百度智能云

TSDB 文档

目录

功能发布记录

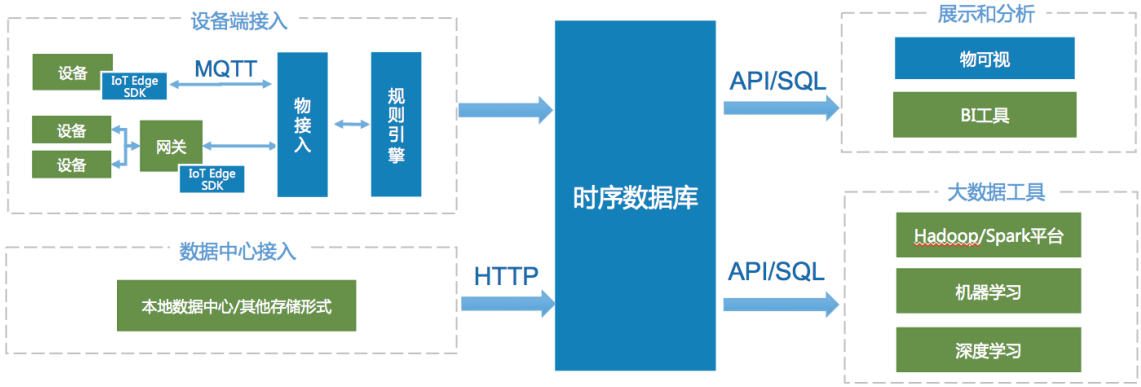
发布时间	功能概述
2022-10	支持自动续费
2022-10	原日志服务接入云审计
2021-11	Node SDK支持IP访问：Node SDK支持通过IP方式访问TSDB服务
2021-10	Python SDK支持IP访问：Python SDK支持通过IP方式访问TSDB服务
2020-09	支持mySQL协议：TSDB支持MySQL协议，用户可以控制台创建mySQL协议账号，采用mySQL shell、mySQL JDBC以及支持mySQL协议的各种BI工具访问TSDB数据，用户可更加便捷的使用TSDB SQL查询。
2020-03	计费模式更新：提供数据免费存储1年的计费模式，存储超过1年的数据收取续存费用
2019-12	对接SugarBI：TSDB已接入百度智能云SugarBI，用户可以通过SugarBI访问TSDB，对TSDB中存储的数据进行多种交互式数据分析。
2019-10	支持时空服务：TSDB目前的SQL查询接口已支持多种与空间地理位置相关的函数，包括二维计算和球面计算。借助这类函数的强大能力，用户可以使用SQL方便地对空间地理位置相关的数据进行计算、分析，挖掘数据价值。
2019-05	支持SQL查询：TSDB支持标准SQL的查询接口，可以通过SQL语言实现对TSDB数据的筛选和查询，也可以充分利用SQL函数的计算能力，挖掘数据价值。
2018-08	新增Python SDK文档，支持写入、查询数据以及管理数据库，支持SQL接口
2018-09	数据导出：支持将所有数据导出为一个文件或将每个度量的数据导出一个文件

产品描述

产品概述

百度智能云时序时空数据库（Time Series Database，简称TSDB）是一种存储和管理时间序列数据的专业化数据库，为时间序列的存储提供高性能读写、低成本存储、强计算能力和多生态支持的多种能力。

在物联网场景下，TSDB有广泛的应用。 如工业生产环境下，每个厂区有大量的监测点，如果以10秒的频率发送数据，50万监测点每年会产生1.58万亿左右数据点。TSDB不仅可以轻松存储海量数据点，还可以对这些数据进行快速查询并做可视化展示，帮助企业管理者分析数据。典型的物联网场景下，数据从端侧到云端的数据流如下所示。



> 注意:
>
> 推荐用户使用chrome, firefox, edge, safari这四类浏览器执行控制台操作，目前暂不支持IE浏览器。

产品功能

数据读写

- 数据写入

  支持Restful API方式高并发写入数据；支持物联网设备通过规则引擎写入数据，可以支撑每秒千万级数据点写入，并可线性扩展。

  - 公有云服务目前支持导入未来5天内的数据
  - 支持写入的数据类型包括Long、Double、Bytes、String、BigDecimal，BigDecimal类型需要点击申请试用

- 数据查询

  支持通过Restful API和控制台来查询数据，可以对数据进行标签过滤、值过滤、分组等查询，并可以支持控制台可视化展示。

时间序列数据管理

- 数据管理

  支持时间序列数据的写入、查询和删除

- 数据时效

  可以开启数据时效，系统可以自动清除不在有效期内的数据

- 数据导入导出

  提供数据导入导出接口

数据计算

- 插值查询

  提供插值查询能力，将未上传的数据补齐，并支持多种插值算法

- 聚合计算

  提供AVG、SUM、MAX等15种聚合函数，可以将数据降精度聚合，并支持嵌套聚合

- 预处理

  提供对数据的预处理能力，可将相关数据提前过滤和聚合，实现快速返回查询结果

压缩存储

采用高效压缩算法，可达10-20倍压缩率，大大降低存储成本。

数据库管理

实时监控，提供对数据库的写入、读取状态进行实时监控

示范数据

提供示范数据，帮助用户体验时序数据库的各项功能

产品优势

- 高性能读写

  每秒千万级数据点写入，亿级数据点聚合结果秒级返回

- 低成本存储

  高效压缩算法，大大节省存储空间

- 强计算能力

  提供插值、预处理等多种计算方式；支持15种聚合函数

- 多生态支持

  支持SQL查询、主流Hadoop/spark等大数据分析平台、多种可视化工具

- 高可靠服务

  三副本、分布式部署，保证数据可靠性

名词解释

TSDB ：Time Series Database，时序数据库，用于保存时间序列（按时间顺序变化）的海量数据。

度量（metric）：数据指标的类别，如发动机的温度、发动机转速、模拟量等。

域（field）：在指定度量下数据的子类别。即一个metric支持多个field，如metric为wind，该metric可以有两个field：direction和speed。

时间戳（timestamp）：数据产生的时间点。

数值（value）：度量对应的数值，如56°C、1000r/s等（实际中不带单位）。如果有多个field，每个field都有相应的value。不同的field支持不同的数据类型写入。对于同一个field，如果写入了某个数据类型的value之后，相同的field不允许写入其他数据类型。

标签（tag）：一个标签是一个key-value对，用于提供额外的信息，如"设备号=95D8-7913"、“型号=ABC123”、“出厂编号=1234567890”等。

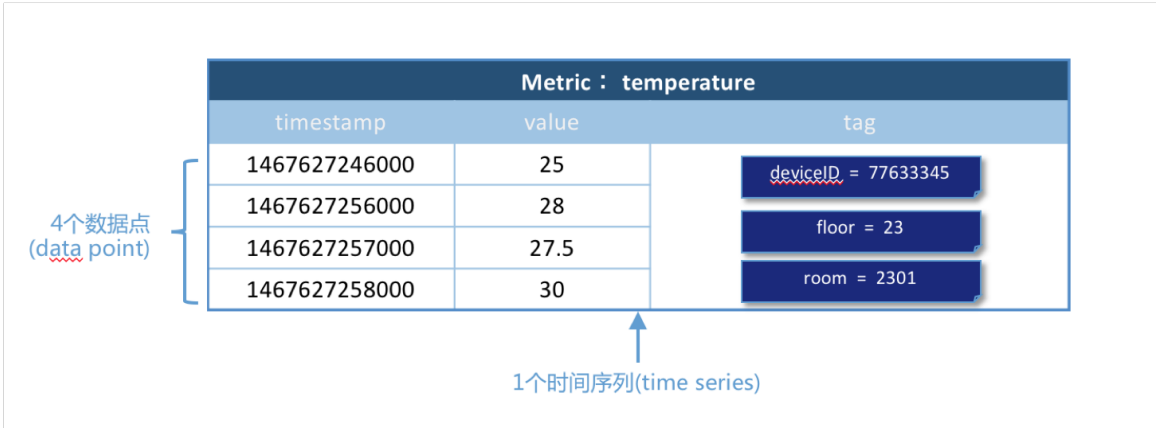
数据点（data point）：“1个metric+1个field（可选）+1个timestamp+1个value + n个tag（n>=1）”唯一定义了一个数据点。当写入的metric、field、timestamp、n个tag都相同时，后写入的value会覆盖先写入的value。

时间序列：“1个metric+1个field（可选） +n个tag（n>=1）”定义了一个时间序列。单域和多域的数据点和时间序列由下图所示：

实践一（单域）

监测温度的值，把温度（temperature）作为一个度量（metric），用标签（tag）来标识每一个数据的额外信息，比如每个数据点都有3个tag，tag是一个key-value对，tag的key分别是deivceID、floor、room。

如图所示，表示对温度的时间序列监测值，共4个数据点。在该图中的4个数据点使用的metric、tag是相同的，所以是同一个时间序列。

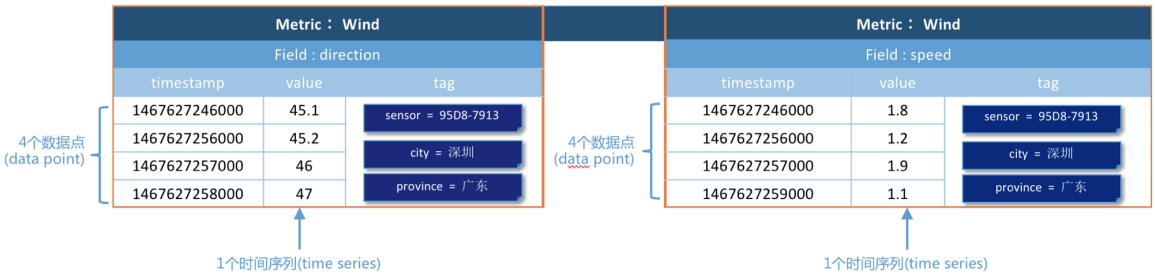


实践二（多域）

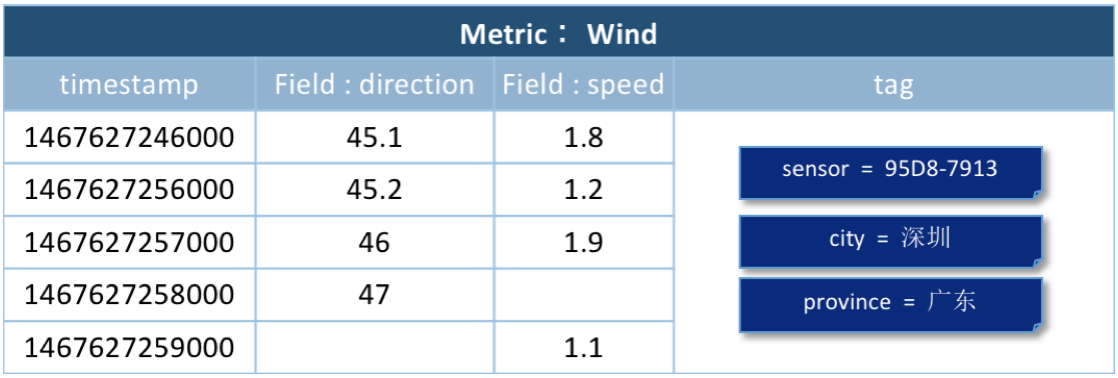
监测风力的值，把风力（wind）作为一个度量（metric），风力（wind）分为两个域：风向（direction）和速度（speed）。这些监测数据是从不同的传感器传输到云端的，用标签（tag）来标识每一个数据的额外信息，比如每个数据点有三个tag，tag是一个key-value对；tag的key分别是sensor、city、province。

为了表示在广东省深圳市传感器编号95D8-7913上传风向（direction）数据，可以将这个数据点的tag为标记为sensor=95D8-7913、city=深圳、province=广东。

如图，展示了metric为wind的两个域(speed和direction)的监测数据。当使用的是metric、field和tag是相同的时，是同一个时间序列。即图中有2个时间序列，8个数据点。



将数据采用metric+field的方式存储的优势在于，可以将同一metric下的数据进行多field联合查询。以上图为例，要查询1467627246000-1467627249000时间内风力(wind)的情况，可以联合查询多个field的值，得到下图的数据。



如果写入时没有数据，在查询时，可以采用插值方案将值补充完整，插值的使用说明见数据库操作相关文档。

- tag的key值和value值都相同才算做同一个tag，即deviceid=1和deviceid=2是两个标签。
- 请不要将时间戳作为tag，否则会导致时间序列超过限制，关于时间序列的限制请参考费率表。

分组（group）：可以按标签（tag）对数据点进行分组。

聚合函数（aggregator）：可以对一段时间的数据点做聚合，如每10分钟的和值、平均值、最大值、最小值等。TSDB目前支持

的聚合函数请参考文档

数据库（database）：一个用户可以有多个数据库，一个数据库可以写入多个“度量”的“数据点”。

数据结构

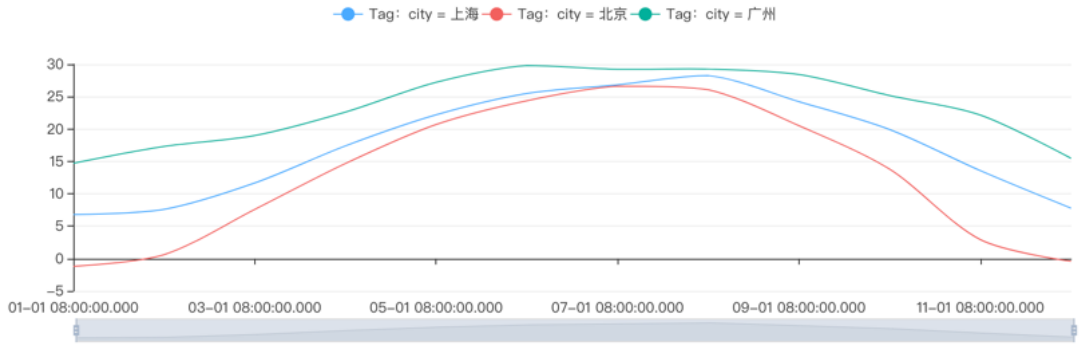
单域：

我们用TSDB提供的示范数据库来举例说明数据结构，请参考示范数据库的操作指南。

示范数据库含有2015年全年北京、上海、广州3个城市多个空气监测站的温度、湿度、风力、PM2.5、紫外线指数等数据（非真实数据），数据来源于每个城市的多个空气监测站。

在TSDB中，温度、湿度、风力、PM2.5、紫外线指数都用metric表示，代表监测维度。不同的城市、不同的空气监测站（用不同的经度和纬度表示）则用Tag的方式表示。拿温度举例，如下图所示。

如果我们希望查看北京、上海、广州在2015年每个月的平均温度，即按照1个月的采样周期进行AVG的聚合，并按照city标签分组。可以得到以下图表:

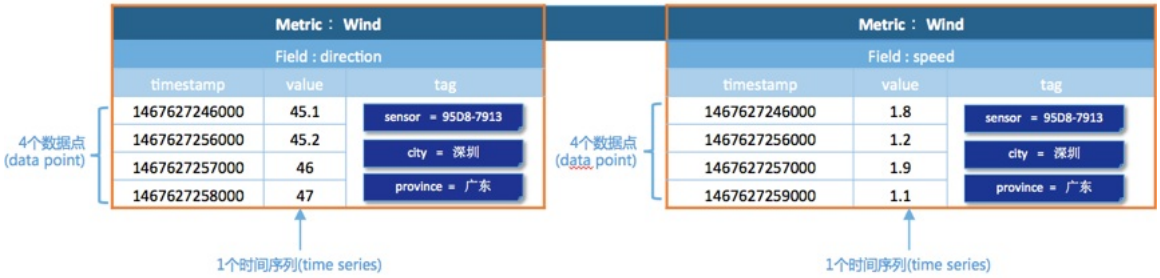


更多操作请参见示范数据库的操作指南。

多域：

多域的结构允许在同一个metric下有多个field，比如风力是一个矢量数据，由风速和风向组成，这样适合将风力和风向作为风（metric）的field存储。

所有查询操作都要先指定metric，如果采用多域的结构，可以在同一个查询中将多个field的值进行过滤，并同时返回多个field。



以上图为例，要查询1467627246000-1467627249000时间内风力(wind)的情况，可以联合查询多个field的值，得到下图的数据。

Metric： Wind			
timestamp	Field : direction	Field : speed	tag
1467627246000	45.1	1.8	sensor = 95D8-7913 city = 深圳 province = 广东
1467627256000	45.2	1.2	
1467627257000	46	1.9	
1467627258000	47		
1467627259000		1.1	

系统限制

资源限制

用户最多可创建100个数据库实例。

- 一个请求最多写入10000个数据点（data point）。
- 一个数据点（data point）最多20个标签（tag）。
- 一个查询中最多8个query。
- 一个查询最多返回100万个数据点（data point）。
- 一个数据库实例时间序列免费额度上限为1000w，最高可支持提高到3000w（详见：预付费-时间序列限制）。
- 长格式类型数据具备默认的最大长度限制，具体如下；最高可提升至默认最大长度的5倍（详见：预付费-长数据长度限制）：
  - String类型的value格式要求： 默认最大长度为256个字符，允许任何字符。
  - BigDecimal类型的value格式要求： 默认最大长度为256个字符。
  - Bytes类型的value格式要求： 默认最大长度为200个字节。
- 每分钟读写限制额度，限制额度根据数据库实例的月写入和月查询额度平均到每分钟计算

命名限制

- 数据库实例名称格式要求： 长度为6-40个字符。 只能包含小写字母和数字。
- 数据库实例描述格式要求： 最大长度为256个字符。 允许为任意字符。
- 度量（metric）格式要求： 长度为1-40个字符。 只能包含大小写字母、数字、下划线，必须以字母开头。
- 域（field）格式要求： 长度为1-40个字符。 只能包含大小写字母、数字、下划线，必须以字母开头。

- 标签键（tag-key）格式要求： 长度为1-100个字符。 只能包含大小写字母、数字、下划线，必须以字母开头。
- 标签值（tag-value）格式要求： 长度为1-200个字符。 允许任何字符。

产品定价

预付费

您可以通过TSDB价格计算器快速获取价格。

TSDB是开箱即用的数据库服务，将数据的写入、存储、查询等能力以API的形式提供，无需用户购买虚机实例，仅需购买每月的写入额度和查询额度即可。

购买前需对每月的写入数据点数和查询能力进行合理预估，可以先购买少量配置，如业务增长，可以通过控制台自动升配。

> 注意：当用户购买时长为10、11或12个月时，由于优惠政策，实际只需要10个月的花费，即可使用12个月的服务。

TSDB计费方式

TSDB计费方式分为数据1年存储免费模式和原数据5年存储免费模式两种模式（5年存储免费模式已不支持新购），具体计费标准请参见：产品价格详情。

时间序列限制

- TSDB实例根据用户购买的月写入额度提供免费的时间序列数量，免费时间序列与月写入额度的对应关系如下表：

每月写入额度（百万点/月）	时间序列限制(个)
1	1,000
2-10	10,000
11-100	100,000
101-1,000	1,000,000
1,001-	10,000,000

- 每个数据库实例免费时间序列上限为10,000,000个；达到上限后如果用户仍然需要更多的时间序列则支持付费增加配额，单价为100元/每百万/月，最高支持增加到30,000,000。

长数据长度限制

- TSDB实例开通长数据存储空间后默认限制String\BigDecimal类型不超过256字符、Byte类型不超过200字节（长数据长度倍数为1）；如果用户需要有更高的长度要求，可通过调整长数据长度倍数来增加（如长数据倍数调整为2，String\BigDecimal类型的长度上限为512字符、Byte类型为400字节），长数据存储的费用等于存储单价*开通空间大小*长数据长度倍数

> 注意：当长数据长度倍数大于1时，bigdecimal类型数据的SQL查询仍会限制数据长度为256字符，普通查询不受影响

- 每个数据库实例长数据长度倍数上限为5

购买指导

场景：

您要监控1000台设备的数据，每个设备有10个监测点，每个监测点上报数据的频率是每分钟一次。其中有一个监测点的数据要以string类型存储，长度为100B。您在控制台进行每天100次查询，每次查询覆盖1000个数据点，业务系统调用查询接口API的查询次数是每分钟查询100次，每次查询覆盖1000个数据点。

- 每个月要上报的数据点数为60*24*30*10*1000 = 432百万点（月写入额度为432百万点/月）
- 时间序列数为10*1000 = 10000个（时间序列免费）
- 查询次数为100*30+100*60*24*30 = 433万次（月查询额度为433万次/月）
- 长数据格式的存储空间为 100B*60*24*30*1000 = 4.32GB（存储空间为4.32GB、长数据长度上限倍数为1倍）

计费的查询接口

接口	方法	API	说明
查询data point	GET	/v1/datapoint?query={json}	查询data point，查询参数在query参数中，包括控制台的调用
查询data point	PUT	/v1/datapoint?query	查询data point，查询参数在body中，包括控制台的调用
SQL查询data point	GET	/v1/row?sql={statement}	使用ANSI SQL查询时序数据，包括控制台调用和API/SDK调用
预处理	控制台调用	控制台调用	预处理规则处理的数据点数

详细接口请参考API文档 系统限制请参考文档

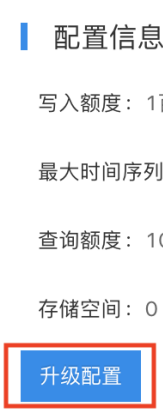
> 注意：
>
> 1、扫描的数据点数（consumedCount）参与计费，consumedCount/1000向上取整就是本次查询的查询单位；
>
> 2、如果有limit参数，会限制每个时间序列的返回点数为limit。系统扫描的数据点数（consumedCount）与limit参数和时间序列数都有关，可以通过减少时间序列数和设置limit参数来降低扫描点数（consumedCount）
>
> 3、系统内部错误不会收取查询费用
>
> 4、超时的查询也会计算查询单位，请合理构建查询语句，以防超时。以下返回时间可作为参考，但不作为SLA保证。
>
> 如果查询请求在100个时间序列内，
>
> a) 扫描10,000个数据点，返回时间在200ms-300ms之间
>
> b) 扫瞄100,000个数据点，返回时间在2000ms-3000ms之间
>
> c) 扫描1,000,000个数据点，返回时间在20s以内

升级配置

用户如出现业务增长的情况，可以按以下操作步骤进行升配，升配收费与预付费计费方式一致：

1、选择“产品服务>时序数据库TSDB>产品名称”，进入数据库详情页面。

2、点击“基本信息>配置信息>升级配置”，进行控制台升配。



到期停服处理

到期停服处理

类型	处理方法
预付费到期	到期后立即停服
停服后处理	停服保留资源30天后删除

快速入门

概述

时序数据库是用于存储和管理时间序列数据的专业化数据库，为时间序列数据提供高性能、稳定可靠的数据库服务。时序数据库特别适用于物联网场景。

在执行具体操作前，用户需了解TSDB相关的核心概念。

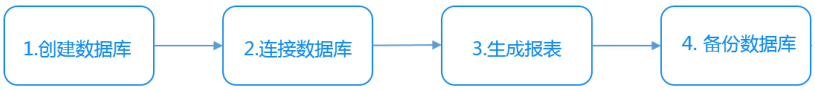
账号注册

> 注意：
>
> 推荐用户使用Firefox或Chrome浏览器执行控制台操作。

在操作专属服务器前，应先完成以下任务：

- 完成百度智能云账号注册。

操作流程介绍



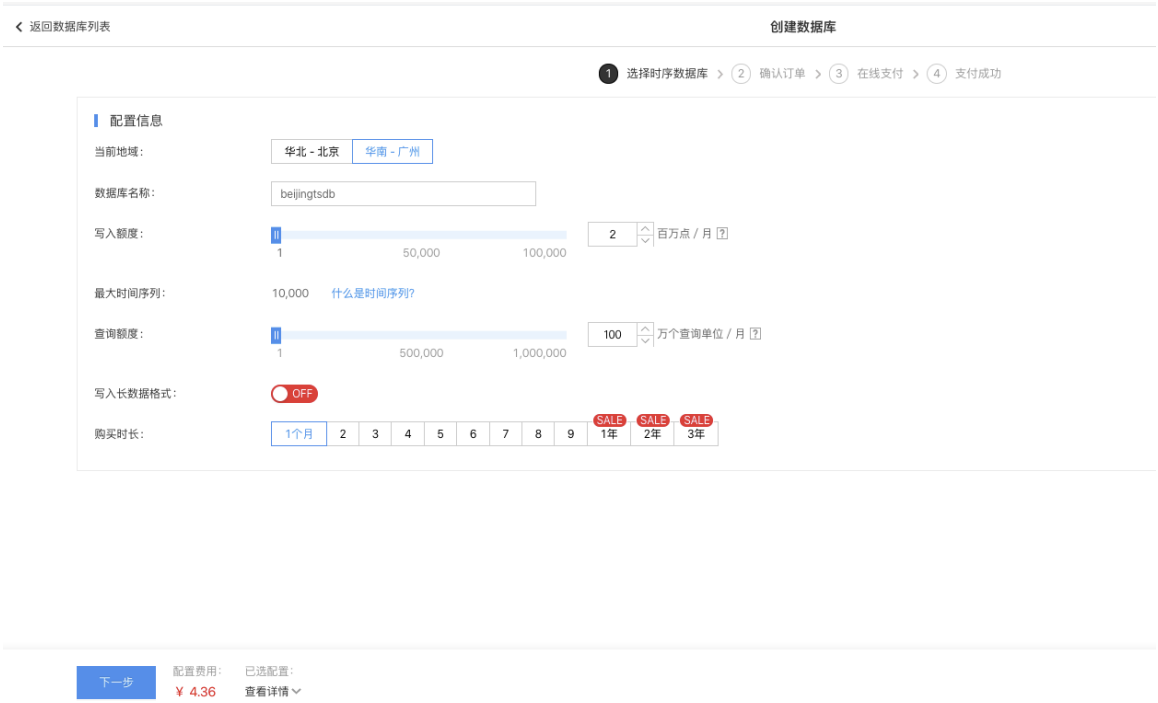
1. 创建数据库：创建TSDB数据库。
2. 连接数据库：配置TSDB从百度智能云的其它产品或第三方服务获取数据，目前仅支持通过restful API写入数据。
3. 生成图表：查看数据库信息，可生成基于时间范围、度量的图表。
4. （可选）备份数据库：通过导出功能可以将当前数据导出至BOS Bucket，用于数据分析或备份用途。

创建数据库

TSDB是用于管理时间序列数据的专业化数据库。区别于传统的关系型数据库，TSDB针对时间序列数据的存储、查询和展现进行了专门的优化，从而获得极高的数据压缩能力、极优的查询性能，特别适用于物联网应用场景。

1. 登录百度智能云官网，点击右上角的“管理控制台”，快速进入控制台界面。
2. 选择“产品服务>时序数据库TSDB”，进入“数据库列表”页面。

3. 点击“创建数据库”，进入创建数据库页面，填写配置信息。

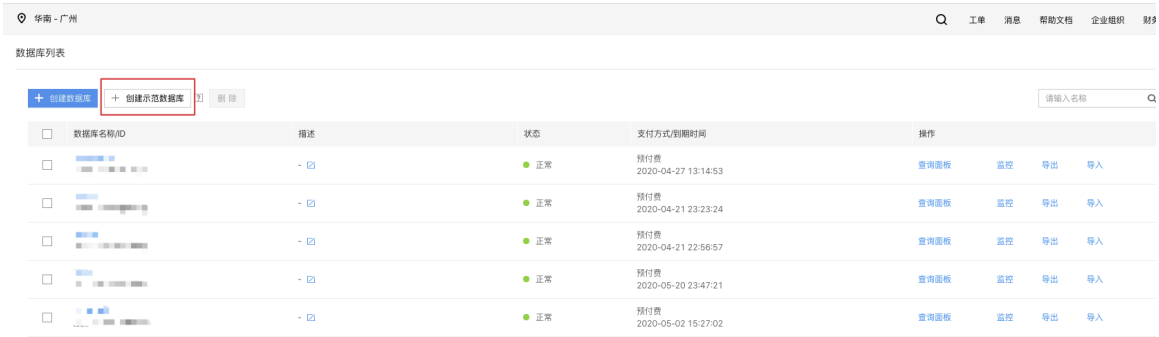


4. 完成配置后点击“确认订单”，完成TSDB数据库的创建。

5. 返回“数据库列表”页面

创建示范数据库

示范数据库用于帮助用户体验时序数据的各项功能。示范数据库的功能及创建方法与普通TSDB数据库相同。创建成功后，系统会自动将气象数据导入示范数据库。



示范数据库含有2015年全年北京、上海、广州3个城市每一个空气监测站的每一天的早晨8点采集的温度、湿度、风力、PM2.5、紫外线指数等气象模拟数据（非真实数据），每一个数据点所属城市城市的名称、邮编，以及该监测站的地理位置（经纬度）等标签，方便您快速体验、理解时序数据库各项查询、聚合及图表展示功能。

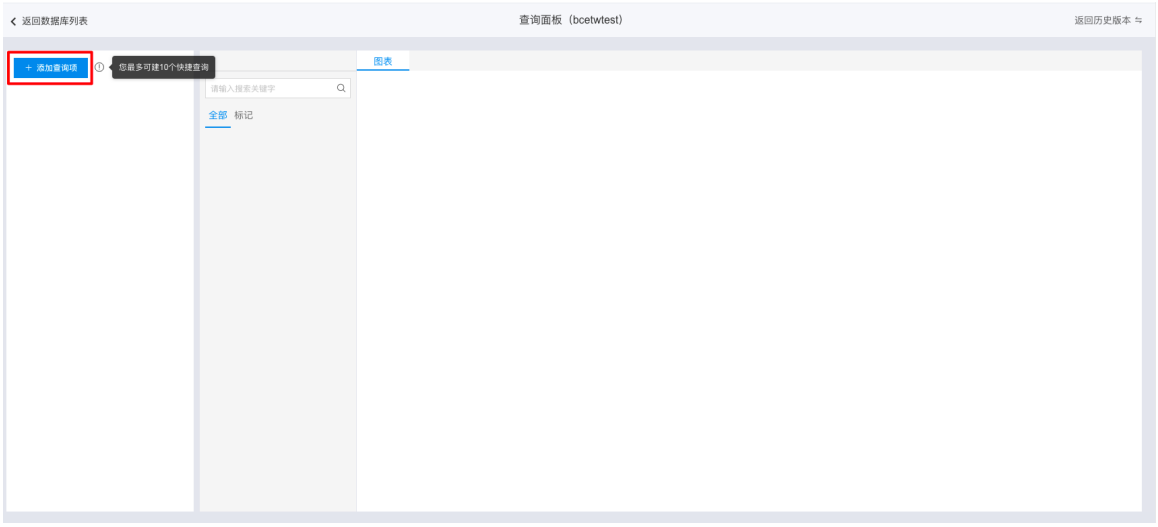
生成示范数据库图表

有关TSDB生成图表功能的具体介绍请参看通过查询面板生成图表，以下仅通过简单示例展示示范数据库生成的图表。

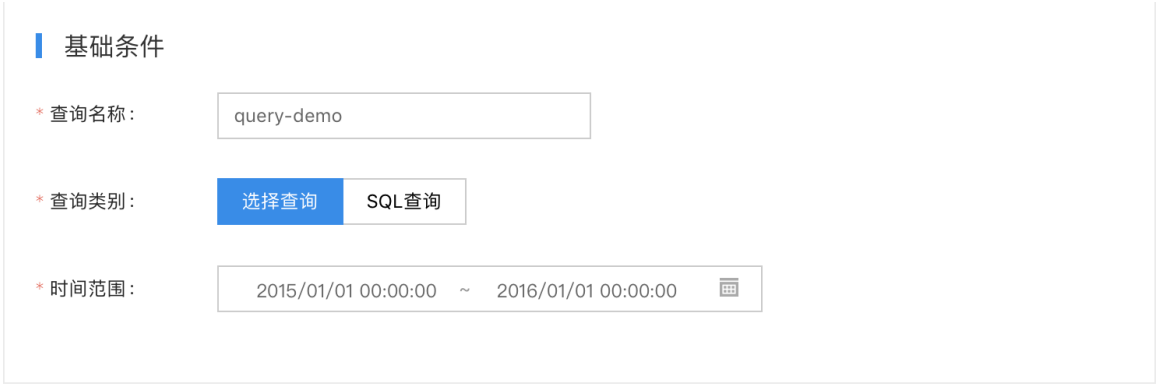
查看北上广2015年温度变化

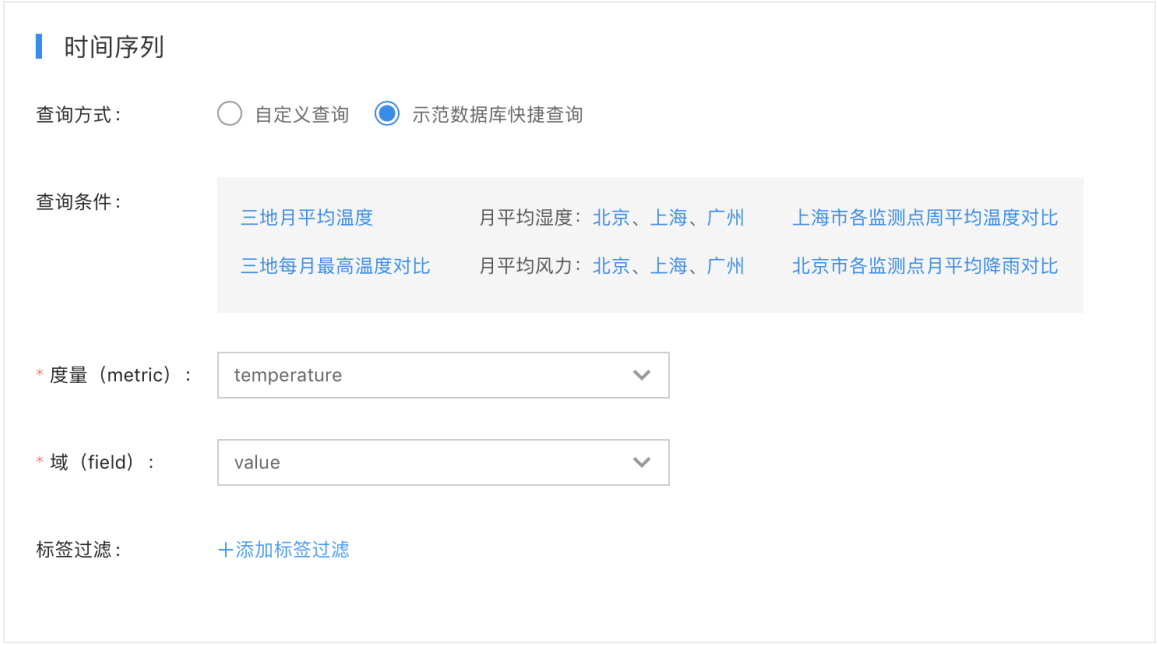
> 注意：
>
> 生成的图表中横轴代表写入数据点的timestamp字段的值。

查询面板添加查询项

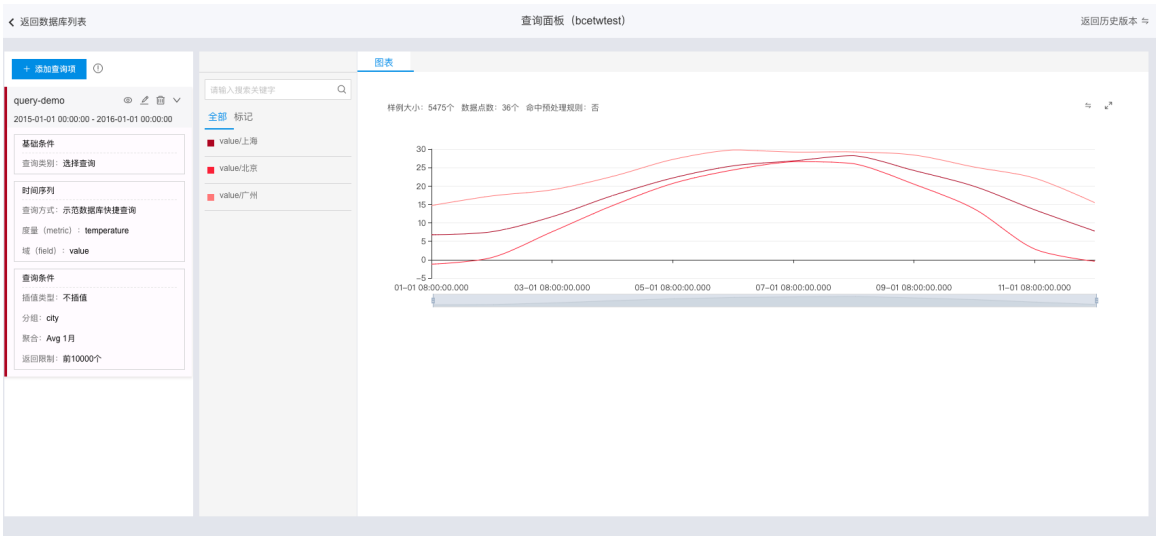


查询项参数设置





生成图表



连接数据库

通过API/SDK写入数据

> 注意:
>
> 不允许写入或导入未来5天后的数据，即新写入数据的时间戳必须小于“当前时间戳+432000秒”。

TSDB支持通过restful API,Java SDK和Node SDK写入数据。

TSDB提供的具体API接口如下：

- 写入数据点
- 获取度量列表
- 获取标签列表
- 查询数据点

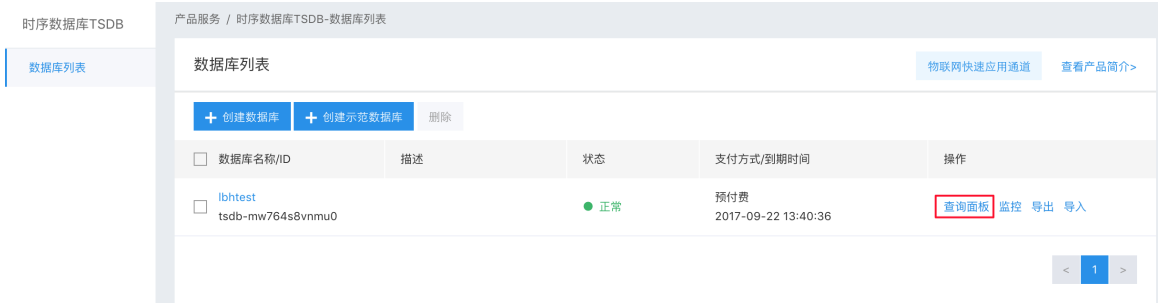
通过控制台导入数据

详情参考：数据导入

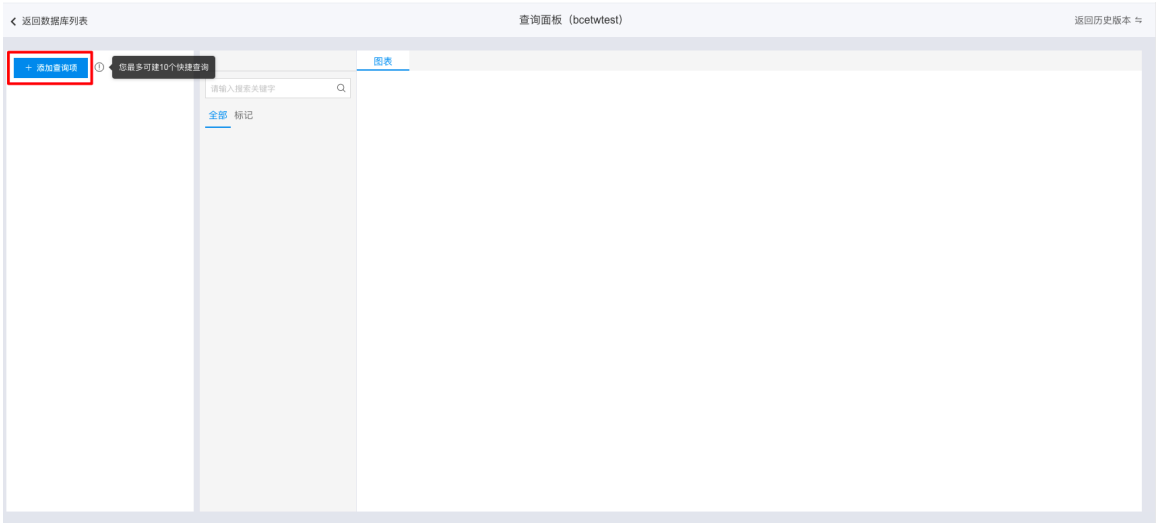
通过查询面板生成图表

通过查询面板，用户可以对当前数据进行筛选并生成图表，具体操作步骤如下：

1. 选择“产品服务>时序数据库TSDB”，进入“数据库列表”页面。

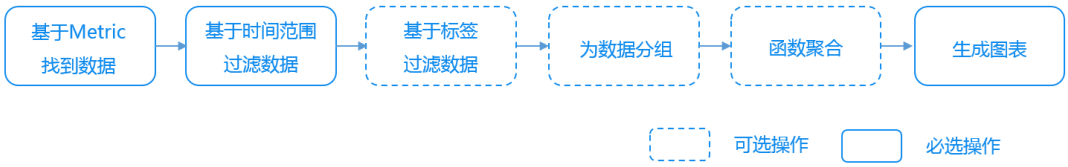


2. 在数据库列表中找到对应的数据库，点击“查询面板”，进入操作界面。



3. 在查询面板点击“添加查询项”，可以配置要查询的参数

生成图表时，系统对数据处理逻辑如下：



- 时间范围设置：支持“绝对时间”和“相对时间”两种设置方式、
- 名称：从下拉列表中选择当前数据库中已有的Metrics名称，用户可点击“添加”，新增多个Metrics进行数据对比。
- 标签过滤：指定标签对数据进行过滤。
- 值过滤：根据指定的数值范围对数据点进行过滤，仅显示出指定的数据点。
- 分组：可以通过“标签”维度进行分组，将携带指定标签的数据分为一组，可输入多个标签。 可将数据分成多组，系统将为每组数据生成图表。

