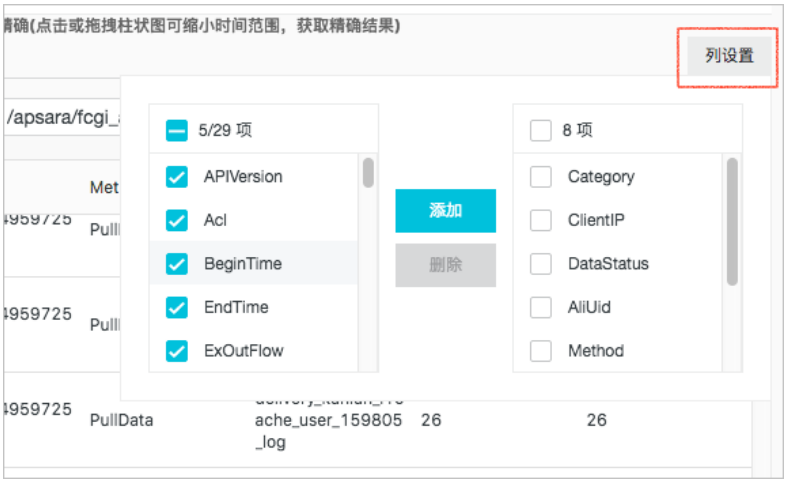
- 根据日志分布直方图（histogram）缩小查询的时间范围。

  LiveTail开启时，日志分布直方图也在进行同步更新。如果发现某个时段的日志分布有异常，例如日志数量显著增加时，可以单击该时段的绿色矩形缩小查询的时间范围。跳转后的原始日志内的时间轴与LiveTail内选择的时间轴是相关联的，可以查看在这段时间内所有的原始日志内容及详细的时间分布。



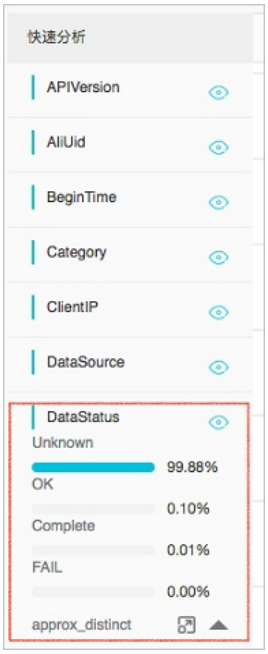
- 通过列设置强调关键信息。

  LiveTail页签中，单击日志列表右上角的**列设置**可以将指定字段单独选设置为一列，使该列的数据更加醒目。可以将需要重点关注的数据设为一列，便于查看和判断异常。



- 对日志数据进行快速分析。

  LiveTail页签中，单击日志列表左上角的箭头，可以展开快速分析区域。快速分析的时间区间是LiveTail开启到停止的时间段，分析的功能与原始日志内提供的快速分析相同。详细说明请参见快速分析。



4.7.3. 日志聚类

日志聚类，指采集文本日志时，将相似度高的数据聚合在一起，提取共同的日志Pattern，快速掌握日志全貌。

背景信息

日志服务提供日志聚类功能，支持多种格式的文本日志聚合，可应用于DevOps中的问题定位、异常检测、版本回归等运维动作，或应用于安全场景下的入侵检测等。您还可以将聚类结果以分析图表的形式保存在仪表盘中，实时查看聚类数据。

功能优势

- 支持任意格式日志：Log4J、Json、单行（syslog）。
- 亿级数据，秒级输出结果。
- 日志数据可以按任意pattern聚类。
- 按pattern聚类的数据可以根据签名反查原始数据。
- 不同时间段Pattern比较。
- 动态调整聚类精度。

计费标准

开启日志聚类功能后，索引总量会增加原始日志大小的10%。例如原始数据为100GB/天，开启该功能后，索引总量增加10GB。

原始日志大小	索引比例	日志聚类功能产生的索引量	索引总量
100GB	20%（20 GB）	100 * 10%	30GB
100GB	40%（40 GB）	100 * 10%	50GB
100GB	100%（100 GB）	100 * 10%	110GB

开启日志聚类功能

日志聚类功能默认为关闭状态，使用前请手动开启。

1. 登录日志服务控制台。
2. 在Project中单击目标Project名称。
3. 单击日志库名称后的图标，选择**查询分析**。
4. 设置索引。
   - 如果设置过索引，单击**查询分析属性 > 设置**。

- 如果没有设置过索引，单击**开启索引**。

5. 设置索引属性，并开启**日志聚类**的功能开关。然后单击**确定**完成设置。
   成功开启日志聚类功能后，日志服务会对采集到的日志数据进行自动聚类，您可以：
   - 查看聚类结果和原始日志
   - 调整聚类精度
   - 对比不同时间段的聚类日志数量

查看聚类结果和原始日志

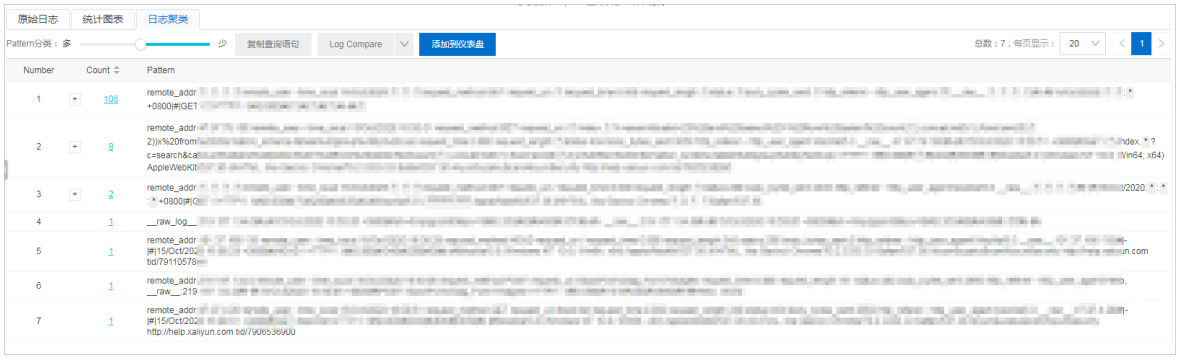
1. 在查询分析页面的查询框中输入查询语句，并单击**查询/分析**。
   可以通过关键字过滤日志，对包含或不包含指定关键字的日志数据进行自动聚类。

   > **说明** 不支持SQL语句，即不能对分析结果进行聚类。

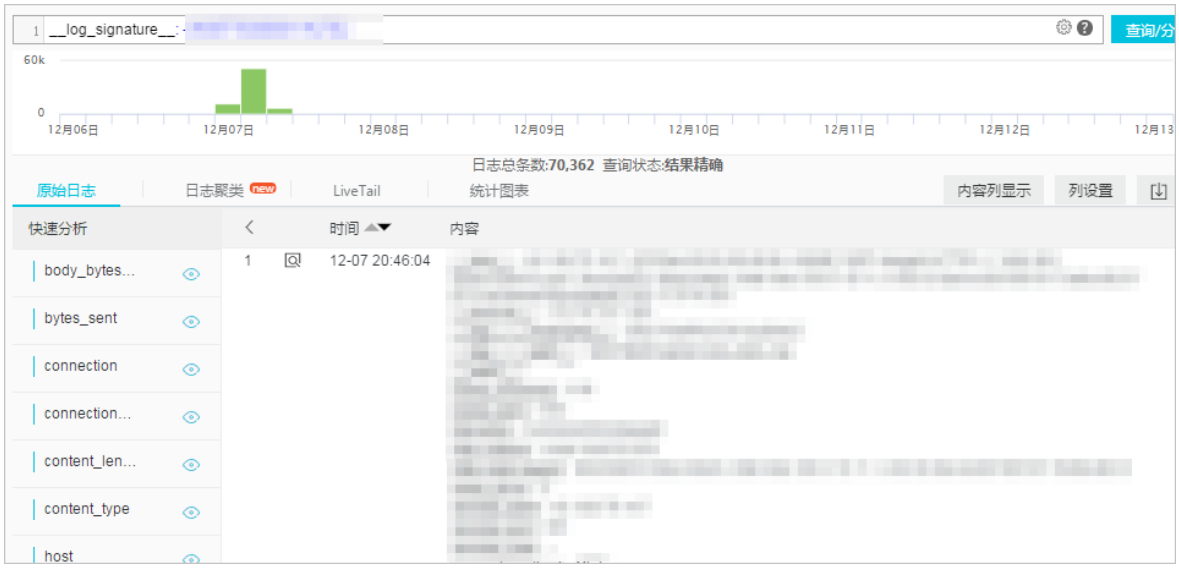
2. 单击进入**日志聚类**页签，查看聚类结果。
   日志聚类页签展示了过滤后的日志聚类结果。

显示项	说明
Number	聚类序号。
Count	该聚类类别的日志条数。
Pattern	具体的日志模式，每个聚类会有一个或多个子Pattern。

   - 鼠标指向**Count**列，有浮动栏展示当前聚类的子Pattern、每个子Pattern的占比。也可以单击数字前的加号+，展开子Pattern列表。



   - 单击子Pattern前的Count值，查看该分类中的原始日志。



调整聚类精度

1. 在查询分析页面单击进入**日志聚类**页签。
2. 在页签右上角的**Pattern分类**中拖拽滑动条，调整聚类的精度。
   - 聚类偏向于**多**，表示聚类结果分类细，Pattern保留的细节多。
   - 聚类偏向于**少**，表示聚类结果分类粗，Pattern细节被隐藏得更多。

对比不同时间段的聚类日志数量

在**LogReduce**页签单击**Log Compare**，设置对比时间后单击**确定**按钮。

显示项	说明
Number	聚类编号。
Pre_Count	当前Pattern在前一时段的原始日志数量。
Count	当前Pattern在当前时段的原始日志数量。
Diff	当前Pattern在两个时段的日志数量差值。
Pattern	某类日志具体的Pattern结果。

SQL语句示例

- 获取日志聚类结果
  - SQL语句：

```
* | select a.pattern, a.count,a.signature, a.origin_signatures from (select log_reduce(3) as a from log) limit 1000
```

> **说明** 查看聚类结果时，您也可以通过**复制查询语句**按钮来获取日志聚类结果的SQL语句。

  - 输入参数：log_reduce（**precision**）

    precision ： 接收整数[1~16]作为聚类精度，数字越小，聚类精度越高，生成的Pattern格式越多，默认为3。

- 返回字段：
  - pattern：某类日志具体的Pattern结果。
  - count：某类日志的个数。
  - signature：某类日志Pattern的签名。
  - origin_signatures：这类日志的二级签名，可以通过这个二级签名，反查原始数据。

- 对比不同时间段日志聚类结果
  - SQL语句

```
* | select v.pattern, v.signature, v.count, v.count_compare, v.diff from (select
compare_log_reduce(3, 86400) as v from log) order by v.diff desc limit 1000
```

> **说明** 通过**Log Compare**按钮来对比日志聚类结果后，您也可以通过**复制查询语句**按钮来获取对应的SQL语句。

  - 输入参数：compare_log_reduce（**precision**, **compare_interval**）
    - **precision** ：接收整数[1~16]作为聚类精度，数字越小，聚类精度越高，生成的Pattern格式越多，默认为3。
    - **compare_interval** ：正整数，表示对比多少秒之前的数据。
  - 返回字段：
    - pattern：某类日志具体的Pattern结果 。
    - signature：某类日志Pattern的签名。
    - count：某类日志的个数。
    - count_compare：对比时间段同signature日志个数。
    - diff：count和count_compare的差值。

4.7.4. 上下文查询

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。

前提条件

已开启索引。

背景信息

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。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。

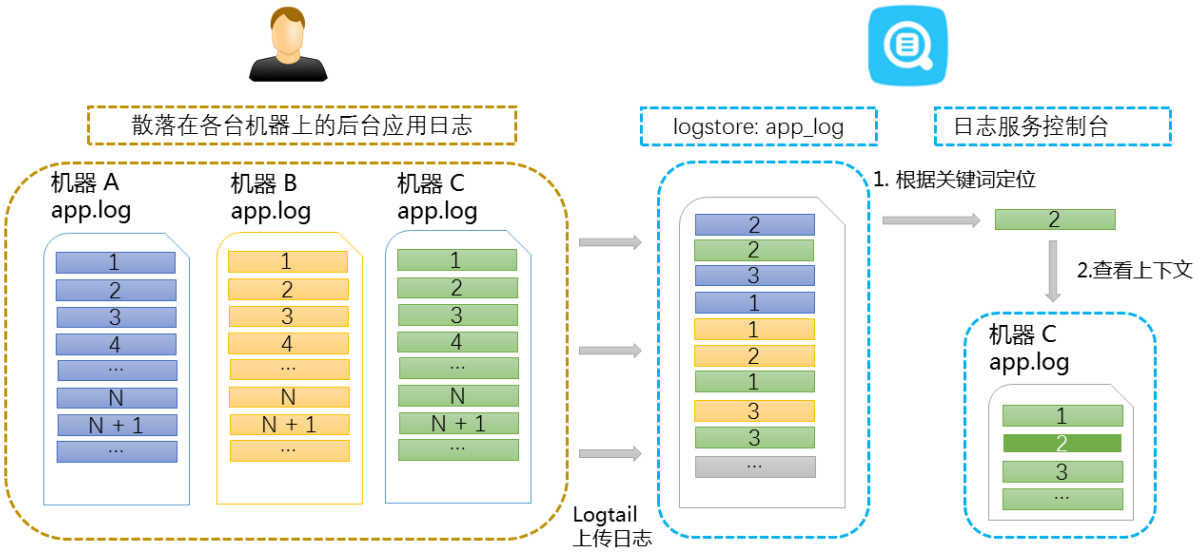
应用场景

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** > **选择物品** > **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）。



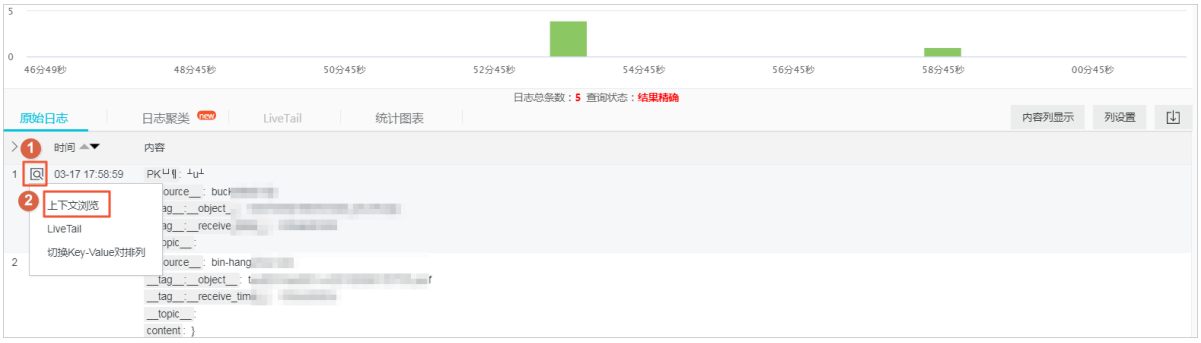
> **说明** 上下文查询功能暂不支持syslog日志。

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，解放了过去需要登录每台机器查看日志文件的痛苦。
- 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。

操作步骤

1. 登录日志服务控制台。
2. 在Project中单击目标Project名称。
3. 单击日志库名称后的图标，选择**查询分析**。
4. 输入您的查询分析语句，选择查询时段并单击**查询/分析**。
   查询结果页中任一条日志的左侧有**上下文浏览**按钮，表明该日志支持上下文查看功能。



5. 选中一条日志，单击**上下文浏览**。在右侧弹出页面中查看目标日志的上下文日志。
6. 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。

4.7.5. 快速查询

快速查询是日志服务提供的一键查询分析功能。

前提条件

已开启并设置索引。

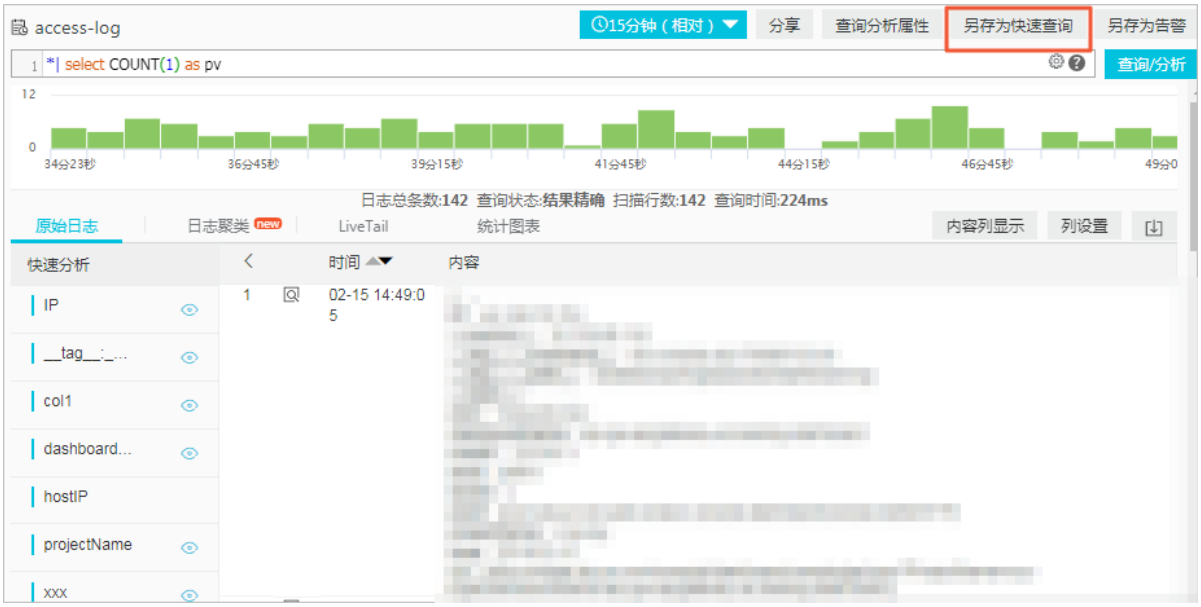
背景信息

当您需要经常查看某一查询分析语句的结果时，可以将其另存为快速查询，下次进行该查询动作时，不需要手动输入查询语句，只需在查询页面左侧单击该快速查询的名称，即可再次进行该项查询动作。您也可以在告警规则中使用该快速查询条件。日志服务会定期执行该快速查询语句，并在查询结果满足预设条件时发送告警信息。

设置下钻分析时，如果需要将下钻事件设置**跳转到快速查询**，必须提前配置快速查询，并在查询语句中设置**占位符**。

操作步骤

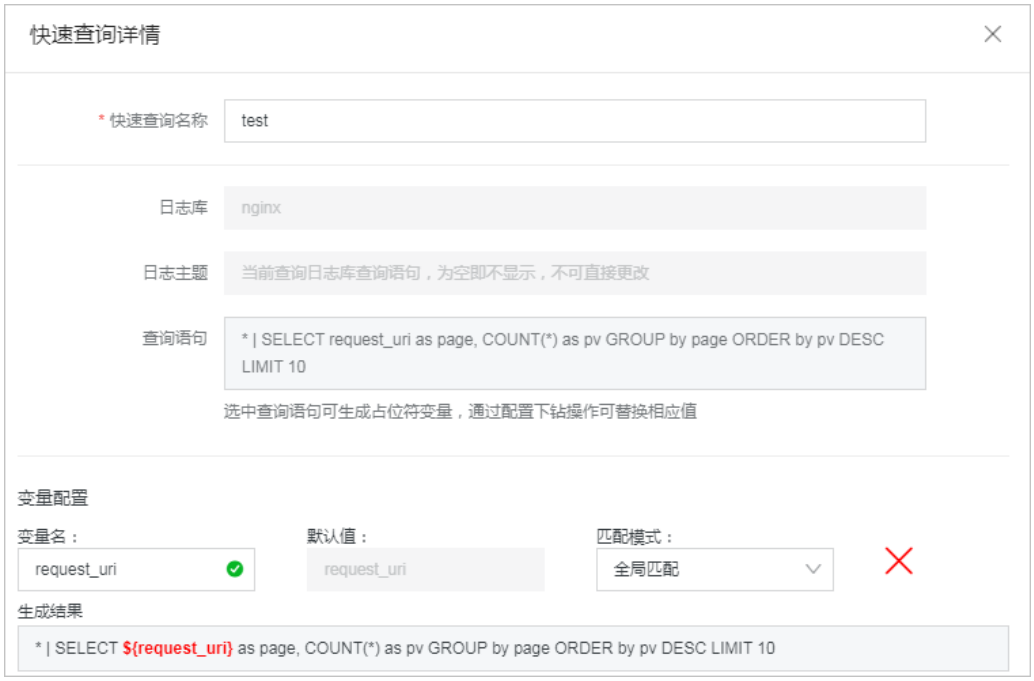
1. 登录日志服务控制台。
2. 单击日志库名称后的图标，选择**查询分析**。
3. 输入您的查询分析语句，设置时间范围，并单击**查询/分析**。
4. 单击页面右上角的**另存为快速查询**。



5. 设置快速查询属性。
   i. 设置**快速查询名称**。
      - 名称仅支持小写字母、数字、连字符（-）和下划线（_）。
      - 必须以小写字母或数字开头和结尾。
      - 名称长度为3~63个字符。
   ii. 确认**日志库**、**日志主题**和**查询语句**。
   若**日志库**和**日志主题**不符合您的需求，请返回至查询页面进入正确的日志库并输入查询语句，再次单击**另存为快速查询**。

iii. **可选：**划选查询语句的部分内容，并单击**生成变量**。
生成的变量为占位符变量。您可以在变量名中为占位符变量命名。默认值是您划选时选中的词。

> **说明** 如果其他图表的下钻事件为跳转到这个快速查询，且图表配置的变量和快速查询的变量名相同。单击其他图表时会执行跳转，占位符变量的默认值替换为触发下钻事件的图表值，并以替换变量后的查询语句执行查询。详细信息请参见下钻分析。



6. **单击确定，结束配置。**

后续步骤

如何修改快速查询？
修改快速查询时，输入新的查询分析语句之后，请单击查询/分析，执行一次查询后再单击修改已有快速查询即可。

4.7.6. 快速分析

日志服务（Log Service）快速分析功能给用户提供了一键式交互查询体验，帮助用户快速分析某一字段在指定时间内的分布情况，降低用户索引关键数据的成本。

功能特点

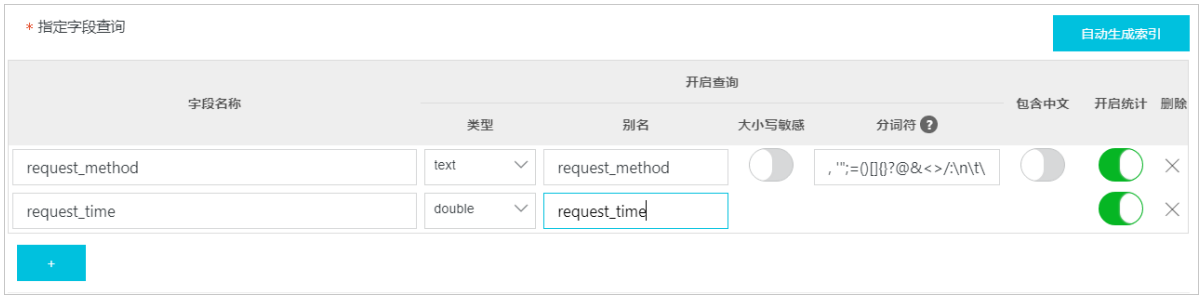
- 支持 `Text` 类型字段前100000条数据的前十项分组统计。
- 支持 `Text` 类型字段快速生成 `approx_distinct` 查询语句。
- 支持 `long` 或者 `double` 类型字段近似分布直方统计。
- 支持 `long` 或者 `double` 类型字段快速查找最大项、最小项、平均值或总和。
- 支持将快速分析查询生成查询语句。

前提条件

快速分析需要用户指定字段查询属性。

- 指定字段查询需要先开启索引以开启查询分析功能，如何开启索引请参见开启并配置索引。

- 设置日志中的 `key` 为字段名称，并设置类型、别名、分词符等。
  如访问日志中存在 `request_method` 和 `request_time` 字段，可以参考如下设置。



使用指南

设置好指定字段查询后，您可以在查询页面的**原始日志**页签左侧**快速分析**一栏处查看到对应的字段。序号顶部按钮可以进行页面折叠，单击**眼睛**图标即可根据**当前时间区间**、**当前的$Search**条件进行快速分析。

图 1. 快速分析

Text类型

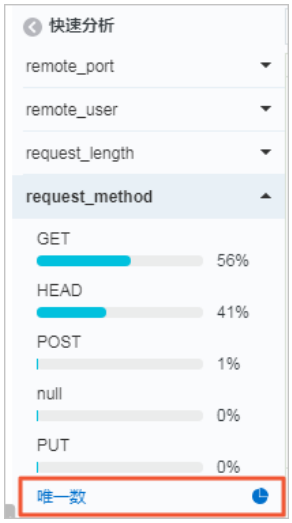
- Text类型分组统计

  单击目标字段右侧的**眼睛**图标，快速对该**Text**类型字段前100,000条数据进行分组，并返回前十项的占比。

  查询语句如下：

```
$Search | select ${keyName} , pv, pv *1.0/sum(pv) over() as percentage from( select count(1) as pv , "${keyName}" from (select "${keyName}" from log limit 100000) group by "${keyName}" order by pv desc) order by pv desc limit 10
```

  `request_method` 按照分组统计可以得到如下结果，`GET` 请求占大多数：



- 检查字段唯一项的个数

  在**快速分析**一栏的目的字段下单击**唯一数**，即可进行检查操作，即检查 `${keyName}` 唯一项的个数。

- 将分组统计的查询语句扩展到搜索框

单击**唯一数**右侧的按钮，将分组统计的查询语句扩展到搜索框，便于进一步操作。

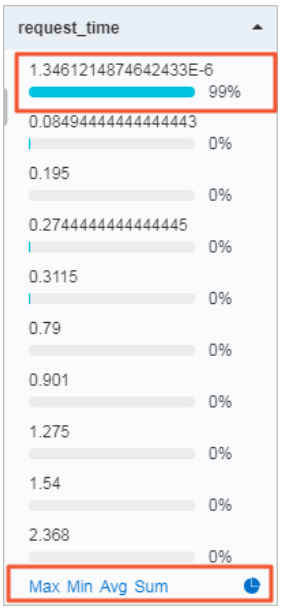
long/double类型

- 近似分布直方统计

  `long/double` 类型由于存在的值的数量太大，计算分组统计意义不大，我们分为10个桶进行近似分布直方统计，查询语句如下：

  ```
  $Search | select numeric_histogram(10, ${keyName})
  ```

  `request_time` 按照近似分布直方统计可以得到如下结果，可以知道绝大多数请求时间分布在0.059周围：

- `Max` 、`Min` 、`Avg` 、`Sum` 语句快速分析

  分别单击目标字段下的 `Max` 、`Min` 、`Avg` 、`Sum` ，快速查找所有项中的最大项、最小项、平均值和总和。

- 将近似分布直方统计的查询语句扩展到搜索框

  单击 `Sum` 右侧的按钮，将近似分布直方统计的查询语句扩展到搜索框，便于进一步操作。

4.7.7. 其他功能

日志服务查询分析功能除了提供日志内容的各种语句查询能力以外，还提供原始日志、统计图表、上下文查询、快速分析、快速查询、仪表盘、告警等多种扩展功能。

原始日志

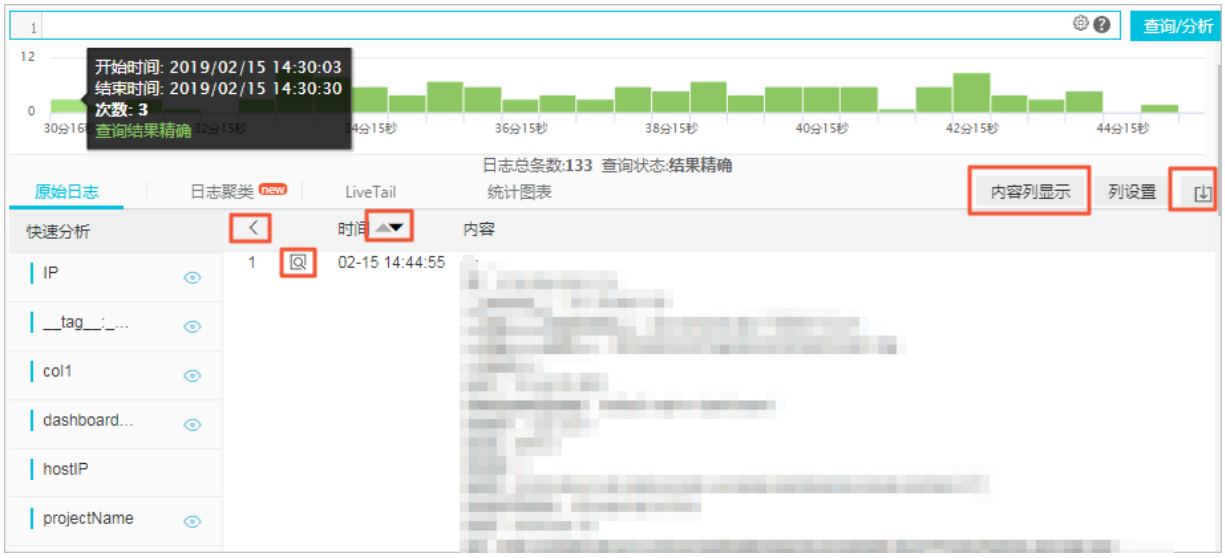
开启索引后，在检索框输入关键字、选择查询时段，单击**查询/分析**后，即可看到日志数量的直方图、原始日志及统计图表。

日志数量的直方图即日志检索的命中数量在时间上的分布，您可以通过直方图查看某个时间段的日志数量变化，单击长方形区域可以缩小时间范围，查看长方形区域表示的时间范围内的日志命中情况，为您的日志检索结果提供更精细的展示方式。

在原始日志页签中，您可以按时间排序，查看被命中的日志内容。

- 单击列名**时间**旁的三角符号，可切换**时间正序**或**时间倒序**。
- 单击列名**内容列显示**，可以选择日志内容**换行**或者**整行**显示，或设置长字符串折叠。
- 单击日志内容中的value**关键字**，可以查看包含该关键字的所有日志内容。
- 单击原始日志页签右上角的 图标，可下载CSV格式的查询结果，单击**列设置**图标，可在原始日志显示结果中增加字段的显示列，您可以更直观地在新增列中查看每条原始日志的目标字段内容。
- 单击**上下文浏览**查看该日志的前后各15条日志。更多信息请参见上下文查询。

  > **说明** 上下文查询功能目前仅支持使用Logtail上传的数据。



统计图表

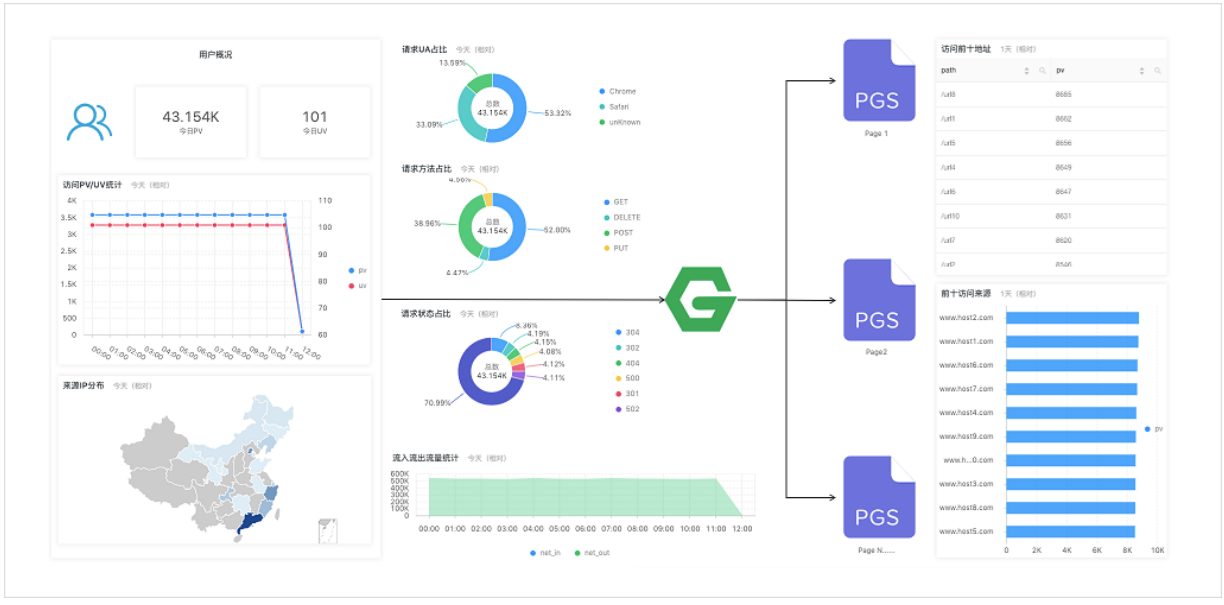
开启索引并输入查询和分析语句后，您可以在**统计图表**页签中查看日志的统计结果。

- 提供表格、折线图等多种统计图表。

  支持根据统计分析的需要选择统计图表类型，支持多种自定义配置。
- 支持将统计图表添加到**仪表盘**，详细内容请参见创建和删除仪表盘。
- 支持为统计图表设置下钻分析，该图表添加到仪表盘之后，单击图表数据即触发下钻事件，深化查询维度。

仪表盘

日志服务提供仪表盘功能，支持将查询分析语句进行可视化展示。详细信息请参见创建和删除仪表盘。



另存为告警

日志服务支持基于您的仪表盘或查询语句进行告警，您可以通过配置规则将具体告警内容发送给您。

详细信息请参见设置告警。

4.8. SQL分析语法与功能

4.8.1. 通用聚合函数

本文档为您介绍通用聚合函数的基本语法及示例。

日志服务查询分析功能支持通过通用聚合函数进行日志分析，详细语句及含义如下：

语句	含义	示例
`arbitrary(x)`	随机返回x列中的一个值。	`latency > 100 | select arbitrary(method)`
`avg(x)`	计算x列的算数平均值。	`latency > 100 | select avg(latency)`
`checksum(x)`	计算某一列的checksum，返回base64编码。	`latency > 100 | select checksum(method)`
`count(*)`	表示所有的行数。	-
`count(x)`	计算某一列非null的个数。	`latency > 100 | select count(method)`
`count(数字)`	`count(数字)`，如`count(1)`，等同于`count(*)`，表示所有的行数。	-
`count_if(x)`	计算x=true的个数。	`latency > 100 | select count_if(url like '%abc')`
`geometric_mean(x)`	计算某一列的几何平均数。	`latency > 100 | select geometric_mean(latency)`
`max_by(x,y)`	返回当y取最大值时，x当前的值。	查询延时最高的时候，对应的method：`latency>100 | select max_by(method,latency)`
`max_by(x,y,n)`	返回y最高的n行，对应的x的值。	查询延时最高的3行，对应的method：`latency > 100 | select max_by(method,latency,3)`
`min_by(x,y)`	返回当y取最小值时，x当前的值。	查询延时最低的请求，对应的method：`* | select min_by(x,y)`
`min_by(x,y,n)`	返回y最小的n行，对应的x的值。	查询延时最小的3行，对应的method：`* | select min_by(method,latency,3)`
`max(x)`	返回最大值。	`latency > 100| select max(inflow)`
`min(x)`	返回最小值。	`latency > 100| select min(inflow)`
`sum(x)`	返回x列的和。	`latency > 10 | select sum(inflow)`
`bitwise_and_agg(x)`	对某一列的所有数值做and计算。	-
`bitwise_or_agg(x)`	对某一列的数值做or计算。	-

4.8.2. 安全检测函数

日志服务依托全球白帽子共享安全资产库，提供安全检测函数，您只需要将日志中任意的IP、域名或者URL传给安全检测函数，即可检测是否安全。

应用场景

- 对服务运维有较强需求的企业和机构如互联网、游戏、资讯等，其IT和安全运维人员可借此及时筛选可疑访问、攻击以及侵入的行为，并支持进一步深入分析和采取一定措施进行防御。

- 对内部资产保护有较强需求的企业和机构如银行、证券、电商等，其IT、安全运维人员可以借此即时发现内部访问危险网站、下载木马等行为，并即时采取行动。

功能特点

- 可靠：依托全球共享的白帽子安全资产库，并及时更新。
- 快速：检测百万IP、域名或URL仅需几秒钟。
- 简单：无缝支持任意网络日志，调用3个SQL函数security_check_ip、security_check_domain、security_check_url即可获得结果。
- 灵活：既可以交互式查询，也可以构建报表视图。还可以建立报警并采取进一步行动。

函数列表

函数名	含义	样例
security_check_ip	检查IP是否安全，其中： • 返回1：命中，表示不安全 • 返回0：未命中	`select security_check_ip(real_client_ip)`
security_check_domain	检查Domain是否安全，其中： • 返回1：命中，表示不安全 • 返回0：未命中	`select security_check_domain(site)`
security_check_url	检查URL是否安全，其中： • 返回1：命中，表示不安全 • 返回0：未命中	`select security_check_domain(concat(host, url))`

示例

- 检查外部可疑访问行为并生成报表

  某电商收集了其运营的Nginx服务器的日志，对其访问的客户端中想要扫描是否存在不安全的客户IP。可以将Nginx的日志中的ClientIP字段传给 `security_check_ip` 函数，并筛选出其返回值为1的IP进行展现，并展示其所在国家、网络运营商等。

  对应查询分析语句为：

```
* | select ClientIP, ip_to_country(ClientIP) as country, ip_to_provider(ClientIP) as provider, count(1) as PV where security_check_ip(ClientIP) = 1 group by ClientIP order by PV desc
```

ClientIP	sec	country	provider	PV
[redacted]	1	中国	电信	575
[redacted]	1	中国	联通	241
[redacted]	1	中国	电信	185
[redacted]	1	中国	联通	179
[redacted]	1	中国	联通	32
[redacted]	1	中国	电信	28

  设置为地图视图展示：

client_ip	country	provider	PV
180[redacted]3	CN	电信	3
103[redacted]	CN		3
180[redacted]7	CN	电信	1

- 检查内部可疑访问行为并报警
  例如，某证券运营商收集了其内部设备通过网关代理访问外网的网络流量的日志，需要检查是否有人访问了有问题的网站，可以执行如下查询：

```
* | select client_ip, count(1) as PV where security_check_ip(remote_addr) = 1 or
security_check_site(site) = 1 or security_check_url(concat(site, url)) = 1 group by client_ip order by
PV desc
```

您也可以将此语句另存为快速查询，并建立安全报警，当有客户端频繁访问危险网站时触发报警，配置每5分钟检查一次是否有人过去1小时内频繁（超过5次）访问危险网站。配置如下：

4.8.3. Map映射函数

本文档为您介绍Map映射函数的语法说明及示例。

日志服务查询分析功能支持通过映射函数进行日志分析，详细语句及含义如下所示。

函数	含义	示例
下标运算符[]	获取map中某个Key对应的结果。	-
histogram(x)	按照x的每个值进行GROUP BY并计算count。语法相当于 `select count group by x` 。 **说明** 返回结果为JSON格式。	`latency > 10 | select histogram(status)` ，等同于 `latency > 10 | select count(1) group by status` 。
histogram_u(x)	按照x的每个值进行GROUP BY并计算count。 **说明** 返回结果为多行多列。	`latency > 10 | select histogram(status)` ，等同于 `latency > 10 | select count(1) group by status` 。
map_agg(Key,Value)	返回Key、Value组成的map，并返回Key随机的Value。	`latency > 100 | select map_agg(method,latency)`

multimap_agg(Key,Value)	返回Key、Value组成的多Value map，并返回Key的所有的Value。	`latency > 100 | select multimap_agg(method,latency)`
cardinality(x) → bigint	获取map的大小。	-
element_at(map `<K, V>`, key) → V	获取Key对应的Value。	-
map() → map `<unknown, unknown>`	返回一个空的map。	-
map(array `<K>`, array `<V>`) → map `<K,V>`	将两个数组转换成Key、Value组成的Map。	`SELECT map(ARRAY[1,3], ARRAY[2,4]); — {1 -> 2, 3 -> 4}`
map_from_entries(array `<row<K, V>>`) → map `<K,V>`	把一个多维数组转化成map。	`SELECT map_from_entries(ARRAY[(1, 'x'), (2, 'y')]); — {1 -> 'x', 2 -> 'y'}`
map_concat(map1 `<K, V>`, map2 `<K, V>`, ..., mapN `<K, V>`) → map `<K,V>`	求多个map的并集，如果某个Key存在于多个map中，则取最后一个。	-
map_filter(map `<K, V>`, function) → map `<K,V>`	请参见Lambda函数中map_filter函数。	-
map_keys(x `<K, V>`) → array `<K>`	获取map中所有的Key并以array的形式返回。	-
map_values(x `<K, V>`) → array `<V>`	获取map中所有的Value并以array的形式返回。	-

4.8.4. 估算函数

本文档为您介绍估算函数的基本语法及示例。

日志服务查询分析功能支持通过估算进行日志分析，详细语句及含义如下：

函数	说明	示例
`approx_distinct(x)`	估算x列的唯一值的个数。	-
`approx_percentile(x,percentage)`	对于x列排序，找出大约处于percentage位置的数值。	找出位于一半位置的数值：`approx_percentile(x,0.5)`。
`approx_percentile(x, percentages)`	与`approx_percentile(x,percentage)`用法类似，但可以指定多个percentage，找出每个percentage对应的数值。	`approx_percentile(x,array[0.1,0.2])`
`numeric_histogram(buckets, Value)`	对于数值列，分多个桶进行统计。即把*Value*一列，分到桶中，桶的个数为*buckets*。 返回内容为每个桶的Key及对应的count数值，相当于针对数值的`select count group by`。 **说明** 返回结果的格式为JSON。	对于POST请求，把延时分为10个桶，查看每个桶的大小：`method:POST | select numeric_histogram(10,latency)`。

`numeric_histogram_u(buckets, Value)`	对于数值列，分多个桶进行统计。即把*Value*一列，分到桶中，桶的个数为*buckets*。 返回内容为每个桶的Key及对应的count数值，相当于针对数值的 `select count group by` 。 **说明** 返回结果的格式为多行多列。	对于POST请求，把延时分为10个桶，查看每个桶的大小： `method:POST | select numeric_histogram_u(10,latency)` 。

说明 上述说明中桶是按照聚集程度均分的，每个结果里显示这个区域的均值和个数。

4.8.5. 数学统计函数

本文档为您介绍数学统计函数的基本语法及示例。

日志服务查询分析功能支持通过数学统计函数进行日志分析，详细语句及含义如下：

语句	含义	示例
`corr(y, x)`	给出两列的相关度，结果从0到1。	`latency>100| select corr(latency,request_size)`
`covar_pop(y, x)`	计算总体协方差。	`latency>100| select covar_pop(request_size,latency)`
`covar_samp(y, x)`	计算样本协方差。	`latency>100| select covar_samp(request_size,latency)`
`regr_intercept(y, x)`	返回输入值的线性回归截距。 y是依赖值，x是独立值。	`latency>100| select regr_intercept(request_size,latency)`
`regr_slope(y,x)`	返回输入值的线性回归斜率。 y是依赖值，x是独立值。	`latency>100| select regr_slope(request_size,latency)`
`stddev(x)` 或 `stddev_samp(x)`	返回x列的样本标准差。	`latency>100| select stddev(latency)`
`stddev_pop(x)`	返回x列的总体标准差。	`latency>100| select stddev_pop(latency)`
`variance(x)` 或 `var_samp(x)`	计算x列的样本方差。	`latency>100| select variance(latency)`
`var_pop(x)`	计算x列的总体方差。	`latency>100| select variance(latency)`

4.8.6. 数学计算函数

本文档为您介绍数学计算函数的基本语法及示例。

日志服务查询分析功能支持通过数学计算函数进行日志分析，您可以结合查询语句和数学计算函数，对日志查询结果进行数学计算。

数学运算符

数学运算符支持 + - * / % 。可以用在SELECT子句中。

样例：

```
*|select avg(latency)/100 , sum(latency)/count(1)
```

数学计算函数说明

日志服务支持以下运算函数：

函数名	含义
abs(x)	返回x列的绝对值。
cbrt(x)	返回x列的立方根。
ceiling (x)	返回x列向上最接近的整数。
cosine_similarity(x,y)	返回稀疏向量x和y之间的余弦相似度。
degrees	把弧度转化为度。
e()	返回自然常数。
exp(x)	返回自然常数的指数。
floor(x)	返回x向下最接近的整数。
from_base(string,radix)	以radix进制解释string。
ln(x)	返回自然对数。
log2(x)	返回以2为底，x的对数。
log10(x)	返回以10为底，x的对数。
log(x,b)	返回以b为底，x的对数。
pi()	返回π。
pow(x,b)	返回x的b次幂。
radians(x)	把度转化成弧度。
rand()	返回随机数。
random(0,n)	返回[0，n)随机数。
round(x)	x四舍五入。
round(x, y)	对x保留y个小数为，例如round(1.012345,2) = 1.01。
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()	返回正无穷的数值。
is_infinity(x)	判断值是否是无限值。
is_finity(x)	判断是否是有限值。
is_nan(x)	判断是否是非数值。

4.8.7. 字符串函数

日志服务查询分析功能支持通过字符串函数进行日志查询和分析。

字符串函数详细语句及含义如下：

函数名	含义
chr(x)	把int类型转化成对应的ASCII码，例如 chr(65) 结果为 A 。
codepoint (x)	把一个ASCII码转化成int类型的编码，例如 codepoint('A') 结果为 65 。
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。
lpad(string, size, padstring)	把**string**对齐到**size**大小，如果小于**size**，用**padstring**从左侧补齐到；如果大于**size**则截取到**size**个。
rpad(string, size, padstring)	类似 lpad ，从右侧补齐**string**。
ltrim(string)	删掉左侧的空白字符。
replace(string, search)	把字符串中**string**中的**search**删掉。
replace(string, search,rep)	把字符串中**string**中的**search**替换为**rep**。
reverse(string)	翻转string。
rtrim(string)	删掉字符串结尾的空白字符。
split(string,delimeter,limit)	把字符串分裂成array，最多取**limit**个值。生成的结果为数组，下标从1开始。
split_part(string,delimeter,offset)	把字符串分裂成array，取第**offset**个字符串。生成的结果为字符串。
split_to_map(string, entryDelimiter, keyValueDelimiter) → map<varchar, varchar>	把**string**按照**entryDelemiter**分割成多个entry，每个entry再按照**keyValueDelimiter**划分成key value。最终返回一个map。
position(substring IN string)	获取**string**中，**substring**最先开始的位置。

strpos(string, substring)	查找字符串中的子串的开始位置。返回结果从1开始，如果不存在则返回0。
substr(string, start)	返回字符串的子串，**start**下标从1开始。
substr(string, start, length)	返回字符串的子串，**start**下标从1开始，**length**指定子串的长度。
trim(string)	删掉字符串开头和结尾的空白字符。
upper(string)	转化为大写字符。
concat(string,string.....)	把两个或多个字符串拼接成一个字符串。
hamming_distance (string1,string2)	获得两个字符串的海明距离。

说明 字符串需要加单引号包裹，双引号表示列名。例如：`a='abc'` 表示 列a=字符串abc ；`"a"="abc"` 表示 a列=abc列 。

4.8.8. 日期和时间函数

日志服务支持时间函数、日期函数、区间函数和时序补全函数，您可以在分析语句中使用本文档中介绍的函数。

日期时间类型

- unixtime：以int类型表示从1970年1月1日开始的秒数，例如 `1512374067` 表示的时间是 `Mon Dec 4 15:54:27 CST 2017` 。日志服务每条日志中内置的时间 `__time__` 即为这种类型。
- timestamp类型：以字符串形式表示时间，例如 `2017-11-01 13:30:00` 。

日期函数

日志服务支持的常见日期函数如下：

函数名	含义	样例
current_date	当天日期。	latency>100| select current_date
current_time	当前时间。	latency>100| select current_time
current_timestamp	结合current_date 和current_time的结果。	latency>100| select current_timestamp
current_timezone()	返回时区。	latency>100| select current_timezone()
from_iso8601_timestamp(string)	把iso8601时间转化成带时区的时间。	latency>100| select from_iso8601_timestamp(iso8601)
from_iso8601_date(string)	把iso8601转化成天。	latency>100| select from_iso8601_date(iso8601)
from_unixtime(unixtime)	把unix时间转化为时间戳。	latency>100| select from_unixtime(1494985275)
from_unixtime(unixtime,string)	以string为时区，把unixtime转化成时间戳。	latency>100| select from_unixtime (1494985275,'Asia/Shanghai')
localtime	本地时间。	latency>100| select localtime

`localtimestamp`	本地时间戳。	`latency>100| select localtimestamp`
`now()`	等同于 `current_timestamp` 。	-
`to_unixtime(timestamp)`	timestamp转化成unixtime。	`*| select to_unixtime('2017-05-17 09:45:00.848 Asia/Shanghai')`

