TDengine用户手册

适用于企业版 2.0

（手册版本 2.1.6.0）

1. TDengine概述

随着数据通讯成本的急剧下降，以及各种传感技术和智能设备的出现，从手环、共享自行车、出租车、智能电表、环境监测设备到电梯、大型设备、工业生产线等都在源源不断地产生海量的实时数据并发往云端。这些海量数据是企业宝贵的财富，能够帮助企业实时监控业务或设备的运行情况，生成各种维度的报表，而且通过大数据分析和机器学习，对业务进行预测和预警，帮助企业进行科学决策、节约成本并创造新的价值。

仔细研究发现，所有机器、设备、传感器、以及金融的交易系统所产生的数据都是时序的，而且很多还带有位置信息。这些数据具有明显的特征，1: 数据是时序的，一定带有时间戳；2：数据是结构化的；3: 数据极少有更新或删除操作；4：无需传统数据库的事务处理；5：相对互联网应用，写多读少；6：用户关注的是一段时间的趋势，而不是某一特点时间点的值；7: 数据是有保留期限的；8：数据的查询分析一定是基于时间段和地理区域的；9：除存储查询外，还往往需要各种统计和实时计算操作；10：数据量巨大，一天采集的数据就可以超过100亿条。

看似简单的事情，但由于数据记录条数巨大，导致数据的实时写入成为瓶颈，查询分析极为缓慢，成为新的技术挑战。传统的关系型数据库、NoSQL数据库以及流式计算引擎由于没有充分利用这些数据的特点，性能提升极为有限，只能依靠集群技术，投入更多的计算资源和存储资源来处理，企业运营维护成本急剧上升。

面对这一高速增长的时序数据市场和技术挑战，涛思数据推出了创新性的时序大数据平台TDengine。它不依赖任何第三方软件，也不是优化或包装了一个开源的数据库或流式计算产品，而是在吸取众多传统关系型数据库、NoSQL数据库、流式计算引擎、消息队列等软件的优点之后自主开发的产品，在时序空间大数据处理上，有着自己独到的优势。

TDengine的模块之一是时序数据库。但除此之外，为减少研发的复杂度、系统维护的难度，TDengine还提供缓存、消息队列、订阅、流式计算等功能，为物联网、工业互联网大数据的处理提供全栈的技术方案，是一个高效易用的物联网大数据平台。与Hadoop等典型的大数据平台相比，它具有如下鲜明的特点：

- **10倍以上的性能提升**：定义了创新的数据存储结构，单核每秒能处理至少2万次请求，插入数百万个数据点，读出一千万以上数据点，比现有通用数据库快十倍以上。
- **硬件或云服务成本降至1/5**：由于超强性能，计算资源不到通用大数据方案的1/5；通过列式存储和先进的压缩算法，存储空间不到通用数据库的1/10。
- **全栈时序数据处理引擎**：将数据库、消息队列、缓存、流式计算等功能融为一体，应用无需再集成Kafka/Redis/HBase/Spark/HDFS等软件，大幅降低应用开发和维护的复杂度成本。
- **强大的分析功能**：无论是十年前还是一秒钟前的数据，指定时间范围即可查询。数据可在时间轴上或多个设备上进行聚合。即席查询可通过Shell, Python, R, MATLAB随时进行。
- **与第三方工具无缝连接**：不用一行代码，即可与Telegraf, Grafana, EMQ, HiveMQ, Prometheus, MATLAB, R等集成。后续将支持OPC, Hadoop, Spark等, BI工具也将无缝连接。
- **零运维成本、零学习成本**：安装集群简单快捷，无需分库分表，实时备份。类标准SQL，支持RESTful, 支持Python/Java/C/C++/C#/Go/Node.js, 与MySQL相似，零学习成本。

采用TDengine，可将典型的物联网、车联网、工业互联网大数据平台的总拥有成本大幅降低。但需要指出的是，因充分利用了物联网时序数据的特点，它无法用来处理网络爬虫、微博、微信、电商、ERP、CRM等通用型数据。

本文档提供TDengine的安装和使用说明。阅读本文档，要求读者具备数据库的基本知识，了解基本的SQL语法以及常用Linux命令。

2. TDengine快速上手

TDengine软件分为服务器、应用驱动和报警模块三部分，目前2.0版服务器仅能在Linux系统安装和运行，后续会提供Windows、macOS版本。

应用驱动

如果应用在Mac上运行，目前只能使用RESTful接口连接服务器。如果应用在Windows和Linux上运行，可使用C/C++/C#/JAVA/Python/Go/Node.js接口连接服务器。

CPU

CPU支持X64/ARM64/MIPS64/Alpha64,后续会支持ARM32、RISC-V等CPU架构。用户可根据需求选择通过源码或者安装包来安装。

服务器

目前TDengine服务器可以运行在以下平台上：

	CentOS 6/7/8	**Ubuntu 16/18/20**	**Other Linux**	**统信 UOS**	**银河/中标麒麟**	**凝思 V60/V80**	**华为 EulerOS**
X64	●	●		○	●	●	●
龙芯 MIPS64			●				
鲲鹏 ARM64		○	○		●		
申威 Alpha64			○	●			
飞腾 ARM64		○ 优麒麟					
海光 X64	●	●	●	○	●	●	
瑞芯微 ARM64			○				
全志 ARM64			○				
炬力 ARM64			○				
华为云 ARM64							●

注：● 表示经过官方测试验证，○ 表示非官方测试验证。

2.1. 安装TDengine

2.1.1. TDengine服务器安装

服务器部分，涛思提供tar.gz包供安装。

- TDengine-enterprise-server-2.x.x.x-Linux-x64.tar.gz

安装步骤如下：

1. 从涛思交付团队获得服务器的tar.gz安装包，如TDengine-enterprise-server-2.0.0.0-Linux-x64.tar.gz

2. 解压缩软件包

将软件包放置在当前用户可读写的任意目录下，然后执行下面的命令：

```
tar -xzvf TDengine-enterprise-server-xxxxxxxxx.tar.gz
```

其中xxxxxxx需要替换为实际版本的字符串。

3. 执行安装脚本

解压软件包之后，会在解压目录下看到以下文件(目录)：

connector: 各开发语言连接器
driver: TDengine应用驱动driver
examples: 示例代码
install.sh: 主安装脚本，用于安装服务端及客户端程序
taos.tar.gz: 主安装包

运行install.sh进行安装。

提示：安装过程中，会提示安装的数据节点是否要加入一个已经存在的TDengine集群，如果是，则需要输入该集群的任何节点的FQDN:端口号(默认6030)作为本机的first EP；如果直接回车，则会新创建一个集群。

详细的集群部署配置，参见《TDengine集群安装、管理》章节。

2.1.2. TDengine应用驱动安装

应用驱动目前支持的平台有：Linux x64, Windwos x64, Windows x86。

【注意】Linux服务器安装包中已包含应用驱动，无需再单独安装。

Linux和Windows x64/x86安装包如下：

- TDengine-client-2.x.x.x-Linux-x64.tar.gz
- TDengine-client-2.x.x.x-Windows-x64.exe
- TDengine-client-2.x.x.x-Windows-x86.exe

以Linux 64应用驱动为例(Windows x64/x86安装类似)介绍安装步骤如下：

1. 从涛思交付团队获得应用驱动的**tar.gz**安装包，如**TDengine-client-2.0.0.0-Linux-x64.tar.gz**
2. 解压缩软件包

将软件包放置在当前用户可读写的任意目录下，然后执行下面的命令：

```
tar -xzvf TDengine-client-xxxx.tar.gz
```

其中xxxx需要替换为实际版本的字符串。

3. 执行安装脚本

解压软件包之后，会在解压目录下看到以下文件(目录)：

Install_client.sh：安装脚本，用于应用驱动程序
taos.tar.gz：应用驱动安装包
driver：TDengine应用驱动driver
connector: 编程语言连接器 Python/JDBC
examples: 示例程序 C/Go/JDBC/Matlab/Python/R

运行install_client.sh进行安装。

4. **配置taos.cfg**

编辑taos.cfg文件(默认路径/etc/taos/taos.cfg)，将firstEP修改为TDengine服务器的End Point，例如：h1.taos.com:6030。

提示：如本机没有部署TDengine服务器，仅安装了应用驱动，则taos.cfg中无需配置FQDN。

TDengine应用驱动在Windows系统的默认安装路径为：`C:\TDengine`。

2.1.3. 报警模块安装

报警模块的Linux安装包如下：

- TDengine-alert-2.x.x.x-Linux-x64.tar.gz

从涛思数据官网下载最新的安装包。下载完成后，使用下面的命令解压即可：

```
$ tar -xzf tdengine-alert-2.x.x.x-Linux-x64.tar.gz
```

报警模块的具体使用，请参见官网博文 使用 TDengine 进行报警监测。

2.2. 启动TDengine服务

安装成功后，用户可使用 `systemctl` 命令来启动 TDengine 的服务进程。

```
1 $ systemctl start taosd
```

检查服务是否正常工作：

```
$ systemctl status taosd
```

如果 TDengine 服务正常工作，那么您可以通过 TDengine 的命令行程序 `taos` 来访问并体验 TDengine。

注意：

- systemctl 命令需要 *root* 权限来运行，如果您非 *root* 用户，请在命令前添加 sudo 。
- 为更好的获得产品反馈，改善产品，TDengine 会采集基本的使用信息，但您可以修改系统配置文件 taos.cfg 里的配置参数 telemetryReporting，将其设为 0，就可将其关闭。
- TDengine 采用 FQDN (一般就是 hostname)作为节点的 ID，为保证正常运行，需要给运行 taosd 的服务器配置好 hostname，在客户端应用运行的机器配置好 DNS 服务或 hosts 文件，保证 FQDN 能够解析。
- `systemctl stop taosd` 指令在执行后并不会马上停止 TDengine 服务，而是会等待系统中必要的落盘工作正常完成。在数据量很大的情况下，这可能会消耗较长时间。
- TDengine 支持在使用 `systemd` 做进程服务管理的 Linux 系统上安装，用 `which systemctl` 命令来检测系统中是否存在 `systemd` 包：

  ```
  $ which systemctl
  ```

  如果系统中不支持 `systemd`，也可以用手动运行 /usr/local/taos/bin/taosd 方式启动 TDengine 服务。

2.3. TDengine命令行程序

执行 TDengine 命令行程序，您只要在 Linux 终端执行 `taos` 即可。

```
$ taos
```

如果 TDengine 终端连接服务成功，将会打印出欢迎消息和版本信息。如果失败，则会打印错误消息出来（请参考 FAQ 来解决终端连接服务端失败的问题）。TDengine 终端的提示符号如下：

```
taos>
```

在 TDengine 终端中，用户可以通过 SQL 命令来创建/删除数据库、表等，并进行插入查询操作。在终端中运行的 SQL 语句需要以分号结束来运行。示例：

```
create database demo;
use demo;
create table t (ts timestamp, speed int);
insert into t values ('2019-07-15 00:00:00', 10);
insert into t values ('2019-07-15 01:00:00', 20);
select * from t;
           ts            |    speed    |
========================================
 2019-07-15 00:00:00.000 |          10 |
 2019-07-15 01:00:00.000 |          20 |
Query OK, 2 row(s) in set (0.003128s)
```

除执行 SQL 语句外，系统管理员还可以从 TDengine 终端进行检查系统运行状态、添加删除用户账号等操作。

命令行参数

您可通过配置命令行参数来改变 TDengine 终端的行为。以下为常用的几个命令行参数：

- -c, --config-dir: 指定配置文件目录，默认为 `/etc/taos`
- -h, --host: 指定服务的 FQDN 地址或 IP 地址，默认为连接本地服务
- -s, --commands: 在不进入终端的情况下运行 TDengine 命令
- -u, --user: 连接 TDengine 服务器的用户名，缺省为 root
- -p, --password: 连接TDengine服务器的密码，缺省为 taosdata
- -?, --help: 打印出所有命令行参数

示例：

```
$ taos -h h1.taos.com -s "use db; show tables;"
```

运行 SQL 命令脚本

TDengine 终端可以通过 `source` 命令来运行 SQL 命令脚本。

```
taos> source <filename>;
```

Shell 小技巧

- 可以使用上下光标键查看历史输入的指令
- 修改用户密码：在 shell 中使用 `alter user` 命令，缺省密码为 taosdata
- ctrl+c 中止正在进行中的查询
- 执行 `RESET QUERY CACHE` 可清除本地缓存的表 schema
- 批量执行 SQL 语句。可以将一系列的 shell 命令（以英文 ; 结尾，每个 SQL 语句为一行）按行存放在文件里，在 shell 里执行命令 `source <file-name>` 自动执行该文件里所有的 SQL 语句
- 输入 q 回车，退出 taos shell

3. 数据模型和整体架构

3.1. 数据模型

3.1.1. 物联网典型场景

在典型的物联网、车联网、运维监测场景中，往往有多种不同类型的数据采集设备，采集一个到多个不同的物理量。而同一种采集设备类型，往往又有多个具体的采集设备分布在不同的地点。大数据处理系统就是要将各种采集的数据汇总，然后进行计算和分析。对于同一类设备，其采集的数据都是很规则的。以智能电表为例，假设每个智能电表采集电流、电压、相位三个量，其采集的数据类似如下的表格：

设备ID	时间戳	采集量			标签	
Device ID	**Time Stamp**	**current**	**voltage**	**phase**	**location**	**groupId**
d1001	1538548685000	10.3	219	0.31	Beijing.Chaoyang	2
d1002	1538548684000	10.2	220	0.23	Beijing.Chaoyang	3
d1003	1538548686500	11.5	221	0.35	Beijing.Haidian	3
d1004	1538548685500	13.4	223	0.29	Beijing.Haidian	2
d1001	1538548695000	12.6	218	0.33	Beijing.Chaoyang	2
d1004	1538548696600	11.8	221	0.28	Beijing.Haidian	2
d1002	1538548696650	10.3	218	0.25	Beijing.Chaoyang	3
d1001	1538548696800	12.3	221	0.31	Beijing.Chaoyang	2

表1：智能电表数据示例

每一条记录都有设备ID，时间戳，采集的物理量（如上图中的电流、电压、相位），还有与每个设备相关的静态标签（如上述表1中的位置Location和分组groupId）。每个设备是受外界的触发，或按照设定的周期采集数据。采集的数据点是时序的，是一个数据流。

3.1.2. 数据特征

除时序特征外，仔细研究发现，物联网、车联网、运维监测类数据还具有很多其他明显的特征：

1. 数据高度结构化；
2. 数据极少有更新或删除操作；
3. 无需传统数据库的事务处理；
4. 相对互联网应用，写多读少；
5. 流量平稳，根据设备数量和采集频次，可以预测出来；
6. 用户关注的是一段时间的趋势，而不是某一特定时间点的值；
7. 数据有保留期限；
8. 数据的查询分析一定是基于时间段和空间区域；
9. 除存储、查询操作外，还需要各种统计和实时计算操作；
0. 数据量巨大，一天可能采集的数据就可以超过100亿条。

充分利用上述特征，TDengine 采取了经特殊优化的存储和计算设计来处理时序数据，它将系统处理能力显著提高，同时大幅降低了系统运维的复杂度。

3.1.3. 关系型数据库模型

因为采集的数据一般是结构化数据，同时为降低学习门槛，TDengine采用传统的关系型数据库模型管理数据。因此用户需要先创建库，然后创建表，之后才能插入或查询数据。TDengine采用的是结构化存储，而不是NoSQL的key-value存储。

3.1.4. 一个数据采集点一张表

为充分利用其数据的时序性和其他数据特点，TDengine要求**对每个数据采集点单独建表**（比如有一千万个智能电表，就需创建一千万张表，上述表格中的d1001, d1002, d1003, d1004都需单独建表），用来存储这个采集点所采集的时序数据。这种设计有几大优点：

1. 能保证一个采集点的数据在存储介质上是以块为单位连续存储的。如果读取一个时间段的数据，它能大幅减少随机读取操作，成数量级的提升读取和查询速度。
2. 由于不同采集设备产生数据的过程完全独立，每个设备的数据源是唯一的，一张表也就只有一个写入者，这样就可采用无锁方式来写，写入速度就能大幅提升。
3. 对于一个数据采集点而言，其产生的数据是时序的，因此写的操作可用追加的方式实现，进一步大幅提高数据写入速度。

如果采用传统的方式，将多个设备的数据写入一张表，由于网络延时不可控，不同设备的数据到达服务器的时序是无法保证的，写入操作是要有锁保护的，而且一个设备的数据是难以保证连续存储在一起的。**采用一个数据采集点一张表的方式，能最大程度的保证单个数据采集点的插入和查询的性能是最优的**。

TDengine 建议用数据采集点的名字(如上表中的D1001)来做表名。每个数据采集点可能同时采集多个物理量(如上表中的curent, voltage, phase)，每个物理量对应一张表中的一列，数据类型可以是整型、浮点型、字符串等。除此之外，表的第一列必须是时间戳，即数据类型为 timestamp。对采集的数据，TDengine将自动按照时间戳建立索引，但对采集的物理量不建任何索引。数据用列式存储方式保存。

3.1.5. 超级表：同一类型数据采集点的集合

由于一个数据采集点一张表，导致表的数量巨增，难以管理，而且应用经常需要做采集点之间的聚合操作，聚合的操作也变得复杂起来。为解决这个问题，TDengine引入超级表(Super Table，简称为STable)的概念。

超级表是指某一特定类型的数据采集点的集合。同一类型的数据采集点，其表的结构是完全一样的，但每个表（数据采集点）的静态属性（标签）是不一样的。描述一个超级表（某一特定类型的数据采集点的集合），除需要定义采集量的表结构之外，还需要定义其标签的schema，标签的数据类型可以是整数、浮点数、字符串，标签可以有多个，可以事后增加、删除或修改。如果整个系统有N个不同类型的数据采集点，就需要建立N个超级表。

在TDengine的设计里，**表用来代表一个具体的数据采集点，超级表用来代表一组相同类型的数据采集点集合**。当为某个具体数据采集点创建表时，用户使用超级表的定义做模板，同时指定该具体采集点（表）的标签值。与传统的关系型数据库相比，表（一个数据采集点）是带有静态标签的，而且这些标签可以事后增加、删除、修改。**一张超级表包含有多张表，这些表具有相同的时序数据schema，但带有不同的标签值**。

当对多个具有相同数据类型的数据采集点进行聚合操作时，TDengine会先把满足标签过滤条件的表从超级表中找出来，然后再扫描这些表的时序数据，进行聚合操作，这样需要扫描的数据集会大幅减少，从而显著提高聚合计算的性能。

