
TSDB 文档

2018-02-07



百度云
cloud.baidu.com

目录

| | | |
|-------|-------------------------|----|
| 1 | 产品描述 | 1 |
| 1.1 | 产品概述 | 1 |
| 1.2 | 产品功能 | 1 |
| 1.3 | 产品优势 | 1 |
| 1.4 | 名词解释 | 2 |
| 1.4.1 | 名词解释 | 2 |
| 1.4.2 | 实践说明 | 3 |
| 1.5 | 系统限制 | 4 |
| 1.5.1 | 资源限制 | 4 |
| 1.5.2 | 命名限制 | 4 |
| 1.6 | 应用场景 | 5 |
| 2 | 操作指南 | 6 |
| 2.1 | 操作准备 | 6 |
| 2.1.1 | 介绍 | 6 |
| 2.1.2 | 账号注册 | 6 |
| 2.1.3 | 操作流程介绍 | 6 |
| 2.2 | 创建数据库 | 7 |
| 2.2.1 | 创建示范数据库 | 8 |
| 2.2.2 | 生成示范数据库图表 | 8 |
| | 查看北京某观测点全年PM2.5大于100的情况 | 9 |
| 2.3 | 连接数据库 | 10 |

| | | |
|-------|----------------------|----|
| 2.3.1 | 通过API/SDK写入数据 | 10 |
| 2.3.2 | 通过控制台导入数据 | 11 |
| 2.4 | 生成图表 | 11 |
| 2.5 | 管理数据库 | 13 |
| 2.5.1 | 导出数据 | 13 |
| | 手动导出数据 | 13 |
| | 自动导出数据 | 15 |
| 2.5.2 | 数据导入 | 15 |
| | 导入示范数据 | 15 |
| | 导入数据 | 16 |
| 2.5.3 | 监控数据库 | 17 |
| 2.5.4 | 数据预处理 | 19 |
| | 新建预处理规则 | 19 |
| | 查看预处理详情 | 20 |
| | 查看预处理结果 | 21 |
| 2.5.5 | 数据清理 | 21 |
| 2.6 | 多用户协作 | 22 |
| 2.7 | 对接hive sql | 24 |
| 2.7.1 | Tsdb storage handler | 24 |
| 2.7.2 | 在Hive CLI或Hue中使用 | 24 |
| 2.7.3 | 场景示例 | 25 |
| | 计算风速 | 25 |
| | 计算车辆在时间上的使用情况 | 25 |
| 3 | 产品定价 | 28 |
| 3.1 | 包年包月计费 | 28 |
| 4 | Java SDK文档 | 30 |
| 4.1 | 概述 | 30 |

| | | |
|--------|------------------|----|
| 4.2 | 安装SDK工具包 | 30 |
| 4.3 | 快速入门 | 31 |
| 4.4 | 创建TsdClient | 31 |
| 4.5 | 写入操作 | 33 |
| 4.5.1 | 写入单域数据点 | 33 |
| 4.5.2 | 写入多域数据点 | 34 |
| 4.6 | 查询操作 | 35 |
| 4.6.1 | 获取度量 (Metric) | 35 |
| 4.6.2 | 获取Field | 35 |
| 4.6.3 | 获取标签 (Tag) | 36 |
| 4.6.4 | 查询数据点 | 36 |
| | 多域查询数据点 | 37 |
| | 查询数据点和对应的tags | 38 |
| | 使用插值方式查询数据点 | 39 |
| | 分页查询原始数据点 | 40 |
| | 查询数据时使用标签过滤 | 41 |
| | 查询涉及的对象 | 44 |
| 4.7 | 生成查询数据点的预签名URL | 44 |
| 4.8 | 写入数据点的gzip压缩说明 | 44 |
| 4.9 | 管理接口 | 45 |
| 4.9.1 | 新建TsdAdminClient | 45 |
| 4.9.2 | 创建时序数据库实例 | 45 |
| 4.9.3 | 删除时序数据库实例 | 46 |
| 4.9.4 | 获取时序数据库实例 | 46 |
| 4.9.5 | 获取时序数据库实例列表 | 46 |
| 4.10 | 版本说明 | 46 |
| 4.10.1 | v0.10.24 | 46 |
| 4.10.2 | v0.10.23 | 47 |

| | | |
|--------|-----------------|----|
| 4.10.3 | v0.10.22 | 47 |
| 4.10.4 | v0.10.20 | 47 |
| 4.10.5 | v0.10.19 | 47 |
| 4.10.6 | v0.10.17 | 47 |
| 4.10.7 | v0.10.12 | 47 |
| 4.10.8 | v0.10.10 | 47 |
| 4.10.9 | v0.10.7 | 47 |
| 5 | Node SDK文档 | 48 |
| 5.1 | 概述 | 48 |
| 5.2 | 安装SDK工具包 | 48 |
| 5.3 | 快速入门 | 49 |
| 5.4 | 创建TsdbClient | 50 |
| 5.5 | 写入操作 | 50 |
| 5.5.1 | 写入单域数据点 | 50 |
| 5.5.2 | 写入多域数据点 | 51 |
| 5.6 | 查询操作 | 52 |
| 5.6.1 | 获取度量 (Metric) | 52 |
| 5.6.2 | 获取Field | 52 |
| 5.6.3 | 获取标签 (Tag) | 53 |
| 5.6.4 | 查询数据点 | 54 |
| | 单域查询数据点 | 54 |
| | 多域查询数据点 | 55 |
| | 使用插值方式查询数据点 | 56 |
| | 分页查询数据点 | 57 |
| 5.7 | 生成查询数据点的预签名URL | 58 |
| 5.8 | 写入数据点的gzip压缩说明 | 59 |
| 5.9 | 管理接口 | 60 |

| | | |
|-------|-------------------------------|----|
| 5.9.1 | 新建TsdbAdminClient | 60 |
| 5.9.2 | 创建时序数据库实例 | 60 |
| 5.9.3 | 删除时序数据库实例 | 61 |
| 5.9.4 | 获取时序数据库实例 | 62 |
| 5.9.5 | 获取时序数据库实例列表 | 62 |
| 5.10 | 错误码 | 63 |
| 5.11 | 版本说明 | 63 |
| 6 | API参考 | 64 |
| 6.1 | 介绍 | 64 |
| 6.1.1 | 名词解释 | 64 |
| 6.1.2 | 调用方式 | 64 |
| | 概述 | 64 |
| | 通用约定 | 64 |
| | 公共请求头 | 64 |
| | 公共响应头 | 65 |
| | 响应状态码 | 65 |
| | 请求消息体格式 (HTTP Request Body) | 65 |
| | 请求返回格式 (HTTP Response) | 65 |
| | 通用错误返回格式 | 66 |
| | 签名认证 | 66 |
| | 签名生成算法 | 67 |
| | 多区域选择 | 67 |
| | 服务域名 | 67 |
| 6.2 | 数据API接口说明 | 67 |
| 6.2.1 | 写入data point | 67 |
| 6.2.2 | 获取metric列表 | 70 |
| 6.2.3 | 获取field列表 | 71 |

| | | |
|--------|----------------|----|
| 6.2.4 | 获取tag列表 | 72 |
| 6.2.5 | 查询data point | 73 |
| 6.3 | 管理API接口说明 | 87 |
| 6.3.1 | 创建database | 87 |
| 6.3.2 | 删除database | 89 |
| 6.3.3 | 获取database信息 | 89 |
| 6.3.4 | 获取database列表 | 90 |
| 6.4 | 聚合函数 | 91 |
| 6.4.1 | Avg | 91 |
| 6.4.2 | Dev | 92 |
| 6.4.3 | Count | 92 |
| 6.4.4 | First | 93 |
| 6.4.5 | Last | 93 |
| 6.4.6 | LeastSquares | 94 |
| 6.4.7 | Max | 94 |
| 6.4.8 | Min | 94 |
| 6.4.9 | Percentile | 95 |
| 6.4.10 | Sum | 95 |
| 6.4.11 | Diff | 96 |
| 6.4.12 | Div | 96 |
| 6.4.13 | Scale | 96 |
| 6.4.14 | Rate | 97 |
| 6.4.15 | AdjacentUnique | 97 |
| 6.5 | 分组方式 | 97 |
| 6.5.1 | Tag | 97 |
| 6.6 | 时间单位 | 98 |
| 6.7 | 附录 | 98 |
| 6.7.1 | 错误状态码 | 98 |

| | | |
|-------|-----------------------|-----|
| 6.7.2 | Database状态表 | 99 |
| 6.8 | 更新历史 | 99 |
| 7 | 时序数据库TSDB服务等级协议 (SLA) | 100 |
| 8 | 常见问题 | 103 |
| 8.1 | 数据库创建及设置 | 103 |
| 8.1.1 | 创建示范数据库和创建数据库有什么不同? | 103 |
| 8.1.2 | 时序数据库的数据可以备份到哪里? | 103 |
| 8.1.3 | 能不能增加时序数据库的数据点配额? | 103 |
| 8.2 | 数据点查询 | 103 |
| 8.2.1 | 为什么查询面板中生成图表仍为空? | 103 |
| 8.3 | 数据点写入 | 104 |
| 8.3.1 | 时序数据库的数据如何导入? | 104 |