- 聚合函数：通过内置函数对数据进行处理，可添加多个函数，系统将按顺序对每个分组应用函数。当前支持的聚合函数包括：
  - Avg：以每个采样时间范围内的value的平均值作为结果
  - Count：以每个采样时间范围内的点的数目作为结果
  - Dev：以每个采样时间范围内的value的标准差作为结果
  - Diff：以每两个相邻的value的差值作为结果
  - Div：以每个value除以一个除数作为结果
  - First：以每个采样时间范围内的第一个value作为结果
  - Last：以每个采样时间范围内的最后一个value作为结果
  - LeastSquares：对每个采样时间范围内的value进行最小二乘法的拟合
  - Max：以每个采样时间范围内的value的最大值作为结果
  - Min：以每个采样时间范围内的value的最小值作为结果
  - Percentile：每个采样时间范围内的value的第p百分位数作为结果。
  - Sum：以每个采样时间范围内的value的总和作为结果
  - Scale：以每个value乘以一个倍数作为结果
  - Rate：以每两个相邻的value在单位时间（见参数timeUnit）的变化率作为结果
- 返回限制：系统将返回指定数量的数据点生成图表，最大值10000。

4. 点击“确定”，系统将根据配置筛选数据并生成图表。可以通过图表右上角在“图”和“表”之间进行切换，具体示例请参看生成示范数据库图表。

> 注意:
>
> 若单次查询时间超过50s，系统将自动终止本次查询。如果出现查询超时，可以采取以下措施优化查询：
>
> - 减少单次请求中query个数
> - 缩短单个query中起止时间的间隔
> - 减少单个query涉及到的时间序列数目

- 对数据进行预处理

当TSDB中存储的数据量较大时，将数据按照用户指定的规则筛选出来需要较长的等待时间，有可能请求导致超时，造成查询失败。

针对这种情况，用户可以配置数据预处理，提前将相关数据过滤、聚合好，实现快速返回查询结果。有关预处理的配置方法，请查看数据预处理

查看预处理结果

预处理规则建立并已进入生效状态后，用户需要按照查看预处理结果中的条件，在查询面板中建立查询查看结果。以下是基于示范数据库建立的预处理规则的查询示例。

1. 选择“产品服务>时序数据库TSDB>数据库名称”，进入数据库详情页面。
2. 点击数据库名称，进入数据库详情页。点击“预处理”页签，进入配置页面。
3. 点击“查询详情”，查看预处理规则。

查看预处理规则时，注意规则信息一栏中的条件，避免产生执行查询后未命中预处理规则的情况。

4. 返回数据库列表，点击“查询面板”。
5. 在查询面板页面，点击“添加查询项”，按照预处理规则配置查询项。

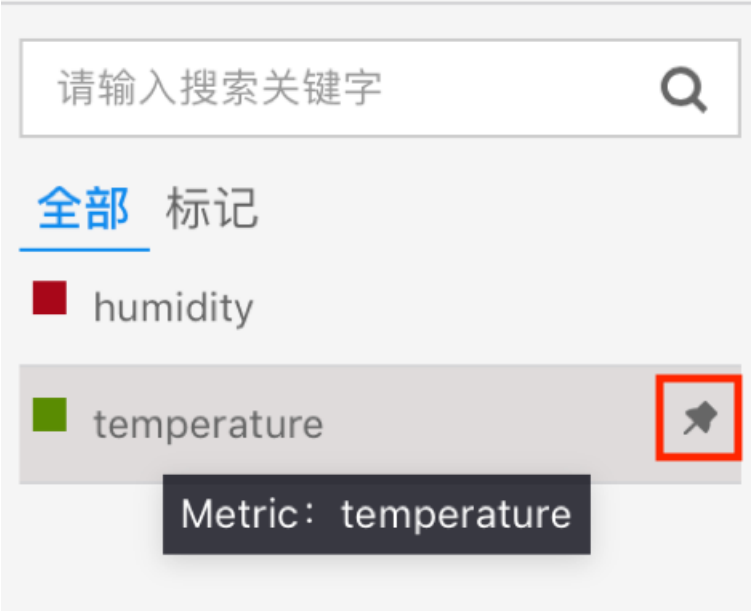
6. 查看生成结果，此时显示已经命中预处理规则(高亮部分)。

7. 可再次返回预处理查询详情页面，查看累积命中次数。

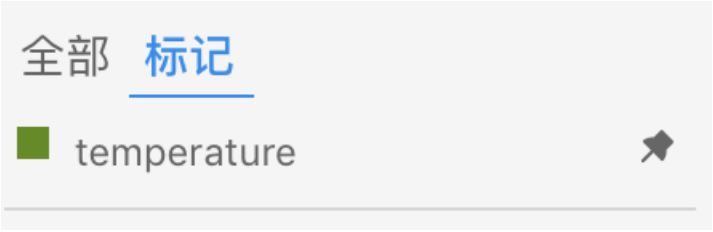
标记

如果出现查询面板建立查询项过多的情况，用户可以选择对重要的查询项进行标记，便于在标记一栏查看这些查询项。

1. 选择“产品服务>时序数据库TSDB>数据库名称”，进入数据库详情页面。
2. 点击“查询面板”进入查询页面。
3. 在查询列表中任意选中一个查询项，注意右边钉子形状图案。



4. 点击图案为选中的查询项添加标记。
5. 在“标记”一栏查看标记过的查询项。



使用API入门

TSDB提供了详细的API文档来写入、查询和分析数据，以及配置各种管理数据。点击查看完整的 API文档 。

操作指南

数据库管理

数据库的操作包括创建数据库，查看数据库，监控数据库，删除数据库。

创建数据库

时序数据库是用于存储和管理时间序列数据的专业化数据库，为时间序列数据提供高性能、稳定可靠的数据库服务。时序数据库特别适用于物联网场景。

1. 登录百度智能云官网，点击右上角的“管理控制台”，快速进入控制台界面。

2. 选择“产品服务>时序数据库TSDB”，进入“数据库列表”页面。

3. 点击“创建数据库”，进入创建数据库页面，填写配置信息，包括：

   - 数据库名称：设置数据库名称，由小写字母和数字组成，6～40个字符。
   - 写入额度：目前支持1～1000000百万点/月。
   - 查询额度：每扫描1000个数据点计为一个查询单位
   - 长数据格式存储空间：只有写入数据为string或byte类型时才需要勾选，且string或byte类型的数据点仍算在写入额度中
   - 购买时长：TSDB为预付费产品，最少每次购买1个月。

4. 完成配置后点击“确认订单”，完成TSDB数据库的创建。

5. 返回“数据库列表”页面

> 注意:
>
> TSDB实例的创建仅支持主用户操作，开通后子用户可以使用实例服务

查看数据库

查看数据库基本信息，可以对数据库做升级配置，续费等操作。

1. 选择“产品服务>时序数据库TSDB”，进入“数据库列表”页面。

2. 点击数据库名称，进入数据库“基本信息”页面。

查看详情

可以查看数据库的基本信息，配置信息和支付信息。基本信息包括数据库名称、数据库ID、描述和域名；配置信息包括写入额度，最大时间序列；支付信息包括支付方式，到期时间和创建时间。

升级配置

点击“升级配置”，升级各个计费项。

续费

点击“续费”，为当前的数据库进行续费，可按月度进行续费（暂不支持多数据库实例批量续费）。

自动续费

TSDB支持自动续费（暂不支持多数据库实例批量续费），功能详细介绍见（TSDB资源到期停服30天后删除资源）：https://cloud.baidu.com/doc/Finance/s/gjwvysrlu 。

- 新建数据库实例可在创建时选择开通自动续费

- 已创建的数据库实例可在百度智能云控制台的“财务中心”-“续费管理”中开通自动续费；在TSDB控制台的“数据库列表”页面中点击“自动续费”可跳转至“续费管理”

监控数据库

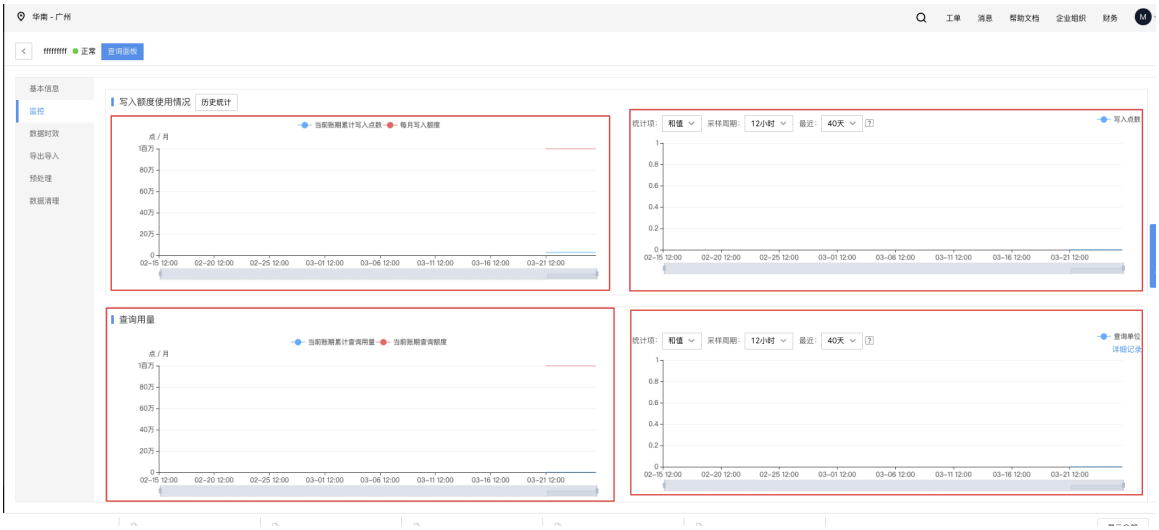
用户可以实时获取TSDB当前的监控信息，了解数据库的使用情况。

1. 选择“产品服务>时序数据库TSDB”，进入“数据库列表”页面。
2. 点击“监控”能进入监控页面。或是点击数据库名称，进入数据库详情页面，点击“监控”页签，进入监控页面。

   可以看到多个维度的监控信息。系统每分钟产生一个监控数据。用户可以通过调整”统计项“和”采样周期“来获取不同维度的统计信息。例如”采样周期“设为1小时，则每个采样周期内会有60个监控数据；如果”统计项“设置的是平均值，则图中每个采样点代表60个监控数据的均值。 采样周期包括：1分钟、5分钟、20分钟、1小时、6小时、12小时和1天。

统计项包括：

- 平均值：采样周期内所有采样点的平均值。
- 和值：采样周期内所有采样点的取值之和。
- 最大值：采样周期内所有采样点中的最大值。
- 最小值：采样周期内所有采样点中的最小值。
- 样本数：采样周期内的样本数。



从监控面板上，用户可实时获取以下信息：

- 写入点数
- 查询用量
- String/Byte型数据存储空间
- 时间序列
- 写入/读取请求数
- 总存储点数

删除数据库

勾选需要删除的数据库，点击“删除”按钮，点击“确定”，将删除选择的数据库。

数据管理

数据管理包括数据导入，导出以及清理数据。

数据导入

导入示范数据

示范数据用于帮助用户体验时序数据的各项功能。示范数据含有2015年全年北京、上海、广州3个城市每一个空气监测站的每一天的早晨8点采集的温度、湿度、风力、PM2.5、紫外线指数等气象模拟数据（非真实数据），每一个数据点所属城市城市的名称、邮编，以及该监测站的地理位置（经纬度）等标签，方便您快速体验、理解时序数据库各项查询、聚合及图表展示功能。

1. 选择“产品服务>时序数据库TSDB>数据库名称”，进入数据库详情页面。
2. 点击“导入>导入示范数据>导入数据”，在弹出对话框中点击“确认”，完成导入示范数据操作。导入后可在下方查看到对应的导入记录。

有关示范数据库的操作示例，请查看生成示范数据库图表。

导入数据

用户可以将前期导出到BOS中的数据重新导入TSDB，也可以导入用户本地的数据。如果从本地导入用户自有数据，需先将数据文件上传至BOS，文件格式为.CSV。

> 注意：
>
> 同一个metric的不同field不要求在同一个导入任务中全部导入，而是可以分多次导入。只要导入的metric、timestamp、tag值相同，即可在一次查询中，同时查询出通过多次导入的多个field。

用户的自有数据中必须包含“metric”、“timestamp”、至少一个“field”和至少一个“tag”。其中：

- 第一列表头必须为“metric”，表示数据点的metric。
- 第二列表头必须为“timestamp”，表示数据点的timestamp，单位为毫秒为格林威治时间1970年01月01日00时00分00秒(北京时间1970年01月01日08时00分00秒)起至现在的总毫秒数。
- 第三列及之后的列：

  * 如果表头包含冒号“:”，则表示该列为数据点的field，冒号前的是field名称，冒号后的是field类型（支持Number、String、Bytes，其中Bytes类型的列的值需要进过base64编码）。如下图中的“value:Number”。

  * 如果表达不包含冒号，则表示该列为数据点的tag，表头为tag名称，列的内容为tag值。下图红框内容为tag名称。

	A	B	C	D	E	F	G
1	metric	timestamp	value:Number	city	zip_code	latitude	longitude
2	temperature	1420070400000	-1	北京	100000	39.913078	116.39718
3	humidity	1420070400000	29	北京	100000	39.913078	116.39718
4	wind	1420070400000	11	北京	100000	39.913078	116.39718
5	precipitation	1420070400000	0	北京	100000	39.913078	116.39718
6	pm25	1420070400000	36.5	北京	100000	39.913078	116.39718
7	temperature	1420070400000	-2	北京	100000	40.051422	116.300488
8	humidity	1420070400000	26	北京	100000	40.051422	116.300488
9	wind	1420070400000	10	北京	100000	40.051422	116.300488
10	precipitation	1420070400000	0	北京	100000	40.051422	116.300488
11	pm25	1420070400000	41.9	北京	100000	40.051422	116.300488
12	temperature	1420070400000	-4	北京	100000	39.881315	116.414208

1. 选择“产品服务>时序数据库TSDB>数据库名称”，进入数据库详情页面。
2. 点击“导入>导入备份数据>导入数据”，在弹出对话框中选择指定的Bucket名称和文件，并设置编码格式；点击“确认”，完成导入示范数据操作。

数据导出

TSDB提供手动导出和自动导出两种数据导出方法。

手动导出数据

通过导出功能可以将当前数据导出至BOS Bucket。BOS(Baidu Object Storage)即百度对象存储，可以为用户提供稳定、安全、高效以及高扩展存储服务。Bucket即BOS中的一个命名空间。

在导出数据之前，用户应先开通对象存储服务并创建Bucket。

导出数据的具体操作如下：

1. 选择“产品服务>时序数据库TSDB”，进入“数据库列表”页面。
2. 点击数据库名称，进入数据库详情页面；点击“导入导出>导出>手动导出数据”页签，进入导出操作页面。

3. 点击“创建导出”，在弹出窗口配置以下信息，包括：
   - 导出地址：存放导出文件的BOS Bucket地址。如果用户已创建BOS Bucket，此时可在“导出地址”的下拉列表中选择指定的Bucket名称；如果用户尚未创建BOS Bucket，也可点击“新建Bucket”，快速创建Bucket。
   - 时间范围：可选择导出全部数据或仅导出指定时间范围内的数据。
   - 度量：可导出所有度量的数据或仅导出指定度量的数据。
   - 导出文件格式：目前只支持CSV格式。
   - 导出文件个数：支持将所有数据导出为一个文件（数据量大时因为bos文件大小限制会拆分为多个文件）或将数据并行导出为多个文件（适合大数据导出）。

   完成配置后点击“确定”完成数据导出操作。

4. 完成导出操作后，用户可返回至导出操作页面，查看导出结果。此时可点击“查看详情”，获取导出的详细信息及对应的BOS Bucket地址。

自动导出数据

打开自动导出功能后，系统将在每日凌晨自动导出前一日至当日00：00时间范围内的数据，具体操作如下：

1. 选择“产品服务>时序数据库TSDB”，进入“数据库列表”页面。
2. 点击数据库名称，进入数据库详情页面；点击“导入导>导出>自动导出数据”页签，进入导出操作页面。

3. 打开“自动定时导出”开关，并设置“导出地址”和“导出文件个数”。
   - 导出地址：存放导出文件的BOS Bucket地址。如果用户已创建BOS Bucket，此时可在“导出地址”的下拉列表中选择指定的Bucket名称；如果用户尚未创建BOS Bucket，也可点击“新建Bucket”，快速创建Bucket。
   - 导出文件个数：支持将所有数据导出为一个文件（数据量大时因为bos文件大小限制会拆分为多个文件）或将数据并行导出为多个文件（适合大数据导出）。
4. 点击“保存”，使配置生效。

数据清理

> 注意：数据清理主要是供删除调试的测试数据使用，对正式数据请谨慎使用。

选择“产品服务>时序数据库TSDB”，进入“数据库列表”页面。

点击数据库名称，进入数据库详情页面；点击“数据清理”页签，进入数据清理页面。

系统默认给出最近1小时内的数据清理信息，如果需要清理更多内容，可点击右面下拉菜单，切换时间范围。

数据时效

选择“产品服务>时序数据库TSDB”，进入“数据库列表”页面。

点击数据库名称，进入数据库详情页面；点击“数据时效”页签，进入数据时效页面。

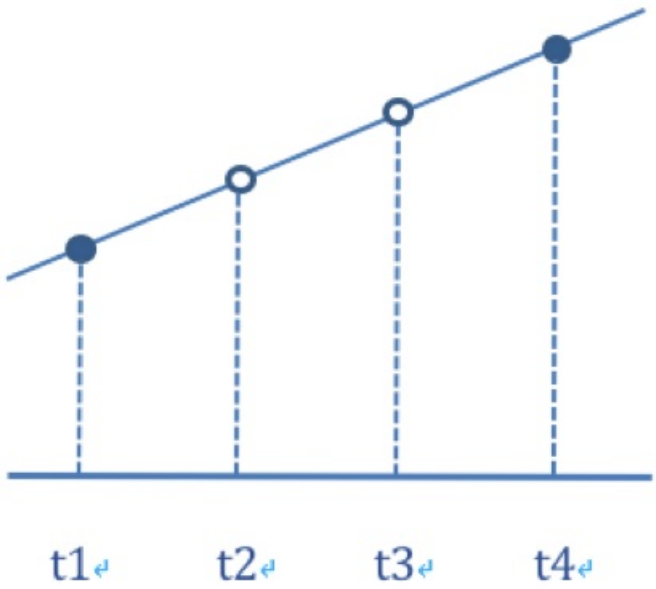
将数据时效设置为 ON（开启），设置时效范围，以天为单位，保存后，系统会保留时效内的数据。

> 注意：
>
> 启用数据时效功能，系统将自动删除时效范围之外的数据。

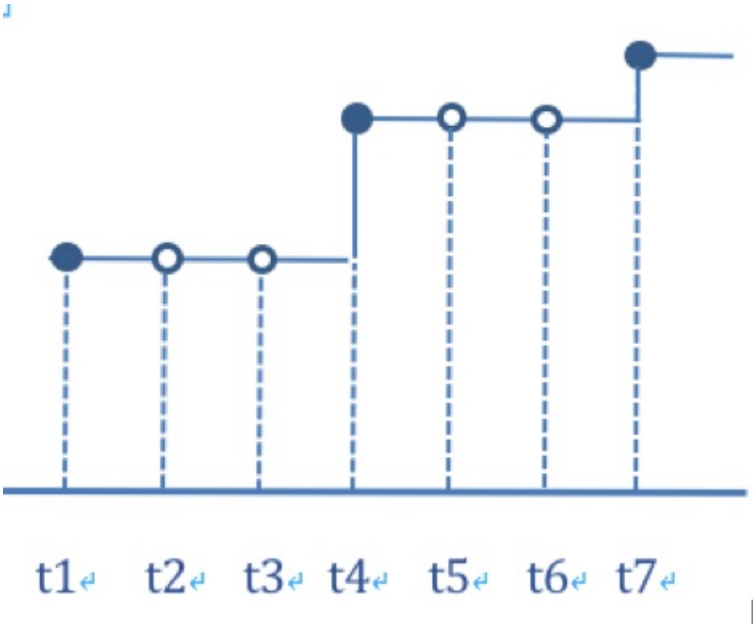
插值查询

在查询时可以将数据进行插值，即使没有数据写入，也可以将数据按照一定的插值算法查询出来。目前支持Linear（线性插值）、Previous（按前一个值插值）、Fixed（固定值插值）三种插值算法。

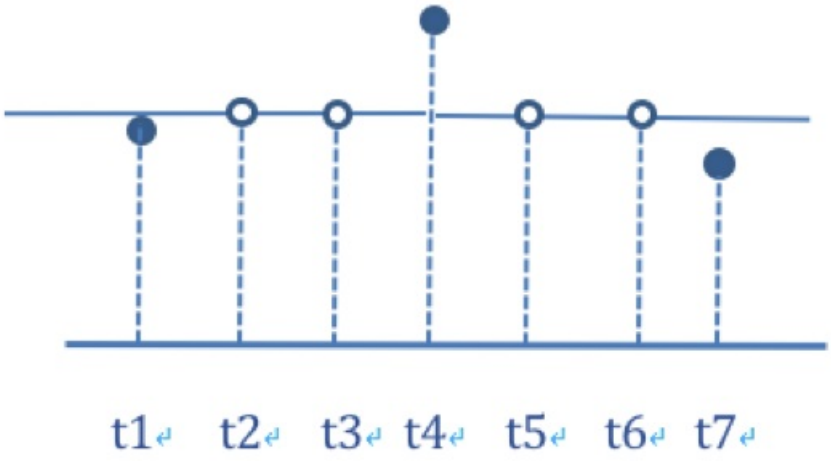
Linear:



Previous:



Fixed:

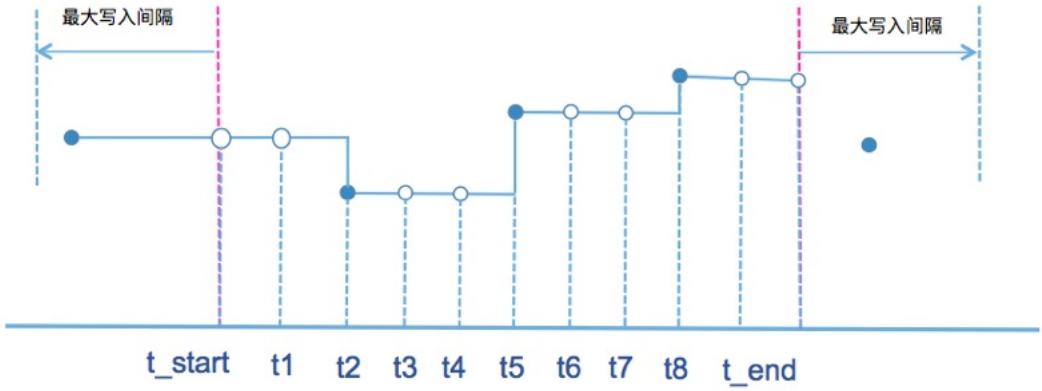


查询时，需要设定最大写入间隔，如15min，即一个时间序列数据的最大写入间隔，TSDB认为在此间隔内必然有值。系统会尝试查找从(start - maxWriteInterval)到start，以及end到(end + maxWriteInterval)的点。

如果(start - maxWriteInterval)到start找不到点，则start到end间第一个点之前的缺少的点会按第一个点的值进行插值；

如果end到(end + maxWriteInterval)找不到点，则start到end间最后一个点之后的缺少的点会按最后一个点的值进行插值。

用Previous插值来举例，如下图，实心的点是实际上传的点，空心的点是系统补的点。



数据预处理

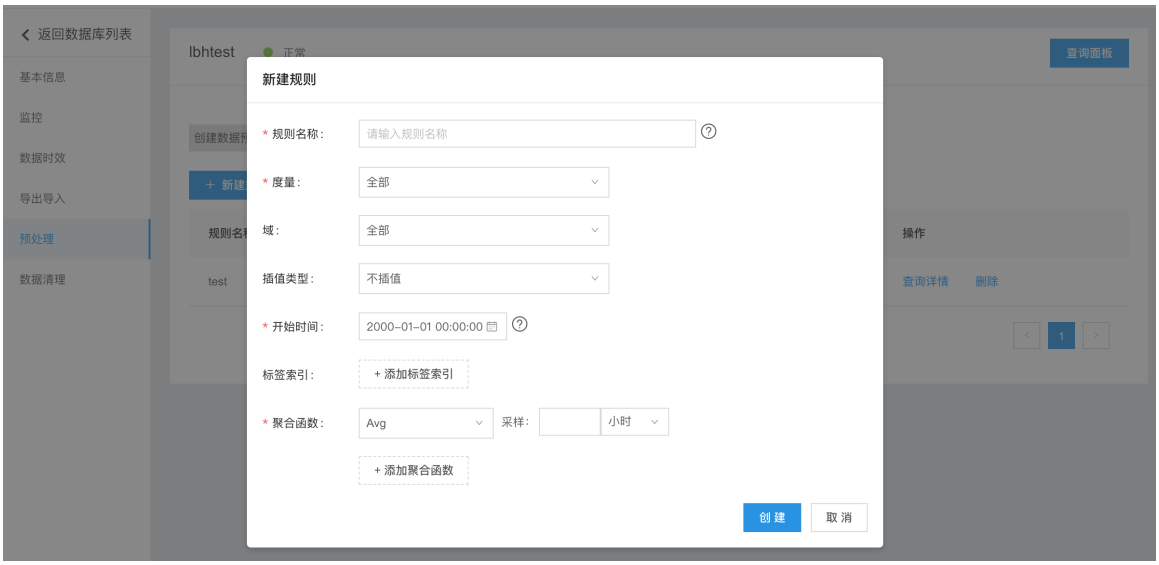
新建预处理规则

当TSDB中存储的数据量较大时，将数据按照用户指定的规则筛选出来需要较长的等待时间，有可能请求导致超时，造成查询失败。

针对这种情况，用户可以配置数据预处理，提前将相关数据过滤、聚合好，实现快速返回查询结果。

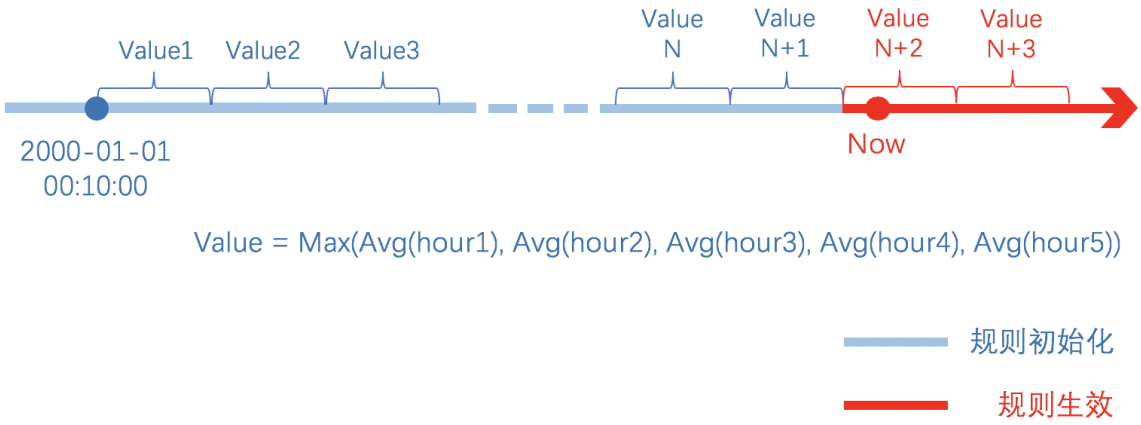
> **注意：**
>
> - 免费数据库无法创建预处理规则。
> - 每个付费数据库只能创建2个预处理规则。
> - 预处理规则标签个数上限5个。
> - 预处理规则多选metric上限10个，仅支持单选或全选。
> - 预处理规则聚合函数上限3个。
> - 预处理规则支持修改重置，重置的间隔时间必须要大于1个小时。

1. 选择“产品服务>时序数据库TSDB>数据库名称”，进入数据库详情页面。
2. 点击数据库名称，进入数据库详情页。点击“预处理”页签，进入配置页面。
3. 点击“新建规则”，在弹出窗口中配置预处理规则。



具体包括以下配置项：

- 规则名称：设置规则名称，用来标识具体规则。
- 度量：从下拉列表中选择当前数据库中已有的Metrics名称，或者选择全部Metrics。
- 开始时间：预处理操作计算原始数据的起始基准时间。在上图配置中，预处理规则原始数据的起始基准时间为 2000-01-01 00:10:00，会以此作为开始时间，先对 2000-01-01 00:10:00 至今的所有数据点进行聚合（对齐到5小时）。如下图所示：



在规则初始化过程中，此时预处理规则尚未生效，查询请求不会命中预处理数据点。规则初始化结束后，预处理规则会进入生效状态，此时匹配的查询请求会命中预处理数据点。规则修改之后，会进入重置状态，重置结束之后，查询请求才会命中。

- 标签索引：指定标签对数据进行过滤，预处理规则标签个数上限5个。
- 聚合函数：通过内置函数对数据进行处理，预处理规则聚合函数上限3个，且必须包含至少一个带采样时间的聚合函数。当前支持的聚合函数包括：
  - Avg：以每个采样时间范围内的value的平均值作为结果
  - Dev：以每个采样时间范围内的value的标准差作为结果
  - Count：以每个采样时间范围内的点的数目作为结果
  - First：以每个采样时间范围内的第一个value作为结果
  - Last：以每个采样时间范围内的最后一个value作为结果
  - LeastSquares：对每个采样时间范围内的value进行最小二乘法的拟合
  - Max：以每个采样时间范围内的value的最大值作为结果
  - Min：以每个采样时间范围内的value的最小值作为结果
  - Percentile：每个采样时间范围内的value的第p百分位数作为结果。
  - Sum：以每个采样时间范围内的value的总和作为结果
  - Diff：以每两个相邻的value的差值作为结果
  - Div：以每个value除以一个除数作为结果
  - Scale：以每个value乘以一个倍数作为结果
  - Rate：以每两个相邻的value在单位时间的变化率作为结果
  - AdjacentUnique：相同的值只保留第一个，非相邻的值不去重

查看预处理详情

新建预处理规则后，可以在规则详情中查看数据预处理情况。

1. 选择“产品服务>时序数据库TSDB>数据库名称”，进入数据库详情页面。
2. 点击数据库名称，进入数据库详情页。点击“预处理”页签，进入配置页面。
3. 找到已经创建的预处理规则，点击“查看详情”，可以获得以下配置信息，如下图所示：

- 累计命中次数：通过查询面板或者API/SDK查看数据点时，命中该规则的次数。
- 累计原始点数：根据预处理规则中的度量值和标签，从数据库中筛选出的原始数据点数。
- 累计规则点数：对“累计原始点数”进行聚合后获得的数据点数。

关于其他信息的解释，请参看新建预处理规则。

查看预处理结果

用户可以通过查询面板、SDK或API查看预处理后的数据。为了确保查询时能够命中预处理后的数据，需要满足以下条件：

1. 查询规则的度量值必须是预处理规则的子集（预处理规则中可指定全部度量值或某一个独立的度量值）。
2. 查询规则中标签过滤加上分组标签指定的tags必须和预处理规则中的标签索引相等。
3. 查询规则中的聚合函数设置必须与预处理规则一致。

通过查询面板生成图表的具体操作请参看生成图表。

与天工产品对接

通过物联网核心套件IoTCore将数据写入TSDB

物联网核心套件IoTCore提供设备数据接入云端的能力，接入后的数据可写入tsdb中进行存储及查询，使用方法请[参考文档]。

通过物可视展示TSDB数据

物可视是基于图形用户界面的数据可视化开发工具，支持拖拽式的交互式设计器，帮助用户无门槛零编程地开发可视化应用。物可视提供对接TSDB实现数据可视化的能力，使用方法请[参考文档]。

多用户访问控制

介绍

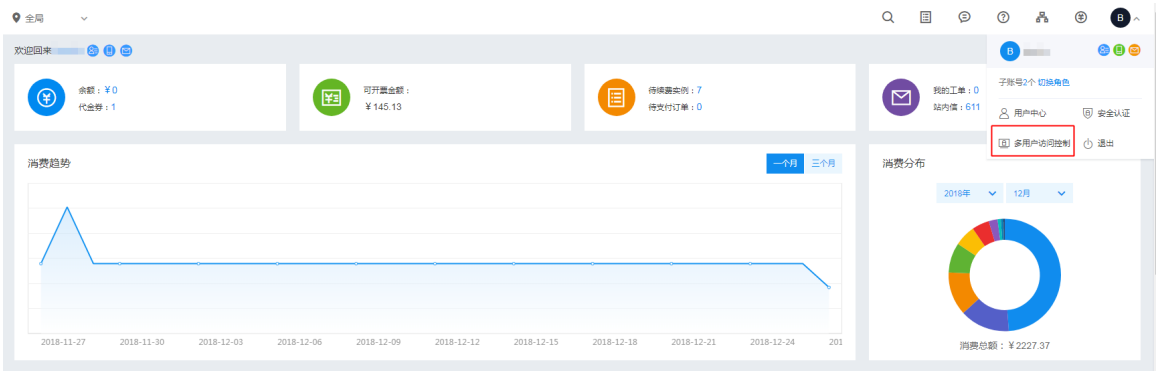
多用户访问控制，主要用于帮助用户管理云账户下资源的访问权限，适用于企业内的不同角色，可以对不同的工作人员赋予使用产品的不同权限，当您的企业存在多用户协同操作资源时，推荐您使用多用户访问控制。

适用于下列使用场景：

- 中大型企业客户：对公司内多个员工授权管理；
- 偏技术型vendor或SAAS的平台商：对代理客户进行资源和权限管理；
- 中小开发者或小企业：添加项目成员或协作者，进行资源管理。

创建用户

1. 主账号用户登录后在控制台选择“多用户访问控制”进入用户管理页面。



2. 在左侧导航栏点击“用户管理”，在“子用户管理列表”页，点击“创建子用户”。

3. 在弹出的“创建子用户”对话框中，完成填写“用户名”和确认，返回“子用户管理列表”区可以查看到刚刚创建的子用户。

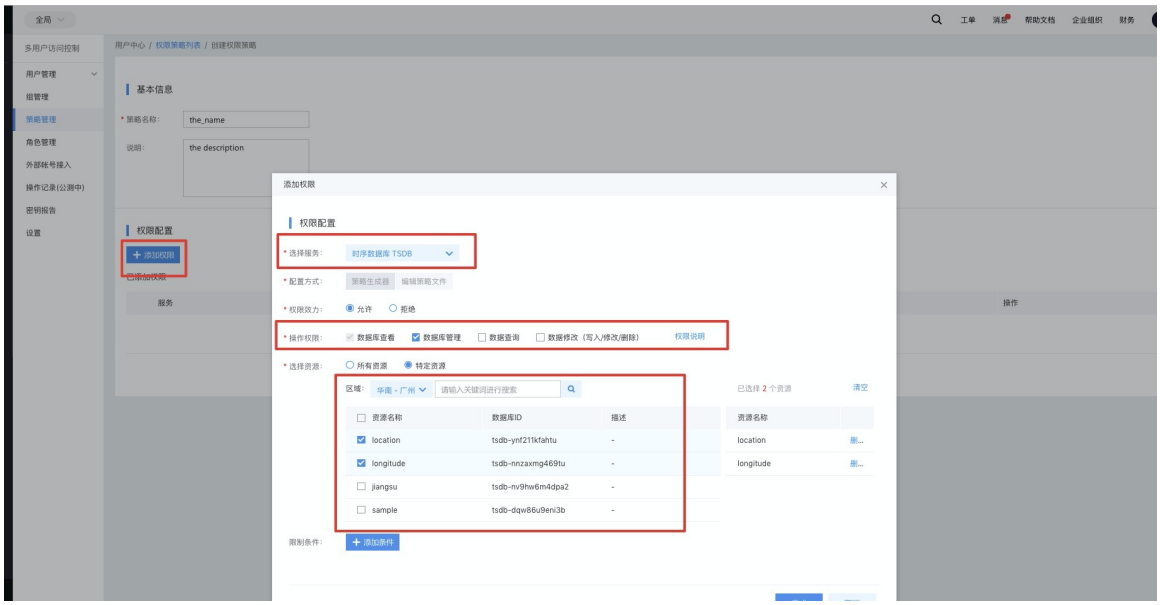
配置策略

TSDB仅支持用户自定义策略，用户可以自己创建权限集，针对单个资源配置权限，灵活的满足账户对不同用户的差异化权限管理。

在多用户访问控制里，点击左侧导航栏“策略管理”，点击“创建策略”。

选择按策略生成器创建

用户填写策略名称并点击“权限配置”，选择服务类型为“时序数据库TSDB”，之后选择所需的操作权限即可。

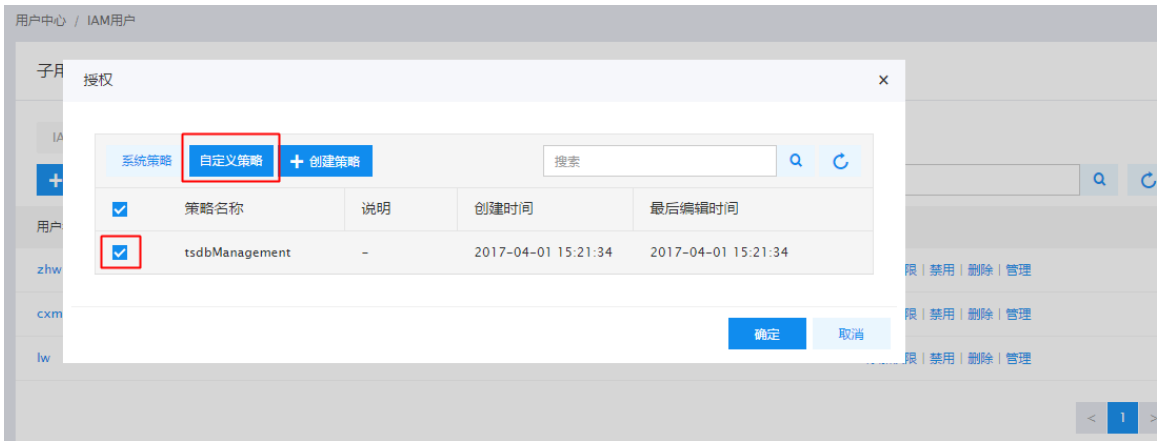


自定义权限范围详细如下：

权限名称	权限内容
数据库查看	查看数据库基本信息 查看自动导出设置 查看自定义查询 查看预处理规则 查看导入\导出\数据清理任务 查看度量 查看标签 查看监控图表
数据库管理	修改数据库描述 修改自动导出设置 创建或删除自定义查询 创建、删除或修改预处理规则 取消导入\导出\数据清理任务
数据查询	查询数据点 创建数据导出任务
数据修改	写数据点 创建数据清理任务（删除数据点） 创建数据导入任务

用户授权

在“用户管理->子用户管理列表页”的对应子用户的“操作”列选择“添加权限”，并为用户选择系统权限或自定义策略进行授权。



> **说明：**如果在不修改已有策略规则的情况下修改某子用户的权限，只能通过删除已有的策略并添加新的策略来实现，不能取消勾选已经添加过的策略权限。

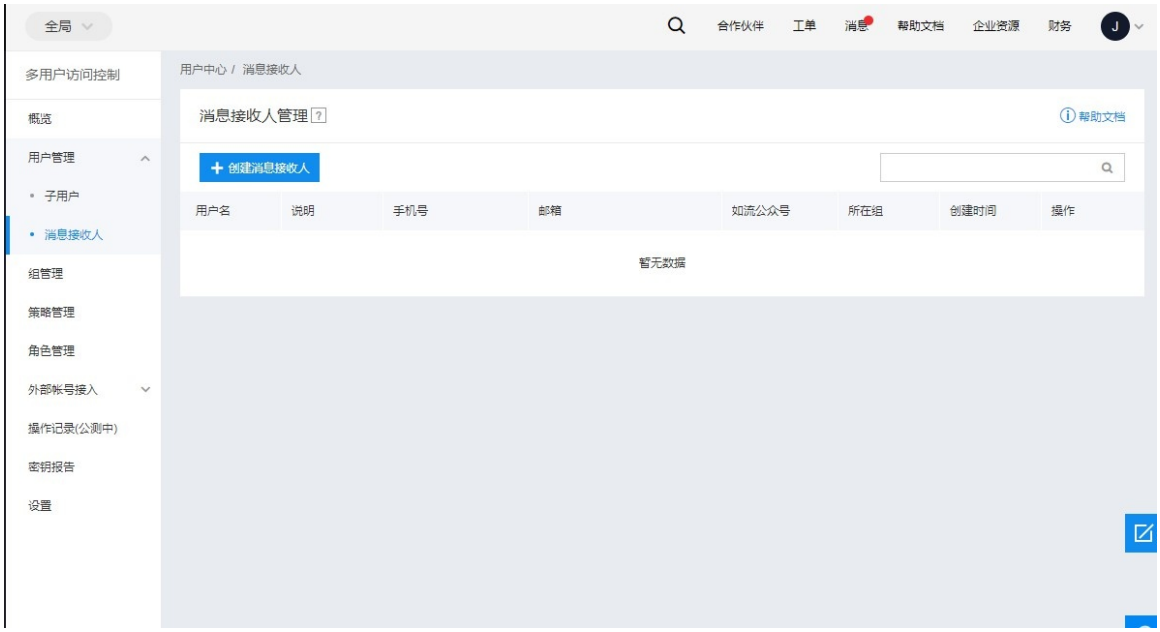
子用户登录

主账号完成对子用户的授权后，可以将链接发送给子用户；子用户可以通过IAM用户登录链接登录主账号的管理控制台，根据被授权的策略对主账户资源进行操作和查看。

消息推送

- 同一个账号配置多个消息接收端

在百度智能云控制台中，通过选择“多用户访问控制>用户管理>消息接收人>创建消息接收人”，添加用户名、手机号、邮箱等信息，用于接收该账号下通知信息，接收消息配置可通过消息接收人管理中的消息订阅进行编辑。创建的消息接收人仅用于消息接收，不能用于控制台登录。