3.2. 集群与基本逻辑单元

TDengine 的设计是基于单个硬件、软件系统不可靠，基于任何单台计算机都无法提供足够计算能力和存储能力处理海量数据的假设进行设计的。因此 TDengine 从研发的第一天起，就按照分布式高可靠架构进行设计，是支持水平扩展的，这样任何单台或多台服务器发生硬件故障或软件错误都不影响系统的可用性和可靠性。同时，通过节点虚拟化并辅以自动化负载均衡技术，TDengine 能最高效率地利用异构集群中的计算和存储资源降低硬件投资。

3.2.1. 主要逻辑单元

TDengine 分布式架构的逻辑结构图如下：

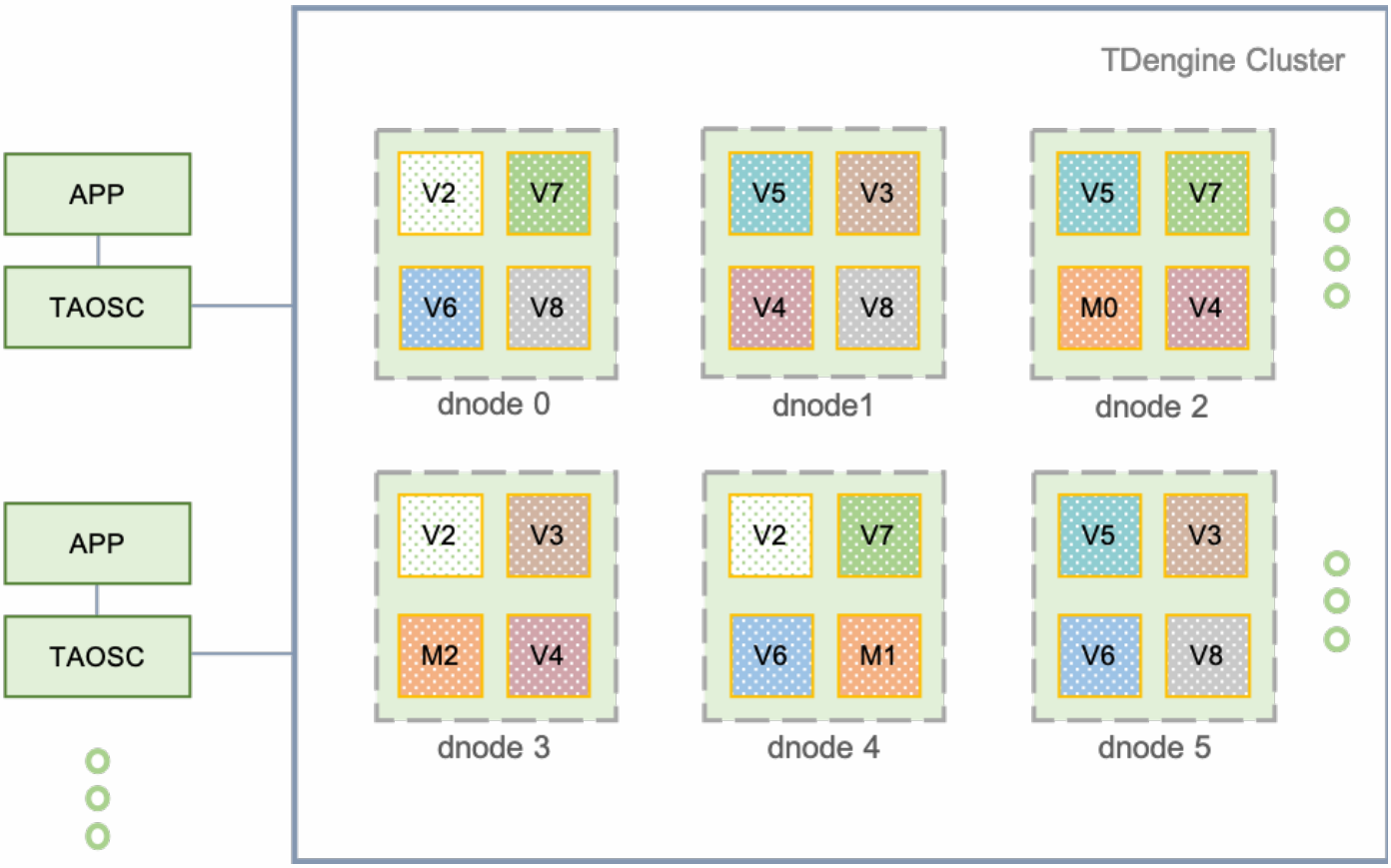


图 1 TDengine架构示意图

一个完整的 TDengine 系统是运行在一到多个物理节点上的，逻辑上，它包含数据节点(dnode)、TDengine应用驱动(taosc)以及应用(app)。系统中存在一到多个数据节点，这些数据节点组成一个集群(cluster)。应用通过taosc的API与TDengine集群进行互动。下面对每个逻辑单元进行简要介绍。

物理节点(pnode)：pnode是一独立运行、拥有自己的计算、存储和网络能力的计算机，可以是安装有OS的物理机、虚拟机或Docker容器。物理节点由其配置的 FQDN(Fully Qualified Domain Name)来标识。TDengine完全依赖FQDN来进行网络通讯，如果不了解FQDN，请看博文《一篇文章说清楚TDengine的FQDN》。

数据节点(dnode)：dnode 是 TDengine 服务器侧执行代码 taosd 在物理节点上的一个运行实例，一个工作的系统必须有至少一个数据节点。dnode包含零到多个逻辑的虚拟节点(vnode)，零或者至多一个逻辑的管理节点(mnode)。dnode在系统中的唯一标识由实例的End Point (EP)决定。EP是dnode所在物理节点的FQDN (Fully Qualified Domain Name)和系统所配置的网络端口号(Port)的组合。通过配置不同的端口，一个物理节点(一台物理机、虚拟机或容器）可以运行多个实例，或有多个数据节点。

虚拟节点(vnode)：为更好的支持数据分片、负载均衡，防止数据过热或倾斜，数据节点被虚拟化成多个虚拟节点(vnode，图中V2, V3, V4等)。每个 vnode 都是一个相对独立的工作单元，是时序数据存储的基本单元，具有独立的运行线程、内存空间与持久化存储的路径。一个 vnode 包含一定数量的表（数据采集点）。当创建一张新表时，系统会检查是否需要创建新的 vnode。一个数据节点上能创建的 vnode 的数量取决于该数据节点所在物理节点的硬件资源。一个 vnode 只属于一个DB，但一个DB可以有多个 vnode。一个 vnode 除存储的时序数据外，也保存有所包含的表的schema、标签值等。一个虚拟节点由所属的数据节点的EP，以及所属的VGroup ID在系统内唯一标识，由管理节点创建并管理。

管理节点(mnode)：一个虚拟的逻辑单元，负责所有数据节点运行状态的监控和维护，以及节点之间的负载均衡(图中M)。同时，管理节点也负责元数据(包括用户、数据库、表、静态标签等)的存储和管理，因此也称为 Meta Node。TDengine 集群中可配置多个(开源版最多不超过3个) mnode，它们自动构建成为一个虚拟管理节点组(图中M0, M1, M2)。mnode 间采用 master/slave 的机制进行管理，而且采取强一致方式进行数据同步，任何数据更新操作只能在 Master 上进行。mnode 集群的创建由系统自动完成，无需人工干预。每个dnode上至多有一个mnode，由所属的数据节点的EP来唯一标识。每个dnode通过内部消息交互自动获取整个集群中所有 mnode 所在的 dnode

的EP。

虚拟节点组(VGroup)：不同数据节点上的 vnode 可以组成一个虚拟节点组(vnode group)来保证系统的高可靠。虚拟节点组内采取master/slave的方式进行管理。写操作只能在 master vnode 上进行，系统采用异步复制的方式将数据同步到 slave vnode，这样确保了一份数据在多个物理节点上有拷贝。一个 vgroup 里虚拟节点个数就是数据的副本数。如果一个DB的副本数为N，系统必须有至少N个数据节点。副本数在创建DB时通过参数 replica 可以指定，缺省为1。使用 TDengine 的多副本特性，可以不再需要昂贵的磁盘阵列等存储设备，就可以获得同样的数据高可靠性。虚拟节点组由管理节点创建、管理，并且由管理节点分配一个系统唯一的ID，VGroup ID。如果两个虚拟节点的vnode group ID相同，说明他们属于同一个组，数据互为备份。虚拟节点组里虚拟节点的个数是可以动态改变的，容许只有一个，也就是没有数据复制。VGroup ID是永远不变的，即使一个虚拟节点组被删除，它的ID也不会被收回重复利用。

TAOSC：taosc是TDengine给应用提供的驱动程序(driver)，负责处理应用与集群的接口交互，提供C/C++语言原生接口，内嵌于JDBC、C#、Python、Go、Node.js语言连接库里。应用都是通过taosc而不是直接连接集群中的数据节点与整个集群进行交互的。这个模块负责获取并缓存元数据；将插入、查询等请求转发到正确的数据节点；在把结果返回给应用时，还需要负责最后一级的聚合、排序、过滤等操作。对于JDBC、C/C++、C#、Python、Go、Node.js接口而言，这个模块是在应用所处的物理节点上运行。同时，为支持全分布式的RESTful接口，taosc在TDengine集群的每个dnode上都有一运行实例。

3.2.2. 节点之间的通讯

通讯方式：TDengine系统的各个数据节点之间，以及应用驱动与各数据节点之间的通讯是通过TCP/UDP进行的。因为考虑到物联网场景，数据写入的包一般不大，因此TDengine 除采用TCP做传输之外，还采用UDP方式，因为 UDP 更加高效，而且不受连接数的限制。TDengine实现了自己的超时、重传、确认等机制，以确保UDP的可靠传输。对于数据量不到15K的数据包，采取UDP的方式进行传输，超过15K的，或者是查询类的操作，自动采取TCP的方式进行传输。同时，TDengine根据配置和数据包，会自动对数据进行压缩/解压缩，数字签名/认证等处理。对于数据节点之间的数据复制，只采用TCP方式进行数据传输。

FQDN配置：一个数据节点有一个或多个FQDN，可以在系统配置文件taos.cfg通过参数"fqdn"进行指定，如果没有指定，系统将自动获取计算机的hostname作为其FQDN。如果节点没有配置FQDN，可以直接将该节点的配置参数fqdn设置为它的IP地址。但不建议使用IP，因为IP地址可变，一旦变化，将让集群无法正常工作。一个数据节点的EP(End Point)由FQDN + Port组成。采用FQDN，需要保证DNS服务正常工作，或者在节点以及应用所在的节点配置好hosts文件。另外，这个参数值的长度需要控制在 96 个字符以内。

端口配置：一个数据节点对外的端口由TDengine的系统配置参数serverPort决定，对集群内部通讯的端口是serverPort+5。为支持多线程高效的处理UDP数据，每个对内和对外的UDP连接，都需要占用5个连续的端口。

- 集群内数据节点之间的数据复制操作占用一个TCP端口，是serverPort+10。
- 集群数据节点对外提供RESTful服务占用一个TCP端口，是serverPort+11。
- 集群内数据节点与Arbitrator节点之间通讯占用一个TCP端口，是serverPort+12。

因此一个数据节点总的端口范围为serverPort到serverPort+12，总共13个TCP/UDP端口。使用时，需要确保防火墙将这些端口打开。每个数据节点可以配置不同的serverPort。（详细的端口情况请参见 TDengine 2.0 端口说明）

集群对外连接：TDengine集群可以容纳单个、多个甚至几千个数据节点。应用只需要向集群中任何一个数据节点发起连接即可，连接需要提供的网络参数是一数据节点的End Point(FQDN加配置的端口号）。通过命令行CLI启动应用taos时，可以通过选项-h来指定数据节点的FQDN, -P来指定其配置的端口号，如果端口不配置，将采用TDengine的系统配置参数serverPort。

集群内部通讯：各个数据节点之间通过TCP/UDP进行连接。一个数据节点启动时，将获取mnode所在的dnode的EP信息，然后与系统中的mnode建立起连接，交换信息。获取mnode的EP信息有三步：

1. 检查mnodeEpSet.json文件是否存在，如果不存在或不能正常打开获得mnode EP信息，进入第二步；
2. 检查系统配置文件taos.cfg，获取节点配置参数firstEp、secondEp（这两个参数指定的节点可以是不带mnode的普通节点，这样的话，节点被连接时会尝试重定向到mnode节点），如果不存在或者taos.cfg里没有这两个配置参数，或无效，进入第三步；
3. 将自己的EP设为mnode EP，并独立运行起来。

获取mnode EP列表后，数据节点发起连接，如果连接成功，则成功加入进工作的集群，如果不成功，则尝试mnode EP列表中的下一个。如果都尝试了，但连接都仍然失败，则休眠几秒后，再进行尝试。

MNODE的选择：TDengine逻辑上有管理节点，但没有单独的执行代码，服务器侧只有一套执行代码taosd。那么哪个数据节点会是管理节点呢？这是系统自动决定的，无需任何人工干预。原则如下：一个数据节点启动时，会检查自己的End Point, 并与获取的mnode EP List进行比对，如果在其中，该数据节点认为自己应该启动mnode模块，成为mnode。如果自己的EP不在mnode EP List里，则不启动mnode模块。在系统的运行过程中，由于负载均衡、宕机等原因，mnode有可能迁移至新的dnode，但一切都是透明的，无需人工干预，配置参数的修改，是mnode自己根据资源做出的决定。

新数据节点的加入：系统有了一个数据节点后，就已经成为一个工作的系统。添加新的节点进集群时，有两个步骤，第一步：使用TDengine CLI连接到现有工作的数据节点，然后用命令"create dnode"将新的数据节点的End Point添加进去; 第二步：在新的数据节点的系统配置参数文件taos.cfg里，将firstEp, secondEp参数设置为现有集群中任意两个数据节点的EP即可。具体添加的详细步骤请见详细的用户手册。这样就把集群一步一步的建立起来。

重定向：无论是dnode还是taosc，最先都是要发起与mnode的连接，但mnode是系统自动创建并维护的，因此对于用户来说，并不知道哪个dnode在运行mnode。TDengine只要求向系统中任何一个工作的dnode发起连接即可。因为任何一个正在运行的dnode，都维护有目前运行的mnode EP List。当收到一个来自新启动的dnode或taosc的连接请求，如果自己不是mnode，则将mnode EP List回复给对方，taosc或新启动的dnode收到这个list, 就重新尝试建立连接。当mnode EP List发生改变，通过节点之间的消息交互，各个数据节点就很快获取最新列表，并通知taosc。

3.2.3. 一个典型的消息流程

为解释vnode、mnode、taosc和应用之间的关系以及各自扮演的角色，下面对写入数据这个典型操作的流程进行剖析。

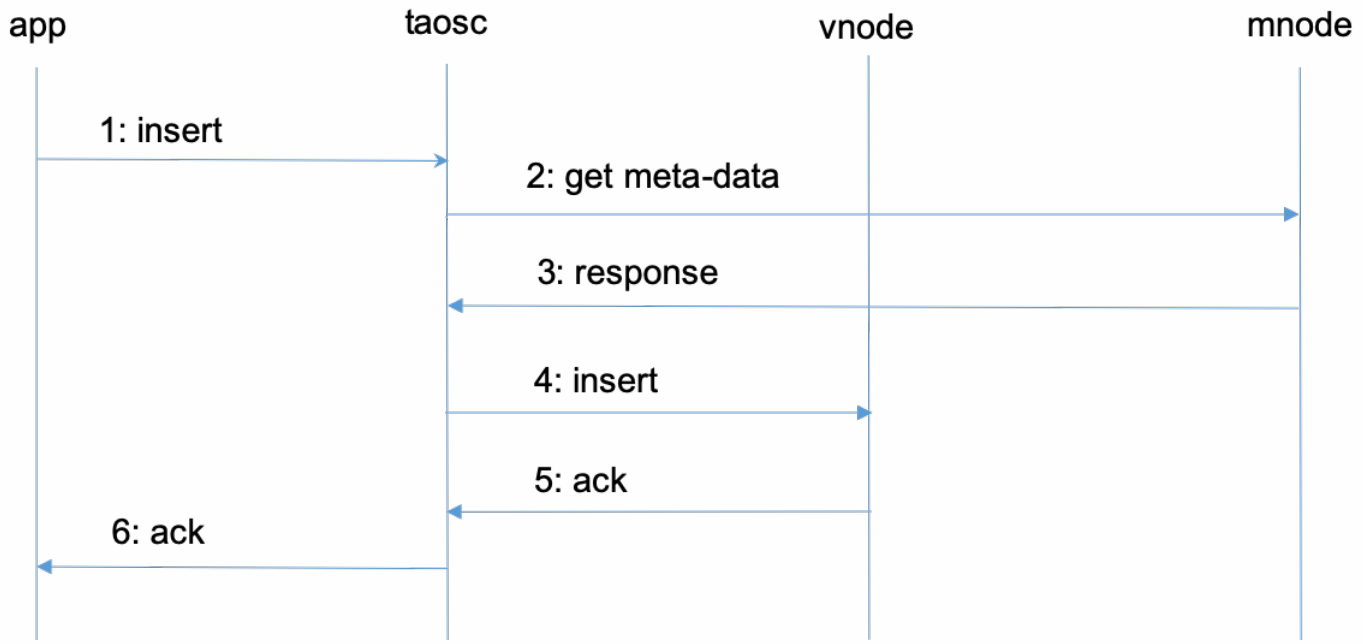


图 2 TDengine典型的操作流程

1. 应用通过JDBC、ODBC或其他API接口发起插入数据的请求。
2. taosc会检查缓存，看是否保存有该表的meta data。如果有，直接到第4步。如果没有，taosc将向mnode发出get meta-data请求。
3. mnode将该表的meta-data返回给taosc。Meta-data包含有该表的schema, 而且还有该表所属的vgroup信息（vnode ID以及所在的dnode的End Point，如果副本数为N，就有N组End Point)。如果taosc迟迟得不到mnode回应，而且存在多个mnode, taosc将向下一个mnode发出请求。
4. taosc向master vnode发起插入请求。
5. vnode插入数据后，给taosc一个应答，表示插入成功。如果taosc迟迟得不到vnode的回应，taosc会认为该节点已经离线。这种情况下，如果被插入的数据库有多个副本，taosc将向vgroup里下一个vnode发出插入请求。
6. taosc通知APP，写入成功。

对于第二和第三步，taosc启动时，并不知道mnode的End Point，因此会直接向配置的集群对外服务的End Point发起请求。如果接收到该请求的dnode并没有配置mnode，该dnode会在回复的消息中告知mnode EP列表，这样taosc会重新向新的mnode的EP发出获取meta-data的请求。

