阿里云
日志服务

查询与分析

文档版本：20190918

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
	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。	**禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能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，恢复业务所需时间约10分钟。
	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。	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。	单击 确定。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
##	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。	`swich` *`{stand | slave}`*

目录

法律声明.....I
通用约定.....I
1 简介..... 1
2 实时分析简介..... 3
3 开启并配置索引..... 7
4 查询日志.....13
5 导出日志.....18
6 索引数据类型.....20
6.1 简介.....20
6.2 文本类型.....22
6.3 JSON类型..... 24
6.4 数值类型.....28
7 查询语法与功能.....29
7.1 查询语法.....29
7.2 LiveTail..... 34
7.3 日志聚类..... 39
7.4 上下文查询..... 46
7.5 快速查询..... 49
7.6 快速分析..... 52
7.7 其他功能..... 56
8 SQL分析语法与功能.....61
8.1 通用聚合函数..... 61
8.2 安全检测函数..... 62
8.3 Map映射函数..... 65
8.4 估算函数..... 67
8.5 数学统计函数..... 68
8.6 数学计算函数..... 69
8.7 字符串函数..... 71
8.8 日期和时间函数..... 72
8.9 URL函数..... 78
8.10 正则式函数..... 79
8.11 JSON函数..... 80
8.12 类型转换函数..... 81
8.13 IP地理函数..... 82
8.14 GROUP BY 语法..... 85
8.15 窗口函数..... 86
8.16 HAVING语法.....89
8.17 ORDER BY语法..... 89

8.18 LIMIT语法.....89
8.19 CASE WHEN和IF分支语法.....90
8.20 嵌套子查询.....92
8.21 数组.....92
8.22 二进制字符串函数.....94
8.23 位运算.....95
8.24 同比和环比函数.....96
8.25 比较函数和运算符.....99
8.26 lambda函数.....102
8.27 逻辑函数.....105
8.28 列的别名.....106
8.29 Logstore和RDS联合查询.....106
8.30 空间几何函数.....109
8.31 地理函数.....112
8.32 Join语法.....112
8.33 unnest语法.....113
8.34 电话号码函数.....120
9 机器学习语法与函数.....123
9.1 简介.....123
9.2 平滑函数.....125
9.3 多周期估计函数.....130
9.4 变点检测函数.....133
9.5 极大值检测函数.....136
9.6 预测与异常检测函数.....137
9.7 序列分解函数.....146
9.8 时序聚类函数.....147
9.9 频繁模式统计函数.....153
9.10 差异模式统计函数.....154
9.11 根因分析函数.....156
9.12 相关性分析函数.....159
10 分析进阶.....163
10.1 优秀分析案例.....163
10.2 优化查询.....164
10.3 时间字段转换示例.....166
11 通过JDBC协议分析日志.....168
12 可视化分析.....172
12.1 分析图表.....172
12.1.1 图表说明.....172
12.1.2 表格.....174
12.1.3 折线图.....177
12.1.4 柱状图.....180
12.1.5 条形图.....183
12.1.6 饼图.....185
12.1.7 面积图.....187

12.1.8 单值图 190
12.1.9 进度条 195
12.1.10 地图 198
12.1.11 流图 202
12.1.12 桑基图 204
12.1.13 词云 206
12.1.14 矩形树图 207
12.2 仪表盘 208
12.2.1 仪表盘简介 208
12.2.2 创建和删除仪表盘 210
12.2.3 显示模式 215
12.2.4 编辑模式 219
12.2.5 订阅仪表盘 231
12.2.6 下钻分析 236
12.2.7 仪表盘过滤器 246
12.2.8 Markdown图表 254
13 最佳实践 262
13.1 查询分析-分页 262
13.2 查询-消息服务(MNS)日志 266
13.3 查询分析-程序日志 272
13.4 查询分析-数据库与日志关联分析 281
13.5 查询分析-日志服务与OSS外表关联分析 284
13.6 分析-销售系统日志 288
13.7 分析-网站日志 291
13.8 分析-Nginx监控日志 302
13.9 分析-Nginx访问日志 308
13.10 分析-行车轨迹日志 315
13.11 分析-分析SLB七层访问日志 319
14 FAQ 329
14.1 日志查询常见问题 329
14.2 查询不到日志数据 330
14.3 日志消费与查询区别 332
14.4 日志查询分析常见报错 333
14.5 查询不精确有哪些原因 334

1 简介

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

功能优势

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

索引

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

日志服务索引包括全文索引和字段索引。

- 全文索引：对日志全文内容开启索引，默认的索引会查询日志中所有Key对应的内容，只要有一个命中，就会被查询到。
- 字段索引：为特定的Key配置索引，配置字段索引后，可以通过查询特定Key的内容，缩小查询范围。

其中，在字段索引中，需要指定字段的数据类型。当前，日志服务支持的字段类型包括：text、json、long和double。关于索引数据类型的更多信息，请查看索引数据类型简介。

查询方式

- 控制台查询：

  在日志服务控制台查询页面指定查询时间段和查询语句进行查询。详细说明和查询语法请参考查询日志和#unique_10。
- API查询：

  通过日志服务API中的接口#unique_11和#unique_12接口可以查询日志数据。

> 说明:
> 查询日志前，请确认您已采集到日志数据，且已开启并配置索引。

查询分析语句格式

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

```
$Search |$Analytics
```

语句类型	是否可选	说明
查询语句（Search）	可选	查询条件，可以包括关键词、模糊、数值、区间范围和组合条件。 如果为空或”*”，表示针对当前时间段所有数据不设置任何过滤条件，即返回所有数据。详细说明请参考#unique_10。
分析语句（Analytics）	可选	对查询结果或全量数据进行计算和统计。 如果为空，表示只返回查询结果，不需要做统计分析。详细说明请参考#unique_13。

注意事项

如果您需要检索的日志数据量很大，例如查询时间跨度非常长，其中数据量在百亿以上时，则一次查询请求无法检索完所有数据。在这种情况下，日志服务会把已有的数据返回给您，并在返回结果中告知您该查询结果并不完整。

同时，服务端会缓存 15 分钟内的查询结果。当查询请求的结果有部分被缓存命中，则服务端会在这次请求中继续扫描未被缓存命中的日志数据。为了减少您合并多次查询结果的工作量，日志服务会把缓存命中的查询结果与本次查询新命中的结果合并返回给您。

因此日志服务可以让您通过以相同参数反复调用该接口来获取最终完整结果。

2 实时分析简介

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

语法示例：

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

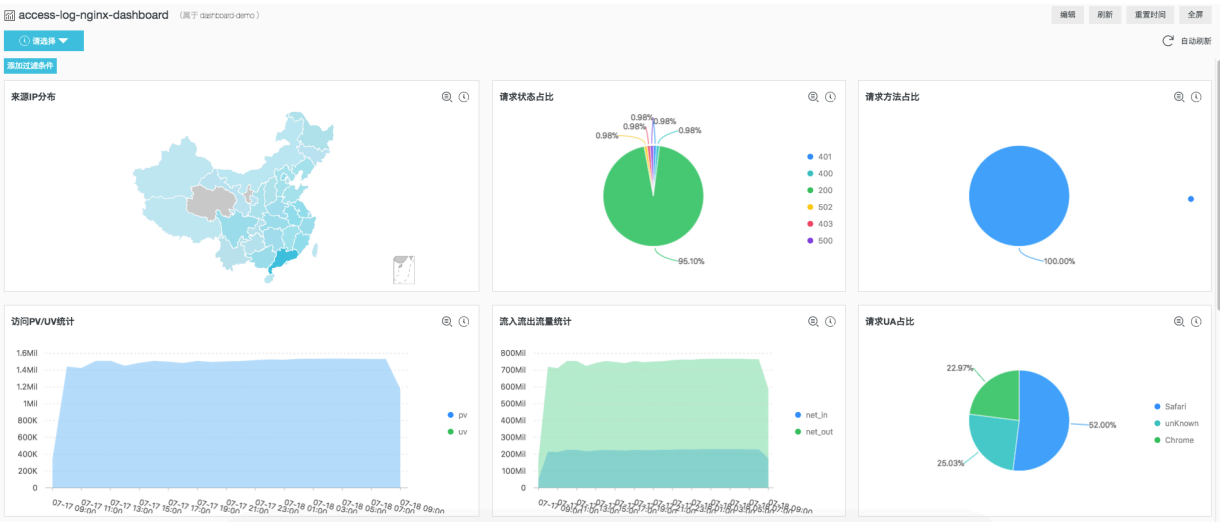
基本语法：

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

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

效果展示

图 2-1: 效果展示



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

前提条件

要使用分析统计功能，必须在查询分析属性设置中打开对应字段的开启统计开关，详情请参考#unique_15。

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

- 开启统计后不会产生额外费用。

支持的SQL语法

日志服务支持以下SQL语法，详细内容请点击链接查看。

图 2-2: 支持的SQL语法

- SELECT聚合计算函数：
  - #unique_16
  - #unique_17
  - #unique_18
  - #unique_19
  - #unique_20
  - #unique_21
  - #unique_22
  - #unique_23
  - #unique_24
  - #unique_25
  - #unique_26
  - #unique_27
  - #unique_28
  - #unique_29
  - #unique_30
  - #unique_31
  - #unique_32
  - #unique_33
  - #unique_34
  - #unique_35
  - #unique_36
  - #unique_37
  - 机器学习函数
  - #unique_39
- #unique_40
- #unique_41
- #unique_42
- #unique_43
- #unique_44
- #unique_45
- #unique_46
- #unique_47

- #unique_48

语法结构

SQL语法结构如下：

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

说明:

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

内置字段

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

字段名	类型	含义
`__time__`	bigint	日志的时间。
`__source__`	varchar	日志来源IP。在搜索时，该字段是source，在SQL中才会带上前后各两个下划线。
`__topic__`	varchar	日志的Topic。

限制说明

1. 每个Project的最高并发为15。
2. 单列varchar，最大长度为2048，超过后会截断。
3. 默认返回100行数据，不支持翻页。若需要返回更多数据，请使用#unique_44。

示例

统计每小时的PV、UV和最高延时对应的用户请求，延时最高的10个延时：

```
*|select date_trunc('hour',from_unixtime(__time__)) as time,
     count(1) as pv,
     approx_distinct(userid)  as uv,
     max_by(url,latency) as top_latency_url,
     max(latency,10) as top_10_latency
     group by  1
     order by time
```

3 开启并配置索引

使用日志服务查询分析功能之前，请先开启并配置索引。

背景信息

开启并配置索引后，您才可以查询配置索引后采集到的日志数据。请根据您的日志字段内容和查询需求，合理配置索引。

> 说明:
>
> - 开启查询和统计后意味着数据将会在后台被索引，会产生索引的流量，以及索引对应存储的空间。
> - 开启和修改索引后，新的索引配置只对新写入的数据生效。
> - 全文索引属性和字段索引属性必须至少启用一种。
> - 打开对应字段的统计功能，才能使用SQL进行统计分析。
> - 如果配置公网IP、Unix时间戳等Tag字段的索引，请配置字段名称为`__tag__:key`，例如`__tag__:__receive_time__`。同时，Tag字段不支持数值类型索引，请将所有Tag字段的索引类型配置为text（文本类型）。例如`__tag__:__receive_time__`字段，在查询时使用模糊查询`__tag__:__receive_time__: 1537928*`或全部匹配字段值`__tag__:__receive_time__: 1537928404`。

采集日志时日志服务会自动将日志来源、时间等信息以Key-Value对的形式添加到日志中，这些字段是日志服务的保留字段。当开启并配置索引时，自动开启这些字段的索引和统计功能。

> 说明:
>
> `__topic__`和`__source__`的索引分词为空，表示在查询这两个字段时，查询关键字必须完全匹配。

表 3-1: 日志服务保留字段

保留字段名称	说明
`__topic__`	日志主题（Topic）。如果您设置了日志主题，日志服务会自动为您的日志添加日志主题字段，Key为`__topic__`，Value为您的主题内容。
`__source__`	日志来源。该字段表示这条日志的来源设备。
`__time__`	使用SDK写入日志数据时指定的日志时间。

操作步骤

1. 登录日志服务控制台，单击Project名称。
2. 单击日志库名称后的图标，选择查询分析。
3. 单击右上角的开启索引。

说明:

若您之前已创建过索引，可以单击查询分析属性 > 设置，修改索引。

4. 配置索引。

日志服务支持配置全文索引和字段索引，请至少配置一种。

说明:

若同时配置了全文索引属性和字段索引属性，以字段索引属性设置为准。

索引类型	说明
全文索引	以文本形式对所有的字段进行建索引，Key和Value都是普通文本，都可查询。long类型字段在查询具体Value时需要指定对应的Key名称，其他类型字段在查询时不必指定Key的名称。
字段索引	配置字段索引后，在查询时要指定Key的名称，当有某个字段配置了字段索引时，该字段上的全文索引不生效。 字段可以配置为多种数据类型，包括： · 文本类型（Text） · JSON类型 · 数值类型（long和double）

a) 配置全文索引。

配置针对全文内容的索引，查询日志时默认查询所有Key对应的内容。

配置	说明	示例
全文索引	对日志全文内容开启索引，默认的索引会查询日志中所有Key对应的内容，只要有一个命中，就会被查询到。	-
大小写敏感	查询时是否区分大小写。其中： · 关闭后，表示不区分大小写，即查询关键字“INTERNALERROR”和“internalerror”都能查询到对应日志。 · 开启后，表示区分大小写，只能通过关键字“internalError”查询到对应日志。	-
包含中文	设置是否区分中英文。 · 开启后，如果日志中包含中文，则对中文按照中文语法进行分词，对英文按照分词字符进行分词。 · 关闭后，对所有内容按照分词符进行分词。	-

配置	说明	示例
分词符	根据指定单字符，将日志内容切分成多个关键词。例如一条日志内容为`a,b;c;D-F`。设置分隔符为逗号（,）、分号（;）和连字符（-），可以将日志内容切分为5个关键词”a” “b” “c” “D” “F”。	`, '";=()[]{}?@&<>/:\n\t`

b) 配置字段索引。

为特定的Key配置索引，配置字段索引后，可以通过查询特定Key的内容，缩小查询范围。

说明:

- 日志服务自动为您创建保留字段`__topic__`、`__source__`和`__time__`的索引和统计功能。
- 本文档以自定义页签为例。Nginx模板和消息服务模板仅用于采集Nginx日志和消息服务日志，不支持自定义配置索引。
- 如果配置公网IP、Unix时间戳等Tag字段的索引，请配置字段名称为`__tag__:key`，例如`__tag__:__receive_time__`。同时，Tag字段不支持数值类型索引，请将所有Tag字段的索引类型配置为text（文本类型）。例如`__tag__:__receive_time__`字段，在查询时使用模糊查询`__tag__:__receive_time__: 1537928*`或全部匹配字段值`__tag__:__receive_time__: 1537928404`。

配置	说明	示例
字段名称	指定日志字段名称。	`_address_`
类型	日志字段内容的数据类型，包括： · text：指定日志字段内容为文本类型。 · long：指定日志字段内容为整数，需要按照数值范围进行查询。 · double：指定日志字段内容为小数，需要按照数值范围进行查询。 · json：指定日志字段内容为json类型。 说明: 数值类型（long和double类型）不支持设置大小写敏感、分词符和包含中文。	-

配置	说明	示例
别名	列的别名。 别名仅用于SQL统计，在底层存储时，仍然是原始名称，搜索时仍需要使用原始名称。 详细说明请参考#unique_47。	`address`
大小写敏感	查询时，英文字母是否区分大小写。其中， · 关闭后，不区分大小写，即查询关键字“INTERNALERROR”和“internalerror”都能查询到样例日志。 · 开启后，区分大小写，只能通过关键字“internalError”查询到样例日志。	-
分词符	根据指定单字符，将日志内容切分成多个关键词。 例如一条日志内容为`a,b;c;D-F`。设置分隔符为逗号“，”、分号“；”和连字符“-”，可以将日志内容切分为5个关键词”a” “b” “c” “D” “F”。	`, '";=()[]{}?@&<>/:\n\t`
包含中文	开启后，如果日志中包含中文，则对中文按照中文语法进行分词，对英文按照分词字符进行分词；关闭后，对所有内容按照分词符进行分词。	-

配置	说明	示例
开启统计	是否开启统计分析功能。该功能默认开启。 开启之后，您可以结合查询语句和分析语句，对日志查询结果进行统计分析。	-

5. 单击确定，结束配置。

> **说明:**
>
> - 索引配置在1分钟之内生效。
> - 开启或修改索引后，新的索引配置只对新写入的数据生效。

4 查询日志

开启并配置索引后，可以在查询页面对采集到的日志进行实时查询与分析。

前提条件

- 已采集到日志数据。
- 已开启并配置索引。

操作步骤

1. 登录日志服务控制台，单击Project名称。
2. 单击日志库名称后的图标，选择查询分析。
3. 在搜索框中输入查询分析语句。

   查询分析语句由查询语句和分析语句构成，格式为查询语句|分析语句。详细说明请参考查询分析语句格式。
4. 在页面右上角单击15分钟（相对），设置查询的时间范围。

   您可以选择相对时间、整点时间和自定义时间范围。

> **说明:**
> 查询结果相对于指定的时间范围来说，有1min以内的误差。

5. 单击查询/分析，查看搜索结果。

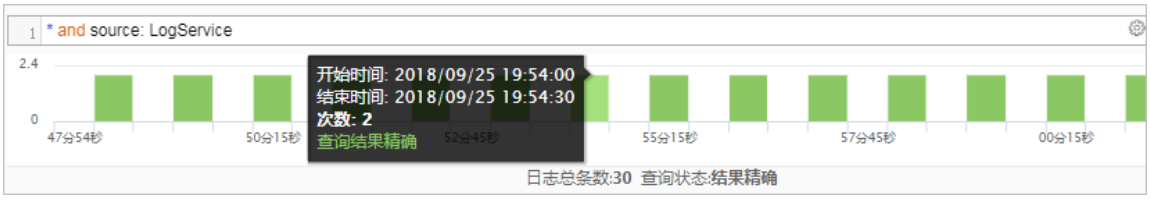
日志服务为您提供日志分布直方图、原始日志和统计图表形式的查询分析结果。

> **说明：**
> 默认返回100个结果，如果您需要返回更多，请使用#unique_44。

- 日志分布直方图：

  日志分布直方图主要展示查询到的日志在时间上分布。

  - 鼠标指向绿色的数据块，可以查看该数据块代表的时间范围和日志命中次数。
  - 单击数据块，可以查看更细时间粒度的日志分布，同时原始日志页签中也会同步展示指定时间范围内的日志查询结果。



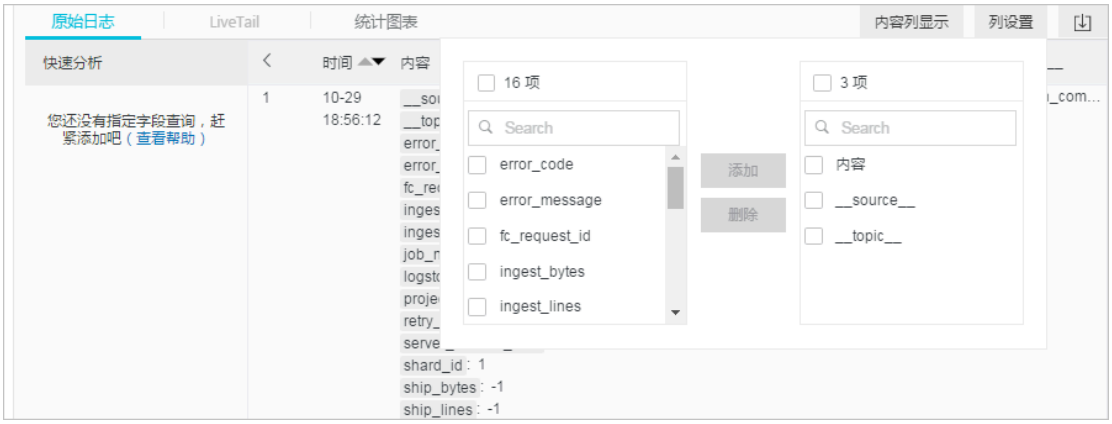
- 原始日志：

  原始日志页签展示当前查询结果，也就是当前查询条件命中的日志。

  - 快速分析：快速分析功能用于快速分析某一字段在一段时间内的分布情况，详细说明请查看#unique_54。
  - 下载日志：单击页签右上角的下载图标，选择下载的范围，并单击确定。
  - 设置列：单击页签右上角的列设置，勾选字段并单击添加，页签中会新增选中字段的列，其中列名称为字段名，内容为每条日志的字段值。

    > **说明：**

内容列需要被选中，页签中才会出现日志内容一列。

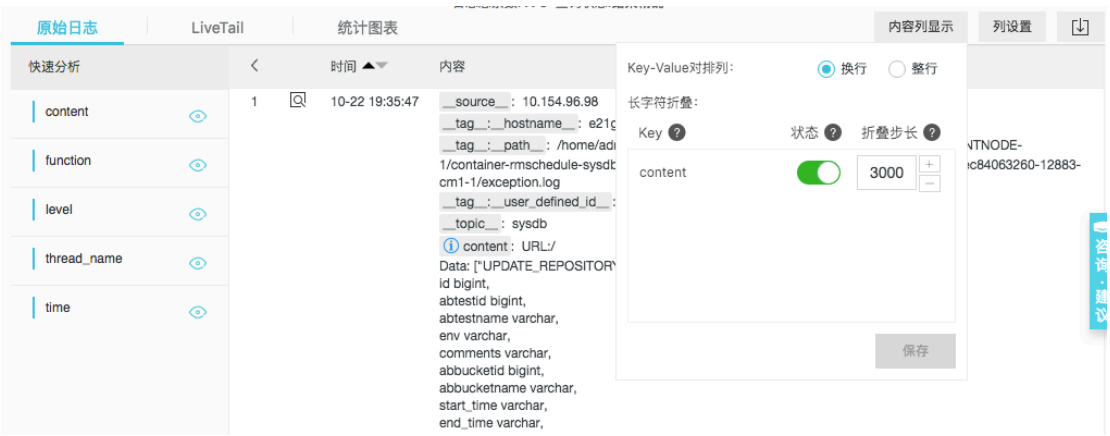


- 设置内容列显示：字段内容如果超出3000字符，会默认折叠处理，并在Key值前显示提醒信息“该字段过长，已做折叠处理”。单击页签右上角的内容列显示，设置Key-Value对排列和长字符折叠。

  说明：
  展开至10000个字符以上时，第10000以外的字符的将做降级处理，降级处理的字符不提供分词的功能。

配置		说明
Key-Value对排列		您可以设置Key-Value对之间为换行显示或整行显示。
长字符折叠	Key	当某一Value值超过3000字符时，默认为Value值设置折叠显示，如果日志中不存在过长的Value值，则此处为空。 Key为过长而被折叠的Value对应的Key。
	状态	是否开启Value值折叠。默认为开启状态。 ■ 开启：表示Key-Value对里的Value值长度超出折叠步长时，会自动对字符进行折叠。单击字符末尾展开按钮可以进行增量展开，增量为折叠步长。 ■ 关闭：表示超出折叠步长时不折叠。

配置		说明
	折叠步长	Value正常显示的最大长度，也是每次增量展开的长度。 单位为字符，取值范围为500~10000，默认为3000。



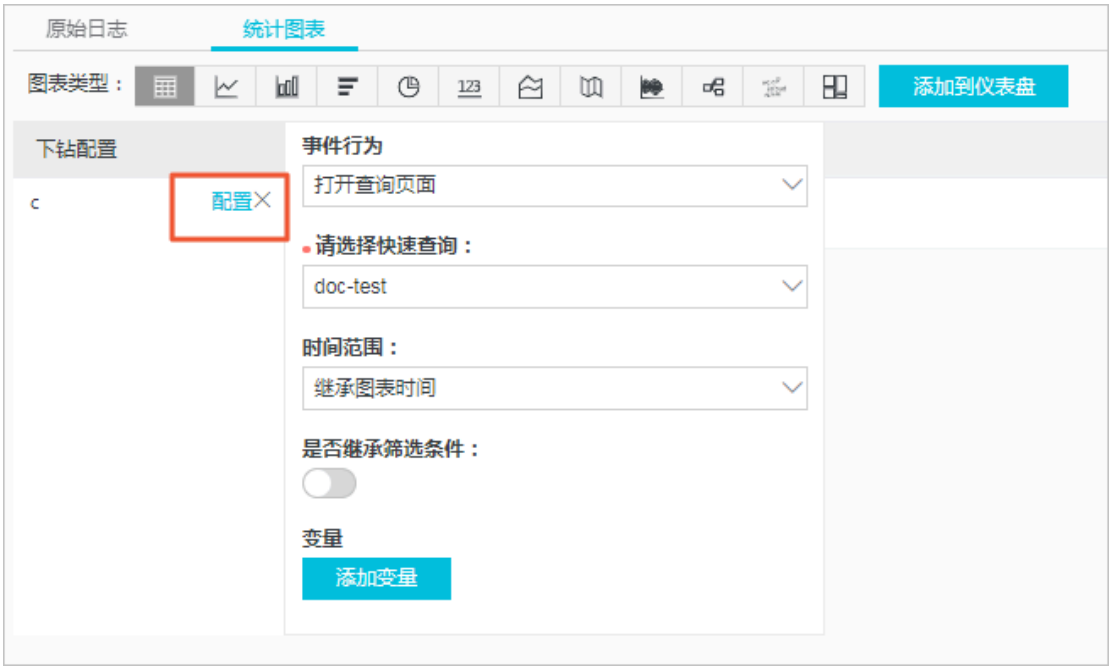
- 统计图表：

  如果在索引设置中开启了统计功能，且在搜索中使用查询分析语句，则可以在统计图表页签中查看分析结果。

  - 查看分析结果：日志服务为您提供表格、折线图、柱状图等多种类型的统计图表，您可以根据分析需求选用合适的图表类型展示分析结果。

  - 添加图表到仪表盘：仪表盘是日志服务提供的实时数据分析大盘。单击添加到仪表盘，将常用的查询语句以图表形式保存到仪表盘中。

- **设置下钻配置：下钻分析是在分析时加深维度，对数据进行层层深入的查看。设置下钻配置并将图表添加到仪表盘，在仪表盘中单击图表值可以获取更深维度的分析结果。详细说明请查看#unique_57。**



另外，在查询页面右上角单击另存为快速查询和另存为告警，可以使用日志服务的快速查询和告警功能。

5 导出日志

日志服务支持导出日志到本地，您可下载本页日志（CSV格式）或全部日志（TXT格式）到本地。

操作步骤

1. 登录日志服务控制台，单击Project名称。
2. 单击日志库名称后的 图标，选择查询分析。
3. 单击原始日志页签右侧的下载图标 打开日志下载对话框。

4. 选择下载类型。

- 下载本页日志

  单击下载本页日志以CSV格式将本页面的日志保存到本地。

- 下载所有日志

  单击通过命令行工具下载所有日志下载所有日志。

a. 安装命令行工具。具体安装说明请参考命令行工具CLI用户手册。

b. 单击安全信息管理查看并复制当前用户的秘钥ID和KEY。

c. 单击复制命令行并用当前用户的秘钥ID和KEY替换该命令行中`【步骤2中的秘钥ID】`和`【步骤2中的秘钥Key】`。

d. 在CLI命令行工具中执行该命令下载日志，日志保存文件为运行命令行当前目录下的*download_data.txt*。

6 索引数据类型

6.1 简介

日志服务支持对采集到的日志设置全文或字段索引。设置全文索引后，Value为整条日志；设置字段索引后，每个Key都可以设置数据类型。

数据类型

目前支持的索引数据类型如下：

查询类别	索引数据类型	说明	查询示例
基础查询	TEXT	文本类型，可以进行关键词+模糊匹配，支持中文分词。	`uri:"login*" method:"post"`
	Long	数值类型，支持区间查询。	`status>200, status in [200, 500]`
	Double	带浮点数数值类型。	`price>28.95, t in [20.0, 37]`
组合查询	JSON	内容为JSON字段，默认为Text类型，支持嵌套模式。可以通过 a.b等路径格式给a层下b元素设置（Text、Long、Double）类型索引，设置后的字段类型以设置为主。	`level0.key>29.95 level0.key2:"action"`
	文本	整条日志当做文本进行查询。	`error and "login fail"`

查询示例

以下一条日志除时间外，还包含4个键值：

序号	Key	类型
0	time	-
1	class	text
2	status	long
3	latency	double
4	message	json

```
0. time:2018-01-01 12:00:00
```

```
1. class:central-log
2. status:200
3. latency:68.75
4. message:
{
    "methodName": "getProjectInfo",
    "success": true,
    "remoteAddress": "1.1.1.1:11111",
    "usedTime": 48,
    "param": {
            "projectName": "ali-log-test-project",
            "requestId": "d3f0c96a-51b0-4166-a850-f4175dde7323"
    },
    "result": {
        "message": "successful",
        "code": "200",
        "data": {
            "clusterRegion": "ap-southeast-1",
            "ProjectName": "ali-log-test-project",
            "CreateTime": "2017-06-08 20:22:41"
        },
        "success": true
    }
}
```

索引设置如下：

图 6-1: 索引设置

其中：

- **①表示可查询json字段中所有string和bool数据。**
- **②表示可查询long类型数据。**
- **③表示配置的字段可进行SQL分析。**

示例：

1. 查询string、bool类型

   · json内字段无需配置。

   · json map、array自动展开，支持多层嵌套，每一层以”.”进行分割。

```
class : cental*
message.traceInfo.requestId : 92.137_1518139699935_5599
message.param.projectName : ali-log-test-project
message.success : true
```

2. 查询Double、Long类型

   需要对json内字段独立配置，字段必须不在array。

```
latency>40
message.usedTime > 40
```

3. 组合查询

```
class : cental* and message.usedTime > 40 not message.param.
projectName:ali-log-test-project
```

6.2 文本类型

和搜索引擎类似，文本类（Text）数据查询基于词（Term）的命中，因此需要配置分词符、大小写敏感，包含中文（中文分词）选项。

配置说明

· 大小写敏感

原始日志查询时是否区分大小写。例如原始日志为“internalError”：

- false（不区分），即查询关键字”INTERNALERROR”和“internalerror”都能查询到样例日志。
- true（区分），只能通过关键字“internalError”查询到样例日志。

- 分词符

  原始日志内容根据分词符可以将日志内容切分成多个关键词。

  例如我们要查询如下日志内容：

  ```
  /url/pic/abc.gif
  ```

  - 不设置任何分词符，整个字符串会作为一个独立单词`/url/pic/abc.gif`，只有通过该完整字符串，或通过模糊查询`/url/pic/*`才能找到。
  - 如果设置分词符为`/`，则原始日志被切分为`url`、`pic`和`abc.gif`三个单词，可以使用任意一个单词或单词模糊查询都可以找到该日志，例如`url`、`abc.gif`或`pi*`，也可以使用`/url/pic/abc.gif`进行查询(查询时会被拆分为`url and pic and abc.gif`三个条件)。
  - 如果设置分词符为`/.`，则原始日志被切分为`url`、`pic`、`abc`和`gif`四个单词。

  说明：

  通过设置合理的分词符，可以放宽查询的范围。

- 包含中文

  如果日志中包含中文，需要打开中文分词。例如如下日志内容：

  ```
  buyer:用户小李飞刀lee
  ```

  默认分词符为”:”，则原始日志会被拆分为`buyer`、`用户小李飞刀lee`这两个单词，如果搜索`用户`，则不会返回`lee`，如果开启包含中文选项后，日志服务后台分词器会智能去理解中文含义，并将日志拆分为`buyer`、`用户`、`小李`、`飞刀`和`lee`五个单词，无论使用`飞刀` 或`小李飞刀`（会被解析为：`小李 and 飞刀`) 都可以查找到日志。

  说明：

  中文分词对写入速度会有一定影响，请根据需求谨慎设置。

- 全文索引

  全文查询（索引）默认会将整条日志（除Time以外所有字段、包括Key）作为文本类型，全文查询默认不需要指定key。例如对以下由4个字段组成的日志（time/status/level/message）。

  ```
  [20180102 12:00:00] 200,error,some thing is error in this field
  ```

  - time:2018-01-02 12:00:00
  - level:”error”
  - status:200
  - message:”some thing is error in this field”

  说明:

  - 全文检索时不需要输入前缀，在检索过程中搜索error时（level和message两个字段中error都会被命中）。
  - 全文检索需要设置分词符，例如当设置分词符为“ ”时，可以“status:200”作为一个短语；如果分词符为“:“时，“status”和“200”分别会作为2个独立短语。
  - 数值类会被作为文本处理，例如200可以检索到该日志，时间字段（time) 不会被作为文本处理。
  - 当输入Key时整条日志也会被命中，例如“status”。

6.3 JSON类型

索引数据类型可设置为JSON类型，支持JSON格式日志的查询和分析功能。

JSON是由文本、布尔、数值、数组（Array）和图（Map）构成的组合类型数据。JSON数据作为一种通用类型的数据类型，其自解析、灵活的特性，使其能够很好满足复杂场景下数据的记录需求，在很多日志内容中格式不固定的部分往往都是以JSON的形式进行记录，例如将一次http请求的request参数和response内容以JSON的形式记录在一条日志中。

日志服务支持在索引中将字段设置为JSON类型，支持JSON格式日志的查询和分析。

配置说明

- 支持json格式解析，所有text、bool类型自动索引

  ```
  json_string.key_map.key_text : test_value
  json_string.key_map.key_bool : true
  ```

- 非json array中的double、long类型数据，可通过配置指定json路径后进行查询

  配置key_map.key_long这个字段的类型为long

```
查询 : json_string.key_map.key_long > 50
```

- 非json array中的text、double、long类型字段，可开启“统计分析”功能，进行sql分析

```
json_string.key_map.key_long > 10 | select count(*) as c ,
    "json_string.key_map.key_text" group by
    "json_string.key_map.key_text"
```

说明:

- 不支持json object、json array类型。
- 字段不能在json array中。
- bool类型字段可以转成text类型。
- 查询分析时，JSON类型字段需要用双引号括起来。

- 支持非完全合法json数据解析

  日志服务会尽可能解析有效内容，直到遇到非法部分结束。

  例如以下示例在key_3之后的数据被截断丢失，对于这种缺失的日志，日志服务可正确解析到json_string.key_map.key_2 这个字段。

```
"json_string":
{
     "key_1" :  "value_1",
     "key_map" :
      {
             "key_2" : "value_2",
             "key_3" : "valu
```

查询语法

指定Key查询需要加上JSON中父路径的前缀，文本、数值类查询语法与其他类型相同，详情请参见#unique_10。

查询示例

以下一条日志除时间外，还包含4个键值，其中"message"字段是json格式。

序号	Key	类型
0	time	-
1	class	text
2	status	long
3	latency	double
4	message	json

```
0. time:2018-01-01 12:00:00
```

```
1. class:central-log
2. status:200
3. latency:68.75
4. message:
{
    "methodName": "getProjectInfo",
    "success": true,
    "remoteAddress": "1.1.1.1:11111",
    "usedTime": 48,
    "param": {
            "projectName": "ali-log-test-project",
            "requestId": "d3f0c96a-51b0-4166-a850-f4175dde7323"
    },
    "result": {
        "message": "successful",
        "code": "200",
        "data": {
            "clusterRegion": "ap-southeast-1",
            "ProjectName": "ali-log-test-project",
            "CreateTime": "2017-06-08 20:22:41"
        },
        "success": true
    }
}
```

索引设置如下：

图 6-2: 索引设置

其中：

- **①表示可查询json字段中所有string和bool数据。**
- **②表示可查询long类型数据。**
- **③表示配置的字段可进行SQL分析。**

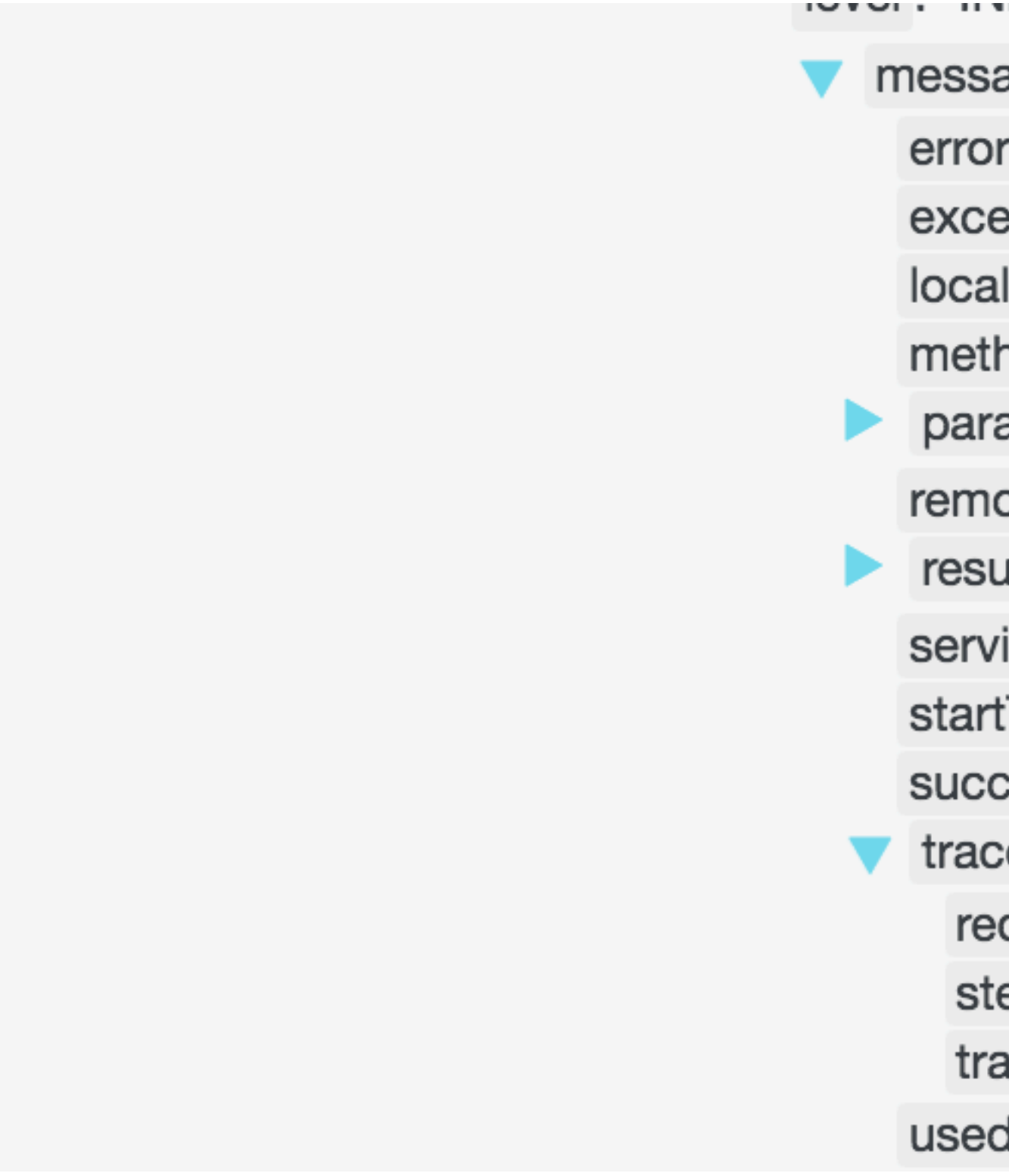
示例：

1. 查询string、bool类型

说明：

- json内字段无需配置。
- json map、array自动展开，支持多层嵌套，每一层以”.”进行分割。

```
message.traceInfo.requestId : 92.137_1518139699935_5599
message.param.projectName : ali-log-test-project
message.success : true
message.result.data.ProjectStatus : Normal
```



2. 查询Double、Long类型

> 说明:
> 需要对json内字段独立配置，字段必须不在array。

```
message.usedTime > 40
```

3. Sql 统计分析

> 说明:
> - 需要对json内字段独立配置，字段必须不在array。
> - 查询字段需要使用引号，或者设置别名。

```
* | select avg("message.usedTime") as avg_time ,
"message.methodName"  group by "message.methodName"
```

6.4 数值类型

在配置索引时，您可以将字段配置为数值类型，并通过数值范围查询键值。

配置说明

支持类型：`long`（长整数）或者`double`（小数），当设置为数值类型后对于该键的查询只能通过数值范围。

查询示例

查询键值范围为(1000 2000]的longkey，可以使用以下查询方式：

- 数值类查询语法，例如：

```
longKey > 1000 and longKey <= 2000
```

- 也可以使用区间查询语法，例如：

```
longKey in (1000 2000]
```

更多语法请参见#unique_10。

7 查询语法与功能

7.1 查询语法

为了能够帮助您更有效地查询日志，日志服务提供一套查询语法用于设置查询条件。

查询方式

#unique_68之后，在日志查询界面输入查询分析语句即可查询日志。

日志查询语句是查询分析语句的前半部分，指定日志查询时的过滤规则，返回符合条件的日志数据。查询语句中支持全文查询和字段查询。

- 全文查询

  全文查询时，将整条日志作为一个特殊的Key-Value对，Value为全部的日志内容。全文查询表示在日志内容中查询关键字，即指定查询条件为包含或不包含某个关键字，满足查询条件的日志会作为结果返回。

  除了普通的全文查询之外，日志服务还支持短语查询和模糊查询。

  - 普通全文查询：指定关键字和规则，包含该关键字并符合规则的日志会作为结果返回。

    例如`a and b`表示查询同时包含关键字`a`和`b`的日志。
  - 短语查询：如果需要查询的短语中包含空格，可以将短语用双引号（“”）包裹，表示将双引号中的内容作为一个完整的关键字查询。

    例如`"http error"`表示查询包含关键字`http error`的日志。
  - 模糊查询：指定一个64个字符以内的词，在词的中间或者末尾加上模糊查询关键字，即`*`和`?`，日志服务会在所有日志中为您查询到符合条件的100个词，返回包含这100个词并满足查询条件的所有日志。

    例如`addr?`表示在所有日志中查找以`addr`开头的100个词，并返回包含这些词的日志。
- 字段查询

  为字段都配置字段索引之后，可以指定字段名称和字段内容进行查询。对于double和long类型的字段，可以指定数值范围进行查询。例如设置字段查询语句为`Latency>5000 and Method:Get* and not Status:200`，表示查询`Latency`字段值大于5000、`Method`字段值为`Get`开头，且`Status`字段值不是200的日志。

  根据字段索引中设置的数据类型，您可以进行多种类型的基础查询和组合查询。字段查询示例请参考索引数据类型简介。

注意事项

- 同时配置全文查询和字段查询时，如果索引设置中两者的分词符不同，以字段索引设置为准，使用全文查询方式无法查出有效数据。
- 设置某字段的数据类型为double或long后，才能通过数值范围查询这些字段的数据。若未设置数据类型、或者数值范围查询的语法错误，日志服务会将该查询条件解释成全文索引，可能与您的期望的结果不同。
- 如果将某字段由文本类型改成数值类型，则修改索引之前采集到的数据只支持=查询。

运算符

查询语句支持如下运算符：

运算符	说明
and	双目运算符。格式为 `query1 and query2`，表示query1和query2 查询结果的交集。如果多个单词间没有语法关键词，默认是and 的关系。
or	双目运算符。格式为query1 or query2，表示query1和query2 查询结果的并集。
not	双目运算符。格式为query1 not query2，表示符合query1 并且不符合query2的结果，相当于query1-query2。如果只有not query1，那么表示从全部日志中选取不包含query1的结果。
(,)	左右括号用于把一个或多个子查询合并成一个查询条件，用于提高括号内查询条件的优先级。
:	用于 key-value 对的查询。term1:term2构成一个 key-value 对。如果 key 或者 value 内有空格、冒号:等保留字符，需要用双引号""把整个 key 或者 value 包括起来。
“	把一个关键词转换成普通的查询字符。左右引号内部的任何一个 term 都会被查询，而不会当成语法关键词。或者在 key-value 查询中把左右引号内的所有 term 当成一个整体。
\	转义符。用于转义引号，转义后的引号表示符号本身，不会当成转义字符，例如"\""。
|	管道运算符，表示前一个计算的基础上进行更多计算，例如 query1 | select count(1)。
count	计数运算符，表示日志条数。

运算符	说明
*	模糊查询关键字，用于替代 0 个或多个字符，例如：que*，会返回que 开头的所有命中词。 说明: 模糊查询最多返回100个符合关键词的日志。
?	模糊查询关键字，用于替代一个字符，例如qu?ry，会返回以qu开头，以ry结尾，并且中间还有一个字符的所有命中词。
__topic__	查询某个 topic 下数据，可以在 query 中查询 0 个或多个 topic 的数据，例如__topic__:mytopicname。
__tag__	查询某个 tag key 下某个 tag value，例如__tag__:tagkey:tagvalue。
source	查询某个 IP 的数据，例如source:127.0.0.1。
>	查询某个字段下大于某个数值的日志，例如latency > 100。
>=	查询某个字段下大于或等于某个数值的日志，例如latency >= 100。
<	查询某个字段下小于某个数值的日志，例如latency < 100。
<=	查询某个字段下小于或等于某个数值的日志，例如latency <= 100。
=	查询某个字段下等于某个数值的日志，例如latency = 100。
in	查询某个字段处于某个范围内的日志，使用中括号表示闭区间，使用小括号表示开区间，括号中间使用两个数字，数字中间为若干个空格。仅支持小写字符in。例如latency in [100 200]或 latency in (100 200]。

说明:

- 运算符不区分大小写。
- 运算符的优先级由高到底排序为:>">( )>and>not>or。
- 日志服务保留以下运算符的使用权，如果您需要使用以下运算符作为查询关键字，请使用双引号包裹起来：sort、asc、desc、group by、avgsum、min、max和limit。

查询示例

查询需求	例句
同时包含 a 和 b 的日志	a and b 或者 a b
包含 a 或者包含 b 的日志	a or b

查询需求	例句
包含 a 但是不包含 b 的日志	`a not b`
所有日志中不包含 a 的日志	`not a`
查询包含 a 而且包含 b，但是不包括 c 的日志	`a and b not c`
包含 a 或者包含 b，而且一定包含 c 的日志	`(a or b ) and c`
包含 a 或者包含 b，但不包括 c 的日志	`(a or b ) not c`
包含 a 而且包含 b，可能包含 c 的日志	`a and b or c`
FILE 字段包含 apsara的日志	`FILE:apsara`
FILE 字段包含 apsara 和 shennong 的日志	`FILE:"apsara shennong"` 或者 `FILE :apsara FILE: shennong` 或者 `FILE: apsara and FILE:shennong`
包含 and 的日志	`and`
FILE 字段包含 apsara 或者 shennong 的日志	`FILE:apsara or FILE:shennong`
file info 字段包含 apsara 的日志	`"file info":apsara`
包括引号的日志	`\"`
查询以 shen 开头的所有日志	`shen*`
查询 FILE 字段下，以 shen 开头的所有日志	`FILE:shen*`
查询 FILE 字段下，值为shen*的所有日志	`FILE: "shen*"`
查询以 shen 开头，以 ong 结尾，中间还有一个字符的日志	`shen?ong`
查询包括以 shen 开头，并且包括以 aps 开头的日志	`shen* and aps*`
查询 topic1 和 topic2 下的所有数据	`__topic__:topic1 or __topic__ : topic2`
查询 tagkey1 下 tagvalue2 的所有数据	`__tag__ : tagkey1 : tagvalue2`
查询latency大于等于100，并且小于200的所有数据	`latency >=100 and latency < 200`或 `latency in [100 200)`
查询latency 大于100的所有请求	`latency > 100`
查询不包含爬虫的日志，并且http_referer中不包含opx的日志	`not spider not bot not http_referer :opx`
查询cdnIP字段不为空的日志	`not cdnIP:""`

查询需求	例句
查询cdnIP字段不存在的日志	`not cdnIP:*`
查询存在cdnIP字段的日志	`cdnIP:*`

指定或跨 Topic（日志主题） 查询

每个 Logstore 根据 Topic 可以划分成一个或多个子空间，当进行查询时，指定 topic 可以限定查询范围，达到更快速度。因此我们推荐对 logstore 有二级分类需求的用户使用 topic 进行划分。

当指定一个或多个 topic 进行查询时，仅从符合条件的 topic 中进行查询。但不输入 topic，默认查询所有 topic 下的数据。

例如，使用Topic来划分不同域名下日志：

图 7-1: 日志Topic

Topic 查询语法：

- 支持查询所有 topic 下的数据，在查询语法和参数中都不指定 topic 意味着查询所有 topic 的数据。
- 支持在 query 中查询 topic，查询语法为 `__topic__:topicName`。同时仍然支持旧的模式，在 url 参数中指定 topic。
- 支持查询多个 topic，例如 `__topic__:topic1 or __topic__:topic2` 表示查询 topic1 和 topic2 下的数据的并集。

模糊查询

日志服务支持单词模糊查询，指定一个64个字符以内的词，在词的中间或者末尾加上模糊查询运算符，即`*`和`?`，日志服务会在所有日志中为您查询到符合条件的100个词，并返回包含这100个词并满足查询条件的日志。

限制说明：

- 查询时必须指定前缀，即`*`和`?`不能出现在词的开头。
- 指定的词越精确，查询结果越精确。
- 查询的词超过64个字符，无法使用模糊查询。建议您把查询的词长度缩小到64个字符以下。
- 模糊查询最多返回100个符合关键词的日志。

7.2 LiveTail

LiveTail是日志服务在控制台提供了日志数据实时监控的交互功能，帮助您实时监控日志内容、提取关键日志信息。

背景信息

在线上运维的场景中，往往需要对日志队列中进入的数据进行实时监控，从最新的日志数据中提取出关键的信息进而快速地分析出异常原因。在传统的运维方式中，如果需要对日志文件进行实时监控，需要到服务器上对日志文件执行命令`tail -f`，如果实时监控的日志信息不够直观，可以加上`grep`或者`grep -v`进行关键词过滤。日志服务在控制台提供了日志数据实时监控的交互功能LiveTail，针对线上日志进行实时监控分析，减轻运维压力。

功能优势

- 监控日志的实时信息，标记并过滤关键词。
- 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。

限制说明

- LiveTail功能仅支持Logtail采集到的日志数据。
- 已成功采集到日志数据后，才能使用LiveTail功能。

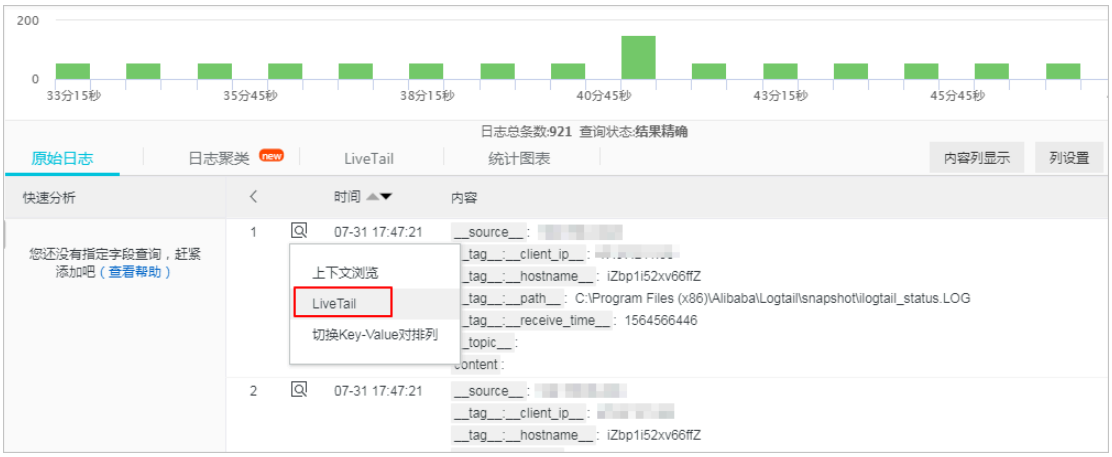
使用LiveTail实时监控日志

1. 登录日志服务控制台，单击Project名称。
2. 单击日志库名称后的 图标，选择查询分析。

3. 您可以通过以下两种方式使用LiveTail功能。

   · 快捷开启LiveTail。

   a. 在原始日志页签中，单击指定原始日志的序号右侧图标 ，并选择LiveTail。



   b. 系统为您自动开启LiveTail，并开始计时。

   其中，来源类型、机器名称和文件名称已预设为指定原始日志的信息。

   开启LiveTail后，Logtail采集到的日志数据会实时显示排列在页面中。最新的日志数据始终在页面底部，且滚动条默认在最下方，即显示最新数据。页面最多显示1000条数据，满1000条后页面自动刷新并重新填充日志数据。

   c. （可选）在搜索框中输入关键词。

   包含关键词的日志才会显示在监控列表中。筛选包含关键词的日志，便于对特定日志进行实时内容监控。

d. 在实时监控日志的时候，如果某些日志数据可能存在异常需要分析的时候，单击停止LiveTail。

停止LiveTail后，LiveTail计时结束，不再实时更新日志数据。

针对监控日志时发现的异常，日志服务提供多种分析方式，详细信息请参考使用LiveTail分析日志。

- 自定义设置LiveTail。

a. 单击LiveTail页签。

b. 配置LiveTail。

配置	是否必选	说明
来源类型	必选	日志的来源，包括： - 普通日志 - Kubernetes - Docker
机器名称	必选	日志来源服务器的名称。
文件名称	必选	日志文件的完整路径及文件名。

配置	是否必选	说明
过滤条件	可选	关键词，设置关键词后，只有包含关键词的日志才会显示在实时监控窗口中。

c. 单击开启LiveTail。

开启LiveTail后，Logtail采集到的日志数据会实时显示排列在页面中。最新的日志数据始终在页面底部，且滚动条默认在最下方，即显示最新数据。页面最多显示1000条数据，满1000条后页面自动刷新并重新填充日志数据。

d. 在实时监控日志的时候，如果某些日志数据可能存在异常需要分析的时候，单击停止LiveTail。

停止LiveTail后，LiveTail计时结束，不再实时更新日志数据。

针对监控日志时发现的异常，日志服务提供多种分析方式，详细信息请参考使用LiveTail分析日志。

使用LiveTail分析日志

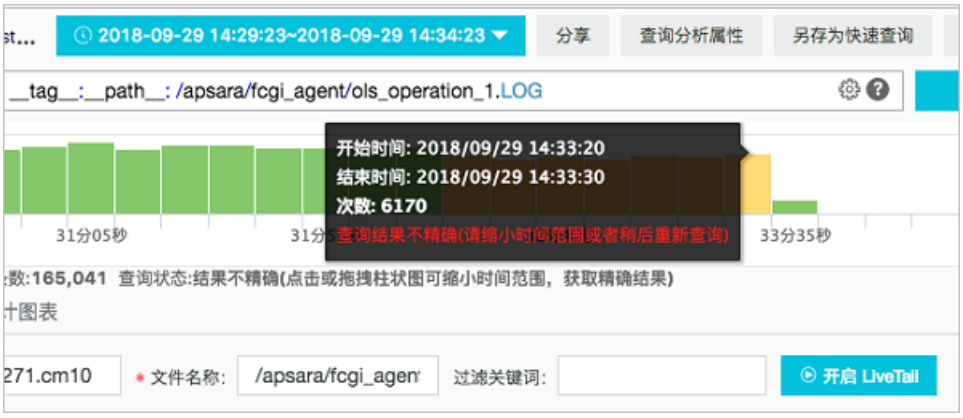
停止LiveTail之后，实时监控窗口不再更新显示日志内容，可以对监控过程中发现的问题进行进一步分析排查。

- 查看包含指定字段的日志内容。

所有的字段都进行过分词处理，单击指定异常字段内容，页面自动跳转到原始日志页签中，按照关键词筛选出该字段相关的所有日志内容。另外也可以对包含该关键字的日志进行上下文查询、查看统计图表等方式进行分析。

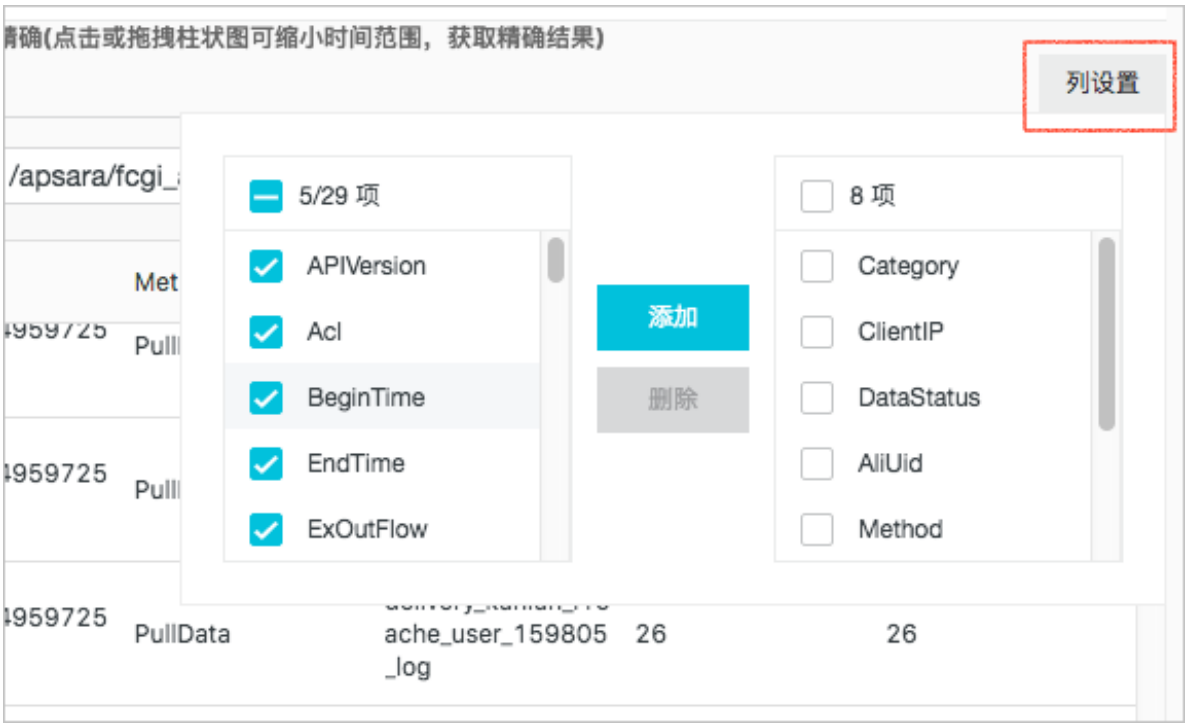
- 根据日志分布直方图（histogram）缩小查询的时间范围。

  LiveTail开启时，日志分布直方图也在进行同步更新。如果发现某个时段的日志分布有异常，例如日志数量显著增加时，可以单击该时段的绿色矩形缩小查询的时间范围。跳转后的原始日志内的时间轴与LivaTail内选择的点击的时间轴是相关联的，可以查看在这段时间内所有的原始日志内容及详细的时间分布。



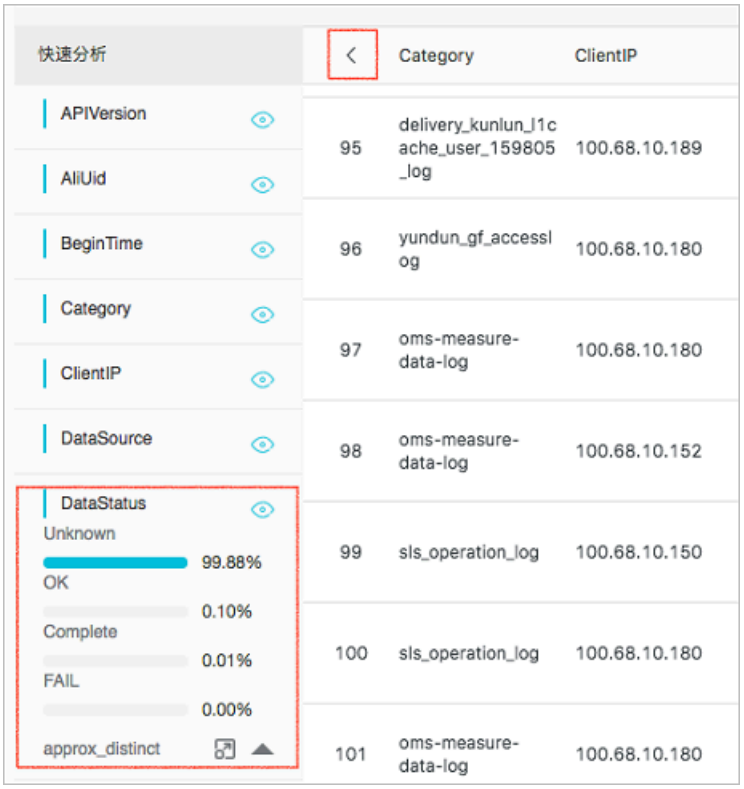
- 通过列设置强调关键信息。

  LiveTail页签中，单击日志列表右上角的列设置可以将指定字段单独选设置为一列，使该列的数据更加醒目。可以将需要重点关注的数据设为一列，便于查看和判断异常。



- 对日志数据进行快速分析。

  LiveTail页签中，单击日志列表左上角的箭头，可以展开快速分析区域。快速分析的时间区间是LiveTail开启到停止的时间段，分析的功能与原始日志内提供的快速分析相同。详细说明请参考#unique_54。



示例

以下视频为您展示如何使用LiveTail进行实时日志监控及分析。

单击观看

7.3 日志聚类

日志聚类，指采集文本日志时，将相似度高的数据聚合在一起，提取共同的日志Pattern，快速掌握日志全貌。

日志服务提供日志聚类功能，支持多种格式的文本日志聚合，可应用于DevOps中的问题定位、异常检测、版本回归等运维动作，或应用于安全场景下的入侵检测等。您还可以将聚类结果以分析图表的形式保存在仪表盘中，实时查看聚类数据。

功能优势

- 支持任意格式日志：Log4J、Json、单行（syslog）。
- 亿级数据，秒级输出结果。

- 日志经任意条件过滤后再聚类。
- 对聚类后Pattern，根据签名反查原始数据。
- 不同时间段Pattern比较。
- 动态调整聚类精度。

计费标准

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

原始日志大小	索引比例	日志聚类功能产生的索引量	索引总量
100GB	20%（20 GB）	100 * 10%	30GB
100GB	40%（40 GB）	100 * 10%	50GB
100GB	100%（100 GB）	100 * 10%	110GB

开启日志聚类功能

日志聚类功能默认为关闭状态，使用前请手动开启。

1. 登录日志服务控制台，单击Project名称。
2. 单击日志库名称后的 图标，选择查询分析。

3. 如果设置过索引，单击查询分析属性 > 设置。如果没有设置过索引，单击开启索引。

图 7-2: 开启索引

图 7-3: 修改索引

4. 设置索引属性，并开启日志聚类的功能开关。

   图 7-4: 开启日志聚类

5. 单击确定。

   成功开启日志聚类功能后，日志服务会对采集到的日志数据进行自动聚类，您可以：

   - 查看聚类结果和原始日志
   - 调整聚类精度
   - 对比不同时间段的聚类日志数量

查看聚类结果和原始日志

1. 在查询分析页面的查询框中输入查询语句，并单击查询/分析。

   可以通过关键字过滤日志，对包含或不包含指定关键字的日志数据进行自动聚类。

   > 说明:
   > 不支持SQL语句，即不能对分析结果进行聚类。

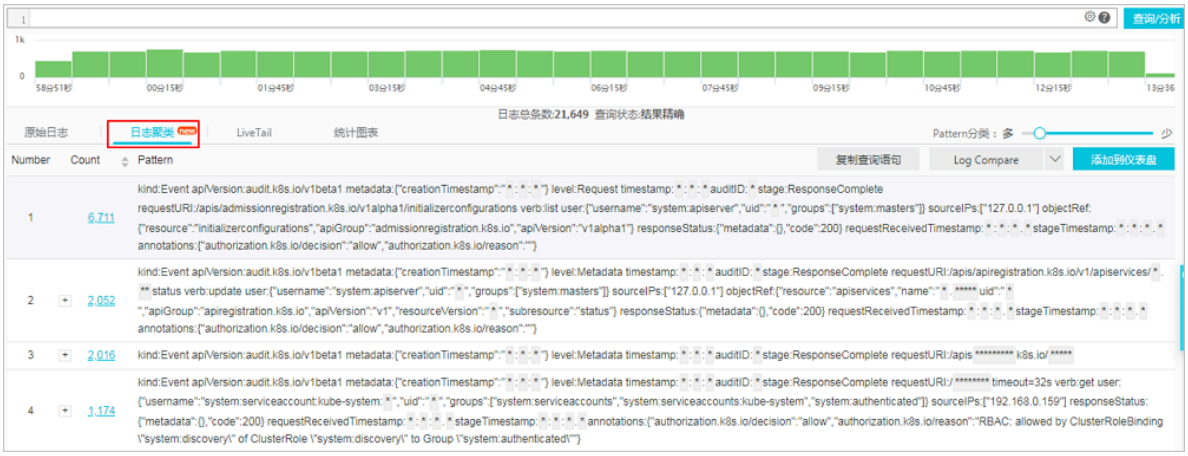
2. 单击进入日志聚类页签，查看聚类结果。

   日志聚类页签展示了过滤后的日志聚类结果。

显示项	说明
Number	聚类序号。
Count	该聚类类别的日志条数。

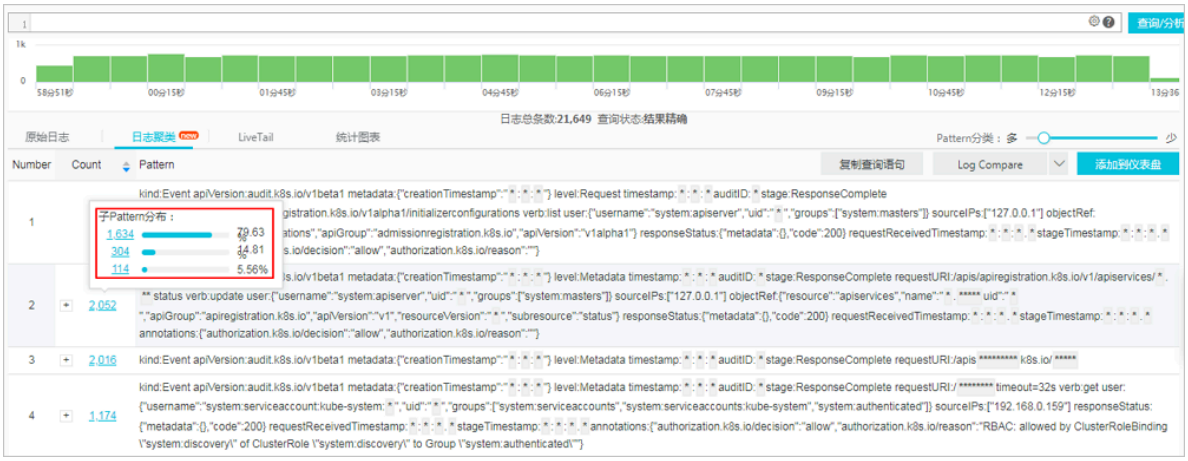
显示项	说明
Pattern	具体的日志模式，每个聚类会有一个或多个子Pattern。

图 7-5: 聚类结果



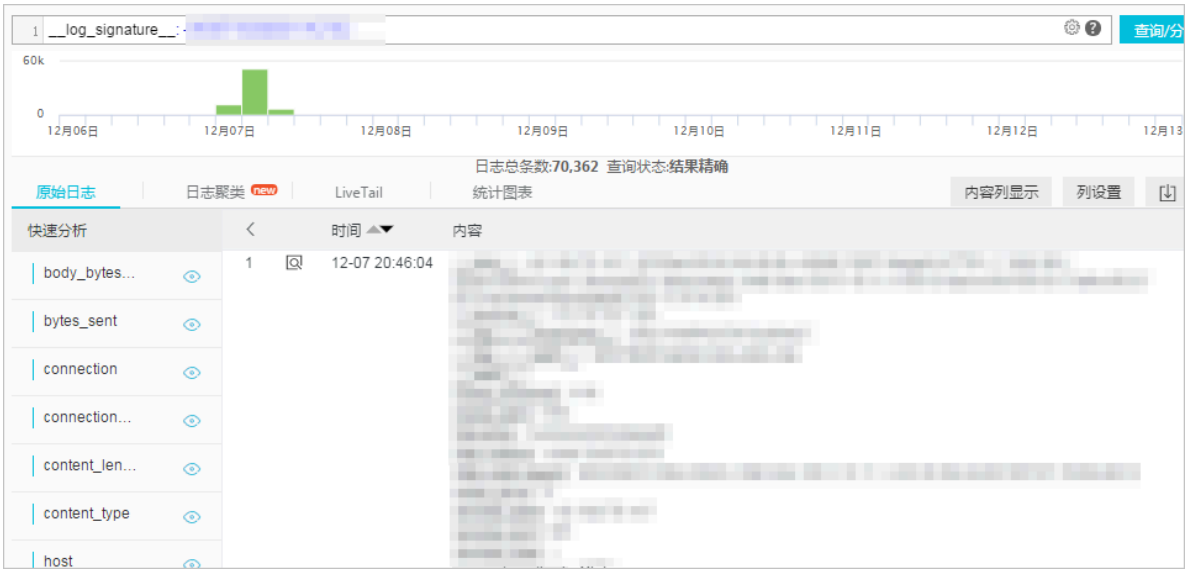
3. 鼠标指向Count列，有浮动栏展示当前聚类的子Pattern、每个子Pattern的占比。也可以单击数字前的加号+，展开子Pattern列表。

图 7-6: 查看聚类详情



4. 单击子Pattern前的Count值，查看该分类中的原始日志。

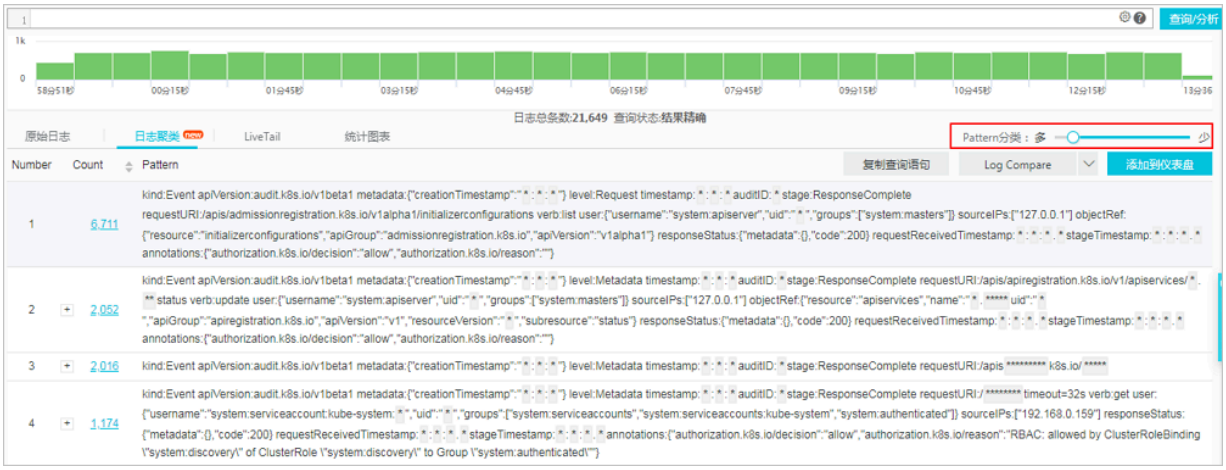
图 7-7: 查看原始数据



调整聚类精度

1. 在查询分析页面单击进入日志聚类页签。
2. 在页签右上角的Pattern分类中拖拽滑动条，调整聚类的精度。
   - 聚类偏向于多，表示聚类结果分类细、Pattern保留的细节多。
   - 聚类偏向于少，表示聚类结果分类粗，Pattern细节被隐藏得更多。

图 7-8: 调整聚类精度



对比不同时间段的聚类日志数量

单击Log Compare，设置对比时间后单击确定按钮。

显示项	说明
Number	聚类编号。
Pre_Count	当前Pattern在之前时段的原始日志数量。
Count	当前Pattern在当前时段的原始日志数量。
Diff	当前Pattern在两个时段的日志数量差值。
Pattern	某类日志具体的Pattern结果 。

API示例

- 获取日志聚类结果
  - SQL语句：

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

> 说明:
> 查看聚类结果时，您也可以通过复制查询语句按钮来获取日志聚类结果的SQL语句。

  - 输入参数：log_reduce(precision)

    precision： 接收整数[1~16]作为聚类精度，数字越小，聚类精度越高，生成的Pattern格式越多，默认为3。

  - 返回字段：
    - pattern： 某类日志具体的Pattern结果。
    - count : 某类日志的个数。
    - signature： 某类日志Pattern的签名。
    - origin_signatures： 这类日志的二级签名，可以通过这个二级签名，反查原始数据。

- 对比不同时间段日志聚类结果
  - SQL语句

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

> 说明:
> 通过Log Compare按钮来对比日志聚类结果后，您也可以通过复制查询语句按钮来获取对应的SQL语句。

  - 输入参数：compare_log_reduce(precision, compare_interval)
    - precision：接收整数[1~16]作为聚类精度，数字越小，聚类精度越高，生成的Pattern格式越多，默认为3。
    - compare_interval：正整数，表示对比多少秒之前的数据。
  - 返回字段：
    - pattern：某类日志具体的Pattern结果 。
    - signature：某类日志Pattern的签名。
    - count： 某类日志的个数。
    - count_compare : 对比时间段同signature日志个数。
    - diff： count和count_compare的差值。

日志聚类视频

日志聚类操作介绍视频：

7.4 上下文查询

当您展开一份日志文件，每一条日志都记录一个事件，并且往往不是孤立存在的，连续的若干条日志可以回放整个事件序列的发生过程。

日志上下文查询是指定日志来源（机器 + 文件）和其中一条日志，将该日志在原始文件中的前若干条（上文）或后若干条日志（下文）也查找出来，尤其是在 DevOps 场景下对于理清问题来龙去脉来说可谓是一把利器。

日志服务控制台提供专门的查询页面，您可以在控制台查看指定日志在原始文件中的上下文信息，体验类似于在原始日志文件中向上或向下翻页功能。通过查看指定日志的上下文信息，您可以在业务故障排查中快速查找相关故障信息，方便定位问题。

应用场景

例如，O2O 外卖网站在服务器上的程序日志里会记录一次订单成交的轨迹：

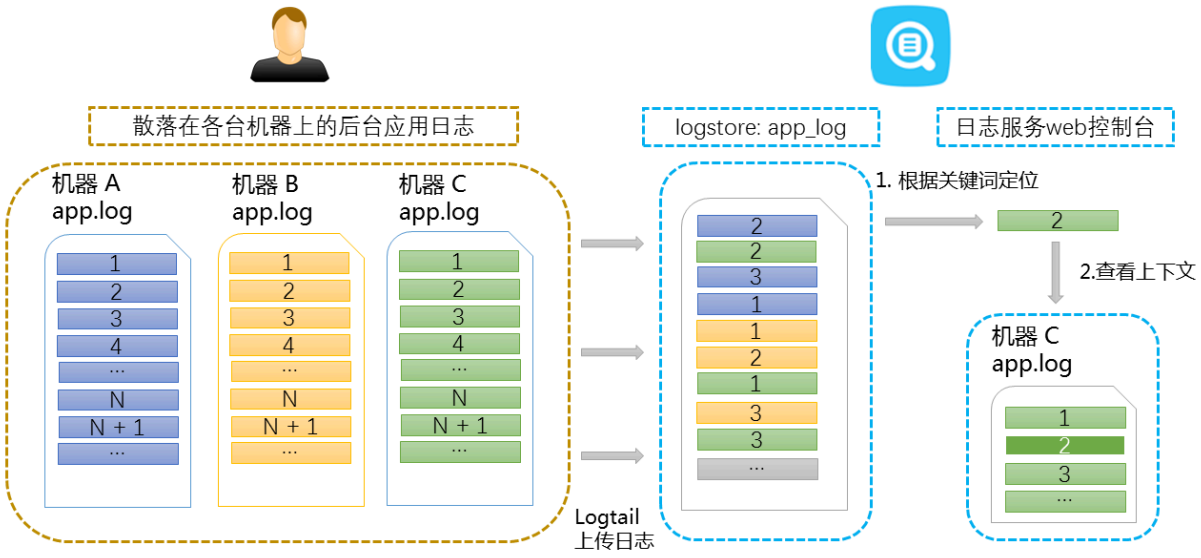
用户登录 > 浏览商品 > 点击物品 > 加入购物车 > 下单 > 订单支付 > 支付扣款 > 生成订单

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

在日志服务中，可以按照以下步骤排查：

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

图 7-9: 应用场景



功能优势

- 不侵入应用程序，日志文件格式无需改动。
- 在日志服务控制台上可以查看任意机器、文件的指定日志上下文信息，解放了过去需要登录每台机器查看日志文件的痛苦。

- 结合事件发生的时间线索，在日志服务控制台指定时间段快速定位可疑日志后再进行上下文查询，往往可以事半功倍。
- 不用担心服务器存储空间不足或日志文件轮转（rotate）造成的数据丢失，到日志服务控制台上随时可以查看过往的数据。

前提条件

- 使用 Logtail 采集日志 上传数据到日志库，除创建机器组、采集配置以外无需其他配置。或使用Producer相关的SDK上传，例如 Producer Library、Log4J、LogBack、C-Producer Library等。
- 开启索引。

> **说明:**
> 上下文查询功能暂不支持syslog日志。

操作步骤

1. 登录日志服务控制台，单击Project名称。
2. 单击日志库名称后的图标，选择查询分析。
3. 输入您的查询分析语句，选择查询时段并单击查询/分析。

   查询结果页中任一条日志的左侧有上下文浏览按钮，表明该日志支持上下文查看功能。

   图 7-10: 查询日志

4. 选中一条日志，单击上下文浏览。在右侧弹出页面中查看目标日志的上下文日志。

5. 使用鼠标在当前页面上下滚动查看选中日志周边的日志信息。如需要继续查看上文和下文，单击 更早 或 更新 进行翻页浏览。

图 7-11: 查询日志

7.5 快速查询

快速查询是日志服务提供的一键查询分析功能。

前提条件

已开启并设置索引。

背景信息

当您需要经常查看某一查询分析语句的结果时，可以将其另存为快速查询，下次进行该查询动作时，不需要手动输入查询语句，只需在查询页面左侧单击该快速查询的名称，即可再次进行该项查