- 子用户接收主用户资源的消息推送

对于主用户已有的子用户，可通过“消息中心>消息接收设置>接收设置”,对接收人进行编辑，勾选希望接收到消息推送的子用户即可。

其他详细操作参考：多用户访问控制。

数据可视化

目前百度智能云时序数据库提供了两种数据可视化方案，**物可视**和**Sugar**，可以根据自己的业务需求来灵活选择。

两种方案的使用场景如下：

- TSDB+物可视：

物可视适合于前端开发人员快速搭建可视化应用，可以加速前端可视化的开发进度。物可视是由百度智能云提供的可视化产品，是一个基于物联网场景的可视化设计器，无需部署和安装插件，可直接对接百度智能云时序数据库及物管理，支持组态和大屏设计，并可以将设计好的面板通过JS代码嵌入到其他可视化应用中。具体操作请参见物可视操作指南。

- TSDB+Sugar：

Sugar是百度智能云推出的敏捷 BI 和数据可视化平台，目标是解决报表和大屏的数据可视化问题，解放数据可视化系统的开发人力。Sugar提供界面优美、体验良好的交互设计，通过拖拽图表组件可实现5分钟搭建数据可视化页面。平台支持直连多种数据源（Excel/CSV、MySQL、SQL Server、PostgreSQL、Hive、Spark、Presto、Teradata、Baidu Palo、Baidu TSDB等），还可以通过API、静态JSON方式绑定可视化图表的数据，简单灵活,大屏与报表的图表数据源可以复用，用户可以方便地为同一套数据搭建不同的展示形式。

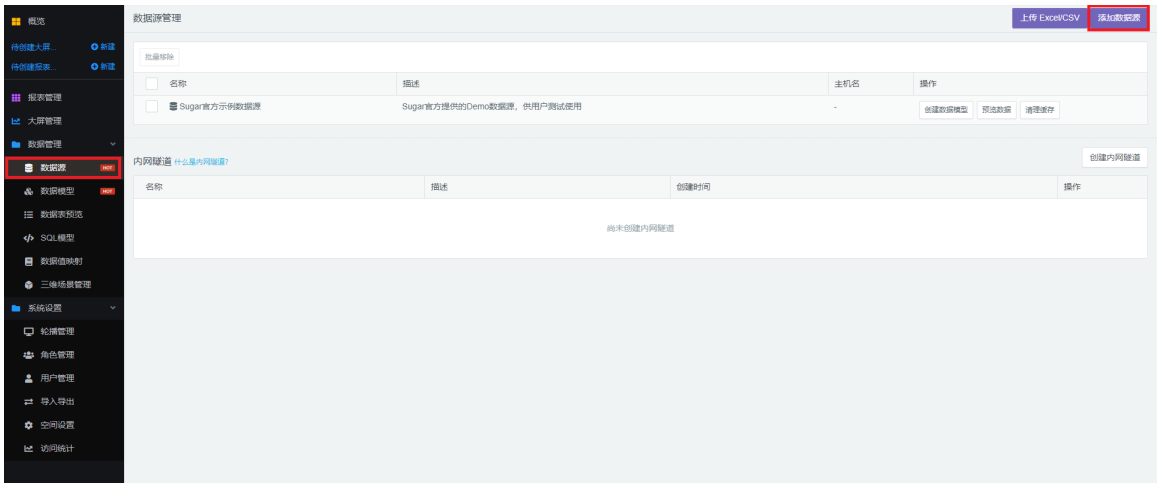
本文重点介绍TSDB利用Sugar做可视化展示的方法。用户可以利用Sugar的可视化能力，更好地分析存储在百度智能云TSDB中的数据。

使用说明

TSDB已接入百度智能云Sugar，用户可以通过SugarBI访问TSDB，对TSDB中存储的数据进行多种交互式数据分析。百度智能云Sugar对所有新老用户提供30天免费全功能试用，30天后需付费使用，请参见Sugar产品定价。

1. 创建数据源

- 登录Sugar会自动进入「空间广场」，点击「我的空间」中的任意空间，进入该空间页面。
- 点击空间页面左侧的「数据管理」>「数据源」>「添加数据源」，添加百度TSDB数据源。



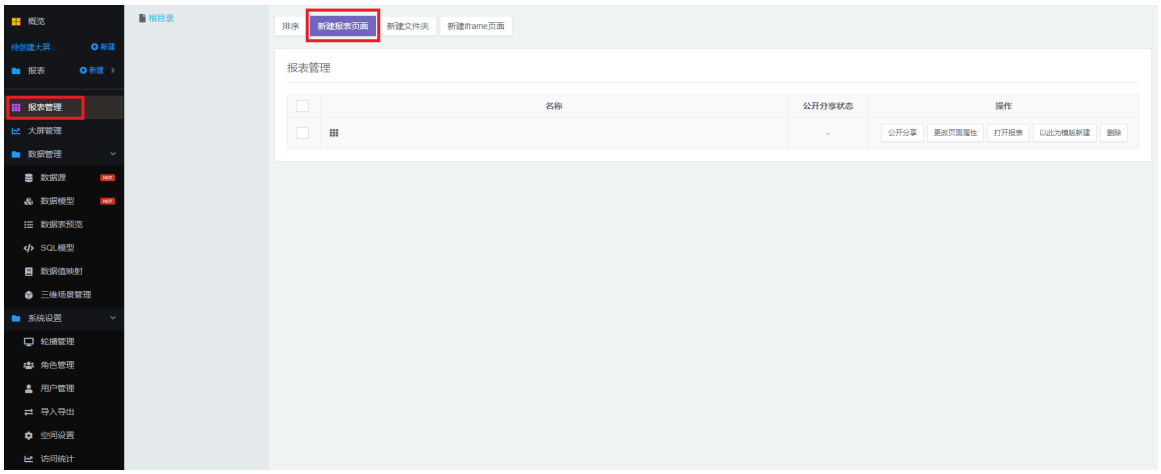
- 在弹出的数据源编辑框中，「类型」选择“Baidu TSDB”，「数据源名称」填写用户自定义的数据源名称，「数据库名」、「地域」填写用户名下想要接入的TSDB实例的名称和所在的区域，底部的「用户名」和「密码」填写用户在TSDB中创建的MySQL协议账号信息。TSDB创建MySQL协议账号具体操作请参见XXXX。

2. 报表示例

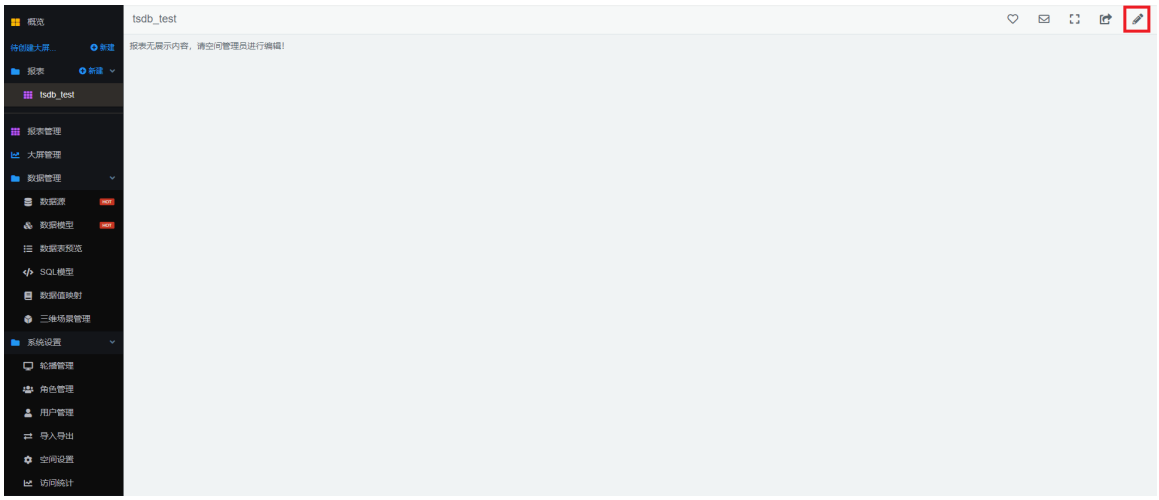
接下来简单说明一下如何制作报表来分析TSDB中存储的数据。

创建报表

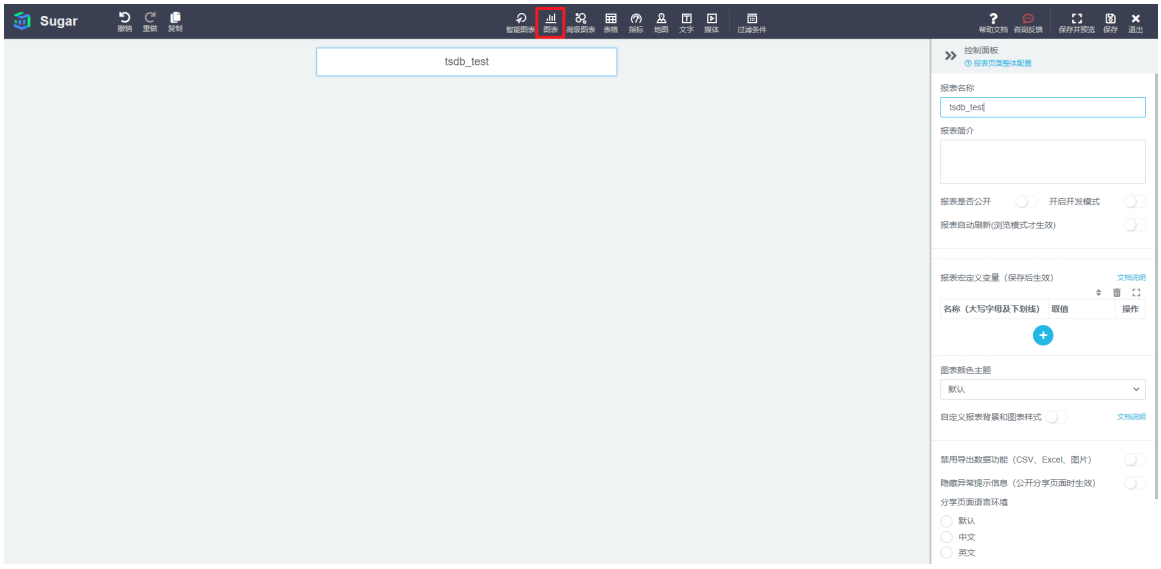
- 首先，在左侧的管理中心中进入「报表管理」，新建一个报表页面。



- 在「报表管理」列表中，选中你想要编辑的报表，点击「打开报表」进入报表详情页面。
- 在报表详情页面，点击右上角的编辑按钮可以进入编辑模式。



- 现在我们可以添加自己喜欢的图表并为其绑定SQL模型来展示我们的数据了。如何添加图表并绑定数据。



示例应用

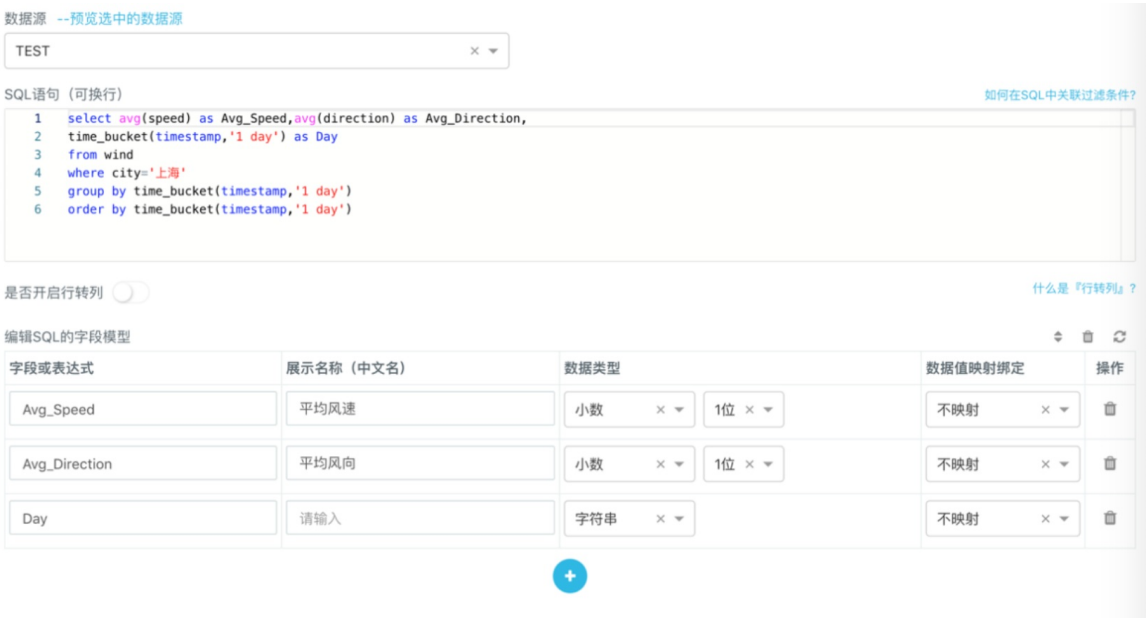
这里我们使用的示例数据源含有2015年全年北京、上海、广州3个城市每一个空气监测站的每一天的早晨8点采集的有关风的气象模拟数据（非真实数据）。在TSDB中这样存储，`Wind`作为`metric`，表示TSDB存储的是有关风的气象数据，`metric`下有`speed`速度和`direction`方向两个域`field`，用每一个数据点所属城市的名称、邮编，以及该数据点对应的监测站的地理位置（经纬度）作为`tag`标签。

根据以上示例数据，我们可以根据以下几种场景制作合适的报表。

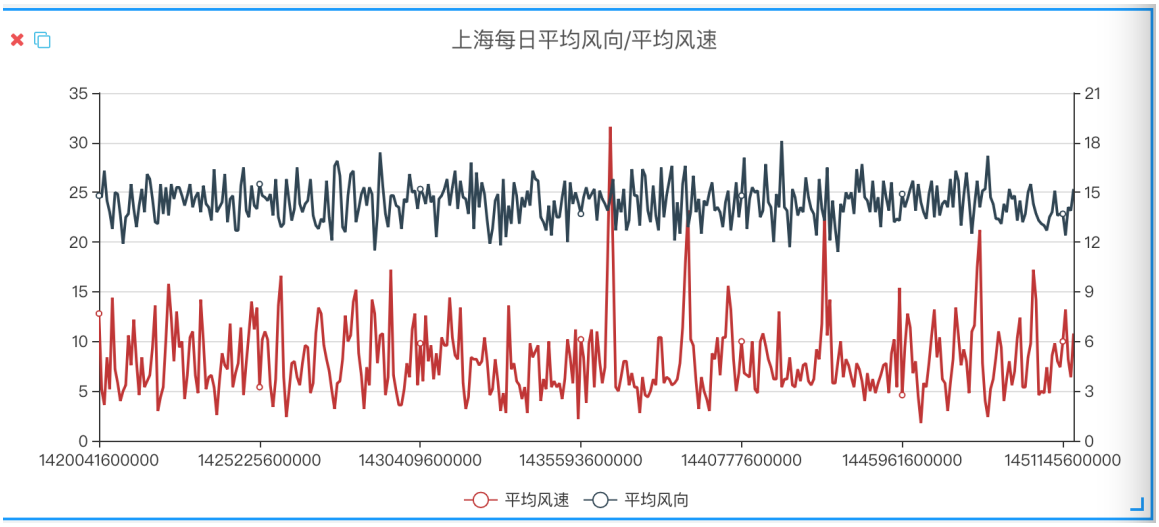
场景一： 制作报表分析2015年内，上海市每天的风向以及平均风速。

- 首先，新建一张图表；为了观察趋势变化，这里我们选择了折线图。
- 在折线图对应的「控制面板」>「数据」>「SQL建模」里新建一个SQL模型。
- 在模型编辑框中输入以下SQL查询语句并设置好字段模型。
- 最后回到「控制面板」>「数据」中为X轴、Y轴绑定好对应数据，点击刷新即可生成我们需要的报表。

```
 select avg(speed) as Avg_Speed, avg(direction) as Avg_Direction,
time_bucket(timestamp, '1 day') as Day
from wind
where city = '上海'
group by time_bucket(timestamp, '1 day')
order by time_bucket(timestamp, '1 day')
```



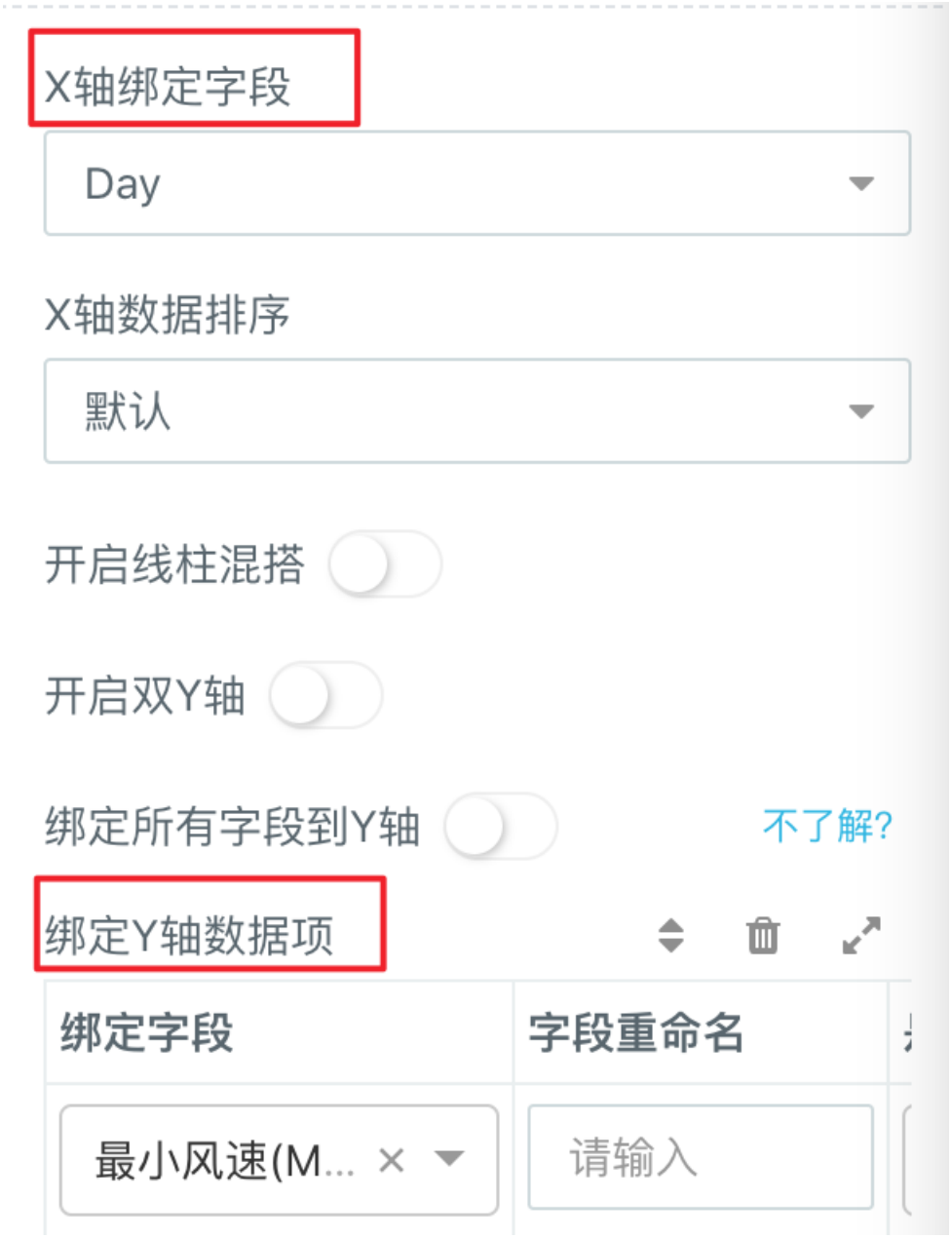

X轴绑定字段
Day
X轴数据排序
从小到大
开启线柱混搭
开启双Y轴
绑定所有字段到Y轴
不了解?
绑定Y轴数据项
绑定字段
字段重命名
平均风速(A...
请输入
平均风向(A...
请输入

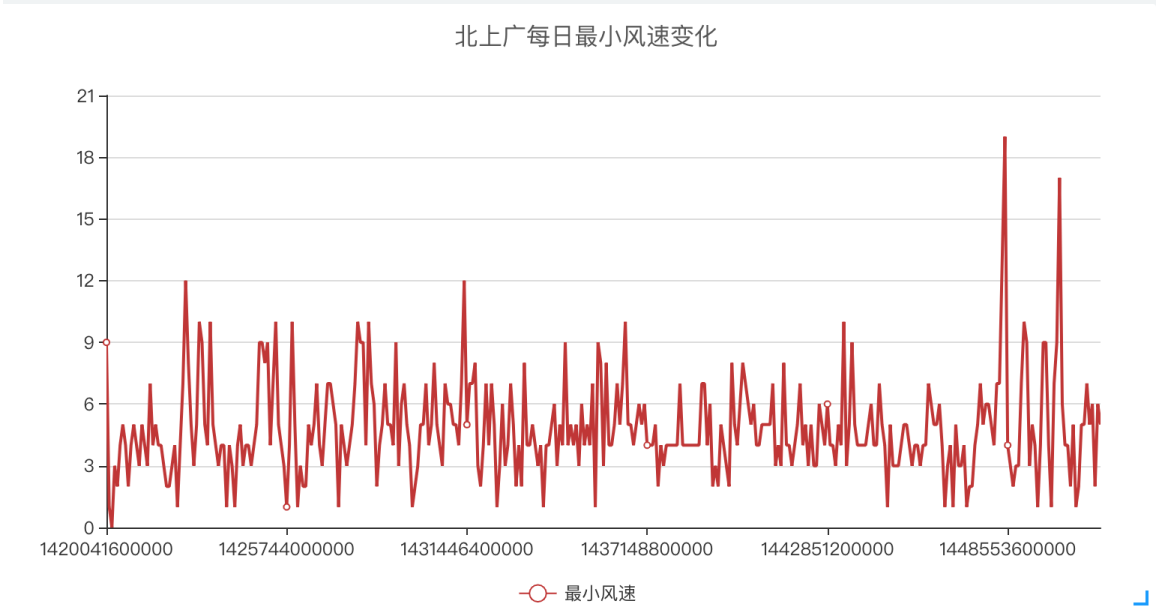


场景二： 制作报表分析2015年内，每天北京、上海、广州三个城市内所有监测点所采集到的最小风速。

- 首先，新建一张图表；为了观察趋势变化，这里我们选择了折线图。
- 在折线图对应的「控制面板」>「数据」>「SQL建模」里新建一个SQL模型。
- 在模型编辑框中输入以下SQL查询语句并设置好字段模型。
- 最后回到「控制面板」>「数据」中为X轴、Y轴绑定好对应数据，点击刷新即可生成我们需要的报表。

```
select min(speed) as Min_Speed, time_bucket(timestamp, '1 day') as Day
from wind
where city in ('北京','上海','广州')
group by time_bucket(timestamp, '1 day')
order by time_bucket(timestamp, '1 day')
```





聚合查询

这里需要注意的是，做聚合查询时，分组时的字段名必须对接time_bucket函数，根据数据采集的周期适当聚合时间戳，否则生成的图表可能出现不必要的间断，无法做到趋势变化的展示。在以上示例中，采集周期为一天一次，因此在time_bucket聚合时

选择以一天周期聚合，生成连续折线图。

```
time_bucket(timestamp, '1 day')
```

3.下钻示例

在完成报表制作后，用户可以选中一张图表，点击「下钻」开启下钻功能；「下钻」是指在点击一张图表的某一部分时，可以打开一个新的图表或超链接，进而查看与图表此部分相关的详细信息。 这里我们用之前生成的上海市平均风速风向的折线图作为示例来简单说明如何下钻。

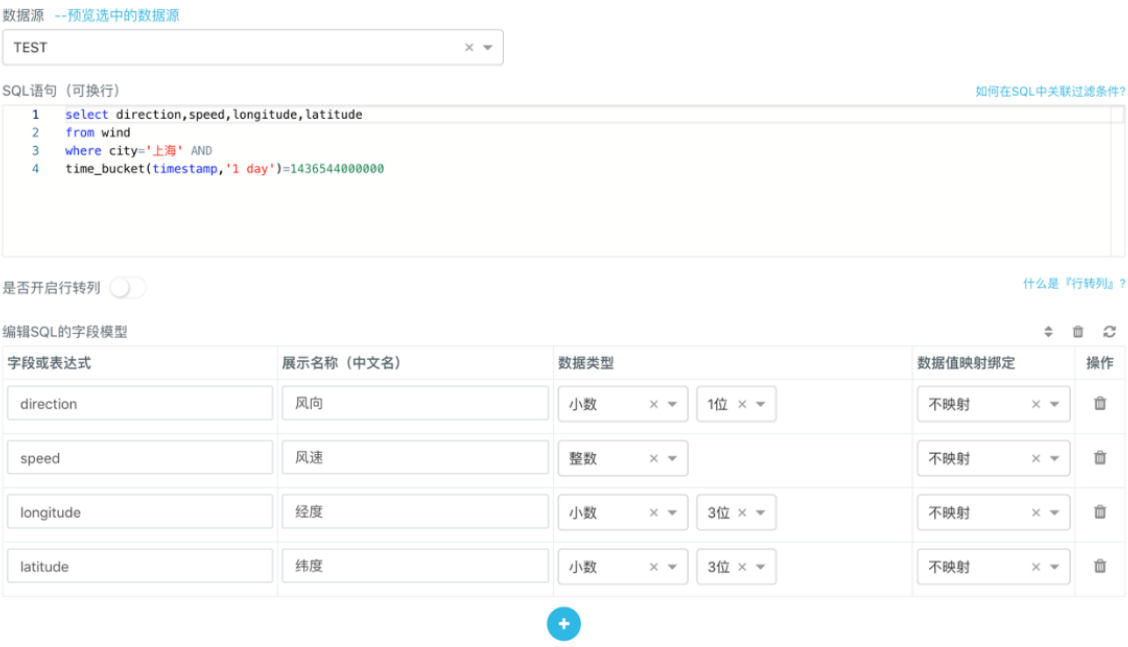
- 点击想要进行下钻的折线图，在右侧「控制面板」> 「下钻」中点击选项卡开启下钻并设置触发的下钻图表类型统一为表格。

> 注：折线图只能触发单一类型下钻，个别类型图表可触发多种类下钻。

- 点击折线图里的任意一点可以触发下钻并打开下钻数据展示框。我们下钻得到的数据会展现在左侧，右侧则是用户可以编辑设置的「控制面板」。

- 通过「下钻」，虽然当前折线图仅展示了上海市的平均风速风向，我们可以选中图中的任意数据点查看该点对应的timestamp里上海市每个监测点分别采集到的风速风向信息。 举个例子，我们在折线图上选中timestamp=1436544000000的任意数据点，在弹出的下钻「控制面板」>「数据」>「SQL建模」里新建一个SQL模型。在模型编辑框中输入以下SQL查询语句并设置好字段模型。

```
select direction, speed, longitude, latitude
from wind
where city='上海' AND time_bucket(timestamp, '1 day') = 1436544000000
```



因为在示例折线图中，timestamp按照time_bucket函数作过聚合，所以在下钻建模时，按照timestamp过滤的条件需要改为按照time_bucket函数过滤，否则无法查询到数据。

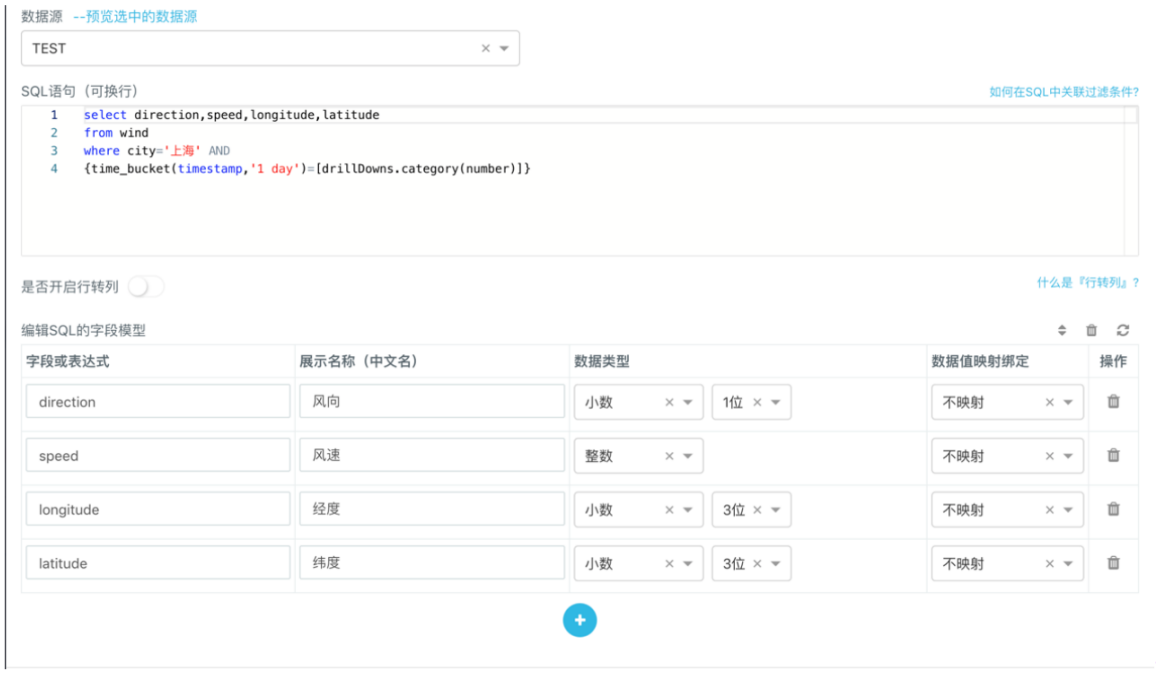
```
time_bucket(timestamp, '1 day') = 1436544000000
```

- 刷新图表后，可以在左侧看到在timestamp=1436544000000的那天，上海市一共5个监测点分别收集到的风速风向信息：

- 更方便的是，我们还可以使用下钻参数来编写用来下钻的SQL模型。 举个例子，我们可以用drillDowns里面的category变量替换之前恒定的timestamp。这样修改后，只要选中折线图上任意一点，展示下钻数据的图表都会直接根据选中点的timestamp自动刷新，生成在选中点对应timestamp里，上海市所有监测点分别采集的风速风向信息，无需手动修改SQL语句。

```
select direction, speed, longitude, latitude
from wind
where city='上海' AND
{time_bucket(timestamp, '1 day')=[drillDowns.category(number)]}
```

```
"drillDowns": [
    {
        "item": {
            "name": "平均风速",
            "value": 31.6,
            "category": "1436544000000",
            "type": "line",
            "_sugar_dd_default_": "1436544000000"
        },
        "fireKey": ""
    }
]
```

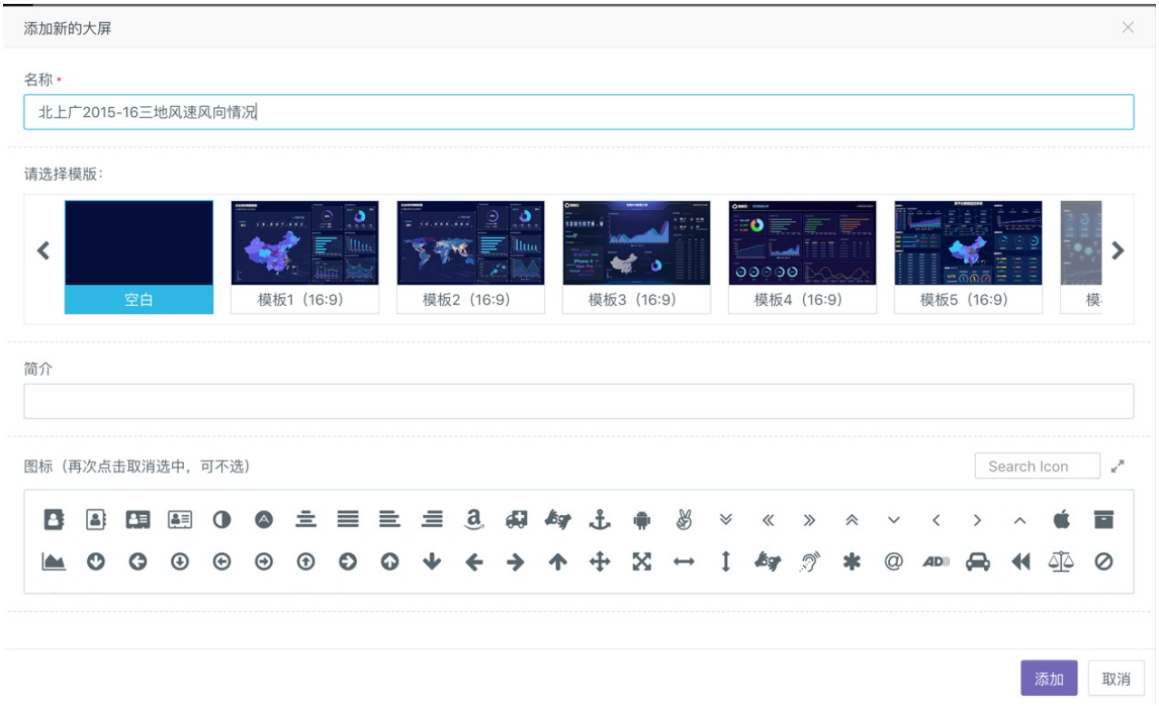


比如此时任意选择一个`timestamp=1420041600000`的数据点，点击触发下钻，弹出的数据展示框已经展示了在`timestamp=1420041600000`的那天，上海市一共5个监测点分别收集到的风速风向信息，无需我们手动更新SQL模型里`timestamp`的值。

4. 大屏展示示例

下面简单介绍如何对报表进行大屏展示，详细的教程可参见大屏制作。

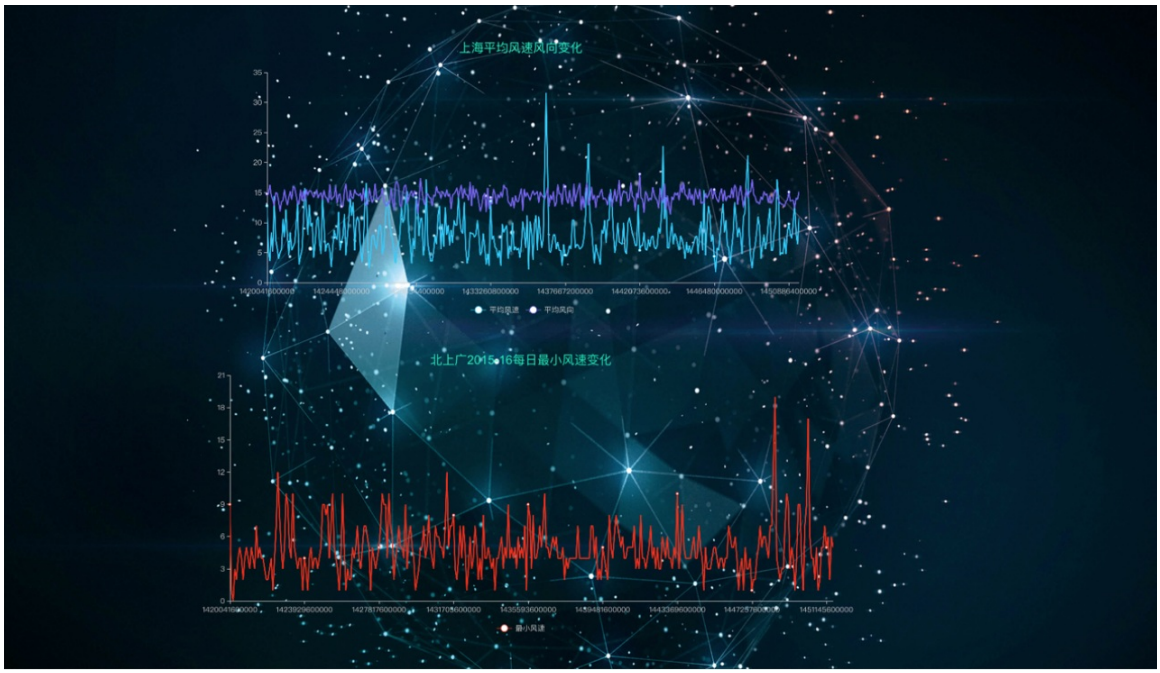
- 在Sugar「空间广场」的左侧管理中心点击进入「系统设置」>「大屏管理」，然后点击「添加新的大屏」，输入自定义信息并选择喜欢的模板来新建大屏。



- 新建完成后，可以在大屏列表中看到刚刚新建的大屏。点击右侧的眼睛图标进入大屏浏览页面，然后再点击右上角的编辑按钮即可以进入编辑模式。

- 进入大屏编辑页面后，新建折线图，利用已经建立的SQL模型（参见上方报表示例）为图表绑定数据，生成图表后利用工具栏中文字选项为大屏上的报表添加合适标题，示例如下：

- 现在可以对整个界面做一些基础美化，比如为整个大屏选择合适的背景，对新建的两个折线图进行排版，按照水平线对齐，经整理后示例效果如下：



时空服务

TSDB目前已支持时空服务，帮助用户更高效的处理与空间地理位置相关的数据。

使用说明

TSDB目前的SQL查询接口已支持多种与空间地理位置相关的函数，包括二维计算和球面计算。借助这类函数的强大能力，用户可以使用SQL方便地对空间地理位置相关的数据进行计算、分析，挖掘数据价值。

场景一：判断某个设备编号为ABC123的设备是否在某块特定区域内

下图表示TSDB存储的设备位置数据：每个设备的位置信息分别用x_value和y_value两个域（field）来存储，x_value和y_value表示设备在二维坐标系中的坐标；同时用licenseID（Tag）来代表设备号。 可以将这些数据看成一个二维表（如下图），针对此二维表可以通过构造空间函数来使用时空服务。

Metric: DeviceLocation			
timestamp	Field: x_value	Field: y_value	LicenseID
1523973600000	100	100	ABC123
1523973660000	100	300	ABC123
1523973720000	300	300	ABC123
1523973780000	500	300	ABC123
1523973840000	500	500	ABC123
1523973600000	150	200	BCD456
1523973660000	300	350	BCD456
1523973720000	450	500	BCD456
1523973780000	600	650	BCD456
1523973840000	750	800	BCD456

举例： 判断设备ID为ABC123的设备是否在某个特定区域（不包含边界）；假定这块区域是由(200, 200), (400, 200), (400,

400),(200, 400)四个点围绕组成的四边形。

SQL语句：

```
select * from DeviceLocation where ST_Contains(ST_GeometryFromText('POLYGON ((200 200,400 200,400 400,200 400,200 200))'), ST_Point(x_value,y_value)) and LicenseID = 'ABC123'
```

返回得到以下数据，证明设备ID为ABC123的设备在时间为1523973720000时在以上指定区域内。

Metric: DeviceLocation			
timestamp	Field: x_value	Field: y_value	LicenseID
1523973720000	300	300	ABC123

场景二：计算某个车辆在移动的轨迹中，是否经过一块以某个点为中心，一定半径以内的区域

TSDB的时空服务同时支持球面距离的计算，车辆位置和轨迹通常都是通过GPS信息来定位，所以真实的轨迹需要通过球面计算来计算。参考上例，我们可以将上述示例中的x_value和y_value用真实的经度（longitude）和纬度（latitude）来代替表示车辆的实际位置：

Metric: VehicleLocation			
timestamp	Field: longitude	Field: latitude	LicenseID
1523973600000	40.16667	80.16667	京 AFR673
1523973660000	40.16944	80.16917	京 AFR673
1523973720000	40.17083	80.17361	京 AFR673
1523973780000	40.17361	80.17778	京 AFR673
1523973840000	40.17500	80.17917	京 AFR673
1523973600000	78.143367	56.275782	沪 B6YT46
1523973660000	78.143489	56.277403	沪 B6YT46
1523973720000	78.143523	56.277564	沪 B6YT46
1523973780000	78.143597	56.277678	沪 B6YT46
1523973840000	78.143645	56.277789	沪 B6YT46

举例： 计算某个车辆在移动的轨迹中，是否经过一块以点（40.17222，80.175）为中心，半径为200m以内的区域。 SQL语句：

```
select * from VehicleLocation where LicenseID='京AFR673'and ST_Distance(to_spherical_geography(ST_Point(longitude, latitude)),to_spherical_geography(ST_Point(40.17222, 80.175))) <= 200
```