对于第四和第五步，没有缓存的情况下，taosc无法知道虚拟节点组里谁是master，就假设第一个vnodeID就是master,向它发出请求。如果接收到请求的vnode并不是master,它会在回复中告知谁是master，这样taosc就向建议的master vnode发出请求。一旦得到插入成功的回复，taosc会缓存master节点的信息。

上述是插入数据的流程，查询、计算的流程也完全一致。taosc把这些复杂的流程全部封装屏蔽了，对于应用来说无感知也无需任何特别处理。

通过taosc缓存机制，只有在第一次对一张表操作时，才需要访问mnode,因此mnode不会成为系统瓶颈。但因为schema有可能变化，而且vgroup有可能发生改变（比如负载均衡发生），因此taosc会定时和mnode交互，自动更新缓存。

3.3. 存储模型与数据分区、分片

3.3.1. 存储模型

TDengine存储的数据包括采集的时序数据以及库、表相关的元数据、标签数据等，这些数据具体分为三部分：

- 时序数据：存放于vnode里，由data、head和last三个文件组成，数据量大，查询量取决于应用场景。容许乱序写入，但暂时不支持删除操作，并且仅在update参数设置为1时允许更新操作。通过采用一个采集点一张表的模型，一个时间段的数据是连续存储，对单张表的写入是简单的追加操作，一次读，可以读到多条记录，这样保证对单个采集点的插入和查询操作，性能达到最优。
- 标签数据：存放于vnode里的meta文件，支持增删改查四个标准操作。数据量不大，有N张表，就有N条记录，因此可以全内存存储。如果标签过滤操作很多，查询将十分频繁，因此TDengine支持多核多线程并发查询。只要计算资源足够，即使有数千万张表，过滤结果能毫秒级返回。
- 元数据：存放于mnode里，包含系统节点、用户、DB、Table Schema等信息，支持增删改查四个标准操作。这部分数据的量不大，可以全内存保存，而且由于客户端有缓存，查询量也不大。因此目前的设计虽是集中式存储管理，但不会构成性能瓶颈。

与典型的NoSQL存储模型相比，TDengine将标签数据与时序数据完全分离存储，它具有两大优势：

- 能够极大地降低标签数据存储的冗余度：一般的NoSQL数据库或时序数据库，采用的K-V存储，其中的Key包含时间戳、设备ID、各种标签。每条记录都带有这些重复的内容，浪费存储空间。而且如果应用要在历史数据上增加、修改或删除标签，需要遍历数据，重写一遍，操作成本极其昂贵。
- 能够实现极为高效的多表之间的聚合查询：做多表之间聚合查询时，先把符合标签过滤条件的表查找出来，然后再查找这些表相应的数据块，这样大幅减少要扫描的数据集，从而大幅提高查询效率。而且标签数据采用全内存的结构进行管理和维护，千万级别规模的标签数据查询可以在毫秒级别返回。

3.3.2. 数据分片

对于海量的数据管理，为实现水平扩展，一般都需要采取分片(Sharding)分区(Partitioning)策略。TDengine是通过vnode来实现数据分片的，通过一个时间段一个数据文件来实现时序数据分区的。

vnode(虚拟数据节点)负责为采集的时序数据提供写入、查询和计算功能。为便于负载均衡、数据恢复、支持异构环境，TDengine将一个数据节点根据其计算和存储资源切分为多个vnode。这些vnode的管理是TDengine自动完成的，对应用完全透明。

对于单独一个数据采集点，无论其数据量多大，一个vnode（或vnode group, 如果副本数大于1）有足够的计算资源和存储资源来处理（如果每秒生成一条16字节的记录，一年产生的原始数据不到0.5G），因此TDengine将一张表（一个数据采集点）的所有数据都存放在一个vnode里，而不会让同一个采集点的数据分布到两个或多个dnode上。而且一个vnode可存储多个数据采集点(表）的数据，一个vnode可容纳的表的数目的上限为一百万。设计上，一个vnode里所有的表都属于同一个DB。一个数据节点上，除非特殊配置，一个DB拥有的vnode数目不会超过系统核的数目。

创建DB时，系统并不会马上分配资源。但当创建一张表时，系统将看是否有已经分配的vnode, 且该vnode是否有空余的表空间，如果有，立即在该有空位的vnode创建表。如果没有，系统将从集群中，根据当前的负载情况，在一个dnode上创建一新的vnode, 然后创建表。如果DB有多个副本，系统不是只创建一个vnode，而是一个vgroup(虚拟数据节点组)。系统对vnode的数目没有任何限制，仅仅受限于物理节点本身的计算和存储资源。

每张表的meda data（包含schema, 标签等）也存放于vnode里，而不是集中存放于mnode，实际上这是对Meta数据的分片，这样便于高效并行的进行标签过滤操作。

3.3.3. 数据分区

TDengine除vnode分片之外，还对时序数据按照时间段进行分区。每个数据文件只包含一个时间段的时序数据，时间段的长度由DB的配置参数days决定。这种按时间段分区的方法还便于高效实现数据的保留策略，只要数据文件超过规定的天数（系统配置参数keep)，将被自动删除。而且不同的时间段可以存放于不同的路径和存储介质，以便于大数据的冷热管理，实现多级存储。

总的来说，**TDengine是通过vnode以及时间两个维度，对大数据进行切分**，便于并行高效的管理，实现水平扩展。

3.3.4. 负载均衡

每个dnode都定时向 mnode(虚拟管理节点)报告其状态（包括硬盘空间、内存大小、CPU、网络、虚拟节点个数等），因此mnode了解整个集群的状态。基于整体状态，当mnode发现某个dnode负载过重，它会将dnode上的一个或多个vnode挪到其他dnode。在挪动过程中，对外服务继续进行，数据插入、查询和计算操作都不受影响。

如果mnode一段时间没有收到dnode的状态报告，mnode会认为这个dnode已经离线。如果离线时间超过一定时长（时长由配置参数offlineThreshold决定），该dnode将被mnode强制剔除出集群。该dnode上的vnodes如果副本数大于1，系统将自动在其他dnode上创建新的副本，以保证数据的副本数。如果该dnode上还有mnode, 而且mnode的副本数大于1，系统也将自动在其他dnode上创建新的mnode, 以保证mnode的副本数。

当新的数据节点被添加进集群，因为新的计算和存储被添加进来，系统也将自动启动负载均衡流程。

负载均衡过程无需任何人工干预，应用也无需重启，将自动连接新的节点，完全透明。

提示：负载均衡由参数balance控制，决定开启/关闭自动负载均衡。

3.4. 数据写入与复制流程

如果一个数据库有N个副本，那一个虚拟节点组就有N个虚拟节点，但是只有一个是master，其他都是slave。当应用将新的记录写入系统时，只有master vnode能接受写的请求。如果slave vnode收到写的请求，系统将通知taosc需要重新定向。

3.4.1. Master Vnode写入流程

Master Vnode遵循下面的写入流程：

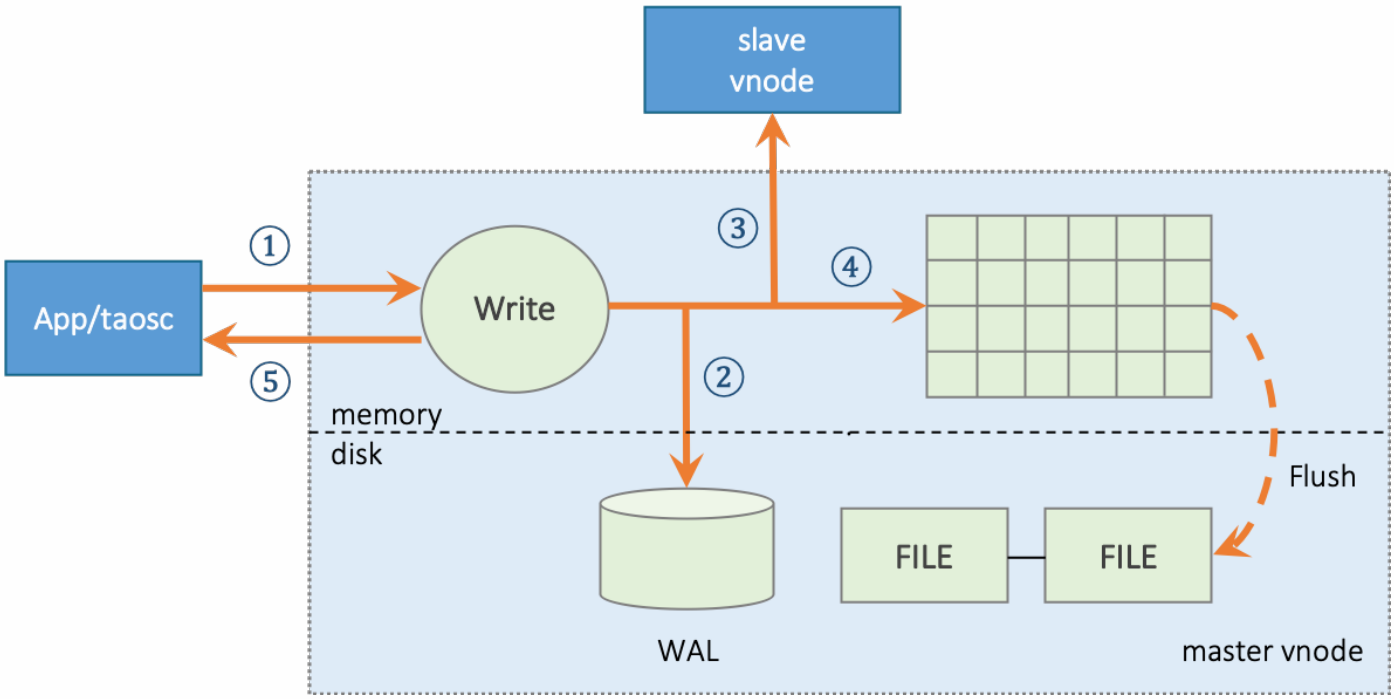


图 3 TDengine Master写入流程

1. master vnode收到应用的数据插入请求，验证OK，进入下一步；
2. 如果系统配置参数walLevel大于0，vnode将把该请求的原始数据包写入数据库日志文件WAL。如果walLevel设置为2，而且fsync设置为0，TDengine还将WAL数据立即落盘，以保证即使宕机，也能从数据库日志文件中恢复数据，避免数据的丢失；
3. 如果有多个副本，vnode将把数据包转发给同一虚拟节点组内的slave vnodes, 该转发包带有数据的版本号(version)；
4. 写入内存，并将记录加入到skip list；
5. master vnode返回确认信息给应用，表示写入成功。
6. 如果第2，3，4步中任何一步失败，将直接返回错误给应用。

3.4.2. Slave Vnode写入流程

对于slave vnode，写入流程是：

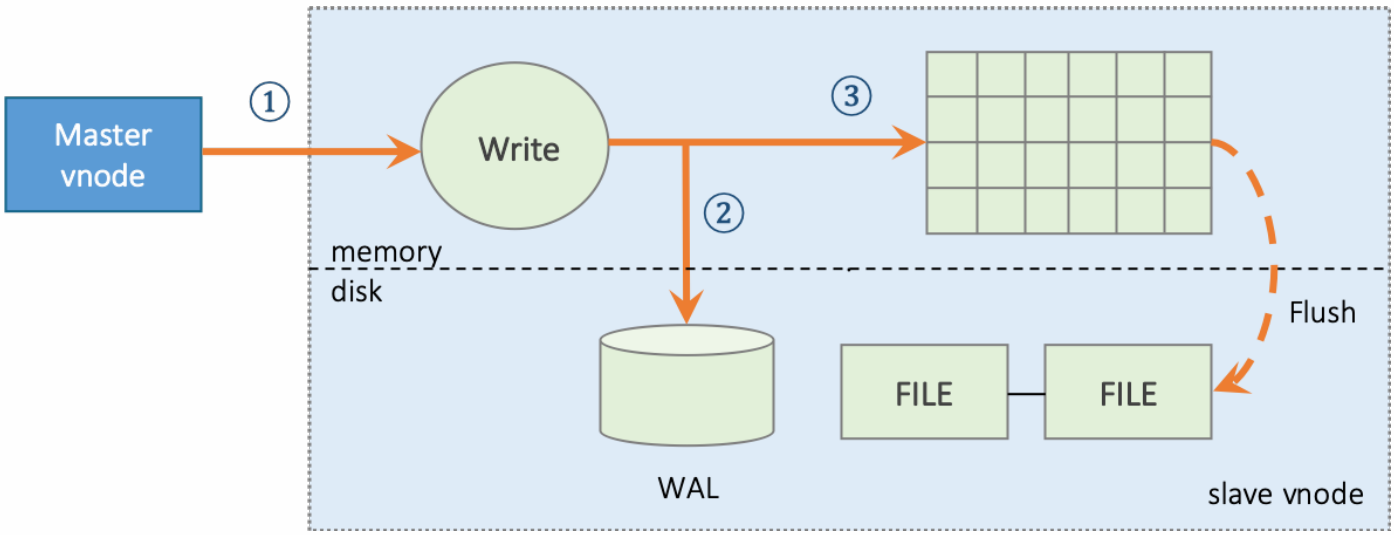


图 4 TDengine Slave写入流程

1. slave vnode收到Master vnode转发了的数据插入请求。检查last version是否与master一致，如果一致，进入下一步。如果不一致，需要进入同步状态。
2. 如果系统配置参数walLevel大于0，vnode将把该请求的原始数据包写入数据库日志文件WAL。如果walLevel设置为2，而且fsync设置为0，TDengine还将WAL数据立即落盘，以保证即使宕机，也能从数据库日志文件中恢复数据，避免数据的丢失。
3. 写入内存，更新内存中的skip list。

与master vnode相比，slave vnode不存在转发环节，也不存在回复确认环节，少了两步。但写内存与WAL是完全一样的。

3.4.3. 异地容灾、IDC迁移

从上述master和slave流程可以看出，TDengine采用的是异步复制的方式进行数据同步。这种方式能够大幅提高写入性能，网络延时对写入速度不会有大的影响。通过配置每个物理节点的IDC和机架号，可以保证对于一个虚拟节点组，虚拟节点由来自不同IDC、不同机架的物理节点组成，从而实现异地容灾。因此TDengine原生支持异地容灾，无需再使用其他工具。

另一方面，TDengine支持动态修改副本数，一旦副本数增加，新加入的虚拟节点将立即进入数据同步流程，同步结束后，新加入的虚拟节点即可提供服务。而在同步过程中，master以及其他已经同步的虚拟节点都可以对外提供服务。利用这一特性，TDengine可以实现无服务中断的IDC机房迁移。只需要将新IDC的物理节点加入现有集群，等数据同步完成后，再将老的IDC的物理节点从集群中剔除即可。

但是，这种异步复制的方式，存在极小的时间窗口，丢失写入的数据。具体场景如下：

1. master vnode完成了它的5步操作，已经给APP确认写入成功，然后宕机
2. slave vnode收到写入请求后，在第2步写入日志之前，处理失败
3. slave vnode将成为新的master，从而丢失了一条记录

理论上，只要是异步复制，就无法保证100%不丢失。但是这个窗口极小，master与slave要同时发生故障，而且发生在刚给应用确认写入成功之后。

3.4.4. 主从选择

Vnode会保持一个数据版本号(version)，对内存数据进行持久化存储时，对该版本号也进行持久化存储。每个数据更新操作，无论是采集的时序数据还是元数据，这个版本号将增加1。

一个vnode启动时，角色(master、slave) 是不定的，数据是处于未同步状态，它需要与虚拟节点组内其他节点建立TCP连接，并互相交换status，其中包括version和自己的角色。通过status的交换，系统进入选主流程，规则如下：

1. 如果只有一个副本，该副本永远就是master
2. 所有副本都在线时，版本最高的被选为master
3. 在线的虚拟节点数过半，而且有虚拟节点是slave的话，该虚拟节点自动成为master
4. 对于2和3，如果多个虚拟节点满足成为master的要求，那么虚拟节点组的节点列表里，最前面的选为master

更多的关于数据复制的流程，请见TDengine 2.0数据复制模块设计。

3.4.5. 同步复制

对于数据一致性要求更高的场景，异步数据复制无法满足要求，因为有极小的概率丢失数据，因此TDengine提供同步复制的机制供用户选择。在创建数据库时，除指定副本数replica之外，用户还需要指定新的参数quorum。如果quorum大于1，它表示每次master转发给副本时，需要等待quorum-1个回复确认，才能通知应用，数据在slave已经写入成功。如果在一定的时间内，得不到quorum-1个回复确认，master vnode将返回错误给应用。

采用同步复制，系统的性能会有所下降，而且latency会增加。因为元数据要强一致，mnode之间的数据同步缺省就是采用的同步复制。

3.5. 缓存与持久化

3.5.1. 缓存

TDengine采用时间驱动缓存管理策略（First-In-First-Out，FIFO），又称为写驱动的缓存管理机制。这种策略有别于读驱动的数据缓存模式（Least-Recent-Used，LRU），直接将最近写入的数据保存在系统的缓存中。当缓存达到临界值的时候，将最早的数据批量写入磁盘。一般意义上来说，对于物联网数据的使用，用户最为关心的是刚产生的数据，即当前状态。TDengine充分利用这一特性，将最近到达的（当前状态）数据保存在缓存中。

TDengine通过查询函数向用户提供毫秒级的数据获取能力。直接将最近到达的数据保存在缓存中，可以更加快速地响应用户针对最近一条或一批数据的查询分析，整体上提供更快的数据库查询响应能力。从这个意义上来说，**可通过设置合适的配置参数将TDengine作为数据缓存来使用，而不需要再部署Redis或其他额外的缓存系统**，可有效地简化系统架构，降低运维的成本。需要注意的是，TDengine重启以后系统的缓存将被清空，之前缓存的数据均会被批量写入磁盘，缓存的数据将不会像专门的key-value缓存系统再将之前缓存的数据重新加载到缓存中。

每个vnode有自己独立的内存，而且由多个固定大小的内存块组成，不同vnode之间完全隔离。数据写入时，类似于日志的写法，数据被顺序追加写入内存，但每个vnode维护有自己的skip list，便于迅速查找。当三分之一以上的内存块写满时，启动落盘操作，而且后续写的操作在新的内存块进行。这样，一个vnode里有三分之一内存块是保留有最近的数据的，以达到缓存、快速查找的目的。一个vnode的内存块的个数由配置参数blocks决定，内存块的大小由配置参数cache决定。

3.5.2. 持久化存储

TDengine采用数据驱动的方式让缓存中的数据写入硬盘进行持久化存储。当vnode中缓存的数据达到一定规模时，为了不阻塞后续数据的写入，TDengine也会拉起落盘线程将缓存的数据写入持久化存储。TDengine在数据落盘时会打开新的数据库日志文件，在落盘成功后则会删除老的数据库日志文件，避免日志文件无限制地增长。