第1章 产品描述

1.1 产品概述

在物联网时代，企业需要处理各种设备产生的带有时间标签的数据，即时间序列数据。时间序列数据主要由各类型实时监测、检查与分析设备所采集或产生，涉及电力行业、化工等行业。这些工业数据的典型特点是：产生频率快、严重依赖于采集时间、测点多信息量大等。普通关系型数据库无法高效地处理此类数据。

为了更好的处理时间序列数据，时序数据库（Time Series Database，下简称TSDB）应运而生。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。

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。

1.2 产品功能

- 数据写入
支持通过REST API方式高并发写入时间序列数据，可线性扩展。
- 极速查询
支持海量时序数据的极速查询，每秒可返回1亿数据点的聚合查询结果。
- 丰富的函数
提供丰富的数据处理能力，支持多达16种聚合函数。
- Web图表
查询的结果可以通过Web图表方式生动、灵活的展现。

1.3 产品优势

- 高性能
基于负载均衡和分布式存储架构，提供极速的数据处理能力，可线性扩展。

- 高速度
轻松支撑每秒百万级数据点的写入，并可线性扩展；1亿数据点查询响应时间 $\leq 1s$ 。
- 高压缩比
提供优于传统关系型数据库数百倍的时序数据压缩能力，大大节约存储空间。
- 高扩展性
基于云端海量的资源，提供弹性、按需的海量数据存储能力，成本更低，免运维。
- 易集成
提供开放标准的API，可通过调用API实现控制台操作，方便第三方应用快速集成云端服务。

1.4 名词解释

1.4.1 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 $56^{\circ}C$ 、 $1000r/s$ 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 + n个tag($n \geq 1$)”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 + n个tag($n \geq 1$)”定义了一个时间序列。如 metric为wind, field为speed, tag为“型号=ABC123”、“出厂编号=1234567890”的数据点都属于同一个时间序列。

- tag的key值和value值都相同才算过同一个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。

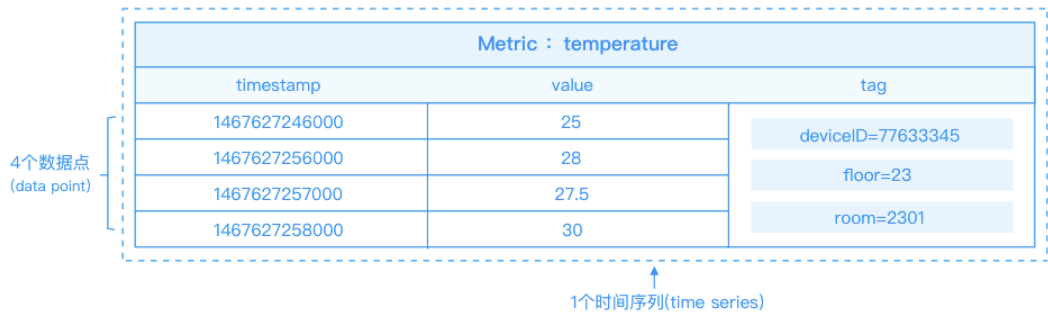
数据库 (database)：一个用户可以有多个数据库，一个数据库可以写入多个“度量”的“数据点”。

1.4.2 实践说明

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deviceID、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。

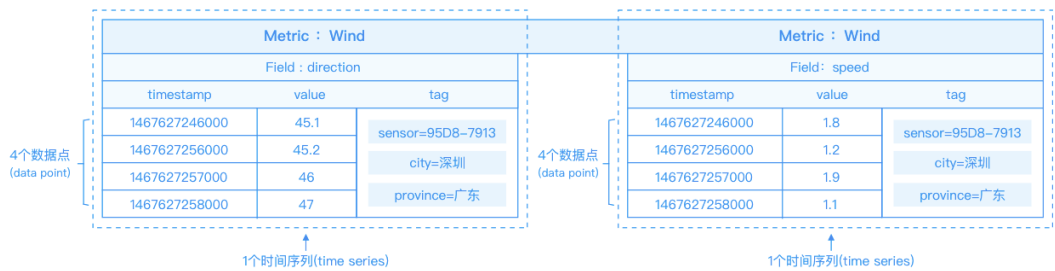


实践二（多域）

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，可以在同一个时间序列下联合查询。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。

| 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 | |

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。

1.5 系统限制

1.5.1 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）。

1.5.2 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。
- String类型的value格式要求： 最大长度为256个字符。 允许任何字符。
- Bytes类型的value格式要求： 最大长度为200个字节。

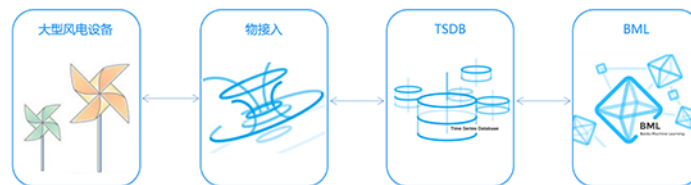
1.6 应用场景

时序数据库拥有广泛的使用场景，在物联网时代，凡是具备数据采集和上报功能的设备都会产生大量的时序数据。通过对时序数据的采集和应用，并配套百度云的大数据和人工智能服务，可以帮助行业用户快速实现万物互联的商业价值。

- 风电机组运维 - 从故障报警到故障预测

风电机组投入使用后将不间断的向电网输送电量，每台大型发电机组每天产生的经济效益可能超过万元。如何更好的维护设备，减少设备的不可用时间，已成为每一个风力发电企业关注的首要问题。在传统情况下，设备的维护主要通过定期巡检、保养等方法。该方法可能带来大量人力、物力成本的浪费，同时也无法保证巡检后的设备能否继续正常运行至下一次巡检。

百度云天工智能物联网平台可帮助企业实现精确的故障预测，如下图所示：



通过应用百度云“天工平台人工智能”解决方案，可提前小时预测风电机组设备故障，故障预测准确率可达，有效的减少了风电机组故障带来的经济损失以及定期巡检造成的资源浪费。

+1290%

第2章 操作指南

2.1 操作准备

2.1.1 介绍

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。

在执行具体操作前，用户需了解TSDB相关的[核心概念](#)。

2.1.2 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：

- 完成百度云[账号注册](#)。

2.1.3 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从白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）导出数据库](#)：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。

2.2 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1～500000百万点/月。
- 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



2.2.1 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。

2.2.2 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[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。

注意：

生成的图表中横轴代表写入数据点的timestamp字段的值。

[查询面板参数设置](#)

查询面板 (test001)

时间范围

* 设置:

时间选择

2015-01-01 00:00:00 - 2015-12-31 00:00:00

相对时间范围

小时

以前, 到

小时

以前

Metrics

temperature

+ 添加

* 名称:

temperature

标签过滤:

+ 添加标签过滤

值过滤:

+ 添加值过滤

分组:

标签

标签: city

+ 添加分组

聚合函数:

+ 添加函数

返回限制:

前

10000

个

生成图表

示范数据库快捷查询

三地月平均温度

三地每月最高温度对比

月平均温度: 北京、上海、广州

月平均风力: 北京、上海、广州

上海市各监测点月平均温度对比

北京市各监测点月平均降雨对比

注意: 快捷查询仅用于示范数据库

生成图表

数据图表: 查询时长: 736毫秒 样本大小: 5460个 数据点数: 364个

— Tag.city=上海 — Tag.city=北京 — Tag.city=广州

查看北京某观测点全年PM2.5大于100的情况 查询面板参数设置

9

时间范围

* 设置:

时间选择

2015-01-01 00:00:00 - 2015-12-31 00:00:00

相对时间范围

小时

以前, 到

小时

以前

Metrics

pm25

+ 添加

* 名称:

pm25

刷新

标签过滤:

标签名: latitude

标签值: 39.913078

×

标签名: longitude

标签值: 116.397180

×

+ 添加标签过滤

值过滤:

+ 添加值过滤

分组:

数值

目标范围: 100

×

+ 添加分组

聚合函数:

+ 添加函数

返回限制:

