日志服务

操作指南

产品文档

腾讯云

文档目录

操作指南
资源管理
日志集

最近更新时间：2022-04-21 14:32:06

概述

日志服务（Cloud Log Service，CLS）以日志主题及日志集的方式管理其上存储的日志数据，日志主题（Topic）是日志数据进行采集、存储、检索和分析的基本单元，日志集则是对日志主题的分类，方便用户管理日志主题。

本文介绍如何通过控制台管理日志集，在实际操作前，建议您阅读 日志主题与日志集，进一步了解相关概念及针对实际场景的使用建议。

前提条件

已登录 日志服务控制台。

操作步骤

创建日志集

1. 在左侧导航栏中，单击**日志主题**。
2. 在日志主题管理页面，选择地域，单击**管理日志集**。

3. 在右侧展开的日志集管理页面中，单击**创建日志集**。

4. 在弹出的创建日志集窗口中，填写日志集名称及标签。

5. 单击**确定**，完成日志集创建。

查看日志集

1. 在左侧导航栏中，单击**日志主题**。
2. 在日志主题管理页面，选择地域，单击**管理日志集**。

3. 在右侧展开的日志集管理页面中，即可查看当前地域下的日志集。

编辑日志集

1. 在左侧导航栏中，单击**日志主题**。

2. 在日志主题管理页面，选择地域，单击**管理日志集**。

3. 在右侧展开的日志集管理页面中，找到您需要编辑的日志集ID/名称，单击**编辑**。

4. 在弹出的编辑日志集窗口中，修改日志集名称。

5. 单击**确定**，保存编辑。

删除日志集

1. 在左侧导航栏中，单击**日志主题**。
2. 在日志主题管理页面，选择地域，单击**管理日志集**。
3. 在右侧展开的日志集管理页面中，找到您需要删除的日志集ID/名称，单击**删除**。

> ⚠ 注意：
> 仅在“**日志主题数量**”为0时，才可删除日志集。若日志集中的日志主题未清空，则无法对日志集进行删除。

4. 在弹出提示框中，单击**确定**，即可删除当前日志集。

日志主题
管理日志主题

最近更新时间：2022-06-14 10:25:43

操作场景

日志服务（Cloud Log Service，CLS）以日志主题及日志集的方式管理其上存储的日志数据，日志主题（Topic）是日志数据进行采集、存储、检索和分析的基本单元，日志集则是对日志主题的分类，方便用户管理日志主题。

本文介绍如何通过控制台管理日志主题，在实际操作前，建议您阅读 日志主题与日志集，进一步了解相关概念及针对实际场景的使用建议。

前提条件

已登录 日志服务控制台。

操作步骤

创建日志主题

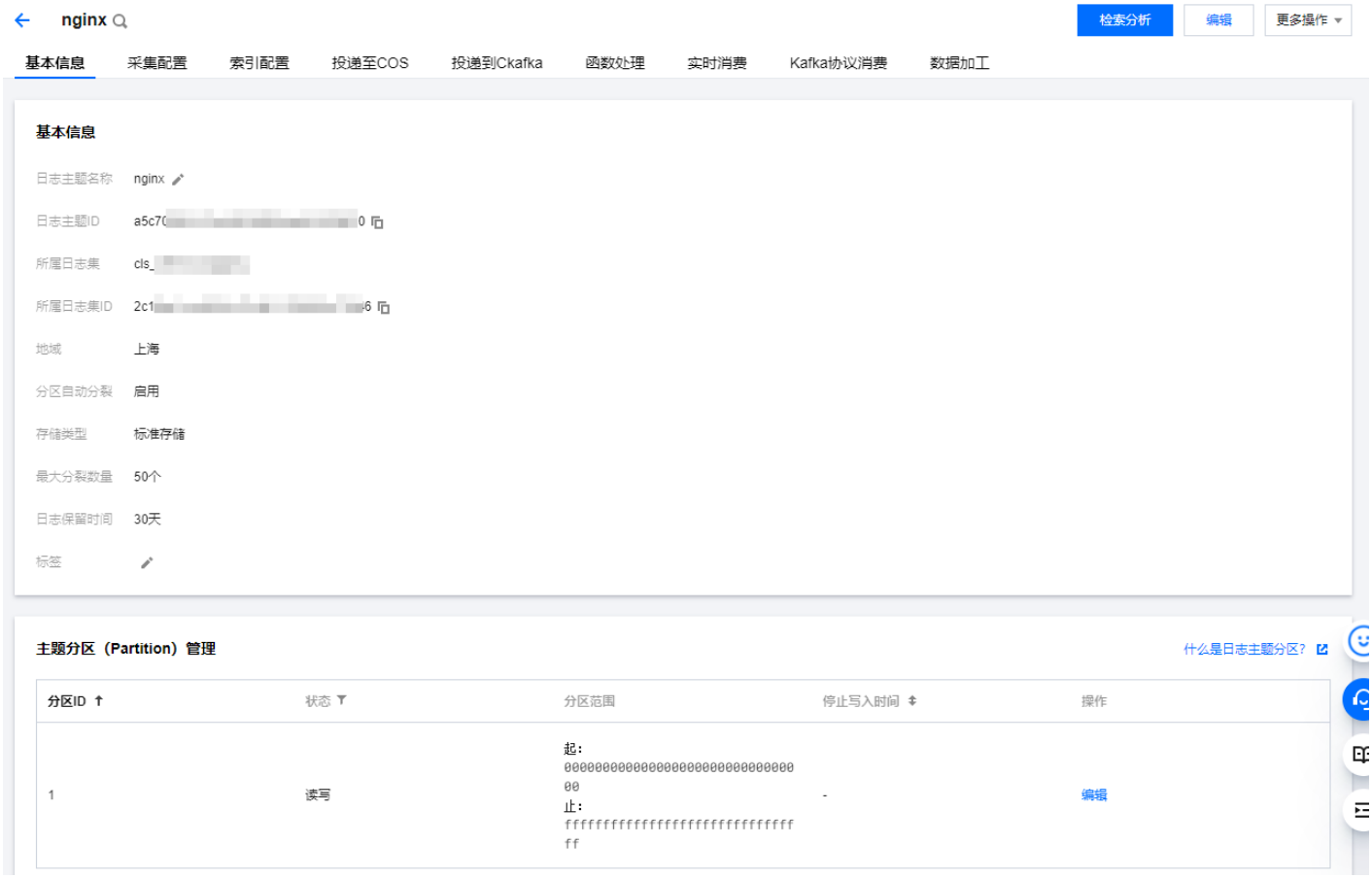
1. 在左侧导航栏中，单击**日志主题**。
2. 在日志主题管理页面，选择日志主题的地域，单击**创建日志主题**。
3. 在弹出的窗口中，填写如下相关信息：

- 日志主题名称：例如 nginx。
- 存储类型：默认为标准存储，关于存储类型请参见 存储类型概述。
- 日志保存时间：默认为30天，可自定义设置日志存储时间，日志过期后将自动清除。使用 LogListener 时，如果日志时间未使用采集时间，而将日志内容中的某个时间字段作为日志时间，将以该时间为准决定日志是否过期。
- 日志集操作：
  - 选择现有日志集：在日志集下拉框中选择目标日志集。
  - 创建日志集：
    日志集名称：例如 cls_test。
- 日志主题标签：为当前创建的日志主题设定标签，便于从不同维度对资源分类管理。
- 高级设置：
  - 分区数量： 默认新建1个分区，关于主题分区请参见 主题分区。
  - 分区自动分裂：默认开启。

- 最大分裂数量：可自定义分裂数量，最大分裂数量为50个。
- 日志集标签：日志集操作为**创建日志集**时显示该功能，可为新创建的日志集设定标签。

4. 单击**确定**，即可创建日志主题。

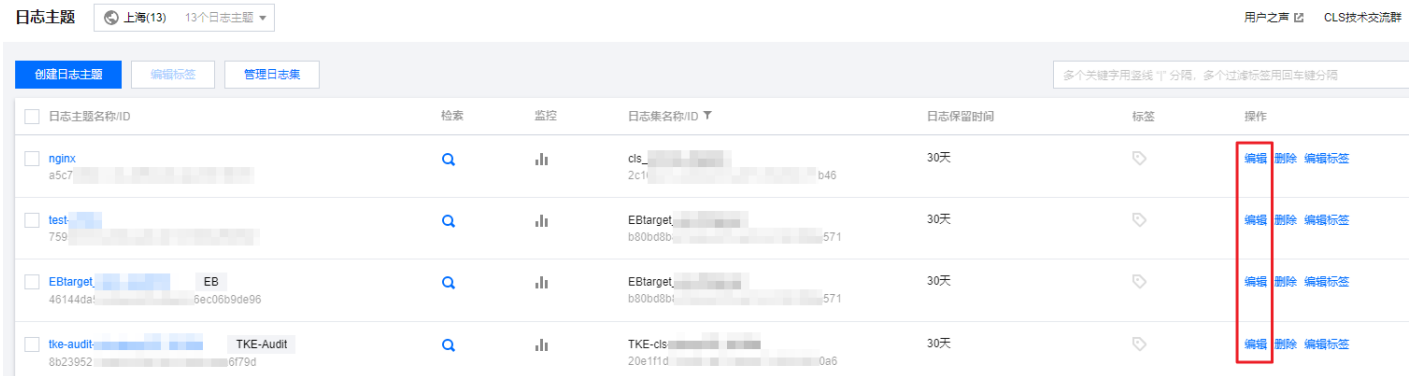
日志主题新增成功后，可在日志管理页面单击日志主题ID/名称，查看日志主题的详情信息。



> **说明：**
>
> - 若收集云服务器或其他云产品日志，日志服务地域建议选择相同地域。
> - 日志主题创建后，无法修改所属的地域及日志集。
> - 日志保存时间可为1 – 3600天，或永久保存。

编辑日志主题

1. 在左侧导航栏中，单击**日志主题**。
2. 在日志主题管理页面，找到您需要编辑的日志主题ID/名称，单击**编辑**。

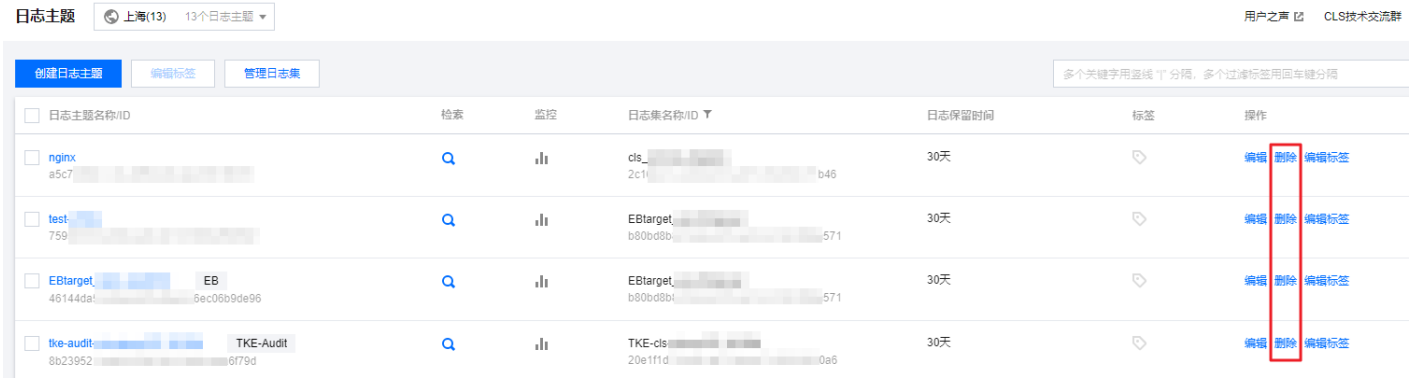


3. 在弹出的窗口中，修改日志主题基本信息。

4. 单击**确定**，日志主题信息更新完成。

删除日志主题

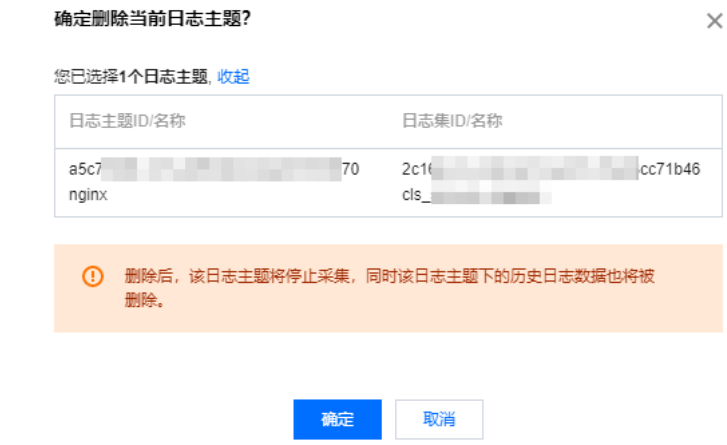
1. 在左侧导航栏中，单击**日志主题**。
2. 在日志主题管理页面，找到您需要删除的日志主题ID/名称，单击**删除**。



> **说明：**
> 页面左上角可切换至不同地域分别删除其内的日志主题。如需删除所有的日志主题，建议使用具备全部资源权限的账号进行操作，否则部分日志主题可能由于无权限查看而无法全部删除。

3. 在弹出的窗口中，单击**确定**，即可删除当前日志主题。

> **注意：**
> 日志主题一旦删除，主题配置和日志数据不可恢复，请谨慎操作。


确定删除当前日志主题?
您已选择1个日志主题, 收起
日志主题ID/名称
日志集ID/名称
a5c7 70
nginx
2c1 cc71b46
cls_
删除后，该日志主题将停止采集，同时该日志主题下的历史日志数据也将被删除。
确定
取消

监控流量统计

最近更新时间：2021-11-12 15:06:58

流量

指标中文名	指标英文名	指标含义	单位	维度
写流量	Write Traffic	上传日志时会产生的流量。	MB	日志主题 ID
索引流量	Index Traffic	开启索引后会产生的流量。	MB	日志主题 ID
内网读流量	Private Network Read Traffic	通过内网下载、消费、投递日志会产生的内网读流量。	MB	日志主题 ID
外网读流量	Public Network Read Traffic	通过外网下载、消费、投递日志会产生的外网读流量。	MB	日志主题 ID
读流量总量	Read Traffic	内、外网读流量总和。	MB	日志主题 ID

存储量

指标中文名	指标英文名	指标含义	单位	维度
日志存储量	Log Storage Capacity	日志数据所占用的存储空间大小。	MB	日志主题 ID
索引存储量	Index Storage Capacity	索引数据所占用的存储空间大小。	MB	日志主题 ID
存储总量	Storage Capacity	日志存储量和索引存储量总和。	MB	日志主题 ID

服务请求数

指标中文名	指标英文名	指标含义	单位	维度
服务请求数	Service Requests	用户使用 Loglistener、API、SDK 调用日志服务接口的请求次数，统计包括上传下载、创建删除、检索分析等读写请求。	COUNT	日志主题 ID

说明：

以上监控指标已上报至云监控，对应的详细指标及参数请参见 日志服务监控指标。

主题分区
分裂主题分区

最近更新时间：2022-04-14 15:55:03

本文介绍如何在日志服务控制台上分裂主题分区。日志服务提供了**“自动分裂”**和**“手动分裂”**两种操作模式。为了减小因流量突发而受到读写限制，建议您 **开启自动分裂**。

操作场景

在创建日志主题时，会设置一个初始的分区数量，后续您可根据实际情况，使用分裂分区或者合并分区的功能来扩缩分区数。由于单个主题分区限制写请求（最高500次/S）和写流量（最高5MB/S），如果您的服务请求超过单个分区上限，可通过分裂分区增大日志主题的吞吐，避免写入失败。

操作步骤

自动分裂

> 说明：
> 开启自动分裂功能后，如果主题分区持续触达了写请求或者写流量的阈值，日志服务会根据实际写入情况，自动分裂至合理的分区数（最大分裂数量为50个）。
> 如果日志主题的分区数已达到所设置的最大值，日志服务将不再触发自动分裂，所超限的部分将被拒绝，并返回 请求超限错误码。

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**，进入日志主题列表页面。
3. 找到您需要开启自动分裂的日志主题，单击**操作**列的**编辑**。
4. 在弹出的对话框中，将分区自动分裂设置为  ，并设置“最大分裂数量”（建议设为最大值50）。
5. 单击**确定**，完成操作。

手动分裂

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**，进入日志主题列表页面。
3. 找到您需要开启手动分裂的日志主题，单击该日志主题ID/名称，进入日志主题基本信息页面。
4. 在“主题分区（Partition）管理”栏中，找到需要分裂的目标主题分区，单击**编辑**。如下图所示：

5. 在弹出的对话框中，将“操作”选择为**分裂**，设置“分区个数”。如下图所示：

> 说明
> 每个分区均有区间范围，且每个区间范围均为左闭右开区间，默认平均分裂。

6. **单击确定，完成分裂操作。**

合并主题分区

最近更新时间：2022-04-14 15:55:08

操作场景

本文介绍如何在日志服务控制台上合并主题分区。通过对相邻区间范围的分区进行合并，可以减少主题分区的数量，避免资源浪费。

注意事项

- 每个日志主题至少有一个分区。若您只有1个分区，将不能进行合并。
- 每次只能合并指定主题分区相邻的一个合并对象。

操作步骤

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**，进入日志主题列表页面。
3. 找到您需要合并主题分区的日志主题，单击该日志主题ID/名称，进入日志主题基本信息页面。
4. 在“主题分区（Partition）管理”栏中，找到需要合并的目标主题分区，单击**编辑**。如下图所示：

5. 在弹出的对话框中，将“操作”选择为**合并**，并选择需要合并的合并对象。如下图所示：

6. 单击**确定**，完成合并操作。

机器组

最近更新时间：2022-01-05 15:23:36

操作场景

机器组是腾讯云日志服务（Cloud Log Service，CLS）中 LogListener 所采日志的服务器对象，通过机器组的方式来配置 LogListener 采集日志的服务器。一般而言，根据不同业务场景来划分不同的机器组，可以方便您管理日志服务。

操作步骤

通过配置机器组 IP 地址创建机器组

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击【机器组管理】。
3. 选择您日志服务所在的地域，例如广州，单击【新建机器组】。
4. 在弹出的窗口中，设置如下信息。

- 机器组名称：例如 cls_test。
- 配置模式：选择**配置机器组IP地址**。
- IP地址：输入机器的 IP 地址。

> ⚠ 注意：
>
> - 每行填写一个 IP 地址，暂不支持 Windows 系统机器。
> - 同地域内，腾讯云服务器填写内网 IP 地址即可，分行隔开。
> - 若机器组需跨地域上报，填写外网 IP 地址。

- LogListener自动升级：默认关闭，请根据实际需求进行配置。
- LogListener服务日志：默认开启 Loglistener 服务日志，记录 Loglistener 运行状态、采集状况等重要日志，详情可参见 LogListener 服务日志。

5. 单击【确定】，即可完成创建。

通过配置机器标识创建机器组

如果您的机器 IP 地址经常变动，利用机器 IP 配置机器组操作繁琐，每次 IP 变更将需要修改对应的机器组配置。
CLS 支持利用机器标识动态配置机器组，您只需要在 Loglistener 的配置信息中填入机器标识，CLS 即可识别并自动将机器添加至机器组。

> ⚠ 注意：
>
> 通过配置机器标识创建机器组仅支持 LogListener 2.3.0 及以上版本，低版本的 LogListener 需手动更新。

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击【机器组管理】。
3. 选择您日志服务所在的地域，例如广州，单击【新建机器组】。

4. 在弹出的窗口中，设置如下信息。

- 机器组名称：例如 cls_test。
- 配置模式：选择**配置机器标识**。
- 机器标识：输入机器的标识。

> ⚠ 注意：
> 每行填写一个标识名称，暂不支持识别 Windows 系统的机器。

- LogListener自动升级：建议开启 LogListener 自动升级，详情可参见 LogListener 升级指南。
- LogListener服务日志：默认开启 LogListener 服务日志，记录 LogListener 运行状态、采集状况等重要日志，详情可参见 LogListener 服务日志。

5. 单击【确定】。
6. 登录进行机器标识的目标机器，并执行如下命令，打开 Loglistener 安装目录下的/etc/loglistener.conf文件。
   此处安装目录以/user/local为例：

```
vi /usr/local/loglistener-2.3.0/etc/loglistener.conf
```

7. 按 i，进入编辑模式。

8. 找到 group_label 参数，填写您自定义的机器标识（多个标识以英文逗号分割）。

```
proxy_host = ap-guangzhou.
proxy_port =
secret_id =
secret_key =
group_ip =
# define Loglistener by labels, separated by ,
group_label = nginx_access
instance_id = loglistener-2059
pos_file = data/pos.dat
# Max file break points. Warn: modify this value, history breaskpoints will lost.
max_file_breakpoints = 8192
mmap_version_file = data/mmap_version_file.dat
file_cache = 0
# The concurrency of the requests.
max_connection = 10
# Max memory loglistener would use.
max_mem = 2097152000
# Max bytes send per second, 0 for unlimited.
max_send_rate = 0
# Max scanning depth corresponding to ** in wildpath
max_depth = 10
# Compress the upload traffic by LZ4.
request_compression = true
# Replace special charactors to space. ['\0'] => ' '
replace_special_charactors = false
# Logistener do its best to save memorys when true.
memory_tight_mode = false
```

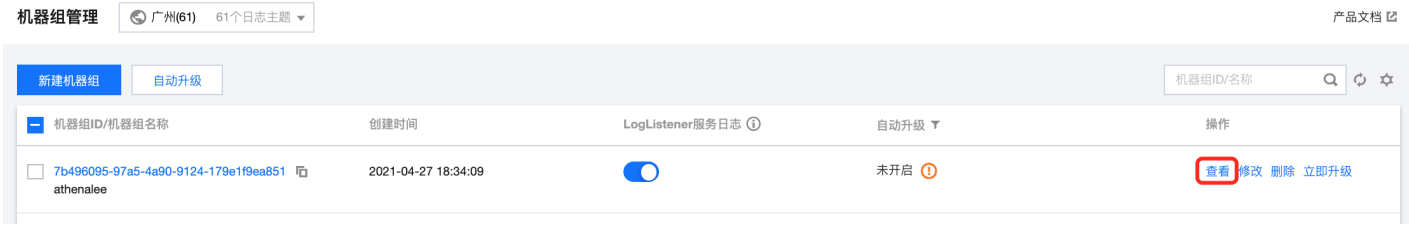
9. 按 **Esc**，退出编辑模式。
0. 输入 **:wq**，按 **Enter**，保存设置。
1. 执行如下命令，重启 LogListener，即可完成创建。

```
/etc/init.d/loglistenerd restart
```

查看机器状态

机器组与日志服务系统之间采用心跳机制保持连接，成功安装过 LogListener 的机器组会定时向日志服务发送心跳。

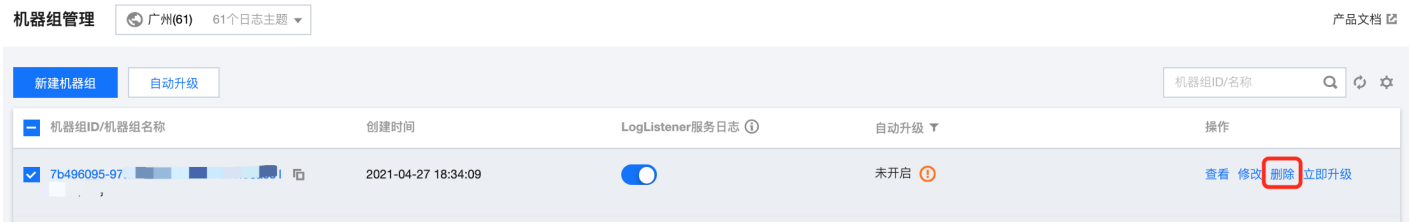
1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击【机器组管理】。
3. 在机器组管理页面，找到需要操作的机器组，单击【查看】。



4. 在弹出的窗口中，即可查看机器组的状态。
   通过查看机器组的状态可识别当前该机器是否工作正常。若状态显示正常，则说明您的服务器可以与腾讯云日志服务正常通信。

删除机器组

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击【机器组管理】。
3. 在机器组管理页面，选择需要删除的机器组，单击【删除】。



4. 在弹出的提示窗口，单击【确认】，完成机器组删除。

> **注意：**
> 机器组一旦删除，所关联的日志主题将无法继续采集日志。

权限管理
基于 CAM 管理权限

最近更新时间：2022-02-17 10:27:04

简介

访问管理（Cloud Access Management，CAM）是腾讯云提供的 Web 服务，主要用于帮助用户安全管理腾讯云账户下资源的访问权限。用户可以通过 CAM 创建、管理和销毁用户（组），并使用身份管理和策略管理控制其他用户使用腾讯云资源的权限，CAM 策略的详细信息及使用请参见 CAM 策略 文档。

主账号可以授权子账号或协作者访问管理权限，访问指定的日志服务资源。

预设权限策略

日志服务预设两条权限策略，可满足最基本的权限管理需求。

- 全读写访问权限（QcloudCLSFullAccess）：具备日志服务所有功能及所有资源的权限，例如创建日志主题、修改索引配置、删除日志主题、检索日志、上传日志等。
- 只读访问权限（QcloudCLSReadOnlyAccess）：仅具备数据查看权限，不能执行新建、编辑或删除类型的操作。

配置方式参见 授权管理 文档。

自定义权限策略

通过自定义权限策略可实现细粒度的权限划分，例如仅允许某个用户查看特定日志主题的数据。

自定义权限策略主要由两部分组成：

- 操作（Action）：用户可进行的操作，例如检索日志、修改索引配置、上传日志、创建告警策略等。
- 资源（Resource）：可操作的资源范围，例如特定的日志主题、仪表盘、数据加工任务等。

日志服务支持授权的资源类型及相关接口参见 可授权的资源类型，配置方式参见 创建自定义策略。

自定义权限策略配置存在一定的复杂度，实际使用过程中可参见 自定义权限策略示例，这些示例能够满足大多数的权限管理需求，也可以基于这些策略示例再进行个性化调整。详细操作方式如下：

1. 主账号（或具备 CAM 管理权限的用户）在 策略 页面，单击**新建自定义策略**。
2. 在弹出的窗口中，单击**按策略语法创建**。
3. 在选择策略模板页面，选择**空白模版**，单击**下一步**。
4. 在编辑策略页面，设置策略名称，输入策略内容，策略内容可从 自定义权限策略示例 中进行复制。
5. 单击**完成**，保存该策略。
   创建完成自定义策略后，可通过 授权管理 将策略关联至用户/用户组，使用户/用户组获得对应的操作权限。

可授权的资源类型

最近更新时间：2022-02-17 10:26:59

简介

日志服务（Cloud Log Service，CLS）包含多种资源类型，部分接口支持按资源为用户配置权限，例如 对指定日志主题具备管理权限。
在访问管理（Cloud Access Management，CAM）中可授权的资源类型如下表，其中“按标签授权”是指该类型资源是否支持通过标签的方式指定用户具备操作权限的资源范围。

资源类型	授权策略中的资源描述方法	按标签授权
日志集	qcs::cls:$region:$account:logset/* qcs::cls:$region:$account:logset/$logsetId	支持
日志主题	qcs::cls:$region:$account:topic/* qcs::cls:$region:$account:topic/$topicId	支持
机器组	qcs::cvm:$region:$account:machinegroup/* qcs::cvm:$region:$account:machinegroup/$machinegroupId	支持
采集配置	qcs::cls:$region:$account:config/* qcs::cls:$region:$account:config/$configId	不支持
仪表盘	qcs::cls:$region:$account:dashboard/* qcs::cls:$region:$account:dashboard/$dashboardId	支持
告警策略	qcs::cls:$region:$account:alarm/* qcs::cls:$region:$account:alarm/$alarmId	不支持
通知渠道组	qcs::cls:$region:$account:alarmNotice/* qcs::cls:$region:$account:alarmNotice/$alarmNoticeId	不支持
数据加工任务	qcs::cls:$region:uin/$account:datatransform/* qcs::cls:$region:uin/$account:datatransform/$TaskId	不支持
投递任务（COS）	qcs::cls:$region:$account:shipper/* qcs::cls:$region:$account:shipper/$shipperId	不支持
其他资源类型（已废弃，仅老版本 API 使用）	仪表盘内的单个图表： qcs::cls:$region:$account:chart/* qcs::cls:$region:$account:chart/$chartId	不支持

其中，$region 、$account 等变量参数需修改为您实际的参数信息。

日志服务支持的所有接口及其对应的资源描述方法，请参见 CAM 的 数据处理与分析 > 日志服务文档。其中，“授权粒度”为“资源级”的接口支持以上述资源类型按资源方式为用户配置权限，“授权粒度”为“接口级”的接口在 CAM 权限策略中对应的资源范围必须为 * 。

实践建议

由于日志服务中不同类型的资源之间具备关联关系，例如日志集包含日志主题，日志主题需应用采集配置至机器组；如果直接按照资源 ID 的方式在 CAM 权限策略中为用户配置权限存在较大的管理难度，容易导致用户在部分接口下出现无权限错误。因此，建议按照如下方式配置CAM权限策略：

- 针对支持按标签授权的资源类型及对应接口，为相关的资源绑定标签，然后通过标签的方式指定用户具备操作权限的资源范围。例如同时为日志主题、日志集及相关的仪表盘绑定标签，然后 对指定标签的日志主题赋予管理权限，对指定标签的日志主题及仪表盘赋予管理权限。这两条策略可以使用户同时具备这三类资源相关接口的操作权限。
- 针对不支持按标签授权的资源类型及对应接口，为简化管理，可直接以*作为 CAM 权限策略中的资源范围，即所有资源。为避免普通用户误操作，可针对普通用户配置只读权限，针对管理用户配置管理权限，例如 管理权限：对所有数据加工任务具备管理权限 和 只读权限：对所有数据加工任务具备只读权限。

更多使用场景，建议参考 自定义权限策略示例。

自定义权限策略示例

最近更新时间：2022-06-22 10:48:29

数据采集相关

使用 Loglistener 采集数据

用户可以使用采集 Agent Loglistener 采集数据，且具备日志上传的能力（本示例展示机器安装 Loglistener 上传日志的最小权限）。

```
{
"version": "2.0",
"statement": [{
"action": [
"cls:pushLog",
"cls:getConfig",
"cls:agentHeartBeat"
],
"resource": "*",
"effect": "allow"
}]
}
```

> 说明：
> 如果您使用的 Loglistener 为2.6.5以前的版本，则需要加上 "cls:listLogset" 权限。

使用 API 上传数据

用户可以通过 API 上传日志到 CLS（本示例展示使用 API 上传日志的最小权限）。

```
{
"version": "2.0",
"statement": [{
"action": [
"cls:UploadLog"
],
"resource": "*",
"effect": "allow"
}]
}
```

使用 Kafka 上传数据

用户可以通过 Kafka 协议上传日志到 CLS（本示例展示使用 Kafka 协议上传日志的最小权限）。

```
{
"version": "2.0",
"statement": [{
"action": [
"cls:RealtimeProducer"
],
"resource": "*",
"effect": "allow"
}]
}
```

管理采集配置及机器组

包括创建/修改/删除采集配置及创建/修改/删除机器组。

- Config 相关接口对应采集配置相关资源。
- MachineGroup 相关接口对应机器组相关资源。
- ConfigExtra 相关的三个接口权限用于管理自建 k8s 上传日志相关的集群配置信息，如不使用自建 k8s 上传日志相关功能可以忽略。

```
{
"version": "2.0",
"statement": [{
"action": [
"cls:DescribeLogsets",
"cls:DescribeTopics",
"cls:CreateConfig",
"cls:CreateConfig",
"cls:DeleteConfig",
"cls:DescribeConfigs",
"cls:ModifyConfig",
"cls:CreateConfigExtra",
"cls:DeleteConfigExtra",
"cls:ModifyConfigExtra",
"cls:CreateMachineGroup",
"cls:DeleteMachineGroup",
"cls:DescribeMachineGroups",
"cls:DeleteConfigFromMachineGroup",
"cls:ApplyConfigToMachineGroup",
"cls:ModifyMachineGroup"
],
"resource": "*",
"effect": "allow"
}
]
}
```

检索分析相关

使用控制台检索日志

管理权限：对所有日志主题具备管理权限

用户可以对所有的日志主题进行检索及管理，包括创建日志主题、删除日志主题和修改索引配置等，不包括采集配置、日志投递和日志加工等。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:CreateLogset",
"cls:CreateTopic",
"cls:CreateExport",
"cls:CreateIndex",
"cls:DeleteLogset",
"cls:DeleteTopic",
"cls:DeleteExport",
"cls:DeleteIndex",
"cls:ModifyLogset",
```

```
"cls:ModifyTopic",
"cls:ModifyIndex",
"cls:MergePartition",
"cls:SplitPartition",
"cls:DescribeLogsets",
"cls:DescribeTopics",
"cls:DescribeExports",
"cls:DescribeIndex",
"cls:DescribeIndexs",
"cls:DescribePartitions",
"cls:SearchLog",
"cls:DescribeLogHistogram",
"cls:DescribeLogContext",
"cls:DescribeLogFastAnalysis",
"cls:DescribeLatestJsonLog",
"cls:DescribeRebuildIndexTasks",
"cls:CreateRebuildIndexTask",
"cls:EstimateRebuildIndexTask",
"cls:CancelRebuildIndexTask"
],
"resource": [
"*"
]
}
]
}
```

管理权限：对指定日志主题具备管理权限

用户能够对指定的日志主题进行检索及管理，包括创建日志主题、删除日志主题和修改索引配置等，不包括采集配置、日志投递和日志加工等。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:CreateLogset",
"cls:CreateTopic",
"cls:CreateExport",
"cls:CreateIndex",
"cls:DeleteLogset",
"cls:DeleteTopic",
"cls:DeleteExport",
"cls:DeleteIndex",
"cls:ModifyLogset",
"cls:ModifyTopic",
"cls:ModifyIndex",
"cls:MergePartition",
"cls:SplitPartition",
"cls:DescribeLogsets",
"cls:DescribeTopics",
"cls:DescribeExports",
"cls:DescribeIndex",
"cls:DescribeIndexs",
"cls:DescribePartitions",
"cls:SearchLog",
```

```
"cls:DescribeLogHistogram",
"cls:DescribeLogContext",
"cls:DescribeLogFastAnalysis",
"cls:DescribeLatestJsonLog",
"cls:DescribeRebuildIndexTasks",
"cls:CreateRebuildIndexTask",
"cls:EstimateRebuildIndexTask",
"cls:CancelRebuildIndexTask"
],
"resource": [
"qcs::cls:ap-guangzhou:100007***827:logset/1c012db7-2cfd-4418-****-7342c7a42516",
"qcs::cls:ap-guangzhou:100007***827:topic/380fe1f1-0c7b-4b0d-****-d514959db1bb"
]
}
]
}
```

管理权限：对指定标签的日志主题具备管理权限

用户可以对包含指定标签的日志主题进行检索及管理，包括创建日志主题、删除日志主题和修改索引配置等，不包括采集配置、日志投递和日志加工等。为日志主题绑定标签时，需同时为其所属的日志集绑定标签。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:CreateLogset",
"cls:CreateTopic",
"cls:CreateExport",
"cls:CreateIndex",
"cls:DeleteLogset",
"cls:DeleteTopic",
"cls:DeleteExport",
"cls:DeleteIndex",
"cls:ModifyLogset",
"cls:ModifyTopic",
"cls:ModifyIndex",
"cls:MergePartition",
"cls:SplitPartition",
"cls:DescribeLogsets",
"cls:DescribeTopics",
"cls:DescribeExports",
"cls:DescribeIndex",
"cls:DescribeIndexs",
"cls:DescribePartitions",
"cls:SearchLog",
"cls:DescribeLogHistogram",
"cls:DescribeLogContext",
"cls:DescribeLogFastAnalysis",
"cls:DescribeLatestJsonLog",
"cls:DescribeRebuildIndexTasks",
"cls:CreateRebuildIndexTask",
"cls:EstimateRebuildIndexTask",
"cls:CancelRebuildIndexTask"
],
```

```
"resource": [
"*"
],
"condition": {
"for_any_value:string_equal": {
"qcs:resource_tag": [
"testCAM&test1"
]
}
}
}
]
}
```

只读权限：对所有日志主题具备只读权限

用户可以对所有的日志主题进行日志检索。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeLogsets",
"cls:DescribeTopics",
"cls:DescribeExports",
"cls:DescribeIndex",
"cls:DescribeIndexs",
"cls:DescribePartitions",
"cls:SearchLog",
"cls:DescribeLogHistogram",
"cls:DescribeLogContext",
"cls:DescribeLogFastAnalysis",
"cls:DescribeLatestJsonLog",
"cls:DescribeRebuildIndexTasks"
],
"resource": [
"*"
]
}
]
}
```

只读权限：对指定日志主题具备只读权限

用户可以对指定的日志主题进行检索。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeLogsets",
"cls:DescribeTopics",
"cls:DescribeExports",
"cls:DescribeIndex",
```

```
"cls:DescribeIndexs",
"cls:DescribePartitions",
"cls:SearchLog",
"cls:DescribeLogHistogram",
"cls:DescribeLogContext",
"cls:DescribeLogFastAnalysis",
"cls:DescribeLatestJsonLog",
"cls:DescribeRebuildIndexTasks"
],
"resource": [
"qcs::cls:ap-guangzhou:100007***827:logset/1c012db7-2cfd-4418-****-7342c7a42516",
"qcs::cls:ap-guangzhou:100007***827:topic/380fe1f1-0c7b-4b0d-****-d514959db1bb"
]
}
]
}
```

只读权限：对指定标签的日志主题具备只读权限

用户可以对包含指定标签的日志主题进行检索。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeLogsets",
"cls:DescribeTopics",
"cls:DescribeExports",
"cls:DescribeIndex",
"cls:DescribeIndexs",
"cls:DescribePartitions",
"cls:SearchLog",
"cls:DescribeLogHistogram",
"cls:DescribeLogContext",
"cls:DescribeLogFastAnalysis",
"cls:DescribeLatestJsonLog",
"cls:DescribeRebuildIndexTasks"
],
"resource": [
"*"
],
"condition": {
"for_any_value:string_equal": {
"qcs:resource_tag": [
"testCAM&test1"
]
}
}
}
]
}
```

使用 API 检索分析日志

只读权限：对所有日志主题具备检索分析只读权限

用户可以通过 API 对所有的日志主题进行日志检索分析。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:SearchLog"
],
"resource": [
"*"
]
}
]
}
```

只读权限：对指定日志主题具备检索分析只读权限

用户可以通过 API 对指定的日志主题进行日志检索分析。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:SearchLog"
],
"resource": [
"qcs::cls:ap-guangzhou:100007***827:logset/1c012db7-2cfd-4418-****-7342c7a42516",
"qcs::cls:ap-guangzhou:100007***827:topic/380fe1f1-0c7b-4b0d-****-d514959db1bb"
]
}
]
}
```

只读权限：对指定标签的日志主题具备检索分析只读权限

用户可以通过 API 对包含指定标签的的日志主题进行日志检索分析。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:SearchLog"
],
"resource": [
"*"
],
"condition": {
"for_any_value:string_equal": {
"qcs:resource_tag": [
"testCAM&test1"
]
}
```

```
}
}
]
}
```

通过仪表盘分析日志

管理权限：对所有仪表盘具备管理权限

用户可以管理所有的仪表盘，包括创建、编辑和删除仪表盘，并能够通过仪表盘查看所有日志主题的数据。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:GetChart",
"cls:GetDashboard",
"cls:ListChart",
"cls:CreateChart",
"cls:CreateDashboard",
"cls:DeleteChart",
"cls:DeleteDashboard",
"cls:ModifyChart",
"cls:ModifyDashboard",
"cls:DescribeDashboards"
],
"resource": "*"
},
{
"effect": "allow",
"action": [
"cls:SearchLog",
"cls:DescribeTopics",
"cls:DescribeLogFastAnalysis",
"cls:DescribeIndex",
"cls:DescribeLogsets"
],
"resource": "*"
}
]
}
```

管理权限：对指定标签的日志主题及仪表盘具备管理权限

用户可以管理包含指定标签的仪表盘，包括创建、编辑和删除仪表盘，并能够通过仪表盘查看包含指定标签的日志主题的数据。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:GetChart",
"cls:GetDashboard",
"cls:ListChart",
"cls:CreateChart",
```

```
"cls:CreateDashboard",
"cls:DeleteChart",
"cls:DeleteDashboard",
"cls:ModifyChart",
"cls:ModifyDashboard",
"cls:DescribeDashboards"
],
"resource": "*",
"condition": {
"for_any_value:string_equal": {
"qcs:resource_tag": [
"key&amp;amp;value"
]
}
}
},
{
"effect": "allow",
"action": [
"cls:SearchLog",
"cls:DescribeTopics",
"cls:DescribeLogFastAnalysis",
"cls:DescribeIndex",
"cls:DescribeLogsets"
],
"resource": "*",
"condition": {
"for_any_value:string_equal": {
"qcs:resource_tag": [
"key&amp;amp;value"
]
}
}
}
]
}
```

管理权限：对指定资源的日志主题及仪表盘具备管理权限

用户可以管理指定仪表盘，包括创建、编辑和删除仪表盘，并能够通过仪表盘查看指定日志主题的数据。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:GetChart",
"cls:GetDashboard",
"cls:ListChart",
"cls:CreateChart",
"cls:CreateDashboard",
"cls:DeleteChart",
"cls:DeleteDashboard",
"cls:ModifyChart",
"cls:ModifyDashboard",
"cls:DescribeDashboards"
```

```
],
"resource": [
"qcs::cls::uin/100000***001:dashboard/dashboard-0769a3ba-2514-409d-****-f65b20b23736"
]
},
{
"effect": "allow",
"action": [
"cls:SearchLog",
"cls:DescribeTopics",
"cls:DescribeLogFastAnalysis",
"cls:DescribeIndex",
"cls:DescribeLogsets"
],
"resource": [
"qcs::cls::uin/100000***001:topic/174ca473-50d0-4fdf-****-2ef681a1e02a"
]
}
]
}
```

只读权限：对所有仪表盘具备只读权限

用户可以查看所有的仪表盘，并能够通过仪表盘查看所有日志主题的数据。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:GetChart",
"cls:GetDashboard",
"cls:ListChart",
"cls:DescribeDashboards"
],
"resource": "*"
},
{
"effect": "allow",
"action": [
"cls:SearchLog",
"cls:DescribeTopics",
"cls:DescribeLogFastAnalysis",
"cls:DescribeIndex",
"cls:DescribeLogsets"
],
"resource": "*"
}
]
}
```

只读权限：对指定标签的日志主题及仪表盘具备只读权限

用户可以查看包含指定标签的仪表盘，并能够通过仪表盘查看包含指定标签的日志主题的数据。

```
{
"version": "2.0",
```

```
"statement": [
{
"effect": "allow",
"action": [
"cls:GetChart",
"cls:GetDashboard",
"cls:ListChart",
"cls:DescribeDashboards"
],
"resource": "*",
"condition": {
"for_any_value:string_equal": {
"qcs:resource_tag": [
"key&value"
]
}
}
},
{
"effect": "allow",
"action": [
"cls:SearchLog",
"cls:DescribeTopics",
"cls:DescribeLogFastAnalysis",
"cls:DescribeIndex",
"cls:DescribeLogsets"
],
"resource": "*",
"condition": {
"for_any_value:string_equal": {
"qcs:resource_tag": [
"key&value"
]
}
}
}
]
}
```

只读权限：对指定资源的日志主题及仪表盘具备只读权限

用户可以查看指定仪表盘，并能够通过仪表盘查看指定日志主题的数据。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:GetChart",
"cls:GetDashboard",
"cls:ListChart",
"cls:DescribeDashboards"
],
"resource": [
"qcs::cls::uin/100000***001:dashboard/dashboard-0769a3ba-2514-409d-****-f65b20b23736"
]
```

```
},
{
"effect": "allow",
"action": [
"cls:SearchLog",
"cls:DescribeTopics",
"cls:DescribeLogFastAnalysis",
"cls:DescribeIndex",
"cls:DescribeLogsets"
],
"resource": [
"qcs::cls::uin/100000***001:topic/174ca473-50d0-4fdf-****-2ef681a1e02a"
]
}
]
}
```

监控告警相关

管理权限：对所有告警策略具备管理权限

用户可以对所有告警策略进行管理，包括创建告警策略、创建通知渠道组和查看告警策略等。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeLogsets",
"cls:DescribeTopics"
],
"resource": [
"*"
]
},
{
"effect": "allow",
"action": [
"cls:DescribeAlarms",
"cls:CreateAlarm",
"cls:ModifyAlarm",
"cls:DeleteAlarm",
"cls:DescribeAlarmNotices",
"cls:CreateAlarmNotice",
"cls:ModifyAlarmNotice",
"cls:DeleteAlarmNotice",
"cam:ListGroups",
"cam:DescribeSubAccountContacts",
"cls:GetAlarmLog",
"cls:DescribeAlertRecordHistory",
"cls:CheckAlarmRule",
"cls:CheckAlarmChannel"
],
"resource": "*"
}
```

```
]
}
```

只读权限：对所有告警策略具备只读权限

用户可以查看所有告警策略。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeLogsets",
"cls:DescribeTopics"
],
"resource": [
"*"
]
},
{
"effect": "allow",
"action": [
"cls:DescribeAlarms",
"cls:DescribeAlarmNotices",
"cls:GetAlarmLog",
"cls:DescribeAlertRecordHistory",
"cam:ListGroups",
"cam:DescribeSubAccountContacts"
],
"resource": "*"
}
]
}
```

数据加工相关

管理权限：对所有数据加工任务具备管理权限

所有日志主题的“数据加工任务”的管理权限。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeLogsets",
"cls:DescribeDataTransformPreviewDataInfo",
"cls:DescribeTopics",
"cls:DescribeIndex",
"cls:CreateDataTransform"
],
"resource": [
"*"
]
},
{
```

```
"effect": "allow",
"action": [
"cls:DescribeFunctions",
"cls:CheckFunction",
"cls:DescribeDataTransformFailLogInfo",
"cls:DescribeDataTransformInfo",
"cls:DescribeDataTransformPreviewInfo",
"cls:DescribeDataTransformProcessInfo",
"cls:DeleteDataTransform",
"cls:ModifyDataTransform"
],
"resource": [
"*"
]
}
]
}
```

只读权限：对所有数据加工任务具备只读权限

所有日志主题的“数据加工任务”的只读权限。由于仅是查看，所以不需要对 DSL 函数进行授权。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeLogsets",
"cls:DescribeTopics"
],
"resource": [
"*"
]
},
{
"effect": "allow",
"action": [
"cls:DescribeDataTransformFailLogInfo",
"cls:DescribeDataTransformInfo",
"cls:DescribeDataTransformPreviewDataInfo",
"cls:DescribeDataTransformPreviewInfo",
"cls:DescribeDataTransformProcessInfo"
],
"resource": [
"*"
]
}
]
}
```

数据投递/消费相关

投递 Ckafka

管理权限：对所有日志主题具备投递 Ckafka 管理权限

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeTopics",
"cls:DescribeLogsets"
],
"resource": "*"
},
{
"effect": "allow",
"action": [
"tag:DescribeResourceTagsByResourceIds",
"tag:DescribeTagKeys",
"tag:DescribeTagValues",
"cam:ListAttachedRolePolicies",
"ckafka:DescribeInstances",
"ckafka:DescribeTopic",
"ckafka:DescribeInstanceAttributes",
"cls:modifyConsumer",
"cls:getConsumer"
],
"resource": "*"
}
]
}
```

管理权限：对指定标签日志主题具备投递 Ckafka 管理权限

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeTopics",
"cls:DescribeLogsets"
],
"resource": "*",
"condition": {
"for_any_value:string_equal": {
"qcs:resource_tag": [
"age&13",
"name&vinson"
]
}
}
},
{
"effect": "allow",
"action": [
"tag:DescribeResourceTagsByResourceIds",
"tag:DescribeTagKeys",
"tag:DescribeTagValues",
```

```
"cam:ListAttachedRolePolicies",
"ckafka:DescribeInstances",
"ckafka:DescribeTopic",
"ckafka:DescribeInstanceAttributes",
"cls:modifyConsumer",
"cls:getConsumer"
],
"resource": "*"
}
]
}
```

只读权限：对所有日志主题具备投递ckafka只读权限

具备所有日志主题投递 Ckafka 的只读权限。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeTopics",
"cls:DescribeLogsets"
],
"resource": "*"
},
{
"effect": "allow",
"action": [
"tag:DescribeResourceTagsByResourceIds",
"tag:DescribeTagKeys",
"tag:DescribeTagValues",
"cam:ListAttachedRolePolicies",
"ckafka:DescribeInstances",
"ckafka:DescribeTopic",
"ckafka:DescribeInstanceAttributes",
"cls:getConsumer"
],
"resource": "*"
}
]
}
```

只读权限：对指定标签日志主题具备投递 Ckafka 只读权限

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeTopics",
"cls:DescribeLogsets"
],
"resource": "*",
"condition": {
```

```
"for_any_value:string_equal": {
"qcs:resource_tag": [
"key&value"
]
}
}
},
{
"effect": "allow",
"action": [
"tag:DescribeResourceTagsByResourceIds",
"tag:DescribeTagKeys",
"tag:DescribeTagValues",
"cam:ListAttachedRolePolicies",
"ckafka:DescribeInstances",
"ckafka:DescribeTopic",
"ckafka:DescribeInstanceAttributes",
"cls:getConsumer"
],
"resource": "*"
}
]
}
```

投递 COS

管理权限：对所有日志主题具备投递 COS 管理权限

具备所有日志主题投递 COS 的管理权限。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeTopics",
"cls:DescribeLogsets",
"cls:DescribeIndex"
],
"resource": "*"
},
{
"effect": "allow",
"action": [
"tag:DescribeResourceTagsByResourceIds",
"tag:DescribeTagKeys",
"tag:DescribeTagValues",
"cls:CreateShipper",
"cls:ModifyShipper",
"cls:DescribeShippers",
"cls:DeleteShipper",
"cls:DescribeShipperTasks",
"cls:RetryShipperTask",
"cam:ListAttachedRolePolicies",
"cos:GetService"
],
"resource": "*"
```

```
}
]
}
```

管理权限：对指定标签日志主题具备投递 COS 管理权限

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeTopics",
"cls:DescribeLogsets",
"cls:DescribeIndex"
],
"resource": "*",
"condition": {
"for_any_value:string_equal": {
"qcs:resource_tag": [
"key&value"
]
}
}
},
{
"effect": "allow",
"action": [
"tag:DescribeResourceTagsByResourceIds",
"tag:DescribeTagKeys",
"tag:DescribeTagValues",
"cls:CreateShipper",
"cls:ModifyShipper",
"cls:DescribeShippers",
"cls:DeleteShipper",
"cls:DescribeShipperTasks",
"cls:RetryShipperTask",
"cam:ListAttachedRolePolicies",
"cos:GetService"
],
"resource": "*"
}
]
}
```

只读权限：对所有日志主题具备投递 COS 只读权限

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeTopics",
"cls:DescribeLogsets" ],
"resource": "*"
```

```
},
{
"effect": "allow",
"action": [
"tag:DescribeResourceTagsByResourceIds",
"tag:DescribeTagKeys",
"tag:DescribeTagValues",
"cls:DescribeShippers",
"cls:DescribeShipperTasks",
"cls:RetryShipperTask",
"cam:ListAttachedRolePolicies"
],
"resource": "*"
}
]
}
```

只读权限：对指定标签日志主题具备投递 COS 只读权限

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeTopics",
"cls:DescribeLogsets"],
"resource": "*",
"condition": {
"for_any_value:string_equal": {
"qcs:resource_tag": [
"key&value"
]
}
}
},
{
"effect": "allow",
"action": [
"tag:DescribeResourceTagsByResourceIds",
"tag:DescribeTagKeys",
"tag:DescribeTagValues",
"cls:DescribeShippers",
"cls:DescribeShipperTasks",
"cls:RetryShipperTask",
"cam:ListAttachedRolePolicies"
],
"resource": "*"
}
]
}
```

投递 SCF

管理权限：对所有日志主题具备投递 SCF 管理权限

具备所有日志主题投递 SCF 的管理权限。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeTopics",
"cls:DescribeLogsets"
],
"resource": "*"
},
{
"effect": "allow",
"action": [
"tag:DescribeResourceTagsByResourceIds",
"tag:DescribeTagKeys",
"tag:DescribeTagValues",
"cam:ListAttachedRolePolicies",
"cls:CreateDeliverFunction",
"cls:DeleteDeliverFunction",
"cls:ModifyDeliverFunction",
"cls:GetDeliverFunction",
"scf:ListFunctions",
"scf:ListAliases",
"scf:ListVersionByFunction"
],
"resource": "*"
}
]
}
```

管理权限：对指定标签日志主题具备投递 SCF 管理权限

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeTopics",
"cls:DescribeLogsets"
],
"resource": "*",
"condition": {
"for_any_value:string_equal": {
"qcs:resource_tag": [
"key&value"
]
}
}
},
{
"effect": "allow",
"action": [
"tag:DescribeResourceTagsByResourceIds",
"tag:DescribeTagKeys",
```

```
"tag:DescribeTagValues",
"cam:ListAttachedRolePolicies",
"cls:CreateDeliverFunction",
"cls:DeleteDeliverFunction",
"cls:ModifyDeliverFunction",
"cls:GetDeliverFunction",
"scf:ListFunctions",
"scf:ListAliases",
"scf:ListVersionByFunction"
],
"resource": "*"
}
]
}
```

只读权限：对所有日志主题具备投递 SCF 只读权限

具备所有日志主题投递 SCF 的只读权限。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeTopics",
"cls:DescribeLogsets"
],
"resource": "*"
},
{
"effect": "allow",
"action": [
"tag:DescribeResourceTagsByResourceIds",
"tag:DescribeTagKeys",
"tag:DescribeTagValues",
"cam:ListAttachedRolePolicies",
"cls:GetDeliverFunction",
"scf:ListFunctions",
"scf:ListAliases",
"scf:ListVersionByFunction"
],
"resource": "*"
}
]
}
```

只读权限：对指定标签日志主题具备投递 SCF 只读权限

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeTopics",
"cls:DescribeLogsets"
```

```
],
"resource": "*",
"condition": {
"for_any_value:string_equal": {
"qcs:resource_tag": [
"key&value"
]
}
}
},
{
"effect": "allow",
"action": [
"tag:DescribeResourceTagsByResourceIds",
"tag:DescribeTagKeys",
"tag:DescribeTagValues",
"cam:ListAttachedRolePolicies",
"cls:GetDeliverFunction",
"scf:ListFunctions",
"scf:ListAliases",
"scf:ListVersionByFunction"
],
"resource": "*"
}
]
}
```

Kafka 协议消费

管理权限：对所有日志主题具备 Kafka 协议消费权限

具备所有日志主题 Kafka 协议消费权限。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeLogsets",
"cls:DescribeTopics"
],
"resource": [
"*"]
},
{
"effect": "allow",
"action": [
"tag:DescribeResourceTagsByResourceIds",
"tag:DescribeTagKeys",
"tag:DescribeTagValues",
"cls:DescribeKafkaConsumer",
"cls:CloseKafkaConsumer",
"cls:ModifyKafkaConsumer",
"cls:OpenKafkaConsumer",
"cam:ListAttachedRolePolicies"
],
"resource": [
```

```
"*"
]
}
]
}
```

管理权限：对指定标签日志主题具备 Kafka 协议消费权限

具备指定标签日志主题 Kafka 协议消费的管理权限。

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:DescribeLogsets",
"cls:DescribeTopics"
],
"resource": [
"*"
],
"condition": {
"for_any_value:string_equal": {
"qcs:resource_tag": [
"key&value"
]
}
}
},
{
"effect": "allow",
"action": [
"tag:DescribeResourceTagsByResourceIds",
"tag:DescribeTagKeys",
"tag:DescribeTagValues",
"cls:DescribeKafkaConsumer",
"cls:CloseKafkaConsumer",
"cls:ModifyKafkaConsumer",
"cls:OpenKafkaConsumer",
"cam:ListAttachedRolePolicies"
],
"resource": [
"*"
]
}
]
}
```

开发者相关

CLS 对接 Grafana

通过 Grafana 展示所有日志主题的数据

```
{
"version": "2.0",
```

```
"statement": [
{
"effect": "allow",
"action": [
"cls:SearchLog"
],
"resource": [
"*"
]
}
]
}
```

通过 Grafana 展示具备指定标签的日志主题的数据

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"cls:SearchLog"
],
"resource": [
"*"
],
"condition": {
"for_any_value:string_equal": {
"qcs:resource_tag": [
"key&value"
]
}
}
}
]
}
```

日志采集
采集概述

最近更新时间：2022-06-21 11:20:38

概述

日志服务（Cloud Log Service，CLS）提供采集客户端， API 接入方式和 SDK 采集方式，方便用户将应用程序日志导入到日志服务。目前日志服务提供结构化上传方式，要求数据按照 key-value 的方式进行上传。

日志结构化

日志的结构化指您的日志数据将以 key-value 的形式存储在 CLS 平台上。结构化后的日志数据可以根据指定的键值进行日志检索、日志分析及日志投递。日志服务允许直接上报结构化的数据，详见以下说明：

例如，一条本地原始日志为：

```
10.20.20.10;[Tue Jan 22 14:49:45 CST 2019 +0800];GET /online/sample HTTP/1.1;127.0.0.1;200;647;35;http://127.0.0.1/
```

指定日志的解析方式为分隔符方式，以 ; 分号作为分隔符，可以将该条日志解析为多个字段组，每个字段组按 key-value 键值对的方式进行组织，为每个 key 定义一个键名称，如下所示：

```
IP: 10.20.20.10
time: [Tue Jan 22 14:49:45 CST 2019 +0800]
request: GET /online/sample HTTP/1.1
host: 127.0.0.1
status: 200
length: 647
bytes: 35
referer: http://127.0.0.1/
```

LogListener 提供多种解析方式，其中单行全文或多行全文上报的日志也会解析成结构化形式，LogListener 会默认添加 __CONTENT__ 作为 key，原文作为 value，以此组成 key-value 键值对，如下表所示：

LogListener 解析方式	键名称（key）	解释说明
单行/多行全文	__CONTENT__	默认 __CONTENT__ 为键名
json 格式	json 中的名称	自动以 json 日志原文中的名称/值，作为 key-value 键值对
分隔符格式	自定义	根据分隔符号划分字段后，需为每组字段自定义 key 名称
完全正则	自定义	根据正则表达式提取字段后，需为每组字段自定义 key 名称

采集方式

日志服务提供多种采集方式：

采集方式	描述
API 方式采集	通过调用 日志服务 API 上传结构化日志至日志服务，详情请参考 上传日志接口 文档
SDK 方式采集	通过使用 SDK 上传结构化日志至日志服务，详情请参考 SDK 采集 文档
LogListener 客户端采集	LogListener 是日志服务提供的日志采集客户端，通过控制台简单配置可快速接入日志服务，详情请参考 LogListener 使用流程

采集方式对比：

类别名称	LogListener 采集	API 方式采集

类别名称	LogListener 采集	API 方式采集
修改代码	对应用程序是无侵入式，无需修改代码	需修改应用程序代码才能上报日志
断点续传	支持断点续传日志	自行代码实现
失败重传	自带重试机制	自行代码实现
本地缓存	支持本地缓存，高峰期间保障数据完整	自行代码实现
资源占用	占用内存、CPU 等资源	无额外资源占用

日志源接入

不同的日志源可以选择不同的日志接入方式，详情参考以下列表：

日志源类别

日志源类别	推荐接入方式
程序直接输出	API
本地日志文件	LogListener

日志源环境

系统环境	推荐接入方式
Linux/Unix	LogListener/Kafka 协议上传/API
Windows	Kafka 协议上传/API
iOS/Android/Web 端	提供 SDK 采集

云产品日志

云产品名称	推荐接入方式
云服务器（Cloud Virtual Machine，CVM）	安装配置 LogListener，采集指引
容器服务（Tencent Kubernetes Engine，TKE）	控制台配置，接入指引
内容分发网络（Content Delivery Network，CDN）	控制台配置，接入指引
负载均衡（Cloud Load Balancer，CLB）	控制台配置，接入指引
云函数（Serverless Cloud Function，SCF）	控制台配置，接入指引
标准直播（Live Video Broadcasting，LVB）	控制台配置，接入指引
网络流日志（Flow Logs，FL）	控制台配置，接入指引
腾讯云 TI 平台 TI-ONE	控制台配置，接入指引
Web 应用防火墙（Web Application Firewall，WAF）	控制台配置，配置指引
消息队列 CKafka（Cloud Kafka）	控制台配置，配置指引

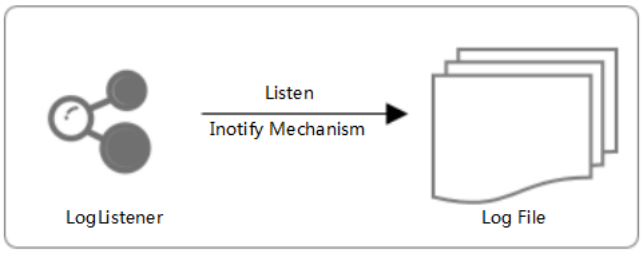
LogListener 采集
采集机制

最近更新时间：2022-01-18 18:47:51

LogListener 是腾讯云日志服务（Cloud Log Service，CLS）所提供的日志采集客户端，它将按照预设的采集策略实时上报日志数据，本篇文档将详细阐述 LogListener 的工作机制。

机制原理

日志服务的 LogListener 成功部署之后，会对所关联的日志文件进行实时监听，主要通过文件系统修改的通知机制 Inotify 来感知目标日志文件的变化，这里的变化不仅是文件内容的变化，对于 Linux 系统而言，也包括文件 inode 发生改变。当 LogListener 感知到日志文件发生变化，就会主动采集上报新写入的日志，并记录当前位置。即使系统重启，也会从记录的位置继续采集日志。



示例说明

为了更直观地说明日志服务 LogListener 采集策略，举例进行说明：

```
2018-01-01 10:00:01 start LogListener
2018-01-01 10:00:02 echo log_1 >> cls.log
2018-01-01 10:00:03 echo log_2 >> cls.log
2018-01-01 10:00:04 echo log_3 >> cls.log
2018-01-01 10:00:05 echo log_4 >> cls.log
......
```

在上述场景中，LogListener 将采集 log_1、log_2、log_3…… 到日志服务当中，并自动监听上报目标文件的所有日志。注意，这里 LogListener 会监控到文件的 inode，若用 vim 修改日志文件 cls.log 时，由于 vim 机制会修改 inode，所以日志系统会认为是一个全新的日志文件，将会采集上报整个文件的内容。

> 说明：
>
> - 机器重启后，会自动拉起 LogListener。
> - LogListener 进程挂掉重启后，会根据所记录的偏移位置继续上报日志。
> - 目前一个日志文件仅能上报到一个日志主题。若有多个日志主题关联到同一个日志文件，配置信息会覆盖，因此该日志文件实际只会上报到最后一个主题中。

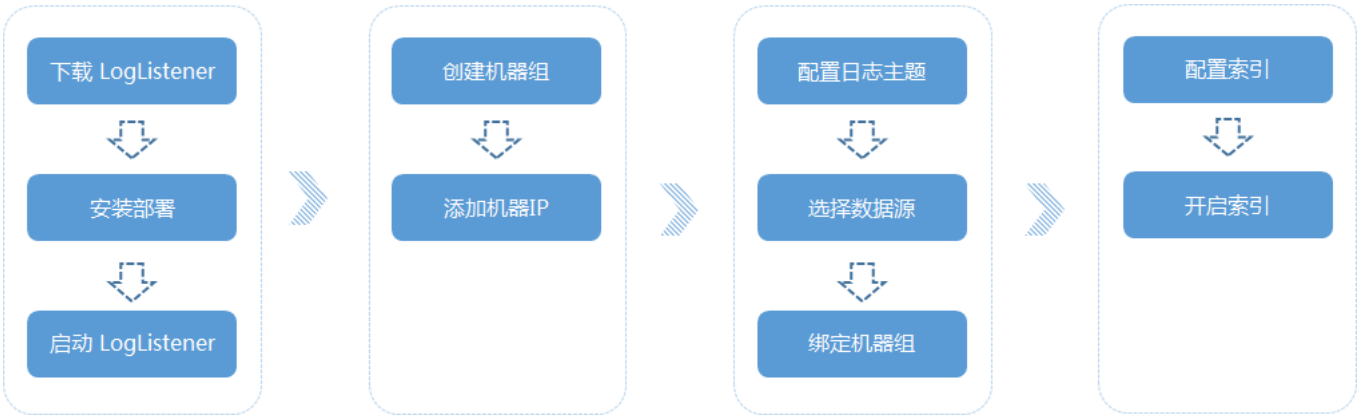
LogListener 使用流程

最近更新时间：2022-01-18 18:44:25

概述

LogListener 是日志服务提供的日志采集客户端，通过安装部署 LogListener，可以方便快速地接入日志服务，无需修改应用程序运行逻辑，是一种对应用服务无侵入式的采集方式。

使用 LogListener 采集日志的流程如下图所示：



流程说明

1. 下载安装最新版本 LogListener。LogListener 成功安装后将自动启动进程，并与日志服务后端保持心跳连接。
2. 前往 日志服务控制台，创建 机器组，并添加机器 IP。
3. 前往日志主题的采集配置页，输入日志路径确定数据源，并绑定机器组。详细操作示例可参阅 单行全文格式 文档。
4. 前往日志主题的索引配置页，配置全文或键值索引，并开启索引。详细操作示例可参阅 单行全文格式 文档。

至此，LogListener 将按照日志主题的采集配置监听符合规则的日志文件，用户可以通过日志检索查看采集上来的日志数据。

LogListener 安装和部署
LogListener 安装指南

最近更新时间：2022-06-07 10:15:19

LogListener 是腾讯云日志服务（Cloud Log Service，CLS）所提供的专用日志采集器，将它安装部署到服务器上，可快速采集日志到日志服务。

安装环境

LogListener 仅支持64位 Linux 操作系统环境（暂不支持 Windows），并适配主流 Linux 操作系统版本，若其他版本环境若安装异常，请 提交工单 联系我们。

操作系统类别	确定可安装环境
CentOS（64位）	CentOS_6.8_64位、CentOS_6.9_64位、CentOS_7.2_64位、CentOS_7.3_64位、CentOS_7.4_64位、CentOS_7.5_64位、CentOS_7.6_64位、CentOS_8.0_64位
Ubuntu（64位）	Ubuntu Server_14.04.1_LTS_64位、Ubuntu Server_16.04.1_LTS_64位、Ubuntu Server_18.04.1_LTS_64位
Debian（64位）	Debian_8.2_64位、Debian_9.0_64位
openSUSE（64位）	openSUSE_42.3_64位

支持功能

LogListener 版本支持重要功能如下，详细版本功能信息请参考 LogListener 版本变更：

LogListener 版本	支持功能	功能说明	相关文档
v2.7.4	支持采集主机名 hostname	Loglistener 会默认采集机器的主机名作为默认字段上报，以 __HOSTNAME__ 作为 key 展现，例如 __HOSTNAME__：VM-108-centos。	-
v2.6.4	支持用户通过组合解析自定义复杂日志解析规则	使用 Loglistener 组合解析格式解析日志，此模式支持用户在控制台输入代码（JSON 格式）用来定义日志解析的流水线逻辑。	组合解析格式
v2.6.0	支持腾讯云 CVM 批量部署功能	支持用户在控制台选择 CVM 实例，接口批量下发部署 LogListener 任务，自动完成 LogListener 的安装部署（包括 accesskey，ID 配置，地域配置）。	CVM 批量部署 LogListener
v2.5.4	支持 LogListener 服务日志功能	LogListener 服务日志功能支持记录 LogListener 端运行状态和采集监控的日志数据并配置可视化视图，提供重要指标数据。	LogListener 服务日志
v2.5.2	支持上传解析失败日志	所有解析失败的日志，均以 LogParseFailure 作为键名称（Key），原始日志内容作为值（Value）进行上传。	-
v2.5.0	支持 LogListener 自动升级功能	支持用户在控制台预设时间段指定机器组进行 agent 自动升级，也可对目标机器实行手动升级。	LogListener 升级指南
v2.4.5	支持多行-完全正则采集模式	LogListener 采集配置规则新增**多行-完全正则**提取模式采集日志。	完全正则（多行）

安装启动

1. 下载安装 LogListener

LogListener 最新版本下载地址：公网下载 LogListener、内网下载 LogListener

以安装路径 /usr/local/ 为例： 下载 LogListener 安装包并解压，解压路径为 /usr/local/ ，解压完成后进入 LogListener 目录 loglistener/tools ，执行安装命令 。

- 公网环境下，操作命令如下：

```
wget https://mirrors.tencent.com/install/cls/loglistener-linux-x64-2.7.9.tar.gz && tar -zxvf loglistener-linux-x64-2.7.9.tar.gz -C /usr/local && cd /usr/local/loglistener-2.7.9/tools && ./loglistener.sh install
```

- 内网环境下，操作命令如下：

```
wget http://mirrors.tencentyun.com/install/cls/loglistener-linux-x64-2.7.9.tar.gz && tar -zxvf loglistener-linux-x64-2.7.9.tar.gz -C /usr/local && cd /usr/local/loglistener-2.7.9/tools && ./loglistener.sh install
```

2. 初始化 LogListener

在 loglistener/tools 路径下，以 root 权限执行 LogListener 初始化命令（默认使用内网方式访问服务），初始化命令如下：

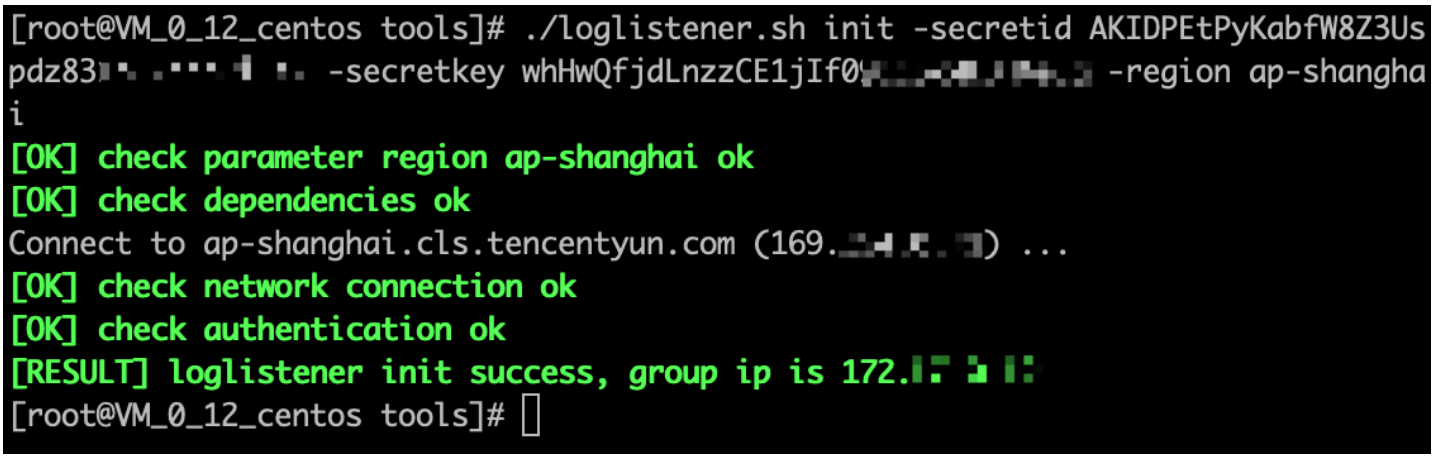
```
./loglistener.sh init -secretid AKIDPEtPyKabfW8Z3Uspdz83xxxxxxxxxxx -secretkey whHwQfjdLnzzCE1jIf09xxxxxxxxxxxx -region ap-xxxxxx
```

> 说明：
> 初始化命令中 -secretid、-secretkey、-region、-network 为需要自主填写的参数，详细介绍请见如下 参数说明。

参数说明

参数名	类型描述
secretid	云 API 密钥 的一部分，SecretId 用于标识 API 调用者身份
secretkey	云 API 密钥 的一部分，SecretKey 是用于加密签名字符串和服务器端验证签名字符串的密钥
region	region 表示日志服务所在的 地域，此处填写域名简称，例如 ap-beijing、ap-guangzhou 等
network	表示 loglistener 通过哪种方式访问服务域名，取值：intra 内网访问（默认），internet 外网访问
ip	机器的 IP 标识。若不填写，loglistener 会自动获取本机的 IP 地址
label	机器组标示，标示机器组需要填写标示信息，多个标示按逗号分隔

默认使用内网域名：



如果需要通过外网方式访问服务域名，需要显式设置网络参数 internet，执行如下命令：

```
./loglistener.sh init -secretid AKIDPEtPyKabfW8Z3Uspdz83xxxxxxxxxxxx -secretkey whHwQfjdLnzzCE1jIf0xxxxxxxxxxxx -region ap-xxxxxx -network internet
```

```
[root@VM_0_12_centos tools]# ./loglistener.sh init -secretid AKIDPEtPyKabfW8Z3Us
pdz83[illegible] -secretkey whHwQfjdLnzzCE1jIf0[illegible] -region ap-shangha
i -network internet
[OK] check parameter region ap-shanghai ok
[OK] check dependencies ok
Connect to ap-shanghai.cls.tencentcs.com (211.[illegible]) ...
[OK] check network connection ok
[OK] check authentication ok
config parameters change, data will be cleaned, do you want to continue? (yes/no
): yes
[RESULT] loglistener init success, group ip is 172.[illegible]
[root@VM_0_12_centos tools]#
```

说明：

- 若主账号已授权协作者日志服务的读写权限，建议使用协作者密钥。
- region 为您所使用的日志服务区域，而非您的业务机器所处的区域。
- 云服务器与日志集同地域的情况下，建议使用内网方式访问服务域名。云服务器与日志集在不同地域的情况下，建议使用外网方式访问服务域名。
- 关于日志采集权限详情，可参考 授权子账号对 CLS 某个日志主题具有日志采集权限 文档。

3. 启动 LogListener

成功安装后，执行 LogListener 启动命令：

```
/etc/init.d/loglistenerd start
```

```
[root@VM_30_69_centos ~]# /etc/init.d/loglistenerd start
[OK] loglistener is running, ip is 10.[illegible]
[OK] start loglistener success
```

LogListener 常用操作

说明：

本文档示例的操作命令说明仅适用于 LogListener-2.2.4 及以上版本，低版本操作命令请参见 低版本 LogListener 安装指南。

1. 查看 LogListener 版本

```
/etc/init.d/loglistenerd -v
```

2. 查看 LogListener 帮助文档

```
/etc/init.d/loglistenerd -h
```

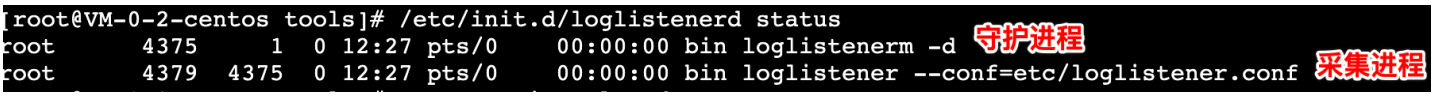
3. LogListener 进程管理

```
/etc/init.d/loglistenerd (start|restart|stop) # 启动、重启、停止
```

4. 查看 LogListener 进程状态

```
/etc/init.d/loglistenerd status
```

LogListener 正常情况会运行两个进程：



5. 检查 LogListener 心跳及配置

```
/etc/init.d/loglistenerd check
```

卸载 LogListener

以管理员权限执行 loglistener/tools 目录下的卸载命令：

```
./loglistener.sh uninstall
```

手动更新 LogListener

复用断点文件（不会重复采集日志）：

1. 使用停止命令停止运行旧版本的 LogListener。
2. 备份旧版本中的断点文件目录（loglistener/data）。例如，将旧版的断点文件备份至/tmp/loglistener-backup目录下。

```
cp -r loglistener-2.2.3/data /tmp/loglistener-backup/
```

3. 使用卸载命令卸载旧版本的 LogListener。
4. 下载最新版本的 LogListener，并使用相关命令安装和初始化新版本 LogListener。
5. 复制所备份的断点文件目录（步骤2）到新版本 LogListener 目录下。

```
cp -r /tmp/loglistener-backup/data loglistener-<version>/
```

请根据实际情况替换 <version>，例如：

```
cp -r /tmp/loglistener-backup/data loglistener-2.7.9/
```

6. 使用启动命令启动运行新版本 LogListener。

不复用断点文件（可能会重复采集日志）：

1. 使用停止命令停止运行旧版本的 LogListener。
2. 使用卸载命令卸载旧版本的 LogListener。
3. 下载最新版本的 LogListener，并使用相关命令安装和初始化新版本 LogListener。
4. 使用启动命令启动运行新版本 LogListener。

CVM 批量部署 LogListener

最近更新时间：2022-04-20 17:36:00

概述

日志服务（Cloud Log Service，CLS）支持使用 LogListener 采集腾讯云云服务器（Cloud Virtual Machine，CVM）实例上的日志，在进行日志采集前，需要在 CVM 上部署安装 LogListener。为了更快捷的支持大量的 CVM 部署安装 LogListener，现支持用户在控制台选择 CVM 实例，接口批量下发部署 LogListener 任务，自动完成 LogListener 的安装部署（包括 accesskey，ID 配置，地域配置）。

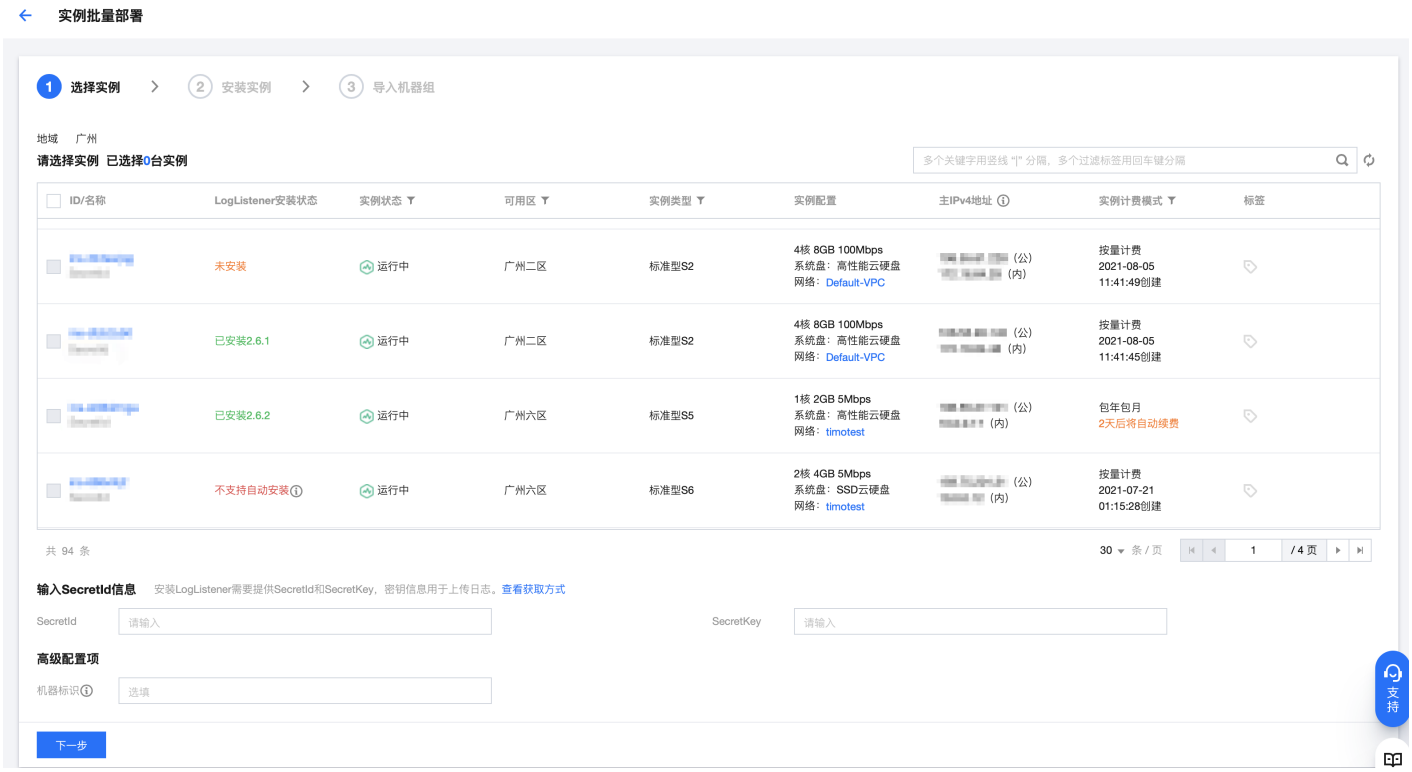
前提条件

CVM 已 安装腾讯云自动化助手（TencentCloud Automation Tools，TAT）。

操作步骤

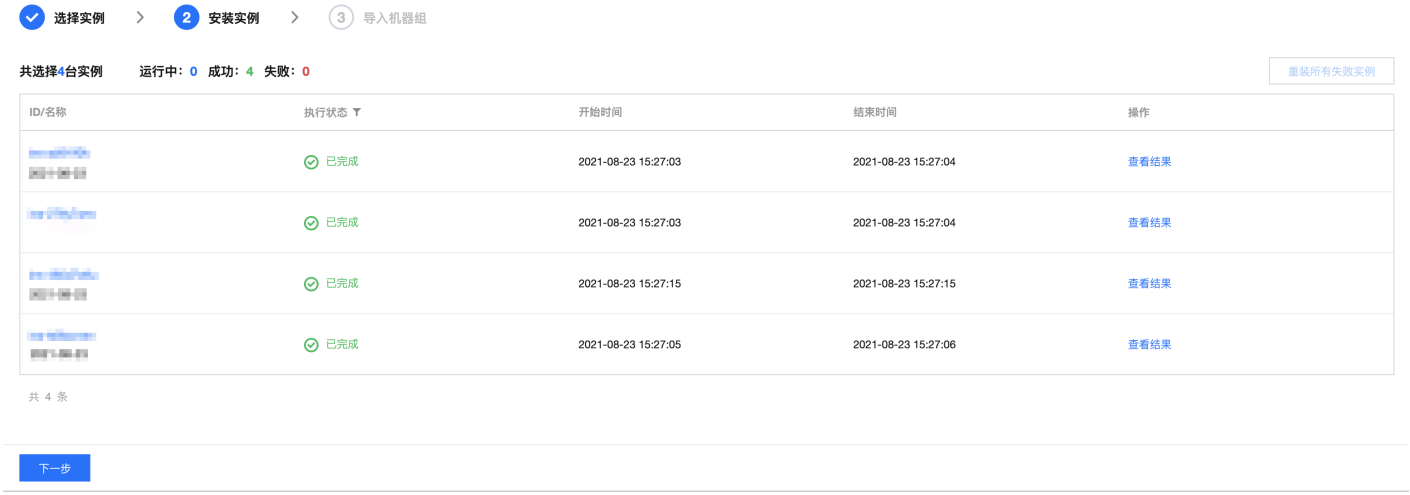
1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**概览**，进入概览页面。
3. 在**快速接入 > 云产品日志**栏中，单击**云服务器CVM**。

4. 在实例批量部署页面，勾选需要部署 LogListener 的 CVM 实例，输入 SecretId 信息（SecretId 和 SecretKey），并根据实际自定义设置机器标识，设置高级配置项。

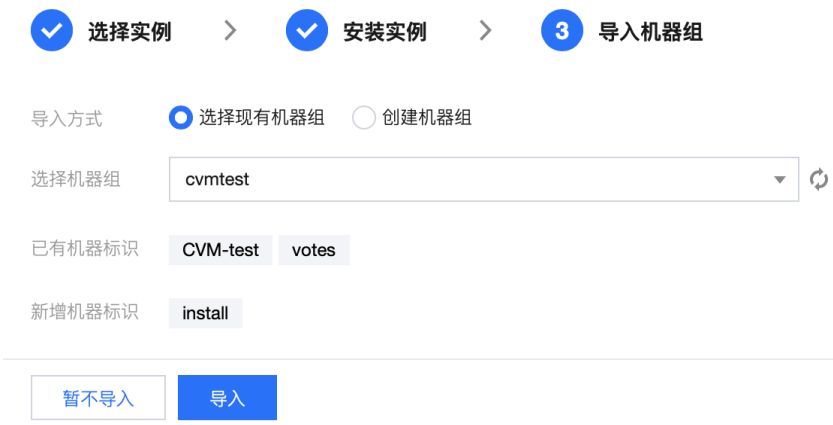


5. 单击**下一步**。
6. 在安装实例页面，待**执行状态**变为**已完成**，即安装完成后，单击**下一步**。

> **说明：**
> 如果安装失败，可以将鼠标移动到安装实例的**执行状态**查看失败原因。



7. 在导入机器组页面，根据实际需求，选择现有机器组或者创建机器组，单击**导入**，即可完成批量安装部署。



说明：

批量安装的 LogListener 为2.6.0及以上版本。

自建 K8S 集群安装 LogListener

最近更新时间：2022-01-20 11:29:01

本文介绍如何在自建 Kubernetes 集群上安装 LogListener 组件，从而将日志收集到日志服务（Cloud Log Service，CLS）。
在自建 Kubernetes 集群上安装 LogListener 组件的过程中，系统自动完成以下操作：

1. 创建 CLS-logconfigs CRD。
2. 部署工作负载 cls-provisioner。
3. 以 DaemonSet 模式安装 LogListener。

操作步骤

1. 登录 Kubernetes 集群。
2. 依次执行如下命令，安装 cls-provisioner Helm。

```
wget https://mirrors.tencent.com/install/cls/k8s/tencentcloud-cls-k8s-install.sh
```

```
chmod 744 tencentcloud-cls-k8s-install.sh
```

```
./tencentcloud-cls-k8s-install.sh --region xxx --secretid xxx --secretkey xxx
```

安装完成后，CLS 会自动创建名为 cls-k8s-随机 ID 的默认机器组。

> **注意：**
> 安装 cls-provisioner Helm 前，请确保已在 Kubernetes 集群中安装 Helm 命令，详情请参见 安装 Helm 文档。

参数说明

- --secretid：腾讯云账户访问密钥 ID。
- --secretkey：腾讯云账户访问密钥 Key。
- --region：CLS 服务地域。
- --docker_root：集群 Docker 的根目录，默认是 /var/lib/docker。

> **说明：**
> - 建议不同的集群使用不同的 cluster_id。
> - 不同的集群数据可以投递到相同的 topic。
> - 默认 ID 生成规则为：cls-k8s-8位随机 ID。

- --network：网络使用类型，内网（intra）或者外网（internet），默认使用内网。
- --api_network：云 API 的网络使用类型，内网（intra）或者外网（internet），默认使用外网（internet）。
- --api_region：云 API 的地域，地域详情请参见 可用地域 文档。
- region 和 api_region 地区保持一致，可分别参考 域名-loglistener，地域-日志服务 API3.0 文档检验网络。

示例

北京组件组件部署：

```
./tencentcloud-cls-k8s-install.sh --secretid xxx --secretkey xx --region ap-beijing --network internet --api_region ap-beijing
```

LogListener 升级指南

最近更新时间：2022-05-19 11:56:19

概述

为了更好的提供日志服务（Cloud Log Service，CLS）的日志采集服务，现支持 LogListener 自动升级、控制台手动升级和脚本半自动升级功能。LogListener 采集端程序版本迭代更新之后，用户无需自主下载新版本安装包进行相关操作的手动升级，只需在控制台预设时间段指定机器组进行 LogListener 自动升级，或对目标机器实行一键手动升级即可。

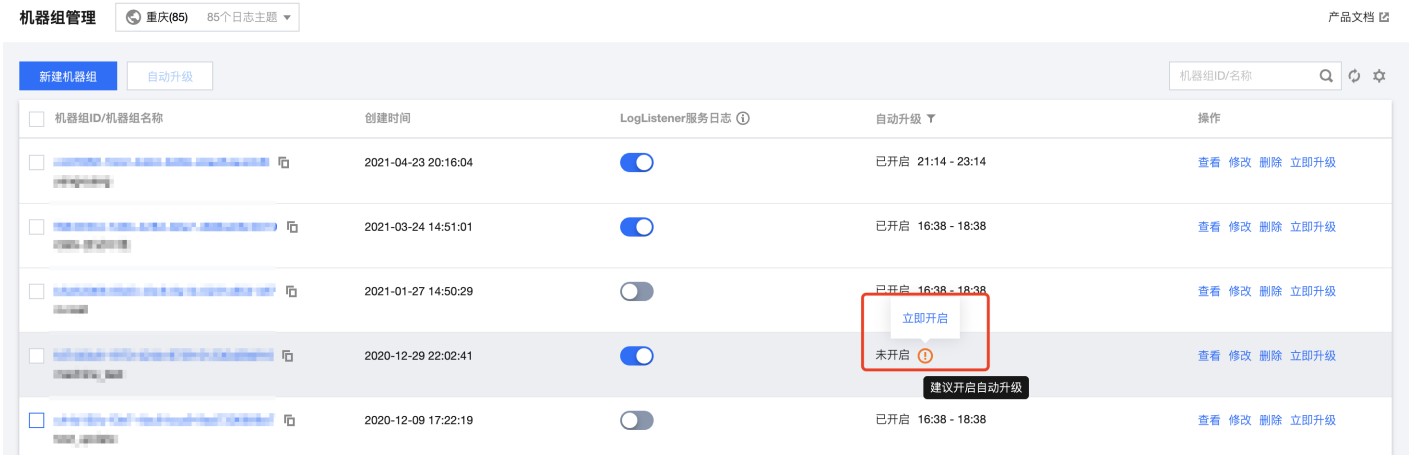
> 说明：
>
> - LogListener 自动升级功能仅在 LogListener 2.5.0 及以上版本开始支持，为了更好的使用体验，建议 前往安装/升级至最新版本。
> - 自动升级功能需 Python2 支持，若采集机器上安装的是 Python3，升级过程将无法完整执行。
> - 脚本半自动升级功能需 Python2.7 支持，若采集机器上安装的不是该版本，将无法使用此方式进行升级。

采集端程序 LogListener 控制台升级，可以使用自动升级和手动升级。
Agent 升级异常或者因为版本限制无法使用自动升级时，可以使用半自动升级。

操作步骤

自动升级

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**机器组管理**，进入机器组管理页面。
3. 找到需要自动升级的目标机器组，并将鼠标移动至“自动升级”栏下的 ，单击**立即开启**。



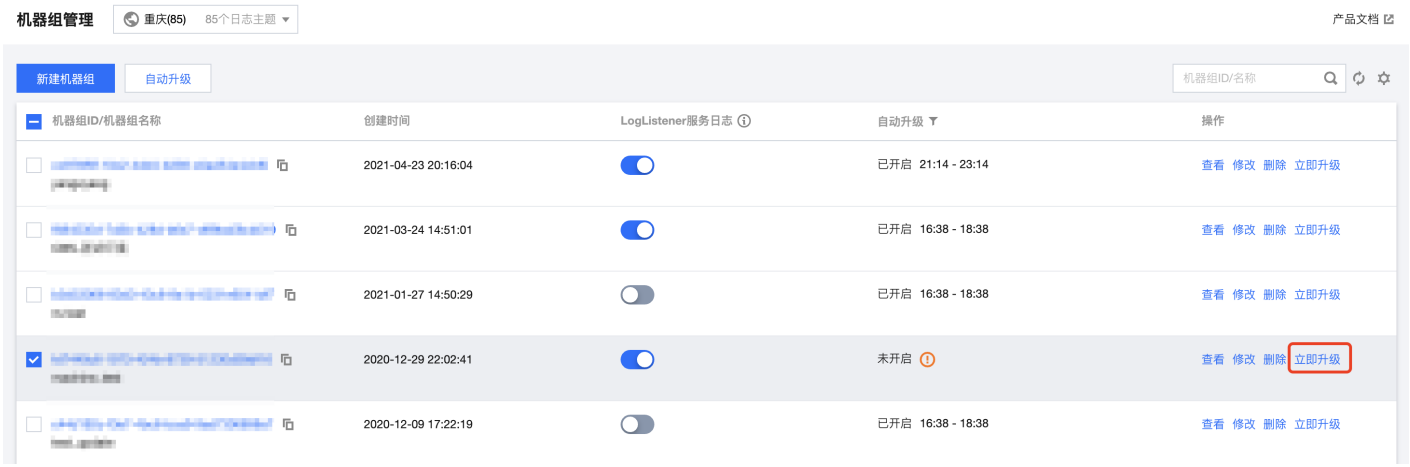
4. 在弹出的窗口中，开启 LogListener 开关，指定升级时间段（默认为当前时间至后两小时，如08:39～10:39）。

5. 单击**确定**，目标机器组的**“自动升级”**栏变为**“已开启”**，即表示开启自动升级 LogListener 成功。

> 说明：
>
> - 自动升级的时间段可选择任意时间段，系统会在用户指定的时间段每天进行检查。若满足升级条件，则进行自动升级；若不满足升级条件，则不进行操作。
> - 如需对多个机器组进行 LogListener 自动升级，可以勾选多个目标机器组，单击**自动升级**进行批量升级。

手动升级

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**机器组管理**，进入机器组管理页面。
3. 找到需要升级的目标机器组，单击“操作”栏的**立即升级**。



4. 在弹出的窗口中，勾选**“升级状态”为“可升级”**的目标机器，单击**手动更新**。

系统默认升级至最新版本，当**“升级状态”为“已是最新版本”**时，即表示手动升级成功。

> 说明：
>
> - 当升级状态显示“不支持更新”时，表示不支持在控制台手动更新 LogListener，需要您自主下载新版本安装包进行相关的手动升级操作，详情请参见 LogListener 安装指南。
> - 当升级状态显示“心跳异常”时，请 检查机器组状态。

半自动升级

1. 执行如下命令，确认 Python 版本。

```
python -V
```

若 Python 的版本非2.7，请 自动升级 或 手动升级。

2. 执行如下命令，下载脚本。

```
wget http://mirrors.tencentyun.com/install/cls/agent-update.py
```

3. 执行如下命令，执行脚本。

```
/usr/bin/python2.7 agent-update.py http://mirrors.tencentyun.com/install/cls/loglistener-linux-x64-x.tar.gz
```

其中，loglistener-linux-x64-x.tar.gz中的 x 代表对应升级到的 LogListener 版本号（例如2.7.2）。LogListener 最新版本以 LogListener 安装指南 版本为准。如果输入的版本不存在，则下载失败。如果输入的版本低于机器上已安装的当前版本，则更新不生效。

LogListener 服务日志

最近更新时间：2022-04-11 10:18:35

概述

LogListener 服务日志功能支持记录 LogListener 端运行状态和采集监控的日志数据并配置可视化视图，提供重要指标数据，便于用户观测了解 LogListener 的运行状态和日志采集统计情况。

默认配置

默认配置项	配置内容
日志主题	当 LogListener 服务日志开启时，会自动为您创建一个 cls_service_logging 日志集，将所有关联的机器组所产生的日志数据都分类保存到对应的日志主题中。默认为您创建以下3个日志主题： • loglistener_status：对应 LogListener 的心跳状态日志。 • loglistener_alarm：对应 LogListener 的采集指标/错误类型监控日志。 • loglistener_business ：对应 LogListener 的采集操作日志，每条日志对应一次请求。
地域	开启 LogListener 日志服务时，默认在同地域机器组下创建日志集和日志主题 。
日志存储时间	默认保存7天，不支持修改存储时间。
索引	默认为采集到的所有日志数据开启全文索引和键值索引。支持修改索引配置，详情请查看 配置索引。
仪表盘	默认创建一个同地域下的仪表盘 service_log_dashboard。

> 说明：
>
> • LogListener 服务日志专属用于 LogListener 采集监控产生的日志，不支持写入其他数据。
> • LogListener 服务日志功能产生的日志数据不产生费用。
> • cls_service_logging 为统一的 LogListener 服务日志的日志集

应用场景

- **查看 LogListener 状态**
  开通 LogListener 服务日志功能后，您可以查看 LogListener 运行状态和采集统计情况。用户可以通过 service_log_dashboard 仪表盘，查看活跃 LogListener 数、LogListener 状态分布等统计指标。
- **采集端监控配置**
  您可以按指标/错误类型，配置采集端监控指标，例如：
  - 根据 MEM、CPU、采集速度、采集延时等指标进行监控。
  - 根据 LogListener 解析错误次数的维度进行监控。
- **文件级监控**
  开通 Loglistener 服务日志功能后，您可以查看文件及目录的监控日志，例如：
  - 某个 IP 上所有文件的采集统计文件。
  - 某个 IP 上某个路径下的采集日志量情况，如 app1 应用日志位于/var/log/app1/，统计这个路径下的日志采集情况。
  - 某个 topic 的采集统计情况。

前提条件

配置机器 IP/标识机器组仅在 LogListener 2.5.4 及以上版本支持采集监控服务日志，您可前往升级至 最新版本。

操作步骤

开通服务日志

1. 登录 日志服务控制台。
2. 在左侧导航栏中单击**机器组管理**，进入机器组列表页。

3. 在机器组列表页，选择目标机器组，单击 ，即可开启 LogListener 服务日志。

关闭服务日志

1. 日志服务控制台。
2. 在左侧导航栏中单击**机器组管理**，进入机器组列表页。
3. 在机器组列表页，选择目标机器组，单击 ，即可关闭 LogListener 服务日志。

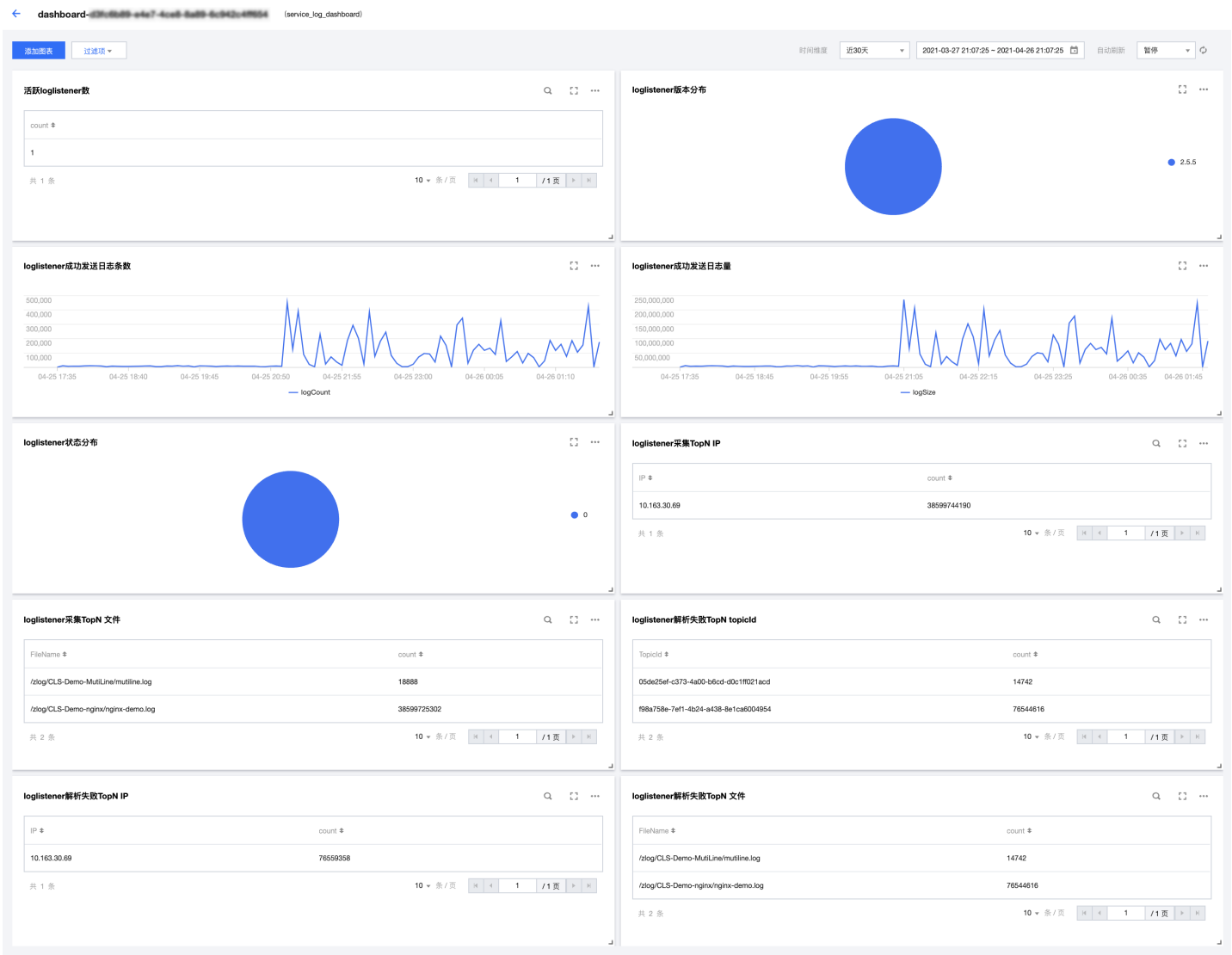
> **说明：**
> 关闭服务日志功能后，日志集 cls_service_logging 中保存的日志数据不会自动删除，如果您需要删除这部分日志数据，可以手动删除保存服务日志的日志集。

服务日志仪表盘

开启 LogListener 服务日志后，日志服务系统会根据记录的日志类型自动创建可视化仪表盘 service_log_dashboard，展示 LogListener 采集监控统计。

采集统计仪表盘

您可以在日志服务控制台的 仪表盘 页面，单击目标仪表盘的 ID，查看 LogListener 采集相关的统计信息，包括 LogListener 状态展示、LogListener 解析失败率、LogListener 发送成功率等指标信息。



日志类型

LogListener 状态日志

日志主题 Loglistener_status 的字段具体说明如下：

字段	描述
InstanceId	LogListener 唯一标识值
IP	机器组 IP
Label	机器标识数组
Version	版本号
MemoryUsed	组件内存使用情况
MemMax	Agent 在该机器上设置的内存使用阈值
CpuUsage	组件 CPU 使用率
Status	LogListener 运行状态
TotalSendLogSize	发送日志量大小
SendSuccessLogSize	发送成功日志量大小
SendFailureLogSize	发送失败日志量大小

SendTimeoutLogSize	发送超时日志量大小
TotalParseLogCount	解析总日志条数
ParseFailureLogCount	解析失败日志条数
TotalSendLogCount	总发送日志条数
SendSuccessLogCount	发送成功日志条数
SendFailureLogCount	发送失败日志条数
SendTimeoutLogCount	发送超时日志条数
TotalSendReqs	总发送请求数
SendSuccessReqs	发送成功请求数
SendFailureReqs	发送失败请求数
SendTimeoutReqs	发送超时请求数
TotalFinishRsps	收到的全部 .rsp 文件
TotalSuccessFromStart	LogListener 启动到现在总的成功数
AvgReqSize	平均请求包大小
SendAvgCost	平均发送耗时
AvailConnNum	可用连接数
QueueSize	排队请求大小

LogListener 告警日志

日志主题 Loglistener_alarm 的字段具体说明如下：

监控指标分类	描述
InstanceId	LogListener 唯一标识值
Label	机器标识数组
IP	机器组 IP
Version	LogListener 版本
AlarmType.count	告警类型统计
AlarmType.example	告警类型样例

AlarmType：

alarm type	type ID	描述
UnknownError	0	初始化 alarm 类型
UnknownError	1	解析失败
CredInvalid	2	认证失败
SendFailure	3	发送失败
RunException	4	LogListener 运行异常
MemLimited	5	触发 mem limited 限制
FileProcException	6	文件处理异常
FilePosGetError	7	获取 file pos 失败
HostIpException	8	host IP 线程异常

alarm type	type ID	描述
StatException	9	获取进程相关信息异常
UpdateException	10	cls update 功能异常
DoSendError	11	dosend 失败
FileAddError	12	文件新增失败
FileMetaError	13	元数据文件新增失败
FileOpenError	14	open file 失败
FileReadError	15	read file 失败
FileStatError	16	stat file 失败
GetTimeError	17	getTimeFromLogContent 失败
HandleEventError	18	handle file event 异常
HandleFileCreateError	19	handleFileCreateEvent() 异常
LineParseError	20	log item 解析失败
Lz4CompressError	21	压缩失败
ReadEventException	22	readEvent 失败
ReadFileBugOn	23	触发 bugon
ReadFileException	24	procReadyFile() 异常
ReadFileInodeChange	25	file inode 发生变化
ReadFileTruncate	26	Readfile 截断
WildCardPathException	27	addWildcardPathInotify() 异常

LogListener 采集日志

日志主题 Loglistener_business 的字段具体说明如下：

字段	描述
InstanceId	LogListener 唯一标识值
Label	机器标识数组
IP	机器组 IP
Version	LogListener 版本
TopicId	文件采集到的目标 topic
FileName	文件路径名
RealPath	文件实际路径
FileInode	文件 inode
FileSize	文件大小
LastReadTime	上次读取文件时间
ParseFailLines	时间窗口，解析失败日志条数
ParseFailSize	时间窗口，解析失败日志大小
ParseSuccessLines	时间窗口，解析成功日志条数
ParseSuccessSize	时间窗口，解析成功日志大小

ReadOffset	读取文件的偏移量，单位字节
TruncateSize	时间窗口内，truncate 的文件大小
ReadAvgDelay	时间窗口内，读取平均时延
TimeFormatFailuresLines	时间窗口内，时间戳匹配错误次数
SendSuccessSize	时间窗口内，发送成功日志大小
SendSuccessCount	时间窗口内，发送成功日志条数
SendFailureSize	时间窗口内，发送失败日志大小
SendFailureCount	时间窗口内，发送失败日志条数
SendTimeoutSize	时间窗口内，发送超时日志大小
SendTimeoutCount	时间窗口内，发送超时日志条数
DroppedLogSize	时间窗口内，丢掉日志大小
DroppedLogCount	时间窗口内，丢掉日志条数
ProcessBlock	标记一个统计周期内，当前文件是否触发过采集阻塞（一个文件的滑动窗口10分钟未移动过，即为触发）

采集文本日志
单行全文格式

最近更新时间：2021-08-13 09:14:12

操作场景

单行全文日志是指一条日志仅包含一行的内容，在采集的时候，将使用换行符 \n 来作为一条日志的结束符。为了统一结构化管理，每条日志都会存在一个默认的键值 __CONTENT__，但日志数据本身不再进行日志结构化处理，也不会提取日志字段，日志属性的时间项由日志采集的时间决定。

前提条件

假设您的一条日志原始数据为：

```
Tue Jan 22 12:08:15 CST 2019 Installed: libjpeg-turbo-static-1.2.90-6.el7.x86_64
```

日志最终被日志服务处理为：

```
__CONTENT__:Tue Jan 22 12:08:15 CST 2019 Installed: libjpeg-turbo-static-1.2.90-6.el7.x86_64
```

操作步骤

登录控制台

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击【日志主题】，进入日志主题管理页面。

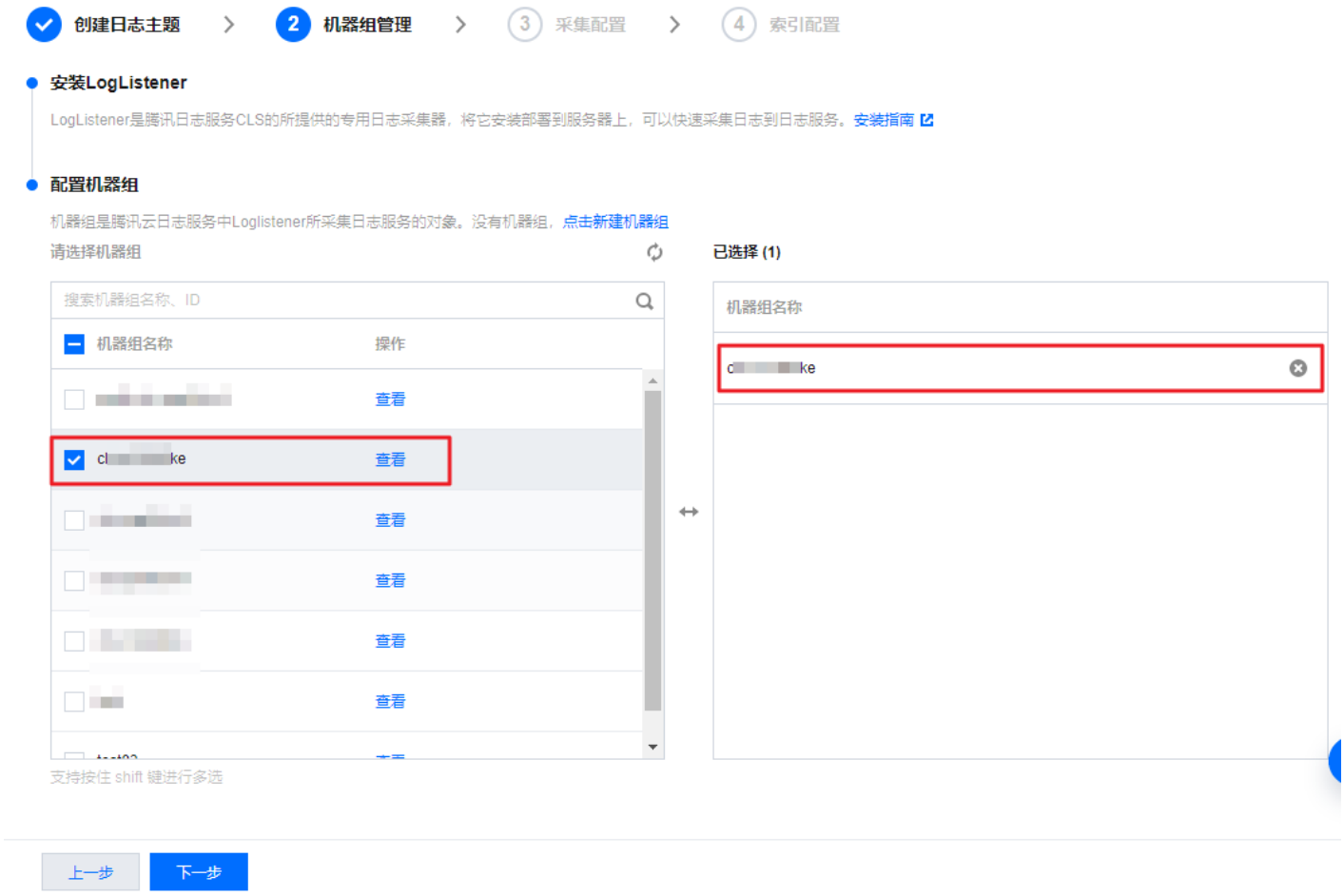
创建日志主题

1. 单击【创建日志主题】。
2. 在弹出的对话框中，将“日志主题名称”填写为“test_full”，单击【确定】，即可新增日志主题。如下图所示：

机器组管理

1. 日志主题创建成功后，进入该日志主题管理页面。
2. 选择【采集配置】页签，单击您需要采集的日志数据源格式。

3. 在“机器组管理”页面，勾选需要与当前日志主题进行绑定的机器组，单击【下一步】。
即可进入采集配置阶段，更多详情请参阅 管理机器组。



采集配置

配置日志文件采集路径

在“采集配置”页面，根据日志采集路径格式，填写“采集路径”。如下图所示：
日志采集路径格式：[目录前缀表达式]/**/[文件名表达式]。

填写日志采集路径后，LogListener 会按照**[目录前缀表达式]**匹配所有符合规则的公共前缀路径，并监听这些目录（包含子层目录）下所有符合**[文件名表达式]**规则的日志文件。其参数详细说明如下：

字段	说明
目录前缀	日志文件前缀目录结构，仅支持通配符 * 和 ？ • * 表示匹配多个任意字符 • ？表示匹配单个任意字符
/**/	表示当前目录以及所有子目录
文件名	日志文件名，仅支持通配符 * 和 ？， • * 表示匹配多个任意字符 • ？表示匹配单个任意字符

常用的配置模式如下：

- [公共目录前缀]/**/[公共文件名前缀]*
- [公共目录前缀]/**/*[公共文件名后缀]
- [公共目录前缀]/**/[公共文件名前缀]*[公共文件名后缀]
- [公共目录前缀]/**/*[公共字符串]*

填写示例如下：

序号	目录前缀表达式	文件名表达式	说明
1.	/var/log/nginx	access.log	此例中，日志路径配置为/var/log/nginx/**/access.log，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以access.log命名的日志文件
2.	/var/log/nginx	*.log	此例中，日志路径配置为/var/log/nginx/**/*.log，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以.log结尾的日志文件
3.	/var/log/nginx	error*	此例中，日志路径配置为/var/log/nginx/**/error*，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以error开头命名的日志文件

⚠ 注意：

- Loglistener 2.3.9及以上版本才可以添加多个采集路径。
- 暂不支持上传的日志内容中含有多种文本格式，可能会导致写入失败，例如 key:"{"substream":XXX}"。
- 建议配置采集路径为 log/*.log，rename日志轮转后的老文件命名为 log/*.log.xxxx。
- 默认情况下，一个日志文件只能被一个日志主题采集。如果一个文件需要对应多个采集配置，请给源文件添加一个软链接，并将其加到另一组采集配置中。

配置单行全文格式

在“采集配置”页面，将“提取模式”设置为【单行全文】。如下图所示：

配置采集策略

- 全量采集：Loglistener 采集文件时，从文件的开头开始读。
- 增量采集：Loglistener 采集文件时，从距离文件末尾1M处开始读取（若文件小于1M，等价全量采集）。

配置过滤器条件

过滤器旨在您根据业务需要添加日志采集过滤规则，帮助您筛选出有价值的日志数据。过滤规则为 Perl 正则表达式，所创建的过滤规则为命中规则，即匹配上正则表达式的日志才会被采集上报。

单行全文模式下，默认使用 __CONTENT__ 作为全文的键（key）名。例如，单行全文日志样例格式为 Tue Jan 22 12:08:15 CST 2019 Installed: libjpeg-turbo-static-1.2.90-6.el7.x86_64，您希望采集1月22号这一天的所有日志，则 key 处填写 __CONTENT__，过滤规则配置 Tue Jan 22.* 。

> **注意：**
> 多条过滤规则之间关系是"与"逻辑；若同一 key 名配置多条过滤规则，规则会被覆盖。

索引配置

1. 单击【下一步】，进入“索引配置”页面。
2. 在“索引配置”页面，设置如下信息。

- 索引状态：确认是否开启。
- 全文索引：确认是否需要设置大小写敏感。
- 全文分词符：默认为“@&()='",;:<>[]{}/ \n\t\r”，确认是否需要修改。
- 键值索引：默认关闭，您可根据 key 名按需进行字段类型、分词符以及是否开启统计分析的配置。若您需要开启键值索引，可将  设置为  。

> **注意：**
> 检索必须开启索引配置，否则无法检索。

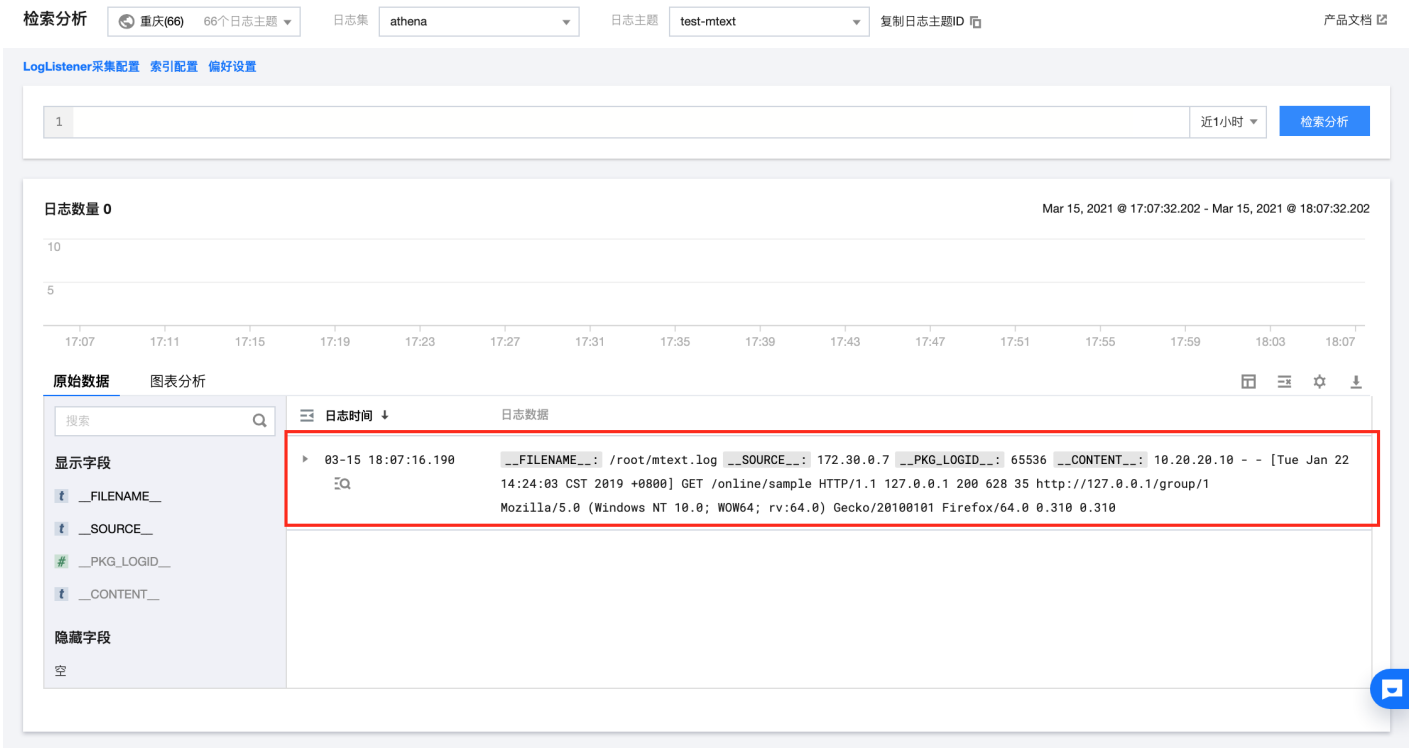
3. 单击【提交】，完成采集配置。

相关操作

检索日志

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击【检索分析】，进入检索分析页面。

3. 根据实际需求，选择地域、日志集与日志主题，单击【检索分析】，即可开始按照设定的查询条件检索日志。如下图所示：



多行全文格式

最近更新时间：2022-05-24 17:44:36

操作场景

多行全文日志是指一条完整的日志数据跨占多行（例如 Java 程序日志）。在这种情况下，以换行符 \n 为日志的结束标识符就显得有些不合理，为了能让日志系统明确区分开每条日志，采用首行正则的方式进行匹配，当某行日志匹配上预先设置的正则表达式，就认为是一条日志的开头，而下一个行首出现作为该条日志的结束标识符。

多行全文也会设置一个默认的键值 __CONTENT__，但日志数据本身不再进行日志结构化处理，也不会提取日志字段，日志属性的时间项由日志采集的时间决定。

前提条件

假设您的一条多行日志原始数据为：

```
10.20.20.10 - - [Tue Jan 22 14:24:03 CST 2019 +0800] GET /online/sample HTTP/1.1 127.0.0.1 200 628 35 http://127.0.0.1/group/1
Mozilla/5.0 (Windows NT 10.0; WOW64; rv:64.0) Gecko/20100101 Firefox/64.0 0.310 0.310
```

该条日志最终被日志服务结构化处理为：