时间函数

日志服务还支持MySQL时间格式，包括%a、%b、%y 等。

函数名	含义	样例
`date_format(timestamp, format)`	把timestamp转化成以format形式表示。	`latency>100 | select date_format (date_parse('2017-05-17 09:45:00','%Y-%m-%d %H:%i:%S'), '%Y-%m-%d')`
`date_parse(string, format)`	把string以format格式解析，转化成timestamp。	`latency>100 | select date_format (date_parse(time,'%Y-%m-%d %H:%i:%S'), '%Y-%m-%d')`

表 1. 格式说明

格式	描述
%a	星期的缩写，即Sun、Sat等。
%b	月份的缩写，即Jan、Dec等。
%c	月份，数值类型，即1~12。
%D	每月的第几天，带后缀，即0th、1st、2nd、3rd等。
%d	每月第几天，十进制格式，范围为01~31。
%e	每月第几天，十进制格式，范围为1~31。
%H	小时，24小时制。
%h	小时，12小时制。
%I	小时，12小时制。
%i	分钟，数值类型，范围为00~59。
%j	每年的第几天，范围为001~366。
%k	小时，范围为0~23。
%l	小时，范围为1~12。
%M	月份的英文表达，范围为January~December。
%m	月份，数值格式，范围为01~12。
%p	AM或PM。
%r	时间，12小时制，格式为 `hh:mm:ss AM/PM` 。
%S	秒，范围为00~59。
%s	秒，范围为00~59。

%T	时间，24时制，格式为 `hh:mm:ss` 。
%U	每年的第几周，星期日是一周的第一天。取值范围为00~53。
%u	每年的第几周，星期一是一周的第一天。范围为00~53。
%V	每年的第几周，星期日是一周的第一天。范围为01~53，与%X同时使用。
%v	每年的第几周，星期一是一周的第一天。范围为01~53，与%x同时使用。
%W	星期几的名称，范围为Sunday到Saturday。
%w	一周的第几天， 星期日为第0天。
%Y	4 位数的年份。
%y	2位数的年份。
%%	%转义字符。

时间段对齐函数

日志服务支持时间段对齐函数，可以按照秒、分钟，小时、日、月、年等对齐。这个函数常用于一些按照时间进行统计的场景。

- 函数语法

```
date_trunc(unit, x)
```

- 参数说明

  x 可以是一个timestamp类型，也可以是unix time。

  Unit的取值包括以下类型，其中x取 `2001-08-22 03:04:05.000` ：

Unit	转化后结果
second	2001-08-22 03:04:05.000
minute	2001-08-22 03:04:00.000
hour	2001-08-22 03:00:00.000
day	2001-08-22 00:00:00.000
week	2001-08-20 00:00:00.000
month	2001-08-01 00:00:00.000
quarter	2001-07-01 00:00:00.000
year	2001-01-01 00:00:00.000

- 示例

  date_trunc只能在按照一些固定时间间隔统计，如果需要按照灵活的时间维度进行统计（例如统计每5分钟数据），需要按照数学取模方法进行GROUP BY。

```
* | SELECT count(1) as pv,  __time__ - __time__% 300 as minute5 group by minute5 limit 100
```

  上述公式中的 `%300` 表示按照5分钟进行取模对齐。

  以下为使用时间格式的一个综合样例。

```
*|select  date_trunc('minute' ,  __time__)  as t,
       truncate (avg(latency) ) ,
       current_date
       group by   t
       order by t  desc
       limit 60
```

时间间隔函数

时间间隔函数用来执行时间段相关的运算，如在日期中添加或减去指定的时间间隔、计算两个日期之间的时间。

函数名	含义	样例
`date_add(unit, value, timestamp)`	在 `timestamp` 上加上 `value` 个 `unit` 。如果要执行减法， `value` 使用负值。	`date_add('day', -7, '2018-08-09 00:00:00')` 表示8月9号之前7天
`date_diff(unit, timestamp1, timestamp2)`	表示 `timestamp1` 和 `timestamp2` 之间相差几个 `unit` 。	`date_diff('day', '2018-08-02 00:00:00', '2018-08-09 00:00:00') = 7`

该函数支持以下区间单位：

单位	说明
millisecond	毫秒
second	秒
minute	分钟
hour	小时
day	天
week	周
month	月
quarter	季度，即三个月
year	年

时序补全函数

时序补全函数time_series用于处理某些时间缺少的情况。

> **说明** 该函数必须和 `group by time order by time` 一起使用，且 `order by` 不支持 `desc` 排序方式

- 函数格式

```
time_series(time_column, window, format, padding_data)
```

- 参数说明

参数	说明
time_column	时间列，例如日志服务提供的默认时间字段 `__time__` 。格式为long类型或timestamp类型。
window	窗口大小，由一个数字和单位组成。单位为s（秒）、m（分）、H（小时）、或d（天）。例如2h、5m、3d。
format	MySQL时间格式，表示最终输出的格式。
padding_data	表示补全的内容，包括： ◦ 0：补零。 ◦ null：补null。 ◦ last：补上一个值。 ◦ next：补下一个值。 ◦ avg：补前后的平均值。

- 示例

按照每两个小时进行格式化：

```
* | select time_series(__time__, '2h', '%Y-%m-%d %H:%i:%s', '0')  as stamp, count(*) as num from log group by stamp order by stamp
```

4.8.9. URL函数

本文档为您介绍URL函数的基本语法及示例。

URL函数支持从标准URL路径中提取字段，一个标准的URL如下：

```
[protocol:][//host[:port]][path][?query][#fragment]
```

常见URL函数

函数名	含义	示例	
		输入样例	输出结果
`url_extract_fragment(url)`	提取出URL中的fragment，结果为varchar类型。	`*| select url_extract_fragment('https://sls.console.aliyun.com/#/project/dashboard-demo/categoryList')`	输出结果为 `/project/dashboard-demo/categoryList` 。
`url_extract_host(url)`	提取出URL中的host，结果为varchar类型。	`*|select url_extract_host('http://www.aliyun.com/product/sls')` 。	输出结果为 `www.aliyun.com` 。
`url_extract_parameter(url, name)`	提取出URL中的query中name对应的参数值，结果为varchar类型。	`*|select url_extract_parameter('http://www.aliyun.com/product/sls?userid=testuser','userid')`	输出结果为 `testuser` 。
`url_extract_path(url)`	提取出URL中的path，结果为varchar类型。	`*|select url_extract_path('http://www.aliyun.com/product/sls?userid=testuser')`	输出结果为 `/product/sls` 。
`url_extract_port(url)`	提取出URL中的端口，结果为bigint类型。	`*|select url_extract_port('http://www.aliyun.com:80/product/sls?userid=testuser')`	输出结果为 `80` 。
`url_extract_protocol(url)`	提取出URL中的协议，结果为varchar类型。	`*|select url_extract_protocol('http://www.aliyun.com:80/product/sls?userid=testuser')`	输出结果为 `http` 。
`url_extract_query(url)`	提取出URL中的query，结果为varchar类型。	`*|select url_extract_query('http://www.aliyun.com:80/product/sls?userid=testuser')`	输出结果为 `userid=testuser` 。
`url_encode(value)`	对url进行转义编码。	`*|select url_encode('http://www.aliyun.com:80/product/sls?userid=testuser')`	输出结果为 `http%3a%2f%2fwww.aliyun.com%3a80%2fproduct%2fsls%3fuserid%3dtestuser` 。

`url_decode(value)`	对url进行解码。	`*|select url_decode('http%3a%2f%2fwww.aliyun.com%3a80%2fproduct%2fsls%3fuserid%3dtestuser')`	输出结果为 `http://www.aliyun.com:80/product/sls?userid=testuser` 。

4.8.10. 正则式函数

日志服务支持正则式函数，您可以在查询分析数据时使用该函数。

正则式函数解析一串字符串，并且返回需要的一部分子串。

常见的正则式函数及含义如下：

函数名	含义	样例
`regexp_extract_all(string, pattern)`	返回字符串中命中正则式的所有子串，返回结果是一个字符串数组。	`*| SELECT regexp_extract_all('5a 67b 890m', '\d+')` ，结果为 `['5','67','890']` ， `*|SELECT regexp_extract_all('5a 67a 890m', '(\d+)a')` 结果为 `['5a','67a']` 。
`regexp_extract_all(string, pattern, group)`	返回字符串中命中正则式的第group个 `()` 内部分，返回结果是一个字符串数组。	`*| SELECT regexp_extract_all('5a 67a 890m', '(\d+)a',1)` 结果为 `['5','67']`
`regexp_extract(string, pattern)`	返回字符串命中的正则式的第一个子串。	`*|SELECT regexp_extract('5a 67b 890m', '\d+')` 结果为 `'5'`
`regexp_extract(string, pattern,group)`	返回字符串命中的正则式的第group个 `()` 内的第1个子串。	`*|SELECT regexp_extract('5a 67b 890m', '(\d+)([a-z]+)',2)` 结果为 `'a'`
`regexp_like(string, pattern)`	判断字符串是否命中正则式，返回bool类型，正则式可以只命中字符串的一部分。	`*|SELECT regexp_like('5a 67b 890m', '\d+m')` 结果为true
`regexp_replace(string, pattern, replacement)`	把字符串中命中正则式的部分替换成replacement。	`*|SELECT regexp_replace('5a 67b 890m', '\d+','a')` 结果为 `'aa ab am'`
`regexp_replace(string, pattern)`	把字符串中命中正则式的部分删除，相当于 `regexp_replace(string,pat term,'')` 。	`*|SELECT regexp_replace('5a 67b 890m', '\d+')` 结果为 `'a b m'`
`regexp_split(string, pattern)`	使用正则式把字符串切分成数组。	`*|SELECT regexp_split('5a 67b 890m', '\d+')` 结果为 `['a','b','m']`

4.8.11. JSON函数

本文档为您介绍JSON函数的基本语法及示例。

JSON函数，可以解析一段字符串为JSON类型，并且提取JSON中的字段。JSON主要有两种结构：map和array。如果一个字符串解析成JSON失败，那么返回的是null。

如果需要把json展开成多行，请参考unnest语法。

日志服务支持以下常见的JSON函数：

函数名	含义	样例
`json_parse(string)`	把字符串转化成JSON类型。	`SELECT json_parse('[1, 2, 3]')` 结果为JSON类型数组

`json_format(json)`	把JSON类型转化成字符串。	`SELECT json_format(json_parse('[1, 2, 3]'))` 结果为字符串
`json_array_contains(json, value)`	判断一个JSON类型数值，或者一个字符串（内容是一个JSON数组）是否包含某个值。	`SELECT json_array_contains(json_parse('[1, 2, 3]'), 2)`或 `SELECT json_array_contains('[1, 2, 3]', 2)`
`json_array_get(json_array, index)`	同 `json_array_contains` ，是获取一个JSON数组的某个下标对应的元素。	`SELECT json_array_get('["a", "b", "c"]', 0)`结果为'a'
`json_array_length(json)`	返回JSON数组的大小。	`SELECT json_array_length('[1, 2, 3]')` 返回结果3
`json_extract(json, json_path)`	从一个JSON对象中提取值，JSON路径的语法类似 `$.store.book[0].title` ，返回结果是一个JSON对象。	`SELECT json_extract(json, '$.store.book');`
`json_extract_scalar(json, json_path)`	类似 `json_extract` ，但是返回结果是字符串类型。	-
`json_size(json,json_path)`	获取JSON对象或数组的大小。	`SELECT json_size('[1, 2, 3]')` 返回结果3

4.8.12. 类型转换函数

类型转换函数用于在查询中转换指定值或指定列的数据类型。

日志服务索引属性中，字段可被配置为long、double、text和json类型。同时日志服务支持查询多种数据类型的字段，包括bigint、double、varchar、timestamp等。如果查询时需要区分更细维度的数据类型，可以使用类型转换函数将索引属性中配置的数据类型转换为查询中使用的数据类型。

函数格式

> **说明** 日志中可能有脏数据时，建议使用try_cast()函数，否则容易因脏数据造成整个查询失败。

- 在查询中将某一列（字段）或某一个值转换成指定类型。其中，如果某一个值转换失败，将终止整个查询。

```
cast([key|value] AS type)
```

- 在查询中将某一列（字段）或某一个值转换成指定类型。如果某一个值转换失败，该值返回NULL，并跳过该值继续处理。

```
try_cast([key|value] AS type)
```

参数	说明
key	日志的Key，表示将该字段所有的值都转换成指定类型。
value	常量值，表示将某个值转换成指定类型。

示例

- 将数字123转换为字符串（varchar）格式

```
cast(123 AS varchar)
```

- 将uid字段转换为字符串（varchar）格式

```
cast(uid AS varchar)
```

4.8.13. IP地理函数

本文档为您介绍 IP 地理函数的基本语法及示例。

IP 地理函数可以识别一个 IP 是内网 IP还是外网 IP，也可以判断 IP 所属的国家、省份、城市。关于 geohash 函数的介绍请参见地理函数。

函数名	含义	样例
`ip_to_domain(ip)`	判断 IP 所在的域，是内网还是外网。返回 intranet 或 internet。	`SELECT ip_to_domain(ip)`
`ip_to_country(ip)`	判断 IP 所在的国家。	`SELECT ip_to_country(ip)`
`ip_to_province(ip)`	判断 IP 所在的省份。	`SELECT ip_to_province(ip)`
`ip_to_city(ip)`	判断 IP 所在的城市。	`SELECT ip_to_city(ip)`
`ip_to_geo(ip)`	判断 IP 所在的城市的经纬度，范围结果格式为 `纬度,经度` 。	`SELECT ip_to_geo(ip)`
`ip_to_city_geo(ip)`	判断 IP 所在的城市的经纬度，返回的是城市经纬度，每个城市只有一个经纬度，范围结果格式为 `纬度,经度` 。	`SELECT ip_to_city_geo(ip)`
`ip_to_provider(ip)`	获取 IP 对应的网络运营商。	`SELECT ip_to_provider(ip)`
`ip_to_country(ip,'en')`	判断 IP 所在的国家，返回国家码。	`SELECT ip_to_country(ip,'en')`
`ip_to_country_code(ip)`	判断 IP 所在的国家，返回国家码。	`SELECT ip_to_country_code(ip)`
`ip_to_province(ip,'en')`	判断 IP 所在的省份，返回英文省名或者中文拼音。	`SELECT ip_to_province(ip,'en')`
`ip_to_city(ip,'en')`	判断 IP 所在的城市，返回英文城市名或者中文拼音。	`SELECT ip_to_city(ip,'en')`

示例

- 在查询中过滤掉内网访问请求，查看请求总数。

```
* | SELECT count(1) where ip_to_domain(ip)!='intranet'
```

- 查看 Top10 的访问省份。

```
* | SELECT count(1) as pv, ip_to_province(ip) as province GROUP BY province order by pv desc limit 10
```

响应结果样例

```
[
    {
        "__source__": "",
        "__time__": "1512353137",
        "province": "浙江省",
        "pv": "4045"
    }, {
        "__source__": "",
        "__time__": "1512353137",
        "province": "上海市",
        "pv": "3727"
    }, {
        "__source__": "",
        "__time__": "1512353137",
        "province": "北京市",
        "pv": "954"
    }, {
        "__source__": "",
        "__time__": "1512353137",
        "province": "内网IP",
        "pv": "698"
    }, {
        "__source__": "",
        "__time__": "1512353137",
        "province": "广东省",
        "pv": "472"
    }, {
        "__source__": "",
        "__time__": "1512353137",
        "province": "福建省",
        "pv": "71"
    }
]
```


上述结果中包含了内网 IP，为了过滤掉这部分访问请求，可以使用下边的分析语句：

- 过滤掉内网请求，查看 Top10 的网络访问省份。

```
* | SELECT count(1) as pv, ip_to_province(ip) as province WHERE ip_to_domain(ip) != 'intranet'  GROUP BY province ORDER BY pv desc limit 10
```

- 查看不同国家的平均响应延时、最大响应延时以及最大延时对应的 request。

```
* | SELECT AVG(latency),MAX(latency),MAX_BY(requestId, latency) ,ip_to_country(ip) as country group by country limit 100
```

- 查看不同网络运营商的平均延时。

```
* | SELECT AVG(latency) , ip_to_provider(ip) as provider group by provider limit 100
```

- 查看 IP 的经纬度，绘制地图。

```
* | SELECT count(1) as pv , ip_to_geo(ip) as geo group by geo order by pv desc
```

返回的格式为：

pv	geo
100	35.3284,-80.7459

4.8.14. GROUP BY 语法

本文档主要为您介绍GROUP BY 语法。

GROUP BY 支持多列。GROUP BY支持通过SELECT的列的别名来表示对应的KEY。

样例：

```
method:PostLogstoreLogs |select avg(latency),projectName,date_trunc('hour',__time__) as hour   group by
projectName,hour
```

别名hour代表第三个SELECT列 `date_trunc('hour',__time__)` 。这类用法对于一些非常复杂的query非常有帮助。

GROUP BY 支持GROUPING SETS、CUBE、ROLLUP。

样例：

```
method:PostLogstoreLogs |select avg(latency)   group by cube(projectName,logstore)
method:PostLogstoreLogs |select avg(latency)   group by GROUPING SETS ( ( projectName,logstore), (projec
tName,method))
method:PostLogstoreLogs |select avg(latency)   group by rollup(projectName,logstore)
```

实践样例

- 按照时间进行GROUP BY

  每条日志都内置了一个时间列 `__time__` ，当打开任意一列的统计功能后，会自动给时间列打开统计。

  使用 `date_trunc` 函数，可以把时间列对齐到小时（hour）、分钟（minute）、天（day）、月（month）、年（year）。 `date_trunc` 接受一个对齐单位，和一个unix time或者timestamp类型的列，例如 `__time__` 。

  - 按照每小时、每分钟统计计算PV

    ```
    * | SELECT count(1) as pv , date_trunc('hour',__time__) as hour group by hour order by hour limit 10
    0
    * | SELECT count(1) as pv , date_trunc('minute',__time__) as minute group by minute order by minute
    limit 100
    ```

    > **说明** limit 100表示最多获取100行，如果不加LIMIT语句，默认最多获取10行数据。

  - 按照灵活的时间维度进行统计，例如统计每5分钟的，date_trunc只能在按照一些固定时间间隔统计，这种场景下，我们需要按照数学取模方法进行GROUP BY。

    ```
    * | SELECT count(1) as pv,  __time__ - __time__% 300 as minute5 group by minute5 limit 100
    ```

    上述公式中的 `%300` 表示按照5分钟进行取模对齐。

- 在GROUP BY 中提取非agg列

  在标准SQL中，如果使用了GROUP BY语法，那么在SELECT时，只能选择SELECT GROUP BY的列原始内容，或者对任意列进行聚合计算，不允许获取非GROUP BY列的内容。

  例如，以下语法是非法的，因为b是非GROUP BY的列，在按照a进行GROUP BY时，有多行b可供选择，系统不知道该选择哪一行输出。

  ```
  *|select a, b , count(c) group by a
  ```

  为了达到以上目的，可以使用 `arbitrary` 函数输出b：

  ```
  *|select a, arbitrary(b), count(c) group by a
  ```

4.8.15. 窗口函数

本文档主要介绍窗口函数的语法。

窗口函数用来跨行计算。普通的SQL聚合函数只能用来计算一行内的结果，或者把所有行聚合成一行结果。窗口函数，可以跨行计算，并且把结果填到到每一行中。

窗口函数语法：

```
SELECT key1, key2, value,
       rank() OVER (PARTITION BY key2
                    ORDER BY value DESC) AS rnk
FROM orders
ORDER BY key1,rnk
```

核心部分是：

```
rank() OVER (PARTITION BY  KEY1 ORDER BY KEY2 DESC)
```

其中rank()是一个聚合函数，可以使用分析语法中的任何函数，也可以使用本文档列出的函数。PARTITION BY 是值按照哪些桶进行计算。

窗口中使用的特殊聚合函数

函数	含义
rank()	在窗口内，按照某一列排序，返回在窗口内的序号。
row_number()	返回在窗口内的行号。
first_value(x)	返回窗口内的第一个value，一般用法是窗口内数值排序，获取最大值。
last_value(x)	含义和first value相反。
nth_value(x, offset)	窗口内的第offset个数。
lead(x,offset,defaut_value)	窗口内x列某行之后offset行的值，如果不存在该行，则取default_value。
lag(x,offset,defaut_value)	窗口内x列某行之前offset行的值，如果不存在该行，则取default_value。

使用样例

- 在整个公司的人员中，获取每个人的薪水在部门内排名

```
* | select department, persionId, sallary , rank() over(PARTITION BY department order by sallary desc) as sallary_rank  order by department,sallary_rank
```

响应结果：

department	persionId	sallary	sallary_rank
dev	john	9000	1
dev	Smith	8000	2
dev	Snow	7000	3
dev	Achilles	6000	4
Marketing	Blan Stark	9000	1
Marketing	Rob Stark	8000	2
Marketing	Sansa Stark	7000	3

- 在整个公司的人员中，获取每个人的薪水在部门内的占比

```
* | select department, persionId, sallary *1.0 / sum(sallary) over(PARTITION BY department ) as sallary_percentage
```

响应结果：

department	persionId	sallary	sallary_percentage
dev	john	9000	0.3
dev	Smith	8000	0.26
dev	Snow	7000	0.23
dev	Achilles	6000	0.2

Marketing	Blan Stark	9000	0.375
Marketing	Rob Stark	8000	0.333
Marketing	Sansa Stark	7000	0.29

- 按天统计，获取每天UV相对前一天的增长情况

```
* | select day ,uv, uv *1.0 /(lag(uv,1,0) over() ) as diff_percentage from
(
select approx_distinct(ip) as uv, date_trunc('day',__time__) as day from log group by day order by
day asc
)
```

响应结果：

day	uv	diff_percentage
2017-12-01 00:00:00	100	null
2017-12-02 00:00:00	125	1.25
2017-12-03 00:00:00	150	1.2
2017-12-04 00:00:00	175	1.16
2017-12-05 00:00:00	200	1.14
2017-12-06 00:00:00	225	1.125
2017-12-07 00:00:00	250	1.11

4.8.16. HAVING语法

本文档主要为您介绍HAVING的语法。

日志服务查询分析功能支持标准SQL的HAVING语法，和GROUP BY配合使用，用于过滤GROUP BY的结果。

格式：

```
method :PostLogstoreLogs |select avg(latency),projectName group by projectName  HAVING avg(latency) > 10
0
```

HAVING和WHERE的区别

HAVING 用于过滤GROUP BY之后的聚合计算的结果，WHERE在聚合计算之间过滤原始数据。

示例

对于气温大于10℃的省份，计算每个省份的平均降雨量，并在最终结果中只显示平均降雨量大于100mL的省份：

```
* | select avg(rain) ,province where temperature > 10 group by province having avg(rain) > 100
```

4.8.17. ORDER BY语法

本文档主要为您介绍ORDER BY的语法。

ORDER BY 用于对输出结果进行排序，目前只支持按照一列进行排序。

- 语法格式

```
order by  列名 [desc|asc]
```

- 样例

```
method :PostLogstoreLogs |select avg(latency) as avg_latency,projectName group by projectName
HAVING avg(latency) > 5700000
order by avg_latency desc
```

4.8.18. LIMIT语法

LIMIT语法用于限制输出结果的行数。

语法格式

日志服务支持以下两种LIMIT语法格式。

- 只读取前N行

```
limit N
```

- 从S行开始读，读取N行

```
limit S , N
```

> **说明**
> - limit 翻页读取时，只用于获取最终的结果，不可用于获取SQL中间的结果。
> - 不支持将limit语法用于子查询内部。例如：
>
> ```
> * | select count(1) from ( select distinct(url) from limit 0,1000)
> ```
>
> - LIMIT翻页的offset不能超过1,000,000。即 `limit S , N` ，S和N之和不能超过1,000,000，N不能超出10,000。

示例

- 只获取100行结果

```
* | select distinct(url) from log limit 100
```

- 获取0行到第999行的结果，共计1000行

```
* | select distinct(url) from log limit 0,1000
```

- 获取第1000行到第1999行的结果，共计1000行

```
* | select distinct(url) from log limit 1000,1000
```

4.8.19. CASE WHEN和IF分支语法

本文档主要为您介绍CASE WHEN和IF分支的语法。

支持CASE WHEN语法，对连续数据进行归类。例如，从http_user_agent中提取信息，归类成Android和iOS两种类型：

```
SELECT
 CASE
 WHEN http_user_agent like '%android%' then 'android'
 WHEN http_user_agent like '%ios%' then 'ios'
 ELSE 'unknown' END
 as http_user_agent,
    count(1) as pv
    group by  http_user_agent
```

样例

- 计算状态码为200的请求占总体请求的比例

```
* | SELECT
  sum(
  CASE
  WHEN status =200 then 1
  ELSE 0 end
  ) *1.0 / count(1) as status_200_percentage
```

- 统计不同延时区间的分布

```
* | SELECT `
 CASE
 WHEN latency < 10 then 's10'
 WHEN latency < 100 then 's100'
 WHEN latency < 1000 then 's1000'
 WHEN latency < 10000 then 's10000'
 else 's_large' end
 as latency_slot,
 count(1) as pv
 group by latency_slot
```

IF语法

if语法逻辑上等同于CASE WHEN语法。

```
CASE
     WHEN condition THEN true_value
     [ ELSE false_value ]
 END
```

- if(condition, true_value)

  如果condition是true，则返回true_value这一列，否则返回null。
- if(condition, true_value, false_value)

  如果condition是true，则返回true_value这一列，否则返回false_value这一列。

COALESCE语法

coalesce 返回多个列的第一个非Null值。

```
coalesce(value1, value2[,...])
```

NULLIF 语法

如果value1和value2相等，返回null，否则返回value1。

```
nullif(value1, value2)
```

TRY 语法

try语法可以捕获一些底层的异常，例如除0错误，返回null值。

```
try(expression)
```

4.8.20. 嵌套子查询

本文档为您介绍如何在查询日志时进行嵌套查询。

针对一些复杂的查询场景，一层SQL无法满足需求，通过SQL嵌套查询可以满足复杂的需求。

嵌套子查询和无嵌套查询的区别在于，要在SQL中指定from 条件。在查询中要指定 `from log` 这个关键字，表示从日志中读取原始数据。

样例：

```
* | select sum(pv) from
(
select count(1) as pv from log group by method
)
```

4.8.21. 数组

本文档为您介绍数组相关函数的语法及示例。

语句	含义	示例
下标运算符[]	[]用于获取数组中的某个元素。	-
array_distinct	数组去重，获取数组中的唯一元素。	-
array_intersect(x, y)	获取x，y两个数组的交集。	-
array_union(x, y) → array	获取x，y两个数组的并集。	-
array_except(x, y) → array	获取x，y两个数组的差集	-
array_join(x, delimiter, null_replacement) → varchar	把字符串数组用delimiter连接，拼接成字符串，null值用null_replacement替代。 说明 使用array_join函数时，返回结果最大为1 KB，超出1 KB的数据会被截断。	-
array_max(x) → x	获取x中的最大值。	-
array_min(x) → x	获取x中的最小值。	-
array_position(x, element) → bigint	获取element在x中的下标，下标从1开始。如果找不到，则返回0。	-
array_remove(x, element) → array	从数组中移除element。	-
array_sort(x) → array	给数组排序，null值放到最后。	-
cardinality(x) → bigint	获取数组的大小。	-
concat(array1, array2, ..., arrayN) → array	连接数组。	-
contains(x, element) → boolean	如果x中包含element，则返回true。	-
filter(array, function) → array	function 是一个Lambda函数，请参见Lambda函数中的filter()。	-
flatten(x) → array	把二维的array拼接成一维的array。	-
reduce(array, initialState, inputFunction, outputFunction) → x	请参见Lambda函数reduce()。	-
reverse(x) → array	把x反向排列。	-
sequence(start, stop) → array	生成从start到stop结束的一个序列，每一步加1。	-
sequence(start, stop, step) → array	生成从start到stop结束的一个序列，每一步加step。	-
sequence(start, stop, step) → array	start和stop是timestamp类型，生成从start到stop结束的timestamp数组。step是INTERVAL类型，可以是DAY到SECOND，也可以是YEAR或MONTH。	-
shuffle(x) → array	重新随机分布array。	-
slice(x, start, length) → array	获取x数组从start开始，length个元素组成新的数组。	-
transform(array, function) → array	请参见Lambda函数transform()。	-
zip(array1, array2[, ...]) → array	合并多个数组。结果的第M个元素的第N个参数，是原始第N个数组的第M个元素，相当于把多个数组进行了转置。	`SELECT zip(ARRAY[1, 2], ARRAY['1b', null, '3b']); -- [ROW(1, '1b'), ROW(2, null), ROW(null, '3b')]`

zip_with(array1, array2, function) → array	请参见Lambda函数zip_with()。	-
array_agg (key)	array_agg (key)是一个聚合函数，表示把key这一列的所有内容变成一个array返回。	`* | select array_agg(key)`
array_transpose(array[array[*x*,*y*,*z*], array[*a*,*b*,*c*]])	对矩阵进行转置操作。	-

4.8.22. 二进制字符串函数

本文档为您介绍二进制字符串函数的语法及示例。

二进制字符串类型varbinary有别于字符串类型varchar。

语句	说明
连接函数 ||	`a || b` 结果为 `ab` 。
length(binary) → bigint	返回二进制的长度。
concat(binary1, ..., binaryN) → varbinary	连接二进制字符串，等同于||。
to_base64(binary) → varchar	把二进制字符串转换成base64。
from_base64(string) → varbinary	把base64转换成二进制字符串。
to_base64url(binary) → varchar	转化成url安全的base64。
from_base64url(string) → varbinary	从url安全的base64转化成二进制字符串。
to_hex(binary) → varchar	把二进制字符串转化成十六进制表示。
from_hex(string) → varbinary	从十六进制转化成二进制。
to_big_endian_64(bigint) → varbinary	把数字转化成大端表示的二进制。
from_big_endian_64(binary) → bigint	把大端表示的二进制字符串转化成数字。
md5(binary) → varbinary	计算二进制字符串的md5。
sha1(binary) → varbinary	计算二进制字符串的sha1。
sha256(binary) → varbinary	计算二进制字符串的sha256 hash。
sha512(binary) → varbinary	计算二进制字符串的sha512。
xxhash64(binary) → varbinary	计算二进制字符串的xxhash64。

4.8.23. 位运算

本文档为您介绍位运算函数的语法及示例。

语句	说明	示例

bit_count(x, bits) → bigint	统计x的二进制表示中，1的个数。	`SELECT bit_count(9, 64); — 2`
		`SELECT bit_count(9, 8); — 2`
		`SELECT bit_count(-7, 64); — 62`
		`SELECT bit_count(-7, 8); — 6`

bitwise_and(x, y) → bigint	以二进制的形式求x,y的and的值。	-
bitwise_not(x) → bigint	以二进制的形式求对x的所有位取反。	-

bitwise_or(x, y) → bigint	以二进制形式对x，y求or。	-
bitwise_xor(x, y) → bigint	以二进制形式对x，y求xor。	-

4.8.24. 同比和环比函数

日志服务支持使用同比环比函数对日志数据进行查询和分析。

同比和环比函数用于比较当前区间的计算结果和之前一个指定区间的结果。

函数	含义	样例
`compare(value, time_window)`	表示将当前时段计算出来的value值和time_window计算出来的结果进行比较。 value为double或long类型，time_window单位为秒；返回值为数组类型。 返回值分别是当前值、time_window之前的值和当前值与之前值的比值。	`*|select compare( pv , 86400) from (select count(1) as pv from log)`
`compare(value, time_window1, time_window2)`	表示当前区间分别和time_window1和time_window2之前的区间值进行比较，结果为json数组。其中，各个值的大小必须满足以下规则：[当前值，time_window1之前的值，time_window2之前的值，当前值/time_window1之前的值，当前值/time_window2之前的值]。	`* | select compare(pv, 86400, 172800) from ( select count(1) as pv from log)`
`compare(value, time_window1, time_window2, time_window3)`	表示当前区间分别和time_window1和time_window2，time_window3之前的区间值进行比较，结果为json数组。其中，各个值的大小必须满足以下规则：[当前值，time_window1之前的值，time_window2之前的值，time_window3之前的值，当前值/time_window1之前的值，当前值/time_window2之前的值，当前值/time_window3之前的值]。	`* | select compare(pv, 86400, 172800,604800) from ( select count(1) as pv from log)`
`ts_compare(value, time_window)`	表示当前区间分别和 `time_window1` 和 `time_window2` 之前的区间值进行比较，结果为json数组。其中，各个值的大小必须遵循以下规则：[当前值，time_window1之前的值，当前值/time_window1之前的值，前一个时间起点的unix时间戳]。 用于时序函数比较，需要在SQL中对时间列进行GROUP BY。	例如，`* | select t, ts_compare(pv, 86400 ) as d from(select date_trunc('minute',__time__ ) as t, count(1) as pv from log group by t order by t ) group by t` 表示将当前时间段每分钟的计算结果和上一个时间段每分钟的计算结果进行比较。 结果为：`d:[1251.0,1264.0, 0.9897151898734177, 1539843780.0,1539757380.0]t:2018-10-19 14:23:00.000` 。

示例

- 计算当前1小时和昨天同一时段的PV比例。

  开始时间为2018-7-25 14:00:00；结束时间为2018-07-25 15:00:00。

  查询分析语句：

  ```
  * | select compare( pv , 86400) from (select count(1) as pv from log)
  ```

  其中，86400表示当前时段减去86400秒。
  返回结果：

  ```
  [9.0,19.0,0.47368421052631579]
  ```

其中，

- 9.0表示从2018-7-25 14:00:00到2018-07-25 15:00:00的PV值。
- 19.0表示2018-7-24 14:00:00到2018-07-24 15:00:00的PV值。
- 0.47368421052631579表示当前时段与之前时段的比值。

如果要把数组展开成3列数字，分析语句为：

```
* | select diff[1],diff[2],diff[3] from(select compare( pv , 86400) as diff from (select count(1) as pv from log))
```

- 计算当前1小时内每分钟的PV和昨天同时段的PV比值，并以折线图展示。
  i. 计算当前1小时内每分钟的PV和昨天同时段的PV比值。开始时间为2018-7-25 14:00:00，结束时间为2018-07-25 15:00:00。

  查询分析语句：

  ```
  *| select t, compare( pv , 86400) as diff from (select count(1) as pv,
  date_format(from_unixtime(__time__), '%H:%i') as t from log group by t) group by t order by t
  ```

  返回结果：

t	diff
14:00	[9520.0,7606.0,1.2516434393899554]
14:01	[8596.0,8553.0,1.0050274757395066]
14:02	[8722.0,8435.0,1.0340248962655603]
14:03	[7499.0,5912.0,1.2684370771312586]

  其中t表示时间，格式为 `小时:分钟` 。diff列的内容是一个数组，分别表示：

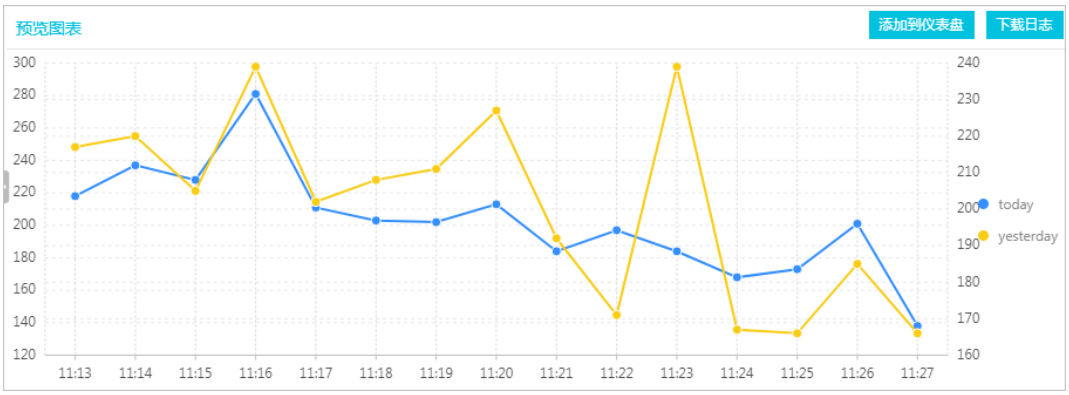
  - 当前时段的PV值。
  - 之前时段的PV值。
  - 当前时段PV值与之前时段比值。

  ii. 通过以下语句将查询结果展开为折线图形式：

  ```
  *|select t, diff[1] as current, diff[2] as yestoday, diff[3] as percentage from(select t, compare( pv , 86400) as diff from (select count(1) as pv, date_format(from_unixtime(__time__), '%H:%i') as t from log group by t) group by t order by t)
  ```

  将查询结果配置为折线图，两条线分别表示今天的值和昨天的值：

  图 1. 折线图



4.8.25. 比较函数和运算符

日志服务支持使用比较函数和运算符对日志数据进行查询和分析。

比较函数和运算符

比较运算判断参数的大小关系，可以应用于任何可比较类型，如int、bigint、double和text等。

比较运算符

比较运算符用于比较两个参数值的大小关系。当用比较运算符比较两个值时，如果逻辑成立，则返回true；否则返回false。

运算符	含义
<	小于
>	大于
<=	小于或等于
>=	大于或等于
=	等于
<>	不等于
!=	不等于

范围运算符 BETWEEN

BETWEEN用于判断一个参数的值是否在另外两个参数之间，范围为闭区间。

- 如果逻辑成立，则返回true；否则返回false。
  示例：`SELECT 3 BETWEEN 2 AND 6;` 逻辑成立，返回true。
  以上样例等同于`SELECT 3 >= 2 AND 3 <= 6;`。
- BETWEEN可以跟在not之后，用于相反逻辑的判断。
  示例：`SELECT 3 NOT BETWEEN 2 AND 6;`，逻辑不成立，返回false。
  以上样例等同于`SELECT 3 < 2 OR 3 > 6;`。
- 如果三个参数中任何一个包含Null，则返回的结果为Null。

IS NULL 和 IS NOT NULL

该运算符用于判断参数是否是Null值。

IS DISTINCT FROM 和 IS NOT DISTINCT FROM

类似于相等和不等判断，区别在于该运算符能够判断存在NULL值的情况。

样例：

```
SELECT NULL IS DISTINCT FROM NULL; -- false
SELECT NULL IS NOT DISTINCT FROM NULL; -- true
```

如下表所示，DISTINCT运算符可以判断多种情况下的参数大小关系。

a	b	a = b	a <> b	a DISTINCT b	a NOT DISTINCT b
1	1	TRUE	FALSE	FALSE	TRUE
1	2	FALSE	TRUE	TRUE	FALSE
1	NULL	NULL	NULL	TRUE	FALSE
NULL	NULL	NULL	NULL	FALSE	TRUE

GREATEST 和 LEAST

用于获取多列中的最大值或者最小值。

示例：

```
select greatest(1,2,3) ; -- 返回3
```

比较判断：ALL、ANY 和 SOME

比较判断用于判断参数是否满足条件。

- ALL用于判断参数是否满足所有条件。如果逻辑成立，则返回true，否则返回false。
- ANY用于判断参数是否满足条件之一。如果逻辑成立，则返回true，否则返回false。
- SOME和ANY一样，用于判断参数是否满足条件之一。
- ALL、ANY 和 SOME必须紧跟在比较运算符之后。

如下表所示，ALL和ANY支持多种情况下的比较判断。

表达式	含义
A = ALL (...)	A等于所有的值时，结果才是true。
A <> ALL (...)	A不等于所有的值时，结果才是true。
A < ALL (...)	A小于所有的值时，结果才是true。
A = ANY (...)	A等于任何一个值时，结果就为true，等同于 A IN (...)。
A <> ANY (...)	A不等于任何一个值时，结果为true。
A < ANY (...)	A小于其中最大值时，结果为true。

示例：

```
SELECT 'hello' = ANY (VALUES 'hello', 'world'); -- true
SELECT 21 < ALL (VALUES 19, 20, 21); -- false
SELECT 42 >= SOME (SELECT 41 UNION ALL SELECT 42 UNION ALL SELECT 43); -- true
```

4.8.26. Lambda函数

日志服务查询分析功能支持通过Lambda函数对日志数据进行分析处理。

Lambda表达式

Lambda表达式的形式为 `->` 。

样例如下：

```
x -> x + 1
(x, y) -> x + y
x -> regexp_like(x, 'a+')
x -> x[1] / x[2]
x -> IF(x > 0, x, -x)
x -> COALESCE(x, 0)
x -> CAST(x AS JSON)
x -> x + TRY(1 / 0)
```

大多数的MySQL表达式都可以在Lambda中使用。

filter(array<T>, function<T, boolean>) → ARRAY<T>

从array中过滤数据，只获取满足function且返回true的元素。

示例如下：

```
SELECT filter(ARRAY [], x -> true); -- []
SELECT filter(ARRAY [5, -6, NULL, 7], x -> x > 0); -- [5, 7]
SELECT filter(ARRAY [5, NULL, 7, NULL], x -> x IS NOT NULL); -- [5, 7]
```

map_filter(map<K, V>, function<K, V, boolean>) → MAP<K,V>

从map中过滤数据，只获取满足function且返回true的元素对。

示例如下：

```
SELECT map_filter(MAP(ARRAY[], ARRAY[]), (k, v) -> true); -- {}
SELECT map_filter(MAP(ARRAY[10, 20, 30], ARRAY['a', NULL, 'c']), (k, v) -> v IS NOT NULL); -- {10 -> a,
30 -> c}
SELECT map_filter(MAP(ARRAY['k1', 'k2', 'k3'], ARRAY[20, 3, 15]), (k, v) -> v > 10); -- {k1 -> 20, k3 ->
15}
```

reduce(array<T>, initialState S, inputFunction<S, T, S>, outputFunction<S, R>) → R

使用reduce函数，从初始状态开始，依次遍历array中的每一个元素，每次在状态S的基础上计算inputFunction(s,t)后生成新的状态。最终应用outputFunction把最终状态S变成输出结果R。详细步骤如下：

1. 初始状态S。
2. 遍历每个元素T。
3. 计算inputFunction(S,T)，生成新状态S。
4. 重复步骤2~步骤3，直到最后一个元素被遍历以及生成新状态。
5. 利用最终状态S，获取最终输出结果R。

示例如下：

```
SELECT reduce(ARRAY [], 0, (s, x) -> s + x, s -> s); -- 0
SELECT reduce(ARRAY [5, 20, 50], 0, (s, x) -> s + x, s -> s); -- 75
SELECT reduce(ARRAY [5, 20, NULL, 50], 0, (s, x) -> s + x, s -> s); -- NULL
SELECT reduce(ARRAY [5, 20, NULL, 50], 0, (s, x) -> s + COALESCE(x, 0), s -> s); -- 75
SELECT reduce(ARRAY [5, 20, NULL, 50], 0, (s, x) -> IF(x IS NULL, s, s + x), s -> s); -- 75
SELECT reduce(ARRAY [2147483647, 1], CAST (0 AS BIGINT), (s, x) -> s + x, s -> s); -- 2147483648
SELECT reduce(ARRAY [5, 6, 10, 20], -- calculates arithmetic average: 10.25
              CAST(ROW(0.0, 0) AS ROW(sum DOUBLE, count INTEGER)),
              (s, x) -> CAST(ROW(x + s.sum, s.count + 1) AS ROW(sum DOUBLE, count INTEGER)),
              s -> IF(s.count = 0, NULL, s.sum / s.count));
```

transform(array<T>, function<T, U>) → ARRAY<U>

对数组中的每个元素，依次调用function，生成新的结果U。

示例如下：

```
SELECT transform(ARRAY [], x -> x + 1); -- []
SELECT transform(ARRAY [5, 6], x -> x + 1); -- [6, 7] 表示对每个元素执行加1操作
SELECT transform(ARRAY [5, NULL, 6], x -> COALESCE(x, 0) + 1); -- [6, 1, 7]
SELECT transform(ARRAY ['x', 'abc', 'z'], x -> x || '0'); -- ['x0', 'abc0', 'z0']
SELECT transform(ARRAY [ARRAY [1, NULL, 2], ARRAY[3, NULL]], a -> filter(a, x -> x IS NOT NULL)); --
[[1, 2], [3]]
```

zip_with(array<T>, array<U>, function<T, U, R>) → array<R>

将两个array合并，根据元素T和元素U，通过函数生成新的array中的元素R。

示例如下：

```
SELECT zip_with(ARRAY[1, 3, 5], ARRAY['a', 'b', 'c'], (x, y) -> (y, x)); --表示调换前后两个数组的元素位置，生
成一个新的数组。结果： [['a', 1], ['b', 3],['c', 5]]
SELECT zip_with(ARRAY[1, 2], ARRAY[3, 4], (x, y) -> x + y); -- 结果[4, 6]
SELECT zip_with(ARRAY['a', 'b', 'c'], ARRAY['d', 'e', 'f'], (x, y) -> concat(x, y)); 表示把前后两个数组的元
素拼接，生成一个新的字符串。结果： ['ad', 'be', 'cf']
```

4.8.27. 逻辑函数

日志服务支持使用逻辑函数对日志数据进行查询和分析。

逻辑运算符

运算符	描述	样例

AND	只有左右运算数都是true时，结果才为true	a AND b
OR	左右运算数任一个为true，结果为true	a OR b
NOT	右侧运算数为false时，结果才为true	NOT a

NULL参与逻辑运算

a和b分别取值TRUE FALSE和NULL时的真值表如下：

表 1. 真值表1

a	b	a AND b	a OR b
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	NULL	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL
NULL	TRUE	NULL	TRUE
NULL	FALSE	FALSE	NULL
NULL	NULL	NULL	NULL

表 2. 真值表2

a	NOT a
TRUE	FALSE
FALSE	TRUE
NULL	NULL

4.8.28. 列的别名

日志服务采集日志时可以设置别名，本文主要介绍别名的设置规范和示例。

在SQL标准中，列名必须由字母、数字、下划线组成，且以字母开头。

如果在日志收集配置中，用户如果配置了不符合SQL标准的列名（例如User-Agent），那么需要在配置统计属性的页面，给列取一个别名，用于查询。别名仅仅用于SQL统计，在底层存储时，仍然是原始名称，搜索时需要使用原始名称。

此外，当用户原始的列名特别长时，也可以取一个别名来代替原始列名查询。

表 1. 别名样例

原始列名	别名
User-Agent	ua
User.Agent	ua
123	col
abceefghijklmnopqrstuvw	a

4.8.29. Logstore和RDS联合查询

日志服务支持Logstore中的日志数据和RDS数据库进行联合查询，以及把查询结果保存到RDS中。

操作步骤

1. 创建RDS VPC，并设置白名单，详情请参见**《云数据库RDS for SQL Server用户指南》 > 快速入门 > 创建实例、设置白名单**。
   创建RDS，并指定VPC环境。创建成功后，得到VPC ID和RDS实例ID。
2. 设置白名单。
   设置RDS白名单：`100.104.0.0/16`、`11.194.0.0/16`和`11.201.0.0/16`。
3. 创建External Store。
   通过以下语句创建External Store，请将参数替换为您的实际参数值。

```
{
"externalStoreName":"storeName",
"storeType":"rds-vpc",
"parameter":
   {
   "region":"cn-qingdao",
   "vpc-id":"vpc-m5eq4irc1pucp*******"
   "instance-id":"i-m5eeo2whsn*******"
   "host":"localhost",
   "port":"3306",
   "username":"root",
   "password":"****",
   "db":"scmc"
   "table":"join_meta"
   }
}
```

表 1. 参数说明

参数	说明
region	您的服务所在区域。
vpc-id	VPC的ID。
instance-id	RDS实例ID。
host	ECS实例ID。
port	ECS实例端口。
username	用户名。
password	密码。
db	数据库。
table	数据表。

> **说明** 目前仅支持北京（cn-beijing）、青岛（cn-qingdao）和杭州（cn-hangzhou）区域。

4. JOIN查询。
   在日志服务控制台**查询/分析**页面执行JOIN语句。
   支持的JOIN语法：
   - INNER JOIN
   - LEFT JOIN
   - RIGHT JOIN
   - FULL JOIN

```
[ INNER ] JOIN
LEFT [ OUTER ] JOIN
RIGHT [ OUTER ] JOIN
FULL [ OUTER ] JOIN
```

> 说明
> - 仅支持Logstore JOIN小表。
> - 在JOIN顺序中，Logstore必须写在前部，External Store写在后部。
> - JOIN中必须写External Store的名称，会自动替换成RDS的db+表名。不能直接填写RDS表名。

Join语法样例：

```
method:postlogstorelogs | select count(1) , histogram(logstore) from log  l join join_meta m on
l.projectid = cast( m.ikey as varchar)
```

5. 保存查询结果到RDS中。
支持通过Insert语法把查询结果插入到RDS中。

```
method:postlogstorelogs | insert into method_output  select
cast(methodasvarchar(65535)),count(1)fromloggroupbymethod
```