前

10000

个

生成图表

示范数据库快捷查询

三地月平均温度

三地每月最高温度对比

月平均温度: 北京、上海、广州

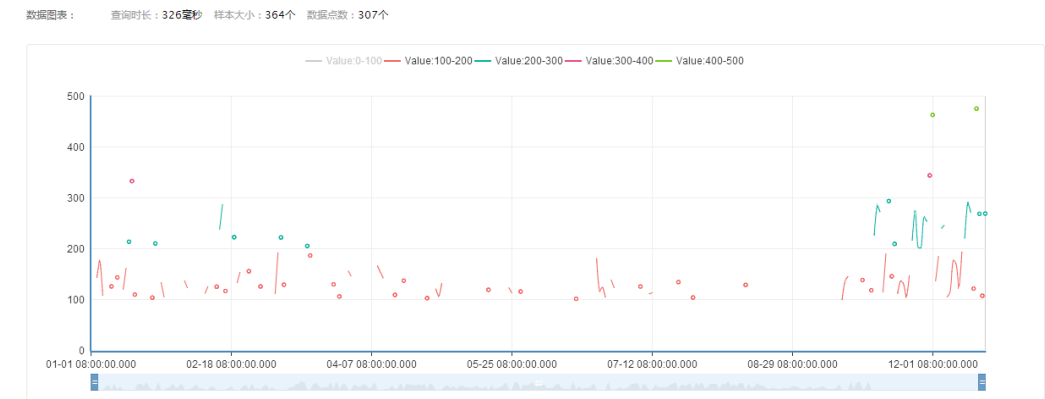
月平均风力: 北京、上海、广州

上海市各监测点月平均温度对比

北京市各监测点月平均降雨对比

注意: 快捷查询仅用于示范数据库

生成图表



2.3 连接数据库

2.3.1 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 <https://cloud.baidu.com/doc/TSDB/API>和Java 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](#)

2.3.2 通过控制台导入数据

详情参考：[数据导入](#)

2.4 生成图表

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：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](#)

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](#)

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“查看面板”，进入操作界面。

查询面板 (test001)

时间范围

设置

时间选择

2015-01-01 00:00:00 ~ 2015-12-31 00:00:00

相对时间范围

小时

以前, 到

小时

以前

Metrics

humidity

+ 添加

名称

humidity

标签过滤

+ 添加标签过滤

值过滤

+ 添加值过滤

分组

+ 添加分组

聚合函数

+ 添加函数

返回限制

前

10000

个

生成图表

示范数据库快速查询

三地月平均温度

三地每月最高温度对比

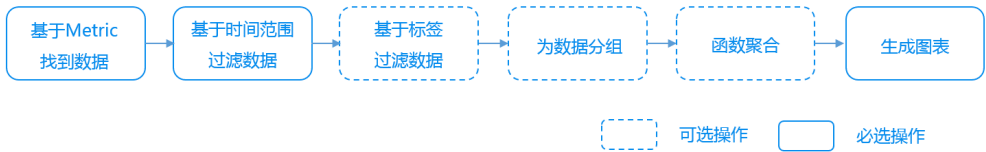
月平均温度; 北京、上海、广州

月平均风力; 北京、上海、广州

上海市各监测点月平均温度对比

北京市各监测点月平均降雨对比

注意: 快捷查询仅用于示范数据库



* 名称：从下拉列表中选择当前数据库中已有的名称，用户可点击“添加”，新增多个进行数据对比。MetricsMetrics

* 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。

* 可基于以下三个维度进行数据分组：

* 标签：将携带指定标签的数据分为一组，可输入多个标签。

* 时间：以给定时间为步长，根据数据的时间戳（），将数据分为指定数量的分组。例如：目标时长为小时，分组个数为，表示将当前数据分成组，每组数据的时间范围为小时。Timestamp112121

* 数值：以给定数值为步长，根据数据的数值（），将数据分组。例如：数值为，则数值在Value100[0,100)范围的数据为第一个分组，数值在[100,200)范围的数据为第二个分组，如此类推。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：

* ：以每个采样时间范围内的的平均值作为结果Avgvalue

* ：以每个采样时间范围内的的标准差作为结果Devvalue

- * : 以每个采样时间范围内的点的数目作为结果Count
- * : 以每个采样时间范围内的第一个作为结果Firstvalue
- * : 以每个采样时间范围内的最后一个作为结果Lastvalue
- * : 对每个采样时间范围内的进行最小二乘法的拟合LeastSquaresvalue
- * : 以每个采样时间范围内的最大值作为结果Maxvalue
- * : 以每个采样时间范围内的最小值作为结果Minvalue
- * : 每个采样时间范围内的第Percentilevaluep百分位数[](http://baike.baidu.com/link?url=0iKg-8Taa9yn8XkmNMn0yFHpEMvA2sP9YXIjFgI12pB4a6P5HMLLzL_YBMjGhkq4oAjxnVOKsRK0Bu8Dq9fwCTnA-d0mDnB2ld_GQI0AZaMAhcblytYg7ttQPg5MeB9dDQeVhgtXurXusIRLrEsfyA)作为结果。
- * : 以每个采样时间范围内的总和作为结果Sumvalue
- * : 以每两个相邻的差值作为结果Diffvalue
- * : 以每个除以一个除数作为结果Divvalue
- * : 以每个乘以一个倍数作为结果Scalevalue

* 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

3. 点击“生成图表”，系统将根据配置筛选数据并生成图表。具体示例请参看[生成示范数据库图表](#)。

2.5 管理数据库

2.5.1 导出数据

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：可到处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或将每个度量的数据导出一个文件。

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或将每个度量的数据导出一个文件。
4. 点击“保存”，使配置生效。

2.5.2 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 |

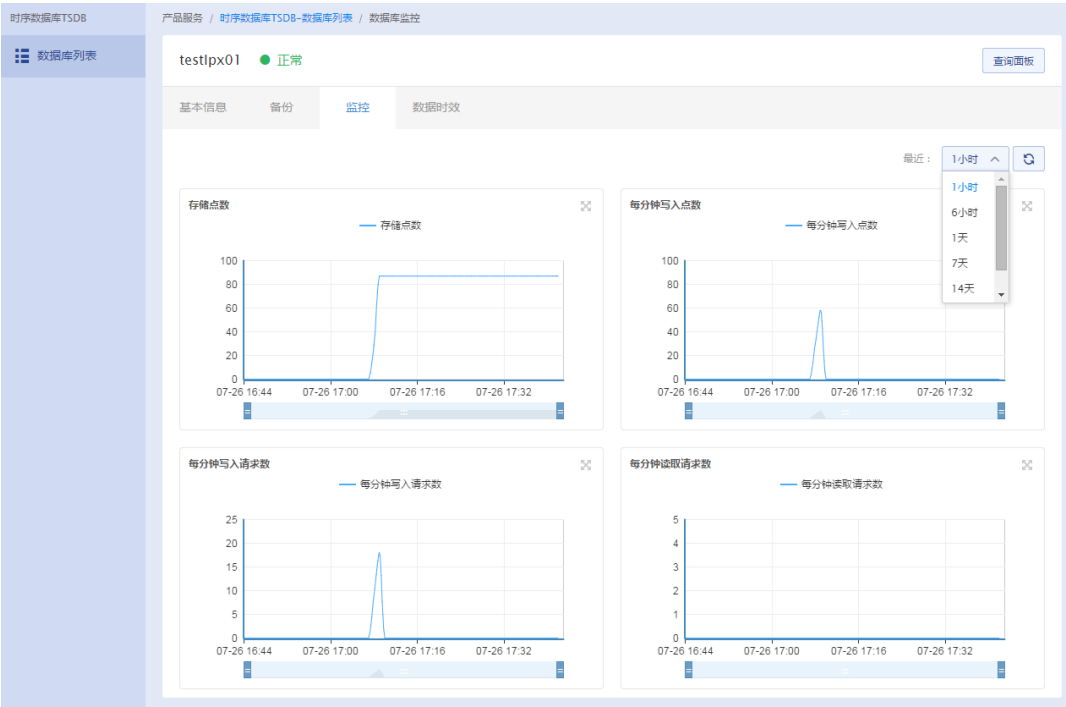
1. 选择“产品服务>时序数据库TSDB>数据库名称”，进入数据库详情页面。
2. 点击“导入>导入备份数据>导入数据”，在弹出对话框中选择指定的Bucket名称和文件，并设置编码格式；点击“确认”，完成导入示范数据操作。



2.5.3 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。
系统默认给出最近1小时内的监控信息，如果需要查看更多内容，可点击左上角下拉菜单，切换时间范围。



