目录

概览

简介

概览通过展示一些关键指标项，让用户快速了解到当前账户下数据读写、存储、查询等统计指标。 查看写入流量、读取流量、存储量、写入次数、查询次数等指标的昨天实际数值、以及日环比（日环比指昨天跟前天比，指标数据的变化情况）。 按工程和日志池查看读取次数、写入次数和存储量。

工程

简介

工程是基本业务组织单元，用户可以为不同业务在指定Region下创建不同的工程。每个工程都包含日志池、日志搜索、仪表盘、监控告警等模块。 工程组成员具有在该工程下创建日志池、写入读取数据、执行日志查询等权限。

创建工程

1. 登录金山云日志服务Klog控制台。
2. 点击导航**工程列表**，进入到**工程列表页**。
3. 在工程列表中点击**创建工程**，进入到创建工程页面。
   - 工程名称：按照格式要求自定义工程名称，名称创建后不支持修改
   - 地域：选择工程所属地域，选择后不支持修改
   - 备注：填写工程的描述信息，保存后支持修改 点击**确定**，完成工程创建。

工程详情

1. 登录金山云日志服务Klog控制台。
2. 点击导航**工程列表**，进入到**工程列表页**。
3. 在工程列表中，点击工程名称，进入到该**工程详情页**。
   - 访问域名：产品提供内网和外网两种访问域名，外网访问会产生外网流量。
   - 日志池：当前工程下的日志池，日志池相关操作详见日志池。

日志池

简介

日志池是日志服务的最小存储单元。日志服务支持用户自定义创建日志池以及定义日志池的分区数（分区数范围2-64）、日志数据存储周期（1-3000天），并允许用户根据业务实际数据流量来调整日志池的分区数。日志服务会自动删除超出存储周期的日志数据。日志池支持存储TEXT、LONG、DOUBLE、DATE等数据类型。

创建日志池

1. 登录金山云日志服务Klog控制台。
2. 在控制台点击**项目列表** > **项目详情页** > **概览**，点击**创建日志池**进入到创建页面。
   - 名称：按照格式要求自定义日志池名称，名称保存后不支持修改
   - 存储周期：范围支持1-3000天，保存后支持修改
   - 分区数：范围支持2-64天，保存后支持修改 点击“确定”完成日志池创建。

编辑日志池

1. 登录金山云日志服务Klog控制台。
2. 在控制台点击**工程列表** > **工程详情页** > **概览**，选中某一日志池，点击**编辑**进入到编辑页面。
   - 存储周期：修改存储周期后，存储周期会在第二天生效，比如今天把存储周期从6天调整为2天后，超出存储周期的4天数据不会被立即删除，会在第二天对超出存储周期范围的数据做删除操作。
   - 分区数：修改分区数后会立即生效。

删除日志池

删除日志池后，日志池中的所有日志数据以及日志池相关联的告警、图表会被一起清除，且不可恢复，请谨慎操作。

klog-filebeat

klog-filebeat介绍

Filebeat是由Elastic开发的一款开源日志采集软件，使用者可以将其部署到需要采集日志的机器上对日志进行采集，并输出到指定的日志接收端如elasticsearch、kafka、logstash等等。KLog团队对开源Filebeat进行了二次开发并提供新增特性，我们称之klog-filebeat，其新增特性如下：

- 支持输出日志到klog：
  - 支持输出到多个工程和日志池
  - 可为每个日志池配置过滤器，让只有符合条件的日志条目输出到对应的日志池
  - 可选择仅输出部分字段
  - 支持动态加载access_key、secret_key
- 支持通过grok方式解析日志，将普通文本解析为json对象。

关于Filebeat自身的详细特性，可以参考filebeat官方文档。

前提条件

1. 已创建工程和日志池。更多信息，请参见工程和日志池。
2. 已开通服务器的80端口和443端口。
3. 已完成Klog-Filebeat环境配置。

下载

- 最新Linux版本的klog-filebeat可以点击此处下载。
- 或直接在您的服务器上下载：

```
wget "https://ks3-cn-beijing.ksyun.com/klog/filebeat/klog-filebeat.latest.tar.gz"
```

安装

安装比较简单，解压缩到您需要的路径即可：

```
tar xvf klog-filebeat.latest.tar.gz
```

运行

执行如下命令，运行klog-filebeat。

```
./filebeat -e
```

采集器配置

klog-filebeat需要2个配置文件，分别是filebeat.yml和credential.ini，这两个配置文件均在klog-filebeat压缩包中，解压后即可看到。

credential.ini

这个配置文件的内容是您的金山云access_key和secret_key。内容示例：

```
[klog]
access_key = AKLT6-bcdesfg-ajsIjfiejI9
secret_key = Jf2390j9E9finfiDIOFJFD8483+dfsDVdOCTFazCnDEAw+mxA/7Lfeh3ugErwoKKb5wNOIei==
```

filebeat.yml

这个是主配置文件，内容示例如下：

```
filebeat.inputs:
- type: log
  enabled: true
  paths:
    # 需要采集的日志文件
    - /var/log/*.log

output.klog:
  endpoint: https://klog-cn-beijing.ksyun.com
  credential:
    path: credential.ini
    check_interval: 60
  targets:
    - project_name: yourProjectName
      pool_name: yourPoolName
```

```
      fields:
        - key: message
```

关于output.klog的详细配置说明可参考下面的yml文件完整配置。其余详细配置可参考filebeat官方文档。

> 注意：在输出到klog的情况下，filebeat的modules功能无法使用。 内网endpoint：http://klog-cn-beijing-internal.ksyun.com

yml文件完整配置

文件完整示例以及配置项含义分别如下：

```
# ------------------------------ Filebeat Input ------------------------------
filebeat.inputs:
- type: log
  enabled: true

  # 需要采集的日志文件
  paths:
    - /your/app/log/path.log

  # 如果您的日志是以json形式打印的，则可设为true或false，这样filebeat会自动解析json内容。
  # 设置为false时，解析出来的map对象存在于filebeat事件对象的名为json的字段中。
  # 设置为true时，解析出来的map对象的各字段直接存在于filebeat事件对象中。
  # 删除这个配置项时，日志作为普通文本，存在于filebeat事件对象的message字段中。
  json.keys_under_root: false

# ------------------------------ KLog Output -------------------------------

# klog-filebeat特有配置。表示将采集到的日志发送到klog服务端。
output.klog:
  # klog服务的日志接收地址。如果使用内网域名，请使用http协议
  endpoint: https://klog-cn-beijing.ksyun.com

  # 最大批量条数，默认为2048
  bulk_max_size: 2048

  # 用于发送的协程数，默认为1
  worker_num: 1

  # 发送时使用的压缩传输方式，默认为lz4。取消压缩写none
  compress_method: lz4

  # 从哪个文件读取您在金山云的 access_key 和 secret_key
  credential:
    # 文件路径
    path: credential.ini

    # 每隔多少秒检查一次ini文件的修改时间。如果发生改变，则重新加载ini
    check_interval: 60

  # 发送到指定的klog工程和日志池。每个target代表一对工程和日志池。
  # 可以设置多个target, 各target之间支持重复设置工程和日志池。
  targets:
      # 目标工程名称
    - project_name: yourProjectName

      # 使用字段yourField1的值作为目标工程名称。
      # project_name和project_name_from_field可以只配置其中一项。如果两项同时配置，则优先使用project_name_from_field。
      # project_name_from_field: yourField1

      # 目标日志池名称
      pool_name: yourPoolName

      # 使用字段yourField2的值作为目标日志池名称。
      # pool_name和pool_name_from_field可以只配置其中一项。如果两项同时配置，则优先使用pool_name_from_field。
      # pool_name_from_field: yourField2

      # 需要发送的段，可以设置多个，不设置则为发送事件的所有字段。
      fields:

        # 需要发送的字段名。字段名里面不可有"."
        - key: message

          # 该字段是map类型时，如果设置为true，则表示只发送它的后代字段。默认为false。
          skip_root: false

      # 过滤器。可以设置多个。匹配所有过滤器的日志条目将会发送到该target。未配置filters的，会全部发送。
      filters:
          # 对哪个字段过滤
        - key: message
```

```
          # 阈值。支持字符串和数值。所有数值在内部都转换成float64
          value: ""

          # 比较符。支持的计算有：>, >=, <, <=, ==, !=, exists, not_exists, contains（仅字符串）, not_contain（仅字符串）
          operator: "!="

# ------------------------------ Klog新增的处理器 -------------------------
  processors:

  #grok处理器可以把普通文本日志解析为json对象。
  #由于grok是基于正则表达式的，所以开启该选项会增加filebeat的资源消耗。
  - grok:
      # 解析哪个字段
      source_field: message

      # 解析结果保存到哪个字段
      target_field: yourField3

      # 用于解析的表达式
      pattern: "%{IPORHOST:client} %{WORD:method} %{URIPATHPARAM:request} %{INT:size} %{NUMBER:duration}"

      # 用于解析的自定义表达式，不是必填的
      patterns:
        MYPATTERN: "\\d+"
```

- 说明：数据解析时，如果存在需要跳过的字符，对字符做解析后不设置字段名称即可，具体示例如下。 原始日志数据内容如下：

  ```
  127.0.0.1 - - [26/Mar/2020:19:09:19 -0400] "GET / HTTP/1.1" 401 - "" "Mozilla/5.0 Gecko" "-"
  ```

  数据解析表达式如下：

  ```
  pattern: "%{IPORHOST}\s-\s-\s\[%{HTTPDATE:timestamp}\]"
  ```

  通过如上表达式，只解析出timestamp字段对应的内容”26/Mar/2020:19:09:19 -0400“。

filebeat事件

filebeat事件是原生Filebeat数据处理过程中的重要概念。

Filebeat采集到的每条日志，在Filebeat内部都表示为一条事件数据。 Filebeat对日志数据字段的读取、转换和新增都是对事件字段的操作。一条简单的事件数据如下：

```
{
    "@timestamp": "2020-09-21T03:08:07.682Z",
    "@metadata": {
        "beat": "filebeat",
        "type": "_doc",
        "version": "7.9.1"
    },
    "log": {
        "offset": 423019,
        "file": {
            "path": "/path/to/your.log"
        }
    },
    "message": "2020-09-21 03:08:07.682 [INFO][47] ipsets.go 304: Finished resync family=\"inet\" numInconsistenciesFound=0 resync
Duration=2.18885ms",
    "input": {
        "type": "log"
    },
    "ecs": {
        "version": "1.5.0"
    },
    "host": {
        "name": "filebeat-cwssh"
    },
    "agent": {
        "name": "filebeat-cwssh",
        "type": "filebeat",
        "version": "7.9.1",
        "hostname": "filebeat-cwssh",
        "ephemeral_id": "decf7010-9f78-41f7-85b6-7a0cc5ba4115",
        "id": "7b2bc79a-9da8-4038-ba55-993af4d9ac71"
    }
}
```

filebeat使用字段massage保存日志原始内容。 在配置filebeat.yml时，可以使用诸如"message"、"log.file.path"、"host.name"、"json.field1"等形式表示事件的字段。

API上传数据

简介

日志服务KLog产品为用户提供数据上传的API，API文档详见Putlogs。接下来介绍如何把原始日志数据序列化成如下格式的Protocol Buffer数据流，然后才能通过API写入服务端。

步骤一、定义Protocol格式

按照如下格式，生成一个klog.proto文件

```
syntax = "proto3";
package klog;

message Log
{
    message Content
    {
        required string key   = 1; // 每组字段的 key
        required string value = 2; // 每组字段的 value
    }
    required int64 time = 1; // 时间戳，UNIX时间格式
    repeated Content contents = 2; // 一条日志里的多个kv组合
}
message LogGroup
{
    repeated Log    logs        = 1; // 多条日志合成的日志数组
    optional string reserved    = 2; // 目前暂无效用
    optional string filename    = 3; // 日志文件名
    optional string source      = 4; // 日志来源，一般使用机器IP
}
```

- 说明：关于Protocol Buffer格式的更多信息请参见Github首页。

步骤二、编译Protocol Buffers

根据上面的内容，定义数据的报文格式后，运行Protocol Buffer编译器，把上面的klog.proto文件编译成特定语言的类。这些类提供了简单的方法访问每个字段，像是访问类的方法一样将结构序列化或反序列化。

- 说明：如需安装编译器，可前往Protobuf版本页选择版本和语言。

这里使用proto编辑器生成Java语言的文件，在klog.proto文件的同一目录下，执行如下编辑命令

```
protoc.exe --java_out=./ klog.proto
```

- 说明：--java_out=./表示编译成 Java 格式并输出当前目录下，./klog.proto表示位于当前目录下的 klog.proto 描述文件。

编辑成功后，输出对应语言的代码文件，这里会生成klog.java文件。

步骤三、调用

将生成的klog.Java和klog.proto文件复制到工程目录下参与编译就可以。

SDK采集

简介

日志服务提供了多个语言版本的SDK，您可以根据业务需要来选择。提供如下能力：

- 对数据接入接口进行了统一封装，降低了上传日志的难度。
- 实现日志的 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。
- 实现API中定义的压缩方法，您不用关心压缩实现的细节。
- 提供统一的异步发送、资源控制、自动重试、优雅关闭等功能。

SDK列表

SDK 语言	GitHub地址
Python	klog-python-sdk

Go	klog-go-sdk
Java	klog-java-sdk

Web Tracking

日志服务KLog支持通过Web Tracking采集HTML、H5、IOS、Android和小程序等客户端的日志。

操作步骤：

一、开通Web Tracking功能 打开创建/编辑日志池页面，开启Web Tracking配置项的开关。

二、写入日志

1、通过 HTTP GET 请求上传日志

```
curl --request GET 'http://${host}/${PoolId}/track?ApiVersion=1.0.0&key1=val1&key2=val2'
```

参数	是否必填	说明
${host}	是	日志服务所在地域的Endpoint。请在工程详情页查看
${PoolId}	是	日志池id,请在工程详情页查看
key1=val1&key2=val2	是	您要上传到日志服务的键值对（Key-Value），支持设置多个键值对，由&隔开。请确保长度小于16KB

2、通过 HTML img 标签上传日志

```
<img src='http://${host}/${PoolId}/track.gif?ApiVersion=1.0.0&key1=val1&key2=val2'/>
<img src='http://${host}/${PoolId}/track_ua.gif?ApiVersion=1.0.0&key1=val1&key2=val2'/>
```

说明：track_ua.gif除了上传自定义的参数外，还会将HTTP头中的UserAgent、referer也作为日志中的字段。

3、通过 HTTP POST 请求上传日志

GET 请求上传日志时，单个请求只能写入一条日志。如果请求的数据量比较大，可以使用 POST 方法上传数据。 请求语法：

```
POST http://${host}/${PoolId}/track HTTP/1.1
```