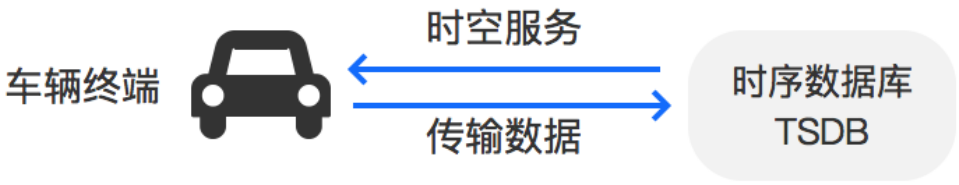
返回得到以下数据，证明车牌为京AFR673的车在会时间点为1523973720000时经过以上特定区域。

Metric: VehicleLocation			
timestamp	Field: longitude	Field: latitude	LicenseID
1523973720000	40.17083	80.17361	京 AFR673

实际应用

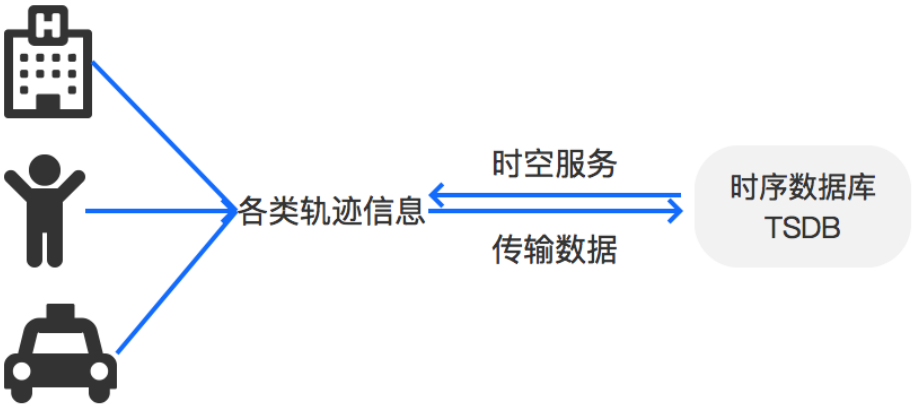
车辆监测

时序数据库用以存储时间、空间数据，借助时空服务，可以实时监测车辆当前行驶路线，在异常情况下，比如车辆脱离既定路线后，能够及时报警；有需要时还能够查看车辆历史行驶轨迹。在检修车作业场景下，判断检修车是否在指定区域内作业；检修车辆的行驶是否偏离预定轨迹；离某个站点最近的救援车是哪一辆等，从而实现车辆的全局优化调控。



轨迹分析

时序数据库用以存储时间、空间数据，借助时空服务，可以计算分析各类轨迹之间的关系，例如分析归纳人流最易聚集的热点区域，特定道路在哪些时间段车辆最多会造成堵塞，在特定区域是否有异常人员、车辆进入。轨迹分析可以应用在非常广泛多样的业务场景中，诸如社区安全、按需选址（如写字楼、医院、商场）、交通分流。



时空服务函数参考

关于SQL语句的使用参考，请参见支持SQL查询。

时空服务构造函数

- **ST_Point**

声明

```
Geometry ST_Point (double X, double Y)
```

作用

根据指定的坐标值（X, Y）返回一个点（Point）类型的几何（Geometry）对象。

示例

```
ST_Point(1.0，2.0)
```

返回一个点（Point）类型的几何（Geometry）对象，其横纵坐标分别为1.0和2.0。

- **ST_LineFromText**

声明

```
Geometry ST_LineFromText (varchar WKT)
```

作用

从Well-Known Text (WKT) 的字符串表达式中返回一个线段（LineString）类型的几何（Geometry）对象。

示例

```
ST_LineFromText('LINESTRING(1.0 1.5，1.0 0.5)')
```

返回一个起点为（1.0，1.5），终点为（1.0，0.5）的线段（LineString）类型的几何（Geometry）对象，

- **ST_Polygon**

声明 `Geometry ST_Polygon (varchar WKT)` **作用**

从Well-Known Text (WKT) 的字符串表达式中返回一个多边形（Polygon）类型的几何（Geometry）对象。

示例

```
ST_Polygon('POLYGON ((1 1，1 4，4 4，4 1))')
```

返回一个由（1，1）、（1，4）、（4，4）、（4，1）四个点组成的多边形（Polygon）类型的几何（Geometry）对象。

- **ST_GeometryFromText**

声明

```
Geometry ST_GeometryFromText (varchar WKT)
```

作用

从Well-Known Text (WKT) 的字符串表达式中返回一个几何（Geometry）对象。

示例

```
ST_GeometryFromText('POLYGON ((0 0，1 0，1 1，0 1，0 0))')
```

返回一个由（0，0）、（0，1）、（1，1）、(1，0)四个点组成的多边形（Polygon）类型的几何（Geometry）对象。

- **to_spherical_geography**

声明

```
SphericalGeography to_spherical_geography (Geometry geometry)
```

作用

把一个几何（Geometry）对象转换为一个球面几何（SphericalGeography）对象。

示例

```
to_spherical_geography(ST_Point(40.172, 80.175))
```

返回一个球面几何对象，标识地球上位于东经40.172度，北纬80.175度的点。【-180，0】为西经，【0，180】为东经；【-90，0】为南纬，【0，90】为北纬。

时空服务关系函数

- **ST_Contains**

声明

```
boolean ST_Contains (Geometry A, Geometry B)
```

作用

如果不存在Geometry B中的点落在Geometry A外，且Geometry B内部至少有一个点落在Geometry A内，，返回`true`，否则返回`false`。

示例

```
ST_Contains(ST_Polygon('POLYGON ((0 0,1 0,1 1,0 1))'), ST_Point(0.5，0.5))=true
```

点（0.5，0.5）完全落在由（0，0）、（0，1）、（1，1）、(1，0)四个点组成的多边形内，所以返回`true`。

- **ST_Equals**

声明

```
boolean ST_Equals(Geometry A, Geometry B)
```

作用

如果几何（Geometry）对象A和几何（Geometry）对象B代表的是同一个对象，返回`true`，否则返回`false`。

示例

```
ST_Equals(ST_GeometryFromText('POLYGON ((1 1, 1 3, 3 3, 3 1))'), ST_GeometryFromText('POLYGON ((3 3，3 1，1 1，1 3))')=true
```

由于上述两个多边形完全重合，所以返回`true`。

- **ST_Intersects**

声明

```
boolean ST_Intersects (Geometry A, Geometry B)
```

作用

如果几何（Geometry）对象A和几何（Geometry）对象B在二维空间上相交，返回`true`，否则返回`false`。

示例

```
ST_Intersects(ST_GeometryFromText('POLYGON((1 1, 1 3, 3 3, 3 1))'), ST_GeometryFromText('POLYGON ((4 4, 4 5, 5 5, 5 4))'))=false
```

由于这两个矩形没有相交，所以返回`false`。

- **ST_Overlaps**

声明

```
boolean ST_Overlaps (Geometry A, Geometry B)
```

作用

如果几何（Geometry）对象A和几何（Geometry）对象B有部分重叠，且其中一个并不完全包含另一个，返回`true`，否则返回`false`。

示例

```
ST_Overlaps(ST_GeometryFromText('POLYGON ((1 1, 1 4, 4 4, 4 1))'), ST_GeometryFromText('LINESTRING (1 1, 4 4)'))=false
```

由于起点为（1，1），终点为（4，4）的线段被完全包含于由（1，1）、（1，4）、（4，4）、(4，1)四个点组成的多边形内，所以即使两者有相交，`ST_Overlaps`仍返回`false`。

时空服务访问函数

- **ST_Area**

声明

```
double ST_Area (Geometry A)
```

作用

返回多边形（Polygon）类型的几何（Geometry）对象A二维空间下的面积。

示例

```
ST_Area(ST_GeometryFromText('POLYGON ((2 2, 2 6, 6 6, 6 2))'))=16.0
```

返回由（2，2）、（2，6）、（6，6）、（6，2）四个点组成的多边形的面积为16.0。

- **ST_Area**

声明

```
double ST_Area (SphericalGeometry A)
```

作用

返回多边形（Polygon）类型的球面几何（SphericalGeometry）对象A在地球球面上的面积。

示例

```
ST_Area(to_spherical_geography(ST_Polygon('POLYGON ((0 0,0 90,90 0))')))=6.375825913974858E13
```

返回由地球球面上的三个点（0，0）、（0，90）、（90，0）组成的多边形的球面面积为6.375825913974858E13平方米。这里地球半径=6371.01千米。

- ST_Distance

声明

```
double ST_Distance (Geometry A, Geometry B)
```

作用

返回几何（Geometry）对象A和几何（Geometry）对象B在二维空间下的笛卡尔最短距离。

示例

```
ST_Distance(ST_Point(50, 100), ST_Point(150, 150))= 111.80339887498948
```

返回上述两个点（50，100）和（150，150）之间的笛卡尔最短距离为111.80339887498948。

- ST_Distance

声明

```
double ST_Distance (SphericalGeometry A, SphericalGeometry B)
```

作用

返回球面几何（SphericalGeometry）对象A和球面几何（SphericalGeometry）对象B在地球球面上的最短距离，单位米。

示例

```
ST_Distance(to_spherical_geography(ST_Point(40.175, 80.175)), to_spherical_geography(ST_Point(40.5, 80.5)))= 36643.77019025462
```

返回地球上两个点(40.175, 80.175)和(40.5, 80.5)之间的球面距离为36643.77019025462米。这里地球半径=6371.01千米。

- ST_Length

声明

```
double ST_Length (LineString A)
```

作用

返回线段（LineString）类型的几何（Geometry）对象A在二维空间下的长度。

示例

```
ST_Length(ST_GeometryFromText('LINESTRING (0 0, 2 2)'))= 2.8284271247461903
```

返回起点为（0，0），终点为（2，2）的线段的长度为2.8284271247461903。

- ST_X

声明

```
double ST_X (Point A)
```

作用

返回点（Point）类型的几何（Geometry）对象A的横（X轴）坐标。

示例

```
ST_X(ST_GeometryFromText('POINT (1 2)'))=1.0
```

返回点（1，2）的横（X轴）坐标为1.0.

- **ST_Y**

声明

```
double ST_Y (Point A)
```

作用

返回点（Point）类型的几何（Geometry）对象A的纵（Y轴）坐标。

示例

```
ST_Y(ST_GeometryFromText('POINT (1 2)'))=2.0
```

返回点（1，2）的纵（Y轴）坐标为2.0.

SQL参考

支持SQL查询

介绍

TSDB支持SQL的查询接口，可以通过SQL语言实现对TSDB数据的筛选和查询。利用SQL的强大能力，用户即可以熟悉方便地对数据进行操作，也可以充分利用SQL函数的计算能力，挖掘数据价值。

基础操作说明

- SQL语句的table对应TSDB的metric
- SQL语句中的field对应TSDB的timestamp、field、tag
- SQL的查询条件中如果包含中文字符，应当使用单引号括起来，如果只包含英文字符，则单引号和双引号均可

举个例子：一个智能电表监控的物联网集成方案，采集了智能电表的各个监控点的数据。在TSDB中这样组织（如下图），metric为SmartMeter，表示TSDB存的是智能电表的数据，每个电表有power和MeterCurrent两个域（field），用两个tag，即MeterID和City，来代表每个数据点来自哪个电表ID和城市。电表每5s上传一次功率值和电流值。

Metric: SmartMeter					
Field:power			Field:MeterCurrent		
timestamp	value	tag	timestamp	value	tag
1523973670011	426	MeterID = 2345HGYE City = "深圳"	1523973670011	4.5	MeterID = 2345HGYE City = "深圳"
1523973675011	560	MeterID = 2345HGYE City = "深圳"	1523973675011	2.3	MeterID = 2345HGYE City = "深圳"
1523973680011	378	MeterID = 2345HGYE City = "深圳"	1523973680011	2.5	MeterID = 2345HGYE City = "深圳"
1523973685011	467	MeterID = 2345HGYE City = "深圳"	1523973685011	4.5	MeterID = 2345HGYE City = "深圳"
1523973690011	389	MeterID = 2345HGYE City = "深圳"	1523973690011	5.5	MeterID = 2345HGYE City = "深圳"
1523973695011	367	MeterID = 2345HGYE City = "深圳"	1523973695011	5.0	MeterID = 2345HGYE City = "深圳"
1523973700011	350	MeterID = 2345HGYE City = "深圳"	1523973700011	5.2	MeterID = 2345HGYE City = "深圳"
1523973705011	256	MeterID = 4536UJYH City = "上海"	1523973705011	5.3	MeterID = 4536UJYH City = "上海"
1523973710011	289	MeterID = 4536UJYH City = "上海"	1523973710011	5.3	MeterID = 4536UJYH City = "上海"

可以将上表看成一个二维表，针对二维表来写SQL语句

Metric: SmartMeter				
timestamp	Field:power	Field:MeterCurrent	MeterID	City
1523973670011	426	4.5	2345HGYE	深圳
1523973675011	560	2.3	2345HGYE	深圳
1523973680011	378	2.5	2345HGYE	深圳
1523973685011	467	4.5	2345HGYE	深圳
1523973690011	389	5.5	2345HGYE	深圳
1523973695011	367	5.0	2345HGYE	深圳
1523973700011	350	5.2	2345HGYE	深圳
1523973705011	256	5.3	4536UJYH	深圳
1523973710011	289	5.3	4536UJYH	深圳

应用场景一：要过滤功率值大于400、电流值小于5，电表ID为2345HDYE的数据

```
select timestamp, power, MeterCurrent, MeterID, City  from SmartMeter where power > 400 and current<5 and MeterID = '2345HDYE'
```

Metric: SmartMeter				
timestamp	power	MeterCurrent	MeterID	City
1523973670011	426	4.5	2345HDYE	深圳
1523973675011	560	2.3	2345HDYE	深圳
1523973685011	467	4.5	2345HDYE	深圳

应用场景二：电表ID为2345HDYE的电表中，返回每10秒的功率平均值

```
select time_bucket(timestamp,  '10 seconds') as TIME, avg(power) as AVG_POWER from SmartMeter  group by time_bucket(timestamp, '10 seconds') order by time_bucket(timestamp, '10 seconds')
```

Metric: SmartMeter	
timestamp	AVG_POWER
1523973670011	492
1523973680011	422.5
1523973690011	378
1523973700011	303
1523973710011	289

应用场景三：join两个metric

两个metric如下表

Metric: SmartMeter				
timestamp	Field:power	Field:MeterCurrent	MeterID	City
1523973670011	426	4.5	2345HGYE	深圳
1523973675011	560	2.3	2345HGYE	深圳
1523973680011	378	2.5	2345HGYE	深圳
1523973685011	467	4.5	6763HUDY	深圳
1523973690011	389	5.5	6763HUDY	深圳

Metric: SmartTemperature			
timestamp	Field:temp	MeterID	City
1523973670011	23	2345HGYE	深圳
1523973675011	25	2345HGYE	深圳
1523973680011	23	2345HGYE	深圳
1523973685011	22	2345HGYE	深圳

用MeterID将两个表进行join，SQL语句如下： `select * from SmartMeter join SmartTemperature on SmartMeter.MeterID = SmartTemperature.MeterID ;`

得到以下数据：

timestamp	Field:temp	Field:power	Field:MeterCurrent	MeterID	City
1523973670011	23	426	4.5	2345HGYE	深圳
1523973675011	25	560	2.3	2345HGYE	深圳
1523973680011	23	378	2.5	2345HGYE	深圳

TSDB_SQL_01-1

SQL函数

- 查询函数举例

类型	SQL	解释
时间查询	select timestamp from metric	timestamp: 固定查询时间戳关键字，系统默认使用UTC时间 metric：用户需要查询的metric名称
单域查询	select value from metric	value：用户需要查询的field的名称 metric：用户需要查询的metric名称
Tag查询	select tag_key from metric	tag_key：用户需要查询的tag的key metric：用户需要查询的metric名称
多列查询	select timestamp,field1 from metric select * from metric	tag_key：用户需要查询的tag的key field1：用户需要查询的field的名称 metric：用户需要查询的metric名称
按照时间排序	select timestamp, value from metric order by timestamp select timestamp, value from metric order by timestamp desc select timestamp, value from metric order by timestamp offset 1 limit 1	order by：查询排序方式 asc/desc：升序/降序 offset和limit：排序后从第offset条开始，取limit条
带过滤查询	select timestamp, value from metric where value > 30 and timestamp >150937263000	value > 30：过滤value的值，UTC时间戳精度毫秒
分组查询	select tag_key, count(1) from metric group by tag_key	group by tag_key：按照tag的Key值分组
聚合查询	select time_bucket(timestamp, '2 days') as DAY, sum(value) as SUM from metric group by time_bucket(timestamp, '2 days') order by time_bucket(timestamp, '2 days')	time_bucket: 时间聚合函数 timestamp：实际数据点中的时间戳字段（不用修改） 2 days：时间窗口是2天 group by time_bucket：按照时间2 days分组 sum(value)：聚合2 days的value的值 time_bucket支持日历对齐和自定义起始时间： 如time_bucket(timestamp, 1dc)，字符'c'表示日历对齐。 时间单位请参考文档时间单位 窗口的默认起始时间是2000-01-01 00:00:00，如需自定义起始时间可以将'2 days'改成'2 days,2019-01-01 00:00:00'，即表示从2019-01-01开始 每2天一个时间窗口。 日历对齐规则示例如下： 如time_bucket(timestamp,'1 dc,2019-01-01 12:00:00') 则第一个窗口是01月01日12:00:00.000 - 01月01日23:59:59.99 第二个窗口是01月02日00:00:00.000 - 01月02日23:59:59.99 以此类推
自定义函数计算查询	select timestamp, ((field2 - field1) * 10) as RESULT, tag_key from metric	(field2 - field1) * 10 ：用户自定义函数计算

- SQL查询常用的时间函数。

分类	支持函数	举例	说明
日期时间操作符	+/-	mysql> select (date '2012-08-08' + interval '2' day) 2012-08-10 mysql> select (timestamp '2012-08-08 01:00' + interval '29' hour) 2012-08-09 06:00:00.000 mysql> select (date '2012-08-08' - interval '2' day) 2012-08-06 mysql> select (timestamp '2012-10-31 01:00' - interval '1' month) 2012-09-30 01:00:00.000	操作符
时区转换	AT TIME ZONE	mysql> SELECT timestamp '2012-10-31 01:00 UTC' AT TIME ZONE 'America/Los_Angeles' 2012-10-30 18:00:00.000 America/Los_Angeles mysql> SELECT timestamp '2012-10-31 01:00 UTC' AT TIME ZONE 'Asia/Shanghai' 2012-10-31 09:00:00.000 Asia/Shanghai	UTC时区转换
日期和时间函数	current_date current_timestamp from_unixtime(unix_timestamp) now() to_unixtime(timestamp) cast(to_unixtime(timestamp) as bigint)	mysql> select current_date 2021-04-21 mysql> select current_timestamp 2021-04-21 10:59:56.378 UTC mysql> select from_unixtime(1619031748); 2021-04-21 19:02:28.000 mysql> select now() 2021-04-21 11:01:38.221 UTC mysql> select to_unixtime(timestamp '2021-04-21 19:02:28') 1.619031748E9 mysql> select cast(to_unixtime(timestamp '2021-04-21 19:02:28') as bigint) 1619031748	日期时间相关
截取函数	date_trunc(unit,timestamp)	mysql> select date_trunc('day',timestamp '2021-04-21 19:02:28') 2021-04-21 00:00:00.000 mysql> select date_trunc('hour',timestamp '2021-04-21 19:02:28') 2021-04-21 19:00:00.000	截取timestamp的一部分
时间段	date_add(unit,bigint,timestamp)	mysql> select date_add('hour', 8, timestamp '2021-04-21 19:02:28') 2021-04-22 03:02:28.000	基于某个时间加减
持续时长	parse_duration(string)	mysql> SELECT parse_duration('5m') 0 00:05:00.000	字符串格式的时间转化
日期格式化	date_format(timestamp,string format)	mysql> select date_format(timestamp '2021-04-21 19:02', '%Y-%m-%d %H:%i:%s') 2021-04-21 19:02:00	格式转化
提取函数	day_of_week(timestamp)	mysql> select day_of_week(timestamp '2021-04-21 19:02') 3	返回bigint

- 自定义函数

TSDB支持标准ANSI SQL语义，选取常用的部分自定义函数如下所示。

自定义函数说明：

分类	支持函数	举例	说明
条件表达式	CASE	SELECT field1, CASE field1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'many' END FROM metric	case表达式
条件表达式	IF	SELECT field1, IF (field1>100,1,0) as result FROM metric	if表达式，if(condition, true_value)或if(condition, true_value, false_value)
条件表达式	COALESCE	SELECT field1, field2, COALESCE (field1, field2) as result FROM metric	返回列表中第一个非空值
计算函数	abs(x)	SELECT field1, abs (field1) as result FROM metric	返回x的绝对值
计算函数	sqrt(x)	SELECT field1, sqrt (field1) as result FROM metric	返回x的平方根
计算函数	cbrt(x)	SELECT field1, cbrt (field1) as result FROM metric	返回x的立方根
计算函数	ceil(x)	SELECT field1, ceil (field1) as result FROM metric	返回不小于x的最小整数值
计算函数	ceiling(x)	同ceil(x)	返回大于或等于x的最小整数值
计算函		SELECT field1, floor (field1) as result	返回小于或

数	floor(x)	FROM metric	等于x的最大整数值
字符串操作	||	SELECT field1 || field2 as result FROM metric	字符串级联
聚合函数	avg(x)	SELECT time_bucket(timestamp, '2 days') as DAY, avg(field1) as result FROM metric group by time_bucket(timestamp, '2 days') order by time_bucket(timestamp, '2 days')	返回field1的平均值
聚合函数	count(*)	SELECT count(*) as result FROM metric where timestamp < 1525611901	返回数量
聚合函数	count(x)	SELECT time_bucket(timestamp, '2 days') as DAY, count(field1) as count FROM metric group by time_bucket(timestamp, '2 days') order by time_bucket(timestamp, '2 days')	返回非空值的数量
聚合函数	max_by(x, y)	SELECT max_by(field1,field2) as result FROM metric where timestamp < 1525611901000	返回y最大时的x，UTC时间戳精度毫秒
聚合函数	min_by(x, y)	SELECT min_by(field1,field2) as result FROM metric where timestamp < 1525611901000	返回y最小时的x，UTC时间戳精度毫秒
聚合函数	max(x)	SELECT max(field1) as result FROM metric where timestamp < 1525611901000	返回x的最大值，UTC时间戳精度毫秒
聚合函数	min(x)	SELECT min(field1) as result FROM metric where timestamp < 1525611901000	返回x的最小值，UTC时间戳精度毫秒
聚合函数	sum(x)	SELECT time_bucket(timestamp, '2 days') as DAY, sum(field1) as sum FROM metric group by time_bucket(timestamp, '2 days') order by time_bucket(timestamp, '2 days')	返回x的和值
聚合函数	first_value(x)	SELECT field2, first_value(field1) OVER (PARTITION BY field2 ORDER BY timestamp) FROM metric WHERE timestamp >=0 AND timestamp <= 1600 AND field2 IN (1, 2) GROUP BY (field2, field1) **（建议field2对应到TSDB的tag）**	返回x的首值
聚合函数	last_value(x)	SELECT field2, last_value(field1) OVER (PARTITION BY field2 ORDER BY timestamp RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) FROM metric WHERE timestamp >=0 AND timestamp <= 1600 AND field2 IN (1, 2) GROUP BY (field2, field1) **（建议field2对应到TSDB的tag）**	返回x的末值

TSDB SDK的SQL使用

TSDB Java-SDK、Node-SDK、Python-SDK以及API均支持了SQL查询，具体使用请分别参考Java-SDK、Node-SDK、Python-SDK、API。

支持MySQL协议

介绍

TSDB现已正式支持Mysql协议，可为用户提供更加便捷的SQL使用体验。用户可通过控制台创建Mysql协议账号后，利用Mysql JDBC、以及支持Mysql JDBC的各类工具查询访问TSDB数据。

Mysql账号管理

创建账号

使用Mysql协议连接TSDB数据库实例时需要提供Mysql协议账号信息（包括用户名、密码），用户可通过TSDB控制台创建Mysql协议账号。

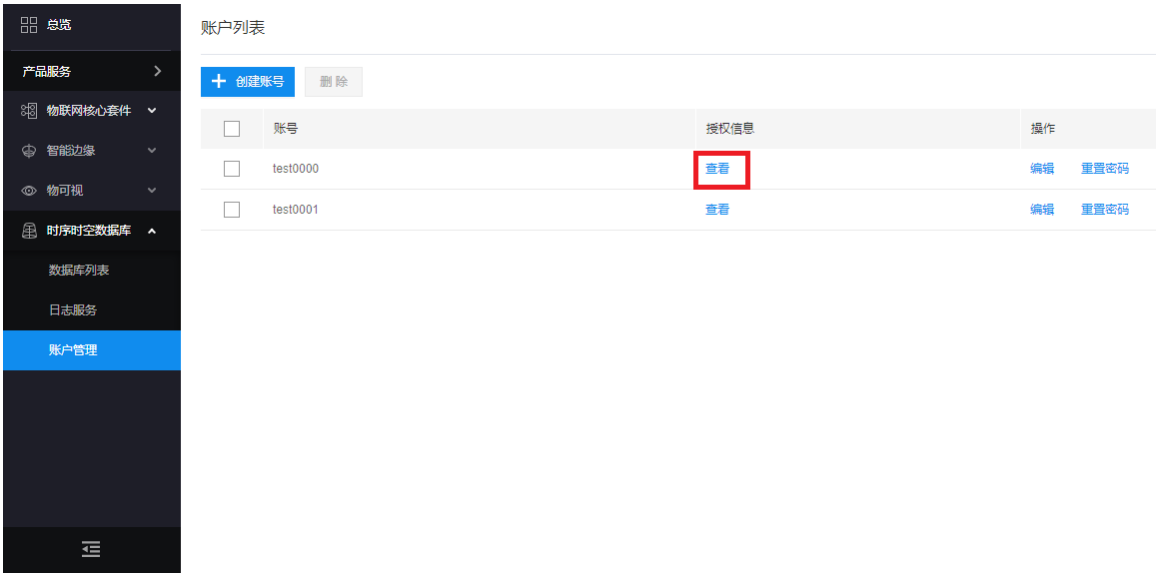
> 使用限制：
>
> - 仅允许主用户进行mysql账号管理相关操作
> - 子用户不能看到主用户创建的mysql账号
> - 每个主用户最多创建10个mysql协议账号

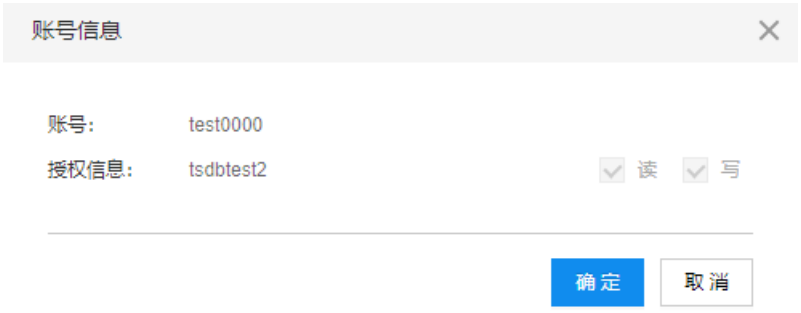
- 登录TSDB控制台，点击「账户管理」后显示已创建的账户列表信息

- 点击「创建账号」，在弹出的编辑窗口中，「账号名称」、「账号密码」、「确认密码」分别填写用户自定义的Mysql用户名及密码，「授权信息」选择希望通过该账号访问到的TSDB实例名称。

> 账号名称：8-16个字符，由字母和数字组成，创建后不可修改
>
> 账号密码：8-16个字符，必须包含大小写字母和数据三种字符
>
> 授权信息：支持选择多个TSDB实例进行关联

- 创建后的账号显示在账户列表中。账号关联的TSDB数据库实例可通过点击「授权信息」>「查看」获得。



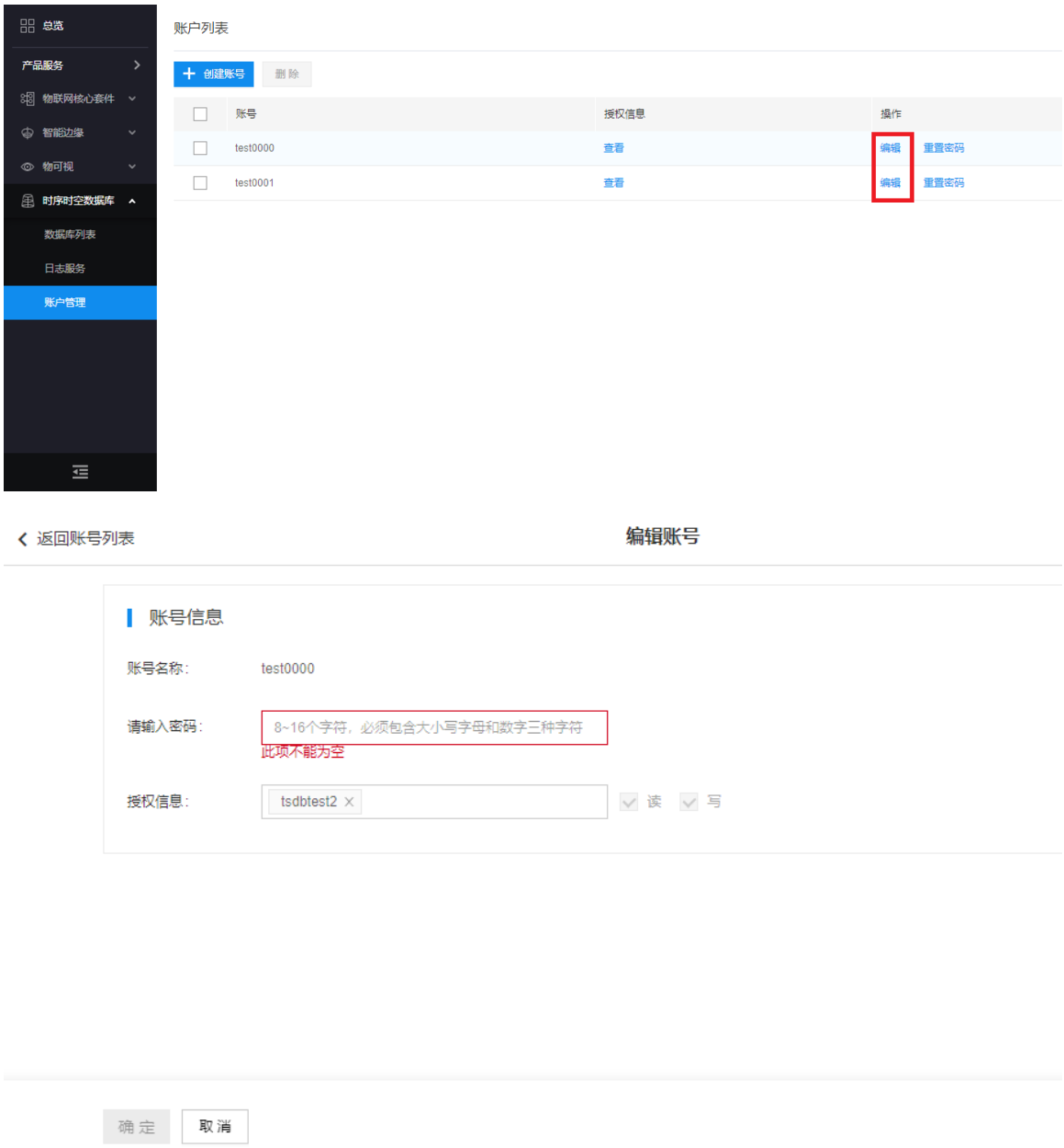


编辑账号

已创建的Mysql账号支持授权信息的修改、账号密码重置。

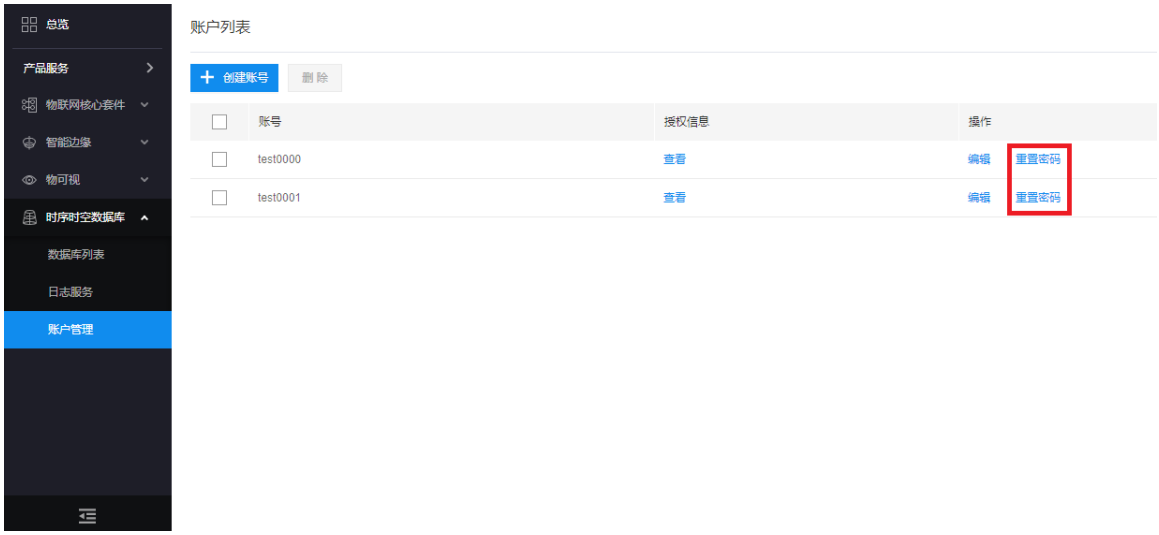
授权信息编辑

点击「账户列表」>「操作」>「编辑」进入编辑账号页面，可对授权信息进行添加和删除，调整完成后需保证输入的账号密码正确，点击确定后将保存本次调整内容。



账号密码重置

点击「账户列表」>「操作」>「重置密码」进入密码编辑弹窗，按照要求输入新密码，点击确定后需要通过发送手机验证码进行二次确认，确认手机号无误后点击「发送验证码」，将接收到的验证码正确输入，点击确定，完成账号密码的重置。



连接示例

账号信息生成后，用户可利用Mysql Shell、Mysql JDBC、以及各种兼容Mysql协议的BI工具访问TSDB，连接Host信息填写TSDB实例的EndPoint地址+端口号3306（示例见：Grafana连接TSDB）；通过Mysql协议支持的查询函数请参考SQL查询函数。

MySQL Shell

示例采用MySQL Shell 5.0版本连接TSDB，如下图：

```
[work@gzhxy-y32-sandbox042.gzhxy.baidu.com bin]$ ./mysql -h10.70.[redacted] -P8312 -ute[redacted] -pTE[redacted] -Dtsdbtest2
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 48353
Server version: 5.1.0

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show tables;
+---------------+
| Table         |
+---------------+
| humidity      |
| pm25          |
| precipitation |
| road          |
| temperature   |
| wind          |
+---------------+
6 rows in set (0.19 sec)

mysql> select * from pm25 limit 1 \G
*************************** 1. row ***************************
timestamp: 1420000000000
    value: 60.0
 latitude: 31.48877
longitude: 121.940402
     city: 上海
 zip_code: 200000
1 row in set (0.13 sec)
```

MySQL Connector/J

示例采用MySQL Connector/J 5.1.26版本连接TSDB。

- 在Maven的pom.xml文件中添加mysql-connector-java 5.1.26的依赖：

```
 <dependency>
   <groupId>mysql</groupId>
   <artifactId>mysql-connector-java</artifactId>
   <version>5.1.26</version>
</dependency>
```

- 使用Connector/J查询TSDB数据库实例中数据的示例代码：

```
 import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class MySQLDemo {

    // MySQL 8.0 以下版本 - JDBC 驱动名及数据库 URL
    private static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    private static final String DB_URL = "jdbc:mysql://10.70.**.**:8312/tsdbtest2";

    // 数据库的用户名与密码，需要根据自己的设置
    private static final String USER = "te******";
    private static final String PASS = "TE******";

    public static void main(String[] args) {
        try {
            // 注册 JDBC 驱动
            Class.forName(JDBC_DRIVER);
            String sql = "SELECT * FROM pm25 limit 1";
            try (
                // 打开链接
                Connection conn = DriverManager.getConnection(DB_URL,USER,PASS);
                // 实例化Statement对象
                Statement stmt = conn.createStatement();
                // 执行查询
                ResultSet rs = stmt.executeQuery(sql);
            ) {
                // 展开结果集数据库
                while(rs.next()){
                    // 通过字段检索
                    long timestamp  = rs.getLong("timestamp");
                    double value = rs.getDouble("value");
                    double latitude = rs.getDouble("latitude");
                    double longitude = rs.getDouble("longitude");
                    String city = rs.getString("city");
                    String zipCode = rs.getString("zip_code");

                    // 输出数据
                    System.out.println("输出结果：");
                    System.out.println("timestamp：" + timestamp);
                    System.out.println("value：" + value);
                    System.out.println("latitude：" + latitude);
                    System.out.println("longitude：" + longitude);
                    System.out.println("city：" + city);
                    System.out.println("zipCode：" + zipCode);
                }
            }
        } catch (Exception e) {
            // 处理异常
            e.printStackTrace();
        }
    }
}
```

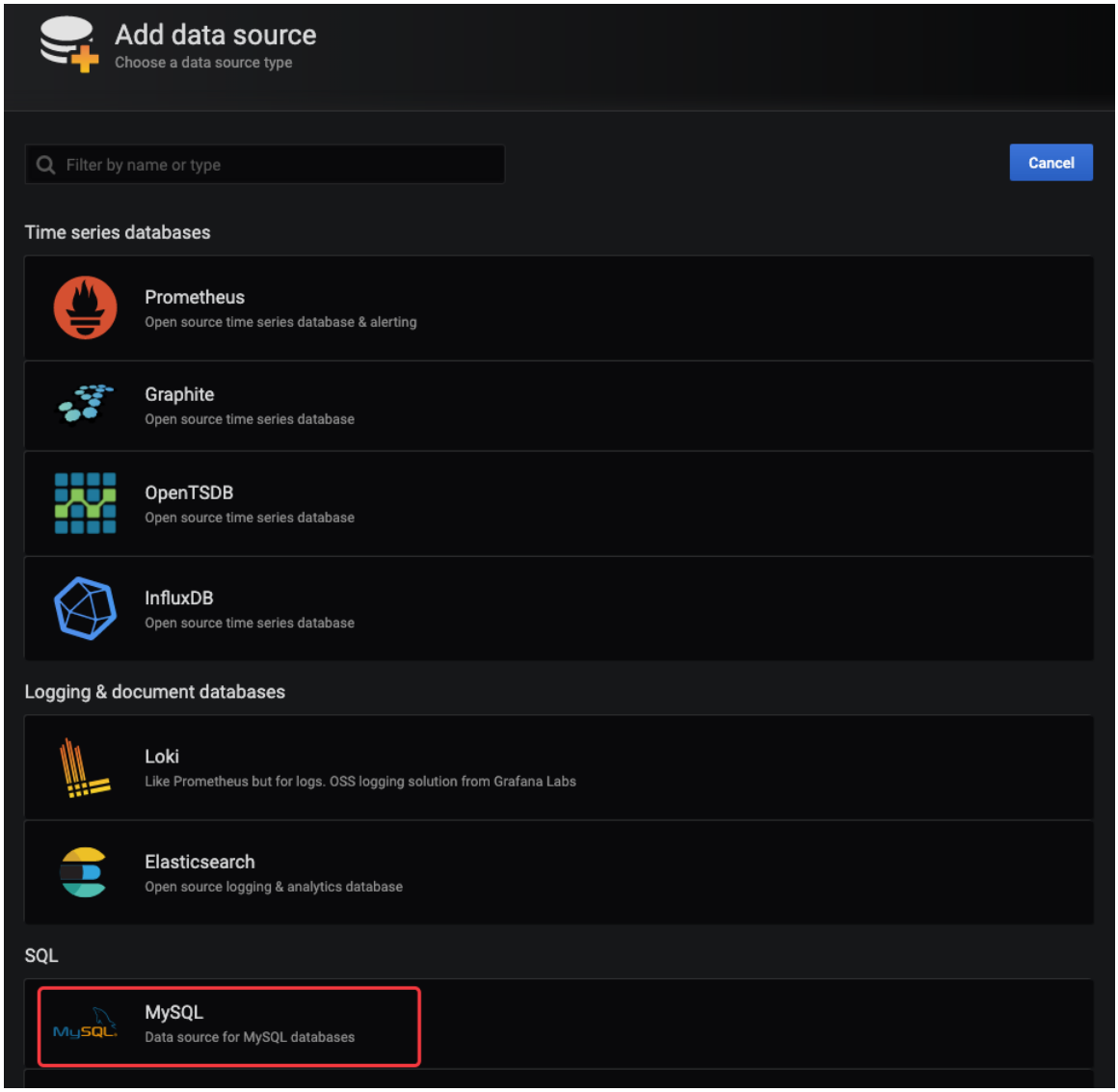
- 示例代码返回的查询结果：

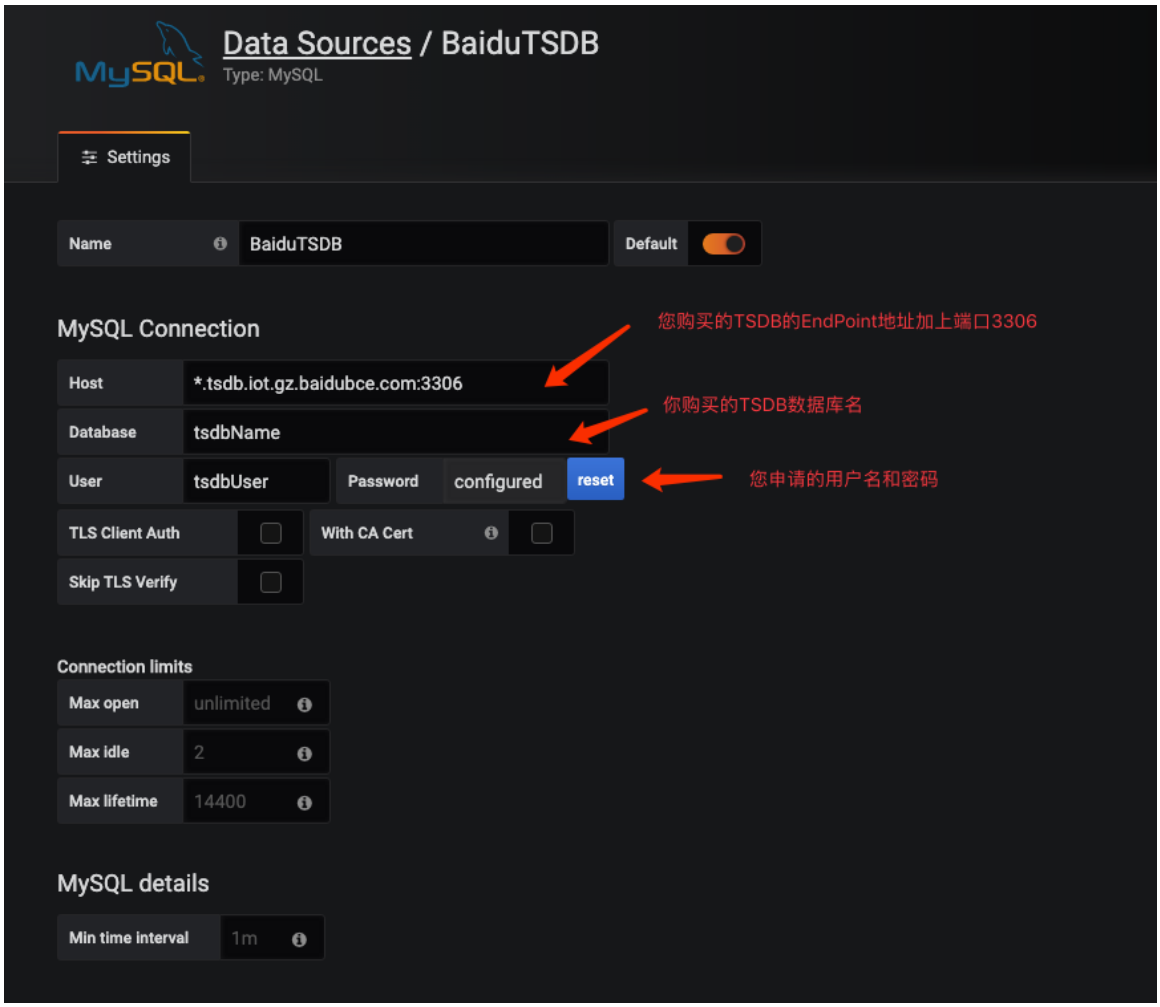
```
 timestamp : 1420000000000
value : 60.0
latitude : 31.48877
longitude : 121.940402
city : 上海
zipCode : 200000
```

