ALIBABA CLOUD

阿里云

日志服务

查询与分析

文档版本：20220708

阿里云

法律声明

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

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

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

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

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

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

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.查询概述

日志服务支持秒级查询十亿到千亿级别的日志数据。

基本语法

查询语句和分析语句以竖线（|）分割。查询语句的语法为日志服务专有语法，更多信息，请参见查询语法。

> 注意
> - 查询语句可单独使用，分析语句必须与查询语句一起使用。即分析功能是基于查询结果或全量数据进行的。
> - 如果您需要查询百亿级的日志数据量，您可以反复执行（10次以内）某查询语句获取最终完整的结果。

- 基本语法

```
查询语句|分析语句
```

语句类型	说明
查询语句	查询语句用于指定日志查询时的过滤规则，返回符合条件的日志。 查询语句可以为关键词、数值、数值范围、空格、星号（*）等。 如果为空格或星号（*），表示无过滤条件。更多信息，请参见查询语法。
分析语句	分析语句用于对查询结果或全量数据进行计算和统计。更多信息，请参见分析简介。

- 示例

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

使用限制

限制项	说明	备注
关键词个数	关键词查询时，除布尔逻辑符外的条件个数。每次查询最多30个。	无
字段值大小	单个字段值最大为10 KB，超出部分不参与查询。	如果单个字段长度大于10 KB，有一定几率无法通过关键词查询到日志，但数据仍然是完整的。
操作并发数	单个Project支持的最大查询操作并发数为100个。	例如100个用户同时在同一个Project的各个Logstore中执行查询操作。
返回结果	每次查询时，每页最多显示100条查询结果，您可翻页读取完整的查询结果。	无
单条日志内容显示	由于网页浏览器性能原因，对于超过10,000个字符的日志，日志服务只会对前10,000个字符进行DOM切词处理。	如果超出10,000个字符，控制台会提示“该日志存在超过10,000个字符的日志数据，部分显示上会有降级处理”。
模糊查询	执行模糊查询时，日志服务最多查询到符合条件的100个词，并返回包含这100个词并满足查询条件的所有日志。更多信息，请参见模糊查询。	无

限制项	说明	备注
查询结果排序	默认按照分钟级时间从最新开始展示。	无

操作方式

> 注意 在查询日志前，请确保您已采集到日志和配置索引。索引是一种存储结构，用于对日志数据中的一列或多列进行排序。更多信息，请参见配置索引。

- 控制台方式

  登录日志服务控制台，在目标Logstore的查询和分析页面执行查询操作。具体操作，请参见查询和分析日志。

- API方式

  通过GetLogs和GetHistograms接口执行查询操作。

2.分析简介

日志服务提供分析功能，该功能结合了查询功能和SQL计算功能。本文介绍分析功能的基本语法、使用限制和SQL函数等信息。

> 说明
> - 如果您要使用分析功能，则必须在配置索引时打开对应字段的**开启统计**开关。更多信息，请参见配置索引。 打开**开启统计**开关，可实现数据的秒级分析且无额外费用。
> - 日志服务默认存在保留字段。如果您要分析保留字段，请参见保留字段。

基础语法

查询语句和分析语句以竖线（|）分割。查询语句可单独使用，分析语句必须与查询语句一起使用。即分析功能是基于查询结果或全量数据进行的。

> 说明
> - 分析语句中不需要填写FROM子句和WHERE子句，默认分析当前Logstore中的数据。
> - 不需要在分析语句末尾加分号表示结束。
> - 分析语句中不区分大小写。

- 基本语法

```
查询语句|分析语句
```

语句类型	说明
查询语句	查询条件，可以为关键词、数值、数值范围、空格、星号（*）等。 如果为空格或星号（*），表示无过滤条件。更多信息，请参见查询语法。
分析语句	对查询结果或全量数据进行计算和统计。

- 示例

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

使用限制

限制项	普通实例	独享实例
操作并发数	单个Project支持的最大分析操作并发数为15个。 例如15个用户同时在同一个Project的各个Logstore中执行分析操作。	单个Project支持的最大分析操作并发数为100个。 例如100个用户同时在同一个Project的各个Logstore中执行分析操作。
数据量	单个Shard单次仅支持分析1 GB数据。	单次分析最大支持扫描2000亿行数据。
开启模式	默认开启。	通过开关开启。具体操作，请参见开启SQL独享版。
费用	免费。	根据实际使用的CPU时间付费。

限制项	普通实例	独享实例
数据生效机制	分析功能只对开启统计功能后写入的数据生效。 如果您需要分析历史数据，请对历史数据重建索引。更多信息，请参见重建索引。	分析功能只对开启统计功能后写入的数据生效。 如果您需要分析历史数据，请对历史数据重建索引。更多信息，请参见重建索引。
返回结果	执行分析操作后，默认最多返回100行数据。 如果您需要返回更多数据，请使用LIMIT语法。更多信息，请参见LIMIT子句。	执行分析操作后，默认最多返回100行数据。 如果您需要返回更多数据，请使用LIMIT语法。更多信息，请参见LIMIT子句。
字段值大小	单个字段值最大为16 KB，超出部分不参与分析。	单个字段值最大为16 KB，超出部分不参与分析。
超时时间	分析操作的最大超时时间为55秒。	分析操作的最大超时时间为55秒。
Double类型的字段值位数	Double类型的字段值最多52位。 如果浮点数编码位数超过52位，会造成精度损失。	Double类型的字段值最多52位。 如果浮点数编码位数超过52位，会造成精度损失。

分析函数和语法

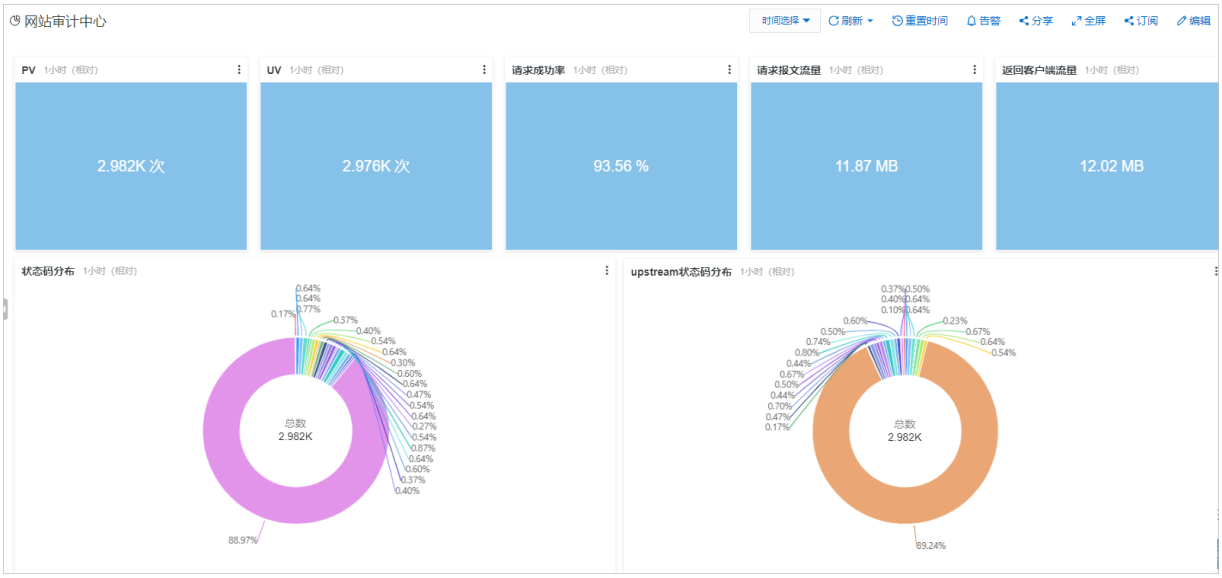
日志服务支持以下分析函数和语法。

- SQL函数
  - 聚合函数
  - 安全检测函数
  - Map映射函数和运算符
  - 估算函数
  - 数学统计函数
  - 数学计算函数
  - 字符串函数
  - 日期和时间函数
  - URL函数
  - 正则式函数
  - JSON函数
  - 类型转换函数
  - IP函数
  - 数组函数和运算符
  - 二进制函数
  - 位运算函数
  - 同比和环比函数
  - 比较运算符
  - Lambda表达式
  - 逻辑运算符
  - 空间几何函数
  - 地理函数
  - 机器学习函数
  - 窗口函数
  - 电话号码函数

- 机器学习函数
  - 平滑函数
  - 多周期估计函数
  - 变点检测函数
  - 极大值检测函数
  - 预测与异常检测函数
  - 序列分解函数
  - 时序聚类函数
  - 频繁模式统计函数
  - 差异模式统计函数
  - URL请求分类函数
  - 根因分析函数
  - 相关性分析函数
  - 核密度估计函数
  - 时序补点函数
  - 异常对比函数
- SQL语法
  - GROUP BY子句
  - ORDER BY子句
  - HAVING子句
  - LIMIT子句
  - UNNEST子句
  - INSERT INTO子句
  - 列的别名
  - 嵌套子查询
  - 条件表达式

分析结果展示

您可以使用仪表盘展示分析结果，如下图所示：



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

3.保留字段

在采集日志或投递数据到其他云产品时，日志服务会将日志来源、时间戳等信息以Key-Value对的形式添加到日志中。这些字段是日志服务的保留字段。本文介绍日志服务的保留字段。

> 注意
>
> - 使用API写入日志数据或添加Logtail配置时，请不要将Key即字段名称设置为这些保留字段，否则可能会造成字段名称重复、查询不精确等问题。
> - 所有`__tag__`前缀的字段均不支持投递。
> - 日志服务为日志数据增加的字段按照按量付费方式正常收费，为其开启索引时也会产生少量索引流量及存储费用。更多信息，请参见按量付费。

保留字段	数据格式	索引与统计设置	说明
`__time__`	整型，Unix标准时间格式。	• 索引设置：`__time__`通过API中的参数from和to选择，无需添加该字段的索引。 • 统计设置：当您为任何一列开启统计后，日志服务默认为`__time__`开启统计。	写入日志数据时指定的日志时间。该字段可用于日志投递、查询、分析。
`__source__`	字符串格式。	• 索引设置：开启索引后，日志服务默认为`__source__`创建索引，索引数据类型为text类型，分词字符为空。查询时输入`source:127.0.0.1`或者`__source__:127.0.0.1`。 • 统计设置：当您为任何一列开启统计后，日志服务默认为`__source__`开启统计。	日志来源设备。该字段可用于日志投递、查询、分析、自定义消费。
`__topic__`	字符串格式。	• 索引设置：开启索引后，日志服务默认为`__topic__`创建索引，索引数据类型为text类型，分词字符为空。查询时输入`__topic__:XXX`。 • 统计设置：当您为任何一列开启统计后，日志服务默认为`__topic__`开启统计。	日志主题（Topic）。如果您设置了日志主题，日志服务会自动为您的日志添加日志主题字段，Key为`__topic__`，Value为您的主题内容。该字段可用于日志投递、查询、分析、自定义消费。更多信息，请参见日志主题。
`__partition_time__`	字符串格式。	日志内容中不存在该字段，无需设置索引。	投递MaxCompute的日志分区时间列，由`__time__`计算得到。该字段可用于日志投递MaxCompute时设置日期格式分区列。更多信息，请参见投递日志到MaxCompute（旧版）。
`__extract_others__`	字符串格式，可反序列化成JSON Map。	日志内容中不存在该字段，无需设置索引。	日志中投递MaxCompute的未配置字段组装为一个JSON Map。该字段可用于日志投递MaxCompute时打包其它未单独配置的字段。更多信息，请参见投递日志到MaxCompute（旧版）。

保留字段	数据格式	索引与统计设置	说明
`_extract_others_`	字符串格式，可反序列化成JSON Map。	日志内容中不存在该字段，无需设置索引。	与 `__extract_others__` 相同，建议使用 `__extract_others__` 。
`__tag__:__client_ip__`	字符串格式。	• 索引设置：开启索引后，日志服务默认为所有字段创建索引，索引数据类型为text类型，分词字符为空，在查询时要完全命中，或采用模糊查询。 • 统计设置：默认没有为该列开启统计。如需开启统计，请手动添加 `__tag__:__client_ip__` 的索引，并开启统计功能。	日志来源设备的公网IP。该字段为系统标签（Tag）。开启记录外网IP功能后，服务端接收日志时为原始日志追加该字段。可用于日志查询、分析、自定义消费。对该字段进行SQL分析时，需要给该字段加上双引号。更多信息，请参见标签（Tags）和记录外网IP。
`__tag__:__receive_time__`	字符串，可转换为整型的Unix标准时间格式。	• 索引设置：开启索引后，日志服务默认为所有标签（Tag）创建索引，索引数据类型为text类型，分词字符为空，在查询时要完全命中，或采用模糊查询。 • 统计设置：默认没有为该列开启统计。如需开启统计，请手动添加 `__tag__:__receive_time__` 的索引，并开启统计功能。	日志到达服务端的时间，该字段为系统标签（Tag）。开启记录外网IP功能后，服务端接收日志时为原始日志追加该字段。该字段可用于日志查询、分析、自定义消费。更多信息，请参见标签（Tags）和记录外网IP。
`__tag__:__path__`	字符串格式。	• 索引设置：开启索引后，日志服务默认为 `__tag__:__path__` 创建索引，索引数据类型为text类型，分词字符为空。查询时输入 `__tag__:__path__:XXX` 。 • 统计设置：默认没有为该列开启统计。如需开启统计，请手动添加 `__tag__:__path__` 的索引，并开启统计功能。	Logtail采集的日志文件路径，Logtail为日志自动填加该字段。可用于日志查询、分析、自定义消费。对该字段进行SQL分析时，需要给该字段加上双引号。
`__tag__:__hostname__`	字符串格式。	• 索引设置：开启索引后，日志服务默认为 `__tag__:__hostname__` 创建索引，索引数据类型为text类型，分词字符为空。查询时输入 `__tag__:__hostname__:XXX` 。 • 统计设置：默认没有为该列开启统计。如需开启统计，请手动添加 `__tag__:__hostname__` 的索引，并开启统计功能。	logtail采集数据的来源机器主机名。Logtail为日志自动填加该字段。可用于日志查询、分析、自定义消费。对该字段进行SQL分析时，需要给该字段加上双引号。
`__raw_log__`	字符串格式。	请手动添加并设置该字段的索引，索引数据类型为text，并根据需求选择是否开启统计。	解析失败的原始日志。关闭丢弃解析失败日志功能后，Logtail在解析日志失败时上传原始日志。其中Key为 `__raw_log__` 、Value为日志内容。该字段可用于日志投递、查询、分析、自定义消费。更多信息，请参见丢弃解析失败日志。

保留字段	数据格式	索引与统计设置	说明
`__raw__`	字符串格式。	请手动添加并设置该字段的索引，索引数据类型为text，并根据需求选择是否开启统计。	解析成功的原始日志。开启上传原始日志功能后，Logtail会将原始日志作为`__raw__`字段，和解析后的日志一并上传。该字段可用于审计、合规审查等场景。该字段可用于日志投递、查询、分析、自定义消费。更多信息，请参见上传原始日志。

4.数据类型

您在建立索引时，可将字段的数据类型设置为text、long、double或JSON。本文介绍各个数据类型的配置示例及注意事项。

text类型

如果您要查询和分析字符串类型的字段，需在配置索引时，将字段的数据类型设置为text，并开启统计功能。

> 说明 开启全文索引后，日志服务默认将整条日志（除__time__以外所有字段）设置为text类型。

- 日志样例

- 配置索引

- 查询和分析语句
  - 查询非GET请求的日志。

    ```
    not request_method : GET
    ```

  - 查询以cn开头的日志。

    ```
    cn*
    ```

- 统计客户端分布情况。

```
* | SELECT ip_to_province(client_ip) as province, count(*) AS pv GROUP BY province ORDER BY pv
```

long和double类型

设置字段的数据类型为long或double后，您才能通过数值范围查询该字段的值。

- 如果日志字段的值为整数类型，建议您在配置索引时，将字段的数据类型设置为long。
- 如果日志字段的值为浮点数类型，建议您在配置索引时，将字段的数据类型设置double。

> 注意
>
> - 如果设置数据类型为long，而实际字段值为浮点数类型，则无法查询该字段。
> - 如果设置数据类型为long或double，而实际字段值为字符串类型，则无法查询该字段。
> - 如果设置数据类型为long或double，则不支持使用星号（*）或半角问号（?）进行模糊查询。
> - 如果字段的值为非法的数值，则使用not key > -1000000语句进行查询，表示查询所有有效数值之外的日志，其中-1000000为足够小的值即可。

- 日志样例

- 配置索引

- 查询和分析语句

- 查询请求时间大于60秒的日志。

```
request_time > 60
```

- 查询请求时间大于等于60秒，并且小于200秒的日志。

```
request_time in [60 200)
```

```
request_time >= 60 and request_time < 200
```

- 查询请求状态码为200的日志。

```
status = 200
```

JSON类型

如果字段的值为JSON类型，您可在配置索引时，将字段的数据类型设置为JSON。

- 针对JSON对象中的字段，您可根据其值，将数据类型设置为long、double或text，并开启统计功能。开启统计功能后，日志服务支持您查询和分析JSON对象中的字段。
- 针对非完全合法的JSON数据，日志服务支持解析合法部分。

  例如以下为非完整的JSON日志，日志服务可正确解析conctent.remote_addr字段、content.request.request_length字段和content.request.request_method字段。

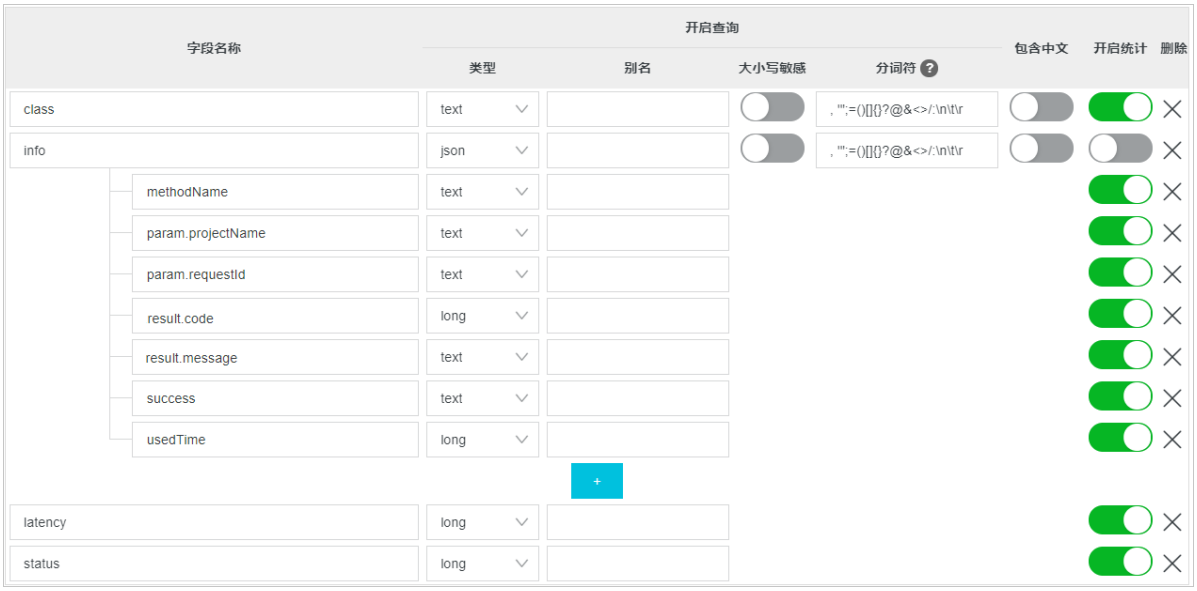
```
content: {
     remote_addr:"192.0.2.0"
     request: {
             request_length:"73"
             request_method:"GE
```

> 注意
>
> - 日志服务支持JSON对象中的叶子节点建立索引，但不支持包含叶子节点的子节点建立索引。
> - 日志服务不支持值为JSON数组的字段建立索引，也不支持JSON数组中的字段建立索引。
> - 如果字段的值为Boolean类型，则您可以在建立索引时，将字段的数据类型设置为text。
> - 查询和分析语句格式为 `查询语句|分析语句` 。在分析语句中，您必须使用双引号（""）包裹字段名称，使用单引号（''）包裹字符串。

- 日志样例

  JSON日志样例如下所示，除日志服务保留字段外，还包括class字段、latency字段、status字段和info字段。其中info字段的值是JSON对象，并存在多层嵌套。

- 配置索引



相关说明如下：

- IP字段和data字段的值为JSON数组，所以您无法为IP字段和data字段建立索引，也无法通过这两个字段进行查询和分析。
- region字段和CreateTime字段在JSON数组中，所以您无法为region字段和CreateTime字段建立索引，也无法通过这两个字段进行查询和分析。

- 查询和分析语句

- 查询usedTime字段的值大于60秒的日志。

```
info.usedTime > 60
```

- 查询success字段的值为true的日志。

```
info.success : true
```

- 查询usedTime字段的值大于60秒且projectName的值不为project01的日志。

```
info.usedTime > 60 not info.param.projectName : project01
```

- 计算获取Project信息的平均时长。

```
methodName = getProjectInfo | SELECT avg("info.usedTime") AS avg_time
```

5.配置索引

索引是一种存储结构，用于对日志数据中的一列或多列进行排序。您只有配置索引后，才能进行查询和分析操作。不同的索引配置，会产生不同的查询和分析结果，请根据您的需求，合理配置索引。如果您同时配置了全文索引和字段索引，以字段索引的配置为准。

前提条件

已采集日志。更多信息，请参见数据采集。

> 注意
>
> - 开启索引后会产生索引流量和索引存储空间，费用说明请参见计费项。
> - 配置索引后，只对新写入的日志数据生效。如果您要查询和分析历史数据，请使用重建索引功能。具体操作，请参见重建索引。
> - 日志服务默认已为部分保留字段配置索引。更多信息，请参见保留字段。其中 `__topic__` 和 `__source__` 的索引分词符为空，查询这两个字段时，查询关键字必须完全匹配。
> - 以__tag__为前缀的字段不支持全文索引，需要您要查询对应的内容，必须设置字段索引。

索引类型

日志服务的索引类型如下：

索引类型	说明
全文索引	日志服务根据您设置的分词符将整条日志拆分成多个词并构建索引。在查询时，字段名称（KEY）和字段值（Value）都是普通文本。例如查询语句 `error` ，表示查询包含 `error` 关键字的日志。
字段索引	配置字段索引后，您可以指定字段名称和字段值（Key:Value）进行查询，缩小查询范围。例如查询语句 `level:error` ，表示查询 `level` 字段值包含 `error` 的日志。 如果您要使用分析功能，必须配置字段索引且开启对应字段的统计功能。开启统计功能不会产生额外的索引流量和索引存储空间。

配置全文索引

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 进入索引配置页面。
   - 如果您还未开启索引，请在Logstore的查询和分析页面，单击**开启索引**。

   - 如果您已开启索引，请在Logstore的查询和分析页面，选择**查询分析属性 > 属性**。

5. 在**查询分析**面板中，配置如下参数，然后单击**确定**。

> **说明** 配置索引后，在1分钟之内生效。

参数	说明
日志聚类	打开**日志聚类**开关后，在采集文本日志时日志服务会自动聚合相似度高的日志，提取共同的日志模式，帮助您快速掌握日志整体情况。更多信息，请参见日志聚类。
全文索引	打开**全文索引**开关，表示开启全文索引功能。
大小写敏感	查询时是否区分英文字母的大小写。 ○ 打开**大小写敏感**开关，则查询时区分大小写。例如某条日志含有 `internalError` ，那么您只能使用 `internalError` 才能查询到该日志。 ○ 关闭**大小写敏感**开关，则查询时不区分大小写。例如某条日志含有 `internalError` ，那么您使用关键字 `INTERNALERROR` 和 `internalerror` 都能查到该日志。
包含中文	查询时是否区分中英文。 ○ 打开**包含中文**开关后，如果日志中包含中文，则按照中文语法拆分中文内容，按照分词符配置拆分英文内容。 > **注意** 中文分词对写入速度会有一定影响，请根据需求谨慎设置。 ○ 关闭**包含中文**开关后，按照分词符配置拆分所有内容。 例如日志内容为**user:SLS日志服务用户张先生**。 ○ 关闭**包含中文**开关后，按照分词符半角冒号（:）进行拆分，日志会被拆分为 `user` 、 `SLS日志服务用户张先生` ，您可以通过 `user` 或 `SLS日志服务用户张先生` 查找该日志。 ○ 打开**包含中文**开关后，日志服务后台分词器将日志拆分为 `user` 、 `SLS` 、 `日志服务` 、 `用户` 和 `张先生` ，您通过 `日志服务` 或 `张先生` 等词都可以查找到该日志。
分词符	根据指定分词符，将日志内容拆分成多个词。分词符包括 `, '";=()[]{}?@&<>/:\n\t\r` ，其中 `\n` 表示换行符， `\t` 表示制表符， `\r` 表示回车符。 例如日志内容为 `/url/pic/abc.gif` 。 ○ 如果不设置任何分词符，整条日志被作为一个词 `/url/pic/abc.gif` ，您只能通过完整字符串 `/url/pic/abc.gif` 或模糊查询 `/url/pic/*` 查找该日志。 ○ 如果设置分词符为正斜线（/），则原始日志被拆分为 `url` 、 `pic` 和 `abc.gif` 三个词，您通过任意一个词或词的模糊查询都可以找到该日志，例如 `url` 、 `abc.gif` 、 `pi*` 、 `/url/pic/abc.gif` 。 ○ 如果设置分词符为正斜线（/）和半角句号（.），则原始日志被拆分为 `url` 、 `pic` 、 `abc` 和 `gif` 四个词。
统计字段（text）最大长度	日志服务默认字段值的最大长度为2048字节，即2 KB。如果您需要修改字段值的最大长度，可设置**统计字段（text）最大长度**，取值范围为64~16384字节。 > **注意** 当单个字段值长度超过最大长度时，超出部分被截断，不参与分析。

配置字段索引

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 进入索引配置页面。
   - 如果您还未开启索引，请在Logstore的查询和分析页面，单击**开启索引**。

   - 如果您已开启索引，请在Logstore的查询和分析页面，选择**查询分析属性 > 属性**。

5. 在**查询分析**面板中，配置如下参数，然后单击**确定**。

> **说明** 配置索引后，在1分钟之内生效。

参数	说明
字段名称	日志字段名称（KEY），例如client_ip。 **说明** - 如果配置公网IP地址、Unix时间戳等Tag字段的索引，请将**字段名称**配置为__tag__:KEY形式，例如：__tag__:__receive_time__。更多信息，请参见保留字段。 - Tag字段不支持数值类型索引，请将所有Tag字段的索引的**类型**配置为**text**。
类型	日志字段值（Value）的数据类型，可选值为text、long、double和json。更多信息，请参见数据类型。 **说明** long类型和double类型不支持设置**大小写敏感**、**包含中文**和**分词符**。
别名	字段的别名，例如ip。 别名仅用于分析语句，查询时仍需使用原始字段名称。更多信息，请参见列的别名。
大小写敏感	查询时是否区分英文字母大小写。 - 打开**大小写敏感**开关，则查询时区分大小写。例如某条日志含有`internalError`，那么您只能使用`internalError`才能查询到该日志。 - 关闭**大小写敏感**开关，则查询时不区分大小写。例如某条日志含有`internalError`，那么您使用关键字`INTERNALERROR`和`internalerror`都能查到该日志。

参数	说明
分词符	根据指定分词符，将日志内容拆分成多个词。分词符包括 `, '";=()[]{}?@&<>/:\n\t\r` ，其中 `\n` 表示换行符， `\t` 表示制表符， `\r` 表示回车符。 例如日志内容为 `/url/pic/abc.gif` 。 ○ 如果不设置任何分词符，整条日志被作为一个词 `/url/pic/abc.gif` ，您只能通过完整字符串 `/url/pic/abc.gif` 或模糊查询 `/url/pic/*` 查找该日志。 ○ 如果设置分词符为正斜线（/），则原始日志被拆分为 `url` 、 `pic` 和 `abc.gif` 三个词，您通过任意一个词或词的模糊查询都可以找到该日志，例如 `url` 、 `abc.gif` 、 `pi*` 、 `/url/pic/abc.gif` 。 ○ 如果设置分词符为正斜线（/）和半角句号（.），则原始日志被拆分为 `url` 、 `pic` 、 `abc` 和 `gif` 四个词。
包含中文	查询时是否区分中英文。 ○ 打开**包含中文**开关后，如果日志中包含中文，则按照中文语法拆分中文内容，按照分词符配置拆分英文内容。 注意 中文分词对写入速度会有一定影响，请根据需求谨慎设置。 ○ 关闭**包含中文**开关后，按照分词符配置拆分所有内容。 例如日志内容为**user:SLS日志服务用户张先生**。 ○ 关闭**包含中文**开关后，按照分词符半角冒号（:）进行拆分，日志会被拆分为 `user` 、 `SLS日志服务用户张先生` ，您可以通过 `user` 或 `SLS日志服务用户张先生` 查找该日志。 ○ 打开**包含中文**开关后，日志服务后台分词器将日志拆分为 `user` 、 `SLS` 、 `日志服务` 、 `用户` 和 `张先生` ，您通过 `日志服务` 或 `张先生` 等词都可以查找到该日志。
开启统计	打开**开启统计**功能后，您才能使用分析功能。
统计字段（text）最大长度	日志服务默认字段值的最大长度为2048字节，即2 KB。如果您需要修改字段值的最大长度，可设置**统计字段（text）最大长度**，取值范围为64~16384字节。 注意 当单个字段值长度超过最大长度时，超出部分被截断，不参与分析。

自动更新索引

当Logstore为云产品专属Logstore或内部Logstore时，默认打开索引**自动更新**开关，后续如有版本更新时可以升级到内置索引最新版本。

注意 删除云产品专属Logstore的索引会影响相关报表、告警等功能的使用。

如果您要自定义设置索引，请在**查询分析**面板中，关闭**自动更新**开关。

索引流量说明

配置索引后，会产生索引流量。

索引类型	说明
全文索引	所有字段名和字段值都将作为text类型存储，即字段名和字段值都被计入在索引流量中。

字段索引	不同数据类型的字段的索引流量计算方式不同。
	• text类型：字段名和字段值都被计入在索引流量中。
	• long类型和double类型：字段名不计入在索引流量中，每个字段值所占的索引流量统一为8字节。
	例如对status字段设置了索引（long类型），字段值为400，则字符串status不会被计入在索引流量中，400的索引流量统一为8字节。
	• JSON类型：字段名和字段值都被计入到索引流量中，包括未被配置索引的子节点。更多信息，请参见如何计算JSON类型字段的索引流量。
	  ○ 如果未对子节点设置索引，则其索引流量按照text类型进行计算。
	  ○ 如果对子节点设置了索引，则其索引流量按照其子节点的数据类型（text类型、long类型或double类型）进行计算。

6.查询和分析日志

配置索引后，您可以在查询分析页面对采集到的日志进行实时查询分析。

前提条件

- 已采集到日志数据。更多信息，请参见数据采集。
- 已配置索引。更多信息，请参见配置索引。

查询和分析

> **说明** 默认情况下，您打开查询分析页面时，系统自动执行查询操作，展示查询结果。您可以单击页面右上角的图标，在**查询设置**页签下，设置查询时间或关闭该功能。

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在输入框中输入查询分析语句。

   查询分析语句由查询语句和分析语句构成，格式为查询语句|分析语句，查询分析语句语法请参见查询语法、SQL分析语法。

   您还可以通过Data Explorer构建查询和分析语句。具体操作，请参见通过Data Explorer构建查询和分析语句。
5. 单击**15分钟（相对）**，设置查询分析的时间范围。

   您可以设置相对时间、整点时间和自定义时间。 此处设置的查询时间最小粒度为分钟。如果需要精确到秒，请在分析语句中指定时间范围，例如 `* | SELECT * FROM log WHERE __time__>1558013658 AND __time__< 1558013660` 。

   > **说明** 查询和分析结果有1分钟以内的误差。
6. 单击**查询/分析**，查看查询分析结果。

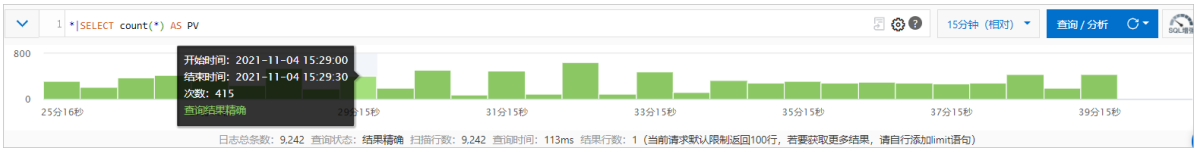
操作查询和分析结果

日志服务为您提供日志分布直方图、原始日志和统计图表形式的展示查询分析结果，并支持设置告警、快速查询等操作。

> **说明** 执行查询和分析语句后，默认只返回100条结果，您可以使用LIMIT语句控制返回结果数量。更多信息，请参见LIMIT子句。

- 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。

  - 鼠标指向绿色数据块时，可以查看该数据块代表的时间范围和日志命中次数。
  - 单击绿色数据块，可以查看更细时间粒度的日志分布，同时在**原始日志**页签中同步展示指定时间范围内的查询结果。
- 原始日志

在**原始日志**页签中展示当前查询结果，您可单击**表格**或**原始**查看日志并可执行如下操作。

- 快速分析：用于快速分析某一字段在一段时间内的分布情况。更多信息，请参见快速分析。

  您还可以单击图标，选择显示Key或Key的别名，该别名可在创建索引时配置。例如host_name的别名为host，如果你选择显示别名，则在快速分析列表中显示host。

  > **说明** 当某字段没有别名时，您选择显示别名，在快速分析列表中仍显示字段名（Key）。

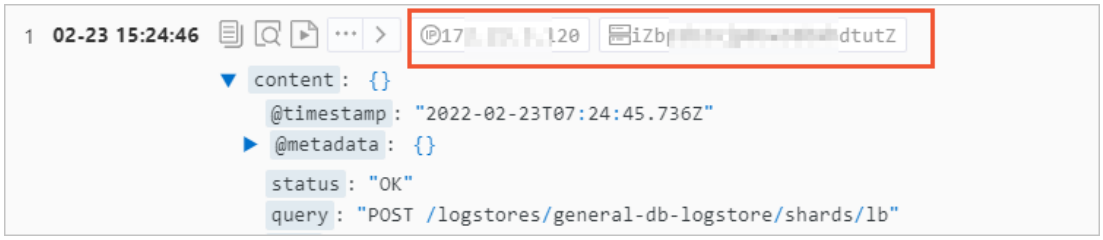
- 上下文浏览：在**原始**页签中，单击目标日志中的图标，查看指定日志在原始文件中的上下文信息。更多信息，请参见上下文查询。

  > **说明** 上下文浏览功能仅支持Logtail采集到的日志数据。

- LiveTail：在**原始**页签中，单击目标日志中的图标，实时监控日志内容，提取关键日志信息。更多信息，请参见LiveTail。

  > **说明** LiveTail功能仅支持Logtail采集到的日志数据。

- 设置Tag：在**原始**页签中，单击图标下的**Tag设置**，将次要的字段内容简化展示。



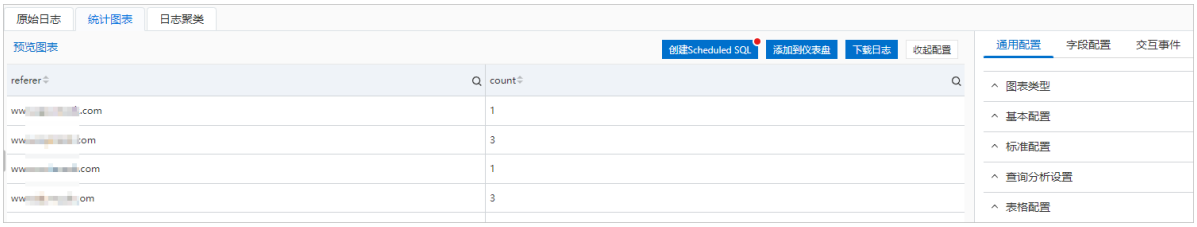
- 设置列：在**表格**页签中，单击图标下的**列设置**，设置表格中要展示的日志信息，其中列名称为字段名，内容为字段值。

- 设置JSON：在**表格**或**原始**页签中，单击图标下的**JSON设置**，设置JSON展开级别。

- 设置事件：在**表格**或**原始**页签中，单击图标下的**事件配置**，为原始日志设置事件。更多信息，请参见事件配置。
- 下载日志：在**表格**或**原始**页签中，单击图标下载日志，支持选择下载范围和下载工具。更多信息，请参见下载日志。

- 统计图表

  执行查询分析语句后，您可以在**统计图表**页签中查看可视化的查询分析结果。



  - 查看查询分析结果：统计图表是日志服务根据查询与分析语句渲染出的结果。日志服务提供表格、线图、柱状图等多种图表类型。目前，统计图表包括Pro版本和普通版本。更多信息，请参见统计图表（Pro版本）概述、统计图表概述。
  - 添加图表到仪表盘：仪表盘是日志服务提供的实时数据分析大盘。单击**添加到仪表盘**，将查询分析结果以图表形式保存到仪表盘中。更多信息，请参见可视化概述。
  - 设置交互事件：交互事件是数据分析中不可缺少的功能之一，通过改变数据维度的层次、变换分析的粒度从而获取数据中更详尽的信息。更多信息，请参见交互事件。

- 日志聚类

  在**日志聚类**页签中，单击**开启日志聚类**，可实现在采集日志时将相似度高的日志聚合。更多信息，请参见日志聚类。

- 告警

  在查询分析页面上，选择**另存为告警 > 新版告警**，可为查询分析结果设置告警。更多信息，请参见快速设置日志告警。

- 快速查询

  在查询分析页面上，单击**另存为快速查询**，将某一查询分析语句保存为快速查询。更多信息，请参见快速查询。

7.通过Data Explorer构建查询和分析语句

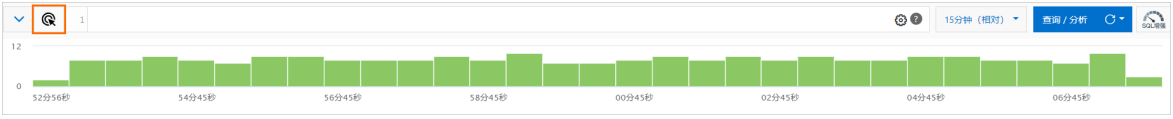
日志服务提供Data Explorer功能，帮助您简单、快速地构建查询和分析语句。

前提条件

已配置索引。更多信息，请参见配置索引。

功能入口

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在查询框中，单击图标。



通过Data Explorer设置查询和分析条件后，您可以单击图标，查看对应的查询和分析语句，也可以单击</>图标返回查询框面板，查询框中将显示对应的查询和分析语句。

构造查询语句

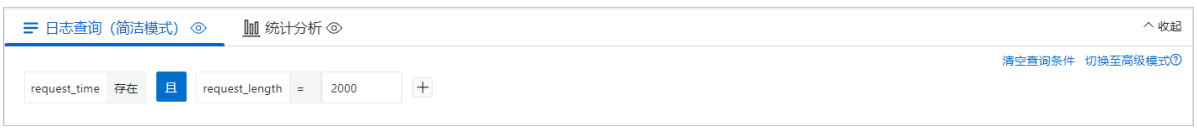
您可以在**日志查询**页签中，构建查询语句。

使用模式

您可以在**日志查询**页签中，通过简洁模式或高级模式配置查询条件。

- 简洁模式

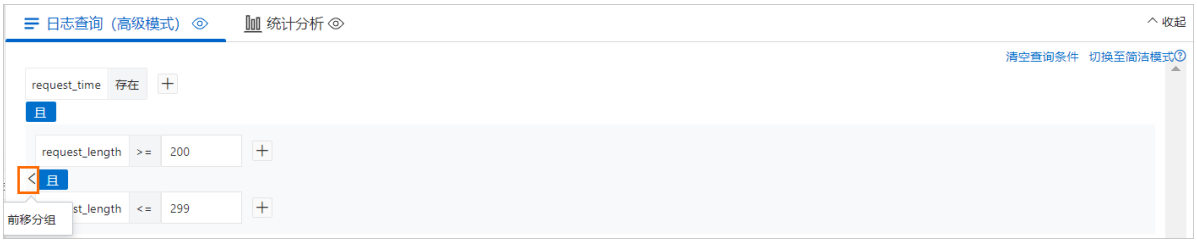
  简洁模式中的多个查询条件是平铺展示的，各个查询条件之间为同级关系。



- 高级模式

高级模式中的多个查询条件是换行展示的，各个查询条件之间可设置层级关系，即对应于查询语句中的括号运算符。

您可以单击**且**、**或**两侧的<图标或>图标，定义各个查询条件之间的层级关系。



逻辑连接符

查询功能支持如下逻辑连接符。

逻辑连接符	说明
且	对应于查询语法中的and运算符。
或	对应于查询语法中的or运算符。
缩进	在高级模式下，您可以使用缩进设置各个查询条件之间的层级关系。对应于( )运算符。
所有值	为一个字段设置了多个查询值时，会出现此逻辑连接符。对应于and运算符。
任意值	为一个字段设置了多个查询值时，会出现此逻辑连接符。对应于or运算符。

运算符

查询功能支持如下运算符。更多信息，请参见运算符。

- 针对全文查询，提供包含、不包含运算符。
- 针对text类型的字段，提供包含、不包含、（字段）存在、（字段）不存在运算符。
- 针对long类型或double类型的字段，提供=、!=、>、<、>=、<=、（字段）存在、（字段）不存在运算符。

构造分析语句

您可以在**统计分析**页签中，构建分析语句。

场景

统计分析功能支持如下场景。

统计分析类型	统计分析场景	说明
基础分析	字段筛选&过滤	支持如下操作。 • 筛选及重命名字段。 • 按条件过滤结果。 • 对结果进行排序。 • 限制返回结果条数。
基础统计	指标统计	统计一个或多个指标，例如日志条数、最大值、最小值、平均值、随机值、方差等。
	分组统计	按一个或多个字段进行分组，分别统计每个分组的指标。
	日志占比	统计满足特定条件的日志数量及其占比。

统计分析类型	统计分析场景	说明
高级统计	Top N	统计目标字段取值频率最高的N个值，并计算每个值的出现频率及百分占比。
	Rare N	统计目标字段取值频率最低的N个值，并计算每个值的出现频率及百分占比。
	IP分布	统计IP地址所属国家、省份、城市、运营商或者内外网的分布情况（数量及占比）。
	时间趋势	按特定时间粒度计算指标，统计指标随时间的变化趋势。 例如按照每分钟的时间粒度计算请求时间的平均值。
	同环比	对比目标字段值相较于特定时间周期之前的变化情况。

单场景统计分析

例如分析网站访问日志，统计过去1天内请求客户端地域分布情况。其中，日志中的client_ip字段记录客户端的IP地址。针对此需求，您只需选择**IP分布**场景，然后通过client_ip字段进行统计。

1. 选择分析场景。

   在**请选择统计分析场景**区域，单击**IP分布**。

2. 设置需分析的指标。

3. 设置查询分析时间，然后单击**查询/分析**。

   执行查询分析操作后，您可以在**统计图表**中查看结果。

多场景统计分析

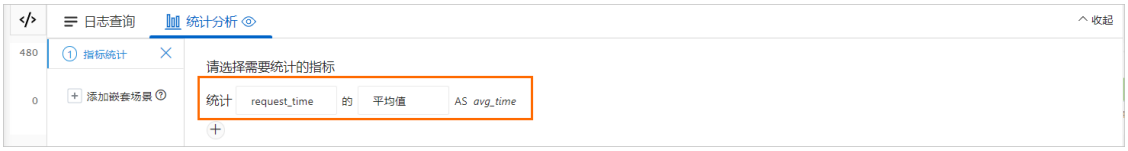
您也可以对多个场景进行组合嵌套使用，来完成复杂的统计分析场景。

> **说明** 在多场景嵌套时，后一个场景基于前一个场景的统计分析结果进行统计。

例如分析网站访问日志，统计过去4小时内所有请求的平均耗时，以及同比昨天同一时间段的变化情况。其中，日志中的request_time字段记录了每个请求的耗时。针对此需求，您只需组合指标统计和同环比两个场景，然后通过request_time字段进行分析。

1. 设置**指标统计**场景相关信息。
   i. 单击**指标统计**。
   ii. 选择**数学计算 > 平均值**。

   iii. 选择request_time字段，并将其分析结果列命名为avg_time。



2. 设置**同环比**场景相关信息。
   i. 单击**同环比**。
   ii. 选择avg_time字段。

3. 设置查询分析时间，然后单击**查询/分析**。

   执行查询分析操作后，您可以在**统计图表**中查看结果。

更多信息

Data Explorer案例

8.开启SQL独享版

SQL独享版是日志服务提供的计费资源，用于SQL分析。SQL独享版针对日志服务免费的SQL分析功能存在的操作并发数限制和分析数据量限制进行了优化。

背景信息

当您在使用SQL分析时，如果数据量较大，日志服务无法在一次查询中完整扫描这个时间段内的所有日志。为了快速返回结果，日志服务限制了每个Shard扫描的数据量，先返回部分不精确的结果。针对该问题，推荐您增加Shard数量，以增加计算资源。但该方法存在如下问题：增加Shard后只对新写入的数据生效，不能解决旧数据的读取问题。另外，增加Shard会影响数据消费，导致消费的客户端过多。

现在，日志服务推出SQL独享版，用于SQL分析。SQL独享版为计费资源，支持更强大的SQL分析；SQL普通版为免费资源，存在更多的资源限制。更多信息，请参见使用限制。

> **说明** SQL独享版和普通版同时存在，您可以在查询和分析时指定所要使用的版本。

优势

日志服务推出的SQL独享版，支持更强大的SQL分析。与普通版相比，具有如下优势：

- 高性能且无数据量限制，支持千亿级数据的高性能分析。
- 高并发数，单个Project支持的最大分析操作并发数从15个调整为100个。
- 独享资源，性能不受以其他用户突发流量影响。

应用场景

SQL独享版主要应用于以下场景：

- 分析性能要求高的场景，例如实时数据分析。
- 长周期的数据分析场景，例如月维度的数据分析。
- 大规模业务的数据分析场景，例如每天TB级别的数据分析。
- 通过日志服务实现多指标多维度（SQL并发数大于15个）的报表需求场景。

开启SQL独享版

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 单击  图标。

   开启SQL独享版后，您可以使用SQL独享版执行查询和分析操作。具体操作，请参见查询和分析日志。

   > **注意** 开启独享版是指在当前查询和分析操作中开启， 不影响其他查询和分析操作。

SDK示例

- 通过Java SDK使用SQL独享版
- 通过Python SDK使用SQL独享版
- 通过Node.js SDK使用SQL独享版
- 通过PHP SDK使用SQL独享版
- 通过C++ SDK使用SQL独享版

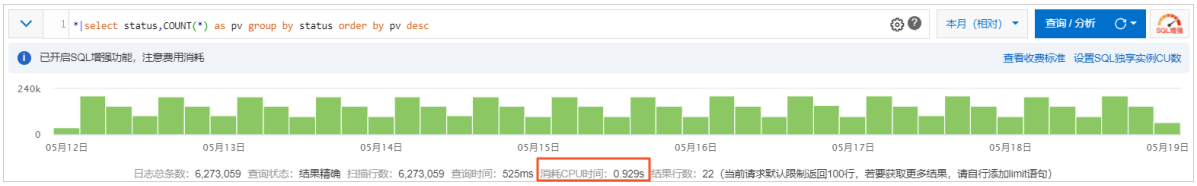
常见问题

- 如何通过API开启SQL独享版？

  您可以在GetLogs接口中，通过powerSql参数或query参数开启SQL独享版。更多信息，请参见GetLogs。

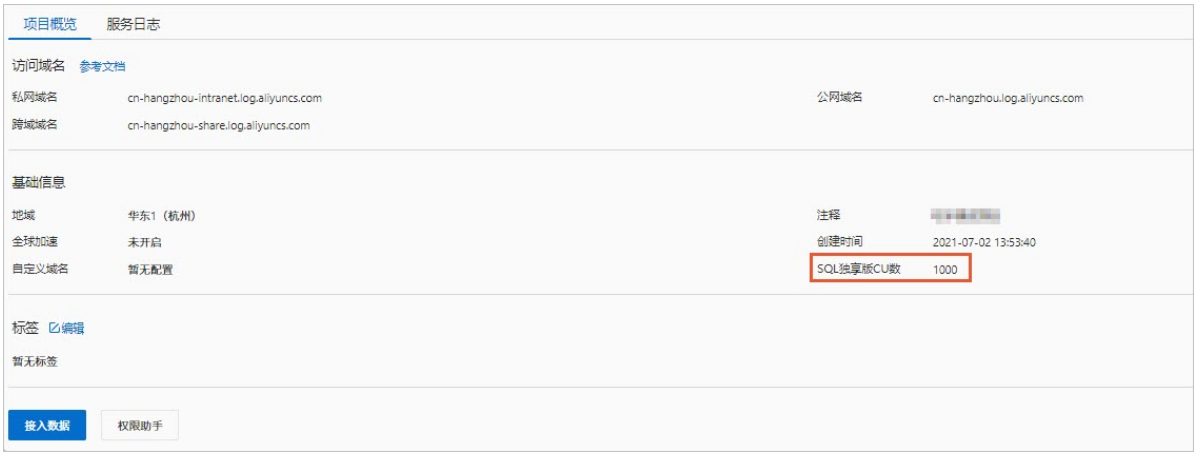
- 如何获取CPU时间？

  您可以在执行查询和分析操作后，获取CPU时间，如下图所示。



- SQL独享版的费用是否可控？

  日志服务通过SQL独享版的CU数来控制SQL独享版的费用。您可以在目标Project的**概览**页面中，配置**SQL独享版CU数**，如下图所示。



- 使用一次SQL独享版的费用是多少？

  在不同的数据量中执行不同的查询和分析语句，会产生不同的SQL独享版费用，案例如下表所示。

查询和分析语句	数据量（行）	平均每次的费用（元）
`* | select avg(double_0) from stress_s1_mil1`	40亿	0.030
`* | select avg(double_0), sum(double_0), max(double_0), min(double_0), count(double_0) from stress_s1_mil1`	40亿	0.044
`* | select avg(double_0), sum(double_1), max(double_2), min(double_3), count(double_4) from stress_s1_mil1`	40亿	0.092
`* | select key_0 , avg(double_0) as pv from stress_s1_mil1 group by key_0 order by pv desc limit 1000`	40亿	0.080
`* | select long_0, avg(double_0) as pv from stress_s1_mil1 group by long_0 order by pv desc limit 1000`	40亿	0.075
`* | select long_0, long_1, avg(double_0) as pv from stress_s1_mil1 group by long_0,long_1 order by pv desc limit 1000`	3亿	0.073
`* | select avg(double_0) from stress_s1_mil1 where key_0 ='key_987'`	40亿	0.0005

9.下载日志

日志服务支持将日志或查询分析结果下载到本地，本文介绍下载方式及操作步骤。

下载方式说明

日志服务提供控制台、Cloud Shell、日志服务CLI或SDK下载方式下载日志。

> 注意 Cloud Shell、日志服务CLI或SDK下载方式无数量限制，但可能由于网络等不确定因素，出现下载中断问题。

比较项	控制台直接下载	本地运行CLI下载	Cloud Shell下载	SDK下载
最大下载量	• 查询：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）和新加坡地域支持下载2000万条日志，其他地域支持下载100万条日志。 最大支持20 GB数据量。 • 分析：10万行数据。 最大支持2 GB数据量。	无数量限制	100万条	无数量限制
部署	无	手动部署	自动部署	手动部署
密钥	无	手动配置	自动配置	手动配置
局域网下载（不产生公网流量费用）	无	支持（需部署在对应地域的ECS上）	仅支持上海地域	支持（需部署在对应地域的ECS上）
NAS集成	无	手动配置	自动配置	手动配置

通过控制台直接下载

日志服务支持通过控制台直接将日志或查询分析结果下载到本地，两者的下载操作类似，本文以下载日志为例进行说明。如果您要下载查询分析结果，可在执行查询分析操作后，在**统计图表**页签中，单击**下载日志**。

注意

- 单次最多下载100万条日志。超出时，仅下载前100万条，如果需要下载全量日志，可缩小查询的时间范围，分多次下载。
- 单次最多下载10万行分析结果。超出时，仅下载前10万条，如果需要下载全量的分析结果，可缩小查询的时间范围，分多次下载。
- 单个阿里云账号最多支持3个并发下载操作（总下载次数无限制）。超出3个并发下载操作或多个RAM账号同时操作时，可能报错，此时您可等待其他操作完成后，再重试。
- 支持保存最近1天内的导出记录，超过1天的导出记录被自动清除。
- 在遇到网络错误或者查询不精确时，系统会自动重试下载任务。如果重试3次后，仍无法完成下载，则下载任务为失败状态。

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. （可选）输入查询语句，选择时间范围，然后单击**查询/分析**。

   更多信息，请参见查询和分析日志。
5. 在**原始日志**页签中，选择 > **下载日志**。
6. 在**日志下载**对话框中，完成如下配置，然后单击**确认**。

参数	说明
任务名	下载任务的名称。
日志数量	选择要下载的日志数量。
数据格式	支持CSV格式和JSON格式。 ○ 采用CSV格式时，文件中的列名将根据前100条日志的字段生成。如果后续日志存在新的字段，则所有新的字段将以JSON格式存放在CSV文件的最后一列（列名为空）。 ○ 采用JSON格式时，单条日志的内容会转换为JSON格式，然后以单行形式写入文件。
quote字符	选择Quote字符，用于包裹日志中的特殊字符，避免被转义。
是否允许下载不精确的结果	如果选择否，则当出现查询结果不精确时，会下载失败。
压缩方式	支持gzip、lz4、zstd等压缩方式，也支持不压缩。 当下载的日志数量比较多时，强烈建议采用压缩方式，可显著降低下载量，减少文件的下载时间。
排序规则	日志的排序规则。

完成上述配置后，单击**确认**，系统将弹出**日志导出历史**对话框，展示直接下载的任务列表。您也可以在**原始日志**页签中，选择 > **日志导出历史**，打开**日志导出历史**对话框。

等待任务状态为**任务成功**后，您可以单击**下载**，下载日志到本地。

通过Cloud Shell下载

您也可以通过Cloud Shell下载日志。更多信息，请参见使用Cloud Shell下载日志数据。

> **说明** 目前Cloud Shell位于上海地域，如果当前Logstore不在上海地域，下载日志会产生一定的公网流量费用。价格详情请参见产品定价。

通过命令行工具下载

当您需要下载更大数量的日志时，可通过命令行工具下载。 更多信息，请参见使用日志服务CLI。

> **说明**
>
> - 通过命令行工具下载日志时，需替换命令中的AK信息。请登录用户信息管理控制台获取阿里云账号AK。如果使用RAM用户进行下载，请登录RAM 控制台创建RAM用户并用RAM用户的AK信息。
> - 如果用于安装命令行工具的机器的所在地域与当前Project所在地域相同，建议单击**切换为内网endpoint**，下载速度更快且不会产生额外的外网带宽费用。

通过SDK下载

当您需要下载更大数量的日志时，可通过SDK下载。 更多信息，请参见SDK参考概述。

10.查询语法与功能

10.1. 查询语法

日志服务提供一套查询语法用于设置查询条件，帮助您更有效地查询日志。

查询方式

查询语句用来指定日志查询时的过滤规则，返回符合条件的日志。根据索引配置方式可分为全文查询和字段查询，根据查询精确程度可分为精确查询和模糊查询。

> 说明
> - 同时配置了全文索引和字段索引时，以字段索引的配置为准。
> - 只有在字段索引中将字段的数据类型设置为double、long后，才能通过数值范围查询对应的日志。如果字段的数据类型不被设置为double、long或者查询时数值范围的语法错误，那么日志服务会按照全文查询方式进行查询，这样查询到的结果可能与您期望的结果不同。例如字段owner_id为非double、long类型，则执行查询语句 `owner_id>100` 时，会返回同时包含owner_id、>（非分词符）、100这三个词的日志。
> - 如果将字段的类型从text类型改成double、long类型，则修改索引之前的日志只支持等号（=）查询。

- 全文查询和字段查询

查询方式	说明	示例
全文查询	配置全文索引后，日志服务根据您设置的分词符将整条日志拆分成多个词。您可以指定关键字（字段名、字段值）和查询规则进行查询。	`PUT and cn-shanghai` 表示查询同时包含关键字PUT和cn-shanghai的日志。
字段查询	配置字段索引后，您可以指定字段名称和字段值（Key:Value）进行查询。根据字段索引中设置的数据类型，您可以进行多种类型的基础查询和组合查询。更多信息，请参见数据类型。	`request_time>60 and request_method:Ge*` 表示查询request_time字段值大于60且request_method字段值以Ge开头的日志。

- 精确查询和模糊查询

查询方式	说明	示例
精确查询	使用完整的词进行查询。	○ `host:example.com` 表示查询host字段值为example.com的日志。 ○ `PUT` 表示查询包含关键字PUT的日志。

查询方式	说明	示例
模糊查询	在查询语句中指定一个64个字符以内的词，在词的中间或者末尾加上模糊查询关键字，即星号（*）或问号（?），日志服务会在所有日志中为您查询到符合条件的100个词，返回包含这100个词并满足查询条件的所有日志。指定的词越精确，查询结果越精确。 **说明** ○ 星号（*）或问号（?）不能用在词的开头。 ○ long数据类型和double数据类型不支持使用星号（*）或问号（?）进行模糊查询。您可以使用数值范围进行模糊查询，例如status in [200 299]。 模糊查询是一种采样查询，查询机制如下所示： ○ 当您开启字段索引，且指定某个字段进行查询时，日志服务从该字段的索引数据中随机采样，返回部分结果并不是全量扫描底层数据。 ○ 当您开启全文索引，且没有指定某个字段进行查询时，日志服务从全文索引数据中随机采样，返回部分结果并不是全量扫描底层数据。	○ `addr*` 表示在所有日志中查找以addr开头的100个词，并返回包含这些词的日志。 ○ `host:www.yl*` 表示在所有日志中查找host字段值以www.yl开头的100个词，并返回包含这些词的日志。 更多信息，请参见模糊查询。

运算符

查询语句支持如下运算符。

说明

- 除in运算符外，其他运算符不区分大小写。
- 日志服务保留以下运算符的使用权，如果您需要使用以下运算符作为查询关键字，请使用双引号（""）包裹：sort、asc、desc、group by、avg、sum、min、max和limit。
- 运算符的优先级由高到低排序如下所示：
  i. 冒号（:）
  ii. 双引号（""）
  iii. 圆括号()
  iv. and、not
  v. or

运算符	说明
and	and运算符。例如 `request_method:GET and status:200` 。 如果多个关键词之间没有语法关键词，默认为and关系，例如 `GET 200 cn-shanghai` 等同于 `GET and 200 and cn-shanghai` 。
or	or运算符。例如 `request_method:GET or status:200` 。
not	not运算符。例如 `request_method:GET not status:200` 、 `not status:200` 。
()	用于提高括号内查询条件的优先级。例如 `(request_method:GET or request_method:POST) and status:200` 。

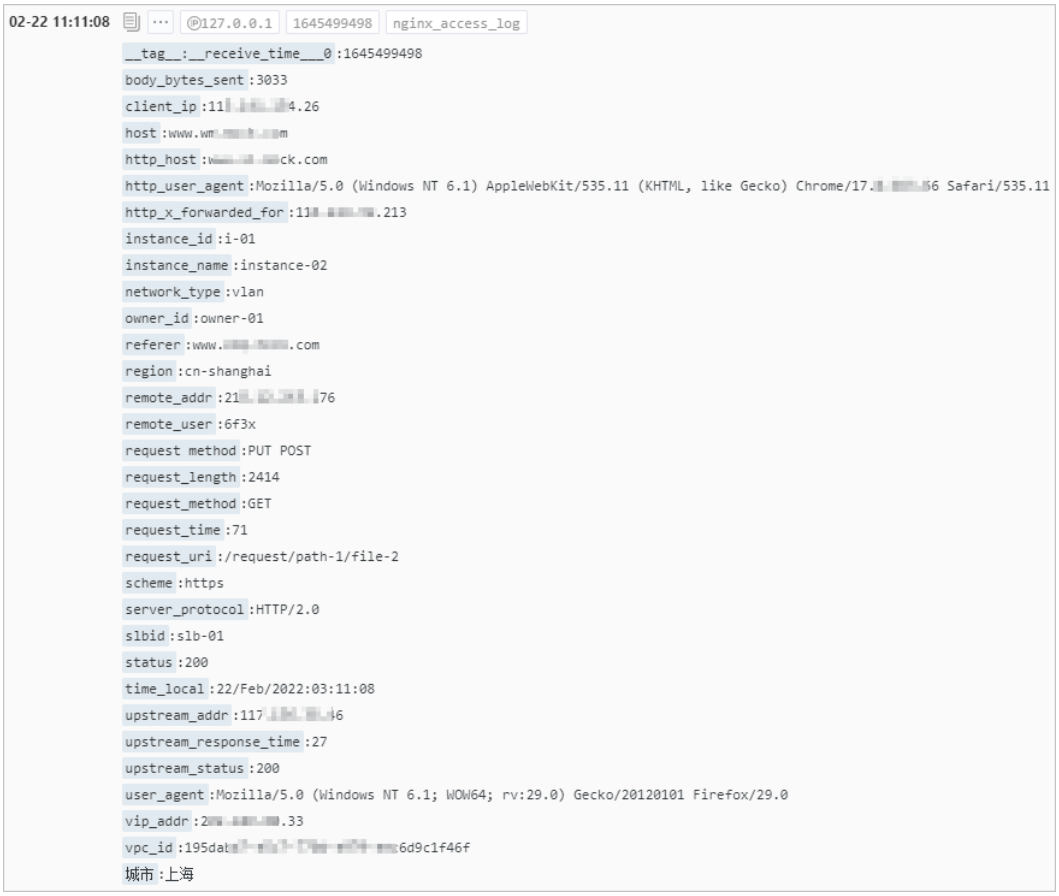
运算符	说明
:	用于字段查询（Key:Value），例如 `request_method:GET` 。 如果字段名称或者字段值内有空格、冒号（:）等保留字符，请使用双引号（""）包裹字段名称或者字段值，例如 `"file info":apsara` 。
""	使用双引号（""）包裹一个语法关键词，可以将该语法关键词转换成普通字符。例如 `"and"` 表示查询包含and的日志，此处的and不代表运算符。 在字段查询中双引号（""）内的所有词被当成一个整体。
\	转义符号，用于转义双引号（""），转义后的引号表示符号本身。例如日志内容为 `instance_id:nginx"01"` ，您可以使用 `instance_id:nginx\"01\"` 进行查询。
*	通配符查询，匹配零个、单个、多个字符。例如 `host:aliyund*c` 。 **说明** 日志服务会在所有日志中为您查询到符合条件的100个词，返回包含这100个词并满足查询条件的所有日志。
?	通配符查询，匹配单个字符。例如 `host:aliyund?c` 。
>	查询某字段值大于某数值的日志。例如 `request_time>100` 。
>=	查询某字段值大于或等于某数值的日志。例如 `request_time>=100` 。
<	查询某字段值小于某数值的日志。例如 `request_time<100` 。
<=	查询某字段值小于或等于某数值的日志。例如 `request_time<=100` 。
=	查询某字段值等于某数值的日志。针对double、long类型的字段，等号（=）和冒号（:）作用相同。例如 `request_time=100` 等同于 `request_time:100` 。
in	查询某字段值处于某数值范围内的日志，中括号表示闭区间，小括号表示开区间，两个数字之间使用空格分隔。例如 `request_time in [100 200]` 或 `request_time in (100 200]` 。 **说明** in只能为小写字母。
__source__	查询某个日志源的日志，支持通配符。例如 `__source__:192.0.2.*` 。 **注意** 日志服务中的__source__为保留字段，可缩写为source。如果您自定义的字段中存在source字段，则会与日志服务保留字段source冲突，此时您需要使用Source、SOURCE等词查询自定义的字段。
__tag__	通过元数据信息查询日志。例如 `__tag__:__receive_time__:1609837139` 。
__topic__	查询某日志主题下的日志。例如 `__topic__:nginx_access_log` 。

查询语句示例

同一条查询语句，针对不同的日志内容和索引配置时，会有不同的查询结果。本文基于如下日志样例和索引介绍查询语句示例。

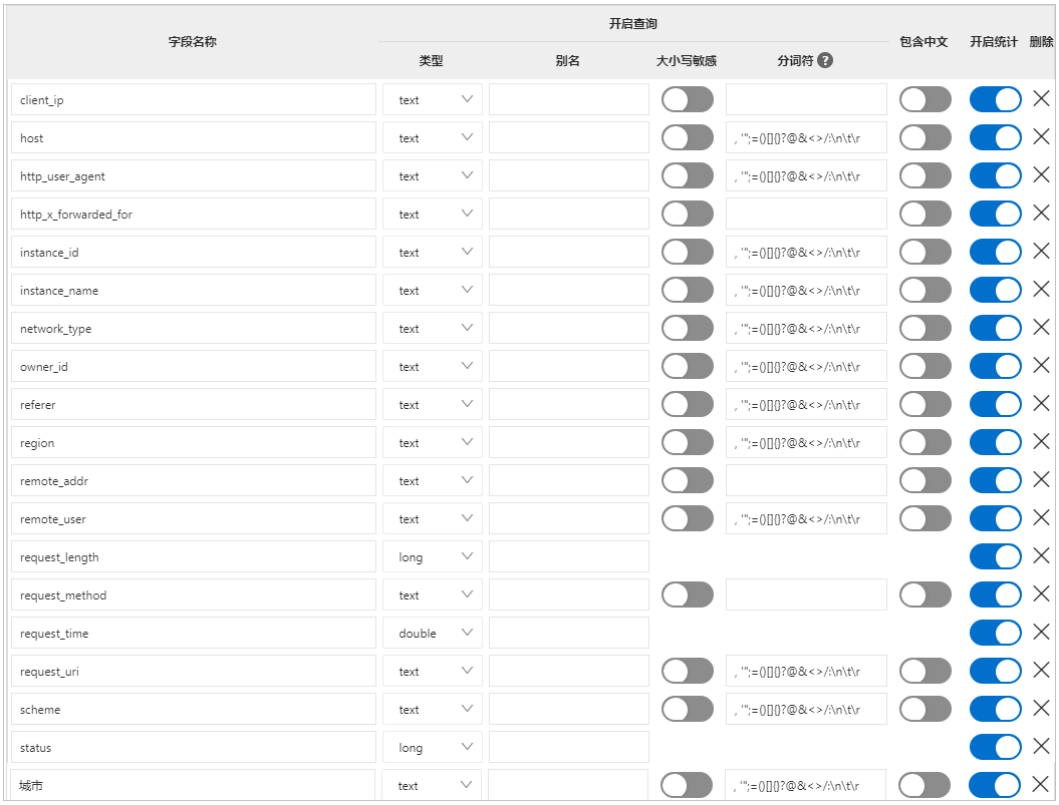
日志样例

本文以Nginx访问日志为例，介绍常见的查询语句。



索引配置

在查询日志前，请确保已配置索引。更多信息，请参见配置索引。



字段名称	开启查询				包含中文	开启统计	删除
	类型	别名	大小写敏感	分词符			
client_ip	text						
host	text			, '";=()[]{}?@&<>/:\n\t\r			
http_user_agent	text			, '";=()[]{}?@&<>/:\n\t\r			
http_x_forwarded_for	text						
instance_id	text			, '";=()[]{}?@&<>/:\n\t\r			
instance_name	text			, '";=()[]{}?@&<>/:\n\t\r			
network_type	text			, '";=()[]{}?@&<>/:\n\t\r			
owner_id	text			, '";=()[]{}?@&<>/:\n\t\r			
referer	text			, '";=()[]{}?@&<>/:\n\t\r			
region	text			, '";=()[]{}?@&<>/:\n\t\r			
remote_addr	text						
remote_user	text			, '";=()[]{}?@&<>/:\n\t\r			
request_length	long						
request_method	text						
request_time	double						
request_uri	text			, '";=()[]{}?@&<>/:\n\t\r			
scheme	text			, '";=()[]{}?@&<>/:\n\t\r			
status	long						
城市	text			, '";=()[]{}?@&<>/:\n\t\r			

普通查询示例

查询需求	查询语句
查询GET请求成功（状态码为200~299）的日志。	`request_method:GET and status in [200 299]`
查询来自非杭州地域的GET请求的日志。	`request_method:GET not region:cn-hangzhou`
查询GET请求或POST请求的日志。	`request_method:GET or request_method:POST`
查询非GET请求的日志。	`not request_method:GET`
查询GET请求或POST请求成功的日志。	`(request_method:GET or request_method:POST) and status in [200 299]`
查询GET请求或POST请求失败的日志。	`(request_method:GET or request_method:POST) not status in [200 299]`

查询需求	查询语句
查询GET请求成功（状态码为200~299）且请求时间小于60秒的日志。	`request_method:GET and status in [200 299] not request_time>=60`
查询请求时间为60秒的日志。	`request_time:60` `request_time=60`
查询请求时间大于等于60秒，并且小于200秒的日志。	`request_time>=60 and request_time<200` `request_time in [60 200)`
查询request_time字段值为空或非法数字的日志。	`request_time:* not request_time > -10000000000`
查询包含request_time字段且字段值为数字的日志。	`status > -1000000000`
查询包含and的日志。	`"and"` **说明** 此处的and为普通字符串，不代表运算符。
查询request method字段值中包含PUT的日志。	`"request method":PUT` **说明** 字段名request method中存在空格，在查询时需使用双引号（""）包裹。
查询日志主题为HTTPS或HTTP的日志。	`__topic__:HTTPS or __topic__:HTTP`

查询需求	查询语句
查询采集于192.0.2.1主机的日志。	`__tag__:__client_ip__:192.0.2.1` **说明** 此处的 `__tag__:__client_ip__` 为日志服务保留字段，表示日志所在主机的IP地址。更多信息，请参见保留字段。 通过数据加工或者Logtail插件处理的日志，其tag中的key会被转换成普通key，即查询时需使用双引号（""）包裹字段名，例如 `"__tag__:__client_ip__":192.0.2.1` 。
查询remote_user字段值不为空的日志。	`not remote_user:""`
查询remote_user字段值为空的日志。	`remote_user:""`
查询remote_user字段值不为null的日志。	`not remote_user:"null"`
查询不存在remote_user字段的日志。	`not remote_user:*`
查询存在remote_user字段的日志。	`remote_user:*`
查询城市字段值不为上海的日志。	`not 城市:上海`

进阶查询示例

- 模糊查询

查询需求	查询语句
查询包含以cn开头的词的日志。	`cn*`
查询region字段值是以cn开头的日志。	`region:cn*`

查询需求	查询语句
查询region字段值包含cn*的日志。	`region:"cn*"` **说明** 此处的 `cn*` 为一个独立词。例如： ○ 如果日志内容为 `region:cn*,en` ，分词符为半角逗号（,），则该日志内容被拆分为 `region` 、 `cn*` 和 `en` ，你可以通过上述语句查询到该日志。 ○ 如果日志内容为 `region:cn*hangzhou` ，则 `cn*hangzhou` 为一个整体，您执行上述语句无法查询到该日志。
查询包含以mozi开头，以la结尾，中间还有一个字符的词的日志。	`mozi?la`
查询包含以mo开头，以la结尾，中间包含零个、单个或多个字符的词的日志。	`mo*la`
查询包含以moz开头的词和以sa开头的词的日志。	`moz* and sa*`
查询region字段值以hai结尾的所有日志。	目前使用查询语句无法查询到对应的日志，您可以使用SQL分析中的LIKE语法进行查询。更多信息，请参见通过SQL的like语法进行精确的模糊查询。 `* | select * from log where region like '%hai'`

- 基于分词符的查询

  日志服务会根据您所指定的分词符（ `, '";=()[]{}?@&<>/:\n\t\r` ），将日志内容拆分成多个词。如果未设置分词符，则字段值将被当成一个整体，您只能通过完整字符串或模糊查询查找对应的日志。如何设置分词符，请参见配置索引。

  说明 当查询关键字中包含分词符时，您可以使用短语查询或者Like语法。例如：
  - 短语查询： `#"redo_index/1"` 。更多信息，请参见短语查询。
  - Like语法： `* | select * from log where key like 'redo_index/1'` 。

  例如http_user_agent字段值为 `Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.2 (KHTML, like Gecko) Chrome/192.0.2.0 Safari/537.2` 。

  - 未设置分词符时，该值为一个整体，则您使用 `http_user_agent:Chrome` 查询语句进行查询时，无法查询到日志。
  - 设置分词符为 `, '";=()[]{}?@&<>/:\n\t\r` 后，该值为拆分为 `Mozilla` 、 `5.0` 、 `Windows` 、 `NT` 、 `6.1` 、 `AppleWebKit` 、 `537.2` 、 `KHTML` 、 `like` 、 `Gecko` 、 `Chrome` 、 `192.0.2.0` 、 `Safari` 、 `537.2` 。您可以使用 `http_user_agent:Chrome` 等查询语句进行查询。

查询需求	查询语句
查询http_user_agent字段值中包含Chrome的日志。	`http_user_agent:Chrome`

查询需求	查询语句
查询http_user_agent字段值中包含Linux和Chrome的日志。	`http_user_agent:"Linux Chrome"` `http_user_agent:Linux and http_user_agent:Chrome`
查询http_user_agent字段值中包含Firefox或Chrome的日志。	`http_user_agent:Firefox or http_user_agent:Chrome`
查询request_uri字段值包含/request/path-2的日志。	`request_uri:/request/path-2`
查询request_uri字段值以/request开头，但不包含/file-0的日志。	`request_uri:/request* not request_uri:/file-0`

- 查询JSON日志（字段值为JSON对象、JSON数组）

  当字段值为JSON格式时，您可以为该字段配置JSON类型的索引或者使用JSON函数进行查询与分析。更多信息，请参见查询和分析JSON日志的常见问题。

10.2. 短语查询

本文介绍短语查询的语法、使用限制和示例。

概述

日志服务查询采用的是分词法，例如查询语句为 `abc def` ，将匹配所有包含 `abc` 或 `def` 的日志，不区分先后顺序，无法精准匹配目标短语。现在日志服务推出短语查询，用于精准匹配一段短语。

日志服务接收到短语查询请求后，执行流程主要分为如下两步：

1. 先执行对应的非短语查询语句进行日志查询。例如执行 `#"/92//docvalue"` 语句，实际先执行 `"/92//docvalue"` 语句。

   > **说明** 为避免查询量太大，目前执行短语查询时，限制步骤1最多返回10,000条结果。

2. 在上述查询结果中再挑选符合短语查询条件的日志，并返回最终的查询结果。

语法

- 字段查询

  ```
  key:#"abc def"
  ```

- 全文查询

  ```
  #"abc def"
  ```

使用限制

- 短语查询的结果只支持向前、向后的连续翻页，不支持随机跳转。
- 执行短语查询后，日志分布直方图展示的是非短语查询的结果。
- 短语查询不支持搭配模糊查询。
- 短语查询语句中必须添加半角双引号（""）。

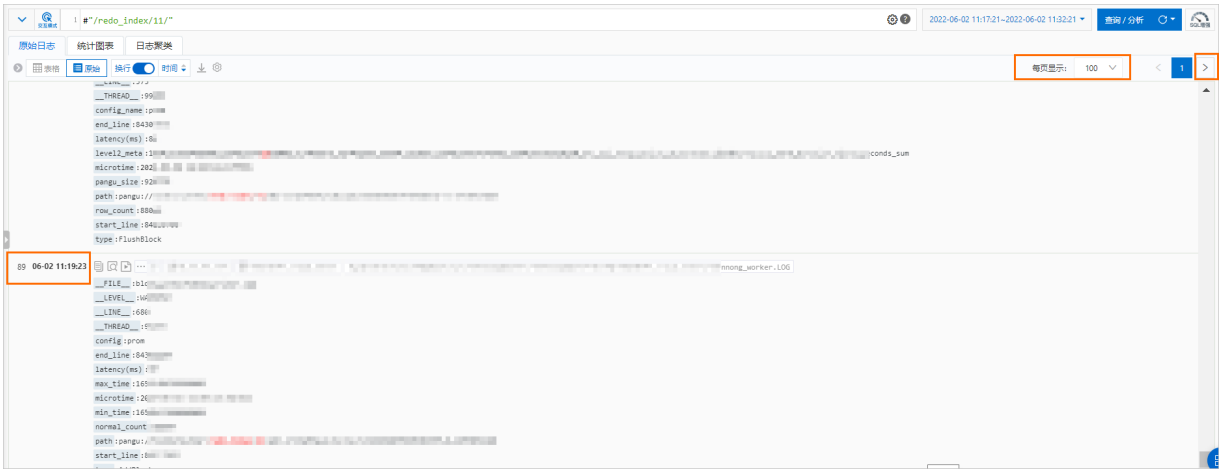
- 短语查询语句中不支持搭配not语句，即不支持 `not #"abc def"` 。
- 短语查询语句中不支持搭配分析语句，即不支持 `#"abc" | select ***` 。因此使用短语查询时，也不支持快速分析功能。

翻页说明

当您执行一次翻页操作时，日志服务会对应执行一次短语查询操作，用于保证查询结果的连续性。

短语查询每次最多查询10,000条日志，在翻页过程中，可能出现某页中显示的日志数量少于**每页显示**对应的数量，但仍支持向后翻页。即表示当前查询的10,000条日志中，满足短语查询条件的日志数量少于**每页显示**对应的数量。

例如日志总数为20,000条，每页支持显示100条，当您执行一次短语查询后，只返回89条且向后翻页功能可用，此时说明前10,000条日志中只有89条日志满足短语查询条件。您可以执行翻页操作，日志服务会自动在后10,000条日志中，执行第二次短语查询，并返回符合条件的日志。

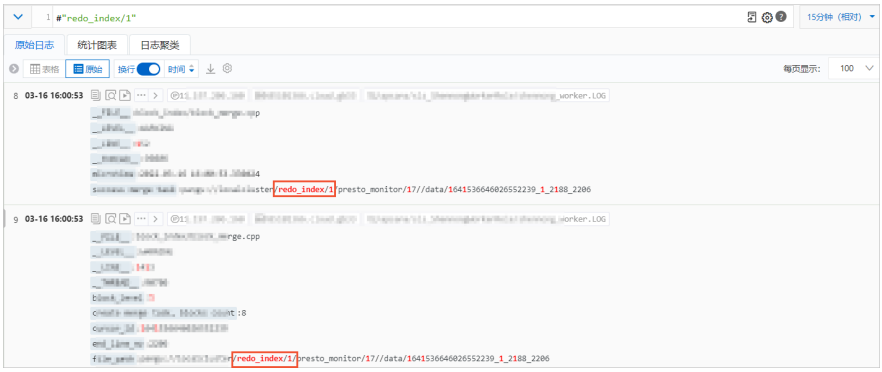


示例

例如您要查询包含 `redo_index/1` 的日志。

- 使用非短语查询语句 `"redo_index/1"` ，日志服务将根据全文索引匹配部分关键词。

- 使用短语查询语句 `#"redo_index/1"` ，日志服务将匹配完整的短语 `redo_index/1` 。



10.3. LiveTail

本文介绍日志服务的LiveTail功能及相关操作。

前提条件

已通过Logtail采集到日志。具体操作，请参见通过Logtail采集日志。

背景信息

在线上运维的场景中，往往需要对日志队列中的数据进行实时监控，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。在传统的运维方式中，如果需要对日志文件进行实时监控，需要在服务器上对日志文件执行tail -f命令，如果实时监控的日志信息不够直观，还需加上grep命令或者grep -v命令进行关键词过滤。日志服务在控制台提供了日志数据实时监控的交互功能LiveTail，针对线上日志进行实时监控分析，减轻运维压力。

功能优势

- 监控日志的实时信息，按关键词过滤日志数据。
- 结合采集配置，对采集的日志进行索引区分。
- 日志字段做分词处理，以便查询包含分词的上下文日志。
- 根据单条日志信息追踪到对应日志文件进行实时监控，无需连接线上机器。

操作步骤

1.
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在**原始日志 > 原始**页签下，单击目标日志中的图标。

> 注意 LiveTail功能仅支持Logtail采集到的日志。

5. 在LiveTail区域，查看结果。

开启LiveTail后，将Logtail采集到的日志数据实时显示在监控列表中。最新的日志数据始终在页面底部，且滚动条默认在最下方，即显示最新数据。最多显示1000条数据，满1000条后页面自动刷新并重新填充日志数据。

更多操作

操作	说明
高亮显示	您可以在**高亮显示**文本框中设置需要高亮显示的一个或多个字符串。设置完成后，监控列表高亮显示设置的字符串。
过滤字符串	您可以在**过滤条件**文本框中设置需要包含的一个或多个字符串。设置完成后，监控列表只显示包含设置的字符串的日志。
过滤字段	您可以在**字段过滤**下拉列表中设置需要过滤的一个或多个字段。设置完成后，监控列表不显示设置的过滤字段。
停止	您可以单击**停止**以分析异常日志数据。LiveTail计时结束，不再实时更新日志数据，您可以对监控过程中发现的问题进行进一步分析排查。

10.4. 日志聚类

本文介绍日志聚类功能及其操作，包括开启日志聚类、查看聚类结果和原始日志、对比不同时间段的聚类日志数量等。

背景信息

日志服务提供日志聚类功能，支持在采集日志时，将相似度高的日志聚合，提取共同的日志模式（Pattern），快速掌握日志全貌。支持多种格式的文本日志聚合，可应用于DevOps中的问题定位、异常检测、版本回归等运维动作，或应用于安全场景下的入侵检测等。您还可以将聚类结果以分析图表的形式保存在仪表盘中，实时查看聚类数据。

功能优势

- 支持任意格式日志，例如Log4j、JSON、单行等。
- 亿级数据，秒级输出结果。
- 日志数据可以按任意模式聚类。
- 按pattern聚类的数据可以根据pattern的签名反查原始数据。
- 比较不同时间段的pattern。
- 动态调整聚类精度。

操作视频

索引流量

开启日志聚类功能后，索引总量会增加原始日志大小的10%。例如原始数据为100 GB/天，开启该功能后，索引总量增加10 GB。

原始日志大小	索引比例	日志聚类功能产生的索引量	索引总量
100 GB	20%（20 GB）	100 * 10%	30 GB
100 GB	40%（40 GB）	100 * 10%	50 GB

原始日志大小	索引比例	日志聚类功能产生的索引量	索引总量
100 GB	100%（100 GB）	100 * 10%	110 GB

开启日志聚类功能

1.
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 开启日志聚类功能。
   i. 单击**查询分析属性 > 属性**。
   如果您还未开启索引，请单击**开启索引**。
   ii. 在**查询分析**面板中，打开**日志聚类**开关。
   iii. （可选）设置聚类字段的白名单和黑名单。

   > **说明** 不支持同时设置黑白名单。

聚类字段过滤	说明
白名单	设置了白名单后，日志服务将根据白名单中的字段进行日志聚类。
黑名单	设置了黑名单后，日志服务不会对黑名单中的字段进行日志聚类。
未设置黑白名单	未设置黑白名单时，日志服务将根据聚类规则对所有的字段进行日志聚类。

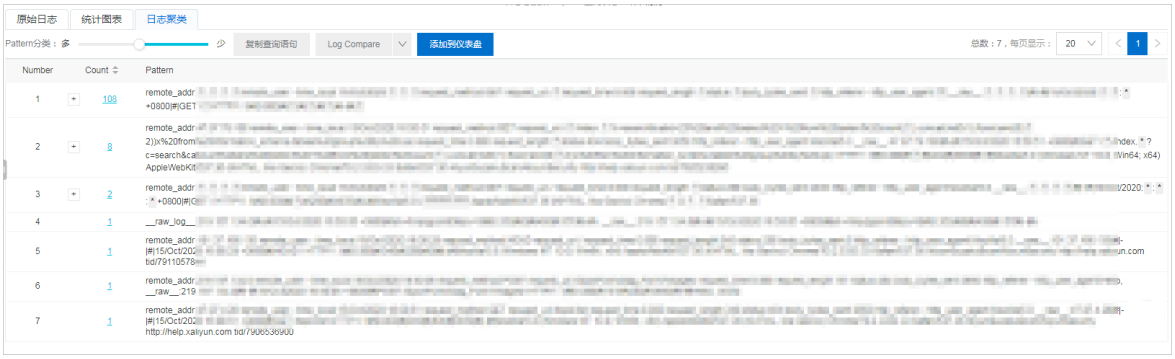
   iv. 单击**确定**。

查看聚类结果和原始日志

1. 在查询分析页面，输入查询语句，设置查询时间范围，然后单击**查询/分析**。

   > **说明** 此处仅支持输入查询语句来过滤日志，但不支持分析语句，即不能对分析结果进行日志聚类。

2. 单击**日志聚类**页签，查看聚类结果。
   您还可以单击**添加到仪表盘**，将聚类结果保存到仪表盘。



参数	说明
Number	聚类序号。

参数	说明
Count	当前指定的查询时间段内，某Pattern对应的日志条数。
Pattern	某类日志的具体模式，每个聚类会有一个或多个子Pattern。

- 鼠标指向**Count**列的数字，在悬浮框展示当前聚类的子Pattern及每个子Pattern的占比。单击数字前的加号（+），展开子Pattern列表。
- 单击**Count**列的数字，跳转到**原始日志**页签，查看对应pattern的原始日志。

调整聚类精度

在**日志聚类**页签中，拖拽**Pattern分类**中的滑动条，调整聚类的精度。

- 聚类偏向于**多**，表示聚类结果分类细，Pattern细节被保留的多。
- 聚类偏向于**少**，表示聚类结果分类粗，Pattern细节被隐藏的多。

对比不同时间段的聚类日志数量

1. 在**日志聚类**页签中，单击**Log Compare**。
2. 设置对比时间，单击**确定**。

   例如：在执行查询操作时，时间范围选择为15分钟。在**Log Compare**中，选择**1天**，则自动显示开始时间和结束时间，时间范围为1天前的15分钟。

参数	说明
Number	聚类编号。
Pre_Count	在**Log Compare**中设置的时间段内，该Pattern对应的日志数量。
Count	当前指定的查询时间段内，某模式对应的日志条数。
Diff	某模式在两个时段的日志数量差值及升降百分比。
Pattern	某类日志的具体模式。

SQL示例

日志服务还支持通过执行查询分析语句获取日志聚类结果。

- 获取日志聚类结果

- 查询分析语句

```
* | select a.pattern, a.count,a.signature, a.origin_signatures from (select log_reduce(3) as a f
rom log) limit 1000
```

> **说明** 查看聚类结果时，您可以单击**复制查询语句**获取日志聚类结果所对应的查询分析语句。

- 修改参数

  修改查询分析语句中的log_reduce(precision)，precision代表聚类精度，取值范围1~16，数字越小，聚类精度越高，生成的模式格式越多，默认为3。

- 返回字段

  在**统计图表**页签中返回日志聚类详细信息。

参数	说明
pattern	某类日志的具体模式。
count	当前指定的查询时间段内，某模式对应的日志条数。
signature	某模式的签名。
origin_signatures	某模式的二级签名，可以通过二级签名，反查原始数据。

- 对比不同时间段日志聚类结果
  - 查询分析语句

```
* | select v.pattern, v.signature, v.count, v.count_compare, v.diff from (select compare_log_red
uce(3, 86400) as v from log) order by v.diff desc limit 1000
```

> **说明** Log Compare对比不同时间段日志聚类结果后，您可以单击**复制查询语句**获取对应的查询分析语句。

  - 修改参数

    修改查询分析语句中的compare_log_reduce(precision, compare_interval)。

    - precision代表聚类精度，取值范围1~16，数字越小，聚类精度越高，生成的模式格式越多，默认为3。
    - compare_interval表示对比N秒之前某一时间段的日志，正整数。

  - 返回字段

参数	说明
pattern	某类日志的具体模式。
count_compare	在前一时间段内，某模式对应的日志数量。
count	当前指定的查询时间段内，某模式对应的日志条数。
diff	count和count_compare的差值。
signature	某模式的签名。

关闭日志聚类功能

如果您不再需要使用日志聚类功能，可关闭该功能。

1. 在查询分析页面，单击**查询分析属性 > 属性**。
2. 关闭**日志聚类**开关。
3. 单击**确定**。

10.5. 上下文查询

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。

前提条件

- 已通过Logtail采集到日志。具体操作，请参见通过Logtail采集日志。
- 已配置索引。具体操作，请参见配置索引。

背景信息

日志上下文查询是指定日志来源（机器+文件）和其中一条日志，将该日志在原始文件中的前若干条（上文）或后若干条日志（下文）也查找出来。通过查看指定日志的上下文信息，您可以在业务故障排查中快速查找相关故障信息，方便定位问题。

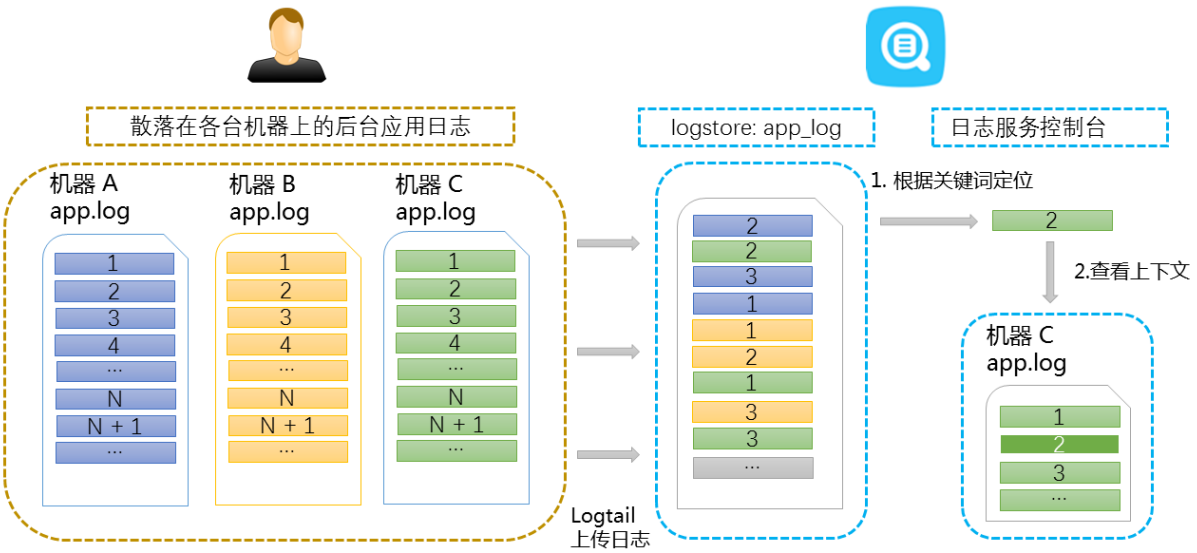
应用场景

例如，O2O外卖网站在服务器上的程序日志里会记录一次订单成交的轨迹：用户登录>浏览商品>选择物品>加入购物车>下单>订单支付>支付扣款>生成订单。

如果客户下单失败，运维人员需要快速定位问题原因。传统的上下文查询中，需要管理员等相关人员添加机器登录权限，然后运维人员依次登录应用所部署的每一台机器，以订单ID为关键词搜索应用程序日志文件，帮助判断下单失败原因。

在日志服务中，可以按照以下步骤排查。

1. 在服务器上安装日志采集客户端Logtail，并在日志服务控制台上添加机器组、Logtail采集配置，然后Logtail开始上传增量日志。
2. 在日志服务控制台日志查询页面，指定时间段根据订单ID找到订单失败日志。
3. 以查到的日志为基准，向上翻页直到发现与之相关的其它日志信息（例如：信用卡扣款失败）。



功能优势

- 不侵入应用程序，无需改动日志文件格式。
- 在日志服务控制台上查看任意服务器、文件的指定日志的上下文信息。无需登录服务器查看日志的上下文。
- 结合事件发生的时间线索，在日志服务控制台指定时间段快速定位可疑日志后再进行上下文查询，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，在日志服务控制台上随时可以查看历史数据。

操作步骤

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 输入查询分析语句，选择时间范围并单击**查询/分析**。
5. 在**原始日志 > 原始**页签下，找到目标日志，单击图标。

> 注意 只有通过Logtail采集到的日志才支持上下文功能。

6. 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**更早**，进行向上翻页浏览。
   - 单击**更新**，进行向下翻页浏览。
   - 在**高亮显示**文本框中设置需要高亮显示的字符串，可实现字符串标红显示。
   - 在**过滤条件**文本框中设置过滤字符串，可实现日志列表中只显示包含过滤字符串的日志。

10.6. 快速查询

当您需要频繁查看某一查询和分析语句的结果时，可以将该查询和分析语句另存为快速查询。快速查询是日志服务提供的用于保存查询和分析操作的功能。您可通过保存的历史操作，快速执行查询和分析操作。

前提条件

已配置索引。具体操作，请参见配置索引。

创建快速查询

1.
2.

3.
4. 输入查询和分析语句，设置时间范围并单击**查询/分析**。

   查询和分析语句由查询语句和分析语句构成，格式为查询语句|分析语句，查询分析语句语法请参见查询语法、SQL分析语法。
5. 单击页面右上角的**另存为快速查询**。
6. 在**快速查询详情**面板中，配置如下参数。

参数	说明
快速查询名称	设置快速查询的名称。
变量配置	如果您需要操作下钻分析，请选中**查询语句**中的内容，单击**生成变量**，可生成占位符变量。 ○ **变量名**：占位符变量命名。 ○ **默认值**：您在**查询语句**中选中的内容。 ○ **匹配模式**：通过下钻操作替换默认值时的匹配模式，包括全局匹配和精确匹配。 ■ 全局匹配：将查询和分析语句中跟您划选的关键字相同的关键词都替换为变量。 ■ 精确匹配：将查询和分析语句中您所划选的关键字替换为变量。 如果其他图表的**事件行为**为**打开快速查询**并指定该快速查询，且图表配置的**变量**和快速查询的**变量名**相同。单击其他图表时会执行跳转，占位符变量的**默认值**替换为触发下钻事件的图表值，并以替换变量后的查询语句执行查询。更多信息，请参见交互事件。 **说明** 设置下钻分析时，如果将**事件行为**设置为**打开快速查询**，则必须提前创建快速查询，并完成变量配置。

7. 单击**确定**。

   创建快速查询后，您在Logstore查询和分析页面，单击搜索框前面的图标，然后单击目标快速查询名称即可使用。

修改快速查询

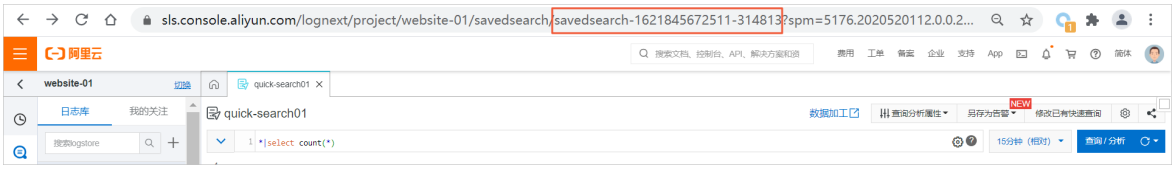
1. 在左侧导航栏中，选择**资源 > 快速查询**。
2. 在快速查询列表中，单击目标快速查询。
3. 输入新的查询和分析语句，单击**查询/分析**。

   查询和分析语句由查询语句和分析语句构成，格式为查询语句|分析语句，查询分析语句语法请参见查询语法、SQL分析语法。
4. 单击页面右上角的**修改已有快速查询**。
5. 在**快速查询详情**面板中，修改快速查询相关信息，单击**确定**。

获取快速查询ID

创建快速查询后，您可以使用快速查询ID将快速查询页面嵌入到自建Web页面。具体操作，请参见配置控制台内嵌参数。

1. 在左侧导航栏中，选择**资源 > 快速查询**。
2. 在快速查询列表中，单击目标快速查询。
3. 在URL中获取快速查询ID。



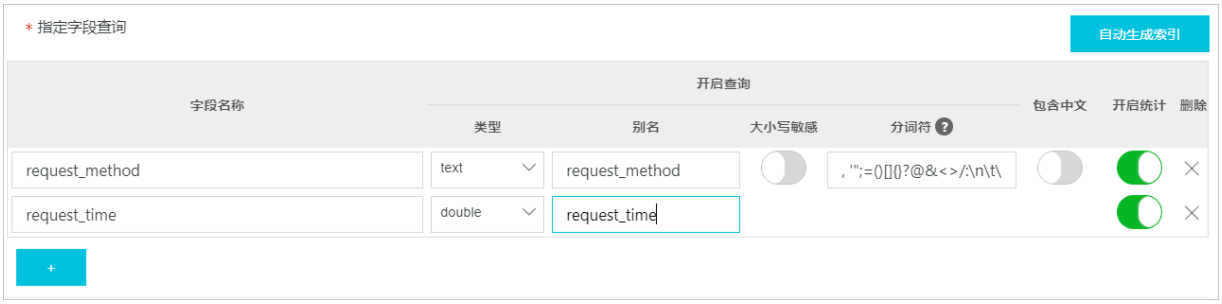
10.7. 快速分析

日志服务快速分析功能提供了一键式交互查询体验，帮助您快速分析某一字段在指定时间内的分布情况。

前提条件

已配置指定字段的索引，并开启统计功能。具体操作，请参见配置索引。

例如访问日志中存在request_method和request_time字段，可参考如下配置。



功能特点

- 支持前100,000条日志统计分析。

  > **说明** 在选中的查询时间段内，采样100,000条数据。如果您使用快速查询生成SQL语句，需删除Limit 100000字段，才能分析全量日志。

- 支持text类型字段分组统计，取占比最多的前十个分组。
- 支持text类型字段快速生成**approx_distinct**（唯一数）查询语句。
- 支持long或者double类型字段近似分布直方统计。

  直方图分布是指把采样数据划分到各个组中，并给出每个组的均值。
- 支持long或者double类型字段快速查找最大项、最小项、平均值或总和。
- 支持将快速分析查询生成查询语句。

操作步骤

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在**原始日志**页签的**快速分析**区域，单击目标字段。

- 提供text类型字段分组统计和long、double类型字段的近似分布直方统计，详情请参见示例（text类型）或示例（long、double类型）。
- 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，并提供对应的查询分析语句。

- 检查字段的唯一数

  单击目标字段下的**唯一数**，即可进行检查操作，即检查 `${keyName}` 的唯一项个数。

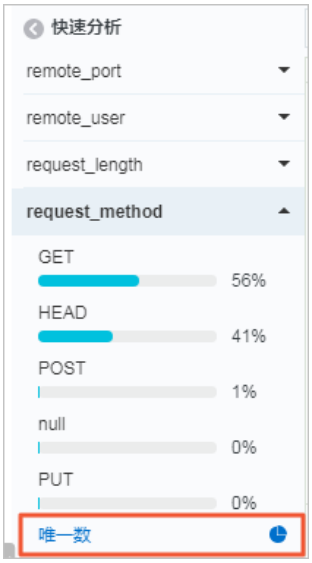
- 显示Key或Key的别名

  单击，选择显示Key或Key的别名，该别名可在创建索引时配置。例如host_name的别名为host，如果你选择显示别名，则在快速分析列表中显示host。

  > **说明** 当某字段没有别名时，您选择显示别名，在快速分析列表中仍显示字段名（Key）。

示例（text类型）

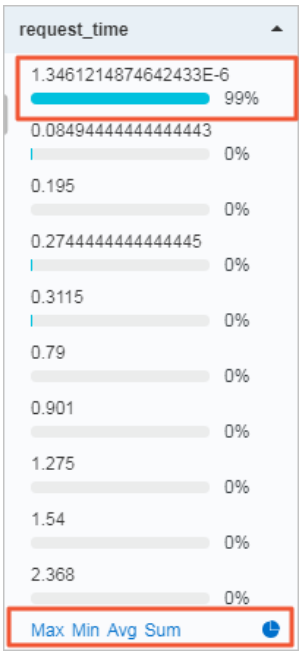
快速分析功能支持text类型字段分组统计，快速对目标字段对应的前100,000条日志进行分组，并返回前十项的占比。例如request_method字段按照分组统计可以得到如下结果，GET请求比重大。



示例（long、double类型）

- 近似分布直方统计

由于long、double类型的字段存在多种类型值，计算分组统计意义不大。所以日志服务支持将数据分为10个分组进行近似分布直方统计。例如request_time字段按照近似分布直方统计得到如下结果，绝大多数请求时间分布约在1.346毫秒。



- 快速统计最大项、最小项、平均值和总和

  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。

10.8. 重建索引

当您修改了索引规则或需要对历史数据配置索引时，可以使用重建索引功能。日志服务支持在选定的时间段中按照最新的索引规则为Logstore重建索引。本文介绍如何在日志服务控制台上重建索引及相关信息。

前提条件

已开启索引。具体操作，请参见配置索引。

使用限制

- 重建索引支持的时间段：30天~15分钟之前。
- 最多支持创建10个重建任务。
- 同一时间内只支持运行1个重建任务。

费用

- 重建索引会另外产生索引费用（只收取一次）和存储费用（每小时计价一次），价格和正常索引一样。更多信息，请参见计费项。
- 删除索引任务后，新生成的索引数据也被删除，不再收取重建索引而产生的存储费用。

操作步骤

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 单击**查询分析属性 > 重建**。

5. 在**重建索引**面板，单击**新建任务**。
6. 根据需求配置**任务名**、**开始时间**和**结束时间**，单击**确定**。

   开始时间和**结束时间**为日志服务接收到数据的服务端时间，支持30天至15分钟前的时间点。

   创建完成后，可查看重建索引的进度，当进度达到100%时，重建完成。

相关操作

创建重建索引任务后，您还可以进行如下操作：

- 停止重建索引

  在**重建索引**面板，单击**停止**，停止当前任务。

  > 注意 停止后，任务无法重启，只允许删除。请谨慎操作。

- 删除重建索引

  在**重建索引**面板，单击**删除**，删除当前任务。

  > 注意 删除任务的同时重建的索引数据也被删除。请谨慎操作。

10.9. 事件配置

事件配置为原始日志提供可视化、易操作的日志钻取功能，方便您获取更详尽的日志信息，包括预设事件配置和高级事件配置。本文介绍如何在日志服务控制台为原始日志配置事件。

前提条件

- 已开启并配置索引。更多信息，请参见配置索引。
- 如果配置高级事件为打开日志库，则需提前创建目标日志库。更多信息，请参见创建Logstore。
- 如果配置高级事件为打开快速查询，则需提前创建目标快速查询。更多信息，请参见快速查询。

  如果要配置变量，则需要在跳转到的目标快速查询中配置查询分析语句的占位符变量。更多信息，请参见设置占位符变量。
- 如果配置高级事件为打开仪表盘，则需提前创建目标仪表盘。更多信息，请参见创建仪表盘。

  如果要配置变量，则在跳转到的目标仪表盘中需有对应的图表已占位符变量。更多信息，请参见设置占位符变量。
- 如果配置高级事件为自定义HTTP链接，则需提前准备好HTTP链接。

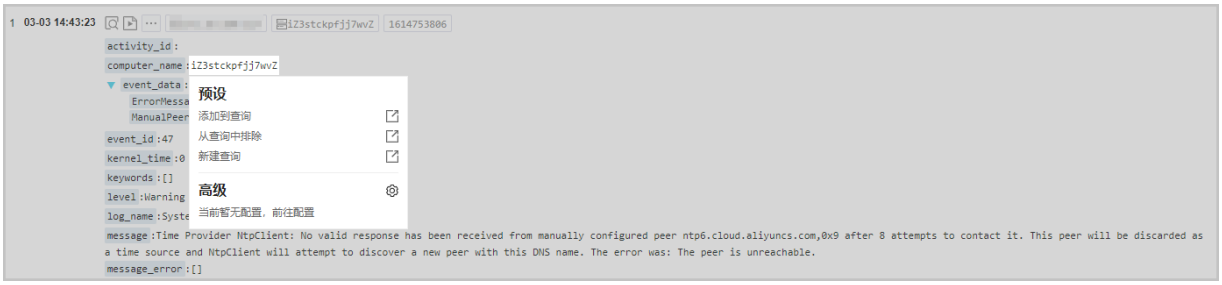
背景信息

钻取是数据分析中不可缺少的功能之一，通过改变数据维度的层次、变换分析的粒度从而获取数据中更详尽的信息。它包括向上钻取（roll up）和向下钻取（drill down）。向下钻取是在分析时加深维度，更深入的查看数据，挖掘更大的数据价值，及时做出更加正确的决策。日志服务通过预设和高级事件，为您提供原始日志的向下钻取功能。

预设事件配置

通过预设事件，您可以快速使用and和not拼接查询语句，也可以新建查询语句。

在**表格**或**原始**页签中，单击任意日志字段的值，会弹出**预设**窗口，提供的操作如下表所述。



例如，您查询和分析框中的语句为 `* | SELECT status as dim, count(1) as c group by dim` 。当您单击原始日志字段host中的203.0.113.1时，查询和分析框中的语句变更为如下：

操作	说明	操作后结果
添加到查询	将您单击的关键词通过and拼接后添加到查询和分析框中，并进行查询。	`* and host: "203.0.113.1" | SELECT status as dim, count(1) as c group by dim`
从查询中删除	将您单击的关键词通过not拼接后添加到查询和分析框中，并进行查询。	`* not host: "203.0.113.1" | SELECT status as dim, count(1) as c group by dim`
新建查询	删除查询和分析框中所有语句，按照您单击的关键词新建查询语句，并进行查询。	`* and host: "203.0.113.1"`

高级事件配置

您可以为日志字段添加不同类型的事件，便于深入分析日志。事件行为包括打开日志库、打开快速查询、打开仪表盘和自定义HTTP链接。

在**表格**或**原始**页签中，单击图标下的**事件配置**，进入高级事件配置窗口。

> **说明** 每个日志字段支持配置最多10个高级事件。

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在**原始日志**的**表格**或**原始**页签中，单击图标下的**事件配置**。
5. 在**高级事件配置**对话框的字段列表中，单击目标字段对应的**添加事件**。
6. 在**事件配置**区域，配置高级事件。

   事件行为包括打开日志库、打开快速查询、打开仪表盘和自定义HTTP链接，具体操作如下所示。

   - **打开日志库**

     设置事件为打开日志库，具体配置如下所示。

参数	说明
配置名称	配置名称。
事件行为	选择**打开日志库**。
打开新窗口	开启该选项，则在触发高级事件时将在新窗口打开目标日志库查询页面。
时间范围	设置目标日志库的查询时间范围。可以设置为： ■ **预设**：在原始日志页面中单击字段值，跳转到目标日志库查询页面后，保持日志服务的默认查询时间范围，即15分钟（相对）。 ■ **继承当前时间**：在原始日志页面单击字段值，跳转到目标日志库查询页面后，对应的查询时间范围为查询原始日志时对应的时间。 ■ **相对时间**：在原始日志页面中单击字段值，跳转到目标日志库查询页面后，对应的查询时间范围为当前指定的相对时间。 ■ **整点时间**：在原始日志页面中单击字段值，跳转到目标日志库查询页面后，对应的查询时间范围为当前指定的整点时间。
请选择日志库	选择目标日志库。在触发事件后，将跳转到该日志库的查询页面。
是否继承过滤	打开**是否继承过滤**开关，将当前查询已有的过滤条件同步到跳转后的目标日志库的查询页面中，并以 `AND` 方式添加到查询和分析语句之前。
过滤	在**过滤**页签中输入过滤语句，可将该过滤语句同步到跳转后的目标日志库的查询页面中，并以 `AND` 方式添加到查询和分析语句之前。 支持在过滤语句中插入**可选参数域**，将对应字段的值作为过滤条件。例如单击输入 `${__topic__}` ，跳转后的日志库的查询语句中会拼接 `AND` 后查询。
变量	暂不支持配置变量。

- 打开快速查询

  设置事件为打开快速查询，具体配置如下所示。

参数	说明
配置名称	配置名称。
事件行为	选择**打开快速查询**。
打开新窗口	开启该选项，则在触发事件时将在新窗口打开目标快速查询页面。
时间范围	设置目标快速查询的时间范围。可以设置为： ■ **预设**：在原始日志页面中单击字段值，跳转到目标快速查询的查询页面后，保持日志服务的默认查询时间范围，即15分钟（相对）。 ■ **继承当前时间**：在原始日志页面单击字段值，跳转到目标快速查询的查询页面后，对应的查询时间范围为查询原始日志时对应的时间。 ■ **相对时间**：在原始日志页面中单击字段值，跳转到目标快速查询的查询页面后，对应的查询时间范围为当前指定的相对时间。 ■ **整点时间**：在原始日志页面中单击字段值，跳转到目标快速查询的查询页面后，对应的查询时间范围为当前指定的整点时间。
请选择快速查询	选择目标快速查询。在触发事件后，将跳转到该快速查询页面。