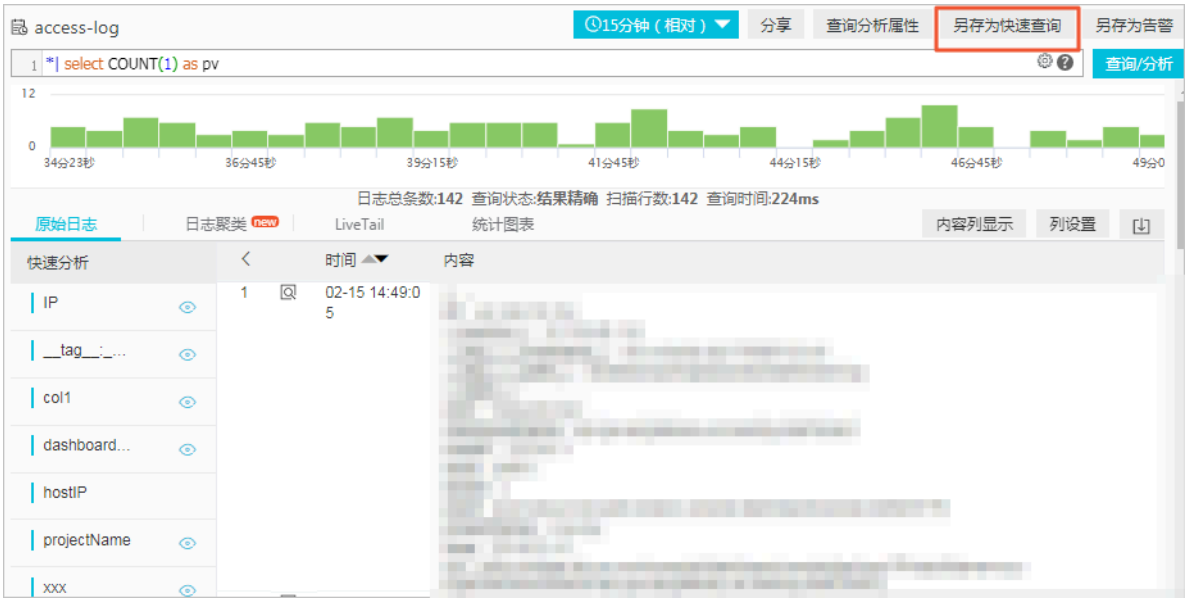
询动作。您也可以在告警规则中使用该快速查询条件。日志服务会定期执行该快速查询语句，并在查询结果满足预设条件时发送告警信息。

设置#unique_57时，如果需要将下钻事件设置跳转到快速查询，必须提前配置快速查询，并在查询语句中设置占位符。

> **说明:**
>
> 如何修改快速查询?
>
> 修改快速查询时，输入新的查询分析语句之后，请单击查询/分析，执行一次查询后再单击修改已有快速查询。

操作步骤

1. 登录日志服务控制台，单击Project名称。
2. 单击日志库名称后的图标，选择查询分析。
3. 输入您的查询分析语句，设置时间范围，并单击查询/分析。
4. 单击页面右上角的另存为快速查询。



5. 设置快速查询属性。

a) 设置快速查询名称。

- 名称仅支持小写字母、数字、连字符（-）和下划线（_）。
- 必须以小写字母或数字开头和结尾。
- 名称长度为3~63个字符。

b) 确认日志库、日志主题和查询语句。

若日志库和日志主题不符合您的需求，请返回至查询页面进入正确的日志库并输入查询语句，再次单击另存为快速查询。

c) （可选）划选查询语句的部分内容，并单击生成变量。

生成的变量为占位符变量。您可以在变量名中为占位符变量命名。默认值是您划选时选中的词。

> **说明:**
> 如果其他图表的下钻事件为跳转到这个快速查询，且图表配置的变量和快速查询的变量名相同，单击其他图表时会执行跳转，占位符变量的默认值替换为触发下钻事件的图表值，并以替换变量后的查询语句执行查询。详细信息请查看#unique_57。

6. 单击确定，结束配置。

7.6 快速分析

日志服务（Log Service）快速分析功能给用户提供了一键式交互查询体验，帮助用户快速分析某一字段在指定时间内的分布情况，降低用户索引关键数据的成本。

功能特点

- 支持Text类型字段前100000条数据的前十项分组统计。
- 支持Text类型字段快速生成approx_distinct查询语句。
- 支持long或者double类型字段近似分布直方统计。
- 支持long或者double类型字段快速查找最大项、最小项、平均值或总和。
- 支持将快速分析查询生成查询语句。

前提条件

快速分析需要用户指定字段查询属性。

1. 指定字段查询需要先开启索引以开启查询分析功能，如何开启索引请参考#unique_15。
2. 设置日志中的key为字段名称，并设置类型、别名、分词符等。

   如访问日志中存在request_method和request_time字段，可以参考如下设置。

   图 7-12: 指定字段查询

使用指南

设置好指定字段查询后，您可以在查询页面的原始日志页签左侧快速分析一栏处查看到对应的字段。序号顶部按钮可以进行页面折叠，单击眼睛图标即可根据当前时间区间、当前的$Search条件进行快速分析。

图 7-13: 原始日志

Text类型

- Text类型分组统计

  单击目标字段右侧的眼睛图标，快速对该Text类型字段前100,000条数据进行分组，并返回前十项的占比。

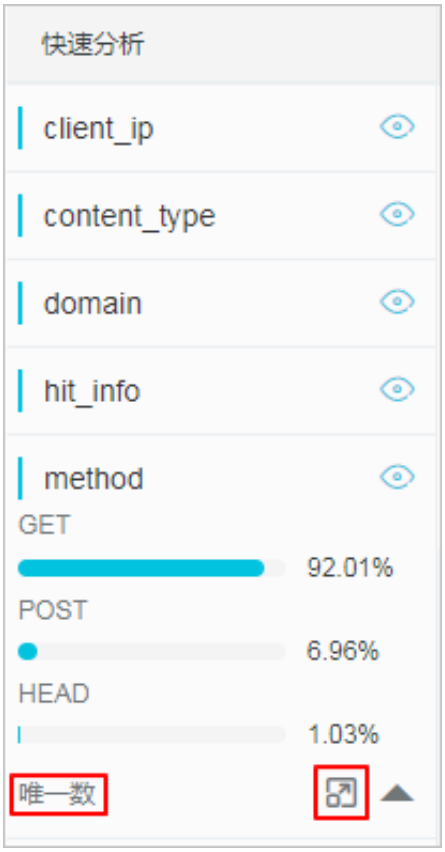
  查询语句如下：

```
$Search | select ${keyName} , pv, pv *1.0/sum(pv) over() as
percentage from( select count(1) as pv , "${keyName}" from (select
```

```
 "${keyName}" from log limit 100000) group by "${keyName}" order by
pv desc) order by pv desc limit 10
```

request_method按照分组统计可以得到如下结果，GET请求占大多数：

图 7-14: 分组统计



- 检查字段唯一项的个数

  在快速分析一栏的目的字段下单击唯一数，即可进行检查操作，即检查${keyName}唯一项的个数。

  request_method按照分组统计可以得到如下结果，GET请求占大多数。

- 将分组统计的查询语句扩展到搜索框

  单击唯一数右侧的 按钮，将分组统计的查询语句扩展到搜索框，便于进一步操作。

long/double类型

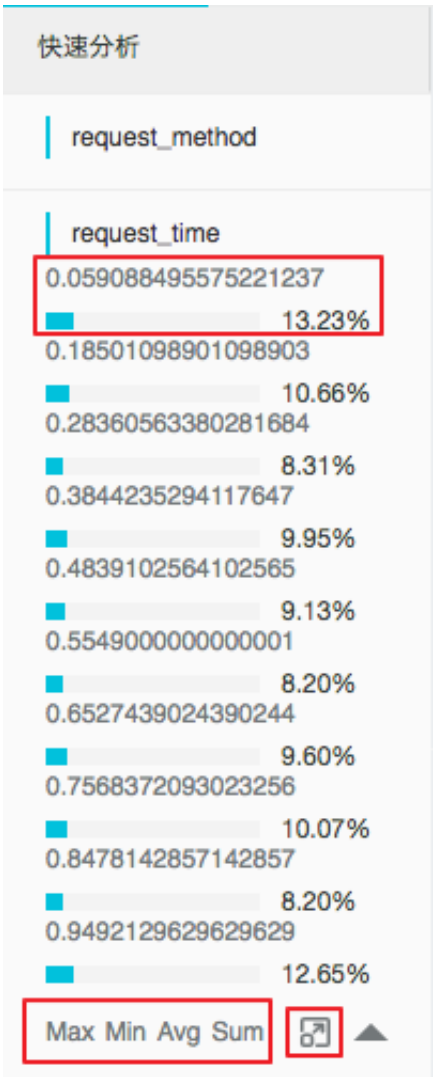
- 近似分布直方统计

  long/double类型由于存在多种类型值，计算分组统计意义不大，我们分为10个桶进行近似分布直方统计，查询语句如下：

  ```
  $Search | select numeric_histogram(10, ${keyName})
  ```

  request_time按照近似分布直方统计可以得到如下结果，可以知道绝大多数请求时间分布在0.059周围：

  图 7-15: 请求分布

- MaxMinAvgSum语句快速分析

  分别单击目标字段下的Max、Min、Avg、Sum，快速查找所有项中的最大项、最小项、平均值和总和。

- 将分组统计的查询语句扩展到搜索框

  单击Sum右侧的 按钮，将近似分布直方统计的查询语句扩展到搜索框，便于进一步操作。

7.7 其他功能

日志服务查询分析功能除了提供日志内容的各种语句查询能力以外，还提供原始日志、统计图表、上下文查询、快速分析、快速查询、仪表盘、告警等多种扩展功能。

- 原始日志
- 统计图表
- 上下文查询
- 快速分析
- 快速查询
- 仪表盘
- 另存为告警

原始日志

开启索引后，在检索框输入关键字、选择查询时段，单击查询/分析后，即可看到日志数量的直方图、原始日志及统计图表。

日志数量的直方图即日志检索的命中数量在时间上的分布，您可以通过直方图查看某个时间段的日志数量变化，单击长方形区域可以缩小时间范围，查看长方形区域表示的时间范围内的日志命中情况，为您的日志检索结果提供更精细的展示方式。

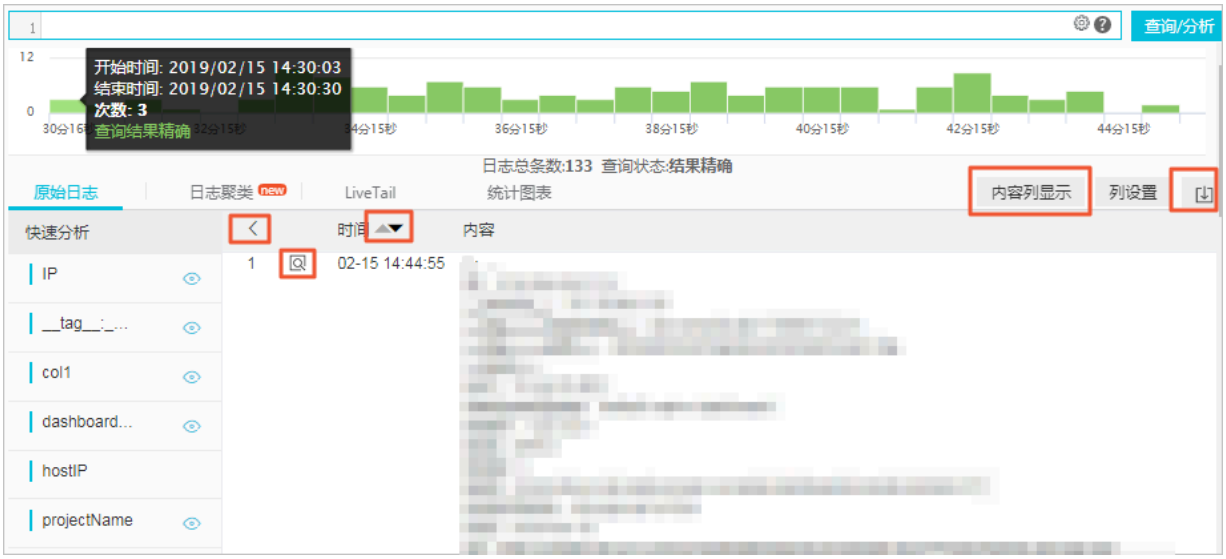
在原始日志页签中，您可以按时间排序，查看被命中的日志内容。

- 单击列名时间旁的三角符号，可切换时间正序或时间倒序。
- 单击列名内容列显示，可以选择日志内容换行或者整行显示，或设置长字符串折叠。
- 单击日志内容中的value关键字，可以查看包含该关键字的所有日志内容。
- 单击原始日志页签右上角的 图标，可下载CSV格式的查询结果，单击列设置图标，可在原始日志显示结果中增加字段的显示列，您可以更直观地在新增列中查看每条原始日志的目标字段内容。
- 单击上下文浏览查看该日志的前后各15条日志。更多信息请参考#unique_81。

  说明:

上下文查询功能目前仅支持使用Logtail上传的数据。

图 7-16: 原始日志



统计图表

开启索引并输入查询和分析语句后，您可以在统计图表页签中查看日志的统计结果。

- 提供表格、折线图等多种统计图表。

  支持根据统计分析的需要选择统计图表类型，支持多种自定义配置。
- 支持将统计图表添加到仪表盘，详细内容请参考#unique_56。
- 支持为统计图表设置#unique_57，该图表添加到仪表盘之后，单击图表数据即触发下钻事件，深化查询维度。

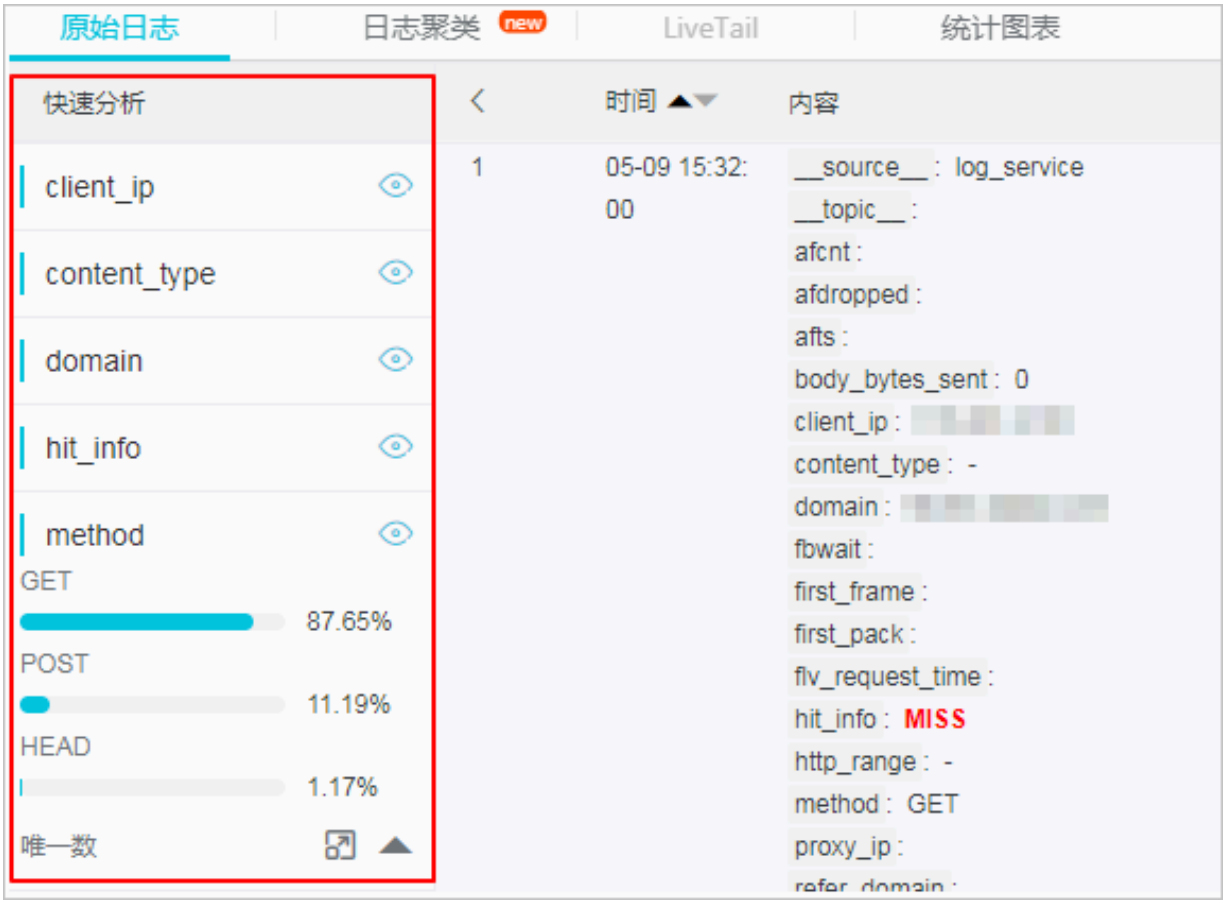
图 7-17: 统计图表

上下文查询

日志服务控制台提供专门的查询页面，您可以在控制台查看指定日志在原始文件中的上下文信息，体验类似于在原始日志文件中向上或向下翻页功能。通过查看指定日志的上下文信息，您可以在业务故障排查中快速查找相关故障信息，方便定位问题。更多信息请参考#unique_81。

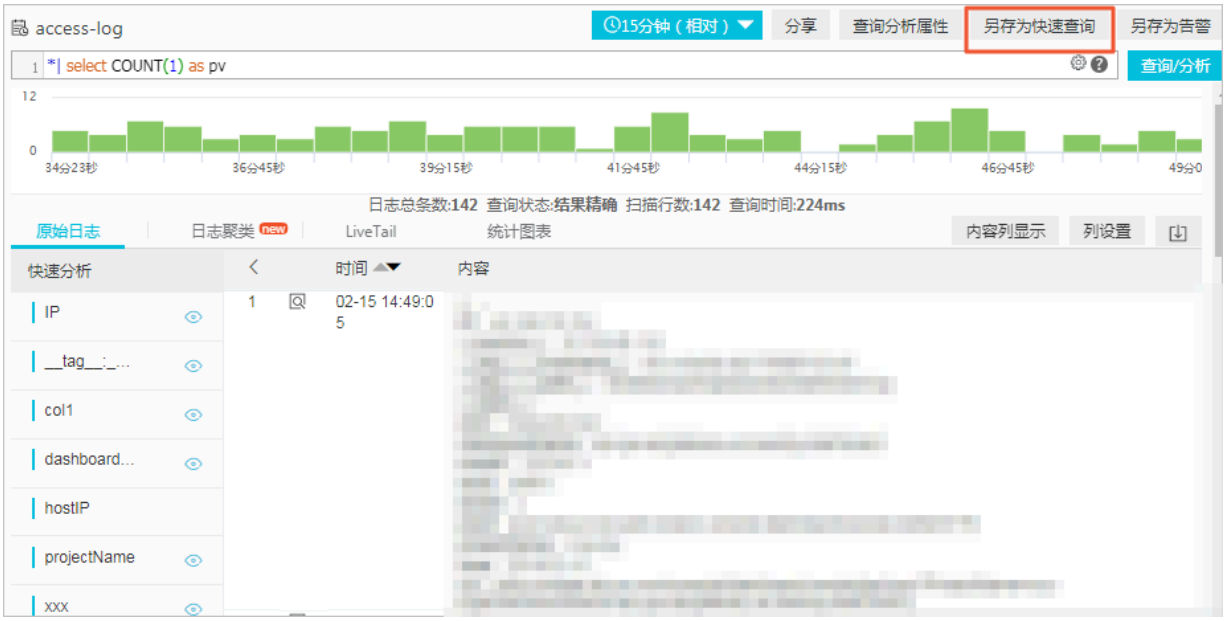
快速分析

日志服务（Log Service）快速分析功能给用户提供一键式交互查询体验，帮助用户快速分析某一字段在指定时间内的分布情况，降低用户索引关键数据的成本。详细说明请参考#unique_54。



快速查询

日志服务支持将当前的查询动作保存为快速查询，下次进行该查询动作时，不需要手动输入查询语句，只需进入该快速查询页面即可再次进行该项查询动作。详细说明请参考#unique_58。

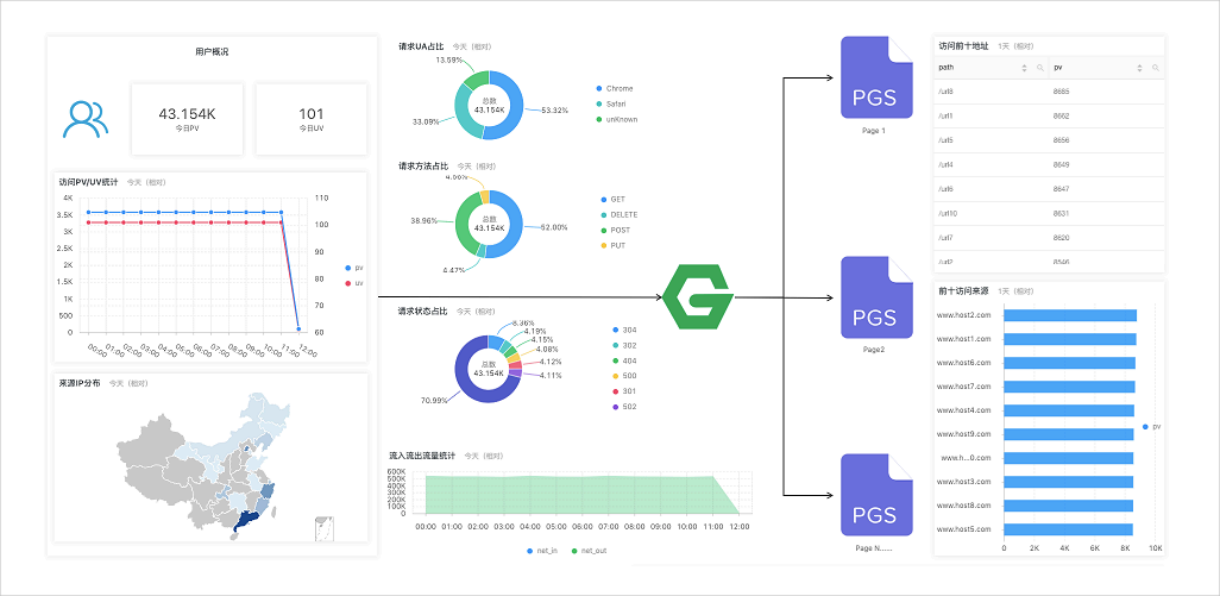


您也可以在告警规则中使用该快速查询条件。设置告警后，日志服务自动定期执行此快速查询，如果查询结果符合预设的阈值，则发送告警信息。

仪表盘

日志服务提供仪表盘功能，支持将查询分析语句进行可视化展示。详细信息请参考#unique_56。

图 7-18: 仪表盘



另存为告警

日志服务支持基于您的仪表盘或查询语句进行告警，您可以通过配置规则将具体告警内容发送给您。

详细信息请参考#unique_59。

8 SQL分析语法与功能

8.1 通用聚合函数

日志服务查询分析功能支持通过通用聚合函数进行日志分析，详细语句及含义如下：

语句	含义	示例
`arbitrary(x)`	随机返回x列中的一个值。	`latency > 100 | select arbitrary(method)`
`avg(x)`	计算x列的算数平均值。	`latency > 100 | select avg(latency)`
`checksum(x)`	计算某一列的checksum，返回base64编码。	`latency > 100 | select checksum(method)`
`count(*)`	表示所有的行数。	-
`count(x)`	计算某一列非null的个数。	`latency > 100 | count(method)`
`count(数字)`	`count(数字)`，如`count(1)`，等同于`count(*)`，表示所有的行数。	-
`count_if(x)`	计算x=true的个数。	`latency > 100 | count_if(url like '%abc')`
`geometric_mean(x)`	计算某一列的几何平均数。	`latency > 100 | select geometric_mean(latency)`
`max_by(x,y)`	返回当y取最大值时，x当前的值。	查询延时最高的时候，对应的method：`latency>100 | select max_by(method, latency)`
`max_by(x,y,n)`	返回y最高的n行，对应的x的值。	查询延时最高的3行，对应的method：`latency > 100 | select max_by(method,latency,3)`
`min_by(x,y)`	返回当y取最小值时，x当前的值。	查询延时最低的请求，对应的method：`* | select min_by(x,y)`

语句	含义	示例
`min_by(x,y,n)`	返回y最小的n行，对应的x的值。	查询延时最小的3行，对应的method：`* | select min_by(method,latency,3)`
`max(x)`	返回最大值。	`latency > 100| select max(inflow)`
`min(x)`	返回最小值。	`latency > 100| select min(inflow)`
`sum(x)`	返回x列的和。	`latency > 10 | select sum(inflow)`
`bitwise_and_agg(x)`	对某一列的所有数值做and计算。	-
`bitwise_or_agg(x)`	对某一列的数值做or计算。	-

8.2 安全检测函数

日志服务依托全球白帽子共享安全资产库，提供安全检测函数，您只需要将日志中任意的IP、域名或者URL传给安全检测函数，即可检测是否安全。

应用场景

1. 对服务运维有较强需求的企业和机构如互联网、游戏、资讯等，其IT和安全运维人员可借此及时筛选可疑访问、攻击以及侵入的行为，并支持进一步深入分析和采取一定措施进行防御。
2. 对内部资产保护有较强需求的企业和机构如银行、证券、电商等，其IT、安全运维人员可以借此即时发现内部访问危险网站、下载木马等行为，并即时采取行动。

功能特点

- 可靠：依托全球共享的白帽子安全资产库，并及时更新。
- 快速：检测百万IP、域名或URL仅需几秒钟。
- 简单：无缝支持任意网络日志，调用3个SQL函数security_check_ip、security_check_domain、security_check_url即可获得结果。
- 灵活：既可以交互式查询，也可以构建报表视图。还可以建立报警并采取进一步行动。

函数列表

函数名	含义	样例
security_check_ip	检查IP是否安全，其中： · 返回1：命中，表示不安全 · 返回0：未命中	`select security_check_ip(real_client_ip)`
security_check_domain	检查Domain是否安全，其中： · 返回1：命中，表示不安全 · 返回0：未命中	`select security_check_domain(site)`
security_check_url	检查URL是否安全，其中： · 返回1：命中，表示不安全 · 返回0：未命中	`select security_check_domain(concat(host, url)`

示例

- 检查外部可疑访问行为并生成报表

  某电商收集了其运营的Ngnix服务器的日志，对其访问的客户端中想要扫描是否存在不安全的客户IP。可以将Ngnix的日志中的ClientIP字段传给`security_check_ip`函数，并筛选出其返回值为1的IP进行展现，并展示其所在国家、网络运营商等。

  对应查询分析语句为：

```
* | select ClientIP, ip_to_country(ClientIP) as country, ip_to_prov
ider(ClientIP) as provider, count(1) as PV where security_check_ip(
ClientIP) = 1 group by ClientIP order by PV desc
```

ClientIP	sec	country	provider	PV
[redacted]	1	中国	电信	575
[redacted]	1	中国	联通	241
[redacted]	1	中国	电信	185
[redacted]	1	中国	联通	179
[redacted]	1	中国	联通	32
[redacted]	1	中国	电信	28

  设置为地图视图展示：

- 检查内部可疑访问行为并报警

  例如，某证券运营商收集了其内部设备通过网关代理访问外网的网络流量的日志，需要检查是否有人访问了有问题的网站，可以执行如下查询：

```
* | select client_ip, count(1) as PV where security_check_ip(
remote_addr) = 1 or security_check_site(site) = 1 or security_c
heck_url(concat(site, url)) = 1 group by client_ip order by PV desc
```

  您也可以将此语句另存为快速查询，并建立安全报警，当有客户端频繁访问危险网站时触发报警，配置每5分钟检查一次是否有人过去1小时内频繁（超过5次）访问危险网站。配置如下：

8.3 Map映射函数

日志服务查询分析功能支持通过映射函数进行日志分析，详细语句及含义如下：

函数	含义	示例
下标运算符[]	获取map中某个key对应的结果。	-
histogram(x)	按照x的每个值GROUP BY，计算count。语法相当于 `select count group by x` 。 说明: 返回结果为JSON格式。	`latency > 10 | select histogram(status)`，等同于`latency > 10 | select count(1) group by status`。
histogram_u(x)	按照x的每个值GROUP BY，计算count。 说明: 返回结果为多行多列。	`latency > 10 | select histogram(status)`，等同于`latency > 10 | select count(1) group by status`。
map_agg(Key,Value)	返回Key、Value组成的map，并展示每个method的随机的latency。	`latency > 100 | select map_agg(method,latency)`
multimap_agg(Key,Value)	返回Key、Value组成的多Value map，并返回每个method的所有的latency。	`latency > 100 | select multimap_agg(method, latency)`
cardinality(x) → bigint	获取map的大小。	-
element_at(map<K, V>, key) → V	获取key对应的value。	-
map() → map<unknown, unknown>	返回一个空的map。	-
map(array<K>, array<V>) → map<K,V>	把两个数组，转换成1对1的Map。	`SELECT map(ARRAY[1,3], ARRAY[2,4]); — {1 -> 2, 3 -> 4}`
map_from_entries(array<row<K, V>>) → map<K,V>	把一个多维数组转化成map。	`SELECT map_from_entries (ARRAY[(1, 'x'), (2, 'y')]); — {1 -> 'x', 2 -> 'y'}`
map_entries(map<K, V>) → array<row<K,V>>	把map中的元素转化成array形式。	`SELECT map_entries(MAP( ARRAY[1, 2], ARRAY['x', 'y'])); — [ROW(1, 'x'), ROW(2, 'y')]`

函数	含义	示例
map_concat(map1<K, V>, map2<K, V>, ⋯, mapN<K, V>) → map<K,V>	求多个map的并集，如果某个key在多个map中存在，则取第一个。	-
map_filter(map<K, V>, function) → map<K,V>	请参考lambda map_filter函数。	-
transform_keys(map<K1, V>, function) → MAP<K2,V>	请参考lambda transform_keys函数。	-
transform_values(map<K, V1>, function) → MAP<K, V2>	请参考lambda transform_values函数。	-
map_keys(x<K, V>) → array<K>	获取map中所有的key，返回array。	-
map_values(x<K, V>) → array<V>	获取map中所有的value，返回array。	-
map_zip_with(map<K, V1>, map<K, V2>, function<K, V1, V2, V3>) → map<K, V3>	请参考lambda中map_zip_with函数。	-

8.4 估算函数

日志服务查询分析功能支持通过估算进行日志分析，详细语句及含义如下：

函数	说明	示例
approx_distinct(x)	估算x列的唯一值的个数。	-
approx_percentile(x, percentage)	对于x列排序，找出大约处于percentage位置的数值。	找出位于一半位置的数值：approx_percentile(x,0.5)
approx_percentile(x, percentages)	与上述用法类似，但可以指定多个percentage，找出每个percentage对应的数值。	approx_percentile(x, array[0.1,0.2])

函数	说明	示例
numeric_histogram(buckets, Value)	对于数值列，分多个桶进行统计。即把*Value*一列，分到桶中，桶的个数为*buckets*。 返回内容为每个桶的Key及对应的count数值，相当于针对数值的select count group by 说明: 返回结果的格式为JSON。	对于POST请求，把延时分为10个桶，查看每个桶的大小：method:POST | select numeric_histogram(10,latency)
numeric_histogram_u(buckets, Value)	对于数值列，分多个桶进行统计。即把*Value*一列，分到桶中，桶的个数为*buckets*。 返回内容为每个桶的Key及对应的count数值，相当于针对数值的select count group by 说明: 返回结果的格式为多行多列。	对于POST请求，把延时分为10个桶，查看每个桶的大小：method:POST | select numeric_histogram_u(10,latency)

8.5 数学统计函数

日志服务查询分析功能支持通过数学统计函数进行日志分析，详细语句及含义如下：

语句	含义	示例
corr(y, x)	给出两列的相关度，结果从0到1。	latency>100| select corr(latency,request_size)
covar_pop(y, x)	计算总体协方差。	latency>100| select covar_pop(request_size, latency)
covar_samp(y, x)	计算样本协方差。	latency>100| select covar_samp(request_size,latency)

语句	含义	示例
regr_intercept(y, x)	返回输入值的线性回归截距。y是依赖值，x是独立值。	latency>100| select regr_intercept(request_size,latency)
regr_slope(y,x)	返回输入值的线性回归斜率。y是依赖值，x是独立值。	latency>100| select regr_slope(request_size,latency)
stddev(x)或stddev_samp(x)	返回x列的样本标准差。	latency>100| select stddev(latency)
stddev_pop(x)	返回x列的总体标准差。	latency>100| select stddev_pop(latency)
variance(x)或 var_samp(x)	计算x列的样本方差。	latency>100| select variance(latency)
var_pop(x)	计算x列的总体方差。	latency>100| select variance(latency)

8.6 数学计算函数

日志服务查询分析功能支持通过数学计算函数进行日志分析，您可以结合查询语句和数学计算函数，对日志查询结果进行数学计算。

数学运算符

数学运算符支持 + - * / % 。可以用在SELECT子句中。

样例：

```
*|select avg(latency)/100 , sum(latency)/count(1)
```

数学计算函数说明

日志服务支持以下运算函数：

函数名	含义
abs(x)	返回x列的绝对值。
cbrt(x)	返回x列的立方根。
ceiling (x)	返回x列向上最接近的整数。
cosine_similarity(x,y)	返回稀疏向量x和y之间的余弦相似度。
degrees	把弧度转化为度。

函数名	含义
e()	返回自然常数。
exp(x)	返回自然常数的指数。
floor(x)	返回x向下最接近的整数。
from_base(string,radix)	以radix进制解释string。
ln(x)	返回自然对数。
log2(x)	返回以2为底，x的对数。
log10(x)	返回以10为底，x的对数。
log(x,b)	返回以b为底，x的对数。
pi()	返回π。
pow(x,b)	返回x的b次幂。
radians(x)	把度转化成弧度。
rand()	返回随机数。
random(0,n)	返回[0，n)随机数。
round(x)	x四舍五入。
round(x, y)	对x保留y个小数为，例如round(1.012345,2) = 1.01。
sqrt(x)	返回x的平方根。
to_base(x, radix)	把x以radix进制表示。
truncate(x)	丢弃掉x的小数部分。
acos(x)	反余弦。
asin(x)	反正弦。
atan(x)	反正切。
atan2(y,x)	y/x的反正切。
cos(x)	余弦。
sin(x)	正弦。
cosh(x)	双曲余弦。
tan(x)	正切。
tanh(x)	双曲正切。

函数名	含义
infinity()	double最大值。
is_infinity(x)	判断是否是最大值。
is_finity(x)	判断是否是最大值。
is_nan(x)	判断是否是数值。

8.7 字符串函数

日志服务查询分析功能支持通过字符串函数进行日志分析，详细语句及含义如下：

函数名	含义
chr(x)	把int类型转化成对应的ASCII码，例如chr(65)结果为’A’。
codepoint (x)	把一个ASCII码转化成int类型的编码，例如codepoint('A')结果为65。
length(x)	字段长度。
levenshtein_distance(string1, string2)	返回两个字符串的最小编辑距离。
lower(string)	转化成小写。
lpad(string, size, padstring)	把string对齐到size大小，如果小于size，用padstring，从左侧补齐到size；如果大于size，则截取到size个。
rpad(string, size, padstring)	类似lpad，从右侧补齐string。
ltrim(string)	删掉左侧的空白字符。
replace(string, search)	把字符串中string中的search删掉。
replace(string, search,rep)	把字符串中string中的search替换为rep。
reverse(string)	翻转string。
rtrim(string)	删掉字符串结尾的空白字符。
split(string,delimeter,limit)	把字符串分裂成array，最多取limit个值。生成的结果为数组，下标从1开始。
split_part(string,delimeter,offset)	把字符串分裂成array，取第offset个字符串。生成的结果为数组，下标从1开始。

函数名	含义
split_to_map(string, entryDelimiter, keyValueDelimiter) → map<varchar, varchar>	把string按照entryDelemiter分割成多个entry，每个entry再按照keyValueDelimiter划分成key value。最终返回一个map。
position(substring IN string)	获取string中，substring最先开始的位置。
strpos(string, substring)	查找字符串中的子串的开始位置。返回结果从1开始，如果不存在则返回0。
substr(string, start)	返回字符串的子串，start下标从1开始。
substr(string, start, length)	返回字符串的子串，start下标从1开始。
trim(string)	删掉字符串开头和结尾的空白字符。
upper(string)	转化为大写字符。
concat(string,string.....)	把两个或多个字符串拼接成一个字符串。
hamming_distance (string1,string2)	获得两个字符串的海明距离。

说明:

字符串需要加单引号包裹，双引号表示列名。例如：a=‘abc’ 表示列a=字符串abc；a=‘abc’ 表示a列=abc列。

8.8 日期和时间函数

日志服务支持时间函数、日期函数、区间函数和时序补全函数，您可以在分析语句中使用本文档中介绍的函数。

日期时间类型

- unixtime：以int类型表示从1970年1月1日开始的秒数，例如1512374067表示的时间是Mon Dec 4 15:54:27 CST 2017。日志服务每条日志中内置的时间__time__即为这种类型。
- timestamp类型：以字符串形式表示时间，例如2017-11-01 13:30:00。

日期函数

日志服务支持的常见日期函数如下：

函数名	含义	样例
current_date	当天日期。	latency>100| select current_date

函数名	含义	样例
current_time	当前时间。	latency>100| select current_time
current_timestamp	结合current_date 和 current_time的结果。	latency>100| select current_timestamp
current_timezone()	返回时区。	latency>100| select current_timezone()
from_iso8601_timestamp(string)	把iso8601时间转化成带时区的时间。	latency>100| select from_iso8601_timestamp(iso8601)
from_iso8601_date(string)	把iso8601转化成天。	latency>100| select from_iso8601_date(iso8601)
from_unixtime(unixtime)	把unix时间转化为时间戳。	latency>100| select from_unixtime(1494985275)
from_unixtime(unixtime, string)	以string为时区，把unixtime转化成时间戳。	latency>100| select from_unixtime (1494985275,'Asia/ Shanghai')
localtime	本地时间。	latency>100| select localtime
localtimestamp	本地时间戳。	latency>100| select localtimestamp
now()	等同于current_timestamp 。	-
to_unixtime(timestamp)	timestamp转化成unixtime 。	*| select to_unixtime(' 2017-05-17 09:45:00.848 Asia/Shanghai')

时间函数

日志服务还支持MySQL时间格式，包括%a、%b、%y 等。

函数名	含义	样例
`date_format(timestamp, format)`	把timestamp转化成以format形式表示。	`latency>100| select date_format (date_parse ('2017-05-17 09:45:00 ','%Y-%m-%d %H:%i:%S'), '%Y-%m-%d')`
`date_parse(string, format)`	把string以format格式解析，转化成timestamp。	`latency>100|select date_format (date_parse (time,'%Y-%m-%d %H:%i:%S'), '%Y-%m-%d')`

表 8-1: 格式说明

格式	描述
%a	星期的缩写，即Sun、Sat等。
%b	月份的缩写，即Jan、Dec等。
%c	月份，数值类型，即1~12。
%D	每月的第几天，带后缀，即0th、1st、2nd、3rd等。
%d	每月第几天，十进制格式，范围为01~31。
%e	每月第几天，十进制格式，范围为1~31。
%H	小时，24小时制。
%h	小时，12小时制。
%I	小时，12小时制。
%i	分钟，数值类型，范围为00~59。
%j	每年的第几天，范围为001~366。
%k	小时，范围为0~23。
%l	小时，范围为1~12。
%M	月份的英文表达，范围为January~December。
%m	月份，数值格式，范围为01~12。
%p	AM或PM。
%r	时间，12小时制，格式为`hh:mm:ss AM/PM`。
%S	秒，范围为00~59。
%s	秒，范围为00~59。

格式	描述
%T	时间，24时制，格式为 `hh:mm:ss`。
%U	每年的第几周，星期日是一周的第一天。取值范围为00~53。
%u	每年的第几周，星期一是一周的第一天。范围为00~53。
%V	每年的第几周，星期日是一周的第一天。范围为01~53，与%X同时使用。
%v	每年的第几周，星期一是一周的第一天。范围为01~53，与%x同时使用。
%W	星期几的名称，范围为Sunday到Saturday。
%w	一周的第几天， 星期日为第0天。
%Y	4 位数的年份。
%y	2位数的年份。
%%	%转义字符。

时间段对齐函数

日志服务支持时间段对齐函数，可以按照秒、分钟，小时、日、月、年等对齐。这个函数常用于一些按照时间进行统计的场景。

- 函数语法：

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

- 参数说明：

  x 可以是一个timestamp类型，也可以是unix time。

  Unit的取值包括以下类型，其中x取`2001-08-22 03:04:05.000`：

Unit	转化后结果
second	2001-08-22 03:04:05.000
minute	2001-08-22 03:04:00.000
hour	2001-08-22 03:00:00.000
day	2001-08-22 00:00:00.000
week	2001-08-20 00:00:00.000
month	2001-08-01 00:00:00.000
quarter	2001-07-01 00:00:00.000
year	2001-01-01 00:00:00.000

- 示例：

  date_trunc只能在按照一些固定时间间隔统计，如果需要按照灵活的时间维度进行统计（例如统计每5分钟数据），需要按照数学取模方法进行GROUP BY。

```
* | SELECT count(1) as pv,  __time__ - __time__% 300 as minute5
group by minute5 limit 100
```

  上述公式中的%300表示按照5分钟进行取模对齐。

  以下为使用时间格式的一个综合样例。

```
*|select  date_trunc('minute' ,  __time__)  as t,
       truncate (avg(latency) ) ,
       current_date
       group by   t
       order by t  desc
       limit 60
```

时间间隔函数

时间间隔函数用来执行时间段相关的运算，如在日期中添加或减去指定的时间间隔、计算两个日期之间的时间。

函数名	含义	样例
date_add(*unit*, *value*, *timestamp*)	在timestamp上加上value个unit。如果要执行减法，value使用负值。	date_add('day', -7, '2018-08-09 00:00:00') 表示8月9号之前7天
date_diff(*unit*, *timestamp1*, *timestamp2*)	表示timestamp1和timestamp2之间相差几个unit。	date_diff('day', '2018-08-02 00:00:00', '2018-08-09 00:00:00') = 7

该函数支持以下区间单位：

单位	说明
millisecond	毫秒
second	秒
minute	分钟
hour	小时
day	天
week	周
month	月
quarter	季度，即三个月

单位	说明
year	年

时序补全函数

时序补全函数time_series用于处理某些时间缺少的情况。

说明:

该函数必须和`group by time order by time`一起使用，且`order by`不支持`desc`排序方式

- 函数格式:

```
time_series(time_column, window, format, padding_data)
```

- 参数说明:

参数	说明
time_column	时间列，例如日志服务提供的默认时间字段`__time__`。格式为long类型或timestamp类型。
window	窗口大小，由一个数字和单位组成。单位为s（秒）、m（分）、H（小时）、或d（天）。例如2h、5m、3d。
format	MySQL时间格式，表示最终输出的格式。
padding_data	表示补全的内容，包括: - 0：补零。 - null：补null。 - last：补上一个值。 - next：补下一个值。 - avg：补前后的平均值。

- 示例:

按照每两个小时进行格式化:

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

输出：

8.9 URL函数

URL函数支持从标准URL路径中提取字段，一个标准的URL如下：

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

常见URL函数

函数名	含义	示例	
		输入样例	输出结果
url_extract_fragment(url)	提取出URL中的fragment，结果为varchar类型。	*| select url_extract_fragment('https://sls.console.aliyun.com/#/project/dashboard-demo/categoryList')	输出结果为/project/dashboard-demo/categoryList。
url_extract_host(url)	提取出URL中的host，结果为varchar类型。	*|select url_extract_host('http://www.aliyun.com/product/sls')。	输出结果为www.aliyun.com。
url_extract_parameter(url, name)	提取出URL中的query中name对应的参数值，结果为varchar类型。	*|select url_extract_parameter('http://www.aliyun.com/product/sls?userid=testuser','userid')	输出结果为testuser。

函数名	含义	示例	
		输入样例	输出结果
url_extract_path(url)	提取出URL中的path，结果为varchar类型。	*|select url_extract_path('http://www.aliyun.com/product/sls?userid=testuser')	输出结果为/product/sls。
url_extract_port(url)	提取出URL中的端口，结果为bigint类型。	*|select url_extract_port('http://www.aliyun.com:80/product/sls?userid=testuser')	输出结果为80。
url_extract_protocol(url)	提取出URL中的协议，结果为varchar类型。	*|select url_extract_protocol('http://www.aliyun.com:80/product/sls?userid=testuser')	输出结果为http。
url_extract_query(url)	提取出URL中的query，结果为varchar类型。	*|select url_extract_query('http://www.aliyun.com:80/product/sls?userid=testuser')	输出结果为userid=testuser。
url_encode(value)	对url进行转义编码。	*|select url_encode('http://www.aliyun.com:80/product/sls?userid=testuser')	输出结果为http%3a%2f%2fwww.aliyun.com%3a80%2fproduct%2fsls%3fuserid%3dtestuser。
url_decode(value)	对url进行解码。	*|select url_decode('http%3a%2f%2fwww.aliyun.com%3a80%2fproduct%2fsls%3fuserid%3dtestuser')	输出结果为http://www.aliyun.com:80/product/sls?userid=testuser。

8.10 正则式函数

正则式函数解析一串字符串，并且返回需要的一部分子串。正则表达式语法规则请参考基本语法。

常见的正则式函数及含义如下：

函数名	含义	样例
regexp_extract_all(string, pattern)	返回字符串中命中正则式的所有子串，返回结果是一个字符串数组。	*| SELECT regexp_extract_all('5a 67b 890m', '\d+')，结果为['5','67','890']，*|SELECT regexp_extract_all('5a 67a 890m', '(\d+)a') 结果为['5a','67a']。
regexp_extract_all(string, pattern, group)	返回字符串中命中正则式的第group个()内部分,返回结果是一个字符串数组。	*| SELECT regexp_extract_all(‘5a 67a 890m’, ‘(\d+)a’,1) 结果为[‘5’,’67’]
regexp_extract(string, pattern)	返回字符串命中的正则式的第一个子串。	*|SELECT regexp_extract('5a 67b 890m', '\d+') 结果为'5'
regexp_extract(string, pattern, group)	返回字符串命中的正则式的第group个()内的第1个子串。	*|SELECT regexp_extract('5a 67b 890m', '(\d+)([a-z]+)',2) 结果为'a'
regexp_like(string, pattern)	判断字符串是否命中正则式，返回bool类型，正则式可以只命中字符串的一部分。	*|SELECT regexp_like('5a 67b 890m', '\d+m') 结果为true
regexp_replace(string, pattern, replacement)	把字符串中命中正则式的部分替换成replacement。	*|SELECT regexp_replace('5a 67b 890m', '\d+','a') 结果为'aa ab am'
regexp_replace(string, pattern)	把字符串中命中正则式的部分删除，相当于regexp_replace(string,patterm,'')。	*|SELECT regexp_replace('5a 67b 890m', '\d+') 结果为'a b m'
regexp_split(string, pattern)	使用正则式把字符串切分成数组。	*|SELECT regexp_split('5a 67b 890m', '\d+') 结果为['a','b','m']

8.11 JSON函数

JSON函数，可以解析一段字符串为JSON类型，并且提取JSON中的字段。JSON主要有两种结构：map和array。如果一个字符串解析成JSON失败，那么返回的是null。

如果需要把json展开成多行，请参考#unique_46。

日志服务支持以下常见的JSON函数：

函数名	含义	样例
json_parse(string)	把字符串转化成JSON类型。	SELECT json_parse('[1, 2, 3]') 结果为JSON类型数组
json_format(json)	把JSON类型转化成字符串。	SELECT json_format(json_parse('[1, 2, 3]')) 结果为字符串
json_array_contains(json, value)	判断一个JSON类型数值，或者一个字符串（内容是一个JSON数组）是否包含某个值。	SELECT json_array_contains(json_parse('[1, 2, 3]'), 2)或 SELECT json_array_contains('[1, 2, 3]', 2)
json_array_get(json_array, index)	同json_array_contains，是获取一个JSON数组的某个下标对应的元素。	SELECT json_array_get('["a", "b", "c"]', 0)结果为'a'
json_array_length(json)	返回JSON数组的大小。	SELECT json_array_length('[1, 2, 3]') 返回结果3
json_extract(json, json_path)	从一个JSON对象中提取值，JSON路径的语法类似$.store.book[0].title，返回结果是一个JSON对象。	SELECT json_extract(json, '$.store.book');
json_extract_scalar(json, json_path)	类似json_extract，但是返回结果是字符串类型。	-
json_size(json, json_path)	获取JSON对象或数组的大小。	SELECT json_size('[1, 2, 3]') 返回结果3

8.12 类型转换函数

类型转换函数用于在查询中转换指定值或指定列的数据类型。

日志服务索引属性中，字段可被配置为long、double、text和json类型。同时日志服务支持查询多种数据类型的字段，包括bigint、double、varchar、timestamp等。如果查询时需要区分更细维度的数据类型，可以使用类型转换函数将索引属性中配置的数据类型转换为查询中使用的数据类型。

函数格式

> 说明:
> 日志中可能有脏数据时，建议使用try_cast()函数，否则容易因脏数据造成整个查询失败。

- 在查询中将某一列（字段）或某一个值转换成指定类型。其中，如果某一个值转换失败，将终止整个查询。

```
cast([key|value] AS type)
```

- 在查询中将某一列（字段）或某一个值转换成指定类型。如果某一个值转换失败，该值返回NULL，并跳过该值继续处理。

```
try_cast([key|value] AS type)
```

参数	说明
key	日志的Key，表示将该字段所有的值都转换成指定类型。
value	常量值，表示将某个值转换成指定类型。

示例

- 将数字123转换为字符串（varchar）格式：

```
cast(123 AS varchar)
```

- 将uid字段转换为字符串（varchar）格式：

```
cast(uid AS varchar)
```

8.13 IP地理函数

IP 识别函数，可以识别一个IP是内网IP还是外网IP，也可以判断IP所属的国家、省份、城市。

函数名	含义	样例
`ip_to_domain(ip)`	判断IP所在的域，是内网还是外网。返回intranet或internet。	`SELECT ip_to_domain(ip)`
`ip_to_country(ip)`	判断IP所在的国家。	`SELECT ip_to_country(ip)`
`ip_to_province(ip)`	判断IP所在的省份。	`SELECT ip_to_province(ip)`
`ip_to_city(ip)`	判断IP所在的城市。	`SELECT ip_to_city(ip)`

函数名	含义	样例
ip_to_geo(ip)	判断IP所在的城市的经纬度，范围结果格式为纬度,经度。	SELECT ip_to_geo(ip)
ip_to_city_geo(ip)	判断IP所在的城市的经纬度，返回的是城市经纬度，每个城市只有一个经纬度，范围结果格式为纬度,经度。	SELECT ip_to_city_geo(ip)
ip_to_provider(ip)	获取IP对应的网络运营商。	SELECT ip_to_provider(ip)
ip_to_country(ip,'en')	判断IP所在的国家，返回国际码。	SELECT ip_to_country(ip,'en')
ip_to_country_code(ip)	判断IP所在的国家，返回国际码。	SELECT ip_to_country_code(ip)
ip_to_province(ip,'en')	判断IP所在的省份，返回英文省名或者中文拼音。	SELECT ip_to_province(ip,'en')
ip_to_city(ip,'en')	判断IP所在的城市，返回英文城市名或者中文拼音。	SELECT ip_to_city(ip,'en')

示例

- **在查询中过滤掉内网访问请求，看请求总数**

```
* | selectcount(1)whereip_to_domain(ip)!='intranet'
```

- **查看Top10的访问省份**

```
* | SELECT count(1) as pv, ip_to_province(ip) as province GROUP BY
province order by pv desc limit 10
```

响应结果样例

```
[
    {
        "__source__": "",
        "__time__": "1512353137",
        "province": "浙江省",
        "pv": "4045"
    }, {
        "__source__": "",
        "__time__": "1512353137",
        "province": "上海市",
        "pv": "3727"
    }, {
        "__source__": "",
        "__time__": "1512353137",
        "province": "北京市",
```

```
        "pv": "954"
    }, {
        "__source__": "",
        "__time__": "1512353137",
        "province": "内网IP",
        "pv": "698"
    }, {
        "__source__": "",
        "__time__": "1512353137",
        "province": "广东省",
        "pv": "472"
    }, {
        "__source__": "",
        "__time__": "1512353137",
        "province": "福建省",
        "pv": "71"
    }, {
        "__source__": "",
        "__time__": "1512353137",
        "province": "阿联酋",
        "pv": "52"
    }, {
        "__source__": "",
        "__time__": "1512353137",
        "province": "美国",
        "pv": "43"
    }, {
        "__source__": "",
        "__time__": "1512353137",
        "province": "德国",
        "pv": "26"
    }, {
        "__source__": "",
        "__time__": "1512353137",
        "province": "吉隆坡",
        "pv": "26"
    }
]
```


在上述结果中包含了内网IP，有时候，开发自己的测试是从内网发出的，为了过滤掉这部分访问请求，可以使用下边的分析语句：

- 过滤掉内网请求，查看Top 10的网络访问省份

```
* | SELECT count(1) as pv, ip_to_province(ip) as province WHERE
ip_to_domain(ip) != 'intranet'  GROUP BY province ORDER BY pv desc
limit 10
```

- 查看不同国家的平均响应延时，最大响应延时，最大延时对应的request

```
* | SELECT AVG(latency),MAX(latency),MAX_BY(requestId, latency) ,
ip_to_country(ip) as country group by country limit 100
```

- 查看不同网络运营商的平均延时

```
* | SELECT AVG(latency) , ip_to_provider(ip) as provider group by
provider limit 100
```

- 查看IP的经纬度，绘制地图

```
* | select count(1) as pv , ip_to_geo(ip) as geo group by geo order
by pv desc
```

返回的格式为：

pv	geo
100	35.3284,-80.7459

8.14 GROUP BY 语法

GROUP BY 支持多列。GROUP BY支持通过SELECT的列的别名来表示对应的KEY。

样例：

```
method:PostLogstoreLogs |select avg(latency),projectName,date_trunc('
hour',__time__) as hour   group by projectName,hour
```

别名hour代表第三个SELECT列`date_trunc('hour',__time__)`。这类用法对于一些非常复杂的query非常有帮助。

GROUP BY 支持GROUPING SETS、CUBE、ROLLUP。

样例：

```
method:PostLogstoreLogs |select avg(latency)   group by cube(
projectName,logstore)
method:PostLogstoreLogs |select avg(latency)   group by GROUPING SETS
 ( ( projectName,logstore), (projectName,method))
method:PostLogstoreLogs |select avg(latency)   group by rollup(
projectName,logstore)
```

实践样例

按照时间进行GROUP BY

每条日志都内置了一个时间列__time__，当打开任意一列的统计功能后，会自动给时间列打开统计。

使用date_trunc函数，可以把时间列对齐到小时(hour)、分钟(minute)、天(day)、月(month)、年(year)。date_trunc接受一个对齐单位，和一个unix time或者timestamp类型的列，例如__time__。

- 按照每小时、每分钟统计计算PV

```
* | SELECT count(1) as pv , date_trunc('hour',__time__) as hour
group by hour order by hour limit 100
* | SELECT count(1) as pv , date_trunc('minute',__time__) as minute
group by minute order by minute limit 100
```

说明:

limit 100表示最多获取100行，如果不加LIMIT语句，默认最多获取10行数据。

- 按照灵活的时间维度进行统计，例如统计每5分钟的，date_trunc只能在按照一些固定时间间隔统计，这种场景下，我们需要按照数学取模方法进行GROUP BY。

```
* | SELECT count(1) as pv,  __time__ - __time__% 300 as minute5
group by minute5 limit 100
```

上述公式中的%300表示按照5分钟进行取模对齐。

在GROUP BY 中提取非agg列

在标准SQL中，如果使用了GROUP BY语法，那么在SELECT时，只能选择SELECT GROUP BY的列原始内容，或者对任意列进行聚合计算，不允许获取非GROUP BY列的内容。

例如，以下语法是非法的，因为b是非GROUP BY的列，在按照a进行GROUP BY时，有多行b可供选择，系统不知道该选择哪一行输出。

```
*|select a, b , count(c) group by a
```

为了达到以上目的，可以使用arbitrary函数输出b:

```
*|select a, arbitrary(b), count(c) group by a
```

8.15 窗口函数

窗口函数用来跨行计算。普通的SQL聚合函数只能用来计算一行内的结果，或者把所有行聚合成一行结果。窗口函数，可以跨行计算，并且把结果填到到每一行中。

窗口函数语法:

```
SELECT key1, key2, value,
```

```
       rank() OVER (PARTITION BY key2
                    ORDER BY value DESC) AS rnk
FROM orders
ORDER BY key1,rnk
```

核心部分是：

```
rank() OVER (PARTITION BY  KEY1 ORDER BY KEY2 DESC)
```

其中rank()是一个聚合函数，可以使用分析语法中的任何函数，也可以使用本文档列出的函数。

PARTITION BY 是值按照哪些桶进行计算。

窗口中使用的特殊聚合函数

函数	含义
rank()	在窗口内，按照某一列排序，返回在窗口内的序号。
row_number()	返回在窗口内的行号。
first_value(x)	返回窗口内的第一个value，一般用法是窗口内数值排序，获取最大值。
last_value(x)	含义和first value相反。
nth_value(x, offset)	窗口内的第offset个数。
lead(x,offset,defaut_value)	窗口内x列某行之后offset行的值，如果不存在该行，则取default_value。
lag(x,offset,defaut_value)	窗口内x列某行之前offset行的值，如果不存在该行，则取default_value。

使用样例

- 在整个公司的人员中，获取每个人的薪水在部门内排名

```
* | select department, persionId, sallary , rank() over(PARTITION
 BY department order by sallary desc) as sallary_rank  order by
department,sallary_rank
```

响应结果：

department	persionId	sallary	sallary_rank
dev	john	9000	1
dev	Smith	8000	2
dev	Snow	7000	3
dev	Achilles	6000	4
Marketing	Blan Stark	9000	1

department	persionId	sallary	sallary_rank
Marketing	Rob Stark	8000	2
Marketing	Sansa Stark	7000	3

- 在整个公司的人员中，获取每个人的薪水在部门内的占比

```
* | select department, persionId, sallary *1.0 / sum(sallary) over(
PARTITION BY department ) as sallary_percentage
```

响应结果：

department	persionId	sallary	sallary_percentage
dev	john	9000	0.3
dev	Smith	8000	0.26
dev	Snow	7000	0.23
dev	Achilles	6000	0.2
Marketing	Blan Stark	9000	0.375
Marketing	Rob Stark	8000	0.333
Marketing	Sansa Stark	7000	0.29

- 按天统计，获取每天UV相对前一天的增长情况

```
* | select day ,uv, uv *1.0 /(lag(uv,1,0) over() ) as diff_perce
ntage from
(
select approx_distinct(ip) as uv, date_trunc('day',__time__) as day
from log group by day order by day asc
)
```

响应结果：

day	uv	diff_percentage
2017-12-01 00:00:00	100	null
2017-12-02 00:00:00	125	1.25
2017-12-03 00:00:00	150	1.2
2017-12-04 00:00:00	175	1.16
2017-12-05 00:00:00	200	1.14
2017-12-06 00:00:00	225	1.125
2017-12-07 00:00:00	250	1.11

8.16 HAVING语法

日志服务查询分析功能支持标准SQL的HAVING语法，和GROUP BY配合使用，用于过滤GROUP BY的结果。

格式：

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

HAVING和WHERE的区别

HAVING 用于过滤GROUP BY之后的聚合计算的结果，WHERE在聚合计算之间过滤原始数据。

示例

对于气温大于10℃的省份，计算每个省份的平均降雨量，并在最终结果中只显示平均降雨量大于100mL的省份：

```
* | select avg(rain) ,province where temperature > 10 group by
province having avg(rain) > 100
```

8.17 ORDER BY语法

ORDER BY 用于对输出结果进行排序，目前只支持按照一列进行排序。

语法格式：

```
orderby  列名 [desc|asc]
```

样例：

```
method :PostLogstoreLogs |select avg(latency) as avg_latency,
projectName group by projectName
HAVING avg(latency) > 5700000
order by avg_latency desc
```

8.18 LIMIT语法

LIMIT语法用于限制输出结果的行数。

语法格式

日志服务支持以下两种LIMIT语法格式。

- 只读取前N行：

```
limit N
```

- 从S行开始读，读取N行：

```
limit S , N
```

说明：

- limit 翻页读取时，只用于获取最终的结果，不可用于获取SQL中间的结果。
- 不支持将limit语法用于子查询内部。例如：

```
* | select count(1) from ( select distinct(url) from limit 0,1000)
```

- LIMIT翻页的offset不能超过1,000,000。即`limit S , N`，S和N之和不能超过1,000,000，N不能超出10,000。

示例

- 只获取100行结果：

```
* | select distinct(url) from log limit 100
```

- 获取0行到第999行的结果，共计1000行：

```
* | select distinct(url) from log limit 0,1000
```

- 获取第1000行到第1999行的结果，共计1000行：

```
* | select distinct(url) from log limit 1000,1000
```

8.19 CASE WHEN和IF分支语法

支持CASE WHEN语法，对连续数据进行归类。例如，从http_user_agent中提取信息，归类成Android和iOS两种类型：

```
SELECT
 CASE
 WHEN http_user_agent like '%android%' then 'android'
 WHEN http_user_agent like '%ios%' then 'ios'
 ELSE 'unknown' END
 as http_user_agent,
    count(1) as pv
    group by  http_user_agent
```

样例

- 计算状态码为200的请求占总体请求的比例：

```
* | SELECT
```

```
 sum(
 CASE
 WHEN status =200 then 1
 ELSE 0 end
 ) *1.0 / count(1) as status_200_percentage
```

- 统计不同延时区间的分布：

```
* | SELECT `
 CASE
 WHEN latency < 10 then 's10'
 WHEN latency < 100 then 's100'
 WHEN latency < 1000 then 's1000'
 WHEN latency < 10000 then 's10000'
 else 's_large' end
 as latency_slot,
 count(1) as pv
 group by latency_slot
```

IF语法

if语法逻辑上等同于CASE WHEN语法。

```
CASE
     WHEN condition THEN true_value
     [ ELSE false_value ]
 END
```

- if(condition, true_value)

  如果condition是true，则返回true_value这一列，否则返回null。

- if(condition, true_value, false_value)

  如果condition是true，则返回true_value这一列，否则返回false_value这一列。

COALESCE语法

coalesce 返回多个列的第一个非Null值。

```
coalesce(value1, value2[,...])
```

NULLIF 语法

如果value1和value2相等，返回null，否则返回value1。

```
nullif(value1, value2)
```

TRY 语法

try语法可以捕获一些底层的异常，例如除0错误，返回null值。

```
try(expression)
```

8.20 嵌套子查询

针对一些复杂的查询场景，一层SQL无法满足需求，通过SQL嵌套查询可以满足复杂的需求。

嵌套子查询和无嵌套查询的区别在于，要在SQL中指定from 条件。在查询中要指定`from log`这个关键字，表示从日志中读取原始数据。

样例：

```
* | select sum(pv) from
(
select count(1) as pv from log group by method
)
```

8.21 数组

语句	含义	示例
下标运算符[]	[]用于获取数组中的某个元素。	-

连接运算符||	||用于把两个数组连接成一个数组。	`SELECT ARRAY [1] || ARRAY [2]; – [1, 2]`
		`SELECT ARRAY [1] || 2; – [1, 2]`
		`SELECT 2 || ARRAY [1]; – [2, 1]`

语句	含义	示例
array_distinct	数组去重，获取数组中的唯一元素。	-
array_intersect(x, y)	获取x，y两个数组的交集。	-
array_union(x, y) → array	获取x，y两个数组的并集。	-
array_except(x, y) → array	获取x，y两个数组的差集	-
array_join(x, delimiter , null_replacement) → varchar	把字符串数组用delimiter连接，拼接成字符串，null值用null_replacement替代。 说明: 使用array_join函数时，返回结果最大为1 KB，超出1 KB的数据会被截断。	-
array_max(x) → x	获取x中的最大值。	-
array_min(x) → x	获取x中的最小值。	-
array_position(x, element) → bigint	获取element在x中的下标，下标从1开始。如果找不到，则返回0。	-
array_remove(x, element) → array	从数组中移除element。	-
array_sort(x) → array	给数组排序，null值放到最后。	-
cardinality(x) → bigint	获取数组的大小。	-
concat(array1, array2, ···, arrayN) → array	连接数组。	-
contains(x, element) → boolean	如果x中包含element，则返回true。	-
filter(array, function) → array	function 是一个Lambda函数，请参考#unique_34中的filter。	-
flatten(x) → array	把二维的array拼接成一维的array。	-
reduce(array, initialState, inputFunction, outputFunction) → x	请参考#unique_34reduce。	-

语句	含义	示例
reverse(x) → array	把x反向排列。	-
sequence(start, stop) → array	生成从start到stop结束的一个序列，每一步加1。	-
sequence(start, stop, step) → array	生成从start到stop结束的一个序列，每一步加step。	-
sequence(start, stop, step) → array	start和stop是timestamp类型，生成从start到stop结束的timestamp数组。step是INTERVAL类型，可以是DAY到SECOND，也可以是YEAR或MONTH。	-
shuffle(x) → array	重新随机分布array。	-
slice(x, start, length) → array	获取x数组从start开始，length个元素组成新的数组。	-
transform(array, function) → array	请参考#unique_34transform()。	-
zip(array1, array2[, ···]) → array	合并多个数组。结果的第M个元素的第N个参数，是原始第N个数组的第M个元素，相当于把多个数组进行了转置。	`SELECT zip(ARRAY[1, 2], ARRAY['1b', null, '3b']); — [ROW(1, '1b'), ROW(2, null), ROW(null, '3b')]`
zip_with(array1, array2, function) → array	请参考#unique_34zip_with。	-
array_agg (key)	array_agg (key)是一个聚合函数，表示把key这一列的所有内容变成一个array返回。	`* | select array_agg(key)`
array_transpose(array[array[a,b,c],array[d,e,f]])	对矩阵进行转置操作。	-

8.22 二进制字符串函数

二进制字符串类型varbinary有别于字符串类型varchar。

语句	说明
连接函数 ||	`a || b` 结果为ab。
length(binary) → bigint	返回二进制的长度。

语句	说明
concat(binary1, ⋯, binaryN) → varbinary	连接二进制字符串，等同于||。
to_base64(binary) → varchar	把二进制字符串转换成base64。
from_base64(string) → varbinary	把base64转换成二进制字符串。
to_base64url(binary) → varchar	转化成url安全的base64。
from_base64url(string) → varbinary	从url安全的base64转化成二进制字符串。
to_hex(binary) → varchar	把二进制字符串转化成十六进制表示。
from_hex(string) → varbinary	从十六进制转化成二进制。
to_big_endian_64(bigint) → varbinary	把数字转化成大端表示的二进制。
from_big_endian_64(binary) → bigint	把大端表示的二进制字符串转化成数字。
md5(binary) → varbinary	计算二进制字符串的md5。
sha1(binary) → varbinary	计算二进制字符串的sha1。
sha256(binary) → varbinary	计算二进制字符串的sha256 hash。
sha512(binary) → varbinary	计算二进制字符串的sha512。
xxhash64(binary) → varbinary	计算二进制字符串的xxhash64。

8.23 位运算

语句	说明	示例

bit_count(x, bits) → bigint	统计x的二进制表示中，1的个数。	`SELECT bit_count(9, 64); — 2`
		`SELECT bit_count(9, 8); — 2`
		`SELECT bit_count(-7, 64); — 62`
		`SELECT bit_count(-7, 8); — 6`

bitwise_and(x, y) → bigint	以二进制的形式求x,y的and的值。	-
bitwise_not(x) → bigint	以二进制的形式求对x的所有位取反。	-
bitwise_or(x, y) → bigint	以二进制形式对x，y求or。	-

语句	说明	示例
bitwise_xor(x, y) → bigint	以二进制形式对x，y求xor。	-

8.24 同比和环比函数

同比和环比函数用于比较当前区间的计算结果和之前一个指定区间的结果。

函数	含义	样例
`compare(value, time_window)`	表示将当前时段计算出来的value值和time_window计算出来的结果进行比较。 value为double或long类型，time_window单位为秒；返回值为数组类型。 返回值分别是当前值、time_window之前的值和当前值与之前值的比值。	`*|select compare( pv , 86400) from (select count(1) as pv from log )`
`compare(value, time_window1, time_window2)`	表示当前区间分别和time_window1和time_window2之前的区间值进行比较，结果为json数组。其中，各个值的大小必须满足以下规则：[当前值，time_window1之前的值，time_window2之前的值，当前值/time_window1之前的值，当前值/time_window2之前的值]。	`* | select compare(pv, 86400, 172800) from ( select count(1) as pv from log)`

函数	含义	样例
compare(value, time_window1, time_window2, time_window3)	表示当前区间分别和time_window1和time_window2,time_window3之前的区间值进行比较，结果为json数组。其中，各个值的大小必须满足以下规则：[当前值, time_window1之前的值，time_window2之前的值，time_window3之前的值，当前值/time_window1之前的值, 当前值/time_window2之前的值,当前值/time_window3之前的值]。	* | select compare(pv, 86400, 172800,604800) from (select count(1) as pv from log)
ts_compare(value, time_window)	表示当前区间分别和time_window1和time_window2之前的区间值进行比较，结果为json数组。其中，各个值的大小必须遵循以下规则：[当前值，time_window1之前的值，当前值/time_window1之前的值，前一个时间起点的unix时间戳]。 用于时序函数比较，需要在SQL中对时间列进行GROUP BY。	例如，* | select t, ts_compare(pv, 86400) as d from(select date_trunc('minute', __time__) as t, count(1) as pv from log group by t order by t) group by t表示将当前时间段每分钟的计算结果和上一个时间段每分钟的计算结果进行比较。 结果为：d:[1251.0,1264.0, 0.9897151898734177, 1539843780.0,1539757380.0]t:2018-10-19 14:23:00.000。

示例

- 计算当前1小时和昨天同一时段的PV比例。

  开始时间为2018-7-25 14:00:00；结束时间为2018-07-25 15:00:00。

  查询分析语句：

  ```
  * | select compare( pv , 86400) from (select count(1) as pv from log
  )
  ```

  其中，86400表示当前时段减去86400秒。

  返回结果：

  ```
  [9.0,19.0,0.47368421052631579]
  ```

  其中，

  - 9.0表示从2018-7-25 14:00:00到2018-07-25 15:00:00的PV值。
  - 19.0表示2018-7-24 14:00:00到2018-07-24 15:00:00的PV值。
  - 0.47368421052631579表示当前时段与之前时段的比值。

  如果要把数组展开成3列数字，分析语句为：

  ```
  * | select diff[1],diff[2],diff[3] from(select compare( pv , 86400)
  as diff from (select count(1) as pv from log))
  ```

- 计算当前1小时内每分钟的PV和昨天同时段的PV比值，并以折线图展示。

  1. 计算当前1小时内每分钟的PV和昨天同时段的PV比值。开始时间为2018-7-25 14:00:00，结束时间为2018-07-25 15:00:00。

     查询分析语句：

     ```
     *| select t, compare( pv , 86400) as diff from (select count(1) as
      pv, date_format(from_unixtime(__time__), '%H:%i') as t from log
     group by t) group by t order by t
     ```

     返回结果：

t	diff
14:00	[9520.0,7606.0,1.2516434393899554]
14:01	[8596.0,8553.0,1.0050274757395066]
14:02	[8722.0,8435.0,1.0340248962655603]

t	diff
14:03	[7499.0,5912.0,1.2684370771312586]

其中t表示时间，格式为`小时:分钟`。diff列的内容是一个数组，分别表示：

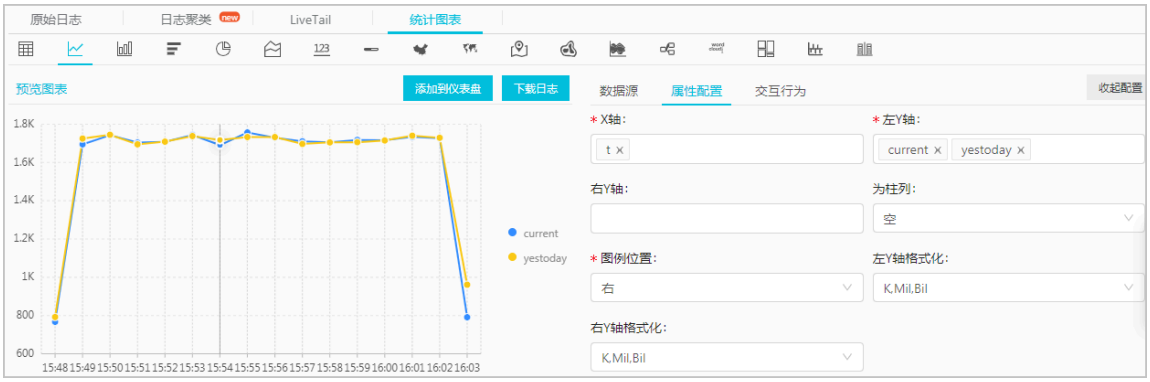
- 当前时段的PV值。
- 之前时段的PV值。
- 当前时段PV值与之前时段比值。

2. 通过以下语句将查询结果展开为折线图形式：

```
*|select t, diff[1] as current, diff[2] as yestoday, diff[3] as
 percentage from(select t, compare( pv , 86400) as diff from (
select count(1) as pv, date_format(from_unixtime(__time__), '%H:%i
') as t from log group by t) group by t order by t)
```

将查询结果配置为折线图，两条线分别表示今天的值和昨天的值：

图 8-1: 折线图



8.25 比较函数和运算符

比较函数和运算符

比较运算判断参数的大小关系，可以应用于任何可比较类型，如int、bigint、double和text等。

比较运算符

比较运算符用于比较两个参数值的大小关系。当用比较运算符比较两个值时，如果逻辑成立，则返回true；否则返回false。

运算符	含义
<	小于
>	大于

运算符	含义
<=	小于或等于
>=	大于或等于
=	等于
<>	不等于
!=	不等于

范围运算符 BETWEEN

BETWEEN用于判断一个参数的值是否在另外两个参数之间，范围为闭区间。

- 如果逻辑成立，则返回true；否则返回false。

  示例：`SELECT 3 BETWEEN 2 AND 6;`逻辑成立，返回true。

  以上样例等同于`SELECT 3 >= 2 AND 3 <= 6;`。
- BETWEEN可以跟在not之后，用于相反逻辑的判断。

  示例：`SELECT 3 NOT BETWEEN 2 AND 6;`，逻辑不成立，返回false。

  以上样例等同于`SELECT 3 < 2 OR 3 > 6;`。
- 如果三个参数中任何一个包含Null，则返回的结果为Null。

IS NULL 和 IS NOT NULL

该运算符用于判断参数是否是Null值。

IS DISTINCT FROM 和 IS NOT DISTINCT FROM

类似于相等和不等判断，区别在于该运算符能够判断存在NULL值的情况。

样例：

```
SELECT NULL IS DISTINCT FROM NULL; -- false
SELECT NULL IS NOT DISTINCT FROM NULL; -- true
```

如下表所示，DISTINCT运算符可以判断多种情况下的参数大小关系。

a	b	a = b	a <> b	a DISTINCT b	a NOT DISTINCT b
1	1	TRUE	FALSE	FALSE	TRUE
1	2	FALSE	TRUE	TRUE	FALSE
1	NULL	NULL	NULL	TRUE	FALSE

a	b	a = b	a <> b	a DISTINCT b	a NOT DISTINCT b
NULL	NULL	NULL	NULL	FALSE	TRUE

GREATEST 和 LEAST

用于获取多列中的最大值或者最小值。

示例：

```
select greatest(1,2,3) ; -- 返回3
```

比较判断：ALL、ANY 和 SOME

比较判断用于判断参数是否满足条件。

- ALL用于判断参数是否满足所有条件。如果逻辑成立，则返回true，否则返回false。
- ANY用于判断参数是否满足条件之一。如果逻辑成立，则返回true，否则返回false。
- SOME和ANY一样，用于判断参数是否满足条件之一。
- ALL、ANY 和 SOME必须紧跟在比较运算符之后。

如下表所示，ALL和ANY支持多种情况下的比较判断。

表达式	含义
A = ALL (…)	A等于所有的值时，结果才是true。
A <> ALL (…)	A不等于所有的值时，结果才是true。
A < ALL (…)	A小于所有的值时，结果才是true。
A = ANY (…)	A等于任何一个值时，结果就为true，等同于 A IN (…)。
A <> ANY (…)	A不等于任何一个值时，结果为true。
A < ANY (…)	A小于其中最大值时，结果为true。

示例：

```
SELECT 'hello' = ANY (VALUES 'hello', 'world'); -- true
SELECT 21 < ALL (VALUES 19, 20, 21); -- false
```

```
SELECT 42 >= SOME (SELECT 41 UNION ALL SELECT 42 UNION ALL SELECT 43);
 -- true
```

8.26 lambda函数

Lambda表达式

lambda表达式的书写形式为`->`。

样例：

```
x -> x + 1
(x, y) -> x + y
x -> regexp_like(x, 'a+')
x -> x[1] / x[2]
x -> IF(x > 0, x, -x)
x -> COALESCE(x, 0)
x -> CAST(x AS JSON)
x -> x + TRY(1 / 0)
```

大多数的MySQL表达式都可以在lambda中使用。

filter(array<T>, function<T, boolean>) → ARRAY<T>

从一个array中过滤数据，只取满足function返回true的元素。

示例：

```
SELECT filter(ARRAY [], x -> true); -- []
SELECT filter(ARRAY [5, -6, NULL, 7], x -> x > 0); -- [5, 7]
SELECT filter(ARRAY [5, NULL, 7, NULL], x -> x IS NOT NULL); -- [5, 7]
```

map_filter(map<K, V>, function<K, V, boolean>) → MAP<K,V>

从map中过滤数据，只取满足function返回true的元素对。

示例：

```
SELECT map_filter(MAP(ARRAY[], ARRAY[]), (k, v) -> true); -- {}
SELECT map_filter(MAP(ARRAY[10, 20, 30], ARRAY['a', NULL, 'c']), (k, v
) -> v IS NOT NULL); -- {10 -> a, 30 -> c}
SELECT map_filter(MAP(ARRAY['k1', 'k2', 'k3'], ARRAY[20, 3, 15]), (k,
v) -> v > 10); -- {k1 -> 20, k3 -> 15}
```

reduce(array<T>, initialState S, inputFunction<S, T, S>, outputFunction<S, R>) → R

reduce函数，从初始状态开始，依次遍历array中的每一个元素，每次在状态S的基础上，计算inputFunction(s,t)，生成新的状态。最终应用outputFunction，把最终状态S变成输出结果R。