说明：如果需要对数据压缩发送，请求头为：x-klog-compress-type:lz4或x-klog-compress-type:gzip

示例：

```
POST /a0f2d00e-7fa5-48b4-9c7a-69a9904d550b/track HTTP/1.1
Host:klog-cn-beijing.ksyun.com
Content-Type:application/json
[
   {
    "k1":"v1",
    "k2":"v2"
   },
    {
    "k1":"v1",
    "k2":"v2"
    }
]
```

简介

数据加工提供对日志内容的实时加工处理，包括日志清洗、过滤、脱敏、富化、分发等。支持源日志池到目标日志池一对一和一对多的情况。

加工流程

步骤1，对源日志池的数据进行读取。 步骤2，通过设定的加工规则对读取到的每一条数据进行加工处理。 步骤3，将加工后的数据写入目标日志池。 步骤4，在目标日志池中查看加工后的结果数据。

费用说明

数据加工会消耗机器与网络资源产生费用，计费为0.15元/GB/日。 如果业务仅需要使用加工后的日志，建议缩短源日志池保存时间，不开启源日志池索引，可节省费用。

创建加工任务

本文介绍如何通过日志服务控制台创建数据加工任务。

前提条件 1、已开通日志服务，创建源日志池、并成功采集日志数据。 2、已创建目标日志池。建议目标日志池为空，便于写入加工好的数据。 3、确保当前操作账号拥有配置数据加工任务的权限。

操作步骤 1、 登录日志服务Klog控制台。 2、从控制台进入工程列表，点击工程名称，进入该工程详情页，在左侧导航栏，点击**数据加工**，进入列表页。 3、点击**新建任务**。 4、在基本信息栏目下，填写相关配置项：

配置项	说明
任务名称	数据加工任务的名称
源日志池	作为任务的输入，选择1个需要加工的日志池
目标名称	目标日志池的别名，一是提高目标日志池的业务可读性，二是在加工函数 output("别名")中使用
目标工程	作为任务的输出，选择目标工程
目标日志池	作为任务的输出，选择目标日志池

说明：

- 数据加工支持一对一和一对多的场景，可添加多个目标。在output函数中需指定目标名称，指定工程和日志池无效。
- 数据加工只能处理实时日志流，不能处理历史日志。

5、 在编辑**加工语句**栏目，进行DSL函数加工语句编写和调试，平台提供两种调试数据：

- 原始日志,在基本信息，选择源日志池后，系统会自动加载近15分钟的100条日志数据，时间可按需筛选。
- 测试数据，如果现有日志无法满足调试需求，您可自定义测试数据，必须为 JSON 格式。也可在原始日志页签，单击加入测试数据，将日志添加到测试数据，根据需要修改后，作为您的调试数据。

6、完成 DSL 加工语句的编写后，单击执行预览，运行和调试 DSL 函数。运行结果会在加工结果页签展示，您可以针对运行结果，调整 DSL 语句，直到满足您的需求。 7、调试符合预期后，单击保存，保存数据加工任务，或单击保存并启动，提交并开始执行任务。

管理加工任务

本文介绍如何在日志服务控制台上管理数据加工任务，包括查看任务详情与状态，修改、启动、停止和删除任务等操作。

查看任务详情 1、登录日志服务Klog控制台。 2、从控制台进入**工程列表**，点击工程名称，进入该工程详情页，在左侧导航栏，点击**数据加工**，进入列表页。 3、点击任务名称，进入该任务详情页。

在详情页签，可以查看到该加工任务的基本配置、源日志池、目标日志池以及加工语句。 此外，您可单击目标日志池，快速跳转到该日志池的查询页面。 说明:任务状态包括待启动、运行中、启动中、停止中、已停止、失败。

在监控页签，您可查看该任务近30天总览指标，包括日志数据输入行数、输出行数、失败行数、过滤行数。异常详情中默认展示最近200条加工失败日志的详情，更多日志详情到当前工程下internal-etl-log日志池中查看。

加工函数总览

流程控制函数

函数名称	说明
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
if_else	基于条件判断，分别进行不同的函数处理
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
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

行处理函数

函数名称	说明
drop	丢弃符合条件的日志
keep	保留符合条件的日志
split	基于日志字段的值分裂出多条日志，并且支持通过JMES提取字段后再进行分裂
output	输出日志到指定日志池

[字段处理函数]

函数名称	说明
v	获取字段值，返回对应字符串
set	添加新字段或为现有字段设置新的字段值
drop_fields	根据字段名进行匹配，丢弃匹配到的字段
keep_fields	根据字段名进行匹配，保留匹配到的字段
pack_fields	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
rename	字段重命名
not_has_field	字段不存在时，返回 True，否则返回 False
has_field	字段存在时，返回 True，否则返回 False

[字段值提取函数]

函数名称	说明
kv_delimit	基于分隔符提取字段值内容
json	提取 JSON 字符串格式的字段值，(JSON展开、JMES提取或者JMES提取后再展开)
kv	基于两级分割符提取字段值
regex	基于正则表达式提取字段值

[值结构化处理函数]

函数名称	说明
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
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
json_parse	将值解析为json对象

[正则表达式函数]

函数名称	说明
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_match	基于正则对数据进行匹配
regex_split	将一个字符串分割成字符串数组
regex_replace	基于正则匹配并替换（部分匹配）
regex_findall	根据正则表达式获得符合条件的所有值的一个列表

[日期时间函数]

函数名称	说明
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_to_timestamp	将日期时间对象转换为Unix时间戳
dt_from_timestamp	将Unix时间戳转换为日期时间对象
dt_now	获取当前日期时间

[字符串处理函数]

函数名称	说明
str_count	在值中指定范围内查找子串，返回子串出现的次数
str_uppercase	将字符串中所有小写字符转换为大写字符
str_lowercase	将字符串中所有大写字符转换为小写字符

str_join	使用拼接字符串，拼接多值
str_replace	替换字符串，返回替换结果字符串
str_format	格式化字符串，返回格式化结果
str_strip	删除字符串中指定的字符
str_lstrip	删除字符串开头的指定字符
str_rstrip	删除字符串结尾的指定字符
str_find	在值中查找子串，并返回子串出现的位置
str_start_with	判断字符串是否以指定字符串开头
str_end_with	判断字符串是否以指定字符串结尾

[逻辑表达式]

函数名称	说明
op_if	根据条件判断，返回相应的值
op_and	对值进行 and 运算，均为 True 时，返回 True，否则返回 False
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_not	对值进行 not 运算
op_eq	对值进行比较，相等则返回 True
op_ne	比较两个指定参数的值，不相等时返回true，当两个参数类型不同，返回true
op_ge	对值进行比较，值1大于或等于值2时返回 True
op_gt	对值进行比较，值1大于值2时返回 True
op_le	对值进行比较，值1小于或等于值2时返回 True
op_lt	对值进行比较，值1小于值2时返回 True
op_add	对值进行求和运算
op_sub	对值进行求差运算
op_mul	对值进行乘积运算
op_div	对值进行除法运算
op_sum	对多值累加求和
op_mod	对值进行模计算

[类型转换函数]

函数名称	说明
ct_int	将字段或表达式的值转换为整数
ct_float	将值转换为浮点型数值
ct_str	将值转换为字符串
ct_bool	将值转换为布尔值

[IP解析函数]

函数名称	说明
geo_parse	解析出函数的地理位置

[编码解码函数]

函数名称	说明
url_decoding	对URL类型数据进行URL解码

编码解码函数

url_decoding函数

函数介绍

对URL类型数据进行URL解码。

语法描述

```
url_decoding(value, plus=false)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	string	是	需要解码的值	
plus	bool	否	是否将加号转换为空格，true为将加号转换为空格。默认为false	

示例1

原始日志

```
{"content":"https://www.ksyun.com?w=%E6%B5%8B%E8%AF%95"}
```

加工规则

```
set("url",url_decoding(v("content")))
```

加工结果

```
{"content":"https://www.ksyun.com?w=%E6%B5%8B%E8%AF%95","url":"https://www.ksyun.com?w=测试"}
```

类型转换函数

ct_int函数

函数介绍

使用ct_int函数将表达式的值转换为整数。

语法描述

```
ct_int(value, base=10)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	数字或数字字符串	是	待转换的值。	
base	Number	否	参数值所代表的进制，默认为十进制。例如base=8，表示将八进制要转成十进制。	

示例1

将字符串转换成整型。

原始日志

```
{
"num":"2"
}
```

加工规则

```
set("int_number", ct_int(v("number")))
```

加工结果

```
{
int_number:0
num:"2"
}
```

示例2

将十六进制转换成十进制。

原始日志

```
{"number":"AB"}
```

加工规则

```
set("int_number", ct_int(v("number"),base=16))
```

加工结果

```
{
int_number:171
number:"AB"
}
```

ct_float函数

函数介绍

使用ct_float函数将表达式的值转换为浮点数。

语法描述

```
ct_float(value)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	数字或数字字符串	是		待转换的值。

示例

将字符串转换成整型。

原始日志

```
{"price":"2"}
```

加工规则

```
set("price_float", ct_float(v("price")))
```

加工结果

```
{
price:"2"
price_float:2
}
```

ct_str函数

函数介绍

使用ct_str函数将字段或表达式的值转换为字符串。

语法描述

```
ct_str(value)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是		待转换的值。

示例

将字符串转换成整型。

原始日志

```
{"field": 123}
```

加工规则

```
set("field", ct_str(v("field")))
```

加工结果

```
{"field":"123"}
```

ct_bool函数

函数介绍

使用ct_bool函数将字段或表达式值转换为布尔值。对于不同类型的值返回真假的策略请参见真假判断。

语法描述

```
ct_bool(value)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是		待转换的值。

示例

将字符串转换成整型。

原始日志

```
{
"num":"2"
}
```

加工规则

```
set("ct_bool", ct_bool(v("num")))
```

加工结果

```
{
  "ct_bool": true,
  "num": "2"
}
```

字段值提取函数

kv_delimit函数

函数介绍

通过分隔符提取目标字段中的键值对

语法描述

```
kv_delimit(key1,key2..., pair_sep="\s", kv_sep="=", prefix="", suffix="", mode="fill-auto")
```

参数说明	参数名称	参数类型	是否必填	参数描述
key	string	是	指定的字段名称	
pair_sep	string	否	用于分隔键值对的正则字符集，注意是字符集非正则表达式，如：abc会分别匹配a	b c。默认是\s
kv_sep	string	否	用于分隔键和值的字符串。默认=	
prefix	string	否	给提取的字段名称添加前缀。	
suffix	string	否	给提取的字段名称添加后缀。	
mode	string	否	字段覆盖模式，默认fill-auto，详见文末字段覆盖模式类型	

示例1

从content字段提取键值对

原始日志

```
{"content":"k1=v1 k2=v2"}
```

加工规则

```
kv_delimit("content")
```

加工结果

```
{"content":"k1=v1 k2=v2","k1":"v1","k2":"v2"}
```

示例2

从content字段提取键值对，并为提取字段添加前后缀。

原始日志

```
{"content":"k1=v1 k2=v2"}
```

加工规则

```
kv_delimit("content",prefix="p_",suffix="_s")
```

加工结果

```
{"content":"k1=v1 k2=v2","p_k1_s":"v1","p_k2_s":"v2"}
```

示例3

根据自定义分隔符，从content字段提取键值对。

原始日志

```
{"content":"k1:v1+k2:v2?k3:v3*k4:v4/k5:v5\\k6:v6.k7:v7  k8:v8"}
```

加工规则

```
kv_delimit("content",pair_sep="?+\-*/\s.",kv_sep=":")
```

加工结果

```
{"content":"k1:v1+k2:v2?k3:v3*k4:v4/k5:v5\k6:v6.k7:v7  k8:v8","k1":"v1","k2":"v2","k3":"v3","k4":"v4","k5":"v5","k6":"v6","k7":"v
7","k8":"v8"}
```

json函数

函数介绍

对指定字段的json对象进行展开、jmes提取或jmes提取后再展开。指定字段值必须为json对象。

语法描述

```
json(key, expand=true, depth=100, prefix="", suffix="", fmt="simple", sep=".",
     expand_array=True, fmt_array="{parent_rlist[0]}_{index}", include_node="",
     exclude_node="", include_path="", exclude_path="",jmes="",
     output="", jmes_ignore_none=true, mode='fill-auto'
)
```

参数说明	参数名称	参数类型	是否必填	参数描述
key	string	是	指定的字段名称	
expand	bool	否	是否将字段展开。jmes为空时，默认为true，表示展开，否则为false。	
depth	number	否	字段展开深度。取值范围1-500。默认100	
prefix	string	否	展开字段名前缀	
suffix	string	否	展开字段名后缀	
fmt	string	否	展开字段格式，默认simple。simple：{prefix}{current}{suffix} full：{parent_list_str}{sep}{prefix}{current}{suffix} parent：{parent}{sep}{prefix}{current}{suffix} root：{parent_list[0]}{sep}{prefix}{current}{suffix}	
sep	string	否	分级节点间的分隔符。默认“.”	
expand_array	bool	否	是否展开数组，true表示展开。默认true	
fmt_array	string	否	数组展开格式。可以自定义格式化占位符，最多五个，默认{parent*rlist[0]*}{index}。parrent_list；parent_rlist；index；sep；prefix；suffix	
include_node	string	否	包含的节点的正则表达式	
exclude_node	string	否	排除的节点的正则表达式	
include_path	string	否	包含路径的正则表达式	
exclude_path	string	否	排除路径的正则表达式	
jmes	string	否	从原日志中提取数据的jmes	
output	string	否	jmes提取内容输出的字段名	

jmes_ignore_none	bool	否	jmes未提取到值时是否忽略，true表示忽略，否则输出一个空字符串。默认true
mode	string	否	字段的覆盖模式。默认fill-auto，详见文末字段覆盖模式类型

补充说明： include_path和exclude_path过滤。 将字段展开后，再对其“full”格式化字段名称从开头进行正则匹配（系统会在用户输入的正则表达式前面加上“^”），分隔符为“.”（不影响最终输出格式和分隔符）。 如果展开后“full”格式的字段名称为：user.address.city include_path="user" //可以匹配 include_path="user.add" //可以匹配 include_path="address.city" //无法匹配

示例1

为展开字段增加前后缀

原始日志

```
{"content":{"k1":"v1","k2":"v2"}}
```

加工规则

```
json("content",prefix="p_",suffix="_s")
```

加工结果

```
{"content":{"k1":"v1","k2":"v2"},"p_k1_s":"v1","p_k2_s":"v2"}
```

示例2

默认格式展开

原始日志