```
__CONTENT__:10.20.20.10 - - [Tue Jan 22 14:24:03 CST 2019 +0800] GET /online/sample HTTP/1.1 127.0.0.1 200 628 35 http://127.0.0.1/group/1
\nMozilla/5.0 (Windows NT 10.0; WOW64; rv:64.0) Gecko/20100101 Firefox/64.0 0.310 0.310
```

操作步骤

登录控制台

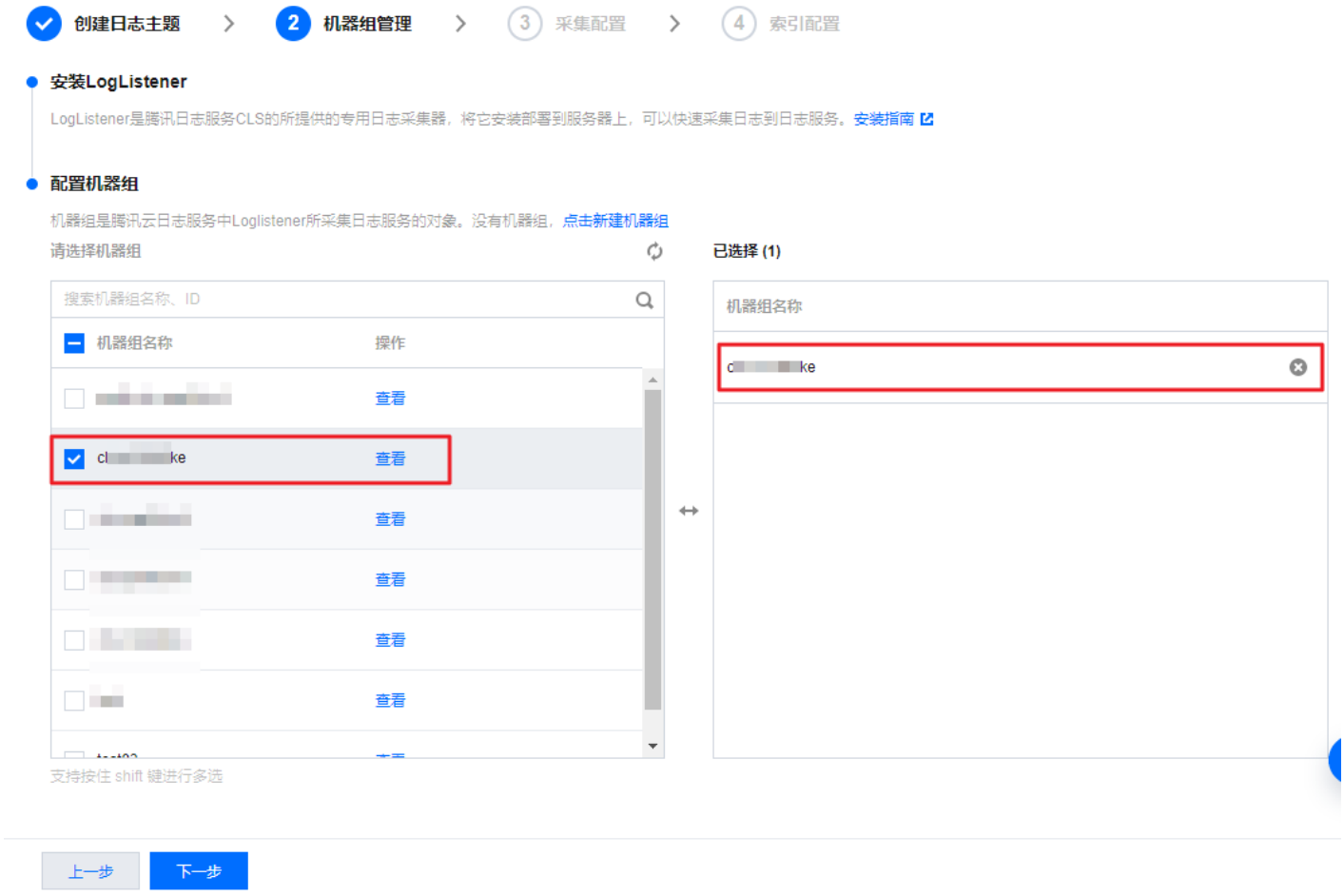
1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**，进入日志主题管理页面。

创建日志主题

1. 单击**创建日志主题**。
2. 在弹出的对话框中，将“日志主题名称”填写为“test-mtext”，单击**确定**，即可新增日志主题。如下图所示：

机器组管理

1. 日志主题创建成功后，进入该日志主题管理页面。
2. 选择**采集配置**页签，在“LogListener采集配置”单击**新增**，选择需要采集的日志数据源格式。
3. 在“机器组管理”页面，勾选需要与当前日志主题进行绑定的机器组，单击**下一步**。
   即可进入采集配置阶段，更多详情请参阅 管理机器组。



采集配置

配置日志文件采集路径

在“采集配置”页面，填写采集规则名称，并根据日志采集路径格式填写“采集路径”。如下图所示：

日志采集路径格式：[目录前缀表达式]//[文件名表达式]。**

填写日志采集路径后，LogListener 会按照[目录前缀表达式]匹配所有符合规则的公共前缀路径，并监听这些目录（包含子层目录）下所有符合[文件名表达式]规则的日志文件。其参数详细说明如下：

字段	说明
目录前缀	日志文件前缀目录结构，仅支持通配符 * 和 ？ • * 表示匹配多个任意字符 • ？表示匹配单个任意字符
/**/	表示当前目录以及所有子目录

字段	说明
文件名	日志文件名，仅支持通配符 * 和 ？， • * 表示匹配多个任意字符 • ？表示匹配单个任意字符

常用的配置模式如下：

- [公共目录前缀]/**/[公共文件名前缀]*
- [公共目录前缀]/**/*[公共文件名后缀]
- [公共目录前缀]/**/[公共文件名前缀]*[公共文件名后缀]
- [公共目录前缀]/**/*[公共字符串]*

填写示例如下：

序号	目录前缀表达式	文件名表达式	说明
1.	/var/log/nginx	access.log	此例中，日志路径配置为 /var/log/nginx/**/access.log，LogListener 将会监听 /var/log/nginx 前缀路径下所有子目录中以 access.log 命名的日志文件
2.	/var/log/nginx	*.log	此例中，日志路径配置为 /var/log/nginx/**/*.log，LogListener 将会监听 /var/log/nginx 前缀路径下所有子目录中以 .log 结尾的日志文件
3.	/var/log/nginx	error*	此例中，日志路径配置为 /var/log/nginx/**/error*，LogListener 将会监听 /var/log/nginx 前缀路径下所有子目录中以 error 开头命名的日志文件

⚠ 注意：

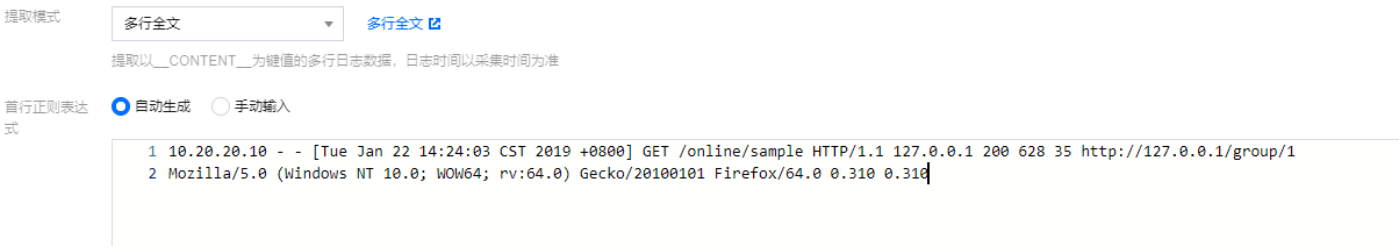
- Loglistener 2.3.9及以上版本才可以添加多个采集路径。
- 暂不支持上传的日志内容中含有多种文本格式，可能会导致写入失败，例如 key:"{"substream":XXX}"。
- 建议配置采集路径为 log/*.log，rename日志轮转后的老文件命名为 log/*.log.xxxx。
- 默认情况下，一个日志文件只能被一个日志主题采集。如果一个文件需要对应多个采集配置，请给源文件添加一个软链接，并将其加到另一组采集配置中。

配置采集策略

- 全量采集：Loglistener 采集文件时，从文件的开头开始读。
- 增量采集：Loglistener 采集文件时，从距离文件末尾1M处开始读取（若文件小于1M，等价全量采集）。

配置多行全文格式

1. 在“采集配置”页面，将“提取模式”设置为**多行全文**。如下图所示：



2. 根据如下规则，定义正则表达式。
   您可选择**自动生成**或者**手动输入**两种方式定义首行正则表达式，系统会根据样例内容进行正则表达式的验证。

- **自动生成：在文本框中，输入日志样例，单击自动生成，系统自动在置灰的文本框中生成行首正则表达式。如下图所示：**

- **手动输入：在文本框中，输入日志样例，输入首行正则表达式，单击验证，系统将判断表达式是否通过。如下图所示：**

配置过滤器条件

过滤器旨在您根据业务需要添加日志采集过滤规则，帮助您筛选出有价值的日志数据。过滤规则为 Perl 正则表达式，所创建的过滤规则为命中规则，即匹配上正则表达式的日志才会被采集上报。

多行全文模式下，默认使用 __CONTENT__ 作为全文的键（key）名。例如，多行全文日志样例格式为：

```
10.20.20.10 - - [Tue Jan 22 14:24:03 CST 2019 +0800] GET /online/sample HTTP/1.1 127.0.0.1 200 628 35 http://127.0.0.1/group/1
Mozilla/5.0 (Windows NT 10.0; WOW64; rv:64.0) Gecko/20100101 Firefox/64.0 0.310 0.310
```

您希望采集10.20.20.10这台机器的所有日志，则 key 处填写 __CONTENT__，过滤规则配置 10.20.20.10.*。

> **注意：**
> **多条过滤规则之间关系是"与"逻辑；若同一 key 名配置多条过滤规则，规则会被覆盖。**

配置上传解析失败日志

建议开启上传解析失败日志。开启后，Loglistener 会上传各式解析失败的日志。若关闭上传解析失败日志，则会丢弃失败的日志。

开启后需要配置解析失败的 Key 值（默认为 LogParseFailure），所有解析失败的日志，均以输入内容作为键名称（Key），原始日志内容作为值（Value）进行上传。

索引配置

1. 单击下一步，进入“索引配置”页面。

2. 在“索引配置”页面，设置如下信息。

- 索引状态：确认是否开启。

  > ⚠ 注意：
  > 检索必须开启索引配置，否则无法检索。

- 全文索引：确认是否需要设置大小写敏感。
- 全文分词符：默认为“@&()='",;:<>[]{}/ \n\t\r”，确认是否需要修改。
- 键值索引：默认关闭，您可根据 key 名按需进行字段类型、分词符以及是否开启统计分析的配置。若您需要开启键值索引，可将  设置为  。

3. 单击**提交**，完成采集配置。

相关操作

检索日志，请参见 检索分析 文档。

完全正则格式（单行）

最近更新时间：2022-05-24 18:03:51

操作场景

单行-完全正则模式适用于日志文本中每行内容为一条原始日志，且每条日志可按正则表达式提取为多个 key-value 键值的日志解析模式。若不需要提取 key-value，请参阅 单行全文格式 进行配置。
配置单行-完全正则模式时，您需要先输入日志样例，再自定义正则表达式。配置完成后，系统将根据正则表达式中的捕获组提取对应的 key-value。
如下内容将为您详细介绍如何采集单行-完全正则模式日志。

前提条件

假设您的一条日志原始数据为：

```
10.135.46.111 - - [22/Jan/2019:19:19:30 +0800] "GET /my/course/1 HTTP/1.1" 127.0.0.1 200 782 9703 "http://127.0.0.1/course/explore?filter%5Btype%5D=all&filter%5Bprice%5D=all&filter%5BcurrentLevelId%5D=all&orderBy=studentNum" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:64.0) Gecko/20100101 Firefox/64.0" 0.354 0.354
```

配置的自定义正则表达式为：

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

系统根据 () 捕获组提取对应的 key-value 后，您可以自定义每组的 key 名称如下所示：

```
body_bytes_sent: 9703
http_host: 127.0.0.1
http_protocol: HTTP/1.1
http_referer: http://127.0.0.1/course/explore?filter%5Btype%5D=all&filter%5Bprice%5D=all&filter%5BcurrentLevelId%5D=all&orderBy=studentNum
http_user_agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:64.0) Gecko/20100101 Firefox/64.0
remote_addr: 10.135.46.111
request_length: 782
request_method: GET
request_time: 0.354
request_url: /my/course/1
status: 200
time_local: [22/Jan/2019:19:19:30 +0800]
upstream_response_time: 0.354
```

操作步骤

登录控制台

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**，进入日志主题管理页面。

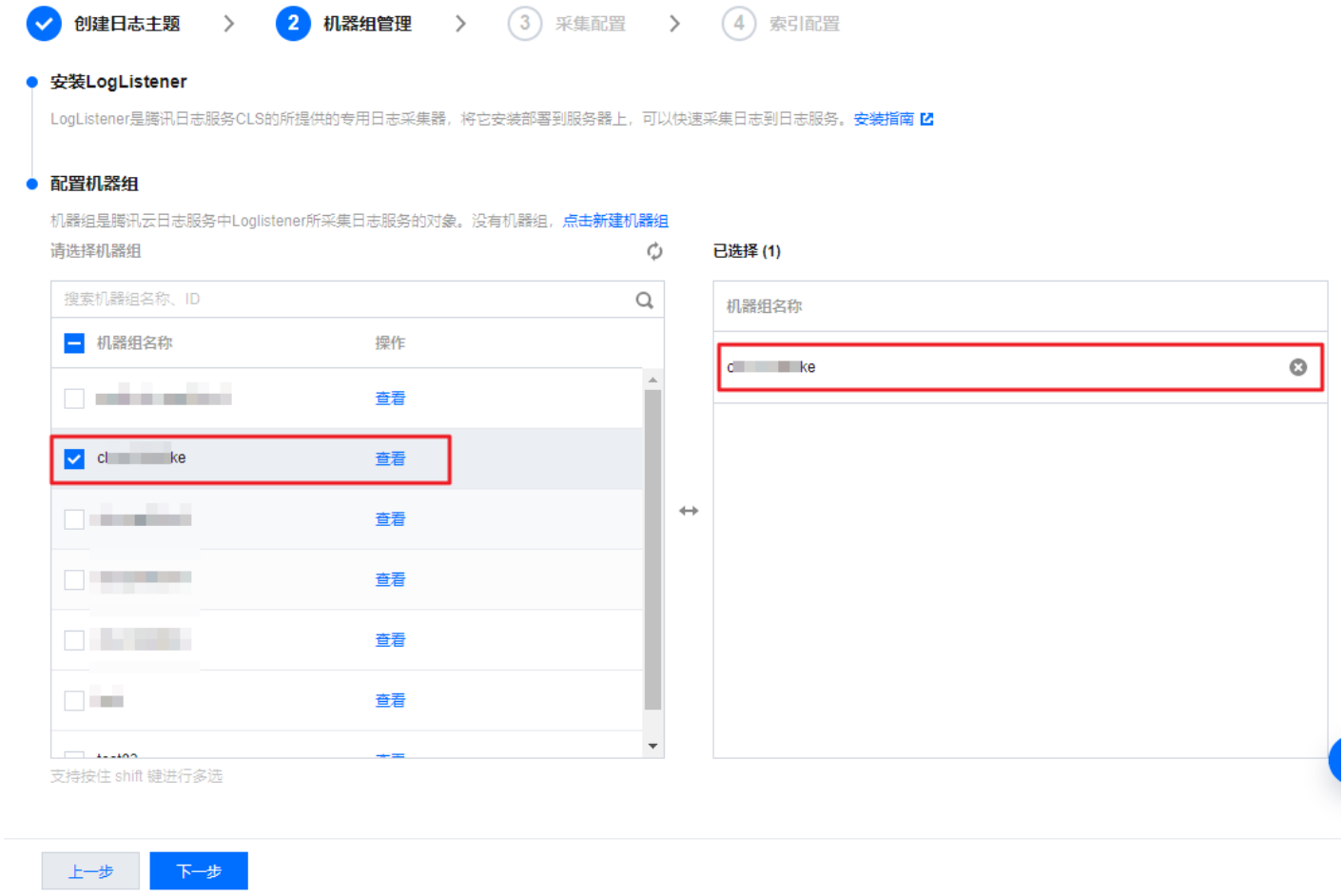
新增日志主题

1. 单击**创建日志主题**。

2. 在弹出的对话框中，将“日志主题名称”填写为“test-whole”，单击**确定**，即可新增日志主题。如下图所示：

机器组管理

1. 日志主题创建成功后，进入该日志主题管理页面。
2. 选择**采集配置**页签，在“LogListener采集配置”单击**新增**，选择需要采集的日志数据源格式。
3. 在“机器组管理”页面，勾选需要与当前日志主题进行绑定的机器组，单击**下一步**。
   即可进入采集配置阶段，更多详情请参阅 管理机器组。



采集配置

配置日志文件采集路径

在“采集配置”页面，填写采集规则名称，并根据日志采集路径格式填写“采集路径”。如下图所示：

日志采集路径格式：[目录前缀表达式]/**/[文件名表达式]。

填写日志采集路径后，LogListener 会按照[目录前缀表达式]匹配所有符合规则的公共前缀路径，并监听这些目录（包含子层目录）下所有符合[文件名表达式]规则的日志文件。其参数详细说明如下：

字段	说明
目录前缀	日志文件前缀目录结构，仅支持通配符 * 和 ？ • * 表示匹配多个任意字符 • ？表示匹配单个任意字符
/**/	表示当前目录以及所有子目录

字段	说明
文件名	日志文件名，仅支持通配符 * 和 ？， • * 表示匹配多个任意字符 • ？表示匹配单个任意字符

常用的配置模式如下：

- [公共目录前缀]/**/[公共文件名前缀]*
- [公共目录前缀]/*/[公共文件名后缀]
- [公共目录前缀]/**/[公共文件名前缀]*[公共文件名后缀]
- [公共目录前缀]/**/*[公共字符串]*

填写示例如下：

序号	目录前缀表达式	文件名表达式	说明
1.	/var/log/nginx	access.log	此例中，日志路径配置为/var/log/nginx/**/access.log，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以access.log命名的日志文件
2.	/var/log/nginx	*.log	此例中，日志路径配置为 /var/log/nginx/**/*.log，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以 .log 结尾的日志文件
3.	/var/log/nginx	error*	此例中，日志路径配置为/var/log/nginx/**/error*，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以error开头命名的日志文件

⚠ 注意：

- Loglistener 2.3.9及以上版本才可以添加多个采集路径。
- 暂不支持上传的日志内容中含有多种文本格式，可能会导致写入失败，例如 key:"{"substream":XXX}"。
- 建议配置采集路径为 log/*.log，rename日志轮转后的老文件命名为 log/*.log.xxxx。
- 默认情况下，一个日志文件只能被一个日志主题采集。如果一个文件需要对应多个采集配置，请给源文件添加一个软链接，并将其加到另一组采集配置中。

配置采集策略

- 全量采集：Loglistener 采集文件时，从文件的开头开始读。
- 增量采集：Loglistener 采集文件时，从距离文件末尾1M处开始读取（若文件小于1M，等价全量采集）。

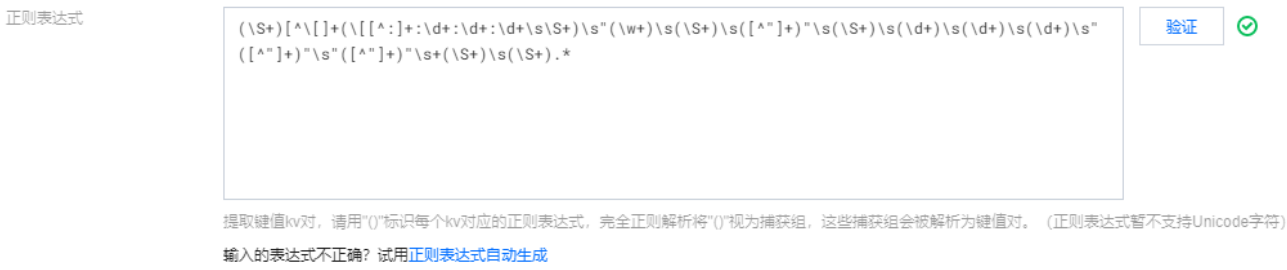
配置单行-完全正则模式

1. 在“采集配置”页面，将“提取模式”设置为**单行-完全正则**，并在“日志样例”文本框中，输入日志样例。如下图所示：

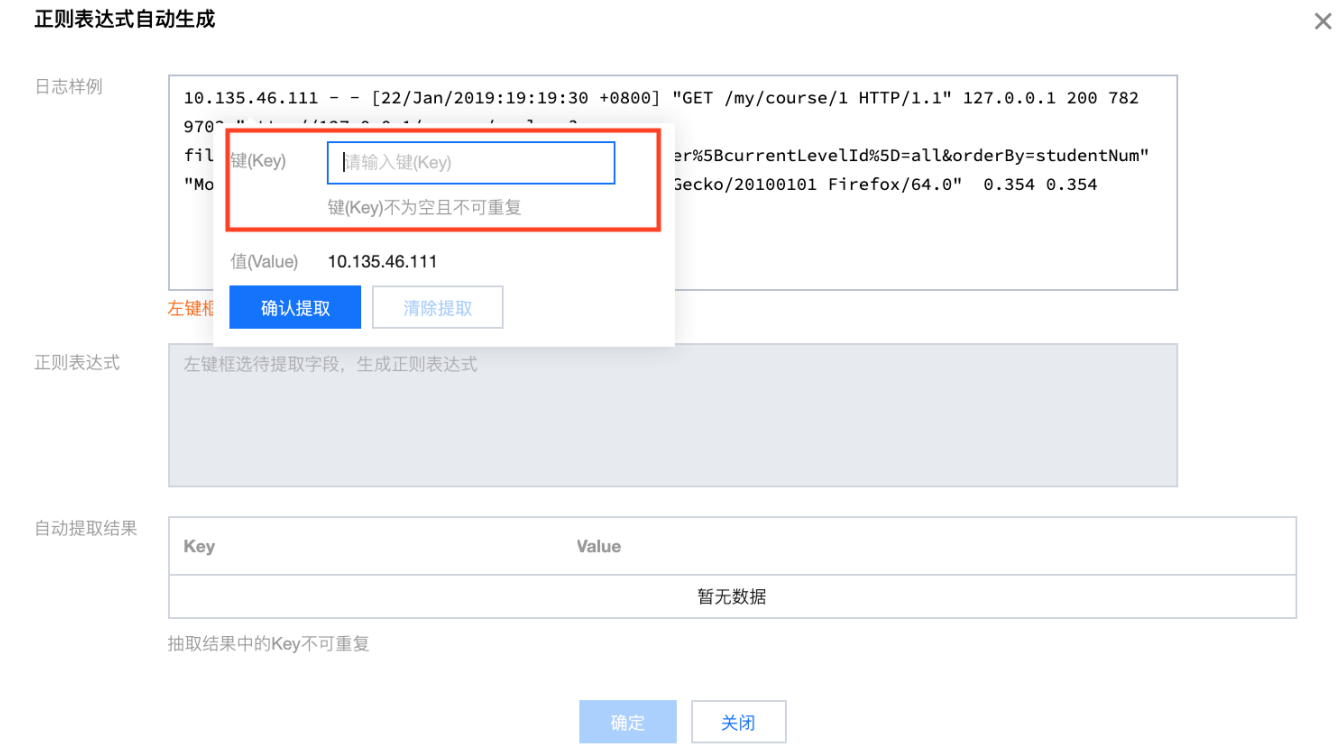
2. 根据如下规则，定义正则表达式。

   系统有**手动模式**和**自动模式**两种方式定义正则表达式。您可手动输入表达式提取 key-value 进行验证，也可单击**正则表达式自动生成**切换为自动模式。系统会根据您选择的模式以及定义好的正则表达式，提取 key-value 进行正则表达式的验证。

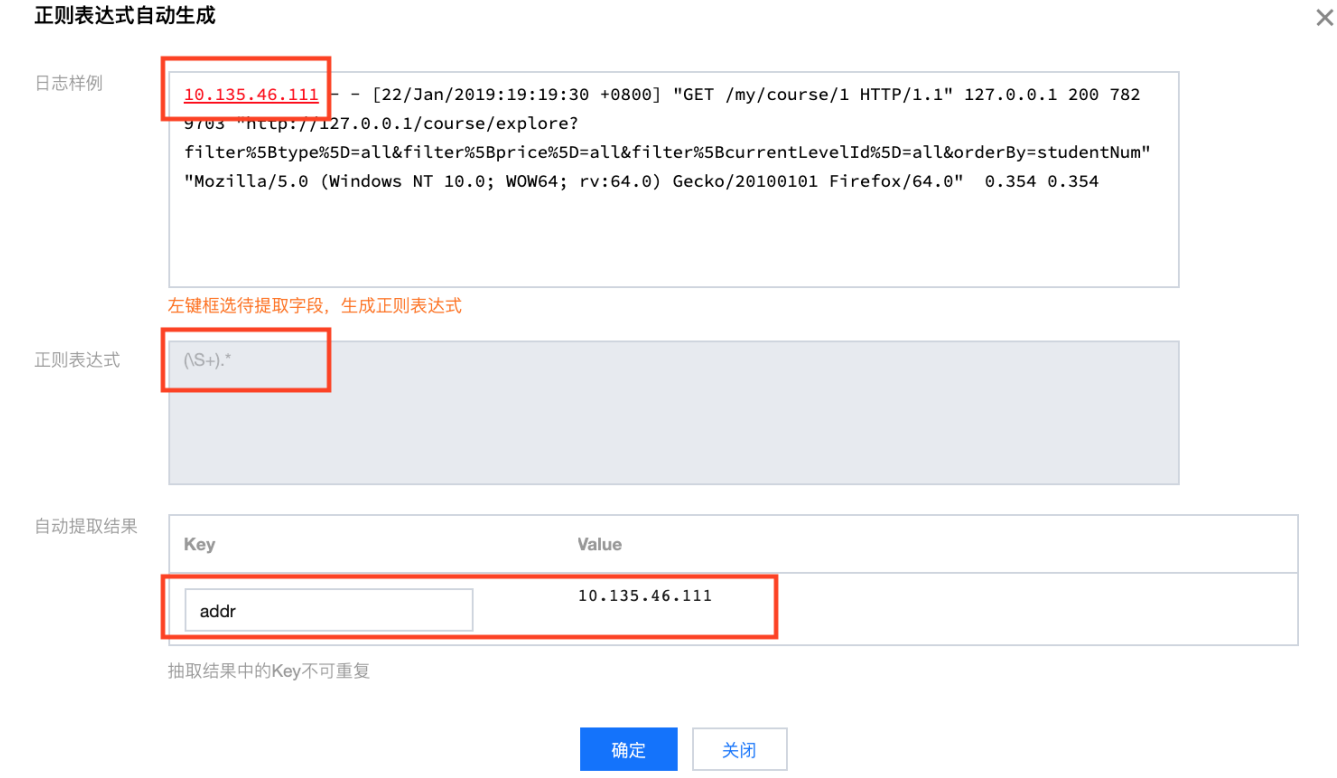
   - **手动模式**：



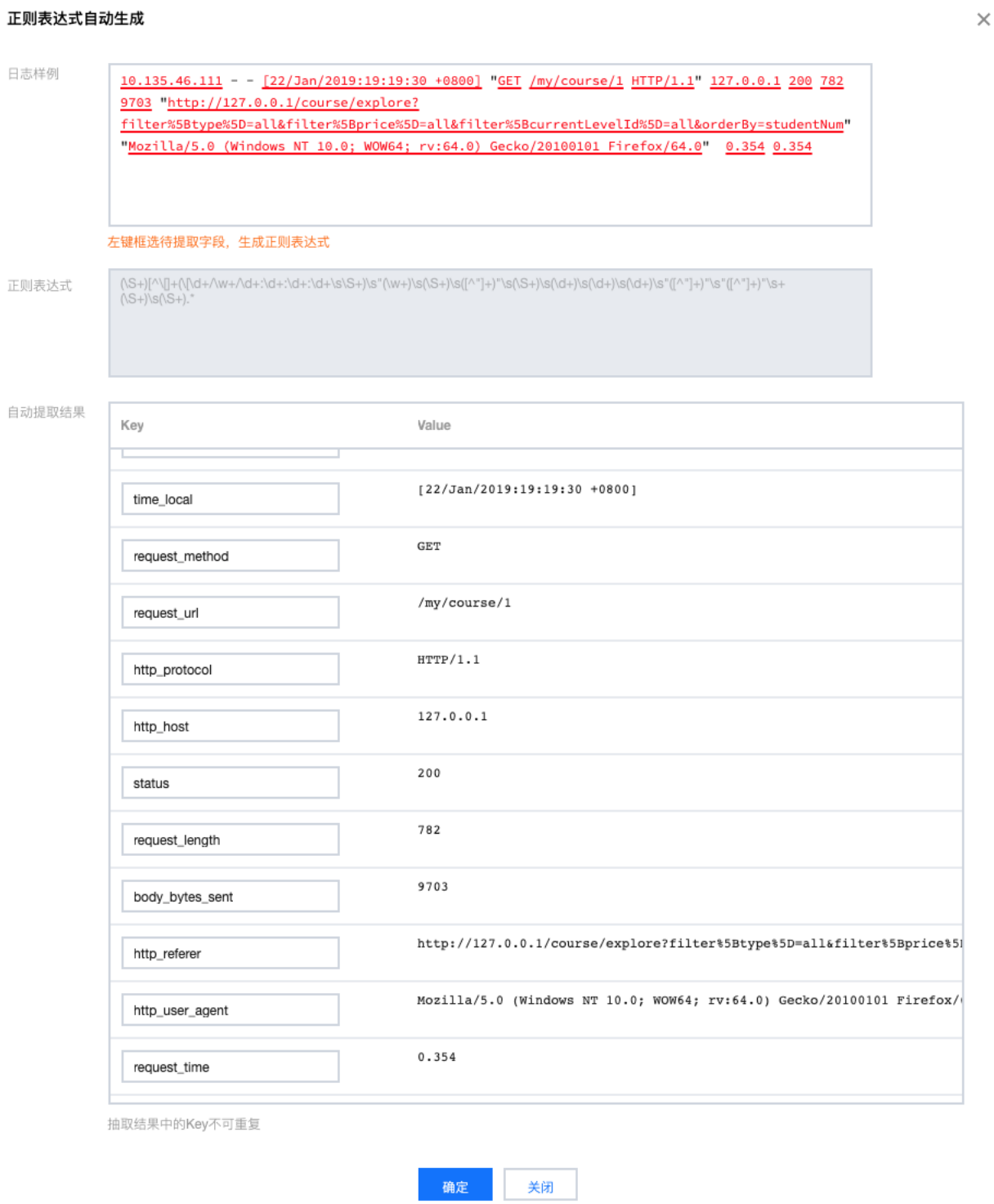
     a. 在“正则表达式”的文本框中，输入正则表达式。
     b. 单击**验证**，系统将判断日志样例与正则表达式是否匹配。
   - **自动模式**（单击**正则表达式自动生成**进行切换）：
     a. 在弹出的“正则表达式自动生成”模态视图中，根据实际的检索分析需求，选中需要提取 key-value 的日志内容，并在弹出的文本框中，输入键(key)名，单击**确认提取**。如下图所示：



系统将自动对该部分内容提取一个正则表达式，自动提取结果会出现在 key-value 表格中。如下图所示：



b. 重复 步骤 a，直到提取完所有的 key-value 对。如下图所示：



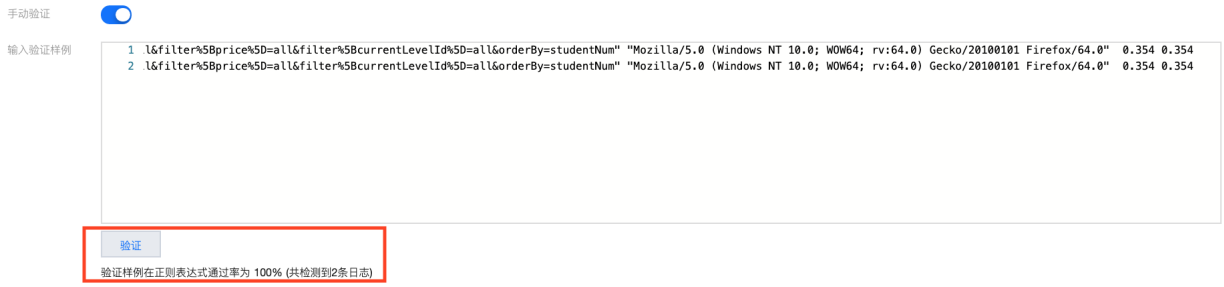
c. 单击**确定**，系统将根据提取好的 key-value 对自动生成完整的正则表达式。

> **说明：**
> 无论选择自动模式还是手动模式，正则提取模式均在完成定义并验证通过后，将提取结果展示在“抽取结果”中。您只需定义每一组 key-value 对的 key 名称，即可将该名称用于日志检索分析。

手动验证

1. 当您的日志数据复杂时，可以将“手动验证”设置为  ，即可开启手动验证。

2. 输入多个日志样例，单击**验证**。系统将验证样例正则表达式的通过率。



配置采集时间

- 日志时间单位为：毫秒。
- 日志的时间属性有如下方式：
  - 采集时间：默认作为日志的时间属性。

  - 原始时间戳：将“使用采集时间”设置为  ，并填写原始时间戳的时间键以及对应的时间解析格式。
    时间解析格式请参见 配置时间格式。

- 采集时间：日志的时间属性由日志服务 CLS 采集该条日志的时间决定。
- 原始时间戳：日志的时间属性由原始日志中时间戳决定。

时间格式解析规则填写的示例如下：

- 示例1：
  日志样例原始时间戳：10/Dec/2017:08:00:00.000，解析格式为：%d/%b/%Y:%H:%M:%S.%f。
- 示例2：
  日志样例原始时间戳：2017-12-10 08:00:00.000，解析格式为：%Y-%m-%d %H:%M:%S.%f 。
- 示例3：
  日志样例原始时间戳：12/10/2017, 08:00:00.000，解析格式为：%m/%d/%Y, %H:%M:%S.%f。

> ⚠ 注意：
> 日志时间支持以毫秒为单位，若时间格式填写错误日志时间将以采集时间为准。

配置过滤器条件

过滤器旨在您根据业务需要添加日志采集过滤规则，帮助您筛选出有价值的日志数据。过滤规则为 Perl 正则表达式，所创建的过滤规则为命中规则，即匹配上正则表达式的日志才会被采集上报。

完全正则采集时，需要根据所自定义的键值对来配置过滤规则。例如，样例日志使用完全正则模式解析后，您希望 status 字段为400或500的所有日志数据被采集，那么 key 处配置 status，过滤规则处配置 400|500。

> ⚠ 注意：
> 多条过滤规则之间关系是"与"逻辑，若同一 key 名配置多条过滤规则，规则会被覆盖。

配置上传解析失败日志

建议开启上传解析失败日志。开启后，Loglistener 会上传各式解析失败的日志。若关闭上传解析失败日志，则会丢弃失败的日志。

开启后需要配置解析失败的 Key 值（默认为 LogParseFailure），所有解析失败的日志，均以输入内容作为键名称（Key），原始日志内容作为值（Value）进行上传。

索引配置

1. 单击**下一步**，进入“索引配置”页面。
2. 在“索引配置”页面，设置如下信息。

- 索引状态：确认是否开启。

> ⚠ 注意：
> 检索必须开启索引配置，否则无法检索。

- 全文索引：确认是否需要设置大小写敏感。
- 全文分词符：默认为“@&()='",;:<>[]{}/ \n\t\r”，确认是否需要修改。
- 键值索引：默认关闭，您可根据 key 名按需进行字段类型、分词符以及是否开启统计分析的配置。若您需要开启键值索引，可将  设置为  。

3. 单击**提交**，完成采集配置。

相关操作

检索日志，请参见 检索分析 文档。

完全正则格式（多行）

最近更新时间：2022-05-24 18:14:26

操作场景

多行-完全正则模式适用于日志文本中一条完整的日志数据跨占多行（例如 Java 程序日志），可按正则表达式提取为多个 key-value 键值的日志解析模式。若不需要提取 key-value，请参阅 多行全文格式 进行配置。
配置多行-完全正则模式时，您需要先输入日志样例，再自定义正则表达式。配置完成后，系统将根据正则表达式中的捕获组提取对应的 key-value。

如下内容将为您详细介绍如何如何采集多行-完全正则模式日志。

> 注意：
> 多行-完全正则模式采集需升级至 Loglistener 2.4.5 版本，请前往 安装最新版本。

前提条件

假设您的一条日志原始数据为：

```
[2018-10-01T10:30:01,000] [INFO] java.lang.Exception: exception happened
at TestPrintStackTrace.f(TestPrintStackTrace.java:3)
at TestPrintStackTrace.g(TestPrintStackTrace.java:7)
at TestPrintStackTrace.main(TestPrintStackTrace.java:16)
```

行首正则表达式为：

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

配置的自定义正则表达式为：

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

系统根据 () 捕获组提取对应的 key-value 后，您可以自定义每组的 key 名称如下所示：

```
time： 2018-10-01T10:30:01,000`
level： INFO`
msg：java.lang.Exception: exception happened
at TestPrintStackTrace.f(TestPrintStackTrace.java:3)
at TestPrintStackTrace.g(TestPrintStackTrace.java:7)
at TestPrintStackTrace.main(TestPrintStackTrace.java:16)
```

操作步骤

登录控制台

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**，进入日志主题管理页面。

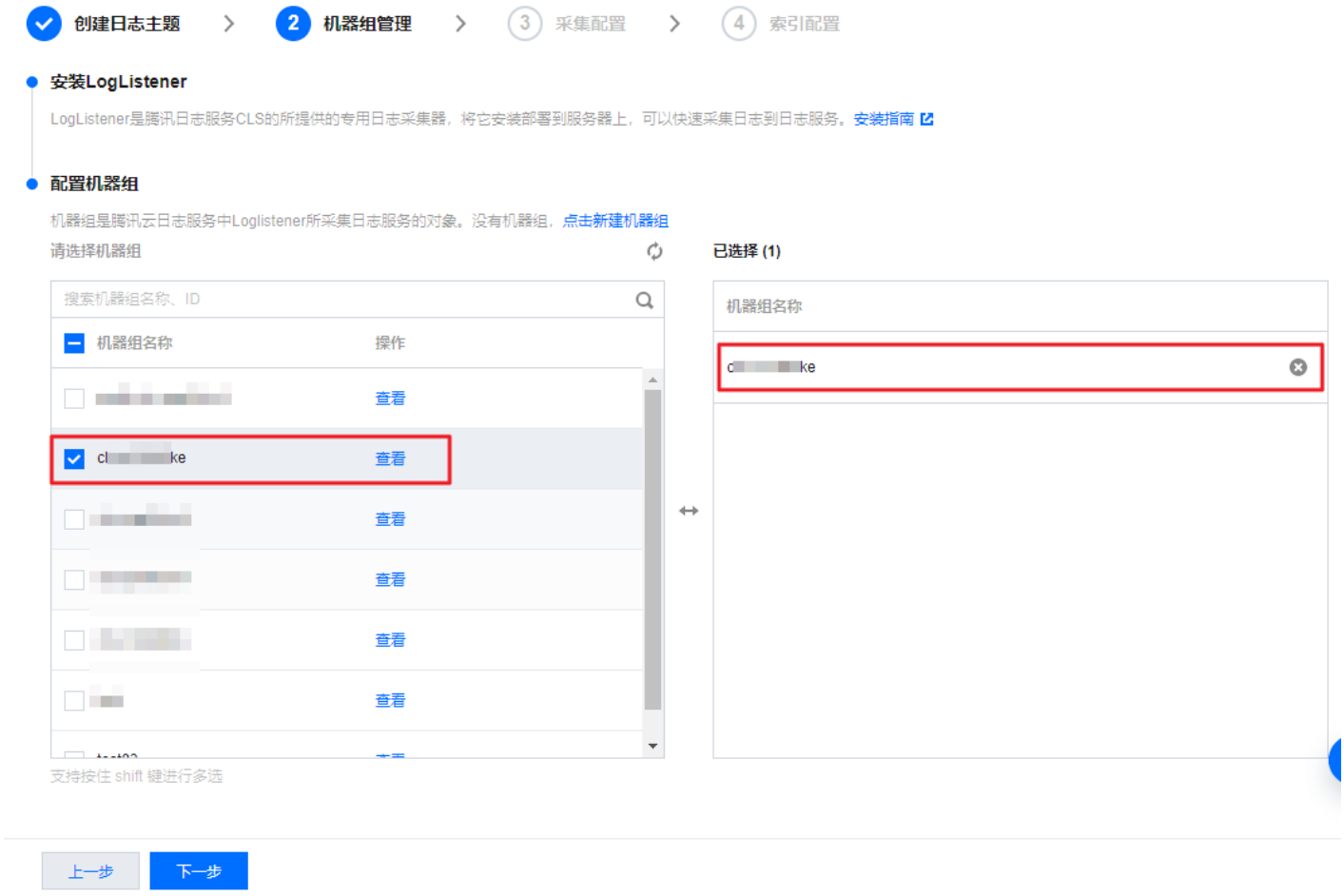
新增日志主题

1. 单击**创建日志主题**。

2. 在弹出的对话框中，将“日志主题名称”填写为“test-multi”，单击**确定**，即可新增日志主题。如下图所示：

机器组管理

1. 日志主题创建成功后，进入该日志主题管理页面。
2. 选择**采集配置**页签，在“LogListener采集配置”单击**新增**，选择需要采集的日志数据源格式。
3. 在“机器组管理”页面，勾选需要与当前日志主题进行绑定的机器组，单击**下一步**。
   即可进入采集配置阶段，更多详情请参阅 管理机器组。



采集配置

配置日志文件采集路径

在“采集配置”页面，填写采集规则名称，并根据日志采集路径格式填写“采集路径”。如下图所示：

日志采集路径格式：[目录前缀表达式]//[文件名表达式]。**

填写日志采集路径后，LogListener 会按照[目录前缀表达式]匹配所有符合规则的公共前缀路径，并监听这些目录（包含子层目录）下所有符合[文件名表达式]规则的日志文件。其参数详细说明如下：

字段	说明
目录前缀	日志文件前缀目录结构，仅支持通配符 * 和 ？ • * 表示匹配多个任意字符 • ？表示匹配单个任意字符
/**/	表示当前目录以及所有子目录

字段	说明
文件名	日志文件名，仅支持通配符 * 和 ？ • * 表示匹配多个任意字符 • ？表示匹配单个任意字符

常用的配置模式如下：

- **[公共目录前缀]/**/[公共文件名前缀]***
- **[公共目录前缀]/*/[公共文件名后缀]**
- **[公共目录前缀]/**/[公共文件名前缀]*[公共文件名后缀]**
- **[公共目录前缀]/**/*[公共字符串]***

填写示例如下：

序号	目录前缀表达式	文件名表达式	说明
1.	/var/log/nginx	access.log	此例中，日志路径配置为/var/log/nginx/**/access.log，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以access.log命名的日志文件
2.	/var/log/nginx	*.log	此例中，日志路径配置为 /var/log/nginx/**/*.log，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以 .log 结尾的日志文件
3.	/var/log/nginx	error*	此例中，日志路径配置为/var/log/nginx/**/error*，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以error开头命名的日志文件

注意：

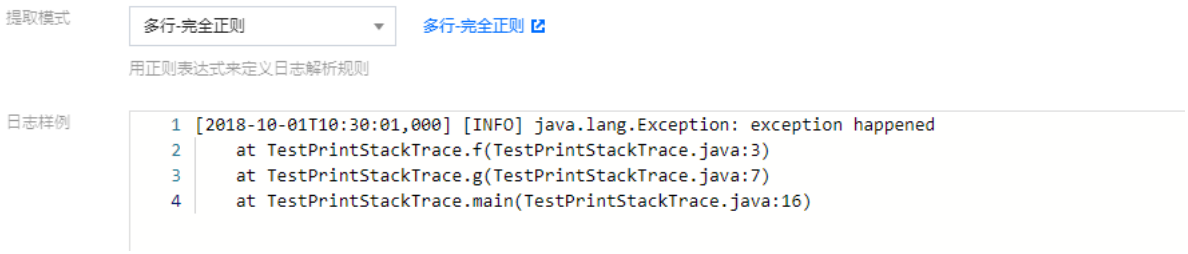
- **Loglistener 2.3.9及以上版本才可以添加多个采集路径。**
- **暂不支持上传的日志内容中含有多种文本格式，可能会导致写入失败，例如 key:"{"substream":XXX}"。**
- **建议配置采集路径为 log/*.log，rename日志轮转后的老文件命名为 log/*.log.xxxx。**
- **默认情况下，一个日志文件只能被一个日志主题采集。如果一个文件需要对应多个采集配置，请给源文件添加一个软链接，并将其加到另一组采集配置中。**

配置采集策略

- **全量采集：Loglistener 采集文件时，从文件的开头开始读。**
- **增量采集：Loglistener 采集文件时，从距离文件末尾1M处开始读取（若文件小于1M，等价全量采集）。**

配置多行-完全正则模式

1. 在“采集配置”页面，将“提取模式”设置为多行-完全正则，并在“日志样例”文本框中，输入日志样例。如下图所示：

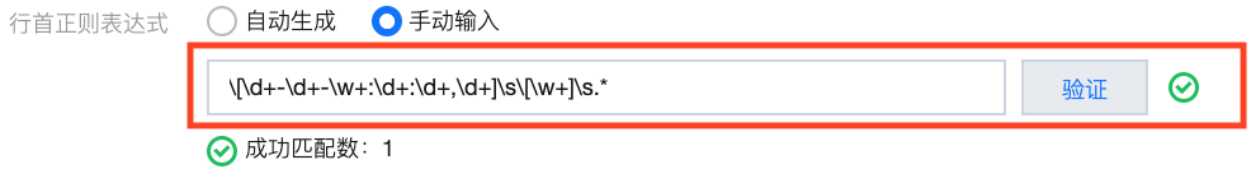


2. 根据如下规则，定义正则表达式。

您可选择自动生成或者手动输入两种方式定义行首正则表达式，确定跨行日志边界。待表达式验证成功后，系统会为您判断行首正则表达式匹配的日志条数。

- **自动生成：单击自动生成，系统自动在置灰的文本框中生成行首正则表达式。如下图所示：**

- **手动输入：在文本框中，手动输入行首正则表达式，单击验证，系统将判断表达式是否通过。如下图所示：**



3. **提取正则表达式**

系统有手动模式和自动模式两种方式定义正则表达式。您可手动输入表达式提取 key-value 进行验证，也可单击正则表达式自动生成切换为自动模式。系统会根据您选择的模式以及定义好的正则表达式，提取 key-value 进行正则表达式的验证。

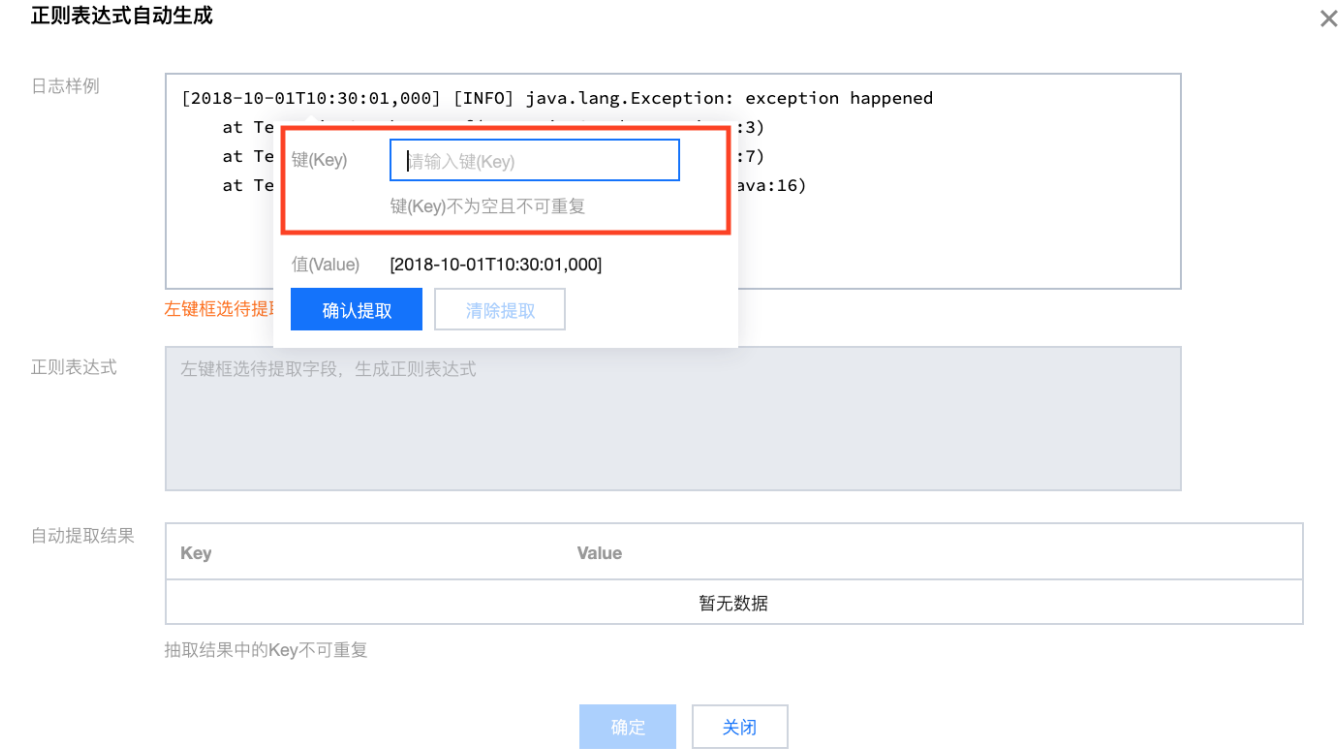
- **手动模式：**

a. **在“正则表达式”的文本框中，输入正则表达式。**

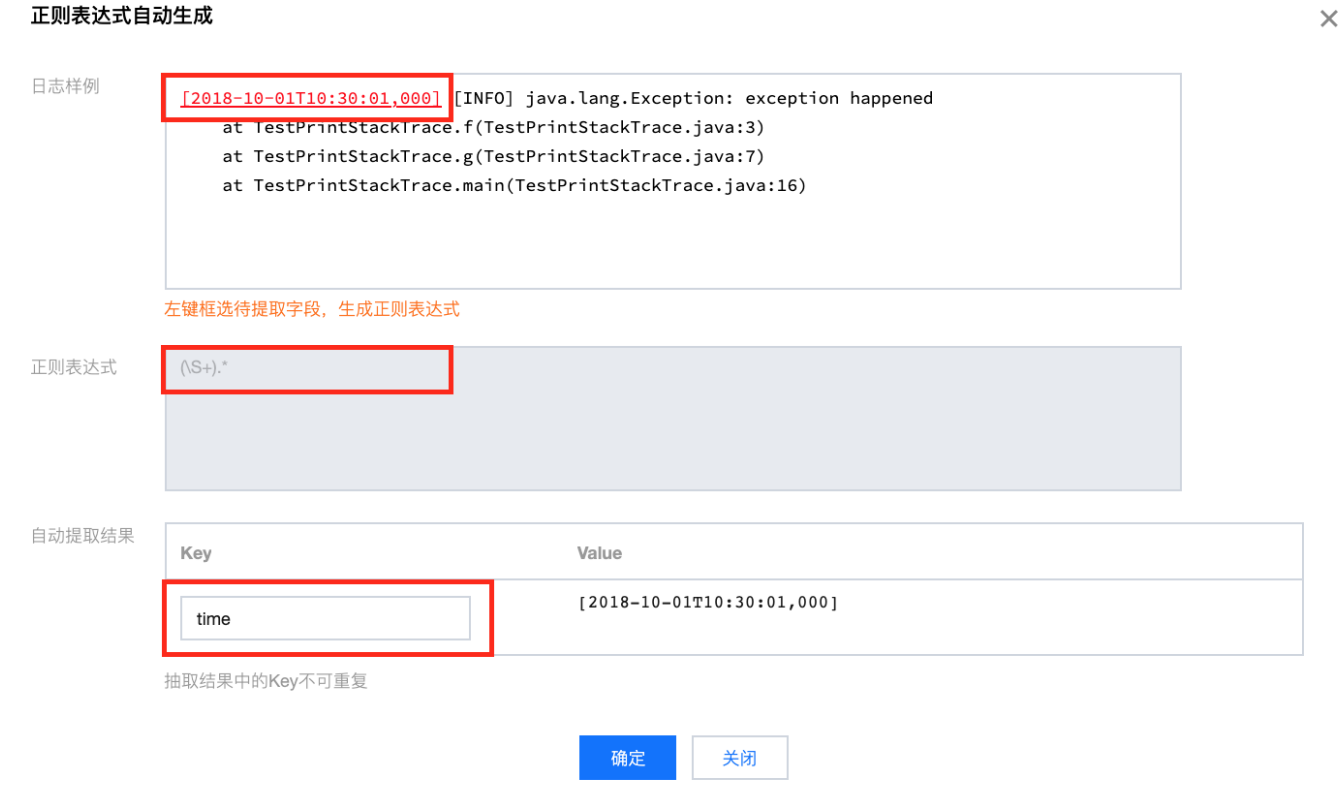
b. **单击验证，系统将判断日志样例与正则表达式是否匹配。**

- **自动模式（单击正则表达式自动生成进行切换）：**

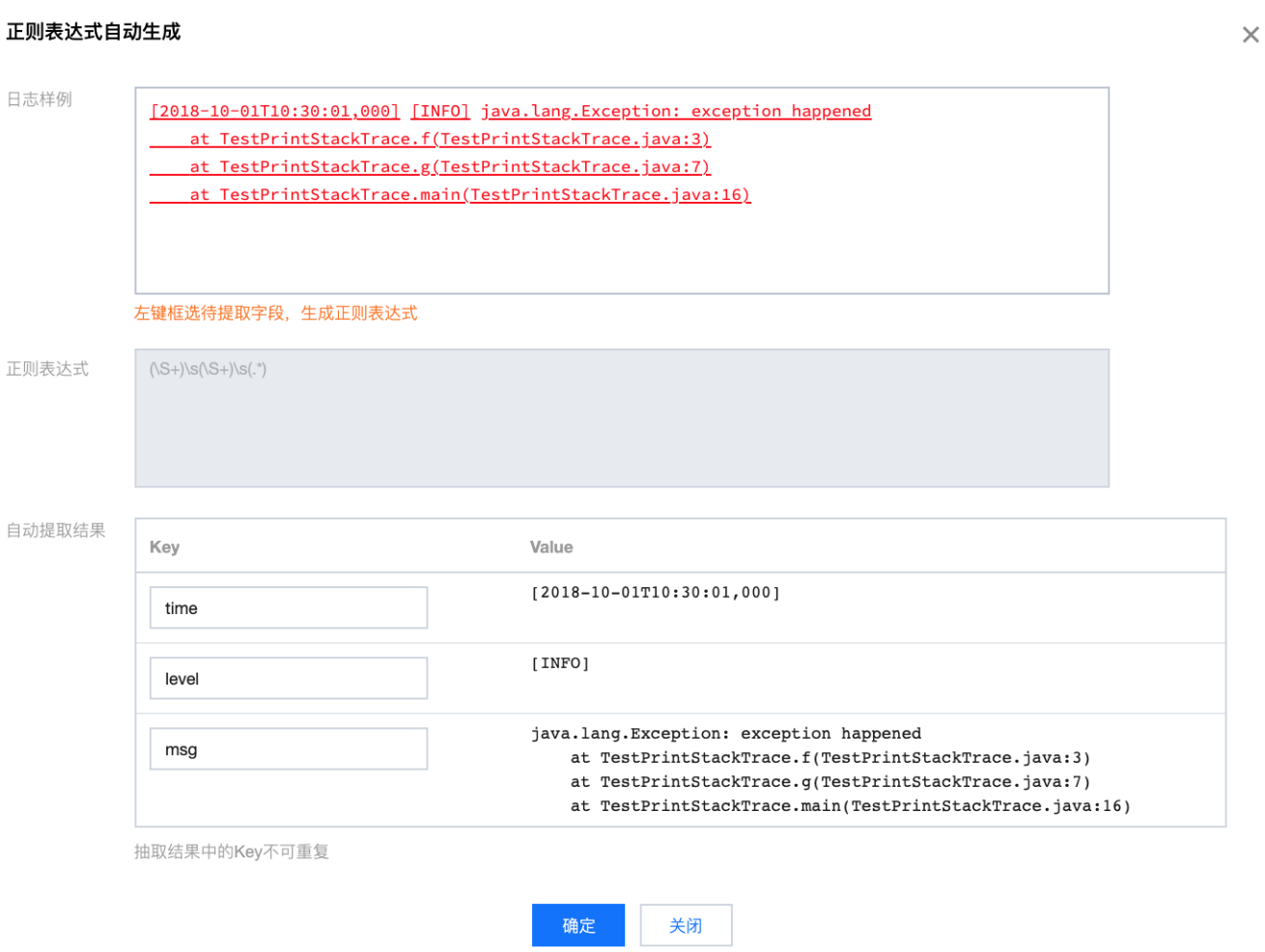
a. **在弹出的“正则表达式自动生成”模态视图中，根据实际的检索分析需求，选中需要提取 key-value 的日志内容，并在弹出的文本框中，输入键(key)名，单击确认提取。如下图所示：**



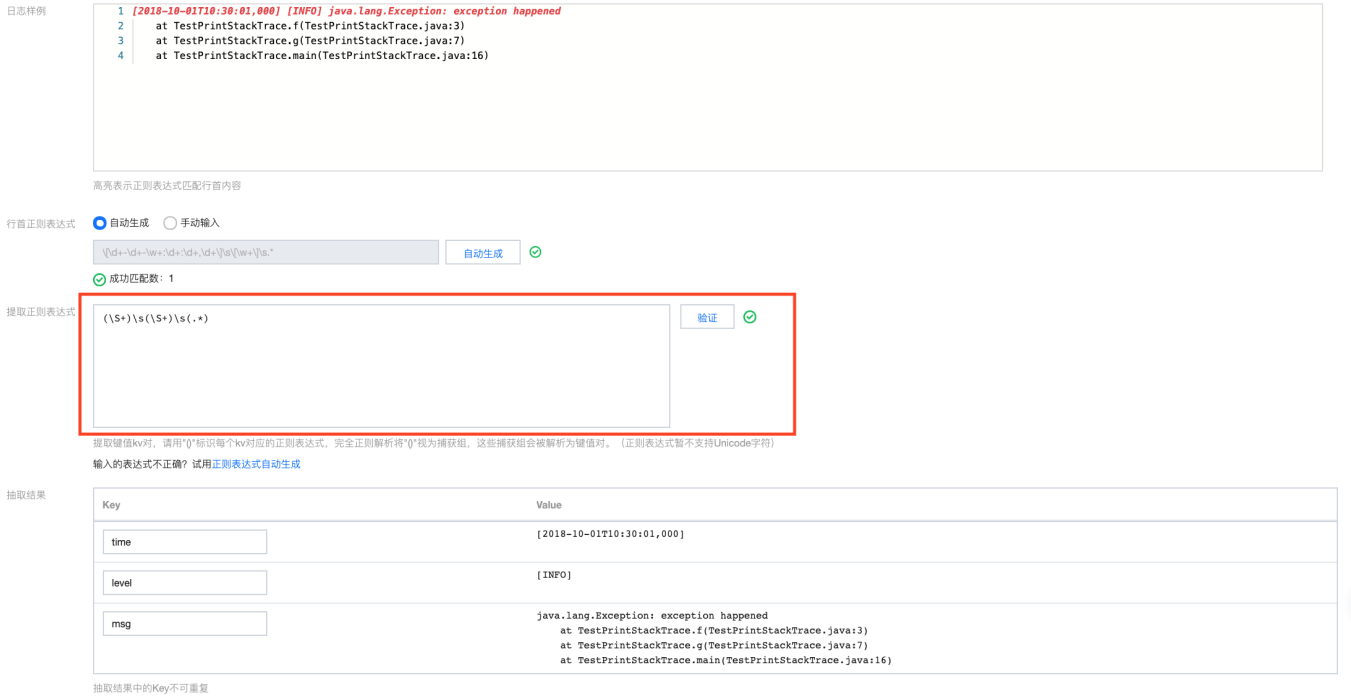
系统将自动对该部分内容提取一个正则表达式，自动提取结果会出现在 key-value 表格中。如下图所示：



b. 重复 步骤 a，直到提取完所有的 key-value 对。如下图所示：



c. 单击**确定**，系统将根据提取好的 key-value 对自动生成完整的正则表达式。如下图所示：

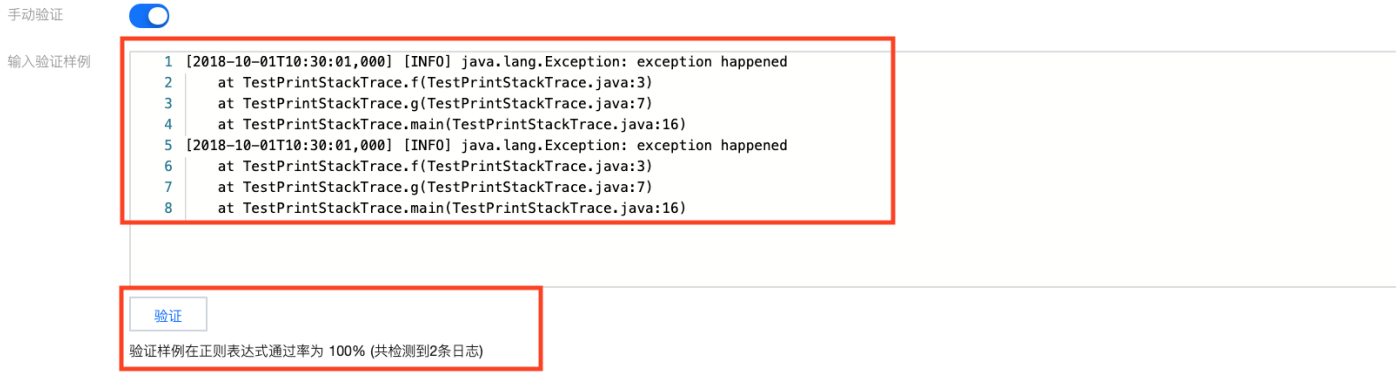


说明：

无论选择自动模式还是手动模式，正则提取模式均在完成定义并验证通过后，将提取结果展示在“抽取结果”中。您只需定义每一组 key-value 对的 key 名称，即可将该名称用于日志检索分析。

手动验证

1. 当您的日志数据复杂时，可以将“手动验证”设置为 ，即可开启手动验证。
2. 输入多个日志样例，单击**验证**。系统将验证样例正则表达式的通过率。



配置采集时间

说明：

- 日志时间单位为：秒，若时间格式填写错误日志时间将以采集时间为准。
- 日志的时间属性有两种方式来定义：采集时间和原始时间戳。
- 采集时间：日志的时间属性由日志服务 CLS 采集该条日志的时间决定。
- 原始时间戳：日志的时间属性由原始日志中时间戳决定。

- **采集时间作为日志的时间属性**：保持采集时间状态为开启状态即可。
- **日志的原始时间戳作为日志时间属性**：关闭采集时间状态，在时间键和时间格式解析处，填写原始时间戳的时间键以及对应的时间解析格式。时间解析格式详情参见 配置时间格式。
  这里举例说明时间格式解析规则填写：
  例1：日志样例原始时间戳：10/Dec/2017:08:00:00，解析格式为：%d/%b/%Y:%H:%M:%S。
  例2：日志样例原始时间戳：2017-12-10 08:00:00，解析格式为：%Y-%m-%d %H:%M:%S。
  例3：日志样例原始时间戳：12/10/2017, 08:00:00，解析格式为：%m/%d/%Y, %H:%M:%S。

配置过滤器条件

过滤器旨在您根据业务需要添加日志采集过滤规则，帮助您筛选出有价值的日志数据。过滤规则为 Perl 正则表达式，所创建的过滤规则为命中规则，即匹配上正则表达式的日志才会被采集上报。

完全正则过来采集时，需要根据所自定义的键值对来配置过滤规则。例如，样例日志使用完全正则模式解析后，您希望 status 字段为400或500的所有日志数据被采集，那么 key 处配置 status，过滤规则处配置 400|500。

注意：
多条过滤规则之间关系是"与"逻辑，若同一 key 名配置多条过滤规则，规则会被覆盖。

配置上传解析失败日志

建议开启上传解析失败日志。开启后，Loglistener 会上传各式解析失败的日志。若关闭上传解析失败日志，则会丢弃失败的日志。

开启后需要配置解析失败的 Key 值（默认为 LogParseFailure），所有解析失败的日志，均以输入内容作为键名称（Key），原始日志内容作为值（Value）进行上传。

索引配置

1. 单击**下一步**，进入“索引配置”页面。
2. 在“索引配置”页面，设置如下信息。

- 索引状态：确认是否开启。

  > **注意：**
  > 检索必须开启索引配置，否则无法检索。

- 全文索引：确认是否需要设置大小写敏感。
- 全文分词符：默认为“@&()='",;:<>[]{}/ \n\t\r”，确认是否需要修改。
- 键值索引：默认关闭，您可根据 key 名按需进行字段类型、分词符以及是否开启统计分析的配置。若您需要开启键值索引，可将  设置为  。

3. 单击**提交**，完成采集配置。

相关操作

检索日志，请参见 检索分析 文档。

JSON 格式

最近更新时间：2022-05-24 18:34:38

操作场景

JSON 格式日志会自动提取首层的 key 作为对应字段名，首层的 value 作为对应的字段值，以该方式将整条日志进行结构化处理，每条完整的日志以换行符 \n 为结束标识符。

前提条件

假设您的一条 JSON 日志原始数据为：

```
{"remote_ip":"10.135.46.111","time_local":"22/Jan/2019:19:19:34 +0800","body_sent":23,"responsetime":0.232,"upstreamtime":"0.232","upstreamhost":"unix:/tmp/php-cgi.sock","http_host":"127.0.0.1","method":"POST","url":"/event/dispatch","request":"POST /event/dispatch HTTP/1.1","xff":"-","referer":"http://127.0.0.1/my/course/4","agent":"Mozilla/5.0 (Windows NT 10.0; WOW64; rv:64.0) Gecko/20100101 Firefox/64.0","response_code":"200"}
```

经过日志服务结构化处理后，该条日志将变为如下：

```
agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:64.0) Gecko/20100101 Firefox/64.0
body_sent: 23
http_host: 127.0.0.1
method: POST
referer: http://127.0.0.1/my/course/4
remote_ip: 10.135.46.111
request: POST /event/dispatch HTTP/1.1
response_code: 200
responsetime: 0.232
time_local: 22/Jan/2019:19:19:34 +0800
upstreamhost: unix:/tmp/php-cgi.sock
upstreamtime: 0.232
url: /event/dispatch
xff: -
```

操作步骤

登录控制台

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**，进入日志主题管理页面。

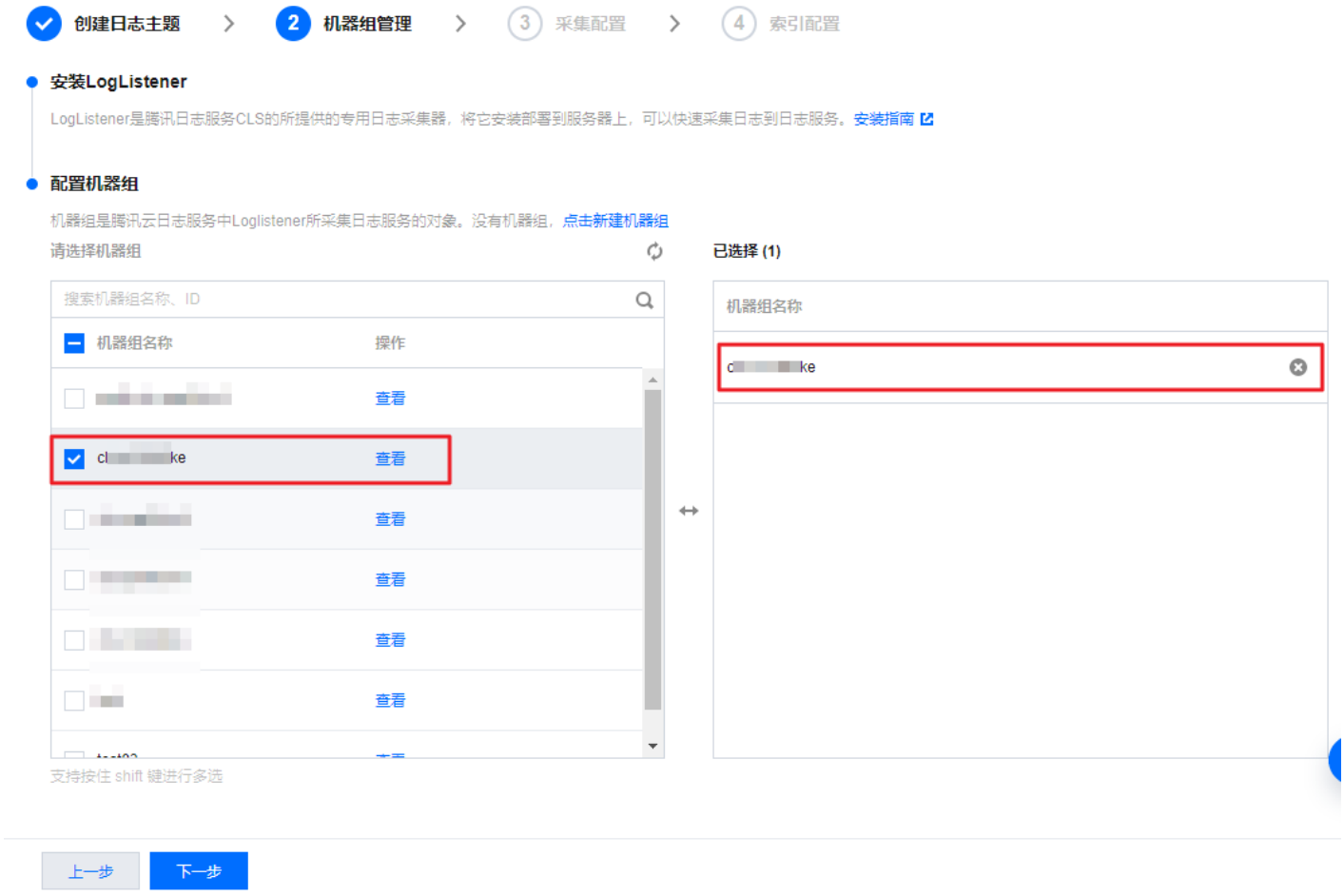
创建日志主题

1. 单击**创建日志主题**。

2. 在弹出的对话框中，将“日志主题名称”填写为“test-json”，单击**确定**，即可新增日志主题。如下图所示：

机器组管理

1. 日志主题创建成功后，进入该日志主题管理页面。
2. 选择**采集配置**页签，在“LogListener采集配置”单击**新增**，选择需要采集的日志数据源格式。
3. 在“机器组管理”页面，勾选需要与当前日志主题进行绑定的机器组，单击**下一步**。
   即可进入采集配置阶段，更多详情请参阅 管理机器组。



采集配置

配置日志文件采集路径

在“采集配置”页面，填写采集规则名称，并根据日志采集路径格式填写“采集路径”。如下图所示：

日志采集路径格式：[目录前缀表达式]//[文件名表达式]。**

填写日志采集路径后，LogListener 会按照[目录前缀表达式]匹配所有符合规则的公共前缀路径，并监听这些目录（包含子层目录）下所有符合[文件名表达式]规则的日志文件。其参数详细说明如下：

字段	说明
目录前缀	日志文件前缀目录结构，仅支持通配符 * 和 ？，* 表示匹配多个任意字符，？表示匹配单个任意字符
/**/	表示当前目录以及所有子目录
文件名	日志文件名，仅支持通配符 * 和 ？，* 表示匹配多个任意字符，？表示匹配单个任意字符

常用的配置模式如下：

- [公共目录前缀]/**/[公共文件名前缀]*
- [公共目录前缀]/**/*[公共文件名后缀]
- [公共目录前缀]/**/[公共文件名前缀]*[公共文件名后缀]
- [公共目录前缀]/**/*[公共字符串]*

填写示例如下：

序号	目录前缀表达式	文件名表达式	说明
1.	/var/log/nginx	access.log	此例中，日志路径配置为/var/log/nginx/**/access.log，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以access.log命名的日志文件
2.	/var/log/nginx	*.log	此例中，日志路径配置为/var/log/nginx/**/*.log，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以.log结尾的日志文件
3.	/var/log/nginx	error*	此例中，日志路径配置为/var/log/nginx/**/error*，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以error开头命名的日志文件

注意：

- Loglistener 2.3.9及以上版本才可以添加多个采集路径。
- 暂不支持上传的日志内容中含有多种文本格式，可能会导致写入失败，例如 key:"{"substream":XXX}"。
- 建议配置采集路径为 log/*.log，rename日志轮转后的老文件命名为 log/*.log.xxxx。
- 默认情况下，一个日志文件只能被一个日志主题采集。如果一个文件需要对应多个采集配置，请给源文件添加一个软链接，并将其加到另一组采集配置中。

配置采集策略

- 全量采集：Loglistener 采集文件时，从文件的开头开始读。
- 增量采集：Loglistener 采集文件时，从距离文件末尾1M处开始读取（若文件小于1M，等价全量采集）。

配置 JSON 模式

在“采集配置”页面，将“提取模式”设置为 JSON。如下图所示：

配置采集时间

说明：

- 日志时间单位为：秒，若时间格式填写错误日志时间将以采集时间为准。
- 日志的时间属性有两种方式来定义：采集时间和原始时间戳。
- 采集时间：日志的时间属性由日志服务 CLS 采集该条日志的时间决定。
- 原始时间戳：日志的时间属性由原始日志中时间戳决定。

- **采集时间作为日志的时间属性**：保持采集时间状态为开启状态即可。
- **日志的原始时间戳作为日志时间属性**：关闭采集时间状态，在时间键和时间格式解析处，填写原始时间戳的时间键以及对应的时间解析格式。时间解析格式详情参见 配置时间格式。
  这里举例说明时间格式解析规则填写：
  例1：日志样例原始时间戳：10/Dec/2017:08:00:00，解析格式为：%d/%b/%Y:%H:%M:%S。
  例2：日志样例原始时间戳：2017-12-10 08:00:00，解析格式为：%Y-%m-%d %H:%M:%S。
  例3：日志样例原始时间戳：12/10/2017, 08:00:00，解析格式为：%m/%d/%Y, %H:%M:%S。

配置过滤器条件

过滤器旨在您根据业务需要添加日志采集过滤规则，帮助您筛选出有价值的日志数据。过滤规则为 Perl 正则表达式，所创建的过滤规则为命中规则，即匹配上正则表达式的日志才会被采集上报。

对于 JSON 格式日志，可以根据所解析成的键值对配置过滤规则。例如，您希望原始 JSON 格式日志内容中 response_code 为400或500的所有日志数据被采集，那么 key 处配置 response_code，过滤规则处配置400|500。

> **注意：**
> 多条过滤规则之间关系是"与"逻辑；若同一 key 名配置多条过滤规则，规则会被覆盖。

配置上传解析失败日志

建议开启上传解析失败日志。开启后，Loglistener 会上传各式解析失败的日志。若关闭上传解析失败日志，则会丢弃失败的日志。

开启后需要配置解析失败的 Key 值（默认为 LogParseFailure），所有解析失败的日志，均以输入内容作为键名称（Key），原始日志内容作为值（Value）进行上传。

索引配置

1. 单击**下一步**，进入“索引配置”页面。
2. 在“索引配置”页面，设置如下信息。

- 索引状态：确认是否开启。

  > **注意：**
  > 检索必须开启索引配置，否则无法检索。

- 全文索引：确认是否需要设置大小写敏感。
- 全文分词符：默认关闭，确认是否需要开启。
- 键值索引：默认开启，您可按需进行字段类型、分词符以及是否开启统计分析的配置。若您需要关闭键值索引，可将 设置为 。

3. 单击**提交**，完成采集配置。

相关操作

检索日志，请参见 检索分析 文档。

分隔符格式

最近更新时间：2022-05-24 18:34:10

操作场景

分隔符日志是指一条日志数据可以根据指定的分隔符将整条日志进行结构化处理，每条完整的日志以换行符 \n 为结束标识符。日志服务在进行分隔符格式日志处理时，您需要为每个分开的字段定义唯一的 key。

前提条件

假设您的一条日志原始数据为：

```
10.20.20.10 - ::: [Tue Jan 22 14:49:45 CST 2019 +0800] ::: GET /online/sample HTTP/1.1 ::: 127.0.0.1 ::: 200 ::: 647 ::: 35 ::: http://127.0.0.1/
```

当日志解析的分隔符指定为 ::: ，该条日志会被分割成八个字段，并为这八个字段定义唯一的 key，如下所示：

```
IP: 10.20.20.10 -
bytes: 35
host: 127.0.0.1
length: 647
referer: http://127.0.0.1/
request: GET /online/sample HTTP/1.1
status: 200
time: [Tue Jan 22 14:49:45 CST 2019 +0800]
```

操作步骤

登录控制台

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**，进入日志主题管理页面。

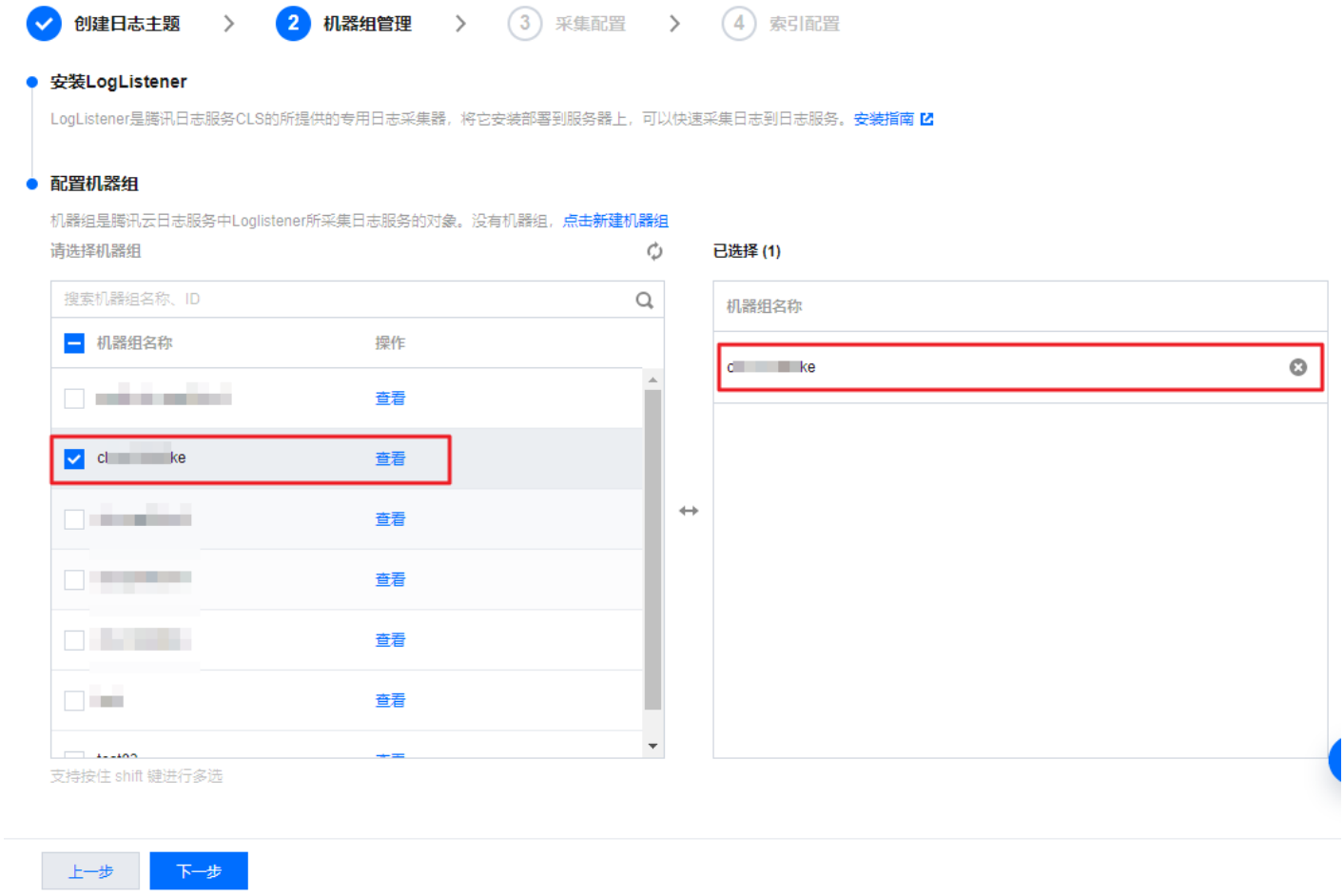
创建日志主题

1. 单击**创建日志主题**。

2. 在弹出的对话框中，将“日志主题名称”填写为“test-separator”，单击**确定**，即可新增日志主题。如下图所示：

机器组管理

1. 日志主题创建成功后，进入该日志主题管理页面。
2. 选择**采集配置**页签，在“LogListener采集配置”单击**新增**，选择需要采集的日志数据源格式。
3. 在“机器组管理”页面，勾选需要与当前日志主题进行绑定的机器组，单击**下一步**。
   即可进入采集配置阶段，更多详情请参阅 管理机器组。



采集配置

在“采集配置”页面，填写采集规则名称，并根据日志采集路径格式填写“采集路径”。如下图所示：

日志采集路径格式：[目录前缀表达式]//[文件名表达式]。**

填写日志采集路径后，LogListener 会按照[目录前缀表达式]匹配所有符合规则的公共前缀路径，并监听这些目录（包含子层目录）下所有符合[文件名表达式]规则的日志文件。其参数详细说明如下：

字段	说明
目录前缀	日志文件前缀目录结构，仅支持通配符 * 和 ？ • * 表示匹配多个任意字符 • ？表示匹配单个任意字符
/**/	表示当前目录以及所有子目录

字段	说明
文件名	日志文件名，仅支持通配符 * 和 ？， • * 表示匹配多个任意字符 • ？表示匹配单个任意字符

常用的配置模式如下：

- [公共目录前缀]/**/[公共文件名前缀]*
- [公共目录前缀]/**/*[公共文件名后缀]
- [公共目录前缀]/**/[公共文件名前缀]*[公共文件名后缀]
- [公共目录前缀]/**/*[公共字符串]*

填写示例如下：

序号	目录前缀表达式	文件名表达式	说明
1.	/var/log/nginx	access.log	此例中，日志路径配置为 /var/log/nginx/**/access.log，LogListener 将会监听 /var/log/nginx 前缀路径下所有子目录中以 access.log 命名的日志文件
2.	/var/log/nginx	*.log	此例中，日志路径配置为 /var/log/nginx/**/*.log，LogListener 将会监听 /var/log/nginx 前缀路径下所有子目录中以 .log 结尾的日志文件
3.	/var/log/nginx	error*	此例中，日志路径配置为 /var/log/nginx/**/error*，LogListener 将会监听 /var/log/nginx 前缀路径下所有子目录中以 error 开头命名的日志文件

⚠ 注意：

- Loglistener 2.3.9及以上版本才可以添加多个采集路径。
- 暂不支持上传的日志内容中含有多种文本格式，可能会导致写入失败，例如 key:"{"substream":XXX}"。
- 建议配置采集路径为 log/*.log，rename日志轮转后的老文件命名为 log/*.log.xxxx。
- 默认情况下，一个日志文件只能被一个日志主题采集。如果一个文件需要对应多个采集配置，请给源文件添加一个软链接，并将其加到另一组采集配置中。

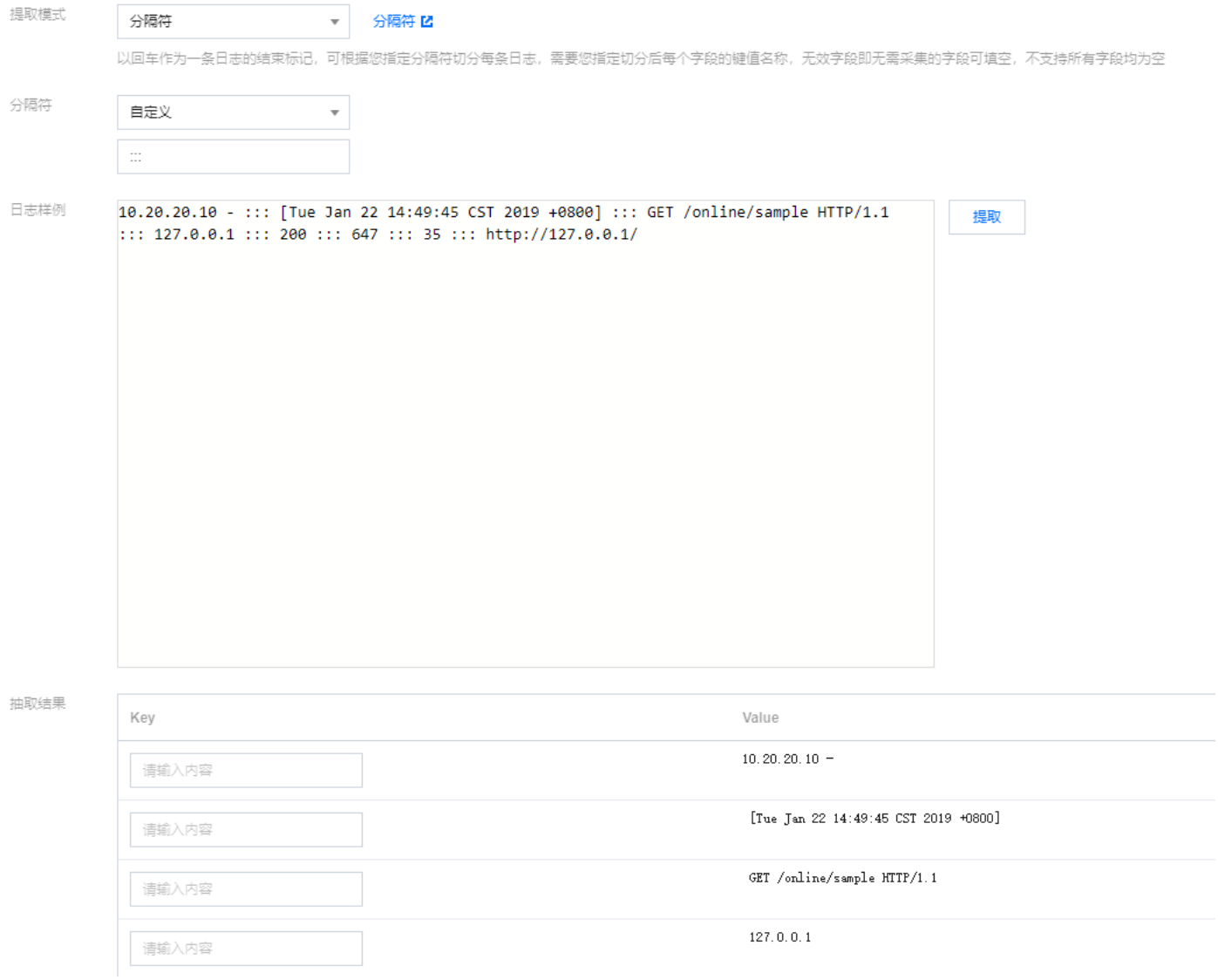
配置采集策略

- 全量采集：Loglistener 采集文件时，从文件的开头开始读。
- 增量采集：Loglistener 采集文件时，从距离文件末尾1M处开始读取（若文件小于1M，等价全量采集）。

配置分隔符模式

1. 将“提取模式”设置为**分隔符**。
2. 选择**分隔符**，并在“日志样例”文本框中，输入日志样例，单击**提取**。
   系统根据确定的分隔符将日志样例进行切分，并展示在抽取结果栏中，您需要为每个字段定义唯一的 key。目前，日志采集支持多种分隔符，常见的分隔符有：空格、制

表符、逗号、分号、竖线，若您的日志数据所采用的分隔符是其他符号，例如 ::: ，也可以通过自定义分词符进行解析。



配置采集时间

> **说明：**
>
> - 日志时间单位为：秒，若时间格式填写错误日志时间将以采集时间为准。
> - 日志的时间属性有两种方式来定义：采集时间和原始时间戳。
> - 采集时间：日志的时间属性由日志服务 CLS 采集该条日志的时间决定。
> - 原始时间戳：日志的时间属性由原始日志中时间戳决定。

- **采集时间作为日志的时间属性**

  保持采集时间状态为开启状态即可，如下图所示：

- **日志的原始时间戳作为日志时间属性**

  关闭采集时间状态，在时间键和时间格式解析处，填写原始时间戳的时间键以及对应的时间解析格式。时间解析格式详情参见 配置时间格式。

这里举例说明时间格式解析规则填写：

例1：日志样例原始时间戳：10/Dec/2017:08:00:00，解析格式为：%d/%b/%Y:%H:%M:%S。

例2：日志样例原始时间戳：2017-12-10 08:00:00，解析格式为：%Y-%m-%d %H:%M:%S。

例3：日志样例原始时间戳：12/10/2017, 08:00:00，解析格式为：%m/%d/%Y, %H:%M:%S。

> **注意：**
> **日志时间支持以秒为单位，若时间格式填写错误日志时间将以采集时间为准。**

配置过滤器条件

过滤器旨在您根据业务需要添加日志采集过滤规则，帮助您筛选出有价值的日志数据。过滤规则为 Perl 正则表达式，所创建的过滤规则为命中规则，即匹配上正则表达式的日志才会被采集上报。

分隔符格式日志需要根据所自定义的键值对来配置过滤规则。例如，样例日志使用分隔符模式解析后，您希望 status 字段为 400 或 500 的所有日志数据被采集，那么 key 处配置 status，过滤规则处配置 400|500。

> **注意：**
> **多条过滤规则之间关系是"与"逻辑；若同一 key 名配置多条过滤规则，规则会被覆盖。**

配置上传解析失败日志

建议开启上传解析失败日志。开启后，Loglistener 会上传各式解析失败的日志。若关闭上传解析失败日志，则会丢弃失败的日志。

开启后需要配置解析失败的 Key 值（默认为 LogParseFailure），所有解析失败的日志，均以输入内容作为键名称（Key），原始日志内容作为值（Value）进行上传。

索引配置

1. 单击下一步，进入“索引配置”页面。

2. 在“索引配置”页面，设置如下信息。

- 索引状态：确认是否开启。

> ⚠ 注意：
> 检索必须开启索引配置，否则无法检索。

- 全文索引：确认是否需要设置大小写敏感。
- 全文分词符：默认为“@&()='",;:<>[]{}/ \n\t\r”，确认是否需要修改。
- 键值索引：默认关闭，您可根据 key 名按需进行字段类型、分词符以及是否开启统计分析的配置。若您需要开启键值索引，可将  设置为  。

3. 单击**提交**，完成采集配置。

相关操作

检索日志，请参见 检索分析 文档。

组合解析格式

最近更新时间：2022-01-12 14:41:24

操作场景

当您的日志结构太过复杂涉及多种解析模式，单种解析模式（如 Nginx 模式、完整正则模式、JSON 模式等）无法满足日志解析需求时，您可以使用 Loglistener 组合解析格式解析日志，此模式支持用户在控制台输入代码（json 格式）用来定义日志解析的流水线逻辑。您可添加一个或多个 Loglistener 插件处理配置，Loglistener 会根据处理配置顺序逐一执行。

前提条件

假设您的一条日志的原始数据为：

```
1571394459，http://127.0.0.1/my/course/4|10.135.46.111|200，status:DEAD,
```

自定义插件内容如下：

```
{
"processors": [
{
"type": "processor_split_delimiter",
"detail": {
"Delimiter": ",",
"ExtractKeys": [ "time", "msg1","msg2"]
},
"processors": [
{
"type": "processor_timeformat",
"detail": {
"KeepSource": true,
"TimeFormat": "%s",
"SourceKey": "time"
}
},
{
"type": "processor_split_delimiter",
"detail": {
"KeepSource": false,
"Delimiter": "|",
"SourceKey": "msg1",
"ExtractKeys": [ "submsg1","submsg2","submsg3"]
},
"processors": []
},
{
"type": "processor_split_key_value",
"detail": {
"KeepSource": false,
"Delimiter": ": ",
"SourceKey": "msg2"
}
}
]
}
]
}
```

经过日志服务结构化处理后，该条日志将变为如下：

```
time: 1571394459
submsg1: http://127.0.0.1/my/course/4
submsg2: 10.135.46.111
submsg3: 200
status: DEAD
```

配置说明

自定义插件种类

插件功能	插件名称	功能
提取字段	processor_log_string	使用 processor_log_string 插件对字段进行多字符解析（换行符），一般用于单行日志的高级功能
提取字段	processor_multiline	使用 processor_multiline 插件（正则模式）对字段进行首行正则解析，一般用于多行日志的高级功能
提取字段	processor_multiline_fullregex	使用 processor_multiline_fullregex 插件（正则模式）对字段进行首行正则解析，一般用于多行日志的高级功能；并对多行日志进行正则提取
提取字段	processor_fullregex	使用 processor_fullregex 插件（正则模式）提取字段（单行日志）
提取字段	processor_json	使用 processor_json 插件对字段值进行 JSON 展开
提取字段	processor_split_delimiter	使用 processor_split_delimiter 插件（单字符/多字符分隔符模式）提取字段
提取字段	processor_split_key_value	使用 processor_split_key_value 插件（键值对模式）提取字段
处理字段	processor_drop	使用 processor_drop 插件丢弃字段
处理字段	processor_timeformat	使用 processor_timeformat 插件，解析原始日志中的时间字段，用于转换时间格式，并可将解析结果设置为日志时间

自定义插件详细参数

插件名称	是否支持子项解析	插件参数	是否必须	功能
processor_multiline	否	BeginRegex	是	定义多行日志的行首匹配正则
processor_multiline_fullregex	是	BeginRegex	是	定义多行日志的行首匹配正则
		ExtractRegex	是	定义提取到多行日志后的提取正则
		ExtractKeys	是	定义提取键值
processor_fullregex	是	ExtractRegex	是	定义提取正则
		ExtractKeys	是	定义提取键值
processor_json	是	SourceKey	否	当前 processor 处理的上一级 processor 中的 key name
		KeepSource	否	最终键值名称中，是否保留 SourceKey
processor_split_delimiter	是	SourceKey	否	当前 processor 处理的上一级 processor 中的 key name
		KeepSource	否	最终键值名称中，是否保留 SourceKey

		Delimiter	是	指定分隔符（单/多字符）
		ExtractKeys	是	定义分隔符分割之后的提取键值
processor_split_key_value	否	SourceKey	否	当前 processor 处理的上一级 processor 中的 key name
		KeepSource	否	最终键值名称中，是否保留 SourceKey
		Delimiter	是	定义字符串中 Key 与 Value 之间的分隔符
processor_drop	否	SourceKey	是	当前 processor 处理的上一级 processor 中的 key name
processor_timeformat	否	SourceKey	是	当前 processor 处理的上一级 processor 中的 key name
		TimeFormat	是	定义对 SourceKey 的值（日志中的时间数据字符串）的时间解析格式

操作步骤

登录控制台

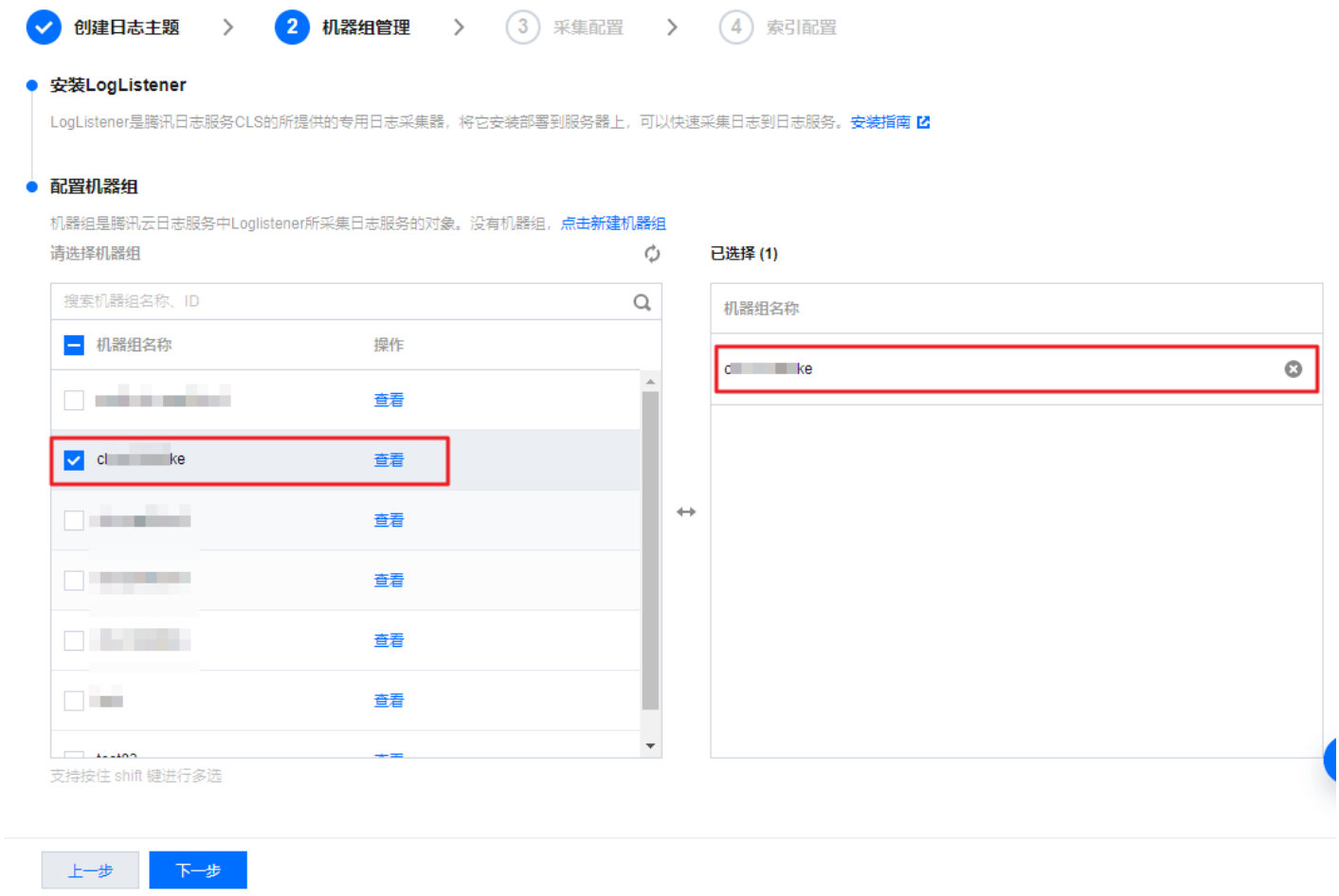
1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**，进入日志主题管理页面。

创建日志主题

1. 单击**创建日志主题**。
2. 在弹出的对话框中，将“日志主题名称”填写为“define-log”，单击**确定**，即可新增日志主题。如下图所示：

机器组管理

1. 日志主题创建成功后，进入该日志主题管理页面。
2. 选择**采集配置**页签，在 **LogListener采集配置**中单击**新增**，选择您需要采集的日志数据源格式。
3. 在机器组管理页面，勾选需要与当前日志主题进行绑定的机器组，单击**下一步**。如下图所示：
   即可进入采集配置阶段，更多详情请参阅 管理机器组。



采集配置

配置日志文件采集路径

在“采集配置”页面，根据日志采集路径格式，填写“采集路径”。如下图所示：

日志采集路径格式：[目录前缀表达式]//[文件名表达式]。**

填写日志采集路径后，LogListener 会按照[目录前缀表达式]匹配所有符合规则的公共前缀路径，并监听这些目录（包含子层目录）下所有符合[文件名表达式]规则的日志文件。其参数详细说明如下：

字段	说明
目录前缀	日志文件前缀目录结构，仅支持通配符 * 和 ？，* 表示匹配多个任意字符，？表示匹配单个任意字符
/**/	表示当前目录以及所有子目录
文件名	日志文件名，仅支持通配符 * 和 ？，* 表示匹配多个任意字符，？表示匹配单个任意字符

常用的配置模式如下：

- **[公共目录前缀]/**/[公共文件名前缀]***

- [公共目录前缀]/**/*[公共文件名后缀]
- [公共目录前缀]/**/[公共文件名前缀]*[公共文件名后缀]
- [公共目录前缀]/**/*[公共字符串]*

填写示例如下：

序号	目录前缀表达式	文件名表达式	说明
1.	/var/log/nginx	access.log	此例中，日志路径配置为/var/log/nginx/**/access.log，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以access.log命名的日志文件
2.	/var/log/nginx	*.log	此例中，日志路径配置为/var/log/nginx/**/*.log，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以.log结尾的日志文件
3.	/var/log/nginx	error*	此例中，日志路径配置为/var/log/nginx/**/error*，LogListener 将会监听/var/log/nginx前缀路径下所有子目录中以error开头命名的日志文件

⚠ 注意：

- Loglistener 2.3.9及以上版本才可以添加多个采集路径。
- 暂不支持上传的日志内容中含有多种文本格式，可能会导致写入失败，例如 key:"{"substream":XXX}"。
- 建议配置采集路径为 log/*.log，rename日志轮转后的老文件命名为 log/*.log.xxxx。
- 默认情况下，一个日志文件只能被一个日志主题采集。如果一个文件需要对应多个采集配置，请给源文件添加一个软链接，并将其加到另一组采集配置中。

配置组合解析模式

在“采集配置”页面，将“提取模式”设置为**组合解析**。如下图所示：

配置采集策略

- 全量采集：Loglistener 采集文件时，从文件的开头开始读。
- 增量采集：Loglistener 采集文件时，从距离文件末尾1M处开始读取（若文件小于1M，等价全量采集）。

使用限制

- 使用组合解析模式解析数据时，loglistener 会需要消耗更多的资源，不建议您使用过于复杂的插件组合来处理数据。
- 使用组合解析模式后，文本模式使用采集功能、过滤器功能将失效，但其中部分功能可通过相关自定义插件实现。
- 使用组合解析模式后，上传解析失败日志功能默认开启，解析失败的日志均以输入名称为键（Key），原始日志内容作为值（Value）进行上传。

相关操作

检索日志

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**检索分析**，进入检索分析页面。
3. 根据实际需求，选择地域、日志集与日志主题，单击**检索分析**，即可开始按照设定的查询条件检索日志。

配置时间格式

最近更新时间：2022-06-21 15:39:13

日志服务要求每条日志必须具有时间属性，以便系统按时间维度对数据进行管理。当使用 Loglistener 采集日志时，时间属性有两种配置方式：

- 默认方式：使用 Loglistener 采集时间作为时间属性。
- 自定义方式：将日志内容中的某个时间字段作为时间属性，需要用户配置时间解析格式。

> 说明：
> Loglistener 采集时间精度为毫秒，所以时间解析格式支持配置到毫秒，按格式配置后不到毫秒级的会自动使用0填充。

解析格式说明

参数格式	说明	示例
%a	星期英文单词名称的简写	Fri
%A	星期英文单词名称的全称	Friday
%b	月份英文单词名称的简写	Jan
%B	月份英文单词名称的全称	January
%d	一个月中的第几天（01 – 31）	31
%h	月份英文单词名称的简写，与%b相同	Jan
%H	小时，24小时制（00 – 23）	22
%I	小时，12小时制（01 – 12）	11
%m	月份（01 – 12），01表示一月份	08
%M	分钟（00 – 59），01表示第一分钟	59
%n	换行符	换行符
%p	上午（AM）或下午（PM）	AM/PM
%r	一种特定的12小时制时间组合格式，等价于%I:%M:%S %p	11:59:59 AM
%R	一种特定的24小时时间组合格式，等价于%H:%M	23:59
%S	秒数（00 – 59）	59
%f	毫秒时间	0.123
%t	tab 制表符	tab 制表符
%y	不包含世纪的年份数字（00 – 99）	19
%Y	包含世纪的年份数字，2018表示2018年	2019
%C	世纪数字（年份除100，范围00 – 99）	20
%e	一个月中的第几天（01 – 31）	31
%j	一年中的第几天（001 – 366）	365
%u	星期数字的表示方式（1 – 7），1表示星期一，7表示星期天	1
%U	一年中的第几周（00 – 53），星期天是一周的开始，即从第一个星期日开始，作为第一周的第一天	23
%w	星期数字的表示方式（0 – 6），0代表星期天，6表示星期六	5
%W	一年中的第几周（00 – 53），星期一是一周的开始，即从第一个星期一开始，作为第一周的第一天	23
%s	秒级（10位）Unix 时间戳	1571394459

参数格式	说明	示例
%F	毫秒级（13位）Unix 时间戳	1571394459123
%z	支持时间字段的时区解析，包括 ISO 8601 时间格式和 GMT 时间格式	UTC/+0800/MST

配置示例

时间表示示例	时间提取格式
2018-07-16 13:12:57.123	%Y-%m-%d %H:%M:%S.%f
[2018-07-16 13:12:57.012]	[%Y-%m-%d %H:%M:%S.%f]
06/Aug/2019 12:12:19 +0800	%d/%b/%Y %H:%M:%S
Monday, 02-Oct-19 16:07:05 MST	%A, %d-%b-%y %H:%M:%S
1571394459	%s
1571394459123	%F
06/Aug/2019 12:12:19 +0800	%d/%b/%Y %H:%M:%S %z
Monday, 02-Oct-19 16:07:05 MST	%A, %d-%b-%y %H:%M:%S %z

LogListener 采集配置导入

最近更新时间：2022-06-24 11:54:03

本文介绍通过 LogListener 采集服务器快速导入其他日志主题的采集配置规则。

操作场景

LogListener 采集配置是指 LogListener 采集服务器在进行日志采集前所设置的采集路径、使用限制、采集模式等采集规则。采集配置规则导入功能支持用户新增或修改采集配置时导入已有日志主题的采集配置，快速配置 LogListener 采集规则，免去多日志主题配置重复的繁琐操作，提高采集配置效率。

> 说明：
>
> - 默认情况下，一个文件只能被一个 LogListener 配置采集。
> - 如果一个文件需对应多个采集配置，请给源文件添加一个软链接，并将其加到另一组采集配置中。
> - LogListener 2.3.9及以上版本才可以添加多个采集路径。

操作步骤

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**，进入日志主题页面。
3. 选择已有的日志主题，单击该日志主题ID/名称，进入日志主题信息页面。
4. 选择**采集配置**页签，进入**采集配置**页面。
5. 单击左上角**导入配置规则**。

6. 在弹出浮窗显示配置规则列表窗口，选择目标日志主题的配置规则，单击**确定**即可导入当前日志主题的采集配置中。

说明：

- 列表默认展示当前地域所有日志主题支持跨地域日志主题配置规则导入。
- 未配置采集路径的日志主题不可导入当前日志主题采集配置规则中。

采集模式

LogListener 支持通过 单行全文格式、多行全文格式、单行完全正则格式、多行完全正则格式、JSON格式、分隔符格式 采集文本日志。

LogListener 版本变更

最近更新时间：2022-06-08 11:13:20

本文档为您介绍日志服务 LogListener 的版本更新记录。

说明：

- 采集 __HOSTNAME__ 功能，在 Loglistener 2.7.4版本中开始支持。
- 组合解析功能，在 LogListener 2.6.4版本中开始支持。
- 全量/增量采集策略功能，在 LogListener 2.6.2版本中开始支持。
- CVM 批量部署功能，在 LogListener 2.6.0版本中开始支持。
- 多行-完全正则采集模式，在 LogListener 2.4.5版本中开始支持。
- LogListener 自动升级功能，在 LogListener 2.5.0 版本中开始支持。
- 解析失败日志上传功能，在 LogListener 2.5.2版本中开始支持。
- 为了更好的使用体验，建议 前往安装/升级至最新版本。

版本号	变更类型	描述
v2.7.9	体验优化	• 增加 loglistener 文件锁校验，默认只能启动一个 agent 实例。 • 优化 containerd stdout 空行处理异常。 • 优化文件句柄泄露导致的磁盘满、业务异常问题。 • 优化多行日志行数过多时，后半部分内容被解析失败的问题。
v2.7.8	体验优化	优化容器场景下 metadata 文件生成延迟，造成日志无 TAG 元数据的问题。
v2.7.7	体验优化	优化 DNS 解析异常恢复后，采集程序网络连接无法恢复的问题。
v2.7.6	体验优化	优化 hostname 提取时的换行符处理。
v2.7.5	体验优化	优化真实文件及其同目录软链接同时采集（不同采集配置）时，文件轮转情况下的处理异常。
v2.7.4	新功能	• 支持采集 hostname 作为元数据。 • 组合解析增加 meta_processor，支持自定义元数据解析（路径）。
	体验优化	• 优化在文件删除场景的漏采问题。 • 由于文件尾无换行符引起的文件大小判断出错，进而引起文件重采。
v2.7.3	新功能	单 agent 实例支持同时多 endpoint 上传日志。
v2.7.2	体验优化	优化轮转文件在移除时无法清理掉对应的配置缓存，造成内存泄漏。
v2.7.1	体验优化	优化大量打印 processor 为空日志的问题。
v2.7.0	体验优化	优化空字符串在封装 PB 时，有可能引发异常，导致采集阻塞的问题。
v2.6.9	体验优化	优化多行解析失败场景下，无效日志超量打印的问题。
v2.6.8	体验优化	• 优化增加 loglistenr 采集规格限制，超限启动保护机制。 • 修复 Ubuntu 开机启动不生效问题。 • 优化黑名单功能，节省内存使用。 • 优化组合解析模式，且 root processor 为正则解析插件时的处理异常。 • 优化部分日志打印。
v2.6.7	新功能	支持单 agent 下多租户采集能力。
v2.6.6	体验优化	修复软链接场景下，对于写入量很小的文件，可能发生漏采/延迟采集的问题。
v2.6.5	新功能	日志时间支持时区信息解析。
	体验优化	• 修复高级数据处理空指针异常。 • 优化当多个文件同时轮转时异常问题。
v2.6.4	新功能	支持用户通过插件自定义日志解析规则。
	体验优化	• 优化日志解析格式 pipeline。 • 修复对毫秒时间戳（%F）格式解析的问题。
v2.6.3	体验优化	• 优化 checkpoint 文件损坏时，loglistener 无法启动的问题。

		• 特殊场景下，黑名单对新文件不生效的问题。
v2.6.2	新功能	支持增量采集功能。
	体验优化	• 优化文件在从扫描到处理之间被移除场景下的采集忽略问题。 • 优化自动升级异常覆盖。
v2.6.1	体验优化	• 优化某些场景下，日志轮转时可能发生回溯采集的问题。 • 调整采集端上传日志超时时间，避免因为 timeout 导致数据重复。
v2.6.0	新功能	• 支持腾讯云 CVM 批量部署功能。 • 支持 secret ID/KEY 密文存储。
	体验优化	• 优化 loglistener install/stop 逻辑。 • 优化 upload 失败场景下的重试策略。 • 增加对老版本 glibc 库造成的 dead lock 的检测修复工具。 • 采集性能优化。
v2.5.9	体验优化	优化资源限制策略。
v2.5.8	体验优化	• 当移除一个目录软链接时，影响到其它指向相同目标的目录软链接的采集的问题。 • 当移除一个目录软链接并再次创建相同软链接后，目录下文件无法采集的问题。
v2.5.7	体验优化	• 当 filesize 大于2G时，会存在重复采集的问题（新引入）。 • 当文件数量特别多的时候，文件发生 rename 有可能会导致程序卡住。 • 文件采集监控中，某字段无法更新。
v2.5.6	体验优化	优化特殊使用场景下，触发采集程序异常，停止工作的问题。
v2.5.5	体验优化	• 文件采集元数据 checkpoint 优化，保证重启不丢数据。 • 支持资源限制可配置及超限处理，内存、CPU、带宽。
v2.5.4	新功能	支持文件采集监控功能。
	体验优化	增强内存资源限制处理，当内存超限持续一段时间后，LogListener 自动加载。
v2.5.3	性能优化	优化内存问题引发 LogListener 工作异常。
v2.5.2	新功能	支持解析失败日志上传需求。
	体验优化	优化黑名单功能，黑名单 FILE 模式支持通配符过滤。
v2.5.1	体验优化	优化当采集文件找不到断点元数据时的处理。
v2.5.0	新功能	• 支持 LogListener 自动升级。 • 支持在 Ubuntu 系统下，LogListener 自启动。
v2.4.6	体验优化	• 变更采集配置时，清理相关配置 cache 的数据残留。 • 修复处理软链接的 IN_DELETE 事件时，影响其他指向此 realpath 文件的软链接文件采集的问题。 • 优化同一源文件同时使用文件软链接和目录软连接进行采集功能。
v2.4.5	新功能	新增 multiline_fullregex_log 日志采集类型支持。
v2.4.4	体验优化	优化 msec 功能导致的日志采集使用日志时间不准确的问题。
v2.4.3	新功能	支持自动检测日志格式（logFormat）。
v2.4.2	体验优化	优化腾讯云容器场景下拉取配置时缓存淘汰问题。
v2.4.1	新功能	支持毫秒采集日志数据。
	体验优化	优化用户日志中无换行符数据引发的工作异常。
v2.4.0	新功能	LogListener 支持进程实例级别监控。
v2.3.9	新功能	支持采集路径配置黑名单。
	性能优化	优化 boost 版本库过低导致的内存泄漏。
v2.3.8	新功能	采集配置支持多路径。
v2.3.6	性能优化	• 修复无效键值 key invalid 导致的停止采集问题。 • 修复请求失败返回502导致的内存泄漏问题。

v2.3.5	新功能	支持日志上下文检索功能。
	性能优化	• 修复在静态配置模式下，在上传日志时返回鉴权失败时后续不再采集的问题。 • 修复在动态配置模式下，内存超过阈值后，不再读取动态配置的问题。 • 修复在日志滚动时，如果生产日志速度过大，偶现重复采集的问题。 • 修复在日志上传重试多次失败时，导致的内存泄漏的问题。
v2.3.1	体验优化	• 内存限制优化。 • 达到内存限制时，超过3s的请求判定为超时。
v2.2.6	新功能	支持分离配置内外网域名。
	性能优化	修复 getip 引发的 LogListener 工作异常。
v2.2.5	新功能	支持腾讯云织云环境部署。
	体验优化	修复 getip 导致 core 的问题。
v2.2.4	体验优化	• 安装和初始化改为：tools/loglistener.sh 的子命令 install 和 init。 • 启动改成: /etc/init.d/loglistenerd start|stop|restart。
v2.2.3	体验优化	日志轮转 rename+create 不丢日志。
v2.2.2	体验优化	日志大小超过512KB自动截断。
更早版本	–	• 2.2.2版本的 LogListener 支持完全正则采集。 • 2.1.4版本的 LogListener 支持多行全文格式。 • 2.1.1版本的 LogListener 支持日志结构化。

使用 Kafka 协议上传日志

最近更新时间：2022-06-02 15:16:03

日志服务（Cloud Log Service，CLS）目前已支持使用 Kafka Producer SDK 和其他 Kafka 相关 agent 上传日志到 CLS。

使用场景

日志应用中使用 Kafka 作为消息管道是非常普遍的场景。先通过机器上的开源采集客户端或者使用 producer 直接写入收集日志，再通过消费管道提供给下游如 spark、flink 等进行消费。CLS 具备完整的 Kafka 数据管道上下行能力，以下主要介绍哪些场景适合您使用 Kafka 协议上传日志，更多 Kafka 协议消费场景请参考 Kafka 协议实时消费 文档。

- **场景1**：您已有基于开源采集的自建系统，不希望有复杂的二次改造，您可以通过修改配置文件将日志上传到 CLS。
  例如，您之前使用 ELK 搭建日志系统的客户，现在只需要通过修改 Filebeat 或者 Logstash 的配置文件，将 Output 配置（详情请参考 filebeat 配置）到 CLS，即可非常方便简洁的将日志上传。
- **场景2**：您希望通过 Kafka producer 来采集日志并上传，不必再安装采集 Agent。
  CLS 支持您使用各类 Kafka producer SDK 采集日志，并通过 Kafka 协议上传到 CLS。（详情请参考本文提供的 SDK 调用示例 ）

相关限制

- 支持 Kafka 协议版本为：0.11.0.X，1.0.X，1.1.X，2.0.X，2.1.X，2.2.X，2.3.X，2.4.X，2.5.X，2.6.X，2.7.X，2.8.X
- 支持压缩方式：gzip，snappy，lz4
- 当前使用 SASL_PLAINTEXT 认证。

配置方式

使用 kafka 协议上传日志时，需要配置一下参数：

参数	说明
链接类型	当前支持 SASL_PLAINTEXT
hosts	初始连接的集群地址，详细参见 服务入口
topic	配置为日志主题 ID。例如：76c63473-c496-466b-XXXX-XXXXXXXXXXXX
username	配置为日志集 ID。例如：0f8e4b82-8adb-47b1-XXXX-XXXXXXXXXXXX
password	格式为 ${SecurityId}#${SecurityKey}。例如：XXXXXXXXXXXXXX#YYYYYYYY

服务入口

地域	网络类型	端口号	服务入口
广州	内网	9095	gz-producer.cls.tencentyun.com:9095
	外网	9096	gz-producer.cls.tencentcs.com:9096

> 注意：
> 本文档以广州地域为例，内外网域名需用不同端口标识，其他地域请替换地址前缀。详情请参考 可用域名-Kafka上传日志。

示例

Agent 调用示例

filebeat/winlogbeat 配置

```
output.kafka:
enabled: true
hosts: ["${region}-producer.cls.tencentyun.com:9096"] # TODO 服务地址；公网端口9096，内网端口9095
```

```
topic: "${topicID}" # TODO topicID
version: "0.11.0.2"
compression: "${compress}" # TODO 配置压缩方式
username: "${logsetID}"
password: "${SecurityId}#${SecurityKey}"
```

logstash 示例

```
output {
kafka {
topic_id => "${topicID}"
bootstrap_servers => "${region}-producer.cls.tencentyun.com:${port}"
sasl_mechanism => "PLAIN"
security_protocol => "SASL_PLAINTEXT"
compression_type => "${compress}"
sasl_jaas_config => "org.apache.kafka.common.security.plain.PlainLoginModule required username='${logsetID}' password='${securityID}#${s
ecurityKEY};"
}
}
```

SDK 调用示例

Golang SDK 调用示例

```
import (
"fmt"
"github.com/Shopify/sarama"
)

func main() {
config := sarama.NewConfig()

config.Net.SASL.Mechanism = "PLAIN"
config.Net.SASL.Version = int16(1)
config.Net.SASL.Enable = true
config.Net.SASL.User = "${logsetID}" // TODO 日志集 ID
config.Net.SASL.Password = "${SecurityId}#${SecurityKey}" // TODO 格式为 ${SecurityId}#${SecurityKey}
config.Producer.Return.Successes = true
config.Producer.RequiredAcks = ${acks} // TODO 根据使用场景选择acks的值
config.Version = sarama.V0_11_0_0
config.Producer.Compression = ${compress} // TODO 配置压缩方式

// TODO 服务地址；公网端口9096，内网端口9095
producer, err := sarama.NewSyncProducer([]string{"${region}-producer.cls.tencentyun.com:9096"}, config)
if err != nil {
panic(err)
}

msg := &sarama.ProducerMessage{
Topic: "${topicID}", // TODO topicID
Value: sarama.StringEncoder("goland sdk sender demo"),
}
// 发送消息
for i := 0; i <= 5; i++ {
partition, offset, err := producer.SendMessage(msg)
if err != nil {
```

```
panic(err)
}
fmt.Printf("send response; partition:%d, offset:%d\n", partition, offset)
}

_ = producer.Close()

}
```

Python SDK 调用示例

```
from kafka import KafkaProducer

if __name__ == '__main__':
produce = KafkaProducer(
# TODO 服务地址；公网端口9096，内网端口9095
bootstrap_servers=["${region}-producer.cls.tencentyun.com:9096"],
security_protocol='SASL_PLAINTEXT',
sasl_mechanism='PLAIN',
# TODO 日志集 ID
sasl_plain_username='${logsetID}',
# TODO 格式为 ${SecurityId}#${SecurityKey}
sasl_plain_password='${SecurityId}#${SecurityKey}',
api_version=(0, 11, 0),
# TODO 配置压缩方式
compression_type="${compress_type}",
)

for i in range(0, 5):
# 发送消息 TODO topicID
future = produce.send(topic="${topicID}", value=b'python sdk sender demo')
result = future.get(timeout=10)
print(result)
```

Java SDK 调用示例

maven 依赖：

```
<dependencies>
<!--https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients-->
<dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>kafka-clients</artifactId>
<version>0.11.0.2</version>
</dependency>
</dependencies>
```

代码示例：

```
import org.apache.kafka.clients.producer.*;

import java.util.Properties;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;
```

```
public class ProducerDemo {
public static void main(String[] args) throws InterruptedException, ExecutionException, TimeoutException {
// 0.配置一系列参数
Properties props = new Properties();
// TODO 使用时
props.put("bootstrap.servers", "${region}-producer.cls.tencentyun.com:9096");
// TODO 以下值根据业务场景设置
props.put("acks", ${acks});
props.put("retries", ${retries});
props.put("batch.size", ${batch.size});
props.put("linger.ms", ${linger.ms});
props.put("buffer.memory", ${buffer.memory});
props.put(ProducerConfig.COMPRESSION_TYPE_CONFIG, "${compress_type}"); // TODO 配置压缩方式
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

props.put("security.protocol", "SASL_PLAINTEXT");
props.put("sasl.mechanism", "PLAIN");
// TODO 用户名为logsetID；密码为securityID和securityKEY的组合 securityID#securityKEY
props.put("sasl.jaas.config",
"org.apache.kafka.common.security.plain.PlainLoginModule required username='${logsetID}' password='${SecurityId}#${SecurityKey}';");

// 1.创建一个生产者对象
Producer<String, String> producer = new KafkaProducer<String, String>(props);

// 2.调用send方法
Future<RecordMetadata> meta = producer.send(new ProducerRecord<String, String>("${topicID}", ${message}));
RecordMetadata recordMetadata = meta.get(${timeout}, TimeUnit.MILLISECONDS);
System.out.println("offset = " + recordMetadata.offset());

// 3.关闭生产者
producer.close();
}
}
```

C SDK 调用示例

```
// https://github.com/edenhill/librdkafka - master
#include <iostream>
#include <librdkafka/rdkafka.h>
#include <string>
#include <unistd.h>

#define BOOTSTRAP_SERVER "${region}-producer.cls.tencentyun.com:${port}"
#define USERNAME "${logsetID}"
#define PASSWORD "${SecurityId}#${SecurityKey}"
#define TOPIC "${topicID}"
#define ACKS "${acks}"
#define COMPRESS_TYPE "${compress_type}"

static void dr_msg_cb(rd_kafka_t *rk, const rd_kafka_message_t *rkmessage, void *opaque) {
if (rkmessage->err) {
fprintf(stdout, "%% Message delivery failed : %s\n", rd_kafka_err2str(rkmessage->err));
} else {
fprintf(stdout, "%% Message delivery successful %zu:%d\n", rkmessage->len, rkmessage->partition);
}
```

```
}

int main(int argc, char **argv) {
// 1. 初始化配置
rd_kafka_conf_t *conf = rd_kafka_conf_new();

rd_kafka_conf_set_dr_msg_cb(conf, dr_msg_cb);

char errstr[512];
if (rd_kafka_conf_set(conf, "bootstrap.servers", BOOTSTRAP_SERVER, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
rd_kafka_conf_destroy(conf);
fprintf(stdout, "%s\n", errstr);
return -1;
}

if (rd_kafka_conf_set(conf, "acks", ACKS, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
rd_kafka_conf_destroy(conf);
fprintf(stdout, "%s\n", errstr);
return -1;
}

if (rd_kafka_conf_set(conf, "compression.codec", COMPRESS_TYPE, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
rd_kafka_conf_destroy(conf);
fprintf(stdout, "%s\n", errstr);
return -1;
}

// 设置认证方式
if (rd_kafka_conf_set(conf, "security.protocol", "sasl_plaintext", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
rd_kafka_conf_destroy(conf);
fprintf(stdout, "%s\n", errstr);
return -1;
}
if (rd_kafka_conf_set(conf, "sasl.mechanisms", "PLAIN", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
rd_kafka_conf_destroy(conf);
fprintf(stdout, "%s\n", errstr);
return -1;
}
if (rd_kafka_conf_set(conf, "sasl.username", USERNAME, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
rd_kafka_conf_destroy(conf);
fprintf(stdout, "%s\n", errstr);
return -1;

}
if (rd_kafka_conf_set(conf, "sasl.password", PASSWORD, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
rd_kafka_conf_destroy(conf);
fprintf(stdout, "%s\n", errstr);
return -1;
}

// 2. 创建 handler
rd_kafka_t *rk = rd_kafka_new(RD_KAFKA_PRODUCER, conf, errstr, sizeof(errstr));
if (!rk) {
rd_kafka_conf_destroy(conf);
fprintf(stdout, "create produce handler failed: %s\n", errstr);
return -1;
}
```