为充分利用时序数据特点，TDengine将一个vnode保存在持久化存储的数据切分成多个文件，每个文件只保存固定天数的数据，这个天数由系统配置参数days决定。切分成多个文件后，给定查询的起止日期，无需任何索引，就可以立即定位需要打开哪些数据文件，大大加快读取速度。

对于采集的数据，一般有保留时长，这个时长由系统配置参数keep决定。超过这个设置天数的数据文件，将被系统自动删除，释放存储空间。

给定days与keep两个参数，一个典型工作状态的vnode中总的数据文件数为：向上取整(keep/days)+1个。总的数据文件个数不宜过大，也不宜过小。10到100以内合适。基于这个原则，可以设置合理的days。目前的版本，参数keep可以修改，但对于参数days，一旦设置后，不可修改。

在每个数据文件里，一张表的数据是一块一块存储的。一张表可以有一到多个数据文件块。在一个文件块里，数据是列式存储的，占用的是一片连续的存储空间，这样大大提高读取速度。文件块的大小由系统参数maxRows（每块最大记录条数）决定，缺省值为4096。这个值不宜过大，也不宜过小。过大，定位具体时间段的数据的搜索时间会变长，影响读取速度；过小，数据块的索引太大，压缩效率偏低，也影响读取速度。

每个数据文件(.data结尾)都有一个对应的索引文件（.head结尾），该索引文件对每张表都有一数据块的摘要信息，记录了每个数据块在数据文件中的偏移量，数据的起止时间等信息，以帮助系统迅速定位需要查找的数据。每个数据文件还有一对应的last文件(.last结尾)，该文件是为防止落盘时数据块碎片化而设计的。如果一张表落盘的记录条数没有达到系统配置参数minRows（每块最小记录条数），将被先存储到last文件，等下次落盘时，新落盘的记录将与last文件的记录进行合并，再写入数据文件。

数据写入磁盘时，根据系统配置参数comp决定是否压缩数据。TDengine提供了三种压缩选项：无压缩、一阶段压缩和两阶段压缩，分别对应comp值为0、1和2的情况。一阶段压缩根据数据的类型进行了相应的压缩，压缩算法包括delta-delta编码、simple 8B方法、zig-zag编码、LZ4等算法。二阶段压缩在一阶段压缩的基础上又用通用压缩算法进行了压缩，压缩率更高。

3.5.3. 多级存储

说明：多级存储功能仅企业版支持，从 2.0.16.0 版本开始提供。

在默认配置下，TDengine会将所有数据保存在/var/lib/taos目录下，而且每个vnode的数据文件保存在该目录下的不同目录。为扩大存储空间，尽量减少文件读取的瓶颈，提高数据吞吐率 TDengine可通过配置系统参数dataDir让多个挂载的硬盘被系统同时使用。

除此之外，TDengine也提供了数据分级存储的功能，将不同时间段的数据存储在挂载的不同介质上的目录里，从而实现不同“热度”的数据存储在不同的存储介质上，充分利用存储，节约成本。比如，最新采集的数据需要经常访问，对硬盘的读取性能要求高，那么用户可以配置将这些数据存储在SSD盘上。超过一定期限的数据，查询需求量没有那么高，那么可以存储在相对便宜的HDD盘上。

多级存储支持3级，每级最多可配置16个挂载点。

TDengine多级存储配置方式如下（在配置文件/etc/taos/taos.cfg中）：

```
dataDir [path] <level> <primary>
```

- path: 挂载点的文件夹路径
- level: 介质存储等级，取值为0，1，2。
  0级存储最新的数据，1级存储次新的数据，2级存储最老的数据，省略默认为0。
  各级存储之间的数据流向：0级存储 -> 1级存储 -> 2级存储。
  同一存储等级可挂载多个硬盘，同一存储等级上的数据文件分布在该存储等级的所有硬盘上。
  需要说明的是，数据在不同级别的存储介质上的移动，是由系统自动完成的，用户无需干预。
- primary: 是否为主挂载点，0（是）或1（否），省略默认为1。

在配置中，只允许一个主挂载点的存在（level=0, primary=0），例如采用如下的配置方式：

```
dataDir /mnt/data1 0 0
dataDir /mnt/data2 0 1
dataDir /mnt/data3 1 1
dataDir /mnt/data4 1 1
dataDir /mnt/data5 2 1
dataDir /mnt/data6 2 1
```

注意：

1. 多级存储不允许跨级配置，合法的配置方案有：仅0级，仅0级+1级，以及0级+1级+2级。而不允许只配置level=0和level=2，而不配置level=1。
2. 禁止手动移除使用中的挂载盘，挂载盘目前不支持非本地的网络盘。
3. 多级存储目前不支持删除已经挂载的硬盘的功能。

3.6. 数据查询

TDengine提供了多种多样针对表和超级表的查询处理功能，除了常规的聚合查询之外，还提供针对时序数据的窗口查询、统计聚合等功能。TDengine的查询处理需要客户端、vnode、mnode节点协同完成。

3.6.1. 单表查询

SQL语句的解析和校验工作在客户端完成。解析SQL语句并生成抽象语法树(Abstract Syntax Tree, AST)，然后对其进行校验和检查。以及向管理节点(mnode)请求查询中指定表的元数据信息(table metadata)。

根据元数据信息中的End Point信息，将查询请求序列化后发送到该表所在的数据节点（dnode）。dnode接收到查询请求后，识别出该查询请求指向的虚拟节点（vnode），将消息转发到vnode的查询执行队列。vnode的查询执行线程建立基础的查询执行环境，并立即返回该查询请求，同时开始执行该查询。

客户端在获取查询结果的时候，dnode的查询执行队列中的工作线程会等待vnode执行线程执行完成，才能将查询结果返回到请求的客户端。

3.6.2. 按时间轴聚合、降采样、插值

时序数据有别于普通数据的显著特征是每条记录均具有时间戳，因此针对具有时间戳的数据在时间轴上进行聚合是不同于普通数据库的重要功能。从这点上来看，与流计算引擎的窗口查询有相似的地方。

在TDengine中引入关键词interval来进行时间轴上固定长度时间窗口的切分，并按照时间窗口对数据进行聚合，对窗口范围内的数据按需进行聚合。例如：

```
SELECT COUNT(*) FROM d1001 INTERVAL(1h);
```

针对d1001设备采集的数据，按照1小时的时间窗口返回每小时存储的记录数量。

在需要连续获得查询结果的应用场景下，如果给定的时间区间存在数据缺失，会导致该区间数据结果也丢失。TDengine提供策略针对时间轴聚合计算的结果进行插值，通过使用关键词fill就能够对时间轴聚合结果进行插值。例如：

```
SELECT COUNT(*) FROM d1001 WHERE ts >= '2017-7-14 00:00:00' AND ts < '2017-7-14 23:59:59' INTERVAL(1h) FILL(PREV);
```

针对d1001设备采集数据统计每小时记录数，如果某一个小时不存在数据，则返回之前一个小时的统计数据。TDengine提供前向插值(prev)、线性插值(linear)、NULL值填充(NULL)、特定值填充(value)。

3.6.3. 多表聚合查询

TDengine对每个数据采集点单独建表，但在实际应用中经常需要对不同的采集点数据进行聚合。为高效的进行聚合操作，TDengine引入超级表（STable）的概念。超级表用来代表一特定类型的数据采集点，它是包含多张表的表集合，集合里每张表的模式（schema）完全一致，但每张表都带有自己的静态标签，标签可以有多个，可以随时增加、删除和修改。应用可通过指定标签的过滤条件，对一个STable下的全部或部分表进行聚合或统计操作，这样大大简化应用的开发。其具体流程如下图所示：

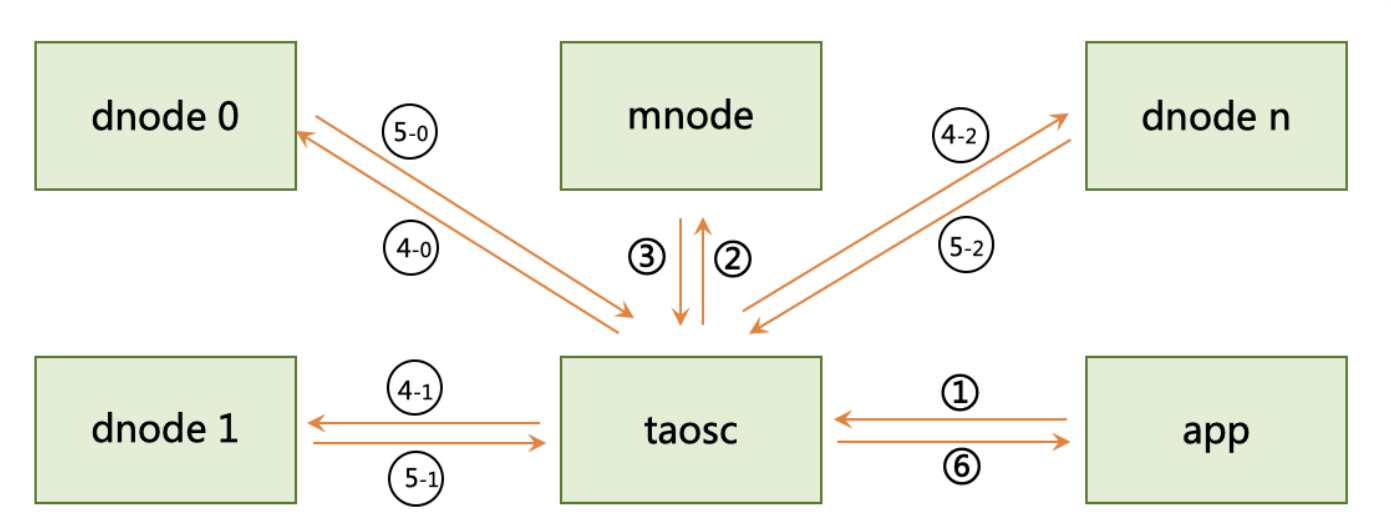


图 5 多表聚合查询原理图

1. 应用将一个查询条件发往系统；
2. taosc将超级表的名字发往 meta node（管理节点）；
3. 管理节点将超级表所拥有的 vnode 列表发回 taosc；
4. taosc将计算的请求连同标签过滤条件发往这些vnode对应的多个数据节点；
5. 每个vnode先在内存里查找出自己节点里符合标签过滤条件的表的集合，然后扫描存储的时序数据，完成相应的聚合计算，将结果返回给taosc；
6. taosc将多个数据节点返回的结果做最后的聚合，将其返回给应用。

由于TDengine在vnode内将标签数据与时序数据分离存储，通过在内存里过滤标签数据，先找到需要参与聚合操作的表的集合，将需要扫描的数据集大幅减少，大幅提升聚合计算速度。同时，由于数据分布在多个vnode/dnode，聚合计算操作在多个vnode里并发进行，又进一步提升了聚合的速度。 对普通表的聚合函数以及绝大部分操作都适用于超级表，语法完全一样，细节请看 TAOS SQL。

3.6.4. 预计算

为有效提升查询处理的性能，针对物联网数据的不可更改的特点，在数据块头部记录该数据块中存储数据的统计信息：包括最大值、最小值、和。我们称之为预计算单元。如果查询处理涉及整个数据块的全部数据，直接使用预计算结果，完全不需要读取数据块的内容。由于预计算数据量远小于磁盘上存储的数据块数据的大小，对于磁盘IO为瓶颈的查询处理，使用预计算结果可以极大地减小读取IO压力，加速查询处理的流程。预计算机制与Postgre SQL的索引BRIN（block range index）有异曲同工之妙。

4. TDengine数据建模

TDengine采用关系型数据模型，需要建库、建表。因此对于一个具体的应用场景，需要考虑库的设计，超级表和普通表的设计。本节不讨论细致的语法规则，只介绍概念。

关于数据建模请参考视频教程。

4.1. 创建库

不同类型的数据采集点往往具有不同的数据特征，包括数据采集频率的高低，数据保留时间的长短，副本的数目，数据块的大小，是否允许更新数据等等。为了在各种场景下TDengine都能最大效率的工作，TDengine建议将不同数据特征的表创建在不同的库里，因为每个库可以配置不同的存储策略。创建一个库时，除SQL标准的选项外，应用还可以指定保留时长、副本数、内存块个数、时间精度、文件块里最大最小记录条数、是否压缩、一个数据文件覆盖的天数等多种参数。比如：

```
CREATE DATABASE power KEEP 365 DAYS 10 BLOCKS 6 UPDATE 1;
```

上述语句将创建一个名为power的库，这个库的数据将保留365天（超过365天将被自动删除），每10天一个数据文件，内存块数为4，允许更新数据。详细的语法及参数请见 TAOS SQL 的数据管理 章节。

创建库之后，需要使用SQL命令USE将当前库切换过来，例如：

```
USE power;
```

将当前连接里操作的库换为power，否则对具体表操作前，需要使用“库名.表名”来指定库的名字。

注意：

- 任何一张表或超级表是属于一个库的，在创建表之前，必须先创建库。
- 处于两个不同库的表是不能进行JOIN操作的。
- 创建并插入记录、查询历史记录的时候，均需要指定时间戳。

4.2. 创建超级表

一个物联网系统，往往存在多种类型的设备，比如对于电网，存在智能电表、变压器、母线、开关等等。为便于多表之间的聚合，使用TDengine, 需要对每个类型的数据采集点创建一个超级表。以表1中的智能电表为例，可以使用如下的SQL命令创建超级表：

```
CREATE STABLE meters (ts timestamp, current float, voltage int, phase float) TAGS
(location binary(64), groupId int);
```

注意：这一指令中的 STABLE 关键字，在 2.0.15 之前的版本中需写作 TABLE 。

与创建普通表一样，创建表时，需要提供表名（示例中为meters），表结构Schema，即数据列的定义。第一列必须为时间戳（示例中为ts），其他列为采集的物理量（示例中为current, voltage, phase），数据类型可以为整型、浮点型、字符串等。除此之外，还需要提供标签的schema (示例中为location, groupId)，标签的数据类型可以为整型、浮点型、字符串等。采集点的静态属性往往可以作为标签，比如采集点的地理位置、设备型号、设备组ID、管理员ID等等。标签的schema可以事后增加、删除、修改。具体定义以及细节请见 TAOS SQL 的超级表管理 章节。

每一种类型的数据采集点需要建立一个超级表，因此一个物联网系统，往往会有多个超级表。对于电网，我们就需要对智能电表、变压器、母线、开关等都建立一个超级表。在物联网中，一个设备就可能有多个数据采集点（比如一台风力发电的风机，有的采集点采集电流、电压等电参数，有的采集点采集温度、湿度、风向等环境参数），这个时候，对这一类型的设备，需要建立多张超级表。一张超级表里包含的采集物理量必须是同时采集的（时间戳是一致的）。

一张超级表最多容许1024列，如果一个采集点采集的物理量个数超过1024，需要建多张超级表来处理。一个系统可以有多个DB，一个DB里可以有一到多个超级表。

4.3. 创建表

TDengine对每个数据采集点需要独立建表。与标准的关系型数据库一样，一张表有表名，Schema，但除此之外，还可以带有一到多个标签。创建时，需要使用超级表做模板，同时指定标签的具体值。以表1中的智能电表为例，可以使用如下的SQL命令建表：

```
CREATE TABLE d1001 USING meters TAGS ("Beijing.Chaoyang", 2);
```

其中d1001是表名，meters是超级表的表名，后面紧跟标签Location的具体标签值"Beijing.Chaoyang"，标签groupId的具体标签值2。虽然在创建表时，需要指定标签值，但可以事后修改。详细细则请见 TAOS SQL 的表管理 章节。

注意：目前 TDengine 没有从技术层面限制使用一个 database （dbA）的超级表作为模板建立另一个 database （dbB）的子表，后续会禁止这种用法，不建议使用这种方法建表。

TDengine建议将数据采集点的全局唯一ID作为表名(比如设备序列号）。但对于有的场景，并没有唯一的ID，可以将多个ID组合成一个唯一的ID。不建议将具有唯一性的ID作为标签值。

自动建表：在某些特殊场景中，用户在写数据时并不确定某个数据采集点的表是否存在，此时可在写入数据时使用自动建表语法来创建不存在的表，若该表已存在则不会建立新表。比如：

```
INSERT INTO d1001 USING meters TAGS ("Beijng.Chaoyang", 2) VALUES (now, 10.2, 219, 0.32);
```

上述SQL语句将记录 (now, 10.2, 219, 0.32) 插入表d1001。如果表d1001还未创建，则使用超级表meters做模板自动创建，同时打上标签值 “Beijing.Chaoyang", 2。

关于自动建表的详细语法请参见 插入记录时自动建表 章节。

4.4. 多列模型 vs 单列模型

TDengine支持多列模型，只要物理量是一个数据采集点同时采集的（时间戳一致），这些量就可以作为不同列放在一张超级表里。但还有一种极限的设计，单列模型，每个采集的物理量都单独建表，因此每种类型的物理量都单独建立一超级表。比如电流、电压、相位，就建三张超级表。

TDengine建议尽可能采用多列模型，因为插入效率以及存储效率更高。但对于有些场景，一个采集点的采集量的种类经常变化，这个时候，如果采用多列模型，就需要频繁修改超级表的结构定义，让应用变的复杂，这个时候，采用单列模型会显得更简单。

5. 高效写入数据

TDengine支持多种接口写入数据，包括SQL, Prometheus, Telegraf, EMQ MQTT Broker, HiveMQ Broker, CSV文件等，后续还将提供Kafka, OPC等接口。数据可以单条插入，也可以批量插入，可以插入一个数据采集点的数据，也可以同时插入多个数据采集点的数据。支持多线程插入，支持时间乱序数据插入，也支持历史数据插入。

5.1. SQL写入

应用通过C/C++、JDBC、GO、C#或Python Connector 执行SQL insert语句来插入数据，用户还可以通过TAOS Shell，手动输入SQL insert语句插入数据。比如下面这条insert 就将一条记录写入到表d1001中：

```
INSERT INTO d1001 VALUES (1538548685000, 10.3, 219, 0.31);
```

TDengine支持一次写入多条记录，比如下面这条命令就将两条记录写入到表d1001中：

```
INSERT INTO d1001 VALUES (1538548684000, 10.2, 220, 0.23) (1538548696650, 10.3,
218, 0.25);
```

TDengine也支持一次向多个表写入数据，比如下面这条命令就向d1001写入两条记录，向d1002写入一条记录：

```
INSERT INTO d1001 VALUES (1538548685000, 10.3, 219, 0.31) (1538548695000, 12.6,
218, 0.33) d1002 VALUES (1538548696800, 12.3, 221, 0.31);
```