```
{"content":{"k1":"v1","k2":["10","20"]}}
```

加工规则

```
json("content")
```

加工结果

```
{"content":{"k1":"v1","k2":["10","20"]},"k1":"v1","k2_0":"10","k2_1":"20"}
```

示例3

将fmt设置为full展开

原始日志

```
{"content":{"k1":"v1","k2":["10","20"]}}
```

加工规则

```
json("content",fmt="full")
```

加工结果

```
{"content":{"k1":"v1","k2":["10","20"]},"content.k1":"v1","content.k2.k2_0":"10","content.k2.k2_1":"20"}
```

示例4

将fmt设置为parent展开

原始日志

```
{"content":{"k1":"v1","k2":["10","20"]}}
```

加工规则

```
json("content",fmt="parent")
```

加工结果

```
{"content":{"k1":"v1","k2":["10","20"]},"content.k1":"v1","k2.k2_0":"10","k2.k2_1":"20"}
```

示例5

将fmt设置为root展开

原始日志

```
{"content":{"k1":"v1","k2":["10","20"]}}
```

加工规则

```
json("content",fmt="root")
```

加工结果

```
{"content":{"k1":"v1","k2":["10","20"]},"content.k1":"v1","content.k2_0":"10","content.k2_1":"20"}
```

示例6

自定义fmt_array

原始日志

```
{"content":[{"name":"ks","age":23},{"name":"gt","age":38}]}
```

加工规则

```
json("content",fmt="parent",fmt_array="{parent_list[0]}--{index}")
```

加工结果

```
{"content--0.age":23,"content--0.name":"ks","content--1.age":38,"content--1.name":"gt","content":[{"age":23,"name":"ks"},{"age":38,"name":"gt"}]}
```

示例7

使用jmes

原始日志

```
{"content":{"k1":"v1","k2":{"k3":"v3","k4":{"k5":"v5","k6":["one","tow"]}}}}
```

加工规则

```
json("content",jmes="k2.k3",output="jmes_content")
```

加工结果

```
{"jmes_content":"v3","content":{"k1":"v1","k2":{"k3":"v3","k4":{"k5":"v5","k6":["one","tow"]}}}}
```

示例8

使用include_path过滤

原始日志

```
{"content":{"k1":{"k1-1":"k1-1.v"},"k2":{"k2-1":"k2-1.v"}}}
```

加工规则

```
json("content",fmt="full",include_path="content.k1")
```

加工结果

```
{"content":{"k1":{"k1-1":"k1-1.v"},"k2":{"k2-1":"k2-1.v"}},"content.k1.k1-1":"k1-1.v"}
```

kv函数

函数介绍

通过quote提取字段中的键值对信息。

语法描述

```
kv(key1,key2,..., sep="=", quote='"', escape=false, prefix="", suffix="", mode="fill-auto")
```

参数说明	参数名称	参数类型	是否必填	参数描述

key	string	是	字段名称
sep	string	否	键与值的分隔符，仅单字符。默认“=”
quote	string	否	用于包裹值的字符。如果提取对象中不包含引用符，则只提取中文、字母、数字以及符号：_-.%~。默认为"
escape	bool	否	是否提取反转字符的值。false表示不提取。例：key="abc\"def"，默认提取值为abc\，当设置escape=true时，提取abc"def。默认为false
prefix	string	否	提取字段名前缀。
suffix	string	否	提取字段名后缀。
mode	string	否	字段覆盖模式，默认fill-auto，详见文末字段覆盖模式类型

示例1

默认参数提取url键值对信息

原始日志

```
{"content":"https://passport.ksyun.com/?k1=v1&k2=v2&k3="}
```

加工规则

```
kv("content")
```

加工结果

```
{"content":"https://passport.ksyun.com/?k1=v1&k2=v2&k3=","k1":"v1","k2":"v2"}
```

示例2

设置escape为true，提取包裹符中嵌套的包裹符。

原始日志

```
{"content":"k1:\"v1\\\"abc\", k2:\"v2\", k3: \"v3\""}
#转义前：{"content":"k1:"v1\"abc", k2:"v2", k3:"v3""}
```

加工规则

```
kv("content",sep=":",escape=true)
```

加工结果

```
{"content":"k1:"v1\"abc",k2:"v2",k3:"v3"","k1":"v1"abc","k2":"v2","k3":"v3"}
```

regex函数

函数介绍

通过正则表达式提取值并赋值给指定字段。

语法描述

```
regex(key,expr,fields_info,mode="fill-auto",pack_json="")
```

参数说明	参数名称	参数类型	是否必填	参数描述
key	string	是	被提取字段名称	
expr	string	否	提取指定字段值的正则表达式。	
fields_info	string/list/map	当正则表达式中没有设置命名捕获的名称时，必须配置该参数	匹配后的目标字段名称。	
mode	string	否	字段覆盖模式。默认fill-auto，详见文末字段覆盖模式类型	
pack_json	string	否	将正则表达式提取的所有结果放入该参数指定的字段中，默认不打包。	

示例1

获取第一个匹配的结果

原始日志

```
{"content":"123abc456"}
```

加工规则

```
regex("content","\d+","count")
```

加工结果

```
{"content":"123abc456","count":"123"}
```

示例2

获取多个匹配结果

原始日志

```
{"content":"123abc456"}
```

加工规则

```
regex("content","\d+",["count","size"])
```

加工结果

```
{"content":"123abc456","count":"123","size":"456"}
```

示例3

使用正则表达式捕获组，并使用map动态命名键和值

原始日志

```
{"content":"abc:123,edf:456"}
```

加工规则

```
regex("content","([a-z]+):(\d+)",{"key_\1":"value_\2"})
```

加工结果

```
{"content":"abc:123,edf:456","key_abc":"value_123","key_edf":"value_456"}
```

示例4

使用正则表达式命名捕获组获取字段值。

原始日志

```
{"content":"abc:456"}
```

加工规则

```
regex("content","(?P<key>[a-z]+):(?P<value>\d+)")
```

加工结果

```
{"content":"abc:456","key":"abc","value":"456"}
```

示例5

将提取字段打包放入pack_json设置的字段中。

原始日志

```
{"content":"abc:456"}
```

加工规则

```
regex("content","\d+","count",pack_json="packed")
```

加工结果

```
{"content":"abc:456","packed":{"count":"456"}}
```

字段覆盖模式类型

参数值	说明
fill	当目标字段不存在或者值为空时，设置目标字段。
fill-auto	当新值非空，且目标字段不存在或者值为空时，设置目标字段。
add	当目标字段不存在时，设置目标字段。
add-auto	当新值非空，且目标字段不存在时，设置目标字段。
overwrite	强制覆盖写入目标字段。
overwrite-auto	若新值非空，覆盖写入目标字段。

流程控制函数

if函数

函数介绍

根据条件确定是否执行操作，可同时传入多组条件和操作，条件和操作必须成对出现。

语法描述

```
if(条件, 操作)
if(条件1, 操作1, 条件2, 操作2...)
```

参数说明	参数名称	参数类型	是否必填	参数描述
条件	任意	是		表达式或其组合
操作	全局操作函数	是		全局操作函数

示例1

根据条件提取需要的字段

原始日志

```
{"k1":"v1","k2":"v2","k3":"v3"}
```

加工规则

```
if(op_eq(v("k1"),"v1"),keep_fields("k1","k2"))
```

加工结果

```
{"k1":"v1","k2":"v2"}
```

示例2

根据条件删除行

原始日志

```
{"k1":"v1","k2":"v2","k3":"v3"},
{"k1":"v1.1","k2":"v2","k3":"v3"}
```

加工规则

```
if(op_eq(v("k1"),"v1"),drop())
```

加工结果

```
{"k1":"v1.1","k2":"v2","k3":"v3"}
```

if_else函数

函数介绍

根据条件执行相应的操作。

语法描述

```
if(条件, 真时操作,假时操作)
```

参数说明	参数名称	参数类型	是否必填	参数描述

条件	any	是	表达式或其组合
真时操作	全局操作函数	是	全局操作函数
假时操作	全局操作函数	全局操作函数	是

示例

原始日志

```
{"gender":0}
```

加工规则

```
if_else(v("gender"),set("dis_gender","男"),set("dis_gender","女"))
```

加工结果

```
{"gender":0,"dis_gender":"女"}
```

switch函数

函数介绍

多组条件和操作，满足其中任一条件，执行对应操作并返回。

语法描述

```
switch(条件1,操作1,条件2,操作2...,default=null)
```

参数说明	参数名称	参数类型	是否必填	参数描述
条件	any	是		表达式或其组合
操作	全局操作函数	是		全局操作函数
default	全局操作函数	是		全局操作函数

示例1 根据age的值设置age_group，如果age小于18设置为未成年，如果age大于等于18设置为成年。

原始日志

```
{"name":"ks","age":17},
{"name":"mike","age":35}
```

加工规则

```
switch(op_lt(v("age"),18),set("age_group","未成年"),op_ge(v("age"),18),set("age_group","成年"))
```

加工结果

```
{"name":"ks","age":17,"age_group":"未成年"}
{"name":"mike","age":35,"age_group":"成年"}
```

示例2

丢弃不符合要求的日志。

原始日志

```
{"type":"goods","size":123,"data":"content"},
{"type":"store","size":123,"data":"content"},
{"type":"unkonw","size":123,"data":"content"}
```

加工规则

```
switch(eq(v("type"),"goods"),keep(),eq(v("type"),"store"),keep(),default=drop())
```

加工结果

```
{"type":"goods","size":123,"data":"content"}
{"type":"store","size":123,"data":"content"}
```

compose函数

函数介绍

组合多个操作，多用于传参。

语法描述

compose(操作1,操作2...)

参数说明	参数名称	参数类型	是否必填	参数描述
操作	全局操作函数	是		全局操作函数

示例 如果content的值为123，则增加k1字段，然后删除name字段。

原始日志

```
{"name":"ks","content":"123"}
```

加工规则

```
if(op_eq(v("content"),"123"),compose(set("k1","v1"),drop_fields("name")))
```

加工结果

```
{"k1":"v1","content":"123"}
```

行处理函数

drop函数

函数介绍

根据条件判断是否丢弃该数据

语法描述

drop(condition)

参数说明	参数名称	参数类型	是否必填	参数描述
condition	bool	否		是否丢弃该数据，一般传入一个条件判断函数的结果

示例1

原始日志

```
{"status":true, "k1":"v1"},
{"status":false, "k1":"v1"}
```

加工规则

```
if(v("status"),drop())
```

加工结果

```
{"status":false, "k1":"v1"}
```

keep函数

函数介绍

根据条件判断是否保留该数据

语法描述

keep(condition)

参数说明	参数名称	参数类型	是否必填	参数描述
condition	bool	否		是否保留该数据，一般传入一个条件判断函数的结果,默认为true

示例

原始日志

```
{"status":true, "k1":"v1"},
```

```
{"status":false, "k1":"v1"}
```

加工规则

```
keep(v("status"))
```

加工结果

```
{"status":true, "k1":"v1"}
```

split函数

函数介绍

基于日志字段的值分裂出多条日志，并且支持通过JMES提取字段后再进行分裂。分裂失败返回原日志。

语法描述

```
split(key, sep=",", quote="\"", lstrip=true,jmes="", output="")
```

参数说明	参数名称	参数类型	是否必填	参数描述
key	string	是	需要分裂的字段名。字段值必须是字符串或数组。	
sep	string	否	分割多个值的分隔符。必须是单个字符。默认","	
quote	string	否	包裹值的引用符。必须是单个字符。默认"	
lstrip	bool	否	是否将值左边的空格去掉。默认true	
jmes	string	否	将字段转换为json对象，再根据JMES提取的值进行分裂。支持字符串和数组。	
output	string	否	为分裂值设置新的字段名。默认会直接使用key参数传入的字段名称	

示例1

原始日志

```
{"from":"a,b,c","name":"ks","age":23}
```

加工规则

```
split("from")
```

加工结果

```
[
    {"from":"a","name":"ks","age":23},
    {"from":"b","name":"ks","age":23},
    {"from":"c","name":"ks","age":23}
]
```

示例2

将分裂值输出到新的字段名。

原始日志

```
{"from":"a,b,c","name":"ks","age":23}
```

加工规则

```
split("from",output="new_column")
```

加工结果

```
[
    {"from":"a,b,c","new_column":"a","name":"ks","age":23},
    {"from":"a,b,c","new_column":"b","name":"ks","age":23},
    {"from":"a,b,c","new_column":"c","name":"ks","age":23}
]
```

示例3

自定义分隔符和引用符。

原始日志

```
{"from":"*a*|*b*|*c*","name":"ks","age":23}
```

加工规则

```
split("from",sep="|",quote="*")
```

加工结果

```
[
    {"from":"a","name":"ks","age":23},
    {"from":"b","name":"ks","age":23},
    {"from":"c","name":"ks","age":23}
]
```

output函数

函数介绍

将日志输出到指定目标日志池。如果目标日志池不存在，日志将被输出到错误日志池。

语法描述

```
output(name,stop=true)
```

参数说明	参数名称	参数类型	是否必填	参数描述
name	string	是	数据加工任务中配置的输出目标的名称。	
stop	bool	否	是否终止之后的函数执行。默认true	

示例1 根据条件将日志输出到指定日志池

原始日志

```
{"level":"info","content":"123"}
```

加工规则

```
switch(op_eq(v("level"),"info"),output("log_info"),op_eq(v("level"),"error"),output("log_error"))
```

加工结果

日志被输出到目标名称为“log_info”的日志池。

示例2 设置stop为false，继续处理日志并将其插入到默认日志池。

原始日志

```
{"level":"info","content":"123"}
```

加工规则

```
if(op_eq(v("level"),"info"),output("log_info",stop=false))
set("content","456")
```

加工结果

```
{"level":"info","content":"123"}：被输出到目标名称为“log_info”的日志池。
{"level":"info","content":"456"}：被输出到默认日志池
```

字段处理函数

v函数

函数介绍

获取指定字段的值，可传入多个字段名，返回第一个存在的字段的值。

语法描述

```
v(key1,key2,...,default=null)
```

参数说明	参数名称	参数类型	是否必填	参数描述
key	string	是	指定的字段名称	
default	any	否	当指定的字段名不存在时，返回default的值。	

示例

根据条件提取需要的字段

原始日志

```
{"name":"ks","age":18}
```

加工规则

```
set("new_name",v("name"))
```

加工结果

```
{"name":"ks","age":18,"new_name":"ks"}
```

set函数

函数介绍

添加新的字段或为字段重新赋值，可同时传递多组key和value。

语法描述