参数	说明
是否继承过滤	打开**是否继承过滤**开关，将当前查询已有的过滤条件同步到跳转后的目标快速查询的查询页面中，并以 `AND` 方式添加到查询和分析语句之前。
过滤	在**过滤**页签中输入过滤语句，可将该过滤语句同步到跳转后的目标快速查询的查询页面中，并以 `AND` 方式添加到查询和分析语句之前。 支持在过滤语句中插入**可选参数域**，将对应字段的值作为过滤条件。例如单击输入 `${__topic__}` ，跳转后的快速查询的查询语句中会拼接 `AND` 后查询。
变量	日志服务支持通过变量灵活修改目标快速查询的查询和分析语句，当此处添加的变量与目标快速查询的查询和分析语句中的变量相同时，会将查询和分析语句中的变量替换为触发事件的字段值。您可以在**变量**页签中配置变量。 **说明** ■ 如果选择添加变量，则需要提前在跳转到的目标快速查询中配置查询和分析语句的占位符变量。更多信息，请参见设置占位符变量。 ■ 最多可添加5个动态变量和5个静态变量。 ■ 动态变量：以您单击触发事件时所在日志对应的字段值作为变量值进行查询。 ■ **动态变量名**：配置变量名。例如您已在快速查询中定义的占位符变量 `dynamic_ip` 。 ■ **动态变量值所在列**：以指定列对应的值动态替换目标快速查询的查询和分析语句中的变量。例如选择 `__source__` 。 表示以 `__source__` 的值替换快速查询中已定义的占位符变量，并进行查询。 ■ 静态变量：以您定义的固定值作为查询值进行查询。 ■ **静态变量名**：配置变量名。例如您已在快速查询中定义的占位符变量 `static_ip` 。 ■ **静态值**：以固定的字段值替换目标快速查询的查询和分析语句中的变量。例如 `203.0.113.1` 。 表示以 `static_ip` 的值 `203.0.113.1` 替换快速查询中已定义的占位符变量，并进行查询。占位符变量值为 `203.0.113.1` 的日志都会被查询到。

- 打开仪表盘

  设置事件为打开仪表盘，具体配置如下所示。

参数	说明
配置名称	配置名称。
事件行为	选择**打开仪表盘**。
打开新窗口	开启该选项，则在触发事件时将在新窗口打开目标仪表盘页面。

参数	说明
时间范围	设置目标仪表盘的时间范围。可以设置为： ■ **预设**：在原始日志页面单击字段值，跳转到目标仪表盘页面后，保持日志服务的默认查询时间范围，即15分钟（相对）。 ■ **继承图表时间**：在原始日志页面单击字段值，跳转到目标仪表盘页面后，对应的查询时间范围为触发下钻事件时图表对应的时间。 ■ **相对时间**：在原始日志页面单击字段值，跳转到目标仪表盘页面后，对应的查询时间范围为当前指定的相对时间。 ■ **整点时间**：在原始日志页面单击字段值，跳转到目标仪表盘页面后，对应的查询时间范围为当前指定的整点时间。
请选择仪表盘	选择目标仪表盘。在触发事件后，将跳转到该仪表盘页面。
是否继承过滤	打开**是否继承过滤**开关，将当前仪表盘已有的过滤条件同步到跳转后的目标仪表盘中。
过滤	在**过滤**页签中输入过滤语句，可将该过滤语句同步到跳转后的目标仪表盘页面中。 支持在过滤语句中插入**可选参数域**，将对应字段的值作为过滤条件。例如单击输入 `${__source__}` ，跳转后的仪表盘的只展示符合 `${__source__}` 取值的日志。
变量	日志服务支持将此处的变量同步到跳转后的目标仪表盘中。您可以在**变量**页签中配置变量。 **说明** ■ 如果选择添加变量，则需要提前在跳转到的目标仪表盘的图表中配置查询和分析语句的占位符变量。更多信息，请参见设置占位符变量。 ■ 最多可添加5个动态变量和5个静态变量。 ■ 动态变量：以您单击触发事件时所在日志对应的字段值作为变量值进行查询。 ■ **动态变量名**：配置变量名。例如您已在快速查询中定义的占位符变量 `dynamic_ip` 。 ■ **动态变量值所在列**：以指定列对应的值动态同步到目标仪表盘中。例如选择 `__source__` 。 表示以 `__source__` 的值替换仪表盘查询中已定义的占位符变量，并进行查询。 ■ 静态变量：以您定义的固定值作为查询值进行查询。 ■ **静态变量名**：配置变量名。例如您已在仪表盘数据源中定义的占位符变量 `static_ip` 。 ■ **静态值**：以固定的字段值同步到目标仪表盘中。例如 `203.0.113.1` 。 表示以 `static_ip` 的值 `203.0.113.1` 替换仪表盘中已定义的占位符变量，并进行查询。占位符变量值为 `203.0.113.1` 的日志都会被查询到。

- 自定义HTTP链接

  设置事件为打开自定义HTTP链接。

  - HTTP链接中的路径部分表示访问的目的端文件的层级路径。
  - 您可以在定义HTTP链接的路径部分添加可选参数域。当您单击原始日志中的高级事件时，会用字段值替换HTTP链接中的可选域参数，跳转到重新定位的HTTP链接中。

配置	说明
配置名称	配置名称。
事件行为	选择**自定义HTTP链接**。
协议	访问自定义链接的协议类型。支持HTTP、自定义等。
请输入链接地址	需要跳转到的目标地址。 例如 `www.example.com/s?wd=${sls_project}` ，表示跳转到这个地址。事件触发后，${sls_project}替换为您所在Project名称。
使用系统变量	打开**使用系统变量**开关后，可选择日志服务的系统变量插入到HTTP链接中，包括${sls_project}、${sls_dashboard_title}、${sls_chart_name}、${sls_chart_title}、${sls_region}、${sls_start_time}、${sls_end_time}、${sls_realUid}和${sls_aliUid}。
是否转码	打开**是否转码**开关后，会将链接中的内容进行encode转码。
可选参数域	日志服务支持将链接中的某一部分替换为触发事件的字段值。触发事件后，自动替换为字段值。

示例

将访问日志采集到名为accesslog的日志库中并创建1个快速查询（IP地址和method的PV分布）实现下钻分析。在原始日志中，为remote_addr字段设置高级事件为打开快速查询。设置完成后，您在原始日志中单击remote_addr即可跳转到快速查询查看对应的PV趋势。

原始日志参考如下：

```
__source__:127.0.0.1
__tag__:__receive_time__:1613759995
__topic__:nginx_access_log
body_bytes_sent:5077
host:www.example.com
http_referer:www.example.com
http_user_agent:Mozilla/5.0 (X11; CrOS i686 12.0.742.91) AppleWebKit/534.30 (KHTML, like Gecko) Chrom
e/192.0.2.2 Safari/534.30
http_x_forwarded_for:192.0.2.1
remote_addr:192.0.2.0
remote_user:gp_02
request_length:3932
request_method:POST
request_time:35
request_uri:/request/path-2/file-4
status:200
time_local:19/Feb/2021:18:39:50
upstream_response_time:0.09
```

其操作步骤如下：

1. 查询请求方法为POST且status为200的访问PV分布情况。创建快速查询**IP地址和method的PV分布**，其查询语句和查询结果如下：

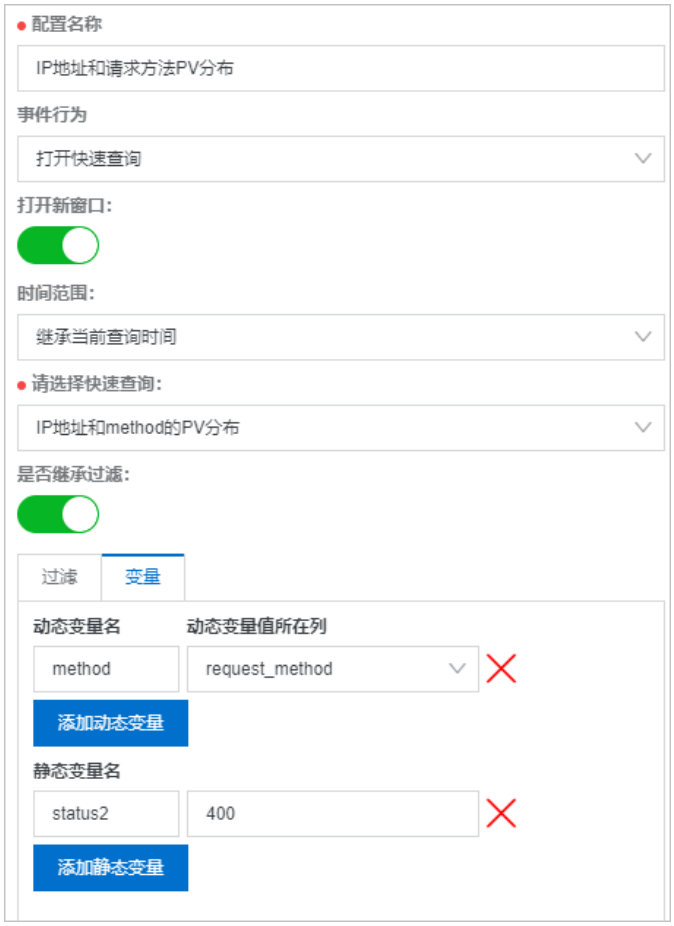
   ```
   * and request_method: POST and status: 200 | select count(*) as pv, remote_addr as ip,request_met
   hod as method group by ip,method order by ip desc
   ```

pv	ip	method
1	99 [redacted] 38	POST
1	99 [redacted] 46	POST
1	98 [redacted] 97	POST
1	98 [redacted] 9	POST
1	98 [redacted] 85	POST
1	98 [redacted] 77	POST

2. 在快速查询中设置变量 `method` 和 `status2` 。生成变量后的语句变更为如下：

```
* and request_method: ${method} and status: ${status2} | select count(*) as pv, remote_addr as ip
,request_method as method group by ip,method order by ip desc
```

3. 在原始日志页签，为remote_addr配置高级事件，事件行为**打开快速查询**，其他关键配置如下：
   - 请选择快速查询：IP地址和method的PV分布
   - 过滤：不配置
   - 变量：静态变量为status2，取值为400。动态变量取值为method，取值为request_method。



4. 在原始日志页签，单击**remote_addr > IP地址和method的PV分布**。

该日志字段中request_method为GET，status为404。

5. 新打开窗口，查询和分析框中语句变更为如下：

```
* and request_method: GET and status: 400 | select count(*) as pv, remote_addr as ip,request_meth
od as method group by ip,method order by ip desc
```

6. 查看快速查询结果。

该示例中，静态变量status2取值为400，对应status字段。您单击触发事件所在日志的request_method取值为GET，所以动态变量method取值为GET。快速查询的结果是GET中status为400的IP地址PV分布。

同样的，当您单击触发事件所在日志的request_method取值为PUT时，快速查询的是PUT中status为400的IP地址的PV分布。

pv	ip	method
1	1	GET
1	6	GET
1		GET
1		GET
1		GET
1		GET
1		GET
1		GET

11.SQL分析语法与功能
11.1. SQL函数
11.1.1. 函数概览

本文列举SQL分析所涉及的函数与运算符。

聚合函数

函数名称	说明
arbitrary函数	返回x中任意一个非空的值。
avg函数	计算x中的算术平均值。
bitwise_and_agg函数	返回x中所有值按位与运算（AND）的结果。
bitwise_or_agg函数	返回x中所有值按位或运算（OR）的结果。
bool_and函数	判断是否所有日志都满足条件。如果是，则返回true。 bool_and函数等同于every函数。
bool_or函数	判断是否存在日志满足条件。如果存在，则返回true。
checksum函数	计算x的校验和。
count函数	统计所有的日志条数。
	统计所有的日志条数，等同于count(*)。
	统计x中值不为NULL的日志条数。
count_if函数	统计满足指定条件的日志条数。
every函数	判断是否所有日志都满足条件。如果是，则返回true。 every函数等同于bool_and函数。
geometric_mean函数	计算x的几何平均数。
kurtosis函数	计算x的峰度。
map_union函数	返回一列Map数据的并集。 如果Map中存在相同的键，则返回的键值为其中任意一个键的值。
max函数	查询x中的最大值。
	查询x中最大的n个值。返回结果为数组。
max_by函数	查询y为最大值时对应的x值。
	查询最大的n个y值对应的x值，返回结果为数组。
min函数	查询x中最小值。
	查询x中最小的n个值。返回结果为数组。

函数名称	说明
min_by函数	查询y为最小值时对应的x值。
	查询最小的n个y值对应的x值。返回结果为数组。
skewness函数	计算x的偏度。
sum函数	计算x的总值。

字符串函数

函数名称	说明
chr函数	将ASCII码转换为字符。
codepoint函数	将字符转换为ASCII码。
concat函数	将多个字符串拼接成一个字符串。
from_utf8函数	将二进制字符串解码为UTF-8编码格式，并使用默认字符U+FFFD替换无效的UTF-8字符。
	将二进制字符串解码为UTF-8编码格式，并使用自定义字符串替换无效的UTF-8字符。
length函数	计算字符串的长度。
levenshtein_distance函数	计算x和y之间的最小编辑距离。
lower函数	将字符串转换为小写形式。
lpad函数	在字符串的开头填充指定字符，直到指定长度后返回结果字符串。
ltrim函数	删除字符串开头的空格。
normalize函数	使用NFC格式将字符串格式化。
position函数	返回目标子串在字符串中的位置。
replace函数	将字符串中所匹配的字符替换为其他指定字符。
	删除字符串中匹配的字符。
reverse函数	返回反向顺序的字符串。
rpad函数	在字符串的尾部填充指定字符，直到指定长度后返回结果字符串。
rtrim函数	删除字符串中结尾的空格。
split函数	使用指定的分隔符拆分字符串，并返回子串集合。
	通过指定的分隔符拆分字符串并使用limit限制字符串拆分的个数，然后返回拆分后的子串集合。
split_part函数	使用指定的分隔符拆分字符串，并返回指定位置的内容。
split_to_map函数	使用指定的第一个分隔符拆分字符串，然后再使用指定的第二个分隔符进行第二次拆分。

函数名称	说明
strpos函数	返回目标子串在字符串中的位置。与position(sub_string in x)函数等价。
substr函数	返回字符串中指定位置的子串，并指定子串长度。
	返回字符串中指定位置的子串。
to_utf8函数	将字符串转换为UTF-8编码格式。
trim函数	删除字符串中开头和结尾的空格。
upper函数	将字符串转化为大写形式。

日期和时间函数

函数名称	说明
current_date函数	返回当前日期。
current_time函数	返回当前时间和时区。
current_timestamp函数	返回当前日期、时间和时区。
current_timezone函数	返回当前时区。
date函数	返回日期和时间表达式中的日期部分。
date_format函数	将timestamp类型的日期和时间表达式转化为指定格式的日期和时间表达式。
date_parse函数	将日期和时间字符串转换为指定格式的timestamp类型的日期和时间表达式。
from_iso8601_date函数	将ISO8601格式的日期表达式转化为date类型的日期表达式。
from_iso8601_timestamp函数	将ISO8601格式的日期和时间表达式转化为timestamp类型的日期和时间表达式。
from_unixtime函数	将UNIX时间戳转化为无时区的timestamp类型的日期和时间表达式。
	将UNIX时间戳转化为带时区的timestamp类型的日期和时间表达式。
	将UNIX时间戳转化为带时区的timestamp类型的日期和时间表达式，其中hours和minutes为时区偏移量。
localtime函数	返回本地时间。
localtimestamp函数	返回本地日期和时间。
now函数	返回当前日期和时间。 now函数等同于current_timestamp函数。
to_iso8601函数	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。
to_unixtime函数	将timestamp类型的日期和时间表达式转化成UNIX时间戳。
day函数	提取日期和时间表达式中的天数，按月计算。 day函数等同于day_of_month函数。

函数名称	说明
day_of_month函数	提取日期和时间表达式中的天数，按月计算。 day_of_month函数等同于day函数。
day_of_week函数	提取日期和时间表达式中的天数，按周计算。 day_of_week函数等同于dow函数。
day_of_year函数	提取日期和时间表达式中的天数，按年计算。 day_of_year函数等同于doy函数。
dow函数	提取日期和时间表达式中的天数，按周计算。 dow函数等同于day_of_week函数。
doy函数	提取日期和时间表达式中的天数，按年计算。 doy函数等同于day_of_year函数。
extract函数	通过指定的field，提取日期和时间表达式中的日期或时间部分。
hour函数	提取日期和时间表达式中的小时数，按24小时制计算。
minute函数	提取日期和时间表达式中的分钟数。
month函数	提取日期和时间表达式中的月份。
quarter函数	计算目标日期所属的季度。
second函数	提取日期和时间表达式中的秒数。
timezone_hour函数	计算时区的小时偏移量。
timezone_minute函数	计算时区的分钟偏移量。
week函数	计算目标日期是在一年中的第几周。 week函数等同于week_of_year函数。
week_of_year函数	计算目标日期是在一年中的第几周。 week_of_year函数等同于week函数。
year函数	提取目标日期中的年份。
year_of_week函数	提取目标日期在ISO周日历中的年份。 year_of_week函数等同于yow函数。
yow函数	提取目标日期在ISO周日历中的年份。 yow函数等同于year_of_week函数。
date_trunc函数	根据您指定的时间单位截断日期和时间表达式，并按照毫秒、秒、分钟，小时、日、月或年对齐。
date_add函数	在x上加上N个时间单位（unit）。

函数名称	说明
date_diff函数	返回两个时间表达式之间的时间差值，例如计算x和y之间相差几个时间单位（unit）。
time_series函数	补全您查询时间窗口内缺失的数据。

JSON函数

函数名称	说明
json_array_contains函数	判断JSON数组中是否包含某个值。
json_array_get函数	获取JSON数组中某个下标对应的元素。
json_array_length函数	计算JSON数组中元素的数量。
json_extract函数	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	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	把JSON类型转化成字符串类型。
json_parse函数	把字符串类型转化成JSON类型。
json_size函数	计算JSON对象或数组中元素的数量。

正则式函数

函数名称	说明
regexp_extract_all函数	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。
	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函数	提取并返回目标字符串中符合正则表达式的第一个子串。
	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	判断目标字符串是否符合正则表达式。
regexp_replace函数	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。
	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	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。

同比与环比函数

函数名称	说明
compare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。
	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。

函数名称	说明
ts_compare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。 注意 ts_compare函数必须按照时间列进行分组（GROUP BY）。
	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。

数组函数和运算符

函数名称	说明
下标运算符	返回数组中的第x个元素。
array_agg函数	以数组形式返回x中的所有值。
array_distinct函数	删除数组中重复的元素。
array_except函数	计算两个数组的差集。
array_intersect函数	计算两个数组的交集。
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。如果数组中包含null元素，则null元素将被忽略。
	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_max函数	获取数组中的最大值。
array_min函数	获取数组中的最小值。
array_position函数	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	删除数组中指定的元素。
array_sort函数	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_transpose函数	对矩阵进行转置，即提取二维数组中索引相同的元素组成一个新的二维数组。
array_union函数	计算两个数组的并集。
cardinality函数	计算数组中元素的个数。
concat函数	将多个数组拼接为一个数组。
contains函数	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	返回数组中的第y个元素。
filter函数	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	把将二维数组转换为一维数组。
reduce函数	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。

函数名称	说明
reverse函数	对数组中的元素进行反向排列。
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。递增间隔为默认值1。
	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	对数组元素进行随机排列。
slice函数	获取数组的子集。
transform函数	将Lambda表达式应用到数组的每个元素中。
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。
zip_with函数	根据Lambda表达式中的定义将两个数组合并为一个数组。

Map映射函数和运算符

函数名称	说明
下标运算符	获取Map中目标键的值。
cardinality函数	计算Map的大小。
element_at函数	获取Map中目标键的值。
histogram函数	对查询和分析结果进行分组，返回结果为JSON格式。
histogram_u函数	对查询和分析结果进行分组，返回结果为多行多列格式。
map函数	返回一个空Map。
	将两个数组映射为一个Map。
map_agg函数	将x和y映射为一个Map。x为Map中的键，y为Map中的键值。当y存在多个值时，随机提取一个值作为键值。
map_concat函数	将多个Map合并为一个Map。
map_filter函数	结合Lambda表达式，用于过滤Map中的元素。
map_keys函数	提取Map中所有的键，并以数组形式返回。
map_values函数	提取Map中所有键的值，并以数组形式返回。
multimap_agg函数	将x和y映射为一个Map。x为Map中的键，y为Map中的键值，键值为数组格式。当y存在多个值时，提取所有的值作为键值。

数学计算函数

函数名称	说明
abs函数	计算x的绝对值。

函数名称	说明
acos函数	计算x的反余弦。
asin函数	计算x的反正弦。
atan函数	计算x的反正切。
atan2函数	计算x和y相除的结果的反正切。
cbrt函数	计算x的立方根。
ceil函数	对x进行向上取整数。 ceil函数是ceiling函数的别名。
ceiling函数	对x进行向上取整数。
cos函数	计算x的余弦。
cosh函数	计算x的双曲余弦。
cosine_similarity函数	计算x和y之间的余弦相似度。
degrees函数	将弧度转换为度。
e函数	返回自然底数e的值。
exp函数	计算自然底数e的x次幂。
floor函数	对x进行向下取整数。
from_base函数	根据BASE编码将x转为y进制的数字。
ln函数	计算x的自然对数。
infinity函数	返回正无穷的数值。
is_nan函数	判断x是否为NaN。
log2函数	计算x以2为底的对数。
log10函数	计算x以10为底的对数。
log函数	计算x以y为底的对数。
mod函数	计算x与y相除的余数。
nan函数	返回一个NaN值。
pi函数	返回π值，精确到小数点后15位。
pow函数	计算x的y次幂。 pow函数是power函数的别名。
power函数	计算x的y次幂。
radians函数	将度转换为弧度。

函数名称	说明
rand函数	返回随机数。
random函数	返回[0,1)之间的随机数。
	返回[0,x)之间的随机数。
round函数	对x进行四舍五入取整数。
	对x进行四舍五入且保留n位小数。
sign函数	返回x的符号，通过1、0、-1表示。
sin函数	计算x的正弦。
sqrt函数	计算x的平方根。
tan函数	计算x的正切。
tanh函数	计算x的双曲正切。
to_base函数	根据BASE编码将x转为y进制的字符串。
truncate函数	截断x的小数部分。
width_bucket函数	将一段数值范围划分成大小相同的多个Bucket，然后返回x所属的Bucket。
	使用数组指定Bucket的范围，然后返回x所属的Bucket。

数学统计函数

函数名称	说明
corr函数	计算x和y的相关度。计算结果范围为[0,1]。
covar_pop函数	计算x和y的总体协方差。
covar_samp函数	计算x和y的样本协方差。
regr_intercept函数	根据输入点 `(x,y)` 拟合成一个线性方程，然后计算该直线的Y轴截距。
regr_slope函数	根据输入点 `(x,y)` 拟合成一个线性方程，然后计算该直线的斜率。
stddev函数	计算x的样本标准差。与stddev_samp函数同义。
stddev_samp函数	计算x的样本标准差。
stddev_pop函数	计算x的总体标准差。
variance函数	计算x的样本方差。与var_samp函数同义。
var_samp函数	计算x的样本方差。
var_pop函数	计算x的总体方差。

类型转换函数

函数名称	说明
cast函数	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	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。 **说明** 日志中可能有脏数据，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。
typeof函数	返回x的数据类型。

安全检测函数

函数名称	说明
security_check_ip函数	检查IP地址是否安全。
security_check_domain函数	检查域名是否安全。
security_check_url函数	检查URL是否安全。

窗口函数

函数名称	说明
聚合函数	所有聚合函数都支持在窗口函数中使用。聚合函数列表请参见聚合函数。
cume_dist函数	统计窗口分区内各个值的累计分布。即计算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。返回值范围为(0,1]。
dense_rank函数	窗口分区内值的排名。相同值拥有相同的排名，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。
ntile函数	将窗口分区内数据按照顺序分成n组。
percent_rank函数	计算窗口分区内各行的百分比排名。
rank函数	窗口分区内值的排名。相同值拥有相同的排名，排名不是连续的，例如有两个相同值的排名为1，则下一个值的排名为3。
row_number函数	窗口分区内值的排名。每个值拥有唯一的序号，从1开始。三个相同值的排名为1、2、3。
first_value函数	返回各个窗口分区内第一行的值。
last_value函数	返回各个窗口分区内最后一行的值。
lag函数	返回窗口分区内位于当前行上方第offset行的值。如果不存在该行，则返回defaut_value。
lead函数	返回窗口分区内位于当前行下方第offset行的值。如果不存在该行，则返回defaut_value。

函数名称	说明
nth_value函数	返回窗口分区中第offset行的值。

IP函数

函数名称	说明
ip_to_city函数	分析目标IP地址所属城市。 返回结果为城市的中文名称。
	分析目标IP地址所属城市。 返回结果为城市的行政区划代码。
ip_to_city_geo函数	分析目标IP地址所属城市的经纬度。此函数返回的是城市经纬度，每个城市只有一个经纬度。
ip_to_country函数	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。
	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代码。
ip_to_country_code函数	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代码。
ip_to_domain函数	判断目标IP地址是内网地址还是外网地址。
ip_to_geo函数	分析目标IP地址所在位置的经纬度。
ip_to_provider函数	分析目标IP地址所对应的网络运营商。
ip_to_province函数	分析目标IP地址所属省份。 返回结果为省份的中文名称。
	分析目标IP地址所属省份。 返回结果为省份的行政区划代码。
ip_prefix函数	获取目标IP地址的前缀。
is_prefix_subnet_of函数	判断目标网段是否为某网段的子网。
is_subnet_of函数	判断目标IP地址是否在某网段内。
ip_subnet_max函数	获取IP网段中的最大IP地址。
ip_subnet_min函数	获取IP网段中的最小IP地址。
ip_subnet_range函数	获取IP网段范围。

URL函数

函数名称	说明
url_encode函数	对URL进行编码。
url_decode函数	对URL进行解码。
url_extract_fragment函数	从URL中提取Fragment信息。
url_extract_host函数	从URL中提取Host信息。
url_extract_parameter函数	从URL的查询部分中提取指定参数的值。
url_extract_path函数	从URL中提取访问路径信息。
url_extract_port函数	从URL中提取端口信息。
url_extract_protocol函数	从URL中提取协议信息。
url_extract_query函数	从URL中提取查询部分的信息。

估算函数

函数名称	说明
approx_distinct函数	估算x中不重复值的个数，默认存在2.3%的标准误差。
	估算xx中不重复值的个数，支持自定义标准误差。
approx_percentile函数	对x进行正序排列，返回大约处于percentage位置的x。
	对x进行正序排列，返回大约处于percentage01、percentage02位置的x。
	对x和权重的乘积进行正序排列，返回大约处于percentage位置的x。
	对x和权重的乘积进行正序排列，返回大约处于percentage01、percentage02位置的x。
	对x和权重的乘积进行正序排列，返回大约处于percentage位置的x。支持设置返回结果的准确度。
numeric_histogram函数	按照bucket数量（直方图列数），统计x的近似直方图，返回结果为JSON类型。
	按照bucket数量（直方图列数），统计x的近似直方图，返回结果为JSON类型。支持对x设置权重。
numeric_histogram_u函数	按照bucket数量（直方图列数），统计x的近似直方图，返回结果为多行多列格式。

二进制函数

函数名称	说明
from_base64函数	将BASE64编码的字符串解码为二进制类型的数据。
from_base64url函数	使用URL安全字符将BASE64编码的字符串解码为二进制类型的数据。
from_big_endian_64函数	将大端模式的二进制类型的数据转化成数字。
from_hex函数	将十六进制类型的数据转化成二进制类型的数据。

函数名称	说明
length函数	计算二进制类型的数据的长度。
md5函数	对二进制类型的数据进行MD5编码。
to_base64函数	对二进制类型的数据进行BASE64编码。
to_base64url函数	使用URL安全字符将二进制类型的数据进行BASE64编码。
to_hex函数	将二进制类型的数据转化成十六进制类型的数据。
to_big_endian_64函数	将数字转化为大端模式的二进制类型的数据。
sha1函数	对二进制类型的数据进行SHA1加密。
sha256函数	对二进制类型的数据进行SHA256加密。
sha512函数	对二进制类型的数据进行SHA512加密。
xxhash64函数	对二进制类型的数据进行xxHash64加密。

位运算函数

函数名称	说明
bit_count函数	统计x中1的个数。
bitwise_and函数	以二进制形式对x和y进行与运算。
bitwise_not函数	以二进制形式对x的所有位进行取反运算。
bitwise_or函数	以二进制形式对x和y进行或运算。
bitwise_xor函数	以二进制形式对x和y进行异或运算。

空间几何函数

函数名称	说明
ST_AsText函数	将一个空间几何体转变为WKT格式的文本。
ST_GeometryFromText函数	根据输入的WKT文本构造一个空间几何体。
ST_LineFromText函数	根据输入的WKT文本构造一条线段。
ST_Polygon函数	根据输入的WKT文本构造一个多边形。
ST_Point函数	根据输入的WKT文本构造一个点。
ST_Boundary函数	返回空间几何体的边界。
ST_Buffer函数	返回距离指定空间几何体一定距离的空间几何体。
ST_Difference函数	返回两个空间几何体不同点的集合。
ST_Envelope函数	返回空间几何体的最小边界框。

函数名称	说明
ST_ExteriorRing函数	返回空间几何体的外环（线段形式）。
ST_Intersection函数	返回两个空间几何体的交集点。
ST_SymDifference函数	返回两个空间几何体的不同点，然后组成一个新的空间几何体。
ST_Contains函数	判断第一个空间几何体是否包含第二个空间几何体（边界可存在交集）。如果包含，则返回true。
ST_Crosses函数	判断两个空间几何体是否存在相同的内部点。如果存在，则返回true。
ST_Disjoint函数	判断两个空间几何体是否没有任何交集。 如果没有，则返回true。
ST_Equals函数	判断两个空间几何体是否完全相同。如果是，则返回true。
ST_Intersects函数	判断两个空间几何体的平面投影是否存在共同点。如果是，则返回true。
ST_Overlaps函数	判断两个空间几何体的维度是否相同。如果两个空间几何体的维度相同且不是包含关系，则返回true。
ST_Relate函数	判断两个空间几何体是否相关。如果是，则返回true。
ST_Touches函数	判断两个空间几何体是否只有边界存在关联，没有共同的内部点。如果是，则返回true。
ST_Within函数	判断第一个空间几何体是否完全在第二个空间几何体内部（边界无交集）。如果是，则返回true。
ST_Area函数	使用欧几里得测量法计算空间几何体在二维平面上的投影面积。
ST_Centroid函数	返回空间几何实体的中心点。
ST_CoordDim函数	返回空间几何体的坐标维度。
ST_Dimension函数	返回空间几何实体的固有维度，必须小于或等于坐标维度。
ST_Distance函数	计算两个空间几何体之间的最小距离。
ST_EndPoint函数	返回线段中的最后一个点。
ST_IsClosed函数	判断输入的空间几何体是否封闭。如果是，则返回true。
ST_IsEmpty函数	判断输入的空间几何体是否为空。如果是，则返回true。
ST_IsRing函数	判断输入的空间几何体是否为闭合的简单线段（环）。如果是，则返回true。
ST_Length函数	使用欧几里得测量法计算线段的二维投影长度。如果存在多条线段，则返回所有线段的长度之和。
ST_NumPoints函数	返回空间几何体中点的个数。
ST_NumInteriorRing函数	计算空间几何体中内部环的数量。
ST_StartPoint函数	返回线段中的第一个点。
ST_X函数	返回输入点的X轴坐标。
ST_XMax函数	返回空间几何体的第一个最大的X轴坐标。

函数名称	说明
ST_XMin函数	返回空间几何体的第一个最小的X轴坐标。
ST_Y函数	返回输入点的Y轴坐标。
ST_YMax函数	返回空间几何体的第一个最大的Y轴坐标。
ST_YMin函数	返回几何体的第一个最小的Y轴坐标。
bing_tile函数	通过X坐标、Y坐标和缩放级别构造一个Bing图块。
	通过四叉树键构造一个Bing图块。
bing_tile_at函数	通过经纬度和缩放级别构造一个Bing图块。
bing_tile_coordinates函数	返回目标Bing图块对应的X坐标和Y坐标。
bing_tile_polygon函数	返回目标Bing图块的多边形格式。
bing_tile_quadkey函数	返回目标Bing图块的四叉树键。
bing_tile_zoom_level函数	返回目标Bing图块的缩放级别。

地理函数

函数名称	说明
geohash函数	对纬度和经度进行geohash编码。

颜色函数

函数名称	说明
bar函数	通过width指定整条ANSI条形图的宽度，其中该ANSI条形图的起始颜色为红色（low_color），结束颜色为绿色（high_color）。然后通过x截取其中一段ANSI条形图并返回。
	通过width指定整条ANSI条形图的宽度，其中该ANSI条形图的起始颜色和结束颜色为自定义颜色。然后通过x截取其中一段ANSI条形图并返回。
color函数	将颜色字符串转换为color类型。
	通过判断x在low和high之间的占比指定low_color和high_color的份量，然后返回处于low_color和high_color之间的一个颜色。
	通过y指定low_color和high_color的份量，然后返回处于low_color和high_color之间的一个颜色。
render函数	通过颜色渲染返回结果。布尔表达式为真时，返回绿色勾；否则返回红色叉。
	通过自定义的颜色渲染返回结果。
rgb函数	通过RGB值返回一个颜色值。

HyperLogLog函数

函数名称	说明
approx_set函数	估算x中不重复值的个数，最大标准误差默认为0.01625。
cardinality函数	将HyperLogLog类型的内容转换为bigint类型。
empty_approx_set函数	返回一个HyperLogLog类型的空值。最大标准误差默认为0.01625。
merge函数	聚合计算所有的HyperLogLog值。

电话号码函数

函数名称	说明
mobile_carrier函数	分析电话号码所属运营商。
mobile_city函数	分析电话号码所属城市。
mobile_province函数	分析电话号码所属省份。

比较运算符

运算符	说明
基础运算符	比较x和y的大小关系。如果逻辑成立，则返回true。
ALL运算符	x满足所有条件时，返回true。
ANY运算符	x满足任意一个条件时，返回true。
BETWEEN运算符	x处在y和z之间时，返回true。
DISTINCT运算符	（x IS DISTINCT FROM y）x不等于y时，返回true。
	（x IS NOT DISTINCT FROM y）x等于y时，返回true。
LIKE运算符	匹配字符串中指定的字符模式。字符串区分大小写。
SOME运算符	x满足任意一个条件时，返回true。
GREATEST运算符	查询x、y中的最大值。
LEAST运算符	查询x、y中的最小值。
NULL运算符	（x IS NULL）x为null时，返回true。
	（x IS NOT NULL）x不为null时，返回true。

逻辑运算符

运算符	说明
AND运算符	x和y的值都为true时，返回结果为true。
OR运算符	x和y中任意一个的值为true时，返回结果为true。
NOT运算符	x的值为false时，返回结果为true。

单位换算函数

函数名称	说明
convert_data_size函数	对数据量单位进行换算，系统自动判断最优的换算单位，返回使用最优单位表示的数据量。返回类型为string。例如将1024 KB换算为1 MB，1024 MB换算为1 GB。
	对数据量单位进行换算，返回使用指定单位表示的数据量。返回类型为string。
format_data_size函数	对Byte单位进行换算，返回使用指定单位表示的数据量。返回类型为string。
parse_data_size函数	对数据量单位进行换算，返回以Byte为单位的数据量。返回类型为decimal。
to_data_size_B函数	对数据量单位进行换算，返回以Byte为单位的数据量。返回类型为double。
to_data_size_KB函数	对数据量单位进行换算，返回以KB为单位的数据量。返回类型为double。
to_data_size_MB函数	对数据量单位进行换算，返回以MB为单位的数据量。返回类型为double。
to_data_size_GB函数	对数据量单位进行换算，返回以GB为单位的数据量。返回类型为double。
to_data_size_TB函数	对数据量单位进行换算，返回以TB为单位的数据量。返回类型为double。
to_data_size_PB函数	对数据量单位进行换算，返回以PB为单位的数据量。返回类型为double。
format_duration函数	对以秒为单位的时间间隔进行格式化，转换为可读的字符串类型。
parse_duration函数	对时间间隔进行格式化，转换为 `0 00:00:00.000` 格式。
to_days函数	对时间间隔单位进行换算，转换为以天为单位的时间间隔。
to_hours函数	对时间间隔单位进行换算，转换为以小时为单位的时间间隔。
to_microseconds函数	对时间间隔单位进行换算，转换为以微秒为单位的时间间隔。
to_milliseconds函数	对时间间隔单位进行换算，转换为以毫秒为单位的时间间隔。
to_minutes函数	对时间间隔单位进行换算，转换为以分钟为单位的时间间隔。
to_most_succinct_time_unit函数	对时间间隔单位进行换算，系统自动判断最优的换算单位，返回使用最优单位表示的时间间隔。
to_nanoseconds函数	对时间间隔单位进行换算，转换为以纳秒为单位的时间间隔。
to_seconds函数	对时间间隔单位进行换算，转换为以秒为单位的时间间隔。

窗口漏斗函数

函数名称	说明
window_funnel函数	在滑动的时间窗口中搜索事件链并计算事件链中发生的最大连续的事件数。

Lambda表达式

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。更多信息，请参见Lambda表达式。

条件表达式

表达式	说明
CASE WHEN表达式	通过条件判断，对数据进行归类。
IF表达式	通过条件判断，对数据进行归类。
COALESCE表达式	返回多个表达式中第一个非NULL的值。
NULLIF表达式	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	捕获异常信息，使得系统继续执行查询和分析操作。

11.1.2. 聚合函数

聚合函数用于对目标数值执行计算并返回结果。本文介绍聚合函数的基本语法及示例。

日志服务支持如下聚合函数。

> 注意 在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。

函数名称	语法	说明
arbitrary函数	arbitrary(*x*)	返回*x*中任意一个非空的值。
avg函数	avg(*x*)	计算*x*的算术平均值。
bitwise_and_agg函数	bitwise_and_agg(*x*)	返回*x*中所有值按位与运算（AND）的结果。
bitwise_or_agg函数	bitwise_or_agg(*x*)	返回*x*中所有值按位或运算（OR）的结果。
bool_and函数	bool_and(*boolean expression*)	判断是否所有日志都满足条件。如果是，则返回true。 bool_and函数等同于every函数。
bool_or函数	bool_or(*boolean expression*)	判断是否存在日志满足条件。如果存在，则返回true。
checksum函数	checksum(*x*)	计算*x*的校验和。
count函数	count(*)	统计所有的日志条数。
	count(1)	统计所有的日志条数，等同于count(*)。
	count(*x*)	统计*x*中值不为NULL的日志条数。
count_if函数	count_if(*boolean expression*)	统计满足指定条件的日志条数。
every函数	every(*boolean expression*)	判断是否所有日志都满足条件。如果是，则返回true。 every函数等同于bool_and函数。
geometric_mean函数	geometric_mean(*x*)	计算*x*的几何平均数。
kurtosis函数	kurtosis(*x*)	计算*x*的峰度。

函数名称	语法	说明
map_union函数	map_union(*x*)	返回一列Map数据的并集。 如果Map中存在相同的键，则返回的键值为其中任意一个键的值。
max函数	max(*x*)	查询*x*中的最大值。
	max(*x*, *n*)	查询*x*中最大的*n*个值。返回结果为数组。
max_by函数	max_by(*x*, *y*)	查询*y*为最大值时对应的*x*值。
	max_by(*x*, *y*, *n*)	查询最大的*n*个*y*值对应的*x*值，返回结果为数组。
min函数	min(*x*)	查询*x*中最小值。
	min(*x*, *n*)	查询*x*中最小的*n*个值。返回结果为数组。
min_by函数	min_by(*x*, *y*)	查询*y*为最小值时对应的*x*值。
	min_by(*x*, *y*, *n*)	查询最小的*n*个*y*值对应的*x*值。返回结果为数组。
skewness函数	skewness(*x*)	计算*x*的偏度。
sum函数	sum(*x*)	计算*x*的总值。

arbitrary函数

arbitrary函数用于返回*x*中任意一个非空的值。

```
arbitrary(x)
```

参数	说明
x	参数值为任意数据类型。

与参数值的数据类型一致。

返回request_method字段中任意一个非空的字段值。

- 查询和分析语句

```
* | SELECT arbitrary(request_method) AS request_method
```

- 查询和分析结果

request_method
GET

avg函数

avg函数用于计算*x*的算术平均值。

```
avg(x)
```

参数	说明
x	参数值为double、bigint、decimal或real类型。

double类型。

返回平均延迟时间高于1000微秒的Project。

- 查询和分析语句

```
method: PostLogstoreLogs | SELECT avg(latency) AS avg_latency, Project GROUP BY Project HAVING avg
_latency > 1000
```

- 查询和分析结果

avg_latency	Project
3223.4162679425835	datalab-14[redacted]61-cn-chengdu

bitwise_and_agg函数

bitwise_and_agg函数用于返回*x*中所有值按位与运算（AND）的结果。

```
bitwise_and_agg(x)
```

参数	说明
x	参数值为bigint类型。

bigint类型（二进制形式）。

对request_time字段的所有值进行按位与运算。

- 查询和分析语句

```
* | SELECT bitwise_and_agg(status)
```

- 查询和分析结果

_col0
0

bitwise_or_agg函数

bitwise_or_agg函数用于返回*x*中所有值按位或运算（OR）的结果。

```
bitwise_or_agg(x)
```

参数	说明
x	参数值为bigint类型。

bigint类型（二进制形式）。

对request_time字段的所有值进行按位或运算。

- 查询和分析语句

```
* | SELECT bitwise_or_agg(request_length)
```

- 查询和分析结果

_col0
16383

bool_and函数

bool_and函数用于判断是否所有日志都满足条件。如果是，则返回true。bool_and函数等同于every函数。

```
bool_and(boolean expression)
```

参数	说明
boolean expression	参数值为布尔表达式。

boolean类型。

判断所有请求的时间是否都小于100秒。如果是，则返回true。

- 查询和分析语句

```
* | SELECT bool_and(request_time < 100)
```

- 查询和分析结果

_col0
true

bool_or函数

bool_or函数用于判断是否存在日志满足条件。如果存在，则返回true。

```
bool_or(boolean expression)
```

参数	说明
boolean expression	参数值为布尔表达式。

boolean类型。

判断是否存在请求时间小于20秒的请求。如果存在，则返回true。

- 查询和分析语句

```
* | SELECT bool_or(request_time < 20)
```

- 查询和分析结果

_col0
true

checksum函数

checksum函数用于计算*x*的校验和。

```
checksum(x)
```

参数	说明
x	参数值为任意数据类型。

string类型（BASE 64编码）。

- 查询和分析语句

```
* | SELECT checksum(request_method) AS request_method
```

- 查询和分析结果

request_method
NDXFdgnd8GE=

count函数

count函数用于计数。

- 统计所有的日志条数。

```
count(*)
```

- 统计所有的日志条数。等同于 `count(*)` 。

```
count(1)
```

- 统计*x*中值不为NULL的日志条数。

```
count(x)
```

参数	说明
x	参数值为任意数据类型。

integer类型。

- 示例1：统计网站访问量。
  - 查询和分析语句

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

  - 查询和分析结果

PV
2009

- 示例2：统计包含request_method字段且字段值不为NULL的日志条数。
  - 查询和分析语句

```
* | SELECT count(request_method) AS count
```

  - 查询和分析结果

count
1954

count_if函数

count_if函数用于统计满足指定条件的日志条数。

```
count_if(boolean expression)
```

参数	说明
boolean expression	参数值为布尔表达式。

integer类型。

统计request_uri字段的值是以 `file-0` 结尾的日志条数。

- 查询和分析语句

```
* | SELECT count_if(request_uri like '%file-0') AS count
```

- 查询和分析结果

count
1954

geometric_mean函数

geometric_mean函数用于计算*x*的几何平均数。

```
geometric_mean(x)
```

参数	说明
x	参数值为double、bigint或real类型。

double类型。

统计请求时长的几何平均值。

- 查询和分析语句

```
* | SELECT geometric_mean(request_time) AS time
```

- 查询和分析结果

time
39.443123208882308

every函数

every函数用于判断是否所有日志都满足条件。如果是，则返回true。every函数等同于bool_and函数。

```
every(boolean expression)
```

参数	说明
boolean expression	参数值为布尔表达式。

boolean类型。

判断所有请求的时间是否都小于100秒。如果是，则返回true。

- 查询和分析语句

```
* | SELECT every(request_time < 100)
```

- 查询和分析结果

_col0
true

kurtosis函数

kurtosis函数用于计算*x*的峰度。

```
kurtosis(x)
```

参数	说明
x	参数值为double、bigint类型。

double类型。

计算请求时间的峰度。

- 查询和分析语句

```
*| SELECT kurtosis(request_time)
```

- 查询和分析结果

kurtosis
-3.000699825017186

map_union函数

map_union函数用于返回一列Map数据的并集。 如果Map中存在相同的键，则返回的键值为其中任意一个键的值。

```
map_union(x)
```

参数	说明
x	参数值为map类型。

map类型。

将etl_context字段的值（map类型）聚合后，随机返回其中一个值（map类型）。

- 字段样例

```
etl_context: {
 project:"datalab-148****6461-cn-chengdu"
 logstore:"internal-etl-log"
 consumer_group:"etl-83****4d1965"
 consumer:"etl-b2d40ed****c8d6-291294"
 shard_id:"0" }
```

- 查询和分析语句

```
* | SELECT map_union(try_cast(json_parse(etl_context) AS map(varchar,varchar)))
```

- 查询和分析结果

max函数

max函数用于查询*x*中最大的值。

- 查询*x*中最大的值。

```
max(x)
```

- 查询*x*中最大的*n*个值，返回结果为数组。

```
max(x, n)
```

参数	说明
x	参数值为任意数据类型。
n	参数值为正整数。

与参数值的数据类型一致。

- 示例1：查询请求时长的最大值。
  - 查询和分析语句

```
* | SELECT max(request_time) AS max_request_time
```

  - 查询和分析结果

- 示例2：查询请求时长的top 10。
  - 查询和分析语句

```
* | SELECT max(request_time,10) AS "top 10"
```

  - 查询和分析结果

max_by函数

max_by函数支持如下两种用法。

- 查询*y*为最大值时对应的*x*值。

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

- 查询最大的*n*个*y*值对应的*x*值，返回结果为数组。

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

参数	说明
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。

与参数值的数据类型一致。

- 示例1：统计最高消费订单对应的时间点。
  - 查询和分析语句

```
* | SELECT max_by(UsageEndTime, PretaxAmount) as time
```

  - 查询和分析结果

time
1625731025

- 示例2：统计请求时长最大的3个请求对应的请求方法。
  - 查询和分析语句

```
* | SELECT max_by(request_method,request_time,3) AS method
```

  - 查询和分析结果

method
["POST","POST","POST"]

min函数

min函数用于查询*x*中最小值。

- 查询*x*中最小值。

```
min(x)
```

- 查询*x*中最小的*n*个值，返回结果为数组。

```
min(x,n)
```

参数	说明
x	参数值为任意数据类型。
n	参数值为正整数。

与参数值的数据类型一致。

- 示例1：查询请求时长的最小值。

- 查询与分析语句

```
* | SELECT min(request_time) AS min_request_time
```

- 查询和分析结果

min_request_time
10.0

- 示例2：查询请求时长最小的10个值。
  - 查询和分析语句

```
* | SELECT min(request_time,10)
```

  - 查询和分析结果

_col0
[10.0,10.0,10.0,10.0,10.0,11.0,12.0,12.0,13.0,13.0]

min_by函数

min_by函数支持如下两种用法。

- 查询*y*为最小值时对应的*x*值。

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

- 查询最小的*n*个*y*值对应的*x*值。返回结果为数组。

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

参数	说明
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。

与参数值的数据类型一致。

- 示例1：返回最小请求时长的请求对应的请求方法。
  - 查询和分析语句

```
* | SELECT min_by(request_method,request_time) AS method
```

  - 查询和分析结果

method
GET

- 示例2：返回请求时长最小的3个请求对应的请求方法。
  - 查询和分析语句

```
* | SELECT min_by(request_method,request_time,3) AS method
```

- 查询和分析结果

method
["POST","POST","POST"]

skewness函数

skewness函数用于计算*x*的偏度。

```
skewness(x)
```

参数	说明
x	参数值为double、bigint类型。

double类型。

计算请求时间的偏度。

- 查询和分析语句

```
*| SELECT skewness(request_time) AS skewness
```

- 查询和分析结果

skewness
0.0004294441602043965

sum函数

sum函数用于计算*x*的总值。

```
sum(x)
```

参数	说明
x	参数值为double、bigint、decimal或real类型。

与参数值的数据类型一致。

计算网站每天的访问流量。

- 查询和分析语句

```
* | SELECT date_trunc('day',__time__) AS time, sum(body_bytes_sent) AS body_bytes_sent GROUP BY ti
me ORDER BY time
```

- 查询和分析结果

time	body_bytes_sent
2021-07-06 00:00:00.000	3925728

11.1.3. 字符串函数

本文介绍字符串函数的基本语法和示例。

日志服务支持如下字符串函数。

> 注意 在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。

函数名称	语法	说明
chr函数	chr(*x*)	将ASCII码转换为字符。
codepoint函数	codepoint(*x*)	将字符转换为ASCII码。
concat函数	concat(*x*, *y*...)	将多个字符串拼接成一个字符串。
from_utf8函数	from_utf8(*x*)	将二进制字符串解码为UTF-8编码格式，并使用默认字符U+FFFD替换无效的UTF-8字符。
	from_utf8(*x*, *replace_string*)	将二进制字符串解码为UTF-8编码格式，并使用自定义字符串替换无效的UTF-8字符。
length函数	length(*x*)	计算字符串的长度。
levenshtein_distance函数	levenshtein_distance(*x*, *y*)	计算*x*和*y*之间的最小编辑距离。
lower函数	lower(*x*)	将字符串转换为小写形式。
lpad函数	lpad(*x*, *length*, *lpad_string*)	在字符串的开头填充指定字符，直到指定长度后返回结果字符串。
ltrim函数	ltrim(*x*)	删除字符串开头的空格。
normalize函数	normalize(*x*)	使用NFC格式将字符串格式化。
position函数	position(*sub_string* in *x*)	返回目标子串在字符串中的位置。
replace函数	replace(*x*, *sub_string*)	删除字符串中匹配的字符。
	replace(*x*, *sub_string*, *replace_string*)	将字符串中所匹配的字符替换为其他指定字符。
reverse函数	reverse(*x*)	返回反向顺序的字符串。
rpad函数	rpad(*x*, *length*, *rpad_string*)	在字符串的尾部填充指定字符，直到指定长度后返回结果字符串。
rtrim函数	rtrim(*x*)	删除字符串中结尾的空格。
split函数	split(*x*, *delimeter*)	使用指定的分隔符拆分字符串，并返回子串集合。
	split(*x*, *delimeter*, *limit*)	通过指定的分隔符拆分字符串并使用*limit*限制字符串拆分的个数，然后返回拆分后的子串集合。
split_part函数	split_part(*x*, *delimeter*, *part*)	使用指定的分隔符拆分字符串，并返回指定位置的内容。
split_to_map函数	split_to_map(*x*, *delimiter01*, *delimiter02*)	使用指定的第一个分隔符拆分字符串，然后再使用指定的第二个分隔符进行第二次拆分。

函数名称	语法	说明
strpos函数	strpos(*x*, *sub_string*)	返回目标子串在字符串中的位置。与position(*sub_string* in *x*)函数等价。
substr函数	substr(*x*, *start*)	返回字符串中指定位置的子串。
	substr(*x*, *start*, *length*)	返回字符串中指定位置的子串，并指定子串长度。
to_utf8函数	to_utf8(*x*)	将字符串转换为UTF-8编码格式。
trim函数	trim(*x*)	删除字符串中开头和结尾的空格。
upper函数	upper(*x*)	将字符串转化为大写形式。

chr函数

CHR函数用于将ASCII码转换为字符。

```
chr(x)
```

参数	说明
x	ASCII码。

varchar类型。

判断region字段值的首字母是否是c开头，其中99为ASCII码，代表小写字母c。

- 字段样例

  ```
  region:cn-shanghai
  ```

- 查询和分析语句

  ```
  * | SELECT substr(region, 1, 1)=chr(99)
  ```

- 查询和分析结果

_col0
true

codepoint函数

codepoint函数用于将字符转换为ASCII码。

```
codepoint(x)
```

参数	说明
x	参数值为varchar类型。

integer类型。

判断region字段值的首字母是否是c开头，其中99为ASCII码，代表小写字母c。

- 字段样例

```
upstream_status:200
```

- 查询和分析语句

```
* | SELECT codepoint(cast (substr(region, 1, 1) AS char(1))) =99
```

- 查询和分析结果

_col0
true
true

concat函数

concat函数用于将多个字符串拼接成一个字符串。

```
concat(x, y...)
```

参数	说明
x	参数值为varchar类型。
y	参数值为varchar类型。

varchar类型。

将region字段和request_method字段的值拼接为一个字符串。

- 字段样例

```
region:cn-shanghai
time_local:14/Jul/2021:02:19:40
```

- 查询和分析语句

```
* | SELECT concat(region,'-',time_local)
```

- 查询和分析结果

_col0
cn-shanghai-14/Jul/2021:01:16:30
cn-shanghai-14/Jul/2021:01:16:30

from_utf8函数

from_utf8函数用于将二进制字符串解码为UTF-8编码格式。

- 使用默认字符U+FFFD替换无效的UTF-8字符。

```
from_utf8(x)
```

- 使用自定义字符替换无效的UTF-8字符。

```
from_utf8(x,replace_string)
```

参数	说明
x	参数值为binary类型。
replace_string	用于替换的字符串。只能为单个字符或空格。

varchar类型。

- 将二进制字符串0x80解码为UTF-8编码格式，并使用默认字符U+FFFD替换返回结果中无效的UTF-8字符。U+FFFD显示形式为�。
  - 查询和分析语句

    ```
    * | SELECT from_utf8(from_base64('0x80'))
    ```

  - 查询和分析结果

_col0
□4

- 将二进制字符串0x80解码为UTF-8编码格式，并使用0替换返回结果中无效的UTF-8字符。
  - 查询和分析语句

    ```
    * | SELECT from_utf8(from_base64('0x80'),'0')
    ```

  - 查询和分析结果

_col0
0□4

length函数

length函数用于计算字符串的长度。

```
length(x)
```

参数	说明
x	参数值为varchar类型。

bigint类型。

计算http_user_agent字段值的长度。

- 字段样例

  ```
  http_user_agent:Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.2 (KHTML, like Gecko) Chrome/22.0.1216.0 Safari/537.2
  ```

- 查询和分析语句

  ```
  * | SELECT length(http_user_agent)
  ```

- 查询和分析结果

_col0
127

levenshtein_distance函数

levenshtein_distance函数用于计算两个字符串的最小编辑距离。

```
levenshtein_distance(x, y)
```

参数	说明
x	参数值为varchar类型。
y	参数值为varchar类型。

bigint类型。

查询instance_id字段值和owner_id字段值的最小编辑距离。

- 字段样例

```
instance_id:i-01
owner_id:owner-01
```

- 查询和分析语句

```
* | SELECT levenshtein_distance(owner_id,instance_id)
```

- 查询和分析结果

_col0
5

lower函数

lower函数用于将字符串转换为小写形式。

```
lower(x)
```

参数	说明
x	参数值为varchar类型。

varchar类型。

将request_method字段的值转换为小写形式。

- 字段样例

```
request_method:GET
```

- 查询和分析语句

```
* | SELECT lower(request_method)
```

- 查询和分析结果

_col0
get

lpad函数

lpad函数用于在目标字符串的开头填充指定的字符，直到指定长度后返回结果字符串。

```
lpad(x, length, lpad_string)
```

参数	说明
x	参数值为varchar类型。
length	整数，用于指定结果字符串的长度。 • 当字符串的长度小于*length*时，在字符串的开头填充指定的字符。 • 当字符串的长度大于*length*时，只返回字符串中的*length*个字符。
lpad_string	新填充的字符。

varchar类型。

将instance_id字段值的长度补充到10位，不足10位时，在字段值的开头补充0。

- 字段样例

```
instance_id:i-01
```

- 查询和分析语句

```
* | SELECT lpad(instance_id,10,'0')
```

- 查询和分析结果

_col0
000000i-01

ltrim函数

ltrim函数用于删除字符串中开头的空格。

```
ltrim(x)
```

参数	说明
x	参数值为varchar类型。

varchar类型。

删除region字段值开头的空格。

- 字段样例

```
region: cn-shanghai
```

- 查询和分析语句

```
* | SELECT ltrim(region)
```

- 查询和分析结果

_col0
cn-shanghai

normalize函数

normalize函数使用NFC格式将字符串格式化。

```
normalize(x)
```

参数	说明
x	参数值为varchar类型。

varchar类型。

使用NFC格式将字符串schön格式化。

- 查询和分析语句

```
* | SELECT normalize('schön')
```

- 查询和分析结果

position函数

position函数用于查询目标子串在字符串中的位置。

```
position(sub_string in x)
```

参数	说明
sub_string	目标子串。
x	参数值为varchar类型。

int类型，从1开始。

查询子串cn在region字段值中位置。

- 字段样例

```
region:cn-shanghai
```

- 查询和分析语句

```
* | SELECT position('cn' in region)
```

- 查询和分析结果

replace函数

replace函数用于删除字符串中所匹配的字符或者将字符串中所匹配的字符替换为其他指定字符。

- 删除字符串中所匹配的字符。

```
replace(x, sub_string)
```

- 将字符串中所匹配的字符替换为其他指定字符。

```
replace(x, sub_string, replace_string)
```

参数	说明
x	参数值为varchar类型。
sub_string	目标子串。
replace_string	用于替换的子串。

varchar类型。

- 示例1：将region字段值中的cn替换为中国。
  - 字段示例

    ```
    region:cn-shanghai
    ```

  - 查询和分析语句

    ```
    * | select replace(region,'cn','中国')
    ```

  - 查询和分析结果

- 示例2：删除region字段值中的cn-。
  - 字段示例

    ```
    region:cn-shanghai
    ```

  - 查询和分析语句

    ```
    * | select replace(region,'cn-')
    ```

  - 查询和分析结果

reverse函数

reverse函数用于返回反向顺序的字符串。

```
reverse(x)
```

参数	说明
x	参数值为varchar类型。

varchar类型。

将request_method字段值反向排序。

- 字段样例

```
request_method:GET
```

- 查询和分析语句

```
* | SELECT reverse(request_method)
```

- 查询和分析结果

rpad函数

rpad函数用于在字符串的尾部填充指定的字符，直到指定长度后返回结果字符串。

```
rpad(x, length, rpad_string)
```

参数	说明
x	参数值为varchar类型。
length	整数，用于指定结果字符串的长度。 • 当字符串的长度小于*length*时，在字符串的尾部填充指定的字符。 • 当字符串的长度大于*length*时，只返回字符串中的*length*个字符。
lpad_string	新填充的字符。

varchar类型。

将instance_id字段值的长度补充到10位，不足10位时，在字段值的尾部补充0。

- 字段样例

```
instance_id:i-01
```

- 查询和分析语句

```
* | SELECT rpad(instance_id,10,'0')
```

- 查询和分析结果

rtrim函数

rtrim函数用于删除字符串中结尾的空格。

```
rtrim(x)
```

参数	说明
x	参数值为varchar类型。

varchar类型。

删除instance_id字段值中结尾的空格。

- 字段样例

```
instance_id:i-01
```

- 查询和分析语句

```
* | SELECT rtrim(instance_id)
```

- 查询和分析结果

_col0
i-01

split函数

split函数用于通过指定的分隔符拆分字符串，并返回拆分后的子串集合。

- 通过指定的分隔符拆分字符串，并返回拆分后的子串集合。

```
split(x, delimeter)
```

- 通过指定的分隔符拆分字符串并使用*limit*限制字符串拆分的个数，然后返回拆分后的子串集合。

```
split(x,delimeter,limit)
```

参数	说明
x	参数值为varchar类型。
delimeter	分隔符。
limit	限制字符串拆分的个数，大于0的整数。

array类型。

- 示例1：使用正斜线（/）将request_uri字段的值拆分成4个子串，并返回子串的集合。
  - 字段样例

```
request_uri:/request/path-1/file-9
```

  - 查询和分析语句

```
* | SELECT split(request_uri,'/')
```

  - 查询和分析结果

_col0
["","request","path-2","file-2"]

- 示例2：使用正斜线（/）将request_uri字段的值拆分成3个子串，并返回子串的集合。

- 字段样例

```
request_uri:/request/path-1/file-9
```

- 查询和分析语句

```
* | SELECT split(request_uri,'/',3)
```

- 查询和分析结果

_col0
["","request","path-1/file-9"]

split_part函数

split_part函数通过指定的分隔符拆分字符串，并返回指定位置的内容。

```
split_part(x, delimeter, part)
```

参数	说明
x	参数值为varchar类型。
delimeter	分隔符。
part	大于0的整数。

varchar类型。

使用英文问号（?）拆分request_uri字段的值并返回第一个子串（即文件路径部分），然后统计不同路径对应的请求数量。

- 查询和分析语句

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

- 查询和分析结果

PV	Path
4355	/request/path-1/file-5
4328	/request/path-1/file-1
4296	/request/path-2/file-3

split_to_map函数

split_to_map函数用于使用指定的第一个分隔符拆分字符串，然后再使用指定的第二个分隔符进行第二次拆分。

```
split_to_map(x, delimiter01, delimiter02)
```

参数	说明
x	参数值为varchar类型。

参数	说明
delimeter01	分隔符。
delimeter02	分隔符。

map类型。

使用英文逗号（,）和英文冒号（:）拆分time字段的值，返回结果为MAP类型。

- 字段样例

```
time:upstream_response_time:"80", request_time:"40"
```

- 查询和分析语句

```
* | SELECT split_to_map(time,',',':')
```

- 查询和分析结果

_col0
{"request_time":"\"40\"","upstream_response_time":"\"80\""}
{"request_time":"\"40\"","upstream_response_time":"\"80\""}

strpos函数

strpos函数用于返回目标子串在字符串中的位置。与position函数等价。

```
strpos(x, sub_string)
```

参数	说明
x	参数值为varchar类型。
sub_string	目标子串。

int类型，从1开始。

返回字母H在server_protocol字段值中的位置。

- 查询和分析语句

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

- 查询和分析结果

_col0
1
1

substr函数

substr函数用于返回字符串中指定位置的子串。

- 返回字符串中指定位置的子串。

```
substr(x, start)
```

- 返回字符串中指定位置的子串，并指定子串长度。

```
substr(x,start,length)
```

参数	说明
x	参数值为varchar类型。
start	开始提取子串的位置，从1开始。
length	子串的长度。

varchar类型。

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

- 字段样例

```
server_protocol:HTTP/2.0
```

- 查询和分析语句

```
* | SELECT substr(server_protocol,1,4) AS protocol, count(*) AS count GROUP BY server_protocol
```

- 查询和分析结果

protocol	count
HTTP	9078

to_utf8函数

to_utf8函数用于将字符串转换为UTF-8编码格式。

```
to_utf8(x)
```

参数	说明
x	参数值为varchar类型。

varbinary类型。

将字符串log转换为UTF-8编码格式。

- 查询和分析语句

```
* | SELECT to_utf8('log')
```

- 查询和分析结果

_col0
bG9n

trim函数

trim函数用于删除字符串中开头和结尾的空格。

```
trim(x)
```

参数	说明
x	参数值为varchar类型。

varchar类型。

删除instance_id字段值的开头和结尾的空格。

- 字段样例

```
instance_id: i-01
```

- 查询和分析语句

```
* | SELECT trim(instance_id)
```

- 查询和分析结果

upper函数

upper函数用于将目标字符串转化为大写形式。

```
upper(x)
```

参数	说明
x	参数值为varchar类型。

varchar类型。

将region字段值转换为大写形式。

- 字段样例

```
region:cn-shanghai
```

- 查询和分析语句

```
* | SELECT upper(region)
```

- 查询和分析结果

11.1.4. 日期和时间函数

日志服务提供时间函数、日期函数、日期和时间提取函数、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。本文介绍日期和时间函数的基本语法及示例。

日志服务支持如下日期和时间函数。

注意

- 日志服务中的日志时间戳精确到秒，所以配置时间格式（format）时，只需配置到秒，无需配置毫秒、微秒等信息。
- 只需为时间字符串中的时间部分配置时间格式（format），其他内容（例如时区）无需配置时间格式。
- 日志服务中的每条日志都包含保留字段__time__，该字段的值为UNIX时间戳格式，例如1592374067，代表2020-06-17 14:07:47。
- 在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。

函数类型	函数名称	语法	说明
日期和时间函数	current_date函数	current_date	返回当前日期。
	current_time函数	current_time	返回当前时间和时区。
	current_timestamp函数	current_timestamp	返回当前日期、时间和时区。
	current_timezone函数	current_timezone()	返回当前时区。
	date函数	date(*x*)	返回日期和时间表达式中的日期部分。
	date_format函数	date_format(*x*, *format*)	将timestamp类型的日期和时间表达式转化为指定格式的日期和时间表达式。
	date_parse函数	date_parse(*x*, *format*)	将日期和时间字符串转换为指定格式的timestamp类型的日期和时间表达式。
	from_iso8601_date函数	from_iso8601_date(*x*)	将ISO8601格式的日期表达式转化为date类型的日期表达式。
	from_iso8601_timestamp函数	from_iso8601_timestamp(*x*)	将ISO8601格式的日期和时间表达式转化为timestamp类型的日期和时间表达式。
	from_unixtime函数	from_unixtime(*x*)	将UNIX时间戳转化为无时区的timestamp类型的日期和时间表达式。
		from_unixtime(*x*, *time zone*)	将UNIX时间戳转化为带时区的timestamp类型的日期和时间表达式。
		from_unixtime(*x*, *hours*, *minutes*)	将UNIX时间戳转化为带时区的timestamp类型的日期和时间表达式，其中*hours*和*minutes*为时区偏移量。
	localtime函数	localtime	返回本地时间。
	localtimestamp函数	localtimestamp	返回本地日期和时间。

函数类型	函数名称	语法	说明
	now函数	now()	返回当前日期和时间。 now函数等同于current_timestamp函数。
	to_iso8601函数	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。
	to_unixtime函数	to_unixtime(*x*)	将timestamp类型的日期和时间表达式转化成UNIX时间戳。
日期和时间提取函数	day函数	day(*x*)	提取日期和时间表达式中的天数，按月计算。 day函数等同于day_of_month函数。
	day_of_month函数	day_of_month(*x*)	提取日期和时间表达式中的天数，按月计算。 day_of_month函数等同于day函数。
	day_of_week函数	day_of_week(*x*)	提取日期和时间表达式中的天数，按周计算。 day_of_week函数等同于dow函数。
	day_of_year函数	day_of_year(*x*)	提取日期和时间表达式中的天数，按年计算。 day_of_year函数等同于doy函数。
	dow函数	dow(*x*)	提取日期和时间表达式中的天数，按周计算。 dow函数等同于day_of_week函数。
	doy函数	doy(*x*)	提取日期和时间表达式中的天数，按年计算。 doy函数等同于day_of_year函数。
	extract函数	extract(*field* from *x*)	通过指定的*field*，提取日期和时间表达式中的日期或时间部分。
	hour函数	hour(*x*)	提取日期和时间表达式中的小时数，按24小时制计算。
	minute函数	minute(*x*)	提取日期和时间表达式中的分钟数。
	month函数	month(*x*)	提取日期和时间表达式中的月份。
	quarter函数	quarter(*x*)	计算目标日期所属的季度。

函数类型	函数名称	语法	说明
	second函数	second(*x*)	提取日期和时间表达式中的秒数。
	timezone_hour函数	timezone_hour(*x*)	计算时区的小时偏移量。
	timezone_minute函数	timezone_minute(*x*)	计算时区的分钟偏移量。
	week函数	week(*x*)	计算目标日期是在一年中的第几周。 week函数等同于week_of_year函数。
	week_of_year函数	week_of_year(*x*)	计算目标日期是在一年中的第几周。 week_of_year函数等同于week函数。
	year函数	year(*x*)	提取目标日期中的年份。
	year_of_week函数	year_of_week(*x*)	提取目标日期在ISO周日历中的年份。 year_of_week函数等同于yow函数。
	yow函数	yow(*x*)	提取目标日期在ISO周日历中的年份。 yow函数等同于year_of_week函数。
时间间隔函数	date_trunc函数	date_trunc(unit, *x*)	根据您指定的时间单位截断日期和时间表达式，并按照毫秒、秒、分钟，小时、日、月或年对齐。
	date_add函数	date_add(*unit*, *N*, *x*)	在*x*上加上N个时间单位（unit）。
	date_diff函数	date_diff(*unit*, *x*, *y*)	返回两个时间表达式之间的时间差值，例如计算*x*和*y*之间相差几个时间单位（unit）。
时序补全函数	time_series函数	time_series(*x*, *window*, *format*, *padding_data*)	补全您查询时间窗口内缺失的数据。

current_date函数

current_date函数用于返回当前日期，格式为YYYY-MM-DD。

```
current_date
```

date类型。

查询昨天的日志。

- 查询和分析语句

```
* |
SELECT
  *
FROM  log
WHERE
  __time__ < to_unixtime(current_date)
  AND __time__ > to_unixtime(date_add('day', -1, current_date))
```

- 查询和分析结果

body_bytes_sent	client_ip	host	http_user_agent	http_x_forwarded_for	instance_id	instance_name	network_type	owner_id	referer
4650	42.	www com	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) ... 展开	6	i-01	instance-01	vlan	owner-01	ww k.com
1492	42. 3	www k.com	Mozilla/5.0 (iPhone; U; ru; CPU iPhone OS 4_2_1... 展开	1	i-02	instance-01	vlan	owner-01	ww :k.cor
3148	59.	www k.com	Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_3_1 lik... 展开	1	i-02	instance-02	vlan	owner-01	ww k.cor

current_time函数

current_time函数用于返回当前时间和时区，格式为HH:MM:SS.Ms Time_zone。

```
current_time
```

time类型。

查询当前时间和时区。

- 查询和分析语句

```
* | select current_time
```

- 查询和分析结果

_col0
22:34:24.034 Asia/Shanghai
22:34:24.034 Asia/Shanghai

current_timestamp函数

current_timestamp函数用于返回当前日期、时间和时区，格式为YYYY-MM-DD HH:MM:SS.Ms Time_zone。

```
current_timestamp
```

timestamp类型。

查询昨天的日志。

- 查询和分析语句

```
* |
SELECT
  *
FROM  log
WHERE
  __time__ < to_unixtime(current_timestamp)
  AND __time__ > to_unixtime(date_add('day', -1, current_timestamp))
```

- 查询和分析结果

body_bytes_sent	client_ip	host	http_user_agent	http_x_forwarded_for	instance_id	instance_name	network_type	owner_id	referer
4650	42.	www c om	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) ... 展开	6	i-01	instance-01	vlan	owner-01	ww k.com
1492	42. 3	www k. com	Mozilla/5.0 (iPhone; U; ru; CPU iPhone OS 4_2_1... 展开	1	i-02	instance-01	vlan	owner-01	ww k. cor
3148	59.	www k. com	Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_3_1 lik... 展开	1	i-02	instance-02	vlan	owner-01	ww k. cor

current_timezone函数

current_timezone函数用于返回当前时区。

```
current_timezone()
```

varchar类型。

查询当前的时区。

- 查询和分析语句

```
* | select current_timezone()
```

- 查询和分析结果

_col0
Asia/Shanghai
Asia/Shanghai

date函数

date函数用于提取日期和时间表达式中的日期部分。date函数等同于 `cast(x as date)` 。更多信息，请参见类型转换函数。

```
date(x)
```

参数	说明
x	参数值为date、timestamp类型。

date类型。

使用current_timestamp函数获取当前日期和时间，然后使用date函数提取日期部分。

- 查询和分析语句

```
* | SELECT current_timestamp, date(current_timestamp)
```

- 查询和分析结果

_col0	_col1
2021-09-07 15:00:52.506 Asia/Shanghai	2021-09-07

date_format函数

date_format函数用于将timestamp类型的日期和时间表达式转换为指定格式的日期和时间表达式。

```
date_format(x, format)
```

参数	说明
x	参数值为timestamp类型的日期和时间表达式。
format	日期和时间表达式的转换格式。更多信息，请参见format说明。

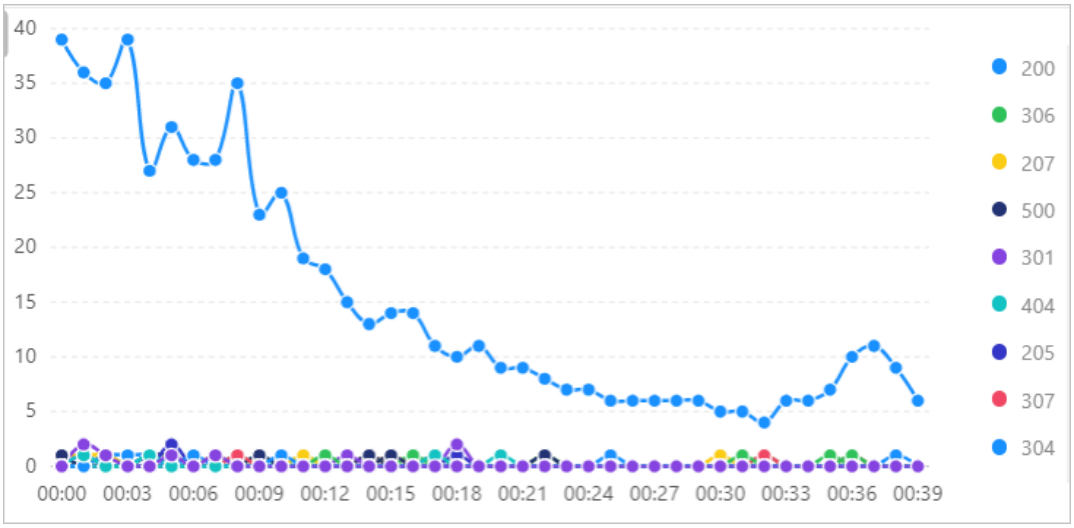
varchar类型。

计算Nginx请求状态及数量，并根据时间顺序展示。首先通过date_trunc函数将日志时间按照分钟对齐，再使用date_format函数转换为 `%H:%i` 格式，然后计算每分钟内每个状态码对应的请求数量，并以流图展示查询和分析结果。

- 查询和分析语句

```
* |
SELECT
  date_format(date_trunc('minute', __time__), '%H:%i') AS time,
  COUNT(1) AS count,
  status
GROUP BY
  time,
  status
ORDER BY
  time
```

- 查询和分析结果



date_parse函数

date_parse函数用于将日期和时间字符串转换为指定格式的timestamp类型的日期和时间表达式。

```
date_parse(x, format)
```

参数	说明
x	参数值为日期和时间字符串。
format	日期和时间表达式的转换格式。更多信息，请参见format说明。

timestamp类型。

将StartTime字段值和EndTime字段值转换为timestamp类型，并计算两者的时间差。

- 查询和分析语句

```
*|
SELECT
    date_parse(StartTime, '%Y-%m-%d %H:%i') AS "StartTime",
    date_parse(EndTime, '%Y-%m-%d %H:%i') AS "EndTime",
    date_diff('hour', StartTime, EndTime) AS "时间差(小时)"
```

- 查询和分析结果

StartTime	EndTime	时间差(小时)
2021-07-21 20:00:00.000	2021-07-21 21:00:00.000	1

from_iso8601_date函数

from_iso8601_date函数用于将ISO8601格式的日期表达式转化为date类型的日期表达式，格式为YYYY-MM-DD。

```
from_iso8601_date(x)
```

参数	说明
x	参数值为ISO8601格式的日期表达式。

date类型。

将time字段的值转换为date类型的日期表达式。

- 字段样例

```
time:2020-05-03
```

- 查询和分析语句

```
* | select from_iso8601_date(time)
```

- 查询和分析结果

from_iso8601_timestamp函数

from_iso8601_timestamp函数用于将ISO8601格式的日期和时间表达式转化为timestamp类型的日期和时间表达式，格式为YYYY-MM-DD HH:MM:SS.Ms Time_zone。

```
from_iso8601_timestamp(x)
```

参数	说明
x	参数值为ISO8601格式的日期和时间表达式。

timestamp类型。

将time字段的值转换为timestamp类型的日期和时间表达式。

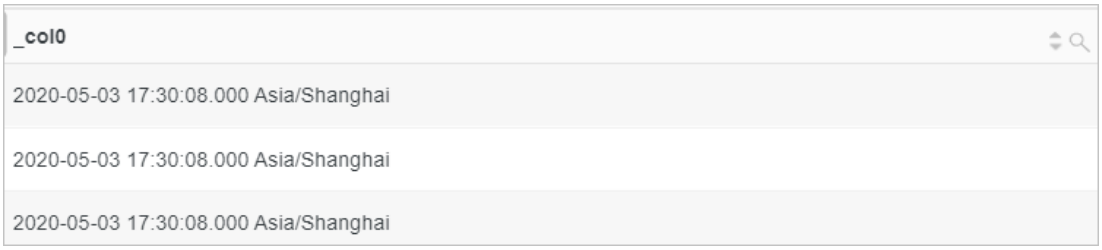
- 字段样例

  ```
  time:2020-05-03T17:30:08
  ```

- 查询和分析语句

  ```
  * | select from_iso8601_timestamp(time)
  ```

- 查询和分析结果



from_unixtime函数

from_unixtime函数用于将UNIX时间戳转化为timestamp类型的日期和时间表达式，格式为YYYY-MM-DD HH:MM:SS.Ms或YYYY-MM-DD HH:MM:SS.Ms Time_zone。

- 转化为无时区的timestamp类型的日期和时间表达式

  ```
  from_unixtime(x)
  ```

- 转化为带时区的timestamp类型的日期和时间表达式

  ```
  from_unixtime(x,time zone)
  ```

- 转化为带时区的timestamp类型的日期和时间表达式，其中*hours*和*minutes*为时区偏移量。

  ```
  from_unixtime(x, hours, minutes)
  ```

参数	说明
x	参数值为UNIX时间戳。
time zone	时区，例如Asia/shanghai。
hours	时区的小时偏移量，例如+07、-09
minutes	时区的分钟偏移量，例如+30、-45。

timestamp类型。

将time字段的值转化为带时区的timestamp类型的日期和时间表达式。

- 字段样例

```
time:1626774758
```

- 查询和分析语句

```
* | select from_unixtime(time,'Asia/shanghai')
```

- 查询和分析结果

_col0
2021-07-20 17:52:38.000 Asia/Shanghai
2021-07-20 17:52:38.000 Asia/Shanghai

localtime函数

localtime函数用于返回本地时间，格式为HH:MM:SS.Ms。

```
localtime
```

time类型。

查询本地时间。

- 查询和分析语句

```
* | select localtime
```

- 查询和分析结果

_col0
02:09:46.213
02:09:46.213

localtimestamp函数

localtimestamp函数用于返回本地的日期和时间，格式为YYYY-MM-DD HH:MM:SS.Ms Time_zone。

```
localtimestamp
```

timestamp类型。

查询昨天的日志。

- 查询和分析语句

```
* |
SELECT
  *
FROM  log
WHERE
  __time__ < to_unixtime(localtimestamp)
  AND __time__ > to_unixtime(date_add('day', -1, localtimestamp))
```

- 查询和分析结果

body_bytes_sent	client_ip	host	http_user_agent	http_x_forwarded_for	instance_id	instance_name	network_type	owner_id	referer
4650	42.[redacted]0	www[redacted]com	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) ... 展开	6[redacted])	i-01	instance-01	vlan	owner-01	ww[redacted]k.com
1492	42.[redacted]3	www[redacted]k.com	Mozilla/5.0 (iPhone; U; ru; CPU iPhone OS 4_2_1... 展开	1[redacted]	i-02	instance-01	vlan	owner-01	ww[redacted]:k.cor[redacted]
3148	59.[redacted]	www[redacted]k.com	Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_3_1 like... 展开	1[redacted]5	i-02	instance-02	vlan	owner-01	ww[redacted]:k.cor[redacted]

now函数

now函数用于返回当前日期和时间，格式为YYYY-MM-DD HH:MM:SS.Ms Time_zone。now函数等同于current_timestamp函数。

```
now()
```

timestamp类型。

查询昨天的日志。

- 查询和分析语句

```
* |
SELECT
  *
FROM  log
WHERE
  __time__ < to_unixtime(now())
  AND __time__ > to_unixtime(date_add('day', -1, now()))
```

- 查询和分析结果

body_bytes_sent	client_ip	host	http_user_agent	http_x_forwarded_for	instance_id	instance_name	network_type	owner_id	referer
4650	42. 0	www com	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) ... 展开	6)	i-01	instance-01	vlan	owner-01	ww k.com
1492	42. 3	www k.com	Mozilla/5.0 (iPhone; U; ru; CPU iPhone OS 4_2_1... 展开	1	i-02	instance-01	vlan	owner-01	ww :k.cor
3148	59.	www k.com	Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_3_1 lik... 展开	1 ;	i-02	instance-02	vlan	owner-01	ww :k.cor

to_iso8601函数

to_iso8601函数用于将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。

```
to_iso8601(x)
```

参数	说明
x	参数值为date、timestamp类型。

varchar类型。

使用current_timestamp函数获取当前日期和时间，然后使用to_iso8601函数将当前的日期和时间表达式转换为ISO8601格式的日期和时间表达式。

- 查询和分析语句

```
* | select to_iso8601(current_timestamp) AS ISO8601
```

- 查询和分析结果

ISO8601
2021-09-02T16:57:56.964+08:00

to_unixtime函数

to_unixtime函数用于将timestamp类型的日期和时间表达式转化成UNIX时间戳。

```
to_unixtime(x)
```

参数	说明
x	参数值为timestamp类型的日期和时间表达式。

double类型。

查询昨天的日志。

- 查询和分析语句

```
* |
SELECT
  *
FROM  log
WHERE
  __time__ < to_unixtime(now())
  AND __time__ > to_unixtime(date_add('day', -1, now()))
```

- 查询和分析结果

body_bytes_sent	client_ip	host	http_user_agent	http_x_forwarded_for	instance_id	instance_name	network_type	owner_id	referer
4650	42.	www.com	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) ... 展开	6	i-01	instance-01	vlan	owner-01	www.com
1492	42.3	www.com	Mozilla/5.0 (iPhone; U; ru; CPU iPhone OS 4_2_1... 展开	1	i-02	instance-01	vlan	owner-01	www.com
3148	59.	www.com	Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_3_1 lik... 展开	1	i-02	instance-02	vlan	owner-01	www.com

day函数

day函数用于提取日期和时间表达式中的天数，按月计算。day函数等同于day_of_month函数。

```
day(x)
```

参数	说明
x	参数值为timestamp、date类型。

bigint类型。

使用current_date函数获取当前日期，然后使用day函数提取当前日期为本月的第几天。

- 查询和分析语句

```
* | SELECT current_date, day(current_date)
```

- 查询和分析结果

_col0	_col1
2021-09-07	7

day_of_month函数

day_of_month函数用于提取日期和时间表达式中的天数，按月计算。day_of_month函数等同于day函数。

```
day_of_month(x)
```

参数	说明
x	参数值为timestamp、date类型。

bigint类型。

使用current_date函数获取当前日期，然后使用day_of_month函数提取当前日期为本月的第几天。

- 查询和分析语句

```
* | SELECT current_date, day_of_month(current_date)
```

- 查询和分析结果

_col0	_col1
2021-09-07	7

day_of_week函数

day_of_week函数用于提取日期和时间表达式中的天数，按周计算。

```
day_of_week(x)
```

参数	说明
x	参数值为timestamp、date类型。

bigint类型。

使用current_date函数获取当前日期，然后使用day_of_week函数提取当前日期为本周的第几天。

- 查询和分析语句

```
* | SELECT current_date, day_of_week(current_date)
```

- 查询和分析结果

_col0	_col1
2021-09-07	2

day_of_year函数

day_of_year函数用于提取日期和时间表达式中的天数，按年计算。

```
day_of_year(x)
```

参数	说明
x	参数值为timestamp、date类型。

bigint类型。

使用current_date函数获取当前日期，然后使用day_of_year函数提取当前日期为本年的第几天。

- 查询和分析语句

```
* | SELECT current_date, day_of_year(current_date)
```

- 查询和分析结果

_col0	_col1
2021-09-07	250

dow函数

dow函数用于提取日期和时间表达式中的天数，按周计算。dow函数等同于day_of_week函数。

```
dow(x)
```

参数	说明
x	参数值为timestamp、date类型。

bigint类型。

使用current_date函数获取当前日期，然后使用dow函数提取当前日期为本周的第几天。

- 查询和分析语句

```
* | SELECT current_date, dow(current_date)
```

- 查询和分析结果

_col0	_col1
2021-09-07	2

doy函数

doy函数用于提取日期和时间表达式中的天数，按年计算。doy函数等同于day_of_year函数。

```
doy(x)
```

参数	说明
x	参数值为timestamp、date类型。

bigint类型。

使用current_date函数获取当前日期，然后使用doy函数提取当前日期为本年的第几天。

- 查询和分析语句

```
* | SELECT current_date, doy(current_date)
```

- 查询和分析结果

_col0	_col1
2021-09-07	250

extract函数

extract函数通过指定的*field*，提取日期和时间表达式中的日期或时间部分。

```
extract(field from x)
```

参数	说明
field	取值为year、quarter、month、week、day、day_of_month、day_of_week、dow、day_of_year、doy、year_of_week、yow、hour、minute、second、timezone_hour、timezone_minute。
x	参数值为date、time、timestamp、interval (actual varchar(9)) 类型。

bigint类型。

使用current_date函数获取当前日期，然后使用extract函数提取当前日期中的年份。

- 查询和分析语句

```
* | SELECT extract(year from current_date)
```

- 查询和分析结果

_col0
2021

hour函数

hour函数用于提取日期和时间表达式中的小时数，按24小时制计算。

```
hour(x)
```

参数	说明
x	参数值为timestamp类型。

bigint类型。

使用current_timestamp函数获取当前日期和时间，然后使用hour函数提取当前时间的小时数。

- 查询和分析语句

```
* | SELECT current_timestamp, hour(current_timestamp)
```

- 查询和分析结果

_col0	_col1
2021-09-07 11:00:48.413 Asia/Shanghai	11

minute函数

minute函数用于提取日期和时间表达式中的分钟数。

```
minute(x)
```

参数	说明
x	参数值为timestamp类型。

bigint类型。

使用current_timestamp函数获取当前日期和时间，然后使用minute函数提取当前时间的分钟数。

- 查询和分析语句

```
* | SELECT current_timestamp, minute(current_timestamp)
```

- 查询和分析结果

_col0	_col1
2021-09-07 11:17:11.676 Asia/Shanghai	17

month函数

month函数用于提取日期和时间表达式中的月份。

```
month(x)
```

参数	说明
x	参数值为date、timestamp类型。

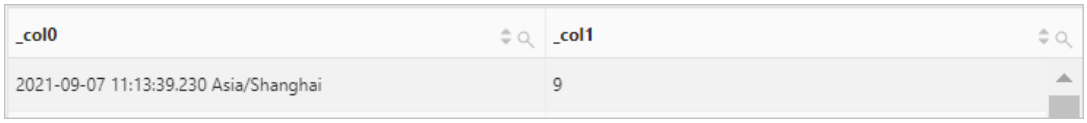
bigint类型。

使用current_timestamp函数获取当前日期和时间，然后使用month函数提取当前日期所属的月份。

- 查询和分析语句

```
* | SELECT current_timestamp, month(current_timestamp)
```

- 查询和分析结果



_col0	_col1
2021-09-07 11:13:39.230 Asia/Shanghai	9

quarter函数

quarter函数用于返回日期所属的季度。

```
quarter(x)
```

参数	说明
x	参数值为date、timestamp类型。

bigint类型。

使用current_timestamp函数获取当前日期和时间，然后使用quarter函数计算当前日期所属的季度。

- 查询和分析语句

```
* | SELECT current_timestamp,quarter(current_timestamp)
```

- 查询和分析结果

_col0	_col1
2021-09-07 11:26:08.720 Asia/Shanghai	3

second函数

second函数用于提取日期和时间表达式中的秒数。

```
second(x)
```

参数	说明
x	参数值为timestamp类型。

bigint类型。

使用current_timestamp函数获取当前日期和时间，然后使用second函数提取当前时间的秒数。

- 查询和分析语句

```
* | SELECT current_timestamp,second(current_timestamp)
```

- 查询和分析结果

_col0	_col1
2021-09-07 11:32:41.695 Asia/Shanghai	41

timezone_hour函数

timezone_hour函数用于计算时区的小时偏移量。

```
timezone_hour(x)
```

参数	说明
x	参数值为timestamp类型。

bigint类型。

使用current_timestamp函数获取当前日期和时间，然后使用timezone_hour函数计算当前时间所属时区的小时偏移量。

- 查询和分析语句

```
* | SELECT current_timestamp, timezone_hour(current_timestamp)
```

- 查询和分析结果

_col0	_col1
2021-09-07 11:35:51.605 Asia/Shanghai	8

timezone_minute函数

timezone_minute函数用于计算时区的分钟偏移量。

```
timezone_minute(x)
```

参数	说明
x	参数值为timestamp类型。

bigint类型。

使用current_timestamp函数获取当前日期和时间，然后使用timezone_minute函数计算当前时间所属时区的分钟偏移量。

- 查询和分析语句

```
* | SELECT current_timestamp,timezone_minute(current_timestamp)
```

- 查询和分析结果

_col0	_col1
2021-09-07 11:41:48.818 Asia/Shanghai	0

week函数

week函数用于计算目标日期是在一年中的第几周。week函数等同于week_of_year函数。

```
week(x)
```

参数	说明
x	参数值为date、timestamp类型。

bigint类型。

使用current_timestamp函数获取当前日期和时间，然后使用week函数计算当前日期是一年中的第几周。

- 查询和分析语句

```
* | SELECT current_timestamp, week(current_timestamp)
```

- 查询和分析结果

_col0	_col1
2021-09-07 13:40:04.940 Asia/Shanghai	36

week_of_year函数

week_of_year函数用于计算目标日期是在一年中的第几周。week_of_year函数等同于week函数。

```
week_of_year(x)
```

参数	说明
x	参数值为date、timestamp类型。

bigint类型。

使用current_timestamp函数获取当前日期和时间，然后使用week_of_year函数计算当前日期是一年中的第几周。

- 查询和分析语句

```
* | SELECT current_timestamp, week_of_year(current_timestamp)
```

- 查询和分析结果

_col0	_col1
2021-09-07 13:40:04.940 Asia/Shanghai	36

year函数

year函数用于提取目标日期中的年份。

```
year(x)
```

参数	说明
x	参数值为date、timestamp类型。

bigint类型。

使用current_timestamp函数获取当前日期和时间，然后使用year函数提取当前日期中的年份。

- 查询和分析语句

```
* | SELECT current_timestamp,year(current_timestamp)
```

- 查询和分析结果

_col0	_col1
2021-09-07 13:49:47.845 Asia/Shanghai	2021

year_of_week函数

year_of_week函数用于返回目标日期在ISO周日历中的年份。year_of_week函数等同于yow函数。

```
year_of_week(x)
```

参数	说明
x	参数值为date、timestamp类型。

bigint类型。

使用current_timestamp函数获取当前日期和时间，然后使用year_of_week函数返回当前日期在ISO周日历中的年份。

- 查询和分析语句

```
* | SELECT current_timestamp,year_of_week(current_timestamp)
```

- 查询和分析结果

_col0	_col1
2021-09-07 13:57:25.576 Asia/Shanghai	2021

yow函数

yow函数用于返回目标日期在ISO周日历中的年份。yow函数等同于year_of_week函数。

```
yow(x)
```

参数	说明
x	参数值为date、timestamp类型。

bigint类型。

使用current_timestamp函数获取当前日期和时间，然后使用yow函数返回当前日期在ISO周日历中的年份。

- 查询和分析语句

```
* | SELECT current_timestamp, yow(current_timestamp)
```

- 查询和分析结果

_col0	_col1
2021-09-07 13:57:25.576 Asia/Shanghai	2021

date_trunc函数

date_trunc函数会根据您指定的时间单位截断日期和时间表达式，并按照毫秒、秒、分钟，小时、日、月或年对齐。该函数常用于需要按照时间进行统计分析的场景。

```
date_trunc(unit, x)
```

参数	说明
unit	时间单位，取值为millisecond、second、minute、hour、day、week、month、quarter、year。更多信息，请参见unit说明。
x	参数值为日期和时间表达式。

> **说明** date_trunc函数只能按照固定的时间间隔统计（例如每分钟、每小时等）。如果您需要按照灵活的时间维度统计，请使用数学取模方法进行分组，例如统计每5分钟的数据。
>
> ```
> * | SELECT count(1) AS pv,  __time__ - __time__ %300 AS time GROUP BY time LIMIT 100
> ```

与参数值的类型一致。

按照每分钟的时间粒度计算请求时间的平均值，并按照时间进行分组和排序。

- 查询和分析语句

```
* |
SELECT
  date_trunc('minute', __time__) AS time,
  truncate (avg(request_time)) AS avg_time,
  current_date AS date
GROUP BY
  time
ORDER BY
  time DESC
LIMIT
  100
```

- 查询和分析结果

time	avg_time	date
2021-07-21 11:26:00.000	47.0	2021-07-21
2021-07-21 11:25:00.000	44.0	2021-07-21
2021-07-21 11:24:00.000	44.0	2021-07-21

date_add函数

date_add函数用于在日期或时间中添加或减去指定的时间间隔。

```
date_add(unit, n, x)
```

参数	说明
unit	时间单位，取值为millisecond、second、minute、hour、day、week、month、quarter、year。更多信息，请参见unit说明。

参数	说明
n	时间间隔。
x	参数值为timestamp类型日期和时间表达式。

timestamp类型。

查询昨天的日志。

- 查询和分析语句

```
* |
SELECT
  *
FROM  log
WHERE
  __time__ < to_unixtime(current_timestamp)
  AND __time__ > to_unixtime(date_add('day', -1, current_timestamp))
```

- 查询和分析结果

body_bytes_sent	client_ip	host	http_user_agent	http_x_forwarded_for	instance_id	instance_name	network_type	owner_id	referer
4650	42.	www com	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) ... 展开	6	i-01	instance-01	vlan	owner-01	ww k.com
1492	42. 3	www k.com	Mozilla/5.0 (iPhone; U; ru; CPU iPhone OS 4_2_1... 展开	1	i-02	instance-01	vlan	owner-01	ww :k.cor
3148	59.	www k.com	Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_3_1 like... 展开	1	i-02	instance-02	vlan	owner-01	ww k.cor

date_diff函数

date_diff函数用于计算两个日期或时间之间的差值。

```
date_diff(unit, x, y)
```

参数	说明
unit	时间单位，取值为millisecond、second、minute、hour、day、week、month、quarter、year。更多信息，请参见unit说明。
x	参数值为timestamp类型日期和时间表达式。
y	参数值为timestamp类型日期和时间表达式。

bigint类型。

通过UsageStartTime字段和UsageEndTime字段计算服务器使用的总时长。

- 查询和分析语句

```
* | SELECT  date_diff('hour', UsageStartTime, UsageEndTime) AS "时间差(小时)"
```

- 查询和分析结果

时间差(小时)
24

time_series函数

time_series函数用于补全您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。

```
time_series(x, window_time, format, padding_data)
```

参数	说明
x	时间列，例如__time__。时间列的值为long类型或timestamp类型。
window_time	窗口大小，单位为s（秒）、m（分）、h（小时）、d（天）。例如2h、5m、3d。
format	返回结果的时间格式。更多信息，请参见format说明。
padding_data	补全的内容。包括： • 0：将缺失的值设置为0。 • null：将缺失的值设置为null。 • last：将缺失的值设置了上一个时间点对应的值。 • next：将缺失的值设置了下一个时间点对应的值。 • avg：将缺失的值设置为前后两个时间点的平均值。

varchar类型。

按照两个小时的时间粒度进行数据补全，将缺失的值设置为0。

- 查询和分析语句

```
* | select time_series(__time__, '2h', '%Y-%m-%d %H:%i:%s', '0')  as time, count(*) as num from log group by time order by time
```

- 查询和分析结果

time	num
2021-07-20 00:00:00	11602
2021-07-20 02:00:00	63089
2021-07-20 04:00:00	36583
2021-07-20 06:00:00	11135
2021-07-20 08:00:00	62746
2021-07-20 10:00:00	18314

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。需加上后缀，例如0th、1st、2nd、3rd。
%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。
%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。数值格式，取值范围为01~12。
%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，星期日是一周的第一天。取值范围为01~53。
%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。
%w	一周的第几天，星期日为第0天。
%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。

unit说明

unit	说明
millisecond	毫秒
second	秒
minute	分钟
hour	小时
day	天
week	周
month	月
quarter	季度
year	年

11.1.5. JSON函数

本文介绍JSON函数的基本语法及示例。

日志服务支持如下JSON函数。

> 注意
>
> - 在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。
> - 如果日志字段的值为JSON类型且需要展开为多行，请使用unnest语法。更多信息，请参见UNNEST子句。
> - 如果字符串被解析成JSON类型失败，则返回null。
> - 如果在采集过程中，JSON日志被截断，则在使用JSON函数进行查询与分析时，系统将报错且中止查询与分析。针对该错误，您可以使用TRY表达式捕获异常信息，使得系统继续执行查询和分析操作。例如 `* | select message, try(json_parse(message))` 。更多信息，请参见TRY表达式。

函数名称	语法	说明
json_array_contains函数	json_array_contains(*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。

```
json_array_contains(x, value)
```

参数	说明
x	参数值为JSON数组。
value	数值。

boolean类型。

判断JSON字符串[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数组下标对应的元素。

```
json_array_get(x, index)
```

参数	说明
x	参数值为JSON数组。
index	JSON下标，从0开始。

varchar类型。

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。

```
json_array_length(x)
```

参数	说明
x	参数值为JSON数组。

bigint类型。

- 示例1：计算Results字段值中JSON元素的数量。
  - 字段样例

    ```
    Results:[{"EndTime":1626314920},{"FireResult":2}]
    ```

  - 查询和分析语句

    ```
    * | SELECT json_array_length(Results)
    ```

  - 查询和分析结果

- 示例2：计算time字段值中JSON元素的数量。
  - 字段样例

    ```
    time:["time_local","request_time","upstream_response_time"]
    ```

  - 查询和分析语句

    ```
    * | SELECT json_array_length(time)
    ```

  - 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）。

> 注意 针对非法的JSON类型，json_extract函数会报错，建议您使用json_extract_scalar函数。

```
json_extract(x, json_path)
```

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。

JSON格式的string类型。

获取Results字段中EndTime字段的值。

- 字段样例

  ```
  Results:[{"EndTime":1626314920},{"FireResult":2}]
  ```

- 查询和分析语句

```
* | SELECT json_extract(Results, '$.0.EndTime')
```

- 查询和分析结果

_col0
1626328420

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）。

```
json_extract_scalar(x, json_path)
```

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。

varchar类型。

从Results字段中获取RawResultCount字段的值，并将该值转换为bigint类型进行求和。

- 字段样例

```
Results:[{"EndTime":1626314920},{"RawResultCount":1}]
```

- 查询和分析语句

```
* | SELECT sum(cast(json_extract_scalar(Results,'$.0.RawResultCount') AS bigint) )
```

- 查询和分析结果

_col0
288

json_format函数

json_format函数用于将JSON类型转化成字符串类型。

```
json_format(x)
```

参数	说明
x	参数值为JSON类型。

varchar类型。

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。一般情况下，json_parse函数使用意义不大，如果您需要从JSON中提取值，建议使用json_extract_scalar函数。

```
json_parse(x)
```

参数	说明
x	参数值为字符串。

JSON类型。

将字符串[1,2,3]转换为JSON数组[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。

```
json_size(x, json_path)
```

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。

bigint类型。

返回status字段中元素的数量。

- 字段样例

```
Results:[{"EndTime":1626314920,"FireResult":2,"RawResults":[{"_col0":"1094"}]}]
```

- 查询和分析语句

```
* | SELECT json_size(Results, '$.0.RawResults')
```

- 查询和分析结果

11.1.6. 正则式函数

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。

> 注意 在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，然后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。

- 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
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。

array类型。

- 示例1：提取server_protocol字段值中所有的数字。
  - 字段样例

    ```
    server_protocol:HTTP/2.0
    ```

  - 查询和分析语句

    ```
    *| SELECT regexp_extract_all(server_protocol, '\d+')
    ```

  - 查询和分析结果

_col0
["2","0"]

- 示例2：提取http_user_agent字段值中的Chrome部分，然后统计由Chrome浏览器发起的请求数量。
  - 字段样例

    ```
    http_user_agent:Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/14.0.803.0 Safari/535.1
    ```

  - 查询和分析语句

    ```
    *| SELECT regexp_extract_all(http_user_agent, '(Chrome)',1) AS Chrome, count(*) AS count GROUP BY Chrome
    ```

  - 查询和分析结果

Chrome	count
["Chrome"]	103440

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子串。

- 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
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。

varchar类型。

- 示例1：提取server_protocol字段值中的第一个数字。
  - 字段样例

    ```
    server_protocol:HTTP/2.0
    ```

- 查询和分析语句

```
*|SELECT regexp_extract(server_protocol, '\d+')
```

- 查询和分析结果

_col0
2

- 示例2：提取request_uri字段值中的文件部分，然后统计各个文件的访问次数。
  - 字段样例

```
request_uri:/request/path-3/file-5
```

  - 查询和分析语句

```
* | SELECT regexp_extract(request_uri, '.*\/(file.*)', 1) AS file, count(*) AS count GROUP BY file
```

  - 查询和分析结果

file	count
file-5	17127

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。

```
regexp_like(x, regular expression)
```

参数	说明
x	参数值为varchar类型。
regular expression	正则表达式。

boolean类型。

判断server_protocol字段值中是否包含数字。

- 字段样例

```
server_protocol:HTTP/2.0
```

- 查询和分析语句

```
*| select regexp_like(server_protocol, '\d+')
```

- 查询和分析结果

_col0
true
true

regexp_replace函数

删除或替换目标字符串中符合正则表达式的子串。

- 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
x	参数值为varchar类型。
regular expression	正则表达式。
replace string	用于替换的子串。

varchar类型。

- 示例1：将region字段值中以cn开头的地域名都替换为**中国**，然后统计来自中国的请求数量。
  - 字段样例

    ```
    region:cn-shanghai
    ```

  - 查询和分析语句

    ```
    * | select regexp_replace(region, 'cn.*','中国') AS region, count(*) AS count GROUP BY region
    ```

  - 查询和分析结果

region	count
中国	168871

- 示例2：删除server_protocol字段值中的版本号部分，然后统计不同通信协议对应的请求数量。
  - 字段样例

    ```
    server_protocol:HTTP/2.0
    ```

  - 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

server_protocol	count
HTTP	168871

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。

```
regexp_split(x, regular expression)
```

参数	说明
x	参数值为varchar类型。
regular expression	正则表达式。

array类型。

使用正斜线（/）分割request_uri字段的值。

- 字段样例

```
request_uri:/request/path-0/file-7
```

- 查询和分析语句

```
* | SELECT regexp_split(request_uri,'/')
```

- 查询和分析结果

11.1.7. 同比和环比函数

本文介绍同比和环比函数的基础语法和示例。

日志服务支持如下同比和环比函数。

> 注意 在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。

函数名称	语法	说明
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、n2、n3秒之前时间周期内的计算结果。
ts_compare函数	ts_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。 注意 ts_compare函数必须按照时间列进行分组（GROUP BY）。
	ts_compare(*x*, *n1*, *n2*, *n3*...)	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。 注意 ts_compare函数必须按照时间列进行分组（GROUP BY）。

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。

- 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
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）。

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, n秒前的UNIX时间戳]。

- 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**1小时（整点时间）**，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），log表示Logstore名称。

  - 查询和分析结果为JSON数组形式
    - 查询和分析语句

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

    - 查询和分析结果

      - **3337.0**表示当前1小时（例如2020-12-25 14:00:00~2020-12-25 15:00:00）的网站访问量。
      - **3522.0**表示昨天同时段（例如2020-12-24 14:00:00~2020-12-24 15:00:00）的网站访问量。
      - **0.947473026689381**表示当前1小时与昨天同时段的网站访问量比值。

- 查询和分析结果为分列显示
  - 查询和分析语句

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

  - 查询和分析结果

today	yesterday	ratio
3337.0	3522.0	0.947473026689381

    - **3337.0**表示当前1小时（例如2020-12-25 14:00:00~2020-12-25 15:00:00）的网站访问量。
    - **3522.0**表示昨天同时段（例如2020-12-24 14:00:00~2020-12-24 15:00:00）的网站访问量。
    - **0.947473026689381**表示当前1小时与昨天同时段的网站访问量比值。

- 示例2：计算今天每小时的网站访问量与昨天同时段、前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**今天（整点时间）**，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），172800表示当前时间减去172800秒（2天）， log表示Logstore名称，date_format(from_unixtime(__time__), '%H:00')表示返回的时间格式。

- 查询和分析结果为JSON数组形式
  - 查询和分析语句

```
* |
SELECT
  time,
  compare(PV, 86400, 172800) as diff
FROM  (
    SELECT
      count(*) as PV,
      date_format(from_unixtime(__time__), '%H:00') as time
    FROM      log
    GROUP BY
      time
  )
GROUP BY
  time
ORDER BY
  time
```

  - 查询和分析结果

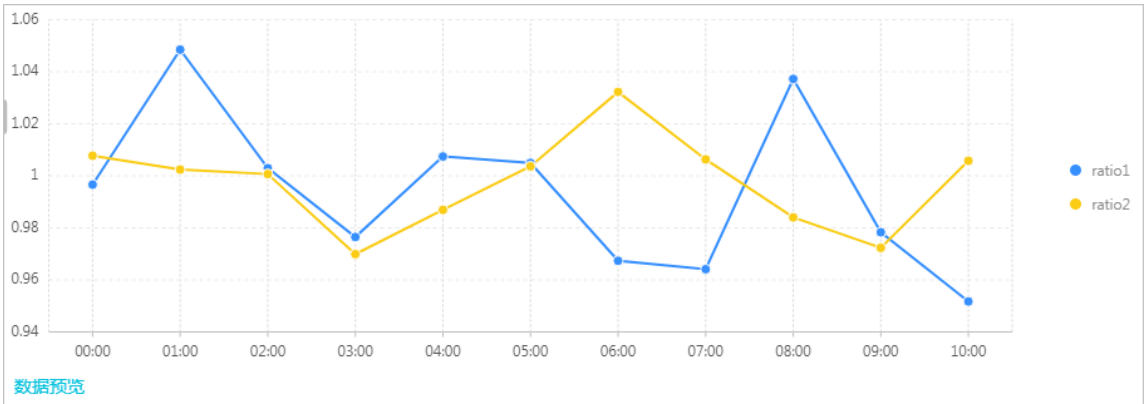
time	diff
00:00	[1176.0,1180.0,1167.0,0.9966101694915255,1.0077120822622108]
01:00	[10077.0,9611.0,10053.0,1.0484861096660077,1.0023873470605789]
02:00	[26921.0,26842.0,26903.0,1.002943148796662,1.0006690703638999]

    - **1176.0**表示当前时段（例如2020-12-25 00:00~01:00）的网站访问量。
    - **1180**表示昨天同时段（例如2020-12-24 00:00~01:00）的网站访问量。
    - **1167.0**表示前天同时段（例如2020-12-23 00:00~01:00）的网站访问量。
    - **0.9966101694915255**表示当前时段与昨天同时段的网站访问量比值。
    - **1.0077120822622108**表示当前时段与前天同时段的网站访问量比值。
- 查询和分析结果为分列显示

- 查询和分析语句

```
* |
SELECT
  time,
  diff [1] AS day1,
  diff [2] AS day2,
  diff [3] AS day3,
  diff [4] AS ratio1,
  diff [5] AS ratio2
FROM  (
    SELECT
      time,
      compare(PV, 86400, 172800) as diff
    FROM      (
        SELECT
          count(*) as PV,
          date_format(from_unixtime(__time__), '%H:00') as time
        FROM          log
        GROUP BY
          time
      )
    GROUP BY
      time
    ORDER BY
      time
  )
```

- 查询和分析结果



数据预览

time	day1	day2	day3	ratio1	ratio2
00:00	1176.0	1180.0	1167.0	0.9966101694915255	1.0077120822622108
01:00	10077.0	9611.0	10053.0	1.0484861096660077	1.0023873470605789

- 示例3：环比12月和11月的网站访问量。

  选择查询和分析的时间范围为**本月（整点时间）**，并执行如下查询和分析语句。其中，2592000表示当前时间减去2592000秒（1个月），log表示Logstore名称，date_trunc('month', __time__)表示使用date_trunc函数将时间对齐到月份。

- 查询和分析语句

```
* |
SELECT
  time,
  compare(PV, 2592000) AS diff
FROM  (
    SELECT
      count(*) AS PV,
      date_trunc('month', __time__) AS time
    FROM      log
    GROUP BY
      time
  )
GROUP BY
  time
ORDER BY
  time
```

- 查询和分析结果

time	diff
2021-01-01 00:00:00.000	[11958378.0,448571.0,26.658829928818404]

ts_compare函数

ts_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。

> 注意 ts_compare函数必须按照时间列进行分组（GROUP BY）。

- 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。

```
ts_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。

```
ts_compare(x, n1, n2, n3...)
```

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）。

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, n秒前的UNIX时间戳]。

环比今天每小时的网站访问量。

选择查询和分析的时间范围为**今天（相对）**，并执行如下查询和分析语句。其中3600表示当前时间减去3600秒（1小时），log表示Logstore名称，date_trunc('hour',__time__)表示使用date_trunc函数将时间对齐到小时。

- 查询和分析语句