Grafana

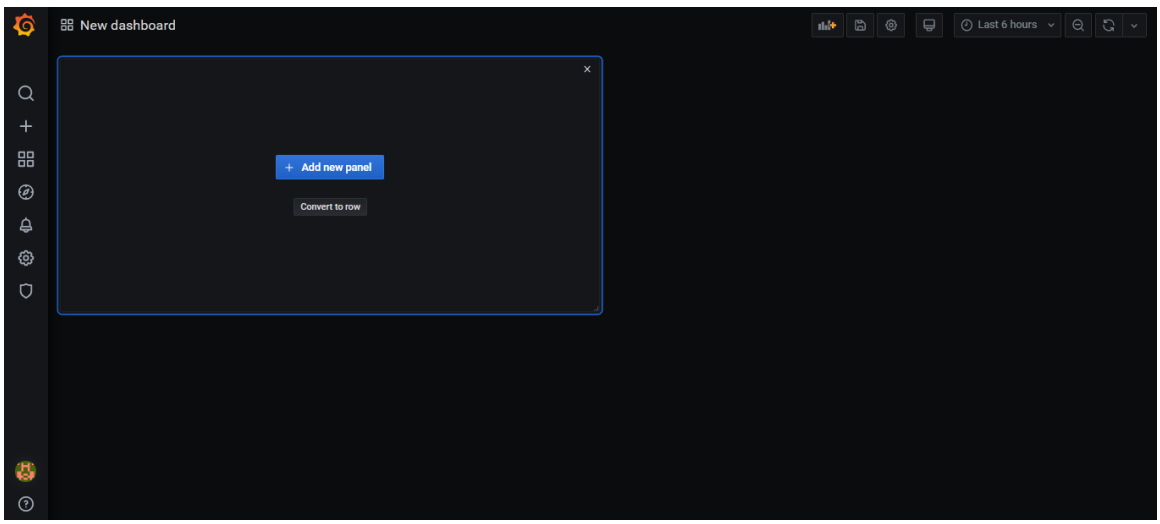
示例采用Grafana 7.1.5版本连接TSDB。

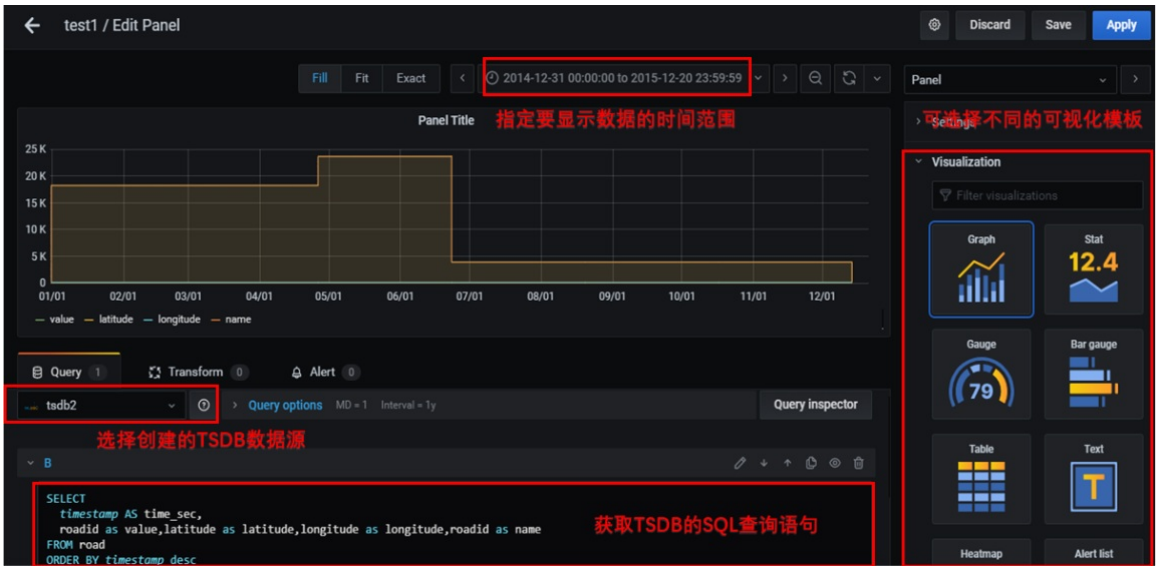
- 首先添加TSDB数据源。在添加数据源窗口种选择添加“Mysql”类型的数据源，并按照下图的指示完成数据库地址、数据库名以及账号密码信息的配置。当连接测试成功后，就可以像通过Grafana操作Mysql一样操作TSDB。

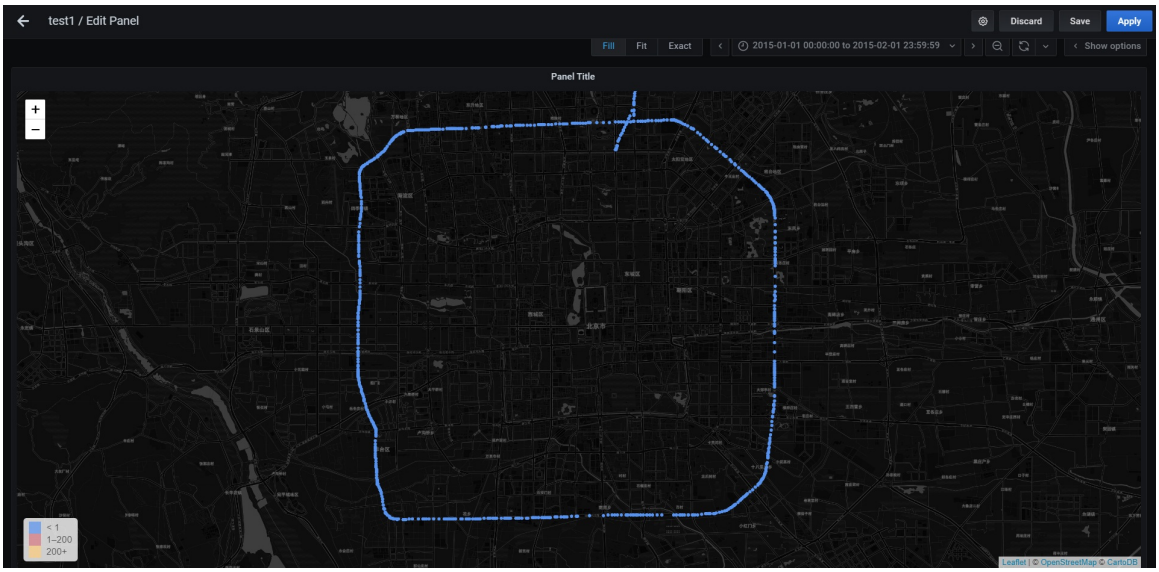




- 添加可视化面板，选择创建的TSDB数据源，输入数据的SQL查询语句后即可显示图表。下图以TSDB中数据为示范数据为例，展示SQL查询语句及对应的显示效果。







TSDB支持SQL的相关内容请参考文档。

对接hive-sql

Tsdb storage handler

TSDB对接hive是通过实现一个TSDB的HiveStorageHandler，支持对tsdb数据的读取。

Jar下载地址：https://sdk.bce.baidu.com/console-sdk/hive-tsdb-handler_all.jar

如果是本地hive集群，请下载jar到本地；如果使用bmr，则上传到bos或者直接使用地址bos://iot-tsdb/hive-tsdb-handler_all.jar

支持的hive 1.2.0，jdk 1.7。

在Hive CLI或Hue中使用

示例及参数说明如下:

```
 add jar /path/to/hive-tsdb-handler_all.jar;   /* 添加jar，如果在bmr中，请使用bos地址，如bos://path/to/hive-tsdb-handler_all.jar */
CREATE EXTERNAL TABLE `wind`(`time` bigint, `value` double, `city` string)    /* 创建表 */
STORED BY 'com.baidubce.tsdb.hive.storage.TsdbStorageHandler'                 /* 设置storage为TsdbStorageHandler */
TBLPROPERTIES (
"tsdb.metric_name" = "wind",                                                  /* Metric名字，默认表名则被当为metric */
"tsdb.timestamp_name" = "time",                                               /* Timestamp对应的列名，默认time为对应的列名 */
"tsdb.field_names" = "value",                                                 /* field列表，只需填写table中用到的，多个field通过逗号分割 */
"tsdb.tag_keys" = "city",                                                     /* tagKey列表，只需填写table中用到的，这个示例中可以去掉改行，但如果tagKey中包含大写字母，则必须填写 */
"tsdb.endpoint" = "ENDPOINT",                                                 /* TSDB实例的endpoint */
"tsdb.access_key" = "AK",                                                     /* AK */
"tsdb.secret_key" = "SK"                                                      /* SK */
);

select time, value from wind where time>=1451500000000 and time<=1451577600000;
```

在endpoint为IP:PORT的情形下：

```
 add jar /path/to/hive-tsdb-handler_all.jar;   /* 添加jar，如果在bmr中，请使用bos地址，如bos://path/to/hive-tsdb-handler_all.jar */
CREATE EXTERNAL TABLE `wind`(`time` bigint, `value` double, `city` string)
STORED BY 'com.baidubce.tsdb.hive.storage.TsdbStorageHandler'
TBLPROPERTIES (
"tsdb.metric_name" = "wind",
"tsdb.timestamp_name" = "time",
"tsdb.field_names" = "value",
"tsdb.tag_keys" = "city",
"tsdb.endpoint" = "http://IP:PORT",
"tsdb.host" = "DATABASE_NAME.tsdb.iot.gz.baidubce.com",
"tsdb.grpc_port" = "GRPC_PORT",
"tsdb.access_key" = "AK",
"tsdb.secret_key" = "SK"
);
select time, value from wind where time>=1451500000000 and time<=1451577600000;
```

场景示例

计算风速

风速数据由传感器定时上传到tsdb中，数据包含两个field分别为x和y，表示x轴和y轴方向的风速，如下由两个垂直方向的风速来计算出总的风速。

```
 add jar /path/to/hive-tsdb-handler_all.jar;   /* 添加jar，如果在bmr中，请使用bos地址，如bos://path/to/hive-tsdb-handler_all.jar */
CREATE EXTERNAL TABLE `WindSpeed`(`time` bigint, `x` double, `y` double)      /* 创建表 */
STORED BY 'com.baidubce.tsdb.hive.storage.TsdbStorageHandler'                 /* 设置storage为TsdbStorageHandler */
TBLPROPERTIES (
"tsdb.metric_name" = "WindSpeed",                                             /* Metric名字，默认表名则被当为metric */
"tsdb.timestamp_name" = "time",                                               /* Timestamp对应的列名，默认time为对应的列名 */
"tsdb.field_names" = "x,y",                                                   /* field列表，多个field通过逗号分割 */
"tsdb.endpoint" = "ENDPOINT",                                                 /* TSDB实例的endpoint */
"tsdb.access_key" = "AK",                                                     /* AK */
"tsdb.secret_key" = "SK"                                                      /* SK */
);

select time, sqrt(pow(x, 2) + pow(y, 2)) as speed from WindSpeed;
```

原始数据

metric:WindSpeed

time	field : x	field : y
1512086400000	3.0	4.0
1512086410000	1.0	2.0
1512086420000	2.0	3.0

结果

time	speed
1512086400000	5.000
1512086410000	2.236
1512086420000	3.606

计算车辆在时间上的使用情况

车辆在行驶过程中会定时（每10秒）将数据上传到tsdb中，数据中包含车速speed。需要统计三种时长：

（1）停止时长：一段时间内这台车子有上报数据，但是上报的车速显示是0，可能是车子在等红灯。

（2）运行时长：一段时间内这台车子有上报数据，且上报的车速显示大于0，这台车子正在行驶中。

（3）离线时长：一段时间内这台车子没有上报数据的时长，这台车子已经停下并熄火了。

```
 add jar /path/to/hive-tsdb-handler_all.jar;   /* 添加jar，如果在bmr中，请使用bos地址，如bos://path/to/hive-tsdb-handler_all.jar */
CREATE EXTERNAL TABLE `vehicle`(`time` bigint, `speed` bigint, `carId` string) /* 创建表 */
STORED BY 'com.baidubce.tsdb.hive.storage.TsdbStorageHandler'                  /* 设置storage为TsdbStorageHandler */
TBLPROPERTIES (
"tsdb.metric_name" = "vehicle",                                    /* Metric名字，默认表名则被当为metric */
"tsdb.timestamp_name" = "time",                                     /* Timestamp对应的列名，默认time为对应的列名 */
"tsdb.field_names" = "speed",                                      /* field列表，多个field通过逗号分割 */
"tsdb.tag_keys" = "carId",                                        /* tag列表，多个tag通过逗号分割 */
"tsdb.endpoint" = "ENDPOINT",                                       /* TSDB实例的endpoint */
"tsdb.access_key" = "AK",                                         /* AK */
"tsdb.secret_key" = "SK"                                          /* SK */
);

/* ID为“123”的车辆在2017年12月每天的停止时长 */
select floor((time - 1512057600000) / 86400000) + 1 as day, count(*) * 10 as stop_seconds from vehicle where carId='123' and time >= 1512057600000 and time < 1514736000000 and speed = 0 group by floor((time - 1512057600000) / 86400000);

/* ID为“123”的车辆在2017年12月每天的运行时长 */
select floor((time - 1512057600000) / 86400000) + 1 as day, count(*) * 10 as run_seconds from vehicle where carId='123' and time >= 1512057600000 and time < 1514736000000 and speed > 0 group by floor((time - 1512057600000) / 86400000);

/* ID为“123”的车辆在2017年12月每天的运行时长 */
select floor((time - 1512057600000) / 86400000) + 1 as day, 2678400 - count(*) * 10 as offline_seconds from vehicle where carId='123' and time >= 1512057600000 and time < 1514736000000 group by floor((time - 1512057600000) / 86400000);
```

原始数据

metric:vehicle

time	field：speed	tag
1512086400000	40	carId=123
1512086410000	60	carId=123
1512086420000	50	carId=123
...	...	carId=123
1512086460000	10	carId=123

结果

day	stop_seconds
1	3612
2	3401
...	...
31	3013

day	run_seconds
1	17976
2	17968
...	...
31	17377

day	offline_seconds
1	64812
2	65031
...	...
31	66010

对接spark-sql

1 BMR spark sql

1.1 Spark-tsdb-connector

TSDB对接spark sql是通过实现org.apache.spark.rdd.RDD(即Resilient Distributed Dataset)和一些相关的接口，方便用户通过spark来查询TSDB的数据。

Jar下载地址：http://tsdb-bos.gz.bcebos.com/spark-tsdb-connector-all.jar

如果是本地spark集群，请下载jar到本地；如果使用bmr，则上传到bos或者直接使用地址bos://iot-tsdb/spark-tsdb-connector-all.jar。

支持的版本：spark 2.1.0，jdk 1.7。

1.2 作业程序

1.2.1 查询tsdb的通用作业程序

Main class：

```
 package com.baidu.cloud.bmr.spark;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SQLContext;

public class TsdbSparkSql {

    public static void main(String[] args) {
        if (args.length != 6) {
            System.err.println("usage: spark-submit com.baidu.cloud.bmr.spark.TsdbSparkSql <endpoint> <ak> <sk> "
                    + "<metric> <sql> <output>");
            System.exit(1);
        }
        String endpoint = args[0];
        String ak = args[1];
        String sk = args[2];
        String metric = args[3];
        String sql = args[4];
        String output = args[5];

        SparkConf conf = new SparkConf().setAppName("TsdbSparkSql");
        JavaSparkContext sc = new JavaSparkContext(conf);
        SQLContext sqlContext = new SQLContext(sc);
        Dataset<Row> dataset = sqlContext.read()
                .format("tsdb")                 // 设置为tsdb源
                .option("endpoint", endpoint)   // tsdb实例endpoint
                .option("access_key", ak)       // AK
                .option("secret_key", sk)       // SK
                .option("metric_name", metric)  // 对应的metric
                .load();
        dataset.registerTempTable(metric);
        sqlContext.sql(sql).rdd().saveAsTextFile(output);   // 执行sql并存储到output中
    }

}
```

endpoint为IP:PORT格式的情形下：

```
 package com.baidu.cloud.bmr.spark;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SQLContext;

public class TsdbSparkSql {

    public static void main(String[] args) {
        if (args.length != 8) {
            System.err.println("usage: spark-submit com.baidu.cloud.bmr.spark.TsdbSparkSql <endpoint> <host>
<grpc_port> <ak> <sk> "
                    + "<metric> <sql> <output>");
            System.exit(1);
        }
        String endpoint = args[0];
        String host = args[1];
        String grpcPort = args[2];
        String ak = args[3];
        String sk = args[4];
        String metric = args[5];
        String sql = args[6];
        String output = args[7];
        SparkConf conf = new SparkConf().setAppName("TsdbSparkSql");
        JavaSparkContext sc = new JavaSparkContext(conf);
        SQLContext sqlContext = new SQLContext(sc);
        Dataset<Row> dataset = sqlContext.read()
                .format("tsdb")                 // 设置为tsdb源
                .option("endpoint", endpoint)   // tsdb实例endpoint
                .option("host", host)           // host
                .option("grpc_port", grpcPort)  // grpc port
                .option("access_key", ak)       // AK
                .option("secret_key", sk)       // SK
                .option("metric_name", metric)  // 对应的metric
                .load();
        dataset.registerTempTable(metric);
        sqlContext.sql(sql).rdd().saveAsTextFile(output);   // 执行sql并存储到output中

    }
}
```

依赖：

```
 <dependency>
   <groupId>org.apache.spark</groupId>
   <artifactId>spark-sql_2.10</artifactId>
   <version>2.1.2</version>
</dependency>
```

需要将程序打包为jar文件，放入bos中，在配置作业时需要用到该文件的bos路径。

1.2.2 更多参数的作业程序

Main class：

```
package com.baidu.cloud.bmr.spark;

import static org.apache.spark.sql.types.DataTypes.DoubleType;
import static org.apache.spark.sql.types.DataTypes.LongType;
import static org.apache.spark.sql.types.DataTypes.StringType;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SQLContext;
import org.apache.spark.sql.types.Metadata;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;

public class TsdbSparkSqlMoreOptions {

    public static void main(String[] args) {
        if (args.length != 6) {
            System.err.println("usage: spark-submit com.baidu.cloud.bmr.spark.TsdbSparkSqlMoreOptions <endpoint> <ak>
<sk> "
                    + "<metric> <sql> <output>");
            System.exit(1);
        }
        String endpoint = args[0];
        String ak = args[1];
        String sk = args[2];
        String metric = args[3];
        String sql = args[4];
        String output = args[5];

        SparkConf conf = new SparkConf().setAppName("TsdbSparkSqlMoreOptions");
        JavaSparkContext sc = new JavaSparkContext(conf);
        SQLContext sqlContext = new SQLContext(sc);
        StructType schema = new StructType(new StructField[] {
                new StructField("time", LongType, false, Metadata.empty()),     // 设置time列，为long
                new StructField("value", DoubleType, false, Metadata.empty()),  // 设置value列，为double
                new StructField("city", StringType, false, Metadata.empty())    // 设置city列，为string
        });
        Dataset<Row> dataset = sqlContext.read()
                .format("tsdb")                 // 设置为tsdb源
                .schema(schema)                 // 设置自定义schema
                .option("endpoint", endpoint)   // tsdb实例endpoint
                .option("access_key", ak)       // AK
                .option("secret_key", sk)       // SK
                .option("metric_name", metric)  // 对应的metric
                .option("field_names", "value") // schema中属于field的列名，用逗号分割，如"field1, field2"，表示有两个field
分别为field1，field2
                .option("tag_names", "city")    // 指定schema中的tag名，用逗号分割，"city,latitude"，表示有两个tag分别为
city，latitude
                .option("split_number", "10")   // 设置split的个数，split数据时会尽量与split number接近。
                .load();
        dataset.registerTempTable(metric);
        sqlContext.sql(sql).rdd().saveAsTextFile(output);
    }

}
```

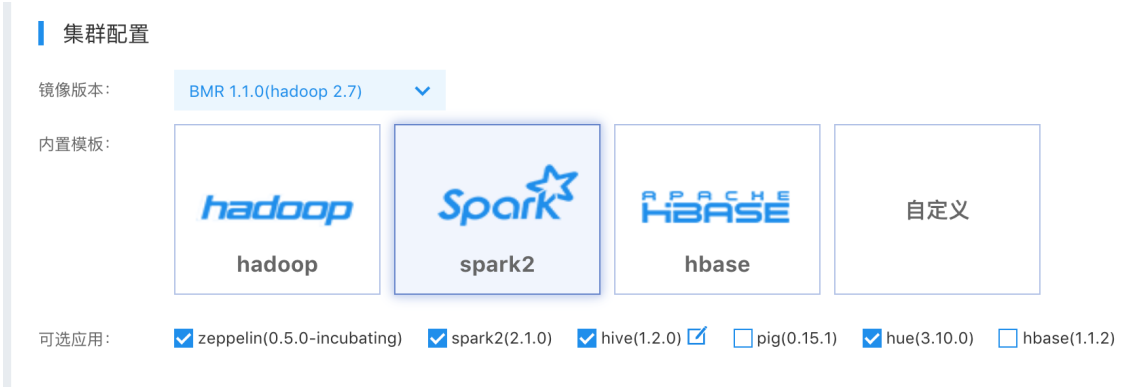
依赖：

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.10</artifactId>
  <version>2.1.2</version>
</dependency>
```

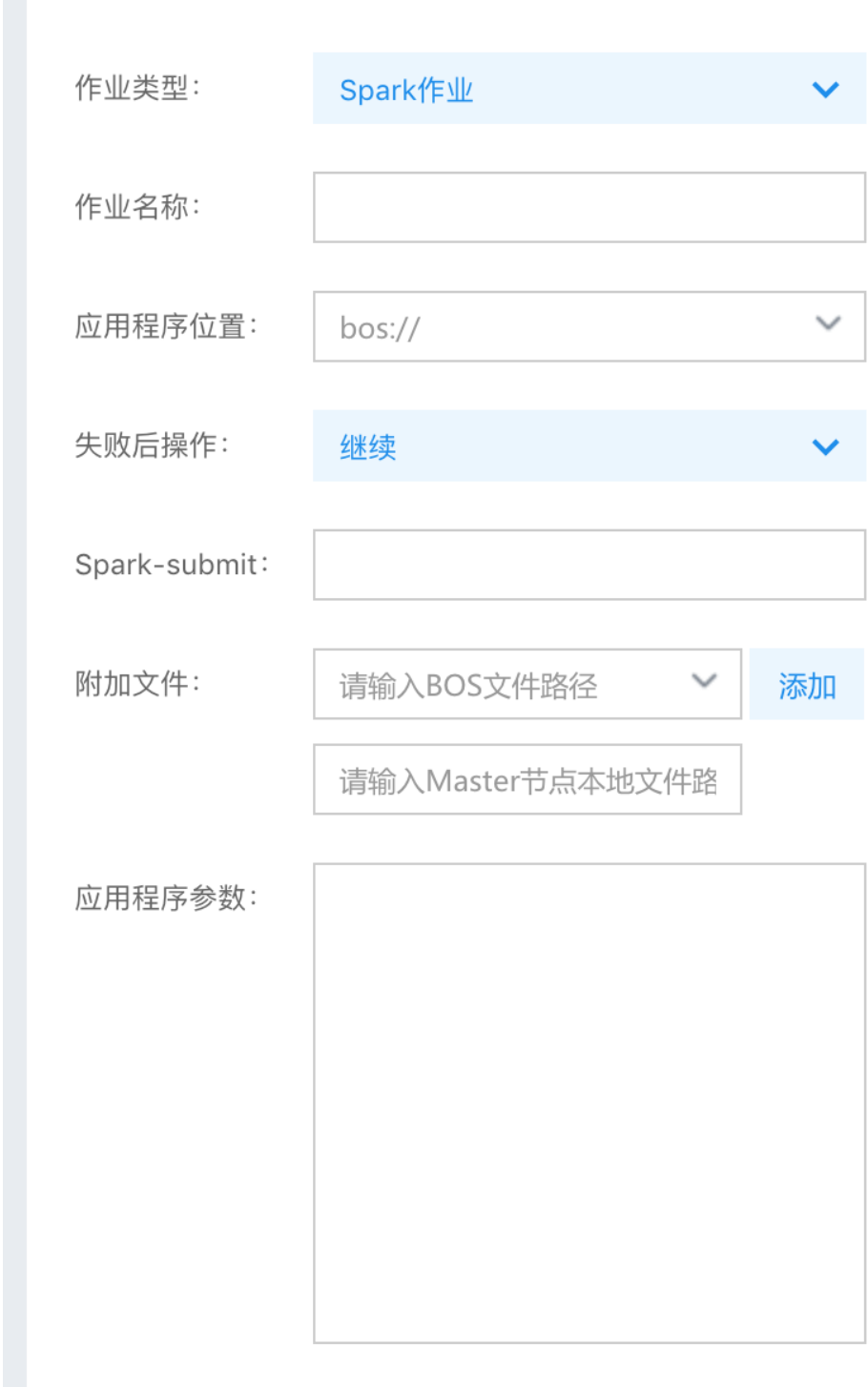
1.3 创建bmr spark集群

在使用BMR时，强烈建议您先阅读BMR文档.

选择BMR1.1.0版本，并选择spark 2.1.0。



1.4 创建作业



作业配置如下：

应用程序位置：bos://<tsdb-spark-sql-sample>.jar

Spark-submit：--class com.baidu.cloud.bmr.spark.TsdbSparkSql --jars bos://<to>/spark-tsdb-connector-all.jar

应用程序参数：<databaseName>.<databaseId>.tsdb.iot.gz.baidubce.com <AK> <SK> <metric> "select count(1) from <metric>" "bos://<to>/output/data"

需要注意的是：

- 应用程序配置其实是与2.1中作业程序相关的，请根据自己的作业程序来配置；
- Spark-submit中记得需要最后的“--jars”参数不能省略，需要指定为1.1中tsdb的connector。

1.5 场景示例

1.5.1 计算风速

风速数据由传感器定时上传到tsdb中，数据包含两个field分别为x和y，表示x轴和y轴方向的风速，如下由两个垂直方向的风速来计算出总的风速。

Main class：

```
package com.baidu.cloud.bmr.spark;

import static org.apache.spark.sql.types.DataTypes.DoubleType;
import static org.apache.spark.sql.types.DataTypes.LongType;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SQLContext;
import org.apache.spark.sql.types.Metadata;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;

public class WindSpeed {

    public static void main(String[] args) {
        String endpoint = "<endpoint>";
        String ak = "<AK>";
        String sk = "<SK>";
        String metric = "WindSpeed";
        String output = "bos://<to>/output/data";

        SparkConf conf = new SparkConf().setAppName("TsdbSparkSqlMoreOptions");
        JavaSparkContext sc = new JavaSparkContext(conf);
        SQLContext sqlContext = new SQLContext(sc);
        StructType schema = new StructType(new StructField[] {
                new StructField("time", LongType, false, Metadata.empty()),  // 设置time列，为long
                new StructField("x", DoubleType, false, Metadata.empty()),   // 设置x列，为double
                new StructField("y", DoubleType, false, Metadata.empty())    // 设置y列，为double
        });
        Dataset<Row> dataset = sqlContext.read()
                .format("tsdb")                 // 设置为tsdb源
                .schema(schema)                 // 设置自定义schema
                .option("endpoint", endpoint)   // tsdb实例endpoint
                .option("access_key", ak)       // AK
                .option("secret_key", sk)       // SK
                .option("metric_name", metric)  // 对应的metric
                .option("field_names", "x,y")   // schema中属于field的列名
                .load();
        dataset.registerTempTable(metric);
        sqlContext.sql("select time, sqrt(pow(x, 2) + pow(y, 2)) as speed from WindSpeed")
                .rdd()
                .saveAsTextFile(output);
    }

}
```

依赖：

```
 <dependency>
   <groupId>org.apache.spark</groupId>
   <artifactId>spark-sql_2.10</artifactId>
   <version>2.1.2</version>
</dependency>
```

原始数据：

metric:WindSpeed

time	field : x	field : y
1512086400000	3.0	4.0
1512086410000	1.0	2.0
1512086420000	2.0	3.0

结果：

结果输出到output指定的bos文件夹中，样例如下

```
 [1512086400000,5.000]
[1512086410000,2.236]
[1512086420000,3.606]
```

1.5.2 计算车辆在时间上的使用情况

车辆在行驶过程中会定时（每10秒）将数据上传到tsdb中，数据中包含车速speed。需要统计三种时长：

（1）停止时长：一段时间内这台车子有上报数据，但是上报的车速显示是0，可能是车子在等红灯。

（2）运行时长：一段时间内这台车子有上报数据，且上报的车速显示大于0，这台车子正在行驶中。

（3）离线时长：一段时间内这台车子没有上报数据的时长，这台车子已经停下并熄火了。

Main class：

```
package com.baidu.cloud.bmr.spark;

import static org.apache.spark.sql.types.DataTypes.LongType;
import static org.apache.spark.sql.types.DataTypes.StringType;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SQLContext;
import org.apache.spark.sql.types.Metadata;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;

public class VehicleSpeed {

    public static void main(String[] args) {
        String endpoint = "<endpoint>";
        String ak = "<AK>";
        String sk = "<SK>";
        String metric = "vehicle";
        String output = "bos://<to>/output/data";

        SparkConf conf = new SparkConf().setAppName("TsdbSparkSqlMoreOptions");
```

```
        SparkConf conf = new SparkConf().setAppName("TsdbSparkSqlMoreOptions");
        JavaSparkContext sc = new JavaSparkContext(conf);
        SQLContext sqlContext = new SQLContext(sc);
        StructType schema = new StructType(new StructField[] {
                new StructField("time", LongType, false, Metadata.empty()),     // 设置time列，为long
                new StructField("speed", LongType, false, Metadata.empty()),    // 设置speed列，为long
                new StructField("carId", StringType, false, Metadata.empty())   // 设置carId列，为string
        });
        Dataset<Row> dataset = sqlContext.read()
                .format("tsdb")                 // 设置为tsdb源
                .schema(schema)                 // 设置自定义schema
                .option("endpoint", endpoint)   // tsdb实例endpoint
                .option("access_key", ak)       // AK
                .option("secret_key", sk)       // SK
                .option("metric_name", metric)  // 对应的metric
                .option("field_names", "speed") // schema中属于field的列名
                .option("tag_names", "cardId")  // 指定tag名
                .load();
        dataset.registerTempTable(metric);
        sqlContext.sql("select floor((time - 1512057600000) / 86400000) + 1 as day, count(*) * 10 as stop_seconds"
                + " from vehicle where carId='123' and time >= 1512057600000 and time < 1514736000000 and speed =
0"
                + " group by floor((time - 1512057600000) / 86400000)")
                .rdd()
                .saveAsTextFile(output + "/stopSeconds");
        sqlContext.sql("select floor((time - 1512057600000) / 86400000) + 1 as day, count(*) * 10 as run_seconds"
                + " from vehicle where carId='123' and time >= 1512057600000 and time < 1514736000000 and speed >
0"
                + " group by floor((time - 1512057600000) / 86400000)")
                .rdd()
                .saveAsTextFile(output + "/runSeconds");
        sqlContext.sql("select floor((time - 1512057600000) / 86400000) + 1 as day, 2678400 - count(*) * 10 as"
                + " offline_seconds from vehicle where carId='123' and time >= 1512057600000 and time <
1514736000000"
                + " group by floor((time - 1512057600000) / 86400000)")
                .rdd()
                .saveAsTextFile(output + "/offlineSeconds");
    }

}
```

依赖：

```
 <dependency>
   <groupId>org.apache.spark</groupId>
   <artifactId>spark-sql_2.10</artifactId>
   <version>2.1.2</version>
</dependency>
```

原始数据

metric：vehicle

time	field：speed	tag
1512057600000	40	carId=123
1512057610000	60	carId=123
1512057620000	50	carId=123
... ...	...	carId=123
1514721600000	10	carId=123

结果

结果输出到output指定的bos文件夹中，样例如下：

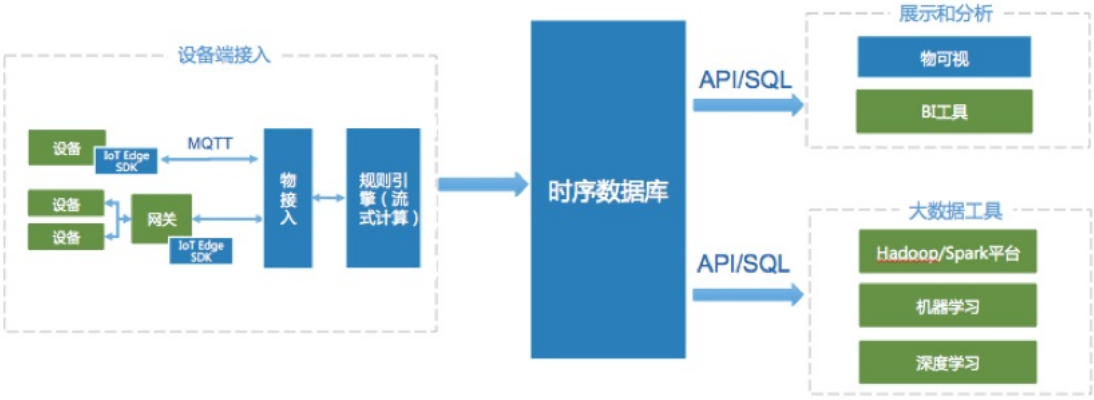
```
 [1,3612]
[2,3401]
...
[31,3013]

[1,17976]
[2,17968]
...
[31,17377]

[1,64812]
[2,65031]
...
[31,66010]
```

典型实践

物联网设备状态监控存储分析

各种物联网设备通过百度智能云天工物联网平台接入上云，设备的状态数据实时高效写入到时序数据库中。您可以：

- 通过时序数据库的数据API接口读取实时数据；
- 利用时序数据库查询快的优势对数据进行各种聚合运算得到数据报表；
- 通过时序数据库的控制台或者物可视直观得到数据的变化趋势和曲线，帮助用户分析数据内涵。

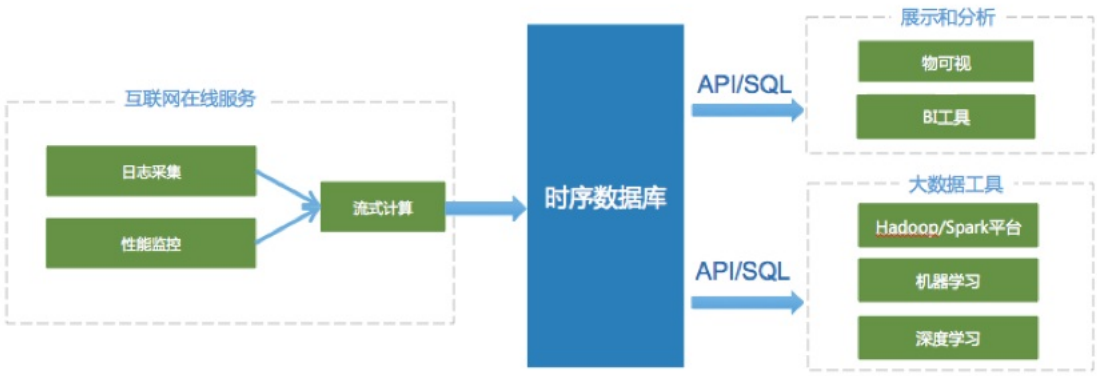


互联网业务性能监控服务

互联网服务可以将用户的访问延迟、查询返回效率、业务服务指标监控数据写入时序数据库中，时序数据库可以做多维度的聚

合分析和监控项展示。举个例子，一个音视频点播的服务商，需要将每个房间的清晰度、流畅度、是否卡顿等监控信息实时记录下来，即可以将这些监控项以一定的频率写入时序数据库。您可以：

- 按照一定的聚合条件得到某一段时间内哪一个运营商的网络更流畅
- 查看过去一天/一周某个房间的流畅度曲线
- 分析同时在线的房间数与房间清晰度、流畅度之间的关系



Java-SDK

概述

本文档主要介绍TSDB Java SDK的安装和使用。在使用本文档前，您需要先了解TSDB的一些基本知识，并已经开通了TSDB服务同时创建了数据库。若您还不了解TSDB，可以参考产品描述和操作指南。

安装SDK工具包

运行环境

Java SDK工具包可在jdk1.7、jdk1.8环境下运行。

方式一：使用Maven安装

在Maven的pom.xml文件中添加bce-java-sdk的依赖：

```
<dependency>
  <groupId>com.baidubce</groupId>
  <artifactId>bce-java-sdk</artifactId>
  <version>{version}</version>
</dependency>
```

其中，{version}为版本号，可以在SDK下载页面找到。

方式二：直接使用JAR包安装

步骤如下：

1. 下载最新版Java SDK压缩工具包。
2. 将下载的bce-java-sdk-{version}.zip解压后，复制到工程文件夹中。
3. 在Eclipse右键“工程 -> Properties -> Java Build Path -> Add JARs”。
4. 添加SDK工具包lib/bce-java-sdk-{version}.jar和第三方依赖工具包third-party/*.jar。

其中，{version}为版本号，可以在SDK下载页面找到。

SDK目录结构