1. 初始状态S
2. 遍历每个元素T。

3. 计算inputFunction(S,T)，生成新状态S。

4. 重复2、3，直到最后一个元素被遍历以及生成新状态。

5. 利用最终状态S，获取最终输出结果R。

示例：

```
SELECT reduce(ARRAY [], 0, (s, x) -> s + x, s -> s); -- 0
SELECT reduce(ARRAY [5, 20, 50], 0, (s, x) -> s + x, s -> s); -- 75
SELECT reduce(ARRAY [5, 20, NULL, 50], 0, (s, x) -> s + x, s -> s);
 -- NULL
SELECT reduce(ARRAY [5, 20, NULL, 50], 0, (s, x) -> s + COALESCE(x, 0
), s -> s); -- 75
SELECT reduce(ARRAY [5, 20, NULL, 50], 0, (s, x) -> IF(x IS NULL, s, s
 + x), s -> s); -- 75
SELECT reduce(ARRAY [2147483647, 1], CAST (0 AS BIGINT), (s, x) -> s
 + x, s -> s); -- 2147483648
SELECT reduce(ARRAY [5, 6, 10, 20], -- calculates arithmetic average:
10.25
              CAST(ROW(0.0, 0) AS ROW(sum DOUBLE, count INTEGER)),
              (s, x) -> CAST(ROW(x + s.sum, s.count + 1) AS ROW(sum
DOUBLE, count INTEGER)),
              s -> IF(s.count = 0, NULL, s.sum / s.count));
```

transform(array<T>, function<T, U>) → ARRAY<U>

对数组中的每个元素，依次调用function，生成新的结果U。

示例：

```
SELECT transform(ARRAY [], x -> x + 1); -- []
SELECT transform(ARRAY [5, 6], x -> x + 1); -- [6, 7] 表示对每个元素执行
加1操作
SELECT transform(ARRAY [5, NULL, 6], x -> COALESCE(x, 0) + 1); -- [6,
1, 7]
SELECT transform(ARRAY ['x', 'abc', 'z'], x -> x || '0'); -- ['x0', '
abc0', 'z0']
SELECT transform(ARRAY [ARRAY [1, NULL, 2], ARRAY[3, NULL]], a ->
filter(a, x -> x IS NOT NULL)); -- [[1, 2], [3]]
```

transform_keys(map<K1, V>, function<K1, V, K2>) → MAP<K2,V>

依次对map中的每个key应用函数，生成新的key。

示例：

```
SELECT transform_keys(MAP(ARRAY[], ARRAY[]), (k, v) -> k + 1); -- {}
SELECT transform_keys(MAP(ARRAY [1, 2, 3], ARRAY ['a', 'b', 'c']), (k
, v) -> k + 1); -- {2 -> a, 3 -> b, 4 -> c}  表示对每个key执行加1操作。
SELECT transform_keys(MAP(ARRAY ['a', 'b', 'c'], ARRAY [1, 2, 3]), (k
, v) -> v * v); -- {1 -> 1, 4 -> 2, 9 -> 3}
SELECT transform_keys(MAP(ARRAY ['a', 'b'], ARRAY [1, 2]), (k, v) -> k
 || CAST(v as VARCHAR)); -- {a1 -> 1, b2 -> 2}
SELECT transform_keys(MAP(ARRAY [1, 2], ARRAY [1.0, 1.4]), -- {one ->
1.0, two -> 1.4}
```

```
                        (k, v) -> MAP(ARRAY[1, 2], ARRAY['one', 'two'])[
k]);
```

transform_values(map<K, V1>, function<K, V1, V2>) → MAP<K, V2>

对map中的所有value应用function函数，把V1变成V2，生成新的map<K,V2>。

```
SELECT transform_values(MAP(ARRAY[], ARRAY[]), (k, v) -> v + 1); -- {}
SELECT transform_values(MAP(ARRAY [1, 2, 3], ARRAY [10, 20, 30]), (k,
v) -> v + 1); -- {1 -> 11, 2 -> 22, 3 -> 33}
SELECT transform_values(MAP(ARRAY [1, 2, 3], ARRAY ['a', 'b', 'c']), (
k, v) -> k * k); -- {1 -> 1, 2 -> 4, 3 -> 9}
SELECT transform_values(MAP(ARRAY ['a', 'b'], ARRAY [1, 2]), (k, v) -
> k || CAST(v as VARCHAR)); -- {a -> a1, b -> b2}
SELECT transform_values(MAP(ARRAY [1, 2], ARRAY [1.0, 1.4]), -- {1 ->
one_1.0, 2 -> two_1.4}
                        (k, v) -> MAP(ARRAY[1, 2], ARRAY['one', 'two
'])[k] || '_' || CAST(v AS VARCHAR));
```

zip_with(array<T>, array<U>, function<T, U, R>) → array<R>

合并两个array，通过函数指定生成的新的array中的元素。第一个数组的元素T和第二个数组元素U，生成新的结果R。

示例：

```
SELECT zip_with(ARRAY[1, 3, 5], ARRAY['a', 'b', 'c'], (x, y) -> (y,
x)); --表示调换前后两个数组的元素位置，生成一个新的数组。结果： [ROW('a', 1),
ROW('b', 3), ROW('c', 5)]
SELECT zip_with(ARRAY[1, 2], ARRAY[3, 4], (x, y) -> x + y); -- 结果[4,
6]
SELECT zip_with(ARRAY['a', 'b', 'c'], ARRAY['d', 'e', 'f'], (x, y) ->
concat(x, y)); 表示把前后两个数组的元素拼接，生成一个新的字符串。结果： ['ad',
 'be', 'cf']
```

map_zip_with(map<K, V1>, map<K, V2>, function<K, V1, V2, V3>) → map<K, V3>

合并两个map，针对每个key，由两个value V1和V2生成V3。生成新的Map<K,V3>。

```
SELECT map_zip_with(MAP(ARRAY[1, 2, 3], ARRAY['a', 'b', 'c']),
                    MAP(ARRAY[1, 2, 3], ARRAY['d', 'e', 'f']),
                    (k, v1, v2) -> concat(v1, v2));  表示把两个map key相
同的value进行合并。-- {1 -> ad, 2 -> be, 3 -> cf}
SELECT map_zip_with(MAP(ARRAY['k1', 'k2'], ARRAY[1, 2]),
                    MAP(ARRAY['k2', 'k3'], ARRAY[4, 9]),
                    (k, v1, v2) -> (v1, v2));表示把两个value生成一个数
组。-- {k1 -> ROW(1, null), k2 -> ROW(2, 4), k3 -> ROW(null, 9)}
SELECT map_zip_with(MAP(ARRAY['a', 'b', 'c'], ARRAY[1, 8, 27]),
                    MAP(ARRAY['a', 'b', 'c'], ARRAY[1, 2, 3]),
```

```
                    (k, v1, v2) -> k || CAST(v1/v2 AS VARCHAR)); 表示在
结果中连接key的值和两个value的相除结果-- {a -> a1, b -> b4, c -> c9}
```

8.27 逻辑函数

逻辑运算符

表 8-2: 逻辑运算符

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

NULL参与逻辑运算

a和b分别取值TRUE FALSE和NULL时的真值表如下：

表 8-3: 真值表1

a	b	a AND b	a OR b
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	NULL	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL
NULL	TRUE	NULL	TRUE
NULL	FALSE	FALSE	NULL
NULL	NULL	NULL	NULL

表 8-4: 真值表2

a	NOT a
TRUE	FALSE

a	NOT a
FALSE	TRUE
NULL	NULL

8.28 列的别名

在SQL标准中，列名必须由字母、数字、下划线组成，且以字母开头。

如果在日志收集配置中，用户如果配置了不符合SQL标准的列名(例如User-Agent)，那么需要在配置统计属性的页面，给列取一个别名，用于查询。别名仅仅用于SQL统计，在底层存储时，仍然是原始名称，搜索时需要使用原始名称。

此外，当用户原始的列名特别长时，也可以取一个别名来代替原始列名查询。

表 8-5: 别名样例

原始列名	别名
User-Agent	ua
User.Agent	ua
123	col
abceefghijklmnopqrstuvw	a

8.29 Logstore和RDS联合查询

背景信息

日志服务支持Logstore中的日志数据和RDS数据库进行联合查询，以及把查询结果保存到RDS中。

操作步骤

1. 创建RDS VPC，并设置白名单。
   a) 创建RDS，并指定VPC环境。创建成功后，得到VPC ID和RDS实例ID。
   b) 设置RDS白名单：`100.104.0.0/16`。
   详细步骤请参考《*RDS*用户指南》中设置白名单章节。
2. 创建External Store。
   通过以下语句创建External Store，请将参数替换为您的实际参数值。

```
{
"externalStoreName":"storeName",
```

```
"storeType":"rds-vpc",
"parameter":
   {
   "region":"cn-qingdao",
   "vpc-id":"vpc-m5eq4irc1pucp*******"
   "instance-id":"i-m5eeo2whsn*******"
   "host":"localhost",
   "port":"3306",
   "username":"root",
   "password":"****",
   "db":"scmc"
   "table":"join_meta"
   }
}
```

表 8-6: 参数说明

参数	说明
region	您的服务所在区域。
vpc-id	VPC的ID。
instance-id	RDS实例ID。
host	ECS实例ID。
port	ECS实例端口。
username	用户名。
password	密码。
db	数据库。
table	数据表。

说明:

目前仅支持北京（cn-beijing）、青岛（cn-qingdao）和杭州（cn-hangzhou）区域。

3. Join查询。

   在日志服务控制台查询页面执行Join语句。

   支持的Join语法：

   - INNER JOIN
   - LEFT JOIN
   - RIGHT JOIN
   - FULL JOIN

```
[ INNER ] JOIN
LEFT [ OUTER ] JOIN
RIGHT [ OUTER ] JOIN
```

```
FULL [ OUTER ] JOIN
```

说明:

- **Join仅支持Logstore Join小表。**
- **在Join顺序中，Logstore必须写在前部，External Store写在后部。**
- **Join中必须写External Store的名称，会自动替换成RDS的db+表名。不能直接填写RDS表名。**

Join语法样例:

```
method:postlogstorelogs | select count(1) , histogram(logstore) from
 log  l join join_meta m on l.projectid = cast( m.ikey as varchar)
```

4. **保存查询结果到RDS中。**

   支持通过Insert语法把查询结果插入到RDS中。

```
method:postlogstorelogs | insert into method_output  select cast(
methodasvarchar(65535)),count(1)fromloggroupbymethod
```

Python 程序样例

```
# encoding: utf-8
from __future__ import print_function
from aliyun.log import *
from aliyun.log.util import base64_encodestring
from random import randint
import time
import os
from datetime import datetime
    endpoint = os.environ.get('ALIYUN_LOG_SAMPLE_ENDPOINT',
 'cn-chengdu.log.aliyuncs.com')
    accessKeyId = os.environ.get('ALIYUN_LOG_SAMPLE_ACCESSID
', '')
    accessKey = os.environ.get('ALIYUN_LOG_SAMPLE_ACCESSKEY
', '')
    logstore = os.environ.get('ALIYUN_LOG_SAMPLE_LOGSTORE',
 '')
    project = "ali-yunlei-chengdu"
    client = LogClient(endpoint, accessKeyId, accessKey,
token)
#创建external store
    res = client.create_external_store(project,ExternalSt
oreConfig("rds_store","region","rds-vpc","vpc 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external store详情
    res = client.get_external_store(project,"rds_store");
    res.log_print()
    res = client.list_external_store(project,"");
    res.log_print();
    # JOIN查询
    req = GetLogStoreLogsRequest(project,logstore,From,To
,"","select count(1) from  "+ logstore +"  s join  meta m on
  s.projectid = cast(m.ikey as varchar)");
```

```
    res = client.get_logs(req)
    res.log_print();
     # 查询结果写入RDS
    req = GetLogStoreLogsRequest(project,logstore,From,
To,"","," insert into rds_store select count(1) from  "+
logstore );
    res = client.get_logs(req)
    res.log_print();
```

8.30 空间几何函数

空间几何概念

空间几何函数支持Well-Known Text (WKT) 格式描述的几何实体。

表 8-7: 几何实体格式

几何实体	Well-Known Text (WKT) 格式
点	`POINT (0 0)`
线段	`LINESTRING (0 0, 1 1, 1 2)`
多边形	`POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0), (1 1, 2 1, 2 2, 1 2, 1 1))`
多点	`MULTIPOINT (0 0, 1 2)`
多线段	`MULTILINESTRING ((0 0, 1 1, 1 2), (2 3, 3 2, 5 4))`
多个多边形	`MULTIPOLYGON (((0 0, 4 0, 4 4, 0 4, 0 0), (1 1, 2 1, 2 2, 1 2, 1 1)), ((-1 -1, -1 -2, -2 -2, -2 -1, -1 -1)))`
空间实体集合	`GEOMETRYCOLLECTION (POINT(2 3), LINESTRING (2 3, 3 4))`

构造空间实体

表 8-8: 构造空间实体函数说明

函数	说明
ST_Point(double, double) → Point	构造一个点。
ST_LineFromText(varchar) → LineString	从WKT格式的文本中构造一个线段。
ST_Polygon(varchar) → Polygon	从WKT格式的文本中构造一个多边形。

函数	说明
ST_GeometryFromText(varchar) → Geometry	从WKT文本中构造一个空间几何实体。
ST_AsText(Geometry) → varchar	把一个空间几何实体转变成WKT格式。

运算符

函数	说明
ST_Boundary(Geometry) → Geometry	计算几何实体的闭包。
ST_Buffer(Geometry, distance) → Geometry	返回一个多边形，该多边形距离输入参数Geometry的距离是distance。
ST_Difference(Geometry, Geometry) → Geometry	返回两个空间实体的不同的点的集合。
ST_Envelope(Geometry) → Geometry	返回空间实体的边界多边形。
ST_ExteriorRing(Geometry) → Geometry	返回多边形的外部环。
ST_Intersection(Geometry, Geometry) → Geometry	返回两个空间实体的交集点。
ST_SymDifference(Geometry, Geometry) → Geometry	返回两个空间实体不同的点，组成的新的空间实体。获取两个几何对象不相交的部分。

空间关系判断

函数	说明
ST_Contains(Geometry, Geometry) → boolean	当第二个实体的所有点都不在第一个实体外部，并且第一个实体至少有一个内部点在第二个实体内部时，返回true。如果第二个实体正好在第一个实体的边上，那么是false。
ST_Crosses(Geometry, Geometry) → boolean	当两个实体有共同内部点时，返回true。
ST_Disjoint(Geometry, Geometry) → boolean	当两个实体没有任何交集时，返回true。
ST_Equals(Geometry, Geometry) → boolean	当两个实体完全相同时，返回true。
ST_Intersects(Geometry, Geometry) → boolean	当两个实体在两个空间上共享时，返回true。
ST_Overlaps(Geometry, Geometry) → boolean	当两个实体维度相同，并且不是包含关系时，返回true。

函数	说明
ST_Relate(Geometry, Geometry) → boolean	当两个实体相关时，返回true。
ST_Touches(Geometry, Geometry) → boolean	当两个实体仅仅边界有联系，没有共同内部点时，返回true。
ST_Within(Geometry, Geometry) → boolean	当第一个实体完全在第二个实体内部时，返回true。如果边界有交集，返回false。

Accessors

函数	说明
ST_Area(Geometry) → double	使用欧几里得测量法，计算多边形在二维平面上的投影面积。
ST_Centroid(Geometry) → Geometry	返回几何实体的中心点。
ST_CoordDim(Geometry) → bigint	返回几何实体的坐标维度。
ST_Dimension(Geometry) → bigint	返回几何实体的固有维度，必须小于或等于坐标维度。
ST_Distance(Geometry, Geometry) → double	计算两个实体之间的最小距离。
ST_IsClosed(Geometry) → boolean	当实体时一个闭合空间时，返回true。
ST_IsEmpty(Geometry) → boolean	当参数时一个空的几何实体集合或者多边形或者点时返回true。
ST_IsRing(Geometry) → boolean	当参数是一条线，并且时闭合的简单的线时，返回true。
ST_Length(Geometry) → double	在二维投影平面上，使用欧几里得测量法计算一个线段或者多条线段的长度。返回一个行字符串或多行字符串的长度。该长度是采用欧几里得测量法基于空间参考对二维平面的预测。
ST_XMax(Geometry) → double	返回几何体边框的X最大值。
ST_YMax(Geometry) → double	返回几何体边框的Y最大值。
T_XMin(Geometry) → double	返回几何体边框的X最小值。
ST_YMin(Geometry) → double	返回结合体边框的Y最小值。
ST_StartPoint(Geometry) → point	返回线段类型几何体的第一个点。
ST_EndPoint(Geometry) → point	返回线段类型几何体的最后一个点。
ST_X(Point) → double	返回点类型的X轴。

函数	说明
ST_Y(Point) → double	返回点类型的Y轴。
ST_NumPoints(Geometry) → bigint	计算几何实体的点的个数。
ST_NumInteriorRing(Geometry) → bigint	返回多边形内部的环的个数。

8.31 地理函数

IP转国家、省、城市、运营商、经纬度，请参考文档#unique_28。

表 8-9: 地理函数

函数名	含义	样例
geohash(string)	将纬度、经度用geohash编码，string为字符串类型，内容是纬度、逗号、经度。	`select geohash('34.1,120.6')= 'wwjcbrdnzs'`
geohash(lat, lon)	将纬度、经度用geohash编码，参数分别是纬度和经度。	`select geohash(34.1,120.6)= 'wwjcbrdnzs'`

8.32 Join语法

Join用于多表中字段之间的联系。日志服务除了支持单个Logstore的Join之外，还支持Logstore和RDS的Join，以及Logstore和Logstore的Join。本文档为您介绍如何使用跨Logstore的Join功能。

操作步骤

1. 下载最新版本Python SDk。
2. 使用GetProjectLogs接口进行查询。

SDK示例

```
#!/usr/bin/env python
#encoding: utf-8
import time,sys,os
from aliyun.log.logexception import LogException
from aliyun.log.logitem import LogItem
from aliyun.log.logclient import LogClient
from aliyun.log.getlogsrequest import GetLogsRequest
from aliyun.log.getlogsrequest import GetProjectLogsRequest
from aliyun.log.putlogsrequest import PutLogsRequest
from aliyun.log.listtopicsrequest import ListTopicsRequest
from aliyun.log.listlogstoresrequest import ListLogstoresRequest
from aliyun.log.gethistogramsrequest import GetHistogramsRequest
from aliyun.log.index_config import *
from aliyun.log.logtail_config_detail import *
```

```
from aliyun.log.machine_group_detail import *
from aliyun.log.acl_config import *
if __name__=='__main__':
    token = None
    endpoint = "http://cn-hangzhou.log.aliyuncs.com"
    accessKeyId  = 'LTAIvKy7U'
    accessKey='6gXLNTLyCfdsfwrewrfhdskfdsfuiwu'
    client = LogClient(endpoint, accessKeyId, accessKey,token)
    logstore = "meta"
    # 在查询语句中，指定两个Logstore，每个Logstore要分别指定各自的时间范围，以及两个Logstore关联的key
    req = GetProjectLogsRequest(project,"select count(1) from sls_operation_log  s join  meta m on s.__date__ >'2018-04-10 00:00:00' and s.__date__ < '2018-04-11 00:00:00' and m.__date__ >'2018-04-23 00:00:00' and m.__date__ <'2018-04-24 00:00:00' and s.projectid = cast(m.ikey as varchar)");
    res = client.get_project_logs(req)
    res.log_print();
    exit(0)
```

8.33 unnest语法

应用场景

处理数据时，一列数据通常为字符串或数字等primitive类型的数据。在复杂的业务场景下，日志数据的某一列可能会是较为复杂的格式，例如数组（array）、对象(map)、JSON等格式。对这种特殊格式的日志字段进行查询分析，可以使用unnest语法。

例如以下日志：

```
__source__:  1.1.1.1
__tag__:__hostname__:  vm-req-170103232316569850-tianchi111932.tc
__topic__:  TestTopic_4
array_column:  [1,2,3]
double_column:  1.23
map_column:  {"a":1,"b":2}
text_column:  商品
```

其中`array_column`字段为数组类型。如果统计`array_column`中所有数值的汇总值，需要遍历每一行的数组中的每一个元素。

unnest语法结构

语法	说明
`unnest( array) as table_alias( column_name)`	表示把array类型展开成多行，行的名称为`column_name`。
`unnest(map) as table(key_name, value_name)`	表示把map类型展开成多行，key的名称为`key_name`，value的名称为`value_name`。

说明：

> 注意，由于unnest接收的是array或者map类型的数据，如果您的输入为字符串类型，那么要先转化成json类型，然后再转化成array类型或map类型，转化的方式为`cast(json_parse(array_column) as array(bigint))`。

遍历数组每一个元素

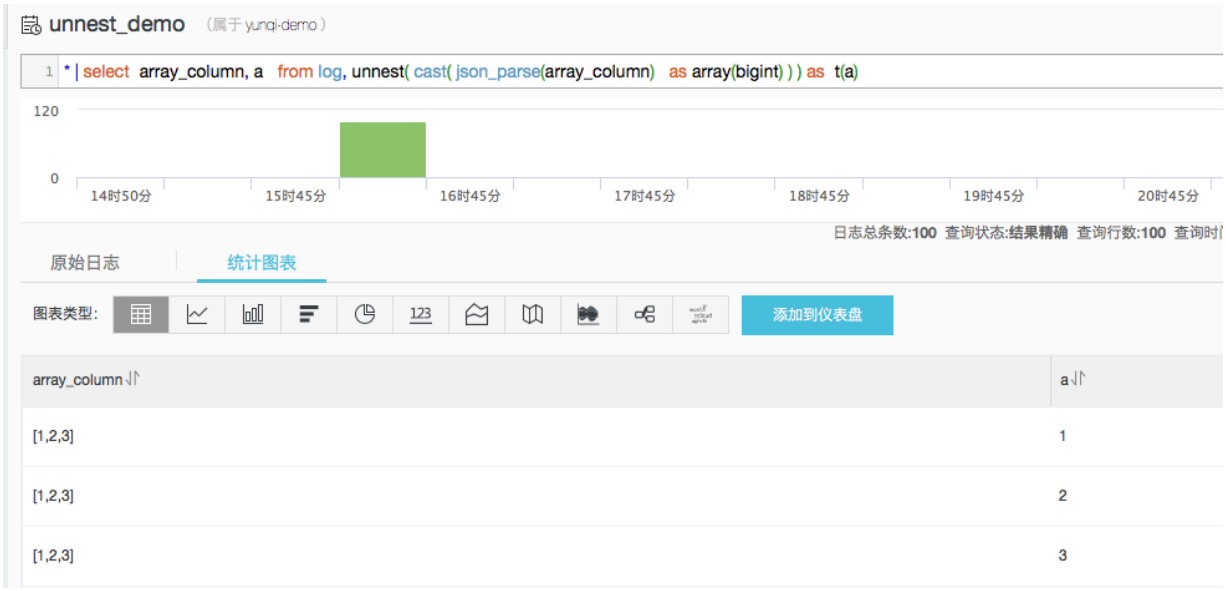
使用SQL把array展开成多行：

```
* | select  array_column, a   from log, unnest( cast( json_parse(array_column)   as array(bigint) ) ) as  t(a)
```

上述SQL把数组展开成多行数字，`unnest( cast( json_parse(array_column) as array(bigint) ) ) as t(a)`，unnest语法把数组展开，以t来命名新生成的表，使用a来引用展开后的列。

结果如下图：

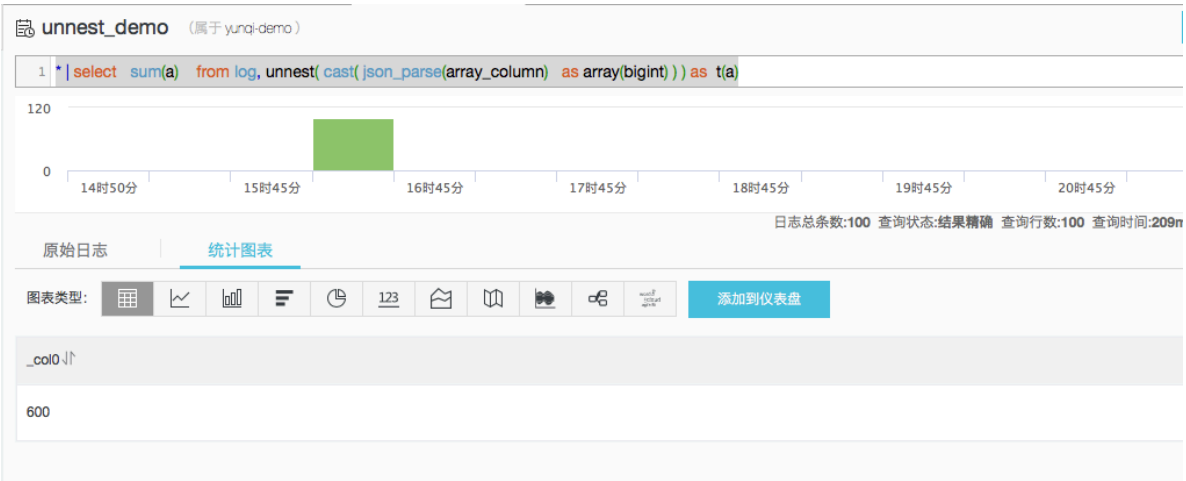
图 8-2: 展开数组



- 统计数组中的每个元素的和：

```
* | select   sum(a)    from log, unnest( cast( json_parse(array_colu
mn)   as array(bigint) ) ) as  t(a)
```

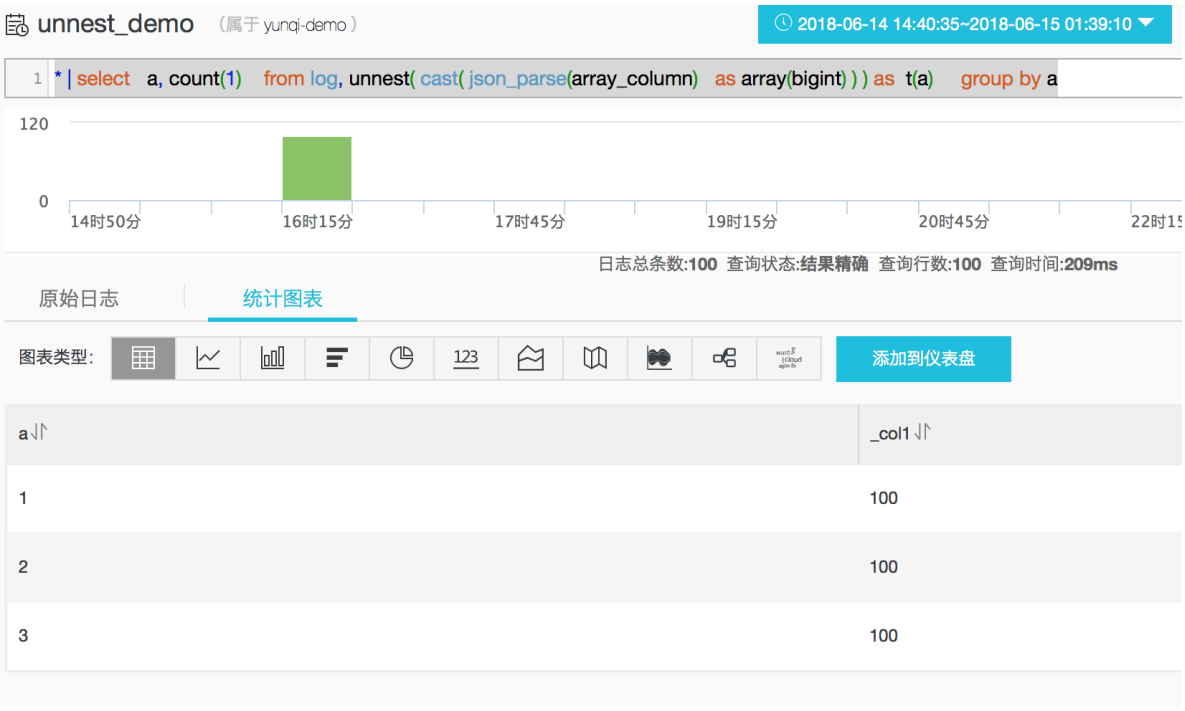
图 8-3: 对数组进行sum计算



- 按照数组中的每个元素进行group by计算：

```
* | select   a, count(1)    from log, unnest( cast( json_parse(
array_column)   as array(bigint) ) ) as  t(a)     group by a
```

图 8-4: 对数组进行group by计算

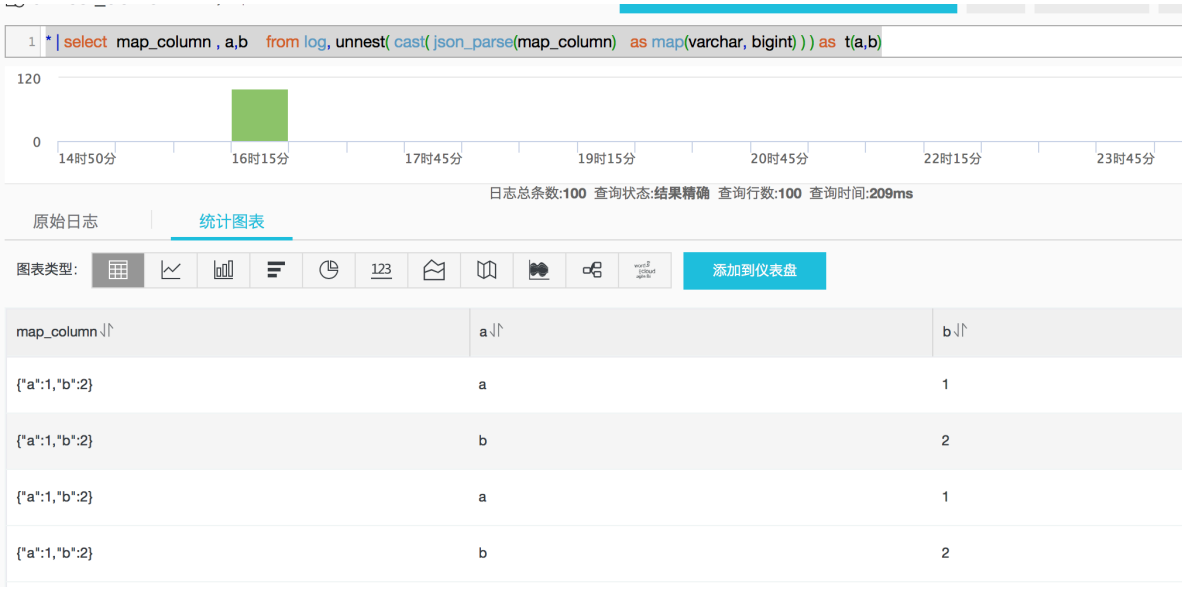


遍历Map

- **遍历Map中的元素：**

```
* | select  map_column , a,b    from log, unnest( cast( json_parse(
map_column)   as map(varchar, bigint) ) ) as  t(a,b)
```

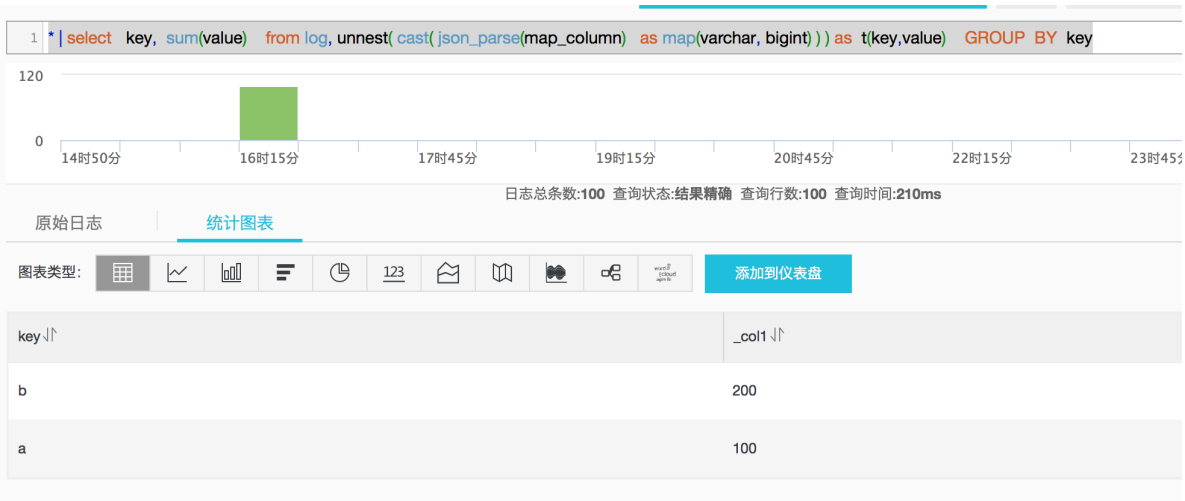
图 8-5: 遍历Map



- **按照Map的key进行group by 统计：**

```
* | select   key,  sum(value)    from log, unnest( cast( json_parse
(map_column)   as map(varchar, bigint) ) ) as  t(key,value)    GROUP
  BY  key
```

图 8-6: 对Key进行group by统计



格式化显示histogram,numeric_histogram的结果

- histogram

  histogram函数类似于count group by 语法。语法请参考#unique_18。

  通常情况下histogram的结果为一串json数据，无法配置视图展示，例如：

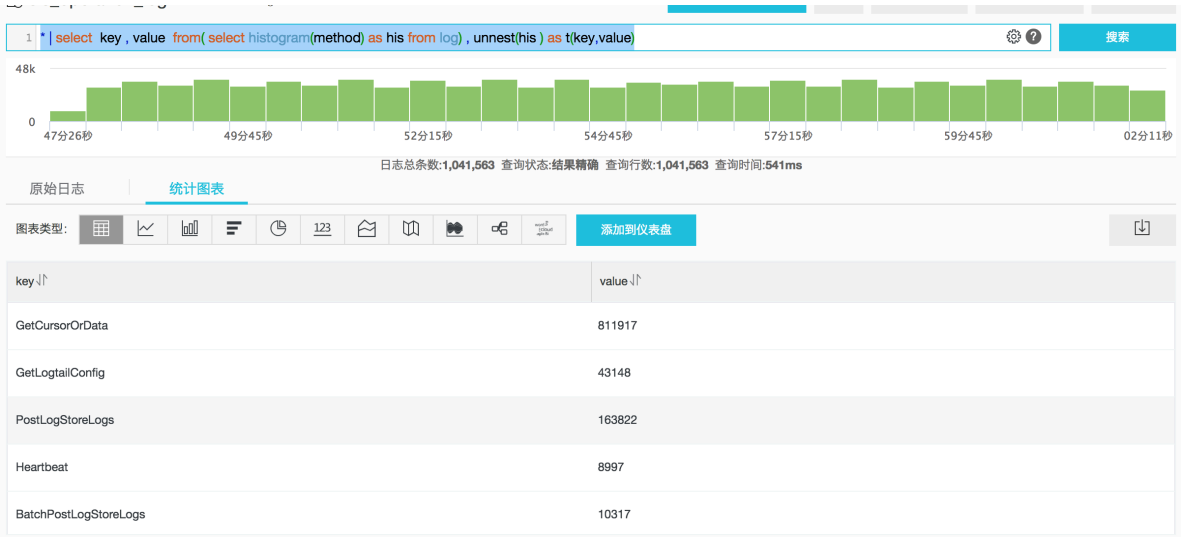
```
* | select histogram(method)
```

图 8-7: 普通histogram结果

您可以通过unnest语法，把JSON展开成多行配置视图，例如：

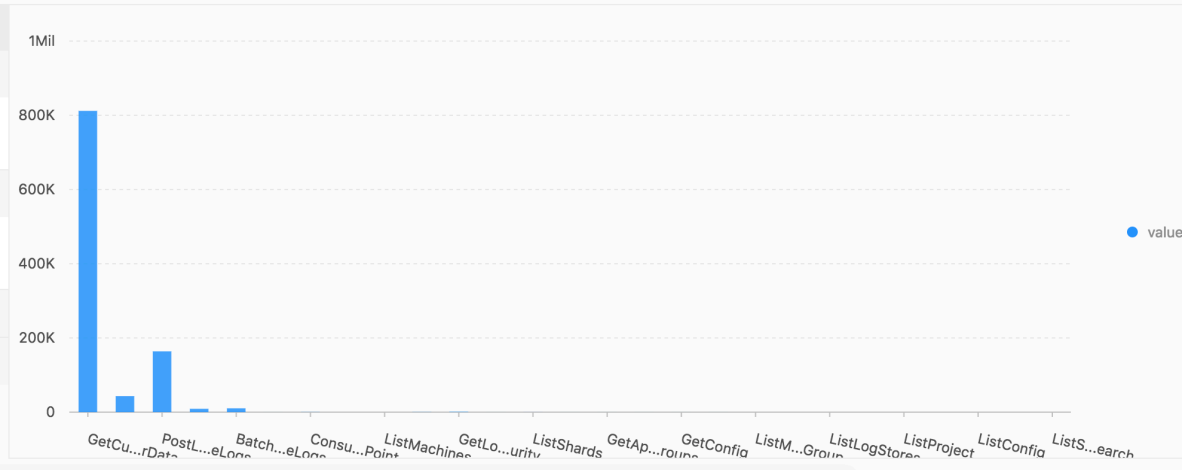
```
* | select  key , value  from( select histogram(method) as his from log) , unnest(his ) as t(key,value)
```

图 8-8: 展开JSON



接下来，可以配置可视化视图：

图 8-9: 可视化视图1



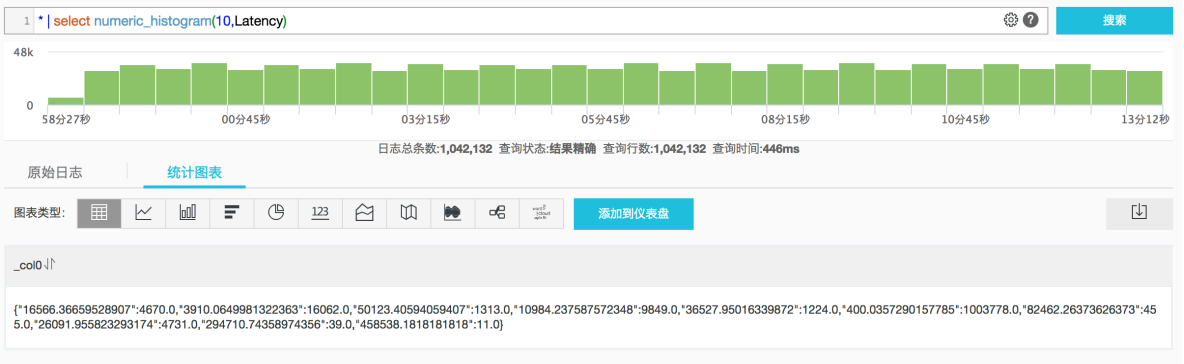
- numeric_histogram

  numeric_histogram语法是为了把数值列分配到多个桶中去，相当于对数值列进行group by，具体语法请参考#unique_19。

```
* | select numeric_histogram(10,Latency)
```

numeric_histogram的输出如下：

图 8-10: numeric_histogram查询结果

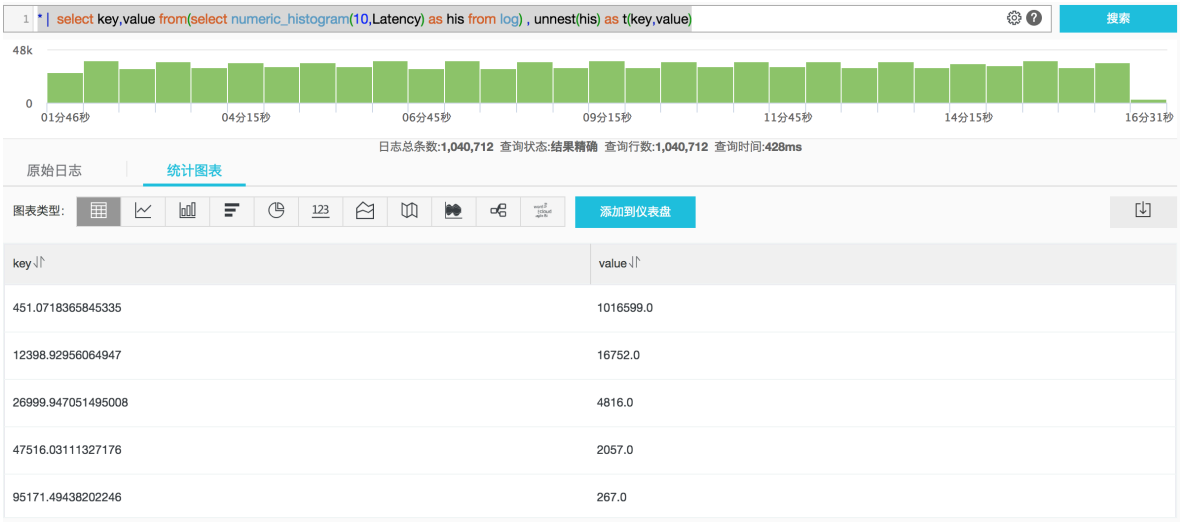


您可以通过以下查询语句格式化展示该结果：

```
* |  select key,value from(select numeric_histogram(10,Latency) as
his from log) , unnest(his) as t(key,value)
```

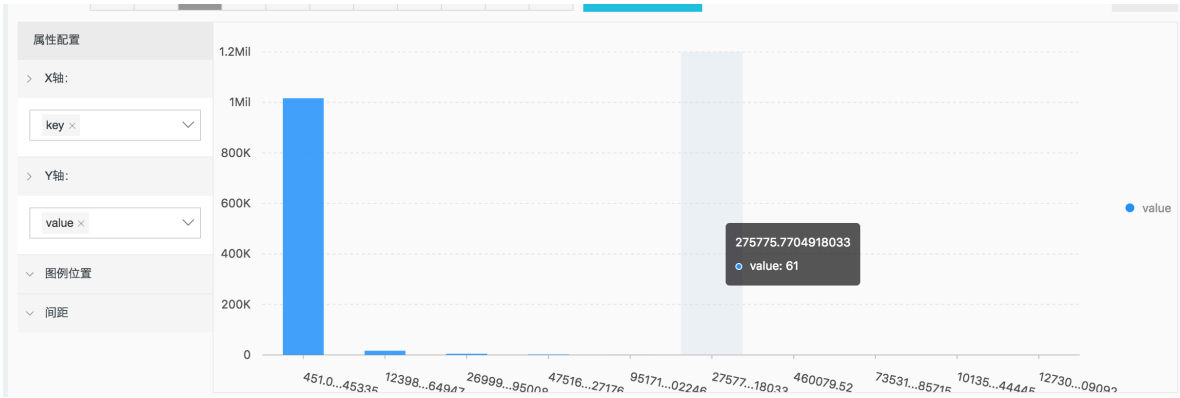
结果如下：

图 8-11: 查询结果



同时配置柱状图的形式展示：

图 8-12: 可视化视图2



8.34 电话号码函数

电话号码函数提供对中国大陆区域电话号码的归属地查询功能。

函数列表

函数名	含义	样例
mobile_province	查看电话号码所属省份，需要传入电话的数字形式。字符串参数可以使用try_cast进行转换。	* | select mobile_province(12345678) * | select mobile_province(try_cast('12345678' as bigint))
mobile_city	查看电话号码所属城市，需要传入电话的数字形式。字符串参数可以使用try_cast进行转换。	* | select mobile_city(12345678) * | select mobile_city(try_cast('12345678' as bigint))
mobile_carrier	查看电话号码所属运营商，需要传入电话的数字形式。字符串参数可以使用try_cast进行转换。	* | select mobile_carrier(12345678) * | select mobile_carrier(try_cast('12345678' as bigint))

应用场景

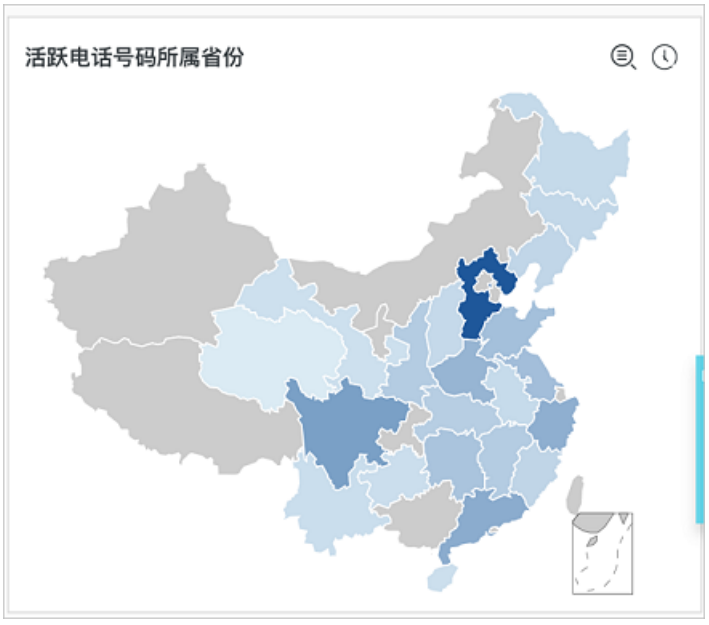
- 查询电话号码所属地并生成报表

  某电商收集客户参加活动的日志信息，其中有用户电话号码的字段，对电话号码归属地进行统计，可以实现如下查询分析语句：

```
SELECT mobile_city(try_cast("mobile" as bigint)) as "城市",
mobile_province(try_cast("mobile" as bigint)) as "省份", count(1) as
 "请求次数" group by "省份", "城市" order by "请求次数" desc limit 100
```

  这里将日志中的`mobile`字段传给了`mobile_city`和`mobile_province`函数，展示其所在省和城市等信息。返回如下：

  还可以选择地图视图，进行可视化如下：



- **根据电话所属地检查并通知**

  例如，某证券运营商收集了根据客户的电话号码所属地，及其访问服务时的IP地址，想要整理出哪些客户平时访问地址与电话所属地不同：

```
* | select mobile, client_ip, count(1) as PV where mobile_cit
y(try_cast("mobile" as bigint)) != ip_to_city(client_ip) and
ip_to_city(client_ip) != '' group by client_ip, mobile order by PV
desc
```

  可以以此创建告警规则，详细说明请参考日志服务告警。

9 机器学习语法与函数

9.1 简介

日志服务（Log Service）机器学习功能为您提供多种功能丰富的算法和便捷的调用方式，您可以在日志查询分析中通过SELECT语句和机器学习函数调用机器学习算法，分析某一字段或若干字段在一段时间内的特征。

尤其是针对时序数据分析场景，日志服务提供了丰富的时序分析算法，可以帮助您快速解决时序预测、时序异常检测、序列分解、多时序聚类等场景问题，兼容SQL标准接口，大大降低了您使用算法的门槛，提高分析问题和解决问题的效率。

功能特点

- 支持单时序序列的多种平滑操作。
- 支持单时序序列的预测、异常检测、变点检测、折点检测、多周期估计算法。
- 支持单时序序列的分解操作。
- 支持多时序序列的多种聚类算法。
- 支持多字段（数值列、文本列）的模式挖掘。

限制说明

- 输入的时序数据必须是基于相同时间间隔的采样数据。
- 输入的时序数据中不能含有重复时间点的数据。

限制项	说明
时序数据处理的有效容量	上限为150,000个连续时间点数据。 若数量超过上限，请进行聚合操作或者降采样。
密度聚类算法的聚类容量	上限为5000条时序曲线，每条时序曲线的长度最大为1440个点。
层次聚类算法的聚类容量	上限为2000条时序曲线，每条时序曲线的长度最大为1440个点。

机器学习函数

	类别	函数	说明
时间序列	平滑函数	ts_smooth_simple	使用Holt Winters算法对时序数据平滑。

	类别	函数	说明
		ts_smooth_fir	使用FIR滤波器对时序数据平滑。
		ts_smooth_iir	使用IIR滤波器对时序数据平滑。
	多周期估计函数	ts_period_detect	对时序数据进行分段周期估计。
	变点检测函数	ts_cp_detect	寻找时序序列中具有不同统计特性的区间，区间端点即为变点。
		ts_breakout_detect	寻找时序序列中，某统计量发生陡升或陡降的点。
	极大值检测函数	ts_find_peaks	极大值检测函数用于在指定窗口中寻找序列的局部极大值。
	预测与异常检测函数	ts_predicate_simple	利用默认参数对时序数据进行建模，并进行简单的时序预测和异常点的检测。
		ts_predicate_ar	使用自回归模型对时序数据进行建模，并进行简单的时序预测和异常点的检测。
		ts_predicate_arma	使用移动自回归模型对时序数据进行建模，并进行简单的时序预测和异常点检测。
		ts_predicate_arima	使用带有差分的移动自回归模型对时序数据进行建模，并进行简单的时序预测和异常点检测。
		ts_regression_predict	针对含有周期性、趋势性的单时序序列，进行准确且长时序预测。
	序列分解函数	ts_decompose	使用STL算法对时序数据进行序列分解。
	时序聚类函数	ts_density_cluster	使用密度聚类方法对多条时序数据进行聚类。
		ts_hierarchical_cluster	使用层次聚类方法对多条时序数据进行聚类。
		ts_similar_instance	查找到指定曲线名称的相似曲线。

	类别	函数	说明
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。
	#unique_129	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。

9.2 平滑函数

平滑函数是针对输入的时序曲线进行平滑和简单的滤波操作，滤波操作通常是发现时序曲线形态的第一步。

函数列表

函数	说明
ts_smooth_simple	默认平滑函数，使用Holt Winters算法对时序数据平滑。
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

ts_smooth_simple

函数格式：

```
select ts_smooth_simple(x, y)
```

参数说明如下：

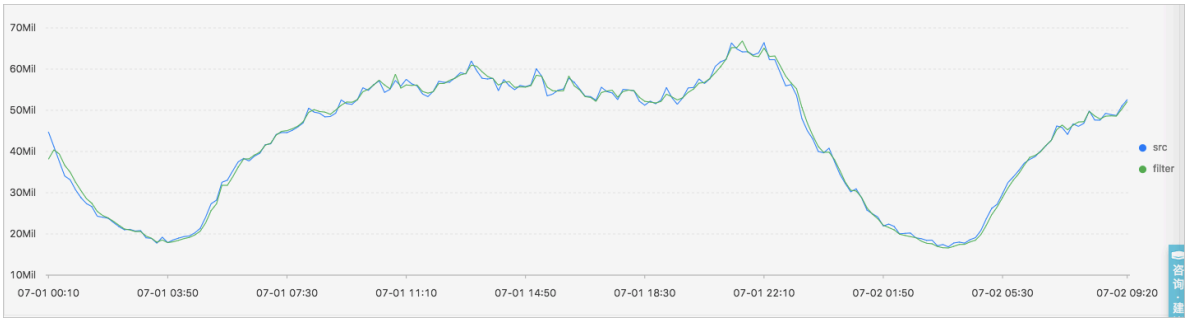
参数	说明	取值
x	时间列，顺序为从小到大。	Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	--

示例：

- **查询分析：**

```
* | 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：**



显示项如下：

显示项		说明
横轴	unixtime	数据的时间戳，单位为秒。
纵轴	src	未滤波前的数据。
	filter	滤波之后的数据。

ts_smooth_fir

函数格式：

- **当您无法确定滤波参数时，可以使用内置窗口的参数进行滤波操作：**

```
select ts_smooth_fir(x, y,winType,winSize,samplePeriod,sampleMethod)
```

- **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[],samplePeriod,sampleMethod)
```

参数说明如下：

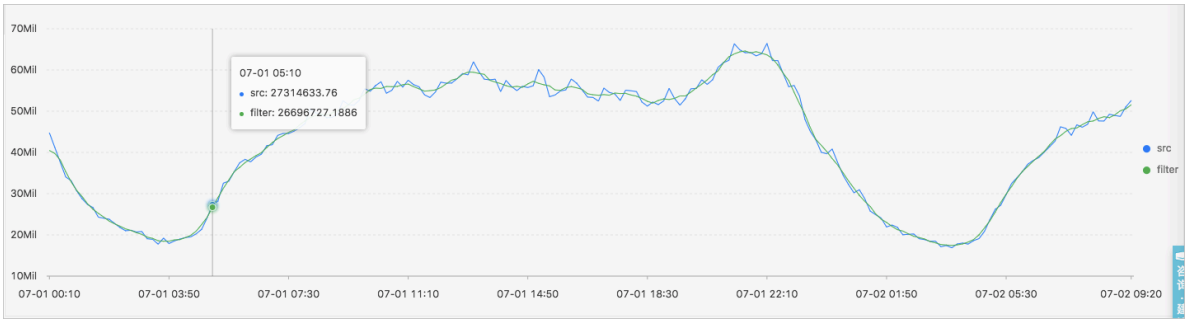
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
winType	滤波的窗口类型。	取值包括： · rectangle：矩形窗口。 · hanning：汉宁窗。 · hamming：汉明窗。 · blackman：布莱克曼窗。 说明: 推荐您选择rectangle类型以获得更好的显示效果。
winSize	滤波窗口的长度。	long类型，取值范围为2~15。
array[]	FIR滤波的具体参数。	格式为数组，且数组中元素的和为1。例如array[0.2, 0.4, 0.3, 0.1]。
samplePeriod	对当前时序数据进行采样的周期。	long类型，单位为秒。取值范围为1~86399。
sampleMethod	针对采样窗口内数据的采样方法。	取值包括： · avg：表示取窗口内数据的平均值。 · max：表示取窗口内数据的最大值。 · min：表示取窗口内数据的最小值。 · sum：表示取窗口内数据的总和。

示例：

- **查询分析：**

```
* | select ts_smooth_fir(stamp, value, 'rectangle', 4, 1, 'avg')
from ( select __time__ - __time__ % 120 as stamp, avg(v) as value
from log GROUP BY stamp order by stamp )
```

输出结果：



- **查询分析：**

```
* | select ts_smooth_fir(stamp, value, array[0.2, 0.4, 0.3, 0.1], 1,
 'avg') from ( select __time__ - __time__ % 120 as stamp, avg(v) as
value from log GROUP BY stamp order by stamp )
```

输出结果：

显示项如下：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
纵轴	src	未滤波前的数据。
	filter	滤波之后的数据。

ts_smooth_iir

函数格式：

```
select
 ts_smooth_iir(x, y, array[], array[], samplePeriod, sampleMethod)
```

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。	-
array[]	IIR滤波算法中关于x_i的具体参数。	数组格式，长度（length）的取值范围为2~15，且数组中元素的和为1。例如array[0.2, 0.4, 0.3, 0.1]。
array[]	IIR滤波算法中关于y_{i-1}的具体参数。	数组格式，长度（length）的取值范围为2~15，且数组中元素的和为1。例如array[0.2, 0.4, 0.3, 0.1]。
samplePeriod	对当前时序数据进行采样的周期。	long类型，单位为秒。取值范围为1~86399。
sampleMethod	针对采样窗口内数据的采样方法。	取值包括： · avg：表示取窗口内数据的平均值。 · max：表示取窗口内数据的最大值。 · min：表示取窗口内数据的最小值。 · sum：表示取窗口内数据的总和。

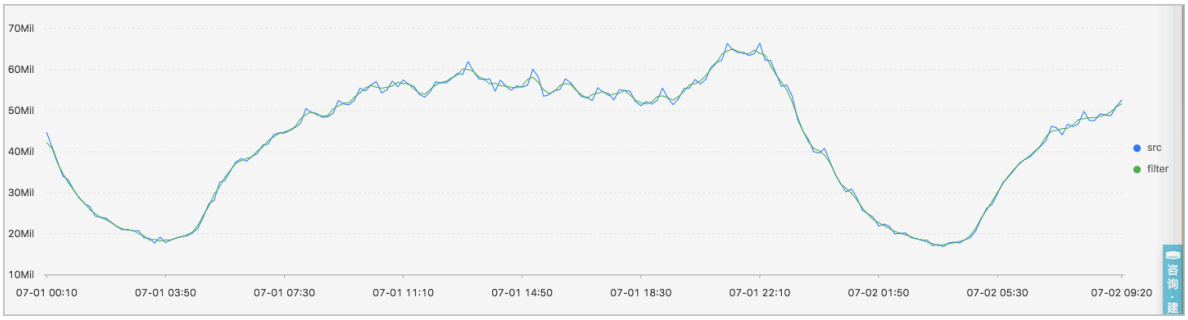
示例：

· **查询分析：**

```
* | select ts_smooth_iir(stamp, value, array[0.2, 0.4, 0.3, 0.1],
array[0.4, 0.3, 0.3], 1, 'avg') from ( select __time__ - __time__ %
```

```
120 as stamp, avg(v) as value from log GROUP BY stamp order by stamp
 )
```

- 输出结果：



显示项如下：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
纵轴	src	未滤波前的数据。
	filter	滤波之后的数据。

9.3 多周期估计函数

多周期估计函数支持对不同时间段内的时序进行周期估计，利用傅立叶变换等一系列操作进行周期的提取。

ts_period_detect

该函数用于对时序数据进行分段周期估计。

函数格式：

```
select
 ts_period_detect(x, y,minPeriod,maxPeriod,samplePeriod,sampleMethod)
```

参数说明如下：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
minPeriod	预估计周期最小长度占序列总长度的比例。	小数形式，取值范围为(0,1]。

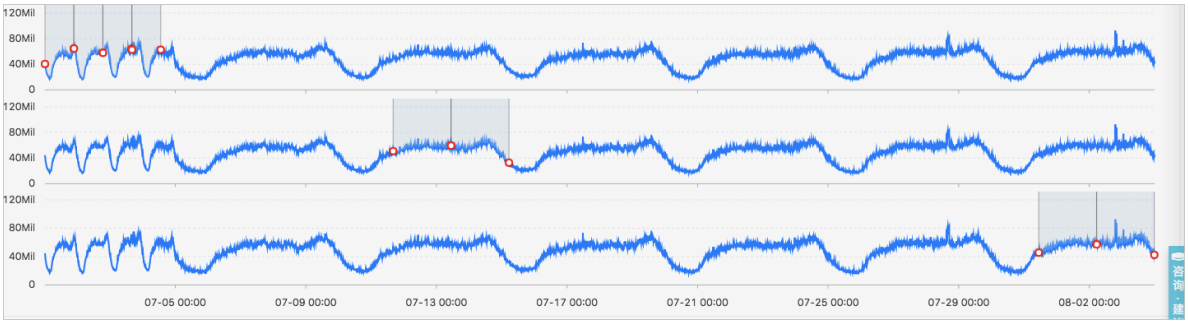
参数	说明	取值
maxPeriod	预估计周期最大长度占序列总长度的比例。 说明: 指定参数时，*maxPeriod*必须大于*minPeriod*。	小数形式，取值范围为(0,1]。
samplePeriod	对当前时序数据进行采样的周期。	long类型，单位为秒。取值范围为1~86399。
sampleMethod	针对采样窗口内数据的采样方法。	取值包括： · avg：表示取窗口内数据的平均值。 · max：表示取窗口内数据的最大值。 · min：表示取窗口内数据的最小值。 · sum：表示取窗口内数据的总和。

示例：

· 查询分析：

```
* | select ts_period_detect(stamp, value, 0.2, 1.0, 1, 'avg') from
 ( select __time__ - __time__ % 120 as stamp, avg(v) as value from
log GROUP BY stamp order by stamp )
```

· 输出结果：



显示项如下：

显示项	说明
period_id	周期编号的数组，长度为1，其中值为0.0时，表示原始序列。
time_series	表示时间戳序列。

显示项	说明
data_series	表示每个时间戳对应的结果。 · 当period_id为0.0时，表示原始序列结果。 · 当period_id不为0.0时，表示滤波之后的周期估计结果。

ts_period_classify

函数格式：

```
select ts_period_classify(stamp, value,instanceName)
```

参数说明如下：

参数	说明	取值
stamp	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
value	数值列，对应某时刻的数据。	-
instanceName	曲线对应的名称。	-

示例：

· **查询分析：**

```
* and h : nu2h05202.nu8 | select ts_period_classify(stamp, value,
name) from log
```

· **输出结果：**

显示项如下：

显示项	说明
line_name	周期编号的数组，长度为1，其中值为0.0时，表示原始序列。
prob	时序曲线中主周期的的占比[0, 1]，实验中可以取0.15。

显示项	说明
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：表示曲线是有明显周期。

9.4 变点检测函数

变点检测函数一般用于对时序数据中的变点检测。

变点检测函数支持两种变点形态：

· 在指定时间段内的某些统计特性发生了变化。
· 序列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，可以使用以下格式的ts_cp_detect函数，该函数调用的算法会默认使用长度等于10的窗口进行检测：

```
select ts_cp_detect(x, y, amplePeriod,sampleMethod)
```

· 若您需要根据业务曲线进行效果调试，可以使用以下格式的ts_cp_detect函数，通过设置参数minSize进行效果调优：

```
select ts_cp_detect(x, y, minSize, samplePeriod, sampleMethod)
```

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。

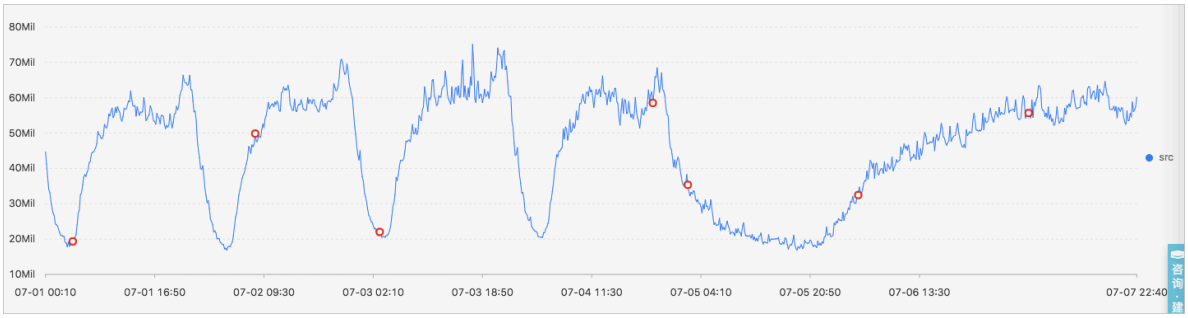
参数	说明	取值
y	数值列，对应某时刻的数据。	-
minSize	最小连续区间长度。	最小值为3，最大值不超过当前输入数据长度的1/10。
samplePeriod	对当前时序数据进行采样的周期。	long类型，取值范围为1~86399。
sampleMethod	针对采样窗口内数据的采样方法。	取值包括： · avg：表示取窗口内数据的平均值。 · max：表示取窗口内数据的最大值。 · min：表示取窗口内数据的最小值。 · sum：表示取窗口内数据的总和。

示例：

· **查询分析：**

```
* | select ts_cp_detect(stamp, value, 3, 1, 'avg') from (select
__time__ - __time__ % 10 as stamp, avg(v) as value from log GROUP BY
 stamp order by stamp)
```

· **输出结果：**



显示项如下：

显示项		说明
横轴	unixtime	数据的时间戳，单位为秒，例如1537071480。
纵轴	src	未滤波前的数据，例如1956092.7647745228。
	prob	该点为变点的概率，值的范围为0~1。

ts_breakout_detect

函数格式：

```
select ts_breakout_detect(x, y, winSize,  samplePeriod, sampleMethod)
```

参数说明如下：

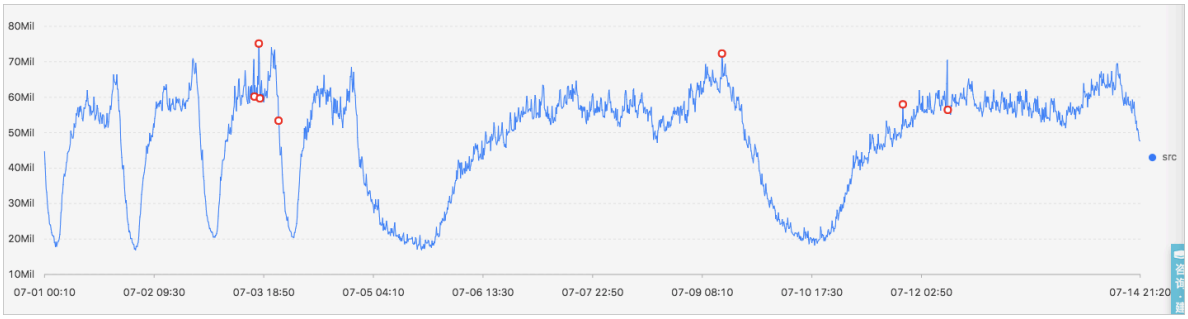
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
winSize	最小连续区间长度。	最小值为3，最大值不超过当前输入数据长度的1/10。
samplePeriod	对当前时序数据进行采样的周期。	long类型，单位为秒。取值范围为1~86399。
sampleMethod	针对采样窗口内数据的采样方法。	取值包括： · avg：表示取窗口内数据的平均值。 · max：表示取窗口内数据的最大值。 · min：表示取窗口内数据的最小值。 · sum：表示取窗口内数据的总和。

示例：

· **查询分析：**

```
* | select ts_breakout_detect(stamp, value, 3, 1, 'avg') from (
select __time__ - __time__ % 10 as stamp, avg(v) as value from log
GROUP BY stamp order by stamp)
```

· **输出结果：**



显示项如下：

显示项		说明
横轴	unixtime	数据的时间戳，单位为秒，例如1537071480。
纵轴	src	未滤波前的数据，例如1956092.7647745228。
	prob	该点为变点的概率，值的范围为0~1。

9.5 极大值检测函数

极大值检测函数用于在指定窗口中寻找序列的局部极大值。

函数格式：

```
select ts_find_peaks(x, y, winSize, samplePeriod, sampleMethod)
```

参数说明如下：

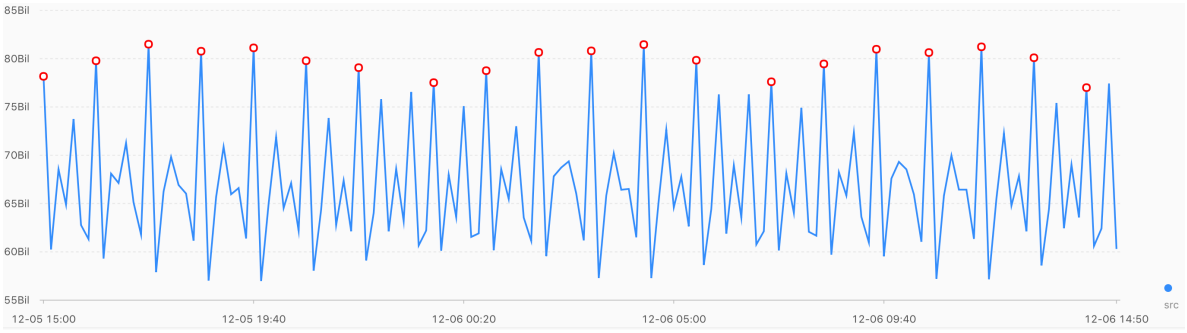
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
winSize	指定最小的检测窗口长度。	long类型，取值范围为1~数据的实际长度。建议指定参数*winSize*的值为数据实际长度的十分之一。
samplePeriod	对当前时序数据进行采样的周期。	long类型，取值范围为1~86399。
sampleMethod	针对采样窗口内数据的采样方法。	取值包括： · avg：表示取窗口内数据的平均值。 · max：表示取窗口内数据的最大值。 · min：表示取窗口内数据的最小值。 · sum：表示取窗口内数据的总和。

示例：

- 查询分析：

```
* and h : nu2h05202.nu8 and m: NET |  select ts_find_peaks(stamp,
value, 30, 1, 'avg') 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。

9.6 预测与异常检测函数

预测与异常检测函数通过预测时序曲线、寻找预测曲线和实际曲线之间误差的Ksigma与分位数等特性进行异常检测。

- 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。

函数	说明
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。 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。

说明:

预测与异常检测系列函数的显示项相同，输出结果和对应说明请查看ts_predicate_simple输出结果示例。

ts_predicate_simple

函数格式：

```
select
 ts_predicate_simple(x, y, nPred, isSmooth, samplePeriod, sampleMethod)
```

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1~5**p*。
isSmooth	是否需要将原始数据做滤波操作。 不指定该参数时，默认为true，即将原始数据做滤波操作。	bool类型，取值包括： · true：表示将原始数据做滤波操作。 · false：表示对原始数据不做滤波操作。 默认为true。

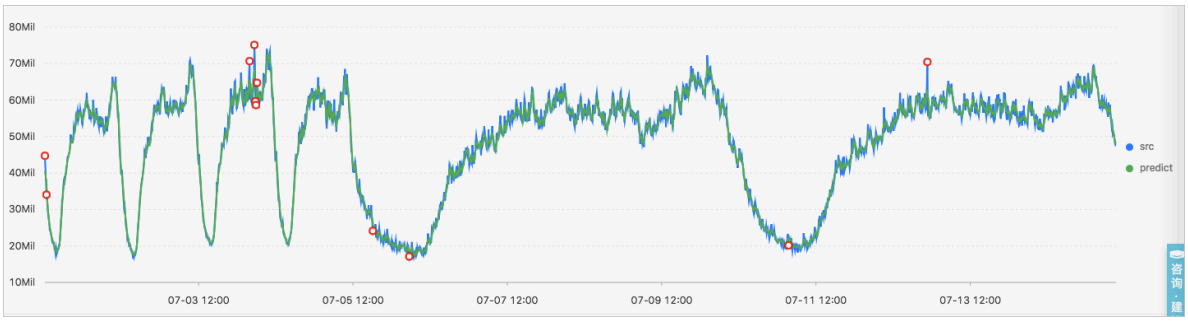
参数	说明	取值
samplePeriod	对当前时序数据进行采样的周期。	long类型，取值范围为1~86399。
sampleMethod	针对采样窗口内数据的采样方法。	取值包括： · avg：表示取窗口内数据的平均值。 · max：表示取窗口内数据的最大值。 · min：表示取窗口内数据的最小值。 · sum：表示取窗口内数据的总和。

示例：

· **查询分析：**

```
* | select ts_predicate_simple(stamp, value, 6, 1, 'avg') from (
select __time__ - __time__ % 60 as stamp, avg(v) as value from log
GROUP BY stamp order by stamp)
```

· **输出结果：**



显示项如下：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
纵轴	src	原始数据。
	predict	滤波之后的数据。
	upper	预测的上界。当前置信度为默认为0.85，且不可修改。
	lower	预测的下界。当前置信度为默认为0.85，且不可修改。
	anomaly_prob	该点是否为异常点的概率，值的范围为0~1。

ts_predicate_ar

函数格式：

```
select
 ts_predicate_ar(x, y, p, nPred, isSmooth, samplePeriod, sampleMethod)
```

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	是否需要将原始数据做滤波操作。 不指定该参数时，默认为true，即将原始数据做滤波操作。	bool类型，取值包括： · true：表示将原始数据做滤波操作。 · false：表示对原始数据不做滤波操作。 默认为true。
samplePeriod	对当前时序数据进行采样的周期。	long类型，取值范围为1~86399。
sampleMethod	针对采样窗口内数据的采样方法。	取值包括： · avg：表示取窗口内数据的平均值。 · max：表示取窗口内数据的最大值。 · min：表示取窗口内数据的最小值。 · sum：表示取窗口内数据的总和。

查询分析示例：

```
* | select ts_predicate_ar(stamp, value, 3, 4, 1, 'avg') from (select
__time__ - __time__ % 60 as stamp, avg(v) as value from log GROUP BY
stamp order by stamp)
```

ts_predicate_arma

函数格式：

```
select
 ts_predicate_arma(x, y, p, q, nPred, isSmooth, samplePeriod, sampleMethod)
```

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	是否需要将原始数据做滤波操作。 不指定该参数时，默认为true，即将原始数据做滤波操作。	bool类型，取值包括： · true：表示将原始数据做滤波操作。 · false：表示对原始数据不做滤波操作。 默认为true。
samplePeriod	对当前时序数据进行采样的周期。	long类型，取值范围为1~86399。
sampleMethod	针对采样窗口内数据的采样方法。	取值包括： · avg：表示取窗口内数据的平均值。 · max：表示取窗口内数据的最大值。 · min：表示取窗口内数据的最小值。 · sum：表示取窗口内数据的总和。

查询分析示例：

```
* | select ts_predicate_arma(stamp, value, 3, 2, 4, 1, 'avg') from (
select __time__ - __time__ % 60 as stamp, avg(v) as value from log
GROUP BY stamp order by stamp)
```

ts_predicate_arima

函数格式：

```
select
 ts_predicate_arima(x, y, p, d,  qnPred, isSmooth, samplePeriod, sampleMethod)
```

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	是否需要将原始数据做滤波操作。 不指定该参数时，默认为true，即将原始数据做滤波操作。	bool类型，取值包括： · true：表示将原始数据做滤波操作。 · false：表示对原始数据不做滤波操作。 默认为true。
samplePeriod	对当前时序数据进行采样的周期。	long类型，取值范围为1~86399。
sampleMethod	针对采样窗口内数据的采样方法。	取值包括： · avg：表示取窗口内数据的平均值。 · max：表示取窗口内数据的最大值。 · min：表示取窗口内数据的最小值。 · sum：表示取窗口内数据的总和。

查询分析示例：

```
* | select ts_predicate_arima(stamp, value, 3, 1, 2, 4, 1, 'avg') from
 (select __time__ - __time__ % 60 as stamp, avg(v) as value from log
GROUP BY stamp order by stamp)
```

ts_regression_predict

函数格式：

```
select
 ts_regression_predict(x, y, nPred, algo_type,processType, samplePeriod, sampleMe
```

参数说明如下：

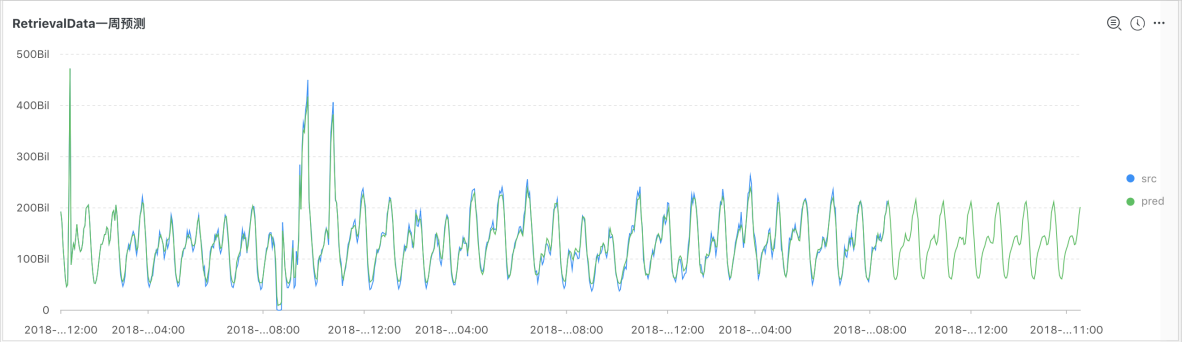
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
nPred	预测未来的点的数量。	long类型，取值范围为1~500。
algo_type	针对的预测的算法类型。	取值包括： · origin：使用GBRT（Gradient Boosted Regression Tree）算法进行预测。 · forest：使用STL序列分解的结果，将分解得到的趋势序列使用GBRT算法进行预测，再将分解出来的序列按照加法模型进行求和后返回。 · linear：使用STL序列分解的结果，将分解得到趋势序列使用Linear Regression算法进行预测，再将分解出来的序列按照加法模型进行求和后返回。
processType	数据对应的预处理流程选择。	· processType = 0：此时并不进行任何的额外数据预处理。 · processType = 1：此时进行去除异常后再进行预测处理。
samplePeriod	对当前时序数据进行采样的周期。	long类型，取值范围为1~86399。
sampleMethod	针对采样窗口内数据的采样方法。	取值包括： · avg：表示取窗口内数据的平均值。 · max：表示取窗口内数据的最大值。 · min：表示取窗口内数据的最小值。 · sum：表示取窗口内数据的总和。

示例：

- **查询分析：**

```
* and h : nu2h05202.nu8 and m: NET |  select ts_regression_predic
t(stamp, value, 200, 'origin', 1, 'avg') from (select __time__ -
__time__ % 60 as stamp, avg(v) as value from log GROUP  BY  stamp
order by stamp)
```

- **输出结果：**



显示项如下：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
纵轴	src	原始数据。
	predict	预测数据。

ts_anomaly_filter

函数格式：

```
select
 ts_anomaly_filter(lineName, ts, ds, preds, probs, nWatch, anomalyType)
```

参数说明如下：

参数	说明	取值
lineName	varchar类型，表示每条曲线的名称。	-
ts	曲线的时间序列，array（double）类型，表示当前这条曲线的时间信息，由小到大排列。	-

参数	说明	取值
ds	曲线的实际值序列，array（double）类型，表示当前这条曲线的数值信息，长度与ts相同。	-
preds	曲线的预测值序列，array（double）类型，表示当前这条曲线的预测值，长度与ts相同。	-
probs	曲线的异常检测序列，array（double），表示当前这条曲线的异常检测结果，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。	· 0：表示关注全部异常。 · 1：表示关注上升沿异常。 · -1：表示下降沿异常。

示例：

· **查询分析：**

```
* | select res.name, res.ts, res.ds, res.preds, res.probs
     from (
         select ts_anomaly_filter(name, ts, ds, preds, probs, cast(5
 as bigint), cast(1 as bigint)) as res
       from (
         select name, res[1] as ts, res[2] as ds, res[3] as preds,
res[4] as uppers, res[5] as lowers, res[6] as probs
     from (
         select name, array_transpose(ts_predicate_ar(stamp, value,
10)) as res
         from (
           select name, stamp, value from log where name like '%asg-
%') group by name)) );
```

· **输出结果：**

```
| name                      | ts
              | ds           | preds     | probs       |
| ------------------------ |
 ---------------------------------------------------- | -----------
 | --------- | ----------- |
```