详细的SQL INSERT语法规则请见 TAOS SQL 的数据写入 章节。

Tips:

- 要提高写入效率，需要批量写入。一批写入的记录条数越多，插入效率就越高。但一条记录不能超过16K，一条SQL语句总长度不能超过64K（可通过参数maxSQLLength配置，最大可配置为1M）。
- TDengine支持多线程同时写入，要进一步提高写入速度，一个客户端需要打开20个以上的线程同时写。但线程数达到一定数量后，无法再提高，甚至还会下降，因为线程频繁切换，带来额外开销。
- 对同一张表，如果新插入记录的时间戳已经存在，默认情形下（UPDATE=0）新记录将被直接抛弃，也就是说，在一张表里，时间戳必须是唯一的。如果应用自动生成记录，很有可能生成的时间戳是一样的，这样，成功插入的记录条数会小于应用插入的记录条数。如果在创建数据库时使用了 UPDATE 1 选项，插入相同时间戳的新记录将覆盖原有记录。
- 写入的数据的时间戳必须大于当前时间减去配置参数keep的时间。如果keep配置为3650天，那么无法写入比3650天还早的数据。写入数据的时间戳也不能大于当前时间加配置参数days。如果days为2，那么无法写入比当前时间还晚2天的数据。

5.2. Prometheus直接写入

Prometheus作为Cloud Native Computing Fundation毕业的项目，在性能监控以及K8S性能监控领域有着非常广泛的应用。TDengine提供一个小工具Bailongma，只需对Prometheus做简单配置，无需任何代码，就可将Prometheus采集的数据直接写入TDengine，并按规则在TDengine自动创建库和相关表项。博文用Docker容器快速搭建一个Devops监控Demo即是采用Bailongma将Prometheus和Telegraf的数据写入TDengine中的示例，可以参考。

5.2.1. 从源代码编译blm_prometheus

用户需要从github下载Bailongma的源码，使用Golang语言编译器编译生成可执行文件。在开始编译前，需要准备好以下条件：

- Linux操作系统的服务器
- 安装好Golang，1.10版本以上
- 对应的TDengine版本。因为用到了TDengine的客户端动态链接库，因此需要安装好和服务端相同版本的TDengine程序；比如服务端版本是TDengine 2.0.0, 则在Bailongma所在的Linux服务器（可以与TDengine在同一台服务器，或者不同服务器）

Bailongma项目中有一个文件夹blm_prometheus，存放了prometheus的写入API程序。编译过程如下：

```
cd blm_prometheus
go build
```

一切正常的情况下，就会在对应的目录下生成一个blm_prometheus的可执行程序。

5.2.2. 安装Prometheus

通过Prometheus的官网下载安装。具体请见：下载地址。

5.2.3. 配置Prometheus

参考Prometheus的配置文档，在Prometheus的配置文件中的<remote_write>部分，增加以下配置：

```
  - url: "bailongma API服务提供的URL"（参考下面的blm_prometheus启动示例章节）
```

启动Prometheus后，可以通过taos客户端查询确认数据是否成功写入。

5.2.4. 启动blm_prometheus程序

blm_prometheus程序有以下选项，在启动blm_prometheus程序时可以通过设定这些选项来设定blm_prometheus的配置。

```
--tdengine-name
如果TDengine安装在一台具备域名的服务器上，也可以通过配置TDengine的域名来访问TDengine。在K8S环境下，可以配置成TDengine所运行的service name。

--batch-size
blm_prometheus会将收到的prometheus的数据拼装成TDengine的写入请求，这个参数控制一次发给TDengine的写入请求中携带的数据条数。

--dbname
设置在TDengine中创建的数据库名称，blm_prometheus会自动在TDengine中创建一个以dbname为名称的数据库，缺省值是prometheus。

--dbuser
设置访问TDengine的用户名，缺省值是'root'。

--dbpassword
设置访问TDengine的密码，缺省值是'taosdata'。

--port
blm_prometheus对prometheus提供服务的端口号。
```

5.2.5. 启动示例

通过以下命令启动一个blm_prometheus的API服务

```
./blm_prometheus -port 8088
```

假设blm_prometheus所在服务器的IP地址为"10.1.2.3"，则在prometheus的配置文件中<remote_write>部分增加url为

```
remote_write:
  - url: "http://10.1.2.3:8088/receive"
```

5.2.6. 查询prometheus写入数据

prometheus产生的数据格式如下：

```
{
  Timestamp: 1576466279341,
  Value: 37.000000,
  apiserver_request_latencies_bucket {
    component="apiserver",
    instance="192.168.99.116:8443",
    job="kubernetes-apiservers",
    le="125000",
    resource="persistentvolumes", s
    cope="cluster",
    verb="LIST",
    version="v1"
  }
}
```

其中，apiserver_request_latencies_bucket为prometheus采集的时序数据的名称，后面{}中的为该时序数据的标签。blm_prometheus会以时序数据的名称在TDengine中自动创建一个超级表，并将{}中的标签转换成TDengine的tag值，Timestamp作为时间戳，value作为该时序数据的值。因此在TDengine的客户端中，可以通过以下指令查到这个数据是否成功写入。

```
use prometheus;
select * from apiserver_request_latencies_bucket;
```

5.3. Telegraf直接写入

Telegraf是一流行的IT运维数据采集开源工具，TDengine提供一个小工具Bailongma，只需在Telegraf做简单配置，无需任何代码，就可将Telegraf采集的数据直接写入TDengine，并按规则在TDengine自动创建库和相关表项。博文用Docker容器快速搭建一个Devops监控Demo即是采用bailongma将Prometheus和Telegraf的数据写入TDengine中的示例，可以参考。

5.3.1. 从源代码编译blm_telegraf

用户需要从github下载Bailongma的源码，使用Golang语言编译器编译生成可执行文件。在开始编译前，需要准备好以下条件：

- Linux操作系统的服务器
- 安装好Golang，1.10版本以上
- 对应的TDengine版本。因为用到了TDengine的客户端动态链接库，因此需要安装好和服务端相同版本的TDengine程序；比如服务端版本是TDengine 2.0.0, 则在Bailongma所在的Linux服务器（可以与TDengine在同一台服务器，或者不同服务器）

Bailongma项目中有一个文件夹blm_telegraf，存放了Telegraf的写入API程序。编译过程如下：

```
cd blm_telegraf
go build
```

一切正常的情况下，就会在对应的目录下生成一个blm_telegraf的可执行程序。

5.3.2. 安装Telegraf

目前TDengine支持Telegraf 1.7.4以上的版本。用户可以根据当前的操作系统，到Telegraf官网下载安装包，并执行安装。下载地址如下：https://portal.influxdata.com/downloads 。

5.3.3. 配置Telegraf

修改Telegraf配置文件/etc/telegraf/telegraf.conf中与TDengine有关的配置项。

在output plugins部分，增加[[outputs.http]]配置项：

- url：Bailongma API服务提供的URL，参考下面的启动示例章节
- data_format："json"
- json_timestamp_units："1ms"

在agent部分：

- hostname: 区分不同采集设备的机器名称，需确保其唯一性。
- metric_batch_size: 100，允许Telegraf每批次写入记录最大数量，增大其数量可以降低Telegraf的请求发送频率。

关于如何使用Telegraf采集数据以及更多有关使用Telegraf的信息，请参考Telegraf官方的文档。

5.3.4. 启动blm_telegraf程序

blm_telegraf程序有以下选项，在启动blm_telegraf程序时可以通过设定这些选项来设定blm_telegraf的配置。

```
--host
TDengine服务端的IP地址，缺省值为空。

--batch-size
blm_telegraf会将收到的telegraf的数据拼装成TDengine的写入请求，这个参数控制一次发给TDengine的写入请求中携带的数据条数。

--dbname
设置在TDengine中创建的数据库名称，blm_telegraf会自动在TDengine中创建一个以dbname为名称的数据库，缺省值是prometheus。
```

```
--dbuser
设置访问TDengine的用户名，缺省值是'root'。

--dbpassword
设置访问TDengine的密码，缺省值是'taosdata'。

--port
blm_telegraf对telegraf提供服务的端口号。
```

5.3.5. 启动示例

通过以下命令启动一个blm_telegraf的API服务：

```
./blm_telegraf -host 127.0.0.1 -port 8089
```

假设blm_telegraf所在服务器的IP地址为"10.1.2.3"，则在telegraf的配置文件中，在output plugins部分，增加[[outputs.http]]配置项：

```
url = "http://10.1.2.3:8089/telegraf"
```

5.3.6. 查询telegraf写入数据

telegraf产生的数据格式如下：

```
{
  "fields": {
    "usage_guest": 0,
    "usage_guest_nice": 0,
    "usage_idle": 89.7897897897898,
    "usage_iowait": 0,
    "usage_irq": 0,
    "usage_nice": 0,
    "usage_softirq": 0,
    "usage_steal": 0,
    "usage_system": 5.405405405405405,
    "usage_user": 4.804804804804805
  },

  "name": "cpu",
  "tags": {
    "cpu": "cpu2",
    "host": "bogon"
  },
```

```
  "timestamp": 1576464360
}
```

其中，name字段为telegraf采集的时序数据的名称，tags字段为该时序数据的标签。blm_telegraf会以时序数据的名称在TDengine中自动创建一个超级表，并将tags字段中的标签转换成TDengine的tag值，timestamp作为时间戳，fields字段中的值作为该时序数据的值。因此在TDengine的客户端中，可以通过以下指令查到这个数据是否成功写入。

```
use telegraf;
select * from cpu;
```

5.4. EMQ Broker 直接写入

MQTT是流行的物联网数据传输协议，EMQ是一开源的MQTT Broker软件，无需任何代码，只需要在EMQ Dashboard里使用“规则”做简单配置，即可将MQTT的数据直接写入TDengine。EMQ X 支持通过 发送到 Web 服务的方式保存数据到 TDEngine，也在企业版上提供原生的 TDEngine 驱动实现直接保存。详细使用方法请参考 EMQ 官方文档。

5.5. HiveMQ Broker 直接写入

HiveMQ 是一个提供免费个人版和企业版的 MQTT 代理，主要用于企业和新兴的机器到机器M2M通讯和内部传输，满足可伸缩性、易管理和安全特性。HiveMQ 提供了开源的插件开发包。可以通过 HiveMQ extension - TDengine 保存数据到 TDengine。详细使用方法请参考 HiveMQ extension - TDengine 说明文档。

6. 高效查询数据

6.1. 主要查询功能

TDengine 采用 SQL 作为查询语言。应用程序可以通过 C/C++, Java, Go, Python 连接器发送 SQL 语句，用户可以通过 TDengine 提供的命令行（Command Line Interface, CLI）工具 TAOS Shell 手动执行 SQL 即席查询（Ad-Hoc Query）。TDengine 支持如下查询功能：

- 单列、多列数据查询
- 标签和数值的多种过滤条件：>, <, =, <>, like 等
- 聚合结果的分组（Group by）、排序（Order by）、约束输出（Limit/Offset）
- 数值列及聚合结果的四则运算
- 时间戳对齐的连接查询（Join Query: 隐式连接）操作
- 多种聚合/计算函数: count, max, min, avg, sum, twa, stddev, leastsquares, top, bottom, first, last, percentile, apercentile, last_row, spread, diff等

例如：在TAOS Shell中，从表d1001中查询出voltage > 215的记录，按时间降序排列，仅仅输出2条。

```
taos> select * from d1001 where voltage > 215 order by ts desc limit 2;
           ts            |       current        |   voltage   |        phase
  |
================================================================================
===
 2018-10-03 14:38:16.800 |             12.30000 |         221 |
 0.31000 |
 2018-10-03 14:38:15.000 |             12.60000 |         218 |
 0.33000 |
Query OK, 2 row(s) in set (0.001100s)
```

为满足物联网场景的需求，TDengine支持几个特殊的函数，比如twa(时间加权平均)，spread (最大值与最小值的差)，last_row(最后一条记录)等，更多与物联网场景相关的函数将添加进来。TDengine还支持连续查询。

具体的查询语法请看 TAOS SQL 的数据查询 章节。

6.2. 多表聚合查询

物联网场景中，往往同一个类型的数据采集点有多个。TDengine采用超级表(STable)的概念来描述某一个类型的数据采集点，一张普通的表来描述一个具体的数据采集点。同时TDengine使用标签来描述数据采集点的静态属性，一个具体的数据采集点有具体的标签值。通过指定标签的过滤条件，TDengine提供了一高效的方法将超级表(某一类型的数据采集点)所属的子表进行聚合查询。对普通表的聚合函数以及绝大部分操作都适用于超级表，语法完全一样。

示例1：在TAOS Shell，查找北京所有智能电表采集的电压平均值，并按照location分组

```
taos> SELECT AVG(voltage) FROM meters GROUP BY location;
       avg(voltage)        |            location            |
============================================================
             222.000000000 | Beijing.Haidian                |
             219.200000000 | Beijing.Chaoyang               |
Query OK, 2 row(s) in set (0.002136s)
```

示例2：在TAOS shell, 查找groupId为2的所有智能电表过去24小时的记录条数，电流的最大值

```
taos> SELECT count(*), max(current) FROM meters where groupId = 2 and ts > now - 24h;
     cunt(*)  |    max(current)  |
==================================
            5 |             13.4 |
Query OK, 1 row(s) in set (0.002136s)
```

TDengine仅容许对属于同一个超级表的表之间进行聚合查询，不同超级表之间的聚合查询不支持。在 TAOS SQL 的数据查询 一章，查询类操作都会注明是否支持超级表。

6.3. 降采样查询、插值

物联网场景里，经常需要通过降采样（down sampling）将采集的数据按时间段进行聚合。TDengine 提供了一个简便的关键词 interval 让按照时间窗口的查询操作变得极为简单。比如，将智能电表 d1001 采集的电流值每10秒钟求和

```
taos> SELECT sum(current) FROM d1001 INTERVAL(10s);
           ts            |       sum(current)        |
=========================================================
 2018-10-03 14:38:00.000 |             10.300000191 |
 2018-10-03 14:38:10.000 |             24.900000572 |
Query OK, 2 row(s) in set (0.000883s)
```

降采样操作也适用于超级表，比如：将北京所有智能电表采集的电流值每秒钟求和

```
taos> SELECT SUM(current) FROM meters where location like "Beijing%" INTERVAL(1s);
           ts            |       sum(current)        |
=========================================================
 2018-10-03 14:38:04.000 |             10.199999809 |
 2018-10-03 14:38:05.000 |             32.900000572 |
 2018-10-03 14:38:06.000 |             11.500000000 |
 2018-10-03 14:38:15.000 |             12.600000381 |
 2018-10-03 14:38:16.000 |             36.000000000 |
Query OK, 5 row(s) in set (0.001538s)
```

降采样操作也支持时间偏移，比如：将所有智能电表采集的电流值每秒钟求和，但要求每个时间窗口从 500 毫秒开始

```
taos> SELECT SUM(current) FROM meters INTERVAL(1s, 500a);
           ts            |       sum(current)        |
=========================================================
 2018-10-03 14:38:04.500 |              11.189999809 |
 2018-10-03 14:38:05.500 |              31.900000572 |
 2018-10-03 14:38:06.500 |              11.600000000 |
 2018-10-03 14:38:15.500 |              12.300000381 |
 2018-10-03 14:38:16.500 |              35.000000000 |
Query OK, 5 row(s) in set (0.001521s)
```

物联网场景里，每个数据采集点采集数据的时间是难同步的，但很多分析算法(比如FFT)需要把采集的数据严格按照时间等间隔的对齐，在很多系统里，需要应用自己写程序来处理，但使用TDengine的降采样操作就轻松解决。如果一个时间间隔里，没有采集的数据，TDengine还提供插值计算的功能。

语法规则细节请见 TAOS SQL 的时间维度聚合 章节。

7. 高级功能

7.1. 连续查询（Continuous Query）

连续查询是TDengine定期自动执行的查询，采用滑动窗口的方式进行计算，是一种简化的时间驱动的流式计算。针对库中的表或超级表，TDengine可提供定期自动执行的连续查询，用户可让TDengine推送查询的结果，也可以将结果再写回到TDengine中。每次执行的查询是一个时间窗口，时间窗口随着时间流动向前滑动。在定义连续查询的时候需要指定时间窗口（time window, 参数interval）大小和每次前向增量时间（forward sliding times, 参数sliding）。

TDengine的连续查询采用时间驱动模式，可以直接使用TAOS SQL进行定义，不需要额外的操作。使用连续查询，可以方便快捷地按照时间窗口生成结果，从而对原始采集数据进行降采样（down sampling）。用户通过TAOS SQL定义连续查询以后，TDengine自动在最后的一个完整的时间周期末端拉起查询，并将计算获得的结果推送给用户或者写回TDengine。

TDengine提供的连续查询与普通流计算中的时间窗口计算具有以下区别：

- 不同于流计算的实时反馈计算结果，连续查询只在时间窗口关闭以后才开始计算。例如时间周期是1天，那么当天的结果只会在23:59:59以后才会生成。
- 如果有历史记录写入到已经计算完成的时间区间，连续查询并不会重新进行计算，也不会重新将结果推送给用户。对于写回TDengine的模式，也不会更新已经存在的计算结果。
- 使用连续查询推送结果的模式，服务端并不缓存客户端计算状态，也不提供Exactly-Once的语意保证。如果用户的应用端崩溃，再次拉起的连续查询将只会从再次拉起的时间开始重新计算最近的一个完整的时间窗口。如果使用写回模式，TDengine可确保数据写回的有效性和连续性。

7.1.1. 使用连续查询

下面以智能电表场景为例介绍连续查询的具体使用方法。假设我们通过下列SQL语句创建了超级表和子表：

```
create table meters (ts timestamp, current float, voltage int, phase float) tags
(location binary(64), groupId int);
create table D1001 using meters tags ("Beijing.Chaoyang", 2);
create table D1002 using meters tags ("Beijing.Haidian", 2);
...
```

我们已经知道，可以通过下面这条SQL语句以一分钟为时间窗口、30秒为前向增量统计这些电表的平均电压。

```
select avg(voltage) from meters interval(1m) sliding(30s);
```

每次执行这条语句，都会重新计算所有数据。 如果需要每隔30秒执行一次来增量计算最近一分钟的数据，可以把上面的语句改进成下面的样子，每次使用不同的 `startTime` 并定期执行：

```
select avg(voltage) from meters where ts > {startTime} interval(1m) sliding(30s);
```

这样做没有问题，但TDengine提供了更简单的方法，只要在最初的查询语句前面加上 `create table {tableName} as` 就可以了，例如：

```
create table avg_vol as select avg(voltage) from meters interval(1m) sliding(30s);
```