```
// 3. 发送数据
std::string value = "test lib kafka ---- ";
for (int i = 0; i < 100; ++i) {
retry:
rd_kafka_resp_err_t err = rd_kafka_producev(
rk, RD_KAFKA_V_TOPIC(TOPIC),
RD_KAFKA_V_MSGFLAGS(RD_KAFKA_MSG_F_COPY),
RD_KAFKA_V_VALUE((void *) value.c_str(), value.size()),
RD_KAFKA_V_OPAQUE(nullptr), RD_KAFKA_V_END);

if (err) {
fprintf(stdout, "Failed to produce to topic : %s, error : %s", TOPIC, rd_kafka_err2str(err));
if (err == RD_KAFKA_RESP_ERR__QUEUE_FULL) {
rd_kafka_poll(rk, 1000);
goto retry;
}
} else {
fprintf(stdout, "send message to topic successful : %s\n", TOPIC);
}

rd_kafka_poll(rk, 0);
}

std::cout << "message flush final" << std::endl;
rd_kafka_flush(rk, 10 * 1000);

if (rd_kafka_outq_len(rk) > 0) {
fprintf(stdout, "%d message were not deliverer\n", rd_kafka_outq_len(rk));
}

rd_kafka_destroy(rk);

return 0;
}
```

使用 Logback Appender 上传日志

最近更新时间：2022-05-13 18:27:53

简介

日志服务（Cloud Log Service，CLS）目前已支持使用 Logback Appender 上传日志到 CLS。

背景信息

Logback 是 Apache 的一个开源项目。通过使用 Logback，我们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。此外，我们还可以通过一个配置文件来灵活地进行配置，而不需要修改应用的代码。

功能优势

- 日志不落盘：产生数据通过网络发给服务端。
- 无需改造：对已使用 Logback 应用，只需简单配置即可采集。
- 异步非阻塞：高并发设计，后台异步发送，适合高并发写入。
- 资源可控制：可以通过参数控制 producer 用于缓存待发送数据的内存大小，同时还可以配置用于执行数据发送任务的线程数量。
- 自动重试： 对可重试的异常，支持配置重试次数。
- 优雅关闭： 退出前会将日志全量进行发送。
- 感知日志上报结果："运行过程中产生的异常通过 AddError 输出出来"。

工程引入和配置

maven 工程中引入依赖

```
<dependency>
<groupId>com.tencentcloudapi.cls</groupId>
<artifactId>tencentcloud-cls-logback-appender</artifactId>
<version>1.0.1</version>
</dependency>
```

修改 logback 配置文件

```
<appender name="LoghubAppender" class="com.tencentcloudapi.cls.LoghubAppender">
<!--必选项-->
<endpoint><region>.cls.tencentcs.com</endpoint>
<accessKeyId>${SecretID}</SecretID>
<accessKeySecret>${SecretKey}</SecretKey>
<topicId>${topicId}</topicId>

<!-- 可选项 详见 '参数说明'-->
<totalSizeInBytes>104857600</totalSizeInBytes>
<maxBlockMs>0</maxBlockMs>
<sendThreadCount>8</sendThreadCount>
<batchSizeThresholdInBytes>524288</batchSizeThresholdInBytes>
<batchCountThreshold>4096</batchCountThreshold>
<lingerMs>2000</lingerMs>
<retries>10</retries>
<baseRetryBackoffMs>100</baseRetryBackoffMs>
<maxRetryBackoffMs>50000</maxRetryBackoffMs>

<!-- 可选项 设置时间格式 -->
<timeFormat>yyyy-MM-dd'T'HH:mm:ssZ</timeFormat>
<timeZone>Asia/Shanghai</timeZone>
<encoder>
```

```
<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger - %msg</pattern>
</encoder>
<mdcFields>THREAD_ID,MDC_KEY</mdcFields>
</appender>
```

Logback Appender SDK

请使用 tencentcloud-cls-logback-appender。

使用 Loghub log4j Appender上传日志

最近更新时间：2022-04-07 10:55:32

简介

日志服务（Cloud Log Service，CLS）目前已支持使用 Log4j Appender 上传日志到 CLS。

背景信息

Log4j 是 Apache 的一个开源项目。通过使用 Log4j，我们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；我们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，我们可以更加细致地控制日志的生成过程。此外，我们还可以通过一个配置文件来灵活地进行配置，而不需要修改应用的代码。

Log4j 由三个重要的组件构成：

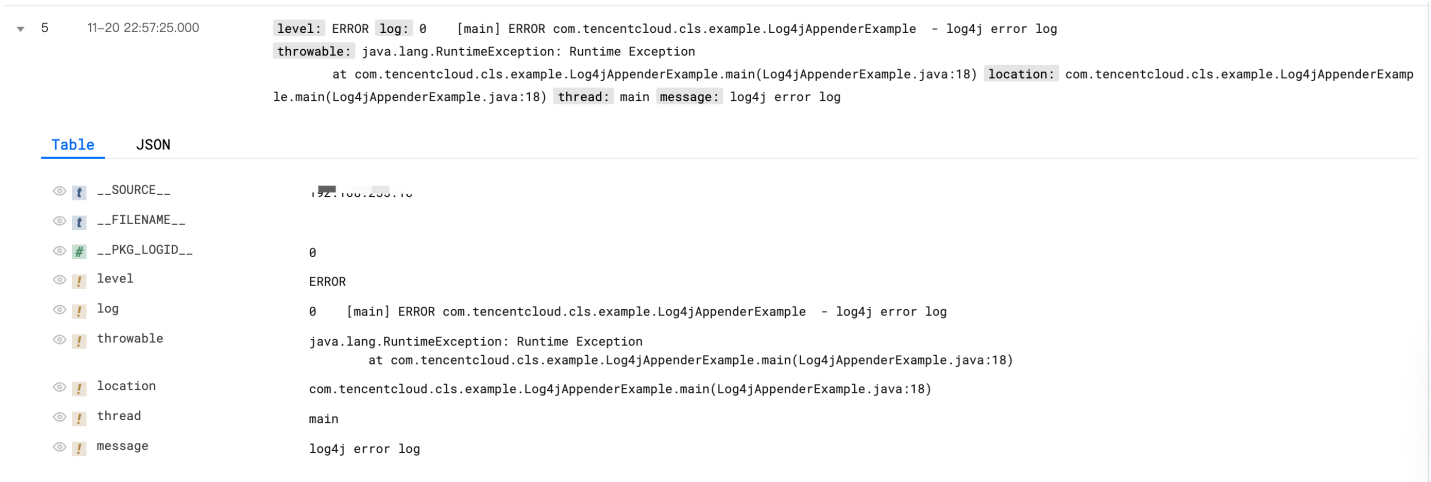
- 日志信息的优先级
  日志信息的优先级从高到低分别为 ERROR、WARN、INFO和DEBUG，分别用来指定这条日志信息的重要程度。
- 日志信息的输出目的地
  日志信息的输出目的地指定了日志将打印到控制台还是文件中。
- 日志信息的输出格式
  而输出格式则控制了日志信息的显示内容。

功能优势

- 日志不落盘：产生数据通过网络发给服务端。
- 无需改造：对已使用 Log4J 应用，只需简单配置即可采集。
- 异步非阻塞：高并发设计，后台异步发送，适合高并发写入。
- 资源可控制：可以通过参数控制 producer 用于缓存待发送数据的内存大小，同时还可以配置用于执行数据发送任务的线程数量。
- 自动重试：对可重试的异常，支持配置重试次数。
- 优雅关闭：退出前会将日志全量进行发送。
- 感知日志上报结果：通过 org.apache.log4j.helpers.LogLog 记录运行过程中产生的异常，在默认情况下 LogLog 会将信息输出到控制台。

使用 Tencent CLS Log4j Appender

通过 Tencent CLS Log4j Appender，您可以控制日志的输出目的地为腾讯云日志服务。写到日志服务中的日志的样式如下：



字段	简介
__SOURCE__	来源 IP
__FILENAME__	文件名称
level	日志级别
location	日志打印语句的代码位置

字段	简介
message	日志内容
throwable	日志异常信息（只有记录了异常信息，这个字段才会出现）
thread	线程名称
time	日志打印时间（可以通过 timeFormat 或 timeZone 配置 time 字段呈现的格式和时区）
log	自定义日志格式

工程引入和配置

maven 工程中引入依赖

```
<dependency>
<groupId>com.tencentcloudapi.cls</groupId>
<artifactId>tencentcloud-cls-log4j-appender</artifactId>
<version>1.0.2</version>
</dependency>
```

修改 Log4J 配置文件

```
#loghubAppender
log4j.appender.loghubAppender=com.tencentcloudapi.cls.LoghubAppender
#日志服务的 http 地址，必选参数
log4j.appender.loghubAppender.endpoint=ap-guangzhou.cls.tencentcs.com
#用户身份标识，必选参数
log4j.appender.loghubAppender.accessKeyId=
log4j.appender.loghubAppender.accessKeySecret=
#设置log字段的格式，必选参数
log4j.appender.loghubAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.loghubAppender.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
#指定日志主题，必选参数
log4j.appender.loghubAppender.topicID =
#指定日志来源，可选参数
log4j.appender.loghubAppender.source =
#单个 producer 实例能缓存的日志大小上限，默认为 100MB。
log4j.appender.loghubAppender.totalSizeInBytes=104857600
#如果 producer 可用空间不足，调用者在 send 方法上的最大阻塞时间，默认为 60 秒。为了不阻塞打印日志的线程，强烈建议将该值设置成 0。
log4j.appender.loghubAppender.maxBlockMs=0
#执行日志发送任务的线程池大小，默认为可用处理器个数。
log4j.appender.loghubAppender.sendThreadCount=8
#当一个 ProducerBatch 中缓存的日志大小大于等于 batchSizeThresholdInBytes 时，该 batch 将被发送，默认为 512 KB，最大可设置成 5MB。
log4j.appender.loghubAppender.batchSizeThresholdInBytes=524288
#当一个 ProducerBatch 中缓存的日志条数大于等于 batchCountThreshold 时，该 batch 将被发送，默认为 4096，最大可设置成 40960。
log4j.appender.loghubAppender.batchCountThreshold=4096
#一个 ProducerBatch 从创建到可发送的逗留时间，默认为 2 秒，最小可设置成 100 毫秒。
log4j.appender.loghubAppender.lingerMs=2000
#如果某个 ProducerBatch 首次发送失败，能够对其重试的次数，默认为 10 次。
#如果 retries 小于等于 0，该 ProducerBatch 首次发送失败后将直接进入失败队列。
log4j.appender.loghubAppender.retries=10
#该参数越大能让您追溯更多的信息，但同时也会消耗更多的内存。
log4j.appender.loghubAppender.maxReservedAttempts=11
#首次重试的退避时间，默认为 100 毫秒。
#Producer 采用指数退避算法，第 N 次重试的计划等待时间为 baseRetryBackoffMs * 2^(N-1)。
log4j.appender.loghubAppender.baseRetryBackoffMs=100
```

```
#重试的最大退避时间，默认为 50 秒。
log4j.appender.loghubAppender.maxRetryBackoffMs=50000
#设置时间格式，可选参数
log4j.appender.loghubAppender.timeFormat=yyyy-MM-dd'T'HH:mm:ssZ
#设置时区为东八区，可选参数
log4j.appender.loghubAppender.timeZone=Asia/Shanghai
#输出 DEBUG 级别及以上的消息
log4j.appender.loghubAppender.Threshold=DEBUG
```

Log4j Appender SDK

- Log4j 1.x版本：请使用 tencentcloud-cls-log4j-appender。
- Log4j 2.x版本：请使用 tencentcloud-cls-log4j2-appender。

SDK 采集

最近更新时间：2022-05-23 12:04:47

简介

为了能让您更高效地使用日志服务（Cloud Log Service，CLS），日志服务专门为日志上传量身打造了多个语言版本的 SDK（Software Development Kit），您可以根据业务需求选择语言版本使用。

使用前须知

1. SDK 对日志服务的数据接入接口做了统一封装，降低了使用上传日志的难度，您可以使用对应语言的 SDK 进行日志上传。
2. 实现日志服务日志的 ProtoBuffer 格式封装，让您在写入日志时不需要关心 ProtoBuffer 格式的具体细节。
3. 实现日志服务 API 中定义的压缩方法，让您不用关心压缩实现的细节。部分语言的 SDK 支持启用压缩模式写入日志（默认为使用压缩方式）。
4. 提供统一的异步发送、资源控制、自动重试、优雅关闭、感知上报等功能，使上报日志功能更完善。

SDK 列表

下表列举了日志服务不同语言 SDK GitHub 源码：

SDK 语言	GitHub 源码
Java	tencentcloud-cls-sdk-java
Go	tencentcloud-cls-sdk-go
Node.js	tencentcloud-cls-sdk-js
Android	tencentcloud-cls-sdk-android
C++	tencentcloud-cls-sdk-c++

数据导入
导入 COS 数据

最近更新时间：2021-12-24 16:05:25

概述

日志服务（Cloud Log Service，CLS）投递功能打通产品生态上游链路，将腾讯云对象存储（Cloud Object Storage，COS）中的数据导入到日志服务，实现日志数据的查询分析、加工等操作，挖掘日志数据价值。您只需要在日志服务控制台进行简单的配置即可完成数据导入。

前提条件

- 开通日志服务，创建日志集与日志主题，并成功采集到日志数据。
- 开通对象存储，并确保待导入的文件已经上传到 COS Bucket 中，更多信息请参考 上传对象。
- 设置当前操作账号拥有访问 COS 的权限，即已授权日志服务使用 CLS_QcsRole 角色访问您的 COS 资源。

配置流程

1. **选择日志主题**：可以选择现有的日志主题（topic）或者创建新的日志主题（topic），用于存储从 COS 导入到 CLS 的数据。
2. **导入数据源配置**：需要配置想要导入的 COS 对象路径，以及压缩方式（gzip/lzop/snappy/无压缩）。
3. **解析配置**：需要配置导入文件的解析格式，目前支持单行全文/JSON/CSV。
4. **索引配置**：需要配置目前 topic 的索引配置，且检索必须开启索引配置。如果选择已存在的日志主题，则新索引配置只对修改后的数据生效。

操作步骤

步骤1：选择日志主题

- 如果您想选择新的日志主题，可执行如下操作：
  i. 登录 日志服务控制台。
  ii. 在左侧导航栏中，单击**概览**，进入概览页面。
  iii. 在其他日志栏下，找到**对象存储COS数据导入**，单击**立即接入**。

  iv. 在创建日志主题页面，根据实际需求，输入日志主题名称，配置日志保存时间等信息，单击**下一步**。
- 如果您想选择现有的日志主题，可执行如下操作：
  i. 登录 日志服务控制台。
  ii. 在左侧导航栏中，单击**日志主题**，选择需要投递的日志主题，进入日志主题管理页面。
  iii. 选择**采集配置**页签，在**数据导入配置**栏下单击**新增**。

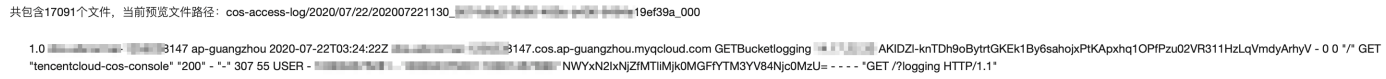
步骤2：数据源配置

1. 在数据源配置页面，依次配置如下信息：

配置项	说明	规则	是否必填
任务名称	配置导入任务的名称。	字母、数字、_和-	是
存储桶地域	配置需要导入的文件所在存储桶的地域位置。如果文件所在地域和导入日志主题所在地域不同，则会因跨地域访问而产生外网费用。	列表选择	是
bucket	选择需要导入的文件所在存储桶。下拉框会列出当前选择地域下的所有存储桶供您选择。	列表选择	是

文件前缀	输入待导入的 OSS 文件所在文件夹的前缀，用于准确定位待导入的文件夹，支持输入文件前缀 csv/，或完整文件路径 csv/object.gz。	输入框	是
压缩模式	选择待导入的 OSS 文件的压缩格式，日志服务根据对应格式进行解压并读取数据，支持 gzip/lzop/snappy/ 无压缩。	列表选择	是

2. 单击**预览**，系统会选择符合配置规则的一条路径数据进行展示，并提示输入的文件前缀下共有多少文件。如下图所示：



3. 确认预览无误后，单击**下一步**。

步骤3：解析配置

1. 在解析配置页面，配置如下主要信息：
   - 提取模式：可选择单行全文，JSON，CSV 三种提取模式。
     - 单行全文：每条日志将被解析为键值为__CONTENT__的一行完全字符串，开启索引后可通过全文检索搜索日志内容。日志时间为采集时间为准。

     - JSON：支持按照 JSON 格式提取键值对。

- CSV：可根据您指定分隔符切分每条日志。需要您指定切分后每个字段的键值名称，无效字段即无需采集的字段可填空，不支持所有字段均为空。

  - 过滤器：
    - 过滤器旨在您根据业务需要添加日志采集过滤规则，帮助您筛选出有价值的日志数据。过滤规则为 Perl 正则表达式，所创建的过滤规则为命中规则，即匹配上正则表达式的日志才会被采集上报。
    - 分隔符格式日志需要根据所自定义的键值对来配置过滤规则。例如，样例日志使用分隔符模式解析后，您希望 status 字段为 400 或 500 的所有日志数据被采集，那么 key 处配置 status，过滤规则处配置 400|500。
  - 使用采集时间：开启后可根据采集时间标记每条日志时间，或关闭此项指定某一字段作为日志时间。

> 说明：
>
> - 日志时间单位为：秒，若时间格式填写错误日志时间将以采集时间为准。
> - 日志的时间属性有两种方式来定义：采集时间和原始时间戳。

- 采集时间：日志的时间属性由日志服务 CLS 从 COS 导入时间决定。
- 原始时间戳：日志的时间属性由原始日志中时间戳决定。
- 日志的原始时间戳作为日志时间属性。
- 关闭采集时间状态，在时间键和时间格式解析处，填写原始时间戳的时间键以及对应的时间解析格式。时间解析格式详情参见 配置时间格式。

- 分隔符：系统根据确定的分隔符将日志样例进行切分，并展示在抽取结果栏中，您需要为每个字段定义唯一的 key。目前，日志采集支持多种分隔符，常见的分隔符有：空格、制表符、逗号、分号、竖线。若您的日志数据所采用的分隔符是其他符号（例如 :::），也可以通过自定义分词符进行解析。

2. 单击**下一步**。

步骤4：索引配置

1. 在索引配置页面，配置如下信息：

- 索引状态：确认是否开启。
- 全文索引：确认是否需要设置大小写敏感。
  - 全文分词符：默认为“@&()='",;:<>[]{}/ \n\t\r”，确认是否需要修改。
  - 是否包含中文：确认是否开启。
- 键值索引：默认关闭，您可根据 key 名按需进行字段类型、分词符以及是否开启统计分析的配置。若您需要开启键值索引，可将  设置为  。

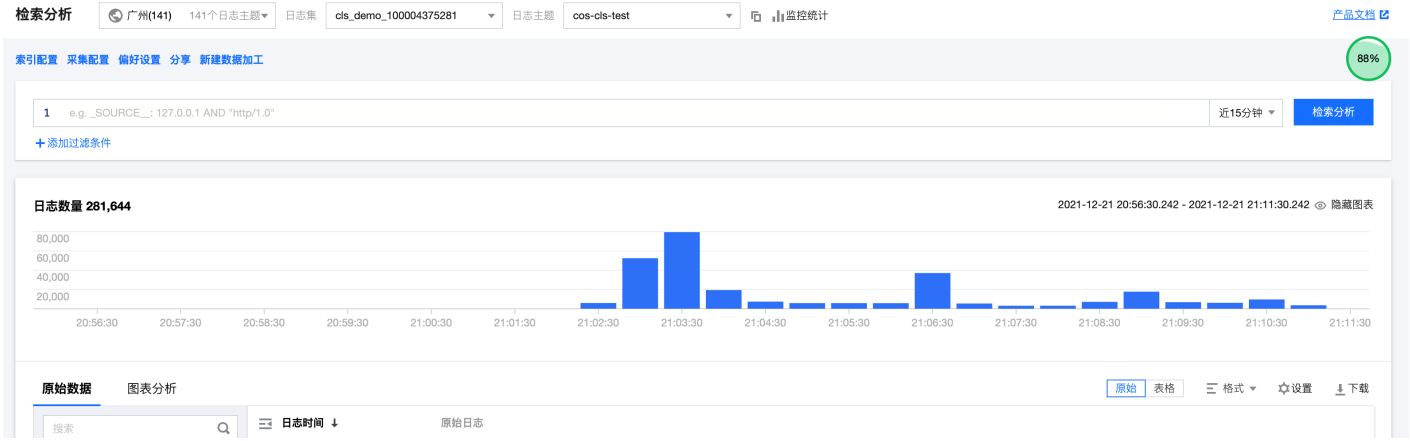
> **注意：**
>
> - 检索必须开启索引配置，否则无法检索
> - 索引规则编辑后仅对新写入的日志生效，已有数据不会更新。

2. 单击**提交**，完成导入配置。

相关操作

导入进度查询

1. 当前日志主题下存在 COS 导入任务时，单击**检索**，进入**检索分析**页面查看当前导入任务的进度。



2. 在检索分析页面右上角的进度悬浮球显示目前完成的导入进度。单击悬浮球，右侧浮窗会展开导入任务详情。

3. 在任务详情页面，单击**查看详情**，即可跳转采集配置页查询导入的详细配置。

云产品日志接入

最近更新时间：2022-04-18 18:27:49

日志服务（Cloud Log Service，CLS）支持采集多个云产品的日志数据，如 SCF 运行日志，CDN 实时日志，TKE 容器日志，CLB 访问日志等。用户可以使用 CLS 服务进行操作记录查看，运营数据统计分析，运行状况监控告警等多种日志数据处理功能。

当前 CLS 支持采集的云产品日志如下：

云产品名称	采集指引
负载均衡（Cloud Load Balancer，CLB）	CLB 控制台配置，配置指引
内容分发网络（Content Delivery Network，CDN）	CDN 控制台配置，配置指引
云服务器（Cloud Virtual Machine，CVM）	安装配置 LogListener，采集指引
容器服务（Tencent Kubernetes Engine，TKE）	TKE 控制台配置，配置指引
云函数（Serverless Cloud Function，SCF）	SCF 控制台配置，配置指引
云审计（CloudAudit）	CloudAudit 控制台配置，配置指引
对象存储（Cloud Object Storage，COS）	COS 控制台配置，配置指引
网络流日志（Flow Logs，FL）	FL 控制台配置，配置指引
腾讯云 TI 平台 TI-ONE	TI-ONE 控制台配置，配置指引
Web 应用防火墙（Web Application Firewall，WAF）	WAF 控制台配置，配置指引
消息队列 CKafka（Cloud Kafka）	CKafka 控制台配置，配置指引

日志存储
存储类型概述

最近更新时间：2022-07-01 16:57:20

根据用户对日志不同的检索时延性及日志处理能力需求，日志服务（Cloud Log Service，CLS）提供两种存储类型：**标准存储**和**低频存储**。

> ⚠ 注意：
> 目前日志低频存储仅支持北京、广州、上海、香港、南京、新加坡、硅谷、法兰克福地域。如果您日志主题所在地域暂未支持，可联系 在线客服 申请。

标准存储

适用于用户对日志有统计分析的需求，提供日志的秒级检索，实时统计分析，实时监控，流式消费等应用能力。

应用场景

- 运维监控排障：对线上问题进行实时诊断，利用日志秒级检索能力，快速检索散落在多个机器的日志内容，定位故障原因并进行修复；基于日志实时计算各项质量指标，一旦指标超过阈值即告警，便于开发运维人员第一时间发现问题修复故障。
- 流式处理：采集散落在多台机器上 PB 级别埋点日志数据，实时流式拉取至用户自建的大数据处理集群用于后续数据湖计算，例如用于推荐系统的模型数据计算业务等。

低频存储

适用于访问频率较低且无统计分析需求的日志，如审计归档日志。低频存储提供全文检索日志的能力，满足用户对历史日志的回溯检索，归档存储等诉求，整体使用成本相比**标准存储**降低80%，详情请查看 低频存储简介。

应用场景

- 历史日志存储：日志数据的爆炸性增长，对于数月乃至数年的日志进行大规模存储和分析，其成本变得十分高昂。这可能会导致用户删除有价值的数据，而错过了长期数据可能产生的重要洞察和见解。**低频存储**能满足用户以低成本的方式对历史数据进行大规模的统计分析，历史回溯。
- 非关键业务日志：排查异常时，研发人员更多需要关注 ERROR/WARN 日志以及针对其进行监控告警，info 等非关键业务日志只作为归档存储，在特定场景下才需要触发日志的检索分析。一般用户对该类日志检索时延无特定要求，使用**低频存储**存储非关键业务日志能显著较低用户使用成本，且满足用户低频检索的诉求。
- 审计日志场景：将操作审计，安全审计的日志采集至**低频存储**，通过 CLS 低频检索能力分析其访问行为，例如某个账号、某个对象的操作记录等，判断是否存在违规操作。同时，满足合规审计需求，日志存储180天以上。

功能对比

功能点	低频存储	标准存储
索引建立	✓（只支持全文索引）	✓
上下文检索	✓	✓
快速分析	×	✓
全文检索	✓（亿条数据2秒响应）	✓（亿条数据0.5秒响应）
键值检索	×	✓
日志下载	✓	✓
SQL 分析	×	✓
仪表盘	×	✓
监控告警	×	✓
投递到 COS	✓	✓
投递到 Ckafka	✓	✓
投递到 ES	✓	✓
投递到 SCF	✓	✓
日志消费	✓	✓

功能点	低频存储	标准存储
数据加工	✓	✓

低频存储简介

最近更新时间：2022-07-01 16:57:26

日志服务（Cloud Log Service，CLS）低频存储定位是以低成本方案解决海量低频日志的检索和存储问题，适用于用户对日志无统计分析要求，且日志保存时间较长的场景。低频存储提供全文检索、上下文检索、投递到对象存储（Cloud Object Storage，COS）、投递到 Ckafka，日志消费等能力，满足用户对历史日志的回溯检索，归档存储等诉求。低频存储不支持 SQL 分析，监控告警，仪表盘等功能。

> ⚠ 注意：
> 目前日志低频存储仅支持北京、广州、上海、香港、南京、新加坡、硅谷、法兰克福地域。如果您日志主题所在地域暂未支持，可联系 在线客服 申请。

特性

- **成本低廉：**仅收取写入流量与日志存储量费用，整体使用成本相比**CLS-标准存储**降低73%。
- **简单易用：**检索语法兼容 Lucene，无额外学习成本；无需配置索引，写入即可检索。
- **稳定可靠：**基于全新自研的计算存储分离的日志系统，水平可扩展，高可用，使用灵活。

适用场景

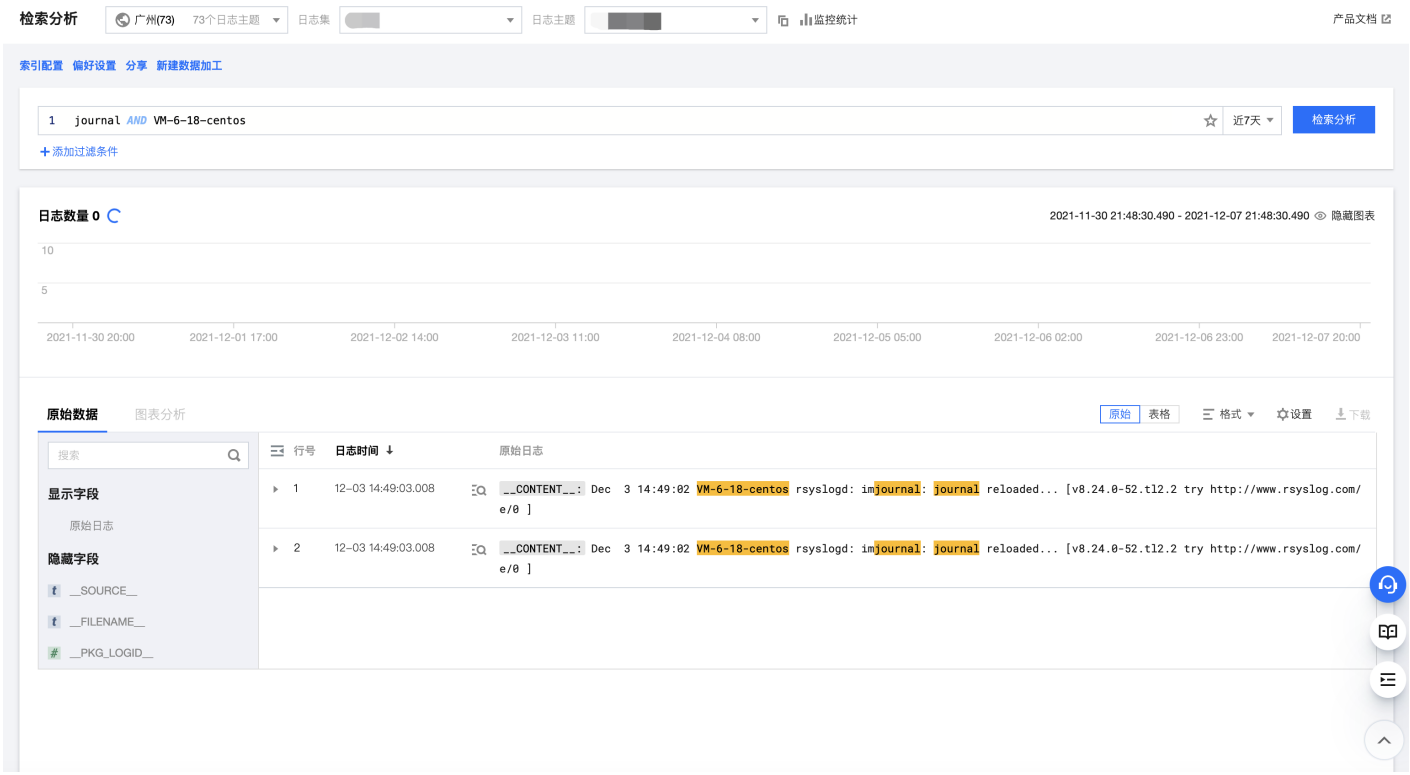
场景	说明
长期存储场景	例如审计/安全日志场景，长期低频存储的存储价格对比标准存储可节省80%以上
简单使用场景	不需要频繁对数据进行统计分析的场景，例如运维日志分析场景，无需仪表盘、监控告警等高阶功能

使用方法

1. 创建日志主题，并将“存储类型”选择为**低频存储**。详情请参考 管理日志主题。

2. 将日志采集至目标 topic 上，详情请参考 采集方式。
3. 进入目标日志主题检索页，输入全文检索语句，例如a AND b NOT c。

4. 单击**检索分析**，查看检索结果。



低频存储与标准存储功能对比

功能点	低频存储	标准存储
索引建立	✓（只支持全文索引）	✓
上下文检索	✓	✓
快速分析	×	✓
全文检索	✓（亿条数据2秒响应）	✓（亿条数据0.5秒响应）
键值检索	×	✓
日志下载	✓	✓
SQL 分析	×	✓
仪表盘	×	✓
监控告警	×	✓
投递到 COS	✓	✓
投递到 Ckafka	✓	✓
投递到 ES	✓	✓
投递到 SCF	✓	✓
日志消费	✓	✓
数据加工	✓	✓

公测结束后低频存储计费标准

示例：每天写入2TB原始日志数据，保存30天，压缩比1：8，索引比例100%，以广州地域定价为例。

计费项	标准存储	低频存储	说明

计费项	标准存储	低频存储	说明
写流量	1350元	1350元	价格均为0.18元/GB/天
索引流量	21500元	6140元	标准存储：0.35元/GB/天 低频存储：0.1元/GB/天
日志存储	2650元	570元	标准存储：0.0115元/GB/天 低频存储：0.0025元/GB/天
索引存储	21200元	4600元	标准存储：0.0115元/GB/天 低频存储：0.0025元/GB/天
其他费用	极少	极少	包含接口调用次数，分区费用等
总价（月）	46700元	12660元（下降73%）	-

检索分析
概述及语法规则

最近更新时间：2022-07-01 09:25:38

概述

日志服务提供日志检索及分析能力，可通过检索分析语句检索匹配特定条件的日志，或针对匹配的日志使用 SQL 进行统计分析，获取日志条数、平均响应时间和错误日志占比等统计结果。

检索分析语句由检索条件和 SQL 语句组成，两者通过竖线 | 分割：

```
[检索条件] | [SQL语句]
```

- 检索条件：指定日志需要匹配的条件，返回符合该条件的日志。例如使用status:404检索响应状态码为404的应用请求日志。语法规则可参见 检索条件语法规则。
- SQL语句：针对符合检索条件的日志进行统计分析，返回统计分析结果。例如使用status:404 | select count(*) as logCounts统计响应状态码为404的应用请求日志数量。SQL 语句符合 SQL-92规范，语法规则可参见 SQL 语句语法规则。

> 说明：
>
> - 只需要检索日志，不需要统计分析时，可省略其中的管道符|及 SQL 语句。
> - 检索分析语句为空时代表检索指定时间范围内的所有日志。

检索条件根据其是否指定了日志字段，分为全文检索和键值检索两种形式：

- 全文检索：无需指定字段名称，日志内任意字段的值符合检索条件即匹配，例如使用error检索所有出现过 error 的日志。
- 键值检索：需指定字段名称，日志内指定字段的值符合检索条件即匹配，例如使用level:error检索日志级别为 error 的日志。

前提条件

- **使用检索条件：**
  - 全文检索：在索引配置中开启了全文索引
  - 键值检索：
    - 采集日志时按字段对日志进行了提取，即 日志结构化。
    - 配置索引 时为需要检索的字段开启了键值索引。
- **使用 SQL 语句：**
  - 采集日志时按字段对日志进行了提取，即 日志结构化。
  - 配置索引 时为需要检索的字段开启了键值索引且**开启统计**。

检索条件语法规则

语法规则

语法	说明
AND	“与”逻辑操作符，例如 level:ERROR AND pid:1234
OR	“或”逻辑操作符，例如 level:ERROR OR level:WARNING
NOT	“非”逻辑操作符，例如 level:ERROR NOT pid:1234
()	分组操作符，控制逻辑运算优先级，例如 (ERROR OR WARNING) AND pid:1234
:	冒号，表示作用于的 key 字段，即键值检索，例如 level:ERROR
""	双引号，引用一个短语，日志需包含短语内的各个词，且各个词的顺序保持不变，例如 name:"john Smith"
*	通配符查询，匹配零个、单个、多个字符，例如 host:www.test*.com 不支持前缀模糊查询，详情请参见 模糊检索 还可以通过 key:* 的方式查询字段（key）存在的日志，等价于 _exists_:key

语法	说明
?	通配符查询，匹配单个字符，例如 host:www.te?t.com 与*类似，不支持前缀模糊查询
>	范围操作符，表示大于某个数值，例如 status:>400
>=	范围操作符，表示大于等于某个数值，例如 status:>=400
<	范围操作符，表示小于某个数值，例如 status:<400
<=	范围操作符，表示小于等于某个数值，例如 status:<=400
TO	“范围”逻辑操作符，例如 request_time:[0.1 TO 1.0]
[]	范围操作符，包含边界值的范围，例如 age:[20 TO 30]
{}	范围操作符，不包含边界值的范围，例如 age:{20 TO 30}
\	转义符号，转义后的字符表示符号本身，例如 url:\/images\/favicon.ico 如不想使用转义符，可使用""包裹，例如url:"/images/favicon.ico"，但需注意，双引号内的词会被当作一个短语，日志需包含短语内的各个词，且各个词的顺序保持不变
exists	_exists_:key，返回 key 存在的日志，例如 _exists_:userAgent 表示搜索存在 userAgent 字段的日志

注意：

- 语法区分大小写，例如 AND、OR 表示检索逻辑操作符，而 and、or 视为普通文本。
- 多个检索条件使用空格连接时，视为”或“逻辑，例如 warning error 等价于 warning OR error。
- 检索关键字中存在特殊字符时，需使用转义符进行转义，特殊字符包括 + - && || ! ( ) { } [ ] ^ " ~ * ? : \。
- 同时使用 AND 和 OR 逻辑运算符时，请使用()对检索条件进行分组，以明确逻辑优先级，例如(ERROR OR WARNING) AND pid:1234。

示例

场景	语句
检索来源为某台机器的日志	__SOURCE__:127.0.0.1 或 __SOURCE__:192.168.0.*
检索来源为某个文件的日志	__FILENAME__:"/var/log/access.log"或__FILENAME__:\/var\/log\/*.log
检索包含 ERROR 的日志	ERROR
检索失败的日志（状态码大于400）	status:>400
检索 GET 请求中失败（状态码大于400）的日志	method:GET AND status:>400
检索 ERROR 或 WARNING 级别的日志	level:ERROR OR level:WARNING
检索非 INFO 级别 的日志	NOT level:INFO
检索192.168.10.10主机上非 INFO 级别的日志	__SOURCE__:192.168.10.10 NOT level:INFO
检索192.168.10.10主机上/var/log/access.log 文件中不包含 INFO 级别的日志	(__SOURCE__:192.168.10.10 AND __FILENAME__:"/var/log/access.log") NOT level:INFO
检索192.168.10.10主机上 ERROR 或 WARNING 级别的日志	__SOURCE__:192.168.10.10 AND (level:ERROR OR level:WARNING)
检索 4XX 状态码的日志	status:[400 TO 500}
检索元数据中容器名为 nginx 的日志	__TAG__.container_name:nginx
检索元数据中容器名为 nginx，且请求延时大于1s 的日志	__TAG__.container_name:nginx AND request_time:>1
检索包含 message 字段的日志	message:* 或 _exists_:message
检索不包含 message 字段的日志	NOT _exists_:message

模糊检索

在日志服务中进行模糊检索，需要在词的中间或末尾加上通配符，可使用 * 匹配零个、单个、多个字符，或 ? 匹配单个字符，例如：

- IP:192.168.1.* 表示在日志中查找 IP 为192. 168.1开头的日志，例如192. 168.1.1、192. 168.1.34等。
- host:www.te?t.com 表示在日志中查找 host 为 www.te 开头、t.com 结尾且中间包含1个字符的日志，例如 www.test.com、www.telt.com 等。

> ⚠ 注意：
>
> - *或?不能用在词的开头，即不支持前缀模糊检索。
> - long 和 double 类型字段不支持使用*或?进行模糊检索，可以使用数值范围进行检索，例如 status:[400 TO 500}。

如果您需要使用前缀模糊查询，可使用如下方法替代：

- 添加分词符：例如日志为 host:www.test.com、host:m.test.com，需要查询字段中间包含 test 的日志，可为该字段添加分词符.，便可以直接使用 host:test 对日志进行检索。
- 使用 SQL中的 LIKE 语法：例如 * | select * where host like '%test%'，但这种方式相比检索条件性能较差，不适合日志数据量过大的场景。

SQL 语句语法规则

语法规则

语法	说明
SELECT	用于从表中选取数据，默认从当前日志主题中获取符合检索条件的数据，例如level:ERROR | select *，非嵌套子查询不需要 from 子句
AS	为列名称（KEY）指定别名，例如level:ERROR | select level as log_level
GROUP BY	用于结合聚合函数，根据一个或多个列（KEY）对结果集进行分组，例如 level:* | select count(*) as logCounts,level group by level
ORDER BY	用于根据指定的 KEY 对结果集进行排序，例如 level:* | select count(*) as logCounts,level group by level order by logCounts desc
LIMIT	用于限制由 SELECT 语句返回的数据数量，例如 level:* | select count(*) as logCounts,level group by level order by logCounts desc limit 5 注意：默认limit 100，最大支持10000
WHERE	用于对查询到的原始数据进行过滤，例如level:ERROR | select * where host like '%test%' 注意： 1. SQL 中的过滤功能相比检索条件性能较差，建议尽可能的使用检索条件满足数据过滤需求，例如使用status:>400 | select count(*) as logCounts 代替* | select count(*) as logCounts where status>400 以更快的获得统计结果 2. WHERE 中不能使用 AS 别名，例如level:* | select level as log_level where log_level='ERROR'，该语句执行会报错，因为其不符合 SQL-92 规范
HAVING	用于对分组聚合后的数据进行过滤，与WHERE的区别在于其作用于分组（GROUP BY）之后，排序（ORDER BY）之前，而WHERE作用于聚合前的原始数据，例如level:* | select count(*) as logCounts,level group by level having count(*) > 100
嵌套子查询	针对一些复杂的统计分析场景，需要先对原始数据进行一次统计分析，再针对该分析结果进行二次统计分析，这时候需要在一个 SELECT 语句中嵌套另一个 SELECT 语句，这种查询方式称为嵌套子查询。例如* | select compare(PV, 86400) from (select count(*) as PV)

> ⚠ 注意：
>
> - SQL 语句对大小写不敏感，SELECT 等效于 select。
> - 字符串必须使用单引号''包裹，无符号包裹或被双引号""包裹的字符表示字段或列名。例如'status'表示字符串 status，status或"status"表示日志字段 status。
> - 字符串内本身包含单引号'时，需使用''（两个单引号）代表单引号本身。例如'{''version'': ''1.0''}'表示原始字符串{'version': '1.0'}。字符串内本身包含双引号"时无需特殊处理。
> - 不需要在末尾加分号表示 SQL 结束。

SQL 函数

日志服务支持大量的 SQL 函数，全部函数详见 SQL 函数，此处列举一些常用的函数：

语法	说明

语法	说明
字符串函数	支持字符串连接、分割、长度计算和大小写转换等
日期和时间函数	支持时间格式转换、按时间分组统计和时间间隔计算等
IP 地理函数	支持从 IP 解析地理信息等
URL 函数	支持从 URL 获取域名、参数和编解码等
通用聚合函数	支持统计日志条数、数值最大值、最小值和平均值等
估算函数	支持统计唯一值个数、数值 P95/P90 分位值等
类型转换函数	支持转换变量类型，常用于对参数的变量类型有特殊要求的函数内
逻辑函数	AND、OR 和 NOT 等逻辑运算
运算符	+、−、*、/等算术运算和>、<等比较运算
条件表达式	CASE WHEN、IF 等条件判断
数组函数	获取数组元素等
同环比函数	对比当前时间周期内的计算结果与 n 秒之前时间周期内的计算结果
JSON 函数	获取 JSON 对象、JSON 类型转换等

示例

场景	语句
统计 GET 请求中失败（状态码大于400）的日志条数	method:GET AND status:>400 | select count(*) as logCounts
按分钟统计 GET 请求中失败（状态码大于400）的日志条数	method:GET AND status:>400 | select histogram(cast(__TIMESTAMP__ as timestamp),interval 1 minute) as analytic_time, count(*) as log_count group by analytic_time order by analytic_time limit 1000
统计请求数量最大的5个 URL	* | select count(*) as PV,URL group by URL order by count(*) desc limit 5
统计平均响应耗时大于1000ms的 URL，并按耗时倒排	* | select avg(responseTime) as time_avg,URL group by URL having avg(responseTime)>1000 order by avg(responseTime) desc limit 10000
统计失败请求百分比	* | select count_if(status>400)*100.00/count(*) as "失败请求百分比"
统计每个 URL 失败请求百分比，并按占比倒排	* | select count_if(status>400)*100.00/count(*) as "失败请求百分比",URL group by URL order by count_if(status>400)*100.00/count(*) desc limit 10000 或 * | select "失败请求百分比",URL from (select count_if(status>400)*100.00/count(*) as "失败请求百分比",URL group by URL) order by "失败请求百分比" desc limit 10000
统计每个省份的请求数量	* | select ip_to_province(ip) as province, count(*) as PV group by province order by PV desc limit 100

限制说明

指标	限制说明	备注
SQL 结果条数	SQL 返回结果条数最大10000条	默认返回100条结果，可通过 LIMIT 最大返回10000条结果
内存占用量	每次 SQL 执行占用的服务端内存不能超过3GB	通常在使用 group by、distinct()、count(distinct()) 时可能触发该限制，是由于被统计的字段在通过 group by 或 distinct() 去重后值过多导致的。建议优化查询语句，使用值更少的字段对数据进行分组统计；或使用 approx_distinct() 替代 count(distinct())。

操作步骤

1. 登录 日志服务控制台。

2. 在左侧导航栏中，选择**检索分析**，进入检索分析页面。
3. 选择待检索的**日志集**和**日志主题**。
4. 输入**检索分析语句**，选择**时间范围**，可以选择近1小时、近4小时、近1天、近3天和自定义时间范围等
5. 单击**检索分析**
6. 当检索分析语句**仅包含检索条件**时：可在**原始数据**中查看匹配检索条件的日志，默认按日志时间倒排。
7. 当检索分析语句**包含SQL语句**时：可在**图表分析**中查看分析结果，并可更改图表类型，以更加直观的查看统计结果。同时还会在“原始数据”中查看符合分析语句中检索条件的日志，以便于对比分析统计结果及原始日志。

> 说明：
>
> - 在图表分析中，可单击**添加至仪表盘**将当前图表添加至新的或已有的仪表盘。
> - 在检索分析语句输入框中，可点击输入框右侧的**收藏**和**快速添加告警**图标收藏当前语句，或新增一条告警策略。

常见问题

- 检索不到日志
- 检索条件不生效
- 检索结果不准确
- 检索分析报错

配置索引

最近更新时间：2022-05-26 14:42:53

概述

索引配置是使用日志服务（Cloud Log Service，CLS）进行检索分析的必要条件，只有开启索引才能对日志进行检索分析。而且不同的索引规则也会产生不同的日志检索分析效果，本文档会详细介绍索引的配置规则及原理。

索引配置的核心是对原始日志进行 分词，以快速且便捷的根据特定的检索条件检索日志，同时在索引配置中还可以针对特定的字段“开启统计”，便于使用 SQL 对日志进行统计分析。日志服务包含以下三类索引：

类别	描述	配置方式
全文索引	全文索引将原始日志整体切分为多个分词进行索引构建，检索时直接通过关键词进行检索（即全文检索） 例如输入 error 表示检索包含 error 关键词的日志	控制台：在索引配置页面中，开启全文索引
键值索引	键值索引将原始日志按字段（即 key:value）分别切分为多个分词进行索引构建，检索时基于键值方式进行检索（即键值检索） 例如输入 level:error 表示检索 level 字段中包含 error的日志	控制台：在索引配置页面中，开启键值索引，并填写对应的字段名称（即key），例如 level
元数据索引	元数据索引也是键值索引的一种，但字段名称会以__TAG__作为前缀进行标识，常用于对日志进行分类 例如输入__TAG__.region:"ap-beijing" 表示检索元数据中 region 字段为ap-beijing 的日志	控制台：在索引配置页面中，开启键值索引，并填写对应元数据字段名城（即 key），例如 __TAG__.region

注意：

- 配置索引会产生相应的索引流量费用及索引存储费用，详情可参见 计费概述。如需节省日志服务使用成本，可参见 节省产品使用成本。
- 索引关闭时采集的日志数据无法被检索，从开启索引到支持日志检索分析约存在一分钟左右的延迟。
- 只有键值索引中“开启统计”的字段支持使用 SQL 进行统计分析。
- 索引规则编辑（包括新增/编辑/删除字段、调整分词符配置等在内的所有操作）后仅对新写入的日志生效，已有数据不会更新。

前提条件

使用 Loglistener 采集日志时，如果采集配置中提取模式为单行全文或多行全文，日志原文存储在 __CONTENT__ 字段中，仅支持配置全文索引。如需为其中的部分内容配置键值索引或开启统计，需在采集配置中进行 日志结构化 处理，使用除单行全文和多行全文以外的日志采集提取模式。

全文索引

全文索引将原始日志整体切分为多个分词进行索引构建，检索时直接通过关键词进行检索（即全文检索）。

配置项	功能描述
全文分词符	对原始日志进行分词的字符集合，仅支持英文符号，控制台默认分词符 @&?|#()='",;:<>[]{}/ \n\t\r
大小写敏感	检索时是否对大小写敏感 例如日志为 Error，若大小写敏感，则使用 error 无法检索到该条日志
是否包含中文	日志中包含中文且需要对中文进行检索时可开启该功能 例如日志原文为“用户登录接口超时”，若未开启该功能，搜索“超时”无法检索到该日志，只有完整的搜索“用户登录接口超时”才能检索到该日志，开启该功能后便可通过搜索“超时”检索到该日志

例如，一条完整的日志如下所示：

```
10.20.20.10;[2018-07-16 13:12:57];GET /online/sample HTTP/1.1;200
```

若使用 分隔符模式 提取日志字段，则上传到日志服务的结构化日志为：

```
IP: 10.20.20.10
request: GET /online/sample HTTP/1.1
status: 200
time: [2018-07-16 13:12:57]
```

若全文分词符为 @&()='",;:<>[]{}/ \n\t\r（包含空格），则原始日志的所有字段值会切分为如下关键词（每行表示一个关键词）：

```
10.20.20.10
GET
online
sample
HTTP
1.1
200
2018-07-16
13
12
57
```

在上述的索引配置下，使用以下的检索条件，获得的结果如下：

- 检索条件 A：

  ```
  \/online\/login
  ```

  - 其中\用于转义/符号（该符号为检索语法保留符号，因此需要转义）。
  - 转义后的/符号是分词符，因此实际的检索条件为 online OR login，日志中只要包含 online 或 login，即符合检索条件。
  - 上述示例日志**符合**该检索条件。
- 检索条件 B：

  ```
  "/online/login"
  ```

  - 由于双引号的存在，/符号无需再进行转义。
  - 双引号内的内容同样会分为两个词，但双引号表示日志需**同时存在**这两个词，且两个词**顺序严格一致**才符合检索条件。
  - 上述示例日志不包含login，**不符合**该检索条件。
- 检索条件 C：

  ```
  "/online/sample"
  ```

  - 上述示例日志同时包含online和sample，且顺序与检索条件一致，**符合**该检索条件。

键值索引

键值索引将原始日志按字段（即 key:value）分别切分为多个分词进行索引构建，检索时基于键值方式进行检索（即键值检索）。进行键值检索的时候，必须指定字段名，语法格式为 key:value，例如 status:200 。若不指定字段名，则会当成全文检索处理。

为满足最基本的日志检索要求，日志服务针对部分内置保留字段自动创建键值索引，但不会产生索引流量，不会增加费用，具体如下：

内置保留字段	说明
__FILENAME__	日志采集的文件名，可以利用该字段检索特定文件下的日志，例如__FILENAME__:"/var/log/access.log" 检索 /var/log/access.log 文件的日志
__SOURCE__	日志采集的源 IP，可以利用该字段检索特定机器下的日志，例如 __SOURCE__:192.168.10.10 检索192.168.10.10 机器的日志
__TIMESTAMP__	日志时间戳（毫秒级别 Unix 时间戳），按时间范围检索日志时，将自动使用该时间对日志进行检索，在控制台显示为“日志时间”
__PKG_LOGID__	日志在 日志组 中的 ID，用于 上下文检索，不建议单独使用

配置项	功能描述	说明
字段名称	结构化日志 中的字段名称	-
数据类型	字段的数据类型，包括text、long、double三种类型，其中text类型支持使用通配符进行模糊检索，long、double 类型支持通过数值范围进行检索	long - 整型（Int 64） double - 浮点型（64 bit） text - 字符串
分词符	对字段值进行分词的字符集合，仅支持英文符号	控制台默认分词符 @&?|#()='",;:<>[]{}/ \n\t\r
包含中文	字段中包含中文且需要对中文进行检索时可开启该功能。 例如日志原文为“用户登录接口超时”，若未开启该功能，搜索“超时”无法检索到该日志，只有完整的搜索“用户登录接口超时”才能检索到该日志，开启该功能后便可通过搜索“超时”检索到该日志	-
开启统计	启用后可对该字段使用SQL进行统计分析，如 group by ${key}，sum(${key}) 等	属于键值索引功能的一部分，不单独计费
大小写敏感	检索时是否对大小写敏感 例如日志为 level:Error，若大小写敏感，则使用level:error 无法检索到该条日志	-

例如，一条完整的日志如下所示：

```
10.20.20.10;[2018-07-16 13:12:57];GET /online/sample HTTP/1.1;200
```

若使用 分隔符模式 提取日志字段，则上传到日志服务的结构化日志为：

```
IP: 10.20.20.10
request: GET /online/sample HTTP/1.1
status: 200
time: [2018-07-16 13:12:57]
```

若键值索引的配置如下：

字段名称	字段类型	分词符	包含中文	开启统计
IP	text	@&()='",;:<>[]{}/ \n\t\r	否	是
request	text	@&()='",;:<>[]{}/ \n\t\r	否	是
status	long	无	否	是
time	text	@&()='",;:<>[]{}/ \n\t\r	否	是

在上述的索引配置下，使用以下的检索条件，获得的结果如下：

- 检索条件 A：

```
request:\/online\/login
```

  - 其中\用于转义/符号（该符号为检索语法保留符号，因此需要转义）。
  - 转义后的/符号是分词符，因此实际的检索条件为 online OR login，日志中只要包含 online 或 login，即符合检索条件。
  - 上述示例日志符合该检索条件。
- 检索条件 B：

```
request:"/online/login"
```

  - 由于双引号的存在，/符号无需再进行转义。
  - 双引号内的内容同样会分为两个词，但双引号表示日志需同时存在这两个词，且两个词顺序严格一致才符合检索条件。
  - 上述示例日志不包含login，不符合该检索条件。

- 检索条件 C：

```
request:"/online/sample"
```

  - 上述示例日志同时包含online和sample，且顺序与检索条件一致，符合该检索条件。

此外，针对开启了统计的字段，还可以使用 SQL 对日志进行统计分析：

- 检索分析语句 A：

```
request:"/online/login" | select count(*) as logCounts
```

  统计 request 为 "/online/login" 的日志条数。
- 检索分析语句 B：

```
* | select count(*) as logCounts,request group by request order by count(*) desc limit 10
```

  统计日志条数最大的10个 request。

元数据索引

上传日志到日志服务时，元数据通过 LogTag 字段传递（详情可参见 上传结构化日志 中的 LogTag 字段 ），而原始日志内容通过 Log 字段传递。所有通过 LogTag 传递的数据，配置索引时，均需配置元数据索引。元数据索引其实也归属键值索引，索引规则以及配置方式与键值索引相同，唯一差别在于，元数据字段有特定前缀 __TAG__. 进行标识，例如配置 region 元数据字段的索引为 __TAG__.region 。

例如一条完整的日志如下所示：

```
10.20.20.10;[2018-07-16 13:12:57];GET /online/sample HTTP/1.1;200
```

若使用 分隔符模式 提取日志字段，且携带元数据 region:ap-beijing，则上传到日志服务的结构化日志为：

```
IP: 10.20.20.10
request: GET /online/sample HTTP/1.1
status: 200
time: [2018-07-16 13:12:57]
__TAG__.region:ap-beijing
```

其中，元数据的索引配置规则：

字段名称	分词符
__TAG__.region	@&()='",;:<>[]{}/ \n\t\r

检索示例：输入 __TAG__.region:"ap-beijing" ，可以检索到该示例日志。

高级设置

为满足部分特殊使用场景，索引配置提供如下高级设置。实际使用过程中，建议您采用推荐配置。通过控制台新建索引配置时，也会默认采用推荐配置。

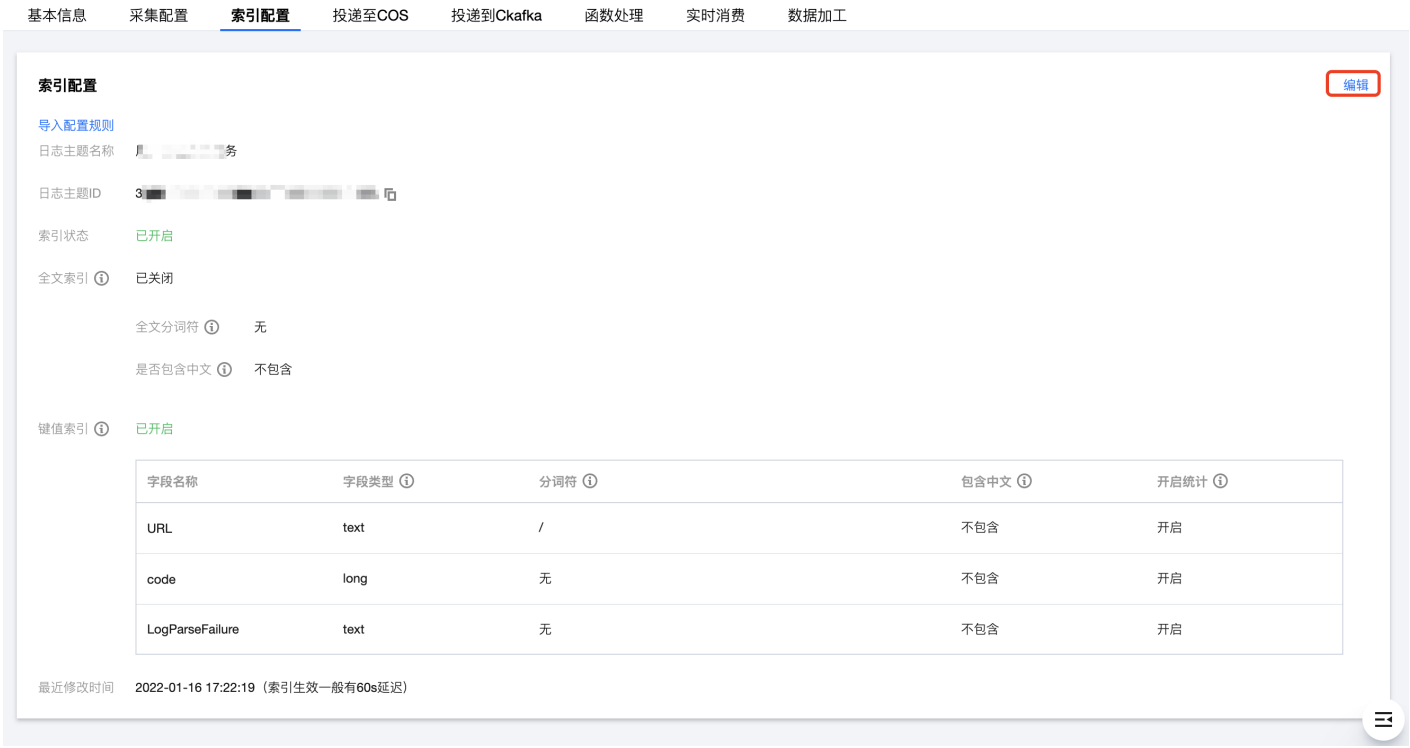
配置项	含义	推荐配置
内置保留字段包含至全文索引	**包含**：全文索引包含__FILENAME__、__HOSTNAME__及__SOURCE__这三个内置字段，可直接使用全文检索的方式检索相关日志字段，例如"/var/log/access.log" **不包含**：全文索引不包含任何内置字段，只能使用键值检索的方式检索相关日志字段，例如__FILENAME__:"/var/log/access.log"	包含

配置项	含义	推荐配置
元数据字段包含至全文索引	**包含**：全文索引包含元数据字段（前缀为__TAG__的字段），可直接使用全文检索的方式检索相关日志字段，例如ap-beijing **不包含**：全文索引不包含任何元数据字段，只能使用键值检索的方式检索相关日志字段，例如__TAG__.region:ap-beijing。低频日志主题不支持键值检索，此时将无法检索这些字段 **仅包含开启键值索引元数据字段**：全文索引包含开启了键值索引的元数据字段，不包含未开启键值索引的元数据字段。低频日志主题无该选项	包含
日志创建索引异常存储	日志创建索引过程中，如果存在日志原文格式异常或索引配置与日志原文不匹配，可能会导致索引创建失败，导致日志无法被检索和查看，此时可将日志的异常部分存储到指定字段（默认为 RAWLOG）中，以避免日志丢失。 详细处理规则参见 创建索引异常日志处理规则	启用

操作步骤

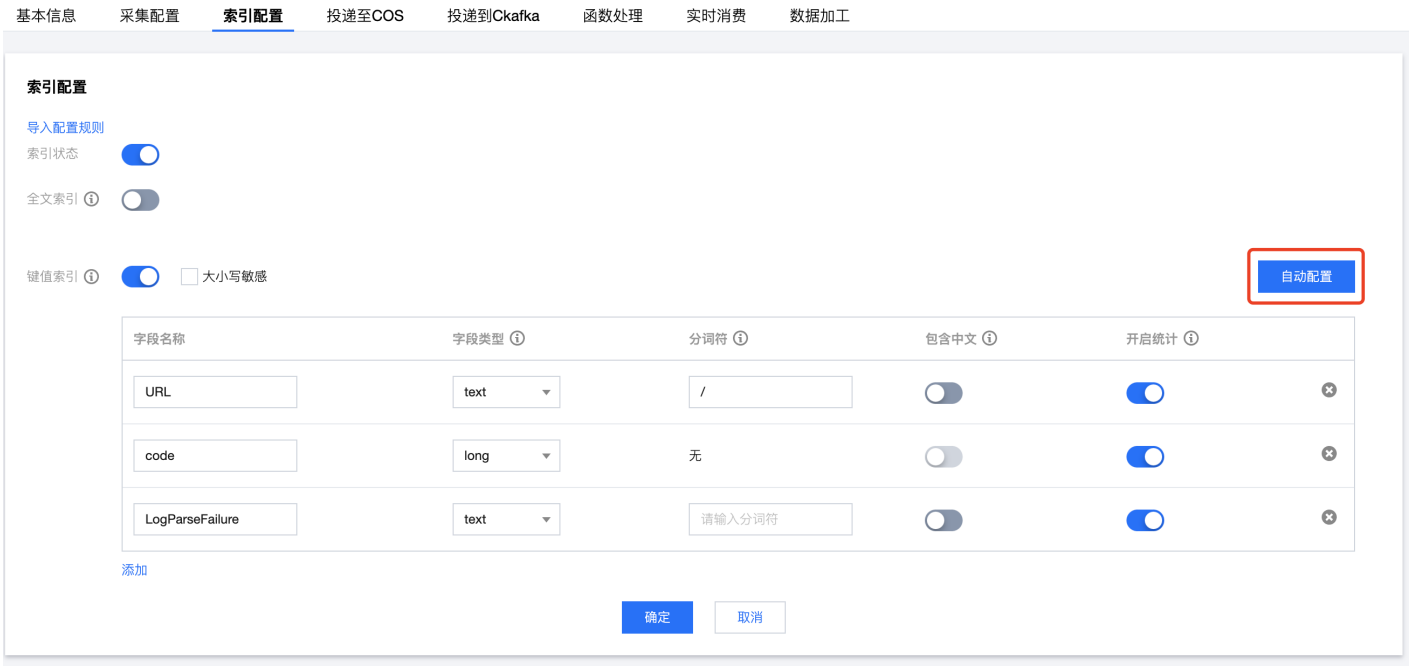
编辑索引配置

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**，进入日志主题列表页面。
3. 单击需要配置索引的日志主题ID/名称，进入日志主题管理页面。
4. 选择**索引配置**页签， 单击**编辑**，进入编辑索引配置页面。



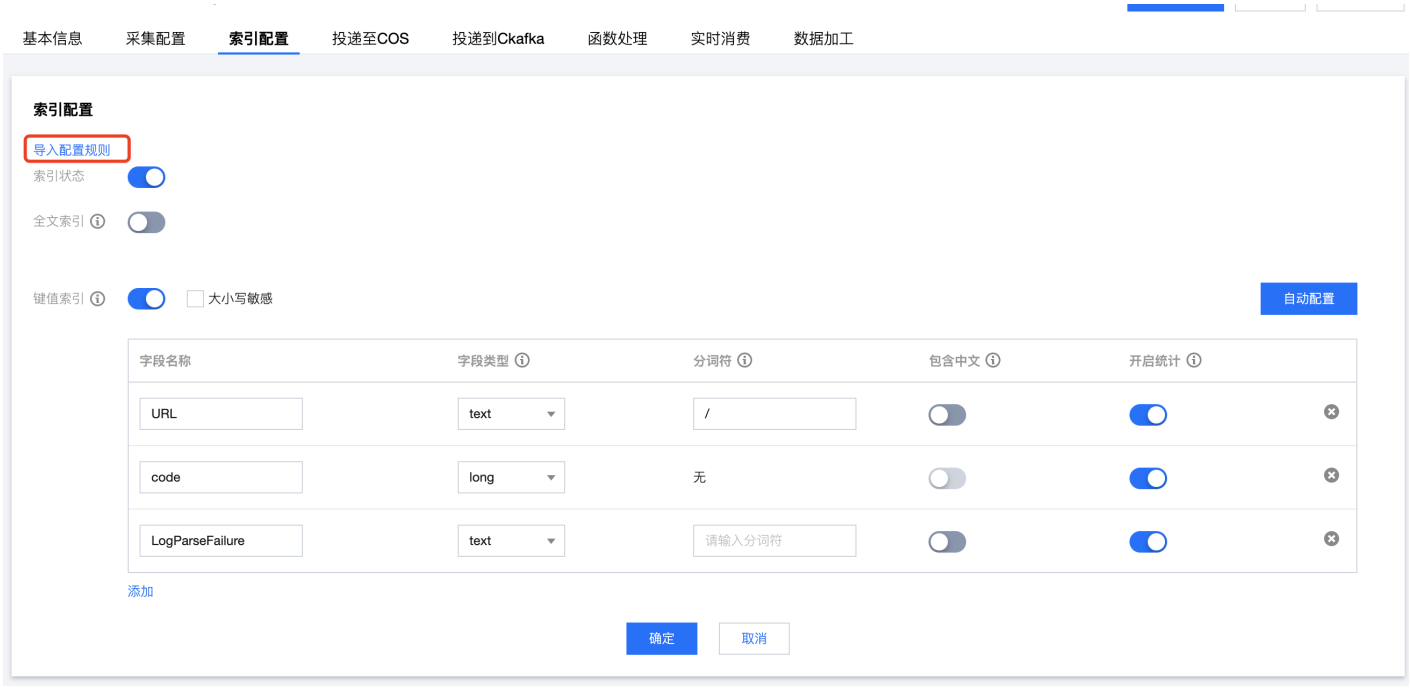
5. 根据实际需求，编辑索引配置，单击**确定**即可保存该索引配置。
   编辑索引配置过程中，您还可以单击**自动配置**，系统将自动获取采集到的最近1条日志作为样例，并将其中的字段解析为键值索引。您可以在自动配置的基础上进行微调，

快速获取最终的索引配置信息。



导入索引配置

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**，进入日志主题列表页面。
3. 单击需要配置索引的日志主题ID/名称，进入日志主题管理页面。
4. 选择**索引配置**页签，单击**导入配置规则**。



5. 在对话框中，选择需要导入的日志主题索引配置，单击**确定**，将选中的日志主题的索引配置填写至当前日志主题的索引配置中。

6. 确认无误后，单击**确定**，即可保存当前日志主题索引配置。

附录

JSON 字段解析规则

在存储过程中，为了避免日志中的 JSON 字段层级过多、对象过多或对象类型不一致，导致日志存储失败，CLS 将仅解析多层 JSON 字段至索引所配置的层级（未修改过索引配置的存量日志主题不受影响），其他层级会以字符串形式存储和展示。该功能升级仅影响原始日志检索结果（即检索语句中仅包含检索条件，不包含 SQL 语句），不影响统计分析结果（即 SQL 结果）。

下面以实际日志为例，详细介绍解析规则。

样例日志：

共三个字段，kye1 为普通字段，kye2 和 kye3 为多层 JSON 字段。

```
{
"kye1": "http://www.example.com",
"kye2": {
"address": {
"country": "中国",
"city": {
"name": "北京",
"code": "065001"
}
},
"contact": {
"phone": {
```

```
"home": "188xxxxxxxx",
"work": "187xxxxxxxx"
},
"email": "xxx@xxx.com"
}
},
"kye3": {
"address": {
"country": "中国",
"city": {
"name": "北京",
"code": "065001"
}
},
"contact": {
"phone": {
"home": "188xxxxxxxx",
"work": "187xxxxxxxx"
},
"email": "xxx@xxx.com"
}
}
}
```


索引配置

键值索引配置如下图，键值索引配置字段为 kye1 和 kye2.address，kye3 未配置键值索引。

键值索引 ⓘ 已开启

字段名称	字段类型 ⓘ	分词符 ⓘ	包含中文 ⓘ	开启统计 ⓘ
kye1	text	@&()='",;:<>[]{}/ \n\t\r\	不包含	开启
kye2.address	text	@&()='",;:<>[]{}/ \n\t\r\	不包含	开启

控制台检索分析效果

JSON 代码：

Table JSON


```
{
  "kye1" : "http://www.example.com"
  "kye2" : {
    "address" : "{"country":"中国","city":{"name":"北京","code":"065001"}}"
    "contact" : "{"phone":{"home":"188xxxxxxxx","work":"187xxxxxxxx"},"email":"xxx@xxx.com"}"
  }
  "kye3" : "{"address":{"country":"中国","city":{"name":"北京","code":"065001"}},"contact":{"phone":{"home":"188xxxxxxxx","work":"187xxxxxxxx"},"email":"xxx@xxx.com"}}"
  "__SOURCE__" : "172.17.0.3"
  "__FILENAME__" : "/root/testLog/jsonParse.log"
  "__PKG_LOGID__" : 65536
}
```

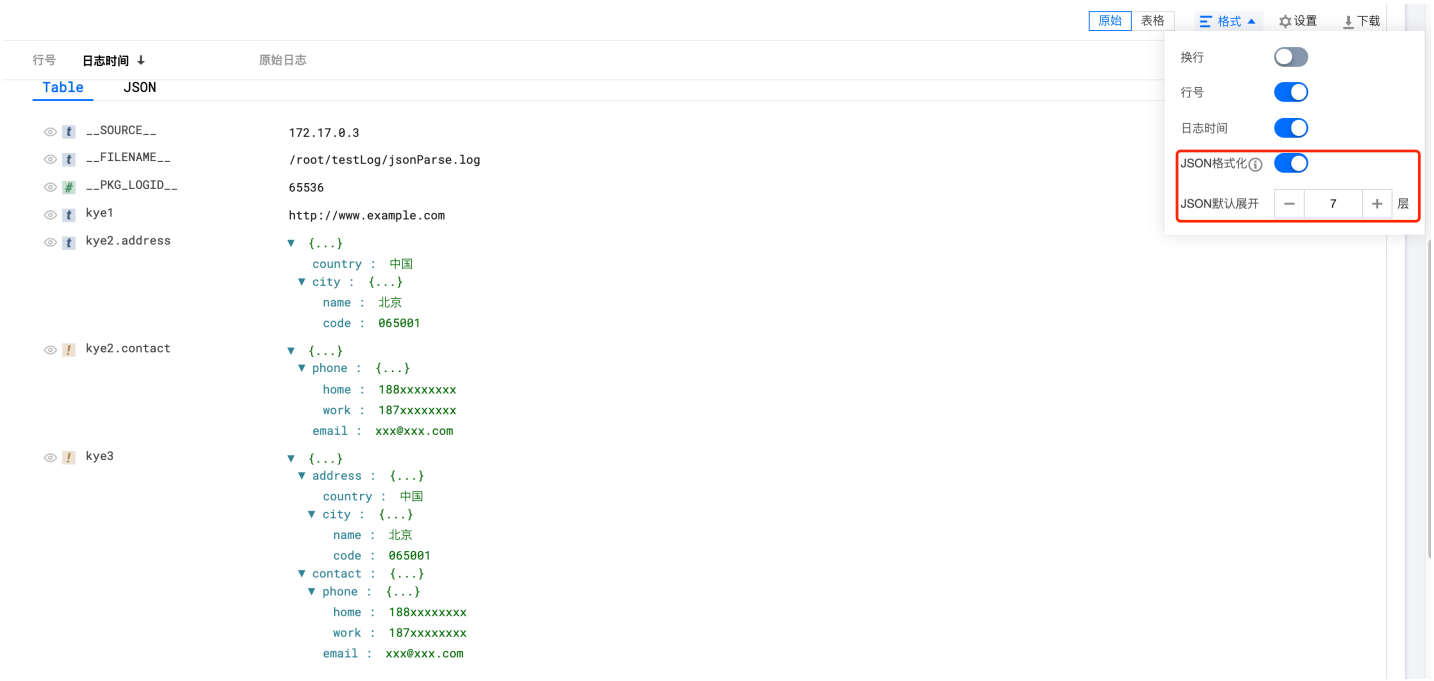
Table 样式：

Table JSON

__SOURCE__	172.17.0.3
__FILENAME__	/root/testLog/jsonParse.log
__PKG_LOGID__	65536
kye1	http://www.example.com
kye2.address	{"country":"中国","city":{"name":"北京","code":"065001"}}
kye2.contact	{"phone":{"home":"188xxxxxxxx","work":"187xxxxxxxx"},"email":"xxx@xxx.com"}
kye3	{"address":{"country":"中国","city":{"name":"北京","code":"065001"}},"contact":{"phone":{"home":"188xxxxxxxx","work":"187xxxxxxxx"},"email":"xxx@xxx.com"}}

- kye2.address 整体显示为字符串，没有进一步展开其下的属性和对象。
- kye2.contact 虽然没有配置键值索引，但由于索引配置到了 kye2.address，kye2.contact 作为同层级对象，同样整体显示为字符串。
- kye3 未配置键值索引，所以没有展开任何属性和对象。

控制台展示日志时提供了 JSON 格式化功能，可将字符串形式的 JSON 字段按层级显示，但该功能仅为控制台显示效果，实际的字段仍为字符串，这一点可在下文的 API 检索分析结果中体现出来。



API 检索分析结果效果

使用 检索分析□志 API 时，输出参数中的 Results 参数值如下（其他参数不受影响，无变化）：


```
{
"Time": 1645065742008,
"TopicId": "f813385f-aee0-4238-xxxx-c99b39aabe78",
"TopicName": "TestJsonParse",
"Source": "172.17.0.2",
"FileName": "/root/testLog/jsonParse.log",
"PkgId": "5CB847DA620DB3D4-10D",
"PkgLogId": "65536",
"HighLights": [],
"Logs": null,
"LogJson": "{\"kye1\":\"http://www.example.com\",\"kye2\":{\"address\":\"{\\\"country\\\":\\\"中国\\\",\\\"city\\\":{\\\"name\\\":\\\"北京\\\",\\\"code\\\":\\\"065001\\\"}}\",\"contact\":\"{\\\"phone\\\":{\\\"home\\\":\\\"188xxxxxxxx\\\",\\\"work\\\":\\\"187xxxxxxxx\\\"},\\\"email\\\":\\\"xxx@xxx.com\\\"}\"},\"kye3\":\"{\\\"address\\\":{\\\"country\\\":\\\"中国\\\",\\\"city\\\":{\\\"name\\\":\\\"北京\\\",\\\"code\\\":\\\"065001\\\"}},\\\"contact\\\":{\\\"phone\\\":{\\\"
```

```
home\\\":\\\"188xxxxxxxx\\\",\\\"work\\\":\\\"187xxxxxxxx\\\"},\\\"email\\\":\\\"xxx@xxx.com\\\"}}\"}"
}
```

- kye2.address 为字符串，因此其值被转义为字符串。
- kye2.contact 作为 kye2.address 的同层级对象，虽然未配置键值索引，其值同样被转义为字符串。
- kye3 未配置键值索引，所以整体被转义为字符串。

如果您在使用代码处理 检索分析□志 的 API 输出参数，请注意此处的转义符，避免处理逻辑出现异常。为方便您对比功能升级前后的 API 输出参数变化细节，升级前的 API 输出参数如下：

```
{
"Time": 1645065742008,
"TopicId": "f813385f-aee0-4238-xxxx-c99b39aabe78",
"TopicName": "zhengxinTestJsonParse",
"Source": "172.17.0.2",
"FileName": "/root/testLog/jsonParse.log",
"PkgId": "25D0A12F620DBB64-D3",
"PkgLogId": "65536",
"HighLights": [],
"Logs": null,
"LogJson": "{\"kye1\":\"http://www.example.com\",\"kye2\":{\"address\":\"{\\\"city\\\":{\\\"code\\\":\\\"065001\\\",\\\"name\\\":\\\"北京\\\"},\\\"country\\\":\\\"中国\\\"}\",\"contact\":{\"phone\":{\"work\":\"187xxxxxxxx\",\"home\":\"188xxxxxxxx\"},\"email\":\"xxx@xxx.com\"}},\"kye3\":{\"address\":{\"country\":\"中国\",\"city\":{\"code\":\"065001\",\"name\":\"北京\"}},\"contact\":{\"phone\":{\"work\":\"187xxxxxxxx\",\"home\":\"188xxxxxxxx\"},\"email\":\"xxx@xxx.com\"}}}"
}
```

创建索引异常日志处理规则

日志创建索引过程中，如果日志原文格式异常或索引配置与日志原文不匹配，可能会导致索引创建失败，无法被检索和查看这些日志。此时，可将日志的异常部分存储到指定字段（默认为 RAWLOG）中，以避免日志丢失。

下面以实际日志为例，详细介绍处理规则。

会导致创建索引异常的日志

- **样例日志1：**
  如下日志，city 对象下的 name 及 code 未使用双引号包裹。

```
{
"kye1": "http://www.example.com",
"kye2": {
"address": {
"country": "中国",
"city": {
name: "北京", //格式错误，缺失双引号
code: "065001" //格式错误，缺失双引号
}
}
}
}
```

- **样例日志2：**
  如下日志，索引配置中添加了kye2.address.city.name字段，但实际上 city 为字符串，并没有子级对象。

```
{
"kye1": "http://www.example.com",
"kye2": {
"address": {
"country": "中国",
```

```
"city": "北京" //city为字符串，并没有子级对象name
}
}
}
```

- **样例日志3：**
  如下日志，索引配置中添加了kye2.address.city字段，但实际上 address 为数组，并没有 city 这一子级对象，CLS 不支持为数组内的对象创建索引。

```
{
"kye1": "http://www.example.com",
"kye2": {
"address": [{ //address为数组，并没有city这一子级对象
"country": "中国",
"city": "北京"
}, {
"country": "中国",
"city": "上海"
}]
}
}
```

处理规则

在索引配置的高级设置中启用日志创建索引异常存储功能后，CLS 若遇到上述异常日志，将尽可能的根据索引规则解析日志并创建索引，同时将日志中的错误部分存储到指定的字段（默认为 RAWLOG）。
例如如下日志，索引配置中包含 kye1 和 kye2.address.city 两个字段。

```
{
"kye1": "http://www.example.com",
"kye2": {
"address": [{ //address为数组，并没有city这一子级对象
"country": "中国",
"city": "北京"
}, {
"country": "中国",
"city": "上海"
}]
}
}
```

kye1字段可正常解析并创建索引，kye2字段由于和索引配置不一致，无法正常解析，则该字段将会转为字符串并存储至 RAWLOG 字段下。可在控制台看到如下日志：

```
kye1：http://www.example.com
RAWLOG：{"kye2":"{\"address\":[{\"country\":\"中国\",\"city\":\"北京\"},{\"country\":\"中国\",\"city\":\"上海\"}]}"}
```

可直接使用全文检索的方式检索该日志（例如"中国"），也可以为 RAWLOG 添加键值索引，使用键值索引的方式检索该日志（例如RAWLOG:"中国"）。通过RAWLOG:* 还可以检索到所有存在创建索引异常的日志，帮助您合理的调整日志输出格式及索引配置。

重建索引

最近更新时间：2022-06-10 16:14:29

概述

由于索引规则编辑后仅对新写入的日志生效，如果需要按照最新的索引规则对历史数据重新创建索引，可使用重建索引功能，常见的需要使用重建索引的场景如下：

- 键值索引新增字段，历史数据也需要使用该字段进行检索。
- 调整分词符配置，历史数据也需要按照新的分词符进行检索。
- 字段之前未开启统计，在索引配置中开启后，历史数据也需要对该字段进行统计（即 SQL）。

重建索引会将指定时间范围内的原始日志重新构建一遍索引，将产生索引流量费用（不产生写流量费用及索引存储费用）。数据量较大时会消耗较长时间并产生较高的费用，建议您尽量避免频繁的修改索引配置并重建索引。

前提条件

已开启索引，且最新的索引配置能满足后续的检索分析需求。

注意事项

- 单个日志主题同时仅允许运行一个重建索引任务，单个日志主题最多同时拥有10个重建索引任务记录，需删除不再需要的任务记录后才能新建索引任务。
- 同一时间范围内的日志，仅允许重建一次索引，需删除之前的任务记录后才能再次重建。
- 删除重建索引任务记录将恢复重建索引前的索引数据。
- 仅支持通过控制台重建写流量500GB以内的时间范围的日志。超过该限制时建议您缩小需要重建索引的日志时间范围，或 联系技术服务 获得更高配额支持。
- 重建索引时间范围以日志时间为准，日志上传时间与重建索引时间范围有超过1小时的偏差时（例如16:00上传了一条02:00的日志到 CLS，重建00:00~12:00的日志索引）不会被重建且后续无法进行检索。新上报一条日志到已经被重建的日志时间范围时，也不会被重建且后续无法进行检索。

操作步骤

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**，进入日志主题列表页面。
3. 单击需要重建索引的日志主题ID/名称，进入日志主题管理页面。

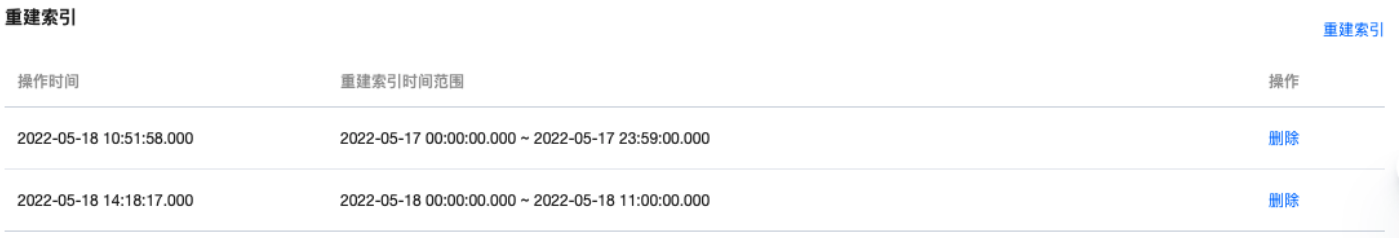
4. 选择**索引配置**页签，单击底部的**重建索引**。

5. 选择需要重建的日志时间范围，单击**开始重建**。

> **说明：**
> 此处将根据选择的时间范围预估需要重建的日志数据对应的写流量及耗时，如果写流量过大、耗时过长，建议您尽可能的缩小需要重建的日志时间范围，以缩短耗时，同时也有助于降低重建索引带来的成本。

开始重建后，可以查看任务的运行进度。

6. 重建完成后，可在列表中查看已经完成的任务。在列表中可删除指定任务，删除任务将恢复重建索引前的索引数据。



SQL 语法
SELECT 语法

最近更新时间：2022-03-17 15:55:00

用于从表中选取数据，默认从当前日志主题中获取符合检索条件的数据。

语法格式

```
* | SELECT [列名(KEY)]
```

语法示例

从日志数据中选取列（KEY）为 remote_addr 以及 method 的值：

```
* | SELECT remote_addr, method
```

从日志数据中选取所有列（KEY）：

```
* | SELECT *
```

SELECT 后面也可以跟算术表达式，如从日志数据中查询下载速度：

下载速度（ speed ）= 总发送字节数（ body_bytes_sent ）/ 请求时长（ request_time ）

```
* | SELECT body_bytes_sent / request_time AS speed
```

AS 语法

最近更新时间：2022-01-19 11:45:35

AS 子句用于为列名称（KEY）指定别名。

语法格式

```
* | SELECT 列名（KEY） AS 别名
```

语法样例

创建中文别名：

```
* | SELECT remote_addr AS "客户端IP", request_time AS "请求时间(单位：秒)"
```

> **注意：**
> 如果别名中包含中文或其他特殊字符，需要使用双引号。

统计访问次数：

```
* | SELECT COUNT(*) AS PV
```

GROUP BY 语法

最近更新时间：2022-03-10 16:50:01

GROUP BY 语法用于结合 聚合函数，根据一个或多个列对分析结果进行分组统计。

语法格式

```
* | SELECT 列名, 聚合函数 GROUP BY [ 列名 | 别名 | 序号 ]
```

> **注意：**
> 在 SQL 语句中，如果您使用了 GROUP BY 语法，则在执行 SELECT 语句时，只能选择 GROUP BY 的列或聚合计算函数，不允许选择非 GROUP BY 的列。例如 * | SELECT status, request_time, COUNT(*) AS PV GROUP BY status 为非法分析语句，因为 request_time 不是 GROUP BY 的列。

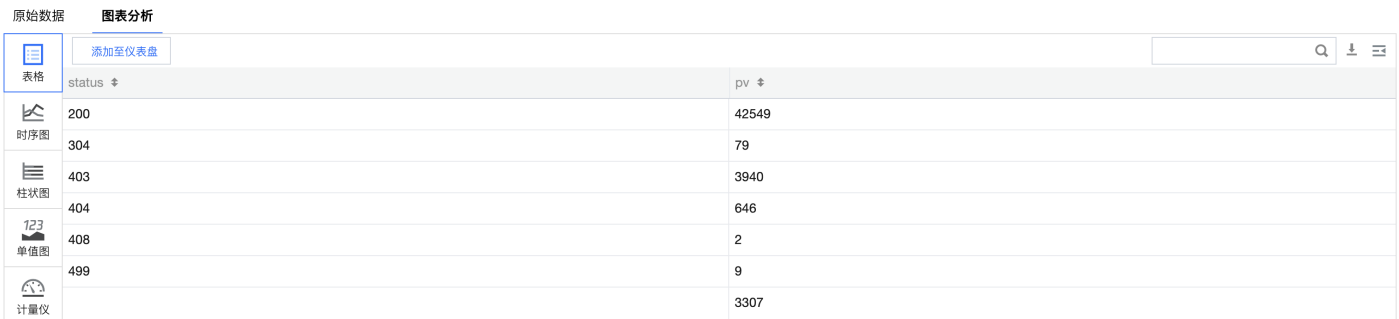
GROUP BY 语法支持按照列名、别名或序号进行分组，详细说明如下表所示：

参数	说明
列名	日志字段名称或聚合函数计算结果列，支持单列或多列
别名	按照日志字段或聚合函数计算结果的别名进行分组
序号	某列在 SELECT 语句中的序号（从1开始）。 例如 status 列的序号为1，所以下面两个语句为等同关系。 • * | SELECT status, count(*) AS PV GROUP BY status • * | SELECT status, count(*) AS PV GROUP BY 1
聚合函数	GROUP BY 语法常与 MIN、MAX、AVG、SUM、COUNT 等聚合函数搭配使用，更多信息请参见 聚合函数。

语法示例

- 统计不同状态码的访问次数：

```
* | SELECT status, count(*) AS pv GROUP BY status
```



- 按照每分钟的时间粒度，计算 PV：

```
* |
SELECT
date_trunc(
'minute',
cast(__TIMESTAMP__ as timestamp)
) AS dt,
count(*) AS pv
GROUP BY
dt
ORDER BY
dt
```

```
limit
10
```

__TIMESTAMP__ 字段为日志服务中的保留字段，表示时间列。**dt** 为 date_trunc('minute', cast(__TIMESTAMP__ as timestamp)) 的别名。date_trunc() 函数的更多信息，请参见 时间截断函数。

原始数据 **图表分析**

添加至仪表盘

表格 时序图 柱状图 单值图 计量仪 饼图

dt	pv
2021-07-18 14:45:00.0	43
2021-07-18 14:46:00.0	643
2021-07-18 14:47:00.0	5006
2021-07-18 14:48:00.0	6862
2021-07-18 14:49:00.0	3987
2021-07-18 14:50:00.0	2471
2021-07-18 14:51:00.0	3594
2021-07-18 14:52:00.0	3780
2021-07-18 14:53:00.0	1911
2021-07-18 14:54:00.0	1611

说明：

- **limit 10表示最多获取10行结果。如果不使用 LIMIT 语法，则默认获取100行结果。**
- **在索引配置中，当您开启任意字段的统计功能后，日志服务会自动开启 __TIMESTAMP__ 字段的统计功能。**

- **按照5分钟的时间粒度统计 PV 及 UV：**
  因为 date_trunc() 函数只能按照固定时间间隔统计，可以使用 histogram 函数进行自定义时间间隔统计

```
* |
SELECT
histogram(
cast(__TIMESTAMP__ as timestamp),
interval 5 minute
) as dt,
count(*) as pv,
count(
distinct(remote_addr)
) as uv
group by
dt
order by
dt
```

原始数据 **图表分析**

添加至仪表盘

表格 时序图 柱状图

dt	pv	uv
2021-07-19 17:15:00.0	3226	93
2021-07-19 17:20:00.0	2607	74
2021-07-19 17:25:00.0	1693	38

ORDER BY 语法

最近更新时间：2022-03-10 16:50:07

ORDER BY 语法用于根据指定的列名对分析结果进行排序。

语法格式

```
ORDER BY 列名 [DESC | ASC]
```

说明：

- 您可以指定多个列名，按照不同的排序方式排序。例如 ORDER BY 列名1[DESC | ASC]，列名2[DESC | ASC]。
- 如果您未配置关键字 DESC 或 ASC，则默认进行升序排列。
- 当排序的目标列中存在相同的值时，每次排序结果可能不同。如果您希望每次序列结果相同，可指定多列进行排序。

参数说明如下表所示：

参数	说明
列名	列名即为日志字段名称或聚合函数计算结果列
DESC	降序排列
ASC	升序排列

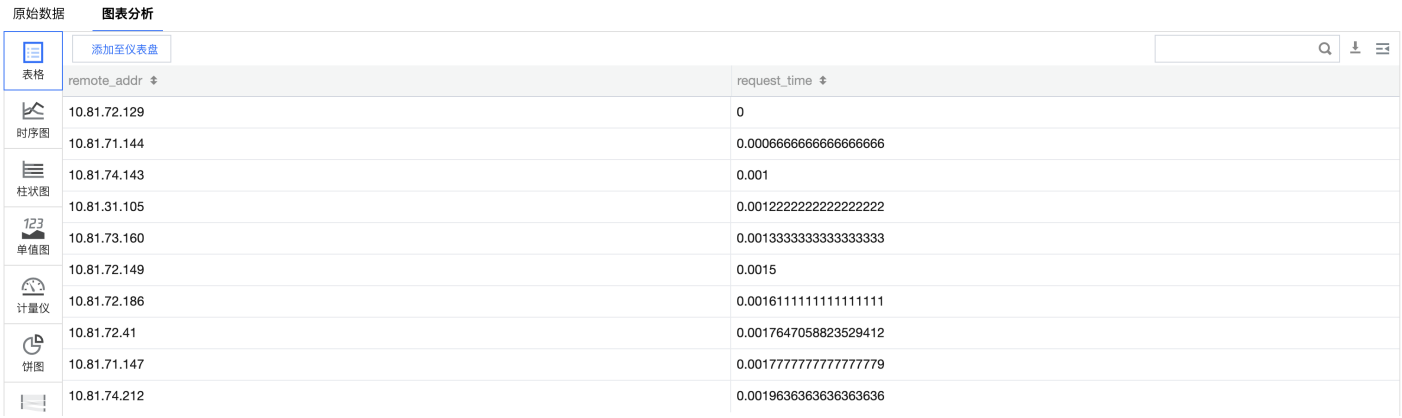
语法示例

- 统计不同请求状态的数量，并按照请求数量降序排列：

```
* |
SELECT
status,
count(*) AS pv
GROUP BY
status
ORDER BY
pv DESC
```

- 计算各服务器的平均请求时间，并按照请求时间进行升序排列：

```
* |
SELECT
remote_addr,
avg(request_time) as request_time
group by
remote_addr
order by
request_time ASC
LIMIT
10
```



remote_addr	request_time
10.81.72.129	0
10.81.71.144	0.0006666666666666666
10.81.74.143	0.001
10.81.31.105	0.0012222222222222222
10.81.73.160	0.0013333333333333333
10.81.72.149	0.0015
10.81.72.186	0.0016111111111111111
10.81.72.41	0.0017647058823529412
10.81.71.147	0.0017777777777777779
10.81.74.212	0.0019636363636363636

LIMIT 语法

最近更新时间：2022-03-10 16:50:18

LIMIT 语法用于限制输出结果的行数。

语法格式

- 读取前 N 行：

```
limit N
```

- 从第 S 行开始读，读取 N 行：

```
offset S limit N
```

语法示例

- 获取10行结果：

```
* | select status, count(*) as pv group by status limit 10
```

- 获取第3行到第42行的结果，共计40行：

```
* | select status, count(*) as pv group by status offset 2 limit 40
```

限制说明

指标	限制说明	备注
SQL 结果条数	SQL 返回结果条数最大10000条	limit 默认100，最大10000

WHERE 语法

最近更新时间：2022-03-10 16:50:43

WHERE 用于提取那些满足指定条件的日志。

语法格式

```
* | SELECT 列名（KEY） WHERE 列名（KEY） 运算符 值
```

运算符可以是 = 、 <> 、 > 、 < 、 >= 、 <= 、 BETWEEN 、 IN 、 LIKE 。

注意：

- SQL 中的过滤功能相比检索条件性能较差，建议尽可能的使用检索条件满足数据过滤需求，例如使用status:>400 | select count(*) as logCounts代替* | select count(*) as logCounts where status>400 以更快的获得统计结果。
- WHERE 中不能使用 AS 别名，例如level:* | select level as log_level where log_level='ERROR'，该语句执行会报错，因为其不符合 SQL-92规范。

语法示例

从日志数据中查询状态码大于400的日志：

```
* | SELECT * WHERE status > 400
```

从日志数据中查询请求方式为 GET 且客户端 IP 为 192.168.10.101 的日志条数：

```
* | SELECT count(*) as count WHERE method='GET' and remote_addr='192.168.10.101'
```

统计 URL 后缀的为 .mp4 的平均请求大小：

```
* | SELECT round(sum(body_bytes_sent) / count(body_bytes_sent), 2) AS avg_size WHERE url like '%.mp4'
```

HAVING 语法

最近更新时间：2022-03-08 15:51:55

HAVING 用于对分组聚合后的数据进行过滤，与 WHERE 的区别在于其作用于分组（GROUP BY）之后，排序（ORDER BY）之前，而 WHERE 作用于聚合前的原始数据。

语法格式

```
* | SELECT 列名, 聚合函数 GROUP BY [ 列名 | 别名 | 序号 ] HAVING 聚合函数 运算符 值
```

运算符可以是 = 、 <> 、 > 、 < 、 >= 、 <= 、 BETWEEN 、 IN 、 LIKE 。

语法示例

统计平均响应耗时大于1000ms的 URL，并按耗时倒排：

```
* |
select
avg(responseTime) as time_avg,
URL
group by
URL
having
avg(responseTime)> 1000
order by
avg(responseTime) desc
limit
10000
```

由于过滤条件为各个 URL 的平均响应耗时，属于聚合后的结果，因此不能使用 WHERE 进行数据过滤。

嵌套子查询

最近更新时间：2022-03-07 09:05:53

本文介绍嵌套子查询的语法使用与操作示例。

语法格式

针对一些复杂的统计分析场景，需要先对原始数据进行一次统计分析，再针对该分析结果进行二次统计分析，这时候需要在一个 SELECT 语句中嵌套另一个 SELECT 语句，这种查询方式称为嵌套子查询。

```
* | SELECT key FROM (subquery)
```

- subquery：子查询，需被包裹在括号中。
- key：需要从子查询中获取哪些字段进行二次统计分析。
- 支持多层嵌套，不局限于两层。

语法示例

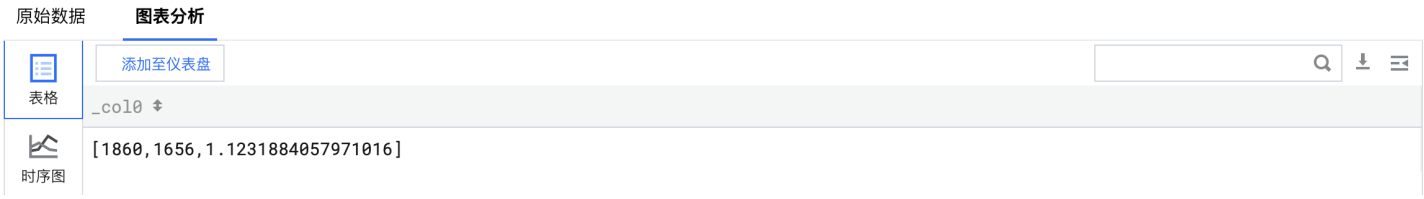
计算当前1小时和昨天同时段的网站访问量比值：
选择查询和分析的时间范围为近1小时，并执行如下查询和分析语句，其中86400表示当前时间减去86400秒（1天）。

查询和分析语句

```
* | SELECT compare(PV, 86400) FROM (SELECT count(*) AS PV)
```

- SELECT count(*) AS PV为第一层统计分析，针对原始日志统计获得网站访问量 PV。
- SELECT compare(PV, 86400) FROM为第二层统计分析，针对第一层统计分析的结果，对 PV 进行二次统计，使用 同环比函数compare 获得1天前的网站访问量 PV。

查询和分析结果



- 1860表示当前1小时的网站访问量。
- 1656表示昨天同时段的网站访问量。
- 1.1231884057971016表示当前1小时与昨天同时段的网站访问量比值。

如需查询和分析结果为分列显示，则可进一步嵌套查询：

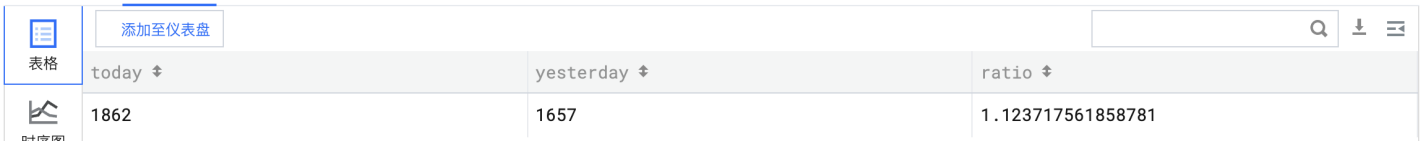
查询和分析语句

```
* |
SELECT compare[1] AS today, compare[2] AS yesterday, compare[3] AS ratio
FROM (
SELECT compare(PV, 86400) AS compare
FROM (
SELECT COUNT(*) AS PV
)
)
```

SELECT compare[1] AS today, compare[2] AS yesterday, compare[3] AS ratio FROM 针对 compare 函数的结果，通过数组下标获取其中特定位置的数值。

查询和分析结果

原始数据 **图表分析**



SQL 函数

字符串函数

最近更新时间：2022-07-01 09:25:55

本文介绍字符串函数的基本语法和示例。

注意：

- 字符串必须使用单引号''包裹，无符号包裹或被双引号""包裹的字符表示字段或列名。例如'status'表示字符串 status，status或"status"表示日志字段 status。
- 字符串内本身包含单引号'时，需使用''（两个单引号）代表单引号本身。例如'{''version'': ''1.0''}'表示原始字符串{'version': '1.0'}。字符串内本身包含双引号"时无需特殊处理。
- 如下函数中的 key 参数均表示日志字段名称。

函数名称	说明	示例
chr(number)	返回与输入参数指定的 ASCII 码位值匹配的字符。返回结果为 varchar 类型。	返回第77位 ASCII 码中匹配的字符值。 * | SELECT chr(77)
codepoint(string)	把 ASCII 码形式的字段值转换为 BigInt 类型。返回结果为 integer 类型。	将 ASCII 码中的字符值转换为对应的位置。 * | SELECT codepoint('M')
concat(key1, ..., keyN)	连接字符串 key1，key2，...keyN，与||连接符连接效果一致。返回结果为 varchar 类型。	将多个字符串拼接成一个字符串。 * | SELECT concat(remote_addr, host, time_local)
concat_ws(split_string,key0, ..., keyN)	以 split_string 作为分隔符连接字符串 key1，key2，...keyN。split_string可以为字符串或变量，如果 split_string 为 null，则结果为 null，且会跳过 key1，key2，...keyN 中的 null 值。返回结果为 varchar 类型。	以/作为分隔符进行字符串之间的连接。 * | SELECT concat_ws('/', remote_addr,host,time_local)
concat_ws(split_string, array(varchar))	以 split_string 作为分隔符将数组内的元素连接为一个字符串。如果 split_string 为 null，则结果为 null，且会跳过数组中的 null 值。返回结果为 varchar 类型。 注意：此函数中 array(varchar)参数为一个数组，不是字符串。	以#作为分割符进行字符串之间的连接，此例中split函数的输出为array * | select concat_ws('#',split('cloud.tencent.com/product/cls', '/'))
format(format，args...)	使用格式 format 对参数 args 进行格式化输出。返回结果为 varchar 类型。	使用 format 格式 ‘IP 地址: %s, 域名: %s’ 对参数 remote_addr 和参数 host 进行格式化输出。 * | SELECT format('IP地址: %s, 域名: %s', remote_addr, host)
hamming_distance(key1, key2)	返回 key1 字符串与 key2 字符串的汉明距离。注意两个字符串长度必须一致。返回结果为 bigint 类型。	返回 remote_addr 字符串与 remote_addr 字符串的汉明距离。 * | SELECT hamming_distance(remote_addr, remote_addr)
length(key)	计算字符串的长度。返回结果为 bigint 类型。	返回 http_user_agent 的字符串长度。 * | SELECT length(http_user_agent)
levenshtein_distance(key1, key2)	返回字符串 key1 和 key2 的编辑距离。返回结果为 bigint 类型。	返回 remote_addr 字符串和 http_protocol 字符串的编辑距离。 * | SELECT levenshtein_distance(remote_addr, http_protocol)
lower(key)	将字符串转换为小写形式。返回结果为 varchar 类型，小写形式。	将 http_protocol 字符串转换为小写形式。 * | SELECT lower(http_protocol)
lpad(key, size, padstring)	使用 padstring 对字符串进行向左补全到 size 大小。如果 size 小于 key 的长度，则对字符串进行截断到 size 大小。size 必须为非负数，padstring 必须非空。返回结果为 varchar 类型。	使用 ‘0’ 对 remote_addr 字符串进行向左补全至32字符。 * | SELECT lpad(remote_addr, 32, '0')
ltrim(key)	移除字符串左侧的空白字符。返回结果为 varchar 类型。	移除 http_user_agent 字符串左侧的空白字符。 * | SELECT ltrim(http_user_agent)

函数名称	说明	示例
position(substring IN key)	返回 substring 在字符串中的位置，从1开始。如果没找到，则返回0。该函数有特殊语法 IN 作为参数。另外参考 strpos()。返回结果为 bigint 类型。	返回 'G' 字符在 http_method 中的位置。 * | select position('G' IN http_method)
replace(key, substring)	从字符串 key 中移除所有 substring 字符串。返回结果为 varchar 类型。	从 time_local 字符串中移除所有 ‘Oct’。 * | select replace(time_local, 'Oct')
replace(key, substring, replace)	替换字符串中所有 substring 为 replace 的字符串。返回结果为 varchar 类型。	替换 time_local 字符串中所有 ‘Oct’ 为 ‘10’。 * | select replace(time_local,'Oct','10')
reverse(key)	翻转字符串 key。返回结果为 varchar 类型。	翻转 host 字符串。 * | select reverse(host)
rpad(key, size, padstring)	使用 padstring 对字符串进行向右补全到 size 大小。如果 size 小于 key 的长度，则对字符串进行截断到 size 大小。size 必须为非负数，padstring 必须非空。返回结果为 varchar 类型。	使用 ‘0’ 对 remote_addr 字符串进行向右补全到32位大小字符。 * | select rpad(remote_addr, 32, '0')
rtrim(key)	移除字符串右侧所有空白字符。返回结果为 varchar 类型。	移除 http_user_agent 字符串右侧所有空白字符。 * | select rtrim(http_user_agent)
split(key, delimiter)	使用指定的分隔符对字符串进行分割，返回字符串数组。	使用指定的 ‘/’ 分隔符对 http_user_agent 字符串进行分割，返回字符串数组。 * | SELECT split(http_user_agent, '/')
split(key, delimiter, limit)	使用指定的分隔符对字符串进行分割，返回字符串数组，数组长度最多为 limit。数组中最后一个元素始终为 key 中剩余的所有部分。limit 必须为正数。	使用指定的 ‘/’ 分隔符对 http_user_agent 字符串进行分割，返回数组长度为10的字符串数组。 * | SELECT split(http_user_agent, '/', 10)
split_part(key, delimiter, index)	使用指定的分隔符对字符串进行分割，返回数组中 index 位置的字符串。index 从1开始。如果 index 大于数组长度，则返回 null。返回结果为 varchar 类型。	使用指定的 ‘/’ 分隔符对 http_user_agent 字符串进行分割，返回第1位的字符串。 * | SELECT split_part(http_user_agent, '/', 1)
strpos(key, substring)	返回 substring 在字符串中的位置，从1开始。如果没找到，则返回0。返回结果为 bigint 类型。	返回 ‘org’ 字符串在 host 字符串中的位置。 * | SELECT strpos(host, 'org')
strpos(key, substring, instance)	返回字符串经 substring 分割后的第 instance 个实例在 string 中的位置。如果 instance 是负数，则从尾到头数。位置结果从1开始，如果没找到，则返回0。返回结果为 bigint 类型。	返回 host 字符中经过 ‘g’ 分割后的第1个实例在 host 字符串中的位置。 * | SELECT strpos(host, 'g', 1)
substr(key, start)	返回字符串中从 start 位置开始的剩余所有字符，位置从1开始算。如果 start 为负数，则是相对于尾部开始算，例如：[...]返回结果为 varchar 类型。	返回 remote_user 字符串中从第2位开始剩余字符。 * | SELECT substr(remote_user, 2)
substr(key, start, length)	返回字符串从 start 开始的最多 length 长度的子串，位置从1开始算。start 为负数，则相对于尾部开始算。返回结果为 varchar 类型。	返回 remote_user 字符串中第2至第5位字符。 * | SELECT substr(remote_user, 2, 5)
translate(key, from, to)	将 key 中出现在 from 中的字符全部替换为 to 中对应位置的字符。如果 from 包含重复字符，则只算第一个字符。如果 source 中没出现 from 中的字符，则 source 直接复制。如果 from 长度超过 to 的长度，则对应字符会被删除。返回结果为 varchar 类型。	将 remote 字符串中 ‘123’ 的字符替换为 ‘ABC’ 对应的字符。 * | SELECT translate(remote_user, '123', 'ABC')
trim(key)	删除字符串中前后的空白字符。返回结果为 varchar 类型。	删除 http_cookies 字符串中前后的空白字符。 * | SELECT trim(http_cookies)
upper(key)	将字符串转化为大写字符。返回结果为 varchar 类型，大写形式。	将 host 字符串中的小写字符转换为大写。 * | SELECT upper(host)
word_stem(word)	返回 word 的英语单词。返回结果为 varchar 类型。	返回 Mozila 的单词。 * | SELECT word_stem('Mozilla')
word_stem(word, lang)	返回 word 的 lang 语言词根。返回结果为 varchar 类型。	返回 selects 词根的单词。 * | SELECT word_stem('selects', 'en')

Unicode 函数

函数名称	说明
normalize(string)	转化 string 为 NFC 标准格式。返回结果为 varchar 类型。
normalize(string, form)	转化 string 为 form 格式。该函数 form 参数需要使用关键字（NFD，NFC，NFKD，NFKC），而非字符串。返回结果为 varchar 类型。
to_utf8(string)	将 string 转化为 UTF-8 格式的二进制字符串 varbinary。返回结果为 varchar 类型。
from_utf8(binary)	将二进制字符串转为 UTF-8 格式的字符串。非法的 UTF-8 字符会被替换为 U+FFFD。返回结果为 varchar 类型。
from_utf8(binary, replace)	将二进制字符串转为 UTF-8 格式的字符串。非法的 UTF-8 字符会被替换为 replace。返回结果为 varchar 类型。

示例

基于如下日志样例介绍查询和分析语句示例。

原始日志数据样例：

```
10.135.46.111 - - [05/Oct/2015:21:14:30 +0800] "GET /my/course/1 HTTP/1.1" 127.0.0.1 200 782 9703 "http://127.0.0.1/course/explore?filter%5Btype%5D=all&filter%5Bprice%5D=all&filter%5BcurrentLevelId%5D=all&orderBy=studentNum" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:64.0) Gecko/20100101 Firefox/64.0" 0.354 0.354
```

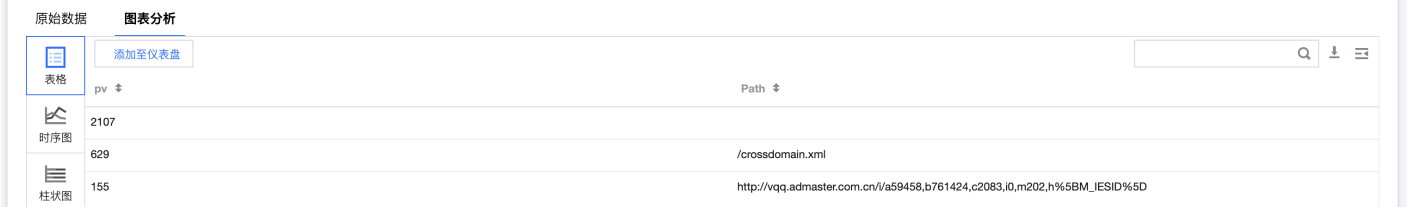
配置单行-完全正则采集模式后，自定义的 key 名称如下所示：

```
body_bytes_sent: 9703
http_host: 127.0.0.1
http_protocol: HTTP/1.1
http_referer: http://127.0.0.1/course/explore?filter%5Btype%5D=all&filter%5Bprice%5D=all&filter%5BcurrentLevelId%5D=all&orderBy=studentNum
http_user_agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:64.0) Gecko/20100101 Firefox/64.0
remote_addr: 10.135.46.111
request_length: 782
request_method: GET
request_time: 0.354
request_url: /my/course/1
status: 200
time_local: [05/Oct/2015:21:14:30 +0800]
upstream_response_time: 0.354
```

分析语句示例：

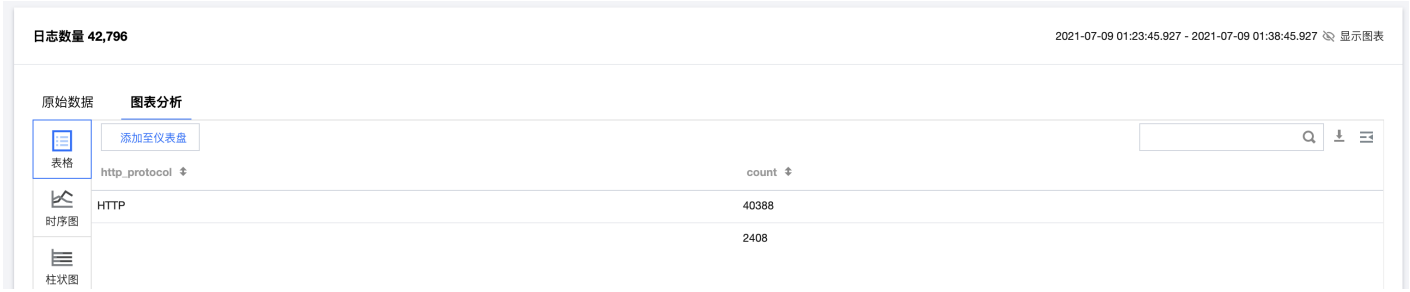
- **使用问号（？）拆分 request_url 字段的值并返回第一个字符串（即文件路径部分），然后统计不同路径对应的请求数量。**

```
* | SELECT count(*) AS pv, split_part(request_url, '?', 1) AS Path GROUP BY Path ORDER BY pv DESC LIMIT 3
```



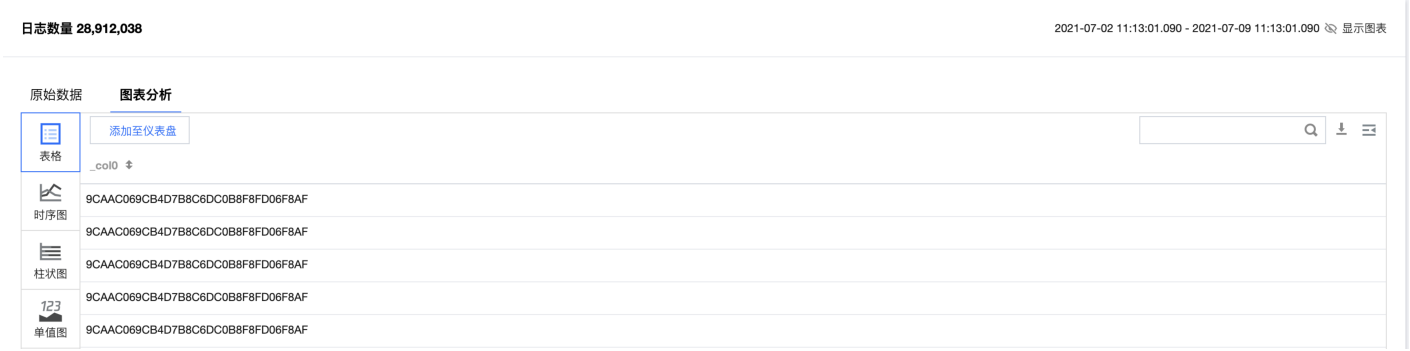
- **提取字段值 http_protocol 中的前4个字符（即HTTP部分），然后统计 HTTP 协议对应的请求数量。**

```
* | SELECT substr(http_protocol,1,4) AS http_protocol, count(*) AS count group by http_protocol
```



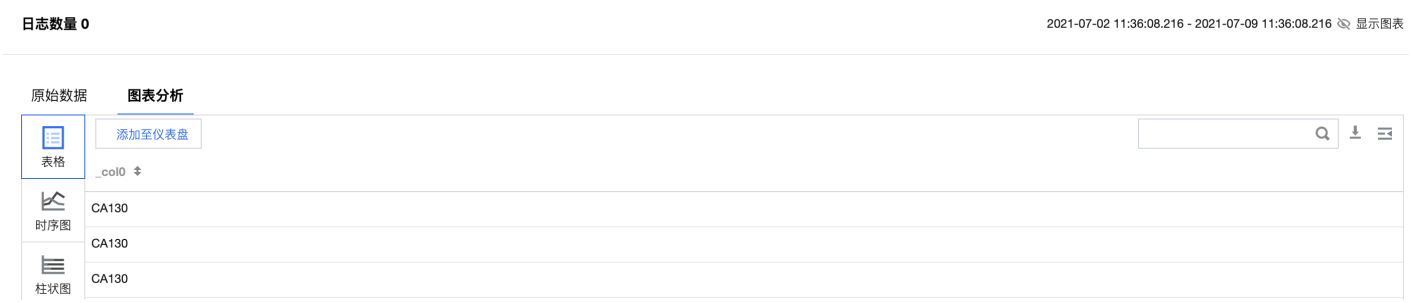
- 将 remote_user 字符串中 123 的字符替换为 ABC 对应的字符，返回结果为 Varchar 类型。

```
* | SELECT translate(remote_user, '123', 'ABC')
```



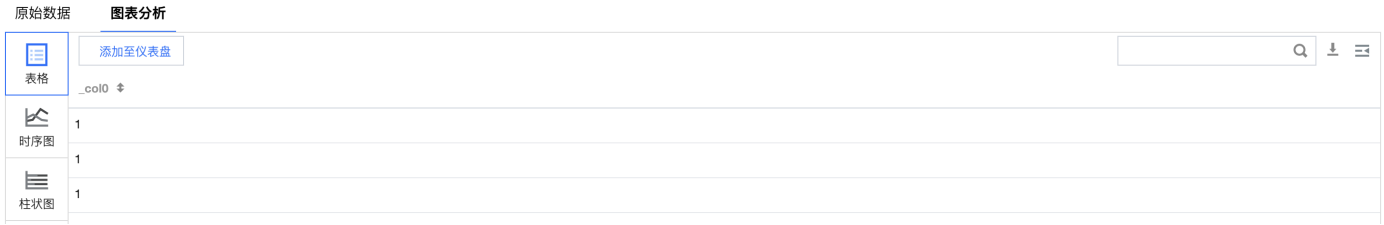
- 从字段值 remote_user 中提取第2位至第5位的值。

```
* | SELECT substr(remote_user, 2, 5)
```



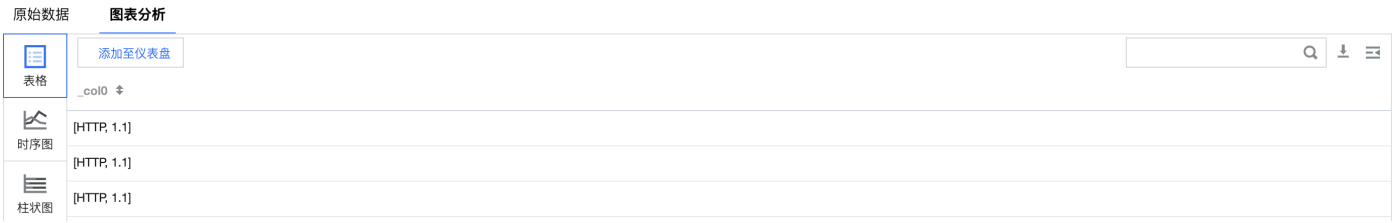
- 返回字母 H 在 http_protocol 字段值中的位置。

```
* | SELECT strpos(http_protocol, 'H')
```



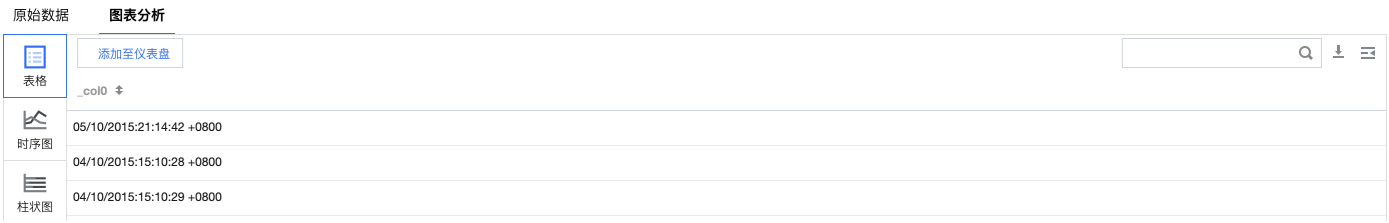
- 使用正斜线（/）将 http_protocol 字段的值拆分成2个子串，并返回子串的集合。

```
* | SELECT split(http_protocol, '/', 2)
```



- **将 time_local 字段值中的 oct 替换成10。**

```
* | select replace(time_local, 'Oct', '10')
```



日期和时间函数

最近更新时间：2022-07-01 09:26:14

日志服务（Cloud Log Service，CLS）提供时间分组函数、时间截断函数、时间间隔函数和时序补全函数，支持对日志中的日期和时间进行格式转换，分组聚合等处理。

> 注意：
> 日期和时间函数中，转换 UNIX 时间戳（unixtime）除 histogram 函数和 time_series 函数采用 UTC+8时区外，其他函数均采用 UTC+0时区。如需转换时区，可使用具备指定时区功能的函数（例如 from_unixtime(__TIMESTAMP__/1000, 'Asia/Shanghai')）或手动为 unixtime 添加时区偏移（例如 date_trunc('second', cast(__TIMESTAMP__+8*60*60*1000 as timestamp))）。

基本函数

函数名称	说明	示例
current_date	返回当前日期。 • 返回值格式：YYYY-MM-DD，例如2021-05-21。 • 返回值类型：DATE	* | select current_date
current_time	返回当前时间。 • 返回值格式：HH:MM:SS.Ms Time zone，例如 17:07:52.143+08:00。 • 返回值类型：TIME	* | select current_time
current_timestamp	返回当前时间时间戳。 • 返回值格式：YYYY-MM-DDTHH:MM:SS.Ms Time zone，例如2021-07-15T17:10:56.735+08:00[Asia/Shanghai]。 • 返回值类型：TIMESTAMP	* | select current_timestamp
current_timezone()	返回 IANA 定义的时区（America/Los_Angeles）或相对于 UTC 的偏移时差（+08:35）。 返回值类型：VARCHAR，例如 Asia/Shanghai。	* | select current_timezone()
localtime	返回本地时间。 • 返回值格式：HH:MM:SS.Ms，例如：19:56:36。 • 返回值类型：TIME	* | select localtime
localtimestamp	返回本地的日期和时间。 • 返回值格式：YYYY-MM-DD HH:MM:SS.Ms，例如：2021-07-15 19:56:26.908。 • 返回值类型：TIMESTAMP	* | select localtimestamp
now()	返回当前日期和时间，与 current_timestamp 函数同等用法。 • 返回值格式：YYYY-MM-DDTHH:MM:SS.Ms Time zone，例如2021-07-15T17:10:56.735+08:00[Asia/Shanghai]。 • 返回值类型：TIMESTAMP	* | select now()
last_day_of_month(x)	返回月份最后一天。 • 返回值格式：YYYY-MM-DD，例如2021-05-31。 • 返回值类型：DATE	* | select last_day_of_month(cast(__TIMESTAMP__ as timestamp))
from_iso8601_date(string)	把 ISO8601 格式的日期表达式转化为 DATE 类型的日期表达式。 • 返回值格式：YYYY-MM-DD，例如2021-05-31。 • 返回值类型：DATE	* | select from_iso8601_date('2021-03-21')
from_iso8601_timestamp(string)	把 ISO8601 格式的日期时间表达式转化为具有时区的 Timestamp 类型的日期时间表达式。 • 返回值格式：HH:MM:SS.Ms Time zone，例如 17:07:52.143+08:00。 • 返回值类型：TIMESTAMP	* | select from_iso8601_timestamp('2020-05-13')
from_unixtime(unixtime)	把 Unix 时间戳转化为 TIMESTAMP 类型的日期时间表达式。 • 返回值格式：YYYY-MM-DD HH:MM:SS.Ms，例如：2017-05-17 01:41:15.000。 • 返回值类型：TIMESTAMP	示例1：* | select from_unixtime(1494985275) 示例2：* | select from_unixtime(__TIMESTAMP__/1000)

函数名称	说明	示例
from_unixtime(unixtime, zone)	把 Unix 时间戳转化为具有时区的 TIMESTAMP 类型的日期时间表达式。 • 返回值格式：YYYY-MM-DD HH:MM:SS.Ms Time zone，例如：2017-05-17T09:41:15+08:00[Asia/Shanghai]。 • 返回值类型：TIMESTAMP	示例1：* | select from_unixtime(1494985275, 'Asia/Shanghai') 示例2：* | select from_unixtime(__TIMESTAMP__/1000, 'Asia/Shanghai')
to_unixtime(timestamp)	把 TIMESTAMP 类型的日期时间表达式转化为 Unixtime 时间戳。 返回值类型：LONG。例如：1626347592.037。	* | select to_unixtime(cast(__TIMESTAMP__ as timestamp))
to_milliseconds(interval)	以毫秒为单位返回间隔的时间值。 返回值类型：BIGINT。例如：300000。	* | select to_milliseconds(INTERVAL 5 MINUTE)
to_iso8601(x)	将 DATE 类型或 TIMESTAMP 类型的日期和时间表达式转换为 ISO8601 格式的日期和时间表达式。	* | select to_iso8601(current_timestamp)
timezone_hour(timestamp)	返回 TIMESTAMP 所属时区的小时偏移量。	* | SELECT current_timestamp, timezone_hour(current_timestamp)
timezone_minute(timestamp)	返回 TIMESTAMP 所属时区的分钟偏移量	* | SELECT current_timestamp, timezone_minute(current_timestamp)

时间分组函数

支持按固定时间间隔对日志数据进行分组聚合统计，例如统计每5分钟的访问次数等场景。

函数格式