操作样例

Python 程序样例

```
# encoding: utf-8
from __future__ import print_function
from aliyun.log import *
from aliyun.log.util import base64_encodestring
from random import randint
import time
import os
from datetime import datetime
    endpoint = os.environ.get('ALIYUN_LOG_SAMPLE_ENDPOINT', 'cn-chengdu.log.aliyuncs.com')
    accessKeyId = os.environ.get('ALIYUN_LOG_SAMPLE_ACCESSID', '')
    accessKey = os.environ.get('ALIYUN_LOG_SAMPLE_ACCESSKEY', '')
    logstore = os.environ.get('ALIYUN_LOG_SAMPLE_LOGSTORE', '')
    project = "ali-yunlei-chengdu"
    client = LogClient(endpoint, accessKeyId, accessKey, token)
#创建external store
    res = client.create_external_store(project,ExternalStoreConfig("rds_store","region","rds-vpc","vpc i
d","实例id","实例ip","实例端口","用户名","密码","数据库","数据库表"));
    res.log_print()
    #获取external store详情
    res = client.get_external_store(project,"rds_store");
    res.log_print()
    res = client.list_external_store(project,"");
    res.log_print();
    # JOIN查询
    req = GetLogStoreLogsRequest(project,logstore,From,To,"","select count(1) from  "+ logstore +"  s jo
in  meta m on  s.projectid = cast(m.ikey as varchar)");
    res = client.get_logs(req)
    res.log_print();
     # 查询结果写入RDS
    req = GetLogStoreLogsRequest(project,logstore,From,To,"," insert into rds_store select count(1)
from  "+ logstore );
    res = client.get_logs(req)
    res.log_print();
```

4.8.30. 空间几何函数

日志服务支持使用空间几何函数对日志数据进行查询和分析。

空间几何概念

空间几何函数支持Well-Known Text（WKT） 格式描述的几何实体。

表 1. 几何实体格式

几何实体	Well-Known Text（WKT）格式
点	`POINT (0 0)`
线段	`LINESTRING (0 0, 1 1, 1 2)`
多边形	`POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0), (1 1, 2 1, 2 2, 1 2, 1 1))`
多点	`MULTIPOINT (0 0, 1 2)`
多线段	`MULTILINESTRING ((0 0, 1 1, 1 2), (2 3, 3 2, 5 4))`
多个多边形	`MULTIPOLYGON (((0 0, 4 0, 4 4, 0 4, 0 0), (1 1, 2 1, 2 2, 1 2, 1 1)), ((-1 -1, -1 -2, -2 -2, -2 -1, -1 -1)))`
空间实体集合	`GEOMETRYCOLLECTION (POINT(2 3), LINESTRING (2 3, 3 4))`

构造空间实体

表 2. 构造空间实体函数说明

函数	说明
ST_Point(double, double) → Point	构造一个点。
ST_LineFromText(varchar) → LineString	从WKT格式的文本中构造一个线段。
ST_Polygon(varchar) → Polygon	从WKT格式的文本中构造一个多边形。
ST_GeometryFromText(varchar) → Geometry	从WKT文本中构造一个空间几何实体。
ST_AsText(Geometry) → varchar	把一个空间几何实体转变成WKT格式。

运算符

函数	说明
ST_Boundary(Geometry) → Geometry	计算几何实体的闭包。
ST_Buffer(Geometry, distance) → Geometry	返回一个多边形，该多边形距离输入参数Geometry的距离是distance。
ST_Difference(Geometry, Geometry) → Geometry	返回两个空间实体的不同的点的集合。
ST_Envelope(Geometry) → Geometry	返回空间实体的边界多边形。
ST_ExteriorRing(Geometry) → Geometry	返回多边形的外部环。
ST_Intersection(Geometry, Geometry) → Geometry	返回两个空间实体的交集点。
ST_SymDifference(Geometry, Geometry) → Geometry	返回两个空间实体不同的点，组成的新的空间实体。获取两个几何对象不相交的部分。

空间关系判断

函数	说明
ST_Contains(Geometry, Geometry) → boolean	当第二个实体的所有点都不在第一个实体外部，并且第一个实体至少有一个内部点在第二个实体内部时，返回true。如果第二个实体正好在第一个实体的边上，那么是false。
ST_Crosses(Geometry, Geometry) → boolean	当两个实体有共同内部点时，返回true。

ST_Disjoint(Geometry, Geometry) → boolean	当两个实体没有任何交集时，返回true。
ST_Equals(Geometry, Geometry) → boolean	当两个实体完全相同时，返回true。
ST_Intersects(Geometry, Geometry) → boolean	当两个实体在两个空间上共享时，返回true。
ST_Overlaps(Geometry, Geometry) → boolean	当两个实体维度相同，并且不是包含关系时，返回true。
ST_Relate(Geometry, Geometry) → boolean	当两个实体相关时，返回true。
ST_Touches(Geometry, Geometry) → boolean	当两个实体仅仅边界有联系，没有共同内部点时，返回true。
ST_Within(Geometry, Geometry) → boolean	当第一个实体完全在第二个实体内部时，返回true。如果边界有交集，返回false。

Accessors

函数	说明
ST_Area(Geometry) → double	使用欧几里得测量法，计算多边形在二维平面上的投影面积。
ST_Centroid(Geometry) → Geometry	返回几何实体的中心点。
ST_CoordDim(Geometry) → bigint	返回几何实体的坐标维度。
ST_Dimension(Geometry) → bigint	返回几何实体的固有维度，必须小于或等于坐标维度。
ST_Distance(Geometry, Geometry) → double	计算两个实体之间的最小距离。
ST_IsClosed(Geometry) → boolean	当实体时一个闭合空间时，返回true。
ST_IsEmpty(Geometry) → boolean	当参数时一个空的几何实体集合或者多边形或者点时返回true。
ST_IsRing(Geometry) → boolean	当参数是一条线，并且时闭合的简单的线时，返回true。
ST_Length(Geometry) → double	在二维投影平面上，使用欧几里得测量法计算一个线段或者多条线段的长度。返回一个行字符串或多行字符串的长度。该长度是采用欧几里得测量法基于空间参考对二维平面的预测。
ST_XMax(Geometry) → double	返回几何体边框的X最大值。
ST_YMax(Geometry) → double	返回几何体边框的Y最大值。
T_XMin(Geometry) → double	返回几何体边框的X最小值。
ST_YMin(Geometry) → double	返回结合体边框的Y最小值。
ST_StartPoint(Geometry) → point	返回线段类型几何体的第一个点。
ST_EndPoint(Geometry) → point	返回线段类型几何体的最后一个点。
ST_X(Point) → double	返回点类型的X轴。
ST_Y(Point) → double	返回点类型的Y轴。
ST_NumPoints(Geometry) → bigint	计算几何实体的点的个数。
ST_NumInteriorRing(Geometry) → bigint	返回多边形内部的环的个数。

4.8.31. 地理函数

本文档为您介绍地理函数的用法。

IP转国家、省、城市、运营商、经纬度，请参考文档IP地理函数。 表 1. 地理函数

函数名	含义	样例

geohash(string)	将纬度、经度用geohash编码，string为字符串类型，内容是纬度、逗号、经度。	`select geohash('34.1,120.6')= 'wwjcbrdnzs'`
geohash(lat,lon)	将纬度、经度用geohash编码，参数分别是纬度和经度。	`select geohash(34.1,120.6)= 'wwjcbrdnzs'`

4.8.32. Join语法

Join用于多表中字段之间的联系。日志服务支持单个Logstore的Join语法，还支持Logstore和RDS之间的Join语法，以及跨Logstore的Join语法。本文介绍如何使用跨Logstore的Join功能。

操作步骤

1. 下载最新版本Python SDK。
2. 使用GetProjectLogs接口进行查询。

SDK示例

```
#!/usr/bin/env python
#encoding: utf-8
import time,sys,os
from aliyun.log.logexception import LogException
from aliyun.log.logitem import LogItem
from aliyun.log.logclient import LogClient
from aliyun.log.getlogsrequest import GetLogsRequest
from aliyun.log.getlogsrequest import GetProjectLogsRequest
from aliyun.log.putlogsrequest import PutLogsRequest
from aliyun.log.listtopicsrequest import ListTopicsRequest
from aliyun.log.listlogstoresrequest import ListLogstoresRequest
from aliyun.log.gethistogramsrequest import GetHistogramsRequest
from aliyun.log.index_config import *
from aliyun.log.logtail_config_detail import *
from aliyun.log.machine_group_detail import *
from aliyun.log.acl_config import *
if __name__=='__main__':
    token = None
    endpoint = "http://cn-hangzhou.log.aliyuncs.com"
    accessKeyId  = '******'
    accessKey = '******'
    client = LogClient(endpoint, accessKeyId, accessKey,token)
    logstore = "meta"
    # 在查询和分析语句中，指定两个Logstore，为每个Logstore指定各自的时间范围，并指定两个Logstore关联的key。
    req = GetProjectLogsRequest(project,"select count(1) from sls_operation_log  s join  meta m on s.__d
ate__ >'2018-04-10 00:00:00' and s.__date__ < '2018-04-11 00:00:00' and m.__date__ >'2018-04-23
00:00:00' and m.__date__ <'2018-04-24 00:00:00' and s.projectid = cast(m.ikey as varchar)");
    res = client.get_project_logs(req)
    res.log_print();
    exit(0)
```

> **说明**
>
> 关于Join语法及示例请参见Join。

4.8.33. unnest语法

本文档主要为您介绍unnest语法。

应用场景

处理数据时，一列数据通常为字符串或数字等primitive类型的数据。在复杂的业务场景下，日志数据的某一列可能会是较为复杂的格式，例如数组（array）、对象（map）、JSON等格式。对这种特殊格式的日志字段进行查询分析，可以使用unnest语法。

例如以下日志：

```
__source__:  1.1.1.1
__tag__:__hostname__:  vm-req-170103232316569850-tianchi111932.tc
__topic__:  TestTopic_4
array_column:  [1,2,3]
double_column:  1.23
map_column:  {"a":1,"b":2}
text_column:  商品
```

其中 `array_column` 字段为数组类型。如果统计 `array_column` 中所有数值的汇总值，需要遍历每一行的数组中的每一个元素。

unnest语法结构

语法	说明
`unnest( array) as table_alias(column_name)`	表示把array类型展开成多行，行的名称为 `column_name` 。
`unnest(map) as table(key_name, value_name)`	表示把map类型展开成多行，key的名称为 `key_name` ，value的名称为 `value_name` 。

> **说明** 注意，由于unnest接收的是array或者map类型的数据，如果您的输入为字符串类型，那么要先转化成json类型，然后再转化成array类型或map类型，转化的方式为 `cast(json_parse(array_column) as array(bigint))` 。

遍历数组每一个元素

使用SQL把array展开成多行：

```
* | select  array_column, a  from log, unnest( cast( json_parse(array_column)  as array(bigint) ) ) as t(a)
```

上述SQL把数组展开成多行数字，`unnest( cast( json_parse(array_column) as array(bigint) ) ) as t(a)` ，unnest语法把数组展开，以t来命名新生成的表，使用a来引用展开后的列。

- 统计数组中的每个元素的和：

  ```
  * | select  sum(a)  from log, unnest( cast( json_parse(array_column)  as array(bigint) ) ) as  t(a)
  ```

- 按照数组中的每个元素进行group by计算：

  ```
  * | select  a, count(1)  from log, unnest( cast( json_parse(array_column)  as array(bigint) ) ) as t(a)  group by a
  ```

遍历Map

- 遍历Map中的元素

  ```
  * | select  map_column , a,b  from log, unnest( cast( json_parse(map_column)  as map(varchar, bigint) ) ) as  t(a,b)
  ```

- 按照Map的key进行group by 统计

  ```
  * | select  key,  sum(value)  from log, unnest( cast( json_parse(map_column)  as map(varchar, bigint) ) ) as  t(key,value)  GROUP  BY  key
  ```

格式化显示histogram,numeric_histogram的结果

- histogram

  histogram函数类似于count group by 语法。语法请参见Map映射函数。

  通常情况下histogram的结果为一串json数据，无法配置视图展示，例如：

  ```
  * | select histogram(method)
  ```

您可以通过unnest语法，把JSON展开成多行配置视图，例如：

```
* | select  key , value  from( select histogram(method) as his from log) , unnest(his ) as t(key,value )
```

- numeric_histogram
  numeric_histogram语法是为了把数值列分配到多个桶中去，相当于对数值列进行group by，具体语法请参见估算函数。

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

您可以通过以下查询语句格式化展示该结果：

```
* |  select key,value from(select numeric_histogram(10,Latency) as his from log) , unnest(his) as t(key,value)
```

4.8.34. 电话号码函数

电话号码函数提供对中国大陆区域电话号码的归属地查询功能。

函数列表

函数名	含义	样例
mobile_province	查看电话号码所属省份，需要传入电话的数字形式。字符串参数可以使用 `try_cast` 进行转换。	`* | select mobile_province(12345678)` `* | select mobile_province(try_cast('12345678' as bigint) )`
mobile_city	查看电话号码所属城市，需要传入电话的数字形式。字符串参数可以使用 `try_cast` 进行转换。	`* | select mobile_city(12345678)` `* | select mobile_city(try_cast('12345678' as bigint) )`
mobile_carrier	查看电话号码所属运营商，需要传入电话的数字形式。字符串参数可以使用 `try_cast` 进行转换。	`* | select mobile_carrier(12345678)` `* | select mobile_carrier(try_cast('12345678' as bigint) )`

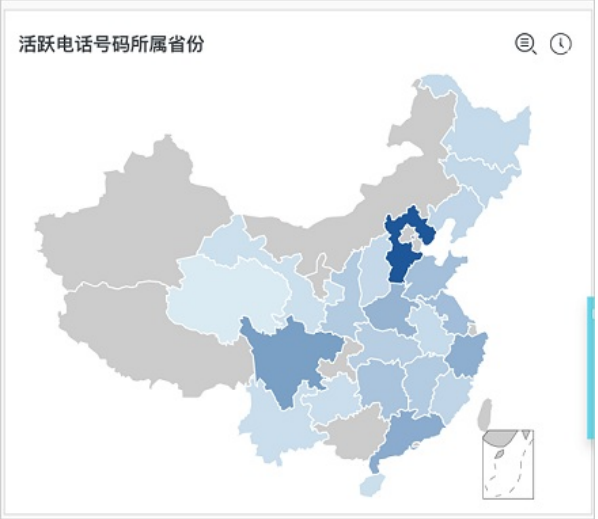
应用场景

- 查询电话号码所属地并生成报表
  某电商收集客户参加活动的日志信息，其中有用户电话号码的字段，对电话号码归属地进行统计，可以实现如下查询分析语句：

```
SELECT mobile_city(try_cast("mobile" as bigint)) as "城市", mobile_province(try_cast("mobile" as bigint)) as "省份", count(1) as "请求次数" group by "省份", "城市" order by "请求次数" desc limit 100
```

这里将日志中的 `mobile` 字段传给了 `mobile_city` 和 `mobile_province` 函数，展示其所在省和城市等信息。返回如下：

还可以选择地图视图，进行可视化如下：



- 根据电话所属地检查并通知
  例如，某证券运营商收集了根据客户的电话号码所属地，及其访问服务时的IP地址，想要整理出哪些客户平时访问地址与电话所属地不同：

```
* | select mobile, client_ip, count(1) as PV where mobile_city(try_cast("mobile" as bigint)) != ip_to_city(client_ip) and ip_to_city(client_ip) != '' group by client_ip, mobile order by PV desc
```

  可以以此创建告警规则，详细说明请参见日志服务告警。

4.9. 机器学习语法与函数

4.9.1. 简介

日志服务（Log Service）机器学习功能为您提供多种功能丰富的算法和便捷的调用方式，您可以在日志查询分析中通过SELECT语句和机器学习函数调用机器学习算法，分析某一字段或若干字段在一段时间内的特征。

尤其是针对时序数据分析场景，日志服务提供了丰富的时序分析算法，可以帮助您快速解决时序预测、时序异常检测、序列分解、多时序聚类等场景问题，兼容SQL标准接口，大大降低了您使用算法的门槛，提高分析问题和解决问题的效率。

功能特点

- 支持单时序序列的多种平滑操作。
- 支持单时序序列的预测、异常检测、变点检测、折点检测、多周期估计算法。
- 支持单时序序列的分解操作。
- 支持多时序序列的多种聚类算法。
- 支持多字段（数值列、文本列）的模式挖掘。

限制说明

- 输入的时序数据必须是基于相同时间间隔的采样数据。
- 输入的时序数据中不能含有重复时间点的数据。

限制项	说明
时序数据处理的有效容量	上限为150,000个连续时间点数据。 若数量超过上限，请进行聚合操作或者降采样。
密度聚类算法的聚类容量	上限为5000条时序曲线，每条时序曲线的长度最大为1440个点。
层次聚类算法的聚类容量	上限为2000条时序曲线，每条时序曲线的长度最大为1440个点。

机器学习函数

类别	函数	说明

平滑函数	ts_smooth_simple	使用Holt Winters算法对时序数据平滑。
	ts_smooth_fir	使用FIR滤波器对时序数据平滑。
	ts_smooth_iir	使用IIR滤波器对时序数据平滑。
多周期估计函数	ts_period_detect	对时序数据进行分段周期估计。
变点检测函数	ts_cp_detect	寻找时序序列中具有不同统计特性的区间，区间端点即为变点。
	ts_breakout_detect	寻找时序序列中，某统计量发生陡升或陡降的点。
极大值检测函数	ts_find_peaks	极大值检测函数用于在指定窗口中寻找序列的局部极大值。
预测与异常检测函数	ts_predicate_simple	利用默认参数对时序数据进行建模，并进行简单的时序预测和异常点的检测。
	ts_predicate_ar	使用自回归模型对时序数据进行建模，并进行简单的时序预测和异常点的检测。
	ts_predicate_arma	使用移动自回归模型对时序数据进行建模，并进行简单的时序预测和异常点检测。
	ts_predicate_arima	使用带有差分的移动自回归模型对时序数据进行建模，并进行简单的时序预测和异常点检测。
	ts_regression_predict	针对含有周期性、趋势性的单时序序列，进行准确且长时序预测。
序列分解函数	ts_decompose	使用STL算法对时序数据进行序列分解。
时序聚类函数	ts_density_cluster	使用密度聚类方法对多条时序数据进行聚类。
	ts_hierarchical_cluster	使用层次聚类方法对多条时序数据进行聚类。
	ts_similar_instance	查找到指定曲线名称的相似曲线。
频繁模式统计函数	pattern_stat	统计模式中的频繁模式，在给定的多属性字段样本中，挖掘出具有一定代表性的属性组合。
差异模式统计函数	pattern_diff	在指定条件下找出导致两个集合差异的模式。
根因分析函数	rca_kpi_search	在时序指标发生异常时，根因分析函数可以快速分析出是哪些相关维度属性发生异常而导致监控指标发生异常。
相关性分析函数	ts_association_analysis	针对系统中的多个观测指标，快速找出和某个指标项相关的指标名称。
ts_similar	针对系统中的多个观测指标，快速找出和用户输入的时序序列相关的指标名称。	
核密度估计函数	kernel_density_estimation	采用平滑的峰值函数来拟合观察到的数据点，从而对真实的概率分布曲线进行模拟。

4.9.2. 平滑函数

平滑函数是针对输入的时序曲线进行平滑和简单的滤波操作，滤波操作通常是发现时序曲线形态的第一步。

函数列表

函数	说明
`ts_smooth_simple`	默认平滑函数，使用Holt Winters算法对时序数据进行滤波操作。
`ts_smooth_fir`	使用FIR滤波器对时序数据进行滤波操作。
`ts_smooth_iir`	使用IIR滤波器对时序数据进行滤波操作。

ts_smooth_simple

- 函数格式：

```
select ts_smooth_simple(x, y)
```

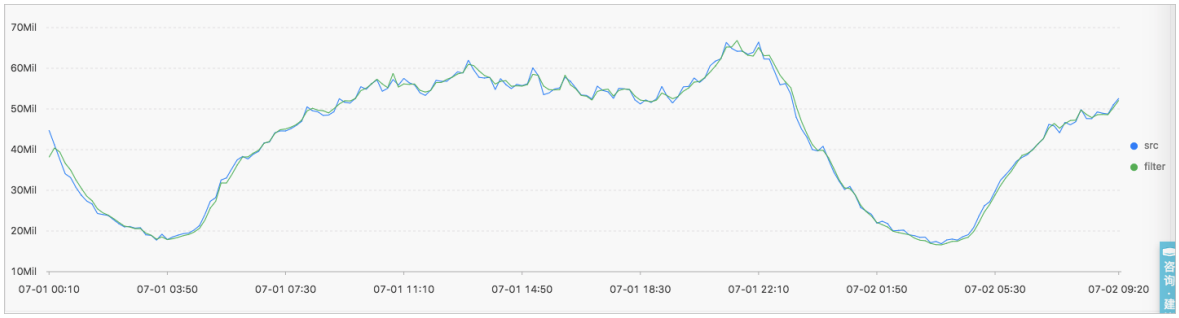
- 参数说明：

参数	说明	取值
x	时间列，顺序为从小到大。	Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-

- 示例
  - 查询分析语句：

```
* | select ts_smooth_simple(stamp, value) from ( select __time__ - __time__ % 120 as stamp, avg(v)
as value from log GROUP BY stamp order by stamp )
```

  - 输出结果：



- 显示项：

显示项		说明
横轴	unixtime	数据的时间戳，单位为秒。
纵轴	src	未滤波前的数据。
	filter	滤波之后的数据。

ts_smooth_fir

- 函数格式：
  - 若您无法确定滤波参数，请使用内置窗口的参数进行滤波操作。

```
select ts_smooth_fir(x, y,winType,winSize)
```

  - 若您可以确定滤波参数，可以根据需求自定义设置滤波参数。

```
select ts_smooth_fir(x, y,array[])
```

- 参数说明：

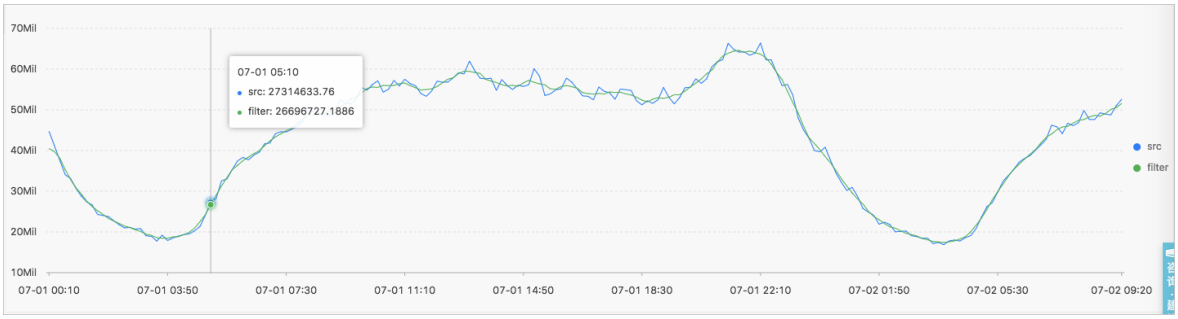
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-

winType	滤波的窗口类型。	取值包括： ◦ rectangle：矩形窗口。 ◦ hanning：汉宁窗。 ◦ hamming：汉明窗。 ◦ blackman：布莱克曼窗。 说明 推荐您选择rectangle类型以获得更好的显示效果。
winSize	滤波窗口的长度。	long类型，取值范围为2~15。
array[]	FIR滤波的具体参数。	格式为数组，且数组中元素的和为1。例如array[0.2, 0.4, 0.3, 0.1]。

- 示例1
  - 查询分析语句：

```
* | select ts_smooth_fir(stamp, value, 'rectangle', 4) from ( select __time__ - __time__ % 120 as stamp, avg(v) as value from log GROUP BY stamp order by stamp )
```

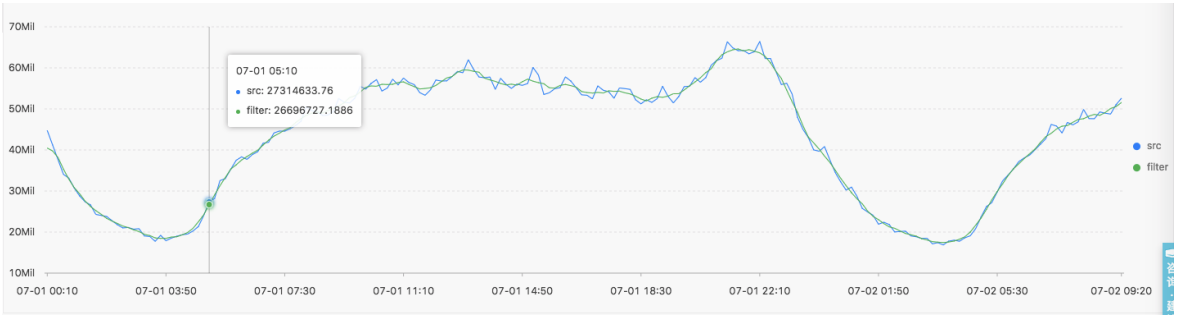
  - 输出结果：



- 示例2
  - 查询分析语句：

```
* | select ts_smooth_fir(stamp, value, array[0.2, 0.4, 0.3, 0.1]) from ( select __time__ - __time__ % 120 as stamp, avg(v) as value from log GROUP BY stamp order by stamp )
```

  - 输出结果：



- 显示项：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
纵轴	src	未滤波前的数据。
	filter	滤波之后的数据。

ts_smooth_iir

- 函数格式：

```
select ts_smooth_iir(x, y, array[], array[] )
```

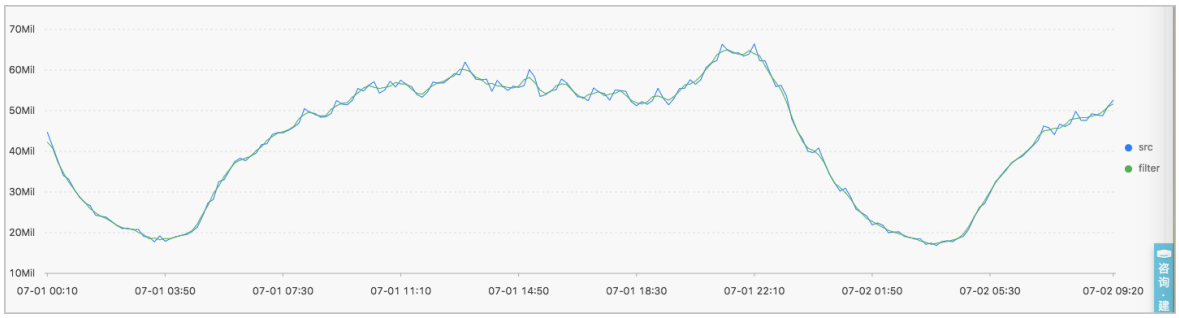
- 参数说明：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
array[]	IIR滤波算法中关于x_i的具体参数。	数组格式，长度（length）的取值范围为2~15，且数组中元素的和为1。例如array[0.2, 0.4, 0.3, 0.1]。
array[]	IIR滤波算法中关于y_{i-1}的具体参数。	数组格式，长度（length）的取值范围为2~15，且数组中元素的和为1。例如array[0.2, 0.4, 0.3, 0.1]。

- 示例
  - 查询分析语句：

```
* | select ts_smooth_iir(stamp, value, array[0.2, 0.4, 0.3, 0.1], array[0.4, 0.3, 0.3]) from ( selec
t __time__ - __time__ % 120 as stamp, avg(v) as value from log GROUP BY stamp order by stamp )
```

  - 输出结果：



- 显示项：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
纵轴	src	未滤波前的数据。
	filter	滤波之后的数据。

4.9.3. 多周期估计函数

多周期估计函数支持对不同时间段内的时序进行周期估计，通过傅立叶变换等一系列操作进行周期的提取。

函数列表

函数	说明
ts_period_detect	对不同时间段内的时序数据进行周期估计。
ts_period_classify	通过傅立叶变换，计算输入时序曲线的周期性。该方法可以较好的用于快速判别曲线的周期性。

ts_period_detect

函数格式：

```
select ts_period_detect(x,y,minPeriod,maxPeriod)
```

参数说明如下：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
minPeriod	预估计周期最小长度占序列总长度的比例。	小数形式，取值范围为(0,1]。
maxPeriod	预估计周期最大长度占序列总长度的比例。 说明 指定参数时，*maxPeriod*必须大于*minPeriod*。	小数形式，取值范围为(0,1]。

示例：

- 查询分析

```
* | select ts_period_detect(stamp, value, 0.2, 1.0) from ( select __time__ - __time__ % 120 as stamp, avg(v) as value from log GROUP BY stamp order by stamp )
```

- 输出结果

显示项如下：

显示项	说明
period_id	周期编号的数组，长度为1，其中值为0时，表示原始序列。
time_series	表示时间戳序列。
data_series	表示每个时间戳对应的结果。 • 当period_id为0时，表示原始序列结果。 • 当period_id不为0时，表示滤波之后的周期估计结果。

ts_period_classify

函数格式：

```
select ts_period_classify(stamp,value,instanceName)
```

参数说明如下：

参数	说明	取值
stamp	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
value	数值列，对应某时刻的数据。	-
instanceName	曲线对应的名称。	-

示例：

- 查询分析

```
* and h : nu2h05202.nu8 | select ts_period_classify(stamp, value, name) from log
```

- 输出结果

显示项如下：

显示项	说明
line_name	周期编号的数组，长度为1，其中值为0时，表示原始序列。
prob	时序曲线中主周期的占比，范围为[0, 1]，实验中可以取0.15。
type	曲线的类别： • type = -1：表示曲线长度太短（小于64个点）。 • type = -2：表示曲线缺失率很高（缺失率超过20%）。 • type = 0：表示曲线有明显的周期性。

4.9.4. 变点检测函数

变点检测函数一般用于对时序数据中的变点进行检测。

变点检测函数支持对如下两种变点形态进行检测：

- 指定时间段内的某些统计特性发生了变化。
- 序列数据中存在较为明显的断层。

函数列表

函数	说明
ts_cp_detect	寻找时序序列中具有不同统计特性的区间，区间端点即为变点。
ts_breakout_detect	寻找时序序列中，某统计量发生陡升或陡降的点。

ts_cp_detect

函数格式：

- 若您无法确定窗口大小，可以使用如下格式的函数，该函数调用的算法默认会使用长度为10的窗口进行检测。

```
select ts_cp_detect(x, y, samplePeriod)
```

- 若您需要根据业务曲线进行效果调试，可以使用如下格式的函数，通过设置参数minSize进行效果调试。

```
select ts_cp_detect(x, y, minSize)
```

参数说明如下：

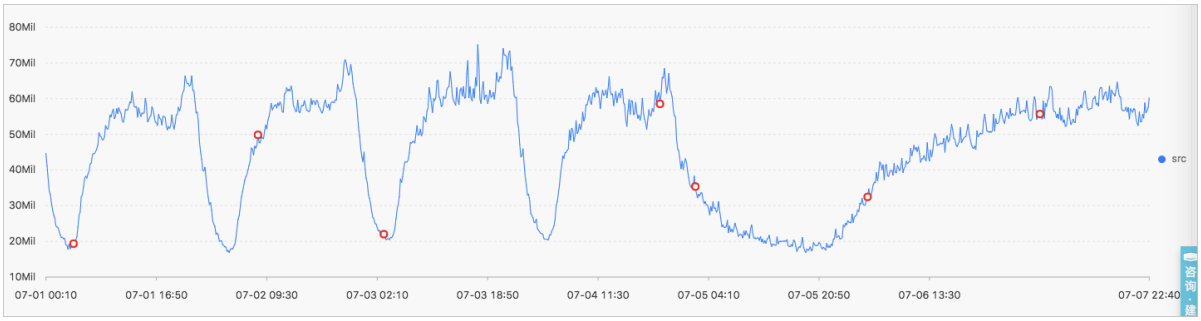
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
minSize	最小连续区间长度。	最小值为3，最大值不超过当前输入数据长度的1/10。

示例：

- 查询分析

```
* | select ts_cp_detect(stamp, value, 3) from (select __time__ - __time__ % 10 as stamp, avg(v) as
value from log GROUP BY stamp order by stamp)
```

- 输出结果



显示项如下：

显示项		说明
横轴	unixtime	数据的时间戳，单位为秒，例如1537071480。
纵轴	src	未滤波前的数据，例如1956092.7647745228。
	prob	该点为变点的概率，取值范围为0~1。

ts_breakout_detect

函数格式：

```
select ts_breakout_detect(x, y, winSize)
```

参数说明如下：

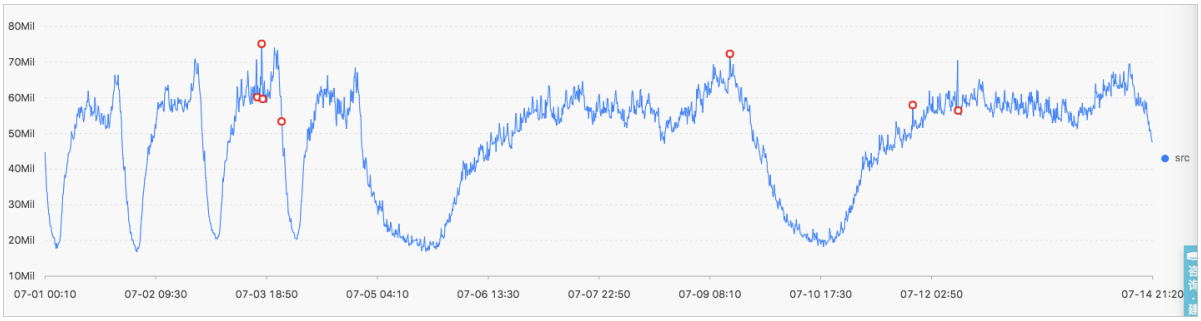
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
winSize	最小连续区间长度。	最小值为3，最大值不超过当前输入数据长度的1/10。

示例：

- 查询分析

```
* | select ts_breakout_detect(stamp, value, 3) from (select __time__ - __time__ % 10 as stamp, avg(v)
as value from log GROUP BY stamp order by stamp)
```

- 输出结果



显示项如下：

显示项		说明
横轴	unixtime	数据的时间戳，单位为秒，例如1537071480。
纵轴	src	未滤波前的数据，例如1956092.7647745228。

	prob	该点为变点的概率，取值范围为0~1。

4.9.5. 极大值检测函数

极大值检测函数用于在指定窗口中寻找序列的局部极大值。

ts_find_peaks

函数格式：

```
select ts_find_peaks(x, y, winSize)
```

参数说明如下：

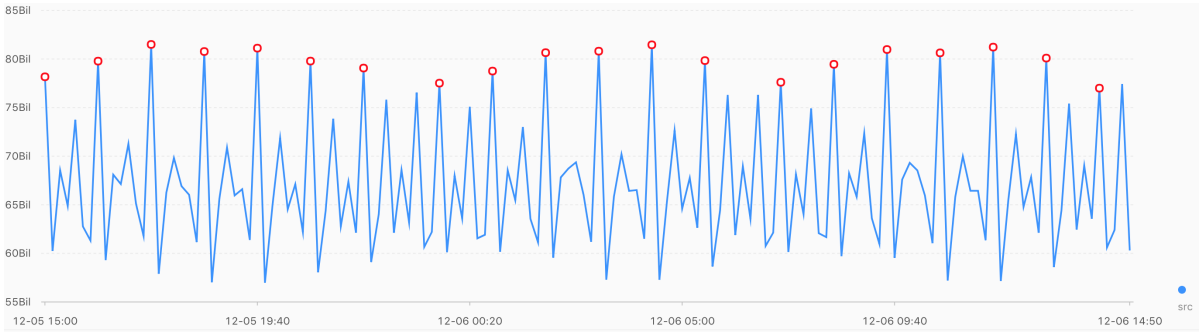
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
winSize	指定最小的检测窗口长度。	long类型，取值范围为大于等于1，小于等于数值的实际长度。建议指定该参数的值为数据实际长度的十分之一。

示例：

- 查询分析：

```
* and h : nu2h05202.nu8 and m: NET |  select ts_find_peaks(stamp, value, 30) from (select __time__ - __time__ % 10 as stamp, avg(v) as value from log GROUP  BY  stamp order by stamp)
```

- 输出结果：



显示项如下：

显示项		说明
横轴	unixtime	数据的时间戳，单位为秒，例如1537071480。
纵轴	src	未滤波前的数据，例如1956092.7647745228。
	peak_flag	该点是否为极大值，其中： • 1.0：表示该点为极大值。 • 0.0：表示该点不是极大值。

4.9.6. 预测与异常检测函数

预测与异常检测函数通过预测时序曲线、寻找预测曲线和实际曲线之间误差的Ksigma与分位数等特性进行异常检测。

关于函数的算法及原理请参见：

- LOG机器学习介绍（01）：时序统计建模
- LOG机器学习介绍（03）：时序异常检测建模
- LOG机器学习介绍（05）：时间序列预测

- LOG机器学习最佳实战：时序异常检测和报警

函数列表

函数	说明
ts_predicate_simple	利用默认参数对时序数据进行建模，并进行简单的时序预测和异常点的检测。
ts_predicate_ar	使用自回归模型对时序数据进行建模，并进行简单的时序预测和异常点的检测。
ts_predicate_arma	使用移动自回归模型对时序数据进行建模，并进行简单的时序预测和异常点检测。
ts_predicate_arima	使用带有差分的移动自回归模型对时序数据进行建模，并进行简单的时序预测和异常点检测。
ts_regression_predict	针对具有周期性、趋势性的单时序序列，进行准确的预测。 使用场景：计量数据的预测、网络流量的预测、财务数据的预测、以及具有一定规律的不同业务数据的预测。
ts_anomaly_filter	针对批量曲线进行时序异常检测后，可以按照用户定义的异常模式来过滤异常检测的结果。能帮助用户快速找出异常的实例曲线。

ts_predicate_simple

函数格式：

```
select ts_predicate_simple(x, y, nPred, isSmooth)
```

参数说明如下：

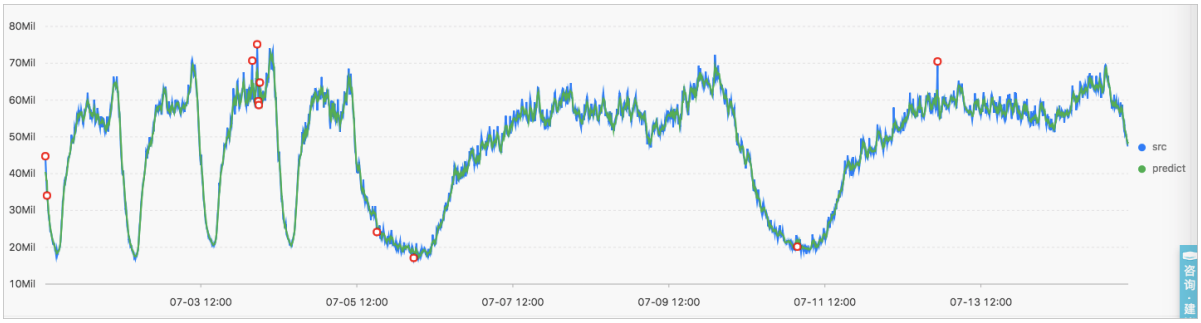
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
nPred	预测未来的点的数量。	long类型，取值大于等于1。
isSmooth	是否需要对原始数据做滤波操作。	bool类型，默认为true表示对原始数据做滤波操作。

示例：

- 查询分析

```
* | select ts_predicate_simple(stamp, value, 6) from (select __time__ - __time__ % 60 as stamp,
avg(v) as value from log GROUP BY stamp order by stamp)
```

- 输出结果



显示项如下：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
	src	原始数据。

纵轴	predict	预测的数据。
	upper	预测的上界。当前置信度为0.85，不可修改。
	lower	预测的下界。当前置信度为0.85，不可修改。
	anomaly_prob	该点为异常点的概率，范围为0~1。

ts_predicate_ar

函数格式：

```
select ts_predicate_ar(x, y, p, nPred, isSmooth)
```

参数说明如下：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
p	自回归模型的阶数。	long类型，取值范围为2~8。
nPred	预测未来的点的数量。	long类型，取值范围为1~5*p。
isSmooth	是否需要对原始数据做滤波操作。	bool类型，默认为true表示对原始数据做滤波操作。

查询分析示例：

```
* | select ts_predicate_ar(stamp, value, 3, 4) from (select __time__ - __time__ % 60 as stamp, avg(v)
as value from log GROUP BY stamp order by stamp)
```

> **说明** 输出结果与ts_predicate_simple函数相似，具体请参见ts_predicate_simple函数的输出结果。

ts_predicate_arma

函数格式：

```
select ts_predicate_arma(x, y, p, q, nPred, isSmooth)
```

参数说明如下：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
p	自回归模型的阶数。	long类型，取值范围为2~100。
q	移动平均模型的阶数。	long类型，取值范围为2~8。
nPred	预测未来的点的数量。	long类型，取值范围为1~5*p。
isSmooth	是否需要对原始数据做滤波操作。	bool类型，默认为true表示对原始数据做滤波操作。

查询分析示例：

```
* | select ts_predicate_arma(stamp, value, 3, 2, 4) from (select __time__ - __time__ % 60 as stamp,
avg(v) as value from log GROUP BY stamp order by stamp)
```

> **说明** 输出结果与ts_predicate_simple函数相似，具体请参见ts_predicate_simple函数的输出结果。

ts_predicate_arima

函数格式：

```
select ts_predicate_arima(x, y, p, d, q, nPred, isSmooth)
```

参数说明如下：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
p	自回归模型的阶数。	long类型，取值范围为2~8。
d	差分模型的阶数。	long类型，取值范围为1~3。
q	移动平均模型的阶数。	long类型，取值范围为2~8。
nPred	预测未来的点的数量。	long类型，取值范围为1~5*p。
isSmooth	是否需要对原始数据做滤波操作。	bool类型，默认为true表示对原始数据做滤波操作。

查询分析示例：

```
* | select ts_predicate_arima(stamp, value, 3, 1, 2, 4) from (select __time__ - __time__ % 60 as stamp,
avg(v) as value from log GROUP BY stamp order by stamp)
```

> **说明** 输出结果与ts_predicate_simple函数相似，具体请参见ts_predicate_simple函数的输出结果。

ts_regression_predict

函数格式：

```
select ts_regression_predict(x, y, nPred, algotype, processType)
```

参数说明如下：

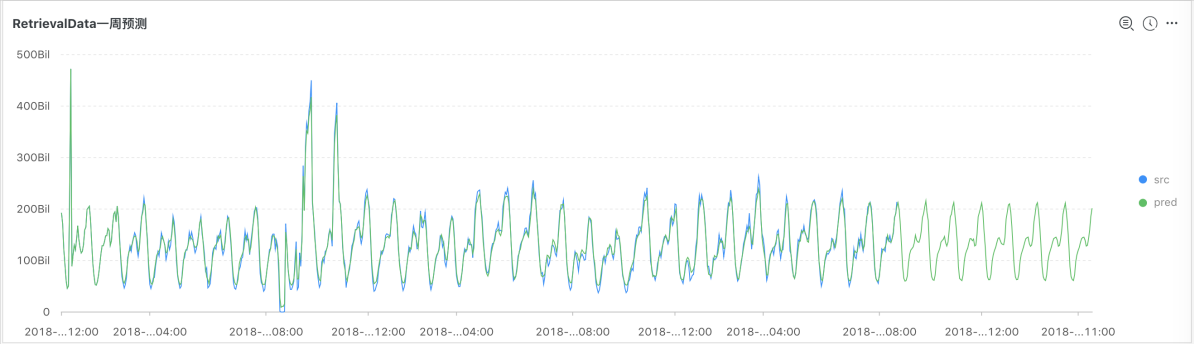
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
nPred	预测未来的点的数量。	long类型，取值范围为1~500。
algotype	针对的预测的算法类型。	取值包括： • origin：使用GBRT（Gradient Boosted Regression Tree）算法进行预测。 • forest：使用STL序列分解的结果，将分解得到的趋势序列使用GBRT算法进行预测，再将分解出来的序列按照加法模型进行求和后返回。 • linear：使用STL序列分解的结果，将分解得到趋势序列使用Linear Regression算法进行预测，再将分解出来的序列按照加法模型进行求和后返回。
processType	数据对应的预处理流程。	取值包括： • 0：不进行任何额外的数据预处理。 • 1：对数据去除异常后再进行预测处理。

示例：

- 查询分析

```
* and h : nu2h05202.nu8 and m: NET |  select ts_regression_predict(stamp, value, 200, 'origin') from
(select __time__ - __time__ % 60 as stamp, avg(v) as value from log GROUP  BY  stamp order by stamp)
```

- 输出结果



显示项如下：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
纵轴	src	原始数据。
	predict	预测数据。

ts_anomaly_filter

函数格式：

```
select ts_anomaly_filter(lineName, ts, ds, preds, probs, nWatch, anomalyType)
```

参数说明如下：

参数	说明	取值
lineName	varchar类型，表示每条曲线的名称。	-
ts	曲线的时间序列，表示当前这条曲线的时间信息。array（double）类型，由小到大排列。	-
ds	曲线的实际值序列，表示当前这条曲线的数值信息。array（double）类型，长度与ts相同。	-
preds	曲线的预测值序列，表示当前这条曲线的预测值。array（double）类型，长度与ts相同。	-
probs	曲线的异常检测序列，表示当前这条曲线的异常检测结果。array（double）类型，长度与ts相同。	-
nWatch	long类型，表示当前曲线中最近观测的实际值的数量，长度必须小于实际的曲线长度。	-
anomalyType	long类型，表示要过滤的异常类型的种类。	取值包括： • 0：表示关注全部异常。 • 1：表示关注上升沿异常。 • -1：表示下降沿异常。

示例：

- 查询分析

```
* | select res.name, res.ts, res.ds, res.preds, res.probs
     from (
         select ts_anomaly_filter(name, ts, ds, preds, probs, cast(5 as bigint), cast(1 as bigint)) as
res
       from (
         select name, res[1] as ts, res[2] as ds, res[3] as preds, res[4] as uppers, res[5] as lowers,
res[6] as probs
     from (
         select name, array_transpose(ts_predicate_ar(stamp, value, 10)) as res
         from (
           select name, stamp, value from log where name like '%asg-%') group by name)) );
```

- 输出结果

```
| name                     | ts                                                          | ds          | preds
| probs       |
| ------------------------ | ----------------------------------------------------------- | ----------- | ------
- | ----------- |
| asg-bp1hylzdi2wx7civ0ivk | [1.5513696E9, 1.5513732E9, 1.5513768E9, 1.5513804E9] | [1,2,3,NaN] | [1,2
,3,4] | [0,0,1,NaN] |
```

4.9.7. 序列分解函数

序列分解函数提供针对业务曲线的分解功能，突出曲线的趋势信息和周期信息。

ts_decompose

函数格式：

```
select ts_decompose(x, y)
```

参数说明如下：

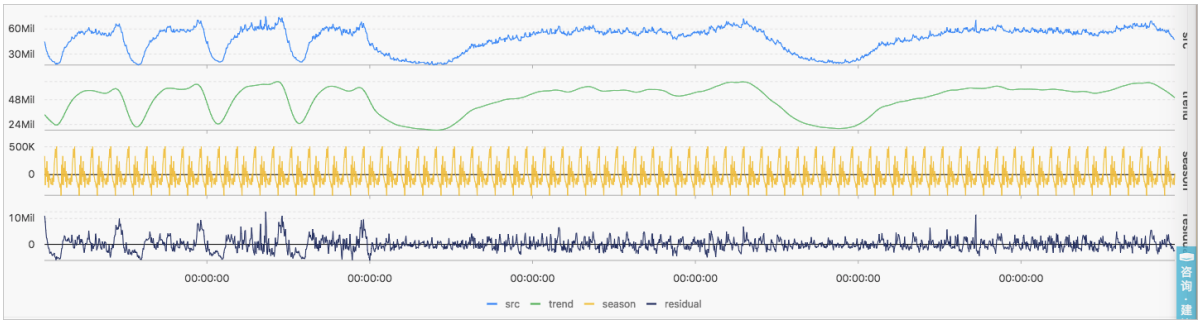
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-

示例：

- 查询分析：

```
* | select ts_decompose(stamp, value) from (select __time__ - __time__ % 60 as stamp, avg(v) as value
from log GROUP BY stamp order by stamp)
```