```
| asg-bp1hylzdi2wx7civ0ivk | [1.5513696E9, 1.5513732E9, 1.5513768E9
, 1.5513804E9] | [1,2,3,NaN] | [1,2,3,4] | [0,0,1,NaN] |
```

9.7 序列分解函数

序列分解函数提供针对业务曲线的分解功能，突出曲线的趋势信息和周期信息。

ts_decompose

函数格式：

```
select ts_decompose(x, y, samplePeriod, sampleMethod)
```

参数说明如下：

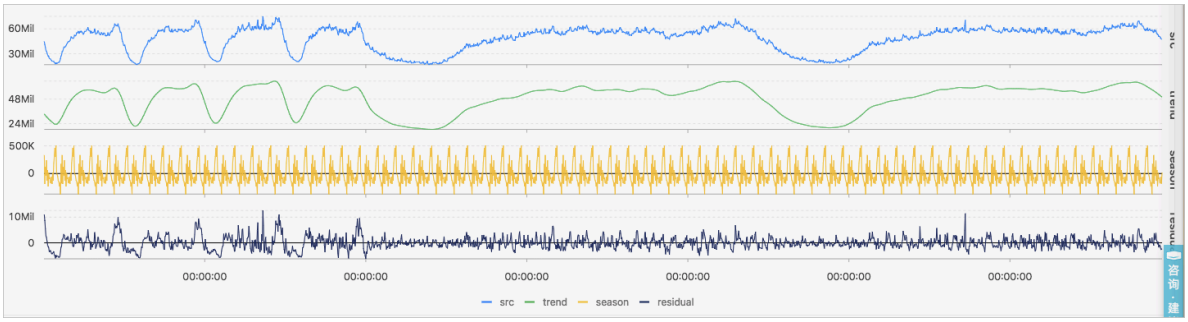
参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
samplePeriod	对当前时序数据进行采样的周期。	long类型，取值范围为1~86399。
sampleMethod	针对采样窗口内数据的采样方法。	取值包括： · avg：表示取窗口内数据的平均值。 · max：表示取窗口内数据的最大值。 · min：表示取窗口内数据的最小值。 · sum：表示取窗口内数据的总和。

示例：

- 查询分析：

```
* | select ts_decompose(stamp, value, 1, 'avg') from (select
__time__ - __time__ % 60 as stamp, avg(v) as value from log GROUP BY
 stamp order by stamp)
```

- 输出结果：



显示项如下：

显示项		说明
横轴	unixtime	数据的Unixtime时间戳，单位为秒。
纵轴	src	原始数据。
	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。

9.8 时序聚类函数

时序聚类函数针对输入的多条时序数据进行聚类，自动聚类出不同的曲线形态，进而快速找到相应的聚类中心和异于聚类中的其它形态曲线。

LOG机器学习介绍（02）：时序聚类建模

函数列表

函数	说明
ts_density_cluster	使用密度聚类方法对多条时序数据进行聚类。
ts_hierarchical_cluster	使用层次聚类方法对多条时序数据进行聚类。
ts_similar_instance	查找到指定曲线名称的相似曲线。

ts_density_cluster

函数格式：

```
select ts_density_cluster(x, y, z)
```

参数说明如下：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
z	每个时刻数据对应的指标名称。	字符串类型，例如machine01.cpu_usr。

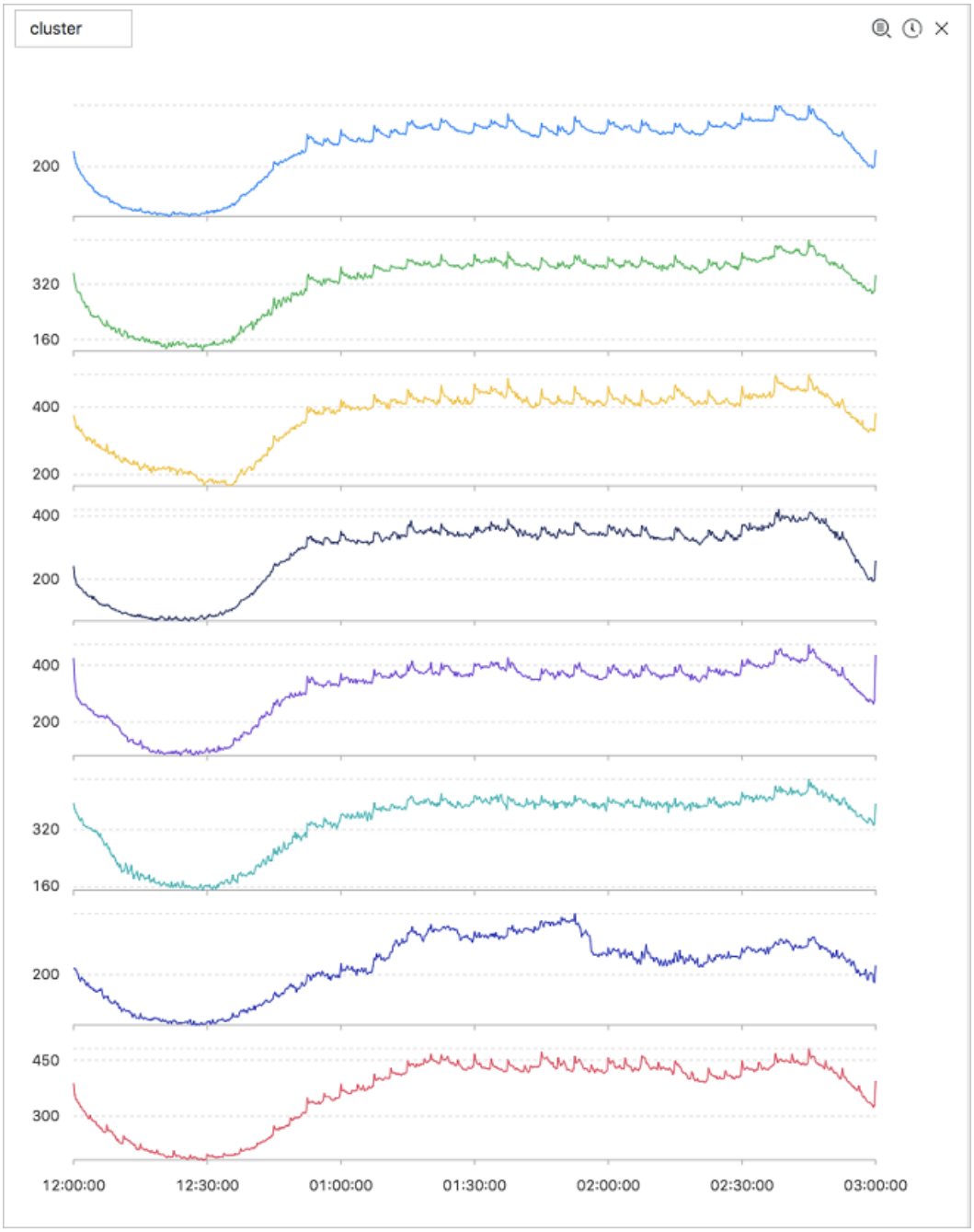
示例：

- **查询分析：**

```
* and (h: "machine_01" OR h: "machine_02" OR h : "machine_03") |
select ts_density_cluster(stamp, metric_value,metric_name ) from (
select __time__ - __time__ % 600 as stamp, avg(v) as metric_value
```

```
, h as metric_name from log GROUP BY stamp, metric_name order BY
metric_name, stamp )
```

- 输出结果：



显示项如下：

显示项	说明
cluster_id	聚类的类别，其中-1表示未能划分到某一聚类中心。
rate	该聚类中的instance占比。
time_series	该聚类中心的时间戳序列。
data_series	该聚类中心的数据序列。

显示项	说明
instance_names	该聚类中心包含的instance的集合。
sim_instance	该类中的某一个instance名称。

ts_hierarchical_cluster

函数格式：

```
select ts_hierarchical_cluster(x, y, z)
```

参数说明如下：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
z	每个时刻数据对应的指标名称。	字符串类型，例如machine01.cpu_usr。

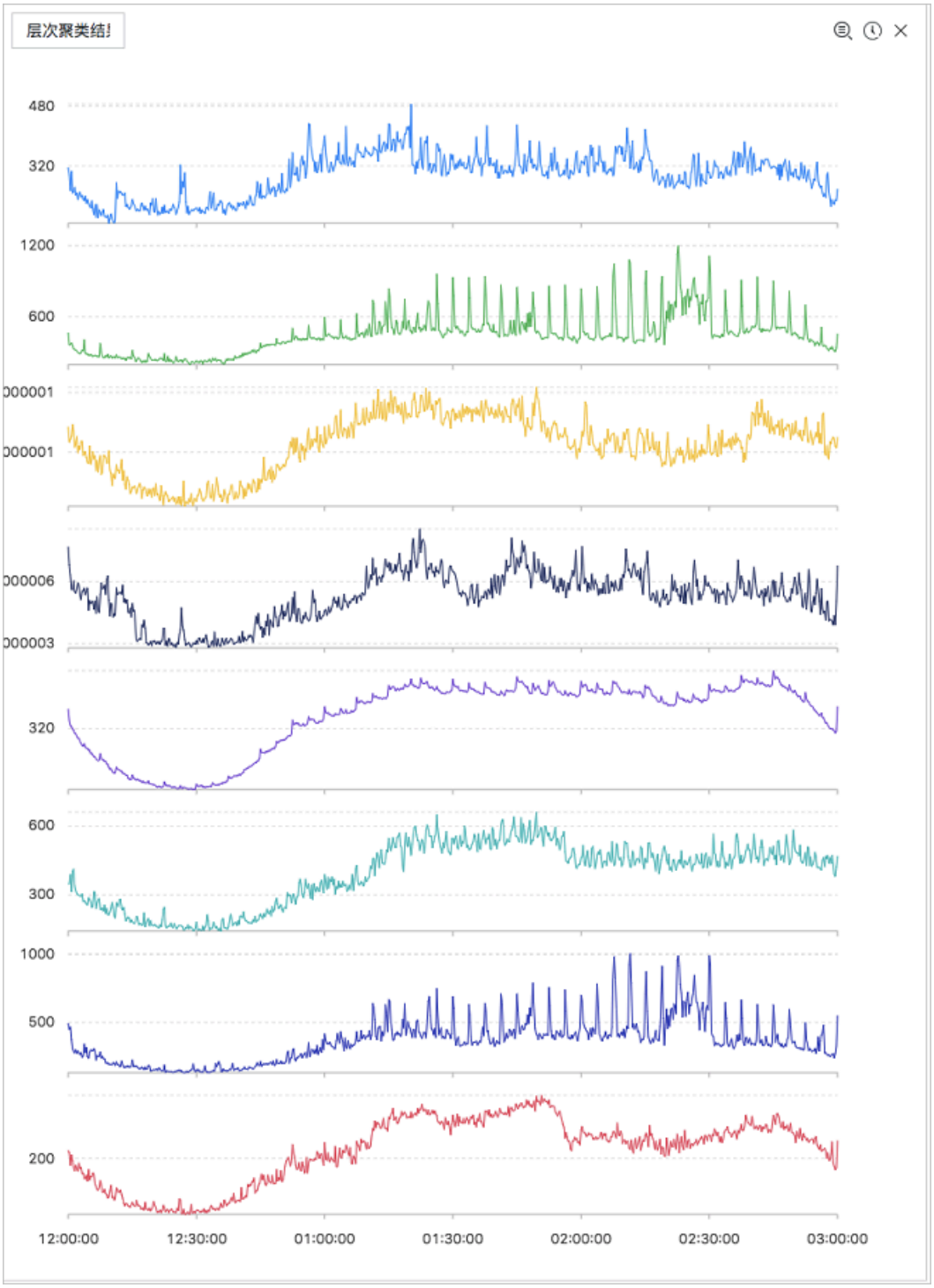
示例：

- **查询分析：**

```
* and (h: "machine_01" OR h: "machine_02" OR h : "machine_03") |
select ts_hierarchical_cluster(stamp, metric_value, metric_name
) from ( select __time__ - __time__ % 600 as stamp, avg(v) as
```

```
metric_value, h as metric_name from log GROUP BY stamp, metric_name
order BY metric_name, stamp )
```

- 输出结果：



显示项如下：

显示项	说明
cluster_id	聚类的类别，其中-1表示未能划分到某一聚类中心。
rate	该聚类中的instance占比。

显示项	说明
time_series	该聚类中心的时间戳序列。
data_series	该聚类中心的数据序列。
instance_names	该聚类中心包含的instance的集合。
sim_instance	该类中的某一个instance名称。

ts_similar_instance

函数格式：

```
select ts_similar_instance(x, y, z, instance_name, topK, metricType)
```

参数说明如下：

参数	说明	取值
x	时间列，从小到大排列。	格式为Unixtime时间戳，单位为秒。
y	数值列，对应某时刻的数据。	-
z	每个时刻数据对应的指标名称。	字符串类型，例如machine01.cpu_usr。
instance_name	指定某个待查找的指标的名字。	集合中某个指标名称，字符串类型，如：machine01.cpu_usr。 说明： 必须是已创建的指标。
topK	最多返回K个与给定相似曲线。	-
metricType	{'shape', 'manhattan', 'euclidean'}，衡量时序曲线之间的相似性指标。	-

查询分析示例：

```
* and m: NET and m: Tcp and (h: "nu4e01524.nu8" OR  h: "nu2i10267.
nu8" OR  h : "nu4q10466.nu8") | select ts_similar_instance(stamp,
metric_value, metric_name, 'nu4e01524.nu8' ) from ( select __time__ -
__time__ % 600 as stamp, sum(v) as metric_value, h as metric_name from
 log GROUP BY stamp, metric_name order BY  metric_name, stamp )
```

显示项如下：

显示项	说明
instance_name	与指定指标相近的结果列表。
time_series	该聚类中心的时间戳序列。

显示项	说明
data_series	该聚类中心的数据序列。

9.9 频繁模式统计函数

频繁模式统计函数可以在给定的多属性字段样本中，挖掘出具有一定代表性的属性组合，用来归纳当前日志。

pattern_stat

函数格式：

```
select pattern_stat(array[col1, col2, col3], array['col1_name',
 'col2_name', 'col3_name'], array[col5, col6], array['col5_name',
 'col6_name'], supportScore, 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']。
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]。
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]。

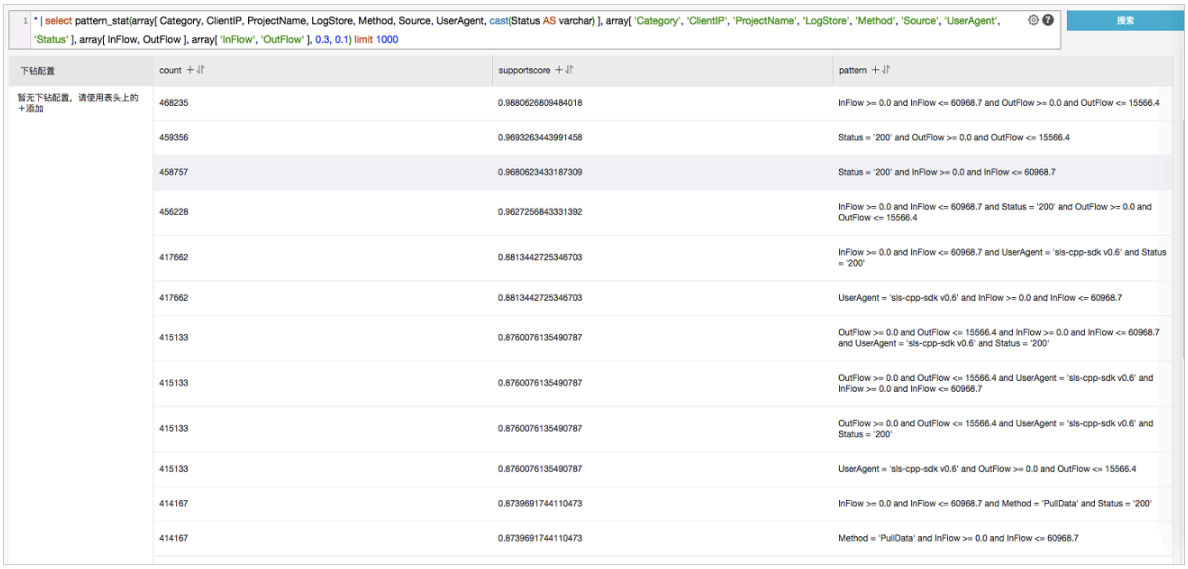
示例：

- 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。
supportScore	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。

9.10 差异模式统计函数

差异模式统计函数基于给定的多属性字段样本，在给定的判别条件下，分析出影响该条件划分的差异化模式集合，帮助您快速诊断导致当前判别条件差异的原因。

pattern_diff

函数格式：

```
select
 pattern_diff(array_char_value, array_char_name, array_numeric_value, array_numer
 )
```

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]。

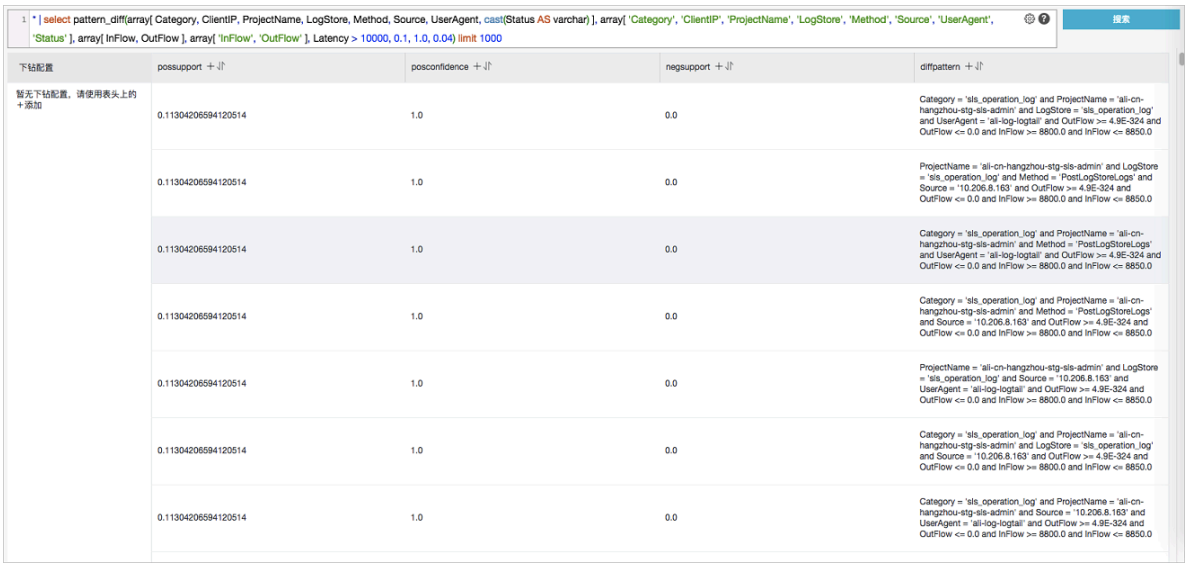
参数	说明	取值
array_char_name	字符型数据的输入列的对应名称。	数组形式，例如：array['clientIP', 'sourceIP', 'path', 'logstore']。
array_numeric_v	数值型数据的输入列。	数组形式，例如：array[Inflow, OutFlow]。
array_numeric_n	数值型数据的输入列的对应名称。	数组形式，例如array['Inflow', 'OutFlow']。
condition	筛选数据的条件。条件为True则为正样本，条件为False则为负样本。	例如：Latency <= 300。
supportScore	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。

9.11 根因分析函数

日志服务提供了强大的告警和分析能力，可以帮助用户快速分析和定位到发生异常的具体的子维度。在时序指标发生异常时，根因分析函数可以快速分析出是哪些相关维度属性发生异常而导致监控指标发生异常。

LOG机器学习最佳实战：根因分析（一）

rca_kpi_search

函数格式

```
select rca_kpi_search(varchar_array, name_array, real, forecast, level
)
```

参数说明如下：

参数	说明	取值
varchar_array	属性维度字段。	数组形式，例如：array[col1, col2, col3]。
name_array	属性名字字段。	数组形式，例如：array['col1', 'col2', 'col3']。
real	varchar_array对应的实际值。	double 类型，取值范围：全体实数。
forecast	varchar_array对应的预测值。	double 类型，取值范围：全体实数。
level	输出的根因集合对应的维度属性的数量，其中level=0表示输出找到的全部根因集合。	long类型，取值范围：0<=level<=分析维度数（对应varchar_array的长度）。

示例：

- **查询分析：**

  先利用子查询去组织每个细粒度属性对应的实际值和预测值，然后直接调用rca_kpi_search函数去分析异常时刻的根因。

```
* not Status:200 |
select rca_kpi_search(
 array[ ProjectName, LogStore, UserAgent, Method ],
 array[ 'ProjectName', 'LogStore', 'UserAgent', 'Method' ], real,
forecast, 1)
from (
select ProjectName, LogStore, UserAgent, Method,
 sum(case when time < 1552436040 then real else 0 end) * 1.0 / sum(
case when time < 1552436040
then 1 else 0 end) as forecast,
 sum(case when time >=1552436040 then real else 0 end) *1.0 / sum(
case when time >= 1552436040
then 1 else 0 end) as real
 from (
select __time__ - __time__ % 60 as time, ProjectName, LogStore,
UserAgent, Method, COUNT(*) as real
from log GROUP by time, ProjectName, LogStore, UserAgent, Method )
GROUP BY ProjectName, LogStore, UserAgent, Method limit 100000000)
```

- **输出结果：**

返回结果结构说明：

```
{
  "rcSets": [
    {
      "rcItems": [
        {
          "kpi": [("attr": "xxx", "val": "xxx")],
          "nleaf": 100,
          "change": 0.524543,
          "score": 0.1454543
        }
      ]
    }
  ]
}
```

显示项如下：

显示项	说明
rcSets	根因集合，value对应一个数组。
rcItems	具体对应一个根因集合。
kpi	根因集合中的一项，数据按照数组形式存储，数组中的每一项是一个json类型的数据，attr表示维度名称，val表示当前维度下对应的属性名称。
nleaf	根因集合中某一项（KPI）在原始数据中覆盖的叶子节点数。 **说明:** 叶子节点：表示最细粒度属性组合的日志。
change	根因集合中某一项（KPI）对应的叶子节点集合的异常变化量占同一时刻总体异常变化量的比例。
score	当前kpi对应的异常程度（0 <= score <= 1）。

输出结果是一个Json，具体格式如下：

```
{
  "rcSets": [
  {
    "rcItems": [
    {
      "kpi": [
      {
        "attr": "country",
        "val": "*"
      },
      {
        "attr": "province",
```

```
        "val": "*"
      },
      {
        "attr": "provider",
        "val": "*"
      },
      {
        "attr": "domain",
        "val": "download.huya.com"
      },
      {
        "attr": "method",
        "val": "*"
      }
      ],
      "nleaf": 119,
      "change": 0.3180687806279939,
      "score": 0.14436007709620113
    }
    ]
  }
  ]
}
```


9.12 相关性分析函数

针对系统中的多个观测指标，可以快速找出与某个指标项相关或者时序序列相关的指标名称。

函数列表

函数	说明
ts_association_analysis	针对系统中的多个观测指标，快速找出和某个指标项相关的指标名称。
ts_similar	针对系统中的多个观测指标，快速找出和用户输入的时序序列相关的指标名称。

ts_association_analysis

函数格式：

```
select
 ts_association_analysis(stamp, params, names, indexName, threshold)
```

参数说明如下：

参数	说明	取值
stamp	long 类型，表示UnixTime时间戳。	-
params	array（double）类型，表示待分析的指标维度。	例如：Latency，QPS，NetFlow等。

参数	说明	取值
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。	-

参数	说明	取值
ts	array（double）类型，表示指定曲线的时间序列信息。	-
ds	array（double）类型，表示指定曲线的数值序列信息。	-
metricType	varchar 类型，表示度量曲线间相关性的类型。	类型如下： SHAPE，RMSE，PEARSON， SPEARMAN，R2，KENDALL

函数格式二：

```
select ts_similar(stamp, value, startStamp, endStamp, step, ds)
select
 ts_similar(stamp, value, startStamp, endStamp, step, ds, metricType )
```

参数说明二：

参数	说明	取值
stamp	long 类型，表示UnixTime时间戳。	-
value	double 类型，表示某指标对应的值。	-
startStamp	long 类型，表示指定曲线的开始时间戳。	-
endStamp	long 类型，表示指定曲线的结束时间戳。	-
step	long类型，表示时序中相邻两个点之间的时间间隔。	-
ds	array（double）类型，表示指定曲线的数值序列信息。	-
metricType	varchar 类型，表示度量曲线间相关性的类型。	类型如下： SHAPE，RMSE，PEARSON， SPEARMAN，R2，KENDALL

输出结果：

- score：该指标与目标指标之间的相关性值，范围在[-1, 1]之间。

代码示例：

```
* | select vhost, metric, ts_similar(time, value, 1560911040,
1560911065, 5, array[5.1,4.0,3.3,5.6,4.0,7.2], 'PEARSON') from log
group by vhost, metric;
```

结果示例：

```
| vhost  | metric          | score                |
| ------ | --------------- | -------------------- |
| vhost1 | redolog         | -0.3519082537204182  |
| vhost1 | kv_qps          | -0.15922168009772697 |
| vhost1 | file_meta_write | NaN                  |
```

10 分析进阶

10.1 优秀分析案例

案例列表

1. 5分钟错误率超过40%时触发报警
2. 当流量暴跌时，触发报警
3. 按照数据区间分桶，在每个桶内计算平均延时
4. 在group by的结果中，返回百分比
5. 统计满足条件的个数

5分钟错误率超过40%时触发报警

统计每分钟的500错误率，当最近5分钟错误率超过40%时触发报警。

```
status:500 | select __topic__, max_by(error_count,window_time)/1.0/sum
(error_count) as error_ratio, sum(error_count) as total_error  from (
select __topic__, count(*) as error_count , __time__ - __time__ % 300
  as window_time  from log group by __topic__, window_time
)
group by __topic__  having  max_by(error_count,window_time)/1.0/sum(
error_count)   > 0.4  and sum(error_count) > 500 order by total_error
desc limit 100
```

当流量暴跌时，触发报警

统计每分钟的流量，当最近的流量出现暴跌时，触发报警。 由于在最近的一分钟内，统计的数据不是一个完整分钟的，所以，需要除以(max(time) - min(time)) 进行归一化，统计每个分钟内的流量均值。

```
* | SELECT SUM(inflow) / (max(__time__) - min(__time__))  as
inflow_per_minute, date_trunc('minute',__time__)  as minute group by
minute
```

按照数据区间分桶，在每个桶内计算平均延时

```
* | select avg(latency) as latency , case when originSize < 5000 then
 's1' when originSize < 20000 then 's2' when originSize < 500000 then
```

```
 's3' when originSize < 100000000 then 's4' else 's5' end as os group
by os
```

在group by的结果中，返回百分比

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 groupby 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
```

10.2 优化查询

介绍优化查询的方法，可以帮助用户提高查询效率。主要包括以下方法：

- 增加更多shard。
- 缩短时间范围和数据量。
- 多次重复查询。
- 优化查询的SQL。

增加更多shard

shard代表的是计算资源，shard越多，计算越快。保证平均每个shard扫描的数据不多于5000万条日志。 增加shard可以通过分裂shard完成。

说明：

分裂shard后，会产生更多费用，且只对新数据起到加速效果，旧数据仍然在旧的shard上。

缩减查询的时间范围和数据量

- 时间范围越大，查询越慢。如果您查询1年的时间，或者查询1个月的时间。 计算是按天完成的，所以，适当的缩短时间可以更快完成计算。
- 数据量越大，查询越慢。请尽量减少查询的数据量。

多次重复查询

当查询不精确时，可以尝试多次重复查询。每次查询时，底层加速机制会充分利用已有的结果进行分析。所以，多次查询可以使结果更加精确。

优化query

计算时间较长的查询语句特点：

- 对字符串列进行group by。
- 对多列（大于5列）字段进行group by。
- 在SQL中有生成字符串的操作。

为了优化query，有以下方法：

- 尽量避免生成字符串的操作
  - 使用date_format函数生成格式化的时间戳，导致查询效率低下。

    ```
    * | select date_format(from_unixtime(__time__) , '%H_%i') as t,
    count(1) group by t
    ```

  - 使用substr生成字符串。 对于这类时间戳函数，建议使用date_trunc或者time_series函数进行分析。
- 尽量避免对字符串列进行GROUP BY计算

  对字符串进行GROUP BY，会导致大量的hash计算，这部分计算量往往会占据整体计算的50%以上。例如：

  ```
  * | select count(1) as pv , date_trunc('hour',__time__) as time
  group by time
  * | select count(1) as pv , from_unixtime(__time__-__time__%3600) as
   time group by __time__-__time__%3600
  ```

  Query 1 和Query 2都是计算每小时的日志count数，但是Query 1 首先把时间转化成字符串，例如2017-12-12 00:00:00， 然后对这个字符串进行GROUP BY。 Query 2是先对时间整点值进行计算，GROUP BY计算后才会转化成字符串类型。所以在执行效率上，Query 2更佳。
- GROUP BY多列时，把字典大的字段放在前面

  例如，province有13个，用户有1亿。

  ```
  快： * | select province,uid,count(1)groupby province,uid
  ```

```
慢: * | select province,uid,count(1)groupby uid,province
```

- 使用估算函数

  估算函数的性能要比精确计算好很多。估算会损失一些可接受的精确度，来达到快速计算的效果。

```
快:  * |select approx_distinct(ip)
慢: * | select count(distinct(ip))
```

- 在SQL中获取需要的列，尽量不要读取所有列

  获取所有列，请使用查询语法。在SQL计算时，尽量只读取需要参与计算的列，可以加快计算。

```
快 : * |select a,b c
慢 : * |select*
```

- 非group by的列，尽量放到聚合函数中

  例如，userid，用户名，必定是一一对应的，我们只需要按照userid进行group by即可。

```
快:  * | select userid, arbitrary(username), count(1)groupby userid
慢: * | select userid, username, count(1)groupby userid,username
```

- 避免使用in语法

  尽量避免在SQL中使用in语法，而应该把in语法用搜索的or语法代替。

```
快: key : a or key :b or key:c | select count(1)
慢: * | select count(1) where key in ('a','b')
```

10.3 时间字段转换示例

在查询分析中，往往需要对日志中的时间字段进行处理，例如将时间戳转换成指定格式等，本文档介绍时间字段的常用转换示例。

日志中可能有多个记录时间的字段，例如：

- `__time__`：用API/SDK写入日志数据时指定的日志时间，该字段可用于日志投递、查询、分析。
- 日志中原有的时间字段：日志在生成时，用于记录日志事件发生时间的字段，是原始日志的字段。

时间字段的格式可能不统一、或不便于查看和阅读，可以在查询分析中将其转换为指定格式。例如：

1. 把__time__转化成时间戳
2. 把__time__以固定格式打印
3. 把timestamp转化成指定格式

把__time__转化成时间戳

把字段__time__转化成时间戳格式，建议使用from_unixtime函数。

```
* | select from_unixtime(__time__)
```

把__time__以固定格式打印

把字段__time__以 年-月-日 时:分:秒的形式打印下来。建议使用date_format函数。

```
* | select date_format(__time__, '%Y-%m-%d %H:%i:%S')
```

把日志中的时间转换成指定格式

把日志中的时间字段转化成指定格式（年-月-日 时:分:秒），只取年-月-日部分，并做Group by处理。建议使用date_format函数。

- 日志样例：

```
__topic__:
body_byte_sent:  307
hostname:  www.host1.com
http_user_agent:  Mozilla/5.0 (iPhone; CPU iPhone OS 10_3_3 like Mac
 OS X) AppleWebKit/603.3.8 (KHTML, like Gecko) Mobile/14G60 QQ/7.1.
8.452 V1_IPH_SQ_7.1.8_1_APP_A Pixel/750 Core/UIWebView NetType/WIFI
QBWebViewType/1
method:  GET
referer:  www.host0.com
remote_addr:  36.63.1.23
request_length:  111
request_time:  2.705
status:  200
upstream_response_time:  0.225582883754
url:  /?k0=v9&
time:2017-05-17 09:45:00
```

- SQL语句样例：

```
* | select date_format (date_parse(time,'%Y-%m-%d %H:%i:%S'), '%Y-%m
-%d') as day, count(1) as uv group by day order by day asc
```

11 通过JDBC协议分析日志

除日志服务提供的API接口外，您还可以使用JDBC + 标准SQL 92进行日志查询与分析。

连接参数

连接参数	示例	说明
host	regionid.example.com	#unique_164，目前仅支持经典网络内网访问和VPC网络访问
port	10005	默认使用10005作为端口号
user	bq2sjzesjmo86kq	访问秘钥 AccesskeyId
password	4fdO1fTDDuZP	访问秘钥Accesskey
database	sample-project	账号下的项目（Project）
table	sample-logstore	项目下的日志库（Logstore）

例如通过MySQL命令连接示例如下：

```
mysql -hcn-shanghai-intranet.log.aliyuncs.com -ubq2sjzesjmo86kq -
p4fdO1fTDDuZP -P10005
use sample-project; // 使用某个Project
```

前提条件

- 访问JDBC接口，必须使用主账号的AK或者子帐号的AK。子帐号必须是Project owner的子帐号，同时子帐号具有Project级别的读权限。
- MySQL JDBC不支持分页。

语法说明

注意事项

在where条件中必须包含`__date__`或`__time__`来限制查询的时间范围。`__date__`是timestamp类型`__time__`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`

上述两种条件必须出现一个。

过滤语法

关于where下过滤（filter）语法如下：

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 = "valu*"`	查询的是分词之后模糊匹配的结果。
数值比较	`num_field > 1`	支持的比较运算符包括`>`、`>=`、`=`、`<`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，需要使用特殊的key（`__line__`）。

计算语法

支持计算操作符参见分析语法。

SQL92语法

过滤 + 计算组合为SQL92语法。

例如对于如下查询：

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

我们可以将查询中过滤部分+ 时间条件组合成为查询的条件，变成标准SQL92语法：

```
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

程序调用

开发者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。例如使用JDBC或者Python MySQLdb。

使用样例：

```
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
            System.out.println("Connecting to a selected database
...");
            conn = DriverManager.getConnection("jdbc:mysql://cn-
shanghai-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),histogram(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 latency < 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();
                }
            }
```

```
            }
        }
    }
```

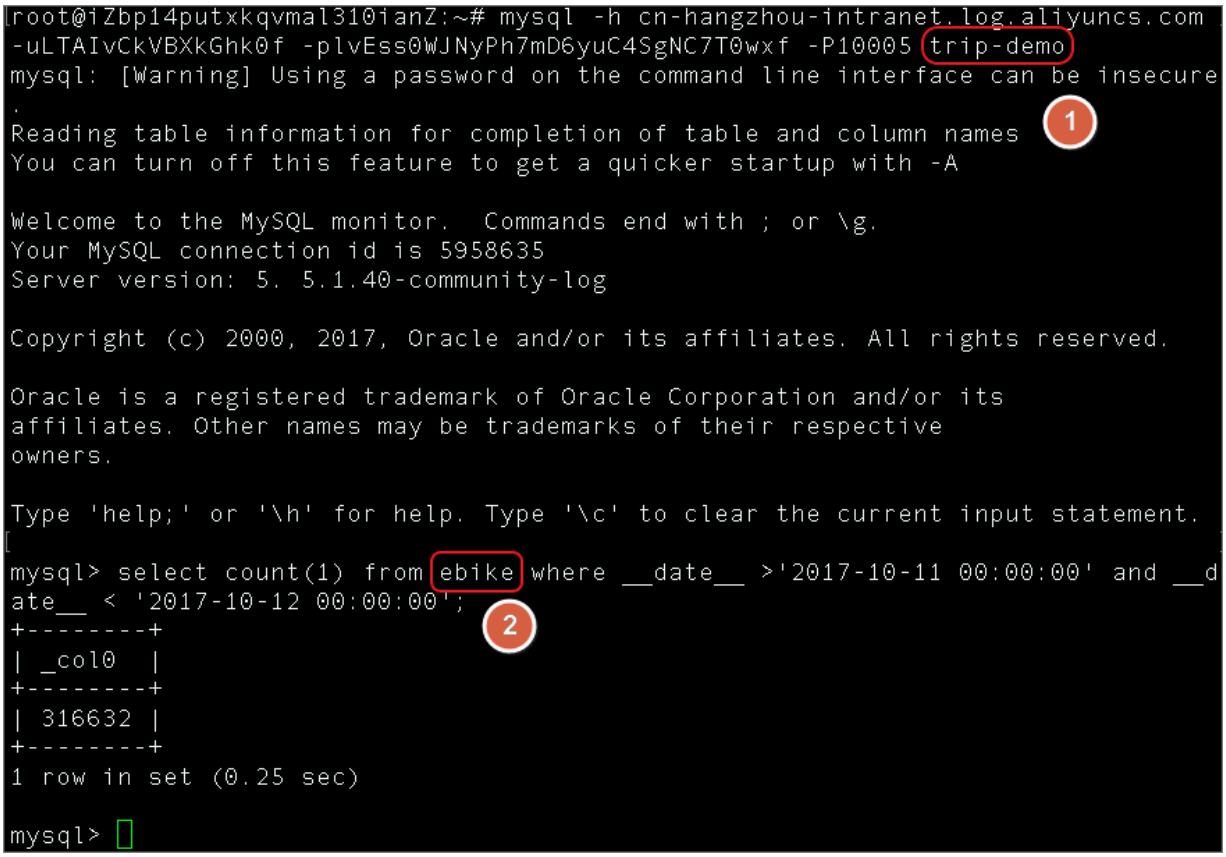
工具类调用

在经典网内网/VPC环境通过MySQL Client进行连接。

说明:

- **①处填写您的Project。**
- **②处填写您的Logstore。**

图 11-1: 连接示例



12 可视化分析

12.1 分析图表

12.1.1 图表说明

日志服务提供类似于SQL的聚合计算功能，一切通过SQL聚合计算的结果都可以通过日志服务提供的可视化图表进行渲染。

说明:

使用可视化图表前，请仔细阅读#unique_13。

前提条件

1. 已#unique_4，并开启分析功能。
2. 只有在查询中使用分析语句，才能根据统计结果为您展示图表。

注意事项

当依次执行多个查询分析语句时，系统无法自动判断您的数值列或X轴、Y轴等信息，可能会默认保留您上次查询时的属性配置，导致当前查询语句无法自动生成分析图表。如果出现以下报错时，请按照当前查询语句重新选择属性配置信息。

- 当前选择的维度不在统计的数据维度中，请检查并调整属性配置。
- 当前无X轴信息或Y轴信息，请检查并调整属性配置。

目前日志服务提供了如下图表类型：

图 12-1: 图表类型

每种图表使用方式请参考如下文档：

- #unique_172
- #unique_173
- #unique_174
- #unique_175
- #unique_176
- #unique_177
- #unique_178
- #unique_179
- #unique_180
- #unique_181
- #unique_182

图表设置

统计图表页签中展示查询分析语句的图形化分析结果，支持在图表栏中设置图表类型。

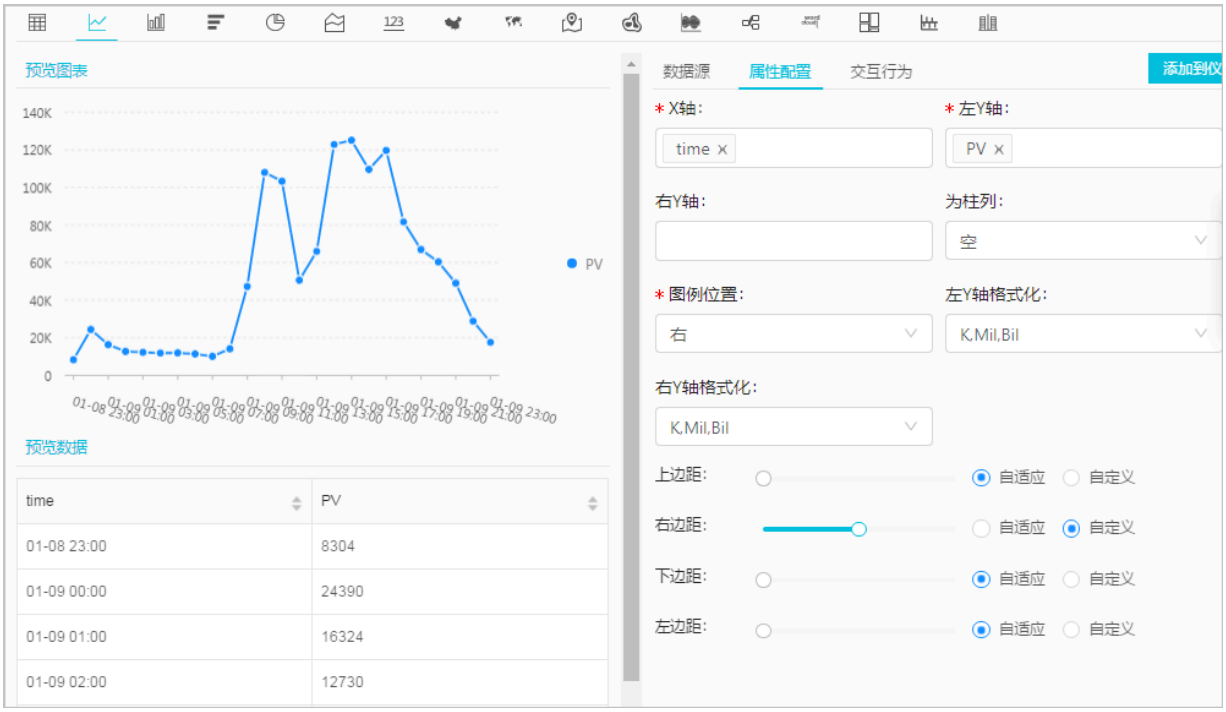
- 统计图表页签左侧为您展示当前查询分析语句的预览图表和预览数据。其中，预览图表是指定的分析图表类型，预览数据以表格形式清晰直观地展示对应的图表数据。

- 统计图表页签右侧可以进行多种图表属性设置，包括：
  - 数据源页签：用于设置占位符变量，如果有图表的下钻行为是跳转到这个图表所在的仪表盘，那么当变量名一致的情况下，会将单击触发下钻的数据替换为此处设置的占位符变量，重新执行分析。详细说明请查看#unique_57。

    适用于下钻场景中的目的仪表盘。
  - 属性配置页签：用于配置图表的显示属性，包括X轴、Y轴数据源、边距、字号等，不同的图表属性不同。详细说明请查看各个图表的文档。

    适用于所有的查询分析场景。
  - 交互行为页签：用于设置该图表的下钻动作，设置后，在仪表盘中单击该图表中的值，即可执行指定的下钻动作。详细说明请查看#unique_57。

    适用于下钻场景中的触发下钻图表。



12.1.2 表格

表格作为最常见的数据展示类型，是组织整理数据最基本的手段，通过的对数据的整理，达到快速引用和分析的目的。日志服务提供类似于SQL的聚合计算功能，通过查询分析语法得到的数据结果默认以表格方式进行展示。

基本构成

- 表头
- 行

- 列

其中：

- SELECT项的个数为列数。
- 行数由当前时间区间日志条数经过计算后的个数决定，默认为LIMIT 100。

使用步骤

1. 在查询页面的查询框中输入查询分析语句，选择时间区间后点击右侧的查询/分析按钮。
2. 页面默认显示统计图表页签，以表格 形式展示结果。
3. 在右侧属性配置页签中配置图表属性。

属性配置

配置项	说明
每页条数	每页显示的数据条数。
显示斑马线	开启后表格以斑马线样式显示。
行列变换	单击可对行列进行变换。
隐藏保留字段	开启后保留字段隐藏不显示。
关闭排序功能	开启后即可关闭排序功能。
关闭搜索功能	开启后即可关闭搜索功能。
高亮设置	通过设置高亮规则，可以使符合规则的行或列高亮显示。

示例

筛选原始日志数据中的列，例如原始日志如下：

图 12-2: 原始日志

1. 筛选其中最近10条日志的`method`、`request_size`和`request_time`。

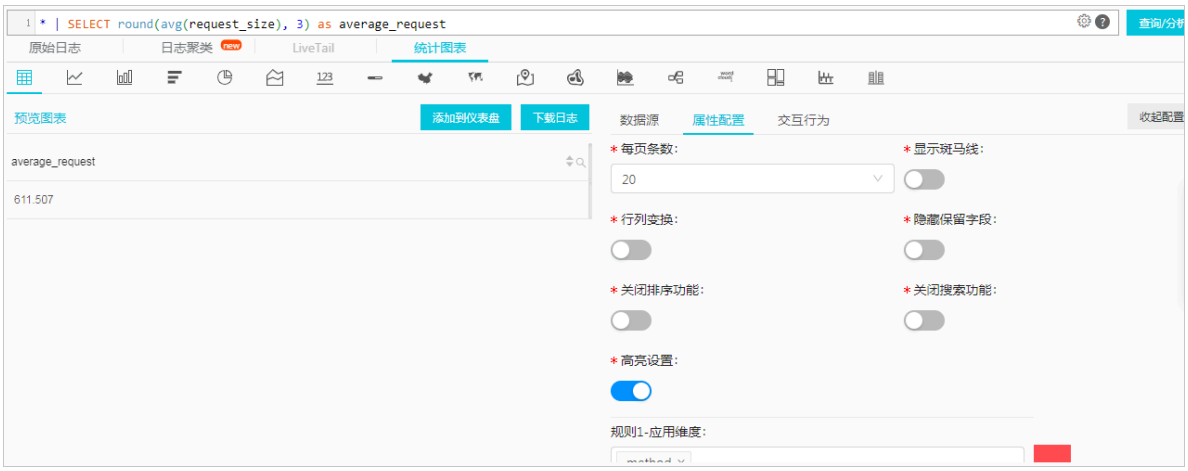
```
* | SELECT method, request_size, request_timetus GROUP BY method,
request_size, request_time LIMIT 10
```

图 12-3: case 1

2. 计算单个数据，如当前时间区间request_size平均值（平均请求时间），并保留3位小数。

```
* | SELECT round(avg(request_size), 3) as average_request
```

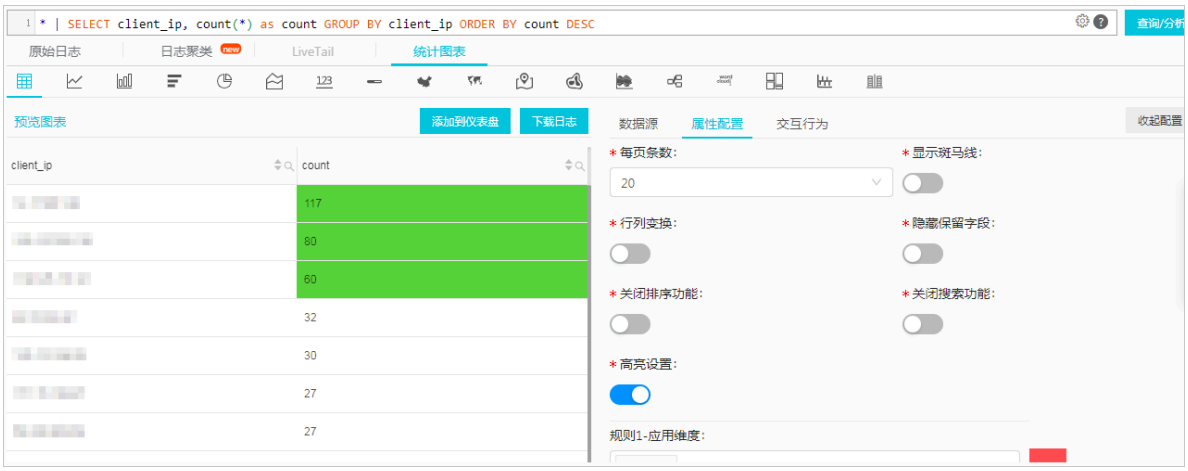
图 12-4: case 2



3. 计算分组数据，如当前时间区间client_ip分布情况，并降序排列。

```
* | SELECT client_ip, count(*) as count GROUP BY client_ip ORDER BY
count DESC
```

图 12-5: case 3



12.1.3 折线图

线图属于趋势类分析图表，一般用于表示一组数据在一个有序数据类别（多为连续时间间隔）上的变化情况，用于直观分析数据变化趋势。

在线图中，我们可以清晰的观测到数据在某一个周期内的变化，主要反映在：

- 递增性或递减性
- 增减的速率情况

- 增减的规律（如周期变化）
- 峰值和谷值

所以，线图是用于分析数据随时间变化趋势的最佳选择。同时，也可以绘制多条线用于分析多组数据在同一时间周期的变化趋势，进而分析诸如数据之间的相互作用和影响（如同增同减，成反比等）。

基本构成

- X轴
- 左Y轴
- 右Y轴（可选）
- 数据点
- 变化趋势线
- 图例

使用步骤

1. 键入查询分析语句，选择时间区间后点击右侧查询/分析按钮。
2. 选择折线图 。
3. 在右侧属性配置页签中配置图表属性。

> 说明:
> 线图单条线的数据记录数要大于2，以免无法分析数据趋势，同时，建议同一个图上不要超过5条线。

属性配置

配置项	说明
X轴	一般为有序数据类别（时间序列）。
左Y轴	可以配置一列或多列数据对应到左轴数值区间。
右Y轴	可以配置一列或多列数据对应到右轴数值区间（右轴图层高于左轴）。
为柱列	将已选择的左Y轴或者右Y轴中的一列以柱状形式表示。
图例位置	图例在图表中的位置，可以配置为上、下、左和右。
左Y轴格式化	将Y轴数据按照指定格式进行显示。

配置项	说明
右Y轴格式化	
边距	坐标轴距离图表边界距离，包括上边距、下边距、右边距和左边距。

示例

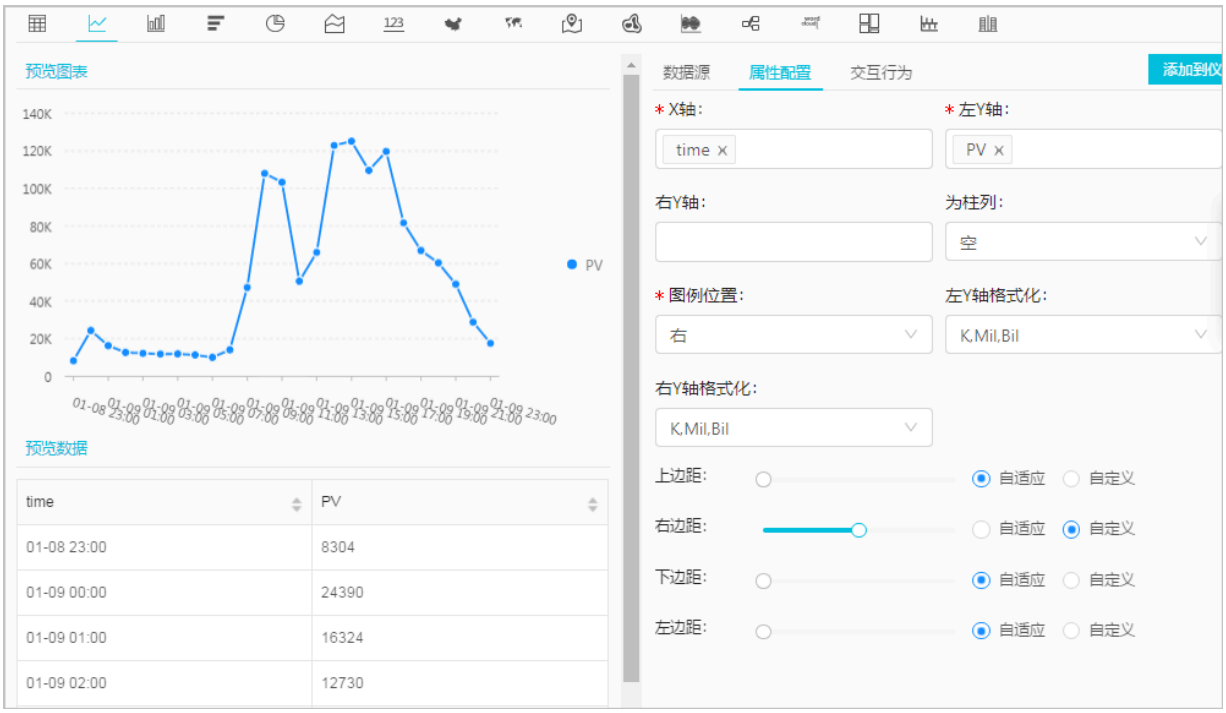
简单折线图

查询`10.0.192.0`这个IP在最近1天内的访问情况：

```
remote_addr: 10.0.192.0 | select date_format(date_trunc('hour',
__time__), '%m-%d %H:%i')
as time, count(1) as PV group by time order by time limit 1000
```

X轴选择`time`，左Y轴选择`PV`并调整图例位置为下方显示，合理改变间距。

图 12-6: 简单折线图



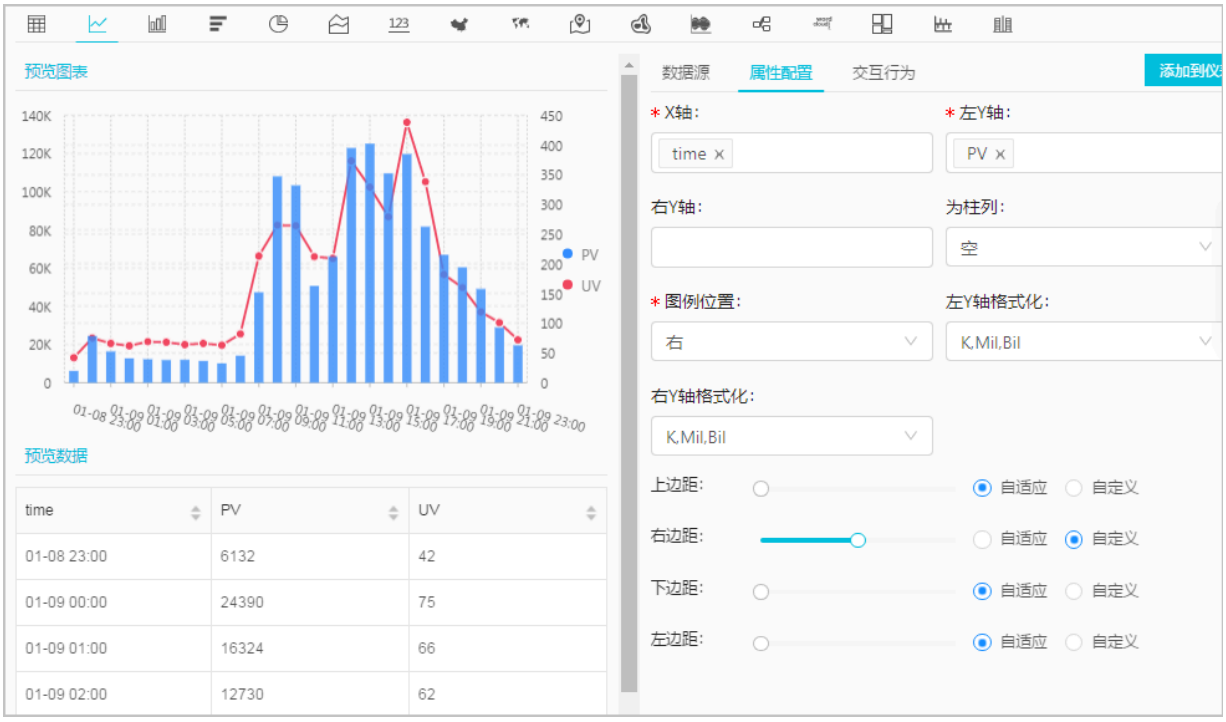
双轴折线图

查询最近1天内的访问PV、UV：

```
* | select date_format(date_trunc('hour', __time__), '%m-%d %H:%i')
as time, count(1) as PV, approx_distinct(remote_addr) as UV group by
time order by time limit 1000
```

X轴选择time，左Y轴选择PV，右Y轴选择UV并指定PV为柱状显示。

图 12-7: 双轴折线图



12.1.4 柱状图

柱状图使用垂直或水平的柱子显示类别之间的数值比较，和折线图的不同之处在于，柱状图描述分类数据，并统计每一个分类中的数量，而折线图描述有序数据。

同时，您也可以绘制多个矩形对应同一个分类属性，分为分组和层叠两种模式，进而分析该分类数据在不同维度上的区别。

基本构成

- X轴（横轴）
- Y轴（纵轴）
- 矩形块
- 图例

日志服务提供的柱状图，默认采用垂直柱子，即矩形块宽度一定，高度代表数值大小。有多列数据映射到Y轴时，采用分组柱状形式显示。

使用步骤

1. 键入查询分析语句，选择时间区间后点击右侧查询/分析按钮。
2. 在图表栏中选择柱状图 。
3. 在右侧属性配置页签中配置图表属性。

> 说明:
> 柱状图适用于不超过20条的数据，建议使用`LIMIT`进行控制，以免横向宽度过宽导致分析对比情况不直观。同时，当有多列数据映射到Y轴时，建议不要超过5个。

属性配置

配置项	说明
X轴	一般为分类数据。
Y轴	可以配置一列或多列数据对应到左轴数值区间。
图例位置	图例在图表中的位置，可以配置为上、下、左和右。
Y轴格式化	将Y轴数据按照指定格式进行显示。
间距	坐标轴距离图表边界距离，包括上边距、下边距、右边距和左边距。

示例

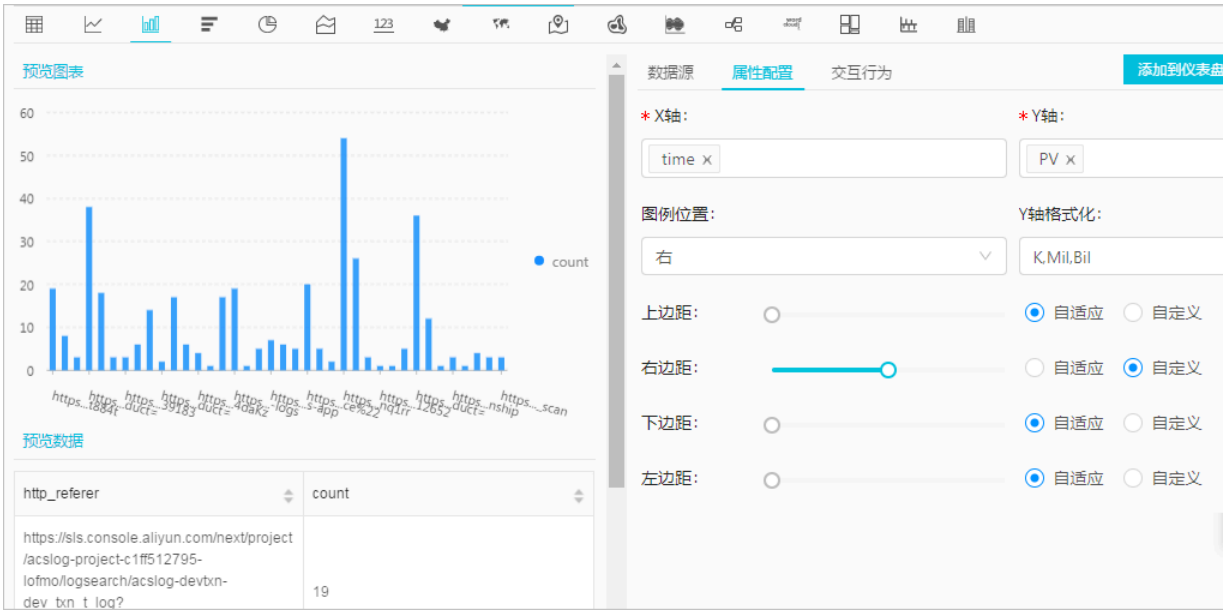
简单柱状图

查看当前时间区间每种`http_referer`的访问次数。

```
* | select  http_referer, count(1) as count group by http_referer
```

X轴选择`http_referer`，Y轴选择`count`。

图 12-8: 简单柱状图



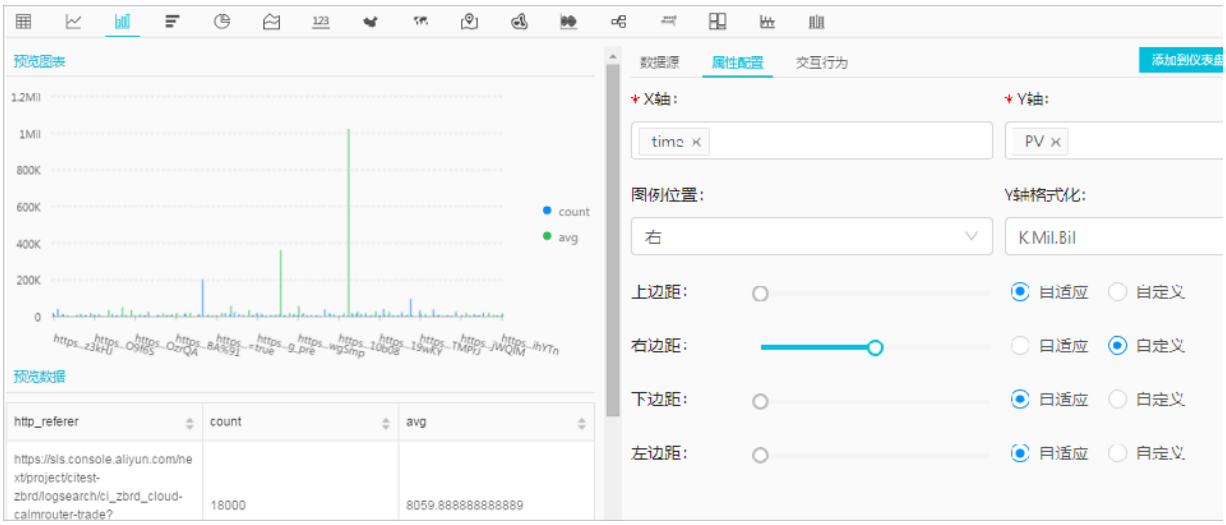
分组柱状图

查看当前时间区间每种http_referer的访问次数和平均字节数。

```
* | select  http_referer, count(1) as count, avg(body_bytes_sent) as
avg group by http_referer
```

X轴选择http_referer，Y轴选择count和avg。

图 12-9: 分组柱状图



12.1.5 条形图

条形图是柱状图另一种形式，即横向柱状图。条形图通常用于分析Top场景，配置方式也和柱状图类似。

基本构成

- X轴（纵轴）
- Y轴（横轴）
- 矩形块
- 图例

条形图矩形块高度一定，宽度代表数值大小。有多列数据映射到Y轴时，采用分组柱状形式显示。

使用步骤

1. 键入查询语句，选择时间区间后单击查询/分析。
2. 在图表栏中选择条形图 。

3. 在右侧属性配置页签中配置图表属性。

> 说明:
>
> - 条形图适用于不超过20条的数据，建议使用LIMIT进行控制，以免纵向高度过高导致分析对比情况不直观，分析Top场景时候使用ORDER BY配合。同时，当有多列数据映射到Y轴时，建议不要超过5个。
> - 支持使用分组条形图，但是条形图仅适用于同增同减的分类。

属性配置

表 12-1: 配置项说明

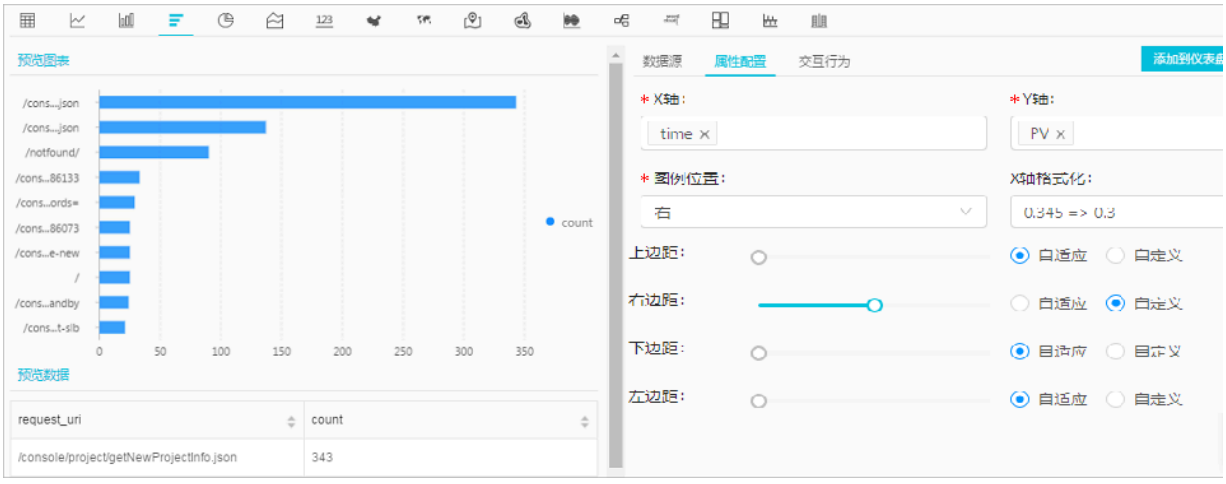
配置项	说明
X轴	一般为分类数据。
Y轴	可以配置一列或多列数据对应到左轴数值区间。
图例位置	图例在图表中的位置，可以配置为上、下、左和右。
X轴格式化	将X轴数据按照指定格式进行显示。
间距	坐标轴距离图表边界距离，包括上边距、下边距、右边距和左边距。

简单条形图示例

分析前十访问的request_uri:

```
* | select  request_uri, count(1) as count group by request_uri order by count desc limit 10
```

图 12-10: 简单条形图



12.1.6 饼图

饼图用于表示不同分类的占比情况，通过弧度大小来对比各种分类。饼图通过将一个圆饼按照分类的占比划分成多个区块，整个圆饼代表数据的总量，每个区块（圆弧）表示该分类占总体的比例大小，所有区块（圆弧）的加和等于100%。

构成

- 扇形
- 文本百分比
- 图例

类型

日志服务提供默认饼图、环图及南丁格尔玫瑰图三种类型的饼图。

环图

环图本质上是将饼图中心挖空，相比于饼图来说有如下优点：

- 在原有构成的基础上增加了总数显示，展示了更多的信息。
- 两个饼图直接进行比较是非常不直观的，两个环图间可以通过环状条长度进行简单的对比。

南丁格尔玫瑰图

南丁格尔玫瑰图本质上并不是环图，而是在极坐标系下画出来的柱状图，每一个分类数据被圆弧平分，使用圆弧的半径长短表示数据的大小，相比于饼图来说有如下优点：

- 饼图适用于不超过10条的分类数据，南丁格尔玫瑰图则适用于分类较多的场景（10-30条数据）。
- 由于半径和面积是成平方的关系，南丁格尔玫瑰图放大了各个分类数据之间值的差异，尤其适合对比大小相近的数值。
- 由于圆形有周期的特性，南丁格尔玫瑰图也适用于表示一个周期的时间概念，比如星期、月份。

使用步骤

1. 键入查询分析语句，选择时间区间后单击右侧查询/分析。
2. 在图表栏中选择饼图 。
3. 在右侧属性配置页签中配置图表属性。

> 说明：
>
> - 饼图和环图适用于10条以内的数据，建议使用`LIMIT`进行控制，以免不同色的分面太多导致分析不直观。

- 分析超过10条数据建议采用南丁格尔玫瑰图或者柱状图。

属性配置

配置项	说明
饼图表型	提供饼图（默认）、环图以及南丁格尔玫瑰图。
分类	分类数据。
数值列	分类数据对应的数值。
图例位置	图例在图表中的位置，可以配置为上、下、左和右。
格式化	将数据按照指定格式进行显示。
间距	坐标轴距离图表边界距离，包括上边距、下边距、右边距和左边距。

示例

饼图

分析访问`requestURI`的占比情况：

```
* | select requestURI as uri , count(1) as c group by uri limit 10
```

图 12-11: 饼图

环图

分析访问requestURI的占比情况：

```
* | select requestURI as uri , count(1) as c group by uri limit 10
```

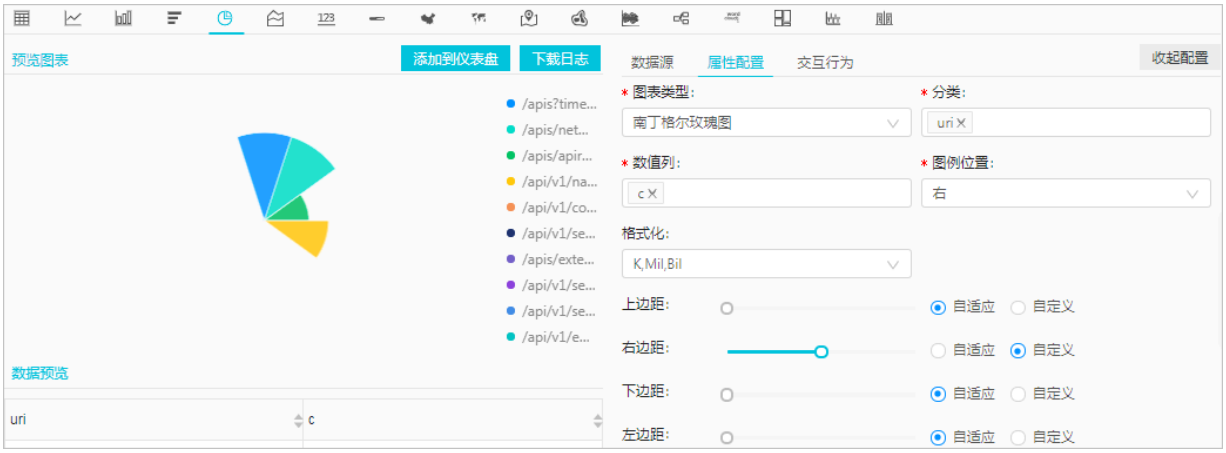
图 12-12: 环图

南丁格尔玫瑰图

分析访问requestURI的占比情况：

```
* | select requestURI as uri , count(1) as c group by uri limit 10
```

图 12-13: 南丁格尔玫瑰图



12.1.7 面积图

面积图是在折线图的基础之上形成的，它将折线图中折线与坐标轴之间的区域使用颜色进行填充，这个填充即为我们所说的面积，颜色的填充可以更好的突出趋势信息。和折线图一样，面积图强调数量随时间而变化的程度，用于突出总值趋势。它们最常用于表现趋势和关系，而不是传达特定的值。

基本构成

- X轴（横轴）
- Y轴（纵轴）
- 面积块

使用步骤

1. 键入查询分析语句，选择时间区间后点击右侧查询/分析按钮。
2. 在图表栏中选择面积图 。
3. 在右侧属性配置中配置图表属性。

> **说明:**
> 面积图单个面积块数据记录数要大于2，以免无法分析数据趋势，同时，建议同一个图上不要超过5组面积块。

属性配置

配置项	说明
X轴	一般为有序数据类别（时间序列）。
Y轴	可以配置一列或多列数据对应到左轴数值区间。
图例位置	图例在图表中的位置，可以配置为上、下、左和右。
格式化	将数据按照指定格式进行显示。
间距	坐标轴距离图表边界距离，包括上边距、下边距、右边距和左边距。

示例

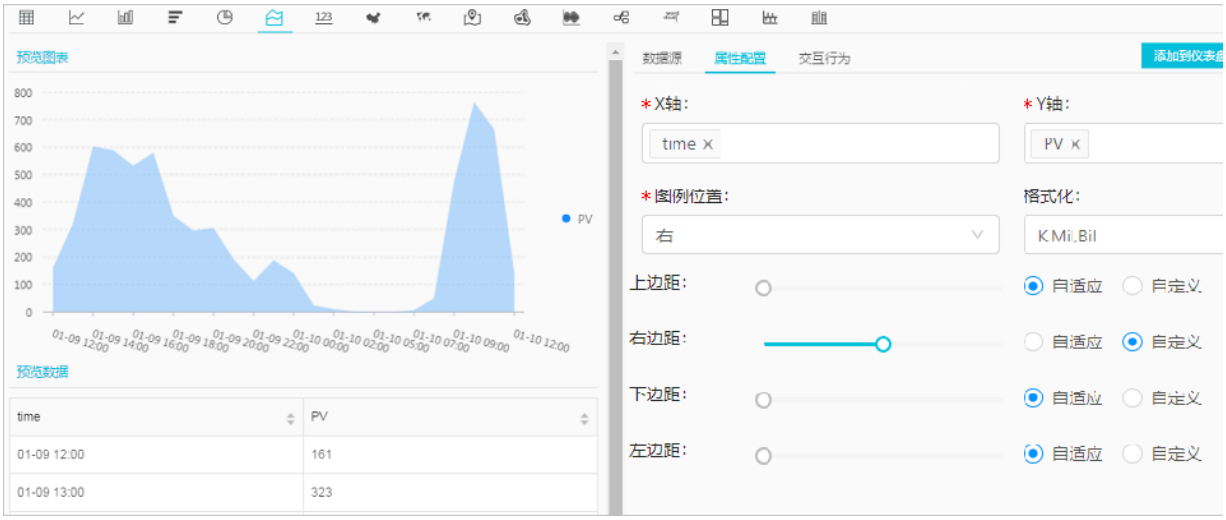
简单面图

`10.0.192.0`这个IP在最近1天内的访问情况：

```
remote_addr: 10.0.192.0 | select date_format(date_trunc('hour',
__time__), '%m-%d %H:%i') as time, count(1) as PV group by time order
by time limit 1000
```

X轴选择`time`，Y轴选择`PV`。

图 12-14: 简单面图

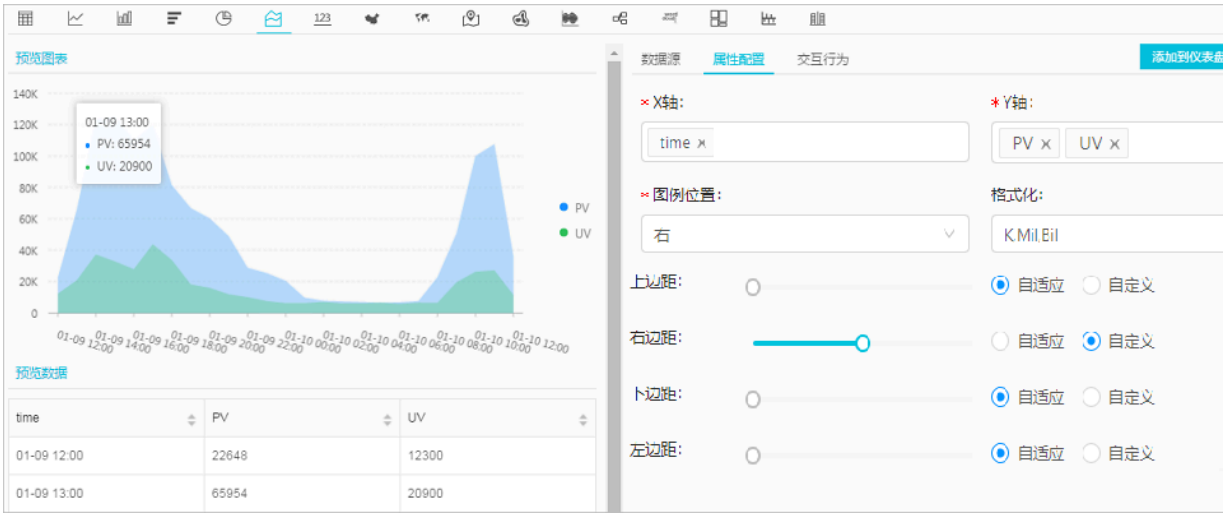


层叠面图

```
* | select date_format(date_trunc('hour', __time__), '%m-%d %H:%i')
as time, count(1) as PV, approx_distinct(remote_addr) as UV group by
time order by time limit 1000
```

X轴选择time，Y轴选择PV和UV。

图 12-15: 层叠面图



12.1.8 单值图

单值图可以用于突出显示单个数值。单值图的类型包括：

- 矩形框：用于展示一般数值。
- 刻度盘：用于查看数值与设定阈值的接近程度。
- 同比环比图：用于查看同比和环比函数的SQL查询结果，分析语法请参考#unique_32。

默认选择矩形框图显示。矩形框图作为最简单直接的数据表现形式，直观清晰地将某一个点上的数据展示出来，一般用于表示某一个时间点上的关键信息。针对比例类指标的显示，可选用刻度盘类型。

构成

- 主文案
- 单位（可选）
- 描述（可选）
- 分类

使用步骤

1. 键入查询语句，选择时间区间后点击右侧查询按钮。

2. 在图表栏中选择单值图 。

3. 在右侧属性配置页签中配置图表属性。

> 说明:
> 日志服务数字图会自动根据数值大小进行归一化操作，如230000会被处理为230K，如果需要自己定义数值格式，请通过#unique_21在实时分析阶段进行处理。

属性配置

- 矩形框配置说明：

矩形框配置	说明
图表类型	矩形框。
数值列	默认选择该列的第一行数据进行展示。
单位	数据的单位，显示在数值之后。
单位字号	单位的字号，可以拖动调整。取值范围为10px~100px。
数值描述	数值的描述，显示在数值之下。
数值描述字号	数值描述的字号，可以拖动调整。取值范围为10px~100px。
格式化	将数据按照指定格式进行显示。
字号	数值的字号，可以拖动调整。取值范围为10px-100px。
字体颜色	数字和文字的颜色，可以选择推荐颜色或自定义设置。
背景颜色	背景的颜色，可以选择推荐颜色或自定义设置。

- 刻度盘配置说明：

配置项	说明
图标类型	将查询结果以刻度盘形式展示。
实际值	默认选择该列的第一行数据进行展示。
单位	刻度盘数值的单位。
字号	数值和单位的字号，取值范围为10px~100px。
数值描述	数值的描述，显示在数值之下。
数值描述字号	数值描述的字号，可以拖动调整。取值范围为10px~100px。
刻度盘最大值	刻度盘显示刻度的最大值，默认为100。
最大值所在列	使用查询结果打开时，刻度盘最大值变成最大值所在列，取值可以从查询结果中选取。

配置项	说明
使用查询结果	使用查询结果的情况下，总值可以从查询结果中选取。
格式化	将数据按照指定格式进行显示。
颜色区域个数	即将刻度盘分为几个数值区域，每个区域以不同颜色表示。 颜色区域个数取值范围为2、3、4、5，默认为3个颜色区域。
区域最大值	刻度盘数值区域的最大值，最后一个区域最大值默认为刻度盘最大值，不需要指定。 说明: 刻度盘默认3个颜色区域，且区域范围默认均分，如果您调整了颜色区域个数，不会改变默认区域的范围，您需要根据需求重新指定每个区域的最大值。
字体颜色	数值在仪表盘中显示的颜色。
区域	默认3个区域，对应的颜色分别为蓝、黄和红。 如果您将颜色区域个数更改为3个以上，新增的区域默认为蓝色，您可以重新调整各个区域的颜色。
显示标题	您可以在仪表盘中添加刻度盘类型的单值图，显示标题用来控制刻度盘形式的单值图标题在仪表盘页面的显示或隐藏。默认为关闭状态，即不显示刻度盘标题。 单击开启后不会在当前页面中显示效果，需要创建或修改报表后在仪表盘页面中查看。