```
histogram(time_column, interval)
```

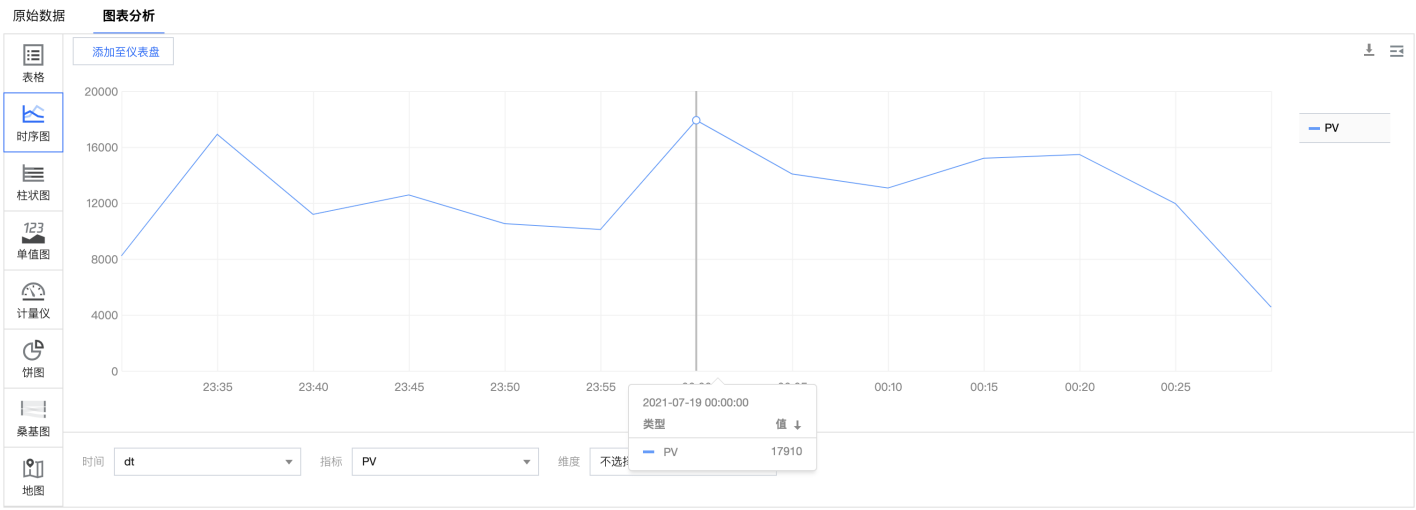
参数说明

参数	说明
time_column	时间列（KEY），例如 __TIMESTAMP__。该列的值必须为毫秒的 LONG 类型 UNIX 时间戳或 TIMESTAMP 类型的日期时间表达式。 如果时间列不符合上述要求，可以使用 cast 函数将 ISO8601 格式的时间字符串转换为 TIMESTAMP 类型，例如cast('2020-08-19T03:18:29.000Z' as timestamp)，或使用 date_parse 函数转换其他自定义类型时间字符串。 时间列使用 TIMESTAMP 时，其对应的日期时间表达式需要为 UTC+0 时区。如果日期时间表达式本身为其他时区，需通过计算调整为 UTC+0 时区。例如原始时间为北京时间（UTC+8）时，使用cast('2020-08-19T03:18:29.000Z' as timestamp) - interval 8 hour进行调整。
interval	时间间隔，支持单位为 SECOND（秒）、MINUTE（分）、HOUR（小时）、DAY（天）、MONTH（月）、YEAR（年）。例如时间间隔5分钟，即 INTERVAL 5 MINUTE。

示例

统计每5分钟访问次数 PV 值。

```
* | select histogram(__TIMESTAMP__, INTERVAL 5 MINUTE) AS dt, count(*) as PV group by dt order by dt limit 1000
```



时间补全函数

time_series() 支持按固定时间间隔对日志数据进行分组聚合统计，与histogram()函数的主要差异在于支持补全查询时间窗口内缺失的数据。

> **注意：**
> time_series() 函数必须搭配 GROUP BY 语法和 ORDER BY 语法使用，且 ORDER BY 语法不支持 desc 排序方式。

函数格式

```
time_series(time_column, interval, format, padding)
```

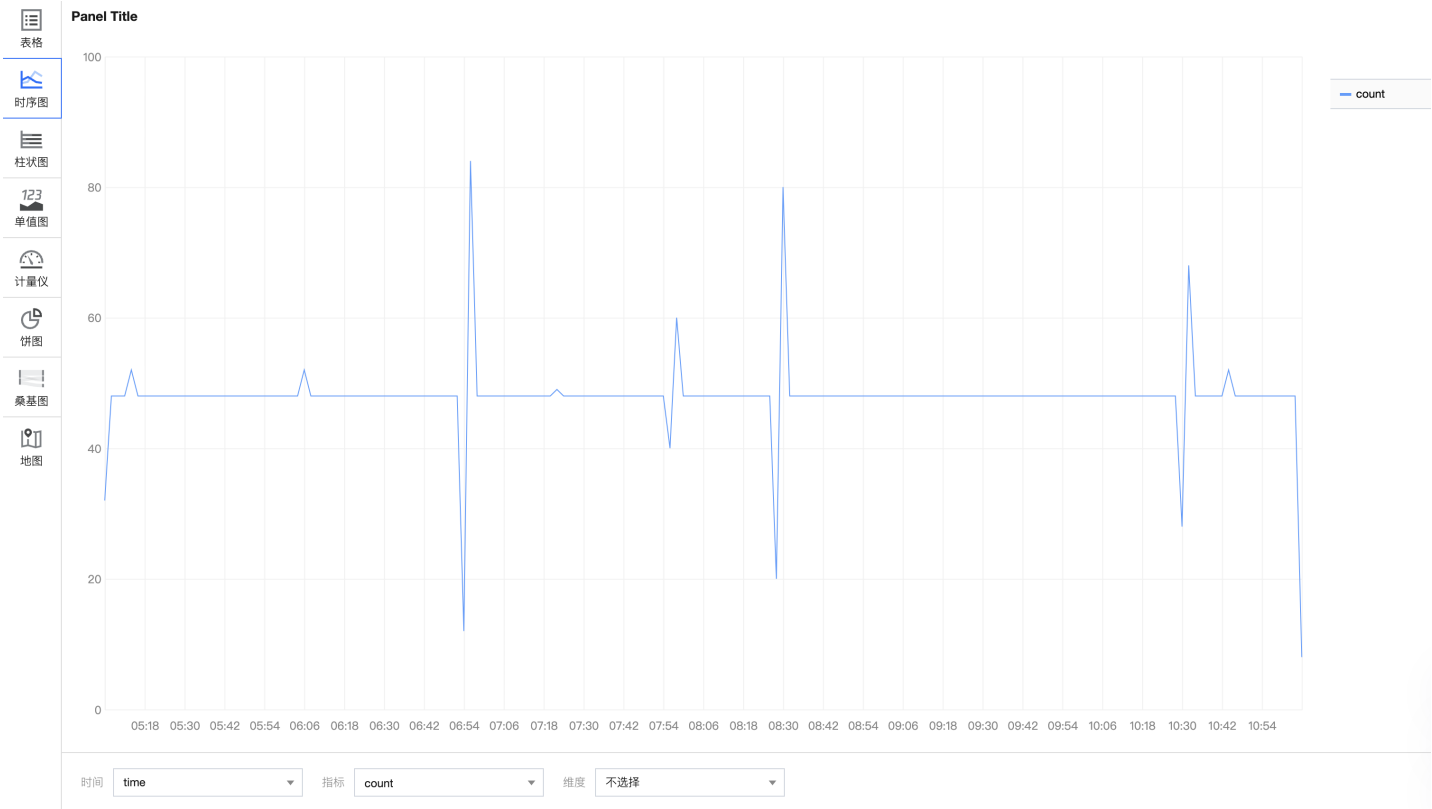
参数说明

参数	说明
time_column	时间列（KEY），例如 __TIMESTAMP__。该列的值必须为毫秒的 LONG 类型 UNIX 时间戳或 TIMESTAMP 类型的日期时间表达式。 如果时间列不符合上述要求，可以使用 cast 函数将 ISO8601 格式的时间字符串转换为 TIMESTAMP 类型，例如cast('2020-08-19T03:18:29.000Z' as timestamp)，或使用 date_parse 函数转换其他自定义类型时间字符串。 时间列使用 TIMESTAMP 时，其对应的日期时间表达式需要为 UTC+0 时区。如果日期时间表达式本身为其他时区，需通过计算调整为 UTC+0 时区。例如原始时间为北京时间（UTC+8）时，使用cast('2020-08-19T03:18:29.000Z' as timestamp) - interval 8 hour进行调整。
interval	时间间隔，支持单位为s（秒）、m（分）、h（小时）、d（天）。例如5分钟，即5m。
format	返回结果为 format 时间格式。
padding	补全的内容，包括： • 0：将缺失的值补全为0。 • null：将缺失的值补全为 null。 • last：将缺失的值补全为上一个时间点的值。 • next：将缺失的值补全为下一个时间点的值。 • avg：将缺失的值补全为前后两个时间点的平均值。

示例

按照2分钟的时间粒度进行数据补全，查询和分析语句如下：

```
* | select time_series(__TIMESTAMP__, '2m', '%Y-%m-%dT%H:%i:%s+08:00', '0') as time, count(*) as count group by time order by time limit 1000
```



时间截断函数

date_trunc() 函数根据您指定的日期时间部分截断日期时间表达式，支持按照秒、分钟，小时、日、月、年对齐。该函数常用于需要按照时间进行统计分析的场景。

函数名	说明	示例
date_trunc(unit,x)	将 x 截断至 unit 单位。x 为 timestamp 类型。	* | SELECT date_trunc('second', cast(__TIMESTAMP__ as timestamp))

截断支持如下粒度：

unit	转化结果	说明
second	2021-05-21 05:20:01.000	-
minute	2021-05-21 05:20:00.000	-
hour	2021-05-21 05:00:00.000	-
day	2021-05-21 00:00:00.000	返回指定日期的零点。
week	2021-05-19 00:00:00.000	返回指定周的周一零点
month	2021-05-01 00:00:00.000	返回指定月份的第一天零点。
quarter	2021-04-01 00:00:00.000	返回指定季度的第一天零点
year	2021-01-01 00:00:00.000	返回本年度第一天零点。

时间提取函数

时间提取函数用于日期和时间表达式中提取指定的时间部分，例如提取年份、月份等。

函数名	说明	示例
extract(field FROM x)	从日期和时间表达式（x）中提取指定的时间部分（field）。	* |select extract(hour from cast('2021-05-21 05:20:01.100' as timestamp))

field 支持如下取值：year、quarter、month、week、day、day_of_month、day_of_week、dow、day_of_year、doy、year_of_week、yow、hour、minute、second。

extract(field FROM x) 也可以简化为 field() 的形式，例如 extract(hour from cast('2021-05-21 05:20:01.100' as timestamp)) 可简化为 hour(cast('2021-05-21 05:20:01.100' as timestamp)) 。

field	提取结果	说明	简化形式
year	2021	提取目标日期中的年份。	year(x)
quarter	2	提取目标日期所属的季度。	quarter(x)
month	5	提取目标日期中的月份。	month(x)
week	20	计算目标日期是在一年中的第几周。	week(x)
day	21	提取目标日期中的天数，按月分别计算，等同于 day_of_month。	day(x)
day_of_month	21	等同于 day。	day(x)
day_of_week	5	计算目标日期是一周中的第几天，等同于 dow。	day_of_week(x)
dow	5	等同于 day_of_week。	day_of_week(x)
day_of_year	141	计算目标日期是一年中的第几天，等同于 doy。	day_of_year(x)
doy	141	等同于 day_of_year。	day_of_year(x)
year_of_week	2021	获取目标日期在 ISO week date 中的年份，等同于 yow。	year_of_week(x)
yow	2021	等同于 year_of_week。	year_of_week(x)
hour	5	提取目标日期中的小时。	hour(x)
minute	20	提取目标日期中的分钟。	minute(x)
second	1	提取目标日期中的秒。	second(x)

时间间隔函数

时间间隔函数用来执行时间段相关的运算，例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	说明	示例
date_add(unit,value,timestamp)	在 timestamp 上加上 N 个时间单位（unit）。value 为负数则为减法。	* | SELECT date_add('day', -1, TIMESTAMP '2020-03-03 03:01:00') 返回2020年3月3日1天前的日期和时间，即2020-03-02 03:01:00。
date_diff(unit, timestamp1, timestamp2)	返回两个时间表达式之间的时间差值，例如计算 timestamp1 和 timestamp2 之间相差几个时间单位（unit）。	* |SELECT date_diff('hour', TIMESTAMP '2020-03-01 00:00:00', TIMESTAMP '2020-03-02 00:00:00') 返回2020年3月1日和3月2日之间相差的时间单位值，即相差1天。

unit 取值如下：

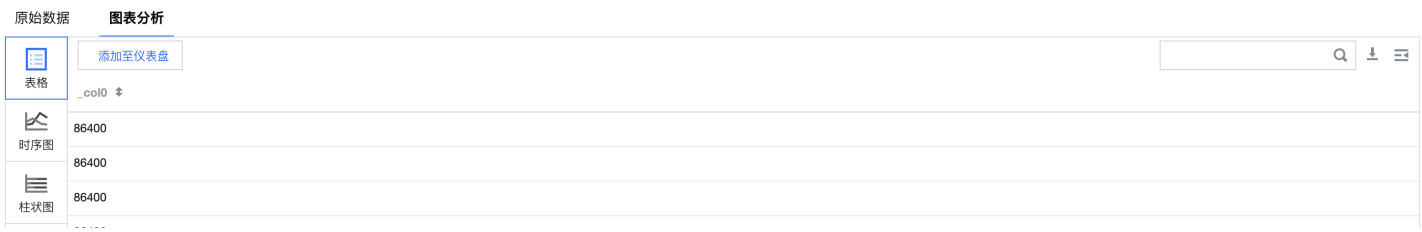
unit	说明
millisecond	毫秒
second	秒
minute	分钟
hour	小时
day	天
week	周

unit	说明
month	月
quarter	季度
year	年

示例

返回 '2020-03-01 00:00:00' 和 '2020-03-02 00:00:00' 在second单位下的间隔数值，查询分析语句如下：

```
* | SELECT date_diff('second', TIMESTAMP '2020-03-01 00:00:00', TIMESTAMP '2020-03-02 00:00:00')
```



时间持续函数

函数名	说明	示例
parse_duration(string)	将单元值字符串转换为时间段表达式。 返回值类型：INTERVAL。例如：0 00:00:00.043（D HH:MM:SS.Ms）	* | SELECT parse_duration('3.81 d')
human_readable_seconds(double)	将单元值字符串转换为时间段表达式。 返回值类型：VARCHAR。例如：1 minutes, 36 seconds。	* | SELECT human_readable_seconds(96)

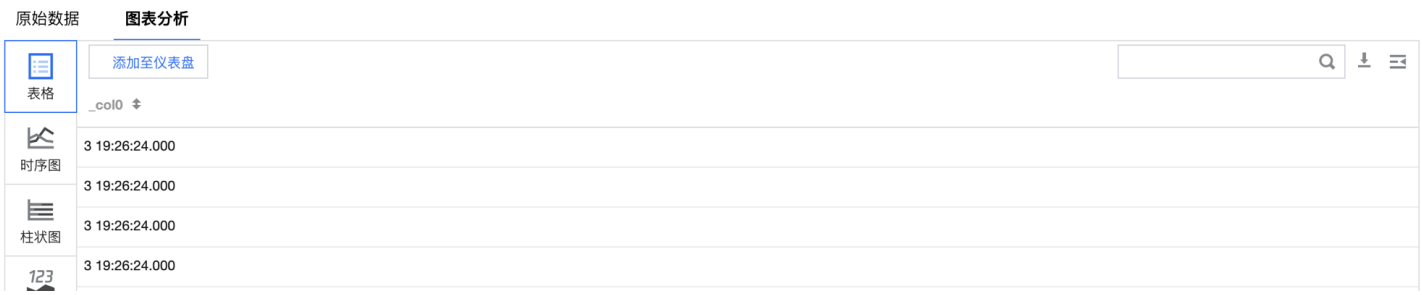
支持粒度如下：

Unit	说明
ns	纳秒
us	微秒
ms	毫秒
s	秒
m	分钟
h	小时
d	天

示例

将单元值'3.81 d' 单元字符串的值转换为间隔字符串的值，查询分析语句如下：

```
* | SELECT parse_duration('3.81 d')
```



时间格式化函数

函数名	说明	示例
date_format(timestamp, format)	把 TIMESTAMP 类型的日期时间转化为 Format 格式的字符串。	* | select date_format(cast(__TIMESTAMP__ as timestamp), '%Y-%m-%d')
date_parse(string, format)	把 Format 格式的日期时间字符串转化为 TIMESTAMP 类型。	* | select date_parse('2017-05-17 09:45:00','%Y-%m-%d %H:%i:%s')

Format 说明：

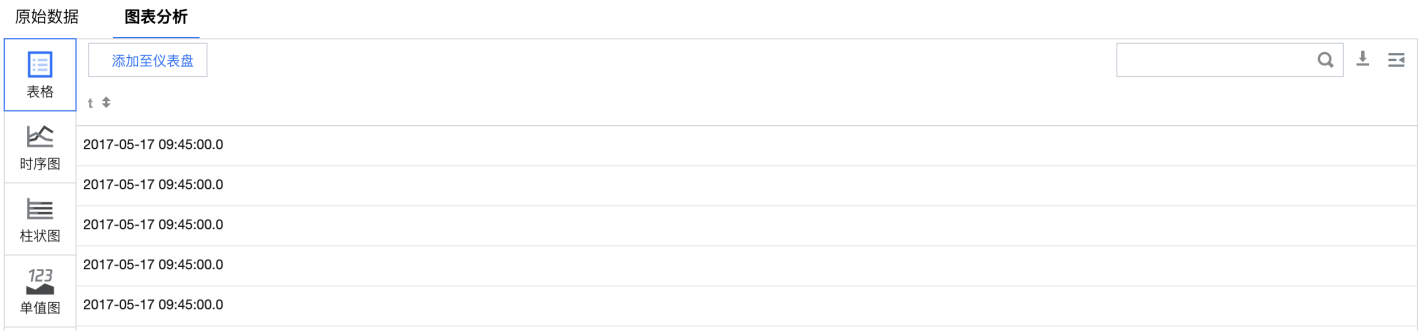
Format	说明
%a	星期的缩写。例如 Sun、Sat。
%b	月份的缩写。例如 Jan、Dec。
%c	月份。数值类型，取值范围为1 – 12。
%d	每月的第几天。十进制格式，取值范围为01 – 31。
%e	每月的第几天。十进制格式，取值范围为1 – 31。
%f	毫秒，取值范围0 – 000000。
%H	小时，24小时制。
%h	小时，12小时制。
%I	小时，12小时制。
%i	分钟。数值类型，取值范围为00 – 59。
%j	每年的第几天。取值范围为001 – 366。
%k	小时。取值范围为0 – 23。
%l	小时。取值范围为1 – 12。
%M	月份的英文表达，例如 January、December。
%m	月份的数字表达，例如 01、02。
%p	AM、PM。
%r	时间。12小时制，格式为hh:mm:ss AM/PM。
%S	秒。取值范围为00 – 59。
%s	秒。取值范围为00 – 59。
%T	时间。24小时制，格式为hh:mm:ss。
%v	每年的第几周，星期一是一周的第一天。取值范围为01 – 53。
%W	星期几的名称。例如 Sunday、Saturday。

Format	说明
%Y	4位数的年份。例如2020。
%y	2位数的年份。例如20。
%%	%的转义字符。

示例

将 fomat 格式的时间字符串 '2017-05-17 09:45:00' 转换为 TIMESTAMP 类型的日期时间表达式，即 '2017-05-17 09:45:00.0' 。查询分析语句如下：

```
* | SELECT date_parse('2017-05-17 09:45:00','%Y-%m-%d %H:%i:%s')
```



IP 地理函数

最近更新时间：2021-10-20 15:24:37

IP 地理函数可用于判断 IP 地址属于内网还是外网，也可用于分析 IP 地址所属的国家、省份、城市。本文介绍 IP 地理函数的基本语法及示例。

IP 地址函数

说明：

- 如下函数中的 KEY 参数表示日志字段（例如 ip），其值为 IP 地址；若其值为内部 IP 地址或非法字段将无法被解析，以 “NULL” 或 “未知” 展示。
- IP 目前仅支持 IPv4 地址，IPv6 暂不支持。
- 基于 IP 地址分配机制的限制，IP 地址库无法100%准确涵盖所有 IP 地址的地理信息，极少部分 IP 地址可能会查询不到详细地理信息或地理信息存在错误。

函数名称	说明	示例
ip_to_domain(KEY)	判断目标 IP 地址是内网地址还是外网地址。内网则返回 intranet，外网则返回 internet，非法 IP 地址则返回 invalid。	* | SELECT ip_to_domain(ip)
ip_to_country(KEY)	分析目标 IP 地址所属国家或地区。返回结果为国家或地区的中文名称。	* | SELECT ip_to_country(ip)
ip_to_country_code(KEY)	分析目标 IP 地址所属国家或地区的代码。返回结果为国家或地区的代码。	* | SELECT ip_to_country_code(ip)
ip_to_country_geo(KEY)	分析目标 IP 地址所属国家或地区的经纬度。返回结果为国家或地区的经纬度。	* | SELECT ip_to_country_geo(ip)
ip_to_province(KEY)	分析目标 IP 地址所属省份。返回结果为省份的中文名称。	* | SELECT ip_to_province(ip)
ip_to_province_code(KEY)	分析目标 IP 地址所属省份的代码。返回结果为省份的行政区划代码。	* | SELECT ip_to_province_code(ip)
ip_to_province_geo(KEY)	分析目标 IP 地址所属省份的经纬度。返回结果为省份的经纬度。	* | SELECT ip_to_province_geo(ip)
ip_to_city	分析目标 IP 地址所属城市。返回结果为城市的中文名称，国外城市为英文名。	* | SELECT ip_to_city(ip)
ip_to_city_code	分析目标 IP 地址所属城市的代码。返回结果为城市的行政区规划代码，暂不支持台湾省城市。	* | SELECT ip_to_city_code(ip)
ip_to_city_geo	分析目标 IP 地址所属城市的经纬度。返回结果为城市的经纬度，不支持国外城市。	* | SELECT ip_to_city_geo(ip)
ip_to_provider(KEY)	分析目标 IP 地址对应的网络运营商，返回结果为网络运营商名称。	* | SELECT ip_to_provider(ip)

IP 网段函数

说明：

- 如下函数中的 KEY 参数表示日志字段（例如 ip），其值为 IP 地址。
- 其中 ip_subnet_min、ip_subnet_max、ip_subnet_range 函数中的字段值为子网掩码格式的 IP 地址（例如：192.168.1.0/24），如果字段值为通用的 IP 地址，则需要使用 cancat 函数将其转换为子网掩码格式。

函数名	说明	示例
ip_prefix(KEY,prefix_bits)	获取目标 IP 地址的前缀。返回结果为子网掩码格式 IP 地址。例如 192.168.1.0/24。	* | SELECT ip_prefix(ip,24)

函数名	说明	示例
ip_subnet_min(KEY)	获取 IP 网段中的最小 IP 地址。返回结果为 IP 地址，例如 192.168.1.0。	* | SELECT ip_subnet_min(concat(ip,'/24'))
ip_subnet_max(KEY)	获取 IP 网段中的最大 IP 地址。返回结果为 IP 地址，例如 192.168.1.255。	* | SELECT ip_subnet_max(concat(ip,'/24'))
ip_subnet_range(KEY)	获取 IP 网段范围。返回结果为 Array 类型的 IP 地址，例如 [[192.168.1.0, 192.168.1.255]]。	* | SELECT ip_subnet_range(concat(ip,'/24'))
is_subnet_of	判断目标 IP 地址是否在某网段内。返回结果为布尔值。	* | SELECT is_subnet_of('192.168.0.1/24', ip)
is_prefix_subnet_of	判断目标网段是否为某网段的子网。返回结果为布尔值。	* | SELECT is_prefix_subnet_of('192.168.0.1/24',concat(ip,'/24'))

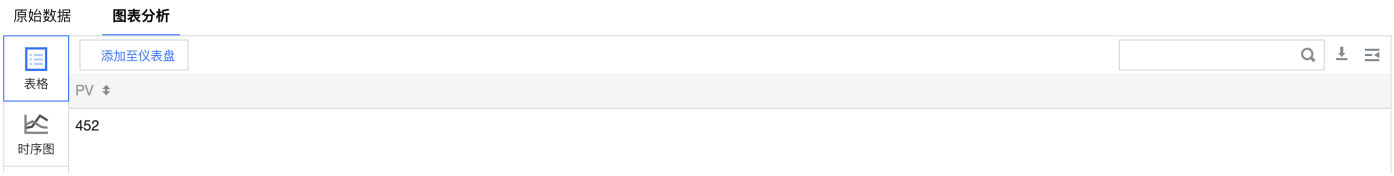
示例

此处列举了 IP 地理函数在不同场景下的查询和分析示例。您在执行查询和分析操作后，还可以选择合适的统计图表展示查询和分析结果。

> **说明：**
> 如下示例中的 ip 为日志字段。

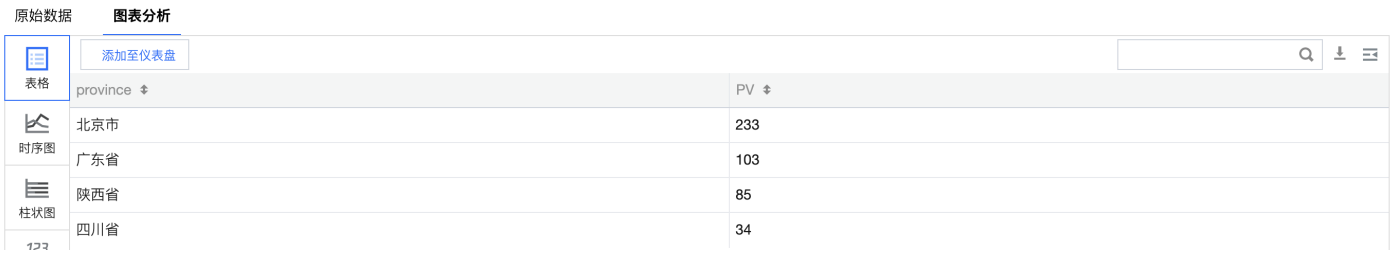
- 统计不是来自内网的请求总数。

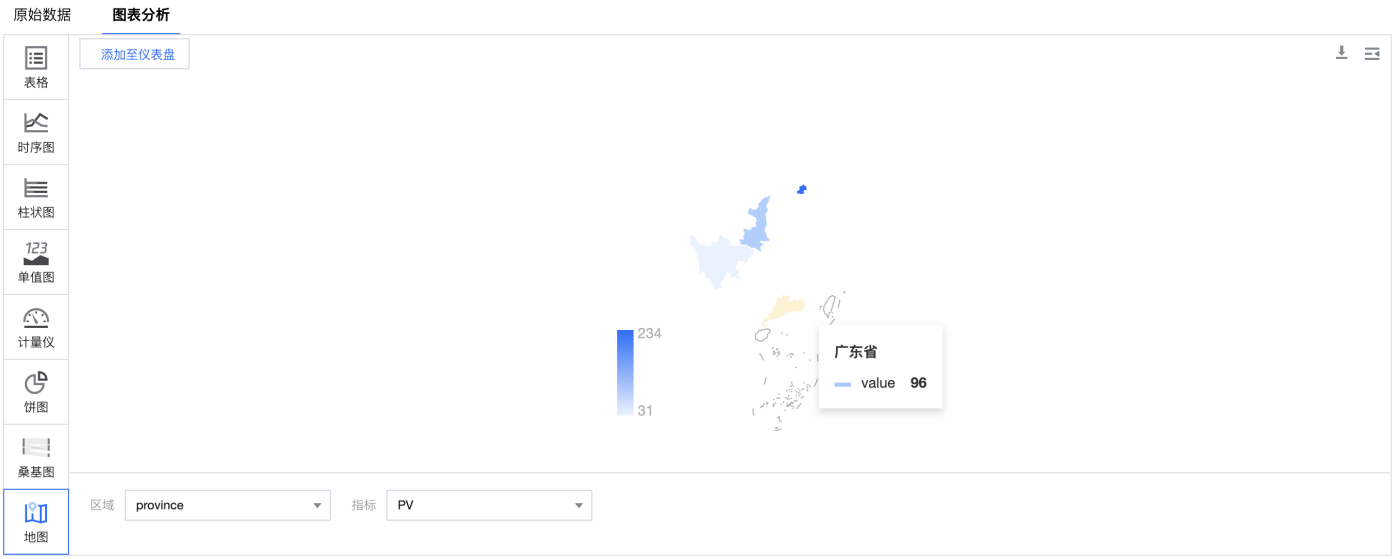
```
* | SELECT count(*) AS PV where ip_to_domain(ip)!='intranet'
```



- 统计请求总数 Top10的省份。

```
* | SELECT ip_to_province(ip) AS province, count(*) as PV GROUP BY province ORDER BY PV desc LIMIT 10
```



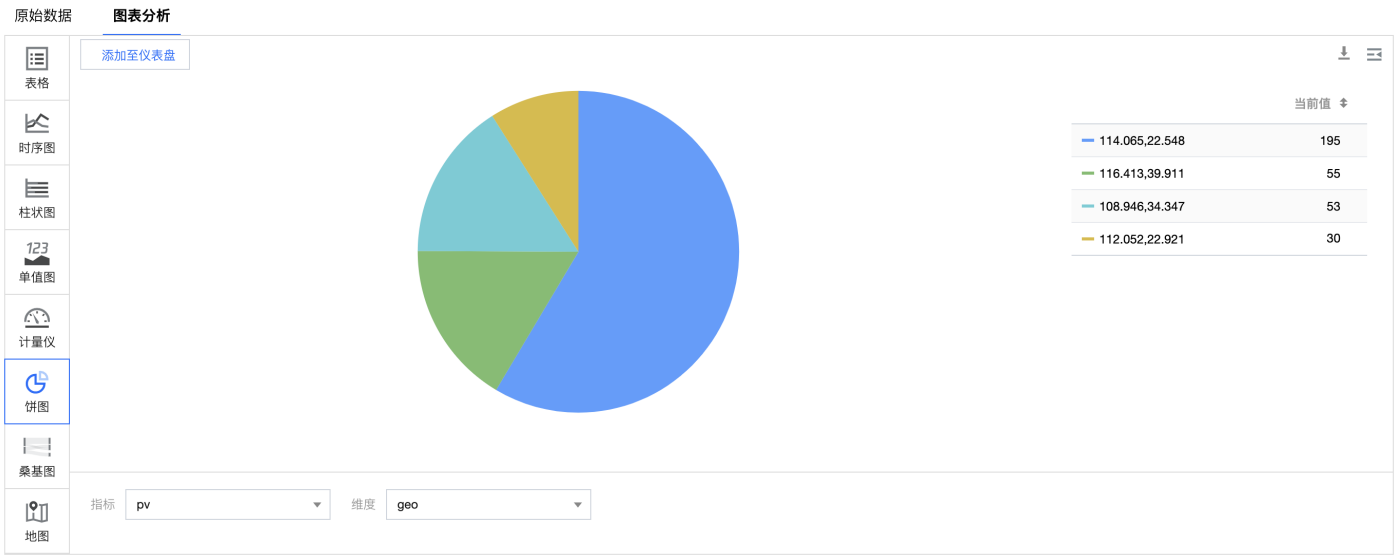


如果上述结果中包含了内网请求，且您希望过滤这部分请求，可参考如下查询和分析语句。

```
* | SELECT ip_to_province(ip) AS province, count(*) as PV where ip_to_domain(ip)!='intranet' GROUP BY province ORDER BY PV desc LIMIT 10
```

- **统计 IP 地址的经纬度，确认客户端分布情况。**

```
* | SELECT ip_to_geo(ip) AS geo, count(*) AS pv GROUP BY geo ORDER BY pv DESC
```



URL 函数

最近更新时间：2021-12-15 16:36:35

本文介绍 URL 函数的语法及示例。

语法格式

URL 函数支持从标准 HTTP URL 路径中提取字段，一个标准的 URL 如下：

```
[protocol:][//host[:port]][path][?query][#fragment]
```

> 注意：
> 提取的字段中不包含 URL 分割符 ： 或 ？。

常见 URL 函数

函数名	说明	示例	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://console.cloud.tencent.com/#/project/dashboard-demo/categoryList')	/project
url_extract_host(url)	提取出 URL 中的 host，结果为 varchar 类型。	* | select url_extract_host('https://console.cloud.tencent.com/cls')	console
url_extract_parameter(url, name)	提取出 URL 中的 query 对应的参数值，结果为 varchar 类型。	* | select url_extract_parameter('https://console.cloud.tencent.com/cls?region=ap-chongqing','region')	ap-chor
url_extract_path(url)	提取出 URL 中的 path，结果为 varchar 类型。	* | select url_extract_path('https://console.cloud.tencent.com/cls?region=ap-chongqing')	cls
url_extract_port(url)	提取出 URL 中的端口，结果为 bigint 类型。	* | select url_extract_port('https://console.cloud.tencent.com:80/cls?region=ap-chongqing')	80
url_extract_protocol(url)	提取出 URL 中的协议，结果为 varchar 类型。	* | select url_extract_protocol('https://console.cloud.tencent.com:80/cls?region=ap-chongqing')	https
url_extract_query(url)	提取出 URL 中的 query，结果为 varchar 类型。	* | select url_extract_query('https://console.cloud.tencent.com:80/cls?region=ap-chongqing')	region=

函数名	说明	示例	输出结果
url_encode(value)	对 value 进行转义编码，使之能应用在 URL_query 中。 • 字母不会被解码。 • .-*_不会被编码。 • 空格被解码为+。 • 其他字符被解码为 UTF8 格式。	* | select url_encode('https://console.cloud.tencent.com:80/cls?region=ap-chongqing')	https%3 chongq
url_decode(value)	对 URL 进行解码。	* | select url_decode('https%3A%2F%2Fconsole.cloud.tencent.com%3A80%2Fcls%3Fregion%3Dap-chongqing')	https://(

数学计算函数

最近更新时间：2022-04-21 16:29:30

本文介绍数学计算函数的语法及示例。

日志服务（Cloud Log Service，CLS）日志分析功能支持对于以 int、long、double 为类型的字段通过数学统计函数和数学计算函数进行日志分析。

> 说明：
>
> - 数学计算函数支持运算符+—*/%。
> - 以下函数中的 x、y 可以为数字、日志字段或计算结果为数字的表达式。

基本语法

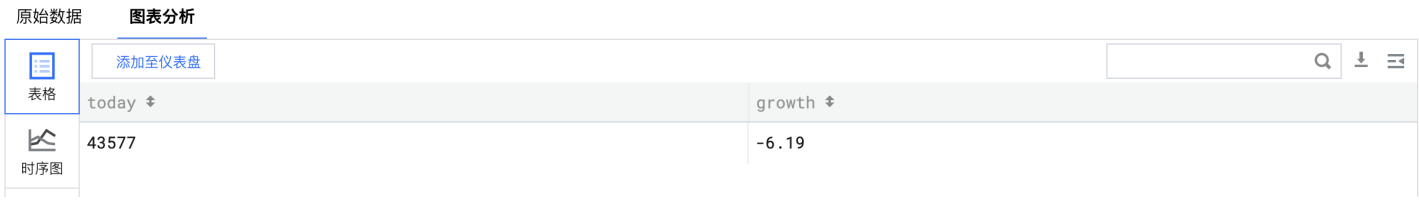
函数名称	说明
abs(x)	计算数字的绝对值。
cbrt(x)	计算数字的立方根。
sqrt(x)	计算数字的平方根。
cosine_similarity(x,y)	计算 x 和 y 之间的余弦相似度。
degrees(x)	将弧度转换为度。
radians(x)	将度转换为弧度。
e()	计算数字的自然对数。
exp(x)	返回自然对数的指数。
ln(x)	计算数字的自然对数。
log2(x)	计算以2为底的对数。
log10(x)	计算以10为底的对数。
log(x,b)	计算以 b 为底的对数。
pi()	返回包含14个小数位的π值。
pow(x,b)	计算数字的 b 次幂。
rand()	返回随机数。
random(0,n)	返回[0,n)之间的随机数。
round(x)	返回四舍五入后的取值。
round(x, N)	保留数字的 N 位小数。
floor(x)	向下取整数。
ceiling(x)	向上取整数。
from_base(varchar, bigint)	根据 BASE 编码将字符串转为数字。
to_base(x, radix)	根据 BASE 编码将数字转为字符串。
truncate(x)	截断数字的小数部分。
acos(x)	计算数字的反余弦。
asin(x)	计算数字的反正弦。
atan(x)	计算数字的反正切。
atan2(y,x)	计算两个数字相除的结果的反正切。

函数名称	说明
cos(x)	计算数字的余弦。
sin(x)	计算数字的正弦。
cosh(x)	计算数字的双曲余弦。
tan(x)	计算数字的正切。
tanh(x)	计算数字的双曲正切。
infinity()	返回正无穷的数值。
is_nan(x)	判断目标值是否为非数值。
nan()	返回一个 NaN 值（Not a Number）。
mod(x, y)	用于计算 x 与 y 相除的余数。
sign(x)	返回 x 的符号，通过1、0、-1表示。
width_bucket(x, bound1, bound2, n)	将一段数值（bound1 – bound2）划分成大小相同的 n 个 Bucket，返回 x 所属的 Bucket。 例如* | select timeCost,width_bucket(timeCost,10,1000,5)
width_bucket(x, bins)	使用数组（bins）指定 Bucket 的范围，返回 x 所属的 Bucket。 例如* | select timeCost,width_bucket(timeCost,array[10,100,1000])

示例

同比今天和昨天的访问 PV，并使用百分数表示。查询和分析语句如下：

```
* | SELECT diff [1] AS today, round((diff [3] -1.0) * 100, 2) AS growth FROM (SELECT compare(pv, 86400) as diff FROM (SELECT COUNT(*) as pv FROM log))
```



数学统计函数

最近更新时间：2021-10-20 15:24:23

本文介绍数学统计函数的语法及示例。

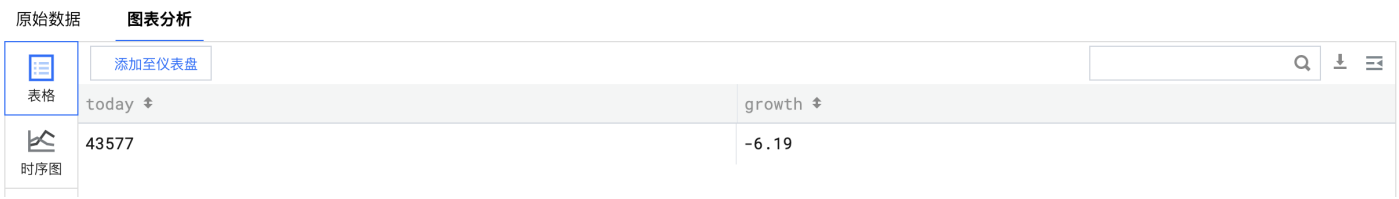
基本语法

函数名称	说明
corr(key1, key2)	计算两列的相关度。计算结果范围为[0,1]。
covar_pop(key1, key2)	计算两列的总体协方差。
covar_samp(key1, key2)	计算两列的样本协方差。
regr_intercept(key1, key2)	返回输入值的线性回归截距。key1 是依赖值，key2 是独立值。
regr_slope(key1, key2)	返回输入值的线性回归斜率。key1 是依赖值，key2 是独立值。
stddev(key)	计算 key 列的样本标准差。与 stddev_samp 函数同义。
stddev_samp(key)	计算 key 列的样本标准差。
stddev_pop(key)	返回 key 列的总体标准差。
variance(key)	计算 key 列的样本方差。与 var_samp 函数同义。
var_samp(key)	计算 key 列的样本方差。
var_pop(key)	计算 key 列的总体方差。

示例

- 示例1：计算两列数据的相关度。查询和分析语句如下：

```
* | SELECT corr(request_length, request_time)
```



- 示例2：查询请求长度的样本标准差和总体标准差。查询和分析语句如下：

```
* | SELECT stddev(request_length) as "样本标准差", stddev_pop(request_length) as "总体标准差", time_series(__TIMESTAMP__, '1m', '%Y-%m-%d %H:%i:%s', '0') AS dt GROUP BY dt
```


原始数据
图表分析
表格
时序图
柱状图
单值图
计量仪
饼图
桑基图
添加至仪表盘
150
120
90
60
30
0
21:03 21:07 21:11 21:15 21:19 21:23 21:27 21:31 21:35 21:39 21:43 21:47 21:51 21:55 21:59 22:03 22:07 22:11 22:15 22:19 22:23 22:27 22:31 22:35
总体标准差
样本标准差
时间
dt
指标
总体标准差, 样本标准差
维度
不选择

通用聚合函数

最近更新时间：2022-06-15 09:30:38

本文介绍通用聚合函数的基本语法及示例。

聚合函数是对一组值执行计算并返回计算结果的函数，日志服务支持如下聚合函数：

> 说明：
> 日志服务分析语句中，表示字符串的字符必须使用单引号（'）包裹，无符号包裹或被双引号（""）包裹的字符表示字段名或列名。例如：'status' 表示字符串 status，status 或 "status" 表示日志字段 status。

函数语句	说明	示例
arbitrary(KEY)	随机返回目标列中一个非 NULL 的值。	* | SELECT arbitrary(request_method) AS request_method
avg(KEY)	计算目标列的算数平均值。	* | SELECT AVG(request_time)
bitwise_and_agg(KEY)	返回目标列汇所有值按位与运算（AND）的结果。	* | SELECT bitwise_and_agg(status)
bitwise_or_agg(KEY)	返回目标列汇所有值按位或运算（OR）的结果。	* | SELECT bitwise_or_agg(request_length)
checksum(KEY)	计算目标列的校验和值，返回结果为 BASE 64编码类型。	* | SELECT checksum(request_method) AS request_method
count(*)	表示所有的行数。	* | SELECT COUNT(*) WHERE http_status >200
count(1)	COUNT(1) 等同于 COUNT(*)，表示所有的行数。	* | SELECT COUNT(1)
count(KEY)	计算某一 KEY 列非 NULL 的行数。	* | SELECT COUNT(request_time) WHERE request_time >5.0
count_if(boolean)	统计满足指定条件的日志条数。	* | select count_if(returnCode>=400) as errorCounts
geometric_mean(KEY)	计算 KEY 的几何平均数，KEY 不允许包含负数，否则结果为 NaN。	* | SELECT geometric_mean(request_time) AS request_time
max(KEY)	查询 KEY 中的最大值。	* | SELECT MAX(request_time) AS max_request_time
max_by(x,y)	返回 y 为最大值时对应的 x 值。	* | SELECT MAX_BY(request_method, request_time) AS method
max_by(x,y,n)	返回最大的 n 个 y 值对应的 x 值，返回结果为 JSON 数组。	* | SELECT max_by(request_method, request_time, 3) AS method
min(KEY)	查询 KEY 中最小值。	* | SELECT MIN(request_time) AS min_request_time
min_by(x,y)	返回 y 为最小值时对应的 x 值。	* | SELECT min_by(request_method, request_time) AS method
min_by(x,y,n)	返回最小的 n 个 y 值对应的 x 值。返回结果为 JSON 数组。	* | SELECT min_by(request_method, request_time, 3) AS method
sum(KEY)	计算 KEY 的总值。	* | SELECT SUM(body_bytes_sent) AS sum_bytes
bool_and(boolean)	是否所有的日志都满足指定的条件，如果是则返回 TRUE ，否则返回 FALSE。	* | select bool_and(returnCode>=400)
bool_or(boolean)	是否有任意一条日志满足指定的条件，如果是则返回 TRUE ，否则返回 FALSE。	* | select bool_or(returnCode>=400)
every(boolean)	等效于 bool_and(boolean)。	* | select every(returnCode>=400)

参数说明

参数	说明
KEY	表示日志字段名称。
x	参数值为任意数据类型。
y	参数值为任意数据类型。
n	大于0的整数。

空间几何函数

最近更新时间：2022-04-21 16:29:46

本文介绍空间几何函数。

基本概念

空间几何函数支持 Well-Known Text（WKT）及 Well-Known Binary（WKB）格式描述的几何实体，相关概念可参考 Well-known text representation of geometry - Wikipedia。

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)))
几何实体集合	GEOMETRYCOLLECTION (POINT(2 3), LINESTRING (2 3, 3 4))

几何实体默认采用平面几何，在平面几何中两点之间的最短距离为直线。几何实体也支持球面几何，球面中两点之间的最短距离为大圆弧。使用 to_spherical_geography() 可将平面几何实体转换为球面几何实体。
例如：
ST_Distance(ST_Point(-71.0882, 42.3607), ST_Point(-74.1197, 40.6976)) 计算平面上的两个点的距离，值为3.4577。
ST_Distance(to_spherical_geography(ST_Point(-71.0882, 42.3607)), to_spherical_geography(ST_Point(-74.1197, 40.6976))) 计算球面上两个点的距离，值为312822.179。

计算长度时（例如 ST_Distance()和ST_Length()），单位为米，计算面积时（例如 ST_Area()），单位为平方米。

构造几何实体

函数	返回值类型	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 文本中构造一个几何实体。
ST_GeomFromBinary(varbinary)	Geometry	从 WKB 中构造一个几何实体。
ST_AsText(Geometry)	varchar	把一个空间几何实体转变成 WKT 格式。
to_spherical_geography(Geometry)	SphericalGeography	将平面几何实体转换为球面几何实体。
to_geometry(SphericalGeography)	Geometry	将球面几何实体转换为平面几何实体。

空间关系判断

函数	返回值类型	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，返回 false。
ST_Crosses(Geometry, Geometry)	boolean	当两个实体有共同内部点时，返回 true。

函数	返回值类型	说明
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。

Operations

函数	返回值类型	说明
geometry_nearest_points(Geometry, Geometry)	row(Point, Point)	返回两个几何实体之间最相近的两个点。
geometry_union(array(Geometry))	Geometry	将多个几何实体合并为一个几何实体。
ST_Boundary(Geometry)	Geometry	返回几何实体的闭包。
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。

Accessors

函数	返回值类型	说明
ST_Area(Geometry)	double	在平面几何中计算多边形面积。
ST_Area(SphericalGeography)	double	在球面几何中计算多边形面积。
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	计算平面几何中两个实体之间的最小距离。
ST_Distance(SphericalGeography, SphericalGeography)	double	计算球面几何中两个实体之间的最小距离。
ST_IsClosed(Geometry)	boolean	当实体是一个闭合空间时，返回 true。
ST_IsEmpty(Geometry)	boolean	当参数是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，并且是闭合的简单的线时，返回 true。

函数	返回值类型	说明
ST_Length(Geometry)	double	在平面几何中，计算一个线段或者多条线段的长度。
ST_Length(SphericalGeography)	double	在球面几何中，计算一个线段或者多条线段的长度。
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。
ST_XMin(Geometry)	double	返回几何体边框的 X 最小值。
ST_YMin(Geometry)	double	返回几何体边框的 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。

二进制字符串函数

最近更新时间：2021-10-20 15:24:11

本文介绍二进制字符串函数的语法。

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。

估算函数

最近更新时间：2022-06-08 15:33:27

本文介绍估算函数的基本语法及示例。

函数名称	函数语法	说明
approx_distinct 函数	approx_distinct(x)	估算 x 列的唯一值的个数。
approx_percentile 函数	approx_percentile(x,percentage)	对于 x 列进行正序排序，返回大约处于 percentage 位置的值。
	approx_percentile(x,array[percentage01, percentage02...])	对于 x 列进行正序排序，返回大约处于 percentage01、percentage02 位置的值。

approx_distinct 函数

approx_distinct 函数用于估算字段值中不重复值的个数，估算结果标准误差为2.3%。

语法

```
approx_distinct(x)
```

参数说明

参数	说明
x	参数值为任意数据类型。

返回值类型

bigint 类型。

示例

使用 count 函数计算 PV，使用 approx_distinct 函数估算不重复的 client_ip 字段值作为 UV。

```
* | SELECT count(*) AS PV, approx_distinct(ip) AS UV
```

approx_percentile 函数

approx_percentile 函数用于对目标字段的值进行正序排列，返回大约处于 percentage 位置的数值。采用 T-Digest 算法进行估算，误差较低，能够满足绝大多数统计分析需求，如有需要，可使用 * | select count_if(x<(select approx_percentile(x,percentage))),count(*) 分别精确统计低于 percentage 的字段值个数和总数来验证该统计误差。

语法

- 返回处于 percentage 位置的数值，返回结果为 double 类型。

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

- 返回处于 percentage01、percentage02 位置的数值，返回结果为 array 类型。

```
approx_percentile(x, array[percentage01,percentage02...])
```

参数说明

参数	说明
x	参数值为 double 类型。
percentage	百分比值，取值范围为[0,1]。

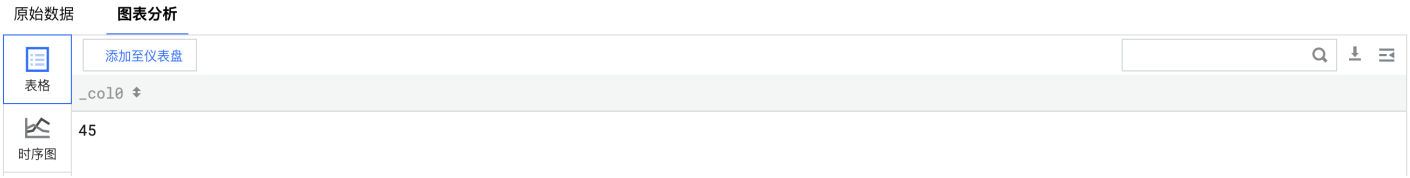
返回值类型

double 类型或 array 类型。

示例

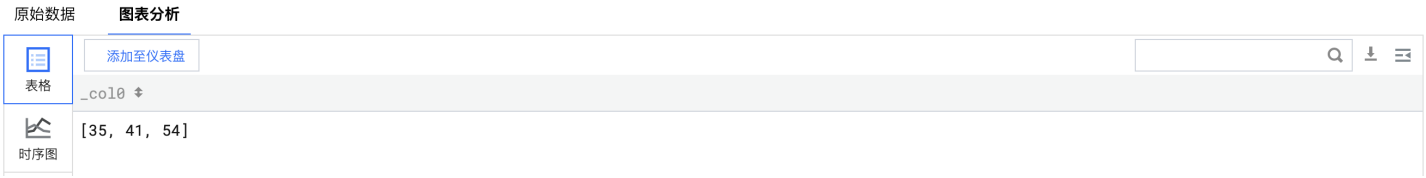
示例1：对 resTotalTime 列进行排列后，返回大约处于50%位置的 resTotalTime 字段的值。

```
* | select approx_percentile(resTotalTime,0.5)
```



示例2：对 resTotalTime 列进行排列后，返回处于10%、20%及60%位置的 resTotalTime 字段的值。

```
* | select approx_percentile(resTotalTime, array[0.2,0.4,0.6])
```



类型转换函数

最近更新时间：2022-04-21 16:31:10

本文介绍类型转换函数的基本语法及示例。

如果您在查询与分析数据时，需要区分更细维度的数据类型，您可以在查询与分析语句中使用类型转换函数转换数据的数据类型。

函数名称	语法	说明
cast 函数	cast(x as type)	转换x的数据类型。 使用 cast 函数转换数据类型时，如果某个值转换失败，将终止整个查询与分析操作。
try_cast 函数	try_cast(x as type)	转换x的数据类型。 使用 try_cast 函数转换数据类型时，如果某个值转换失败，该值返回 NULL，并跳过该值继续处理。
typeof 函数	typeof(x)	返回 x 的数据类型。

> 说明：
> 日志中可能有脏数据时，建议使用 try_cast 函数，避免因脏数据造成整个查询与分析操作失败。

cast 函数

cast 函数用于转换 x 的数据类型。使用 cast 函数转换数据类型时，如果某个值转换失败，将终止整个查询与分析操作。

语法

```
cast(x as type)
```

参数说明

参数	说明
x	参数值可以为任意类型。
type	SQL 数据类型，可选值为 bigint、varchar、double、boolean、timestamp、decimal、array 或 map。 索引数据类型和 SQL 数据类型的映射关系，请参见 附录：数据类型映射关系。

type 为 timestamp 时，x 需为毫秒级时间戳或 ISO 8601格式的时间字符串，例如1597807109000或2019-12-25T16:17:01+08:00。

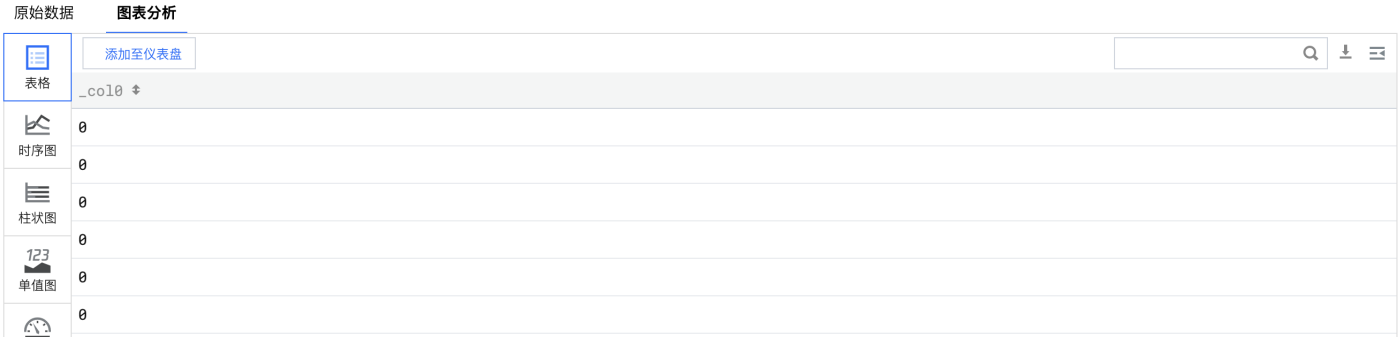
返回值类型

由您配置的 type 参数决定。

示例

1. 将数值0.01转换为 bigint 格式。

```
* | select cast(0.01 as bigint)
```



2. 将日志服务附带的日志采集时间__TIMESTAMP__转换为 TIMESTAMP 类型

```
* | select cast(__TIMESTAMP__ as timestamp)
```

try_cast 函数

try_cast 函数用于转换x的数据类型。使用 try_cast 函数转换数据类型时，如果某个值转换失败，该值返回 NULL，并跳过该值继续处理。

语法

```
try_cast(x as type)
```

参数说明

参数	说明
x	参数值可以为任意类型。
type	SQL 数据类型，可选值为 bigint、varchar、double、boolean、timestamp、decimal、array 或 map。 索引数据类型和 SQL 数据类型的映射关系，请参见 附录：数据类型映射关系。

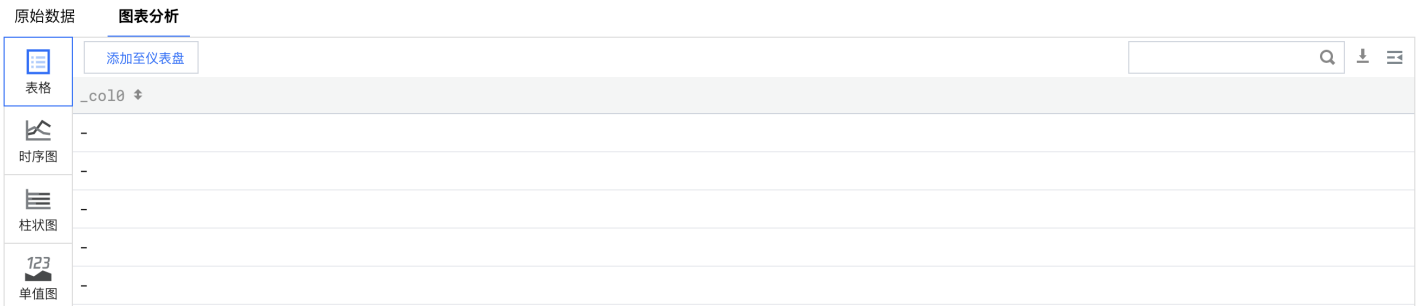
返回值类型

由您配置的 type 参数决定。

示例

将 **remote_user** 字段值转换为 varchar 格式。

```
* | select try_cast(remote_user as varchar)
```



附录：数据类型映射关系

索引数据类型和 SQL 数据类型的对应关系如下表所示：

索引的数据类型	SQL 的数据类型
long	bigint
text	varchar
double	double
json	varchar

逻辑函数

最近更新时间：2021-12-28 15:32:34

本文介绍逻辑运算的语法和示例。

逻辑运算符

运算符	描述	示例
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 时，结果为 TRUE。	a OR b
NOT	右侧运算数为 FALSE 时，结果才为 TRUE。	NOT a

与 NULL 相关的逻辑运算

a 和 b 分别取值 TRUE，FALSE 和 NULL 时的真值表如下：

AND，OR 的真值表

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

NOT 的真值表

a	NOT a
TRUE	FALSE
FALSE	TRUE
NULL	NULL

运算符

最近更新时间：2022-04-21 16:29:40

运算符是一个保留字或字符，主要用于指定 SQL语句中的条件，并在语句中连接多个条件。

算术运算符

算术运算符是用来处理四则运算的符号，是最简单最常用的符号，尤其是数字的处理，几乎都会使用到算术运算符号。

假设变量 a=1，变量 b=2，则：

运算符	描述	示例
+	加法：运算符两边的值相加	a + b
-	减法：运算符左边减去运算符右边	a - b
*	乘法 - 把运算符两边的值相乘	a * b
/	除法 - 运算符左边除以运算符右边	b / a
%	取模 - 运算符左边除以运算符右边后得到的余数	b % a

比较运算符

比较运算符用于判断值的大小关系，支持任何可比较的类型，例如 int、long、double 和 text 等。

假设变量 a=1，变量 b=2，则：

运算符	描述	示例
=	判断运算符两边的值是否相等，如果相等则条件为真。	a = b
!=	判断运算符两边的值是否相等，如果不相等则条件为真。	a != b
<>	判断运算符两边的值是否相等，如果不相等则条件为真。	a <> b
>	判断运算符左边的值是否大于运算符右边的值，如果是则条件为真。	a > b
<	判断运算符左边的值是否小于运算符右边的值，如果是则条件为真。	a < b
>=	判断运算符左边的值是否大于等于运算符右边的值，如果是则条件为真。	a >= b
<=	判断运算符左边的值是否小于等于运算符右边的值，如果是则条件为真。	a <= b
IN	IN 运算符用于把某个值与一系列指定列表的值进行比较。	status IN (200,206,404)
NOT IN	IN 运算符的对立面，用于把某个值与不在一系列指定列表的值进行比较。	status NOT IN (200,206,404)
BETWEEN AND	BETWEEN 运算符用于在给定最小值和最大值范围内的一系列值中搜索值。	status between 200 AND 400
LIKE	LIKE 运算符用于把某个值与使用通配符运算符的相似值进行比较。%代表零个、一个或者多个字；_代表单个数字或者字符。	url LIKE '%.mp4'
IS NULL	NULL 运算符用于把某个值与 NULL 值进行比较，为空为真。	status IS NULL
IS NOT NULL	NULL 运算符用于把某个值与 NULL 值进行比较，不为空为真。	status IS NOT NULL
DISTINCT	语法：x IS DISTINCT FROM y 或 x IS NOT DISTINCT FROM y。 用于对比 x 和 y 是否相等，与<>的差异在于 DISTINCT 可用于对 null 的比较，详见 <> 与 DISTINCT 运算符差异。	NULL IS NOT DISTINCT FROM NULL

运算符	描述	示例
LEAST	语法：LEAST(x, y...) 返回 x,y...中的最小值。	LEAST(1,2,3)
GREATEST	语法：GREATEST(x, y...) 返回 x,y...中的最大值。	GREATEST(1,2,3)
ALL	语法：x expression operator ALL (subquery) x 满足所有条件时，返回 true，operator 支持<、>、<=、>=、=、<>、!=。	示例1：21 < ALL (VALUES 19, 20, 21) 示例2：* | SELECT 200 = ALL(SELECT status)
ANY / SOME	语法：x expression operator ANY (subquery) 或 x expression operator SOME (subquery) x 满足任意一个条件时，返回 true，operator 支持<、>、<=、>=、=、<>、!=。	示例1：'hello' = ANY (VALUES 'hello', 'world') 示例2：* | SELECT 200 = ANY(SELECT status)

<> 与 DISTINCT 运算符差异：

x	y	x = y	x <> y	x IS DISTINCT FROM y	x IS NOT DISTINCT FROM y
1	1	true	false	false	true
1	2	false	true	true	false
1	null	null	null	true	false
null	null	null	null	false	true

逻辑运算符

运算符	描述
AND	AND 运算符要求运算符两边条件同时存在为真。
OR	OR 运算符要求运算符两边任一条件存在即为真。
NOT	NOT 运算符是所用的逻辑运算符的对立面。例如 NOT EXISTS、NOT BETWEEN、NOT IN 等。

位运算

最近更新时间：2021-11-01 16:14:25

本文介绍位运算函数的基本语法及示例。

函数名称	语句	说明
bit_count 函数	bit_count(x, bits)	统计 x 的二进制表示中1的个数。
bitwise_and 函数	bitwise_and(x, y)	以二进制的形式对 x，y 进行与运算。
bitwise_not 函数	bitwise_not(x)	以二进制的形式对 x 的所有位进行取反运算。
bitwise_or 函数	bitwise_or(x, y)	以二进制形式对 x，y 进行或运算。
bitwise_xor 函数	bitwise_xor(x, y)	以二进制形式对 x，y 进行异或运算。

bit_count 函数

bit_count 函数用于统计 x 中1的个数。

语法

```
bit_count(x, bits)
```

参数说明

参数	说明
x	参数值为 bigint 类型。
bits	位数，例如64位。

返回值类型

bigint 类型。

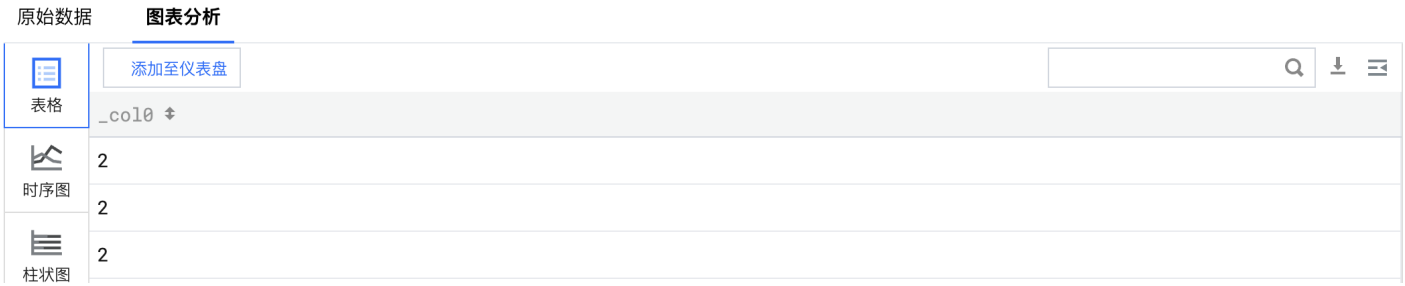
示例

计算数字24的二进制数，并返回其二进制数中1的个数。

- 查询和分析语句

```
* | SELECT bit_count(24, 64)
```

- 查询和分析结果



bitwise_and 函数

bitwise_and 函数以二进制形式对 x 和 y 进行与运算。

语法

```
bitwise_and(x, y)
```

参数说明

参数	说明
x	参数值为 bigint 类型。
y	参数值为 bigint 类型。

返回值类型

bigint 类型。

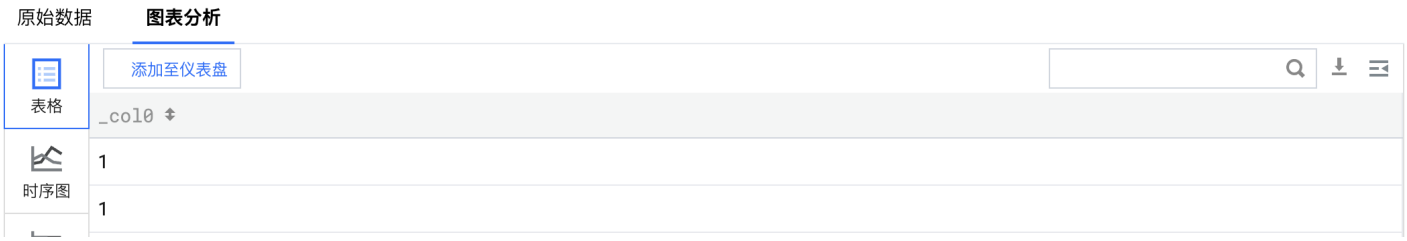
示例

以二进制形式对数字3和5进行与运算。

- 查询和分析语句

```
* | SELECT bitwise_and(3, 5)
```

- 查询和分析结果



bitwise_not 函数

bitwise_not 函数以二进制形式对 x 的所有位进行取反运算。

语法

```
bitwise_not(x)
```

参数说明

参数	说明
x	参数值为 bigint 类型。

返回值类型

bigint 类型。

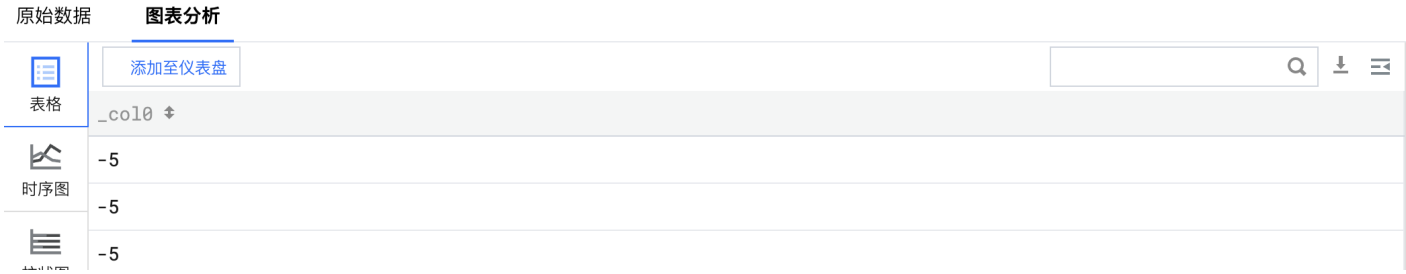
示例

以二进制形式对数字4的所有位进行取反运算。

- 查询和分析语句

```
* | SELECT bitwise_not(4)
```

- 查询和分析结果



bitwise_or 函数

bitwise_or 函数以二进制形式对 x 和 y 进行或运算。

语法

```
bitwise_or(x, y)
```

参数说明

参数	说明
x	参数值为 bigint 类型。
y	参数值为 bigint 类型。

返回值类型

bigint 类型。

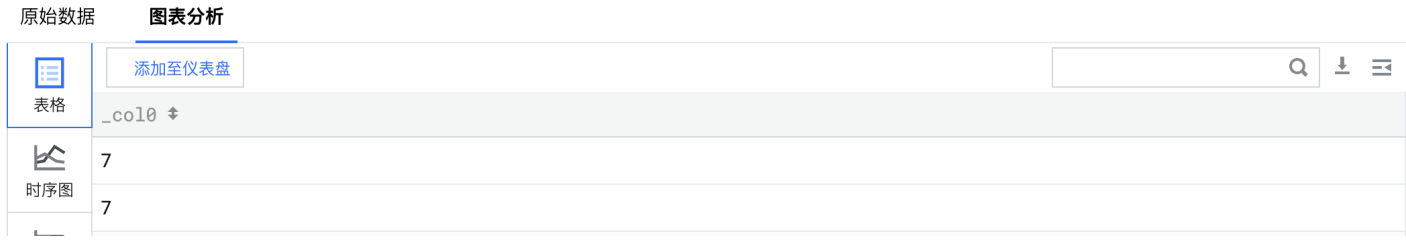
示例

以二进制形式对数字3和5进行或运算。

- 查询和分析语句

```
* | SELECT bitwise_or(3, 5)
```

- 查询和分析结果



bitwise_xor 函数

bitwise_xor 函数以二进制形式对 x 和 y 进行异或运算。

语法

```
bitwise_xor(x, y)
```

参数说明

参数	说明
x	参数值为 bigint 类型。

参数	说明
y	参数值为 bigint 类型。

返回值类型

bigint 类型。

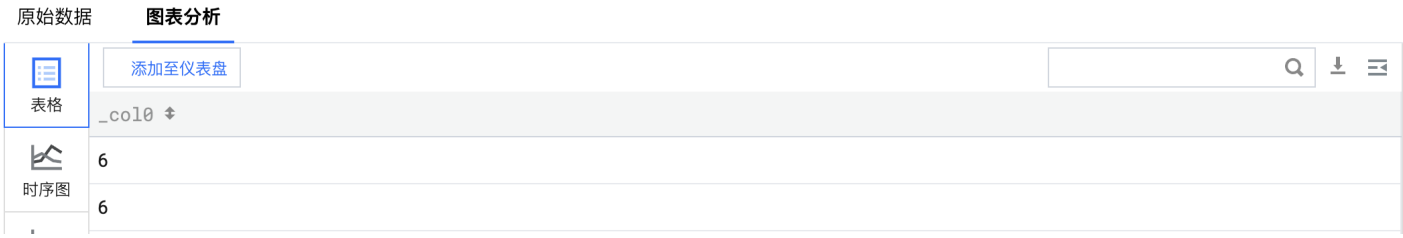
示例

以二进制形式对数字3和5进行异或运算。

- 查询和分析语句

```
* | SELECT bitwise_xor(3, 5)
```

- 查询和分析结果



正则式函数

最近更新时间：2021-12-28 10:59:02

本文介绍正则式函数基本语法及示例。

日志服务支持如下正则式函数：

函数名称	语法	说明
regexp_extract_all 函数	regexp_extract_all(x, regular expression)	提取目标字符串中符合正则表达式的子串，并返回所有子串的合集。
	regexp_extract_all(x, regular expression, n)	提取目标字符串中符合正则表达式的子串，并返回与目标捕获组匹配的子串合集。
regexp_extract 函数	regexp_extract(x, regular expression)	提取并返回目标字符串中符合正则表达式的第一个子串。
	regexp_extract(x, regular expression, n)	提取目标字符串中符合正则表达式的子串，然后返回与目标捕获组匹配的第一个子串。
regexp_like 函数	regexp_like(x, regular expression)	判断目标字符串是否符合正则表达式。
regexp_replace 函数	regexp_replace(x, regular expression)	删除目标字符串中符合正则表达式的子串，返回未被删除的子串。
	regexp_replace(x, regular expression, replace string)	替换目标字符串中符合正则表达式的子串，返回被替换后的字符串。
regexp_split 函数	regexp_split(x, regular expression)	使用正则表达式分割目标字符串，返回被分割后的子串合集。

regexp_extract_all 函数

regexp_extract_all 函数用于提取目标字符串中符合正则表达式的子串合集。

语法

- 提取目标字符串中符合正则表达式的子串，并返回所有子串的合集。

```
regexp_extract_all(x, regular expression)
```

- 提取目标字符串中符合正则表达式的子串，然后返回与目标捕获组匹配的子串合集。

```
regexp_extract_all(x, regular expression, n)
```

参数说明

参数	说明
x	参数值为 varchar 类型。
regular expression	包含捕获组的正则表达式。例如(\d)(\d)(\d)表示三个捕获组 。
n	第 n 个捕获组。n 为从1开始的整数。

返回值类型

array 类型。

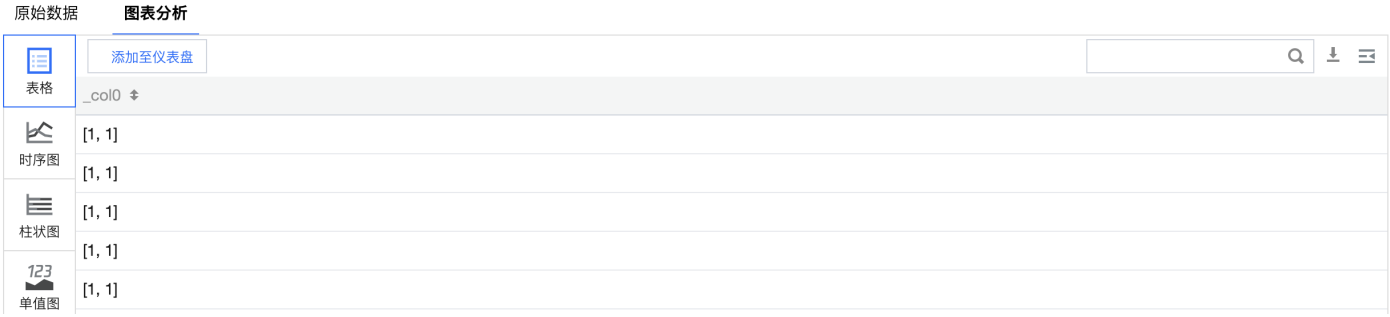
示例

提取 http_protocol 字段中所有的数字。

- 查询和分析语句

```
* | SELECT regexp_extract_all(http_protocol, '\d+')
```

- 查询和分析结果



regexp_extract 函数

regexp_extract 函数用于提取目标字符串中符合正则表达式的第一个子串。

语法

- 提取并返回目标字符串中符合正则表达式的第一个子串。

```
regexp_extract(x, regular expression)
```

- 提取目标字符串中符合正则表达式的子串，然后返回与目标捕获组匹配的第一个子串。

```
regexp_extract(x, regular expression, n)
```

参数说明

参数	说明
x	参数值为 varchar 类型。
regular expression	包含捕获组的正则表达式。例如(\d)(\d)(\d)表示三个捕获组 。
n	第 n 个捕获组。n 为从1开始的整数。

返回值类型

varchar 类型。

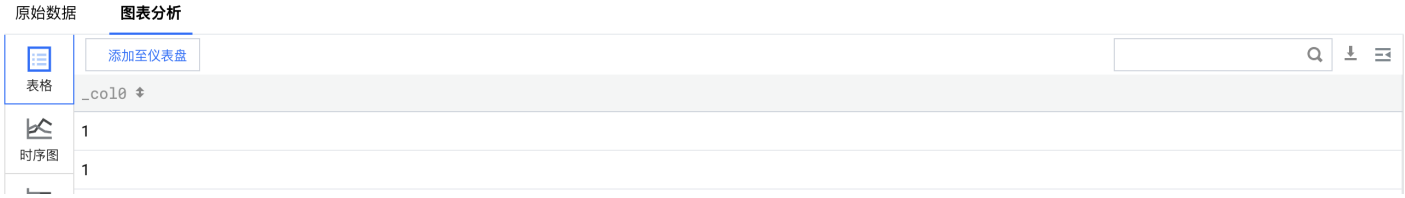
示例

示例1：提取 http_protocol 字段值中的第一个数字。

- 查询和分析语句

```
* | SELECT regexp_extract_all(http_protocol, '\d+')
```

- 查询和分析结果

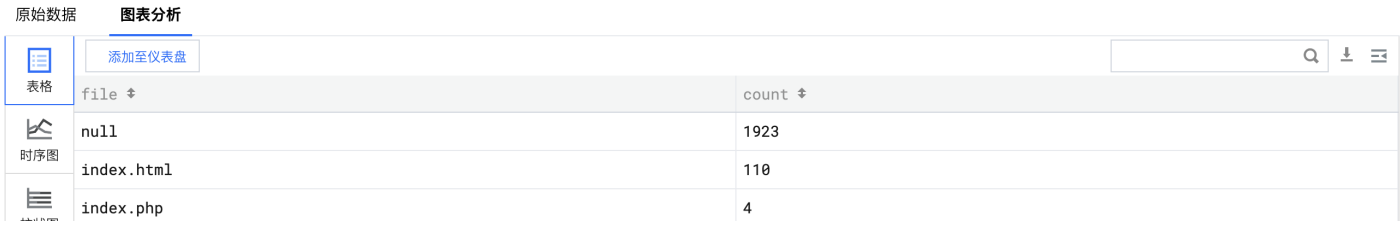


示例2：提取 request_uri 字段值中的文件部分，并统计各个文件的访问次数。

- 查询和分析语句

```
* | SELECT regexp_extract(request_uri, '.*\/(index.*)', 1) AS file, count(*) AS count GROUP BY file
```

- 查询和分析结果



regexp_like 函数

regexp_like 函数用于判断目标字符串是否符合正则表达式。

语法

```
regexp_like ( x, regular expression )
```

参数说明

参数	说明
x	参数值为 varchar 类型。
regular expression	正则表达式。

返回值类型

boolean 类型。

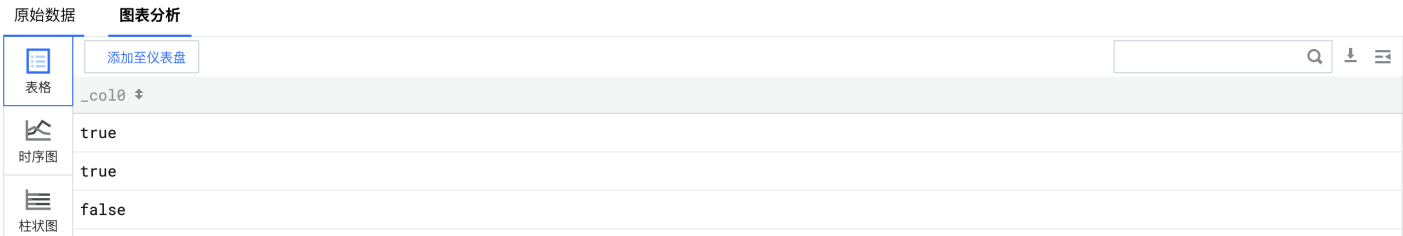
示例

判断 server_protocol 字段值中是否包含数字。

- 查询和分析语句

```
* | select regexp_like(server_protocol, '\d+')
```

- 查询和分析结果



regexp_replace 函数

regexp_replace 函数用于删除或替换目标字符串中符合正则表达式的子串。

语法

- 删除目标字符串中符合正则表达式的子串，返回未被删除的子串。

```
regexp_replace ( x, regular expression )
```

- 替换目标字符串中符合正则表达式的子串，返回被替换后的字符串。

```
regexp_replace ( x, regular expression, replace string )
```

参数说明

参数	说明
x	参数值为 varchar 类型。
regular expression	正则表达式。
replace string	用于替换的子串.

返回值类型

string 类型。

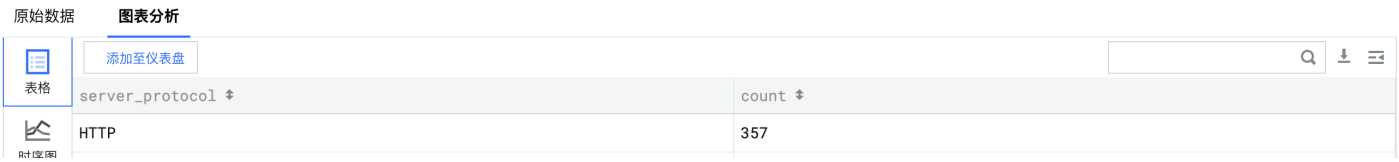
示例

删除 **server_protocol** 字段值中的版本号部分，并统计不同通信协议对应的请求数量。

- 查询和分析语句

```
* | select regexp_replace(server_protocol, '.\d+') AS server_protocol, count(*) AS count GROUP BY server_protocol
```

- 查询和分析结果



regexp_split 函数

regexp_split 函数用于分割目标字符串，返回被分割后的子串合集。

语法

```
regexp_split ( x, regular expression )
```

参数说明

参数	说明
x	参数值为 varchar 类型。
regular expression	正则表达式。

返回值类型

array类型。

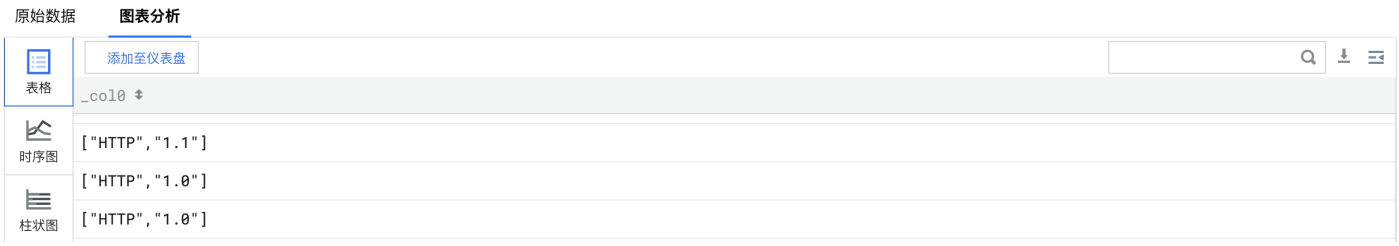
示例

使用正斜线（/）分割 **server_protocol** 字段的值。

- 查询和分析语句

```
* | select regexp_split(server_protocol, '/')
```

- **查询和分析结果**



Lambda 函数

最近更新时间：2021-11-10 11:51:03

本文介绍 Lambda 函数对日志数据进行分析处理的语法及示例。

日志服务支持您在 SQL 分析语句中定义 Lambda 表达式，并将该表达式传递给指定函数，丰富函数的表达。

语法

Lambda 表达式需与函数一起使用，例如 filter 函数、reduce 函数、transform 函数、zip_with 函数。Lambda 表达式的语法如下：

```
parameter -> expression
```

参数	说明
parameter	用于传递参数的标识符。

expression	表达式，大多数的 Mysql 表达式都可以在 Lambda 表达式使用。例如：
	`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)`

示例

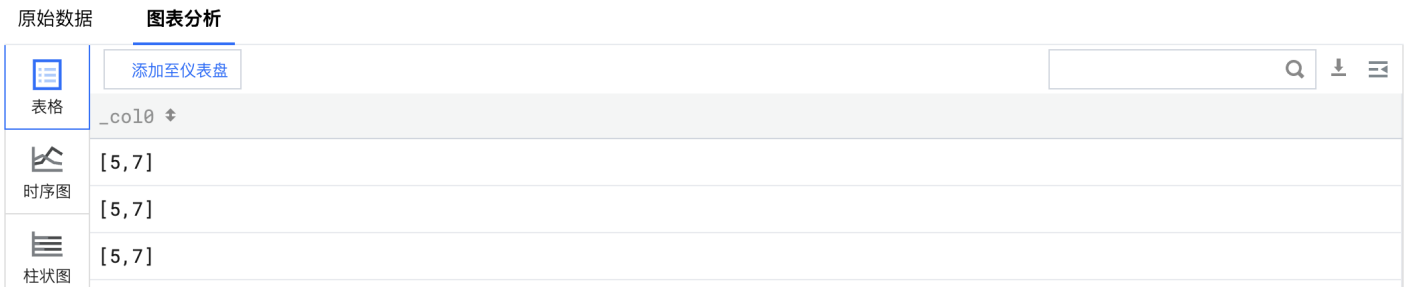
示例1：使用 Lambda 表达式 x-> x is not null

返回数组[5，null，7，null]中非 null 的元素。

- 查询和分析语句

```
* | SELECT filter(array[5, null, 7, null], x -> x is not null)
```

- 查询和分析结果



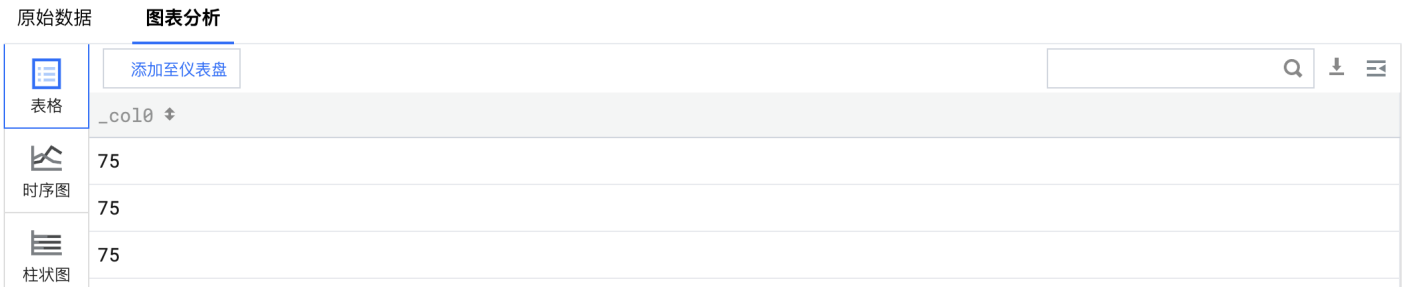
示例2：使用 Lambda 表达式0，(s, x) -> s + x, s -> s

返回数组[5，20，50]中各个元素相加的结果。

- 查询和分析语句

```
* | SELECT reduce(array[5, 20, 50], 0, (s, x) -> s + x, s -> s)
```

- 查询和分析结果



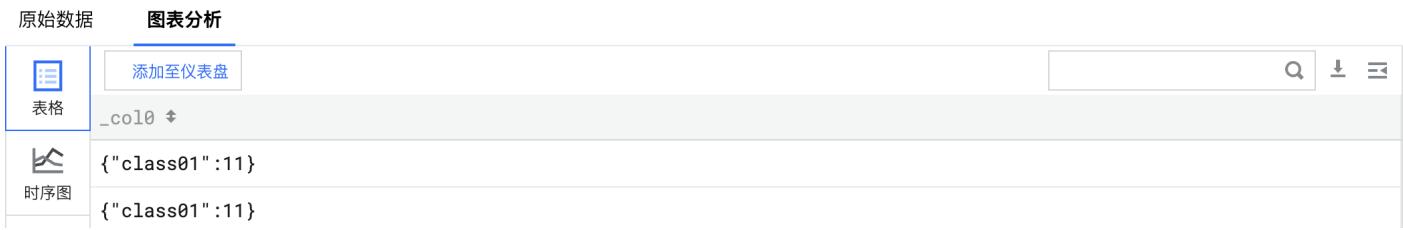
示例3：使用 Lambda 表达式(k, v) -> v> 10

将两个数组映射为一个 Map 且 Map 中的键值大于10。

- 查询和分析语句

```
* | SELECT map_filter(map(array['class01', 'class02', 'class03'], array[11, 10, 9]), (k,v) -> v > 10)
```

- 查询和分析结果



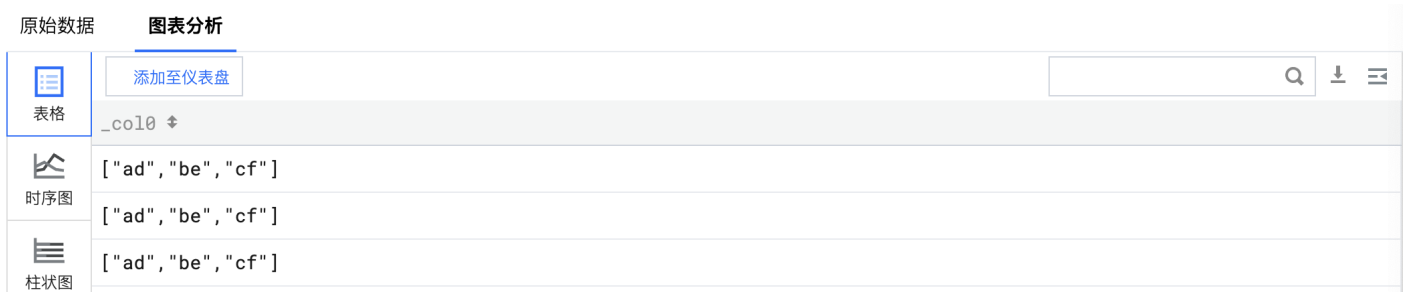
示例4：使用 Lambda 表达式(x, y) -> (y, x)

将对换两个数组的元素位置，并提取数组中索引相同的元素组成一个新的二维数组。

- 查询和分析语句

```
* | SELECT zip_with(array['a', 'b', 'c'], array['d', 'e', 'f'], (x, y) -> concat(x, y))
```

- 查询和分析结果



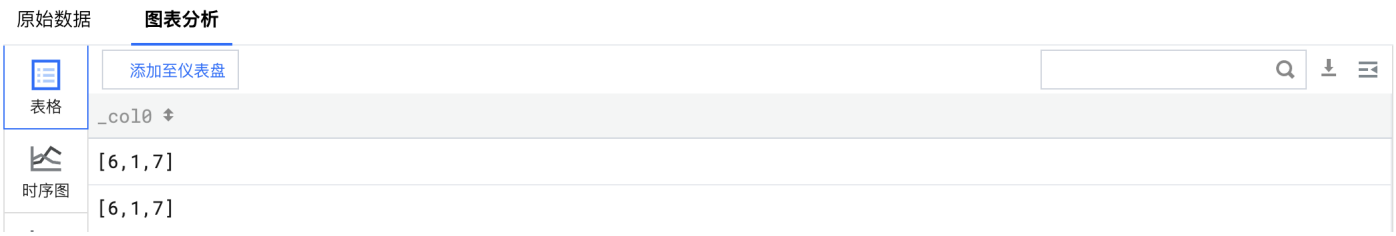
示例5：使用 Lambda 表达式 x -> coalesce（x, 0）+1

将数组[5, null, 6]的各个元素加1，然后返回。如果数组中包含 null 元素，则转换为0，再加1。

- 查询和分析语句

```
* | SELECT transform(array[5, NULL, 6], x -> coalesce(x, 0) + 1)
```

- 查询和分析结果



其他示例

```
* | SELECT filter(array[], x -> true)
* | SELECT map_filter(map(array[],array[]), (k, v) -> true)
* | 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))
* | SELECT reduce(array[2147483647, 1], cast(0 AS bigint), (s, x) -> s + x, s -> s)
* | SELECT reduce(array[5, 20, null, 50], 0, (s, x) -> s + x, s -> s)
* | SELECT transform(array[array[1, null, 2], array[3, null]], a -> filter(a, x -> x is not null))
```

条件表达式

最近更新时间：2022-02-21 10:29:46

本文介绍条件表达式的语法和示例。

表达式	语法	说明
CASE WHEN 表达式	CASE WHEN condition1 THEN result1 [WHEN condition2 THEN result2] [ELSE result3] END	过条件判断，对数据进行归类。
IF 表达式	IF(condition, result1)	如果 condition 为 true，则返回 result1，否则返回 null。
	IF(condition, result1, result2)	如果 condition 为 true，则返回 result1，否则返回 result2。
NULLIF 表达式	NULLIF(expression1, expression2)	比较两个表达式的值是否相等。如果相等，则返回 null，否则返回第一个表达式的值。
TRY 表达式	TRY(expression)	捕获异常信息，使得系统继续执行查询和分析操作。
COALESCE 表达式	COALESCE(expression1, expression2...)	获取多个表达式中的第一个非 NULL 值。

CASE WHEN 语法

CASE WHEN 语法用于对数据进行归类。

语法

```
CASE WHEN condition1 THEN result1
[WHEN condition2 THEN result2]
[ELSE result3]
END
```

参数说明

参数	说明
condition	条件表达式。
result	返回结果。

示例

- 示例1：从 **http_user_agent** 字段值中提取浏览器信息，归为 Chrome、Safari 和 unknown 三种类型并计算三种类型对应的访问 PV。
  - 查询和分析语句

```
* |
SELECT
CASE
WHEN http_user_agent like '%Chrome%' then 'Chrome'
WHEN http_user_agent like '%Safari%' then 'Safari'
ELSE 'unknown'
END AS http_user_agent,
count(*) AS pv
GROUP BY
http_user_agent
```

- 查询和分析结果

原始数据 图表分析

表格 时序图 柱状图

添加至仪表盘

http_user_agent	pv
Chrome	43
Safari	4
unknown	1829

- 示例2：统计不同请求时间的分布情况。
  - 查询和分析语句

```
* |
SELECT
CASE
WHEN request_time < 0.001 then 't0.001'
WHEN request_time < 0.01 then 't0.01'
WHEN request_time < 0.1 then 't0.1'
WHEN request_time < 1 then 't1'
ELSE 'overtime'
END AS request_time,
count(*) AS pv
GROUP BY
request_time
```

  - 查询和分析结果

原始数据 图表分析

表格 时序图 柱状图 单值图

添加至仪表盘

request_time	pv
overtime	52
t0.001	292
t0.01	1479
t0.1	42
t1	11

IF 语法

IF 语法用于对数据进行归类，类似于 CASE WHEN 表达式。

语法

- 如果 condition 为 true，则返回 result1，否则返回 null。

```
IF(condition, result1)
```

- 如果 condition 为 true，则返回 result1，否则返回 result2。

```
IF(condition, result1, result2)
```

参数说明

参数	说明
condition	条件表达式。
result	返回结果。

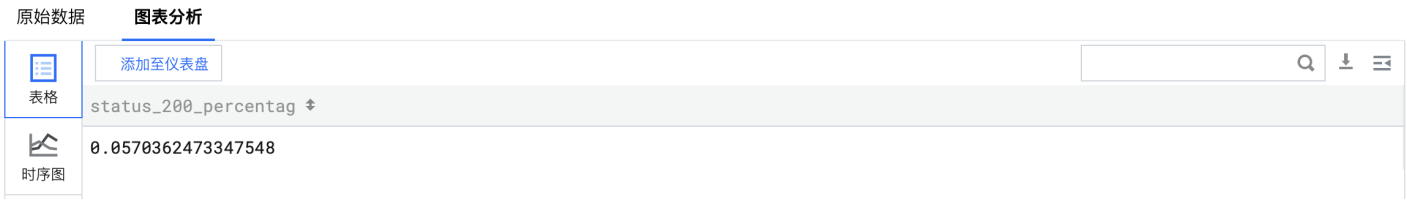
示例

计算状态码为200的请求占所有请求的比例。

- 查询和分析语句

```
* |
SELECT
sum(IF(status = 200, 1, 0)) * 1.0 / count(*) AS status_200_percentag
```

- 查询和分析结果



NULLIF 表达式

NULLIF 表达式用于比较两个表达式的值是否相等。如果相等，则返回 null，否则返回第一个表达式的值。

语法

```
NULLIF(expression1, expression2)
```

参数说明

参数	说明
expression	任何有效的标量表达式。

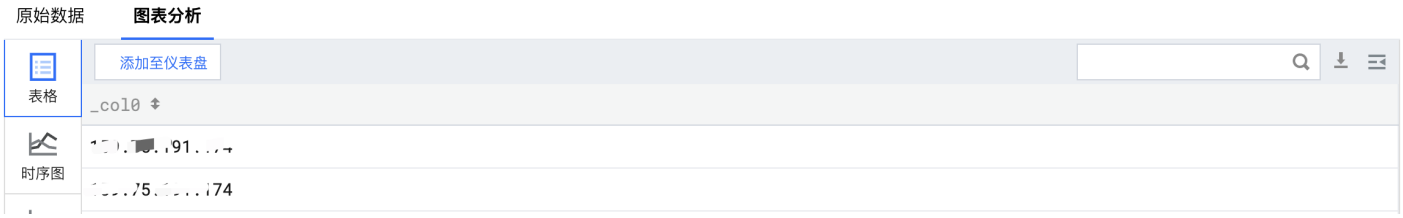
示例

判断 **server_addr**、**http_host** 两个字段的值是否相同。当不相同时，返回 **server_addr** 字段的值。

- 查询和分析语句

```
* | SELECT NULLIF(server_addr,http_host)
```

- 查询和分析结果



TRY 表达式

TRY 表达式用于捕获异常信息，使得系统继续执行查询和分析操作。

语法

```
TRY(expression)
```

参数说明

参数	说明
expression	任何类型的表达式。

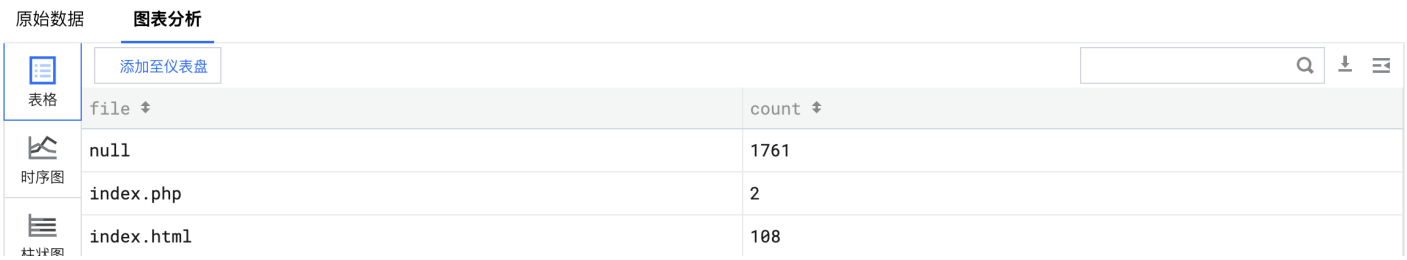
示例

当执行 regexp_extract 函数发生异常时，try函数会捕获异常信息并继续查询和分析操作，返回查询和分析结果。

- 查询和分析语句

```
* |
SELECT
TRY(regexp_extract(uri, '.*\/(index.*)', 1))
AS file, count(*)
AS count
GROUP BY
file
```

- 查询和分析结果



COALESCE 表达式

COALESCE 表达式用于获取多个表达式中的第一个非 NULL 值。

语法

```
COALESCE(expression1, expression2...)
```

参数说明

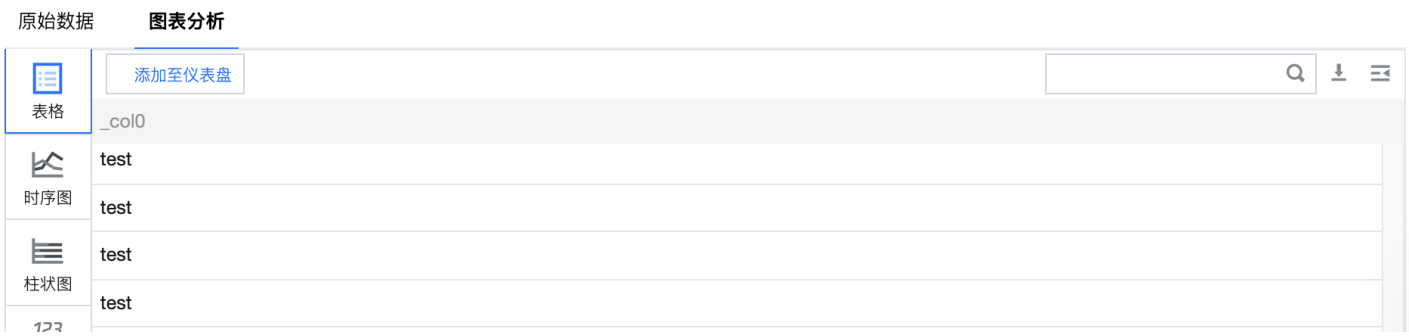
参数	说明
expression	任何有效的标量表达式。

示例

- 查询和分析语句

```
* | select COALESCE(null, 'test')
```

- 查询和分析结果



数组函数

最近更新时间：2022-04-21 16:31:32

本文档为您介绍数组相关函数的语法及示例。

函数名称	语句	含义
下标运算符 []	[x]	[]用于获取数组中的某个元素。等同于 element_at 函数。
array_agg 函数	array_agg(x)	以数组形式返回 x 中的所有值。
array_distinct 函数	array_distinct(x)	数组去重，获取数组中的唯一元素。
array_except 函数	array_except(x,y)	计算 x，y 两个数组的差集。
array_intersect 函数	array_intersect(x, y)	计算 x，y 两个数组的交集。
array_join 函数	array_join(x, delimiter)	使用指定的连接符将数组中的元素拼接为一个字符串。如果数组中包含 null 元素，则 null 元素将被忽略。 **说明：** 使用 array_join 函数时，返回结果最大为1KB，超出1KB的数据会被截断。
	array_join(x, delimiter, null_replacement)	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1KB，超出1KB的数据会被截断。
array_max 函数	array_max(x)	获取数组中的最大值。
array_min 函数	array_max(x)	获取数组中的最小值。
array_position 函数	array_position(x, element)	获取指定元素的下标，下标从1开始。如果指定元素不存在，则返回0。
array_remove 函数	array_remove(x, element)	删除数组中指定的元素。
array_sort 函数	array_sort(x)	对数组元素进行升序排序。如果有 null 元素，则 null 元素排在最后。
array_union 函数	array_union(x, y)	计算两个数组的并集。
cardinality 函数	cardinality(x)	计算数组中元素的个数。
concat 函数	concat(x, y...)	将多个数组拼接为一个数组。
contains 函数	contains(x, element)	判断数组中是否包含指定元素。如果包含，则返回 true。
element_at 函数	element_at(x, y)	返回数组中的第 y 个元素。
filter 函数	filter(x, lambda_expression)	结合 Lambda 表达式，用于过滤数组中的元素，只返回满足 Lambda 表达式的元素。
flatten 函数	flatten(x)	把将二维数组转换为一维数组。
reduce 函数	reduce(x, lambda_expression)	根据 Lambda 表达式的定义，对数组中的各个元素进行相加计算，然后返回计算结果。
reverse 函数	reverse(x)	对数组中的元素进行反向排列。
sequence 函数	sequence(x, y)	通过指定的起始值返回一个数组，其元素为起始值范围内一组连续且递增的值。递增间隔为默认值1。
	sequence(x, y, step)	通过指定的起始值返回一个数组，其元素为起始值范围内一组连续且递增的值。自定义递增间隔。
shuffle 函数	shuffle(x)	对数组元素进行随机排列。
slice 函数	slice(x, start, length)	获取数组的子集。
transform 函数	transform(x, lambda_expression)	将 Lambda 表达式应用到数组的每个元素中。
zip 函数	zip(x, y)	将多个数组合并为一个二维数组，且各个数组中下标相同的元素组成一个新的数组。
zip_with 函数	zip_with(x, y, lambda_expression)	根据 Lambda 表达式中的定义将两个数组合并为一个数组。

下标运算符 []

下标运算符用于返回数组中的第 x 个元素。等同与 element_at 函数。

语法

```
[x]
```

参数说明

参数	说明
x	数组下标，从1开始。参数值为 bigint 类型。

返回值类型

返回指定元素的数据类型。

示例

返回 number 字段值中的第2个元素。

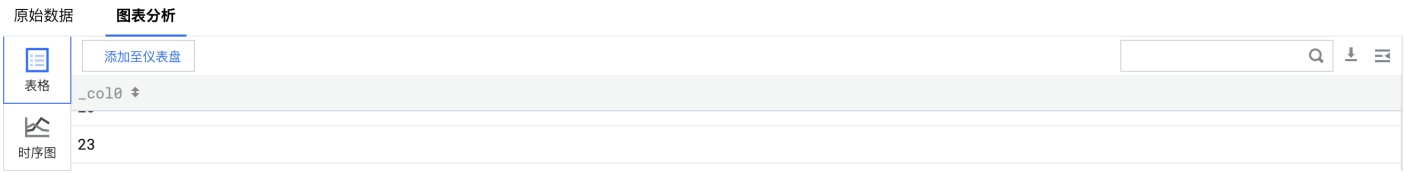
- 字段样例

```
array:[12,23,26,48,26]
```

- 查询和分析语句

```
* | SELECT cast(json_parse(array) as array(bigint)) [2]
```

- 查询和分析结果



array_agg 函数

array_agg 函数会以数组形式返回 x 中所有的值。

语法

```
array_agg(x)
```

参数说明

参数	说明
x	参数值为任意数据类型。

返回值类型

array 类型。

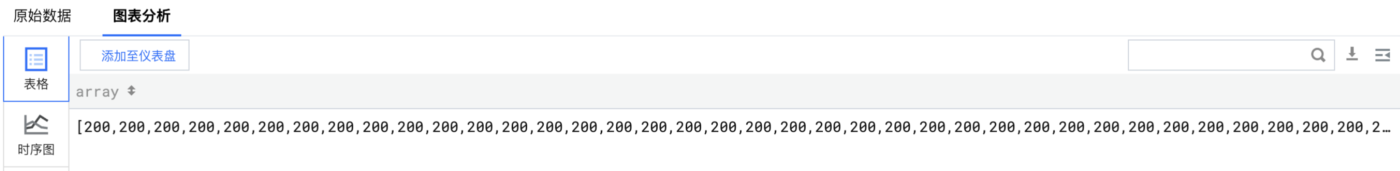
示例

以数组形式返回 **status** 字段的值。

- 查询和分析语句

```
* | SELECT array_agg(status) AS array
```

- 查询和分析结果



array_distinct 函数

array_distinct 函数用于删除数组中重复的元素。

语法

```
array_distinct(x)
```

参数说明

参数	说明
x	参数值为 array 类型。

返回值类型

array 类型。

示例

删除 **array** 字段值中重复的元素。

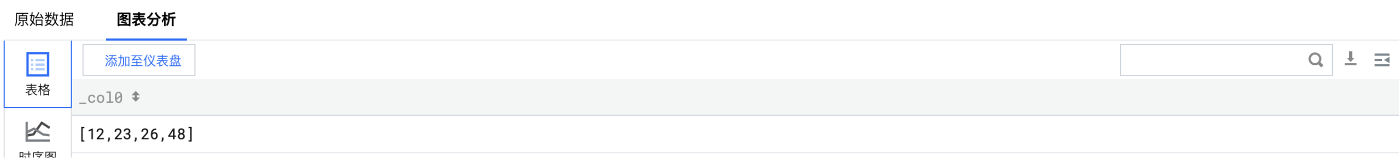
- 字段样例

```
array:[12,23,26,48,26]
```

- 查询和分析语句

```
* | SELECT array_distinct(cast(json_parse(array) as array(bigint)))
```

- 查询和分析结果



array_except 函数

array_except 函数用于计算两个数组的差集。

语法

```
array_except(x, y)
```

参数说明

参数	说明
x	参数值为 array 类型。
y	参数值为 array 类型。

返回值类型

array 类型。

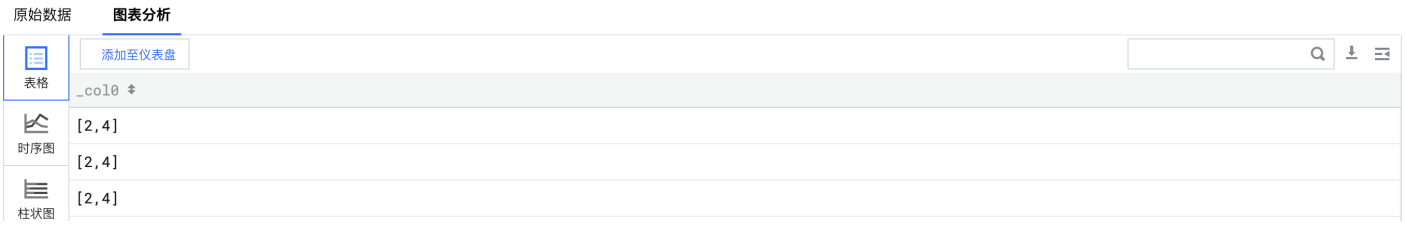
示例

计算数组[1，2，3，4，5]和[1，3，5，7]的差集。

- 查询和分析语句

```
* | SELECT array_except(array[1,2,3,4,5],array[1,3,5,7])
```

- 查询和分析结果



array_intersect 函数

array_intersect 函数用于计算两个数组的交集。

语法

```
array_intersect(x, y)
```

参数说明

参数	说明
x	参数值为 array 类型。
y	参数值为 array 类型。

返回值类型

array 类型。

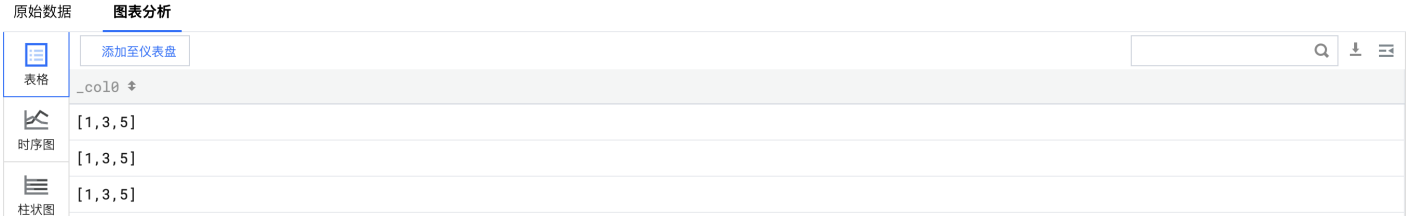
示例

计算数组[1，2，3，4，5]和[1，3，5，7]的差集。

- 查询和分析语句

```
* | SELECT array_intersect(array[1,2,3,4,5],array[1,3,5,7])
```

- 查询和分析结果



array_join 函数

array_join 函数使用指定的连接符将数组中的元素拼接为一个字符串。

语法

- 使用指定的连接符将数组中的元素拼接为一个字符串。如果数组中包含 null 元素，则 null 元素将被忽略。

```
array_join(x, delimiter)
```

- 使用指定的连接符将数组中的元素拼接为一个字符串。如果数组中包含 null 元素，则 null 元素将被替换为 null_replacement。

```
array_join(x, delimiter, null_replacement)
```

参数说明

参数	说明
x	参数值为 array 类型。
delimiter	连接符，可以为字符串。
null_replacement	用于替换 null 元素的字符串。

返回值类型

varchar 类型。

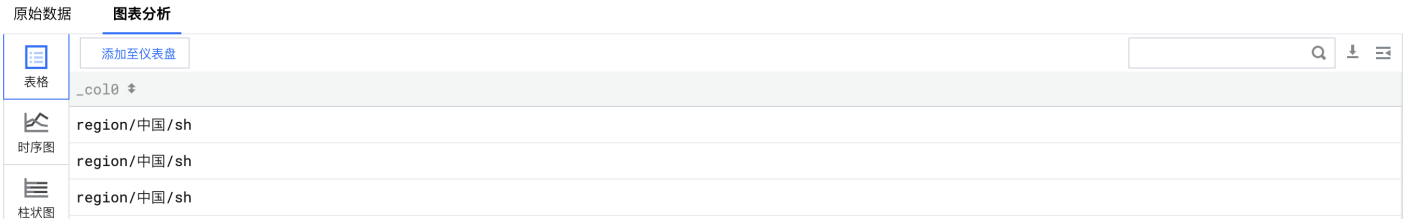
示例

使用空格将数组[null, '中国','sh']中的元素拼接为一个字符串，其中 null 元素替换为 ‘region’ 。

- 查询和分析语句

```
* | SELECT array_join(array[null,'中国','sh'],'/','region')
```

- 查询和分析结果



array_max 函数

array_max 函数用于获取数组中的最大值。

语法

```
array_max(x)
```

参数说明

参数	说明
x	参数值为 array 类型。

返回值类型

与参数值中元素的数据类型一致。

示例

获取数组中[12,23,26,48,26]的最大值48。

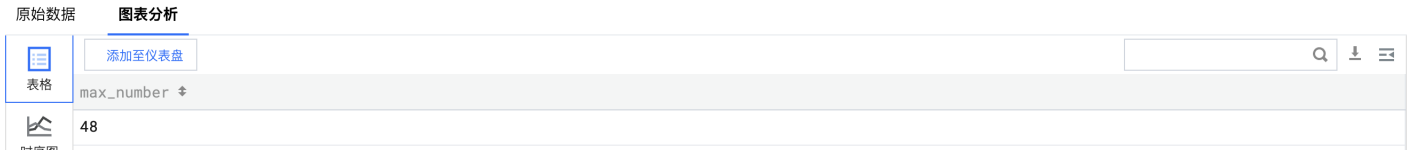
- 字段样例

```
array:[12,23,26,48,26]
```

- 查询和分析语句

```
* | SELECT array_max(try_cast(json_parse(array) as array(bigint))) AS max_number
```

- 查询和分析结果



array_min 函数

array_min 函数用于获取数组中的最小值。

语法

```
array_min(x)
```

参数说明

参数	说明
x	参数值为 array 类型。

返回值类型

与参数值中元素的数据类型一致。

示例

获取数组中[12,23,26,48,26]的最小值12。

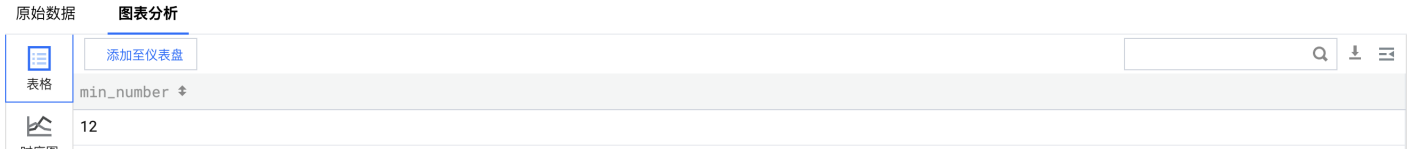
- 字段样例

```
array:[12,23,26,48,26]
```

- 查询和分析语句

```
* | SELECT array_min(try_cast(json_parse(array) as array(bigint))) AS min_number
```

- 查询和分析结果



array_position 函数

array_position 函数用于获取指定元素的下标，下标从1开始。如果指定元素不存在，则返回0。

语法

```
array_position(x, element)
```

参数说明

参数	说明
x	参数值为数组类型。

参数	说明
element	数组中的一个元素。

返回值类型

bigint 类型。

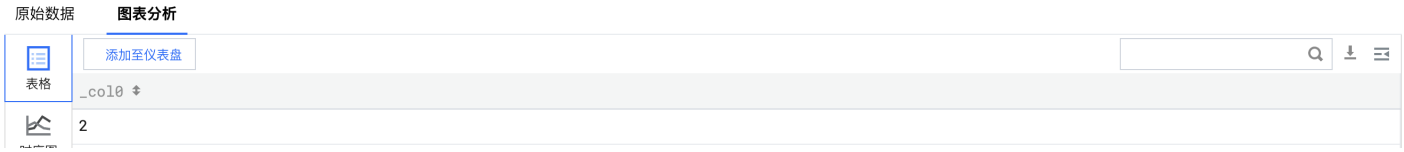
示例

返回数组[23,46,35]中46的下标。

- 查询和分析语句

```
* | SELECT array_position(array[23,46,35],46)
```

- 查询和分析结果



array_remove 函数

array_remove 函数用于删除数组中指定的元素。

语法

```
array_remove(x, element)
```

参数说明

参数	说明
x	参数值为数组类型。
element	数组中的一个元素。

返回值类型

array 类型。

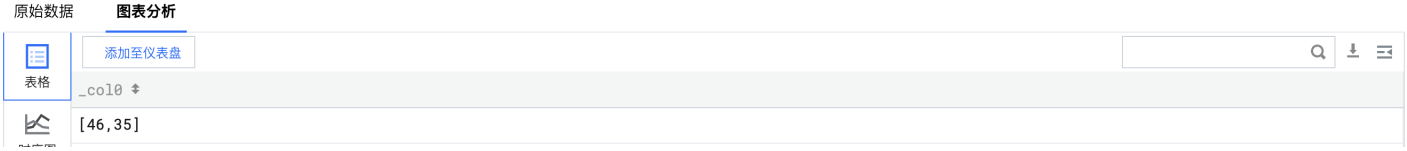
示例

删除数组[23,46,35]中23。

- 查询和分析语句

```
* | SELECT array_remove(array[23,46,35],23)
```

- 查询和分析结果



array_sort 函数

array_sort 函数用于对数组元素进行升序排序。

语法

```
array_sort(x)
```

参数说明

参数	说明
x	参数值为 array 类型。

返回值类型

array 类型。

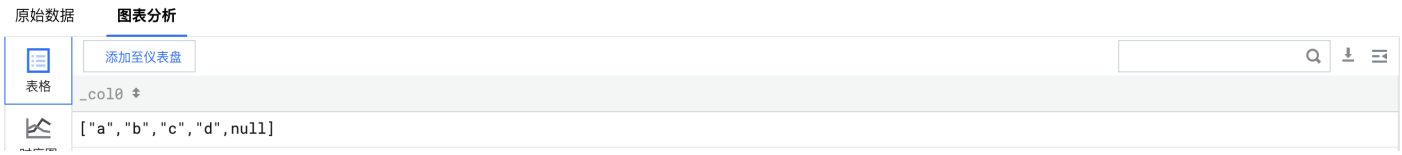
示例

对数组['b', 'd', null, 'c', 'a']进行升序排序。

- 查询和分析语句

```
* | SELECT array_sort(array['b','d',null,'c','a'])
```

- 查询和分析结果



array_union 函数

array_uinon 函数用于计算两个数组的并集。

语法

```
array_union(x, y)
```

参数说明

参数	说明
x	参数值为 array 类型。
y	参数值为 array 类型。

返回值类型

array 类型。

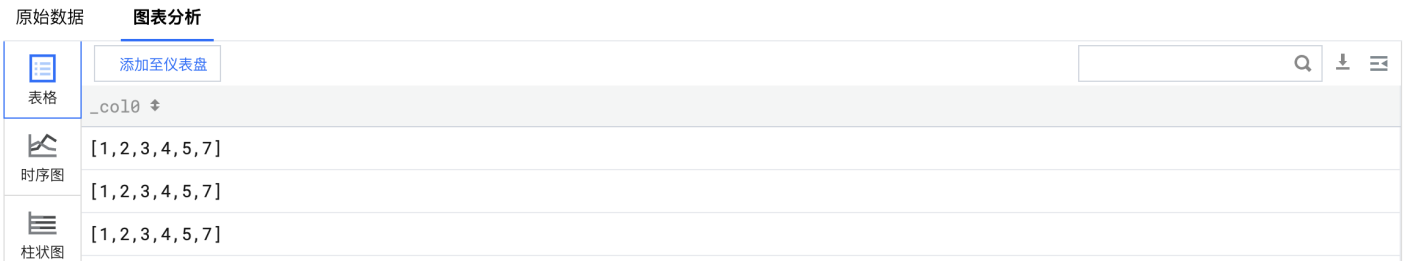
示例

计算数组[1,2,3,4,5]和[1,3,5,7]的并集。

- 查询和分析语句

```
* | SELECT array_union(array[1,2,3,4,5],array[1,3,5,7])
```

- 查询和分析结果



cardinality 函数

cardinality 函数用于计算数组中元素的个数。

语法

```
cardinality(x)
```

参数说明

参数	说明
x	参数值为 array 类型。

返回值类型

bigint 类型。

示例

计算 number 字段值中元素的个数。

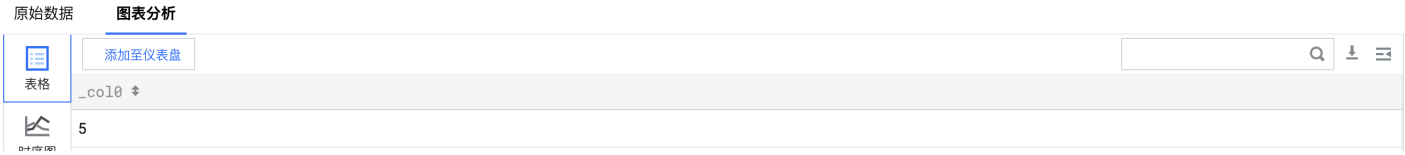
- 字段样例

```
array:[12,23,26,48,26]
```

- 查询和分析语句

```
* | SELECT cardinality(cast(json_parse(array) as array(bigint)))
```

- 查询和分析结果



concat 函数

concat 函数用于将多个数组拼接成一个数组。

语法

```
concat(x, y...)
```

参数说明

参数	说明
x	参数值为 array 类型。

参数	说明
y	参数值为 array 类型。

返回值类型

array 类型。

示例

将数组['red','blue']和['yellow','green']拼接为一个数组。

- 查询和分析语句

```
* | SELECT concat(array['red','blue'],array['yellow','green'])
```

- 查询和分析结果

contains 函数

contains 函数用于判断数组中是否包含指定元素。如果包含，则返回 true。

语法

```
contains(x, element)
```

参数说明

参数	说明
x	参数值为数组类型。
element	数组中的一个元素。

返回值类型

boolean 类型。

示例

判断 **array** 字段值中是否包含23。

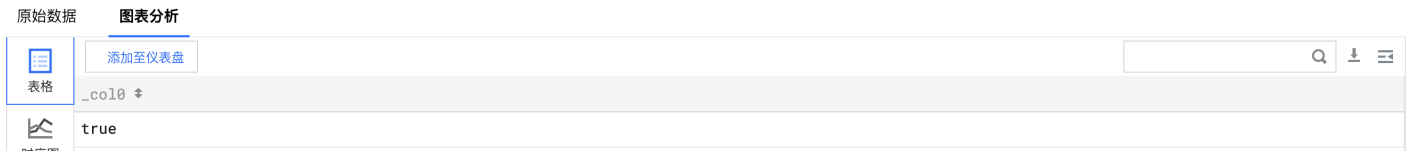
- 字段样例

```
array:[12,23,26,48,26]
```

- 查询和分析语句

```
* | SELECT contains(cast(json_parse(array) as array(varchar)),'23')
```

- 查询和分析结果



element_at 函数

element_at 函数用于返回数组中的第 y 个元素。

语法

```
element_at(x, y)
```

参数说明

参数	说明
x	参数值为 array 类型。
element	数组下标，从1开始。参数值为 bigint 类型。

返回值类型

任意数据类型。

示例

返回 number 字段值中的第2个元素。

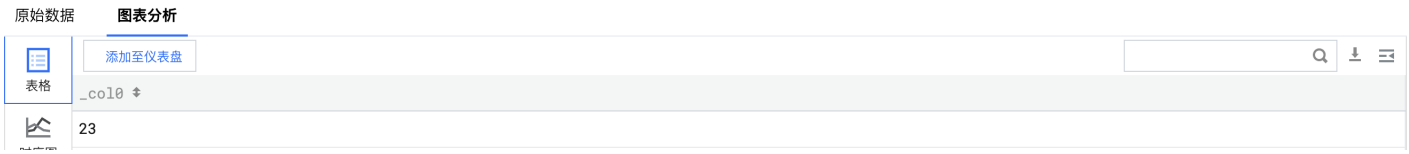
- 字段样例

```
array:[12,23,26,48,26]
```

- 查询和分析语句

```
* | SELECT element_at(cast(json_parse(number) AS array(varchar)), 2)
```

- 查询和分析结果



filter 函数

filter 函数和 Lambda 表达式结合，用于过滤数组中的元素。只返回满足 Lambda 表达式的元素。

语法

```
filter(x, lambda_expression)
```

参数说明

参数	说明
x	参数值为 array 类型。
lambda_expression	Lambda 表达式。更多信息，请参见 Lambda 函数。

返回值类型

array 类型。

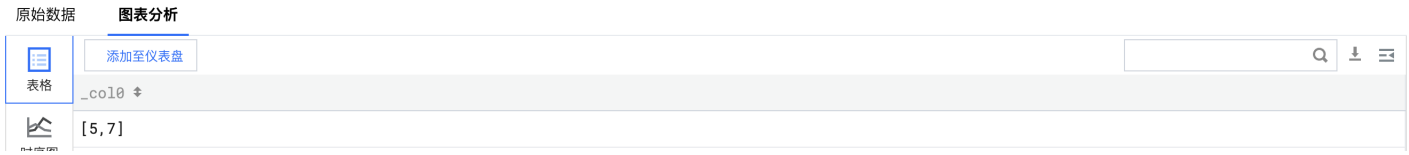
示例

返回数组[5,-6,null,7]中大于0的元素，其中 x -> x > 0 为 Lambda 表达式。

- 查询和分析语句

```
* | SELECT filter(array[5,-6,null,7],x -> x > 0)
```

- 查询和分析结果



flatten 函数

flatten 函数用于将二维数组转换为一维数组。

语法