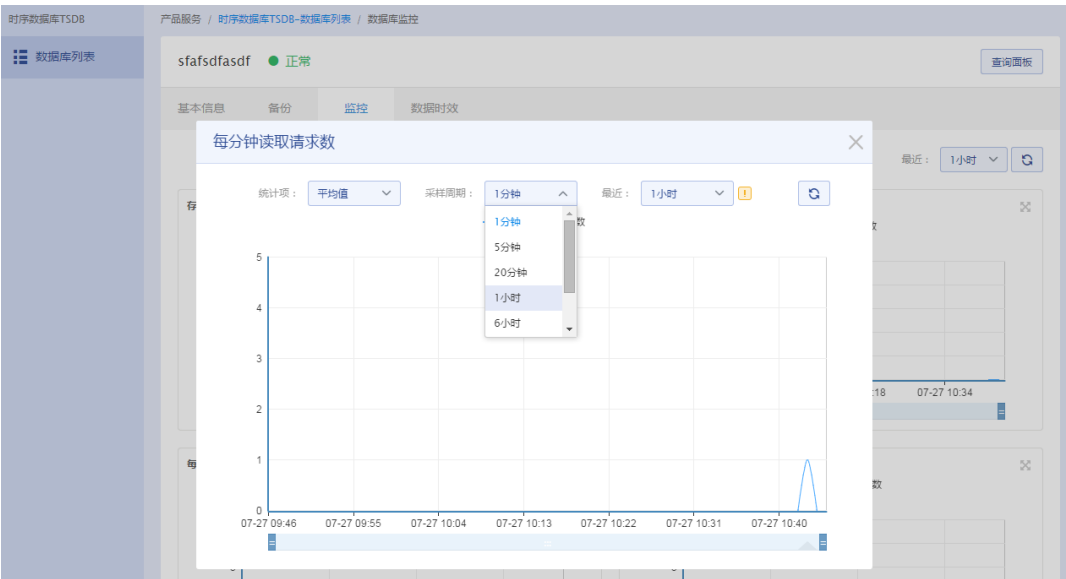
* 存储点数：如果已有存储点数达到套餐上限，新增数据点将无法被保存。此时可通过设置数据时效 [(https://cloud.baidu.com/doc/TSDB/GUIGettingStarted.html#E8.AE.BE.E7.BD.AE.E6.95.B0.E6.8D.AE.E6.97.B6.E6.95.88)]的方法，删除时效范围之外的数据。

* 每分钟写入点数：上限值由套餐规定。

* 每分钟写入请求数

* 每分钟读取请求数用户可以通过点击图表右上方的图标 “

![(figures/TSDB_monitering_03.png)]”，查看详细监控信息，如下图所示：



采样周期包括：分钟、分钟、分钟、小时、小时、小时和天。

152016121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。

2.5.4 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。

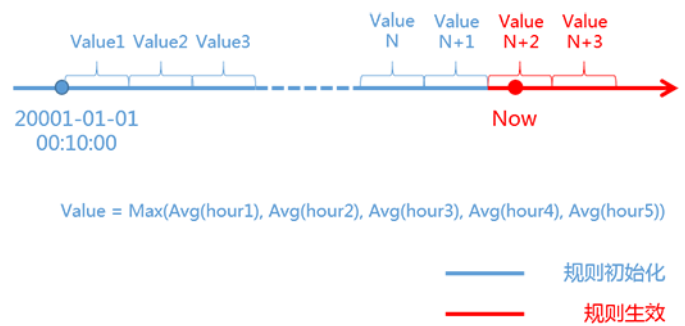
注意：

- 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。
- 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。



- * 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名称，或者选择全部。MetricsMetrics
- * 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，会以此作为开始时间，先对00:10:00 2000-01-01 00:10:00 至今的所有数据点进行聚合（对齐到小时）。如下图所示：5



- * 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个。5
- * 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个。当前支持的聚合函数包括：3

- *：以每个采样时间范围内的的平均值作为结果Avgvalue
- *：以每个采样时间范围内的的标准差作为结果Devvalue
- *：以每个采样时间范围内的的点的数目作为结果Count
- *：以每个采样时间范围内的的第一个作为结果Firstvalue
- *：以每个采样时间范围内的的最后一个作为结果Lastvalue
- *：对每个采样时间范围内的的进行最小二乘法的拟合LeastSquaresvalue
- *：以每个采样时间范围内的的的最大值作为结果Maxvalue
- *：以每个采样时间范围内的的的最小值作为结果Minvalue
- *：每个采样时间范围内的的的第Percentilevaluep百分位

数[](http://baike.baidu.com/link?url=0iKg-8Taa9yn8XkmNMn0yFHpEMvA2sP9YXIjFgI12pB4a6P5HLMlZL_YBMjGhkq4oAjxnVOKsRK0Bu8Dq9fwCTnA-d0mDnB2ld_GQI0AZaMAhcblytYg7ttQPg5MeB9dDQeVhgtXurXusIRLrEsfya)作为结果。

- *：以每个采样时间范围内的的总和作为结果Sumvalue
- *：以每两个相邻的的差值作为结果Diffvalue
- *：以每个除以一个除数作为结果Divvalue
- *：以每个乘以一个倍数作为结果Scalevalue

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

[(<https://cloud.baidu.com/doc/TSDB/GUIGettingStarted.html#.E6.96.B0.E5.BB.BA.E9.A2.84.E5.A4.84.E7.90.86.E8.A7.84.E5.88.99>)]。

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标的具体操作请参看[生成图表](#)。

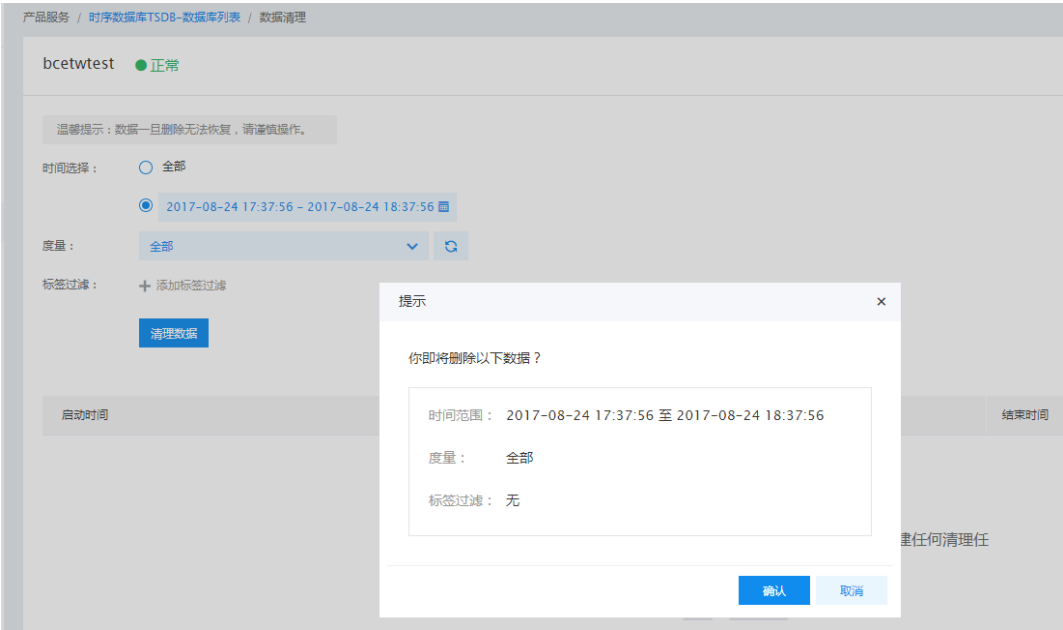
2.5.5 数据清理

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。



2.6 多用户协作

多用户访问控制功能实现了多用户协同开发，项目管理者可以基于数据库为其他开发测试人员开放查看、管理、查询等权限。项目管理者通过IAM子账号方式进行授权，即在主账号下添加子用户的账号并对子用户进行策略管理，子用户可以通过“IAM用户登录链接”访问主用户的资源。

TSDB子账户权限范围说明如下：

| 权限资源粒度 | 权限 |
|--------|--|
| 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 |

配置方法

1. 主账号用户登录后在控制台左下角选择“多用户访问控制”进入用户中心/IAM用户页面。

- 在“子用户管理列表”页面点击新建用户，填写“用户名”，用户名为子用户的昵称，说明主要用于标识用户类型或其它说明。

创建成功后，系统会自动为该用户生成AK/SK，请妥善记录保管。

- 点击“去绑定”，将该用户与子用户的百度账号绑定。

子用户管理列表

IAM用户登录链接: <http://bce.baidu.com>

[+ 新建用户](#) [Q](#) [C](#)

| 用户名 | 百度账号 | 说明 | 状态 | 创建时间 | 操作 |
|-----|-------------------------|----|------|---------------------|---|
| zhw | 未绑定 去绑定 | - | ● 活跃 | 2017-03-23 10:50:25 | 添加权限 禁用 删除 管理 |
| | | - | ● 活跃 | 2016-05-31 17:49:13 | 添加权限 禁用 删除 管理 |
| | | - | ● 活跃 | 2016-05-31 17:46:37 | 添加权限 禁用 删除 管理 |
| | | - | ● 活跃 | 2016-04-05 14:52:54 | 添加权限 禁用 删除 管理 |