- 同比环比图配置说明：

配置项	说明
图表类型	将查询结果以同比环比图形式展示。
显示值	显示在同比环比图中心的数值，一般设置为同比环比函数中当前时段的统计结果。
对比值	用于和阈值比较的数值，一般设置为同比环比函数中当前时段和之前时段的对比结果。
字号	显示值的字号，取值范围为10px~100px。
单位	显示值的单位，显示在显示值之后。
单位字号	显示值单位的字号，取值范围为10px~100px。
比较值单位	比较值的单位，显示在比较值之后。

配置项	说明
比较值字号	比较值及其单位的字号，取值范围为10px~100px。
数值描述	对显示的数值及增长趋势的描述，显示在数值下方。
数值描述字号	数值描述的字号，取值范围为10px~100px。
趋势比较阈值	用于衡量对比值变化趋势的数值。 例如对比值和阈值的差值为-1： - 设置趋势比较阈值为0，页面会显示下降箭头，表示数值变化呈下降趋势。 - 设置趋势比较阈值为-1，系统认为数据无变化，页面不显示变化趋势。 - 设置趋势比较阈值为-2，页面显示上升箭头，表示数值变化呈上升趋势。
格式化	将数据按照指定格式进行显示。
字体颜色	显示值和数值描述的字体颜色。
增长字体颜色	对比值大于阈值时，对比值显示的字体颜色。
增长背景颜色	对比值大于阈值时，显示的背景颜色。
下降字体颜色	对比值小于阈值时，对比值显示的字体颜色
下降背景颜色	对比值小于阈值时，显示的背景颜色。
相等背景颜色	对比值等于阈值时，显示的背景颜色。

示例

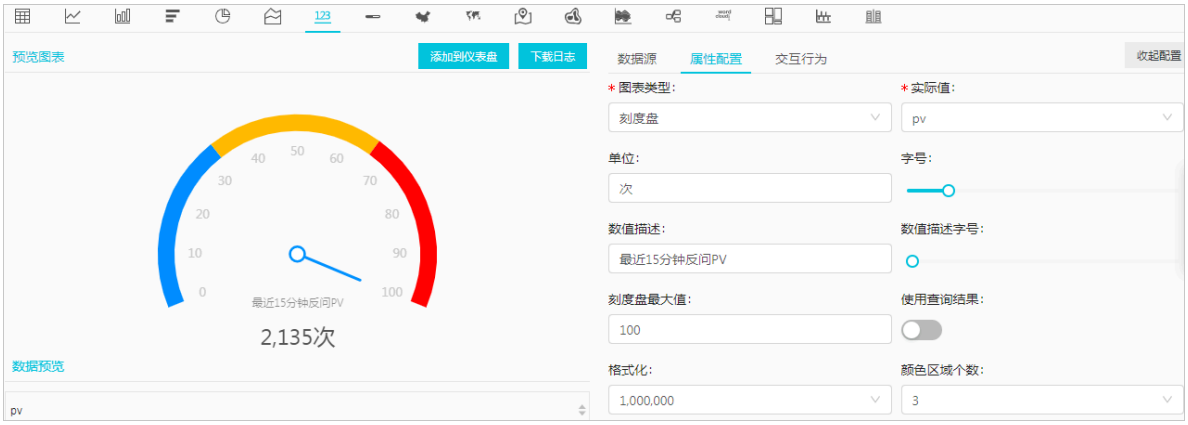
执行以下查询分析语句查看访问量，并以图表方式展示分析结果：

- **矩形框**

```
* | select count(1) as pv
```

- **刻度盘**

```
* | select count(1) as pv
```



- 同比环比图

  查看今天与昨天访问量的对比：

```
* | select diff[1],diff[2], diff[1]-diff[2] from (select compare( pv
 , 86400) as diff from (select count(1) as pv from log))
```

12.1.9 进度条

进度条图表用于显示百分比内容，您可以通过进度条的属性配置调整进度条的样式，并设置进度条的显示规则。

构成

- 实际值
- 单位（可选）
- 总值

使用步骤

1. 输入查询语句，选择时间区间后单击右侧查询/分析按钮。
2. 选择 ，即进度条图。
3. 进行图表属性配置。

属性配置

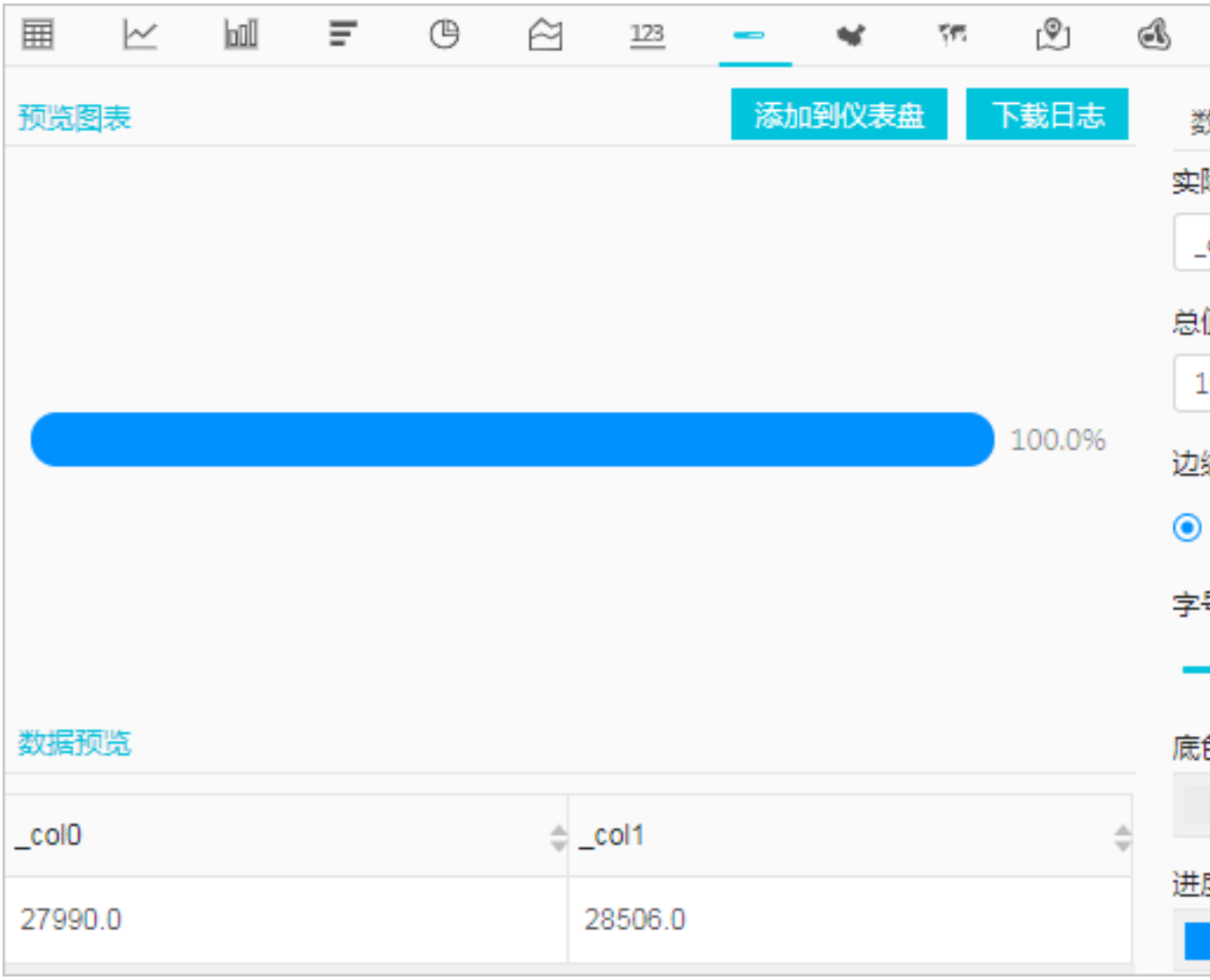
配置项	说明
实际值	默认选择该列的第一行数据进行展示。
单位	进度条数值的单位。
总值	进度条的总数值，默认100。
最大值所在列	使用查询结果打开时，总值变成最大值所在列，取值可以从查询结果中选取。

配置项	说明
使用查询结果	使用查询结果的情况下，总值可以从查询结果中选取。
边缘形状	进度条的边缘形状。
垂直显示	使进度条垂直显示。
字号	可以调整进度条字体的大小。
粗细	可以调整进度条的粗细。
底色	进度条的底色。
字体颜色	进度条的字体颜色。
进度条默认颜色	默认的进度条颜色。
颜色显示方式	进度条的颜色显示方式。
开始颜色	当颜色显示方式选择渐变时，可以设置进度条的开始颜色。
结束颜色	当颜色显示方式选择渐变时，可以设置进度条的结束颜色。
显示颜色	当颜色显示方式选择按照规则显示时，可以设置进度条的显示颜色。 说明: 将实际值与设置的临界值根据判断条件进行比较，符合判断条件则按照此处设置的颜色进行显示，否则显示默认颜色。
判断条件	当颜色显示方式选择按照规则显示时，可以设置进度条颜色显示的判断条件。
临界值	当颜色显示方式选择按照规则显示时，可以设置进度条颜色显示的判断条件的临界值。

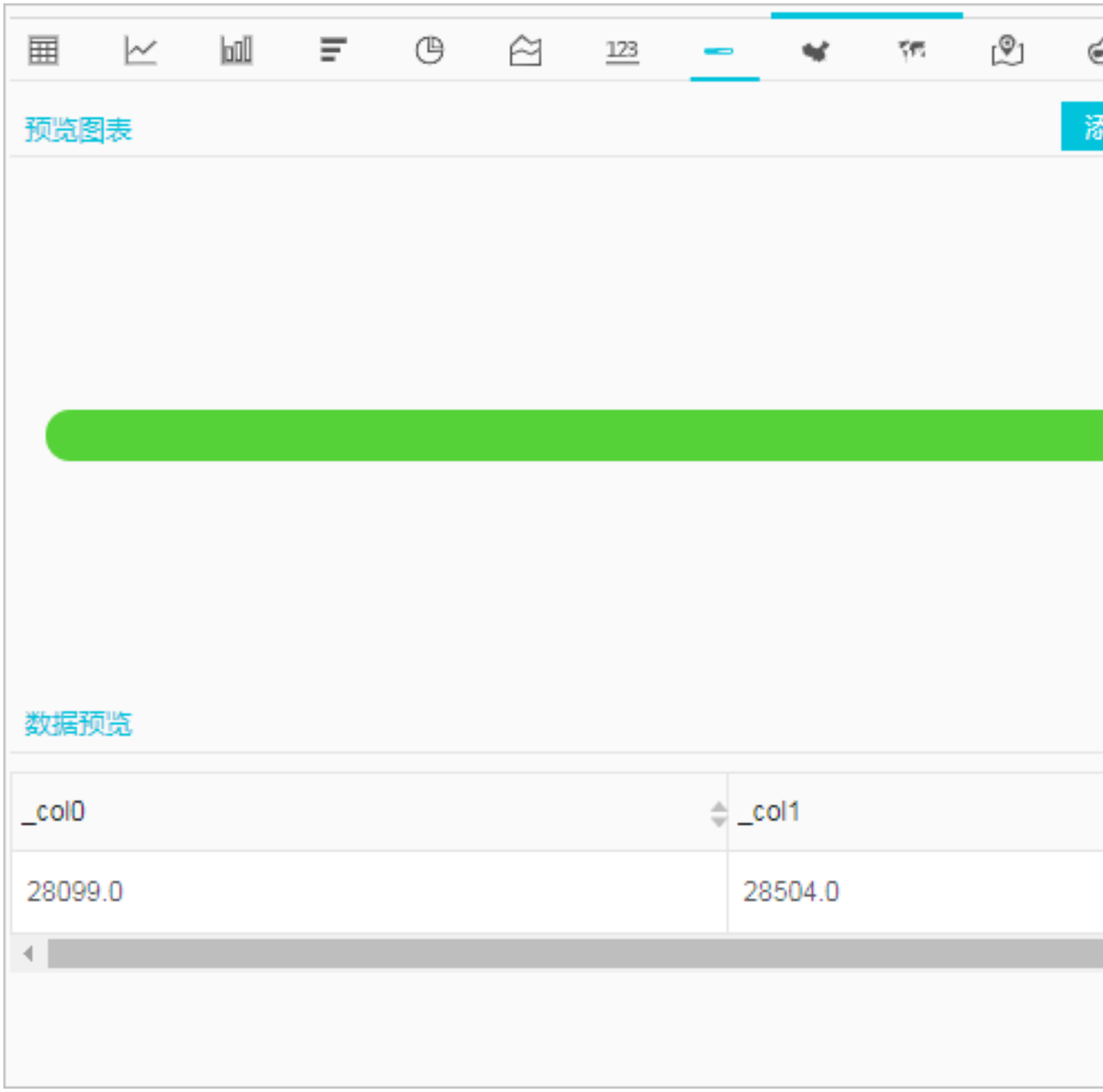
应用场景

进度条主要应用于百分比指标或数据占比的显示。

```
* | select diff[1],diff[2] from (select compare( pv , 86400) as diff
from (select count(1) as pv from log))
```



如果颜色显示方式选择按照规则显示，可以根据设定的规则动态显示颜色，如果不满足设置的规则，则显示默认颜色。



12.1.10 地图

以地图作为背景，通过图形颜色、图像标记的方式展示地理数据信息。日志服务提供了三种地图方式，分别为：中国地图、世界地图以及高德地图（高德地图分为点图和热力图。您可以在查询分析语句中使用特定的函数，日志服务会将您的分析结果以地图方式展示出来。

基本构成

- 地图画布
- 色块

配置项

配置项	说明
位置信息	日志数据中记录的位置信息，在不同的地图类型中以不同的尺度表示。 · 省份（中国地图） · 国家（世界地图） · 经纬度（高德地图）
数值列	位置信息对应的数据量。

使用步骤

1. 键入查询语句，选择时间区间后点击右侧查询按钮。
   - 中国地图：使用`ip_to_province`函数。
   - 世界地图：使用`ip_to_country`函数。
   - 高德地图：使用`ip_to_geo`函数。
2. 选择地图 。
3. 进行图表属性配置。

应用场景

中国地图

支持使用`ip_to_province`函数生成中国地图。

· SQL语句：

```
* | select  ip_to_province(remote_addr) as address, count(1) as count
group by address order by count desc limit 10
```

· 数据集：

address	count
广东省	163
浙江省	110
福建省	107
北京市	89
重庆市	28
黑龙江省	19

- 省份信息选择address，数值列选择count。

图 12-16: 中国地图

世界地图

支持使用ip_to_country函数生成世界地图。

- SQL语句：

```
* | select  ip_to_country(remote_addr) as address, count(1) as count
 group by address order by count desc limit 10
```

- 数据集：

address	count
中国	8354
美国	142

- 国家信息选择address，数值列选择count。

图 12-17: 世界地图

高德地图

支持使用ip_to_geo函数生成高德地图。数据集先纬后经，以“，”为分隔符，如果数据为两列lng（经度）和lat（纬度），可以使用concat('lat', ',', lng')合并为一列。

- SQL语句：

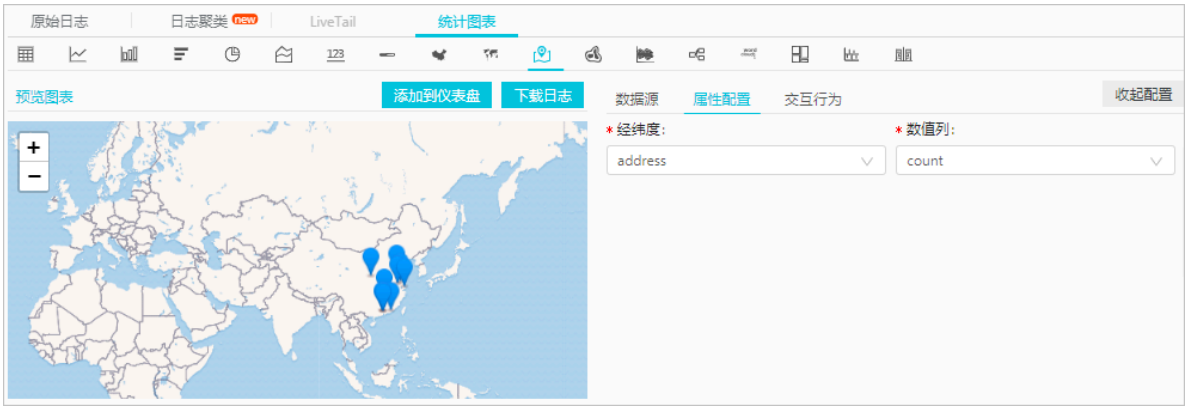
```
* | select  ip_to_geo(remote_addr) as address, count(1) as count
group by address order by count desc limit 10
```

- 数据集：

address	count
39.9289,116.388	771
39.1422,117.177	724
29.5628,106.553	651
30.2936,120.161420	577
26.0614,119.306	545
34.2583,108.929	486

- 经纬度信息选择address，数值列选择count。

图 12-18: 高德地图-点图



默认返回点图。如数据点分布密集，您也可以切换为热力图。

图 12-19: 高德地图-热力图

12.1.11 流图

流图（Flow Chart）也叫主题河流图（ThemeRiver），是围绕中心轴线进行布局的一种堆叠面积图。不同颜色的条带状分支代表了不同的分类信息，条状带的宽度映射了对应的数值大小。此外，原数据集中的时间属性，映射到X轴上，是一个三维关系的展现。

流图可以通过图表类型切换为线图和柱状图，需要注意的是柱状图默认以层叠形式展现，不同分类数据的起点是从上个柱状的顶部开始。

基本构成

- X轴（横轴）
- Y轴（纵轴）
- 条状

使用步骤

1. 键入查询分析语句，选择时间区间后点击右侧查询/分析按钮。
2. 在图表栏中选择 ，即流图。
3. 在右侧属性配置中配置图表属性。

配置项

配置项	说明
图表类型	提供线图（默认）、面积图、以及柱状图（层叠）。
X轴	一般为有序数据类别（时间序列）。
Y轴	可以配置一列或多列数据对应到左轴数值区间。
聚合列	需要在第三维上聚合的信息。
图例位置	图例在图表中的位置，可以配置为上、下、左和右。
格式化	将数据按照指定格式进行显示。
间距	坐标轴距离图表边界距离，包括上边距、下边距、右边距和左边距。

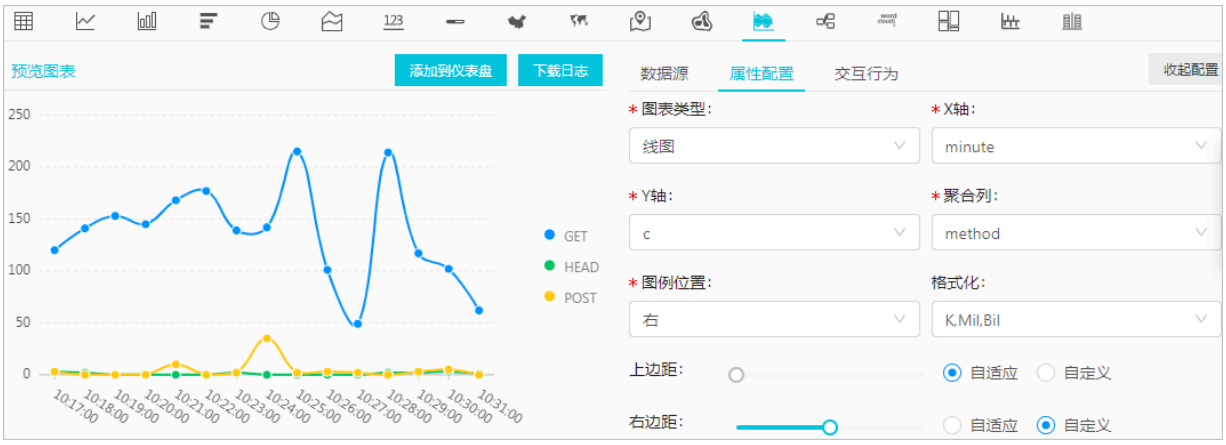
示例

流图适合三维关系的展示，时间-分类-数值的展现。

```
* | select date_format(from_unixtime(__time__ - __time__% 60), '%H:%i:%S') as minute, count(1) as c,  request_method group by minute, request_method order by minute asc limit 100000
```

X轴选择`minute`，Y轴选择`c`，聚合列选择`request_method`。

图 12-20: 流图



12.1.12 桑基图

桑基图 (Sankey Diagram)，是一种特定类型的流图，用于描述一组值到另一组值的流向。适合网络流量等场景，通常包含3组值`source`、`target`以及`value`。`source`和`target`描述了节点的关系，而`value`描述了该`source`和`target`之间边的关系。

功能特点

桑基图具有以下特点：

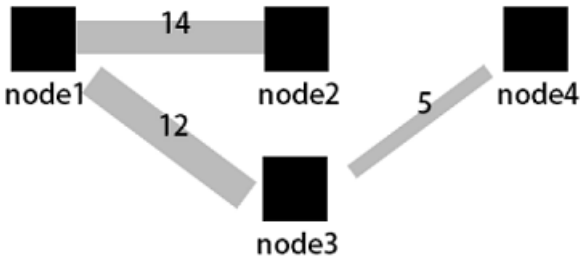
- 起始流量和结束流量相同，所有主支宽度的总和与所有分出去的分支宽度总和相等，保持能量的平衡。
- 在内部，不同的线条代表了不同的流量分流情况，它的宽度成比例地显示此分支占有的流量。
- 节点不同的宽度代表了特定状态下的流量大小。

例如以下数据可以用桑基图表示：

source	target	value
node1	node2	14
node1	node3	12
node3	node4	5
…	..	…

桑基图如此描述上述数据的关系：

图 12-21: 桑基图的数据关系



基本构成

- 节点
- 边

使用步骤

1. 键入查询分析语句，选择时间区间后单击查询/分析按钮。

2. 在图标蓝中选择桑基图。

3. 在右侧属性配置中配置图表属性。

属性配置

配置项	说明
起点列	描述起始节点。
终点列	描述终点节点。
数值列	链接起点节点和终点节点的值。
边距	坐标轴距离图表边界距离，包括上边距、下边距、右边距和左边距。

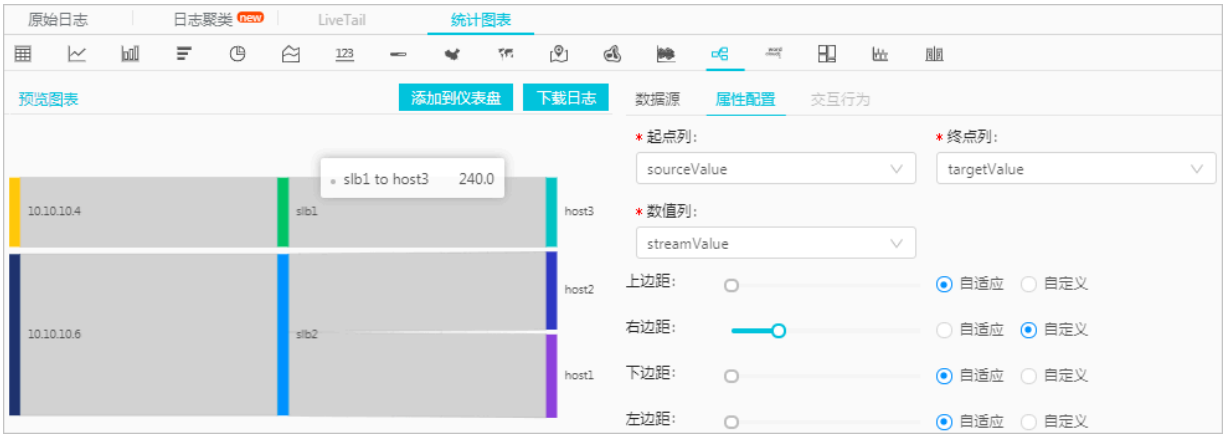
示例

普通桑基图

如果日志字段包含了`source`、`target`和`value`，即每条日志本身就是节点和边的关系，可以通过#unique_48获取到`steamValue`的总和。

```
* | select sourceValue, targetValue, sum(streamValue) as streamValue
from (select sourceValue, targetValue,
 streamValue, __time__ from log group by  sourceValue, targetValue,
streamValue, __time__ order by __time__ desc) group by sourceValue,
 targetValue
```

图 12-22: 普通桑基图

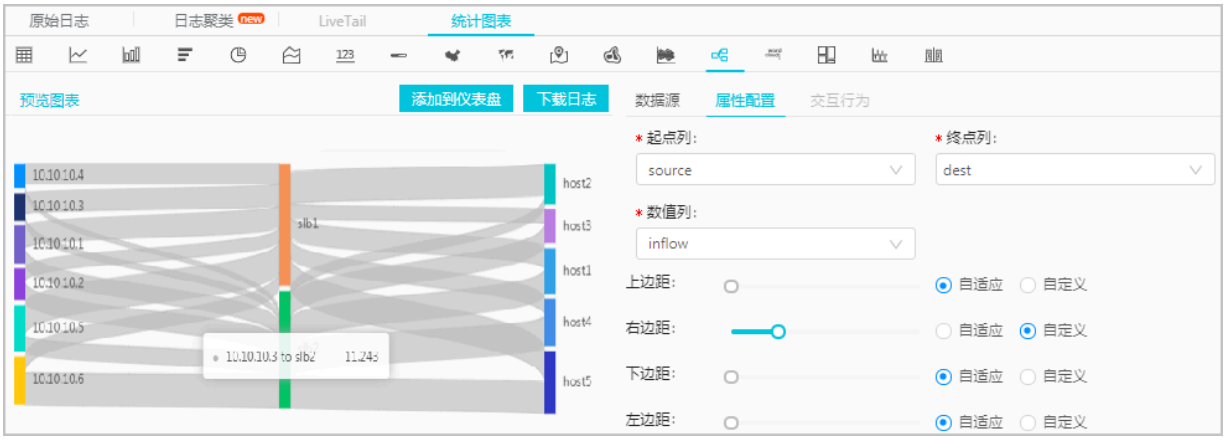


负载均衡7层访问日志场景

日志服务支持#unique_194，可以直接通过访问日志绘制桑基图。

```
* | select COALESCE(client_ip, slbid, host) as source, COALESCE(host
, slbid, client_ip) as dest, sum(request_length) as inflow group by
grouping sets( (client_ip, slbid), (slbid, host))
```

图 12-23: 嵌套子查询



12.1.13 词云

词云，是文本数据的视觉表示，由词汇组成类似云的彩色图形，用于展示大量文本数据。每个词的重要性以字体大小或颜色显示，能最让用户最快速地感知某一些关键词的权重大小。

基本构成

词云类型的图表为您展示经过数据计算排列的词。

使用步骤

1. 键入查询分析语句，选择时间区间后点击右侧查询/分析按钮。
2. 在图标栏中选择词云。
3. 在右侧属性配置中配置图表属性。

配置项

配置项	说明
词列	代表要展示的一组词的信息。
数值列	每一个词对应的数值信息。
字号	合理调整字号范围以适应画布。 · 最大字号（10px-24px） · 最小字号（50px-80px）

示例

分析NGINX日志中hostname分布：

```
* | select hostname, count(1) as count group by hostname order by
count desc limit 1000
```

词列为`hostname`，数值列为`count`。

图 12-24: 词云

12.1.14 矩形树图

矩形树图，即矩形式树状结构图（Treemap），用矩形面积表示数据的大小。各个小矩形的面积越大，表示占比越大。

基本构成

计算排列得到的矩形块。

使用步骤

1. 键入查询分析语句，选择时间区间后单击右侧的查询/分析。
2. 在图表栏中选择矩形树图 。
3. 在右侧属性配置中配置图表属性。

属性配置

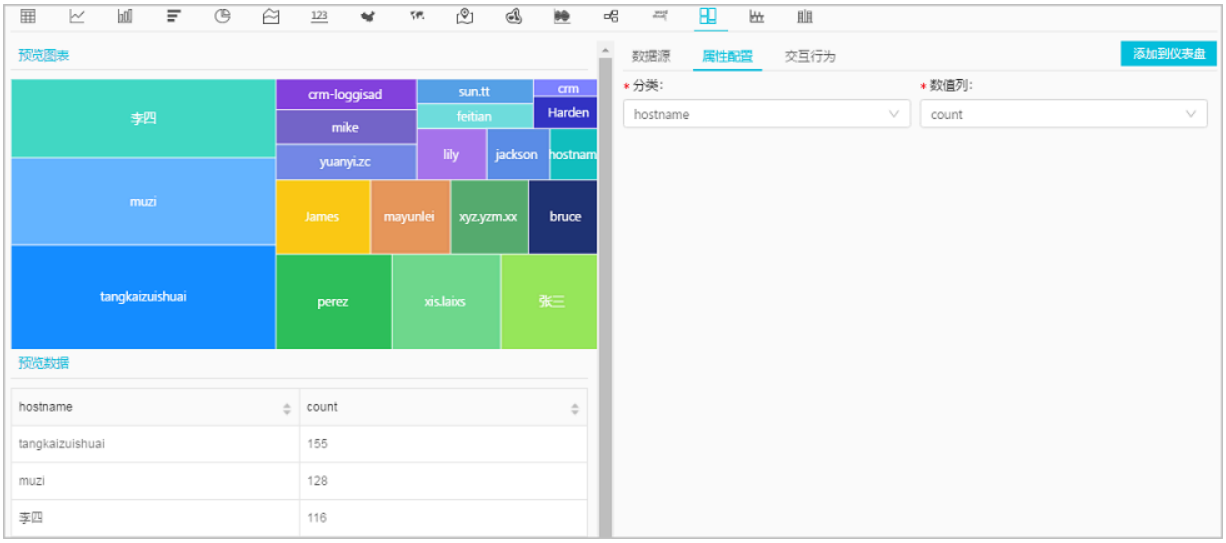
配置	说明
分类	表示数据类别的字段。
数值列	数值字段，某个类别对应的数值越大，其矩形框越大。

示例

分析NGINX日志中hostname分布。

```
* | select hostname, count(1) as count group by hostname order by
count desc limit 1000
```

设置分类为`hostname`，数值列为`count`。



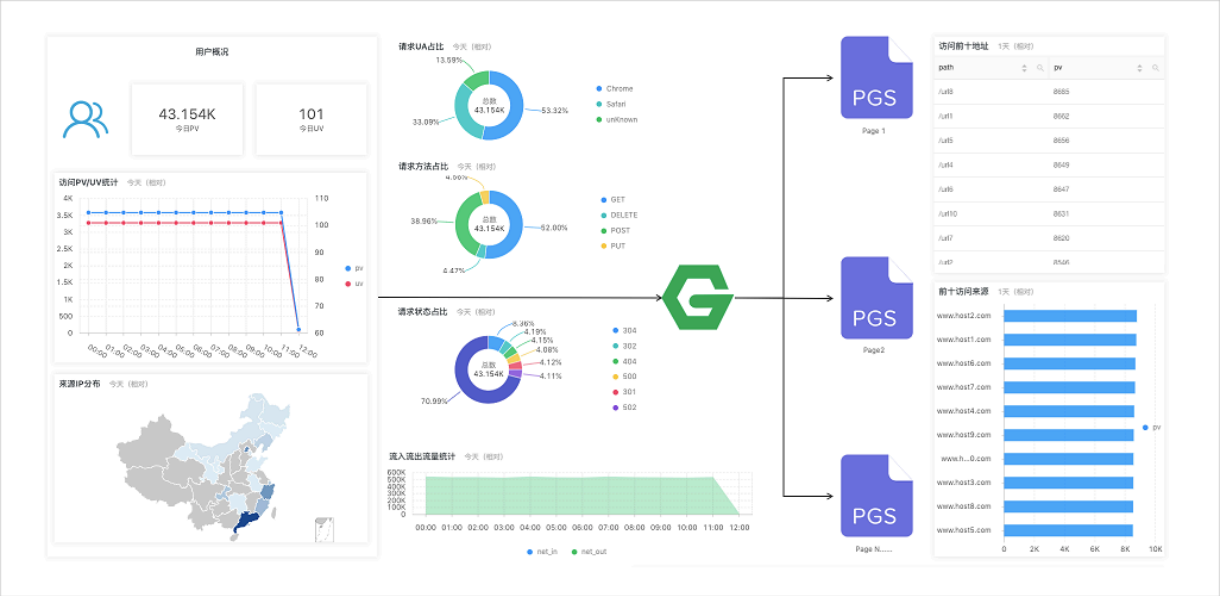
12.2 仪表盘

12.2.1 仪表盘简介

仪表盘是日志服务提供的实时数据分析大盘。您可以将常用的查询语句以图表形式展示，并将多项分析图表保存到仪表盘中。

通过仪表盘可以一次性查看多个分析语句的分析图表，当您打开或刷新仪表盘时，这些分析图表会自动执行一遍查询分析语句。

日志服务同时支持#unique_199功能，您不仅可以在日志服务控制台中查看仪表盘，还可以将某个仪表盘页面外嵌到其他网站页面中，让您的数据分析与数据展示手段更加多样化。另外，添加图表到仪表盘时，还可以设置下钻分析，设置之后，在仪表盘页面中单击该图表，可以得到更深维度的分析结果。



限制说明

- 每个Project最多可创建50个仪表盘。
- 每个仪表盘最多可包含50张分析图表。

功能试用

试用链接：请单击此处试用

用户名：sls_reader1@**********

密码：pnX-32m-MHH-xbm

功能介绍

仪表盘分为显示模式和编辑模式。

- 显示模式

  在显示模式下，支持对仪表盘页面进行多种显示设置，包括：

  - 仪表盘显示设置：例如设置仪表盘的全局时间、对图表设置告警、设置仪表盘页面自动刷新、设置仪表盘全屏显示、设置标题显示方式、根据过滤器过滤图表数据等。
  - 图表显示设置：查看指定图表的分析详情、设置指定图表的时间区间、对指定图表设置告警、下载日志、下载图表、查看是否设置了下钻等。

- 编辑模式

  在编辑模式下，支持对仪表盘进行多种变更操作，包括：

  - 仪表盘设置：支持将仪表盘作为画布，为其添加图表元素，如Markdown图表、自定义图表、文本、图标等图表元素；还可以在图表元素之间添加连接线，连接线支持根据图表位置

自适应；添加过滤器，添加后在显示模式下可以过滤图表数据。另外，为便于排版，可以设置显示网格线，让图标等元素的位置工整有序。

- 图表设置：支持在仪表盘编辑模式下编辑图表，例如修改分析图表的语句、属性、下钻配置等交互行为；

12.2.2 创建和删除仪表盘

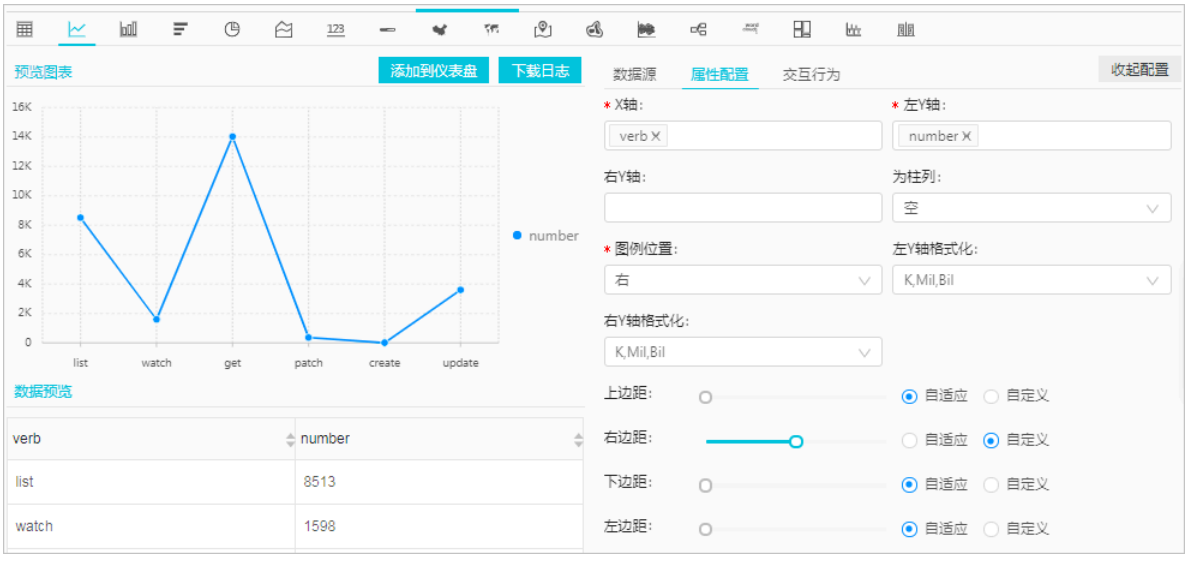
在日志服务控制台中输入查询分析语句，设置图表之后，可以将图表保存在仪表盘中，方便下次查看。仪表盘中可以展示50张分析图表，支持多种显示和自定义编辑设置。

前提条件

- 已成功采集到日志。
- 已#unique_4。

创建仪表盘

1. 登录日志服务控制台，单击Project名称。
2. 单击日志库名称后的 图标，选择查询分析。
3. 在搜索框中输入查询分析语句，并单击查询/分析。
4. 页面自动跳转到统计图表页面，请在右侧属性配置页签中设置图表属性。



5. （可选）设置占位符变量。

如果其他图表的下钻事件为跳转到这个仪表盘，设置占位符变量后，单击其他图表时会跳转到这个仪表盘，占位符变量替换为触发下钻事件的图表值，并以替换变量后的查询语句刷新仪表盘。

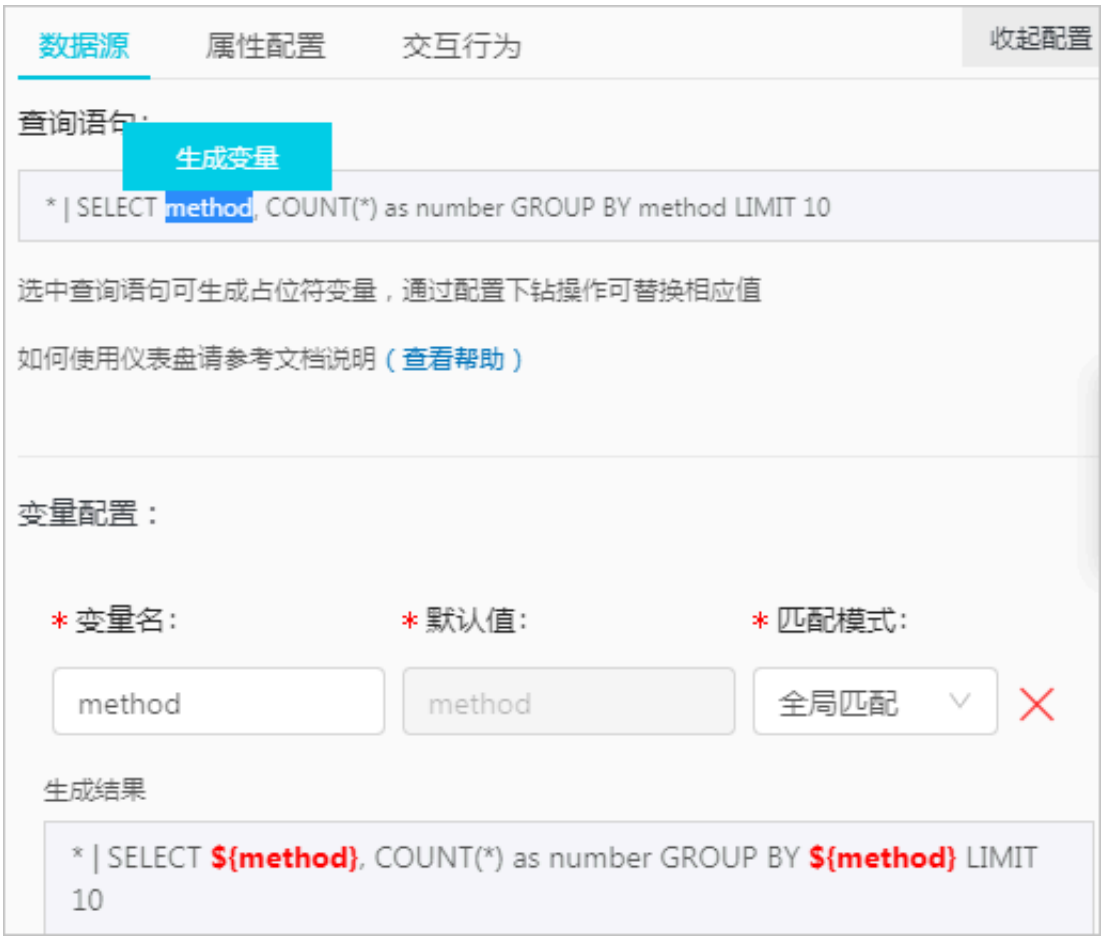
详细信息请查看#unique_57。

a. 进入数据源页签，在查询语句中划选部分查询语句。

b. 单击生成变量，生成占位符变量。

c. 设置变量配置。

配置	说明
变量名	为占位符变量命名。如果占位符变量名称与触发下钻事件的仪表盘图表设置的变量相同时，在下钻事件中占位符变量替换为触发下钻事件的图表值。
默认值	占位符变量在当前仪表盘中的默认值。
生成结果	确认变量配置。



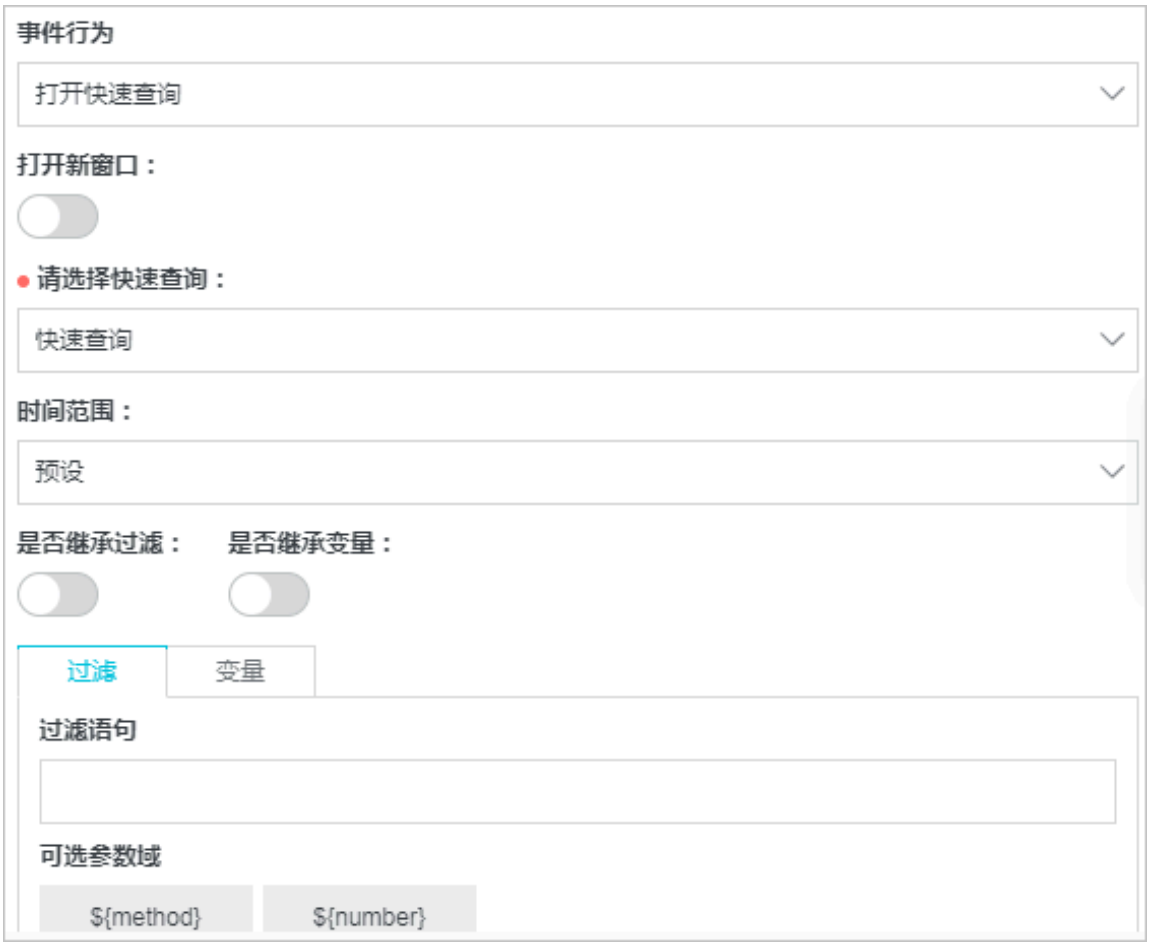
6. （可选）设置#unique_57。

   设置下钻分析后，该图表在仪表盘中可以单击进入更深粒度的查询分析，例如跳转到其他仪表盘、跳转到快速查询等。详细信息请查看#unique_57。

   a. 打开交互行为页签。

   b. 指定开启下钻分析的列，展开对应的列，并设置事件行为。

   c. 填写事件行为对应的设置。



7. 单击添加到仪表盘并指定仪表盘和图表名称。

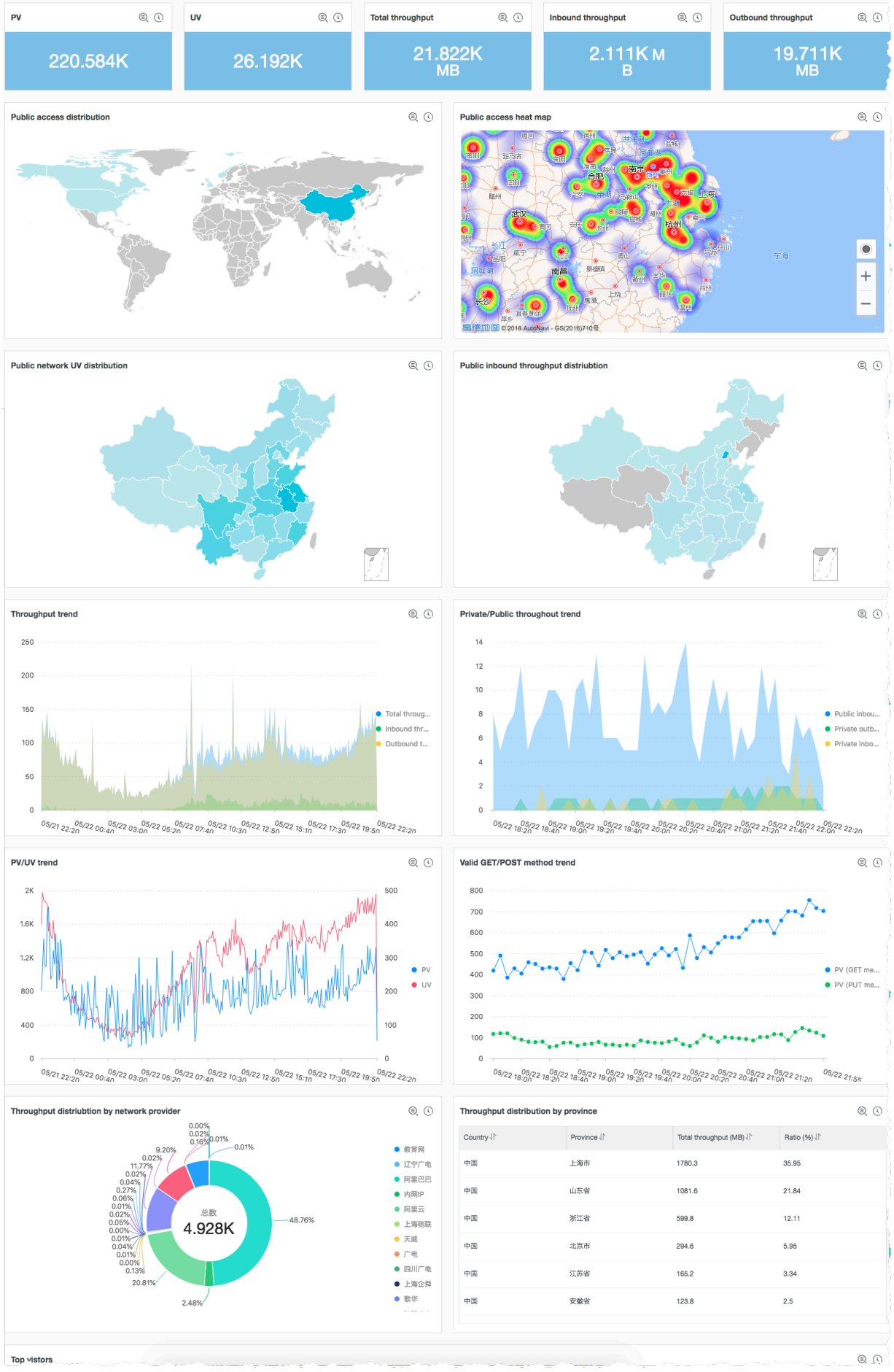
配置	说明
操作类型	· 添加到已有仪表盘：将当前分析图表添加到已有的仪表盘。 · 新建仪表盘：将当前分析图表添加到新建的仪表盘中。
仪表盘列表	选择已有的仪表盘名称。 说明: 仅在操作类型为添加到已有仪表盘时指定。

配置	说明
Dashboard名称	新建的仪表盘名称。 说明: 仅在操作类型为新建仪表盘时指定。
图表名称	为当前分析图表命名。该名称会作为图表标题显示在仪表盘页面中。

8. 单击确定，结束配置。

您可以通过多次添加的方式，将多个分析图表添加到一个仪表盘中。

添加了多个分析图表的仪表盘：


PV
220.584K
UV
26.192K
Total throughput
21.822K MB
Inbound throughput
2.111K MB
Outbound throughput
19.711K MB
Public access distribution
Public access heat map
Public network UV distribution
Public inbound throughput distriubtion
Throughput trend
Total throug...
Inbound thr...
Outbound t...
Private/Public throughout trend
Public inbou...
Private outb...
Private inbo...
PV/UV trend
PV
UV
Valid GET/POST method trend
PV (GET me...
PV (PUT me...
Throughput distriubtion by network provider
总数
4.928K
48.76%
20.81%
2.48%
Throughput distribution by province
Country
Province
Total throughput (MB)
Ratio (%)
中国 上海市 1780.3 35.95
中国 山东省 1081.6 21.84
中国 浙江省 599.8 12.11
中国 北京市 294.6 5.95
中国 江苏省 165.2 3.34
中国 安徽省 123.8 2.5
Top visitors

删除仪表盘

当不需要某个仪表盘时，可以删除仪表盘。删除后不可恢复。

1. 在日志服务控制台单击Project名称。
2. 单击左侧导航栏的仪表盘图标。
3. 单击目标仪表盘后的 图标，选择 删除。
4. 在弹出提示框中单击确认。

12.2.3 显示模式

查看仪表盘时，默认处于显示模式下，显示模式下可以直观、清晰地查看该仪表盘下的所有分析图表。日志服务同时提供一系列针对仪表盘的显示配置，包括添加页面元素、设置自动刷新、设置标题显示方式等。

在日志服务控制台Logstore列表页面左侧导航栏中单击仪表盘，并在仪表盘列表页面单击仪表盘名称，即可进入指定仪表盘。您也可以在查询分析页面、快速查询页面等页面中，单击左侧折叠导航栏中的仪表盘名称进入指定仪表盘。

支持的显示设置

- 仪表盘显示设置

  显示模式下仪表盘功能项主要集中在仪表盘右上方，从左到右依次为：时间选择、编辑、告警、刷新、分享、全屏、标题设置以及重置时间。

- 分析图表显示设置

  显示模式下，可以通过单击图表右上角，展开折叠按钮中的功能列表，获取该图表的详细信息。

  说明:

不同图标类型有不同的展示选项，例如自定义添加的图标、Markdown图表等特殊图表无法查看分析详情，因为它不是一个查询分析图表。

设置仪表盘时间

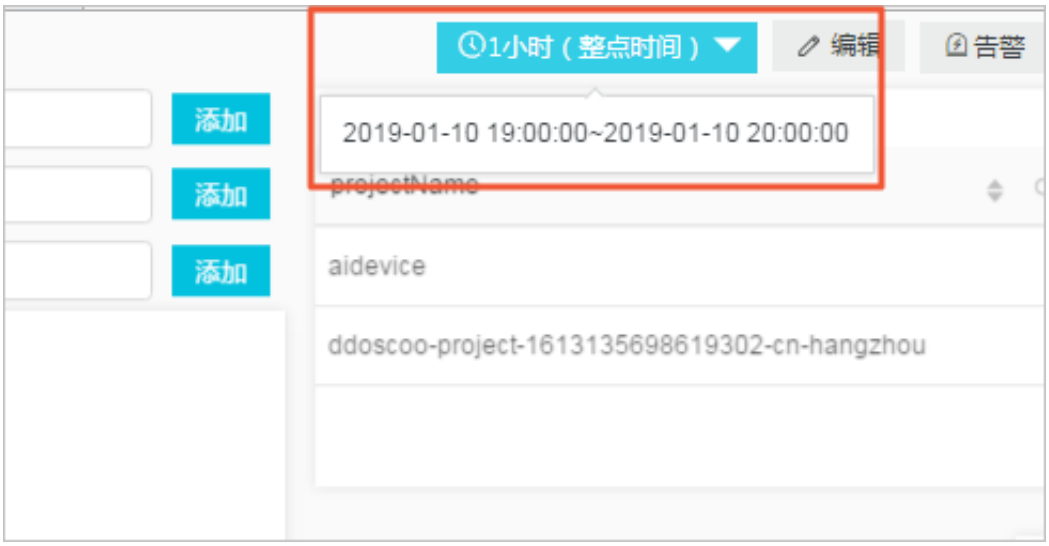
仪表盘时间即仪表盘中所有分析图表统一的时间，设置后，所有分析图表展示的是同一时段的查询分析结果。如果需要设置单个图表的时间范围，请查看设置指定图表的时间范围。

说明:
时间选择器仅在当前页面提供临时的图表查看方式，系统不保存该设置。您下次查看报表时，系统仍会为您展示默认的时间范围。

1. 登录日志服务控制台，单击Project名称。
2. 单击左侧导航栏的仪表盘图标。
3. 单击目标仪表盘后的图标，选择 详情。
4. 单击请选择。
5. 单击选择时间范围，并单击确定。

   支持设置仪表盘时间为：

   - 相对时间：表示查询距离当前时间1分钟、5分钟、15分钟等时间区间的的日志数据。例如当前时间为19:20:31，设置相对时间1小时，表示查询18:30:31~19:20:31的日志数据。
   - 整点时间：表示查询最近整点1分钟、5分钟、15分钟等时间区间的日志数据。例如当前时间为19:20:31，设置整点时间1小时，表示查询18:00:00~19:00:00的日志数据。
   - 自定义时间：表示查询指定时间范围的日志数据。
6. 请选择按钮变更为当前设置的时间范围，鼠标移至按钮上，可核对当前设置的时间范围。



进入编辑模式

单击编辑进入仪表盘的编辑模式，在编辑模式下，支持对仪表盘进行多种变更操作，包括将仪表盘作为画布，为其添加图表元素，如Markdown图表、自定义图表、文本、图标等图表元素等操作。详细说明请参考编辑模式。

设置告警

在仪表盘右上角单击告警 > 新建或告警 > 修改可以新建告警或设置告警，告警中需要关联一个或多个分析图表。

如何设置告警，请查看#unique_59。

设置页面刷新方式

刷新仪表盘时可以选择手动刷新一次或设置自动刷新：

- 单击刷新 > 仅一次，表示立即刷新一次仪表盘。
- 单击刷新 > 自动刷新，表示按照指定间隔自动刷新仪表盘。

  自动刷新可设置为15秒、60秒、5分钟或15分钟。

> **说明:**
> 当浏览器页签处于不活跃状态时，时间计数可能存在偏差。

分享仪表盘页面

单击分享，可以自动复制当前仪表盘的链接，您可以将该链接手动发送给有仪表盘查看权限的其他用户。其他用户看到的仪表盘页面中保留分享者一系列设置，例如图标时间、标题显示方式等。

> **说明:**
> 分享仪表盘页面前，必须赋予其他用户查看该仪表盘的权限。

全屏展示

单击全屏，即可进入全屏模式，页面全屏显示仪表盘中的图表信息。适用于数据展示和报告场景。

设置标题显示方式

可以设仪表盘中所有分析图表标题的显示样式，包括：

- 并排显示标题+时间
- 滚动显示标题+时间
- 渐变显示标题+时间
- 仅名称
- 仅时间

重置时间

单击重置时间可以重置所有图表的时间范围，即恢复所有分析图表的默认时间，多用于改变时间维度后还原到初始状态。

分析图表的查看设置

- 查看分析详情

  展开指定分析图表右上角的折叠列表，并单击查看分析详情，页面会自动跳转到对应的查询分析页面，页面显示当前图表的查询语句和属性设置等信息。
- 设置指定图表的时间范围

  展开指定分析图表右上角的折叠列表，并单击选择时间区间，可以设置指定图表的时间范围。设置后，仅该图表的时间区间会变更，其他图表时间保持不变。
- 设置图表告警

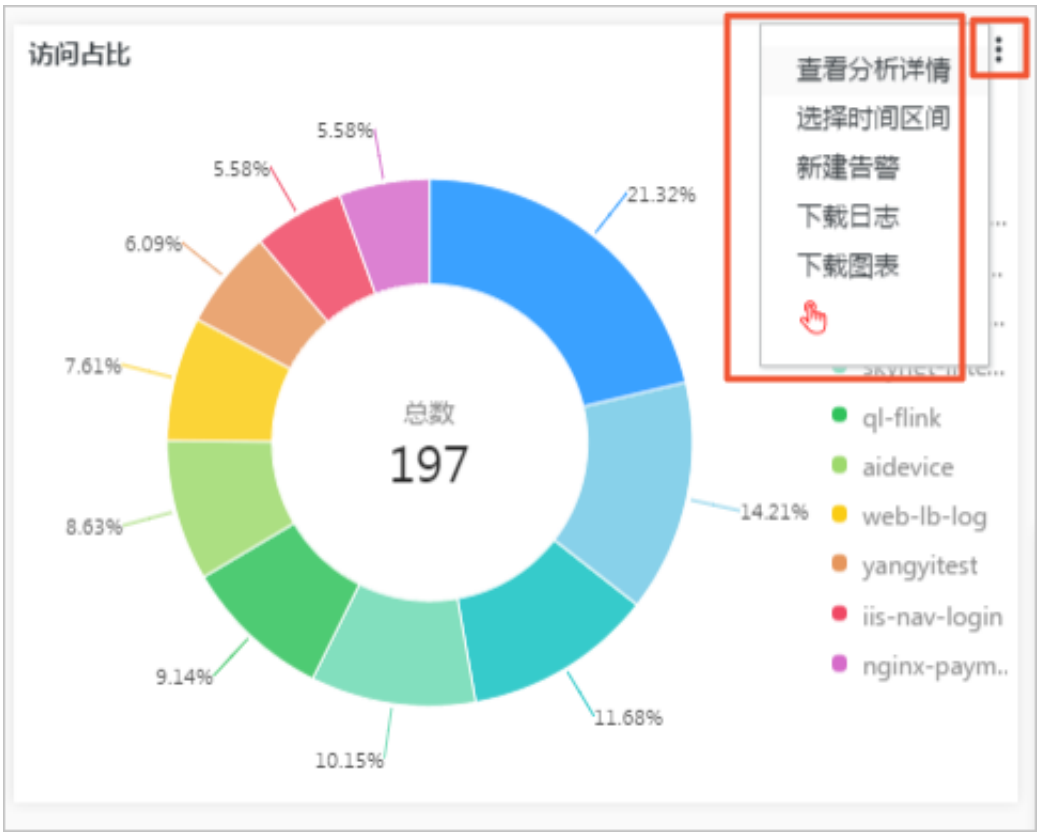
  展开指定分析图表右上角的折叠列表，并单击新建告警，可以设置基于该分析图表的告警和通知方式。如何设置告警，请查看#unique_59。
- 下载日志

  展开指定分析图表右上角的折叠列表，并单击下载日志，以CSV格式下载当前时间区间对应的原始日志分析结果。
- 下载图表

  展开指定分析图表右上角的折叠列表，并单击下载图表，以PNG图片格式下载当前的查询分析图表。

- 查看是否设置了下钻分析

  展开指定分析图表右上角的折叠列表，如果列表中的手指图表为红色，表示当前图表已设置下钻分析；否则图表为灰色。



12.2.4 编辑模式

编辑仪表盘时，仪表盘处于编辑模式下。编辑模式支持对仪表盘进行一系列变更和设置操作。

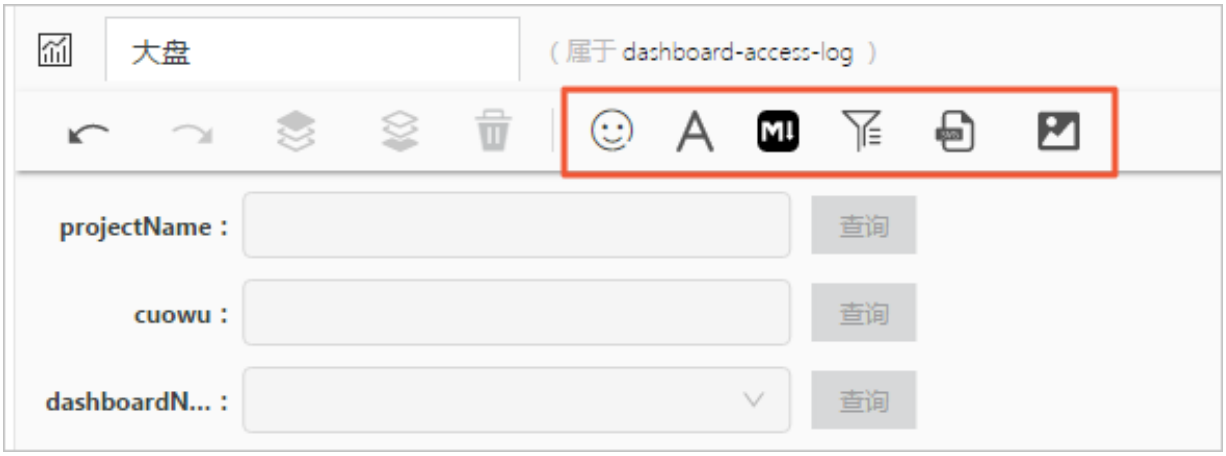
在编辑模式下，支持对仪表盘进行多种变更操作，包括：

- 仪表盘设置：
  - 在页面左上角修改仪表盘名称。
  - 支持将仪表盘作为画布，为其添加图表元素，如Markdown图表、自定义图表、文本、图标等图表元素。
  - 在图表元素之间添加连接线，连接线支持根据图表位置自适应。
  - 添加过滤器，添加后在显示模式下可以过滤图表数据。
  - 为便于排版，可以设置显示网格线，让图标等元素的位置工整有序。
  - 通过菜单栏控制仪表盘中图表的属性设置，例如添加、删除、撤销操作、配置层级和图表大小、位置。
- 图表设置：支持在仪表盘编辑模式下编辑图表，例如修改分析图表的语句、属性、下钻配置等交互行为。

说明:

在仪表盘编辑模式下的所有变更，都必须在页面右上角单击保存才会生效。

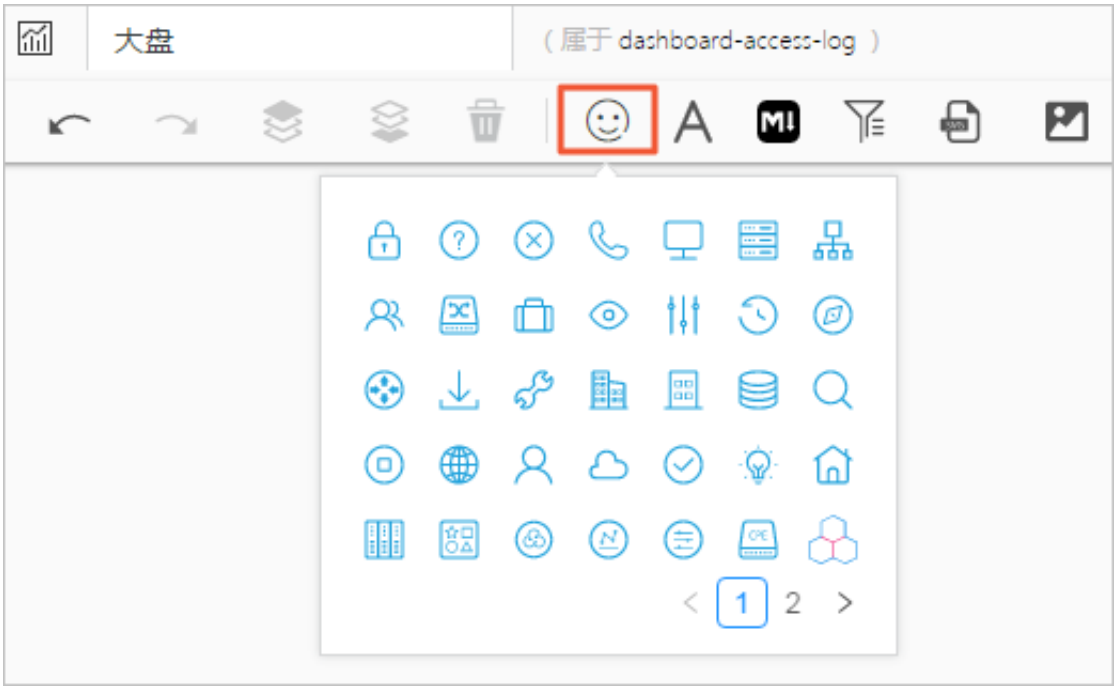
图表元素

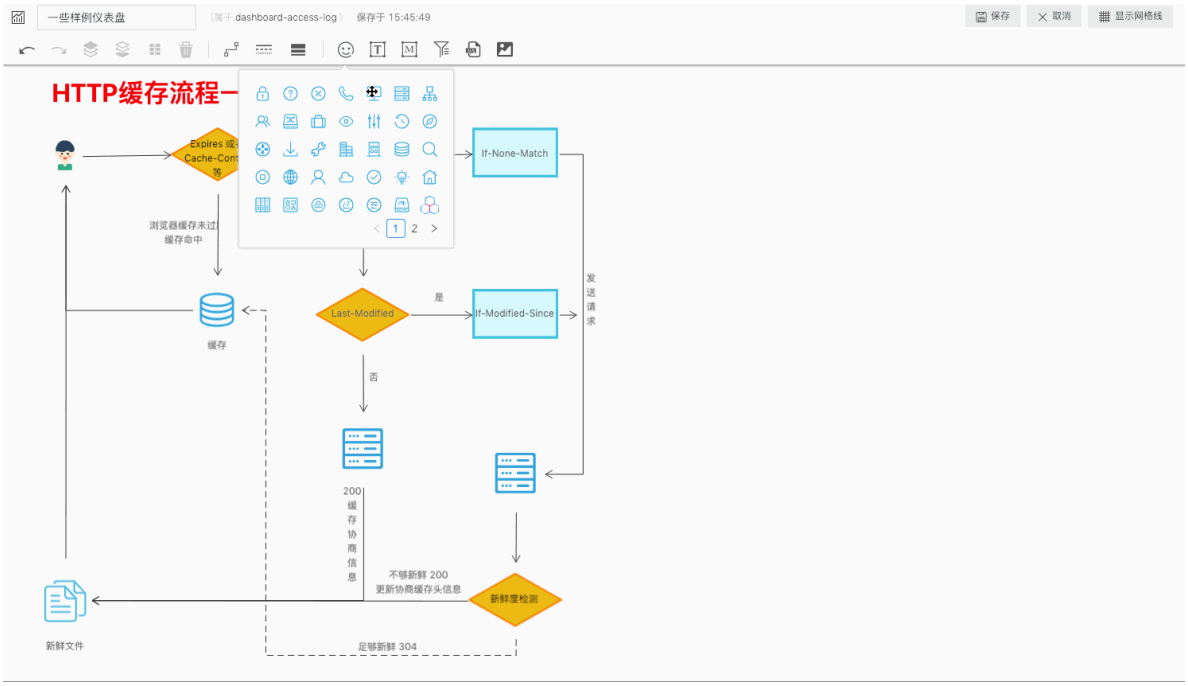


仪表盘编辑模式下，支持插入以下图表元素：

- 常见图标

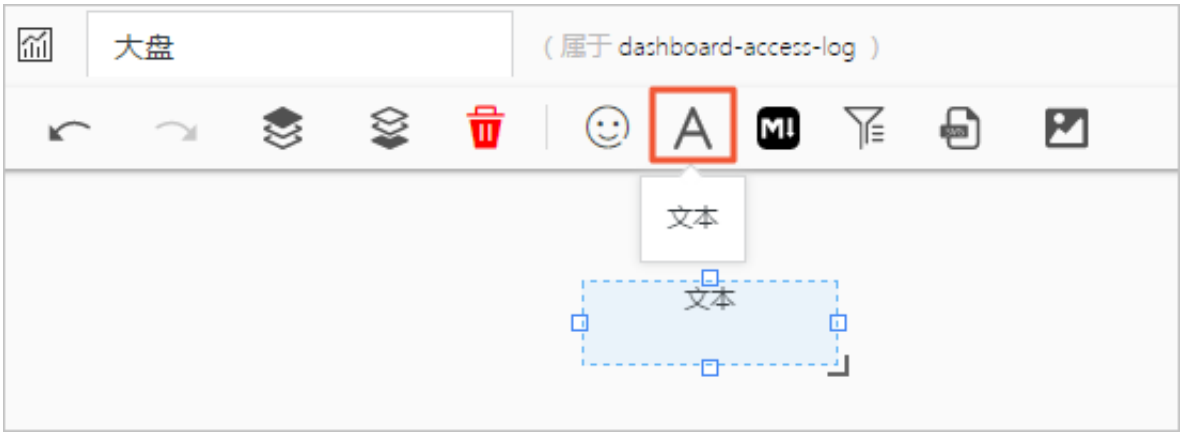
日志服务提供一系列常见图标以便在仪表盘中展示。在菜单栏中拖动该图标到指定位置即可。





- 文本

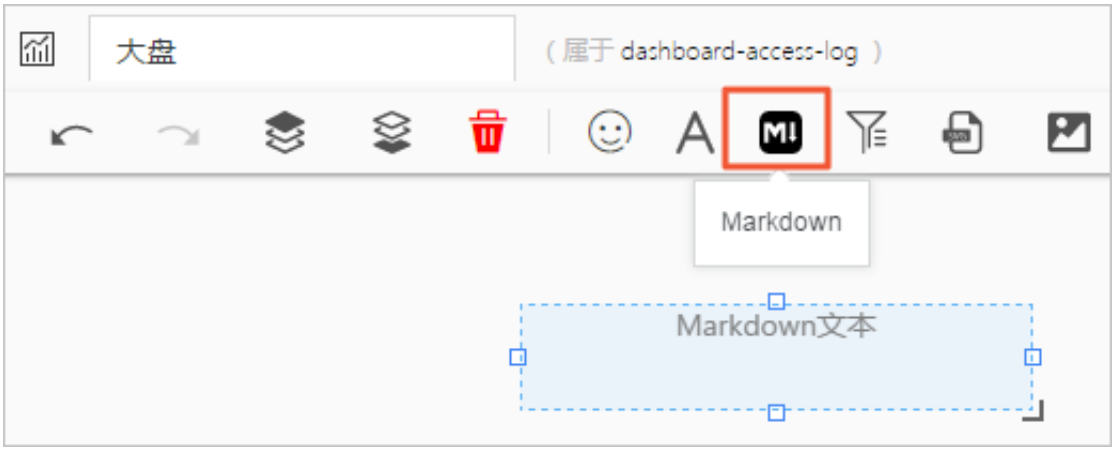
  在仪表盘菜单栏中拖动文本图标到指定位置，可以插入文本。双击文本框可以修改文本内容。



- Markdown图表

  日志服务还支持在仪表盘中增加Markdown图表，该图表使用Markdown语言编辑。

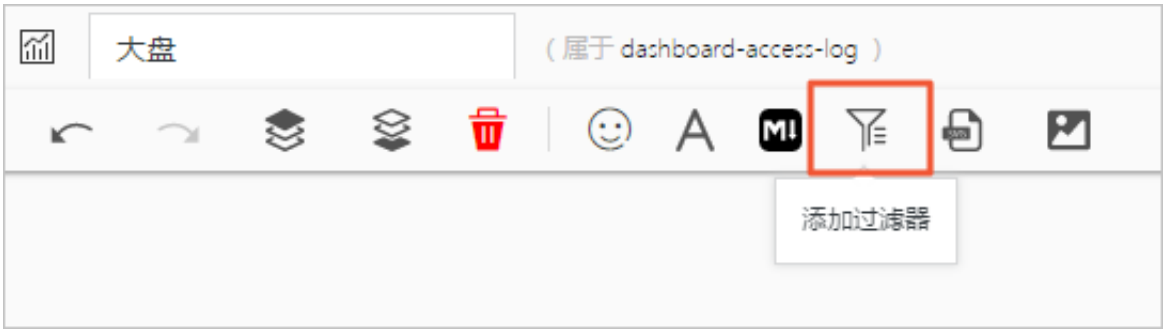
  在仪表盘菜单栏中拖动Markdown图标到指定位置，可以插入Markdown文本。在其右上角的隐藏菜单中单击编辑可以设置Markdown文本内容。



- **过滤器**

  在日志服务仪表盘中增加过滤器配置，可以过滤器缩小查询范围或替换占位符变量，即对整个仪表盘进行查询过滤（Filter）和变量替换（Variables）操作。

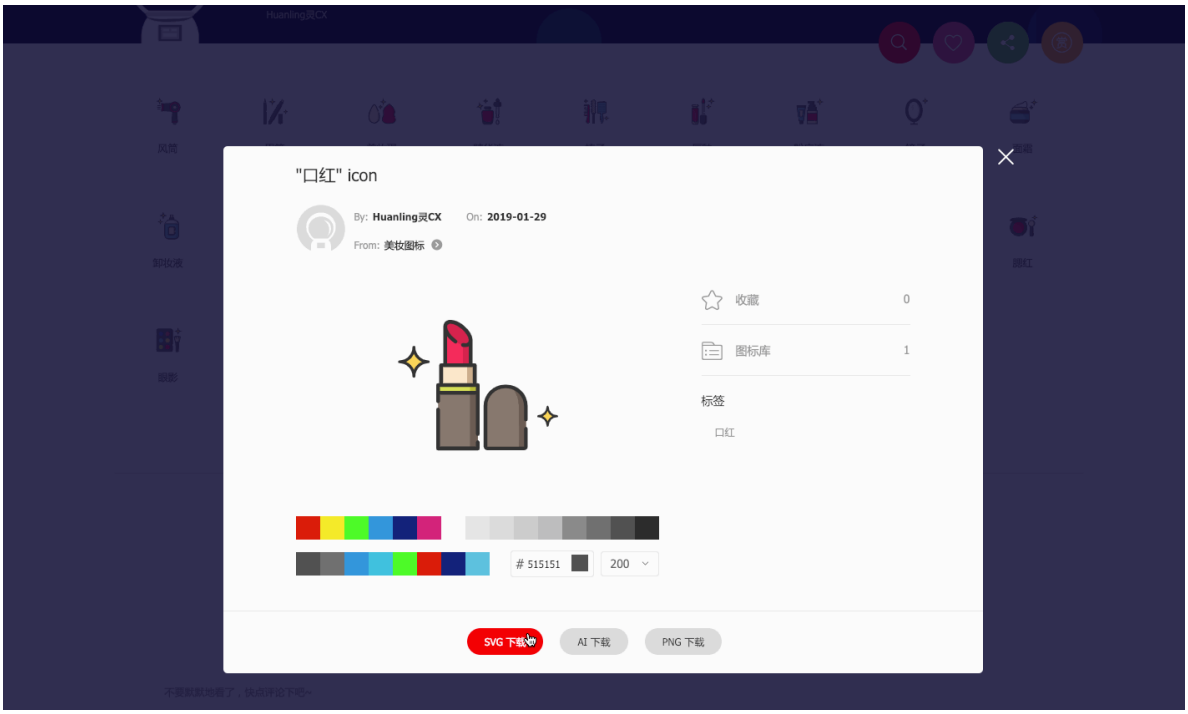
  在仪表盘菜单栏单击过滤器图标，在弹出页面中设置过滤器，仪表盘过滤器默认位置为仪表盘左上角。在其右上角的隐藏菜单中单击编辑可以修改过滤器设置。