```
flatten(x)
```

参数说明

参数	说明
x	参数值为 array 类型。

返回值类型

array 类型。

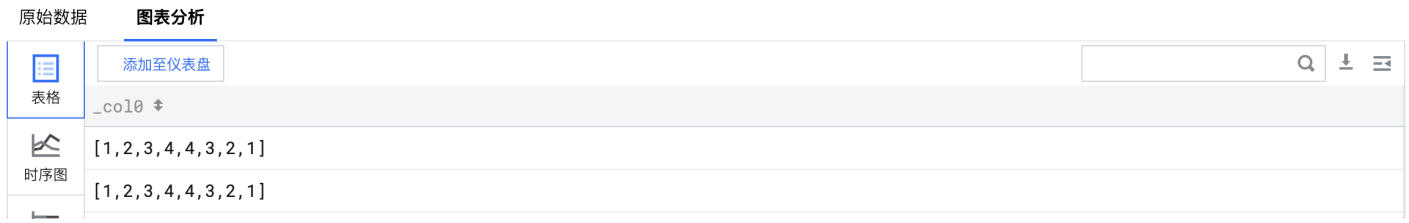
示例

将数组array[1,2,3,4],array[4,3,2,1]转换为一维数组。

- 查询和分析语句

```
* | SELECT flatten(array[array[1,2,3,4],array[4,3,2,1]])
```

- 查询和分析结果



reduce 函数

reduce 函数将根据 Lambda 表达式中的定义，对数组中的各个元素进行相加计算，然后返回计算结果。

语法

```
reduce(x, lambda_expression)
```

参数说明

参数	说明
x	参数值为 array 类型。
lambda_expression	Lambda 表达式。更多信息，请参见 Lambda 函数。

返回值类型

bigint 类型。

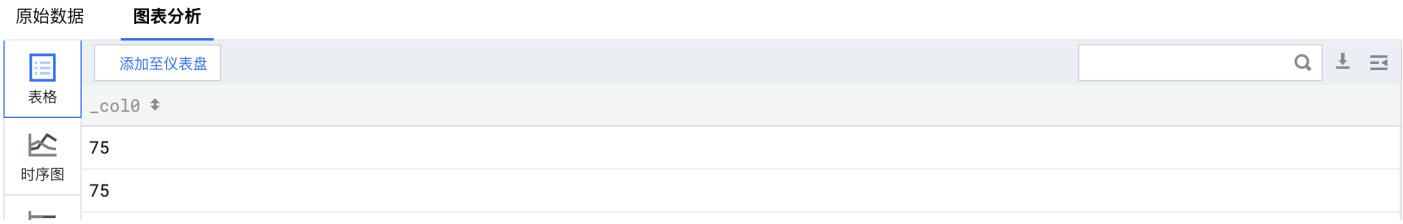
示例

返回数组[5, 20, 50]中各个元素相加的结果。

- 查询和分析语句

```
* | SELECT reduce(array[5,20,50],0,(s, x) -> s + x, s -> s)
```

- 查询和分析结果



reverse 函数

reverse 函数用于对数组中的元素进行反向排列。

语法

```
reverse(x)
```

参数说明

参数	说明
x	参数值为 array 类型。

返回值类型

array 类型。

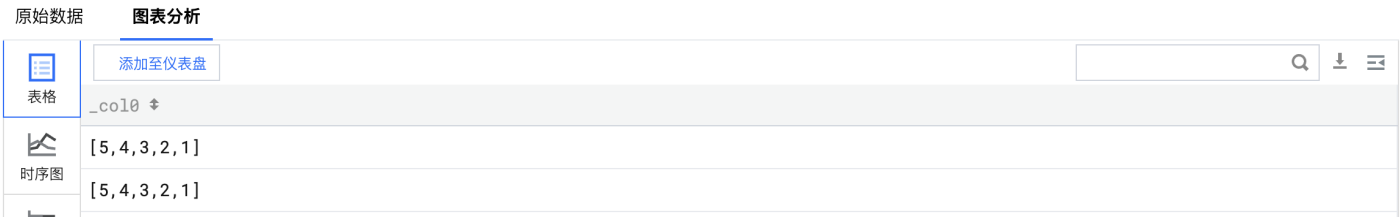
示例

将数组[1,2,3,4,5]中的元素反向排序。

- 查询和分析语句

```
* | SELECT reverse(array[1,2,3,4,5])
```

- 查询和分析结果



sequence 函数

sequence 函数通过指定的起始值返回一个数组，其元素为起始值范围内一组连续且递增的值。

语法

- 递增间隔为默认值1。

```
sequence(x, y)
```

- 自定义递增间隔。

```
sequence(x, y, step)
```

参数说明

参数	说明
x	参数值为 bigint 类型、timestamp 类型（Unix 时间戳、日期和时间表达式）。
y	参数值为 bigint 类型、timestamp 类型（Unix 时间戳、日期和时间表达式）。
step	数值间隔。 当参数值为日期和时间表达式时，step 格式如下： – interval 'n' year to month，表示间隔为 n 年。 – interval 'n' day to second，表示间隔为 n 天。

返回值类型

array 类型。

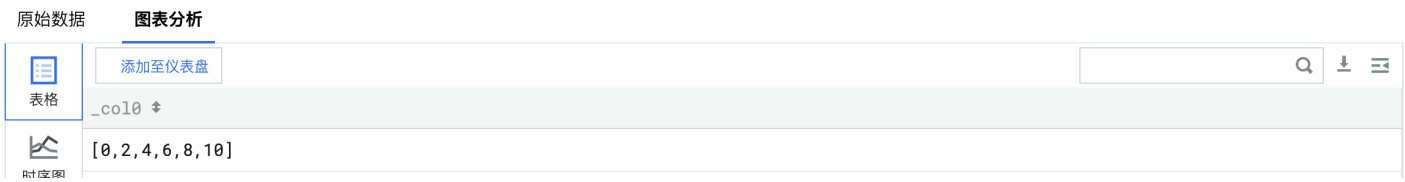
示例

示例1：返回0 – 10之间的偶数。

- **查询和分析语句**

```
* | SELECT sequence(0,10,2)
```

- **查询和分析结果**

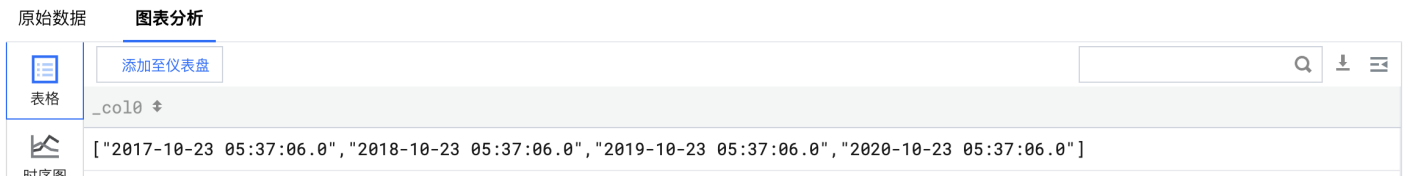


示例2：返回2017-10-23到2021-08-12之间的日期，间隔为1年。

- **查询和分析语句**

```
* | SELECT sequence(from_unixtime(1508737026),from_unixtime(1628734085),interval '1' year to month )
```

- **查询和分析结果**

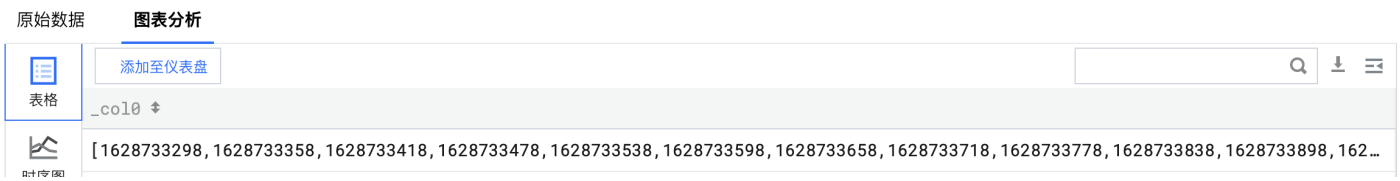


示例3：返回1628733298,1628734085之间的 Unix 时间戳，间隔为60秒。

- **查询和分析语句**

```
* | SELECT sequence(1628733298,1628734085,60)
```

- **查询和分析结果**



shuffle 函数

shuffle 函数用于对数组元素进行随机排列。

语法

```
shuffle(x)
```

参数说明

参数	说明
x	参数值为 array 类型。

返回值类型

array 类型。

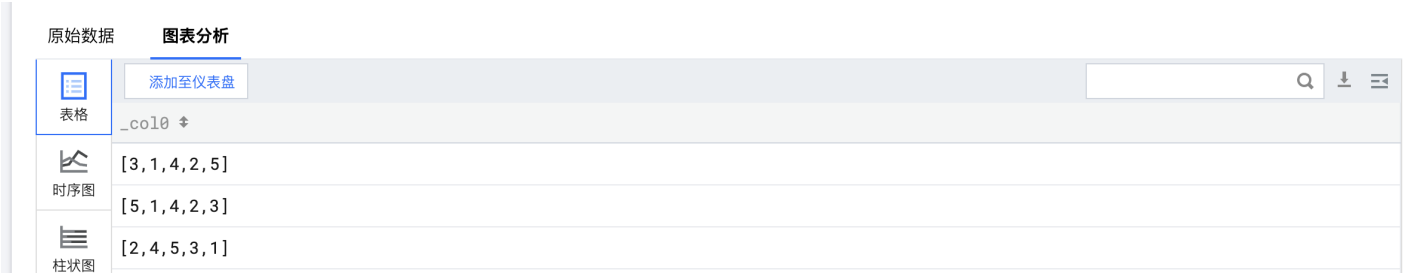
示例

对数组[1,2,3,4,5]中的元素进行随机排序。

- 查询和分析语句

```
* | SELECT shuffle(array[1,2,3,4,5])
```

- 查询和分析结果



slice 函数

slice 函数用于返回数组的子集。

语法

```
slice(x, start, length)
```

参数说明

参数	说明
x	参数值为 array 类型。
start	指定索引开始的位置。 – 如果 start 为负数，则从末尾开始。 – 如果 start 为正数，则从头部开始。
length	指定子集中元素的个数。

返回值类型

array 类型。

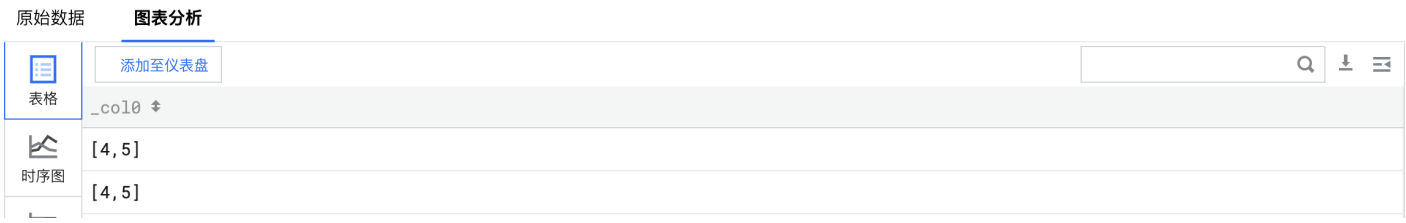
示例

返回数组[1,2,4,5,6,7,7]的子集，从第三个元素开始返回，子集元素个数为2。

- 查询和分析语句

```
* | SELECT slice(array[1,2,4,5,6,7,7],3,2)
```

- 查询和分析结果



transform 函数

transform 函数用于将 Lambda 表达式应用到数组的每个元素中。

语法

```
transform(x, lambda_expression)
```

参数说明

参数	说明
x	参数值为 array 类型。
lambda_expression	Lambda 表达式。更多信息，请参见 Lambda 函数。

返回值类型

array 类型。

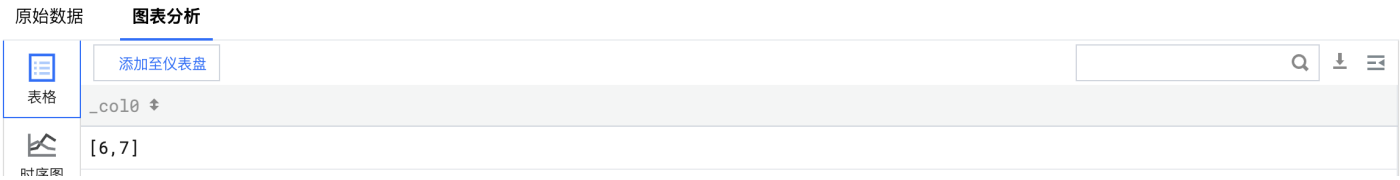
示例

将数组[5,6]中的各个元素加1，然后返回。

- 查询和分析语句

```
* | SELECT transform(array[5,6],x -> x + 1)
```

- 查询和分析结果



zip 函数

zip 函数将多个数组合并为一个二维数组，且各个数组中下标相同的元素组成一个新的数组。

语法

```
zip(x, y)
```

参数说明

参数	说明
x	参数值为 array 类型。

参数	说明
y	参数值为 array 类型。

返回值类型

array 类型。

示例

将数组[1,2]和[3,4]合并为一个二维数组。

- 查询和分析语句

```
* | SELECT zip(array[1,2], array[3,4])
```

- 查询和分析结果

```
["{1, 3}","{2, 4}"]
```

zip_with 函数

zip_with 函数将根据 Lambda 表达式中的定义将两个数组合并为一个数组。

语法

```
zip_with(x, y, lambda_expression)
```

参数说明

参数	说明
x	参数值为 array 类型。
y	参数值为 array 类型。
lambda_expression	Lambda 表达式。更多信息，请参见 Lambda 函数。

返回值类型

array 类型。

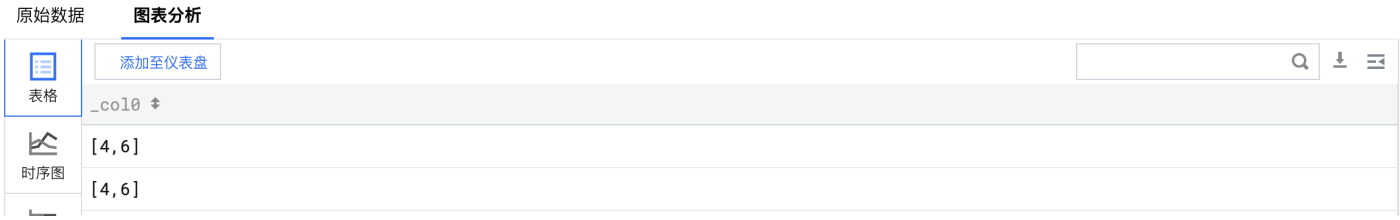
示例

使用 Lambda 表达式 (x, y) -> x + y 使数组[1,2]和[3,4]中的元素分别相加后，以数组类型返回相加的结果。

- 查询和分析语句

```
* | SELECT zip_with(array[1,2], array[3,4],(x,y) -> x + y)
```

- 查询和分析结果



同环比函数

最近更新时间：2021-12-10 16:26:34

本文介绍同环比函数的基础语法和示例。

日志服务（Cloud Log Service，CLS）支持如下同环比函数。

函数名称	语句	含义
compare 函数	compare(x,n)	对比当前时间周期内的计算结果与n秒之前时间周期内的计算结果。
	compare(x,n1,n2,n3...)	对比当前时间周期内的计算结果与n1秒、n2秒、n3秒之前时间周期内的计算结果。

compare 函数

compare 函数用于对比当前时间周期内的计算结果与 n 秒之前时间周期内的计算结果。

语法

- 对比当前时间周期内的计算结果与n秒之前时间周期内的计算结果。

```
compare ( x, n )
```

- 对比当前时间周期内的计算结果与n1、n2、n3秒之前时间周期内的计算结果。

```
compare ( x, n1, n2, n3... )
```

参数说明

参数	说明
x	参数值为 double 类型或 long 类型。
n	时间窗口，单位为秒。例如3600（1小时）、86400（1天）、604800（1周）、31622400（1年）。

返回值类型

JSON 数组。格式为[当前计算结果，n 秒前的计算结果，当前计算结果与 n 秒前计算结果的比值]。

示例

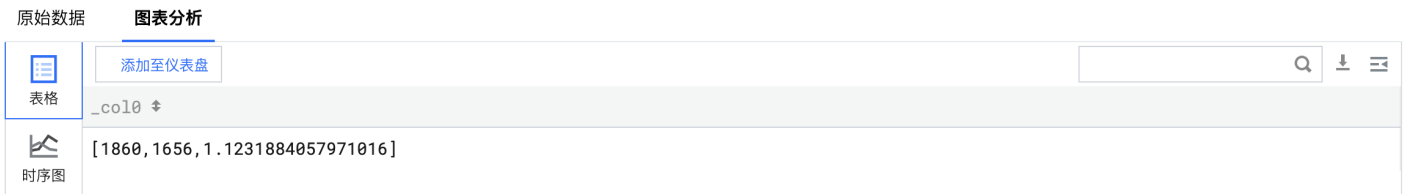
示例1：计算当前1小时和昨天同时段的网站访问量比值

选择查询和分析的时间范围为近1小时，并执行如下查询和分析语句，其中86400表示当前时间减去86400秒（1天）。

- 查询和分析语句

```
* | SELECT compare(PV, 86400) FROM (SELECT count(*) AS PV)
```

查询和分析结果

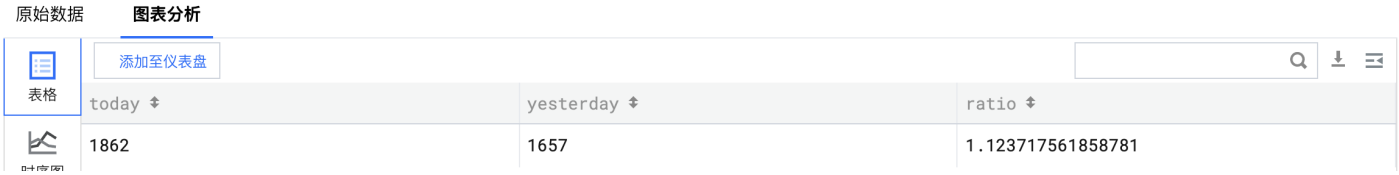


  - 1860表示当前1小时的网站访问量。
  - 1656表示昨天同时段的网站访问量。
  - 1.1231884057971016表示当前1小时与昨天同时段的网站访问量比值。
- 如需查询和分析结果为分列显示，则查询和分析语句为：

```
* |
SELECT compare[1] AS today, compare[2] AS yesterday, compare[3] AS ratio
```

```
FROM (
SELECT compare(PV, 86400) AS compare
FROM (
SELECT COUNT(*) AS PV
)
)
```

查询和分析结果



示例2：计算今天每5分钟网站访问量变化趋势与昨天同时间段的对比

选择查询和分析的时间范围为今天，并执行如下查询和分析语句，其中86400表示当前时间减去86400秒（1天），current_date 表示今天的日期。

> **注意：**
> 时间范围不能跨天，因为示例中已经将日志时间裁剪为%H:%i:%s，仅包含时、分、秒，未包含日期，跨天会导致数据统计错误。

- 查询和分析语句

```
* |
select concat(cast(current_date as varchar),' ',time) as time,compare[1] as today,compare[2] as yesterday from (
select time,compare(pv, 86400) as compare from (
select time_series(__TIMESTAMP__, '5m', '%H:%i:%s', '0') as time, count(*) as pv group by time limit 1000)
limit 1000)
order by time limit 1000
```

- 查询和分析结果

JSON 函数

最近更新时间：2022-03-04 14:30:47

本文介绍 JSON 函数的基本语法及示例。

函数名称	语句	含义
json_array_contains 函数	json_array_contain(x，value)	判断 JSON 数组中是否包含某个值。
json_array_get 函数	json_array_get(x，index)	获取 JSON 数组中某个下标对应的元素。
json_array_length 函数	json_array_length(x)	计算 JSON 数组中元素的数量。
json_extract 函数	json_extract(x，json_path)	从 JSON 对象或 JSON 数组中提取一组 JSON 值（数组或对象）。
json_extract_scalar 函数	json_extract_scalar(x，json_path)	从 JSON 对象或 JSON 数组中提取一组标量值（字符串、整数或布尔值）。类似于 json_extract 函数。
json_format 函数	json_format(x)	把 JSON 类型转化成字符串类型。
json_parse 函数	json_parse(x)	把字符串类型转化成 JSON 类型。
json_size 函数	json_size(x，json_path)	计算 JSON 对象或数组中元素的数量。

json_array_contains 函数

json_array_contains 函数用于判断 JSON 数组中是否包含某个值。

语法

```
json_array_contains(x, value)
```

参数说明

参数	说明
x	参数值为 JSON 数组。
value	数值。

返回值类型

boolean 类型。

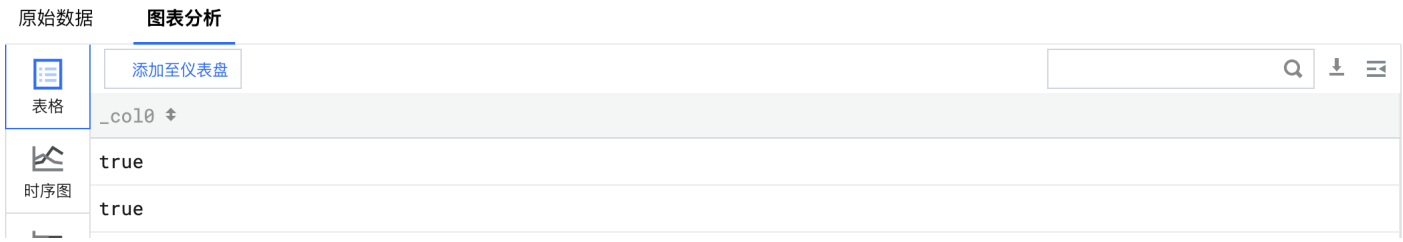
示例

判断 JSON 字符[1，2，3]中是否包含2。

- 查询和分析语句

```
* | SELECT json_array_contains('[1, 2, 3]', 2)
```

- 查询和分析结果



json_array_get 函数

json_array_get 函数用于获取 JSON 数组中某个下标对应的元素。

语法

```
json_array_get(x, index)
```

参数说明

参数	说明
x	参数值为 JSON 数组。
index	JSON 下标，从0开始。

返回值类型

varchar类型。

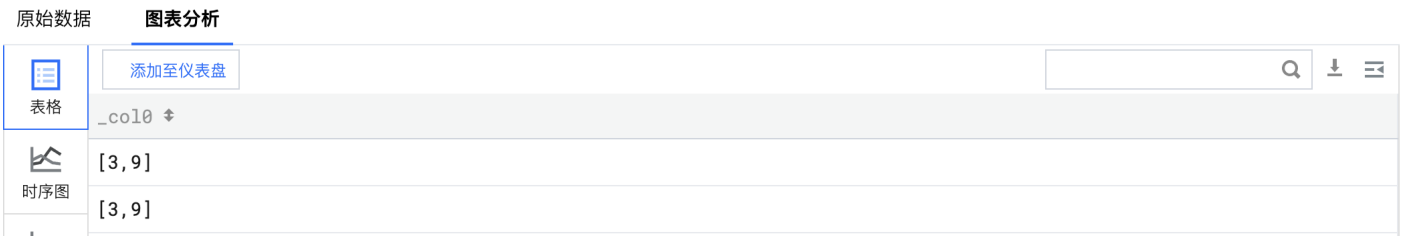
示例

返回 JSON 数组["a", [3, 9], "c"]下标为1的元素。

- 查询和分析语句

```
* | SELECT json_array_get('["a", [3, 9], "c"]', 1)
```

- 查询和分析结果



json_array_length 函数

json_array_length 函数用于计算 JSON 数组中元素的数量。

语法

```
json_array_length(x)
```

参数说明

参数	说明
x	参数值为 JSON 数组。

返回值类型

bigint 类型。

示例

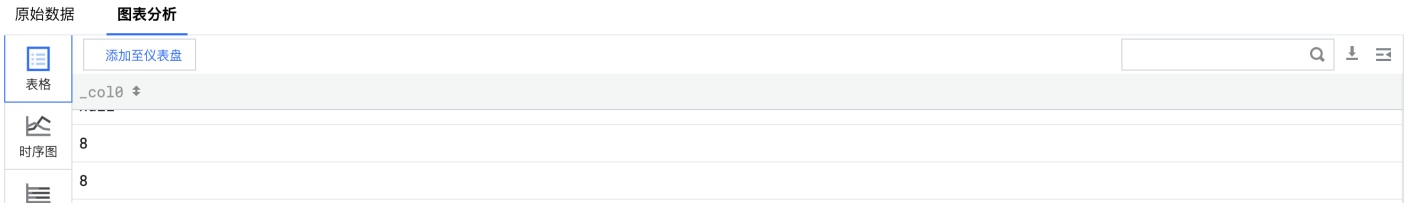
- 示例1：计算 apple.message 字段值中 JSON 元素的数量。
  - 字段样例

```
apple.message:[{"traceName":"StoreMonitor"},{"topicName":"persistent://apache/pulsar/test-partition-17"},{"producerName":"pulsar-mini-338-36"},{"localAddr":"pulsar://pulsar-mini-broker-5.pulsar-mini-broker.pulsar.svc.cluster.local:6650"},{"sequenceId":826},{"storeTime":1635905306062},{"messageId":"19422-24519"},{"status":"SUCCESS"}]
```

- 查询和分析语句

```
* | SELECT json_array_length(apple.message)
```

- 查询和分析结果



json_extract 函数

json_extract 函数用于从 JSON 对象或 JSON 数组中提取一组 JSON 值（数组或对象）。

语法

```
json_extract(x, json_path)
```

参数说明

参数	说明
x	参数值为 JSON 对象或 JSON 数组。
json_path	JSONPath 形式的 JSON 路径，例如 $.store.book[0].title。 注意：不支持需要遍历数组元素的 JSON 语法，例如 $.store.book[*].author、$..book[(@.length-1)]、$..book[?(@.price<10)] 等。

返回值类型

JSON 格式的 string 类型。

示例

获取 apple.instant 字段中 epochSecond 的值。

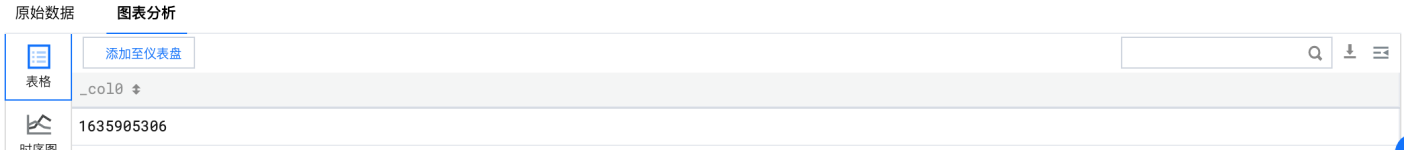
- 字段样例

```
apple.instant:{"epochSecond":1635905306,"nanoOfSecond":63001000}
```

- 查询和分析语句

```
* | SELECT json_extract(apple.instant, '$.epochSecond')
```

- 查询和分析结果



json_extract_scalar 函数

json_extract_scalar 函数用于从 JSON 对象或 JSON 数组中提取一组标量值（字符串、整数或布尔值）。

语法

```
json_extract_scalar(x, json_path)
```

参数说明

参数	说明
x	参数值为 JSON 数组。
json_path	JSONPath 形式的 JSON 路径，例如 $.store.book[0].title。 注意：不支持需要遍历数组元素的 JSON 语法，例如 $.store.book[*].author、$..book[(@.length-1)]、$..book[?(@.price<10)] 等。

返回值类型

varchar 类型。

示例

从 apple.instant 字段中获取 epochSecond 字段的值，并将该值转换为 bigint 类型进行求和。

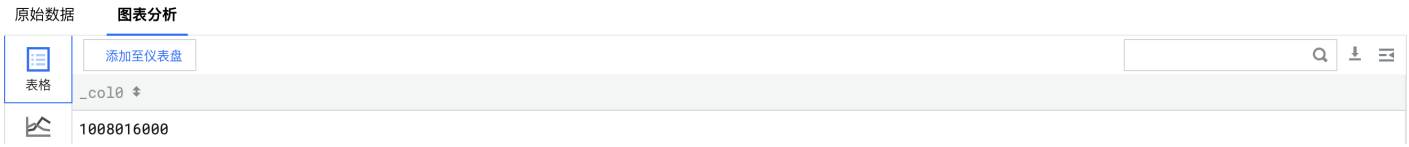
- 字段样例

```
apple.instant:{"epochSecond":1635905306,"nanoOfSecond":63001000}
```

- 查询和分析语句

```
* | SELECT sum(cast(json_extract_scalar(apple.instant,'$.epochSecond') AS bigint) )
```

- 查询和分析结果



json_format 函数

json_format 函数用于将 JSON 类型转化成字符串类型。

语法

```
json_format(x)
```

参数说明

参数	说明
x	参数值为 JSON 类型。

返回值类型

varchar 类型。

示例

将 JSON 数组[1,2,3]转换为字符串[1, 2, 3]。

- 查询和分析语句

```
* | SELECT json_format(json_parse('[1, 2, 3]'))
```

- 查询和分析结果

json_parse 函数

json_parse 函数只用于将字符串类型转化成 JSON 类型，判断是否符合 JSON 格式。

语法

```
json_parse(x)
```

参数说明

参数	说明
x	参数值为字符串。

返回值类型

JSON类型。

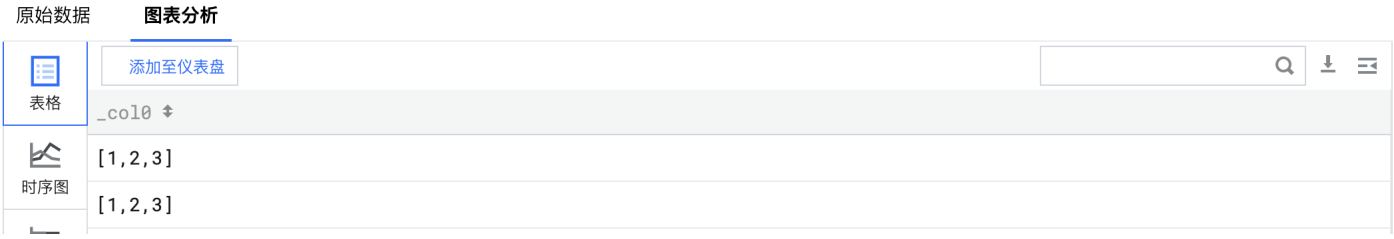
示例

将 JSON 数组[1,2,3]转换为字符串[1, 2, 3]。

- 查询和分析语句

```
* | SELECT json_parse('[1, 2, 3]')
```

- 查询和分析结果



json_size 函数

json_size 函数用于计算 JSON 对象或 JSON 数组中元素的数量。

语法

```
json_size(x, json_path)
```

参数说明

参数	说明
x	参数值为 JSON 对象或 JSON 数组。
json_path	JSON 路径，格式为 $.store.book[0].title。

返回值类型

bigint 类型。

示例

将 JSON 数组[1,2,3]转换为字符串[1, 2, 3]。

- 查询和分析语句

```
* | SELECT json_size(json_parse('[1, 2, 3]'))
```

- 查询和分析结果
  

窗口函数

最近更新时间：2022-02-21 10:28:17

本文介绍窗口函数的基础语法和示例。

窗口函数用于对指定的多行数据进行计算并返回计算的结果，与 GROUP BY 的区别在于其仅会将计算结果附加到每一行数据上，而不会对行本身进行合并。

语法

```
window_function (expression) OVER (
[ PARTITION BY part_key ]
[ ORDER BY order_key ]
[ { ROWS | RANGE } BETWEEN frame_start AND frame_end ] )
```

参数说明

参数	说明
window_function	窗口值计算方法，支持通用聚合函数、排序函数和取值函数
PARTITION BY	窗口分区依据
ORDER BY	窗口分区内多行数据排序依据
{ ROWS | RANGE } BETWEEN frame_start AND frame_end	窗口帧，即窗口分区内每行数据计算值时使用到的数据范围（行），未指定时代表窗口分区内的所有行 使用示例： rows between current row and 1 following：当前行及后一行 rows between 1 preceding and current row：当前行及前一行 rows between 1 preceding and 1 following：前一行至后一行（共三行） rows between current row and unbounded following：当前行及后续所有行 rows between unbounded preceding and current row：当前行及前面的所有行

通用聚合函数

支持所有的 通用聚合函数，例如 sum()，avg()等，通用聚合函数将针对每行数据计算其在窗口帧内的统计值。

排序函数

排序函数不能使用窗口帧

函数	说明
rank()	对每个窗口分区分别进行排名并返回值。相同的值拥有相同的排名，所以排名可能不是连续的。例如，有两个相同值的排名为1，则下一个值的排名为3。
dense_rank()	与 rank() 类似，区别在于该函数排名是连续的。例如，有两个相同值的排名为1，则下一个值的排名为2。
cume_dist()	统计窗口分区内各个值的累计分布，即窗口分区内值小于等于当前值的行数占窗口内总行数的比例。
ntile(n)	将窗口分区内数据按照顺序分为n组，如果分区内的行数没有被平均分成n组，则从第一组开始，每组分配一个剩余值。例如，数据有6行，需要分为4组，则每行数据的组号为 1、1、2、2、3、4。
percent_rank()	计算每行数据在窗口分区内的百分比排名，计算方式为(r − 1) / (n − 1)，其中r为通过 rank() 获取到排名值，n为窗口分区内的总行数
row_number()	返回窗口分区内每行数据（根据排序规则排序后）的序号，从1开始，每行均为唯一值。

取值函数

函数	说明
first_value(key)	返回窗口分区内的第一个值。
last_value(key)	返回窗口分区内最后一个值。
nth_value(key, offset)	返回窗口分区内指定偏移后的值，offset 从1开始计算。如果 offset 为 null 或超过了窗口分区内行的数量，则返回null。offset 不允许为0或负数。
lead(key[, offset[, default_value]])	返回窗口分区内当前行向后指定偏移后的值，offset 从0开始计算，即当前行。offset 默认为1，如果 offset 为 null，则返回 null。如果偏移后的行超出了窗口分区，则返回 default_value，未指定时返回 null。 使用该函数时必须制定窗口分区内的排序规则（ORDER BY），并且不能使用窗口帧。

函数	说明
lag(key[, offset[, default_value]])	与 lead(key[, offset[, default_value]]) 函数类似，区别在于该函数会返回当前行向前指定偏移后的值。

示例

示例1：查询最近1小时每个接口最慢的5个请求及其请求 ID

选择时间范围为近1小时，并执行如下查询和分析语句，其中 action 为接口名称，timeCost 为接口响应时间，seqId 为请求ID。

- 查询和分析语句

```
* | select * from (select action,timeCost,seqId,rank() over (partition by action order by timeCost desc) as ranking order by action,ranking,seqId) where ranking<=5 limit 10000
```

- 查询和分析结果

action	timeCost	seqId	ranking
ModifyXXX	151	d75427b3-c562-6d7a-354f-469963aab689	1
ModifyXXX	104	add0d353-1099-2c73-e9c9-19ad02480474	2
CreateXXX	1254	c7d591f0-2da6-292c-8abf-98a0716ff8c6	1
CreateXXX	970	d920cf7a-7e7b-524b-68e9-a957c454c328	2
CreateXXX	812	16357f6d-33b3-83ea-0ae3-b1a2233d4858	3
CreateXXX	795	0efdab5e-af5f-4a4a-0618-7961420d17a1	4
CreateXXX	724	fb0481f2-dcfc-9500-cb44-a139b774aceb	5
DescribeXXX	55242	4129dcda-46d7-9213-510e-f58cba29daf5	1
DescribeXXX	17413	e36cdeb0-cbc5-ce2b-dec7-f485818ab6c7	2
DescribeXXX	10171	cd6228f7-4644-ba45-f539-0fce7b09455b	3
DescribeXXX	9475	48b6f6e3-6d08-5a31-cd68-89006a346497	4
DescribeXXX	9337	940b5398-e2ae-9141-801b-b7f0ca548875	5

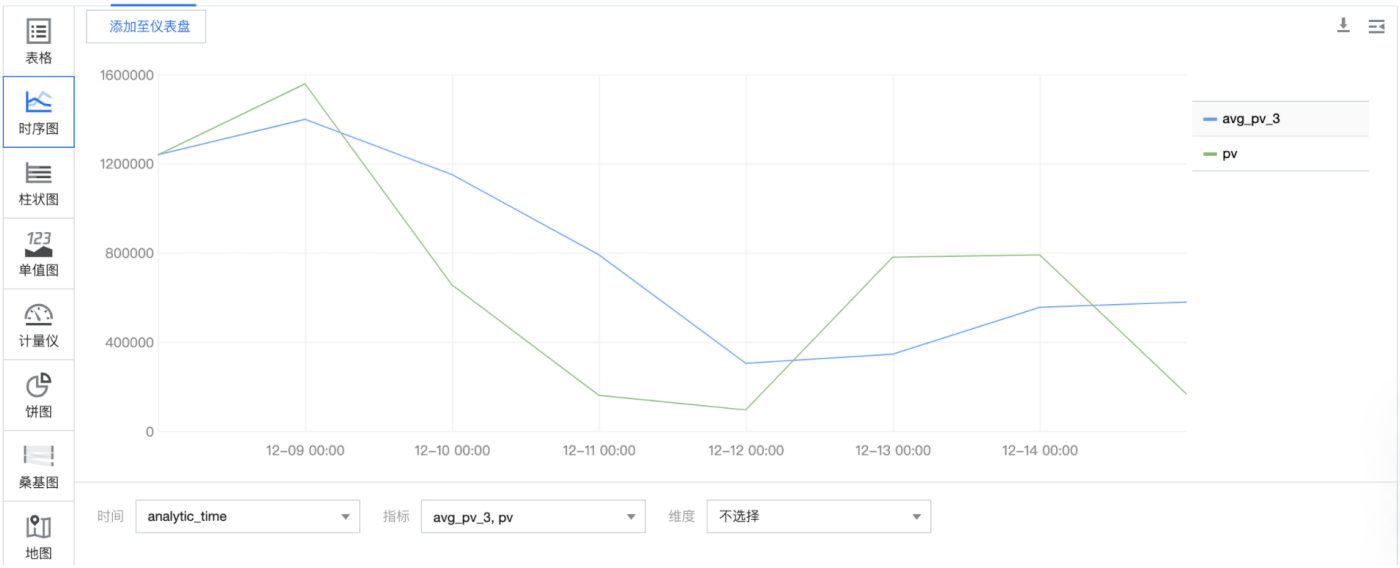
示例2：查询应用吞吐量的3日移动平均值变化趋势

选择查询和分析的时间范围为最近7天，并执行如下查询和分析语句，其中 pv 为每天的应用吞吐量，avg_pv_3 是3天移动平均后的应用吞吐量。

- 查询和分析语句

```
* | select avg(pv) over(order by analytic_time rows between 2 preceding and current row) as avg_pv_3,pv,analytic_time from (select histogram(cast(__TIMESTAMP__ as timestamp),interval 1 day) as analytic_time, count(*) as pv group by analytic_time order by analytic_time)
```

- **查询和分析结果**


添加至仪表盘
表格
时序图
柱状图
单值图
计量仪
饼图
桑基图
地图
1600000
1200000
800000
400000
0
12-09 00:00
12-10 00:00
12-11 00:00
12-12 00:00
12-13 00:00
12-14 00:00
avg_pv_3
pv
时间
analytic_time
指标
avg_pv_3, pv
维度
不选择

快速分析

最近更新时间：2022-04-20 17:50:05

操作场景

日志服务提供了针对字段的快速统计分析功能，无需编写查询语句即可快速分析字段值的分布、随时间变化趋势和数值统计，例如统计 Top5请求接口、接口响应时间变化趋势等。

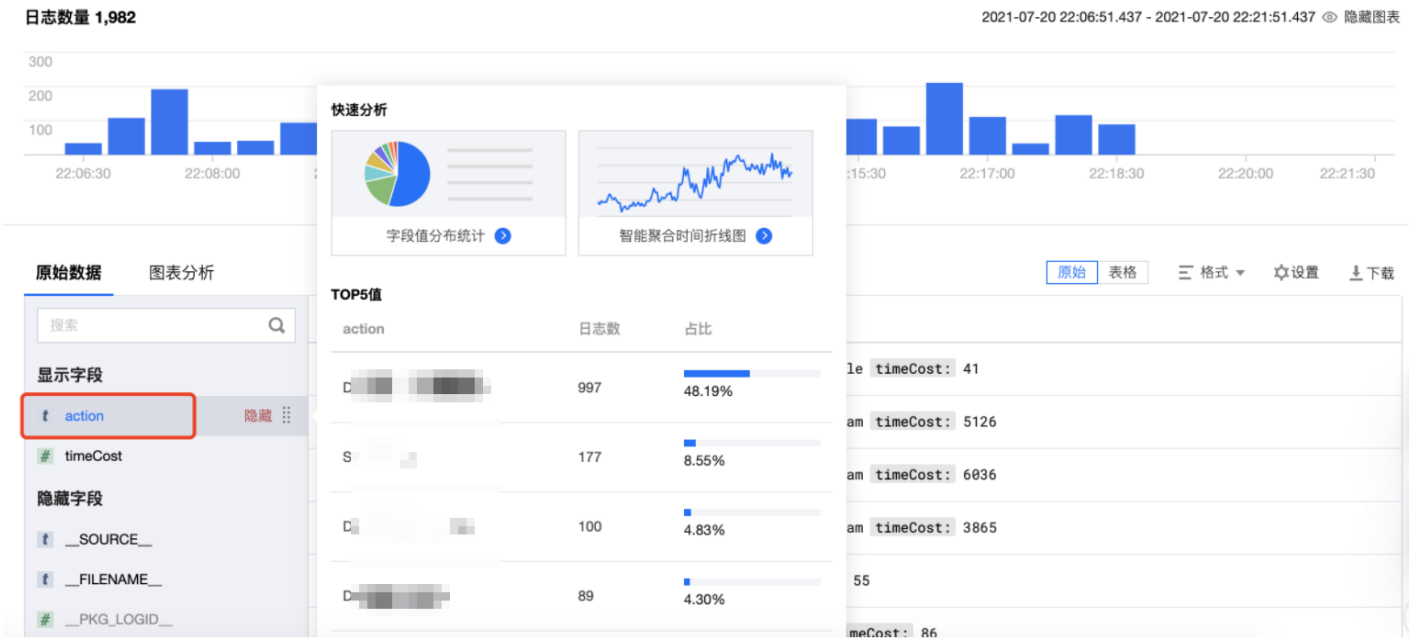
前提条件

已在索引配置中开启了统计的字段。

操作步骤

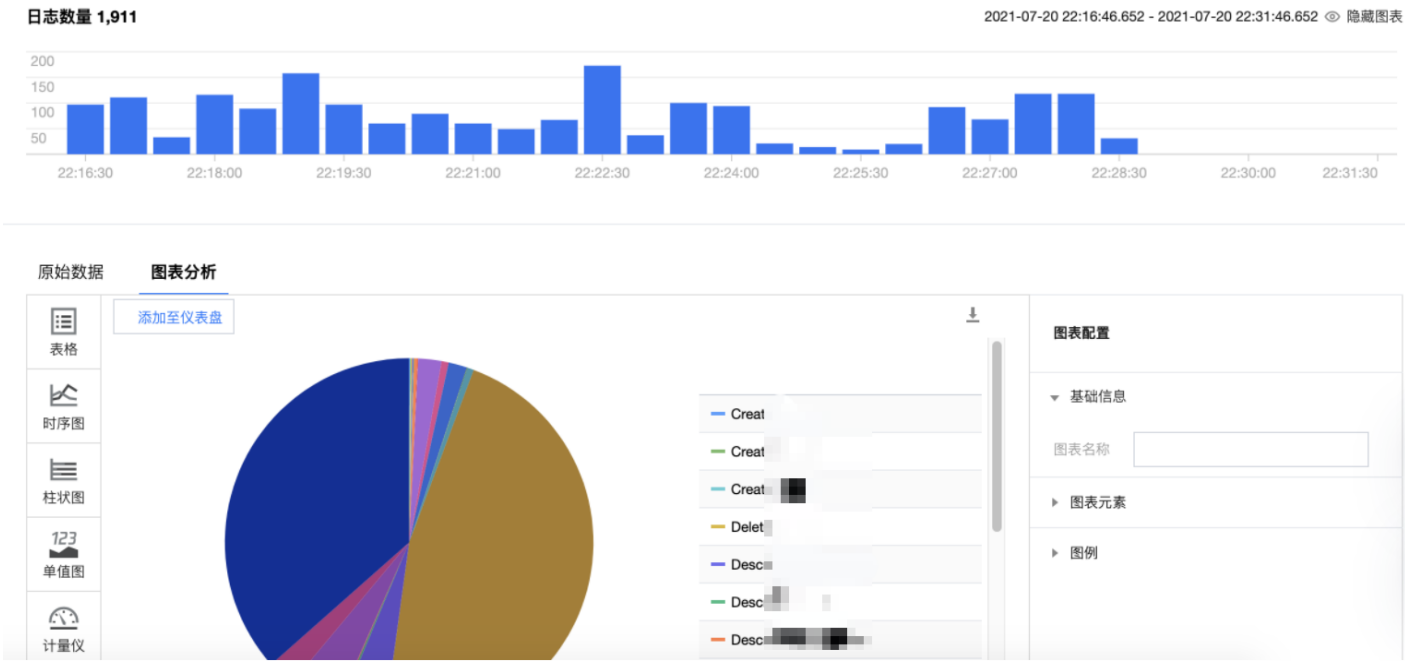
1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**检索分析**，进入检索分析页面。
3. 单击“日志集”和“日志主题”的下拉框，选择待检索的日志主题。

4. 单击查询时间范围，选择需要检索的日志时间。
   您可以选择最近时间、相对时间，也可以自定义时间范围。
5. 在左侧导航栏中，单击字段名称。



> 说明：
>
> - 字符串（text）类型字段与数值（long、double）类型字段的快速分析功能略有不同：
>   - 字符串：字段值分布统计、智能聚合时间折线图、TOP5值。
>   - 数值：智能聚合时间折线图、TOP5值。
> - 灰色字段不支持点击，是由于该字段未开启统计，例如上图中的 __PKG_LOGID__。

6. 在快速分析弹框中，单击某一统计图表，即可自动生成对应的检索语句及图表。您可以在此查询图表详细数据，也可以进一步修改检索语句及图表配置以获取更加符合您特定需求的分析图表。


日志数量 1,911
2021-07-20 22:16:46.652 - 2021-07-20 22:31:46.652
隐藏图表
200
150
100
50
22:16:30
22:18:00
22:19:30
22:21:00
22:22:30
22:24:00
22:25:30
22:27:00
22:28:30
22:30:00
22:31:30
原始数据
图表分析
表格
时序图
柱状图
单值图
计量仪
添加至仪表盘
Creat
Creat
Creat
Delet
Desc
Desc
Desc
图表配置
基础信息
图表名称
图表元素
图例

上下文检索

最近更新时间：2022-05-11 14:25:18

本文介绍如何在日志服务控制台查看指定日志在原始文件中的上下文信息。

操作场景

日志上下文检索是指把目标日志数据在原始文件中的前若干条日志（上文）或后若干条日志（下文）检索查询出来。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故障信息，方便定位问题。

功能优势

- 免除登录机器的繁琐步骤，在检索分析页面、快速查看任意机器，文件的指定日志的上下文信息。
- 结合事件发生的时间线索，在检索分析页指定时间段快速定位可疑日志后再进行上下文查询，快速定位问题。
- 不用担心服务器存储空间不足或日志文件轮转造成的数据丢失，在检索分析页上随时可以查看历史数据。

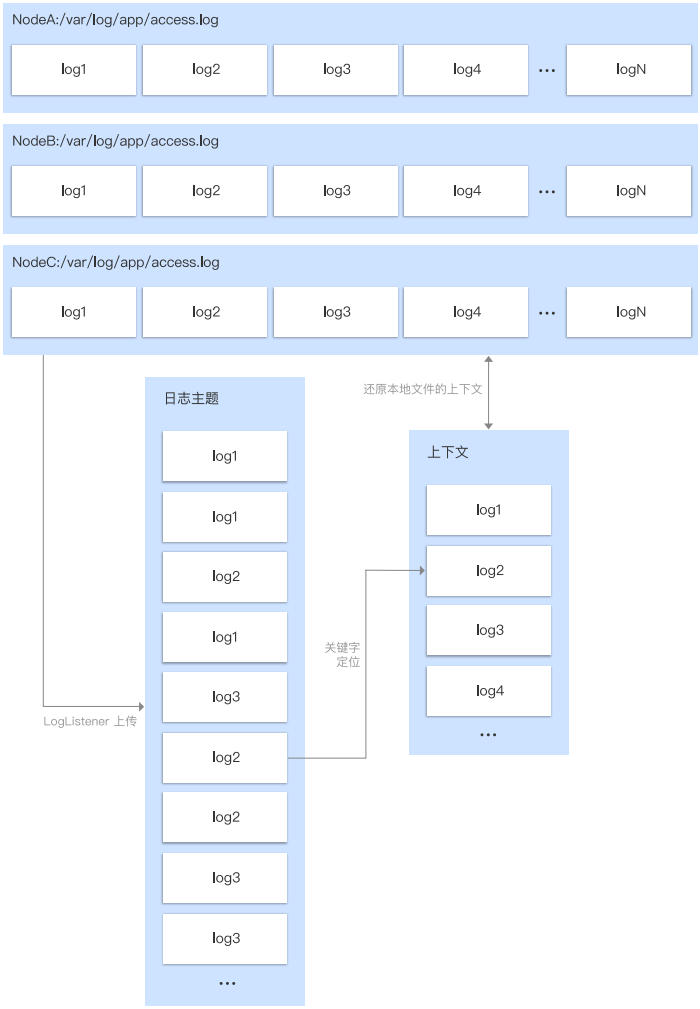
前提条件

- 上下文检索分析仅支持2.3.5以上版本使用，建议 安装或升级至最新版本。
- 只有通过 LogListener 采集到的日志才支持上下文功能。
- 已开启并配置索引，详情请参见 配置索引。

场景示例

订单交易轨迹：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
使用场景：若用户的某一单交易失败，可先通过订单号查询错误日志，再根据查到的错误日志还原上下文，找到失败原因。例如扣款失败。
在日志服务中，可以按照以下步骤排查：

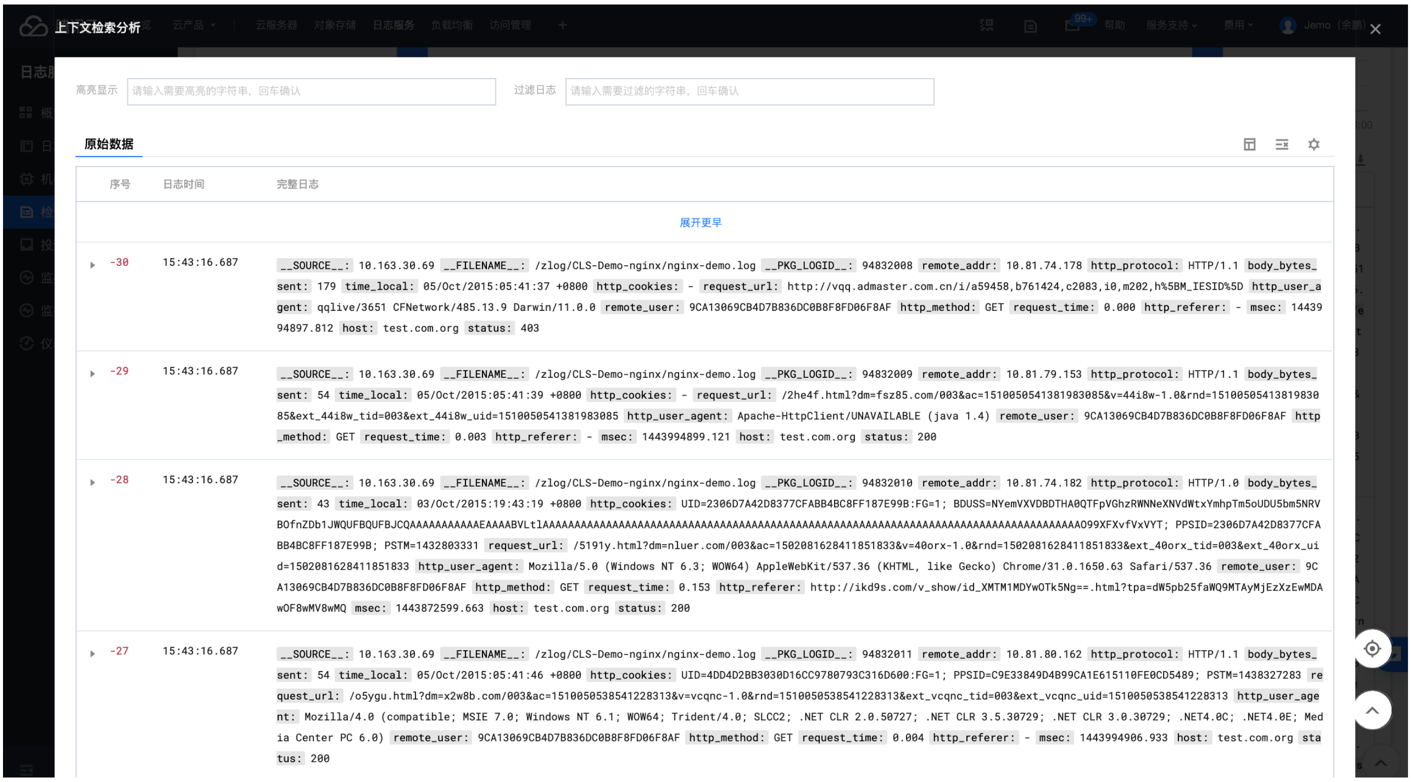
1. 在日志服务控制台的**检索分析**页面，结合事件发生的时间线索，输入**关键字**订单 ID 找到订单失败日志。
2. 以查到的日志为基准，向上或向下滚动直到发现与之相关的上下文日志信息。



操作步骤

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**检索分析**，进入检索分析管理页面。
3. 根据实际需求，选择**地域**、**日志集**和**日志主题**。
4. 输入检索分析语句，选择时间范围， 单击**检索分析**。
5. 在“**原始数据**”页签下，找到目标日志的日志时间，单击 ，进入上下文检索分析页面。

6. 在上下文检索分析页面，系统仅检索出目标日志的前10条日志数据（上文）和后10条日志数据（下文）。



您可以在此页面进行如下操作：

- 使用鼠标在当前页面上下滚动查看指定日志的上下文信息。
  - 单击**展示更早**，进行向上翻页浏览，每次多展示前50条日志数据。
  - 单击**展示更多**，进行向下翻页浏览，每次多展示后50条日志数据。
- 在**高亮显示**文本框中，输入关键字进行高亮显示，实现关键字黄色填充显示。
- 在**过滤日志**文本框中，输入需要过滤的字符串进行高亮显示。

下载日志

最近更新时间：2021-12-10 10:03:39

操作场景

日志服务提供了日志数据导出的功能，您可以将采集到的日志数据下载至本地查看。
下载日志功能如下：

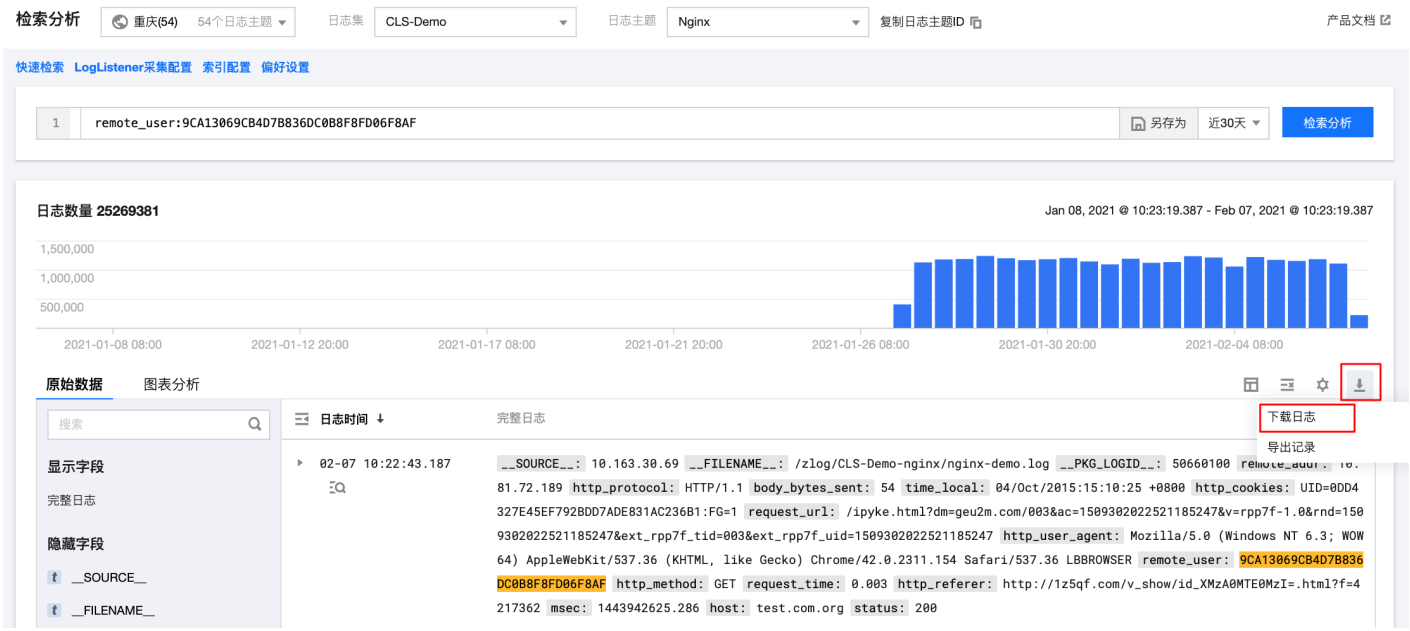
- 单次下载日志条数最多为5000万条。
- 支持指定检索条件，检索时间范围。
- 支持 JSON，CSV 导出格式。

> 说明：
> 下载日志仅支持下载原始日志，如需下载统计分析结果（即 SQL 执行结果），可在图表右上角单击 进行下载。

操作步骤

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**检索分析**，进入检索分析管理页面。
3. 单击“日志集”和“日志主题”的下拉框，选择待检索的内容。

4. 单击日志时间，选择需要检索的日志时间。
   您可以选择最近时间、相对时间，也可以自定义时间范围。
5. 单击**检索分析**。
6. 待检索到日志数据后，单击 > **下载日志**。



> 说明：
> 若未检索到日志数据，则无法下载日志。

7. 在弹出的窗口中，确认需下载日志的时间范围和检索语句，并根据如下信息设置下载日志数据。

- 数据格式：提供了 “CSV格式” 和 “JSON格式” 两种格式，请按需选择。
- 日志排序：提供了 “升序” 和 “降序” 两种规则，默认按时间降序。
- 日志数量：默认导出 “全部日志” ，您也可以选择 “自定义日志数量” ，自定义导出所需日志数量。

> 说明：
>
> - CSV 格式下只能导出已配置索引的字段，可能出现下载日志为空的现象。
> - JSON 格式下允许导出全部字段，不受索引配置的限制。
> - 需要下载的日志条数超出5000万时，建议缩小日志检索范围（例如采用更加精确的检索条件、缩小查询时间范围）或仅下载指定数量的日志。
> - 如果确实需要下载超出5000万的日志，可通过限定日志查询时间范围创建多个下载任务，分批进行下载。

8. 单击**导出**，切换为**导出记录**界面。

> 说明：
>
> 状态共包含 “等待中” 、 “文件生成中” 和 “文件已生成” 三个状态。新创建的下载任务会在等待片刻后进入 “文件生成中” 状态，并开始在后台生成数据文件。如有同时多个下载任务，系统会按照创建时间依次执行，创建较晚的任务会处于 “等待中” 。

9. 待 “文件名/任务ID” 状态变为**文件已生成**后，单击**下载**，即可导出日志。

> 注意：
>
> 导出的日志文件仅保留3天。

可视化分析
图表分析
图表概述

最近更新时间：2022-05-20 15:07:28

日志服务提供丰富的图表类型，支持用户使用 SQL 灵活统计日志内的系统及业务指标，并通过图表进行展示。用户还可以将图表分析结果保存到仪表盘，进行长期监控。

图表类型

图表类型	使用场景	详细信息
表格	表格是最常见的数据展示类型，通过对数据结构化的整理，实现数据的对比与统计。大多数场景均适用。	表格
时序图	时序图需要统计数据具备时序字段，依据时间顺序组织与聚合指标。可直观反映指标随时间的变化趋势。统计近一周，每天404错误出现的次数等趋势分析场景适用。	时序图
柱状图	柱状图描述的是分类数据，直观表现每一个分类项的大小对比关系。统计近一天各错误码类型出现的次数等分类统计场景适用。	柱状图
饼图	饼图描述的是不同分类的占比情况，通过扇区大小来衡量各分类项的占比情况。错误码占比情况分析等占比统计场景适用。	饼图
单值图	单值图描述的是单个指标，一般选择具备有业务价值的关键性指标。统计天、周、月 PV、UV 等单指标场景适用。	单值图
计量仪	计量仪描述的是单个指标，与单值图不同的是，计量仪一般搭配阈值使用，用来衡量该指标的状态。系统健康度监控等有分级标准的场景适用。	计量仪
地图	地图通过图形的位置来表现数据的地理位置，通常来展示数据在不同地理区域上的分布情况。攻击 IP 地理分布等地理位置统计场景适用。	地图
桑基图	桑基图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。防火墙目的与源 IP 流量统计等场景适用。	桑基图
词云图	词云是文本数据的视觉表示，用于展示文本数据的出现频率。在高频操作人员统计等审计场景适用。	词云图

常见功能操作

功能名称	功能描述	详细信息
数据转换	数据转换支持用户对检索结果进行二次处理，包括字段类型修改、选择制图字段、合并分组等。可以在不修改 SQL 语句的情况下，满足制图的需求。	数据转换
单位换算	图表支持自动单位换算，选择一个原始单位后，当数值满足换算进率时，自动换算为更高一级别的单位。单位支持精度配置。	单位配置
添加图表到仪表盘	支持将图表分析结果保存到仪表盘，进行长期监控。	添加图表到仪表盘

表格

最近更新时间：2022-05-20 15:07:24

表格是最常见的数据展示类型，通过对数据结构化的整理，实现数据的对比与统计。大多数场景均适用。

图表配置

配置项	说明
基础信息	图表名称：设置图表的显示名称，可为空。
表格	对齐：设置表格内容在单元格里的对齐方式。默认配置下，指标类型的字段右对齐，维度类型的字段左对齐。
标准配置	设置图表内所有指标类型的字段单位。详情请参考 单位配置。

图表操作

搜索表格内容

在表格顶部的**搜索框**中，输入搜索内容的关键词，单击**搜索**。

按照某一列排序

单击需要排序的列的**表头**，将会按照升序排列。再次单击，切换为降序排列。

server	# 入流量(KB) ↓	# 出流量(KB)	# 总流量(MB)
192.168.0.23	95.6	20002.93	19.63
192.168.0.206	94.68	19217.17	18.86
192.168.0.106	93.16	25394.2	24.89
192.168.0.224	92.6	17762.99	17.44
192.168.0.216	92.08	16423.64	16.13
192.168.0.43	87.37	25059.93	24.56
192.168.0.252	79.11	17114.31	16.79
192.168.0.134	78.75	16614.44	16.3
192.168.0.1	76.16	18497.78	18.14

时序图

最近更新时间：2022-05-20 15:07:20

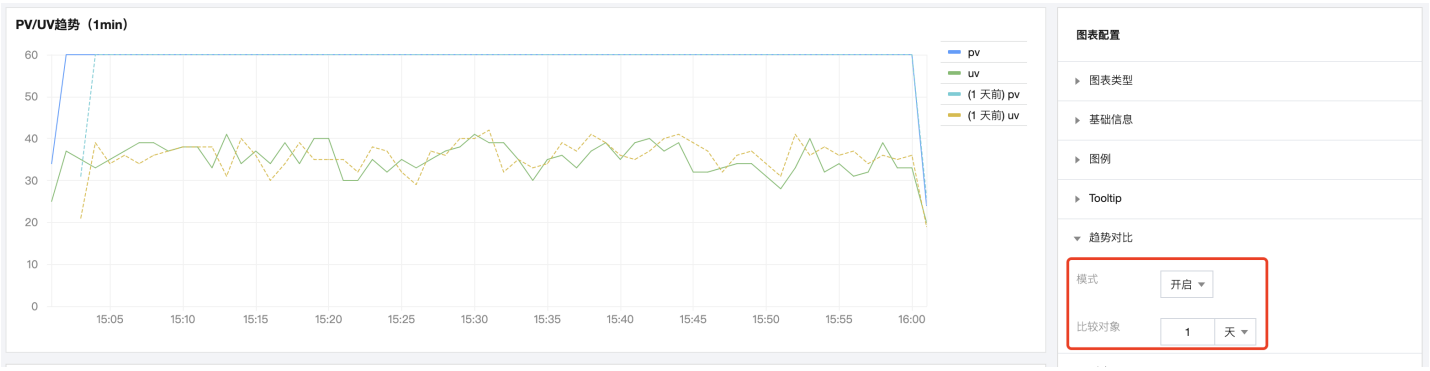
时序图需要统计数据具备时序字段，依据时间顺序组织与聚合指标。可直观反映指标随时间的变化趋势。统计近一周，每天404错误出现的次数等趋势分析场景适用。

图表配置

通用配置

配置项	说明
基础信息	图表名称：设置图表的显示名称，可为空。
图例	设置图表的图例内容，可以控制图例的样式与位置。同时也支持在图例中添加对比数据。
Tooltip	控制鼠标悬浮时触发的气泡提示的内容样式。
标准配置	设置图表内所有指标类型的字段单位。详情请参考 单位配置。

趋势对比



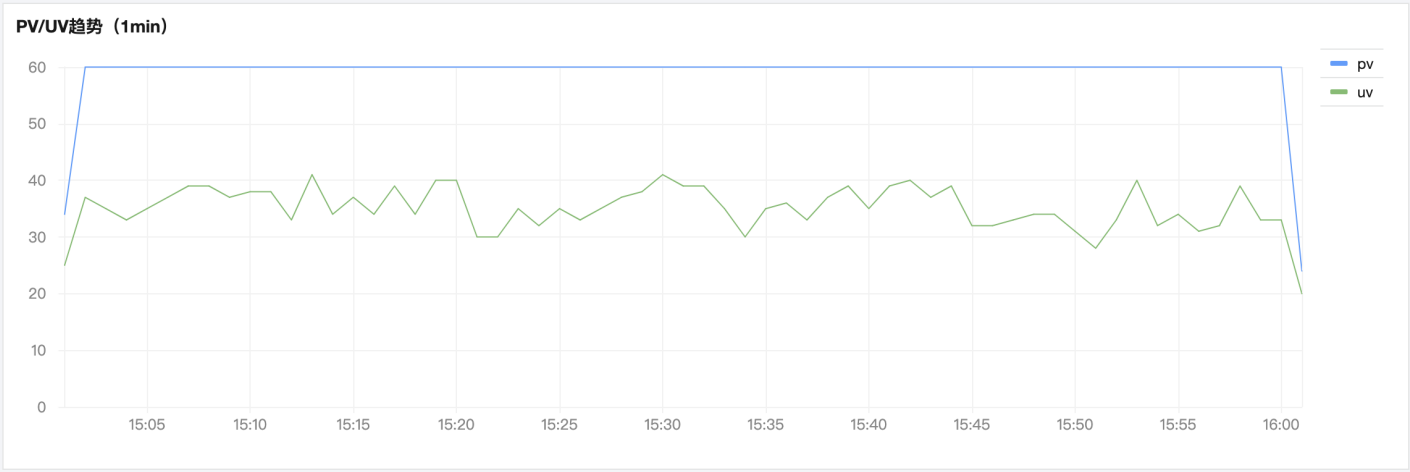
配置项	说明
趋势对比	开启趋势对比后，可以选择和 X 小时、天、月、年以前的同周期数据进行对比。对比数据以虚线的方式显示在图表中。

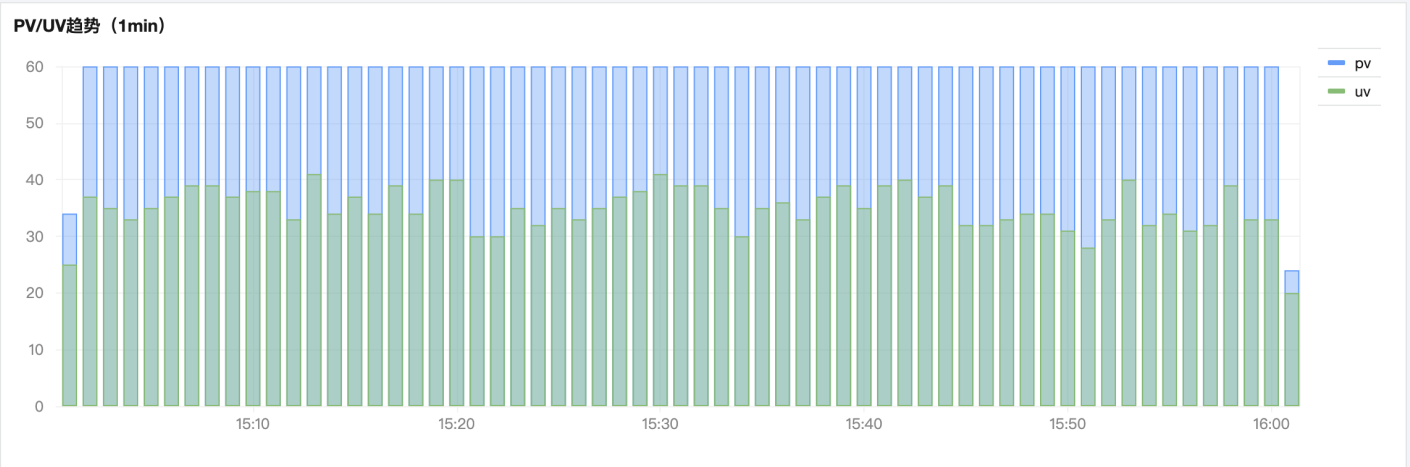
时序图配置

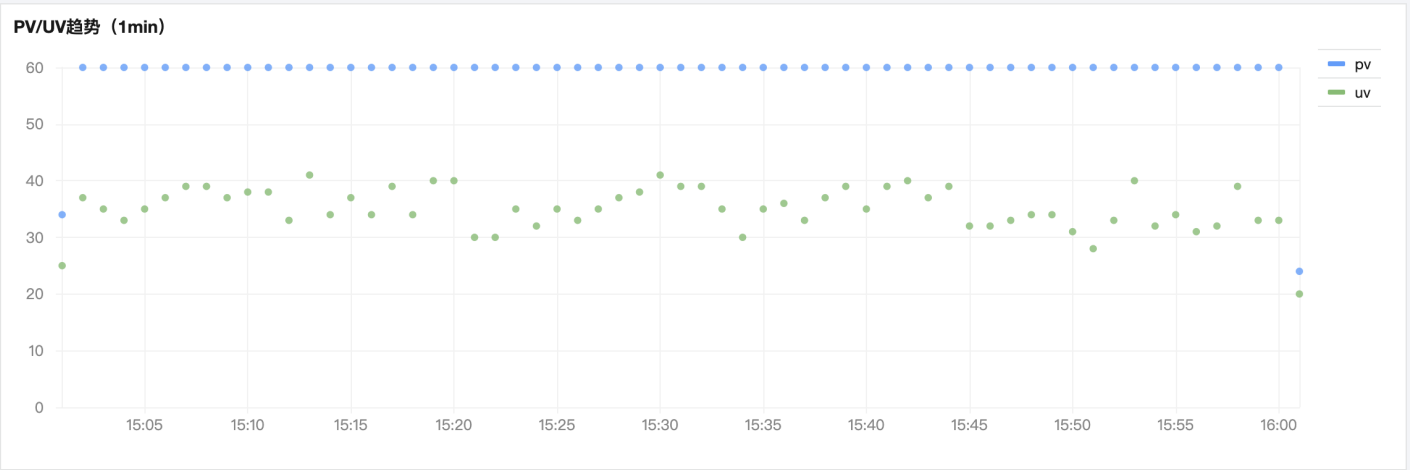
配置项	说明

配置项	说明
时序图	绘制样式：数据在坐标轴上的显示样式，选择线则为折线图，选择柱则为直方图，选择点则为散点图。 线性：点与点之间的连线是否平滑处理。 线宽：控制线条的粗细。 填充：控制填充区域的透明度，为0时不填充。 显示点：显示数据点，无数据则不显示。 空值：控制时序序列点位上无数据存在时，该点位的处理，默认填充为0。 堆叠：控制是否将数据堆叠显示。
坐标轴	显示：显示/隐藏坐标轴。 最大值/最小值：控制坐标轴显示的最大值与最小值，大于最大值，小于最小值的坐标区域不显示。

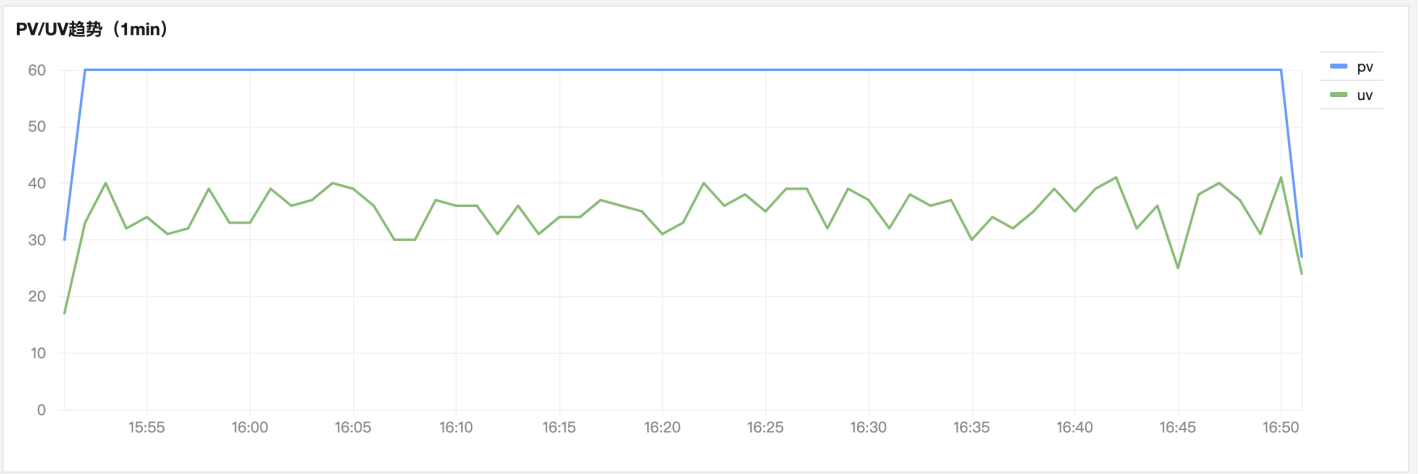
绘制样式示例

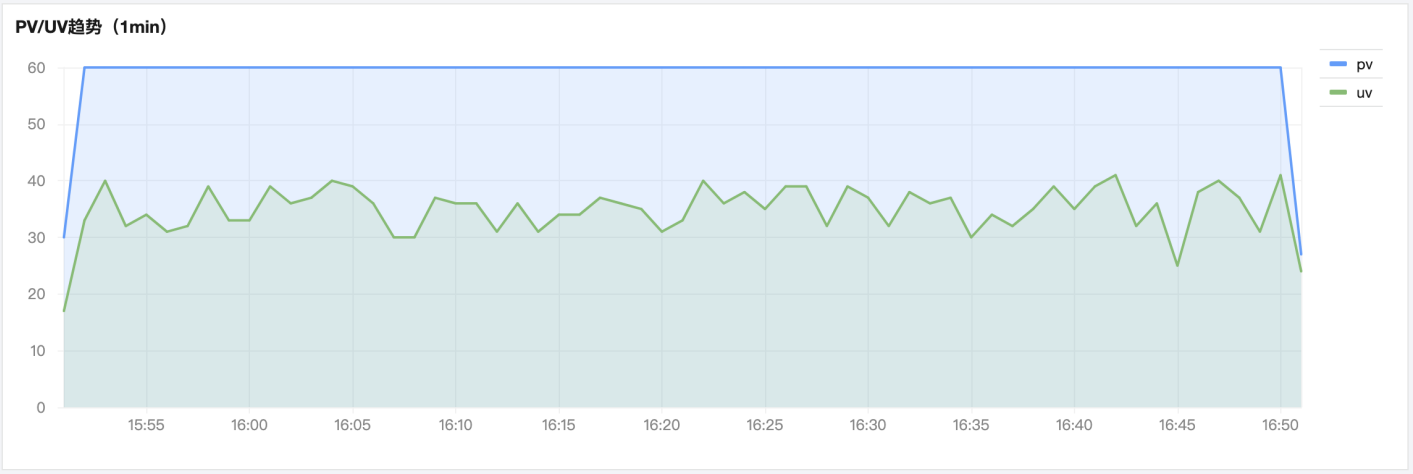




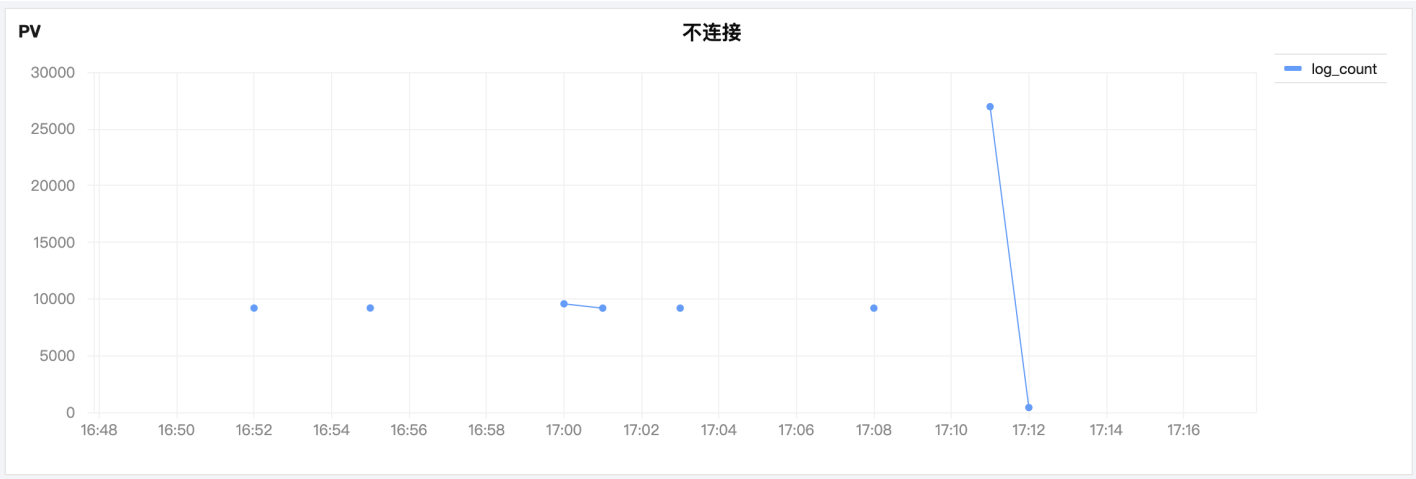


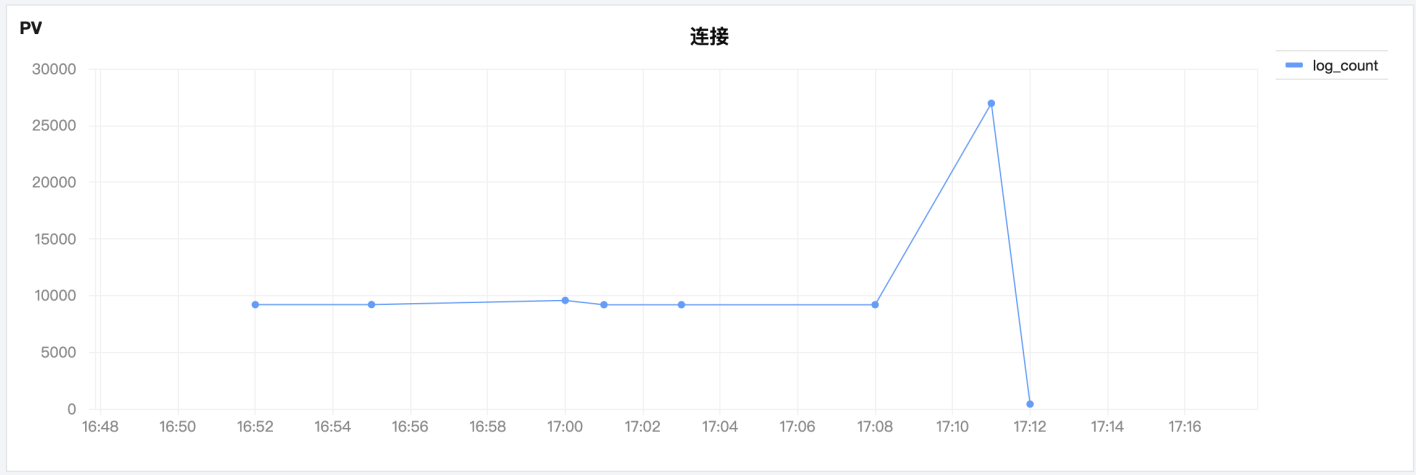
填充示例

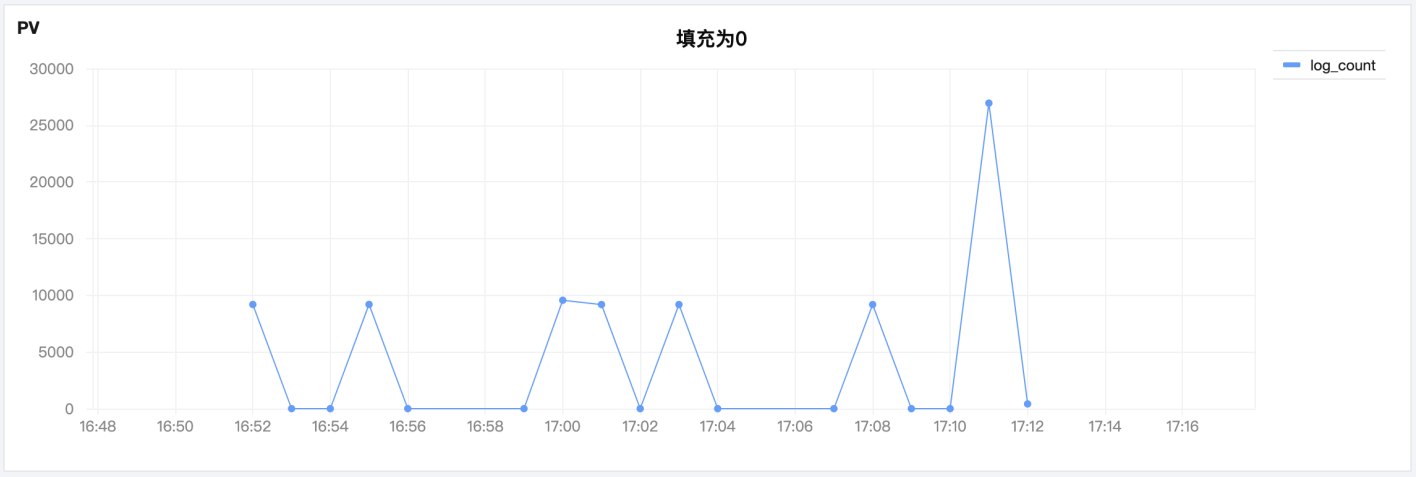

PV/UV趋势（1min）
pv
uv
60
50
40
30
20
10
0
15:55
16:00
16:05
16:10
16:15
16:20
16:25
16:30
16:35
16:40
16:45
16:50


PV/UV趋势（1min）
pv
uv
60
50
40
30
20
10
0
15:55
16:00
16:05
16:10
16:15
16:20
16:25
16:30
16:35
16:40
16:45
16:50

空值示例


PV
不连接
log_count
30000
25000
20000
15000
10000
5000
0
16:48
16:50
16:52
16:54
16:56
16:58
17:00
17:02
17:04
17:06
17:08
17:10
17:12
17:14
17:16


PV
连接
log_count
30000
25000
20000
15000
10000
5000
0
16:48
16:50
16:52
16:54
16:56
16:58
17:00
17:02
17:04
17:06
17:08
17:10
17:12
17:14
17:16

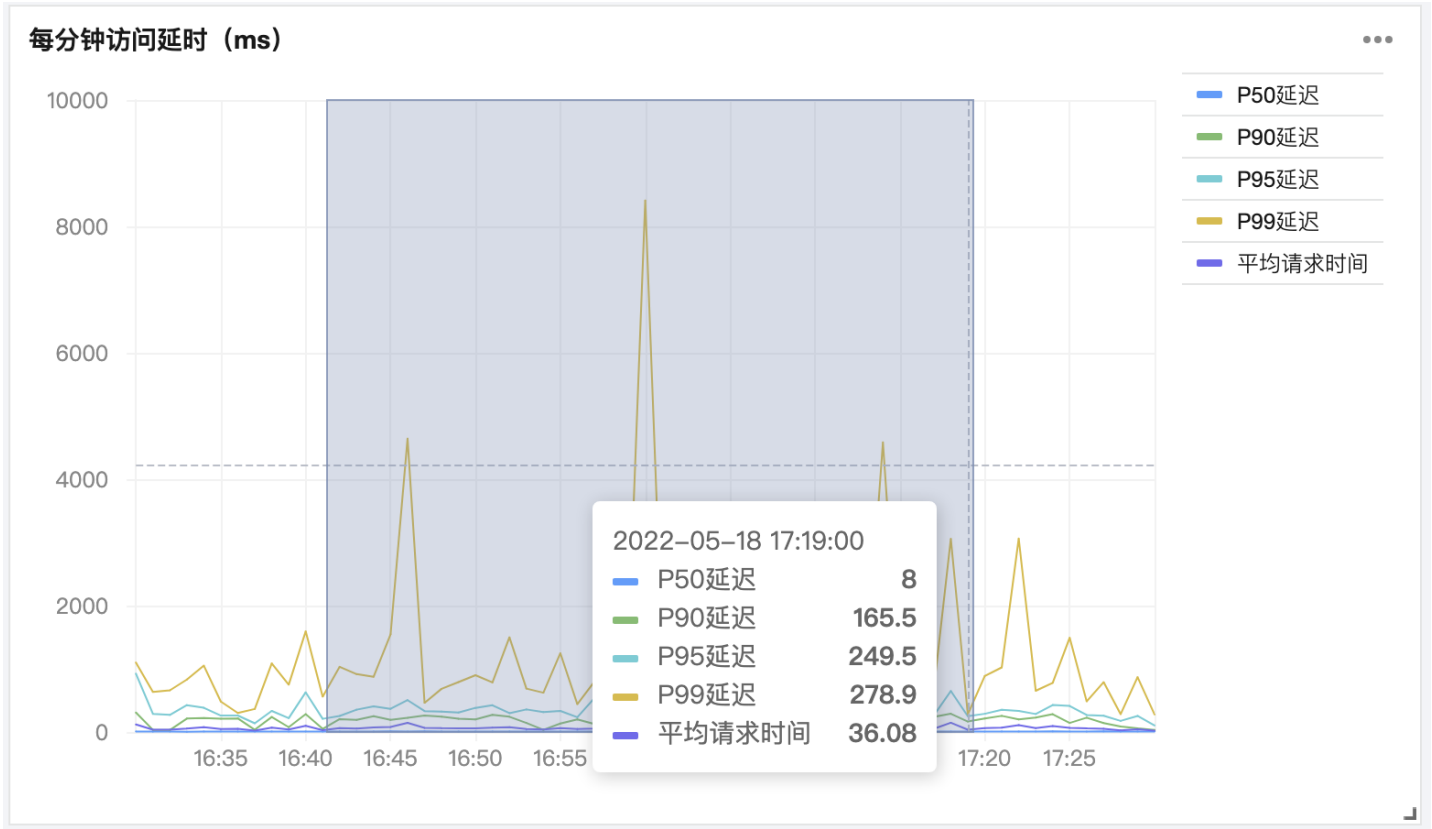

PV
填充为0
log_count
30000
25000
20000
15000
10000
5000
0
16:48
16:50
16:52
16:54
16:56
16:58
17:00
17:02
17:04
17:06
17:08
17:10
17:12
17:14
17:16

阈值配置

配置项	说明
阈值配置	阈值点：设置阈值点，可以添加多个阈值区间，点击阈值对应色彩可打开色盘自定义颜色。 阈值展示：控制阈值展示的样式，包含阈值线、区域填充、阈值线和区域填充三种模式。选择关闭时，不使用阈值。

图表操作

框选时间范围



鼠标悬浮在图表上，长按并拖拽触发框选，时间范围将会使用框选区域的时间。适用于异常点的时间范围下钻等场景。

柱状图

最近更新时间：2022-05-20 15:07:15

柱状图描述的是分类数据，直观表现每一个分类项的大小对比关系。统计近一天各错误码类型出现的次数等分类统计场景适用。

图表配置

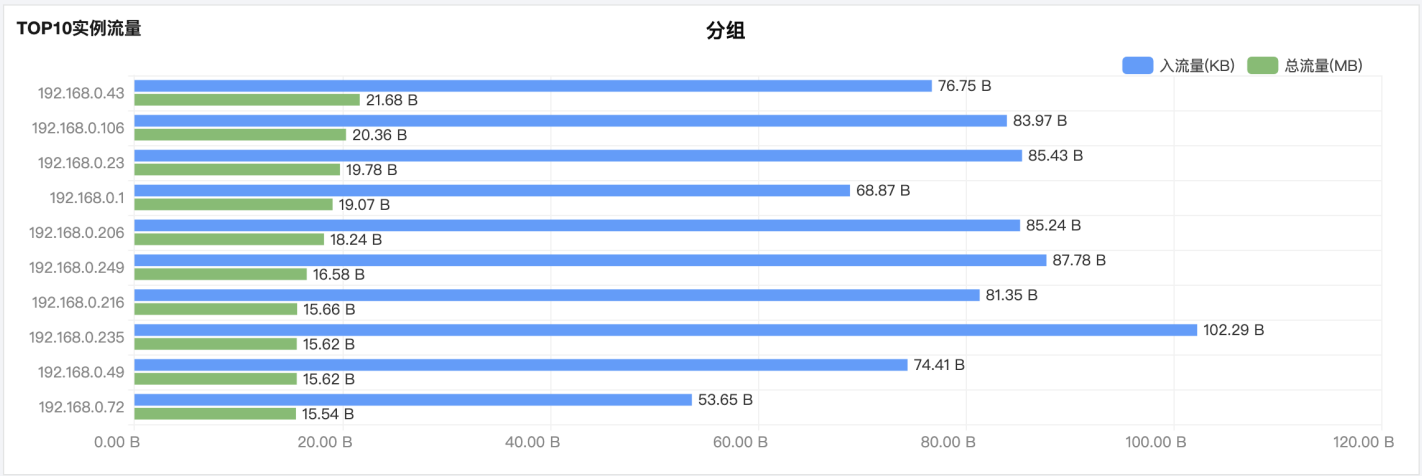
通用配置

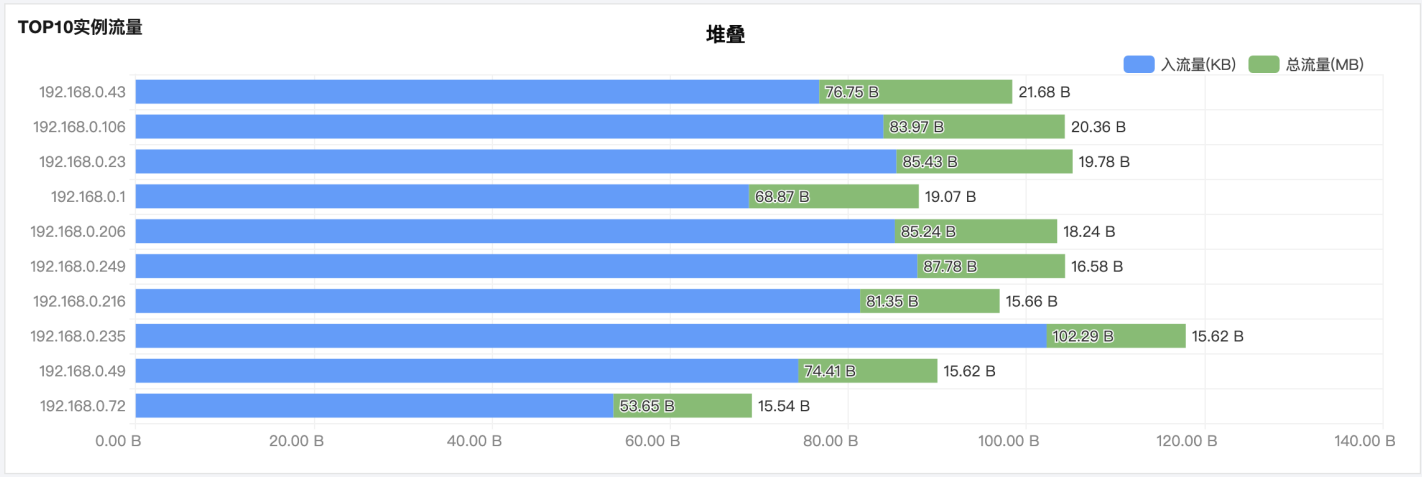
配置项	说明
基础信息	图表名称：设置图表的显示名称，可为空。
标准配置	设置图表内所有指标类型的字段单位。详情请参考 单位配置。

柱状图配置

配置项	说明
柱状图	方向：控制条柱的方向，水平方向即为条形图，垂直方向为柱形图。 排序方式：控制条柱的排序方式，支持按照指标升序与降序排列。在多个有指标时需要指定一个用于排序的指标。默认不排序。 显示数值：控制是否显示每个条柱的数值标签。 条柱模式：支持条柱分组或堆叠显示。

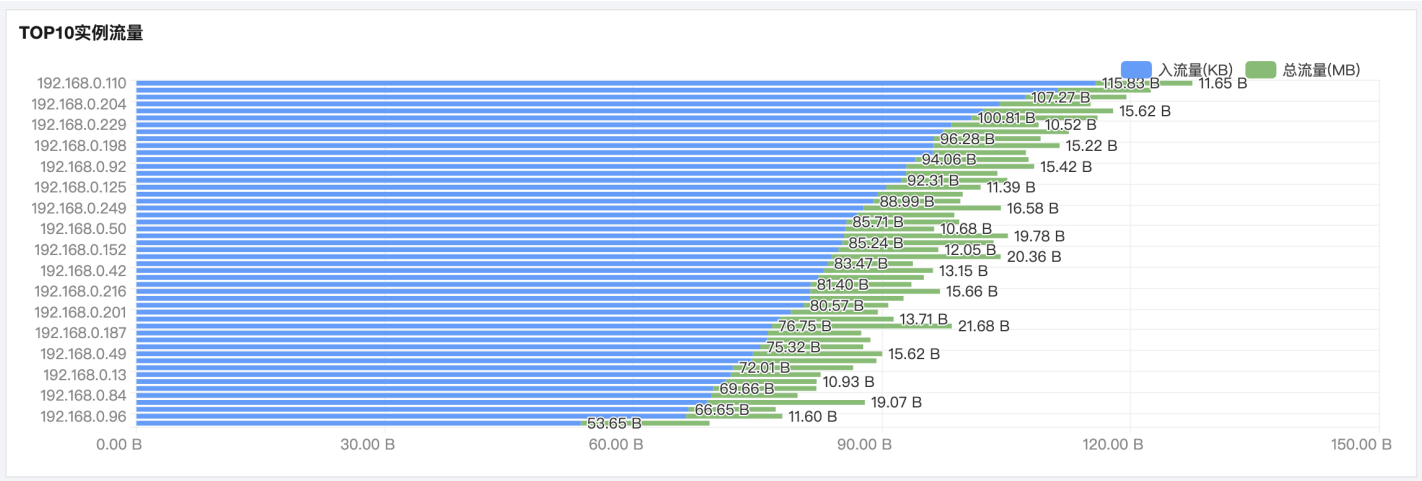
条柱模式示例：





图表操作

局部放大



使用中统计结果过多时，会造成上图所示，条柱密集，标签发生重叠隐藏，影响分析的情况。此时，鼠标悬浮在图表上，滑动鼠标滚轮，可以放大或缩小显示范围，从而聚焦局部内容，显示完整的信息。

饼图

最近更新时间：2022-05-20 15:07:10

饼图描述的是不同分类的占比情况，通过扇区大小来衡量各分类项的占比情况。错误码占比情况分析等占比统计场景适用。

图表配置

通用配置

配置项	说明
基础信息	图表名称：设置图表的显示名称，可为空。
图例	设置图表的图例内容，可以控制图例的样式与位置。同时也支持在图例中添加对比数据。
标准配置	设置图表内所有指标类型的字段单位。详情请参考 单位配置。

饼图配置

饼图

展示方式 实心

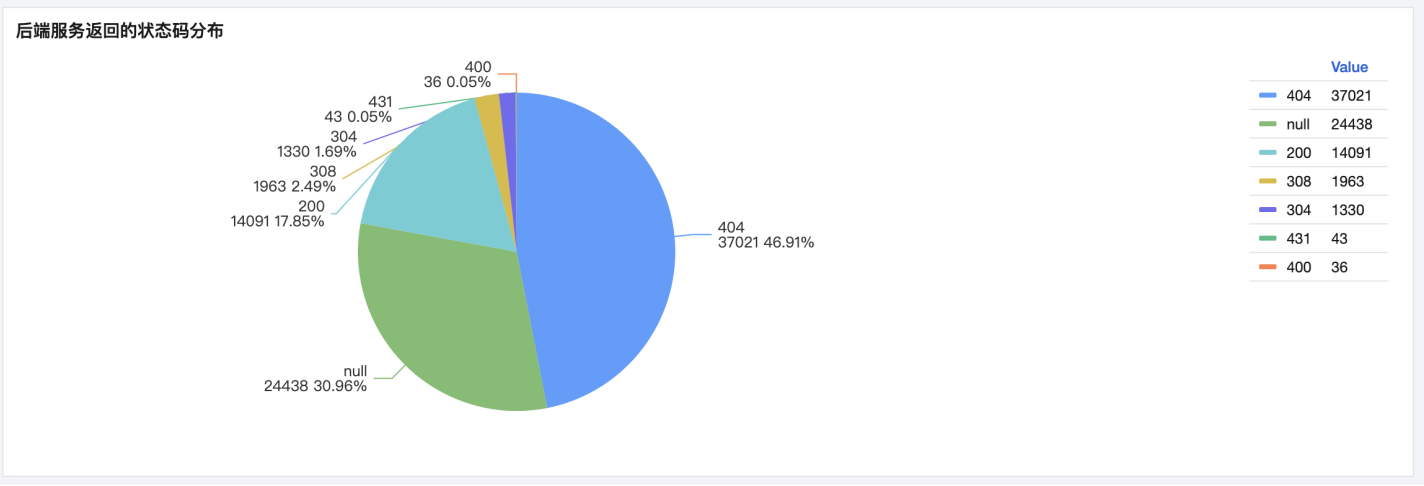
排序方式 降序(DESC)

扇区合并 auto

标签 请选择

配置项	说明
饼图	展示方式：控制饼图的样式，实心为饼图，空心为环形图。 排序方式：控制扇区的排序方式，支持升序或降序，默认不排序。 扇区合并：合并 TOPN 以外的扇区为一个其他分区。当扇区分类过多时，可用此功能聚焦 TOPN 对象。默认不合并。 标签：显示饼图标签，可选择标签内容为：名称、数值、百分比中的一项或多项。

标签示例：



单值图

最近更新时间：2022-05-20 15:07:05

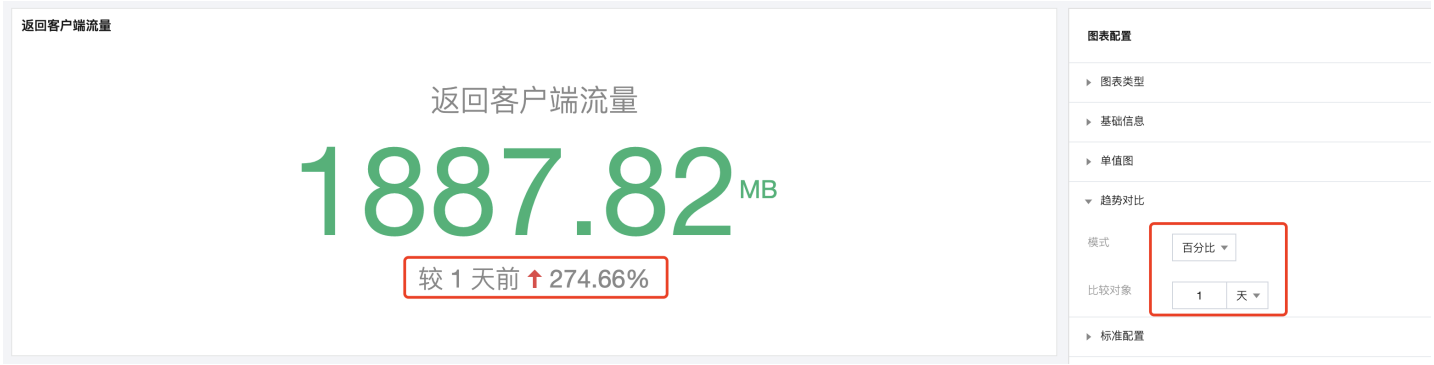
单值图描述的是单个指标，一般选择具备有业务价值的关键性指标。统计天、周、月 PV、UV 等单指标场景适用。

图表配置

通用配置

配置项	说明
基础信息	图表名称：设置图表的显示名称，可为空。
标准配置	设置图表内所有指标类型的字段单位。详情请参考 单位配置。

趋势对比



配置项	说明
趋势对比	开启趋势对比后，可以选择和 X 小时、天、月、年以前的同周期数据进行对比。趋势对比可以选择绝对值与百分比两类。

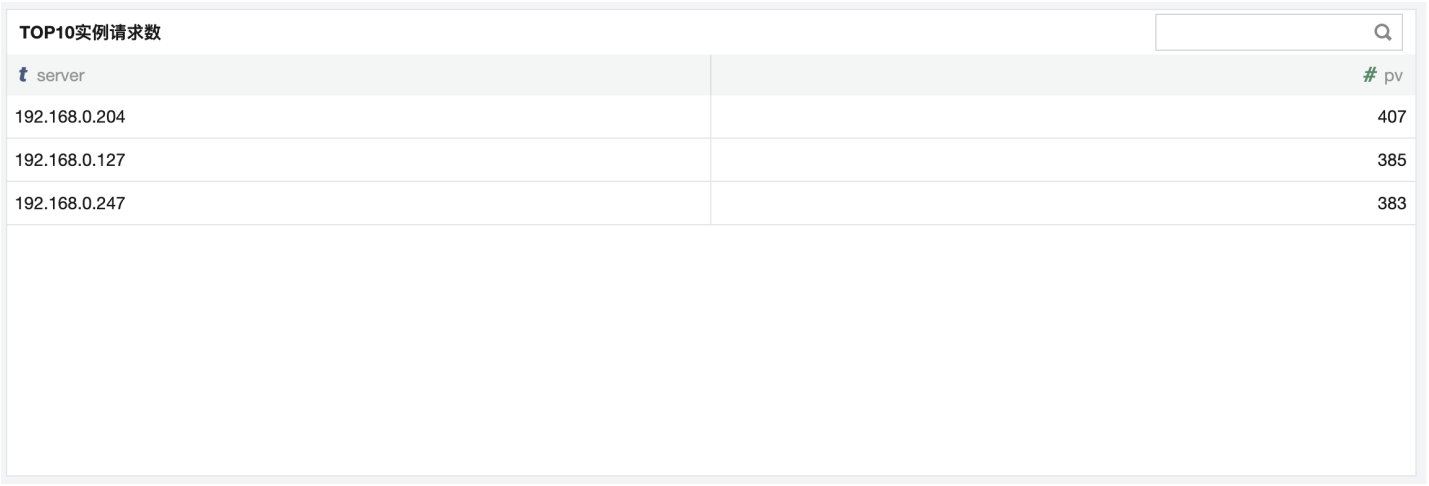
单值图配置

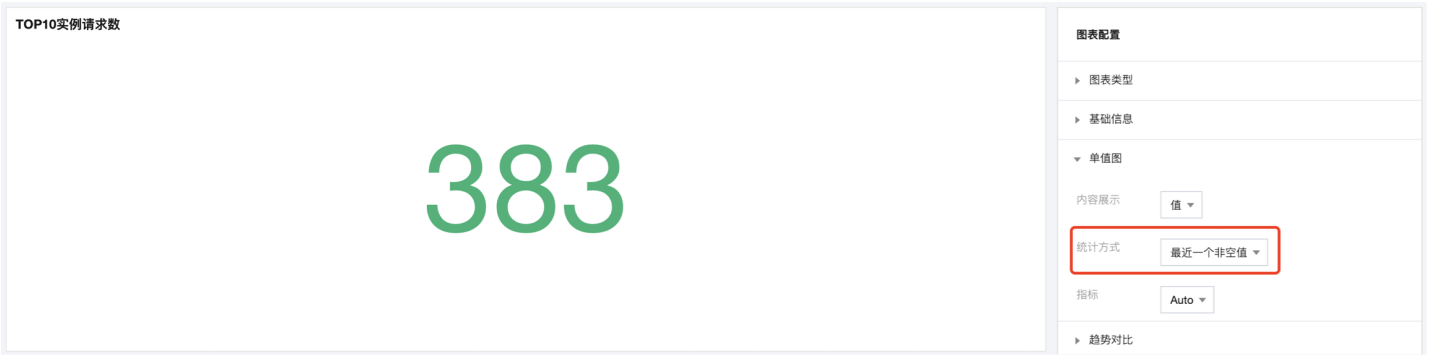
配置项	说明

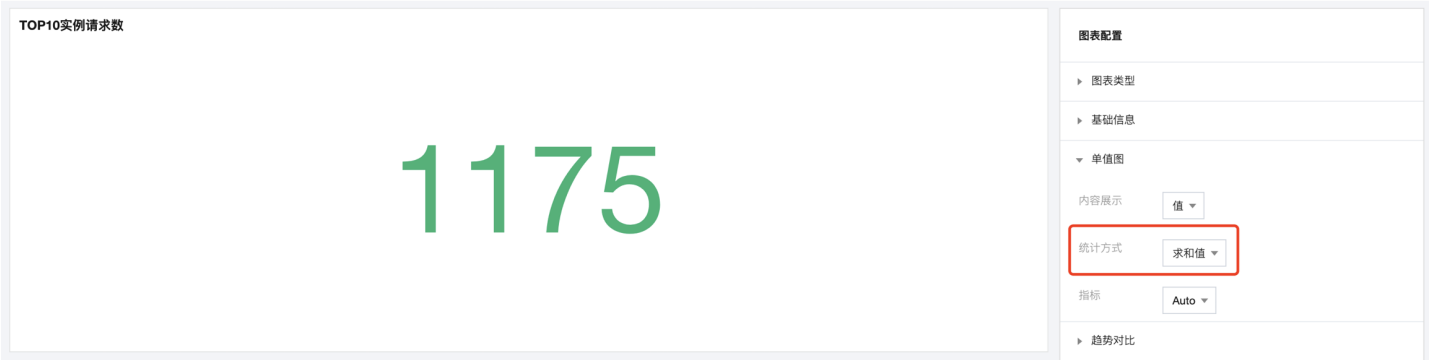
配置项	说明
单值图	内容展示：控制是否显示单值图的指标名称。 统计方式：当统计结果中指标有多个结果时，单值图需要把结果聚合为一个数值显示或选择其中一个显示。默认使用最近一个非空值。 指标：统计的目标指标，默认为自动，会选择返回的数据中排在第一的指标字段。

统计方式示例：

返回3条数据，默认选择最近一个非空值，即排在末尾的值383显示。若修改为求和值，则会把3条数据结果求和后显示。







阈值配置

▼ 阈值配置

配置项	说明
阈值配置	阈值点：设置阈值点，可以添加多个阈值区间。单击阈值对应色彩可打开色盘自定义颜色。

计量仪

最近更新时间：2022-05-20 15:07:00

计量仪描述的也是单个指标，但与单值图不同的是，计量仪一般搭配阈值使用，用来衡量该指标的状态。系统健康度监控等有分级标准的场景适用。

图表配置

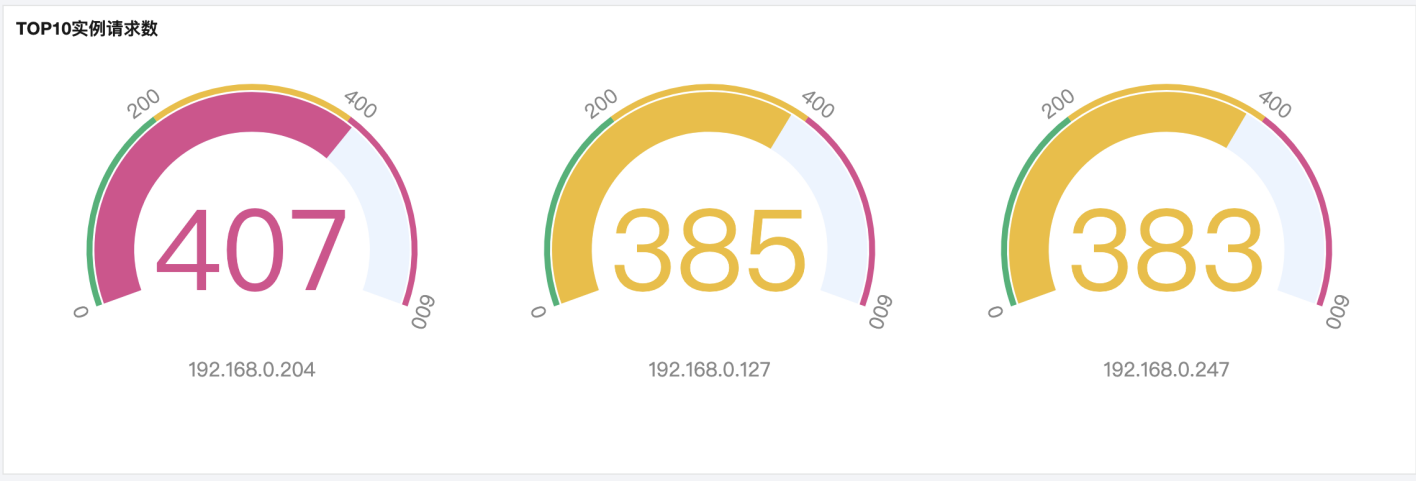
通用配置

配置项	说明
基础信息	图表名称：设置图表的显示名称，可为空。
标准配置	设置图表内所有指标类型的字段单位。详情请参考 单位配置。

阈值配置

配置项	说明
阈值配置	阈值点：设置阈值点，可以添加多个阈值区间。单击阈值对应色彩可打开色盘自定义颜色。 最大值/最小值：控制计量仪扇形区的最大刻度与最小刻度，区间外的数据不会在图表上显示。

阈值配置示例：


TOP10实例请求数
0
200
400
600
407
192.168.0.204
0
200
400
600
385
192.168.0.127
0
200
400
600
383
192.168.0.247

地图

最近更新时间：2022-05-20 15:06:56

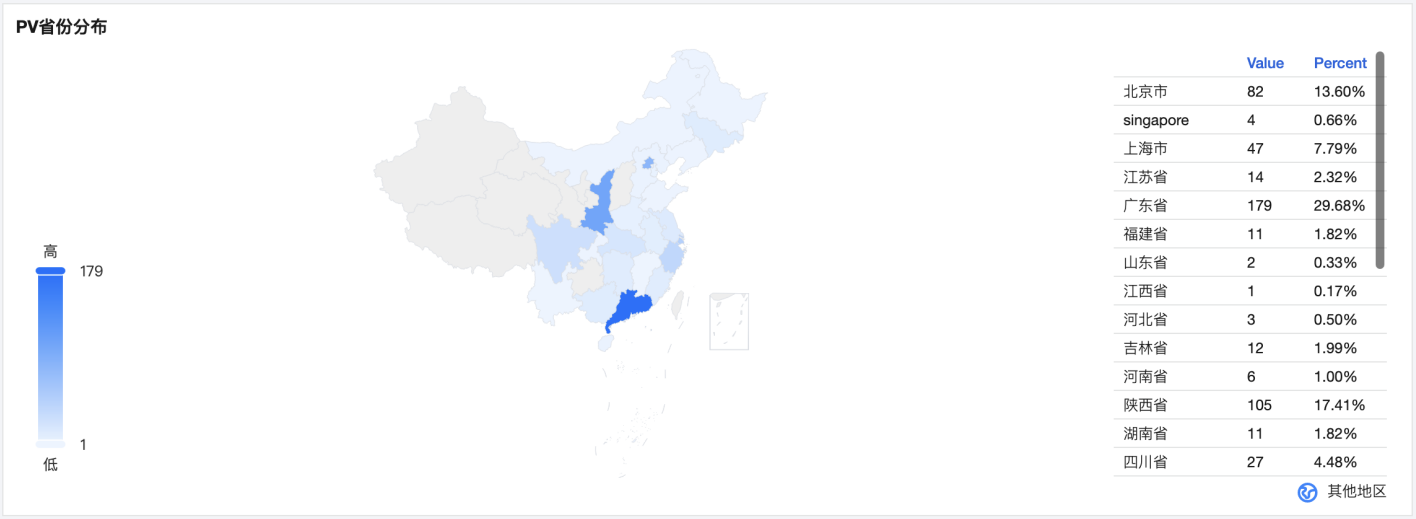
地图通过图形的位置来表现数据的地理位置，通常来展示数据在不同地理区域上的分布情况。攻击 IP 地理分布等地理位置统计场景适用。

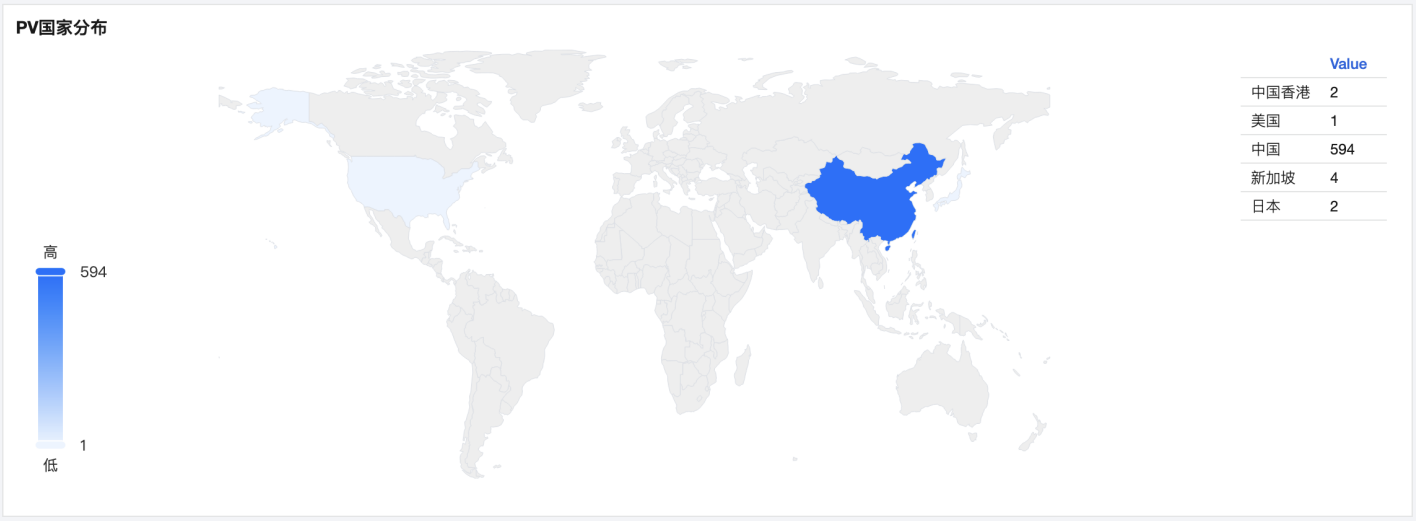
图表配置

通用配置

配置项	说明
基础信息	图表名称：设置图表的显示名称，可为空。
地图	区域定位：设置地图显示的区域范围，包含中国地图和世界地图两种。默认为自动，根据数据包含的区域信息自动适配。
图例	设置图表的图例内容，可以控制图例的样式与位置。同时也支持在图例中添加对比数据。

地图示例：

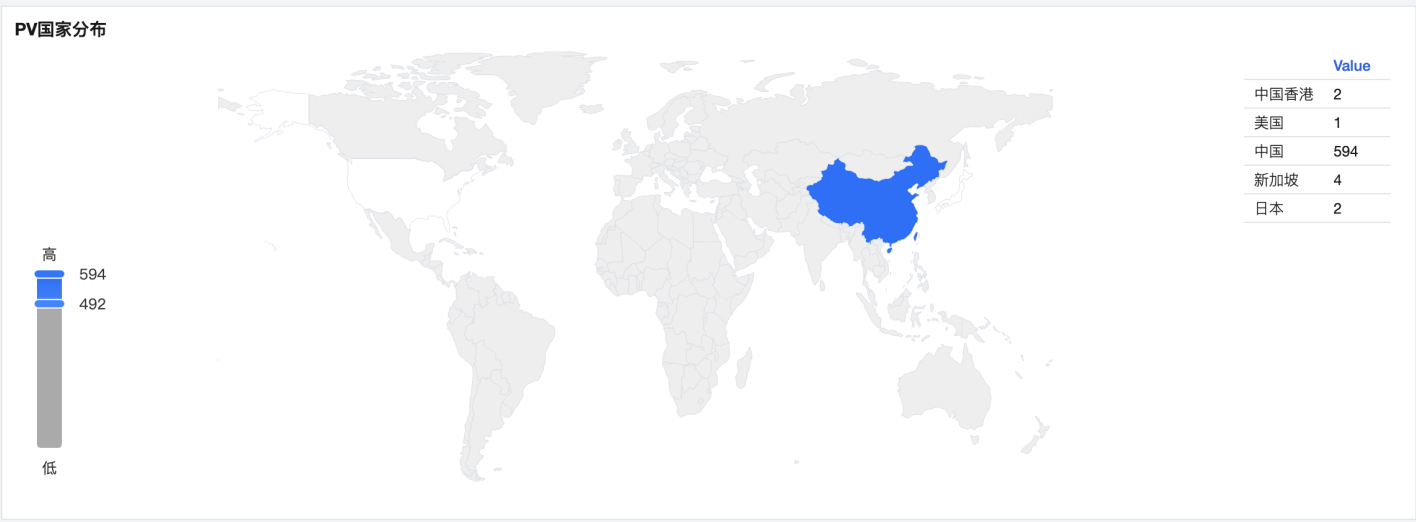




图表操作

根据数值过滤区域

拖拽地图左下角的数值范围，可只显示区间内的区域。



桑基图

最近更新时间：2022-05-20 15:06:52

桑基图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。防火墙目的与源IP流量统计等场景适用。

图表配置

通用配置

基础信息

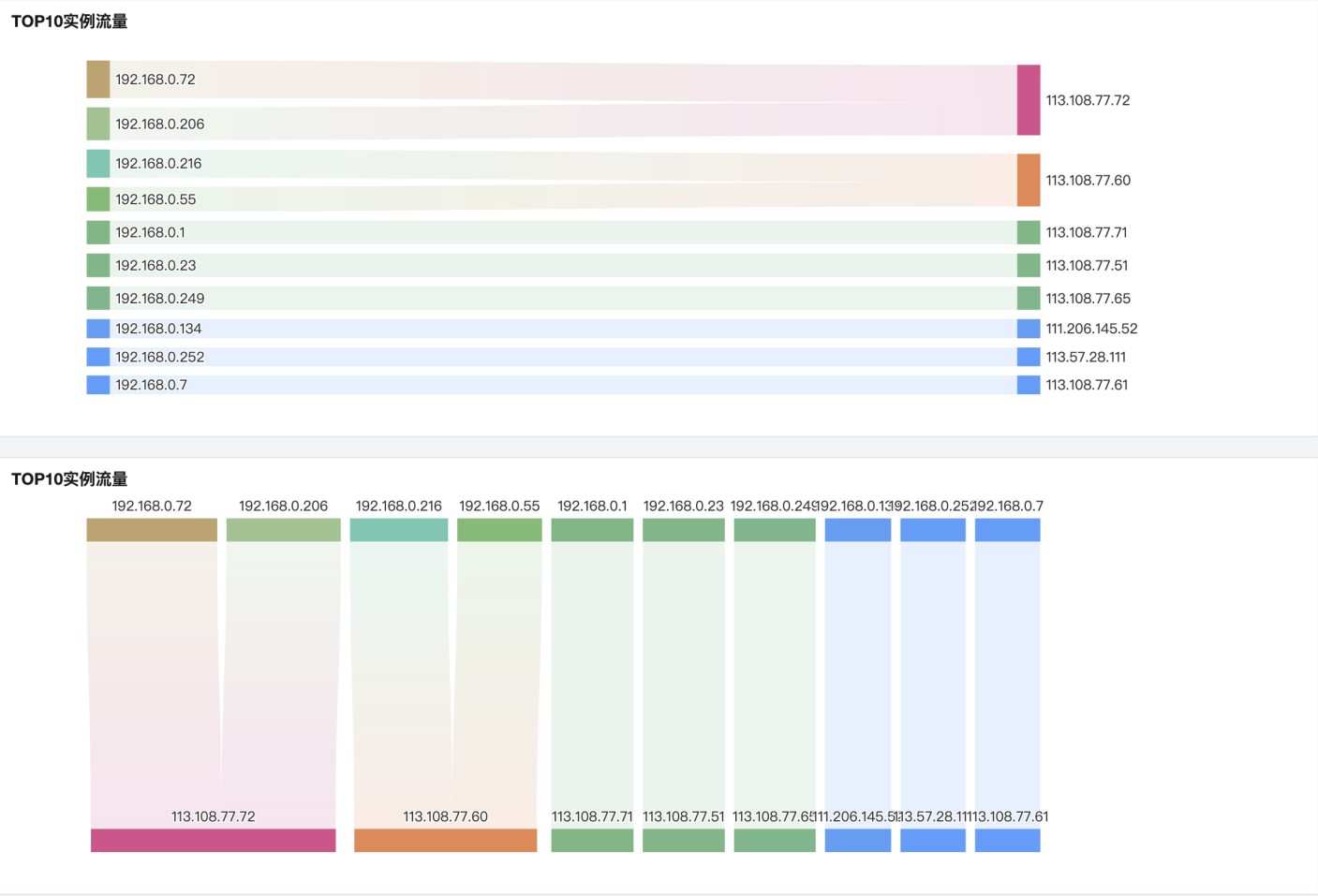
图表名称 TOP10实例流量

桑基图

方向 垂直

配置项	说明
基础信息	图表名称：设置图表的显示名称，可为空。
桑基图	方向：设置桑基图的显示方向。

桑基图方向示例：



词云图

最近更新时间：2022-05-26 10:55:58

词云是文本数据的视觉表示，用于展示文本数据的出现频率。在高频操作人员统计等审计场景适用。

图表配置

通用配置

基础信息

图表名称 PV省份分布

词云

显示词条数 78

字体大小 8 64

标准配置

单位 none

精度 auto

配置项	说明
基础信息	图表名称：设置图表的显示名称，可为空。
词云	显示词条数：控制最大显示多少条数据，取 TOPN，最大取100条。 字体大小：控制词云中词条的字体大小范围。
标准配置	设置图表内所有指标类型的字段单位。详情请参考 单位配置。

词云图示例：

PV省份分布

数据转换

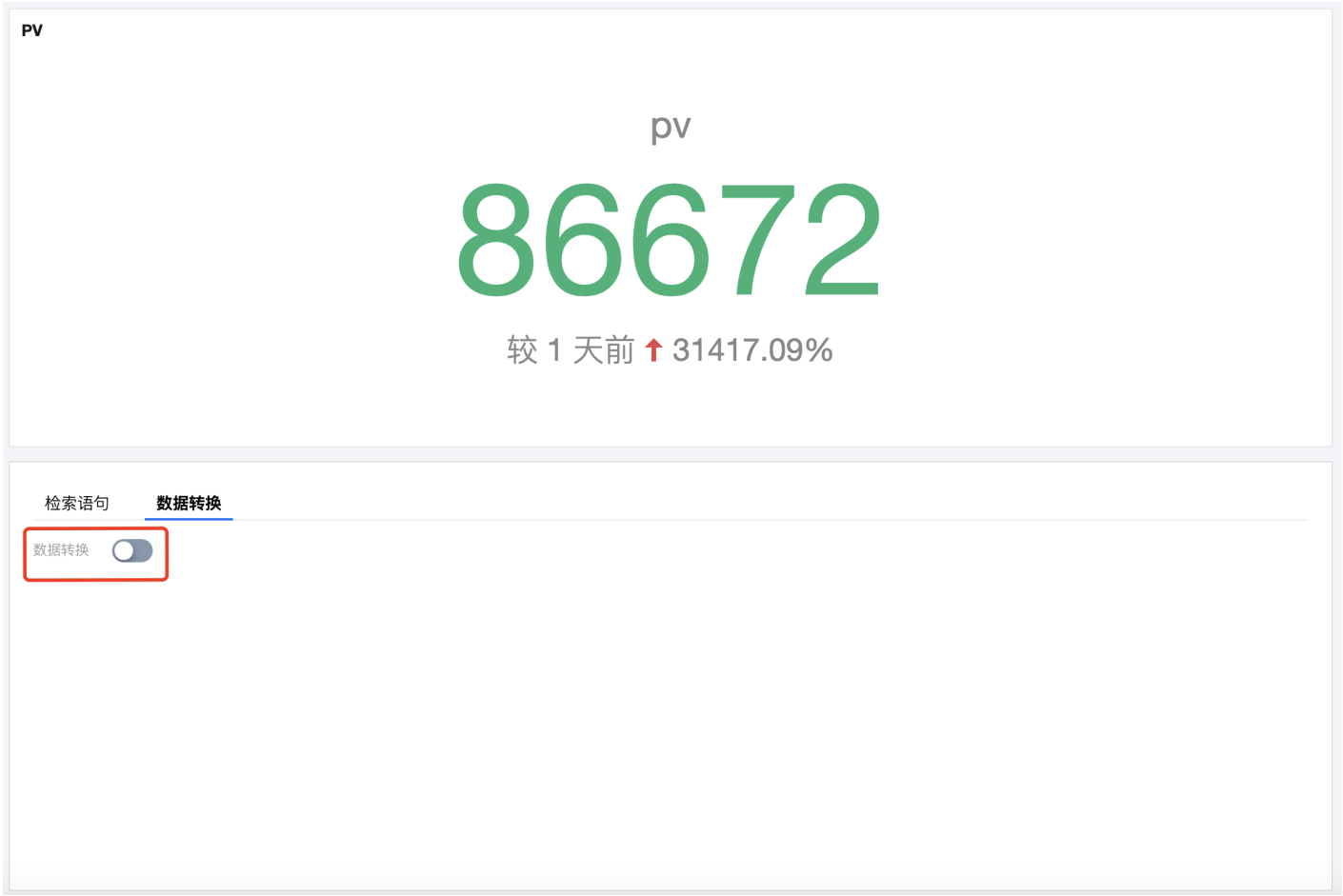
最近更新时间：2022-05-20 15:06:43

数据转换支持用户对检索结果进行二次处理，包括字段类型修改、选择制图字段、合并分组等。可以在不修改 SQL 语句的情况下，满足制图的需求。

开启数据转换

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**检索分析**，进入检索分析页面。
3. 选择**图表分析**页签，将下方的**数据转换**设置为 。

4. 在**仪表盘图表编辑**页面，选择**数据转换**页签，将**数据转换**设置为 。



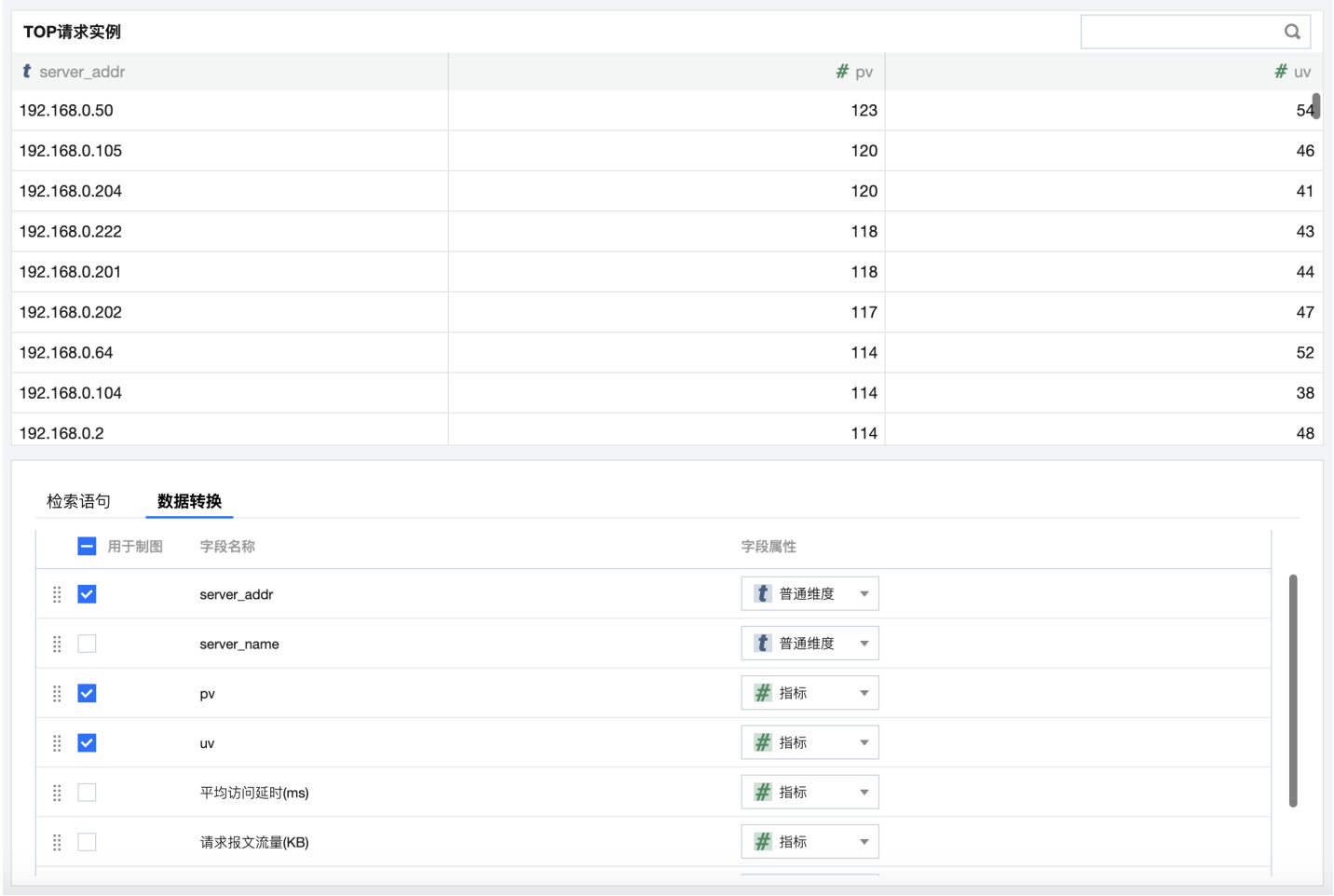
什么时候使用数据转换

执行一个 SQL 分析语句后，想要在不同的图表类型上做可视化。因为不同的图表需要的字段数量与属性不同，在不匹配时就会出现以上情况。此时，用户可以通过修改 SQL，使其适配图表要求。也可以通过数据转换配置，对已有结果进行二次处理，满足制图需求。

使用数据转换

选择制图字段

通过 SQL 检索出了以上列表中的结果，此时通过勾选字段列表中的字段，可以控制图表选中指定字段来制图，等价于 select。以下为选中部分字段后的结果：

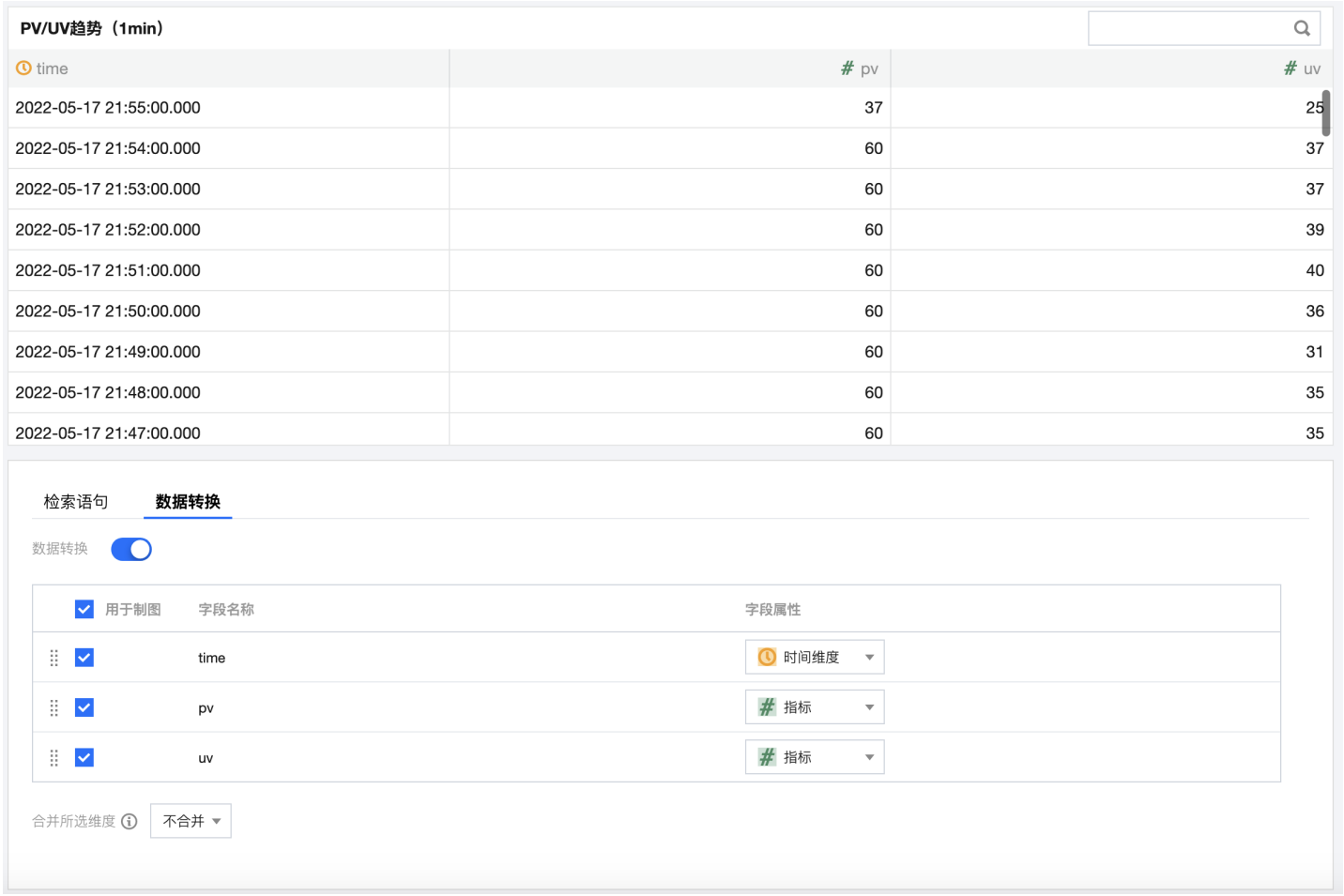


转换字段类型

SQL统计的结果中，每个字段都有一个默认的类型，图表依据类型识别 #数值类型 的字段为指标， t字符串类型 的字段为普通维度， 时间类型 的字段作为时间维度。并将其与制图所需的字段属性对应。例如在时序图中，就需要时间维度字段作为X轴，指标字段作为Y轴。

上图示例中的 time 字段被当做了一个普通维度，此时，无法满足时序图的字段属性要求。而通过修改 time 字段属性为时间类型，会发现 time 字段转变为时间格式（等价于 CAST 函数），此时即可使用时序图。

常见用户使用 histogram 函数处理字段，其结果可以是非标准的时间格式，因此若默认为普通维度，无法使用时序图。需要提前通过 cast 函数转换为时间类型，或者通过数据转换修改字段属性为时间维度。

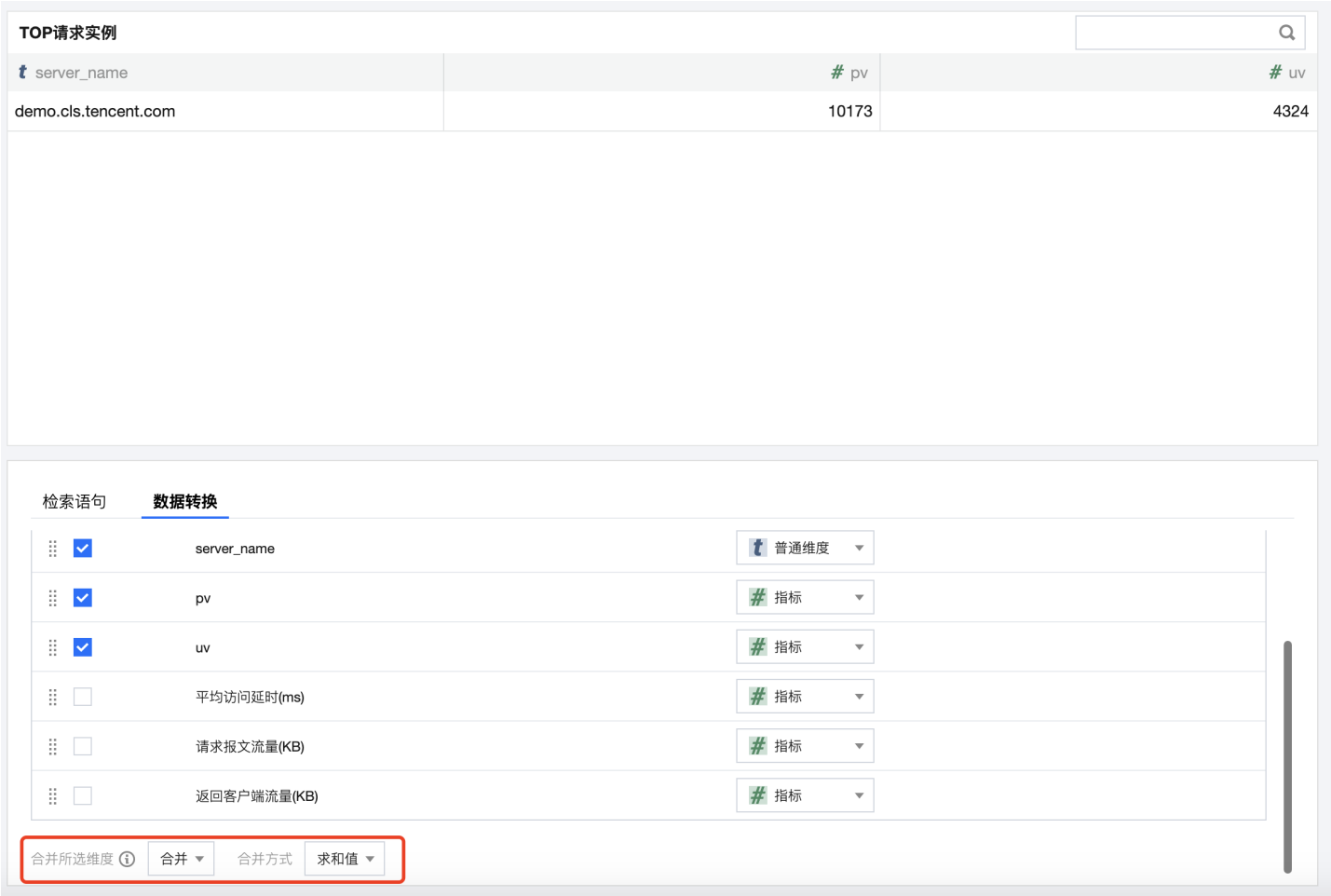

PV/UV趋势（1min）
time
pv
uv
2022-05-17 21:55:00.000 37 25
2022-05-17 21:54:00.000 60 37
2022-05-17 21:53:00.000 60 37
2022-05-17 21:52:00.000 60 39
2022-05-17 21:51:00.000 60 40
2022-05-17 21:50:00.000 60 36
2022-05-17 21:49:00.000 60 31
2022-05-17 21:48:00.000 60 35
2022-05-17 21:47:00.000 60 35
检索语句
数据转换
数据转换
用于制图
字段名称
字段属性
time
时间维度
pv
指标
uv
指标
合并所选维度
不合并

合并分组

TOP请求实例

server_addr	server_name	pv	uv
192.168.0.50	demo.cls.tencent.com	123	54
192.168.0.105	demo.cls.tencent.com	120	46
192.168.0.204	demo.cls.tencent.com	120	41
192.168.0.222	demo.cls.tencent.com	118	43
192.168.0.201	demo.cls.tencent.com	118	44
192.168.0.202	demo.cls.tencent.com	117	47
192.168.0.64	demo.cls.tencent.com	114	52
192.168.0.104	demo.cls.tencent.com	114	38
192.168.0.2	demo.cls.tencent.com	114	48

通过 SQL 检索出以上结果，按照 "server_addr" 与 "server_name" 进行分组，统计每个分组的 PV、UV。此时，想要在当前的结果上对 "server_name" 进行合并分组统计，可以通过隐藏 "server_addr" 字段，然后合并所选的普通维度 "server_name"，等价于 Group By 函数。得到的结果如下：


TOP请求实例
server_name
pv
uv
demo.cls.tencent.com
10173
4324
检索语句
数据转换
server_name
普通维度
pv
指标
uv
指标
平均访问延时(ms)
指标
请求报文流量(KB)
指标
返回客户端流量(KB)
指标
合并所选维度
合并
合并方式
求和值

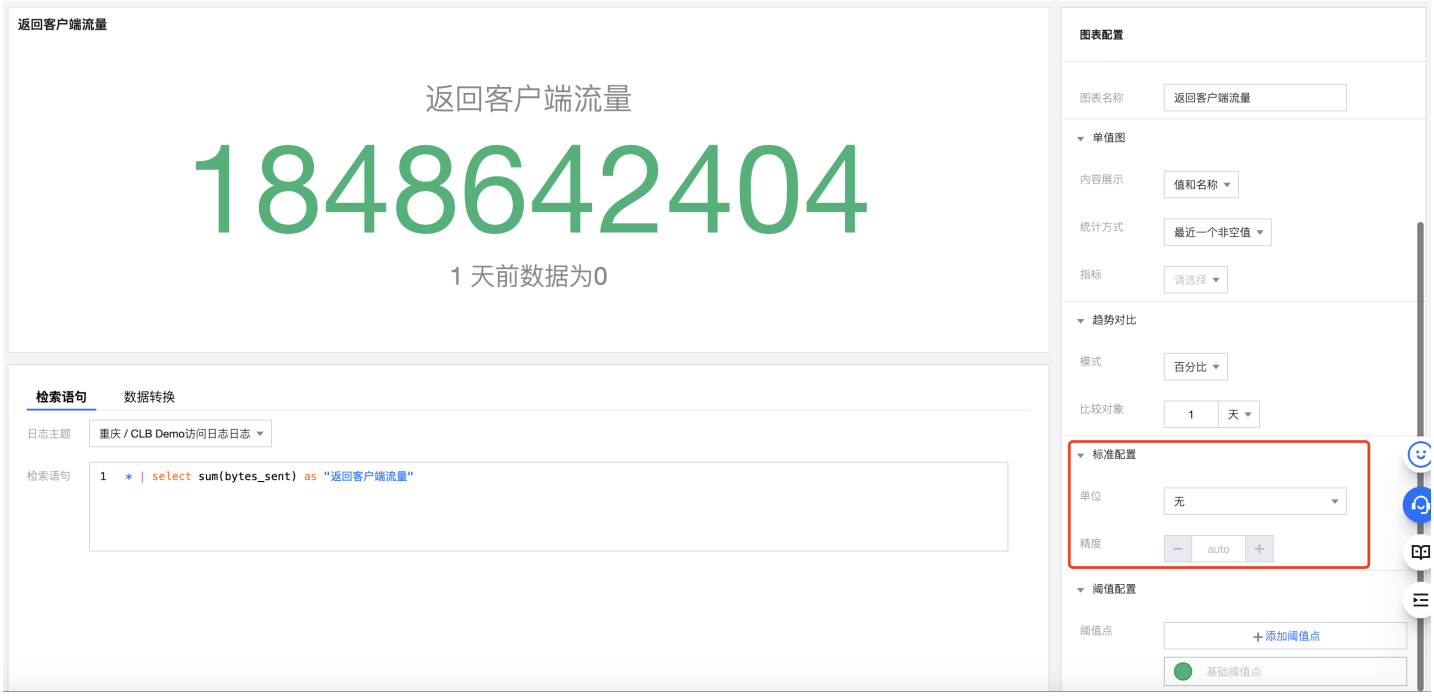
单位配置

最近更新时间：2022-05-20 15:06:38

图表支持自动单位换算，选择一个原始单位后，当数值满足换算进率时，自动换算为更高一级别的单位。单位支持精度配置。

单位配置

使用 SQL 聚合的结果默认是没有单位的数值。当遇到以下这种情况时，需要采用单位换算，提高数据的可读性。



选择一个单位并自动换算

在日志服务（Cloud Log Service，CLS）图表分析中，SQL 聚合结果默认是没有单位的原始数值，需要用户指定一个基本单位，如例中的数值单位 byte。用户选择一个初始单位后，当数值满足更高级单位的进率，就会自动换算。如例中所示，单位从 bytes 自动换算 GiB。

单位配置里的精度影响小数位数，默认 Auto 保留2位小数，通过修改可以控制保留的小数位数。

手动输入单位

在使用中，如果数据具有特殊单位，而单位配置选项里没有时。用户可以通过手动输入，自定义添加一个单位（带 custom 前缀）。但请注意，自定义的单位是固定不变的，不会自动换算。因此，若不希望触发单位自动换算，统一使用原始单位，可以通过手动输入添加一个固定单位。

仪表盘
创建仪表盘

最近更新时间：2022-02-23 16:17:14

仪表盘提供的数据分析监控的全局视图。您可以在仪表盘查看多个基于查询与分析结果的统计图表。本文介绍如何创建仪表盘以其相关操作。

操作步骤

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**仪表盘 > 仪表盘列表**，进入仪表盘列表页面。
3. 在仪表盘列表页面，单击**新建**。
4. 在弹出的窗口中，输入仪表盘名称，单击**确定**。
5. 单击**保存**。

相关操作

创建仪表盘后，您可以在仪表盘中执行以下操作：

操作	说明
添加图表	仪表盘也提供统计图表创建的入口，支持简易图表创建和自定义图表创建，详情请见 添加图表。
删除图表	删除已添加的图表。
编辑图表	进入图表编辑页面。
复制图表	支持复制到当前仪表盘或其他仪表盘。
图表数据导出	支持图表数据以 CSV 格式导出到本地。
在检索分析查看	支持快速添加该图表使用的检索语句，日志主题等信息到检索分析页面。
快速添加告警	支持快速添加该图表使用的检索语句，日志主题等信息到告警编辑页面。
全屏浏览	支持单个图表与整个仪表盘的全屏浏览。
刷新	支持自动时间间隔与手动数据刷新。
添加模板变量	模板变量支持仪表盘更灵活的定义与修改相关数据查询与过滤的参数，可以提高仪表盘的复用率和分析的颗粒度，详情请见 模板变量。
查看/编辑仪表盘	支持仪表盘查看与布局的编辑。

您也可以仪表盘管理页面执行以下操作：

操作	说明
新建仪表盘	进入仪表盘创建页面。
删除仪表盘	删除已存在的仪表盘。
编辑仪表盘标签	支持单个与批量仪表盘标签编辑。
打开仪表盘	打开所选仪表盘，进入仪表盘查看/编辑页面。
搜索与过滤仪表盘	支持基于仪表盘 ID、名称、地域、标签等属性的复合过滤查询。

添加图表

最近更新时间：2022-04-07 10:43:05

日志服务（Cloud Log Service，CLS）支持对检索分析结果进行可视化图表统计，并支持将统计图表保存到仪表盘以方便持续监控查看，同时仪表盘也提供统计图表创建的入口，支持简易图表创建和自定义图表创建。

操作步骤

通过检索分析添加图表

> 注意：
> 此方式需通过检索数据生成图表内容，详情请参考 日志分析。

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**检索分析**，进入检索分析管理页面。
3. 选择**图表分析**，单击**添加到仪表盘**。
4. 在弹出的窗口中，填写信息，单击**确定**。

表单元素	说明
图表名称	图表名称。
目标仪表盘	图表将要添加到的仪表盘类别。 ◦ 选择已有仪表盘，会添加图表到已有仪表盘中。 ◦ 选择新建仪表盘，则需要新建一个仪表盘，并将图表添加进去。
仪表盘	仪表盘名称。

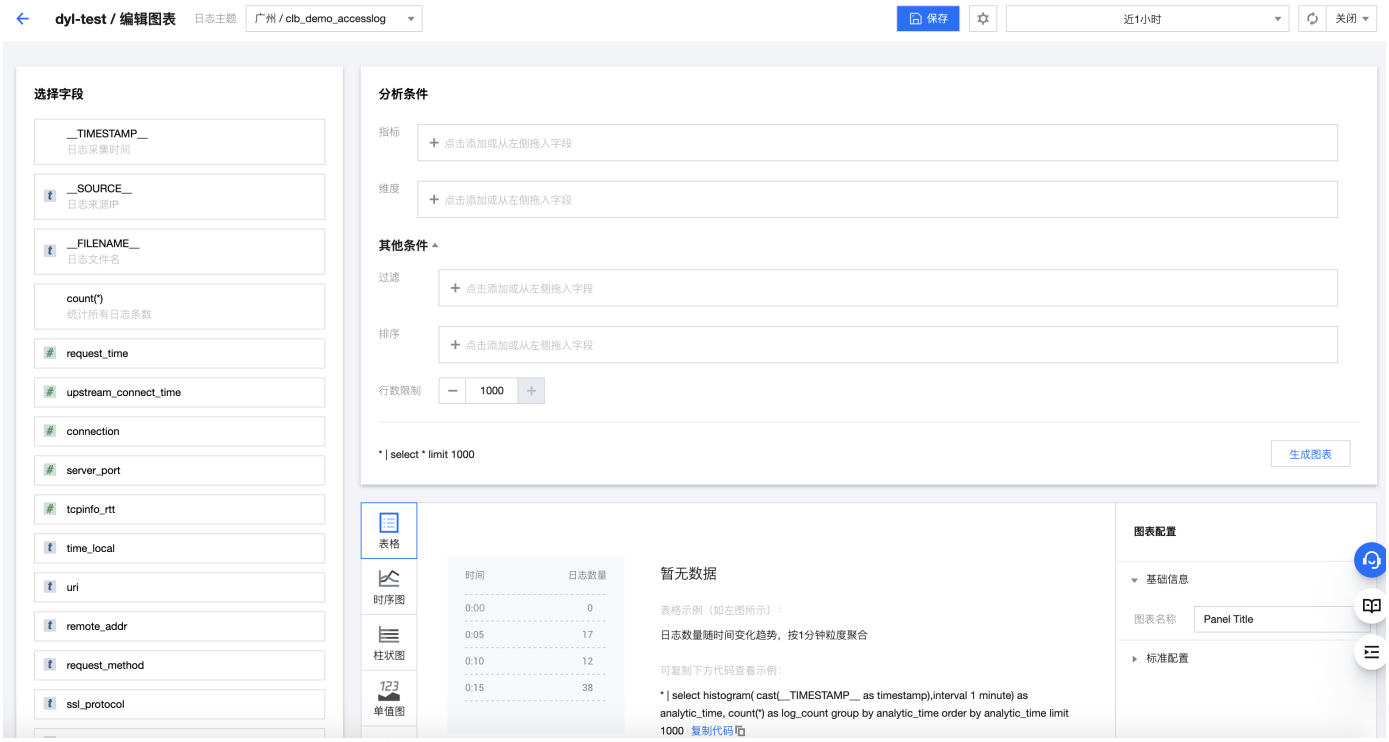
通过仪表盘添加图表

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**仪表盘**，进入仪表盘管理页面。
3. 单击需要操作的仪表盘 ID/名称，进入该仪表盘详情页面。

> 说明：
> 如果您没有仪表盘，请 新建仪表盘。

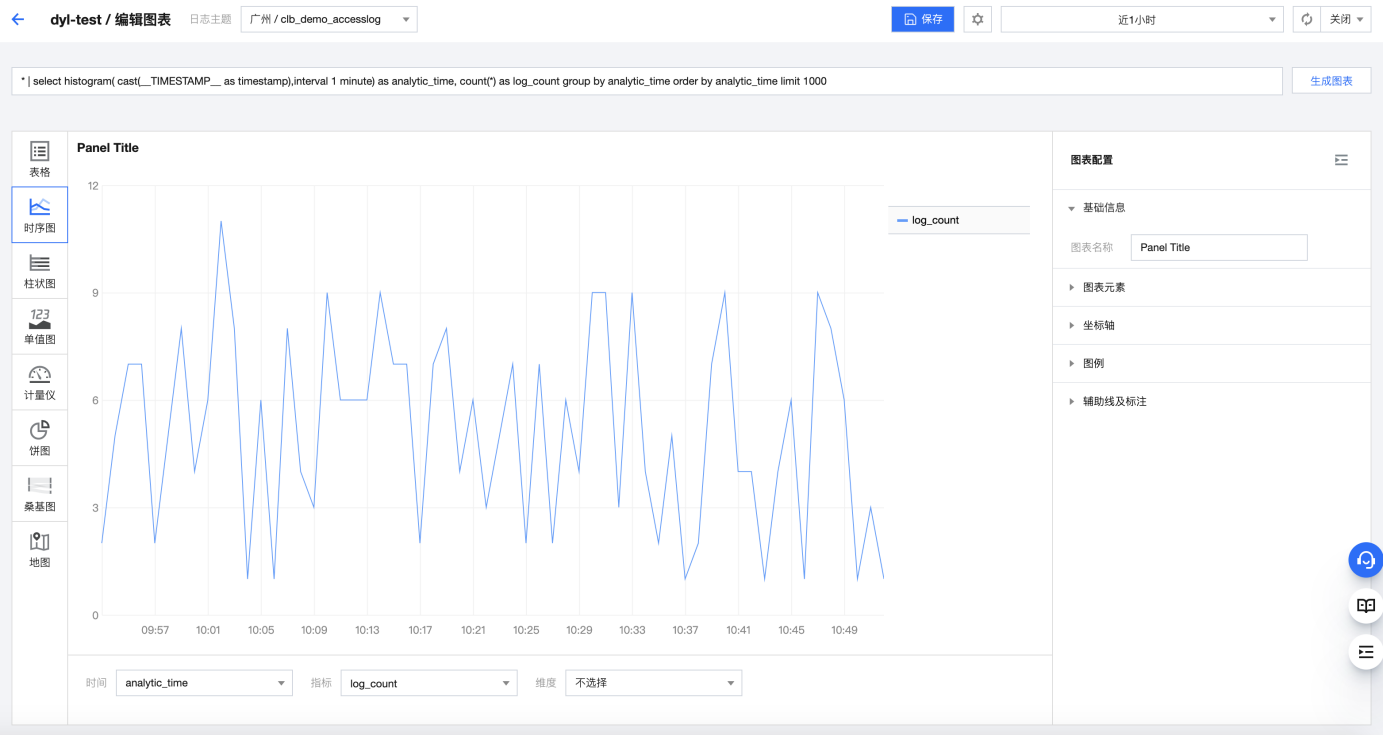
4. 单击**新建**，根据实际需求选择图表类型，创建图表。

- **简易图表：通过可视化拖拽方式轻松创建图表，但统计分析功能较弱。**



图表元素	说明
选择字段	展示当前日志主题字段信息的列表，可以单击或直接鼠标拖拽添加到右侧条件输入框。
指标	指标是衡量事物某个特征的参数，一般为数值类型字段。可以从左侧字段列表拖入字段，也可以单击“+”号触发下拉菜单选择字段。字段填充后可以单击字段旁的设置图标,修改字段的聚合计算方式，系统默认使用 “平均值” 。
维度	维度是分析指标的视角分组，一般为字符串类型描述事物属性的字段。可以从左侧字段列表拖入字段，也可以单击“+”号触发下拉菜单选择字段。
过滤	过滤字段用于数据的属性过滤。可以从左侧字段列表拖入字段，也可以单击“+”号触发下拉菜单选择字段。字段填充后可以单击字段旁的设置图标，修改字段的过滤方式，系统默认使用“字段存在” 。
排序	排序字段用于统计结果的排列。排列字段仅支持依据条件字段排列，其他非条件字段不可用。建议单击“+”号触发下拉菜单选择字段。字段填充后可以单击字段旁的设置图标,修改排序方法，系统默认使用 “升序” 。
行数限制	行数限制用于统计结果的数量过滤。设置后将按照排序从前往后显示设定数量的统计结果。行数限制取值范围为1 – 1000，系统默认使用“1000”。

- 自定义图表：通过自定义检索语句创建图表，则支持全部统计分析功能。



填写检索分析语句后，单击**生成图表**即可。

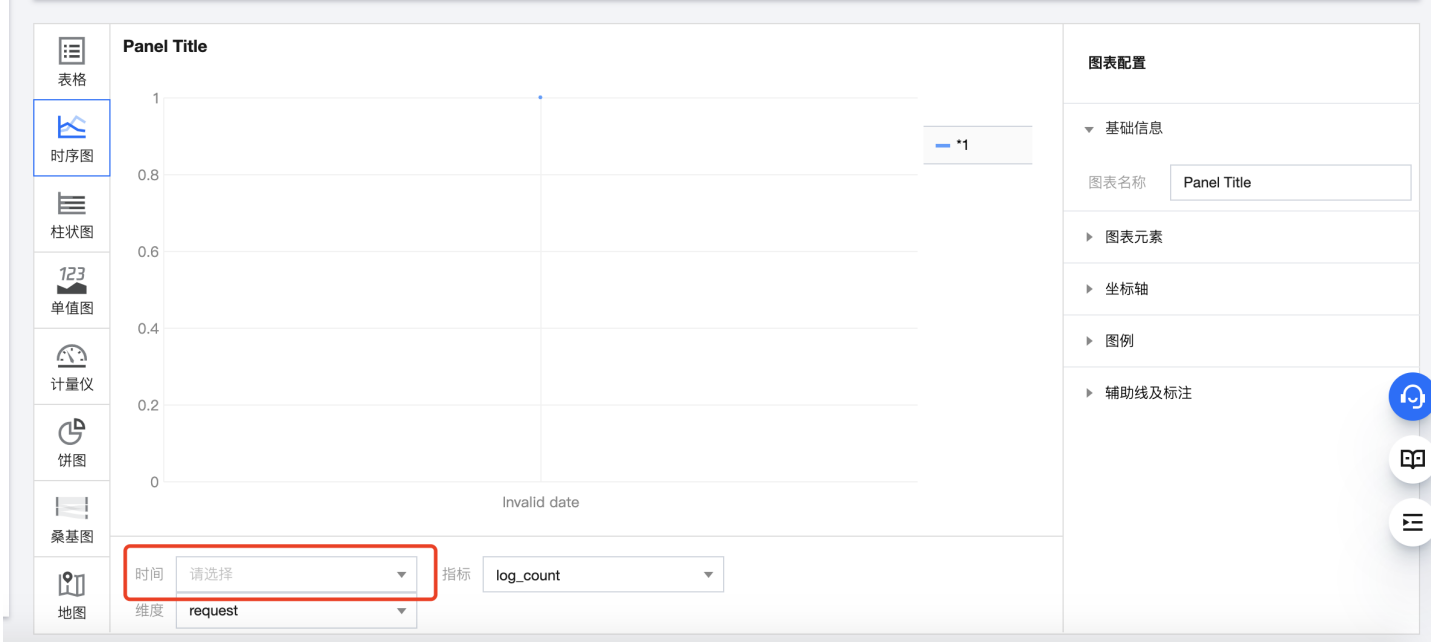
5. 检查数据配置是否符合要求，系统将会根据图表数据结构要求自动填充。但必要时用户可手动调整数据结构配置。

6. 单击**保存**。

常见问题

配置完成后，单击生成图表或切换图表类型，发现图表没有生成，该怎么办？

当检索的数据结构与图表所要求的数据结构一致时，系统会默认自动填充。如果数据结构不一致，将因为缺少必要字段，导致图表无法生成（例如下图的缺少时间字段），因此，当出现这种情况时，请检查数据是否符合要求。



模板变量

最近更新时间：2022-03-28 10:48:13

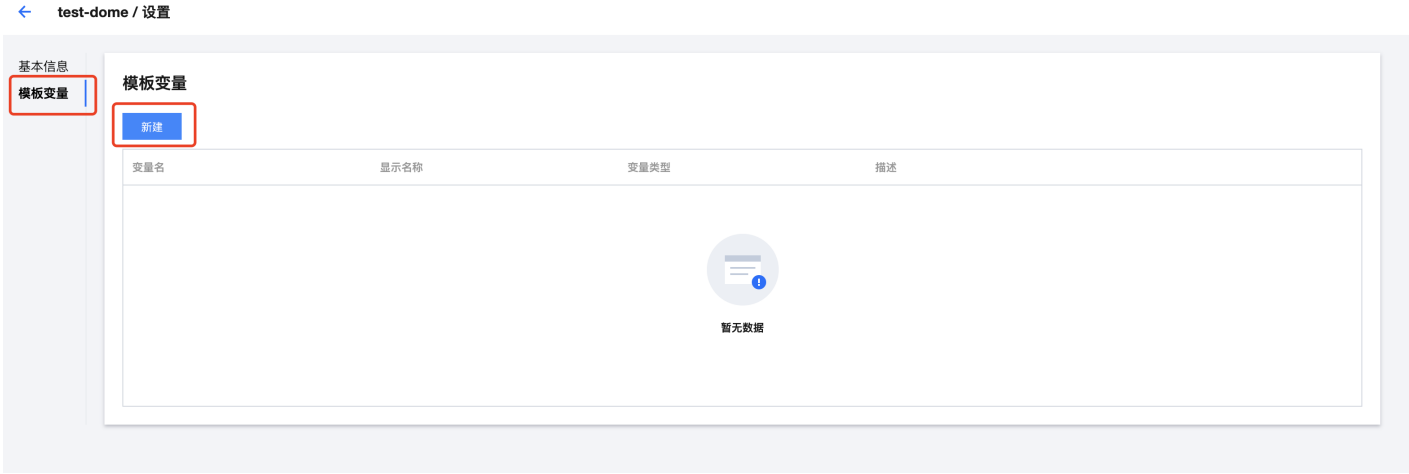
变量类型	说明	生效范围
数据源	数据源变量支持批量切换仪表盘内图表的数据源，适用于一个仪表盘应用到多个日志主题、仪表盘内数据进行蓝绿对比等场景	仪表盘内使用该变量的图表
快速过滤	快速过滤变量支持通过指定字段对仪表盘内所有图表的数据进行过滤，相当于在图表查询语句中增加了过滤条件	仪表盘内所有图表

数据源变量的配置

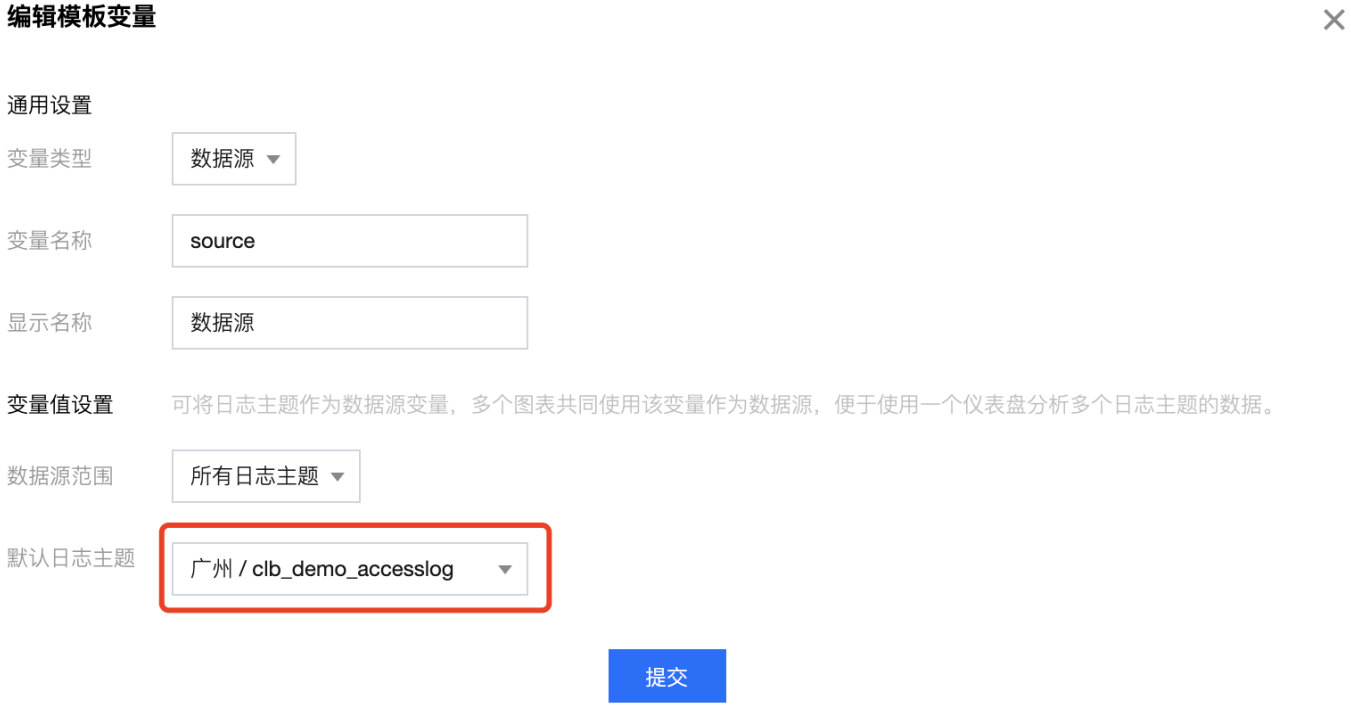
操作步骤

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**仪表盘**，进入仪表盘管理页面。
3. 单击需要操作的仪表盘 ID/名称，进入该仪表盘详情页面。
4. 单击顶部的  ，进入设置页面。

5. 选择**模板变量**，单击**新建**。



6. 在弹出的窗口中，设置模板变量信息，单击**提交**。

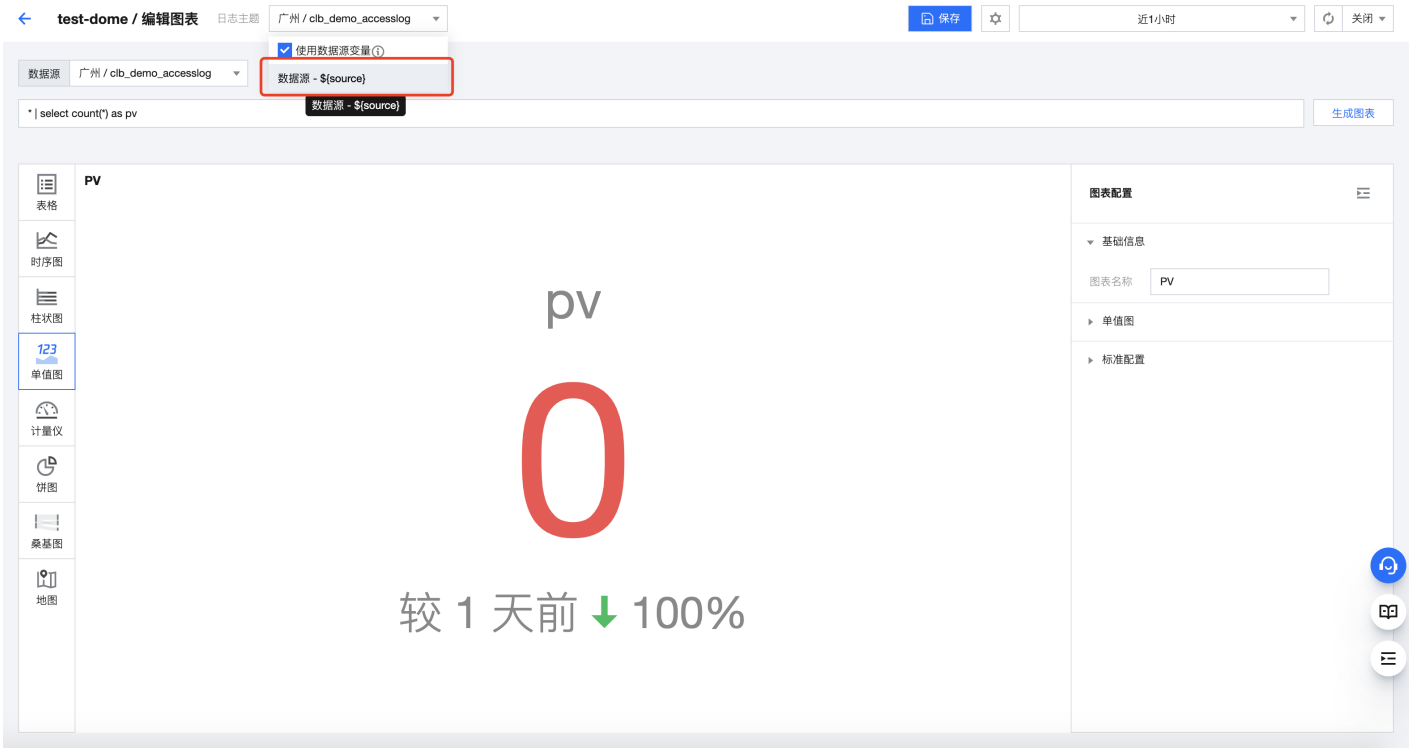


表单元素	说明
变量类型	变量的类别，不同的类别对应不同的配置项与应用场景，此处选择数据源。
变量名称	查询检索语句中变量的命名，仅支持字母与数字。
显示名称	仪表盘上变量的显示名称，非必填项目，为空时自动使用变量名称为显示名称。
数据源范围	变量值的可选范围，当前仅支持“所有日志主题”，即不限制数据源范围。
默认日志主题	默认使用的日志主题。

7. 返回仪表盘详情页面，单击**更多 > 编辑图表**。

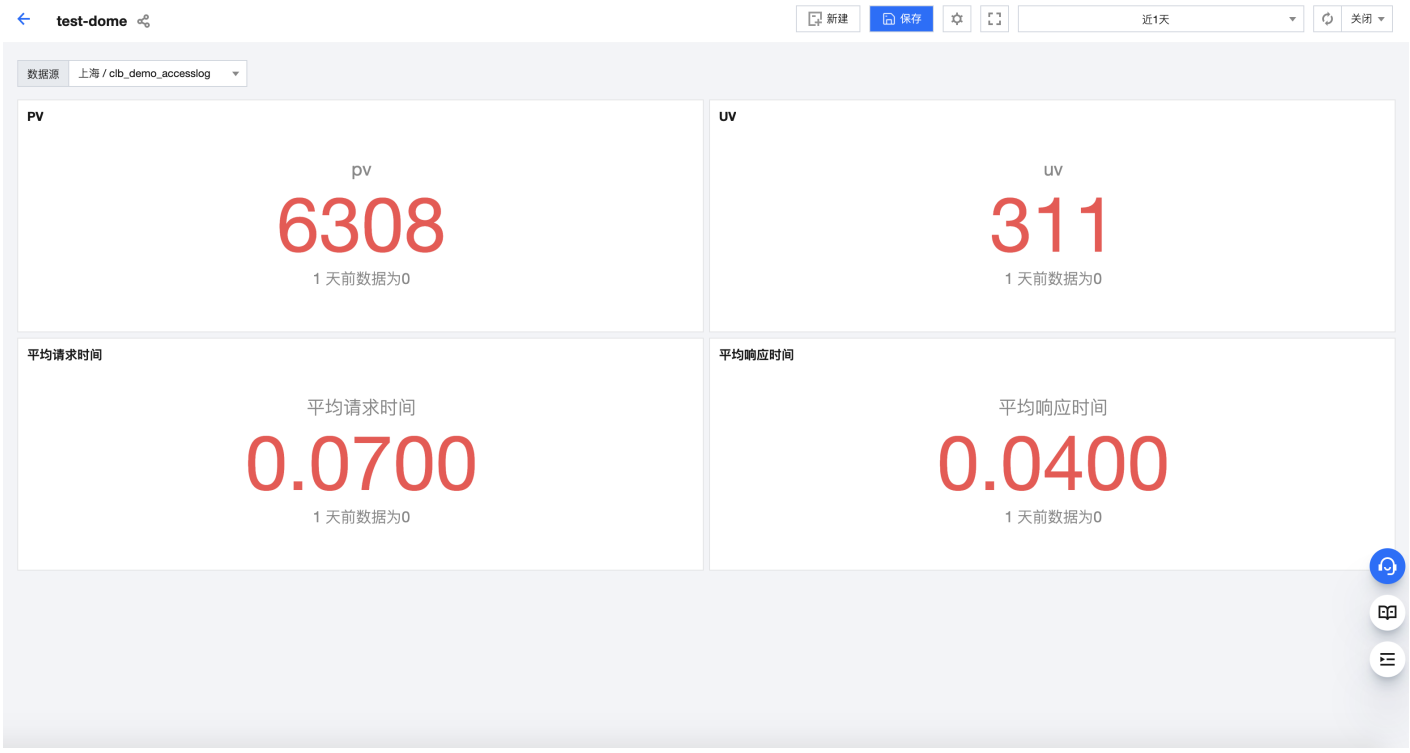
> **说明：**
> 如果您的仪表盘中没有图表，请 添加图表。

8. 在编辑图表上方，单击**日志主题**，勾选**使用数据源变量**，选择刚新建的模板变量。



9. 单击**保存**。

0. 返回仪表盘详情页面，单击上方的数据源变量下拉框，将日志主题切换为其他日志主题，使用该变量的图表将自动切换数据源。



常见问题

配置并使用数据源变量后，为什么没有生效，或只有部分图表生效?

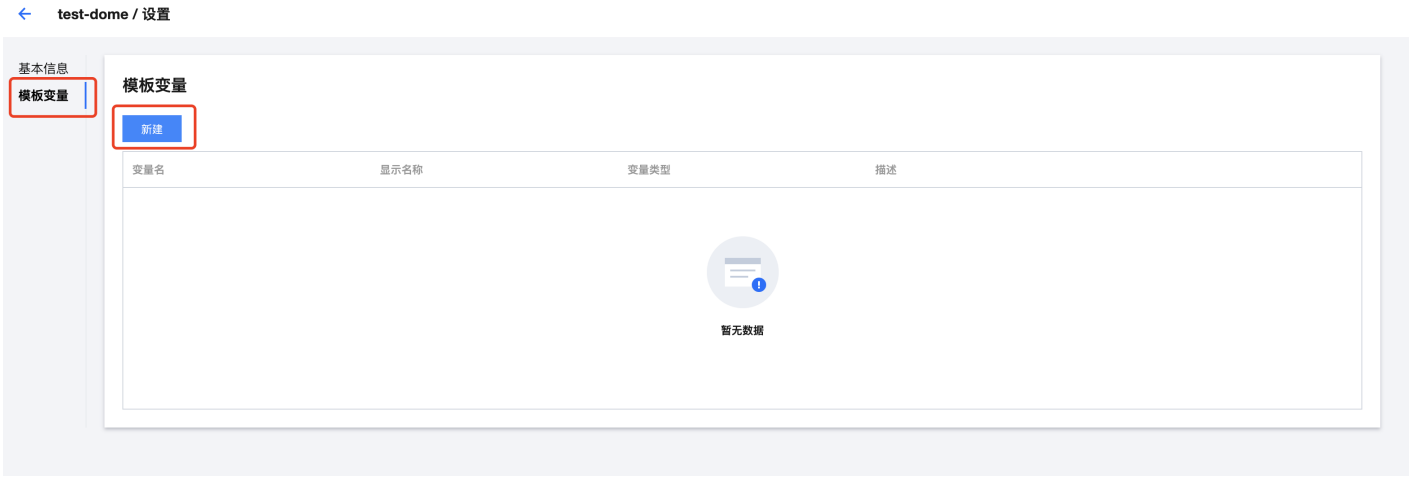
数据源变量并不会直接针对仪表盘内所有图表生效，只有在图表编辑页面中使用了该变量的图表才会生效。

快速过滤变量的配置

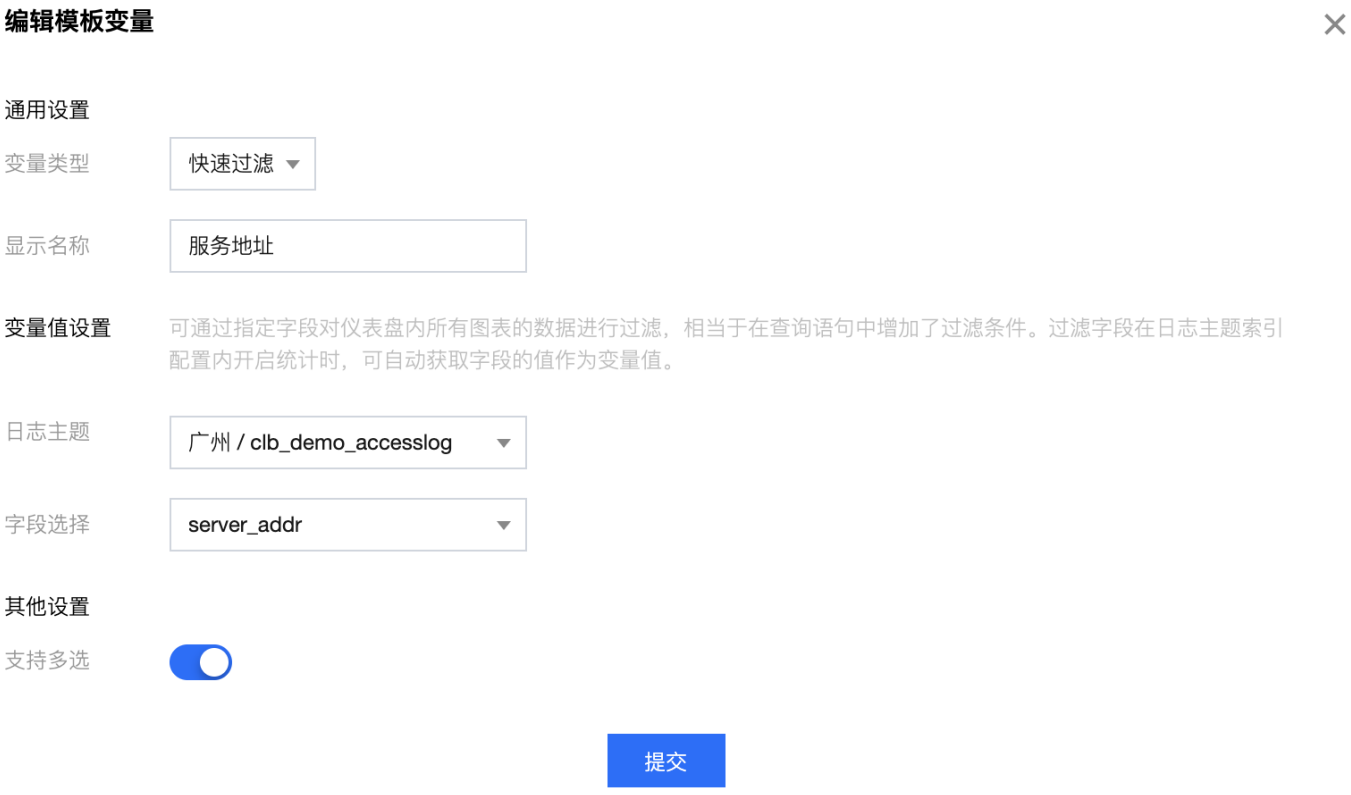
1. 登录 日志服务控制台。

2. 在左侧导航栏中，单击**仪表盘**，进入仪表盘管理页面。
3. 单击需要操作的仪表盘 ID/名称，进入该仪表盘详情页面。
4. 单击顶部的  ，进入设置页面。

5. 选择**模板变量**，单击**新建**。

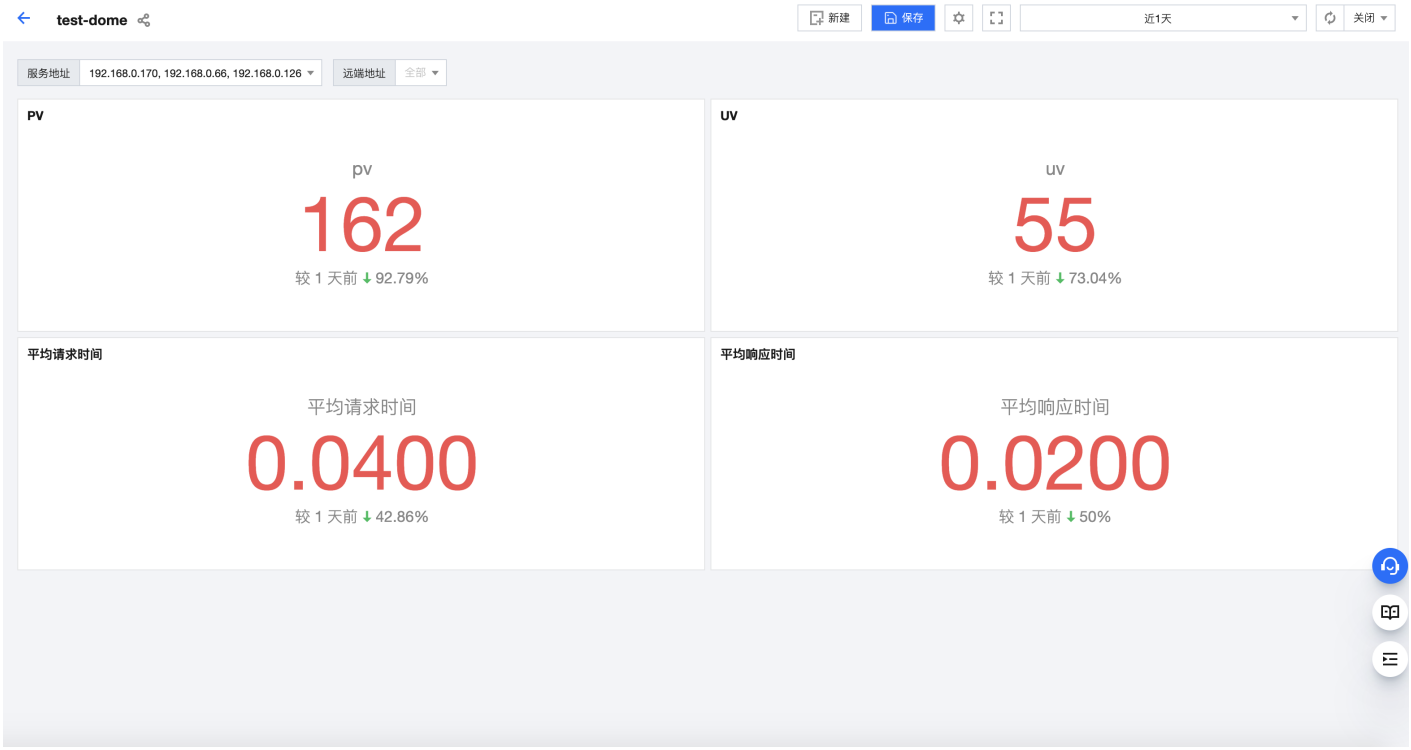


6. 在弹出的窗口中，设置模板变量信息，单击**提交**。



表单元素	说明
变量类型	变量的类别，不同的类别对应不同的配置项与应用场景，此处选择快速过滤。
显示名称	界面上变量控件的的命名，非必填项目，空置时自动使用选择字段为显示名称。
日志主题	变量字段来源的日志主题。
字段选择	过滤字段。
支持多选	开启后可以选择多个变量值作为过滤条件。

7. 返回仪表盘详情页面，单击变量控件，选择过滤字段，仪表盘数据将刷新为过滤后内容。



使用案例

在仪表盘内分析不同应用接口的性能指标（快速过滤变量）

需求场景

日志主题 A 为某应用的 nginx 访问日志，需要通过仪表盘查看该**应用整体**及**指定某个接口**的吞吐量、错误请求数和响应时间。样例日志如下：