会自动创建一个名为 `avg_vol` 的新表，然后每隔30秒，TDengine会增量执行 `as` 后面的 SQL 语句，并将查询结果写入这个表中，用户程序后续只要从 `avg_vol` 中查询数据即可。例如：

```
taos> select * from avg_vol;
           ts           |        avg_voltage_    |
=================================================
 2020-07-29 13:37:30.000 |            222.0000000 |
 2020-07-29 13:38:00.000 |            221.3500000 |
 2020-07-29 13:38:30.000 |            220.1700000 |
 2020-07-29 13:39:00.000 |            223.0800000 |
```

需要注意，查询时间窗口的最小值是10毫秒，没有时间窗口范围的上限。

此外，TDengine还支持用户指定连续查询的起止时间。如果不输入开始时间，连续查询将从第一条原始数据所在的时间窗口开始；如果没有输入结束时间，连续查询将永久运行；如果用户指定了结束时间，连续查询在系统时间达到指定的时间以后停止运行。比如使用下面的SQL创建的连续查询将运行一小时，之后会自动停止。

```
create table avg_vol as select avg(voltage) from meters where ts > now and ts <= now + 1h interval(1m) sliding(30s);
```

需要说明的是，上面例子中的 `now` 是指创建连续查询的时间，而不是查询执行的时间，否则，查询就无法自动停止了。另外，为了尽量避免原始数据延迟写入导致的问题，TDengine中连续查询的计算有一定的延迟。也就是说，一个时间窗口过去后，TDengine并不会立即计算这个窗口的数据，所以要稍等一会（一般不会超过1分钟）才能查到计算结果。

7.1.2. 管理连续查询

用户可在控制台中通过 `show streams` 命令来查看系统中全部运行的连续查询，并可以通过 `kill stream` 命令杀掉对应的连续查询。后续版本会提供更细粒度和便捷的连续查询管理命令。

7.2. 数据订阅（Publisher/Subscriber）

基于数据天然的时间序列特性，TDengine的数据写入（insert）与消息系统的数据发布（pub）逻辑上一致，均可视为系统中插入一条带时间戳的新记录。同时，TDengine在内部严格按照数据时间序列单调递增的方式保存数据。本质上来说，TDengine中里每一张表均可视为一个标准的消息队列。

TDengine内嵌支持轻量级的消息订阅与推送服务。使用系统提供的API，用户可使用普通查询语句订阅数据库中的一张或多张表。订阅的逻辑和操作状态的维护均是由客户端完成，客户端定时轮询服务器是否有新的记录到达，有新的记录到达就会将结果反馈到客户。

TDengine的订阅与推送服务的状态是客户端维持，TDengine服务器并不维持。因此如果应用重启，从哪个时间点开始获取最新数据，由应用决定。

TDengine的API中，与订阅相关的主要有以下三个：

```
taos_subscribe
taos_consume
taos_unsubscribe
```

这些API的文档请见 C/C++ Connector，下面仍以智能电表场景为例介绍一下它们的具体用法（超级表和子表结构请参考上一节“连续查询”），完整的示例代码可以在 这里 找到。

如果我们希望当某个电表的电流超过一定限制（比如10A）后能得到通知并进行一些处理， 有两种方法：一是分别对每张子表进行查询，每次查询后记录最后一条数据的时间戳，后续只查询这个时间戳之后的数据：

```
select * from D1001 where ts > {last_timestamp1} and current > 10;
select * from D1002 where ts > {last_timestamp2} and current > 10;
...
```

这确实可行，但随着电表数量的增加，查询数量也会增加，客户端和服务端的性能都会受到影响，当电表数增长到一定的程度，系统就无法承受了。

另一种方法是对超级表进行查询。这样，无论有多少电表，都只需一次查询：

```
select * from meters where ts > {last_timestamp} and current > 10;
```

但是，如何选择 last_timestamp 就成了一个新的问题。因为，一方面数据的产生时间（也就是数据时间戳）和数据入库的时间一般并不相同，有时偏差还很大；另一方面，不同电表的数据到达TDengine的时间也会有差异。所以，如果我们在查询中使用最慢的那台电表的数据的时间戳作为 last_timestamp，就可能重复读入其它电表的数据；如果使用最快的电表的时间戳，其它电表的数据就可能被漏掉。

TDengine的订阅功能为上面这个问题提供了一个彻底的解决方案。

首先是使用taos_subscribe创建订阅：

```
TAOS_SUB* tsub = NULL;
if (async) {
   // create an asynchronized subscription, the callback function will be called every 1s
   tsub = taos_subscribe(taos, restart, topic, sql, subscribe_callback, &blockFetch, 1000);
} else {
   // create an synchronized subscription, need to call 'taos_consume' manually
   tsub = taos_subscribe(taos, restart, topic, sql, NULL, NULL, 0);
}
```

TDengine中的订阅既可以是同步的，也可以是异步的，上面的代码会根据从命令行获取的参数async的值来决定使用哪种方式。这里，同步的意思是用户程序要直接调用taos_consume来拉取数据，而异步则由API在内部的另一个线程中调用taos_consume，然后把拉取到的数据交给回调函数subscribe_callback去处理。（注意，subscribe_callback 中不宜做较为耗时的操作，否则有可能导致客户端阻塞等不可控的问题。）

参数taos是一个已经建立好的数据库连接，在同步模式下无特殊要求。但在异步模式下，需要注意它不会被其它线程使用，否则可能导致不可预计的错误，因为回调函数在API的内部线程中被调用，而TDengine的部分API不是线程安全的。

参数sql是查询语句，可以在其中使用where子句指定过滤条件。在我们的例子中，如果只想订阅电流超过10A时的数据，可以这样写：

```
select * from meters where current > 10;
```

注意，这里没有指定起始时间，所以会读到所有时间的数据。如果只想从一天前的数据开始订阅，而不需要更早的历史数据，可以再加上一个时间条件：

```
select * from meters where ts > now - 1d and current > 10;
```

订阅的topic实际上是它的名字，因为订阅功能是在客户端API中实现的，所以没必要保证它全局唯一，但需要它在一台客户端机器上唯一。

如果名为topic的订阅不存在，参数restart没有意义；但如果用户程序创建这个订阅后退出，当它再次启动并重新使用这个topic时，restart就会被用于决定是从头开始读取数据，还是接续上次的位置进行读取。本例中，如果restart是 **true**（非零值），用户程序肯定会读到所有数据。但如果这个订阅之前就存在了，并且已经读取了一部分数据，且restart是 **false**（**0**），用户程序就不会读到之前已经读取的数据了。

taos_subscribe的最后一个参数是以毫秒为单位的轮询周期。在同步模式下，如果前后两次调用taos_consume的时间间隔小于此时间，taos_consume会阻塞，直到间隔超过此时间。异步模式下，这个时间是两次调用回调函数的最小时间间隔。

taos_subscribe的倒数第二个参数用于用户程序向回调函数传递附加参数，订阅API不对其做任何处理，只原样传递给回调函数。此参数在同步模式下无意义。

订阅创建以后，就可以消费其数据了，同步模式下，示例代码是下面的 else 部分：

```
if (async) {
   getchar();
} else while(1) {
   TAOS_RES* res = taos_consume(tsub);
   if (res == NULL) {
      printf("failed to consume data.");
      break;
   } else {
      print_result(res, blockFetch);
      getchar();
   }
}
```

这里是一个 **while** 循环，用户每按一次回车键就调用一次taos_consume，而taos_consume的返回值是查询到的结果集，与taos_use_result完全相同，例子中使用这个结果集的代码是函数print_result：

```
void print_result(TAOS_RES* res, int blockFetch) {
    TAOS_ROW row = NULL;
    int num_fields = taos_num_fields(res);
    TAOS_FIELD* fields = taos_fetch_fields(res);
    int nRows = 0;
    if (blockFetch) {
        nRows = taos_fetch_block(res, &row);
        for (int i = 0; i < nRows; i++) {
            char temp[256];
            taos_print_row(temp, row + i, fields, num_fields);
            puts(temp);
        }
    } else {
        while ((row = taos_fetch_row(res))) {
            char temp[256];
            taos_print_row(temp, row, fields, num_fields);
            puts(temp);
            nRows++;
        }
    }
    printf("%d rows consumed.\n", nRows);
}
```

其中的 taos_print_row 用于处理订阅到数据，在我们的例子中，它会打印出所有符合条件的记录。而异步模式下，消费订阅到的数据则显得更为简单：

```
void subscribe_callback(TAOS_SUB* tsub, TAOS_RES *res, void* param, int code) {
    print_result(res, *(int*)param);
}
```

当要结束一次数据订阅时，需要调用taos_unsubscribe：

```
taos_unsubscribe(tsub, keep);
```

其第二个参数，用于决定是否在客户端保留订阅的进度信息。如果这个参数是**false**（**0**），那无论下次调用taos_subscribe时的restart参数是什么，订阅都只能重新开始。另外，进度信息的保存位置是 *{DataDir}/subscribe/* 这个目录下，每个订阅有一个与其topic同名的文件，删掉某个文件，同样会导致下次创建其对应的订阅时只能重新开始。

代码介绍完毕，我们来看一下实际的运行效果。假设：

- 示例代码已经下载到本地
- TDengine 也已经在同一台机器上安装好
- 示例所需的数据库、超级表、子表已经全部创建好

则可以在示例代码所在目录执行以下命令来编译并启动示例程序：

```
$ make
$ ./subscribe -sql='select * from meters where current > 10;'
```

示例程序启动后，打开另一个终端窗口，启动 TDengine 的 shell 向 **D1001** 插入一条电流为 12A 的数据：

```
$ taos
> use test;
> insert into D1001 values(now, 12, 220, 1);
```

这时，因为电流超过了10A，您应该可以看到示例程序将它输出到了屏幕上。您可以继续插入一些数据观察示例程序的输出。

7.2.1. Java 使用数据订阅功能

订阅功能也提供了 Java 开发接口，相关说明请见 Java Connector。需要注意的是，目前 Java 接口没有提供异步订阅模式，但用户程序可以通过创建 `TimerTask` 等方式达到同样的效果。

下面以一个示例程序介绍其具体使用方法。它所完成的功能与前面介绍的 C 语言示例基本相同，也是订阅数据库中所有电流超过 10A 的记录。

7.2.1.1. 准备数据

```
# 创建 power 库
taos> create database power;
# 切换库
taos> use power;
# 创建超级表
taos> create table meters(ts timestamp, current float, voltage int, phase int) tags(location binary(64), groupId int);
# 创建表
taos> create table d1001 using meters tags ("Beijing.Chaoyang", 2);
taos> create table d1002 using meters tags ("Beijing.Haidian", 2);
# 插入测试数据
taos> insert into d1001 values("2020-08-15 12:00:00.000", 12, 220, 1),("2020-08-15 12:10:00.000", 12.3, 220, 2),("2020-08-15 12:20:00.000", 12.2, 220, 1);
taos> insert into d1002 values("2020-08-15 12:00:00.000", 9.9, 220, 1),("2020-08-15 12:10:00.000", 10.3, 220, 1),("2020-08-15 12:20:00.000", 11.2, 220, 1);
# 从超级表 meters 查询电流大于 10A 的记录
taos> select * from meters where current > 10;
           ts            |    current   |    voltage   |  phase |         location         |   groupid   |
================================================================================================================
 2020-08-15 12:10:00.000 |    10.30000  |     220      |      1 | Beijing.Haidian      |           2 |
 2020-08-15 12:20:00.000 |    11.20000  |     220      |      1 | Beijing.Haidian      |           2 |
 2020-08-15 12:00:00.000 |    12.00000  |     220      |      1 | Beijing.Chaoyang     |           2 |
 2020-08-15 12:10:00.000 |    12.30000  |     220      |      2 | Beijing.Chaoyang     |           2 |
```

```
 2020-08-15 12:20:00.000 |    12.20000  |     220      |      1 |
Beijing.Chaoyang     |           2 |
Query OK, 5 row(s) in set (0.004896s)
```

7.2.1.2. 示例程序

```java
public class SubscribeDemo {
    private static final String topic = "topic-meter-current-bg-10";
    private static final String sql = "select * from meters where current > 10";

    public static void main(String[] args) {
        Connection connection = null;
        TSDBSubscribe subscribe = null;

        try {
            Class.forName("com.taosdata.jdbc.TSDBDriver");
            Properties properties = new Properties();
            properties.setProperty(TSDBDriver.PROPERTY_KEY_CHARSET, "UTF-8");
            properties.setProperty(TSDBDriver.PROPERTY_KEY_TIME_ZONE, "UTC-8");
            String jdbcUrl = "jdbc:TAOS://127.0.0.1:6030/power?
user=root&password=taosdata";
            connection = DriverManager.getConnection(jdbcUrl, properties);
            subscribe = ((TSDBConnection) connection).subscribe(topic, sql, true);
// 创建订阅
            int count = 0;
            while (count < 10) {
                TimeUnit.SECONDS.sleep(1); // 等待1秒，避免频繁调用 consume，给服务端造
成压力
                TSDBResultSet resultSet = subscribe.consume(); // 消费数据
                if (resultSet == null) {
                    continue;
                }
                ResultSetMetaData metaData = resultSet.getMetaData();
                while (resultSet.next()) {
                    int columnCount = metaData.getColumnCount();
                    for (int i = 1; i <= columnCount; i++) {
                        System.out.print(metaData.getColumnLabel(i) + ": " +
resultSet.getString(i) + "\t");
                    }
                    System.out.println();
                    count++;
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (null != subscribe)
                    subscribe.close(true); // 关闭订阅
                if (connection != null)
                    connection.close();
```

```
            } catch (SQLException throwables) {
                throwables.printStackTrace();
            }
        }
    }
}
```

运行示例程序，首先，它会消费符合查询条件的所有历史数据：

```
# java -jar subscribe.jar

ts: 1597464000000 current: 12.0 voltage: 220  phase: 1  location: Beijing.Chaoyang groupid : 2
ts: 1597464600000 current: 12.3 voltage: 220  phase: 2  location: Beijing.Chaoyang groupid : 2
ts: 1597465200000 current: 12.2 voltage: 220  phase: 1  location: Beijing.Chaoyang groupid : 2
ts: 1597464600000 current: 10.3 voltage: 220  phase: 1  location: Beijing.Haidian groupid : 2
ts: 1597465200000 current: 11.2 voltage: 220  phase: 1  location: Beijing.Haidian groupid : 2
```

接着，使用 taos 客户端向表中新增一条数据：

```
# taos
taos> use power;
taos> insert into d1001 values("2020-08-15 12:40:00.000", 12.4, 220, 1);
```

因为这条数据的电流大于10A，示例程序会将其消费：

```
ts: 1597466400000 current: 12.4 voltage: 220  phase: 1  location: Beijing.Chaoyang groupid: 2
```

7.3. 缓存（Cache）

TDengine采用时间驱动缓存管理策略（First-In-First-Out，FIFO），又称为写驱动的缓存管理机制。这种策略有别于读驱动的数据缓存模式（Least-Recent-Used，LRU），直接将最近写入的数据保存在系统的缓存中。当缓存达到临界值的时候，将最早的数据批量写入磁盘。一般意义上来说，对于物联网数据的使用，用户最为关心最近产生的数据，即当前状态。TDengine充分利用了这一特性，将最近到达的（当前状态）数据保存在缓存中。

TDengine通过查询函数向用户提供毫秒级的数据获取能力。直接将最近到达的数据保存在缓存中，可以更加快速地响应用户针对最近一条或一批数据的查询分析，整体上提供更快的数据库查询响应能力。从这个意义上来说，可通过设置合适的配置参数将TDengine作为数据缓存来使用，而不需要再部署额外的缓存系统，可有效地简化系统架构，降低运维的成本。需要注意的是，TDengine重启以后系统的缓存将被清空，之前缓存的数据均会被批量写入磁盘，缓存的数据将不会像专门的key-value缓存系统再将之前缓存的数据重新加载到缓存中。

TDengine分配固定大小的内存空间作为缓存空间，缓存空间可根据应用的需求和硬件资源配置。通过适当的设置缓存空间，TDengine可以提供极高性能的写入和查询的支持。TDengine中每个虚拟节点（virtual node）创建时分配独立的缓存池。每个虚拟节点管理自己的缓存池，不同虚拟节点间不共享缓存池。每个虚拟节点内部所属的全部表共享该虚拟节点的缓存池。

TDengine将内存池按块划分进行管理，数据在内存块里是以行（row）的形式存储。一个vnode的内存池是在vnode创建时按块分配好，而且每个内存块按照先进先出的原则进行管理。在创建内存池时，块的大小由系统配置参数cache决定；每个vnode中内存块的数目则由配置参数blocks决定。因此对于一个vnode，总的内存大小为：`cache * blocks`。一个cache block需要保证每张表能存储至少几十条以上记录，才会有效率。

你可以通过函数last_row快速获取一张表或一张超级表的最后一条记录，这样很便于在大屏显示各设备的实时状态或采集值。例如：

```
1 select last_row(voltage) from meters where location='Beijing.Chaoyang';
```

该SQL语句将获取所有位于北京朝阳区的电表最后记录的电压值。

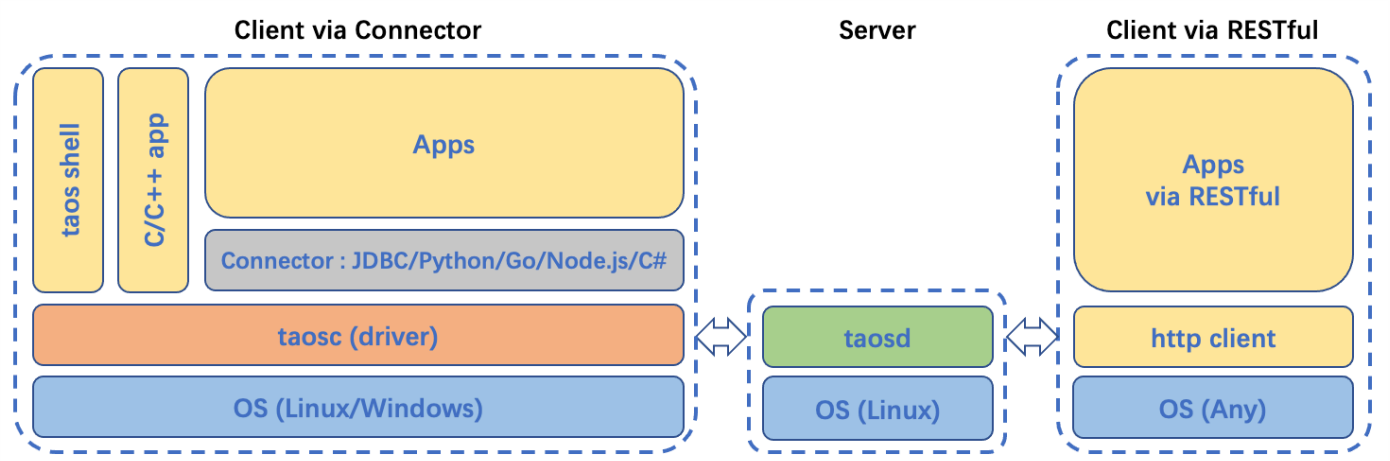
7.4. 报警监测（Alert）

在 TDengine 的应用场景中，报警监测是一个常见需求，从概念上说，它要求程序从最近一段时间的数据中筛选出符合一定条件的数据，并基于这些数据根据定义好的公式计算出一个结果，当这个结果符合某个条件且持续一定时间后，以某种形式通知用户。