- 输出结果：



显示项如下：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
	src	原始数据。

纵轴	trend	分解出来的趋势数据。
	season	分解出来的周期数据。
	residual	分解出来的残差数据。

4.9.8. 时序聚类函数

时序聚类函数针对输入的多条时序数据进行聚类，自动聚类出不同的曲线形态，进而快速找到相应的聚类中心和异于聚类中的其它形态曲线。

关于函数的算法及实现原理请参见LOG机器学习介绍（02）：时序聚类建模。

函数列表

函数	说明
`ts_density_cluster`	使用密度聚类方法对多条时序数据进行聚类。
`ts_hierarchical_cluster`	使用层次聚类方法对多条时序数据进行聚类。
`ts_similar_instance`	查找到指定曲线名称的相似曲线。

ts_density_cluster

函数格式：

```
select ts_density_cluster(x, y, z)
```

参数说明如下：

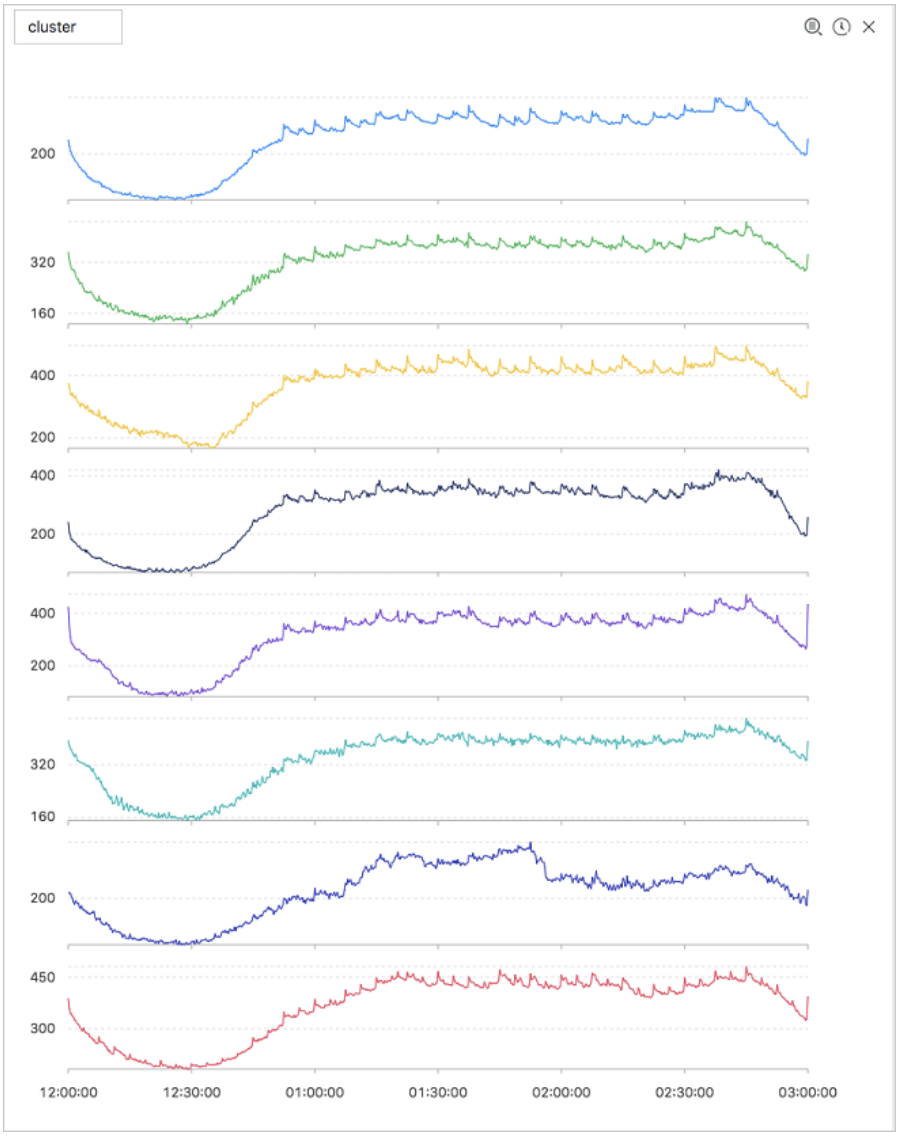
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
z	某个时刻数据对应的曲线名称。	字符串类型，例如machine01.cpu_usr。

示例：

- 查询分析：

```
* and (h: "machine_01" OR h: "machine_02" OR h : "machine_03") | select ts_density_cluster(stamp, metr
ic_value,metric_name ) from ( select __time__ - __time__ % 600 as stamp, avg(v) as metric_value, h as
metric_name from log GROUP BY stamp, metric_name order BY metric_name, stamp )
```

- 输出结果：



显示项如下：

显示项	说明
cluster_id	聚类的类别，其中-1表示未能划分到某一聚类中心。
rate	该聚类中的instance占比。
time_series	该聚类中心的时间戳序列。
data_series	该聚类中心的数据序列。
instance_names	该聚类中心包含的instance的集合。
sim_instance	该类中的某一个instance名称。

ts_hierarchical_cluster

函数格式：

```
select ts_hierarchical_cluster(x, y, z)
```

参数说明如下：

参数	说明	取值

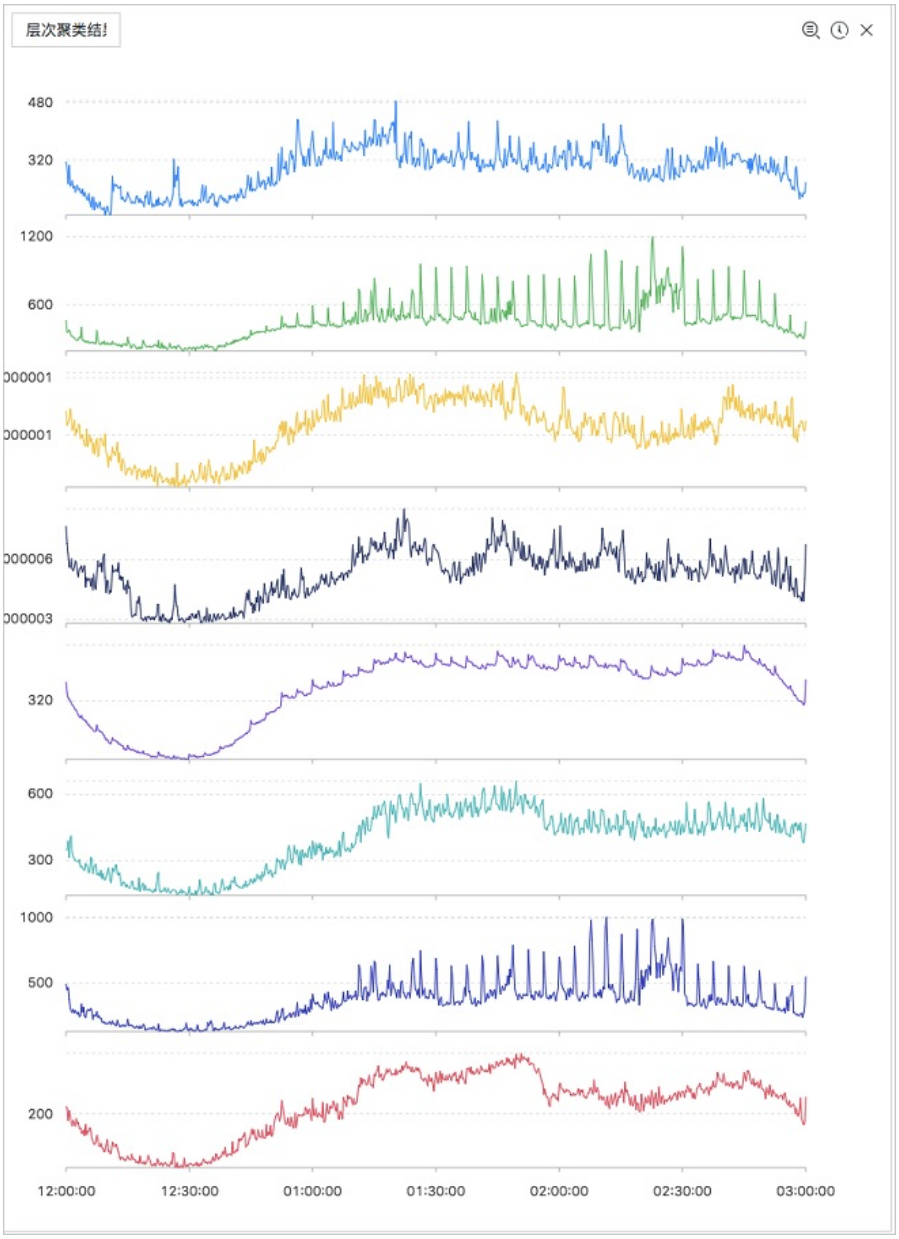
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
z	某个时刻数据对应的曲线名称。	字符串类型，例如machine01.cpu_usr。

示例：

- **查询分析：**

```
* and (h: "machine_01" OR h: "machine_02" OR h : "machine_03") | select ts_hierarchical_cluster(stamp, metric_value, metric_name) from ( select __time__ - __time__ % 600 as stamp, avg(v) as metric_value, h as metric_name from log GROUP BY stamp, metric_name order BY metric_name, stamp )
```

- **输出结果：**



显示项如下：

显示项	说明
cluster_id	聚类的类别，其中-1表示未能划分到某一聚类中心。

rate	该聚类中的instance占比。
time_series	该聚类中心的时间戳序列。
data_series	该聚类中心的数据序列。
instance_names	该聚类中心包含的instance的集合。
sim_instance	该类中的某一个instance名称。

ts_similar_instance

函数格式：

```
select ts_similar_instance(x, y, z,instance_name,topK,metricType)
```

参数说明如下：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
z	某个时刻数据对应的曲线名称。	字符串类型，例如machine01.cpu_usr。
instance_name	指定某个待查找的曲线名称。	集合中某个曲线名称，字符串类型，如：machine01.cpu_usr。 说明 必须是已创建的曲线。
topK	最多返回K个与给定曲线相似的曲线。	-
metricType	`{'shape', 'manhattan', 'euclidean'}`，衡量时序曲线之间的相似性指标。	-

查询分析示例：

```
* and m: NET and m: Tcp and (h: "nu4e01524.nu8" OR  h: "nu2i10267.nu8" OR  h : "nu4q10466.nu8") |
select ts_similar_instance(stamp, metric_value, metric_name, 'nu4e01524.nu8' ) from ( select __time__ -
__time__ % 600 as stamp, sum(v) as metric_value, h as metric_name from log GROUP BY stamp, metric_name
order BY  metric_name, stamp )
```

显示项如下：

显示项	说明
instance_name	与指定指标相近的结果列表。
time_series	该曲线的时间戳序列。
data_series	该曲线的数据序列。

4.9.9. 频繁模式统计函数

频繁模式统计函数可以在给定的多属性字段样本中，挖掘出具有一定代表性的属性组合，用来归纳当前日志。

pattern_stat

函数格式：

```
select pattern_stat(array[col1, col2, col3], array['col1_name', 'col2_name', 'col3_name'], array[col5, c
ol6], array['col5_name', 'col6_name'], support_score, sample_ratio)
```

参数说明如下：

参数	说明	取值
array[col1, col2, col3]	字符型数据的输入列。	数组形式，例如：array[clientIP, sourceIP, path, logstore]。
array['col1_name', 'col2_name', 'col3_name']	字符型数据的输入列的对应名称。	数组形式，例如：array['clientIP', 'sourceIP', 'path', 'logstore']。
array[col5, col6]	数值型数据的输入列。	数组形式，例如：array[Inflow, OutFlow]。
array['col5_name', 'col6_name']	数值型数据的输入列的对应名称。	数组形式，例如array['Inflow', 'OutFlow']。
support_score	样本在进行模式挖掘时的支持度。	double类型，取值为(0,1]。
sample_ratio	采样比率，默认为0.1，表示只拿10%全量集合。	double类型，取值为(0,1]。

示例：

- **查询分析**

```
* | select pattern_stat(array[ Category, ClientIP, ProjectName, LogStore, Method, Source, UserAgent ], array[ 'Category', 'ClientIP', 'ProjectName', 'LogStore', 'Method', 'Source', 'UserAgent' ], array[ InFlow, OutFlow ], array[ 'InFlow', 'OutFlow' ], 0.45, 0.3) limit 1000
```

- **显示项**

显示项	说明
count	当前模式所含样本的数量。
support_score	当前模式的支持度。
pattern	模式的具体内容，按照条件查询的形式组织。

4.9.10. 差异模式统计函数

差异模式统计函数基于给定的多属性字段样本，在给定的判别条件下，分析出影响该条件划分的差异化模式集合，帮助您快速诊断导致当前判别条件差异的原因。

pattern_diff

函数格式：

```
select pattern_diff(array_char_value, array_char_name, array_numeric_value, array_numeric_name, condition, supportScore,posSampleRatio,negSampleRatio )
```

参数说明如下：

参数	说明	取值
array_char_value	字符型数据的输入列。	数组形式，例如：array[clientIP, sourceIP, path, logstore]。
array_char_name	字符型数据的输入列的对应名称。	数组形式，例如：array['clientIP', 'sourceIP', 'path', 'logstore']。
array_numeric_value	数值型数据的输入列。	数组形式，例如：array[Inflow, OutFlow]。
array_numeric_name	数值型数据的输入列的对应名称。	数组形式，例如array['Inflow', 'OutFlow']。
condition	筛选数据的条件。条件为True则为正样本，条件为False则为负样本。	例如：Latency <= 300。
supportScore	正负样本在进行模式挖掘时的支持度。	double类型，取值为(0,1]。
posSampleRatio	正样本的采样率。默认为0.5，表示只取50%正样本集合。	double类型，取值为(0,1]。

negSampleRatio	负样本的采样率，默认为0.5，表示只取50%负样本集合。	double类型，取值为(0,1]。

示例：

- 查询分析

```
* | select pattern_diff(array[ Category, ClientIP, ProjectName, LogStore, Method, Source, UserAgent ], array[ 'Category', 'ClientIP', 'ProjectName', 'LogStore', 'Method', 'Source', 'UserAgent' ], array[ InFlow, OutFlow ], array[ 'InFlow', 'OutFlow' ], Latency > 300, 0.2, 0.1, 1.0) limit 1000
```

- 显示项

显示项	说明
possupport	挖掘出来的模式在正样本中的支持度。
posconfidence	挖掘出来的模式在正样本中的置信度。
negsupport	挖掘出来的模式在负样本中的支持度。
diffpattern	挖掘出来的具体模式内容。

4.9.11. 根因分析函数

日志服务提供了强大的告警和分析能力，可以帮助用户快速分析和定位到发生异常的具体的子维度。在时序指标发生异常时，根因分析函数可以快速分析出是哪些相关维度属性发生异常而导致监控指标发生异常。

LOG机器学习最佳实战：根因分析（一）

rca_kpi_search

函数格式

```
select rca_kpi_search(varchar_array, name_array, real, forecast, level)
```

参数说明如下：

参数	说明	取值
varchar_array	属性维度字段。	数组形式，例如：array[col1, col2, col3]。
name_array	属性名字字段。	数组形式，例如：array['col1', 'col2', 'col3']。
real	varchar_array对应的实际值。	double 类型，取值范围：全体实数。
forecast	varchar_array对应的预测值。	double 类型，取值范围：全体实数。
level	输出的根因集合对应的维度属性的数量，其中level=0表示输出找到的全部根因集合。	long类型，取值范围：0<=level<=分析维度数（对应varchar_array的长度）。

示例：

- 查询分析：

  先利用子查询去组织每个细粒度属性对应的实际值和预测值，然后直接调用rca_kpi_search函数去分析异常时刻的根因。

```
* not Status:200 |
select rca_kpi_search(
 array[ ProjectName, LogStore, UserAgent, Method ],
 array[ 'ProjectName', 'LogStore', 'UserAgent', 'Method' ], real, forecast, 1)
from (
select ProjectName, LogStore, UserAgent, Method,
 sum(case when time < 1552436040 then real else 0 end) * 1.0 / sum(case when time < 1552436040
then 1 else 0 end) as forecast,
 sum(case when time >=1552436040 then real else 0 end) *1.0 / sum(case when time >= 1552436040
then 1 else 0 end) as real
 from (
select __time__ - __time__ % 60 as time, ProjectName, LogStore, UserAgent, Method, COUNT(*) as real
from log GROUP by time, ProjectName, LogStore, UserAgent, Method )
GROUP BY ProjectName, LogStore, UserAgent, Method limit 100000000)
```

- **输出结果：**

返回结果结构说明：

```
{
  "rcSets": [
    {
      "rcItems": [
        {
          "kpi": [{"attr": "xxx", "val": "xxx"}],
          "nleaf": 100,
          "change": 0.524543,
          "score": 0.1454543
        }
      ]
    }
  ]
}
```

显示项如下：

显示项	说明
rcSets	根因集合，value对应一个数组。
rcItems	具体对应一个根因集合。
kpi	根因集合中的一项，数据按照数组形式存储，数组中的每一项是一个json类型的数据，attr表示维度名称，val表示当前维度下对应的属性名称。
nleaf	根因集合中某一项（KPI）在原始数据中覆盖的叶子节点数。 **说明** 叶子节点：表示最细粒度属性组合的日志。
change	根因集合中某一项（KPI）对应的叶子节点集合的异常变化量占同一时刻总体异常变化量的比例。

score	当前kpi对应的异常程度（0 <= score <= 1）。

输出结果是一个Json，具体格式如下：

```
{
  "rcSets": [
  {
    "rcItems": [
    {
      "kpi": [
      {
        "attr": "country",
        "val": "*"
      },
      {
        "attr": "province",
        "val": "*"
      },
      {
        "attr": "provider",
        "val": "*"
      },
      {
        "attr": "domain",
        "val": "download.huya.com"
      },
      {
        "attr": "method",
        "val": "*"
      }
      ],
      "nleaf": 119,
      "change": 0.3180687806279939,
      "score": 0.14436007709620113
    }
    ]
  }
  ]
}
```

4.9.12. 相关性分析函数

针对系统中的多个观测指标，可以快速找出与某个指标项相关或者时序序列相关的指标名称。

函数列表

函数	说明
`ts_association_analysis`	针对系统中的多个观测指标，快速找出和某个指标项相关的指标名称。
`ts_similar`	针对系统中的多个观测指标，快速找出和用户输入的时序序列相关的指标名称。

ts_association_analysis

函数格式：

```
select ts_association_analysis(stamp, params, names, indexName, threshold)
```

参数说明如下：

参数	说明	取值

stamp	long 类型，表示UnixTime时间戳。	-
params	array（double）类型，表示待分析的指标维度。	例如：Latency，QPS，NetFlow等。
names	array（varchar ）类型，表示待分析的指标名称。	例如：Latency，QPS，NetFlow等。
indexName	varchar 类型，表示分析目标指标的名称。	例如：Latency。
threshold	double 类型，表示其它分析指标与目标指标间的相关性阈值。	取值范围在：[0，1]。

结果输出：

- **name**：指标的名称。
- **score**：该指标与目标指标之间的相关性值，范围在[0，1]之间。

代码示例

```
* | select ts_association_analysis(
              time,
              array[inflow, outflow, latency, status],
              array['inflow', 'outflow', 'latency', 'status'],
              'latency',
              0.1) from log;
```

结果示例：

```
| results               |
| --------------------- |
| ['latency', '1.0']    |
| ['outflow', '0.6265'] |
| ['status', '0.2270']  |
```

ts_similar

函数格式一：

```
select ts_similar(stamp, value, ts, ds)
select ts_similar(stamp, value, ts, ds, metricType)
```

参数说明一：

参数	说明	取值
stamp	long 类型，表示UnixTime时间戳。	-
value	double 类型，表示某指标对应的值。	-
ts	array（double）类型，表示指定曲线的时间序列信息。	-
ds	array（double）类型，表示指定曲线的数值序列信息。	-
metricType	varchar 类型，表示度量曲线间相关性的类型。	类型如下： **SHAPE**，**RMSE**，**PEARSON**，**SPEARMAN**，**R2**，**KENDALL**

函数格式二：

```
select ts_similar(stamp, value, startStamp, endStamp, step, ds)
select ts_similar(stamp, value, startStamp, endStamp, step, ds, metricType )
```

参数说明二：

参数	说明	取值
stamp	long 类型，表示UnixTime时间戳。	-
value	double 类型，表示某指标对应的值。	-
startStamp	long 类型，表示指定曲线的开始时间戳。	-
endStamp	long 类型，表示指定曲线的结束时间戳。	-
step	long类型，表示时序中相邻两个点之间的时间间隔。	-
ds	array（double）类型，表示指定曲线的数值序列信息。	-
metricType	varchar 类型，表示度量曲线间相关性的类型。	类型如下： **SHAPE**，**RMSE**，**PEARSON**，**SPEARMAN**，**R2**，**KENDALL**

- 输出结果

  score：该指标与目标指标之间的相关性值，范围在[-1, 1]之间。

- 代码示例

```
* | select vhost, metric, ts_similar(time, value, 1560911040, 1560911065, 5,
array[5.1,4.0,3.3,5.6,4.0,7.2], 'PEARSON') from log  group by vhost, metric;
```

- 结果示例

```
| vhost  | metric          | score                |
| ------ | --------------- | -------------------- |
| vhost1 | redolog         | -0.3519082537204182  |
| vhost1 | kv_qps          | -0.15922168009772697 |
| vhost1 | file_meta_write | NaN                  |
```

4.9.13. 核密度估计函数

核密度估计（Kernel Density Estimation）是在概率论中用来估计未知的密度函数，属于非参数检验方法之一。

核密度估计函数采用平滑的峰值函数来拟合观察到的数据点，从而对真实的概率分布曲线进行模拟。

- 函数格式

```
select kernel_density_estimation(bigint stamp, double value, varchar kernelType)
```

- 参数说明

参数	说明
stamp	UnixTime 时间戳数据，单位为秒。
value	对应的观测数值。
kernelType	◦ box：矩形。 ◦ epanechnikov：Epanechnikov 曲线。 ◦ gaussian：高斯曲线。

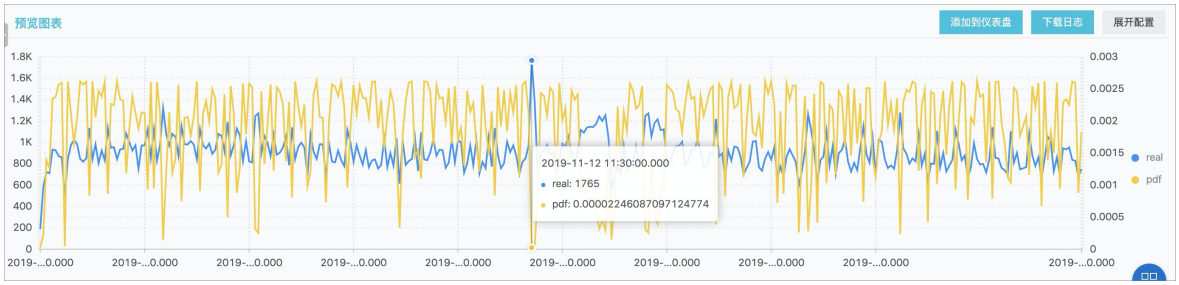
- 输出结果

显示项	说明
unixtime	对应原始数据的时间数据。
real	对应的观测数值。
pdf	对应的每个观测点的概率值。

- 示例
  - 代码示例

```
* |
select
    date_trunc('second', cast(t1[1] as bigint)) as time, t1[2] as real, t1[3] as pdf from (
        select kernel_density_estimation(time, num, 'gaussian') as res from (
            select __time__ - __time__ % 10 as time, COUNT(*) * 1.0 as num from log group by time or
der by time)
        ), unnest(res) as t(t1)  limit 1000
```

  - 结果示例



4.10. 分析进阶

4.10.1. 优化查询

本文主要介绍优化查询的方法，可以帮助用户提高查询效率。

主要包括以下方法：

- 增加更多shard。
- 缩短时间范围和数据量。
- 多次重复查询。
- 优化查询的SQL。

增加更多shard

shard代表的是计算资源，shard越多，计算越快。保证平均每个shard扫描的数据不多于5000万条日志。 增加shard可以通过分裂Shard完成。

> **说明** 分裂shard后，会产生更多费用，且只对新数据起到加速效果，旧数据仍然在旧的shard上。

缩减查询的时间范围和数据量

- 时间范围越大，查询越慢。如果您查询1年的时间，或者查询1个月的时间。 计算是按天完成的，所以，适当的缩短时间可以更快完成计算。
- 数据量越大，查询越慢。请尽量减少查询的数据量。

多次重复查询

当查询不精确时，可以尝试多次重复查询。每次查询时，底层加速机制会充分利用已有的结果进行分析。所以，多次查询可以使结果更加精确。

优化query

计算时间较长的查询语句特点：

- 对字符串列进行group by。
- 对多列（大于5列）字段进行group by。
- 在SQL中有生成字符串的操作。

为了优化query，有以下方法：

- 尽量避免生成字符串的操作

- 使用date_format函数生成格式化的时间戳，导致查询效率低下。

```
* | select date_format(from_unixtime(__time__) , '%H_%i') as t, count(1) group by t
```

- 使用substr生成字符串。 对于这类时间戳函数，建议使用date_trunc或者time_series函数进行分析。

- 尽量避免对字符串列进行GROUP BY计算
对字符串进行GROUP BY，会导致大量的hash计算，这部分计算量往往会占据整体计算的50%以上。例如：

```
* | select count(1) as pv , date_trunc('hour',__time__) as time group by time
* | select count(1) as pv , from_unixtime(__time__-__time__%3600) as time group by __time__-__time__%3
600
```

Query 1 和Query 2都是计算每小时的日志count数，但是Query 1 首先把时间转化成字符串，例如2017-12-12 00:00:00， 然后对这个字符串进行GROUP BY。 Query 2是先对时间整点值进行计算，GROUP BY计算后才会转化成字符串类型。所以在执行效率上，Query 2更佳。

- GROUP BY多列时，把字典大的字段放在前面
例如，province有13个，用户有1亿。

```
快： * | select province,uid,count(1)groupby province,uid
慢: * | select province,uid,count(1)groupby uid,province
```

- 使用估算函数
估算函数的性能要比精确计算好很多。估算会损失一些可接受的精确度，来达到快速计算的效果。

```
快： * |select approx_distinct(ip)
慢: * | select count(distinct(ip))
```

- 在SQL中获取需要的列，尽量不要读取所有列
获取所有列，请使用查询语法。在SQL计算时，尽量只读取需要参与计算的列，可以加快计算。

```
快 : * |select a,b c
慢 :* |select*
```

- 非group by的列，尽量放到聚合函数中
例如，userid，用户名，必定是一一对应的，我们只需要按照userid进行group by即可。

```
快： * | select userid, arbitrary(username), count(1)groupby userid
慢: * | select userid, username, count(1)groupby userid,username
```

- 避免使用in语法
尽量避免在SQL中使用in语法，而应该把in语法用搜索的or语法代替。

```
快：key : a or key :b or key:c | select count(1)
慢：* | select count(1) where key in ('a','b')
```

4.10.2. 优秀分析案例

本文档向您展示日志数据分析的一些案例。

5分钟错误率超过40%时触发报警

统计每分钟的500错误率，当最近5分钟错误率超过40%时触发报警。

```
status:500 | select __topic__, max_by(error_count,window_time)/1.0/sum(error_count) as error_ratio, sum(
error_count) as total_error  from (
select __topic__, count(*) as error_count , __time__ - __time__ % 300  as window_time  from log group
by __topic__, window_time
)
group by __topic__  having  max_by(error_count,window_time)/1.0/sum(error_count)   > 0.4  and sum(error_
count) > 500 order by total_error desc limit 100
```

统计流量并设置告警

统计每分钟的流量，当最近的流量出现暴跌时，触发报警。 由于在最近的一分钟内，统计的数据不是一个完整分钟的，所以需要除以 `(max(time) - min(time))` 进行归一化，统计每个分钟内的流量均值。

```
* | SELECT SUM(inflow) / (max(__time__) - min(__time__))  as inflow_per_minute,
date_trunc('minute',__time__)  as minute group by minute
```

计算不同数据区间的平均延时

按照数据区间分桶，在每个桶内计算平均延时。

```
* | select avg(latency) as latency , case when originSize < 5000 then 's1' when originSize < 20000 then 's2' when originSize < 500000 then 's3' when originSize < 100000000 then 's4' else 's5' end as os group by os
```

返回不同结果的百分比

返回不同部门的count结果，及其所占百分比。该query结合了子查询、窗口函数。其中 `sum(c) over()` 表示计算所有行的和。

```
* |  select   department, c*1.0/ sum(c) over ()  from(select  count(1) as c, department   from log group by department)
```

统计满足条件的个数

在URL路径中，我们需要根据URL不同的特征来计数，这种情况可以使用CASE WHEN语法，但还有个更简单的语法是count_if。

```
* | select  count_if(uri like '%login') as login_num, count_if(uri like '%register') as register_num, date_format(date_trunc('minute', __time__), '%m-%d %H:%i') as time  group by time order by time limit 100
```

4.10.3. 时间字段转换示例

在查询分析中，往往需要对日志中的时间字段进行处理，例如将时间戳转换成指定格式等，本文档介绍时间字段的常用转换示例。

日志中可能有多个记录时间的字段，例如：

- `__time__` ：用API/SDK写入日志数据时指定的日志时间，该字段可用于日志投递、查询、分析。
- 日志中原有的时间字段：日志在生成时，用于记录日志事件发生时间的字段，是原始日志的字段。

时间字段的格式可能不统一、或不便于查看和阅读，可以在查询分析中将其转换为指定格式。例如：

1. 把__time__转化成时间戳
2. 把__time__以固定格式打印
3. 把timestamp转化成指定格式

把__time__转化成时间戳

把字段 `__time__` 转化成时间戳格式，建议使用from_unixtime函数。

```
* | select from_unixtime(__time__)
```

把__time__以固定格式打印

把字段 `__time__` 以 `年-月-日 时:分:秒` 的形式打印下来。建议使用date_format函数。

```
* | select date_format(__time__, '%Y-%m-%d %H:%i:%S')
```

把日志中的时间转换成指定格式

把日志中的时间字段转化成指定格式（ `年-月-日 时:分:秒` ），只取 `年-月-日` 部分，并做Group by处理。建议使用date_format函数。

- 日志样例

```
__topic__:
body_byte_sent:  307
hostname:  www.host1.com
http_user_agent:  Mozilla/5.0 (iPhone; CPU iPhone OS 10_3_3 like Mac OS X) AppleWebKit/603.3.8
(KHTML, like Gecko) Mobile/14G60 QQ/7.1.8.452 V1_IPH_SQ_7.1.8_1_APP_A Pixel/750 Core/UIWebView NetType
/WIFI QBWebViewType/1
method:  GET
referer:  www.host0.com
remote_addr:  36.63.1.23
request_length:  111
request_time:  2.705
status:  200
upstream_response_time:  0.225582883754
url:  /?k0=v9&
time:2017-05-17 09:45:00
```

- SQL语句样例

```
* | select date_format (date_parse(time,'%Y-%m-%d %H:%i:%S'), '%Y-%m-%d') as day, count(1) as uv
group by day order by day asc
```

4.11. 可视化分析

4.11.1. 分析图表

4.11.1.1. 图表简介

日志服务提供类似于SQL的聚合计算功能，一切通过SQL聚合计算的结果都可以通过日志服务提供的可视化图表进行渲染。

前提条件

- 已开启并配置索引，并开启分析功能。
- 只有在查询中使用分析语句，才能根据统计结果为您展示图表。

注意事项

当依次执行多个查询分析语句时，系统无法自动判断您的**数值列**或**X轴**、**Y轴**等信息，可能会默认保留您上次查询时的属性配置，导致当前查询语句无法自动生成分析图表。如果出现以下报错时，请按照当前查询语句重新选择**属性配置信息**。

- 当前选择的维度不在统计的数据维度中，请检查并调整属性配置。
- 当前无X轴信息或Y轴信息，请检查并调整属性配置。

图表设置

统计图表页签中展示查询分析语句的图形化分析结果，支持在图表栏中设置图表类型。

- **统计图表**页签左侧为您展示当前查询分析语句的**预览图表**和**预览数据**。其中，**预览图表**是指定的分析图表类型，**预览数据**以表格形式清晰直观地展示对应的图表数据。
- **统计图表**页签右侧可以进行多种图表属性设置，包括：
  - **数据源**页签：用于设置占位符变量，如果有图表的下钻行为是跳转到这个图表所在的仪表盘，那么当变量名一致的情况下，会将单击触发下钻的数据替换为此处设置的占位符变量，重新执行分析。详细说明请查看下钻分析。

    适用于下钻场景中的目的仪表盘。
  - **属性配置**页签：用于配置图表的显示属性，包括X轴、Y轴数据源、边距、字号等，不同的图表属性不同。详细说明请查看各个图表的文档。

    适用于所有的查询分析场景。
  - **交互行为**页签：用于设置该图表的下钻动作，设置后，在仪表盘中单击该图表中的值，即可执行指定的下钻动作。详细说明请查看下钻分析。

    适用于下钻场景中的触发下钻图表。

4.11.1.2. 表格

表格作为最常见的数据展示类型，是组织整理数据最基本的手段，通过的对数据的整理，达到快速引用和分析的目的。日志服务提供类似于SQL的聚合计算功能，通过查询分析语法得到的数据结果默认以表格方式进行展示。

基本构成

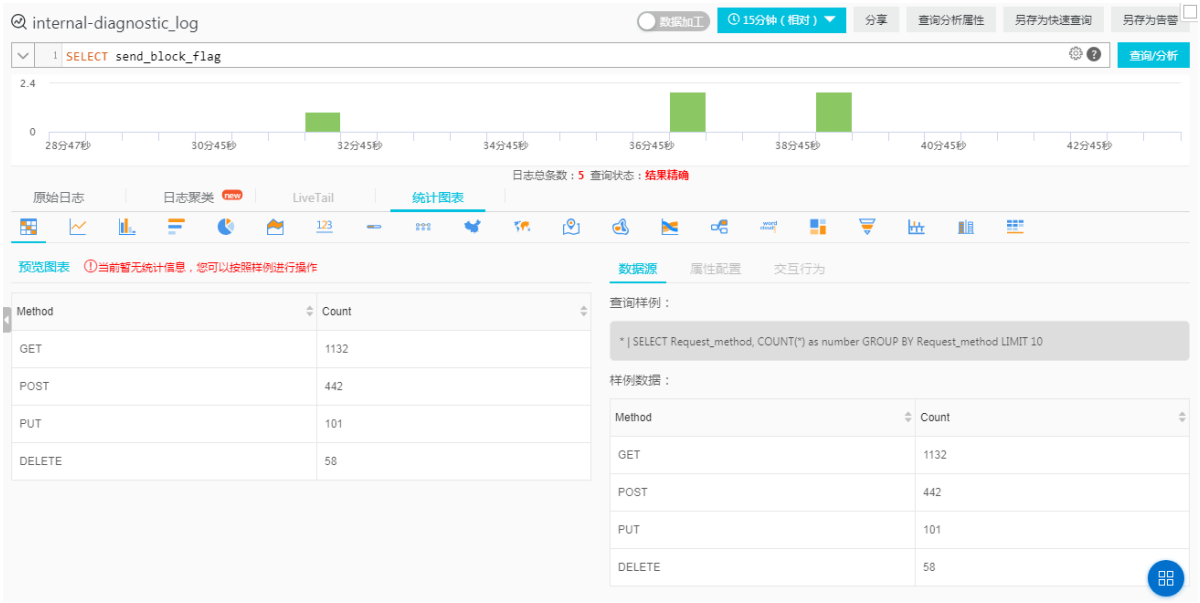
- 表头
- 行
- 列

其中：

- `SELECT` 项的个数为列数。
- 行数由当前时间区间日志条数经过计算后的个数决定，默认为 `LIMIT 100` 。

操作步骤

1. 在查询分析页面的查询框中输入查询分析语句，选择时间区间后单击**查询/分析**。



2. 页面默认显示**统计图表**页签，以  形式展示结果。

3. 在右侧**属性配置**页签中配置图表属性。

属性配置

配置项	说明
每页条数	每页显示的数据条数。
显示斑马线	开启后表格以斑马线样式显示。
行列变换	单击可对行列进行变换。
隐藏保留字段	开启后保留字段隐藏不显示。
关闭排序功能	开启后即可关闭排序功能。
关闭搜索功能	开启后即可关闭搜索功能。
高亮设置	通过设置高亮规则，可以使符合规则的行或列高亮显示。

4.11.1.3. 折线图

线图属于趋势类分析图表，一般用于表示一组数据在一个有序数据类别（多为连续时间间隔）上的变化情况，用于直观分析数据变化趋势。

在线图中，我们可以清晰的观测到数据在某一个周期内的变化，主要反映在：

- 递增性或递减性

- 增减的速率情况
- 增减的规律（如周期变化）
- 峰值和谷值

所以，线图是用于分析数据随时间变化趋势的最佳选择。同时，也可以绘制多条线用于分析多组数据在同一时间周期的变化趋势，进而分析诸如数据之间的相互作用和影响（如同增同减，成反比等）。

基本构成

- X轴
- 左Y轴
- 右Y轴（可选）
- 数据点
- 变化趋势线
- 图例

使用步骤

1.

2. 在**统计图表**页签中，选择折线图 。

3. 在右侧**属性配置**页签中配置图表属性。

> **说明** 线图单条线的数据记录数要大于2，以免无法分析数据趋势，同时，建议同一个图上不要超过5条线。

属性配置

配置项	说明
X轴	一般为有序数据类别（时间序列）。
左Y轴	可以配置一列或多列数据对应到左轴数值区间。
右Y轴	可以配置一列或多列数据对应到右轴数值区间（右轴图层高于左轴）。
为柱列	将已选择的左Y轴或者右Y轴中的一列以柱状形式表示。
图例位置	图例在图表中的位置，可以配置为上、下、左和右。
左Y轴格式化	将Y轴数据按照指定格式进行显示。
右Y轴格式化	
边距	坐标轴距离图表边界距离，包括**上边距**、**下边距**、**右边距**和**左边距**。

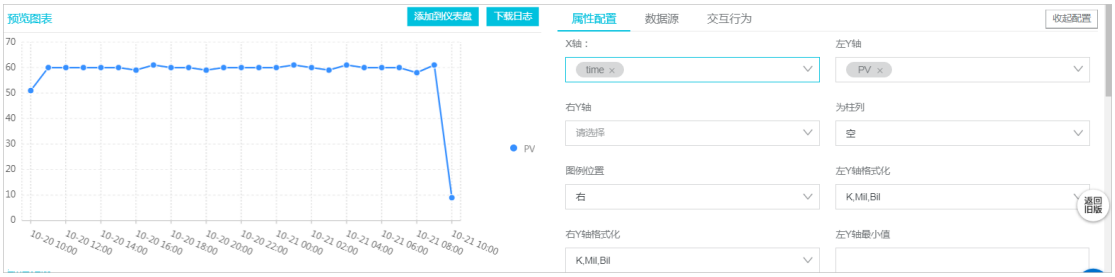
简单折线图

查询 `10.0.192.0` 这个IP在最近1天内的访问情况：

```
remote_addr: 10.0.192.0 | select date_format(date_trunc('hour', __time__), '%m-%d %H:%i')
as time, count(1) as PV group by time order by time limit 1000
```

X轴选择 `time` ，左Y轴选择 `PV` 并调整图例位置为下方显示，合理改变间距。

图 1. 简单折线图



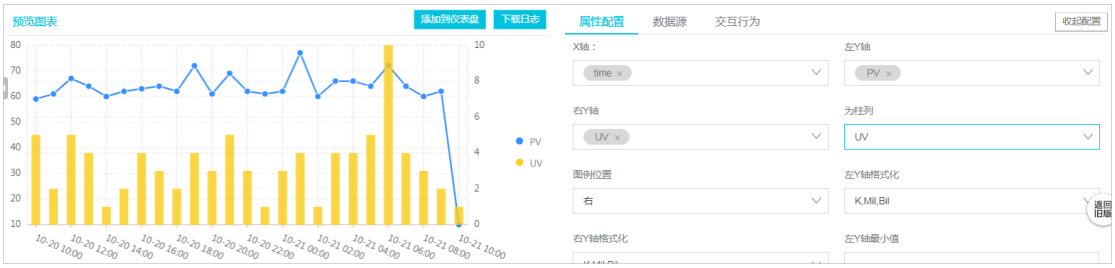
双轴折线图

查询最近1天内的访问PV、UV：

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

X轴选择 `time` ，左Y轴选择 `PV` ，右Y轴选择 `UV` 并指定 `PV` 为柱状显示。

图 2. 双轴折线图



4.11.1.4. 柱状图

柱状图使用垂直或水平的柱子显示类别之间的数值比较，和折线图的不同之处在于，柱状图描述分类数据，并统计每一个分类中的数量，而折线图描述有序数据。

基本构成

- X轴（横轴）
- Y轴（纵轴）
- 矩形块
- 图例

日志服务提供的柱状图，默认采用垂直柱子，即矩形块宽度一定，高度代表数值大小。有多列数据映射到Y轴时，采用分组柱状形式显示。

使用步骤

1.

2. 在**统计图表**页签中，选择柱状图  。

3. 在右侧**属性配置**页签中配置图表属性。

> **说明** 柱状图适用于不超过20条的数据，建议使用 `LIMIT` 进行控制，以免横向宽度过宽导致分析对比情况不直观。同时，当有多列数据映射到Y轴时，建议不要超过5个。

属性配置

配置项	说明
X轴	一般为分类数据。
Y轴	可以配置一列或多列数据对应到左轴数值区间。

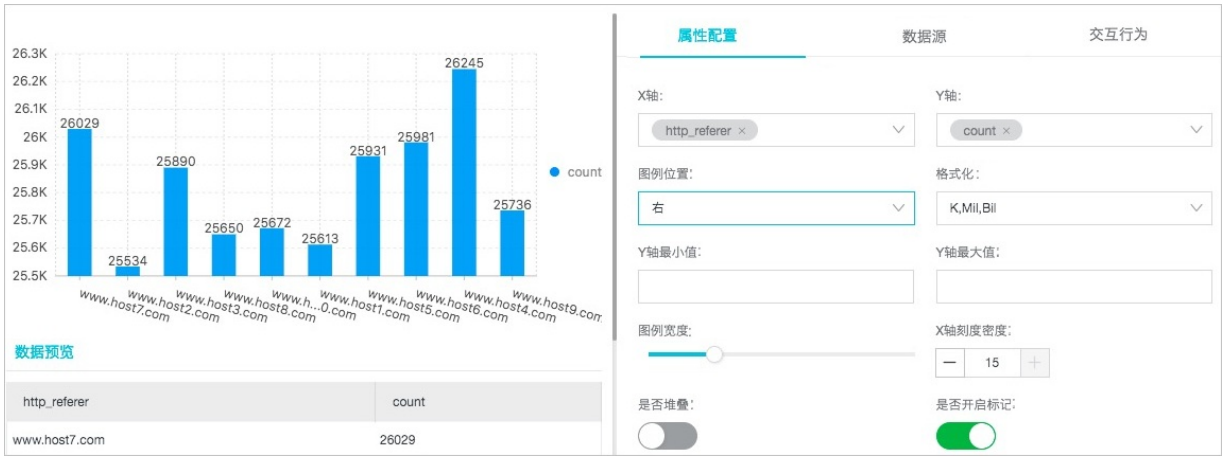
图例位置	图例在图表中的位置，可以配置为上、下、左和右。
Y轴格式化	将Y轴数据按照指定格式进行显示。
间距	坐标轴距离图表边界距离，包括**上边距**、**下边距**、**右边距**和**左边距**。

简单柱状图

查看当前时间区间每种 `http_referer` 的访问次数。

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

X轴选择 `http_referer` ，Y轴选择 `count` 。

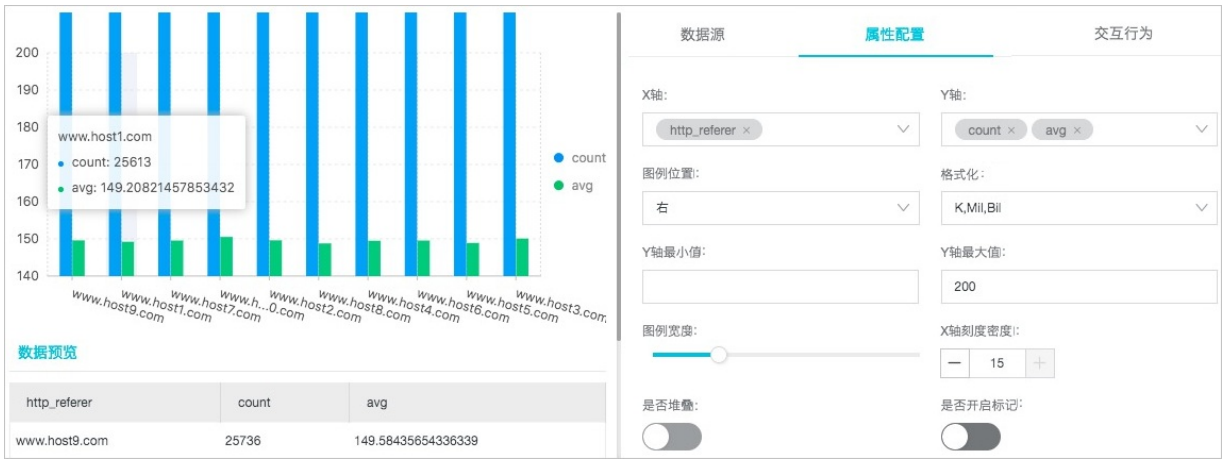


分组柱状图

查看当前时间区间每种 `http_referer` 的访问次数和平均字节数。

```
* | select  http_referer, count(1) as count, avg(body_bytes_sent) as avg group by http_referer
```

X轴选择 `http_referer` ，Y轴选择 `count` 和 `avg` 。



4.11.1.5. 条形图

条形图是柱状图另一种形式，即横向柱状图。条形图通常用于分析Top场景，配置方式也和柱状图类似。

基本构成

- X轴（纵轴）
- Y轴（横轴）
- 矩形块

- 图例

条形图矩形块高度一定，宽度代表数值大小。有多列数据映射到Y轴时，采用分组柱状形式显示。

使用步骤

1.

2. 在**统计图表**页签中，选择条形图

3. 在右侧**属性配置**页签中配置图表属性。

> **说明**
> - 条形图适用于不超过20条的数据，建议使用 `LIMIT` 进行控制，以免纵向高度过高导致分析对比情况不直观，分析Top场景时候使用 `ORDER BY` 配合。同时，当有多列数据映射到Y轴时，建议不要超过5个。
> - 支持使用分组条形图，但是条形图仅适用于同增同减的分类。

属性配置

表 1. 配置项说明

配置项	说明
X轴	一般为分类数据。
Y轴	可以配置一列或多列数据对应到左轴数值区间。
图例位置	图例在图表中的位置，可以配置为上、下、左和右。
X轴格式化	将X轴数据按照指定格式进行显示。
间距	坐标轴距离图表边界距离，包括**上边距**、**下边距**、**右边距**和**左边距**。

简单条形图示例

分析前十访问的 `request_uri` ：

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

4.11.1.6. 饼图

饼图用于表示不同分类的占比情况，通过弧度大小来对比各种分类。

构成

- 扇形
- 文本百分比
- 图例

类型

日志服务提供默认饼图、环图及南丁格尔玫瑰图三种类型的饼图。

- 饼图

  饼图通过将一个圆饼按照分类的占比划分成多个区块，整个圆饼代表数据的总量，每个区块（圆弧）表示该分类占总体的比例大小，所有区块（圆弧）的加和等于100%。
- 环图
  环图本质上是将饼图中心挖空，相比于饼图来说有如下优点：
  - 在原有构成的基础上增加了总数显示，展示了更多的信息。
  - 两个饼图直接进行比较是非常不直观的，两个环图间可以通过环状条长度进行简单的对比。
- 南丁格尔玫瑰图
  南丁格尔玫瑰图本质上并不是环图，而是在极坐标系下画出来的柱状图，每一个分类数据被圆弧平分，使用圆弧的半径长短表示数据的大小，相比于饼图来说有如下优点：
  - 饼图适用于不超过10条的分类数据，南丁格尔玫瑰图则适用于分类较多的场景（10-30条数据）。
  - 由于半径和面积是成平方的关系，南丁格尔玫瑰图放大了各个分类数据之间值的差异，尤其适合对比大小相近的数值。
  - 由于圆形有周期的特性，南丁格尔玫瑰图也适用于表示一个周期的时间概念，例如星期、月份。

使用步骤

1.