```
body_bytes_sent:1344
client_ip:127.0.0.1
host:www.example.com
http_method:POST
http_referer:www.example.com
http_user_agent:Mozilla/5.0
proxy_upstream_name:proxy_upstream_name_4
remote_user:example
req_id:5EC4EE87A478DA3436A79550
request_length:13506
request_time:1
http_status:201
time:27/Oct/2021:03:25:24
upstream_addr:219.147.70.216
upstream_response_length:406
upstream_response_time:18
upstream_status:200
interface:proxy/upstream/example/1
```

解决方案

1. 创建仪表盘。
2. 针对应用性能指标，分别创建三个图表（时序图）。其对应的查询语句分别如下：
   - 吞吐量：

```
* | select histogram( cast(__TIMESTAMP__ as timestamp),interval 1 minute) as analytic_time, count(*) as pv group by analytic_time order by analytic_time limit 1000
```

- **错误请求数：**

```
http_status:>=400 | select histogram( cast(__TIMESTAMP__ as timestamp),interval 1 minute) as analytic_time, count(*) as pv_lost group by analytic_time order by analytic_time limit 1000
```

- **平均响应时间：**

```
* | select histogram( cast(__TIMESTAMP__ as timestamp),interval 1 minute) as analytic_time, avg(request_time) as response_time group by analytic_time order by analytic_time limit 1000
```

3. **添加模板变量。**
   - **变量类型：快速过滤**
   - **显示名称：接口名称**
   - **日志主题：日志主题 A**
   - **字段选择：interface**
4. **返回仪表盘详情页面，即可在页面顶部看到该变量。**
   - **接口名称没有指定值时，表示不进行数据过滤，仪表盘内各个图表展示的是全部数据，即应用的整体性能指标。**

- **接口名称**指定具体值时，仪表盘内所有图表以该接口作为过滤条件进行数据过滤，展示该接口的性能指标。

在仪表盘内分别查看生产环境及测试环境的性能指标（数据源变量）

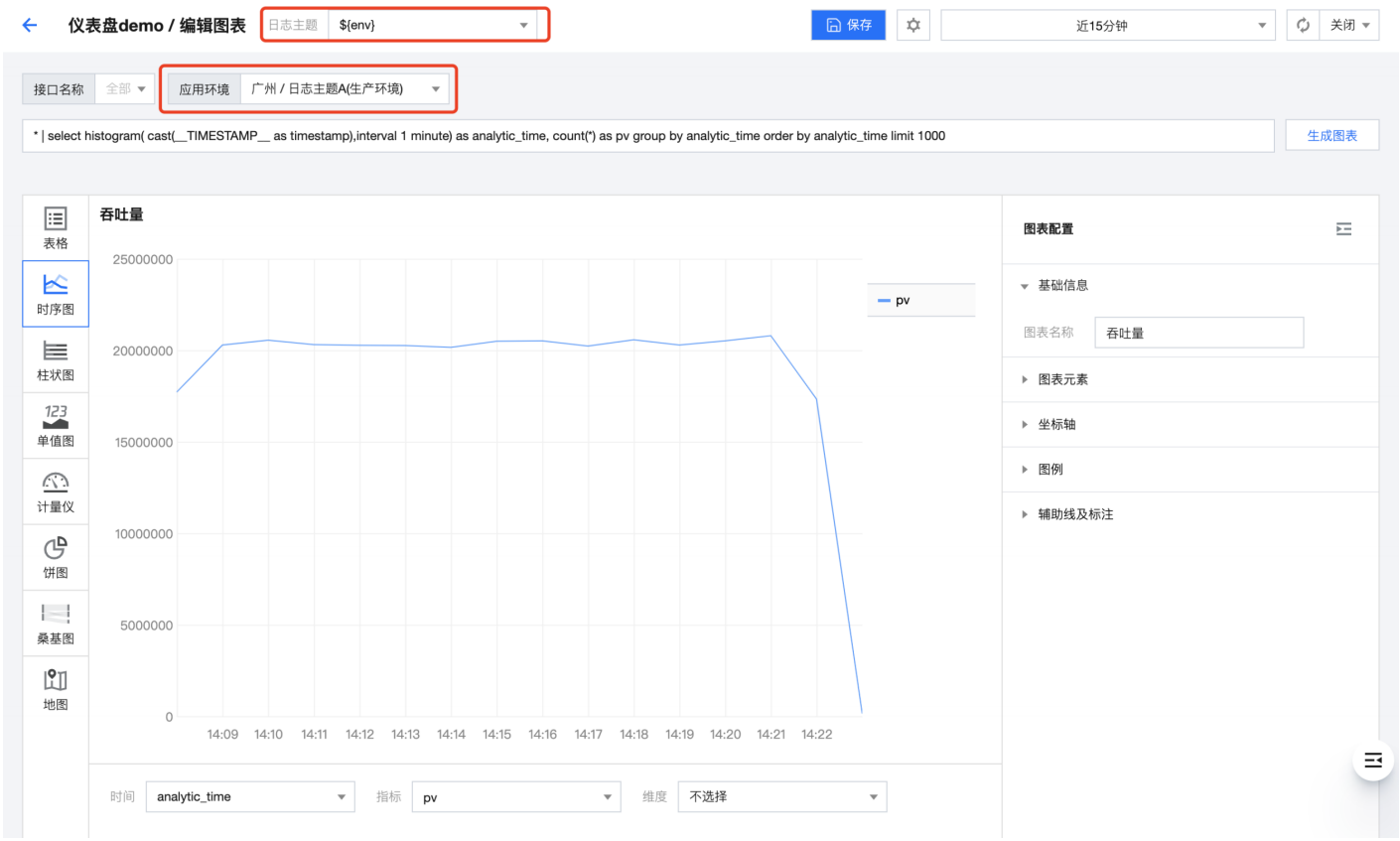
需求场景

某应用具备生产环境及测试环境，分别将日志采集到“日志主题 A（生产环境）”和“日志主题 B（测试环境）”中。因此，在应用的开发、测试及运维过程中，需要同时关注两个环境的性能指标。

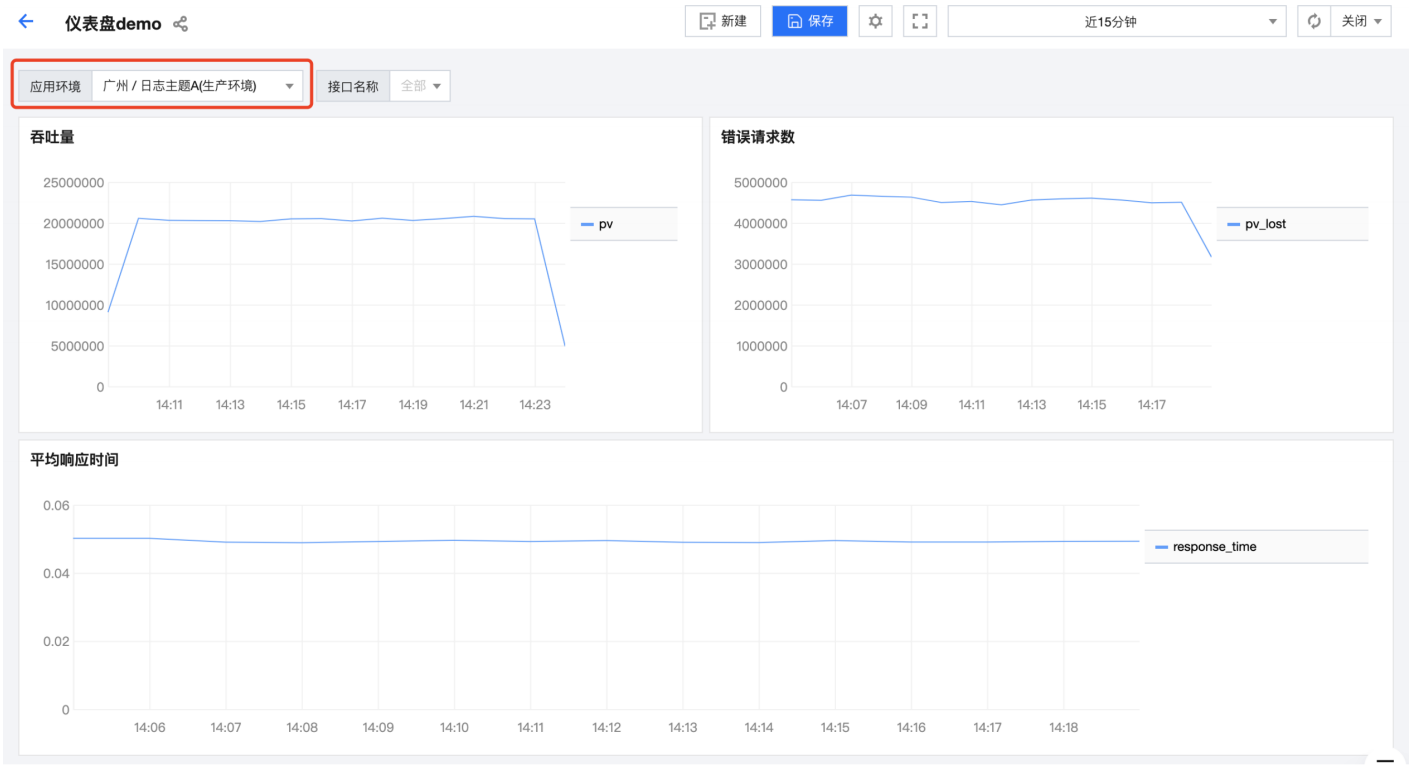
解决方案

1. 创建仪表盘。
2. 添加模板变量。
   - 变量类型：数据源
   - 变量名称：env
   - 显示名称：应用环境
   - 数据源范围：所有日志主题
   - 默认日志主题：日志主题 A（生产环境）
3. 添加图表。

   在**日志主题**下拉菜单中，勾选“使用数据源变量”，选择上一步中创建的`${env}`变量。图表将使用该变量的值作为当前图表的数据源，即日志主题 A（生产环境）。



4. 重复执行步骤3，添加其他图表。
5. 返回仪表盘详情页面，并在该页面顶部单击数据源变量“应用环境”，在该变量的下拉菜单中切换日志主题。同时，使用该变量的图表也将切换日志主题。



图标自定义时间

最近更新时间：2022-07-01 18:46:32

操作场景

仪表盘图表支持自定义时间配置。

与自定义时间相对的是仪表盘全局时间。仪表盘图表创建时，默认使用仪表盘全局时间，其数据范围受仪表盘全局时间控制。修改仪表盘全局时间，所有关联图表的时间范围都会发生改变。

而配置自定义时间后，图表时间独立控制，不再随全局时间的变化而变化，仪表盘的图表间可以有不同的时间范围，支持更丰富的对比场景。

操作步骤

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**查看仪表盘**，进入仪表盘页面。
3. 单击编辑图表或添加图表，进入图表编辑页面。
4. 单击图表时间范围，开启**图表自定义时间**。

5. 查看图表，图表时间与仪表盘时间不同。

预置仪表盘

最近更新时间：2022-06-27 09:03:12

操作场景

日志服务（Cloud Log Service，CLS）支持接入多种云产品标准日志，详情见 云产品日志接入。针对这些云产品日志，CLS 提供开箱即用的日志分析仪表盘，用户接入日志后即可快速分析。

此外，可以通过 CLS 提供的免费 云产品 Demo 日志 快速体验预置仪表盘。

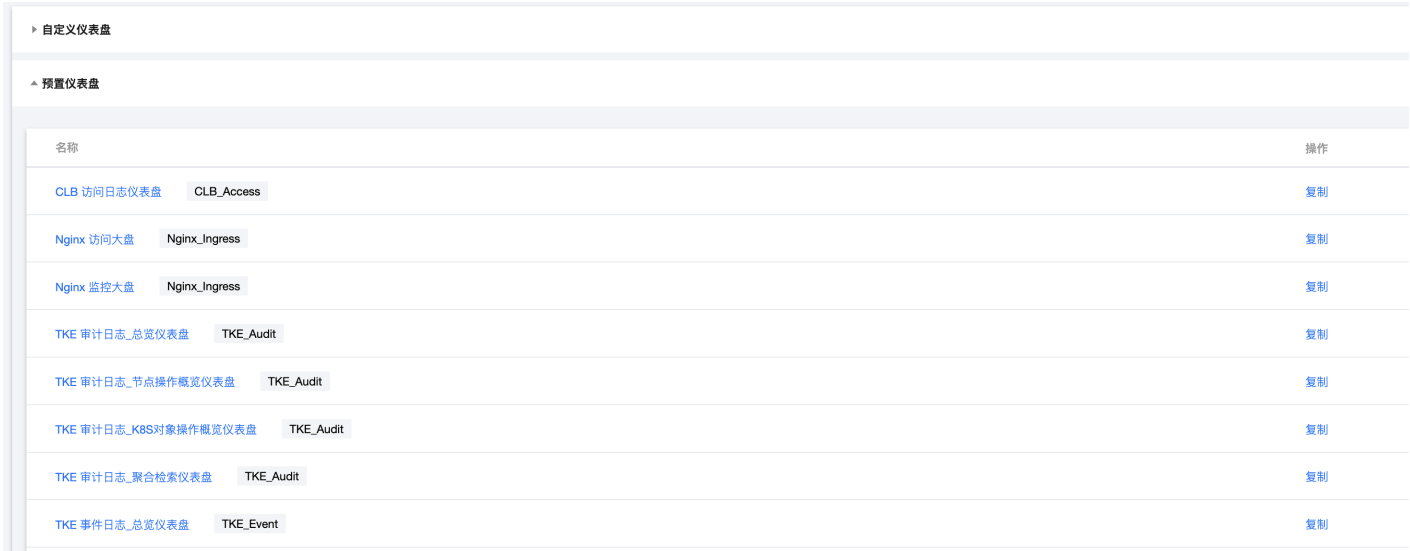
预置仪表盘列表

云产品	预置仪表盘
负载均衡（Cloud Load Balancer，CLB）	CLB 访问日志仪表盘
Nginx访问日志	Nginx 访问大盘 Nginx 监控大盘
内容分发网络（Content Delivery Network，CDN）	CDN 访问日志-质量监控分析仪表盘 CDN 访问日志-用户行为分析仪表盘
对象存储（Cloud Object Storage，COS）	COS 访问日志分析仪表盘
网络流日志（Flow Logs，FL）	ENI 流日志-高级分析仪表盘 CCN 流日志-高级分析仪表盘
容器服务（Tencent Kubernetes Engine，TKE）	TKE 审计日志-总览仪表盘 TKE 审计日志-节点操作概览仪表盘 TKE 审计日志-K8S 对象操作概览仪表盘 TKE 事件日志-总览仪表盘 TKE 事件日志-异常事件聚合检索仪表盘

操作步骤

查看预置仪表盘

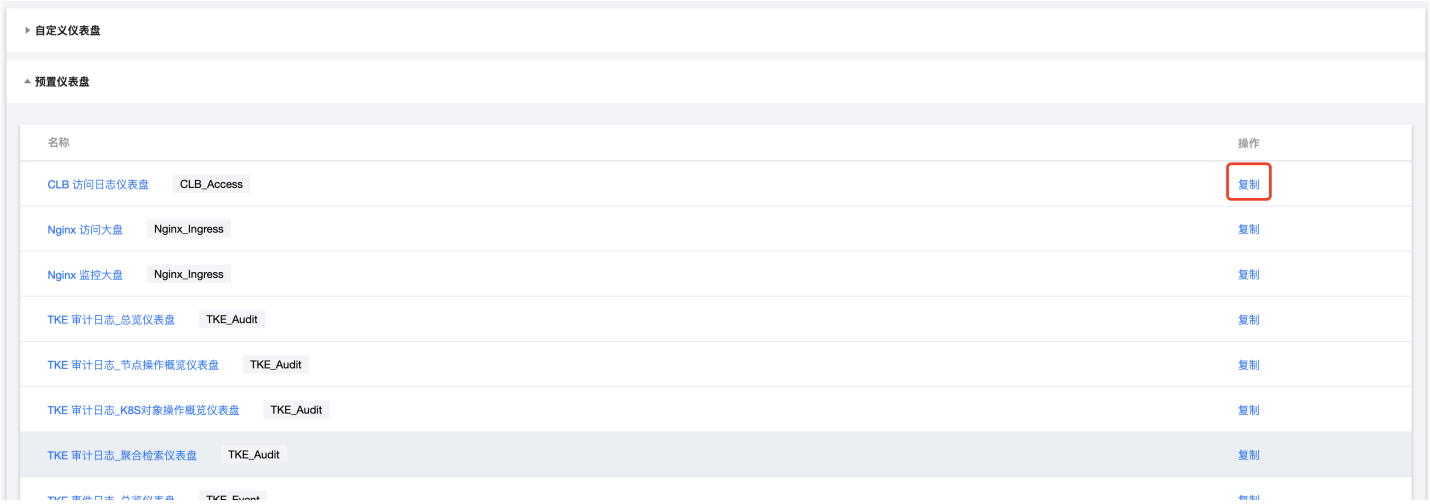
1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**仪表盘 > 仪表盘列表**，进入仪表盘列表页面。
3. 在仪表盘列表页面，展开**预置仪表盘**，选择并打开目标预置仪表盘。



4. 选择对应云产品的日志主题。

基于预置仪表盘做编辑

如果当前的预置仪表盘不满足需求，需要基于预置仪表盘添加更丰富的统计图表，可单击**复制**，复制预置仪表盘到自定义仪表盘列表，进行自定义编辑。

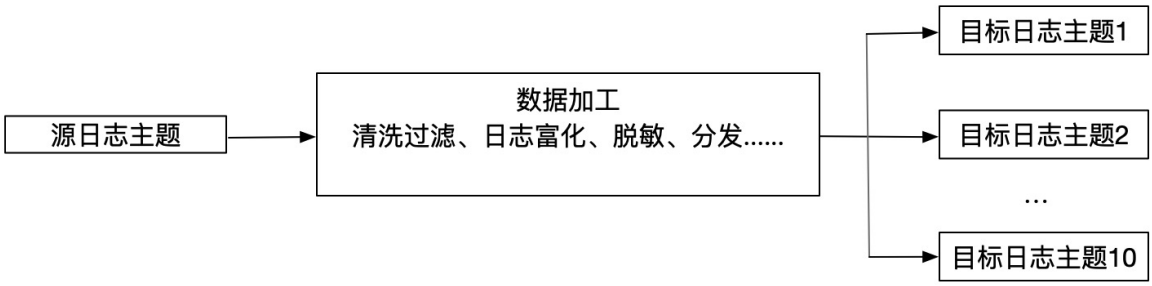


数据加工
创建加工任务

最近更新时间：2022-04-27 14:36:27

概述

数据加工提供对日志数据的过滤、清洗、脱敏、富化、分发等能力。



用户选择一个源日志主题，编写一段 DSL（Domain Specific Language）加工函数的语句，DSL 函数将针对一行一行的日志，进行加工，最后可以根据一定的条件，将满足条件的日志发送到指定的目标日志主题中。

适用场景

对于大部分用户而言，数据加工的目标可能是下面的几种：

- 提取结构化的数据，方便后续检索分析，生成仪表盘等。
- 日志减肥瘦身，节约后续使用成本。丢弃不需要的日志数据，节约存储成本、流量成本。
- 敏感数据脱敏，例如将用户的身份证、手机号码脱敏。
- 日志分类投递，例如按照日志级别：ERROR、WARNING、INFO 将日志分类，然后分发到不同的日志主题。

常见的 DSL 函数

根据不同的数据加工场景，介绍一些常见的 DSL 加工函数，这些函数可以在**新建数据加工任务 > 编辑加工语句**页面的**DSL语句生成器**中看到使用示例。用户可以直接复制示例，将示例中的参数修改为自己实际的参数，即可快速使用。

- 提取结构化数据：按照分隔符提取 ext_sep(),json 提取 ext_json,ext_json_jmes，正则提取 ext_regex()。
- 日志减肥瘦身：丢弃一行日志 log_drop，丢弃日志字段 fields_drop()。
- 敏感数据脱敏：regex_replace()。
- 日志分类：先使用条件判断语句 t_if_else(),t_switch()，然后分发日志 log_output()。
- 字段编辑：新增（重置）字段 fields_set()，重命名字段 fields_rename()。
- 判断日志中有某个字段或者某种数据：存在某个字段 has_field()，存在某个数据 regex_match()。
- 判断日志中数值的大小：大于 op_gt()，大于等于 op_ge()，等于 op_eq()，可能还会对其进行加减乘除，如 op_mul(),op_add()。
- 对日志中文字进行处理：大小写 str_uppercase()，文字替换 str_replace()。
- 对日志时间进行处理：转UTC时间戳，dt_to_timestamp()。

以上是一些常见的日志数据加工场景所用到的 DSL 函数。此外，我们还提供了其他 DSL 函数，均在 DSL 语句生成器中，方便用户查看。如有需要，将其复制到 DSL 函数编辑框中，修改参数，即可使用。

下面将为您详细介绍如何创建数据加工任务。

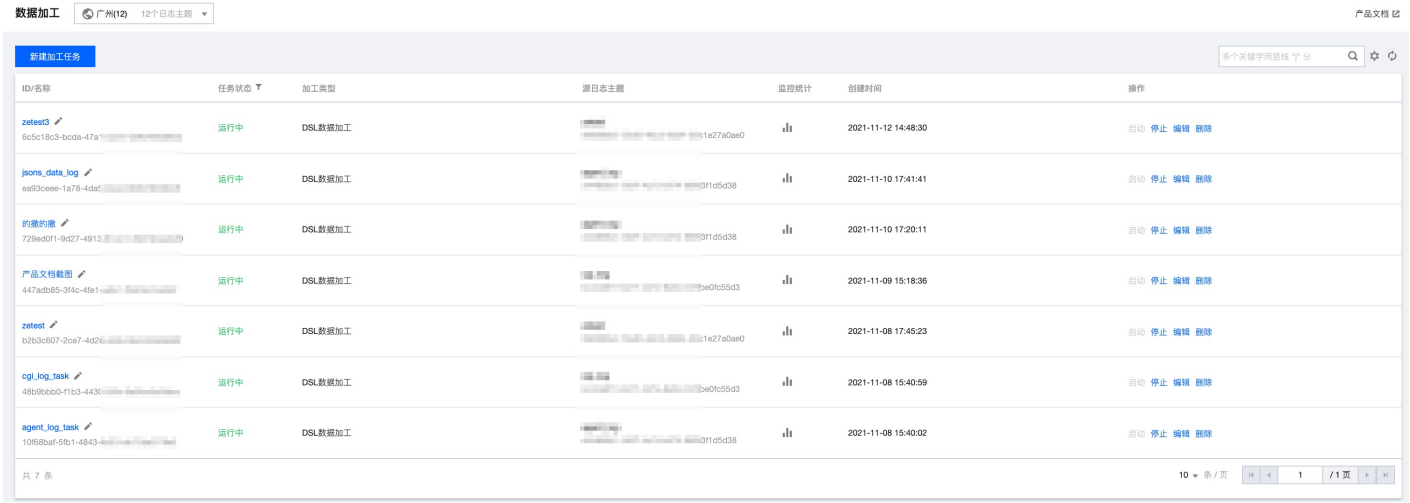
前提条件

- 已开通日志服务，创建源日志主题、并成功采集到日志数据。
- 已创建目标日志主题。建议目标日志主题为空主题，便于加工好的数据写入。
- 确保当前操作账号拥有配置数据加工任务的权限。

> **注意：**
>
> - 数据加工只能处理实时日志流，不能处理历史日志。
> - 如果数据加工的控制台没有自动加载原始日志数据，那么可能是您的日志主题没有实时日志流。需要您手动添加 JSON 格式的自定义日志，来完成数据加工脚本的撰写。

操作步骤

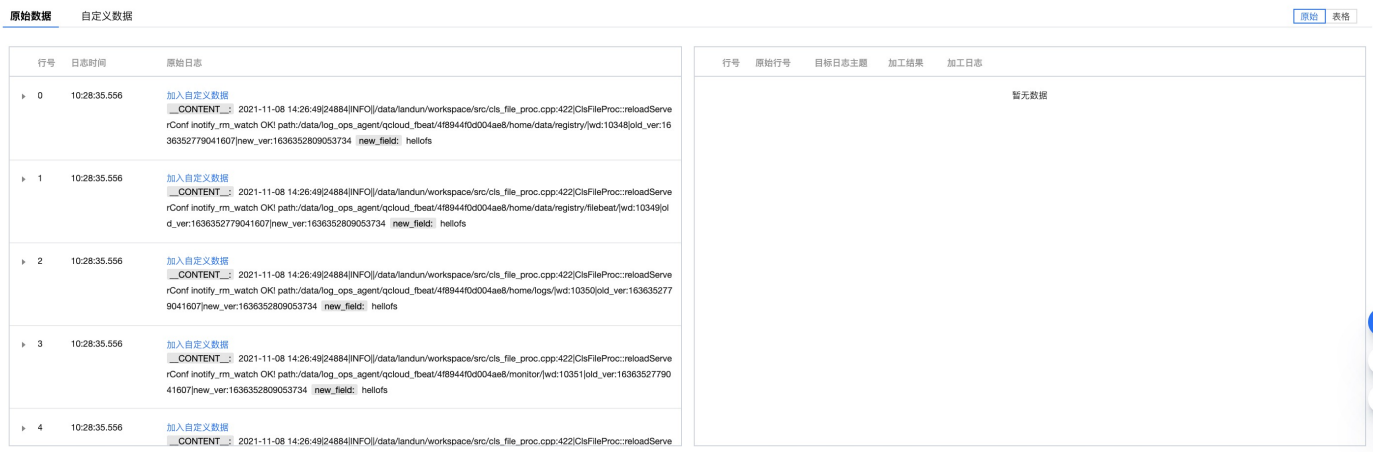
1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**数据加工**。
3. 单击**新建加工任务**。



4. 在**基本信息**页面，设置如下信息：

- **任务名称**：请输入自定义的任务名称。
- **源日志主题**：请选择源日志主题
- **目标日志主题**：请输入目标名称，选择日志主题。
  其中，目标名称可以理解为目标日志主题的别名，主要用作为编写 DSL 函数的入参。目标日志主题最多可以输入10个。

5. 单击**下一步**。
6. 在**编辑加工语句**页面，进行如下操作：
   i. 测试 DSL 函数的数据有两种，第一种是原始日志数据， 第二种是自定义数据。 系统会自动加载原始日志数据，默认100条，如果您觉得原始数据不能满足测试 DSL 函数的需求，可以直接在**自定义数据**页签，输入自定义数据（只支持JSON格式）。您也可以在**原始数据**页签，单击**加入自定义数据**，然后对其进行修改，作为您的自定义数据。



> **注意：**
> 自定义的数据必须为 JSON 格式。

自定义的数据如有多条，可参照如下格式添加：

```
[
{
"content": "2021-10-07 13: 32: 21|100|客人未入住|韦小宝|北京|101123199001230123|"
},
{
"content": "2021-10-07 13: 32: 21|500|入住成功|阿珂|三亚|501123199001230123|"
},
{
"content": "2021-10-07 13: 32: 21|1000|退房|韦春花|三亚|101123196001230123|"
}
]
```

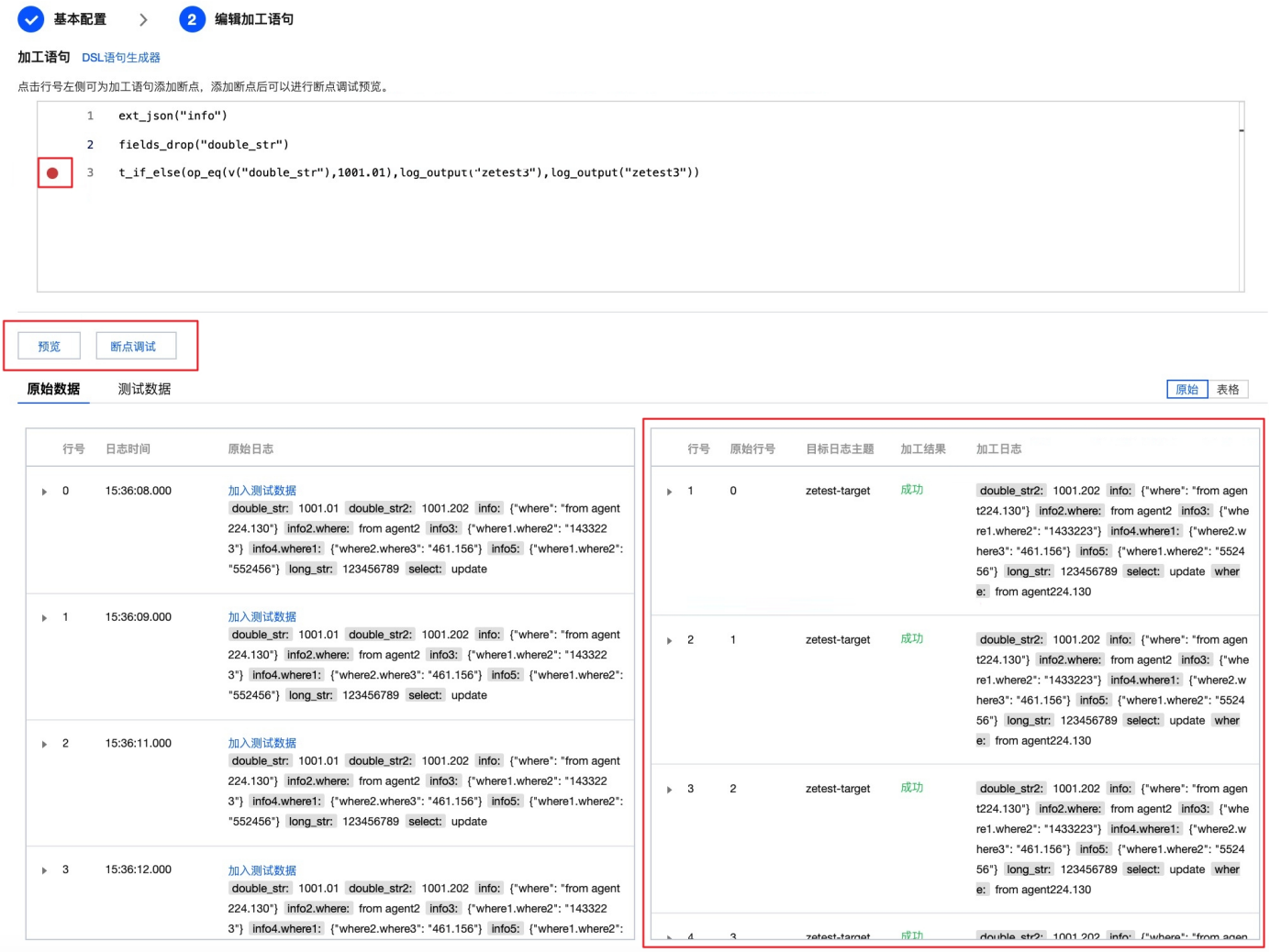
ii. 单击 DSL语句生成器。

iii. 在弹出的窗口中，选择函数，单击**插入函数**。

DSL 语句生成器提供了多种函数的说明和示例，您可以将示例复制/粘贴到加工语句的编辑框中，修改对应的参数，完成 DSL 函数的编写；也可以参考 加工示例 文

档，帮助您更快地理解 DSL 函数的编写。

iv. 完成 DSL 加工语句的编写后，单击**预览**或者**断点调试**，运行和调试 DSL 函数。
运行结果会在右下方展示，您可以针对运行结果，调整 DSL 语句，直到满足您的需求。



7. 单击确定，提交数据加工任务。

查看加工详情

最近更新时间：2021-11-18 11:07:23

操作场景

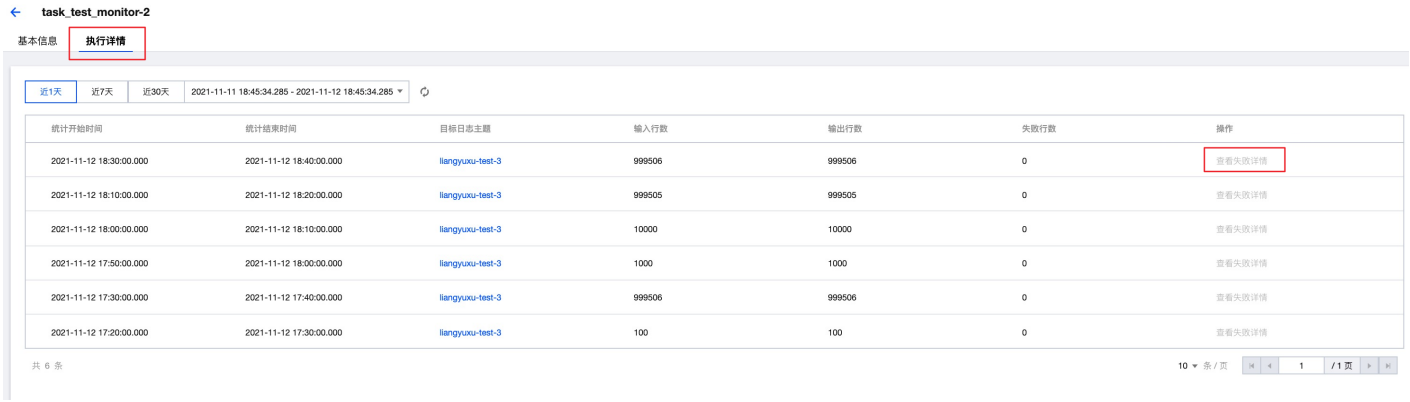
数据加工详情展示该任务处理了多少行日志、处理结果、失败原因。下面将为您介绍如何进入数据加工详情页面，查看加工任务的处理内容。

操作步骤

查看任务执行详情

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**数据加工**，进入数据加工管理页面。
3. 单击需要查看详情的加工任务 ID/名称，进入该加工任务的详情页面。

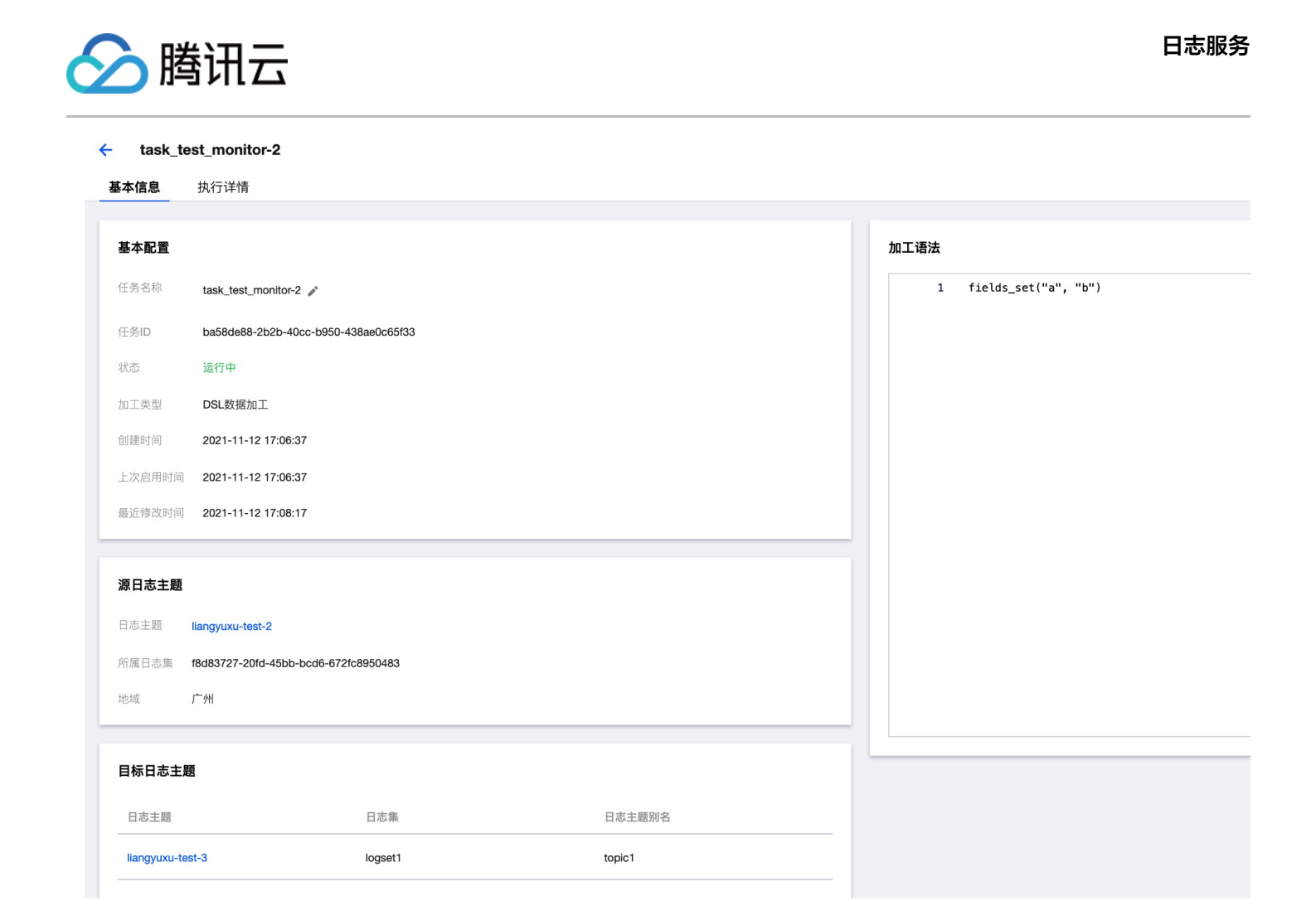
4. 选择**执行详情**页签，可以按照时间段查看数据加工任务的执行详情。
   执行详情为10分钟一个统计周期，统计了数据加工的日志数据输入行数、输出行数、失败行数。如果执行详情有失败的情况，可以单击**查看失败详情**进行查看。



查看任务基本信息

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**数据加工**，进入数据加工管理页面。
3. 单击需要查看详情的加工任务 ID/名称，进入该加工任务的详情页面。

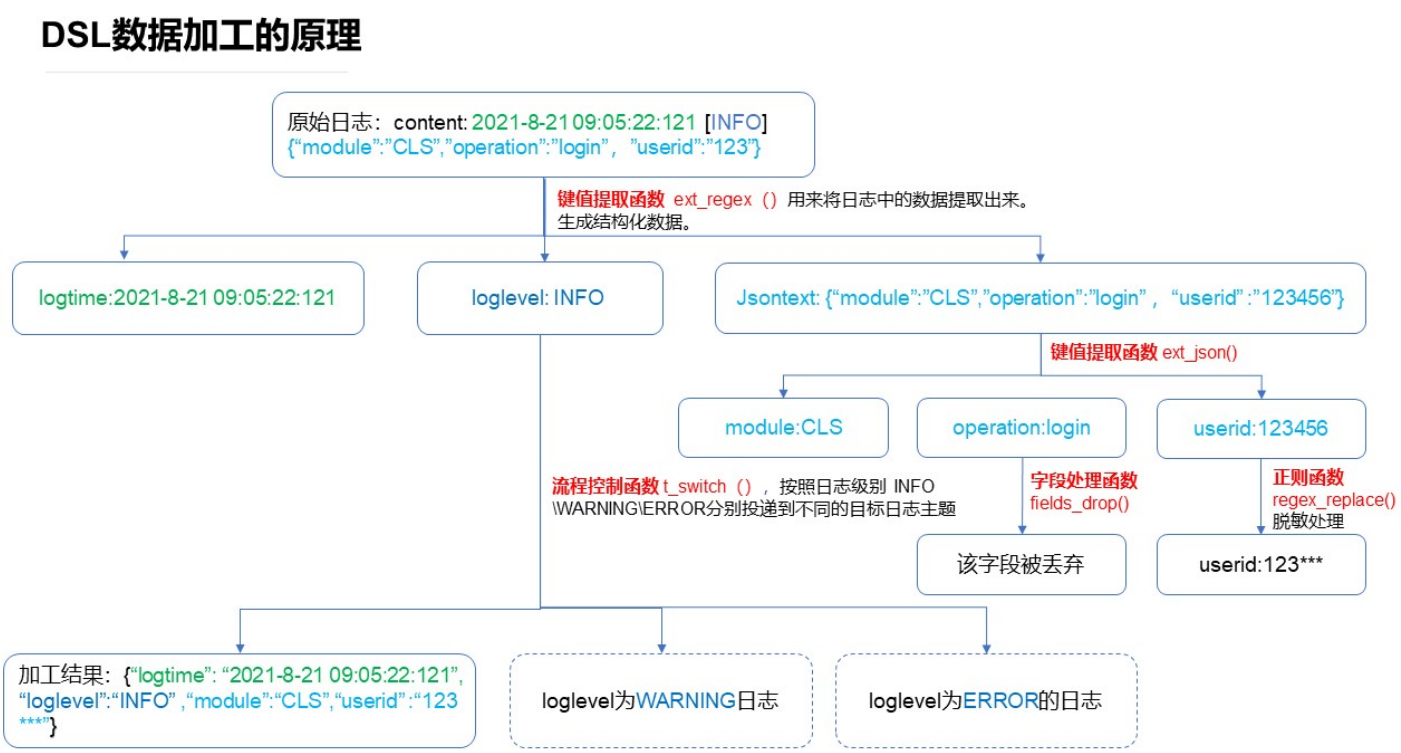
4. 选择**基本信息**页签，可以查看到该加工任务的基本配置、源日志主题、目标日志主题以及加工语句。
   此外，您还可以通过单击目标日志主题的日志主题，进入索引分析页面。（该操作需要先打开索引配置，才能查看目标日志主题的数据）


task_test_monitor-2
基本信息
执行详情
基本配置
任务名称 task_test_monitor-2
任务ID ba58de88-2b2b-40cc-b950-438ae0c65f33
状态 运行中
加工类型 DSL数据加工
创建时间 2021-11-12 17:06:37
上次启用时间 2021-11-12 17:06:37
最近修改时间 2021-11-12 17:08:17
源日志主题
日志主题 liangyuxu-test-2
所属日志集 f8d83727-20fd-45bb-bcd6-672fc8950483
地域 广州
目标日志主题
日志主题 日志集 日志主题别名
liangyuxu-test-3 logset1 topic1
加工语法
1 fields_set("a", "b")

数据加工函数
函数总览

最近更新时间：2022-07-04 09:16:37

日志服务（Cloud Log Service，CLS）数据加工函数可以自由组合，来完成日志的清洗、结构化、过滤、分发、脱敏等场景。
如下图是一条含有 JSON 的日志，经过数据加工处理之后，变成结构化数据、然后对某个字段值进行脱敏、最后分发的流程。



如下是数据加工函数的总览。

- **键值提取函数类**：从日志文本中提取字段/字段值

函数名称	函数功能	函数语法描述	返回值类型
ext_sep	基于分隔符提取字段值内容	ext_sep("源字段名", "目标字段1,目标字段2,目标字段...", sep="分隔符", quote="不参与分割的部分"", restrict=False, mode="overwrite")	返回提取后的日志(LOG)
ext_sepstr	基于指定字符（串）提取字段值内容	ext_sepstr("源字段名","目标字段1,目标字段2,目标字段...", sep="abc", restrict=False, mode="overwrite")	返回提取后的日志(LOG)
ext_json	提取 JSON 字符串格式的字段值	返回提取后的日志(LOG) ext_json("源字段名",prefix="",suffix="",format="full",exclude_node="不平铺的JSON节点")	返回提取后的日志(LOG)
ext_json_jmes	使用 jmes 表达式提取字段值	ext_json_jmes(“源字段名”, jmes= "提取JSON的公示", output="目标字段", ignore_null=True, mode="overwrite")	返回提取后的日志(LOG)
ext_kv	基于两级分割符提取字段值	ext_kv("源字段名", pair_sep=r"\s", kv_sep="=", prefix="", suffix="", mode="fill-auto")	返回提取后的日志(LOG)
ext_regex	基于正则表达式提取字段值	ext_regex(“源字段名”, regex="正则表达式", output=“目标字段1，目标字段2，目标字段.....”, mode="overwrite")	返回提取后的日志(LOG)
ext_first_notnull	返回参数中第一个非 null 且非空字符的结果值	ext_first_notnull(值1, 值2, ...)	返回参数中第一个非 null 结果值

- **富化函数类**：在已有字段的基础上，根据规则，新增字段

函数名称	函数功能	函数语法描述	返回值类型
enrich_table	使用 csv 结构数据对日志中的字段进行匹配。当值相同时，可以将 csv 中的其他字段和值，添加到源日志中	enrich_table(“csv源数据”，“csv富化字段”，output=“目标字段1，目标字段2，目标字段....”，mode="overwrite")	返回映射后的日志(LOG)
enrich_dict	使用 dict 结构对日志中的字段值进行匹配。当指定的字段的值和 dict 中的 key 相同时，将此 key 对应的 value 赋值给日志中的另一字段	enrich_dict(“JSON字典”, "源字段名", output=目标字段, mode="overwrite")	返回映射后的日志(LOG)

- **流程控制函数类**：条件判断

函数名称	函数功能	函数语法描述	返回值类型
compose	组合操作函数，类似于分支代码块的组合能力，可以组合多个操作函数，并按顺序执行，可以结合分支、输出函数使用	compose("函数1","函数2", ...)	返回日志(LOG)
t_if	对符合条件的日志，进行相应的函数处理，否则不进行任何处理	t_if("条件", 函数)	返回日志(LOG)
t_if_not	对不符合条件的日志，进行相应的函数处理，否则不进行任何处理	t_if_not("条件",函数)	返回日志(LOG)
t_if_else	基于条件判断，分别进行不同的函数处理	t_if_else("条件", 函数1, 函数2)	返回日志(LOG)
t_switch	基于多分支条件，分别进行不同的函数处理，如果存在不符合所有条件的数据，将被丢弃	t_switch("条件1", 函数1, "条件2", 函数2, ...)	返回日志(LOG)

- **行处理函数类**：日志的分发、丢弃、拆分

函数名称	函数功能	函数语法描述	返回值类型
log_output	输出到指定的目标主题。可以配合分支条件使用，也可以单独使用	log_output(日志主题别名)，该参数在新建数据加工任务时，目标日志主题别名处配置	无返回，对目前的数据流进行输出
log_split	使用分隔符结合 jmes 表达式，对特定字段进行拆分，拆分结果分裂为多行日志	log_split(字段名, sep=",", quote="\"", jmes="", output="")	返回日志(LOG)
log_drop	丢弃符合条件的日志	log_drop(条件1)	返回日志(LOG)
log_keep	保留符合条件的日志	log_keep(条件1)	返回日志(LOG)
log_split_jsonarray_jmes	将日志根据 jmes 语法将 JSON 数组拆分和展开	log_split_jsonarray_jmes("field", jmes="items", prefix="")	返回日志(LOG)

- **字段处理函数**：字段的增删改查、重命名

函数名称	函数功能	函数语法描述	返回值类型
fields_drop	根据字段名进行匹配，丢弃匹配到的字段	fields_drop(字段名1, 字段名2, ..., regex=False,nest=False)	返回日志(LOG)
fields_keep	根据字段名进行匹配，保留匹配到的字段	fields_keep(字段名1, 字段名2, ..., regex=False)	返回日志(LOG)
fields_pack	根据正则表达式来匹配字段名，并将匹配到的字段打包到新的字段，新字段值使用 JSON 格式进行组织	fields_pack(目标字段名, include=".*", exclude="", drop_packed=False)	返回日志(LOG)
fields_set	用来设置字段值，或者增加新字段	fields_set(字段名1, 字段值1, 字段名2, 字段值2, mode="overwrite")	返回日志(LOG)
fields_rename	字段重命名	fields_rename(字段名1, 新字段名1, 字段名2, 新字段名2, regex=False)	返回日志(LOG)
has_field	字段存在时，返回 True，否则返回 False	has_field(字段名)	返回条件值(BOOL)
not_has_field	字段不存在时，返回 True，否则返回	not_has_field(字段名)	返回条件值(BOOL)

	False		
v	获取字段值，返回对应字符串	v(字段名)	返回值字符串类型(STRING)

- **值结构化处理函数类**：JSON 数据的处理

函数名称	函数功能	函数语法描述	返回值类型
json_select	通过jmes表达式，提取 JSON 字段值，并返回 jmes 提取结果的 JSON 字符串	json_select(v(字段名), jmes="")	返回值字符串类型(STRING)
xml_to_json	解析 xml 值并转换为 JSON 字符串，输入值必须为 xml 字符串结构，否则会导致转换异常	xml_to_json(字段值)	返回值字符串类型(STRING)
json_to_xml	解析 JSON 字符串值并转换为 xml 字符串	json_to_xml(字段值)	返回值字符串类型(STRING)
if_json	判断是否为 JSON 字符串	if_json(字段值)	返回条件值(BOOL)

- **正则处理函数类**：使用正则公式对文本中的字符进行匹配、替换

函数名称	函数功能	函数语法描述	返回值类型
regex_match	基于正则对数据进行匹配，返回是否匹配成功，可以选择全匹配还是部分匹配	regex_match(字段值, regex="", full=True)	返回条件值(BOOL)
regex_select	基于正则对数据进行匹配，返回相应的部分匹配结果，可以指定匹配结果的第几个表达式，以及第几个分组（部分匹配+指定捕获组序号），如果最终没有匹配结果，则返回空字符串	regex_select(字段值, regex="", index=1, group=1)	返回值字符串类型(STRING)
regex_split	基于正则对数据进行分割，返回 JSON Array 字符串（部分匹配）	regex_split(字段值, regex=\"\", limit=100)	返回值字符串类型(STRING)
regex_replace	基于正则匹配并替换（部分匹配）	regex_replace(字段值, regex="", replace="", count=0)	返回值字符串类型(STRING)
regex_findall	基于正则进行匹配，并将匹配结果添加到 JSON 数组中，并返回 Array 字符串（部分匹配）	regex_findall(字段值, regex="")	返回值字符串类型(STRING)

- **日期值处理类**

函数名称	函数功能	函数语法描述	返回值类型
dt_str	将时间类的字段值（特定格式的日期字符串或者时间戳），转换为指定时区、格式的目标日期字符串	dt_str(值, format="格式化字符串", zone="")	返回值字符串类型(STRING)
dt_to_timestamp	将时间类的字段值（特定格式的日期字符串），同时指定字段对应的时区，转换为 UTC 时间戳	dt_to_timestamp(值, zone="")	返回值字符串类型(STRING)
dt_from_timestamp	将时间类的时间戳字段，指定目标时区后，转换为时间字符串	dt_from_timestamp(值, zone="")	返回值字符串类型(STRING)
dt_now	获取加工计算时的本地时间	dt_now(format="格式化字符串", zone="")	返回值字符串类型(STRING)

- **字符串处理类**

函数名称	函数功能	函数语法描述	返回值类型
str_count	在值中指定范围内查找子串，返回子串出现的次数	str_count(值, sub="", start=0, end=-1)	返回子串次数(INT)
str_len	返回字符串长度	str_len(值)	返回字符串长度(INT)

str_uppercase	返回大写字符串	str_uppercase(值)	返回值字符串类型(STRING)
str_lowercase	返回小写字符串	str_lowercase(值)	返回值字符串类型(STRING)
str_join	使用拼接字符串，拼接多值	str_join(拼接字符串1, 值1, 值2, ...)	返回值字符串类型(STRING)
str_replace	替换字符串，返回替换结果字符串	str_replace(值, old="", new="", count=0)	返回值字符串类型(STRING)
str_format	格式化字符串，返回格式化结果	str_format(格式化字符串, 值1, 值2, ...)	返回值字符串类型(STRING)
str_strip	剔除用户指定的字符序列中的字符，从字符串开头和结尾同时剔除，返回剔除后的结果	str_strip(值, chars="\t\r\n")	返回值字符串类型(STRING)
str_lstrip	剔除用户指定的字符序列中的字符，从字符串左侧开头剔除，返回剔除后的结果	str_strip(值, chars="\t\r\n")	返回值字符串类型(STRING)
str_rstrip	剔除用户指定的字符序列中的字符，从字符串右侧结尾部分剔除，返回剔除后的结果	str_strip(值, chars="\t\r\n")	返回值字符串类型(STRING)
str_find	在值中查找子串，并返回子串出现的位置	str_find(值, sub="", start=0, end=-1)	返回指定第一次出现在值中的子字符串的位置(INT)
str_start_with	判断字符串是否以指定字符串开头	str_start_with(值, sub="", start=0, end=-1)	返回是否匹配的结果(BOOL)
str_end_with	判断字符串是否以指定字符串结尾	str_end_with(值, sub="", start=0, end=-1)	返回是否匹配的结果(BOOL)

- **逻辑表达式**

函数名称	函数功能	函数语法描述	返回值类型
op_if	根据条件判断，返回相应的值	op_if(条件1, 值1, 值2)	条件为 true 时，返回值1，否则返回值2
op_and	对值进行 and 运算，均为 True 时，返回 True，否则返回 False	op_and(值1, 值2, ...)	返回计算的结果(BOOL)
op_or	对值进行 or 运算，若存在参数值为 False 则返回 False，否则返回 True	op_or(值1, 值2, ...)	返回计算的结果(BOOL)
op_not	对值进行 not 运算	op_not(值)	返回计算的结果(BOOL)
op_eq	对值进行比较，相等则返回 True	op_eq(值1, 值2)	返回比较的结果(BOOL)
op_ge	对值进行比较，值1大于或等于值2时返回 True	op_ge(值1, 值2)	返回比较的结果(BOOL)
op_gt	对值进行比较，值1大于值2时返回 True	op_gt(值1, 值2)	返回比较的结果(BOOL)
op_le	对值进行比较，值1小于或等于值2时返回 True	op_le(值1, 值2)	返回比较的结果(BOOL)
op_lt	对值进行比较，值1小于值2时返回 True	op_lt(值1, 值2)	返回比较的结果(BOOL)
op_add	对值进行求和运算	op_add(值1, 值2)	返回求值结果
op_sub	对值进行求差运算	op_sub(值1, 值2)	返回求值结果
op_mul	对值进行乘积运算	op_mul(值1, 值2)	返回求值结果
op_div	对值进行除法运算	op_div(值1, 值2)	返回求值结果
op_sum	对多值累加求和	op_sum(值1, 值2, ...)	返回求值结果
op_mod	对值进行模计算	op_mod(值1, 值2)	返回求值结果
op_null	对值进行是否为 null 判断，是则返回 true，	op_null(值)	返回计算的结果(BOOL)

	否则返回 false		
op_notnull	对值进行是否为非 null 判断，是则返回 true，否则返回 false	op_notnull(值)	返回计算的结果(BOOL)
op_str_eq	对字符串值进行比较，相等则返回 true	op_str_eq(值1, 值2, ignore_upper=False)	返回计算的结果(BOOL)

- **类型转换函数类**

函数名称	函数功能	函数语法描述	返回值类型
ct_int	对值进行整型转换，可指定原值的进制，转为十进制数值	ct_int(值1, base=10)	返回求值结果
ct_float	将值转换为浮点型数值	ct_float(值)	返回求值结果
ct_str	将值转换为字符串	ct_str(值)	返回求值结果
ct_bool	将值转换为布尔值	ct_bool(值)	返回求值结果

- **编解码函数类**

函数名称	函数功能	函数语法描述	返回值类型
decode_url	将编码 URL 进行解码	decode_url(值)	返回值字符串类型(STRING)

- **IP 解析函数类**

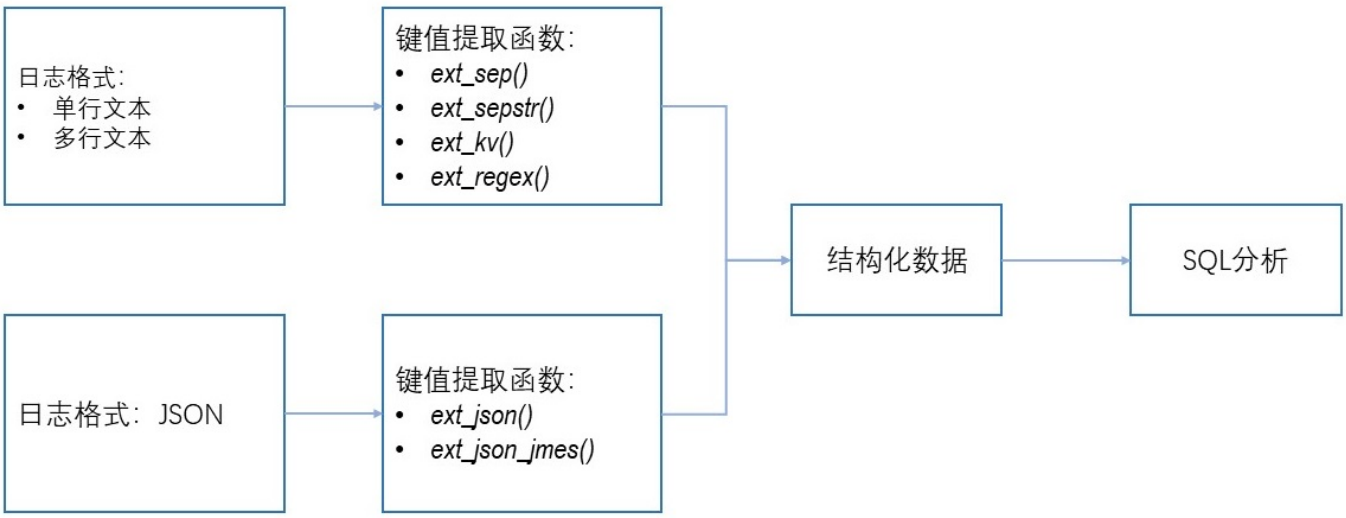
函数名称	函数功能	函数语法描述	返回值类型
geo_parse	解析出函数的地理位置	geo_parse(字段值, keep=("country","province","city"), ip_sep=",")	返回 JSON 字符串

键值提取函数

最近更新时间：2022-07-04 09:16:54

简介

键值提取函数常见的使用场景如下图，处理为结构化数据之后，可以进一步用于 SQL 分析的场景。



ext_sep() 函数

函数定义

基于分隔符(单字符)提取字段值内容

语法描述

```
ext_sep("源字段名", "目标字段1,目标字段2,目标字段...", sep="分隔符", quote="不参与分割的部分"", restrict=False, mode="overwrite")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
field	待提取的字段名	string	是	-	用户日志中已存在的字段名
output	单个字段名或以英文半角逗号拼接的多个新字段名	string	是	-	-
sep	分隔符	string	否	,	任意单字符
quote	将值包括起来的字符	string	否	-	-
restrict	默认为 False。当提取的值个数与用户输入的目标字段数不一致时： True：忽略，不进行任何提取处理 False：尽量匹配前几个字段	bool	否	False	-
mode	新字段的写入模式。默认强制覆盖	string	否	overwrite	-

示例

- 示例1：使用逗号作为分隔符，提取日志中的值。

  原始日志：

```
{"content": "hello Go,hello Java,hello python"}
```

加工规则：

```
//使用逗号作为分割符号，将content字段值分割为三段，分别对应f1,f2,f3三个字段。
ext_sep("content", "f1, f2, f3", sep=",", quote="", restrict=False, mode="overwrite")
//丢弃content字段
fields_drop("content")
```

加工结果：

```
{"f1":"hello Go","f2":"hello Java","f3":"hello python"}
```

- 示例2：使用 quote 将字符串整体处理。

原始日志：

```
{"content": " Go,%hello ,Java%,python"}
```

加工规则：

```
ext_sep("content", "f1, f2", quote="%", restrict=False)
```

加工结果：

```
//%hello ,Java%虽然也有逗号，但它作为一个整体，就不参与分隔符提取。
{"content":" Go,%hello ,Java%,python","f1":" Go","f2":"hello ,Java"}
```

- 示例3：当 restrict=True, 表示当分割的值的个数和目标字段不同时，函数不执行。

原始日志：

```
{"content": "1,2,3"}
```

加工规则：

```
ext_sep("content", "f1, f2", restrict=True)
```

加工结果：

```
{"content":"1,2,3"}
```

ext_sepstr() 函数

函数定义

基于指定的字符（多字符）提取字段值内容

语法描述

```
ext_sepstr("源字段名","目标字段1,目标字段2,目标字段...", sep="abc", restrict=False, mode="overwrite")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
field	待提取的字段名	string	是	-	用户日志中已存在的字段名
output	单个字段名或以英文半角逗号拼接的多个新字段名	string	是	-	-
sep	分隔字符（串）	string	否	,	-

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
restrict	默认为False。当提取的值个数与用户输入的目标字段数不一致时： True：忽略，不进行任何提取处理。 False：尽量匹配前几个字段	bool	否	False	-
mode	新字段的写入模式。默认强制覆盖	string	否	overwrite	-

示例

原始日志：

```
{"message":"1##2##3"}
```

加工规则：

```
//使用“##”作为分割符，提取键值。
ext_sepstr("message", "f1,f2,f3,f4", sep="##")
```

加工结果：

```
//当目标字段的个数超出了分隔出的值，超出的字段返回"".
{"f1":"1","f2":"2","message":"1##2##3","f3":"3","f4":""}
```

ext_json() 函数

函数定义

从 JSON 中提取字段值

语法描述