```
com.baidubce
    ├─── auth                          //BCE签名相关类
    ├─── http                          //BCE的Http通信相关类
    ├─── internal                      //SDK内部类
    ├─── model                         //BCE公用model类
    ├─── services
    │       └─── tsdb                  //TSDB服务相关类
    │           ├─── model             //TSDB内部model，如Request或Response
    │           ├─── TsdbClient.class      //TSDB客户端入口类
    │           └─── TsdbConstants.class       //TSDB特有的常量
    ├─── util                          //BCE公用工具类
    ├─── BceClientConfiguration.class          //对BCE的HttpClient的配置
    ├─── BceClientException.class              //BCE客户端的异常类
    ├─── BceServiceException.class             //与BCE服务端交互后的异常类
    ├─── ErrorCode.class               //BCE通用的错误码
    └─── Region.class                  //BCE提供服务的区域
```

Demo工程下载

为了帮助您更好的使用JAVA SDK，我们提供Demo工程，点击下载。

快速入门

1. 创建TsdbClient

   TsdbClient是与TSDB服务交互的客户端，TSDB Java SDK的TSDB操作都是通过TsdbClient完成的。用户可以参考创建TsdbClient，完成初始化客户端的操作。

2. 写入数据点

   构建数据点，包括Metric、Tag和Value，并将数据点写入TSDB。

3. 查询操作

   获取Metric、Tag列表，对当前数据进行过滤、分组和筛选。

4. 其它可选操作：

   - 生成查询数据点的预签名URL

     预签名URL可以用于前端页面查询数据点。前端请求服务器生成预签名url并返回给前端，前端使用该URL发起ajax请求查询数据点。

   - 写入数据点的gzip压缩

     写入数据点默认开启gzip压缩。如果不需要使用，可以关闭gzip压缩。

创建TsdbClient

用户可以参考如下代码新建一个TsdbClient：

HTTP Client

```
 String ACCESS_KEY_ID = <your-access-key-id>;              // 用户的Access Key ID
String SECRET_ACCESS_KEY = <your-secret-access-key>;      // 用户的Secret Access Key
String ENDPOINT = <your-tsdb-database-endpoint>;          // 用户的时序数据库域名，形式如
databasename.tsdb.iot.gz.baidubce.com

// 创建配置
BceClientConfiguration config = new BceClientConfiguration()
            .withCredentials(new DefaultBceCredentials(ACCESS_KEY_ID, SECRET_ACCESS_KEY))
            .withEndpoint(ENDPOINT);

// 初始化一个TsdbClient
TsdbClient tsdbClient = new TsdbClient(config);
```

HTTPS Client

```
 String ACCESS_KEY_ID = <your-access-key-id>;              // 用户的Access Key ID
String SECRET_ACCESS_KEY = <your-secret-access-key>;      // 用户的Secret Access Key
String ENDPOINT = <your-tsdb-database-endpoint>;          // 用户的时序数据库域名，形式如
databasename.tsdb.iot.gz.baidubce.com

// 创建配置
BceClientConfiguration config = new BceClientConfiguration()
            .withProtocol(Protocol.HTTPS)             // 使用HTTPS协议;不设置，默认使用HTTP协议
            .withCredentials(new DefaultBceCredentials(ACCESS_KEY_ID, SECRET_ACCESS_KEY))
            .withEndpoint(ENDPOINT);

// 初始化一个TsdbClient
TsdbClient tsdbClient = new TsdbClient(config);
```

在上面代码中，变量ACCESS_KEY_ID与SECRET_ACCESS_KEY是由系统分配给用户的，均为字符串，用于标识用户，为访问TSDB做签名验证。其中ACCESS_KEY_ID对应控制台中的“Access Key ID”，SECRET_ACCESS_KEY对应控制台中的“Access Key Secret”，获取方式请参考获取AK/SK。

通过IP访问

在一些场景下比如正向代理，无法采用域名直接访问tsdb，java sdk也支持通过IP访问

```
 String ACCESS_KEY_ID = <your-access-key-id>;              // 用户的Access Key ID
String SECRET_ACCESS_KEY = <your-secret-access-key>;      // 用户的Secret Access Key
String ENDPOINT = <ip>;                                   // 时序数据库的ip或者正向代理的ip
String databaseName = <database-name>;                    // 创建的时序数据库名称

// 创建配置
BceClientConfiguration config = new BceClientConfiguration()
            .withCredentials(new DefaultBceCredentials(ACCESS_KEY_ID, SECRET_ACCESS_KEY))
            .withEndpoint(ENDPOINT);

// 初始化一个TsdbClient
TsdbClient tsdbClient = new TsdbClient(config, databaseName);
```

参数说明

BceClientConfiguration中有更多的配置项，可配置如下参数：

参数	说明
connectionTimeoutInMillis	建立连接的超时时间（单位：毫秒）
localAddress	本地地址
maxConnections	允许打开的最大HTTP连接数
protocol	连接协议类型
proxyDomain	访问NTLM验证的代理服务器的Windows域名
proxyHost	代理服务器主机地址
proxyPassword	代理服务器验证的密码
proxyPort	代理服务器端口
proxyPreemptiveAuthenticationEnabled	是否设置用户代理认证
proxyUsername	代理服务器验证的用户名
proxyWorkstation	NTLM代理服务器的Windows工作站名称
retryPolicy	连接重试策略
socketBufferSizeInBytes	Socket缓冲区大小
socketTimeoutInMillis	通过打开的连接传输数据的超时时间（单位：毫秒）
userAgent	用户代理，指HTTP的User-Agent头

写入操作

写入单域数据点

用户可以参考如下代码写入单域数据点：

> 注意：当写入的metric、field、tags、timestamp都相同时，后写入的value会覆盖先写入的value。

```
 String METRIC = "wind";                                  // metric
String TAG_KEY = "city";                                 // 标签名称
String TAG_VALUE = "ShangHai";                               // 标签值
String FIELD = "direction";                              // 域

// 创建数据点
List<Datapoint> datapoints = Arrays.asList(new Datapoint()
            .withMetric(METRIC)                           // 设置Metric
            .withField(FIELD)                            // 设置数据点域，可选，不填使用默认域名 value
            .addTag(TAG_KEY, TAG_VALUE)                          // 设置Tag
            .addDoubleValue(System.currentTimeMillis(), 0.1));  // 添加一个数据点


tsdbClient.writeDatapoints(datapoints);
```

一个Datapoint对象可以同时添加多个数据点，这些数据点使用相同的metric和标签，相同的域。多个相同metric和标签的数据放入同一个Datapoint对象，可以减少payload。

```
Datapoint datapoint = new Datapoint()
            .withMetric(METRIC)                           // 设置Metric
            .withField(FIELD)                            // 设置数据点域，可选，不填使用默认域名 value
            .addTag(TAG_KEY, TAG_VALUE)                          // 设置Tag
            .addDoubleValue(System.currentTimeMillis(), 0.1)        // 添加一个数据点
            .addDoubleValue(System.currentTimeMillis() + 1, 0.2);    // 再添加一个数据点
```

Datapoint对象可以添加double，long和String类型的数据点。对于同一个field，如果写入了某个数据类型的value之后，相同的field不允许写入其他数据类型。

```
 // 添加Double类型数据点
Datapoint datapoint1 = new Datapoint()
        .withMetric("wind")                         // 设置Metric
        .withField(FIELD)                          // 设置数据点域，可选，不填使用默认域名 value
        .addTag(TAG_KEY, TAG_VALUE)                     // 设置Tag
        .addDoubleValue(System.currentTimeMillis(), 0.1);      // 添加Double类型数据点

// 添加Long类型数据点
Datapoint datapoint2 = new Datapoint()
        .withMetric("temperature")                      // 设置Metric
        .addTag(TAG_KEY, TAG_VALUE)                     // 设置Tag
        .addLongValue(System.currentTimeMillis(), 10L);        // 添加Long类型数据点

// 添加String类型数据点
Datapoint datapoint3 = new Datapoint()
        .withMetric("humidity")                         // 设置Metric
        .addTag(TAG_KEY, TAG_VALUE)                     // 设置Tag
        .addStringValue(System.currentTimeMillis(), "string");  // 添加String类型数据点

// 添加BigDecimal类型数据点
Datapoint datapoint4 = new Datapoint()
        .withMetric("metric")                                   // 设置Metric
        .addTag(TAG_KEY, TAG_VALUE)                                 // 设置Tag
        .addBigDecimalValue(System.currentTimeMillis(), BigDecimal.valueOf(1.234));  // 添加BigDecimal类型数据点
```

写入多域数据点

不同的域并不需要同时写入，可以认为不同的域都是独立的。但如果查询时要用一条语句查出来，需要保证metric、所有的tag、时间戳都是一致的。

可以参考以下代码写入多域数据点：

```
 String METRIC = "wind";                                // metric
String TAG_KEY = "city";                               // 标签名称
String TAG_VALUE = "ShangHai";                             // 标签值
String FIELD_1 = "direction";                           // 域1
String FIELD_2 = "speed";                               // 域2
long TIME = System.currentTimeMillis();                        // 时间

// 添加FIELD_1的数据点
Datapoint datapoint1 = new Datapoint()
        .withMetric(METRIC)                             // 设置Metric
        .withField(FIELD_1)                            // 设置域1
        .addTag(TAG_KEY, TAG_VALUE)                         // 设置Tag
        .addDoubleValue(TIME, 0.1);                       // 指定时间写入Double类型数据点

// 添加FIELD_2的数据点
Datapoint datapoint2 = new Datapoint()
        .withMetric(METRIC)                             // 设置Metric,需要和FIELD1的一样
        .withField(FIELD_2)                            // 设置域2
        .addTag(TAG_KEY, TAG_VALUE)                         // 设置Tag，需要和FIELD1的一样
        .addLongValue(TIME, 10L);                         // 指定时间添加Long类型数据点，时间需要和FIELD_1的一样
tsdbClient.writeDatapoints(Arrays.asList(datapoint1, datapoint2));
```

查询操作

获取度量（Metric）

如下代码可以获取metric列表：

```
 // 获取Metric
GetMetricsResponse response = tsdbClient.getMetrics();

// 打印结果
for(String metric : response.getMetrics()) {
    System.out.println(metric);
}
```

获取Field

如下代码可以获取Field列表：

```
 String METRIC = "metric";

// 获取Field
GetFieldsResponse response = tsdbClient.getFields(METRIC);

// 打印结果
for(Map.Entry<String, FieldInfo> entry : response.getFields().entrySet()) {
    System.out.println(entry.getKey() + ":");
    System.out.println("\t" + entry.getValue().getType());
}
```

获取标签（Tag）

如下代码可以获取标签（Tag）列表：

```
 String METRIC = "cpu_idle";                          // 设置需要获取tag的metric

// 获取标签
GetTagsResponse response = tsdbClient.getTags(METRIC);

// 打印结果
for(Map.Entry<String, List<String>> entry : response.getTags().entrySet()) {
    System.out.println(entry.getKey() + ":");
    for(String tagValue : entry.getValue()) {
        System.out.println("\t" + tagValue);
    }
}
```

查询数据点

如下代码可以查询数据点：

```
 String metric = "wind";
String field = "direction";

// 构造查询对象
List<Query> queries = Arrays.asList(new Query()              // 创建Query对象
        .withMetric(metric)                                  // 设置metric
        .withField(field)                                   // 设置查询的域名，不设置表示查询默认域
        .withFilters(new Filters()                           // 创建Filters对象
                .withRelativeStart("2 hours ago"))           // 设置相对的开始时间，这里设置为2小时前
        .withLimit(100)                                     // 设置返回数据点数目限制
        .addAggregator(new Aggregator()                      // 创建Aggregator对象
                .withName(TsdbConstants.AGGREGATOR_NAME_SUM)  // 设置聚合函数为Sum
                .withSampling("1 second")));                 // 设置采样

// 查询数据
QueryDatapointsResponse response = tsdbClient.queryDatapoints(queries);

// 打印结果
for (Result result : response.getResults()) {
    System.out.println("Result:");
    for (Group group : result.getGroups()) {
        System.out.println("\tGroup:");

        for (GroupInfo groupInfo : group.getGroupInfos()) {
            System.out.println("\t\tGroupInfo:");
            System.out.println("\t\t\tName:" + groupInfo.getName());
        }
        System.out.println("\t\tTimeAndValue:");
        try {
            for (Group.TimeAndValue timeAndValue : group.getTimeAndValueList()) {
                if (timeAndValue.isDouble()) {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getDoubleValue() + "]");
                } else if (timeAndValue.isLong()) {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getLongValue() + "]");
                } else {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getStringValue() + "]");
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

多域查询数据点

如下代码可以查询数据点：

```
 tring metric = "wind";
String field1 = "direction";
String field2 = "speed";
String tag = "city";

// 构造查询对象
List<String> fields = Arrays.asList(field1, field2);
List<Query> queries = Arrays.asList(new Query()            // 创建Query对象
        .withMetric(metric)                                // 设置metric
        .withFields(fields)                                // 设置查询的域名列表，不设置表示查询默认域，和field冲突
        .withTags(Arrays.asList(tag))                      // 设置查询的Tag列表，不设置表示不查询，
        .withFilters(new Filters()                         // 创建Filters对象
                .withRelativeStart("2 hours ago")          // 设置相对的开始时间，这里设置为2小时前
                .withRelativeEnd("1 second ago"))          // 设置相对的结束时间，不设置则默认为到当前时间为止
        .withLimit(100));

// 查询数据，返回结果的顺序和请求的field顺序相同
QueryDatapointsResponse response = tsdbClient.queryDatapoints(queries);

// 打印结果
for (Result result : response.getResults()) {
    System.out.println("Result:");
    for (Group group : result.getGroups()) {
        System.out.println("\tGroup:");

        for (GroupInfo groupInfo : group.getGroupInfos()) {
            System.out.println("\t\tGroupInfo:");
            System.out.println("\t\t\tName:" + groupInfo.getName());
        }
        System.out.println("\t\tTimeAndValues:");
        try {
            for (Group.TimeAndValue timeAndValue : group.getTimeAndValueList()) {
                System.out.print("\t\t\t[" + timeAndValue.getTime());
                for (int index = 0; index < fields.size(); index++) {
                    System.out.print(", ");
                    if (!timeAndValue.isNull(index)) {
                        if (timeAndValue.isDouble(index)) {
                            System.out.print(timeAndValue.getDoubleValue(index));
                        } else if (timeAndValue.isLong(index)) {
                            System.out.print(timeAndValue.getLongValue(index));
                        } else {
                            System.out.print(timeAndValue.getStringValue(index));

                        }
                    } else {
                        System.out.print("null");
                    }
                }
                System.out.println("]");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

查询数据点和对应的tags

如下代码可以查询数据点的同时返回对应的tags信息：

```
 String metric = "wind";
String field = "direction";
String tagKey1 = "city";
String tagKey2 = "province";                                    // 构造查询对象
List<Query> queries = Arrays.asList(new Query()                     // 创建Query对象
        .withMetric(metric)                                   // 设置metric
        .withField(field)                                    // 设置查询的域名，不设置表示查询默认域
        .withTags(Arrays.asList(tagKey1, tagKey2))                  // 设置需要同时返回的tag key列表
        .withFilters(new Filters()                            // 创建Filters对象
                .withRelativeStart("2 hours ago"))                  // 设置相对的开始时间
        .withLimit(100));                                     // 设置返回数据点数目限制


// 查询数据
QueryDatapointsResponse response = tsdbClient.queryDatapoints(queries);

// 打印结果
for (Result result : response.getResults()) {
    System.out.println("Result:");
    for (Group group : result.getGroups()) {
        System.out.println("\tGroup:");

        for (GroupInfo groupInfo : group.getGroupInfos()) {
            System.out.println("\t\tGroupInfo:");
            System.out.println("\t\t\tName:" + groupInfo.getName());
        }
        System.out.println("\t\tTimeAndValue:");
        try {
            for (Group.TimeAndValue timeAndValue : group.getTimeAndValueList()) {
                // 打印的格式为 [时间,FIELD_VALUE,TAG_1_VALUE,TAG_2_VALUE]
                if (timeAndValue.isDouble()) {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getDoubleValue(0) + "," +
timeAndValue.getStringValue(1) + "," + timeAndValue.getStringValue(2) + "]");
                } else if (timeAndValue.isLong()) {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getLongValue(0) + "," +
timeAndValue.getStringValue(1) + "," + timeAndValue.getStringValue(2) + "]");
                } else {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getStringValue(0) + "," +
timeAndValue.getStringValue(1) + "," + timeAndValue.getStringValue(2) + "]");
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

使用插值方式查询数据点

如下代码可以对数据点进行插值：

```
 String metric = "wind";
String field = "direction";

// 构造查询对象
List<Query> queries = Arrays.asList(new Query()                    // 创建Query对象
        .withMetric(metric)                                        // 设置metric
        .withField(field)                                          // 设置查询的域名，不设置表示查询默认域
        .withFilters(new Filters()                                 // 创建Filters对象
                .withRelativeStart("2 hours ago")                  // 设置相对的开始时间
                .withRelativeEnd("1 second ago"))                  // 设置相对的结束时间，不设置则默认为到当前时间为止
        .withFill(new Fill()
                .withType(TsdbConstants.FILL_TYPE_LINEAR)          // 设置插值类型，这里设置成线性插值
                .withInterval("5 minutes")                         // 设置插值间隔
                .withMaxWriteInterval("10 minutes")));             // 设置最大写入间隔

// 查询数据
QueryDatapointsResponse response = tsdbClient.queryDatapoints(queries);

// 打印结果
for (Result result : response.getResults()) {
    System.out.println("Result:");
    for (Group group : result.getGroups()) {
        System.out.println("\tGroup:");

        for (GroupInfo groupInfo : group.getGroupInfos()) {
            System.out.println("\t\tGroupInfo:");
            System.out.println("\t\t\tName:" + groupInfo.getName());
        }
        System.out.println("\t\tTimeAndValue:");
        try {
            for (Group.TimeAndValue timeAndValue : group.getTimeAndValueList()) {
                if (timeAndValue.isDouble()) {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getDoubleValue() + "]");
                } else if (timeAndValue.isLong()) {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getLongValue() + "]");
                } else {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getStringValue() + "]");
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

分页查询原始数据点

TSDB支持两种分页查询的方式，一种是marker + limit的方式，一种是offset + limit的方式。 marker + limit的方式通过marker能在索引层确定查询的起始位置，因此查询速度更快，查询成本消耗更少。 offset + limit的方式需要将查询执行完成后，才能找到结果集offset的位置，因此查询成本消耗相对较大。当offset值过大时，查询返回耗时可能变大。

如下代码为marker + limit的分页查询原始数据点：

```
 String metric = "wind";
String field = "direction";

// 构造查询对象
Query query = new Query()                                   // 创建Query对象
        .withMetric(metric)                                 // 设置metric
        .withField(field)                                   // 设置查询的域名，不设置表示查询默认域
        .withFilters(new Filters()                          // 创建Filters对象
                .withRelativeStart("1 week ago")            // 设置相对的开始时间，这里设置为1周前
                .withRelativeEnd("1 second ago"));          // 设置相对的结束时间，不设置则默认为到当前时间为止

while (true) {
    // 查询数据
    QueryDatapointsResponse response = tsdbClient.queryDatapoints(Arrays.asList(query));
    // 打印结果
    Result result = response.getResults().get(0);
    System.out.println("Result:");
    for (Group group : result.getGroups()) {
        System.out.println("\tGroup:");
        for (GroupInfo groupInfo : group.getGroupInfos()) {
            System.out.println("\t\tGroupInfo:");
            System.out.println("\t\t\tName:" + groupInfo.getName());
        }
        System.out.println("\t\tTimeAndValue:");
        try {
            for (Group.TimeAndValue timeAndValue : group.getTimeAndValueList()) {
                if (timeAndValue.isDouble()) {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getDoubleValue() + "]");
                } else if (timeAndValue.isLong()) {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getLongValue() + "]");
                } else {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getStringValue() + "]");
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    // 如果没有下一页数据，则退出循环
    if (!result.isTruncated()) {
        break;
    }
    // 设置下一页数据的marker
    query.setMarker(result.getNextMarker());
}
```

如下是offset + limit的分页查询原始数据点：

```
 String metric = "wind";
String field = "direction";
int pageSize = 10;
Query query = new Query()                                    // 创建Query对象
        .withMetric(metric)                                  // 设置metric
        .withField(field)                                    // 设置查询的域名，不设置表示查询默认域
        .withFilters(new Filters()                           // 创建Filters对象
                .withRelativeStart("1 week ago")             // 设置相对的开始时间，这里设置为1周前
                .withRelativeEnd("1 second ago"))            // 设置相对的结束时间，不设置则默认为到当前时间为止
        .withLimit(pageSize)                                 // 设置每页大小为10
        .withOffset(0);                                      // 查询第1页的数据
QueryDatapointsResponse response = tsdbClient.queryDatapoints(Arrays.asList(query));

// 查询第2页的数据
query.setOffset(pageSize * 1);
response = tsdbClient.queryDatapoints(Arrays.asList(query));

// 查询第3页的数据
query.setOffset(pageSize * 2);
response = tsdbClient.queryDatapoints(Arrays.asList(query));
```

查询数据时使用标签过滤

示例代码如下：

```
 String metric = "wind";
String field = "direction";

// 构造查询对象
List<Query> queries = Arrays.asList(new Query()                    // 创建Query对象
        .withMetric(metric)                                        // 设置metric
        .withField(field)                                          // 设置查询的域名，不设置表示查询默认域
        .withFilters(new Filters()                                 // 创建Filters对象
                .withRelativeStart("2 hours ago")                  // 设置相对的开始时间，这里设置为2小时前
                .withRelativeEnd("1 second ago")                   // 设置相对的结束时间，不设置则默认为到当前时间为止
                .addTagFilter(new TagFilter()                      // 创建TagFilter对象
                        .withTag("tag1")                           // 设置要过滤的tagKey
                        .addNotIn("value2"))));                    // 设置不允许出现的tagValue。还可以设置允许出现的tagValue，
或者设置like匹配。
// 查询数据
QueryDatapointsResponse response = tsdbClient.queryDatapoints(queries);

// 打印结果
for (Result result : response.getResults()) {
    System.out.println("Result:");
    for (Group group : result.getGroups()) {
        System.out.println("\tGroup:");
        System.out.println("\t\tTimeAndValue:");
        try {
            for (Group.TimeAndValue timeAndValue : group.getTimeAndValueList()) {
                if (timeAndValue.isDouble()) {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getDoubleValue() + "]");
                } else if (timeAndValue.isLong()) {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getLongValue() + "]");
                } else {
                    System.out.println("\t\t\t[" + timeAndValue.getTime() + "," + timeAndValue.getStringValue() + "]");
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Filters对象设置示例

```
 // 使用绝对时间
Filters filters1 = new Filters()
        .withAbsoluteStart(System.currentTimeMillis() - 5000)
        .withAbsoluteEnd(System.currentTimeMillis() - 1000);

// 添加Tags：逐个添加
Filters filters2 = new Filters()
        .withRelativeStart("5 minutes ago")
        .withRelativeEnd("2 minutes ago")
        .addTag("tagKey1", "tagValue1", "tagValue2")
        .addTag("tagKey2", "tagValue1");

// 添加Tags：通过map添加
Map<String, List<String>> tags = new HashMap<String, List<String>>();      // 创建tags
tags.put("tagKey1", Arrays.asList("tagValue1", "tagValue2"));               // 添加tag
tags.put("tagKey2", Arrays.asList("tagValue1"));                            // 添加tag
Filters filters3 = new Filters()
        .withRelativeStart("5 minutes ago")
        .withRelativeEnd("2 minutes ago")
        .withTags(tags);
```

按值过滤的设置示例

```
 // 对long型数据进行过滤，支持的比较符为 >，<，>=，<=，= 和 !=
Filters filters1 = new Filters()
        .withAbsoluteStart(System.currentTimeMillis() - 5000)
        .withAbsoluteEnd(System.currentTimeMillis() - 1000)
        .withValue(ValueFilter.createValueFilter(ValueFilter.LESS_OR_EQUAL, 100L));    // 过滤条件为 "<= 100"

// 对double型数据进行过滤，支持的比较符为 >，<，>=，<=，= 和 !=
Filters filters1 = new Filters()
        .withAbsoluteStart(System.currentTimeMillis() - 5000)
        .withAbsoluteEnd(System.currentTimeMillis() - 1000)
        .withValue(ValueFilter.createValueFilter(ValueFilter.GREATER, 99.9));          // 过滤条件为 "> 99.9"

// 对String型数据进行过滤，支持的比较符为 >，<，>=，<=，= 和 !=
Filters filters1 = new Filters()
        .withAbsoluteStart(System.currentTimeMillis() - 5000)
        .withAbsoluteEnd(System.currentTimeMillis() - 1000)
        .withValue(ValueFilter.createValueFilter(ValueFilter.EQUAL, "stringvalue"));   // 过滤条件为 "= 'stringvalue'"

// 将数据与标签tagKey1的值进行比较过滤，支持的比较符为 >，<，>=，<=，= 和 !=
Filters filters1 = new Filters()
        .withAbsoluteStart(System.currentTimeMillis() - 5000)
        .withAbsoluteEnd(System.currentTimeMillis() - 1000)
        .withValue(ValueFilter.createValueFilterOfTag(ValueFilter.EQUAL, "tagKey1"));   // 过滤条件为 "= tagKey1"

// 对所有多域数据使用统一value进行过滤，支持的比较符为 >，<，>=，<=，= 和 !=
Filters filters1 = new Filters()
        .withAbsoluteStart(System.currentTimeMillis() - 5000)
        .withAbsoluteEnd(System.currentTimeMillis() - 1000)
        .withValue(ValueFilter.createValueFilter(ValueFilter.LESS_OR_EQUAL, 100L));    // 过滤条件为 "<= 100"

// 对多域数据使用不同的value进行过滤，支持的比较符为 >，<，>=，<=，= 和 !=
FieldFilter fieldFilter1 = new FieldFilter("field1", ValueFilter.createValueFilter("<", 100L));
FieldFilter fieldFilter2 = new FieldFilter("field2", ValueFilter.createValueFilter(">", 200L));
Filters filters1 = new Filters()
        .withAbsoluteStart(System.currentTimeMillis() - 5000)
        .withAbsoluteEnd(System.currentTimeMillis() - 1000)
        .withFields(Arrays.asList(fieldFilter1, fieldFilter2));                    // 过滤条件为 "field1< 100 && field2 > 200"
```

查询涉及的对象

查询涉及到的对象包括：Query对象，Filters对象，GroupBy对象和Aggregator对象，关于对象的具体介绍，请参看API参考中的[查询data point](TSDB/API参考/数据API接口说明.md#查询data point)。

SQL查询接口

v0.10.32版本的SDK中，支持了sql的查询。示例如下：

```
 String METRIC = "wind";
String FIELD = "direction";

String sql = "select " + FIELD + " from " + METRIC;

// 查询数据
GetRowWithSqlResponse response = tsdbClient.getRowsWithSql(sql);

// 打印表头
for (Column column : response.getColumns()) {
    System.out.print(column.getName() + "\t");
}
System.out.println();
// 打印数据行
for (List<Object> row : response.getRows()) {
    for (Object item : row) {
        System.out.print(item.toString() + "\t");
    }
    System.out.println();
}
```

生成查询数据点的预签名URL

预签名URL可以用于前端页面查询数据点。用法：前端请求服务器生成预签名url并返回给前端，前端使用该URL发起ajax请求查询数据点。

```
 String METRIC = "cpu_idle";                                    // 设置需要获取tag的metric
String FIELD = "temperature";

// 构造查询对象
List<Query> queries = Arrays.asList(new Query()                      // 创建Query对象
        .withMetric(METRIC)                                    // 设置metric
        .withField(FIELD)                                     // 设置域，不设置表示查询默认域
        .withFilters(new Filters()                             // 创建Filters对象
                .withRelativeStart("5 seconds ago")                  // 设置相对的开始时间，这里设置为5秒前
                .withRelativeEnd("1 second ago"))                    // 设置相对的结束时间，不设置则默认为到当前时间为止
        .withLimit(100)                                       // 设置返回数据点数目限制
        .addAggregator(new Aggregator()                              // 创建Aggregator对象
                .withName(TsdbConstants.AGGREGATOR_NAME_SUM)         // 设置聚合函数为Sum
                .withSampling("1 second")));                         // 设置采样

// 获取预签名URL
URL url = tsdbClient.generatePresignedUrlForQueryDatapoints(queries, 120); // 设置签名超时时间为120s
```

写入数据点的gzip压缩说明

v0.10.10版本的sdk中，写入数据点默认开启gzip压缩。如果不需要使用gzip压缩，可参考如下代码：

```
String METRIC = "cpu_idle";                                    // metric
String TAG_KEY = "server";                                     // 标签名称
String TAG_VALUE = "server1";                                  // 标签值

// 创建数据点
List<Datapoint> datapoints = Arrays.asList(new Datapoint()
            .withMetric(METRIC)                                // 设置Metric
            .addTag(TAG_KEY, TAG_VALUE)                        // 设置Tag
            .addDoubleValue(System.currentTimeMillis(), 0.1));  // 添加一个数据点
boolean useGzip = false;                                       // 不使用gzip

tsdbClient.writeDatapoints(new WriteDatapointsRequest().withDatapoints(datapoints), useGzip);
```

管理接口

v0.10.17版本的sdk中，支持用户通过API创建/删除/查询时序数据库实例。

新建TsdbAdminClient

```
String ACCESS_KEY_ID = <your-access-key-id>;               // 用户的Access Key ID
String SECRET_ACCESS_KEY = <your-secret-access-key>;       // 用户的Secret Access Key
String ADMIN_ENDPOINT = "tsdb.gz.baidubce.com";            // 注意：与新建TsdbClient时使用的endpoint不同

// 创建配置
BceClientConfiguration config = new BceClientConfiguration()
            .withCredentials(new DefaultBceCredentials(ACCESS_KEY_ID, SECRET_ACCESS_KEY))
            .withEndpoint(ADMIN_ENDPOINT);

// 初始化一个TsdbAdminClient
TsdbAdminClient tsdbAdminClient = new TsdbAdminClient(config);
```

BceClientConfiguration中的配置项参考新建TsdbAdminClient。同样的，TsdbAdminClient也支持使用HTTPS，配置方式参考TsdbClient。

创建时序数据库实例

示例代码如下：

```
String databaseName = "databasename";                          // 实例的名字
String description = "description";                            // 实例描述，可不填写
int ingestDataPointsMonthly = 1;                               // 写入额度，单位：百万点/月
int queryUnitsMonthly = 10;                                    // 查询额度，单位：万个/月
int purchaseLength = 1;                                        // 购买时长，单位：月
String couponName = <your-coupon-name>;                        // 代金券号，可不填写

CreateDatabaseRequest request = new CreateDatabaseRequest()
    .withDatabaseName(databaseName)
    .withDescription(description)
    .withIngestDataPointsMonthly(ingestDataPointsMonthly)
    .withQueryUnitsMonthly(queryUnitsMonthly)
    .withPurchaseLength(purchaseLength)
    .withCouponName(couponName);

String clientToken = <your-client-token>;                      // ClientToken, 用于保证幂等性，重试发送创建请求时，使用同一个clientToken。
CreateDatabaseResponse response = tsdbAdminClient.createDatabase(request, clientToken);
```

返回值所包含字段请参考API文档。

删除时序数据库实例

示例代码如下：

```
 String databaseId = <your-database-id>;                    // 实例的ID
tsdbAdminClient.deleteDatabase(databaseId);
```

注意，只允许删除到期的时序数据库实例，否则将报错。

获取时序数据库实例

示例代码如下：

```
 String databaseId = <your-database-id>;                    // 实例的ID
GetDatabaseResponse database = tsdbAdminClient.getDatabase(databaseId);
```

返回值所包含字段请参考API文档。

获取时序数据库实例列表

示例代码如下：

```
 ListDatabaseResponse database = tsdbAdminClient.listDatabase();
```

创建删除数据点任务

示例代码如下：

```
 // 假定用户要删除wind和temperature这两个metric下的数据点，并且每个metric都使用tagFilter进行过滤

// windTaskTagFilter是wind这个metric对应的filter
TaskTagFilter windTaskTagFilter = new TaskTagFilter();
windTaskTagFilter.setTagKey("city");
windTaskTagFilter.addIn("bj");

// temperatureTaskTagFilter是temperature这个metric对应的filter
TaskTagFilter temperatureTaskTagFilter = new TaskTagFilter();
temperatureTaskTagFilter.setTagKey("city");
temperatureTaskTagFilter.addIn("gz");

// tags的key为metric, value为metric对应的TagFilter的列表
Map<String, List<TaskTagFilter>> tags = new HashMap<String, List<TaskTagFilter>>();
// 添加wind及其对应的filter
tags.put("wind", Arrays.asList(windTaskTagFilter));
// 添加temperature及其对应的filter
tags.put("temperature", Arrays.asList(temperatureTaskTagFilter));

// 创建删除请求并获取response
DeleteDatapointsRequest deleteRequest = new DeleteDatapointsRequest();
deleteRequest.setDatabaseId("tsdb_sdfd434xxxx");
// 设置需要删除的metric，如果不设置metrics字段或者metricFieldsList字段，则会删除所有metric和field的数据，
// 删除时会去tags中查询metric对应的filter进行过滤
deleteRequest.setMetrics(Arrays.asList("wind", "temperature"));
deleteRequest.setTags(tags);
DeleteDatapointsResponse deleteResponse = tsdbAdminClient.deleteDatapoints(deleteRequest);
```

创建导出数据点任务

示例代码如下：

```
 // 假定用户要导出wind和temperature这两个metric下的数据点，并且每个metric都使用tagFilter进行过滤

// windTaskTagFilter是wind这个metric对应的filter
TaskTagFilter windTaskTagFilter = new TaskTagFilter();
windTaskTagFilter.setTagKey("city");
windTaskTagFilter.addIn("bj");

// temperatureTaskTagFilter是temperature这个metric对应的filter
TaskTagFilter temperatureTaskTagFilter = new TaskTagFilter();
temperatureTaskTagFilter.setTagKey("city");
temperatureTaskTagFilter.addIn("gz");

// tags的key为metric, value为metric对应的TagFilter的列表
Map<String, List<TaskTagFilter>> tags = new HashMap<String, List<TaskTagFilter>>();
// 添加wind及其对应的filter
tags.put("wind", Arrays.asList(windTaskTagFilter));
// 添加temperature及其对应的filter
tags.put("temperature", Arrays.asList(temperatureTaskTagFilter));

ExportDatapointsRequest exportRequest = new ExportDatapointsRequest();
exportRequest.setDatabaseId("tsdb_sdfd434xxxx");
exportRequest.setBosUrl("bos://iot-tsdb/test/");
exportRequest.setFormat("csv");
exportRequest.setSingleFile(true);
// 设置需要导出的metrics，如果不设置，会导出所有的metric，对于每个需要导出的metric，会去tags中查询对应的filter
// 进行过滤
exportRequest.setMetrics(Arrays.asList("wind", "temperature"));
exportRequest.setTags(tags);
ExportDatapointsResponse exportResponse = tsdbAdminClient.exportDatapoints(exportRequest);
```

获取任务信息

示例代码如下：

```
 GetTaskRequest getTaskRequest = new GetTaskRequest();
getTaskRequest.setDatabaseId("tsdb_sdfd434xxxx");
getTaskRequest.setTaskId("taskId");
GetTaskResponse response = tsdbAdminClient.getTask(getTaskRequest);
```

版本说明

v0.10.70

- 新增对offset方式的分页查询支持

v0.10.66

- 新增对升配和续费接口的支持

v0.10.52

- 创建db接口添加查询额度和存储额度字段
- 获取db信息接口返回结果添加查询额度和存储额度字段
- 查询数据点接口返回结果添加consumedCount字段

v0.10.45

- 添加创建删除数据点任务接口

- 添加创建导出数据点任务接口
- 添加获取任务信息接口

v.0.10.32

- 支持使用SQL查询数据

v0.10.24

- 标签过滤支持like匹配
- 按值过滤支持与标签值比较

v0.10.23

- 支持标签过滤
- 支持or条件

v0.10.22

- 支持分页查询
- 支持固定值插值插值函数

v0.10.20

- 支持多域同时查询
- 支持查询时对数据进行插值

v0.10.19

支持对单域的数据的查询、写入

v0.10.17

支持创建/删除/查询时序数据库实例

v0.10.12

- 查询支持按值过滤
- 数据点的值支持byte数组

v0.10.10

- 写入数据点支持gzip压缩
- 支持生成预签名的查询数据点请求

v0.10.7

首次发布。

Node-SDK

概述

本文档主要介绍TSDB Node SDK的安装和使用。在使用本文档前，您需要先了解TSDB的一些基本知识，并已经开通了TSDB服务同时创建了数据库。TSDB JavaScript SDK可以运行在Node.js环境中，开发者可以通过调用API简单便捷地使用TSDB的各项服

务。

- 若您还不了解TSDB，可以参考产品描述和操作指南。
- 请您注册百度智能云账户并登录，在安全认证页面 获取 AK/SK。

安装SDK工具包

版本检测

1. `$ npm -v` // 查看npm是否正确安装
2. `$ node -v` // 检查node版本

支持的Node.js版本

- 4.x
- 5.x

安装SDK工具包

JavaScript包已经上传npm管理器，直接使用npm安装SDK的开发包。

`npm install @baiducloud/sdk`

然后在程序中使用：

```
 import {TsdbDataClient} from '@baiducloud/sdk';
// 需要使用babel转义为 require关键字
// 或者使用 var TsdbDataClient = require('@baiducloud/sdk').TsdbDataClient;

const config = {
    endpoint: <Endpoint>,                // 用户的时序数据库域名，形式如 http://{databaseName}.tsdb.iot.gz.baidubce.com
    credentials: {
        ak: <AccessKeyID>,               //您的AccessKey
        sk: <SecretAccessKey>              //您的SecretAccessKey
    }
};

let client = new TsdbDataClient(config);

client.getMetrics()                          // 调用所需要的的接口
    .then(response => console.log(response))    // 成功
    .catch(error => console.error(error));      // 失败
```

Demo工程下载

为了帮助您更好的使用Node SDK，我们提供Demo工程，点击下载。

快速入门

1. 创建TsdbClient

   TsdbClient是与TSDB服务交互的客户端，TSDB node SDK的TSDB操作都是通过TsdbClient完成的。用户可以参考创建TsdbClient，完成初始化客户端的操作。

2. 写入操作

   构建数据点，包括Metric、Tag和Value，并将数据点写入TSDB。

3. 查询操作

获取Metric、Tag列表，对当前数据进行过滤、分组和筛选。

4. 其它可选操作：

   - 生成查询数据点的预签名URL

     预签名URL可以用于前端页面查询数据点。前端请求服务器生成预签名url并返回给前端，前端使用该URL发起ajax请求查询数据点。

   - 写入数据点的gzip压缩

     写入数据点默认开启gzip压缩。如果不需要使用，可以关闭gzip压缩。

   - 错误码 使用sdk常见错误码以及其原因。

创建TsdbClient

Tsdb sdk 主要主要有两种类型的API，管理接口和数据接口。管理接口主要对数据库进行操作，包括增、删、查看等。数据接口主要是对某个具体的数据库里的数据进行增、删、查看等；用户需要根据具体需求创建不同的client。

基本流程

1. 确定Endpoint。Endpoint是指TSDB服务在各个区域的域名地址。
2. 传入您的AK/SK。
3. 将配置好的config传入TsdbDataClient。

示例代码：

```
 import {TsdbDataClient} from '@baiducloud/sdk';
// 需要使用babel转义为 require关键字
// 或者使用 var TsdbDataClient = require('@baiducloud/sdk').TsdbDataClient;

const config = {
    endpoint: <Endpoint>,           // 用户的时序数据库域名，形式如 http://{databaseName}.tsdb.iot.gz.baidubce.com
    credentials: {
        ak: <AccessKeyID>,          // 用户的Access Key ID
        sk: <SecretAccessKey>       // 用户的Secret Access Key
    }
};

// 初始化一个TsdbClient
const client = new TsdbDataClient(config);
```

通过IP访问

在一些场景下比如正向代理，无法采用域名直接访问tsdb，node sdk也支持通过IP访问

```
 import {TsdbDataClient} from '@baiducloud/sdk';
// 需要使用babel转义为 require关键字
// 或者使用 var TsdbDataClient = require('@baiducloud/sdk').TsdbDataClient;

const config = {
endpoint: <Endpoint>,            // 用户的时序数据库ip+端口号
credentials: {
        ak: <AccessKeyID>,          // 用户的Access Key ID
        sk: <SecretAccessKey>       // 用户的Secret Access Key
    }
};

// 初始化一个TsdbClient
const client = new TsdbDataClient(config, <database_name>);
```

写入操作

写入单域数据点

基本流程

1. 创建TsdbDataClient。

2. 执行writeDatapoints()方法，您需要提供写入的数据的具体信息。

用户可以参考如下代码写入单域数据点：

> 注意：当写入的metric、field、tags、timestamp都相同时，后写入的value会覆盖先写入的value。

```
 // 构建想要写入的datapoints
var datapoints = [
{
        "metric": "cpu_idle",
        "field": "test",
        "tags": {
            "host": "server1",
            "rack": "rack1"
        },
        "type": "Long",
        "timestamp": Math.round(new Date().getTime() / 1000),  // 用于生成时间戳
        "value": 51
    }
];
// 获取并返回结果
client.writeDatapoints(datapoints)
    .then(response => console.log(response))        // 获取成功
    .catch(error => console.error(error));          // 获取失败，并返回错误类型
```