```
* |
SELECT
  time,
  ts_compare(PV, 3600) AS data
FROM(
    SELECT
      date_trunc('hour', __time__) AS time,
      count(*) AS PV
    FROM      log
    GROUP BY
      time
    ORDER BY
      time
  )
GROUP BY
  time
```

- 查询和分析结果

time	data
2021-01-27 00:00:00.000	[1160.0,10034.0,0.11560693641618497,1611673200.0]
2021-01-27 01:00:00.000	[10177.0,1160.0,8.773275862068966,1611676800.0]
2021-01-27 02:00:00.000	[26804.0,10177.0,2.6337820575808195,1611680400.0]

11.1.8. 数组函数和运算符

本文介绍数组函数和运算符的基础本语法及示例。

日志服务支持如下数组函数和运算符。

> 注意 在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。

函数名称	语法	说明
下标运算符	[*x*]	返回数组中的第*x*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*)	删除数组中重复的元素。
array_except函数	array_except(*x*, *y*)	计算两个数组的差集。
array_intersect函数	array_intersect(*x*, *y*)	计算两个数组的交集。
	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。 注意 使用array_join函数时，返回结果大小最大为1 KB，超出1 KB的数据会被截断。

array_join函数 函数名称	语法	说明
	array_join(*x*, *delimiter*, *null_replacement*)	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*。 **注意** 使用array_join函数时，返回结果大小最大为1 KB，超出1 KB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_min(*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_transpose函数	array_transpose(*x*)	对矩阵进行转置，即提取二维数组中索引相同的元素组成一个新的二维数组。
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。
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。

函数名称	语法	说明
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。等同于element_at函数。

```
[x]
```

参数	说明
x	数组下标，从1开始。参数值为bigint类型。

返回指定元素的数据类型。

返回number字段值中的第1个元素。

- 字段样例

```
number:[49,50,45,47,50]
```

- 查询和分析语句

```
* | SELECT cast(json_parse(number) as array(bigint)) [1]
```

- 查询和分析结果

array_agg函数

array_agg函数会以数组形式返回*x*中的所有值。

```
array_agg (x)
```

参数	说明
x	参数值为任意数据类型。

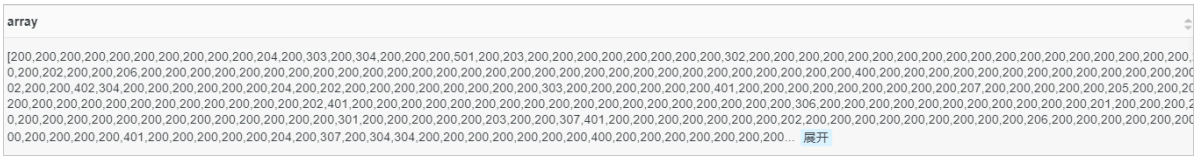
array类型。

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。

```
array_distinct(x)
```

参数	说明
x	参数值为array类型。

array类型。

删除number字段值中重复的元素。

- 字段样例

  ```
  number:[49,50,45,47,50]
  ```

- 查询和分析语句

  ```
  *| SELECT array_distinct(cast(json_parse(number) 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。

```
array_except(x, y)
```

参数	说明
x	参数值为array类型。
y	参数值为array类型。

array类型。

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。

```
array_intersect(x, y)
```

参数	说明
x	参数值为array类型。
y	参数值为array类型。

array类型。

计算数组[1,2,3,4,5]和[1,3,5,7]的交集。

- 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。

- 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
x	参数值为任意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。

varchar类型。

使用空格将数组[null, 'Log','Service']中的元素拼接为一个字符串，其中null元素替换为Alicloud。

- 查询和分析语句

```
* | SELECT array_join(array[null,'Log','Service'],' ','Alicloud')
```

- 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。

```
array_max(x)
```

参数	说明
x	参数值为array类型。 注意 如果数组中包含null，则返回结果为null。

与参数值中元素的数据类型一致。

获取数组中的最大值。

- 字段样例

```
number:[49,50,45,47,50]
```

- 查询和分析语句

```
*| SELECT array_max(try_cast(json_parse(number) as array(bigint))) AS max_number
```

- 查询和分析结果

max_number
50

array_min函数

array_min函数用于获取数组中的最小值。

```
array_min(x)
```

参数	说明
x	参数值为array类型。 注意 如果数组中包含null，则返回结果为null。

与参数值中元素的数据类型一致。

获取数组中的最小值。

- 字段样例

```
number:[49,50,45,47,50]
```

- 查询和分析语句

```
*| SELECT array_min(try_cast(json_parse(number) 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。

```
array_position(x, element)
```

参数	说明
x	参数值为数组类型。
element	数组中的一个元素。 **说明** 如果待获取下标的元素为null，则返回结果也为null。

bigint类型。

返回数组[49,45,47]中45的下标。

- 查询和分析语句

```
* | SELECT array_position(array[49,45,47],45)
```

- 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。

```
array_remove(x, element)
```

参数	说明
x	参数值为array类型。
element	数组中的一个元素。 **说明** 如果待删除的元素为null，则返回结果也为null。

array类型。

删除数组[49,45,47]中45。

- 查询和分析语句

```
* | SELECT array_remove(array[49,45,47],45)
```

- 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。如果有null元素，则null元素排在最后。

```
array_sort(x)
```

参数	说明
x	参数值为array类型。

array类型。

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_transpose函数

array_transpose函数用于对矩阵进行转置，即提取二维数组中索引相同的元素组成一个新的二维数组。

```
array_transpose(x)
```

参数	说明
x	参数值为array(double)类型。

array(double)类型。

提取二维数组中索引相同的元素组成一个新的二维数组，例如数组[0,1,2,3]、[10,19,18,17]、[0,9,8,7]中的0、10、9的索引都为1，则组成数组[0.0,10.0,9.0]。

- 查询和分析语句

```
* | SELECT array_transpose(array[array[0,1,2,3],array[10,19,18,17],array[9,8,7]])
```

- 查询和分析结果

array_union函数

array_union函数用于计算两个数组的并集。

```
array_union(x, y)
```

参数	说明
x	参数值为array类型。
y	参数值为array类型。

array类型。

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。

```
cardinality(x)
```

参数	说明
x	参数值为array类型。

bigint类型。

计算number字段值中元素的个数。

- 字段样例

```
number:[49,50,45,47,50]
```

- 查询和分析语句

```
*| SELECT cardinality(cast(json_parse(number) 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为一个数组。

```
concat(x, y…)
```

参数	说明
x	参数值为array类型。
y	参数值为array类型。

array类型。

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。

```
contains(x, element)
```

参数	说明
x	参数值为数组类型。
element	数组中的一个元素。

boolean类型。

判断region字段值中是否包含cn-beijing。

- 字段样例

```
region:["cn-hangzhou","cn-shanghai","cn-beijing"]
```

- 查询和分析语句

```
*| SELECT contains(cast(json_parse(region) as array(varchar)),'cn-beijing')
```

- 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。

```
element_at(x, y)
```

参数	说明
x	参数值为array类型。
y	数组下标，从1开始。参数值为bigint类型。

任意数据类型。

返回number字段值中的第2个元素。

- 字段样例

```
number:[49,50,45,47,50]
```

- 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。

```
filter(x, lambda_expression)
```

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表达式。

array类型。

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。

```
flatten(x)
```

参数	说明
x	参数值为array类型。

array类型。

将数组[array[1,2,3,4],array[5,2,2,4]转换为一维数组。

- 查询和分析语句

```
* | SELECT flatten(array[array[1,2,3,4],array[5,2,2,4]])
```

- 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。

```
reduce(x, lambda_expression)
```

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表达式。

bigint类型。

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。

```
reverse(x)
```

参数	说明
x	参数值为array类型。

array类型。

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。

- 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
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。

array类型。

- 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

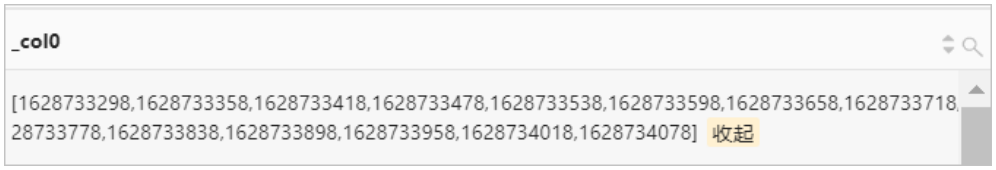
```
ww* | SELECT  sequence(from_unixtime(1508737026),from_unixtime(1628734085),interval '1' year to month )
```

  - 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。

```
shuffle(x)
```

参数	说明
x	参数值为array类型。

array类型。

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

_col0
[3,1,2,4,5]
[5,1,2,4,3]
[2,5,3,1,4]

slice函数

slice函数用于返回数组的子集。

```
slice(x, start, length)
```

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。

array类型。

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

_col0
[4,5]

transform函数

transform函数用于将Lambda表达式应用到数组的每个元素中。

```
transform(x, lambda_expression)
```

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表达式。

array类型。

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

_col0
[6,7]

zip函数

zip函数用于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(x, y...)
```

参数	说明
x	参数值为array类型。
y	参数值为array类型。

array类型。

将数组[1,2,3]、['1b',null,'3b']和[1,2,3]合并为一个二维数组。

- 查询和分析语句

```
* | SELECT zip(array[1,2,3], array['1b',null,'3b'],array[1,2,3])
```

- 查询和分析结果

_col0
[[1,"1b",1],[2,null,2],[3,"3b",3]]

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。

```
zip_with(x, y, lambda_expression)
```

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表达式。

array类型。

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

_col0
[4,6]

11.1.9. Map映射函数和运算符

本文介绍Map映射函数和运算符的基本语法及示例。

日志服务支持如下Map映射函数和运算符。

> **注意** 在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。

函数名称	语法	说明
下标运算符	[*x*]	获取Map中目标键的值。
cardinality函数	cardinality(*x*)	计算Map的大小。
element_at函数	element_at(*x*, *key*)	获取Map中目标键的值。
histogram函数	histogram(*x*)	对查询和分析结果进行分组，返回结果为JSON格式。
histogram_u函数	histogram_u(*x*)	对查询和分析结果进行分组，返回结果为多行多列格式。
map函数	map()	返回一个空Map。
	map(*x*, *y*)	将两个数组映射为一个Map。
map_agg函数	map_agg(*x*, *y*)	将*x*和*y*映射为一个Map。*x*为Map中的键，*y*为Map中的键值。当*y*存在多个值时，随机提取一个值作为键值。
map_concat函数	map_concat(*x*, *y*...)	将多个Map合并为一个Map。
map_filter函数	map_filter(*x*, *lambda_expression*)	结合Lambda表达式，用于过滤Map中的元素。

函数名称	语法	说明
map_keys函数	map_keys(*x*)	提取Map中所有的键，并以数组形式返回。
map_values函数	map_values(*x*)	提取Map中所有键的值，并以数组形式返回。
multimap_agg函数	multimap_agg(*x*, *y*)	将*x*和*y*映射为一个Map。*x*为Map中的键，*y*为Map中的键值，键值为数组格式。当*y*存在多个值时，提取所有的值作为键值。

下标运算符

下标运算符用于获取Map中的目标键的值。

```
[x]
```

参数	说明
x	参数值为varchar类型。

任意数据类型。

日志服务数据加工日志中etl_context字段值为map类型，您可以使用下标运算符获取etl_context字段值中project的值。

- 字段样例

```
etl_context: {
 project:"datalab-148****6461-cn-chengdu"
 logstore:"internal-etl-log"
 consumer_group:"etl-83****4d1965"
 consumer:"etl-b2d40ed****c8d6-291294"
 shard_id:"0" }
```

- 查询和分析语句

```
* | SELECT try_cast(json_parse(etl_context) AS map(varchar, varchar))['project']
```

- 查询和分析结果

cardinality函数

cardinality函数用于计算Map的大小。

```
cardinality(x)
```

参数	说明
x	参数值为map类型。

bigint类型。

使用histogram函数获取各个请求方法对应的请求数量，再通过cardinality函数获取请求方法的种类数。

- 查询和分析语句

```
* |
SELECT
  histogram(request_method) AS request_method,
  cardinality(histogram(request_method)) AS "kinds"
```

- 查询和分析结果

request_method	kinds
{"DELETE":5,"POST":7,"GET":41,"PUT":4}	4

element_at函数

element_at函数用于获取Map中目标键的值。

```
element_at(x, key)
```

参数	说明
x	参数值为map类型。
key	Map中的一个键。

任意数据类型。

使用histogram函数获取各个请求方法对应的请求数量，然后通过element_at函数获取DELETE字段的值。

- 查询和分析语句

```
* |
SELECT
  histogram(request_method) AS request_method,
  element_at(histogram(request_method),'DELETE') AS "count"
```

- 查询和分析结果

request_method	count
{"HEAD":686,"DELETE":13619,"POST":34365,"GET":138097,"PUT":34056}	13619

histogram函数

histogram函数用于对查询和分析结果进行分组，返回结果为JSON格式。类似于 `* | SELECT count(*) GROUP BY x` 。

```
histogram(x)
```

参数	说明
x	参数值为任意数据类型。

map类型。

使用histogram函数获取各个请求方法对应的请求数量。

- 查询和分析语句

```
* | SELECT histogram(request_method) AS request_method
```

- 查询和分析结果

request_method
{"HEAD":30,"DELETE":564,"POST":1382,"GET":5420,"PUT":1334}

histogram_u函数

histogram_u函数用于对查询和分析结果进行分组，返回结果为多行多列。

```
histogram_u(x)
```

参数	说明
x	参数值为任意数据类型。

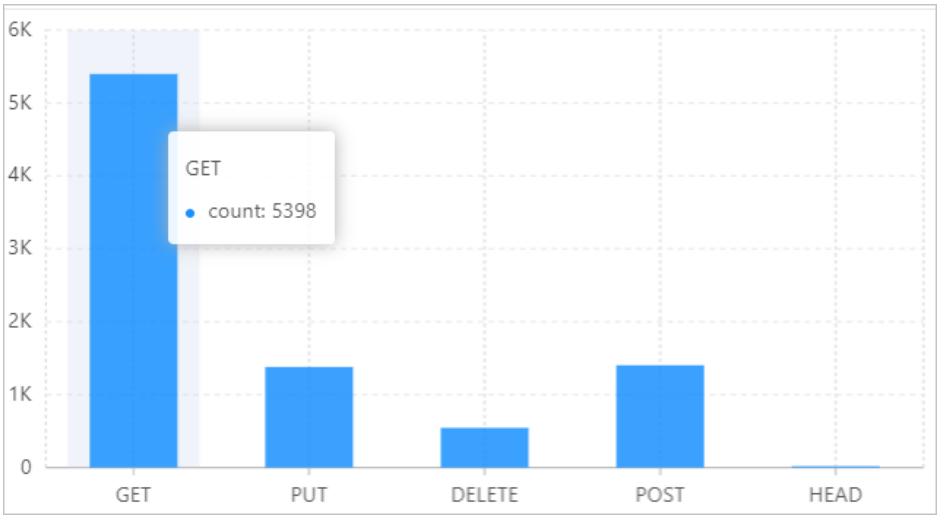
bigint类型。

使用histogram_u函数获取各个请求方法对应的请求数量，并以柱形图展示查询和分析结果。

- 查询和分析语句

```
*|SELECT  histogram_u(request_method) as request_method
```

- 查询和分析结果



map函数

map函数用于返回一个空Map或者将两个数组映射为一个Map。

- 返回一个空Map。

```
map()
```

- 将两个数组映射为一个Map。

```
map(x,y)
```

参数	说明
x	参数值为array类型。
y	参数值为array类型。

map类型。

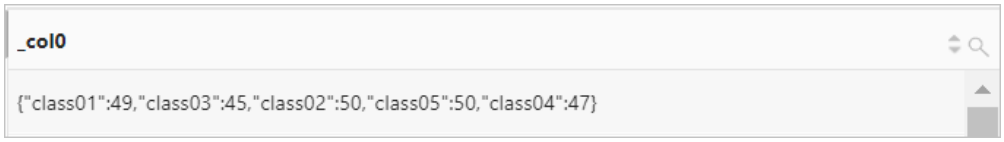
- 示例1：class字段表示班级，number字段表示班级人数，字段值为array类型。现使用map函数将两个字段的值（两个数组）映射为一个Map，将班级和班级人数一一对应。
  - 字段样例

    ```
    class:["class01","class02","class03","class04","class05"]
    number:[49,50,45,47,50]
    ```

  - 查询和分析语句

    ```
    * | SELECT map(try_cast(json_parse(class) AS array(varchar)) ,try_cast(json_parse(number) AS arr
    ay(bigint)))
    ```

  - 查询和分析结果

- 示例2：返回一个空Map。
  - 查询和分析语句

    ```
    *| SELECT map()
    ```

  - 查询和分析结果

map_agg函数

map_agg函数用于将*x*和*y*映射为一个Map。*x*为Map中的键，*y*为Map中的键值。当*y*存在多个值时，随机提取一个值作为键值。

```
map_agg(x, y)
```

参数	说明
x	参数值为任意数据类型。
y	参数值为任意数据类型。

map类型。

提取request_method字段值和request_time字段值，然后映射为一个Map。request_method字段值为Map中的键，request_time字段值为Map中的键值。

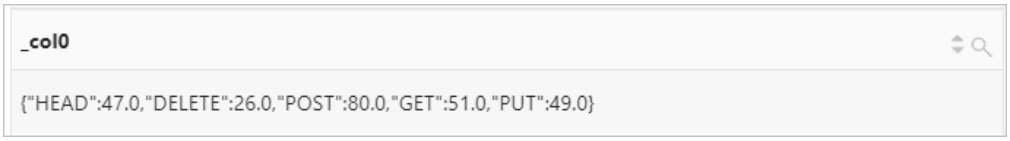
- 字段样例

```
request_method:POST
request_time:80
```

- 查询和分析语句

```
* | SELECT map_agg(request_method,request_time)
```

- 查询和分析结果



map_concat函数

map_concat函数用于将多个Map合并为一个Map。

```
map_concat(x, y)
```

参数	说明
x	参数值为map类型。
y	参数值为map类型。

map类型。

日志服务数据加工日志中etl_context字段值和progress字段值都为map类型，您可以使用map_concat函数将这两个字段值合并为一个Map。

- 字段示例

```
etl_context: {
 project:"datalab-148****6461-cn-chengdu"
 logstore:"internal-etl-log"
 consumer_group:"etl-83****4d1965"
 consumer:"etl-b2d40ed****c8d6-291294"
 shard_id:"0" }
progress: {
 accept:3
 dropped:0
 delivered:3
 failed:0 }
```

- 查询和分析语句

```
* |
SELECT
  map_concat(
    cast (
      json_parse(etl_context) AS map(varchar, varchar)
    ),
    cast (json_parse(progress) AS map(varchar, varchar))
  )
```

- 查询和分析结果

map_filter函数

map_filter函数和Lambda表达式结合，用于过滤Map中的元素。

```
map_filter(x, lambda_expression)
```

参数	说明
x	参数值为map类型。
lambda_expression_expression	Lambda表达式。更多信息，请参见Lambda表达式。

map类型。

将两个数组映射为一个新的Map，且Map中的键值不为null。其中 `(k, v) -> v is not null` 为Lambda表达式。

- 查询和分析语句

```
* | SELECT map_filter(map(array[10, 20, 30], array['a', NULL, 'c']), (k, v) -> v is not null)
```

- 查询和分析结果

map_keys函数

map_keys函数用于提取Map中所有的键，并以数组形式返回。

```
map_keys(x)
```

参数	说明
x	参数值为map类型。

array类型。

日志服务数据加工日志中etl_context字段值为map类型，您可以使用map_keys函数提取etl_context字段值中所有的键。

- 字段样例

```
etl_context: {
 project:"datalab-148****6461-cn-chengdu"
 logstore:"internal-etl-log"
 consumer_group:"etl-83****4d1965"
 consumer:"etl-b2d40ed****c8d6-291294"
 shard_id:"0" }
```

- 查询和分析语句

```
* | SELECT map_keys(try_cast(json_parse(etl_context) AS map(varchar, varchar)))
```

- 查询和分析结果

map_values函数

map_values函数用于提取Map中所有键的值，并以数组形式返回。

```
map_values(x)
```

参数	说明
x	参数值为map类型。

array类型。

日志服务数据加工日志中etl_context字段值为map类型，您可以使用map_values函数提取etl_context字段值中所有键的值。

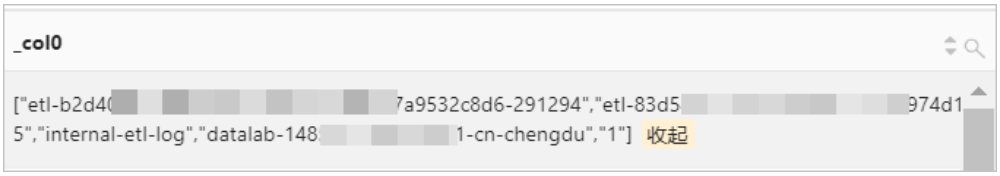
- 字段样例

```
etl_context: {
 project:"datalab-148****6461-cn-chengdu"
 logstore:"internal-etl-log"
 consumer_group:"etl-83****4d1965"
 consumer:"etl-b2d40ed****c8d6-291294"
 shard_id:"0" }
```

- 查询和分析语句

```
* | SELECT map_values(try_cast(json_parse(etl_context) AS map(varchar, varchar)))
```

- 查询和分析结果



multimap_agg函数

multimap_agg函数用于将*x*和*y*映射为一个Map。*x*为Map中的键，*y*为Map中的键值，键值为数组格式。当*y*存在多个值时，提取所有的值作为键值。

```
multimap_agg(x, y)
```

参数	说明
x	参数值为任意数据类型。
y	参数值为任意数据类型。

map类型。

提取request_method字段和request_time字段的所有值，然后映射为一个Map。request_method字段值为Map中的键，request_time字段值为Map中的键值，键值为数组格式。

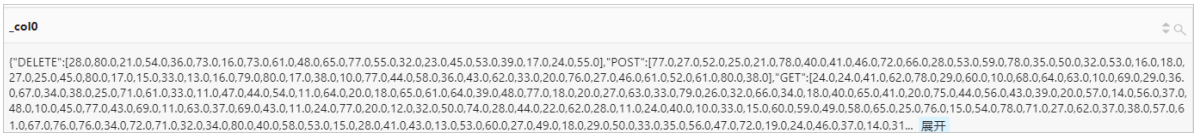
- 字段样例

```
request_method:POST
request_time:80
```

- 查询和分析语句

```
* | SELECT multimap_agg(request_method,request_time)
```

- 查询和分析结果



11.1.10. 数学计算函数

本文介绍数学计算函数的基本语法和示例。

日志服务支持如下数学计算函数。

> **说明**
>
> - 支持如下运算符：
>
> +-*/%
>
> - 在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。

函数名称	语法	说明
abs函数	abs(*x*)	计算*x*的绝对值。
acos函数	acos(*x*)	计算*x*的反余弦。
asin函数	asin(*x*)	计算*x*的反正弦。
atan函数	atan(*x*)	计算*x*的反正切。
atan2函数	atan2(*x*, *y*)	计算*x*和*y*相除的结果的反正切。
cbrt函数	cbrt(*x*)	计算*x*的立方根。
ceil函数	ceil(*x*)	对*x*进行向上取整数。 ceil函数是ceiling函数的别名。
ceiling函数	ceiling(*x*)	对*x*进行向上取整数。
cos函数	cos(*x*)	计算*x*的余弦。
cosh函数	cosh(*x*)	计算*x*的双曲余弦。
cosine_similarity函数	cosine_similarity(*x*, *y*)	计算*x*和*y*之间的余弦相似度。
degrees函数	degrees(*x*)	将弧度转换为度。

函数名称	语法	说明
e函数	e()	返回自然底数e的值。
exp函数	exp(*x*)	计算自然底数e的*x*次幂。
floor函数	floor(*x*)	对*x*进行向下取整数。
from_base函数	from_base(*x*, *y*)	根据BASE编码将*x*转为*y*进制的数字。
ln函数	ln(*x*)	计算*x*的自然对数。
infinity函数	infinity()	返回正无穷的数值。
is_nan函数	is_nan(*x*)	判断*x*是否为NaN。
log2函数	log2(*x*)	计算*x*以2为底的对数。
log10函数	log10(*x*)	计算*x*以10为底的对数。
log函数	log(*x*, *y*)	计算*x*以*y*为底的对数。
mod函数	mod(*x*, *y*)	计算*x*与*y*相除的余数。
nan函数	nan()	返回一个NaN值。
pi函数	pi()	返回π值，精确到小数点后15位。
pow函数	pow(*x*, *y*)	计算*x*的*y*次幂。 pow函数是power函数的别名。
power函数	power(*x*, *y*)	计算*x*的*y*次幂。
radians函数	radians(*x*)	将度转换为弧度。
rand函数	rand()	返回随机数。
random函数	random()	返回[0,1)之间的随机数。
	random(*x*)	返回[0,*x*)之间的随机数。
round函数	round(*x*)	对*x*进行四舍五入取整数。
	round(*x*, *n*)	对*x*进行四舍五入且保留n位小数。
sign函数	sign(*x*)	返回*x*的符号，通过1、0、-1表示。
sin函数	sin(*x*)	计算*x*的正弦。
sqrt函数	sqrt(*x*)	计算*x*的平方根。
tan函数	tan(*x*)	计算*x*的正切。
tanh函数	tanh(*x*)	计算*x*的双曲正切。
to_base函数	to_base(*x*, *y*)	根据BASE编码将*x*转为*y*进制的字符串。
truncate函数	truncate(*x*)	截断*x*的小数部分。

函数名称	语法	说明
width_bucket函数	width_bucke(*x*, *bound1*, *bound2*, *numBuckets*)	将一段数值范围划分成大小相同的多个Bucket，然后返回*x*所属的Bucket。
	width_bucke(*x*, *bins*)	使用数组指定Bucket的范围，然后返回*x*所属的Bucket。

abs函数

abs函数用于计算*x*的绝对值。

```
abs(x)
```

参数	说明
x	参数值为smallint类型、integer类型、real类型、tinyint类型、bigint类型、double类型或decimal类型。

与参数值的类型一致。

计算-25的绝对值。

- 查询和分析语句

```
* | select abs(-25)
```

- 查询和分析结果

_col0
25

acos函数

acos函数用于计算*x*的反余弦。

```
acos(x)
```

参数	说明
x	参数值为double类型，取值范围为[-1,1]。 如果超出[-1,1]，则返回NaN。

double类型。

计算45°角的反余弦。

- 查询和分析语句

```
* | SELECT acos(pi()/4)
```

- 查询和分析结果

_col0
0.9033391107665127

asin函数

asin函数用于计算*x*的反正弦。

```
asin(x)
```

参数	说明
x	参数值为double类型，取值范围为[-1,1]。 如果超出[-1,1]，则返回NaN。

double类型。

计算45°角的反正弦。

- 查询和分析语句

```
* | SELECT  asin(pi()/4)
```

- 查询和分析结果

_col0
0.9033391107665127

atan函数

atan函数用于计算*x*的反正切。

```
atan(x)
```

参数	说明
x	参数值为double类型。

double类型。

计算45°角的反正切。

- 查询和分析语句

```
* | SELECT atan(pi()/4)
```

- 查询和分析结果

_col0
0.6657737500283538

atan2函数

atan2函数用于计算*x*和*y*相除的结果的反正切。

```
atan2(x, y)
```

参数	说明
x	参数值为double类型。
y	参数值为double类型。

double类型。

计算30°角的反正切。

- 查询和分析语句

```
* | SELECT atan2(pi(),6)
```

- 查询和分析结果

cbrt函数

cbrt函数用于计算*x*的立方根。

```
cbrt(x)
```

参数	说明
x	参数值为double类型。

double类型。

计算100的立方根。

- 查询和分析语句

```
* | select cbrt(100)
```

- 查询和分析结果

ceil函数

ceil函数用于对*x*进行向上取整数。ceil函数是ceiling函数的别名。

```
ceil(x)
```

参数	说明
x	参数值为tinyint、smallint、integer、real、bigint、double、decimal类型 。 • *x*为正数，则朝远离0的方向取整。 • *x*为负数，则朝向0的方向取整。

与参数值的类型一致。

对request_time字段的值进行向上取整数。

- 字段样例

```
request_time:9.3
```

- 查询和分析语句

```
* | SELECT ceil(request_time) AS request_time
```

- 查询和分析结果

request_time
10.0

ceiling函数

ceiling函数用于对*x*进行向上取整数。

```
ceiling(x)
```

参数	说明
x	参数值为tinyint、smallint、integer、real、bigint、double、decimal类型 。 • *x*为正数，则朝远离0的方向取整。 • *x*为负数，则朝向0的方向取整。

与参数值的类型一致。

对request_time字段的值进行向上取整数。

- 字段样例

```
request_time:9.3
```

- 查询和分析语句

```
* | SELECT ceiling(request_time) AS request_time
```

- 查询和分析结果

request_time
10.0

cos函数

cos函数用于计算*x*的余弦。

```
cos(x)
```

参数	说明
x	参数值为double类型。

double类型。

计算30°角的余弦。

- 查询和分析语句

```
* | SELECT cos(pi()/6)
```

- 查询和分析结果

_col0
0.8660254037844387

cosh函数

cosh函数用于计算*x*的双曲余弦。

```
cosh(x)
```

参数	说明
x	参数值为double类型。

double类型。

计算30°角的双曲余弦。

- 查询和分析语句

```
* | SELECT cosh(pi()/6)
```

- 查询和分析结果

_col0
1.1402383210764287

cosine_similarity函数

cosine_similarity函数用于计算*x*和*y*之间的余弦相似度。

```
cosine_similarity(x, y)
```

参数	说明
x	参数值为map(varchar,double)类型。
y	参数值为map(varchar,double)类型。

double类型。

计算两个向量之间的余弦相似度。

- 查询和分析语句

```
* | SELECT cosine_similarity(MAP(ARRAY['a'], ARRAY[1.0]), MAP(ARRAY['a'], ARRAY[2.0]))
```

- 查询和分析结果

_col0
1.0

degrees函数

degrees函数用于将弧度转换为度。

```
degrees(x)
```

参数	说明
x	参数值为double类型。

double类型。

将弧度π转换为度。

- 查询和分析语句

```
* | SELECT degrees(pi())
```

- 查询和分析结果

e函数

e函数用于返回自然底数e的值。

```
e()
```

double类型。

返回自然底数e的值。

- 查询和分析语句

```
* | SELECT e()
```

- 查询和分析结果

exp函数

exp函数用于计算自然底数e的x次幂。

```
exp(x)
```

参数	说明
x	参数值为double类型。

double类型。

计算自然底数e的3次幂。

- 查询和分析语句

```
* | SELECT exp(3)
```

- 查询和分析结果

floor函数

floor函数用于对*x*进行向下取整数。

```
floor(x)
```

参数	说明
x	参数值为tinyint、smallint、integer、real、bigint、double、decimal类型 。 • *x*为正数，则朝向0的方向取整。 • *x*为负数，则朝远离0的方向取整。

double类型。

对request_time字段的值进行向下取整数。

- 字段样例

```
request_time:10.3
```

- 查询和分析语句

```
* | SELECT ceiling(request_time) AS request_time
```

- 查询和分析结果

from_base函数

from_base函数将根据BASE编码将*x*转为*y*进制的数字。

```
from_base(x, y)
```

参数	说明
x	参数值为varchar类型。
y	参数值为bigint类型。进制，取值范围为[2,36]。

bigint类型。

将字符串1101转换为数字。

- 查询和分析语句

```
* | SELECT  from_base('1101',2)
```

- 查询和分析结果

ln函数

ln函数用于计算*x*的自然对数。

```
ln(x)
```

参数	说明
x	参数值为double类型且大于0。

double类型。

计算2的自然对数。

- 查询和分析语句

```
* | SELECT ln(2)
```

- 查询和分析结果

infinity函数

infinity函数用于返回正无穷的数值。

```
infinity()
```

double类型。

用于返回正无穷的数值。

- 查询和分析语句

```
* | SELECT infinity()
```

- 查询和分析结果

is_nan函数

is_nan函数用于判断*x*是否为NaN。如果是，则返回true。

```
is_nan(x)
```

参数	说明
x	参数值为double类型。

boolean类型。

判断status字段的值是否为NaN。

- 查询和分析语句

```
* | SELECT is_nan(status)
```

- 查询和分析结果

log2函数

log2函数用于计算*x*以2为底的对数。

```
log2(x)
```

参数	说明
x	参数值为double类型。

double类型。

计算100以2为底的对数。

- 查询和分析语句

```
* | SELECT log2(100)
```

- 查询和分析结果

log10函数

log10函数用于计算*x*以10为底的对数。

```
log10(x)
```

参数	说明
x	参数值为double类型。

double类型。

计算100以10为底的对数。

- 查询和分析语句

```
* | SELECT log10(100)
```

- 查询和分析结果

log函数

log函数用于计算*x*以*y*为底数的对数。

```
log(x, y)
```

参数	说明
x	参数值为double类型。
y	参数值为double类型。

double类型。

计算100以5为底数的对数。

- 查询和分析语句

```
* | SELECT log(100,5)
```

- 查询和分析结果

mod函数

mod函数用于计算*x*与*y*相除的余数。

```
mod(x, y)
```

参数	说明
x	参数值为tinyint、smallint、integer、real、bigint、double、decimal类型。
y	参数值为tinyint、smallint、integer、real、bigint、double、decimal类型。

与参数值的数据类型一致。

计算100与30相除的余数。

- 查询和分析语句

```
* | SELECT mod(100,30)
```

- 查询和分析结果

nan函数

nan函数用于返回一个NaN值（Not a Number）。

```
nan()
```

double类型。

返回一个NaN值。

- 查询和分析语句

```
* | SELECT nan()
```

- 查询和分析结果

pi函数

pi函数用于返回π值，精确到小数点后15位。

```
pi()
```

double类型。

返回π值，精确到小数点后15位。

- 查询和分析语句

```
* | SELECT pi()
```

- 查询和分析结果

pow函数

pow函数用于计算*x*的*y*次幂。pow函数是power函数的别名。

```
pow(x, y)
```

参数	说明
x	参数值为double类型。
y	参数值为double类型。

double类型。

计算2的5次幂。

- 查询和分析语句

```
* | SELECT pow(2,5)
```

- 查询和分析结果

power函数

power用于计算*x*的*y*次幂。

```
power(x, y)
```

参数	说明
x	参数值为double类型。
y	参数值为double类型。

double类型。

计算2的5次幂。

- 查询和分析语句

```
* | SELECT power(2,5)
```

- 查询和分析结果

radians函数

radians函数用于将度转换为弧度。

```
radians(x)
```

参数	说明
x	参数值为double类型。

double类型。

计算180°对应的弧度。

- 查询和分析语句

```
* | SELECT radians(180)
```

- 查询和分析结果

rand函数

rand函数用于返回随机数。

```
rand()
```

double类型。

返回一个随机数。

- 查询和分析语句

```
* | select rand()
```

- 查询和分析结果

random函数

random函数用于返回[0,x)之间的随机数。

- 返回[0,1)之间的随机数。

```
random()
```

- 返回[0,x)之间的随机数。

```
random(x)
```

参数	说明
x	参数值为tinyint、smallint、integer或bigint类型。

与参数值的类型一致。

返回[0,100)之间的随机数。

- 查询和分析语句

```
* | select random(100)
```

- 查询和分析结果

round函数

round函数用于对*x*进行四舍五入。如果*n*存在，则保留n位小数；如果*n*不存在，则对*x*进行四舍五入取整数。

- 对*x*进行四舍五入取整数。

```
round(x)
```

- 对*x*进行四舍五入且保留n位小数。

```
round(x, n)
```

参数	说明
x	参数值为tinyint、smallint、integer或bigint类型。
n	n位小数。

与参数值的数据类型一致。

同比今天与昨天的访问PV，并使用百分数表示。

- 查询和分析语句

```
* | SELECT diff [1] AS today, round((diff [3] -1.0) * 100, 2) AS growth FROM (SELECT compare(pv, 8
6400) as diff FROM (SELECT COUNT(*) as pv FROM website_log))
```

- 查询和分析结果

sign函数

sign函数用于返回*x*的符号，通过1、0、-1表示。

```
sign(x)
```

参数	说明
x	参数值为integer、smallint、tinyint、real、 double、bigint或decimal(p,s)类型。 • *x*为正数，返回1。 • *x*为0，返回0。 • *x*为负数，返回-1。

与参数值的数据类型一致。

计算数字10的符号。

- 查询和分析语句

```
* | SELECT sign(10)
```

- 查询和分析结果

sin函数

sin函数用于计算*x*的正弦。

```
sin(x)
```

参数	说明
x	参数值为double类型。

double类型。

计算90°角的正弦。

- 查询和分析语句

```
* | select sin(pi()/2)
```

- 查询和分析结果

_col0
1.0

sqrt函数

sqrt函数用于计算*x*的平方根。

```
sqrt(x)
```

参数	说明
x	参数值为double类型。

double类型。

计算100的平方根。

- 查询和分析语句

```
* | select sqrt(100)
```

- 查询和分析结果

_col0
10.0

tan函数

tan函数用于计算*x*的正切。

```
tan(x)
```

参数	说明
x	参数值为double类型。

double类型。

计算tan30°角的正切。

- 查询和分析语句

```
* | SELECT tan(pi()/6)
```

- 查询和分析结果

_col0
0.5773502691896257

tanh函数

tanh函数用于计算*x*的双曲正切。

```
tanh(x)
```

参数	说明
x	参数值为double类型。

double类型。

计算30°的双曲正切。

- 查询和分析语句

```
* | SELECT tanh(pi()/6)
```

- 查询和分析结果

to_base函数

to_base函数将根据BASE编码将*x*转为*y*进制的字符串。

```
to_base(x, y)
```

参数	说明
x	参数值为bigint类型。
y	参数值为bigint类型。进制，取值范围为[2,36]。

varchar类型。

将180转换为二进制字符串。

- 查询和分析语句

```
* | SELECT  to_base(180, 2)
```

- 查询和分析结果

truncate函数

truncate函数用于截断*x*的小数部分。

```
truncate(x)
```

参数	说明
x	参数值为double类型。

double类型。

截断11.11的小数部分。

- 查询和分析语句

```
* | SELECT  truncate(11.11)
```

- 查询和分析结果

width_bucket函数

width_bucket函数用于返回*x*所属的Bucket。

- 将一段数值范围划分成大小相同的多个Bucket，然后返回*x*所属的Bucket。

```
width_bucket(x, bound1, bound2, numBuckets)
```

- 使用数组指定Bucket的范围，然后返回*x*所属的Bucket。

```
width_bucket(x, bins)
```

参数	说明
x	参数值为double类型。
bound1	提供数值范围的下限。
bound2	提供数值范围的上限。
numBuckets	Bucket数量。大于0的整数。
bins	使用数组指定Bucket的范围。*bins*为double类型的数组。

bigint类型。

> 说明
> - *x*在指定范围内，返回*x*的所属的Bucket。
> - *x*在下限范围外，返回0。
> - *x*在上限范围外，返回*numBuckets+1*。

- 示例1：将[10,80)范围等分为7个Bucket，然后返回request_time字段的各个值所属的Bucket。
  - 查询和分析语句

```
* | SELECT request_time, width_bucket(request_time, 10, 80,7) AS numBuckets
```

- 查询和分析结果

request_time	numBuckets
26.0	2
49.0	4
10.0	1
34.0	3
40.0	4
77.0	7
28.0	2
66.0	6
23.0	2

- 示例2：使用数组指定7个Bucket的范围，然后返回request_time字段的各个值所属的Bucket。
  - 查询和分析语句

```
* | SELECT request_time, width_bucket(request_time, array[10,20,30,40,50,60,70,80]) AS numBuckets
```

  - 查询和分析结果

request_time	numBuckets
26.0	2
13.0	1
28.0	2
30.0	3
67.0	6
12.0	1
54.0	5
10.0	1
45.0	4

11.1.11. 数学统计函数

本文介绍数学统计函数的基础语法及示例。

日志服务支持如下数学统计函数。

> 注意 在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。

函数名称	语法	说明
corr函数	corr(*x*, *y*)	计算*x*和*y*的相关度。计算结果范围为[0,1]。
covar_pop函数	covar_pop(*x*, *y*)	计算*x*和*y*的总体协方差。

函数名称	语法	说明
covar_samp函数	covar_samp(*x*, *y*)	计算*x*和*y*的样本协方差。
regr_intercept函数	regr_intercept(*y*, *x*)	根据输入点（*x*, *y*）拟合成一个线性方程，然后计算该直线的Y轴截距。
regr_slope函数	regr_slope(*y*, *x*)	根据输入点（*x*, *y*）拟合成一个线性方程，然后计算该直线的斜率。
stddev函数	stddev(*x*)	计算*x*的样本标准差。与stddev_samp函数同义。
stddev_samp函数	stddev_samp(*x*)	计算*x*的样本标准差。
stddev_pop函数	stddev_pop(*x*)	计算*x*的总体标准差。
variance函数	variance(*x*)	计算*x*的样本方差。与var_samp函数同义。
var_samp函数	var_samp(*x*)	计算*x*的样本方差。
var_pop函数	var_pop(*x*)	计算*x*的总体方差。

corr函数

corr函数用于计算*x*和*y*的相关度。返回的值越大表示两列的相关性越高。

```
corr(x, y)
```

参数	说明
x	参数值为double类型。
y	参数值为double类型。

double类型，取值范围[0,1]。

计算request_length字段值与request_time字段值的相关度。

- 查询和分析语句

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

- 查询和分析结果

_col0
0.0008096234574114261

covar_pop函数

covar_pop函数用于计算*x*和*y*的总体协方差。

```
covar_pop(x, y)
```

参数	说明
x	参数值为double类型。

参数	说明
y	参数值为double类型。

double类型。

计算每分钟内税前利润和税前营业额的总体协方差。

- 查询和分析语句

```
*|
SELECT
  covar_pop(PretaxGrossAmount, PretaxAmount) AS "总体协方差",
  time_series(__time__, '1m', '%H:%i:%s', '0') AS time
GROUP BY
  time
```

- 查询和分析结果

样本协方差	time
4.402252063414699	04:24:00
2.2477524300733857	04:47:00

covar_samp函数

covar_samp函数用于计算*x*和*y*的样本协方差。

```
covar_samp(x, y)
```

参数	说明
x	参数值为double类型。
y	参数值为double类型。

double类型。

计算每分钟内税前利润和税前营业额的样本协方差。

- 查询和分析语句

```
*|
SELECT
  covar_samp(PretaxGrossAmount, PretaxAmount) AS "样本协方差",
  time_series(__time__, '1m', '%H:%i:%s', '0') AS time
GROUP BY
  time
```

- 查询和分析结果

样本协方差	time
2.2910940581194376	15:50:00
4.721554417070316	15:49:00

regr_intercept函数

regr_intercept函数会根据输入点（x,y）拟合成一个线性方程，然后计算该直线的Y轴截距。x是依赖值，y是独立值。

```
regr_intercept(y, x)
```

参数	说明
y	参数值为double类型。
x	参数值为double类型。

double类型。

计算由request_time字段值和request_length字段值组成的直线的Y轴截距。

- 查询和分析语句

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

- 查询和分析结果

_col0
4128.22910642988

regr_slope函数

regr_slope函数会根据输入点（x,y）拟合成一个线性方程，然后计算该直线的斜率。x是依赖值，y是独立值。

```
regr_slope(y, x)
```

参数	说明
y	参数值为double类型。
x	参数值为double类型。

double类型。

计算由request_time字段值和request_length字段值组成的直线的斜率。

- 查询和分析语句

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

- 查询和分析结果

_col0
1.9022724330993215

stddev函数

stddev函数用于计算x的样本标准差。与stddev_samp函数同义。

```
stddev(x)
```

参数	说明
x	参数值为double类型或bigint类型。

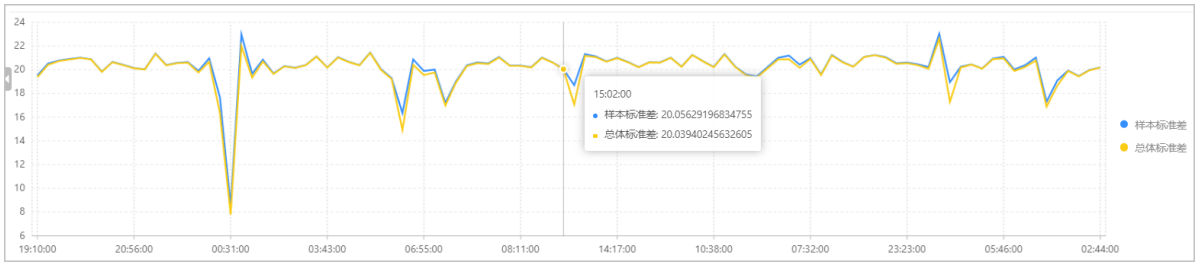
double类型。

查询税前收入的样本标准差和总体标准差，并通过折线图展示。

- 查询和分析语句

```
* |
SELECT
  stddev(PretaxGrossAmount) as "样本标准差",
  stddev_pop(PretaxGrossAmount) as "总体标准差",
  time_series(__time__, '1m', '%H:%i:%s', '0') AS time
GROUP BY
  time
```

- 查询和分析结果



stddev_samp函数

stddev_samp函数用于计算*x*的样本标准差。

```
stddev_samp(x)
```

参数	说明
x	参数值为double类型或bigint类型。

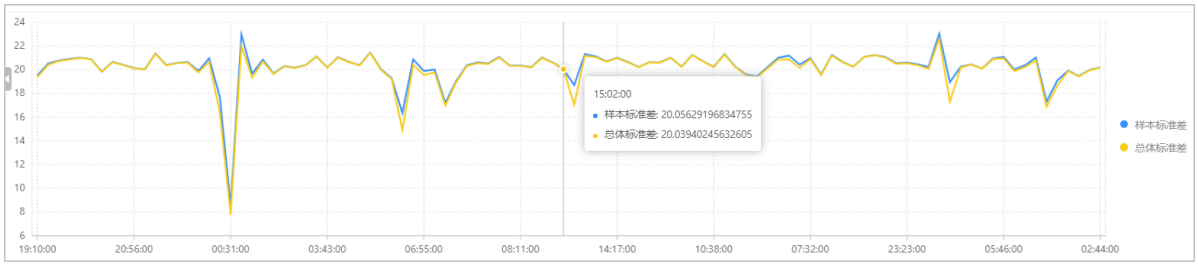
double类型。

查询税前收入的样本标准差和总体标准差，并通过折线图展示。

- 查询和分析语句

```
* |
SELECT
  stddev_samp(PretaxGrossAmount) as "样本标准差",
  stddev_pop(PretaxGrossAmount) as "总体标准差",
  time_series(__time__, '1m', '%H:%i:%s', '0') AS time
GROUP BY
  time
```

- 查询和分析结果



stddev_pop函数

stddev_pop函数用于计算x的总体标准差。

```
stddev_pop(x)
```

参数	说明
x	参数值为double类型或bigint类型。

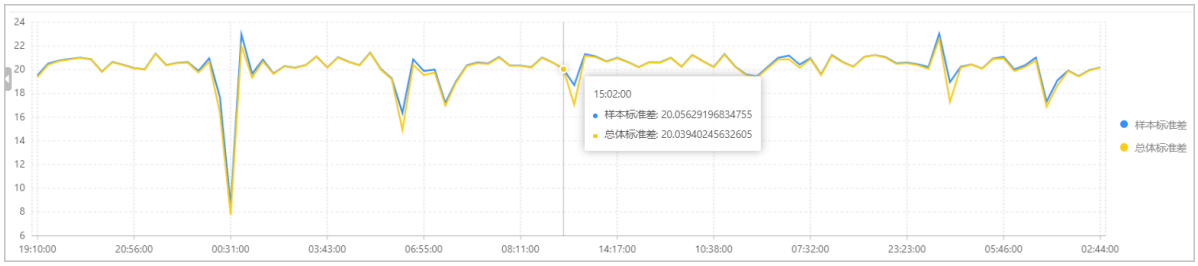
double类型。

查询税前收入的样本标准差和总体标准差，并通过折线图展示。

- 查询和分析语句

```
* |
SELECT
  stddev(PretaxGrossAmount) as "样本标准差",
  stddev_pop(PretaxGrossAmount) as "总体标准差",
  time_series(__time__, '1m', '%H:%i:%s', '0') AS time
GROUP BY
  time
```

- 查询和分析结果



variance函数

variance函数用于计算x的样本方差。与var_samp函数同义。

```
variance(x)
```

参数	说明
x	参数值为double类型或bigint类型。

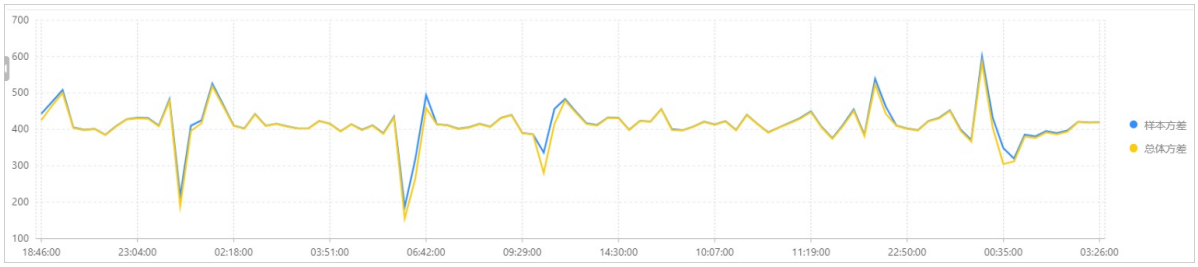
double类型。

查询税前收入的样本方差和总体方差，并通过折线图展示。

- 查询和分析语句

```
* |
SELECT
  variance(PretaxGrossAmount) as "样本方差",
  var_pop(PretaxGrossAmount) as "总体方差",
  time_series(__time__, '1m', '%H:%i:%s', '0') as time
GROUP BY
  time
```

- 查询和分析结果



var_samp函数

var_samp函数用于计算*x*的样本方差。

```
var_samp(x)
```

参数	说明
x	参数值为double类型或bigint类型。

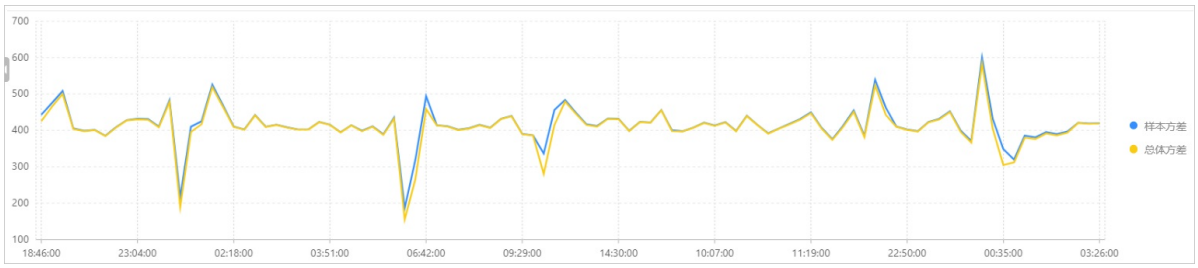
double类型。

查询税前收入的样本方差和总体方差，并通过折线图展示。

- 查询和分析语句

```
* |
SELECT
  var_samp(PretaxGrossAmount) as "样本方差",
  var_pop(PretaxGrossAmount) as "总体方差",
  time_series(__time__, '1m', '%H:%i:%s', '0') as time
GROUP BY
  time
```

- 查询和分析结果



var_pop函数

var_pop函数用于计算*x*的总体方差。

```
var_pop(x)
```

参数	说明
x	参数值为double类型或bigint类型。

double类型。

查询税前收入的样本方差和总体方差，并通过折线图展示。

- 查询和分析语句

```
* |
SELECT
  variance(PretaxGrossAmount) as "样本方差",
  var_pop(PretaxGrossAmount) as "总体方差",
  time_series(__time__, '1m', '%H:%i:%s', '0') as time
GROUP BY
  time
```

- 查询和分析结果

11.1.12. 类型转换函数

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。

日志服务支持如下类型转换函数。

> 注意 在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。 **说明** 日志中可能有脏数据，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。
typeof函数	typeof(*x*)	返回*x*的数据类型。

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。

```
cast(x as type)
```

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。 例如 `cast(json_parse(key) as array(varchar))` 。 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。

由您配置的*type*参数决定。

将数字1转换为boolean格式。

- 查询和分析语句

```
* | select cast(1 as boolean)
```

- 查询和分析结果

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。

```
try_cast(x as type)
```

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。 例如 `try_cast(json_parse(key) as map(varchar, varchar))` 。 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。

由您配置的*type*参数决定。

将uid字段值转换为varchar类型。

- 查询和分析语句

```
* | select try_cast(uid as varchar)
```

- 查询和分析结果

typeof函数

typeof函数用于返回*x*的数据类型。

```
typeof(x)
```

参数	说明
x	参数值可以为任意数据类型。

varchar类型。

判断request_time字段值的数据类型。

- 查询和分析语句

```
* |SELECT typeof(request_time)
```

- 查询和分析结果

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

11.1.13. 安全检测函数

日志服务依托全球白帽子共享安全资产库，提供安全检测函数。您只需通过日志中的IP地址、域名或者URL，即可检测其是否安全。本文介绍安全检测函数的基本语法及示例。

应用场景

安全检测函数适用如下场景：

- 针对服务运维有较强需求的企业和机构（例如互联网、游戏、咨询等），其IT人员和安全运维人员可及时发现可疑访问、攻击及侵入网站等行为，并及时采取措施。
- 针对内部资产保护有较强需求的企业和机构（例如银行、证券、电商等），其IT人员和安全运维人员可及时发现内部访问危险网站及下载木马等行为，并及时采取行动。

功能特点

安全检测函数具备如下功能特点：

- 可靠：依托全球共享的白帽子安全资产库，并及时更新。
- 快速：检测百万IP地址、域名或URL仅需几秒钟。
- 简单：支持任何网络日志，调用3个SQL函数security_check_ip、security_check_domain、security_check_url即可获得结果。
- 灵活：交互式查询及可视化展示，支持创建告警。

函数列表

日志服务支持如下安全检查函数。

> 注意 在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。

函数名称	语法	说明
security_check_ip函数	security_check_ip(*x*)	检查IP地址是否安全。
security_check_domain函数	security_check_domain(*x*)	检查域名是否安全。
security_check_url函数	security_check_url(*x*)	检查URL是否安全。

security_check_ip函数

security_check_ip函数用于检查IP地址是否安全。

```
security_check_ip(x)
```

参数	说明
x	参数值为IP地址。

bigint类型，取值说明：

- 1：不安全。
- 0：安全。

通过client_ip字段统计网站的不安全客户端信息。

- 查询和分析语句

```
* |
SELECT
  client_ip,
  ip_to_country(client_ip,'en') AS country,
  ip_to_provider(client_ip) AS provider,
  count(1) AS PV
WHERE
  security_check_ip(client_ip) = 1
GROUP BY
  client_ip
ORDER BY
  PV DESC
```

- 查询和分析结果

client_ip	country	provider	PV
180 3	CN	电信	3
103	CN		3
180 7	CN	电信	1

security_check_domain函数

security_check_domain函数用于检查域名是否安全。

```
security_check_domain(x)
```

参数	说明
x	参数值为域名。

bigint类型，取值说明：

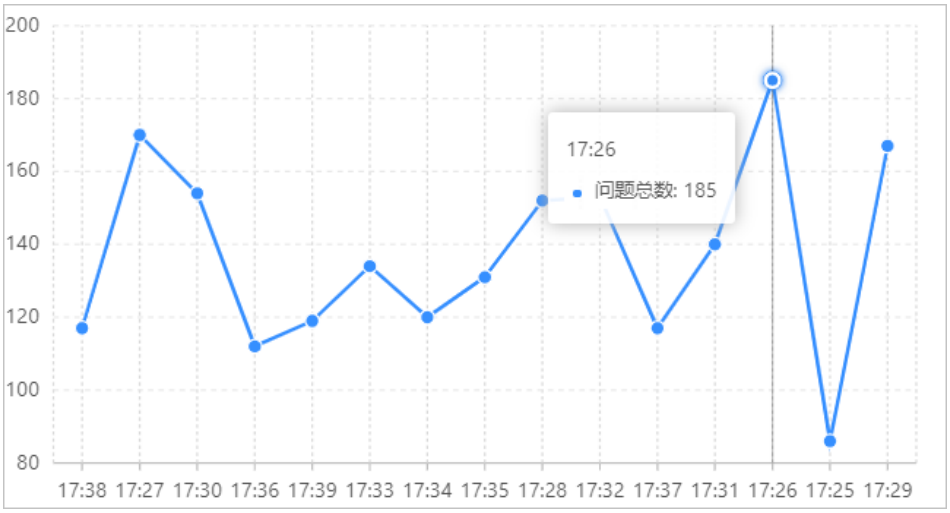
- 1：不安全。
- 0：安全。

通过网站域名统计网站每分钟出现不安全访问的次数，并通过折线图展示。

- 查询和分析语句

```
status : * |
SELECT
  count_if(
    security_check_domain (http_referer) != 0
  ) AS "问题总数",
  time_series(__time__, '1m', '%H:%i:%s', '0') AS time
GROUP BY
  time
```

- 查询和分析结果



security_check_url函数

security_check_url函数用于检查URL地址是否安全。

```
security_check_url(x)
```

参数	说明
x	参数值为URL地址。

bigint类型，取值说明：

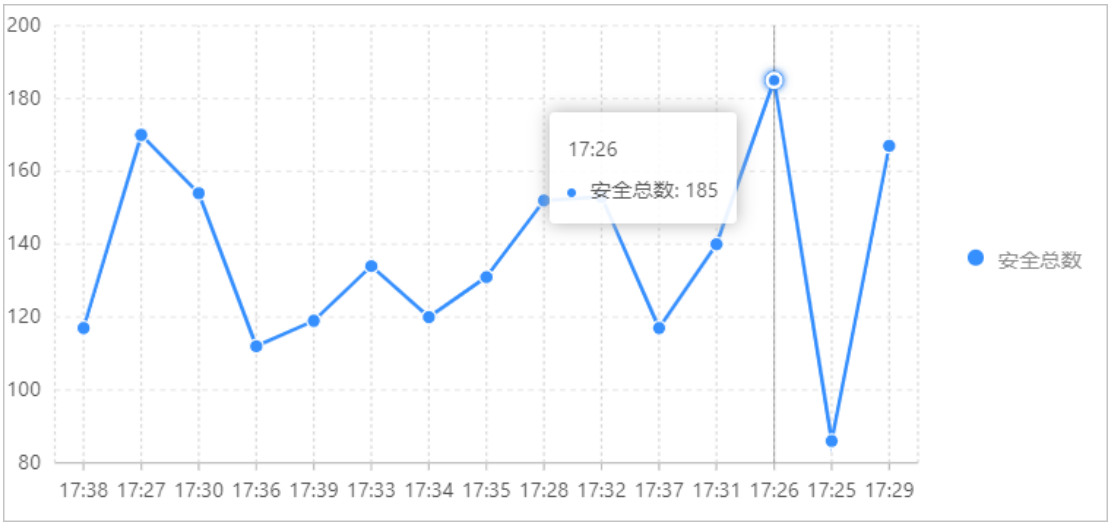
- 1：不安全。
- 0：安全。

通过URL地址统计网站每分钟安全访问的次数，并通过折线图展示。

- 查询和分析语句

```
status : * |
SELECT
  count_if(
    security_check_url (request_uri) = 0
  ) AS "安全总数",
  time_series(__time__, '1m', '%H:%i', '0') as time
GROUP BY
  time
LIMIT
  20
```

- 查询和分析结果



11.1.14. 窗口函数

本文介绍窗口函数的基本语法及示例。

简介

普通的聚合函数只能用来计算一行内的结果或把所有行聚合成一行结果，而窗口函数支持为每一行生成一个结果。窗口函数包含分区、排序和框架这3个核心元素。更多信息，请参见Window Function Concepts and Syntax。

```
function over (
    [partition by partition_expression]
    [order by order_expression]
    [frame]
)
```

- 分区：分区元素由partition by子句定义。partition by子句用于划分窗口分区，如果没有指定partition by子句，则整个查询与分析结果集作为一个窗口分区。
- 排序：排序元素由order by子句定义。order by子句用于对窗口分区内的行进行排序。

  > **说明** 使用order by子句对重复的数值进行排序时，排序结果不稳定。如果您希望每次排序结果相同，可指定多个列进行排序。例如 `order by request_time, request_method` 。

- 框架：框架元素在窗口分区内对行进一步限制。框架元素不适用于排名函数。框架子句的语法为 `{ rows | range} { frame_start | frame_between }` ，例如 `range between unbounded preceding and unbounded following` 。更多信息，请参见Window Function Frame Specification。

函数列表

分类	函数名称	语法	说明
聚合函数	聚合函数	无	所有聚合函数都支持在窗口函数中使用。聚合函数列表请参见聚合函数。
排名函数	cume_dist函数	cume_dist()	统计窗口分区内各个值的累计分布。即计算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。返回值范围为(0,1]。
	dense_rank函数	dense_rank()	窗口分区内值的排名。相同值拥有相同的排名，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。
	ntile函数	ntile(*n*)	将窗口分区内数据按照顺序分成N组。
	percent_rank函数	percent_rank()	计算窗口分区内各行的百分比排名。
	rank函数	rank()	窗口分区内值的排名。相同值拥有相同的排名，排名不是连续的，例如有两个相同值的排名为1，则下一个值的排名为3。
	row_number函数	row_number()	窗口分区内值的排名。每个值拥有唯一的序号，从1开始。三个相同值的排名为1、2、3。
偏移函数	first_value函数	first_value(*x*)	返回各个窗口分区内第一行的值。
	last_value函数	last_value(*x*)	返回各个窗口分区内最后一行的值。
	lag函数	lag(*x*, *offset*, *defaut_value*)	返回窗口分区内位于当前行上方第*offset*行的值。如果不存在该行，则返回*defaut_value*。
	lead函数	lead(*x*, *offset*, *defaut_value*)	返回窗口分区内位于当前行下方第*offset*行的值。如果不存在该行，则返回*defaut_value*。
	nth_value函数	nth_value(*x*, *offset*)	返回窗口分区中第*offset*行的值。

聚合函数

所有聚合函数都支持在窗口函数中使用。聚合函数列表请参见聚合函数。此处以sum函数为例。

```
sum() over (
    [partition by partition_expression]
    [order by order_expression]
    [frame]
)
```

参数	说明
partition by *partition_expression*	窗口分区，根据分区表达式将数据划分成不同的分区。
order by *order_expression*	窗口排序，根据排序表达式对各个分区内的每一行进行排序。
frame	窗口框架，例如 `range between unbounded preceding and unbounded following` 。

double类型。

按照部门分区，获取每个员工薪水在部门内的占比。

- 查询和分析语句

```
* |
SELECT
  department,
  staff_name,
  salary,
  round ( salary * 1.0 / sum(salary) over(partition by department), 3) AS salary_percentage
```

- 查询和分析结果

department	staff_name	salary	salary_percentage
dev	Rob	9000	0.277
dev	Blan	8500	0.262
dev	Sansa	8000	0.246
dev	Snow	7000	0.215
Marketing	Achilles	8500	0.362
Marketing	San	8000	0.340
Marketing	Blan	7000	0.298

cume_dist函数

cume_dist函数用于统计窗口分区内各个值的累计分布。即计算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。返回值范围为(0,1]。

```
cume_dist() over (
    [partition by partition_expression]
    [order by order_expression]
)
```

参数	说明
partition by *partition_expression*	窗口分区，根据分区表达式将数据划分成不同的分区。

参数	说明
order by *order_expression*	窗口排序，根据排序表达式对各个分区内的每一行进行排序。

double类型。

统计名为bucket00788的OSS Bucket内各个对象的大小的累计分布。

- 查询和分析语句

```
bucket=bucket00788 |
select
  object,
  object_size,
  cume_dist() over (
    partition by object
    order by
      object_size
  ) as cume_dist
from  oss-log-store
```

- 查询和分析结果

object	object_size	cume_dist
dashboard%2F2020%2F05%2F20%2F16%2F47.csv	4591	0.5
dashboard%2F2020%2F05%2F20%2F16%2F47.csv	6469	1.0
oss-log-562[redacted]11-cn-hangzhou%2Foss-log-store_oss_access_center_en%2Freport-1581933277-637788	1526	0.3333333333333333
oss-log-56[redacted]911-cn-hangzhou%2Foss-log-store_oss_access_center_en%2Freport-1581933277-637788	1921	0.6666666666666666
oss-log-56[redacted]11-cn-hangzhou%2Foss-log-store_oss_access_center_en%2Freport-1581933277-637788	6074	1.0
245-da918c.model	1818	0.08333333333333333
245-da918c.model	1854	0.16666666666666667

dense_rank函数

dense_rank函数用于窗口分区内值的排名。相同值拥有相同的排名，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。

```
dense_rank() over (
    [partition by partition_expression]
    [order by order_expression]
)
```

参数	说明
partition by *partition_expression*	窗口分区，根据分区表达式将数据划分成不同的分区。
order by *order_expression*	窗口排序，根据排序表达式对各个分区内的每一行进行排序。

bigint类型。

按照部门分区，计算员工薪水在部门内的排名。

- 查询和分析语句

```
* |
select
  department,
  staff_name,
  salary,
  dense_rank() over(
    partition by department
    order by
      salary desc
  ) as salary_rank
order by
  department,
  salary_rank
```

- 查询和分析结果

department	staff_name	salary	salary_rank
Marketing	Blan Stark	9000	1
Marketing	Smith	9000	1
Marketing	Achilles	8000	2
dev	Rob	9000	1
dev	Blan	8500	2
dev	Sansa	8000	3

ntile函数

ntile函数用于将窗口分区内数据按照顺序分成N组。

```
ntile(n) over (
    [partition by partition_expression]
    [order by order_expression]
)
```

参数	说明
n	组数。
partition by *partition_expression*	窗口分区，根据分区表达式将数据划分成不同的分区。
order by *order_expression*	窗口排序，根据排序表达式对各个分区内的每一行进行排序。

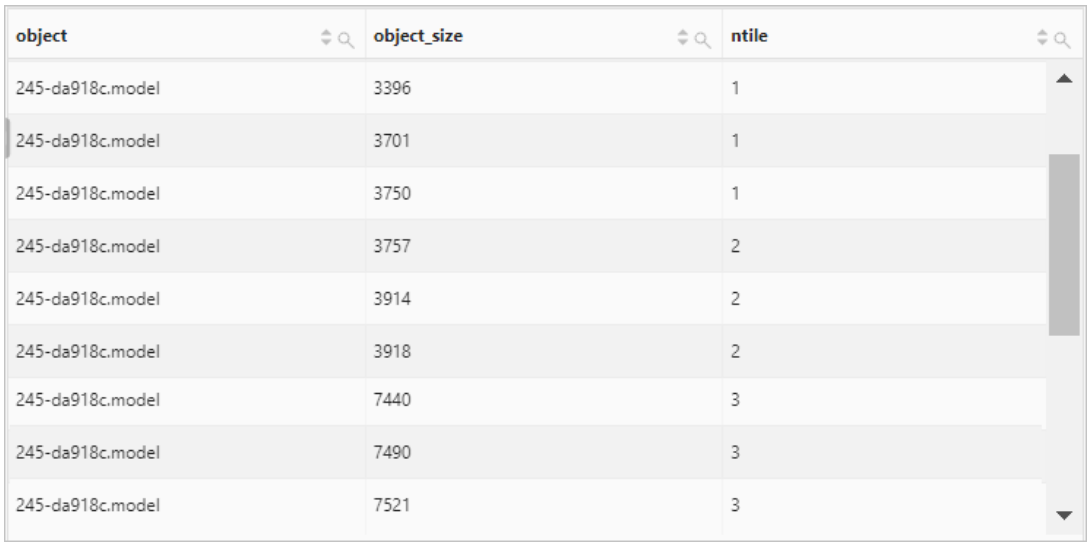
bigint类型。

将指定对象中的数据分成3组。

- 查询和分析语句

```
object=245-da918c.model |
select
  object,
  object_size,
  ntile(3) over (
    partition by object
    order by
      object_size
  ) as ntile
from  oss-log-store
```

- 查询和分析结果



object	object_size	ntile
245-da918c.model	3396	1
245-da918c.model	3701	1
245-da918c.model	3750	1
245-da918c.model	3757	2
245-da918c.model	3914	2
245-da918c.model	3918	2
245-da918c.model	7440	3
245-da918c.model	7490	3
245-da918c.model	7521	3

percent_rank函数

函数用于计算窗口分区内各行的百分比排名。计算公式为 `(rank - 1) / (total_rows - 1)` ，其中rank为当前行的排名，total_rows为当前窗口分区内的总行数。

```
percent_rank() over (
    [partition by partition_expression]
    [order by order_expression]
)
```

参数	说明
partition by *partition_expression*	窗口分区，根据分区表达式将数据划分成不同的分区。
order by *order_expression*	窗口排序，根据排序表达式对各个分区内的每一行进行排序。

double类型。

计算目标OSS对象的不同大小的百分比排名。

- 查询和分析语句

```
object=245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model|
select
  object,
  object_size,
 percent_rank() over (
    partition by object
    order by
      object_size
  ) as ntile
FROM  oss-log-store
```

- 查询和分析结果

object	object_size	ntile
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	7442	0.0
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	7635	0.2
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	8221	0.4
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	8272	0.6
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	8706	0.8
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	8988	1.0

rank函数

函数用于窗口分区内值的排名。相同值拥有相同的排名，排名不是连续的，例如有两个相同值的排名为1，则下一个值的排名为3。

```
rank() over (
    [partition by partition_expression]
    [order by order_expression]
)
```

参数	说明
partition by *partition_expression*	窗口分区，根据分区表达式将数据划分成不同的分区。
order by *order_expression*	窗口排序，根据排序表达式对各个分区内的每一行进行排序。

bigint类型。

按照部门分区，计算员工薪水在部门内的排名。

- 查询和分析语句

```
* |
select
  department,
  staff_name,
  salary,
  rank() over(
    partition by department
    order by
      salary desc
  ) as salary_rank
order by
  department,
  salary_rank
```

- 查询和分析结果

department	staff_name	salary	salary_rank
Marketing	Blan Stark	9000	1
Marketing	Smith	9000	1
Marketing	Achilles	8000	3
dev	Rob	9000	1
dev	Blan	8500	2
dev	Sansa	8000	3

row_number函数

row_number函数用于窗口分区内值的排名。每个值拥有唯一的序号，从1开始。

```
row_number() over (
    [partition by partition_expression]
    [order by order_expression]
)
```

参数	说明
partition by *partition_expression*	窗口分区，根据分区表达式将数据划分成不同的分区。
order by *order_expression*	窗口排序，根据排序表达式对各个分区内的每一行进行排序。

bigint类型。

按照部门分区，计算员工薪水在部门内的排名。

- 查询和分析语句

```
* |
select
  department,
  staff_name,
  salary,
  row_number() over(
    partition by department
    order by
      salary desc
  ) as salary_rank
order by
  department,
  salary_rank
```

- 查询和分析结果

department	staff_name	salary	salary_rank
Marketing	Blan Stark	9000	1
Marketing	Smith	9000	2
Marketing	Achilles	8000	3
dev	Rob	9000	1
dev	Blan	8500	2
dev	Sansa	8000	3

first_value函数

first_value函数用于返回各个窗口分区内第一行的值。

```
first_value(x) over (
    [partition by partition_expression]
    [order by order_expression]
    [frame]
)
```

参数	说明
x	列名，可以为任意数据类型。
partition by *partition_expression*	窗口分区，根据分区表达式将数据划分成不同的分区。
order by *order_expression*	窗口排序，根据排序表达式对各个分区内的每一行进行排序。
frame	窗口框架，例如 `range between unbounded preceding and unbounded following` 。

与*x*的数据类型一致。

获取目标OSS Bucket中各个对象的最小值。

- 查询和分析语句

```
bucket :bucket90 |
select
  object,
  object_size,
  last_value(object_size) over (
    partition by object
    order by
      object_size
     range between unbounded preceding and unbounded following
  ) as last_value
from  oss-log-store
```

- 查询和分析结果

object	object_size	first_value
oss-log-56[redacted]11-cn-hangzhou%2Foss-log-store_oss_access_center_en%2Freport-1581933277-637788.	1157	1157
oss-log-562[redacted]11-cn-hangzhou%2Foss-log-store_oss_access_center_en%2Freport-1581933277-637788..	6751	1157
dashboard%2F2020%2F05%2F20%2F16%2F47.csv	6949	6949
dashboard%2F2020%2F05%2F20%2F16%2F47.csv	8749	6949
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	1195	1195
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	5775	1195
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	5856	1195

last_value函数

last_value函数用于返回各个窗口分区内最后一行的值。

```
last_value(x) over (
    [partition by partition_expression]
    [order by order_expression]
    [frame]
)
```

参数	说明
x	列名，可以为任意数据类型。
partition by *partition_expression*	窗口分区，根据分区表达式将数据划分成不同的分区。
order by *order_expression*	窗口排序，根据排序表达式对各个分区内的每一行进行排序。
frame	窗口框架，例如 `range between unbounded preceding and unbounded following` 。

与*x*的数据类型一致。

获取目标OSS Bucket中各个对象的最大值。

- 查询和分析语句

```
bucket :bucket90 |
select
  object,
  object_size,
  last_value(object_size) over (
    partition by object
    order by
      object_size
     range between unbounded preceding and unbounded following
  ) as last_value
from  oss-log-store
```

- 查询和分析结果

object	object_size	last_value
245-da918c.model	2383	6936
245-da918c.model	2975	6936
245-da918c.model	3375	6936
245-da918c.model	4999	6936
245-da918c.model	5199	6936
245-da918c.model	6125	6936
245-da918c.model	6936	6936
dashboard%2F2020%2F05%2F20%2F16%2F47.csv	2435	2603
dashboard%2F2020%2F05%2F20%2F16%2F47.csv	2603	2603

lag函数

lag函数用于返回窗口分区内位于当前行上方第*offset*行的值。

```
lag(x, offset, defaut_value) over (
    [partition by partition_expression]
    [order by order_expression]
    [frame]
)
```

参数	说明
x	列名，可以为任意数据类型。
offset	偏离量。如果*offset*为0，则返回当前行的值。
defaut_value	如果不存在指定的偏离行，则返回*defaut_value*。
partition by *partition_expression*	窗口分区，根据分区表达式将数据划分成不同的分区。
order by *order_expression*	窗口排序，根据排序表达式对各个分区内的每一行进行排序。
frame	窗口框架，例如 `range between unbounded preceding and unbounded following` 。

与*x*的数据类型一致。

按天统计网站访问UV，获取每天网站访问UV相比前一天的增长情况。

- 查询和分析语句

```
* |
select
  day,
  UV,
  UV * 1.0 /(lag(UV, 1, 0) over()) as diff_percentage
from  (
    select
      approx_distinct(client_ip) as UV,
      date_trunc('day', __time__) as day
    from  log
    group by
      day
    order by
      day asc
  )
```

- 查询和分析结果

day	UV	diff_percentage
2021-08-02 00:00:00.000	184332	Infinity
2021-08-03 00:00:00.000	386788	2.098322591845149
2021-08-04 00:00:00.000	377834	0.9768503676432567
2021-08-05 00:00:00.000	366409	0.9697618530889226
2021-08-06 00:00:00.000	390285	1.0651621548597333
2021-08-07 00:00:00.000	373103	0.9559757613026378
2021-08-08 00:00:00.000	386960	1.037139878264179
2021-08-09 00:00:00.000	226578	0.5855333884639239

lead函数

函数用于返回窗口分区内位于当前行下方第*offset*行的值。

```
lead(x, offset, defaut_value) over (
    [partition by partition_expression]
    [order by order_expression]
    [frame]
)
```

参数	说明
x	列名，可以为任意数据类型。
offset	偏离量。如果*offset*为0，则返回当前行的值。
defaut_value	如果不存在指定的偏离行，则返回*defaut_value*。
partition by *partition_expression*	窗口分区，根据分区表达式将数据划分成不同的分区。
order by *order_expression*	窗口排序，根据排序表达式对各个分区内的每一行进行排序。
frame	窗口框架，例如 `range between unbounded preceding and unbounded following` 。

与*x*的数据类型一致。

计算2021-08-26当天，当前一小时网站访问UV与后一小时的占比情况。

- 查询和分析语句

```
* |
select
  time,
  UV,
  UV * 1.0 /(lead(UV, 1, 0) over()) as diff_percentage
from  (
    select
      approx_distinct(client_ip) as uv,
      date_trunc('hour', __time__) as time
    from    log
    group by
      time
    order by
      time asc
  )
```

- 查询和分析结果

time	UV	diff_percentage
2021-08-26 00:00:00.000	1185	0.11647336347552585
2021-08-26 01:00:00.000	10174	0.37027331950358485
2021-08-26 02:00:00.000	27477	0.8010787172011662
2021-08-26 03:00:00.000	34300	1.3467352467705838
2021-08-26 04:00:00.000	25469	2.646404821280133
2021-08-26 05:00:00.000	9624	7.947151114781173
2021-08-26 06:00:00.000	1211	0.11748156771439658
2021-08-26 07:00:00.000	10308	0.37978041411834059

nth_value函数

nth_value函数用于返回窗口分区中第*offset*行的值。

```
nth_value(x, offset) over (
    [partition by partition_expression]
    [order by order_expression]
    [frame]
)
```

参数	说明
x	列名，可以为任意数据类型。
offset	偏离量。
partition by *partition_expression*	窗口分区，根据分区表达式将数据划分成不同的分区。
order by *order_expression*	窗口排序，根据排序表达式对各个分区内的每一行进行排序。
frame	窗口框架，例如 `range between unbounded preceding and unbounded following` 。

与*x*的数据类型一致。

按照部门分区，统计各个部门中薪水第二高的员工。

- 查询和分析语句

```
* |
select
  department,
  staff_name,
  salary,
  nth_value(staff_name, 2) over(
    partition by department
    order by
      sallary desc
      range between unbounded preceding and unbounded following
  ) as second_highest_sallary from log
```

- 查询和分析结果

department	staff_name	salary	second_highest_salary
dev	Rob	9000	Blan '
dev	Blan	8500	Blan
dev	Sansa	8000	Blan
dev	Snow	7000	Blan
Marketing	Achilles	8500	San
Marketing	San	8000	San
Marketing	Blan	7000	San

11.1.15. IP函数

本文介绍IP函数的基本语法及示例。

日志服务支持如下IP函数。

> 注意 在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。

函数类型	函数名称	语法	说明
	ip_to_city函数	ip_to_city(*x*)	分析目标IP地址所属城市。 返回结果为城市的中文名称。
		ip_to_city(*x*, 'en')	分析目标IP地址所属城市。 返回结果为城市的行政区划代码。
	ip_to_city_geo函数	ip_to_city_geo(*x*)	分析目标IP地址所属城市的经纬度。此函数返回的是城市经纬度，每个城市只有一个经纬度。
	ip_to_country函数	ip_to_country(*x*)	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。
		ip_to_country(*x*, 'en')	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代码。

IP地址函数 函数类型	函数名称	语法	说明
	ip_to_country_code函数	ip_to_country_code(*x*)	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代码。
	ip_to_domain函数	ip_to_domain(*x*)	判断目标IP地址是内网地址还是外网地址。
	ip_to_geo函数	ip_to_geo(*x*)	分析目标IP地址所在位置的经纬度。
	ip_to_provider函数	ip_to_provider(*x*)	分析目标IP地址所对应的网络运营商。
	ip_to_province函数	ip_to_province(*x*)	分析目标IP地址所属省份。 返回结果为省份的中文名称。
		ip_to_province(*x*, 'en')	分析目标IP地址所属省份。 返回结果为省份的行政区划代码。
IP网段函数	ip_prefix函数	ip_prefix(*x*, *prefix_bits*)	获取目标IP地址的前缀。
	is_prefix_subnet_of函数	is_prefix_subnet_of(*x*, *y*)	判断目标网段是否为某网段的子网。
	is_subnet_of函数	is_subnet_of(*x*, *y*)	判断目标IP地址是否在某网段内。
	ip_subnet_max函数	ip_subnet_max(*x*)	获取IP网段中的最大IP地址。
	ip_subnet_min函数	ip_subnet_min(*x*)	获取IP网段中的最小IP地址。
	ip_subnet_range函数	ip_subnet_range(*x*)	获取IP网段范围。

ip_to_city函数

ip_to_city函数用于分析目标IP地址所属城市。

- 返回城市的中文名称。

```
ip_to_city(x)
```

- 返回城市的行政区划代码。

```
ip_to_city(x,'en')
```

参数	说明
x	参数值为IP地址。

varchar类型。

统计来自不同城市的请求的平均时间、最大时间以及最大时间对应的请求ID。

- 查询和分析语句

```
* |
SELECT
  AVG(request_time) AS avg_request_time,
  MAX(request_time) AS max_request_time,
  MAX_BY(requestId, request_time) AS requestId,
  ip_to_city(client_ip) AS city
GROUP BY
  city
```

- 查询和分析结果

avg_request_time	max_request_time	requestId	city
44.84665110432753	80.0	i-02	天津市
46.47286821705426	80.0	i-01	丽江市
43.87214611872146	80.0	i-01	鹤岗市

ip_to_city_geo函数

ip_to_city_geo函数用于分析目标IP地址所属城市的经纬度。此函数返回的是城市经纬度，每个城市只有一个经纬度。

```
ip_to_city_geo(x)
```

参数	说明
x	参数值为IP地址。

varchar类型，格式为 `纬度,经度` 。

统计IP地址的经纬度，确认客户端分布情况。

- 查询和分析语句

```
* |
SELECT
  count(*) AS PV,
  ip_to_city_geo(client_ip) AS geo
GROUP BY
  geo
ORDER BY
  PV DESC
```

- 查询和分析结果

PV	geo
9113	39.9288,116.389
5784	31.2222,121.458060

ip_to_country函数

ip_to_country函数用于分析目标IP地址所属国家或地区。

- 返回国家或地区的中文名称。

```
ip_to_country(x)
```

- 返回国家或地区的代码。

```
ip_to_country(x,'en')
```

参数	说明
x	参数值为IP地址。

varchar类型。

统计来自不同国家或地区的请求的平均时间、最大时间以及最大时间对应的请求ID。

- 查询和分析语句

```
* |
SELECT
  AVG(request_time) AS avg_request_time,
  MAX(request_time) AS max_request_time,
  MAX_BY(requestId, request_time) AS requestId,
  ip_to_country(client_ip) AS country
GROUP BY
  country
```

- 查询和分析结果

avg_request_time	max_request_time	requestId	country
45.26589740373761	80.0	i-02	中国香港
47.07692307692308	80.0	i-01	澳大利亚
39.392857142857149	77.0	i-01	印度

ip_to_country_code函数

ip_to_country_code函数用于分析目标IP地址所属国家或地区，返回国家或地区的代码。

```
ip_to_country_code(x)
```

参数	说明
x	参数值为IP地址。

varchar类型。

统计来自不同国家或地区的请求的平均时间、最大时间以及最大时间对应的请求ID。

- 查询和分析语句

```
* |
SELECT
  AVG(request_time) AS avg_request_time,
  MAX(request_time) AS max_request_time,
  MAX_BY(requestId, request_time) AS requestId,
  ip_to_country_code(client_ip) AS country
GROUP BY
  country
```

- 查询和分析结果

avg_request_time	max_request_time	requestId	country
45.01220462217606	80.0	i-01	MO
46.285714285714288	77.0	i-02	GB

ip_to_domain函数

ip_to_domain函数用于判断目标IP地址是内网地址还是外网地址。

```
ip_to_domain(x)
```

参数	说明
x	参数值为IP地址。

varchar类型，返回intranet或internet。

- intranet表示内网。
- internet表示外网。

统计不是来自内网的请求总数。

- 查询和分析语句

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

- 查询和分析结果

ip_to_geo函数

ip_to_geo函数用于分析目标IP地址所在位置的经纬度。关于geohash函数的详细信息，请参见地理函数。

```
ip_to_geo(x)
```

参数	说明
x	参数值为IP地址。

varchar类型，格式为 `纬度,经度` 。

统计IP地址的经纬度，确认客户端分布情况。

- 查询和分析语句

```
* |
SELECT
  count(*) AS PV,
  ip_to_geo(client_ip) AS geo
GROUP BY
  geo
ORDER BY
  PV DESC
```

- 查询和分析结果

PV	geo
5122	39.1423,117.173
4960	29.5569,106.553

ip_to_provider函数

ip_to_provider函数用于分析目标IP地址对应的网络运营商。

```
ip_to_provider(x)
```

参数	说明
x	参数值为IP地址。

varchar类型。

统计通过不同网络运营商发起的请求的平均请求时间。

- 查询和分析语句

```
* |
SELECT
  avg(request_time) AS avg_request_time,
  ip_to_provider(client_ip) AS provider
GROUP BY
  provider
ORDER BY
  avg_request_time
```

- 查询和分析结果

avg_request_time	provider
18.0	信实通信
25.0	CAT
26.0	网宿

ip_to_province函数

ip_to_province函数分析目标IP地址所属省份。

- 返回省份的中文名称。

```
ip_to_province(x)
```

- 返回省份的行政区划代码。

```
ip_to_province(x,'en')
```

参数	说明
x	参数值为IP地址。

varchar类型。

统计请求总数Top10的省份。

- 查询和分析语句

```
* | SELECT count(*) as PV, ip_to_province(client_ip) AS province GROUP BY province ORDER BY PV des
c 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 count(*) AS PV, ip_to_province(client_ip) AS province WHERE ip_to_domain(client_ip) !=
'intranet'  GROUP BY province ORDER BY PV DESC LIMIT 10
```

- 查询和分析结果

PV	province
368	广东省
330	山东省
330	江苏省
323	北京市
308	香港特别行政区
255	台湾省
246	贵州省
220	四川省
219	
216	河北省

ip_prefix函数

ip_prefix函数用于获取目标IP地址的前缀。返回子网掩码格式的IP地址，例如192.168.1.0/24。

```
ip_prefix(x, prefix_bits)
```

参数	说明
x	参数值为IP地址。
prefix_bits	前缀位数。

varchar类型。

获取client_ip字段值的IP地址前缀。

- 查询和分析语句

```
* | SELECT ip_prefix(client_ip,24) AS client_ip
```

- 查询和分析结果

is_prefix_subnet_of函数

is_prefix_subnet_of函数用于判断目标网段是否为某网段的子网。

```
is_prefix_subnet_of(x, y)
```

参数	说明
x	参数值为IP网段。y网段是否属于x网段内。
y	参数值为IP网段。

boolean类型。

判断client_ip字段值所在网段是否属于192.168.0.1/24网段内。

- 查询和分析语句

```
* | SELECT is_prefix_subnet_of('192.168.0.1/24',concat(client_ip,'/24'))
```

- 查询和分析结果

is_subnet_of函数

is_subnet_of函数用于判断目标IP地址是否在某网段内。

```
is_subnet_of(x, y)
```

参数	说明
x	参数值为IP网段。
y	参数值为IP地址。

boolean类型。

判断client_ip字段值是否属于192.168.0.1/24网段内。

- 查询和分析语句

```
* | SELECT is_subnet_of('192.168.0.1/24',client_ip)
```

- 查询和分析结果

ip_subnet_min函数

ip_subnet_min函数用于获取IP网段中的最小IP地址。

```
ip_subnet_min(x)
```

参数	说明
x	参数值为IP网段。

varchar类型。

获取client_ip字段值所在网段的最小IP地址。

- 查询和分析语句

```
* | SELECT ip_subnet_min(concat(client_ip,'/24'))
```

- 查询和分析结果

ip_subnet_max函数

ip_subnet_max函数用于获取IP网段中最大IP地址。

```
ip_subnet_max(x)
```

vac

参数	说明
x	参数值为IP网段。

varchar类型。

获取client_ip字段值所在网段的最大IP地址。

- 查询和分析语句

```
* | SELECT ip_subnet_max(concat(client_ip,'/24'))
```

- 查询和分析结果

ip_subnet_range函数

ip_subnet_range用于获取IP网段范围。

```
ip_subnet_range(x)
```

参数	说明
x	参数值为IP网段。

JSON类型。

获取client_ip字段值所在网段的范围。

- 查询和分析语句

```
* | SELECT ip_subnet_range(concat(client_ip,'/24'))
```

- 查询和分析结果

11.1.16. URL函数

本文介绍URL函数的基本语法和示例。

日志服务支持如下URL函数。

> 注意
>
> - URL格式为 `[protocol:][//host[:port]][path][?query][#fragment]` 。
> - 在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。

函数名称	语法	说明
url_encode函数	url_encode(*x*)	对URL进行编码。
url_decode函数	url_decode(*x*)	对URL进行解码。
url_extract_fragment函数	url_extract_fragment(*x*)	从URL中提取Fragment信息。
url_extract_host函数	url_extract_host(*x*)	从URL中提取Host信息。
url_extract_parameter函数	url_extract_parameter(*x*, *parameter name*)	从URL的查询部分中提取指定参数的值。
url_extract_path函数	url_extract_path(*x*)	从URL中提取访问路径信息。
url_extract_port函数	url_extract_port(*x*)	从URL中提取端口信息。
url_extract_protocol函数	url_extract_protocol(*x*)	从URL中提取协议信息。
url_extract_query函数	url_extract_query(*x*)	从URL中提取查询部分的信息。

url_encode函数

url_encode函数用于对URL进行编码。

```
url_encode(x)
```

参数	说明
x	参数值为具体的URL地址。

varchar类型。

对url字段的值进行编码。

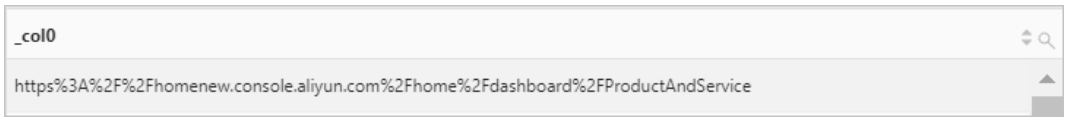
- 字段样例

```
url:https://homenew.console.aliyun.com/home/dashboard/ProductAndService
```

- 查询和分析语句

```
* | select url_encode(url)
```

- 查询和分析结果



url_decode函数

url_decode函数对URL进行解码。

```
url_decode(x)
```

参数	说明
x	参数值为编码过的URL地址。

varchar类型。

对url字段值进行解码。

- 字段样例

```
url:http%3A%2F%2Fwww.aliyun.com%3A80%2Fproduct%2Fsls
```

- 查询和分析语句

```
* | SELECT url_decode(url) AS decode
```

- 查询和分析结果

url_extract_fragment函数

url_extract_fragment函数用于从URL中提取Fragment信息。

```
url_extract_fragment(x)
```

参数	说明
x	参数值为具体的URL地址。

varchar类型。

从url字段值中提取Fragment信息。

- 字段样例

```
url:https://sls.console.aliyun.com/#/project/dashboard-demo/categoryList
```

- 查询和分析语句

```
* | SELECT url_extract_fragment(url)
```

- 查询和分析结果

url_extract_host函数

url_extract_host函数用于从URL中提取Host信息。

```
url_extract_host(x)
```

参数	说明
x	参数值为具体的URL地址。

varchar类型。

从url字段值中提取Host信息。

- 字段样例

```
url:https://homenew.console.aliyun.com/home/dashboard/ProductAndService
```

- 查询和分析语句

```
* | SELECT url_extract_host(url) AS host
```

- 查询和分析结果

url_extract_parameter函数

url_extract_parameter函数用于从URL的查询部分中提取指定参数的值。

```
url_extract_parameter(x, parameter name)
```

参数	说明
x	参数值为具体的URL地址。

参数	说明
parameter name	URL查询部分中的参数名称。

varchar类型。

从url字段值中提取accounttraceid参数的值。

- 字段样例

```
url:https://sls.console.aliyun.com/lognext/project/dashboard-all/logsearch/nginx-demo?accounttraceid=d6241a173f88471c91d3405cda010ff5ghdw
```

- 查询和分析语句

```
* | SELECT url_extract_parameter(url,'accounttraceid') AS accounttraceid
```

- 查询和分析结果

url_extract_path函数

url_extract_path用于从URL中提取访问路径信息。

```
url_extract_path(x)
```

参数	说明
x	参数值为具体的URL地址。

varchar类型。

从url字段值中提取访问路径信息。

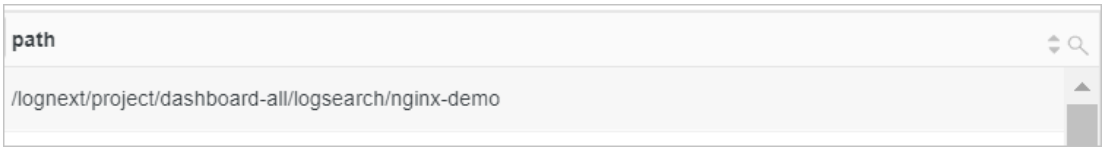
- 字段样例

```
url:https://sls.console.aliyun.com/lognext/project/dashboard-all/logsearch/nginx-demo?accounttraceid=d6241a173f88471c91d3405cda010ff5ghdw
```

- 查询和分析语句

```
* | SELECT url_extract_path(url) AS path
```

- 查询和分析结果



url_extract_port函数

url_extract_port函数用于从URL中提取端口信息。

```
url_extract_port(x)
```

参数	说明
x	参数值为具体的URL地址。

varchar类型。

从url字段值中提取端口信息。

- 字段样例

```
url:http://localhost:8080/lognext/profile
```

- 查询和分析语句

```
* | SELECT url_extract_port(url) AS port
```

- 查询和分析结果

url_extract_protocol函数

url_extract_protocol用于从URL中提取协议信息。

```
url_extract_port(x)
```

参数	说明
x	参数值为具体的URL地址。

varchar类型。

从url字段值中提取协议信息。

- 字段样例

```
url:https://homenew.console.aliyun.com/home/dashboard/ProductAndService
```

- 查询和分析语句

```
* | SELECT url_extract_protocol(url) AS protocol
```

- 查询和分析结果

url_extract_query函数

url_extract_query函数用于从URL中提取查询部分的信息。

```
url_extract_query(x)
```

参数	说明
x	参数值为具体的URL地址。

varchar类型。

从url字段值中提取查询部分的信息。

- 字段样例

```
url:https://sls.console.aliyun.com/lognext/project/dashboard-all/logsearch/nginx-demo?accounttrace
id=d6241a173f88471c91d3405cda010ff5ghdw
```

- 查询和分析语句

```
* | SELECT url_extract_query(url)
```

- 查询和分析结果

11.1.17. 估算函数

本文介绍估算函数的基本语法及示例。

日志服务支持如下估算函数。

> 注意 在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。

函数名称	语法	说明
approx_distinct函数	approx_distinct(*x*)	估算*x*中不重复值的个数，默认存在2.3%的标准误差。
	approx_distinct(*x*, *e*)	估算*x*中不重复值的个数，支持自定义标准误差。
approx_percentile函数	approx_percentile(*x*, *percentage*)	对*x*进行正序排列，返回大约处于*percentage*位置的*x*。
	approx_percentile(*x*, array[*percentage01*, *percentage02*...])	对*x*进行正序排列，返回大约处于*percentage01*、*percentage02*位置的*x*。
	approx_percentile(*x*, *weight*, *percentage*)	对*x*和权重的乘积进行正序排列，返回大约处于*percentage*位置的*x*。
	approx_percentile(*x*, *weight*, array[*percentage01*, *percentage02*...])	对*x*和权重的乘积进行正序排列，返回大约处于*percentage01*、*percentage02*位置的*x*。
	approx_percentile(*x*, *weight*, *percentage*, *accuracy*)	对*x*和权重的乘积进行正序排列，返回大约处于*percentage*位置的*x*。支持设置返回结果的准确度。
numeric_histogram函数	numeric_histogram(*bucket*, *x*)	按照bucket数量（直方图列数），统计*x*的近似直方图，返回结果为JSON类型。
	numeric_histogram(*bucket*, *x*, *weight*)	按照bucket数量（直方图列数），统计*x*的近似直方图，返回结果为JSON类型。支持对*x*设置权重。
numeric_histogram_u函数	numeric_histogram_u(*bucket*, *x*)	按照bucket数量（直方图列数），统计*x*的近似直方图，返回结果为多行多列格式。

approx_distinct函数

approx_distinct函数用于估算*x*中不重复值的个数。

- 估算*x*中不重复值的个数，默认存在2.3%的标准误差。

```
approx_distinct(x)
```

- 估算*x*中不重复值的个数，支持自定义标准误差。

```
approx_distinct(x, e)
```

参数	说明
x	参数值为任意数据类型。
e	自定义标准误差，取值为[0.0115, 0.26]。

bigint类型。

- 示例1：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，标准误差为2.3%。
  - 查询和分析语句

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

  - 查询和分析结果

PV	UV
941787	723040

- 示例2：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，自定义标准误差为10%。
  - 查询和分析语句

```
* |SELECT count(*) AS PV, approx_distinct(client_ip,0.1) AS UV
```

  - 查询和分析结果

PV	UV
9095	7946

approx_percentile函数

approx_percentile函数用于对*x*进行正序排列，返回大约处于*percentage*位置的数值。

- 对*x*进行正序排列，返回处于*percentage*位置的*x*，返回结果为double类型。

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

- 对*x*进行正序排列，返回处于percentage01、percentage02位置的*x*，返回结果为array(double,double)类型。

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

- 对*x*和权重的乘积进行正序排列，返回大约处于*percentage*位置的*x*，返回结果为double类型。

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

- 对*x*和权重的乘积进行正序排列，返回处于percentage01、percentage02位置的*x*，返回结果为array(double,double)类型。

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

- 对*x*和权重的乘积进行正序排列，返回大约处于*percentage*位置的*x*，返回结果为double类型。支持设置返回结果的准确度。

```
approx_percentile(x, weight, percentage, accuracy)
```

参数	说明
x	参数值为double类型。
percentage	百分比值，取值范围为[0,1]。
accuracy	准确度，取值范围为(0,1)。
weight	权重，大于1的整数。 设置权重后，系统根据*x*与权重的乘积进行排序。

double类型或array(double,double)类型。

- 示例1：对request_time列进行排列后，返回大约处于50%位置的request_time字段的值。
  - 查询和分析语句

```
*| SELECT approx_percentile(request_time,0.5)
```

  - 查询和分析结果

_col0
45.0

- 示例2：对request_time列进行排列后，返回处于10%、20%及70%位置的request_time字段的值。
  - 查询和分析语句

```
*| SELECT approx_percentile(request_time,array[0.1,0.2,0.7])
```

  - 查询和分析结果

_col0
[17.0,24.0,59.0]

- 示例3：根据request_time与权重的乘积对request_time列进行排列后，返回大约处于50%位置的request_time字段的值。其中，request_time<20时权重为100，否则权重为10。
  - 查询和分析语句

```
* |
SELECT
  approx_percentile(
    request_time,case
      when request_time < 20 then 100
      else 10
    end,
    0.5
  )
```

- 查询和分析结果

_col0
18.0

- 示例4：根据request_time与权重的乘积对request_time列进行排列后，返回大约处于80%和90%位置的request_time字段的值。其中，request_time<20时权重为100，否则权重为10。
  - 查询和分析语句

```
* |
SELECT
  approx_percentile(
    request_time,case
      when request_time < 20 then 100
      else 10
    end,
    array [0.8,0.9]
  )
```

  - 查询和分析结果

_col0
[48.0,64.0]

- 示例5：根据request_time与权重的乘积对request_time列进行排列后，返回大约处于50%位置的request_time字段的值，准确度为0.2。其中，request_time<20时权重为100，否则权重为10。
  - 查询和分析语句

```
* |
SELECT
  approx_percentile(
    request_time,case
      when request_time < 20 then 100
      else 10
    end,
    0.5,
    0.2
  )
```

  - 查询和分析结果

_col0
18.0

numeric_histogram函数

numeric_histogram函数按照bucket数量（直方图列数），统计*x*的近似直方图。返回结果为JSON类型。

- 按照bucket数量（直方图列数），统计*x*的近似直方图。

```
numeric_histogram(bucket, x)
```

- 按照bucket数量（直方图列数），统计*x*的近似直方图。支持为*x*设置权重。

```
numeric_histogram(bucket, x, weight)
```

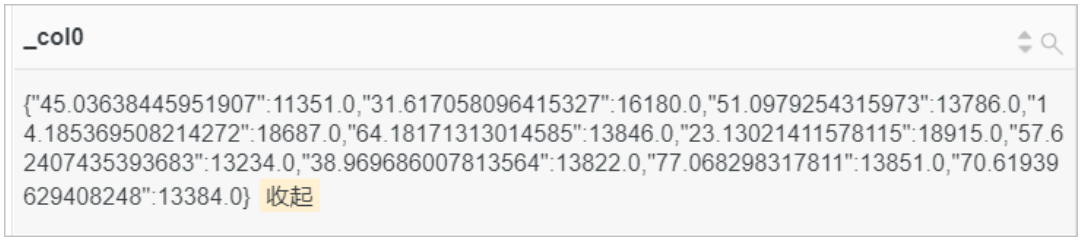
参数	说明
bucket	直方图中列的个数，bigint类型。
x	参数值为double类型。
weight	权重，大于0的整数。 设置权重后，系统根据*x*与权重的乘积进行分组。

JSON类型。

- 示例1：统计POST方法对应的请求时长的近似直方图。
  - 查询和分析语句

```
request_method:POST | SELECT numeric_histogram(10,request_time)
```

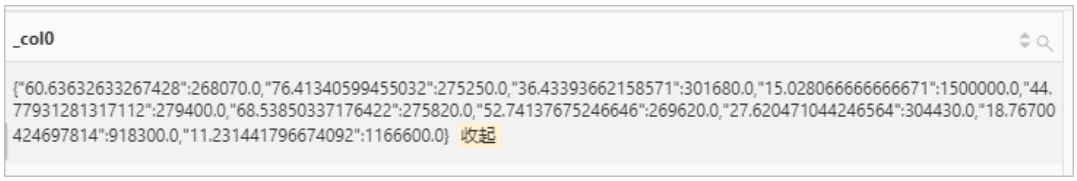
  - 查询和分析结果



- 示例2：根据request_time与权重的乘积对请求时长进行分组，从而统计POST方法对应的请求时长的近似直方图。
  - 查询和分析语句

```
request_method:POST| SELECT numeric_histogram(10, request_time,case when request_time<20 then 10 else 10 end)
```

  - 查询和分析结果



numeric_histogram_u函数

numeric_histogram_u函数按照bucket数量（直方图列数），统计*x*的近似直方图。返回结果为多行多列格式。

```
numeric_histogram_u(bucket, x)
```

参数	说明
bucket	直方图中列的个数，bigint类型。
x	参数值为double类型。

double类型。

统计POST方法对应的请求时长的近似直方图。

- 查询和分析语句

```
request_method:POST | select numeric_histogram_u(10,request_time)
```

- 查询和分析结果

bucket_avg	count
14.204509199191757	18806.0
22.91593094528485	17783.0
31.346545071904595	17106.0
38.84105278635488	13602.0
45.27102729998429	12674.0
51.33304216628288	12593.0
57.56119293078056	13580.0
64.28756532321565	13969.0
70.78247284630263	13442.0

11.1.18. 二进制函数

本文介绍二进制函数的基本语法及示例。

日志服务支持如下二进制函数。

> 注意
>
> - 在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。
> - varbinary类型是二进制字符类型，varchar类型是可变长度字符类型。

函数名称	语法	说明
from_base64函数	from_base64(*x*)	将BASE64编码的字符串解码为二进制类型的数据。
from_base64url函数	from_base64url(*x*)	使用URL安全字符将BASE64编码的字符串解码为二进制类型的数据。
from_big_endian_64函数	from_big_endian_64(*x*)	将大端模式的二进制类型的数据转化成数字。
from_hex函数	from_hex(*x*)	将十六进制类型的数据转化成二进制类型的数据。
length函数	length(*x*)	计算二进制类型的数据的长度。
md5函数	md5(*x*)	对二进制类型的数据进行MD5编码。
to_base64函数	to_base64(*x*)	对二进制类型的数据进行BASE64编码。

函数名称	语法	说明
to_base64url函数	to_base64url(*x*)	使用URL安全字符将二进制类型的数据进行BASE64编码。
to_hex函数	to_hex(*x*)	将二进制类型的数据转化成十六进制类型的数据。
to_big_endian_64函数	to_big_endian_64(*x*)	将数字转化为大端模式的二进制类型的数据。
sha1函数	sha1(*x*)	对二进制类型的数据进行SHA1加密。
sha256函数	sha256(*x*)	对二进制类型的数据进行SHA256加密。
sha512函数	sha512(*x*)	对二进制类型的数据进行SHA512加密。
xxhash64函数	xxhash64(*x*)	对二进制类型的数据进行xxHash64加密。

from_base64函数

from_base64函数用于将BASE64编码的字符串解码为二进制类型的数据。

```
from_base64(x)
```

参数	说明
x	参数值为binary类型。

varbinary类型。

> 注意 varbinary类型存在不可见字符，返回结果仍以BASE64编码格式展示。
> - 如果返回值是二进制类型的不可见字符，则您可以使用to_hex函数将其转换为十六进制类型的数据。
> - 如果返回值是二进制类型的可见字符，则您可以使用from_utf8函数将其转换为UTF-8字符串。

将BASE64编码的字符串解码为二进制类型的数据后，再转换为十六进制类型的数据。

- 查询和分析语句

```
* | SELECT to_hex(from_base64('c2xz'))
```

- 查询和分析结果

_col0
736C73

from_base64url函数

from_base64url函数使用URL安全字符将BASE64编码的字符串解码为二进制类型的数据。

```
from_base64url(x)
```

参数	说明
x	参数值为binary类型。

varbinary类型。

> 注意 varbinary类型存在不可见字符，返回结果仍以BASE64编码格式展示。
> - 如果返回值是二进制类型的不可见字符，则您可以使用to_hex函数将其转换为十六进制类型的数据。
> - 如果返回值是二进制类型的可见字符，则您可以使用from_utf8函数将其转换为UTF-8字符串。

使用URL安全字符将BASE64编码的字符串解码为二进制类型的数据。

- 查询和分析语句

```
* | SELECT to_hex(from_base64url('c2xz'))
```

- 查询和分析结果

_col0
736C73

from_big_endian_64函数

from_big_endian_64函数用于将大端模式的二进制类型的数据转化成数字。

```
from_big_endian_64(x)
```

参数	说明
x	参数值为binary类型。

bigint类型。

将大端模式的二进制类型的数据（10）转化成数字。

- 查询和分析语句

```
* | SELECT from_big_endian_64(to_big_endian_64(10))
```

- 查询和分析结果

_col0
10

from_hex函数

from_hex函数将十六进制类型的数据转化成二进制类型的数据。

```
from_hex(x)
```

参数	说明
x	参数值为varbinary类型。

varbinary类型。

将十六进制类型的数据（D74D）转换为二进制类型的数据。

- 查询和分析语句

```
* | SELECT from_hex('D74D')
```

- 查询和分析结果

_col0
100=

length函数

length函数用于计算二进制类型的数据的长度。

```
length(x)
```

参数	说明
x	参数值为binary类型。

bigint类型。

计算region字段值的长度。

- 查询和分析语句

```
* | SELECT length('00101000')
```

- 查询和分析结果

_col0
8

md5函数

md5函数用于对二进制类型的数据进行MD5编码。

```
md5(x)
```

参数	说明
x	参数值为varbinary类型。

varbinary类型。

对二进制类型的数据（1101）进行MD5编码。

- 查询和分析语句

```
* | SELECT MD5(from_base64('1101')) AS md5
```

- 查询和分析结果

md5
rG3lOanun9N57opubMnPDw==

to_base64函数

to_base64函数用于对二进制类型的数据进行BASE64编码。

```
to_base64(x)
```

参数	说明
x	参数值为binary类型。

varchar类型。

对二进制类型的数据（10）进行BASE64编码。

- 查询和分析语句

```
* | SELECT  to_base64(from_base64('10')) AS base64
```

- 查询和分析结果

base64
1w==

to_base64url函数

to_base64url函数使用URL安全字符对二进制类型的数据进行BASE64编码。

```
to_base64url(x)
```

参数	说明
x	参数值为binary类型。

varchar类型。

使用URL安全字符对二进制类型的数据（100）进行BASE64编码。

- 查询和分析语句

```
* | SELECT  to_base64url(from_base64('100'))
```

- 查询和分析结果

_col0
100=

to_hex函数

to_hex函数用于将二进制类型的数据转化成十六进制类型的数据。

```
to_hex(x)
```

参数	说明
x	参数值为binary类型。

varchar类型。

将二进制类型的数据（100）转化成十六进制类型的数据。

- 查询和分析语句

```
* | SELECT to_hex(from_base64('100'))
```

- 查询和分析结果

to_big_endian_64函数

to_big_endian_64函数用于将数字转化为大端模式的二进制类型的数据。

```
to_big_endian_64(x)
```

参数	说明
x	参数值为bigint类型。

varbinary类型。

将数字0转换为大端模式的二进制类型的数据。

- 查询和分析语句

```
* | SELECT to_big_endian_64(0)
```

- 查询和分析结果

sha1函数

sha1函数用于对二进制类型的数据进行SHA1加密。

```
sha1(x)
```

参数	说明
x	参数值为binary类型。

varbinary类型。

对二进制类型的数据（1101）进行SHA1加密。

- 查询和分析语句

```
* | SELECT sha1(from_base64('1101')) AS sha1
```

- 查询和分析结果

sha256函数

sha256函数用于对二进制类型的数据进行SHA256加密。