- 子用户添加完成后需要对子用户进行策略管理，点击“策略管理” > “创建策略”，自定义TSDB子账户权限。

权限配置

* 服务类型: [时序数据库 TSDB](#)

策略生成方式: [策略生成器](#) [编辑策略文件](#)

* 操作权限: ☒ 数据库查看 ☒ 数据库管理 ☒ 数据查询 ☐ 数据修改 (写入/修改/删除) [权限说明](#)

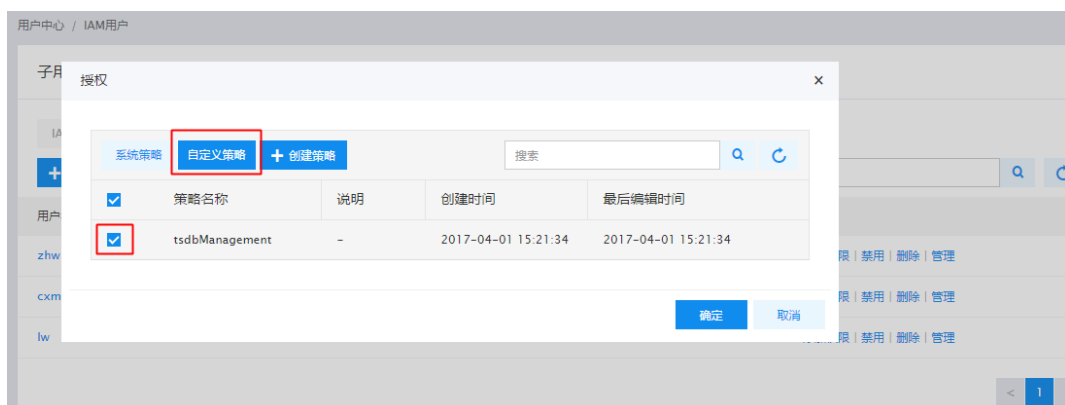
* 资源选择: [华南 - 广州](#)

[Q](#) 已选择1个资源

| <input type="checkbox"/> 资源名称 | 数据库ID | 描述 |
|---|-------------------|----|
| <input type="checkbox"/> bcewtest | tsdb-s36cbjydb1t1 | - |
| <input checked="" type="checkbox"/> test001 | tsdb-1bx934pqex2s | - |

[完成](#) [取消](#)

- 返回子用户管理列表，点击“添加权限”，在弹出对话框中点击“自定义策略”，选择步骤4中创建的策略。



- 子用户的账户和权限添加完成后，用户可以直接通过子用户管理列表的“IAM用户登录链接”登录主账号的TSDB资源，登录成功后用户可根据设定的权限享有对主账户资源的操作和查看权限。

2.7 对接hive sql

2.7.1 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：<http://iot-tsdB.gz.bcebos.com/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。

2.7.2 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，如果在本地，请使用本地地址，
    如jarbmrbosbos://path/to/hive-tsdB-handler-all.jar */
CREATE EXTERNAL TABLE {\color{emcolor}\textbf{wind}}({\color{emcolor}\textbf{time}} bigint, {\color{emcolor}\textbf{value}} double, {\color{emcolor}\textbf{city}} string) /* 创建表 */
STORED BY 'com.baidubce.tsdB.hive.storage.TsdBStorageHandler' /* 设置为storageTsdBStorageHandler */
TBLPROPERTIES (
" tsdB.metric_name" = " wind" , /* 名字，默认表名则被当为Metricmetric */
" tsdB.timestamp_name" = " time" , /* 对应的列名，默认对应的列名Timestamptime */
" tsdB.field_names" = " value" , /* 列表，只需填写中用到的，多个通过逗号分割 fieldtablefield */
" tsdB.tag_keys" = " city" , /* 列表，只需填写中用到的，这个示例中可以去掉改行，但如果中包含大写字母，则必须填写tagKeytabletagKey */
" tsdB.endpoint" = " ENDPOINT" , /* 实例的TSDBendpoint */
" tsdB.access_key" = " AK" , /* AK */
" tsdB.secret_key" = " SK" /* SK */
);

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


2.7.3 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，如果在hive中，请使用地址，
如jarbmrbosbos://path/to/hive-tsdb-handler-all.jar */
CREATE EXTERNAL TABLE {\color{emcolor}\textbf{WindSpeed}}({\color{emcolor}\textbf{time}}
bigint, {\color{emcolor}\textbf{x}} double, {\color{emcolor}\textbf{y}} double) /* 创建
表 */
STORED BY 'com.baidubce.tsdb.hive.storage.TsdbStorageHandler' /* 设置
为storageTsdbStorageHandler */
TBLPROPERTIES (
"tsdb.metric_name" = "WindSpeed", /* 名字，默认表名
则被当为Metricmetric */
"tsdb.timestamp_name" = "time", /* 对应的列名，默
认为对应的列名Timestamptime */
"tsdb.field_names" = "x,y", /* 列表，多个通过
逗号分割 fieldfield */
"tsdb.endpoint" = "ENDPOINT", /* 实例
的TSDBendpoint */
"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，如果在H中，请使用地址，
如jarbmrbosbos://path/to/hive-tsdb-handler-all.jar */
CREATE EXTERNAL TABLE {\color{emcolor}\textbf{vehicle}}({\color{emcolor}\textbf{time}} bigint, {\color{emcolor}\textbf{speed}} bigint, {\color{emcolor}\textbf{carId}} string) /* 创建
表 */
STORED BY 'com.baidubce.tsdb.hive.storage.TsdbStorageHandler' /* 设置
为storageTsdbStorageHandler */
TBLPROPERTIES (
"tsdb.metric_name" = "vehicle", /* 名字，默认表名
则被当为Metricmetric */
"tsdb.timestamp_name" = "time", /* 对应的列名，默
认为对应的列名Timestamptime */
"tsdb.field_names" = "speed", /* 列表，多个通过
逗号分割 fieldfield */
"tsdb.tag_keys" = "carId", /* 列表，多个通过
逗号分割 tagtag */
"tsdb.endpoint" = "ENDPOINT", /* 实例
的TSDBendpoint */
"tsdb.access_key" = "AK", /* AK */
"tsdb.secret_key" = "SK" /* SK */
);

/* 为 "" 的车辆在年月每天的停止时长ID123201712 */
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);

/* 为 "" 的车辆在年月每天的运行时长ID123201712 */
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);

/* 为 "" 的车辆在年月每天的运行时长ID123201712 */
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 |

第3章 产品定价

3.1 包年包月计费

您可以通过[TSDB价格计算器](#)快速获取价格。

TSDB为预付费服务，实行阶梯定价，按照每月最高可写入数据点数计费。用户在购买服务前需对每月写入数据点数进行预估，选择购买相应的配额。

每个用户可以创建一个免费TSDB实例，享有每月100万数据点的免费额度（可升级至更高额度，具体费用参见费率表）。

购买前需保证账户无欠款。

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。

费率表

注意：

- 当月用量超过所购买的额度后，新增数据点将不会被接收处理。
- 当月剩余额度不累计至次月。
- 不同的数据点额度，会有相应的时间序列额度限制，参考下表。
- 数据点和时间序列的概念请参考[名词解释](#)，时间序列的限制将于2018年2月1日执行。
- 2017年6月1日起，5年内数据存储完全免费，5年后实行有偿收费原则。

| 规格(百万点/月) | 时间序列限制(个) | 月单价(元/百万点/月) | 年单价(元/百万点/年) |
|---------------|------------|--------------|--------------|
| 0-1 | 1,000 | ¥4.00 | ¥40.00 |
| 2-10 | 10,000 | ¥3.60 | ¥36.00 |
| 11-50 | 50,000 | ¥3.20 | ¥32.00 |
| 51-100 | 100,000 | ¥2.80 | ¥28.00 |
| 101-500 | 500,000 | ¥2.40 | ¥24.00 |
| 501-5,000 | 5,000,000 | ¥2.00 | ¥20.00 |
| 5,001-500,000 | 10,000,000 | ¥2.00 | ¥20.00 |

| 规格(百万点/月) | 时间序列限制(个) | 月单价(元/百万点/月) | 年单价(元/百万点/年) |
|-----------|-----------|--------------|--------------|
| 500,000+ | 请提交工单申请 | 请提交工单申请 | 请提交工单申请 |

计算公式

以下通过一个实例介绍TSDB的费用计算方法。

以用户选择购买每月写入1亿（即100百万）数据点额度为例，用户需要支付的月费用计算公式为：

$$4.00 \times 1 + 3.60 \times 9 + 3.20 \times 40 + 2.80 \times 50 = 304.4 \text{元/月}$$