这时，可在对应数据库，点击查询面板，在选项Metrics下出现一个新的metric。

对于同一个field，如果写入了某个数据类型的value之后，相同的field不允许写入其他数据类型。

写入多域数据点

基本流程同写入单域数据点。

不同的域并不需要同时写入，可以认为不同的域都是独立的。但如果查询时要用一条语句查出来，需要保证metric、所有的tag、时间戳都是一致的。

可以参考以下代码写入多域数据点：

```
 // 构建想要写入的datapoints
var datapoints = [
{
       "metric": "cpu_idle3",
       "field" : "field2",
       "tags": {
          "host": "server1",
          "rack": "rack1"
      },
       "type": "Long",
       "timestamp": Math.round(new Date().getTime() / 1000),
       "value": 51
   },
   {
       "metric": "cpu_idle3",
       "field": "field1",
       "tags": {
          "host": "server1",
          "rack": "rack1"
     },
       "type": "Long",
       "values": [
          [Math.round(new Date().getTime() / 1000), 60]   // 用于生成时间戳
       ]
   }
];
// 获取并返回结果
client.writeDatapoints(datapoints)
    .then(response => console.log(response))         // 写入成功
    .catch(error => console.error(error));           // 写入失败，并返回错误类型
```

查询操作

获取度量（Metric）

基本流程

1. 创建TsdbDataClient。

2. 执行getMetrics()方法。

如下代码可以获取metric列表：

```
 // 获取并打印Metric
client.getMetrics()
    .then(response => console.log(response.body))        // 获取成功
    .catch(error => console.error(error));               // 获取失败，并返回错误类型
```

执行结果：

```
 //  终端返回结果
{
 metrics: [
  'cpu_idle',
  'cpu_idle1'
 ]
}
```

获取Field

基本流程

1. 创建TsdbDataClient。

2. 执行getFields()方法，您需要提供查询的metricName。

如下代码可以获取Field列表：

```
 var metricName = <metricName>;
// 获取并打印Field
client.getFields(<metricName>)
    .then(response => console.log(response.body))        // 获取成功
    .catch(error => console.error(error));               // 获取失败，并返回错误类型
```

执行结果：

```
 // 终端返回结果
{
 "fields": [
  "field1",
  "field2"
 ]
 }
```

获取标签（Tag）

基本流程

1. 创建TsdbDataClient。

2. 执行getTags()方法，您需要提供查询的metricName。

如下代码可以获取标签（Tag）列表：

```
 var metricName = <metricName>;
// 获取并打印Tag
client.getTags(<metricName>)
    .then(response => console.log(response.body))        // 获取成功
    .catch(error => console.error(error));               // 获取失败，并返回错误类型
```

执行结果：

```
 // 终端返回结果
{
 tags: [
  'tags1',
  'tags2'
 ]
}
```

查询数据点

单域查询数据点

基本流程

1. 创建TsdbDataClient。

2. 执行getDatapoints()方法，您需要提供根据需求构建的查询列表。

如下代码可以查询数据点：

```
 // 构建想要查询的queryList
var queryList = [
   {
    "metric": "cpu_idle1",
    "filters": {
       "start": "1 hour ago",
       "tags": {
          "host": [
             "server1",
             "server2"
          ]
       },
       "value": ">= 10"
    },
    "groupBy": [
       {
          "name": "Tag",
          "tags": [
             "rack"
          ]
       }
    ]
 }
];

// 获取并打印查询结果
client.getDatapoints(<queryList>)
    .then(response => console.log(JSON.stringify(response.body)))            // 获取成功
    .catch(error => console.error(error));                    // 获取失败，并返回错误类型
```

执行结果：

```
 //  终端返回类似结果
{
 results: [
  {
    metric: 'humidity',
    field: 'value',
    groups: [],
    rawCount: 0
  }
 ]
}
```

多域查询数据点

基本流程

1. 创建TsdbDataClient。

2. 执行getDatapoints()方法，您需要提供根据需求构建的查询列表。

如下代码可以查询数据点：

```
 // 构建想要查询的queryList
var queryList = [
   {
      "metric": "cpu_idle3",
      "fields": [
         "field1",
         "field2"
      ],
      "tags": [
         "rack",
         "host"
      ],
      "filters": {
         "start": "5 hour ago",
         "fields": [
            {
               "field": "field1",
               "value": ">= 10"
            },
            {
               "field": "field2",
               "value": "<= 10"
            }
         ],
         "tags": {
            "rack": [
               "rack1"
            ],
            "host": [
               "server1"
            ]
         },

      },
      "groupBy": [
         {
            "name": "Tag",
            "tags": [
               "rack",
               "host"
            ]
         }
      ],
      "limit": 1000
   }
];
// 获取并打印查询结果
client.getDatapoints(<queryList>)
    .then(response => console.log(JSON.stringify(response.body)))           // 获取成功
    .catch(error => console.error(error));                  // 获取失败，并返回错误类型
```

执行结果：

```
 // 终端返回类似结果
{
 results: [
  {
   metric: 'humidity',
    field: 'value',
   groups: [],
   rawCount: 0
    }
 ]
}
```

使用插值方式查询数据点

基本流程

1. 创建TsdbDataClient。

2. 执行getDatapoints()方法，您需要提供根据需求构建的查询列表。

如下代码可以对数据点进行插值：

```
 // 构建想要查询的queryList
var queryList = [
   {
       "metric": "cpu_idle3",
       "field": "field1",
       "filters": {
          "start": "1 hour ago",
          "tags": {
             "host": [
                "server1"
                ]
             }
        },
        "fill": {
           "type": "Linear",
           "interval": "5 minutes",
           "maxWriteInterval": "30 minutes"
       }
    }
 ];
// 获取并打印查询结果
client.getDatapoints(<queryList>)
    .then(response => console.log(JSON.stringify(response.body)))            // 获取成功
    .catch(error => console.error(error));                  // 获取失败，并返回错误类型
```

执行结果：

```
 //  终端返回类似结果
{
 results: [
  {
   metric: 'humidity',
   field: 'value',
    groups: [],
    rawCount: 0
   }
  ]
}
```

分页查询数据点

基本流程

1. 创建TsdbDataClient。

2. 执行getDatapoints()方法，您需要提供根据需求构建的查询列表。

3. 根据返回结果的result.truncated判断是否还有下一页数据，有则执行2，否则结束。

示例代码：

```
 // 构建想要查询的query
var query = {
    "metric": "cpu_idle1",
    "filters": {
        "start": "1 hour ago",
    }
};
var fetchNext = nextMarker => {
    query.marker = nextMarker;                          // 设置marker，以便获取后面的数据
    client.getDatapoints([query,])                      // 获取数据
        .then(deealWithResponse)                        // 设置处理结果的callback
        .catch(dealWithError);                          // 设置处理error的callback
};
var deealWithResponse = response => {                   // 处理结果
    console.log(JSON.stringify(response.body))          // 打印结果
    if (response.body.results[0].truncated) {           // 后面还有数据
        fetchNext(response.body.results[0].nextMarker);    // 获取下一页
    }
}
var dealWithError = error => console.error(error);      // 处理error

client.getDatapoints([query,])                          // 获取数据
    .then(deealWithResponse)                            // 设置处理结果的callback
    .catch(dealWithError);                              // 设置处理error的callback
```

SQL查询接口

NodeSDK在0.3.1版本中支持了SQL查询接口，支持标准的ANSI SQL语义。

基本流程

1. 创建TsdbDataClient。

2. 执行getRowsWithSql(sql)方法

如下代码可以使用SQL查询数据点：

```
  var sql = 'select * from cpu_idle';
client.getRowsWithSql(sql)
    .then(function (response) {
        console.log(response.body);
    });
```

生成查询数据点的预签名URL

预签名URL可以用于前端页面查询数据点。用法：前端请求服务器生成预签名url并返回给前端，前端使用该URL发起ajax请求查询数据点。

基本流程

1. 创建TsdbDataClient。
2. 执行generatePresignedUrl()方法或generatePresignedUrlWithSql()方法，您需要提供根据需求构建的查询列表或SQL、URL的超时时间、时间戳等。

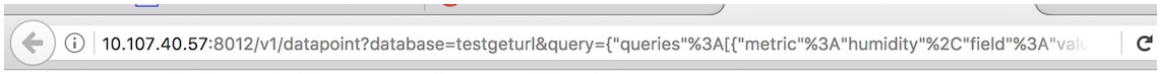
如下代码可以生成查询数据点的预签名URL：

```
 // 构建想要查询的queryList
var queryList = [
   {
    "metric": "cpu_idle3",
    "fields": [
       "field1",
       "field2"
    ],
    "tags": [
       "rack",
       "host"
    ],
    "filters": {
       "start": "5 hour ago",
       "fields": [
          {
             "field": "field1",
             "value": ">= 10"
          },
          {
             "field": "field2",
             "value": "<= 10"
          }
       ],
       "tags": {
          "rack": [
             "rack1"
          ],
          "host": [
             "server1"
          ]
       },

    },
    "groupBy": [
       {
          "name": "Tag",
          "tags": [
             "rack",
             "host"
          ]
       }
    ],
    "limit": 1000
 }
 ];

var url = client.generatePresignedUrl(queryList, 0, 1800, null, {})
console.log(url)
```

执行结果：

```
 //  终端返回类似结果
http://testdb.tsdb.iot.bj.baidubce.com/v1/....
```

可在浏览器里查看数据点



{"results":[{"metric":"humidity","field":"value","groups":[],"rawCount":0}]}

写入数据点的gzip压缩说明

基本流程

1. 创建TsdbDataClient。

2. 执行writeDatapoints()方法，您需要提供要写入的数据点。

写入数据点默认开启gzip压缩。如果不需要使用gzip压缩，可参考如下代码：

```
 var useGzip = false;
// 构建想要查询的queryList
var datapoints = [
{
    "metric": "cpu_idle",
    "field": "test",
    "tags": {
        "host": "server1",
        "rack": "rack1"
    },
    "timestamp": Math.round(newDate().getTime()/1000),
    "value": 51
}
];
// 获取并打印查询结果
client.writeDatapoints(<datapoints>，Gzip)
    .then(response => console.log(response.body))           // 写入成功
    .catch(error => console.error(error));                  // 写入失败，并返回错误类型
```

管理接口

新建TsdbAdminClient

```
 var TsdbAdminClient = require('@baiducloud/sdk').TsdbAdminClient;

const config = {
    endpoint: <Endpoint>,           // 管理接口的域名地址，形如tsdb.{region}.baidubce.com，注意：与新建TsdbClient时使用的endpoint不同。
    credentials: {
        ak: <AccessKeyID>,          // 用户的Access Key ID
        sk: <SecretAccessKey>       // 用户的Secret Access Key
    }
};

// 初始化一个TsdbAdminClient
const client = new TsdbAdminClient(config);
```

创建时序数据库实例

基本流程

1. 创建TsdbAdminClient。

2. 执行createDatabase()方法，您需要设置数据库的各个信息，具体参考API设计文档。

示例代码如下：

```
 // 设置实例各参数
var databaseName = "test" ;                  // 实例的名字
var clientToken = UUID.v4();                 // ClientToken, 用于保证幂等性，重试发送创建请求时，使用同一个clientToken。
var description = 'This is just a test for TSDB.';    // 实例描述，可不填写
var ingestDataPointsMonthly = 1;                    // 写入额度，单位：百万点/月
var storeBytesQuota = 1;                           // 存储空间额度，单位：GB
var purchaseLength = 1;                            // 购买时长，单位：月
var couponName = <your-coupon-name>;                    // 代金券号，可不填写

// 创建并返回创建结果
client.createDatabase(clientToken, databaseName, ingestDataPointsMonthly, purchaseLength,
description，storeBytesQuota)
   .then(response => console.log(response.body))       // 创建成功
   .catch(error => console.error(error));              // 创建失败，并返回错误类型
```

返回值所包含字段请参考API文档。

删除时序数据库实例

基本流程

1. 创建TsdbAdminClient。

2. 执行deleteDatabase()方法，您需要提供将要删除的database的databaseId。

示例代码如下：

```
 // 删除实例的ID
var databaseId = 'tsdb-dvb9we5yfkcc';

// 删除实例并返回结果
client.deleteDatabase(databaseId)
    .then(response => console.log(response))        // 删除成功
    .catch(error => console.error(error));                // 删除失败，并返回错误类型
```

注意，只允许删除到期的时序数据库实例，否则将报错。

执行结果：

```
 //  删除失败（一般为数据库未到期）
{
  status_code: 400,
  message: 'Can not delete unexpired database',
  code: 'DeleteUnexpiredDatabaseFailed',
  request_id: '2a92ae76-87a2-432f-92a9-a426d2b447b6'
 }
// 删除成功（数据库到期）
  body: {}
```

获取时序数据库实例

基本流程

1. 创建TsdbAdminClient。

2. 执行getDatabaseInfo()方法，您需要提供将要获取的database的databaseId。

示例代码如下：

```
 // 获取实例的ID
var databaseId = 'tsdb-dvb9we5yfkcc';

// 获取实例并返回实例信息
client.getDatabaseInfo(databaseId)
    .then(response => console.log(response.body))          // 获取成功，返回信息列表
    .catch(error => console.error(error));                 // 获取失败，并返回错误类型
```

执行结果：

```
 //  终端返回类似结果
{
    databaseId: 'tsdb-tfu33g88m658',
    databaseName: 'testgeturl',
    description: '',
    endpoint: 'testgeturl.tsdb.iot.gz.baidubce.com',
    quota: {
        ingestDataPointsMonthly: 1
    },
    status: 'Active',
    autoExport: false,
    createTime: '2017-10-11T08:51:09Z',
    expiredTime: '2017-11-11T08:51:09Z'
}
```

获取时序数据库实例列表

基本流程

1. 创建TsdbAdminClient。

2. 执行listDatabase()方法。

示例代码如下：

```
 // 获取并返回结果
client.listDatabase()
    .then(response => console.log(response.body))          // 获取成功，返回包含每个数据库的信息的列表
    .catch(error => console.error(error));                 // 删除失败，并返回错误类型
```

执行结果：

```
 // 终端返回类似结果，会打印所有database
[
   {
      "databaseId": "tsdb-dvb9we5yfkcc",
      "databaseName": "viztest",
      "description": "Ownedby物可视,Don\'tremove",
      "endpoint": "viztest.tsdb.iot.gz.baidubce.com",
      "quota": {
         "ingestDataPointsMonthly": 100
      },
      "status": "Active",
      "autoExport": "false",
      "createTime": "2017-06-14T07:41:46Z",
      "expiredTime": "2018-06-14T07:41:47Z"
   },
   {
      "databaseId": "tsdb-n88psnkq965c",
      "databaseName": "vizyingyan",
      "description": "物可视地图测试数据",
      "endpoint": "vizyingyan.tsdb.iot.gz.baidubce.com",
      "quota": {
         "ingestDataPointsMonthly": 10
      },
      "status": "Active",
      "autoExport": "false",
      "createTime": "2017-06-20T04:01:03Z",
      "expiredTime": "2018-06-20T04:01:03Z"
   }
]
```

错误码

错误码	错误码信息	可能原因
400	InvalidArgument	查询参数或查询list不对
404	not found	database名称是否定义并正确使用 或者 endpoint 设置不正确

版本说明

v1.0.0-rc.0

- 版本号：1.0.0-rc.0
- 时间：2018-08-02
- 增加SQL查询接口，SDK包名统一更新为'@baiducloud/sdk'

Python-SDK

概述

本文档主要介绍TSDB Python SDK的安装和使用。在使用本文档前，您需要先了解TSDB的一些基本知识，并已经开通了TSDB服务同时创建了数据库。若您还不了解TSDB，可以参考产品描述和操作指南。

安装SDK工具包

环境准备

1. 运行环境

   Python SDK工具包支持在Python 2.7 以上环境运行。

2. 安装pycrypto依赖

   安装SDK之前，需要先执行命令`pip install pycrypto`安装pycrypto依赖。
   如果安装失败，请执行`pip install pycryptodome`

下载和安装

方式一：通过pip安装

您可以通过pip安装的方式将百度智能云Python SDK安装到您的环境中。 联网状态下，在命令行中执行如下命令：

```
pip install bce-python-sdk
```

即可将Python SDK安装到本地。 **方式二：将源码包下载到本地后进行安装**

1. 在开发者资源中心下载Python SDK压缩工具包。
2. 命令行移动到压缩包所在路径，执行如下命令（version替换为包名称中的版本号）：

   ```
   pip install bce-python-sdk-version.zip
   ```

   即可将Python SDK安装到本地。

您也可以解压压缩包后执行如下命令（version替换为包名称中的版本号）

```
cd bce-python-sdk-version

python setup.py install
```

Demo工程下载

为了帮助您更好的使用Python SDK，我们提供Demo工程，点击下载。

创建TsdbClient

TSDB的数据接口在Python SDK中通过TsdbClient对象来访问，用户可以参考如下代码新建一个TsdbClient：

```
 import baidubce.protocol
from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.tsdb.tsdb_client import TsdbClient
# when use https as the protocol, you may find certificate expire problem, this can be resovled by adding the following
lines
# import ssl
# ssl._create_default_https_context = ssl._create_unverified_context

##########必填配置#############
HOST = 'Fill host here' # 用户的时序数据库域名，形式如databasename.tsdb.iot.gz.baidubce.com
AK = 'Fill AK here'     # 用户的百度智能云 Access Key ID
SK = 'Fill SK here'     # 用户的百度智能云 Secret Access Key

##########可选配置#############
#使用HTTP协议
protocol=baidubce.protocol.HTTP
#使用HTTPS协议
# protocol= baidubce.protocol.HTTPS
connection_timeout_in_mills=None  #连接超时时间
send_buf_size=None                #发送缓冲区大小
recv_buf_size=None                #接收缓冲区大小
retry_policy=None                 #重试策略

#生成config对象
config = BceClientConfiguration(
        credentials=BceCredentials(AK, SK),
        endpoint=HOST,
        protocol=protocol,
        connection_timeout_in_mills=connection_timeout_in_mills,
        send_buf_size=send_buf_size,
        recv_buf_size=recv_buf_size,
        retry_policy=retry_policy)

#创建TsdbCient
tsdb_client = TsdbClient(config)
```

通过IP访问

在一些场景下比如正向代理，无法采用域名直接访问tsdb，python sdk也支持通过IP访问

```
 ##########必填配置#############
# HOST = '<ip>:<port>'
# AK = '<your ak>'
# SK = '<your sk>'
# DATABASE = '<database_name>'

########生成config对象########
config = BceClientConfiguration(
    credentials=BceCredentials(AK, SK),
    endpoint=HOST,
    protocol=protocol,
    connection_timeout_in_mills=connection_timeout_in_mills,
    send_buf_size=send_buf_size,
    recv_buf_size=recv_buf_size,
    retry_policy=retry_policy)

########创建TsdbCient########
tsdb_client = TsdbClient(config,DATABASE)
```

写入操作

写入单域数据点

用户可参考如下代码写入单域数据点：

> 注意：当写入的metric、field、tags、timestamp都相同时，后写入的value会覆盖先写入的value。

```
 #构建要写入的数据点数据点对象数组
datapoints = [{
    "metric": "wind",
    "tags": {
        "city": "ShangHai"
    },
    "field": "direction",  #域名，缺少此field字段的话默认为value
    "timestamp": 1531985379000,
    "type": "Long",
    "value": 1
}]
try:
    print tsdb_client.write_datapoints(datapoints)  #写入数据点
except BaseException as e:
    print e  #通过捕获异常获取详细的错误信息
```

对于同一个field，如果写入了某个数据类型的value之后，相同的field不允许写入其他数据类型。

返回结果：

```
 {metadata:{keep_alive:'timeout=10',server:'BWS',connection:'keep-alive',pragma:'no-cache',date:'Wed, 01 Aug 2018 12:41:28 GMT',bce_request_id:'d46b3f74-76c4-4e1a-be47-39354b7b81a3',cache_control:'no-cache'}}
```

写入多域数据点

多域数据点是指同一个metric下可以有多个不同名字的field，不同field的数据类型可以不同。

不同的域并不需要同时写入，可以认为不同的域都是独立的。但如果查询时要用一条语句查出来，需要保证metric、所有的tag、时间戳都是一致的。

```
 #构建要写入的数据点数据点对象数组
datapoints = [{
    "metric": "wind",
    "tags": {
        "city": "ShangHai"
    },
    "field": "direction",
    "timestamp": 1531985380000,
    "type": "Long",
    "value": 1
}, {
    "metric": "wind",
    "tags": {
        "city": "ShangHai"
    },
    "field": "speed",
    "timestamp": 1531985380000,
    "type": "Double",
    "value": 4.5
}]
try:
    tsdb_client.write_datapoints(datapoints)  #写入数据点
    print 'single filed datapoint write success'
except BaseException as e:
    print e  #通过捕获异常获取详细的错误信息
```

查询操作

获取度量(Metric)

如下代码可以获取metric列表：

```
 # 获取metric列表
result = tsdb_client.get_metrics()
print result.metrics
```

返回结果：

```
 [u'wind']
```

获取域(Field)

如下代码可以获取field对象：

```
 metric = 'wind' # 要获取field所属的metric
result = tsdb_client.get_fields(metric) # 获取指定metric下的所有field
print result.fields
```

返回结果：

```
 {direction:{type:u'Number'},speed:{type:u'Number'}}
```

获取标签(Tag)

如下代码可以获取tag对象：

```
 metric = 'wind' # 要获取tag所属的metric
result = tsdb_client.get_tags(metric)
print result.tags
```

返回结果：

```
 {city:[u'ShangHai']}
```

查询数据点

查询单域数据点

如下代码可以查询单域数据点：

```
 #构建查询对象
query_list = [{
    "metric": "wind",
    "field": "direction",
    "filters": {
        "start": 1531985370000,
        "end": 1531985400000,
        "tags": {
            "city": ["ShangHai"]
        },
        "value": ">= 0"
    },
    "groupBy": [{
        "name": "Tag",
        "tags": ["city"]
    }],
    "limit": 1000,
    "aggregators": [{
        "name": "Sum",
        "sampling": "10 minutes"
    }]
}]
result = tsdb_client.get_datapoints(query_list)
print result.results
```

返回结果：

```
 [{field:u'direction',metric:u'wind',groups:[{group_infos:[{name:u'Tag',tags:{city:u'ShangHai'}}],values:[[1531985379000,
2]]}],rawcount:2}]
```

查询多域数据点

如下代码可以查询多域数据点：

```
 #构建查询对象
query_list = [{
    "metric": "wind",
    "fields": ["direction", "speed"],  #查询多个域
    "filters": {
        "start": 1531985370000,
        "end": 1531985400000,
        "tags": {
            "city": ["ShangHai"]
        },
        "value": ">= 0"
    },
    "groupBy": [{
        "name": "Tag",
        "tags": ["city"]
    }],
    "limit": 1000,
    "aggregators": [{
        "name": "Sum",
        "sampling": "10 minutes"
    }]
}]
result = tsdb_client.get_datapoints(query_list)
print result.results
```

返回结果：

```
 [{fields:[u'direction', u'speed'],metric:u'wind',groups:[{group_infos:[{name:u'Tag',tags:{city:u'ShangHai'}}],values:
[[1531985380000, 1, 4.5]]}],rawcount:1}]
```

使用插值方式查询数据

使用如下代码进行插值查询：

```
 # 构建查询对象
query_list = [{
    "metric": "wind",
    "fields": ["direction", "speed"],
    "filters": {
        "start": 1531985370000,
        "end": 1532985370000,
        "tags": {
            "city": ["ShangHai"]
        },
        "value": ">= 0"
    },
    "fill": {                          #配置插值参数
        "type": "Linear",                 #插值类型
        "interval": "1 day",              #插值间隔
        "maxWriteInterval": "30 minutes"  #最大写入间隔
    },
    "groupBy": [{
        "name": "Tag",
        "tags": ["city"]
    }],
    "limit": 1000,
    "aggregators": [{
        "name": "Sum",
        "sampling": "10 minutes"
    }]
}]

result = tsdb_client.get_datapoints(query_list)
print result.results
```

返回结果：

```
 [{fields:[u'direction', u'speed'],metric:u'wind',groups:[{group_infos:[{name:u'Tag',tags:{city:u'ShangHai'}}],values:
[[1531985380000, 1, 4.5], [1532071780000, 1, 4.5], [1532158180000, 1, 4.5], [1532244580000, 1, 4.5],
[1532330980000, 1, 4.5], [1532417380000, 1, 4.5], [1532503780000, 1, 4.5], [1532590180000, 1, 4.5],
[1532676580000, 1, 4.5], [1532762980000, 1, 4.5], [1532849380000, 1, 4.5], [1532935780000, 1,
4.5]]}],rawcount:12}]
```

分页查询数据点

如果查询到的原始数据点过大，会分多次返回，使用marker参数来进行分页查询。

```
 # 通过循环来多次查询数据

query_list = [{
    "metric": "wind",
    "field": "direction",
    "filters": {
        "start": 1531985300000,
        "end": 1531985400000,
        "tags": {
            "city": ["ShangHai"]
        }
    },
    "limit": 1
}]
count = 0
while True:
    count += 1
    if len(query_list) >0:
        result = tsdb_client.get_datapoints(query_list)
        print count, result.results
    else:
        print 'end query'
        break
    next_query = []
    for i in range(len(query_list)):
        if result.results[i].truncated:
            query_list[i]['marker'] = result.results[i].next_marker
            next_query.append(query_list[i])
    query_list = next_query
```

返回结果：

```
 1 [{field:u'direction',next_marker:u'AAABZLFxwqBjaXR5PVNoYW5nSGFp',groups:[{group_infos:[],values:
[[1531985379000, 1]]}],truncated:True,metric:u'wind',rawcount:1}]
2 [{field:u'direction',metric:u'wind',groups:[{group_infos:[],values:[[1531985380000, 1]]}],rawcount:1}]
end query
```

使用SQL查询数据点

TSDB支持使用SQL进行数据查询，示例代码如下：

```
 sql = "select * from wind"
result = tsdb_client.get_rows_with_sql(sql)
print result
```

返回结果：

```
 {rows:[[1531985379000, 1.0, None, u'ShangHai'], [1531985380000, 1.0, 4.5, u'ShangHai']],columns:
[{name:u'timestamp'}, {name:u'direction'}, {name:u'speed'}, {name:u'city'}],metadata:
{content_length:'168',content_type:'application/json; charset=UTF-
8',keep_alive:'timeout=10',server:'BWS',connection:'keep-alive',pragma:'no-cache',date:'Thu, 02 Aug 2018 00:35:28
GMT',bce_request_id:'46f5d19e-58de-4654-bc35-91bb67b58e76',cache_control:'no-cache'}}
```

生成查询数据点的预签名URL

预签名URL可以用于前端页面查询数据点用法：前端请求服务器生成预签名url并返回给前端，前端使用该URL发起ajax请求查询数据点。

生成基于Query对象的预签名URL

```
 timestamp = int(time.time())  #指定计算签名的时间戳
expiration_in_seconds = 1800  #指定url过期时间
per_signed_url = tsdb_client.generate_pre_signed_url(query_list,
      timestamp=timestamp, expiration_in_seconds=expiration_in_seconds)  #生成预签名的URL
print per_signed_url
```

返回结果：

```
 http://databasename.tsdb.iot.bj.baidubce.com/v1/datapoint?authorization=bce-auth-
v1%2Fa85ed2d7649141e08bf79fbbd88edc12%2F2018-07-
23T10%3A06%3A46Z%2F1800%2F%2Fbe601040db0e5aa8f959248651b5ea7373c5d22fbece0d34eb21eb5cc8763f02&di
D
```

生成基于SQL的预签名URL

可以参考如下代码：

```
 sql = "select * from wind"
timestamp = int(time.time())
per_signed_url_sql = tsdb_client.generate_pre_signed_url_with_sql(sql,
      timestamp=timestamp, expiration_in_seconds=1800)
print per_signed_url_sql
```

返回结果：

```
 http://databasename.tsdb.iot.bj.baidubce.com/v1/row?authorization=bce-auth-
v1%2Fa85ed2d7649141e08bf79fbbd88edc12%2F2018-07-
23T10%3A26%3A33Z%2F1800%2F%2F73c721bff21300c17d8ec4e1fab94b6cbf91672c33417276808c534c85163b1d&sc
d
```

写入数据点的Gzip压缩说明

写入数据点默认开启Gzip压缩，如果想要关闭此功能，可以在调用`write_datapoints`方法时，传入参数 `use_gzip=False`。

可以参考如下代码：

```
 response = tsdb_client.write_datapoints(datapoints, use_gzip=False)
```

管理接口

TSDB数据库的管理操作通过管理接口进行调用，包括创建数据库、删除数据库和获取数据库信息。在Python SDK中使用TsdbAdminClient类实现对管理接口的封装。

新建TsdbAdminClient

```
 import baidubce.protocol
from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.tsdb.tsdb_admin_client import TsdbAdminClient
# when use https as the protocol, you may find certificate expire problem, this can be resovled by adding the following lines
# import ssl
# ssl._create_default_https_context = ssl._create_unverified_context

HOST = 'tsdb.<region>.baidubce.com'
AK = '<your ak>'
SK = '<your sk>'

############optional config##############
protocol=baidubce.protocol.HTTP
# protocol= baidubce.protocol.HTTPS
connection_timeout_in_mills=None
send_buf_size=None
recv_buf_size=None
retry_policy=None
########################################



config = BceClientConfiguration(
        credentials=BceCredentials(AK, SK),
        endpoint=HOST,
        protocol=protocol,
        connection_timeout_in_mills=connection_timeout_in_mills,
        send_buf_size=send_buf_size,
        recv_buf_size=recv_buf_size,
        retry_policy=retry_policy)

tsdb_admin_client = TsdbAdminClient(config)
```

创建时序数据库

使用如下代码创建数据库：

```
 # 创建数据库
database_name = 'pythonsdksample'  # 数据库名称
description = 'description'        # 数据库描述。可选
ingest_datapoints_monthly = 1      # 月写入数据点数额度，单位，点/百万
store_bytes_quota = 0              # 字节存储空间额度，单位，byte。 可选，默认为0
purchase_length = 1                # 购买时长，单位，月
coupon_name = ''                   # 代金券编号。可选，如填写，则默认优先使用代金券付款

client_token = 'testtttt'          # 任意指定字符串，用于保证创建的幂等性。即，不会因网络重试等问题导致重复创建
try:
    # use str.encode("UTF-8") in python 3.0+, since str.decode is not supported in unicode,
    # you need to make changes in description, database_name and coupon_name.
    result = tsdb_admin_client.create_database(
            client_token=client_token,
            description=description,
            database_name=database_name,
            ingest_datapoints_monthly=ingest_datapoints_monthly,
            purchase_length=purchase_length,
            store_bytes_quota=store_bytes_quota,
            coupon_name=coupon_name)
    print result                    # 创建成功会返回订单号、数据库id、花费金额、到期时间的信息
except BaseException as e:
    print e                         # 如果创建失败，会抛出异常，异常中含有错误的详细信息
```

删除时序数据库

使用如下代码删除数据库：

```
 # 删除指定数据库
database_id = 'tsdb-xxxxxx'  # database_id 可以官网控制台数据库的详细页查看
try:
    response = tsdb_admin_client.delete_database(database_id)
    print response
except BaseException as e:
    print e                 # 删除失败会在异常中包含详细的错误信息
```

获取时序数据库实例

使用如下代码获取时序数据库实例：

```
 # 获取数据库实例
database_id = 'tsdb-xxxxxx'
print tsdb_admin_client.get_database(database_id)
```

返回结果：

```
 {'status': u'Active', 'autoExport': False, 'databaseId': u'tsdb-xxxxx', 'endpoint': u'pythonsdksample.tsdb-xxxxx.tsdb.iot.bj.baidubce.com', 'createTime': u'2018-07-20T08:57:53Z', 'description': u'', 'databaseName': u'pythonsdksample', 'quota': {ingestDataPointsMonthly:1,storeBytesQuota:0}, 'expiredTime': u'2018-08-20T08:57:54Z'}
```

获取时序数据库实例列表

使用如下代码获取时序数据库实例列表：

```
 # 获取数据库实例列表
result = tsdb_admin_client.get_all_databases()
print result
```

返回结果：

```
 [{'status': u'Active', 'autoExport': False, 'databaseId': u'tsdb-xxxxx', 'endpoint': u'pythonsdksample.tsdb-xxxxx.tsdb.iot.bj.baidubce.com', 'createTime': u'2018-07-20T08:57:53Z', 'description': u'', 'databaseName': u'pythonsdksample', 'quota': {ingestDataPointsMonthly:1,storeBytesQuota:0}, 'expiredTime': u'2018-08-20T08:57:54Z'}, {...},..
```

版本说明

V0.8.23

- 版本号：0.8.23
- 时间：2018-08-21
- 修复返回结果中属性名称被自动转化为python风格导致无法获取原始值的问题。

V0.8.22

- 版本号：0.8.22
- 时间：2018-08-02
- 首次发布。支持写入、查询数据以及管理数据库。支持SQL接口。

API参考

介绍

名词解释

在使用TSDB API前，用户需了解TSDB相关的核心概念。

调用方式

概述

TSDB API的设计采用了Restful风格，每个API功能（也可以称之为资源）都使用URI（Universal Resource Identifier）来唯一确定。对资源的请求方式是通过向资源对应的URI发送标准的 HTTP 请求，比如 GET、PUT、POST等，同时，请求需要遵守签名算法，并包含约定的请求参数。

通用约定

- 所有编码都采用UTF-8
- 日期格式采用yyyy-MM-dd方式，如2015-08-10
- 时间格式采用UTC格式：yyyy-MM-ddTHH:mm:ssZ, 如2015-08-20T01:24:32Z
- Content-type为application/json; charset=UTF-8
    - object类型的key必须使用双引号（"）括起来
    - object类型的key必须使用lowerCamelCase表示

公共请求头

头域（Header）	是否必须	说明
Authorization	必须	包含Access Key与请求签名
Host	必须	包含API的域名
x-bce-date	必须	表示时间的字符串，符合时间格式要求
Content-Type	可选	application/json; charset=utf-8

公共响应头

头域（Header）	说明
Content-Type	只支持JSON格式，application/json; charset=utf-8
x-bce-request-id	TSDB后端生成，并自动设置到响应头域中

响应状态码

返回的响应状态码遵循RFC 2616 section 6.1.1

- 1xx: Informational - Request received, continuing process.
- 2xx: Success - The action was successfully received, understood, and accepted.
- 3xx: Redirection - Further action must be taken in order to complete the request.
- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled.
- 5xx: Server Error - The server failed to fulfill an apparently valid request.

请求消息体格式（HTTP Request Body）

TSDB服务要求使用JSON格式的结构体来描述一个请求的具体内容。

示例

以下是一个标准的写入data point时的请求消息体格式：

```
 {
   "datapoints": [{
      "metric": "cpu_idle",
      "tags": {
         "host": "server1",
         "rack": "rack1"
      },
      "timestamp": 1465376157007,
      "value": 51
   }]
}
```

请求返回格式（HTTP Response）

TSDB服务均采用JSON格式的消息体作为响应返回的格式。

示例

以下是一个标准的获取metric列表的请求返回：

```
{
  "metrics": [
     "cpu_idle",
     "mem_used"
  ]
}
```

通用错误返回格式

当调用接口出错时，将返回通用的错误格式。Http的返回状态码为4xx或5xx，返回的消息体将包括全局唯一的请求、错误代码以及错误信息。调用方可根据错误码以及错误信息定位问题，当无法定位到错误原因时，可以发工单联系百度技术人员，并提供requestid以便于快速地帮助您解决问题。

消息体定义

参数名	类型	说明
requestId	String	请求的唯一标识
code	String	错误类型代码
message	String	错误的信息说明

错误返回示例

```
{
  "requestId": "47e0ef1a-9bf2-11e1-9279-0100e8cf109a",
  "code":"NoSuchKey",
  "message":"The resource you requested does not exist"
}
```

签名认证

TSDB API会对每个访问的请求进行身份认证，以保障用户的安全。安全认证采用Access Key与请求签名机制。Access Key由Access Key ID和Secret Access Key组成，均为字符串，由百度智能云官方颁发给用户。其中Access Key ID用于标识用户身份，Access Key Secret 是用于加密签名字符串和服务器端验证签名字符串的密钥，必须严格保密。

对于每个HTTP请求，用户需要使用下文所描述的方式生成一个签名字符串，并将认证字符串放在HTTP请求的Authorization头域里。

签名字符串格式

bce-auth-v{version}/{accessKeyId}/{timestamp}/{expireTime}/{signedHeaders}/{signature}

其中：

- version是正整数，目前取值为1。
- timestamp是生成签名时的时间。时间格式符合通用约定。
- expireTime表示签名有效期限，单位为秒，从timestamp所指定的时间开始计算。
- signedHeaders是签名算法中涉及到的头域列表。头域名字之间用分号（;）分隔，如host;x-bce-date。列表按照字典序排列。当signedHeaders为空时表示取默认值。
- signature是256位签名的十六进制表示，由64个小写字母组成，生成方式由如下签名生成算法给出。

签名生成算法

有关签名生成算法的具体介绍，请参看鉴权认证机制。

多区域选择

Region代表着一个独立的地域，是百度智能云中的重要概念，请参考区域选择说明。百度智能云中的服务除了极少数如账号服务全局有效之外，绝大部分服务都是区域间隔离的。每个区域的服务独立部署互不影响。服务间共享数据需要通过显式拷贝完成。在API中引用区域必须使用其ID。目前TSDB支持http和https调用。

服务域名

数据API接口服务域名，其中{database-name}为数据库名称。

区域	ID	域名	协议
华北-北京	bj	{database-name}.{database-id}.tsdb.iot.bj.baidubce.com	http和https
华南-广州	gz	{database-name}.{database-id}.tsdb.iot.gz.baidubce.com	http和https

管理API接口服务域名为

区域	ID	域名	协议
华北-北京	bj	tsdb.bj.baidubce.com	http和https
华南-广州	gz	tsdb.gz.baidubce.com	http和https

通过IP访问数据API

通过上面服务域名可以了解到数据API接口是通过域名的前缀来区分不同的时序数据库的。举个数据点写入的例子，头信息如下：

```
 POST /v1/datapoint HTTP/1.1
HOST: {database-name}.tsdb.iot.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-08T16:49:51Z
```

而用IP来访问时序数据库服务，是使用url的parameter来区分不同的时序数据库的，格式是database={database-name}。同样是数据点写入的例子，头信息如下：

```
 POST /v1/datapoint?database={database-name} HTTP/1.1
HOST: {ip}
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-08T16:49:51Z
```

数据API接口说明

写入data point

方法	API	说明
POST	/v1/datapoint	写入data point

请求参数

参数名称	参数类型	是否必须	说明
metric	String	必须	metric的名称
field	String	可选	field的名称，默认名称为value。不同的field支持不同的数据类型写入。对于同一个field，如果写入了某个数据类型的value之后，相同的field不允许写入其他数据类型
tags	Object	必须	data point对应的所有tag，Object中的一对key-value表示一个tag的key-value
type	String	必须	目前支持Long/Double/String/Bytes/BigDecimal。代表value字段的类型。bytes是种特殊类型，表示value是经过base64编码后的String，TSDB存储时会反编码成byte数组存储
timestamp	Int	可选	Unix时间戳，单位是毫秒；如果timestamp为空，value不为空，timestamp自动填入系统当前时间；如果timestamp的位数小于等于10位，将认为精度是秒，自动乘以1000；timestamp+value与values两者必须二选一
value	Int/Double/String	可选	data point的值，timestamp+value与values两者必须二选一。当写入的metric、field、tags、timestamp都相同时，后写入的value会覆盖先写入的value
values	List<List<Any>>	可选	对于相同的metric+tags的data point，可以通过合并成一个values的List来减少payload，values是个二维数组，里面的一维必须是两个元素，第一个元素是timestamp，是unix时间戳，类型是Int，第二个元素是value，类型是Int/Double/String；如果timestamp的位数小于等于10位，将认为精度是秒，自动乘以1000

请求示例 HOST中{database}字段表示数据库名称

```
 POST /v1/datapoint HTTP/1.1
HOST: {database}.tsdb.iot.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-08T16:49:51Z

{
    "datapoints": [{
        "metric": "cpu_idle",
        "tags": {
            "host": "server1",
            "rack": "rack1"
        },
        "type": "Long",
        "timestamp": 1465376157007,
        "value": 51
    }, {
        "metric": "cpu_idle",
        "tags": {
            "host": "server2",
            "rack": "rack2"
        },
        "type": "Long",
        "values": [
            [1465376269769, 67],
            [1465376325057, 60]
        ]
    }]
}
```

返回示例

```
 HTTP/1.1 204 No Content
x-bce-request-id: 72492aee-1470-46d0-8a4d-0dab7b8e67b7
```

获取metric列表

方法	API	说明
GET	/v1/metric	获取database中所有的metric的列表

返回参数

参数名称	参数类型	说明
metrics	List<String>	metric的列表

请求示例

```
 GET /v1/metric HTTP/1.1
Host: {database}.tsdb.iot.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-08T16:49:51Z
```

返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 72492aee-1470-46d0-8a4d-0dab7b8e67b7

{
    "metrics": [
        "cpu_idle",
        "mem_used"
    ]
}
```

获取field列表

方法	API	说明
GET	/v1/metric/{metric}/field	获取database中metric的field列表

返回参数

参数名称	参数类型	说明
fields	Object	field列表，Object的每个key对应一个field的key，Object的每个value都是Object类型的，表示该field的类型

请求示例

```
 GET /v1/metric/wind/field HTTP/1.1
Host: {database}.tsdb.iot.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-08T16:49:51Z
```

返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 72492aee-1470-46d0-8a4d-0dab7b8e67b7

{
    "fields": {
        "power": {
            "type": "String"
            },
        "direction": {
            "type": "Number"
        }
    }
}
```

获取tag列表

方法	API	说明
GET	/v1/metric/{metric}/tag?start=0&end=1562573168000	获取metric对应的所有tag的key和value列表

请求参数

参数名称	参数类型	是否必须	说明
metric	String	必须	metric的名称
start	Int	可选	查询一段时间的所有tag，此为起始时间，默认为0
end	Int	可选	查询一段时间的所有tag，此为结束时间，默认为2^{63} - 1