```
sha256(x)
```

参数	说明
x	参数值为binary类型。

varbinary类型。

对二进制类型的数据（1101）进行SHA256加密。

- 查询和分析语句

```
* | SELECT sha256(from_base64('1101')) AS sha256
```

- 查询和分析结果

sha256
wpRzHlu46qD4aJWFx3rxLSuTfMhX75dxpnThvydaDjg=

sha512函数

sha512函数用于对二进制类型的数据进行SHA512加密。

```
sha512(x)
```

参数	说明
x	参数值为binary类型。

varbinary类型。

对二进制类型的数据（1101）进行SHA512加密。

- 查询和分析语句

```
* | SELECT sha512(from_base64('1101')) AS sha512
```

- 查询和分析结果

sha512
K+2++tdKnpVZJMk4JYj4NMy4Mk5XVhlc/IN+rVju5OU0Yl79XBW4f9VzH+LF62O1A5WFz0cx7Yb9t1vhtg 8w==

xxhash64函数

xxhash64函数用于对二进制类型的数据进行xxHash64加密。

```
xxhash64(x)
```

参数	说明
x	参数值为binary类型。

varbinary类型。

对二进制类型的数据（10）进行xxhash64加密。

- 查询和分析语句

```
* | SELECT xxhash64(from_base64('10'))
```

- 查询和分析结果

11.1.19. 位运算函数

本文介绍位运算函数的基本语法及示例。

日志服务支持如下位运算函数。

> 注意 在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。

函数名称	语法	说明
bit_count函数	bit_count(*x*, *bits*)	统计*x*中1的个数。
bitwise_and函数	bitwise_and(*x*, *y*)	以二进制形式对*x*和*y*进行与运算。
bitwise_not函数	bitwise_not(*x*)	以二进制形式对*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*和*y*进行或运算。
bitwise_xor函数	bitwise_xor(*x*, *y*)	以二进制形式对*x*和*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。

```
bit_count(x, bits)
```

参数	说明
x	参数值为bigint类型。
bits	位数，例如64位。

bigint类型。

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。

```
bitwise_and(x, y)
```

参数	说明
x	参数值为bigint类型。
y	参数值为bigint类型。

bigint类型。

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函数以二进制的形式对*x*的所有位进行取反运算。

```
bitwise_not(x)
```

参数	说明
x	参数值为bigint类型。

bigint类型。

以二进制的形式对数字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。

```
bitwise_or(x, y)
```

参数	说明
x	参数值为bigint类型。
y	参数值为bigint类型。

bigint类型。

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

_col0
7

bitwise_xor函数

bitwise_xor函数以二进制形式对*x*和*y*进行异或运算。

```
bitwise_xor(x, y)
```

参数	说明
x	参数值为bigint类型。
y	参数值为bigint类型。

bigint类型。

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

```
?1* | SELECT bitwise_xor(3, 5)
```

- 查询和分析结果

_col0
6

11.1.20. 空间几何函数

本文介绍空间几何函数的基本语法及示例。

空间几何概念

以ST_前缀开头的空间几何函数支持SQL/MM标准并符合开放地理空间联盟 (OGC) 的OpenGIS规范。空间几何函数使用Well-Known Text（WKT）格式描述空间几何体（例如点、线段、多边形等），详细说明如下表所示。

空间几何体	Well-Known Text（WKT）格式
点	POINT (0 0)
线段	LINESTRING (0 0, 1 1, 1 2)
多边形	POLYGON((0 0, 4 0, 4 4, 0 4, 0 0), (1 1, 2 1, 2 2, 1 2, 1 1))
多点	MULTIPOINT(0 0, 1 2)
多线段	MULTILINESTRING((0 0, 1 1, 1 2), (2 3, 3 2, 5 4))
多个多边形	MULTIPOLYGON(((0 0, 4 0, 4 4, 0 4, 0 0), (1 1, 2 1, 2 2, 1 2, 1 1)), ((-1 -1, -1 -2, -2 -2, -2 -1, -1 -1)))
空间几何体集合	GEOMETRYCOLLECTION(POINT(2 3), LINESTRING(2 3, 3 4))

函数列表

分类	函数名称	语法	说明
构造空间实体	ST_AsText函数	ST_AsText(*x*)	将一个空间几何体转变为WKT格式的文本。
	ST_GeometryFromText函数	ST_GeometryFromText(*x*)	根据输入的WKT文本构造一个空间几何体。
	ST_LineFromText函数	ST_LineFromText(*x*)	根据输入的WKT文本构造一条线段。
	ST_Polygon函数	ST_Polygon(*x*)	根据输入的WKT文本构造一个多边形。
	ST_Point函数	ST_Point(*x, y*)	根据输入的WKT文本构造一个点。
运算符	ST_Boundary函数	ST_Boundary(*x*)	返回空间几何体的边界。
	ST_Buffer函数	ST_Buffer(*x, distance*)	返回距离指定空间几何体一定距离的空间几何体。
	ST_Difference函数	ST_Difference(*x, y*)	返回两个空间几何体不同点的集合。
	ST_Envelope函数	ST_Envelope(*x*)	返回空间几何体的最小边界框。
	ST_ExteriorRing函数	ST_ExteriorRing(*x*)	返回空间几何体的外环（线段形式）。
	ST_Intersection函数	ST_Intersection(*x, y*)	返回两个空间几何体的交集点。
	ST_SymDifference函数	ST_SymDifference(*x, y*)	返回两个空间几何体的不同点，然后组成一个新的空间几何体。
	ST_Contains函数	ST_Contains(*x, y*)	判断第一个空间几何体是否包含第二个空间几何体（边界可存在交集）。如果包含，则返回true。

分类	函数名称	语法	说明
空间关系判断	ST_Crosses函数	ST_Crosses(*x*, *y*)	判断两个空间几何体是否存在相同的内部点。如果存在，则返回true。
	ST_Disjoint函数	ST_Disjoint(*x*, *y*)	判断两个空间几何体是否没有任何交集。 如果没有，则返回true。
	ST_Equals函数	ST_Equals(*x*, *y*)	判断两个空间几何体是否完全相同。如果是，则返回true。
	ST_Intersects函数	ST_Intersects(*x*, *y*)	判断两个空间几何体的平面投影是否存在共同点。如果是，则返回true。
	ST_Overlaps函数	ST_Overlaps(*x*, *y*)	判断两个空间几何体的维度是否相同。如果两个空间几何体的维度相同且不是包含关系，则返回true。
	ST_Relate函数	ST_Relate(*x*, *y*, *patternMatrix string*)	判断两个空间几何体是否相关。如果是，则返回true。
	ST_Touches函数	ST_Touches(*x*, *y*)	判断两个空间几何体是否只有边界存在关联，没有共同的内部点。如果是，则返回true。
	ST_Within函数	ST_Within(*x*, *y*)	判断第一个空间几何体是否完全在第二个空间几何体内部（边界无交集）。如果是，则返回true。
Accessors	ST_Area函数	ST_Area(*x*)	使用欧几里得测量法计算空间几何体在二维平面上的投影面积。
	ST_Centroid函数	ST_Centroid(*x*)	返回空间几何实体的中心点。
	ST_CoordDim函数	ST_CoordDim(*x*)	返回空间几何体的坐标维度。
	ST_Dimension函数	ST_Dimension(*x*)	返回空间几何实体的固有维度，必须小于或等于坐标维度。
	ST_Distance函数	ST_Distance(*x*, *y*)	计算两个空间几何体之间的最小距离。
	ST_EndPoint函数	ST_EndPoint(*x*)	返回线段中的最后一个点。
	ST_IsClosed函数	ST_IsClosed(*x*)	判断输入的空间几何体是否封闭。如果是，则返回true。
	ST_IsEmpty函数	ST_IsEmpty(*x*)	判断输入的空间几何体是否为空。如果是，则返回true。
	ST_IsRing函数	ST_IsRing(*x*)	判断输入的空间几何体是否为闭合的简单线段（环）。如果是，则返回true。
	ST_Length函数	ST_Length(*x*)	使用欧几里得测量法计算线段的二维投影长度。如果存在多条线段，则返回所有线段的长度之和。
	ST_NumPoints函数	ST_NumPoints(*x*)	返回空间几何体中点的个数。

分类	函数名称	语法	说明
	ST_NumInteriorRing函数	ST_NumInteriorRing(*x*)	计算空间几何体中内部环的数量。
	ST_StartPoint函数	ST_StartPoint(*x*)	返回线段中的第一个点。
	ST_X函数	ST_X(*x*)	返回输入点的第一个X轴坐标。
	ST_XMax函数	ST_XMax(*x*)	返回空间几何体的第一个最大的X轴坐标。
	ST_XMin函数	ST_XMin(*x*)	返回空间几何体的第一个最小的X轴坐标。
	ST_Y函数	ST_Y(*x*)	返回输入点的第一个Y轴坐标。
	ST_YMax函数	ST_YMax(*x*)	返回空间几何体的第一个最大的Y轴坐标。
	ST_YMin函数	ST_YMin(*x*)	返回几何体的第一个最小的Y轴坐标。
Bing图块	bing_tile函数	bing_tile(*x, y, zoom_level*)	通过X坐标、Y坐标和缩放级别构造一个Bing图块。
		bing_tile(*quadKey*)	通过四叉树键构造一个Bing图块。
	bing_tile_at函数	bing_tile_at(*x, y, zoom_level*)	通过经纬度和缩放级别构造一个Bing图块。
	bing_tile_coordinates函数	bing_tile_coordinates(*x*)	返回目标Bing图块对应的X坐标和Y坐标。
	bing_tile_polygon函数	bing_tile_polygon(*x*)	返回目标Bing图块的多边形格式。
	bing_tile_quadkey函数	bing_tile_quadkey(*x*)	返回目标Bing图块的四叉树键。
	bing_tile_zoom_level函数	bing_tile_zoom_level(*x*)	返回目标Bing图块的缩放级别。

ST_AsText函数

ST_AsText函数用于将一个空间几何体转变成WKT文本。

```
ST_AsText(x)
```

参数	说明
x	参数值为geometry类型。

varchar类型。

将一个点转变成WKT格式的文本。

- 查询和分析语句

```
* | SELECT ST_AsText(ST_Point(1,1))
```

- 查询和分析结果

ST_GeometryFromText函数

ST_GeometryFromText函数会根据您输入的WKT文本构造一个空间几何体。

```
ST_GeometryFromText(x)
```

参数	说明
x	参数值为varchar类型。

geometry类型。

构造多个多边形。

- 查询和分析语句

```
* | SELECT ST_GeometryFromText('multipolygon(((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,
60 40,50 40)))')
```

- 查询和分析结果

ST_LineFromText函数

ST_LineFromText函数会根据您输入的WKT文本构造一条线段。

```
ST_LineFromText(x)
```

参数	说明
x	参数值为varchar类型。

linestring类型。

构造一条线段。

- 查询和分析语句

```
* | SELECT ST_LineFromText('linestring(10 10,20 20)')
```

- 查询和分析结果

ST_Polygon函数

ST_Polygon函数会根据您输入的WKT文本构造一个多边形。

```
ST_Polygon(x)
```

参数	说明
x	参数值为varchar类型。

polygon类型。

构造一个多边形。

- 查询和分析语句

```
* | SELECT ST_Polygon('polygon((10 10,10 20,20 20,20 15,10 10))')
```

- 查询和分析结果

_col0
POLYGON ((10 10, 20 15, 20 20, 10 20, 10 10))

ST_Point函数

ST_Point函数会根据您输入的WKT文本构造一个点。

```
ST_Point(x, y)
```

参数	说明
x	参数值为geometry类型。
y	参数值为geometry类型。

point类型。

构造一个点。

- 查询和分析语句

```
* | SELECT ST_Point(0,0)
```

- 查询和分析结果

_col0
POINT (0 0)

ST_Boundary函数

ST_Boundary函数用于返回空间几何体的边界。

- 点的边界为空，即返回POINT EMPTY。
- 线段的边界由线段的端点组成。
- 多边形的边界由构成多边形的外环及其所有内环的线段组成。

```
ST_Boundary(x)
```

参数	说明
x	参数值为geography类型。

geography类型。

使用ST_Polygon函数构造一个多边形，然后使用ST_Boundary返回多边形的边界。

- 查询和分析语句

```
* | SELECT  ST_Boundary(ST_Polygon('polygon((10 10,10 20,20 20,20 15,10 10))'))
```

- 查询和分析结果

ST_Buffer函数

ST_Buffer函数用于返回距离指定空间几何体一定距离的空间几何体。

```
ST_Buffer(x, distance)
```

参数	说明
x	参数值为geometry类型。
distance	距离。

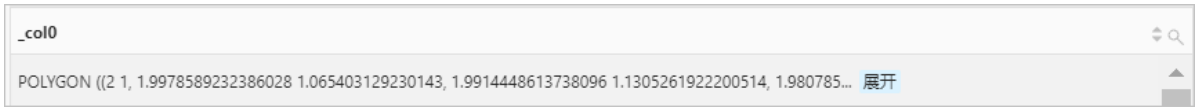
geometry类型。

使用ST_Point函数构造一个点，然后使用ST_Buffer函数返回距离该点一定距离的多边形。

- 查询和分析语句

```
* | SELECT ST_Buffer(ST_Point(1,1),1)
```

- 查询和分析结果



ST_Difference函数

ST_Difference函数用于返回两个空间几何体不同点的集合。

```
ST_Difference(x, y)
```

参数	说明
x	参数值为geometry类型。
y	参数值为geometry类型。

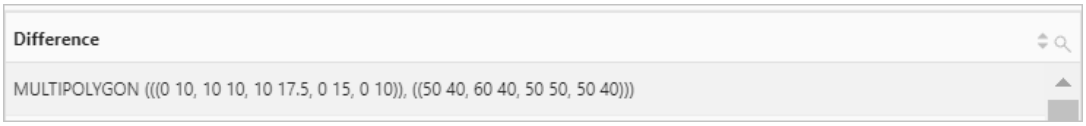
geometry类型。

使用ST_GeometryFromText函数构造两个空间几何体，然后使用ST_Difference函数返回两个空间几何体不同点的集合。

- 查询和分析语句

```
* |
SELECT
  ST_Difference(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,0 15,0 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  ) AS "Difference"
```

- 查询和分析结果



ST_Envelope函数

ST_Envelope函数用于返回空间几何体的最小边界框。

```
ST_Envelope(x)
```

参数	说明
x	参数值为geometry类型。

geometry类型。

使用ST_GeometryFromText函数构造一个空间几何体，然后使用ST_Envelope函数返回该空间几何体的最小边界框。

- 查询和分析语句

```
* |
SELECT
  ST_Envelope(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- 查询和分析结果

ST_ExteriorRing函数

ST_ExteriorRing函数用于返回空间几何体的外环（线段形式）。

```
ST_ExteriorRing(x)
```

参数	说明
x	参数值为geometry类型。

geometry类型。

使用ST_GeometryFromText函数构造一个空间几何体，然后使用ST_ExteriorRing函数返回该空间几何体的外环。

- 查询和分析语句

```
* |
SELECT
  ST_ExteriorRing(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- 查询和分析结果

_col0
LINESTRING (10 10, 20 15, 20 20, 10 20, 10 10)

ST_Intersection函数

ST_Intersection函数用于返回两个空间几何体的交集点。

```
ST_Intersection(x, y)
```

参数	说明
x	参数值为geometry类型。
y	参数值为geometry类型。

geometry类型。

使用ST_GeometryFromText函数构造两个空间几何体，然后使用ST_Intersection函数返回两个空间几何体的交集点。

- 查询和分析语句

```
* |
SELECT
  ST_Intersection(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- 查询和分析结果

_col0
MULTIPOLYGON (((10 10, 20 15, 20 20, 10 20, 10 10)), ((60 40, 60 50, 50 50, 60 40)))

ST_SymDifference函数

ST_SymDifference函数用于返回两个空间几何体的不同点，然后组成一个新的空间几何体。

```
ST_SymDifference(x, y)
```

参数	说明
x	参数值为geometry类型。
y	参数值为geometry类型。

geometry类型。

使用ST_GeometryFromText函数构造两个空间几何体，然后使用ST_SymDifference函数返回两个空间几何体的不同点，组成一个新的空间几何体。

- 查询和分析语句

```
* |
SELECT
  ST_SymDifference(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- 查询和分析结果

_col0
POLYGON ((50 40, 60 40, 50 50, 50 40))

ST_Contains函数

ST_Contains函数用于判断第一个空间几何体是否包含第二个空间几何体（边界可存在交集）。如果包含，则返回true。

```
ST_Contains(x, y)
```

参数	说明
x	参数值为geometry类型。
y	参数值为geometry类型。

boolean类型。

使用ST_GeometryFromText函数构造两个空间几何体，然后使用ST_Contains函数判断第一个空间几何体是否包含第二个空间几何体（边界可存在交集）。

- 查询和分析语句

```
* |
SELECT
  ST_Contains(
    ST_GeometryFromText(
      'polygon((10 10,10 20,20 20,20 15,10 10))'
    ),
    ST_GeometryFromText(
      'point(11 11)'
    )
  )
```

- 查询和分析结果

ST_Crosses函数

ST_Crosses函数用于判断两个空间几何体是否存在相同的内部点。如果存在，则返回true。

```
ST_Crosses(x, y)
```

参数	说明
x	参数值为geometry类型。
y	参数值为geometry类型。

boolean类型。

使用ST_GeometryFromText函数构造两个空间几何体，然后使用ST_Crosses函数判断两个空间几何体是否存在相同的内部点。

- 查询和分析语句

```
* |
SELECT
  ST_Crosses(
    ST_GeometryFromText(
      'multipolygon (((10 10, 10 20, 20 20, 20 15 , 10 10), (50 40, 50 50, 60 50, 60 40, 50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10, 10 20, 20 20, 20 15 , 10 10), (50 40, 50 50, 60 50, 60 40, 50 50)))'
    )
  )
```

- 查询和分析结果

ST_Disjoint函数

ST_Disjoint函数用于判断两个空间几何体是否没有任何交集。 如果没有，则返回true。

```
ST_Disjoint(x, y)
```

参数	说明
x	参数值为geometry类型。
y	参数值为geometry类型。

boolean类型。

使用ST_GeometryFromText函数构造两个空间几何体，然后使用ST_Disjoint函数判断两个空间几何体是否存在交集。

- 查询和分析语句

```
* |
SELECT
   ST_Disjoint(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- 查询和分析结果

ST_Equals函数

ST_Equals函数用于判断两个空间几何体是否完全相同。如果是，则返回true。

```
ST_Equals(x, y)
```

参数	说明
x	参数值为geometry类型。
y	参数值为geometry类型。

boolean类型。

使用ST_GeometryFromText函数构造两个空间几何体，然后使用ST_Equals函数判断两个空间几何体是否完全相同。

- 查询和分析语句

```
* |
SELECT
   ST_Equals(
    ST_GeometryFromText(
      'multipolygon(((10 10,10 20,20 20,20 15,10 10),(50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon(((10 10,10 20,20 20,20 15,10 10),(50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- 查询和分析结果

ST_Intersects函数

ST_Intersects函数用于判断两个空间几何体的平面投影是否存在共同点。如果是，则返回true。

```
ST_Intersects(x, y)
```

参数	说明
x	参数值为geometry类型。
y	参数值为geometry类型。

boolean类型。

使用ST_GeometryFromText函数构造两个空间几何体，然后使用ST_Intersects函数判断两个空间几何体的平面投影是否存在共同点。

- 查询和分析语句

```
* |
SELECT
   ST_Intersects(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- 查询和分析结果

ST_Overlaps函数

ST_Overlaps函数用于判断两个空间几何体的维度是否相同。如果两个空间几何体的维度相同且不是包含关系，则返回true。

```
ST_Overlaps(x, y)
```

参数	说明
x	参数值为geometry类型。
y	参数值为geometry类型。

boolean类型。

使用ST_GeometryFromText函数构造两个空间几何体，然后使用ST_Overlaps函数判断两个空间几何体的维度是否相同。

- 查询和分析语句

```
* |
SELECT
  ST_Overlaps(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- 查询和分析结果

ST_Relate函数

ST_Relate函数用于判断两个空间几何体是否相关（内部或边界以任何方式相关）。如果是，则返回true。

```
ST_Relate(x, y, patternMatrix string)
```

参数	说明
x	参数值为geometry类型。
y	参数值为geometry类型。
patternMatrix string	DE-9IM模式矩阵字符串，参数值为varchar类型。

boolean类型。

使用ST_GeometryFromText函数构造两个空间几何体，然后使用ST_Relate函数判断两个空间几何体是否相关。

- 查询和分析语句

```
* |
SELECT
  ST_Relate(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    ),  '****T****'
  )
```

- 查询和分析结果

ST_Touches函数

ST_Touches函数用于判断两个空间几何体是否只有边界存在关联，没有共同的内部点。如果是，则返回true。

```
ST_Touches(x, y)
```

参数	说明
x	参数值为geometry类型。
y	参数值为geometry类型。

boolean类型。

使用ST_GeometryFromText函数构造两个空间几何体，然后使用ST_Touches函数判断两个空间几何体是否只有边界存在关联，没有共同的内部点。

- 查询和分析语句

```
* |
SELECT
   ST_Touches(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- 查询和分析结果

_col0
false

ST_Within函数

ST_Within函数用于判断第一个空间几何体是否完全在第二个空间几何体内部（边界无交集）。如果是，则返回true。

```
ST_Within(x, y)
```

参数	说明
x	参数值为geometry类型。
y	参数值为geometry类型。

boolean类型。

使用ST_GeometryFromText函数构造两个空间几何体，然后使用ST_Within函数判断第一个空间几何体是否完全在第二个空间几何体内部。

- 查询和分析语句

```
* |
SELECT
  ST_Within(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- 查询和分析结果

_col0
false

ST_Area函数

ST_Area函数使用欧几里得测量法计算空间几何体在二维平面上的投影面积。

```
ST_Area(x)
```

参数	说明
x	参数值为geometry类型。

double类型。

使用ST_GeometryFromText函数构造一个空间几何体，然后使用ST_Area函数计算该空间几何体在二维平面上的投影面积。

- 查询和分析语句

```
* |
SELECT
  ST_Area(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- 查询和分析结果

_col0
-25.0

ST_Centroid函数

ST_Centroid函数用于返回空间几何体的中心点。

```
ST_Centroid(x)
```

参数	说明
x	参数值为geometry类型。

geometry类型。

使用ST_GeometryFromText函数构造一个空间几何体，然后使用ST_Centroid函数返回该空间几何体的中心点。

- 查询和分析语句

```
* |
SELECT
  ST_Centroid(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- 查询和分析结果

_col0
POINT (176.66666666666669 131.66666666666669)

ST_CoordDim函数

ST_CoordDim函数用于返回空间几何体的坐标维度。

```
ST_CoordDim(x)
```

参数	说明
x	参数值为geometry类型。

bigint类型。

使用ST_GeometryFromText函数构造一个空间几何体，然后使用ST_CoordDim函数返回该空间几何体的坐标维度。

- 查询和分析语句

```
* |
SELECT
  ST_CoordDim(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- 查询和分析结果

_col0
2

ST_Dimension函数

ST_Dimension函数用于返回空间几何体的固有维度，必须小于或等于坐标维度。

```
ST_Dimension(x)
```

参数	说明
x	参数值为geometry类型。 • *x*为点或空的空间几何体时，返回值为0。 • *x*为线段时，返回值为1。 • *x*为多边形时，返回值为2。 • *x*为空间几何体时，返回值为集合的最大维度。

bigint类型。

使用ST_GeometryFromText函数构造一个空间几何体，然后使用ST_Dimension函数返回该空间几何体的固有维度。

- 查询和分析语句

```
* |
SELECT
  ST_Dimension(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- 查询和分析结果

_col0
2

ST_Distance函数

ST_Distance函数用于计算两个空间几何体之间的最小距离。

```
ST_Distance(x, y)
```

参数	说明
x	参数值为geometry类型。
y	参数值为geometry类型。

double类型。

使用ST_GeometryFromText函数构造两个空间几何体，然后使用ST_Distance函数计算两个空间几何体之间的最小距离。

- 查询和分析语句

```
* |
SELECT
  ST_Distance(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- 查询和分析结果

_col0
0.0

ST_EndPoint函数

ST_EndPoint函数用于返回线段中的最后一个点。

```
ST_EndPoint(x)
```

参数	说明
x	参数值为geometry类型。

point类型。

使用ST_LineFromText函数构造一条线段，然后使用ST_EndPoint函数返回线段中的最后一个点。

- 查询和分析语句

```
* |
SELECT
  ST_EndPoint(
    ST_LineFromText(
      'linestring (10 10,20 20)'
    )
  )
```

- 查询和分析结果

ST_IsClosed函数

ST_IsClosed函数用于判断空间几何体是否封闭。如果是，则返回true。

```
ST_IsClosed(x)
```

参数	说明
x	参数值为geometry类型。

boolean类型。

使用ST_LineFromText函数构造一条线段，然后使用ST_IsClosed函数判断该线段是否封闭。

- 查询和分析语句

```
* |
SELECT
  ST_IsClosed(
    ST_LineFromText(
      'linestring (10.05 10.28 , 20.95 20.89 )'
    )
  )
```

- 查询和分析结果

ST_IsEmpty函数

ST_IsEmpty函数用于判断输入的空间几何体是否为空。如果是，则返回true。

```
ST_IsEmpty(x)
```

参数	说明
x	参数值为geometry类型。

boolean类型。

使用ST_Point函数构造一个点，然后使用ST_IsEmpty函数判断该点是否为空。

- 查询和分析语句

```
* | SELECT ST_IsEmpty(ST_Point(1,1))
```

- 查询和分析结果

ST_IsRing函数

ST_IsRing函数用于判断输入的空间几何体是否为闭合的简单线段（环）。如果是，则返回true。

```
ST_IsRing(x)
```

参数	说明
x	参数值为geometry类型。

boolean类型。

使用ST_LineFromText函数构造一条线段，然后使用ST_IsRing函数判断该线段是否为一个环。

- 查询和分析语句

```
* |
SELECT
  ST_IsRing(
    ST_LineFromText(
      'linestring (10.05 10.28,20.95 20.89 )'
    )
  )
```

- 查询和分析结果

_col0
false

ST_Length函数

ST_Length函数使用欧几里得测量法计算线段的二维投影长度。如果存在多条线段，则返回所有线段的长度之和。

```
ST_Length(x)
```

参数	说明
x	参数值为geometry类型。

double类型。

使用ST_LineFromText函数构造一条线段，然后使用ST_Length函数计算线段的长度。

- 查询和分析语句

```
* |
SELECT
  ST_Length(
    ST_LineFromText(
      'linestring (10.05 10.28,20.95 20.89)'
    )
  )
```

- 查询和分析结果

_col0
15.211249126879752

ST_NumPoints函数

ST_NumPoints函数用于返回空间几何体中点的个数。

```
ST_NumPoints(x)
```

参数	说明
x	参数值为geometry类型。

bigint类型。

使用ST_LineFromText函数构造一条线段，然后使用ST_NumPoints函数返回线段中点的个数。

- 查询和分析语句

```
* |
SELECT
  ST_NumPoints(
    ST_LineFromText('linestring (10 10,20 20)')
  )
```

- 查询和分析结果

_col0
2

ST_NumInteriorRing函数

ST_NumInteriorRing函数用于计算空间几何体中内部环的数量。

```
ST_NumInteriorRing(x)
```

参数	说明
x	参数值为geometry类型。

bigint类型。

使用ST_GeometryFromText函数构造一个空间几何体，然后使用ST_NumInteriorRing函数返回该几何体中内部环的数量。

- 查询和分析语句

```
* |
SELECT
  ST_NumInteriorRing(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- 查询和分析结果

_col0
1

ST_StartPoint函数

ST_StartPoint函数用于返回线段中的第一个点。

```
ST_StartPoint(x)
```

参数	说明
x	参数值为geometry类型。

point类型。

使用ST_LineFromText函数构造一条线段，然后使用ST_StartPoint函数返回该线段中的第一个点。

- 查询和分析语句

```
* |
SELECT
  ST_StartPoint(
    ST_LineFromText(
      'linestring (10 10,20 20 )'
    )
  )
```

- 查询和分析结果

_col0
POINT (10 10)

ST_X函数

ST_X函数用于返回输入点的X轴坐标。

```
ST_X(x)
```

参数	说明
x	参数值为point类型。

double类型。

使用ST_Point函数构造一个点，然后使用ST_X函数返回该点的X轴坐标。

- 查询和分析语句

```
* | SELECT ST_X(ST_Point(1,3))
```

- 查询和分析结果

_col0
1.0

ST_XMax函数

ST_XMax函数用于返回空间几何体的第一个最大的X轴坐标。

```
ST_XMax(x)
```

参数	说明
x	参数值为geometry类型。

double类型。

使用ST_GeometryFromText函数构造一个空间几何体，然后使用ST_XMax函数返回该空间几何体的第一个最大的X轴坐标。

- 查询和分析语句

```
* |
SELECT
  ST_XMax(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- 查询和分析结果

_col0
60.0

ST_XMin函数

ST_XMin函数用于返回空间几何体的第一个最小的X轴坐标。

```
ST_XMin(x)
```

参数	说明
x	参数值为geometry类型。

double类型。

使用ST_GeometryFromText函数构造一个空间几何体，然后使用ST_XMin函数返回该空间几何体的第一个最小的X轴坐标。

- 查询和分析语句

```
* |
SELECT
  ST_XMin(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- 查询和分析结果

_col0
10.0

ST_Y函数

ST_Y函数用于返回输入点的Y轴坐标。

```
ST_Y(x)
```

参数	说明
x	参数值为point类型。

double类型。

使用ST_Point函数构造一个点，然后使用ST_Y函数返回该点的Y轴坐标。

- 查询和分析语句

```
* | SELECT ST_Y(ST_Point(1,3))
```

- 查询和分析结果

ST_YMax函数

ST_YMax函数用于返回空间几何体的第一个最大的Y轴坐标。

```
ST_YMax(x)
```

参数	说明
x	参数值为geometry类型。

double类型。

使用ST_GeometryFromText函数构造一个空间几何体，然后使用ST_YMax函数返回该空间几何体的第一个最大的Y轴坐标。

- 查询和分析语句

```
* |
SELECT
  ST_YMax(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- 查询和分析结果

ST_YMin函数

ST_YMin函数用于返回空间几何体的第一个最小的Y轴坐标。

```
ST_YMin(x)
```

参数	说明
x	参数值为geometry类型。

double类型。

使用ST_GeometryFromText函数构造一个空间几何体，然后使用ST_YMin函数返回空间几何体的第一个最小的Y轴坐标。

- 查询和分析语句

```
* |
SELECT
  ST_YMin(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- 查询和分析结果

bing_tile函数

bing_tile函数用于构造一个Bing图块。

- 通过X坐标、Y坐标和缩放级别构造一个Bing图块。

```
bing_tile(x, y, zoom_level)
```

- 通过四叉树键构造一个Bing图块。

```
bing_tile(quadKey)
```

参数	说明
x	X坐标，参数值为integer类型。
y	Y坐标，参数值为integer类型。
zoom_level	缩放级别，取值范围为[1,23]，参数值为integer类型。
quadKey	四叉树键。

BingTile类型。

- 示例1：通过X坐标、Y坐标和缩放级别构造一个Bing图块。
  - 查询和分析语句

```
* | SELECT bing_tile(10, 20, 20)
```

  - 查询和分析结果

- 示例2：通过四叉树键构造一个Bing图块。

- 查询和分析语句

```
* | SELECT bing_tile(bing_tile_quadkey(bing_tile(10, 20, 20)))
```

- 查询和分析结果

bing_tile_at函数

bing_tile_at函数通过经纬度和缩放级别构造一个Bing图块。

```
bing_tile_at(x, y, zoom_level)
```

参数	说明
x	纬度，取值范围为[-85.05112878,85.05112878]，参数值为double类型。
y	经度，取值范围为[-180,180]，参数值为double类型。
zoom_level	缩放级别，取值范围为[1,23]，参数值为integer类型。

BingTile类型。

创建一个Bing图块。

- 查询和分析语句

```
* | SELECT bing_tile_at(47.265511, -122.465691, 12)
```

- 查询和分析结果

bing_tile_coordinates函数

bing_tile_coordinates函数用于返回目标Bing图块对应的X坐标和Y坐标。

```
bing_tile_coordinates(x)
```

参数	说明
x	参数值为BingTile类型。

array(integer,integer)类型。

通过输入的Bing图块返回对应的X坐标和Y坐标。

- 查询和分析语句

```
* | SELECT bing_tile_coordinates(bing_tile_at(47.265511, -122.465691, 12))
```

- 查询和分析结果

bing_tile_polygon函数

bing_tile_polygon函数用于返回目标Bing图块的多边形格式。

```
bing_tile_polygon(x)
```

参数	说明
x	参数值为BingTile类型。

polygon类型。

返回Bing图块的多边形格式。

- 查询和分析语句

```
* | SELECT bing_tile_polygon(bing_tile_at(30.26, 120.19, 12))
```

- 查询和分析结果

bing_tile_quadkey函数

bing_tile_quadkey函数用于返回目标Bing图块的四叉树键。

```
bing_tile_quadkey(x)
```

参数	说明
x	参数值为BingTile类型。

varchar类型。

返回目标Bing图块的四叉树键。

- 查询和分析语句

```
* | SELECT bing_tile_quadkey(bing_tile(10, 20, 20))
```

- 查询和分析结果

bing_tile_zoom_level函数

bing_tile_zoom_level函数用于返回目标Bing图块的缩放级别。

```
bing_tile_zoom_level(x)
```

参数	说明
x	参数值为BingTile类型。

double类型。

返回目标Bing图块的缩放级别。

- 查询和分析语句

```
* | SELECT bing_tile_zoom_level(bing_tile(10, 20, 20))
```

- 查询和分析结果

11.1.21. 地理函数

本文介绍地理函数的基本语法及示例。

日志服务支持如下地理函数。

> 注意 在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。

函数名称	语法	说明
geohash函数	geohash(*x*)	对纬度和经度进行geohash编码。 说明 日志服务支持将IP地址转换为国家、省份、城市、运营商或经纬度等信息。更多信息，请参见IP函数。

geohash函数

geohash函数用于对纬度和经度进行geohash编码。

```
geohash(x)
```

参数	说明
x	参数值为string类型。内容为经纬度，例如`ip_to_geo(经度,纬度)`。

string类型。

使用ip_to_geo函数将client_ip字段值转换为经纬度形式，再使用geohash函数进行编码。

- 查询和分析语句

```
* | SELECT geohash(ip_to_geo(client_ip)) AS hash
```

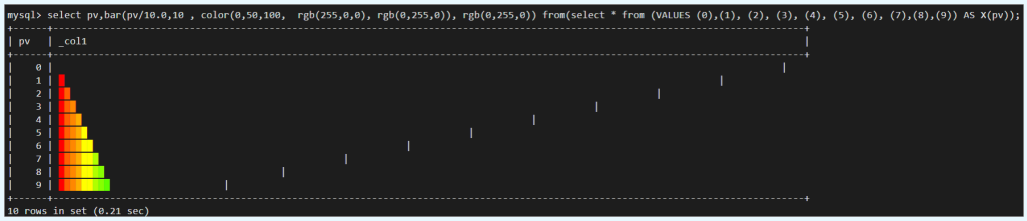
- 查询和分析结果

hash
wwgqdmdu2v96

11.1.22. 颜色函数

本文介绍颜色函数的基本语法及示例。

日志服务支持如下颜色函数。

> 注意
>
> - 在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。
> - 在日志服务控制台上使用颜色函数时，查询和分析结果的可视化效果并不理想，推荐您在终端服务器上展示查询和分析结果。
>   - 控制台展示效果
>
> | _col0 |
> | --- |
> | □[38;5;1m□□[0m |
>
>   - 终端展示效果
>
> 
>

函数名称	语法	说明
bar函数	bar(*x*, *width*)	通过*width*指定整条ANSI条形图的宽度，其中该ANSI条形图的起始颜色为红色（low_color），结束颜色为绿色（high_color）。然后通过*x*截取其中一段ANSI条形图并返回。
	bar(*x*, *width*, *low_color*, *high_color*)	通过*width*指定整条ANSI条形图的宽度，其中该ANSI条形图的起始颜色和结束颜色为自定义颜色。然后通过*x*截取其中一段ANSI条形图并返回。
color函数	color(string)	将颜色字符串转换为color类型。
	color(*x*, *low*, *high*, *low_color*, *high_color*)	通过判断*x*在*low*和*high*之间的占比指定*low_color*和*high_color*的份量，然后返回处于*low_color*和*high_color*之间的一个颜色。
	color(*y*, *low_color*, *high_color*)	通过*y*指定*low_color*和*high_color*的份量，然后返回处于*low_color*和*high_color*之间的一个颜色。

函数名称	语法	说明
render函数	render(*boolean expression*)	通过颜色渲染返回结果。布尔表达式为真时，返回绿色勾；否则返回红色叉。
	render(*x*, *color*)	通过自定义的颜色渲染返回结果。
rgb函数	rgb(*red*, *green*, *blue*)	通过RGB值返回一个颜色值。

bar函数

bar函数用于绘制一条ANSI条形图。

- 通过*width*指定整条ANSI条形图的宽度，其中该ANSI条形图的起始颜色为红色（low_color），结束颜色为绿色（high_color）。然后通过*x*截取其中一段ANSI条形图并返回。

```
bar(x, width)
```

- 通过*width*指定整条ANSI条形图的宽度，其中该ANSI条形图的起始颜色和结束颜色为自定义颜色。然后通过*x*截取其中一段ANSI条形图并返回。

```
bar(x, width, low_color, high_color)
```

参数	说明
x	用于指定返回的条形图占整条ANSI条形图的比例。参数值为double类型，取值范围为[0,1]。
width	整条ANSI条形图的宽度。
low_color	颜色梯度中起始颜色的RGB值。
high_color	颜色梯度中结束颜色的RGB值。

varchar类型。

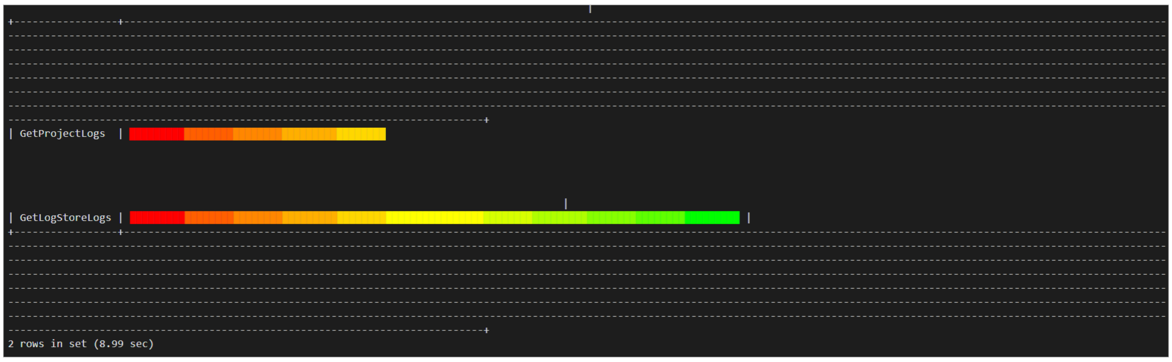
- 示例1：计算1小时PV占总PV的比值，然后通过ANSI条形图表示计算结果。
  - 查询和分析语句

```
* |
SELECT
  Method,
  bar(pv/m,100)
FROM(
    SELECT
      *,
      max(pv) over() AS m
    FROM(
        SELECT
          Method,
          count(1) AS pv
        FROM         internal-operation_log
        WHERE
          __date__ > '2021-09-10 00:00:00'
          AND __date__ < '2021-09-10 01:00:00'
        GROUP BY
          Method
      )
  )
```

- 查询和分析结果（控制台）

Method	_col1
GetProjectLogs	□[38;5;196m█□[38;5;196m█□[38;5;196m█□[38;5;196m█□[38;5;196m█□[38;5;196m█□[38;5;196m█□[38;5;196m█□[38... 展开
GetLogStoreLogs	□[38;5;196m█□[38;5;202m█□[38;5;214m█□[38;5;220m█□[38;5;226m█□[38;5;190m█□[38;5;154m█□[38;5;82m█□[38;... 展开

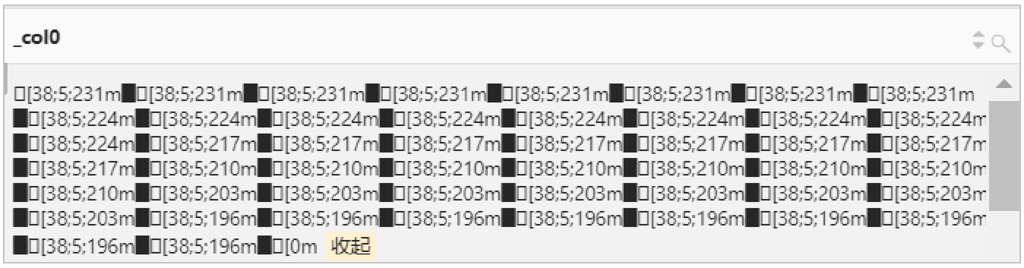
- 查询和分析结果（终端）



- 示例2：使用白色和红色绘制一条ANSI条形图，宽度为50。
  - 查询和分析语句

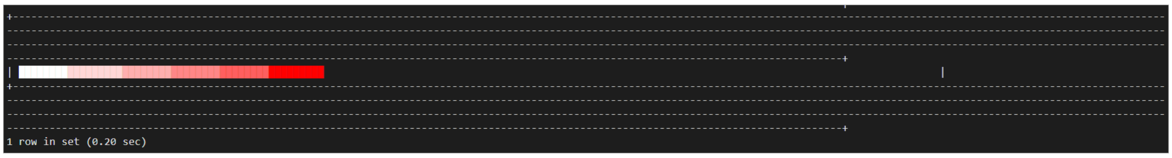
```
* | SELECT bar(1,50,rgb(255,255,255),rgb(255,0,0))
```

  - 查询和分析结果（控制台）



_col0
□[38;5;231m█□[38;5;231m█□[38;5;231m█□[38;5;231m█□[38;5;231m█□[38;5;231m█□[38;5;231m█□[38;5;231m█□[38;5;224m█□[38;5;224m█□[38;5;224m█□[38;5;224m█□[38;5;224m█□[38;5;224m█□[38;5;224m█□[38;5;224m█□[38;5;224m█□[38;5;217m█□[38;5;217m█□[38;5;217m█□[38;5;217m█□[38;5;217m█□[38;5;217m█□[38;5;217m█□[38;5;217m█□[38;5;210m█□[38;5;210m█□[38;5;210m█□[38;5;210m█□[38;5;210m█□[38;5;210m█□[38;5;210m█□[38;5;210m█□[38;5;203m█□[38;5;203m█□[38;5;203m█□[38;5;203m█□[38;5;203m█□[38;5;203m█□[38;5;203m█□[38;5;203m█□[38;5;196m█□[38;5;196m█□[38;5;196m█□[38;5;196m█□[38;5;196m█□[38;5;196m█□[38;5;196m█□[38;5;196m█□[38;5;196m█□[0m 收起

  - 查询和分析结果（终端）



color函数

color函数用于获取目标值对应的颜色。

- 将颜色字符串转换为color类型。

```
color(string)
```

- 通过判断*x*在*low*和*high*之间的占比指定*low_color*和*high_color*的份量，然后返回处于*low_color*和*high_color*之间的一个颜色。

```
color(x, low, high, low_color, high_color)
```

- 通过*y*指定*low_color*和*high_color*的份量，然后返回处于*low_color*和*high_color*之间的一个颜色。

```
color(y, low_color, high_color)
```

参数	说明
x	参数值为double类型。
y	参数值为double类型，取值范围为[0,1]。
low	最小值，参数值为double类型。
high	最大值，参数值为double类型。
low_color	颜色梯度中起始颜色的RGB值。
high_color	颜色梯度中结束颜色的RGB值。
string	字符串，可选值为black、red、green、yellow、blue、magenta、cyan、white或CSS样式的RGB值（例如#000）。

color类型。

- 示例1：将颜色字符串转换为color类型。
  - 查询和分析语句

```
* | SELECT color('#000')
```

  - 查询和分析结果（控制台）

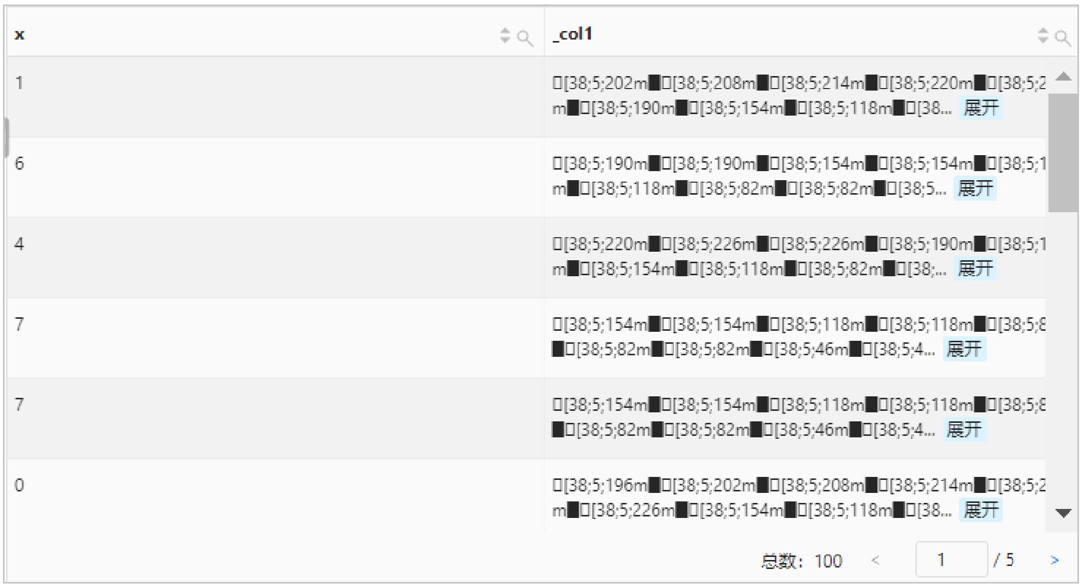
  - 查询和分析结果（终端）

```
mysql> SELECT color('#000');
+---------+
| _col0   |
+---------+
| #000000 |
+---------+
1 row in set (0.21 sec)
```

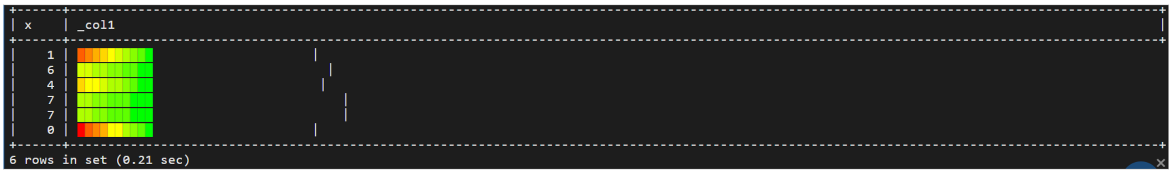
- 示例2：对request_length字段值进行取余计算并将计算结果传递给color函数，color函数根据该计算结果返回一个颜色，然后将该颜色传递给bar函数，最后通过bar函数绘制出相应的ANSI条形图。
  - 查询和分析语句

```
*|SELECT x,bar(10,10, color(x, 0,10, rgb(255,0,0), rgb(0,255,0)), rgb(0,255,0)) FROM(SELECT  *FR
OM (SELECT  request_length%10 x FROM  log))
```

- 查询和分析结果（控制台）



- 查询和分析结果（终端）

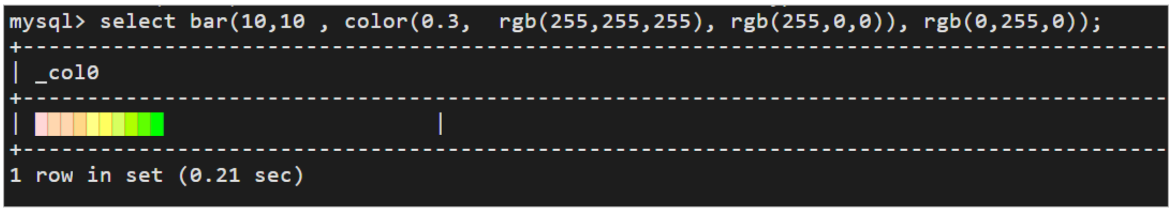


- 示例3：通过color函数返回一种颜色，然后传递给bar函数，绘制出相应的ANSI条形图。
  - 查询和分析语句

```
*|SELECT bar(10,10, color(0.3, rgb(255,255,255), rgb(255,0,0)), rgb(0,255,0))
```

  - 查询和分析结果（控制台）

  - 查询和分析结果（终端）



render函数

render函数通过颜色渲染返回结果。

- 通过颜色渲染返回结果。布尔表达式为真时，返回绿色勾；否则返回红色叉。

```
render(boolean expression)
```

- 通过自定义的颜色渲染返回结果。

```
render(x, color)
```

参数	说明
boolean expression	布尔表达式。
x	X坐标，参数值为integer类型。
color	颜色，参数值为color类型。

varchar类型。

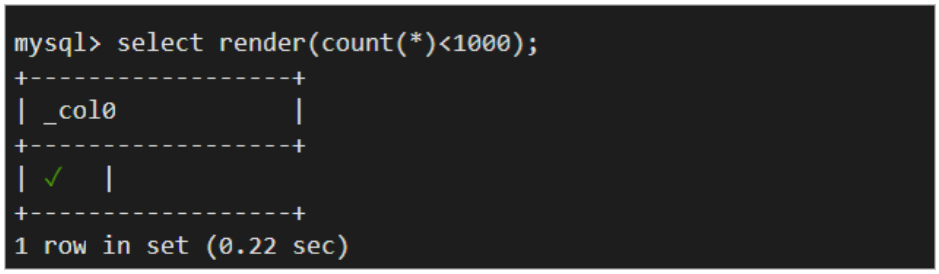
- 示例1：通过count函数统计网站访问PV，然后通过render函数判断PV是否小于1000。如果小于，则返回绿色勾。
  - 查询和分析语句

    ```
    * | SELECT render(count(*)<1000)
    ```

  - 查询和分析结果（控制台）

  - 查询和分析结果（终端）

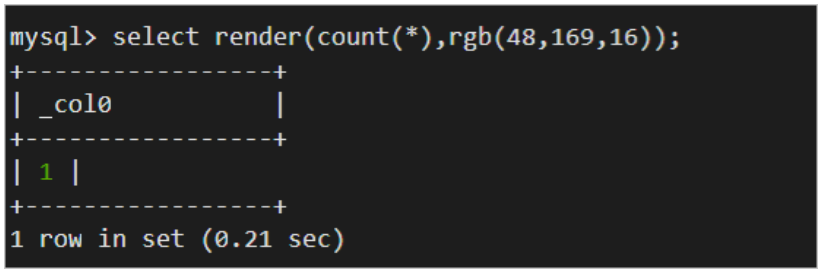
- 示例2：通过count函数统计日志总数，然后使用绿色渲染计算结果。
  - 查询和分析语句

    ```
    * | SELECT render(count(*),rgb(48,169,16))
    ```

  - 查询和分析结果（控制台）

  - 查询和分析结果（终端）

rgb函数

rgb函数会根据RGB值返回一个颜色。

```
rgb(red, green, blue)
```

参数	说明
red	颜色中的红色份量，取值范围为[0,255]，参数值为integer类型。
green	颜色中的绿色份量，取值范围为[0,255]，参数值为integer类型。
blue	颜色中的蓝色份量，取值范围为[0,255]，参数值为integer类型。

color类型。

根据RGB值返回一个颜色。

- 查询和分析语句

```
*|SELECT rgb(255,0,0)
```

- 查询和分析结果（控制台）

- 查询和分析结果（终端）

```
mysql> select rgb(255,0,0);
+---------+
| _col0   |
+---------+
| #ff0000 |
+---------+
1 row in set (0.21 sec)
```

11.1.23. HyperLogLog函数

HyperLogLog函数为近似聚合函数，类似于approx_distinct函数。当计算的数据量非常大时，使用HyperLogLog函数可快速返回估算结果。本文介绍HyperLogLog函数的基本语法以及示例。

日志服务支持如下HyperLogLog函数。

> 注意 在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。

函数名称	语法	说明
approx_set函数	approx_set(*x*)	估算*x*中不重复值的个数，最大标准误差默认为0.01625。
cardinality函数	cardinality(*x*)	将HyperLogLog类型的内容转换为bigint类型。
empty_approx_set函数	empty_approx_set()	返回一个HyperLogLog类型的空值。最大标准误差默认为0.01625。
merge函数	merge(*x*)	聚合计算所有的HyperLogLog值。

approx_set函数

approx_set函数用于估算*x*中不重复值的个数，最大标准误差默认为0.01625。

```
approx_set(x)
```

参数	说明
x	参数值为任意数据类型。

HyperLogLog类型。

使用approx_set函数估算每分钟的网站访问UV，返回结果为HyperLogLog编码格式。

- 查询和分析语句

```
* |
SELECT
  date_trunc('minute', __time__) AS Time,
  approx_set(client_ip) AS UV
FROM  website_log
GROUP BY
  Time
ORDER BY
  Time
```

- 查询和分析结果

cardinality函数

cardinality函数用于将HyperLogLog类型的内容转换为bigint类型。

```
cardinality(x)
```

参数	说明
x	参数值为HyperLogLog类型。

bigint类型。

使用approx_set函数估算每分钟的网站访问UV，返回结果为HyperLogLog类型。然后使用cardinality函数将HyperLogLog类型转换为bigint类型。

- 查询和分析语句

```
* |
SELECT
  Time,
  cardinality(UV) AS UV
FROM  (
    SELECT
      date_trunc('minute', __time__) AS Time,
      approx_set(client_ip) AS UV
    FROM      website_log
    GROUP BY
      Time
    ORDER BY
      Time
  ) AS UV
```

- 查询和分析结果

Time	UV
2021-09-09 15:12:00.000	78
2021-09-09 15:13:00.000	561
2021-09-09 15:14:00.000	658
2021-09-09 15:15:00.000	625
2021-09-09 15:16:00.000	852
2021-09-09 15:17:00.000	594
2021-09-09 15:18:00.000	644
2021-09-09 15:19:00.000	562
2021-09-09 15:20:00.000	607

empty_approx_set函数

empty_approx_set函数用于返回一个HyperLogLog类型的空值。最大标准误差默认为0.01625。

```
empty_approx_set()
```

HyperLogLog类型。

返回一个HyperLogLog类型的空值。

- 查询和分析语句

```
* | SELECT  empty_approx_set()
```

- 查询和分析结果

_col0
AgwAAA==

merge函数

merge函数用于聚合计算所有的HyperLogLog值。

```
merge(x)
```

参数	说明
x	参数值为HyperLogLog类型。

HyperLogLog类型。

使用approx_set函数估算每分钟的网站访问UV，使用merge函数聚合计算15分钟内所有的UV，然后使用cardinality函数将HyperLogLog类型转换为bigint类型。

- 查询和分析语句

```
* |
SELECT
  Time,
  cardinality(UV) AS UV,
  cardinality(merge(UV) over()) AS Total_UV
FROM  (
    SELECT
      date_trunc('minute', __time__) AS Time,
      approx_set(client_ip) AS UV
    FROM      log
    GROUP BY
      Time
    ORDER BY
      Time
  )
```

- 查询和分析结果

Time	UV	Total_UV
2021-09-09 15:13:00.000	561	8564
2021-09-09 15:14:00.000	658	8564
2021-09-09 15:15:00.000	625	8564
2021-09-09 15:16:00.000	852	8564
2021-09-09 15:17:00.000	594	8564
2021-09-09 15:18:00.000	644	8564
2021-09-09 15:19:00.000	562	8564
2021-09-09 15:20:00.000	607	8564
2021-09-09 15:21:00.000	611	8564

总数：15 < 1 / 1 >

11.1.24. 电话号码函数

电话号码函数用于分析中国内地地域电话号码的归属地、运营商等信息。本文介绍电话号码函数的基本语法及示例。

日志服务支持如下电话号码函数。

> 注意 在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。

函数名称	语法	说明
mobile_carrier函数	mobile_carrier(*x*)	分析电话号码所属运营商。
mobile_city函数	mobile_city(*x*)	分析电话号码所属城市。
mobile_province函数	mobile_province(*x*)	分析电话号码所属省份。

mobile_carrier函数

mobile_carrier函数用于分析电话号码所属运营商。

```
mobile_carrier(x)
```

参数	说明
x	参数值为数字形式的电话号码。 当参数值不为bigint类型时，您可以使用tyr_cast函数进行转换。更多信息，请参见try_cast函数。

varchar类型。

通过mobile字段查询电话号码所属运营商。

- 字段样例

  ```
  mobile:1881111****
  ```

- 查询和分析语句

  ```
  * | SELECT mobile_carrier(mobile)
  ```

- 查询和分析结果

_col0
中国移动

mobile_city函数

mobile_city函数用于分析电话号码所属的城市。

```
mobile_carrier(x)
```

参数	说明
x	参数值为数字形式的电话号码。 当参数值不为bigint类型时，您可以使用tyr_cast函数进行转换。更多信息，请参见try_cast函数。

varchar类型。

电商公司A通过访问日志中的mobile字段和client_ip字段，分析哪些客户的电话号码所在地和其访问公司网站的IP地址所在地不同。

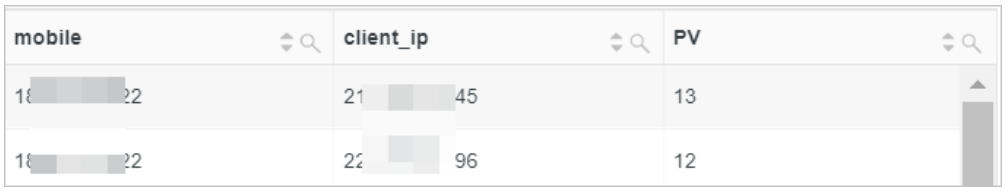
- 字段样例

```
mobile:1881111****
client_ip:192.168.2.0
```

- 查询和分析语句

```
* |
SELECT
  mobile,
  client_ip,
  count(*) as PV
WHERE
  mobile_city(mobile) != ip_to_city(client_ip)
  AND ip_to_city(client_ip) != ''
GROUP BY
  client_ip,
  mobile
ORDER BY
  PV DESC
```

- 查询和分析结果



mobile	client_ip	PV
18[redacted]22	21[redacted]45	13
18[redacted]22	22[redacted]96	12

mobile_province函数

mobile_province函数用于分析电话号码所属省份。

```
mobile_province(x)
```

参数	说明
x	参数值为数字形式的电话号码。 当参数值不为bigint类型时，您可以使用tyr_cast函数进行转换。更多信息，请参见try_cast函数。

varchar类型。

电商公司A通过访问日志中的mobile字段，分析客户电话号码所在省份以及所在省份的客户数量。

- 字段样例

```
mobile:1881111****
```

- 查询和分析语句

```
* |
SELECT
  mobile_province(mobile) AS Province,
  count(1) AS PV
GROUP BY
  Province
ORDER BY
  PV DESC
```

- 查询和分析结果

Province	PV
浙江省	234782

11.1.25. 比较运算符

比较运算符用于判断参数的大小关系，适用于任意可比较的数据类型（double、bigint、varchar、timestamp和date）。本文介绍比较运算符的基本语法以及示例。

日志服务支持如下比较运算符。

> 注意 在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。

运算符	语法	说明
基础运算符	*x* < *y*	*x*小于*y*时，返回true。
	x > *y*	*x*大于*y*时，返回true。
	x <= *y*	*x*小于或等于*y*时，返回true。
	x >= *y*	*x*大于或等于*y*时，返回true。
	x = *y*	*x*等于*y*时，返回true。
	x <> *y*	*x*不等于*y*时，返回true。
	x != *y*	*x*不等于*y*时，返回true。
ALL运算符	*x relational operator* ALL(*subquery*)	*x*满足所有条件时，返回true。
ANY运算符	*x relational operator* ANY(*subquery*)	*x*满足任意一个条件时，返回true。
BETWEEN运算符	*x* BETWEEN *y* AND *z*	*x*处在*y*和*z*之间时，返回true。
DISTINCT运算符	*x* IS DISTINCT FROM *y*	*x*不等于*y*时，返回true。
	x IS NOT DISTINCT FROM *y*	*x*等于*y*时，返回true。
LIKE运算符	*x* LIKE *pattern* [escape '*escape_character*']	用于匹配字符串中指定的字符模式。字符串区分大小写。
SOME运算符	*x relational operator* SOME(*subquery*)	*x*满足任意一个条件时，返回true。
GREATEST运算符	GREATEST(*x*, *y*...)	查询*x*、*y*中的最大值。

运算符	语法	说明
LEAST运算符	LEAST(*x*, *y*...)	查询*x*、*y*中的最小值。
NULL运算符	*x* IS NULL	*x*为null时，返回true。
	x IS NOT NULL	*x*不为null时，返回true。

基础运算符

基础运算符用于比较*x*和*y*的大小关系。如果逻辑成立，则返回true。

语法	说明
x < *y*	*x*小于*y*
x > *y*	*x*大于*y*
x <= *y*	*x*小于或等于*y*
x >= *y*	*x*大于或等于*y*
x = *y*	*x*等于*y*
x <> *y*	*x*不等于*y*
x != *y*	*x*不等于*y*

参数	说明
x	参数值为任意可比较的数据类型。
y	参数值为任意可比较的数据类型。

boolean类型。

- 示例1：查询昨天的日志。
  - 查询和分析语句

```
* |
SELECT
  *
FROM  log
WHERE
  __time__ < to_unixtime(current_date)
  AND __time__ > to_unixtime(date_add('day', -1, current_date))
```

- 查询和分析结果

body_bytes_sent	client_ip	host	http_user_agent	http_x_forwarded_for	instance_id	instance_name	network_type	owner_id	referer
4650	42. 0	www .com	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) ... 展开	6	i-01	instance-01	vlan	owner-01	ww k.com
1492	42. 3	www k.com	Mozilla/5.0 (iPhone; U; ru; CPU iPhone OS 4_2_1... 展开	1	i-02	instance-01	vlan	owner-01	ww k.cor
3148	59.	www k.com	Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_3_1 lik... 展开	1	i-02	instance-02	vlan	owner-01	ww k.cor

- 示例2：电商公司A通过访问日志中的mobile字段和client_ip字段，分析哪些客户的电话号码所在地和其访问电商网站的IP地址所在地不同。
  - 字段样例

```
mobile:1881111****
client_ip:192.168.2.0
```

  - 查询和分析语句

```
* |
SELECT
  mobile,
  client_ip,
  count(*) AS PV
WHERE
  mobile_city(mobile) != ip_to_city(client_ip)
  AND ip_to_city(client_ip) != ''
GROUP BY
  client_ip,
  mobile
ORDER BY
  PV DESC
```

  - 查询和分析结果

mobile	client_ip	PV
1 22	21 45	13
1 22	22 96	12

ALL运算符

ALL运算符用于判断*x*是否满足所有条件。如果满足，则返回true。

```
x relational operator ALL(subquery)
```

参数	说明
x	参数值为任意可比较的数据类型。

参数	说明
比较运算符	<、>、<=、>=、=、<>、!= 注意 ALL运算符必须紧跟在基础运算符（<、>、<=、>=、=、<>、!=）后面。
subquery	SQL子查询。

boolean类型。

实例i-01相关的所有请求的状态码是否都为200。

- 字段样例

```
instance_id:i-01
status:200
```

- 查询和分析语句

```
* | select 200 = ALL(select status where instance_id='i-01')
```

- 查询和分析结果

_col0
false

ANY运算符

ANY运算符用于判断*x*是否满足任意一个条件。如果满足，则返回true。

```
x relational operator ANY(subquery)
```

参数	说明
x	参数值为任意可比较的数据类型。
比较运算符	<、>、<=、>=、=、<>、!= 注意 ANY运算符必须紧跟在比较运算符（<、>、<=、>=、=、<>、!=）后面。
subquery	SQL子查询。

boolean类型。

实例i-01相关的请求中，是否存在请求状态码为200的请求。

- 字段样例

```
instance_id:i-01
status:200
```

- 查询和分析语句

```
* | SELECT 200 = ANY(SELECT status WHERE instance_id='i-01')
```

- 查询和分析结果

_col0
true

BETWEEN运算符

BETWEEN用于判断*x*是否处在*y*和*z*之间。如果是，则返回true。*y*和*z*之间的范围为闭区间。

```
x BETWEEN y AND z
```

参数	说明
x	参数值为任意可比较的数据类型。
y	参数值为任意可比较的数据类型。
z	参数值为任意可比较的数据类型。

> 注意
> - *x*、*y*和*z*的数据类型必须一致。
> - *x*、*y*和*z*中任意一个的值包含null，则返回结果为null。

boolean类型。

- 示例1：判断status字段值是否在[200,299]范围内。
  - 查询和分析语句

    ```
    * | SELECT status BETWEEN 200 AND 299
    ```

  - 查询和分析结果

_col0
true
true
true

- 示例2：计算status字段值不在[200,299]范围内的日志条数。
  - 查询和分析语句

    ```
    * | SELECT count(*) AS count FROM log WHERE status NOT BETWEEN 200 AND 299
    ```

  - 查询和分析结果

count
250

DISTINCT运算符

DISTINCT运算符用于判断*x*和*y*是否相同。

- IS DISTINCT FROM表示*x*不等于*y*时，返回true。

```
x IS DISTINCT FROM y
```

- IS NOT DISTINCT FROM表示*x*等于*y*时，返回true。

```
x IS NOT DISTINCT FROM y
```

参数	说明
x	参数值为任意可比较的数据类型。
y	参数值为任意可比较的数据类型。

与基础运算符（=、<>）对比，区别在于DISTINCT运算符可用于null的对比。

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

boolean类型。

将0和null进行对比。

- 查询和分析语句

```
* | select 0 IS DISTINCT FROM null
```

- 查询和分析结果

LIKE运算符

LIKE运算符用于匹配字符串中指定的字符模式。字符串区分大小写。

```
x LIKE pattern [escape 'escape_character']
```

参数	说明
x	参数值为任意可比较的数据类型。
pattern	字符模式，包括字符串和通配符。通配符说明如下： • 百分号（%）代表任意个字符。 • 下划线（_）代表单个字符。
escape_character	对字符模式中的通配符进行转义的字符表达式。

> 说明 LIKE运算符主要用于日志的精准查询。更多信息，请参见如何精准查询日志。

boolean类型。

- 示例1：查询request_uri字段值是以file-8或file-6结尾的日志。
  - 字段样例

    ```
    request_uri:/request/path-2/file-6
    ```

  - 查询和分析语句

    ```
    *|SELECT * WHERE request_uri LIKE '%file-8' OR request_uri LIKE '%file-6'
    ```

  - 查询和分析结果

ion	remote_addr	remote_user	request_length	request_method	request_time	request_uri	scheme	server_protocol	slbid	status	time_local
shanghai	1[redacted]01	kqt37	1608	DELETE	21.0	/request/path-2/file-6	https	HTTP/1.0	slb-02	200	09/Aug/2021 06:18
shanghai	11[redacted]26	lhuu	1500	PUT	29.0	/request/path-0/file-6	http	HTTP/1.1	slb-02	200	09/Aug/2021 06:18

- 示例2：判断request_uri字段值是否以file-6结尾。
  - 字段样例

    ```
    request_uri:/request/path-2/file-6
    ```

  - 查询和分析语句

    ```
    * | SELECT request_uri LIKE '%file-6'
    ```

  - 查询和分析结果

_col0
true

SOME运算符

SOME运算符用于判断*x*是否满足任意一个条件。如果满足，则返回true。

```
x relational operator SOME(subquery)
```

参数	说明
x	参数值为任意可比较的数据类型。
比较运算符	<、>、<=、>=、=、<>、!= 注意 SOME运算符必须紧跟在比较运算符（<、>、<=、>=、=、<>、!=）后面。
subquery	SQL子查询。

boolean类型。

实例i-01相关的请求中，是否存在请求时长小于20s的请求。

- 字段样例

```
instance_id:i-01
request_time:16
```

- 查询和分析语句

```
* | SELECT 20 > SOME(SELECT request_time WHERE instance_id='i-01')
```

- 查询和分析结果

_col0
true

GREATEST运算符

GREATEST运算符用于获取*x*、*y*中的最大值。

> 说明 GREATEST运算符用于横向对比，max函数用于纵向对比。

```
GREATEST(x, y...)
```

参数	说明
x	参数值为任意可比较的数据类型。
y	参数值为任意可比较的数据类型。

double类型。

对同一行中的request_time字段值和status字段值进行对比，获取其中的最大值。

- 字段样例

```
request_time:38
status:200
```

- 查询和分析语句

```
* |  SELECT GREATEST(request_time,status)
```

- 查询和分析结果

_col0
200.0

LEAST运算符

LEAST运算符用于获取*x*、*y*中的最小值。

> 说明 LEAST运算符用于横向对比，min函数用于纵向对比。

```
LEAST(x, y...)
```

参数	说明
x	参数值为任意可比较的数据类型。
y	参数值为任意可比较的数据类型。

double类型。

对同一行中的request_time字段值和status字段值进行对比，获取其中的最小值。

- 字段样例

```
request_time:77
status:200
```

- 查询和分析语句

```
* |  SELECT LEAST(request_time,status)
```

- 查询和分析结果

NULL运算符

NULL运算符用于判断*x*是否为null。

- IS NULL表示参数值为null时，返回true。

```
x IS NULL
```

- IS NOT NULL表示参数值不为null时，返回true。

```
x IS NOT NULL
```

参数	说明
x	参数值为任意可比较的数据类型。

boolean类型。

- 示例1：判断status字段值是否为null。
  - 查询和分析语句

```
* | select status IS NULL
```

  - 查询和分析结果

- 示例2：统计status字段值不为空的日志条数。
  - 查询和分析语句

```
* | SELECT count(*) AS count FROM log WHERE status IS NOT NULL
```

- 查询和分析结果

count
1340

11.1.26. 逻辑运算符

本文介绍逻辑运算符的基本语法及示例。

日志服务支持如下逻辑运算符。

> 注意
> - 在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。
> - 逻辑运算符优先级从高到低为not、and、or。您可以使用圆括号改变默认的计算顺序。
> - 逻辑运算只支持输入值为true、false或null的布尔表达式。

运算符	语法	说明
AND运算符	*x* AND *y*	*x*和*y*的值都为true时，返回结果为true。
OR运算符	*x* OR *y*	*x*和*y*中任意一个的值为true时，返回结果为true。
NOT运算符	NOT *x*	*x*的值为false时，返回结果为true。

AND运算符

*x*和*y*的值都为true时，返回结果为true。

```
x AND y
```

参数	说明
x	参数值为布尔表达式。
y	参数值为布尔表达式。

boolean类型。

如果status字段值为200且request_method字段值为GET，则返回true。否则返回false。

- 查询和分析语句

```
*|SELECT status=200 AND request_method='GET'
```

- 查询和分析结果

_col0
true
false

OR运算符

*x*和*y*中任意一个的值为true时，返回结果为true。

```
x OR y
```

参数	说明
x	参数值为布尔表达式。
y	参数值为布尔表达式。

boolean类型。

查找request_uri字段值是以file-8或file-6的结尾的日志。

- 查询和分析语句

```
*|SELECT *  WHERE request_uri LIKE '%file-8' OR request_uri LIKE '%file-6'
```

- 查询和分析结果

ion	remote_addr	remote_user	request_length	request_method	request_time	request_uri	scheme	server_protocol	slbid	status	time_local
shanghai	1[redacted]01	kqt37	1608	DELETE	21.0	/request/path-2/file-6	https	HTTP/1.0	slb-02	200	09/Aug/2021 06:18
shanghai	11[redacted]26	lhuu	1500	PUT	29.0	/request/path-0/file-6	http	HTTP/1.1	slb-02	200	09/Aug/2021 06:18

NOT运算符

*x*的值为false时，返回结果为true。

```
NOT x
```

参数	说明
x	参数值为布尔表达式。

boolean类型。

统计请求状态码不为200时的请求时长。

- 查询和分析语句

```
*|SELECT request_time WHERE NOT status=200
```

- 查询和分析结果

附录：真值表

*x*和*y*的值为true、false或null时，真值表如下所示。

x	*y*	*x* AND *y*	*x* OR *y*	NOT *x*
true	true	true	true	false
true	false	false	true	false
true	null	null	true	false
false	true	false	true	true
false	false	false	false	true
false	null	false	null	true
null	true	null	true	null
null	false	false	null	null
null	null	null	null	null

11.1.27. 单位换算函数

日志服务提供单位换算函数，帮助您换算数据量或时间间隔的单位。本文介绍单位换算函数的基本语法及示例。

日志服务支持如下单位换算函数。

> 注意 在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。

函数分类	函数名称	语法	说明
	convert_data_size函数	convert_data_size(*x*)	对数据量单位进行换算，系统自动判断最优的换算单位，返回使用最优单位表示的数据量。返回类型为string。例如将1024 KB换算为1 MB，1024 MB换算为1 GB。
		convert_data_size(*x*, *unit*)	对数据量单位进行换算，返回使用指定单位表示的数据量。返回类型为string。
	format_data_size函数	format_data_size(*x*, *unit*)	对Byte单位进行换算，返回使用指定单位表示的数据量。返回类型为string。

函数分类	函数名称	语法	说明
数据量单位转换函数	parse_data_size函数	parse_data_size(*x*)	对数据量单位进行换算，返回以Byte为单位的数据量。返回类型为decimal。
	to_data_size_B函数	to_data_size_B(*x*)	对数据量单位进行换算，返回以Byte为单位的数据量。返回类型为double。
	to_data_size_KB函数	to_data_size_KB(*x*)	对数据量单位进行换算，返回以KB为单位的数据量。返回类型为double。
	to_data_size_MB函数	to_data_size_MB(*x*)	对数据量单位进行换算，返回以MB为单位的数据量。返回类型为double。
	to_data_size_GB函数	to_data_size_GB(*x*)	对数据量单位进行换算，返回以GB为单位的数据量。返回类型为double。
	to_data_size_TB函数	to_data_size_TB(*x*)	对数据量单位进行换算，返回以TB为单位的数据量。返回类型为double。
	to_data_size_PB函数	to_data_size_PB(*x*)	对数据量单位进行换算，返回以PB为单位的数据量。返回类型为double。
时间间隔单位转换函数	format_duration函数	format_duration(*x*)	对以秒为单位的时间间隔进行格式化，转换为可读的字符串类型。
	parse_duration函数	parse_duration(*x*)	对时间间隔进行格式化，转换为`0 00:00:00.000`格式。
	to_days函数	to_days(*x*)	对时间间隔单位进行换算，转换为以天为单位的时间间隔。
	to_hours函数	to_hours(*x*)	对时间间隔单位进行换算，转换为以小时为单位的时间间隔。
	to_microseconds函数	to_microseconds(*x*)	对时间间隔单位进行换算，转换为以微秒为单位的时间间隔。
	to_milliseconds函数	to_milliseconds(*x*)	对时间间隔单位进行换算，转换为以毫秒为单位的时间间隔。
	to_minutes函数	to_minutes(*x*)	对时间间隔单位进行换算，转换为以分钟为单位的时间间隔。
	to_most_succinct_time_unit函数	to_most_succinct_time_unit(*x*)	对时间间隔单位进行换算，系统自动判断最优的换算单位，返回使用最优单位表示的时间间隔。
	to_nanoseconds函数	to_nanoseconds(*x*)	对时间间隔单位进行换算，转换为以纳秒为单位的时间间隔。
	to_seconds函数	to_seconds(*x*)	对时间间隔单位进行换算，转换为以秒为单位的时间间隔。

convert_data_size函数

convert_data_size函数用于对数据量单位进行换算。

- 对数据量单位进行换算，系统自动判断最优的换算单位，返回使用最优单位表示的数据量。

```
convert_data_size(x)
```

- 对数据量单位进行换算，返回使用指定单位表示的数据量。

```
convert_data_size(x, unit)
```

参数	说明
x	数据量，参数值为string类型。
unit	数据的存储单位，取值范围为KB、MB、GB、PB、TB、EB、ZB、YB。

string类型。

- 示例1：将1200 KB换算为其他数据单位的值。
  - 查询和分析语句

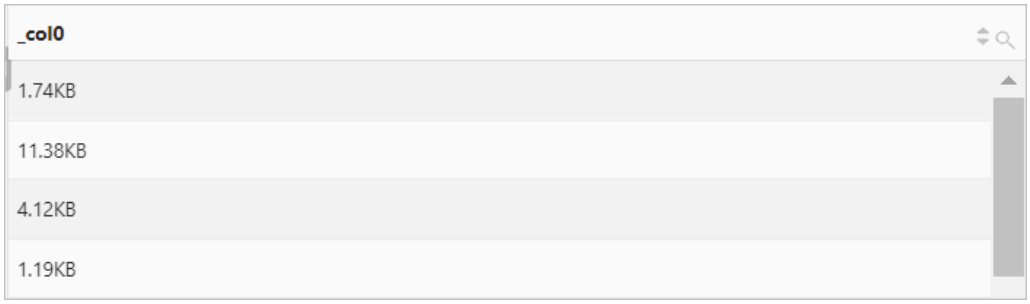
```
* | SELECT convert_data_size('1200KB')
```

  - 查询和分析结果

- 示例2：body_bytes_sent字段表示发送给客户端的字节数，单位为Byte。通过convert_data_size函数将字段值换算为以KB为单位的值。
  - 查询和分析语句

```
* | select convert_data_size(format_data_size(body_bytes_sent, 'KB'))
```

  - 查询和分析结果



format_data_size函数

format_data_size函数用于对Byte单位进行换算，返回使用指定单位表示的数据量。

```
format_data_size(x, unit)
```

参数	说明
x	以Byte为单位的数据量，参数值为bigint类型。
unit	数据的存储单位，取值范围为KB、MB、GB、PB、TB、EB、ZB、YB。

string类型。

- 示例1：body_bytes_sent字段表示发送给客户端的字节数，单位为Byte。通过format_data_size函数将字段值换算

为以KB为单位的值。

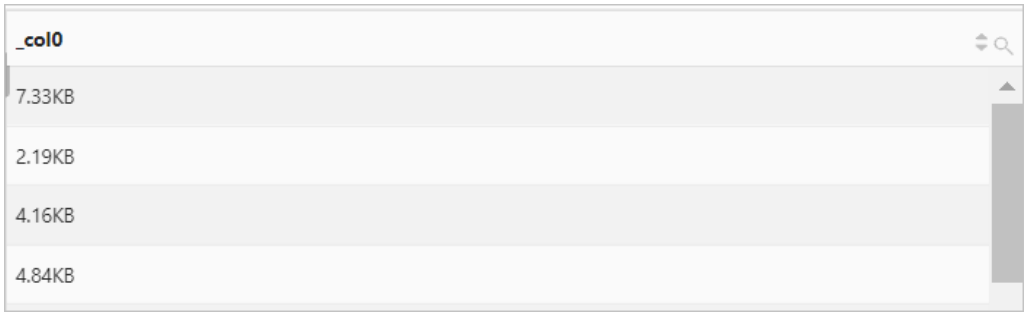
- 字段样例

```
body_bytes_sent:4619
```

- 查询和分析语句

```
* | select format_data_size(body_bytes_sent, 'KB')
```

- 查询和分析结果



- 示例2：body_bytes_sent字段表示发送给客户端的字节数，单位为Byte。通过sum函数计算总字节数，再通过format_data_size函数将总字节数换算为以GB为单位的值。
  - 字段样例

```
body_bytes_sent:4619
```

  - 查询和分析语句

```
* | select format_data_size(sum(body_bytes_sent), 'GB')
```

  - 查询和分析结果

parse_data_size函数

parse_data_size函数用于对数据量单位进行换算，返回以Byte为单位的数据量。

```
parse_data_size(x)
```

参数	说明
x	数据量，参数值为string类型。

decimal类型。

将1024 KB换算为以Byte为单位的值。

- 查询和分析语句

```
*| SELECT parse_data_size('1024KB')
```

- 查询和分析结果

to_data_size_B函数

to_data_size_B函数用于对数据量单位进行换算，返回以Byte为单位的数据量。

```
to_data_size_B(x)
```

参数	说明
x	数据量，参数值为string类型。

double类型。

将1024 KB换算为以Byte为单位的值。

- 查询和分析语句

```
* | select to_data_size_B('1024KB')
```

- 查询和分析结果

_col0
1048576.0

to_data_size_KB函数

to_data_size_KB函数用于对数据量单位进行换算，返回以KB为单位的数据量。

```
to_data_size_KB(x)
```

参数	说明
x	数据量，参数值为string类型。

double类型。

body_bytes_sent字段表示发送给客户端的字节数，单位为Byte。通过to_data_size_KB函数将字段值换算为以KB为单位的值。

- 查询和分析语句

```
* | select to_data_size_KB(format_data_size(body_bytes_sent, 'KB'))
```

- 查询和分析结果

_col0
3.52
4.25
3.2
1.69

to_data_size_MB函数

to_data_size_MB函数用于对数据量单位进行换算，返回以MB为单位的数据量。

```
to_data_size_MB(x)
```

参数	说明
x	数据量，参数值为string类型。

double类型。

body_bytes_sent字段表示发送给客户端的字节数，单位为Byte。通过to_data_size_MB函数将总字节数换算为以MB为单位的值。

- 查询和分析语句

```
* | select to_data_size_MB(format_data_size(sum(body_bytes_sent), 'KB'))
```

- 查询和分析结果

to_data_size_GB函数

to_data_size_GB函数用于对数据量单位进行换算，返回以GB为单位的数据量。

```
to_data_size_GB(x)
```

参数	说明
x	数据量，参数值为string类型。

double类型。

body_bytes_sent字段表示发送给客户端的字节数，单位为Byte。通过to_data_size_GB函数将总字节数换算为以GB为单位的值。

- 查询和分析语句

```
* | select to_data_size_GB(format_data_size(sum(body_bytes_sent), 'KB'))
```

- 查询和分析结果

to_data_size_TB函数

to_data_size_TB函数用于对数据量单位进行换算，返回以TB为单位的数据量。

```
to_data_size_TB(x)
```

参数	说明
x	数据量，参数值为string类型。

double类型。

body_bytes_sent字段表示发送给客户端的字节数，单位为Byte。通过to_data_size_TB函数将总字节数换算为以TB为单位的值。

- 查询和分析语句

```
* | select to_data_size_TB(format_data_size(sum(body_bytes_sent), 'KB'))
```

- 查询和分析结果

to_data_size_PB函数

to_data_size_PB函数用于对数据量单位进行换算，返回以PB为单位的数据量。

```
to_data_size_PB(x)
```

参数	说明
x	数据量，参数值为string类型。

double类型。

将1048576 GB换算为以PB为单位的值。

- 查询和分析语句

```
*| SELECT to_data_size_PB('1048576GB')
```

- 查询和分析结果

format_duration函数

format_duration函数用于对以秒为单位的时间间隔进行格式化，转换为可读的字符串类型。

```
format_duration(x)
```

参数	说明
x	时间间隔，参数值为double类型。

string类型。

将235秒转换为 `3 minutes, 55 seconds` 格式。

- 查询和分析语句

```
* | SELECT format_duration(235)
```

- 查询和分析结果

parse_duration函数

parse_duration函数用于对时间间隔进行格式化，转换为 `0 00:00:00.000` 格式。

```
parse_duration(x)
```

参数	说明
x	时间间隔，参数值为string类型。

interval类型。

将1340毫秒转换为 `0 00:00:01.340` 格式。

- 查询和分析语句

```
* | SELECT parse_duration('1340ms')
```

- 查询和分析结果

to_days函数

to_days函数用于对时间间隔单位进行换算，转换为以天为单位的时间间隔。

```
to_days(x)
```

参数	说明
x	时间间隔，参数值为varchar类型。

double类型。

将192848s转换为以天为单位的时间间隔。

- 查询和分析语句

```
*| SELECT to_days('192848s')
```

- 查询和分析结果

to_hours函数

to_hours函数用于对时间间隔单位进行换算，转换为以小时为单位的时间间隔。

```
to_hours(x)
```

参数	说明
x	时间间隔，参数值为varchar类型。

double类型。

将1.2天转换为以小时为单位的时间间隔。

- 查询和分析语句

```
* | SELECT to_hours('1.2d')
```

- 查询和分析结果

to_microseconds函数

to_microseconds函数用于对时间间隔单位进行换算，转换为以微秒为单位的时间间隔。

```
to_microseconds(x)
```

参数	说明
x	时间间隔，参数值为varchar类型。

double类型。

将3600纳秒转换为以微秒为单位的时间间隔。

- 查询和分析语句

```
* | SELECT to_microseconds('3600ns')
```

- 查询和分析结果

to_milliseconds函数

to_milliseconds函数用于对时间间隔单位进行换算，转换为以毫秒为单位的时间间隔。

```
to_milliseconds(x)
```

参数	说明
x	时间间隔，参数值为varchar类型。

double类型。

将1.2秒转换为以毫秒为单位的时间间隔。

- 查询和分析语句

```
* | SELECT to_milliseconds('1.2s')
```

- 查询和分析结果

to_minutes函数

to_minutes函数用于对时间间隔单位进行换算，转换为以分钟为单位的时间间隔。

```
to_minutes(x)
```

参数	说明
x	时间间隔，参数值为varchar类型。

double类型。

将1.2小时转换为以分钟为单位的时间间隔。

- 查询和分析语句

```
* | SELECT to_minutes('1.2h')
```

- 查询和分析结果

to_most_succinct_time_unit函数

to_most_succinct_time_unit函数用于对时间间隔单位进行换算，系统自动判断最优的换算单位，返回使用最优单位表示的时间。

```
to_most_succinct_time_unit(x)
```

参数	说明
x	时间间隔，参数值为varchar类型。

varchar类型。

将1340ms转换为以秒为单位的时间间隔。

- 查询和分析语句

```
* | SELECT to_most_succinct_time_unit('1340ms')
```

- 查询和分析结果

to_nanoseconds函数

to_nanoseconds函数用于对时间间隔单位进行换算，转换为以纳秒为单位的时间间隔。

```
to_nanoseconds(x)
```

参数	说明
x	时间间隔，参数值为varchar类型。

double类型。

将125毫秒转换为以纳秒为单位的时间间隔。

- 查询和分析语句

```
* | SELECT to_nanoseconds('125ms')
```

- 查询和分析结果

to_seconds函数

to_seconds函数用于对时间间隔单位进行换算，转换为以秒为单位的时间间隔。

```
to_seconds(x)
```

参数	说明
x	时间间隔，参数值为varchar类型。

double类型。

将1340毫秒转换为以秒为单位的时间间隔。

- 查询和分析语句

```
* | SELECT to_seconds('1340ms')
```

- 查询和分析结果

11.1.28. 窗口漏斗函数

日志服务提供窗口漏斗函数，可用于分析用户行为、APP流量、产品目标转化等数据。本文介绍窗口漏斗函数的基本语法和示例。

日志服务支持如下窗口漏斗函数。

> 注意 在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。

函数名称	语法	说明

函数名称	语法	说明
window_funnel函数	window_funnel(*sliding_window*, *timestamp*, *event_id*, ARRAY[*event_list01*, *event_list02*...])	在滑动的时间窗口中搜索事件链并计算事件链中发生的最大连续的事件数。 如果数据中已定义事件列（event_id），可选择该语法。
	window_funnel(*sliding_window*, *timestamp*, ARRAY[*event_id*=*event_list01*, *event_id*=*event_list02*...])	在滑动的时间窗口中搜索事件链并计算事件链中发生的最大连续的事件数。 如果您想要自定义事件的非枚举值，可选择该语法，更具有灵活性。

原理

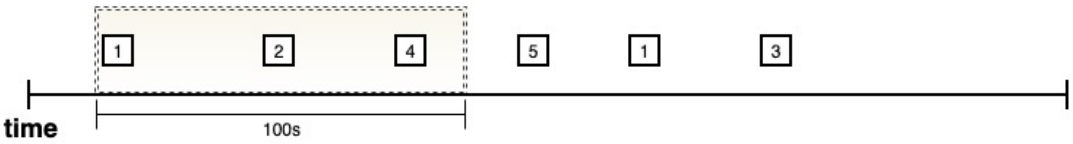
窗口漏斗函数用于在滑动的时间窗口中搜索事件链并计算事件链中发生的最大连续的事件数。根据您定义的事件链，从第一个事件开始匹配，依次做有序最长的匹配，返回最大连续事件数。

窗口漏斗函数所采用的算法的详细说明如下：

- 从事件链中的第一个事件开始并将事件计算器设置为1，然后开启滑动窗口。
- 在滑动窗口内，如果事件链中的后续事件按顺序发生，则事件计数器依次递增。
- 在滑动窗口内，如果事件序列中断，则停止本轮搜索，事件计数器不再增加，并开始新一轮的搜索，直到搜索结束。
- 结束搜索时，如果存在多个搜索结果，则函数将返回最大值，即最长事件链的大小。

例如：您定义的滑动时间窗口为100秒，事件链及其顺序为事件1->事件2->事件3->事件4->事件5，实际发生的事件序列为事件1->事件2->事件4->事件5->事件1->事件3，则函数返回值为2，即表示在100秒内按照您定义的事件链发生的最大连续事件数为2（事件1->事件2）。

> 注意
> - 搜索必须从第一个事件开始。例如实际发生的事件序列为事件2->事件3->事件4，则函数返回值为0。
> - 搜索必须有序，不能跳过事件链中的某个事件。例如事件4也在滑动窗口内，但是未发生事件3，因此不计入结果。



语法

日志服务支持如下两种格式的窗口漏斗函数。

- 如果数据中已定义事件列（event_id），可选择如下语法。

```
window_funnel(sliding_window, timestamp, event_id, ARRAY[event_list01, event_list02...])
```

- 如果您想要自定义事件的非枚举值，可选择如下语法，更具有灵活性。

```
window_funnel(sliding_window, timestamp, ARRAY[event_id=event_list01, event_id=event_list02...])
```

参数说明

参数	说明
sliding_window	滑动的时间窗口，单位为秒。参数值为bigint类型。
timestamp	时间戳，单位为秒。参数值为bigint类型。推荐使用日志服务内置的时间字段__time__。
event_id	日志字段名，该字段值表示事件，例如：A、B、C。参数值为varchar类型。
event_list	自定义的事件链，最多支持32个事件。参数值为array类型。例如： • ARRAY['A', 'B', 'C'] • ARRAY[event_id='A', event_id='B', event_id='C']

示例

某电商店铺举行了一次推广活动，现通过窗口漏斗函数分析本次推广活动的转化效果（转化路径为浏览商品、加入购物车、购买商品）。日志服务采集到的日志样例如下：

日志字段	说明
behavior_type	用户行为类型。包括 • pv：浏览商品。 • cart：将商品加入购物车。 • buy：购买商品。
category_id	商品类目ID。
item_id	商品ID。
timestamp	用户行为发生的时间点。
user_id	用户ID。

示例1

分析用户在24小时内的购买行为。

- 查询和分析语句

```
* |
SELECT
  user_id,
  window_funnel(
    86400,
    timestamp,
    ARRAY [behavior_type='pv', behavior_type='cart',behavior_type='buy']
  ) AS levels
GROUP BY
  user_id
ORDER BY
  user_id
LIMIT
  1000
```

- 查询和分析结果
  - 例如用户24的levels值为3，表示该用户按照浏览商品、将商品加入购物车、购买商品的顺序，完成了购买。
  - 例如用户14的levels值为2，表示该用户浏览了商品以及将商品加入购物车，但是未购买。

user_id	levels
9	2
14	2
24	3
28	1
29	2
31	2
35	2
37	2
38	3

总数：1000 < 1 / 50 >

示例2

分析不同用户行动对应的人数。

- 查询和分析语句

```
* |
SELECT
  levels,
  count,
  sum(count) over(
    ORDER BY
      levels DESC
  ) AS total
FROM  (
    SELECT
      levels,
      count(1) AS count
    FROM      (
        SELECT
          user_id,
          window_funnel(
            86400,
            timestamp,
            ARRAY [behavior_type='pv', behavior_type='cart',behavior_type='buy']
          ) AS levels
        FROM          log
        GROUP BY
          user_id
      )
    GROUP BY
      levels
    ORDER BY
      levels
  )
```

- 查询和分析结果
  - 浏览了商品的人数为513194，其中浏览商品后就离开的人数为138491。
  - 将商品加入购物车的人数为374703，其中加入购物车后离开的人数为198642人。
  - 购买商品的人数为176061。

levels	count	total
3	176061	176061
2	198642	374703
1	138491	513194
0	2072	515266

示例3

计算本次推广活动的转化率。

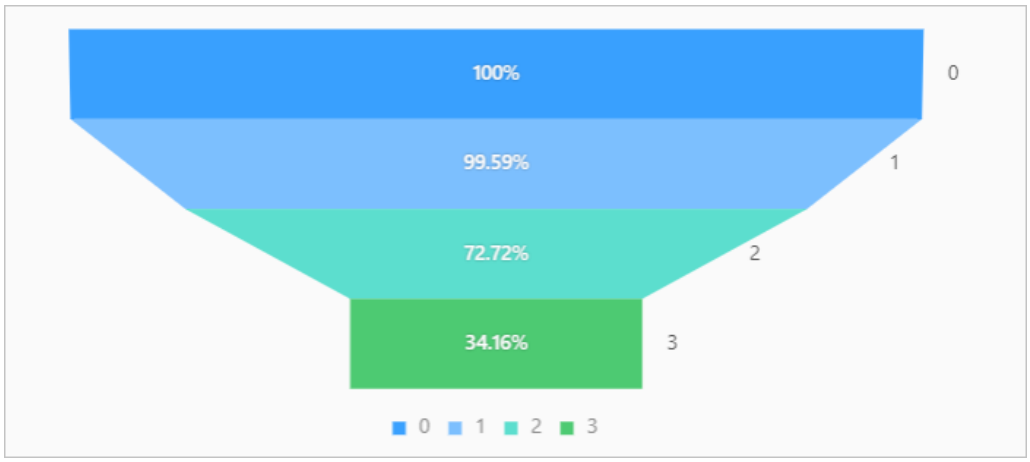
- 绝对转化率：每步用户行为的人数占总人数的比例。
- 相对转化率：每步用户行为的人数占上一步人数的比例。
- 查询和分析语句

```
* |
SELECT
  *,
  100.0 * total /(sum(count) over()) AS "绝对转化率",
  if(
    lag(total, 1, 0) over() = 0,
    100,
    (100.0 * total / lag(total, 1, 0) over())
  ) AS "相对转化率"
FROM  (
    SELECT
      levels,
      count,
      sum(count) over(
        ORDER BY
          levels DESC
      ) AS total
    FROM      (
        SELECT
          levels,
          count(1) AS count
        FROM          (
            SELECT
              user_id,
              window_funnel(
                86400,
                timestamp,
                ARRAY [behavior_type='pv', behavior_type='cart',behavior_type='buy']
              ) AS levels
            FROM              log
            GROUP BY
              user_id
          )
        GROUP BY
          levels
      )
    ORDER BY
      levels
  )
```

- 查询和分析结果
  - 表格

levels	count	total	绝对转化率	相对转化率
0	2072	515266	100.0	100.0
1	138491	513194	99.59787760108371	99.59787760108371
2	198642	374703	72.72030368780396	73.01390897009708
3	176061	176061	34.168953511390238	46.98681355633662

- 漏斗图



11.1.29. Lambda表达式

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。本文介绍Lambda表达式的基本语法及示例。

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、map_filter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

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

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

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

将对换两个数组的元素位置，然后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[1, 3, 5], array['a', 'b', 'c'], (x, y) -> (y, x))
```

- 查询和分析结果

将数组[5, NULL, 6]中的各个元素加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

```
* | 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))
* | SELECT zip_with(array['a', 'b', 'c'], array['d', 'e', 'f'], (x, y) -> concat(x, y))
```

示例1：使用Lambda表达式x -> x is not null

示例2：使用Lambda表达式0, (s, x) -> s + x, s -> s

示例3：使用Lambda表达式(k,v) -> v > 10

示例4：使用Lambda表达式(x, y) -> (y, x)

示例5：使用Lambda表达式x -> coalesce(x, 0) + 1

其他示例

11.1.30. 条件表达式

本文介绍条件表达式的基本语法和示例。

日志服务支持如下条件表达式。

> 注意 在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。

表达式	语法	说明
CASE WHEN表达式	CASE WHEN *condition1* THEN *result1* [WHEN *condition2* THEN *result2*] [ELSE *result3*] END	通过条件判断，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*。
COALESCE表达式	COALESCE(*expression1*, *expression2*, *expression3*...)	返回多个表达式中第一个非null的值。
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。

CASE WHEN表达式

CASE WHEN表达式用于对数据进行归类。

```
CASE WHEN condition1 THEN result1
     [WHEN condition2 THEN result2]
     [ELSE result3]
END
```

参数	说明
condition	条件表达式。
result1	返回结果。

- 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

http_user_agent	pv
Chrome	5563
Safari	1842
unknown	1666

- 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 < 10 then 't10'
    WHEN request_time < 100 then 't100'
    WHEN request_time < 1000 then 't1000'
    WHEN request_time < 10000 then 't10000'
    ELSE 'large'
  END AS request_time,
  count(*) AS pv
GROUP BY
  request_time
```

- 查询和分析结果

request_time	pv
t100	1563542
large	533

IF表达式

IF表达式用于对数据进行归类，类似于CASE WHEN表达式。

- 如果*condition*为true，则返回*result 1*，否则返回null。

```
IF(condition, result1)
```

- 如果*condition*为true，则返回*result 1*，否则返回*result 2*。

```
IF(condition, result1, result2)
```

参数	说明
condition	条件表达式。
result	返回结果

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

status_200_percentage
0.884686775009848

COALESCE表达式

COALESCE表达式用于返回多个表达式中第一个非null的值。

```
COALESCE(expression1, expression2, expression3...)
```

参数	说明
expression	任何类型的表达式。

计算昨天消费金额与上月同一天的比值。

- 查询和分析语句

```
* |
SELECT
  compare("昨天消费金额", 604800) AS diff
FROM  (
    SELECT
      COALESCE(sum(PretaxAmount), 0) AS "昨天消费金额"
    FROM log
  )
```

- 查询和分析结果

diff
[6514393413.0,19578267596.0,0.33273594719539659]

  - 6514393413.0表示昨天的消费金额。
  - 19578267596.0表示上月同一天的消费金额。
  - 0.33273594719539659表示昨天与上月同一天的消费金额比值。

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。

```
NULLIF(expression1, expression2)
```

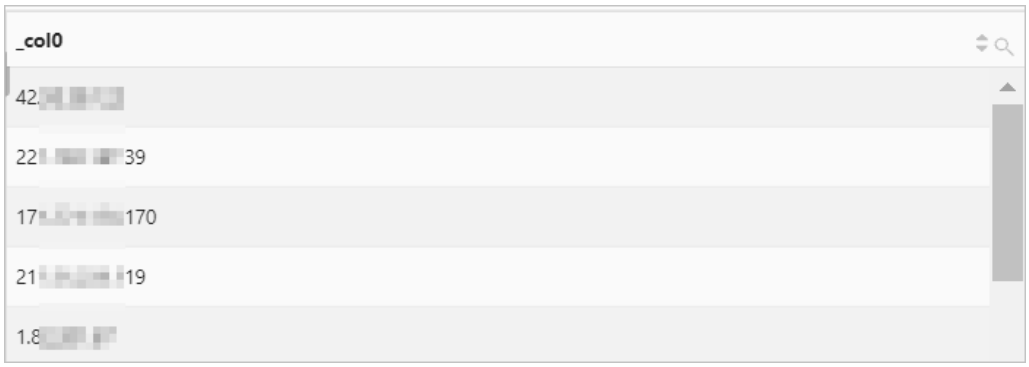
参数	说明
expression	任何有效的标量表达式。

判断client_ip、host两个字段的值是否相同。当不相同时，返回client_ip字段的值。

- 查询和分析语句

```
* | SELECT NULLIF(client_ip,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。

```
TRY(expression)
```

参数	说明
expression	任何类型的表达式。

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(request_uri, '.*\/(file.*)', 1)) AS file,
  count(*) AS count
GROUP BY
  file
```

- 查询和分析结果

file	count
file-5	851
file-7	928
file-3	837
file-4	863

11.2. SQL语法

11.2.1. EXCEPT子句

EXCEPT子句用于组合两个SELECT子句的结果集，并返回两个结果集的差集。即返回的行存在于第一个SELECT子句的结果集中但不存在于第二个SELECT子句的结果集中。本文介绍EXCEPT子句的基本语法和示例。

语法

```
SELECT key1... FROM logstore1
EXCEPT
SELECT key2... FROM logstore2
```

> 注意
>
> - 每个SELECT子句必须拥有相同数量的列，对应列的顺序和数据类型相同。
> - EXCEPT子句会删除最终结果集中重复的行，即返回的每一行都是唯一的。

参数说明

参数	说明
key	字段名、列名或表达式。 *key1*和*key2*的名称可不同，但数据类型必须相同。
logstore	Logstore名称。

示例

名为internal-diagnostic_log的Logstore用于记录重要日志，包括各个Logstore的消费延时、告警、采集等日志；名为internal-operation_log的Logstore用于记录详细日志，包括Project内所有资源的操作日志。您可以使用EXCEPT子句，查询哪些Logstore生成了详细日志，但没有生成重要日志。

- 查询和分析语句

```
* |
SELECT
  logstore
FROM  internal-operation_log
EXCEPT
SELECT
  logstore
FROM  internal-diagnostic_log
```

- 查询和分析结果

11.2.2. EXISTS子句

EXISTS子句用于判断子查询中是否存在查询结果。当EXISTS子句内的查询结果存在时，返回true，并执行外层SQL语句。

语法

```
SELECT...FROM...WHERE EXISTS (subquery)
```

参数说明

参数	说明
subquery	该子查询为一条SELECT语句。

示例

判断Logstore读写数据的延时是否大于1000。如果存在1000的延时，则返回消费组信息。

- 查询和分析语句

```
* |
SELECT
  consumer_group
FROM  "internal-diagnostic_log"
WHERE
  EXISTS (
    SELECT
     Latency
    FROM     internal-operation_log
    WHERE
      "internal-diagnostic_log".LogStore = "internal-operation_log".logstore and latency >1000
  )
```

- 查找和分析结果

11.2.3. GROUP BY子句

GROUP BY子句用于结合聚合函数，根据一个或多个列对分析结果进行分组。GROUP BY子句还支持搭配ROLLUP子句、CUBE子句和GROUPING SETS子句，扩展分组功能。

语法

- GROUP BY

  对分析结果进行分组。

```
SELECT
 key,
  ...
 aggregate function
 GROUP BY
 key,...
```

- GROUP BY ROLLUP

  GROUP BY ROLLUP子句按照汇总分组，支持为每个分组返回一个小计，为所有分组返回一个总计。例如GROUP BY ROLLUP (a, b)，结果集为(a, b)、(a, null) 和 (null, null) 。

```
SELECT
key,
 ...
aggregate function
GROUP BY ROLLUP (key,...)
```

- GROUP BY CUBE

  GROUP BY CUBE子句按照所有可能的列组合进行分组。例如GROUP BY CUBE (a, b)，结果集为(a, b)、(null, b)、(a, null) 和 (null, null) 。

```
SELECT
key,
...
aggregate function
GROUP BY CUBE (key,...)
```

- GROUP BY GROUPING SETS

GROUP BY GROUPING SETS子句按照列依次进行分组。例如GROUP BY GROUPING SETS (a, b)，结果集为(a, null)和(null, b)。

```
SELECT
key,
...
aggregate function
GROUP BY GROUPING SETS (key,...)
```

注意 在SQL语句中，如果您使用了GROUP BY子句，则在执行SELECT语句时，只能选择GROUP BY的列或者对任意列进行聚合计算，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。正确语句为 `* | SELECT status, arbitrary(request_time), count(*) AS PV GROUP BY status` 。

参数说明

参数	说明
key	日志字段名称或聚合函数计算结果列，即支持按照日志字段名称或聚合函数计算结果列进行分组。 GROUP BY子句支持单列或多列。
aggregate function	聚合函数。GROUP BY子句常与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。

示例

示例1

统计不同状态码对应的请求次数。

- 查询和分析语句

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

- 查询和分析结果

status	PV
205	59
301	41
204	48
402	30
202	68
307	66
401	31
203	64

示例2

按照每小时的时间粒度计算网站访问PV。其中，__time__字段为日志服务中的保留字段，表示时间列。time为 `date_trunc('hour', __time__)` 的别名。date_trunc函数的更多信息，请参见date_trunc函数。

- 查询和分析语句

```
* |
SELECT
  count(*) AS PV,
  date_trunc('hour', __time__) AS time
GROUP BY
  time
ORDER BY
  time
LIMIT
  1000
```

- 查询和分析结果

PV	time
1202	2021-08-10 00:00:00.000
10159	2021-08-10 01:00:00.000
28001	2021-08-10 02:00:00.000

示例3

按照每5分钟的时间粒度计算PV。

- 查询和分析语句

  因为date_trunc函数只能按照固定时间间隔统计。如果您需要按照自定义的时间进行统计分析，请按照数学取模方法进行分组。例如%300表示按照5分钟的时间粒度进行取模对齐。

```
* |
SELECT
  count(*) AS PV,
  __time__-__time__ % 300 AS time
GROUP BY
  time
LIMIT
  1000
```

- 查询和分析结果

PV	time
143	1628525100
31	1628526600
44	1628526900

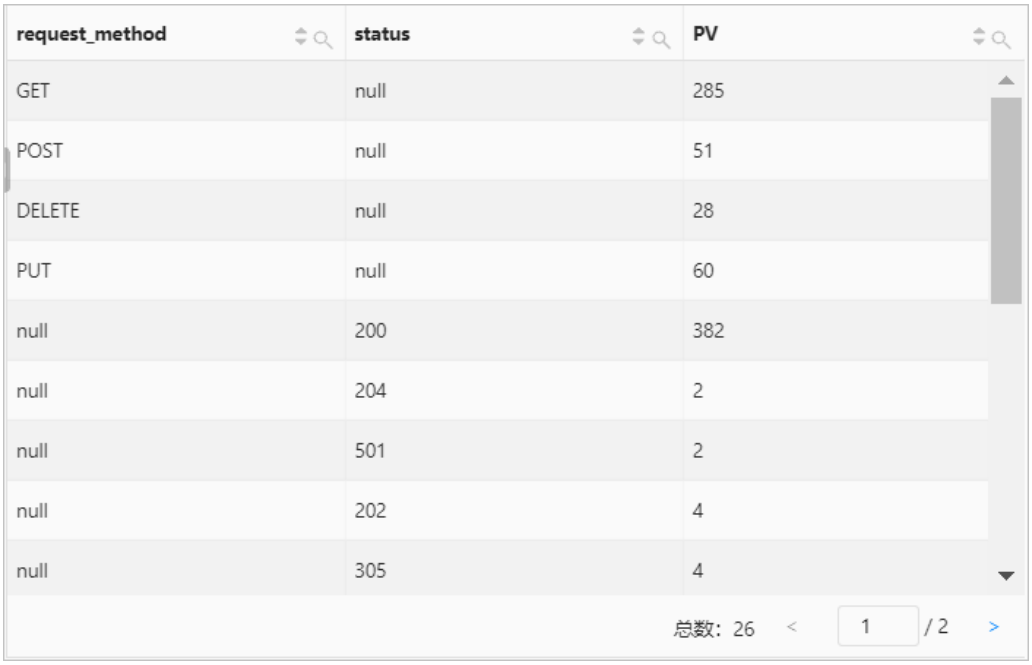
示例4

根据请求方法和请求状态分组，先计算各个请求方法对应的访问数据，再计算各个请求状态对应的访问数量。

- 查询和分析语句

```
* |
SELECT
  request_method,
  status,
  count(*) AS PV
GROUP BY
  GROUPING SETS (request_method, status)
```

- 查询和分析结果



request_method	status	PV
GET	null	285
POST	null	51
DELETE	null	28
PUT	null	60
null	200	382
null	204	2
null	501	2
null	202	4
null	305	4

总数：26 < 1 / 2 >

示例5

根据请求方法和请求状态分组，分组集包括(null, null)、(request_method, null)、(null, status)和(request_method, status)，计算以上各个分组的访问数量。

- 查询和分析语句

```
* |
SELECT
  request_method,
  status,
  count(*) AS PV
GROUP BY
 CUBE (request_method, status)
```

- 查询和分析结果

request_method	status	PV
null	null	285
GET	null	189
PUT	null	39
POST	null	47
DELETE	null	9
HEAD	null	1
null	200	259
null	201	2
null	202	2

总数：43 < 1 / 3 >

示例6

根据请求方法和请求状态分组，分组集包括(request_method, status)、(request_method, null)和(null, null)，计算以上各个分组的访问数量。

- 查询和分析语句

```
* |
SELECT
  request_method,
  status,
  count(*) AS PV
GROUP BY
 ROLLUP (request_method, status)
```

- 查询和分析结果

request_method	status	PV
PUT	205	1
DELETE	305	1
PUT	302	1
POST	null	51
GET	null	162
PUT	null	46
DELETE	null	25
HEAD	null	1
null	null	285

总数：33 < 2 / 2 >

11.2.4. HAVING子句

HAVING子句用于指定过滤分组结果（GROUP BY）或聚合计算结果的条件。

语法