```
set(key1,value1,key2,value2,...,mode="overwrite")
```

参数说明	参数名称	参数类型	是否必填	参数描述
key	string	是	指定字段名称	
value	any	是	指定字段值	
mode	string	否	字段覆盖模式，overwrite	

示例1

添加一个新字段k2

原始日志

```
{"k1":"v1"}
```

加工规则

```
set("k2","v2")
```

加工结果

```
{"k1":"v1","k2":"v2"}
```

示例2

将k1的字段值赋给新字段k2

原始日志

```
{"k1":"v1"}
```

加工规则

```
set("k2",v("k1"))
```

加工结果

```
{"k1":"v1","k2":"v1"}
```

示例3

如果存在k1字段，将其值赋给新字段k2

原始日志

```
{"k1":"v1"}
```

加工规则

```
if(has_field("k1"),set("k2",v("k1")))
```

加工结果

```
{"k1":"v1","k2":"v1"}
```

drop_fields函数

函数介绍

删除指定的字段。

语法描述

```
drop_fields(key1,key2,...,regex=false)
```

参数说明	参数名称	参数类型	是否必填	参数描述
key	string	是	需要删除的字段名称，可以是正则表达式。	
regex	bool	否	是否将参数key作为正则表达式进行匹配，设置为false表示不作为正则表达式匹配，默认为false	

示例 当存在字段k1时，删除字段k2。

原始日志

```
{"k1":"v1","k2":"v2","k3":"v3"}
```

加工规则

```
if(has_field("k1"),drop_fields("k2"))
```

加工结果

```
{"k1":"v1","k3":"v3"}
```

keep_fields函数

函数介绍

保留指定字段，其它字段将被删除。

语法描述

```
keep_fields(key1,key2,...,regex=true)
```

参数说明	参数名称	参数类型	是否必填	参数描述
key	string	是	需要保留的字段名称，可以是正则表达式，其它字段将被删除	
regex	bool	否	是否将参数key作为正则表达式进行匹配，设置为false表示不作为正则表达式匹配。默认是true	

示例 当存在字段k1时，保留字段k1，k2。

原始日志

```
{"name":"ks","content":"123"}
```

加工规则

```
if(has_field("k1"),keep_fields("k1","k2"))
```

加工结果

```
{"k1":"v1","k2":"v2"}
```

pack_fields函数

函数介绍

将符合条件的字段打包后输出到新的字段中。

语法描述

```
pack_fields(output_field,include=".*",exclude="",drop_packed=true)
```

参数说明	参数名称	参数类型	是否必填	参数描述
output_field	string	是	打包后输出的目标字段名称	
include	string	否	包含字段的正则表达式，符合该表达式的字段将会被打包	
exclude	string	否	排除字段的正则表达式，符合该表达式的字段将不会被打包	
drop_packed	bool	否	是否删除被打包的原字段，true表示删除，默认是true	

示例1 将所有字段打包输出到新字段k_all中

原始日志

```
{"k1":"v1","k2":"v2","k3":"v3"}
```

加工规则

```
pack_fields("k_all")
```

加工结果

```
{"k_all":{"k1":"v1","k2":"v2","k3":"v3"}}
```

示例2 打包符合正则表达式的字段到新字段k_all中，并保留被打包字段

原始日志

```
{"k1":"v1","k2":"v2","content":"content"}
```

加工规则

```
pack_fields("k_all",include="[a-z]+\d+",drop_packed=false)
```

加工结果

```
{"k_all":{"k1":"v1","k2":"v2"},"k1":"v1","k2":"v2","content":"content"}
```

rename函数

函数介绍

对符合条件的字段进行重命名

语法描述

```
rename(key1,new_key1,key2,new_key2,...,regex=false)
```

参数说明	参数名称	参数类型	是否必填	参数描述
key	string	是	字段名，可以为正则表达式。	
new_key	string	是	字段新名称	
regex	bool	否	设置为true时，表示将key参数作为正则进行匹配，默认为true	

示例

将字段名称k1修改为name

原始日志

```
{"k1":"v1"}
```

加工规则

```
rename("k1","name")
```

加工结果

```
{"name":"v1"}
```

正则表达式函数

regex_select函数

函数介绍

根据正则表达式提取符合条件的值。

语法描述

```
regex_select(value, expr, mi=0, gi=0)
```

参数说明	参数名称	参数类型	是否必填	参数描述
key	any	是	需要匹配的值	
expr	string	是	正则表达式	
mi	number	否	表示匹配到的第几个表达式，0表示第一个。默认是0	
gi	number	是	表示匹配到的第几个分组，0表示第一个。默认是0	

示例1

从字段content中获取第一个满足正则表达式的值，并存入新字段selected中

原始日志

```
{"content":"123abc456edf789"}
```

加工规则

```
set("selected",regex_select(v("content"),"\\d+"))
```

加工结果

```
{"content":"123abc456edf789","selected":"123"}
```

示例2

从字段content中获取第二个满足正则表达式的值，并存入新字段selected中。

原始日志

```
{"content":"123abc456edf789"}
```

加工规则

```
set("selected",regex_select(v("content"),"\\d+",mi=1))
```

加工结果

```
{"content":"123abc456edf789","selected":"456"}
```

示例3 从字段content中获取第二个满足正则表达式的第二组值，并存入新字段selected中。

原始日志

```
{"content":"123abc456edf789"}
```

加工规则

```
set("selected",regex_select(v("content"),"(\\d+)([a-z]+)",mi=1,gi=1))
```

加工结果

```
{"content":"123abc456edf789","selected":"edf"}
```

regex_match函数

函数介绍

判断是否匹配正则表达式。

语法描述

```
regex_match(value, expr, full=false)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是	需要匹配的值	

expr	string	是	正则表达式
full	bool	否	是否完全匹配正则表达式，默认false

示例1

字段content中是否包含数字

原始日志

```
{"content":"123abc456edf789"}
```

加工规则

```
set("matched",regex_match(v("content"),"\\d+"))
```

加工结果

```
{"content":"123abc456edf789","matched":true}
```

示例2

字段content中是否都是数字

原始日志

```
{"content":"123abc456edf789"}
```

加工规则

```
set("matched",regex_match(v("content"),"\\d+",full=true))
```

加工结果

```
{"content":"123abc456edf789","matched":false}
```

regex_split函数

函数介绍

使用正则表达式将字符串分裂为数组。

语法描述

```
regex_split(value, expr, maxsplit=0)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是	需要分裂的值	
expr	string	是	正则表达式	
maxsplit	number	否	最大分裂次数。0表示全部分裂，1表示只分裂第一个匹配，剩余部分不再分裂。	

示例 通过逗号分裂content字段的值，将结果作为新字段split的值

原始日志

```
{"content":"a,b,c,d"}
```

加工规则

```
set("split",regex_split(v("content"),","))
```

加工结果

```
{"content":"a,b,c,d","split":["a","b","c","d"]}
```

regex_replace函数

函数介绍

根据正则表达式替换字符串中的指定字符。

语法描述

```
regex_replace(value, expr, replace="", count=0)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是	需要被替换的值	
expr	string	是	正则表达式	
replace	string	否	替换后的新字符。可以使用捕获组。 \1表示第一个分组；\2表示第二个分组；依此类推。默认空字符串	
count	number	否	表示替换次数。0表示替换所有。默认是0	

示例1 将字段content中的所有数字替换成*。

原始日志

```
{"content":"123abc456edf789"}
```

加工规则

```
set("replaced",regex_replace(v("content"),"\\d",replace="*"))
```

加工结果

```
{"content":"123abc456edf789","replaced":"***abc***edf***"}
```

示例2 将字段content中的第一个数字替换成*。

原始日志

```
{"content":"123abc456edf789"}
```

加工规则

```
set("replaced",regex_replace(v("content"),"\\d",replace="*",count=1))
```

加工结果

```
{"content":"123abc456edf789","replaced":"*23abc456edf789"}
```

regex_findall函数

函数介绍

根据正则表达式获取符合条件的值，将其组成一个数组返回。

语法描述

```
regex_findall(value, expr)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是	需要匹配的值	
expr	string	是	正则表达式	

示例1 从content字段中获取所有字母的列表。

原始日志

```
{"content":"123abc456edf789"}
```

加工规则

```
set("found",regex_findall(v("content"),"[a-z]+"))
```

加工结果

```
{"content":"123abc456edf789","found":["abc","edf"]}
```

日期时间函数

dt_str函数

函数介绍

将值或时间表达式的值转换为字符串。

语法描述

```
dt_str(value, fmt="format_string", tz=None)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	字符串、Unix时间戳、日期时间对象	是	值或时间表达式。	
fmt	String	否	格式化字符串。更多信息，请参见日期时间格式化指令。默认为空，则格式保持不变。	
tz	String	否	表示时区，默认为None。更多信息，请参见时区列表。	

示例1

把time字段的值转换成fmt形式，时区东京。

原始日志

```
{
"time":"2019-06-03 02:41:26",
"fmt":"%Y/%m/%d %H-%M-%S",
"tz":"Asia/Tokyo"
}
```

加工规则

```
set("dt_str", dt_str(v("time"),fmt=v("fmt"),tz=v("tz")))
```

加工结果

```
{
dt_str:"2019/06/03 02-41-26"
fmt:"%Y/%m/%d %H-%M-%S"
time:"2019-06-03 02:41:26"
tz:"Asia/Tokyo"
}
```

示例2

把time字段的值（Unix时间戳）转换成fmt形式。

原始日志

```
{
"time":"1559500886",
"fmt":"%Y/%m/%d %H-%M-%S"
}
```

加工规则

```
set("dt_str", dt_str(v("time"),fmt=v("fmt")))
```

加工结果

```
{
dt_str:"2019/06/02 18-41-26"
fmt:"%Y/%m/%d %H-%M-%S"
time:"1559500886"
}
```

示例3

把time字段的值转换成默认形式

原始日志

```
{
"time":"2019-06-03 02:41:26"
}
```

加工规则

```
set("dt_str", dt_str(v("time")))
```

加工结果

```
{
dt_str:"2019-06-03 02:41:26"
time:"2019-06-03 02:41:26"
}
```

dt_to_timestamp函数

函数介绍

将日期时间对象转换为Unix时间戳。

语法描述

```
dt_to_timestamp (timeString)
```

参数说明	参数名称	参数类型	是否必填	参数描述
timeString	日期时间字符串	是		需要被转换的日期时间对象。

示例

原始日志

```
{
"time":"2019-06-03 02:41:26"
}
```

加工规则

```
set("dt_to_timestamp",dt_to_timestamp(v("time")))
```

加工结果

```
{
dt_to_timestamp:1559529686
time:"2019-06-03 02:41:26"
}
```

dt_from_timestamp函数

函数介绍

将Unix时间戳转换为日期时间对象。

语法描述

```
dt_from_timestamp(value, tz=None)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	String	是	值或时间表达式。	
tz	String	否	表示时区，默认为None。更多信息，请参见时区列表。	

示例1

原始日志

```
{
"time":"1559500886"
}
```

加工规则

```
set("dt_from_timestamp",dt_from_timestamp(v("time")))
```

加工结果

```
{
dt_from_timestamp:"2019-06-02 18:41:26"
time:"1559500886"
```

```
}
```

示例2

原始日志

```
{
"time":"1559500886",
"tz":"Asia/Shanghai"
}
```

加工规则

```
set("dt_from_timestamp",dt_from_timestamp(v("time"),tz=v("tz")))
```

加工结果

```
{
  "dt_from_timestamp": "2019-06-03 02:41:26+08:00",
  "time": "1559500886",
  "tz": "Asia/Shanghai"
}
```

dt_now函数

函数介绍

获取当前日期时间对象。

语法描述

```
dt_now(tz=None)
```

参数说明	参数名称	参数类型	是否必填	参数描述
tz	String	否	表示时区，默认为None。更多信息，请参见时区列表。	

示例1

获取当前时间，时区是上海。

原始日志

```
{
"tz":"Asia/Shanghai"
}
```

加工规则

```
set("dt_now",dt_now(tz=v("tz")))
```

加工结果

```
{
  "dt_now": "2023-04-23 10:46:15.78605745+08:00",
  "tz": "Asia/Shanghai"
}
```

字符串处理函数

str_count函数

函数介绍

查找字符串中某个子串出现的次数。

语法描述

```
str_count(value, sub, start, end)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是	被查找的字符串	
sub	any	是	需要查找的子串	

start	number	否	开始查找的位置。 0：第一个字符。
end	number	否	查找结束的位置。 -1：最后一个字符。

示例1

获取字段content中子串abc的数量。

原始日志

```
{"content":"abc123abc456"}
```

加工规则

```
set("count",str_count(v("content"),"abc"))
```

加工结果

```
{"content":"abc123abc456","count":2}
```

str_uppercase函数

函数介绍

将指定字符串中的全部字母转换为大写。

语法描述

```
str_uppercase(value)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是		需要被转换的字符串

示例

将字段content的值转换为大写。

原始日志

```
{"content":"abc123abc456"}
```

加工规则

```
set("upper",str_uppercase(v("content")))
```

加工结果

```
{"content":"abc123abc456","upper":"ABC123ABC456"}
```

str_lowercase函数

函数介绍 将指定字符串中的全部字母转换为小写。

语法描述

```
str_lowercase(value)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是		需要被转换的字符串

示例

将字段content的值转换为大写。

原始日志

```
{"content":"ABC123ABC456"}
```

加工规则

```
set("upper",str_lowercase(v("content")))
```

加工结果

```
{"content":"abc123abc456","upper":"abc123abc456"}
```

str_join函数

函数介绍 使用连接符将多个字符串连接成一个新的字符串。

语法描述

```
str_join(connector, value1, value2, ....)
```

参数说明	参数名称	参数类型	是否必填	参数描述
connector	any	是		连接符
value1	any	是		待连接的字符串
value2	any	是		待连接的字符串

示例

将字段province和city使用连接符 “-” 进行连接，并存入新字段address。

原始日志

```
{"province":"湖北省","city":"武汉市"}
```

加工规则

```
set("address",str_join("-",v("province"),v("city")))
```

加工结果

```
{"address":"湖北省-武汉市","city":"武汉市","province":"湖北省"}
```

str_replace函数

函数介绍 将字符串中的指定子串替换为新值。

语法描述

```
str_replace(value, old, new, count)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是		需要被替换的字符串
old	any	是		需要被替换的子串
new	any	是		用于替换的新值
count	any	否		替换次数，0表示替换所有。默认是0

示例

将字段content值中的 “=” 替换为 “:”。

原始日志