为了满足用户对报警监测的需求，TDengine 以独立模块的形式提供了这一功能，有关它的安装使用方法，请参考博客 使用 TDengine 进行报警监测 。

8. 连接器

TDengine提供了丰富的应用程序开发接口，其中包括C/C++、Java、Python、Go、Node.js、C# 、RESTful 等，便于用户快速开发应用。



目前TDengine的连接器可支持的平台广泛，包括：X64/X86/ARM64/ARM32/MIPS/Alpha等硬件平台，以及Linux/Win64/Win32等开发环境。对照矩阵如下：

CPU	X64 64bit	X64 64bit	X64 64bit	X86 32bit	ARM64	ARM32	MIPS 龙芯	Alpha 申威	X64 海光
OS	Linux	Win64	Win32	Win32	Linux	Linux	Linux	Linux	Linux
C/C++	●	●	●	○	●	●	○	○	○
JDBC	●	●	●	○	●	●	○	○	○
Python	●	●	●	○	●	●	○	--	○
Go	●	●	●	○	●	●	○	--	--
NodeJs	●	●	○	○	●	●	○	--	--
C#	●	●	○	○	○	○	○	--	--
RESTful	●	●	●	●	●	●	○	○	○

其中 ● 表示经过官方测试验证， ○ 表示非官方测试验证。

注意：

- 在没有安装TDengine服务端软件的系统中使用连接器（除RESTful外）访问 TDengine 数据库，需要安装相应版本的客户端安装包来使应用驱动（Linux系统中文件名为libtaos.so，Windows系统中为taos.dll）被安装在系统中，否则会产生无法找到相应库文件的错误。
- 所有执行 SQL 语句的 API，例如 C/C++ Connector 中的 `tao_query`、`taos_query_a`、`taos_subscribe` 等，以及其它语言中与它们对应的API，每次都只能执行一条 SQL 语句，如果实际参数中包含了多条语句，它们的行为是未定义的。

- 升级 TDengine 到 2.0.8.0 版本的用户，必须更新 JDBC。连接 TDengine 必须升级 taos-jdbcdriver 到 2.0.12 及以上。详细的版本依赖关系请参见 taos-jdbcdriver 文档。
- 无论选用何种编程语言的连接器，2.0 及以上版本的 TDengine 推荐数据库应用的每个线程都建立一个独立的连接，或基于线程建立连接池，以避免连接内的“USE statement”状态量在线程之间相互干扰（但连接的查询和写入操作都是线程安全的）。

8.1. 安装连接器驱动步骤

服务器应该已经安装TDengine服务端安装包。连接器驱动安装步骤如下：

Linux

1. 从涛思官网下载 或 从涛思交付团队获得应用驱动的tar.gz安装包：

- X64硬件环境：TDengine-client-2.x.x.x-Linux-x64.tar.gz
- ARM64硬件环境：TDengine-client-2.x.x.x-Linux-aarch64.tar.gz
- ARM32硬件环境：TDengine-client-2.x.x.x-Linux-aarch32.tar.gz

2. 解压缩软件包

将软件包放置在当前用户可读写的任意目录下，然后执行下面的命令：

```
tar -xzvf TDengine-client-xxxxxxxxx.tar.gz
```

其中xxxxxxxxx需要替换为实际版本的字符串。

3. 执行安装脚本

解压软件包之后，会在解压目录下看到以下文件(目录)：

install_client.sh：安装脚本，用于应用驱动程序
taos.tar.gz：应用驱动安装包
driver：TDengine应用驱动driver
connector: 各种编程语言连接器（go/grafanaplugin/nodejs/python/JDBC）
examples: 各种编程语言的示例程序(c/C#/go/JDBC/MATLAB/python/R)

运行install_client.sh进行安装。

4. 配置taos.cfg

编辑taos.cfg文件(默认路径/etc/taos/taos.cfg)，将firstEP修改为TDengine服务器的End Point，例如：h1.taos.com:6030

提示： 如本机没有部署TDengine服务，仅安装了应用驱动，则taos.cfg中仅需配置firstEP，无需配置FQDN。

Windows x64/x86

1. 从涛思官网下载 或 从涛思交付团队获得Windows x64/x86的应用驱动的安装包：

- X64硬件环境：TDengine-client-2.X.X.X-Windows-x64.exe
- X86硬件环境：TDengine-client-2.X.X.X-Windows-x86.exe

2. 执行安装程序，按提示选择默认值，完成安装

3. 安装路径

默认安装路径为：C:\TDengine，其中包括以下文件(目录)：

taos.exe：taos shell命令行程序

cfg : 配置文件目录
driver: 应用驱动动态链接库
examples: 示例程序 bash/C/C#/go/JDBC/Python/Node.js
include: 头文件
log : 日志文件
unins000.exe: 卸载程序

4. 配置taos.cfg

编辑taos.cfg文件(默认路径C:\TDengine\cfg\taos.cfg)，将firstEP修改为TDengine服务器的End Point，例如：h1.taos.com:6030

提示：

1. 如利用FQDN连接服务器，必须确认本机网络环境DNS已配置好，或在hosts文件中添加FQDN寻址记录，如编辑C:\Windows\system32\drivers\etc\hosts，添加如下的记录：`192.168.1.99  h1.taos.com`
2. 卸载：运行unins000.exe可卸载TDengine应用驱动。

8.1.1. 安装验证

以上安装和配置完成后，并确认TDengine服务已经正常启动运行，此时可以执行taos客户端进行登录。

Linux环境：

在Linux shell下直接执行 taos，应该就能正常连接到TDegine服务，进入到taos shell界面，示例如下：

```
$ taos
Welcome to the TDengine shell from Linux, Client  Version:2.0.5.0
Copyright (c) 2017 by TAOS Data, Inc. All rights  reserved.
taos> show databases;
name       |   created_time    |   ntables  |  vgroups   | replica | quorum | days |    keep1,keep2,keep(D)   | cache(MB)|   blocks  |  minrows   |  maxrows  |  wallevel |  fsync    | comp | precision |    status  |
================================================================================================================================================================================================
test       | 2020-10-14  10:35:48.617 |     10 |      1 |    1 |   1 |     2 |  3650,3650,3650        |     16|      6 |     100 |    4096 |    1 |    3000 |  2 |  ms       | ready    |
log        | 2020-10-12  09:08:21.651 |      4 |      1 |    1 |   1 |    10 |  30,30,30              |      1|      3 |     100 |    4096 |    1 |    3000 |  2 |  us     | ready    |
Query OK, 2 row(s) in set (0.001198s)
taos>
```

Windows（x64/x86）环境：

在cmd下进入到c:\TDengine目录下直接执行 taos.exe，应该就能正常链接到tdegine服务，进入到taos shell界面，示例如下：

```
C:\TDengine>taos
Welcome to the TDengine  shell from Linux, Client Version:2.0.5.0
Copyright (c) 2017 by  TAOS Data, Inc. All rights reserved.
taos> show  databases;
name       |   created_time    |   ntables  |  vgroups   | replica | quorum | days |    keep1,keep2,keep(D)   | cache(MB)   |  blocks  |   minrows  |  maxrows | wallevel |  fsync  | comp | precision |  status    |
================================================================================================================================================================================================================================
test       | 2020-10-14  10:35:48.617 |     10 |      1 |    1 |   1 |     2 | 3650,3650,3650        |     16 |      6 |     100 |    4096 |    1 |    3000 |  2 | ms    | ready    |
log        | 2020-10-12  09:08:21.651 |      4 |      1 |    1 |   1 |    10 | 30,30,30              |      1 |      3 |     100 |    4096 |    1 |    3000 |  2 | us    | ready    |
Query OK, 2 row(s) in  set (0.045000s)
taos>
```

8.2. C/C++ Connector

C/C++连接器支持的系统有：

CPU类型	x64（64bit）			ARM64	ARM32
OS类型	Linux	Win64	Win32	Linux	Linux
支持与否	支持	支持	支持	支持	开发中

C/C++的API类似于MySQL的C API。应用程序使用时，需要包含TDengine头文件 *taos.h*，里面列出了提供的API的函数原型。安装后，taos.h位于：

- Linux：`/usr/local/taos/include`
- Windows：`C:\TDengine\include`

```
#include <taos.h>
```

注意：

- 在编译时需要链接TDengine动态库。Linux 为 *libtaos.so* ，安装后，位于 */usr/local/taos/driver*。Windows为 taos.dll，安装后位于 *C:\TDengine*。
- 如未特别说明，当API的返回值是整数时，*0* 代表成功，其它是代表失败原因的错误码，当返回值是指针时，

NULL 表示失败。

- 在 taoserror.h中有所有的错误码，以及对应的原因描述。

8.2.1. 示例程序

使用C/C++连接器的示例代码请参见 https://github.com/taosdata/TDengine/tree/develop/tests/examples/c 。

示例程序源码也可以在安装目录下的 examples/c 路径下找到：

apitest.c、asyncdemo.c、demo.c、prepare.c、stream.c、subscribe.c

该目录下有makefile，在Linux环境下，直接执行make就可以编译得到执行文件。

在一台机器上启动TDengine服务，执行这些示例程序，按照提示输入TDengine服务的FQDN，就可以正常运行，并打印出信息。

提示：在ARM环境下编译时，请将makefile中的-msse4.2打开，这个选项只有在x64/x86硬件平台上才能支持。

8.2.2. 基础API

基础API用于完成创建数据库连接等工作，为其它API的执行提供运行时环境。

- `void taos_init()`

  初始化运行环境。如果应用没有主动调用该API，那么应用在调用`taos_connect`时将自动调用，故应用程序一般无需手动调用该API。

- `void taos_cleanup()`

  清理运行环境，应用退出前应调用此API。

- `int taos_options(TSDB_OPTION option, const void * arg, ...)`

  设置客户端选项，目前支持区域设置（`TSDB_OPTION_LOCALE`）、字符集设置（`TSDB_OPTION_CHARSET`）、时区设置（`TSDB_OPTION_TIMEZONE`）、配置文件路径设置（`TSDB_OPTION_CONFIGDIR`）。区域设置、字符集、时区默认为操作系统当前设置。

- `char *taos_get_client_info()`

  获取客户端版本信息。

- `TAOS *taos_connect(const char *host, const char *user, const char *pass, const char *db, int port)`

  创建数据库连接，初始化连接上下文。其中需要用户提供的参数包含：

  - host：TDengine管理主节点的FQDN
  - user：用户名
  - pass：密码
  - db：数据库名字，如果用户没有提供，也可以正常连接，用户可以通过该连接创建新的数据库，如果用户提供了数据库名字，则说明该数据库用户已经创建好，缺省使用该数据库
  - port：TDengine管理主节点的端口号

返回值为空表示失败。应用程序需要保存返回的参数，以便后续API调用。

- `char *taos_get_server_info(TAOS *taos)`

  获取服务端版本信息。

- `int taos_select_db(TAOS *taos, const char *db)`

  将当前的缺省数据库设置为db。

- `void taos_close(TAOS *taos)`

  关闭连接，其中taos是taos_connect函数返回的指针。

8.2.3. 同步查询API

传统的数据库操作API，都属于同步操作。应用调用API后，一直处于阻塞状态，直到服务器返回结果。TDengine支持如下API：

- `TAOS_RES* taos_query(TAOS *taos, const char *sql)`

  该API用来执行SQL语句，可以是DQL、DML或DDL语句。 其中的taos参数是通过taos_connect获得的指针。不能通过返回值是否是 NULL 来判断执行结果是否失败，而是需要用taos_errno函数解析结果集中的错误代码来进行判断。

- `int taos_result_precision(TAOS_RES *res)`

  返回结果集时间戳字段的精度，`0` 代表毫秒，`1` 代表微秒，`2` 代表纳秒。

- `TAOS_ROW taos_fetch_row(TAOS_RES *res)`

  按行获取查询结果集中的数据。

- `int taos_fetch_block(TAOS_RES *res, TAOS_ROW *rows)`

  批量获取查询结果集中的数据，返回值为获取到的数据的行数。

- `int taos_num_fields(TAOS_RES *res)` 和 `int taos_field_count(TAOS_RES *res)`

  这两个API等价，用于获取查询结果集中的列数。

- `int* taos_fetch_lengths(TAOS_RES *res)`

  获取结果集中每个字段的长度。 返回值是一个数组，其长度为结果集的列数。

- `int taos_affected_rows(TAOS_RES *res)`

  获取被所执行的 SQL 语句影响的行数。

- `TAOS_FIELD *taos_fetch_fields(TAOS_RES *res)`

  获取查询结果集每列数据的属性（列的名称、列的数据类型、列的长度），与taos_num_fileds配合使用，可用来解析taos_fetch_row返回的一个元组(一行)的数据。 TAOS_FIELD 的结构如下：

```
typedef struct taosField {
  char     name[65];  // 列名
  uint8_t  type;      // 数据类型
  int16_t  bytes;     // 长度，单位是字节
} TAOS_FIELD;
```

- `void taos_stop_query(TAOS_RES *res)`

停止一个查询的执行。

- `void taos_free_result(TAOS_RES *res)`

  释放查询结果集以及相关的资源。查询完成后，务必调用该API释放资源，否则可能导致应用内存泄露。但也需注意，释放资源后，如果再调用taos_consume等获取查询结果的函数，将导致应用Crash。

- `char *taos_errstr(TAOS_RES *res)`

  获取最近一次API调用失败的原因,返回值为字符串。

- `int taos_errno(TAOS_RES *res)`

  获取最近一次API调用失败的原因，返回值为错误代码。

注意：2.0及以上版本 TDengine 推荐数据库应用的每个线程都建立一个独立的连接，或基于线程建立连接池。而不推荐在应用中将该连接 (TAOS*) 结构体传递到不同的线程共享使用。基于 TAOS 结构体发出的查询、写入等操作具有多线程安全性，但 “USE statement” 等状态量有可能在线程之间相互干扰。此外，C 语言的连接器可以按照需求动态建立面向数据库的新连接（该过程对用户不可见），同时建议只有在程序最后退出的时候才调用 taos_close 关闭连接。

8.2.4. 异步查询API

同步API之外，TDengine还提供性能更高的异步调用API处理数据插入、查询操作。在软硬件环境相同的情况下，异步API处理数据插入的速度比同步API快2～4倍。异步API采用非阻塞式的调用方式，在系统真正完成某个具体数据库操作前，立即返回。调用的线程可以去处理其他工作，从而可以提升整个应用的性能。异步API在网络延迟严重的情况下，优点尤为突出。

异步API都需要应用提供相应的回调函数，回调函数参数设置如下：前两个参数都是一致的，第三个参数依不同的API而定。第一个参数param是应用调用异步API时提供给系统的，用于回调时，应用能够找回具体操作的上下文，依具体实现而定。第二个参数是SQL操作的结果集，如果为空，比如insert操作，表示没有记录返回，如果不为空，比如select操作，表示有记录返回。

异步API对于使用者的要求相对较高，用户可根据具体应用场景选择性使用。下面是两个重要的异步API：

- `void taos_query_a(TAOS *taos, const char *sql, void (*fp)(void *param, TAOS_RES *, int code), void *param);`

  异步执行SQL语句。

  - taos：调用taos_connect返回的数据库连接
  - sql：需要执行的SQL语句
  - fp：用户定义的回调函数，其第三个参数code用于指示操作是否成功，0表示成功，负数表示失败(调用taos_errstr获取失败原因)。应用在定义回调函数的时候，主要处理第二个参数TAOS_RES *，该参数是查询返回的结果集
  - param：应用提供一个用于回调的参数

- `void taos_fetch_rows_a(TAOS_RES *res, void (*fp)(void *param, TAOS_RES *, int numOfRows), void *param);`

  批量获取异步查询的结果集，只能与taos_query_a配合使用。其中：

  - res：taos_query_a回调时返回的结果集
  - fp：回调函数。其参数param是用户可定义的传递给回调函数的参数结构体；numOfRows是获取到的数据的行数（不是整个查询结果集的函数）。 在回调函数中，应用可以通过调用taos_fetch_row前向迭代获取批量记录中每一行记录。读完一块内的所有记录后，应用需要在回调函数中继续调用taos_fetch_rows_a获取下一

批记录进行处理，直到返回的记录数（numOfRows）为零（结果返回完成）或记录数为负值（查询出错）。

TDengine的异步API均采用非阻塞调用模式。应用程序可以用多线程同时打开多张表，并可以同时对每张打开的表进行查询或者插入操作。需要指出的是，**客户端应用必须确保对同一张表的操作完全串行化**，即对同一个表的插入或查询操作未完成时（未返回时），不能够执行第二个插入或查询操作。

8.2.5. 参数绑定 API

除了直接调用 `taos_query` 进行查询，TDengine 也提供了支持参数绑定的 Prepare API，与 MySQL 一样，这些 API 目前也仅支持用问号 `?` 来代表待绑定的参数。

从 2.1.1.0 和 2.1.2.0 版本开始，TDengine 大幅改进了参数绑定接口对数据写入（INSERT）场景的支持。这样在通过参数绑定接口写入数据时，就避免了 SQL 语法解析的资源消耗，从而在绝大多数情况下显著提升写入性能。此时的典型操作步骤如下：

1. 调用 `taos_stmt_init` 创建参数绑定对象；
2. 调用 `taos_stmt_prepare` 解析 INSERT 语句；
3. 如果 INSERT 语句中预留了表名但没有预留 TAGS，那么调用 `taos_stmt_set_tbname` 来设置表名；
4. 如果 INSERT 语句中既预留了表名又预留了 TAGS（例如 INSERT 语句采取的是自动建表的方式），那么调用 `taos_stmt_set_tbname_tags` 来设置表名和 TAGS 的值；
5. 调用 `taos_stmt_bind_param_batch` 以多列的方式设置 VALUES 的值，或者调用 `taos_stmt_bind_param` 以单行的方式设置 VALUES 的值；
6. 调用 `taos_stmt_add_batch` 把当前绑定的参数加入批处理；
7. 可以重复第 3 ~ 6 步，为批处理加入更多的数据行；
8. 调用 `taos_stmt_execute` 执行已经准备好的批处理指令；
9. 执行完毕，调用 `taos_stmt_close` 释放所有资源。