- 自定义SVG

  支持直接上传SVG到仪表盘。在操作栏中单击SVG图标，单击或拖拽上传SVG即可。

  说明:
  SVG大小不超过10 KB。



- 自定义HTTP图片

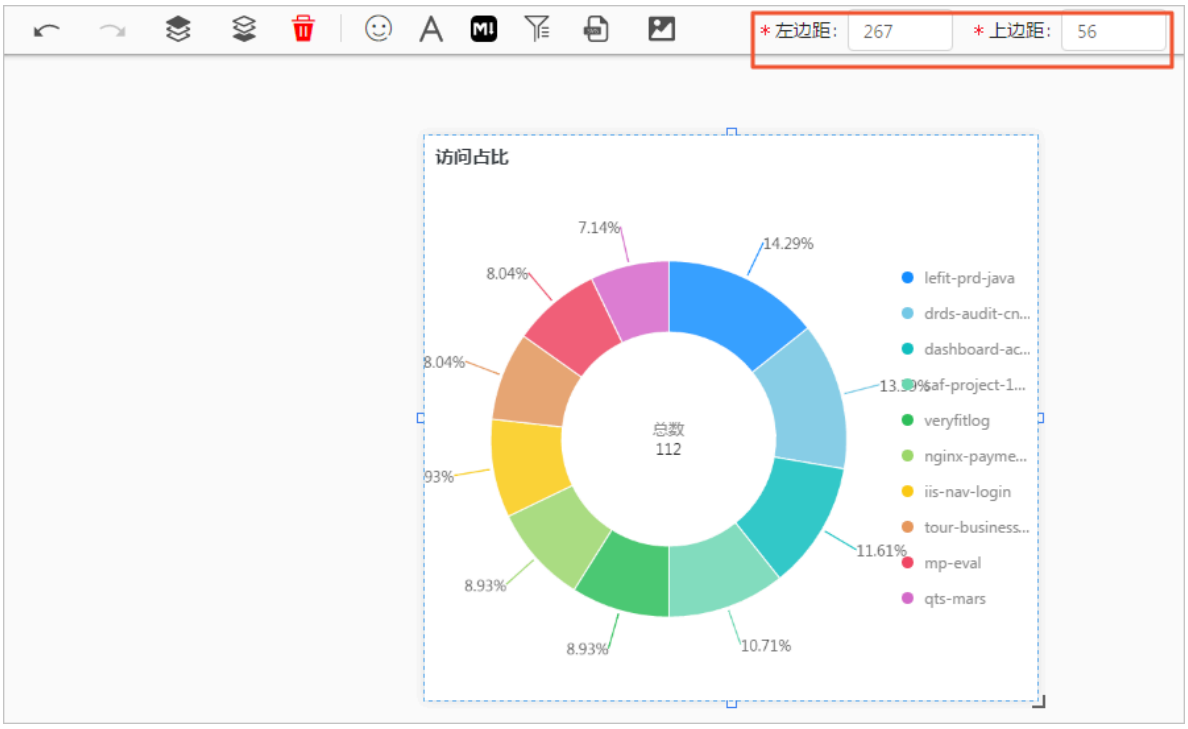
  支持直接上传HTTP图片到仪表盘。在操作栏中单击对应图标，输入图片的HTTP链接，并单击确定即可。

排版布局

在仪表盘编辑模式下，所有的分析图表与各类图表元素都被至于一个可以随意进行拖拽的网格画布中，您可以自由地对每一个图表进行拖动和缩放（连接线除外）。画布水平方向限制为1000个单位，每个单位宽度为当前浏览器宽度 / 1000，垂直方向无限制，每个单位为1像素。排版前，可以在右上角单击显示网格线，网格线便于设置图表的位置和间距。

支持进行以下排版操作：

- 调整图表位置
  - 直接拖动图表到指定位置。
  - 选中指定图表后，在操作栏中通过设置左边距和上边距，调整图片的位置。

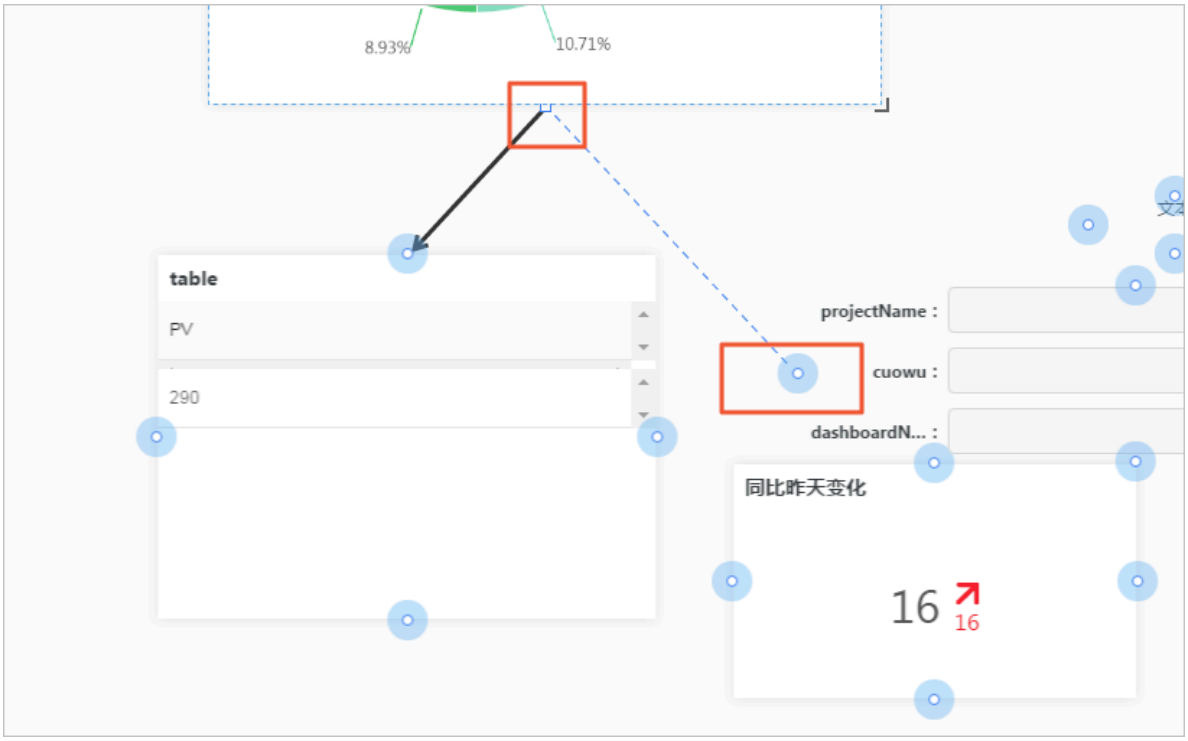


- **调整图表宽度、高度**
  - **选中指定图表后，在右下角拖拽，调整图表大小。**
  - **选中指定图表后，在操作栏中通过设置宽度和高度，设置图表的大小。**

- 添加图表连接线

  在图表之间添加带方向的连接线后，调整表格的位置和大小时，连接线会同时移动，便于展示图表间的相对关系。

  选中图表后，长按其边框中的方框标识，此处为连接线的起点，页面会自动展示可作为连接线终点的区域，将鼠标移动至该位置即可。



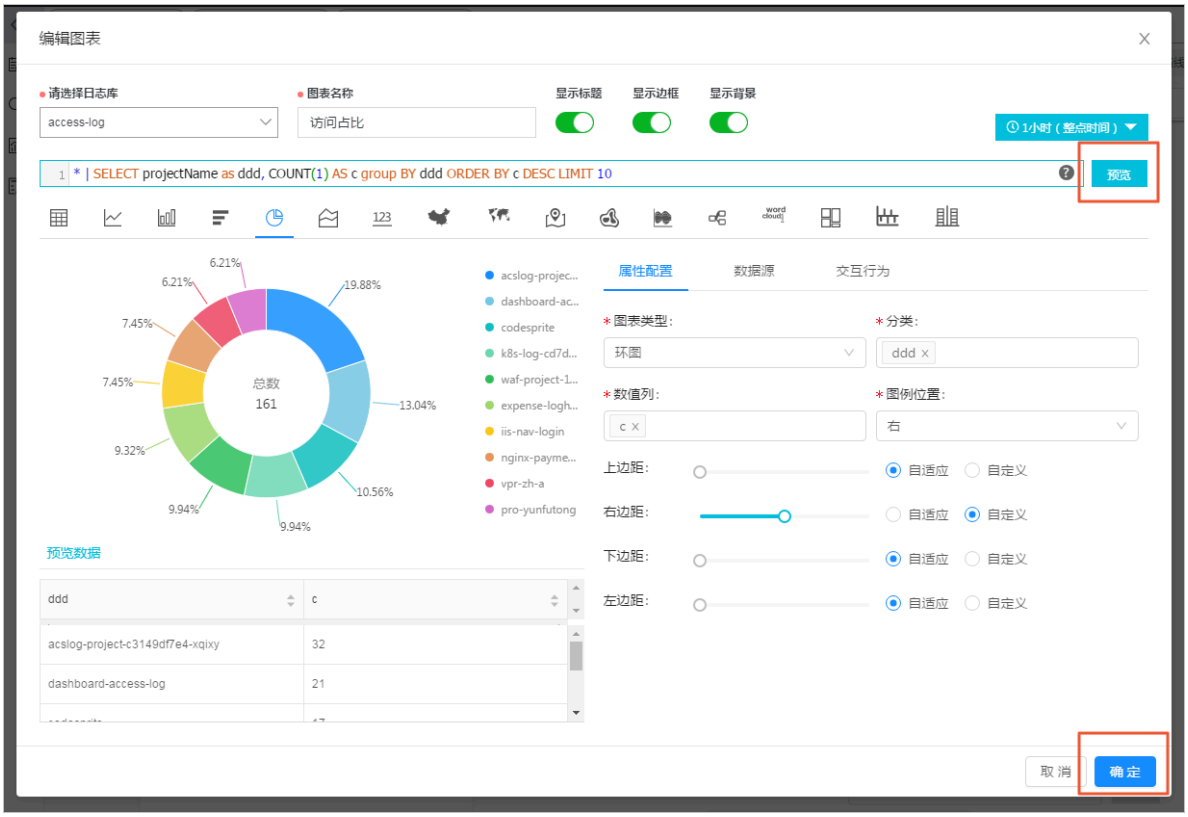
- 图表间支持设置层级关系，选中指定图表后，通过操作栏可将图表层级设置为置顶或置底。

图表设置

在仪表盘编辑模式下，支持对图表元素进行以下操作：

- 编辑：修改分析图表的语句、属性、下钻配置等交互行为。

  1. 在仪表盘页面右上角单击编辑。
  2. 在图表右上角展开隐藏菜单，并单击编辑。
  3. 修改分析图表的查询语句、属性配置、数据源信息或交互行为。
  4. 单击预览，并单击确定。
  5. 在仪表盘页面右上角单击保存。



- 复制：创建指定图表元素的副本，保留所有配置信息。

  1. 在仪表盘页面右上角单击编辑。
  2. 在图表右上角展开隐藏菜单，并单击复制。
  3. 拖动图片副本到指定位置，设置边距和大小。
  4. 在仪表盘页面右上角单击保存。

- 删除：从仪表盘中删除指定图表元素。

  1. 在仪表盘页面右上角单击编辑。
  2. 在图表右上角展开隐藏菜单，并单击删除。
  3. 在仪表盘页面右上角单击保存。

12.2.5 订阅仪表盘

日志服务仪表盘提供了丰富的可视化组件，方便您基于日志查询分析结果轻松构建自己的仪表盘。现在日志服务仪表盘订阅功能支持用户订阅仪表盘，即定期将仪表盘渲染为图片，通过邮件或者钉钉群消息的方式发送给指定对象。

限制说明

- 每个仪表盘只能创建一个订阅任务。
- 每个账户每天最多发送50个邮件。
- Cron表达式最小单位为分钟，但建议设置间隔为1小时以上。
- 每个Project 订阅和告警的总个数不能超过100个。如果有特殊需求，请提工单申请调整限额。
- 如果表格数据分页显示，订阅仪表盘时，仅支持发送表格第一页的数据截图。

说明:

- 显示模式下所有的时间选择都是临时的，仅供用户动态查阅不同时间段的图表数据。
- 仪表盘图表的默认时间可以通过进入编辑模式后，单击图表的编辑按钮进行修改。
- 订阅的仪表盘的数据查询时间为仪表盘中图表的查询时间。

创建仪表盘订阅任务

1. 登录日志服务控制台，单击Project名称。
2. 在左侧导航栏中单击仪表盘图标。
3. 单击仪表盘名称，进入指定仪表盘。
4. 在页面右上角单击订阅。
5. 设置订阅任务，并单击下一步。

设置	说明	示例
订阅名称	订阅任务的名称，1~64字。	仪表盘订阅

设置	说明	示例
频率	订阅仪表盘后，通知消息的发送频率。 支持设置为： · 每小时 · 每天 · 每周 · 固定间隔（天） · 通过Cron表达式指定时间间隔 其中，Cron表达式最小单位为分钟，但建议设置间隔为1小时以上。	指定频率为Cron表达式`* 0/1 * * *`，表示从0点开始，每隔1小时发送一次。
添加水印	在生成的报表图片上添加水印，水印内容为通知渠道地址，即邮箱地址或钉钉机器人WebHook中的access_token。	-

6. 设置通知方式。

仪表盘订阅的通知方式可以设置为邮件和WebHook-钉钉机器人。

- 邮件

在收件人中填写邮箱地址，并设置邮件主题即可。如果没有设置主题，日志服务将使用默认的主题日志服务报表。

邮件示例：

- **WebHook-钉钉机器人**

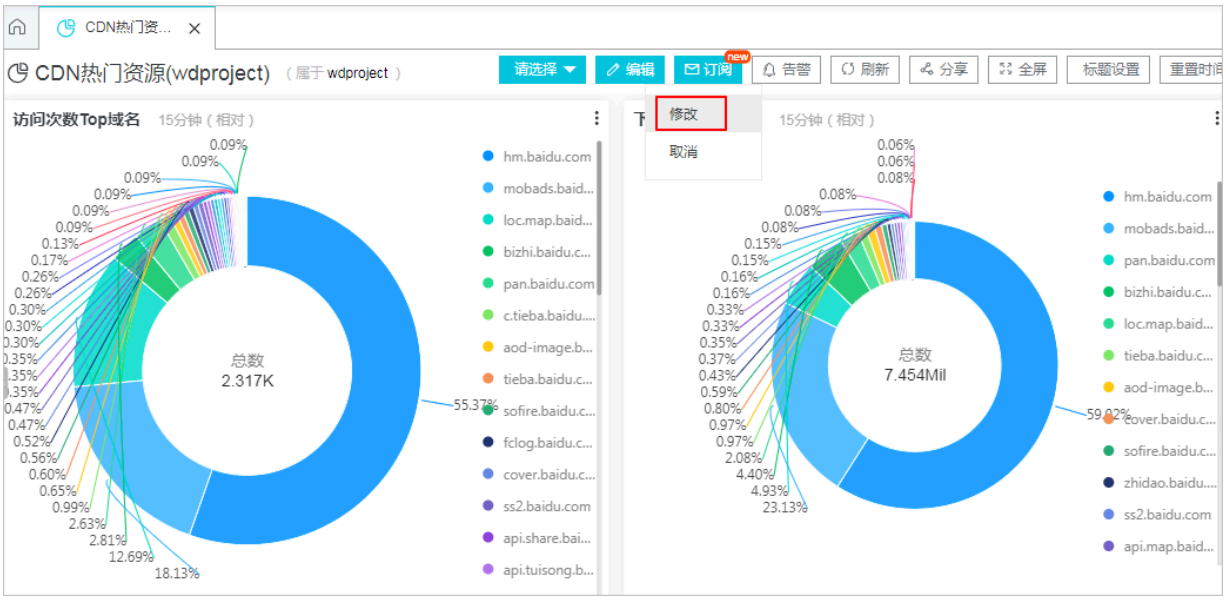
  在请求地址中填写钉钉机器人的WebHook地址即可。如何获取该地址，请查看自定义机器人。

钉钉机器人消息示例：

修改和取消订阅

如果设置了仪表盘订阅，则在该仪表盘页面右上角单击订阅，即可修改或取消仪表盘订阅。

取消订阅后，不再向指定渠道发送订阅消息。



12.2.6 下钻分析

日志服务分析图表除了提供最基本的数据可视化能力之外，还提供了向下钻取（drill down）的功能，您可以在添加一个图表到仪表盘的时候，通过改变下钻列表中的各个配置项，从而使得仪表盘中的分析图表具备更强大的功能。

钻取是在数据分析中不可缺少的功能之一，通过改变展现数据维度的层次、变换分析的粒度从而关注数据中更详尽的信息。它包括向上钻取（roll up）和向下钻取（drill down）。上钻是沿着维度的层次向上聚集汇总数据，下钻是在分析时加深维度，对数据进行层层深入的查看。通过逐层下钻，数据更加一目了然，更能充分挖掘数据背后的价值，及时做出更加正确的决策。

日志服务支持对仪表盘中分析图表的下钻分析，设置下钻的维度和层次后，可以在仪表盘中通过鼠标点击数据点跳转到更深维度的分析页面。仪表盘中的分析图表实际上是查询语句的结果，如果为请求状态表格设置下钻分析、并添加到仪表盘，在仪表盘中单击某个请求状态类型，可以查看请求状态为特定类型的日志信息。

限制说明

日志服务中，支持下钻分析的图表包括：

- 表格
- 线图
- 柱状图
- 条形图
- 饼图
- 单值图
- 面积图
- 矩形树图

前提条件

1. 已开启并配置索引。
2. 已配置要跳转到的快速查询、仪表盘和自定义链接。
3. 如果选择添加变量，则需要在跳转到的快速查询和仪表盘配置中配置查询语句变量占位符。详情请参考快速查询和#unique_56。

配置步骤

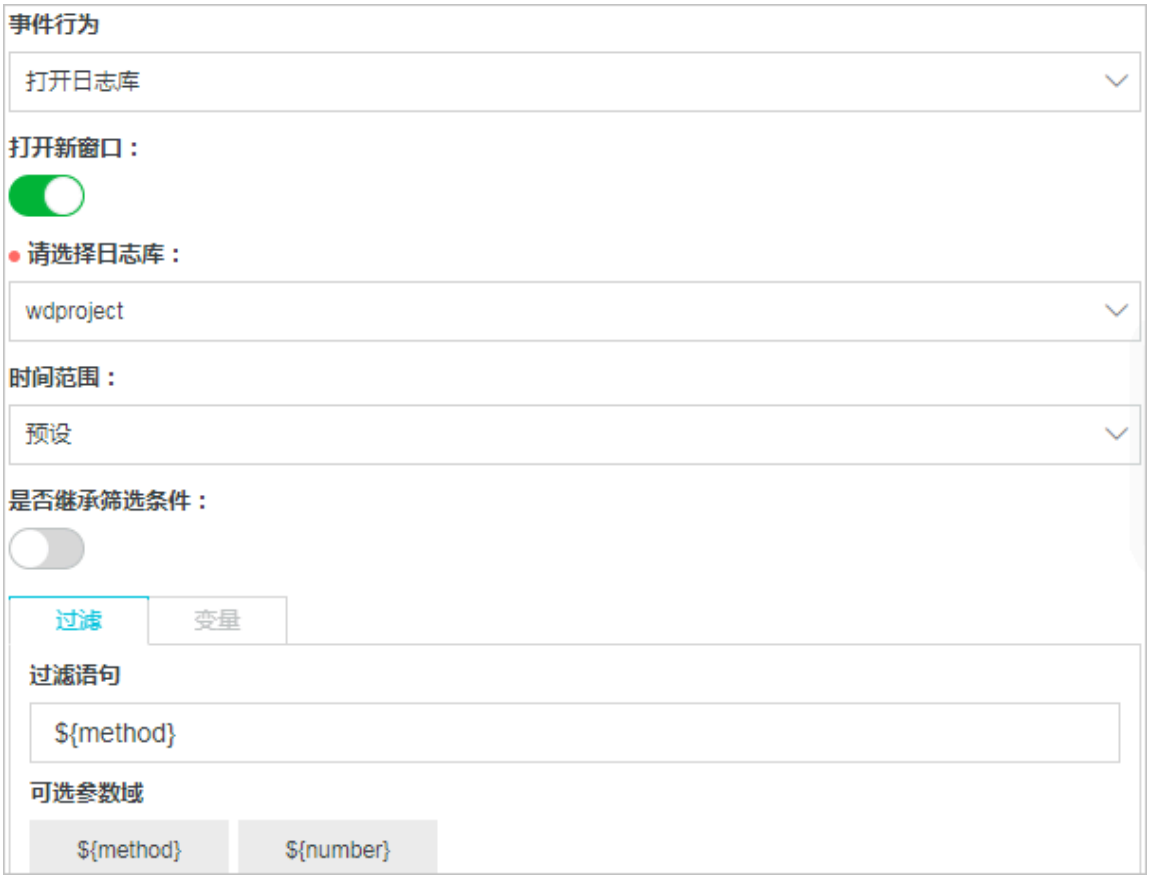
1. 登录日志服务控制台，单击Project名称。
2. 单击日志库名称后的 图标，选择查询分析。
3. 输入您的查询分析语句，设置时间范围，并单击查询/分析。

4. 在统计图表页签中选择图表类型，并设置属性配置。
5. 在交互行为页签中设置下钻事件行为。

   下钻事件行为指在仪表盘页面中单击分析图表而触发的事件，默认为关闭状态。设置下钻事件后，在仪表盘中单击这张图表中的数据，根据您配置的事件行为，自动跳转到对应页面。您可以选择以下5种配置。

   - 不开启：表示不开启下钻功能。
   - 打开日志库：表示开启下钻功能，下钻事件为打开日志库。

     单击图表内容时，如果设置了过滤，会自动为跳转到的日志库增加查询语句。暂不支持设置变量。



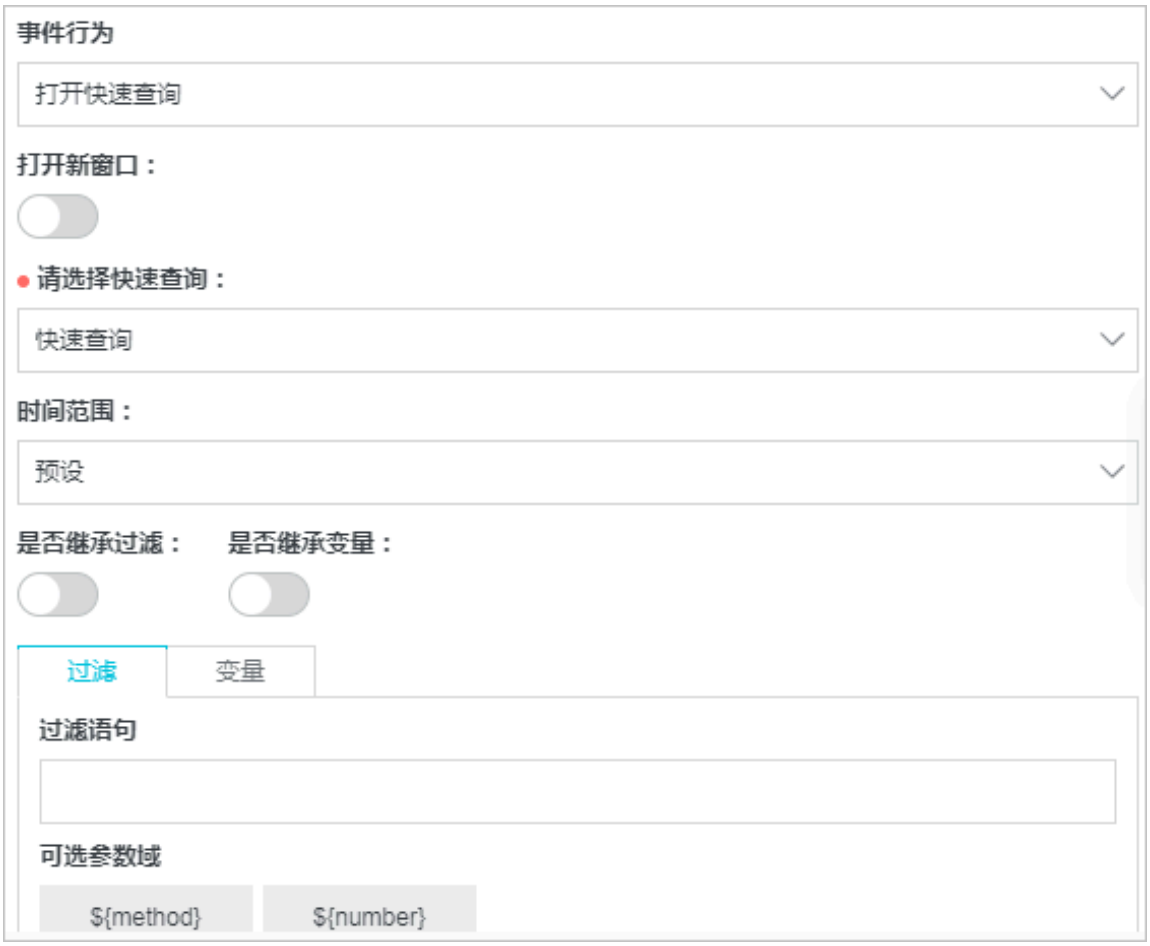
配置	说明
请选择日志库	需要跳转到的日志库名称。如何创建日志库请参考#unique_211 。
打开新窗口	开启该选项后，当触发交互行为时将在新窗口打开对日志库。

配置	说明
时间范围	设置跳转到的日志库的查询分析时间范围。可以设置为： - 预设：仪表盘页面中单击图表跳转到日志库后，保持快速查询的默认时间范围，即15分钟（相对）。 - 继承图表时间：跳转后，日志库的查询语句对应的时间范围默认为触发事件时仪表盘中设置的图表的时间。 - 相对时间：跳转后，将跳转后日志库的快速查询时间设置为指定的相对时间。 - 整点时间：跳转后，将跳转后日志库的快速查询时间设置为指定的整点时间。 默认为预设。
是否继承筛选条件	如果选择继承筛选条件，则会把触发事件仪表盘中添加的筛选条件同步到对应日志库的快速查询中，并以AND的方式添加到查询语句之前。

配置	说明
过滤	在过滤页签中输入过滤语句，语句中可以包含可选参数域。 如果配置了过滤，在仪表盘图表中单击跳转后，会自动为跳转到的日志库的快速查询增加查询语句，查询语句为此处配置的过滤语句。

- 打开查询页面：表示开启下钻功能，下钻事件为打开查询页面。

  单击图表内容时，如果设置了变量，会用单击的图表值替换快速查询语句中设置的占位符，基于图表值进行更深层次的查询；如果设置了过滤，会自动为跳转到的快速查询增加查询语句。支持同时设置变量和占位符。



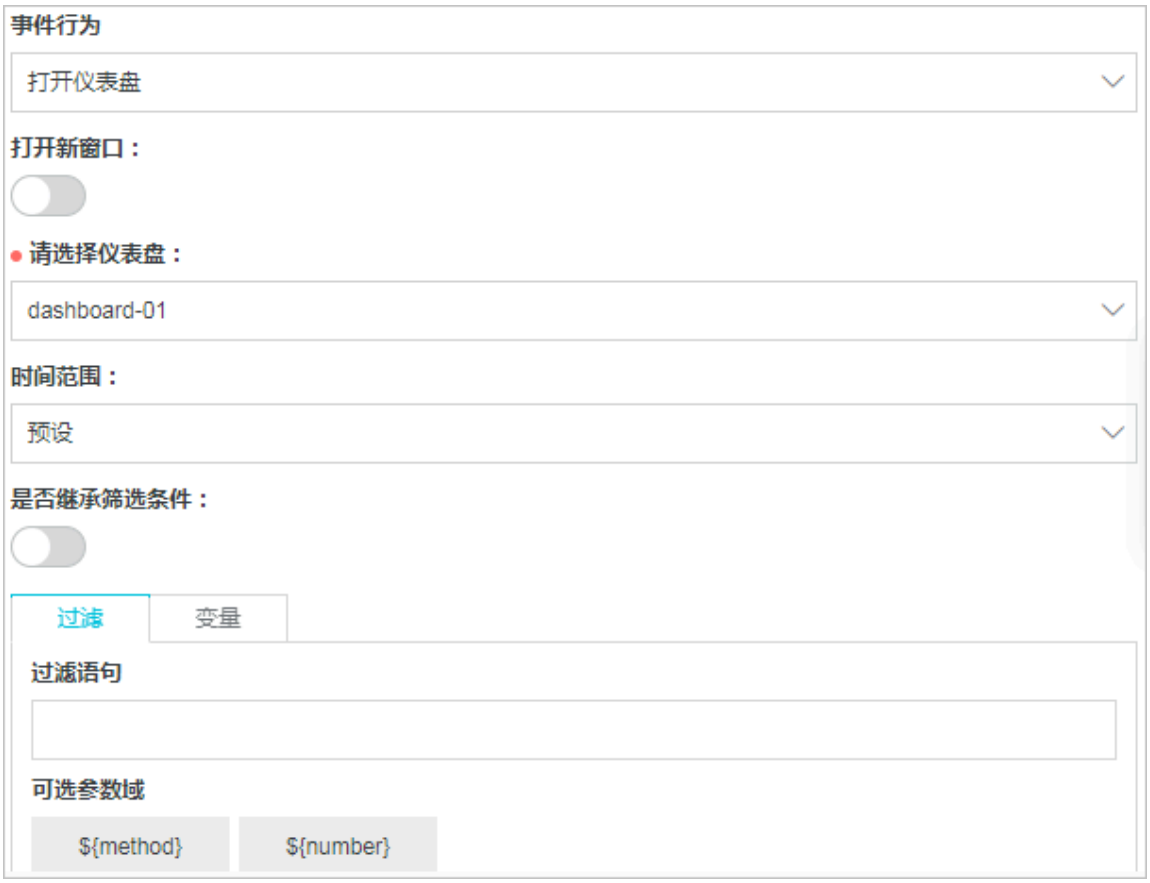
配置	说明
请选择快速查询	需要跳转到的快速查询名称。如何配置快速查询请参考快速查询 。
打开新窗口	开启该选项后，当触发交互行为时将在新窗口打开对应快速查询。

配置	说明
时间范围	设置跳转到的快速查询的时间范围。可以设置为： - 预设：仪表盘页面中单击图表跳转到快速查询后，保持快速查询的默认时间范围，即15分钟（相对）。 - 继承图表时间：跳转后，查询语句对应的时间范围默认为触发事件时仪表盘中设置的图表的时间。 - 相对时间：跳转后，将跳转后的快速查询时间设置为指定的相对时间。 - 整点时间：跳转后，将跳转后的快速查询时间设置为指定的整点时间。 默认为预设。
是否继承筛选条件	如果选择继承筛选条件，则会把触发事件仪表盘中添加的筛选条件同步到快速查询中，并以AND的方式添加到查询语句之前。
过滤	在过滤页签中输入过滤语句，语句中可以包含可选参数域。 如果配置了过滤，在仪表盘图表中单击跳转后，会自动为跳转到的快速查询增加查询语句，查询语句为此处配置的过滤语句。
变量	在变量页签中单击添加变量，并指定： - 替换变量名：触发下钻分析的变量，单击即可跳转。 - 替换值所在列：以指定列的对应值进行替换。当有多列时，可以设置为当前列和其他列。当前列为设置下钻的列，即替换值所在列所在的列；其他列可以是设置下钻分析的图表中其他任意列。 当跳转到的快速查询中的查询语句变量和本次添加的变量名称一致时，会将快速查询语句中的变量替换为触发下钻事件的图表值，从而灵活改变目标快速查询中的查询语句。 说明： - 如果选择添加变量，则需要事先在跳转到的快速查询中配置查询语句变量占位符。 - 最多可以添加5个变量。

· 打开仪表盘：表示开启下钻功能，下钻事件为打开仪表盘。

仪表盘中的图表实际上是查询语句的图表形式的结果。单击上层仪表盘中的图表内容时，如果设置了变量，且预先在跳转到的仪表盘图表查询语句中设置了占位符，会用单击的图表值

替换预设的占位符；如果设置了过滤，会为跳转到的仪表盘增加过滤条件，基于图表值进行更深层次的查询。



配置	说明
请选择仪表盘	需要跳转到的目标仪表盘名称，如何配置仪表盘请参考#unique_56。
打开新窗口	开启该选项后，当触发交互行为时将在新窗口打开对应仪表盘。
时间范围	设置跳转到的仪表盘的时间范围。可以设置为： - 预设：仪表盘页面中单击图表跳转到仪表盘后，跳转到的仪表盘时间范围保持不变，即保留所有图表的预设时间。 - 继承图表时间：跳转后，仪表盘中图表对应的时间范围默认为触发事件时仪表盘中设置的图表的时间。 - 相对时间：跳转后，将跳转后的仪表盘时间设置为指定的相对时间。 - 整点时间：跳转后，将跳转后的仪表盘时间设置为指定的整点时间。 默认为预设。
是否继承筛选条件	如果选择继承筛选条件，则会把触发事件仪表盘中添加的筛选条件同步到跳转到的仪表盘中，并以AND的方式添加到查询语句之前。

配置	说明
过滤	在过滤页签中输入过滤语句，语句中可以包含可选参数域。 如果配置了过滤，在仪表盘图表中单击跳转后，会自动为跳转到的仪表盘添加过滤条件，过滤条件为此处配置的过滤语句。
变量	在变量页签中单击添加变量，并指定： - 替换变量名：触发下钻分析的变量，单击即可跳转。 - 替换值所在列：以指定列的对应值进行替换。当有多列时，可以设置为默认列和其他列。默认列即当前列，也就是设置下钻分析的列；其他列可以是设置下钻分析的图表中其他任意列。 当跳转到的仪表盘中的分析图表查询语句变量和本次添加的变量名称一致时，会将分析图表查询语句中的变量替换为触发下钻事件的图表值，从而灵活改变目标仪表盘中分析图表的查询语句。 说明: - 如果选择添加变量，则需要事先在跳转到的仪表盘中配置查询语句变量占位符。 - 最多可以添加5个变量。

- 自定义http链接：表示开启下钻功能，下钻事件为打开自定义http链接。

  http链接中的路径部分表示访问的目的端文件的层级路径，您可以在定义http链接的路径部分添加可选参数域，单击仪表盘中的图表内容时，会用图表值替换http链接中的参数，跳转到重新定位的http链接中。

配置	说明
请输入链接地址	需要跳转到的目标地址。

配置	说明
可选参数域	单击可选参数变量，可以将链接地址中的某一部分替换为触发下钻事件的图表值。

6. 单击添加到仪表盘，配置仪表盘，并单击确定。

后续您可以在仪表盘页面中查看该分析图表，单击图表即可查看更深层次的分析结果。

示例

例如，在名为accesslog的Logstore中存放采集到的Nginx访问日志，名为RequestMethod的仪表盘中展示Nginx日志的常见分析场景，名为destination_drilldown的仪表盘展示PV随时间分布的趋势。您可以为请求方法的分类表格设置下钻分析，并将其添加到RequestMethod仪表盘中，并将下钻事件设置为跳转到destination_drilldown仪表盘。在RequestMethod仪表盘中单击各个请求方法即可跳转到destination_drilldown仪表盘查看对应的PV趋势。

流程如下：

1. 设置跳转到的仪表盘（destination_drilldown）。

   a. 根据请求类型筛选日志，并查看PV随时间的变化。

   查询语句：

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

   b. 通过折线图表示查询结果，并将折线图保存到仪表盘中。

   保存到仪表盘时，将`*`设置为占位符，并命名为method，如果跳转到这个快速查询的下钻事件变量同样为method，即可用单击的图表值替换`*`，再次执行查询分析。

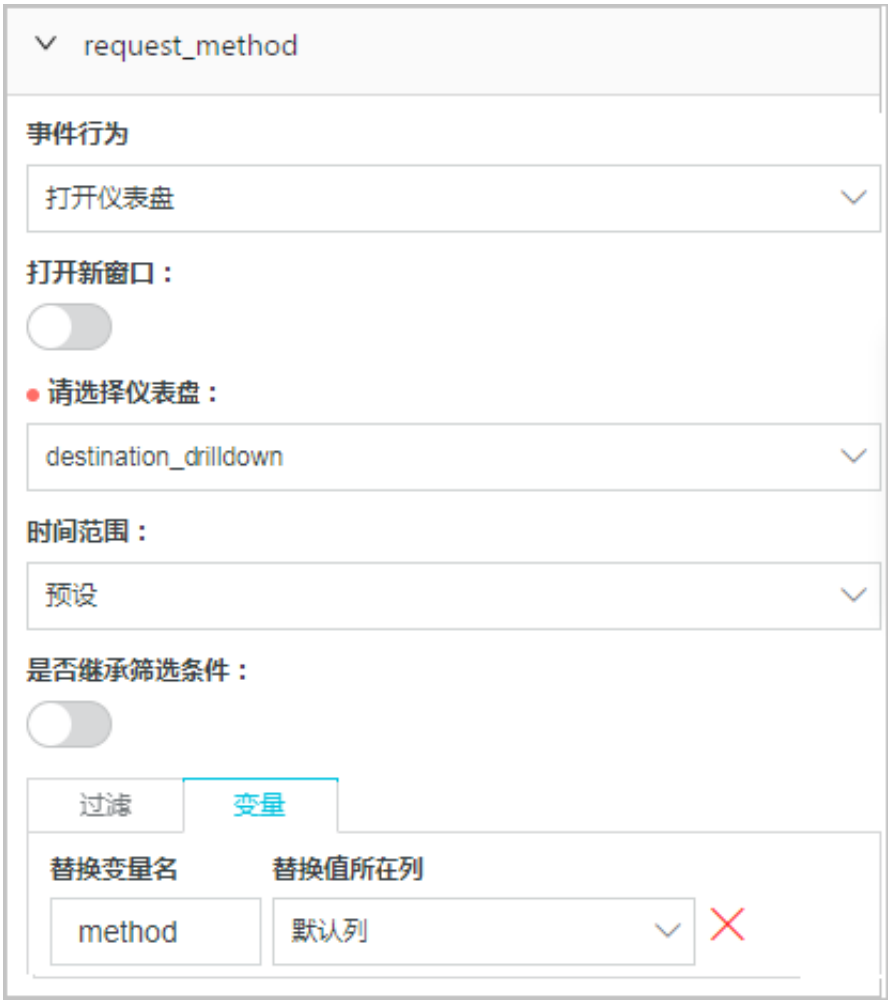
2. 设置触发下钻分析的图表，并将其添加到仪表盘（RequestMethod）。

   a. 在查询页面通过SQL语句分析Nginx访问日志中各种请求方法（request_method）的日志条数，并将结果以表格形式表示。

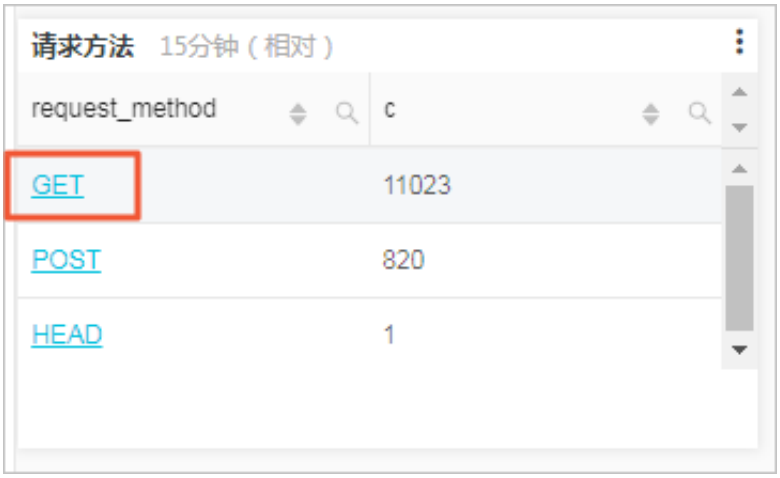
```
*|SELECT request_method, COUNT(1) AS c GROUP BY request_method
ORDER BY c DESC LIMIT 10
```

   查询结果：

   b. 为`request_method`一列设置下钻分析：



3. 在RequestMethod仪表盘中单击GET请求。



4. 成功跳转到destination_drilldown仪表盘。

   页面自动跳转到1中设置的仪表盘，原查询语句中的`*`已替换为单击的图表值`GET`，表示查看GET请求PV随时间的变化。

12.2.7 仪表盘过滤器

在日志服务仪表盘中增加过滤器配置，可以缩小查询范围或替换占位符变量，即对整个仪表盘进行查询过滤（Filter）和变量替换（Variables）操作。

日志服务仪表盘中的每一张图表是一个查询分析语句，在仪表盘中增加过滤器也就是为所有图表批量增加过滤条件，或者批量替换所有图表中设置的占位符变量。过滤器配置分为以下两种：

- 过滤器类型：指定key和value，并将其作为过滤条件增加到查询语句`[serch query]`前。新的查询语句为`key: value AND [serch query]`，表示在原查询语句的结果中，查找包含`key:value`的日志。

  过滤器类型的过滤器中，value可以多选，也可以直接输入。多选时，过滤条件之间为`or`关系。
- 变量替换类型：指定变量占位符，如果仪表盘中有已设置该变量占位符的图表，则将图表查询语句中的该占位符变量替换为选择的value值。

基本构成

每个过滤器图表可以有一个或者多个过滤器构成，每个过滤器主要包含以下元素：

- 过滤器操作Key值。
- Key对应的列表项。

前提条件

1. 已#unique_4。
2. 已创建仪表盘，如果过滤器类型为变量替换，需要设置好变量占位符。

操作步骤

1. 登录日志服务控制台，单击Project名称。
2. 单击左侧导航栏的仪表盘图标。
3. 在仪表盘列表中单击指定仪表盘名称。
4. 在仪表盘页面右上角单击编辑，进入编辑模式。
5. 单击过滤器图标 ，并设置过滤器配置。完成后单击确定。

表 12-2: 过滤器图表配置项

配置项	说明
图表名称	过滤器图表名称。
显示标题	选择显示标题，会在仪表盘中展示过滤器图表的标题。
显示边框	选择显示边框，为过滤器图表增加边框。
显示背景	选择显示背景，为过滤器图表添加白色背景。

配置项		说明
Key值		· 过滤器类型中，Key值为过滤条件中的key。 · 变量替换类型：Key值为指定的变量占位符。 说明: 变量占位符必须是前提条件中已配置的变量占位符，才能成功替换。
类型		过滤器的类型，包括： · 过滤器 · 变量替换 说明: · 过滤器类型中，列表项表示过滤条件中的Value。您可以设置多个Value，生成过滤器之后可以在查看仪表盘时根据需求选择Value。 · 变量替换类型中，列表项为指定变量占位符的替换值。您可以设置多个替换值，生成过滤器之后可以在查看仪表盘时根据需求选择替换值。
别名		列的别名，仅在过滤器类型中指定。设置后，在仪表盘过滤器中显示别名。
全局过滤		是否在所有字段中过滤Value，默认为关闭状态，仅在过滤器类型中指定。 · 开启全局过滤，表示在所有字段中过滤Value。 · 关闭全局过滤，表示仅在指定Key中过滤Value。
列表项	静态列表项	过滤器中预置的列表项。 在添加静态列表项右侧的输入框中输入列表项的值，并单击添加设置列表项。

配置项		说明
	动态列表项	过滤器中动态显示的列表项，列表项为预设的查询分析语句当前的查询结果。 打开添加动态列表项，输入查询分析语句，并单击查询，可以预览动态列表项。

应用场景

过滤器多用于在当前仪表盘中动态修改查询条件和对图表中已经存在的变量占位符进行变量替换。每一张图表实际为一个查询分析语句，满足 `[search query] | [sql query]` 的形式，过滤器实质上会操作该查询分析语句。

- 如果为过滤器，则会在`[serch query]`前加上过滤的值，以`AND`连接为新的查询语句，即`key: value AND [serch query]`
- 如果为变量替换过滤器，则会查询整个仪表盘存在变量占位符的图表，将对应名称的变量占位符替换为选择的`value`值

示例

例如，采集Nginx日志后，需要对采集到的日志数据进行实时查询与分析。

- 场景1：基于不同时间粒度

  通过分析语句可以查看每分钟的访问PV，当需要查看秒级别的数据时，需要调整`__time__ - __time__ % 60`的值，传统做法为修改查询分析语句，多次查询时操作繁琐。此时可以通过过滤器完成变量替换。

  1. 通过以下语句查看分钟访问PV的数据。

     ```
     * | SELECT date_format(__time__ - __time__ % 60, '%H:%i:%s') as time, count(1) as count GROUP BY time ORDER BY time
     ```

  2. 将分析图表添加到仪表盘，并选中`60`作为变量占位符的默认值，变量名为`interval`。

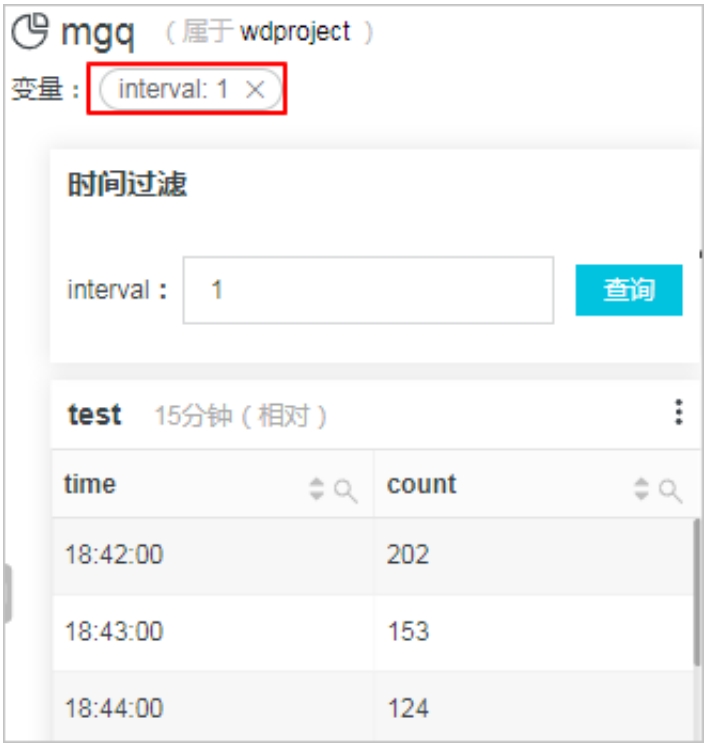
  3. 添加过滤器，并设置类型为变量替换。其中：

     - 类型为变量替换。
     - Key值为`interval`。
     - 列表项为`1`（表示每秒）和`120`（表示每2分钟）。

4. **在过滤器中选择1，此时仪表盘为秒级别的粒度。**

   替换变量后的查询语句：

```
* | SELECT date_format(__time__ - __time__ % 1, '%H:%i:%s') as
time, count(1) as count GROUP BY time ORDER BY time
```



- 场景2：动态切换过滤方法

  通过添加动态列表项还可以动态切换不同的请求方法（method）。场景1中，查询语句为*，表示不设置任何过滤条件，即所有的日志都在查询范围之中。此时，可以再添加一个过滤器便于查看不同method的访问情况。

  1. 增加过滤器并打开添加动态列表项开关，设置如下。
     - 类型为过滤器。
     - Key值为method。
     - 请选择日志库为当前仪表盘所在日志库。
     - 添加动态列表项：通过输入查询语句，动态获取列表项。

  2. 在过滤器的下拉列表项中选择POST。

图表中只显示method为POST的访问。实质上查询分析语句已经变为：

```
(*) and (method: POST) | SELECT date_format(__time__ - __time__ % 60, '%H:%i:%s') as time, count(1) as count GROUP BY time ORDER BY time
```

12.2.8 Markdown图表

日志服务支持在仪表盘中增加Markdown图表，在Markdown图表插入图片、链接、视频等多种元素，使您的仪表盘页面更加友好。

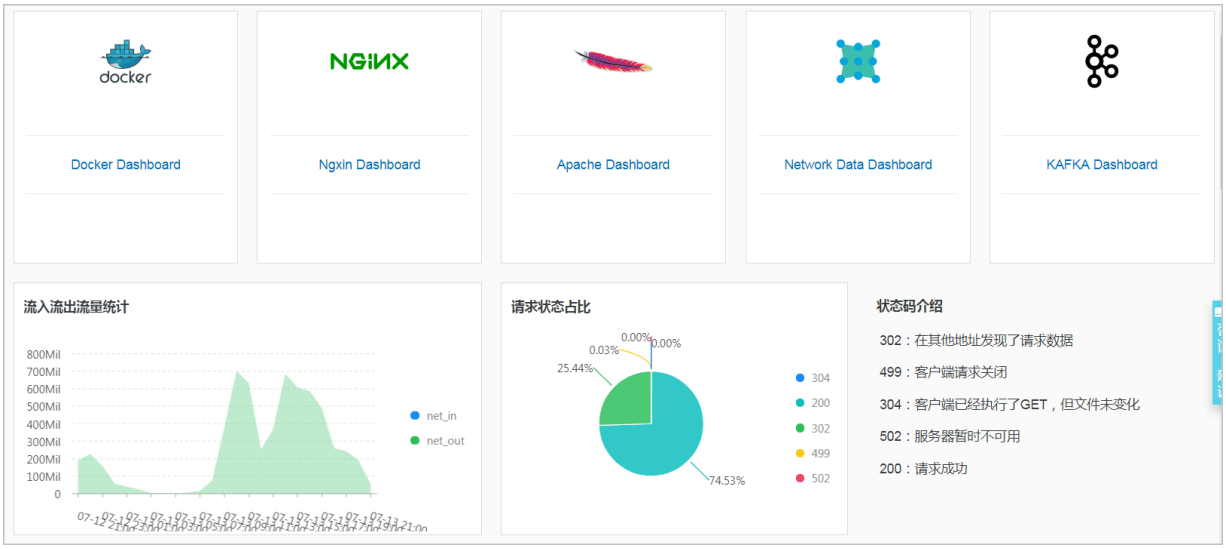
在查询分析日志数据时，将多个分析图表添加到仪表盘中，有利于您快速查看多项分析结果、实时监控多项业务的状态信息。日志服务还支持在仪表盘中增加Markdown图表，该图表使用Markdown语言编辑。您可以在Markdown中插入图片、链接、视频等多种元素，使您的仪表盘页面更加友好。

Markdown图表都是根据不同的需求来创建的。您可以在Markdown图表中插入背景信息、图表说明、页面注释和扩展信息等文字内容，优化仪表盘的信息表达；插入快速查询或其他Project的仪表盘链接，方便其他查询页面的跳转；插入自定义的图片，让您的仪表盘信息更加丰富、功能更为灵活。

应用场景

通过Markdown图表可以自定义跳转链接到当前Project的其他仪表盘，同时还可以插入图片以便快速区分。当需要对图表参数进行介绍时，也可以插入Markdown图表进行说明。

图 12-25: 应用场景



前提条件

1. 已成功采集到日志数据。
2. 已配置仪表盘。

操作步骤

1. 在仪表盘页面单击右上角的编辑按钮。
2. 在编辑模式下，将操作栏中的Markdown图标 拖动到指定位置，即可创建markdown图表。
3. 选中新建的Markdown图表，从右上角展开菜单，并单击编辑。
4. 在弹出页面中设置Markdown图表属性。

配置项	说明
图表名称	您创建的Markdown图表名称。
显示边框	选择显示边框，为您的Markdown图表增加边框。
显示标题	选择显示标题，会在仪表盘中为您展示Markdown图表的标题。
显示背景	选择显示背景，为您的Markdown图表添加白色背景。

配置项	说明
绑定查询	选择绑定查询并设置查询属性后，会在Markdown图表中动态显示查询结果。

5. 绑定查询（可选）

   a. 选择待查询的日志库，在查询框中输入完整的查询分析语句。查询分析语句由查询语句和分析语句构成，格式为查询语句|分析语句。详细说明请参考#unique_216。

   b. 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。

   c. 单击查询，显示当前查询结果的第一条数据。

   d. 单击字段旁的“⊕”，即可将查询结果放置在markdown内容中光标所在位置。

6. 编辑Markdown内容。

   在Markdown内容中输入您的Markdown语句，右侧的图表展示区域会实时展示预览界面。您可以根据预览内容调整Markdown语句。

7. 配置完成后单击确定。

图 12-26: 创建Markdown图表

配置完成后，您可以在当前仪表盘中查看Markdown图表。

修改Markdown图表

- 修改图表位置和大小

  1. 在仪表盘页面单击右上角的编辑。
  2. 鼠标拖动Markdown图标到指定位置，拖动图表右下角调整图表大小。
  3. 在页面右上角单击保存。

- 修改图表标题

  1. 在仪表盘页面单击右上角的编辑。
  2. 单击指定Markdown图表，并在右上角的折叠列表中单击编辑。
  3. 图表名称中输入新的标题，并单击确定。
  4. 仪表盘页面右上角单击保存，退出仪表盘，并在弹出对话框中单击确定。

- 修改图表内容

  1. 在仪表盘页面单击右上角的编辑。
  2. 单击指定Markdown图表，并在右上角的折叠列表中单击编辑。
  3. 修改图表配置，并单击确定。
  4. 仪表盘页面右上角单击保存，退出编辑模式，并在弹出对话框中单击确定。

- 删除图表

  1. 在仪表盘页面单击右上角的编辑。
  2. 单击指定Markdown图表，并在右上角的折叠列表中单击删除。
  3. 仪表盘页面右上角单击保存，退出编辑模式，并在弹出对话框中单击确定。

常用Markdown语法

- 标题

  Markdown语句：

```
# 一级标题
## 二级标题
```

```
### 三级标题
```

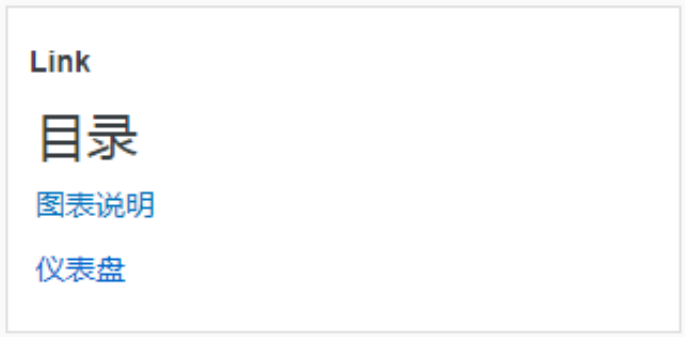
图 12-27: 标题预览

- 链接

  Markdown语句：

```
### 目录
[图表说明](https://help.aliyun.com/document_detail/69313.html)
[仪表盘](https://help.aliyun.com/document_detail/59324.html)
```

图 12-28: 链接预览



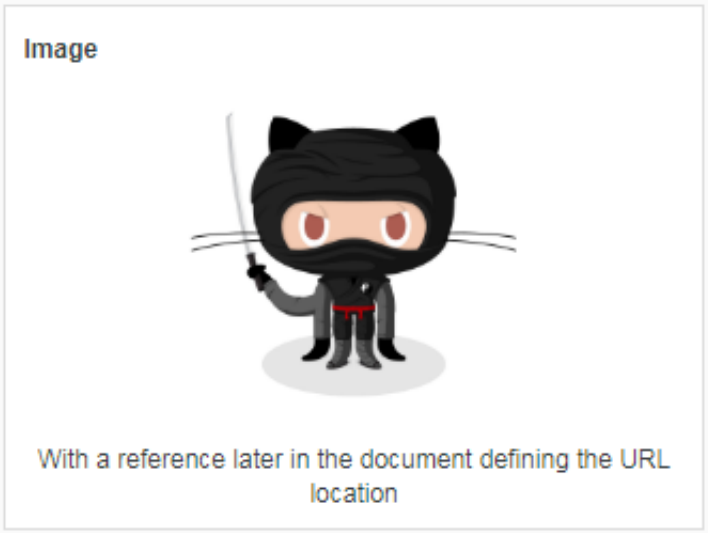
- 图片

  Markdown语句：

```
<div align=center>
![Alt txt][id]
With a reference later in the document defining the URL location
```

```
[id]: https://octodex.github.com/images/dojocat.jpg  "The Dojocat"
```

图 12-29: 图片预览



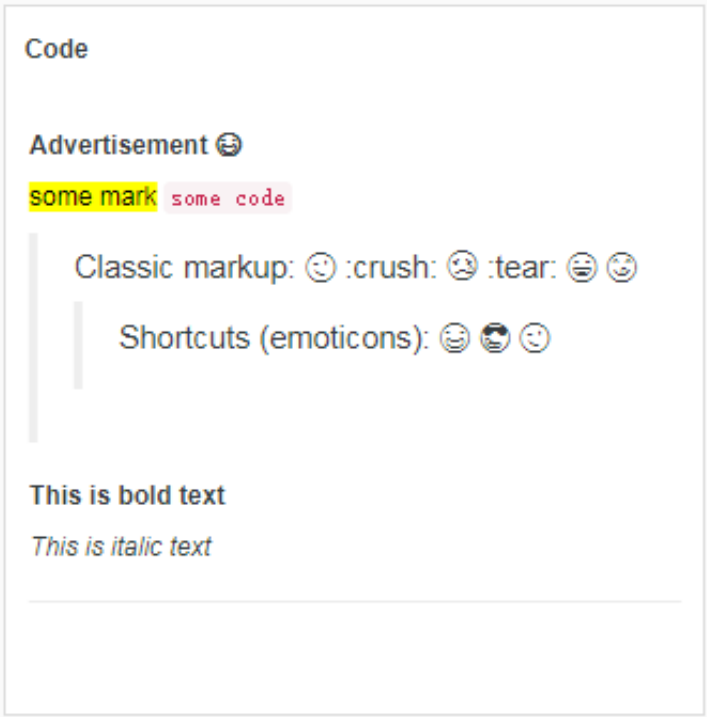
- 特殊标记

Markdown语句:

```
---

__Advertisement :)__

==some mark==    `some code`
> Classic markup: :wink: :crush: :cry: :tear: :laughing: :yum:
>> Shortcuts (emoticons): :-)  8-) ;)

__This is bold text__

*This is italic text*
```

```
---
```

图 12-30: 特殊标记预览



关于Markdown语法的详细说明，请查看Markdown语法。

13 最佳实践

13.1 查询分析-分页

查询日志时，查询结果内容过多会影响显示速度和查询体验。通过API接口进行分页查询可以指定查询结果分页显示，控制每次返回的数据量。

日志服务提供了分页的功能，不仅可以分页读取原始日志内容，也可以把SQL的计算结果分页读取到本地。开发者可以通过日志服务提供的SDK或者CLI，通过读数据接口分页读取日志。

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。GetLogstoreLogs接口是日志服务提供的日志查询入口，您既可以根据关键字查询日志原始内容，也可以进行SQL计算，获取分析结果。查询分析语句中的查询部分和分析部分使用不同的分页方式。

- 查询语句：使用关键字查询，获取原始日志内容。通过API中的参数offset和lines来分页获取所有内容。
- 分析语句：使用SQL对查询结果进行分析，获取统计结果。通过SQL的limit语法来达到分页效果。

查询结果分页

在GetLogStoreLogs API中，参数offset和lines可以用来设置分页。

- offset：指定从第一行开始读取日志。
- lines：指定当前的请求读取多少行，该参数最大可设置为100，如果设置该参数大于100，则仍然返回100行。

在分页读取时，不停的增大offset，直到读取到某个offset后，获取的结果行数为0，并且结果的progress为complete状态，则认为读取到了全部数据。

查询分页代码示例

- 分页的伪代码：

```
offset = 0                                  // 从第0行开始读取
lines = 100                                 //每次读取100行
query = "status:200"                        //查询status字段包含200的所有日志
while True:
     response = get_logstore_logs(query, offset, lines) // 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 && response.is_complete()
          则读取结束，跳出当前循环
      否则
         offset += 100                                  offset增加100，读取下
一个100行
```

- Python分页读取：

  详细案例请参考#unique_222。

```
    endpoint = ''# 选择与上面步骤创建Project所属区域匹配的Endpoint
    accessKeyId = ''# 使用您的阿里云访问密钥AccessKeyId
    accessKey = ''# 使用您的阿里云访问密钥AccessKeySecret
    project = ''# 上面步骤创建的项目名称
    logstore = ''# 上面步骤创建的日志库名称
    client = LogClient(endpoint, accessKeyId, accessKey)
    topic = ""
    query = "index"
    From = int(time.time()) - 600
    To = int(time.time())
    log_line = 100
    offset = 0
    while True:
        res4 = Nonefor retry_time in range(0, 3):
            req4 = GetLogsRequest(project, logstore, From, To, topic
, query, log_line, offset, False)
            res4 = client.get_logs(req4)
            if res4 isnotNoneand res4.is_completed():
                break
            time.sleep(1)
        offset += 100if res4.is_completed() && res4.get_count() == 0
:
              break;
        if res4 isnotNone:
            res4.log_print()  # 这里处理结果
```

- Java分页读取：

  更详细的案例参考#unique_223。

```
        int log_offset = 0;
        int log_line = 100;//log_line 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。offset和lines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 log offset,一次读取 10 行 log，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

GetLogStoreLogs API参数中的offset和lines不支持用于SQL分析。如果按照上文分页读取原始内容的方式，遍历offset分页，那么每次SQL执行的结果都是一样的。如果在一次调用中获取全部的计算结果，可能会由于结果集太大而产生以下问题：

- 网络上传输大量数据延时比较高。
- 客户端的内存要保存大量的结果以供进一步处理，影响内存占用率。

为了解决SQL分页的问题，日志服务提供了标准SQL的limit分页语法：

```
limit Offset, Line
```

- Offset表示从第几行开始读取结果。
- Line表示读取多少行。Line没有大小限制，但是如果一次读取太多，会影响网络延时和客户端的处理速度。

例如，分析语句`* | selectcount(1) , url group by url` 返回2000条日志，可以通过分页指定每次读取500行，共4次读取完成：

```
* | selectcount(1) , url  group by url  limit 0, 500
* | selectcount(1) , url  group by url  limit 500, 500
* | selectcount(1) , url  group by url  limit 1000, 500
* | selectcount(1) , url  group by url  limit 1500, 500
```

分析分页代码示例

- SQL分页的伪代码：

```
    offset = 0// 从第0行开始读取
    lines = 500//每次读取500行
    query = "* | select count(1) , url  group by url  limit "
whileTrue:
    real_query = query + offset + "," +  lines
    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100，读取下一个500行
```

- Python程序代码：

```
    endpoint = ''# 选择与上面步骤创建Project所属区域匹配的Endpoint
```

```
    accessKeyId = ''# 使用您的阿里云访问密钥AccessKeyId
    accessKey = ''# 使用您的阿里云访问密钥AccessKeySecret
    project = ''# 上面步骤创建的项目名称
    logstore = ''# 上面步骤创建的日志库名称
    client = LogClient(endpoint, accessKeyId, accessKey)
    topic = ""
    origin_query = "* | select count(1) , url  group by url  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
, query)
            res4 = client.get_logs(req4)
            if res4 isnotNoneand res4.is_completed():
                break
            time.sleep(1)
        offset += 100if res4.is_completed() && res4.get_count() == 0
:
              break;
        if res4 isnotNone:
            res4.log_print()  # 这里处理结果
```

- Java程序代码：

```
        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 log offset,一次读取 500 行 log，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;
            }

        }
```

13.2 查询-消息服务(MNS)日志

阿里云消息服务(MNS)支持将日志推送到日志服务，本文为您介绍日志成功推送后，如何通过日志查询特定信息。

消息服务支持队列消息操作日志以及主题消息操作日志，其中日志包含了消息生命周期的所有内容，时间、地点、操作和上下文等。您可以通过实时查询、实时计算和离线计算三种方法对日志进行分析。

实时查询

本文档仅介绍几种常用场景的查询，用户可以通过组合多个关键字来实现更加复杂的查询。

查看队列消息的消息轨迹

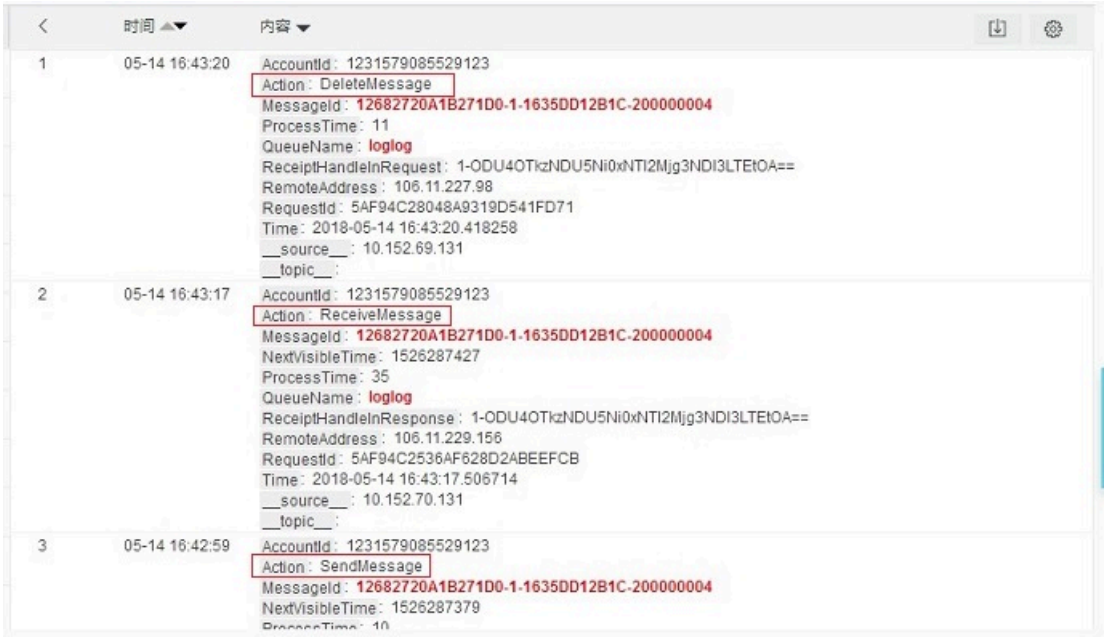
- 步骤

  1. 在搜索框中输入队列名称和MessageId。格式为`$queuename and $messageid`。
  2. 选择合适的时间范围后，单击查询/分析即可查看该消息的详细操作日志。

- 示例

  查看loglog队列中MessageId为`12682720A1B271D0-1-1635DD12B1C-200000004`的消息轨迹。

  - 查询语句：`loglog and 12682720A1B271D0-1-1635DD12B1C-200000004`
  - 查询结果：如下图所示，查询结果中展示了该消息从发送到删除的过程。



查看队列消息写入量

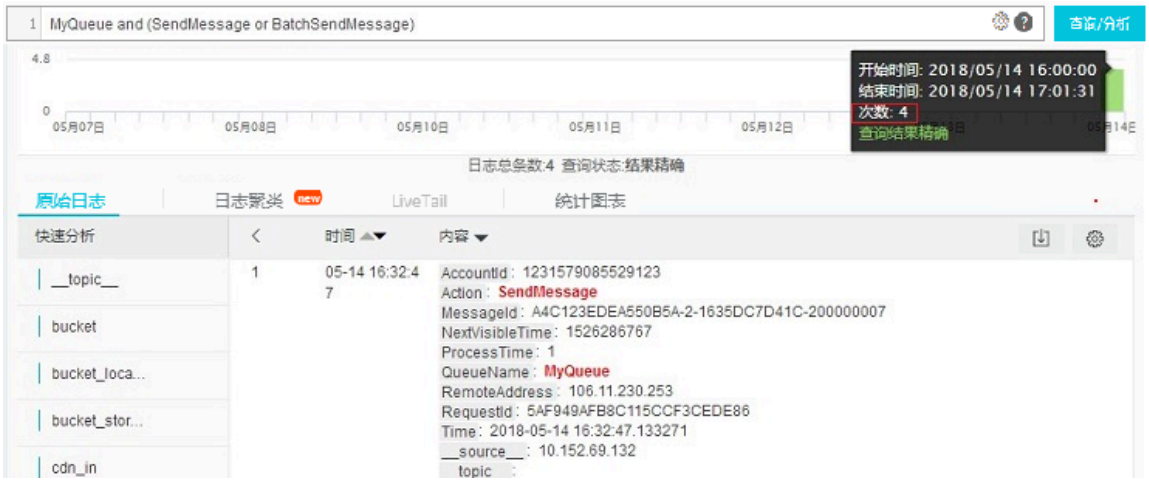
- 步骤

  1. 在搜索框中输入队列名称和写入操作。格式为`$queuename and (SendMessage or BatchSendMessage)`。
  2. 选择合适的时间范围后，单击查询/分析即可查看该队列的所有写入消息。鼠标移至绿色柱状图上，可以查看当前时间段内的具体消息数量。

- 示例

  查看MyQueue队列中的消息写入量。

  - 查询语句：`MyQueue and (SendMessage or BatchSendMessage)`
  - 查询结果：如下图所示，当前查询时段内，有4条写入操作。



查看队列消息消费量

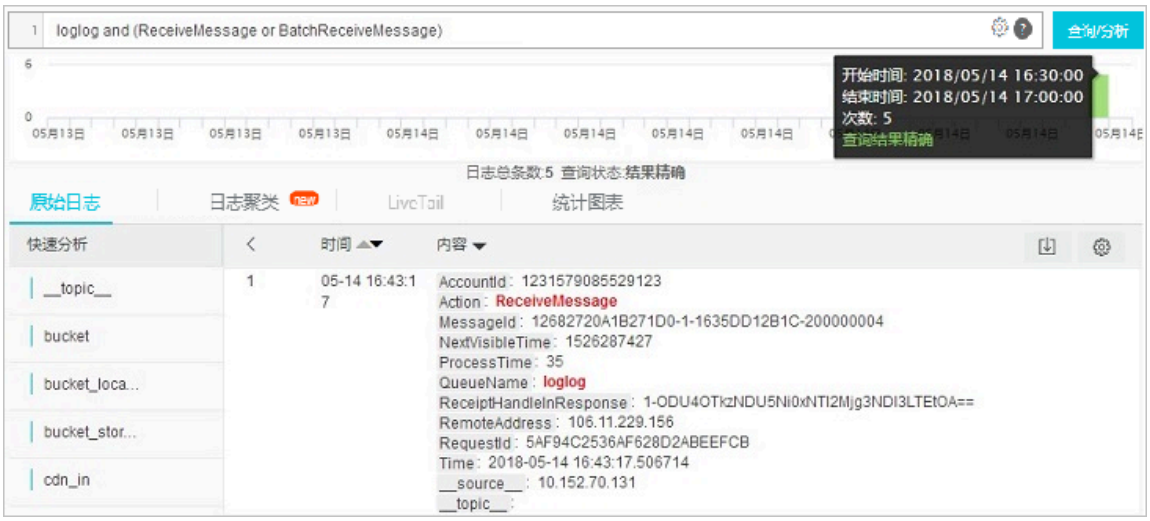
- 步骤

  1. 在搜索框中输入队列名称和消费操作。格式为`$queuename and (ReceiveMessage or BatchReceiveMessage)`。
  2. 选择合适的时间范围后，单击查询/分析即可查看该队列消息消费量。

- 示例

  查看loglog队列中的消息消费量。

  - 查询语句：`loglog and (ReceiveMessage or BatchReceiveMessage)`
  - 查询结果：如下图所示，当前查询时段内，有5条消费记录。



查看队列消息删除量

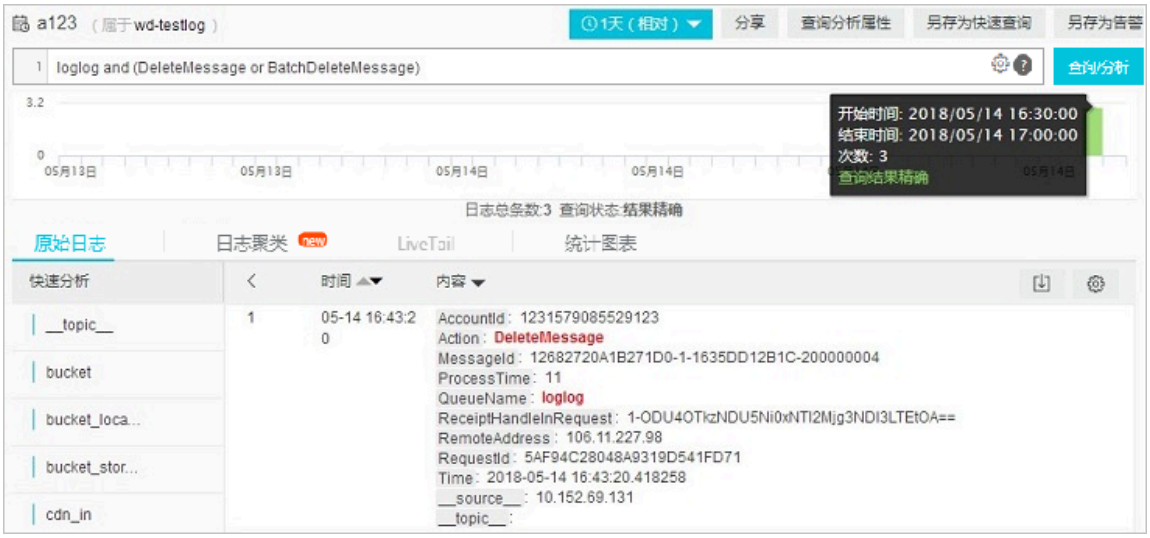
- 步骤

  1. 在搜索框中输入队列名称和删除操作。格式为：`$queuename and (DeleteMessage or BatchDeleteMessage)`。
  2. 选择合适的时间范围后，单击查询/分析即可查看该队列消息删除量。

- 示例

  查看名为loglog的队列消息删除量。

  - 查询语句：`loglog and (DeleteMessage or BatchDeleteMessage)`
  - 查询结果：如下图所示，搜索结果中展示了loglog队列消息的删除日志，您可以查看删除量。



查看主题消息的消息轨迹

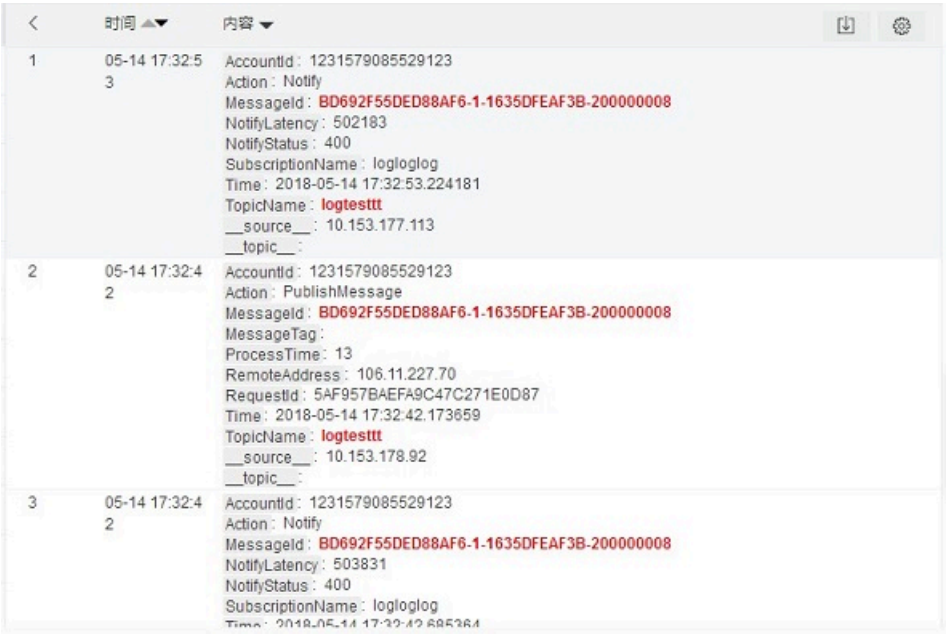
- 步骤

  1. 在搜索框中输入主题名称和messageid。格式为`$topicname and $messageid`。
  2. 选择合适的时间范围后，单击查询/分析即可查看该主题消息的消息轨迹。

- 示例

  查看名为logtesttt主题中MessageId为`BD692F55DED88AF6-1-1635DFEAF3B-200000008`的消息轨迹。

  - 查询语句：`logtesttt and BD692F55DED88AF6-1-1635DFEAF3B-200000008`
  - 查询结果：如下图所示，搜索结果中展示了logtesttt主题中消息从发送到通知的过程。



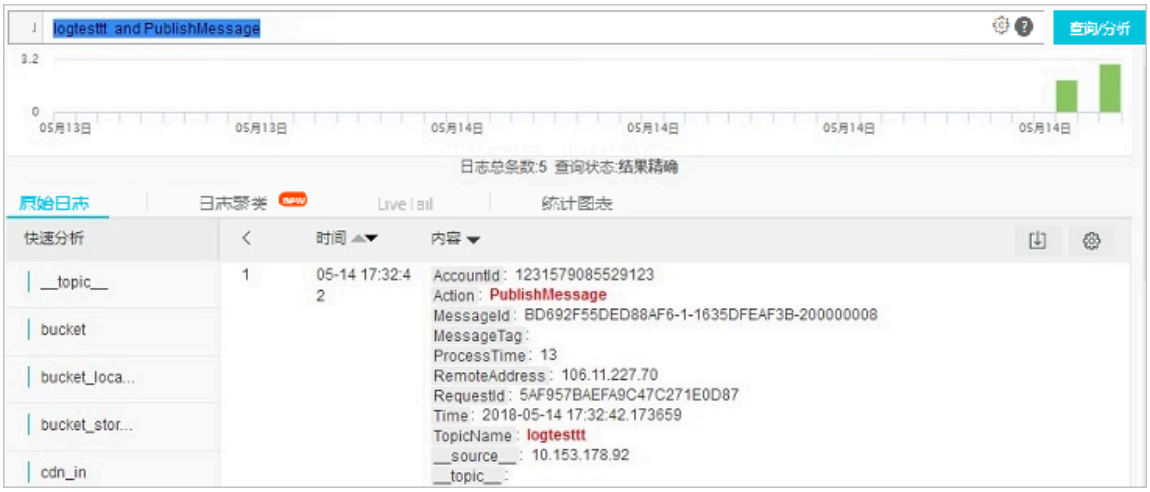
	时间	内容
1	05-14 17:32:53	AccountId: 1231579085529123 Action: Notify MessageId: BD692F55DED88AF6-1-1635DFEAF3B-200000008 NotifyLatency: 502183 NotifyStatus: 400 SubscriptionName: logloglog Time: 2018-05-14 17:32:53.224181 TopicName: logtesttt __source__: 10.153.177.113 __topic__:
2	05-14 17:32:42	AccountId: 1231579085529123 Action: PublishMessage MessageId: BD692F55DED88AF6-1-1635DFEAF3B-200000008 MessageTag: ProcessTime: 13 RemoteAddress: 106.11.227.70 RequestId: 5AF957BAEFA9C47C271E0D87 Time: 2018-05-14 17:32:42.173659 TopicName: logtesttt __source__: 10.153.178.92 __topic__:
3	05-14 17:32:42	AccountId: 1231579085529123 Action: Notify MessageId: BD692F55DED88AF6-1-1635DFEAF3B-200000008 NotifyLatency: 503831 NotifyStatus: 400 SubscriptionName: logloglog Time: 2018-05-14 17:32:42.695364

查看主题消息发布量

- 步骤

  1. 在搜索框中输入主题名称和发布操作。格式：`$topicname and PublishMessage`。
  2. 选择合适的时间范围后，单击查询/分析即可查看该主题消息发布量。

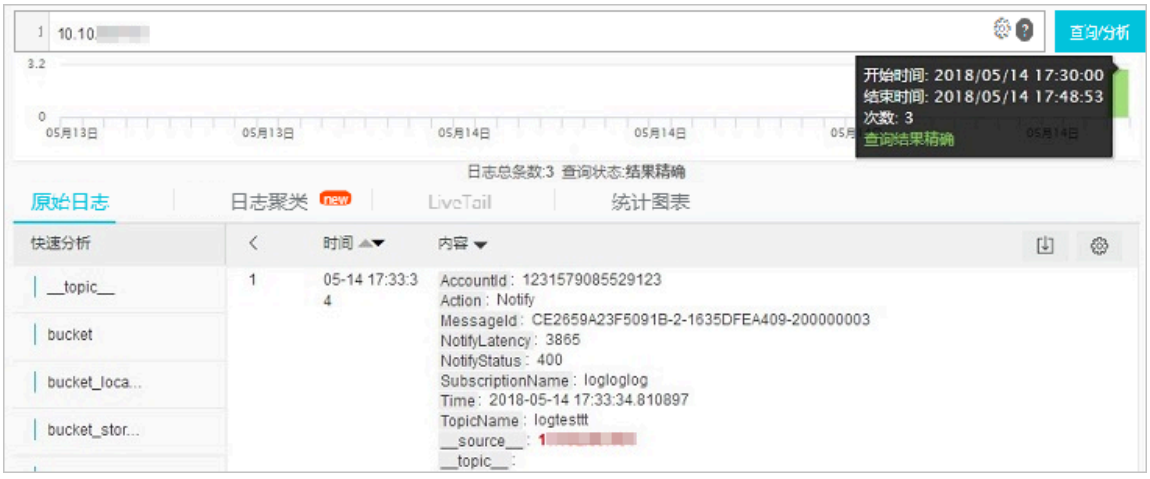
- 示例：查询名为logtesttt的主题消息发布量。
  - 查询语句：`logtesttt and PublishMessage`
  - 查询结果：如下图所示，当前查询时段内，有5条消息发布记录。



查看某个客户端消息处理量

- 步骤
  1. 索框中输入客户端IP。格式：`$ClientIP`，如果希望查询某个客户端的某类操作日志，搜索框中增加具体操作即可，例如：`$ClientIP and (SendMessage or BatchSendMessage)`。
  2. 选择合适的时间范围后，单击搜索按钮即可查看该客户端所有的消息操作日志。
- 示例：

  查询IP为10.10.XX.XX的客户端消息处理量。
  - 查询语句：`10.10.XX.XX`,
  - 查询结果：如下图所示，当前查询时段内，有3条消息处理记录。



实时计算 & 离线计算

- 实时计算：使用Spark、Storm或StreamCompute，Consumer Library等方式可以实时对消息服务日志进行分析。例如：
  - 对一个队列而言，Top 10 消息的产生者、消费者分别是谁哪些IP?
  - 生产和消费的速度是否均衡？某些消费者在处理延时上是否有瓶颈?
- 离线：使用MaxCompute 或 E-MapReduce/Hive进行大时间跨度的计算。
  - 最近一周内，消息从发布到被消费平均延迟是什么?
  - 对比升级前和升级后两个时间段内性能变化如何?

13.3 查询分析-程序日志

程序日志（AppLog）有什么特点?

- 内容最全：程序日志是由程序员给出，在重要的地点、变量数值以及异常都会有记录，可以说线上90%以上Bug都是依靠程序日志输出定位到的。
- 格式比较随意：代码往往经过不同人开发，每个程序员都有自己爱好的格式，一般非常难进行统一，并且引入的一些第三方库的日志风格也不太一样。
- 有一定共性：虽然格式不统一，但一般都会有一些共性的地方。例如对Log4J日志而言，会有如下几个固定字段：
  - 时间
  - 级别（Level）
  - 所在文件或类（file or class）
  - 行数（Line Number）
  - 线程号（ThreadId）

处理程序日志会有哪些挑战?

- 数据量大

  程序日志一般会比访问日志大1个数量级：假设一个网站一天有100W次独立访问，每个访问大约有20逻辑模块，在每个逻辑模块中有10个主要逻辑点需要记录日志。

  则日志总数为：