```
{"content":"k1=v1,k2=v2"}
```

加工规则

```
set("replaced",str_replace(v("content"),"=",":"))
```

加工结果

```
{"content":"k1=v1,k2=v2","replaced":"k1:v1,k2:v2"}
```

str_format函数

函数介绍 格式化字符串。

语法描述

```
str_format(format_string, value1, value2, ...)
```

参数说明	参数名称	参数类型	是否必填	参数描述

format_string	any	是	转换后的字符串格式。占位符：{}
value	any	是	待格式化的值

示例

将字段province和city的值格式化为新字段format。

原始日志

```
{"province":"湖北","city":"武汉"}
```

加工规则

```
set("format",str_format("省份：{}，城市：{}",v("province"),v("city")))
```

加工结果

```
{"province":"湖北","city":"武汉","format":"省份：湖北，城市：武汉"}
```

str_strip函数

函数介绍 删除字符串首尾指定字符集。

语法描述

```
str_strip(value, chars)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是	原字符串	
chars	any	否	字符串首尾需要被删除的字符集，默认\t\r\n\v\f	

示例1

删除字段content值首尾的空格。

原始日志

```
{"content":" abc123cba "}
```

加工规则

```
set("striped",str_strip(v("content")))
```

加工结果

```
{"content":" abc123cba ","striped":"abc123cba"}
```

示例2

删除字段content值首尾的字符集abc。

原始日志

```
{"content":"abc123cba"}
```

加工规则

```
set("striped",str_strip(v("content"),"abc"))
```

加工结果

```
{"content":"abc123cba","striped":"123"}
```

str_lstrip函数

函数介绍 删除字符串开头的指定字符集。

语法描述

```
str_lstrip(value, chars)
```

参数说明	参数名称	参数类型	是否必填	参数描述

value	any	是	原字符串
chars	any	否	字符串开头需要被删除的字符集，默认\t\r\n\v\f

示例1

删除字段content值开头的空格。

原始日志

```
{"content":" abc123cba "}
```

加工规则

```
set("lstriped",str_lstrip(v("content")))
```

加工结果

```
{"content":" abc123cba ","lstriped":"abc123cba "}
```

示例2

删除字段content值开头的字符集abc。

原始日志

```
{"content":"abccb123cba"}
```

加工规则

```
set("lstriped",str_lstrip(v("content"),"abc"))
```

加工结果

```
{"content":"abccb123cba","lstriped":"123cba"}
```

str_rstrip函数

函数介绍 删除字符串结尾的指定字符集。

语法描述

```
str_rstrip(value, chars)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是	原字符串	
chars	any	否	字符串结尾需要被删除的字符集，默认\t\r\n\v\f	

示例1

删除字段content值结尾的空格。

原始日志

```
{"content":" abc123cba "}
```

加工规则

```
set("rstriped",str_rstrip(v("content")))
```

加工结果

```
{"content":" abc123cba ","rstriped":" abc123cba"}
```

示例2

删除字段content值结尾的字符集abc。

原始日志

```
{"content":"abccb123cbabc"}
```

加工规则

```
set("rstriped",str_rstrip(v("content"),"abc"))
```

加工结果

```
{"content":"abccb123cbabc","rstriped":"abccb123"}
```

str_find函数

函数介绍 判断字符串中是否包含指定子串。返回指定子串在原字符串中出现的索引位置，如果有多个，则只返回第一个。

语法描述

```
str_find(value, str, begin, end)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是	被查找的字符串。	
str	any	是	指定查找的子串。	
begin	number	否	开始查找的索引位置。0表示第一个字符。默认是0	
end	nunber	否	结束查找的索引位置。-1表示最后一个字符。默认是-1	

示例

原始日志

```
{"content":"kingcloud"}
```

加工规则

```
set("found_index",str_find(v("content"),"cloud"))
```

加工结果

```
{"content":"kingcloud","found_index":4}
```

str_start_with函数

函数介绍 判断字符串是否以指定子串开头。返回查找结果bool，true表示以指定子串开头。

语法描述

```
str_start_with(value, prefix, start, end)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是	被判断的字符串。	
prefix	any	是	指定的前缀子串。	
begin	number	否	开始查找的索引位置。0表示第一个字符。默认是0	
end	nunber	否	结束查找的索引位置。-1表示最后一个字符。默认是-1	

示例

原始日志

```
{"content":"kingcloud"}
```

加工规则

```
set("start-with",str_start_with(v("content"),"king"))
```

加工结果

```
{"content":"kingcloud","start-with":true}
```

str_end_with函数

函数介绍 判断字符串是否以指定子串结尾。返回查找结果bool，true表示以指定子串开头。

语法描述

```
str_end_with(value, suffix, start, end)
```

参数说明	参数名称	参数类型	是否必填	参数描述

value	any	是	被判断的字符串。
prefix	any	是	指定的后缀子串。
begin	number	否	开始查找的索引位置。0表示第一个字符。默认是0
end	nunber	否	结束查找的索引位置。-1表示最后一个字符。默认是-1

示例

原始日志

```
{"content":"kingcloud"}
```

加工规则

```
set("end-with",str_end_with(v("content"),"cloud"))
```

加工结果

```
{"content":"kingcloud","end-with":true}
```

逻辑表达式

op_if函数

函数介绍

根据条件返回不同表达式的值。

语法描述

```
op_if(condition, expr1, expr2)
```

参数说明	参数名称	参数类型	是否必填	参数描述
condition	any	是	判断条件。非bool值会转换成bool，转换规则详见文末bool判断说明	
expr1	any	是	判断条件为true时，返回该表达式的值。	
expr2	any	是	判断条件为false时，返回该表达式的值。	

示例1

判断status的值，并根据判断结果往新字段message中写入相应的值。

原始日志

```
{"status":true}
```

加工规则

```
set("message",op_if(v("status"),"成功","失败"))
```

加工结果

```
{"message":"成功","status":true}
```

op_and函数

函数介绍

对指定参数进行真假判断，当所有参数值为true时返回true。

语法描述

```
op_and(value1, value2, ...)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value1	any	是	运算值1。非bool值会转换成bool，转换规则详见文末bool判断说明	
value2	any	是	运算值2。非bool值会转换成bool，转换规则详见文末bool判断说明	

示例2

判断status的值，并根据判断结果往新字段message中写入相应的值。

原始日志

```
{"condition1":"abc","condition2":123}
```

加工规则

```
set("op_and",op_and(v("condition1"),v("condition2")))
```

加工结果

```
{"condition1":"abc","condition2":123,"op_and":true}
```

op_or函数

函数介绍 对指定参数进行真假判断，当任意参数值为true时返回true。

语法描述

```
op_or(value1, value2, ...)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value1	any	是	运算值1。非bool值会转换成bool，转换规则详见文末bool判断说明	
value2	any	是	运算值2。非bool值会转换成bool，转换规则详见文末bool判断说明	

示例

判断字段condition1和condition2的值进行or运算，将结果存入新字段op_or中。

原始日志

```
{"condition1":"abc","condition2":false}
```

加工规则

```
set("op_or",op_or(v("condition1"),v("condition2")))
```

加工结果

```
{"condition1":"abc","condition2":123,"op_or":true}
```

op_not函数

函数介绍 对指定参数进行真假判断，对判断结果取反。

语法描述

```
op_not(value)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是	需要被判断的值。非bool值会转换成bool，转换规则详见文末bool判断说明	

示例

对字段status的布尔结果取反，并将结果存入op_not中。

原始日志

```
{"status":123}
```

加工规则

```
set("op_not",op_not(v("status")))
```

加工结果

```
{"op_not":false,"status":123}
```

op_eq函数

函数介绍

比较两个指定参数的值，相等时返回true，当两个参数类型不同，返回false。

语法描述

```
op_eq(value1, value2)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value1	any	是		运算值1。
value2	any	是		运算值2。

示例1

原始日志

```
{"k1":"123","k2":"123"}
```

加工规则

```
set("op_eq",op_eq(v("k1"),v("k2")))
```

加工结果

```
{"k1":"123","k2":"123","op_eq":true}
```

示例2

原始日志

```
{"k1":"123","k2":123}
```

加工规则

```
set("op_eq",op_eq(v("k1"),v("k2")))
```

加工结果

```
{"k1":"123","k2":123,"op_eq":false}
```

op_ne函数

函数介绍

比较两个指定参数的值，不相等时返回true，当两个参数类型不同，返回true。

语法描述

```
op_ne(value1, value2)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value1	any	是		运算值1。
value2	any	是		运算值2。

示例1

原始日志

```
{"k1":"123","k2":"123"}
```

加工规则

```
set("op_ne",op_ne(v("k1"),v("k2")))
```

加工结果

```
{"k1":"123","k2":"123","op_ne":false}
```

示例2

原始日志

```
{"k1":"123","k2":123}
```

加工规则

```
set("op_ne",op_ne(v("k1"),v("k2")))
```

加工结果

```
{"k1":"123","k2":123,"op_ne":true}
```

op_ge函数

函数介绍 获取指定参数 value1>=value2 的bool结果，两个参数类型不同时返回false。

语法描述

```
op_ge(value1, value2)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value1	any	是		运算值1。
value2	any	是		运算值2。

示例

原始日志

```
{"k1":20,"k2":10}
```

加工规则

```
set("op_ge",op_ge(v("k1"),v("k2")))
```

加工结果

```
{"k1":20,"k2":10,"op_ge":true}
```

op_gt函数

函数介绍 获取指定参数 value1>value2 的bool结果，两个参数类型不同时返回false。

语法描述

```
op_gt(value1, value2)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value1	any	是		运算值1。
value2	any	是		运算值2。

示例1

原始日志

```
{"k1":20,"k2":10}
```

加工规则

```
set("op_gt",op_gt(v("k1"),v("k2")))
```

加工结果

```
{"k1":20,"k2":10,"op_gt":true}
```

op_le函数

函数介绍 获取指定参数 value1<=value2 的bool结果，两个参数类型不同时返回false。

语法描述

```
op_le(value1, value2)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value1	any	是		运算值1。
value2	any	是		运算值2。

示例1

原始日志

```
{"k1":20,"k2":10}
```

加工规则

```
set("op_le",op_le(v("k1"),v("k2")))
```

加工结果

```
{"k1":20,"k2":10,"op_le":false}
```

op_lt函数

函数介绍 获取指定参数 value1<value2 的bool结果，两个参数类型不同时返回false。

语法描述

```
op_lt(value1, value2)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value1	any	是	运算值1。	
value2	any	是	运算值2。	

示例1

原始日志

```
{"k1":20,"k2":10}
```

加工规则

```
set("op_lt",op_lt(v("k1"),v("k2")))
```

加工结果

```
{"k1":20,"k2":10,"op_lt":false}
```

str_find函数

函数介绍 判断字符串中是否包含指定子串。返回指定子串在原字符串中出现的索引位置，如果有多个，则只返回第一个。

语法描述

```
str_find(value, str, begin, end)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是	被查找的字符串。	
str	any	是	指定查找的子串。	
begin	number	否	开始查找的索引位置。0表示第一个字符。默认是0	
end	nunber	否	结束查找的索引位置。-1表示最后一个字符。默认是-1	

示例

原始日志

```
{"content":"kingcloud"}
```

加工规则

```
set("found_index",str_find(v("content"),"cloud"))
```

加工结果

```
{"content":"kingcloud","found_index":4}
```

str_start_with函数

函数介绍 判断字符串是否以指定子串开头。返回查找结果bool，true表示以指定子串开头。

语法描述

```
str_start_with(value, prefix, start, end)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是	被判断的字符串。	
prefix	any	是	指定的前缀子串。	
begin	number	否	开始查找的索引位置。0表示第一个字符。默认是0	
end	nunber	否	结束查找的索引位置。-1表示最后一个字符。默认是-1	

示例

原始日志

```
{"content":"kingcloud"}
```

加工规则

```
set("start-with",str_start_with(v("content"),"king"))
```

加工结果

```
{"content":"kingcloud","start-with":true}
```

str_end_with函数

函数介绍 判断字符串是否以指定子串结尾。返回查找结果bool，true表示以指定子串开头。

语法描述

```
str_end_with(value, suffix, start, end)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	any	是	被判断的字符串。	
prefix	any	是	指定的后缀子串。	
begin	number	否	开始查找的索引位置。0表示第一个字符。默认是0	
end	nunber	否	结束查找的索引位置。-1表示最后一个字符。默认是-1	

示例

原始日志

```
{"content":"kingcloud"}
```

加工规则

```
set("end-with",str_end_with(v("content"),"cloud"))
```

加工结果

```
{"content":"kingcloud","end-with":true}
```

bool判断说明

类型	说明
number	0：false，非0：true
string	字符串长度等于0：false，字符串长度大于0：true
array	元素数量等于0：false，元素数量大于0：true
map	键值对数量等于0：false，键值对数量大于0：true。对象属性均按键值对处理。

IP解析函数

geo_parse函数

函数介绍

根据IP解析地址。

语法描述

```
geo_parse(ip, keep_fields=null)
```

参数说明	参数名称	参数类型	是否必填	参数描述
ip	string	是	需要解析的ip	
keep_fields	数组、map	否	指定返回的结果字段，或对字段进行重命名。isp：运营商；country：国家；province：省份；city：城市	

示例1

原始日志

```
{"ip":"171.83.125.16"}
```

加工规则

```
set("geo",geo_parse(v("ip")))
```

加工结果

```
{"geo":{"city":"武汉","country":"中国","isp":"电信","province":"湖北"},"ip":"171.83.125.16"}
```

示例2

设置keep_fields，指定返回province和city。

原始日志

```
{"ip":"171.83.125.16"}
```

加工规则

```
set("geo",geo_parse(v("ip"),keep_fields=["province","city"]))
```

加工结果

```
{"geo":{"city":"武汉","province":"湖北"},"ip":"171.83.125.16"}
```

示例3

设置keep_fields，指定返回province和city，并重命名为省和市。

原始日志

```
{"ip":"171.83.125.16"}
```

加工规则

```
set("geo",geo_parse(v("ip"),keep_fields={"province":"省","city":"市"}))
```

加工结果

```
{"geo":{"市":"武汉","省":"湖北"},"ip":"171.83.125.16"}
```

值结构化处理函数

json_select函数

函数介绍

根据jmes提取指定的值。

语法描述

```
json_select(value, jmes, default="", restrict=false)
```

参数说明	参数名称	参数类型	是否必填	参数描述

value	json对象/json字符串	是	待提取的json值，可以是json对象或json字符串
jems	string	是	jmes表达式
default	string	否	当提取字段不存在时，返回此值
restrict	bool	否	当json字符串解析失败时，是否丢弃日志。false：忽略失败，返回default的值，继续处理。true：直接报错，丢弃该条日志。默认false

示例1

提取k1.k1_1的值，并赋值给新字段filtered。

原始日志