```
ext_json("源字段名",prefix="",suffix="",format="full",exclude_node="不平铺的JSON节点")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
field	待提取的字段名	string	是	-	-
prefix	新字段前缀	string	否	-	-
suffix	新字段后缀	string	否	-	-
format	full：字段名格式为全路径（parent + sep + prefix + key + suffix） simple：非全路径（prefix + key + suffix）	string	否	simple	-
sep	拼接符，用来拼接节点名使用	string	否	#	-
depth	最大展开层级深度，超过此层级的节点不再展开	number	否	100	1-500
expand_array	数组节点是否展开	bool	否	False	-
include_node	基于正则匹配节点名称的白名单	string	否	-	-
exclude_node	基于正则匹配节点名称的黑名单	string	否	-	-
include_path	基于正则匹配节点路径的白名单	string	否	-	-
exclude_path	基于正则匹配节点路径的黑名单	string	否	-	-

示例

- 示例1：提取全部节点的 KV 成为新字段，提取时不分层级，该例子是多层嵌套。
  原始日志：

```
{
"data": "{ \"k1\": 100, \"k2\": { \"k3\": 200, \"k4\": { \"k5\": 300}}}"
}
```

加工规则：

```
ext_json("data")
```

加工结果：

```
{"data":"{ \"k1\": 100, \"k2\": { \"k3\": 200, \"k4\": { \"k5\": 300}}}","k1":"100","k3":"200","k5":"300"}
```

- **示例2：sub_field1 不参与提取。**
  原始日志：

```
{"content": "{\"sub_field1\":1,\"sub_field2\":\"2\"}"}
```

加工规则：

```
//exclude_node=subfield1意思就是这个节点不提取
ext_json("content", format="full", exclude_node="sub_field1")
```

加工结果：

```
{"sub_field2":"2","content":"{\"sub_field1\":1,\"sub_field2\":\"2\"}"}
```

- **示例3：给子节点加上前缀 prefix。**
  原始日志：

```
{"content": "{\"sub_field1\":{\"sub_sub_field3\":1},\"sub_field2\":\"2\"}"}
```

加工规则1：

```
//sub_field2被提取出来时，自动加上前缀udf\_，成为udf\_\_sub\_field2
ext_json("content", prefix="udf_", format="simple")
```

加工结果1：

```
{"content":"{\"sub_field1\":{\"sub_sub_field3\":1},\"sub_field2\":\"2\"}","udf_sub_field2":"2","udf_sub_sub_field3":"1"}
```

加工规则2：

```
//format=full表示提取的字段名自带层级关系，sub_field2被提取出来时，自动加上它的父节点名称，成为#content#__sub_field2
ext_json("content", prefix="__", format="full")
```

加工结果2：

```
{"#content#__sub_field2":"2","#content#sub_field1#__sub_sub_field3":"1","content":"{\"sub_field1\":{\"sub_sub_field3\":1},\"sub_field2\":\"2\"
}"}
```

ext_json_jmes() 函数

函数定义

从 JSON 中提取字段值

语法描述

```
ext_json_jmes(“源字段名”, jmes= "提取JSON的公示", output="目标字段", ignore_null=True, mode="overwrite")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
field	待提取的字段名	string	是	-	-
jmes	jmes 表达式，请参考 JMESPath	string	是	-	-
output	输出字段名，仅支持单个字段	string	是	-	-
ignore_null	是否忽略节点值为 null 的节点，默认 True，忽略 null 字段值，否则当提取结果为 null 时，将输出空字符串	bool	否	True	-
mode	新字段的写入模式。默认强制覆盖	string	否	overwrite	-

示例

- 示例1：只提取多层 JSON 中的一个节点。

  原始日志：

  ```
  {"content": "{\"a\":{\"b\":{\"c\":{\"d\":\"value\"}}}}"}
  ```

  加工规则：

  ```
  //jmes="a.b.c.d"的意思是取a.b.c.d的值。
  ext_json_jmes("content", jmes="a.b.c.d", output="target")
  ```

  加工结果：

  ```
  {"content":"{\"a\":{\"b\":{\"c\":{\"d\":\"value\"}}}}","target":"value"}
  ```

- 示例2:

  原始日志：

  ```
  {"content": "{\"a\":{\"b\":{\"c\":{\"d\":\"value\"}}}}"}
  ```

  加工规则：

  ```
  //jmes="a.b.c.d"的意思是取a.b.c的值。
  ext_json_jmes("content", jmes="a.b.c", output="target")
  ```

  加工结果：

  ```
  {"content":"{\"a\":{\"b\":{\"c\":{\"d\":\"value\"}}}}","target":"{\"d\":\"value\"}"}
  ```

ext_regex() 函数

函数定义

基于正则表达式提取字段值

语法描述

```
ext_regex(“源字段名”, regex="正则表达式", output=“目标字段1，目标字段2，目标字段.....”, mode="overwrite")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
field	待提取的字段名	string	是	-	-
regex	regex 表达式，如果包含了特殊字符，用户输入时需要进行转义，否则提示语法错误	string	是	-	-
output	单个字段名或以英文半角逗号拼接的多个新字段名	string	否	-	-
mode	新字段的写入模式。默认强制覆盖	string	否	overwrite	-

示例

- **示例1：匹配数字**

  原始日志：

  ```
  {"content": "1234abcd5678"}
  ```

  加工规则：

  ```
  ext_regex("content", regex="\d+", output="target1,target2")
  ```

  加工结果：

  ```
  {"target2":"5678","content":"1234abcd5678","target1":"1234"}
  ```

- **示例2：有命名捕获，自动填充部分字段值。**

  原始日志：

  ```
  {"content": "1234abcd"}
  ```

  加工规则：

  ```
  ext_regex("content", regex="(?<target1>\d+)(.*)", output="target2")
  ```

  加工结果：

  ```
  {"target2":"abcd","content":"1234abcd","target1":"1234"}
  ```

ext_kv() 函数

函数定义

基于两级分割符提取字段值

语法描述

```
ext_kv("源字段名", pair_sep=r"\s", kv_sep="=", prefix="", suffix="", mode="fill-auto")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
field	待提取的字段名	string	是	-	-
pair_sep	一级分隔符，分割多个键值对	string	是	-	-
kv_sep	二级分隔符，分割键和值	string	是	-	-
prefix	新字段前缀	string	否	-	-

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
suffix	新字段后缀	string	否	-	-
mode	新字段的写入模式。默认强制覆盖	string	否	-	-

示例

日志中有两级分割符“|”和“=”。

原始日志：

```
{"content": "a=1|b=2|c=3"}
```

加工规则：

```
ext_kv("content", pair_sep="|", kv_sep="=")
```

加工结果：

```
{"a":"1","b":"2","c":"3","content":"a=1|b=2|c=3"}
```

ext_first_notnull() 函数

函数定义

返回参数中第一个非 null 且非空字符的结果值

语法描述

```
ext_first_notnull(值1, 值2, ...)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
可变参列表	参与计算的参数或表达式	string	是	-	-

示例

原始日志：

```
{"data1": null, "data2": "", "data3": "first not null"}
```

加工规则：

```
fields_set("result", ext_first_notnull(v("data1"), v("data2"), v("data3")))
```

加工结果：

```
{"result":"first not null","data3":"first not null","data2":"","data1":"null"}
```

富化函数

最近更新时间：2022-03-04 10:26:29

enrich_table 函数

函数定义

使用 CSV 结构数据对日志中的字段进行匹配，当值相同时，可以将 CSV 中的其他字段和值添加到源日志中。

语法描述

```
enrich_table(csv数据字符串, csv列名字符串, output=目标字段名或列表, mode="overwrite")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	输入的 CSV 数据，第一行为列名，其余行对应值。例如：region,count\nbj, 200\ngz, 300	string	是	-	-
fields	待匹配列名称。CSV 中的字段名称，与实际日志中同名的字段进行匹配。单个字段名或以英文半角逗号拼接的多个新字段名	string	是	-	-
output	输出字段列表，单个字段名或以英文半角逗号拼接的多个新字段名	string	是	-	-
mode	新字段的写入模式。默认强制覆盖	string	否	overwrite	-

示例

原始日志：

```
{"region": "gz"}
```

加工规则：

```
enrich_table("region,count\nbj,200\ngz,300", "region", output="count")
```

加工结果：

```
{"count":"300","region":"gz"}
```

enrich_dict 函数

函数定义

使用 dict 结构对日志中的字段值进行匹配，当指定的字段的值和 dict 中的 key 相同时，将此 key 对应的 value 赋值给日志中的另一字段。

语法描述

```
enrich_dict(dict数据字符串, 源字段名, output=目标字段, mode="overwrite")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	输入的 dict 数据，这里必须是 JSON 对象的转义字符串，例如：enrich_dict("{"200":"SUCCESS","500":"FAILED"}", "status", output="message")	string	是	-	-
fields	待匹配字段名称。当 dict 中的 key 和指定字段对应的值相同时，匹配成功。单个字段名或以英文半角逗号拼接的多个新字段名	string	是	-	-

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
output	目标字段列表。匹配成功后，将 dict 中对应的 value 写入到目标字段列表。单个字段名或以英文半角逗号拼接的多个新字段名	string	是	-	-
mode	新字段的写入模式。默认强制覆盖	string	否	overwrite	-

示例

原始日志：

```
{"status": "500"}
```

加工规则：

```
enrich_dict("{\"200\":\"SUCCESS\",\"500\":\"FAILED\"}", "status", output="message")
```

加工结果：

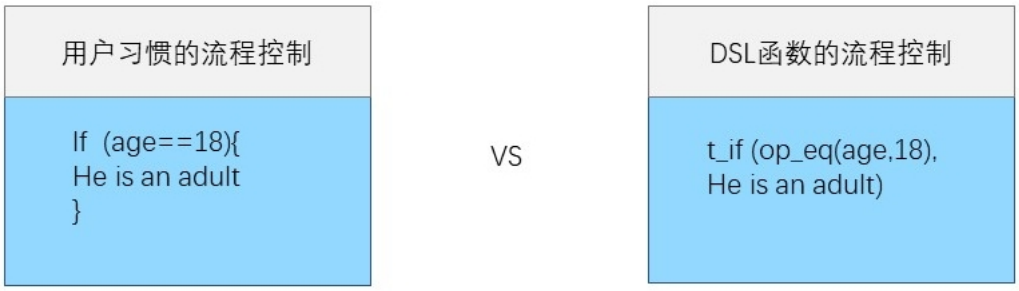
```
{"message":"FAILED","status":"500"}
```

流程控制

最近更新时间：2022-03-29 16:05:12

简介

流程控制的逻辑在常用的编程语言和 DSL 函数中是一样的，区别在于写法不同。如下图所示：



compose 函数

函数定义

组合操作函数，类似于分支代码块的组合能力，可以组合多个操作函数，并按顺序执行，可以结合分支、输出函数使用。

语法描述

```
compose(函数1,函数2, ...)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
可变参，函数	参数必须为返回类型为 LOG 的函数	string	至少一个函数参数	-	-

示例

- 示例1：顺序调用函数，先执行 enrich 函数，然后是 fields_set 函数。

  原始日志：

  ```
  {"status": "500"}
  ```

  加工规则：

  ```
  //1. enrich函数：使用dict中的数据，对原始日志进行富化，status字段值为500，富化出新字段message，值为Failed。
  //2. fields_Set函数，新增一个字段new，赋值1。
  compose(enrich_dict("{\"200\":\"SUCCESS\",\"500\":\"FAILED\"}", "status", output="message"), fields_set("new", 1))
  ```

  加工结果：

  ```
  //最后结果是三个字段，如下：
  {"new":"1","message":"FAILED","status":"500"}
  ```

- 示例2

  原始日志：

  ```
  {"status": "500"}
  ```

加工规则：

```
compose(fields_set("new", 1))
```

加工结果：

```
{"new":"1","status":"500"}
```

- 示例3

原始日志：

```
{"condition1": 0,"condition2": 1, "status": "500"}
```

加工规则：

```
t_if_else(v("condition2"), compose(fields_set("new", 1),log_output("target")), log_output("target2"))
```

加工结果，target 输出：

```
{"new":"1","condition1":"0","condition2":"1","status":"500"}
```

t_if 函数

函数定义

对符合条件的日志，进行相应的函数处理，否则不进行任何处理。

语法描述

```
t_if(条件1, 函数1)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
condition	返回值为 bool 类型的函数表达式	bool	是	-	-
function	返回值为 LOG 类型的函数表达式	string	是	-	-

示例

- 示例1

原始日志：

```
{"condition": 1, "status": "500"}
```

加工规则：

```
t_if(True, fields_set("new", 1))
```

加工结果：

```
{"new":"1","condition":"1","status":"500"}
```

- 示例2

原始日志：

```
//如果 condition 字段的值为真，新增一个字段 new 并赋值1。
{"condition": 1, "status": "500"}
```

加工规则：

```
t_if(v("condition"), fields_set("new", 1))
```

加工结果：

```
{"new":"1","condition":"1","status":"500"}
```

t_if_not 函数

函数定义

对不符合条件的日志，进行相应的函数处理，否则不进行任何处理。

语法描述

```
t_if_not(条件1, 函数1)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
condition	返回值为 bool 类型的函数表达式	bool	是	-	-
function	返回值为 LOG 类型的函数表达式	string	是	-	-

示例

原始日志：

```
{"condition": 0, "status": "500"}
```

加工规则：

```
t_if_not(v("condition"), fields_set("new", 1))
```

加工结果：

```
{"new":"1","condition":"0","status":"500"}
```

t_if_else 函数

函数定义

基于条件判断，分别进行不同的函数处理。

语法描述

```
t_if_else(条件1, 函数1, 函数2)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
condition	返回值为 bool 类型的函数表达式	bool	是	-	-

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
function	返回值为 LOG 类型的函数表达式	string	是	-	-
function	返回值为 LOG 类型的函数表达式	string	是	-	-

示例

原始日志：

```
{"condition": 1, "status": "500"}
```

加工规则：

```
t_if_else(v("condition"), fields_set("new", 1), fields_set("new", 2))
```

加工结果：

```
{"new":"1","condition":"1","status":"500"}
```

t_switch 函数

函数定义

基于多分支条件，分别进行不同的函数处理，如果存在不符合所有条件的数据，将被丢弃。

语法描述

```
t_switch(条件1, 函数1, 条件2, 函数2, ...)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
可变参，交替的条件和函数表达式作为参数	参考 t_if 函数，类似多个 t_if 组合	-	-	-	-

示例

原始日志：

```
{"condition1": 0,"condition2": 1, "status": "500"}
```

加工规则：

```
//condition1字段值为真时，新增字段new并赋值1，由于此处字段值返回假，因此不会新增"new":"1",第二个条件返回真，所以新增了"new":"2"
t_switch(v("condition1"), fields_set("new", 1), v("condition2"), fields_set("new", 2))
```

加工结果：

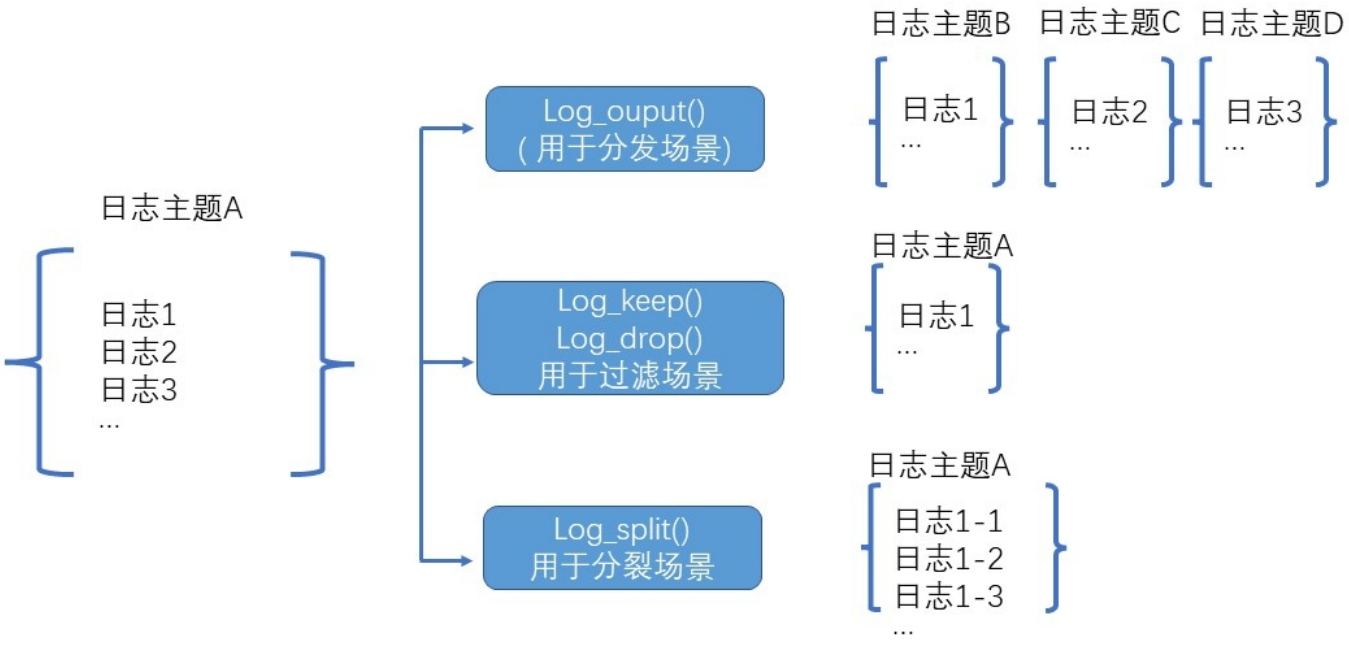
```
{"new":"2","condition1":"0","condition2":"1","status":"500"}
```

行处理函数

最近更新时间：2022-07-04 09:17:13

简介

对一行日志进行处理，包括过滤、分发、分裂等函数。



log_output 函数

函数定义

输出到指定的目标主题。可以配合分支条件使用，也可以单独使用。

语法描述

log_output(别名)，别名在配置加工任务时定义。如下图所示：

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
alias	目标主题的别名	string	是	-	-

示例

按照 loglevel 字段值为 waring/info/error 的情况，分发到三个不同的日志主题中。

原始日志：

```
[
{
"loglevel": "warning"
},
{
"loglevel": "info"
},
{
"loglevel": "error"
```

```
}
]
```

加工规则：

```
//按照loglevel字段值为waring/info/error的情况，分发到三个不同的日志主题中。
t_switch(regex_match(v("loglevel"),regex="info"),log_output("info_log"),regex_match(v("loglevel"),regex="warning"),log_output("warning_log"),regex_match(v("loglevel"),regex="error"),log_output("error_log"))
```

加工结果：如下图所示

log_split 函数

函数定义

使用分隔符结合 jmes 表达式，对特定字段进行拆分，拆分结果分裂为多行日志。

语法描述

```
log_split(字段名, sep=",", quote="\"", jmes="", output="")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
field	待提取的字段名	string	是	-	-
sep	分隔符	string	否	,	任意单字符
quote	将值包括起来的字符	string	否	-	-
jmes	jmes 表达式，详情请参考 JMESPath	string	否	-	-
output	单个字段名	string	是	-	-

示例

- **示例1：字段中有多个值的日志分裂**

```
{"field": "hello Go,hello Java,hello python","status":"500"}
```

加工规则：

```
//使用分割符“，”，将日志分隔成三条。
log_split("field", sep=",", output="new_field")
```

加工结果：

```
{"new_field":"hello Go","status":"500"}
{"new_field":"hello Java","status":"500"}
{"new_field":"hello python","status":"500"}
```

- **示例2：利用 JMES 对日志进行分裂。**

```
{"field": "{\"a\":{\"b\":{\"c\":{\"d\":\"a,b,c\"}}}}", "status": "500"}
```

加工规则：

```
//a.b.c.d节点的值为“a,b,c”
log_split("field", jmes="a.b.c.d", output="new_field")
```

加工结果：

```
{"new_field":"a","status":"500"}
{"new_field":"b","status":"500"}
{"new_field":"c","status":"500"}
```

- 示例3：包含有 JSON 数组的日志分裂。

```
{"field": "{\"a\":{\"b\":{\"c\":{\"d\":[\"a\",\"b\",\"c\"]}}}}", "status": "500"}
```

加工规则：

```
log_split("field", jmes="a.b.c.d", output="new_field")
```

加工结果：

```
{"new_field":"a","status":"500"}
{"new_field":"b","status":"500"}
{"new_field":"c","status":"500"}
```

log_drop 函数

函数定义

丢弃符合条件的日志。

语法描述

```
log_drop(条件1)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
condition	返回值为 bool 类型的函数表达式	bool	是	-	-

示例

丢弃 status=200的日志，其余保留。

原始日志：

```
{"field": "a,b,c", "status": "500"}
{"field": "a,b,c", "status": "200"}
```

加工规则：

```
log_drop(op_eq(v("status"), 200))
```

加工结果：

```
{"field":"a,b,c","status":"500"}
```

log_keep 函数

函数定义

保留符合条件的日志。

语法描述

```
log_keep(条件1)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
condition	返回值为 bool 类型的函数表达式	bool	是	-	-

示例

保留 status=500的日志，其余丢弃。

原始日志：

```
{"field": "a,b,c", "status": "500"}
{"field": "a,b,c", "status": "200"}
```

加工规则：

```
log_keep(op_eq(v("status"), 500))
```

加工结果：

```
{"field":"a,b,c","status":"500"}
```

log_split_jsonarray_jmes 函数

函数定义

将日志根据 jmes 语法将 JSON 数组拆分和展开。

语法描述

```
log_split_jsonarray_jmes("field", jmes="items", prefix="")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
field	待提取的字段名	string	是	-	-

示例

- 示例1

  原始日志：

  ```
  {"common":"common","result":"{\"target\":[{\"a\":\"a\"},{\"b\":\"b\"}]}"}
  ```

  加工规则：

  ```
  log_split_jsonarray_jmes("result",jmes="target")
  fields_drop("result")
  ```

加工结果：

```
{"common":"common", "a":"a"}
{"common":"common", "b":"b"}
```

- 示例2

原始日志：

```
{"common":"common","target":"[{\"a\":\"a\"},{\"b\":\"b\"}]"}
```

加工规则：

```
log_split_jsonarray_jmes("target",prefix="prefix_")
fields_drop("target")
```

加工结果：

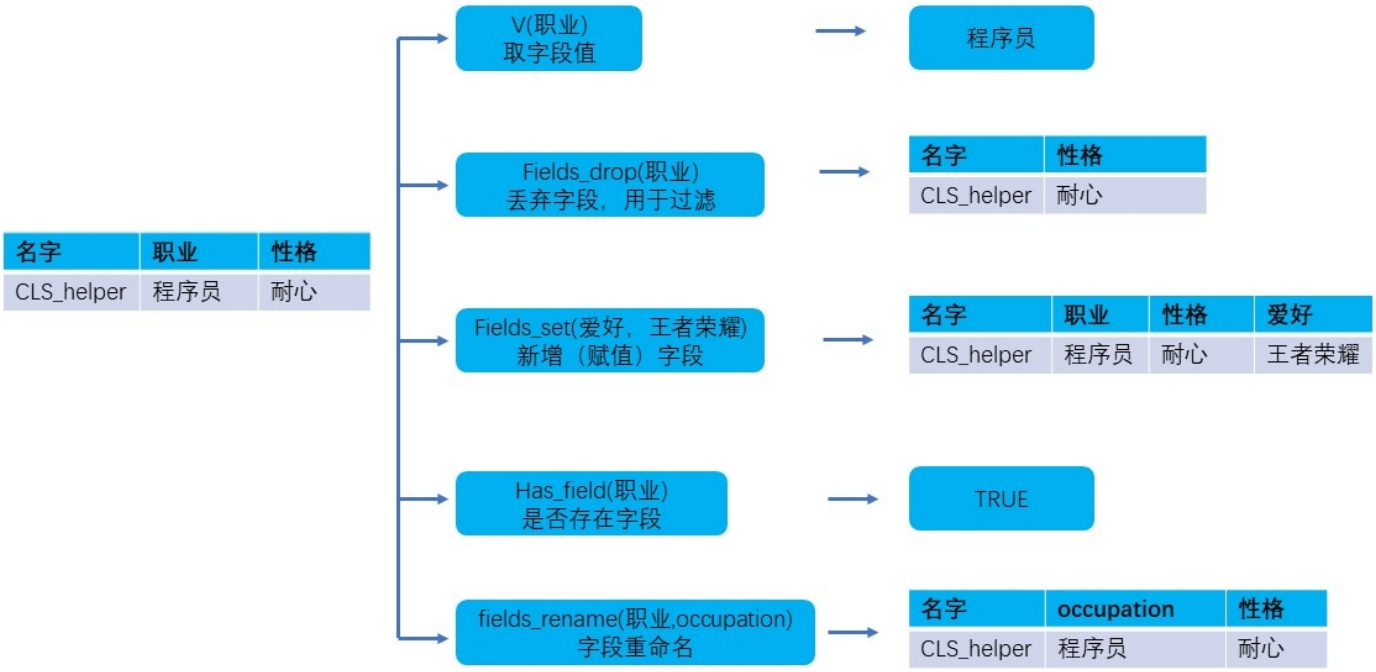
```
{"prefix_a":"a", "common":"common"}
{"prefix_b":"b", "common":"common"}
```

字段处理函数

最近更新时间：2022-03-29 16:05:22

简介

字段处理函数，顾名思义，就是对日志中的字段进行处理，如下图所示：



v 函数

函数定义

获取字段值，返回对应字符串。

语法描述

```
v(字段名)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
field	字段名	string	是	-	-

示例

获取 "message" 字段的值，将它赋值给一个新字段 "new_message"。

原始日志：

```
{"message": "failed", "status": "500"}
```

加工规则：

```
fields_set("new_message", v("message"))
```

加工结果：

```
{"message": "failed", "new_message": "failed","status": "500"}
```

fields_drop 函数

函数定义

根据字段名进行匹配，丢弃匹配到的字段。

语法描述

```
fields_drop(字段名1, 字段名2, ..., regex=False,nest=False)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
可变参，字段名或字段名的正则表达式	可变参，字段名或字段名的正则表达式	string	是	-	-
regex	是否启用正则表达式，匹配模式为全匹配	bool	否	False	-
nest	是否是嵌套字段	bool	否	False	-

示例

- 示例1：丢弃字段名为 "field" 的字段。

  原始日志：

  ```
  {"field": "a,b,c", "status": "500"}
  ```

  加工规则：

  ```
  fields_drop("field")
  ```

  加工结果：

  ```
  {"status":"500"}
  ```

- 示例2：嵌套字段的处理。

  原始日志：

  ```
  {"condition":"{\"a\":\"aaa\", \"c\":\"ccc\", \"e\":\"eee\"}","status":"500"}
  ```

  加工规则：

  ```
  //nest=True,表示该字段是嵌套字段，将condition.a和condition.c丢弃后，只剩下condition.e字段。
  t_if(if_json(v("condition")), fields_drop("condition.a", "condition.c", nest=True))
  ```

  加工结果：

  ```
  {"condition":"{\"e\":\"eee\"}","status":"500"}
  ```

- 示例3：嵌套字段的处理。

  原始日志：

  ```
  {"App": "thcomm","Message": "{\"f_httpstatus\": \"200\",\"f_requestId\": \"2021-11-09 08:40:17.832\tINFO\tservices/http_service.go:361\tbb20ac02-fcbc-4a56-b1f1-4064853b79da\",\"f_url\": \"wechat.wecity.qq.com/trpcapi/MbpsPaymentServer/scanCode\"}"}
  ```

加工规则：

```
//nest=True,表示该字段是嵌套字段，将Message.f_requestId", Message.f_url丢弃后，只剩下字f_httpstatus段。
t_if(if_json(v("Message")), fields_drop("Message.f_requestId", "Message.f_url", nest=True))
```

加工结果：

```
{"App":"thcomm","Message":"{\"f_httpstatus\":\"200\"}"}
```

fields_keep 函数

函数定义

根据字段名进行匹配，保留匹配到的字段

语法描述

```
fields_keep(字段名1, 字段名2, ..., regex=False)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
可变参，字段名或字段名的正则表达式	可变参，字段名或字段名的正则表达式	string	是	-	-
regex	是否启用正则表达式，匹配模式为全匹配	bool	否	False	-

示例

保存字段名为 "field" 的字段，其余字段丢弃。

原始日志：

```
{"field": "a,b,c", "status": "500"}
```

加工规则：

```
fields_keep("field")
```

加工结果：

```
{"field":"a,b,c"}
```

fields_pack 函数

函数定义

根据正则表达式来匹配字段名，并将匹配到的字段打包到新的字段，新字段值使用 JSON 格式进行组织。

语法描述

```
fields_pack(目标字段名, include=".*", exclude="", drop_packed=False)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
output	打包后的新字段名称	string	是	-	-
include	包含字段名称的正则	string	否	-	-

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
exclude	排除字段名称的正则	string	否	-	-
drop_packed	是否删除原有被打包的字段	bool	否	False	-

示例

原始日志：

```
{"field_a": "a,b,c","field_b": "abc", "status": "500"}
```

加工规则：

```
fields_pack("new_field","field.*", drop_packed=False)
```

加工结果：

```
{"new_field":"{\"field_a\":\"a,b,c\",\"field_b\":\"abc\"}","field_a":"a,b,c","field_b":"abc","status":"500"}
```

fields_set 函数

函数定义

用来设置字段值，或者增加新字段。

语法描述

```
fields_set(字段名1, 字段值1, 字段名2, 字段值2, mode="overwrite")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
可变参	key、value 交替的参数列表	string	-	-	-
mode	字段覆写模式	string	否	overwrite	-

示例

- **示例1：将级别从 “Info” 修改为 “Waring”。**

  原始日志：

  ```
  {"级别": "Info"}
  ```

  加工规则：

  ```
  fields_set("级别", "Warning")
  ```

  加工结果：

  ```
  {"级别", "Warning"}
  ```

- **示例2：新增两个字段，"new"和"new2"。**

  原始日志：

  ```
  {"a": "1", "b": "2", "c": "3"}
  ```

  加工规则：

```
fields_set("new", v("b"), "new2", v("c"))
```

加工结果：

```
{"a":"1","b":"2","c":"3","new":"2","new2":"3"}
```

fields_rename 函数

函数定义

字段重命名。

语法描述

```
fields_rename(字段名1, 新字段名1, 字段名2, 新字段名2, regex=False)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
可变参	旧字段名、新字段名交替的参数列表	string	-	-	-
regex	字段名是否启用正则匹配，启用使用正则匹配旧字段名，不启用则是相等匹配	bool	否	False	-

示例

原始日志：

```
{"regieeen": "bj", "status": "500"}
```

加工规则：

```
fields_rename("reg.*", "region", regex=True)
```

加工结果：

```
{"region":"bj","status":"500"}
```

has_field 函数

函数定义

字段存在时，返回 True，否则返回 False。

语法描述

```
has_field(字段名)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
field	字段名	string	是	-	-

示例

原始日志：

```
{"regiooon": "bj", "status": "500"}
```

加工规则：

```
t_if(has_field("regiooon"), fields_rename("regiooon", "region"))
```

加工结果：

```
{"region":"bj","status":"500"}
```

not_has_field 函数

函数定义

字段不存在时，返回 True，否则返回 False。

语法描述

```
not_has_field(字段名)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
field	字段名	string	是	-	-

示例

原始日志：

```
{"status": "500"}
```

加工规则：

```
t_if(not_has_field("message"), fields_set("no_message", True))
```

加工结果：

```
{"no_message":"TRUE","status":"500"}
```

值结构化处理函数

最近更新时间：2022-03-10 11:33:14

简介

值结构化处理函数，包括 JSON 指定节点的值选取，XML 和 JSON 互转，判断值是否是 JSON。

json_select 函数

函数定义

通过 jmes 表达式，提取 JSON 字段值，并返回 jmes 提取结果的 JSON 字符串。

语法描述

```
json_select(v(字段名), jmes="")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字段值，可以通过其他函数提取字段值	string	是	-	-
jmes	jmes 表达式	string	是	-	-

示例

原始日志：

```
{"field": "{\"a\":{\"b\":{\"c\":{\"d\":\"success\"}}}}", "status": "500"}
```

加工规则：

```
fields_set("message", json_select(v("field"), jmes="a.b.c.d"))
```

加工结果：

```
{"field":"{\"a\":{\"b\":{\"c\":{\"d\":\"success\"}}}}","message":"success","status":"500"}
```

xml_to_json 函数

函数定义

解析 XML 值并转换为 JSON 字符串，输入值必须为 XML 字符串结构，否则会导致转换异常。

语法描述

```
xml_to_json(字段值)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字段值	string	是	-	-

示例

原始日志：

{"xml_field": "<note><to>B</to><from>A</from><heading>Reminder</heading><body>Don't forget me this weekend!</body></note>", "status": "500"}

加工规则：

```
fields_set("json_field", xml_to_json(v("xml_field")))
```

加工结果：

{"xml_field":"<note><to>B</to><from>A</from><heading>Reminder</heading><body>Don't forget me this weekend!</body></note>","json_field":"{\"to\":\"B\",\"from\":\"A\",\"heading\":\"Reminder\",\"body\":\"Don't forget me this weekend!\"}","status":"500"}

json_to_xml 函数

函数定义

解析 JSON 字符串值并转换为 XML 字符串。

语法描述

```
json_to_xml(字段值)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字段值	string	是	-	-

示例

原始日志：

{"json_field":"{\"to\":\"B\",\"from\":\"A\",\"heading\":\"Reminder\",\"body\":\"Don't forget me this weekend!\"}", "status": "200"}

加工规则：

```
fields_set("xml_field", json_to_xml(v("json_field")))
```

加工结果：

{"json_field":"{\"to\":\"B\",\"from\":\"A\",\"heading\":\"Reminder\",\"body\":\"Don't forget me this weekend!\"}","xml_field":"<ObjectNode><to>B</to><from>A</from><heading>Reminder</heading><body>Don't forget me this weekend!</body></ObjectNode>","status":"200"}

if_json 函数

函数定义

判断是否为 JSON 字符串。

语法描述

```
if_json(字段值)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字段值	string	是	-	-

示例

- **示例1**

  原始日志：

  ```
  {"condition":"{\"a\":\"b\"}","status":"500"}
  ```

  加工语句：

  ```
  t_if(if_json(v("condition")), fields_set("new", 1))
  ```

  加工结果：

  ```
  {"new":"1","condition":"{\"a\":\"b\"}","status":"500"}
  ```

- **示例2**

  原始日志：

  ```
  {"condition":"haha","status":"500"}
  ```

  加工语句：

  ```
  t_if(if_json(v("condition")), fields_set("new", 1))
  ```

  加工结果：

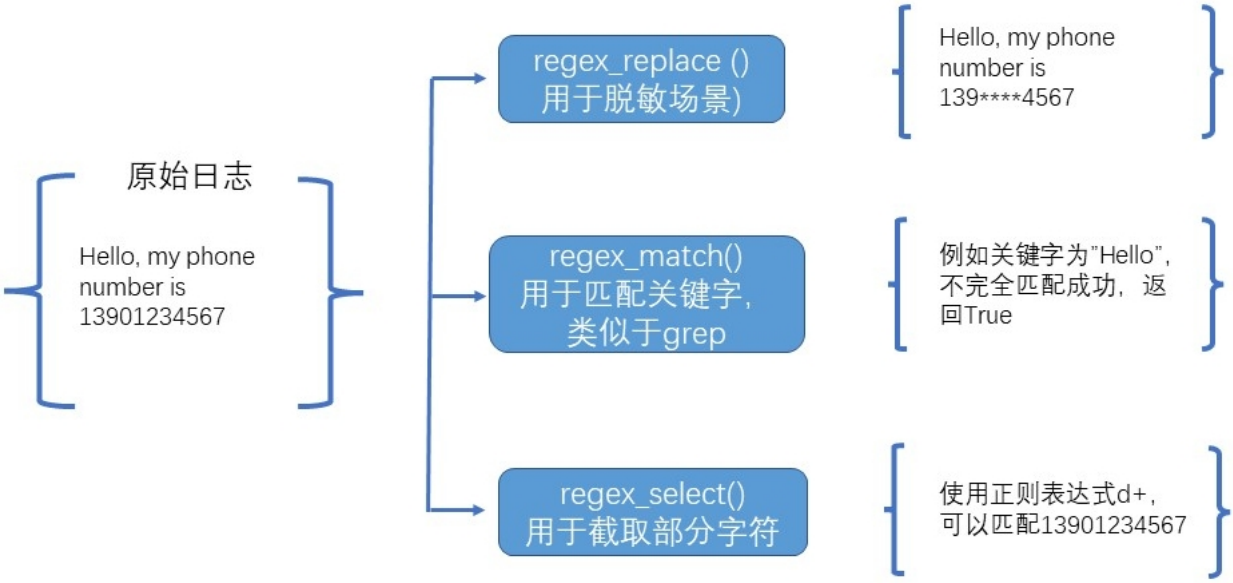
  ```
  {"condition":"haha","status":"500"}
  ```

正则处理函数

最近更新时间：2022-03-29 16:05:29

简介

日志含有大量的文本，在对文本的处理过程中，正则函数可以较为灵活的提取关键字、做脱敏、或者判断是否包含有指定的字符。如下图所示：



日志场景中常见的正则举例，可以 在线测试正则公式。

用途	日志原文	正则表达式	提取结果
提取大括号中的内容	[2021-11-24 11:11:08,232][328495eb-b562-478f-9d5d-3bf7e][INFO] curl -H 'Host: ' http://abc.com:8080/pc/api -d '{"version": "1.0", "user": "CGW", "password": "123", "timestamp": 1637723468, "interface": {"Name": "ListDetail", "para": {"owner": "1253", "limit": [10, 14], "orderField": "createTime"}}}	\{[^\}]+\}	{"version": "1.0", "user": "CGW", "password": "123", "timestamp": 1637723468, "interface": {"Name": "ListDetail", "para": {"owner": "1253", "limit": [10, 10], "orderField": "createTime"}
提取中括号的内容	[2021-11-24 11:11:08,232][328495eb-b562-478f-9d5d-3bf7e][INFO] curl -H 'Host: ' http://abc.com:8080/pc/api -d '{"version": "1.0", "user": "CGW", "password": "123", "timestamp": 1637723468, "interface": {"Name": "ListDetail", "para": {"owner": "1253", "limit": [10, 14], "orderField": "createTime"}}}	\[\S+\]	[328495eb-b562-478f-9d5d-3bf7e] [INFO]
提取时间	[2021-11-24 11:11:08,232][328495eb-b562-478f-9d5d-3bf7e][INFO] curl -H 'Host: ' http://abc.com:8080/pc/api -d '{"version": "1.0", "user": "CGW", "password": "123", "timestamp": 1637723468, "interface": {"Name": "ListDetail", "para": {"owner": "1253", "limit": [10, 14], "orderField": "createTime"}}}	\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2},\d{3}	2021-11-08 11:11:08,232
提取特定长度的大写字符	[2021-11-24 11:11:08,232][328495eb-b562-478f-9d5d-3bf7e][INFO] curl -H 'Host: ' http://abc.com:8080/pc/api -d '{"version": "1.0", "user": "CGW", "password": "123", "timestamp": 1637723468, "interface": {"Name": "ListDetail", "para": {"owner": "1253", "limit": [10, 14], "orderField": "createTime"}}}	[A-Z]{4}	INFO

用途	日志原文	正则表达式	提取结果
提取特定长度的小写字符	[2021-11-24 11:11:08,232][328495eb-b562-478f-9d5d-3bf7e][INFO] curl -H 'Host: ' http://abc.com:8080/pc/api -d '{"version": "1.0", "user": "CGW", "password": "123", "timestamp": 1637723468, "interface": {"Name": "ListDetail", "para": {"owner": "1253", "limit": [10, 15], "orderField": "createTime"}}}	[a-z]{6}	versio passwo timest interf create
提取字母+数字	[2021-11-24 11:11:08,232][328495eb-b562-478f-9d5d-3bf7e][INFO] curl -H 'Host: ' http://abc.com:8080/pc/api -d '{"version": "1.0", "user": "CGW", "password": "123", "timestamp": 1637723468, "interface": {"Name": "ListDetail", "para": {"owner": "1253", "limit": [10, 14], "orderField": "createTime"}}}	([a-z]{3}):([0-9]{4})	com:8080

regex_match 函数

函数定义

基于正则对数据进行匹配，返回是否匹配成功，可以选择全匹配还是部分匹配。

语法描述

```
regex_match(字段值, regex="", full=True)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字段值	string	是	-	-
regex	正则表达式	string	是	-	-
full	是否开启全匹配，对于全匹配，必须值完全满足正则，部分匹配则表示值中部分内容满足正则	bool	否	True	-

示例

- **示例1：判断正则公式"192\.168.*"和字段 IP 的值192.168.0.1是否完全匹配（full=True），regex_match 函数返回 True。**

  原始日志：

  ```
  {"IP":"192.168.0.1", "status": "500"}
  ```

  加工规则：

  ```
  //判断正则公式"192\.168.*"和字段ip的值192.168.0.1是否完全匹配，将结果保存到新字段"matched"中。
  t_if(regex_match(v("IP"), regex="192\.168.*", full=True), fields_set("matched", True))
  ```

  加工结果：

  ```
  {"IP":"192.168.0.1","matched":"TRUE","status":"500"}
  ```

- **示例2：判断正则公式"192*"和字段 IP 的值192.168.0.1是否部分匹配（full=Flase），regex_match 函数返回 True。**

  原始日志：

  ```
  {"IP":"192.168.0.1", "status": "500"}
  ```

  加工规则：

  ```
  t_if(regex_match(v("ip"), regex="192", full=False), fields_set("matched", True))
  ```

  加工结果：

```
{"IP":"192.168.0.1","matched":"TRUE","status":"500"}
```

regex_select 函数

函数定义

基于正则对数据进行匹配，返回相应的部分匹配结果，可以指定匹配结果的第几个表达式，以及第几个分组（部分匹配+指定捕获组序号），如果最终没有匹配结果，则返回空字符串。

语法描述

```
regex_select(字段值, regex="", index=1, group=1)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字段值	string	是	-	-
regex	正则表达式	string	是	-	-
index	匹配结果中的第几个表达式	number	否	默认第一个	-
group	匹配结果中的第几个分组	number	否	默认第一个	-

示例

根据正则表达式，对字段值进行不同的截取。

原始日志：

```
{"data":"hello123,world456", "status": "500"}
```

加工规则：

```
fields_set("match_result", regex_select(v("data"), regex="[a-z]+(\d+)",index=0, group=0))
fields_set("match_result1", regex_select(v("data"), regex="[a-z]+(\d+)", index=1, group=0))
fields_set("match_result2", regex_select(v("data"), regex="([a-z]+)(\d+)",index=0, group=0))
fields_set("match_result3", regex_select(v("data"), regex="([a-z]+)(\d+)",index=0, group=1))
```

加工结果：

```
{"match_result2":"hello123","match_result1":"world456","data":"hello123,world456","match_result3":"hello","match_result":"hello123","status":"500"}
```

regex_split 函数

函数定义

基于正则对数据进行分割，返回 JSON array 字符串（部分匹配）。

语法描述

```
regex_split(字段值, regex=\"\", limit=100)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字段值	string	是	-	-

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
regex	正则表达式	string	是	-	-
limit	分割最大数组长度，当超过长度时，剩余未分割部分将作为一个元素，添加到数组	number	否	默认值100	-

示例

原始日志：

```
{"data":"hello123world456", "status": "500"}
```

加工规则：

```
fields_set("split_result", regex_split(v("data"), regex="\d+"))
```

加工结果：

```
{"data":"hello123world456","split_result":"[\"hello\",\"world\"]","status":"500"}
```

regex_replace 函数

函数定义

基于正则匹配并替换（部分匹配），主要用于脱敏场景。

语法描述

```
regex_replace(字段值, regex="", replace="", count=0)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字段值	string	是	-	-
regex	正则表达式	string	是	-	-
replace	目标字符串，使用此字符串替换匹配结果	string	是	-	-
count	替换次数，默认0，进行全部替换	number	否	默认值0	-

示例

- **示例1：根据正则表达式，对字段值进行替换。**
  原始日志：

  ```
  {"data":"hello123world456", "status": "500"}
  ```

  加工规则：

  ```
  fields_set("replace_result", regex_replace(v("data"), regex="\d+", replace="", count=0))
  ```

  加工结果：

  ```
  {"replace_result":"helloworld","data":"hello123world456","status":"500"}
  ```

- **示例2：对用户 ID、手机号码、IP 地址进行脱敏。**
  原始日志：

```
{"Id": "dev@12345","Ip": "11.111.137.225","phonenumber": "13912345678"}
```

加工规则：

```
//对 Id 字段进行脱敏处理，结果为dev@***45
fields_set("Id",regex_replace(v("Id"),regex="\d{3}", replace="***",count=0))
fields_set("Id",regex_replace(v("Id"),regex="\S{2}", replace="**",count=1))
//对 phonenumber 字段进行脱敏处理，将中间的4位数替换为****，结果为139****5678
fields_set("phonenumber",regex_replace(v("phonenumber"),regex="(\d{0,3})\d{4}(\d{4})", replace="$1****$2"))
//对 IP 字段进行脱敏处理，将第二段替换为***，结果为11.***137.225。
fields_set("Ip",regex_replace(v("Ip"),regex="(\d+\.)\d+(\.\d+\.\d+)", replace="$1***$2",count=0))
```

加工结果：

```
{"Id":"**v@***45","Ip":"11.***.137.225","phonenumber":"139****5678"}
```

regex_findall 函数

函数定义

基于正则进行匹配，并将匹配结果添加到 JSON 数组中，并返回 array 字符串（部分匹配）。

语法描述

```
regex_findall(字段值, regex="")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字段值	string	是	-	-
regex	正则表达式	string	是	-	-

示例

原始日志：

```
{"data":"hello123world456", "status": "500"}
```

加工规则：

```
fields_set("result", regex_findall(v("data"), regex="\d+"))
```

加工结果：

```
{"result":"[\"123\",\"456\"]","data":"hello123world456","status":"500"}
```

时间值处理函数

最近更新时间：2022-03-29 16:06:12

简介

日志中的时间处理函数，包括将 Date 类型转为 String 类型，时间类字段值和 UTC 时间互转，以及获取当前时间。

dt_str 函数

函数定义

将时间类的字段值（特定格式的日期字符串或者时间戳），转换为指定时区、格式的目标日期字符串。

语法描述

```
dt_str(值, format="格式化字符串", zone="")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字段值，可支持的解析格式可参考 dateparser	string	是	-	-
format	格式化日期格式可参考 DateTimeFormatter	string	否	-	-
zone	默认 UTC 时间，不指定时区。时区定义可参考 ZoneId	string	否	UTC+00:00	-

示例

原始日志：

```
{"date":"2014-04-26 13:13:44 +09:00"}
```

加工规则：

```
fields_set("result", dt_str(v("date"), format="yyyy-MM-dd HH:mm:ss", zone="UTC+8"))
```

加工结果：

```
{"date":"2014-04-26 13:13:44 +09:00","result":"2014-04-26 12:13:44"}
```

dt_to_timestamp 函数

函数定义

将时间类的字段值（特定格式的日期字符串），同时指定字段对应的时区，转换为 UTC 时间戳。

语法描述

```
dt_to_timestamp(值, zone="")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字段值，可支持的解析格式可参考 dateparser	string	是	-	-
zone	默认 UTC 时间，不指定时区。如果指定此值，则必须和时间字段值对应，否则会出现时区错误问题。时区定义可参考 ZoneId	string	否	UTC+00:00	-

示例

原始日志：

```
{"date":"2021-10-26 15:48:15"}
```

加工规则：

```
fields_set("result", dt_to_timestamp(v("date"), zone="UTC+8"))
```

加工结果：

```
{"date":"2021-10-26 15:48:15","result":"1635234495000"}
```

dt_from_timestamp 函数

函数定义

将时间类的时间戳字段，指定目标时区后，转换为时间字符串。

语法描述

```
dt_from_timestamp(值, zone="")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字段值，可支持的解析格式可参考 dateparser	string	是	-	-
zone	默认 UTC 时间，不指定时区。时区定义可参考 ZoneId	string	否	UTC+00:00	-

示例

原始日志：

```
{"date":"1635234495000"}
```

加工规则：

```
fields_set("result", dt_from_timestamp(v("date"), zone="UTC+8"))
```

加工结果：

```
{"date":"1635234495000","result":"2021-10-26 15:48:15"}
```

dt_now 函数

函数定义

获取加工计算时的本地时间。

语法描述

```
dt_now(format="格式化字符串", zone="")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
format	格式化日期格式，可参考DateTimeFormatter	string	否	-	-
zone	默认 UTC 时间，不指定时区。时区定义可参考 ZoneId	string	否	UTC+00:00	-

示例

原始日志：

```
{"date":"1635234495000"}
```

加工规则：

```
fields_set("now", dt_now(format="yyyy-MM-dd HH:mm:ss", zone="UTC+8"))
```

加工结果，仅参考，具体结果和系统时间相关：

```
{"date":"1635234495000","now":"2021-MM-dd HH:mm:ss"}
```

字符串处理函数

最近更新时间：2022-03-10 16:51:09

简介

字符串处理函数，支持字符串长度计算、大小写转换、拼接字符串、替换、剔除、查找字符所在位置、前后缀匹配等。
需要注意的是，正则和字符串函数使用的场景并不相同。**正则更加适合于**在非结构化的日志中提取字段和字段值。例如，在如下的日志中提取 log_time、log_level，正则更为合适。

```
{
“日志内容”: "2021-12-02 14:33:35.022 [1] INFO org.apache.Load - Response:status: 200, resp msg: OK, resp content: { \"TxnId\": 58322, \"Label\": \"flink_connector_20211202_1de749d8c80015a8\", \"Status\": \"Success\", \"Message\": \"OK\", \"TotalRows\": 1, \"LoadedRows\": 1, \"FilteredRows\": 0, \"CommitAndPublishTimeMs\": 16}"
}
```

而字符串函数更加适用于在结构化日志数据中，对某个字段值进行处理。例如：

```
"resonsebody": {"method": "GET","user": "Tom"}
```

str_count 函数

函数定义

在值中指定范围内查找子串，返回子串出现的次数。

语法描述

```
str_count(值, sub="", start=0, end=-1)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字符串类型的值	string	是	-	-
sub	准备查找的子字符串	string	是	-	-
start	查找的起始位置	number	否	默认为0	-
end	查找的结束位置	number	否	默认为-1	-

示例

原始日志：

```
{"data": "warn,error,error"}
```

加工规则：

```
fields_set("result", str_count(v("data"), sub="err"))
```

加工结果：

```
{"result":"2","data":"warn,error,error"}
```

str_len 函数

函数定义

返回字符串长度。

语法描述

```
str_len(值)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字符串类型的值	string	是	-	-

示例

原始日志：

```
{"data": "warn,error,error"}
```

加工规则：

```
fields_set("result", str_len(v("data")))
```

加工结果：

```
{"result":"16","data":"warn,error,error"}
```

str_uppercase 函数

函数定义

返回大写字符串。

语法描述

```
str_uppercase(值)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字符串类型的值	string	是	-	-

示例

原始日志：

```
{"data": "warn,error,error"}
```

加工规则：

```
fields_set("result", str_uppercase(v("data")))
```

加工结果：

```
{"result":"WARN,ERROR,ERROR","data":"warn,error,error"}
```

str_lowercase 函数

函数定义

返回小写字符串。

语法描述

```
str_lowercase(值)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字符串类型的值	string	是	-	-

示例

原始日志：

```
fields_set("result", str_lowercase(v("data")))
```

加工规则：

```
{"data": "WARN,ERROR,ERROR"}
```

加工结果：

```
{"result":"warn,error,error","data":"WARN,ERROR,ERROR"}
```

str_join 函数

函数定义

使用拼接字符串，拼接多值。

语法描述

```
str_join(拼接字符串1, 值1, 值2, ...)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
join	字符串类型的值	string	是	-	-
值参数，可变参数列表	字符串类型的值	string	是	-	-

示例

原始日志：

```
{"data": "WARN,ERROR,ERROR"}
```

加工规则：

```
fields_set("result", str_join(",", v("data"), "INFO"))
```

加工结果：

```
{"result":"WARN,ERROR,ERROR,INFO","data":"WARN,ERROR,ERROR"}
```

str_replace 函数

函数定义

替换字符串，返回替换结果字符串。

语法描述

```
str_replace(值, old="", new="", count=0)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字符串类型的值	string	是	-	-
old	将要被替换的字符串	string	是	-	-
new	替换的目标字符串	string	是	-	-
count	最大替换次数，默认全部替换	number	否	默认为0	-

示例

将字段 data 的值中的 "WARN" 替换为 "ERROR"。

原始日志：

```
{"data": "WARN,ERROR,ERROR"}
```

加工规则：

```
fields_set("result", str_replace( v("data"), old="WARN", new="ERROR"))
```

将替换的结果保存到新字段 "result" 中。

加工结果：

```
{"result":"ERROR,ERROR,ERROR","data":"WARN,ERROR,ERROR"}
```

str_format 函数

函数定义

格式化字符串，返回格式化结果。

语法描述

```
str_format(格式化字符串, 值1, 值2, ...)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
format	目标格式，占位符使用{}，例如："The disk "{1}" contains {0} file(s)."。其中{}中的数字，对应后面参数值的序号，从0开始。具体使用可参考 MessageFormat.format 函数的使用	string	是	-	-
值参数，可变参数列表	字符串类型的值	string	是	-	-

示例

原始日志：

```
{"status": 200, "message":"OK"}
```

加工规则：

```
fields_set("result", str_format("status:{0}, message:{1}", v("status"), v("message")))
```

加工结果：

```
{"result":"status:200, message:OK","message":"OK","status":"200"}
```

str_strip 函数

函数定义

剔除用户指定的字符序列中的字符，从字符串开头和结尾同时剔除，返回剔除后的结果。

语法描述

```
str_strip(值, chars="\t\r\n")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字符串类型的值	string	是	-	-
chars	将要被剔除的字符序串	string	否	\t\r\n	-

示例

- 示例1

  原始日志：

  ```
  {"data": " abc "}
  ```

  加工规则：

  ```
  fields_set("result", str_strip(v("data"), chars=" "))
  ```

  加工结果：

  ```
  {"result":"abc","data":" abc "}
  ```

- 示例2

  原始日志：

  ```
  {"data": " **abc** "}
  ```

  加工规则：

  ```
  fields_set("result", str_strip(v("data"), chars=" *"))
  ```

  加工结果：

  ```
  {"result":"abc","data":" **abc** "}
  ```

str_lstrip 函数

函数定义

剔除用户指定的字符序列中的字符，从字符串左侧开头剔除，返回剔除后的结果。

语法描述

```
str_strip(值, chars="\t\r\n")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字符串类型的值	string	是	-	-
chars	将要被剔除的字符序串	string	否	\t\r\n	-

示例

原始日志：

```
{"data": " abc "}
```

加工规则：

```
fields_set("result", str_lstrip(v("data"), chars=" "))
```

加工结果：

```
{"result":"abc ","data":" abc "}
```

str_rstrip 函数

函数定义

剔除用户指定的字符序列中的字符，从字符串右侧结尾部分剔除，返回剔除后的结果。

语法描述

```
str_strip(值, chars="\t\r\n")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字符串类型的值	string	是	-	-
chars	将要被剔除的字符序串	string	否	\t\r\n	-

示例

原始日志：

```
{"data": " abc "}
```

加工规则：

```
fields_set("result", str_rstrip(v("data"), chars=" "))
```

加工结果：

```
{"result":" abc","data":" abc "}
```

str_find 函数

函数定义

在值中查找子串，并返回子串出现的位置。

语法描述

```
str_find(值, sub="", start=0, end=-1)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字符串类型的值	string	是	-	-
sub	准备查找的子字符串	string	是	-	-
start	查找的起始位置	number	否	默认为0	-
end	查找的结束位置	number	否	默认为-1	-

示例

原始日志：

```
{"data": "warn,error,error"}
```

加工规则：

```
fields_set("result", str_find(v("data"), sub="err"))
```

加工结果：

```
{"result":"5","data":"warn,error,error"}
```

str_start_with 函数

函数定义

判断字符串是否以指定前缀开头。

语法描述

```
str_start_with(值, sub="", start=0, end=-1)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字符串类型的值	string	是	-	-
sub	前缀字符串或字符	string	是	-	-
start	查找的起始位置	number	否	默认为0	-
end	查找的结束位置	number	否	默认为-1	-

示例

- 示例1

  原始日志：

  ```
  {"data": "something"}
  ```

加工规则：

```
fields_set("result", str_start_with(v("data"), sub="some"))
```

加工结果：

```
{"result":"true","data":"something"}
```

- 示例2

原始日志：

```
{"data": "something"}
```

加工规则：

```
fields_set("result", str_start_with(v("data"), sub="*"))
```

加工结果：

```
{"result":"false","data":"something"}
```

str_end_with 函数

函数定义

判断字符串是否以指定前缀开头。

语法描述

```
str_end_with(值, sub="", start=0, end=-1)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	字符串类型的值	string	是	-	-
sub	前缀字符串或字符	string	是	-	-
start	查找的起始位置	number	否	默认为0	-
end	查找的结束位置	number	否	默认为-1	-

示例

原始日志：

```
{"data": "endwith something"}
```

加工规则：

```
fields_set("result", str_end_with(v("data"), sub="ing"))
```

加工结果：

```
{"result":"true","data":"endwith something"}
```

类型转换函数

最近更新时间：2022-03-11 18:41:33

简介

函数提供了常见的类型转换，将字段值专为 int，float，bool 和 Str。

ct_int 函数

函数定义

对值进行整型转换，可指定原值的进制，转为十进制数值。

语法描述

```
ct_int(值1, base=10)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	数值类型的值或可转为数值的字符串	number	是	-	-
base	进制	number	否	默认为10	[2-36]

示例

- 示例1

  原始日志：

  ```
  {"field1": "10"}
  ```

  加工规则：

  ```
  fields_set("result", ct_int(v("field1")))
  ```

  加工结果：

  ```
  {"result":"10","field1":"10"}
  ```

- 示例2

  原始日志：

  ```
  {"field1": "AB"}
  ```

  加工规则：

  ```
  fields_set("result", ct_int(v("field1"), 16))
  ```

  加工结果：

  ```
  {"result":"171","field1":"AB"}
  ```

ct_float 函数

函数定义

将值转换为浮点型数值。

语法描述

```
ct_float(值)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	数值类型的值或可转为数值的字符串	number	是	-	-

示例

原始日志：

```
{"field1": "123"}
```

加工规则：

```
fields_set("result", ct_float(v("field1")))
```

加工结果：

```
{"result":"123.0","field1":"123"}
```

ct_str 函数

函数定义

将值转换为字符串。

语法描述

```
ct_str(值)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	数值类型的值或可转为数值的字符串	number	是	-	-

示例

原始日志：

```
{"field1": 123}
```

加工规则：

```
fields_set("result", ct_str(v("field1")))
```

加工结果：

```
{"result":"123","field1":"123"}
```

ct_bool 函数

函数定义

将值转换为布尔值。

语法描述

```
ct_bool(值)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	数值类型的值或可转为数值的字符串	number	是	-	-

示例

- 示例1

  原始日志：

  ```
  {}
  ```

  加工规则：

  ```
  fields_set("result", ct_bool(0))
  ```

  加工结果：

  ```
  {"result":"false"}
  ```

- 示例2

  原始日志：

  ```
  {}
  ```

  加工规则：

  ```
  fields_set("result", ct_bool(1))
  ```

  加工结果：

  ```
  {"result":"true"}
  ```

- 示例3

  原始日志：

  ```
  {"field1": 1}
  ```

  加工规则：

  ```
  fields_set("result", ct_bool(v("field1")))
  ```

  加工结果：

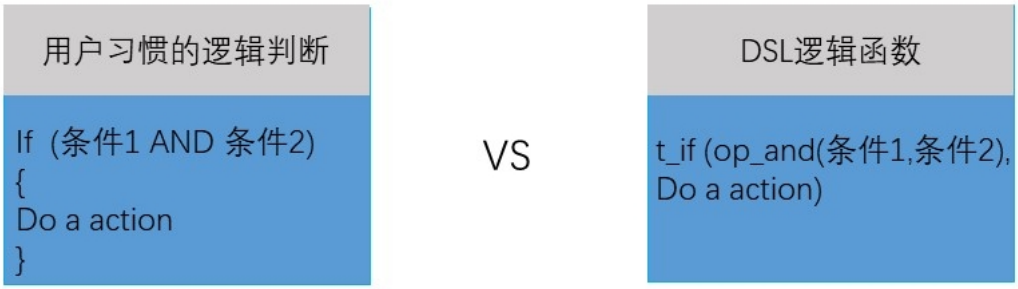
  ```
  {"result":"true","field1":"1"}
  ```

逻辑表达式函数

最近更新时间：2022-07-04 09:17:32

简介

逻辑与数学函数，包含了与或非、大于小于等于、加减乘除取模的运算。写起来和我们在程序语言中稍有区别，如下图所示：



op_if 函数

函数定义

根据条件判断，返回相应的值。

语法描述

```
op_if(条件1, 值1, 值2)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
condition	条件表达式	bool	是	-	-
data1	条件为 True 时，返回此参数值	string	是	-	-
data2	条件为 False 时，返回此参数值	string	是	-	-

示例

- 示例1

  原始日志：

  ```
  {"data": "abc"}
  ```

  加工规则：

  ```
  fields_set("result", op_if(True, v("data"), "false"))
  ```

  加工结果：

  ```
  {"result":"abc","data":"abc"}
  ```

- 示例2

  原始日志：

  ```
  {"data": "abc"}
  ```

加工规则：

```
fields_set("result", op_if(False, v("data"), "123"))
```

加工结果：

```
{"result":"123","data":"abc"}
```

op_and 函数

函数定义

对值进行 and 运算，均为 True 时，返回 True，否则返回 False。

语法描述

```
op_and(值1, 值2, ...)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
可变参列表	参与计算的参数或表达式	string	是	-	-

示例

- 示例1

  原始日志：

  ```
  {}
  ```

  加工规则：

  ```
  fields_set("result", op_and(True, False))
  ```

  加工结果：

  ```
  {"result":"false"}
  ```

- 示例2

  原始日志：

  ```
  {}
  ```

  加工规则：

  ```
  fields_set("result", op_and(1, 1))
  ```

  加工结果：

  ```
  {"result":"true"}
  ```

- 示例3

  原始日志：

  ```
  {"data":"false"}
  ```

加工规则：

```
fields_set("result", op_and(1, v("data")))
```

加工结果：

```
{"result":"false","data":"false"}
```

op_or 函数

函数定义

对值进行 or 运算，若存在参数值为 False 则返回 False，否则返回 True。

语法描述

```
op_or(值1, 值2, ...)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
可变参列表	参与计算的参数或表达式	string	是	-	-

示例

原始日志：

```
{}
```

加工规则：

```
fields_set("result", op_or(True, False))
```

加工结果：

```
{"result":"true"}
```

op_not 函数

函数定义

对值进行 not 运算。

语法描述

```
op_not(值)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	任意类型的值	any	是	-	-

示例

- 示例1

  原始日志：

```
{}
```

加工规则：

```
fields_set("result", op_not(True))
```

加工结果：

```
{"result":"false"}
```

- 示例2

原始日志：

```
{}
```

加工规则：

```
fields_set("result", op_not("True"))
```

加工结果：

```
{"result":"false"}
```

op_eq 函数

函数定义

对值进行比较，相等则返回 True。

语法描述

```
op_eq(值1, 值2)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data1	数值类型的值或可转为数值的字符串	number	是	-	-
data2	数值类型的值或可转为数值的字符串	number	是	-	-

示例

- 示例1：判断字段 "Post" 和 "Get" 的值是否相等。

原始日志：

```
{"Post": "10", "Get": "11"}
```

加工规则：

```
fields_set("result", op_eq(v("Post"), v("Get")))
```

将结果保存到result。

加工结果：

```
{"result":"false","Post":"10","Get":"11"}
```

- 示例2：判断字段 "field1" 和 "field2" 的值是否相等。

原始日志：

```
{"field1": "1", "field2": "1"}
```

加工规则：

```
fields_set("result", op_eq(v("field1"), v("field2")))
```

加工结果：

```
{"result":"true","field1":"1","field2":"1"}
```

op_ge 函数

函数定义

对值进行比较，值1大于或等于值2时返回 True。

语法描述

```
op_ge(值1, 值2)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data1	数值类型的值或可转为数值的字符串	number	是	-	-
data2	数值类型的值或可转为数值的字符串	number	是	-	-

示例

- 示例1

  原始日志：

  ```
  {"field1": "20", "field2": "9"}
  ```

  加工规则：

  ```
  fields_set("result", op_ge(v("field1"), v("field2")))
  ```

  加工结果：

  ```
  {"result":"true","field1":"20","field2":"9"}
  ```

- 示例2

  原始日志：

  ```
  {"field1": "2", "field2": "2"}
  ```

  加工规则：

  ```
  fields_set("result", op_ge(v("field1"), v("field2")))
  ```

  加工结果：

  ```
  {"result":"true","field1":"2","field2":"2"}
  ```

op_gt 函数

函数定义

对值进行比较，值1大于值2时返回 True。

语法描述

```
op_gt(值1, 值2)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data1	数值类型的值或可转为数值的字符串	number	是	-	-
data2	数值类型的值或可转为数值的字符串	number	是	-	-

示例

原始日志：

```
{"field1": "20", "field2": "9"}
```

加工规则：

```
fields_set("result", op_ge(v("field1"), v("field2")))
```

加工结果：

```
{"result":"true","field1":"20","field2":"9"}
```

op_le 函数

函数定义

对值进行比较，值1小于或等于值2时返回 True。

语法描述

```
op_le(值1, 值2)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data1	数值类型的值或可转为数值的字符串	number	是	-	-
data2	数值类型的值或可转为数值的字符串	number	是	-	-

示例

原始日志：

```
{"field1": "2", "field2": "2"}
```

加工规则：

```
fields_set("result", op_le(v("field1"), v("field2")))
```

加工结果：

```
{"result":"true","field1":"2","field2":"2"}
```

op_lt 函数

函数定义

对值进行比较，值1小于值2时返回 True。

语法描述

```
op_lt(值1, 值2)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data1	数值类型的值或可转为数值的字符串	number	是	-	-
data2	数值类型的值或可转为数值的字符串	number	是	-	-

示例

原始日志：

```
{"field1": "2", "field2": "3"}
```

加工规则：

```
fields_set("result", op_lt(v("field1"), v("field2")))
```

加工结果：

```
{"result":"true","field1":"2","field2":"3"}
```

op_add 函数

函数定义

对值进行求和运算。

语法描述

```
op_add(值1, 值2)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data1	数值类型的值或可转为数值的字符串	number	是	-	-
data2	数值类型的值或可转为数值的字符串	number	是	-	-

示例

原始日志：

```
{"field1": "1", "field2": "2"}
```

加工规则：

```
fields_set("result", op_add(v("field1"), v("field2")))
```

加工结果：

```
{"result":"3","field1":"1","field2":"2"}
```

op_sub 函数

函数定义

对值进行求差运算。

语法描述

op_sub(值1, 值2)

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data1	数值类型的值或可转为数值的字符串	number	是	-	-
data2	数值类型的值或可转为数值的字符串	number	是	-	-

示例

原始日志：

```
{"field1": "1", "field2": "2"}
```

加工规则：

```
fields_set("result", op_sub(v("field1"), v("field2")))
```

加工结果：

```
{"result":"-1","field1":"1","field2":"2"}
```

op_mul 函数

函数定义

对值进行乘积运算。

语法描述

```
op_mul(值1, 值2)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data1	数值类型的值或可转为数值的字符串	number	是	-	-
data2	数值类型的值或可转为数值的字符串	number	是	-	-

示例

原始日志：

```
{"field1": "1", "field2": "2"}
```

加工规则：

```
fields_set("result", op_mul(v("field1"), v("field2")))
```

加工结果：

```
{"result":"2","field1":"1","field2":"2"}
```

op_div 函数

函数定义

对值进行除法运算。

语法描述

```
op_div(值1, 值2)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data1	数值类型的值或可转为数值的字符串	number	是	-	-
data2	数值类型的值或可转为数值的字符串	number	是	-	-

示例

- 示例1

  原始日志：

  ```
  {"field1": "1", "field2": "2"}
  ```

  加工规则：

  ```
  fields_set("result", op_div(v("field1"), v("field2")))
  ```

  加工结果：

  ```
  {"result":"0","field1":"1","field2":"2"}
  ```

- 示例2

  原始日志：

  ```
  {"field1": "1.0", "field2": "2"}
  ```

  加工规则：

  ```
  fields_set("result", op_div(v("field1"), v("field2")))
  ```

  加工结果：

  ```
  {"result":"0.5","field1":"1.0","field2":"2"}
  ```

op_sum 函数

函数定义

对多值累加求和。

语法描述

```
op_sum(值1, 值2, ...)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
可变参列表	数值类型的值或可转为数值的字符串	string	是	-	-

示例

原始日志：

```
{"field1": "1.0", "field2": "10"}
```

加工规则：

```
fields_set("result", op_sum(v("field1"), v("field2")))
```

加工结果：

```
{"result":"11.0","field1":"1.0","field2":"10"}
```

op_mod 函数

函数定义

对值进行模计算。

语法描述

```
op_mod(值1, 值2)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data1	数值类型的值或可转为数值的字符串	number	是	-	-

示例

- **示例1**

  原始日志：

  ```
  {"field1": "1.0", "field2": "0"}
  ```

  加工规则：

  ```
  fields_set("result", op_mod(v("field1"), v("field2")))
  ```

  加工结果：

  ```
  {"result":"2","field1":"1","field2":"2"}
  ```

- **示例2**

  原始日志：

  ```
  {"field1": "1.0", "field2": "5"}
  ```

加工规则：

```
fields_set("result", op_mod(v("field1"), v("field2")))
```

加工结果：

```
{"result":"1.0","field1":"1.0","field2":"5"}
```

- 示例3

原始日志：

```
{"field1": "6", "field2": "4"}
```

加工规则：

```
fields_set("result", op_mod(v("field1"), v("field2")))
```

加工结果：

```
{"result":"2","field1":"6","field2":"4"}
```

op_null 函数

函数定义

对值进行是否为 null 判断，是则返回 true，否则返回 false。

语法描述

```
op_null(值)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	任意类型的值	any	是	-	-

示例

- 示例1

原始日志：

```
{}
```

加工规则：

```
fields_set("result", op_null("null"))
```

加工结果：

```
{"result":"true"}
```

- 示例2

原始日志：

```
{"data": null}
```

加工规则：

```
fields_set("result", op_null(v("data")))
```

加工结果：

```
{"data": "null", "result":"true"}
```

op_notnull 函数

函数定义

对值进行是否为非 null 判断，是则返回 true，否则返回 false。

语法描述

```
op_notnull(值)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	任意类型的值	any	是	-	-

示例

- 示例1

  原始日志：

  ```
  {}
  ```

  加工规则：

  ```
  fields_set("result", op_notnull("null"))
  ```

  加工结果：

  ```
  {"result":"false"}
  ```

- 示例2

  原始日志：

  ```
  {"data": null}
  ```

  加工规则：

  ```
  fields_set("result", op_notnull(v("data")))
  ```

  加工结果：

  ```
  {"data": "null", "result":"false"}
  ```

op_str_eq 函数

函数定义

对字符串值进行比较，相等则返回 true。

语法描述

```
op_str_eq(值1, 值2, ignore_upper=False)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data1	字符串类型的值	string	是	-	-
data2	字符串类型的值	string	是	-	-
ignore_upper	是否区分大小写	bool	否	False	-

示例

- 示例1

  原始日志：

  ```
  {"field": "cls"}
  ```

  加工规则：

  ```
  fields_set("result", op_str_eq(v("field"), "cls"))
  ```

  加工结果：

  ```
  {"result":"true","field":"cls"}
  ```

- 示例2

  原始日志：

  ```
  {"field": "cls"}
  ```

  加工规则：

  ```
  fields_set("result", op_str_eq(v("field"), "etl|cls|data"))
  ```

  加工结果：

  ```
  {"result":"true","field":"cls"}
  ```

- 示例3

  原始日志：

  ```
  {"field": "CLS"}
  ```

  加工规则：

  ```
  fields_set("result", op_str_eq(v("field"), "cls", ignore_upper=True))
  ```

  加工结果：

  ```
  {"result":"true","field":"CLS"}
  ```

- 示例4

  原始日志：

  ```
  {"field": "CLS"}
  ```

加工规则：

```
fields_set("result", op_str_eq(v("field"), "etl|cls|data", ignore_upper=True))
```

加工结果：

```
{"result":"true","field":"CLS"}
```

编解码函数

最近更新时间：2022-07-04 09:17:52

decode_url 函数

函数定义

将编码 URL 进行解码。

语法描述

```
decode_url(值)
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
url	URL 值	string	是	-	-

示例

原始日志：

```
{"url":"https%3A%2F%2Fcloud.tencent.com%2F"}
```

加工规则：

```
fields_set("result",decode_url(v("url")))
```

加工结果：

```
{"result":"https://cloud.tencent.com/","url":"https%3A%2F%2Fcloud.tencent.com%2F"}
```

IP 解析函数

最近更新时间：2022-07-04 09:18:13

geo_parse 函数

函数定义

解析出函数的地理位置。

语法描述

```
geo_parse(字段值, keep=("country","province","city"), ip_sep=",")
```

参数说明

参数名称	参数描述	参数类型	是否必须	参数默认值	参数取值范围
data	IP 值，支持多个 IP 通过分隔符分隔	string	是	-	-
keep	要留存的字段	string	否	("country","province","city")	-
ip_sep	匹配结果中的第几个表达式	string	否	-	-

示例

- 示例1

  原始日志：

  ```
  {"ip":"101.132.57.150"}
  ```

  加工规则：

  ```
  fields_set("result", geo_parse(v("ip")))
  ```

  加工结果：

  ```
  {"ip":"101.132.57.150","result":"{\"country\":\"中国\",\"province\":\"上海市\",\"city\":\"上海市\"}"}
  ```

- 示例2

  原始日志：

  ```
  {"ip":"101.132.57.150,101.14.57.157"}
  ```

  加工规则：

  ```
  fields_set("result", geo_parse(v("ip"),keep="province,city",ip_sep=","))
  ```

  加工结果：

  ```
  {"ip":"101.132.57.150,101.14.57.157", "result":"{\"101.14.57.157\":{\"province\":\"台湾省\",\"city\":\"NULL\"},\"101.132.57.150\":{\"province\":\"上海市\",\"city\":\"上海市\"}}"}
  ```

加工实战案例
案例总览

最近更新时间：2022-03-22 16:41:30

操作场景

您可参考本文提供的案例，对数据加工有个总体、感性的认识，同时，也可以复制这些案例中的函数，完成自己的数据加工。

注意事项

- v 函数的使用：**v("字段A")**，意思是 value of “字段A”，即字段 A 的值。有些函数的参数是**字段**，有些函数的参数是**字段值**，常见错误是：函数参数为字段值，却没有使用**v函数**取字段值，导致函数执行失败。
- 分发到多个日志主题，需要提前配好**目标日志主题**及其**目标名称**，目标名称用于**分发函数**。
- fields_set 函数的使用：通过 fields_set 函数设置字段值，保存数据加工函数处理后的内容。例如 fields_set("A+B",op_add(v("字段A") ,v("字段B")))，意思是将字段 A 的值和字段 B 的值相加，op_add 函数需要借助 fields_set 函数来完成结果的写入和保存。

操作场景

您可参考以下案例，完成自己的数据加工。

- 日志过滤和分发
- 单行文本日志结构化
- 数据脱敏
- 嵌套 JSON 的处理
- 多格式的日志结构化
- 利用分割符提取日志指定内容

日志过滤和分发

最近更新时间：2022-03-22 16:41:37

场景描述

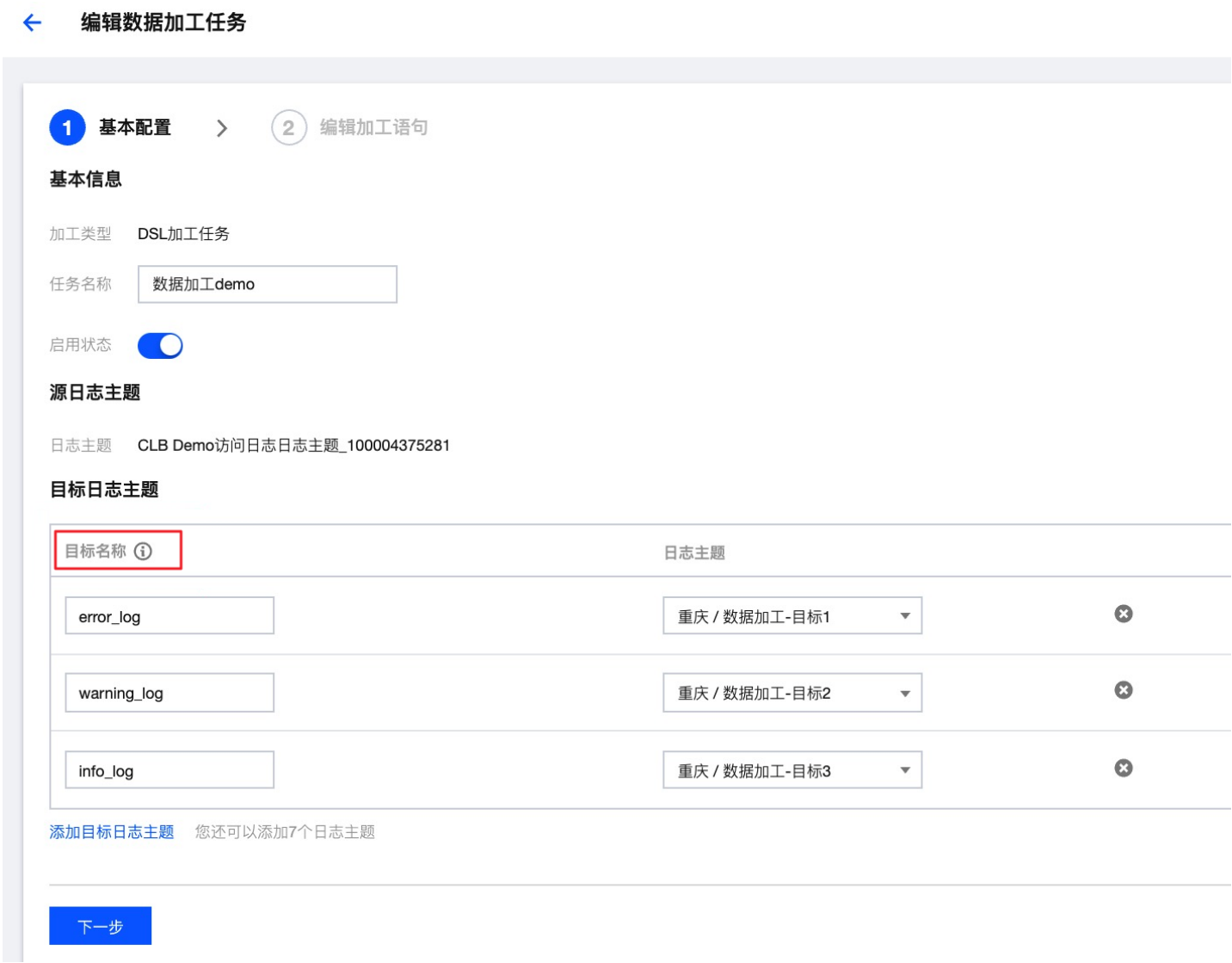
小王将日志采集到了日志服务（Cloud Log Service，CLS），日志通过双竖线 || 分割，有日志的时间、日志级别、日志内容、任务 ID、进程名称、主机 IP 等。现在小王想将日志结构化，便于后续索引、仪表盘展示。并按照 ERROR、WARNING、INFO 三个级别，把日志**分发**到三个不同的目标日志主题中，便于后续的分析。最后，当日志内容中有 “team B is working” 字样时，将该条**日志过滤（丢弃）**。

场景分析

梳理一下小王的加工需求，加工思路如下：

1. 日志中如有 team B is working 字符，将该条日志过滤（丢弃）；并把丢弃日志放在前面，可以减少后续的运算量。
2. 日志结构化：按照**双竖线**||分割符，对日志进行结构化。
3. 日志分发：按照按照 ERROR、WARNING、INFO 三个级别，把日志分发到三个不同的目标日志主题。

> **注意：**
> 分发到多个目标日志主题，需要在新建数据加工任务时，定义好日志主题的目标名称。该名称将用在 log_output("目标名称") 函数中。



原始日志

```
[
{
"message": "2021-12-09 11:34:28.279||team A is working||INFO||605c643e29e4||BIN--COMPILE||192.168.1.1"
},
{
"message": "2021-12-09 11:35:28.279||team A is working ||WARNING||615c643e22e4||BIN--Java||192.168.1.1"
},
{
"message": "2021-12-09 11:36:28.279||team A is working ||ERROR||635c643e22e4||BIN--Go||192.168.1.1"
},
{
"message": "2021-12-09 11:37:28.279||team B is working||WARNING||665c643e22e4||BIN--Python||192.168.1.1"
}
]
```

DSL 加工函数

```
log_drop(regex_match(v("message"),regex="team B is working",full=False))
ext_sepstr("message","time,log,loglevel,taskId,ProcessName,ip",sep="\|\|")
fields_drop("message")
t_switch(regex_match(v("loglevel"),regex="INFO",full=True),log_output("info_log"),regex_match(v("loglevel"),regex="WARNING",full=True),log_ou
tput("warning_log"),regex_match(v("loglevel"),regex="ERROR",full=True),log_output("error_log"))
```

DSL 加工函数详解

1. 当日志中含有 **team B is working** 关键字时，丢弃该条日志。因为第四条日志中包含有 **team B is working**，所以第四条日志会被丢弃。

```
log_drop(regex_match(v("message"),regex="team B is working",full=False))
```

2. 根据**双竖线||**分隔符来提取结构化数据。

```
ext_sepstr("message","time,log,loglevel,taskId,ProcessName,ip",sep="\|\|")
```

3. 丢弃 **message** 字段。

```
fields_drop("message")
```

4. 按照 **loglevel** 的字段值，**INFO/WARNING/ERROR**，将日志分发到不同的目标日志主题。

```
t_switch(regex_match(v("loglevel"),regex="INFO",full=True),log_output("info_log"),regex_match(v("loglevel"),regex="WARNING",full=True),log_
output("warning_log"),regex_match(v("loglevel"),regex="ERROR",full=True),log_output("error_log"))
```

加工结果

> ⚠ 注意：
> 必须要提前配置好目标日志主题和目标名称。

该日志分发到 **info_log**，即数据加工-目标3，可参看上图中的目标名称和日志主题的对应关系。

```
{"ProcessName":"BIN--COMPILE","ip":"192.168.1.1","log":"team A is working","loglevel":"INFO","taskId":"605c643e29e4","time":"2021-12-09 11:3
4:28.279"}
```

该日志分发到 **warning_log**，即数据加工-目标2。

{"ProcessName":"BIN--COMPILE","ip":"192.168.1.1","log":"team A is working","loglevel":"INFO","taskId":"605c643e29e4","time":"2021-12-09 11:34:28.279"}

该日志分发到 **error_log**，即数据加工-目标1。

{"ProcessName":"BIN--Go","ip":"192.168.1.1","log":"team A is working ","loglevel":"ERROR","taskId":"635c643e22e4","time":"2021-12-09 11:36:28.279"}

单行文本日志结构化

最近更新时间：2022-03-22 16:41:44

场景描述

小王将日志采集到日志服务（Cloud Log Service，CLS），但是没有固定的分割符号，是单行文本格式。现在小王想将日志结构化，从文本中**提取日志时间、日志级别、操作、URL 信息**，便于后续的检索分析。

场景分析

梳理一下小王的加工需求，加工思路如下：

1. {...}中的内容是**操作**的详情，可以通过正则提取。
2. 使用正则提取**日志时间、日志级别、URL**。

原始日志

```
{
"content": "[2021-11-24 11:11:08,232][328495eb-b562-478f-9d5d-3bf7e][INFO] curl -H 'Host: ' http://abc.com:8080/pc/api -d {\"version\": \"1.0\",
\"user\": \"CGW\",\"password\": \"123\",\"interface\": {\"Name\": \"ListDetail\",\"para\": {\"owner\": \"1253\",\"orderField\": \"createTime\"}}}"
}
```

DSL 加工函数

```
fields_set("Action",regex_select(v("content"),regex="\{[^\}]+\}",index=0,group=0))
fields_set("loglevel",regex_select(v("content"),regex="\[[A-Z]{4}\]",index=0,group=0))。
fields_set("logtime",regex_select(v("content"),regex="\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2},\d{3}",index=0,group=0))
fields_set("Url",regex_select(v("content"),regex="([a-z]{3}).([a-z]{3}):([0-9]{4})",index=0,group=0))
fields_drop("content")
```

DSL 加工函数详解

1. 新建一个字段 **Action**，使用正则{[^}]+}，匹配**{...}**。

```
fields_set("Action",regex_select(v("content"),regex="\{[^\}]+\}",index=0,group=0))
```

2. 新建一个字段 **loglevel**，使用正则[A-Z]{4}可以匹配 **INFO**。

```
fields_set("loglevel",regex_select(v("content"),regex="\[[A-Z]{4}\]",index=0,group=0))。
```

3. 新建一个字段 **logtime**，使用正则d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2},\d{3}匹配**2021-11-24 11:11:08**。

```
fields_set("logtime",regex_select(v("content"),regex="\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2},\d{3}",index=0,group=0))
```

4. 新建一个字段 **Url**，使用正则[a-z]{3}.[a-z]{3}匹配 **abc.com**，[0-9]{4}匹配**8080**。

```
fields_set("Url",regex_select(v("content"),regex="([a-z]{3}).([a-z]{3}):([0-9]{4})",index=0,group=0))
```

5. 丢弃 **content** 字段。

```
fields_drop("content")
```

加工结果

{"Action":"{\"version\": \"1.0\",\"user\": \"CGW\",\"password\": \"123\",\"interface\": {\"Name\": \"ListDetail\",\"para\": {\"owner\": \"1253\",\"orderField\": \"createTime\"}","Url":"abc.com:8080","loglevel":"[INFO]","logtime":"2021-11-24 11:11:08,232"}

数据脱敏

最近更新时间：2022-03-22 16:41:54

场景描述

小王将日志采集到日志服务（Cloud Log Service，CLS），日志数据中含有用户 ID（**dev@12345**）、登录的 IP 地址（**11.111.137.225**）、手机号码（**13912345678**），小王想将这些敏感信息脱敏。

场景分析

该日志本身是一个结构化的日志，因此可以直接对字段进行脱敏处理。

原始日志

```
{
"Id": "dev@12345",
"Ip": "11.111.137.225",
"phonenumber": "13912345678"
}
```

DSL 加工函数

```
fields_set("Id",regex_replace(v("Id"),regex="\d{3}", replace="***",count=0))
fields_set("Id",regex_replace(v("Id"),regex="\S{2}", replace="**",count=1))
fields_set("phonenumber",regex_replace(v("phonenumber"),regex="(\d{0,3})\d{4}(\d{4})", replace="$1****$2"))
fields_set("Ip",regex_replace(v("Ip"),regex="(\d+\.)\d+(\.\d+\.\d+)", replace="$1***$2",count=0))
```

DSL 加工函数详解

1. 对 **Id** 字段进行脱敏处理，结果为dev@***45。

```
fields_set("Id",regex_replace(v("Id"),regex="\d{3}", replace="***",count=0))
```

2. 对 **Id** 字段进行二次脱敏处理，结果为**v@***45。

```
fields_set("Id",regex_replace(v("Id"),regex="\S{2}", replace="**",count=1))
```

3. 对 **phonenumber** 字段进行脱敏处理，将中间的4位数替换为****，结果为139****5678。

```
fields_set("phonenumber",regex_replace(v("phonenumber"),regex="(\d{0,3})\d{4}(\d{4})", replace="$1****$2"))
```

4. 对 **IP** 字段进行脱敏处理，将第二段替换为***，结果为11.***137.225。

```
fields_set("Ip",regex_replace(v("Ip"),regex="(\d+\.)\d+(\.\d+\.\d+)", replace="$1***$2",count=0))
```

加工结果

```
{"Id":"**v@***45","Ip":"11.***.137.225","phonenumber":"139****5678"}
```

嵌套 JSON 的处理

最近更新时间：2022-03-22 16:42:01

场景描述

小王将日志以 JSON 格式采集到日志服务（Cloud Log Service，CLS）。JSON 是多层嵌套，小王想提取 **user** 和 **App** 字段。其中 **user** 是二级嵌套字段。

原始日志

```
[
{
"content": {
"App": "App-1",
"start_time": "2021-10-14T02:15:08.221",
"resonsebody": {
"method": "GET",
"user": "Tom"
},
"response_code_details": "3000",
"bytes_sent": 69
}
},
{
"content": {
"App": "App-2",
"start_time": "2222-10-14T02:15:08.221",
"resonsebody": {
"method": "POST",
"user": "Jerry"
},
"response_code_details": "2222",
"bytes_sent": 1
}
}
]
```

DSL 加工函数

- **方法一：不展开所有的 KV，使用 jmes 公式直接提取字段。**

```
ext_json_jmes("content", jmes="resonsebody.user", output="user")
ext_json_jmes("content", jmes="App", output="App")
```

- **方法二：平铺所有的 KV，丢弃不需要的字段。**

```
ext_json("content")
fields_drop("content")
fields_drop("bytes_sent","method","response_code_details","start_time")
```

DSL 加工函数详解

- **方法一：**
  i. **使用 jmes 公式 resonsebody.user，直接指定二级嵌套的 user 字段。**

```
ext_json_jmes("content", jmes="resonsebody.user", output="user")
```

ii. 使用 jmes 公式 App，直接指定 App 字段。

```
ext_json_jmes("content", jmes="App", output="App")
```

- 方法二：
  i. 使用 ext_json 函数从 JSON 数据中提取结构化数据，默认会平铺所有的字段。

```
ext_json("content")
```

  ii. 丢弃 content 字段。

```
fields_drop("content")
```

  iii. 丢弃不需要的字段 bytes_sent,method,response_code_details,start_time

```
fields_drop("bytes_sent","method","response_code_details","start_time")
```

加工结果

```
[{"App":"App-1","user":"Tom"},
{"App":"App-2","user":"Jerry"}]
```

多格式的日志结构化

最近更新时间：2022-03-22 16:42:08

场景描述

小王把**用户操作和结果**的日志，以单行文本的格式采集到日志服务（Cloud Log Service，CLS），日志的格式和内容**不完全相同**。小王想编写一套语句，对不同格式的日志进行结构化。
经过梳理查看，日志基本分为三种格式：第一种包含有 **uin、requestid、action、Reqbody** 四个字段，第二种包含 **uin、requestid、action** 三个字段，第三种包含 **requestid、action、TaskId** 三个字段。

场景分析

梳理一下小王的加工需求，加工思路如下：

1. 由于三种格式的日志都包含 **requestid、action** 字段，因此使用正则提取 **requestid** 和 **action** 字段。
2. 对 **uin、reqbody、TaskId** 字段进行特殊处理，先判断这个字段是否存在，如果存在，再进行提取。

原始日志

```
[
{
"__CONTENT__": "2021-11-29 15:51:33.201 INFO request 7143a51d-caa4-4a6d-bbf3-771b4ac9e135 action: Describe uin: 15432829 reqbody {\"Key\": \"config\",\"Values\": \"appisrunnning\",\"Action\": \"Describe\",\"RequestId\": \"7143a51d-caa4-4a6d-bbf3-771b4ac9e135\",\"AppId\": 1302953499,\"Uin\": \"100015432829\"}"
},
{
"__CONTENT__": "2021-11-2915: 51: 33.272 ERROR request 2ade9fc4-2db2-49d8-b3e0-a6ea78ce8d96 has error action DataETL uin 15432829"
},
{
"__CONTENT__": "2021-11-2915: 51: 33.200 INFO request 6059b946-25b3-4164-ae93-9178c9e73d75 action: UploadData hUWZSs69yGc5HxgQ TaskId 51d-caa-a6d-bf3-7ac9e"
}
]
```

DSL 加工函数

```
fields_set("requestid",regex_select(v("__CONTENT__"),regex="request [A-Za-z0-9]+-[A-Za-z0-9]+-[A-Za-z0-9]+-[A-Za-z0-9]+-[A-Za-z0-9]+",index=0,group=0))
fields_set("action",regex_select(v("__CONTENT__"),regex="action: \S+|action \S+",index=0,group=0))
t_if(regex_match(v("__CONTENT__"),regex="uin", full=False),fields_set("uin",regex_select(v("__CONTENT__"),regex="uin: \d+|uin \d+",index=0,group=0)))
t_if(regex_match(v("__CONTENT__"),regex="TaskId", full=False),fields_set("TaskId",regex_select(v("__CONTENT__"),regex="TaskId [A-Za-z0-9]+-[A-Za-z0-9]+-[A-Za-z0-9]+-[A-Za-z0-9]+-[A-Za-z0-9]+",index=0,group=0)))
t_if(regex_match(v("__CONTENT__"),regex="reqbody", full=False),fields_set("requestbody",regex_select(v("__CONTENT__"),regex="reqbody \{[^\}]+\}")))
t_if(has_field("requestbody"),fields_set("requestbody",str_replace(v("requestbody"),old="reqbody",new="")))
fields_drop("__CONTENT__")
fields_set("requestid",str_replace(v("requestid"),old="request",new=""))
t_if(has_field("action"),fields_set("action",str_replace(v("action"),old="action:|action",new="")))
t_if(has_field("uin"),fields_set("uin",str_replace(v("uin"),old="uin:|uin",new="")))
t_if(has_field("TaskId"),fields_set("TaskId",str_replace(v("TaskId"),old="TaskId",new="")))
```

DSL 加工函数详解

1. 新建一个字段 **requestid**，使用正则公式匹配 “**request 7143a51d-caa4-4a6d-bbf3-771b4ac9e135**”。

```
fields_set("requestid",regex_select(v("__CONTENT__"),regex="request [A-Za-z0-9]+-[A-Za-z0-9]+-[A-Za-z0-9]+-[A-Za-z0-9]+-[A-Za-z0-9]+",index=0,group=0))
```

2. 新建一个字段 **action**，使用正则公式匹配 **“action: UploadData”** 或者 **“action DataETL”**。因为原始日志中两种格式都有。

```
fields_set("action",regex_select(v("__CONTENT__"),regex="action: \S+|action \S+",index=0,group=0))
```

- 如果 **__CONTENT__** 字段中有 **uin** 这个字符，新建 **uin** 字段，使用正则**"uin: \d+|uin \d+"**来匹配**uin: 15432829**或者**uin 15432829**。

```
t_if(regex_match(v("__CONTENT__"),regex="uin", full=False),fields_set("uin",regex_select(v("__CONTENT__"),regex="uin: \d+|uin \d+",index=0,group=0)))
```

- 如果有 **TaskId** 这个字符，新建 **TaskId** 字段，使用正则来匹配 **“TaskId 51d-caa-a6d-bf3-7ac9e”**。

```
t_if(regex_match(v("__CONTENT__"),regex="TaskId", full=False),fields_set("TaskId",regex_select(v("__CONTENT__"),regex="TaskId [A-Za-z0-9]+-[A-Za-z0-9]+-[A-Za-z0-9]+-[A-Za-z0-9]+",index=0,group=0)))
```

- 如果有 **reqbody** 这个字符，新建 **requestbody** 字段，使用正则来匹配 **reqbody{...}**。

```
t_if(regex_match(v("__CONTENT__"),regex="reqbody", full=False),fields_set("requestbody",regex_select(v("__CONTENT__"),regex="reqbody \{[^\}]+\}")))
```

3. 丢弃**__CONTENT__**字段。

```
fields_drop("__CONTENT__")
```

以上我们已经提取出了我们需要的字段，但是由于在正则匹配过程中，产生了多余的字符 action、uin、requestbody、requestid、TaskId。所以需要使用 **str_replace()** 函数将多余的字符去掉，并且使用 **fields_set()** 函数对字段值进行重置。

1. 如果有 **requestbody** 字段，将字段值中多余的 **reqbody** 字符去掉。

```
t_if(has_field("requestbody"),fields_set("requestbody",str_replace(v("requestbody"),old="reqbody",new="")))
```

2. 将字段值v(**"requestid"**)中多余的 **requestid** 去掉。因为每条日志中都有 **requestid**，所以没有判断这个字段是否存在。

```
fields_set("requestid",str_replace(v("requestid"),old="request",new=""))
```

3. 如果有 **action** 字段，那么字段值中多余的 **action:** 或 **action** 字符去掉。

```
t_if(has_field("action"),fields_set("action",str_replace(v("action"),old="action:|action",new="")))
```

4. 如果有 **uin** 字段，那么字段值中多余的 **uin:** 或者 **uin** 去掉。

```
t_if(has_field("uin"),fields_set("uin",str_replace(v("uin"),old="uin:|uin",new="")))
```

5. 如果有 **TaskId** 字段，那么字段值中多余的 **TaskId** 字符去掉。

```
t_if(has_field("tTaskId"),fields_set("TaskId",str_replace(v("TaskId"),old="TaskId",new="")))
```

加工结果

```
[
{"action":" Describe","requestid":" 7143a51d-caa4-4a6d-bbf3-771b4ac9e135","requestbody":" {\"Key\": \"config\",\"Values\": \"appisrunnning\",\"Action\": \"Describe\",\"RequestId\": \"7143a51d-caa4-4a6d-bbf3-771b4ac9e135\",\"AppId\": 1302953499,\"Uin\": \"100015432829\"}","uin":" 154
```

32829"},
{"action":" DataETL","requestid":" 2ade9fc4-2db2-49d8-b3e0-a6ea78ce8d96","uin":" 15432829"},
{"action":" UploadData","requestid":" 6059b946-25b3-4164-ae93-9178c9e73d75","TaskId":" 51d-caa-a6d-bf3-7ac9e"}
]

利用分割符提取日志指定内容

最近更新时间：2022-03-22 16:42:17

场景描述

小王将 Flink 任务运行的日志，以单行文本采集到日志服务（Cloud Log Service，CLS）。日志内容里面包含了逗号"," ,冒号": "，这些分割符将日志分割成了几小段。其中有一段是转义 JSON，它里面是 Flink 任务执行的详情，小王想将任务详情提取出来，然后对其进行结构化。

场景分析

梳理一下小王的加工需求，加工思路如下：

1. 将转义 JSON 提取出来。
2. 从 JSON 中提取结构化数据。

原始日志

```
{
"regex": "2021-12-02 14:33:35.022 [1] INFO org.apache.Load - Response:status: 200, resp msg: OK, resp content: { \"TxnId\": 58322, \"Label\": \"
flink_connector_20211202_1de749d8c80015a8\", \"Status\": \"Success\", \"Message\": \"OK\", \"TotalRows\": 1, \"LoadedRows\": 1, \"FilteredRows\"
: 0, \"CommitAndPublishTimeMs\": 16}"
}
```

DSL 加工函数

```
ext_sepstr("regex", "f1, f2, f3", sep=",")
fields_drop("regex")
fields_drop("f1")
fields_drop("f2")
ext_sepstr("f3", "f1,resp_content", sep=":")
fields_drop("f1")
fields_drop("f3")
ext_json("resp_content", prefix="")
fields_drop("resp_content")
```

DSL 加工函数详解

1. 使用逗号将该条日志截成3段，第三段f3是 resp content:{JSON}。

```
ext_sepstr("regex", "f1, f2, f3", sep=",")
```

2. 将不需要的字段丢弃。

```
fields_drop("regex")
fields_drop("f1")
fields_drop("f2")
```

3. 使用冒号将f3字段截成两段。

```
ext_sepstr("f3", "f1,resp_content", sep=":")
```

4. 丢弃无用的字段。

```
fields_drop("f1")
fields_drop("f3")
```

5. 使用 ext_json 函数，从 resp_content 字段中，提取结构化数据。

```
ext_json("resp_content", prefix="")
```

6. 丢弃 **resp_content** 字段。

```
fields_drop("resp_content")
```

加工结果

```
{"CommitAndPublishTimeMs":"16","FilteredRows":"0","Label":"flink_connector_20211202_1de749d8c80015a8","LoadedRows":"1","Message":"OK"
,"Status":"Success","TotalRows":"1","TxnId":"58322"}
```

投递与消费
投递权限配置
投递权限查看及配置

最近更新时间：2022-03-24 14:31:00

操作场景

当您创建投递到 COS/Ckafka 的任务时，需对应实际使用的方式进行授权：

- **通过日志服务控制台**：根据控制台的权限流程指引进行配置。
- **通过 API 调用**：需手动创建投递 COS/Ckafka 的权限。

> ⚠ 注意
> 若您在 API 调用前未配置权限，则会导致投递数据失败，请参考本文检查并配置对应权限。

本文介绍如何检查是否已具备投递 COS/CKafka 权限，及其创建步骤。若您在检查后无相关权限，请创建后再进行权限配置操作。

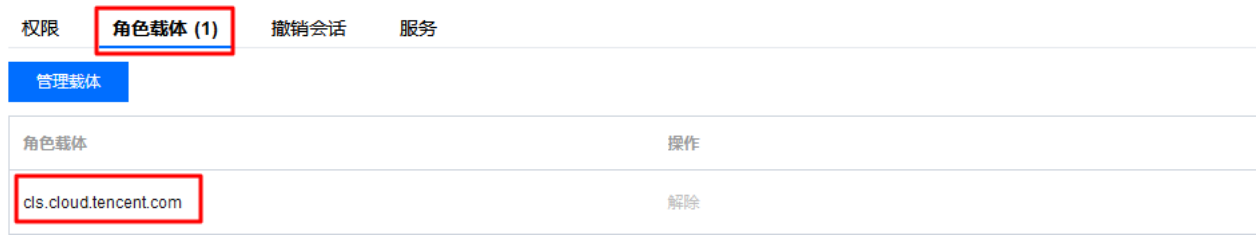
操作步骤

检查是否已有投递 COS/CKafka 权限

1. 登录访问管理控制台，选择左侧导航栏中的 **角色**。
2. 在“角色”页面中，查看是否具备 CLS_QcsRole 角色。可通过列表右上角的搜索框进行搜索，如下图所示：

3. 单击角色名称，进入角色详情页。
   - 选择**权限**，查看是否已具备 QcloudCOSAccessForCLSRole 及 QcloudCKAFKAAccessForCLSRole 权限。如下图所示：

   - 选择**角色载体**，查看角色载体是否为 cls.cloud.tencent.com。如下图所示：



上述检查步骤确认无误后，则您已具备对应权限，可创建投递到 COS/Ckafka 的任务。若不具备角色及相关权限，则请参考下文进行创建。

创建投递 COS/Ckafka 权限

您可通过以下方式创建投递 COS/Ckafka 权限：

通过控制台自动创建

您在首次通过控制台创建投递到 COS/Ckafka 的任务时，请根据页面指引创建角色和策略：

1. 在弹出的“该功能需创建服务角色”窗口中，单击**前往访问管理**。如下图所示：

2. 在“角色管理”页面中，单击**同意授权**。如下图所示：

使用访问管理手动创建

1. 登录访问管理控制台，选择左侧导航栏中的 **角色**。
2. 在“角色”页面中，单击**新建角色**。
3. 在弹出的“选择角色载体”窗口中，选择**腾讯云产品服务**。
4. 在“新建自定义角色”中，进行如下配置：
   i. 在“输入角色载体信息”中，勾选“日志服务 (cls)”，并单击**下一步**。
   ii. 在“配置角色策略”中，使用 clsrole 进行搜索，并在结果中勾选 QcloudCKAFKAAccessForCLSRole 及 QcloudCOSAccessForCLSRole 策略后，单击**下一步**。如下图所示：

iii. 在“审阅”中，输入角色名 CLS_QcsRole，并单击**完成**。如下图所示：

至此您已完成投递 COS/Ckafka 权限创建，若您使用主账号，则可直接进行投递操作。若您需使用子账号或协作者账号，则请使用主账号参考 权限说明 授予相关授权策略后再进行投递。

权限说明

最近更新时间：2022-03-24 14:31:05

日志服务（Cloud Log Service，CLS）将日志投递至对象存储（Cloud Object Storage，COS）或 Ckafka 的过程中，需要用户确认授予日志服务具有写 COS 或 Ckafka 的权限。用户需要为日志服务的唯一服务角色 CLS_QcsRole 授权 QcloudCOSAccessForCLSRole 或 QcloudCKAFKAAccessForCLSRole 策略权限。

若您需授予协作者或子账号配置投递任务权限，则请参考本文进行操作。

通过控制台为协作者或子账号配置投递任务

协作者或子账号在日志服务控制台上进行投递任务配置时，需要主账号授权必要的权限，否则无法顺利进行投递配置。协作者或子账号在配置投递任务时，会依赖以下权限：

权限名称	描述说明	应用场景
CLS 预设策略：QcloudCLSFullAccess	协作者或子账号可以操作日志服务的权限	协作者或子账号需要此权限才能操作日志服务进行投递任务的配置
CAM 预设策略：QcloudCamSubaccountsAuthorizeRoleFullAccess	协作者或子账号可以授权服务角色的权限	日志投递至 COS 或 Ckafka 时，需要协作者或子账号确认角色授权，使 CLS 具有写 COS 或写 Ckafka 的权限，即为服务角色 CLS_QcsRole 授权 QcloudCOSAccessForCLSRole 或 QcloudCKAFKAAccessForCLSRole 策略权限
COS 接口权限：GetService（或预设策略：QcloudCOSReadOnlyAccess）	协作者或子账号可以获取 COS 存储桶列表的权限	控制台配置投递至 COS 任务的过程中，需要拉取 COS 存储桶列表，然后选择投递的目标存储桶
CKafka 接口权限：ListInstance，ListTopic（或预设策略：QcloudCkafkaReadOnlyAccess）	协作者或子账号可以获取 CKafka 资源列表的权限	控制台配置投递至 Ckafka 任务的过程中，需要拉取 CKafka 的资源列表，然后选择投递的目标 Ckafka 实例的主题

主账号授权步骤

1. 使用主账号登录访问管理控制台，选择左侧导航栏中的**用户** > **用户列表**。
2. 在“用户列表”页面中，单击需操作的用户名称，进入用户详情页。
3. 在用户详情页面中，选择**关联策略**。
4. 在“添加策略”页面的“设置用户权限”中，选择**从策略列表中选取策略关联**。如下图所示：

5. 为协作者或子账号关联预设策略：QcloudCLSFullAccess、QcloudCamSubaccountsAuthorizeRoleFullAccess、QcloudCOSReadOnlyAccess、QcloudCkafkaReadOnlyAccess。

> ⚠ 注意
> 主账号必须完成授权操作，否则协作者或子账号无法正常进行投递配置。

6. 单击**下一步**。
7. 在“审阅”中确认配置，并单击**确定**即可完成授权。

投递至 COS
投递简介

最近更新时间：2022-01-19 15:32:03

日志投递简介

对象存储配置与日志投递

日志服务（Cloud Log Service，CLS）投递功能打通产品生态下游链路，将日志数据投递到对象存储（Cloud Object Storage，COS）等其他腾讯云产品中，进一步满足日志场景的述求，挖掘日志数据价值。日志投递为异步过程，凡是收集到日志服务的数据，都能通过配置将数据投递到其他云产品中进行存储和分析，您只需要在日志服务控制台进行简单的配置即可完成数据投递。

日志投递到 COS

日志服务可以将日志主题中的数据投递到 COS，以满足其他应用场景需求，例如：

- 通过设置 COS 的生命周期，对日志数据进行长时间的归档存储。
- 通过离线计算或其他计算程序处理 COS 上的日志数据。

投递数据格式	描述说明
按分隔符格式投递	日志数据按照分隔符格式投递到 COS
按 JSON 格式投递	日志数据按照 JSON 格式投递到 COS
按原文格式投递	日志数据按照原文格式投递到 COS

⚠ 注意：

- 数据投递到 COS 之后，COS 侧将产生相应的存储费用，计费详情请参见 COS 计费概述。
- 若需要在投递 COS 前对日志进行清洗加工过滤，可参考使用 通过云函数转储至 COS 操作。

分隔符格式投递

最近更新时间：2022-06-24 12:11:04

概述

您可以通过 日志服务控制台，将数据按照分隔符格式投递到对象存储（Cloud Object Storage，COS），下面将为您详细介绍如何创建分隔符格式日志投递任务。

前提条件

1. 开通日志服务，创建日志集与日志主题，并成功采集到日志数据。
2. 开通腾讯云对象存储服务，并且在待投递日志主题的地域已创建存储桶，详细配置请参见 创建存储桶 文档。
3. 确保当前操作账号拥有配置投递的权限，子账号/协作者投递权限问题请参见 子账号配置投递 文档。

操作步骤

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**。
3. 单击需要投递的日志主题ID/名称，进入日志主题管理页面。
4. 单击**投递至 COS** 页签，进入投递至 COS 配置页面，依次填写配置信息。

配置项说明如下：

配置项	解释说明	规则	是否必填
投递任务名称	配置投递任务的名称。	字母、数字、_和-	必填
COS存储桶	与当前日志主题同地域的存储桶作为投递目标存储桶 。	列表选择	必填
目录前缀	日志服务支持自定义目录前缀，日志文件会投递至对象存储 Bucket 的该目录下。目前默认是直接放在存储桶下，文件路径为{COS 存储桶}{目录前缀}{分区格式}_{random}_{index}.{type}，其中{random}_{index}是一个随机数。	非/开头	可选
分区格式	将投递任务创建时间按照 strftime 的语法自动生成目录 ，其中斜线/表示一级 COS 目录。	strftime 格式	必填
投递文件大小	指定在该投递时间间隔中未压缩的投递文件上限，意味着在该时间间隔中，日志文件最大将为您设置的值，超过该上限，将被分成多个日志文件，上限支持100MB - 256MB。	100 - 256，单位：MB	必填
投递间隔时间	指定投递的时间间隔，支持300s - 900s。假设您设置投递时间间隔为5分钟，那么意味着您的日志数据将每5分钟产生一个日志文件，每隔一段时间（半小时内），多个日志文件会一起投递至您的存储桶。	300 - 900，单位：s	必填

上表中的分区格式请按照 strftime 格式 要求填写，不同的分区格式会影响投递到对象存储的文件路径。以下举例说明分区格式的用法，例如投递至 bucket_test 存储桶，目录前缀为logset/，投递时间 2018/7/31 17:14，则对应的投递文件路径如下：

存储桶名称	目录前缀	分区格式	COS 文件路径
bucket_test	logset/	%Y/%m/%d	bucket_test:logset/2018/7/31_{random}_{index}
bucket_test	logset/	%Y%m%d/%H	bucket_test:logset/20180731/14_{random}_{index}
bucket_test	logset/	%Y%m%d/log	bucket_test:logset/20180731/log_{random}_{index}

5. 单击**下一步**，进入高级配置，选择投递格式为 CSV，依次填写相关配置参数。

配置项说明如下：

配置项	解释说明	规则	是否必填
键值名称（key）	指定写入 CSV 文件的键值（key）字段（填写的 key 必须是日志结构化后的 key 名称或保留字段，否则将视为无效 key）。	字母、数字、_和-	必填
分隔符	CSV 文件中各字段间的分隔符。	列表选择	必填
转义符	若正常字段内出现了分隔符的字符，需用转义符包裹该字符，防止读取数据时被错误识别。	列表选择	必填
无效字段	若配置的键值字段（key）不存在时，则会用无效字段进行填充 。	列表选择	必填
首行 Key	在 CSV 文件的首行增加字段名的描述，即将键值（key）写入 CSV 文件的首行，默认不写入。	开/关	必填
压缩投递	是否对日志文件进行压缩后投递，在投递时的未压缩文件大小上限为10GB 。目前支持的压缩方式有 gzip、lzop 和 snappy。	开/关	必填

JSON 格式投递

最近更新时间：2022-01-19 15:49:33

概述

您可以通过 日志服务控制台，将数据按照 JSON 格式投递到对象存储（Cloud Object Storage，COS），下面将为您详细介绍如何创建 JSON 格式日志投递任务。

前提条件

1. 开通日志服务，创建日志集与日志主题，并成功采集到日志数据。
2. 开通腾讯云对象存储服务，并且在待投递日志主题的地域已创建存储桶，详细配置请参见 创建存储桶 文档。
3. 确保当前操作账号拥有配置投递的权限，子账号/协作者投递权限问题请参见 子账号配置投递 文档。

操作步骤

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**。
3. 单击需要投递的日志主题ID/名称，进入日志主题管理页面。

4. 单击**投递至 COS** 页签，进入投递至 COS 配置页面，依次填写配置信息。

配置项说明如下：

配置项	解释说明	规则	是否必填
投递任务名称	配置投递任务的名称。	字母、数字、_和-	必填
COS 存储桶	与当前日志主题同地域的存储桶作为投递目标存储桶。	列表选择	必填
目录前缀	日志服务支持自定义目录前缀，日志文件会投递至对象存储 Bucket 的该目录下。目前默认是直接放在存储桶下，文件路径为 {COS 存储桶}{目录前缀}{分区格式}_{random}_{index}.{type}，其中 {random}_{index}是一个随机数。	非/开头	可选
分区格式	将投递任务创建时间按照 strftime 的语法自动生成目录，其中斜线/表示一级 COS 目录。	strftime 格式	必填
投递文件大小	指定在该投递时间间隔中未压缩的投递文件上限，意味着在该时间间隔中，日志文件最大将为您设置的值，超过该上限，将被分成多个日志文件，上限支持100MB - 256MB。	100 - 256，单位：MB	必填
投递间隔时间	指定投递的时间间隔，支持300s - 900s。假设您设置投递时间间隔为5分钟，那么意味着您的日志数据将每5分钟产生一个日志文件，每隔一段时间（半小时内），多个日志文件会一起投递至您的存储桶。	300 - 900，单位：s	必填

上表中的分区格式请按照 strftime 格式 要求填写，不同的分区格式会影响投递到对象存储的文件路径。 以下举例说明分区格式的用法，例如投递至 bucket_test 存储桶，目录前缀为logset/，投递时间 2018/7/31 17:14，则对应的投递文件路径如下：

存储桶名称	目录前缀	分区格式	COS 文件路径
bucket_test	logset/	%Y/%m/%d	bucket_test:logset/2018/7/31_{random}_{index}
bucket_test	logset/	%Y%m%d/%H	bucket_test:logset/20180731/14_{random}_{index}
bucket_test	logset/	%Y%m%d/log	bucket_test:logset/20180731/log_{random}_{index}

5. 单击**下一步**，进入高级配置，选择投递格式为 json，依次填写相关配置参数。

配置项说明如下：

配置项	解释说明	规则	是否必填
压缩投递	是否对日志文件进行压缩后投递，在投递时的未压缩文件大小上限为10GB 。目前支持的压缩方式有 gzip 和 lzop，snappy。	开/关	必填

原文格式投递

最近更新时间：2022-01-19 15:34:22

概述

您可以通过 日志服务（Cloud Log Service，CLS）控制台，将数据按照日志原文格式投递到对象存储（Cloud Object Storage，COS），下面将为您详细介绍如何创建原文格式日志投递任务。

日志服务投递 COS，目前支持的基本格式有：CSV 格式、JSON 格式。而原文格式投递是在基本格式投递基础上，通过配置特殊参数从而达到还原日志原文的效果。不同的日志原文格式是否支持原文投递，请见下表：

日志原文格式	是否支持原文投递
单行全文	支持，参考 单行全文原文投递。
多行全文	支持，参考 多行全文原文投递。
JSON 格式	不支持。
分隔符（CSV）格式	不一定，参考 分隔符格式原文投递。
完全正则	不支持。

操作步骤

单行（或多行）全文原文投递

单行全文或多行全文可在 CSV 格式投递 的基础上，通过高级配置的特殊参数达到按原文投递的效果。

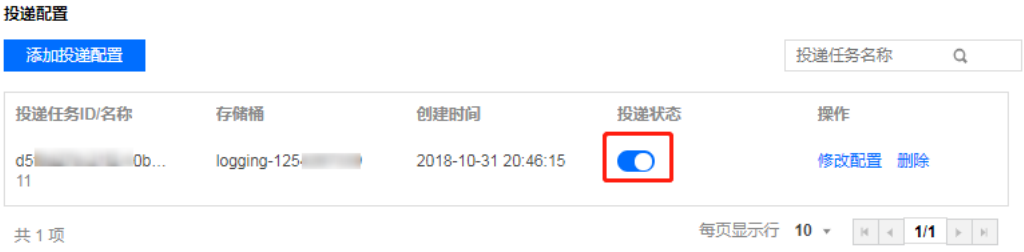
1. 按照 CSV 格式投递 的指引说明完成第一步“基本配置”。
2. 在高级配置中，投递格式选择 CSV，并在键值名称（key）填写 __CONTENT__，分隔符选择空格，转义符选择空格，无效字段填充选空，关闭“首行 Key”，如下图所示：

配置项详细说明如下：

配置项	填写	解释说明

配置项	填写	解释说明
键值名称（key）	__CONTENT__	单行或多行全文系统会默认 __CONTENT__ 作为键名称（key），日志原文作为值（value），原文投递时键名称填入 __CONTENT__。
分隔符	空格	对于单行或多行全文而言，分隔符选择空格。
转义符	空	为防止因转义符改变原文内容， 转义符选择空。
无效字段	空	无效字段选择空。
首行Key	关	原文投递无需在 CSV 文件的首行增加字段名的描述。

3. 单击**确定**，即可看到投递状态已开启。



分隔符格式原文投递

> 注意：
> CSV 格式投递 仅支持有限的分隔符（空格、制表符、逗号、分号、竖线），所以当且仅当日志原文中的分隔符与 CSV 格式投递所支持的分隔符一致时，才可以按原文投递，否则不可以投递原文。

1. 按照 CSV 格式投递 的指引说明完成第一步“基本配置”。
2. 在高级配置中，投递格式选择 CSV，按如下参数说明配置进行。

配置项	填写内容	解释说明
键值名称（key）	键（key）名称	按顺序填写原文中每组键值对所对应的键（key）名称。
分隔符	列表选择	选择原文对应的分隔符，若无相同的分隔符则不能按原文投递。
转义符	空	为防止因转义符改变原文内容， 转义符选择空即可。
无效字段	空	无效字段选择空。
首行 Key	关	原文投递无需在 CSV 文件的首行增加字段名的描述。

例如，原始日志为：

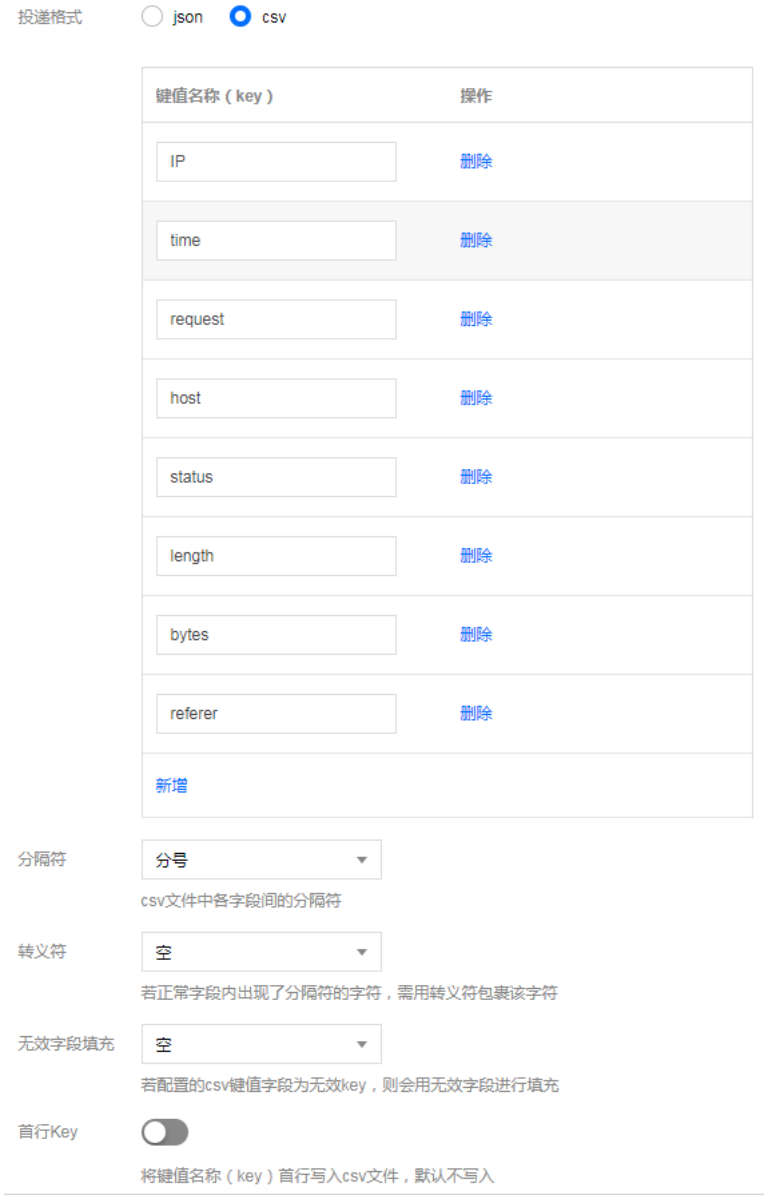
```
10.20.20.10;[Tue Jan 22 14:49:45 CST 2019 +0800];GET /online/sampleHTTP/1.1;127.0.0.1;200;647;35;http://127.0.0.1/
```

定义分隔符为分号(；)，并为每个字段定义如下键值（key）名称，如图所示：

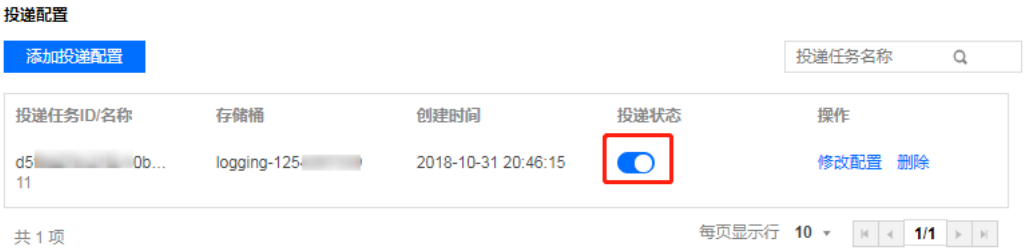
抽取结果

key	value
ip	10.20.20.10
time	[Tue Jan 22 14:49:45 CST 2019 +0800]
request	GET /online/sample HTTP/1.1
host	127.0.0.1
status	200
length	647
bytes	35
referer	http://127.0.0.1/

此时若需按原文投递，则需在投递高级配置时选择 CSV 格式的分隔符为分号(；)，完整配置填写如图所示：



3. 单击**确定**，即可看到投递状态已开启。



投递任务管理

最近更新时间：2022-01-19 15:52:06

概述

日志服务（Cloud Log Service，CLS）提供了投递任务管理的功能，您可以查看每个时间点的投递任务，可根据时间、日志集、日志主题、投递任务名称，查看相关的投递任务。

操作步骤

查看投递任务

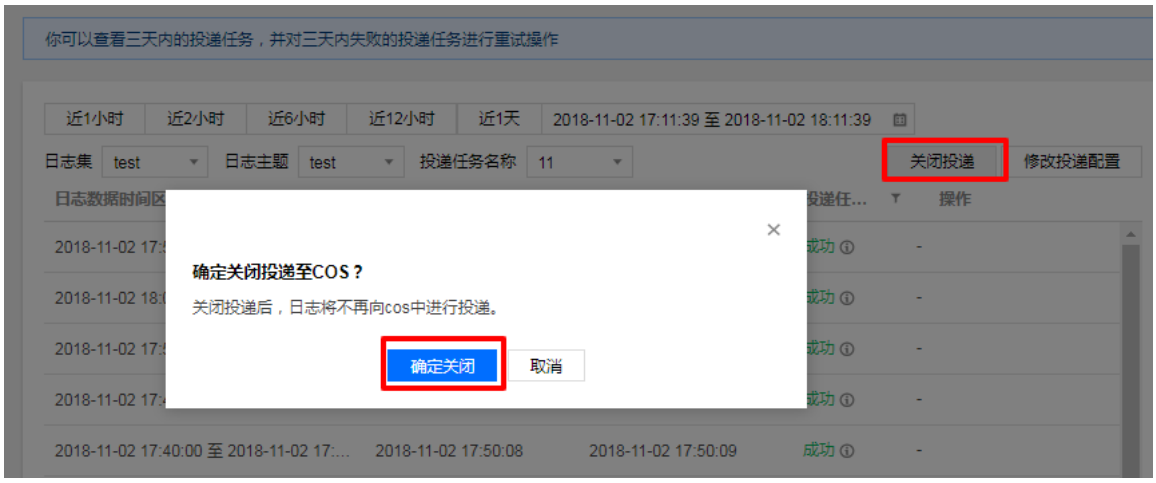
1. 登录 日志服务控制台。
2. 在左侧导航栏中，选择**投递任务管理**，进入投递任务管理页面。
3. 在投递任务管理界面中，选择投递任务的地域、日志集、日志主题、投递任务名称，即可查看投递任务详情。

修改投递配置

1. 登录 日志服务控制台。
2. 在左侧导航栏中，选择**投递任务管理**，进入投递任务管理页面。
3. 在投递任务管理界面中，选择投递任务的地域、日志集、日志主题、投递任务名称，找到对应的投递任务，然后单击右侧的**修改投递配置**，即可修改投递配置。

关闭投递任务

1. 登录 日志服务控制台.
2. 在左侧导航栏中，选择**投递任务管理**，进入投递任务管理页面。
3. 在投递任务管理界面中，选择投递任务的地域、日志集、日志主题、投递任务名称，找到对应的投递任务，单击右侧的关闭投递按钮，在弹窗中单击**确定关闭**，即可关闭日志投递。



投递至 Ckafka
创建投递任务

最近更新时间：2022-06-22 10:34:10

简介

您可以将日志主题的数据投递到腾讯云 Ckafka，然后用于您的实时流计算、或者入库等场景。如果您没有购买腾讯云 Ckafka 实例，可以考虑使用日志服务（Cloud Log Service，CLS）自带的 Kafka 协议消费功能。

前提条件

- 已开通腾讯云消息队列（CKafka）。
- 确保当前操作账号拥有开通投递到 Ckafka 的权限。如果您是子账号，一般需要主账号进行相关授权，方可使用。权限问题请参见 自定义权限策略示例。

操作步骤

1. 在日志主题同地域下，创建一个 Ckafka 实例。详情请参见 创建实例。
2. 在日志主题同地域下，根据如下配置参数，创建一个 Topic。详情请参见 创建 Topic。

- **预设ACL策略**：关闭预设 ACL 策略。
- **展示高级配置**：
  - **CleanUp.policy**：选择 **delete**。该参数需设置为 delete，否则会投递失败。
  - **max.message.bytes**：设置为 ≥ 8MB。该参数若小于8MB，会因 CLS 侧的单条 message 过大，无法写入 Ckafka Topic，导致投递失败。

3. 前往 日志服务控制台，并按需选择不同的操作，进入投递任务管理页面或者日志主题管理页面。
   - 在左侧导航栏中，单击**投递任务管理**，选择地域、日志集和日志主题。

   - 在左侧导航栏中，单击**日志主题**，选择需要配置投递到 Ckafka 任务的日志主题，进入日志主题管理页面。

4. 单击**投递到Ckafka** 页签，进入投递到 Ckafka 配置页面。
5. 单击右侧的**编辑**，开启投递到 Ckafka 开关，选择相应的 Ckafka 实例以及对应的 Topic，选择需要投递的日志字段。

6. 单击**确定**，启动投递到 Ckafka，任务状态显示为“已开启”则表示开启成功。

> ⚠ 注意：

> 如需在投递至 Ckafka 前对日志进行清洗加工过滤，可参考使用 数据加工 操作。

常见问题

日志数据无法投递 Ckafka，怎么办?

如果您在腾讯云 Ckafka 开启了 ACL 鉴权，会导致日志数据无法投递。请关闭该 Topic 的 ACL。

提示没有读写 Ckafka Topic 的权限，怎么办?

如果您直接使用 API 接口投递数据到 Ckafka，可能会存在读写 Ckafka Topic 的权限问题。因为，如果您在控制台使用该功能，系统会引导您完成相关授权，如果您直接调用 API 投递，则需要手动授权。具体的排查和解决方案请参见 投递权限查看及配置。

投递至 ES
通过云函数转储至 ES

最近更新时间：2022-03-30 14:54:54

操作场景

本文为您介绍如何通过云函数 SCF 将 CLS 日志转储至 Elasticsearch Service（ES）。其中，CLS 主要用于日志采集，SCF 主要提供节点计算能力。数据处理流程图请参见 函数处理概述。

操作步骤

创建云函数 SCF

1. 登录云函数控制台，选择左侧导航栏中的 **函数服务**。
2. 在“函数服务”页面上方选择**北京**地域，并单击**新建**进入新建函数页面，配置以下参数：
   - **函数名称**：命名为 “CLSdemo”。
   - **运行环境**：选择 “Python 2.7”。
   - **创建方式**：选择**模板函数**。
   - **模糊搜索**：输入“CLS 消息转储至 ES”，并进行搜索。
3. 单击模板中的**查看详情**，即可在弹出的“模板详情”窗口中查看相关信息，支持下载操作。

4. 基本信息配置完成之后，单击**下一步**，进入函数配置页面。
5. 函数配置保持默认配置，单击**完成**，完成函数的创建。

> ⚠ 注意：

函数需要在**函数配置**页面中，选择和 CLS 相同的 VPC 和子网。如下图所示：

配置 CLS 触发器

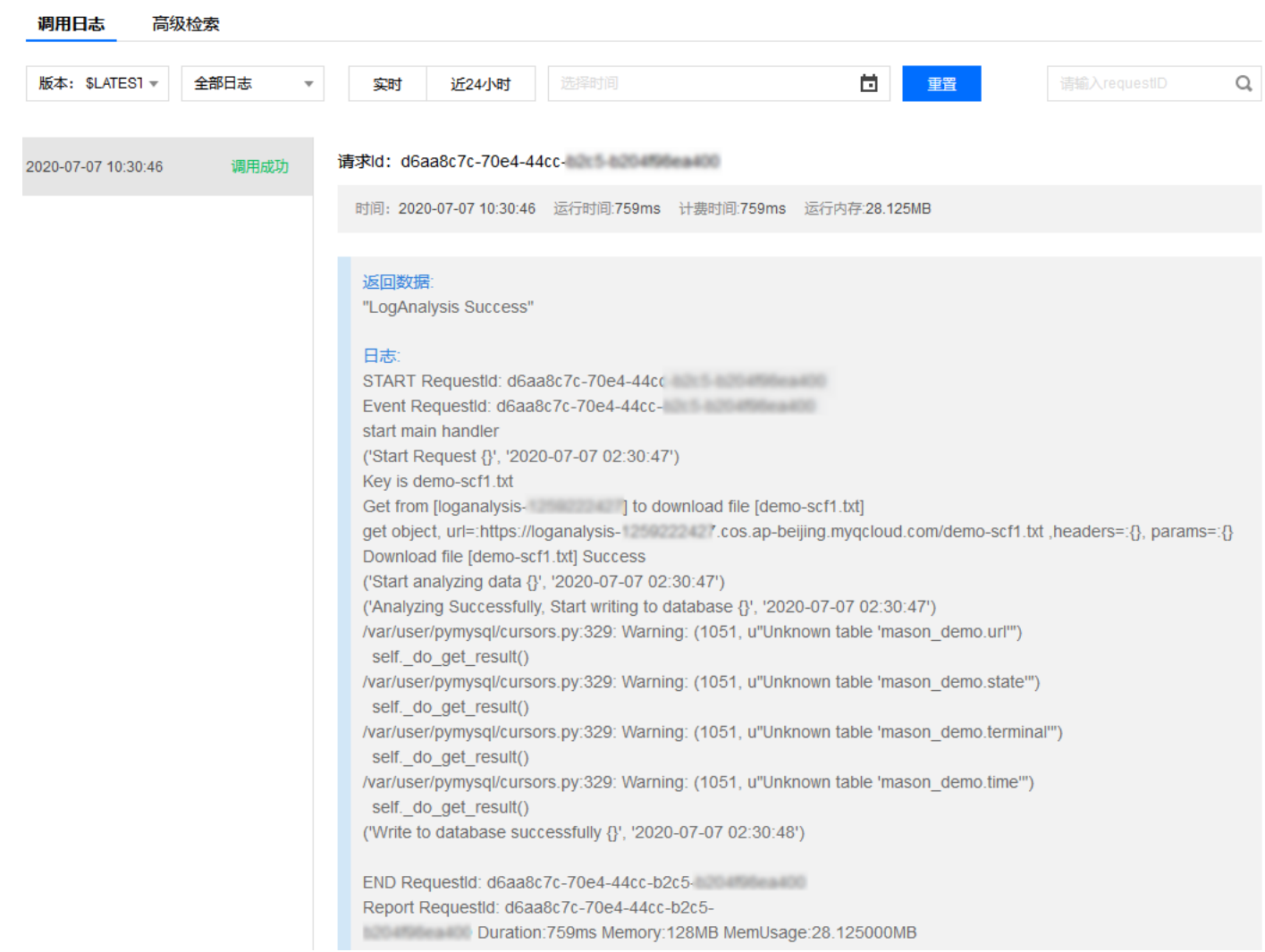
在日志主题详情页面，选择**函数处理**并单击**新建**。在弹出的“函数处理”窗口中添加已创完成的函数。如下图所示：

主要参数信息如下，其余配置项请保持默认：

- **命名空间**：选择函数所在的命名空间。
- **函数名**：选择 创建云函数 SCF 步骤中已创建的云函数。
- **别名**：选择函数别名。
- **最长等待时间**：单次事件拉取的最长等待事件，默认60s。

测试函数功能

1. 下载 测试样例 中的日志文件，并解压出 demo-scf1.txt，导入至源端 CLS 服务。
2. 切换至 云函数控制台，查看执行结果。
   在函数详情页面中选择**日志查询**页签，可以看到打印出的日志信息。如下图所示：



3. 切换至 Elasticsearch Service 控制台，查看数据转储及加工结果。

> **说明：**
> 您可以根据自身的需求编写具体的数据加工处理方法。

Kafka 协议消费

最近更新时间：2022-06-24 15:00:12

概述

您可以通过 Kafka 协议消费，将采集到日志服务（Cloud Log Service，CLS）的数据，消费到下游的大数据组件或者数据仓库。例如，Kafka、HDFS、Hive、Flink，以及腾讯云产品 Oceanus、EMR 等。

本文提供了 Flink、Logstash 消费日志主题的 demo。

支持的 Kafka 协议版本

Kafka 1.1.1及更早的版本

内网消费和外网消费说明

- 内网和外网的定义：例如您在广州地域的日志主题，使用 Kafka 消费协议，消费到广州地域的腾讯云 Oceanus，则属于内网消费。若消费到上海地域的腾讯云 Oceanus，则属于外网消费。
- 计费的区别：内网流量费用0.18元/GB，外网流量费用0.8元/GB。
- 消费服务域名的区别：在控制台页面会给出内网服务域名和外网服务域名，请按需选择。

使用限制

- 目前仅支持当前数据消费，不支持历史数据的消费。
- 当您在 SDK 中配置 auto_offset_reset='earliest' 时，可以回溯两小时前的数据。

前提条件

- 已开通日志服务，创建日志集与日志主题，并成功采集到日志数据。
- 确保当前操作账号拥有开通 Kafka 协议消费的权限，权限问题请参见 自定义权限策略示例。

操作步骤

1. 登录日志服务控制台，选择左侧导航栏中的 **日志主题**。
2. 在“日志主题”页面，单击需要使用 Kafka 协议消费的日志主题 ID/名称，进入日志主题管理页面。
3. 在日志主题管理页面中，单击 **Kafka协议消费** 页签。
4. 单击右侧的**编辑**，将“当前状态”的开关按钮设置为打开状态后，单击**确定**。
5. 根据 CLS 给出消费的 Topic 信息，构造消费者。完成后如下图所示：

Kafka协议消费

Flink 消费 CLS 日志

开启日志的 kafka 消费协议

参考 操作步骤 开启日志的 kafka 消费协议，并获取消费的服务域名和 Topic。

确认 flink-connector-kafka 依赖

确保 flink lib 中有 flink-connector-kafka 后，直接在 sql 中注册 kafka 表即可使用。依赖如下：