```
HAVING bool_expression
```

> 注意
> - HAVING子句用于过滤分组结果或聚合计算结果，WHERE子句用于在聚合计算之前过滤原始数据。
> - HAVING子句的过滤操作发生在分组（GROUP BY）之后，排序（ORDER BY）之前。

参数说明

参数	说明
bool_expression	布尔表达式。

示例

- 示例1：返回平均请求时长大于40秒的请求地址。
  - 查询和分析语句

```
* |
SELECT
  avg(request_time) AS avg_time,
  request_uri
GROUP BY
  request_uri
HAVING
  avg(request_time) > 40
```

  - 查询和分析结果

avg_time	request_uri
45.03659584919563	/request/path-1/file-0
45.388778550148959	/request/path-3/file-6
45.269188395152408	/request/path-2/file-4
44.91338974614236	/request/path-0/file-0
44.924772223590249	/request/path-2/file-3
45.09470581009704	/request/path-0/file-3
45.013995215311009	/request/path-3/file-4
45.099331306990887	/request/path-3/file-8
44.946835443037979	/request/path-2/file-2

总数：40 < 1 / 2 >

- 示例2：通过服务日志中查询Project的写入情况，返回写入延时大于1000微秒的Project。

- 查询和分析语句

```
method: PostLogstoreLogs |
SELECT
  avg(latency) AS avg_latency,
  Project
GROUP BY
  Project
HAVING
  avg_latency > 1000
```

- 查询和分析结果

avg_latency	Project
1569.909090909091	datala[redacted]engdu

11.2.5. INSERT INTO子句

INSERT INTO子句支持将SQL计算结果写入到同一Project下的其他Logstore中。

语法

```
INSERT INTO target_logstore (key)
SELECT key FROM source_logstore
```

> 注意
>
> - 已在目标Logstore中，为目标字段（例如*key*）创建索引及开启统计功能。
> - *target_logstore*后面必须有待写入的目标字段，例如 `* | INSERT INTO target_logstore SELECT...` 为错误语句。
> - 如果字段的数据类型不匹配，请在SELECT语句中使用类型转换函数转换字段的数据类型。更多信息，请参见类型转换函数。
> - 一次执行最多支持写入10000条数据。
> - 目前只有中国地域支持，海外地域不支持。

参数说明

参数	说明
target_logstore	目标Logstore。 **说明** 目标Logstore和源Logstore不能相同。
source_logstore	源Logstore。
key	字段名或列名。

示例

在名为website_log的Logstore中统计不同状态码的访问次数，然后将统计结果写入到名为test_insert的Logstore中。

> 注意 在执行如下语句前，您需在名为test_insert的Logstore中，为status字段和PV字段创建索引并开启统计功能。

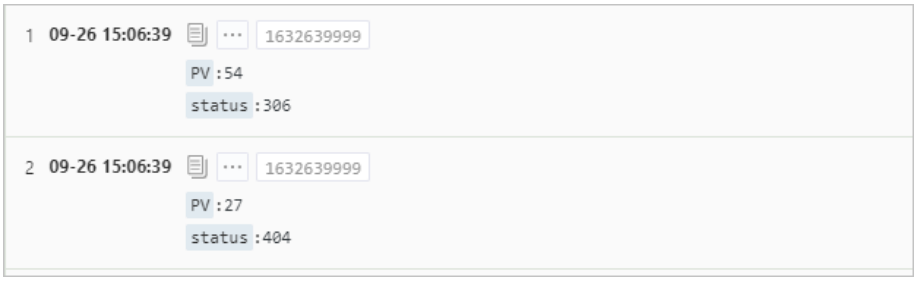
- 查询和分析语句

```
* | INSERT INTO test_insert(status,PV) SELECT status, count(*) AS PV FROM website_log GROUP BY status
```

- 查询和分析结果（源Logstore）

rows	info
22	

- 查询和分析结果（目标Logstore）



11.2.6. INTERSECT子句

INTERSECT子句用于组合两个SELECT子句的结果集，并仅返回两个结果集中共同存在的行。本文介绍INTERSECT子句的基本语法和示例。

语法

```
SELECT key1... FROM logstore1
INTERSECT
SELECT key2... FROM logstore2
```

> 注意
> - 每个SELECT子句必须拥有相同数量的列，对应列的顺序和数据类型相同。
> - INTERSECT子句会删除最终结果集中重复的行，即返回的每一行都是唯一的。

参数说明

参数	说明
key	字段名、列名或表达式。 *key1*和*key2*的名称可不同，但数据类型必须相同。
logstore	Logstore名称。

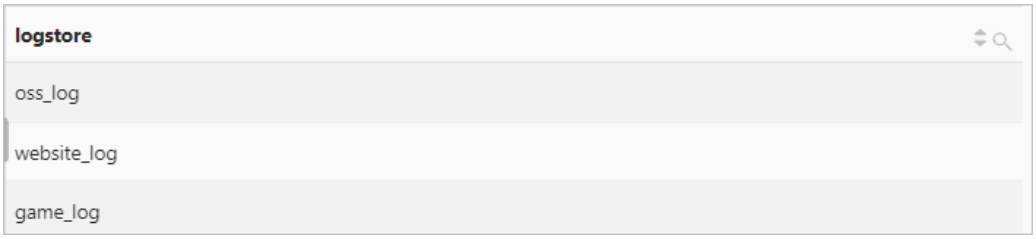
示例

名为internal-diagnostic_log的Logstore用于记录重要日志，包括各个Logstore的消费延时、告警、采集等日志；名为internal-operation_log的Logstore用于记录详细日志，包括Project内所有资源的操作日志。您可以使用INTERSECT子句，查询哪些Logstore即生成了详细日志，又生成了重要日志。

- 查询和分析语句

```
* |
SELECT
  logstore
FROM  internal-operation_log
INTERSECT
SELECT
  logstore
FROM  internal-diagnostic_log
```

- 查询和分析结果



11.2.7. JOIN子句

JOIN子句用于连接多个表。日志服务支持跨Logstore、Logstore和MySQL、Logstore和OSS的联合查询。本文介绍JOIN子句的基本语法和示例。

语法

```
SELECT table.key
FROM table1
INNER|LEFT|RIGHT|FULL OUTER JOIN table2
ON table1.key=table2.key
```

日志服务支持您在SELECT语句中使用INNER JOIN子句、LEFT JOIN子句、RIGHT JOIN子句和OUTER JOIN子句。更多信息，请参见JOIN。

JOIN方式	说明
INNER JOIN	所有表存在交集时，返回满足条件的SELECT结果。
LEFT JOIN	即使右表（table2）中没有匹配的数据，也从左表（table1）返回所有SELECT结果。
RIGHT JOIN	即使左表（table1）中没有匹配的数据，也从右表（table2）返回所有SELECT结果。
FULL OUTER JOIN	只要一个表中存在匹配的数据，则返回满足条件的SELECT结果。

参数说明

参数	说明
key	日志字段、表达式等，参数值为任意数据类型。
table	*table1*为Logstore，*table2*为Logstore、MySQL数据库或OSS Bucket。更多信息，请参见关联MySQL数据源、关联OSS数据源。

示例

名为internal-diagnostic_log的Logstore用于记录各个Logstore的消费延时、告警、采集等日志，名为internal-operation_log的Logstore用于记录Project内所有资源的操作日志。您可以使用JOIN子句，联合查询两个Logstore，从而获取各个Logstore的消费组信息、延时时间和请求方法。

示例1：INNER JOIN

- 查询和分析语句

```
* |
SELECT
  "internal-diagnostic_log".consumer_group,
  "internal-diagnostic_log".logstore,
  "internal-operation_log".Latency,
  "internal-operation_log".Method
FROM  "internal-diagnostic_log"
  INNER JOIN "internal-operation_log" ON "internal-diagnostic_log".logstore = "internal-operation_log".logstore
LIMIT
  10000
```

- 查询和分析结果

  返回满足条件的1328条数据。

consumer_group	logstore	latency	method
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	513	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	591	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	465	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	485	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	550	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	531	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	570	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	539	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	500	PullData

总数：1328 < 1 / 67 >

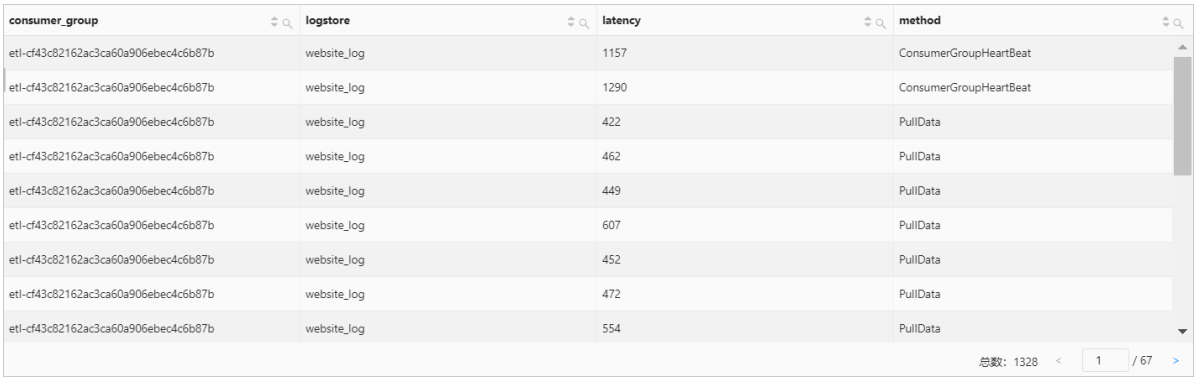
示例2：LEFT JOIN

- 查询和分析语句

```
* |
SELECT
  "internal-diagnostic_log".consumer_group,
  "internal-diagnostic_log".logstore,
  "internal-operation_log".Latency,
  "internal-operation_log".Method
FROM  "internal-diagnostic_log"
  LEFT JOIN "internal-operation_log" ON "internal-diagnostic_log".logstore = "internal-operation_log".logstore
LIMIT
  10000
```

- 查询和分析结果

  从名为internal-diagnostic_log的Logstore中返回1328条数据。



consumer_group	logstore	latency	method
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	1157	ConsumerGroupHeartBeat
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	1290	ConsumerGroupHeartBeat
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	422	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	462	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	449	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	607	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	452	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	472	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	554	PullData

总数：1328 < 1 / 67 >

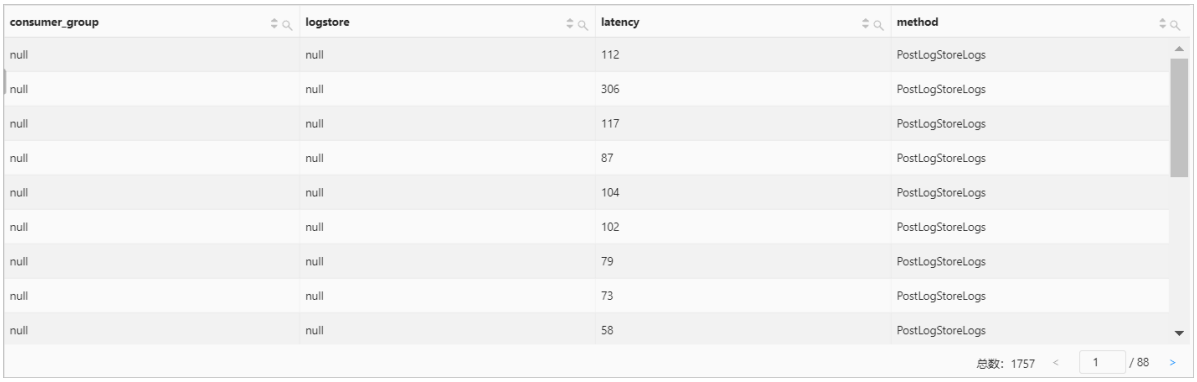
示例3：RIGHT JOIN

- 查询和分析语句

```
* |
SELECT
  "internal-diagnostic_log".consumer_group,
  "internal-diagnostic_log".logstore,
  "internal-operation_log".Latency,
  "internal-operation_log".Method
FROM  "internal-diagnostic_log"
  RIGHT JOIN "internal-operation_log" ON "internal-diagnostic_log".logstore = "internal-operation_
log".logstore
LIMIT
  10000
```

- 查询和分析结果

  从名为internal-operation_log的Logstore中返回1757条数据。



consumer_group	logstore	latency	method
null	null	112	PostLogStoreLogs
null	null	306	PostLogStoreLogs
null	null	117	PostLogStoreLogs
null	null	87	PostLogStoreLogs
null	null	104	PostLogStoreLogs
null	null	102	PostLogStoreLogs
null	null	79	PostLogStoreLogs
null	null	73	PostLogStoreLogs
null	null	58	PostLogStoreLogs

总数：1757 < 1 / 88 >

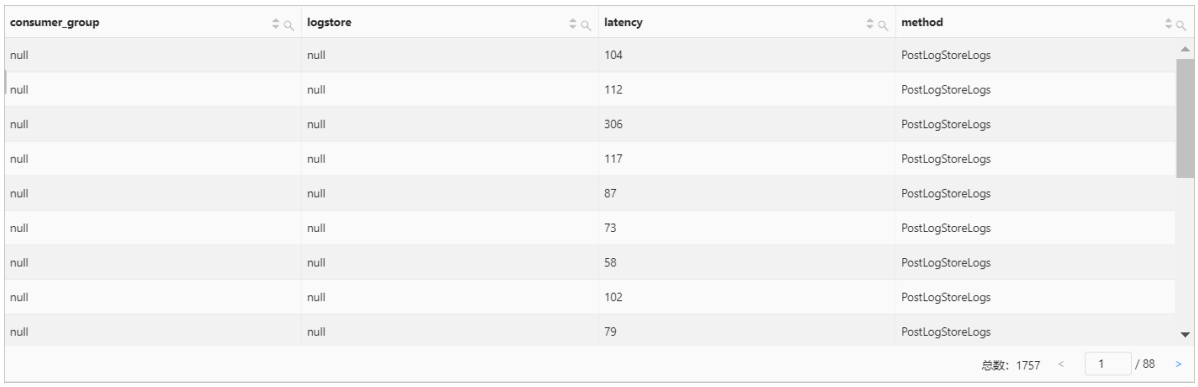
示例4：FULL OUTER JOIN

- 查询和分析语句

```
* |
SELECT
  "internal-diagnostic_log".consumer_group,
  "internal-diagnostic_log".logstore,
  "internal-operation_log".Latency,
  "internal-operation_log".Method
FROM  "internal-diagnostic_log"
  FULL OUTER JOIN "internal-operation_log" ON "internal-diagnostic_log".logstore = "internal-opera
tion_log".logstore
LIMIT
  10000
```

- 查询和分析结果

  返回满足条件的1757条数据。



consumer_group	logstore	latency	method
null	null	104	PostLogStoreLogs
null	null	112	PostLogStoreLogs
null	null	306	PostLogStoreLogs
null	null	117	PostLogStoreLogs
null	null	87	PostLogStoreLogs
null	null	73	PostLogStoreLogs
null	null	58	PostLogStoreLogs
null	null	102	PostLogStoreLogs
null	null	79	PostLogStoreLogs

总数：1757 < 1 / 88 >

11.2.8. LIMIT子句

日志服务默认返回100行计算结果，您也可以使用LIMIT子句指定返回结果的行数。

语法

日志服务支持以下两种LIMIT子句格式。

- 返回计算结果中的前x行数据。

```
LIMIT x
```

- 返回计算结果中从y行开始的x行数据。

```
LIMIT y, x
```

> 注意
>
> - LIMIT子句只用于获取最终的结果，不支持获取SQL中间的结果。
> - 不支持在子查询内部使用LIMIT子句。例如 `* | select count(1) from ( select distinct(url) from limit 0,1000)` 为错误用法。

参数说明

参数	说明
x	指定返回结果中的行数。 • 使用 `LIMIT x` 时，*x*的取值范围为[0,1000000]。 • 使用 `LIMIT y, x` 时，*x*的取值范围为[0,10000]。
y	偏移量。取值范围为[0,1000000]。

> 注意 *x*和*y*之和不能超过1000000。

示例

- 返回计算结果中的前200行。

- 查询和分析语句

```
* | SELECT  request_time LIMIT 200
```

- 查询和分析结果

- 返回计算结果中的第100行到第1100行，共计1000行。
  - 查询和分析语句

```
* | SELECT  request_time LIMIT 100,1000
```

  - 查询和分析结果

- 返回请求时间最长的前3个请求地址。

- 查询和分析语句

```
* |
SELECT
  request_uri AS top_3,
  request_time
ORDER BY
  request_time DESC
LIMIT
  3
```

- 查询和分析结果

top_3	request_time
/request/path-3/file-2	80.0
/request/path-2/file-4	80.0
/request/path-0/file-8	79.0

11.2.9. ORDER BY子句

ORDER BY子句用于根据指定的列名对查询和分析结果进行排序。

语法

```
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，则系统默认对查询和分析结果进行升序排列。
> - 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，可指定多个列进行排序。

参数说明

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，即支持按照日志字段名称（KEY）或聚合函数计算结果列进行排序。
DESC	降序排列。
ASC	升序排列。

示例

- 示例1：统计不同请求状态码对应的请求次数，并按照请求次数降序排列。

- 查询和分析语句

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

- 查询和分析结果

PV	status
163135	200
1224	206
1186	207
1185	305
1184	301
1182	307
1180	302
1177	203

- 示例2：计算写入日志到各个Logstore的平均延迟时间，并按照平均延迟时间进行降序排列。
  - 查询和分析语句

```
method :PostLogstoreLogs |
SELECT
  avg(latency) AS avg_latency,
  LogStore
GROUP BY
  LogStore
ORDER BY
  avg_latency DESC
```

  - 查询和分析结果

avg_latency	Logstore
3833.0	test
2691.133333333333	website_log
2608.0555555555557	date

- 示例3：计算不同请求时长对应的请求数量，并按照请求时长进行升序排序。

  其中，content、time和request_time为JSON日志中的字段。

> 注意
>
> 在查询和分析JSON类型的日志时，需注意以下事项。更多信息，请参见查询和分析JSON日志。
>
> - 必须给字段名称加上JSON中父路径前缀，例如content.time.request_time。
> - 分析语句中的JSON字段名称必须使用双引号（""）包裹，例如"content.time.request_time"。

- 查询和分析语句

```
* |
SELECT
  "content.time.request_time",
  count(*) AS count
GROUP BY
  "content.time.request_time"
ORDER BY
  "content.time.request_time"
```

- 查询和分析结果

content.time.request_time	count
10.0	145
11.0	123
12.0	113

11.2.10. UNION子句

UNION子句用于合并多个SELECT语句的分析结果。

语法

```
SELECT key1 FROM logstore1
UNION
SELECT key2 FROM logstore2
UNION
SELECT key3 FROM logstore3
```

> 注意 UNION内每个SELECT子句必须拥有相同数量的列，对应列的数据类型相同。

参数说明

参数	说明
key	字段名或列名。 *key1*、*key2*和*key3*的名称可不同，但数据类型必须相同。
logstore	Logstore名称。

示例

在名为website_log的Logstore和名为internal-operation_log的Logstore中统计不同状态码的访问次数，然后合并统计结果。

- 查询和分析语句

```
* |
SELECT
  status,
  count(*) AS PV
FROM  website_log
GROUP BY
  status
UNION
SELECT
  status,
  count(*) AS PV
FROM  internal-operation_log
GROUP BY
  status
```

- 查询和分析结果

status	PV
400	1
206	44
200	180
204	50
205	52
301	43
402	30
401	46
400	32

总数：24 < 1 / 2 >

11.2.11. UNNEST子句

在复杂的业务场景下，日志字段的值可能为数组（array）、对象（map）等类型。对这种特殊类型的日志字段进行查询和分析时，您可以先使用UNNEST子句将字段值展开。

语法

- 将array类型的数据展开为多行单列形式，列名为*column_name*。

```
UNNEST(x) AS table_alias(column_name)
```

- 将map类型的数据展开为多行多列形式，列名为*key_name*和*value_name*。

```
UNNEST(y) AS table(key_name,value_name)
```

注意 UNNEST子句处理的是array或者map类型的数据。如果您输入的数据为字符串类型，则需要先转化为JSON类型，然后再转化为array类型或map类型，转化方法为 `try_cast(json_parse(array_column) as array(bigint))` 。更多信息，请参见类型转换函数。

参数说明

参数	说明
x	数据类型为array类型。
column_name	将array类型的数据展开后，指定一个列名。该列用于存放array中的元素。
y	数据类型为map类型。
key_name	将map类型的数据展开后，指定一个列名。该列用于存放map中的键。
value_name	将map类型的数据展开后，指定一个列名。该列用于存放map中的键值。

示例

示例1

将number字段的值（array类型）展开为多行单列形式。

- 字段样例

```
number:[49, 50, 45, 47, 50]
```

- 查询和分析语句

```
* |
SELECT
  a
FROM  log,
  UNNEST(cast(json_parse(number) AS array(bigint))) AS t(a)
```

- 查询和分析结果

a
49
50
45
47
50

示例2

将number字段的值（array类型）展开为多行单列形式，并进行求和计算。

- 字段样例

  此处仅提供一条日志样例，求和计算是针对所有日志，即对所有日志中的number字段的值进行求和。

```
number:[49, 50, 45, 47, 50]
```

- 查询和分析语句

```
* |
SELECT
  sum(a) AS sum
FROM  log,
  UNNEST(cast(json_parse(number) as array(bigint))) AS t(a)
```

- 查询和分析结果

sum
368248

示例3

将number字段的值（array类型）展开为多行单列形式，并对各个值进行分组统计。

- 字段样例

```
number:[49, 50, 45, 47, 50]
```

- 查询和分析语句

```
* |
SELECT
  a, count(*) AS count
FROM  log,
  UNNEST(cast(json_parse(number) as array(bigint))) AS t(a) GROUP BY a
```

- 查询和分析结果

a	count
50	1194
47	597
49	597
45	597

示例4

将number字段的值（map类型）展开为多行多列形式。

- 字段样例

```
result:{
  anomaly_type:"OverThreshold"
  dim_name:"request_time"
  is_anomaly:true
  score:1
  value:"3.000000"}
```

- 查询和分析语句

```
* |
select
  key,
  value
FROM  log,
  UNNEST(
    try_cast(json_parse(result) as map(varchar, varchar))
  ) AS t(key, value)
```

- 查询和分析结果

key	value
anomaly_type	OverThreshold
dim_name	request_time
is_anomaly	true
score	1
value	33.000000

示例5

将number字段的值（map类型）展开为多行多列形式，并对各个键进行分组统计。

- 字段样例

```
result:{
  anomaly_type:"OverThreshold"
  dim_name:"request_time"
  is_anomaly:true
  score:1
  value:"3.000000"}
```

- 查询和分析语句

```
* |
select
  key,
  count(*) AS count
FROM  log,
  UNNEST(
    try_cast(json_parse(result) as map(varchar, varchar))
  ) AS t(key, value)
GROUP BY
  key
```

- 查询和分析结果

key	count
anomaly_type	5422
dim_name	5422
is_anomaly	5422
score	5422
value	5422

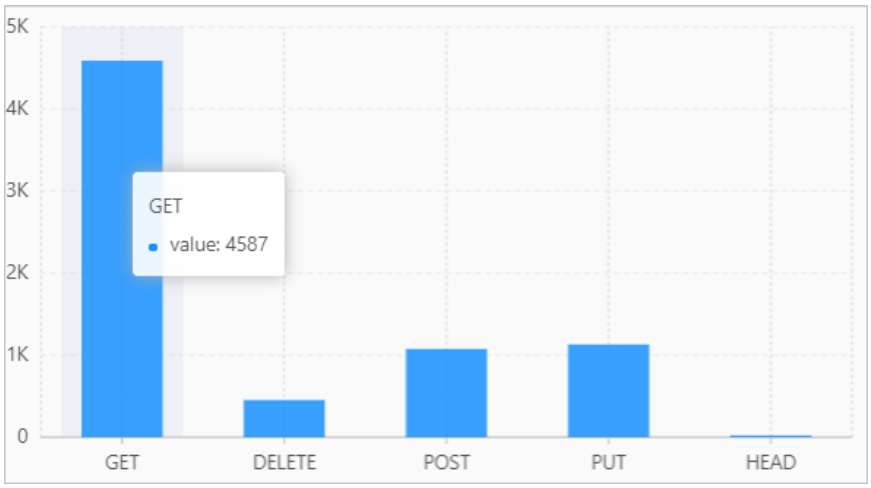
示例6

使用histogram函数获取各个请求方法对应的请求数量，返回结果为map类型。然后通过unnest子句将histogram函数的返回结果展开为多行多列形式，并通过柱状图展示。

- 查询和分析语句

```
* |
SELECT
  key,
  value
FROM(
    SELECT
      histogram(request_method) AS result
    FROM      log
  ),
  UNNEST(result) AS t(key, value)
```

- 查询和分析结果



11.2.12. VALUES子句

VALUES子句用于构造数据，您可以通过VALUES子句向表中插入少量的临时数据用于查询与分析。本文介绍VALUES子句的基本语法和示例。

语法

```
VALUES(column_value01, column_value02...) table_name(column_name01,column_name02...)
```

参数说明

参数	说明
column_value	待插入的列值，支持常量、表达式、函数等。
table_name	表名，VALUES子句将数据插入到该表中。
column_name	列名，VALUES子句将数据插入到该列中。

示例

使用VALUES子句向名为access的表中插入一列数据，列名为pv。

- 查询和分析语句

```
* |
SELECT
  pv
FROM  (
    VALUES
      (0),
      (1),
      (2),
      (3),
      (4),
      (5),
      (6),
      (7),
      (8),
      (9)
  ) AS access(pv)
```

- 查询和分析结果

11.2.13. WITH子句

WITH子句支持将子查询结果保存到临时表中，从而实现后续的SQL分析可在临时表中执行。通过WITH子句可简化SQL语句，提高可阅读性。本文介绍WITH子句的基本语法和示例。

语法

```
WITH table_name AS (select_statement) select_statement
```

参数说明

参数	说明
table_name	临时表名称。
select_statement	完整的SELECT语句。

示例

在名为website_log的Logstore中分析每台主机对应的平均请求长度，并将分析结果保存到表T1中；在名为access_log的Logstore中分析每台主机对应的平均请求长度，并将分析结果保存到表T2中。然后联合查询表T1和表T2，获取两个表中相同主机对应的平均请求长度。

- 查询和分析语句

```
* | with T1 AS (
  SELECT
    host,
    avg(request_length) length
  FROM    website_log
  GROUP BY
    host
),
T2 AS (
  SELECT
    host,
    avg(request_length) length
  FROM    access_log
  GROUP BY
    host
)
SELECT
  T1.host,
  T1.length,
  T2.length
FROM  T1
  JOIN T2 ON T1.host = T2.host
```

- 查询和分析结果

host	length	length
www.l[redacted]com	4150.775370581528	4150.775370581528
www.f[redacted]com	3767.3465346534654	3767.3465346534654
www.v[redacted]k.com	4252.855555555556	4252.855555555556
www.l[redacted].com	3665.181818181818	3665.181818181818
www.e[redacted]com	4227.140536149472	4227.140536149472
www.x[redacted]com	4197.650477707007	4197.650477707007
www.l[redacted].com	3556.4367816091954	3556.4367816091954
www.c[redacted].com	4124.819313466616	4124.819313466616

11.3. 保留字

本文介绍日志服务SQL分析语句中所有的保留字。

```
AND
AS
BETWEEN
BY
CASE
CAST
CROSS
CUBE
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
DISTINCT
ELSE
END
ESCAPE
EXCEPT
EXISTS
FROM
GROUP
GROUPING
HAVING
IN
INNER
INSERT
INTERSECT
INTO
IS
JOIN
LEFT
LIKE
LIMIT
LOCALTIME
LOCALTIMESTAMP
NATURAL
NOT
NULL
ON
OR
ORDER
OUTER
RIGHT
ROLLUP
SELECT
THEN
TRUE
UNION
UNNEST
VALUES
WHEN
WHERE
WITH
```

11.4. 列的别名

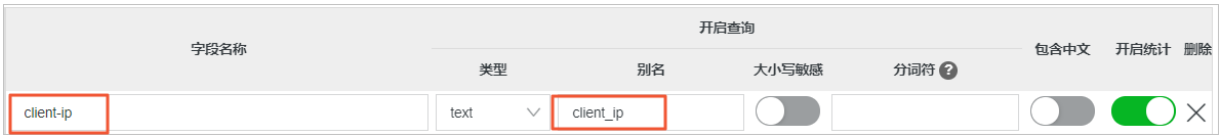
本文介绍别名的规范和示例。

别名规范

在SQL标准中，列名必须由字母、数字和下划线（_）组成，且以字母开头。如果您在采集日志时设置了不符合SQL92语法的列名，则需要在配置索引时为目标列名设置一个别名。如何配置索引，请参见配置索引。

当日志中的原始列名特别长时，您也可以设置一个简短的别名，用于SQL分析。

> 注意 别名仅仅用于SQL分析，底层的存储仍然使用的是原始列名。即查询语句中使用原始列名，分析语句中使用别名。



示例

原始列名	别名
User-Agent	User_Agent
User.Agent	user_agent
123	col
abceefghijklmnopqrstuvw	abc

11.5. 嵌套子查询

嵌套子查询是指将一个SELECT语句嵌套在另一个SELECT语句中。针对复杂的分析场景，您可以使用嵌套子查询。

基本语法

使用嵌套子查询时，需在SELECT语句中指定FROM子句。

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

> 注意
> - 子查询语句需被包裹在半角圆括号 `()` 中。
> - 在子查询语句中，需指定关键字 `FROM log` ，表示在当前Logstore中执行SQL分析。

示例

示例1

计算各个请求方法对应的请求数量，然后获取最小的请求数量。

- 查询和分析语句

```
* |
SELECT
 min(PV)
FROM  (
    SELECT
      count(1) as PV
    FROM      log
    GROUP BY
      request_method
  )
```

- 查询和分析结果

min
13

示例2

计算当前1小时和昨天同时段的网站访问量比值。其中，选择查询和分析的时间范围为**1小时（整点时间）**，86400表示当前时间减去86400秒（1天），log表示Logstore名称。

- 查询和分析语句

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

- 查询和分析结果

today	yesterday	ratio
3337.0	3522.0	0.947473026689381

  - **3337.0**表示当前1小时（例如2020-12-25 14:00:00~2020-12-25 15:00:00）的网站访问量。
  - **3522.0**表示昨天同时段（例如2020-12-24 14:00:00~2020-12-24 15:00:00）的网站访问量。
  - **0.947473026689381**表示当前1小时与昨天同时段的网站访问量比值。

示例3

统计各个访问页面的访问次数及占比。

- 查询和分析语句

```
* |
SELECT
  request_uri AS "访问页面",
  c AS "次数",
  round(c * 100.0 /(sum(c) over()), 2) AS "百分比%"
FROM  (
    SELECT
      request_uri AS request_uri,
      count(*) AS c
    FROM      log
    GROUP BY
      request_uri
    ORDER BY
      c DESC
  )
```

- 查询和分析结果

访问页面	次数	百分比%
/request/path-1/file-2	92	3.37
/request/path-1/file-8	87	3.19
/request/path-3/file-5	80	2.93
/request/path-1/file-5	80	2.93
/request/path-3/file-9	78	2.86
/request/path-1/file-7	74	2.71
/request/path-2/file-4	73	2.68
/request/path-1/file-4	72	2.64
/request/path-2/file-2	72	2.64

11.6. Logstore和MySQL联合查询

日志服务支持通过Join语法将Logstore和MySQL数据库进行联合查询，并把查询结果保存到MySQL数据库中。

前提条件

已创建ExternalStore。具体操作，请参见关联MySQL数据源。

操作步骤

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 执行查询分析语句。

   支持的Join语法有INNER JOIN、LEFT JOIN、RIGHT JOIN和FULL JOIN。

```
[ INNER ] JOIN
LEFT [ OUTER ] JOIN
RIGHT [ OUTER ] JOIN
FULL [ OUTER ] JOIN
```

JOIN语法样例如下所示。更多信息，请参见关联Logstore与MySQL数据库进行查询分析。

```
method:postlogstorelogs | select count(1) , histogram(logstore) from log  l join join_meta m on l
.projectid = cast( m.ikey as varchar)
```

> **注意**
>
> - 仅支持Logstore与MySQL数据库小表（数据量小于20 MB）进行联合查询。
> - 查询和分析语句中，Logstore必须写在join关键字前面，ExternalStore写在join关键字后面。
> - 查询和分析语句中，必须写ExternalStore名称，系统自动替换成MySQL数据库名+表名。请勿直接填写MySQL表名。

5. 保存查询结果到MySQL数据库中。

   日志服务支持通过Insert语法将查询结果插入到MySQL数据库中。Insert语法样例如下所示：

```
method:postlogstorelogs | insert into method_output  select cast(method as varchar(65535)),count(
1) from log group by method
```

Python程序样例

```
# encoding: utf-8
from __future__ import print_function
from aliyun.log import *
from aliyun.log.util import base64_encodestring
from random import randint
import time
import os
from datetime import datetime
    endpoint = os.environ.get('ALIYUN_LOG_SAMPLE_ENDPOINT', 'cn-chengdu.log.aliyuncs.com')
    accessKeyId = os.environ.get('ALIYUN_LOG_SAMPLE_ACCESSID', '')
    accessKey = os.environ.get('ALIYUN_LOG_SAMPLE_ACCESSKEY', '')
    logstore = os.environ.get('ALIYUN_LOG_SAMPLE_LOGSTORE', '')
    project = "ali-yunlei-chengdu"
    client = LogClient(endpoint, accessKeyId, accessKey, token)
    #创建ExternalStore。
    res = client.create_external_store(project,ExternalStoreConfig("rds_store","region","rds-vpc","vp
c id","实例id","实例ip","实例端口","用户名","密码","数据库","数据库表"));
    res.log_print()
    #获取ExternalStore详情。
    res = client.get_external_store(project,"rds_store");
    res.log_print()
    res = client.list_external_store(project,"");
    res.log_print();
    # JOIN查询。
    req = GetLogsRequest(project,logstore,From,To,"","select count(1) from  "+ logstore +"  s join  m
eta m on  s.projectid = cast(m.ikey as varchar)");
    res = client.get_logs(req)
    res.log_print();
     # 将查询和分析结果写入MySQL数据库。
    req = GetLogsRequest(project,logstore,From,To,""," insert into rds_store select count(1) from  "+
logstore );
    res = client.get_logs(req)
    res.log_print();
```

12.机器学习语法与函数

12.1. 概述

日志服务机器学习功能为您提供多种功能丰富的算法和便捷的调用方式，您可以在日志查询分析中通过分析语句和机器学习函数调用机器学习算法，分析某一字段或若干字段在一段时间内的特征。

针对时序数据分析场景，日志服务提供了丰富的时序分析算法，可以帮助您快速解决时序预测、时序异常检测、序列分解、多时序聚类等场景问题，兼容SQL标准接口，大大降低了您使用算法的门槛，提高分析问题和解决问题的效率。

功能特点

- 支持单时序序列的多种平滑操作。
- 支持单时序序列的预测、异常检测、变点检测、折点检测、多周期估计算法。
- 支持单时序序列的分解操作。
- 支持多时序序列的多种聚类算法。
- 支持多字段（数值列、文本列）的模式挖掘。

使用限制

使用日志服务机器学习函数须遵循以下限制：

- 输入的时序数据必须是基于相同时间间隔的采样数据。
- 输入的时序数据中不能含有重复时间点的数据。
- 处理容量限制。

限制项	说明
时序数据处理的有效容量	上限为150,000个连续时间点数据。 若数量超过上限，请进行聚合操作或者降采样操作。
密度聚类算法的聚类容量	上限为5000条时序曲线，每条时序曲线的长度最大为1440个点。
层次聚类算法的聚类容量	上限为2000条时序曲线，每条时序曲线的长度最大为1440个点。

机器学习函数

类别		函数	说明
	平滑函数	ts_smooth_simple	使用Holt Winters算法对时序数据平滑。
		ts_smooth_fir	使用FIR滤波器对时序数据平滑。
		ts_smooth_iir	使用IIR滤波器对时序数据平滑。
	多周期估计函数	ts_period_detect	对时序数据进行分段周期估计。
	变点检测函数	ts_cp_detect	寻找时序序列中具有不同统计特性的区间，区间端点即为变点。
		ts_breakout_detect	寻找时序序列中，某统计量发生陡升或陡降的点。

类别		函数	说明
时间序列	极大值检测函数	ts_find_peaks	极大值检测函数用于在指定窗口中寻找序列的局部极大值。
	预测与异常检测函数	ts_predicate_simple	利用默认参数对时序数据进行建模，并进行简单的时序预测和异常点的检测。
		ts_predicate_ar	使用自回归模型对时序数据进行建模，并进行简单的时序预测和异常点的检测。
		ts_predicate_arma	使用移动自回归模型对时序数据进行建模，并进行简单的时序预测和异常点检测。
		ts_predicate_arima	使用带有差分的移动自回归模型对时序数据进行建模，并进行简单的时序预测和异常点检测。
		ts_regression_predict	针对含有周期性、趋势性的单时序序列，进行准确且长时序预测。
	序列分解函数	ts_decompose	使用STL算法对时序数据进行序列分解。
	时序聚类函数	ts_density_cluster	使用密度聚类方法对多条时序数据进行聚类。
		ts_hierarchical_cluster	使用层次聚类方法对多条时序数据进行聚类。
		ts_similar_instance	查找到指定曲线名称的相似曲线。
	核密度估计函数	kernel_density_estimation	核密度估计函数采用平滑的峰值函数来拟合观察到的数据点，从而对真实的概率分布曲线进行模拟。
	时序补点函数	series_padding	如果时间序列中存在数据缺失问题，可以使用时序补点函数补齐缺失的数据。
	异常对比函数	anomaly_compare	用于比较某个观测对象在两个时间段的差异程度。
模式挖掘	频繁模式统计	pattern_stat	统计模式中的频繁模式，在给定的多属性字段样本中，挖掘出具有一定代表性的属性组合。
	差异模式统计	pattern_diff	在指定条件下找出导致两个集合差异的模式。
	根因分析函数	rca_kpi_search	在时序指标发生异常时，根因分析函数可以快速分析出是哪些相关维度属性发生异常而导致监控指标发生异常。
	相关性分析函数	ts_association_analysis	针对系统中的多个观测指标，快速找出和某个指标项相关的指标名称。
		ts_similar	针对系统中的多个观测指标，快速找出和用户输入的时序序列相关的指标名称。

类别		函数	说明
	URL请求分类函数	url_classify	URL请求分类函数会自动将您输入的URL请求路径进行归类打标签，并提供类别的正则表达式，帮助您更好的归类URL。

12.2. 平滑函数

平滑函数是针对输入的时序曲线进行平滑和简单的滤波操作，滤波操作通常是发现时序曲线形态的第一步。

函数列表

函数	说明
ts_smooth_simple	默认平滑函数，使用Holt Winters算法对时序数据进行滤波操作。
ts_smooth_fir	使用FIR滤波器对时序数据进行滤波操作。
ts_smooth_iir	使用IIR滤波器对时序数据进行滤波操作。

ts_smooth_simple

- 函数格式：

```
select ts_smooth_simple(x, y)
```

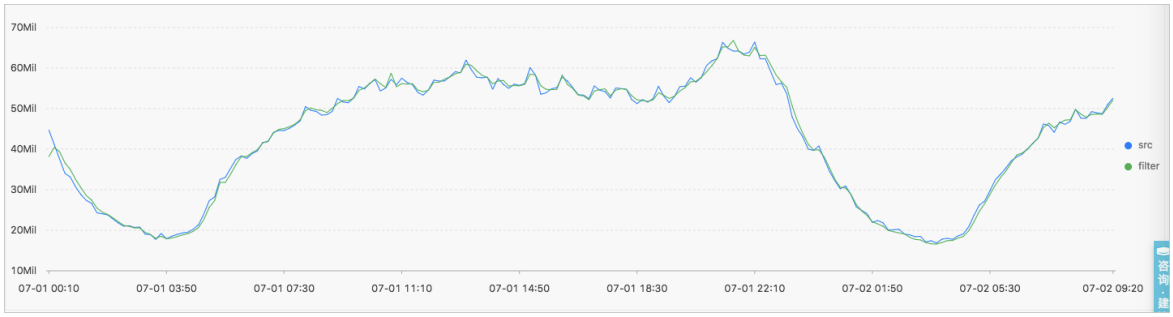
- 参数说明：

参数	说明	取值
x	时间列，顺序为从小到大。	Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-

- 示例
  - 查询分析语句：

```
* | select ts_smooth_simple(stamp, value) from ( select __time__ - __time__ % 120 as stamp, avg(v) as value from log GROUP BY stamp order by stamp )
```

  - 输出结果：



- 显示项：

显示项		说明
横轴	unixtime	数据的时间戳，单位为秒。
纵轴	src	滤波前的数据。
	filter	滤波后的数据。

ts_smooth_fir

- 函数格式：
  - 若您无法确定滤波参数，请使用内置窗口的参数进行滤波操作。

    ```
    select ts_smooth_fir(x, y,winType,winSize)
    ```

  - 若您可以确定滤波参数，可以根据需求自定义设置滤波参数。

    ```
    select ts_smooth_fir(x, y,array[])
    ```

- 参数说明：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
winType	滤波的窗口类型。	取值包括： ○ rectangle：矩形窗口。 ○ hanning：汉宁窗。 ○ hamming：汉明窗。 ○ blackman：布莱克曼窗。 **说明** 推荐您选择rectangle类型以获得更好的显示效果。
winSize	滤波窗口的长度。	long类型，取值范围为2~15。
array[]	FIR滤波的具体参数。	格式为数组，且数组中元素的和为1。例如array[0.2, 0.4, 0.3, 0.1]。

- 示例1
  - 查询分析语句：

    ```
    * | select ts_smooth_fir(stamp, value, 'rectangle', 4) from ( select __time__ - __time__ % 120 a
    s stamp, avg(v) as value from log GROUP BY stamp order by stamp )
    ```

- 输出结果：

- 示例2
  - 查询分析语句：

```
* | select ts_smooth_fir(stamp, value, array[0.2, 0.4, 0.3, 0.1]) from ( select __time__ - __tim
e__ % 120 as stamp, avg(v) as value from log GROUP BY stamp order by stamp )
```

  - 输出结果：

- 显示项：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
纵轴	src	滤波前的数据。
	filter	滤波后的数据。

ts_smooth_iir

- 函数格式：

```
select ts_smooth_iir(x, y, array[], array[] )
```

- 参数说明：

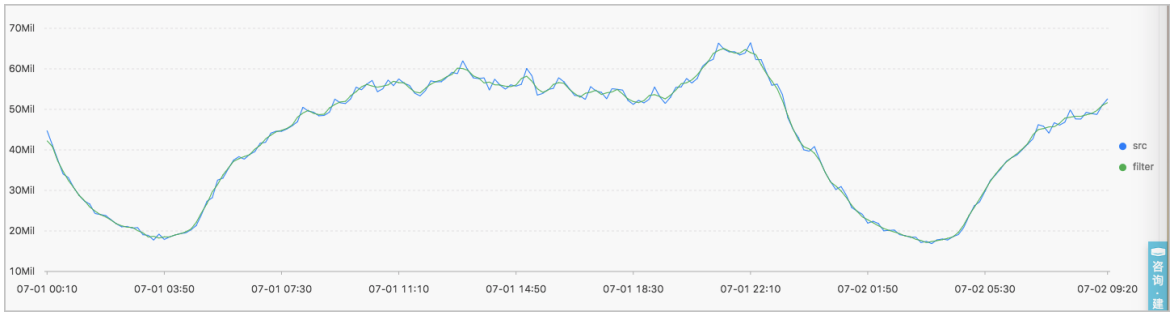
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
array[]	IIR滤波算法中关于x ¡的具体参数。	数组格式，长度（length）的取值范围为2~15，且数组中元素的和为1。例如array[0.2, 0.4, 0.3, 0.1]。

参数	说明	取值
array[]	IIR滤波算法中关于y_{i-1}的具体参数。	数组格式，长度（length）的取值范围为2~15，且数组中元素的和为1。例如array[0.2, 0.4, 0.3, 0.1]。

- 示例
  - 查询分析语句：

```
* | select ts_smooth_iir(stamp, value, array[0.2, 0.4, 0.3, 0.1], array[0.4, 0.3, 0.3]) from ( select __time__ - __time__ % 120 as stamp, avg(v) as value from log GROUP BY stamp order by stamp )
```

  - 输出结果：



- 显示项：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
纵轴	src	滤波前的数据。
	filter	滤波后的数据。

12.3. 多周期估计函数

多周期估计函数支持对不同时间段内的时序进行周期估计，通过傅立叶变换等一系列操作进行周期的提取。

函数列表

函数	说明
ts_period_detect	对不同时间段内的时序数据进行周期估计。
ts_period_classify	通过傅立叶变换，计算输入时序曲线的周期性。该方法可以较好的用于快速判别曲线的周期性。

ts_period_detect

函数格式：

```
select ts_period_detect(x,y,minPeriod,maxPeriod)
```

参数说明如下：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	无
minPeriod	根据您的时序曲线，预估计周期最小长度占序列总长度的比例。	小数形式，取值范围为(0,1]。
maxPeriod	根据您的时序曲线，预估计周期最大长度占序列总长度的比例。 **注意** *maxPeriod*必须大于*minPeriod*，且不超过0.5。如果您设置的*maxPeriod*超过0.5，系统默认按0.5执行。	小数形式，取值范围为(0,1]。

示例：

- 查询分析

```
* | select ts_period_detect(stamp, value, 0.2, 0.5) from ( select __time__ - __time__ % 120 as sta
mp, avg(v) as value from log GROUP BY stamp order by stamp )
```

- 返回结果

  返回结果为数组类型，包括Unix时间戳、统计的数值（例如平均流量）以及周期状态码（一个1.0状态码代表下图中的一个红圈）。您可以使用时序图展示返回结果，如下图所示：

  下图中两个红圈之间的阴影部分表示一个周期，各个周期间的曲线趋于相同。

ts_period_classify

函数格式：

```
select ts_period_classify(stamp,value,instanceName)
```

参数说明如下：

参数	说明	取值
stamp	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
value	数值列，对应某时刻的数据。	无
instanceName	曲线对应的名称。	无

示例：

- 查询分析

```
* and h : nu2h05202.nu8 | select ts_period_classify(stamp, value, name) from log
```

- 返回结果

显示项如下：

显示项	说明
line_name	曲线名称。
prob	时序曲线中主周期的占比，范围为[0, 1]，实验中可以取0.15。
type	曲线的类别： • type = -1：表示曲线长度太短（小于64个点）。 • type = -2：表示曲线缺失率很高（缺失率超过20%）。 • type = 0：表示曲线有明显的周期性。

12.4. 变点检测函数

变点检测函数一般用于对时序数据中的变点进行检测。

变点检测函数支持对如下两种变点形态进行检测：

- 指定时间段内的某些统计特性发生了变化。
- 序列数据中存在较为明显的断层。

函数列表

函数	说明
ts_cp_detect	寻找时序序列中具有不同统计特性的区间，区间端点即为变点。
ts_breakout_detect	寻找时序序列中，某统计量发生陡升或陡降的点。

ts_cp_detect

函数格式：

- 若您无法确定窗口大小，可以使用如下格式的函数，该函数调用的算法默认会使用长度为10的窗口进行检测。

```
select ts_cp_detect(x, y, samplePeriod)
```

- 若您需要根据业务曲线进行效果调试，可以使用如下格式的函数，通过设置参数minSize进行效果调试。

```
select ts_cp_detect(x, y, minSize)
```

参数说明如下：

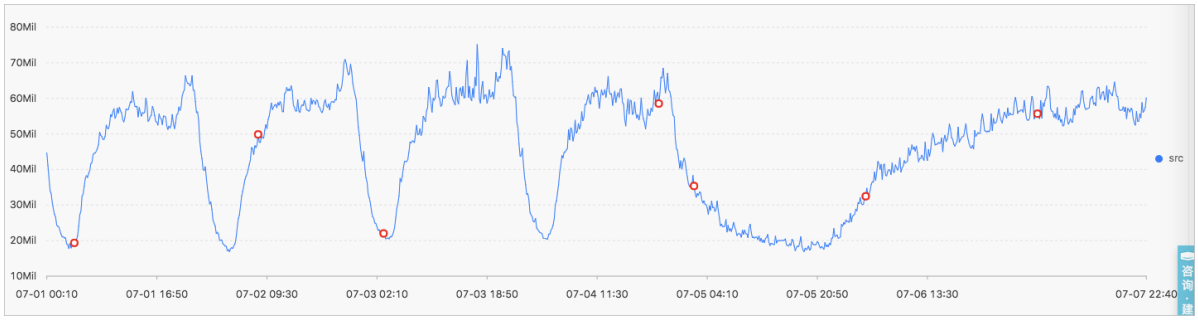
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
minSize	最小连续区间长度。	最小值为3，最大值不超过当前输入数据长度的1/10。

示例：

- 查询分析

```
* | select ts_cp_detect(stamp, value, 3) from (select __time__ - __time__ % 10 as stamp, avg(v) as value from log GROUP BY stamp order by stamp)
```

- 输出结果



显示项如下：

显示项		说明
横轴	unixtime	数据的时间戳，单位为秒，例如1537071480。
纵轴	src	滤波前的数据，例如1956092.7647745228。
	prob	该点为变点的概率，取值范围为0~1。

ts_breakout_detect

函数格式：

```
select ts_breakout_detect(x, y, winSize)
```

参数说明如下：

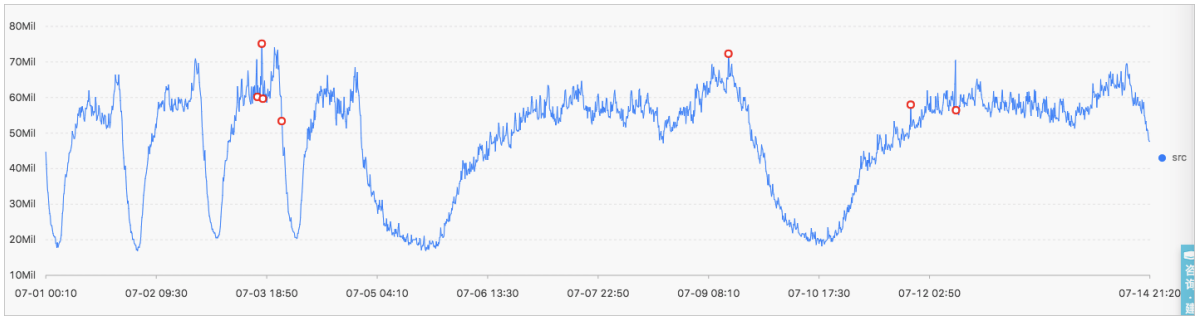
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
winSize	最小连续区间长度。	最小值为3，最大值不超过当前输入数据长度的1/10。

示例：

- 查询分析

```
* | select ts_breakout_detect(stamp, value, 3) from (select __time__ - __time__ % 10 as stamp, avg(v) as value from log GROUP BY stamp order by stamp)
```

- 输出结果



显示项如下：

显示项		说明
横轴	unixtime	数据的时间戳，单位为秒，例如1537071480。
纵轴	src	滤波前的数据，例如1956092.7647745228。
	prob	该点为变点的概率，取值范围为0~1。

12.5. 极大值检测函数

极大值检测函数用于在指定窗口中寻找序列的局部极大值。

ts_find_peaks

函数格式：

```
select ts_find_peaks(x, y, winSize)
```

参数说明如下：

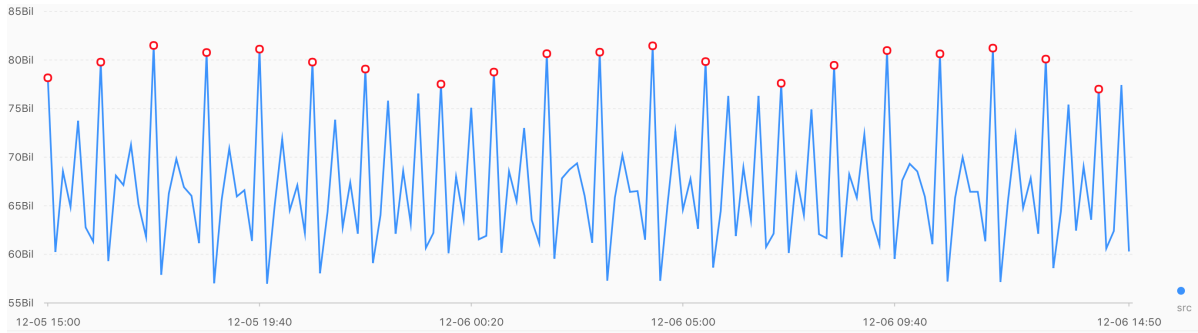
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
winSize	指定最小的检测窗口长度。	long类型，取值范围为大于等于1，小于等于数值的实际长度。建议指定该参数的值为数据实际长度的十分之一。

示例：

- 查询分析：

```
* and h : nu2h05202.nu8 and m: NET |  select ts_find_peaks(stamp, value, 30) from (select __time__ - __time__ % 10 as stamp, avg(v) as value from log GROUP  BY  stamp order by stamp)
```

- 输出结果：



显示项如下：

显示项		说明
横轴	unixtime	数据的时间戳，单位为秒，例如1537071480。
纵轴	src	未滤波前的数据，例如1956092.7647745228。
	peak_flag	该点是否为极大值，其中： • 1.0：表示该点为极大值。 • 0.0：表示该点不是极大值。

12.6. 预测与异常检测函数

预测与异常检测函数通过预测时序曲线、寻找预测曲线和实际曲线之间误差的Ksigma与分位数等特性进行异常检测。

关于函数的算法及原理请参见：

- LOG机器学习介绍（01）：时序统计建模
- LOG机器学习介绍（03）：时序异常检测建模
- LOG机器学习介绍（05）：时间序列预测
- LOG机器学习最佳实战：时序异常检测和报警

函数列表

函数	说明
ts_predicate_simple	利用默认参数对时序数据进行建模，并进行简单的时序预测和异常点的检测。
ts_predicate_ar	使用自回归模型对时序数据进行建模，并进行简单的时序预测和异常点的检测。
ts_predicate_arma	使用移动自回归模型对时序数据进行建模，并进行简单的时序预测和异常点检测。
ts_predicate_arima	使用带有差分的移动自回归模型对时序数据进行建模，并进行简单的时序预测和异常点检测。
ts_regression_predict	针对具有周期性、趋势性的单时序序列，进行准确的预测。 使用场景：计量数据的预测、网络流量的预测、财务数据的预测、以及具有一定规律的不同业务数据的预测。
ts_anomaly_filter	针对批量曲线进行时序异常检测后，可以按照用户定义的异常模式来过滤异常检测的结果。能帮助用户快速找出异常的实例曲线。

ts_predicate_simple

函数格式：

```
select ts_predicate_simple(x, y, nPred, isSmooth)
```

参数说明如下：

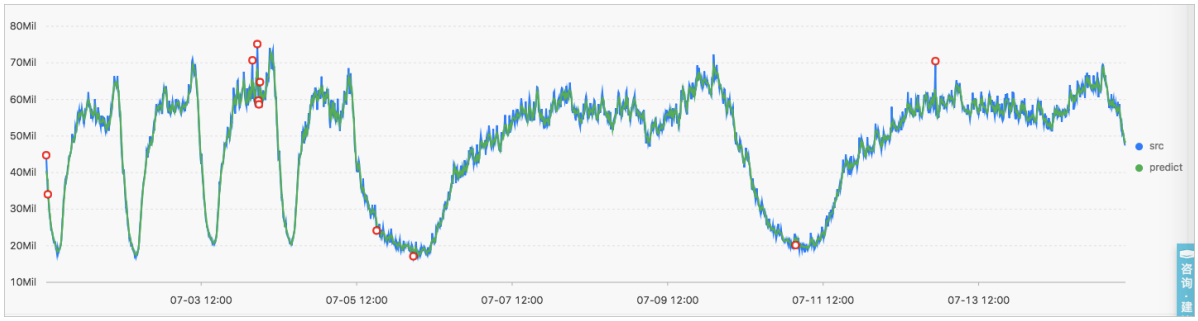
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
nPred	预测未来的点的数量。	long类型，取值大于等于1。
isSmooth	是否需要对原始数据做滤波操作。	bool类型，默认为true表示对原始数据做滤波操作。

示例：

- 查询分析

```
* | select ts_predicate_simple(stamp, value, 6) from (select __time__ - __time__ % 60 as stamp, av
g(v) as value from log GROUP BY stamp order by stamp)
```

- 输出结果



显示项如下：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
纵轴	src	原始数据。
	predict	预测的数据。
	upper	预测的上界。当前置信度为0.85，不可修改。
	lower	预测的下界。当前置信度为0.85，不可修改。
	anomaly_prob	该点为异常点的概率，范围为0~1。

ts_predicate_ar

函数格式：

```
select ts_predicate_ar(x, y, p, nPred, isSmooth)
```

参数说明如下：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
p	自回归模型的阶数。	long类型，取值范围为2~8。
nPred	预测未来的点的数量。	long类型，取值范围为1~5*p。
isSmooth	是否需要对原始数据做滤波操作。	bool类型，默认为true表示对原始数据做滤波操作。

查询分析示例：

```
* | select ts_predicate_ar(stamp, value, 3, 4) from (select __time__ - __time__ % 60 as stamp, avg(v) as value from log GROUP BY stamp order by stamp)
```

> **说明** 输出结果与ts_predicate_simple函数相似，具体请参见ts_predicate_simple函数的输出结果。

ts_predicate_arma

函数格式：

```
select ts_predicate_arma(x, y, p, q, nPred, isSmooth)
```

参数说明如下：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
p	自回归模型的阶数。	long类型，取值范围为2~100。
q	移动平均模型的阶数。	long类型，取值范围为2~8。
nPred	预测未来的点的数量。	long类型，取值范围为1~5*p。
isSmooth	是否需要对原始数据做滤波操作。	bool类型，默认为true表示对原始数据做滤波操作。

查询分析示例：

```
* | select ts_predicate_arma(stamp, value, 3, 2, 4) from (select __time__ - __time__ % 60 as stamp, avg(v) as value from log GROUP BY stamp order by stamp)
```

> **说明** 输出结果与ts_predicate_simple函数相似，具体请参见ts_predicate_simple函数的输出结果。

ts_predicate_arima

函数格式：

```
select ts_predicate_arima(x, y, p, d, q, nPred, isSmooth)
```

参数说明如下：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
p	自回归模型的阶数。	long类型，取值范围为2~8。
d	差分模型的阶数。	long类型，取值范围为1~3。
q	移动平均模型的阶数。	long类型，取值范围为2~8。
nPred	预测未来的点的数量。	long类型，取值范围为1~5*p。
isSmooth	是否需要对原始数据做滤波操作。	bool类型，默认为true表示对原始数据做滤波操作。

查询分析示例：

```
* | select ts_predicate_arima(stamp, value, 3, 1, 2, 4) from (select __time__ - __time__ % 60 as stamp, avg(v) as value from log GROUP BY stamp order by stamp)
```

> **说明** 输出结果与ts_predicate_simple函数相似，具体请参见ts_predicate_simple函数的输出结果。

ts_regression_predict

函数格式：

```
select ts_regression_predict(x, y, nPred, algotype,processType)
```

参数说明如下：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
nPred	预测未来的点的数量。	long类型，取值范围为1~500。
algotype	针对的预测的算法类型。	取值包括： • origin：使用GBRT（Gradient Boosted Regression Tree）算法进行预测。 • forest：使用STL序列分解的结果，将分解得到的趋势序列使用GBRT算法进行预测，再将分解出来的序列按照加法模型进行求和后返回。 • linear：使用STL序列分解的结果，将分解得到趋势序列使用Linear Regression算法进行预测，再将分解出来的序列按照加法模型进行求和后返回。

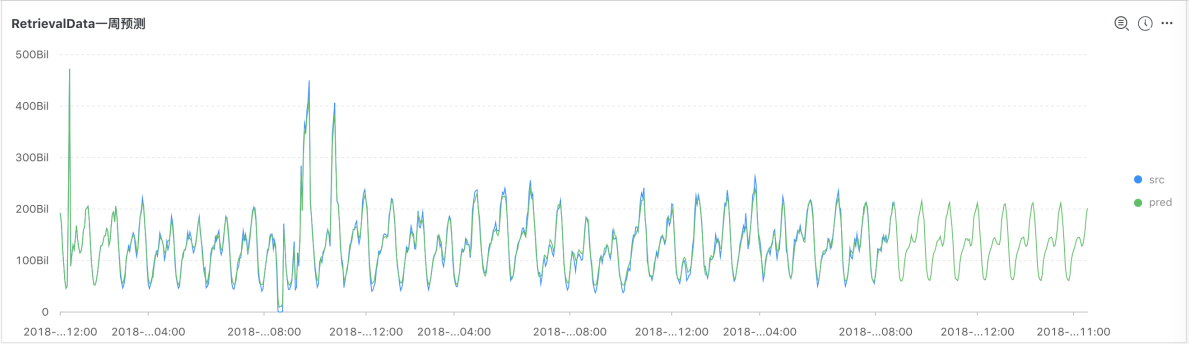
参数	说明	取值
processType	数据对应的预处理流程。	取值包括： • 0：不进行任何额外的数据预处理。 • 1：对数据去除异常后再进行预测处理。

示例：

- 查询分析

```
* and h : nu2h05202.nu8 and m: NET |  select ts_regression_predict(stamp, value, 200, 'origin') fr
om (select __time__ - __time__ % 60 as stamp, avg(v) as value from log GROUP  BY  stamp order by s
tamp)
```

- 输出结果



显示项如下：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
纵轴	src	原始数据。
	predict	预测数据。

ts_anomaly_filter

函数格式：

```
select ts_anomaly_filter(lineName, ts, ds, preds, probs, nWatch, anomalyType)
```

参数说明如下：

参数	说明	取值
lineName	varchar类型，表示每条曲线的名称。	-
ts	曲线的时间序列，表示当前这条曲线的时间信息。array（double）类型，由小到大排列。	-
ds	曲线的实际值序列，表示当前这条曲线的数值信息。array（double）类型，长度与ts相同。	-

参数	说明	取值
preds	曲线的预测值序列，表示当前这条曲线的预测值。array（double）类型，长度与ts相同。	-
probs	曲线的异常检测序列，表示当前这条曲线的异常检测结果。array（double）类型，长度与ts相同。	-
nWatch	long类型，表示当前曲线中最近观测的实际值的数量，长度必须小于实际的曲线长度。	-
anomalyType	long类型，表示要过滤的异常类型的种类。	取值包括： • 0：表示关注全部异常。 • 1：表示关注上升沿异常。 • -1：表示下降沿异常。

示例：

- 查询分析

```
* | select res.name, res.ts, res.ds, res.preds, res.probs
     from (
         select ts_anomaly_filter(name, ts, ds, preds, probs, cast(5 as bigint), cast(1 as bigint)
) as res
       from (
         select name, res[1] as ts, res[2] as ds, res[3] as preds, res[4] as uppers, res[5] as low
ers, res[6] as probs
     from (
         select name, array_transpose(ts_predicate_ar(stamp, value, 10)) as res
         from (
           select name, stamp, value from log where name like '%asg-%') group by name)) );
```

- 输出结果

```
| name                     | ts                                                          | ds          |
preds     | probs       |
| ------------------------ | ----------------------------------------------------------- | ----------- |
--------- | ----------- |
| asg-bp1hylzdi2wx7civ0ivk | [1.5513696E9, 1.5513732E9, 1.5513768E9, 1.5513804E9] | [1,2,3,NaN] |
[1,2,3,4] | [0,0,1,NaN] |
```

12.7. 序列分解函数

序列分解函数提供针对业务曲线的分解功能，突出曲线的趋势信息和周期信息。

ts_decompose

函数格式：

```
select ts_decompose(x, y)
```

参数说明如下：

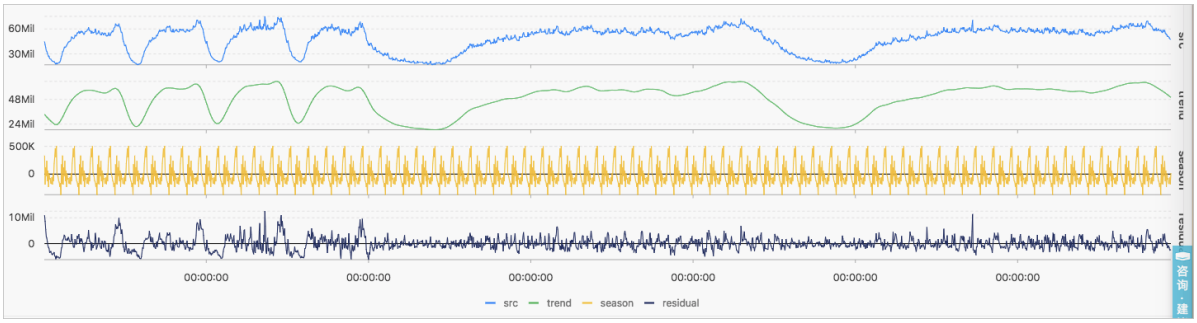
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-

示例：

- 查询分析：

```
* | select ts_decompose(stamp, value) from (select __time__ - __time__ % 60 as stamp, avg(v) as va
lue from log GROUP BY stamp order by stamp)
```

- 输出结果：



显示项如下：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
纵轴	src	原始数据。
	trend	分解出来的趋势数据。
	season	分解出来的周期数据。
	residual	分解出来的残差数据。

12.8. 时序聚类函数

时序聚类函数针对输入的多条时序数据进行聚类，自动聚类出不同的曲线形态，进而快速找到相应的聚类中心和异于聚类中的其它形态曲线。

关于函数的算法及实现原理请参见LOG机器学习介绍（02）：时序聚类建模。

函数列表

函数	说明
ts_density_cluster	使用密度聚类方法对多条时序数据进行聚类。
ts_hierarchical_cluster	使用层次聚类方法对多条时序数据进行聚类。
ts_similar_instance	查找到指定曲线名称的相似曲线。

ts_density_cluster

函数格式如下所示：

```
select ts_density_cluster(x, y, z)
```

参数说明如下所示：

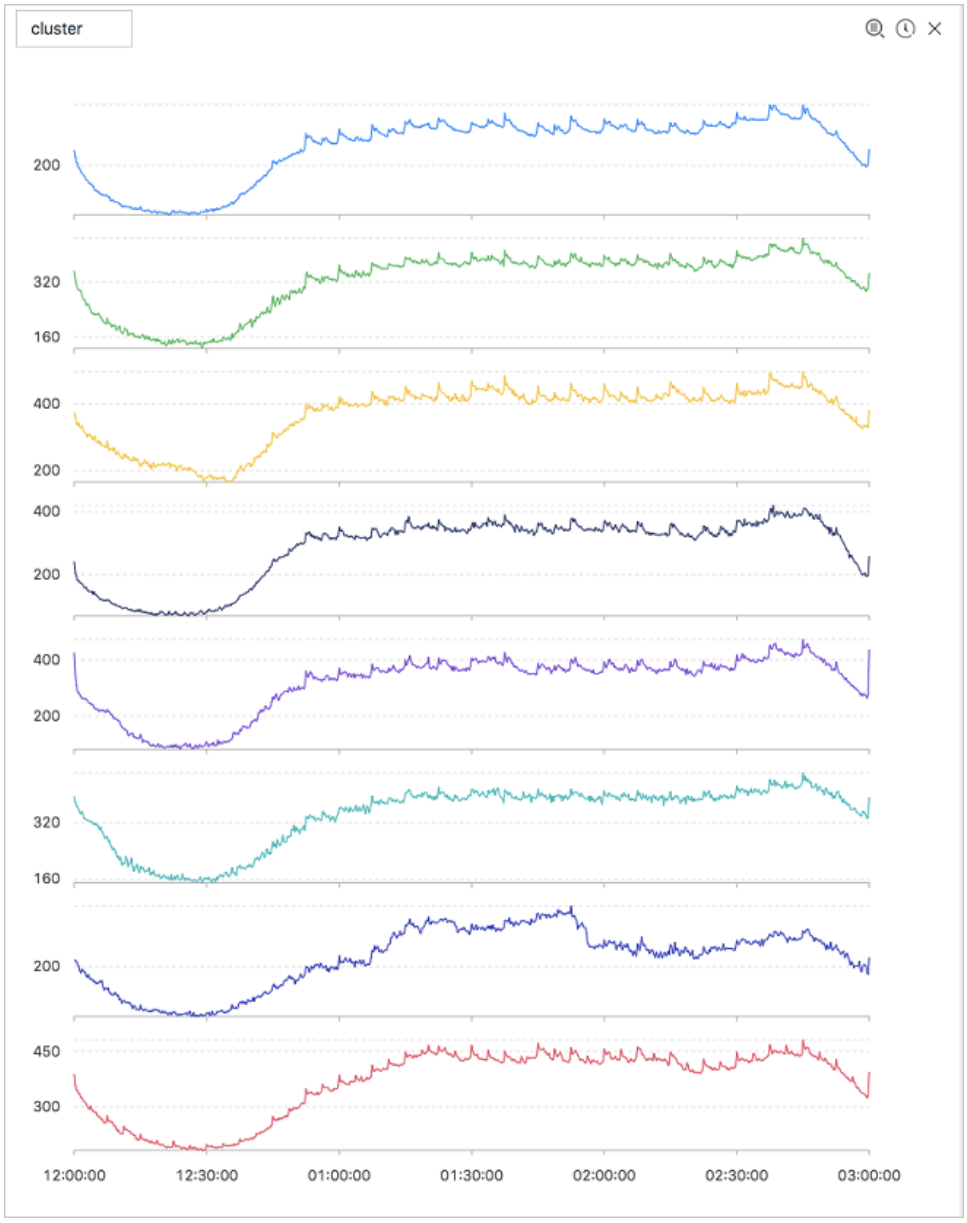
参数	说明	取值
x	时间列，从小到大排列。	Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	无
z	某个时刻数据对应的曲线名称。	字符串类型，例如machine01.cpu_usr。

示例：

- 查询分析语句如下所示：

```
* and (h: "machine_01" OR h: "machine_02" OR h : "machine_03") | select ts_density_cluster(stamp, metric_value,metric_name ) from ( select __time__ - __time__ % 600 as stamp, avg(v) as metric_value, h as metric_name from log GROUP BY stamp, metric_name order BY metric_name, stamp )
```

- 输出结果如下所示：



显示项如下所示：

显示项	说明
cluster_id	聚类的类别，其中-1表示未能划分到某一聚类中心。
rate	该聚类中的instance占比。
time_series	该聚类中心的时间戳序列。
data_series	该聚类中心的数据序列。
instance_names	该聚类中心包含的instance的集合。
sim_instance	该类中的某一个instance名称。

ts_hierarchical_cluster

函数格式如下所示：

```
select ts_hierarchical_cluster(x, y, z)
```

参数说明如下所示：

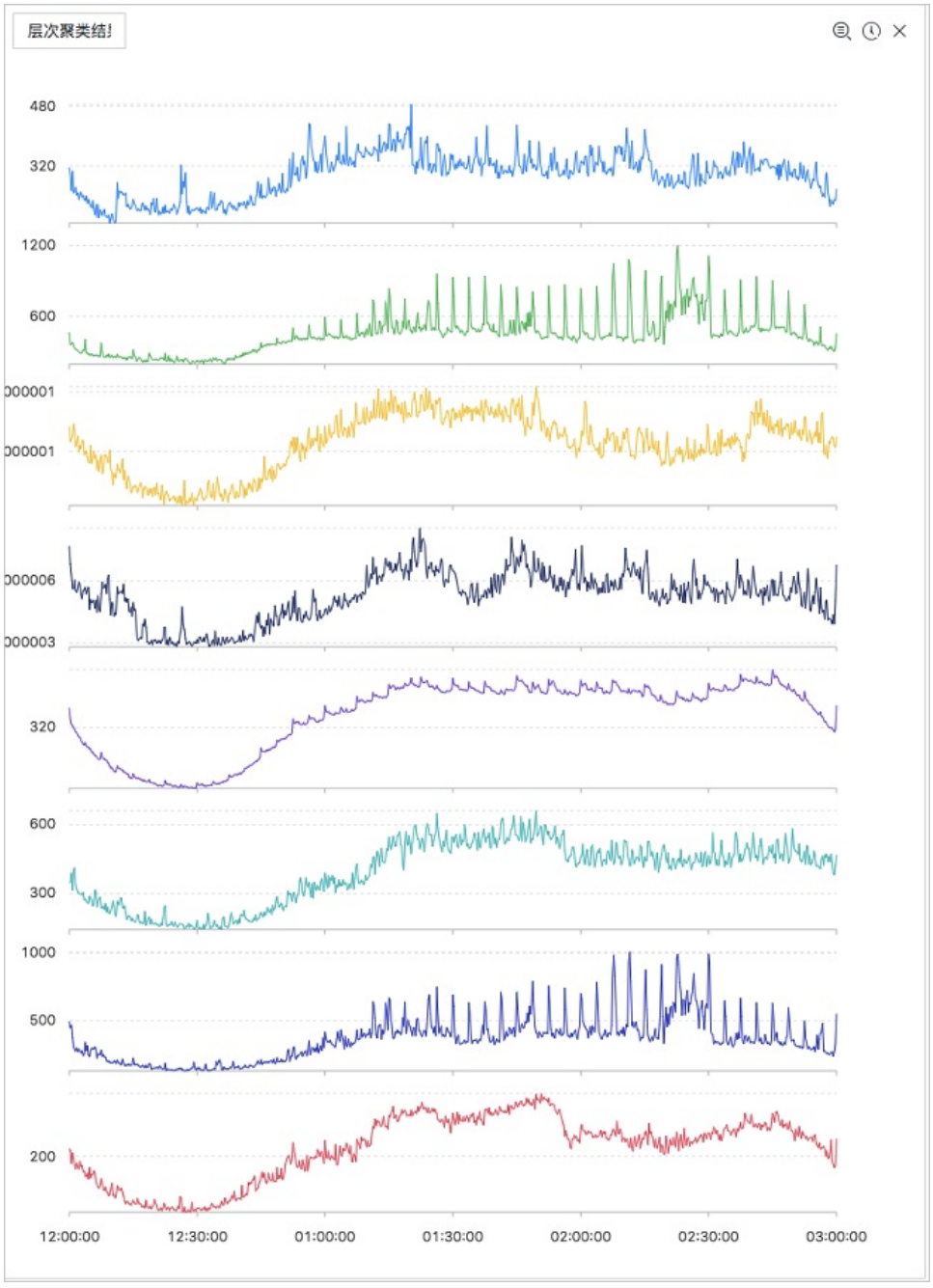
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	无
z	某个时刻数据对应的曲线名称。	字符串类型，例如machine01.cpu_usr。

示例：

- 查询分析语句如下所示：

```
* and (h: "machine_01" OR h: "machine_02" OR h : "machine_03") | select ts_hierarchical_cluster(st
amp, metric_value, metric_name) from ( select __time__ - __time__ % 600 as stamp, avg(v) as metric
_value, h as metric_name from log GROUP BY stamp, metric_name order BY metric_name, stamp )
```

- 输出结果如下所示：



显示项如下所示：

显示项	说明
cluster_id	聚类的类别，其中-1表示未能划分到某一聚类中心。
rate	该聚类中的instance占比。
time_series	该聚类中心的时间戳序列。
data_series	该聚类中心的数据序列。
instance_names	该聚类中心包含的instance的集合。
sim_instance	该类中的某一个instance名称。

ts_similar_instance

函数格式如下所示：

```
select ts_similar_instance(x, y, z, instance_name, topK, metricType)
```

参数说明如下所示：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	无
z	某个时刻数据对应的曲线名称。	字符串类型，例如machine01.cpu_usr。
instance_name	指定某个待查找的曲线名称。	集合中某个曲线名称，字符串类型，例如machine01.cpu_usr。 **说明** 必须是已创建的曲线。
topK	最多返回K个与给定曲线相似的曲线。	无
metricType	`{'shape', 'manhattan', 'euclidean'}`，衡量时序曲线之间的相似性指标。	无

查询分析语句如下所示：

```
* and m: NET and m: Tcp and (h: "nu4e01524.nu8" OR  h: "nu2i10267.nu8" OR  h : "nu4q10466.nu8") | sel
ect ts_similar_instance(stamp, metric_value, metric_name, 'nu4e01524.nu8' ) from ( select __time__ -
__time__ % 600 as stamp, sum(v) as metric_value, h as metric_name from log GROUP BY stamp, metric_nam
e order BY  metric_name, stamp )
```

显示项如下所示：

显示项	说明
instance_name	与指定指标相近的结果列表。
time_series	该曲线的时间戳序列。
data_series	该曲线的数据序列。

12.9. 频繁模式统计函数

频繁模式统计函数可以在给定的多属性字段样本中，挖掘出具有一定代表性的属性组合，用来归纳当前日志。

pattern_stat

函数格式：

```
select pattern_stat(array[col1, col2, col3], array['col1_name', 'col2_name', 'col3_name'], array[col5
, col6], array['col5_name', 'col6_name'], support_score, sample_ratio)
```

参数说明如下：

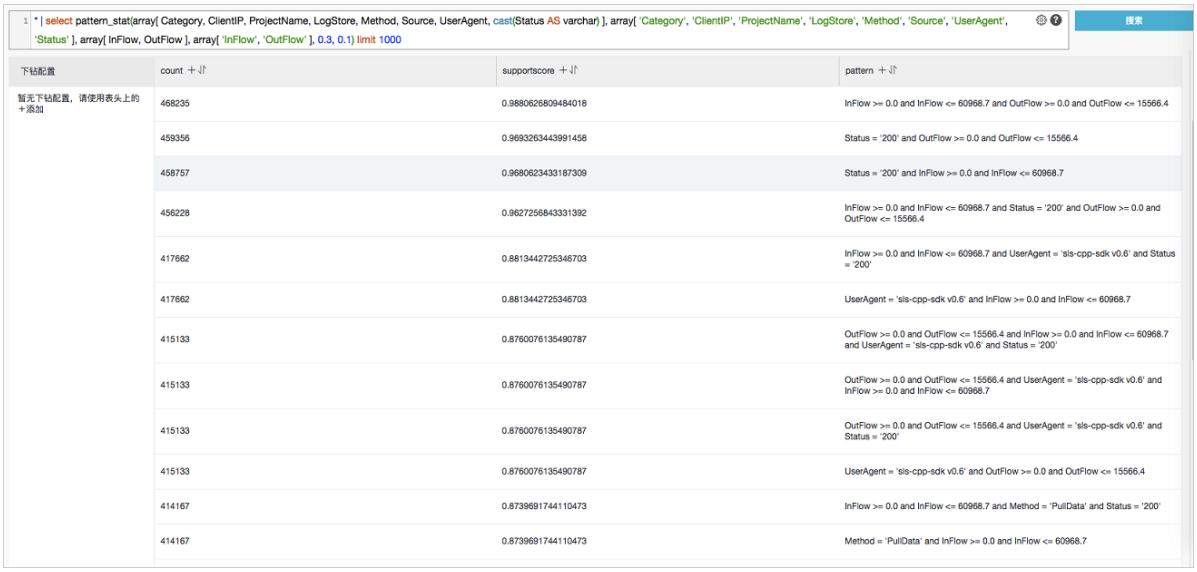
参数	说明	取值
array[col1, col2, col3]	字符型数据的输入列。	数组形式，例如：array[clientIP, sourceIP, path, logstore]。
array['col1_name', 'col2_name', 'col3_name']	字符型数据的输入列的对应名称。	数组形式，例如：array['clientIP', 'sourceIP', 'path', 'logstore']。
array[col5, col6]	数值型数据的输入列。	数组形式，例如：array[Inflow, OutFlow]。
array['col5_name', 'col6_name']	数值型数据的输入列的对应名称。	数组形式，例如array['Inflow', 'OutFlow']。
support_score	样本在进行模式挖掘时的支持度。	double类型，取值为(0,1]。
sample_ratio	采样比率，默认为0.1，表示只拿10%全量集合。	double类型，取值为(0,1]。

示例：

- 查询分析：

```
* | select pattern_stat(array[ Category, ClientIP, ProjectName, LogStore, Method, Source, UserAgent ], array[ 'Category', 'ClientIP', 'ProjectName', 'LogStore', 'Method', 'Source', 'UserAgent' ], array[ InFlow, OutFlow ], array[ 'InFlow', 'OutFlow' ], 0.45, 0.3) limit 1000
```

- 输出结果：



显示项如下：

显示项	说明
count	当前模式所含样本的数量。
support_score	当前模式的支持度。
pattern	模式的具体内容，按照条件查询的形式组织。

12.10. 差异模式统计函数

差异模式统计函数基于给定的多属性字段样本，在给定的判别条件下，分析出影响该条件划分的差异化模式集合，帮助您快速诊断导致当前判别条件差异的原因。

pattern_diff

函数格式：

```
select pattern_diff(array_char_value, array_char_name, array_numeric_value, array_numeric_name, condition, supportScore,posSampleRatio,negSampleRatio )
```

参数说明如下：

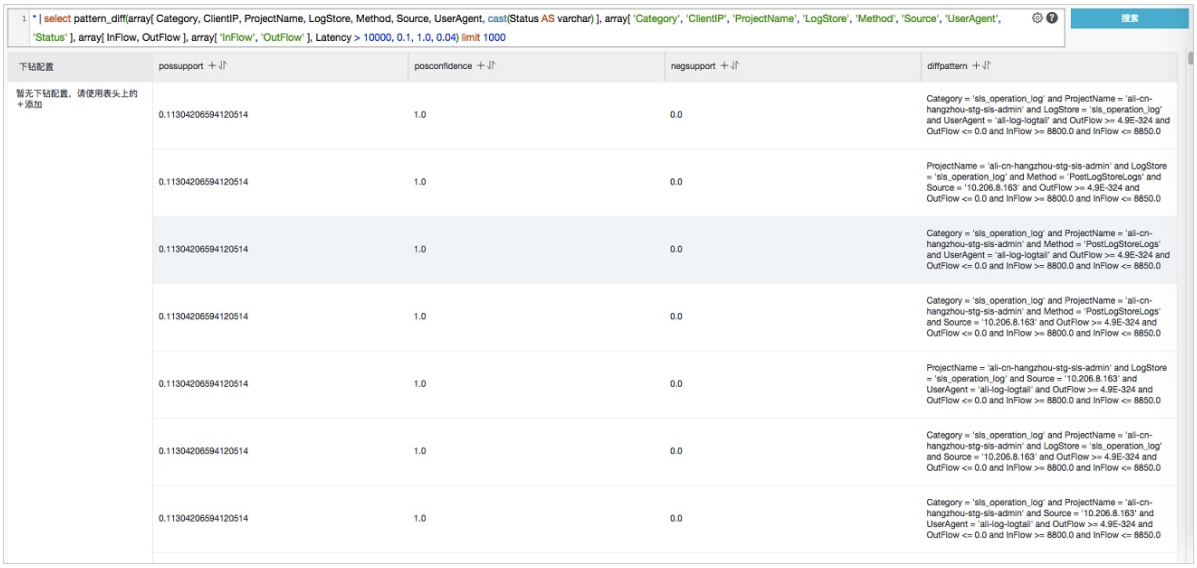
参数	说明	取值
array_char_value	字符型数据的输入列。	数组形式，例如：array[clientIP, sourceIP, path, logstore]。
array_char_name	字符型数据的输入列的对应名称。	数组形式，例如：array['clientIP', 'sourceIP', 'path', 'logstore']。
array_numeric_value	数值型数据的输入列。	数组形式，例如：array[Inflow, OutFlow]。
array_numeric_name	数值型数据的输入列的对应名称。	数组形式，例如array['Inflow', 'OutFlow']。
condition	筛选数据的条件。条件为True则为正样本，条件为False则为负样本。	例如：Latency <= 300。
supportScore	正负样本在进行模式挖掘时的支持度。	double类型，取值为(0,1]。
posSampleRatio	正样本的采样率。默认为0.5，表示只取50%正样本集合。	double类型，取值为(0,1]。
negSampleRatio	负样本的采样率，默认为0.5，表示只取50%负样本集合。	double类型，取值为(0,1]。

示例：

- 查询分析：

```
* | select pattern_diff(array[ Category, ClientIP, ProjectName, LogStore, Method, Source, UserAgent ], array[ 'Category', 'ClientIP', 'ProjectName', 'LogStore', 'Method', 'Source', 'UserAgent' ], array[ InFlow, OutFlow ], array[ 'InFlow', 'OutFlow' ], Latency > 300, 0.2, 0.1, 1.0) limit 1000
```

- 输出结果：



显示项如下：

显示项	说明
possupport	挖掘出来的模式在正样本中的支持度。
posconfidence	挖掘出来的模式在正样本中的置信度。
negsupport	挖掘出来的模式在负样本中的支持度。
diffpattern	挖掘出来的具体模式内容。

12.11. URL请求分类函数

URL请求分类函数会自动将您输入的URL请求路径进行归类打标签，并提供类别的正则表达式，帮助您更好的归类URL，查询结果可供ETL使用。

> **说明** 目前，URL请求分类函数只支持华北2（北京）、华东2（上海）地域。

- 调用方式

```
select url_classify(url_path varchar);
select url_classify(url_path varchar, weight long);
```

- 输入参数

参数	说明
url_path	URL请求路径。
weight	URL请求路径的数量。

- 输出参数

参数	说明
url_path	URL请求路径。
api_path	通过函数推导出URL请求路径对应的接口。

参数	说明
regex_tpl	通过算法推导出的正则表达式。

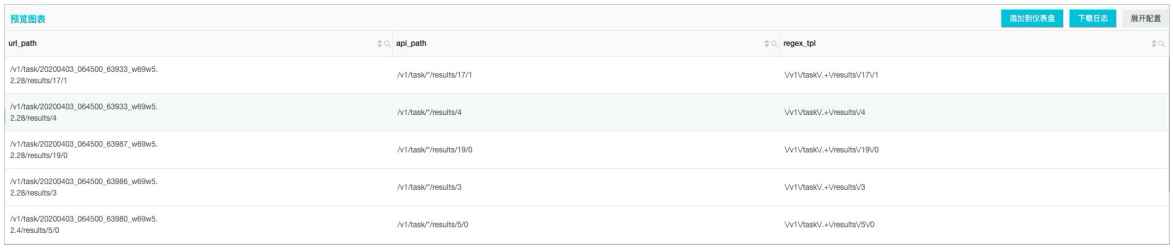
- 输出结果

```
     url_path                          |     api_path                          |          regex_
tpl
-----------------------------------+-----------------------------+-----------------------------
--------
 /gl/balance/666398186799140          | /gl/balance/*                          | \/gl\/balance
\/[0-9].+
 /gl/glaccount/30579281472076         | /gl/glaccount/*                       | \/gl\/glaccount
\/[0-9].+
 /gl/balance/709016207098025          | /gl/balance/*                          | \/gl\/balance\
/[0-9].+
```

- 示例
  - 查询分析语句

```
* | select url_classify(uri, num) from (select uri, COUNT(*) as num from log group by uri limit
1000)
```

  - 查询分析结果



12.12. 根因分析函数

日志服务提供了强大的告警和分析能力，可以帮助用户快速分析和定位到发生异常的具体的子维度。在时序指标发生异常时，根因分析函数可以快速分析出是哪些相关维度属性发生异常而导致监控指标发生异常。

LOG机器学习最佳实战：根因分析（一）

rca_kpi_search

函数格式

```
select rca_kpi_search(varchar_array, name_array, real, forecast, level)
```

参数说明如下：

参数	说明	取值
varchar_array	属性维度字段。	数组形式，例如：array[col1, col2, col3]。
name_array	属性名字字段。	数组形式，例如：array['col1', 'col2', 'col3']。
real	varchar_array对应的实际值。	double 类型，取值范围：全体实数。

参数	说明	取值
forecast	varchar_array对应的预测值。	double 类型，取值范围：全体实数。
level	输出的根因集合对应的维度属性的数量，其中level=0表示输出找到的全部根因集合。	long类型，取值范围：0<=level<=分析维度数（对应varchar_array的长度）。

示例：

- 查询分析：

  先利用子查询去组织每个细粒度属性对应的实际值和预测值，然后直接调用rca_kpi_search函数去分析异常时刻的根因。

```
* not Status:200 |
select rca_kpi_search(
 array[ ProjectName, LogStore, UserAgent, Method ],
 array[ 'ProjectName', 'LogStore', 'UserAgent', 'Method' ], real, forecast, 1)
from (
select ProjectName, LogStore, UserAgent, Method,
 sum(case when time < 1552436040 then real else 0 end) * 1.0 / sum(case when time < 1552436040
then 1 else 0 end) as forecast,
 sum(case when time >=1552436040 then real else 0 end) *1.0 / sum(case when time >= 1552436040
then 1 else 0 end) as real
 from (
select __time__ - __time__ % 60 as time, ProjectName, LogStore, UserAgent, Method, COUNT(*) as rea
l
from log GROUP by time, ProjectName, LogStore, UserAgent, Method )
GROUP BY ProjectName, LogStore, UserAgent, Method limit 100000000)
```

- 输出结果：

返回结果结构说明：

```
{
  "rcSets": [
    {
      "rcItems": [
        {
          "kpi": [("attr": "xxx", "val": "xxx")],
          "nleaf": 100,
          "change": 0.524543,
          "score": 0.1454543
        }
      ]
    }
  ]
}
```

显示项如下：

显示项	说明
rcSets	根因集合，value对应一个数组。
rcItems	具体对应一个根因集合。
kpi	根因集合中的一项，数据按照数组形式存储，数组中的每一项是一个JSON类型的数据，attr表示维度名称，val表示当前维度下对应的属性名称。
nleaf	根因集合中某一项（KPI）在原始数据中覆盖的叶子节点数。 **说明** 叶子节点：表示最细粒度属性组合的日志。
change	根因集合中某一项（KPI）对应的叶子节点集合的异常变化量占同一时刻总体异常变化量的比例。
score	当前kpi对应的异常程度（0 <= score <= 1）。

输出结果是一个JSON，具体格式如下：

```
{
  "rcSets": [
  {
    "rcItems": [
    {
      "kpi": [
      {
        "attr": "country",
        "val": "*"
      },
      {
        "attr": "province",
        "val": "*"
      },
      {
        "attr": "provider",
        "val": "*"
      },
      {
        "attr": "domain",
        "val": "example.com"
      },
      {
        "attr": "method",
        "val": "*"
      }
      ],
      "nleaf": 119,
      "change": 0.3180687806279939,
      "score": 0.14436007709620113
    }
    ]
  }
  ]
}
```


12.13. 相关性分析函数

针对系统中的多个观测指标，可以快速找出与某个指标项相关或者时序序列相关的指标名称。

函数列表

函数	说明
ts_association_analysis	针对系统中的多个观测指标，快速找出和某个指标项相关的指标名称。
ts_similar	针对系统中的多个观测指标，快速找出和用户输入的时序序列相关的指标名称。

ts_association_analysis

函数格式：

```
select ts_association_analysis(stamp, params, names, indexName, threshold)
```

参数说明如下：

参数	说明	取值
stamp	long 类型，表示UnixTime时间戳。	-
params	array（double）类型，表示待分析的指标维度。	例如：Latency，QPS，NetFlow等。
names	array（varchar ）类型，表示待分析的指标名称。	例如：Latency，QPS，NetFlow等。
indexName	varchar 类型，表示分析目标指标的名称。	例如：Latency。
threshold	double 类型，表示其它分析指标与目标指标间的相关性阈值。	取值范围在：[0，1]。

结果输出：

- name：指标的名称。
- score：该指标与目标指标之间的相关性值，范围在[0，1]之间。

代码示例

```
* | select ts_association_analysis(
              time,
              array[inflow, outflow, latency, status],
              array['inflow', 'outflow', 'latency', 'status'],
              'latency',
              0.1) from log;
```

结果示例：

```
| results               |
| --------------------- |
| ['latency', '1.0']    |
| ['outflow', '0.6265'] |
| ['status', '0.2270']  |
```

ts_similar

函数格式一：

```
select ts_similar(stamp, value, ts, ds)
select ts_similar(stamp, value, ts, ds, metricType)
```

参数说明一：

参数	说明	取值
stamp	long 类型，表示UnixTime时间戳。	-
value	double 类型，表示某指标对应的值。	-
ts	array（double）类型，表示指定曲线的时间序列信息。	-
ds	array（double）类型，表示指定曲线的数值序列信息。	-

参数	说明	取值
metricType	varchar 类型，表示度量曲线间相关性的类型。	类型如下： SHAPE，RMSE，PEARSON，SPEARMAN，R2，KENDALL

函数格式二：

```
select ts_similar(stamp, value, startStamp, endStamp, step, ds)
select ts_similar(stamp, value, startStamp, endStamp, step, ds, metricType )
```

参数说明二：

参数	说明	取值
stamp	long 类型，表示UnixTime时间戳。	-
value	double 类型，表示某指标对应的值。	-
startStamp	long 类型，表示指定曲线的开始时间戳。	-
endStamp	long 类型，表示指定曲线的结束时间戳。	-
step	long类型，表示时序中相邻两个点之间的时间间隔。	-
ds	array（double）类型，表示指定曲线的数值序列信息。	-
metricType	varchar 类型，表示度量曲线间相关性的类型。	类型如下： SHAPE，RMSE，PEARSON，SPEARMAN，R2，KENDALL

输出结果：

- score：该指标与目标指标之间的相关性值，范围在[-1, 1]之间。

代码示例：

```
* | select vhost, metric, ts_similar(time, value, 1560911040, 1560911065, 5, array[5.1,4.0,3.3,5.6,4.0,7.2], 'PEARSON') from log  group by vhost, metric;
```

结果示例：

```
| vhost   | metric          | score                |
| ------- | --------------- | -------------------- |
| vhost1 | redolog         | -0.3519082537204182  |
| vhost1 | kv_qps          | -0.15922168009772697 |
| vhost1 | file_meta_write | NaN                  |
```

12.14. 核密度估计函数

核密度估计（Kernel Density Estimation）在概率论中用来估计未知的密度函数，属于非参数检验方法之一。

核密度估计函数采用平滑的峰值函数来拟合观察到的数据点，从而对真实的概率分布曲线进行模拟。

- 函数格式

```
select kernel_density_estimation(bigint stamp, double value, varchar kernelType)
```

- 参数说明

参数	说明
stamp	UnixTime 时间戳数据，单位为秒。
value	对应的观测数值。
kernelType	○ box：矩形。 ○ epanechnikov：Epanechnikov 曲线。 ○ gaussian：高斯曲线。

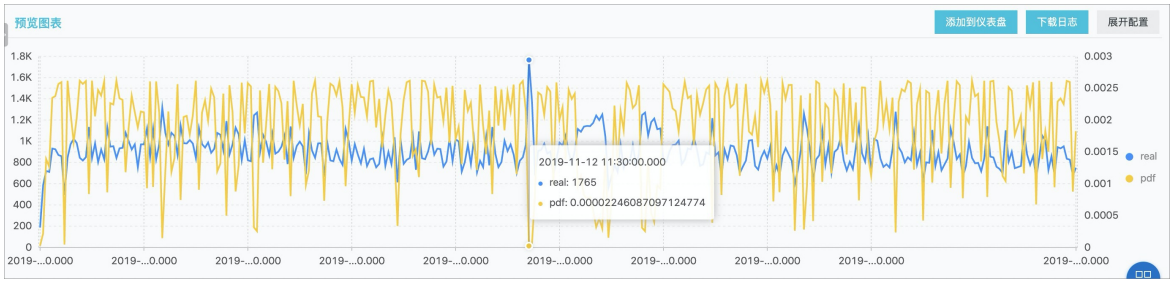
- 输出结果

显示项	说明
unixtime	对应原始数据的时间数据。
real	对应的观测数值。
pdf	对应的每个观测点的概率值。

- 示例
  - 代码示例：

```
* |
select
    date_trunc('second', cast(t1[1] as bigint)) as time, t1[2] as real, t1[3] as pdf from (
        select kernel_density_estimation(time, num, 'gaussian') as res from (
            select __time__ - __time__ % 10 as time, COUNT(*) * 1.0 as num from log group by tim
e order by time)
        ), unnest(res) as t(t1)  limit 1000
```

  - 结果示例：



12.15. 时序补点函数

如果时间序列中存在数据缺失问题，可以使用时序补点函数补齐缺失的数据。

- 调用方式

```
select series_padding(long stamp, double value, long interval, varchar padType)
```

- 输入参数

参数	说明
stamp	数据的UnixTime时间戳。
value	每个时刻对应的数据。
interval	采集数据的间隔，例如：每10秒进行一次采集，则interval为10。
padType	数据缺失时填充的类型，可选值：zero、mean、forward、backward。 ○ zero：缺失点数据填充为0。 ○ mean：缺失点数据填充为缺失点两端有效值的均值。 ○ forward：缺失点数据填充为缺失点左端有效数据。 ○ backward：缺失点数据填充为缺失点右端有效数据。

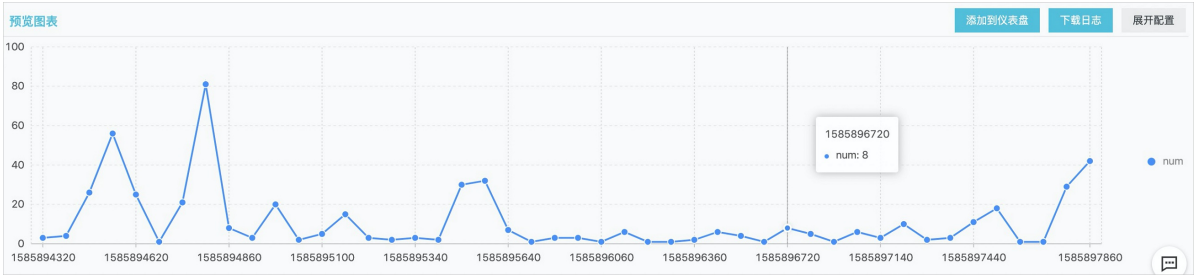
- 输出结果

```
unixtime    |       pad_value
------------+-----------------------
 1.5513696E9 |   0.11243584740434608
 1.5513732E9 |   0.09883780706698506
 1.5513768E9 |   0.08240823914341992
 1.5513804E9 |    0.0728240514818139
  1.551384E9 |   0.05888517541914705
 1.5513876E9 |   0.04953931499029833
 1.5513912E9 |  0.043698605551761895
 1.5513948E9 |   0.04400292632222124
 1.5513984E9 |   0.04727081764249449
  1.551402E9 |  0.054632234293121314
 1.5514056E9 |   0.05331214064978596
 1.5514092E9 |   0.05093117289934144
 1.5514128E9 |  0.053620170319174806
 1.5514164E9 |   0.05405914786225842
```

- 示例

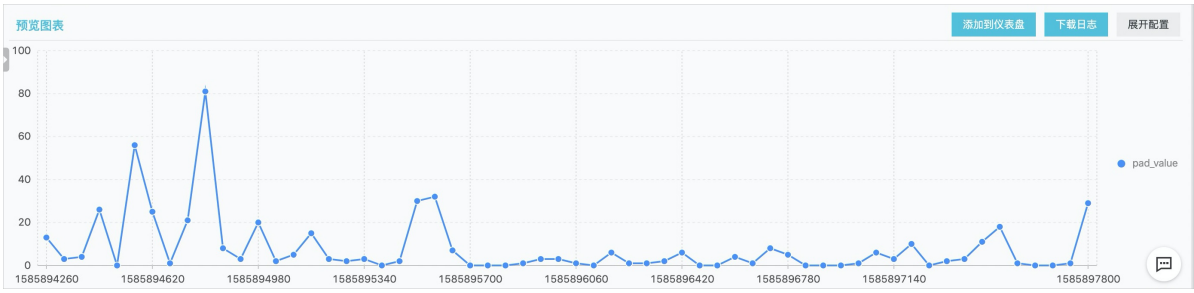
下图为通过查询语句得到的原始折线图，存在数据缺失问题。

```
* and Method: GetLogStoreLogs and ProjectName: lunar and LogStore: geos and Latency > 800000 | sel
ect __time__ - __time__ 60% as time, COUNT(*) * 1.0 as num from log group by time order by time as
c limit 1000
```



执行时序补点函数进行数据补齐，结果如下图所示。

```
* and Method: GetLogStoreLogs and ProjectName: lunar and LogStore: geos and Latency > 800000 | sel
ect series_padding(time, num, 60, 'zero') from (select __time__ - __time__ 60% as time, COUNT(*) *
1.0 as num from log group by time order by time asc limit 1000)
```



12.16. 异常对比函数

异常对比函数用于比较某个观测对象在两个时间段的差异程度。

- 调用方式一
  - 调用函数

```
select anomaly_compare(long stamp, array[ feature_1, feature_2 ], long timePoint, long interval)
select anomaly_compare(long stamp, array[ feature_1, feature_2 ], array[ feature1_name, feature2_name ], long timePoint, long interval)
```

  - 输入参数

参数	说明
stamp	数据的UnixTime时间戳。
array[features]	某个时刻，观测对象的特征数据。
array[featureNames]	特征数据的描述信息。
timePoint	观测对象发生变更时对应的时间点的UnixTime时间戳。
interval	采集数据的间隔，例如：每10秒进行一次采集，则interval为10。

- 调用方式二
  - 调用函数

```
select anomaly_compare(long stamp, array[ feature_1, feature_2 ], array[ feature1_name, feature2_name ], long version)
```

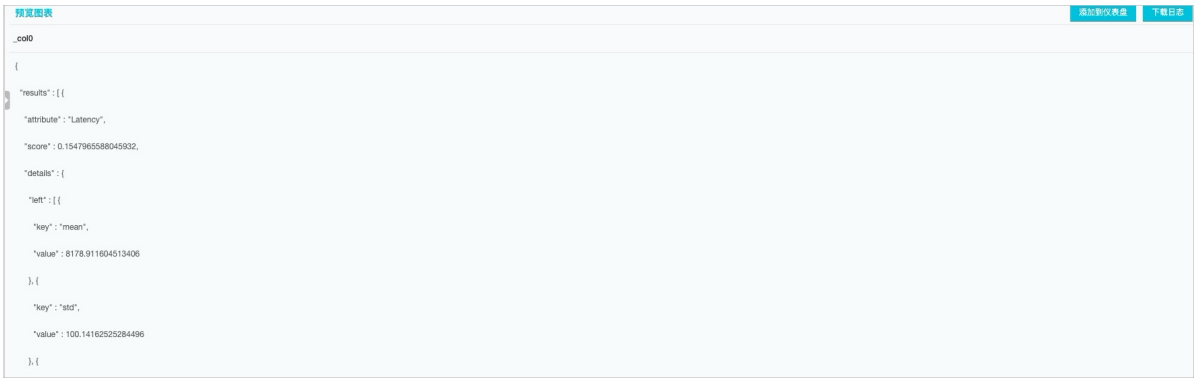
  - 输入参数

参数	说明
stamp	数据的UnixTime时间戳。
array[features]	某个时刻，观测对象的特征数据。
array[featureNames]	上述特征数据的文字描述信息。
version	时间序列的版本号。 ■ version=0表示原始数据的版本号。 ■ version=1表示新数据的版本号。

- 输出结果

```
{
   "results" : [ {
     "attr" : "cpu",
     "anomalyScore" : 0.01106371634297909,
     "details" : {
       "left" : [ {
         "key" : "mean",
         "value" : 0.07002069952622482
       }, {
         "key" : "std",
         "value" : 0.1364542814430179
       }, {
         "key" : "median",
         "value" : 0.04467685956328345
       }, {
         "key" : "variance",
         "value" : 0.018619770924130346
       } ],
       "rightMetrics" : [ {
         "key" : "mean",
         "value" : 0.4472823405432968
       }, {
         "key" : "std",
         "value" : 0.22405908739288383
       }, {
         "key" : "median",
         "value" : 0.42513225830553775
       }, {
         "key" : "variance",
         "value" : 0.05020247464333195
       } ]
     }
   } ]
 }
```


- 输出说明
  - 针对单条时序提供的统计信息包括mean、std、median、variance。
  - 如果您指定特征的名字，则按照指定的名字填充attr字段；否则将column_和特征在数组中的下标进行拼接作为attr，例如：column_0。
  - anomalyScore：由函数计算出来的异常分数，取值范围：[0, 1]。数据趋于0，表示差异不大；数据趋于1表示差异较大。
- 示例



13.Scheduled SQL

13.1. 工作原理

日志服务提供Scheduled SQL功能，用于定时分析数据、存储聚合数据、投影与过滤数据。本文介绍Scheduled SQL功能的背景信息、功能简介、基本概念、调度与执行场景、使用建议等信息。

背景信息

基于时间的数据（日志、指标）在日积月累后的数量是惊人的。例如每天产生1000万条数据，则一年为36亿条数据。一方面，长时间的数据存储需要巨大的存储空间，而通过减少存储周期的方式减少存储空间，虽然降低了存储成本，但也丢失了有价值的数据。另一方面，大量的数据将造成分析上的性能压力。

数据的存储和分析具备以下特征：

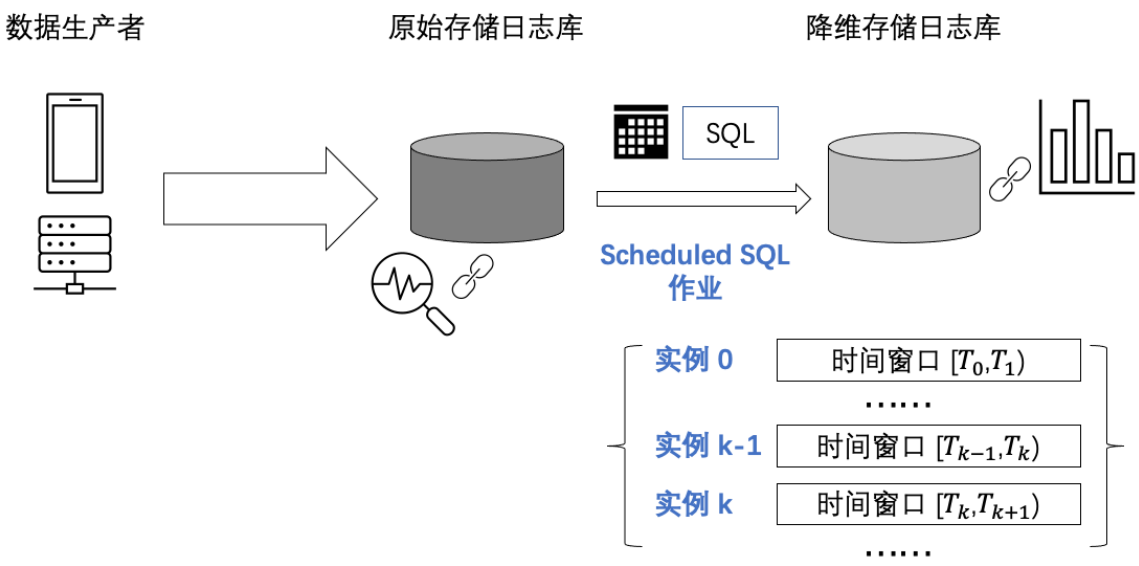
- 大部分时序数据具有时效性特征。历史数据可以为分钟或小时级别的精度，而新产生的数据需要更高的精度。
- 数据使用者（例如数据运营、数据科学家）需要存储全量的数据以备分析。
- 在数据分析阶段，需同时兼顾全量数据和快速查询响应。

针对上述特征，日志服务推出Scheduled SQL功能，用于将高精度的历史数据压缩为低精度数据后，长期存储。使用Scheduled SQL功能后，您可以根据业务需求为源库设置较低的存储周期（例如15天），将目标库的存储周期设置为永久保存。实现长时间范围数据的低延迟分析、低成本存储。

功能简介

Scheduled SQL支持标准SQL92语法、日志服务查询和分析语法，按照调度规则周期性执行，并将运行结果写入到目标库（Logstore或Metricstore）中。

- 定时分析数据：根据业务需求设置SQL语句或查询分析语句，定时执行数据分析，并将分析结果存储到目标库中。
- 全局聚合：对全量、细粒度的数据进行聚合存储，汇总为存储大小、精度适合的数据，相当于一定程度的有损压缩数据。例如：
  - 按照秒级别对36亿条数据进行聚合存储，存储结果为3150万条数据，存储大小为全量数据的0.875%。
  - 按照分钟级别对36亿条数据进行聚合存储，存储结果为52.5万条数据，存储大小为全量数据的0.015%。



- 投影与过滤：对原始数据的字段进行筛选，按照一定条件过滤数据并存储到目标Logstore中。

  该功能还可以通过数据加工实现，数据加工的DSL语法比SQL语法具备更强的ETL表达能力。更多信息，请参见加工原理。

基本概念

- 作业：一个Scheduled SQL任务对应一个作业，包括计算配置、调度配置等信息。
- 实例：一个Scheduled SQL作业按照调度配置按时生成执行实例。每一个实例对原始数据进行SQL计算并将计算结果写入目标库。
- 实例ID：执行实例的唯一标识。
- 创建时间：实例的创建时间。一般是按照您配置的调度规则生成，在补运行或追赶延迟时会立即生成实例。
- 执行时间：实例开始执行的时间。如果重试作业，则表示最后一次开始执行的时间。
- 结束时间：实例执行结束的时间。如果重试作业，则表示最后一次执行结束的时间。
- 调度时间：由调度规则生成，不会受到上一个实例执行超时、延迟、补运行等情况的影响。