第4章 Java SDK文档

4.1 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](#)。

4.2 安装SDK工具包

运行环境

Java SDK工具包可在jdk1.6、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

| | |
|--------------------------------|---------------------------------|
| auth | //签名相关类BCE |
| http | //的通信相关类BCEHttp |
| internal | //内部类SDK |
| model | //公用类BCEmodel |
| services | |
| tsdb | //服务相关类TSDB |
| model | //内部，如或TSDBmodelRequestResponse |
| TsdbClient.class | //客户端入口类TSDB |
| TsdbConstants.class | //特有的常量TSDB |
| util | //公用工具类BCE |
| BceClientConfiguration.class 对 | //的的配置BCEHttpClient |
| BceClientException.class | //客户端的异常类BCE |
| BceServiceException.class 与 | //服务端交互后的异常类BCE |
| ErrorCode.class | //通用的错误码BCE |
| Region.class | //提供服务的区域BCE |

4.3 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。

4.4 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 //协议HTTPS不设置，默认使用;协议HTTP
    .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](#)。

参数说明

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

| 参数 | 说明 |
|----------------------------|--------------------------|
| connectionTimeo utInMillis | 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 |

| 参数 | 说明 |
|---------------------------------------|-------------------------|
| proxyPassword | 代理服务器验证的密码 |
| proxyPort | 代理服务器端口 |
| proxyPreemptive AuthenticationEnabled | 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 |

4.5 写入操作

4.5.1 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 = "cpu_idle" ; // metric
String TAG_KEY = "server" ; // 标签名称
String TAG_VALUE = "server1" ; // 标签值
String FIELD = "temperature" ; // 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
    .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，不同的域。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。

```

// 添加类型数据点Double
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); // 添加类型数据点Double

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

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

```

4.5.2 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 = " cpu_idle" ; // metric
String TAG_KEY = " server" ; // 标签名称
String TAG_VALUE = " server1" ; // 标签值
String FIELD_1 = " temperature" ; // 域1
String FIELD_2 = " humidity" ; // 域2
long TIME = System.currentTimeMillis(); // 时间

// 添加的数据点FIELD_1
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); // 指定时间写入类型数据
        点Double

// 添加的数据点FIELD_2
Datapoint datapoint2 = new Datapoint()
        .withMetric(METRIC) // 设置Metric需要和,的一
        样FIELD1
        .withField(FIELD_2) // 设置域2
        .addTag(TAG_KEY, TAG_VALUE) // 设置, 需要和的一
        样TagFIELD1
        .addLongValue(TIME, 10L); // 指定时间添加类型数据
        点, 时间需要和的一样LongFIELD_1
tsdbClient.writeDatapoints(Arrays.asList(datapoint1, datapoint2));

```

4.6 查询操作

4.6.1 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);
}

```

4.6.2 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());
}

```

4.6.3 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
                               的tagmetric

// 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);
    }
}
```

4.6.4 查询数据点

如下代码可以查询数据点：

```
String METRIC = " cpu_idle" ;
String FIELD = " temperature" ;

// 构造查询对象
List<Query> queries = Arrays.asList(new Query() // 创建对象Query
    .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 () // 创建对象 Filters
        .withRelativeStart(" 2 seconds ago" )) // 设置相对的开始时间，这里设置为秒前2
    .withLimit(100) // 设置返回数据点数目限制
    .addAggregator(new Aggregator() // 创建对象Aggregator
        .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\t\tGroupInfo:" );
        System.out.println(" \t\t\t\tName:" + groupInfo.getName());
    }
    System.out.println(" \t\t\tTimeAndValue:" );
    for(TimeAndValue timeAndValue : group.getTimeAndValueList()) {
        if (timeAndValue.isDouble()) {
            System.out.println(" \t\t\t\t[" + timeAndValue.getTime() + " , " + timeAndValue.
                getDoubleValue() + " ]" );
        } else if (timeAndValue.isLong()) {
            System.out.println(" \t\t\t\t[" + timeAndValue.getTime() + " , " + timeAndValue.
                getLongValue() + " ]" );
        } else {
            System.out.println(" \t\t\t\t[" + timeAndValue.getTime() + " , " + timeAndValue.
                getStringValue() + " ]" );
        }
    }
}
}
}

```

多域查询数据点 如下代码可以查询数据点：

```

String METRIC = " cpu_idle" ;
String FIELD1 = " temperature" ;
String FIELD2 = " power" ;
String TAG= " city" ;

// 构造查询对象
List<String> fields = Arrays.asList(FIELD1, FIELD2)
List<Query> queries = Arrays.asList(new Query() // 创建对象Query
    .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, 和冲突, 不能同时设置, 返回结果的顺序和请求的顺序相
        同fieldFIELD
    .withTags(Arrays.asList(TAG)) // 设置查询的列表, 不设置
        表示不查询, Tag
    . withFilters (new Filters() // 创建对象 Filters
        .withRelativeStart(" 5 seconds ago" ) // 设置相对的开始时间,
            这里设置为秒前5
        .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
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\t\tGroupInfo:" );
            System.out.println(" \t\t\t\tName:" + groupInfo.getName());
        }
        System.out.println(" \t\t\tTimeAndValue:" );
        for(TimeAndValue timeAndValue : group.getTimeAndValueList()) {
            System.out.println(" \t\t\t\t[" + timeAndValue.getTime());
            for (int index = 0; index < fields.size(); index++) {
                System.out.println(" , " );
                if (timeAndValue.isDouble(index)) {
                    System.out.println(timeAndValue.getDoubleValue(index));
                } else if (timeAndValue.isLong(index)) {
                    System.out.println(timeAndValue.getLongValue(index));
                } else {
                    System.out.println(timeAndValue.getStringValue(index));
                }
            }
            System.out.println(" ]" );
        }
    }
}

```

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 = " cpu_idle" ;
String FIELD = " temperature" ;
String TAG_1_KEY = " server" ;
String TAG_2_KEY = " city" ;// 构造查询对象
List<Query> queries = Arrays.asList(new Query()                // 创建对象Query
    .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(TAG_1_KEY, TAG_2_KEY))            // 设置需要同时返回的tag 列表key
    .withFilters (new Filters ()                             // 创建对象 Filters
        .withRelativeStart(" 2 seconds ago" ))              // 设置相对的开始时间，这里设置为秒前2
    .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:" );
        for (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) +
                        "]" );
            }
        }
    }
}

```

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 = " cpu_idle" ;
String FIELD = " wind" ;

// 构造查询对象
List<Query> queries = Arrays.asList(new Query() // 创建对象Query
    .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() // 创建对象 Filters
        .withRelativeStart(" 5 seconds ago" ) // 设置相对的开始时间，
        // 这里设置为秒前5
        .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:" );
        for(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() + " ]" );
            }
        }
    }
}

```

分页查询原始数据点 如下代码可以分页查询原始数据点:

```

String METRIC = " cpu_idle" ;
String FIELD = " wind" ;

// 构造查询对象
Query query = new Query()                                // 创建对象Query
    .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 ()                          // 创建对象 Filters
        .withRelativeStart(" 1 week ago" )               // 设置相对的开始时间,
        // 这里设置为周前1
        .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:" );
    for(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 if (timeAndValue.isString()) {
            System.out.println(" \t\t\t[" + timeAndValue.getTime() + " , " + timeAndValue.
                getStringValue() + " ]" );
        } else if (timeAndValue.isBytes()) {
            System.out.println(" \t\t\t[" + timeAndValue.getTime() + " , " + timeAndValue.
                getBytesValue() + " ]" );
        }
    }
}
// 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());
}

```

查询数据时使用标签过滤 示例代码如下：

```

String METRIC = " cpu_idle" ;
String FIELD = " temperature" ;

// 构造查询对象
List<Query> queries = Arrays.asList(new Query()                                // 创建对象Query
    .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 ()                                                // 创建对象 Filters
        .withRelativeStart(" 5 seconds ago" )                                // 设置相对的开始时间，这里设置为秒前5
    )
);

```

```

        .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() // 创建对象TagFilter
            .withTag(" tag1" ) // 设置要过滤的tagKey
            .addNotIn(" value2" )); // 设置不允许出现的。还可以设置允许出现的, 或者设置匹配。tagValueTagValueLike

// 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:" );
        for(TimeAndValue timeAndValue : group.getTimeAndValueList()) {
            if (timeAndValue.isDouble()) {
                System.out.println(" \t\t\t\t" + timeAndValue.getTime() + " ," + timeAndValue.
                    getDoubleValue() + " ]" );
            } else if (timeAndValue.isLong()) {
                System.out.println(" \t\t\t\t" + timeAndValue.getTime() + " ," + timeAndValue.
                    getLongValue() + " ]" );
            } else if (timeAndValue.isString()) {
                System.out.println(" \t\t\t\t" + timeAndValue.getTime() + " ," + timeAndValue.
                    getStringValue() + " ]" );
            } else if (timeAndValue.isBytes()) {
                System.out.println(" \t\t\t\t" + timeAndValue.getTime() + " ," + timeAndValue.
                    getBytesValue() + " ]" );
            }
        }
    }
}
}