```
{"content":{"k1":{"k1_1":"v1_1"}}}
```

加工规则

```
set("filtered",json_select(v("content"),"k1.k1_1"))
```

加工结果

```
{"content":{"k1":{"k1_1":"v1_1"}},"filtered":"v1_1"}
```

示例2

提取字段json转换失败，返回default设置的值。

原始日志

```
{"content":"abc"}
```

加工规则

```
set("filtered",json_select(v("content"),"k1.k1_1",default="json parse failed"))
```

加工结果

```
{"content":"abc","filtered":"json parse failed"}
```

xml_to_json函数

函数介绍

将指定的xml转换为json。

语法描述

```
xml_to_json(xml)
```

参数说明	参数名称	参数类型	是否必填	参数描述
xml	string	是		需要被转换的xml

示例

将字段content的xml值转换为json对象并输出到字段json_from_xml。

原始日志

```
{"content":"<user>
                <name>ks</name>
                <age>23</age>
                <address nation=\"china\" province=\"hb\"/>
                <score>
                    <a>100</a>
                    <b>200</b>
                </score>
            </user>"}
```

加工规则

```
set("json_from_xml",xml_to_json(v("content")))
```

加工结果

```
{"content":"<user>
                <name>ks</name>
                <age>23</age>
                <address nation=\"china\" province=\"hb\"/>
                <score>
                    <a>100</a>
                    <b>200</b>
                </score>
            </user>",
            ,"json_from_xml":{"user":{"address":{"-nation":"china","-province":"hb"},"age":"23","name":"ks","score":{"a":"100","b
":"200"}}}}
```


json_parse函数

函数介绍

将json字符串转换为json对象。

语法描述

```
json_parse(value,default="",restrict=false)
```

参数说明	参数名称	参数类型	是否必填	参数描述
value	string	是	需要被转换的内容	
default	string	否	当json字符串为空时返回的默认内容	
restrict	bool	否	当json字符串解析失败时，是否丢弃日志。false：忽略失败，返回default的值，继续处理。true：直接报错，丢弃该条日志。	

示例

原始日志

```
{"content":"{\"k1\":\"v1\",\"k2\":\"v2\"}"}
```

加工规则

```
set("json",json_parse(v("content")))
```

加工结果

```
{"content":"{\"k1\":\"v1\",\"k2\":\"v2\"}","json":{"k1":"v1","k2":"v2"}}
```

真假判断

数据类型	True的条件	False的条件
布尔	True，true	False，false
空	无	总是False
数值	非0或非0.0	0或0.0
string	非空	空串
byte	非空	空字节
list	非空	空列表
map	非空	空字典
日期时间	存在即为True	空对象（None）

时区列表

时区参数tz的所有可能取值如下。为方便查看，每行显示四个时区字符串且以半角逗号（,）分隔。