  大部分场景下，连续生成的实例的调度时间是连续的，可处理完整的数据集。
- SQL时间窗口：Scheduled SQL作业运行时，日志服务仅分析该时间范围内的数据。SQL时间窗口基于调度时间计算而得，左闭右开格式，且与实例的创建时间、执行时间无关。例如调度时间为2021/01/01 10:00:00，SQL时间窗口的表达式为[@m - 10m, @m)，则实际的SQL时间窗口为[2021/01/01 09:50:00, 2021/01/01 10:00:00)。
- 执行状态：Scheduled SQL执行实例的执行状态，包括运行中（RUNNING）、重试中（STARTING）、成功（SUCCEEDED）、失败（FAILED）。
- 延迟执行：Scheduled SQL作业中的配置参数，表示在实例的调度时间点上，往后延迟N秒才真正开始执行实例。主要用于避免数据延迟等情况导致计算结果不精确问题。如果不需要使用延迟时间来保证结果正确性，您可以将**延迟执行**设置为0秒。

  例如，您设置**调度间隔**为**每小时**、**延迟执行**为**30**秒，那么一天生成24个实例，其中某实例的调度时间为2021/4/6 12:00:00，执行时间为2021/4/6 12:00:30。

调度与执行场景

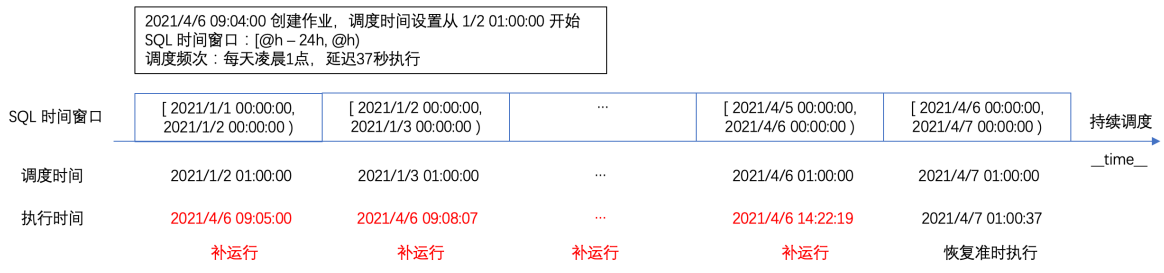
一个作业可生成多个实例，无论是正常被调度还是您触发异常实例重试的情况，同时只有一个实例处于运行中，不存在多个实例并发执行的情况。主要的调度与执行场景如下：

- 场景一：实例延迟执行

  无论实例是否延迟执行，实例的调度时间都是根据调度规则预先生成的。虽然前面的实例发生延迟时，可能导致后面的实例也延迟执行，但通过追赶执行进度，可逐渐减少延迟，直到恢复准时运行。

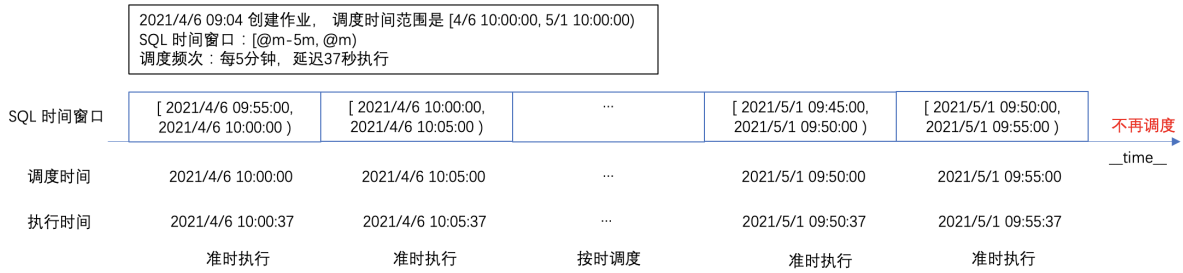
- 场景二：从某个历史时间点开始执行Scheduled SQL作业

  在当前时间点创建Scheduled SQL作业后，按照调度规则对历史数据进行处理，从调度的开始时间创建补运行的实例，补运行的实例依次执行直到追上数据处理进度后，再按照预定计划执行新实例。



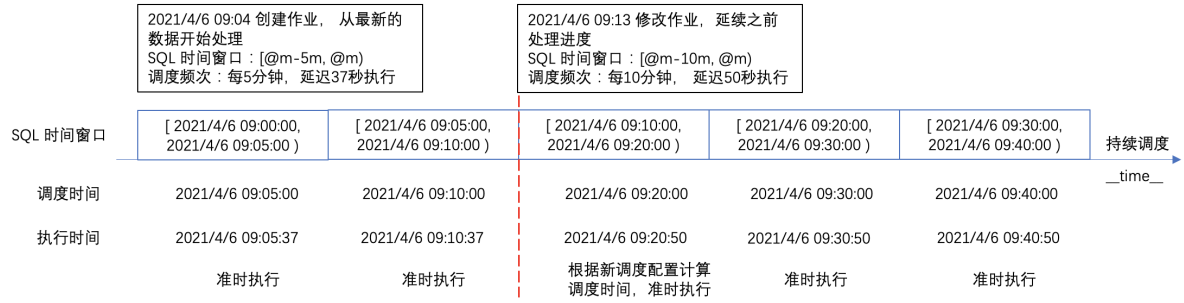
- 场景三：固定时间内执行Scheduled SQL作业

  如果需要对指定时间段的日志做调度，则可设置调度的时间范围。如果设置了调度的结束时间，则最后一个实例（调度时间小于调度结束时间）执行完成后，不再产生新的实例。



- 场景四：修改调度配置对生成实例的影响

  修改调度配置后，下一个实例按照新配置生成。一般建议同步修改SQL时间窗口、调度频率等配置，使得实例之间的SQL时间范围可以连续。



- 场景五：重试失败的实例

  正常情况下，一个Scheduled SQL作业按照调度时间的递增顺序生成执行实例。如果实例执行失败（例如权限不足、源库不存在、目标库不存在、SQL语法不合法），系统支持自动重试，当重试次数超过您配置的最大重试次数或重试时间超过您配置的最大运行时间时，重试结束，该实例状态被置为失败，然后系统继续执行下一个实例。

  您可以对失败的实例设置告警通知并进行手动重试。您可以对最近7天内创建的实例进行查看、重试操作。调度执行完成后，系统会根据实际执行情况变更实例状态为成功或失败。如何重试，请参见重试Scheduled SQL作业实例。

使用建议

使用Scheduled SQL时，建议根据业务情况，同时兼顾数据实时性和准确性。

- 考虑数据上传日志服务存在延迟情况，您可以结合数据采集延迟以及业务能够容忍的最大结果可见延迟，设置**执行延迟**和**SQL时间窗口**（结束时间往前一点），避免实例执行时SQL时间窗口内的数据未全部到达。
- 建议SQL时间窗口按分钟对齐（例如整分钟、整小时），以保证上传局部乱序数据时的数据准确度。

扩展阅读

日志服务汇总数据指南

13.2. 使用限制

本文介绍Scheduled SQL的使用限制。

查询与分析

> 注意 Scheduled SQL仅支持SQL独享版引擎。

限制项	说明
操作并发数	单个Project支持的最大分析操作并发数为150个。 例如150个用户同时在同一个Project的各个Logstore中执行分析操作。
数据量	单次分析最大支持扫描2000亿行数据。
数据生效机制	分析功能只对开启统计功能后写入的数据生效。 如果您需要分析历史数据，请对历史数据重建索引。更多信息，请参见重建索引。
返回结果	• 执行分析操作后，默认最多返回100行数据，超出部分不会返回。 • 如果您需要返回更多数据，请使用LIMIT 语法（最大支持返回100万行数据）。更多信息，请参见LIMIT 子句。超出LIMIT 语法限制的部分不会返回。 • 最大输出的数据量限制为20 GB，超出部分不会返回。
字段值大小	单个字段值默认为2048字节（2 KB），最大为16384字节（16 KB），超出部分不参与分析。 您可以在配置索引时，修改字段值的最大长度（64字节~16384字节）。具体操作，请参见配置索引。
超时时间	分析操作的最大超时时间为10分钟。
Double类型的字段值位数	Double类型的字段值最多52位。 如果浮点数编码位数超过52位，会造成精度损失。
模糊查询	执行模糊查询时，日志服务最多查询到符合条件的100个词，并返回包含这100个词并满足查询条件的所有日志。
查询不精确	结果不精确不会报错，会记录在实例状态以及作业执行记录（需手动开启）中。
数据延迟	当数据存在延迟时，可能存在数据漏查的风险。即如果某时间点的数据在对应的调度实例执行完成之后才到达，则在下一个调度实例中也不会被执行。更多信息，请参见如何保证SQL分析的数据准确性。
时间窗口	单次查询时间窗口最大为24小时，最小为1分钟。

数据写入

限制项	说明
目标Logstore写入阈值	如果写入数据时超过阈值，Scheduled SQL作业将重试10分钟以上。超过重试时间后，将返回错误信息。更多信息，请参见数据读写。
跨地域传输	中国内的跨地域传输数据时，网络较为稳定，但会有较高延迟（延迟大小随地域的不同而不同）。 国际网络无法保证。

作业执行

限制项	说明

限制项	说明
超时时间	最大超时时间为1800秒，超过将视为本次作业执行失败。 建议添加告警监控任务，便于及时发现问题重试错误实例。更多信息，请参见为Scheduled SQL作业设置告警、重试Scheduled SQL作业实例。
重试次数	最大重试次数为100次，超过将视为本次作业执行失败。
延迟执行	延迟执行时间最大为120秒，延迟执行使用场景实例请参见调度与执行场景。
历史执行记录	单个作业的历史执行记录最多保存14天。 建议添加告警监控任务，便于及时发现问题重试错误实例。更多信息，请参见为Scheduled SQL作业设置告警、重试Scheduled SQL作业实例。

13.3. 创建Scheduled SQL作业

13.3.1. 从Logstore到Logstore

Scheduled SQL功能用于定时分析数据、存储聚合数据、投影与过滤数据。日志服务支持源Logstore中的数据通过Scheduled SQL处理后存储到目标Logstore中。

前提条件

- 已采集数据到源Logstore。具体操作，请参见数据采集。
- 已创建目标Logstore。具体操作，请参见创建Logstore。
- 已配置源Logstore和目标Logstore的索引。具体操作，请参见配置索引。

操作步骤

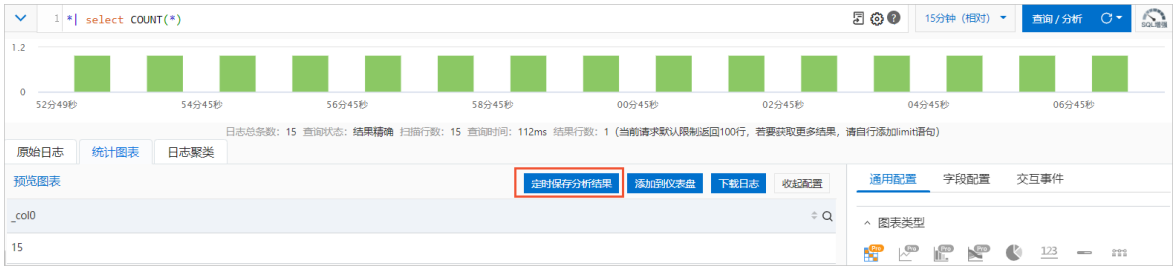
> **说明** 目前，Scheduled SQL功能在公测阶段，仅收取SQL独享版计算资源消耗费用。费用说明请参见计费项。

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 输入查询和分析语句，选择时间范围，然后单击**查询/分析**。

   查询分析语句由查询语句和分析语句构成，格式为查询语句|分析语句，查询分析语句语法请参见查询语法、SQL分析语法。

   > **说明** 本步骤为Scheduled SQL作业的预览操作，用于验证您所使用的查询和分析语句是否正确，执行结果是否有数据。

5. 在**统计图表**页签中，单击**定期保存分析结果**。



6. 创建Scheduled SQL作业。
   i. 在**计算配置**向导中，完成如下配置，然后单击**下一步**。

参数	描述
作业名	Scheduled SQL作业的名称。
作业描述	Scheduled SQL作业的描述。
资源池	日志服务提供增强型资源池用于数据分析。 增强型资源池复用SQL独享版的计算能力，提供足够的分析并发数，与您在控制台上的SQL分析操作进行资源隔离。增强型资源池根据SQL分析操作所消耗的CPU时间收取费用。更多信息，请参见开启SQL独享版。
写入模式	选择**日志库导入日志库**，即表示源Logstore中的数据通过Scheduled SQL处理后将存储到目标Logstore中。
SQL代码	显示您在步骤中输入的查询和分析语句。此处的预览操作与步骤中的操作一致，用于验证您所使用的查询和分析语句是否正确，执行结果是否有数据。 SQL作业运行时，日志服务将执行该查询和分析语句分析数据。
目标Region	目标Project所在地域。
目标Project	用于存储SQL分析结果的目标Project名称。
目标库	用于存储SQL分析结果的目标Logstore名称。
写目标授权	您可以通过如下方式授予Scheduled SQL作业写数据到目标Logstore的权限。 ■ **默认角色**：授权Scheduled SQL作业使用阿里云系统角色AliyunLogETLRole将运行结果写入目标Logstore。 **注意** 仅在首次配置时需要操作，且需要由目标Project所属的阿里云账号完成。 ■ **自定义角色**：授权Scheduled SQL作业使用自定义角色将运行结果写入目标Logstore。 您需先授予自定义角色写数据到目标Logstore的权限，然后在**角色ARN**中输入您自定义角色的ARN。如何获取ARN，请参见如下说明： ■ 如果源Logstore和目标Logstore属于同一阿里云账号，请参见步骤二：授予RAM角色写目标Logstore的权限。 ■ 如果源Logstore和目标Logstore属于不同的阿里云账号，请参见步骤二：授予账号B下的RAM角色b写目标Logstore的权限。

参数	描述
执行SQL授权	您可以通过如下方式授予Scheduled SQL作业读取源Logstore数据以及在当前Project下执行SQL分析操作的权限。 ■ **默认角色**：授权Scheduled SQL作业使用阿里云系统角色AliyunLogETLRole执行对应操作。 **注意** 仅在首次配置时需要操作，且需要由目标Project所属的阿里云账号完成。 ■ **自定义角色**：授权Scheduled SQL作业使用自定义角色执行对应操作。 您需先授予自定义角色相关权限，然后在**角色ARN**中输入您自定义角色的ARN。更多信息，请参见步骤一：授予RAM角色分析源Logstore的权限。

ii. 在**调度配置**向导中，完成如下配置，然后单击**确定**。

参数	描述
调度间隔	调度Scheduled SQL作业的频率，每调度一次Scheduled SQL作业产生一个执行实例。调度间隔决定每个执行实例的调度时间。 ■ **每小时**：每小时调度一次Scheduled SQL作业。 ■ **每天**：在每天的某个固定时间点调度一次Scheduled SQL作业。 ■ **每周**：在周几的某个固定时间点调度一次Scheduled SQL作业。 ■ **固定间隔**：按照固定间隔调度Scheduled SQL作业。 ■ **Cron**：通过Cron表达式指定时间间隔，按照指定的时间间隔调度Scheduled SQL作业。 Cron表达式的最小精度为分钟，24小时制，例如0 0/1 * * *表示从00:00开始，每隔1小时运行一次。 当您需要配置时区时，需选择**Cron**模式。常见的时区列表请参见时区列表。
调度时间范围	调度的时间范围，具体说明如下： ■ **某时间开始**：指定第一个实例被调度的开始时间。 ■ **特定时间范围**：指定实例被调度的起止时间，即Scheduled SQL作业仅在该时间范围内可被执行。 **注意** 实例的调度时间必须在该范围内，超出该范围时，Scheduled SQL作业不再产生新实例。
SQL时间窗口	Scheduled SQL作业运行时，仅分析该时间范围内的日志。时间窗口与实例调度时间共同作用生效。该时间范围不能大于**调度间隔**的5倍且不能超过1天。更多信息，请参见时间表达式语法。 例如，**调度间隔**为**固定间隔10分钟**，**起始时间**为**2021-04-01 00:00:00**，**延迟执行**为**30秒**，**SQL时间窗口**为**[@m-10m,@m)**，则SQL作业运行时，在00:00:30时刻生成第一个执行实例，分析的是[23:50:00~00:00:00)期间的日志。更多信息，请参见调度与执行场景。
SQL超时	执行SQL分析操作失败时自动重试的阈值。当重试时间超过指定的最大时间或者重试次数超过最大次数时，该执行实例结束，状态为失败。您可以根据失败原因，手动重试该实例。具体操作，请参见重试Scheduled SQL作业实例。
延迟执行	调度时间点往后延迟执行的时间。取整范围：0~120，单位：秒。 当数据写入Logstore存在延迟等情况时，可通过延迟执行来保证数据的完整性。

13.3.2. 从Logstore到MetricStore

Scheduled SQL功能用于定时分析数据、存储聚合数据、投影与过滤数据。日志服务支持源Logstore中的数据通过Scheduled SQL处理后存储到目标MetricStore中。

前提条件

- 已采集数据到源Logstore。具体操作，请参见数据采集。
- 已配置源Logstore的索引。具体操作，请参见配置索引。
- 已创建目标MetricStore。具体操作，请参见创建MetricStore。

操作步骤

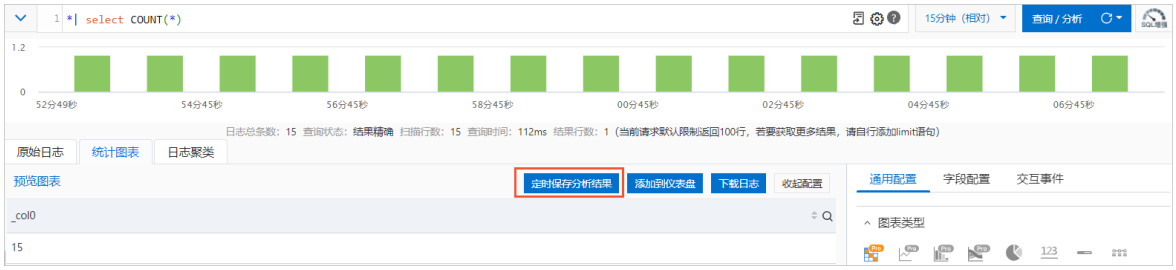
> **说明** 目前，Scheduled SQL功能在公测阶段，仅收取SQL独享版计算资源消耗费用。费用说明请参见计费项。

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 输入查询和分析语句，选择时间范围，然后单击**查询/分析**。

   查询分析语句由查询语句和分析语句构成，格式为查询语句|分析语句，查询分析语句语法请参见查询语法、SQL分析语法。

   > **说明** 本步骤为Scheduled SQL作业的预览操作，用于验证您所使用的查询和分析语句是否正确，执行结果是否有数据。

5. 在**统计图表**页签中，单击**定期保存分析结果**。



6. 创建Scheduled SQL作业。
   i. 在**计算配置**向导中，完成如下配置，然后单击**下一步**。

参数	描述
作业名	Scheduled SQL作业的名称。
作业描述	Scheduled SQL作业的描述。
资源池	日志服务提供增强型资源池用于数据分析。 增强型资源池复用SQL独享版的计算能力，提供足够的分析并发数，与您在控制台上的SQL分析操作进行资源隔离。增强型资源池根据SQL分析操作所消耗的CPU时间收取费用。更多信息，请参见开启SQL独享版。
写入模式	选择**日志库导入时序库**，即表示源Logstore中的数据通过Scheduled SQL处理后将存储到目标MetricStore中。
SQL代码	显示您在步骤中输入的查询和分析语句。此处的预览操作与步骤中的操作一致，用于验证您所使用的查询和分析语句是否正确，执行结果是否有数据。 SQL作业运行时，日志服务将执行该查询和分析语句分析数据。
SQL配置	
指标列	日志服务会根据您输入的查询和分析语句聚合数据，您可以选择查询和分析结果中列值为数值类型的一列或多列作为指标列。更多信息，请参见时序数据（Metric）。
Labels	日志服务会根据您输入的查询和分析语句聚合数据，您可以选择查询和分析结果中的一列或多列作为Label数据。更多信息，请参见时序数据（Metric）。

参数	描述
Rehash	打开**Rehash**开关后，您可以配置**哈希列**，用于将同一列值的数据写入到一个Shard中，增强数据局部性，提升查询效率。 **哈希列**的取值取决于查询和分析结果。您可以选择查询和分析结果中的一列或多列作为哈希列。例如您配置**哈希列**为status，则status字段值相同的数据将被写入到同一个Shard中。
时间列	■ 如果您选择查询和分析结果中的时间列（列值为Unixtime时间戳，例如`atime: 1627025331`），则系统将以该时间列作为时序数据的时间。 ■ 如果您选择**空**，则系统将以查询和分析时间范围中的开始时间作为时序数据的时间。 更多信息，请参见时序数据（Metric）。
附加Labels	添加静态标签，键值对形式，可用于标识指标的相关属性。 例如配置label_key为app，配置label_value为ingress-nginx。
目标	
目标Region	目标Project所在地域。
目标Project	用于存储SQL分析结果的目标Project名称。
目标库	用于存储SQL分析结果的目标MetricStore名称。
写目标授权	您可以通过如下方式授予Scheduled SQL作业写数据到目标MetricStore的权限。 ■ **默认角色**：授权Scheduled SQL作业使用阿里云系统角色AliyunLogETLRole将运行结果写入目标MetricStore。 **注意** 仅在首次配置时需要操作，且需要由目标Project所属的阿里云账号完成。 ■ **自定义角色**：授权Scheduled SQL作业使用自定义角色将运行结果写入目标MetricStore。 您需先授予自定义角色写数据到目标MetricStore的权限，然后在**角色ARN**中输入您自定义角色的ARN。详细说明如下： ■ 如果源Logstore和目标MetricStore属于同一阿里云账号，请参见步骤二：授予RAM角色写目标Logstore的权限。 ■ 如果源Logstore和目标MetricStore属于不同的阿里云账号，请参见步骤二：授予账号B下的RAM角色b写目标Logstore的权限。

参数	描述
执行SQL授权	您可以通过如下方式授予Scheduled SQL作业读取源Logstore数据以及在当前Project下执行SQL分析操作的权限。 ■ **默认角色**：授权Scheduled SQL作业使用阿里云系统角色AliyunLogETLRole执行对应操作。 **注意** 仅在首次配置时需要操作，且需要由目标Project所属的阿里云账号完成。 ■ **自定义角色**：授权Scheduled SQL作业使用自定义角色执行对应操作。 您需先授予自定义角色相关权限，然后在**角色ARN**中输入您自定义角色的ARN。更多信息，请参见步骤一：授予RAM角色分析源Logstore的权限。

ii. 在**调度配置**向导中，完成如下配置，然后单击**确定**。

参数	描述
调度间隔	调度Scheduled SQL作业的频率，每调度一次Scheduled SQL作业产生一个执行实例。调度间隔决定每个执行实例的调度时间。 ■ **每小时**：每小时调度一次Scheduled SQL作业。 ■ **每天**：在每天的某个固定时间点调度一次Scheduled SQL作业。 ■ **每周**：在周几的某个固定时间点调度一次Scheduled SQL作业。 ■ **固定间隔**：按照固定间隔调度Scheduled SQL作业。 ■ **Cron**：通过Cron表达式指定时间间隔，按照指定的时间间隔调度Scheduled SQL作业。 Cron表达式的最小精度为分钟，24小时制，例如0 0/1 * * *表示从00:00开始，每隔1小时运行一次。 当您需要配置时区时，需选择**Cron**模式。常见的时区列表请参见时区列表。
调度时间范围	调度的时间范围，具体说明如下： ■ **某时间开始**：指定第一个实例被调度的开始时间。 ■ **特定时间范围**：指定实例被调度的起止时间，即Scheduled SQL作业仅在该时间范围内可被执行。 **注意** 实例的调度时间必须在该范围内，超出该范围时，Scheduled SQL作业不再产生新实例。
SQL时间窗口	Scheduled SQL作业运行时，仅分析该时间范围内的日志。时间窗口与实例调度时间共同作用生效。该时间范围不能大于**调度间隔**的5倍且不能超过1天。更多信息，请参见时间表达式语法。 例如，**调度间隔**为**固定间隔10分钟**，**起始时间**为**2021-04-01 00:00:00**，**延迟执行**为**30秒**，**SQL时间窗口**为**[@m-10m,@m)**，则SQL作业运行时，在00:00:30时刻生成第一个执行实例，分析的是[23:50:00~00:00:00)期间的日志。更多信息，请参见调度与执行场景。
SQL超时	执行SQL分析操作失败时自动重试的阈值。当重试时间超过指定的最大时间或者重试次数超过最大次数时，该执行实例结束，状态为失败。您可以根据失败原因，手动重试该实例。具体操作，请参见重试Scheduled SQL作业实例。
延迟执行	调度时间点往后延迟执行的时间。取整范围：0~120，单位：秒。 当数据写入MetricStore存在延迟等情况时，可通过延迟执行来保证数据的完整性。

13.3.3. 从MetricStore到MetricStore

Scheduled SQL功能用于定时分析数据、存储聚合数据、投影与过滤数据。日志服务支持源MetricStore中的数据通过Scheduled SQL处理后存储到目标MetricStore中。

前提条件

- 已采集数据到源MetricStore。具体操作，请参见接入时序数据。
- 已创建目标MetricStore。具体操作，请参见创建MetricStore。

操作步骤

> **说明** 目前，Scheduled SQL功能在公测阶段，仅收取SQL独享版计算资源消耗费用。费用说明请参见计费项。

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**时序存储 > 时序库**页签中，单击目标MetricStore。
4. 查询和分析数据。
   i. 在页面右上角，单击**15分钟（相对）**，设置查询和分析的时间范围。
   ii. 在**查询配置**页签中选择目标指标，单击**预览**。

   > **说明** 本步骤为Scheduled SQL作业的预览操作，用于验证您所使用的查询和分析语句是否正确，执行结果是否有数据。

5. 单击**定时保存分析结果**。

6. 创建Scheduled SQL作业。
   i. 在**计算配置**向导中，完成如下配置，然后单击**下一步**。

参数	描述
作业名	Scheduled SQL作业的名称。
作业描述	Scheduled SQL作业的描述。
资源池	日志服务提供增强型资源池用于数据分析。 增强型资源池复用SQL独享版的计算能力，提供足够的分析并发数，与您在控制台上的SQL分析操作进行资源隔离。增强型资源池根据SQL分析操作所消耗的CPU时间收取费用。更多信息，请参见开启SQL独享版。
写入模式	选择**时序库导入时序库**，即表示源MetricStore中的数据通过Scheduled SQL处理后将存储到目标MetricStore中。
SQL代码	显示您在步骤中输入的查询和分析语句。此处的预览操作与步骤中的操作一致，用于验证您所使用的查询和分析语句是否正确，执行结果是否有数据。 SQL作业运行时，日志服务将执行该查询和分析语句分析数据。
SQL配置	

参数	描述
结果指标名	如果您要修改您所分析的指标名，您可以输入修改后的指标名。更多信息，请参见时序数据（Metric）。 **注意** 建议分析的对象为单个指标时，修改指标名，实现重命名。 如果分析对象为多个指标，则修改指标名后，会将所有的指标名修改为同一个相同的指标名。
Rehash	打开**Rehash**开关后，您可以配置**哈希列**，用于将同一Label值的数据写入到一个Shard中，增强数据局部性，提升查询效率。 **哈希列**的取值取决于时序数据已有的Label信息。例如时序数据已有的Label信息为`{"alert_id":"alert-1608815762-545495","alert_name":"告警恢复关闭","status":"inactive"}`，则**哈希列**的可选值为alert_id、alert_name、status。如果您配置**哈希列**为status，则status字段值相同的数据将被写入到同一个Shard中。
附加Labels	添加静态标签，键值对形式，可用于标识指标的相关属性。 例如配置label_key为app，配置label_value为ingress-nginx。
目标	
目标Region	目标Project所在地域。
目标Project	用于存储SQL分析结果的目标Project名称。
目标库	用于存储SQL分析结果的目标MetricStore名称。
写目标授权	您可以通过如下方式授予Scheduled SQL作业写数据到目标MetricStore的权限。 ■ **默认角色**：授权Scheduled SQL作业使用阿里云系统角色AliyunLogETLRole将运行结果写入目标MetricStore。 **注意** 仅在首次配置时需要操作，且需要由目标Project所属的阿里云账号完成。 ■ **自定义角色**：授权Scheduled SQL作业使用自定义角色将运行结果写入目标MetricStore。 您需先授予自定义角色写数据到目标MetricStore的权限，然后在**角色ARN**中输入您自定义角色的ARN。详细说明如下： ■ 如果源MetricStore和目标MetricStore属于同一阿里云账号，请参见步骤二：授予RAM角色写目标Logstore的权限。 ■ 如果源MetricStore和目标MetricStore属于不同的阿里云账号，请参见步骤二：授予账号B下的RAM角色b写目标Logstore的权限。

参数	描述
执行SQL授权	您可以通过如下方式授予Scheduled SQL作业读取源MetricStore数据以及在当前Project下执行SQL分析操作的权限。 ■ **默认角色**：授权Scheduled SQL作业使用阿里云系统角色AliyunLogETLRole执行对应操作。 **注意** 仅在首次配置时需要操作，且需要由目标Project所属的阿里云账号完成。 ■ **自定义角色**：授权Scheduled SQL作业使用自定义角色执行对应操作。 您需先授予自定义角色相关权限，然后在**角色ARN**中输入您自定义角色的ARN。更多信息，请参见步骤一：授予账号A下的RAM角色a分析源Logstore的权限。

ii. 在**调度配置**向导中，完成如下配置，然后单击**确定**。

参数	描述
调度间隔	调度Scheduled SQL作业的频率，每调度一次Scheduled SQL作业产生一个执行实例。调度间隔决定每个执行实例的调度时间。 ■ **每小时**：每小时调度一次Scheduled SQL作业。 ■ **每天**：在每天的某个固定时间点调度一次Scheduled SQL作业。 ■ **每周**：在周几的某个固定时间点调度一次Scheduled SQL作业。 ■ **固定间隔**：按照固定间隔调度Scheduled SQL作业。 ■ **Cron**：通过Cron表达式指定时间间隔，按照指定的时间间隔调度Scheduled SQL作业。 Cron表达式的最小精度为分钟，24小时制，例如0 0/1 * * *表示从00:00开始，每隔1小时运行一次。 当您需要配置时区时，需选择**Cron**模式。常见的时区列表请参见时区列表。
调度时间范围	调度的时间范围，具体说明如下： ■ **某时间开始**：指定第一个实例被调度的开始时间。 ■ **特定时间范围**：指定实例被调度的起止时间，即Scheduled SQL作业仅在该时间范围内可被执行。 **注意** 实例的调度时间必须在该范围内，超出该范围时，Scheduled SQL作业不再产生新实例。
SQL时间窗口	Scheduled SQL作业运行时，仅分析该时间范围内的日志。时间窗口与实例调度时间共同作用生效。该时间范围不能大于**调度间隔**的5倍且不能超过1天。更多信息，请参见时间表达式语法。 例如，**调度间隔**为**固定间隔10分钟**，**开始时间**为**2021-04-01 00:00:00**，**延迟执行**为**30秒**，**SQL时间窗口**为**[@m-10m,@m)**，则SQL作业运行时，在00:00:30时刻生成第一个执行实例，分析的是[23:50:00~00:00:00)期间的日志。更多信息，请参见调度与执行场景。
SQL超时	执行SQL分析操作失败时自动重试的阈值。当重试时间超过指定的最大时间或者重试次数超过最大次数时，该执行实例结束，状态为失败。您可以根据失败原因，手动重试该实例。具体操作，请参见重试Scheduled SQL作业实例。
延迟执行	调度时间点往后延迟执行的时间。取整范围：0~120，单位：秒。 当数据写入MetricStore存在延迟等情况时，可通过延迟执行来保证数据的完整性。

13.4. 管理Scheduled SQL作业

您可以在Scheduled SQL的管理页面进行查看Scheduled SQL作业基础信息、查看执行实例、重试、修改或删除Scheduled SQL作业等操作。

前提条件

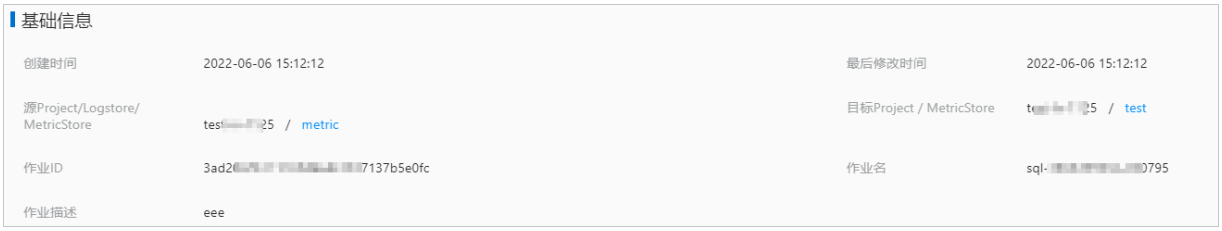
已创建Scheduled SQL作业。具体操作，请参见从Logstore到Logstore、从Logstore到MetricStore或从MetricStore到MetricStore。

操作入口

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在左侧导航栏，选择**作业 > Scheduled SQL**。
4. 单击目标Scheduled SQL作业。

查看Scheduled SQL作业基础信息

您可以**基础信息**区域查看Scheduled SQL作业创建时间、最后修改时间、作业ID等信息。



查看执行实例

在Scheduled SQL作业运行期间，日志服务会根据调度间隔生成多个执行实例。您可以在**执行实例**区域查看目标Scheduled SQL作业的所有执行实例。

参数	说明
实例ID	Scheduled SQL执行实例的唯一标识。
作业执行时间	Scheduled SQL作业的运行时间。
SQL查询区范围	SQL分析的时间范围，即该实例分析的是该时间范围内的数据。
处理数据量	SQL分析相关的数据量。具体说明如下： • **SQL处理行数**：该执行实例在SQL时间窗口内读取到的日志行数。参与计算的数据量。 • **SQL结果行数**：该执行实例在执行SQL分析后，分析结果中对应的日志行数。写入目标库（Logstore、MetricStore）的数据量。 • **SQL处理字节**：该执行实例在SQL时间窗口内读取到的日志字节数。参与计算的数据量。 • **Logstore行数**：该实例将SQL分析结果成功写入目标日志库的行数。实际写入目标Logstore的数据量。
执行状态	Scheduled SQL执行实例的执行状态，包括运行中、重试中、成功、失败。

重试Scheduled SQL作业实例

当Scheduled SQL作业实例状态为成功或失败时，您可以重新运行Scheduled SQL作业实例。例如某实例状态为失败，且失败原因为授权不正确，则您可以修改授权配置后，单击目标实例对应的**重试**。

> 注意 一般在补数据场景，可对执行成功的实例进行重试，但建议谨慎操作。一般情况下，只有失败的作业需要重试。

删除Scheduled SQL作业

如果您不再需要运行该Scheduled SQL作业，您可以在Scheduled SQL的管理页面的右上角单击**删除**。

> 警告 删除Scheduled SQL作业后，不可恢复，请谨慎操作。

修改Scheduled SQL作业

如果您需要修改该Scheduled SQL作业的相关配置，您可以在Scheduled SQL的管理页面的右上角单击**修改配置**。相关参数说明，请参见从Logstore到Logstore、从Logstore到MetricStore或从MetricStore到MetricStore。

13.5. 为Scheduled SQL作业设置告警

日志服务Scheduled SQL已内置监控规则模板，您只需添加对应的告警实例即可实时监控Scheduled SQL作业，并可通过钉钉等渠道接收到告警通知。本文介绍设置告警的相关操作。

前提条件

已创建Scheduled SQL作业。

- 如果是将源Logstore中的数据通过Scheduled SQL处理后存储到目标Logstore，请参见从Logstore到Logstore。
- 如果是将源Logstore中的数据通过Scheduled SQL处理后存储到目标MetricStore中，请参见从Logstore到MetricStore。
- 如果是将源MetricStore中的数据通过Scheduled SQL处理后存储到目标MetricStore中，请参见从MetricStore到MetricStore。

步骤一：开启作业运行日志

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。

   该Project为Scheduled SQL作业所在的Project。
3. 在页面左上方，单击图标。
4. 进入开通作业运行日志页面。
   - 如果您开未通过该Project的详细日志，则在**服务日志**页签中，单击**开通服务日志**。
   - 如果您已开通过该Project的详细日志，则在**服务日志**页签中，单击图标。
5. 设置如下参数，然后单击**确定**。

参数	说明
作业运行日志	打开**作业运行日志**开关后，系统将在您指定的Project中自动创建一个名为internal-diagnostic_log的Logstore，用于存储Scheduled SQL、MaxCompute投递、OSS投递、数据导入等作业的运行日志与错误日志。日志字段说明，请参见Scheduled SQL作业运行日志。

参数	说明
日志存储位置	开通作业运行日志功能后，需要选择日志的存储位置，即需要指定Project。可以设置为： ○ 自动创建（推荐）。 ○ 当前Project。 ○ 同一地域下的其他Project。

开通作业运行日志后，如果Scheduled SQL任务执行失败，您可以在指定Project下的internal-diagnostic_log Logstore中查看Scheduled SQL作业的错误日志。其中Scheduled SQL作业的错误日志的日志主题（__topic__）为scheduled_sql_alert。

步骤二：配置行动策略

1. 登录日志服务控制台。
2. 进入行动策略管理页面。
   i. 在Project列表区域，单击任意的Project。
   ii. 在左侧导航栏中，单击**告警**。
   iii. 选择**告警管理 > 行动策略**。
3. 找到目标行动策略（sls.app.scheduled_sql.builtin），单击**修改**。

   您也可以创建新的行动策略用于告警通知。具体操作，请参见创建行动策略。
4. 在**编辑行动策略**页面中，将**请求地址**修改为钉钉群机器人的Webhook地址。其他选项，保持默认配置。

   如何获取钉钉群机器人的WebHook地址，请参见钉钉-自定义。您也可以根据业务需求，使用其他告警渠道。具体操作，请参见通知渠道说明。
5. 单击**确认**。

步骤三：添加告警实例

日志服务已内置如下两种监控规则模板，您只需根据业务需求，添加对应的告警实例即可。两种告警实例的配置参数类似，此处以添加ScheduledSQL任务执行延迟监控规则对应的告警实例为例。

- ScheduledSQL任务执行错误监控：每5分钟检测一次，当Scheduled SQL任务出现错误后，触发告警。
- ScheduledSQL任务执行延迟监控：每5分钟检测一次，当Scheduled SQL任务的延迟时间超过指定阈值后，触发告警。

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。

   该Project为您internal-diagnostic_log Logstore所在的Project。
3. 在左侧导航栏中，单击**告警**。
4. 在**规则/事务**页签中，单击**SLS ScheduledSQL(2)**。

5. 在规则列表中，单击**ScheduledSQL任务执行延迟监控**对应的**添加**。
6. 在**参数设置**对话框中，配置监控规则，然后单击**设置并开启**。

参数	说明
告警名称	告警名称，支持自定义。
延迟阈值	ScheduledSQL任务执行的延迟时间超过该阈值后，触发告警。默认值为10分钟。
监控的Project	需监控的Project名称。 ○ 默认值为 `.*` ，表示监控您当前阿里云账号下的所有Project。 ○ 多个Project之间可以使用竖线（|）分隔。您还可以使用正则表达式 `.*` 进行配置，例如sche.*，表示监控以sche开头的Project。
监控的任务名称	需监控的Scheduled SQL任务名称。 ○ 默认值为 `.*` ，表示监控您所指定的Project下的所有Scheduled SQL任务。 ○ 多个Scheduled SQL任务之间可以使用竖线（|）分隔。您还可以使用正则表达式 `.*` 进行配置，例如sche.*，表示监控以sche开头的Scheduled SQL任务。
行动策略	当前告警所绑定的行动策略，日志服务将通过该行动策略给指定用户发送告警通知。 默认为 `sls.app.scheduled_sql.builtin` （SLS Scheduled SQL内置行动策略）。您也可以自定义行动策略。具体操作，请参见创建行动策略。
严重度	定义告警消息的严重度。

添加完成后，您可以在监控规则列表中，单击**ScheduledSQL任务执行延迟监控**规则对应的**+**，查看已开启的告警实例。

13.6. 查询Scheduled SQL结果数据

本文介绍如何在目标Logstore或MetricStore中查询Scheduled SQL结果数据。

前提条件

已创建Scheduled SQL作业。

- 如果是将源Logstore中的数据通过Scheduled SQL处理后存储到目标Logstore，请参见从Logstore到Logstore。
- 如果是将源Logstore中的数据通过Scheduled SQL处理后存储到目标MetricStore中，请参见从Logstore到MetricStore。
- 如果是将源MetricStore中的数据通过Scheduled SQL处理后存储到目标MetricStore中，请参见从MetricStore到MetricStore。

在目标Logstore中查询

当您将Scheduled SQL结果数据存储到Logstore后，您可以在目标Logstore中通过Scheduled SQL作业名查询目标Scheduled SQL结果数据。

1. 登录日志服务控制台。
2. 获取Scheduled SQL作业名。
   i. 在Project列表区域，单击目标Project。

ii. 在左侧导航栏中，选择**作业 > Scheduled SQL**，然后在**Scheduled SQL**列表中，单击目标作业。

iii. 在**基础信息**区域，获取Scheduled SQL作业名。

3. 查询Scheduled SQL结果数据。

i. 找到用于存储Scheduled SQL结果数据的Logstore。具体操作，请参见查询和分析日志。

ii. 在输入框中输入 `__tag__:__job__: job-name` 。

其中*job-name*为Scheduled SQL作业名，请根据您在步骤中获取的Scheduled SQL作业名进行替换。

iii. 单击**15分钟（相对）**，设置查询的时间范围。

> **说明** 如果没有查询到数据，可以适当增大查询的时间范围。

iv. 单击**查询/分析**。

在目标MetricStore中查询

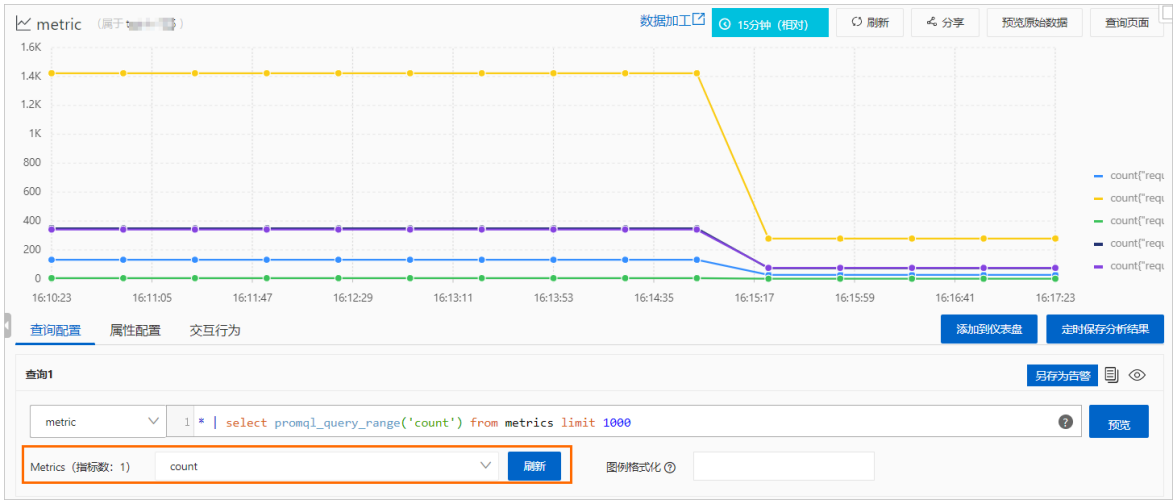
当您将Scheduled SQL结果数据存储到MetricStore后，您可以在目标MetricStore中通过您设置的指标查询目标Scheduled SQL结果数据。

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**时序存储 > 时序库**页签中，单击目标MetricStore。
4. 单击**15分钟（相对）**，设置查询的时间范围。

> **说明** 如果没有查询到数据，可以适当增大查询的时间范围。

5. 在**查询配置**页签中，选择对应的指标，然后单击**预览**。

该指标为您在创建Scheduled SQL作业时设置的指标。



13.7. 授予RAM用户操作Scheduled SQL的权限

本文介绍如何授予RAM用户操作Scheduled SQL。

前提条件

已创建RAM用户。具体操作，请参见步骤一：创建RAM用户。

操作步骤

1. 登录RAM控制台。
2. 创建权限策略。
   i. 在左侧导航栏中，选择**权限管理 > 权限策略**。
   ii. 单击**创建权限策略**。
   iii. 在**创建权限策略**页面的**脚本编辑**页签中，将配置框中的原有脚本替换为如下内容，然后单击**下一步**。

   请根据实际情况替换脚本中的*Project名称*和*Logstore名称*。

   > **注意**
   > - 如果您要使用RAM用户配置Scheduled SQL作业告警，还需授予RAM用户告警操作权限。更多信息，请参见授予RAM用户告警操作权限。
   > - 权限策略中的Logstore包括了Logstore和MetricStore。当您的操作对象为MetricStore时，如下策略同样适用。

```
{
    "Version": "1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "log:GetJobInstance",
                "log:ModifyJobInstance",
                "log:ListJobInstance"
            ],
            "Resource": "acs:log:*:*:project/Project名称/job/*/jobinstance/*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "log:*"
            ],
            "Resource": "acs:log:*:*:project/Project名称/job/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "log:CreateJob"
            ],
            "Resource": "acs:log:*:*:project/Project名称/job/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "log:ListLogStores",
                "log:ListSavedSearch",
                "log:ListDashboard"
            ],
            "Resource": "acs:log:*:*:project/Project名称/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "log:GetLogStore",
                "log:GetIndex",
                "log:GetLogStoreHistogram",
                "log:GetLogStoreLogs"
            ],
            "Resource": "acs:log:*:*:project/Project名称/logstore/Logstore名称"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ram:PassRole",
                "ram:GetRole",
                "ram:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "log:CreateLogStore",
                "log:CreateIndex",
                "log:UpdateIndex"
            ],
            "Resource": [
                "acs:log:*:*:project/sls-alert-*/logstore/internal-alert-center-log"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "log:CreateDashboard",
                "log:CreateChart",
                "log:UpdateDashboard"
            ],
```

```
            "Resource": [
                "acs:log:*:*:project/sls-alert-*/dashboard/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "log:CreateProject"
            ],
            "Resource": [
                "acs:log:*:*:project/sls-alert-*"
            ]
        }
    ]
}
```

iv. 设置**名称**，然后单击**确定**。

3. 为RAM用户授权。
   i. 在左侧导航栏中，选择**身份管理 > 用户**。
   ii. 找到目标RAM用户，单击**添加权限**。
   iii. 在**添加权限**面板的**选择权限**区域，单击**自定义策略**，选中步骤中创建的权限策略。
   iv. 单击**确定**。

13.8. 配置自定义角色权限（同账号场景）

日志服务Scheduled SQL作业运行时，将在源Logstore中执行SQL分析操作，并将分析结果保存到目标存储中。您可以授予Scheduled SQL作业使用自定义角色来完成以上操作。如果源Logstore和目标Logstore属于同一个阿里云账号，您可参考本文档完成授权操作。Scheduled SQL作业支持Logstore和MetricStore，本文以Logstore为例进行说明。

前提条件

已创建RAM角色。具体操作，请参见步骤一：创建RAM角色。

步骤一：授予RAM角色分析源Logstore的权限

授予RAM角色分析源Logstore的权限后，Scheduled SQL作业可以使用该角色在Logstore中执行SQL分析操作。

1. 使用阿里云账号登录RAM控制台。
2. 创建具备分析源Logstore日志权限的策略。
   i. 在左侧导航栏，选择**权限管理 > 权限策略**。
   ii. 单击**创建权限策略**。
   iii. 在**新建自定义权限策略**页面中，配置如下参数，并单击**确定**。

参数	说明
策略名称	配置策略名称。例如log-scheduled-sql-policy。
配置模式	选择**脚本配置**。
	将配置框中的原有脚本替换为如下内容。

参数	说明

策略内容	■ 精确授权。
	例如源Project名称为log-project-prod，源Logstore名称为website_log。在实际场景中，请根据实际情况替换Project名称和Logstore名称。
	注意 权限策略中的Logstore包括了Logstore和MetricStore。当您的操作对象为MetricStore时，如下策略同样适用。
	{
	"Version":"1",
	"Statement":[
	{
	"Action":[
	"log:PostProjectQuery"
	],
	"Resource":[
	"acs:log:*:*:project/log-project-prod/logstore/website_log",
	"acs:log:*:*:project/log-project-prod/logstore/website_log/*"
	],
	"Effect":"Allow"
	},
	{
	"Action":[
	"log:GetProjectQuery",
	"log:PutProjectQuery",
	"log:DeleteProjectQuery"
	],
	"Resource":[
	"acs:log:*:*:project/log-project-prod"
	],
	"Effect":"Allow"
	}
	]
	}

参数	说明
	■ 模糊匹配授权。 例如源Project名称为log-project-dev-a、log-project-dev-b、log-project-dev-c等，源Logstore名称为website_a_log、website_b_log、website_c_log等，则您可以使用模糊匹配授权。在实际场景中，请根据实际情况替换Project名称和Logstore名称。 `{ "Version":"1", "Statement":[{ "Action":["log:PostProjectQuery"], "Resource":["acs:log:*:*:project/log-project-dev-*/logstore/website_*_log", "acs:log:*:*:project/log-project-dev-*/logstore/website_*_log/*"], "Effect":"Allow" }, { "Action":["log:GetProjectQuery", "log:PutProjectQuery", "log:DeleteProjectQuery"], "Resource":["acs:log:*:*:project/log-project-dev-*"], "Effect":"Allow" }] }`

3. 为RAM角色授权。
   i. 在左侧导航栏中，选择**身份管理 > 角色**。
   ii. 单击目标RAM角色对应的**添加权限**。
   iii. 选择**自定义策略**，并选中步骤2中创建的权限策略（例如log-scheduled-sql-policy），单击**确定**。
   iv. 确认授权结果，单击**完成**。
4. 获取RAM角色标识（ARN）。
   在该角色的**基本信息**中查看，例如acs:ram::13****44:role/logrole。请记录该信息，如果您在创建Scheduled SQL作业时使用的是自定义角色，则需要输入该信息。

步骤二：授予RAM角色写目标Logstore的权限

授予RAM角色写目标Logstore的权限后，Scheduled SQL作业可以使用该角色将SQL分析结果写入到目标Logstore中。

1. 使用阿里云账号登录RAM控制台。
2. 创建具备访问目标Logstore权限的策略。
   i. 在左侧导航栏，选择**权限管理 > 权限策略**。
   ii. 单击**创建权限策略**。

iii. 在**新建自定义权限策略**页面中，配置如下参数，并单击**确定**。

参数	说明
策略名称	配置策略名称。例如log-sink-write-policy。
配置模式	选择**脚本配置**。

策略内容	将配置框中的原有脚本替换为如下内容。
	■ 精确授权。
	例如目标Project名称为log-project-prod，目标Logstore名称为website_log_output。在实际场景中，请根据实际情况替换Project名称和Logstore名称。
	注意 权限策略中的Logstore包括了Logstore和MetricStore。当您的操作对象为MetricStore时，如下策略同样适用。
	`{ "Version": "1", "Statement": [ { "Action": [ "log:Post*", "log:BatchPost*" ], "Resource": "acs:log:*:*:project/log-project-prod/logstore/website_log_output", "Effect": "Allow" } ] }`
	■ 模糊匹配授权。
	例如目标Project名称为log-project-dev-a、log-project-dev-b、log-project-dev-c等，目标Logstore名称为website_a_log_output、website_b_log_output、website_c_log_output等，则您可以使用模糊匹配授权。在实际场景中，请根据实际情况替换Project名称和Logstore名称。
	`{ "Version": "1", "Statement": [ { "Action": [ "log:Post*", "log:BatchPost*" ], "Resource": "acs:log:*:*:project/log-project-dev-*/logstore/website_*_log_output", "Effect": "Allow" } ] }`

3. 为RAM角色授权。
   i. 在左侧导航栏中，选择**身份管理 > 角色**。
   ii. 单击目标RAM角色对应的**添加权限**。

iii. 选择**自定义策略**，并选中步骤2中创建的权限策略（例如log-sink-write-policy），单击**确定**。

iv. 确认授权结果，单击**完成**。

4. 获取RAM角色标识（ARN）。

   在该角色的**基本信息**中查看，例如acs:ram::13****44:role/logrole。请记录该信息，如果您在创建Scheduled SQL作业时使用的是自定义角色，则需要输入该信息。

13.9. 配置自定义角色权限（跨账号场景）

日志服务Scheduled SQL作业运行时，将在源Logstore中执行SQL分析操作，并将分析结果保存到目标存储中。您可以授予日志服务使用自定义角色来完成以上操作。如果源Logstore和目标Logstore不属于同一个阿里云账号，您可参考本文档完成授权操作。Scheduled SQL作业支持Logstore和MetricStore，本文以Logstore为例进行说明。

前提条件

已在阿里云账号A下创建RAM角色a，在账号B下创建RAM角色b。具体操作，请参见步骤一：创建RAM角色。

步骤一：授予账号A下的RAM角色a分析源Logstore的权限

授予账号A下的RAM角色a分析源Logstore的权限后，Scheduled SQL作业可以使用该角色在账号A下的Logstore中执行SQL分析操作。

1. 使用账号A登录RAM控制台。
2. 创建具备分析源Logstore日志权限的策略。
   i. 在左侧导航栏，选择**权限管理 > 权限策略**。
   ii. 单击**创建权限策略**。
   iii. 在**新建自定义权限策略**页面中，配置如下参数，并单击**确定**。

参数	说明
策略名称	配置策略名称。例如log-scheduled-sql-policy。
配置模式	选择**脚本配置**。
	将配置框中的原有脚本替换为如下内容。

参数	说明

策略内容	■ 精确授权。
	例如源Project名称为log-project-prod，源Logstore名称为website_log。在实际场景中，请根据实际情况替换Project名称和Logstore名称。
	注意 权限策略中的Logstore包括了Logstore和MetricStore。当您的操作对象为MetricStore时，如下策略同样适用。
	{
	"Version":"1",
	"Statement":[
	{
	"Action":[
	"log:PostProjectQuery"
	],
	"Resource":[
	"acs:log:*:*:project/log-project-prod/logstore/website_log",
	"acs:log:*:*:project/log-project-prod/logstore/website_log/*"
	],
	"Effect":"Allow"
	},
	{
	"Action":[
	"log:GetProjectQuery",
	"log:PutProjectQuery",
	"log:DeleteProjectQuery"
	],
	"Resource":[
	"acs:log:*:*:project/log-project-prod"
	],
	"Effect":"Allow"
	}
	]
	}

参数	说明
	■ 模糊匹配授权。 例如源Project名称为log-project-dev-a、log-project-dev-b、log-project-dev-c等，源Logstore名称为website_a_log、website_b_log、website_c_log等，则您可以使用模糊匹配授权。在实际场景中，请根据实际情况替换Project名称和Logstore名称。 `{ "Version":"1", "Statement":[{ "Action":["log:PostProjectQuery"], "Resource":["acs:log:*:*:project/log-project-dev-*/logstore/website_*_log", "acs:log:*:*:project/log-project-dev-*/logstore/website_*_log/*"], "Effect":"Allow" }, { "Action":["log:GetProjectQuery", "log:PutProjectQuery", "log:DeleteProjectQuery"], "Resource":["acs:log:*:*:project/log-project-dev-*"], "Effect":"Allow" }] }`

3. 为RAM角色a授权。
   i. 在左侧导航栏中，选择**身份管理 > 角色**。
   ii. 单击RAM角色a对应的**添加权限**。
   iii. 选择**自定义策略**，并选中步骤中创建的权限策略（例如log-scheduled-sql-policy），单击**确定**。
   iv. 确认授权结果，单击**完成**。
4. 获取RAM角色a的ARN。
   在该角色的**基本信息**中查看，例如acs:ram::11****27:role/logsource。请记录该信息，如果您在创建Scheduled SQL作业时使用的是自定义角色，则需要输入该信息。

步骤二：授予账号B下的RAM角色b写目标Logstore的权限

授予账号B下的RAM角色b写目标Logstore的权限后，Scheduled SQL作业可以使用该角色将账号A下的SQL分析结果写入到账号B下的Logstore中。

1. 使用账号B登录RAM控制台。
2. 创建具备写目标Logstore权限的策略。
   i. 在左侧导航栏，选择**权限管理 > 权限策略**。
   ii. 单击**创建权限策略**。

iii. 在**新建自定义权限策略**页面中，配置如下参数，并单击**确定**。

参数	说明
策略名称	配置策略名称。例如log-sink-write-policy。
配置模式	选择**脚本配置**。

策略内容	将配置框中的原有脚本替换为如下内容。
	■ 精确授权。
	例如目标Project名称为log-project-prod，目标Logstore名称为website_log_output。在实际场景中，请根据实际情况替换Project名称和Logstore名称。
	注意 权限策略中的Logstore包括了Logstore和MetricStore。当您的操作对象为MetricStore时，如下策略同样适用。
	`{ "Version": "1", "Statement": [ { "Action": [ "log:Post*", "log:BatchPost*" ], "Resource": "acs:log:*:*:project/log-project-prod/logstore/website_log_output", "Effect": "Allow" } ] }`
	■ 模糊匹配授权。
	例如目标Project名称为log-project-dev-a、log-project-dev-b、log-project-dev-c等，目标Logstore名称为website_a_log_output、website_b_log_output、website_c_log_output等，则您可以使用模糊匹配授权。在实际场景中，请根据实际情况替换Project名称和Logstore名称。
	`{ "Version": "1", "Statement": [ { "Action": [ "log:Post*", "log:BatchPost*" ], "Resource": "acs:log:*:*:project/log-project-dev-*/logstore/website_*_log_output", "Effect": "Allow" } ] }`

3. 为RAM角色b授权。
   i. 在左侧导航栏中，选择**身份管理 > 角色**。
   ii. 单击目标RAM角色b对应的**添加权限**。

iii. 选择**自定义策略**，并选中步骤中创建的权限策略（例如log-sink-write-policy），单击**确定**。

iv. 确认授权结果，单击**完成**。

4. 修改RAM角色b的信任策略。

i. 在RAM角色列表中，单击RAM角色b。

ii. 在**信任策略管理**页签中，单击**修改信任策略**。

iii. 修改信任策略。

在Service配置项中添加**阿里云账号A的ID**，并根据实际情况替换该值。您可以在账号中心查看阿里云账号ID。

该策略表示账号A有权限通过日志服务获取临时Token来操作账号B中的资源。

```
{
    "Statement": [
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "log.aliyuncs.com",
                    "阿里云账号A的ID@log.aliyuncs.com"
                ]
            }
        }
    ],
    "Version": "1"
}
```

5. 获取RAM角色b的ARN。

在该角色的**基本信息**中查看，例如acs:ram::13****44:role/logtarget。请记录该信息，如果您在创建Scheduled SQL作业时使用的是自定义角色，则需要输入该信息。

13.10. Scheduled SQL Exactly-Once

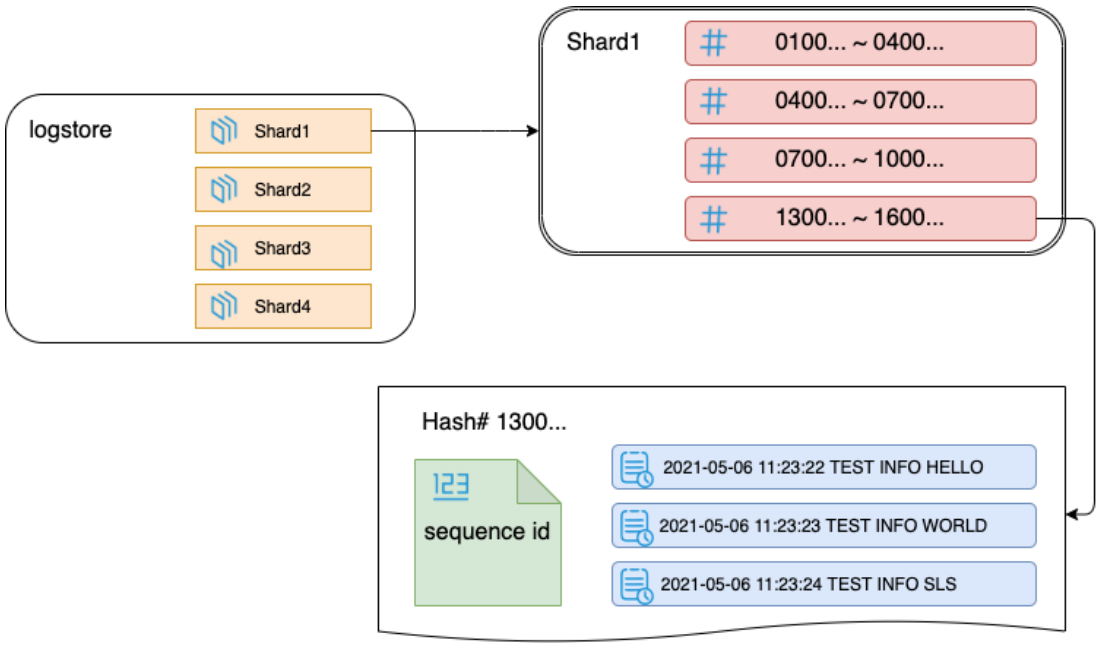
日志服务Scheduled SQL保证每次作业的计算结果都以Exactly-Once方式写入到目标存储库（Logstore和MetricStore），确保数据不会重复写入，也不会丢失。日志服务基于Logstore的幂等写入来实现Scheduled SQL结果数据的Exactly-Once。

> **说明** 本文内容适用于Logstore和MetricStore，仅以Logstore为例进行说明。

背景信息

日志服务Logstore中包含多个Shard，每个Shard对应一个Hash Key区间（左闭右开）。您可通过负载均衡模式或者指定Hash Key的模式写数据到Logstore，数据最终落到某个Hash Key中。更多信息，请参见分区（Shard）。

- 负载均衡模式：每个数据包随机写入当前可用的Shard中。
- 指定Hash Key模式（幂等写入）：指定Hash Key的范围，数据将被写入包含该Hash Key范围的Shard中。

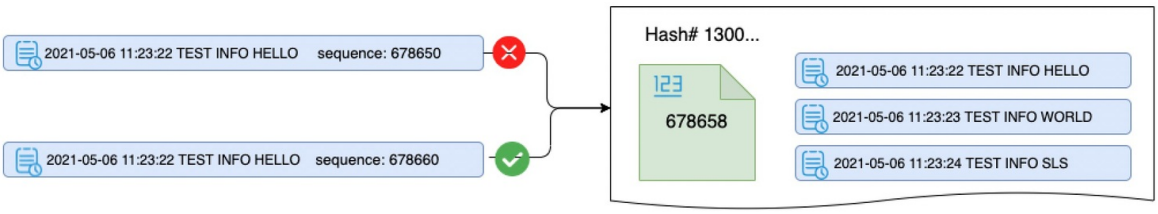


幂等写入

日志服务通过指定hash key和sequence id，实现数据的幂等写入。

- hash key用于指定数据写入的Hash Key范围。
- sequence id用于指定待写入数据在该Hash Key中的ID。

  您需保证Sequence ID单调增长，进而保证数据的幂等写入，避免写入作业多次重试时导致数据重复。
  - 当您将数据写入到相同的Hash Key中时，如果待写入数据的Sequence ID小于等于Hash Key中记录的最近一次写入的数据的Sequence ID，则日志服务将拒绝数据写入并返回错误。
  - 当您将数据写入到相同的Hash Key中时，如果待写入数据的Sequence ID大于Hash Key中记录的最近一次写入的数据的Sequence ID，则日志服务将允许数据写入，并更新Hash Key中的Sequence ID。

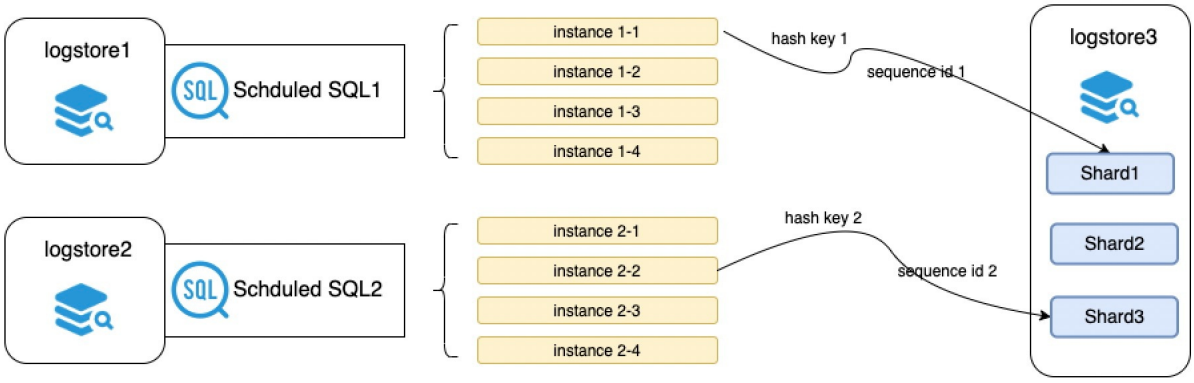


基于幂等写入的Scheduled SQL Exactly-Once

Scheduled SQL采用幂等写入方式向目标Logstore写数据。在幂等写入方式的基础上，Scheduled SQL会为每次作业的计算结果生成具有以下特性的Hash Key以及Sequence ID，从而实现Exactly-Once写入。

- 为两个不同的Scheduled SQL作业实例生成不同的Hash Key。

  当多个Scheduled SQL作业向同一个Logstore写入数据时，Scheduled SQL会为各个作业生成不同的Hash Key，避免Sequence ID冲突。
- 为同一个Scheduled SQL作业中的不同实例生成递增的Sequence ID。



注意 当您通过以下方式向同一个Logstore中的同一个Hash Key写入数据时，可能出现Sequence ID冲突，导致其中一方写入数据失败，引起数据缺失问题。

- 通过API或SDK写入数据，并指定hash key参数和sequence id参数。
- 通过Logtail写入数据，并开启ExactlyOnce功能。具体操作，请参见Logtail配置。
- 通过Scheduled SQL作业写入数据。

13.11. 时间表达式语法

您在创建Scheduled SQL作业时，可指定SQL时间窗口。Scheduled SQL作业运行时，日志服务仅分析该SQL时间窗口内的日志。本文介绍SQL时间窗口相关的时间表达式语法。

操作符

时间表达式支持的操作符如下表所示：

操作符	说明
+	加号
-	减号
@	取整操作符，根据时间向下取整。例如以小时为单位对时间01:40进行取整，取整后为01:00。

时间表达式的计算单元为±{num}{unit}或@{unit}，其中{num}为正整数，{unit}为时间单位。

- 如果操作符为加号（+）、减号（-），则计算单元的格式为±{num}{unit}。其中{num}可省略，省略后的默认值为1。例如时间表达式为-h，则表示减1小时。
- 如果操作符为at符号（@），则计算单元的格式为@{unit}。

时间单位

时间表示式所支持的时间单位如下表所示：

时间单位	说明
h	时
m	分
s	秒

示例

时间表达式示例如下表所示：

时间表达式	说明
-15m@m	先减15分钟再向下取整到分钟。 例如，创建Scheduled SQL作业时，配置**调度间隔**为**每天00:00**，**延迟执行**为**30秒**，**SQL时间窗口**为**[-15m@m,-5m@m)**，则表示在00:00:30时刻执行SQL作业，分析[23:45~23:55)期间的数据。
-h@h	先减1小时再向下取整到小时。 例如，创建Scheduled SQL作业时，配置**调度间隔**为**每天00:00**，**延迟执行**为**30秒**，**SQL时间窗口**为**[-h@h,-5m@m）**，则表示在00:00:30时刻执行SQL作业，分析[23:00~23:55)期间的数据。
-50m@h	先减50分钟再向下取整到小时。 例如，创建Scheduled SQL作业时，配置**调度间隔**为**每天00:00**，**延迟执行**为**30秒**，**SQL时间窗口**为**[-50m@h,-5m@m）**，则表示在00:00:30时刻执行SQL作业，分析[23:00~23:55)期间的数据。
-12h+5m	先减12小时再加5分钟，即减11小时55分钟。 例如，创建Scheduled SQL作业时，配置**调度间隔**为**每天00:00**，**延迟执行**为**30秒**，**SQL时间窗口**为**[-12h+5m,-5m）**，则表示在00:00:30时刻执行SQL作业，分析[12:05~23:55)期间的数据。

13.12. 时区列表

本文介绍Scheduled SQL功能所涉及的时区格式以及常见的时区列表。

时区格式设置

Scheduled SQL功能中的时区格式为`{±偏移小时数}{分钟数}`。

- 偏移小时数的取值范围：[-12,+14]。
- 分钟数的取值范围：00、30、45。
- 整体取值范围：[-1200,+1400]

> **说明** 小时和分钟都必须为两位数字。

您可以在创建Scheduled SQL时，配置**调度间隔**为**Cron**，并选择时区。

常见时区列表

时区格式	全称	中文名称
+0800	China Standard Time、Hong Kong Time	中国标准时间、中国香港时间
	Australian Western Time	澳大利亚西部时间
	Korea Standard Time	韩国标准时间
	Malaysia Time	马来西亚时间
	Philippile Time	菲律宾时间
	Singapore Time	新加坡时间
	Central Indonesian Time	印度尼西亚中部时间
	Ulaanbaatar Time	乌兰巴托时间
	Choibalsan Time	乔巴山时间
+0900	Japan Standard Time	日本标准时间
	Ulaanbaatar Summer Time	乌兰巴托夏令时
+0930	Australian Central Standard Time	澳大利亚中部标准时间
+1200	New Zealand Standard Time	新西兰标准时间
+0530	India Standard TIme	印度标准时间
-0000	Greenwich Mean Time	格林威治标准时间
	Western Europen Time	西欧时间
-0400	Atlantic Standard Time	大西洋标准时间（美国）
-0500	Eastern Standard Time	东部标准时间（美国）
-0600	Central Standard Time	中部标准时间（美国）

时区格式	全称	中文名称
-0700	Mountain Standard Time	山区标准时间（美国）
-0800	Pacific Standard Time	太平洋标准时间（美国）

13.13. 常见问题

本文介绍Scheduled SQL功能的常见问题。

如何保证SQL分析的数据准确性？

数据延迟写入或实例的调度配置不恰当时，可能发生数据分析不准确问题。

- 数据写入存在延迟。例如数据写入日志服务延迟了5分钟，实例执行时间为12:03:00，SQL时间窗口为相对一分钟，即[12:02:00，12:03:00)，则查询不到最新的数据。
- 数据写入日志服务到能够被查询到，这期间存在延时（一般少于3秒）。即使延时很低，也存在数据漏查的风险。例如实例执行时间为12:03:30，SQL时间窗口为相对一分钟，即[12:02:30，12:03:30)，对于12:03:29秒写入的日志，不能保证在12:03:30肯定能够查询到。
- 同一分钟内包含不同时间的日志时，由于日志索引的构建，时间较迟的日志的索引可能落盘到较早的时间点。例如实例执行时间为12:03:30，SQL时间窗口为相对一分钟，即[12:02:30，12:03:30)。如果在12:02:50秒写入日志，这些日志时间为12:02:20、12:02:50等，那么这些日志的索引可能落盘在12:02:20这个时间点，导致在[12:02:30，12:03:30)期间查询不到日志。

使用Scheduled SQL时，建议根据业务情况，同时兼顾数据实时性和准确性。

- 考虑数据上传日志服务存在延迟情况，您可以结合数据采集延迟以及业务能够容忍的最大结果可见延迟，设置**执行延迟**和**SQL时间窗口**（结束时间往前一点），避免实例执行时SQL时间窗口内的数据未全部到达。
- 建议SQL时间窗口按分钟对齐（例如整分钟、整小时），以保证上传局部乱序数据时的数据准确度。

如何防止SQL分析执行失败？

- 确保输入正确的SQL语法。
- 确保已为待分析的字段创建正确的索引。例如查询和分析语句为 `* | select uid` ，则您需要为uid字段开启统计功能。更多信息，请参见配置索引。
- 确保已配置正确的权限。例如执行SQL分析的权限，读取日志库数据的权限。
- 确保已提供足够的计算资源。资源限制说明请参见查询与分析的使用限制。
- 避免使用逻辑过于复杂的SQL语句以及SQL时间窗口设置过大，否则将导致计算超时（服务端超时时间为10分钟）。

通过Scheduled SQL作业写入数据到目标库时，是否进行索引检查？

日志服务将Scheduled SQL的计算结果写入目标库时，不做索引检查。如果目标库未配置索引，则您无法进行SQL分析，但不影响日志消费和查询操作。

建议您在创建Scheduled SQL作业前，完成目标库的索引配置。如果未提前配置索引，可通过重建索引功能为历史数据配置索引。具体操作，请参见重建索引。

某个实例执行超时，是否会影响后续的执行计划？

某个实例执行超时不会影响后续的执行计划的生成。新实例的调度时间是延续上一个实例的调度时间，但是后续实例的创建时间和执行时间会延迟，并且需要花费时间来追赶原定的执行计划，逐步缩减结果数据的延迟。

一般来说，追赶固定的数据量时，调度周期越大，追赶速度越快。例如：

- 追赶一天的数据量，调度周期为1分钟时，需要执行1440个实例，每个实例运行20秒。
- 追赶一天的数据量，调度周期为1小时时，需要执行24个实例，每个实例运行2分钟。

  日志服务提供服务分布式查询和分析能力，在面对更多数据时将使用更多的计算资源。

写入到目标库的数据如何溯源？

通过Scheduled SQL写入到目标库的数据，默认被添加如下__tag__字段，用于数据溯源。

- __tag__:__instance_id__:2b06486746f0cb38-5bffe67493825-1e2606a：作业实例的ID
- __tag__:__job__:from_now：作业名称
- __tag__:__project__:ali-sls-etl-staging：作业所属于的Project
- __tag__:__schedule_time__:1618474200：作业的调度时间，Unix时间戳，单位：秒。
- __tag__:__trigger_time__:1618474259：作业的执行时间，Unix时间戳，单位：秒。

14.代码诊断

日志服务与云效代码管理Codeup联合推出代码诊断功能，帮助您一键定位并跳转到对应代码的位置，快速排查和修复代码问题。

前提条件

- 日志服务
  - 已采集日志到Logstore。具体操作，请参见数据采集。
  - 已配置索引。具体操作，请参见配置索引。
- 云效代码管理Codeup

  已创建代码库并提交代码。具体操作，请参见极速上手指引。

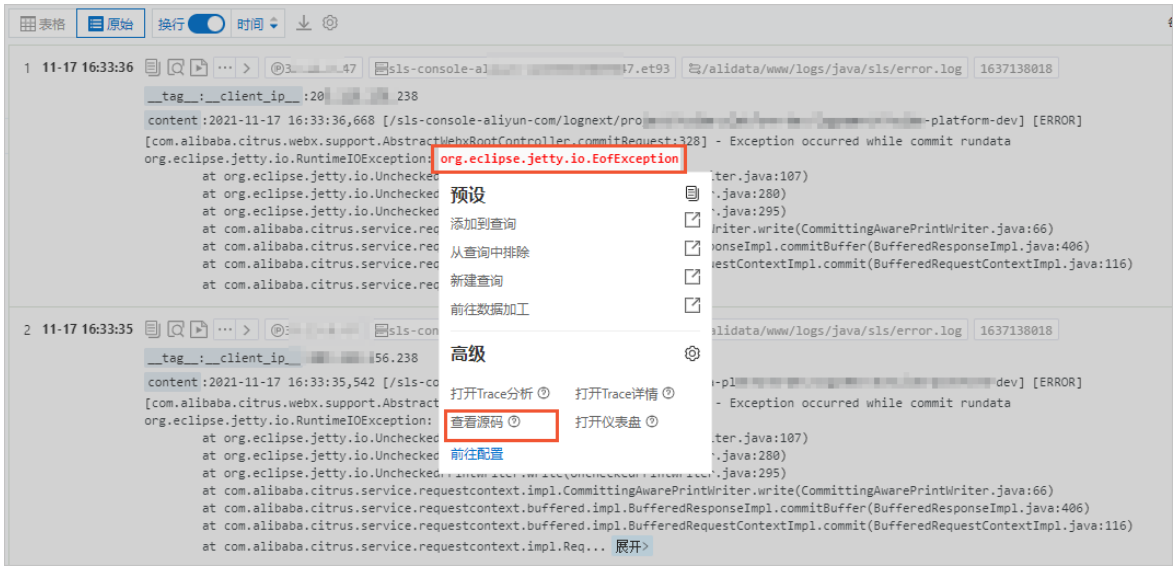
背景信息

云效代码管理Codeup是阿里云出品的一款企业级代码管理平台，提供代码托管、代码评审、代码扫描、质量检测等功能，全方位保护企业代码资产，帮助企业实现安全、稳定、高效的研发管理。日志服务与云效代码管理Codeup联合推出代码诊断功能。您可以将代码上传至云效代码管理Codeup，将日志上传至日志服务，代码诊断功能将基于日志中打印的代码位置信息，快速定位并跳转至代码的位置，帮助您排查和修复代码问题。

操作步骤

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. （可选）在输入框中输入查询语句，选择时间范围，然后单击**查询/分析**。

   您还可以通过Data Explorer构建查询和分析语句。具体操作，请参见通过Data Explorer构建查询和分析语句。关于查询语法的更多信息，请参见查询语法。
5. 在**原始日志**页签中，单击目标字符串，然后单击**查看源码**。



6. 在**高级事件配置**对话框中，设置交互事件。
   i. 在**字段列表**区域，选择目标字段，然后单击**添加事件**。

      支持为多个字段分别设置交互事件。

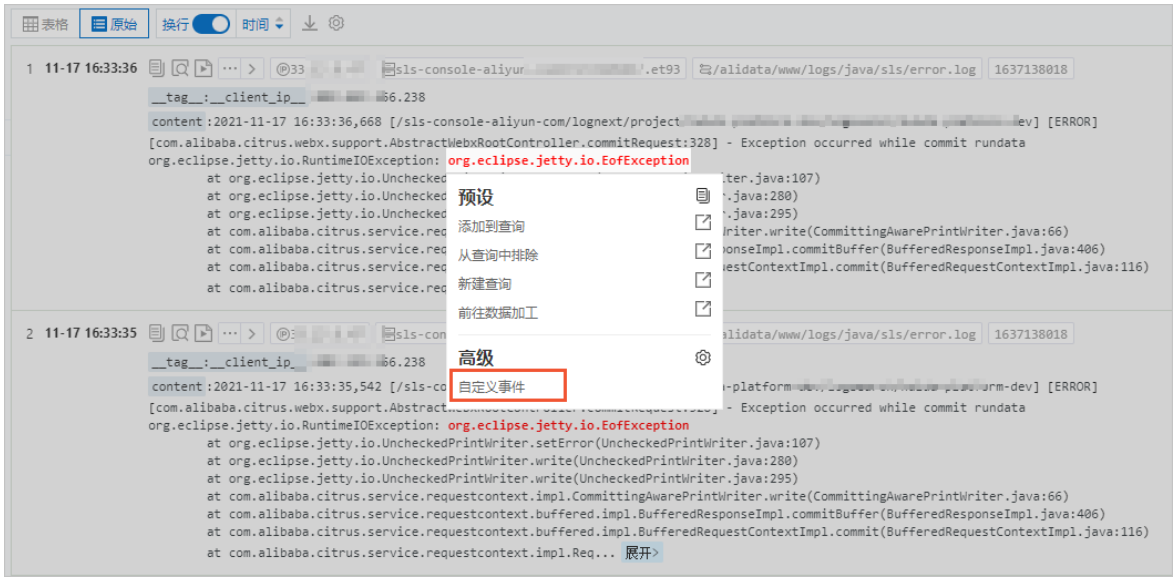
ii. 在**事件配置**区域，添加交互事件。

> **说明** 当您使用的RAM用户无代码库访问权限时，请选中**为无库访问权限的子账号默认添加代码库浏览者权限**复选框，否则将无法查看代码。
>
> - 勾选该选项后，系统自动将对应的RAM用户加入到企业成员中，并赋予当前代码库浏览者权限。浏览者仅允许查看代码库，不允许写入代码。
> - 在此处为RAM用户添加权限是一次性操作。如果需要变更权限，请前往目标代码库中进行修改。更多信息，请参见权限。

参数	说明
配置名称	设置事件的名称。
事件行为	设置事件行为。此处选择**查看源码**。
企业	选择您已创建的企业。 云效代码管理Codeup是企业级别的代码管理平台，企业级的数据隔离。更多信息，请参见我的企业。
代码路径	选择您已创建的代码库。 云效代码管理Codeup提供代码库一键导入功能，当您的代码存储在第三方代码库（例如GitHub）中时，您可以先将第三方代码库导入到云效代码管理Codeup中，然后查看源码。相关配置如下： - **源代码库地址**：第三方代码库的地址。 - **三方账号**：第三方代码库所在账号。 - **Access Token**：第三方代码库所在账号的Access Token。如何获取，请参见Access Token添加说明。 您所提供的Access Token仅用于导入代码库，阿里云不会进行记录及另做其他用途。

iii. 单击**确认**。

设置完成后，单击对应字段的值，然后单击您所添加的交互事件，即可跳转到目标代码库中。



15.通过JDBC协议分析日志

日志服务支持您在数据库（例如MySQL）中使用JDBC协议连接日志服务，并通过标准的SQL92语法查询和分析日志。

前提条件

- 已创建阿里云账号的AccessKey或RAM用户的AccessKey。具体操作，请参见访问密钥。

  如果使用RAM用户的AccessKey，则RAM用户必须是当前Project所属阿里云账号的RAM用户，且具备Project级别的读权限。
- 已创建Logstore。具体操作，请参见创建Logstore。

版本说明

目前日志服务仅支持JDBC 5.1.49版本。

连接参数

此处以MySQL数据库为例，介绍连接参数。

> **说明** MySQL JDBC不支持分页。

- 连接语法的格式

```
mysql -hhost -user -password -port
use database;
```

- 示例

```
mysql -hmy-project.cn-hangzhou-intranet.log.aliyuncs.com -ubq****mo86kq -p4f****uZP -P10005
use my-project;
```

- 参数说明

参数	说明
host	日志服务访问域名，需添加Project名称，例如my-project.cn-hangzhou-intranet.log.aliyuncs.com。 目前仅支持经典网络和VPC网络的访问域名。更多信息，请参见经典网络及VPC网络服务入口。
port	默认使用10005。
user	阿里云账号的AccessKey ID。如何获取，请参见访问密钥。
password	阿里云账号的AccessKey Secret。
database	日志服务Project名称。 **说明** 一个数据库同时只支持连接一个Project。
table	日志服务Logstore名称。 **说明** Logstore需在查询和分析语句中指定。

查询和分析语法

- 过滤语法

  > **说明** 在where条件中必须包含__date__字段或__time__字段来限制查询的时间范围。__date__字段是timestamp类型，__time__字段是bigint类型。例如：
  > - __date__ > '2017-08-07 00:00:00' and __date__ < '2017-08-08 00:00:00'
  > - __time__ > 1502691923 and __time__ < 1502692923

  where过滤语法说明如下表所示。

语义	示例	说明
字符串搜索	key = "value"	查询的是分词之后的结果。
字符串模糊搜索	◦ key has 'valu*' ◦ key like 'value_%'	查询的是分词之后模糊匹配的结果。
数值比较	num_field > 1	比较运算符包括>、>=、=、<和<=。
逻辑运算	and or not	例如a = "x" and b ="y"或a = "x" and not b ="y"。
全文搜索	__line__ ="abc"	如果使用全文索引搜索，需使用特殊的key（__line__）。

- 计算语法

  支持计算操作符。更多信息，请参见分析语法。

- SQL92语法

  过滤语法和计算语法组合成为SQL92语法。

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

  您可以将上述查询和分析语句中的分析语句与时间条件组合成为查询条件，变成标准SQL92语法，如下所示：

```
select avg(latency),max(latency) ,count(1) as c from sample-logstore where status>200 and __time__
>=1500975424 and __time__ < 1501035044 GROUP BY  method  ORDER BY c DESC  LIMIT 20
```

通过JDBC协议访问日志服务

- 程序调用

  您可以在任何一个支持MySQL connector的程序中使用MySQL语法连接日志服务。例如JDBC、Python MySQLdb，示例如下：

```
import com.mysql.jdbc.*;
import java.sql.*;
import java.sql.Connection;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
public class testjdbc {
    public static void main(String args[]){
        Connection conn = null;
        Statement stmt = null;
        try {
            //STEP 2: Register JDBC driver
            Class.forName("com.mysql.jdbc.Driver");
            //STEP 3: Open a connection
```

```
            //STEP 3: Open a connection
            System.out.println("Connecting to a selected database...");
            conn = DriverManager.getConnection("jdbc:mysql://projectname.cn-hangzhou-intranet.log.
aliyuncs.com:10005/sample-project","accessid","accesskey");
            System.out.println("Connected database successfully...");
            //STEP 4: Execute a query
            System.out.println("Creating statement...");
            stmt = conn.createStatement();
            String sql = "SELECT method,min(latency,10)  as c,max(latency,10) from sample-logstore
where  __time__>=1500975424 and __time__ < 1501035044 and latency > 0  and latency < 6142629 and
not (method='Postlogstorelogs' or method='GetLogtailConfig') group by method " ;
            String sql_example2 = "select count(1) ,max(latency),avg(latency), histogram(method),h
istogram(source),histogram(status),histogram(clientip),histogram(__source__) from  test10 where __
date__  >'2017-07-20 00:00:00'  and  __date__ <'2017-08-02 00:00:00' and __line__='abc#def' and la
tency < 100000 and (method = 'getlogstorelogS' or method='Get**' and method <> 'GetCursorOrData' )
";
            String sql_example3 = "select count(1) from  sample-logstore where     __date__  >
'2017-08-07 00:00:00' and  __date__ <     '2017-08-08 00:00:00' limit 100";
            ResultSet rs = stmt.executeQuery(sql);
            //STEP 5: Extract data from result set
            while(rs.next()){
                //Retrieve by column name
                ResultSetMetaData data = rs.getMetaData();
                System.out.println(data.getColumnCount());
                for(int i = 0;i < data.getColumnCount();++i) {
                    String name = data.getColumnName(i+1);
                    System.out.print(name+":");
                    System.out.print(rs.getObject(name));
                }
                System.out.println();
            }
            rs.close();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

- 工具类调用

在经典网络或VPC网络中通过MySQL客户端进行连接。

```
[root@iZbp14putxkqvmal310ianZ:~# mysql -h cn-hangzhou-intranet.log.aliyuncs.com
-uLTAIvCkVBXkGhk0f -plvEss0WJNyPh7mD6yuC4SgNC7T0wxf -P10005 trip-demo
mysql: [Warning] Using a password on the command line interface can be insecure
.
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5958635
Server version: 5. 5.1.40-community-log

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> select count(1) from ebike where __date__ >'2017-10-11 00:00:00' and __d
ate__ < '2017-10-12 00:00:00';
+--------+
| _col0  |
+--------+
| 316632 |
+--------+
1 row in set (0.25 sec)

mysql>
```

1
2

- ①处配置为您的Project。
- ②处配置为您的Logstore。

16.SQL案例中心

日志服务SQL案例中心提供精选SQL案例，帮助您了解不同场景所需的SQL语句及熟悉查询分析操作。本文介绍SQL案例中心的使用方法。

操作步骤

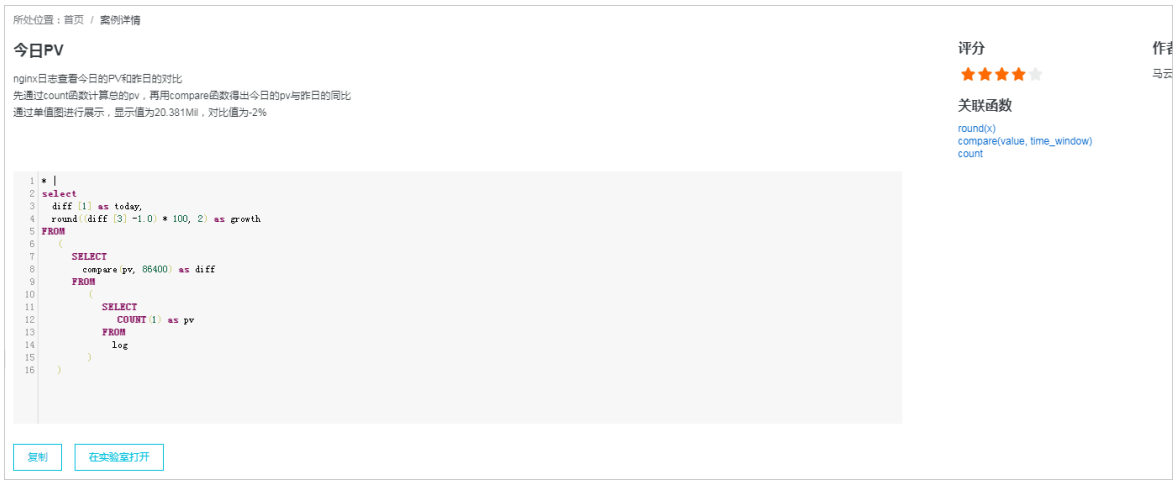
1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在左侧导航栏中，选择**其他 > SQL案例中心**。
4. 在SQL案例中心页面，查找您所需的案例。

   支持如下查询方式：

   - 在查询框中输入关键字，筛选相关案例。
   - 通过场景分类、函数分类，筛选相关案例。
   - 单击案例列表中的作者或场景分类，筛选相关案例。

5. 单击案例名称或关联函数，查看SQL案例详情。

   您可以在**案例详情**页面查看案例描述，及该案例所涉及的SQL语句、关联函数及查询结果。



日志服务数据实验室为您提供各个场景的模拟数据，便于您模拟查询分析操作。您可以直接单击**在实验室打开**，进行查询分析操作，数据实验室功能介绍请参见使用数据实验室。

17.分析进阶

17.1. 优秀分析案例

本文为您提供日志数据分析的一些案例。

案例列表

5分钟错误率超过40%时触发报警

统计每分钟的500错误率，当最近5分钟错误率超过40%时触发报警。

```
status:500 | select __topic__, max_by(error_count,window_time)/1.0/sum(error_count) as error_ratio, s
um(error_count) as total_error  from (
select __topic__, count(*) as error_count , __time__ - __time__ % 300  as window_time  from log group
by __topic__, window_time
)
group by __topic__  having  max_by(error_count,window_time)/1.0/sum(error_count)   > 0.4  and sum(err
or_count) > 500 order by total_error desc limit 100
```

统计流量并设置告警

统计每分钟的流量，当最近的流量出现暴跌时，触发报警。 由于在最近的一分钟内，统计的数据不是一个完整分钟的，所以需要除以 `greatest(max(__time__) - min(__time__),1)` 进行归一化，统计每个分钟内的流量均值。

```
* | SELECT SUM(inflow) / greatest(max(__time__) - min(__time__),1) as inflow_per_minute, date_trunc('
minute',__time__)  as minute group by minute
```

计算不同数据区间的平均延时

按照数据区间分桶，在每个桶内计算平均延时。

```
* | select avg(latency) as latency , case when originSize < 5000 then 's1' when originSize < 20000 th
en 's2' when originSize < 500000 then 's3' when originSize < 100000000 then 's4' else 's5' end as os
group by os
```

返回不同结果的百分比

返回不同部门的count结果，及其所占百分比。该query结合了子查询、窗口函数。其中 `sum(c) over()` 表示计算所有行的和。

```
* |  select   department, c*1.0/ sum(c) over ()  from(select  count(1) as c, department   from log gr
oup by department)
```

统计满足条件的个数

在URL路径中，我们需要根据URL不同的特征来计数，这种情况可以使用CASE WHEN语法，但还有个更简单的语法是count_if。

```
* | select  count_if(uri like '%login') as login_num, count_if(uri like '%register') as register_num, date_format(date_trunc('minute', __time__), '%m-%d %H:%i') as time  group by time order by time limit 100
```

17.2. 优化查询

本文介绍优化查询的方法，用于提高查询效率。

增加Shard数量

Shard表示计算资源，Shard越多，计算越快，您需要保证平均每个Shard扫描的数据不多于5000万条。您可以通过分裂Shard，增加Shard数量。具体操作，请参见分裂Shard。

> 注意 分裂Shard会产生更多费用，且只对新数据起到加速效果，旧数据仍然在旧Shard中。

缩减查询的时间范围和数据量

- 时间范围越大，查询越慢。

  适当缩短查询的时间范围可以更快地完成计算。

- 数据量越大，查询越慢。

  请尽量减少查询的数据量。

多次重复查询

当查询不精确时，可以尝试多次重复查询。每次查询时，底层加速机制会充分利用已有的结果进行分析，因此多次查询可以使结果更加精确。

优化SQL分析语句

计算时间较长的查询分析语句具备如下特点。

- 使用GROUP BY语法基于字符串列进行分组统计。
- 使用GROUP BY语法基于多列（大于5列）进行分组统计。
- 在SQL分析语句中有生成字符串的操作。

您可以通过如下方法优化分析语句。

- 尽量避免生成字符串的操作。

  例如使用date_format函数生成格式化的时间戳，导致查询效率低。针对时间戳的计算，建议使用date_trunc或者time_series函数进行计算。示例如下：

  ```
  * | select date_format(from_unixtime(__time__) , '%H_%i') as t, count(1) group by t
  ```

- 尽量避免对字符串列进行分组统计。

  使用GROUP BY语法基于字符串列进行分组统计，会导致大量的Hash计算，这部分计算量占据整体计算量的50%以上。示例如下：

  - 查询和分析语句（速度快）

    ```
    * | select count(1) as pv , from_unixtime(__time__-__time__%3600) as time group by __time__-__time__%3600
    ```

  - 查询和分析语句（速度慢）

    ```
    * | select count(1) as pv , date_trunc('hour',__time__) as time group by time
    ```

上述两条查询分析语句都是计算每小时的日志条数。第二条语句先把时间戳转化成字符串格式（例如2021-12-12 00:00:00），然后对这个字符串列进行分组统计。第一条语句对时间整点值进行计算，并且通过分组统计后再将时间戳转化为字符串格式。

- 基于多列进行分组统计时，把字典大的字段放在前面。

  例如字段的值有13个，uid字段的值有1亿个，则建议在GROUP BY子句中将uid字段放在前面。示例如下：

  - 查询和分析语句（速度快）

    ```
    * | select province,uid,count(1) group by uid,province
    ```

  - 查询和分析语句（速度慢）

    ```
    * | select province,uid,count(1) group by province,uid
    ```

- 使用估算函数。

  估算函数的性能比精算函数的好。估算会损失一定的精确度，用于达到快速计算的效果。示例如下：

  - 查询和分析语句（速度快）

    ```
    * |select approx_distinct(ip)
    ```

  - 查询和分析语句（速度慢）

    ```
    * | select count(distinct(ip))
    ```

- 在SQL分析语句中指定获取需要的列，尽量不要读取所有列。

  在SQL分析语句中，尽量只读取需要参与计算的列。如果要获取所有列，请使用查询语法。示例如下：

  - 查询和分析语句（速度快）

    ```
    * |select a,b c
    ```

  - 查询和分析语句（速度慢）

    ```
    * |select *
    ```

- 不是用于分组的列，尽量放在聚合函数中。

  例如userid与用户名必定是一一对应的，您只需使用GROUP BY语法对userid进行分组统计即可。示例如下：

  - 查询和分析语句（速度快）

    ```
    * | select userid, arbitrary(username), count(1) group by userid
    ```

  - 查询和分析语句（速度慢）

    ```
    * | select userid, username, count(1) group by userid,username
    ```

- 尽量避免使用IN语法

  尽量避免在分析语句中使用IN语法，您可以在查询语句中使用OR语法代替。示例如下：

  - 查询和分析语句（速度快）

    ```
    key: a or key: b or key: c | select count(1)
    ```

  - 查询和分析语句（速度慢）

    ```
    * | select count(1) where key in ('a','b')
    ```

17.3. 时间字段转换示例

在查询分析中，往往需要对日志中的时间字段进行处理，例如将时间戳转换成指定格式等，本文档介绍时间字段的常用转换示例。

日志中可能有多个记录时间的字段，例如：

- `__time__`：用API/SDK写入日志数据时指定的日志时间，该字段可用于日志投递、查询、分析。
- 日志中原有的时间字段：日志在生成时，用于记录日志事件发生时间的字段，是原始日志的字段。

时间字段的格式可能不统一、或不便于查看和阅读，可以在查询分析中将其转换为指定格式。例如：

1. 把__time__转化成时间戳
2. 把__time__以固定格式打印
3. 把timestamp转化成指定格式

把__time__转化成时间戳

把字段`__time__`转化成时间戳格式，建议使用from_unixtime函数。更多信息，请参见from_unixtime函数。

```
* | select from_unixtime(__time__)
```

把__time__以固定格式打印

把字段`__time__`以`年-月-日 时:分:秒`的形式打印下来，建议使用date_format函数。更多信息，请参见date_format函数。

```
* | select date_format(__time__, '%Y-%m-%d %H:%i:%S')
```

把日志中的时间转换成指定格式

把日志中的时间字段转化成指定格式（`年-月-日 时:分:秒`），只取`年-月-日`部分，并做Group by处理，建议使用date_format函数。更多信息，请参见date_format函数。

- 日志样例：

```
__topic__:
body_byte_sent:  307
hostname:  example.com
http_user_agent:  Mozilla/5.0 (iPhone; CPU iPhone OS 10_3_3 like Mac OS X) AppleWebKit/603.3.8 (KH
TML, like Gecko) Mobile/14G60 QQ/192.0.2.1 V1_IPH_SQ_7.1.8_1_APP_A Pixel/750 Core/UIWebView NetTyp
e/WIFI QBWebViewType/1
method:  GET
referer:  www.example.com
remote_addr:  192.0.2.0
request_length:  111
request_time:  2.705
status:  200
upstream_response_time:  0.225582883754
url:  /?k0=v9&
time:2017-05-17 09:45:00
```

- SQL语句样例：

```
* | select date_format (date_parse(time,'%Y-%m-%d %H:%i:%S'), '%Y-%m-%d') as day, count(1) as uv g
roup by day order by day asc
```

18.关联外部数据源

18.1. 简介

日志服务提供外部存储功能，可用于日志服务与MySQL数据库、阿里云对象存储OSS、托管的CSV文件进行关联。本文介绍日志服务外部存储功能的应用场景、功能优势等信息。

应用场景

在日志分析场景中，您可能经常遇到数据分散存储的问题，例如用户操作、行为等相关数据存储在日志服务中，用户属性、注册信息、资金、道具等相关数据存储在数据库中。类似场景下，您需要对用户进行分层统计，将最后的计算结果写入到数据库提供的报表系统中。

针对上述场景，传统做法是将数据迁移到统一的存储系统中，再进行分析。在迁移过程中既涉及网络传输，又涉及数据的清洗和格式化，耗时又耗精力。日志服务提供的外部存储API支持以下功能：

- 通过API为外部存储定义映射，不需要迁移数据。
- 提供统一的查询分析引擎，支持通过JOIN语法对日志、外部存储等多种数据源进行联合查询。
- 支持将多种分析结果保存到外部存储中。

功能优势

- 节省成本
  - 节省数据迁移成本。不同存储系统的格式和API都不同，在迁移过程中涉及到复杂的数据转换。使用日志服务外部存储无需搬迁数据。
  - 节省数据维护成本。如果采用迁移数据方式，如果有数据更新，需及时维护。
- 方便快捷
  - 通过SQL语句分析数据，实现秒级别获得分析结果。
  - 将常用视图添加到仪表盘，打开仪表盘页面即可快速查看相关信息。

支持的外部存储

外部存储功能支持日志服务与MySQL数据库、阿里云OSS、托管的CSV文件建立关联，详细信息如下表所示。

外部存储名称	从外部数据源读取	写入外部数据源	创建方式	地域
MySQL	支持	支持	API、SDK、CLI	所有地域
OSS	支持	支持	SQL create table	所有地域
托管的CSV文件	支持	不支持	SDK	华东2（上海）和俄罗斯西部1（莫斯科）

18.2. 关联MySQL数据源

本文介绍如何创建外部存储，建立日志服务与MySQL数据库的关联。

前提条件

- 已采集数据到日志服务。具体操作，请参见数据采集。
- 已存储数据到MySQL数据库。

背景信息

日志服务外部存储功能支持日志服务与阿里云RDS MySQL数据库、在阿里云ECS上自建的MySQL数据库关联，您还可以

将查询分析结果写入MySQL数据库中，便于进一步处理结果。创建外部MySQL存储的最佳实践，请参见关联Logstore与MySQL数据库进行查询分析。

操作步骤

1. 设置白名单。
   - 如果是RDS MySQL数据库，需添加白名单地址100.104.0.0/16、11.194.0.0/16和11.201.0.0/16。更多信息，请参见设置IP白名单。
   - 如果是专有网络下ECS上自建的MySQL数据库且ECS上设置了安全组，需设置安全组规则，允许100.104.0.0/16、11.194.0.0/16和11.201.0.0/16网段访问。具体操作，请参见添加安全组规则。
2. 创建ExternalStore。
   i. 安装日志服务CLI。更多信息，请参见CLI概述。
   ii. 创建配置文件*/root/config.json*。
   iii. 在*/root/config.json*文件中添加如下脚本，并根据实际情况替换参数配置。

```
{
"externalStoreName":"storename",
"storeType":"rds-vpc",
"parameter":
   {
   "region":"cn-qingdao",
   "vpc-id":"vpc-m5eq4irc1pucp*******",
   "instance-id":"i-m5eeo2whsn*******",
   "host":"localhost",
   "port":"3306",
   "username":"root",
   "password":"****",
   "db":"scmc",
   "table":"join_meta"
   }
}
```

参数	说明
externalStoreName	ExternalStore名称，必须小写。
storeType	数据源类型，固定为rds-vpc。
region	地域。详细说明如下： ■ 如果是RDS MySQL数据库，则配置region为RDS实例所在地域。 ■ 如果是专有网络下ECS上自建的MySQL数据库，则配置region为ECS实例所在地域。 **说明** RDS实例或ECS实例必须与日志服务Project处于同一地域。
vpc-id	VPC ID。详细说明如下： ■ 如果是专有网络下的RDS MySQL数据库，则配置vpc-id为RDS实例所属专有网络的ID。 ■ 如果是专有网络下ECS上自建的MySQL数据库，则配置vpc-id为ECS实例所属专有网络的ID。

参数	说明
instance-id	实例ID。详细说明如下： ■ 如果是RDS MySQL数据库，则配置instance-id为RDS实例的VpcCloudInstanceId。 您需要通过调用DescribeDBInstanceAttribute接口获取VpcCloudInstanceId。具体操作，请参见获取RDS实例的VpcCloudInstanceId。 ■ 如果是专有网络下ECS上自建的MySQL数据库，则配置instance-id为ECS实例ID。
host	数据库地址。详细说明如下： ■ 如果是专有网络下的RDS MySQL数据库，则配置host为RDS实例的内网地址。 ■ 如果是专有网络下ECS上自建的MySQL数据库，则配置host为ECS的私网IP地址。
port	端口号。详细说明如下： ■ 如果是RDS MySQL数据库，则配置port为RDS实例的端口号。 ■ 如果是专有网络下ECS上自建的MySQL数据库，则配置port为ECS上MySQL的服务端口。
username	数据库用户名。
password	数据库密码。
db	数据库。
table	数据库表。

iv. 创建ExternalStore。

其中project_name为日志服务Project名称，请根据实际情况替换。

```
aliyunlog log create_external_store --project_name="log-rds-demo" --config="file:///root/config.json"
```

相关操作

- 更新MySQL外部存储。

```
aliyunlog log update_external_store --project_name="log-rds-demo" --config="file:///root/config.json"
```

- 删除MySQL外部存储。

```
aliyunlog log delete_external_store --project_name="log-rds-demo" --store_name=abc
```

- 获取RDS实例的VpcCloudInstanceId。
  i. 使用RDS实例所在的阿里云账号登录RDS控制台。
  ii. 打开DescribeDBInstanceAttribute调试页面。
  iii. 配置**DBInstanceId**，然后单击**发起调用**。
  iv. 在返回结果中，获取VpcCloudInstanceId。

后续步骤

Logstore和MySQL联合查询

18.3. 关联OSS数据源

本文介绍如何创建外部存储，建立日志服务与OSS的关联。

前提条件

- 已采集日志。更多信息，请参见数据采集。
- 已开启并配置索引。更多信息，请参见配置索引。
- 已创建OSS Bucket。更多信息，请参见创建存储空间。
- 已上传CSV格式文件到OSS Bucket。更多信息，请参见上传文件。

功能优势

与OSS进行关联查询分析，具有如下优势：

- 节省费用：将更新频率低的数据保存在OSS上，只需要支付少量的存储费用，并且可以通过内网读数据，免去流量费用。
- 降低运维工作：在轻量级的联合分析平台中，不需要搬迁数据到同一个存储系统中。
- 节省时间：使用SQL分析数据，分析结果秒级可见，并可以将常用的分析结果定义为报表，打开即可看到结果。

操作步骤

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 输入查询与分析语句，单击**查询/分析**。

   通过SQL定义虚拟外部表（此处以user_meta1为例），映射到OSS文件，如果执行结果中的**result**为**true**，表示执行成功。

   ```
   * | create table user_meta1 ( userid bigint, nick varchar, gender varchar, province varchar, gend
   er varchar,age bigint) with ( endpoint='oss-cn-hangzhou-internal.aliyuncs.com',accessid='********
   ******************',accesskey ='**************************',bucket='testoss*********',objects=A
   RRAY['user.csv'],type='oss')
   ```

   在查询分析语句中定义外部存储名称、表的Schema等信息，并通过WITH语法指定OSS访问信息及文件信息，详细信息如下表所示。

配置项	说明
外部存储名称	外部存储名称，即虚拟表的名称，例如user_meta1。
表的Schema	定义表的属性，包括表的列名及格式，例如(userid bigint, nick varchar, gender varchar, province varchar, gender varchar,age bigint)。
endpoint	OSS内网访问域名。更多信息，请参见访问域名和数据中心。
accessid	您的AccessKey ID。更多信息，请参见访问密钥。
accesskey	您的AccessKey Secret。更多信息，请参见访问密钥。

配置项	说明
bucket	CSV文件所在的OSS Bucket名称。
objects	CSV文件路径。 **说明** objects为array类型，可以包含多个OSS文件。
type	固定为oss，表示外部存储类型为OSS。

5. 验证是否已成功定义外部存储。

   执行如下语句，返回结果为您之前定义的表内容，则表示已定义外部存储成功。

```
* | select * from user_meta1
```

userid	nick	gender	province	age
1	用户A	male	上海	18
2	用户B	female	浙江	19
3	用户C	male	广东	18

6. 通过JOIN语法完成日志服务和OSS的联合查询。

   例如，执行如下查询分析语句关联日志服务中日志的ID和OSS文件中的userid，补全日志信息。其中，test_accesslog为Logstore名称，l为Logstore别名，user_meta1为您定义的外部存储，请根据实际情况替换。

```
* | select * from test_accesslog l join user_meta1 u on l.userid = u.userid
```

__line__	useragent	action	action_i...	blood	magic	money	network	payment	pos_x	pos_y	status	userid	__time__	__sourc...	__date__	__topic__	__sls_q...	userid	nick	gender	province	age
null	Mozilla/5.0 (Linux;	logout	item_3921	51	88	847	wifi	cash	835	794	200	1	1531798068		null	null	null	1	阳光男孩	male	上海	18

关联OSS数据源的最佳实践请参见关联Logstore与OSS外表进行查询和分析。

18.4. 关联托管的CSV数据源

日志服务支持通过SDK方式将本地CSV文件上传到日志服务进行托管，并建立Logstore与CSV文件的关联。本文介绍如何在日志服务Logstore中联合托管的CSV文件进行数据分析。

前提条件

- 已采集日志。更多信息，请参见数据采集。
- 已配置索引。更多信息，请参见配置索引。
- 已创建CSV文件。
- 已安装Python SDK。更多信息，请参见安装Python SDK。

  支持aliyun-log-python-sdk 0.7.3及以上版本，您可以通过**pip install aliyun-log-python-sdk -U**命令进行升级。

使用限制

- 仅支持关联一个CSV文件。
- 最大支持50 MB的CSV文件。CSV文件经SDK压缩后被上传至日志服务，压缩后的大小需小于9.9 MB。

数据样例

例如Logstore用于记录用户的登录操作，CSV文件用于记录用户的基本信息（性别、年龄等）。关联Logstore和CSV文件后，可用于分析与用户属性相关的指标。

- Logstore

```
userid:100001
action:login
__time__:1637737306
```

- CSV文件

userid	nick	gender	province	age
100001	User_A	male	Liaoning	24
100002	User_B	male	Beijing	23
100003	User_C	female	Zhejiang	22
100004	User_D	female	Jiangxi	21
100005	User_E	male	Guangxi	20

操作步骤

1. 通过Python SDK创建外部存储（ExternalStore）。

   关于Python SDK的更多信息，请参见Python SDK概述。