```

```

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

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

// 添加: 通过添加Tagsmap
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);
```

// 对型数据进行过滤，支持的比较符为long，，，，><>=<== 和!=

```
Filters filters1 = new Filters()
    .withAbsoluteStart(System.currentTimeMillis() - 5000)
    .withAbsoluteEnd(System.currentTimeMillis() - 1000)
    .withValue(Filter.createValueFilter(ValueFilter.LESS_OR_EQUAL, 100L)); // 过滤条件为 " <= 100"
```

// 对型数据进行过滤，支持的比较符为double，，，，><>=<== 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"
```

// 对型数据进行过滤，支持的比较符为String，，，，><>=<== 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' "
```

// 将数据与标签的值进行比较过滤，支持的比较符为tagKey1，，，，><>=<== 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"
```

// 对所有域数据使用统一进行过滤，支持的比较符为value，，，，><>=<== 和!=

```
Filters filters1 = new Filters()
    .withAbsoluteStart(System.currentTimeMillis() - 5000)
    .withAbsoluteEnd(System.currentTimeMillis() - 1000)
    .withValue(Filter.createValueFilter(ValueFilter.LESS_OR_EQUAL, 100L)); // 过滤条件为 " <= 100"
```

// 对多域数据使用不同的进行过滤，支持的比较符为value，，，，><>=<== 和!=

```
FieldFilter fieldFilter1 = new FieldFilter(field1, ValueFilter.createValueFilter("<", 100));
FieldFilter fieldFilter2 = new FieldFilter(field2, ValueFilter.createValueFilter(">", 200));
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](#)。

4.7 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
    的tagmetric
String FIELD = " temperature" ;

// 构造查询对象
List<Query> queries = Arrays.asList(new Query()                  // 创建对象Query
    .withMetric(METRIC)                                         // 设置metric
    .withField(FIELD)                                           // 设置域，不设置表示查
        询默认域
    .withFilters (new Filters ()                                // 创建对象 Filters
        .withRelativeStart(" 5 seconds ago" )                  // 设置相对的开始时间，
            这里设置为秒前5
        .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()                             // 创建对象Aggregator
        .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
```

4.8 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
bool useGzip = false; // 不使用gzip

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

4.9 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。

4.9.1 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 ENDPOINT = "tsdb.gz.baidubce.com"; // 注意：与新建时使用的不
    同TsdbClientendpoint

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

4.9.2 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 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)
.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](#)。

4.9.3 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。

4.9.4 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](#)。

4.9.5 获取时序数据库实例列表

示例代码如下：

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

4.10 版本说明

4.10.1 v0.10.24

- 标签过滤支持like匹配
- 按值过滤支持与标签值比较

4.10.2 v0.10.23

- 支持标签过滤
- 支持or条件

4.10.3 v0.10.22

- 支持分页查询
- 支持固定值插值插值函数

4.10.4 v0.10.20

- 支持多域同时查询
- 支持查询时对数据进行插值

4.10.5 v0.10.19

支持对单域的数据的查询、写入

4.10.6 v0.10.17

支持创建/删除/查询时序数据库实例

4.10.7 v0.10.12

- 查询支持按值过滤
- 数据点的值支持byte数组

4.10.8 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

4.10.9 v0.10.7

首次发布。

第5章 Node SDK文档

5.1 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[注册百度云账户](#)并[登录](#)，在[安全认证页面](#) 获取 AK/SK。

5.2 安装SDK工具包

版本检测

1. `{\color{emcolor}\textbf{\$ npm -v}}` // 查看npm是否正确安装
2. `{\color{emcolor}\textbf{\$ node -v}}` // 检查node版本

支持的Node.js版本

- 4.x
- 5.x

安装SDK工具包

首先使用npm或者bower工具安装SDK的工具包。

bower工具：

```
\$ bower install bce-sdk-js
\$ cd <bower_components/bce-sdk-js>
\$ npm install
```

npm工具：

```
\$ npm install bce-sdk-js
```


然后在程序中使用：

```
// 需要使用转义为babel 关键字require
// 或者使用 var TsdbDataClient = require(' ./tsdb_data_client' );
import {TsdbDataClient} from ' bce-sdk-js' ;

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

5.3 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Tsdb-Client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。

5.4 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：

```
// 引入所需的（数据接口），需要使用转义为clientbabel 关键字require
// 或者使用 var TsdbDataClient = require(' ./tsdb_data_client' );
import {TsdbDataClient} from ' bce-sdk-js' ;

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);
```

5.5 写入操作

5.5.1 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"
    },
    "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。

5.5.2 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"
    },
    "timestamp": Math.round(new Date().getTime() / 1000),
    "value": 51
  },
  {
    "metric": "cpu_idle3",
```

```

    "field": "field1",
    "tags": {
      "host": "server1",
      "rack": "rack1"
    },
    "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

```

5.6 查询操作

5.6.1 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'
  ]
}

```

5.6.2 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"
  ]
}
```

5.6.3 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'
  ]
}
```

5.6.4 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.getDapoints(<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getDapoints()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，以便获取后面的数据marker
  client .getDatapoints([query]) // 获取数据
    .then(deealWithResponse) // 设置处理结果的callback
    .catch(dealWithError); // 设置处理的errorcallback
};

```

```

var dealWithResponse = 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(dealWithResponse) // 设置处理结果的callback
    .catch(dealWithError); // 设置处理的errorcallback

```

5.7 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，您需要提供根据需求构建的查询列表、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"
                }
            ]
        }
    }
]

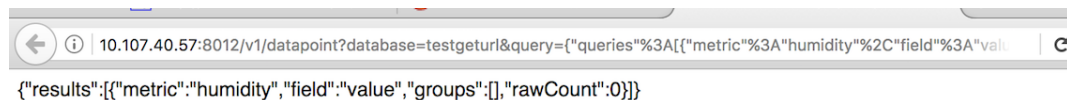
```

```
    }  
  },  
  " 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}]}
```

5.8 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(new Date().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

```

5.9 管理接口

5.9.1 新建TsdbAdminClient

```

var TsdbAdminClient = require('./tsdb_admin_client');

const config = {
  endpoint: <Endpoint>,          // 管理接口的域名地址，形如tsdb.{region}.baidubce.，注意：与新建时使用的不同。comTsdbClientendpoint
  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);

```

5.9.2 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 dbName = "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 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, dbName, ingestDataPointsMonthly, purchaseLength,
    description, couponName)
    .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](#)。

5.9.3 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: {}
```

5.9.4 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'
}
```

5.9.5 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" : " 物可视Ownedby,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"
  }
]
```

5.10 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 |

5.11 版本说明

首次发布。

第6章 API参考

6.1 介绍

6.1.1 名词解释

在使用TSDB API前，用户需了解TSDB相关的[核心概念](#)。

6.1.2 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

| 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 |

公共请求头

| 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 |

公共响应头

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"

```
" 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，是百度云中的重要概念，请参考[区域选择说明](#)。百度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](#)服务域名为（其中{database-name}为数据库名称，{database-id}为数据库ID，可以在控制台中获取）：

| 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 |

6.2 数据API接口说明

6.2.1 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 |
|------------|-----------------|------|--------------------------------|
| datapoints | List<Datapoint> | 必须 | datapoint列表, 由Datapoint对象组成的数组 |

Datapoint对象

| 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 | 可选 | 目前支持Long/Double/ String/ Bytes。代表value字段的类型, 如果不填会根据解析出来的类型为准。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。 |

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|-----------|-------------------|------|--|
| 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，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](#)

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

返回示例

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

6.2.2 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"
  ]
}
```

6.2.3 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"
    }
  }
}
```

6.2.4 获取tag列表

| 方法 | API | 说明 |
|-----|-------------------------|------------------------------|
| GET | /v1/metric/{metric}/tag | 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263 - 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"]
  }
}
```

6.2.5 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 |
| 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 |
| 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263 - 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 |

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|--------|-------------------|------|--|
| 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 |

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|-------|--------|------|---|
| 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；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 |

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、Least-Squares、Max、Min、Percentile、Sum、Diff、Div、Scale、Rate、AdjacentUnique |
| sampling | String | 可选 | 采样的时间长度，如“10 minutes”，name为 Avg、Dev、Count、First、Last、Least-Squares、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 |

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|----------|--------|------|--------------------|
| timeUnit | String | 可选 | 时间单位，name为Rate时必须填 |