返回参数

参数名称	参数类型	说明
tags	Object	tag列表，Object的每个key对应一个tag的key，Object的每个value都是List<String>类型的，表示该tag的所有value的列表

请求示例

```
 GET /v1/metric/cpu_idle/tag HTTP/1.1
Host: {database}.tsdb.iot.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-08T16:49:51Z
```

返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 72492aee-1470-46d0-8a4d-0dab7b8e67b7

{
    "tags": {
        "host": ["server1", "server2"],
        "rack": ["rack1", "rack2"]
    }
}
```

查询data point

> **注意：**
>
> 若单次查询时间超过50s，系统将自动终止本次查询。如果出现查询超时，可以采取以下措施优化查询： 1. 减少单次请求中query个数 2. 缩短单个query中起止时间的间隔 3. 减少单个query涉及到的时间序列数目 4. 对数据进行预处理

方法	API	说明
GET	/v1/datapoint?query={json}	查询data point，查询参数编码在query参数中
PUT	/v1/datapoint?query	查询data point，查询参数在body中

请求参数

参数名称	参数类型	是否必须	说明
queries	List<Query>	必须	查询条件列表，由Query对象组成的数组
disablePresampling	Boolean	可选	是否禁用预处理结果查询，默认false

Query对象

参数名称	参数类型	是否必须	说明
metric	String	必须	需要查询的metric的名称
field	String	可选	需要查询的metric的field名称，默认名称为value
fields	List<String>	可选	需要查询的metric的field的列表。fields和field冲突，不能同时存在。
tags	List<String>	可选	需要查询的metric的tag的key列表
filters	Object	必须	过滤条件，类型为Filters
groupBy	List<GroupBy>	可选	分组条件，由GroupBy对象组成的数组
offset	Int	可选	数据分页，查询返回数据点的偏移量，不填默认为0
limit	Int	可选	返回的data point数目限制，不填时默认为1,000,000
aggregators	List<Aggregator>	可选	聚合条件，由Aggregator对象组成的数组
order	String	可选	支持Asc和Desc，默认是Asc
fill	Object	可选	插值选项，类型为Fill，插值只作用于原始数据，每个时间序列单独进行插值
fills	List<Fill>	可选	多个插值选项，由Fill对象组成的数组。fills和fill冲突，不能同时存在。如果存在对于同一个域有多个插值，那么对于这个域只有第一个插值生效，譬如第一个fill是对全部插值，第二个fill是对a进行插值，那么对于a只有第一个fill生效
marker	String	可选	用于分页查询，从marker开始返回，应使用上一次查询返回的nextMarker值

Filters对象

参数名称	参数类型	是否必须	说明
start	Int/String	必须	起始时间，可以是绝对时间（即时间戳，类型为Int，单位为毫秒），也可以是相对时间（类型为String，如"5 days ago"）。对于绝对时间，如果位数少于等于10位，将认为精度是秒，自动乘以1000。 注意：包含or时，最外层start不可填写，单个filter内部start为必须。
end	Int/String	可选	结束时间，可以是绝对时间或者是相对时间，默认为$2^{63}-1$。对于绝对时间，如果位数少于等于10位，将认为精度是秒，自动乘以1000
tags	Object	可选	可以是一个Object，Object的每个key对应一个tag的key，Object的每个value都是List<String>类型的，表示该tag的需要匹配的value的列表 可以是TagFilter对象组成的列表
value	String	可选	单field查询的值过滤，分为符号和值两部分 符号支持=，!=，>，<，>=和<= 值允许Number（包括long和double）、String和Tag，其中String需包含于单引号中；Tag为tag的key，不加单引号，过滤时会被自动解析为对应的类型（譬如field的类型是Number，该tag的value会被解析为Number） 例如："> 111" 或 "< 11.1" 或 "= 'abc'" 或 "> threshold"
fields	List<FieldFilter>	可选	多field查询的值过滤，由FieldFilter对象组成的数组，fields和value冲突，不能同时存在。
or	List<Filters>	可选	或filter查询条件，or和其他查询条件冲突，不能同时存在。

FieldFilter对象

参数名称	参数类型	是否必须	说明
field	String	必须	需要过滤的metric的field名称
value	String	必须	多field查询的值过滤，分为符号和值两部分 符号支持=，!=，>，<，>=和<= 值允许Number（包括long和double）、String和Tag，其中String需包含于单引号中；Tag为tag的key，不加单引号，过滤时会被自动解析为对应的类型（譬如field的类型是Number，该tag的value会被解析为Number） 例如："> 111" 或 "< 11.1" 或 "= 'abc'" 或 "> threshold"

TagFilter对象

参数名称	参数类型	是否必须	说明
tag	String	必须	需要过滤的metric的tag的key
in	List<String>	可选	可以包含的tag的value列表，即tag的value只需要是in中的任何一个值。不可与notIn和like同时存在
notIn	List<String>	可选	需要过滤的tag的value列表，即tag的value不能是notIn中的任何一个值。不可与in和like同时存在
like	String	可选	需要匹配的规则，“%”匹配任意数量的字符，“_”匹配单个字符，转义字符为“/”；不可与in和notIn同时存在

GroupBy对象

参数名称	参数类型	是否必须	说明
name	String	必须	分组方式，目前仅支持Tag
tags	List<String>	可选	按照哪些tag进行分组，name为Tag时必填

Aggregator对象

参数名称	参数类型	是否必须	说明
name	String	必须	聚合的方式，目前支持Avg、Dev、Count、First、Last、LeastSquares、Max、Min、Percentile、Sum、Diff、Div、Scale、Rate、AdjacentUnique
sampling	String	可选	采样的时间长度，如"10 minutes"。支持自然日历对齐的查询，时间单位需增加字符“c”，如“1 hc”，详细请参考时间单位页面。name为Avg、Dev、Count、First、Last、LeastSquares、Max、Min、Percentile、Sum时才需填此项，若不填写则sampling为整个查询时间范围。
percentile	Double	可选	百分数，取值范围为(0,1]，如0.1表示10%，name为Percentile时必填
divisor	Double	可选	除数，name为Div时必填
factor	Double	可选	倍数，name为Scale时必填
timeUnit	String	可选	时间单位，name为Rate时必填

日历对齐的规则如下：

如果查询的时间范围是 6月15日12:12:12.000 - 9月17日11:11:11.000，通过“自然日历对齐”查询，采样周期1月，则生成4个值。

第1个值是6月15日12:12:12.000 - 6月30日23:59:59.999的聚合值，

第2个值是7月1日00:00:00.000 - 7月31日23:59:59.999的聚合值，

第3个值是8月1日00:00:00.000 - 8月31日23:59:59.999的聚合值，

第4个值是9月1日00:00:00.000 - 9月17日11:11:11.000 的聚合值。

Fill对象

参数名称	参数类型	是否必须	说明
type	String	必须	插值类型，目前支持Linear（线性插值）、Previous（按前一个值插值）、Fixed（固定值插值）
interval	String	必须	插值间隔，一个时间序列在此间隔内没有值则进行插值，格式请参考时间单位
maxWriteInterval	String	可选	最大写入间隔，一个时间序列的数据最大写入间隔（在此间隔内必然有值），默认为0。格式请参考时间单位。 系统会尝试查找从(start - maxWriteInterval)到start，以及end到(end + maxWriteInterval)的点。 如果(start - maxWriteInterval)到start找不到点，则start到end间第一个点之前的缺少的点会按第一个点的值进行插值； 如果end到(end + maxWriteInterval)找不到点，则start到end间最后一个点之后的缺少的点会按最后一个点的值进行插值。 type为Fixed时忽略该参数
value	Int/Double/String	可选	固定值插值所插入的固定值，type为Fixed时必填
field	String	可选	需要插值的field，不填此参数默认为对所有field进行插值

返回参数

参数名称	参数类型	说明
results	List<Result>	结果列表，与queries一一对应，由Result对象组成的数组

Result对象

参数名称	参数类型	说明
metric	String	结果的metric名称
field	String	结果的field名称
fields	List<String>	结果的fields列表
tags	List<String>	结果的tags列表
rawCount	Int	原始的data point数目
consumedCount	Int	扫描计费的data point数目
groups	List<Group>	结果的分组列表，由Group对象组成的数组
truncated	Boolean	是否所有数据都返回了，true表示后面还有数据，false表示后面已经没有数据，默认是false 当使用了groupBy或aggregators时，没有此项
nextMarker	String	用于分页查询，获取下一批数据所需要传递的marker值，当truncated为true时才有此项
presamplingRuleId	String	预处理规则命中ID，没有参数表示没有命中

Group对象

参数名称	参数类型	说明
groupInfos	List<GroupInfo>	该分组的信息，由GroupInfo对象组成的数组
values	List<List<Any>>	时间与值的序列，二维数组，第一个元素是timestamp，类型是Int，后面多个元素是field对应的value，个数为fields的个数，类型是Int/Double/String。之后多个元素是tag对应的value，个数为tags的个数，类型是String。

GroupInfo对象

参数名称	参数类型	说明
name	String	分组方式，目前仅支持Tag
tags	Object	该分组的tag，Object中的一对key-value表示一个tag的key-value，name为Tag时有此项

单域请求示例

```
 PUT /v1/datapoint?query HTTP/1.1
Host: {database}.tsdb.iot.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-08T16:49:51Z

{
    "queries": [{
        "metric": "cpu_idle",
        "field": "test",
        "filters": {
            "start": "1 hour ago",
            "tags": {
                "host": ["server1", "server2"]
            },
            "value": ">= 10"
        },
        "groupBy": [{
            "name": "Tag",
            "tags": ["rack"]
        }],
        "limit": 1000,
        "aggregators": [{
            "name": "Sum",
            "sampling": "10 minutes"
        }]
    }],
    "disablePresampling": false
}
```

单域返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 72492aee-1470-46d0-8a4d-0dab7b8e67b7

{
    "results": [{
        "metric": "cpu_idle",
        "field": "test",
        "rawCount": 1000,
        "groups": [{
            "groupInfos": [{
                "name": "Tag",
                "tags": {
                    "rack": "rack1"
                }
            }],
            "values": [
                [1465718968506, 10],
                [1465718985346, 12],
                [1465718992879, 15]
            ]
        }],
        "presamplingRuleId": "21"
    }]
}
```

多域请求示例

```
 PUT /v1/datapoint?query HTTP/1.1
Host: {database}.tsdb.iot.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-08T16:49:51Z

{
    "queries": [{
        "metric": "cpu_idle",
        "fields": ["field1","field2"],
        "tags": ["rack"],
        "filters": {
            "fields": [{
                    "field": "field1",
                    "value": ">= 10"
                },{
                    "field": "field2",
                    "value": "<= 10"
            }],
            "start": "1 hour ago",
            "tags": {
                "host": ["server1", "server2"]
            },
        },
        "groupBy": [{
            "name": "Tag",
            "tags": ["rack"]
        }],
        "limit": 1000,
        "aggregators": [{
            "name": "Sum",
            "sampling": "10 minutes"
        }]
    }],
    "disablePresampling": false
}
```

多域返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 72492aee-1470-46d0-8a4d-0dab7b8e67b7

{
    "results": [{
        "metric": "cpu_idle",
        "fields": ["field1","field2"],
        "tags": ["rack"],
        "rawCount": 1000,
        "groups": [{
            "groupInfos": [{
                "name": "Tag",
                "tags": {
                    "rack": "rack1"
                }
            }],
            "values": [
                [1465718968506, 10, 1.0, "rack1"],
                [1465718985346, 12, 2.0, "rack1"],
                [1465718992879, 15, 11.0, "rack1"]
            ]
        }],
    }]
}
```

或查询请求示例

```
 PUT /v1/datapoint?query HTTP/1.1
Host: {database}.tsdb.iot.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-08T16:49:51Z

{
    "queries": [{
        "metric": "cpu_idle",
        "fields": ["field1","field2"],
        "tags": ["rack"],
        "filters": {
            "or": [{
                "fields": [{
                        "field": "field1",
                        "value": ">= 10"
                    },{
                        "field": "field2",
                        "value": "<= 10"
                 }],
                "start": "1 hour ago"
            },{
                "tags":[{
                    "tag": "rack",
                    "in": ["rack1", "rack2"]
                 }],
                "start": "2 hour ago"
            }]
        },
        "groupBy": [{
            "name": "Tag",
            "tags": ["rack"]
        }],
        "limit": 1000,
        "aggregators": [{
            "name": "Sum",
            "sampling": "10 minutes"
        }]
    }]
    "disablePresampling": false
}
```

或查询返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 72492aee-1470-46d0-8a4d-0dab7b8e67b7

{
    "results": [{
        "metric": "cpu_idle",
        "fields": ["field1","field2"],
        "tags": ["rack"],
        "rawCount": 1000,
        "groups": [{
            "groupInfos": [{
                "name": "Tag",
                "tags": {
                    "rack": "rack1"
                }
            }],
            "values": [
                [1465718968506, 10, 1.0, "rack1"],
                [1465718985346, 12, 2.0, "rack1"],
                [1465718992879, 15, 11.0, "rack1"]
            ]
        }],
    }]
}
```

插值请求示例

```
 PUT /v1/datapoint?query HTTP/1.1
Host: {database}.tsdb.iot.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-08T16:49:51Z

{
    "queries": [{
        "metric": "cpu_idle",
        "filters": {
            "start": "1 hour ago"
            "tags": {
                "host": ["server1"]
            }
        },
        "fill": {
            "type": "Linear",
            "interval": "5 minutes",
            "maxWriteInterval": "30 minutes"
        }
    }]
}
```

插值返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 72492aee-1470-46d0-8a4d-0dab7b8e67b7

{
    "results": [{
        "metric": "cpu_idle",
        "field": "value",
        "rawCount": 2,
        "groups": [{
            "groupInfos": [],
            "values": [
                [1499072400000, 20.4],
                [1499072700000, 17.85],
                [1499073000000, 15.3],
                [1499073300000, 22.95],
                [1499073600000, 30.6],
                [1499073900000, 38.25],
                [1499074200000, 45.9],
                [1499074500000, 53.55],
                [1499074800000, 61.2],
                [1499075100000, 61.2],
                [1499075400000, 61.2],
                [1499075700000, 61.2]
            ]
        }]
    }]
}
```

标签匹配请求示例

```
 PUT /v1/datapoint?query HTTP/1.1
Host: {database}.tsdb.iot.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-08T16:49:51Z

{
    "queries": [{
        "metric": "cpu_idle",
        "tags": ["tag1"],
        "filters": {
            "start": "1 hour ago"
            "tags": [{
                "tag": "tag1",
                "like": "value%"
            }]
        }
    }]
}
```

标签匹配返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 72492aee-1470-46d0-8a4d-0dab7b8e67b7

{
    "results": [{
        "metric": "cpu_idle",
        "field": "value",
        "tags": ["tag1"],
        "rawCount": 2,
        "groups": [{
            "groupInfos": [],
            "values": [
                [1499072400000, 20.4, "value1"],
                [1499072700000, 17.85, "value2"]
            ]
        }]
    }]
}
```

SQL查询接口

方法	API	说明
GET	/v1/row?sql={statement}	使用ANSI SQL查询时序数据

请求参数

参数名称	参数类型	是否必须	说明
sql	String	必须	ASIC语法的 SQL 查询语句，Eg. select * from metric 放在get请求中时需要进行url encode

返回参数

参数名称	参数类型	说明
Columns	List<Column>	SQL查询的schema信息，由Column对象组成的数组
rows	List<List<Object>>	SQL查询的结果列表，由Object对象组成的二维数组

Column对象

参数名称	参数类型	说明
name	String	Column的名称

请求示例

```
 GET /v1/row?sql=select%20%2A%20from%20metric HTTP/1.1
Host: {database}.tsdb.iot.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-08T16:49:51Z
```

返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 72492aee-1470-46d0-8a4d-0dab7b8e67b7

{
    "columns": [{
        "name": "time"
    },{
        "name": "field1",
    },{
        "name": "tag1"
    }],
    "rows": [
        [1499072400000, 20.4, "value1"],
        [1499072700000, 17.85, "value2"]
    ]
}
```

管理API接口说明

创建database

方法	API	说明
POST	/v1/database?clientToken={clientToken}	创建database

请求参数

参数名称	参数类型	是否必须	说明
clientToken	String	必须	用于保证接口幂等性
databaseName	String	必须	database的名称
description	String	可选	database的描述
ingestDataPointsMonthly	Int	必须	写入额度，单位：百万点/月
queryUnitsMonthly	Int	可选	查询额度，单位：万个/月
storeBytesQuota	Int	可选	存储空间额度，单位：byte
purchaseLength	Int	必须	购买时长，单位：月
couponName	String	可选	代金券号

> 支付说明： couponName不为空且是合法的代金券号，那么将优先使用代金券支付。 如果代金券金额不足时，则将使用代金券+账户余额支付。 如果代金券号不可用时，则使用账户余额支付。 当用户购买时长为10、11或12个月时，由于优惠政策，实际只需要10个月的花费，即可使用12个月的服务。

返回参数

参数名称	参数类型	说明
databaseId	String	database的ID
charge	BigDecimal	消费金额，单位：元
expiredTime	Date	过期时间
orderId	String	订单ID

请求示例

```
POST /v1/database?clientToken=ed35ec43-0f2a-4648-8828-de9b28caceda HTTP/1.1
Host: tsdb.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-06T11:52:41Z

{
    "databaseName": "test",
    "description": "test",
    "ingestDataPointsMonthly": 1,
    "purchaseLength": 1
}
```

返回示例

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 28597c34-096d-489c-908f-38844c92032e

{
    "databaseId": "tsdb-5atue8m3sxsv",
    "charge": 29.9,
    "expiredTime": "2016-07-06T11:52:41Z",
    "orderId": "d04da3c16bd042af9e916a75bb1fa19g"
}
```

删除database

方法	API	说明
DELETE	/v1/database/{databaseId}	删除database

请求参数

参数名称	参数类型	是否必须	说明
databaseId	String	必须	database的ID

请求示例

```
DELETE /v1/database/tsdb-5atue8m3sxsv HTTP/1.1
Host: tsdb.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-06T11:52:41Z
```

返回示例

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 28597c34-096d-489c-908f-38844c92032e
```

获取database信息

方法	API	说明
GET	/v1/database/{databaseId}	获取ID为{databaseId}的database信息

请求参数

参数名称	参数类型	是否必须	说明
databaseId	String	必须	database的ID

返回参数

返回一个Database对象。

Database对象

参数名称	参数类型	说明
databaseId	String	database的ID
databaseName	String	database的名称
description	String	database的描述
endpoint	String	database的访问域名
quota.ingestDataPointsMonthly	Int	写入额度，单位：百万点/月
quota.queryUnitsMonthly	Int	查询额度，单位：万个/月
quota.storeBytesQuota	Int	存储空间额度单位：GB
status	String	database状态，详见Database状态表
autoExport	Boolean	是否启用自动导出
createTime	Date	database的创建时间
expiredTime	Date	过期时间

请求示例

```
 GET /v1/database/tsdb-5atue8m3sxsv HTTP/1.1
Host: tsdb.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-06T11:52:41Z
```

返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 28597c34-096d-489c-908f-38844c92032e

{
    "databaseId": "tsdb-5atue8m3sxsv",
    "databaseName": "test",
    "description": "test",
    "endpoint": "test.tsdb.iot.gz.baidubce.com",
    "quota": {
        "ingestDataPointsMonthly": 1 ,
        "queryUnitsMonthly": 10,
        "storeBytesQuota": 1
    },
    "status": "Active",
    "autoExport": true,
    "createTime": "2016-06-06T14:21:01Z",
    "expiredTime": "2016-07-06T14:21:01Z"
}
```

获取database列表

方法	API	说明
GET	/v1/database	获取database列表

返回参数

参数名称	参数类型	说明
databases	List<Database>	database列表，由Database对象组成的数组

请求示例

```
 GET /v1/database HTTP/1.1
Host: tsdb.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-06T11:52:41Z
```

返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 28597c34-096d-489c-908f-38844c92032e

{
    "databases": [{
        "databaseId": "tsdb-5atue8m3sxsv",
        "databaseName": "test",
        "description": "test",
        "endpoint": "test.tsdb.iot.gz.baidubce.com",
        "quota": {
            "ingestDataPointsMonthly": 1,
            "queryUnitsMonthly": 10,
            "storeBytesQuota": 1
        },
        "status": "Active",
        "autoExport": true,
        "createTime": "2016-06-06T14:21:01Z",
        "expiredTime": "2016-07-06T14:21:01Z"
    }]
}
```

创建删除任务

方法	API	说明
PUT	/v1/database/{databaseId}?delete	创建删除任务

请求参数

参数名称	参数类型	是否必须	说明
databaseId	String	必选	要删除数据的databaseId
metrics	List<String>	可选	要被删除的数据点对应的metric列表。metricFieldsList和metrics必须有且只能有一个存在
metricFieldsList	List<MetricFields>	可选	要被删除的数据点对应的field列表。metricFieldsList和metrics必须有且只能有一个存在
start	Long	可选	要被删除的数据点对应的起始时间，默认为Long.Min
end	Long	可选	要被删除的数据点对应的截止时间，默认为Long.Max
tags	Object	可选	要被删除的数据点过滤的tags，Object的每个key对应一个metirc，Object的每个value都是List<TagFilter>类型的，表示要删除的数据点的Tag需要满足的条件。tags中出现的metric必须在metrics或metricFieldsList中

MetricFields 对象

参数名称	参数类型	是否必须	说明
metric	String	必选	要删除的数据点对应的metric
fields	List<String>	可选	要删除的数据点对应的field列表

返回参数

参数名称	参数类型	说明
taskId	String	任务的ID

请求示例

```
 PUT /v1/database/tsdb-5atue8m3sxsv?delete HTTP/1.1
Host: tsdb.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-06T11:52:41Z

{
    "metrics": [
        "cpu_idle",
        "server"
    ],
    "tags": {
        "cpu_idel": [{
            "tagKey": "rack",
            "in": ["rack1", "rack2"]
        }],
        "server":[{
            "tagKey": "host",
            "in": ["0.0.0.0"]
        }]
    },
    "start": 13897024887,
    "end": 13897029887
}
```

返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 28597c34-096d-489c-908f-38844c92032e

{
    "taskId": "123456"
}
```

创建导出任务

方法	API	说明
PUT	/v1/database/{databaseId}?export	创建导出任务

请求参数

参数名称	参数类型	是否必须	说明
databaseId	String	必须	database的ID
metrics	List<String>	可选	支持列表metrics导出，未设置或者空，则导出database中的所有metric
format	String	必须	导出文件的格式，目前仅支持"CSV"
start	Long	可选	导出数据的起始时间戳，未设置则默认为Long.Min_Value
end	Long	可选	导出数据的截止时间戳，未设置则默认为Long.Max_Value
tags	Object	可选	要被导出的data points过滤的tags，Object的key是metric，value是由TagFilter组成的列表
bosUrl	String	必须	导出到bos的目录地址
singleFile	Boolean	必须	是否导出为单个文件，未设置则默认为false

TagFilter对象

参数名称	参数类型	是否必须	说明
tagKey	String	必须	需要过滤的metric的tag的key
in	List<String>	可选	可以包含的tag的value列表，不可与notIn和like同时存在
notIn	List<String>	可选	需要过滤的tag的value列表，不可与in和like同时存在
like	String	可选	需要匹配的规则，“%”匹配任意数量的字符，“_”匹配单个字符，转义字符为“\”。不可与in和notIn同时存在

返回参数

参数名称	参数类型	说明
taskId	String	任务的ID

请求示例

```
 Put /v1/database/{databaseId}?export HTTP/1.1
Host: tsdb.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-06T11:52:41Z
{
"bosUrl": "bos://bucket1/",
"metrics": [
    "cpu_idle"
],
"singleFile": true,
"format": "csv"
}
```

返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 28597c34-096d-489c-908f-38844c92032e

{
    "taskId": "i4jzt228238nxueyq932"
}
```

获取任务信息

方法	API	说明
GET	/v1/database/{databaseId}/task/{taskId}	获取ID为{databaseId}的ID为{taskId}的任务信息

请求参数

参数名称	参数类型	是否必须	说明
databaseId	String	必须	database的ID
taskId	String	必须	task的ID

返回参数

返回一个Task对象，具体说明见下文Task对象。

Task对象

参数名称	参数类型	说明
taskId	String	任务的ID
type	String	任务类型，Export或Delete
params	String	任务参数，根据不同类型有不同，内容为json字符串
status	String	任务状态，Pending/Running/Completed/Failed/Canceled
error.code	String	任务失败代号，任务状态为Failed时有此项
error.message	String	任务失败信息，任务状态为Failed时有此项
results	String	任务结果，根据不同类型不同，内容为json字符串。Export任务包含exportedDataPointsCount、exportedLinesCount、exportedTime、bucketName和objectNames。Delete任务包含deletedDataPointsCount, failures
createTime	Date	任务的创建时间
completionTime	Date	任务的完成时间，只有status为Completed/Failed时才有这个字段

请求示例

```
GET /v1/database/tsdb-5atue8m3sxsv/task/123457 HTTP/1.1
Host: tsdb.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-06T11:52:41Z
```

返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 28597c34-096d-489c-908f-38844c92032e

{
    "taskId":"idwby9gvtk0eg8a4x3hn",
    "type":"Export",
    "params":{
        "bosUrl":"bos://bucket1/",
        "metrics":[
        ],
        "tags":{}
        "start":0,
        "end":9223372036854775807,
        "singleFile":false
    },
    "status":"Completed",
    "error":{
        "code":"",
        "message":""
    },
    "results":{
        "exportedDataPointsCount":27375,
        "exportedLinesCount":27375,
        "objectNames":["file1","file2"]
    },
    "createTime":"2018-11-05T06:29:52Z",
    "completionTime":"2018-11-05T06:30:10Z"
}
```

添加预处理规则

方法	API	说明
POST	/v1/database/{databaseId}/presampling	创建database预处理规则

请求参数

参数名称	参数类型	是否必须	说明
ruleName	String	必须	规则的名称，3~40个字符
databaseId	String	必须	database的ID
metrics	List<String>	必须	适用metric的名称列表 **["*"]：表示适用所有列表**
fields	List<String>	可选	field的名称列表，空表示对该规则应用于所有fields
indexes	List<String>	可选	索引列表，查询filter的tagKey列表
aggregators	List<Aggregator>	必须	Aggregator列表，由Aggregator组成的列表，聚合函数上限3个，且至少有一个函数带采样时间；多个聚合函数将按照顺序进行嵌套聚合计算
ttlInDays	Int	可选	数据保留时间，单位为天，0表示永久保留
baseTime	Long	可选	预处理规则的基准时间，默认是2000/1/1 00:00:00 周是这个时间之后的第一个周日
fills	List	可选	插值列表，见IOT-TSDB Data API#TSDBDataAPI-Fill，固定值插值时，只允许插入Long/Double类型数据

返回参数

方法	API	说明
ruleId	String	预处理规则ID

请求示例

```
 POST /v1/database/{databaseId}/presampling HTTP/1.1
Host: tsdb.iot.gz.bce-internal.baidu.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
x-bce-date: 2016-06-06T11:52:41Z
{
    "ruleName": "test",
    "metrics": ["cpu","memory"],
    "indexes": ["server1", "server2"],
    "aggregators:": [
        {"name": "Div", "divisor": "3"}
        {"name": "Avg","sampling": "1 hour"}
    ]
}
```

返回示例

```
 HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 28597c34-096d-489c-908f-38844c92032e
{
    "ruleId": "55"
}
```

聚合函数

Avg

名称	说明	支持类型
Avg	平均值，以每个采样时间范围内的value的平均值作为结果	Number

请求参数

参数名称	参数类型	是否必须	说明
sampling	String	可选	采样的时间长度，如"10 minutes"，若不填写则sampling为整个查询时间范围

Dev

名称	说明	支持类型
Dev	标准差，以每个采样时间范围内的value的标准差作为结果	Number

请求参数

参数名称	参数类型	是否必须	说明
sampling	String	可选	采样的时间长度，如"10 minutes"，若不填写则sampling为整个查询时间范围

Count

名称	说明	支持类型
Count	数目，以每个采样时间范围内的点的数目作为结果	Number/String

请求参数

参数名称	参数类型	是否必须	说明
sampling	String	可选	采样的时间长度，如"10 minutes"，若不填写则sampling为整个查询时间范围

First

名称	说明	支持类型
First	第一个，以每个采样时间范围内的第一个value作为结果	Number/String

请求参数

参数名称	参数类型	是否必须	说明
sampling	String	可选	采样的时间长度，如"10 minutes"，若不填写则sampling为整个查询时间范围

Last

名称	说明	支持类型
Last	最后一个，以每个采样时间范围内的最后一个value作为结果	Number/String

请求参数

参数名称	参数类型	是否必须	说明
sampling	String	可选	采样的时间长度，如"10 minutes"，若不填写则sampling为整个查询时间范围

LeastSquares

名称	说明	支持类型
LeastSquares	最小二乘法，对每个采样时间范围内的value进行最小二乘法的拟合	Number

请求参数

参数名称	参数类型	是否必须	说明
sampling	String	可选	采样的时间长度，如"10 minutes"，若不填写则sampling为整个查询时间范围

Max

名称	说明	支持类型
Max	最大值，以每个采样时间范围内的value的最大值作为结果	Number

请求参数

参数名称	参数类型	是否必须	说明
sampling	String	可选	采样的时间长度，如"10 minutes"，若不填写则sampling为整个查询时间范围

Min

名称	说明	支持类型
Min	最小值，以每个采样时间范围内的value的最小值作为结果	Number

请求参数

参数名称	参数类型	是否必须	说明
sampling	String	可选	采样的时间长度，如"10 minutes"，若不填写则sampling为整个查询时间范围

Percentile

名称	说明	支持类型
Percentile	百分位数，以每个采样时间范围内的value的第p（见请求参数percentile）百分位数作为结果	Number

请求参数

参数名称	参数类型	是否必须	说明
sampling	String	可选	采样的时间长度，如"10 minutes"，若不填写则sampling为整个查询时间范围
percentile	Double	必须	百分数，取值范围为(0,1]，如0.1表示10%

Sum

名称	说明	支持类型
Sum	和值，以每个采样时间范围内的value的总和作为结果	Number

请求参数

参数名称	参数类型	是否必须	说明
sampling	String	可选	采样的时间长度，如"10 minutes"，若不填写则sampling为整个查询时间范围

Diff

名称	说明	支持类型
Diff	差值，以每两个相邻的value的差值作为结果	Number

Div

名称	说明	支持类型
Div	以每个value除以一个除数（见请求参数divisor）作为结果	Number

请求参数

参数名称	参数类型	是否必须	说明
divisor	Double	必须	除数，不能为0

Scale

名称	说明	支持类型
Scale	以每个value乘以一个倍数（见请求参数factor）作为结果	Number

请求参数

参数名称	参数类型	是否必须	说明
factor	Double	必须	倍数

Rate

名称	说明	支持类型
Rate	变化率，以每两个相邻的value在单位时间（见参数timeUnit）的变化率作为结果	Number

请求参数

参数名称	参数类型	是否必须	说明
timeUnit	String	必须	时间单位

AdjacentUnique

名称	说明	支持类型
AdjacentUnique	相邻值去重，相同的值只保留第一个，非相邻的值不去重。譬如“1, 1, 2, 2, 2, 1, 1, 3”，相邻值去重后的结果为“1, 2, 1, 3”	Number/String/Bytes

分组方式

Tag

名称	说明
Tag	按照指定的tag进行分组

请求参数

参数名称	参数类型	是否必须	说明
tags	List<String>	必须	按照哪些tag进行分组，为tag key的数组

返回参数

参数名称	参数类型	说明
tags	Object	该分组的tag，Object中的一对key-value表示一个tag的key-value

时间单位

相对时间的格式为"数字 时间单位 ago"（如"5 days ago"），时间范围的格式为"数字 时间单位"（如"12 hours"）。

其中时间单位有以下这些：

时间单位	说明
milliseconds	毫秒
millisecond	毫秒
seconds	秒
second	秒
minutes	分钟
minute	分钟
hours	小时
hour	小时
days	天
day	天
weeks	周
week	周
months	月
month	月
years	年
year	年
sc	自然秒
mc	自然分
hc	自然时
dc	自然天
nc	自然月
yc	自然年

附录

错误状态码

Code	HTTP状态码	说明
AccessDenied	403 Forbidden	没有权限访问该database
InsufficientQuota	429 Too Many Requests	配额不足，请升级套餐
ResourceNotFound	404 Not Found	请求的资源未找到，请检查URL
InvalidArgument	400 Bad Request	请求参数错误，请检查参数
ParameterMissing	400 Bad Request	缺少请求参数
MethodNotSupported	405 Method Not Allowed	请求的方法错误
ContentTypeNotSupported	415 Unsupported Media Type	不支持请求的Content-Type头指定的格式
NotAcceptable	406 Not Acceptable	请求的Accept头没有包含需要返回的application/json格式
InternalError	500 Internal Server Error	系统内部错误，请稍后重试

Database状态表

状态	说明
Active	Database处于正常状态
Stopped	因为欠费而被停止

更新历史

2020-11-18

- 写入data point接口type改为必填项

2019-08-20

- 查询数据点接口新增offset字段

2019-03-22

- 创建db接口新增查询额度参数
- 查询数据点接口查询结果新增consumedCount字段

2018-12-04

- 新增创建删除数据点任务接口
- 新增创建导出数据点任务接口
- 新增获取任务信息接口

2018-06-25

- 启用存储空间限额，创建数据库接口新增字节存储额度参数

2018-06-14

- 数据API接口新增SQL查询功能

2017-12-05

- 查询data point接口增加分页和tag like功能。

服务等级协议SLA

时序数据库TSDB服务等级协议SLA(V2.0)

协议生效时间：2019 年 1 月 15 日

本服务等级协议（Service Level Agreement，简称“SLA”）规定了百度智能云向客户提供时序数据库（TSDB）的服务可用性等级指标及赔偿方案。

1. 定义

服务周期：一个服务周期为一个自然月，如不满一个自然月不计算为一个服务周期。

服务周期总分钟数：按照服务周期内的总天数 * 24（小时）* 60（分钟）计算。

服务不可用分钟数：当用户所有试图与指定的TSDB实例建立连接的连续尝试均失败，且状态持续超过5分钟以上，则视为该分钟内该TSDB实例服务不可用。 在一个服务周期内TSDB实例不可用分钟数之和即服务不可用分钟数。

月度服务费用：用户在一个自然月中就单个TSDB实例所支付的服务费用总额，如果用户一次性支付了多个月份的服务费用，则将按照所购买的月数分摊计算月度服务费用.

2. 服务可用性

2.1 服务可用性计算公式

服务可用性以单个实例为维度，按照如下方式计算：

$$\text{服务可用性} = \left(\frac{\text{服务周期总分钟数} - \text{服务不可用分钟数}}{\text{服务周期总分钟数}}\right) \times 100\%$$

TSDB服务可用性不低于99.9%，即用户每月业务可用时间不低于30*24*60*99.9%=43156.8（分钟），每月不可用时间不超过43200-43178.4=43.2（分钟）。

2.2 不可用时间

以下情况不计入不可用时间：

(1) 百度智能云预先通知用户后进行系统维护所引起的，包括割接、维修、升级和模拟故障演练；

(2) 任何百度智能云所属设备以外的网络、设备故障或配置调整引起的；

(3) 用户的应用程序受到黑客攻击而引起的；

(4) 用户发送至 TSDB 的复杂慢查询语句查询超时，超时时间请参见产品文档；

(5) 用户维护不当或保密不当致使数据、口令、密码等丢失或泄漏所引起的；

(6) 用户的疏忽或由用户授权的操作所引起的；

(7) 用户未遵循百度智能云产品使用文档或使用建议引起的；

(8) 不可抗力或者其他意外事件引起的；

(9) 其他非百度智能云原因所造成的不可用。

3. 赔偿方案

3.1 赔偿标准

每个TSDB实例按单实例月度服务可用性，按照下表中的标准计算赔偿金额，赔偿方式仅限于用于购买TSDB产品的代金券，且赔偿总额不超过未达到服务可用性承诺当月用户就该TSDB实例支付的月度服务费用。

服务可用性	赔偿代金券金额
低于99.9%但等于或高于99.00%	月度服务费用的15%
低于99.00%但等于或高于95.00%	月度服务费用的30%
低于95.00%	月度服务费用的100%

3.2 赔偿时限

用户可以在每月第五（5）个工作日后对上个月没有达到可用性的实例提出赔偿申请。赔偿申请必须限于在TSDB实例没有达到可用性的相关月份结束后两（2）个月内提出。超出申请时限的赔偿申请将不被受理。

4. 其他

(1) 在法律法规允许的范围内，百度智能云负责对本协议进行解释说明。
(2) 百度智能云有权对本SLA条款作出修改。如本SLA条款有任何修改，百度智能云以网站公示或发送邮件的方式通知您。如您不同意百度智能云对SLA所做的修改，您有权停止使用TSDB服务，如您继续使用TSDB服务，则视为您接受修改后的SLA。
(3) 本协议项下百度智能云TSDB服务对于用户所有的通知均可通过网页公告、站内信、电子邮件、手机短信或其他形式等方式进行；该类通知于发送之日视为已送达收件人。百度智能云不对用户承担因此产生的任何损失。
(4) 本协议的订立、执行和解释及争议的解决均应适用中国法律并受中国法院管辖。如双方就本协议内容或其执行发生任何争议，双方应尽量友好协商解决；协商不成时，任何一方均可向北京市海淀区人民法院提起诉讼。
(5) 本协议构成双方对本协议之约定事项及其他有关事宜的完整协议，除本协议规定的之外，未赋予本协议各方其他权利。
(6) 如本协议中的任何协议无论因何种原因完全或部分无效或不具有执行力，本协议的其余协议仍应有效并且有约束力。
(7) 关于用户约束条款，详见《用户服务协议》中的"用户的权利与义务"相关条款内容。
(8) 关于服务商免责条款，详见《用户服务协议》中的"免责声明"相关条款内容。

常见问题

常见问题总览

数据库创建及设置

- 创建示范数据库和创建数据库有什么不同？
- 时序数据库的数据可以备份到哪里？
- 能不能增加时序数据库的数据点配额？

数据点查询

- 为什么查询面板中生成图表仍为空？

数据点写入

- 时序数据库的数据如何导入？

数据管理

- 为什么Postman请求数据管理接口提示IamSignatureInvalid？
- 为什么使用官方Postman鉴权计算脚本请求服务接口会时而可用、时而报403？

用量提示

- 时序数据库是否提供用量报警功能？

数据库创建及设置

创建示范数据库和创建数据库有什么不同？

示范数据库中含有示范数据，用户可直接通过查询面板等工具来查看数据图表。

时序数据库的数据可以备份到哪里？

目前可以备份到百度对象存储BOS。

能不能增加时序数据库的数据点配额？

如果想增加时序数据库的数据点配额，请提交工单申请。

数据点查询

为什么查询面板中生成图表仍为空？

原因1：时间范围设置错误。

图表的横轴是指数据库实例中的存储点数的timestamp字段的值，而不是导入的时间。出现这种情况，很有可能是timestamp字段的值与导入时间不一致而导致的。

原因2：数据点的类型为string类型。

如下例所示，原始数据如下：

```
{
  "time": 1465376157007,
  "name": "cpu_idle",
  "score": "51",
  "host": "server1",
  "rack": "rack1",
  "other": "something"
}
```

通过规则引擎将该数据转发至TSDB，规则引擎的规则设置如下：

```
name AS metric, score AS _value, `time` AS _timestamp, host, rack
```

此时，写入TSDB的数据点为：

```
{
  "metric":"cpu_idle",
  "_value":"51",
  "_timestamp":1465376157007,
  "host":"server1",
  "rack":"rack1"
}
```

由于原始数据为`"score": "51"`，此时该数据为字符串类型。字符串类型数据写入TSDB后将无法在查看面板中生成图表。

可以通过规则引擎对数据类型进行转换，解决上述问题，具体规则配置如下：

```
name AS metric, CAST(score AS DOUBLE) AS _value, `time` AS _timestamp, host, rack
```

有关CAST函数的介绍，请参看常用SQL函数。

数据点写入

时序数据库的数据如何导入？

目前有两种方式导入时序数据库，一种通过调用open API写入数据点；一种可以将BOS里的数据导入时序数据库，导入的文件类型为csv，具体操作请查看连接数据库

数据管理

为什么Postman请求数据管理接口提示IamSignatureInvalid？

如采用的是百度智能云提供的postman签名信息生成脚本、且确认配置无误，可在postman中查看Headers中是否存在置灰不用的key-value，如果有，可将其删除后再尝试请求是否正常。

为什么使用官方Postman鉴权计算脚本请求服务接口会时而可用、时而报403？

在Postman中新增环境变量"signedHeaders",将该变量的INITIAL VALUE设置为"host;x-bce-date "即可

VARIABLE	INITIAL VALUE	CURRENT VALUE
signedHeaders	host;x-bce-date	host;x-bce-date

用量提示

TSDB用量提示

TSDB在各项配额（月写入量、月查询单位、时间序列、存储空间）达到75%、90%、100%时会自动触发报警，通过短信、邮件等方式通知用户及时关注使用情况，避免因额度不足导致的服务不可用。

申请邀请

申请成为邀测用户

邀测功能：

支持长数据格式存储

如果您需要使用TSDB存储String超过256字符、Byte类型超过200字节、或者试用BigDecimal类型，请申请试用。