2. 在**统计图表**页签中，选择饼图  。

3. 在右侧**属性配置**页签中配置图表属性。

> **说明**
> - 饼图和环图适用于10条以内的数据，建议使用 `LIMIT` 进行控制，以免不同色的分面太多导致分析不直观。
> - 分析超过10条数据建议采用南丁格尔玫瑰图或者柱状图。

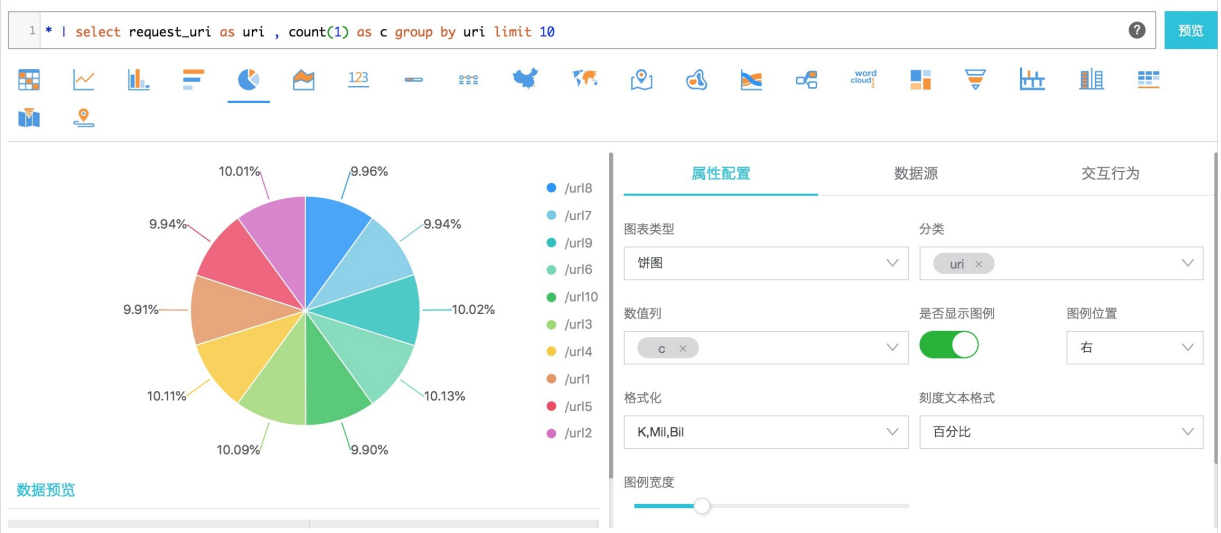
属性配置

配置项	说明
饼图表型	提供饼图（默认）、环图以及南丁格尔玫瑰图。
分类	分类数据。
数值列	分类数据对应的数值。
是否显示图例	开启后显示图例。
图例位置	图例在图表中的位置，可以配置为上、下、左和右。
格式化	将数据按照指定格式进行显示。
刻度文本格式	设置刻度的文本格式。
间距	坐标轴距离图表边界距离，包括**上边距**、**下边距**、**右边距**和**左边距**。

饼图

分析访问 `requestURI` 的占比情况：

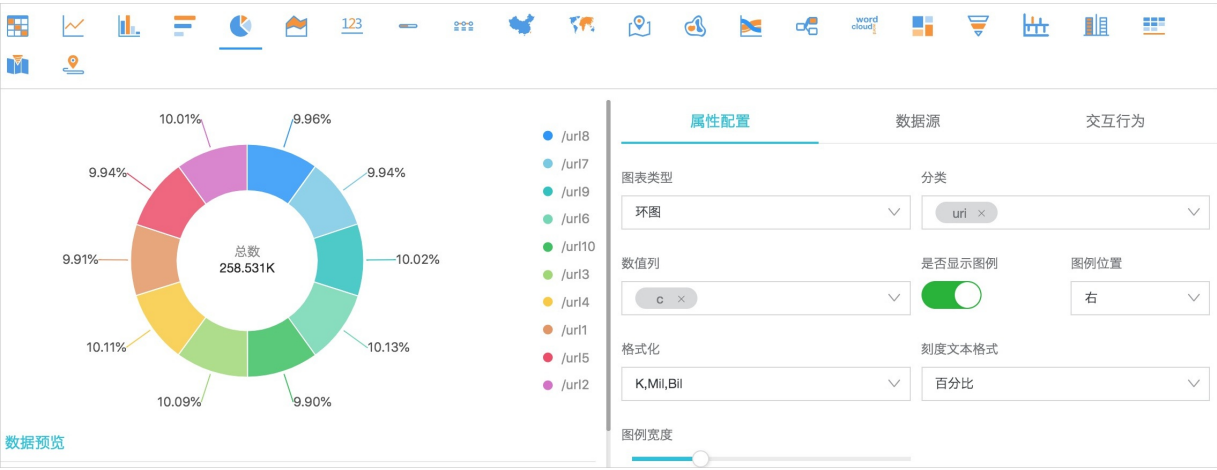
```
* | select requestURI as uri , count(1) as c group by uri limit 10
```



环图

分析访问 `requestURI` 的占比情况：

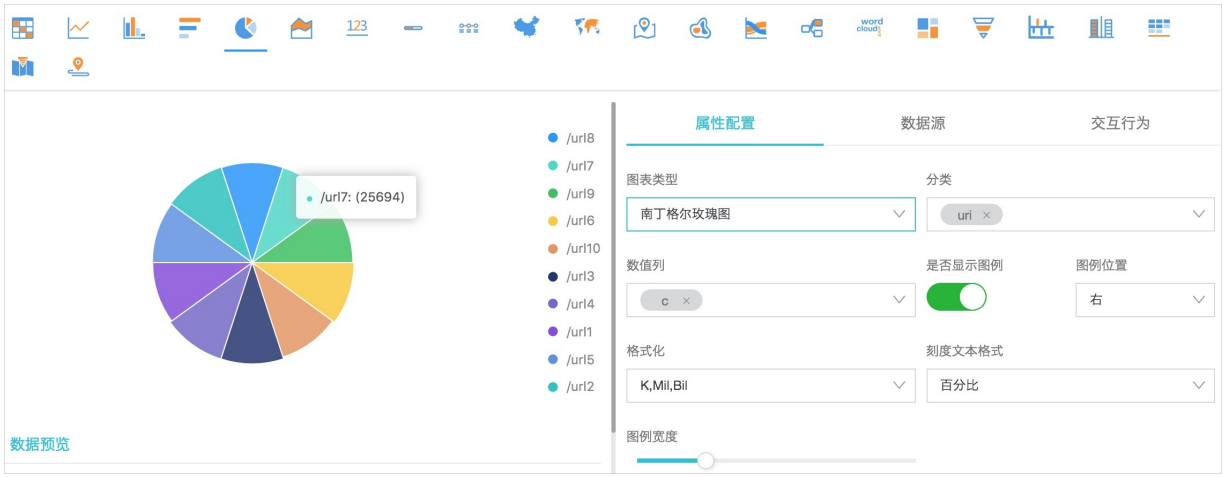
```
* | select requestURI as uri , count(1) as c group by uri limit 10
```



南丁格尔玫瑰图

分析访问 `requestURI` 的占比情况：

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



4.11.1.7. 面积图

面积图是在折线图的基础之上形成的，它将折线图中折线与坐标轴之间的区域使用颜色进行填充，这个填充即为我们所说的面积，颜色的填充可以更好的突出趋势信息。和折线图一样，面积图强调数量随时间而变化的程度，用于突出总值趋势。它们最常用于表现趋势和关系，而不是传达特定的值。

基本构成

- X轴（横轴）
- Y轴（纵轴）
- 面积块

使用步骤

1.

2. 在**统计图表**页签中，选择面积图。

3. 在右侧**属性配置**中配置图表属性。

> **说明** 面积图单个面积块数据记录数要大于2，以免无法分析数据趋势，同时，建议同一个图上不要超过5组面积块。

属性配置

配置项	说明
X轴	一般为有序数据类别（时间序列）。
Y轴	可以配置一列或多列数据对应到左轴数值区间。
图例位置	图例在图表中的位置，可以配置为上、下、左和右。
格式化	将数据按照指定格式进行显示。
间距	坐标轴距离图表边界距离，包括**上边距**、**下边距**、**右边距**和**左边距**。

简单面图

`10.0.192.0` 这个IP在最近1天内的访问情况：

```
remote_addr: 10.0.192.0 | select date_format(date_trunc('hour', __time__), '%m-%d %H:%i') as time, count(1) as PV group by time order by time limit 1000
```

X轴选择 `time` ，Y轴选择 `PV` 。

层叠面图

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

X轴选择 `time` ，Y轴选择 `PV` 和 `UV` 。

4.11.1.8. 单值图

单值图可以用于突出显示单个数值。

单值图的类型包括：

- 矩形框：用于展示一般数值。
- 刻度盘：用于查看数值与设定阈值的接近程度。
- 同比环比图：用于查看同比和环比函数的SQL查询结果，分析语法请参见同比和环比函数。

默认选择矩形框图显示。矩形框图作为最简单直接的数据表现形式，直观清晰地将某一个点上的数据展示出来，一般用于表示某一个时间点上的关键信息。针对比例类指标的显示，可选用刻度盘类型。

构成

- 主文案：数列值
- 分类：图表类型

使用步骤

1.

2. 在**统计图表**页签中，选择单值图 。

3. 在右侧**属性配置**页签中配置图表属性。

> **说明** 日志服务数字图会自动根据数值大小进行归一化操作，如 `230000` 会被处理为 `230K` ，如果需要自己定义数值格式，请通过数学计算函数在实时分析阶段进行处理。

属性配置

- 矩形框配置说明：

矩形框配置	说明
图表类型	矩形框。
数值列	默认选择该列的第一行数据进行展示。
单位	数据的单位。
单位字号	单位的字号，可以拖动调整。取值范围为10px~100px。
数值描述	数值的描述。
数值描述字号	数值描述的字号，可以拖动调整。取值范围为10px~100px。
格式化	将数据按照指定格式进行显示。
字号	数值的字号，可以拖动调整。取值范围为10px-100px。
字体颜色	数字和文字的颜色，可以选择推荐颜色或自定义设置。
背景颜色	背景的颜色，可以选择推荐颜色或自定义设置。

- 刻度盘配置说明：

配置项	说明
图标类型	将查询结果以**刻度盘**形式展示。
实际值	默认选择该列的第一行数据进行展示。
单位	刻度盘数值的单位。
字号	数值和单位的字号，取值范围为10px~100px。
数值描述	数值的描述。
数值描述字号	数值描述的字号，可以拖动调整。取值范围为10px~100px。
刻度盘最大值	刻度盘显示刻度的最大值，默认为100。
最大值所在列	**使用查询结果**打开时，**刻度盘最大值**变成**最大值所在列**，取值可以从查询结果中选取。
使用查询结果	使用查询结果的情况下，总值可以从查询结果中选取。
格式化	将数据按照指定格式进行显示。
颜色区域个数	即将刻度盘分为几个数值区域，每个区域以不同颜色表示。 颜色区域个数取值范围为2、3、4、5，默认为3个颜色区域。

区域最大值	刻度盘数值区域的最大值，最后一个区域最大值默认为刻度盘最大值，不需要指定。 **说明** 刻度盘默认3个颜色区域，且区域范围默认均分，如果您调整了**颜色区域个数**，不会改变默认区域的范围，您需要根据需求重新指定每个区域的最大值。
字体颜色	数值在仪表盘中显示的颜色。
区域	默认3个区域，对应的颜色分别为蓝、黄和红。 如果您将**颜色区域个数**更改为3个以上，新增的区域默认为蓝色，您可以重新调整各个区域的颜色。
显示标题	您可以在仪表盘中添加刻度盘类型的单值图，**显示标题**用来控制刻度盘形式的单值图标题在仪表盘页面的显示或隐藏。默认为关闭状态，即不显示刻度盘标题。 单击开启后不会在当前页面中显示效果，需要创建或修改报表后在仪表盘页面中查看。

- 同比环比图配置说明：

配置项	说明
图表类型	将查询结果以同比环比图形式展示。
显示值	显示在同比环比图中心的数值，一般设置为同比环比函数中当前时段的统计结果。
对比值	用于和阈值比较的数值，一般设置为同比环比函数中当前时段和之前时段的对比结果。
字号	显示值的字号，取值范围为10px~100px。
单位	显示值的单位。
单位字号	显示值单位的字号，取值范围为10px~100px。
比较值单位	比较值的单位。
比较值字号	比较值及其单位的字号，取值范围为10px~100px。
数值描述	对显示的数值及增长趋势的描述。
数值描述字号	数值描述的字号，取值范围为10px~100px。
趋势比较阈值	用于衡量对比值变化趋势的数值。 例如对比值是-1： ◦ 设置**趋势比较阈值**为0，页面会显示下降箭头，表示数值变化呈下降趋势。 ◦ 设置**趋势比较阈值**为-1，系统认为数据无变化，页面不显示变化趋势。 ◦ 设置**趋势比较阈值**为-2，页面显示上升箭头，表示数值变化呈上升趋势。
格式化	将数据按照指定格式进行显示。
字体颜色	显示值和数值描述的字体颜色。
增长字体颜色	对比值大于阈值时，对比值显示的字体颜色。
增长背景颜色	对比值大于阈值时，显示的背景颜色。
下降字体颜色	对比值小于阈值时，对比值显示的字体颜色
下降背景颜色	对比值小于阈值时，显示的背景颜色。
相等背景颜色	对比值等于阈值时，显示的背景颜色。

示例

执行以下查询分析语句查看访问量，并以图表方式展示分析结果：

- 矩形框

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

- 刻度盘

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

- 同比环比图
  查看今天与昨天访问量的对比：

```
* | select diff[1],diff[2], diff[1]-diff[2] from (select compare( pv , 86400) as diff from (select
count(1) as pv from log))
```

4.11.1.9. 进度条

进度条图表用于显示百分比内容，您可以通过进度条的属性配置调整进度条的样式，并设置进度条的显示规则。

构成

- 实际值
- 单位（可选）
- 总值

使用步骤

1.

2. 在**统计图表**页签中，选择进度条。

3. 进行图表属性配置。

属性配置

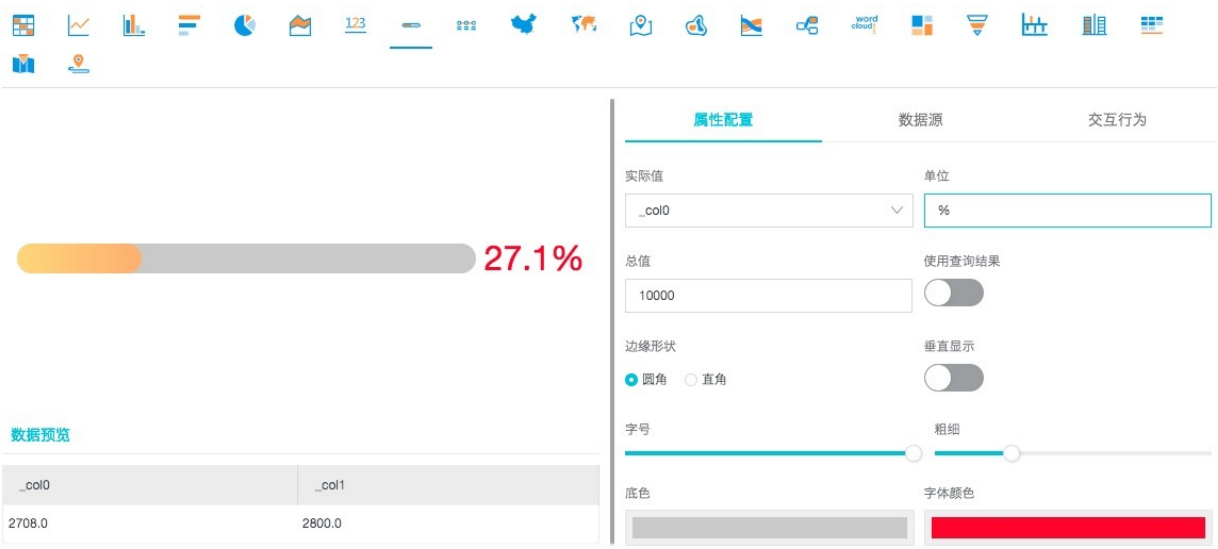
配置项	说明
实际值	默认选择该列的第一行数据进行展示。
单位	进度条数值的单位。

总值	进度条的总数值，默认100。
最大值所在列	**使用查询结果**打开时，**总值**变成**最大值所在列**，取值可以从查询结果中选取。
使用查询结果	使用查询结果的情况下，总值可以从查询结果中选取。
边缘形状	进度条的边缘形状。
垂直显示	使进度条垂直显示。
字号	可以调整进度条字体的大小。
粗细	可以调整进度条的粗细。
底色	进度条的底色。
字体颜色	进度条的字体颜色。
进度条默认颜色	默认的进度条颜色。
颜色显示方式	进度条的颜色显示方式。
开始颜色	当**颜色显示方式**选择**渐变**时，可以设置进度条的开始颜色。
结束颜色	当**颜色显示方式**选择**渐变**时，可以设置进度条的结束颜色。
显示颜色	当**颜色显示方式**选择**按照规则显示**时，可以设置进度条的显示颜色。 **说明** 将**实际值**与设置的**临界值**根据**判断条件**进行比较，符合**判断条件**则按照此处设置的颜色进行显示，否则显示默认颜色。
判断条件	当**颜色显示方式**选择**按照规则显示**时，可以设置进度条颜色显示的判断条件。
临界值	当**颜色显示方式**选择**按照规则显示**时，可以设置进度条颜色显示的判断条件的临界值。

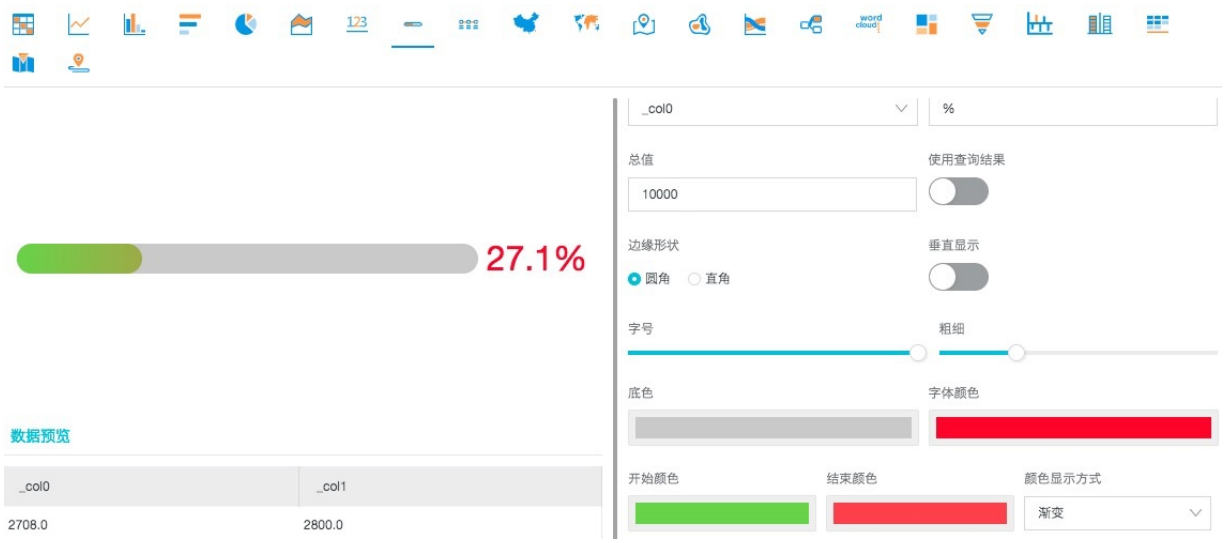
应用场景

进度条主要应用于百分比指标或数据占比的显示。

```
* | select diff[1],diff[2] from (select compare( pv , 86400) as diff from (select count(1) as pv from log))
```



如果颜色显示方式选择**按照规则显示**，可以根据设定的规则动态显示颜色，如果不满足设置的规则，则显示默认颜色。



4.11.1.10. 地图

以地图作为背景，通过图形颜色、图像标记的方式展示地理数据信息。日志服务提供了三种地图方式，分别为：中国地图、世界地图以及高德地图（高德地图分为点图和热力图）。您可以在查询分析语句中使用特定的函数，日志服务会将您的分析结果以地图方式展示出来。

基本构成

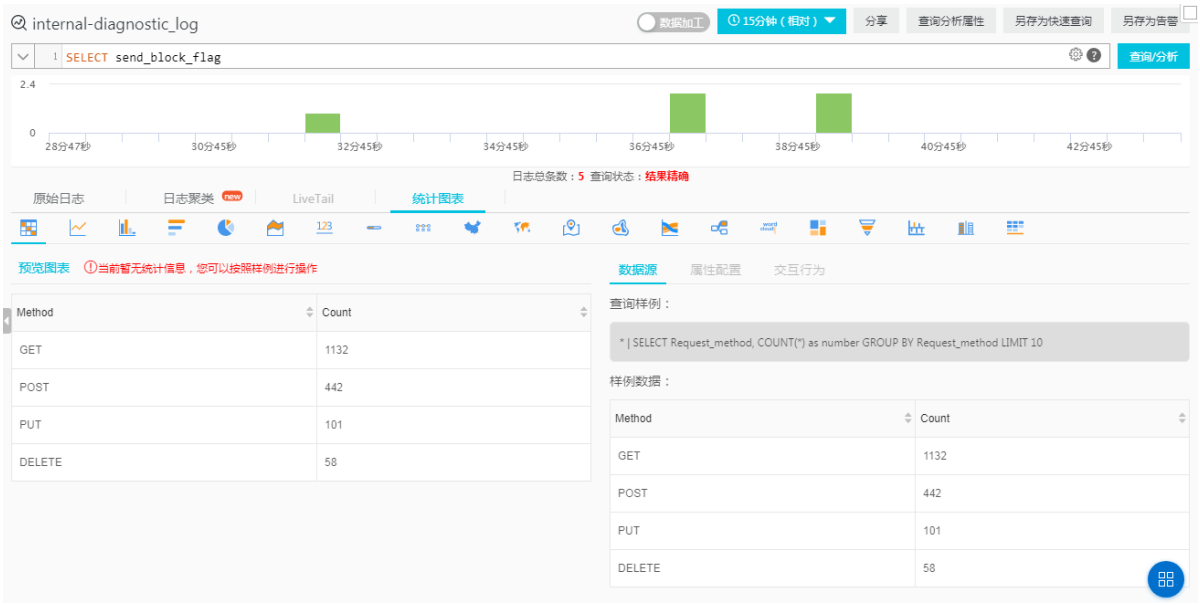
- 地图画布
- 色块

配置项

配置项	说明
位置信息	日志数据中记录的位置信息，在不同的地图类型中以不同的尺度表示。 • 省份（中国地图） • 国家（世界地图） • 经纬度（高德地图）
数值列	位置信息对应的数据量。

使用步骤

1. 在查询分析页面的查询框中输入查询分析语句，选择时间区间后单击**查询/分析**。
   - 中国地图：使用 `ip_to_province` 函数。
   - 世界地图：使用 `ip_to_country` 函数。
   - 高德地图：使用 `ip_to_geo` 函数。



2. 在**统计图表**页签中，选择地图  。

3. 进行图表属性配置。

中国地图

支持使用`ip_to_province`函数生成中国地图。

- SQL语句

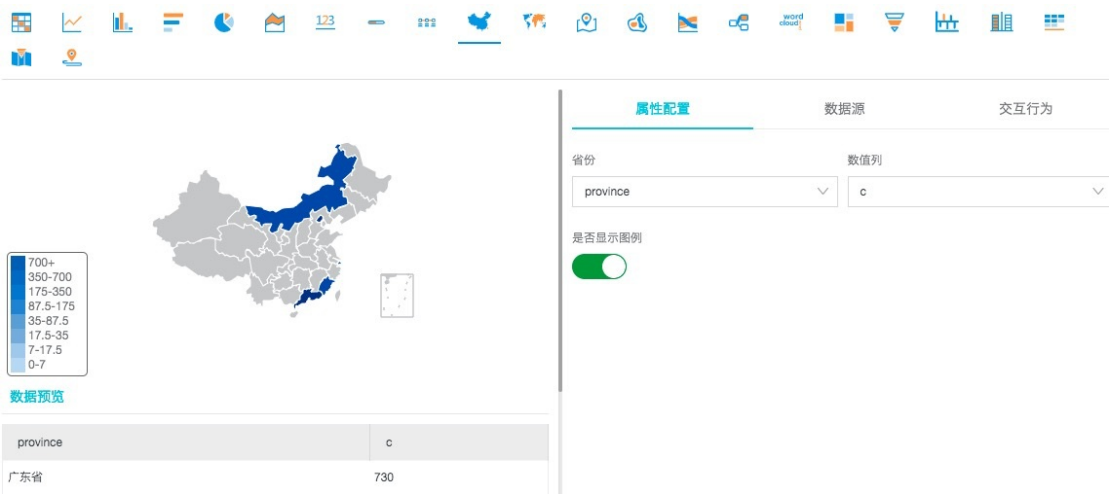
```
* | select  ip_to_province(remote_addr) as address, count(1) as count group by address order by count desc limit 10
```

- 数据集

address	count
广东省	163
浙江省	110
福建省	107
北京市	89
重庆市	28
黑龙江省	19

省份信息选择address，**数值列**选择count。

图 1. 中国地图



世界地图

支持使用 `ip_to_country` 函数生成世界地图。

- SQL语句

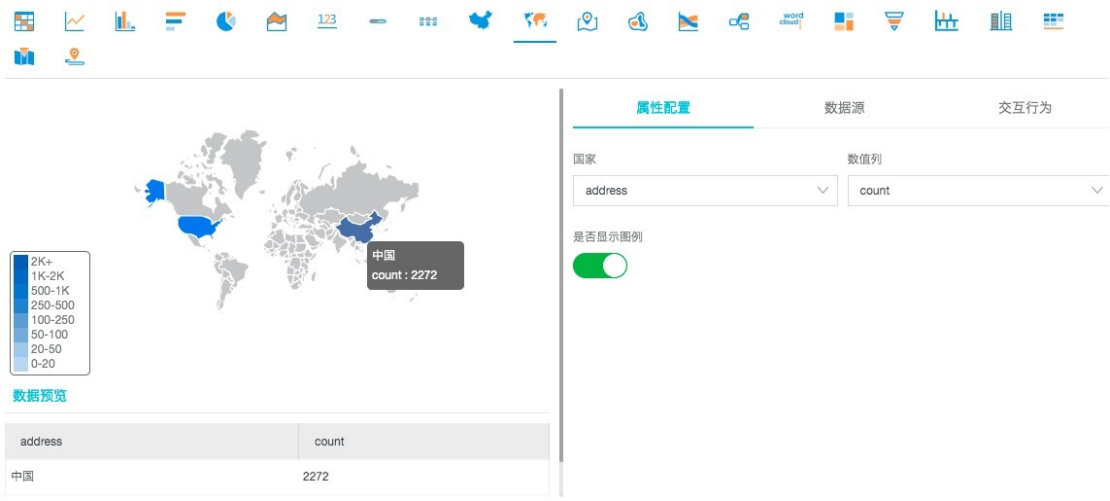
```
* | select  ip_to_country(remote_addr) as address, count(1) as count group by address order by count desc limit 10
```

- 数据集

address	count
中国	8354
美国	142

国家信息选择address，**数值列**选择count。

图 2. 世界地图



高德地图

支持使用 `ip_to_geo` 函数生成高德地图。数据集先纬后经，以“，”为分隔符，如果数据为两列lng（经度）和lat（纬度），可以使用 `concat('lat', ',', lng')` 合并为一列。

- SQL语句

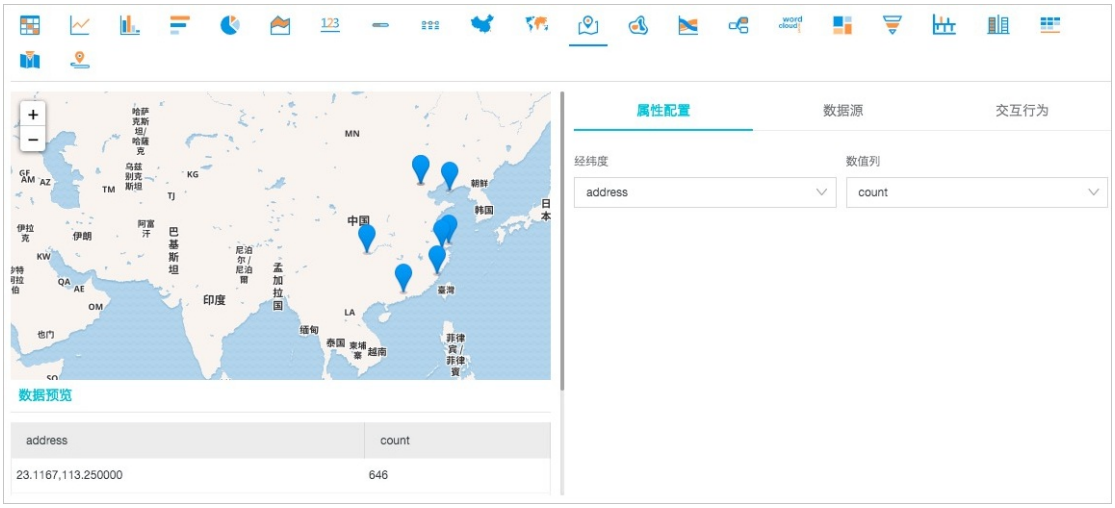
```
* | select  ip_to_geo(remote_addr) as address, count(1) as count group by address order by count desc limit 10
```

- 数据集

address	count
39.9289,116.388	771
39.1422,117.177	724
29.5628,106.553	651
30.2936,120.161420	577
26.0614,119.306	545
34.2583,108.929	486

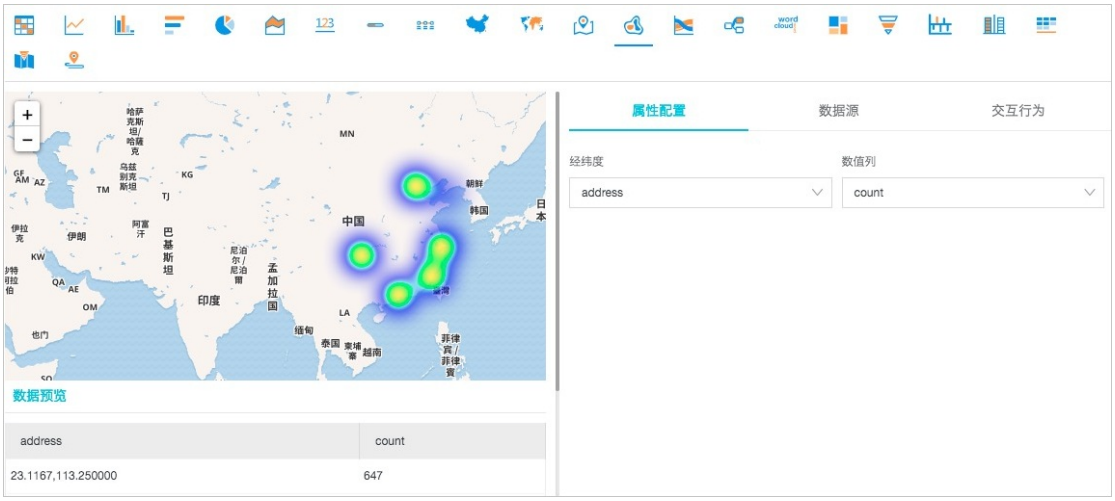
经纬度信息选择address，**数值列**选择count。

图 3. 高德地图-点图



默认返回点图。如数据点分布密集，您也可以切换为热力图。

图 4. 高德地图-热力图



4.11.1.11. 流图

流图（Flow Chart）也叫主题河流图（ThemeRiver），是围绕中心轴线进行布局的一种堆叠面积图。不同颜色的条带状分支代表了不同的分类信息，条状带的宽度映射了对应的数值大小。此外，原数据集中的时间属性，默认映射到X轴上，是一个三维关系的展现。

流图可以通过图表类型切换为线图和柱状图，需要注意的是柱状图默认以层叠形式展现，不同分类数据的起点是从上个柱状的顶部开始。

基本构成

- X轴（横轴）
- Y轴（纵轴）
- 条状

使用步骤

1.

2. 在**统计图表**页签中，选择  。

3. 在右侧**属性配置**中配置图表属性。

配置项

配置项	说明
图表类型	提供线图（默认）、面积图、以及柱状图（层叠）。
X轴	一般为有序数据类别（时间序列）。
Y轴	可以配置一列或多列数据对应到左轴数值区间。
聚合列	需要在第三维上聚合的信息。
图例位置	图例在图表中的位置，可以配置为上、下、左和右。
格式化	将数据按照指定格式进行显示。
间距	坐标轴距离图表边界距离，包括**上边距**、**下边距**、**右边距**和**左边距**。

示例

流图适合三维关系的展示，时间-分类-数值的展现。

```
* | select date_format(from_unixtime(__time__ - __time__% 60), '%H:%i:%S') as minute, count(1) as c,  re
quest_method group by minute, request_method order by minute asc limit 100000
```

X轴选择 `minute` ，Y轴选择 `c` ，聚合列选择 `request_method` 。

图 1. 流图

4.11.1.12. 桑基图

桑基图 （Sankey Diagram），是一种特定类型的流图，用于描述一组值到另一组值的流向。

桑基图 适合网络流量等场景，通常包含3组值 `source` 、 `target` 以及 `value` 。 `source` 和 `target` 描述了节点的关系，而 `value` 描述了该 `source` 和 `target` 之间边的关系。

功能特点

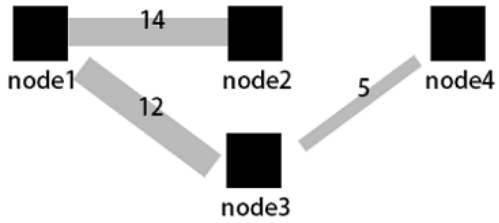
桑基图具有以下特点：

- 起始流量和结束流量相同，所有主支宽度的总和与所有分出去的分支宽度总和相等，保持能量的平衡。
- 在内部，不同的线条代表了不同的流量分流情况，它的宽度成比例地显示此分支占有的流量。
- 节点不同的宽度代表了特定状态下的流量大小。

例如以下数据可以用桑基图表示：

source	target	value
node1	node2	14
node1	node3	12
node3	node4	5
...	...	...

桑基图如此描述上述数据的关系：



基本构成

- 节点
- 边

使用步骤

1.

2. 在**统计图表**页签中，选择桑基图。

3. 在右侧**属性配置**中配置图表属性。

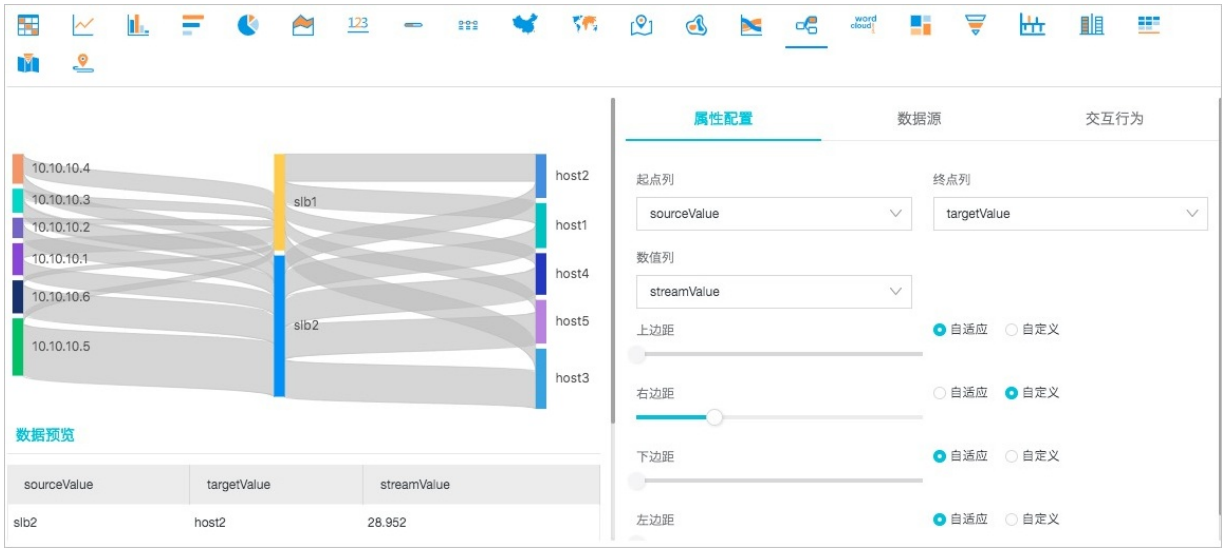
属性配置

配置项	说明
起点列	描述起始节点。
终点列	描述终点节点。
数值列	链接起点节点和终点节点的值。
边距	坐标轴距离图表边界距离，包括**上边距**、**下边距**、**右边距**和**左边距**。

桑基图示例

如果日志字段包含了 `source` 、 `target` 和 `value` ，即每条日志本身就是节点和边的关系，可以通过嵌套子查询获取到 `steamValue` 的总和。

```
* | select sourceValue, targetValue, sum(streamValue) as streamValue from (select sourceValue, targetVal
ue,
 streamValue, __time__ from log group by  sourceValue, targetValue, streamValue, __time__ order by __tim
e__ desc) group by sourceValue,
 targetValue
```



4.11.1.13. 词云

词云，是文本数据的视觉表示，由词汇组成类似云的彩色图形，用于展示大量文本数据。每个词的重要性以字体大小或颜色显示，能最让用户最快速地感知某一些关键词的权重大小。

基本构成

词云类型的图表为您展示经过数据计算排列的词。

使用步骤

1.

2. 在**统计图表**页签中，选择词云。

3. 在右侧**属性配置**中配置图表属性。

配置项

配置项	说明
词列	代表要展示的一组词的信息。
数值列	每一个词对应的数值信息。
字号	合理调整字号范围以适应画布。 • 最小字号（10px-24px） • 最大字号（50px-80px）

示例

分析NGINX日志中domain分布：

```
* | select domain, count(1) as count group by domainorder by count desc limit 1000
```

词列为 `domain` ，数值列为 `count` 。

4.11.1.14. 矩形树图

矩形树图，即矩形式树状结构图（Treemap），用矩形面积表示数据的大小。各个小矩形的面积越大，表示占比越大。

基本构成

计算排列得到的矩形块。

使用步骤

1.

2. 在**统计图表**页签中，选择矩形树图。

3. 在右侧**属性配置**中配置图表属性。

属性配置

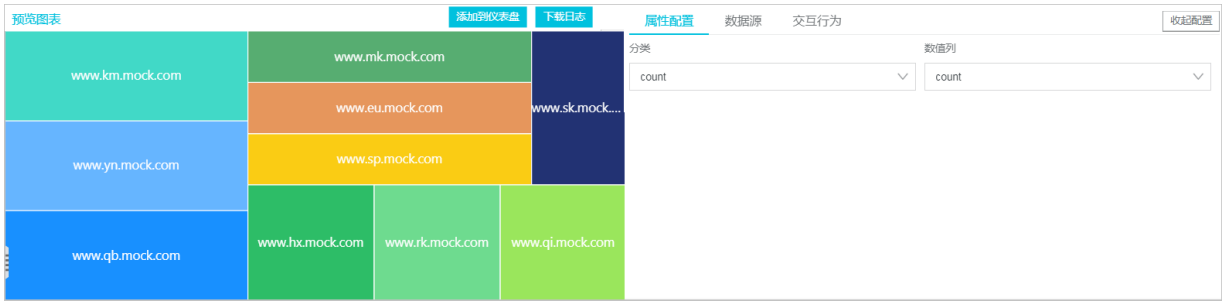
配置	说明
分类	表示数据类别的字段。
数值列	数值字段，某个类别对应的数值越大，其矩形框越大。

示例

分析NGINX日志中hostname分布。

```
* | select hostname, count(1) as count group by hostname order by count desc limit 1000
```

设置**分类**为 `hostname` ，**数值列**为 `count` 。



4.11.2. 仪表盘

4.11.2.1. 仪表盘简介

仪表盘是日志服务提供的实时数据分析大盘。您可以将常用的查询语句以图表形式展示，并将多项分析图表保存到仪表盘中。通过仪表盘可以一次性查看多个分析语句的分析图表，当您打开或刷新仪表盘时，这些分析图表会自动执行一遍查询分析语句。添加图表到仪表盘时，还可以设置下钻分析，设置之后，在仪表盘页面中单击该图表，可以得到更深维度的分析结果。

限制说明

- 每个Project最多可创建50个仪表盘。
- 每个仪表盘最多可包含50张分析图表。

功能介绍

仪表盘分为显示模式和编辑模式。

- 显示模式
  在显示模式下，支持对仪表盘页面进行多种显示设置，包括：
  - 仪表盘显示设置：例如设置仪表盘的全局时间、对图表设置告警、设置仪表盘页面自动刷新、设置仪表盘全屏显示、设置标题显示方式、根据仪表盘过滤器过滤图表数据等。
  - 图表显示设置：查看指定图表的分析详情、设置指定图表的时间区间、对指定图表设置告警、下载日志、下载图表、查看是否设置了下钻等。
- 编辑模式
  在编辑模式下，支持对仪表盘进行多种变更操作，包括：
  - 仪表盘设置：支持将仪表盘作为画布，为其添加图表元素，如Markdown图表、自定义图表、文本、图标等图表元素；还可以在图表元素之间添加连接线，连接线支持根据图表位置自适应；添加仪表盘过滤器，添加后在显示模式下可以过滤图表数据。另外，为便于排版，可以设置显示网格线，让图标等元素的位置工整有序。
  - 图表设置：支持在仪表盘编辑模式下编辑图表，例如修改分析图表的语句、属性、下钻配置等交互行为。

4.11.2.2. 创建和删除仪表盘

在日志服务控制台中输入查询分析语句，设置图表之后，可以将图表保存在仪表盘中，方便下次查看。仪表盘中可以展示50张分析图表，支持多种显示和自定义编辑设置。

前提条件

已开启并配置索引。

创建仪表盘

1. 登录日志服务控制台。
2. 单击目标Project名称。
3. 单击日志库名称后的  图标，选择**查询分析**。
4. 在搜索框中输入查询分析语句，并单击**查询/分析**。
5. 页面自动跳转到**统计图表**页面，请在右侧**属性配置**页签中设置图表属性。
6. **可选：**设置占位符变量。
   如果其他图表的下钻事件为跳转到这个仪表盘，设置占位符变量后，单击其他图表时会跳转到这个仪表盘，占位符变量替换为触发下钻事件的图表值，并以替换变量后的查询语句刷新仪表盘。详细信息请参见下钻分析。
   i. 进入**数据源**页签，在**查询语句**中划选部分查询语句。
   ii. 单击**生成变量**生成占位符变量。

iii. **设置变量配置。**

配置	说明
变量名	为占位符变量命名。如果占位符变量名称与触发下钻事件的仪表盘图表设置的变量相同时，在下钻事件中占位符变量替换为触发下钻事件的图表值。
默认值	占位符变量在当前仪表盘中的默认值。
匹配模式	可以选择全局匹配或者精确匹配。
生成结果	确认变量配置。

7. 设置下钻分析。
   设置下钻分析后，该图表在仪表盘中可以单击进入更深粒度的查询分析，例如跳转到其他仪表盘、跳转到快速查询等。详细信息请参见下钻分析。
   i. **打开交互行为**页签。
   ii. **指定开启下钻分析的列，展开对应的列，并设置事件行为。**
   iii. **填写事件行为对应的设置。**

8. **单击添加到仪表盘并指定仪表盘和图表名称。**

配置	说明
操作类型	◦ **添加到已有仪表盘**：将当前分析图表添加到已有的仪表盘。 ◦ **新建仪表盘**：将当前分析图表添加到新建的仪表盘中。
仪表盘列表	选择已有的仪表盘名称。 **说明** 仅在**操作类型**为**添加到已有仪表盘**时指定。
Dashboard名称	新建的仪表盘名称。 **说明** 仅在**操作类型**为**新建仪表盘**时指定。
图表名称	为当前分析图表命名。该名称会作为图表标题显示在仪表盘页面中。

9. 单击**确定**，结束配置。
   您可以通过多次添加的方式，将多个分析图表添加到一个仪表盘中。

删除仪表盘

当不需要某个仪表盘时，可以删除仪表盘。删除后不可恢复。

1. 在日志服务控制台单击Project名称。
2. 单击左侧导航栏的**仪表盘**图标。
3. 单击目标仪表盘后的图标，选择 **删除**。

4.11.2.3. 显示模式

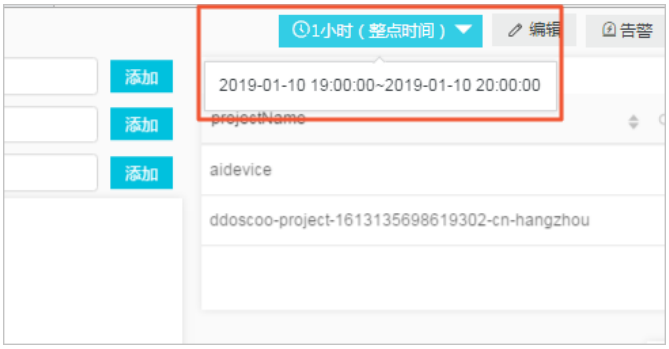
查看仪表盘时，默认处于显示模式下，显示模式下可以直观、清晰地查看该仪表盘下的所有分析图表。日志服务同时提供一系列针对仪表盘的显示配置，包括添加页面元素、设置自动刷新、设置标题显示方式等。

设置仪表盘时间

仪表盘时间即仪表盘中所有分析图表统一的时间，设置后，所有分析图表展示的是同一时段的查询分析结果。如果需要设置单个图表的时间范围，请参见设置指定图表的时间范围。

> **说明** 在仪表盘页面单击**时间选择**，您可以选择查询时间范围。但是选定的查询时间范围仅供临时查看结果，系统不会保存。您下次查看报表时，系统仍会为您展示默认的时间范围。

1. 登录日志服务控制台。
2. 单击目标Project名称。
3. 单击左侧导航栏的**仪表盘**图标。
4. 单击目标仪表盘后的图标，选择 **详情**。
5. 单击**时间选择**选择时间范围。
   支持设置仪表盘时间为：
   - 相对时间：表示查询距离当前时间1分钟、5分钟、15分钟等时间区间的的日志数据。例如当前时间为19:20:31，设置相对时间1小时，表示查询18:20:31~19:20:31的日志数据。
   - 整点时间：表示查询最近整点1分钟、5分钟、15分钟等时间区间的日志数据。例如当前时间为19:20:31，设置整点时间1小时，表示查询18:00:00~19:00:00的日志数据。
   - 自定义时间：表示查询指定时间范围的日志数据。
6. **时间选择**按钮变更为当前设置的时间范围，鼠标移至按钮上，可核对当前设置的时间范围。



进入编辑模式

单击**编辑**进入仪表盘的编辑模式，在编辑模式下，支持对仪表盘进行多种变更操作，包括将仪表盘作为画布，为其添加图表元素，如Markdown图表、自定义图表、文本、图标等图表元素等操作。详细说明请参见编辑模式。

设置告警

在仪表盘右上角单击**告警列表 > 新建**或**告警列表 > 修改**可以新建告警或设置告警，告警中需要关联一个或多个分析图表。

如何设置告警，请参见设置告警。

设置页面刷新方式

刷新仪表盘时可以选择手动刷新一次或设置自动刷新：

- 单击**刷新 > 仅一次**，表示立即刷新一次仪表盘。
- 单击**刷新 > 自动刷新**，表示按照指定间隔自动刷新仪表盘。

  自动刷新可设置为15秒、60秒、5分钟或15分钟。

> **说明** 当浏览器页签处于不活跃状态时，时间计数可能存在偏差。

分享仪表盘页面

单击**分享**，可以自动复制当前仪表盘的链接，您可以将该链接手动发送给有仪表盘查看权限的其他用户。其他用户看到的仪表盘页面中保留分享者一系列设置，例如图表的查询时间范围、标题显示方式等。

> **说明** 分享仪表盘页面前，必须授予其他用户查看该仪表盘的权限。

全屏展示

单击**全屏**，即可进入全屏模式，页面全屏显示仪表盘中的图表信息。适用于数据展示和报告场景。

设置标题显示方式

在仪表盘页面中单击**标题设置**，可以设置仪表盘中所有分析图表标题的显示样式，包括：

- 并排显示标题+时间
- 仅标题
- 仅时间

重置时间

单击**重置时间**可以重置所有图表的时间范围，即恢复所有分析图表的默认时间，多用于改变时间维度后还原到初始状态。

分析图表的查看设置

- 查看分析详情

  展开指定分析图表右上角的折叠列表，并单击**查看分析详情**，页面会自动跳转到对应的查询分析页面，页面显示当前图表的查询语句和属性设置等信息。

- 设置指定图表的时间范围

  展开指定分析图表右上角的折叠列表，并单击**选择时间区域**，可以设置指定图表的时间范围。设置后，仅该图表的时间区间会变更，其他图表时间保持不变。

- 设置图表告警

  展开指定分析图表右上角的折叠列表，并单击**新建告警**，可以设置基于该分析图表的告警和通知方式。如何设置告警，请查看设置告警。

- 下载日志

  展开指定分析图表右上角的折叠列表，并单击**下载日志**，以CSV格式下载当前时间区间对应的原始日志分析结果。

- 下载图表

  展开指定分析图表右上角的折叠列表，并单击**下载图表**，以PNG图片格式下载当前的查询分析图表。

- 查看是否设置了下钻分析

  展开指定分析图表右上角的折叠列表，如果列表中的手指图标为红色，表示当前图表已设置下钻分析；否则图标为灰色。

> **说明** 不同图表类型有不同的展示选项，例如自定义添加的图表、Markdown图表等特殊图表无法查看分析详情，因为它不是一个查询分析图表。

4.11.2.4. 编辑模式

编辑仪表盘时，仪表盘处于编辑模式下。编辑模式支持对仪表盘进行一系列变更和设置操作。

- 仪表盘设置：
  - 在页面左上角修改仪表盘名称。
  - 支持将仪表盘作为画布，为其添加图表元素，如Markdown图表、自定义图表、文本、图标等图表元素。
  - 在图表元素之间添加连接线，连接线支持根据图表位置自适应。
  - 添加仪表盘过滤器，添加后在显示模式下可以过滤图表数据。
  - 为便于排版，可以设置显示网格线，让图标等元素的位置工整有序。
  - 通过菜单栏控制仪表盘中图表的属性设置，例如添加、删除、撤销操作、配置层级和图表大小、位置。
- 图表设置：支持在仪表盘编辑模式下编辑图表，例如修改分析图表的语句、属性、下钻配置等交互行为。