```
Africa/Abidjan, Africa/Accra, Africa/Addis_Ababa, Africa/Algiers
Africa/Asmara, Africa/Asmera, Africa/Bamako, Africa/Bangui
Africa/Banjul, Africa/Bissau, Africa/Blantyre, Africa/Brazzaville
Africa/Bujumbura, Africa/Cairo, Africa/Casablanca, Africa/Ceuta
Africa/Conakry, Africa/Dakar, Africa/Dar_es_Salaam, Africa/Djibouti
Africa/Douala, Africa/El_Aaiun, Africa/Freetown, Africa/Gaborone
Africa/Harare, Africa/Johannesburg, Africa/Juba, Africa/Kampala
Africa/Khartoum, Africa/Kigali, Africa/Kinshasa, Africa/Lagos
Africa/Libreville, Africa/Lome, Africa/Luanda, Africa/Lubumbashi
Africa/Lusaka, Africa/Malabo, Africa/Maputo, Africa/Maseru
```

Africa/Mbabane, Africa/Mogadishu, Africa/Monrovia, Africa/Nairobi
Africa/Ndjamena, Africa/Niamey, Africa/Nouakchott, Africa/Ouagadougou
Africa/Porto-Novo, Africa/Sao_Tome, Africa/Timbuktu, Africa/Tripoli
Africa/Tunis, Africa/Windhoek, America/Adak, America/Anchorage
America/Anguilla, America/Antigua, America/Araguaina, America/Argentina/Buenos_Aires
America/Argentina/Catamarca, America/Argentina/ComodRivadavia, America/Argentina/Cordoba, America/Argentina/Jujuy
America/Argentina/La_Rioja, America/Argentina/Mendoza, America/Argentina/Rio_Gallegos, America/Argentina/Salta
America/Argentina/San_Juan, America/Argentina/San_Luis, America/Argentina/Tucuman, America/Argentina/Ushuaia
America/Aruba, America/Asuncion, America/Atikokan, America/Atka
America/Bahia, America/Bahia_Banderas, America/Barbados, America/Belem
America/Belize, America/Blanc-Sablon, America/Boa_Vista, America/Bogota
America/Boise, America/Buenos_Aires, America/Cambridge_Bay, America/Campo_Grande
America/Cancun, America/Caracas, America/Catamarca, America/Cayenne
America/Cayman, America/Chicago, America/Chihuahua, America/Coral_Harbour
America/Cordoba, America/Costa_Rica, America/Creston, America/Cuiaba
America/Curacao, America/Danmarkshavn, America/Dawson, America/Dawson_Creek
America/Denver, America/Detroit, America/Dominica, America/Edmonton
America/Eirunepe, America/El_Salvador, America/Ensenada, America/Fort_Nelson
America/Fort_Wayne, America/Fortaleza, America/Glace_Bay, America/Godthab
America/Goose_Bay, America/Grand_Turk, America/Grenada, America/Guadeloupe
America/Guatemala, America/Guayaquil, America/Guyana, America/Halifax
America/Havana, America/Hermosillo, America/Indiana/Indianapolis, America/Indiana/Knox
America/Indiana/Marengo, America/Indiana/Petersburg, America/Indiana/Tell_City, America/Indiana/Vevay
America/Indiana/Vincennes, America/Indiana/Winamac, America/Indianapolis, America/Inuvik
America/Iqaluit, America/Jamaica, America/Jujuy, America/Juneau
America/Kentucky/Louisville, America/Kentucky/Monticello, America/Knox_IN, America/Kralendijk
America/La_Paz, America/Lima, America/Los_Angeles, America/Louisville
America/Lower_Princes, America/Maceio, America/Managua, America/Manaus
America/Marigot, America/Martinique, America/Matamoros, America/Mazatlan
America/Mendoza, America/Menominee, America/Merida, America/Metlakatla
America/Mexico_City, America/Miquelon, America/Moncton, America/Monterrey
America/Montevideo, America/Montreal, America/Montserrat, America/Nassau
America/New_York, America/Nipigon, America/Nome, America/Noronha
America/North_Dakota/Beulah, America/North_Dakota/Center, America/North_Dakota/New_Salem, America/Ojinaga
America/Panama, America/Pangnirtung, America/Paramaribo, America/Phoenix
America/Port-au-Prince, America/Port_of_Spain, America/Porto_Acre, America/Porto_Velho
America/Puerto_Rico, America/Punta_Arenas, America/Rainy_River, America/Rankin_Inlet
America/Recife, America/Regina, America/Resolute, America/Rio_Branco
America/Rosario, America/Santa_Isabel, America/Santarem, America/Santiago
America/Santo_Domingo, America/Sao_Paulo, America/Scoresbysund, America/Shiprock
America/Sitka, America/St_Barthelemy, America/St_Johns, America/St_Kitts
America/St_Lucia, America/St_Thomas, America/St_Vincent, America/Swift_Current
America/Tegucigalpa, America/Thule, America/Thunder_Bay, America/Tijuana
America/Toronto, America/Tortola, America/Vancouver, America/Virgin
America/Whitehorse, America/Winnipeg, America/Yakutat, America/Yellowknife
Antarctica/Casey, Antarctica/Davis, Antarctica/DumontDUrville, Antarctica/Macquarie
Antarctica/Mawson, Antarctica/McMurdo, Antarctica/Palmer, Antarctica/Rothera
Antarctica/South_Pole, Antarctica/Syowa, Antarctica/Troll, Antarctica/Vostok
Arctic/Longyearbyen, Asia/Aden, Asia/Almaty, Asia/Amman
Asia/Anadyr, Asia/Aqtau, Asia/Aqtobe, Asia/Ashgabat
Asia/Ashkhabad, Asia/Atyrau, Asia/Baghdad, Asia/Bahrain
Asia/Baku, Asia/Bangkok, Asia/Barnaul, Asia/Beirut
Asia/Bishkek, Asia/Brunei, Asia/Calcutta, Asia/Chita
Asia/Choibalsan, Asia/Chongqing, Asia/Chungking, Asia/Colombo
Asia/Dacca, Asia/Damascus, Asia/Dhaka, Asia/Dili
Asia/Dubai, Asia/Dushanbe, Asia/Famagusta, Asia/Gaza
Asia/Harbin, Asia/Hebron, Asia/Ho_Chi_Minh, Asia/Hong_Kong
Asia/Hovd, Asia/Irkutsk, Asia/Istanbul, Asia/Jakarta
Asia/Jayapura, Asia/Jerusalem, Asia/Kabul, Asia/Kamchatka
Asia/Karachi, Asia/Kashgar, Asia/Kathmandu, Asia/Katmandu
Asia/Khandyga, Asia/Kolkata, Asia/Krasnoyarsk, Asia/Kuala_Lumpur
Asia/Kuching, Asia/Kuwait, Asia/Macao, Asia/Macau
Asia/Magadan, Asia/Makassar, Asia/Manila, Asia/Muscat
Asia/Nicosia, Asia/Novokuznetsk, Asia/Novosibirsk, Asia/Omsk
Asia/Oral, Asia/Phnom_Penh, Asia/Pontianak, Asia/Pyongyang
Asia/Qatar, Asia/Qostanay, Asia/Qyzylorda, Asia/Rangoon
Asia/Riyadh, Asia/Saigon, Asia/Sakhalin, Asia/Samarkand
Asia/Seoul, Asia/Shanghai, Asia/Singapore, Asia/Srednekolymsk
Asia/Taipei, Asia/Tashkent, Asia/Tbilisi, Asia/Tehran
Asia/Tel_Aviv, Asia/Thimbu, Asia/Thimphu, Asia/Tokyo
Asia/Tomsk, Asia/Ujung_Pandang, Asia/Ulaanbaatar, Asia/Ulan_Bator
Asia/Urumqi, Asia/Ust-Nera, Asia/Vientiane, Asia/Vladivostok
Asia/Yakutsk, Asia/Yangon, Asia/Yekaterinburg, Asia/Yerevan
Atlantic/Azores, Atlantic/Bermuda, Atlantic/Canary, Atlantic/Cape_Verde
Atlantic/Faeroe, Atlantic/Faroe, Atlantic/Jan_Mayen, Atlantic/Madeira
Atlantic/Reykjavik, Atlantic/South_Georgia, Atlantic/St_Helena, Atlantic/Stanley
Australia/ACT, Australia/Adelaide, Australia/Brisbane, Australia/Broken_Hill
Australia/Canberra, Australia/Currie, Australia/Darwin, Australia/Eucla
Australia/Hobart, Australia/LHI, Australia/Lindeman, Australia/Lord_Howe
Australia/Melbourne, Australia/NSW, Australia/North, Australia/Perth
Australia/Queensland, Australia/South, Australia/Sydney, Australia/Tasmania
Australia/Victoria, Australia/West, Australia/Yancowinna, Brazil/Acre
Brazil/DeNoronha, Brazil/East, Brazil/West, CET
CST6CDT, Canada/Atlantic, Canada/Central, Canada/Eastern

Canada/Mountain, Canada/Newfoundland, Canada/Pacific, Canada/Saskatchewan
Canada/Yukon, Chile/Continental, Chile/EasterIsland, Cuba
EET, EST, EST5EDT, Egypt
Eire, Etc/GMT, Etc/GMT+0, Etc/GMT+1
Etc/GMT+10, Etc/GMT+11, Etc/GMT+12, Etc/GMT+2
Etc/GMT+3, Etc/GMT+4, Etc/GMT+5, Etc/GMT+6
Etc/GMT+7, Etc/GMT+8, Etc/GMT+9, Etc/GMT-0
Etc/GMT-1, Etc/GMT-10, Etc/GMT-11, Etc/GMT-12
Etc/GMT-13, Etc/GMT-14, Etc/GMT-2, Etc/GMT-3
Etc/GMT-4, Etc/GMT-5, Etc/GMT-6, Etc/GMT-7
Etc/GMT-8, Etc/GMT-9, Etc/GMT0, Etc/Greenwich
Etc/UCT, Etc/UTC, Etc/Universal, Etc/Zulu
Europe/Amsterdam, Europe/Andorra, Europe/Astrakhan, Europe/Athens
Europe/Belfast, Europe/Belgrade, Europe/Berlin, Europe/Bratislava
Europe/Brussels, Europe/Bucharest, Europe/Budapest, Europe/Busingen
Europe/Chisinau, Europe/Copenhagen, Europe/Dublin, Europe/Gibraltar
Europe/Guernsey, Europe/Helsinki, Europe/Isle_of_Man, Europe/Istanbul
Europe/Jersey, Europe/Kaliningrad, Europe/Kiev, Europe/Kirov
Europe/Lisbon, Europe/Ljubljana, Europe/London, Europe/Luxembourg
Europe/Madrid, Europe/Malta, Europe/Mariehamn, Europe/Minsk
Europe/Monaco, Europe/Moscow, Europe/Nicosia, Europe/Oslo
Europe/Paris, Europe/Podgorica, Europe/Prague, Europe/Riga
Europe/Rome, Europe/Samara, Europe/San_Marino, Europe/Sarajevo
Europe/Saratov, Europe/Simferopol, Europe/Skopje, Europe/Sofia
Europe/Stockholm, Europe/Tallinn, Europe/Tirane, Europe/Tiraspol
Europe/Ulyanovsk, Europe/Uzhgorod, Europe/Vaduz, Europe/Vatican
Europe/Vienna, Europe/Vilnius, Europe/Volgograd, Europe/Warsaw
Europe/Zagreb, Europe/Zaporozhye, Europe/Zurich, GB
GB-Eire, GMT, GMT+0, GMT-0
GMT0, Greenwich, HST, Hongkong
Iceland, Indian/Antananarivo, Indian/Chagos, Indian/Christmas
Indian/Cocos, Indian/Comoro, Indian/Kerguelen, Indian/Mahe
Indian/Maldives, Indian/Mauritius, Indian/Mayotte, Indian/Reunion
Iran, Israel, Jamaica, Japan
Kwajalein, Libya, MET, MST
MST7MDT, Mexico/BajaNorte, Mexico/BajaSur, Mexico/General
NZ, NZ-CHAT, Navajo, PRC
PST8PDT, Pacific/Apia, Pacific/Auckland, Pacific/Bougainville
Pacific/Chatham, Pacific/Chuuk, Pacific/Easter, Pacific/Efate
Pacific/Enderbury, Pacific/Fakaofo, Pacific/Fiji, Pacific/Funafuti
Pacific/Galapagos, Pacific/Gambier, Pacific/Guadalcanal, Pacific/Guam
Pacific/Honolulu, Pacific/Johnston, Pacific/Kiritimati, Pacific/Kosrae
Pacific/Kwajalein, Pacific/Majuro, Pacific/Marquesas, Pacific/Midway
Pacific/Nauru, Pacific/Niue, Pacific/Norfolk, Pacific/Noumea
Pacific/Pago_Pago, Pacific/Palau, Pacific/Pitcairn, Pacific/Pohnpei
Pacific/Ponape, Pacific/Port_Moresby, Pacific/Rarotonga, Pacific/Saipan
Pacific/Samoa, Pacific/Tahiti, Pacific/Tarawa, Pacific/Tongatapu
Pacific/Truk, Pacific/Wake, Pacific/Wallis, Pacific/Yap
Poland, Portugal, ROC, ROK
Singapore, Turkey, UCT, US/Alaska
US/Aleutian, US/Arizona, US/Central, US/East-Indiana
US/Eastern, US/Hawaii, US/Indiana-Starke, US/Michigan
US/Mountain, US/Pacific, US/Samoa, UTC
Universal, W-SU, WET, Zulu

简介

在使用KLog进行日志数据检索、SQL查询分析之前，请先配置索引规则。字段索引规则包括全文索引、字段索引两部分。

索引类型	说明
全文索引	以文本形式为所有字段的Value创建索引，并根据分词符对Value进行分词。
字段索引	配置字段索引后可根据指定字段做数据查询、聚合统计。

说明

- 同时配置全文索引和指定字段索引的情况下，以指定字段索引的配置为准。
- 当某个字段配置了字段索引时，则该字段的全文索引仍会生效。
- 如果字段不配置字段索引，该字段不能进行聚合统计等查询操作。

索引配置

配置索引

1. 登录金山云日志服务Klog控制台。
2. 从控制台进入**项目列表**，点击某一项目名称，进入该项目的项目详情页，点击**日志搜索**，选择某一日志池。

3. 点击页面右上角的**配置索引**，开始配置索引。

- 索引状态

  默认开启状态，开启状态下，支持全文索引和字段查询索引两种；关闭状态下，无法进行全文检索和字段检索，但可以使用SQL查询进行数据筛选。

- 全文索引

参数	说明
中文分词	支持IK中文分词，开启中文分词后，无法自定义英文分词符
区分大小写	关闭区分大小写，查询时关键词不区分大小写；开启区分大小写，查询时关键词区分大小写
分词符	根据指定分词符，将日志数据的Value切分成多个关键词

- 字段查询

参数	说明
字段名称	日志字段名称，必填项
数据类型	必选项，日志字段值的数据类型，支持text、long、double、date和json
别名	选填项，字段的别名，可以使用别名进行数据查询，多个字段的数据别名可以相同，相同情况下，会按照别名进行跨字段查询
区分大小写	默认关闭，关闭区分大小写，查询时关键词不区分大小写；开启区分大小写，查询时关键词区分大小写
分词符	字段数据类型为text时，支持设置分词符对Value进行分词；根据指定的分词符，将日志数据的Value切分成多个关键词

4. 点击**确定**，完成索引属性配置。

   说明

   - 保存索引配置后，索引配置只对保存配置后写入的数据生效。如需查询历史数据，需根据历史索引配置进行查询。

重新获取索引

当实际日志数据发生改变时，之前保存的索引配置已经无法满足当前的数据查询需求，需要重新配置索引规则。

1. 登录金山云日志服务Klog控制台。
2. 从控制台进入**项目列表**，点击某一项目名称，进入该项目的项目详情页，点击**日志搜索**，选择某一日志池。
3. 点击页面右上角的**配置索引**，进入到配置索引页面，点击**重新获取索引**，可获取当前日志数据最新的字段索引属性。
4. 重新获取的索引属性不支持修改，如需修改，可以先点击**引用**，把索引属性自动加载到**配置索引**页面后，再根据业务情况自定义配置索引。
5. 点击**确定**，完成索引配置。

日志搜索

搜索日志

1. 登录金山云日志服务Klog控制台。
2. 从控制台进入**工程列表**，点击某一工程名称，进入该工程的详情页，点击**日志搜索**，进入到日志搜索页面。
3. 选择日志池、查询时间范围，并输入查询语句，并点击**查询**，即可在下方查看到返回的数据结果。
   - 产品支持全文检索、键值对检索和SQL查询，用户可以在输入框中输入检索语句或者SQL语句，通过检索语句检索并返回原始日志；通过SQL语句查询并返回聚合或者筛选后的数据表格。具体详见查询语法。
   - 查询语句中的使用限制详见使用限制。

搜索结果

原始日志

在输入框中输入检索语句后，在“原始日志”可查看日志字段、日志分布直方图和日志内容。

- 日志内容 展示原始日志，点击左侧的下拉按钮，日志数据支持以表格和json方式展示。

- 日志字段 列出了全部日志字段，点击 可切换该日志字段的隐藏和显示。点击下拉按钮，提供字段快速分析功能。
- 日志分布直方图 日志分布直方图主要展示查询到的日志在时间上的分布。鼠标指向蓝色数据块时，可以查看该数据块代表的时间范围和日志命中次数。

实时日志（Tail）

1.背景：实时监控日志队列中的数据，从最新的日志数据中提取出关键信息进而快速地分析出异常原因。而tail命令是实时显示日志文件的最常用解决方案，klog在控制台提供了日志实时监控功能，对线上日志进行实时分析。
2.功能：实时监控日志信息，按照关键词筛选日志数据；结合采集配置，对采集的日志进行索引区分。
3.步骤：①.登录金山云控制台首页。②.点击**工程列表** ，选择工程名称。③.点击**日志搜索**，选择日志池 ，单击Tail按钮。④.在弹出的Tail页面，点击启动按钮，即显示最新数据，可以查看结果。

统计图表

使用SQL语句查询数据后，在“统计图表”查看到数据返回结果，默认显示表格样式。可以修改图表类型（比如折线图,柱状图，饼图等）。具体详见图表。

另存为告警

在查询分析页面上，单击**另存为告警**，可为查询结果设置告警。

添加至仪表盘

仪表盘是日志服务提供的实时数据分析大盘。单击“添加至仪表盘”，将查询语句以图表形式保存到仪表盘中，详情请参见仪表盘。

快速分析

快速分析

快速分析帮助您快速查看某一字段在指定时间内字段值的分布情况。 仅未分词字段支持该功能。

操作步骤 1、登录金山云日志服务Klog控制台。 2、在**工程列表**，点击某一工程名称，进入该工程的详情页，找到对应日志池，点击**搜索** 3、进入日志搜索页面，在原始日志页签的快速分析区域，单击目标字段。

功能说明：

- 提供text、date类型字段分组统计和long、double类型字段的分布直方统计。
- 提供查询分析语句，单击目标字段下的图标，跳转到统计图表页签，自动生成检索语句，可查看字段值分布详细信息。
- 针对long、double类型的字段，分别单击目标字段下的Max、Min、Avg、Sum，快速查找所有项中的最大项、最小项、平均值和总和。

上下文查询

上下文查询

上下文查询支持您在控制台查看指定日志在原始文件（同机器+同文件）中的前若干条日志（上文）或后若干条日志（下文）。在业务故障排查过程中，可快速查找相关故障信息，定位问题。

使用限制

1、如果您使用的采集端是2023.9.19之前的版本，需更新到新版本，下载 2、如果使用API采集，需在Log.contents中，添加id字段，并使用雪花算法生成id值；同时在LogGroup中指定filename、source的参数信息。

操作步骤

1、在日志搜索页，找到目标日志，展开日志，点击上下文查询。

2、在弹出页面上下滚动鼠标，查看指定日志的上下文信息。

- 点击上翻，浏览当前日志更早的日志。
- 点击下翻，浏览当前日志更晚的日志。
- 高亮显示，可实现输入字符串的高亮显示。
- 过滤日志，可实现日志列表中只显示包含过滤字符串的日志。

统计图表

前提条件

已对当前日志池配置索引规则。 **注意**：只有在日志搜索时使用SQL分析语句，才能根据统计结果为您展示图表。

统计图表

1. 登录金山云日志服务Klog控制台。
2. 从控制台进入**项目列表**，点击某一项目名称，进入该项目的项目详情页，点击**日志搜索**，进入日志搜索页面。
3. 输入SQL语句查询数据后，在**统计图表**查看到数据返回结果，默认显示表格样式。可以修改图表类型（比如折线图，柱状图，饼状图等）。

制作图表

折线图： 选择图表类型为**折线图**，在右侧设置X轴、Y轴字段，X轴只需要选中一列字段，Y轴可以选中多个字段，每个字段会对应显示一个折线。

通过类似的操作，还可以制作饼图、柱状图等。

饼状图：

柱状图：

添加至仪表盘

制作完图表后，如需保留图表以便于随时查看该图表，可点击**添加至仪表盘**，填写新仪表盘名称或者选择已有仪表盘，填写图表名称后，即可把当前图表保存到指定仪表盘中。

仪表盘

创建仪表盘

1. 登录金山云日志服务Klog控制台。
2. 从控制台进入**项目列表**，点击某一项目名称，进入该项目的项目详情页，点击**仪表盘**进入到仪表盘列表页。页面显示当前项目下已有的仪表盘。
3. 点击创建仪表盘按钮来创建新的仪表盘。您也可以在日志搜索页面，点击图标右侧的添加至仪表盘来创建新的仪表盘。

修改仪表盘

1. 登录金山云日志服务Klog控制台。
2. 从控制台进入**项目列表**，点击某一项目名称，进入该项目的项目详情页，点击**仪表盘**进入到仪表盘列表页。
3. 点击某一仪表盘名称，进入该仪表盘详情页。点击**修改仪表盘**，该仪表盘进入到编辑状态。

   编辑图表

   鼠标移入某个图表右上角的更多图标，点击**编辑图表**，弹出图表编辑框。填写图表信息后，点击**刷新图表**加载出编辑后的图表效果，点击**确定**完成编辑。

- 说明：编辑图表后，需点击页面右上角的**保存**，当前修改的内容才会保存到仪表盘中

删除图表

如需删除仪表盘中的图表，点击图表右上角的**删除图表**，即可删除当前图表。

简介

日志服务支持为查询或分析结果设置告警。设置告警后，日志服务定期检查查询或分析结果，当检查结果满足预设条件时发送告警通知，实现实时的服务状态监控。

告警限制

日志服务告警相关限制说明如下表所示。

限制项	说明
告警名称	支持2-64个字符，仅允许字母、中文、数字和-_@#.
图表名称	支持小写字母、数字、连字符（-）和下划线（_），需要以数字或小写字母为开头和结尾
检查频率	支持检测周期和检测间隔的自由组合，其中，检测周期可选固定间隔、每天、每周、每小时，而检测间隔则可选择小时/分钟/天
触发条件	支持加(+)、减(-)、乘(*)、除(/)、4种基础运算符和>、>=、<、<=、==、!= 6种比较运算符，&&、2种逻辑运算
通知方式	提供短信和邮件两种通知方式

告警管理

告警列表

1. 登录金山云日志服务Klog控制台。
2. 从控制台进入**项目列表**，点击某一项目名称，进入该项目的项目详情页，点击**告警管理**，进入到告警列表页。

用户可通过筛选仪表盘、告警状态以及输入告警名称来查询指定的告警。

修改告警状态

在告警列表中的状态一栏，点击**禁用**或者**启动**来修改当前告警的状态。 启动状态下，服务会按照执行频率定时执行告警规则。当满足触发告警通知时，以短信、邮件方式实时发送告警消息。 禁用状态下，则不再执行告警规则，也不会触发告警。

新建告警

1. 登录金山云日志服务Klog控制台。
2. 从控制台进入**项目列表**，点击某一项目名称，进入该项目的项目详情页，点击**告警管理**，进入到告警列表页。
3. 在告警列表页，点击**新建告警**，进入到新建告警页面。
   - 告警名称：必填项，名称支持2-64个字符，仅允许字母中文数字-_@#.
   - 仪表盘：必选项，选择当前已有仪表盘
   - 关联图表：基于所选图表的数据内容来配置告警出发规则，支持关联同一仪表盘下的多个图表；每个图表具备一个序列号，例如：第一个图表的序列号为0，第二个图表序列号为1，以此类推。
   - 检查频率：该告警执行数据查询的频率，支持固定间隔和crontab模式。其中，固定间隔支持分钟、小时、天粒度的间隔。

     cron表达式说明如下：
     0/5 从0分钟开始，每隔5分钟发送一次
     0 0/1 *从0点0分开始，每隔1小时发送一次*
     0 18 每天18点0分发送一次
     0 0 1 每月1日0点0分发送一次
   - 触发条件：通过表达式来设置触发告警的条件，支持加(+)、减(-)、乘(*)、除(/)、4种基础运算符和>、>=、<、<=、==、!= 6种比较运算符,&&、|| 2种逻辑运算
   - 触发次数阈值：触发告警的次数上限，当某次执行中查询到满足表达式的数据，累计1次，当累计值达到触发次数阈值时，会触发告警。
   - 通知间隔：发送告警通知的时间间隔。比如设置5分钟间隔时，会每隔5分钟查看当前累计触发次数是否超出阈值，超出时则发送告警。 点击下一步，设置告警通知方式。
   - 选择告警接收组：告警通知会发送给接受组下的用户。如果没有接受组，需先添加联系组。
   - 短信和邮件：选中发送通知方式，并填写报警内容，内容支持使用模板变量。 点击”确定“完成告警创建。

修改告警

修改告警后，会按照修改后的告警规则来执行告警判断。修改告警中各项配置项的逻辑请参照上面的创建告警。

告警详情

点击某一告警名称，进入到告警详情页。

- 告警统计：告警次数是指昨天触发告警的次数之和；通知成功次数是指昨天告警通知发送成功的次数之和；日环比是昨天跟前天的环比。
- 告警历史：告警规则执行的历史记录。

查看告警历史

告警历史

告警历史主要展示已有告警的执行历史记录。

1. 登录金山云日志服务Klog控制台。
2. 从控制台进入**项目列表**，点击某一项目名称，进入该项目的项目详情页，点击**告警管理**，进入到告警列表页。
3. 在告警列表页，点击**告警历史**。

告警历史页面显示当前项目下所有告警规则的执行记录。可通过告警名称、仪表盘名称、时间范围来筛选历史记录。

投递简介

概述

日志服务Klog的投递功能，支持通过控制台将日志池中的数据，定时投递至对象存储KS3、托管kafka等云产品中。

功能限制

不支持历史数据投递。 不支持跨region投递，工程和KS3存储桶、kafka实例需在同一个region。 不支持跨账号投递。

投递到ks3

投递到ks3

前提条件： 1、开通日志服务，创建工程与日志池，并成功采集到日志数据。 2、需开通对象存储KS3服务，并在同region下，已创建KS3存储桶。默认是KS3的标准存储，如需其它存储类型，请在KS3控制台操作，参考存储类型转换。 3、子账号已有主账号授权，授权步骤参考权限管理 4、已授权日志服务角色访问KS3的权限。如果未授权，如图，系统会引导用户完成授权，同意即可。

新建投递任务

步骤： 1、登录日志服务控制台。 2、在左侧导航栏中，单击工程列表。 3、单击工程名称，进入工程详情页面。 4、在左侧导航栏中，点击投递管理，新建投递任务，投递类型选择ks3，填写配置信息参考下表。

配置项	说明	规则	是否必填
任务名称	投递任务名称	以数字或小写字母开头或结尾，支持小写字母、数字、连字符（-）和下划线（_),长度3~63	必填
日志池	选择待投递的日志池	列表选择	必填
ks3存储桶	仅支持同区域的存储桶作为投递目标	列表选择	必填
目录前缀	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。最终投递目录的格式{ks3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，默认为根目录	可选
分区格式	根据strftime时间格式化自动生成目录	strftime格式，如/%Y/%m/%d/%H/	必填

压缩投递	是否对日志文件进行压缩后投递，目前支持的压缩方式有 gzip、lz4	开/关	必填
投递文件大小	指定在该投递时间间隔中，投递文件大小上限（未压缩），超过该上限，将被分成多个日志文件	100MB - 256MB	必填
投递时间间隔	大小和时间满足一个即会触发一次投递	5~15分钟	必填
存储格式	支持CSV、JSON	列表选择	必填

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	说明	规则	是否必填
csv字段	指定写入 CSV 文件的键值（key）字段	多个字段用逗号分隔	必填	
分隔符	CSV文件中各字段间的分隔符	列表选择，可选逗号、竖线、空格、制表符	必填	
转义符	出现与分隔符一样的内容，需要使用转义符转义	列表选择，可选单引号、双引号	必填	
无效字段填充	如果字段不存在，需要填写自定义数据	自定义，默认空	可选	
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	

查看投递任务

在投递管理列表页，点击任务ID/任务名称，即可查看该任务监控信息。

投递到托管kafka

投递到托管kafka

前提条件： 1、开通日志服务，创建工程与日志池，并成功采集到日志数据。 2、同region，同账号下，已创建目标kafka实例。

新建投递任务

步骤： 1、登录日志服务控制台。 2、在左侧导航栏中，单击工程列表。 3、单击工程名称，进入工程详情页面。 4、在左侧导航栏中，点击投递管理，新建投递任务，投递类型选择kafka，填写配置信息参考下表。

配置项	说明	规则	是否必填
任务名称	投递任务名称	以数字或小写字母开头或结尾，支持小写字母、数字、连字符（-）和下划线（_),长度3~63	必填
日志池	待投递的日志池	列表选择	必填
kafka实例	投递目标kafka实例	为保证Klog与kafka实例的联通，后台会自动创建slb和PrivateLink，需勾选允许	必填
kafka topic	目标kafka实例的topic，填写已有kafka topic,如topic不存在,系统会为您自动创建	必填	
投递数据格式	投递数据格式，可选PB和JSON	必填	
压缩格式	对日志文件进行压缩后投递，可选snappy、lz4压缩格式	必填	

查看投递任务

在投递管理列表页，点击任务ID/任务名称，即可查看该任务监控信息。

日志下载

操作步骤

1. 登录金山云日志服务Klog服务台。
2. 从控制台进入**工程列表**，点击某一工程名称。
3. 进入该工程的详情页后，点击**下载任务**，进入下载任务界面。
4. 点击**新建下载**，选择开始时间、结束时间和日志池，确定之后完成创建。目前只支持下载近72h的数据。

5.创建任务后，待状态变为“准备完成”，点击**获取下载地址**，单击URL下载数据。每超过512M（压缩后）会生成一个新的URL文件链接。

检索语法

检索规则

在进行日志搜索前，需先选择日志池、时间范围，以及在输入框中输入检索语句。产品支持全文检索、键值检索和模糊关键字检索。

全文检索

日志服务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。全文检索支持普通查询、短语查询和模糊查询。

- 普通查询：直接输入关键字，或者指定字段和关键字，将返回符合关键字的数据结果。例如 method:GET and status=200 表示查询 method 是 GET 并且 status 等于 200 的日志。
- 短语查询：如果需要查询的关键字中包含检索语法运算符或空格，可以将关键字用双引号（""）包裹，表示将双引号中的内容作为多个关键字查询。例如 msg:"not avaliable" 表示 msg 查询包含关键字 not 和 avaliable 的日志，等价于查询 msg:service and msg:"not" and msg:avaliable。 模糊查询：支持关键字中间或末尾加上模糊查询字符（* 和 ?）。例如 http_user:andr? 表示在所有日志中查找 http_user字段包含以 andr 开头的词的日志。

键值检索

用户可以指定字段名称和字段内容进行查询。对于double和long类型的字段，可以指定数值范围进行查询。例如查询语句为count>5000 AND Status:200，表示查询count值大于5000且Status字段值为200的日志。

运算符语法

语法	语义	示例
key:value	键值搜索格式，默认开启所有字段的索引，其中 value 支持?、*模糊搜索	level:INFO
A AND B	“与”逻辑，返回 A 与 B 的交集结果	level:INFO AND line:165
A OR B	“或”逻辑，返回 A 或 B 的并集结果，如果多个单词间没有语法关键词，默认是 OR 的关系	line:171 OR line:165 或者 line:(171 OR 165)
?	匹配单个字符，用于替代单个字符	line:15?
*	匹配0到多个字符	line:1*
TO	range 用法，如status:[400 TO 499]，或status:[400 TO 499} ’}’为不包含499	timestamp:[0 TO 1640171559877]
>	返回字段下大于某个数值的日志	line:>158
>=	返回字段大于或等于某个数值的日志	line:>=158
<	返回字段小于某个数值的日志	line:<158
<=	返回字段小于或等于某个数值的日志	line:<=158
=	多用于返回字段等于某个数值的日志	line=158
!a:b	返回a字段不包含b的数据	!msg:开始

SQL语法

语法支持

日志服务支持基础的 SELECT 查询，具体查询语法是