```
from aliyun.log import *
endpoint='cn-shanghai.log.aliyuncs.com'
accessKeyId='test-project'
accessKey='TAI****YDw'
project='lr****VM'
ext_logstore='user_meta'
csv_file='./user.csv'
client = LogClient(endpoint, accessKeyId, accessKey)
res = client.create_external_store(project,
    ExternalStoreCsvConfig(ext_logstore, csv_file,
        [
            {"name" : "userid", "type" : "bigint"},
            {"name" : "nick", "type" : "varchar"},
            {"name" : "gender", "type" : "varchar"},
            {"name" : "province", "type" : "varchar"},
            {"name" : "age", "type" : "bigint"}
        ]))
res.log_print()
```

参数	说明
endpoint	日志服务的域名。更多信息，请参见服务入口。
accessKeyId	阿里云访问密钥AccessKey ID。更多信息，请参见访问密钥。 警告 建议您使用RAM用户的AccessKey进行操作，有效降低AccessKey泄露的风险。

参数	说明
accessKey	阿里云访问密钥AccessKey Secret。更多信息，请参见访问密钥。
project	目标Logstore所在的Project。
ext_logstore	外部存储名称，即虚拟表的名称。命名规则如下： ○ 仅支持小写字母、数字、短划线（-）和下划线（_）。 ○ 必须以小写字母或数字开头和结尾。 ○ 名称长度为3~63个字符。
csv_file	本地CSV文件所在路径及文件名。
表的Schema	用于定义虚拟表的属性，包括表的列名及格式。例如下述脚本表示表的Schema，请根据实际情况替换。 `[ {"name" : "userid", "type" : "bigint"}, {"name" : "nick", "type" : "varchar"}, {"name" : "gender", "type" : "varchar"}, {"name" : "province", "type" : "varchar"}, {"name" : "age", "type" : "bigint"} ]`

2. 登录日志服务控制台。
3. 在Project列表区域，单击目标Project。
4. 在**日志存储 > 日志库**页签中，单击目标Logstore。
5. 执行如下语句，验证是否成功创建外部存储。

   其中 `user_meta` 为外部存储的名称，请根据实际情况替换。

   ```
   * | SELECT * FROM user_meta
   ```

   如果返回结果为CSV文件的内容，则表示创建外部存储成功。

userid	nick	gender	province	age
100001	User_A	male	Liaoning	24
100002	User_B	male	Beijing	23
100003	User_C	female	Zhejiang	22
100004	User_D	female	Jiangxi	21
100005	User_E	male	Guangxi	20

6. 执行如下语句，建立Logstore与CSV文件的联合查询。

   本案例中通过Logstore中的userid字段和CSV文件中的userid字段，建立联合查询。其中，website_log为Logstore名称，user_meta为您定义的外部存储，请根据实际情况替换。

   ```
   * | SELECT * FROM website_log JOIN user_meta ON website_log.userid = user_meta.userid
   ```

action	userid	__time__	userid	nick	gender	province	age
login	100003	1637738249	100003	User_C	female	Zhejiang	22
login	100004	1637738249	100004	User_D	female	Jiangxi	21
login	100005	1637738249	100005	User_E	male	Guangxi	20
login	100002	1637738249	100002	User_B	male	Beijing	23
login	100004	1637738249	100004	User_D	female	Jiangxi	21
login	100003	1637738249	100003	User_C	female	Zhejiang	22
login	100001	1637738249	100001	User_A	male	Liaoning	24

19.最佳实践

19.1. 查询和分析网站日志

本文以查询和分析网站日志为例，帮助您快速上手查询和分析操作。

前提条件

已采集到网站访问日志。更多信息，请参见使用完整正则模式采集日志。

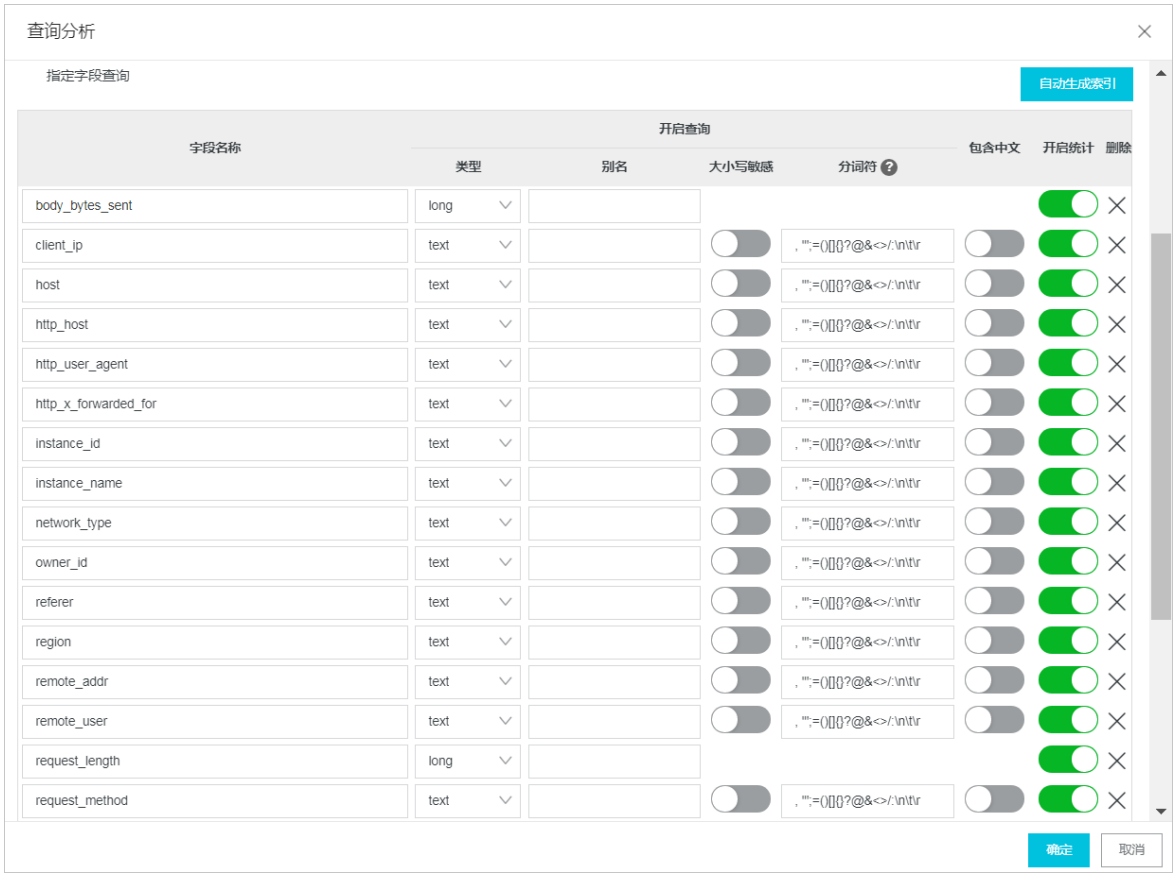
步骤1：配置索引

1.
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在Logstore的查询和分析页面，单击页面右上角的**查询分析属性 > 属性**。

   如果您还未开启索引，请单击页面右上角的**开启索引**。
5. 配置字段索引。

   您可以手动逐条配置索引，也可以单击**自动生成索引**，日志服务会根据预览数据中的第一条日志自动配置索引。

   > **说明**
   > - 配置索引后，只对新写入的数据生效。如果您要查询历史数据，请使用重建索引功能。具体操作，请参见重建索引。
   > - 如果您要使用分析功能，必须在配置索引时打开对应字段的统计功能。
   > - 日志服务默认已为部分保留字段开启索引。更多信息，请参见保留字段。



6. 单击**确定**。

步骤2：查询日志

您可以在Logstore的查询和分析页面，输入查询语句，选择时间范围，单击**查找/分析**，进行日志查询查找。

> **说明** 日志服务查询和分析语句格式为 `查询语句|分析语句` 。查询语句可单独使用，分析语句必须与查询语句一起使用。即分析功能是基于查询结果或全量数据进行的。

- 查询包含Chrome的日志。

```
Chrome
```

- 查询请求时间大于60秒的日志。

```
request_time > 60
```

- 查询请求时间在60秒~120秒之间的日志。

```
request_time in [60 120]
```

- 查询GET请求成功（状态码为200~299）的日志。

```
request_method : GET and status in [200 299]
```

- 查询request_uri字段值为/request/path-2/file-2的日志。

```
request_uri:/request/path-2/file-2
```

步骤3：分析日志

您可以在Logstore的查询和分析页面，输入查询和分析语句，选择时间范围，单击**查找/分析**，进行日志查询和分析操作。

> **说明** 执行查询和分析语句后，默认只返回100条结果，您可以使用LIMIT语句控制返回结果数量。更多信息，请参见LIMIT子句。

- 统计网站访问PV。

  使用COUNT函数统计网站访问PV。

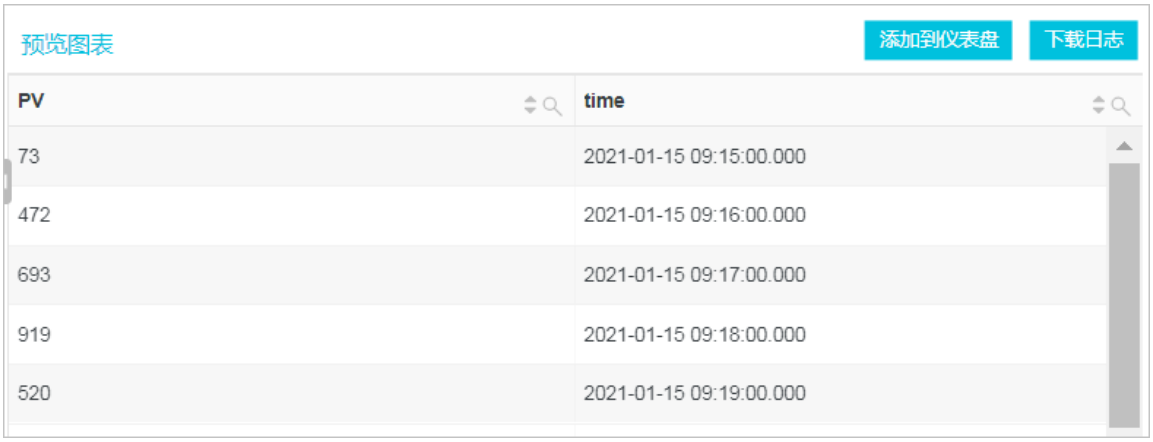
  ```
  * | SELECT COUNT(*) AS PV
  ```

- 根据每分钟的时间粒度，统计网站访问PV。

  使用date_trunc函数将时间对齐到每分钟并根据时间进行分组，然后使用COUNT函数计算每分钟的访问PV并根据时间排序。

  ```
  * | SELECT COUNT(*) as PV, date_trunc('minute', __time__) as time GROUP BY time ORDER BY time
  ```

- 根据每5分钟的时间粒度，统计每个请求方法的请求次数。

  使用__time__ - __time__ %300将时间对齐到5分钟并根据时间进行分组，然后使用COUNT函数计算每5分钟的请求次数并根据时间进行排序。

  ```
  * | SELECT request_method, COUNT(*) as count, __time__ - __time__ %300 as time GROUP BY time, requ
  est_method ORDER BY time
  ```

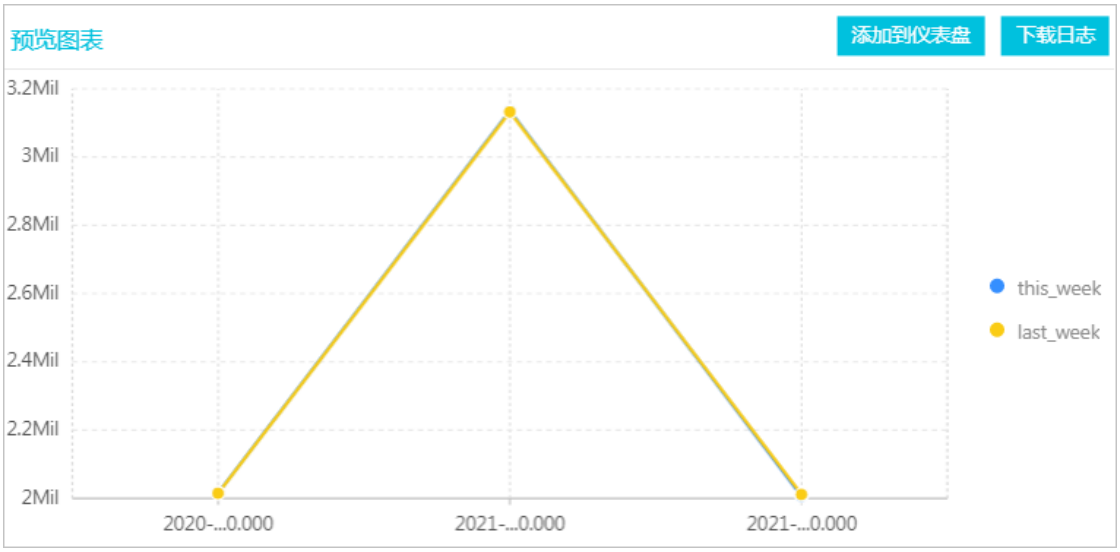
预览图表 添加到仪表盘 下载日志

request_method	count	time
PUT	242	1610673300
DELETE	101	1610673300
POST	231	1610673300
GET	778	1610673300
HEAD	4	1610673300

- 环比上周的网站访问PV。

  使用COUNT函数计算总PV数，再使用ts_compare函数得出本周与上周的环比。其中，website_log为Logstore名称。

```
* | SELECT diff[1] as this_week, diff[2] as last_week, time FROM (SELECT ts_compare(pv, 604800) as diff, time FROM (SELECT COUNT(*) as pv, date_trunc('week', __time__) as time FROM website_log GROUP BY time ORDER BY time) GROUP BY time)
```



- 统计客户端地址分布情况。

  使用ip_to_province函数获取IP地址对应的省份并根据省份进行分组，然后再使用COUNT函数计算每个地址出现的次数并根据次数进行排序。

```
* | SELECT COUNT(*) as count, ip_to_province(client_ip) as address GROUP BY address ORDER BY count DESC
```

预览图表 添加到仪表盘 下载日志

count	address
451	广东省
447	江苏省
433	北京市
425	山东省

- 统计访问前10的请求路径。

  根据请求路径进行分组，然后使用COUNT函数计算每个路径的访问次数并根据访问次数排序。

```
* | SELECT COUNT(*) as PV, request_uri as PATH GROUP BY PATH ORDER BY PV DESC LIMIT 10
```

PV	PATH
283	/request/path-0/file-9
277	/request/path-0/file-6
263	/request/path-3/file-1
262	/request/path-0/file-1

- 查询request_uri字段的值以%file-7结尾的日志。

  > **说明** 在查询语句中，模糊查询的通配符星号（*）和问号（?）只能出现在词的中间或末尾。如果您要查询以某字符结尾的字段，可以在分析语句中使用LIKE语法进行查询。

```
* | select * from website_log where request_uri like '%file-7'
```

  其中，website_log为Logstore名称。

预览图表 添加到仪表盘 下载日志

request_method	request_time	request_uri	scheme	server_protocol	slbid	status
POST	67	/request/path-2/file-7	https	HTTP/2.0	slb-01	200
GET	41	/request/path-0/file-7	https	HTTP/2.0	slb-02	305

- 统计请求路径访问情况。

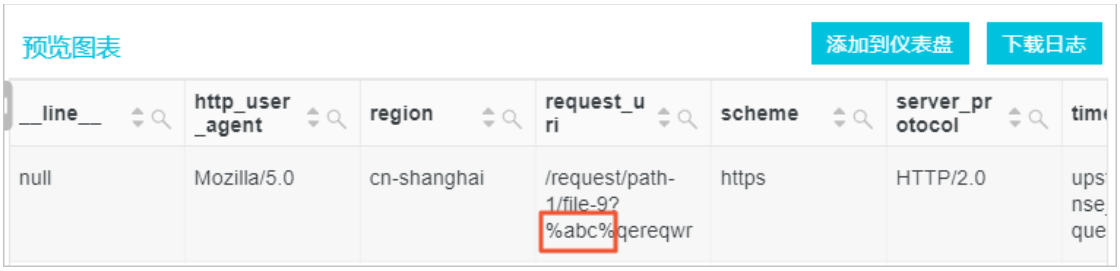
  使用regexp_extract函数提取request_uri字段中的文件部分，然后再使用COUNT函数计算各个请求路径的访问次数。

```
* | SELECT regexp_extract(request_uri, '.*\/(file.*)', 1) file, count(*) as count group by file
```

file	count
file-5	17127

- 查询request_uri字段中包含%abc%的日志。

```
* | SELECT * where request_uri like '%/%abc/%%' escape '/'
```


预览图表

添加到仪表盘 下载日志

__line__	http_user_agent	region	request_uri	scheme	server_protocol	time
null	Mozilla/5.0	cn-shanghai	/request/path-1/file-9?%abc%qereqwr	https	HTTP/2.0	ups nse que

参考信息：日志样例

```
__tag__:__client_ip__:192.0.2.0
__tag__:__receive_time__:1609985755
__source__:198.51.100.0
__topic__:website_access_log
body_bytes_sent:4512
client_ip:198.51.100.10
host:example.com
http_host:example.com
http_user_agent:Mozilla/5.0 (Macintosh; U; PPC Mac OS X 10_5_8; ja-jp) AppleWebKit/533.20.25 (KHTML,
like Gecko) Version/5.0.4 Safari/533.20.27
http_x_forwarded_for:198.51.100.1
instance_id:i-02
instance_name:instance-01
network_type:vlan
owner_id:%abc%-01
referer:example.com
region:cn-shanghai
remote_addr:203.0.113.0
remote_user:neb
request_length:4103
request_method:POST
request_time:69
request_uri:/request/path-1/file-0
scheme:https
server_protocol:HTTP/2.0
slbid:slb-02
status:200
time_local:07/Jan/2021:02:15:53
upstream_addr:203.0.113.10
upstream_response_time:43
upstream_status:200
user_agent:Mozilla/5.0 (X11; Linux i686) AppleWebKit/534.33 (KHTML, like Gecko) Ubuntu/9.10 Chromium/
13.0.752.0 Chrome/13.0.752.0 Safari/534.33
vip_addr:192.0.2.2
vpc_id:3db327b1****82df19818a72
```

19.2. 查询和分析JSON日志

本文以查询和分析JSON类型的网站日志为例，帮助您快速上手JSON日志的查询和分析操作。

前提条件

已采集到JSON日志。更多信息，请参见使用极简模式采集日志。

注意事项

在查询和分析JSON日志中的字段时，需注意以下事项：

- 查询和分析语句格式为 `查询语句|分析语句` 。在分析语句中，您必须使用双引号（""）包裹字段名称，使用单引号（''）包裹字符串。
- 您需为目标字段加上所有的父路径，格式为KEY1.KEY2.KEY3。例如concent.request.request_length。
- 日志服务支持查询和分析JSON对象中的叶子节点，但不支持查询和分析包含叶子节点的子节点。
- 日志服务不支持查询和分析值为JSON数组的字段，也不支持查询和分析JSON数组中的字段。

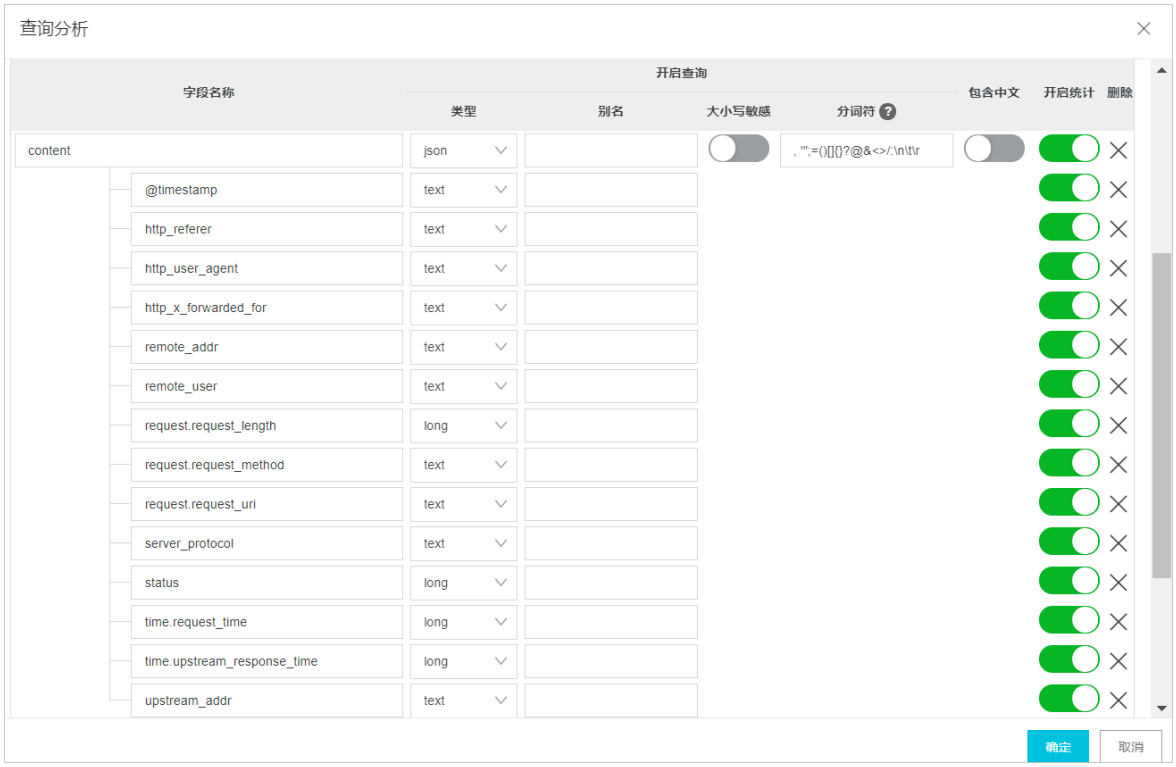
步骤1：配置索引

1.
2. 在**Project列表**区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在Logstore的查询和分析页面，单击页面右上角的**查询分析属性 > 属性**。

   如果您还未开启索引，请单击页面右上角的**开启索引**。
5. 配置字段索引。

   您可以手动逐条配置索引，也可以单击**自动生成索引**，日志服务会根据预览数据中的第一条日志自动配置索引。

   > **说明**
   > - 如果您要使用分析功能，必须在配置索引时打开对应字段的统计功能。更多信息，请参见配置索引。
   > - 日志服务默认已为部分保留字段开启索引。更多信息，请参见保留字段。
   > - 日志服务支持JSON对象中的叶子节点建立索引，但不支持包含叶子节点的子节点建立索引。例如您可以为request_time字段建立索引，但不能为time字段建立索引。
   > - 日志服务不支持值为JSON数组的字段建立索引，也不支持JSON数组中的字段建立索引。例如body_bytes_sent字段的值为JSON数组，不能建立索引。
   > - 为JSON对象中的字段配置索引时，需加父路径，格式为KEY1.KEY2。例如time.request_time。



6. 单击**确定**。

> **说明** 配置索引后，只对新采集的数据生效。如果您要查询历史数据，请使用重建索引功能。具体操作，请参见重建索引。

步骤2：查询日志

您可以在Logstore的查询和分析页面，输入查询语句，选择时间范围，单击**查找/分析**，进行日志查询操作。

- 查询请求状态为200的日志。

```
content.status:200
```

- 查询请求长度大于70的日志。

```
content.request.request_length > 70
```

- 查询GET请求的日志。

```
content.request.request_method:GET
```

步骤3：分析日志

您可以在Logstore的查询和分析页面，输入查询和分析语句，选择时间范围，单击**查找/分析**，进行日志分析操作。

- 统计不同请求状态对应的日志数量。

```
* | SELECT "content.status", COUNT(*) AS PV GROUP BY "content.status"
```

content.status	PV
200	45793
null	11137

- 计算不同请求时长对应的请求数量，并按照请求时长进行升序排序。

```
* | SELECT "content.time.request_time", COUNT(*) AS count GROUP BY "content.time.request_time" ORD
ER BY "content.time.request_time"
```

content.time.request_time	count
10.0	145
11.0	123
12.0	113

- 计算不同请求方法对应的平均请求时长。

```
* | SELECT avg("content.time.request_time") AS avg_time,"content.request.request_method"  GROUP BY
"content.request.request_method"
```

预览图表 添加到仪表盘 下载日志

avg_time	content.request.request_method
45	GET
11	PUT

参考信息：日志样例

JSON日志样例如下所示：

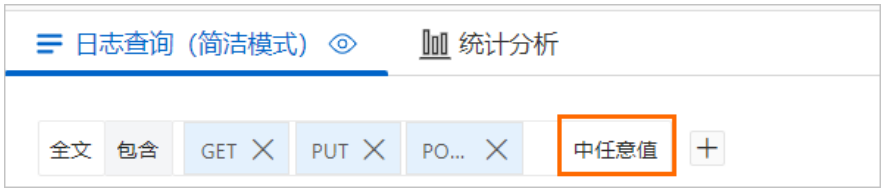
19.3. Data Explorer案例

日志服务提供Data Explorer功能，帮助您简单、快速地构建查询和分析语句。本文介绍通过Data Explorer构建查询和分析语句的案例。

查询示例

示例1

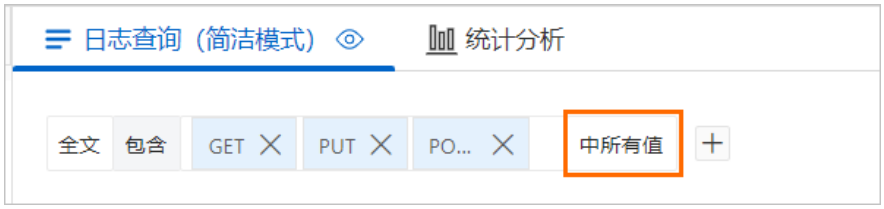
- 查询包含GET、PUT、POST中任意值的日志。
  - Data Explorer配置



  - 查询语句

```
(GET or PUT or POST)
```

- 查询同时包含GET、PUT、POST中的日志。
  - Data Explorer配置



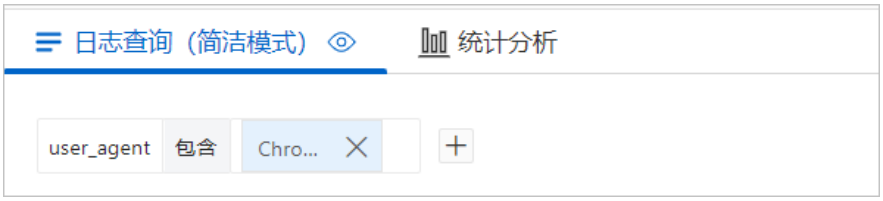
- 查询语句

```
(GET and PUT and POST)
```

示例2

查询user_agent字段的值中包含Chrome的日志。

- Data Explorer配置



- 查询语句

```
user_agent : Chrome
```

示例3

查询请求时间小于1秒的成功请求的日志中包含SLS字符串的日志。

- Data Explorer配置

- 查询语句

```
SLS and request_time < 1 and ( status >= 200 and status <= 299 )
```

分析示例

示例1

在网站访问日志中，统计过去一天内网站请求的每小时平均耗时以及随时间的变化情况。

- Data Explorer配置

- 查询和分析语句

```
|
select
  date_trunc('hour', __time__) as time,
  avg("request_time") as "_col0"
FROM  log
group by
  time
order by
  time
limit
  10000
```

- 查询分析结果

  执行查询分析操作后，您可以在**统计图表**中查看结果。此处通过线图（Pro版本）展示网站请求平均耗时随时间的变化情况。

示例2

在网站访问日志中，统计HTTP请求方法Top 5的分布情况。

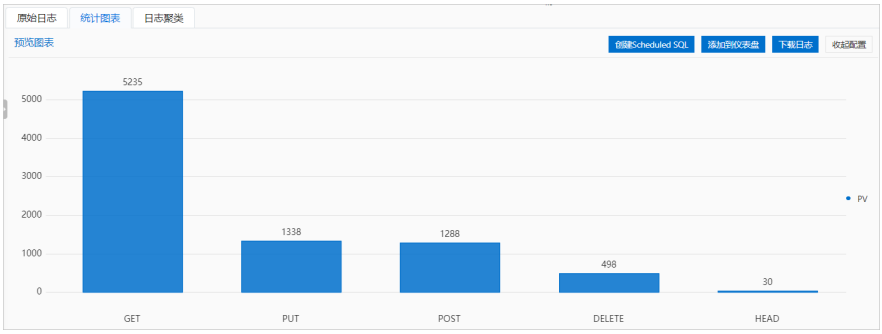
- Data Explorer配置

- 查询和分析语句

```
|
select
  "request_method",
  count,
  count * 1.0 / sum(count) over() as percent
FROM  (
    select
      count(1) as count,
      "request_method"
    FROM      log
    group by
      "request_method"
    order by
      count desc
    limit
      5
  )
```

- 查询分析结果

  执行查询分析操作后，您可以在**统计图表**中查看结果。此处通过柱状图（Pro版本）展示HTTP请求方法Top 5的分布情况。



示例3

在RDS数据库审计日志中，统计每个RDS实例的SQL平均执行延迟情况。

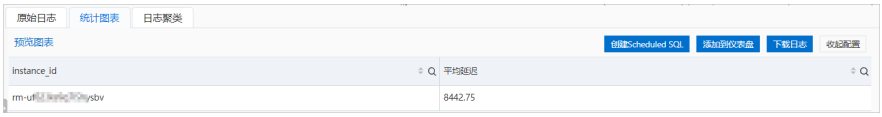
- Data Explorer配置

- 查询和分析语句

```
|
select
  "instance_id",
  avg("latency") as "平均延迟"
FROM  log
group by
  "instance_id"
order by
  "平均延迟" desc
```

- 查询分析结果

  执行查询分析操作后，您可以在**统计图表**中查看结果。此处通过表格（Pro版本）展示每个RDS实例的SQL平均执行延迟情况。



示例4

在RDS数据库审计日志中，统计执行频率Top 10的SQL语句及其执行次数，并统计每个SQL语句同比昨天的执行次数的变化情况。

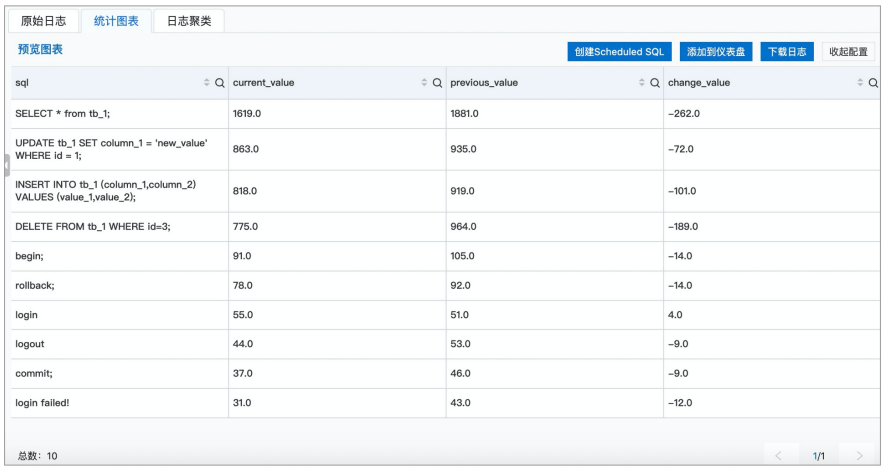
- Data Explorer配置

- 查询和分析语句

```
|
select
  "sql",
  diff [1] as current_value,
  diff [2] as previous_value,
  (diff [1]-diff [2]) as change_value
FROM  (
    select
      "sql",
      compare("count", 86400) as diff
    FROM      (
        select
          "sql",
          count,
          count * 1.0 / sum(count) over() as percent
        FROM          (
            select
              count(1) as count,
              "sql"
            FROM              log
            group by
              "sql"
            order by
              count desc
            limit
              10
          )
      )
    group by
      "sql"
  )
```

- 查询分析结果

  执行查询分析操作后，您可以在**统计图表**中查看结果。此处通过表格（Pro版本）展示统计结果。



19.4. 关联Logstore与MySQL数据库进行查询分析

本文以游戏公司数据分析场景为例，介绍日志服务Logstore与MySQL数据库关联分析功能。

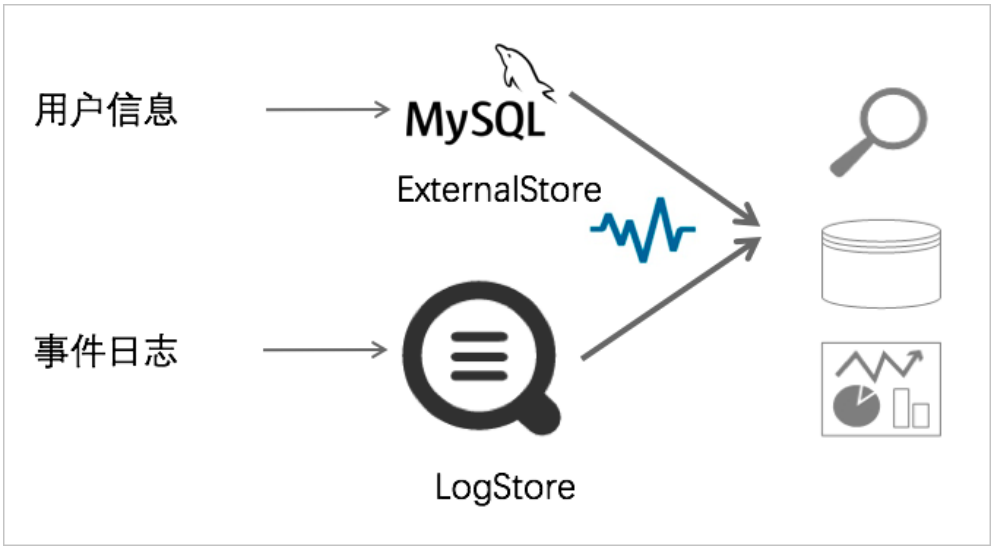
前提条件

- 已采集日志到日志服务。更多信息，请参见数据采集。
- 已为日志字段创建索引。更多信息，请参见配置索引。
- 已有可用的MySQL数据库。更多信息，请参见创建数据库和账号。

背景信息

某游戏公司，主要包括两大类数据：用户游戏日志数据和用户元数据。日志服务可实时采集用户游戏日志数据，包括操作、目标、血、魔法值、网络、支付手段、点击位置、状态码、用户ID等信息。然而用户元数据，包括用户的性别、注册时间、地域等信息，不能打印到日志中，所以一般存储到数据库中。现在该公司希望将用户游戏日志与用户元数据进行联合分析，获得最佳的游戏运营方案。

针对上述需求，日志服务查询分析引擎，提供Logstore和外部数据源（ExternalStore，例如MySQL数据库、OSS等）联合查询分析功能。您可以使用SQL的JOIN语法把用户游戏日志和用户元数据关联起来，分析与用户属性相关的指标。除此之外，您还可以将计算结果直接写入外部数据源中，便于结果的进一步处理。



操作步骤

1. 在MySQL数据库中，创建用户属性表。

   创建一张名为chiji_user的数据表，用于保存用户ID、昵称、性别、年龄、注册时间、账户余额和注册地域。

   ```
   CREATE TABLE `chiji_user` (
     `uid` int(11) NOT NULL DEFAULT '0',
     `user_nick` text,
     `gender` tinyint(1) DEFAULT NULL,
     `age` int(11) DEFAULT NULL,
     `register_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
     `balance` float DEFAULT NULL,
     `region` text, PRIMARY KEY (`uid`)
   ) ENGINE=InnoDB DEFAULT CHARSET=utf8
   ```

2. 添加白名单。
   - 如果是RDS MySQL数据库，需添加白名单地址100.104.0.0/16、11.194.0.0/16和11.201.0.0/16。更多信息，请参见设置IP白名单。
   - 如果是专有网络下ECS上自建的MySQL数据库且ECS上设置了安全组，需设置安全组规则，允许100.104.0.0/16、11.194.0.0/16和11.201.0.0/16网段访问。具体操作，请参见添加安全组规则。
3. 创建ExternalStore。
   i. 安装日志服务CLI。更多信息，请参见CLI概述。
   ii. 创建配置文件*/root/config.json*。

iii. 在*/root/config.json*文件中添加如下脚本，并根据实际情况替换参数配置。

```
{
"externalStoreName":"storename",
"storeType":"rds-vpc",
"parameter":
   {
   "region":"cn-qingdao",
   "vpc-id":"vpc-m5eq4irc1pucp*******",
   "instance-id":"i-m5eeo2whsn*******",
   "host":"localhost",
   "port":"3306",
   "username":"root",
   "password":"****",
   "db":"scmc",
   "table":"join_meta"
   }
}
```

参数	说明
externalStoreName	ExternalStore名称，必须小写。
storeType	数据源类型，固定为rds-vpc。
region	地域。详细说明如下： ■ 如果是RDS MySQL数据库，则配置region为RDS实例所在地域。 ■ 如果是专有网络下ECS上自建的MySQL数据库，则配置region为ECS实例所在地域。 ■ 如果是其他场景下自建的MySQL数据库，则配置region为空字符串，即**"region": ""**。
vpc-id	VPC ID。详细说明如下： ■ 如果是专有网络下的RDS MySQL数据库，则配置vpc-id为RDS实例所属专有网络的ID。 ■ 如果是专有网络下ECS上自建的MySQL数据库，则配置vpc-id为ECS实例所属专有网络的ID。 ■ 如果是经典网络下的RDS MySQL数据库或其他场景下自建的MySQL数据库，则配置vpc-id空字符串，即**"vpc-id": ""**。
instance-id	实例ID。详细说明如下： ■ 如果是RDS MySQL数据库，则配置instance-id为RDS实例的VpcCloudInstanceId。 您需要通过调用DescribeDBInstanceAttribute接口获取VpcCloudInstanceId。具体操作，请参见获取RDS实例的VpcCloudInstanceId。 ■ 如果是专有网络下ECS上自建的MySQL数据库，则配置instance-id为ECS实例ID。 ■ 如果是其他场景下自建的MySQL数据库，则配置instance-id为空字符串，即**"instance-id": ""**。
host	数据库地址。详细说明如下： ■ 如果是专有网络下的RDS MySQL数据库，则配置host为RDS实例的内网地址。 ■ 如果是专有网络下ECS上自建的MySQL数据库，则配置host为ECS的私网IP地址。 ■ 如果是其他场景下自建的MySQL数据库，则配置host为其可访问的host地址。

参数	说明
port	端口号。详细说明如下： ■ 如果是RDS MySQL数据库，则配置port为RDS实例的端口号。 ■ 如果是专有网络下ECS上自建的MySQL数据库，则配置port为ECS上MySQL的服务端口。 ■ 如果是其他场景下自建的MySQL数据库，则配置port为MySQL的服务端口。
username	数据库用户名。
password	数据库密码。
db	数据库。
table	数据库表。

iv. 创建ExternalStore。

其中project_name为日志服务Project名称，请根据实际情况替换。

```
aliyunlog log create_external_store --project_name="log-rds-demo" --config="file:///root/config.json"
```

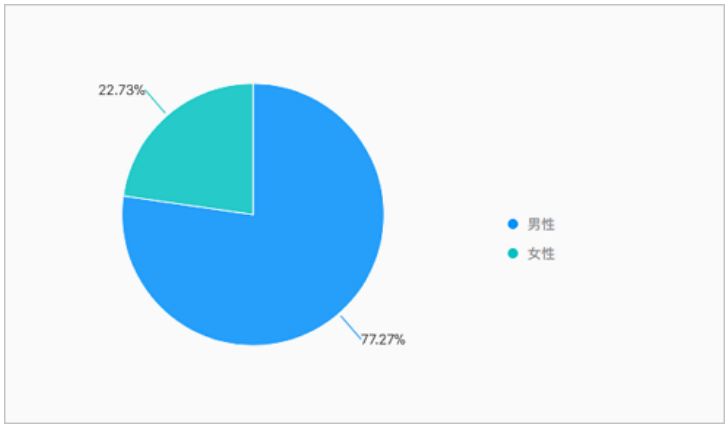
4. 使用JOIN语法进行联合查询分析。
   i. 登录日志服务控制台。
   ii. 在**Project列表**区域，单击目标Project。
   iii. 在**日志存储 > 日志库**页签中，单击目标Logstore。

iv. 执行查询分析语句。

指定日志中的userid字段和数据库表中的uid字段关联Logstore和MySQL数据库。

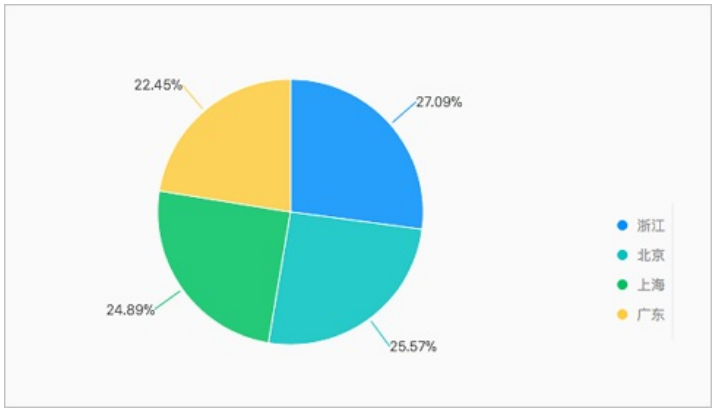
- 分析活跃用户的性别分布。

```
* | select case gender when 1 then '男性' else '女性' end as gender , count(1) as pv from l
og l join chiji_user u on l.userid = u.uid group by gender order by pv desc
```



- 分析不同地域活跃度。

```
* | select region , count(1) as pv from log l join chiji_user u on l.userid = u.uid group
by region order by pv desc
```



- 分析不同性别的消费情况。

```
* | select case gender when 1 then '男性' else '女性' end as gender , sum(money) as money f
rom log l join chiji_user u on l.userid = u.uid group by gender order by money desc
```

5. 保存查询分析结果到MySQL数据库中。

i. 在MySQL数据库中，创建名为report的数据表，该表存储每分钟的PV值。

```
CREATE TABLE `report` (
  `minute` bigint(20) DEFAULT NULL,
  `pv` bigint(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

ii. 参见步骤为report表创建ExternalStore。

iii. 在日志服务Logstore的查询分析页面中，执行如下查询语句将分析结果保存到report表中。

```
* | insert into report select __time__- __time__ % 300 as min, count(1) as pv group by min
```

保存成功后，您可以在MySQL数据库中查看保存结果。

```
mysql> select * from report;
+------------+-------+
| minute     | pv    |
+------------+-------+
| 1526448600 |  3000 |
| 1526448540 |  9900 |
| 1526448780 |  3100 |
| 1526448480 |  5400 |
| 1526448720 |  3000 |
| 1526448960 |  3000 |
| 1526448900 |  3000 |
| 1526449080 |  3000 |
| 1526449140 |  3000 |
| 1526448660 |  2900 |
| 1526449260 |  3000 |
```

19.5. 关联Logstore与OSS外表进行查询和分析

在进行日志数据查询和分析时，经常需要结合外部表格对日志数据进行分析。本文介绍如何在日志服务中联合OSS外表进行数据分析。

前提条件

- 已采集日志。更多信息，请参见数据采集。
- 已配置索引。更多信息，请参见配置索引。
- 已创建OSS Bucket。更多信息，请参见创建存储空间。

背景信息

某支付公司，想要分析用户年龄、地域、性别等因素对支付习惯的影响。该公司已通过日志服务实时采集用户支付行为（支付方式、支付费用等）日志，并将用户属性（地域、年龄、性别等）信息保存在OSS中。针对该场景，日志服务查询和分析引擎提供Logstore和外部数据源（ExternalStore，例如MySQL数据库、OSS等）联合查询和分析功能。您可以使用SQL的JOIN语法把用户属性数据和行为数据进行联合，分析与用户属性相关的指标。

与OSS进行关联查询和分析，具有如下优势：

- 节省费用：将更新频率低的数据保存在OSS上，只需要支付少量的存储费用，并且可以通过内网读数据，免去流量费用。
- 降低运维工作：在轻量级的联合分析平台中，不需要搬迁数据到同一个存储系统中。
- 节省时间：使用SQL分析数据，分析结果秒级可见，并可以将常用的分析结果定义为报表，打开即可看到结果。

操作步骤

1. 创建CSV文件并上传到OSS。
   i. 创建名为*user.csv*的文件。

```
userid,nick,gender,province,age
1,用户A,male,上海,18
2,用户B,female,浙江,19
3,用户C,male,广东,18
```

   ii. 上传*user.csv*文件到OSS。具体操作，请参见上传文件。

2. 登录日志服务控制台。
3. 在Project列表区域，单击目标Project。
4. 在**日志存储 > 日志库**页签中，单击目标Logstore。
5. 输入查询和分析语句，单击**查询/分析**。

   通过SQL定义虚拟外部存储（此处以user_meta1为例），映射到OSS文件，如果执行结果中的**result**为**true**，表示执行成功。

```
* | create table user_meta1 ( userid bigint, nick varchar, gender varchar, province varchar, age bigint) with ( endpoint='oss-cn-hangzhou.aliyuncs.com',accessid='LTAI5t8y9c113M7V****',accesskey='Y45H7bqvvgapWZR****',bucket='testoss',objects=ARRAY['user.csv'],type='oss')
```

   在查询和分析语句中定义外部存储名称、表的Schema等信息，并通过WITH语法指定OSS访问信息及文件信息，详细信息如下表所示。

配置项	说明	示例
外部存储名称	外部存储名称，即虚拟表的名称。	user_meta1
表的Schema	定义表的属性，包括表的列名及格式，例如(userid bigint, nick varchar, gender varchar, province varchar, age bigint)。	(userid bigint, nick varchar, gender varchar, province varchar, age bigint)
endpoint	OSS内网访问域名。更多信息，请参见访问域名和数据中心。	oss-cn-hangzhou.aliyuncs.com
accessid	您的AccessKey ID。更多信息，请参见访问密钥。	LTAI5t8y9c113M7V****
accesskey	您的AccessKey Secret。更多信息，请参见访问密钥。	Y45H7bqvvgapWZR****
bucket	CSV文件所在的OSS Bucket名称。	testoss
objects	CSV文件路径。 说明 objects为array类型，可以包含多个OSS文件。	user.csv
type	固定为oss，表示外部存储类型为OSS。	oss

6. 验证是否成功定义外部存储。

   执行如下语句，如果返回结果为您之前定义的表内容，则表示定义外部存储成功。其中，user_meta1为您定义的外部存储，请根据实际情况替换。

```
* | select * from user_meta1
```

userid	nick	gender	province	age
1	用户A	male	上海	18
2	用户B	female	浙江	19
3	用户C	male	广东	18

7. 通过JOIN语法完成Logstore和OSS外表的联合查询。

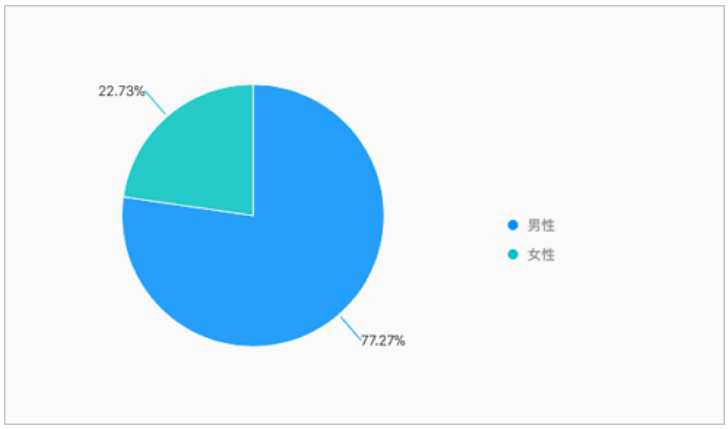
   执行如下查询和分析语句关联日志服务中日志的ID和OSS文件中的userid，补全日志信息。其中，test_accesslog为Logstore名称，l为Logstore别名，user_meta1为您定义的外部存储表，请根据实际情况替换。

   ```
   * | select * from test_accesslog l join user_meta1 u on l.userid = u.userid
   ```

   联合查询示例：

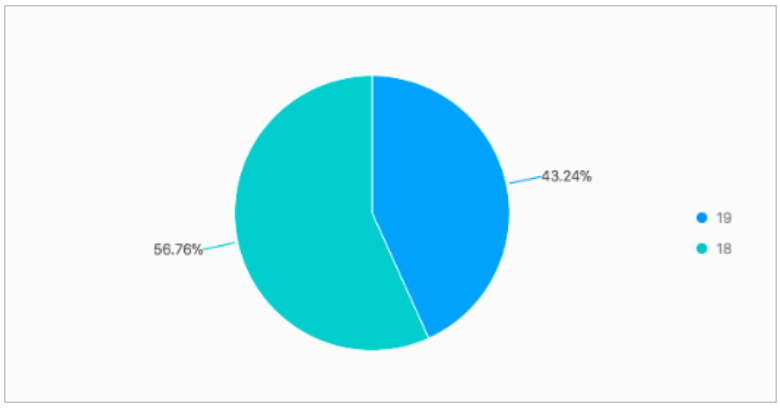
   - 统计不同性别用户的访问情况。

     ```
     * | select u.gender, count(1) from test_accesslog l join user_meta1 u on l.userid = u.userid g
     roup by u.gender
     ```

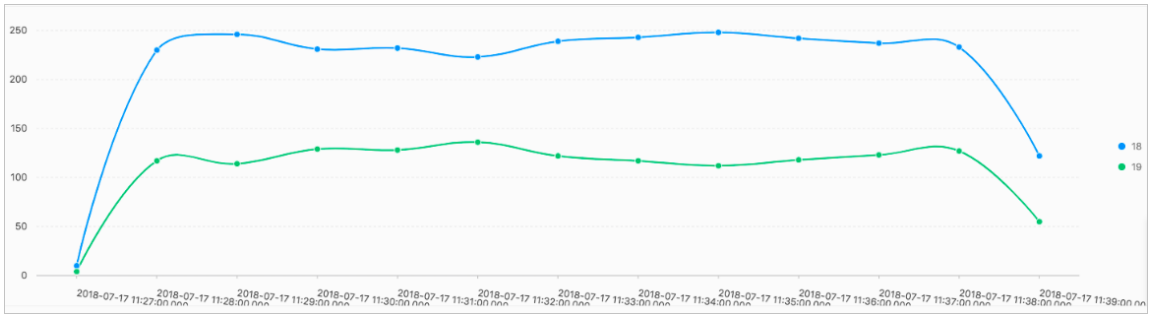
   - 统计不同年龄用户的访问情况。

     ```
     * | select u.age, count(1) from test_accesslog l join user_meta1 u on l.userid = u.userid grou
     p by u.age
     ```

   - 统计不同年龄段在时间维度上的访问趋势。

     ```
     * | select date_trunc('minute',__time__) as minute, count(1) ,u.age from test_accesslog l join
     user_meta1 u on l.userid = u.userid group by u.age,minute
     ```



19.6. 查询MNS日志

阿里云消息服务（MNS）日志推送到日志服务后，可进行实时查询，本文介绍实时查询的常用场景及操作步骤，您也可以通过多个关键字组合方式实现更加复杂的查询。

前提条件

- 已采集到MNS日志，详情请参见MNS日志。
- 已开启并配置索引，详情请参见配置索引。

背景信息

MNS日志包括队列消息操作日志和主题消息操作日志，日志内容包含消息生命周期的所有信息，例如时间、客户端、操作等。您可以通过实时查询、实时计算和离线计算三种方式对日志进行分析计算。

- 实时查询：在日志服务控制台上进行实时查询，例如：查询消息轨迹、写入量、删除量等。
- 实时计算：使用Spark、Storm、StreamCompute、Consumer Library等方式对MNS日志进行实时计算。例如：计算某个队列中，Top 10消息的产生者和消费者；计算生产和消费的速度，确认是否均衡；计算某些消费者的处理延时，确认是否存在瓶颈等。
- 离线计算：使用MaxCompute、E-MapReduce、Hive进行长时间跨度的计算，例如：计算最近一周内消息从发布到被消费的平均延迟。

查询队列消息的消息轨迹

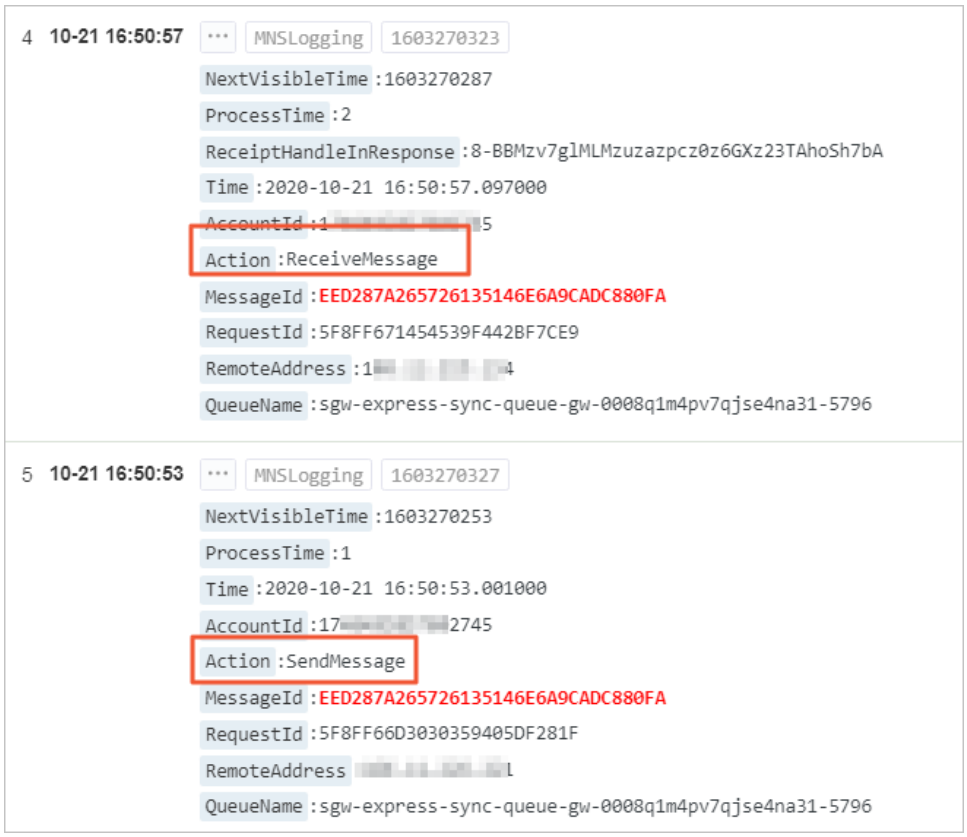
1.
2.
3.
4. 输入查询语句。

   本案例要查询队列消息的消息轨迹，即输入队列名称和消息ID，格式为**$QueueName and $MessageId**，例如**log and FF973C9C6572630D7F963C527CC5A82C**。
5. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。

   您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。
6. 单击**查询/分析**。

查询结果如下所示，记录了某条消息从发送到接收的过程。



查询队列消息发送量

1. 在目标Logstore的查询分析页面，输入查询语句。

   本案例要查询队列消息发送量，即输入队列名称和发送操作，查询语句格式为**$QueueName and (SendMessage or BatchSendMessage)**，例如**log and (SendMessage or BatchSendMessage)**。

2. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。

   您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

3. 单击**查询/分析**。

   查询结果如下所示，当前查询时段内，生产者向log队列发送了3条队列消息。

查询队列消息消费量

1. 在目标Logstore的查询分析页面，输入查询语句。

   本案例要查询队列消息消费量，即输入队列名称和消费操作，查询语句格式为**$QueueName and (ReceiveMessage or BatchReceiveMessage)**，例如**log and (ReceiveMessage or BatchReceiveMessage)**。

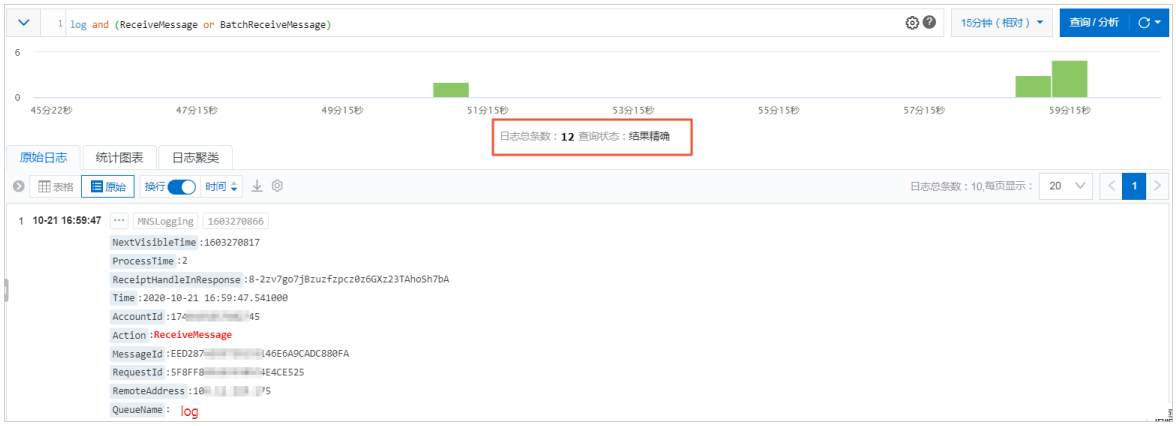
2. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。

   您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

3. 单击**查询/分析**。

   查询结果如下所示，当前查询时段内，log队列中有12条消息被消费。



查询队列消息删除量

1. 在目标Logstore的查询分析页面，输入查询语句。

   本案例要查询队列消息删除量，即输入队列名称和删除操作，查询语句格式为**$QueueName and (DeleteMessage or BatchDeleteMessage)**，例如**log and (DeleteMessage or BatchDeleteMessage)**。

2. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。

   您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

3. 单击**查询/分析**。

   查询结果如下所示，当前查询时段内，61条log队列消息被删除。

查询主题消息的消息轨迹

1. 在目标Logstore的查询分析页面，输入查询语句。

   本案例要查询主题消息的消息轨迹，即输入主题名称和MessageId，查询语句格式为**$TopicName and $MessageId**，例如**logtest and 979628CD657261357FCB3C8A68BFA0E3**。

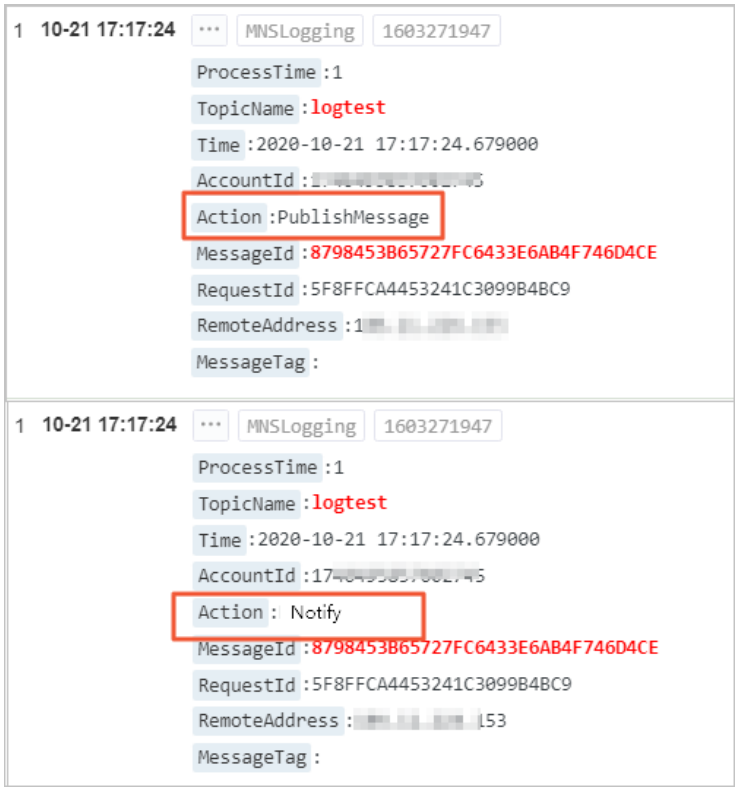
2. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。

   您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

3. 单击**查询/分析**。

   查询结果如下图所示，记录了某条消息从发送到通知的过程。



查询主题消息发布量

1. 在目标Logstore的查询分析页面，输入查询语句。

   本案例要查询主题消息发布量，即输入主题名称和发布操作，查询语句格式为**$TopicName and PublishMessage**，例如**logtest and PublishMessage**。

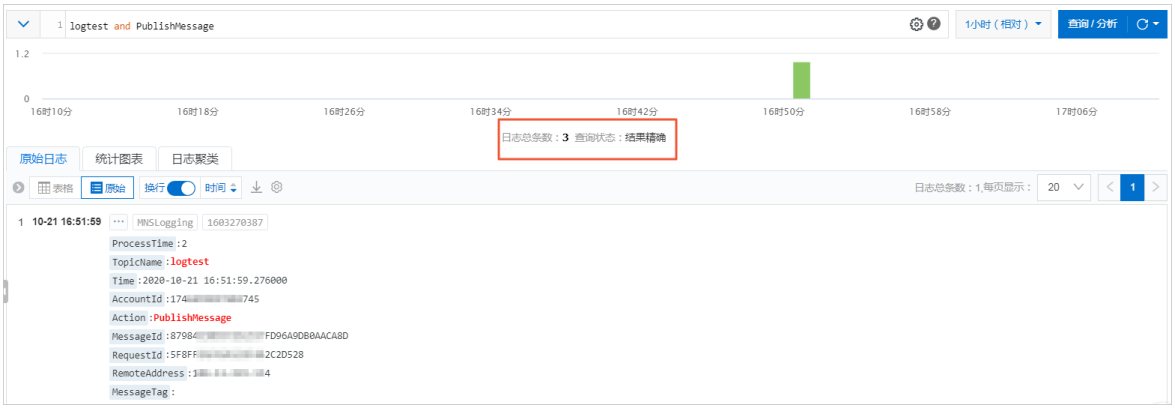
2. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。

   您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

3. 单击**查询/分析**。

查询结果如下图所示，当前查询时段内，生产者向logtest主题发布了3条消息。



查询某个客户端消息处理量

1. 在目标Logstore的查询分析页面，输入查询语句。

   本案例要查询某个客户端消息处理量，即输入客户端IP地址，查询语句格式为**$ClientIP**，例如**10.10.10.0**。

   如果您要查询某个客户端的某类操作日志，可使用多个关键字组合方式，例如**$ClientIP and (SendMessage or BatchSendMessage)**。

2. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。

   您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

3. 单击**查询/分析**。

   查询结果如下图所示，当前查询时段内，该客户端处理了66条消息。

19.7. 采集及分析Nginx监控日志

Nginx中的自建状态页，可帮忙您监控Nginx状态。日志服务支持通过Logtail插件采集Nginx状态信息，并对监控日志进行查询分析、告警等操作，全方位监控您的Nginx集群。

前提条件

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

> **说明** 目前支持Linux Logtail 0.16.0及以上版本，Window Logtail 1.0.0.8及以上版本。

步骤一：准备环境

请按照以下步骤，开启Nginx status插件。

1. 执行以下命令确认Nginx已具备status功能。

   ```
   nginx -V 2>&1 | grep -o with-http_stub_status_module
   with-http_stub_status_module
   ```

   如果回显信息为 `with-http_stub_status_module`，表示支持status功能。

2. 配置Nginx status。

   在Nginx配置文件（默认为/etc/nginx/nginx.conf）中开启status功能，配置示例如下所示，详情请参见Nginx status。

   > **说明** allow 10.10.XX.XX表示只允许IP地址为10.10.XX.XX的服务器访问nginx status功能。

   ```
   location /private/nginx_status {
     stub_status on;
     access_log   off;
     allow 10.10.XX.XX;
     deny all;
   }
   ```

3. 执行如下命令验证安装Logtail的服务器具备nginx status访问权限。

   ```
   $curl http://10.10.XX.XX/private/nginx_status
   ```

   如果回显信息如下所示，则表示已完成Nginx status配置。

   ```
   Active connections: 1
   server accepts handled requests
   2507455 2507455 2512972
   Reading: 0 Writing: 1 Waiting: 0
   ```

步骤二：采集Nginx监控日志

1.
2. 在**接入数据**区域，选择**自定义数据插件**。
3. 选择目标Project和Logstore，单击**下一步**。
4. 创建机器组。
   - 如果您已有可用的机器组，请单击**使用现有机器组**。
   - 如果您还没有可用的机器组，请执行以下操作（以ECS为例）。
     a. 在**ECS机器**页签中，通过手动选择实例方式选择目标ECS实例，单击**立即执行**。

        更多信息，请参见安装Logtail（ECS实例）。

        > **说明** 如果您的服务器是与日志服务属于不同账号的ECS、其他云厂商的服务器和自建IDC时，您需要手动安装Logtail。更多信息，请参见安装Logtail（Linux系统）或安装Logtail（Windows系统）。手动安装Logtail后，您还需要在该服务器上手动配置用户标识。具体操作，请参见配置用户标识。

     b. 安装完成后，单击**确认安装完毕**。

     c. 在**创建机器组**页面，输入**名称**，单击**下一步**。

        日志服务支持创建IP地址机器组和用户自定义标识机器组，详细参数说明请参见创建IP地址机器组和创建用户自定义标识机器组。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

   > **注意** 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为FAIL，您可单击**自动重试**。如果还未解决，请参见Logtail机器组无心跳进行排查。

6. 在**数据源设置**页签中，配置**配置名称**和**插件配置**，然后单击**下一步**。
   - inputs为Logtail采集配置，必选项，请根据您的数据源配置。

> **说明** 一个inputs中只允许配置一个类型的数据源。

- processors为Logtail处理配置，可选项。您可以配置一种或多种处理方式，详情请参见概述。

```
{
"inputs": [
 {
      "type": "metric_http",
      "detail": {
          "IntervalMs": 60000,
          "Addresses": [
              "http://10.10.XX.XX/private/nginx_status",
              "http://10.10.XX.XX/private/nginx_status",
              "http://10.10.XX.XX/private/nginx_status"
          ],
          "IncludeBody": true
      }
 }
],
"processors": [
 {
      "type": "processor_regex",
      "detail": {
          "SourceKey": "content",
          "Regex": "Active connections: (\\d+)\\s+server accepts handled requests\\s+(\\d+)\\s+(\
\d+)\\s+(\\d+)\\s+Reading: (\\d+) Writing: (\\d+) Waiting: (\\d+)[\\s\\S]*",
          "Keys": [
              "connection",
              "accepts",
              "handled",
              "requests",
              "reading",
              "writing",
              "waiting"
          ],
          "FullMatch": true,
          "NoKeyError": true,
          "NoMatchError": true,
          "KeepSource": false
      }
 }
]
}
```

重要参数说明如下表所示：

参数	类型	是否必须	说明
type	string	是	数据源类型，固定为metric_http。
IntervalMs	int	是	每次请求的间隔，单位：ms。
Addresses	数组	是	配置为您需要监控的URL列表。
IncludeBody	boolean	否	是否采集请求的body，默认值：false。如果为true，则将请求body内容存放在名为content的字段中。

完成采集配置1分钟后，即可查看日志数据，样例如下所示。并且日志服务默认生成nginx_status仪表盘，展示查询分析结果。

```
_address_:http://10.10.XX.XX/private/nginx_status
_http_response_code_:200
_method_:GET
_response_time_ms_:1.83716261897
_result_:success
accepts:33591200
connection:450
handled:33599550
reading:626
requests:39149290
waiting:68
writing:145
```

步骤三：查询分析日志

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。

   您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

5. 在搜索框中输入查询分析语句，单击**查询/分析**。

   查询分析语句语法，请参见基本语法。您还可以为查询结果设置告警，更多信息，请参见快速设置日志告警。

   - 查询日志
     - 查询某IP地址的请求状态。

       ```
       _address_ : 10.10.0.0
       ```

     - 查询响应时间超过100 ms的请求。

       ```
       _response_time_ms_ > 100
       ```

     - 查询状态码非200的请求。

       ```
       not _http_response_code_ : 200
       ```

   - 分析日志
     - 每5分钟统计waiting、reading、writing、connection的平均值。

       ```
       *| select  avg(waiting) as waiting, avg(reading)  as reading,  avg(writing)  as writing,  av
       g(connection)  as connection,  from_unixtime( __time__ - __time__ % 300) as time group by __
       time__ - __time__ % 300 order by time limit 1440
       ```

     - 统计最大等待连接数排名前十的服务器。

       ```
       *| select  max(waiting) as max_waiting, address, from_unixtime(max(__time__)) as time group
       by address order by max_waiting desc limit 10
       ```

- 统计请求IP的数量以及请求失败的IP数量。

```
* | select  count(distinct(address)) as total
```

```
not _result_ : success | select  count(distinct(address))
```

- 统计最近十次访问失败的IP地址。

```
not _result_ : success | select _address_ as address, from_unixtime(__time__) as time  order by __time__ desc limit 10
```

- 每5分钟统计请求处理总数。

```
*| select  avg(handled) * count(distinct(address)) as total_handled, avg(requests) * count(distinct(address)) as total_requests,  from_unixtime( __time__ - __time__ % 300) as time group by __time__ - __time__ % 300 order by time limit 1440
```

- 每5分钟统计平均请求延迟。

```
*| select  avg(_response_time_ms_) as avg_delay,  from_unixtime( __time__ - __time__ % 300) as time group by __time__ - __time__ % 300 order by time limit 1440
```

- 统计请求成功的数量和失败的数量。

```
not _http_response_code_ : 200  | select  count(1)
```

```
_http_response_code_ : 200  | select  count(1)
```

19.8. 分析Nginx访问日志

日志服务支持采集Nginx日志，并进行多维度分析。本文介绍分析网站访问情况、诊断及调优网站和重要场景告警的分析案例。

前提条件

已采集Nginx访问日志，详情请参见使用Nginx模式采集日志。

在日志采集配置向导中，已根据日志字段自动生成索引，如果您要修改索引，详情请参见配置索引。

背景信息

Nginx是一款主流的网站服务器，当您选用Nginx搭建网站时，Nginx日志是运维网站的重要信息。传统模式下，需使用CNZZ等方式，在前端页面插入JS，记录访问请求。或者利用流计算、离线计算分析Nginx访问日志，此方式还需要搭建环境，在实时性以及分析灵活性上难以平衡。

日志服务支持通过数据接入向导一站式采集Nginx日志，并为Nginx日志创建索引和仪表盘。nginx_Nginx访问日志仪表盘包括来源IP分布、请求状态占比、请求方法占比、访问PV/UV统计、流入流出流量统计、请求UA占比、前十访问来源、访问前十地址和请求时间前十地址等信息，全方位展示网站访问情况。您还可以使用日志服务的查询分析语句，分析网站的延时情况，及时调优网站。针对性能问题、服务器错误、流量变化等重要场景，您还可以设置告警，当满足告警条件时给您发送告警信息。

分析网站访问情况

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**中，单击目标Logstore左侧的>。
4. 在**可视化仪表盘**中，单击**nginx_Nginx访问日志**。

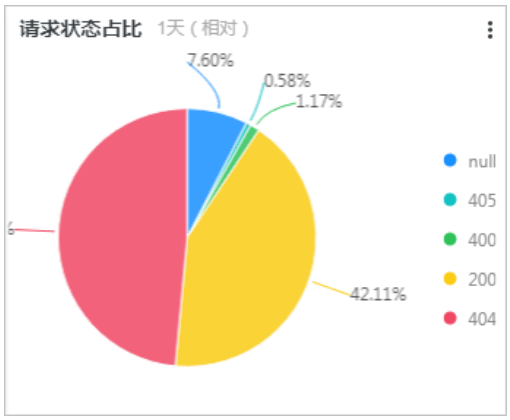
   nginx_Nginx访问日志仪表盘中的重要图表说明如下所示：

   - **来源IP分布**图展示最近一天访问IP地址的来源情况，所关联的查询分析语句如下所示：

```
* | select count(1) as c, ip_to_province(remote_addr) as address group by address limit 100
```

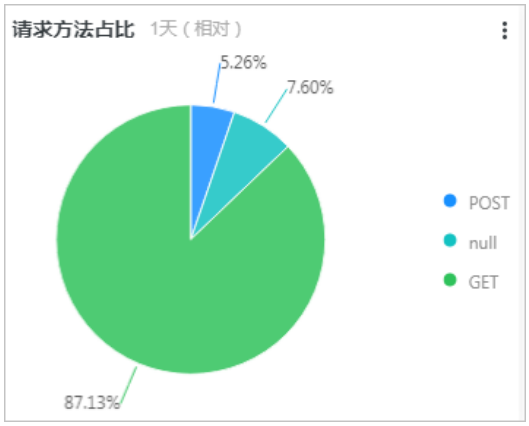
- **请求状态占比**图展示最近一天各HTTP状态码的占比情况，所关联的查询分析语句如下所示：

```
 * | select count(1) as pv,
        status
        group by status
```



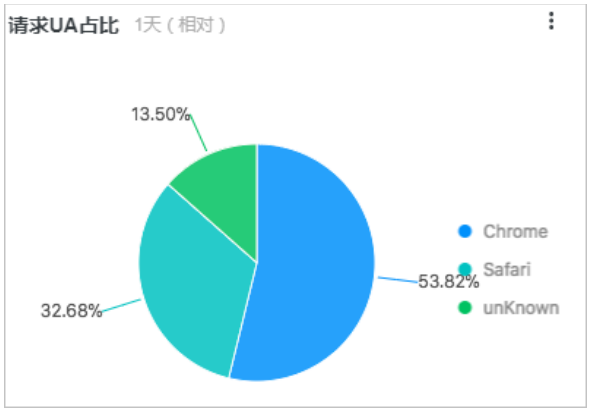
- **请求方法占比**图展示最近一天各请求方法的占比情况，所关联的查询分析语句如下所示：

```
* | select count(1) as pv ,request_method group by request_method
```



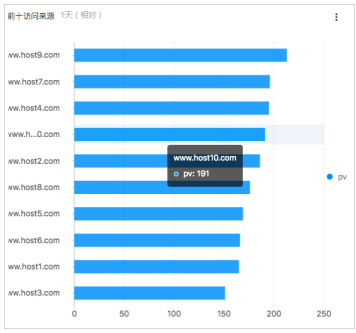
- **请求UA占比**图展示最近一天各种浏览器的占比情况，所关联的查询分析语句如下所示：

```
* | select count(1) as pv, case when http_user_agent like '%Chrome%' then 'Chrome' when http_u
ser_agent like '%Firefox%' then 'Firefox' when http_user_agent like '%Safari%' then 'Safari' e
lse 'unKnown' end as http_user_agent  group by case when http_user_agent like '%Chrome%' then
'Chrome' when http_user_agent like '%Firefox%' then 'Firefox' when http_user_agent like '%Safa
ri%' then 'Safari' else 'unKnown' end   order by pv desc limit 10
```



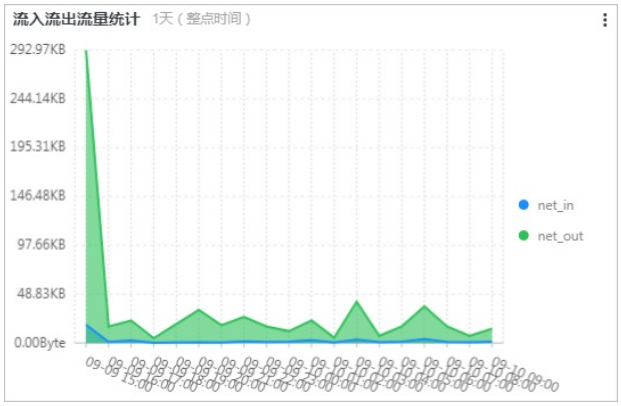
- **前十访问来源**图展示最近一天PV数最多的前十个访问来源页面，所关联的查询分析语句如下所示：

```
* | select count(1) as pv , http_referer  group by http_referer order by pv desc limit 10
```



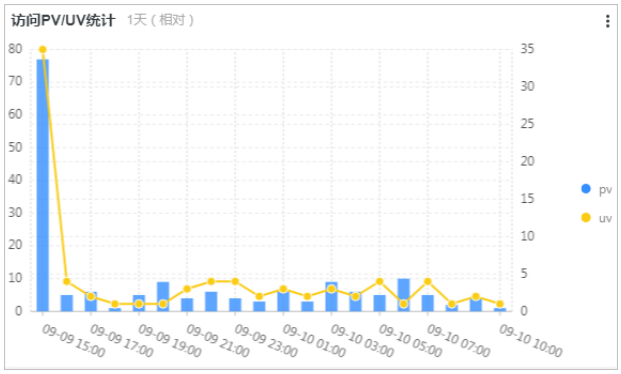
- **流入流出流量统计**图展示最近一天流量的流入和流出情况，所关联的查询分析语句如下所示：

```
* | select sum(body_bytes_sent) as net_out, sum(request_length) as net_in ,date_format(date_tr
unc('hour', __time__), '%m-%d %H:%i')  as time group by date_format(date_trunc('hour', __time_
_), '%m-%d %H:%i') order by time limit 10000
```



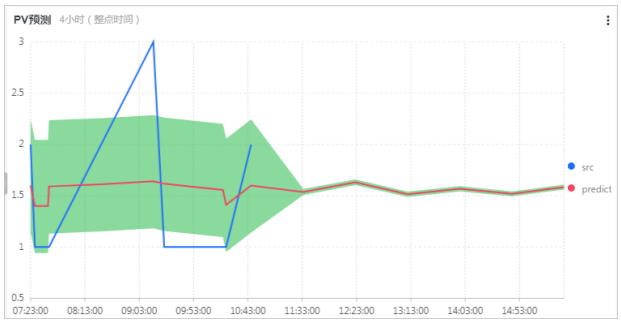
- **访问PV/UV统计**图展示最近一天内的PV数和UV数，所关联的查询分析语句如下所示：

```
*| select approx_distinct(remote_addr) as uv ,count(1) as pv , date_format(date_trunc('hour',
__time__), '%m-%d %H:%i')  as time group by date_format(date_trunc('hour', __time__), '%m-%d %
H:%i')  order by time limit 1000
```



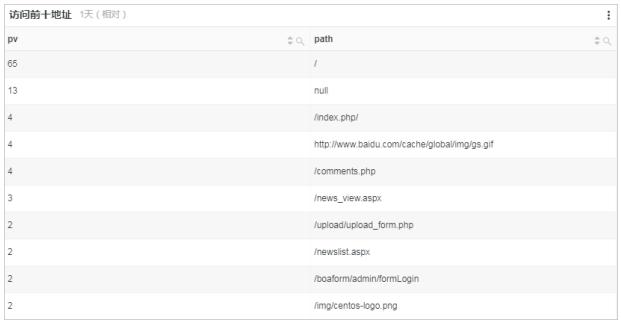
- **PV预测**图预测未来4小时的PV数，所关联的查询分析语句如下所示：

```
* | select ts_predicate_simple(stamp, value, 6, 1, 'sum') from (select __time__ - __time__ % 6
0 as stamp, COUNT(1) as value from log GROUP BY stamp order by stamp) LIMIT 1000
```



- **访问前十地址**图展示最近一天PV数最多的前十个访问地址，所关联的查询分析语句如下所示：

```
* | select count(1) as pv, split_part(request_uri,'?',1) as path  group by path order by pv de
sc limit 10
```



诊断及调优网站

在网站运行过程中，还需关注请求延时问题，例如处理请求延时情况如何、哪些页面的延时较大等。您可以自定义查询分析语句分析延迟情况，相关案例如下所示，操作步骤可参见查询和分析日志。

- 计算每5分钟请求的平均延时和最大延时，从整体了解延时情况。

```
* | select from_unixtime(__time__ -__time__% 300) as time,
        avg(request_time) as avg_latency ,
        max(request_time) as max_latency
        group by __time__ -__time__% 300
```

- 统计最大延时对应的请求页面，进一步优化页面响应。

```
* | select from_unixtime(__time__ - __time__% 60) ,
        max_by(request_uri,request_time)
        group by __time__ - __time__%60
```

- 统计分析网站所有请求的延时分布，将延时分布分成10个组，分析每个延时区间的请求个数。

```
* |select numeric_histogram(10,request_time)
```

- 计算最大的十个延时及其对应值。

```
* | select max(request_time,10)
```

- 对延时最大的页面进行调优。

  例如/url2页面的访问延时最大，需要对/url2页面进行调优，则需计算/url2页面的访问PV、UV、各种请求方法次数、各种请求状态次数、各种浏览器次数、平均延时和最大延时。

```
request_uri:"/url2" | select count(1) as pv,
        approx_distinct(remote_addr) as uv,
        histogram(method) as method_pv,
        histogram(status) as status_pv,
        histogram(user_agent) as user_agent_pv,
        avg(request_time) as avg_latency,
        max(request_time) as max_latency
```

告警

针对性能问题、网站错误、流量急跌或暴涨等情况，您可以设置查询分析语句，并设置告警，操作步骤请参见设置告警。

- 错误告警

  在网站运行过程中，一般需关注500错误，即服务器错误。您可以使用如下查询分析语句计算单位时间内的错误数c，并将告警触发条件设置为c > 0。

```
status:500 | select count(1) as c
```

> **说明** 对于一些业务压力较大的服务，偶尔出现几个500错误是正常现象。针对此情况，您可以在创建告警时，设置**触发通知阈值**为2，即只有连续2次检查都符合条件才产生告警。

- 性能告警

  如果在服务器运行过程出现延迟增大情况，您可以针对延迟创建告警。例如您可以使用如下查询分析语句分析 `/adduser` 接口所有请求方法为 `Post` 的写请求延时，并将告警触发条件设置为 l > 300000，即当延时平均值超过300ms则产生告警。

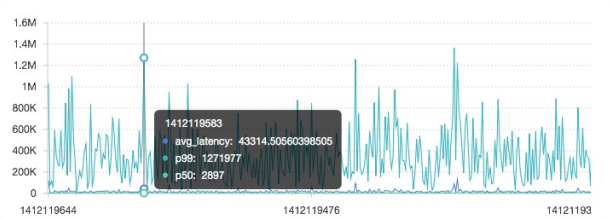
```
Method:Post and URL:"/adduser" | select avg(Latency) as l
```

使用平均值创建告警的方式比较简单，但会造成一些个体请求延时被平均，无法反映真实情况。针对此问题，您可以使用数学统计中的百分数（例如99%最大延时）来作为告警触发条件，例如使用如下查询分析语句计算99%分位的延时大小。

```
Method:Post and URL:"/adduser" | select approx_percentile(Latency, 0.99) as p99
```

在监控场景中，您可以使用如下查询分析语句计算一天窗口（1440分钟）内各分钟的平均延时大小、50%分位的延时大小和90%分位的延时大小。

```
* | select avg(Latency) as l, approx_percentile(Latency, 0.5) as p50, approx_percentile(Latency, 0
.99) as p99, date_trunc('minute', time) as t group by t order by t desc limit 1440
```



- 流量急跌或暴涨告警

  如果在网站运行过程中出现流量急跌或暴涨情况，一般属于不正常现象。针对此问题，您可以计算流程大小，并设置告警。一般根据如下参考信息反映流量的急跌或暴涨情况：

  - 上一个时间窗口：环比上一个时间段。
  - 上一天该时间段的窗口：环比昨天。
  - 上一周该时间段的窗口：环比上周。

  本案例以第一种情况为例，计算流量大小的变动率，日志查询范围为5分钟。

  i. 定义一个计算窗口。

  定义一个1分钟的窗口，计算该分钟内的流量大小。

  ```
  * | select sum(inflow)/(max(__time__)-min(__time__)) as inflow , __time__-__time__%60  as wind
  ow_time from log group by window_time order by window_time limit 15
  ```

  从分析结果中看，每个窗口内的平均流量是均匀的。

window_time	inflow
1513045740	315574947
1513045800	333233937
1513045860	335821584
1513045920	330556452
1513045980	316785257

  ii. 计算窗口内的差异值。

- 计算最大值或最小值与平均值的变化率，此处以最大值max_ratio为例。

  本示例中计算结果max_ratio为1.02，您可以定义告警条件为max_ratio > 1.5（变化率超过50%）则告警。

```
 * | select max(inflow)/avg(inflow) as max_ratio from (select sum(inflow)/(max(__time__)-min
(__time__)) as inflow , __time__-__time__%60  as window_time from log group by window_time o
rder by window_time limit 15)
```

window_time	inflow
1513045740	315574947
1513045800	333233937
1513045860	335821584
1513045920	330556452
1513045980	316785257

- 计算最近值变化率，查看最新的数值是否有波动或已恢复。

  通过max_by方法获取窗口中的最大流量进行判断，本案例中的计算结果lastest_ratio为0.97。

```
 * | select max_by(inflow, window_time)/1.0/avg(inflow) as lastest_ratio from (select sum(in
flow)/(max(__time__)-min(__time__)) as inflow , __time__-__time__%60  as window_time from lo
g group by window_time order by window_time limit 15)
```

> **说明** max_by函数计算结果为字符类型，需强制转换成数字类型。 如果您要计算变化相对率，可以用（1.0-max_by(inflow, window_time)/1.0/avg(inflow)) as lastest_ratio。

window_time	inflow
1513045740	315574947
1513045800	333233937
1513045860	335821584
1513045920	330556452
1513045980	316785257

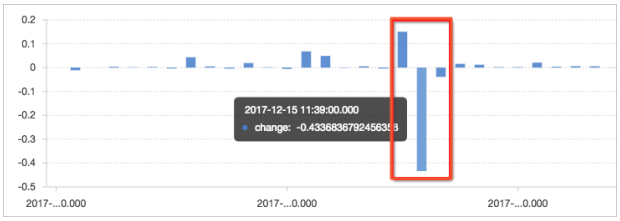
- 计算波动率，即计算当前窗口值与上个窗口值的变化值。

window_time	inflow
1513308660	8138522256
1513308720	8584340710
1513308780	9210706832
1513308840	9684619494

使用窗口函数lag提取当前流量inflow与上一个周期流量inflow "lag(inflow, 1, inflow)over() "进行差值计算，并除以当前流量inflow获取变化率。例如11点39分流量有一个较大的降低，窗口之间变化率为40%以上。

> **说明** 如果要定义一个绝对变化率，可以使用abs函数对计算结果进行统一。

```
 * | select (inflow- lag(inflow, 1, inflow)over() )*1.0/inflow as diff, from_unixtime(window
_time) from (select sum(inflow)/(max(__time__)-min(__time__)) as inflow , __time__-__time__%
60  as window_time from log group by window_time order by window_time limit 15)
```



19.9. 分析Apache日志

日志服务支持采集Apache日志，并进行多维度分析。本文通过PV、UV、访问地域分布、错误请求、客户端类型等维度分析Apache日志，以评估网站访问情况。

前提条件

已采集Apache日志，详情请参见使用Apache模式采集日志。

在数据接入向导中，已根据日志字段自动生成索引，如果您要修改索引，详情请参见配置索引。

背景信息

Apache是一款主流的网站服务器，当您选用Apache搭建网站时，Apache日志是运维网站的重要信息。

日志服务支持通过数据接入向导一站式采集Apache日志，并为Apache日志创建索引和仪表盘。apache_Apache访问日志仪表盘包括来源IP分布、请求状态占比、请求方法占比、访问PV/UV统计、流入流出流量统计、请求UA占比、前十访问来源、访问前十地址和请求时间前十地址等信息，全方位展示网站访问情况。

操作步骤

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**中，单击目标Logstore左侧的>。
4. 在**可视化仪表盘**中，单击**apache_Apache访问日志**。

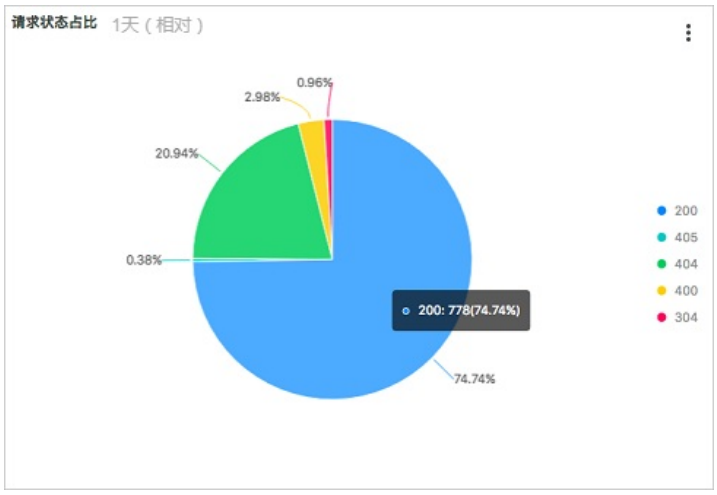
   apache_Apache访问日志仪表盘包括如下图表：

   - **来源IP分布**图展示访问IP地址的来源情况，所关联的查询分析语句如下所示：

     ```
     * | select ip_to_province(remote_addr) as address, count(1) as c group by ip_to_province(remote_addr) limit 100
     ```

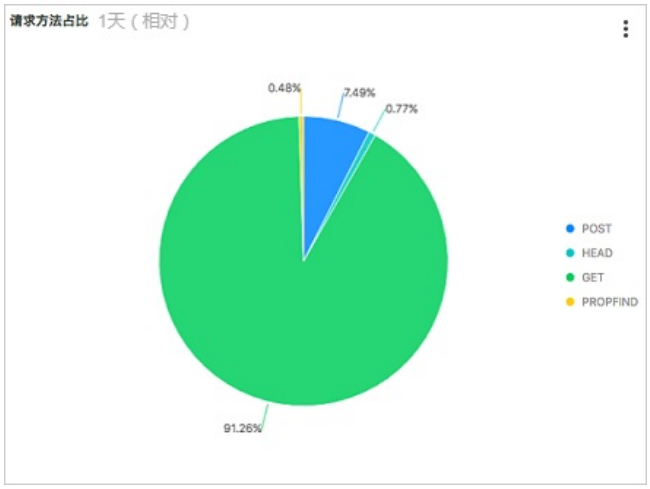
   - **请求状态占比**图展示最近一天各HTTP状态码的占比情况，所关联的查询分析语句如下所示：

     ```
     * | select status, count(1) as pv group by status
     ```

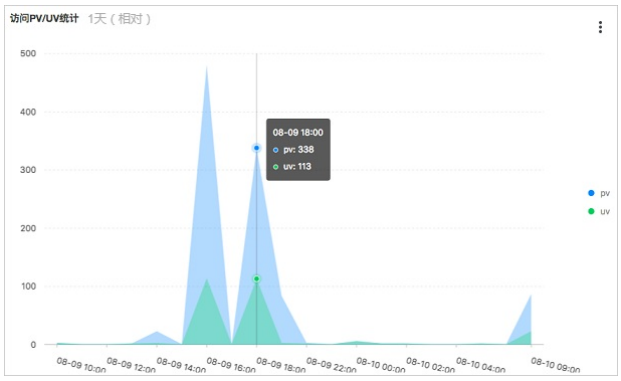
   - **请求方法占比**图展示最近一天各请求方法的占比情况，所关联的查询分析语句如下所示：

     ```
     * | select request_method, count(1) as pv group by request_method
     ```



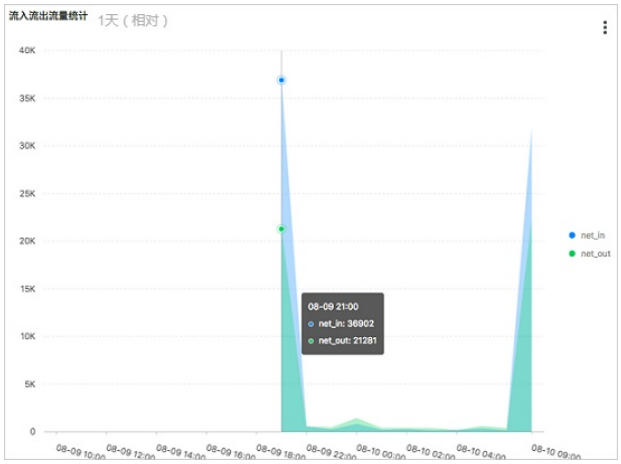
- **访问PV/UV统计**图展示最近一天内的PV数和UV数，所关联的查询分析语句如下所示：

```
* | select date_format(date_trunc('hour', __time__), '%m-%d %H:%i')  as time, count(1) as pv, approx_distinct(remote_addr) as uv group by date_format(date_trunc('hour', __time__), '%m-%d %H:%i') order by time limit 1000
```



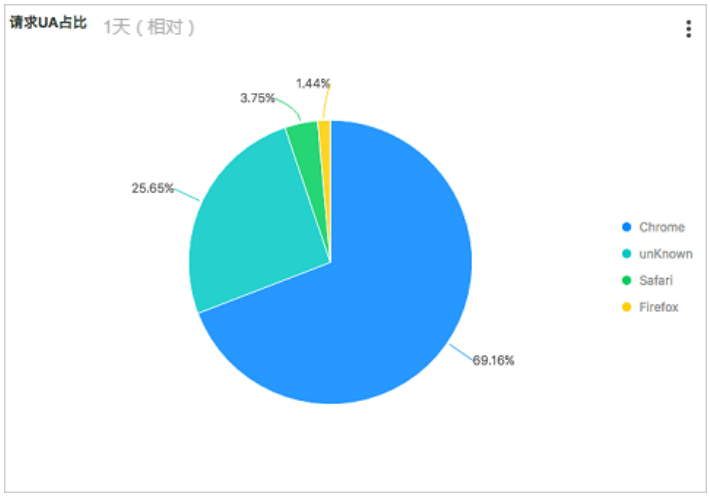
- **流入流出流量统计**图展示流量的流入和流出情况，所关联的查询分析语句如下所示：

```
* | select date_format(date_trunc('hour', __time__), '%m-%d %H:%i') as time, sum(bytes_sent) as net_out, sum(bytes_received) as net_in group by time order by time limit 10000
```



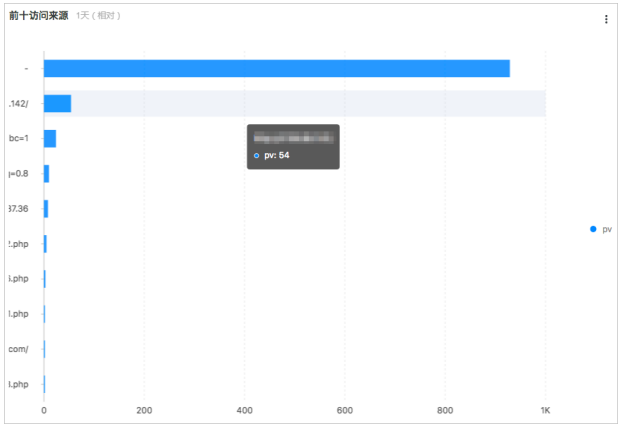
- **请求UA占比**图展示最近一天各种浏览器的占比情况，所关联的查询分析语句如下所示：

```
* | select case when http_user_agent like '%Chrome%' then 'Chrome' when http_user_agent like '%Firefox%' then 'Firefox' when http_user_agent like '%Safari%' then 'Safari' else 'unKnown' end as http_user_agent, count(1) as pv group by case when http_user_agent like '%Chrome%' then 'Chrome' when http_user_agent like '%Firefox%' then 'Firefox' when http_user_agent like '%Safari%' then 'Safari' else 'unKnown' end  order by pv desc limit 10
```



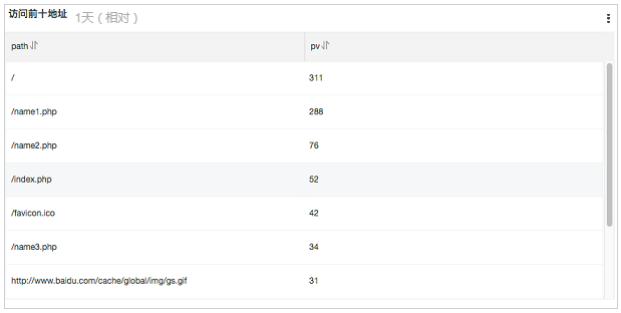
- **前十访问来源**图展示最近一天PV数最多的前十个访问来源页面，所关联的查询分析语句如下所示：

```
* | select  http_referer, count(1) as pv group by http_referer order by pv desc limit 10
```



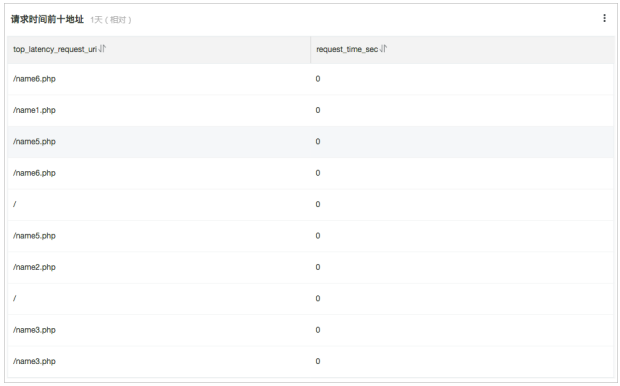
- **访问前十地址**展示最近一天PV数最多的前十个访问地址，所关联的查询分析语句如下所示：

```
* | select split_part(request_uri,'?',1) as path,  count(1) as pv group by split_part(request_uri,'?',1) order by pv desc limit 10
```



- **请求时间前十地址**图展示最近一天请求响应延时最长的前十个地址，所关联的查询分析语句如下所示：

```
* | select request_uri as top_latency_request_uri,
            request_time_sec
            order by request_time_sec desc limit 10 10
```



请求时间前十地址 1天（相对）

top_latency_request_uri	request_time_sec
/name6.php	0
/name1.php	0
/name5.php	0
/name6.php	0
/	0
/name5.php	0
/name2.php	0
/	0
/name3.php	0
/name3.php	0

19.10. 分析IIS日志

日志服务支持采集IIS日志，并进行多维度分析。本文通过PV、UV、访问地域分布、错误请求、请求方法等维度分析IIS日志，以评估网站访问情况。

前提条件

已采集IIS日志，详情请参见使用IIS模式采集日志。

在日志采集配置向导中，已根据日志字段自动生成索引，如果您要修改索引，详情请参见配置索引。

背景信息

IIS是一款主流的网站服务器，具备简单易用、安全性能高等优势。当您选用IIS搭建网站时，IIS日志是运维网站的重要信息。

日志服务推荐选用W3C日志格式，W3C配置格式如下所示：

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

IIS日志样例如下所示：

```
#Software: Microsoft Internet Information Services 7.5
#Version: 1.0
#Date: 2020-09-08 09:30:26
#Fields: date time s-sitename s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username c-ip cs(User
-Agent) sc-status sc-substatus sc-win32-status sc-bytes cs-bytes time-taken
2009-11-26 06:14:21 W3SVC692644773 125.67.67.* GET /index.html - 80 - 192.0.2.0 Baiduspider+(+http://
www.example.com)200 0 64 185173 296 0
```

- 字段前缀说明

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

前缀	说明
sc-	服务器到客户端的操作

- 各个字段说明

字段	说明
date	客户端发送请求的日期.。
time	客户端发送请求的时间。
s-sitename	服务名，表示客户端所访问的站点的Internet服务和实例的号码。
s-computername	服务器名，表示生成日志的服务器名称。
s-ip	服务器IP地址，表示生成日志的服务器的IP地址。
cs-method	请求方法，例如：GET、POST。
cs-uri-stem	URI资源，表示请求访问的地址。
cs-uri-query	URI查询，表示查询HTTP请求中问号（?）后的信息。
s-port	服务器端口，表示连接客户端的服务器端口号。
cs-username	通过验证的域或用户名。 对于通过身份验证的用户，格式为 `域\用户名` ；对于匿名用户，显示短划线（-）。
c-ip	客户端IP地址，表示访问服务器的客户端真实IP地址。
cs-version	协议版本，例如：HTTP 1.0、HTTP 1.1。
cs(User-Agent)	用户代理，表示在客户端使用的浏览器。
Cookie	Cookie，表示发送或接受的Cookie内容，如果没有Cookie，则显示短划线（-）。
referer	引用站点，表示用户访问的前一个站点。
cs-host	主机信息。
sc-status	协议返回状态，表示HTTP或FTP的操作状态。
sc-substatus	HTTP子协议的状态。
sc-win32-status	win32状态，表示操作状态。
sc-bytes	服务器发送的字节数。
cs-bytes	服务器接收的字节数。
time-taken	操作所花费的时间，单位：毫秒。

操作步骤

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。

3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。

   您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

5. 在搜索框中输入查询分析语句，单击**查询/分析**。

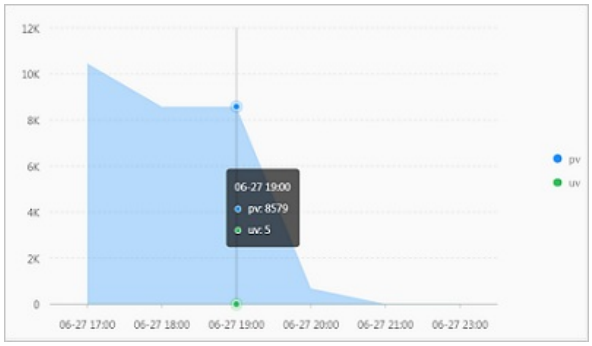
   查询分析语句详情请参见查询概述。

   - 通过IP地址来源分析访问地域，查询分析语句如下所示：

     ```
     | select ip_to_geo("c-ip") as country, count(1) as c group by ip_to_geo("c-ip") limit 100
     ```

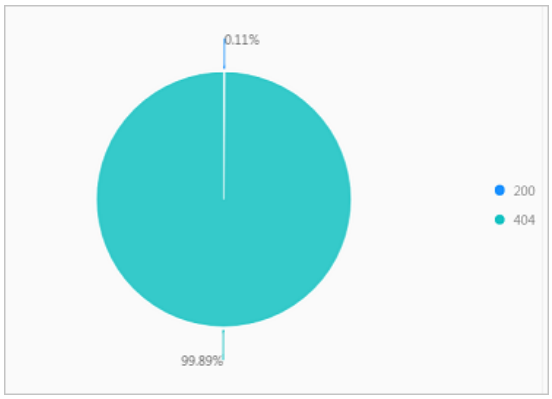
   - 统计最近的PV数和UV数，查询分析语句如下所示：

     ```
     *| select approx_distinct("c-ip") as uv ,count(1) as pv , date_format(date_trunc('hour', __time__), '%m-%d %H:%i') as time group by date_format(date_trunc('hour', __time__), '%m-%d %H:%i') order by time limit 1000
     ```

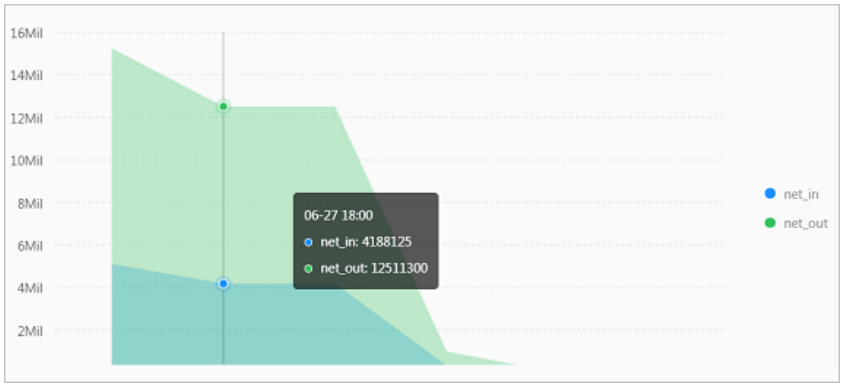
   - 统计HTTP请求状态码的占比，查询分析语句如下所示：

     ```
     *| select count(1) as pv ,"sc-status" group by "sc-status"
     ```

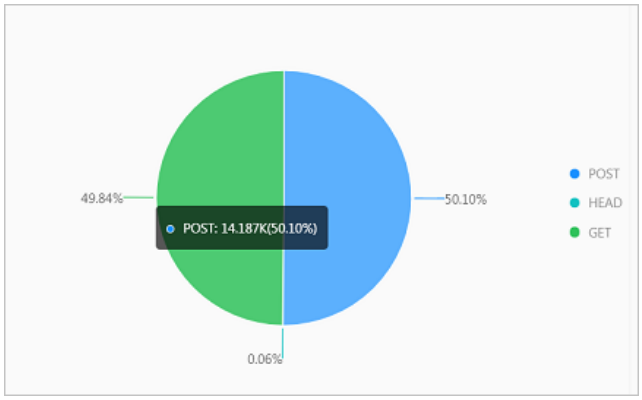
   - 统计流量的流入和流出情况，查询分析语句如下所示：

     ```
     *| select sum("sc-bytes") as net_out, sum("cs-bytes") as net_in ,date_format(date_trunc('hour', time), '%m-%d %H:%i') as time group by date_format(date_trunc('hour', time), '%m-%d %H:%i') order by time limit 10000
     ```



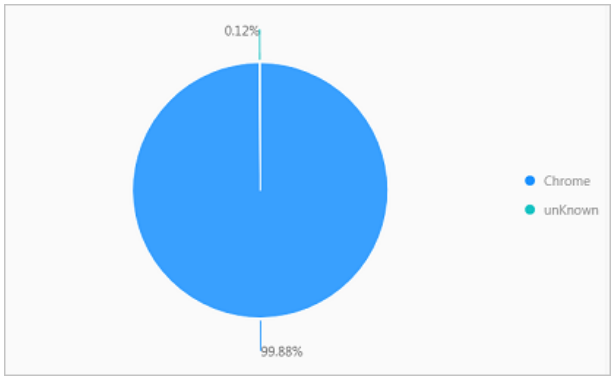
- 统计各种请求方法的占比，查询分析语句如下所示：

```
*| select count(1) as pv ,"cs-method" group by "cs-method"
```



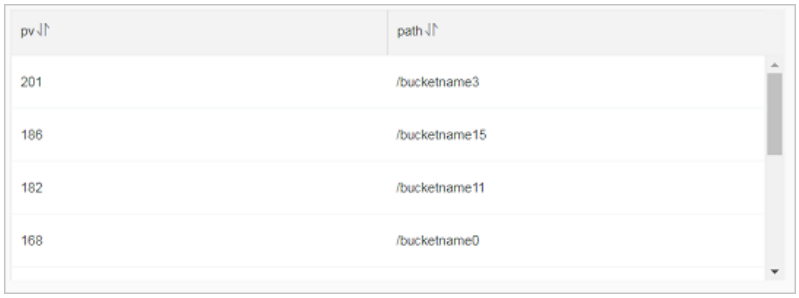
- 统计各种浏览器的占比，查询分析语句如下所示：

```
*| select count(1) as pv, case when "user-agent" like '%Chrome%' then 'Chrome' when "user-agen
t" like '%Firefox%' then 'Firefox' when "user-agent" like '%Safari%' then 'Safari' else 'unKno
wn' end as "user-agent" group by case when "user-agent" like '%Chrome%' then 'Chrome' when "us
er-agent" like '%Firefox%' then 'Firefox' when "user-agent" like '%Safari%' then 'Safari' else
'unKnown' end order by pv desc limit 10
```



- 统计访问数量前十的地址，查询分析语句如下所示：

```
*| select count(1) as pv, split_part("cs-uri-stem",'?',1) as path group by split_part("cs-uri-
stem",'?',1) order by pv desc limit 10
```



pv	path
201	/bucketname3
186	/bucketname15
182	/bucketname11
168	/bucketname0

19.11. 分析Log4j日志

本文以电商平台的日志为例，为您介绍Log4j日志的分析操作。

前提条件

- 已采集Log4j日志。更多信息，请参见采集Log4j日志。
- 已开启并配置索引。更多信息，请参见配置索引。

  本案例的索引如下图所示。

背景信息

Log4j是Apache的一个开放源代码项目，通过Log4j可以控制日志的优先级、输出目的地和输出格式。日志级别从高到低为ERROR、WARN、INFO、DEBUG，日志的输出目的地指定了将日志打印到控制台还是文件中，输出格式控制了输出的日志内容格式。

例如某电商公司，希望通过分析用户行为习惯数据（例如用户登录方式、上线的时间点及时长、浏览页面、页面停留时间、平均下单时间、消费水平等）、平台稳定性、系统报错、数据安全性等信息获取平台的最佳运营方案。针对此需求，日志服务提供一站式数据采集与分析功能，帮助客户存储并分析日志。日志服务采集到的日志样例如下所示。

- 记录用户登录行为的日志：

```
level:  INFO
location:  com.aliyun.log4jappendertest.Log4jAppenderBizDemo.login(Log4jAppenderBizDemo.java:38)
message:  User login successfully. requestID=id4 userID=user8
thread:  main
time:  2018-01-26T15:31+0000
```

- 记录用户购买行为的日志：

```
level:  INFO
location:  com.aliyun.log4jappendertest.Log4jAppenderBizDemo.order(Log4jAppenderBizDemo.java:46)
message:  Place an order successfully. requestID=id44 userID=user8 itemID=item3 amount=9
thread:  main
time:  2018-01-26T15:31+0000
```

操作步骤

1. 登录日志服务控制台。

2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。
   您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1 min以内的误差。

5. 在搜索框中输入查询分析语句，单击**查询/分析**。
   查询分析语句语法，请参见基本语法。
   - 统计最近1小时发生错误最多的3个位置。

     ```
     level: ERROR | select location ,count(*) as count GROUP BY  location  ORDER BY count DESC LIMI
     T 3
     ```

   - 统计最近15分钟各种日志级别产生的日志条数。

     ```
     | select level ,count(*) as count GROUP BY level ORDER BY count DESC
     ```

   - 统计最近1小时登录次数最多的三个用户。

     ```
     login | SELECT regexp_extract(message, 'userID=(?<userID>[a-zA-Z\d]+)', 1) AS userID, count(*)
     as count GROUP BY userID ORDER BY count DESC LIMIT 3
     ```

   - 统计最近15分钟每个用户的付款总额。

     ```
     order | SELECT regexp_extract(message, 'userID=(?<userID>[a-zA-Z\d]+)', 1) AS userID, sum(cast
     (regexp_extract(message, 'amount=(?<amount>[a-zA-Z\d]+)', 1) AS double)) AS amount GROUP BY us
     erID
     ```