> **说明** 在仪表盘编辑模式下的所有变更，都必须在页面右上角单击**保存**才会生效。

图表元素

仪表盘编辑模式下，支持插入以下图表元素：

- 常见图标

  日志服务提供一系列常见图标以便在仪表盘中展示。在菜单栏中拖动该图标到指定位置即可。

- 文本

  在仪表盘菜单栏中拖动文本图标到指定位置，可以插入文本。双击文本框可以修改文本内容。

- Markdown图表

  日志服务还支持在仪表盘中增加Markdown图表，该图表使用Markdown语言编辑。

  在仪表盘菜单栏中拖动Markdown图标到指定位置，可以插入Markdown文本。在其右上角的隐藏菜单中单击**编辑**可以设置Markdown文本内容。

- 过滤器

  在日志服务仪表盘中增加过滤器配置，可以过滤器缩小查询范围或替换占位符变量，即对整个仪表盘进行查询过滤（Filter）和变量替换（Variables）操作。

  在仪表盘菜单栏单击过滤器图标，在弹出页面中设置过滤器，仪表盘过滤器默认位置为仪表盘左上角。在其右上角的隐藏菜单中单击**编辑**可以修改过滤器设置。

- 自定义SVG
  支持直接上传SVG到仪表盘。在操作栏中单击SVG图标，单击或拖拽上传SVG即可。

> **说明** SVG大小不超过10 KB。

- 自定义HTTP图片

  支持直接上传HTTP图片到仪表盘。在操作栏中单击对应图标，输入图片的HTTP链接，并单击**确定**即可。

排版布局

在仪表盘编辑模式下，所有的分析图表与各类图表元素都被至于一个可以随意进行拖拽的网格画布中，您可以自由地对每一个图表进行拖动和缩放（连接线除外）。画布水平方向限制为1000个单位，每个单位宽度为 `当前浏览器宽度 / 1000` ，垂直方向无限制，每个单位为1像素。排版前，可以在右上角单击**显示网格线**，网格线便于设置图表的位置和间距。

支持进行以下排版操作：

- 调整图表位置
  - 直接拖动图表到指定位置。
  - 选中指定图表后，在操作栏中通过设置**左边距**和**上边距**，调整图片的位置。
- 调整图表宽度、高度
  - 选中指定图表后，在右下角拖拽，调整图表大小。
  - 选中指定图表后，在操作栏中通过设置**宽度**和**高度**，设置图表的大小。
- 添加图表连接线

  在图表之间添加带方向的连接线后，调整表格的位置和大小时，连接线会同时移动，便于展示图表间的相对关系。

  选中图表后，长按其边框中的方框标识，此处为连接线的起点，页面会自动展示可作为连接线终点的区域，将鼠标移动至该位置即可。
- 图表间支持设置层级关系，选中指定图表后，通过操作栏可将图表层级设置为置顶或置底。

图表设置

在仪表盘编辑模式下，支持对图表元素进行以下操作：

- 编辑：修改分析图表的语句、属性、下钻设置等交互行为。
  i. 在仪表盘页面右上角单击**编辑**。
  ii. 找到目标图表，单击图表右上角的 ⋮ > **编辑**。
  iii. 修改分析图表的查询语句、**属性配置**、**数据源**信息或**交互行为**。
  iv. 单击**预览**，并单击**确定**。
  v. 在仪表盘页面右上角单击**保存**。
- 复制：创建指定图表元素的副本，保留所有配置信息。
  i. 在仪表盘页面右上角单击**编辑**。
  ii. 找到目标图表，单击图表右上角的 ⋮ > **复制**。
  iii. 拖动图片副本到指定位置，设置边距和大小。
  iv. 在仪表盘页面右上角单击**保存**。
- 删除：从仪表盘中删除指定图表元素。
  i. 在仪表盘页面右上角单击**编辑**。
  ii. 找到目标图表，单击图表右上角的 ⋮ > **删除**。
  iii. 在仪表盘页面右上角单击**保存**。

4.11.2.5. 下钻分析

日志服务分析图表提供了向下钻取（drill down）的功能，您可以添加一个图表到仪表盘，通过改变下钻列表中的配置项，从而使得仪表盘中的分析图表具备更强大的功能。

前提条件

- 已开启并配置索引。详情请参见开启并配置索引。
- 已配置要跳转到的快速查询、仪表盘和自定义链接。
- 如果选择添加变量，则需要在跳转到的快速查询和仪表盘配置中配置查询语句变量占位符。详情请参见快速查询和创建和删除仪表盘。

背景信息

钻取是数据分析中不可缺少的功能之一，通过改变数据维度的层次、变换分析的粒度从而关注数据中更详尽的信息。它包括向上钻取（roll up）和向下钻取（drill down），向下钻取是在分析时加深维度，对数据进行深入的查看。通过逐层下钻，能挖掘数据更大的价值，及时做出更加正确的决策。

日志服务支持下钻分析的图标包括：表格、线图、柱状图、条形图、饼图、单值图、面积图、矩形树图。

配置步骤

1. 登录日志服务控制台。
2. 单击目标Project名称。
3. 单击日志库名称后的  图标，选择**查询分析**。
4. 输入查询分析语句并设置时间范围，然后单击**查询/分析**。
5. 在**统计图表**页签中选择**图表类型**，并设置**属性配置**。
6. 在**交互行为**页签中设置下钻**事件行为**。
   - **不开启**：表示不开启下钻功能。
   - **打开日志库**：设置下钻事件为打开日志库。

     如果您在页面上设置了过滤语句，单击图表内容时，该过滤语句会自动增加到跳转到的日志库页面上作为查询语句。

配置	说明
请选择日志库	需要跳转到的日志库名称。
打开新窗口	开启该选项后，当触发交互行为时将在新窗口打开对日志库。
时间范围	设置跳转到的日志库的查询分析时间范围。可以设置为： ▪ **预设**：仪表盘页面中单击图表跳转到日志库后，保持查询时的默认时间范围，即15分钟（相对）。 ▪ **继承图表时间**：跳转后，日志库的查询语句对应的时间范围默认为触发事件时仪表盘中设置的图表的时间。 ▪ 相对时间：跳转后，将跳转后日志库的查询时间设置为指定的相对时间。 ▪ 整点时间：跳转后，将跳转后日志库的查询时间设置为指定的整点时间。 默认为**预设**。
是否继承过滤	如果选择**继承筛选条件**，则会把触发事件仪表盘中添加的筛选条件同步到对应日志库的查询中，并以 `AND` 的方式添加到查询语句之前。
过滤	在**过滤**页签中输入**过滤语句**，语句中可以包含**可选参数域**。 如果配置了**过滤**，在仪表盘图表中单击跳转后，会自动为跳转到的日志库的查询增加查询语句，查询语句为此处配置的**过滤语句**。

   - **打开快速查询**：设置下钻事件为打开快速查询页面。
     支持同时设置变量和过滤。单击图表内容：
     ▪ 如果设置了变量，会用单击的图表值替换快速查询语句中设置的占位符，基于图表值进行更深层次的查询。
     ▪ 如果设置了过滤，会自动为跳转到的快速查询增加查询语句。

配置	说明
请选择快速查询	需要跳转到的快速查询名称。如何配置快速查询请参见快速查询 。
打开新窗口	开启该选项后，当触发交互行为时将在新窗口打开对应快速查询。
时间范围	设置跳转到的快速查询的时间范围。可以设置为： ▪ **预设**：仪表盘页面中单击图表跳转到快速查询后，保持快速查询的默认时间范围，即15分钟（相对）。 ▪ **继承图表时间**：跳转后，查询语句对应的时间范围默认为触发事件时仪表盘中设置的图表的时间。 ▪ 相对时间：跳转后，将跳转后的快速查询时间设置为指定的相对时间。 ▪ 整点时间：跳转后，将跳转后的快速查询时间设置为指定的整点时间。 默认为**预设**。

是否继承过滤	如果选择**继承筛选条件**，则会把触发事件仪表盘中添加的筛选条件同步到快速查询中，并以 `AND` 的方式添加到查询语句之前。
过滤	在**过滤**页签中输入**过滤语句**，语句中可以包含**可选参数域**。 如果配置了**过滤**，在仪表盘图表中单击跳转后，会自动为跳转到的快速查询增加查询语句，查询语句为此处配置的**过滤语句**。
变量	在**变量**页签中单击**添加变量**，并指定： ■ **替换变量名**：触发下钻分析的变量，单击即可跳转。 ■ **替换值所在列**：以指定列的对应值进行替换。当有多列时，可以设置为当前列和其他列。当前列为设置下钻的列，即**替换值所在列**所在的列；其他列可以是设置下钻分析的图表中其他任意列。 当跳转到的快速查询中的查询语句变量和本次添加的变量名称一致时，会将快速查询语句中的变量替换为触发下钻事件的图表值，从而灵活改变目标快速查询中的查询语句。 **说明** ■ 如果选择添加变量，则需要事先在跳转到的快速查询中配置查询语句变量占位符。 ■ 最多可以添加5个变量。

- **打开仪表盘**：设置下钻事件为打开仪表盘。
  仪表盘中的图表实际上是查询语句的图表形式的结果。单击上层仪表盘中的图表内容时：
  - 如果设置了变量，且预先在跳转到的仪表盘图表查询语句中设置了占位符，会用单击的图表值替换预设的占位符。
  - 如果设置了过滤，会为跳转到的仪表盘增加过滤条件，基于图表值进行更深层次的查询。

配置	说明
请选择仪表盘	需要跳转到的目标仪表盘名称，如何配置仪表盘请参见创建和删除仪表盘。
打开新窗口	开启该选项后，当触发交互行为时将在新窗口打开对应仪表盘。
时间范围	设置跳转到的仪表盘的时间范围。可以设置为： ■ **预设**：仪表盘页面中单击图表跳转到仪表盘后，跳转到的仪表盘时间范围保持不变，即保留所有图表的预设时间。 ■ **继承图表时间**：跳转后，仪表盘中图表对应的时间范围默认为触发事件时仪表盘中设置的图表的时间。 ■ 相对时间：跳转后，将跳转后的仪表盘时间设置为指定的相对时间。 ■ 整点时间：跳转后，将跳转后的仪表盘时间设置为指定的整点时间。 默认为**预设**。
是否继承过滤	如果选择**继承筛选条件**，则会把触发事件仪表盘中添加的筛选条件同步到跳转到的仪表盘中，并以 `AND` 的方式添加到查询语句之前。
过滤	在**过滤**页签中输入**过滤语句**，语句中可以包含**可选参数域**。 如果配置了**过滤**，在仪表盘图表中单击跳转后，会自动为跳转到的仪表盘添加过滤条件，过滤条件为此处配置的**过滤语句**。
变量	在**变量**页签中单击**添加变量**，并指定： ■ **替换变量名**：触发下钻分析的变量，单击即可跳转。 ■ **替换值所在列**：以指定列的对应值进行替换。当有多列时，可以设置为**默认列**和其他列。**默认列**即当前列，也就是设置下钻分析的列；其他列可以是设置下钻分析的图表中其他任意列。 当跳转到的仪表盘中的分析图表查询语句变量和本次添加的变量名称一致时，会将分析图表查询语句中的变量替换为触发下钻事件的图表值，从而灵活改变目标仪表盘中分析图表的查询语句。 **说明** ■ 如果选择添加变量，则需要事先在跳转到的仪表盘中配置查询语句变量占位符。 ■ 最多可以添加5个变量。

- **自定义http链接**：设置下钻事件为打开自定义http链接。

http链接中的路径部分表示访问的目的端文件的层级路径，您可以在定义http链接的路径部分添加可选参数域，单击仪表盘中的图表内容时，会用图表值替换http链接中的参数，跳转到重新定位的http链接中。

配置	说明
请输入链接地址	需要跳转到的目标地址。
可选参数域	单击可选参数变量，可以将链接地址中的某一部分替换为触发下钻事件的图表值。

7. 单击**添加到仪表盘**，设置仪表盘名称和图标名称。

示例

例如，在名为accesslog的Logstore中存放采集到的Nginx访问日志，名为RequestMethod的仪表盘中展示Nginx日志的常见分析场景，名为destination_drilldown的仪表盘展示PV随时间分布的趋势。您可以为请求方法的分类表格设置下钻分析，将其添加到RequestMethod仪表盘中，并将下钻事件设置为跳转到destination_drilldown仪表盘。在RequestMethod仪表盘中单击各个请求方法即可跳转到destination_drilldown仪表盘查看对应的PV趋势。

1. 设置跳转到的仪表盘。
   i. 根据请求类型筛选日志，并查看PV随时间的变化。
   查询语句如下：

```
request_method: * | SELECT date_format(date_trunc('minute', __time__), '%H:%i:%s') AS time,
COUNT(1) AS PV GROUP BY time ORDER BY time
```

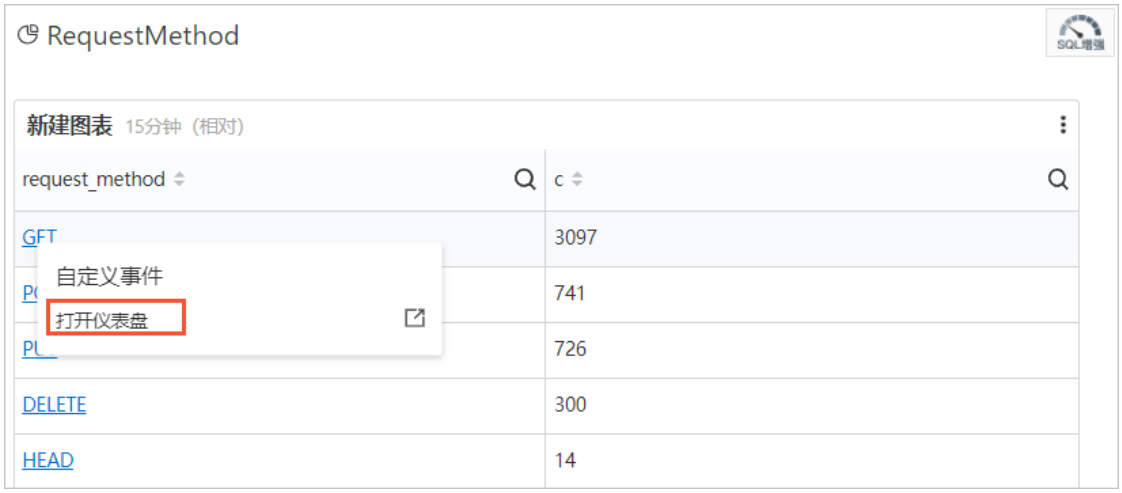
   ii. 通过折线图表示查询结果，并将折线图保存到仪表盘中。
   保存到仪表盘时，将 `*` 设置为占位符，并命名为method，如果跳转到这个快速查询的下钻事件变量同样为method，即可用单击的图表值替换 `*` ，再次执行查询分析。

2. 设置触发下钻分析的图表，并将其添加到仪表盘。
   i. 在查询页面通过SQL语句分析Nginx访问日志中各种请求方法（request_method）的日志条数，并将结果以表格形式表示。
   查询分析语句：

```
*|SELECT request_method, COUNT(1) AS c GROUP BY request_method ORDER BY c DESC LIMIT 10
```

   ii. 为 `request_method` 一列设置下钻分析。

iii. 在RequestMethod仪表盘中单击GET请求。



iv. 成功跳转到destination_drilldown仪表盘。
页面自动跳转到1中设置的仪表盘，原查询语句中的 `*` 已替换为单击的图表值 `GET` ，表示查看GET请求PV随时间的变化。

4.11.2.6. 仪表盘过滤器

在日志服务仪表盘中增加过滤器配置，可以缩小查询范围或替换占位符变量，即对整个仪表盘进行查询过滤（Filter）和变量替换（Variables）操作。

前提条件

- 已开启并配置索引。

- 已创建仪表盘，如果过滤器类型为变量替换，需要设置好占位符变量。

背景信息

日志服务仪表盘中的每一张图表是一个查询分析语句，在仪表盘中增加过滤器也就是为所有图表批量增加过滤条件，或者批量替换所有图表中设置的占位符变量。过滤器配置分为以下两种。

- 过滤器类型：指定key和value，并将其作为过滤条件增加到查询语句`[search query]`前。新的查询语句为`key: value AND [search query]`，表示在原查询语句的结果中，查找包含`key:value`的日志。**过滤器**类型的过滤器中，value可以多选，也可以直接输入。多选时过滤条件之间为`or`关系。
- 变量替换类型：指定变量占位符，如果仪表盘中有已设置该变量占位符的图表，则将图表查询语句中的该占位符变量替换为选择的value值。

基本构成

每个过滤器图表可以有一个或者多个过滤器构成，每个过滤器主要包含以下元素。

- 过滤器操作Key值。
- Key对应的列表项。

操作步骤

1. 登录日志服务控制台。
2. 单击目标Project名称。
3. 单击左侧导航栏的**仪表盘**图标。
4. 在仪表盘列表中单击指定仪表盘名称。
5. 在仪表盘页面右上角单击**编辑**，进入编辑模式。
6. 单击过滤器图标，并设置过滤器配置。完成后单击**确定**。

表 1. 过滤器配置项

配置项	说明
过滤器名称	过滤器名称。
显示设置	包含如下选项： ◦ 标题：打开标题开关，为过滤器增加标题。 ◦ 边框：打开边框开关，为过滤器增加边框。 ◦ 背景：打开背景开关，为过滤器增加白色背景。
类型	过滤器的类型，包括： ◦ **过滤器**：列表项表示过滤条件中的Value。您可以设置多个Value，生成过滤器之后可以在查看仪表盘时根据需求选择Value。 ◦ **变量替换**：列表项为指定变量占位符的替换值。您可以设置多个替换值，生成过滤器之后可以在查看仪表盘时根据需求选择替换值。
Key值	◦ **过滤器**类型：**Key值**为过滤条件中的key。 ◦ **变量替换**类型：**Key值**为指定的变量占位符。 **说明** 变量占位符必须是前提条件中已配置的变量占位符，才能成功替换。
别名	列的别名，仅在**过滤器**类型中指定。设置后，在仪表盘过滤器中显示别名。
全局过滤	是否在所有字段中过滤Value，默认为关闭状态，仅在**过滤器**类型中指定。 ◦ 开启**全局过滤**，表示在所有字段中过滤Value。 ◦ 关闭**全局过滤**，表示仅在指定Key中过滤Value。
静态列表项	在输入框中输入列表项的值，并单击**添加**，设置列表项。
添加动态列表项	过滤器中动态显示的列表项，列表项为预设的查询分析语句当前的查询结果。 打开**添加动态列表项**，选择日志库，选择是否继承过滤（查询语句是否继承当前页面的过滤条件），在输入框中输入查询分析语句，选择时间后单击**查询**，可以预览动态列表项。

应用场景

过滤器多用于在当前仪表盘中动态修改查询条件和对图表中已经存在的变量占位符进行变量替换。每一张图表实际为一个查询分析语句，满足 `[search query] | [sql query]` 的形式，过滤器实质上会操作该查询分析语句。

- 如果为过滤器，则会在 `[search query]` 前加上过滤的值，以 `AND` 连接为新的查询语句，即 `key: value AND [search query]`
- 如果为变量替换过滤器，则会查询整个仪表盘存在变量占位符的图表，将对应名称的变量占位符替换为选择的 `value` 值

示例1：基于不同时间粒度

例如采集到日志数据后，需要对采集到的日志数据进行实时查询与分析。

通过分析语句可以查看每分钟的访问PV，当需要查看秒级别的数据时，需要调整 `__time__ - __time__ % 60` 的值，传统做法为修改查询分析语句，多次查询时操作繁琐。此时可以通过过滤器完成变量替换。

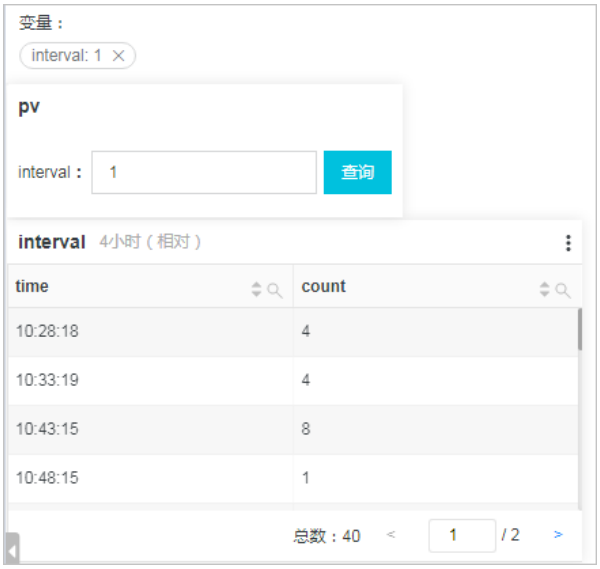
1. 通过以下语句查看分钟访问PV的数据。

```
* | SELECT date_format(__time__ - __time__ % 60, '%H:%i:%s') as time, count(1) as count GROUP BY time ORDER BY time
```

2. 将分析图表添加到仪表盘，并选中查询语句中的 `60` 生成变量占位符，变量名为 `interval` 。

3. 在仪表盘编辑页面添加过滤器。
   - **类型为变量替换**。
   - **Key值为** `interval` 。
   - **静态列表项为** `1` （表示每秒）和 `120` （表示每2分钟）。
4. 在过滤器中选择 `1` ，此时仪表盘为秒级别的粒度。
   替换变量后的查询语句。

```
* | SELECT date_format(__time__ - __time__ % 1, '%H:%i:%s') as time, count(1) as count GROUP BY time ORDER BY time
```

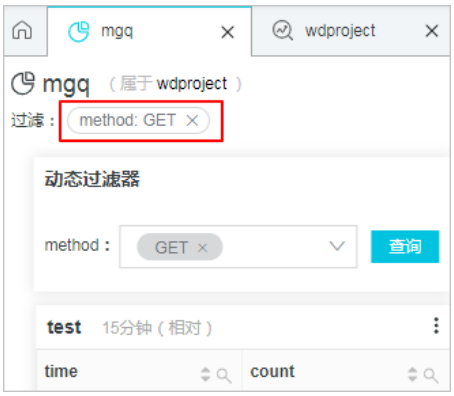


示例2：动态切换请求方法

通过添加动态列表项还可以动态切换不同的请求方法（method）。示例1中，查询语句为 `*` ，表示不设置任何过滤条件，即所有的日志都在查询范围之中。此时，可以再添加一个过滤器便于查看不同 `method` 的访问情况。

1. 增加过滤器并打开**添加动态列表项**开关。
   设置如下。
   - **类型**为过滤器。
   - **Key值**为 `method` 。
   - **请选择日志库**为当前仪表盘所在日志库。
   - **添加动态列表项**：通过输入查询语句，动态获取列表项。
2. 在过滤器的下拉列表项中选择 `POST` 。
   图表中只显示 `method` 为 `POST` 的访问。实质上已经变为如下查询分析语句。

```
(*) and (method: POST) | SELECT date_format(__time__ - __time__ % 60, '%H:%i:%s') as time, count(1) a
s count GROUP BY time ORDER BY time
```



4.11.2.7. Markdown图表

日志服务支持在仪表盘中增加Markdown图表，在Markdown图表插入图片、链接、视频等多种元素，使您的仪表盘页面更加友好。

背景信息

Markdown图表都是根据不同的需求来创建的。您可以在Markdown图表中插入背景信息、图表说明、页面注释和扩展信息等文字内容，优化仪表盘的信息表达；插入快速查询或其他Project的仪表盘链接，方便其他查询页面的跳转；插入自定义的图片，让您的仪表盘信息更加丰富、功能更为灵活。

应用场景

通过Markdown图表可以自定义跳转链接到当前Project的其他仪表盘，同时还可以插入图片以便快速区分。当需要对图表参数进行介绍时，也可以插入Markdown图表进行说明。

操作步骤

1. 登录日志服务控制台。
2. 单击目标Project名称。
3. 单击左侧导航栏的**仪表盘**图标。
4. 在仪表盘列表中单击指定仪表盘名称。
5. 在仪表盘页面右上角单击**编辑**，进入编辑模式。
6. 在编辑模式下，将操作栏中的Markdown图标  拖动到指定位置，即可创建markdown图表。
7. 选中新建的Markdown图表，从右上角展开菜单，并单击**编辑**。

配置项	说明
图表名称	您创建的Markdown图表名称。
显示边框	选择**显示边框**，为您的Markdown图表增加边框。
显示标题	选择**显示标题**，会在仪表盘中为您展示Markdown图表的标题。
显示背景	选择**显示背景**，为您的Markdown图表添加白色背景。
绑定查询	选择**绑定查询**并设置查询属性后，会在Markdown图表中动态显示查询结果。

8. **可选：**绑定查询。
   i. 选择待查询的日志库，在查询框中输入完整的查询分析语句。查询分析语句由查询语句和分析语句构成，格式为`查询语句 | 分析语句`。
   ii. 单击**15分钟（相对）**，设置查询的时间范围。
   iii. 单击**查询**，显示当前查询结果的第一条数据。
   iv. 单击字段旁的“⊕”，即可将查询结果放置在**markdown内容**中光标所在位置。
9. 编辑**Markdown内容**。
   在**Markdown内容**中输入您的Markdown语句，右侧的**图表展示**区域会实时展示预览界面。您可以根据预览内容调整Markdown语句。

修改Markdown图表

- 修改图表位置和大小
  i. 在**仪表盘**页面单击右上角的**编辑**。
  ii. 鼠标拖动Markdown图标到指定位置，拖动图表右下角调整图表大小。
  iii. 在页面右上角单击**保存**。
- 修改图表标题
  i. 在**仪表盘**页面单击右上角的**编辑**。
  ii. 单击指定Markdown图表，并在右上角的折叠列表中单击**编辑**。
  iii. **图表名称**中输入新的标题，并单击**确定**。
  iv. 仪表盘页面右上角单击**保存**，退出仪表盘，并在弹出对话框中单击**确定**。
- 修改图表内容
  i. 在**仪表盘**页面单击右上角的**编辑**。
  ii. 单击指定Markdown图表，并在右上角的折叠列表中单击**编辑**。
  iii. 修改图表配置，并单击**确定**。
  iv. 仪表盘页面右上角单击**保存**，退出编辑模式，并在弹出对话框中单击**确定**。
- 删除图表
  i. 在**仪表盘**页面单击右上角的**编辑**。
  ii. 单击指定Markdown图表，并在右上角的折叠列表中单击**删除**。
  iii. 仪表盘页面右上角单击**保存**，退出编辑模式，并在弹出对话框中单击**确定**。

常用Markdown语法

- 标题
  Markdown语句：

```
# 一级标题
## 二级标题
### 三级标题
```

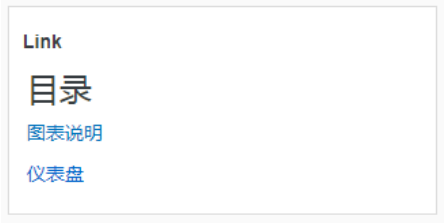
- 链接
  Markdown语句：

```
### 目录

[图表说明](https://xxx)

[仪表盘](https://xxx)
```

图 1. 链接预览



- 图片
  Markdown语句：

```
<div align=center>

![Alt txt][id]

With a reference later in the document defining the URL location

[id]: https://octodex.github.com/images/dojocat.jpg  "The Dojocat"
```

- 特殊标记
  Markdown语句：

```
---

__Advertisement :)__

==some mark==    `some code`
> Classic markup: :wink: :crush: :cry: :tear: :laughing: :yum:
>> Shortcuts (emoticons): :-)  8-) ;)

__This is bold text__

*This is italic text*

---
```

5.告警

5.1. 简介

日志服务支持根据仪表盘中的查询图表设置告警，实现实时的服务状态监控。

日志服务的告警功能基于仪表盘中的查询图表实现。在日志服务控制台**查询页面**或**仪表盘页面**设置告警规则，并指定告警规则的配置、检查条件和通知方式。设置告警后，日志服务定期对仪表盘的查询结果进行检查，检查结果满足预设条件时发送告警通知，实现实时的服务状态监控。

使用限制

限制项	说明
组合查询	组合查询的个数为1~3个。
字符串	字符串长度如果超过1024个字符，只会截取前1024个字符用于计算。
条件表达式	• 条件表达式长度为1~128个字符。 • 每个查询只会取查询结果的前100条用于计算条件表达式。 • 条件表达式计算次数不超过1000次，如使用组合查询，则组合计算的次数最多只会计算1000次。
查询区间	每个查询语句的查询区间时间跨度不能超过24小时。

告警中的查询语句

告警基于仪表盘中的分析图表，而分析图表实质上是一条查询分析语句的可视化查询结果。其中，查询语句可以是查询语句或查询分析语句。

- 查询语句：直接返回查询条件命中的日志数据。
- 查询分析语句：对查询条件命中的日志进行统计，返回统计结果。
- 查询语句
  例如，查询最近 15 分钟内包含 error 的数据，条件为 error，一共有 154 条。每条内容都是 Key、Value 组合，您可以对某个 Key 下的 Value 设置告警条件。

  > **说明** 对于查询结果一次超过 100 条的情况，告警规则只判断前100条 ，只有前100条中任意一条符合条件，才会触发告警。

- 查询分析语句
  例如查询所有日志中状态码为ok的日志比例，查询语句如下（查询语法请参见查询语法）：

  ```
  * | select sum(case when status='ok' then 1 else 0 end) *1.0/count(1) as ratio
  ```

  因此，可以设置告警检查条件为 `ratio < 0.9` ，表示当状态码为ok的日志小于总日志数的90%时进行告警。

5.2. 设置告警任务

5.2.1. 设置告警

日志服务支持在查询页面或仪表盘页面设置告警，并在满足告警条件时发送告警信息。

前提条件

- 已采集到日志数据。
- 已开启并配置索引。

背景信息

告警基于查询分析图表设置，您可以在查看图表时，将图表保存在仪表盘中，同时另存为告警，也可以在仪表盘页面中对已有的图表设置告警。

- 创建图表并设置告警
  将当前的查询分析语句保存在仪表盘中，并为查询分析语句设置告警。在查询页面设置告警时，您需要指定图表保存到的仪表盘名称和图表名称。

- 在仪表盘中对已有图表设置告警
  可对仪表盘中的一个或多个图表设置告警。为多个图表设置告警时，可以设置组合触发条件。

本文档以在仪表盘中对已有图表设置告警为例。

> **说明** 如果仪表盘中的分析图表绑定了告警规则，更新图表的查询分析语句后，需要手动更新告警规则，将告警规则中绑定的查询分析语句修改为更新后的语句。详细说明请参见修改告警规则。

常见告警配置案例请参见告警常见问题。

操作步骤

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project名称。
3. 在左侧导航栏中单击**仪表盘**图标。
4. 单击指定仪表盘名称。
5. 在页面右上角单击**告警列表 > 新建**。
6. 设置告警规则并单击 **下一步**。
   告警配置信息如下：

规则	说明
告警名称	告警的名称。名称长度为1~64个字符。

关联图表	设置告警中关联的图表。 设置关联图表时，**查询区间**为服务端每次执行查询时，读取的数据时间范围，支持相对时间与整点时间。例如，执行时间点为14:30:06，设置**查询区间**为15分钟（相对），则查询区间为 14:15:06- 14:30:06；设置**查询区间**为15分钟（整点时间），则查询区间为：14:15:00- 14:30:00。 需要添加多个图表时，只需多次添加并设置即可。图表名称前的编号为该图表在告警中的编号，您可以在触发条件中通过编号指定关联的图表。
频率	服务端每次执行告警检查的时间。
触发条件	判断告警是否触发的条件表达式，满足该条件时会根据**执行间隔**和**通知间隔**发送告警通知。 例如，您可以设置为 `pv%100 > 0 && uv > 0` 。 **说明** 触发条件中，通过 `$编号` 区分不同的关联图表，例如 `$0` 表示编号为0的图表。
高级选项	
触发通知阈值	累计触发次数达到该阈值时根据**通知间隔**发送告警。不满足触发条件时不计入统计。 默认**触发通知阈值**为1，即满足一次**触发条件**即可检查**通知间隔**。 通过配置触发通知阈值可以实现多次触发、一次通知。例如，配置触发通知阈值为100，则累计触发次数达到100次时检查**通知间隔**。如果同时满足**触发通知阈值**和**通知间隔**，则发送通知。发送通知之后，累计次数会清零。如果因网络异常等原因执行检查失败，不计入累计次数。
通知间隔	两次告警通知之间的时间间隔。 如果某次执行满足了触发条件，而且累计的触发次数已经达到**触发通知阈值**，且距离上次发送通知已经达到了通知间隔，则发送通知。如设置通知间隔为5分钟，则5分钟内至多收到一次通知。默认无间隔。 **说明** 通过配置触发通知阈值和通知间隔可以实现告警抑制的功能，防止收到过多的告警信息。

7. 设置通知方式。
   以指定形式发送到自定义WebHook地址中。需要指定**请求地址**、**请求方法**和**请求内容**。 通知方式的详细说明及示例请参见通知方式。
   - **请求地址**为自定义WebHook地址，例如：https://webhook.com/notify。
   - **请求方法**可以设置为GET、PUT、POST、DELETE、和OPTIONS。
   - **请求内容**为通知的内容，支持使用模板变量，长度为1~500个字符。
8. 单击**提交**。

5.2.2. 为RAM用户设置告警

可以通过授权，为RAM用户开通告警相关功能。

背景信息

仅为RAM用户授予创建及修改告警的权限，不授予其他日志服务管理权限。创建自定义权限策略，并为RAM用户授予该自定义权限策略。详细步骤请参考本文档。

操作步骤

1.
2. 创建RAM角色。
3. 新增权限策略。
   请使用如下策略内容，并替换其中的 `<Project名称>` 。

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "log:CreateLogStore",
        "log:CreateIndex",
        "log:UpdateIndex"
      ],
      "Resource": "acs:log:*:*:project/<Project名称>/logstore/internal-alert-history"
    },
    {
      "Effect": "Allow",
      "Action": [
        "log:CreateDashboard",
        "log:CreateChart",
        "log:UpdateDashboard"
      ],
      "Resource": "acs:log:*:*:project/<Project名称>/dashboard/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "log:*"
      ],
      "Resource": "acs:log:*:*:project/<Project名称>/job/*"
    }
  ]
}
```

4. 创建用户。
5. 新建用户组。
6. 添加用户到用户组
7. 为RAM角色授权。

5.2.3. 通知方式

本文档为您介绍日志服务告警的通知方式。

WebHook-自定义

告警的通知方式可设置为WebHook，当触发告警时，告警通知会以指定方式发送到自定义WebHook地址中。

> **说明** 告警通知方式为**WebHook-自定义**时，超时时间为5秒。如果发出请求后5秒内没有返回，则视作发送失败。

1. 在日志服务控制台设置告警。通知类型设置为**WebHook-自定义**。
2. **请求地址**中填写自定义的WebHook地址，并指定**请求方法**。
3. **可选：**单击**添加请求头**可以追加请求头（Header）信息。
   默认包含Header `Content-Type: application/json;charset=utf-8`，您也可以追加Header。
4. 填写**请求内容**。
   发生告警后会以指定方式将告警内容发到自定义WebHook地址。
5. 单击**提交**。

模板变量

配置通知方式时，必须设置**发送内容**，即通知的内容，通知内容中支持通过${fieldName}的方式引用一些告警触发时的模板变量。日志服务发送告警时，会将**发送内容**中的模板变量替换为真实值，如${Project}替换为告警规则所属的Project名称。

> **说明** 引用变量时变量名称必须完全匹配，对于不存在的变量或者不合法的引用会渲染为空字符串。如果引用的值为对象类型，则会转换为JSON字符串展示。

以下是目前支持的所有可用变量及引用方式。

变量	说明	举例	引用举例
Aliuid	Project所属的用户AliUid。	1234567890	用户${Aliuid}的告警规则已经触发。
Project	告警规则所属Project。	my-project	项目${Project}中的告警触发。
AlertID	执行的唯一ID。	0fdd88063a611aa114938f9371daeeb6-1671a52eb23	告警执行ID是${AlertID}。
AlertName	告警规则名称，Project内唯一。	alert-1542111415-153472	告警规则${AlertName} 已经触发。
AlertDisplayName	告警规则显示名称。	我的告警规则	告警名称${AlertDisplayName}已经触发。
Condition	触发告警时的条件表达式。其中，以触发告警的值替换设置的变量，并使用大括号包裹。	[5] > 1	告警条件表达式为${Condition}。
RawCondition	原始的条件表达式，即condition中不替换变量的原始表达式。	count > 1	原始条件表达式为${RawCondition}。
Dashboard	告警关联的仪表盘名称。	mydashboard	告警关联的仪表盘名称${Dashboard}。
DashboardUrl	告警关联的仪表盘地址。	https://sls.console.aliyun.com/next/project/myproject/dashboard/mydashboard	告警关联的仪表盘地址${DashboardUrl}。
FireTime	触发时间。	2018-01-02 15:04:05	告警触发时间${FireTime}。
FullResultUrl	告警触发历史记录的查询地址URL。	https://sls.console.aliyun.com/next/project/my-project/logsearch/internal-alert-history?endTime=1544083998&queryString=AlertID%3A9155ea1ec10167985519fccede4d5fc7-1678293caad&queryTimeType=99&startTime=1544083968	单击查看告警详情${FullResultUrl}

Results	查询参数和结果，数组类型。字段解释请参见告警日志字段。
	说明 最多发送100条告警信息。
		<pre>[
		  {
		    "EndTime":
		1542507580,
		    "FireResult": {
		      "__time__":
		"1542453580",
		      "count": "0"
		    },
		    "LogStore": "test-
		logstore",
		    "Query": "* |
		SELECT COUNT(*) as
		count",
		    "RawResultCount":
		1,
		    "RawResults": [
		      {
		        "__time__":
		"1542453580",
		        "count": "0"
		      }
		    ],
		    "StartTime":
		1542453580
		  }
		]</pre>
			第一个查询的开始时间为${Results[0].StartTime}；结束时间为${Results[0].EndTime}；count的值为${Results[0].FireResult.count}。
			说明 其中0为图表查询/分析语句的编号。 关于图表编号请参见如何查看图表编号？。

5.3. 修改与查看告警

5.3.1. 修改告警规则

创建告警后，您可以修改告警图表后更新告警规则；基于查询语句的告警，可以直接在告警中修改查询语句。

注意事项

- 只有为查询语句设置的告警规则支持修改查询语句，且只能修改为查询语句，不支持修改为查询分析语句（ `查询语句|分析语句` ）。

  例如，为查询语句 `request_method: GET` 绑定告警规则后，可以将查询语句修改为 `error` ，但不能修改为 `error| select count(1) as c` 。
- 告警规则可以在**告警配置**页面中单击**修改**，或者在配置了告警的仪表盘页面右上角单击**告警 > 修改**。

修改告警绑定的查询语句

在日志服务查询页面执行的查询语句如果被绑定了告警规则，绑定后可以修改查询语句。

1. 登录日志服务控制台。
2. 单击目标Project名称。
3. 单击左侧导航栏的**仪表盘**图标。
4. 在仪表盘列表中单击指定仪表盘名称。
5. 在页面右上角单击**告警 > 修改**。
6. 找到需要修改的查询语句，单击其右侧的 。

   只有为查询语句设置的告警规则支持修改查询语句，且只能修改为查询语句，不支持修改为查询分析语句（ `查询语句|分析语句` ）。
7. 输入新的查询语句，并单击其右侧的**预览**按钮，通过校验后单击**确认**。
8. 根据需求确认是否修改**频率**和**触发条件**，并单击**下一步**。
9. 设置通知方式，并单击**提交**。

修改告警关联图表

创建告警规则后，可以随时修改告警规则。

1. 在仪表盘列表中单击指定仪表盘名称。
2. 在仪表盘页面右上角单击**告警 > 修改**。
3. 找到需要修改告警的**关联图表**，在**查询语句**右侧单击 。
4. 输入新的查询语句，并单击其右侧的**预览**按钮，通过校验后单击**确认**。
5. 确认**频率**和**触发条件**，并单击**下一步**。
6. 重新设置告警配置及通知方式。
7. 单击**提交**使修改生效。

5.3.2. 查看告警记录

日志服务以告警日志方式提供告警历史记录信息，并自动创建仪表盘以可视化展示所有告警规则的执行与通知情况。

在Logstore中查看告警日志

创建告警规则时，日志服务自动为告警所属的Project创建一个Logstore **internal-alert-history**。Project内所有告警规则的每一次执行无论是否触发告警，都会产生一条日志并写入到这个Logstore中，日志字段内容请参见告警日志字段。

> **说明** 该Logstore不会产生任何费用，不支持删除和修改。日志保存时间为7天。

1. 登录日志服务控制台。
2. 单击目标Project名称。
3. 单击**internal-alert-history** Logstore后的 图标，选择**查询分析**。
4. 根据需求查询告警日志信息。

查看告警记录仪表盘

创建告警规则之后，日志服务默认会在该告警规则所属的Project创建一个仪表盘**internal-alert-analysis**用于展示告警记录。告警记录仪表盘中记录了当前Project中所有告警动作的信息，如告警次数、执行成功率、执行成功时通知率、告警规则执行次数Top10等信息。

> **说明** 不支持删除或修改该仪表盘。

1. 单击左侧导航栏的**仪表盘**图标。
2. 单击**告警历史统计**，进入**告警历史统计**仪表盘页面。

告警历史统计仪表盘中详细展示了告警历史，包括报警是否被触发、触发状态的原因、错误信息及说明等信息。

5.3.3. 管理告警配置

配置告警后，可以在告警概览页面查看告警规则详情与状态等信息。

背景信息

告警概览页面支持关闭与启用告警、暂停与恢复告警、修改与删除告警、查看告警规则更新时间等操作。

查看告警配置信息

1. 登录日志服务控制台。
2. 单击目标Project名称。
3. 单击左侧导航栏的**告警列表**图标。
4. 在告警列表中，单击指定的告警规则名称。
   告警概览页面中展示了所属仪表盘名称、创建时间、上次更新时间、检查频率、启用状态、通知状态等信息。

关闭与启用告警

创建告警后可以随时关闭或启用告警。告警关闭后不会定期执行告警检查、发送通知。

1. 单击左侧导航栏的**告警列表**图标。
2. 在告警列表中，单击指定的告警规则名称。
   在**告警概览**页面中，单击**启用状态**后的**开启**或**关闭**按钮。

暂停与恢复告警通知

开启状态的告警可以设置暂停告警通知，在指定的时段内会定期执行告警检查，但即使满足预设条件也不会发送告警通知。

1. 单击左侧导航栏的**告警列表**图标。
2. 在告警列表中，单击指定的告警规则名称。
   在**告警概览**页面中，单击**通知状态**后的**设置**按钮。

3. 指定关闭通知的时长，并单击**确认**。
   暂停告警通知后，可以在**通知状态**列查看告警通知的恢复时间。单击**通知状态**后的**设置**按钮，可以在自动恢复告警通知之前，手动恢复告警通知。

删除告警

告警删除后不可恢复，请谨慎操作。

1. 单击左侧导航栏的**告警**图标。

2. 在告警列表中单击指定告警规则名称。
3. 在**告警概览**页面中，单击右上方的**删除告警**。
4. 在弹出对话框中单击**确认**。

5.4. 参考信息

5.4.1. 告警条件表达式语法

告警支持用户配置条件表达式，根据表达式的结果是否为真来判断是否满足告警条件。

在判断表达式是否为真时，用户配置的查询的执行结果将作为输入，日志字段作为变量，一旦条件为真则触发告警并返回。

限制说明

- 负数需要使用括号，如 x+(-100)<100。
- 数值类型都被当成64位浮点数，如果使用比较操作如等于可能存在误差。
- 变量只能包含字母和数字，且首字母必须是字母。
- 表达式长度最多支持128个字符。
- 组合求值时最多只会计算1000种组合，如果没有找到结果为真的组合，则视为false。
- 最多只支持三个查询。
- 当且仅当表达式的值为布尔值true的时候，才会触发告警。如 100+100，计算结果为200，不会触发告警。
- `true`、`false`、`$`和`.`是保留字，不能作为变量使用。

基础语法

告警条件表达式支持以下语法类型。

语法类型	说明	示例
基础运算符	支持加减乘除、取模运算符，即： +-*/%。	x*100+y>200 x%10>5
比较运算符	支持大于（>）、大于等于（>=）、小于（<）、小于等于（<=）、等于（==）、不等于（!=）、正则匹配（=~）、正则不匹配（!~）8种比较运算符。 **说明** • 斜杠需要转义。 • 正则表达式目前支持符合RE2规范的语法。	x >= 0 x < 100 x <= 100 x == 100 x == "foo" 正则匹配：x =~ "\\w+"
逻辑操作符	支持逻辑操作符：与（&&）、或（||）。	x >=0&&y <=100 x > 0 || y > 0
取反前缀操作	支持取反前缀操作（!）。	!(a < 1 && a > 100)
数值常量	支持数值常量，作为64位浮点数处理。	x > 100
字符串常量	支持字符串常量，形式为单引号引起来的字符串。	foo == 'string'
布尔常量	支持布尔常量：true和false。	(x > 100) == true
括号	支持使用括号改变计算的优先级。	x*(y+100)>100
contains函数	支持使用contains函数判断是否包含子串， 如 contains(field, 'xxxx') 返回true则表示 field 包含 xxxx 这个子串。	contains(foo, 'hello')

多个结果集组合求值

- 语法
  告警支持用户关联多个图表的查询，在使用多个查询结果进行计算时，变量需要加上特定前缀以区分从哪个结果集中获取对应的变量值，格式为 `$N.fieldname` ，其中N为查询的编号。目前支持用户最多配置三个查询，因此N的取值范围为[0,2]。如$0.foo表示第1个查询的foo字段。当仅有一个查询时，前缀可以省略。

  > **说明** 如何查看图表编号？
  >
  > 在告警配置步骤中，**关联图表**一栏显示了各个图表/查询分析语句的编号。其中第一个图表/查询分析语句编号为0，第二个图表/查询分析语句编号为1，第三个图表/查询分析语句编号为2。

- 表达式求值

  在多个查询结果返回时，根据表达式的变量来判断需要使用哪些结果集求值。例如用户配置了三个查询，每个查询分别返回了x,y,z 条结果。而用户配置的表达式为$0.foo > 100 && $1.bar < 100，则说明判断表达式的值只需要使用前两个结果集，进行**x*y**次求值直到某次求值返回true，或者达到计算次数上限后直接返回false，目前上限为1000次。

运算方式

> **说明**
> - number为64位浮点数类型。
> - string常量需要以单引号或英文双引号包含起来，如'string'、“string”。
> - 布尔值包括true和false。

运算符	运算方式		
	变量与变量运算	非string常量与变量运算	string常量与变量运算
四则运算（+-*/%）	左右值转number后运算。		不支持。
比较运算： 大于（>）、大于等于（>=）、小于（<）、小于等于（<=）、等于（==）、不等于（!=）	按照以下优先级决定运算顺序： 1. 左右值转number后按照数值序运算，如转换失败则执行下一优先级的运算。 2. 左右值按string类型字典序运算。	左右值转number后运算（数值序）。	左右值按string类型运算（字典序）。
正则是否匹配： 正则匹配（=~）、正则不匹配（!~）	左右值按string类型运算。	不支持。	左右值按string类型运算。
逻辑运算： 与（&&）、或（||）	不支持对查询结果字段直接应用该运算符，左右值必须分别为子运算式，且运算结果为bool类型。		
取反前缀（!）	不支持对查询结果字段直接应用该运算符，被取反的值必须为子运算式，且运算结果为bool类型。		
字符串查找（contains）	左右值转string类型运算。	不支持。	左右值按string类型运算。
括号()	决定运算结合顺序与优先级。		

5.4.2. 告警日志字段

设置告警规则后，日志服务自动创建Logstore，以日志方式记录告警的执行与通知信息。本文档介绍告警日志的字段。

告警执行历史日志字段

字段名称	说明	示例
AlertDisplayName	告警规则显示名称。	告警规则测试
AlertID	每次执行的唯一ID。	0fdd88063a611aa114938f9371daeeb6-1671a52eb23
AlertName	每个Project内部唯一的告警规则名称。	alert-1542111415-153472
Condition	条件表达式。	$0.count > 1
Dashboard	告警规则关联的仪表盘。	my-dashboard
FireCount	上次通知之后的累积触发次数。	1
Fired	是否触发告警，取值为true或者false。	true
LastNotifiedAt	上次通知时间，Unix时间戳。	1542164541
NotifyStatus	通知状态，可能的值为： • Success：成功。 • Failed：失败。 • NotNotified：未通知。 • PartialSuccess：部分成功。	Success
Reason	失败或者未通知的原因。	result type is not bool

Results	查询参数和结果，数组类型，字段说明请参见Result字段说明。	[
		{
		"EndTime": 1542334900,
		"FireResult": null,
		"LogStore": "test-logstore",
		"Query": "* | select count(1) as count",
		"RawResultCount": 1,
		"RawResults": [
		{
		"__time__": "1542334840",
		"count": "0"
		}
		],
		"StartTime": 1542334840
		}
		]

Status	执行结果，取值为Success或者Failed。	Success

Result字段说明

字段名称	说明	示例
Query	查询语句。	* | select count(1) as count
LogStore	查询的目标Logstore。	my-logstore
StartTime	查询开始时间。	2019-01-02 15:04:05
StartTimeTs	查询开始时间，Unix时间戳。	1542334840
EndTime	查询结束时间。	2019-01-02 15:19:05
EndTimeTs	查询结束时间，Unix时间戳。注意，实际查询区间为 `[StartTime, EndTime)` 。	1542334900

RawResults	查询原始结果，数组类型，每个元素为一条日志。数组长度和日志内容大小有关，最多包含100条。	<pre>[
		  {
		  "__time__": "1542334840",
		  "count": "0"
		  }
		]</pre>

RawResultsAsKv	按照key-value格式化的原始查询结果。 **说明** 该字段只可以作为模版变量引用，不会保存到Logstore。	[foo:0]
RawResultCount	原始结果条数。	1

FireResult	触发告警的日志。如果告警未触发则为null。	<pre>{
		  "__time__": "1542334840",
		  "count": "0"
		}</pre>

FireResultAsKv	按照key-value格式化的触发告警的日志。 **说明** 该字段只可以作为模版变量引用，不会保存到Logstore。	[foo:0]

6.实时消费

6.1. 简介

日志服务支持多种方式的日志消费。

数据收集至日志服务 LogHub 后，有二种方法可以处理日志：

方式	场景	实时性	存储时间
实时消费（LogHub）	实时计算等	实时	自定义
索引查询（LogSearch）	适合最近热数据的在线查询	实时	自定义

实时消费

日志服务LogHub功能提供Pull接口，支持日志数据实时消费。对于一个 Shard 中的日志，消费过程如下：

1. 根据开始时间（Begin）和结束时间 （End）等条件获得游标等条件获得游标。
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。

- SDK消费

  日志服务提供多语言（Java 、Python、Go 等） SDK，且这些语言的 SDK 都支持日志消费接口。

- 消费组消费

  消费组是日志服务对 LogHub 消费者提供的高级模式。消费组提供了一个轻量级计算框架，解决多个消费者并行消费 Logstore 的能力，消费组提供自动分配 Shard、支持保序、断点续传等功能。目前 Go、Python、Java 等语言均提供消费组 SDK。

- 实时计算系统消费
  - Spark Streaming Client。
  - Storm Spout。
  - Flink Connector。

- 开源产品消费

  Flume消费：使用 Flume 消费日志，并将日志导入到 HDFS 实例。

查询分析

- 使用日志服务控制台查询日志。
- 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。

6.2. 普通消费

日志服务提供多语言（Java 、Python、Go等） 的 SDK，且这些语言的SDK都支持日志消费接口。

SDK消费

以Java SDK为例，以下代码展示如何消费 ShardId 中的数据。

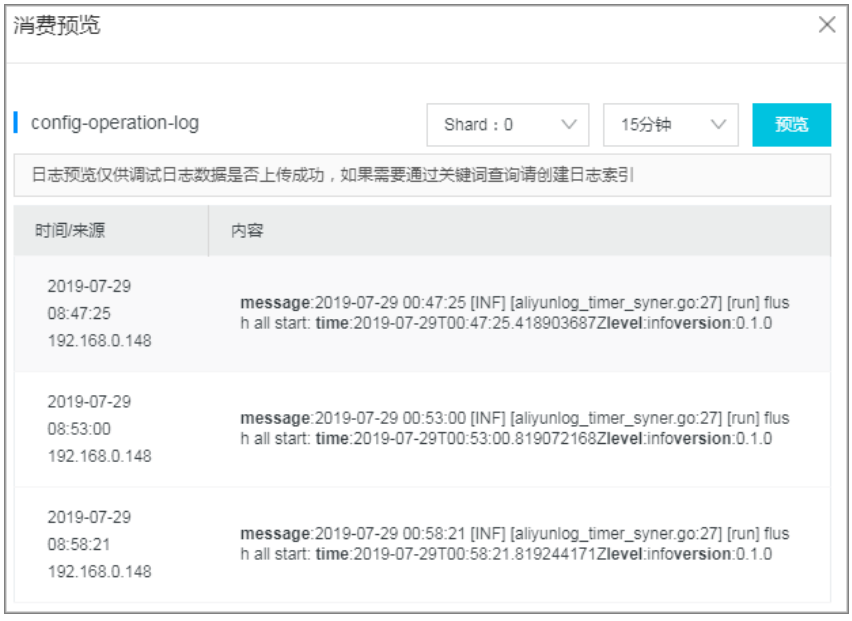
```
Client client = new Client(host, accessId, accessKey);