  ```
  100W * 20 * 10 = 2 * 10^8 条
  ```

  每条长度为200字节，则存储大小为：

  ```
  2 * 10^8 * 200 = 4 * 10^10 = 40 GB
  ```

  这个数据会随着业务系统复杂变得更大，一天100-200GB日志对一个中型网站而言是很常见的。

- 分布服务器多

  大部分应用都是无状态模式，跑在不同框架中，例如:

  - 服务器
  - Docker（容器）
  - 函数计算（容器服务）

  对应实例数目会从几个到几千，需要有一种跨服务器的日志采集方案。

- 运行环境复杂

  程序会落到不同的环境上，例如：

  - 应用相关的会在容器中
  - API相关日志会在FunctionCompute中
  - 旧系统日志在传统IDC中
  - 移动端相关日志在用户处
  - 网页端（M站）在浏览器里

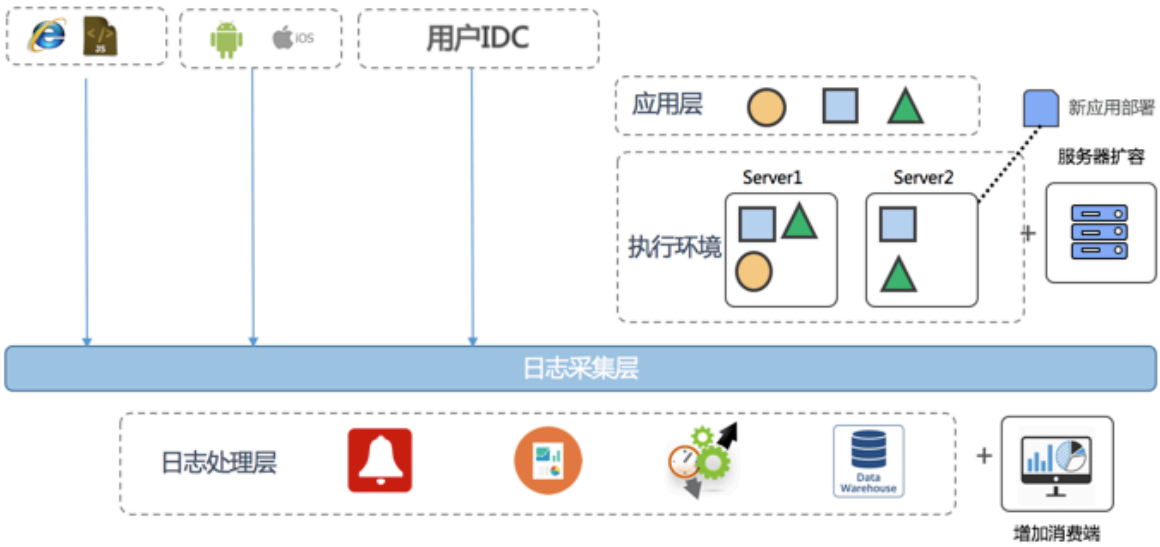
  为了能够获得全貌，我们必须把所有数据统一并存储起来。

如何解决程序日志需求

统一存储

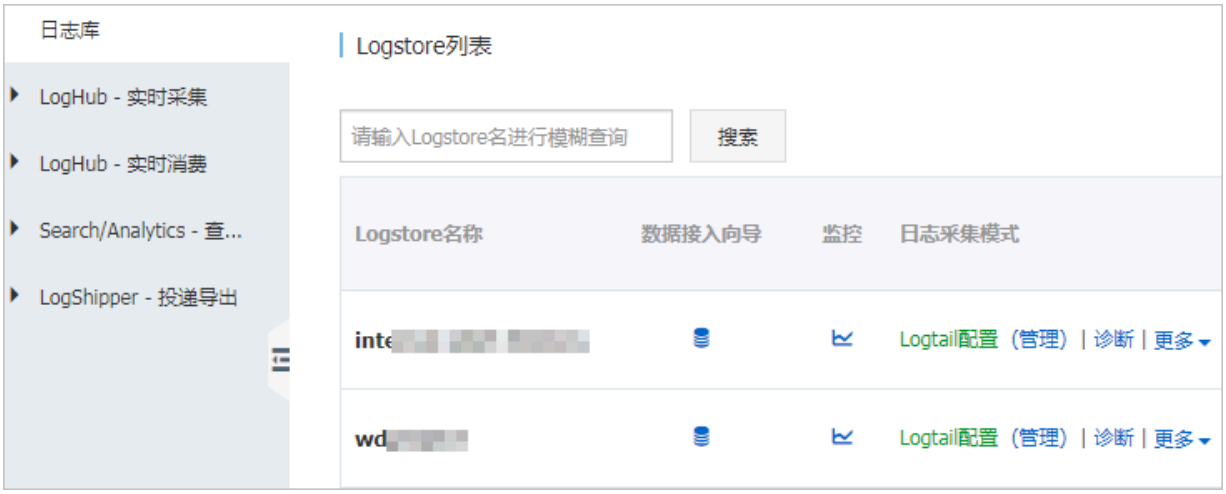
目标：要把各渠道数据采集到一个集中化中心，打通才可以做后续事情。

我们可以在日志服务中创建一个项目来存储应用日志，日志服务提供30+种日志采集手段：无论是在硬件服务器中埋点，还是网页端JS，或是服务器上输出日志都支持采集。



在服务器日志上，除了使用SDK等直接写入外，日志服务提供便捷、稳定、高性能Agent-Logtail。Logtail提供Windows、Linux两个版本，在控制台定义好机器组，日志采集配置后，就能够实时将服务日志进行采集。

在创建完成一个日志采集配置后，我们就可以在项目中操作各种日志了。



可能有人要问到，日志采集Agent非常多，有Logstash，Flume，FluentD，以及Beats等，Logtail和这些相比有什么特点吗?

- 使用便捷：提供API、远程管理与监控功能，融入阿里集团百万级服务器日志采集管理经验，配置一个采集点到几十万设备只需要几秒钟。
- 适应各种环境：无论是公网、VPC、用户自定义IDC等都可以支持，https以及断点续传功能使得接入公网数据也不再话下。
- 性能强，对资源消耗非常小：经过多年磨练，在性能和资源消耗方面比开源要好。

快速查找定位

目标：无论数据量如何增长、服务器如何部署，都可以保证定位问题时间是恒定的。

例如有一个订单错误，一个延时很长，我们如何能够在一周几TB数据量日志中快速定位到问题。其中还会涉及到各种条件过滤和排查等。

1. 例如我们对于程序中记录延时的日志，调查延时大于1秒，并且方法以Post开头的请求数据：

```
Latency > 1000000 and Method=Post*
```

2. 对于日志中查找包含error关键词，不包含merge关键词的日志：

   - 一天的结果

   - 一周的结果

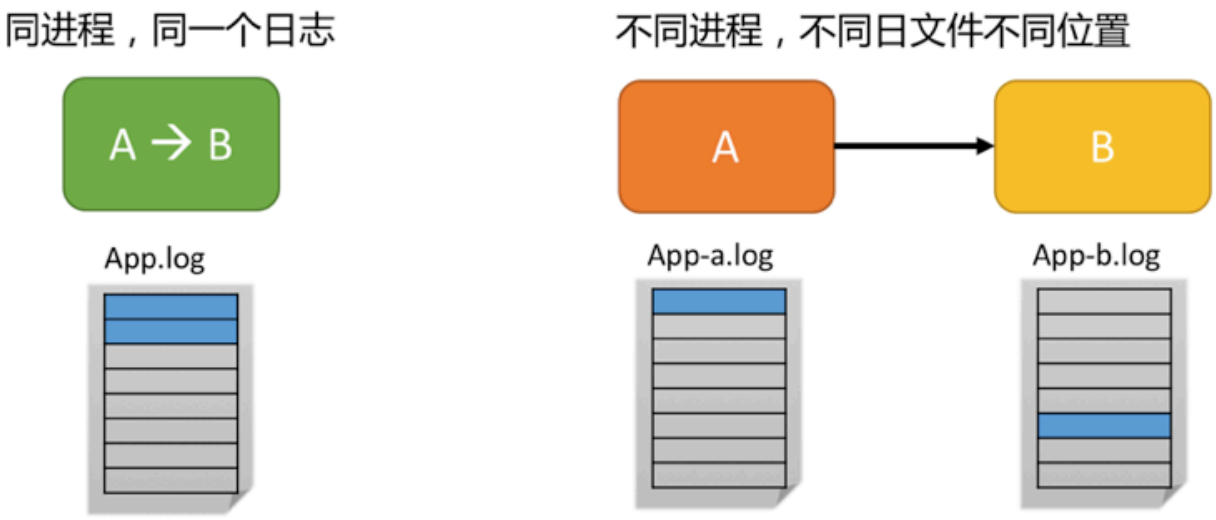
   - 更长时间结果

   这些查询都是在不到1秒时间内可以返回。

关联分析

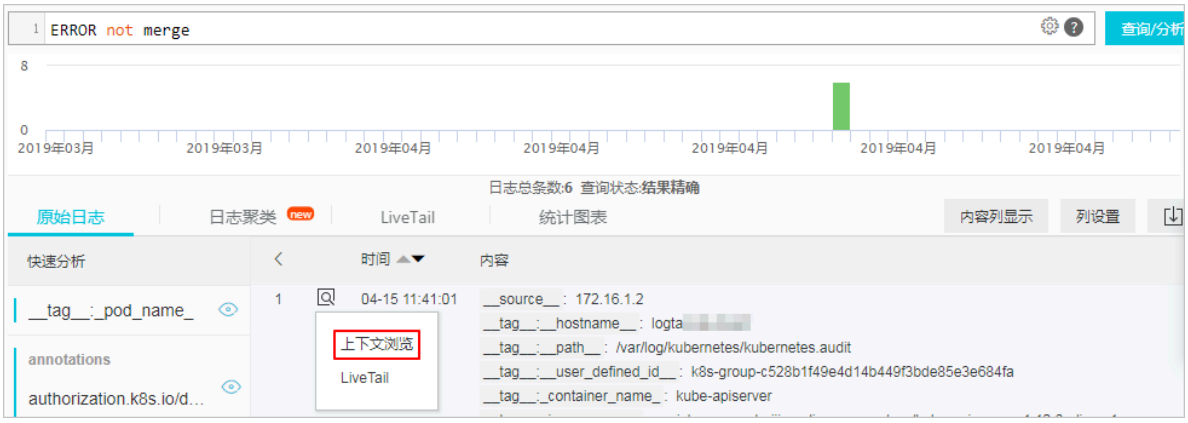
关联有两种类型，进程内关联与跨进程关联。我们先来看看两者有什么区别：

- 进程内关联：一般比较简单，因为同一个函数前后日志都在一个文件中。在多线程环节中，我们只要根据线程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，一般会通过RPC中传入TracerId来进行关联。


同进程，同一个日志
A → B
App.log
不同进程，不同日文件不同位置
A
B
App-a.log
App-b.log

· 上下文关联

还是以使用日志服务控制台举例，线上通过关键词查询定位到一个异常日志：



点击上下文浏览后，即跳转到前后N条上下文。

- 显示框可以通过更早、更新等按钮加载更多上下文。
- 也可以输入关键词进行高亮显示。

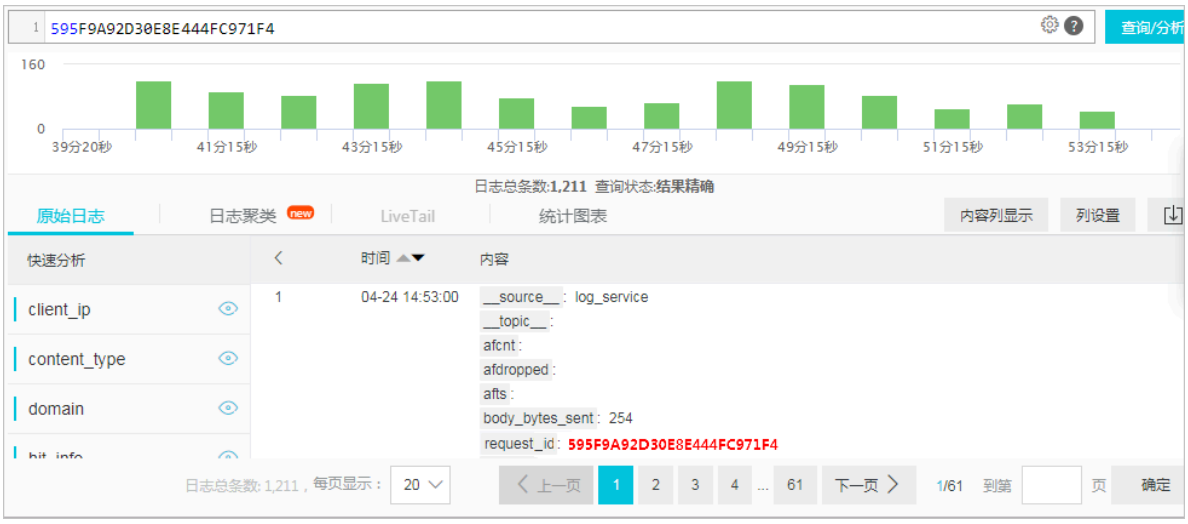
- 跨进程关联

跨进程关联也叫Tracing，最早的工作是Google在2010年的那篇著名“Dapper, a Large-Scale Distributed Systems Tracing Infrastructure”，之后开源界借鉴了Google思想做成过了平民化的各种Tracer版本。目前比较有名的有：

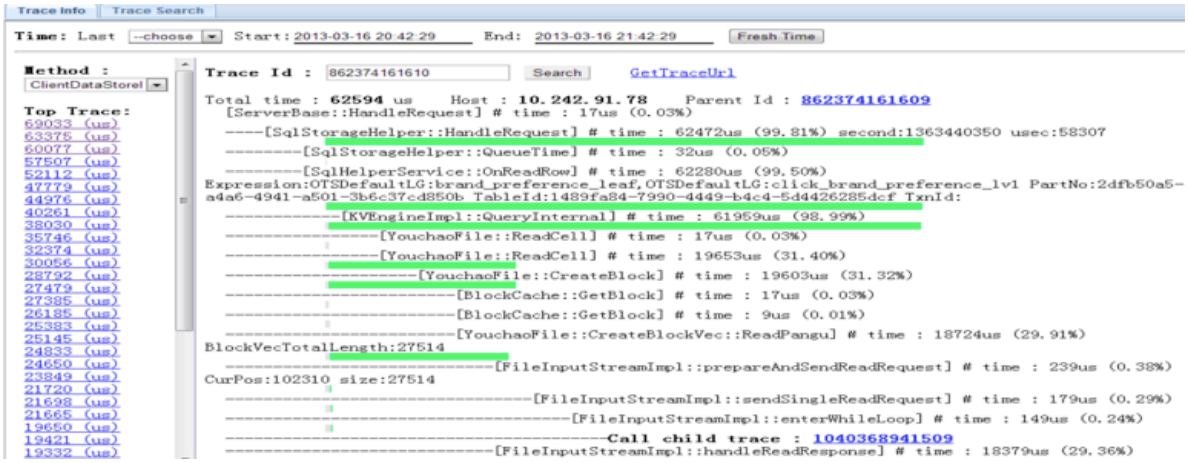
- Dapper(Google): 各 tracer基础
- StackDriver Trace (Google)，现在兼容了ZipKin
- Zipkin：twitter开源的Tracing系统
- Appdash：golang版本
- 鹰眼：阿里巴巴集团中间件技术技术部研发
- X-ray：AWS在2016年Re：Invent上推出技术

如果从0开始使用Tracer会相对容易一些，但在现有系统中去使用，则会有改造的代价和挑战。

今天我们可以基于日志服务，实现一个基本Tracing功能：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，即可拿到所有相关的日志。



例如我们可以通过SDK查询 前端机，后端机，支付系统，订单系统等日志，拿到结果后做一个前端的页面将跨进程调用关联起来，以下就是基于日志服务快速搭建的Tracing系统。



统计分析

查找到特点日志后，我们有时希望能做一些分析，例如线上有多少种不同类型的错误日志。

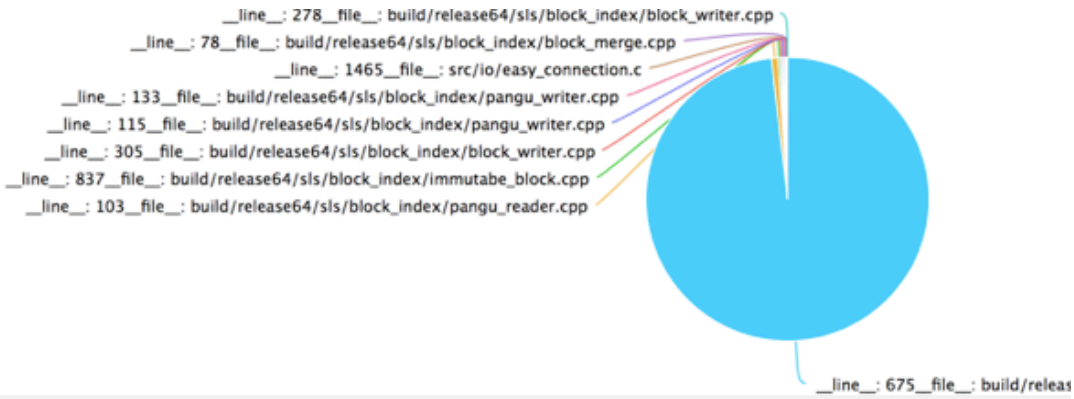
1. 我们先通过对`__level__`这个日志级别字段进行查询，得知一天内有20个错误：

```
__level__:error
```

2. 接下来，我们可以根据file，line这两个字段（确定唯一的日志类型）来进行统计聚合。

```
__level__:error | select __file__, __line__, count(*) as c group by
__file__, __line__ order by c desc
```

即可得出所有错误发生的类型和位置的分布。



__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

其他还有诸如根据错误码、高延时等条件进行IP定位与分析等。

其他

- 备份日志审计

  可以将日志备份至OSS或存储成本更低的IA，或直接到MaxCompute。

- 关键词报警

  目前有如下方式可以进行报警。

  - 日志服务告警
  - 通过云监控报警

- 日志查询权限分配管理。

  可以通过子账号+授权组方法隔离开发、PE等权限。

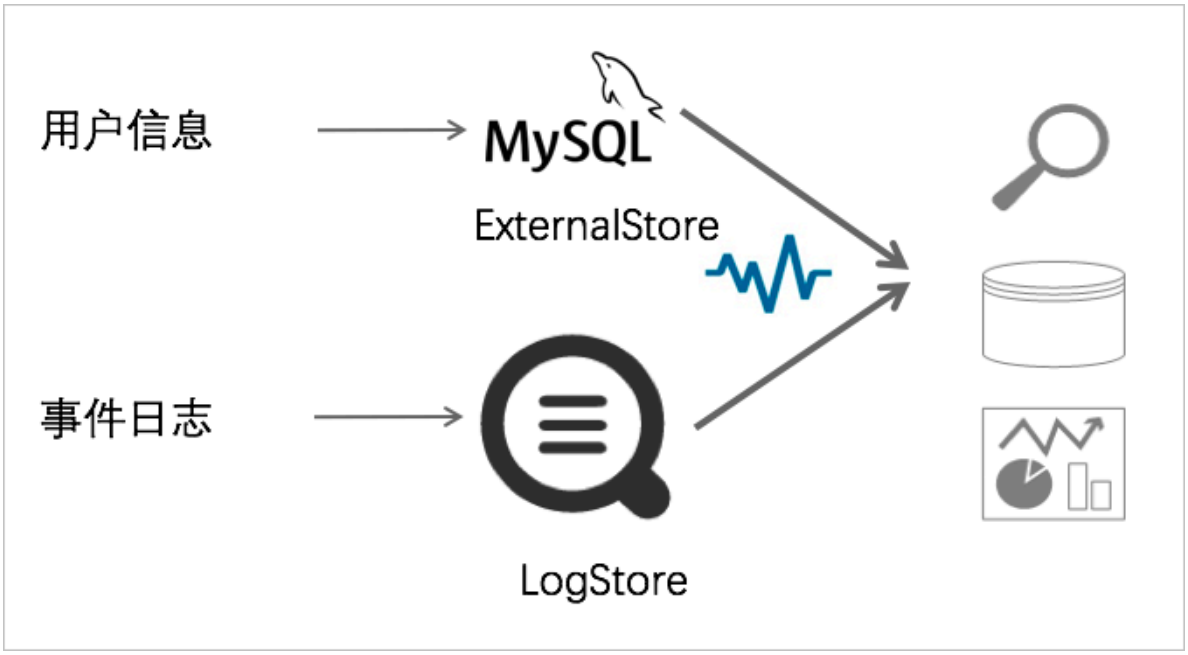
  最后说一下价格与成本，程序日志主要会用到日志服务LogHub + LogSearch功能，和开源方案对比，在查询成本上是开源方案25%，使用非常便捷，令您的开发工作事半功倍。

13.4 查询分析-数据库与日志关联分析

在日志分析场景中，我们经常会遇到数据分散在各地的场景。而我们需要用日志和数据库中的数据对用户进行分层统计，将最后的计算结果写入数据库中供报表系统查询，因此，我们要在日志服务Logstore和其他数据源中进行关联查询。

背景信息

- 用户日志数据：以游戏日志为例，一条经典的游戏日志，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。
- 用户元数据：日志表示的是增量的事件，一些静态的用户信息，例如用户的性别、注册时间、地区等是固定不变的，在客户端很难获取，不能够打印到日志里。我们把这些信息称为用户元信息。
- 日志服务和MySQL关联分析：日志服务查询分析引擎，提供跨Logstore和ExternalStore的查询分析功能，使用SQL的join语法把日志和用户元信息关联起来，用来分析跟用户属性相关的指标。除在查询过程中引用ExternalStore，日志服务还支持将计算结果直接写入ExternalStore中（例如MySQL），方便结果的进一步处理。
  - 日志服务Logstore：提供日志的收集、存储、查询分析。
  - 日志服务ExternalStore：映射到RDS表，开发者把用户信息放到RDS表中。



操作步骤

1. 采集日志到日志服务。
   - 移动端日志采集：Android，iOS。
   - 服务器日志采集：ilogtail
2. 创建用户属性表。

   创建一张chiji_user表，保存用户的id、昵称、性别、年龄、注册时间、账户余额、注册省份。

```
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,
  `province` text, PRIMARY KEY (`uid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

3. 创建ExternalStore。

   a. 创建ExternalStore需要使用日志服务CLI，因此需要安装CLI。

```
pip install -U aliyun-log-cli
```

   b. 创建ExternalStore，指定所属的Project，以及ExternalStore的配置文件/root/config.json。

```
aliyunlog log create_external_store --project_name="log-rds-demo"
 --config="file:///root/config.json"
```

   c. 在配置文件中，指定外部存储的名称，参数。RDS VPC需要指定的参数有：vpc-id，RDS实例id，域名、端口、用户名、密码、数据库和表名、RDS所属region。

```
{
     "externalStoreName": "chiji_user",
     "storeType": "rds-vpc",
     "parameter": {
     "vpc-id": "vpc-m5eq4irc1pucpk85f****",
     "instance-id": "rm-m5ep2z57814qs****",
     "host": "example.com",
     "port": "3306",
     "username": "testroot",
     "password": "123456789",
     "db": "chiji",
     "table": "chiji_user",
     "region": "cn-qingdao"
  }
}
```

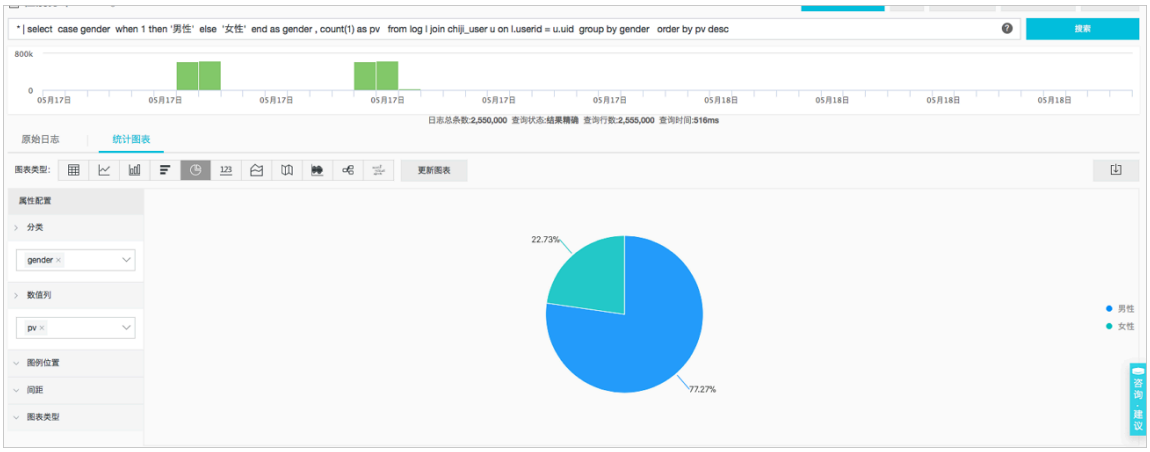
4. 添加白名单。
   - 在RDS中，添加白名单地址`100.104.0.0/16`。
   - 如果是MySQL，请添加该地址到安全组。

5. 关联分析。

- 分析活跃用户的性别分布。

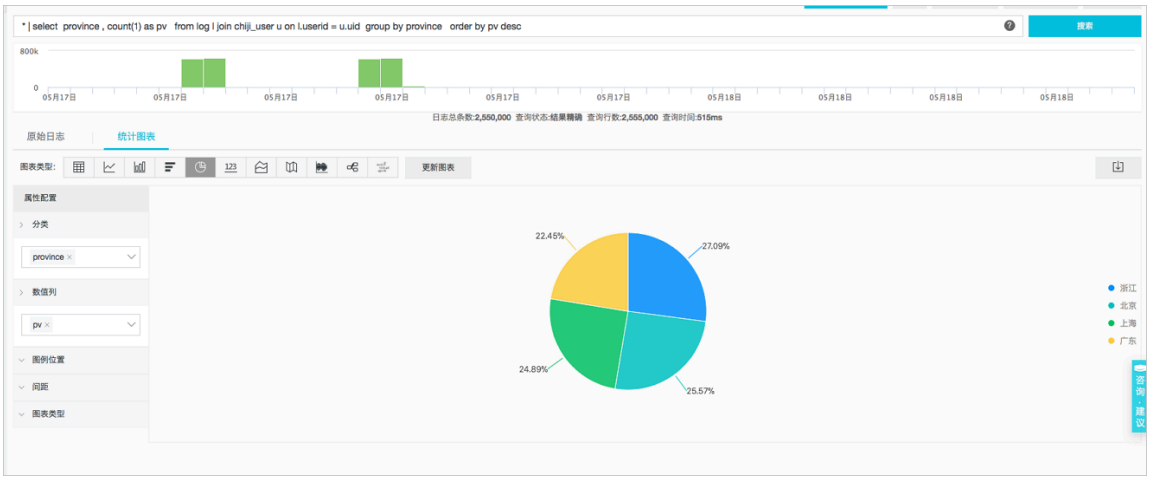
  使用join语法，通过指定日志中的userid和RDS中的uid相等来关联日志和用户属性。

```
* | select case gender when 1 then '男性' else '女性' end as gender
 , count(1) as pv from log l join chiji_user u on l.userid = u.uid
 group by gender order by pv desc
```



- 分析不同省份活跃度。

```
* | select province , count(1) as pv from log l join chiji_user u
on l.userid = u.uid group by province 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 from log l join chiji_user u on l.userid =
u.uid group by gender order by money desc
```

6. 保存查询分析结果。

a. 创建结果表，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
```

b. 参考上述创建ExternalStore的方法给report表创建ExternalStore，然后将结果保存到report。

```
* | insert into report select __time__- __time__ % 300 as min,
count(1) as pv group by min
```

SQL返回的结果是最终输出到RDS中的行数。

```
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 |
```

13.5 查询分析-日志服务与OSS外表关联分析

在日志分析场景中，我们经常遇到日志中的信息不完善。例如，日志中包含了用户的点击行为，但是却缺少用户的属性，例如注册信息、资金等信息。 而分析日志的时候，往往需要联合分析用户的属性和行为，例如分析用户地域对付费习惯的影响等。

背景信息

日志服务提供的这种跨数据源（OSS）的分析能力，可以帮助用户解决以下问题：

- 节省费用
  - 异构数据，根据数据的特性选择合适的存储系统，最大限度的节省成本。 对于更新少的数据，选择存放在OSS上，只需要支付少量的存储费用。如果存放在MySQL上，还要支付计算实例的费用。
  - OSS是阿里云的存储系统，可以走内网读取数据，免去了流量费用。

- 节省精力

  在我们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，节省了用户精力。

- 节省时间
  - 当用户需要分析数据时，使用一条SQL，秒级别即可获得结果。
  - 把常用的视图，定义成报表，打开即可看到结果。

操作步骤

1. 上传CSV文件到OSS。

   a. 定义一份属性文件，包含用户ID、用户昵称、性别、省份、年龄。

   ```
   userid,nick,gender,province,age
   1,阳光男孩,male,上海,18
   2,么么茶,female,浙江,19
   3,刀锋1937,male,广东,18
   ```

   b. 保存该文件为`user.csv`，使用osscmd上传到OSS。

   ```
   osscmd   put  ~/user.csv   oss:/testossconnector/user.csv
   ```

2. 定义外部存储。

   通过SQL定义一个存储名为user_meta虚拟外部表。

   ```
   * | create table user_meta ( userid bigint, nick varchar, gender
    varchar, province varchar, gender varchar,age bigint) with (
   endpoint='example.com',accessid='<youraccessid>',accesskey='<
   accesskey>',bucket='testossconnector',objects=ARRAY['user.csv'],type
   ='oss')
   ```

   说明：

   - 在SQL中，需要指定三部分信息：
     - 表的schema：包含的列，以及每一列的属性。
     - OSS访问信息：OSS的域名，accessid 和accesskey。
     - OSS文件信息：文件所属bucket，以及文件的object路径。

- objects是一个数组，可以同时包含多个文件。

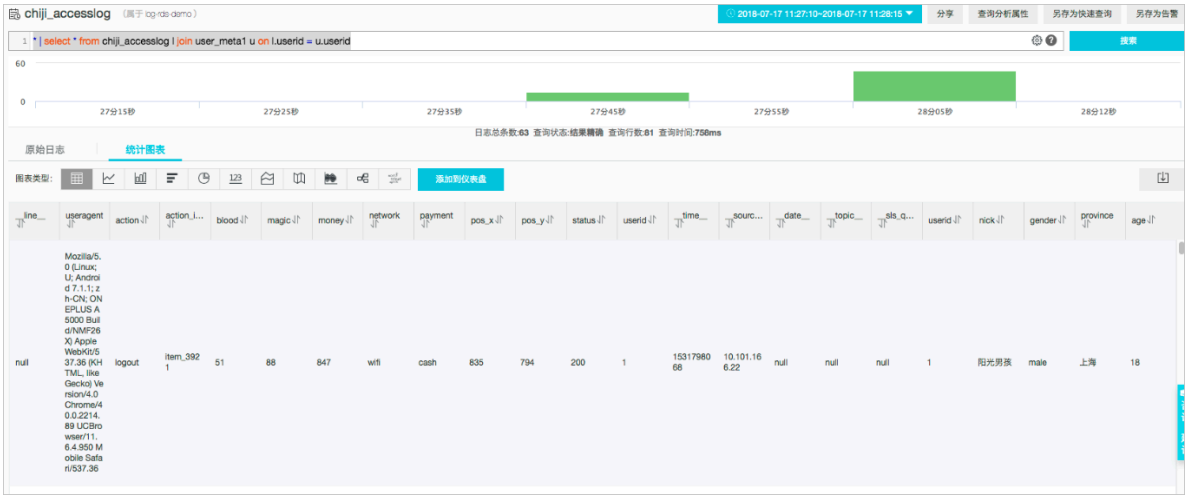
执行结果为true，表示执行成功。

执行SQL查看结果：`select * from user_meta`。

3. 联合分析。

在原始日志中，包含了用户的ID信息，我们可以通过SQL关联日志中的ID和OSS文件中的userid，补全日志的信息。

```
* | select * from chiji_accesslog l join user_meta1 u on l.userid = u.userid
```

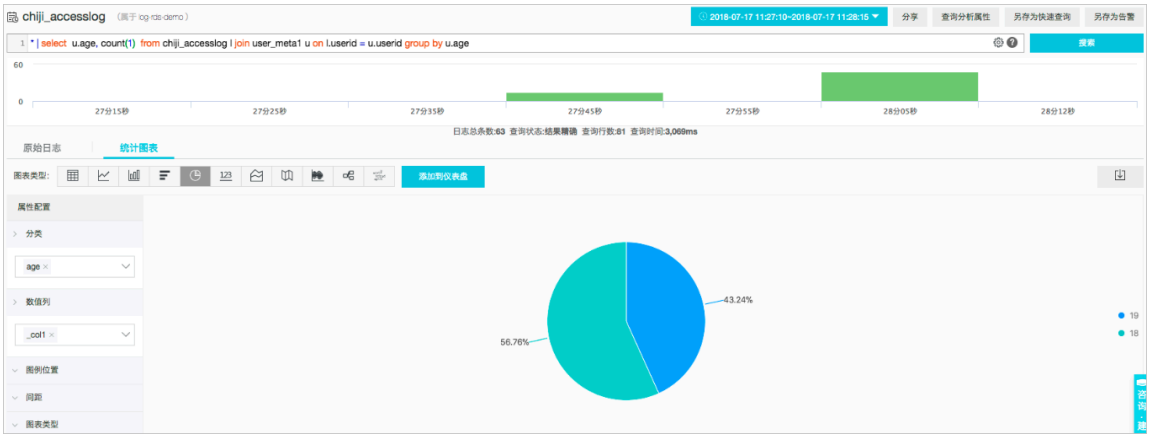


- 统计用户性别的访问情况：

```
* | select u.gender, count(1) from chiji_accesslog l join user_meta1 u on l.userid = u.userid group by u.gender
```

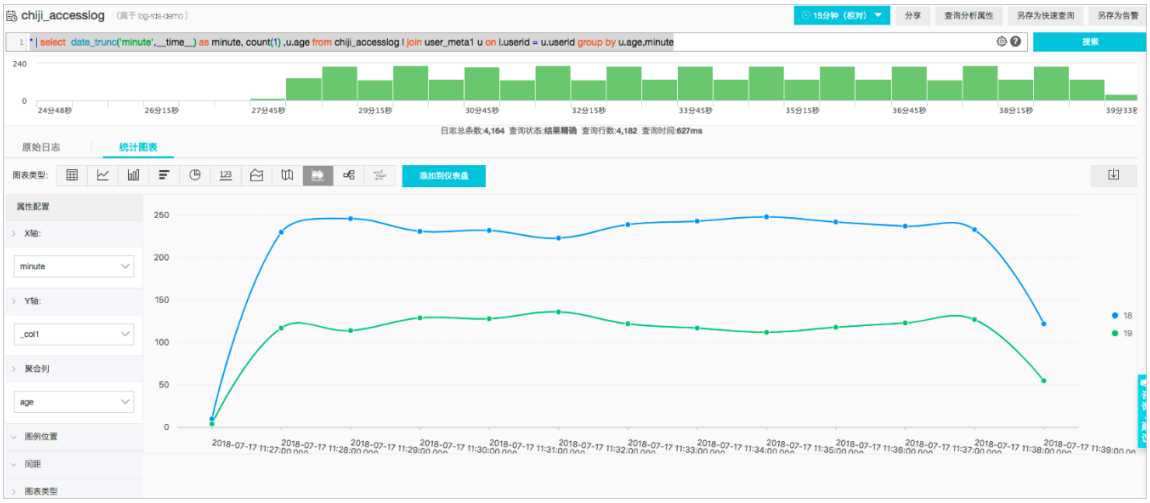
- 统计用户年龄的访问情况：

```
* | select u.age, count(1) from chiji_accesslog l join user_meta1
u on l.userid = u.userid group by u.age
```



- 统计不同年龄段在时间维度上的访问趋势：

```
* | selectdate_trunc('minute',__time__) as minute, count(1) ,u.age
 from chiji_accesslog l join user_meta1 u on l.userid = u.userid
group by u.age,minute
```



13.6 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：

```
__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: 060f0e0d08
0e0b05060307010c0f0209010e0e010c0a0605000606050b0c0400 pay_with:
alipay real_price: 52.0 suggest_price: 52.0
```

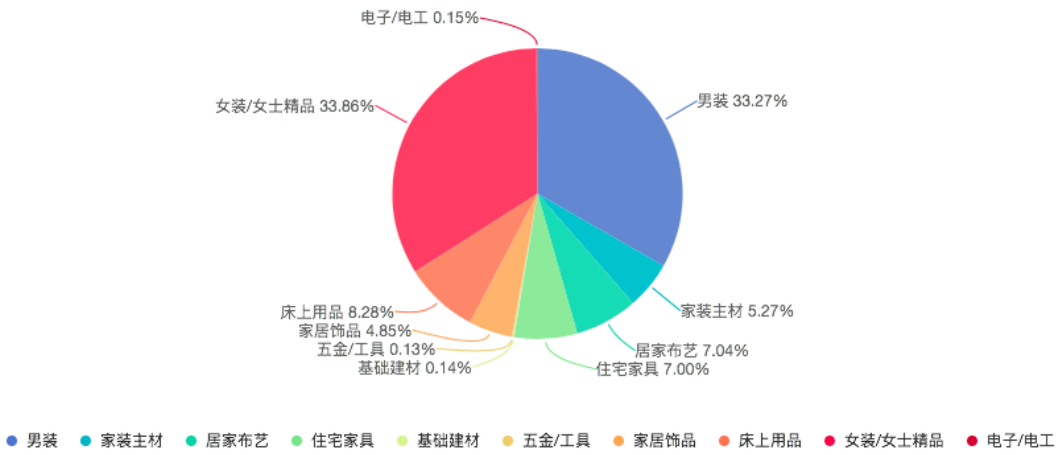
		时间	category	commodity	discount	member_discount	member_level	mobile	pay_with	real_price	suggest_price
1		08-31 20:12:02	住宅家具	小户型沙发盒家双人沙发两人咖啡厅卡座网吧沙发布艺单人沙发椅		3.19	gold	1398187088	cash	524.81	528.0
2		08-31 20:12:02	男装	天天特价秋季长袖衬衫韩版男士条纹衬衣潮流男装青年休闲格子寸衫		0.59	gold	1398185946	cash	42.41	43.0
3		08-31 20:12:02	男装	2016秋冬季新款男士嫌贝玻璃球服夹克古着两面穿偏但修身刺绣外套		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】九月质量 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秋季韩版宽松痛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#unique_15。

1. 查看产品的销售占比

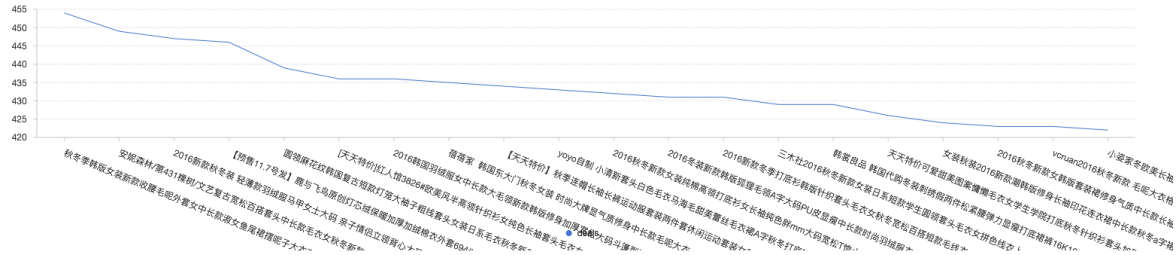
```
*|select count(1) as pv ,category group by category limit 100
```



2. 查看女装销售

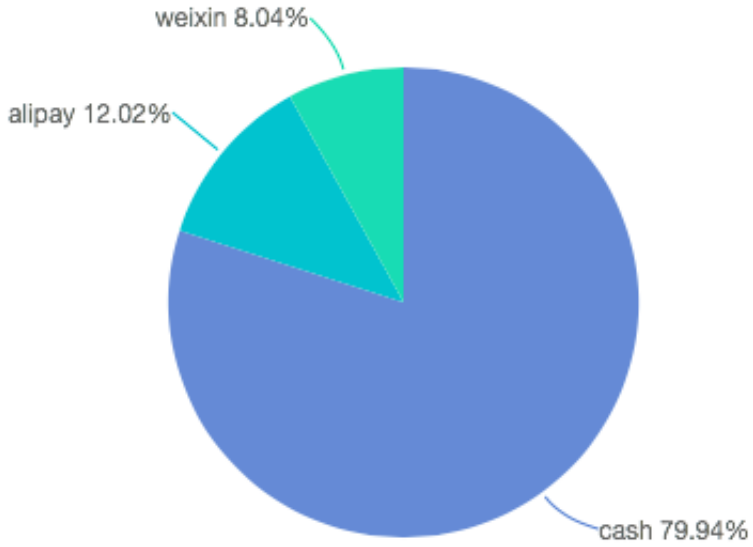
```
category: 女装/女士精品 | select count(1) as deals , commodity
```

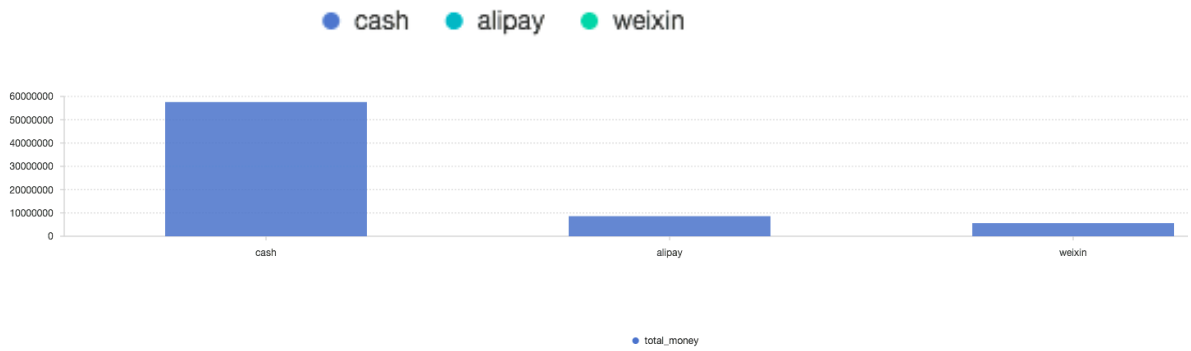
```
group by commodity order by deals  desc limit 20
```



3. 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  desc limit 20
```



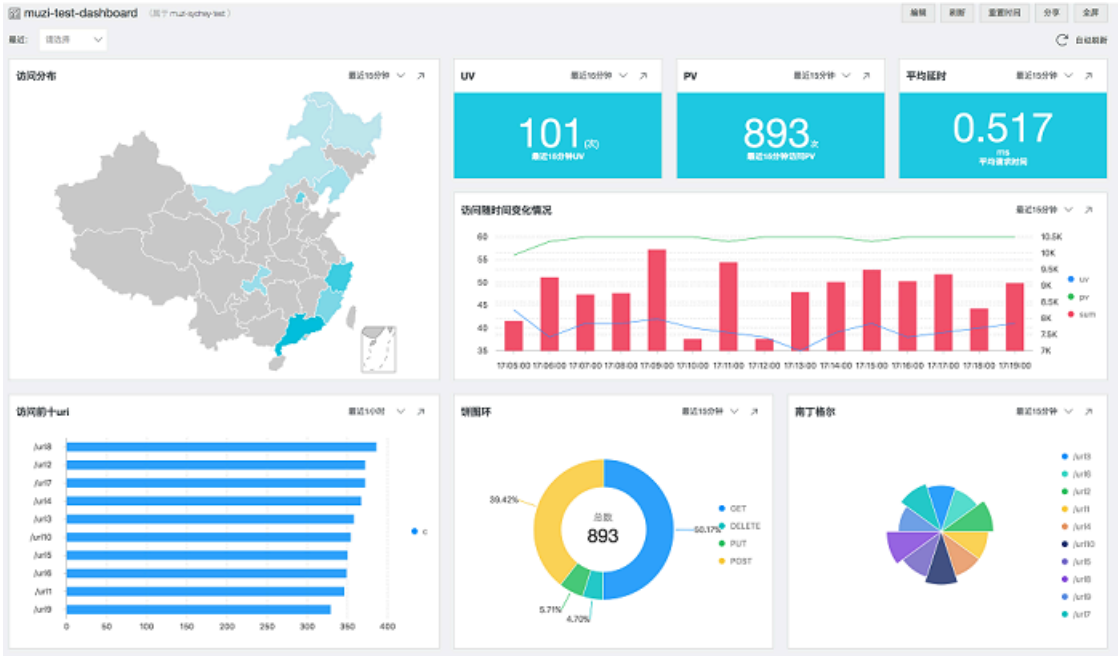


13.7 分析-网站日志

网站的访问日志信息对于个人站长来说至关重要，其中包含的PV、UV、访问地域分布以及访问前十页面等情况是网站访问情况的数据体现。应用的日志信息对于应用的开发者来说必不可少，针对Top方法的分析、优化可以直接提升应用质量。运维人员通过服务器日志可以监控数据、追溯异常，通过实时监控日志数据，可以获取最近1个小时的服务器响应时间变化、请求客户端负载均衡到某一台机器流量是否有异常情况等信息，并通过数据大屏展示日志监控数据，可以直观获取关键信息。

基于以上场景，日志服务提供多样化的日志数据采集、分析一站式解决方案，其中实时分析功能（LogSearch/Analytics），可以使用查询+SQL92语法对日志进行实时分析，并且可以在查询分析结果上支持自带Dashboard、DataV、Grafana、Tableua(通过JDBC)、QuickBI等可视化方式。

更为便捷的是，日志服务提供数据分析图表，即对日志的实时检索分析结果进行图表方式的直观展示，并通过仪表盘功能为您创建多种场景下的日志数据分析大盘。



点我试用

用户名：sls_reader1@**********

密码：pnX-32m-MHH-xbm

功能特点

- 无需事先定义：任何计算方法、任何过滤条件可以应用到任意时间段，秒级出图。
- 交互式分析：图表和原始日志无缝切换，双向打通。

· 场景化支持：通过数据接入向导直接生成分析大盘，无需复杂配置。

图标类型

目前，日志服务提供的可视化功能包含了如下图表类型：

流程架构

1. 数据采集。日志服务支持客户端、网页、协议、SDK/API（移动、游戏）等多种日志采集方式，所有采集方式均基于Restful API实现，除此之外您也可以通过API/SDK实现新的采集方式。
2. 设置数据索引并采用查询分析语法进行查询分析。
3. 可视化展示。日志服务提供基于Restful的开放式API，我们可以选择适合的方式对我们的日志数据进行可视化处理，本文档主要演示日志服务自带的可视化以及仪表盘（Dashboard）功能。

示例

本文档为您展示各种图形的典型应用场景。要对数据进行查询和分析，请先#unique_15。

1. 表格

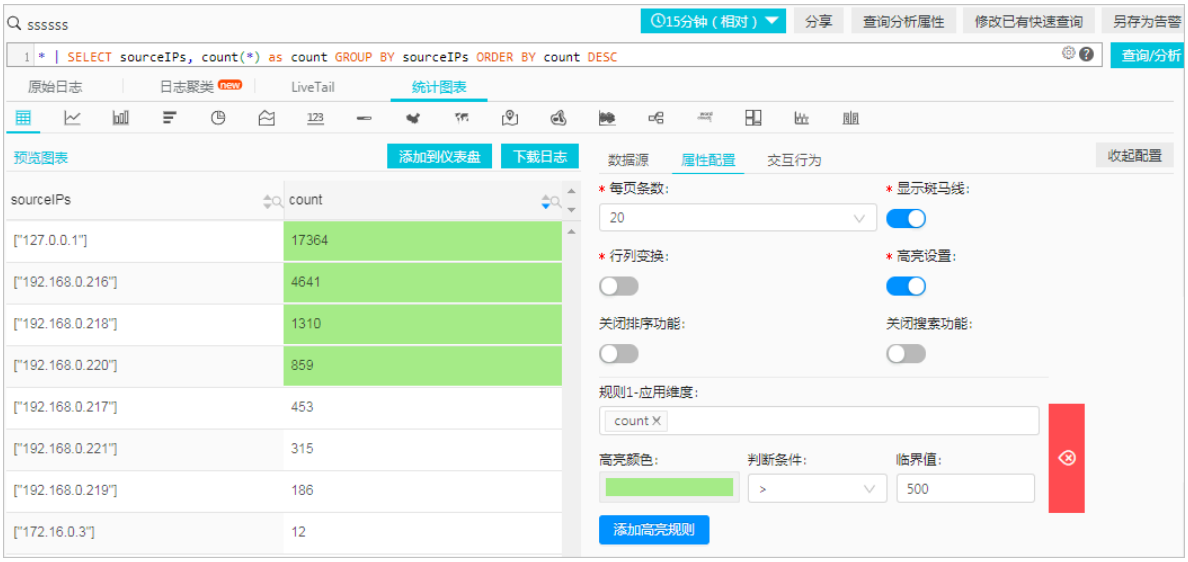
表格作为最常见的数据展示类型，由一组或多组单元格组成，用于显示数字和其他项以便快速引用和分析，表格中的项被组织为行和列，表格的第一行称为表头，指明表格每一列的内容和意义。

在日志服务中，我们通过查询分析语法得到的结果信息默认以表格方式进行展示。

例如，查看当前时间区间sourceIPs分布情况，并降序排列：

```
* | SELECT sourceIPs, count(*) as count GROUP BY sourceIPs ORDER BY
count DESC
```

表格结果如下所示，您可以利用表头上的排序按钮对某一列进行排序。



2. 折线图

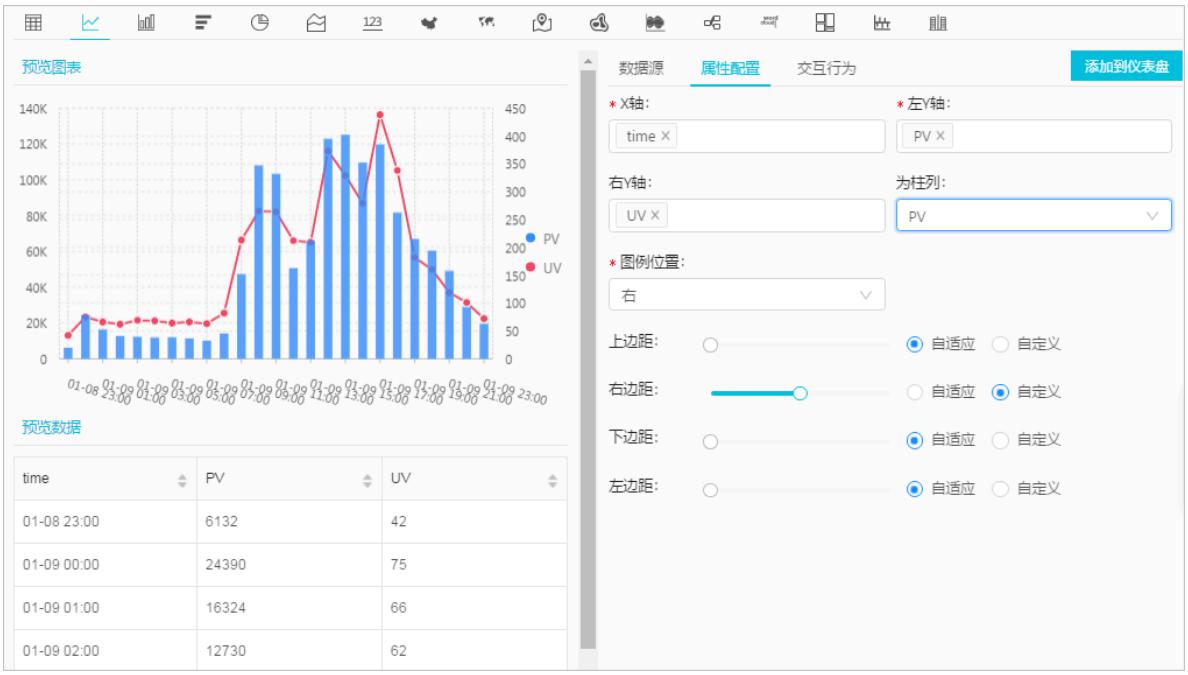
折线图属于趋势类分析图表，一般用于表示一组数据在一个有序数据类别（多为连续时间间隔）上的变化情况，用于直观分析数据变化趋势。

分析在最近15分钟内PV、UV以及平均响应时间的变化：

```
* | 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
```

选择minutes作为X轴，pv、uv放在左Y轴，avg为右Y轴并且设置uv为柱状，结果如下图所示：

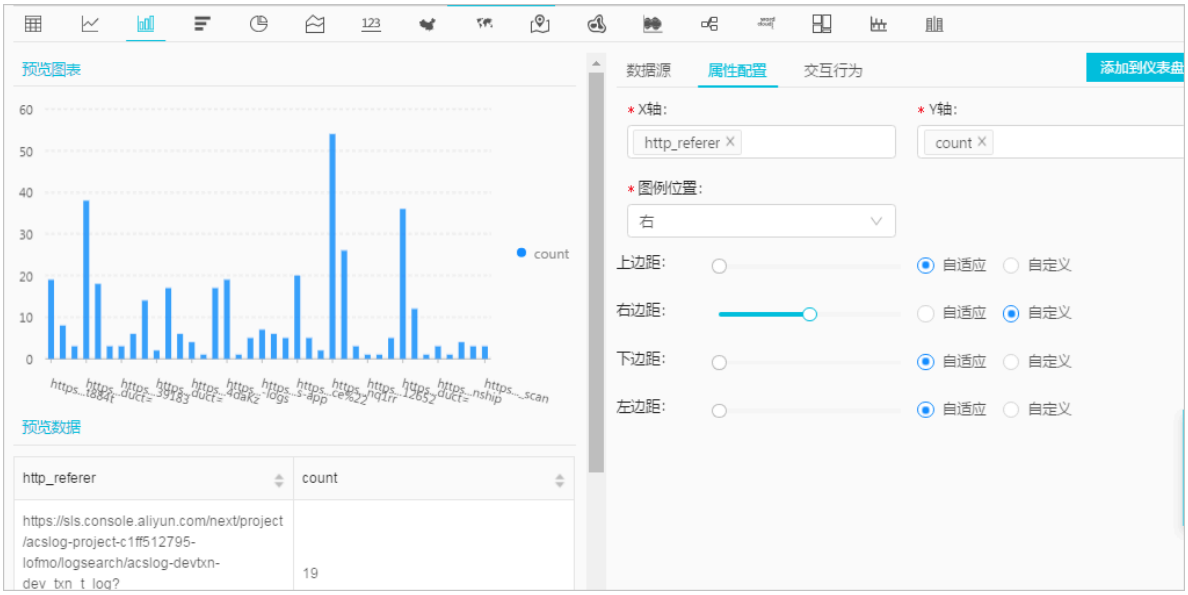


3. 柱状图

柱状图使用垂直或水平的柱子显示类别之间的数值比较，和#unique_234的不同之处在于，柱状图描述分类数据，并统计每一个分类中的数量，而折线图描述有序数据。

分析最近15分钟内不同`http_referer`的访问次数：

```
* | select http_referer, count(1) as count group by http_referer
```

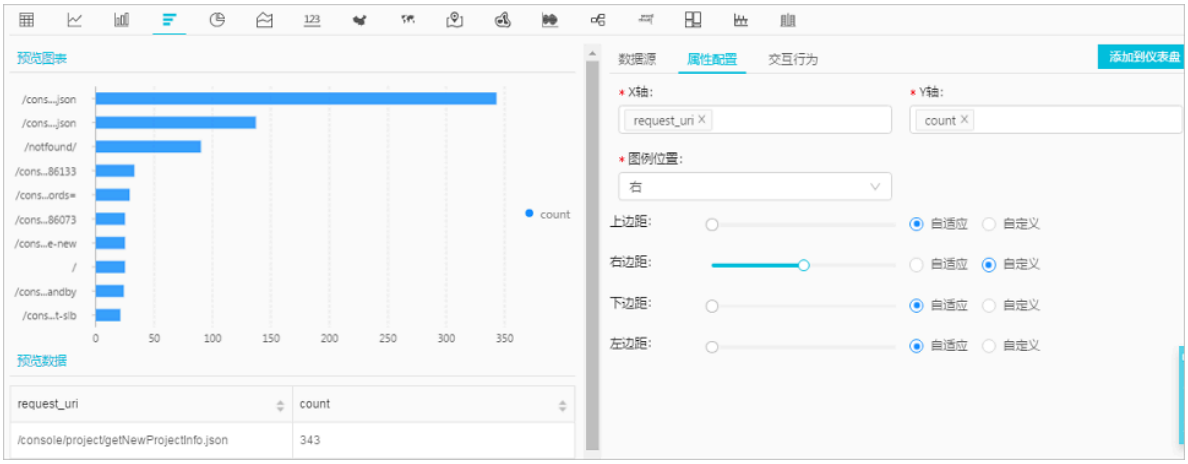


4. 条形图

条形图即为横向柱状图，适合分析分类数据的top情况。

分析最近15分钟内访问前十的`request_uri`：

```
* | select  request_uri, count(1) as count group by request_uri
order by count desc limit 10
```



5. 饼图

饼图用于表示不同分类的占比情况，通过弧度大小来对比各种分类。饼图通过将一个圆饼按照分类的占比划分成多个区块，整个圆饼代表数据的总量，每个区块（圆弧）表示该分类占总体的比例大小，所有区块（圆弧）的加和等于100%。

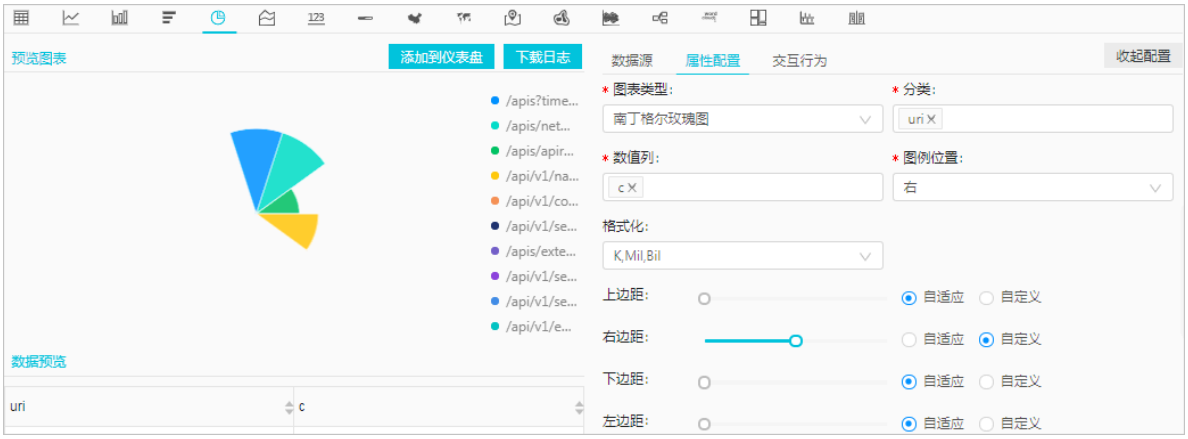
分析最近15分钟访问页面的分布：

```
* | select requestURI as uri , count(1) as c group by uri limit 10
```

饼图：

环图：

南丁格尔玫瑰图：

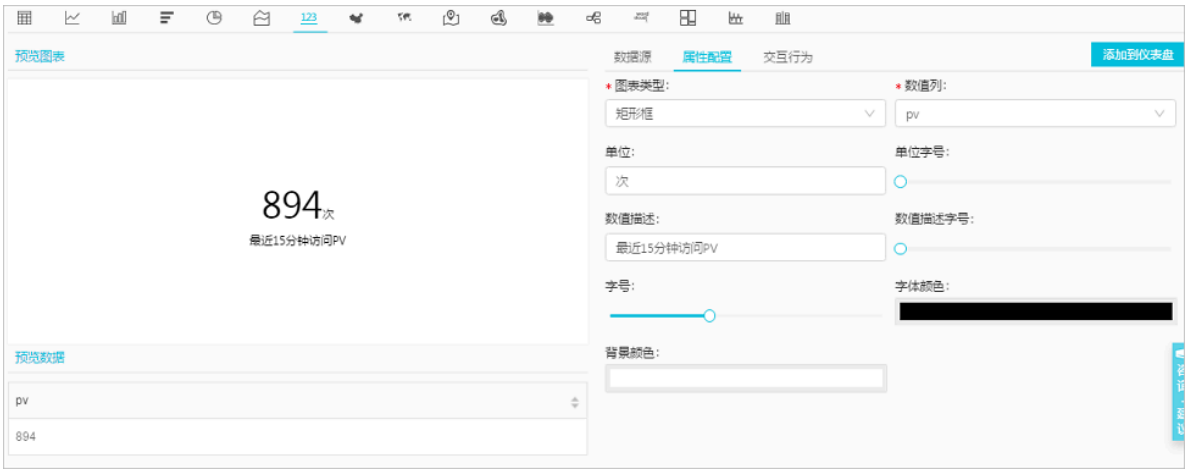


6. **单值图**

单值图作为最简单直接的数据表现形式，直观清晰地将某一个点上的数据展示出来，一般用于表示某一个时间点上的关键信息。

统计最近15分钟的PV：

```
* | select count(1) as PV
```

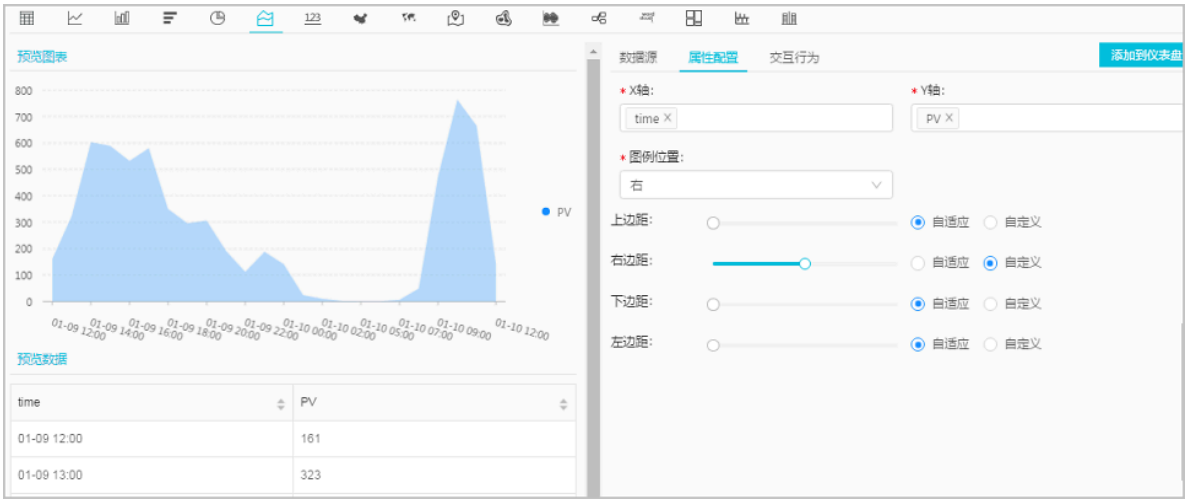


7. 面积图

面积图是在折线图的基础之上形成的，它将折线图中折线与坐标轴之间的区域使用颜色进行填充，这个填充即为面积，颜色的填充可以更好的突出趋势信息。

如统计`10.0.XX.XX`这个IP在最近1天内的访问情况：

```
remote_addr: 10.0.XX.XX | select date_format(date_trunc('hour',
__time__), '%m-%d %H:%i') as time, count(1) as PV group by time
order by time limit 1000
```



8. 地图

以地图作为背景，通过图形颜色、图像标记的方式展示地理数据信息。日志服务提供了三种地图方式，分别为：中国地图、世界地图以及高德地图（高德地图分为点图和热力图）。

通过remote_addr来绘制三种地图，统计前十的访问区域：

- 中国地图

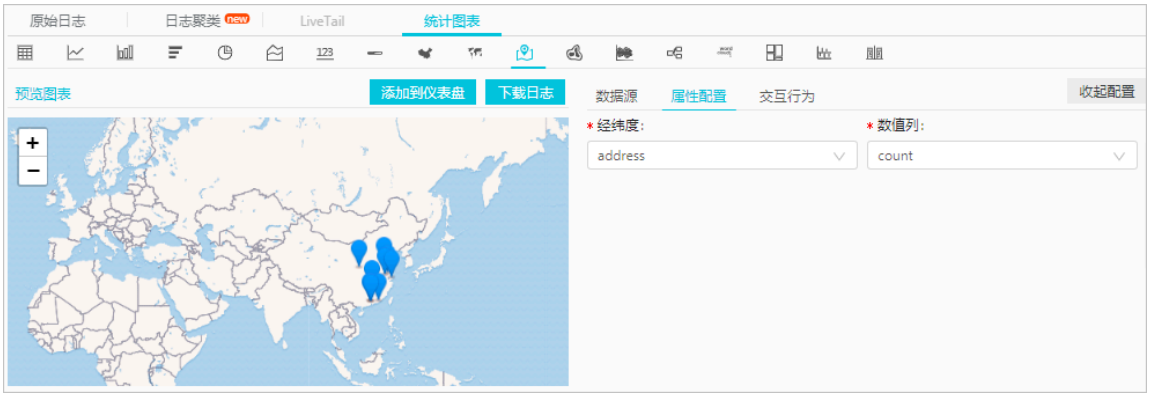
```
* | select ip_to_province(remote_addr) as address, count(1) as
count group by address order by count desc limit 10
```

- **世界地图**

```
* | select ip_to_country(remote_addr) as address, count(1) as
count group by address order by count desc limit 10
```

- **高德地图**

```
* | select ip_to_geo(remote_addr) as address, count(1) as count
group by address order by count desc limit 10
```


原始日志
日志聚类 new
LiveTail
统计图表
预览图表
添加到仪表盘
下载日志
数据源
属性配置
交互行为
收起配置
* 经纬度:
address
* 数值列:
count

原始日志
日志聚类 new
LiveTail
统计图表
预览图表
添加到仪表盘
下载日志
数据源
属性配置
交互行为
收起配置
* 经纬度:
address
* 数值列:
count

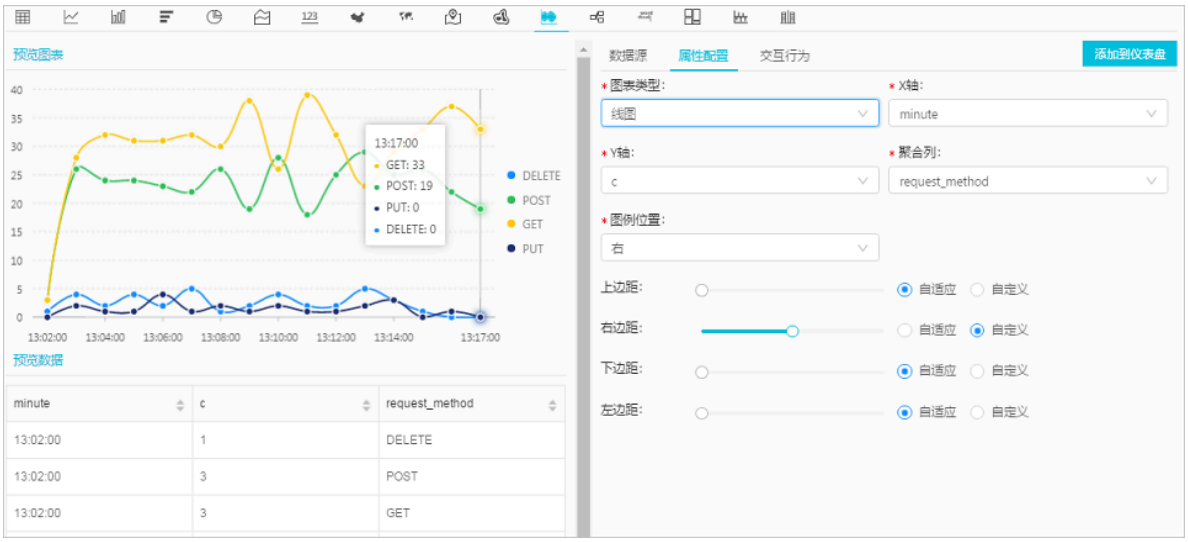
9. 流图

不同颜色的条带状分支代表了不同的分类信息，条状带的宽度映射了对应的数值大小。此外，原数据集中的时间属性，映射到X轴上，是一个三维关系的展现。

统计最近15分钟，不同method方法请求次数随时间变化趋势情况：

```
* | 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
```

X轴选择minute，Y轴选择c，按照request_method聚合。

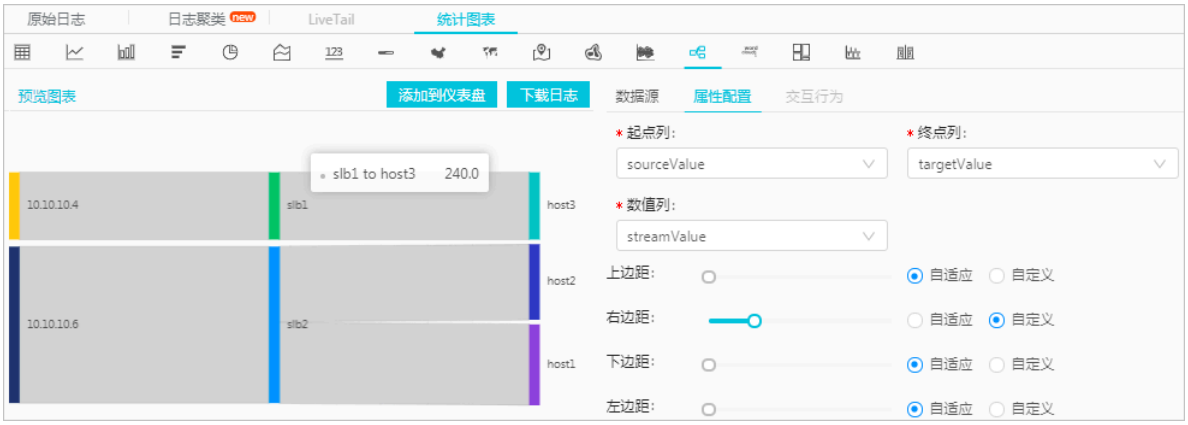


10.桑基图

桑基图 (Sankey Diagram)，是一种特定类型的流图，用于描述一组值到另一组值的流向。适合网络流量等场景，通常包含3组值source、target以及value。source和target描述了点的关系，而value描述了该source和target之间边的关系。

负载均衡场景示例：

```
* | select sourceValue, targetValue, streamValue group by sourceValu
e, targetValue, streamValue order by streamValue
```

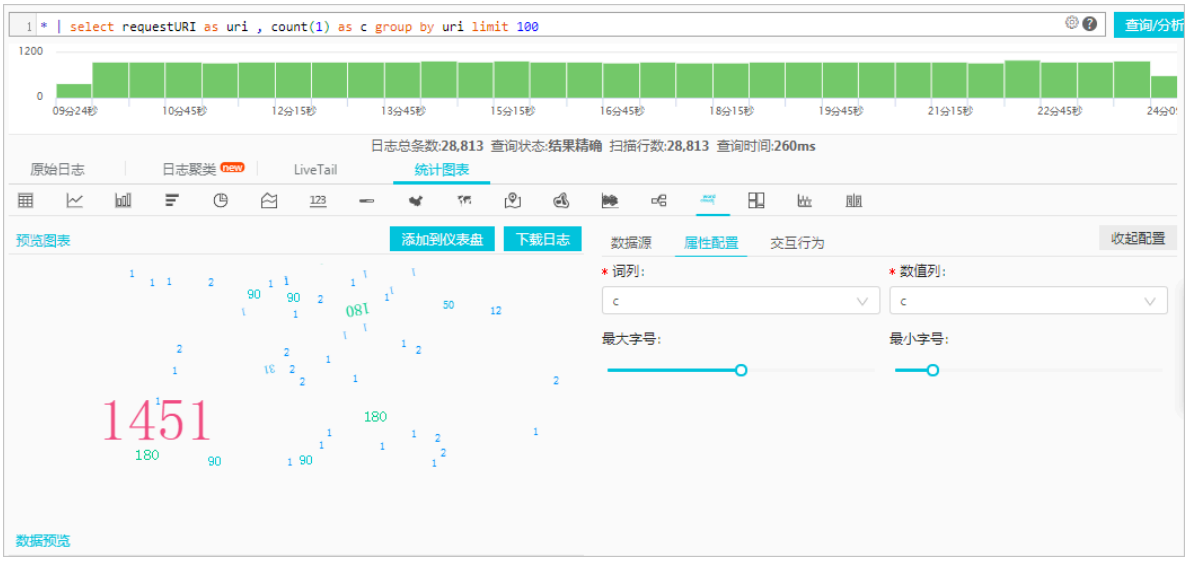


11.词云

词云，是文本数据的视觉表示，由词汇组成类似云的彩色图形，用于展示大量文本数据。每个词的重要性以字体大小或颜色显示，为您直观展示某一些关键词的权重大小。

统计最近15分钟访问requestURI的情况：

```
* | select requestURI as uri , count(1) as c group by uri limit 100
```



添加到仪表盘

所有通过查询分析语法获得的可视化图表都可以保存在一个仪表盘（Dashboard）中，再经过灵活的布局调整，就可以做出一张全面的仪表盘了。

您可以单击添加到仪表盘，建立一个仪表盘。建立仪表盘后可以通过标签快速打开仪表盘，实时查看数据。

演示视频：

13.8 分析-Nginx监控日志

Nginx和php-fpm、Docker、Apache等很多软件一样内建了一个状态页，对于Nginx的状态查看以及监控提供了很大帮助。本文档主要介绍通过日志服务Logtail采集Nginx status信息，并对采集的status信息进行查询、统计、搭建仪表盘、建立自定义报警，对您的Nginx集群进行全方位的监控。

环境准备

请按照以下步骤，开启Nginx status插件。

1. 确认Nginx是否具备status功能。

   执行以下命令查看Nginx是否具备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的配置文件（默认为/etc/nginx/nginx.conf）中开启status功能，样例配置如下：

   ```
       location /private/nginx_status {
         stub_status on;
         access_log   off;
         allow 10.10.XX.XX;
         deny all;
       }
   ```