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 |
| 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。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 |

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|-------|-------------------|------|----------------------------|
| value | Int/Double/String | 可选 | 固定值插值所插入的固定值，type为Fixed时必填 |

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 | 原始的数据点数目 |
| 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 |
| presamplingRule Id | 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"
  }],
  " 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" : [ " key1" , " key2" ],

```

```

    " 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" : [ " key1" , " key2" ],
    " rawCount" : 1000,
    " groups" : [{
      " groupInfos" : [{
        " name" : " Tag" ,
        " tags" : {
          " rack" : " rack1"
        }
      }],
      " values" : [
        [1465718968506, 10, 1.0, " val11" , " val21" ],
        [1465718985346, 12, 2.0, " val12" , " val22" ],
        [1465718992879, 15, 11.0, " val13" , " val23" ]
      ]
    }
  ]
}

```

```
    },  
  }  
}
```

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" : [ " key1" , " key2" ],  
    " filters" : {  
      " or" : [{  
        " fields" : [{  
          " field" : " field1" ,  
          " value" : " >= 10"  
        }, {  
          " field" : " field2" ,  
          " value" : " <= 10"  
        }],  
        " start" : " 1 hour ago"  
      }],  
      " tags" : {  
        " tag" : " key1" ,  
        " in" : [ " val1" , " val2" ]  
      },  
      " 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": ["key1", "key2"],
    "rawCount": 1000,
    "groups": [{
      "groupInfos": {
        "name": "Tag",
        "tags": {
          "rack": "rack1"
        }
      },
      "values": [
        [1465718968506, 10, 1.0, "val11", "val21"],
        [1465718985346, 12, 2.0, "val12", "val22"],
        [1465718992879, 15, 11.0, "val13", "val23"]
      ]
    }
  ]
}
```

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"]
      ]
    }]
  }]
}
```

6.3 管理API接口说明

6.3.1 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 |

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|----------------|--------|------|-----------|
| 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.iot.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-5atue8m3xsxsv" ,
  " charge" : 29.9,
  " expiredTime" : " 2016-07-06T11:52:41Z" ,
  " orderId" : " d04da3c16bd042af9e916a75bb1fa19g"
}
```


6.3.2 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-5atue8m3sxs HTTP/1.1
Host: tsdb.iot.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
```

6.3.3 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 |
| 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-5atue8m3sxsx 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
```

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-5atue8m3sxsx" ,
  " databaseName" : " test" ,
  " description" : " test" ,
  " endpoint" : " test.tsdb.iot.gz.baidubce.com" ,
  " quota" : {
    " ingestDataPointsMonthly" : 1
  },
  " status" : " Active" ,
  " autoExport" : true,
  " createTime" : " 2016-06-06T14:21:01Z" ,
  " expiredTime" : " 2016-07-06T14:21:01Z"
}
```

6.3.4 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.iot.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-5atue8m3sxs",
    "databaseName": "test",
    "description": "test",
    "endpoint": "test.tsdb.iot.gz.baidubce.com",
    "quota": {
      "ingestDataPointsMonthly": 1
    },
    "status": "Active",
    "autoExport": true,
    "createTime": "2016-06-06T14:21:01Z",
    "expiredTime": "2016-07-06T14:21:01Z"
  }]
}
```

6.4 聚合函数

6.4.1 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 |

6.4.2 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 |

6.4.3 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 |

6.4.4 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 |

6.4.5 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 |

6.4.6 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 |

6.4.7 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 |

6.4.8 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 |

6.4.9 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% |

6.4.10 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 |

6.4.11 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 |

6.4.12 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 |

6.4.13 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 |

6.4.14 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 |

6.4.15 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, 1, 1, 3”，相邻值去重后的结果为“1, 2, 1, 3” | Number/String/Bytes |

6.5 分组方式

6.5.1 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 |

6.6 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 |
| months | 月 |
| month | 月 |
| years | 年 |
| year | 年 |

6.7 附录

6.7.1 错误状态码

| Code | HTTP状态码 | 说明 |
|--------------|---------------|-----------------|
| AccessDenied | 403 Forbidden | 没有权限访问该database |

| Code | HTTP状态码 | 说明 |
|-------------------------|----------------------------|---------------------------------------|
| 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 |
| InternalServerError | 500 Internal Server Error | 系统内部错误，请稍后重试 |

6.7.2 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 |

6.8 更新历史

2017-12-05

查询data point接口增加分页和tag like功能。

第7章 时序数据库TSDB服务等级协议（SLA）

服务可用性

- TSDB的服务可用性：不低于99.9%。
- 可用性按服务周期统计，一个服务周期为一个自然月，如不满一个月不计算为一个服务周期。
- 不可用时间：TSDB所提供的服务在连续的5分钟或更长时间内不可使用方计为不可用时间，不可使用的服务时间低于5分钟的，不计入不可用时间。TSDB不可用时间不包括日常系统维护时间、由用户原因、第三方原因或不可抗力导致的不可用时间。

TSDB服务可用性的计算方法如下：

- 月总请求次数低于10万的用户不做统计；
- 失败请求数 = 返回错误码为5xx的请求数量；
- 估算的失败请求数 = 前7天用户单位时间请求数的平均值 × 服务不可用时间（当出现服务不可用且无失败请求返回时）；
- TSDB的可用性 = $1 - (\text{失败请求数} + \text{估算的失败请求数}) / \text{总请求数} (\text{正常请求数} + \text{失败请求数} + \text{估算的失败请求数})$ 。

数据持久性

1. TSDB的数据持久性不低于99.9999999%。数据持久性按服务周期统计，一个服务周期为一个自然月，如不满一个月不计算为一个服务周期。
2. 按TSDB的数据持久性计算，如用户在TSDB存储十亿个数据点，每月最多只有1个数据点发生数据丢失的可能性。

数据可销毁性

1. 用户主动删除的数据，该数据将无法复原。
2. 当TSDB的服务器整机下架或故障硬盘下架时，会相应的进行数据清除工作，相关数据将无法复原。

数据可迁移性

用户可以通过API, SDK等工具对存储在TSDB上的数据进行读写操作, 并根据需要进行迁移。TSDB不会对用户上传的数据做任何的修改。

数据私密性

- 用户存储在TSDB上的数据, 在未经用户合法授权的情况下, 其他用户无法访问其数据。
- 百度云为用户分配Access Key ID/Secret Access Key密钥对, 从TSDB访问接口上进行权限控制和隔离, 保障用户数据的私密性。
- 百度云的所有运维操作都会被记录并保存。

数据知情权

除应当地法律法规、或政府监管部门的监管、审计要求, 用户的所有数据、应用及行为日志不会提供给第三方。用户的所有数据不会被存在境外的数据中心, 也不会被用于境外业务或数据分析。目前百度云的数据中心位于北京。用户的数据存储在用户选择的区域(数据中心)。TSDB的数据都存储在百度中国的数据中心。

业务可审查性

百度不会在未经合法用户授权时, 公开、编辑或透露用户的个人信息及保存在百度云时序数据库中的内容, 除非有下列情况:

- 有关法律、法规规定或百度云时序数据库合法服务程序规定;
- 在紧急情况下, 为维护用户及公众的权益;
- 为维护百度的商标权、专利权及其他任何合法权益;
- 其他依法需要公开、编辑或透露个人信息的情况。

服务变更、终止条款

鉴于网络服务的特殊性, 用户了解并同意, TSDB服务有权变更、中断或终止部分或全部的TSDB服务。如需变更、中断或终止服务, TSDB服务会通过百度云站内信, 邮件, 或者短信的方式对用户进行通知。

服务故障恢复能力

百度云为付费用户的云服务提供7×24小时的运行维护, 并以在线工单和电话报障等方式提供技术支持, 具备完善的故障监控、自动告警、快速定位、快速恢复等一系列故障应急响应机制。

服务赔偿条款

赔偿范围

因百度云故障导致时序数据库无法正常使用。

赔偿标准<\b>

故障时长 = 故障解决时间 - 故障开始时间。

TSDB服务的实例以代金券的形式赔付，按单个实例故障时长100倍赔偿。

单个实例的赔付金额 = 故障前24小时平均每小时费用 / 60 × 故障时间 × 100。

说明<\b>

1. 如果用户的使用时间小于24小时，按实际使用时长的平均值计算费用。
2. 故障时长按分钟计算。
3. 单个实例的单次赔付金额不超过该实例累计实付金额。

第8章 常见问题

8.1 数据库创建及设置

8.1.1 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。

8.1.2 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。

8.1.3 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。

8.2 数据点查询

8.2.1 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：

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：

有关CAST函数的介绍，请参看[常用SQL函数](#)。

8.3 数据点写入

8.3.1 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](#)