```
select_expr [, select_expr] ...
[WHERE where_condition]
[GROUP BY {col_name | expr}, ... ]
[ORDER BY {col_name | expr} [ASC | DESC], ...]
[LIMIT [offset,] row_count]
```

说明：SQL查询语法使用限制详见使用限制。 下面是一个查询语句示例

```
SELECT id,sum(cost) AS result
WHERE price>500
GROUP BY id
```

运算符

比较函数

运算符	含义
<	小于
>	大于
<=	小于或等于
>=	大于或等于
=	等于
<>	不等于
BETWEEN	查询处于两个参数之间的数据
IS NULL or IS NOT NULL	判断参数是否是Null值

逻辑运算函数

运算符	含义
AND	只有左右运算数都是true时，结果才为true
OR	左右运算数任一个为true，结果为true
NOT	右侧运算数为false时，结果才为true

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

数学计算函数

运算符	含义
+	两个参数相加
-	两个参数相减
*	两个参数相乘
/	两个参数相除求整数
%	两个参数相除求余数
log10(x)	返回以10为底，x的对数
round(x)	x四舍五入

聚合函数

运算符	含义	示例
avg(x)	计算x列的算数平均值	select avg(request_time) as avg_times
count(*)	表示所有的行数	select count(*) where method = GET
max(x)	返回最大值	select max(request_time) as max_times
min(x)	返回最小值	select min(request_time) as min_times
sum(x)	返回x列的和	select sum(request_time) as sum_times
stats(x)	返回x列的聚合计算结果，包括avg、max、min、sum等	_
data_histogram(x, interval)	根据指定间隔获取所有匹配的值	select data_histogram(timestamp) where status_code=200
percentiles	指定分位符	percentiles=[0.25 , 0.5 ,0.75]

其他函数

运算符	含义	示例
floor	返回小于或等于指定数值表达式的最大整数，向下取整	select floor(num) as nt
split	按指定符号分割字符串，返回分割后的元素个数	select split(newtype,',') as nt
trim	用来移除掉一个字串中的字头或字尾	select trim(newtype) as nt
log	返回x的自然对数，x相对于基数e的对数	_
log10	返回x的基数为10的对数	-
substring	用来截取字符串中的一部分字符	select floor(floor(substring(time,0,14)/100)/5)*5 as nt
round	返回数字表达式并四舍五入为指定的长度或精度	_
sqrt	用来求给定值的平方根	-
concat_ws	将多个字符串连接成一个字符串，但是可以一次性指定分隔符	_
+	加运算	-
-	减运算	-
*	乘运算	-
/	除运算	-
date_format	用于以不同的格式显示日期或时间数据	_
coalesce	返回最近的一个为非空值的值	select name, coalesce(age2, age1) as myAge
if	用于条件判断	select name, if(age <= 18, 'Y', 'O') as myGender
percentile_ranks	计算当前值按百分比计算所处在百分位位置	select percentile_ranks(age, 3, 8, 12, alias=rankAge)
movingavg	滑动平均或移动平均，预测时间序列	select income / 10 AS myincome, sum(income / 10) AS incomeSum, movingavg(field=incomeSum,window=3,alias=incomeSume_avg)
rollingstd	计算滚动标准差	select income / 10 AS myincome, sum(income / 10) AS incomeSum, rollingstd(field=incomeSum,window=2,alias=incomeSume_avg)
parse	将字符串转换为数字和日期和时间格式	select parse(hobby, '(?\S+)球', 'NOT_MATCH') AS ballType, COUNT(_index)
now	返回当前的日期和时间	select * FROM myindex WHERE time >= date(date_add(date(now()), interval -100 day)) AND time <= now()
date	提取日期或日期/时间表达式的日期部分	select * WHERE time >= date(date_add(date(now()), interval -100 day)) AND time <= now()
date_add	向日期添加指定的时间间隔	select * WHERE time >= date(date_add(date(now()), interval -100 day)) AND time <= now()

查询语法示例

查询说明	示例
术语聚合	SELECT COUNT(*) GROUP BY gender
多重聚合	SELECT * GROUP BY (gender, state, age),(state),(age)
范围聚合	SELECT COUNT (age) GROUP BY range(age, 20 , 25 , 30 , 35 , 40)
日期直方图聚合	SELECT online GROUP BY date_histogram(field='insert_time','interval'='1d','alias'='yourAlias','extended_bounds'='{"min":"1547083500000","max":"1547343000000"}',format='epoch_millis')
日期范围聚合	SELECT online GROUP BY date_histogram(field='insert_time','interval'='1d','alias'='yourAlias','extended_bounds'='{"min":"1547083500000","max":"1547343000000"}',format='epoch_millis')
脚本化指标	SELECT scripted_metric('map_script'='yourMapScript','init_script'='yourInitScript','combine_script'='yourCombineScript','reduce_script'='yourReduceScript')
基本查询和条件	SELECT *ORDER BY balance DESC LIMIT 500 ; SELECT balance, include('Name'), exclude('lastName')*

权限管理

访问控制（ Identity and Access Management ，IAM）是金山云提供的管理用户身份与资源访问权限的基础服务。 可以实现安全且精细化管理金山云云服务和资源的访问。 主账号可以授权子账号访问管理权限，以及日志服务的资源权限。

预设系统策略

日志服务预设两条系统策略，可满足最基本的权限管理需求。进入访问控制 > 策略 > 系统策略，选择日志服务产品，两条系统策略如下：

全读写权限（KsyunKLogDefaultPolicy）：具备日志服务所有功能及所有资源的权限，例如创建日志池、修改索引配置、删除日志池、检索日志、上传日志等。 只读权限（KlogReadOnlyAccess）：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。

自定义策略

通过自定义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池的数据。 按策略语法授权样例：

```
{
    "Version": "2015-11-01",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "klog:List*",
                "klog:Get*",
                "klog:Describe*"
            ],
            "Resource": [
                "krn:ksc:klog::2000003485:logpool/857f4f79-f5f5-42f2-be94-56ad1e9a8dcc"
            ]
        }
    ]
}
```

对接Grafana

本文介绍如何通过Grafana，可视化分析日志服务的日志。

操作步骤

安装 Grafana

安装Grafana 8.0以上版本，详见Grafana 官网文档 。若当前版本低于Grafana 8.0，需进行配置备份和升级，详见Grafana 升级指南

安装klog对接Grafana插件

1、下载数据源插件压缩包 plugin.tar。 2、解压缩plugin.tar。 3、运行install.sh脚本，根据提示输入所需信息即可自动安装。 4、重启grafana。

注意：windows不支持自动安装，需要手动安装。步骤如下：

1、下载数据源插件压缩包 plugin.tar。 2、修改grafana配置，allow_loading_unsigned_plugins = kscklog-dsapp-datasource 。如果有多个插件，使用逗号分隔。 3、解压缩plugin.tar解压缩，将klog-ds-plugin文件夹复制到插件目录下。 4、klog-ds-plugin/datasource 中的二进制文件设置为可执行（ chmod 744 ）。 5、重启grafana。

添加数据源

1、在浏览器中访问地址 http://${GrafanaIP地址}:3000 （默认端口为3000），登录 Grafana。 2、在左侧导航栏中，选择 Data Sources。 3、在Data Sources页面，单击Add data source。 4、选择klog数据源。 5、配置数据源。填入AccessKey和SecretKey以及 Endpoint。点击 Save & test ，如果显示 DataSource Connection OK 则说明数据源添加成功，并且可用。

配置 dashboard

1、在左侧导航栏中，单击 New Dashboard。 2、在Dashboard 页面，单击 Add new panel。 3、选择数据源后，选择工程和日志池，并输入对应的检索分析语句。单击右上角时间刷新，即可查看到请求展示的效果。