    String cursor = client.GetCursor(project, logStore, shardId, CursorMode.END).GetCursor();
    System.out.println("cursor = " +cursor);
    try {
      while (true) {
        PullLogsRequest request = new PullLogsRequest(project, logStore, shardId, 1000, cursor);
        PullLogsResponse response = client.pullLogs(request);
        System.out.println(response.getCount());
        System.out.println("cursor = " + cursor + " next_cursor = " + response.getNextCursor());
        if (cursor.equals(response.getNextCursor())) {
            break;
                }
        cursor = response.getNextCursor();
        Thread.sleep(200);
      }
    }
    catch(LogException e) {
      System.out.println(e.GetRequestId() + e.GetErrorMessage());
    }
```

控制台预览

日志预览也是一种日志消费。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的部分日志。

1. 登录日志服务控制台。
2. 单击目标Project。
3. 在日志库列表中，单击日志库名称后的图标，然后选择**消费预览**。
4. 在消费预览页面，指定预览的Shard，时间段，然后单击**预览**。
   预览页面向您展示指定时间区间开始的10个数据包的日志数据。



6.3. 消费组消费

6.3.1. 通过消费组消费日志

与使用SDK消费数据相比，使用消费组（ConsumerGroup）消费日志数据的优点在于，用户无需关心日志服务的实现细节和消费者之间的负载均衡、failover等，只需要专注于业务逻辑即可。

基本概念

使用消费组消费日志数据之前，请阅读以下日志服务消费组基本概念，包括消费组（ConsumerGroup）和消费者（Consumer）。

概念	说明
消费组	一个消费组由多个消费者构成，同一个消费组下面的消费者共同消费一个Logstore中的数据，消费者之间不会重复消费数据。
消费者	消费组的构成单元，实际承担消费任务，同一个消费组下面的消费者名称必须不同。

在日志服务中，一个Logstore下面会有多个Shard，协同消费库的功能就是将Shard分配给一个消费组下面的消费者，分配方式遵循以下原则：

- 每个Shard只会分配到一个消费者。
- 一个消费者可以同时拥有多个Shard。

新的消费者加入一个消费组，这个消费组下面的Shard从属关系会调整，以达到消费负载均衡的目的，但是仍遵循分配原则，分配过程对用户透明。

协同消费库的另一个功能是保存Checkpoint，方便程序故障恢复时能接着从断点继续消费，从而保证数据不会被重复消费。

操作步骤

日志服务消费组通过JAVA语言和Python语言实现，本文档以JAVA为例。

1. 添加maven依赖。

```
<dependency>
  <groupId>com.google.protobuf</groupId>
  <artifactId>protobuf-java</artifactId>
  <version>2.5.0</version>
</dependency>
<dependency>
  <groupId>com.aliyun.openservices</groupId>
  <artifactId>loghub-client-lib</artifactId>
  <version>0.6.16</version>
</dependency>
```

2. 创建main .java文件。

```
import com.aliyun.openservices.loghub.client.ClientWorker;
import com.aliyun.openservices.loghub.client.config.LogHubConfig;
import com.aliyun.openservices.loghub.client.exceptions.LogHubClientWorkerException;

public class Main {
    // 日志服务域名，根据实际情况填写。
    private static String sEndpoint = "cn-hangzhou.log.aliyuncs.com";
    // 日志服务项目名称，根据实际情况填写。
    private static String sProject = "ali-cn-hangzhou-sls-admin";
    // 日志库名称，根据实际情况填写。
    private static String sLogstore = "sls_operation_log";
    // 消费组名称，根据实际情况填写。
    private static String sConsumerGroup = "consumerGroupX";
    // 消费数据的ak，根据实际情况填写。
    private static String sAccessKeyId = "";
    private static String sAccessKey = "";

    public static void main(String[] args) throws LogHubClientWorkerException, InterruptedException {
        // 第二个参数是消费者名称，同一个消费组下面的消费者名称必须不同，可以使用相同的消费组名称，不同的消费者名称在多台机器上启动多个进程，来均衡消费一个Logstore，这个时候消费者名称可以使用机器ip来区分。第9个参数（maxFetchLogGroupSize）是每次从服务端获取的最大LogGroup数目，使用默认值即可，如有调整请注意取值范围(0,1000]。
        LogHubConfig config = new LogHubConfig(sConsumerGroup, "consumer_1", sEndpoint, sProject, sLogstore, sAccessKeyId, sAccessKey, LogHubConfig.ConsumePosition.BEGIN_CURSOR);
        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sleep 30s。
        Thread.sleep(30 * 1000);
    }
}
```

3. 创建SampleLogHubProcessor.java文件

```
import com.aliyun.openservices.log.common.FastLog;
import com.aliyun.openservices.log.common.FastLogContent;
import com.aliyun.openservices.log.common.FastLogGroup;
import com.aliyun.openservices.log.common.FastLogTag;
import com.aliyun.openservices.log.common.LogGroupData;
import com.aliyun.openservices.loghub.client.ILogHubCheckPointTracker;
import com.aliyun.openservices.loghub.client.exceptions.LogHubCheckPointException;
import com.aliyun.openservices.loghub.client.interfaces.ILogHubProcessor;
import com.aliyun.openservices.loghub.client.interfaces.ILogHubProcessorFactory;

import java.util.List;

public class SampleLogHubProcessor implements ILogHubProcessor {
    private int shardId;
    // 记录上次持久化 checkpoint 的时间。
    private long mLastCheckTime = 0;

    public void initialize(int shardId) {
        this.shardId = shardId;
    }

    // 消费数据的主逻辑，这里面的所有异常都需要捕获，不能抛出去。
    public String process(List<LogGroupData> logGroups,
                          ILogHubCheckPointTracker checkPointTracker) {
        // 这里简单的将获取到的数据打印出来。
        for (LogGroupData logGroup : logGroups) {
            FastLogGroup flg = logGroup.GetFastLogGroup();
```

```
System.out.println(String.format("\tcategory\t:\t%s\n\tsource\t:\t%s\n\ttopic\t:\t%s\n\tmachineUUID\t:\t
,
                    flg.getCategory(), flg.getSource(), flg.getTopic(), flg.getMachineUUID()));
            System.out.println("Tags");
            for (int tagIdx = 0; tagIdx < flg.getLogTagsCount(); ++tagIdx) {
                FastLogTag logtag = flg.getLogTags(tagIdx);
                System.out.println(String.format("\t%s\t:\t%s", logtag.getKey(), logtag.getValue()));
            }
            for (int lIdx = 0; lIdx < flg.getLogsCount(); ++lIdx) {
                FastLog log = flg.getLogs(lIdx);
                System.out.println("--------\nLog: " + lIdx + ", time: " + log.getTime() + ", GetConten
tCount: " + log.getContentsCount());
                for (int cIdx = 0; cIdx < log.getContentsCount(); ++cIdx) {
                    FastLogContent content = log.getContents(cIdx);
                    System.out.println(content.getKey() + "\t:\t" + content.getValue());
                }
            }
        }
        long curTime = System.currentTimeMillis();
        // 每隔 30 秒，写一次 checkpoint 到服务端，如果 30 秒内，worker crash，
        // 新启动的 worker 会从上一个 checkpoint 取消费数据，有可能有少量的重复数据。
        if (curTime - mLastCheckTime > 30 * 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，后台默认隔60s会将ch
eckpoint刷新到服务端。
                checkPointTracker.saveCheckPoint(true);
            } catch (LogHubCheckPointException e) {
                e.printStackTrace();
            }
            mLastCheckTime = curTime;
        }
        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();
        }
    }
}

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

> **说明** 运行以上代码，可以将一个Logstore下的所有数据打印出来，如果需要多个消费者共同消费一个Logstore，可以按程序注释修改程序，用同样的消费组名称加不同的消费者名称，启动另外的消费进程。

限制与异常诊断

每个Logstore创建消费组个数的上限为10。超出时将报错 `ConsumerGroupQuotaExceed` 。

建议为消费者程序配置log4j，将消费组内部遇到的错误信息打印出来，方便定位异常。例如，放置log4j.properties文件到resources目录下执行程序可以看到类似如下异常：

```
[WARN ] 2018-03-14 12:01:52,747
method:com.aliyun.openservices.loghub.client.LogHubConsumer.sampleLogError(LogHubConsumer.java:159)
com.aliyun.openservices.log.exception.LogException: Invalid loggroup count, (0,1000]
```

以下配置为log4j.properties典型配置：

```
log4j.rootLogger = info,stdout
log4j.appender.stdout = org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target = System.out
log4j.appender.stdout.layout = org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern = [%-5p] %d{yyyy-MM-dd HH:mm:ss,SSS} method:%l%n%m%n
```

高阶操作

普通用户使用上述程序即可消费数据，高阶用户还可以参考以下内容完成高阶操作。

- 消费某个时间开始的数据
  上面代码中的LoghubConfig有两个构造函数：

  ```
  // consumerStartTimeInSeconds参数表示1970之后的秒数，含义是消费这个时间点之后的数据。
  public LogHubConfig(String consumerGroupName,
                      String consumerName,
                      String loghubEndPoint,
                      String project, String logStore,
                      String accessId, String accessKey,
                      int consumerStartTimeInSeconds);
  // position是个枚举变量，LogHubConfig.ConsumePosition.BEGIN_CURSOR表示从最老的数据开始消
  费，LogHubConfig.ConsumePosition.END_CURSOR表示从最新的数据开始消费。
  public LogHubConfig(String consumerGroupName,
                      String consumerName,
                      String loghubEndPoint,
                      String project, String logStore,
                      String accessId, String accessKey,
                      ConsumePosition position);
  ```

  可以按照消费需求，使用不同的构造方法，但是注意，如果服务端保存有checkpoint，那么开始消费位置以服务端保存的checkpoint为准。

- 重置Checkpoint

  在补数据或重复计算等场景中，需要将某个ConsumerGroup点位设置为某一个时间点，使当前消费组能够从新位置开始消费。可通过以下两种方式完成。

  - 删除消费组。
    a. 停止消费程序。
    b. 在控制台删除消费组。
    c. 修改代码，使用指定时间点消费。
    d. 重新启动消费程序。
  - 通过SDK将当前消费组重置到某一个时间点。
    a. 停止消费程序。
    b. 使用SDK修改位点。
    c. 重新启动消费程序。

  ```
  public static void updateCheckpoint() throws Exception {
          Client client = new Client(host, accessId, accessKey);
          long timestamp = Timestamp.valueOf("2017-11-15 00:00:00").getTime() / 1000;
          ListShardResponse response = client.ListShard(new ListShardRequest(project, logStore));
          for (Shard shard : response.GetShards()) {
              int shardId = shard.GetShardId();
              String cursor = client.GetCursor(project, logStore, shardId, timestamp).GetCursor();
              client.UpdateCheckPoint(project, logStore, consumerGroup, shardId, cursor);
          }
      }
  ```

使用RAM子账号进行访问

子账号访问需要设置消费组相关的RAM权限，设置方法请参见为RAM角色授权。

可以设置的相关Action如下：

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:UpdateConsumerGroup	acs:log:*${regionName}*:*${projectOwnerAliUid}*:project/*${projectName}*/logstore/*${logstoreName}*/consumergroup/*${consumerGroupName}*
log:GetConsumerGroupCheckPoint	acs:log:*${regionName}*:*${projectOwnerAliUid}*:project/*${projectName}*/logstore/*${logstoreName}*/consumergroup/*${consumerGroupName}*

例如，Project所在地域为cn-hangzhou，Project名称为project-test，消费的Logstore名称为logstore-test，Project的所属阿里云账号ID为1234567，消费组名称为consumergroup-test，则需要为RAM子账号设置以下权限。

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "log:GetCursorOrData"
      ],
      "Resource": "acs:log:cn-hangzhou:1234567:project/project-test/logstore/logstore-test"
    },
    {
      "Effect": "Allow",
      "Action": [
        "log:CreateConsumerGroup",
        "log:ListConsumerGroup"
      ],
      "Resource": "acs:log:cn-hangzhou:1234567:project/project-test/logstore/logstore-
test/consumergroup/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "log:ConsumerGroupUpdateCheckPoint",
        "log:ConsumerGroupHeartBeat",
        "log:UpdateConsumerGroup",
        "log:GetConsumerGroupCheckPoint"
      ],
      "Resource": "acs:log:cn-hangzhou:1234567:project/project-test/logstore/logstore-
test/consumergroup/consumergroup-test"
    }
  ]
}
```

6.3.2. 消费组状态

协同消费组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

1. 登录日志服务控制台。
2. 单击目标Project。
3. 找到目标日志库，单击 > 数据消费。
4. 单击指定的ConsumerGroup之后，即可查看每个shard消费数据的进度。

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.getCheckP
oint()).GetCursorTime();
                        consumerPrg = "" + prg;
                    }
                    catch(LogException e){
                        if(e.GetErrorCode() == "InvalidCursor")
```

```
                        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);
            }
        }
    }
}
```

6.4. Storm消费

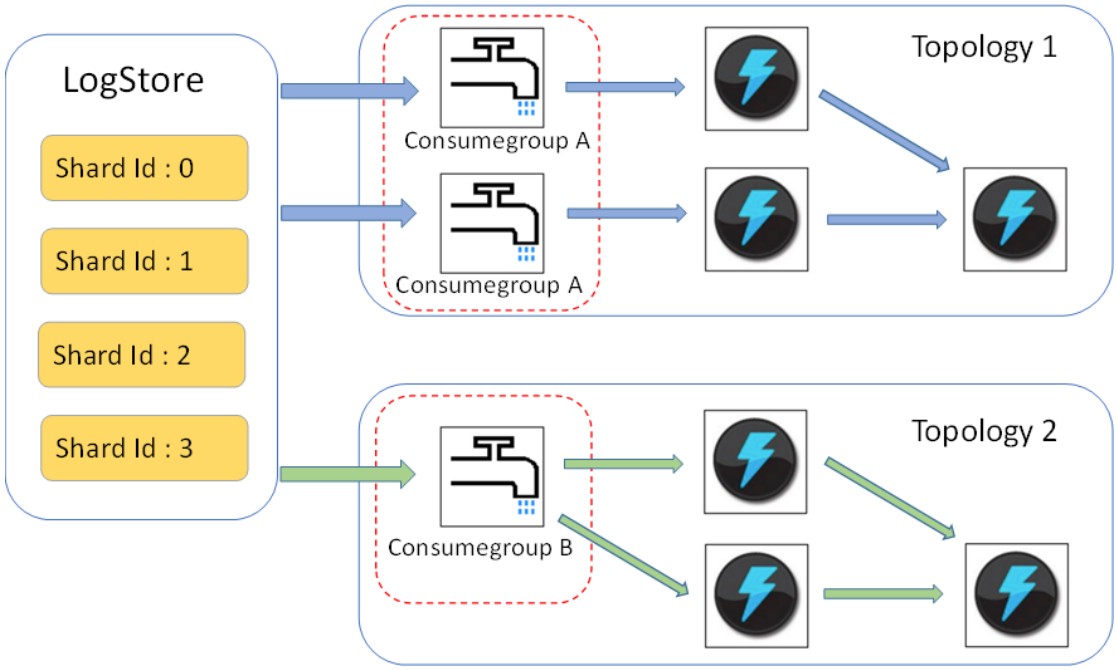
日志服务的LogHub提供了高效、可靠的日志通道功能，您可以通过Logtail、SDK等多种方式来实时收集日志数据。收集日志之后，可以通过Spark Stream、Storm 等各实时系统来消费写入到LogHub中的数据。

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

- 下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通过消费组（ConsumerGroup）来完成负载均衡和自动failover，消费组详情请参见通过消费组消费日志。
- 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。

图 1. 基本结构和流程



使用限制

- 为了防止滥用，每个Logstore最多支持 10 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);
        }
    }
}
```

- 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>0.6.6</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>0.1.3</version>
</dependency>
```

6.5. Flume消费

日志服务LogHub支持通过aliyun-log-flume插件与Flume进行对接，实现日志数据的写入和消费。

aliyun-log-flume是一个实现日志服务（Loghub）与Flume对接的插件，与Flume对接后，日志服务可以通过Flume与其它数据系统如HDFS，Kafka等对接。aliyun-log-flume提供Sink和Source方式实现日志服务与Flume的对接。

- Sink：Flume读取其他数据源的数据然后写入Loghub。
- Source：Flume消费Loghub的数据然后写入其他系统。

Loghub Sink

通过Sink的方式将其他数据源的数据通过Flume接入Loghub。目前支持两种解析格式：

- SIMPLE：将整个Flume Event作为一个字段写入Loghub。
- DELIMITED：将整个Flume Event作为分隔符分隔的数据，根据配置的列名解析成对应的字段写入Loghub。

Sink方式支持的配置如下：

配置项	是否必选	说明
type	必选	固定为com.aliyun.loghub.flume.sink.LoghubSink。
endpoint	必选	服务入口。
project	必选	Project名称。
logstore	必选	Logstore名称。
accessKeyId	必选	阿里云访问密钥AccessKeyId。
accessKey	必选	阿里云访问密钥AccessKey。
batchSize	可选	每批次写入Loghub的数据条数。默认1000条。
maxBufferSize	可选	缓存队列的大小。默认1000条。
serializer	可选	Event序列化格式。支持的模式如下： • **DELIMITED**：设置解析格式为分隔符模式。当设置为该模式时**columns**参数为必选。 • **SIMPLE**：设置解析格式为单行模式。默认为该模式。 • 自定义**serializer**：设置解析格式为自定义的序列化模式，设置为该模式时需要填写完整类名称。
columns	可选	**serializer**为**DELIMITED**时，必须指定该字段列表，用逗号（,）分隔，顺序与实际数据中的字段顺序一致。
separatorChar	可选	**serializer**为**DELIMITED**时，用于指定数据的分隔符，必须为单个字符。默认为逗号（,）。
quoteChar	可选	**serializer**为**DELIMITED**时，用于指定Quote字符。默认为双引号（“”）。
escapeChar	可选	**serializer**为**DELIMITED**时，用于指定转义字符。默认为双引号（“”）。
useRecordTime	可选	是否使用数据中的**timestamp**字段作为日志时间。默认为false表示使用当前时间。

Loghub Source

通过Source的方式将Loghub的数据通过Flume投递到其他的数据源。目前支持两种输出格式。

- DELIMITED：数据以分隔符日志的形式写入Flume。
- JSON：数据以JSON日志的形式写入Flume。

Source方式支持的配置如下：

配置项	是否必选	说明
type	必选	固定为com.aliyun.loghub.flume.source.LoghubSource。
endpoint	必选	服务入口。
project	必选	Project名称。

logstore	必选	Logstore名称。
accessKeyId	必选	阿里云访问密钥accessKeyId。
accessKey	必选	阿里云访问密钥accessKey。
heartbeatIntervalMs	可选	客户端和Loghub的心跳间隔，默认30000，单位毫秒。
fetchIntervalMs	可选	Loghub数据拉取间隔，默认100，单位毫秒。
fetchInOrder	可选	是否按顺序消费。默认为false。
batchSize	可选	每批次读取的数据条数，默认为100。
consumerGroup	可选	读取的消费组名称（随机产生）。
initialPosition	可选	读取数据的起点位置，支持begin、end、timestamp。默认为begin。 **说明** 如果服务端已经存在checkpoint，会优先使用服务端的checkpoint。
timestamp	可选	当**initialPosition**为**timestamp**时，必须指定时间戳。Unix时间戳格式。
deserializer	必选	Event反序列化格式，支持的模式如下： • **DELIMITED**：设置解析格式为分隔符模式。默认为该模式。当设置为该模式时**columns**参数为必选。 • **JSON**：设置解析格式为JSON模式。 • 自定义**deserializer**：设置解析格式为自定义的反序列化模式，设置为该模式时需要填写完整类名称。
columns	可选	**deserializer**为**DELIMITED**时，必须指定字段列表，用逗号（,）分隔，顺序与实际数据中的字段顺序一致。
separatorChar	可选	**deserializer**为**DELIMITED**时，用于指定数据的分隔符，必须为单个字符。默认为逗号（,）。
quoteChar	可选	**deserializer**为**DELIMITED**时，用于指定Quote字符。默认为引号（“）。
escapeChar	可选	**deserializer**为**DELIMITED**时，用于指定转义字符。默认为引号（“）。
appendTimestamp	可选	**deserializer**为**DELIMITED**时，是否将时间戳作为一个字段自动添加到每行末尾。默认为false。
sourceAsField	可选	**deserializer**为**JSON**时，是否将日志Source作为一个字段，字段名称为__source__。默认为false。
tagAsField	可选	**deserializer**为**JSON**时，是否将日志Tag作为字段，字段名称为__tag__：{tag名称}。默认为false。
timeAsField	可选	**deserializer**为**JSON**时，是否将日志时间作为一个字段，字段名称为__time__。默认为false。
useRecordTime	可选	是否使用数据中的**timestamp**字段作为日志时间。默认为false表示使用当前时间。

6.6. 开源Flink消费

Flink Log Connector是阿里云日志服务提供的，用于对接Flink的工具。本文档主要介绍如何对接Flink工具对日志数据进行消费。

前提条件

- 已启用Access Key，并创建了Project和Logstore。
- 若您选择使用子账号操作日志服务，请确认已正确设置了Logstore的RAM授权策略。详细内容请参见为RAM角色授权。

背景信息

Flink Log Connector包括两部分，消费者（Consumer）和生产者（Producer）。

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

```
<dependency>
 <groupId>com.aliyun.openservices</groupId>
 <artifactId>flink-log-connector</artifactId>
 <version>0.1.13</version>
</dependency>
<dependency>
 <groupId>com.google.protobuf</groupId>
 <artifactId>protobuf-java</artifactId>
 <version>2.5.0</version>
</dependency>
```

Log Consumer

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

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

Flink Log Consumer 会用到的阿里云日志服务接口如下：

- GetCursorOrData

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

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

- ListShards

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

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

- CreateConsumerGroup

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

- ConsumerGroupUpdateCheckPoint

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

当RAM子账号使用Flink Log Consumer时，需要对以上几个API进行授权：

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

关于子账号及其授权操作请参见为RAM角色授权。

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

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

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

2. 设置消费起始位置。
   Flink Log Consumer支持设置Shard的消费起始位置，通过设置属性ConfigConstants.LOG_CONSUMER_BEGIN_POSITION，就可以定制消费从Shard的头尾或者某个特定时间开始消费，另外，Connector也支持从某个具体的ConsumerGroup中恢复消费。具体取值如下：
   - Consts.LOG_BEGIN_CURSOR： 表示从Shard的头开始消费，也就是从Shard中最旧的数据开始消费。
   - Consts.LOG_END_CURSOR： 表示从Shard的尾开始，也就是从Shard中最新的数据开始消费。
   - Consts.LOG_FROM_CHECKPOINT：表示从某个特定的ConsumerGroup中保存的Checkpoint开始消费，通过ConfigConstants.LOG_CONSUMERGROUP指定具体的ConsumerGroup。
   - UnixTimestamp： 一个整型数值的字符串，用1970-01-01到现在的秒数表示， 含义是消费Shard中这个时间点之后的数据。

   四种取值示例如下：

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

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

3. **可选：**设置消费进度监控。
   Flink Log Consumer支持设置消费进度监控，所谓消费进度就是获取每一个Shard实时的消费位置，这个位置使用时间戳表示，详细概念可以参考文档消费组状态。

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

   > **说明** 该设置为可选配置项，如果设置，Consumer会首先创建ConsumerGroup，如果ConsumerGroup已经存在，则不执行任何操作，Consumer中的snapshot会自动同步到日志服务的ConsumerGroup中，用户可以在日志服务的控制台查看Consumer的消费进度。

4. 设置容灾和exactly once语义支持。
   当打开Flink的Checkpointing功能时，Flink Log Consumer会周期性的将每个Shard的消费进度保存起来，当作业失败时，Flink会恢复Log Consumer，并从保存的最新的Checkpoint开始消费。
   写Checkpoint的周期定义了当发生失败时，最多有多少的数据会被回溯，即重新消费，使用代码如下：

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

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

Log Producer

Flink Log Producer 用于将数据写到阿里云日志服务中。

> **说明** Producer只支持Flink at-least-once语义，在发生作业失败的情况下，写入日志服务中的数据有可能会重复，但是不会丢失。

Producer会用到的日志服务API接口如下：

- PostLogStoreLogs
- ListShards

当RAM子账号使用Producer时，需要对以上两个API进行授权：

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

关于子账号及其授权操作请参见为RAM角色授权。

1. 初始化Producer。
   i. 初始化配置参数Properties。
   Producer初始化步骤与Consumer类似。 Producer包含以下参数，一般情况下使用默认值即可，如有需要，可以自定义配置。

```
// 用于发送数据的io线程的数量，默认是8
ConfigConstants.LOG_SENDER_IO_THREAD_COUNT
// 该值定义日志数据被缓存发送的时间，默认是3000
ConfigConstants.LOG_PACKAGE_TIMEOUT_MILLIS
// 缓存发送的包中日志的数量，默认是4096
ConfigConstants.LOG_LOGS_COUNT_PER_PACKAGE
// 缓存发送的包的大小，默认是3Mb
ConfigConstants.LOG_LOGS_BYTES_PER_PACKAGE
// 作业可以使用的内存总的大小，默认是100Mb
ConfigConstants.LOG_MEM_POOL_BYTES
```

> **说明** 上述参数不是必选参数，用户可以不设置，直接使用默认值。

ii. 重载LogSerializationSchema，定义将数据序列化成RawLogGroup的方法。
RawLogGroup是log的集合，如果用户需要使用日志服务的Shard HashKey功能，指定数据写到某一个shard中，可以使用LogPartitioner产生数据的HashKey。
示例：

```
FlinkLogProducer<String> logProducer = new FlinkLogProducer<String>(new SimpleLogSerializer(),
configProps);
logProducer.setCustomPartitioner(new LogPartitioner<String>() {
      // 生成32位hash值
      public String getHashKey(String element) {
          try {
              MessageDigest md = MessageDigest.getInstance("MD5");
              md.update(element.getBytes());
              String hash = new BigInteger(1, md.digest()).toString(16);
              while(hash.length() < 32) hash = "0" + hash;
              return hash;
          } catch (NoSuchAlgorithmException e) {
          }
          return  "00000000000000000000000000000000000000000000000000000000000000000000000000000000";
      }
  });
```

> **说明** LogPartitioner为可选项，如您没有配置，数据会随机写入某一个Shard。

2. 执行以下示例语句，将模拟产生的字符串写入日志服务。

```
// 将数据序列化成日志服务的数据格式
class SimpleLogSerializer implements LogSerializationSchema<String> {
    public RawLogGroup serialize(String element) {
        RawLogGroup rlg = new RawLogGroup();
        RawLog rl = new RawLog();
        rl.setTime((int)(System.currentTimeMillis() / 1000));
        rl.addContent("message", element);
        rlg.addLog(rl);
        return rlg;
    }
}
public class ProducerSample {
    public static String sEndpoint = "cn-hangzhou.log.aliyuncs.com";
    public static String sAccessKeyId = "";
    public static String sAccessKey = "";
    public static String sProject = "ali-cn-hangzhou-sls-admin";
    public static String sLogstore = "test-flink-producer";
    private static final Logger LOG = LoggerFactory.getLogger(ConsumerSample.class);
    public static void main(String[] args) throws Exception {
        final ParameterTool params = ParameterTool.fromArgs(args);
        final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        env.getConfig().setGlobalJobParameters(params);
        env.setParallelism(3);
        DataStream<String> simpleStringStream = env.addSource(new EventsGenerator());
        Properties configProps = new Properties();
        // 设置访问日志服务的域名
        configProps.put(ConfigConstants.LOG_ENDPOINT, sEndpoint);
        // 设置访问日志服务的ak
        configProps.put(ConfigConstants.LOG_ACCESSSKEYID, sAccessKeyId);
        configProps.put(ConfigConstants.LOG_ACCESSKEY, sAccessKey);
        // 设置日志写入的日志服务project
        configProps.put(ConfigConstants.LOG_PROJECT, sProject);
        // 设置日志写入的日志服务Logstore
        configProps.put(ConfigConstants.LOG_LOGSTORE, sLogstore);
        FlinkLogProducer<String> logProducer = new FlinkLogProducer<String>(new SimpleLogSerializer(),
configProps);
        simpleStringStream.addSink(logProducer);
        env.execute("flink log producer");
    }
    // 模拟产生日志
    public static class EventsGenerator implements SourceFunction<String> {
        private boolean running = true;
        @Override
        public void run(SourceContext<String> ctx) throws Exception {
            long seq = 0;
            while (running) {
                Thread.sleep(10);
                ctx.collect((seq++) + "-" + RandomStringUtils.randomAlphabetic(12));
            }
        }
        @Override
        public void cancel() {
            running = false;
        }
    }
}
```

6.7. Logstash消费

日志服务除了支持各种语言、实时计算消费方式外，还支持通过Logstash消费日志数据，您可以直接配置日志服务的Logstash输入源插件来获取日志服务中数据，写入到其他系统中，例如Kafka、HDFS等。

功能特性

- **支持分布式协同消费**：可配置多台服务器同时消费某一Logstore。
- **高性能**：基于Java ConsumerGroup实现，单核消费速度可达20MB/s。
- **高可靠**：消费进度保存到服务端，异常恢复后会从上一次消费的checkpoint处自动恢复消费。
- **自动负载均衡**：根据消费者数量自动分配Shard，消费者增加或退出后会自动Rebalance。

6.8. Spark Streaming消费

日志采集到日志服务之后，可以使用日志服务提供的Spark SDK在Spark Streaming中对数据进行处理。

日志服务提供的Spark SDK实现了Receiver和Direct两种消费模式。

Maven依赖：

```
<dependency>
  <groupId>com.aliyun.emr</groupId>
  <artifactId>emr-logservice_2.11</artifactId>
  <version>1.7.2</version>
</dependency>
```

Receiver模式

Receiver模式通过消费组从日志服务消费数据并暂存在Spark Executor，Spark Streaming Job启动之后从Executor读取并处理数据。每条数据以JSON字符串的形式返回。消费组自动定时保存checkpoint到服务端，无需手动更新checkpoint。

- 使用示例。

```
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.aliyun.logservice.LoghubUtils
import org.apache.spark.streaming.{Milliseconds, StreamingContext}
import org.apache.spark.SparkConf

object TestLoghub {
  def main(args: Array[String]): Unit = {
    if (args.length < 7) {
      System.err.println(
        """Usage: TestLoghub <project> <logstore> <loghub group name> <endpoint>
          |         <access key id> <access key secret> <batch interval seconds>
        """.stripMargin)
      System.exit(1)
    }

    val project = args(0)
    val logstore = args(1)
    val consumerGroup = args(2)
    val endpoint = args(3)
    val accessKeyId = args(4)
    val accessKeySecret = args(5)
    val batchInterval = Milliseconds(args(6).toInt * 1000)

    def functionToCreateContext(): StreamingContext = {
      val conf = new SparkConf().setAppName("Test Loghub")
      val ssc = new StreamingContext(conf, batchInterval)
      val loghubStream = LoghubUtils.createStream(
        ssc,
        project,
        logstore,
        consumerGroup,
        endpoint,
        accessKeyId,
        accessKeySecret,
        StorageLevel.MEMORY_AND_DISK)

      loghubStream.checkpoint(batchInterval * 2).foreachRDD(rdd =>
        rdd.map(bytes => new String(bytes)).top(10).foreach(println)
      )
      ssc.checkpoint("hdfs:///tmp/spark/streaming") // set checkpoint directory
      ssc
    }

    val ssc = StreamingContext.getOrCreate("hdfs:///tmp/spark/streaming", functionToCreateContext _)

    ssc.start()
    ssc.awaitTermination()
  }
}
```

- **参数说明。**

参数名称	类型	说明
project	String	日志服务Project。
logstore	String	日志服务Logstore。
consumerGroup	String	消费组名称。
endpoint	String	日志服务Project所在region的endpoint。
accessKeyId	String	访问日志服务的Access Key ID。

accessKeySecret	String	访问日志服务的Access Key Secret。

- 注意事项。

  默认配置下，这种方式在异常情况下可能导致数据丢失，为了避免此类情况发生，可以开启Write-Ahead Logs开关（Spark 1.2 之后支持）。请参见Spark部署章节了解更多关于Write-Ahead Logs的细节。

Direct 模式

Direct模式与Receiver相比不再需要消费组，而是直接使用API在任务运行时从服务端直接请求数据。与Receiver模式相比，Direct模式具有如下优势。

- 简化并行：Spark partition个数与Logstore Shard总数一致。只需分裂Shard即可提高任务的并行度。
- 更加高效：不再需要Write-Ahead Logs来保证数据不丢失。
- Exactly-once 语义：数据直接从服务端按需获取，任务成功之后再提交checkpoint。极端情况下由于Spark异常退出或者其他原因，出现任务未正常结束的情况，可能会导致部分数据被重复消费。

Direct模式需要依赖ZooKeeper环境，用于临时保存中间状态。同时，必须设置checkpoint目录。中间状态数据保存在ZooKeeper内对应的checkpoint目录内。如果任务重启之后希望重新消费，需要在ZooKeeper内删除该目录，并更换消费组名称。

- 使用示例。

```
import com.aliyun.openservices.loghub.client.config.LogHubCursorPosition
import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Milliseconds, StreamingContext}
import org.apache.spark.streaming.aliyun.logservice.{CanCommitOffsets, LoghubUtils}

object TestDirectLoghub {
  def main(args: Array[String]): Unit = {
    if (args.length < 7) {
      System.err.println(
        """Usage: TestDirectLoghub <project> <logstore> <loghub group name> <endpoint>
          |         <access key id> <access key secret> <batch interval seconds> <zookeeper
host:port=localhost:2181>
        """.stripMargin)
      System.exit(1)
    }

    val project = args(0)
    val logstore = args(1)
    val consumerGroup = args(2)
    val endpoint = args(3)
    val accessKeyId = args(4)
    val accessKeySecret = args(5)
    val batchInterval = Milliseconds(args(6).toInt * 1000)
    val zkAddress = if (args.length >= 8) args(7) else "localhost:2181"

    def functionToCreateContext(): StreamingContext = {
      val conf = new SparkConf().setAppName("Test Direct Loghub")
      val ssc = new StreamingContext(conf, batchInterval)
      val zkParas = Map("zookeeper.connect" -> zkAddress,
        "enable.auto.commit" -> "false")
      val loghubStream = LoghubUtils.createDirectStream(
        ssc,
        project,
        logStore,
        consumerGroup,
        accessKeyId,
        accessKeySecret,
        endpoint,
        zkParas,
        LogHubCursorPosition.END_CURSOR)

      loghubStream.checkpoint(batchInterval).foreachRDD(rdd => {
        println(s"count by key: ${rdd.map(s => {
          s.sorted
          (s.length, s)
        }).countByKey().size}")
        loghubStream.asInstanceOf[CanCommitOffsets].commitAsync()
      })
      ssc.checkpoint("hdfs:///tmp/spark/streaming") // set checkpoint directory
      ssc
    }

    val ssc = StreamingContext.getOrCreate("hdfs:///tmp/spark/streaming", functionToCreateContext _)
    ssc.start()
    ssc.awaitTermination()
  }
}
```

- **参数说明。**

参数名称	类型	说明
project	String	日志服务Project。

logstore	String	日志服务logstore。
consumerGroup	String	消费组名称（仅用于保存消费位置）。
endpoint	String	日志服务Project所在region的endpoint。
accessKeyId	String	访问日志服务的Access Key ID。
accessKeySecret	String	访问日志服务的Access Key Secret。
zkAddress	String	ZooKeeper连接地址。

- 参数设置。
  使用Direct模式消费时，需要指定每个批次每个shard消费的数据行数，否则数据读取过程无法结束。可以通过如下两个配置对单个批次进行限流。

配置	描述	默认值
spark.loghub.batchGet.step	单次请求返回LogGroup个数。	100
spark.streaming.loghub.maxRatePerShard	单个shard每秒希望处理的日志条数。	10000

  每个batch实际处理的日志条数为：`shard个数 * max(spark.loghub.batchGet.step * n * 日志组中的日志条数, spark.streaming.loghub.maxRatePerShard * duration)`。
  - **n**：返回总行数达到 `spark.streaming.loghub.maxRatePerShard * duration` LogGroup中的平均日志条数。
  - **duration**：批处理间隔，单位为秒。

  如果需要Union多个DStream，那Shard个数就是所有Logstore的Shard个数之和。
  - 举例说明。
    假设Shard个数为100，每个LogGroup包含50条日志，批处理间隔为2秒。希望每批次处理20000条日志，可以按照如下方式配置。
    - `spark.loghub.batchGet.step: 4`。
    - `spark.streaming.loghub.maxRatePerShard: 100`。

    如果每个LogGroup包含60条日志，希望每批次处理20000条日志，在上述配置下，实际会处理 `60*4*100 = 24000` 行数据。
  - 精确限流。

    `spark.loghub.batchGet.step` 越小限流越精确，但是可能造成请求次数增加。建议先统计平均每次写入请求（一个LogGroup）内包含多少行日志，然后合理设置如上两个配置。

6.9. 实时计算（Blink）消费

阿里云实时计算（Realtime Compute/Blink） 可以通过创建日志服务源表的方式，直接消费日志服务中的数据。

日志服务本身是流数据存储，实时计算能将其作为流式数据输入。对于日志服务而言，每条日志包含一组Key-Value形式的字段，假设有如下的日志内容：

```
__source__:  11.85.123.199
__tag__:__receive_time__:  1562125591
__topic__:  test-topic
a:  1234
b:  0
c:  hello
```

StreamCompute可以定义如下的DDL：

```
create table sls_stream(
  a int,
  b int,
  c varchar
) with (
  type ='sls',
  endPoint ='<your endpoint>',
  accessId ='<your access key id>',
  accessKey ='<your access key>',
  startTime = '2017-07-05 00:00:00',
  project ='ali-cloud-streamtest',
  logStore ='stream-test',
  consumerGroup ='consumerGroupTest1'
);
```

属性字段

除日志字段外，支持提取如下三个属性字段和tag中的自定义字段，例如 `__receive_time__` ： 表 1. 属性字段

字段名	说明
`__source__`	日志来源。
`__topic__`	日志主题。
`__timestamp__`	日志时间。

属性字段的提取需要添加HEADER声明，例如：

```
create table sls_stream(
  __timestamp__ bigint HEADER,
  __receive_time__ bigint HEADER
  b int,
  c varchar
) with (
  type ='sls',
  endPoint ='<your endpoint>',
  accessId ='<your access key id>',
  accessKey ='<your access key>',
  startTime = '2017-07-05 00:00:00',
  project ='ali-cloud-streamtest',
  logStore ='stream-test',
  consumerGroup ='consumerGroupTest1'
);
```

WITH参数

WITH参数说明请参见如下表格。

参数	是否必选	说明
endPoint	是	日志服务Endpoint。
accessId	是	访问日志服务的AccessKey ID。
accessKey	是	访问日志服务的密钥AccessKey Secret。
project	是	日志服务Project名称。
logStore	是	日志服务Logstore名称。
consumerGroup	否	消费组名称。
startTime	否	消费日志开始的时间点。

heartBeatIntervalMills	否	消费客户端的心跳间隔时间。默认为10秒。
maxRetryTimes	否	读取最大重试次数，默认5次。
batchGetSize	否	单次读取logGroup条数，默认为10条。如果Blink的版本是1.4.2+，则默认为100条， 最大1000条。
columnErrorDebug	否	是否打开调试开关，如果打开，会把解析异常的日志打印出来。默认为 `false` 。

类型映射

日志服务中的字段都是字符串类型，和实时计算字段类型对应关系如下所示。强烈建议用户使用该对应关系进行DDL声明：

日志服务字段类型	实时计算字段类型
STRING	VARCHAR

如果使用其他类型也会尝试自动转换，例如“1000”或者“2018-01-12 12:00:00”也可以定义为bigint和timestamp类型。

> **说明**
> - Blink 2.2.0 以下的版本不支持Shard的扩容和缩容，如果分裂或者缩容了某个正在实时计算读取的LogStore，会导致任务持续出错，不可自动恢复，该情况下需要重新启动任务来使任务恢复正常。
> - 所有Blink版本均不支持对正在消费的LogStore进行删除重建。
> - 1.6.0及之前版本，在Shards数目很多的情况下指定consumerGroup可能会影响读取性能。
> - 建表时，暂不支持将日志服务的数据转换成MAP类型。
> - 对于不存在的字段会置为null。
> - 字段顺序支持无序（建议字段顺序和表中定义一致）。
> - batchGetSize指明的是logGroup获取的数量，如果单条日志的大小和batchGetSize都很大，可能会导致JAVA系统频繁的对内存数据进行垃圾回收。

注意事项

- 如果一个Shard没有新数据写入，会导致Job的整体延迟增加。遇到这种情况，只需要把并发数调整为读写的Shard数量即可。
- 并发数建议和Shard个数一致，否则当两个Shard读取速度差异较大时，理论上在追数据场景存在数据被过滤掉的风险。
- 针对 `__tag__:__hostname__` 、 `__tag__:__path__` 等字段，去掉前面的tag，按照获取属性字段的方式获取即可。

  > **说明** 调试时无法抽取到该类型数据，建议用户用本地DEBUG方法，上线运行打印到日志中查看。

7.访问控制RAM

7.1. 简介

RAM（Resource Access Management）是阿里云专有云提供的资源访问控制服务。

通过RAM管理，您可以集中管理您的用户（例如员工、系统或应用程序），并控制用户可以访问您名下哪些资源的权限。

RAM管理主要有以下两个功能：

- RAM角色

  当您需要在授权一级组织内的云服务使用或查看当前组织内的其他资源时，需要创建对应的RAM角色，该角色包含了云服务可以对资源进行的操作。

  RAM角色只有系统管理员和一级组织管理员可以创建。

- 用户组

  当您需要多个用户共同使用某个组织下的云资源时，可以在该组织下创建多个用户来实现多个用户对同一个云资源的不同操作权限。

  访问控制（RAM）通过用户组对职责相同的用户进行分类并授权，可以更加高效地管理用户及其权限。

  您可以通过创建RAM授权策略将不同的操作权限分配给不同用户组。

7.2. 创建RAM角色

当您需要在授权一级组织内的云服务使用或查看当前组织内的其他资源时，需要创建对应的RAM角色，该角色包含了云服务可以对资源进行的操作。

操作步骤

1. 使用管理员账号登录Apsara Uni-manager控制台。
   具体登录方法请参见登录日志服务控制台。
2. 在页面顶部菜单栏上，单击**企业**。
3. 在**企业**页面的左侧导航栏中，单击**角色管理**。
4. 单击页面右上角**创建RAM角色**。
5. 在**角色管理-创建RAM角色**详情页面，填写**角色名称**和**描述**信息。
6. 单击**创建RAM角色并配置**。

7.3. 创建用户

管理员可以创建一个用户，并为用户赋予不同的角色，以满足不同用户对系统的访问控制需求。

操作步骤

1.
2. 在页面顶部菜单栏上，单击**企业**。
3. 在**企业**页面的左侧导航栏中，单击**用户管理**。
4. 单击**创建**按钮。
5. 在弹出的对话框中，配置用户信息。

配置	说明
用户名	用户的个人云平台账号名。
显示名	用户账号的显示名。
角色	选择用户的角色。
组织	选择用户所在的组织。
登录策略	登录策略将限制用户的登录时间和登录地址。默认情况下，新创建的用户会自动绑定默认策略。

手机	用户的手机号码。手机号码用于系统以短信形式通知用户资源的申请和使用情况，请填写正确的手机号码。
座机	（可选）用户的座机号码。
Email	用户的邮箱地址。邮箱用于系统以邮件方式通知用户资源的申请和使用情况，请填写正确的邮箱地址。
钉钉密钥	（可选）根据需求配置钉钉密钥。
启用短信通知	勾选**启用短信通知**，系统以短信形式通知用户资源的申请和使用情况。
启用邮件通知	勾选**启用邮件通知**，系统以邮件形式通知用户资源的申请和使用情况。
启用钉钉通知	勾选**启用钉钉通知**，系统以钉钉形式通知用户资源的申请和使用情况。

6. 单击**确定**。

7.4. 新建用户组

您可以在所选组织中创建用户组，批量对用户进行授权。

前提条件

在新建用户组前，需要已创建组织。

背景信息

用户组与用户关系：

- 用户组里可以包含0个到多个用户。
- 用户可以选择性的加入用户组，不是必须加入。
- 用户可以加入多个用户组。

用户组与组织关系：

- 用户组只能属于一个组织。
- 一个组织下可以创建多个用户组。

用户组与角色关系：

- 用户组目前只允许绑定一个角色。
- 一个角色可以关联到多个用户组。
- 角色关联用户组时，自动为用户组内的用户授予该角色权限。

用户组与资源集关系：

- 资源集可以添加0到多个用户组。
- 用户组可以加入到多个资源集。

操作步骤

1.
2. 在页面顶部菜单栏上，单击**企业**。
3. 在**企业**页面的左侧导航栏中，单击**用户组管理**。
4. 单击页面右上角**新建用户组**。
5. 在弹出的对话框中填写**用户组名**和**组织**。
6. 单击**确定**。