   说明：

   该配置只允许ip为`10.10.XX.XX`的机器访问`nginx status`功能。

3. 验证Logtail安装的机器具有`nginx status`访问权限。

   可通过如下命令测试：

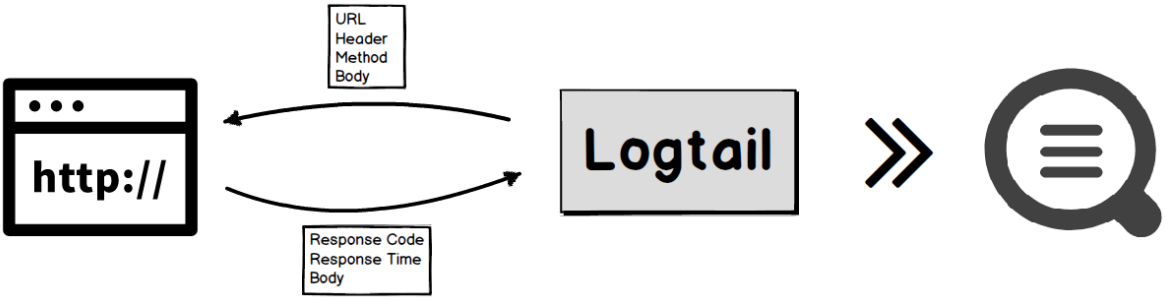
   ```
   $curl http://10.10.XX.XX/private/nginx_status
   Active connections: 1
   server accepts handled requests
   2507455 2507455 2512972
   Reading: 0 Writing: 1 Waiting: 0
   ```

数据采集

1. 安装Logtail

   安装logtail，确认版本号在0.16.0及以上。若低于0.16.0版本请根据文档提示升级到最新版本。

2. 填写采集配置



a. 在日志服务控制台创建一个新的Logstore，采集向导中选择自建软件中的Nginx监控。

b. 根据提示配置Nginx监控的URL以及相关参数（基于http采集功能实现）。

> 说明:
>
> · 将样例配置中Addresses字段内容修改为您需要监控的url列表。
>
> · 如果您的Nginx status返回的信息和默认的不同，请修改processors用以支持http的body解析。

样例配置如下：

```
{
"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
        }
    }
]
}
```

数据预览

应用配置1分钟后，点击预览可以看到状态数据已经成功采集。Logtail的http采集除了将body解析上传，还会将url、状态码、方法名、响应时间、是否请求成功一并上传。

说明:

若无数据，请先检查配置是否为合法json。

```
_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
```

查询分析

要对数据进行查询和分析，请先#unique_15。

自定义查询

查询相关帮助文档参见#unique_237。

1. 查询某一ip的status信息： `_address_ : 10.168.0.0`
2. 查询响应时间超过100ms的请求: `_response_time_ms_ > 100`
3. 查看状态码非200的请求: `not _http_response_code_ : 200`

统计分析

统计分析语法参见#unique_13。

- 每5分钟统计 waiting reading writing connection 平均值：

```
*| select  avg(waiting) as waiting, avg(reading)  as reading,  avg
(writing)  as writing,  avg(connection)  as connection,  from_unixt
ime( __time__ - __time__ % 300) as time group by __time__ - __time__
 % 300 order by time limit 1440
```

```
```

- 统计top 10的 waiting：

```
*| select  max(waiting) as max_waiting, address, from_unixtime(max(
__time__)) as time group by address order by max_waiting desc limit
10
```

- 目前Nginx总数以及invalid数量：

```
* | select  count(distinct(address)) as total
```

```
not _result_ : success | select  count(distinct(address))
```

- 最近 top 10 失败的请求：

```
not _result_ : success | select _address_ as address, from_unixtime(
__time__) as time  order by __time__ desc limit 10
```

- 每5分钟统计统计请求处理总数：

```
*| select  avg(handled) * count(distinct(address)) as total_hand
led, 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
```

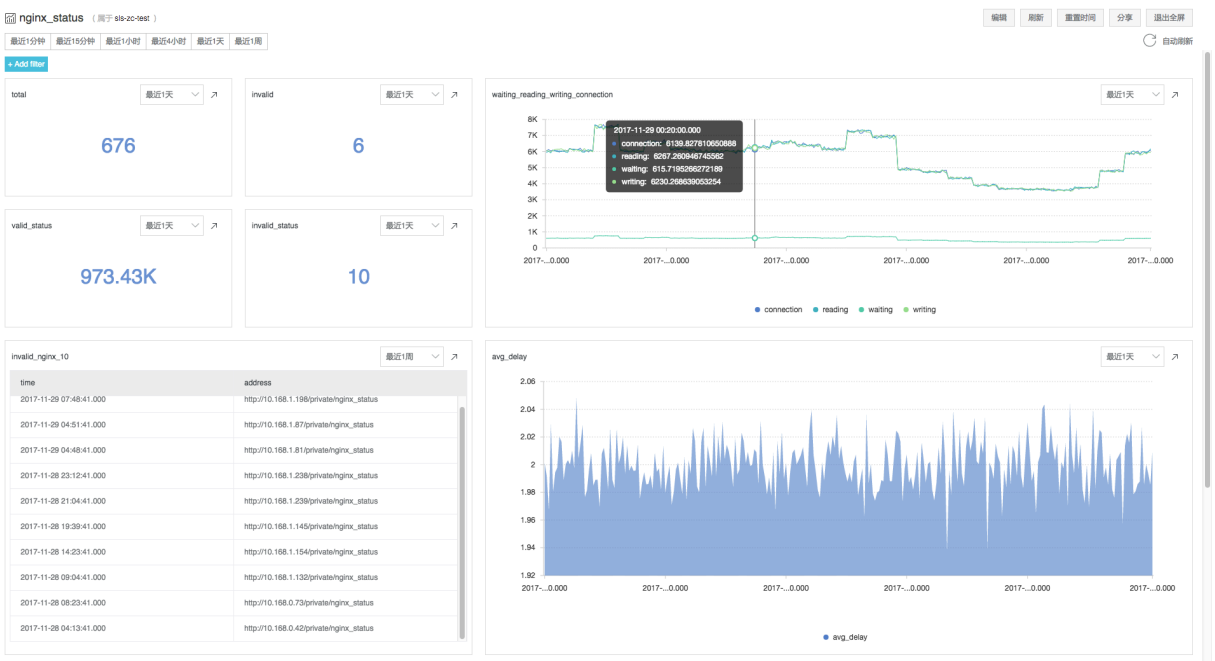
- 请求有效数/无效数：

```
not _http_response_code_ : 200  | select  count(1)
```

```
_http_response_code_ : 200  | select  count(1)
```

仪表盘

日志服务默认对于Nginx监控数据提供了仪表盘，您可以在Nginx status的仪表盘，仪表盘搭建参见#unique_56。



设置报警

1. 将以下查询另存为快速查询，名称为`invalid_nginx_status`：`not _http_response_code_ : 200 | select count(1) as invalid_count`。
2. 根据该快速查询#unique_59，样例如下：

配置	取值
告警名称	invalid_nginx_alarm
添加到仪表盘	nginx_status
图表名称	total
查询语句	* | select count(distinct(__source__)) as total
查询区间	15分钟
执行间隔	15分钟
触发条件	total>100
触发通知阈值	1
通知类型	通知中心
通知内容	nginx status 获取异常，请前往日志服务查看具体异常信息，project : xxxxxxxx, logstore : nginx_status

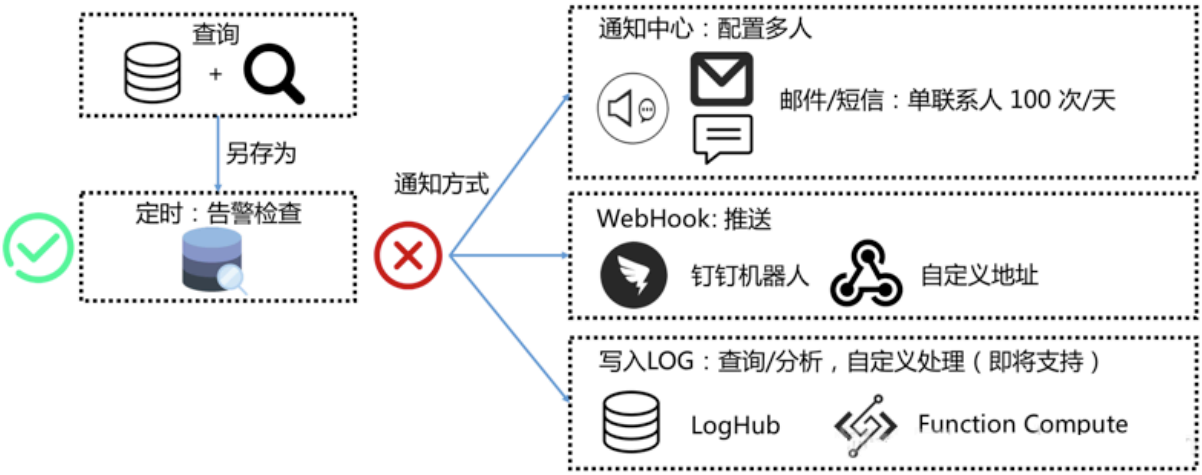
13.9 分析-Nginx访问日志

目前，日志服务支持保存查询语句为快速查询，对查询设置触发周期（间隔），并对执行结果设定判断条件并且告警。您还可以设置告警动作，即定期运行的快速查询结果一旦触发告警条件，将以何种方式通知您。

目前支持通知方式有以下3种：

- 通知中心：在阿里云通知中心可以设置多个联系人，通知会通过邮件和短信方式发送。
- WebHook：包括钉钉机器人，及自定义WebHook等。
- （即将支持）写回日志服务（logstore）：可以通过流计算，函数服务进行事件订阅；也可以对告警生成视图和报表。

告警功能配置与操作请参考#unique_239。 除通过日志服务监控并告警外，您还可以通过云监控产品对日志服务各项指标进行监控，并在触发告警条件时为您发送提醒消息。



实践场景

本文以Nginx日志为例，为您示范如何使用日志服务对采集到的日志信息定时查询分析，并通过日志查询结果判断以下业务问题：

- 是否有错误。
- 是否有性能问题。
- 是否有流量急跌或暴涨。

准备工作（Nginx日志接入）

1. 采集日志数据。

   a. 在概览页面单击接入数据，并选择NGINX-文本日志。

   b. 选择日志空间。

      如果您是通过日志库下的数据接入后的加号进入采集配置流程，系统会直接跳过该步骤。

   c. 创建机器组。

      在创建机器组之前，您需要首先确认已经安装了Logtail。

      - 集团内部机器：默认自动安装，如果没有安装，请根据界面提示进行咨询。
      - ECS机器： 勾选实例后单击安装进行一键式安装。Windows系统不支持一键式安装，请参考#unique_240手动安装。
      - 自建机器：请根据界面提示进行安装。或者参考#unique_241或#unique_242文档进行安装。

      安装完Logtail后单击确认安装完毕创建机器组。如果您之前已经创建好机器组，请直接单击使用现有机器组。

   d. 机器组配置。

      选择一个机器组，将该机器组从源机器组移动到应用机器组。

   e. 填写配置名称，日志路径，Nginx日志配置，NGINX键名称，并酌情配置高级选项。

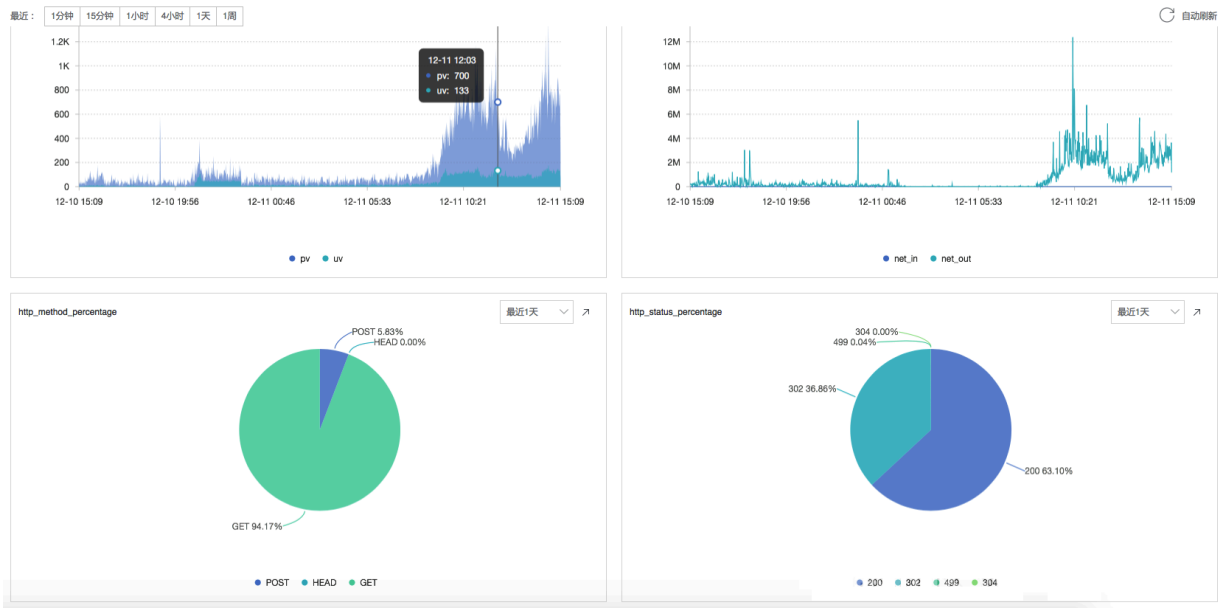
   f. 单击下一步，进入配置索引步骤。

2. 查询分析设置。

   详细内容请参考#unique_15与可视化或最佳实践网站日志分析案例。

3. 对关键指标设置视图和告警。

Sample视图：



操作步骤

1. 判断是否有错误

   错误一般有这样几类：404（请求无法找到地址）/502/500（服务端错误），我们一般只需关心500（服务端错误）。

   判断是否有500错误，您可以使用以下query统计单位时间内错误数c，并将报警规则设置为c > 0 则发送告警。

   ```
   status:500 | select count(1) as c
   ```

   这种方式比较简单，但往往过于敏感，对于一些业务压力较大的服务而言有零星几个500是正常的。为了应对这种情况，您可以在告警条件中设置触发次数为2次，即只有连续2次检查都符合条件后再发告警。

2. 判断是否有性能问题

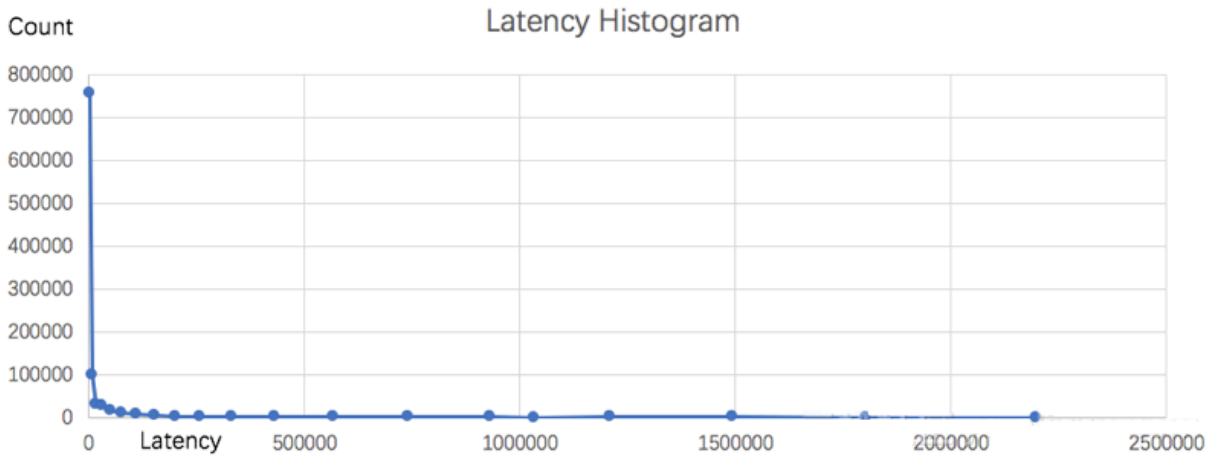
   服务器运行过程中虽然没有错误，但有可能会出现延迟（Latency）增大情况，您可以针对延迟进行告警。

例如您可以通过以下方式计算某个接口（“/adduser"）所有写请求（”Post “）延时。告警规则设置为 l > 300000 即当平均值超过300ms后告警。

```
Method:Post and URL:"/adduser" | select avg(Latency) as l
```

利用平均值来报警简单而直接，但这种方法往往会使得一些个体请求延时被平均掉，无法反馈问题。例如，对该时间段的Latency计算一个数学上的分布，即划分20个区间，计算每个区间内的数目，从分布图上可以看到大部分请求延时非常低（<20ms），但最高的延时有2.5s。

```
Method:Post and URL:"/adduser" | select numeric_histogram(20, Latency)
```



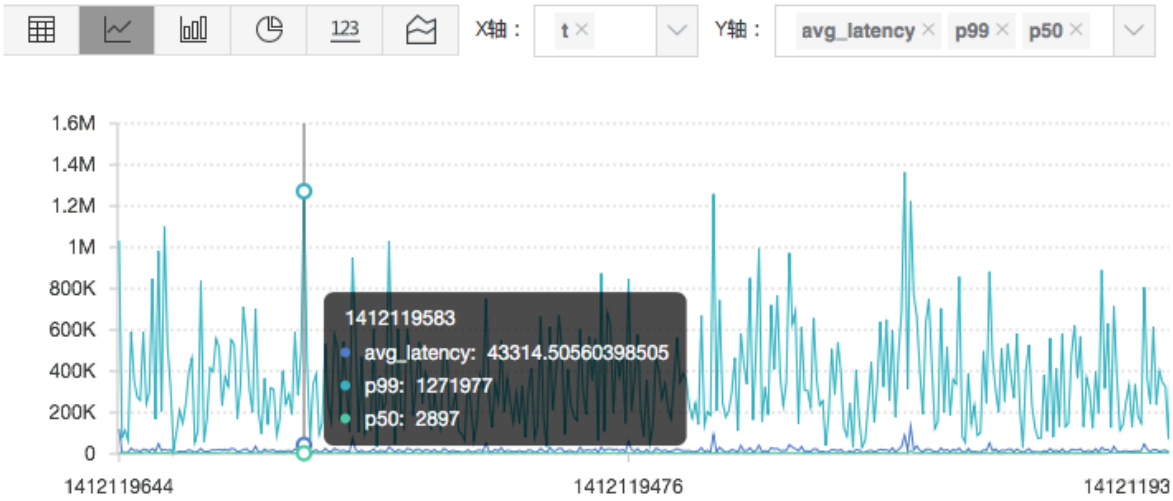
为应对这种情况，您可以用数学统计中的百分数（99%最大延时）来作为报警条件，这样既可以排除偶发的延时高引起误报，也能对整体的演示更有代表性。以下的语句计算了99%分位的延时大小

approx_percentile(Latency, 0.99)，同样您也可以修改第二个参数进行其他分位的划分，例如中位数的请求延时 approx_percentile(Latency, 0.5)。

```
Method:Post and URL:"/adduser" | select approx_percentile(Latency, 0.
99) as p99
```

在监控的场景中，您也可以在一个图上绘出平均延时，50%分位延时，以及90%分位延时。以下是按一天的窗口（1440分钟）统计各分钟内延时的图：

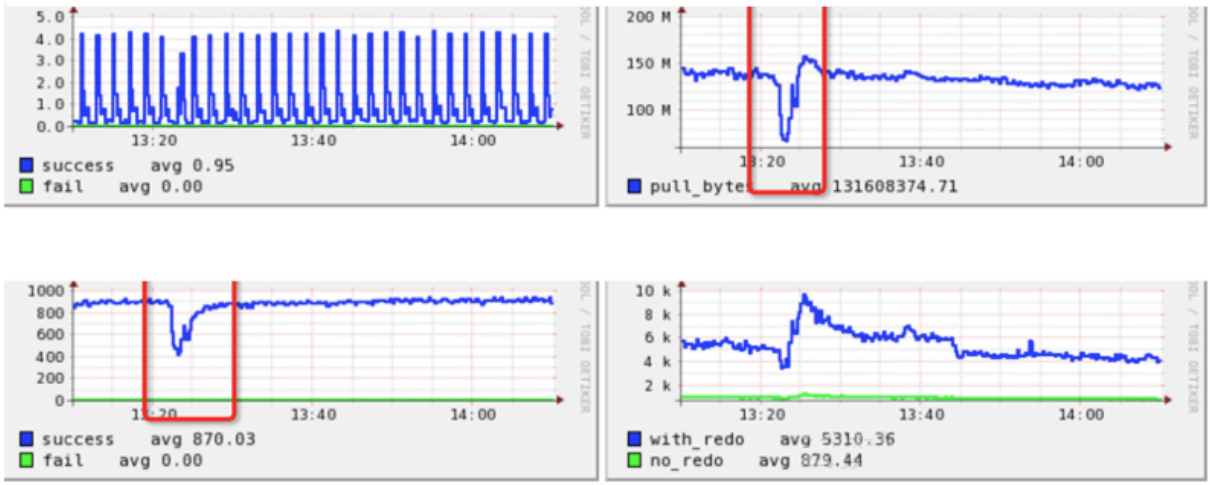
```
* | select avg(Latency) as l, approx_percentile(Latency, 0.5) as p50,
approx_percentile(Latency, 0.99) as p99, date_trunc('minute', time) as
 t group by t order by t desc limit 1440
```



3. 判断是否有流量急跌或暴涨

服务器端自然流量一般符合概率上的分布，会有一个缓慢上涨或下降过程。流量急跌或暴涨表示短时间内流量变化非常大，一般都是不正常的现象，需要留意。

如以下监控图所示，在2分钟时间内流量大小下跌30%以上，在2分钟内后又迅速恢复。



急跌和暴涨一般会有如下参考系：

- 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。

本文以第一种情况为例，计算流量infow数据的变动率，也可以换成QPS等流量。

3.1 首先定义一个计算窗口

定一个1分钟的窗口，统计该分钟内的流量大小，以下是一个5分钟区间统计：

```
* | 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
```

从结果分布上看，每个窗口内的平均流量 `sum(inflow)/(max(__time__)-min(__time__))` 应该是均匀的。

window_time	inflow
1513045740	315574947
1513045800	333233937
1513045860	335821584
1513045920	330556452
1513045980	316785257

3.2 计算窗口内的差异值（最大值变化率）

这里我们会用到子查询，我们写一个查询，从上述结果中计算最大值或最小值与平均值的变化率（这里的max_ratio），例如如下计算结果max_ratio为 1.02。我们可以定义一个告警规则，如果max_ratio > 1.5（变化率超过50%）就告警。

```
 * | 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 order by window_time
limit 15)
```

window_time	inflow
1513045740	315574947
1513045800	333233937
1513045860	335821584
1513045920	330556452
1513045980	316785257

最大值

3.3 计算窗口内的差异值（最近值变化率）

在一些场景中我们更关注最新的数值是否有波动（是否已经恢复），那可以通过max_by方法获取最大windows_time中的流量来进行判断，这里计算的最近值为lastest_ratio=0.97。

```
 * | select max_by(inflow, window_time)/1.0/avg(inflow) as lastest_ra
tio 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)
```

说明:

这里的max_by函数计算结果为字符类型，我们需要强转成数字类型。 如果要计算变化相对率，可以用（1.0-max_by(inflow, window_time)/1.0/avg(inflow)) as lastest_ratio代替。

window_time	inflow
1513045740	315574947
1513045800	333233937
1513045860	335821584
1513045920	330556452
1513045980	316785257

最近值

3.4 计算窗口内的差异值（定义波动率，上一个值与下一个变化率）

波动率另外一种计算方法是数学上一阶导数，即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

Diff

使用窗口函数(lag)进行计算，窗口函数中提取当前inflow与上一个周期inflow "lag(inflow, 1, inflow)over() " 进行差值，并除以当前值作为一个变化比率：

```
 * | 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)
```

在示例中，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）：

如果要定义一个绝对变化率，可以使用abs函数（绝对值）对计算结果进行统一。

总结

日志服务查询分析能力是完整SQL92，支持各种数理统计与计算等，只要会用SQL都能进行快速分析，欢迎尝试！

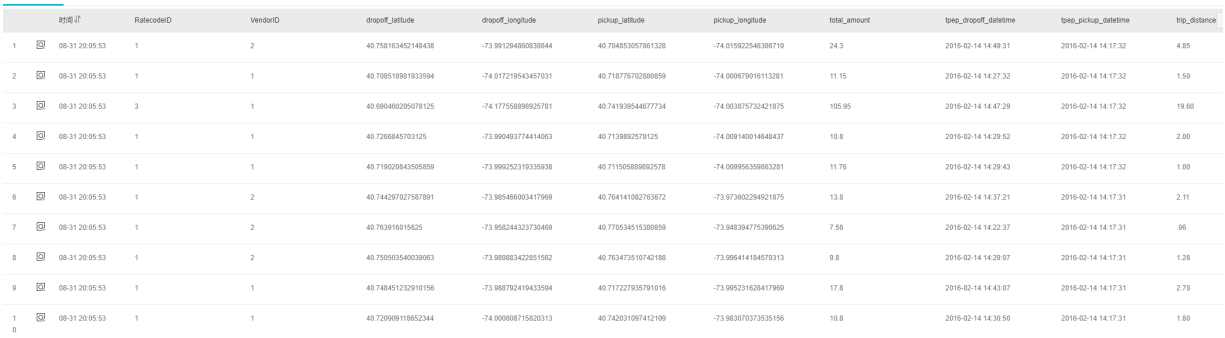
13.10 分析-行车轨迹日志

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。

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。

数据样例：

```
RatecodeID:  1VendorID:  2__source__:  11.164.232.105    __topic__:
  dropoff_latitude:  40.743995666503906    dropoff_longitude:  -73.
983505249023437extra:  0    fare_amount:  9    improvement_surcharg
e:  0.3    mta_tax:  0.5    passenger_count:  2    payment_type:
1    pickup_latitude:  40.761466979980469    pickup_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:  1455418993    tpep_picku
p_datetime:  2016-02-14 10:53:57    tpep_pickup_time:  1455418437
trip_distance:  2.02
```



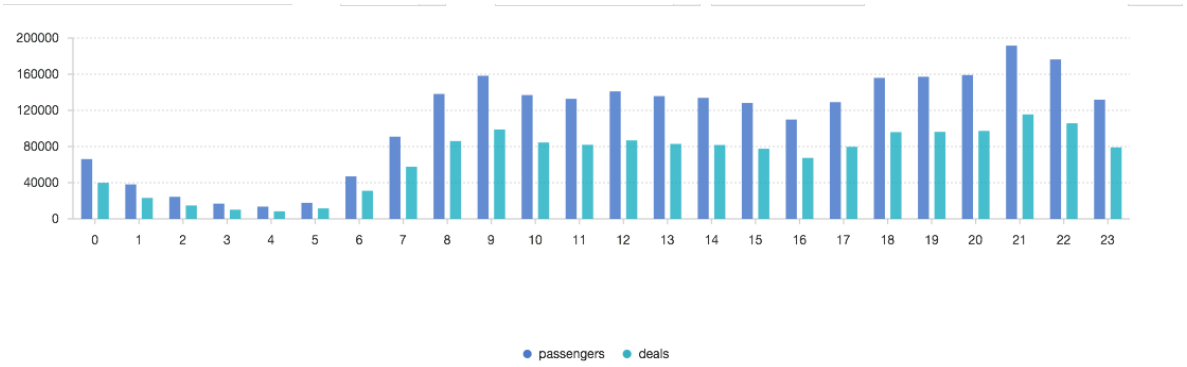
	时间	RatecodeID	VendorID	dropoff_latitude	dropoff_longitude	pickup_latitude	pickup_longitude	total_amount	tpep_dropoff_datetime	tpep_pickup_datetime	trip_distance
1	08-31 20:05:53	1	2	40.758163452148438	-73.991294860839844	40.704853057861328	-74.015922546386719	24.3	2016-02-14 14:49:31	2016-02-14 14:17:32	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	40.726684570312 5	-73.990493774414063	40.713989257812 5	-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#unique_15。

1. 分时段乘车人次，查看哪些时段比较热门

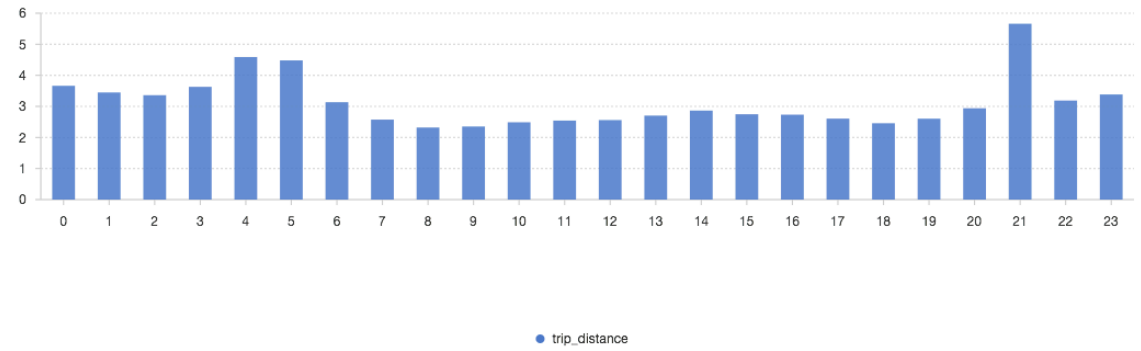
```
 *| select count(1) as deals, sum(passenger_count) as passengers,

 (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。

2. 分时段平均乘车里程

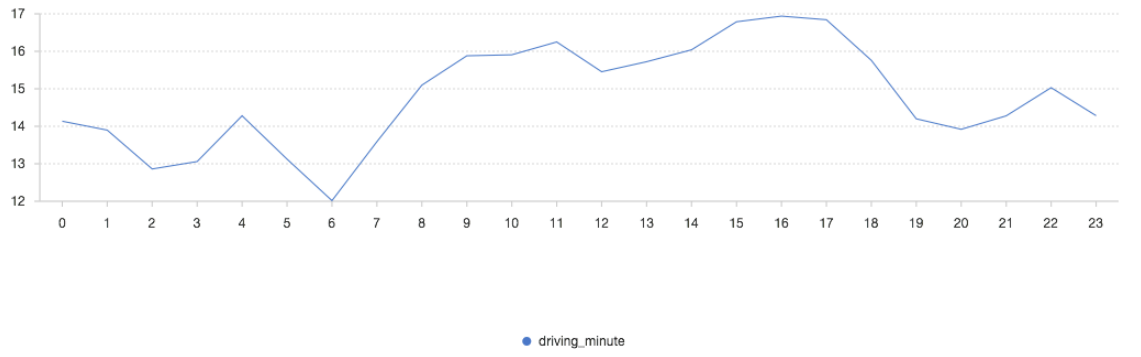
```
*| select  avg(trip_distance)  as trip_distance,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit
 24
```



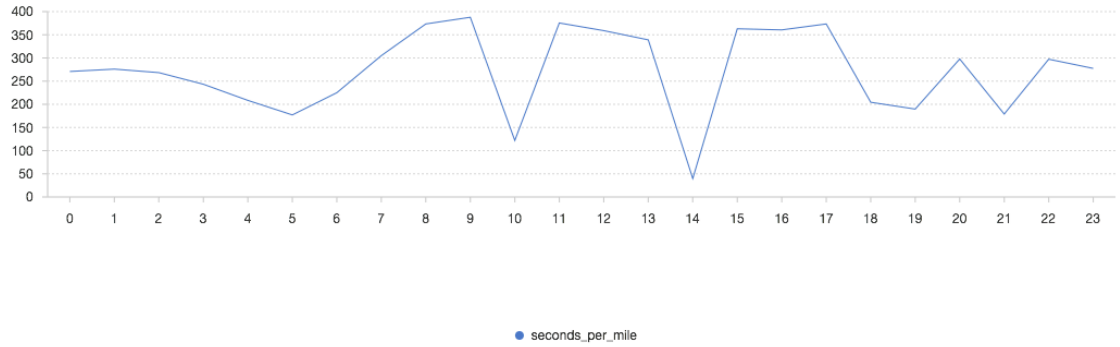
某些时刻，对乘车里程的需求也挺旺盛，出租车公司在对应的时候也需要准备更多的车辆。

3. 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_mi
nutes,
(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
```



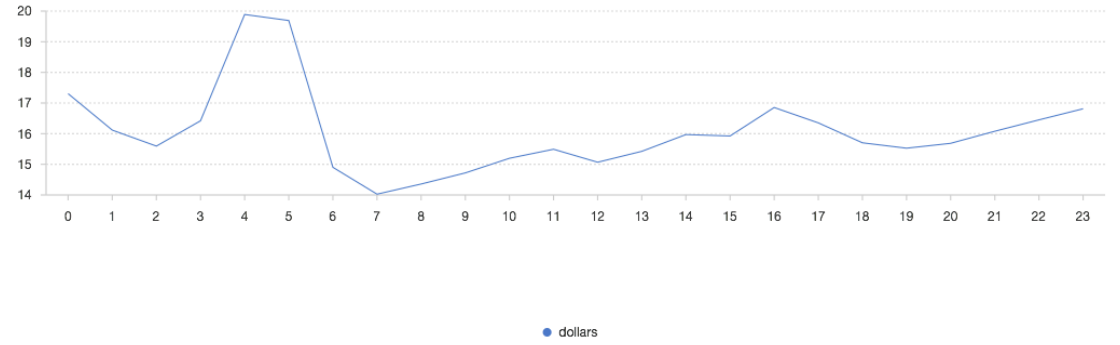

```
*| select  sum(tpep_dropoff_time-tpep_pickup_time)/sum(trip_distance
)  as driving_minutes,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit
 24
```



一些时刻特别堵，需要准备更多车辆来应对需求。

4. 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
```

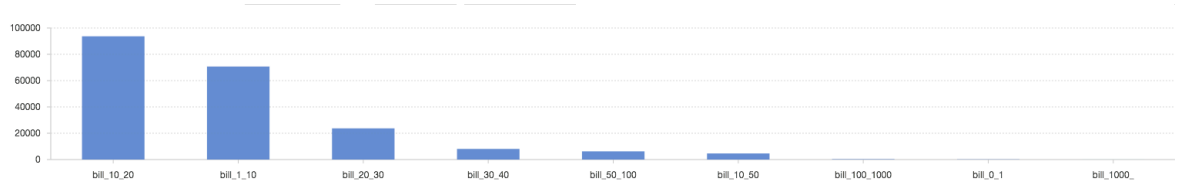


凌晨4点钟的客单价比较高，有经济压力的驾驶员可以选择在这个时候提供服务。

5. 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。

13.11 分析-分析SLB七层访问日志

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，通过消除单点故障提升应用系统的可用性。负载均衡对于大部分云上架构来说都是基础设施组件，因此，对SLB持续的监控、探测、诊断和报告是一个强需求。用户一般可以通过云厂商内置的监控报表来了解SLB实例运行状况。

SLB访问日志功能当前支持基于HTTP/HTTPS的七层负载均衡，访问日志内容丰富，提供近30个字段，例如：收到请求的时间、客户端的IP地址、处理Latency、请求URI、后端RealServer（阿里云ECS）地址、返回状态码等。完整字段及功能说明请参考负载均衡7层访问日志功能。

本文档基于阿里云日志服务的可视化和日志实时查询（OLTP+OLAP）能力，为您介绍SLB七层访问日志的实时采集、查询与分析最佳方案，并举例说明SLB实例的一些典型报表统计、日志查询分析的方法。该方案结合了可视化报表和查询分析引擎，可以实时、交互分析SLB实例状况。

目前SLB7层访问日志已在所有区域开放，欢迎使用。

前提条件

1. 已开通日志服务与SLB七层负载均衡。
2. 已成功采集到SLB七层负载均衡日志。详细步骤请参考采集步骤。

可视化分析

业务概览

负载均衡支持RealServer的水平扩展和故障冗余恢复，为应用提供大规模、高可靠的并发web访问服务支撑。

说明:

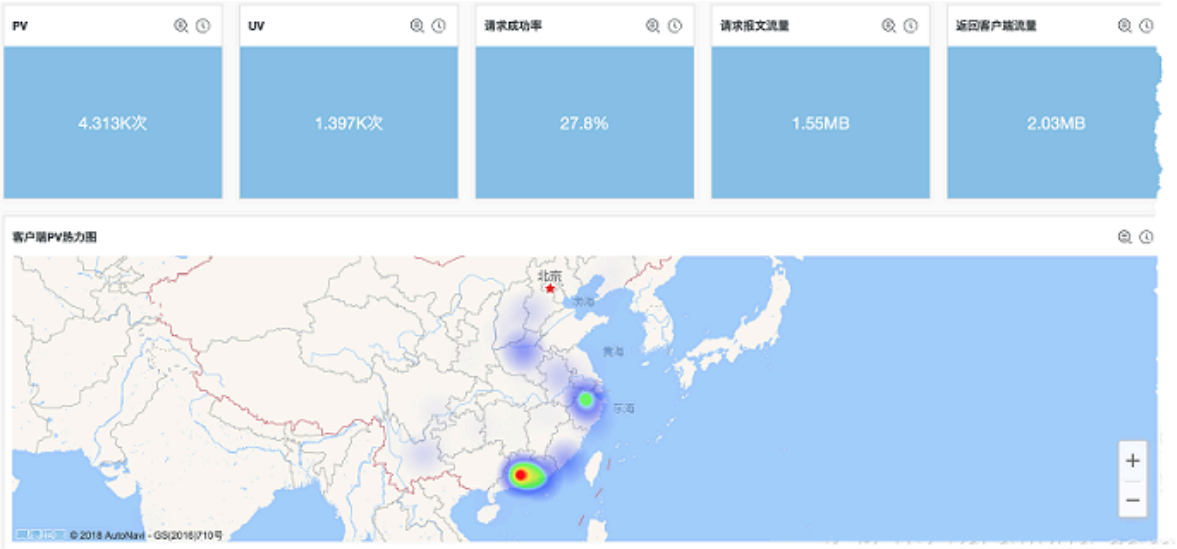
典型的概览指标包括：

- PV：client（请求源IP）发起的HTTP(S)请求次数。

- UV：对于相同client IP只计算一次，合计的总体请求次数。
- 请求成功率：状态码为2XX的请求次数占总体PV的比例。
- 请求报文流量：客户端请求报文长度（request_length字段）的总和。
- 返回客户端流量：SLB返回给客户端的HTTP body的字节数（body_bytes_sent字段）总和。
- 请求的热点分布：计算client IP的地理位置，按照每个地理位置来统计每个区域的PV情况。

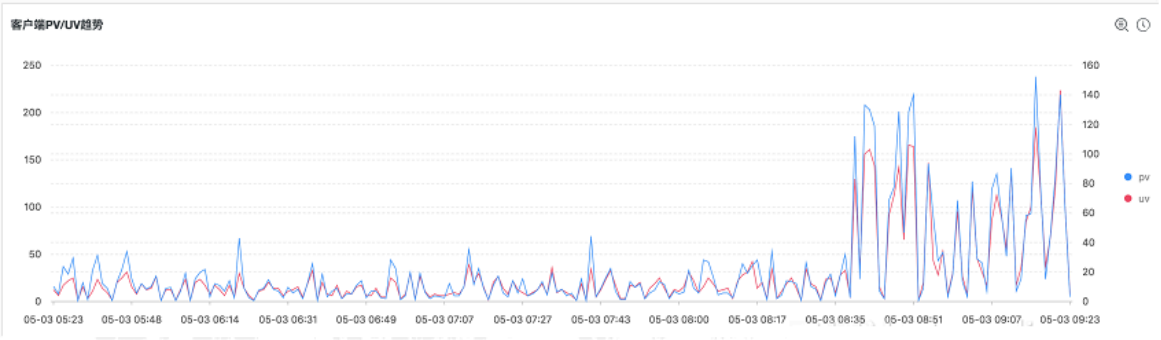
- 查看用户请求的来源地区。

  如下图所示，通过地图可以看到用户请求主要来自珠三角和长三角区域。



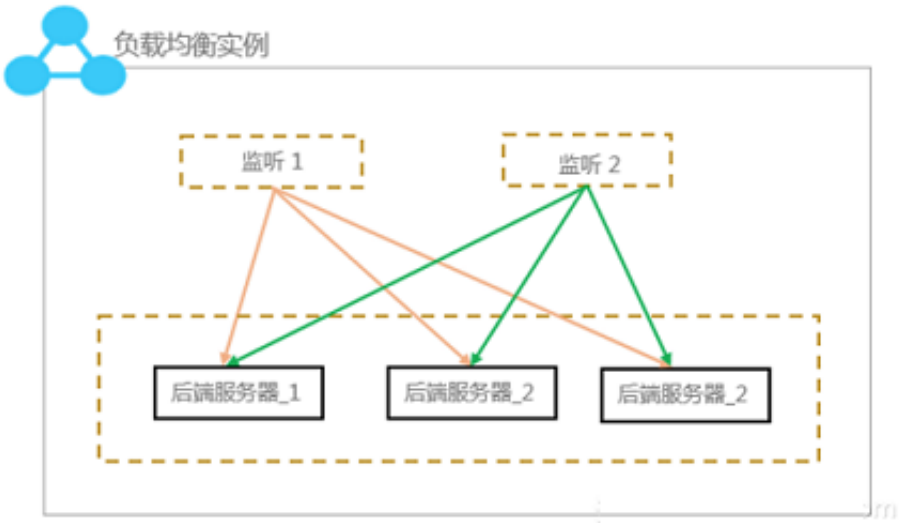
- 在日志服务的Dashboard中，通过添加过滤条件可以在当前图表中筛选符合条件的数据指标来展示，例如：client IP维度、SLB实例ID维度。

  例如，查询一个指定SLB实例ID的PV、UV随时间的变化趋势。



请求调度分析

从客户端过来的流量会先被SLB处理，并分发到多台RealServer的一台上做实际的业务逻辑处理。SLB可以检测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将流量重新加上去，这个过程是自动完成的。



对SLB实例添加一个监听，监听可以设置三种调度方法：轮询、加权轮询（WRR）和加权最小连接数（WLC）。

- 轮询：按照访问次数依次将外部请求依序分发到后端ECS上。
- 加权轮询：您可以对每台后端服务器设置权重值，权重值越高的服务器，被轮询到的次数（概率）也越高。
- 加权最小连接数：除了根据每台后端服务器设定的权重值来进行轮询，同时还考虑后端服务器的实际负载（即连接数）。当权重值相同时，当前连接数越小的后端服务器被轮询到的次数（概率）也越高。

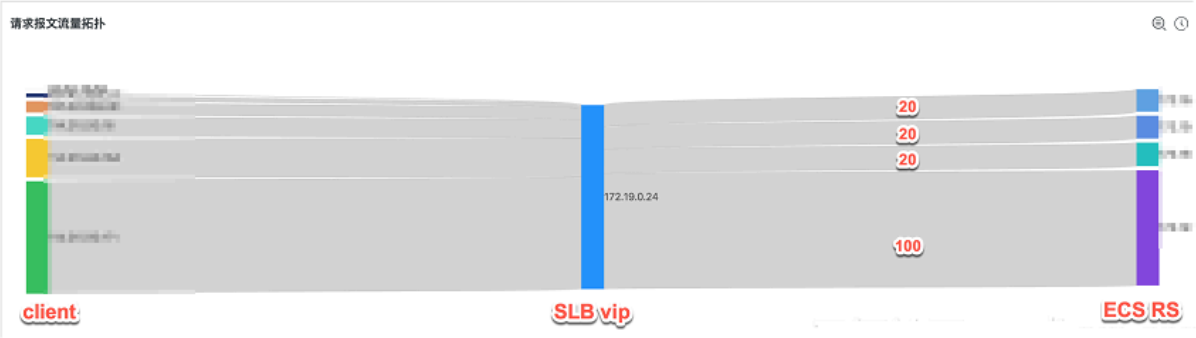
例如，`172.19.39.**` 机器同时兼有跳板机职能，其性能是其它三台机器的4倍，这里为它设置权重100，其余设置权重为20。

基于实例的访问日志，通过如下一条查询语句可以完成和两个维度的流量聚合：

```
* | select COALESCE(client_ip, vip_addr, upstream_addr) as source,
COALESCE(upstream_addr, vip_addr, client_ip) as dest, sum(request_le
```

```
ngth) as inflow group by grouping sets( (client_ip, vip_addr), (
vip_addr, upstream_addr))
```

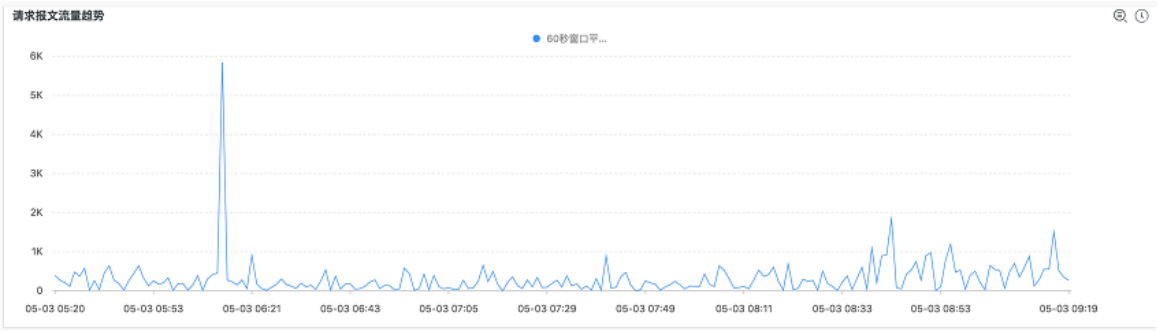
结合桑基图对SQL查询结果做vip维度的聚合可视化，最终得到请求报文流量拓扑图。多个client IP向SLB vip（`172.19.0.**`）发起请求，请求报文流量基本遵循20:20:20:100比例转发到后端的RealServer进行处理。桑基图清晰地表述了每台RealServer的负载情况。

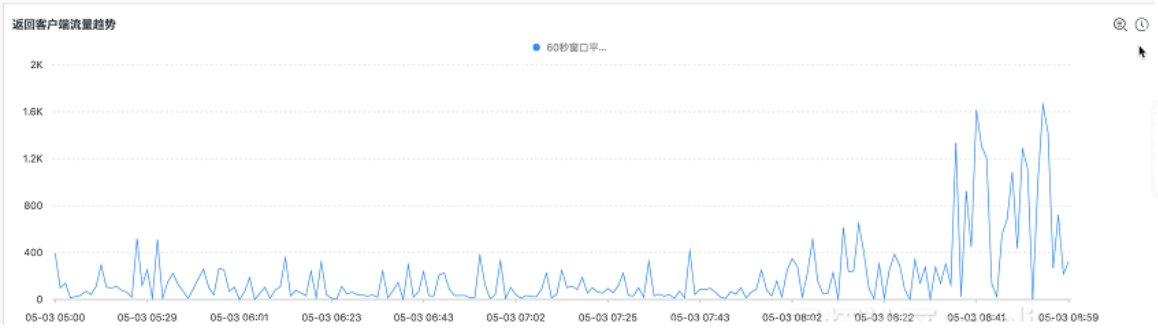


流量与Latency分析

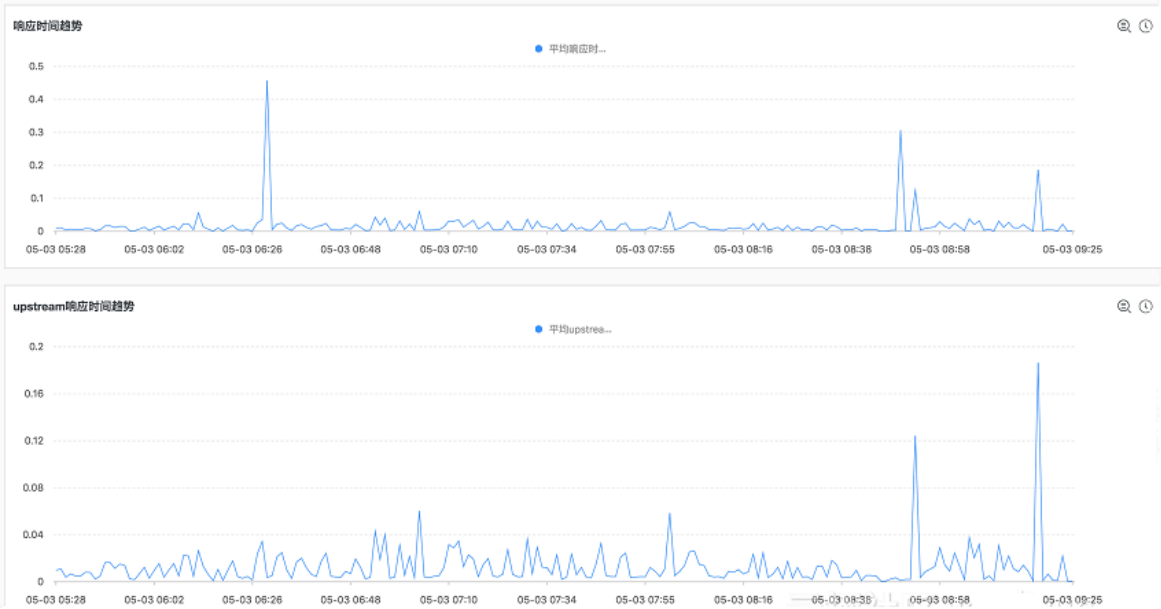
按照1分钟时间维度对流量与latency指标做聚合计算：

- request_length、body_bytes_sent统计





- request_time、upstream_response_time统计



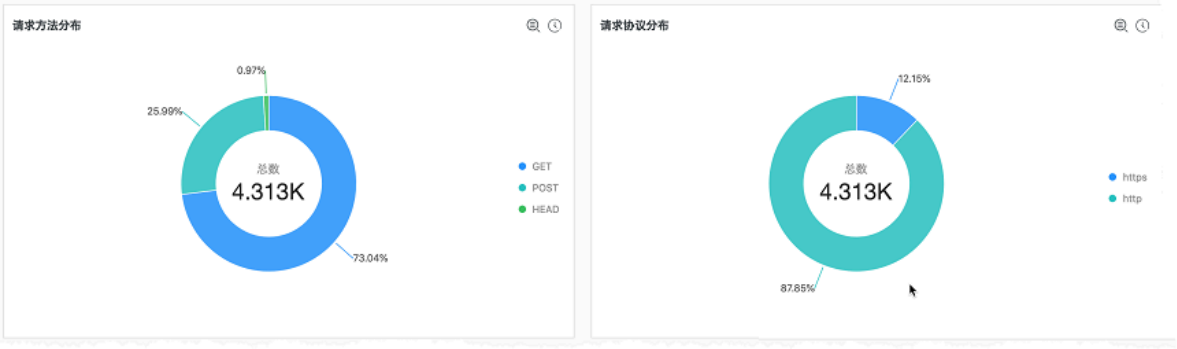
- 高延迟RealServer统计

用户请求概览

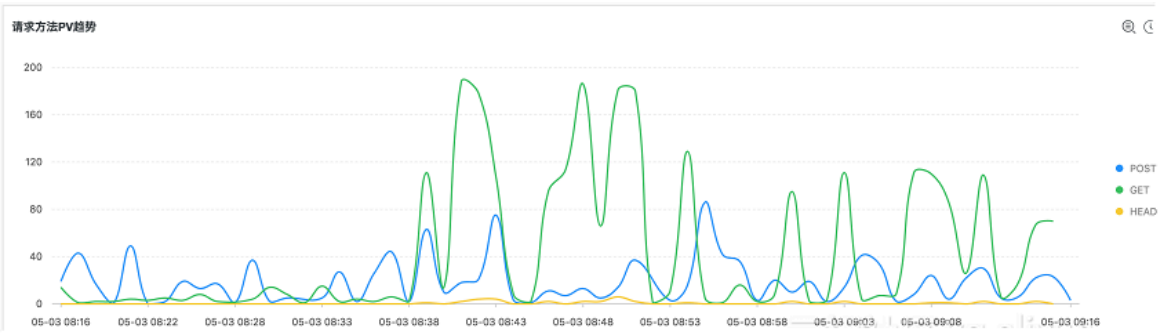
根据请求的方法、协议、状态码等维度分析访问日志中HTTP(S)请求。

指定时间段、状态码快速定位RealServer的需求，通过日志服务的查询分析功能可以迅速得到结果：

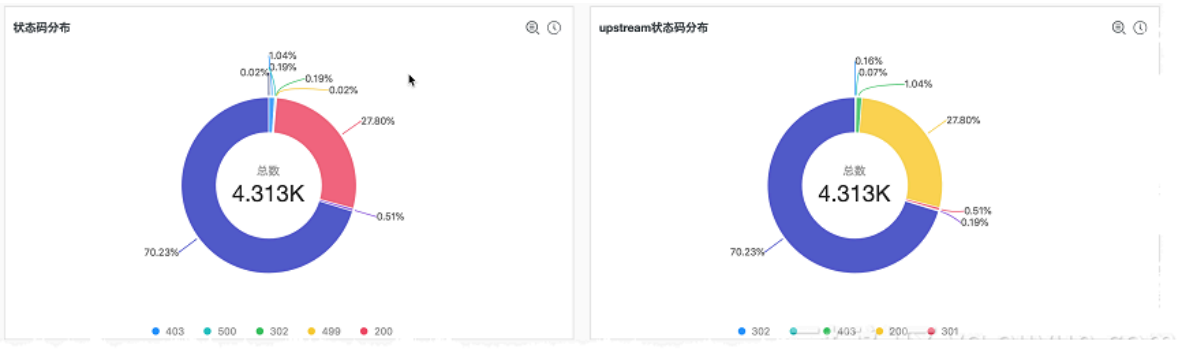
1. 在一个时间段内，请求方法维度上可以做PV分布统计。



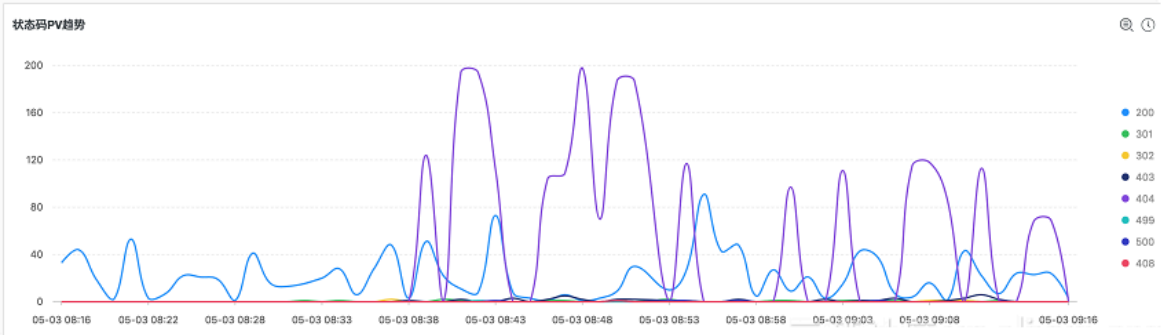
2. 加上时间属性，同时在时间、请求方法两个维度上可以统计出各方法的PV趋势。



3. 请求响应状态码分布可以展示服务的基本状况，如果大量的500状态码则意味着后端RealServer的应用程序在发生内部错误。



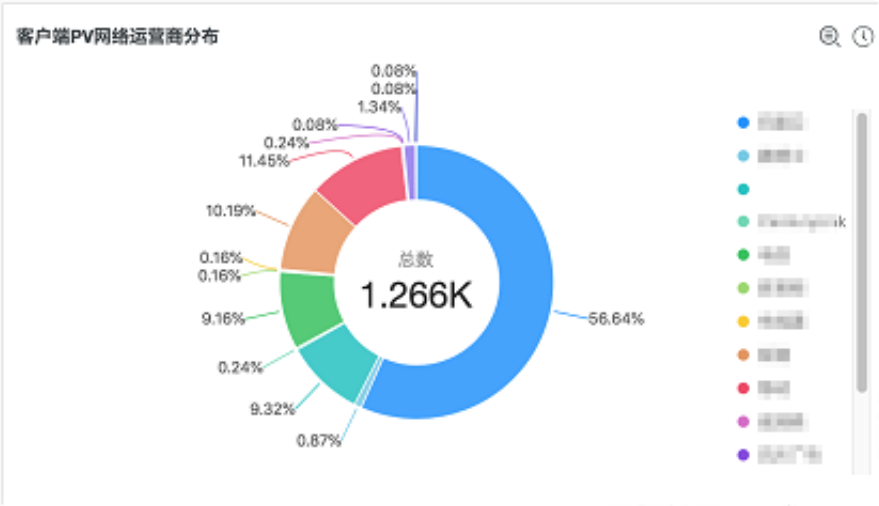
4. 围绕着每一个状态码，可以查看其随时间变化趋势。



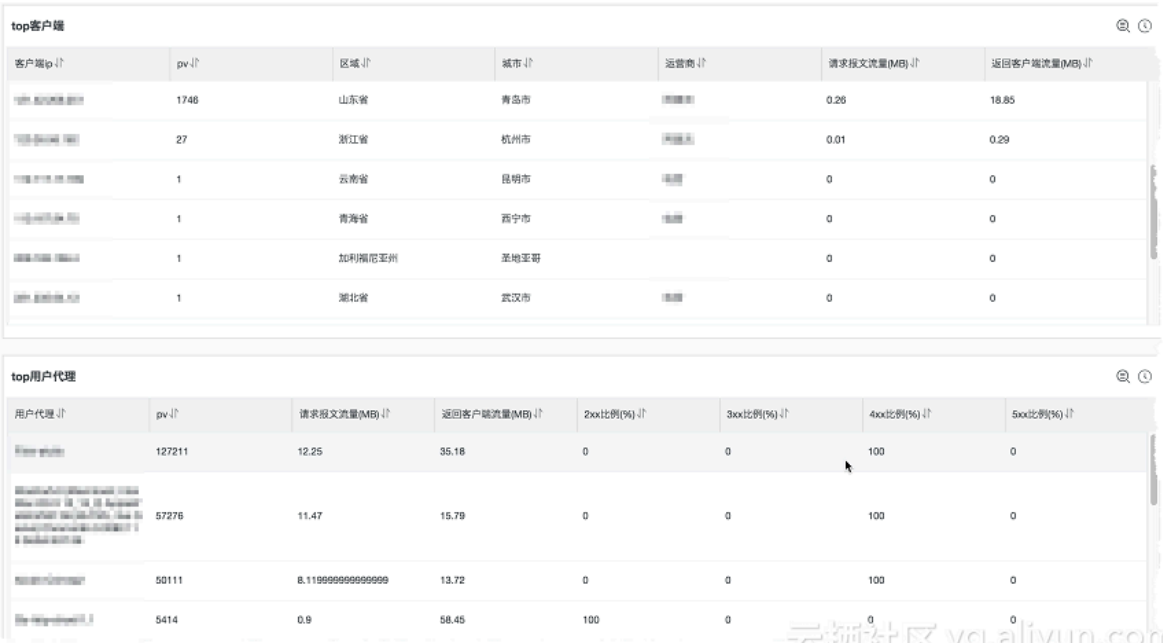
请求源分析

对client的IP做计算可以得到每条请求的发起地理位置（国家、省份、城市）、电信运营商信息。

1. 对用户请求IP的运营商做PV分布图。



2. 按照请求PV降序，对client IP做统计可以展示大用户请求的具体来源。

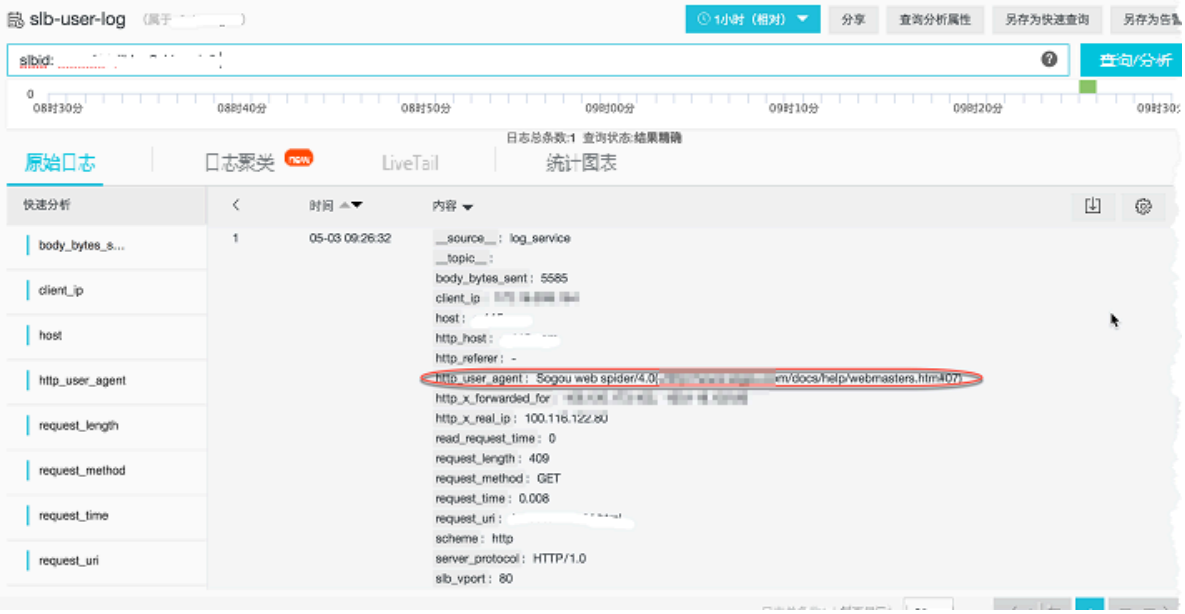


3. 查看http_user_agent。

用户代理（http_user_agent）也是常常需要关注的对象，可以据此区分出谁在访问我们的网站或服务。比如搜索引擎会使用爬虫机器人扫描或下载网站资源，一般情况下的低频爬虫访问可以让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，则可能对服务的性能和机器资源造成浪费，需要实时监控高PV请求状况并采取手段来控制影响。

访问日志中根据查询SLB ID或应用host、http_user_agent关键词，可以很快检索出相关记录。下图是一条搜狗爬虫程序的GET请求日志，请求很稀疏，对于应用无影响。

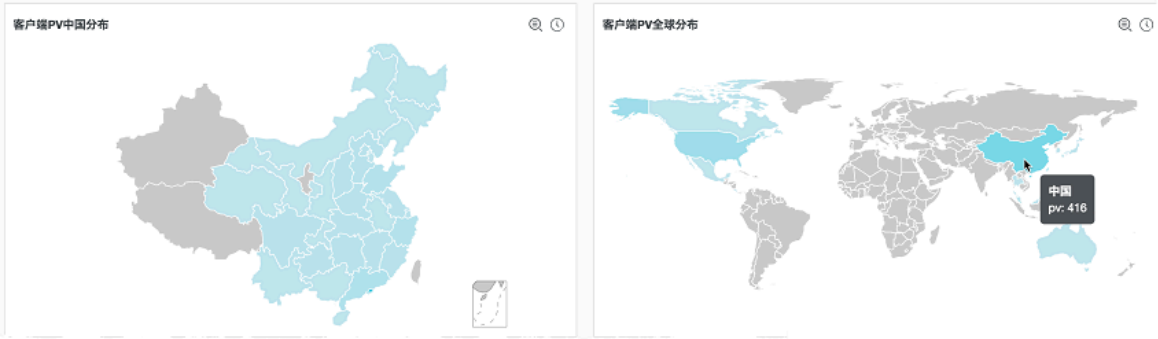


运营概览

SLB访问日志在运营同学手里同样发挥着重要作用，可以基于日志分析出来流量模式，进而辅助业务决策。

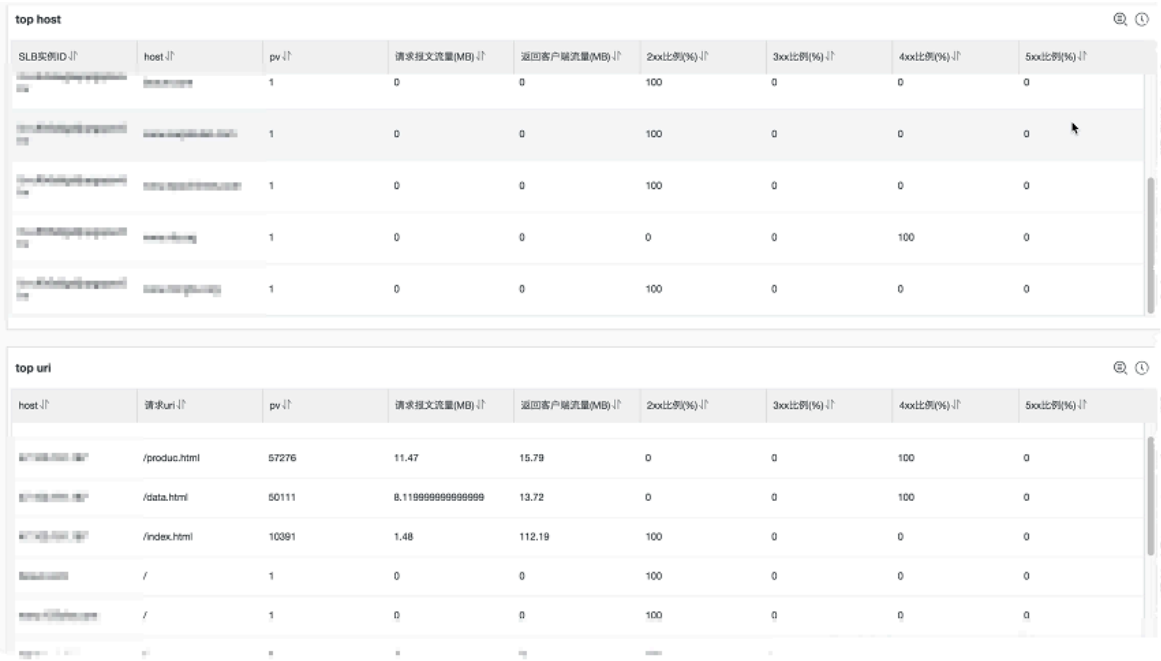
1. 查看PV的热点分布。

   在地理维度上，通过PV的热点分布，可以清晰了解到我们服务的重点客户在哪里，PV低的区域可能需要再推广加强。



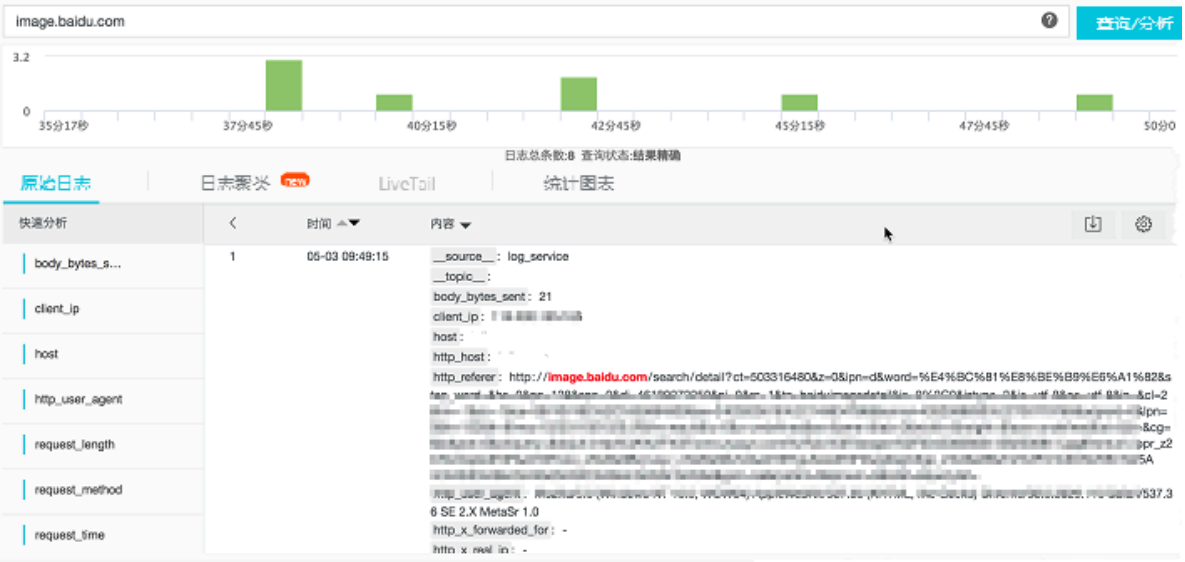
2. 查看host/URI。

对于一个网站而言，通过分析访客的行为可以为网站内容建设提供有力的参考。哪些内容好，哪些内容不好？这个问题可以用头部、尾部PV的host/URI来回答。



3. 查看request_uri。

对于热门资源（访问日志的request_uri字段），可以关注一下日志详情的http_referer字段，查看网站的请求来源。好的导流入口需要持续加强，对于盗链行为则需要想办法克制。下图是在日志查询页面中找到一条来自百度图片的跳转请求。



14 FAQ

14.1 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:**
> 需要提前开启日志索引功能并给__tag__:__hostname__字段开启统计功能。

```
* | select "__tag__:__hostname__" , count(1) as count group by "
__tag__:__hostname__"
```

如何在日志数据中搜索IP地址?

在日志数据中搜索IP地址，支持全部匹配的方式检索。您可以直接在日志数据中直接搜索指定IP地址相关的日志信息，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。如果需要的话，建议自行过滤，比如用SDK先下载数据，然后在代码里用正则或者用string.indexof等方法判断。

例如，在日志服务的Project中搜索条件如下。

```
not ip:121.42.0 not status:200 not 360jk not DNSPod-Monitor not status
:302 not jiankongbao
```

```
not 301 and status:403
```

搜索结果中仍会出现121.42.0网段地址。因为日志服务会认为121.42.0.x是一个词，所以只有搜121.42.0.x能搜到结果，而121.42.0的话不会搜到这个结果，同理加上not也就不会过滤该地址。

如何在日志中搜索包含空格的关键字?

搜索包含空格的关键字时，如果直接搜索，则会得到包含空格左侧关键字或右侧关键字的所有日志。建议您在查询的包含空格的关键字时，把关键字用双引号包裹起来，将引号中的内容作为一个关键字进行搜索，搜索结果就是符合条件的日志内容。

例如，在以下日志中搜索包含关键字`POS version`的日志。

```
post():351]:&amp;nbsp;device_id:&amp;nbsp;BTAddr&amp;nbsp;:&amp;nbsp;B6:xF:xx:65:xx:A1
&amp;nbsp;IMEI&amp;nbsp;:&amp;nbsp;35847xx22xx81x9&amp;nbsp;WifiAddr&amp;nbsp;:&amp;nbsp;4c:xx
:0e:xx:4e:xx&amp;nbsp;|&amp;nbsp;user_id:&amp;nbsp;bb07263xxd2axx43xx9exxea26e39e
5f&amp;nbsp;POS&amp;nbsp;version:903
```

如果直接搜索`POS version`，则会得到包含`POS`或者`version`的所有日志，不符合搜索要求。如果搜索`"POS version"`，则会得到包含关键字`POS version`的所有日志。

如何完成双重条件检索?

双重条件检索时，只需同时输入两个语句即可。

例如，需要在Logstore中搜索数据状态不是OK或者Unknown的日志。直接搜索`not OK not Unknown`即可得到符合条件的日志。

日志服务提供哪些渠道查询采集的日志?

日志服务提供了三种方式查询日志：

1. 通过日志服务控制台查询。
2. 通过SDK查询。
3. 通过Restful API查询。

14.2 查询不到日志数据

在使用日志服务产品的日志查询功能时，如果查询不到日志数据，请按照以下原因进行排查。

1. 未成功采集日志数据

如果并未成功采集日志数据到日志服务，则无法查询到目标日志。请在预览界面查看是否有日志数据。

如果有日志数据，说明日志数据已成功采集到日志服务中，建议您排查其他原因。

如果没有日志数据，可能是以下原因造成，请进一步排查。

- 日志源没有生产日志数据。

  日志源没有日志产生的情况下，没有日志可以投递到日志服务。请检查您的日志源。
- Logtail无心跳。

  请在机器组状态页面中查看机器是否有心跳。没有心跳请参考#unique_251。
- 监控文件没有实时写入。

  如果监控文件有实时写入，您可以打开*/usr/local/ilogtail/ilogtail.LOG*查看报错信息。常见错误如下：

  - parse delimiter log fail：分割符收集日志出错。
  - parse regex log fail：正则收集日志出错。

2. 分词设置错误

查看已设置的分词符，检验根据分词符对日志内容进行分割后，是否刚好得到关键字。例如分割符为默认的`,;=()[]{}?@&<>/:'`那么用户的日志里如果是有abc”defg,hij会被分割成abc”defg和hij两部分，用abc就搜不到这条日志。

同时支持模糊查询，具体查询语法，请参考#unique_10。

说明:

- 为了节约您的索引费用，日志服务进行了索引优化，配置了字段索引的Key，不进全文索引。例如，日志中有名为message的key，并且配置了字段索引，加了空格做分词（加空格做分词，请把空格加到分词字符串的中间）。”message: this is a test message“可以用 key: value 的格式 message:this 查到，但是直接查this查询不到，因为配置了字段索引的key，不进全文索引。
- 创建索引或者对索引做任何更改，只对新进的数据有效，旧数据一律无效。

  您可以点开索引属性，检查已设置的分词是否符合要求。

3. 其他原因

如果日志有产生，可以先在查询处修改查询的时间范围。另外由于日志预览的功能数据是实时的，但是查询的功能是有最多1分钟的延迟的，所以用户可以在日志产生后等1分钟再查。

如您的问题仍未解决，请提工单联系我们。

14.3 日志消费与查询区别

日志服务提供日志消费和查询的功能，都属于对日志的读操作，区别在于消费提供收集和分发通道，查询提供日志查询功能。

日志消费与日志查询区别

日志服务提供了两项功能都和“读”有关：

日志收集与消费（LogHub)：提供公共的日志收集、分发通道。全量数据顺序（FIFO）读写，提供类似Kafka的功能

- 每个LogStore有一个或多个Shard，数据写入时，随机落到某一个shard中
- 可以从指定shard中，按照日志写入shard的顺序批量读取日志
- 可以根据server端接收日志的时间，设置批量拉取shard日志的起始位置（cursor）

日志查询（Search/Analytics）：在LogHub基础上提供海量日志查询+分析功能，根据条件进行日志查询与统计

- 通过查询条件查找符合要求的数据
- 支持关键词 AND、NOT、OR的布尔组合和结果SQL统计
- 数据查询不区分shard

两者区别：

功能	日志查询(LogSearch)	日志收集与消费(LogHub)
关键词查找	支持	不支持
小量数据读取	快	快
全量数据读取	慢(100条日志100ms，不建议通过该方式读取数据)	快 （1MB日志10ms，推荐方式）
读取是否区分topic	区分	不区分，只以shard作为标识
读取是否区分shard	不区分，查询所有shard	区分，单次读取需要指定shard
费用	较高	低
适用场景	监控、问题调查与分析等场景	流式计算、批量处理等全量处理场景

14.4 日志查询分析常见报错

日志查询分析的常见报错如下。

1. line 1:44: Column ‘my_key_field’ cannot be resolved;please add the column in the index attribute

报错原因：`my_key_field`这个Key不存在，所以您在query中无法引用该Key。

解决方案：在查询页面，右上角查询分析属性里，添加该字段为字段索引，同时打开统计功能。

2. Column ‘xxxxline‘ not in GROUP BY clause;please add the column in the index attribute

报错原因：您在查询中使用了GROUP BY语法，但是在Select中引用了一个非agg字段，该字段没有出现在GROUP BY中。例如`select key1, avg(latency) group by key2`，key1没有出现在GROUP BY中。

解决方案：正确语法是`select key1,avg(latency) group by key1,key2`。

3. sql query must follow search query,please read syntex doc

报错原因：没有指定filter条件，例如`select ip,count(*) group by ip`。

解决方案：正确的写法为`*|select ip,count(*) group by ip`。

4. please read syntex document,and make sure all related fields are indexed. error after select .error detail:line 1:10: identifiers must not start with a digit; surround the identifier with double quotes

报错原因：SQL中引用到的列名、变量名等以数字开头，不符合规范。

解决方案：建议更改该名称，以字母开头。

5. please read syntex document,and make sure all related fields are indexed. error after select .error detail:line 1:9: extraneous input ‘’ expecting

报错原因：有单词拼写错误。

解决方案：请根据报错中指出的错误位置，修改至正确。

6. key (category) is not config as key value config,if symbol : is in your log,please wrap : with quotation mark "

报错原因：category字段未配置字段索引，不能在分析语句中使用。

解决方案：请在查询分析属性中设置该字段的索引。详细说明请参考#unique_4。

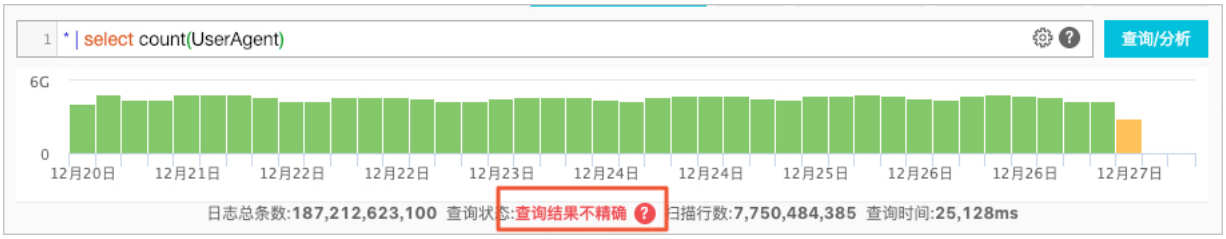
7. Query exceeded max memory size of 3GB

报错原因：当前query使用服务端内存超过3 GB。通常原因为GROUP BY查询的去重后value太多。

解决方案：请优化GROUP BY的查询语句，减少GROUP BY的Key的个数。

14.5 查询不精确有哪些原因

查询分析日志时，控制台可能会提示查询不精确，表示日志服务在查询分析时未能扫描全部日志数据，返回的查询分析结果不是基于全部日志数据的精确结果。



查询不精确一般由以下原因造成：

1. 查询时间范围太大

错误原因：时间范围较大时，例如3个月或1年，一次查询无法完整扫描这个时间段的所有日志数据。

解决方案：请缩小查询的时间范围，分多次查询。

2. 查询条件过于复杂

错误原因：查询条件过于复杂、或者查询条件中包含了一些高频词汇时，日志服务不能一次性读取结果。

解决方案：请缩小查询的范围，分多次查询。

3. SQL计算要读取的数据量太多

错误原因：SQL计算要读取的数据量太多时，容易造成查询不精确。比如读取多个字符串列时，每个Shard限制读取1个G数据，超过后会返回不精确结果。

解决方案：请缩小查询的范围，分多次查询。