19.12. 查询分析程序日志

本文通过查询、关联分析、统计分析等场景介绍如何使用日志服务对程序日志进行查询和分析。

背景信息

程序日志内容全、存在一定共性，它是运维程序的重要信息，但程序日志具有如下不便于存储与分析的特性：

- 格式随意：不同开发者的代码风格不同，对应的日志风格也不同，难以统一。
- 数据量大：程序日志一般比访问日志大1个数量级。
- 分布的服务器多：大部分应用为无状态模式，运行在不同框架中，例如云服务器、容器服务等，对应的实例数从几个到数千个，需要有一种跨服务器的日志采集方案。
- 运行环境复杂：程序运行在不同的环境中，产生的日志也保存在不同的环境中，例如应用相关的日志在容器中、API相关日志在FunctionCompute中、旧系统日志在本地IDC中、移动端相关日志在用户处、网页端日志在浏览器中等。

为了能够获得全量日志，必须把所有日志统一存储。针对该场景，日志服务提供多样化的日志采集方式及一站式分析功能，您可通过查询+SQL92语法对日志进行实时分析，并以图表形式直观展示分析结果。和开源方案对比，日志服务提供的解决方案在查询分析成本上仅是开源方案的25%。

查询程序日志

例如某App出现订单错误或请求延时等问题，您可以通过查询语句在TB级数据量的日志中快速（1s内）定位问题。还可以根据业务需求，设置时间范围、查询关键字等信息，更精准地返回查询结果。

- 查询延时大于1s，并且请求方法是以Post开头的请求数据。

  ```
  Latency > 1000000 and Method=Post*
  ```

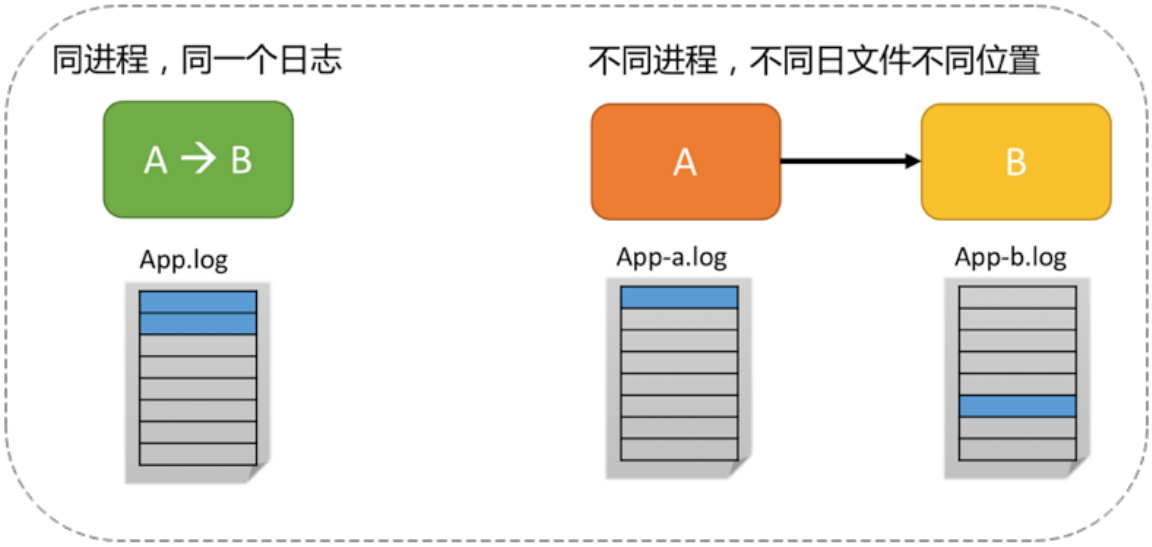
- 查找包含error关键词但不包含merge关键词的日志。

```
error not merge
```

关联分析程序日志

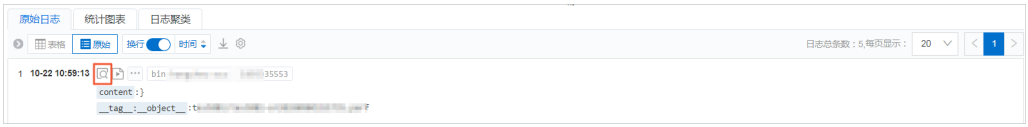
关联分析包括进程内关联与跨进程关联，区别如下：

- 进程内关联：一般比较简单，因为同一个进程前后日志都在一个文件中。在多线程环节中，只需根据线程ID进行过滤即可。
- 跨进程关联：跨进程的请求一般没有明确线索，一般通过RPC中传入的TracerId来进行关联。

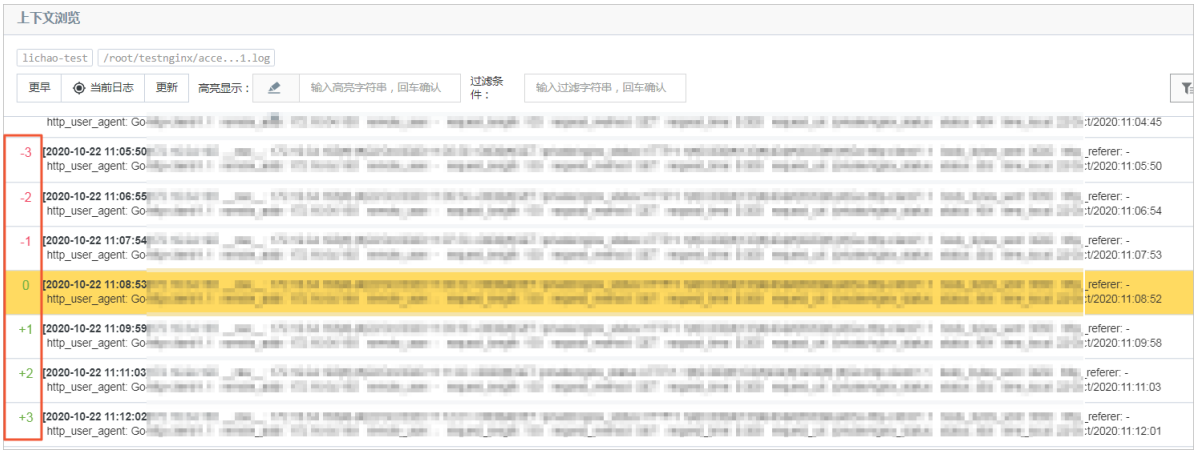


- 进程内关联

  通过上下文查询查看关联日志。例如通过关键词查询定位到一个异常日志，然后单击**上下文浏览**，查看该日志前后N条日志，操作步骤请参见上下文查询。



  上下文查询结果如下所示：

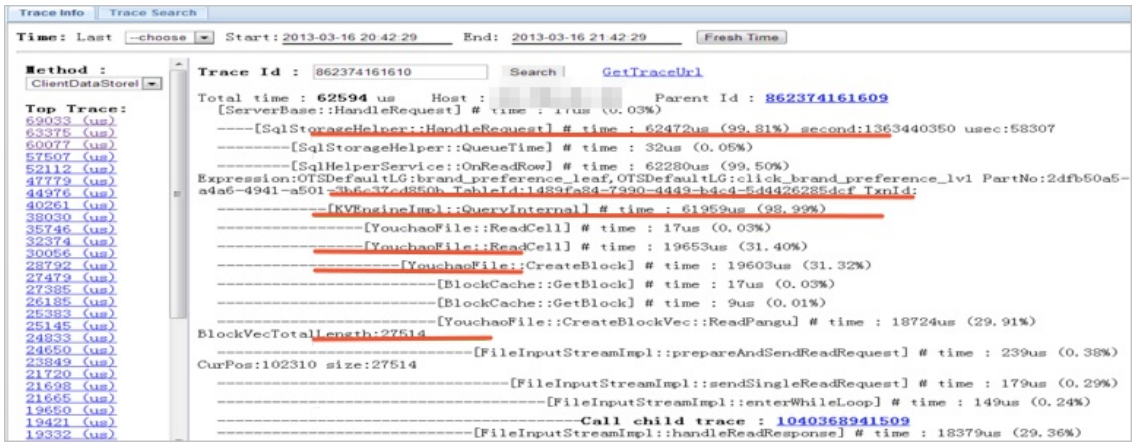


- 跨进程关联

  跨进程关联也叫Tracing，比较常见的工具有鹰眼、Dapper、StackDriver Trace、Zipkin、Appdash、X-ray等。

此处基于日志服务，实现基本的Tracing功能。您可以在各模块日志中输出Request_id、OrderId等可以关联的标示字段，通过在不同的日志库中查找，获取所有相关日志。

例如通过SDK查询前端机、后端机、支付系统、订单系统等日志。获得结果后，制作一个前端页面将跨进程分析关联起来，如下图所示。

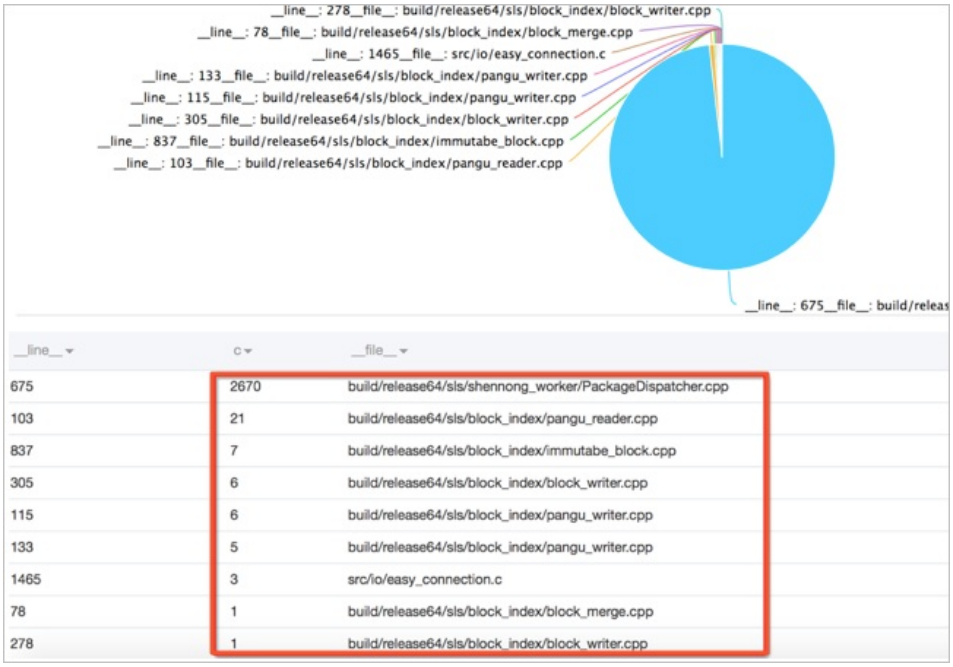


统计分析程序日志

查询到日志后，您还可以做进一步统计分析。

执行如下查询分析语句，统计所有错误发生的类型和位置的分布。

```
__level__:error | select __file__, __line__, count(*) as c group by __file__, __line__ order by c desc
```



__line__	c	__file__
675	2670	build/release64/sls/shennong_worker/PackageDispatcher.cpp
103	21	build/release64/sls/block_index/pangu_reader.cpp
837	7	build/release64/sls/block_index/immutabe_block.cpp
305	6	build/release64/sls/block_index/block_writer.cpp
115	6	build/release64/sls/block_index/pangu_writer.cpp
133	5	build/release64/sls/block_index/pangu_writer.cpp
1465	3	src/io/easy_connection.c
78	1	build/release64/sls/block_index/block_merge.cpp
278	1	build/release64/sls/block_index/block_writer.cpp

相关操作

- 备份日志

  将日志备份至OSS、MaxCompute等产品中。

- 关键词告警
  - 通过日志服务告警
  - 通过云监控告警
- 日志查询权限分配管理

  可以通过RAM用户授权方法隔离开发、PE等权限。

19.13. 分析网站日志

日志服务支持通过SQL92语法分析日志，并提供丰富的统计图表（例如表格、线图、柱状图、饼图、流图、地图等）展示分析结果。本文介绍如何在日志服务控制台上分析网站日志，并通过合适的统计图表可视化展示分析结果。

前提条件

- 已采集网站日志。更多信息，请参见数据采集。
- 已配置索引。更多信息，请参见配置索引。

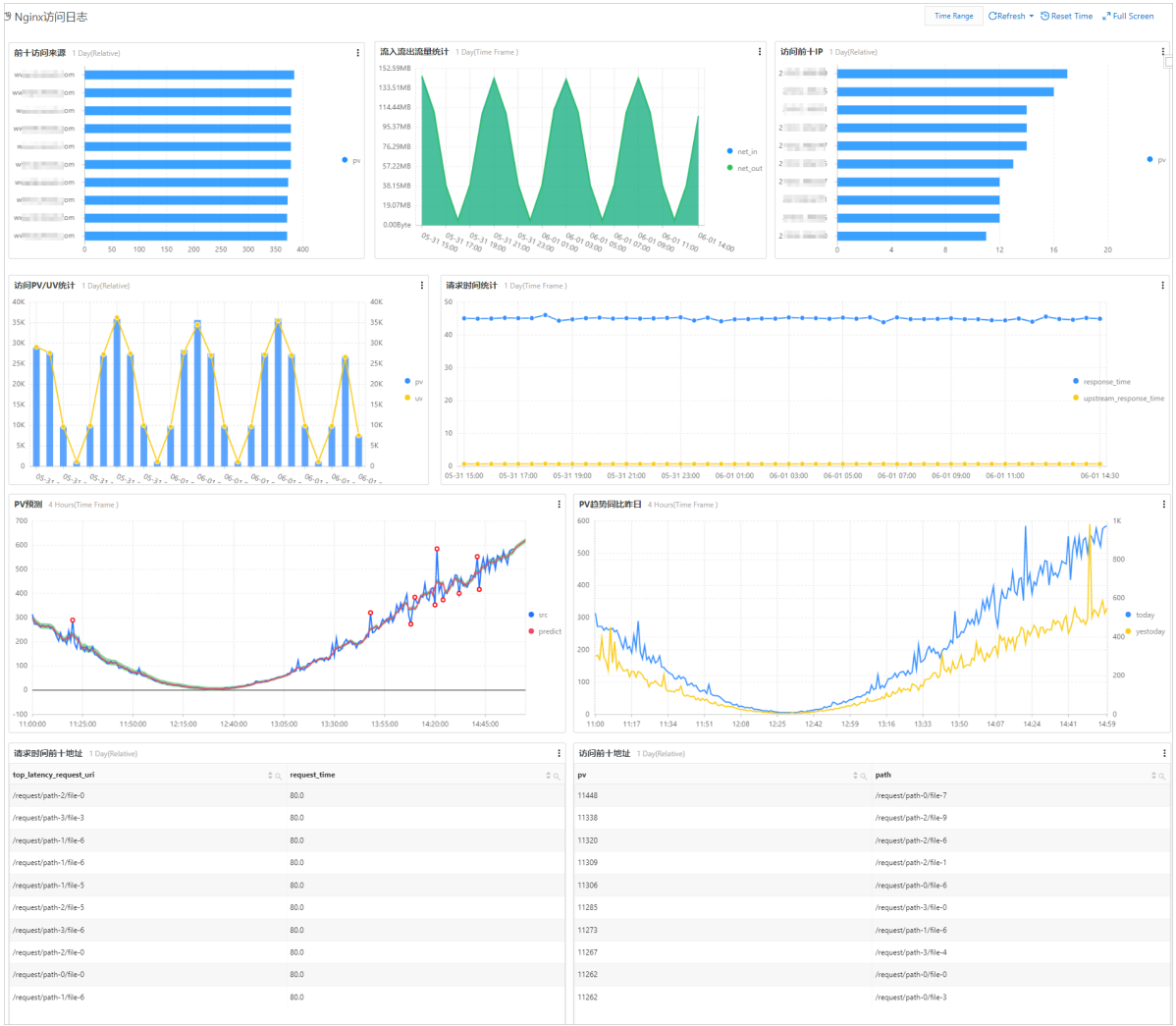
背景信息

网站日志是运维网站的重要信息，包含PV、UV、访问地域分布以及访问前十页面等信息。日志服务提供多样化的日志采集方式及一站式分析功能，您可通过查询+SQL92语法对日志进行实时分析，并以图表形式直观展示分析结果。日志服务还支持通过自带的仪表盘、DataV、Grafana、Tableua（通过JDBC链接）、Quick BI等可视化方式创建多种场景下的日志数据分析大盘。

功能体验

- 通过测试仪表盘体验查询分析功能。

  试用链接：仪表盘



- 通过数据实验室功能体验查询分析功能。更多信息，请参见使用数据实验室。

操作步骤

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**页签中，单击目标Logstore。
4. 在页面右上角，单击**15分钟（相对）**，设置查询的时间范围。

   您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询结果相对于指定的时间范围来说，有1min以内的误差。

5. 在搜索框中输入查询分析语句，单击**查询/分析**。

   日志服务提供丰富的统计图表，用于展示查询分析结果。更多信息，请参见统计图表。

   - 通过表格展示最近1天客户端（remote_addr）访问情况，并降序排列。

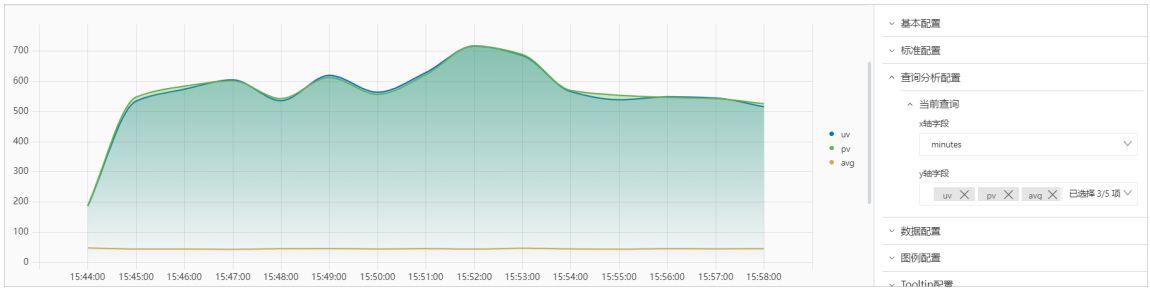
     ```
     * | SELECT remote_addr, count(*) as count GROUP BY remote_addr ORDER BY count DESC
     ```

remote_addr	count
[redacted]1	14
[redacted]7	13
[redacted]	13
[redacted]8	12
[redacted]3	12
[redacted]5	12
[redacted]4	12
[redacted]5	12

   - 通过线图展示最近15分钟PV、UV以及平均响应时间的变化情况。

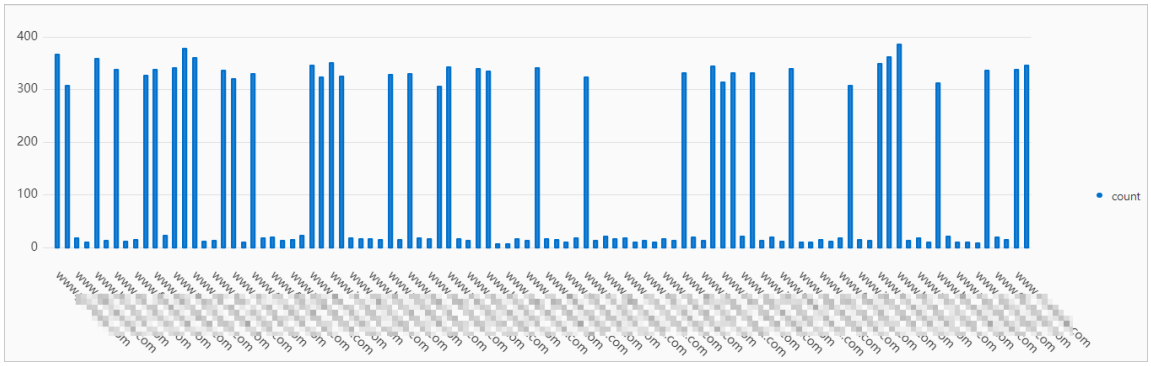
     ```
     * | select date_format(from_unixtime(__time__ - __time__% 60), '%H:%i:%S') as minutes, approx_distinct(remote_addr) as uv, count(1) as pv, avg(request_time) as avg group by minutes order by minutes asc limit 100000
     ```

     在**查询分析配置**中，设置**X轴字段**为*minutes*，**y轴字段**为*pv*、*uv*和*avg*，统计图表如下所示。

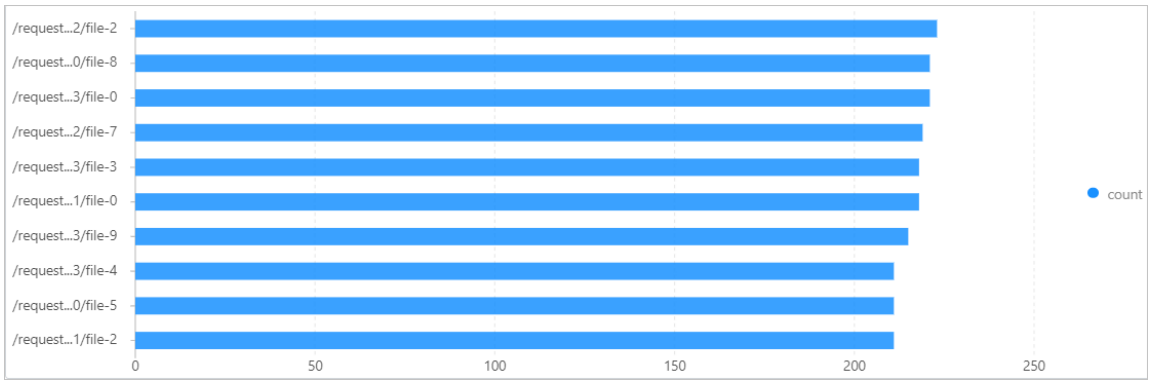
   - 通过柱状图展示最近15分钟不同来源地址（referer）的访问次数。

     ```
     * | select referer, count(1) as count group by referer
     ```



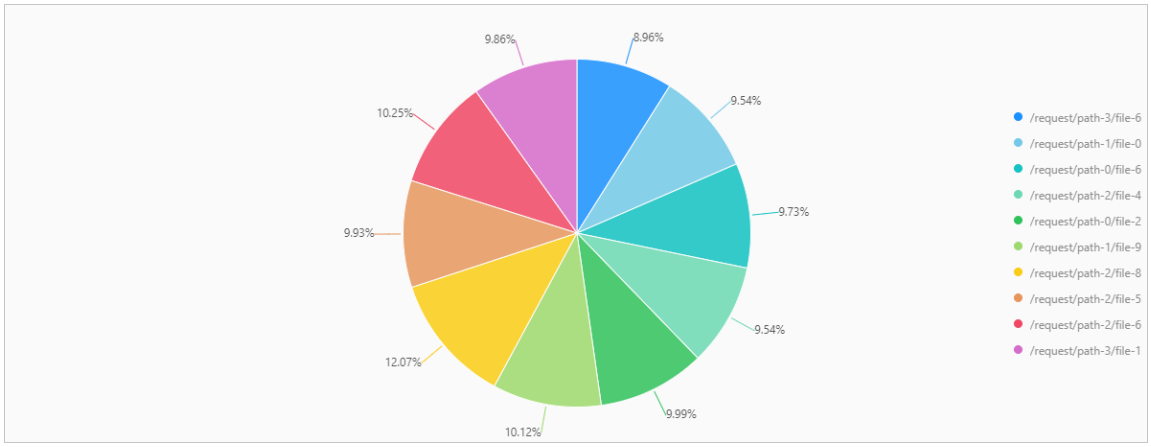
- 通过条形图展示最近15分钟访问前十的页面（request_uri）。

```
* | select  request_uri, count(1) as count group by request_uri order by count desc limit 10
```



- 通过饼图展示最近15分钟页面（request_uri）访问情况。

```
* | select request_uri as uri , count(1) as c group by uri limit 10
```



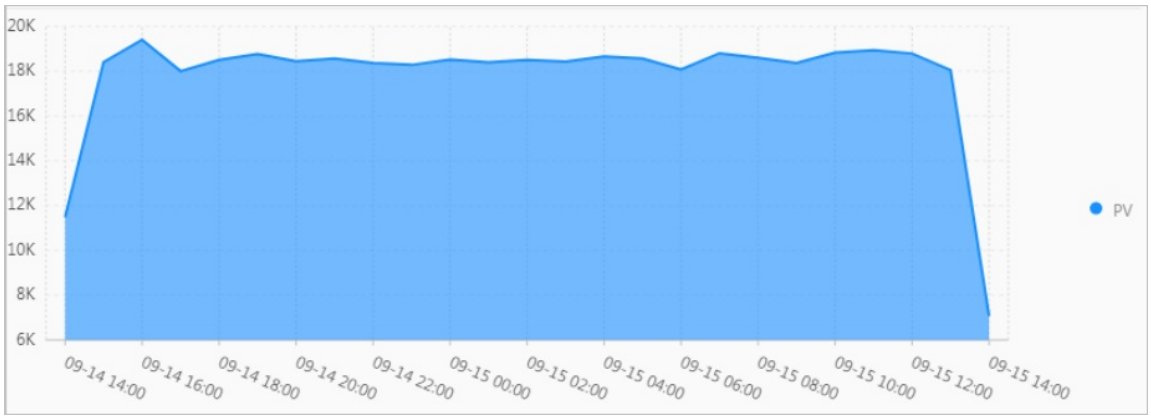
- 通过单值图展示最近15分钟的PV数。

```
* | select count(1) as PV
```

- 通过面积图展示最近1天某IP地址的访问情况。

```
remote_addr: 10.0.XX.XX | select date_format(date_trunc('hour', __time__), '%m-%d %H:%i') as t
ime, count(1) as PV group by time order by time limit 1000
```

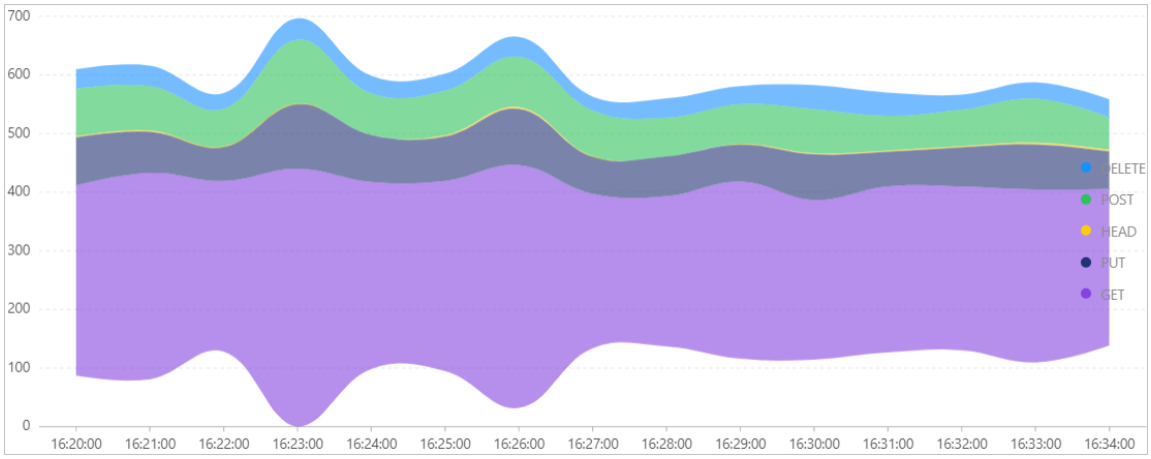
配置**X轴**为*time*，**Y轴**为*PV*，统计图表如下所示。



- 通过流图展示最近15分钟不同方法（request_method）的请求次数随时间的变化趋势。

```
* | select date_format(from_unixtime(__time__ - __time__% 60), '%H:%i:%S') as minute, count(1)
as c, request_method group by minute, request_method order by minute asc limit 100000
```

配置**X轴**为*minute*，**Y轴**为*c*，**聚合列**为*request_method*，统计图表如下所示。



6. 添加统计图表到仪表盘。

   您可以单击**添加到仪表盘**，完成操作。更多信息，请参见添加统计图表到仪表盘。

19.14. 分析负载均衡7层访问日志

本文档基于日志服务的可视化和实时查询分析能力，为您介绍负载均衡（SLB）7层访问日志查询分析的典型案例。

前提条件

已采集到SLB 7层负载均衡日志，详情请参见开通访问日志功能。

背景信息

对于大部分云上架构而言，负载均衡是基础设施组件，对SLB持续的监控、探测、诊断和报告是一个强需求。阿里云SLB是对多台云服务器进行流量分发的负载均衡服务，可以通过流量分发扩展应用系统对外的服务能力。通过消除单点故障，为应用提供大规模、高可靠的并发Web访问支撑。

SLB访问日志功能当前支持基于HTTP/HTTPS的7层负载均衡，访问日志内容丰富，完整字段说明请参见日志字段详情。SLB典型指标如下所示：

- PV：客户端发起HTTP、HTTPS请求的次数。
- UV：对于相同客户端只计算一次，合计总体请求次数。
- 请求成功率：状态码为2XX的请求次数占总PV的比例。
- 请求报文流量：客户端请求报文长度总和。
- 返回客户端流量：SLB返回给客户端的HTTP Body字节数总和。
- 请求的热点分布：统计客户端地理位置，按照地理位置统计每个地域的PV情况。

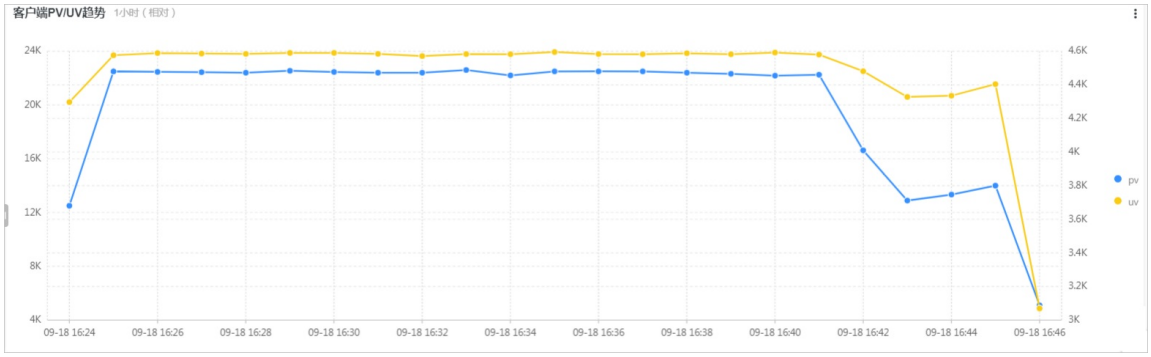
常用分析

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**日志存储 > 日志库**中，单击目标Logstore左侧的>。
4. 在**可视化仪表盘**中，单击目标仪表盘。

   目标仪表盘包括slb-user-log-slb_layer7_operation_center_cn和slb-user-log-slb_layer7_access_center_cn。

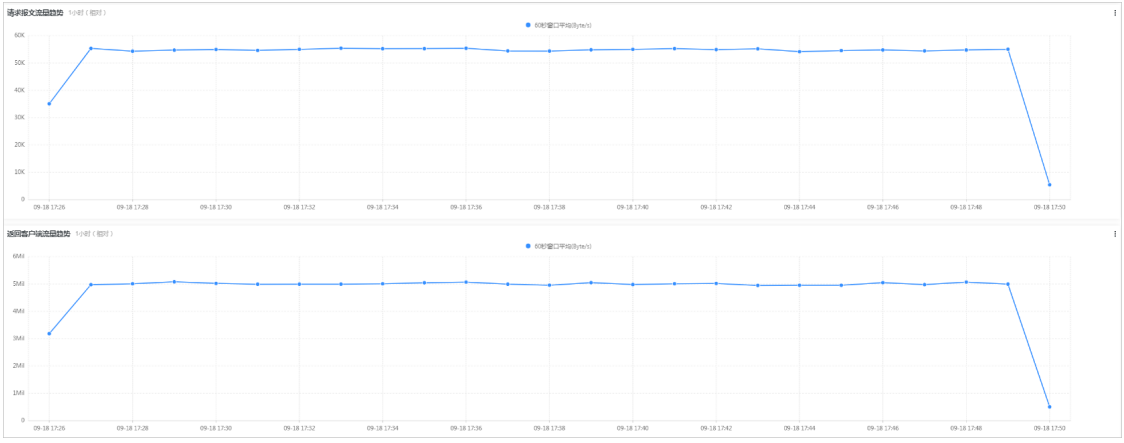
   - 业务概览

     通过过滤器筛选某指定SLB实例的PV、UV随时间的变化趋势。过滤器操作请参见添加过滤器。

   - 分析流量与延迟情况

- 统计一定时间范围内请求报文的流量和返回客户端的流量。



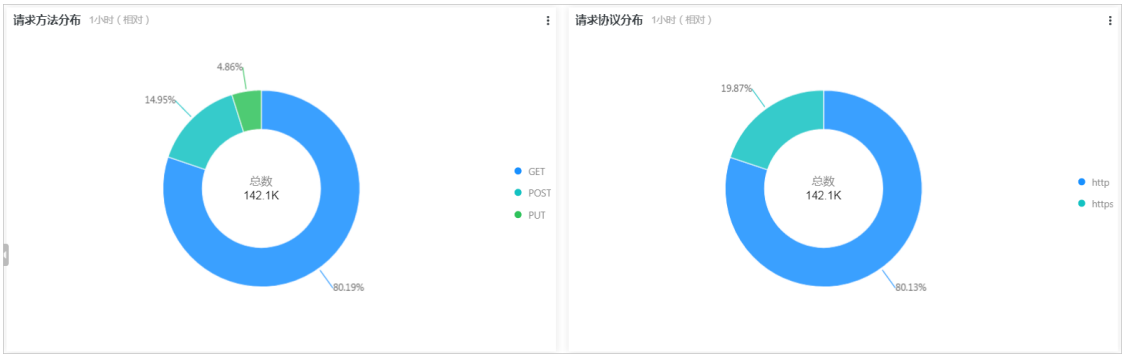
- 统计一定时间范围内请求的响应时间变化趋势和upstream响应时间变化趋势。

- 统计一定时间范围内高延迟的请求。

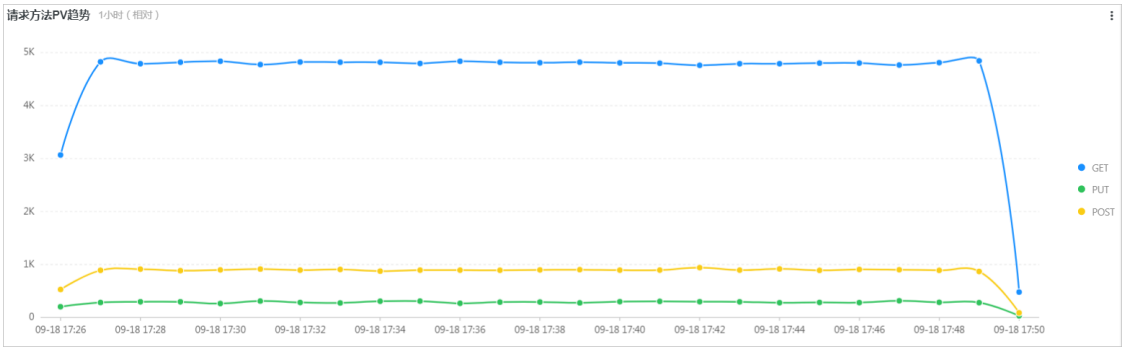
top upstream响应时间 1小时（相对）

SLB实例ID	后端服务器	平均upstream响应时间(s)	pv	请求报文流量(MB)	返回客户端流量(MB)	2xx比例(%)	3xx比例(%)	4xx比例(%)	5xx比例(%)
null	1 19:80	0.6475	24	0.01	1.23	83.333333	0.0	12.5	4.166667
null	1 17:80	0.645167	24	0.01	1.25	83.333333	0.0	16.666667	0.0
null	1 11:80	0.6446	25	0.01	1.01	88.0	0.0	12.0	0.0
null	1 19:80	0.639962	26	0.01	1.24	80.769231	0.0	11.538462	7.692308
null	1 5:80	0.638	32	0.02	1.47	87.5	0.0	9.375	3.125
null	1 10:80	0.636417	24	0.01	0.94	66.666667	0.0	25.0	8.333333
null	1 17:80	0.635609	23	0.01	1.25	82.608696	0.0	8.695652	8.695652
null	1 0:80	0.635481	27	0.02	1.41	92.592593	0.0	3.703704	3.703704
null	1 19:80	0.634474	19	0.01	0.9	68.421053	0.0	31.578947	0.0

- 分析用户请求情况
  - 统计一定时间范围内的请求方法、请求协议分布情况。



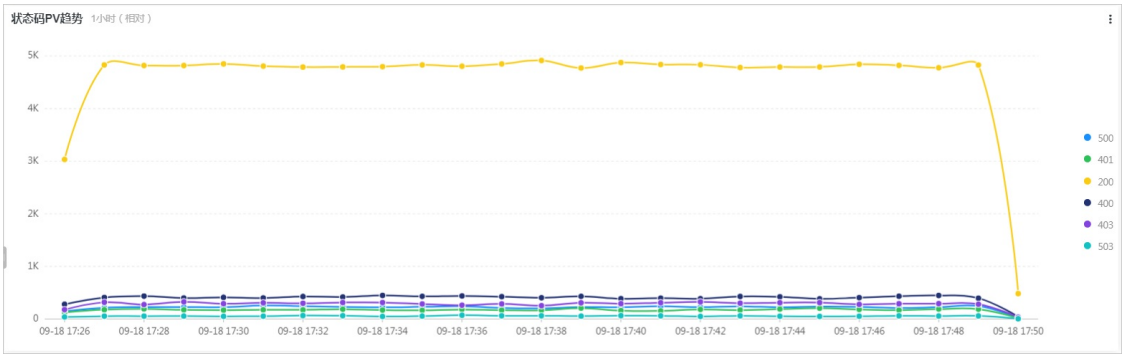
- 统计一定时间范围内各种请求方法的PV趋势。



- 统计一定时间范围内服务运行情况。

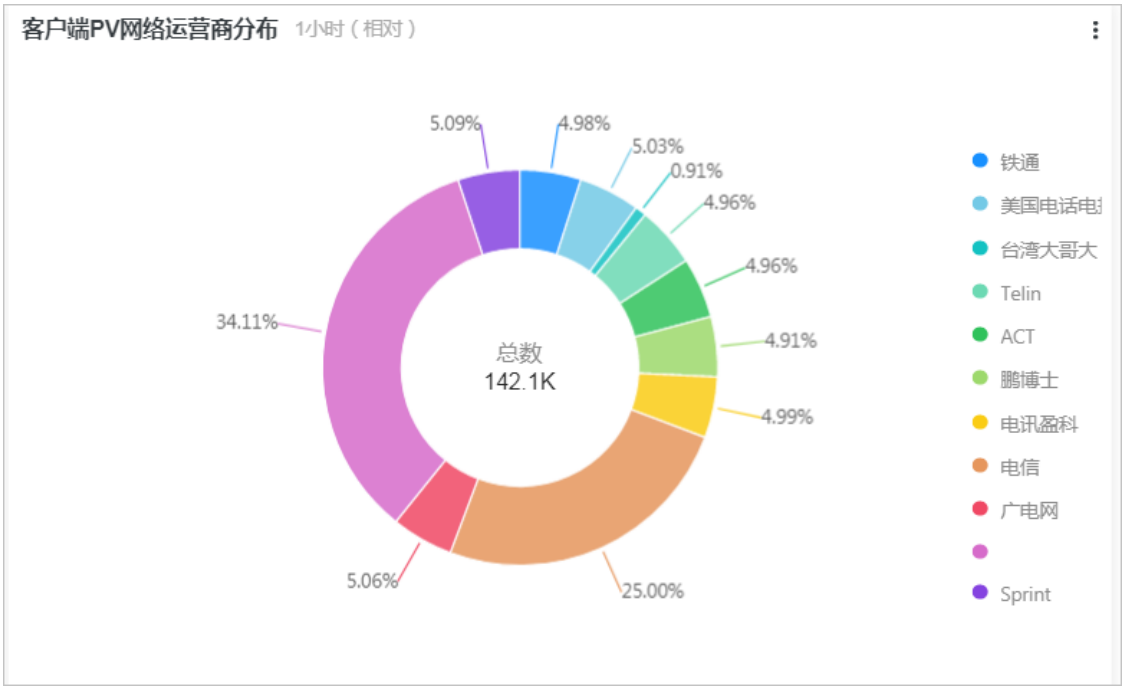
  如果出现大量的500状态码则表示后端RealServer的应用程序发生内部错误。

- 统计一定时间范围内各种状态码的PV变化趋势。



- 分析请求源

- 统计客户端所属的网络运营商分布情况。



- 统计客户端所在的地理位置（国家、省份、城市）。

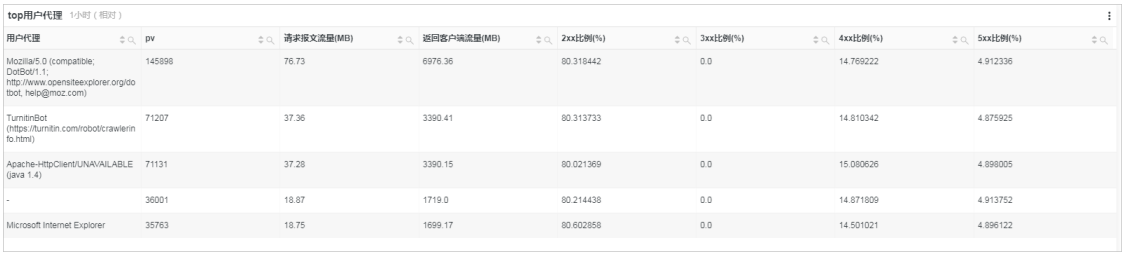
top客户端 1小时（相对）

客户端ip	pv	区域	城市	运营商	请求报文流量(MB)	返回客户端流量(MB)
[illegible]	110	福建省	厦门市	广电网	0.05	5.21
[illegible]	109	-1	-1		0.06	4.99
[illegible]	108	北京市	北京市		0.06	5.25
[illegible]	108	北京市	北京市	电信	0.06	5.55
[illegible]	107	塔萨斯州	-1	Sprint	0.05	5.31
[illegible]	106	-1	-1		0.06	5.35
[illegible]	105	福建省	厦门市	广电网	0.06	5.05
[illegible]	105	广东省	中山市	电信	0.06	5.14
[illegible]	105	山东省	济宁市	电信	0.06	4.66

总数：100 < 1 / 5 >

- 查看用户代理信息。

  通过用户代理（http_user_agent）可得知哪些用户在访问网站或服务。例如搜索引擎会使用爬虫机器人扫描或下载网站资源，一般情况下低频爬虫访问可以帮助搜索引擎及时更新网站内容，有助于网站的推广和SEO。但如果高PV的请求都来自于爬虫，则可能影响服务性能及浪费机器资源。



top用户代理 1小时（相对）

用户代理	pv	请求报文流量(MB)	返回客户端流量(MB)	2xx比例(%)	3xx比例(%)	4xx比例(%)	5xx比例(%)
Mozilla/5.0 (compatible; DotBot/1.1; http://www.opensiteexplorer.org/dotbot, help@moz.com)	145898	76.73	6976.36	80.318442	0.0	14.769222	4.912336
TurnitinBot (https://turnitin.com/robot/crawlerinfo.html)	71207	37.36	3390.41	80.313733	0.0	14.810342	4.875925
Apache-HttpClient/UNAVAILABLE (java 1.4)	71131	37.28	3390.15	80.021369	0.0	15.080626	4.898005
-	36001	18.87	1719.0	80.214438	0.0	14.871809	4.913752
Microsoft Internet Explorer	35763	18.75	1699.17	80.602858	0.0	14.501021	4.896122

- 运营概览

运营人员可基于SLB访问日志分析流量情况，进而辅助业务决策。例如通过分析Host和URI信息获知访客最关注的内容，为网站内容建设提供有力的参考。

top host 1小时（相对）

SLB实例ID	host	pv	请求报文流量(MB)	返回客户端流量(MB)	2xx比例(%)	3xx比例(%)	4xx比例(%)	5xx比例(%)
[illegible]	www.[illegible]ain.com	11430	6.01	549.35	80.437445	0.0	14.724409	4.838145
[illegible]	www.[illegible]ain.com	11411	5.99	547.68	80.334765	0.0	14.95925	4.705985
[illegible]	www.[illegible]ain.com	11382	6.04	546.72	80.416447	0.0	14.558074	5.025479
[illegible]	www.[illegible]ain.com	11363	5.97	541.98	79.908475	0.0	15.154449	4.937076
[illegible]	www.[illegible]ain.com	11349	6.0	538.69	79.857256	0.0	15.261256	4.881487
[illegible]	www.[illegible]ain.com	11321	5.89	541.37	80.487589	0.0	15.104673	4.407738
[illegible]	www.[illegible]ain.com	11283	5.95	538.88	80.643446	0.0	14.570593	4.785961
[illegible]	www.[illegible]ain.com	11222	5.86	534.48	79.6293	0.0	15.380503	4.990198
[illegible]	www.[illegible]ain.com	11202	5.89	535.41	80.985538	0.0	14.434922	4.579539

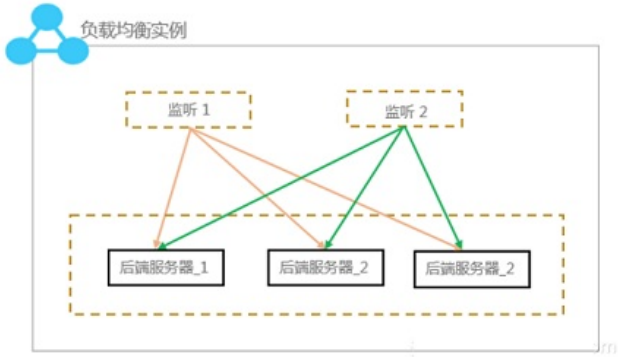
总数：80 < 1 / 4 >

top uri 1小时（相对）

host	请求uri	pv	请求报文流量(MB)	返回客户端流量(MB)	2xx比例(%)	3xx比例(%)	4xx比例(%)	5xx比例(%)
[illegible]ain.com	/mock/ref-uri/y4	432	0.23	21.21	80.092593	0.0	14.814815	5.092593
[illegible]ain.com	/mock/ref-uri/y0	423	0.22	21.1	82.978723	0.0	12.056738	4.964539
[illegible]ain.com	/mock/ref-uri/x7	422	0.23	20.16	82.227488	0.0	11.848341	5.924171
[illegible]ain.com	/mock/ref-uri/x4	421	0.22	19.99	80.047506	0.0	14.726841	5.225653
[illegible]ain.com	/mock/ref-uri/z4	418	0.22	20.42	80.861244	0.0	14.354067	4.784689
[illegible]ain.com	/mock/ref-uri/x4	416	0.22	19.49	81.971154	0.0	13.221154	4.807692

请求调度分析

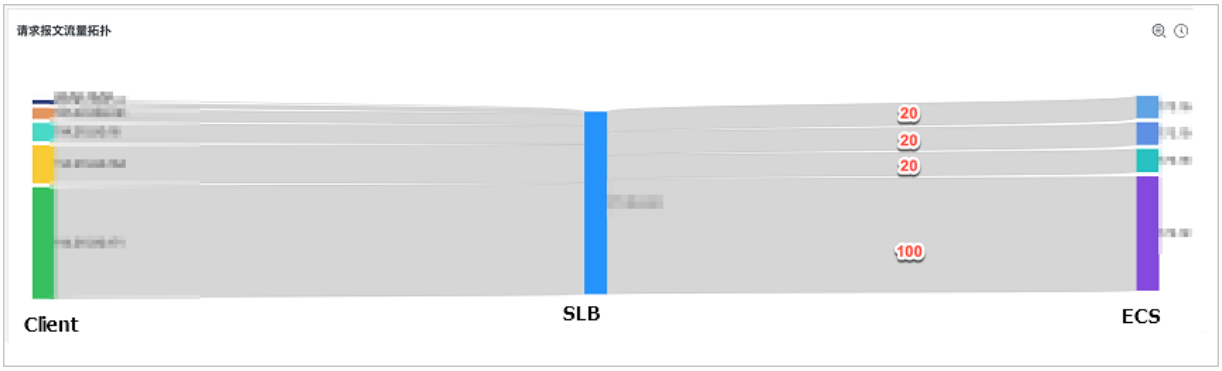
客户端流量会先被SLB处理，分发到其中一台RealServer中进行实际的业务逻辑处理。SLB可自动检测到不健康的机器并重新分配流量到其它正常服务的RealServer上，等异常机器恢复后再重新分配流量。



为SLB实例添加一个监听，例如服务器（192.168.0.0）同时兼有跳板机职能，其性能是其它三台服务器的4倍，为该服务器设置监听权重为100，其余服务器监听权重为20。执行如下查询分析语句分析请求流量分布情况。

```
* | select COALESCE(client_ip, vip_addr, upstream_addr) as source, COALESCE(upstream_addr, vip_addr, client_ip) as dest, sum(request_length) as inflow group by grouping sets( (client_ip, vip_addr), (vip_addr, upstream_addr))
```

桑基图展示每台RealServer的负载情况，多个客户端向SLB发起请求，请求报文流量基本遵循20:20:20:100比例转发到后端RealServer中。



19.15. 分页显示查询分析结果

查询分析日志时，查询分析结果内容过多会影响显示速度和查询体验。日志服务提供分页功能，可控制每次返回的日志数量。本文介绍查询结果和分析结果的分页方法。

分页方式

日志服务查询和分析功能支持在查询分析语句中同时实现关键字查询和查询结果的SQL分析。您可以调用GetLogs API，根据关键字查询日志原始内容，也可以进行查询结果的SQL计算，获取分析结果。查询结果和分析结果使用不同的分页方法。 更多信息，请参见GetLogs。

- 查询语句：使用关键字查询，获取原始日志内容。您可以通过GetLogs API中的offset和line参数实现分页。更多信息，请参见查询概述。
- 分析语句：使用SQL对查询结果进行分析，获取统计结果。您可以通过SQL中的LIMIT语法实现分页。更多信息，请参见分析简介和LIMIT子句。

查询结果分页

GetLogs API中的offset、line参数说明如下：

- offset：指定从某一行开始读取查询结果。
- line：指定当前请求读取的行数，最大值为100。如果大于100，则仍然返回100行。

在分页读取时，不停地增大offset的值，直到读取到某个offset值后，获取的结果行数为0，并且结果的progress为complete状态，则表示读取了所有数据。

- 分页的示例代码逻辑

```
offset = 0                          //从第0行开始读取。
line = 100                          //每次读取100行。
query = "status:200"                //查询status字段是200的所有日志。
while True:
    response = get_logstore_logs(query, offset, line) //执行读取请求。
    process (response)                                //调用自定义逻辑，处理返回结果。
    如果 response.get_count() == 0 && response.is_complete()
        则读取结束，跳出当前循环
    否则
       offset += 100                          //offset增加到100，读取下一个100行。
```

- Python代码示例

  更多信息，请参见Python SDK概述。

```
endpoint = ''       //日志服务的域名。更多信息，请参见服务入口。
accessKeyId = ''    //阿里云访问密钥AccessKey ID。更多信息，请参见访问密钥。阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维。
accessKey = ''      //阿里云访问密钥AccessKey Secret。
project = ''        //Project项目名称。
logstore = ''       //Logstore名称。
client = LogClient(endpoint, accessKeyId, accessKey)
topic = ""
From = int(time.time()) - 600
To = int(time.time())
log_line = 100
offset = 0
while True:
    res4 = None
for retry_time in range(0, 3):
        req4 = GetLogsRequest(project, logstore, From, To, topic=topic, line=log_line, offset=offs
et)
        res4 = self.client.get_logs(req4)
        if res4 is not None and res4.is_completed():
            break
        time.sleep(1)
        offset += 100
        if res4.is_completed() and res4.get_count() == 0:
          break;
        if res4 is not None:
        res4.log_print()  //处理结果。
```

- Java示例

  更多信息，请参见Java SDK概述。

```
int log_offset = 0;
int log_line = 100;   //log_line的最大值为100，每次获取100行数据。若需要读取更多数据，请使用offset分页。of
fset和line只对关键字查询有效，若使用SQL查询，则无效。在SQL查询中返回更多数据，请使用limit语法。
while (true) {
    GetLogsResponse res4 = null;
    //对于每个Offset，一次读取100行日志，如果读取失败，最多重复读取3次。
    for (int retry_time = 0; retry_time < 3; retry_time++) {
          GetLogsRequest req4 = new GetLogsRequest(project, logstore, from, to, topic, query, log_
offset,
               log_line, false);
          res4 = client.GetLogs(req4);
          if (res4 != null && res4.IsCompleted()) {
                    break;
               }
               Thread.sleep(200);
            }
            System.out.println("Read log count:" + String.valueOf(res4.GetCount()));
            log_offset += log_line;
            if (res4.IsCompleted() && res4.GetCount() == 0) {
                       break;
            }
 }
```

分析结果分页

您可以使用SQL中的Limit语法实现分析结果分析显示，示例如下所示：

```
limit Offset, Line
```

参数说明如下所示：

- offset：指定从某一行开始读取分析结果。
- line：指定当前请求读取的行数，最大值为1,000,000。如果一次读取太多，会影响网络延时和客户端的处理速度。

例如，通过 `* | select count(1) , url group by url` 语句进行查询分析，指定返回2000行日志。您可以通过分页指定每次读取500行，共4次读取完成，示例如下：

```
* | select count(1) , url  group by url  limit 0, 500
* | select count(1) , url  group by url  limit 500, 500
* | select count(1) , url  group by url  limit 1000, 500
* | select count(1) , url  group by url  limit 1500, 500
```

- 分析结果分页的示例代码逻辑

```
offset = 0     //从第0行开始读取。
line = 500    //每次读取500行。
query = "* | select count(1) , url  group by url  limit "
while True:
real_query = query + offset + "," +  line
response = get_logstore_logs(real_query) //执行读取请求。
process (response)                       //调用自定义逻辑，处理返回的结果。
如果 response.get_count() == 0
    则读取结束，跳出当前循环
否则
    offset += 500                        //offset增加到500，读取下一个500行。
```

- Python代码示例

  更多信息，请参见Python SDK概述。

```
endpoint = ''      //日志服务的域名。更多信息，请参见服务入口。
accessKeyId = ''   //阿里云访问密钥AccessKey ID。更多信息，请参见访问密钥。阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维。
accessKey = ''     //阿里云访问密钥AccessKey Secret。
project = ''       //Project名称。
logstore = ''      //Logstore名称。
client = LogClient(endpoint, accessKeyId, accessKey)
topic = ""
origin_query = "* | select *  limit "
From = int(time.time()) - 600
To = int(time.time())
log_line = 100
offset = 0
while True:
    res4 = None
    query = origin_query + str(offset) + " , " + str(log_line)
    for retry_time in range(0, 3):
        req4 = GetLogsRequest(project, logstore, From, To, topic=topic, query)
        res4 = self.client.get_logs(req4)
        if res4 is not None and res4.is_completed():
            break
        time.sleep(1)
        offset += 100
        if res4.is_completed() and res4.get_count() == 0:
          break;
        if res4 is not None:
        res4.log_print()  //处理结果。
```

- Java代码示例

更多信息，请参见Java SDK概述。

```
int log_offset = 0;
int log_line = 500;
String origin_query = "* | select count(1) , url  group by url  limit "
while (true) {
            GetLogsResponse res4 = null;
            //对于每个Offset，一次读取500行日志。如果读取失败，最多重复读取3次。
            query = origin_query + log_offset + "," + log_line;
            for (int retry_time = 0; retry_time < 3; retry_time++) {
                GetLogsRequest req4 = new GetLogsRequest(project, logstore, from, to, topic, query
);
                res4 = client.GetLogs(req4);
                if (res4 != null && res4.IsCompleted()) {
                    break;
                }
                Thread.sleep(200);
            }
            System.out.println("Read log count:" + String.valueOf(res4.GetCount()));
            log_offset += log_line;
            if (res4.GetCount() == 0) {
                        break;
            }
}
```

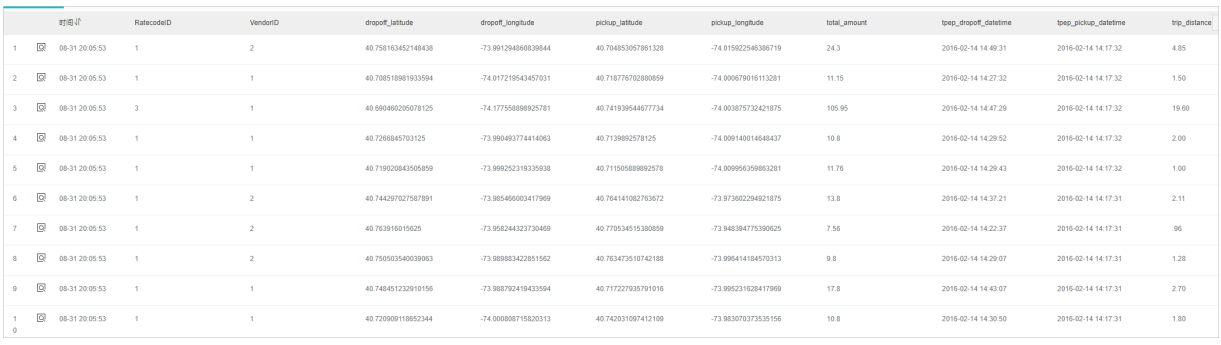
19.16. 分析-行车轨迹日志

出租车公司把载客日志保存在阿里云日志服务上，利用日志服务可靠的存储，以及快速统计计算，挖掘日志中有用信息。本文将展示出租车公司如何使用阿里云日志服务来挖掘数据中的信息。

出租车公司记录了每一次载客交易发生的信息细节，包括上下客时间、经纬度、路程距离、支付方式、支付金额、缴税额等信息。详细的数据，为出租车公司的运营提供了极大的帮助。例如，了解哪些时间段比较热门，对应增加运行车次；哪些地区需求比较广泛，调度更多车辆前往。这些数据，使得乘客的需求得到了及时的响应，而驾驶员的收入也得到了提高，进而整个社会的效率得到了提高。

数据样例：

```
RatecodeID:  1VendorID:  2__source__:  192.0.2.1    __topic__:  dropoff_latitude:  40.743995666503906
dropoff_longitude:  -73.983505249023437extra:  0    fare_amount:  9    improvement_surcharge:  0.3
mta_tax:  0.5    passenger_count:  2    payment_type:  1    pickup_latitude:  40.761466979980469    p
ickup_longitude:  -73.96246337890625    store_and_fwd_flag:  N    tip_amount:  1.96    tolls_amount:
0    total_amount:  11.76    tpep_dropoff_datetime:  2016-02-14 11:03:13    tpep_dropoff_time:  14554
18993    tpep_pickup_datetime:  2016-02-14 10:53:57    tpep_pickup_time:  1455418437    trip_distance
:  2.02
```



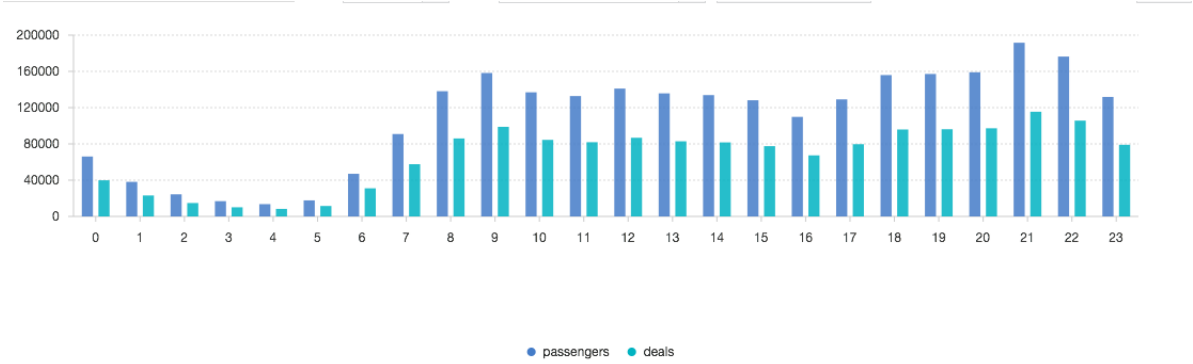
	时间	RatecodeID	VendorID	dropoff_latitude	dropoff_longitude	pickup_latitude	pickup_longitude	total_amount	tpep_dropoff_datetime	tpep_pickup_datetime	trip_distance
1	08-31 20:05:53	1	2	40.758163452148438	-73.991294860839844	40.704853057861328	-74.015922546386719	24.3	2016-02-14 14:49:31	2016-02-14 14:17:32	4.85
2	08-31 20:05:53	1	1	40.708518981933594	-74.017219543457031	40.718776702880859	-74.000679016113281	11.15	2016-02-14 14:27:32	2016-02-14 14:17:32	1.50
3	08-31 20:05:53	3	1	40.690460205078125	-74.177558898925781	40.741939544677734	-74.003875732421875	105.95	2016-02-14 14:47:29	2016-02-14 14:17:32	19.60
4	08-31 20:05:53	1	1	40.7266845703125	-73.990493774414063	40.7139892578125	-74.009140014648437	10.8	2016-02-14 14:29:52	2016-02-14 14:17:32	2.00
5	08-31 20:05:53	1	1	40.719020843505859	-73.999252319335938	40.711505889892578	-74.009956359863281	11.76	2016-02-14 14:29:43	2016-02-14 14:17:32	1.00
6	08-31 20:05:53	1	2	40.744297027587891	-73.985466003417969	40.764141082763672	-73.973602294921875	13.8	2016-02-14 14:37:21	2016-02-14 14:17:31	2.11
7	08-31 20:05:53	1	2	40.763916015625	-73.958244323730469	40.770534515380859	-73.948394775390625	7.56	2016-02-14 14:22:37	2016-02-14 14:17:31	.96
8	08-31 20:05:53	1	2	40.750503540039063	-73.989883422851562	40.763473510742188	-73.996414184570313	9.8	2016-02-14 14:29:07	2016-02-14 14:17:31	1.28
9	08-31 20:05:53	1	1	40.748451232910156	-73.988792419433594	40.717227935791016	-73.995231628417969	17.8	2016-02-14 14:43:07	2016-02-14 14:17:31	2.70
10	08-31 20:05:53	1	1	40.720909118652344	-74.000808715820313	40.742031097412109	-73.983070373535156	10.8	2016-02-14 14:30:50	2016-02-14 14:17:31	1.80

常见的统计

要对数据进行查询和分析，请先配置索引。

- 分时段乘车人次，查看哪些时段比较热门。

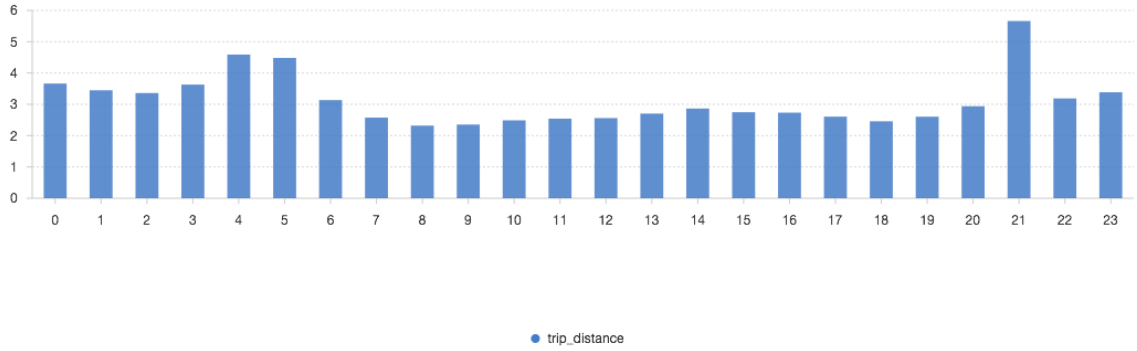
```
*| select count(1) as deals, sum(passenger_count) as passengers,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit 24
```



从结果中可以看出，上午上班时间以及晚上下班后，是乘车需求最旺盛的时候，出租车公司可以相应的调度更多的车辆。

- 分时段平均乘车里程。

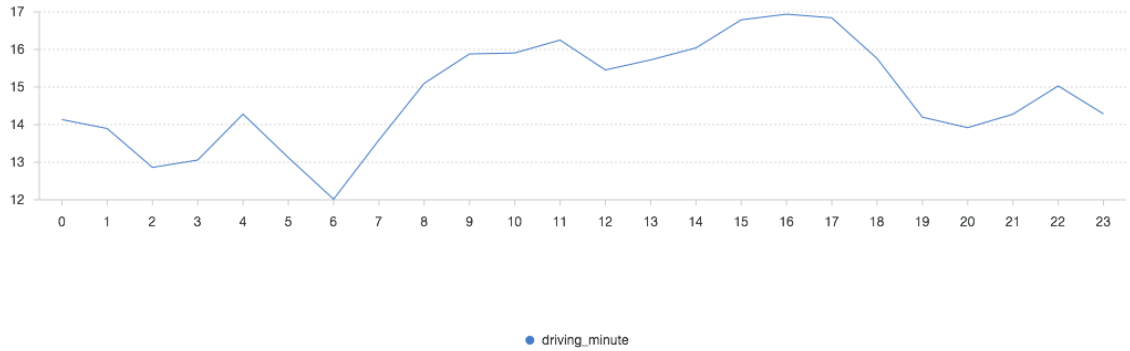
```
*| select  avg(trip_distance)  as trip_distance,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit 24
```



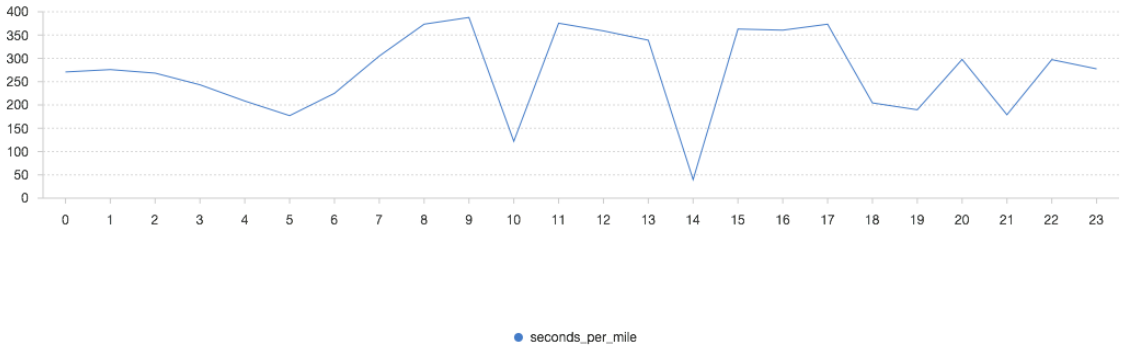
某些时刻，对乘车里程的需求也挺旺盛，出租车公司在对应的时候也需要准备更多的车辆。

- 分时段平均乘车分钟数，单位里程需要的秒数，看看哪些时段比较堵。

```
*| select  avg(tpep_dropoff_time-tpep_pickup_time)/60  as driving_minutes,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit 24
```



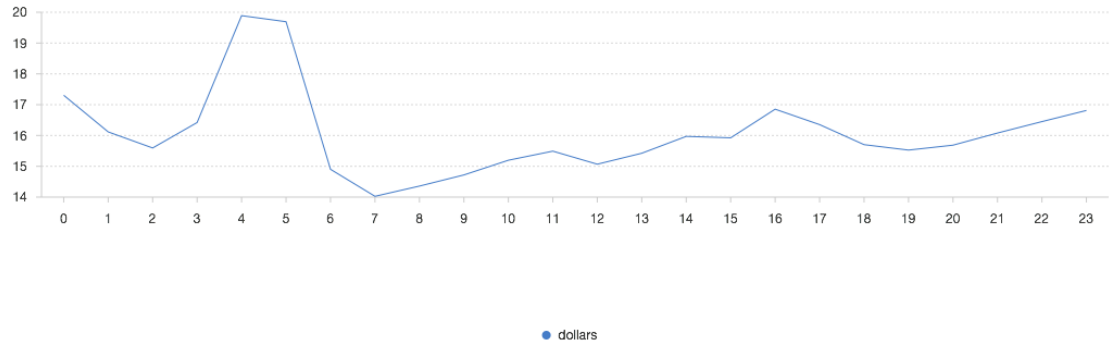

```
*| select  sum(tpep_dropoff_time-tpep_pickup_time)/sum(trip_distance)  as driving_minutes,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit 24
```



一些时刻特别堵，需要准备更多车辆来应对需求。

- 分时段平均乘车费用，看看哪些时间赚的多。

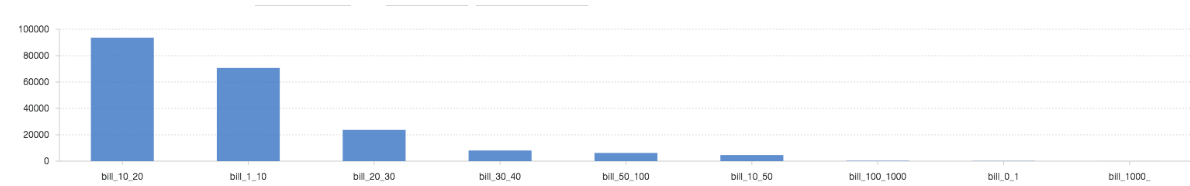
```
*| select  avg(total_amount)  as dollars,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit 24
```



凌晨4点钟的客单价比较高，有经济压力的驾驶员可以选择在这个时候提供服务。

- 看看账单范围分布情况。

```
*| select case when total_amount < 1 then 'bill_0_1'
when total_amount < 10 then 'bill_1_10'
when total_amount < 20 then 'bill_10_20'
when total_amount < 30 then 'bill_20_30'
when total_amount < 40 then 'bill_30_40'
when total_amount < 50 then 'bill_10_50'
when total_amount < 100 then 'bill_50_100'
when total_amount < 1000 then 'bill_100_1000'
else 'bill_1000_'  end
as bill_level , count(1) as count group by
case when total_amount < 1 then 'bill_0_1'
when total_amount < 10 then 'bill_1_10'
when total_amount < 20 then 'bill_10_20'
when total_amount < 30 then 'bill_20_30'
when total_amount < 40 then 'bill_30_40'
when total_amount < 50 then 'bill_10_50'
when total_amount < 100 then 'bill_50_100'
when total_amount < 1000 then 'bill_100_1000'
else 'bill_1000_'  end
order by count desc
```



从成交金额的成交区间，可以看出大部分的成交金额在1到20美元之间。

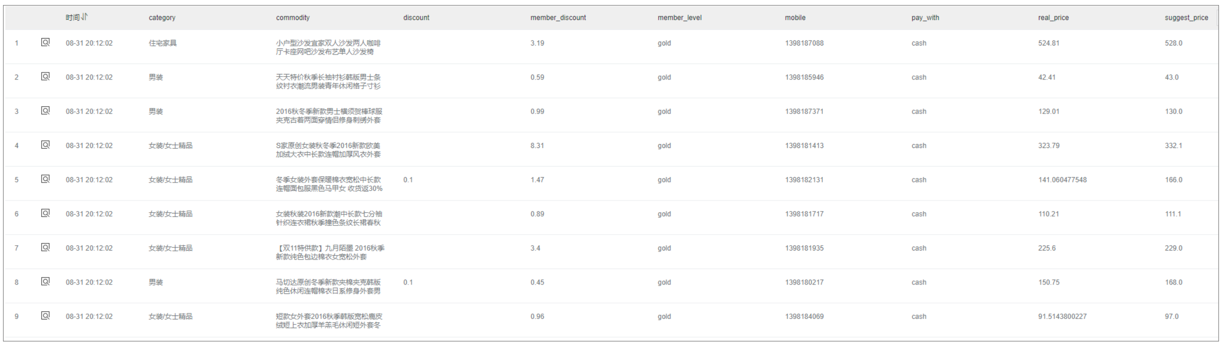
19.17. 分析-销售系统日志

成交账单是电商公司的核心数据，是一系列营销和推广活动最终的转化成果。这些数据包含了很多有价值的信息，从这些数据出发，可以描绘出用户画像，为下一步的营销提供方向。账单数据还能提供货物的受欢迎程度，为下一步备货提供准备。

账单信息以日志的形式保存在阿里云日志服务上，日志服务能够提供快速的查询和SQL统计，在秒级别计算上亿条日志。本文将以几个例子来讲解如何挖掘有用信息。

一个完整的成交账单，包含了货物的信息（名称、价格）、成交的价格信息（成交的价格、支付手段、优惠信息）、交易对手的信息（会员信息），账单日志样例如下。

```
__source__:  10.164.232.105       __topic__:  bonus_discount:  category:  男装    commodity:  天天特价秋冬款青少年加绒加厚紧身牛仔裤男士冬季修身型小脚裤子 commodity_id:  443 discount:  member_discount:  member_level:  nomember_point:  memberid:  mobile:  pay_transaction_id:  060f0e0d080e0b05060307010c0f0209010e0e010c0a0605000606050b0c0400 pay_with:  alipay real_price:  52.0 suggest_price:  52.0
```



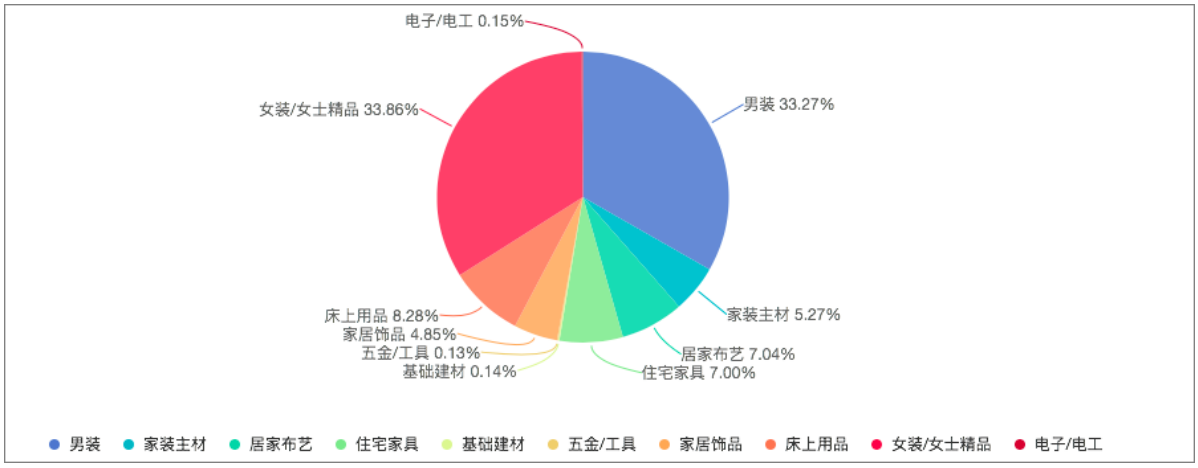
		时间	category	commodity	discount	member_discount	member_level	mobile	pay_with	real_price	suggest_price
1		08-31 20:12:02	住宅家具	小户型沙发直家双人沙发两人咖啡厅卡座网吧沙发布艺单人沙发椅		3.19	gold	1398187088	cash	524.81	528.0
2		08-31 20:12:02	男装	天天特价秋季长袖衬衫韩版男士条纹衬衣潮流男装青年休闲格子寸衫		0.59	gold	1398185946	cash	42.41	43.0
3		08-31 20:12:02	男装	2016秋冬季新款男士横须贺棒球服夹克古着两面穿情侣修身刺绣外套		0.99	gold	1398187371	cash	129.01	130.0
4		08-31 20:12:02	女装/女士精品	S家原创女装秋冬季2016新款欧美加绒大衣中长款连帽加厚风衣外套		8.31	gold	1398181413	cash	323.79	332.1
5		08-31 20:12:02	女装/女士精品	冬季女装外套保暖棉衣宽松中长款连帽面包服黑色马甲女 收货返30%	0.1	1.47	gold	1398182131	cash	141.060477548	166.0
6		08-31 20:12:02	女装/女士精品	女装秋装2016新款潮中长款七分袖针织连衣裙秋季撞色条纹长裙春秋		0.89	gold	1398181717	cash	110.21	111.1
7		08-31 20:12:02	女装/女士精品	【双11特供款】九月陌墨 2016秋季新款纯色包边棉衣女宽松外套		3.4	gold	1398181935	cash	225.6	229.0
8		08-31 20:12:02	男装	马切达原创冬季新款夹棉夹克韩版纯色休闲连帽棉衣日系修身外套男	0.1	0.45	gold	1398180217	cash	150.75	168.0
9		08-31 20:12:02	女装/女士精品	短款女外套2016秋季韩版宽松鹿皮绒短上衣加厚羊羔毛休闲短外套冬		0.96	gold	1398184069	cash	91.5143800227	97.0

统计分析

要对数据进行查询和分析，请先配置索引。

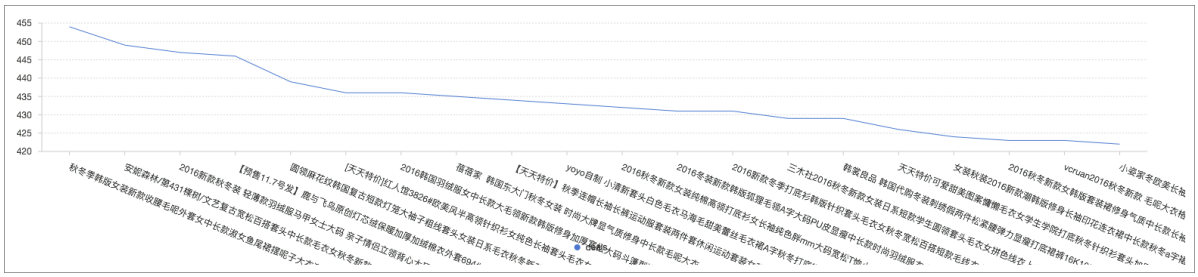
- 查看产品的销售占比。

```
*|select count(1) as pv ,category group by category limit 100
```



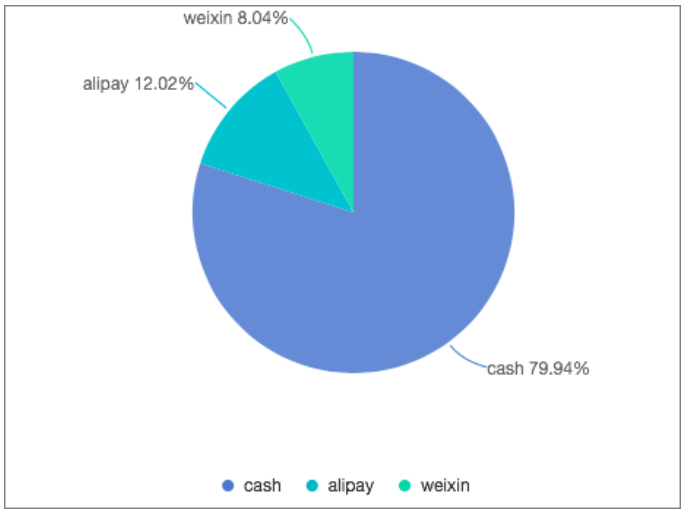
- 查看女装销售。

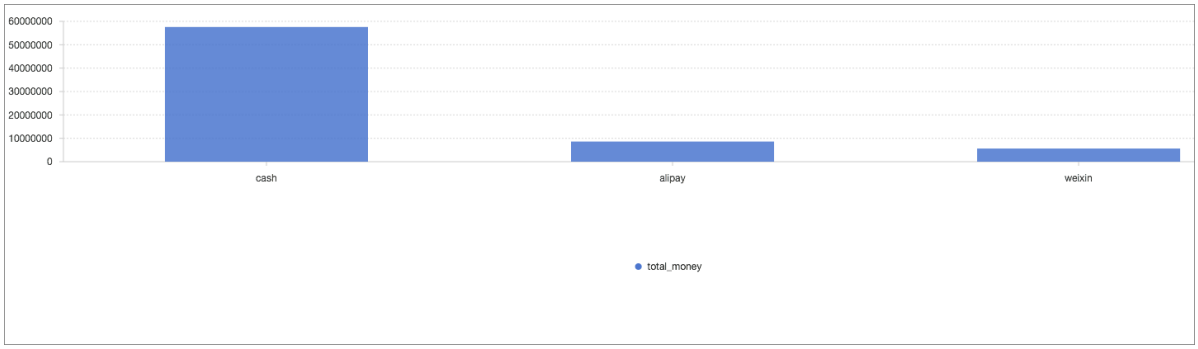
```
category:  女装/女士精品  | select count(1) as deals , commodity
group by commodity order by deals  desc limit 20
```



- 不同支付方式所占份额、成交额。

```
*  | select count(1) as deals , pay_with group by pay_with  order by deals  desc limit 20
*  | select sum(real_price)  as total_money , pay_with group by pay_with  order by total_money  de
sc limit 20
```





19.18. 通过日志服务实现数据库MySQL入湖OSS实践

本实践内容来自实战派，由阿里云专家创作，为您介绍如何借助日志服务将关系型数据库MySQL数据入湖和实践，并介绍数据在入湖之前可以日志服务可以提供哪些开箱即用的功能。

目标读者

需要将MySQL数据导入SLS或者OSS进行分析和存储的开发运维人员。

实施流程

直达实战派-通过日志服务实现据库MySQL入湖OSS实践

20.常见问题

20.1. 查询与分析常见问题

本文列举日志服务查询与分析的常见问题。

- 日志查询常见问题
- 查询不到日志的排查思路
- 查询与分析日志的常见报错
- 日志消费与查询区别
- 模糊查询
- 如何精确查询日志？
- 查询不精确有哪些原因？
- 如何设置字段索引？
- 如何修改SQL查询语句输出结果的行数？
- 如何查询日志的来源机器和日志条目？
- 如何精确地按照时间排序查询日志？
- 如何将日志下载到本地？

20.2. 日志查询常见问题

本文介绍日志查询常见问题。

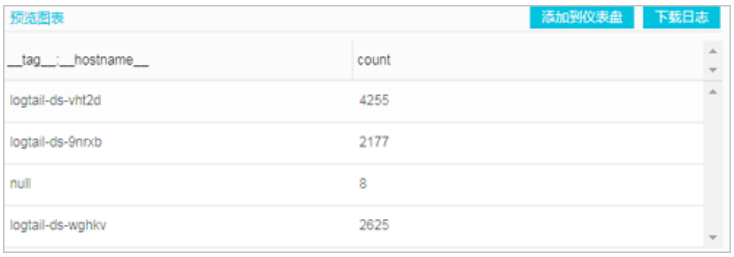
如何在查询时判断日志的来源机器

如果通过Logtail采集日志时，机器组类型为IP地址机器组，机器组中的机器通过内网IP区分。在查询时，可以通过hostname和自定义配置的工作IP来判断日志的来源机器。

例如，统计日志中不同hostname出现的次数。

> **说明** 已配置__tag__:__hostname__的字段索引并开启统计功能。

```
* | select "__tag__:__hostname__" , count(1) as count group by "__tag__:__hostname__"
```



如何在日志数据中搜索IP地址？

在日志数据中搜索IP地址，支持完全匹配的方式查询。您可以直接在日志数据中直接搜索指定IP地址相关的日志信息，比如包含指定IP地址、过滤指定IP地址等。但是目前尚不支持部分匹配的方式检索，即不能直接搜索IP地址的一部分，因为小数点（.）不是日志服务默认的分词项。如果需要部分匹配，建议自行过滤，例如使用SDK先下载数据后，在代码里使用正则表达式或者用string.indexof等方法搜索。

例如执行如下查询语句，查询结果中仍会出现121.42.0网段地址。

```
not ip:121.42.0 not status:200 not 360jk not DNSPod-Monitor not status:302 not jiankongbao
        not 301 and status:403
```

如何完成双重条件查询？

需要使用两个条件查询日志时，只需同时输入两个语句即可。

例如，需要在Logstore中查询数据状态不是OK或者Unknown的日志。直接搜索 `not OK not Unknown` 即可得到符合条件的日志。

日志服务提供哪些渠道查询日志？

日志服务提供如下三种方式查询日志：

1. 通过日志服务控制台查询。更多信息，请参见查询和分析日志。
2. 通过SDK查询。更多信息，请参见SDK。
3. 通过Restful API查询。更多信息，请参见GetLogs。

20.3. 查询不到日志的排查思路

本文主要介绍在日志服务控制台中使用日志查询功能时，查询不到日志的排查思路。

未成功采集日志

如果未成功采集日志到日志服务，则无法查询到目标日志。请在采集的预览界面查看是否有日志。如果有，说明已采集到日志，建议您排查其他原因。如果没有，可能是以下原因造成，请进一步排查。

- 日志源端没有产生日志。

  请检查您的日志源端。

- Logtail无心跳。

  请在**机器组状态**页面中查看机器是否有心跳。如果没有心跳，请参见Logtail机器无心跳进行排查。

- 待采集的日志文件没有实时写入数据。

  您可以打开*/usr/local/ilogtail/ilogtail.LOG*查看报错信息。常见错误如下：

  - parse delimiter log fail：通过分隔符模式采集日志出错。
  - parse regex log fail：通过完整正则模式采集日志出错。

分词符设置错误

查看已设置的分词符，根据分词符对日志内容进行分割后，确认是否可以得到关键字。例如分词符为 `,;=()[]{}?@&<>/:'` ，日志内容为 `abc"defg,hij` ，则该日志会被分割为 `abc"defg` 和 `hij` ，当您使用 `hij` 查询时，可以查询到该日志，但是当您使用 `abc` 进行查询时，无法查询到该日志。

> **说明**
>
> - 为了节约您的索引费用，推荐使用字段索引。更多信息，请参见索引类型。
> - 配置索引后，只对新写入的日志数据生效。如果您要查询和分析历史数据，请使用重建索引功能。具体操作，请参见重建索引。
>
>   您可以在**查询分析**面板中，检查已设置的分词符是否符合要求。更多信息，请参见配置索引。

其他原因

- 确认您的日志时间是否在查询的时间范围内。
- 由于日志预览功能是实时的，但是查询功能存在最多1分钟的延迟，所以您可以在日志产生后等待1分钟再进行查询。

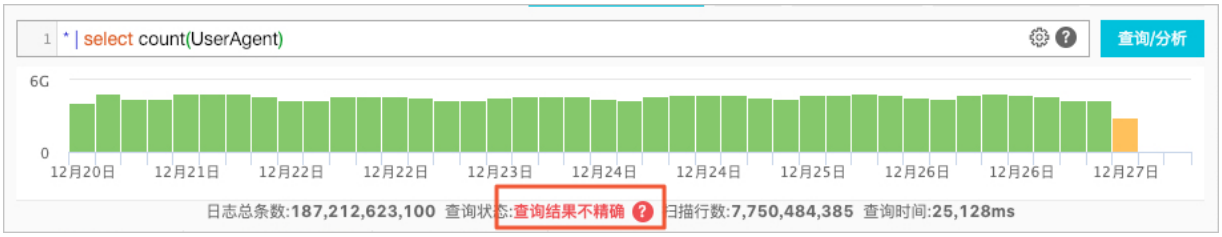
如果您的问题仍未解决，请提交工单。

20.4. 控制台提示“查询结果不精确”，如何解决？

在您查询和分析日志时，如果日志服务控制台提示**查询结果不精确**，可参考本文进行排查。

问题描述

在您查询和分析日志时，如果日志服务控制台提示**查询结果不精确**，表示日志服务未能扫描全部日志，返回的查询和分析结果不是基于全部日志的精确结果。



可能原因

查询结果不精确一般由以下原因造成。

- 查询时间范围太大

  例如查询时间范围为3个月或1年时，日志服务无法在一次查询中完整扫描这个时间段内的所有日志。为了快速返回结果，先返回部分不精确的结果。

- 查询条件过于复杂

  例如您在同一个查询语句中设置了30个查询条件，则日志服务无法一次性读取查询结果。

- SQL计算时要读取的数据量太大

  如果您在一个分析语句中设置了多个字段，而这些字段对应的数据量很大，超过Shard读取能力（每个Shard仅能读取1 GB数据），则返回结果不精确。

解决方案

缩小查询和分析的时间范围，多次（10次以内）执行查询和分析语句。

20.5. 查询与分析日志的常见报错

本文介绍查询与分析日志的常见报错及对应的解决方法。

line 1:44: Column 'XXX' cannot be resolved;please add the column in the index attribute

- 报错原因

  未对XXX字段建立索引。

- 解决方法

  为目标字段设置索引，并开启统计功能。具体操作，请参见配置索引。

ErrorType:QueryParseError.ErrorMessage:syntax error error position is from column:10 to column:11,error near < : >

- 报错原因

  查询和分析语句的语法错误，错误位置在冒号（:）附近。

- 解决方法

  检查及修改查询和分析语句，然后重新执行。

Column 'XXX' not in GROUP BY clause;please add the column in the index attribute

- 报错原因

  在SQL语句中，如果您使用了GROUP BY子句，则在执行SELECT语句时，只能选择GROUP BY的列或者对任意列进行聚合计算，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。

- 解决方法

  修改查询和分析语句，然后重新执行。例如上述示例的正确语句为 `* | SELECT status, arbitrary(request_time), count(*) AS PV GROUP BY status` 。更多信息，请参见GROUP BY子句。

sql query must follow search query,please read syntax doc

- 报错原因

  仅使用了分析语句。在日志服务中，分析语句必须与查询语句一起使用，格式为 `查询语句|分析语句` 。

- 解决方法

  在分析语句前加上查询语句，例如 `* | SELECT status, count(*) AS PV GROUP BY status` 。更多信息，请参见基础语法。

line 1:10: identifiers must not start with a digit; surround the identifier with double quotes

- 报错原因

  分析语句中的列名、变量名等以数字开头，不符合规范。在SQL标准中，列名必须由字母、数字和下划线（_）组成，且以字母开头。

- 解决方法

  更改别名。如何修改，请参见列的别名。

line 1:9: extraneous input ‘’ expecting

- 报错原因

  输入了多余的中文引号。

- 解决方法

  检查及修改查询和分析语句，然后重新执行。

key (XXX) is not config as key value config,if symbol : is in your log,please wrap : with quotation mark "

- 报错原因

  未对XXX字段建立索引，或者您所使用的字段中包含了特殊字符（例如空格）但未使用双引号（""）包裹。

- 解决方法
  - 为目标字段设置索引，并开启统计功能。具体操作，请参见配置索引。
  - 使用双引号（""）包裹目标字段。

Query exceeded max memory size of 3GB

- 报错原因

当前查询和分析语句所使用服务端的内存超过3 GB。该问题通常是因为使用GROUP BY子句去重后，值太多导致的。

- 解决方法

  优化GROUP BY子句，减少GROUP BY子句中字段的个数。

ErrorType:ColumnNotExists.ErrorPosition,line:0,column:1.ErrorMessage:line 1:123: Column 'XXX' cannot be resolved; it seems XXX is wrapper by ";if XXX is a string ,not a key field, please use 'XXX'

- 报错原因

  XXX不是索引字段，不能使用双引号（""）包裹。在分析语句中，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。

- 解决方法
  - 如果XXX为分析字段，您需要为该字段配置索引，并开启统计功能。具体操作，请参见配置索引。
  - 如果XXX为普通字符串，需要使用单引号（''）包裹。

user can only run 15 query concurrently

- 报错原因

  分析操作并发数超过了15个。在日志服务中，单个Project支持的最大分析操作并发数为15个。

- 解决方法

  合理控制分析操作的并发数。

unclosed string quote

- 报错原因

  查询和分析语句的双引号（"）未成对出现。

- 解决方法

  检查及修改查询和分析语句，然后重新执行。

error after :.error detail:error after :.error detail:line 1:147: mismatched input 'in' expecting {<EOF>, 'GROUP', 'ORDER', 'HAVING', 'LIMIT', 'OR', 'AND', 'UNION', 'EXCEPT', 'INTERSECT'}

- 报错原因

  输入了错误的关键词in。

- 解决方法

  检查及修改查询和分析语句，然后重新执行。

Duplicate keys (XXX) are not allowed

- 报错原因

  建立索引时，为字段配置了重复的索引。

- 解决方法

  检查您的索引配置。具体操作，请参见配置索引。

only support * or ? in the middle or end of the query

- 报错原因

  使用模糊查询时，未正确使用通配符。

- 解决方法

修改查询和分析语句中的通配符。相关说明如下：

- 支持在词的中间或者末尾加上模糊查询关键字，即星号（*）或问号（?）。
- 星号（*）或问号（?）不能用在词的开头。
- long数据类型和double数据类型不支持使用星号（*）或问号（?）进行模糊查询。

logstore (xxx) is not found

- 报错原因

  日志库XXX不存在或未配置索引。

- 解决方法

  请检查对应的日志库是否存在。如果存在，请确保已为日志库开启统计，并至少存在一列索引字段。

condition number 43 is more than 30

- 报错原因

  您在查询语句中使用的字段数为43个，超过日志服务所限制的30个。

- 解决方法

  请修改查询语句，将字段数量控制在30个以内。

ErrorType:SyntaxError.ErrorPosition,line:1,column:19.ErrorMessage:line 1:19: Expression "data" is not of type ROW

- 报错原因

  查询和分析语句中所使用的字段的数据类型错误。

- 解决方法

  检查及修改查询和分析语句，然后重新执行。

20.6. 日志消费与查询区别

日志服务提供日志消费和查询功能，都均为对日志数据的读操作。

日志消费

全量数据顺序（FIFO）读写，提供类似Kafka的功能。

- 每个LogStore有一个或多个Shard，数据写入时，随机落到某一个Shard中。
- 可以从指定Shard中，按照日志写入Shard的顺序批量读取日志。
- 根据接收日志的时间，设置批量读取Shard日志的起始位置（cursor）。

日志查询（LogSearch）

提供海量日志查询和分析功能，根据条件进行日志查询与统计。

- 通过查询条件查找符合要求的数据。
- 支持运算符AND、NOT、OR的多条件组合查询和结果SQL统计。
- 数据查询不区分Shard。

区别

对比项目	日志查询	日志消费
查找关键词	支持	不支持

对比项目	日志查询	日志消费
读取少量数据	快	快
读取全量数据	慢（100条日志100ms，不建议通过该方式读取数据。）	快（1 MB日志10ms，推荐方式。）
读取是否区分日志主题	区分	不区分，只以Shard作为标识。
读取是否区分Shard	不区分，查询所有Shard。	区分，单次读取需要指定Shard。
费用	较高	低
适用场景	监控、问题调查与分析等场景。	流式计算、批量处理等全量处理场景。

20.7. 模糊查询

本文介绍不同方式的模糊查询。

通过查询语法进行模糊查询

在日志服务查询语法中，星号（*）代表多个任意字符的词，问号（?）代表单个字符的词。例如：**abc***代表查询以abc开头的词。**ab?d**代表查询以ab开头、d结尾且中间包含单个字符的词， 详情请参见查询语法。

> **说明** 这种模糊查询的结果是不精确的，首先查询到命中的100个词，然后再查询这100个词命中的日志。所以如果你查询的模糊条件前缀很短且日志中超过100个词，那么查询结果是不精确的。通过这种查询语法，再结合not语句时，会发现某些查询条件未被过滤。例如：执行**not abcd***查询语句，在结果中出现以abcd开头的词。

通过SQL的like语法进行精确的模糊查询

like语法满足标准的SQL like语法，在like语法中百分号（%）代表任意个字符。下划线 （_）代表单个字符。

示例：查询key满足abcd开头的所有日志，对应的查询分析语句如下所示。

```
* | select * from log where key like 'abcd%'
```

通过SQL的正则式函数进行模糊查询

通过正则式函数，可以在一个正则式中查询多个词。并且正则式的表达语义比like语法更强大，可以搜索满足数字的词以及满足特定字符的词等，详情请参见正则式函数。

示例：

- *** | select * from log where regexp_like(key, abc*)** 表示查询以abc开头的词。
- *** | select * from log where regexp_like(key, abc\d+)** 表示查询以abc开头且后面跟着数字的词。
- *** | select * from log where regexp_like(key, abc[xyz])**表示查询以abc开头且后面跟着xyz中的某个字符的词。

20.8. 如何精确查询日志?

当您要精确查询包含多个关键字的日志时，您可以使用like语法进行查询。

- 日志样例

```
body_bytes_sent:1061
http_user_agent:Mozilla/5.0 (Windows; U; Windows NT 5.1; ru-RU) AppleWebKit/533.18.1 (KHTML, like
Gecko) Version/5.0.2 Safari/533.18.5
remote_addr:192.0.2.2
remote_user:vd_yw
request_method:DELETE
request_uri:/request/path-1/file-5
status:207
time_local:10/Jun/2021:19:10:59
```

- 查询需求

  查询http_user_agent字段值中包含 `like Gecko` 的日志。

- 错误的查询语句

  ```
  "like" and "Gecko"
  ```

  使用该查询语句，无法精确查询，查询结果将包含 `like Gecko` 、 `Gecko like` 、 `like abc Gecko` 和 `Gecko abc like` 的日志。

- 正确的查询语句

  ```
  * | Select * where http_user_agent like '%like Gecko%'
  ```

  其中，http_user_agent为日志字段。

  like语法满足标准的SQL like语法，在like语法中百分号（%）代表任意个字符。下划线 （_）代表单个字符。

20.9. 查询和分析JSON日志的常见问题

本文介绍查询和分析JSON日志的常见问题。

日志样例

本文中介绍的各个案例是基于如下JSON格式的订单处理系统日志。

```
{
    "request":{
        "clientIp":"20[redacted]30",
        "http.path":"/order/buy",
        "param":{
            "userId": "186453",
            "orders":[
                {
                    "commodity":["bread","milk","meat"],
                    "payment":132
                },
                {
                    "commodity":["milk","beer"],
                    "payment":49
                }
            ]
        }
    },
    "response":"{\"errcode\":400,\"msg\":\"insufficient\"}"
}
```

- request字段为订单请求信息，JSON格式。一个请求中包含一个用户的多个订单，订单中包含购买的商品和支付总价。
- response字段为订单处理结果。
  - 请求成功时，response字段值为SUCCESS。
  - 请求失败时，response字段值为JSON格式，包含errcode和msg信息。

您可以通过Logtail将该日志采集到日志服务中，进行查询与分析。具体操作，请参见使用JSON模式采集日志。

如何设置索引？

索引是一种存储结构，用于对日志中的一列或多列进行排序。您只有设置索引后，才能进行查询和分析操作。在为JSON日志设置索引时，可能涉及如下方面的问题。

如何选择索引类型？

日志服务索引分为全文索引和字段索引，您可以参考如下说明，选择索引类型。更多信息，请参见配置索引。

- 如果您需要查询日志中的所有字段，建议创建全文索引；如果您明确仅查询部分字段，可针对目标字段建立字段索引，减少索引费用。
- 如果对字段有SQL分析需求，则必须对目标字段建立索引，并开启统计功能。

> **说明** 如果您同时配置了全文索引和字段索引，则配置了字段索引的字段，以字段索引的配置为准。

例如您要统计分析request字段和response字段，则需要创建这两个字段的字段索引，并开启统计功能。

在索引配置中，如何选择字段的数据类型？

在设置索引时，字段的数据类型分为text、long、double和JSON。更多信息，请参见数据类型。

当您设置JSON字段的数据类型时，可参考如下思路。

- 如果字段值不是标准JSON格式，可能只是包含了JSON格式的内容，则设置为text类型；如果字段值是标准JSON格式，则设置为JSON类型。

  > **说明** 针对非完全合法的JSON日志，日志服务支持解析合法部分。

- 将某个字段设置为JSON类型后，如果对JSON对象中的某个叶子节点有进一步的分析需求，可以为叶子节点建立索引，这样可以加快叶子节点的查询和分析速度，但同时也会产生额外的索引费用。
- 日志服务支持JSON对象中的叶子节点建立索引，但不支持包含叶子节点的子节点建立索引。
- 日志服务不支持值为JSON数组的字段建立索引，也不支持为JSON数组中的字段建立索引。

例如基于本文的日志样例，您可以创建如下索引。

- request字段
  - request字段为JSON格式，设置为JSON类型，并开启统计功能。
  - request.clientIp字段需要经常分析，建议单独建立索引，设置为text类型，并开启统计功能。
  - request.http.path字段很少需要分析，可不用单独建立索引。在需要分析时，直接通过JSON函数进行解析。
  - request.param字段为包含叶子节点的子节点，不支持建立索引。
  - request.param.userId字段需要经常分析，建议单独建立索引，设置为text类型，并开启统计功能。
  - request.param.orders字段值为JSON数组，不支持建立索引。
- response字段

  response字段不一定是JSON格式，因此设置为text类型，并开启统计功能。

创建索引后，新采集的日志将显示为如下格式。

如何设置别名？

JSON叶子节点的路径较长，您可以为其设置别名。更多信息，请参见列的别名。

> **说明**
>
> - 在设置索引时，不同字段的字段名或别名不能重复。
> - 对于JSON类型的字段，JSON叶子节点的名称是按照全路径进行重名判定的。例如为response字段设置别名为clientIp，系统不会判定该别名与request.clientIp字段名重复。

如何查询和分析有索引的JSON字段？

查询和分析语句格式为 `查询语句|分析语句` 。在分析语句中，您必须使用双引号（""）包裹字段名称，使用单引号（''）包裹字符串。您还需为目标字段加上所有的父路径，格式为 `Key1.Key2.Key3` 。例如 `request.clientIp` 、 `request.param.userId` 。更多信息，请参见查询和分析JSON日志。

例如统计186499用户的客户端IP地址，可执行如下语句。

```
*
and request.param.userId: 186499 |
SELECT
  distinct("request.clientIp")
```

查询和分析结果如下所示。

request.clientIp
18
18

何时使用JSON函数？

首先，在查询和分析JSON日志时，如果数据量很大或结构复杂但相对固定，并且您对查询分析性能有要求时，建议对JSON叶子节点建立字段索引，然后再进行查询分析。如果数据量比较小，出于成本考虑，您可以不对JSON叶子节点建立字段索引，而是使用JSON函数进行查询和分析。使用JSON函数，可以灵活地对JSON日志进行动态处理和分析。另外，针对一些特殊情况，只能使用JSON函数进行查询与分析。

- 字段值不一定是JSON格式或者需要先进行一些预处理。

  例如response字段，只有在请求失败时是JSON格式，并且包含errcode字段。那么您要分析errcode的分布情况，则需先使用查询语句过滤出请求失败的日志，然后在分析语句中使用JSON函数动态提取errcode字段值。

  ```
  *  not response :SUCCESS |
  SELECT
    json_extract_scalar(response, '$.errcode') AS errcode
  ```

  查询和分析结果如下所示。

errcode
400
400

- 不支持建立索引的JSON节点，只能使用JSON函数实时分析。例如request.param字段和request.param.orders字段。

如何选择json_extract函数和json_extract_scalar函数？

json_extract函数和json_extract_scalar函数都是用于从JSON对象或JSON数组中提取内容，用法类似，主要区别如下：

- json_extract函数的返回值是JSON类型，json_extract_scalar函数的返回值是varchar类型。

  > **说明** 此类型是指SQL语法中的数据类型，例如varchar、bigint、boolean、JSON、array、date等，与日志服务索引中的数据类型不同。您可以通过typeof函数查看SQL分析对象的数据类型。更多信息，请参见typeof函数。

- json_extract函数可以解析JSON对象中任意一块子结构，json_extract_scalar函数只解析标量类型（字符串、布尔值或者整形值）的叶子节点，返回对应的字符串。

例如提取request字段中的clientIp字段，两个函数都支持。

- 使用json_extract函数进行提取。

  ```
  * |
  SELECT
    json_extract(request, '$.clientIp')
  ```

  查询和分析结果如下所示。

_col0
"18[illegible]9"
"18[illegible]9"

- 使用json_extract_scalar函数进行提取。

```
* | SELECT json_extract_scalar(request, '$.clientIp')
```

查询和分析结果如下所示。

_col0
18[illegible]69
18[illegible]69

在上述基础上，如果要提取clientIp字段值中的第一部分，您需要先使用json_extract_scalar函数提取clientIp的值，然后使用split_part函数提取IP地址中的第一个数字。此处不支持使用json_extract函数，因为split_part函数的入参需为varchar类型。

```
* |
SELECT
  split_part(
    json_extract_scalar(request, '$.clientIp'),
    '.',
    1
  ) AS segment
```

查询和分析结果如下所示。

segment
186
189

一般情况下，如果您需要从JSON对象中提取字段进行分析，直接使用json_extract_scalar函数即可。因为json_extract_scalar函数的返回值为varchar类型，便于与其他函数结合使用。但是当您需要对JSON结构本身进行分析时，需要使用json_extract函数。例如您要统计一次请求中的订单数，即统计request.param.orders字段中JSON数组的长度，可使用如下查询分析语句。

```
* |
SELECT
  json_array_length((json_extract(request, '$.param.orders')))
```

查询和分析结果如下所示。

_col0
2
2

> **注意** json_extract_scalar函数的返回值是varchar类型。例如您上述示例中的数值2，其数据类型也是varchar类型，如果您要对该值进行求和等计算，需要先使用cast函数，将其转换为bigint类型。更多信息，请参见类型转换函数。

如何设置json_path？

使用json_extract等函数从JSON日志中提取字段时，您需指定json_path，用于标明需要提取JSON对象中的哪一部分。json_path格式为 `$.a.b` ，美元符号（$）代表当前JSON对象的根节点，然后通过半角句号（.）引用到待提取的节点。

如果JSON对象中存在 `a.b` 格式的字段（例如http.path字段），则需要使用中括号[]代替半角句号（.），然后使用双引号（""）包裹字段名，例如 `* |SELECT json_extract_scalar(request, '$["http.path"]')` 。

> **说明** 如果是通过SDK进行查询和分析，则需要对双引号（""）进行转义，例如 `* | select json_extract_scalar(request, '$[\"http.path\"]')` 。

提取JSON数组中的某个元素时，可以用中括号[]。在中括号中，通过数字来表示下标，下标从0开始。例如：

- 查看用户第一个订单的金额。

```
* |
SELECT
  json_extract_scalar(request, '$.param.orders[0].payment')
```

查询和分析结果如下所示。

_col0
132
132

- 查看用户第一个订单中购买的第二件商品。

```
* |
SELECT
  json_extract_scalar(request, '$.param.orders[0].commodity[1]')
```

查询和分析结果如下所示。

_col0
milk
milk

如何分析JSON数组？

当日志中有JSON数组时，您可以结合cast函数和UNNEST子句，展开JSON数组，再进行聚合统计。

示例1

例如您要统计所有请求成功的订单的金额，可参见如下思路。

1. 使用查询语句过滤出请求成功的日志，然后在分析语句中使用json_extract函数提取出orders字段的值。

```
*
and response: SUCCESS |
SELECT
  json_extract(request, '$.param.orders')
```

查询和分析结果如下所示。

_col0
[{"commodity":["bread","milk","meat"],"payment":132},{"commodity":["milk","beer","rice"],"payment":49}]
[{"commodity":["bread","milk","meat"],"payment":132},{"commodity":["milk","beer","rice"],"payment":49}]

2. 将上述提取的JSON数组（JSON类型）转换为array类型。

```
*
and response: SUCCESS |
SELECT
  cast(
    json_extract(request, '$.param.orders') AS array(json)
  )
```

查询和分析结果如下所示。

_col0
["{\"commodity\":[\"bread\",\"milk\",\"meat\"],\"payment\":132}","{\"commodity\":[\"milk\",\"beer\",\"rice\"],\"payment\":49}"]
["{\"commodity\":[\"bread\",\"milk\",\"meat\"],\"payment\":132}","{\"commodity\":[\"milk\",\"beer\",\"rice\"],\"payment\":49}"]

3. 使用UNNEST子句将数组展开。

```
*
and response: SUCCESS |
SELECT
  orderinfo
FROM  log,
  unnest(
    cast(
      json_extract(request, '$.param.orders') AS array(json)
    )
  ) AS t(orderinfo)
```

查询和分析结果如下所示。

orderinfo
{"commodity":["milk","beer","rice"],"payment":49}
{"commodity":["milk","beer","rice"],"payment":49}

4. 使用json_extract_scalar提取payment字段值，再使用cast函数将其转换为bigint类型，最后进行求和计算。

```
*
and response: SUCCESS |
SELECT
  sum(
    cast(
      json_extract_scalar(orderinfo, '$.payment') AS bigint
    )
  )
FROM  log,
  unnest(
    cast(
      json_extract(request, '$.param.orders') AS array(json)
    )
  ) AS t(orderinfo)
```

查询和分析结果如下所示。

_col0
10860

示例2

统计所有成功的请求中，每一种商品被购买的数量。您可以先提取order字段，将其转换为array(json)类型，再使用UNNEST语句将其展开，展开结果中的每一行代表一个订单。然后使用json_extract函数提取commodity字段，将其转换为array(json)类型，再使用UNNEST语句将其展开，展开结果中的每一行代表一个商品。最后再进行分组求和。具体思路请参见示例1。

```
*
and response: SUCCESS |
SELECT
  item,
  count(1) AS cnt
FROM  (
    SELECT
      orderinfo
    FROM      log,
      unnest(
        cast(
          json_extract(request, '$.param.orders') AS array(json)
        )
      ) AS t(orderinfo)
  ),
  unnest(
    cast(
      json_extract(orderinfo, '$.commodity') AS array(json)
    )
  ) AS t(item)
GROUP  BY
  item
ORDER BY
  cnt DESC
```

查询和分析结果如下所示。

item	cnt
"milk"	120
"bread"	60
"rice"	60
"beer"	60
"meat"	60