7.5. 添加用户到用户组

您可以为用户组添加成员用户。

操作步骤

1.
2. 在页面顶部菜单栏上，单击**企业**。
3. 在**企业**页面的左侧导航栏中，单击**用户组管理**。
4. 选择待添加用户的用户组，单击**操作**列中的**添加用户**。

5. 从待选用户名列表中选择需要添加的用户名，单击向右的箭头，移动到已添加的用户名列表中。
6. 单击**确定**。

7.6. 新增权限策略

当您需要使用某个云服务访问您其他的云资源时，您需要创建权限策略，以绑定用户组。

操作步骤

1.
2. 在页面顶部菜单栏上，单击**企业**。
3. 在**企业**页面的左侧导航栏中，单击**角色管理**。
4. 在角色名列表中，单击目标RAM角色名后面的**更多 > 修改**。
5. 单击**权限管理**。
6. 单击**新增权限策略**。
7. 在**添加新权限策略**对话框中，填写权限策略信息。

您可以参见RAM自定义授权场景填写策略内容。

7.7. 为RAM角色授权

为RAM角色授权后，用户组中的所有RAM用户将拥有该用户组的所有权限。

操作步骤

1.
2. 在页面顶部菜单栏上，单击**企业**。

3. 在**企业**页面的左侧导航栏中，单击**角色管理**。
4.
5. 在**权限管理**页签下，单击**选择已有权限策略**。
6. 在**选择已有RAM权限策略**对话框中，选择RAM权限策略，单击**确定**。
   如果没有可选的RAM权限策略，请先新增权限策略。更多信息，请参见新增权限策略。

7.8. RAM自定义授权场景

通过访问控制RAM可以为名下的RAM用户（子用户）授权，本文档为您介绍常见的自定义授权场景和授权内容。

背景信息

基于安全考虑，建议您为RAM用户授予最小可用权限。通常情况下，您需要为RAM用户授予Project列表的只读权限，否则RAM用户无法进入Project列表查看资源。

控制台场景

- Project只读权限
  例如主账号需要授予RAM用户以下权限：
  - RAM用户具有主账号Project列表的查看权限。
  - RAM用户具有主账号指定Project的只读权限。

  同时满足上述权限的权限策略如下：

```
{
   "Version": "1",
   "Statement": [
     {
       "Action": ["log:ListProject"],
       "Resource": ["acs:log:*:*:project/*"],
       "Effect": "Alow"
      },
     {
       "Action": [
         "log:Get*",
         "log:List*"
       ],
       "Resource": "acs:log:*:*:project/<指定的project名称>/*",
       "Effect": "Allow"
     }
   ]
 }
```

- 指定Logstore的只读权限和快速查询的创建、使用权限
  例如主账号需要授予RAM用户以下权限：
  - RAM用户具有主账号Project列表的查看权限。
  - RAM用户具有指定Logstore的只读权限，同时具有创建并管理快速查询的权限。

  同时满足上述权限的权限策略如下：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "log:ListProject"
      ],
      "Resource": "acs:log:*:*:project/*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:List*"
      ],
      "Resource": "acs:log:*:*:project/<指定的Project名称>/logstore/*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:Get*",
        "log:List*"
      ],
      "Resource": [
        "acs:log:*:*:project/<指定的Project名称>/logstore/<指定的Logstore名称>"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:List*"
      ],
      "Resource": [
        "acs:log:*:*:project/<指定的Project名称>/dashboard",
        "acs:log:*:*:project/<指定的Project名称>/dashboard/*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:Get*",
        "log:List*",
        "log:Create*"
      ],
      "Resource": [
        "acs:log:*:*:project/<指定的Project名称>/savedsearch",
        "acs:log:*:*:project/<指定的Project名称>/savedsearch/*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

> **说明** 授权策略中Resource内容结尾不带*仅表示当前资源，带*表示当前资源下的其他资源，内容用*代替。

- 指定Logstore的只读权限及指定Project中快速查询和仪表盘的只读权限
  例如主账号需要授予RAM用户以下权限：
  - RAM用户具有主账号Project列表的查看权限。
  - RAM用户具有指定Logstore的只读权限，同时具有查看该Project中所有的快速查询和仪表盘列表的权限。

  同时满足上述权限的权限策略如下：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "log:ListProject"
      ],
      "Resource": "acs:log:*:*:project/*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:List*"
      ],
      "Resource": "acs:log:*:*:project/<指定的Project名称>/logstore/*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:Get*",
        "log:List*"
      ],
      "Resource": [
        "acs:log:*:*:project/<指定的Project名称>/logstore/<指定的Logstore名称>"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:Get*",
        "log:List*"
      ],
      "Resource": [
        "acs:log:*:*:project/<指定的Project名称>/dashboard",
        "acs:log:*:*:project/<指定的Project名称>/dashboard/*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:Get*",
        "log:List*"
      ],
      "Resource": [
        "acs:log:*:*:project/<指定的Project名称>/savedsearch",
        "acs:log:*:*:project/<指定的Project名称>/savedsearch/*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

API场景

- 指定Project的写入权限
  授予RAM用户向指定Project写入数据的权限，不包含查询等其他操作权限。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "log:Post*"
      ],
      "Resource": "acs:log:*:*:project/<指定的project名称>/*",
      "Effect": "Allow"
    }
  ]
}
```

- 指定Project的消费权限
  授予RAM用户消费指定Project数据的权限，不包含数据写入、查询等其他操作权限。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "log:ListShards",
        "log:GetCursorOrData",
        "log:GetConsumerGroupCheckPoint",
        "log:UpdateConsumerGroup",
        "log:ConsumerGroupHeartBeat",
        "log:ConsumerGroupUpdateCheckPoint",
        "log:ListConsumerGroup",
        "log:CreateConsumerGroup"
      ],
      "Resource": "acs:log:*:*:project/<指定的project名称>/*",
      "Effect": "Allow"
    }
  ]
}
```

- 指定Logstore的消费权限
  授予RAM用户消费指定Logstore数据的权限，不包含数据写入、查询等其他操作权限。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "log:ListShards",
        "log:GetCursorOrData",
        "log:GetConsumerGroupCheckPoint",
        "log:UpdateConsumerGroup",
        "log:ConsumerGroupHeartBeat",
        "log:ConsumerGroupUpdateCheckPoint",
        "log:ListConsumerGroup",
        "log:CreateConsumerGroup"
      ],
      "Resource": [
        "acs:log:*:*:project/<指定的project名称>/logstore/<指定的Logstore名称>",
        "acs:log:*:*:project/<指定的project名称>/logstore/<指定的Logstore名称>/*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

8.FAQ

8.1. 日志采集

8.1.1. Logtail排查简介

配置Logtail采集日志后，如果预览页面为空，或查询页面显示无数据，可以根据本文步骤进行排查。

操作步骤

1. 确认机器组内Logtail心跳是否正常。

   在日志服务控制台查看Logtail机器组心跳状态，详细步骤请参见查看机器组状态。

   如果心跳OK则直接进入下一步；如果心跳FAIL则需要进一步排查。详细排查步骤请参见Logtail 机器无心跳。

2. 确认是否创建了Logtail采集配置。
   确认Logtail客户端状态正常后，则需要确认是否已创建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。目录结构支持完整路径和通配符两种模式。

3. 确认Logtail配置是否已应用到机器组。
   参见管理机器组配置确认目标配置是否已经应用到机器组。

4. 检查采集错误。
   确认Logtail配置正常后，您需要确认日志文件是否实时产生新数据。Logtail只采集增量数据，存量文件如果没有更新则不会被读取。如日志有更新但未在日志服务中查询到，您可以通过以下方式诊断。
   - 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
     - Linux：/usr/local/ilogtail/logtail_plugin.LOG （http、mysql binlog、mysql 查询结果等输入源的日志记录）
     - 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。参见配置启动参数。

8.1.2. Logtail 机器无心跳

配置Logtail采集日志数据，如果Logtail机器组心跳状态不正常，请参考本文档进行处理。

背景信息

如果使用Logtail采集日志，在服务器上安装Logtail之后，Logtail会定时向服务端发送心跳包。如果机器组状态页面显示机器无心跳，说明客户端和服务端连接失败。

步骤1：检查是否已安装Logtail

请执行如下命令查看Logtail状态。

- Linux服务器

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

  如果显示 `ilogtail is running` ，表示已安装Logtail，例如：

  ```
  [root@****************~]# sudo /etc/init.d/ilogtaild status
  ilogtail is running
  ```

- Windows服务器

  i. 使用快捷键Win+R，输入services.msc进入本地服务。

  ii. 查看LogtailDaemon、LogtailWorker两个Windows Service的运行状态。如果正在运行，表示已安装Logtail。

如未安装Logtail客户端，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）进行安装，安装时请务必按照您日志服务Project所属Region进行安装。如果Logtail正在运行，请执行下一步检查。

步骤2：检查Logtail安装参数是否正确

安装Logtail时，需要为客户端指定正确的服务端访问入口，即根据日志服务Project所在地域选择Logtail安装参数。如果安装参数或安装脚本错误，可能会导致Logtail机器无心跳。

Logtail配置文件ilogtail_config.json中记录了Logtail安装参数及所选的安装方式，该文件的路径为：

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

检查ilogtail_config.json中 `config_server_address` 字段配置的Endpoint。例如ilogtail_config.json中记录Logtail配置的Endpoint为 `logtail.cn-qingdao-env25-d01.sls-pub.inter.env25.shuguang.com` ，则可以通过执行如下命令检查连通性。

- Linux服务器

```
curl logtail.cn-qingdao-env25-d01.sls-pub.inter.env25.shuguang.com
```

- Windows服务器

```
telnet logtail.cn-qingdao-env25-d01.sls-pub.inter.env25.shuguang.com 80
```

如果安装时选择了错误的参数，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）重新安装。

如果Logtail已正确安装，请执行下一步检查。

步骤3：检查机器组配置的IP地址是否正确

机器组中配置的IP地址必须和Logtail获取到的服务器地址一致，否则机器组无心跳、或无法采集到日志数据。Logtail获取机器IP的方式如下：

- 如果没有设置主机名绑定，会取服务器的第一块网卡IP。
- 如果在文件/etc/hosts中设置了主机名绑定，则会取绑定主机名对应的IP。

> **说明** 可以通过**hostname**查看主机名。

排查步骤：

1. 查看Logtail获取的IP地址。
   文件app_info.json的 `ip` 字段中记录了Logtail获取的IP地址，该文件的路径为：
   - Linux服务器：/usr/local/ilogtail/app_info.json
   - Windows x64服务器：C:\Program Files (x86)\Alibaba\Logtail\app_info.json
   - Windows x86服务器：C:\Program Files\Alibaba\Logtail\app_info.json

   > **说明**
   > - 如果 `app_info.json` 文件中 `ip` 字段为空，Logtail无法工作。此时需为服务器设置IP地址并重启Logtail。
   > - 文件 `app_info.json` 仅做记录，修改该文件并不会改变Logtail获取的IP地址。

2. 查看机器组中配置的地址。
   在日志服务控制台单击Project名称，然后在左侧导航选择**机器组**，单击目标机器组名称后在**机器组配置**页面查看状态。

   如果服务端机器组内填写的IP与客户端获取的IP不一致，则需要修改。
   - 若服务端机器组填写了错误IP，请修改机器组内IP地址并保存，等待1分钟再查看心跳状态。
   - 若修改了机器上的网络配置（如修改/etc/hosts），请重新启动Logtail以获取新的IP，并根据 `app_info.json` 文件中的 `ip` 字段修改机器组内设置的IP地址。

重启Logtail的方式：

- Linux服务器

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

- Windows服务器

  使用快捷键Win+R，输入services.msc进入本地服务，找到并重启LogtailWorker。

8.1.3. 查询本地采集状态

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

使用指南

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

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

```
/etc/init.d/ilogtaild -h
Usage: ./ilogtaild { start | stop (graceful, flush data and save checkpoints) | force-stop | status | -
h for help}$
logtail insight, version : 0.1.0
commond list :
       status all [index]
             get logtail running status
       status active [--logstore | --logfile] index [project] [logstore]
             list all active logstore | logfile. if use --logfile, please add project and logstore. defa
ult --logstore
       status logstore [--format=line | json] index project logstore
             get logstore status with line or json style. default --format=line
       status logfile [--format=line | json] index project logstore fileFullPath
             get log file status with line or json style. default --format=line
       status history beginIndex endIndex  project logstore [fileFullPath]
             query logstore | logfile history status.
index :   from 1 to 60. in all, it means last $(index) minutes; in active/logstore/logfile/history, it m
eans last $(index)*10 minutes
```

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

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

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

all命令

命令格式

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

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

示例

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

输出信息描述

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

active命令

命令格式

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

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

示例

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

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

输出信息描述

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

logstore命令

命令格式

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

说明

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

示例

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

输出信息描述

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

Logstore状态列表

状态	含义	处理方式
ok	状态正常	无需处理。
process_block	日志解析阻塞	检查日志产生速度是否过高，若一直出现，按需调整配置启动参数修改CPU使用上限或网络发送并发限制。

parse_fail	日志解析失败	检查日志格式与日志采集配置是否一致。
send_block	当前发送出现阻塞	检查日志产生速度是否过高以及网络状态是否正常，若一直出现，按需调整配置启动参数修改CPU使用上限或网络发送并发限制。

logfile命令

命令格式

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

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

示例

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

输出信息描述

关键字	含义	单位
status	该日志文件当前窗口期的采集状态，参见logstore命令的status。	无
time_begin_readable	可读的开始时间。	无
time_end_readable	可读的截止时间。	无
time_begin	统计开始时间。	Unix时间戳，秒
time_end	统计结束时间。	Unix时间戳，秒
project	Project名。	无
logstore	Logstore名。	无
file_path	该日志文件路径。	无

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

history命令

命令格式

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

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

示例

```
/etc/init.d/ilogtaild status history 1 3 sls-zc-test release-test-same /disk2/test/normal/access.log
        begin_time          status      read  parse_success  parse_fail      last_read_time  read_count
avg_delay    device     inode  file_size  read_offset
 17-08-29 11:26:11              ok   62.12MB         231000           0   17-08-29 11:36:11         671
0B     64800  22544459    18.22MB      18.22MB
 17-08-29 11:16:11              ok   62.02MB         230615           0   17-08-29 11:26:10         667
0B     64800  22544456    16.36MB      16.36MB
 17-08-29 11:06:11              ok   62.12MB         231000           0   17-08-29 11:16:11         687
0B     64800  22544452    14.46MB      14.46MB
$/etc/init.d/ilogtaild status history 2 5 sls-zc-test release-test-same
        begin_time          status      read  parse_success  parse_fail      last_read_time  read_count
avg_delay  send_queue  network_error  quota_error  discard_error  send_success  send_block  send_valid
max_unsend          min_unsend    max_send_success
 17-08-29 11:16:11              ok   62.02MB         230615           0   17-08-29 11:26:10         667
0B           0              0            0              0           300       false        true   70-01-01
08:00:00   70-01-01 08:00:00   17-08-29 11:26:08
 17-08-29 11:06:11              ok   62.12MB         231000           0   17-08-29 11:16:11         687
0B           0              0            0              0           303       false        true   70-01-01
08:00:00   70-01-01 08:00:00   17-08-29 11:16:10
 17-08-29 10:56:11              ok   62.02MB         230615           0   17-08-29 11:06:10         687
0B           0              0            0              0           302       false        true   70-01-01
08:00:00   70-01-01 08:00:00   17-08-29 11:06:08
 17-08-29 10:46:11              ok   62.12MB         231000           0   17-08-29 10:56:11         692
0B           0              0            0              0           302       false        true   70-01-01
08:00:00   70-01-01 08:00:00   17-08-29 10:56:10
```

输出信息描述

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

命令返回值

正常返回值

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

```
/etc/init.d/ilogtaild status logfile --format=json 1 error-project error-logstore /no/this/file
null
echo $?
0
/etc/init.d/ilogtaild status all
ok
echo $?
0
```

异常返回值

返回值非0时，说明发生异常，请参考以下信息。

返回值	类型	输出	问题排查
10	无效命令或缺少参数	`invalid param, use -h for help.`	输入 `-h` 查看帮助。
1	查询超过1-60的时间窗口	`invalid query interval`	输出 `-h` 查看帮助。
1	无法查询到指定时间窗口	`query fail, error: $(error)` ，具体参见errno释义	可能原因是logtail启动时间小于查询时间跨度，其他情况请提交工单处理。
1	查询窗口时间不匹配	`no match time interval, please check logtail status`	检查Logtail是否在运行，其他情况请提交工单处理。

1	查询窗口内没有数据	`invalid profile, maybe logtail restart`	检查Logtail是否在运行，其他情况请提交工单处理。

示例

```
/etc/init.d/ilogtaild status nothiscmd
invalid param, use -h for help.
echo $?
10
/etc/init.d/ilogtaild status/all 99
invalid query interval
echo $?
1
```

功能使用场景示例

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

监控Logtail运行状态

通过 `all` 命令实现Logtail运行状态监控。

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

具体报警持续时间以及监控的状态范围可根据具体场景中日志采集重要程度调整。

监控日志采集进度

通过 `logstore` 命令实现具体日志库采集进度监控。

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

具体 `avg_delay_bytes` 报警阈值可根据日志采集流量调整。

判断日志文件是否采集完毕

通过 `logfile` 命令判断日志文件是否采集完毕。

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

日志采集问题排查

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

1. `send_block_flag` 为true，则说明因网络不稳定导致日志采集进度延迟。
   - 若 `send_network_quota_count` 大于0时，需要分裂Shard。
   - 若 `send_network_error_count` 大于0时，需要检查网络连通性。
   - 若无相关network error，则需要调整Logtail的发送并发以及流量限制，详情请参见配置启动参数。
2. 发送部分相关参数正常但 `avg_delay_bytes` 较高。
   - 可根据 `read_bytes` 计算出日志平均解析速度，判断日志产生流量是否异常。
   - 可适当调整logtail的资源使用限制，详情请参见配置启动参数。
3. `parse_fail_lines` 大于0。

   检查日志采集解析配置是否能够匹配所有日志。

8.1.4. 如何调试正则表达式

在配置Logtail采集文本日志时，如果选择完整正则模式解析、采集日志，需要您根据自己的日志样例配置正则表达式。本文档介绍如何调试正则表达式。

背景信息

如果您希望对您所在日志服务控制台所设置的正则表达式进行调试，您可以直接在界面上使用**验证**按钮所提供的功能来进行检

查：

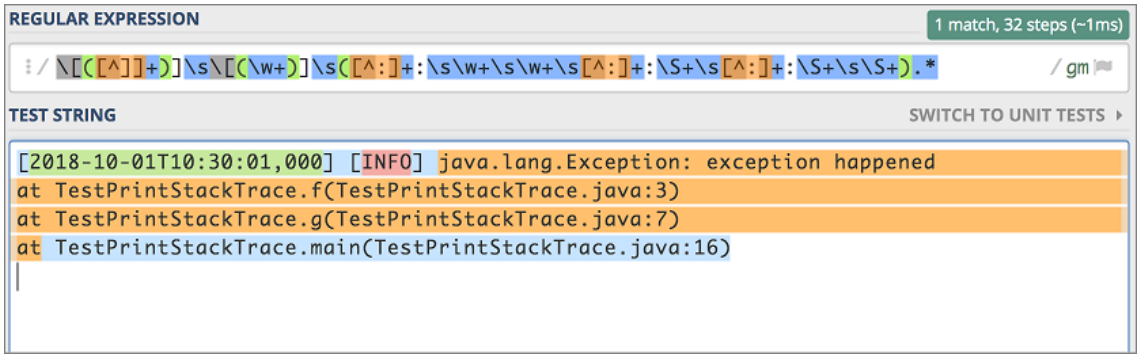
- 对于行首正则表达式，检查一下当前设置能否正确匹配出您期望的日志数量。
- 对于提取字段，检查一下各个字段中的值是否是您所希望的。

如果您希望进行更多的验证乃至调试正则表达式，您可以利用诸如Regex101、RegexTester之类的在线工具，将控制台为您自动生成的正则表达式拷贝粘贴到这些工具上，然后填充您的实际日志来进行检查、调试。

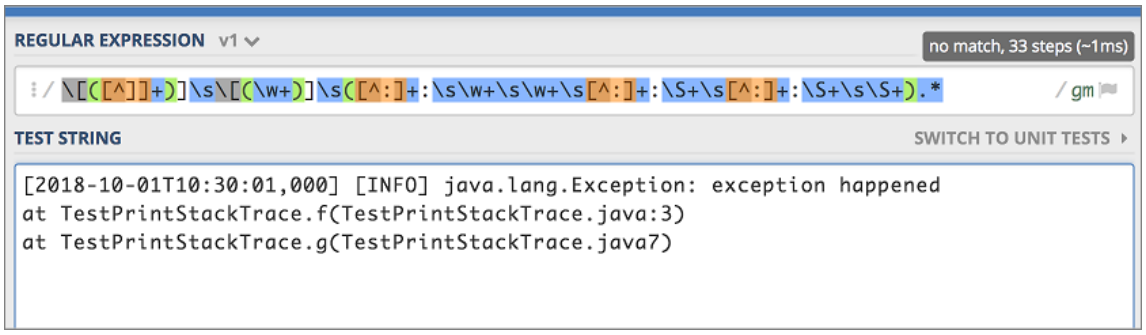
完整正则模式提供自动生成功能，可能会为多行日志的message字段生成不合适的正则。本文档以Regex101为例，对该正则进行以下检查。

操作步骤

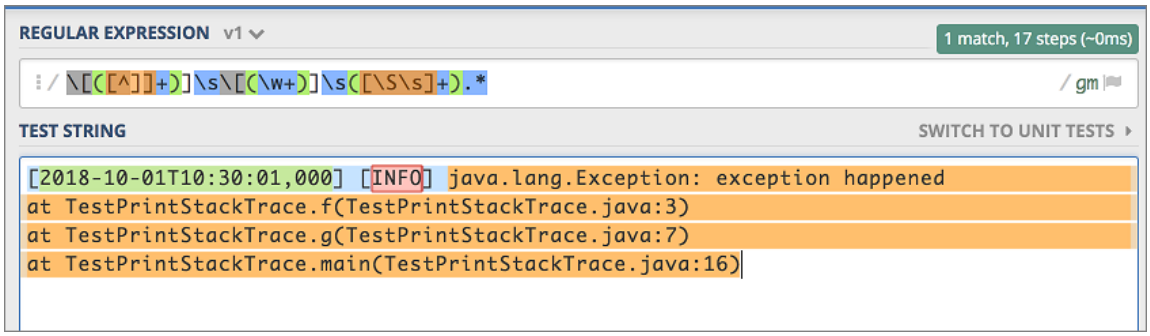
1. 拷贝日志服务根据日志样例自动生成的完整正则。
2. 打开Regex101网站。
3. 在**REGULAR EXPRESSION**中粘贴自动生成的完整正则。
   在界面的右侧，您还可以看到该正则的含义。
4. 在**TEST STRING**中贴入日志样例中的日志。
   以下示例表示正则式与日志部分匹配，`at` 之后的内容并没有被包含到 `message` 字段中（表示为橘色和蓝色），因此该表达式不完全匹配样例日志，即对于该样例日志来说，这条正则表达式是错误的，使用这条正则表达式无法正常采集到所有日志数据。



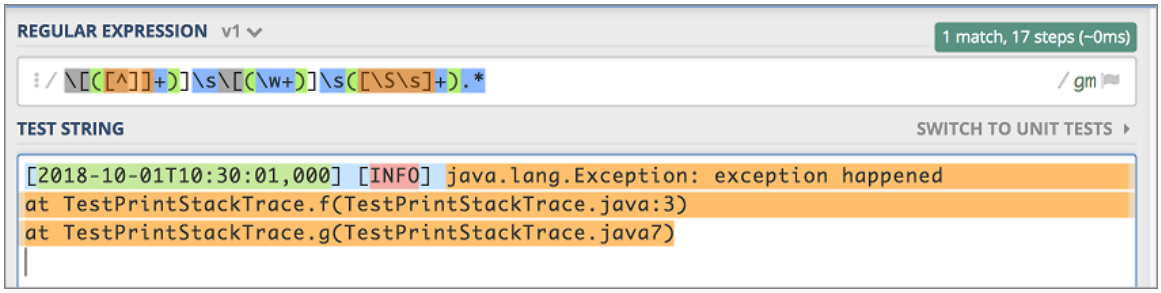
5. 验证日志中两个冒号的情况。
   以下示例表示匹配失败。



6. 将最后一个正则表达式替换为 `[\S\s]+` ，并再次尝试检查匹配程度。
   `at` 之后的内容如下：



只有两个冒号的日志：



您可以按照以上方法来对您的正则表达式进行调试、修改，最终应用于Logtail采集配置中。

8.1.5. 如何优化正则表达式的性能

通过优化正则表达式的性能，可以达到优化采集性能的目的。

关于如何优化正则表达式，为您提供以下建议：

- 使用更为精确的字符

  不要盲目地使用 `.*` 来匹配字段，这个表达式包含了很大的搜索空间，很容易就发生误匹配、导致匹配性能下降。例如您要提取的字段只由字母组成，那么使用 `[A-Za-z]` 即可。

- 使用正确的量词

  不盲目地使用 +, *。例如您使用 `\d` 来匹配 IP 地址，那么相比 `\d+` ，`\d{1,3}` 可能会具有更高的性价比。

- 多次调试

  调试类似于排查错误，您同样可以在网站**Regex101**上对您的正则表达式所花费的时间进行调试，一旦发现大量的回溯，可以及时优化。

8.1.6. 如何通过完整正则模式采集多种格式日志

完整正则模式要求日志必须采用统一的格式，但有些时候日志中可能会包含多种格式，您可以采用Schema-On-Write和Schema-On-Read两种模式处理。

以 Java 日志为例，作为一个程序日志，它一般既包含正常信息，也会包含异常栈等错误信息：

- WARNING 类型的多行日志
- INFO 类型的简单文本日志
- DEBUG 类型的键值日志

```
[2018-10-01T10:30:31,000] [WARNING] java.lang.Exception: another exception happened
    at TestPrintStackTrace.f(TestPrintStackTrace.java:3)
    at TestPrintStackTrace.g(TestPrintStackTrace.java:7)
    at TestPrintStackTrace.main(TestPrintStackTrace.java:16)
[2018-10-01T10:30:32,000] [INFO] info something
[2018-10-01T10:30:33,000] [DEBUG] key:value key2:value2
```

对此，有两种方案可以考虑：

- Schema-On-Write：为同样的一份日志应用多个采集配置，每个采集配置具有不同的正则配置，从而能够正确地实现字段提取。

  > **说明** Logtail 不支持对同一个文件同时应用多个采集配置，您需要为该文件所在的目录建立多个软链接，每个配置作用于不同的软链接目录来间接实现多个配置同时采集一个文件。

- Schema-On-Read：使用它们共同的正则表达式来采集。

  比如说采用多行日志采集，将时间和日志等级作为行首正则，剩余的部分都作为 message。如果希望进一步分析 message，可以为该字段建立索引，然后利用日志服务的正则提取等查询分析功能，从 message 字段提取需要的内容，基于该内容进行分析。

  > **说明** 此方案仅推荐应用于同时分析的日志数量较小的场景下（比如千万级）。

8.1.7. 如何配置时间格式

在Logtail采集配置中设置时间格式，请遵循以下注意事项。

- 日志服务的时间戳只支持到秒，所以时间格式只需配置到秒，无需配置毫秒、微秒等信息。
- 只需配置time字段中的时间部分即可，其他内容无需配置。

常见日志格式配置示例如下：

```
自定义1 2017-12-11 15:05:07
%Y-%m-%d %H:%M:%S
自定义2 [2017-12-11 15:05:07.012]
[%Y-%m-%d %H:%M:%S
RFC822      02 Jan 06 15:04 MST
%d %b %y %H:%M
RFC822Z     02 Jan 06 15:04 -0700
%d %b %y %H:%M
RFC850      Monday, 02-Jan-06 15:04:05 MST
%A, %d-%b-%y %H:%M:%S
RFC1123     Mon, 02 Jan 2006 15:04:05 MST
%A, %d-%b-%y %H:%M:%S
RFC3339     2006-01-02T15:04:05Z07:00
%Y-%m-%dT%H:%M:%S
RFC3339Nano 2006-01-02T15:04:05.999999999Z07:00
%Y-%m-%dT%H:%M:%S
```

8.1.8. 如何在日志样例中设置不可见字符

本文档主要介绍如何在日志样例中填写不可见字符。

背景信息

在使用分隔符方式进行数据采集时，日志服务支持将分隔符和引用符设置为不可见字符。不可见字符是ASCII码中编号为1~31及127的字符，指定分隔符和引用符为不可见字符时，您需要查找不可见字符在ASCII码中对应的十六进制数，输入的格式为 `0x不可见字符在ASCII码中对应的十六进制数` 。假设日志样例为 `123456780` ，本文以分隔符为0x01，引用符为0x02作为示例，在日志样例5和6之间输入一个不可见字符0x01。

操作步骤

1. 登录日志服务控制台。
2. 在浏览器空白处右键选择**检查**。
3. 单击**Console**页签。
4. 输入 `"\x01"` 后回车。
5. 选中并复制返回的结果。
   双引号中间即为不可见字符。

6. 将复制的结果粘贴到日志样例的5和6之间。

   请在**Logtail配置**页签的**日志样例**中配置，详情请参见分隔符日志。

7. 删除5之后及6之前的双引号。
   删除双引号后，日志样例中的不可见字符设置完成。

8.1.9. 排查容器日志采集异常

当您使用Logtail采集容器（标准容器、Kubernetes）的日志时，如果采集状态异常，可以根据本文进行排查问题、检查运行状态等运维操作。

其他运维操作

- 登录Logtail容器
- 查看Logtail的运行日志
- Logtail的容器标准输出（stdout）
- 查看Kubernetes集群中日志相关组件状态
- 查看Logtail的版本号信息、IP地址、启动时间

排查机器组心跳异常

您可以通过检查机器组心跳状态的方式判断容器上的Logtail是否已正确安装。

1. 查看机器组心跳状态。
   i. 登录日志服务控制台。
   ii. 单击目标Project名称。
   iii. 在左侧导航栏中单击**机器组**。
   iv. 单击需要查看状态的机器组名称。
   在**机器组状态**中记录心跳状态为**OK**的节点数。
2. 检查集群中Worker节点数。
   在Kubernetes 集群中，执行命令 `kubectl get node | grep -v master` ，查看集群中Worker节点数。

```
$kubectl get node | grep -v master
NAME                                STATUS    ROLES     AGE       VERSION
cn-hangzhou.i-bp17enxc2us3624wexh2  Ready     <none>    238d      v1.10.4
cn-hangzhou.i-bp1ad2b02jtqd1shi2ut  Ready     <none>    220d      v1.10.4
```

3. 对比心跳状态为**OK**的节点数是否和集群中Worker节点数一致。根据对比结果选择排查方式。
   - 机器组中所有节点的心跳状态均为**Failed**。
     - 若您使用标准Docker日志采集流程，请参见参数说明检查*${your_region_name}*、*${your_aliyun_user_id}*和*${your_machine_group_user_defined_id}*是否填写正确。
     - 若您使用Kubernetes日志采集流程，请提交工单至日志服务。
     - 若您使用自建Kubernetes安装方式，请参见参数说明检查*{your-project-suffix}*、*{regionId}*、*{aliuid}*、*{access-key-id}*和*{access-key-secret}*是否已正确填写，若填写错误请执行命令 `helm del --purge alibaba-log-controller` 删除安装包，并重新安装。
   - 机器组心跳数少的节点于集群中Worker节点数。
     a. 判断是否使用yaml文件手动部署DaemonSet。

        执行命令 `kubectl get po -n kube-system -l k8s-app=logtail` ，若有返回结果，表示您之前使用yaml文件手动部署了DaemonSet。
     b. 下载最新版本的DaemonSet模板。
     c. 将其中的*${your_region_name}*、*${your_aliyun_user_id}*、*${your_machine_group_name}*替换为您的真实内容。
     d. 执行命令 `kubectl apply -f ./logtail-daemonset.yaml` 更新DaemonSet yaml文件。

     其他情况，请提交工单至日志服务。

排查容器日志采集异常

如果您在控制台的**消费预览**或**查询分析**页面未查看到日志数据，说明日志服务并未采集到您的容器日志数据。请确认容器状态后，执行以下检查。

1. 确认机器组状态是否正常。
2. 检查采集配置标识是否正确。
   检查配置中**IncludeLabel**、**ExcludeLabel**、**IncludeEnv**、**ExcludeEnv**是否和您需要采集的容器配置匹配。

   > **说明** 其中的 `Label` 为容器Label（**docker inspect**中的Label信息），并不是Kubernetes中定义的Label。您可以通过将**IncludeLabel**、**ExcludeLabel**、**IncludeEnv**和**ExcludeEnv**配置临时去除，查看是否可以正常采集到日志，若可以采集则可以确定为标记配置问题。

3. 检查其他注意事项。
   若您配置采集容器内文件，请注意：
   - 若下发采集配置后文件没有修改事件，Logtail则不采集。
   - 采集容器日志文件，只支持采集容器默认存储或挂载到本地的文件，暂不支持其他存储方式。

登录Logtail容器

- 普通Docker
  i. 在宿主机执行 `docker ps | grep logtail` 搜索Logtail容器。
  ii. 执行 `docker exec -it ****** bash` 登录。

```
$docker ps | grep logtail
223fbd3ed2a6e        registry.cn-hangzhou.aliyuncs.com/log-service/logtail
"/usr/local/ilogta..."   8 days ago          Up 8 days                                  logtail-iba
$docker exec -it 223fbd3ed2a6e bash
```

- Kubernetes
  i. 执行 `kubectl get po -n kube-system | grep logtail` 搜索Logtail的Pod。
  ii. 执行 `kubectl exec -it -n kube-system ****** bash` 登录Pod。

```
$kubectl get po -n kube-system | grep logtail
logtail-ds-g5wgd                                        1/1       Running    0          8d
logtail-ds-slpn8                                        1/1       Running    0          8d
$kubectl exec -it -n kube-system logtail-ds-g5wgd bash
```

查看Logtail的运行日志

Logtail日志存储在Logtail容器中的/usr/local/ilogtail/目录中，文件名为ilogtail.LOG和logtail_plugin.LOG。

1. 登录Logtail容器。
2. 打开Logtail容器目录/usr/local/ilogtail/。

```
cd /usr/local/ilogtail
```

3. 查看文件ilogtail.LOG和logtail_plugin.LOG。

```
cat ilogtail.LOG
cat logtail_plugin.LOG
```

Logtail的容器标准输出（stdout）

容器stdout并不具备参考意义，请忽略以下标准输出（stdout）：

```
start umount useless mount points, /shm$|/merged$|/mqueue$
umount:
/logtail_host/var/lib/docker/overlay2/3fd0043af174cb0273c3c7869500fbe2bdb95d13b1e110172ef57fe840c82155/mer
: must be superuser to unmount
umount:
/logtail_host/var/lib/docker/overlay2/d5b10aa19399992755de1f85d25009528daa749c1bf8c16edff44beab6e69718/mer
: must be superuser to unmount
umount:
/logtail_host/var/lib/docker/overlay2/5c3125daddacedec29df72ad0c52fac800cd56c6e880dc4e8a640b1e16c22dbe/mer
: must be superuser to unmount
......
xargs: umount: exited with status 255; aborting
umount done
start logtail
ilogtail is running
logtail status:
ilogtail is running
```

查看Kubernetes集群中日志相关组件状态

执行命令 `helm status alibaba-log-controller` 可以查看Kubernetes集群中日志相关组件状态。

查看Logtail的版本号信息、IP地址、启动时间

相关信息存储在Logtail容器中的/usr/local/ilogtail/目录中的app_info.json，示例如下：

```
kubectl exec logtail-ds-gb92k -n kube-system cat /usr/local/ilogtail/app_info.json
{
   "UUID" : "",
   "hostname" : "logtail-gb92k",
   "instance_id" : "0EBB2B0E-0A3B-11E8-B0CE-0A58AC140402_10.10.10.10_1517810940",
   "ip" : "10.10.10.10",
   "logtail_version" : "0.16.2",
   "os" : "Linux; 3.10.0-693.2.2.el7.x86_64; #1 SMP Tue Sep 12 22:26:13 UTC 2017; x86_64",
   "update_time" : "2018-02-05 06:09:01"
}
```

8.2. 日志查询

8.2.1. 日志查询常见问题

本文档为您介绍日志服务查询时的常见问题及处理方法。

如何在查询时判断日志的来源机器？

如果通过Logtail采集日志时，机器组类型为IP地址机器组，机器组中的机器通过内网IP区分。在查询时，可以通过hostname和自定义配置的工作IP来判断日志的来源机器。

例如，统计日志中不同hostname出现的次数。

> **说明** 需要提前开启日志索引功能并给__tag__:__hostname__字段开启统计功能。

```
* | select "__tag__:__hostname__" , count(1) as count group by "__tag__:__hostname__"
```

如何在日志数据中搜索IP地址？

在日志数据中搜索IP地址，支持全部匹配的方式检索。您可以直接在日志数据中直接搜索指定IP地址相关的日志信息，例如包含指定IP地址、过滤指定IP地址等。但是目前尚不支持部分匹配的方式检索，即不能直接搜索IP地址的一部分，因为小数点不是日志服务默认的分词项。如果需要的话，建议自行过滤，例如用SDK先下载数据，然后在代码里用正则或者用string.indexof等方法判断。

例如，在日志服务的Project中搜索条件如下。

```
not ip:121.42.0 not status:200 not 360jk not DNSPod-Monitor not status:302 not jiankongbao
        not 301 and status:403
```

搜索结果中仍会出现121.42.0网段地址。因为日志服务会认为121.42.0.x是一个词，所以只有搜121.42.0.x能搜到结果，而121.42.0的话不会搜到这个结果，同理加上not也就不会过滤该地址。

如何在日志中搜索包含空格的关键字？

搜索包含空格的关键字时，如果直接搜索，则会得到包含空格左侧关键字或右侧关键字的所有日志。建议您在查询的包含空格的关键字时，把关键字用双引号包裹起来，将引号中的内容作为一个关键字进行搜索，搜索结果就是符合条件的日志内容。

例如，在以下日志中搜索包含关键字 `POS version` 的日志。

```
post():351]:&amp;nbsp;device_id:&amp;nbsp;BTAddr&amp;nbsp;:&amp;nbsp;B6:xF:xx:65:xx:A1&amp;nbsp;IMEI&amp;nbsp;:&amp;nbsp;35847xx22xx81
nbsp;WifiAddr&amp;nbsp;:&amp;nbsp;4c:xx:0e:xx:4e:xx&amp;nbsp;|&amp;nbsp;user_id:&amp;nbsp;bb07263xxd2axx43xx9exxea26e39e5f&amp;nbs
OS&amp;nbsp;version:903
```

如果直接搜索 `POS version` ，则会得到包含 `POS` 或者 `version` 的所有日志，不符合搜索要求。如果搜索 `"POS version"` ，则会得到包含关键字 `POS version` 的所有日志。

如何完成双重条件检索？

双重条件检索时，只需同时输入两个语句即可。

例如，需要在Logstore中搜索数据状态不是OK或者Unknown的日志。直接搜索 `not OK not Unknown` 即可得到符合条件的日志。

日志服务提供哪些渠道查询采集的日志？

日志服务提供了三种方式查询日志：

- 通过日志服务控制台查询。
- 通过SDK查询。
- 通过Restful API查询。

8.2.2. 查询不到日志数据

在使用日志服务产品的日志查询功能时，如果查询不到日志数据，请按照以下原因进行排查。

未成功采集日志数据

如果并未成功采集日志数据到日志服务，则无法查询到目标日志。请在预览界面查看是否有日志数据。

如果有日志数据，说明日志数据已成功采集到日志服务中，建议您排查其他原因。

如果没有日志数据，可能是以下原因造成，请进一步排查。

- 日志源没有生产日志数据

  日志源没有日志产生的情况下，没有日志可以投递到日志服务。请检查您的日志源。

- Logtail无心跳

  请在**机器组状态**页面中查看机器是否有心跳。没有心跳请参见Logtail 机器无心跳。

- 监控文件没有实时写入

  如果监控文件有实时写入，您可以打开/usr/local/ilogtail/ilogtail.LOG查看报错信息。常见错误如下：

- parse delimiter log fail：分割符收集日志出错。
- parse regex log fail：正则收集日志出错。

分词设置错误

查看已设置的分词符，检验根据分词符对日志内容进行分割后，是否刚好得到关键字。例如分割符为默认的 `,;=()[]{}?@&<>/:'` 那么用户的日志里如果是有abc”defg,hij会被分割成abc”defg和hij两部分，用abc就搜不到这条日志。

同时支持模糊查询，具体查询语法，请参见查询语法。

> **说明**
> - 为了节约您的索引费用，日志服务进行了索引优化，配置了字段索引的Key，不进行全文索引。例如，日志中有名为message的key，并且配置了字段索引，加了空格做分词（加空格做分词，请把空格加到分词字符串的中间）。”message: this is a test message“可以用 key:value 的格式 message:this 查到，但是直接查this查询不到，因为配置了字段索引的key，不进行全文索引。
> - 创建索引或者对索引做任何更改，只对新进的数据有效，旧数据一律无效。
>
> 您可以点开索引属性，检查已设置的分词是否符合要求。

其他原因

如果日志有产生，可以先在查询处修改查询的时间范围。另外由于日志预览的功能数据是实时的，但是查询的功能是有最多1分钟的延迟的，所以用户可以在日志产生后等1分钟再查。

8.2.3. 日志消费与查询区别

日志服务提供日志消费和查询的功能，都属于对日志的读操作，区别在于消费提供收集和分发通道，查询提供日志查询功能。

日志服务提供了两项功能都和“读”有关：

日志收集与消费（LogHub）：提供公共的日志收集、分发通道。全量数据顺序（FIFO）读写，提供类似Kafka的功能。

- 每个LogStore有一个或多个Shard，数据写入时，随机落到某一个shard中
- 可以从指定shard中，按照日志写入shard的顺序批量读取日志
- 可以根据server端接收日志的时间，设置批量拉取shard日志的起始位置（cursor）

日志查询（Search/Analytics）：在LogHub基础上提供海量日志查询+分析功能，根据条件进行日志查询与统计

- 通过查询条件查找符合要求的数据
- 支持关键词 AND、NOT、OR的布尔组合和结果SQL统计
- 数据查询不区分shard

两者区别：

功能	日志查询（LogSearch）	日志收集与消费（LogHub）
关键词查找	支持	不支持
小量数据读取	快	快
全量数据读取	慢（100条日志100ms，不建议通过该方式读取数据）	快 （1MB日志10ms，推荐方式）
读取是否区分topic	区分	不区分，只以shard作为标识
读取是否区分shard	不区分，查询所有shard	区分，单次读取需要指定shard
费用	较高	低
适用场景	监控、问题调查与分析等场景	流式计算、批量处理等全量处理场景

8.2.4. 日志查询分析常见报错

日志查询分析的常见报错如下。

line 1:44: Column ‘my_key_field’ cannot be resolved;please add the column in the index attribute

- 报错原因

  `my_key_field` 这个Key不存在，所以您在query中无法引用该Key。

- 解决方案

  i. 登录日志服务控制台。

  ii. 找到目标Project。

  iii. 单击日志库名称后的，选择**查询分析**。

  iv. 在查询分析页面，选择**查询分析属性 > 设置**，添加该字段为字段索引，同时打开统计功能。

Column ‘xxxxline‘ not in GROUP BY clause;please add the column in the index attribute

- 报错原因

  您在查询中使用了GROUP BY语法，但是在Select中引用了一个非agg字段，该字段没有出现在GROUP BY中。例如 `select key1, avg(latency) group by key2` ，key1没有出现在GROUP BY中。

- 解决方案

  正确语法是 `select key1,avg(latency) group by key1,key2` 。

sql query must follow search query,please read syntax doc

- 报错原因

  没有指定filter条件，例如 `select ip,count(*) group by ip` 。

- 解决方案

  正确的写法为 `*|select ip,count(*) group by ip` 。

please read syntax document,and make sure all related fields are indexed. error after select .error detail:line 1:10: identifiers must not start with a digit; surround the identifier with double quotes

- 报错原因

  SQL中引用到的列名、变量名等以数字开头，不符合规范。

- 解决方案

  建议更改该名称，以字母开头。

please read syntax document,and make sure all related fields are indexed. error after select .error detail:line 1:9: extraneous input ‘’ expecting

- 报错原因

  有单词拼写错误。

- 解决方案

  请根据报错中指出的错误位置，修改至正确。

key (category) is not config as key value config,if symbol : is in your log,please wrap : with quotation mark "

- 报错原因

  category字段未配置字段索引，不能在分析语句中使用。

- 解决方案

  请在查询分析属性中设置该字段的索引。详细说明请参见开启并配置索引。

Query exceeded max memory size of 3GB

- 报错原因

  当前query使用服务端内存超过3GB。通常原因为使用GROUP BY语法去重后value太多。

- 解决方案

  请优化GROUP BY的查询语法，减少GROUP BY的Key的个数。

ErrorType:ColumnNotExists.ErrorPosition,line:0,column:1.ErrorMessage:line 1:123: Column '__raw_log__' cannot be resolved; it seems __raw_log__ is wrapper by ";if __raw_log__ is a string ,not a key field, please use '__raw_log__'

- 报错原因

  `__raw_log__` 这个Key不存在，所以您在query中无法引用该Key。

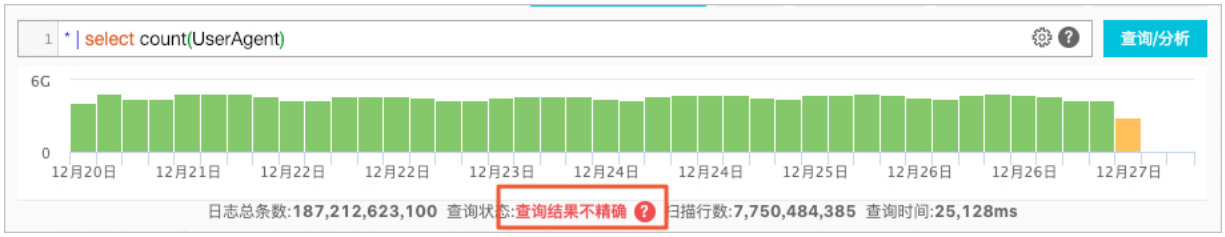
- 解决方案

  在查询页面，右上角查询分析属性里，添加该字段为字段索引，同时打开统计功能。

8.2.5. 查询不精确有哪些原因

本文档为您介绍在日志服务控制台查询数据不精确时的原因及方案。

查询分析日志时，控制台可能会提示**查询不精确**，表示日志服务在查询分析时未能扫描全部日志数据，返回的查询分析结果不是基于全部日志数据的精确结果。



查询不精确一般由以下原因造成：

查询时间范围太大

- 错误原因

  时间范围较大时，例如3个月或1年，一次查询无法完整扫描这个时间段的所有日志数据。

- 解决方案

  请缩小查询的时间范围，分多次查询。

查询条件过于复杂

- 错误原因

  查询条件过于复杂、或者查询条件中包含了一些高频词汇时，日志服务不能一次性读取结果。

- 解决方案

  请缩小查询的范围，分多次查询。

SQL计算要读取的数据量太多

- 错误原因

  SQL计算要读取的数据量太多时，容易造成查询不精确。例如读取多个字符串列时，每个Shard限制读取1个G数据，超过后会返回不精确结果。

- 解决方案

  请缩小查询的范围，分多次查询。

8.2.6. 如何为历史日志配置索引

日志服务不支持直接为历史日志配置索引，您可以通过Dataworks将日志重新写入另一个Logstore，或使用CLI实现。

索引配置仅对设置索引后采集到的日志数据有效，设置索引之前的历史数据不能被查询和分析。如果要对这部分历史数据设置索引，可以通过以下方式完成：

- 通过DataWorks将数据重新写入，并设置索引。

  为新的Logstore设置索引后，通过阿里云Dataworks产品，将历史日志从Logstore中导出，然后导入到新的Logstore即可。重新写入的历史数据就可以被查询和分析了。

- 通过日志服务CLI将数据重新写入，并设置索引，详情请参见日志服务CLI。

> **说明** 以上两种方式均为复制数据并导入的方式，不会修改或删除您已经采集到的历史日志。

8.3. 告警

8.3.1. 告警常见问题

本文档为您介绍在日志服务中设置告警时的常见问题及处理方法。

如何将错误日志的原始日志展示在告警的通知内容中

- 问题

  假设在过去的5分钟内，错误日志达到5条以上并且触发了告警，那么如何将错误日志的原始日志展示在告警的通知内容中呢？

- 处理方法
  - 关联的查询语句
    - 编号0：`level：ERROR`
    - 编号1：`level：ERROR | select COUNT(*) as count`
  - 触发条件：`$1.count > 5`
  - 通知内容：`${results[0].rawresults}`

- 配置示例