```
<dependency>
<groupId>org.apache.flink</groupId>
<artifactId>flink-connector-kafka</artifactId>
```

```
<version>1.14.4</version>
</dependency>
```

注册 flink 表

```
CREATE TABLE `nginx_source`
(
`@metadata` STRING, -- 日志中字段
`@timestamp` TIMESTAMP, -- 日志中字段
`agent` STRING, -- 日志中字段
`ecs` STRING, -- 日志中字段
`host` STRING, -- 日志中字段
`input` STRING, -- 日志中字段
`log` STRING, -- 日志中字段
`message` STRING, -- 日志中字段
`partition_id` BIGINT METADATA FROM 'partition' VIRTUAL, -- kafka分区
`ts` TIMESTAMP(3) METADATA FROM 'timestamp'
) WITH (
'connector' = 'kafka',
'topic' = 'YourTopic', -- cls kafka协议消费控制台给出的的主题名称，例如out-633a268c-XXXX-4a4c-XXXX-7a9a1a7baXXXX
'properties.bootstrap.servers' = 'kafkaconsumer-ap-guangzhou.cls.tencentcs.com:9096', -- cls kakfa协议消费控制台给出的服务地址，例子中是广州地域的外网消费地址，请按照您的实际情况填写
'properties.group.id' = 'kafka_flink', -- kafka 消费组名称
'scan.startup.mode' = 'earliest-offset',
'format' = 'json',
'json.fail-on-missing-field' = 'false',
'json.ignore-parse-errors' = 'true' ,
'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required username="your username" password="your password";',--用户名是日志主题所属的日志集合ID，例如ca5cXXXX-dd2e-4ac0-af12-92d4b677d2c6，密码是用户的secretid#secrectkey组合的字符串，比AKIDWrwkHYYHjvqhz1mHVS8YhXXXX#XXXXuXtymIXT0Lac注意不要丢失#。
'properties.security.protocol' = 'SASL_PLAINTEXT',
'properties.sasl.mechanism' = 'PLAIN'
);
```

查询使用

执行成功后，即可查询使用。

```
select count(*) , host from nginx_source group by host;
```

Flume 消费 CLS 日志

若您需将日志数据消费到自建的 HDFS，Kafka 集群，则可以通过 Flume 组件来中转，具体操作参考如下示例。

开启日志的 kafka 消费协议

参考 操作步骤 开启日志的 kafka 消费协议，并获取消费的服务域名和 Topic。

Flume 配置

```
a1.sources = source_kafka
a1.sinks = sink_local
a1.channels = channel1

// 配置Source
a1.sources.source_kafka.type = org.apache.flume.source.kafka.KafkaSource
a1.sources.source_kafka.batchSize = 10
```

```
a1.sources.source_kafka.batchDurationMillis = 200000
a1.sources.source_kafka.kafka.bootstrap.servers = kafkaconsumer-ap-guangzhou.cls.tencentcs.com:9096 --cls kakfa协议消费控制台给出的服务地址，例子中是广州地域的外网消费地址，请按照您的实际情况填写
a1.sources.source_kafka.kafka.topics = YourClsTopic -- cls kafka协议消费控制台给出的的主题名称,例如out-633a268c-XXXX-4a4c-XXXX-7a9a1a7baXXXX
a1.sources.source_kafka.kafka.consumer.group.id = cls_flume_kafka
a1.sources.source_kafka.kafka.consumer.auto.offset.reset = earliest
a1.sources.source_kafka.kafka.consumer.security.protocol = SASL_PLAINTEXT
a1.sources.source_kafka.kafka.consumer.sasl.mechanism = PLAIN
a1.sources.source_kafka.kafka.consumer.sasl.jaas.config = org.apache.kafka.common.security.plain.PlainLoginModule required username="YourUsername"
password="YourPassword";--用户名是日志主题所属的日志集合ID，例如ca5cXXXX-dd2e-4ac0-af12-92d4b677d2c6，密码是用户的secretid#secrectkey组合的字符串，例如AKIDWrwkHYYHjvqhz1mHVS8YhXXXX#XXXXuXtymIXT0Lac注意不要丢失#



//配置sink
a1.sinks.sink_local.type = logger

a1.channels.channel1.type = memory
a1.channels.channel1.capacity = 1000
a1.channels.channel1.transactionCapacity = 100

//将source和sink绑定到channel
a1.sources.source_kafka.channels = channel1
a1.sinks.sink_local.channel = channel1
```

Python SDK

```
import uuid
from kafka import KafkaConsumer,TopicPartition,OffsetAndMetadata
consumer = KafkaConsumer(
//将控制台中的主题名称填写到此处，您将使用这个topic进行消费
'out-633a268c-XXXX-4a4c-XXXX-7a9a1a7baXXXX',
group_id = uuid.uuid4().hex,
auto_offset_reset='earliest',
//Kafka协议服务地址，将上图中服务接入信息填写到此处，如果您是在外网消费，请填写外网服务域名+端口，如果您在内网消费，请填写内网服务域名+端口。例子中使用的是内网服务。
bootstrap_servers = ['kafkaconsumer-ap-guangzhou.cls.tencentyun.com:9096'],
security_protocol = "SASL_PLAINTEXT",
sasl_mechanism = 'PLAIN',
//SASL信息，将日志主题所属的日志集id填入此处
sasl_plain_username = "ca5cXXXX-dd2e-4ac0-af12-92d4b677d2c6",
//SASL信息，#填入用户的secretid#secrectkey组合的字符串，注意不要丢失#
sasl_plain_password = "AKIDWrwkHYYHjvqhz1mHVS8YhXXXX#XXXXuXtymIXT0Lac",
api_version = (1,1,1)
)
print('begin')
for message in consumer:
print('begins')
print ("Topic:[%s] Partition:[%d] Offset:[%d] Value:[%s]" % (message.topic, message.partition, message.offset, message.value))
print('end')
```

监控告警
监控告警简介

最近更新时间：2022-04-20 18:25:40

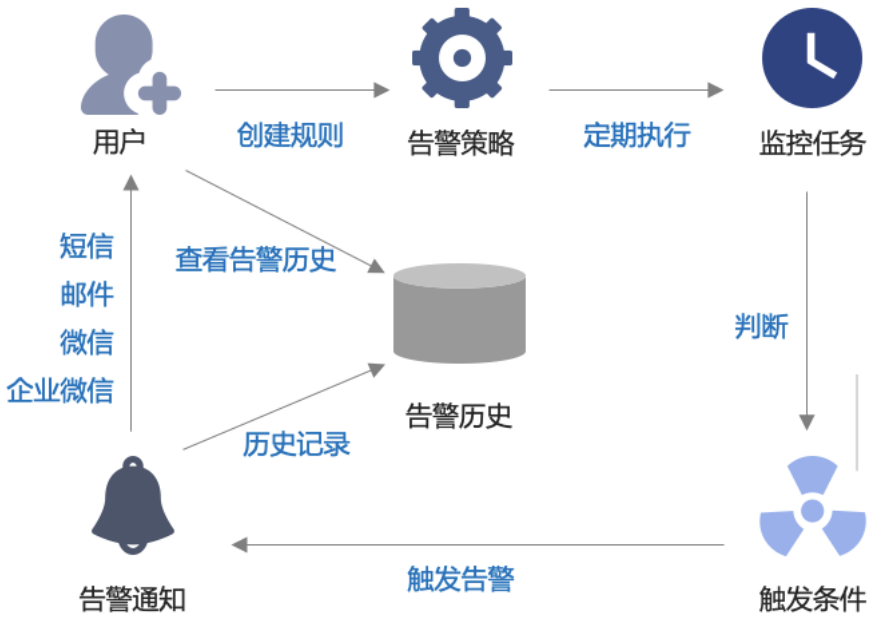
简介

日志服务支持对一个或多个日志主题设置告警策略，告警策略会周期性地执行监控任务，当查询分析结果满足触发条件时发送告警通知，方便用户及时发现异常问题。

相关概念

名称	描述
告警策略	监控告警的管理单元，一条告警策略包括监控对象、监控周期、触发条件、告警频率、通知模板等信息
监控对象	以日志主题为监控对象，并对该日志主题执行一种查询或分析语句，然后检查查询或分析结果
触发条件	检查查询分析结果，若触发条件表达式为 true，则会触发告警
监控周期	策略执行周期，支持固定周期（如每隔5分钟）和固定时间（如每天12:00）
告警频率	满足触发条件后的告警频率，避免频繁告警通知
通知渠道组	支持短信、微信、语音、邮件、webhook 等告警通知方式

流程图



管理告警策略
配置告警策略

最近更新时间：2022-04-11 18:29:52

操作场景

基于日志的监控告警场景，可以通过配置告警策略来完成。本文主要介绍如何在日志服务控制台上配置告警策略。

前提条件

- 日志已上传到某个日志主题。
- 日志主题已 配置索引。

操作步骤

步骤1：创建告警策略

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**监控告警 > 告警策略**，进入告警策略管理页面。
3. 单击**新建**，创建告警策略。

步骤2：配置监控对象和监控任务

- 监控对象
  选择需要监控的日志主题，并配置相应的分析语句及查询时间范围，详细参考以下：

名称	描述	示例
日志主题	需要配置监控告警的目标日志主题	例如，nginx 日志主题
分析语句	作用于日志主题的分析语句，分析语法参考	例1，统计出现 error 状态的日志条数 status:error | select count(*) as ErrCount 例2，统计域名（url:aaa.com）的平均请求延时情况 url:"aaa.com" | select avg(request_time) as Latency
查询时间范围	每次运行分析语句的数据时间范围	例1，统计近5分钟出现 error 状态的日志条数 例2，统计近1分钟请求的平均延时

- 监控周期
  表示监控任务的执行频率，日志服务提供如下两种周期配置方式：

周期配置方式	说明	示例
固定频率	按固定的时间间隔执行一次监控任务 时间间隔：1~1440 分钟，粒度：分钟级	每隔5分钟执行一次监控
固定时间	按固定的时间点执行一次监控任务 时间点范围：00:00~23:59，粒度：分钟级	每天02:00点执行一次监控

步骤3：配置告警策略

- 触发条件：判断是否满足触发告警的条件表达式，当满足条件时进行告警。
  日志服务提供 $N.keyname的方式引用分析结果，其中$N 表示当前告警策略中的第N个监控对象（详情参考 如何查看编号），keyname 表示对应的字段名称，例如 $1.status>500 表示编号为1的查询的 status字段大于500时触发告警，更多表达式语法参考 触发条件表达式语法。
- 告警频率：当持续满足触发条件达到一定次数（默认为1，有效值范围：1－10）以后，日志服务根据告警频率进行通知触达；通过配置持续周期的阈值可以避免一些不重要的偶发情况。例如，配置持续5个周期满足触发条件，表示累计触发次数达到5次以后，再进行通知触达。当修改了触发条件表达式，或计算过程中不满足表达式条件，累计次数会清零。
- 通知收敛：通过设置两次通知的时间间隔，避免频繁发送告警通知。当某次监控执行的结果满足触发条件，所持续累积的触发周期满足阈值，且离上次发送的通知时间满足通知间隔，则发送通知。例如，每15分钟告警一次，表示15分钟内只会收到一次告警通知。

步骤4：测试监控任务

配置告警策略完成后，单击**测试监控任务**，验证分析语句和触发条件语法是否正确。

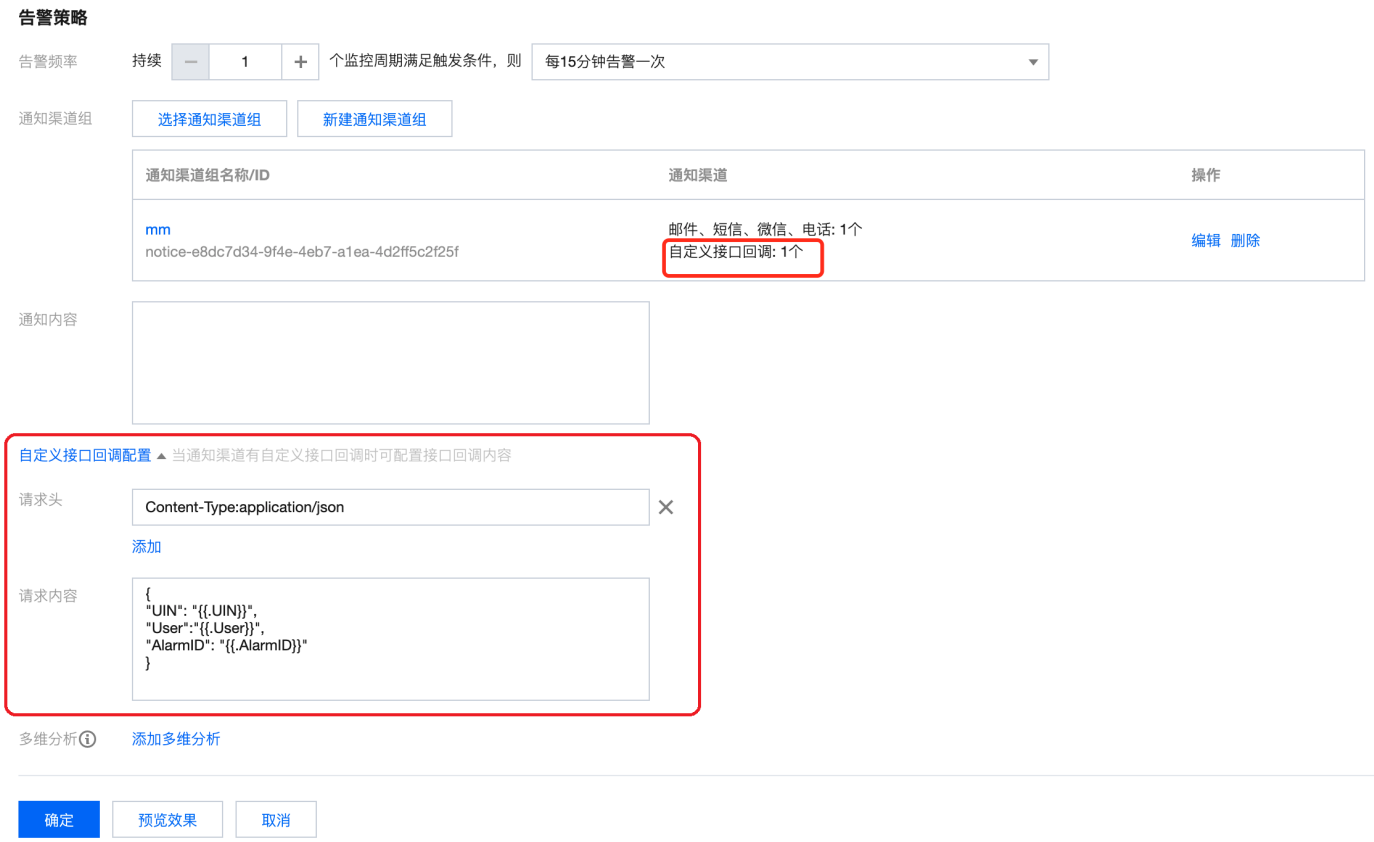
返回类似如下结果，即表示语法正确：

步骤5：告警通知

- **通知渠道组**
  通过关联通知渠道组，设置发送通知的方式及对象，支持短信、邮件、电话、微信、企业微信、自定义回调接口（webhook）等通知方式。详情参考 管理通知渠道组。
- **通知内容**
  用户可以通过在通知内容加入预设的变量，第一时间了解告警的具体内容。具体变量列表和说明请参考 通知内容变量。

- **自定义接口回调配置**
  如果用户选中的通知渠道组里包含自定义回调接口，则会出现自定义回调接口配置的输入框。具体变量列表和说明请参考 自定义接口回调变量。这里的变量与上面的**通知**

内容变量有重复的部分，但不完全相同，使用时以文档中列出的变量清单为准。



- **多维分析**

  用户单击添加多维分析可以在告警通知中增加字段TOP5及占比统计或自定义检索分析语句的多维分析内容。目前每条告警只支持配置最多3个多维分析。

多维分析的作用范围是告警执行语句规定的查询时间范围内的日志。如果某告警策略有多条执行语句，以其中最大的查询时间范围为准。例如，如果下图的告警策略被触发，则多维分析的作用范围是第二条执行语句的近15分钟。

监控任务

执行语句 1 **执行语句1**

执行语句: error | select count(*) as count

查询时间范围: 近5分钟

2 **执行语句2**

执行语句: warning | select count(*) as count

查询时间范围: 近15分钟

添加执行语句

执行周期: 固定频率 每 1 分钟

触发条件: $1.count > 10 && $2.count > 10

通过$N.keyname引用结果来定义表达式，如 $1.status>500 表示编号为1查询的status字段大于500时触发告警，详情参考帮助文档。

选中字段TOP5及占比统计后，用户要选择需要分析的字段（此处只能选择当前日志主题下开启了索引的字段）。

多维分析

名称: 请求客户端

类型: 字段TOP5及占比统计

在触发告警的日志中，针对目标字段进行分组统计，并返回前5项占比。例如，针对某接口错误率设置告警，日志中有userid字段记录用户信息，则可在告警信息中显示受影响用户Top5信息。

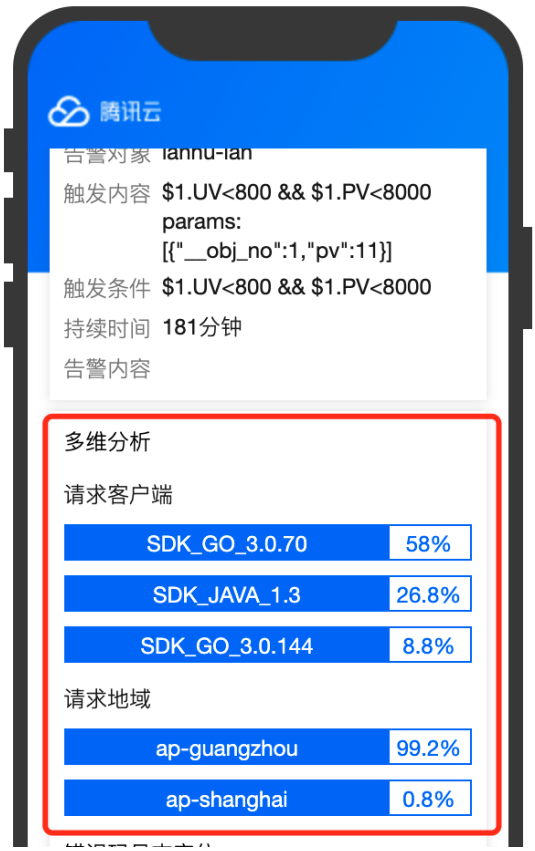
字段: key

key

名称

类型: 自定义检索分析语句

多维分析会展示在查询范围内的所有日志中，目标字段下出现最多的5个内容。



如果用户需要更灵活的内容配置，可以选择自定义检索分析语句，格式和告警的分析语句一致。

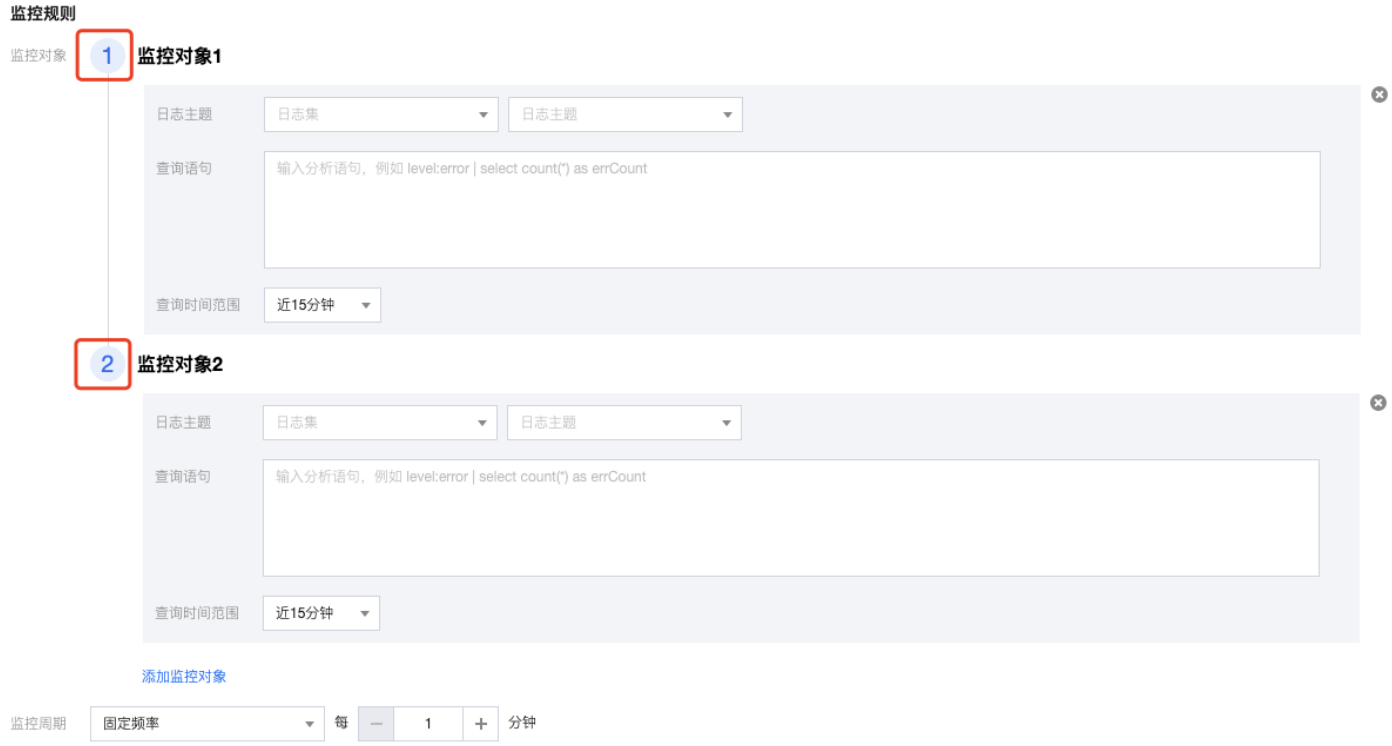
最终网页通知的展示效果如图所示：

完成配置后，单击**预览效果**预览各渠道的展示效果。

常见问题

如何查看编号?

在监控对象左侧显示当前查询的编号。第1个监控对象的查询编号为1，第2个监控对象的查询编号为2，以此类推。



触发条件表达式

最近更新时间：2022-04-20 18:33:51

触发条件表达式用于判断是否触发告警通知。监控对象执行的查询分析结果作为触发表达式的输入变量，当表达式为真时会触发告警。

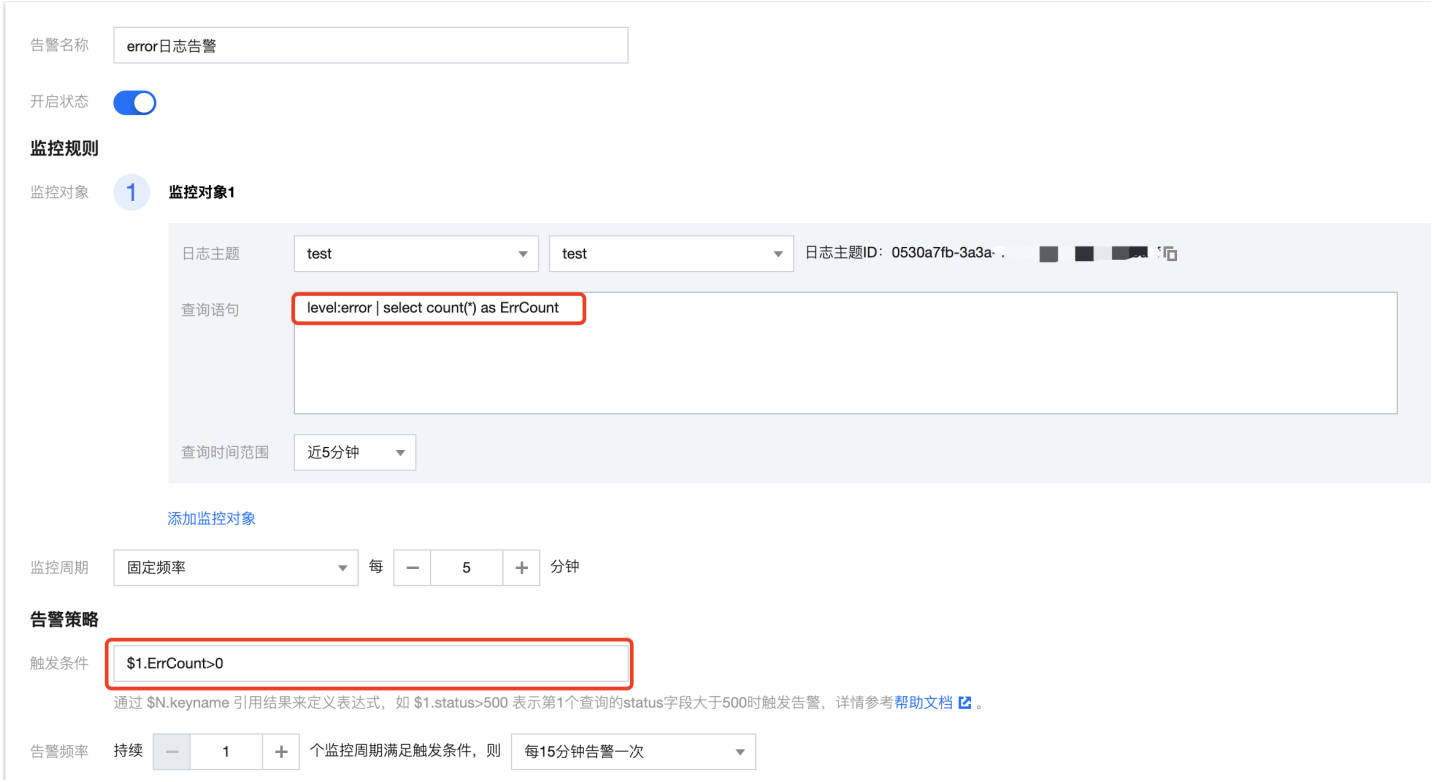
语法说明

运算符	说明	示例
$N.keyname	引用查询分析的结果，N为对应的监控对象编号，keyname 为查询分析结果中的字段名（keyname 首字符必须是字母，可以包含字母、数字、下滑线，推荐使用 AS 语法 对分析结果设置别名）	$1.ErrCount
+	求和运算符	$1.ErrCount+$1.FatCount>10
-	减法运算符	$1.Count-$1.InfoCount>100
*	乘法运算符	$1.RequestMilSec*1000>10
/	除法运算符	$1.RequestSec/1000>0.01
%	取模运算符	$1.keyA%10==0
==	比较运算符：等于	$1.ErrCount==100 $1.level=="Error"
>	比较运算符：大于	$1.ErrCount>100
<	比较运算符：小于	$1.pv<100
>=	比较运算符：大于等于	$1.ErrCount>=100
<=	比较运算符：小于等于	$1.pv<=100
!=	比较运算符：不等于	$1.level!="Info"
()	括号，控制运算优先级	($1.a+$1.b)/$1.c>100
&&	逻辑运算符：与	$1.ErrCount>100 && $1.level=="Error"
||	逻辑运算符：或	$1.ErrCount>100 || $1.level=="Error"

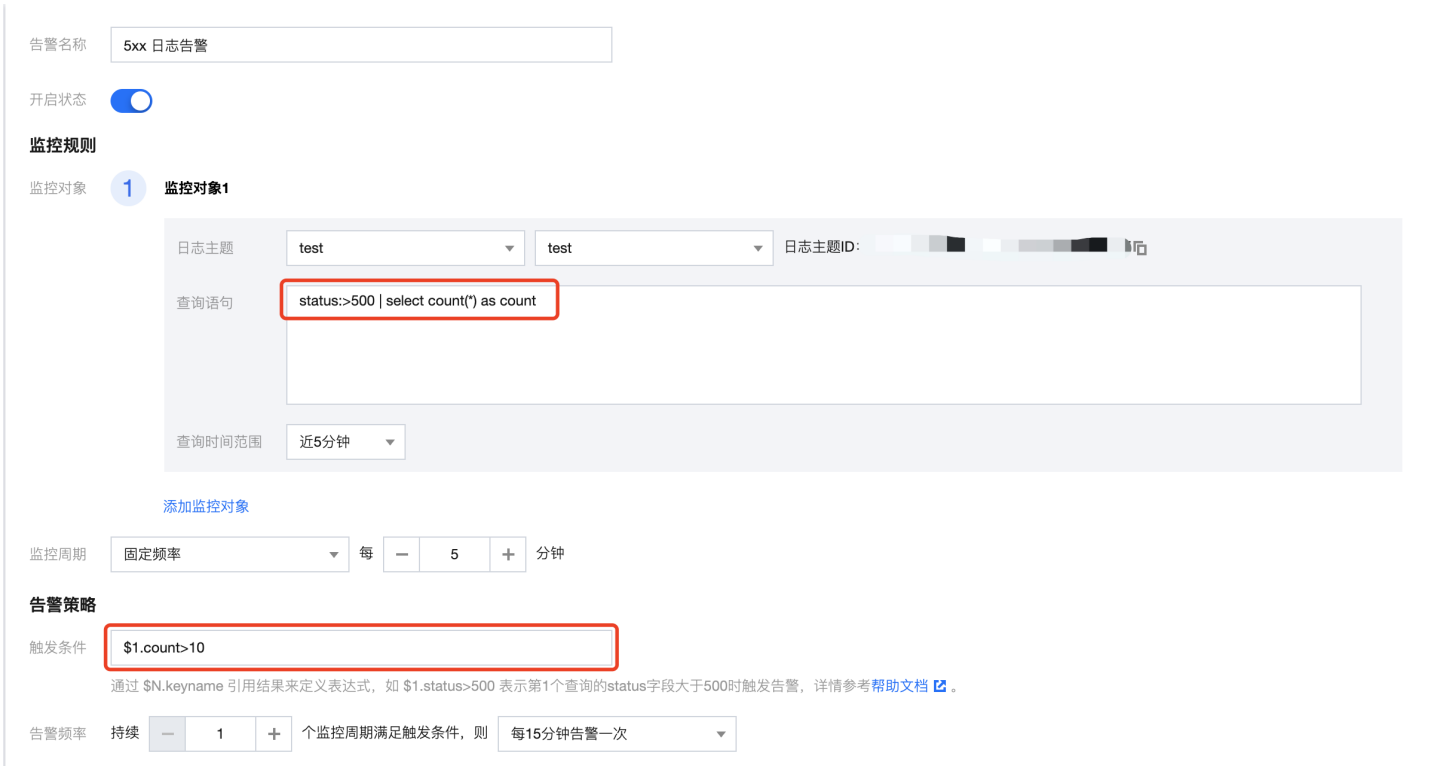
- 只有表达式判断为真时，才会触发告警。例如， $1.a+$1.b 计算结果为100，不会触发，$1.a+$1.b>=100则会触发。
- $N.keyname 中的keyname 为查询分析结果后的字段名（首字符必须是字母，可以包含字母、数字、下滑线）。例如，level:error | select count(*) AS errCount，errCount 可直接作为触发条件表达式中的 keyname。若字段名含有特殊字符，需要用[]将引用变量括起来，例如 [$1.count(*)]，推荐在分析语句中使用 AS 分析语句对分析结果字段名设置别名。
- 一个告警策略中可以设置多个监控对象（最多3个），每个监控对象有编号标识（从1开始以此递增），例如 $1.key1 引用编号为1的查询中的key1字段名， $2.key2 引用编号为2的查询中的key2字段名。
- 当查询分析结果返回多个值时，会根据返回结果依次计算1000次，当计算结果为 true 时停止。例如，表达式为 $1.a+$2.b>100，若分析1返回 m 条结果，分析2返回 n 条结果，则会进行 m * n 次计算，计算过程中当结果满足 $1.a+$2.b>100 为真时停止或计算超过1000次停止。

表达式示例

示例1：当近5分钟出现 error 级别日志时触发告警



示例2：当近5分钟5xx状态码出现10次触发告警



示例3: 当某 VIP 用户（uid:10001）的请求延时大于10s时触发告警


告警名称
VIP用户【uid:10001】请求延时告警
开启状态
监控规则
监控对象
1
监控对象1
日志主题
test
test
日志主题ID：0530a7fb-3a3a-47ea-aef3-0d33f9a4ba7f
查询语句
uid:10001 | select uid, avg(request_time) as latency group by uid
查询时间范围
近5分钟
添加监控对象
监控周期
固定频率
每
5
分钟
告警策略
触发条件
$1.latency>10
通过 $N.keyname 引用结果来定义表达式，如 $1.status>500 表示第1个查询的status字段大于500时触发告警，详情参考帮助文档 。
告警频率
持续
1
个监控周期满足触发条件，则
每15分钟告警一次

告警通知变量

最近更新时间：2022-06-16 11:56:55

在告警策略中配置**通知内容**和**自定义接口回调配置**时，可使用告警通知变量来自定义告警通知内容，使得收到的告警通知更加清晰准确的描述告警原因。

快速开始

在 配置告警策略 中，为通知内容填写如下配置信息：

```
详细日志：
{{.QueryLog[0][0]}}
```

配置完成后如下图所示：

收到告警通知时，通知内容将自动替换为下值，代表触发告警时最近的一条详细日志：

```
详细日志：
{"content":{"body_bytes_sent":"33352","http_referer":"-","http_user_agent":"Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/30.0.1599.17 Safari/537.36","remote_addr":"201.80.83.199","remote_user":"-","request_method":"GET","request_uri":"/content/themes/test-com/images/header_about.jpg","status":"404","time_local":"01/Nov/2018:01:16:31"},"fileName":"/root/testLog/nginx.log","pkg_id":"285A243662909DE3-70A","source":"172.17.0.2","time":1653831150008,"topicId":"a54de372-ffe0-49ae-a12e-c340bb2b03f2"}
```

通知变量

变量	含义	变量值示例	说明
{{.UIN}}	账号 ID	100007xxx827	-
{{.Nickname}}	账号昵称	xx企业	-
{{.Region}}	地域	广州	-
{{.Alarm}}	告警策略名称	Nginx 错误日志过多	-
{{.AlarmID}}	告警策略 ID	notice-3abd7ad6-15b7-4168-xxxx-52e5b961a561	-
{{.ExecuteQuery}}	执行语句	["status:>=400 | select count(*) as errorLogCount","status:>=400 | select count(*) as errorLogCount,request_uri group by request_uri order by count(*) desc"]	数组结构，{{.ExecuteQuery[0]}}代表第1个执行语句的详细日志，{{.ExecuteQuery[1]}}代表第2个执行语句的详细日志，以此类推
{{.Condition}}	触发条件	$1.errorLogCount > 1	-
{{.HappenThreshold}}	告警所需的触发条件持续满足次数	1	-

变量	含义	变量值示例	说明
{{.AlertThreshold}}	告警间隔时间	15	单位：分钟
{{.Topic}}	日志主题名称	nginxLog	-
{{.TopicId}}	日志主题 ID	a54de372-ffe0-49ae-xxxx-c340bb2b03f2	-
{{.StartTime}}	第一次告警触发时间	2022-05-28 18:56:37	时区：Asia/Shanghai
{{.StartTimeUnix}}	第一次告警触发时间戳	1653735397099	毫秒级 UNIX 时间戳
{{.NotifyTime}}	本次告警通知时间	2022-05-28 19:41:37	时区：Asia/Shanghai
{{.NotifyTimeUnix}}	本次告警通知时间戳	1653738097099	毫秒级 UNIX 时间戳
{{.NotifyType}}	告警通知类型	1	1代表告警通知，2代表恢复通知
{{.ConsecutiveAlertNums}}	连续告警次数	2	-
{{.Duration}}	告警持续时间	0	单位：分钟
{{.TriggerParams}}	告警触发时参数	$1.errorLogCount=5;	-
{{.DetailUrl}}	告警详情页面链接	https://alarm.cls.tencentcs.com/MDv2xxJh	无需登录账号
{{.QueryUrl}}	第一个执行语句的检索分析链接	https://alarm.cls.tencentcs.com/T0pkxxMA	-
{{.Message}}	通知内容	-	特指告警策略配置中填写的“通知内容”
{{.QueryResult}}	执行语句执行结果	详见 {{.QueryResult}} 说明	-
{{.QueryLog}}	执行语句中检索条件匹配到的详细日志	详见 {{.QueryLog}} 说明	-
{{.AnalysisResult}}	多维分析结果	详见 {{.AnalysisResult}} 说明	-

相关说明如下：

{{.QueryResult}}

- **含义：执行语句执行结果**
- **说明：数组结构，**{{.QueryResult[0]}}**代表第1个执行语句的执行结果，**{{.QueryResult[1]}}**代表第2个执行语句的执行结果，以此类推。**
- **变量值示例：**

  假设告警策略存在两条执行语句：

  第1个执行语句：status:>=400 | select count(*) as errorLogCount
  第2个执行语句：status:>=400 | select count(*) as errorLogCount,request_uri group by request_uri order by count(*) desc

  则变量值为：

```
[
[{
"errorLogCount": 7
}],
[{
"errorLogCount": 3,
"request_uri": "/apple-touch-icon-144x144.png"
}, {
"errorLogCount": 3,
"request_uri": "/feed"
}, {
"errorLogCount": 1,
"request_uri": "/opt/node_apps/test-v5/app/themes/basic/public/static/404.html"
```

```
}]
]
```

{{.QueryLog}}

- **含义：执行语句中检索条件匹配到的详细日志（不包含 SQL 中的过滤条件）**
- **说明：数组结构，{{.QueryLog[0]}} 代表第1个执行语句的详细日志，{{.QueryLog[1]}} 代表第2个执行语句的详细日志，以此类推。每个执行语句最多包含最近的10条详细日志。**
- **变量值示例：**


```
[
[{
"content": {
"__TAG__": {
"pod": "nginxPod",
"cluster": "testCluster"
},
"body_bytes_sent": "32847",
"http_referer": "-",
"http_user_agent": "Opera/9.80 (Windows NT 6.1; U; en-US) Presto/2.7.62 Version/11.01",
"remote_addr": "105.86.148.186",
"remote_user": "-",
"request_method": "GET",
"request_uri": "/apple-touch-icon-144x144.png",
"status": "404",
"time_local": "01/Nov/2018:00:55:14"
},
"fileName": "/root/testLog/nginx.log",
"pkg_id": "285A243662909DE3-5CD",
"source": "172.17.0.2",
"time": 1653739000013,
"topicId": "a54de372-ffe0-49ae-a12e-c340bb2b03f2"
}, {
"content": {
"__TAG__": {
"pod": "nginxPod",
"cluster": "testCluster"
},
"body_bytes_sent": "33496",
"http_referer": "-",
"http_user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/27.0.1453.93 Safari/537.3
6",
"remote_addr": "222.18.168.242",
"remote_user": "-",
"request_method": "GET",
"request_uri": "/opt/node_apps/test-v5/app/themes/basic/public/static/404.html",
"status": "404",
"time_local": "01/Nov/2018:00:54:37"
},
"fileName": "/root/testLog/nginx.log",
"pkg_id": "285A243662909DE3-5C8",
"source": "172.17.0.2",
"time": 1653738975008,
"topicId": "a54de372-ffe0-49ae-a12e-c340bb2b03f2"
}]
]
```

{{.AnalysisResult}}

- **含义：多维分析结果**
- **说明：对象结构，第一层级对象分别对应每个多维分析的结果。一级对象的 key 为多维分析的名称，value 为该多维分析的结果。**
- **变量值示例：**

  假设告警策略存在3个多维分析：

```
名称：Top URL
类型：字段TOP5及占比统计
字段：request_uri

名称：错误日志URL分布
类型：自定义检索分析
分析语句：status:>=400 | select count(*) as errorLogCount,request_uri group by request_uri order by count(*) desc

名称：详细错误日志
类型：自定义检索分析
分析语句：status:>=400
```

  则变量值为：


```
{
"Top URL": [{
"count": 77,
"ratio": 0.45294117647058824,
"value": "/"
}, {
"count": 20,
"ratio": 0.11764705882352941,
"value": "/favicon.ico"
}, {
"count": 7,
"ratio": 0.041176470588235294,
"value": "/blog/feed"
}, {
"count": 5,
"ratio": 0.029411764705882353,
"value": "/test-tile-service"
}, {
"count": 3,
"ratio": 0.01764705882352941,
"value": "/android-chrome-192x192.png"
}],
"详细错误日志": [{
"content": {
"__TAG__": {
"pod": "nginxPod",
"cluster": "testCluster"
},
"body_bytes_sent": "32847",
"http_referer": "-",
"http_user_agent": "Opera/9.80 (Windows NT 6.1; U; en-US) Presto/2.7.62 Version/11.01",
"remote_addr": "105.86.148.186",
"remote_user": "-",
"request_method": "GET",
"request_uri": "/apple-touch-icon-144x144.png",
"status": "404",
"time_local": "01/Nov/2018:00:55:14"
```

```
},
"fileName": "/root/testLog/nginx.log",
"pkg_id": "285A243662909DE3-5CD",
"source": "172.17.0.2",
"time": 1653739000013,
"topicId": "a54de372-ffe0-49ae-a12e-c340bb2b03f2"
}, {
"content": {
"__TAG__": {
"pod": "nginxPod",
"cluster": "testCluster"
},
"body_bytes_sent": "33496",
"http_referer": "-",
"http_user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/27.0.1453.93 Safari/537.3
6",
"remote_addr": "222.18.168.242",
"remote_user": "-",
"request_method": "GET",
"request_uri": "/opt/node_apps/test-v5/app/themes/basic/public/static/404.html",
"status": "404",
"time_local": "01/Nov/2018:00:54:37"
},
"fileName": "/root/testLog/nginx.log",
"pkg_id": "285A243662909DE3-5C8",
"source": "172.17.0.2",
"time": 1653738975008,
"topicId": "a54de372-ffe0-49ae-a12e-c340bb2b03f2"
}],
"错误日志URL分布": [{
"errorLogCount": 3,
"request_uri": "/apple-touch-icon-144x144.png"
}, {
"errorLogCount": 3,
"request_uri": "/feed"
}, {
"errorLogCount": 1,
"request_uri": "/opt/node_apps/test-v5/app/themes/basic/public/static/404.html"
}]
}
```


变量语法

变量语法类似 Go Template 语法，可对告警通知变量进行提取和格式化处理，以更加清晰的呈现在告警通知内容中。所有变量及变量语法均位于 {{ }} 中，外部的文本不会进行处理。

变量提取

语法格式:

```
{{.variable[x]}} 或 {{index .variable x}}
{{.variable.childNodeName}} 或 {{index .variable "childNodeName"}}
```

语法说明:

- 变量为数组时，使用 {{.variable[x]}} 按数组下标提取对应的数组元素，其中 x 为大于等于0的整数，等价于 {{index .variable x}}。
- 变量为对象时，使用 {{.variable.childNodeKey}} 按子级对象名称（key）提取对应的子级对象值（value），等价于 {{index .variable "childNodeName"}}。

> ⚠ 注意：
> 子级对象名称包含空格时，请使用 {{index .variable "childNodeName"}} 形式的语法，例如 {{index .AnalysisResult "Top URL"}} 。

使用示例：

{{.QueryResult}} 变量值为：

```
[
[{
"errorLogCount": 7 //提取该值
}],
[{
"errorLogCount": 3,
"request_uri": "/apple-touch-icon-144x144.png"
}, {
"errorLogCount": 3,
"request_uri": "/feed"
}, {
"errorLogCount": 1,
"request_uri": "/opt/node_apps/test-v5/app/themes/basic/public/static/404.html"
}]
]
```

通过以下表达式获取第一个数组中的 errorLogCount 的值：

```
{{.QueryResult[0][0].errorLogCount}}
```

返回结果：

```
7
```

循环遍历

语法格式：

```
{{range .variable}}
自定义内容{{.childNode1}}自定义内容{{.childNode2}}...
{{end}}
```

或

```
{{range $key,$value := .variable}}
自定义内容{{$key}}自定义内容{{$value}}...
{{end}}
```

语法说明：

变量为数组或包含多个子级对象的对象时，可使用该语法将其中的每个元素/对象按指定格式展示出来。

使用示例：

{{.QueryResult}} 变量值为：

```
[
[{
"errorLogCount": 7
}],
[{
```

```
"errorLogCount": 3,
"request_uri": "/apple-touch-icon-144x144.png"
}, {
"errorLogCount": 3,
"request_uri": "/feed"
}, {
"errorLogCount": 1,
"request_uri": "/opt/node_apps/test-v5/app/themes/basic/public/static/404.html"
}]
]
```

通过以下表达式分别展示第二个数组中每个 request_uri 对应的 errorLogCount：

```
{{range .QueryResult[1]}}
* {{.request_uri}}错误日志数：{{.errorLogCount}}
{{end}}
```

返回结果：

```
* /apple-touch-icon-144x144.png错误日志数：3

* /feed错误日志数：3

* /opt/node_apps/test-v5/app/themes/basic/public/static/404.html错误日志数：1
```

条件判断

语法格式：

```
{{if boolen}}
xxx
{{end}}
```

或

```
{{if boolen}}
xxx
{{else}}
xxx
{{end}}
```

语法说明：

根据条件判断结果分别执行对应的表达式，条件判断支持使用 and or not 进行逻辑运算，支持比较大小值。

```
eq arg1 arg2 ： arg1 == arg2时为true
ne arg1 arg2 ： arg1 != arg2时为true
lt arg1 arg2 ： arg1 < arg2时为true
le arg1 arg2 ： arg1 <= arg2时为true
gt arg1 arg2 ： arg1 > arg2时为true
ge arg1 arg2 ： arg1 >= arg2时为true
```

使用示例：

{{.QueryResult}} **变量值为：**

```
[
[{
"errorLogCount": 7
}],
[{
"errorLogCount": 3,
"request_uri": "/apple-touch-icon-144x144.png"
}, {
"errorLogCount": 3,
"request_uri": "/feed"
}, {
"errorLogCount": 1,
"request_uri": "/opt/node_apps/test-v5/app/themes/basic/public/static/404.html"
}]
]
```

通过以下表达式分别展示第二个数组中 errorLogCount 大于等于2且小于等于100的 request_uri 及对应的 errorLogCount：

```
{{range .QueryResult[1]}}
{{if and (ge .errorLogCount 2) (le .errorLogCount 100)}}
* {{.request_uri}}错误日志数：{{.errorLogCount}}
{{end}}
{{end}}
```

返回结果：

```
* /apple-touch-icon-144x144.png错误日志数：3

* /feed错误日志数：3
```

移除空白符号

语法格式：

```
{{- xxx}} 或 {{xxx -}}
```

语法说明：

变量语法在执行过程中会保留语法本身带来的空格、缩进和换行等空白符号，例如循环迭代和条件判断使用示例中的返回结果中包含较多空白行，影响展示效果，此时可在 {{ }} 的头部或尾部使用 - 移除其前面或后面的空白符号。

使用示例：

沿用条件判断中的示例，表达式修改为：

```
{{- range .QueryResult[1]}}
{{- if and (ge .errorLogCount 2) (le .errorLogCount 100)}}
* {{.request_uri}}错误日志数：{{.errorLogCount}}
{{- end}}
{{- end}}
```

返回结果：

```
* /apple-touch-icon-144x144.png错误日志数：3
* /feed错误日志数：3
```

变量函数

转义特殊字符

语法格式：

```
{{escape .variable}}
```

语法说明：

告警变量中很多变量包含特殊符号，如果直接将变量拼接在 JSON 字符串中进行自定义接口回调，可能会导致 JSON 格式错误，导致回调失败。此时可对原始变量值进行转义，再拼接至 JSON 字符串中。

使用示例：

{{.ExecuteQuery[0]}} 的变量值为 status:>=400 | select count(*) as "错误日志数"
如果不使用转义，自定义接口回调配置中请求内容为：

```
{
"Query":"{{.ExecuteQuery[0]}}"
}
```

返回结果如下（不是一个合法的 JSON）：

```
{
"Query":"status:>=400 | select count(*) as "错误日志数""
}
```

此时可使用转义，将自定义接口回调配置中请求内容修改为：

```
{
"Query":"{{escape .ExecuteQuery[0]}}"
}
```

返回结果如下（符合 JSON 语法要求）：

```
{
"Query":"status:>=400 | select count(*) as \"错误日志数\""
}
```

字符串截取

按长度截取

语法格式：

```
{{substr .variable start}} 或 {{substr .variable start length}}
```

语法说明：

按指定起始位置和长度（可选）对字符串进行截取。

使用示例：

{{.QueryLog[0][0].fileName}} 变量值为：

```
/root/testLog/nginx.log
```

通过以下表达式获取第6个字符开始、总长7个字符的字符串：

```
{{substr .QueryLog[0][0].fileName 6 7 }}
```

返回结果：

```
testLog
```

按首尾字符截取

语法格式：

```
{{extract .variable "startstring" ["endstring"]}}
```

语法说明：

按指定起始字符和结束字符（可选）对字符串进行截取。

使用示例：

{{.QueryLog[0][0].fileName}} 变量值为：

```
/root/testLog/nginx.log
```

通过以下表达式获取 /root/ 和 /nginx 之间的字符串：

```
{{extract .QueryLog[0][0].fileName "/root/" "/nginx"}}
```

返回结果：

```
testLog
```

是否包含指定字符串

语法格式：

```
{{containstr .variable "searchstring"}}
```

语法说明：

变量值中是否包含指定的字符串，判断结果可用于条件判断语法中。

使用示例：

{{.QueryLog[0][0].fileName}} 变量值为：

```
/root/testLog/nginx.log
```

通过以下表达式获取 /root/ 和 /nginx 之间的字符串：

```
{{if containstr .QueryLog[0][0].fileName "test"}}
测试日志
{{else}}
非测试日志
{{end}}
```

返回结果：

```
测试日志
```

UNIX 时间戳转换

语法格式：

```
{{fromUnixTime .variable}} 或 {{fromUnixTime .variable "timezone"}}
```

语法说明：

将 UNIX 时间戳（支持毫秒及秒级时间戳）转换为可读的日期时间。其中时区可选，默认为 Asia/Shanghai。

使用示例：

{{.QueryLog[0][0].time}} 变量值为：

```
1653893435008
```

通过以下表达式分别获区不同时区下的日期时间：

```
{{fromUnixTime .QueryLog[0][0].time}}
{{fromUnixTime .QueryLog[0][0].time "Asia/Shanghai"}}
{{fromUnixTime .QueryLog[0][0].time "Asia/Tokyo"}}
```

返回结果：

```
2022-05-30 14:50:35.008 +0800 CST
2022-05-30 14:50:35.008 +0800 CST
2022-05-30 15:50:35.008 +0900 JST
```

参考案例

案例1：在告警通知中展示最近一条详细日志

需求场景：

将符合执行语句检索条件的最近一条详细日志添加至告警通知中，以 key:value 的形式展示，每行一个 key，不包含 CLS 预置字段和元数据字段。

通知内容配置：

```
{{range $key,$value := .QueryLog[0][0].content}}
{{if not (containstr $key "__TAG__")}}
{{- $key}}:{{$value}}
{{- end}}
{{- end}}
```

其中 .QueryLog[0][0] 代表符合告警策略第一条执行语句检索条件的最近一条详细日志，其值为：

```
{
"content": {
"__TAG__": {
"a": "b12fgfe",
"c": "fgerhcdhgj"
},
"body_bytes_sent": "33704",
```

```
"http_referer": "-",
"http_user_agent": "Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.3319.102 Safari/537.36",
"remote_addr": "247.0.249.191",
"remote_user": "-",
"request_method": "GET",
"request_uri": "/products/hadoop)",
"status": "404",
"time_local": "01/Nov/2018:07:54:08"
},
"fileName": "/root/testLog/nginx.log",
"pkg_id": "285A243662909DE3-210B",
"source": "172.17.0.2",
"time": 1653908859008,
"topicId": "a54de372-ffe0-49ae-a12e-c340bb2b03f2"
}
```


告警通知内容：

```
remote_addr:247.0.249.191
time_local:01/Nov/2018:07:54:08
http_user_agent:Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.3319.102 Safari/537.36
remote_user:-
http_referer:-
body_bytes_sent:33704
request_method:GET
request_uri:/products/hadoop)
status:404
```

案例2：告警通知中展示执行语句执行结果

需求场景：

将执行语句执行结果中符合触发条件的部分添加至告警通知中，以列表的形式展示。

告警策略执行语句为： status:>=400 | select count(*) as errorLogCount,request_uri group by request_uri order by count(*) desc

触发条件为： $1.errorLogCount > 10

通知内容配置：

```
{{range .QueryResult[0]}}
{{- if gt .errorLogCount 10}}
{{.request_uri}}错误日志数：{{.errorLogCount}}
{{- end}}
{{- end}}
```

其中 .QueryResult[0] 代表告警策略第一条执行语句的执行结果，其值为：


```
[{
"errorLogCount": 161,
"request_uri": "/apple-touch-icon-144x144.png"
}, {
"errorLogCount": 86,
"request_uri": "/opt/node_apps/test-v5/app/themes/basic/public/static/404.html"
}, {
"errorLogCount": 33,
"request_uri": "/feed"
}, {
"errorLogCount": 26,
"request_uri": "/wp-login.php"
```

```
}, {
"errorLogCount": 10,
"request_uri": "/safari-pinned-tab.svg"
}, {
"errorLogCount": 7,
"request_uri": "/mstile-144x144.png"
}, {
"errorLogCount": 4,
"request_uri": "/atom.xml"
}, {
"errorLogCount": 3,
"request_uri": "/content/plugins/prettify-gc-syntax-highlighter/launch.js?ver=3.5.2?ver=3.5.2"
}]
```

告警通知内容：

```
/apple-touch-icon-144x144.png错误日志数：161
/opt/node_apps/elastic-v5/app/themes/basic/public/static/404.html错误日志数：86
/feed错误日志数：33
/wp-login.php错误日志数：26
```

案例3：使用钉钉/飞书接收告警通知，并将常用的基本信息添加至告警通知中

需求场景：

通过自定义接口回调将告警通知发送至钉钉/飞书群中，通知内容包含常用的基本字段，与短信告警和邮件告警类似。

通知内容配置：

```
{{- if eq .NotifyType 1 -}}
告警策略：{{.Alarm}}
日志主题：{{.Topic}}
触发条件：{{.Condition}}
当前数据：{{.TriggerParams}}
持续时间：{{.Duration}}分钟
多维分析：
{{- range $key,$value := .AnalysisResult}}
{{$key}}
{{$value}}
{{- end}}
详细报告：{{.DetailUrl}}
查询日志：{{.QueryUrl}}
{{- end}}

{{- if eq .NotifyType 2 -}}
告警策略：{{.Alarm}}
日志主题：{{.Topic}}
触发条件：{{.Condition}}
恢复时间：{{.NotifyTime}}，持续{{.Duration}}分钟
{{- end}}
```

自定义接口回调配置：

请求头：

```
Content-Type: application/json
```

飞书请求内容：

```
{
"msg_type": "post",
"content": {
"post": {
"zh_cn": {
"title": "{{if eq .NotifyType 1}}【告警】账号{{escape .UIN}}({{escape .User}})的日志服务触发告警{{else}}【告警恢复】账号{{escape .UIN}}({{escape .User}})的日志服务以下告警已恢复{{end}}",
"content": [
[{
"tag": "text",
"text": "{{substr (escape .Message) 0 10000}}"
}],
[{
"tag": "at",
"user_id": "all"
}]
]
}
}
}
}
```

钉钉请求内容：

```
{
"msgtype": "text",
"text": {
"content": "{{if eq .NotifyType 1}}【告警】账号{{escape .UIN}}({{escape .User}})的日志服务触发告警{{else}}【告警恢复】账号{{escape .UIN}}({{escape .User}})的日志服务以下告警已恢复{{end}}\n{{substr (escape .Message) 0 10000}}"
},
"at": {
"atMobiles": [
""
],
"atUserIds": [
""
],
"isAtAll": true
}
}
```

> **说明：**
> **飞书/钉钉请求内容中** {{substr (escape .Message) 0 10000}} **表示通知内容最长1万个字符，超出后将截断，以避免超出飞书/钉钉 API 长度限制，导致通知发送失败。**

告警效果：

飞书告警效果如下图所示：


机器人Battle 机器人 通过webhook将自定义服务的消息推送至飞书

【告警】账号10000 的日志服务触发告警
告警策略：Nginx错误日志过多
日志主题：nginxLog
触发条件：$1.errorLogCount > 1
当前数据：$1.errorLogCount=407;
持续时间：0分钟
多维分析：
错误日志URL分布
[{"errorLogCount":172,"request_uri":"/apple-touch-icon-144x144.png"},
{"errorLogCount":90,"request_uri":"/opt/node_apps/ -
v5/app/themes/basic/public/static/404.html"},{"errorLogCount":36,"request_uri":"/feed"},
{"errorLogCount":30,"request_uri":"/wp-login.php"},{"errorLogCount":12,"request_uri":"/safari-
pinned-tab.svg"},{"errorLogCount":7,"request_uri":"/mstile-144x144.png"},
{"errorLogCount":4,"request_uri":"/atom.xml"},
{"errorLogCount":3,"request_uri":"/content/themes/ org/js/froogaloop.min.js?
ver=1?ver=1"},{"errorLogCount":3,"request_uri":"/content/plugins/prettify-gc-syntax-
highlighter/launch.js?ver=3.5.2?ver=3.5.2"},
{"errorLogCount":2,"request_uri":"/blog/undefined"}]
详细报告：https://alarm.cls.tencentcs.com/
查询日志：https://alarm.cls.tencentcs.com/
@所有人

钉钉告警效果如下图所示：


CLS告警 机器人 20:17

【告警】账号1000075)的日志服务触发告警
告警策略：Nginx错误日志过多
日志主题：nginxLog
触发条件：$1.errorLogCount > 1
当前数据：$1.errorLogCount=407;
持续时间：0分钟
多维分析：
错误日志URL分布
[{"errorLogCount":172,"request_uri":"/apple-touch-icon-144x144.png"},
{"errorLogCount":90,"request_uri":"/opt/node_apps/ -
v5/app/themes/basic/public/static/404.html"},{"errorLogCount":36,"request_uri":"/feed"},
{"errorLogCount":30,"request_uri":"/wp-login.php"},{"errorLogCount":12,"request_uri":"/safari-
pinned-tab.svg"},{"errorLogCount":7,"request_uri":"/mstile-144x144.png"},
{"errorLogCount":4,"request_uri":"/atom.xml"},
{"errorLogCount":3,"request_uri":"/content/themes/ /js/froogaloop.min.js?
ver=1?ver=1"},{"errorLogCount":3,"request_uri":"/content/plugins/prettify-gc-syntax-
highlighter/launch.js?ver=3.5.2?ver=3.5.2"},
{"errorLogCount":2,"request_uri":"/blog/undefined"}]
详细报告：https://alarm.cls.tencentcs.com/
查询日志：https://alarm.cls.tencentcs.com/ @所有人

收到 回复

管理告警接收方式
短信及电话接收告警通知

最近更新时间：2022-06-30 09:02:06

操作场景

本文将为您介绍如何通过短信及电话接收告警通知。

相关限制

电话告警存在发送频率限制，建议仅通过电话接收非常紧急的告警。具体频率限制如下：

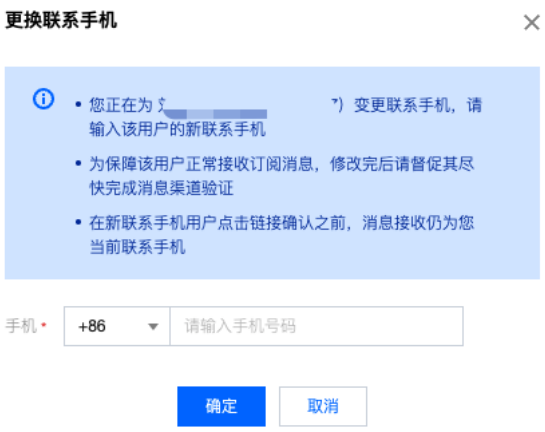
- 同一号码30秒内最多接收1条电话告警，10分种内最多接收3条电话告警，1天内最多接收50条电话告警。
- 同一号码22:00至次日08:00最多接收3条电话告警。

以上限制不局限于日志服务产生的电话告警通知，云监控及消息中心产生的电话通知会共享该限制。

操作步骤

验证手机号码

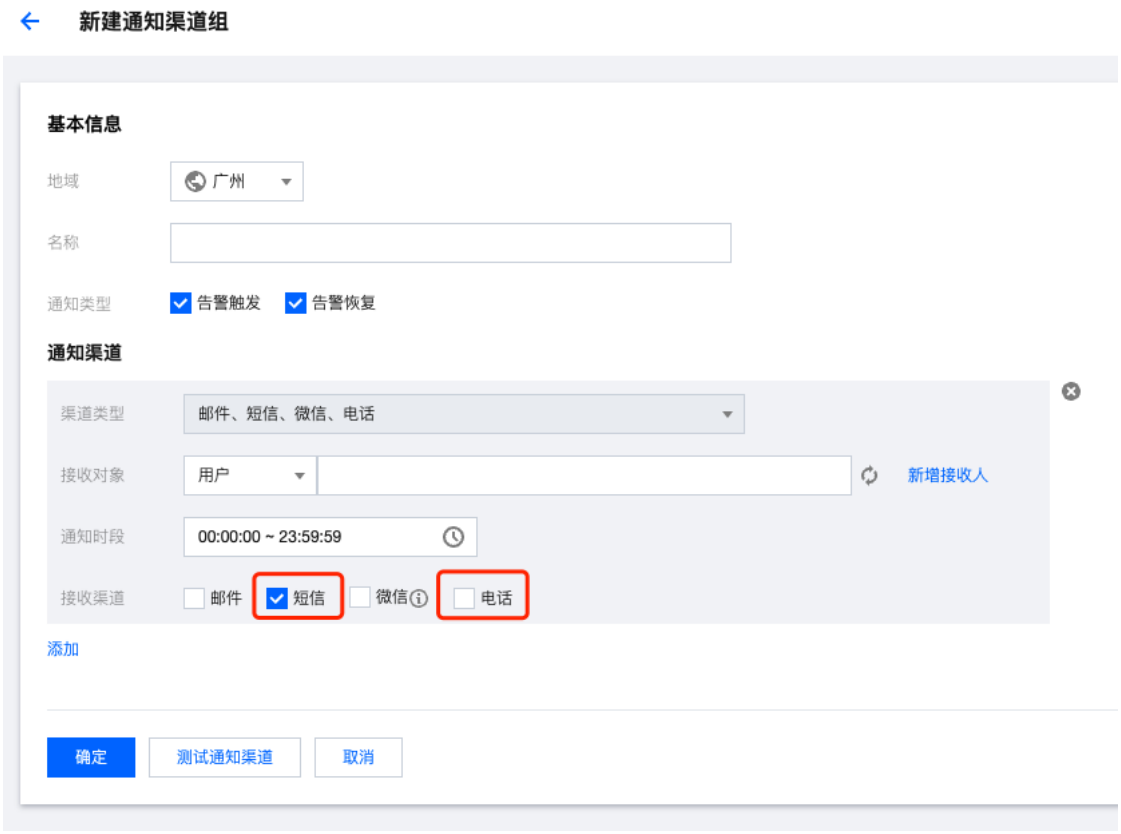
1. 登录 访问管理控制台。
2. 在左侧导航栏中，单击**用户 > 用户列表**，进入用户列表页面。
3. 找到需要配置接收短信及电话通知的用户，单击用户名称进入用户详情页面。
4. 在“联系手机”栏中，单击 。
5. 在弹出的窗口中，填写您的手机号码，单击**确定**，并按照其中的提示完成消息渠道验证。



新建短信及电话告警渠道

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**监控告警 > 通知渠道组**，进入通知渠道组列表页面。
3. 单击**新建**。

4. 在**新建通知渠道组**页面中，设置如下主要信息，单击**确定**。



- **名称**：自定义通知渠道组名称。
- **接收对象**：选择需要通知和触达的用户/用户组。
- **渠道类型**：选择“邮件、短信、微信、电话”。
- **接收渠道**：勾选“短信”或“电话”。

邮件接收告警通知

最近更新时间：2022-04-20 18:30:02

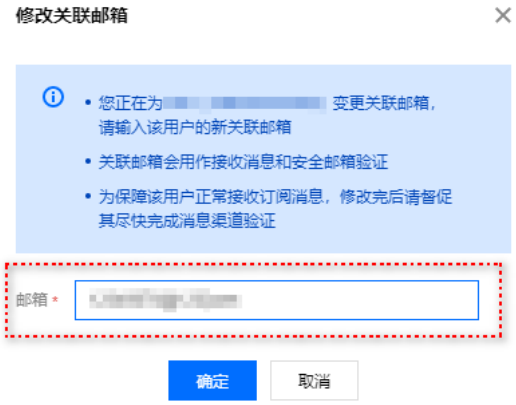
操作场景

本文将为您介绍如何通过邮件接收告警通知。

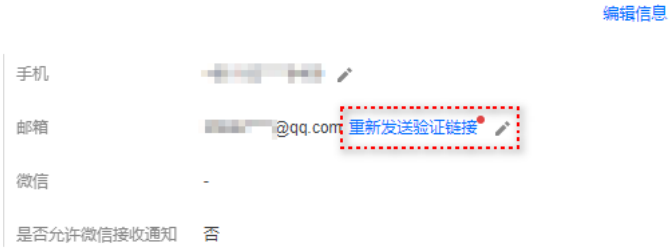
操作步骤

验证邮件

1. 登录 访问管理控制台。
2. 在左侧导航栏中，单击**用户 > 用户列表**，进入用户列表页面。
3. 找到需要配置接收邮件通知的用户，单击用户名称进入用户详情页面。
4. 在“联系邮箱”栏中，单击 。
5. 在弹出的窗口中，填写您的邮箱，单击**确定**。



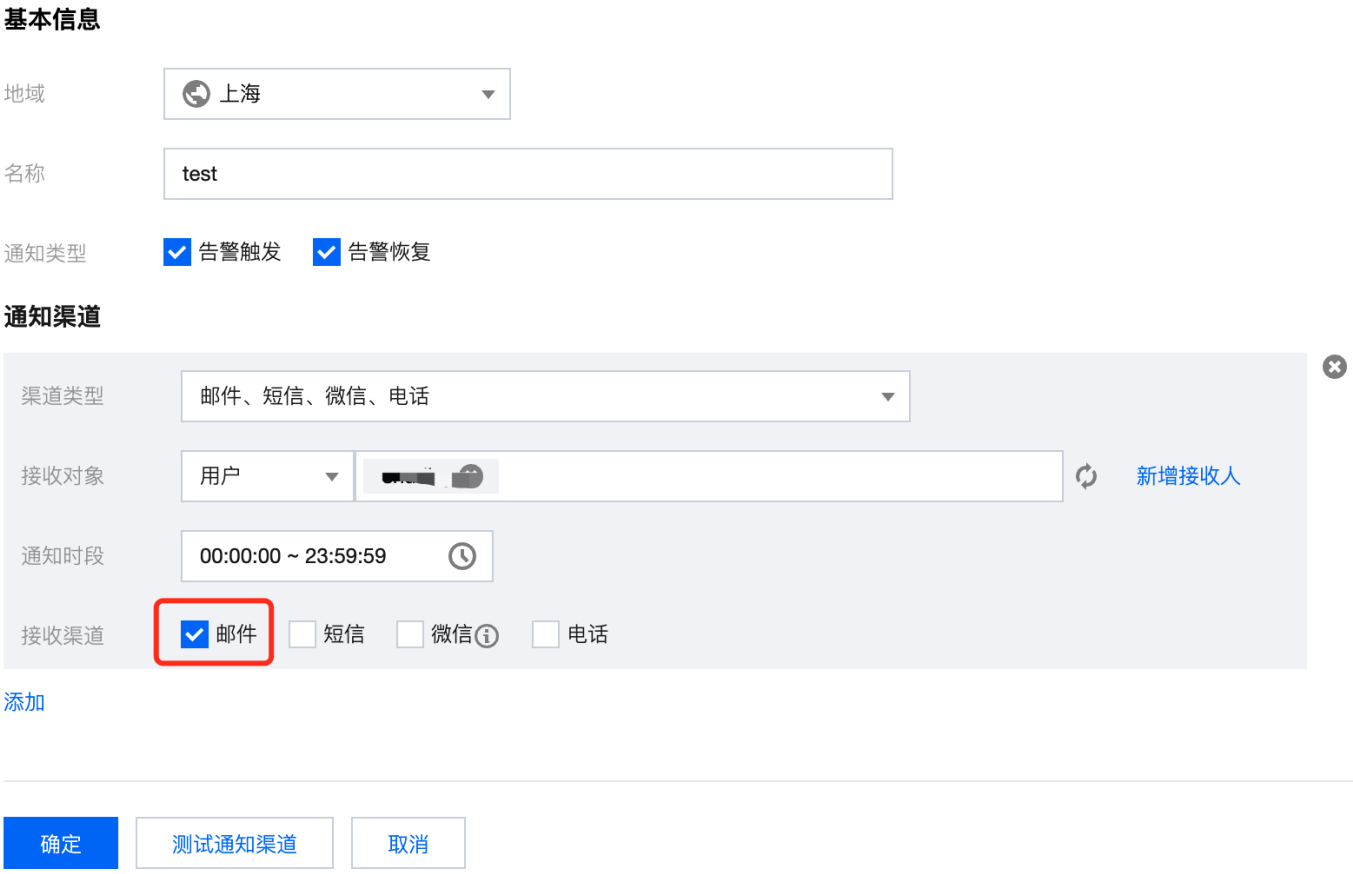
6.在“邮箱”栏中，单击**发送链接验证**。



6. 登录您所填写的邮箱，在“【腾讯云】邮箱接收消息验证”邮件中，单击**确认接收**即可。

选择邮件告警渠道

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**监控告警 > 告警策略**，进入告警策略列表页面。
3. 找到需要启用微信告警的策略，单击策略名称，进入策略编辑页面。
4. 勾选接收组，单击**编辑**。
5. 在弹出的配置框中，勾选**邮件**，单击**确定**，即可启用邮件告警渠道。



微信接收告警通知

最近更新时间：2022-05-09 18:14:46

操作场景

本文将为您介绍如何通过微信接收告警通知。

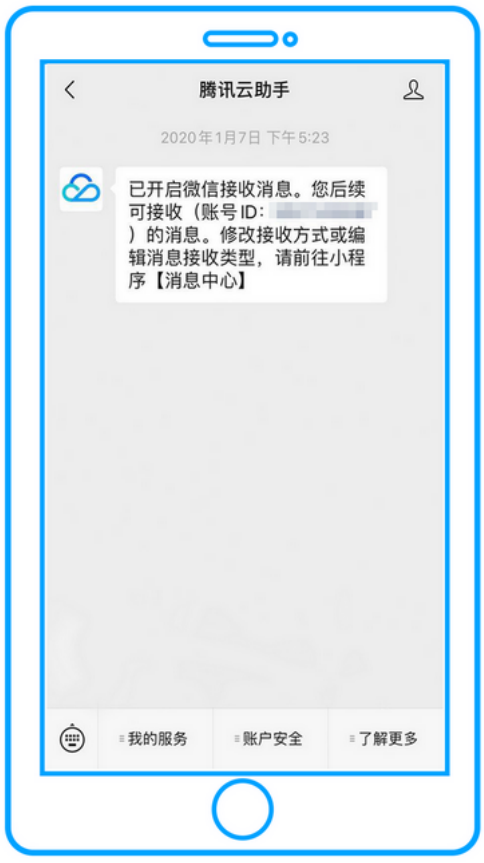
操作步骤

开启微信接收

1. 登录 访问管理控制台。
2. 在左侧导航栏中，单击**用户 > 用户列表**，进入用户列表页面。
3. 找到需要配置接收微信通知的用户，单击用户名称进入用户详情页面。
4. 在“微信”栏中，单击 。

> 注意：
> 企业微信子用户暂不支持该功能，无法通过微信接收告警通知，建议使用 企业微信接收告警通知。

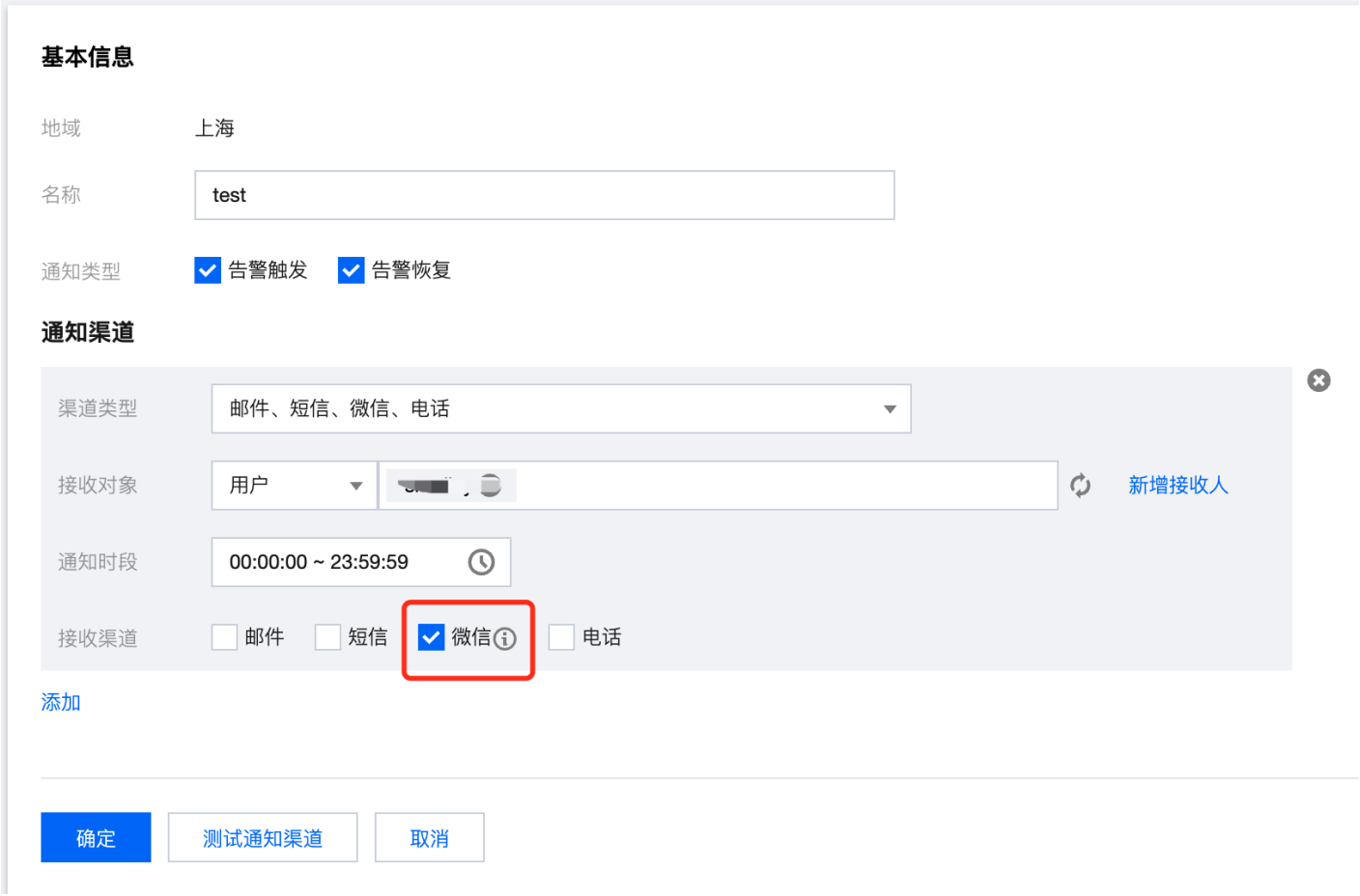
5. 在弹出的窗口中，单击**确定**。
6. 微信扫码并关注“腾讯云助手”公众号，即可成功更换联系微信。
   绑定成功如下图所示：



选择微信告警渠道

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**监控告警 > 告警策略**，进入告警策略列表页面。
3. 找到需要启用微信告警的策略，单击策略名称，进入策略编辑页面。
4. 勾选接收组，单击**编辑**。

5. 在弹出的窗口中，勾选**微信**，单击**保存**，即可启用微信告警渠道。



企业微信接收告警通知

最近更新时间：2021-12-27 09:58:20

操作场景

本文将为您介绍如何通过企业微信接收告警通知。

相关限制

企业微信群消息发送频率限制：每个机器人发送的消息不能超过20条/分钟。若您的告警策略较多，建议多创建几个机器人，分散绑定告警策略。避免多个告警策略在同一时间触发告警时，导致您无法接收部分告警通知。

> 说明：
> 您成功创建企业微信机器人和设置回调地址后，日志服务自动推送告警消息到企业微信机器人。您即可在企业微信群收到告警通知。

操作步骤

在企业微信添加机器人

PC 版

1. 在 PC 版企业微信中，找到需要接收告警通知的企业微信群。
2. 选中并右键单击企业微信群。
3. 在弹框中，单击【添加群机器人】。
4. 在弹框中，单击【新创建一个机器人】。
5. 在弹框中，自定义机器人名称，单击【添加机器人】。
6. 单击【复制地址】，复制 webhook 地址，配置告警接口回调。

Web 版

1. 在企业微信 Web 版中，打开您需要接收告警通知的企业微信群。
2. 单击右上角的群设置图标。
3. 在群设置页面，单击【群机器人】>【添加机器人】。
4. 在添加机器人管理页面，自定义机器人名称。
5. 单击【添加】，复制 webhook 地址，配置告警接口回调。

配置告警接口回调

在日志服务通知渠道中，填写 webhook 地址，单击【确定】即可。

基本信息

地域 上海

名称 [redacted]

通知类型 ☑ 告警触发 ☑ 告警恢复

通知渠道

渠道类型 邮件、短信、微信、电话

接收对象 用户 test-acc 新增接收人

通知时段 10:00:00 ~ 13:10:59

接收渠道 ☑ 邮件 ☑ 短信 ☑ 微信 ☑ 电话

渠道类型 企业微信

请求地址 http://in.qyapi.weixin.qq.com/cgi-bin/webhook/send?key[redacted]-5d37

添加

确定 测试通知渠道 取消

配置成功后，当告警策略被触发时，您可以在企业微信群接收到群机器人发送的告警通知，如下图所示：

自定义回调接收告警通知

最近更新时间：2021-12-17 14:21:59

操作场景

本文将为您介绍如何通过自定义回调接口（webhook）接收告警通知。

相关限制

自定义回调接口消息发送频率限制：每个自定义回调接口发送的消息不能超过20条/分钟。若您的告警策略较多，建议多创建几个自定义回调接口，分散绑定告警策略。避免多个告警策略在同一时间触发告警时，导致您无法接收部分告警通知。

> 说明：
> 您成功创建自定义回调接口并设置回调地址后，日志服务自动根据配置向自定义回调接口发送请求。

操作步骤

为目标服务生成自定义回调链接

根据需要回调的自定义服务（如钉钉，slack 等），生成对应的自定义回调链接（webhook）以接受告警通知。

配置自定义接口回调（webhook-自定义）

在日志服务通知渠道中，填写自定义回调链接的地址，并根据需要配置自定义的请求内容，配置详情请参考 新增通知渠道组。

配置成功后，当告警策略被触发时，日志服务将根据配置的请求格式调用自定义接口。

最佳实践

使用钉钉或飞书接收告警通知

管理通知渠道组

最近更新时间：2022-04-20 17:48:19

操作场景

本文档指导您管理通知渠道组。

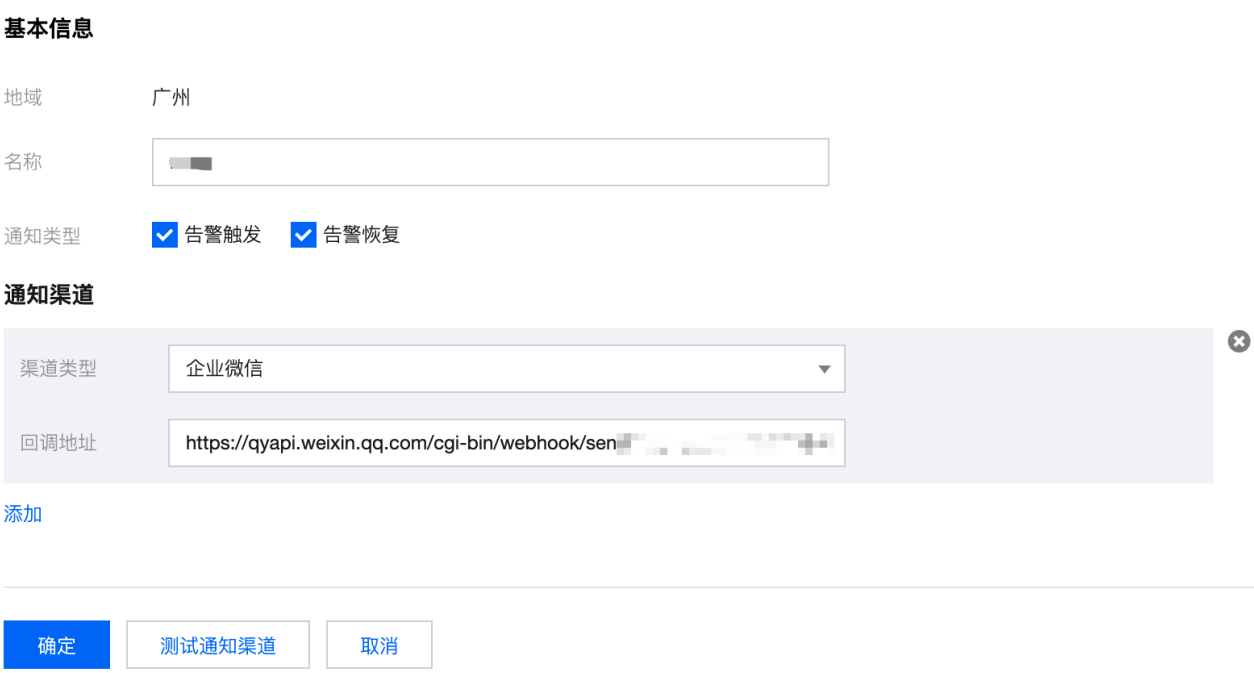
操作步骤

新增通知渠道组

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**监控告警 > 通知渠道组**，进入通知渠道组管理页面。
3. 单击**新建**，并在“新建通知渠道组”中，填写如下信息：
   - 基本信息
     - 渠道组名称：自定义渠道组名称。
     - 通知类型：
       - 触发通知：当监控结果满足告警触发表达式时，则会发送触发通知。
       - 恢复通知：当上一个周期满足触发条件，且当前周期未满足触发条件，且已发送过触发通知，则会发送恢复通知。
   - 通知渠道：配置用户通知，填写用户接收信息。
     - 渠道类型：根据实际需求进行选择。
     - 接收对象：可以对指定用户进行通知触达，也可以对指定用户组（包含多个用户）进行通知触达。
     - 通知时段：定义接收告警时间段。
     - 接收渠道：支持邮箱、短信、微信、电话四种告警渠道。
     - 回调地址：配置接口回调，填写回调地址信息。

       > ⚠ 注意：
       > 接口回调支持自定义 webhook 和 企业微信 webhook，最多支持10个。

       - webhook-企业微信机器人
         创建企业微信机器人后，只需填入回调的 URL 即可。详情请参考 企业微信群接收告警通知。



       - webhook-自定义
         输入自定义的回调地址，接收告警后进行二次处理转发。您可以自定义请求内容，日志服务告警相关的信息可以通过变量的方式进行引用，当告警回调被触发时，

会替换成当前告警策略的对应内容。详情请参考 自定义回调接收告警通知。

复制通知渠道组

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**监控告警 > 通知渠道组**，进入通知渠道组管理页面。
3. 选择需要复制的通知渠道组，单击**复制**。

修改通知渠道组

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**监控告警 > 通知渠道组**，进入通知渠道组管理页面。
3. 选择需要修改的通知渠道组，单击**编辑**。

删除通知渠道组

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**监控告警 > 通知渠道组**，进入通知渠道组管理页面。
3. 选择需要删除的通知渠道组，单击**删除**。

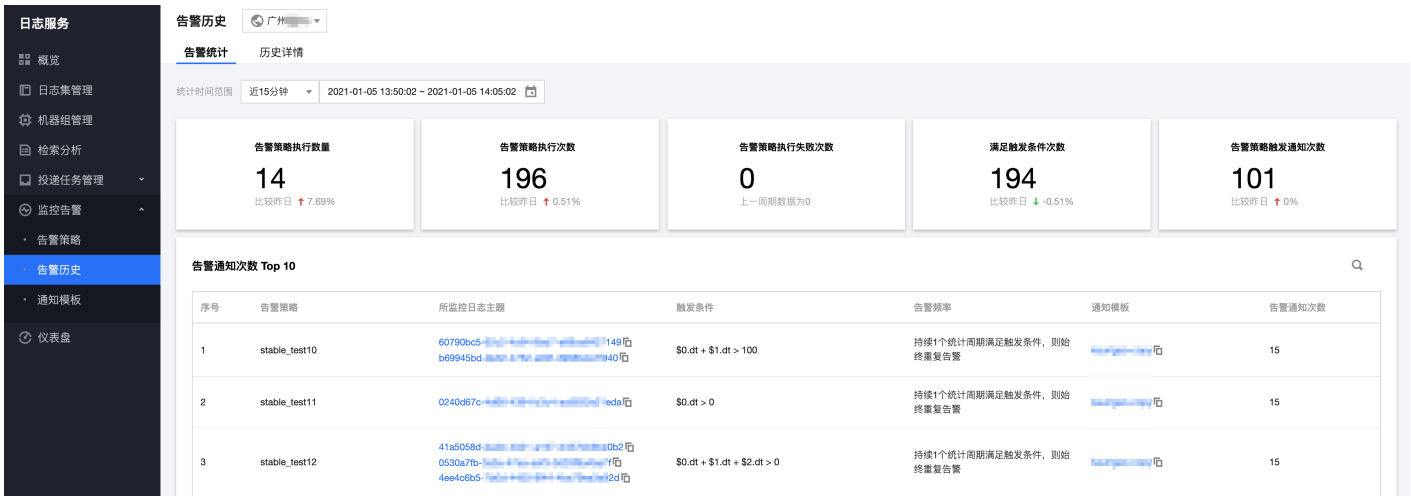
查看告警历史

最近更新时间：2022-04-20 18:32:45

本文将为您介绍如何在日志服务控制台查看告警历史。

操作步骤

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**监控告警 > 告警历史**，进入告警历史查看页面。



相关说明

日志服务提供近30天的告警历史信息查看。

告警统计

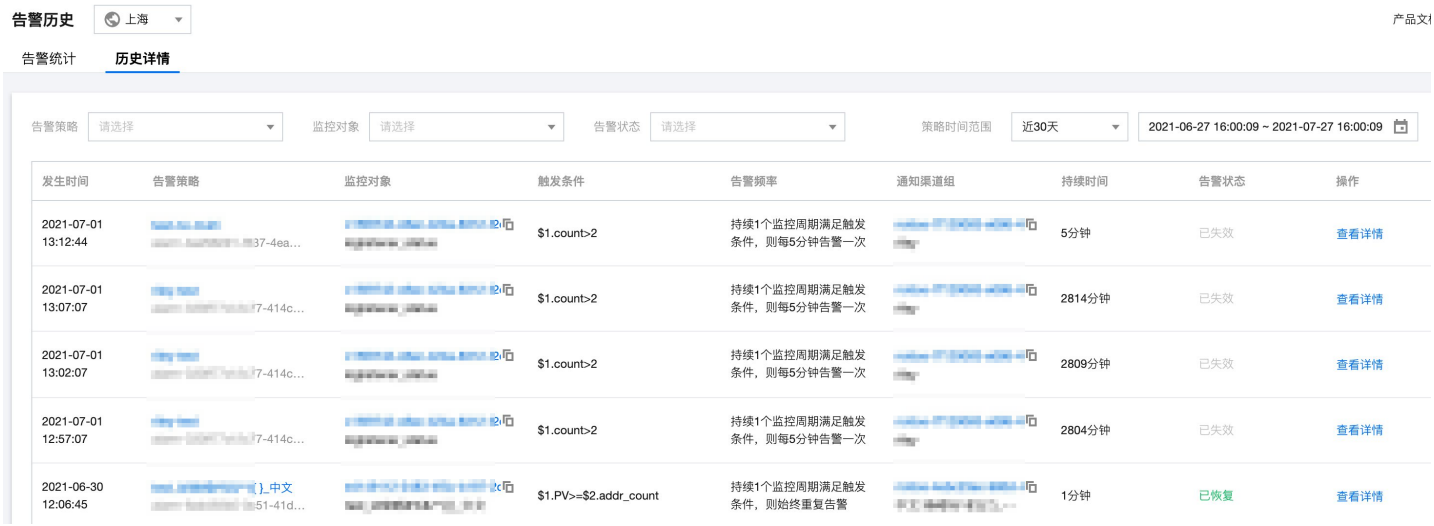
告警统计展示了当前地域下的告警重要信息，包括告警策略统计、监控任务执行情况等，统计指标详细如下：

统计指标	说明
告警策略执行数量	统计时间范围内，执行过的告警策略个数
告警策略执行次数	统计时间范围内，运行告警策略中的查询分析语句的次数
告警策略执行失败次数	统计时间范围内，告警策略执行结果失败的次数，执行失败所包含的错误范围（详情参考 执行结果状态码）：AlarmConfigNotFound、QuerySyntaxError、QueryError、QueryResultParseError、ConditionSyntaxError、ConditionEvaluateError、ConditionValueTypeError
满足触发条件次数	统计时间范围内，告警策略的查询分析语句运行成功，且返回结果满足触发条件的次数
告警策略触发通知次数	统计时间范围内，告警策略执行后触发通知发送的次数
告警通知次数 Top10	统计时间范围内，触发通知发送次数最多的10个告警策略

历史详情

告警策略一旦生效，会周期性执行监控任务，每次监控任务执行的情况会记录在**历史详情**中，主要信息包括每次执行的结果情况，通过查看告警策略执行的记录，方便进行告警任务的历史追溯。

> **说明：**
> 日志服务提供近30天的历史信息查看。



策略执行结果：

执行结果	说明
AlarmConfigNotFound	缺少告警策略配置，请检查告警策略及监控对象配置是否正确；
QuerySyntaxError	监控对象的分析语句有语法错误，请检查语句是否正确，语法参考 分析语句
QueryError	分析语句执行异常，请检查分析语句和日志主题的索引配置
QueryResultParseError	分析结果格式解析失败
ConditionSyntaxError	触发条件表达式有语法错误，请检查 表达式语法 格式
ConditionEvaluateError	触发条件计算错误，请检查所引用的变量是否在分析结果中存在
ConditionValueTypeError	触发条件计算结果非 bool 值，请检查触发条件表达式是否正确
EvalTimesLimited	触发条件计算次数超过1000次，仍不满足触发条件
QueryResultUnmatch	当前监控周期的分析结果不满足告警触发条件
UnreachedThreshold	满足告警触发条件，但未达到告警收敛阈值，不发送告警通知 • HappenThreshold Unreached：未满足周期收敛条件，例如，持续5个监控周期才触发告警 • AlertThreshold Unreached：未满足告警时间间隔条件，例如，每隔15分钟告警一次
TemplateUnmatched	不符合通知模板的告警配置信息，具体原因包含： • TypeUnmatched：不符合通知模板的告警通知类型（告警通知、告警恢复），不发送告警通知 • TimeUnmatched：不符合通知模板的告警通知时间段，不发送告警通知 • SendFail：通知发送失败
Matched	满足告警条件，且成功通知告警

策略告警状态：

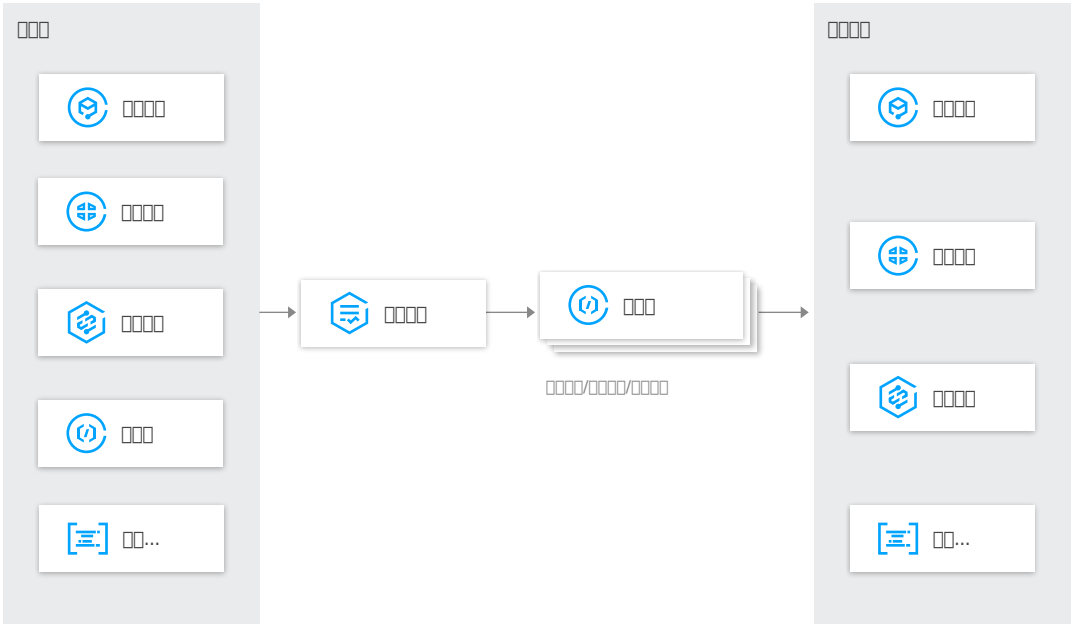
告警状态	说明
未恢复	系统持续满足触发条件达到阈值，触发告警
已恢复	当前监控周期未满足触发条件
已失效	告警策略发生删除或修改

函数处理
函数处理简介

最近更新时间：2022-03-30 14:55:30

通过函数处理服务，可以快速完成云服务器（Cloud Virtual Machine，CVM）等云上资源的运行日志采集、ETL 和消息转储等复杂日志处理任务。函数处理为异步过程，凡是收集到日志服务的数据，都能通过配置将数据投递到云函数进行消费处理，您只需要在日志服务控制台进行简单的配置即可完成日志服务（Cloud Log Service，CLS）对接云函数消费。

整体数据处理流程如下：



通过 CLS 函数触发器将日志源信息提交到 SCF，再通过 serverless 无服务架构的函数计算提供数据处理、分析，事件触发，弹性伸缩，无需运维，按需付费。

使用函数处理优势

- 提供一站式采集、存储、处理、分析和展示
- 全托管日志处理任务，按时间周期进行触发执行，自动重试
- 持续增加内置函数模板，降低主流需求下的日志处理开发成本
- 基于云函数提供数据处理及自定义代码逻辑
- 基于云函数提供计算能力，拥有弹性伸缩，免运维，按需付费等特性

函数处理场景实践

日志服务可以将日志主题中的数据通过 CLS 日志触发器 投递至云函数进行处理，以满足日志处理/日志清洗等应用场景需求，如下：

函数处理场景	描述说明
ETL 日志处理	日志数据通过云函数进行日志清洗，日志处理，格式转换等操作

⚠ 注意：
数据投递至云函数，云函数侧将产生相应的计算费用，计费详情请参见云函数 计费概述。

ETL 日志处理

最近更新时间：2022-04-20 10:42:37

操作场景

本文为您介绍使用 云函数（Serverless Cloud Function，SCF）对日志服务（Cloud Log Service，CLS）日志进行处理。其中，CLS 主要用于日志采集，SCF 主要提供节点计算能力。
数据流程如下：

前提条件

已登录 日志服务控制台。

操作步骤

创建日志主题

参看 新增日志主题 文档，创建两个日志主题。

> 说明：
> ETL 数据处理的源端和终端均为 CLS，故至少需创建两个主题。

创建云函数 SCF

参看 通过模板创建函数 文档，创建函数。
主要配置参数如下：

- 地域：选择**北京**地域。
- **函数名称**：命名为 “CLSdemo”。
- **创建方式**：使用**模板创建**，选择**CLS日志ETL模板**。

> 注意：
> 函数需要在**函数配置**页面中，选择和 CLS 相同的 VPC 和子网。如下图所示：
> 
>

配置 CLS 触发器

1. 登录 日志服务控制台。
2. 在左侧导航栏中，单击**日志主题**，进入日志主题管理页面。
3. 找到刚创建的日志主题，单击日志主题ID/名称，进入该日志主题详情页面。
4. 在日志主题详情页面，选择**函数处理**页签，单击**创建**。

5. 在弹出的“函数处理”窗口中，添加已创完成的函数，单击**确定**。如下图所示：

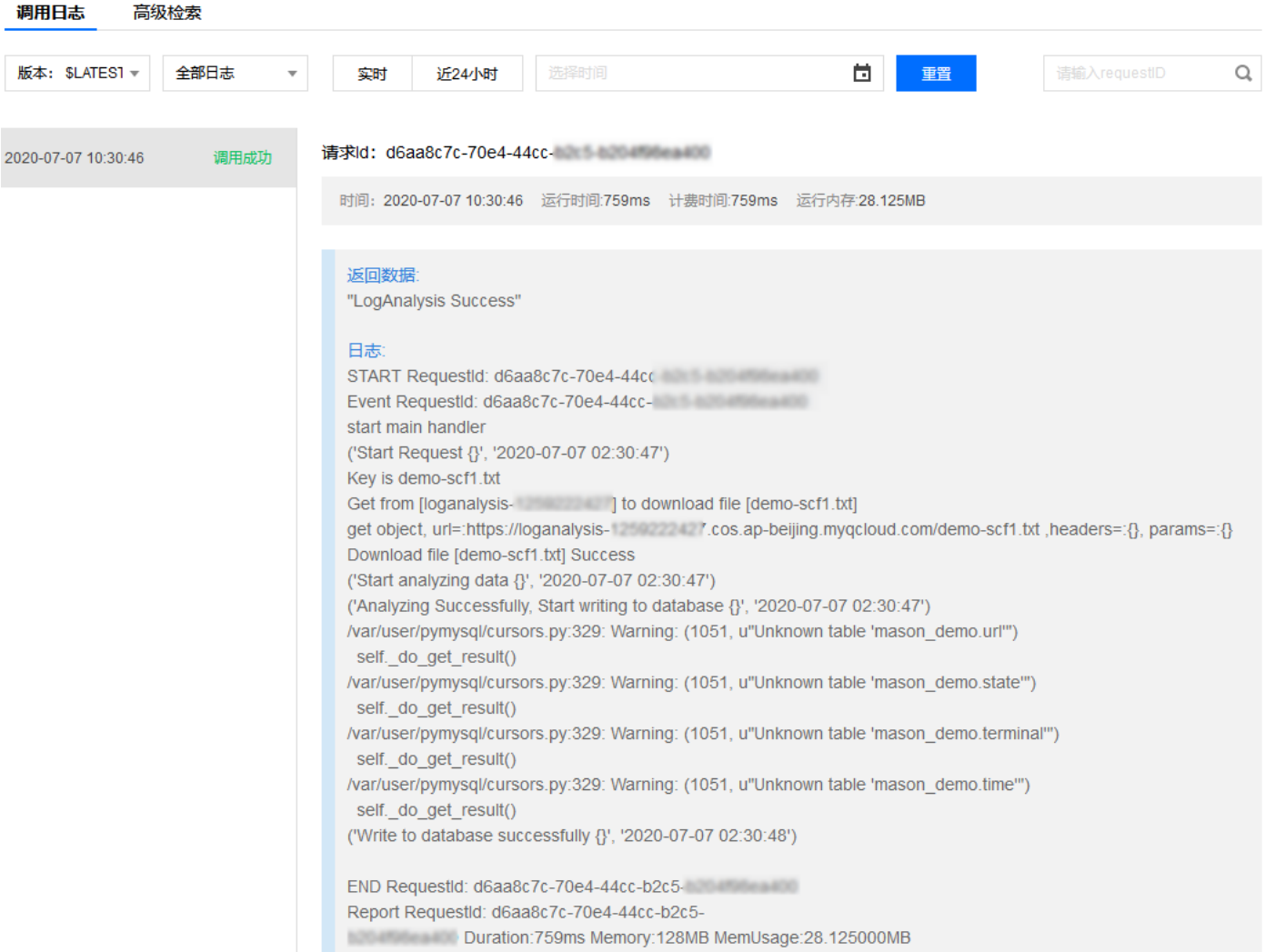
主要参数信息如下，其余配置项请保持默认：

- **命名空间**：选择函数所在的命名空间。
- **函数名**：选择 创建云函数 SCF 步骤中已创建的云函数。
- **别名**：选择函数别名。
- **最长等待时间**：单次事件拉取的最长等待事件，默认60s。

测试函数功能

1. 下载 测试样例 中的日志文件，并解压出 demo-scf1.txt，导入至源端 CLS 服务。
2. 切换至 云函数控制台，查看执行结果。

在函数详情页面中选择**日志查询**页签，可以看到打印出的日志信息。如下图所示：



3. 切换至终端 CLS，查看数据处理结果。

> 说明：
> 您可以根据自身的需求编写具体的数据处理方法。

历史文档

低版本 LogListener 操作指南

最近更新时间：2020-06-29 16:01:13

> ⚠ 注意：
> 本文档提供2.2.4以下版本的 LogListener 操作指南，后续可能不再维护，建议您更新到最新版本，最新版本安装操作详见 LogListener 安装指南。

启动 LogListener

进入 loglistener 安装目录，通过以下脚本启动 LogListener：

```
cd loglistener/tools; ./start.sh
```

停止 LogListener

进入 loglistener 安装目录，通过以下脚本停止 LogListener：

```
cd loglistener/tools; ./stop.sh
```

查看 LogListener 进程状态

进入 loglistener 安装目录，通过以下命令查看 LogListener 进程：

```
cd loglistener/tools; ./p.sh
```

正常情况下，将存在以下三个进程：

```
bin/loglistenerm -d #守护进程
bin/loglistener --conf=etc/loglistener.conf #主进程
bin/loglisteneru -u --conf=etc/loglistener.conf #更新进程
```

卸载 LogListener

进入 loglistener 安装目录，通过以下命令卸载 LogListener：

```
cd loglistener/tools; ./uninstall.sh
```

检查 LogListener 心跳及配置

进入 loglistener 安装目录，通过以下命令卸载 LogListener：

```
cd loglistener/tools; ./check.sh
```

低版本 LogListener 异常状态排查

最近更新时间：2020-06-29 15:54:48

> ⚠ 注意：
> 本文档提供2.2.4以下版本的 LogListener 故障处理排查。最新版本处理排查，请参见 机器组异常 处理文档。

现象描述

日志采集异常，检查关联的机器组，状态异常。

可能原因

机器组与日志服务系统之间的心跳中断，导致日志无法采集上报。导致机器组异常的可能原因有：

1. IP 地址填写错误。
2. 网络断开。
3. LogListener 进程失败。
4. LogListener 配置错误。

解决思路

针对以上可能原因，逐步进行排查。

处理步骤

1. 请检查核对机器组中所添加的 IP 地址是否正确。
   i. 查看 LogListener 获取的 IP 地址，执行命令如下：

```
cd loglistener/tools && ./check.sh
```

```
[root@VM_30_69_centos tools]# ./check.sh
group ip:10.163.30.69
host:ap-chengdu.cls.myqcloud.com
port:80
```

   ii. 登录 日志服务控制台，单击【机器组管理】，查看机器组配置的 IP 地址，此处的 IP 地址必须与采集端获取的地址完全一致。

查看机器组

IP	状态
10.163.30.69	正常

2. 请使用以下命令确认当前网络环境是否连通：

```
telnet <region>.cls.myqcloud.com 80
```

其中，<region> 为日志服务所在地域简称，具体地域信息请参见 可用地域 文档。
正常连通下，将显示如下代码。否则，显示连接失败，需排查网络环境，保证正常连网。

```
[ ]# telnet ap-shanghai.cls.myqcloud.com 80
Trying ...
Connected to ap-shanghai.cls.myqcloud.com.
Escape character is '^]'.
```

3. 请查看确认 LogListener 进程是否正常运行，进入安装目录查看命令如下：

```
cd loglistener/tools && ./p.sh
```

正常情况下，将存在以下三个进程：

```
bin/loglistenerm -d #守护进程
bin/loglistener --conf=etc/loglistener.conf #主进程
bin/loglisteneru -u --conf=etc/loglistener.conf #更新进程
```

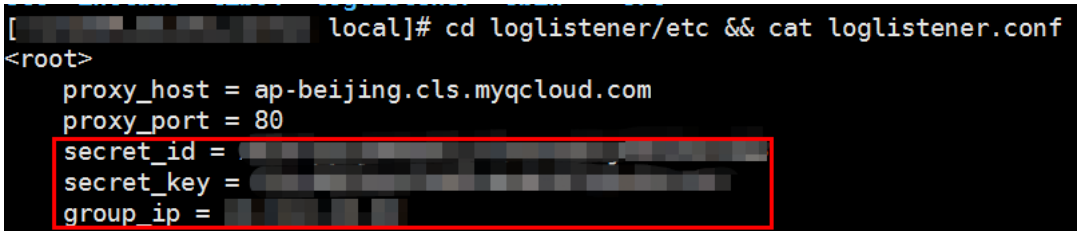
若有失败进程，需要重新启动，进入安装目录执行如下命令：

```
cd loglistener/tools && ./start.sh
```

4. 请检查确认 LogListener 中密钥及 IP 标识配置信息是否有误，进入安装目录查看配置信息，命令如下：

```
cd loglistener/etc && cat loglistener.conf
```

如图所示：



- 密钥为腾讯云账号或协作者的 API 密钥，但不支持项目密钥。
- 配置文件中 group_ip 信息必须与控制台机器组所填写的 IP 一致，LogListener 会自动获取机器 IP，当服务器有多个网卡时需留意检查是否一致。

原 CLS 语法检索语法规则

最近更新时间：2020-12-16 14:49:12

日志服务提供丰富的检索功能，您可以通过查询语法、检索规则实时查询日志，检索结果秒级返回，帮助您快速了解业务情况。

开始检索

日志检索是 CLS 所提供的非常重要的能力，您可以自定义查询条件秒级查询亿级别的日志数据，具体的操作步骤如下：

1. 登录 日志服务控制台。
2. 在控制台左侧导航栏中单击【检索分析】，进入检索页面。
3. 选择查询时间区间，日志集，日志主题。
4. 输入关键词后单击【检索分析】，即可获得查询结果。

查询语法

检索支持以下查询语法：

语法	语义
key:value	键值搜索格式，需要开启配置 键值索引，其中 value 支持?、*模糊搜索
A and B	“与”逻辑，返回 A 与 B 的交集结果，若多个关键词，使用空格分割则默认为 and
A or B	“或”逻辑，返回 A 或 B 的并集结果
not B	“非”逻辑，返回不包含 B 的结果
A not B	“减”逻辑，返回符合 A 但不符合 B 的结果，即 A-B
'a'	字符 a 将被视为普通字符，不会当作语法关键词处理
"A"	A 中的所有关键词都将被视为普通字符，不会当作语法关键词处理
\	转义字符，转义后的字符表示符号本身，例如转义引号\"，转义冒号\:等
*	模糊查询关键字，匹配零个、单个或多个任意字符，不支持开头*，例如：输入abc*，会返回以abc开头的所有命中日志
?	模糊查询关键字，特定位置匹配单个字符。例如，输入ab?c，会返回以ab为开头，以c为结尾字符，且两者之间且有一个字符的所有命中日志
>	大于，针对数值类型的字段
>=	大于等于，针对数值类型的字段
<	小于，针对数值类型的字段
<=	小于等于，针对数值类型的字段
=	等于，可以是数值类型的字段，也可以是文本类型（不支持模糊搜索，若要使用模糊搜索，参考键值搜索 key:value）
__SOURCE__	查询某个 source 源的日志，支持通配符，例如__SOURCE__:127.0.0.*
__FILENAME__	查询来源为某个文件的日志，支持通配符 ，例如__FILENAME__:/var/log/access.*

注意：

- 数值型必须设置成 double 或 long 类型后，才能进行范围检索，否则返回的结果会与预期不符。
- 若 b 是文本，a=b与a：b的区别在于前者是 a 全等于 b，后者是 a 包含 b（按分词逻辑处理，支持模糊搜索）。

检索规则

在进行日志检索时，检索时间范围、日志集、日志主题是三个不可缺少的查询条件。默认检索时间为当天有效日志数据，而日志集与日志主题需要您根据查询需求进行选择。搜索框内容允许为空，若为空，即视为无筛选条件，系统将返回所有有效的日志数据。

全文检索

对于开启全文索引的日志主题，系统会根据分词符将每条日志数据拆分为多个词组，以支持您根据特定的关键字来检索日志。

例如，想查询包含 error 关键字的日志数据，在搜索框里输入 error 进行查询即可。

键值检索

对于开启键值索引的日志主题，系统根据所指定的 kv 键值对进行日志数据管理，您可以通过指定特定的 key 字段来进行日志检索。

例如，想查询 status 字段为 error 的日志数据，在搜索框里输入 status:error 进行查询即可。

模糊关键字检索

日志服务提供模糊查询的能力，通过特殊的模糊关键字进行日志检索，具体说明如下：

元字符	描述
*	模糊查询关键字，匹配零个、单个或多个任意字符。例如，输入abc*，会返回以abc开头的所有命中日志
?	模糊查询关键字，特定位置匹配单个字符。例如，输入ab?c，会返回以ab为开头，以c为结尾字符，且两者之间且有一个字符的所有命中日志

注意事项

1. 在进行日志检索时请确认相应的日志主题已开启检索，否则将无法检索到有效日志。
2. 在进行键值检索时，请确认所查询的索引字段在日志主题的索引配置中已成功配置。
3. 日志检索结果默认为倒序排列。