说明：如果 `taos_stmt_execute` 执行成功，假如不需要改变 SQL 语句的话，那么是可以复用 `taos_stmt_prepare` 的解析结果，直接进行第 3 ~ 6 步绑定新数据的。但如果执行出错，那么并不建议继续在当前的环境上下文下继续工作，而是建议释放资源，然后从 `taos_stmt_init` 步骤重新开始。

除 C/C++ 语言外，TDengine 的 Java 语言 JNI Connector 也提供参数绑定接口支持，具体请另外参见：参数绑定接口的 Java 用法。

接口相关的具体函数如下（也可以参考 apitest.c 文件中使用对应函数的方式）：

- `TAOS_STMT* taos_stmt_init(TAOS *taos)`

  创建一个 TAOS_STMT 对象用于后续调用。

- `int taos_stmt_prepare(TAOS_STMT *stmt, const char *sql, unsigned long length)`

  解析一条 SQL 语句，将解析结果和参数信息绑定到 stmt 上，如果参数 length 大于 0，将使用此参数作为 SQL 语句的长度，如等于 0，将自动判断 SQL 语句的长度。

- `int taos_stmt_bind_param(TAOS_STMT *stmt, TAOS_BIND *bind)`

  不如 `taos_stmt_bind_param_batch` 效率高，但可以支持非 INSERT 类型的 SQL 语句。
  进行参数绑定，bind 指向一个数组（代表所要绑定的一行数据），需保证此数组中的元素数量和顺序与 SQL 语句中的参数完全一致。TAOS_BIND 的使用方法与 MySQL 中的 MYSQL_BIND 一致，具体定义如下：

```
typedef struct TAOS_BIND {
  int            buffer_type;
  void *         buffer;
  unsigned long  buffer_length;  // 未实际使用
  unsigned long *length;
  int *          is_null;
  int            is_unsigned;    // 未实际使用
  int *          error;          // 未实际使用
} TAOS_BIND;
```

- `int taos_stmt_set_tbname(TAOS_STMT* stmt, const char* name)`

  （2.1.1.0 版本新增，仅支持用于替换 INSERT 语句中的参数值）
  当 SQL 语句中的表名使用了 ? 占位时，可以使用此函数绑定一个具体的表名。

- `int taos_stmt_set_tbname_tags(TAOS_STMT* stmt, const char* name, TAOS_BIND* tags)`

  （2.1.2.0 版本新增，仅支持用于替换 INSERT 语句中的参数值）
  当 SQL 语句中的表名和 TAGS 都使用了 ? 占位时，可以使用此函数绑定具体的表名和具体的 TAGS 取值。最典型的使用场景是使用了自动建表功能的 INSERT 语句（目前版本不支持指定具体的 TAGS 列）。tags 参数中的列数量需要与 SQL 语句中要求的 TAGS 数量完全一致。

- `int taos_stmt_bind_param_batch(TAOS_STMT* stmt, TAOS_MULTI_BIND* bind)`

  （2.1.1.0 版本新增，仅支持用于替换 INSERT 语句中的参数值）
  以多列的方式传递待绑定的数据，需要保证这里传递的数据列的顺序、列的数量与 SQL 语句中的 VALUES 参数完全一致。TAOS_MULTI_BIND 的具体定义如下：

```
typedef struct TAOS_MULTI_BIND {
  int          buffer_type;
  void *       buffer;
  uintptr_t    buffer_length;
  int32_t *    length;
  char *       is_null;
  int          num;              // 列的个数，即 buffer 中的参数个数
} TAOS_MULTI_BIND;
```

- `int taos_stmt_add_batch(TAOS_STMT *stmt)`

  将当前绑定的参数加入批处理中，调用此函数后，可以再次调用 `taos_stmt_bind_param` 或 `taos_stmt_bind_param_batch` 绑定新的参数。需要注意，此函数仅支持 INSERT/IMPORT 语句，如果是 SELECT 等其他 SQL 语句，将返回错误。

- `int taos_stmt_execute(TAOS_STMT *stmt)`

  执行准备好的语句。目前，一条语句只能执行一次。

- `TAOS_RES* taos_stmt_use_result(TAOS_STMT *stmt)`

  获取语句的结果集。结果集的使用方式与非参数化调用时一致，使用完成后，应对此结果集调用 `taos_free_result` 以释放资源。

- `int taos_stmt_close(TAOS_STMT *stmt)`

  执行完毕，释放所有资源。

- `char * taos_stmt_errstr(TAOS_STMT *stmt)`

（2.1.3.0 版本新增）
用于在其他 stmt API 返回错误（返回错误码或空指针）时获取错误信息。

8.2.6. 连续查询接口

TDengine提供时间驱动的实时流式计算API。可以每隔一指定的时间段，对一张或多张数据库的表(数据流)进行各种实时聚合计算操作。操作简单，仅有打开、关闭流的API。具体如下：

- `TAOS_STREAM *taos_open_stream(TAOS *taos, const char *sql, void (*fp)(void *param, TAOS_RES *, TAOS_ROW row), int64_t stime, void *param, void (*callback)(void *))`

  该API用来创建数据流，其中：

  - taos：已经建立好的数据库连接。
  - sql：SQL查询语句（仅能使用查询语句）。
  - fp：用户定义的回调函数指针，每次流式计算完成后，TDengine将查询的结果（TAOS_ROW）、查询状态（TAOS_RES）、用户定义参数（PARAM）传递给回调函数，在回调函数内，用户可以使用taos_num_fields获取结果集列数，taos_fetch_fields获取结果集每列数据的类型。
  - stime：是流式计算开始的时间。如果是“64位整数最小值”，表示从现在开始；如果不为“64位整数最小值”，表示从指定的时间开始计算（UTC时间从1970/1/1算起的毫秒数）。
  - param：是应用提供的用于回调的一个参数，回调时，提供给应用。
  - callback: 第二个回调函数，会在连续查询自动停止时被调用。

  返回值为NULL，表示创建失败；返回值不为空，表示成功。

- `void taos_close_stream (TAOS_STREAM *tstr)`

  关闭数据流，其中提供的参数是taos_open_stream的返回值。用户停止流式计算的时候，务必关闭该数据流。

8.2.7. 数据订阅接口

订阅API目前支持订阅一张或多张表，并通过定期轮询的方式不断获取写入表中的最新数据。

- `TAOS_SUB *taos_subscribe(TAOS* taos, int restart, const char* topic, const char *sql, TAOS_SUBSCRIBE_CALLBACK fp, void *param, int interval)`

  该函数负责启动订阅服务，成功时返回订阅对象，失败时返回 `NULL`，其参数为：

  - taos：已经建立好的数据库连接
  - restart：如果订阅已经存在，是重新开始，还是继续之前的订阅
  - topic：订阅的主题（即名称），此参数是订阅的唯一标识
  - sql：订阅的查询语句，此语句只能是 `select` 语句，只应查询原始数据，只能按时间正序查询数据
  - fp：收到查询结果时的回调函数（稍后介绍函数原型），只在异步调用时使用，同步调用时此参数应该传 `NULL`
  - param：调用回调函数时的附加参数，系统API将其原样传递到回调函数，不进行任何处理
  - interval：轮询周期，单位为毫秒。异步调用时，将根据此参数周期性的调用回调函数，为避免对系统性能造成影响，不建议将此参数设置的过小；同步调用时，如两次调用`taos_consume`的间隔小于此周期，API将会阻塞，直到时间间隔超过此周期。

- `typedef void (*TAOS_SUBSCRIBE_CALLBACK)(TAOS_SUB* tsub, TAOS_RES *res, void* param, int code)`

  异步模式下，回调函数的原型，其参数为：

  - tsub：订阅对象
  - res：查询结果集，注意结果集中可能没有记录
  - param：调用 `taos_subscribe`时客户程序提供的附加参数
  - code：错误码

  注意：在这个回调函数里不可以做耗时过长的处理，尤其是对于返回的结果集中数据较多的情况，否则有可能导致客户端阻塞等异常状态。如果必须进行复杂计算，则建议在另外的线程中进行处理。

- `TAOS_RES *taos_consume(TAOS_SUB *tsub)`

  同步模式下，该函数用来获取订阅的结果。 用户应用程序将其置于一个循环之中。 如两次调用`taos_consume`的间隔小于订阅的轮询周期，API将会阻塞，直到时间间隔超过此周期。 如果数据库有新记录到达，该API将返回该最新的记录，否则返回一个没有记录的空结果集。 如果返回值为`NULL`，说明系统出错。 异步模式下，用户程序不应调用此API。

  注意：在调用 `taos_consume()` 之后，用户应用应确保尽快调用 `taos_fetch_row()` 或 `taos_fetch_block()` 来处理订阅结果，否则服务端会持续缓存查询结果数据等待客户端读取，极端情况下会导致服务端内存消耗殆尽，影响服务稳定性。

- `void taos_unsubscribe(TAOS_SUB *tsub, int keepProgress)`

  取消订阅。 如参数 `keepProgress` 不为0，API会保留订阅的进度信息，后续调用 `taos_subscribe` 时可以基于此进度继续；否则将删除进度信息，后续只能重新开始读取数据。

8.3. Java Connector

8.3.1. 安装

Java连接器支持的系统有： Linux 64/Windows x64/Windows x86。

安装前准备：

- 已安装TDengine服务器端
- 已安装好TDengine应用驱动，具体请参照 安装连接器驱动步骤 章节

TDengine 为了方便 Java 应用使用，提供了遵循 JDBC 标准(3.0)API 规范的 `taos-jdbcdriver` 实现。目前可以通过 Sonatype Repository 搜索并下载。

由于 TDengine 的应用驱动是使用C语言开发的，使用 taos-jdbcdriver 驱动包时需要依赖系统对应的本地函数库。

- libtaos.so 在 Linux 系统中成功安装 TDengine 后，依赖的本地函数库 libtaos.so 文件会被自动拷贝至 /usr/lib/libtaos.so，该目录包含在 Linux 自动扫描路径上，无需单独指定。
- taos.dll 在 Windows 系统中安装完客户端之后，驱动包依赖的 taos.dll 文件会自动拷贝到系统默认搜索路径 C:/Windows/System32 下，同样无需要单独指定。

注意：在 Windows 环境开发时需要安装 TDengine 对应的 windows 客户端，Linux 服务器安装完 TDengine 之后默认已安装 client，也可以单独安装 Linux 客户端 连接远程 TDengine Server。

8.3.1.1. 如何获取 TAOS-JDBCDriver

maven仓库

目前 taos-jdbcdriver 已经发布到 Sonatype Repository 仓库，且各大仓库都已同步。

- sonatype
- mvnrepository
- maven.aliyun

maven 项目中使用如下 pom.xml 配置即可：

```
<dependency>
 <groupId>com.taosdata.jdbc</groupId>
 <artifactId>taos-jdbcdriver</artifactId>
 <version>2.0.18</version>
</dependency>
```

源码编译打包

下载 TDengine 源码之后，进入 taos-jdbcdriver 源码目录 `src/connector/jdbc` 执行 `mvn clean package -Dmaven.test.skip=true` 即可生成相应 jar 包。

8.3.1.2. 示例程序

示例程序源码位于install_directory/examples/JDBC，有如下目录：

JDBCDemo　　JDBC示例源程序

JDBCConnectorChecker　JDBC安装校验源程序及jar包

Springbootdemo　　springboot示例源程序

SpringJdbcTemplate　SpringJDBC模板

8.3.1.3. 安装验证

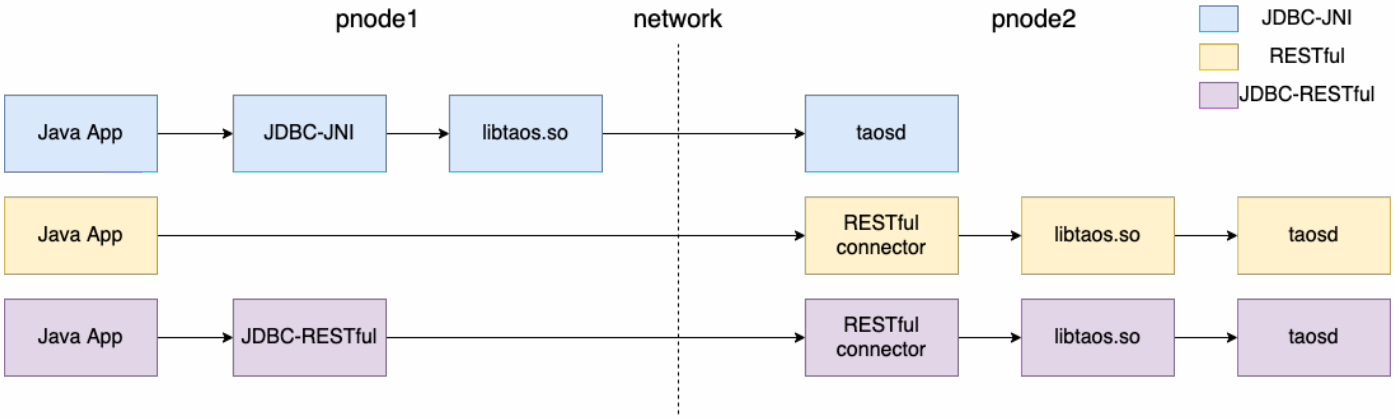
运行如下指令：

```
cd {install_directory}/examples/JDBC/JDBCConnectorChecker
java -jar JDBCConnectorChecker.jar -host <fqdn>
```

验证通过将打印出成功信息。

8.3.2. Java连接器的使用

`taos-jdbcdriver` 的实现包括 2 种形式： JDBC-JNI 和 JDBC-RESTful（taos-jdbcdriver-2.0.18 开始支持 JDBC-RESTful）。 JDBC-JNI 通过调用客户端 libtaos.so（或 taos.dll ）的本地方法实现， JDBC-RESTful 则在内部封装了 RESTful 接口实现。



上图显示了 3 种 Java 应用使用连接器访问 TDengine 的方式：

- JDBC-JNI：Java 应用在物理节点1（pnode1）上使用 JDBC-JNI 的 API ，直接调用客户端 API（libtaos.so 或 taos.dll）将写入和查询请求发送到位于物理节点2（pnode2）上的 taosd 实例。
- RESTful：应用将 SQL 发送给位于物理节点2（pnode2）上的 RESTful 连接器，再调用客户端 API（libtaos.so）。
- JDBC-RESTful：Java 应用通过 JDBC-RESTful 的 API ，将 SQL 封装成一个 RESTful 请求，发送给物理节点2的 RESTful 连接器。

TDengine 的 JDBC 驱动实现尽可能与关系型数据库驱动保持一致，但时序空间数据库与关系对象型数据库服务的对象和技术特征存在差异，导致 `taos-jdbcdriver` 与传统的 JDBC driver 也存在一定差异。在使用时需要注意以下几点：

- TDengine 目前不支持针对单条数据记录的删除操作。
- 目前不支持事务操作。
- 目前不支持嵌套查询（nested query）。
- 对每个 Connection 的实例，至多只能有一个打开的 ResultSet 实例；如果在 ResultSet 还没关闭的情况下执行了新的查询，taos-jdbcdriver 会自动关闭上一个 ResultSet。

8.3.2.1. JDBC-JNI和JDBC-RESTful的对比

对比项	JDBC-JNI	JDBC-RESTful
支持的操作系统	linux、windows	全平台
是否需要安装 client	需要	不需要
server 升级后是否需要升级 client	需要	不需要
写入性能	JDBC-RESTful 是 JDBC-JNI 的 50% ~ 90%	
查询性能	JDBC-RESTful 与 JDBC-JNI 没有差别	

注意：与 JNI 方式不同，RESTful 接口是无状态的，因此 `USE db_name` 指令没有效果，RESTful 下所有对表名、超级表名的引用都需要指定数据库名前缀。

8.3.2.2. TAOS-JDBCDriver 版本以及支持的 TDengine 版本和 JDK 版本

taos-jdbcdriver 版本	TDengine 版本	JDK 版本
2.0.31	2.1.3.0 及以上	1.8.x
2.0.22 - 2.0.30	2.0.18.0 - 2.1.2.x	1.8.x
2.0.12 - 2.0.21	2.0.8.0 - 2.0.17.x	1.8.x
2.0.4 - 2.0.11	2.0.0.0 - 2.0.7.x	1.8.x
1.0.3	1.6.1.x 及以上	1.8.x
1.0.2	1.6.1.x 及以上	1.8.x
1.0.1	1.6.1.x 及以上	1.8.x

8.3.2.3. TDengine DataType 和 Java DataType

TDengine 目前支持时间戳、数字、字符、布尔类型，与 Java 对应类型转换如下：

TDengine DataType	Java DataType
TIMESTAMP	java.sql.Timestamp
INT	java.lang.Integer
BIGINT	java.lang.Long
FLOAT	java.lang.Float
DOUBLE	java.lang.Double
SMALLINT	java.lang.Short
TINYINT	java.lang.Byte
BOOL	java.lang.Boolean
BINARY	byte array
NCHAR	java.lang.String

8.3.2.4. 获取连接

8.3.2.4.1. 指定URL获取连接

通过指定URL获取连接，如下所示：

```
Class.forName("com.taosdata.jdbc.rs.RestfulDriver");
String jdbcUrl = "jdbc:TAOS-RS://taosdemo.com:6041/test?
user=root&password=taosdata";
Connection conn = DriverManager.getConnection(jdbcUrl);
```

以上示例，使用 **JDBC-RESTful** 的 driver，建立了到 hostname 为 taosdemo.com，端口为 6041，数据库名为 test 的连接。这个 URL 中指定用户名（user）为 root，密码（password）为 taosdata。

使用 JDBC-RESTful 接口，不需要依赖本地函数库。与 JDBC-JNI 相比，仅需要：

1. driverClass 指定为“com.taosdata.jdbc.rs.RestfulDriver”；
2. jdbcUrl 以“jdbc:TAOS-RS://”开头；
3. 使用 6041 作为连接端口。

如果希望获得更好的写入和查询性能，Java 应用可以使用 **JDBC-JNI** 的driver，如下所示：

```
Class.forName("com.taosdata.jdbc.TSDBDriver");
String jdbcUrl = "jdbc:TAOS://taosdemo.com:6030/test?user=root&password=taosdata";
Connection conn = DriverManager.getConnection(jdbcUrl);
```

以上示例，使用了 JDBC-JNI 的 driver，建立了到 hostname 为 taosdemo.com，端口为 6030（TDengine 的默认端口），数据库名为 test 的连接。这个 URL 中指定用户名（user）为 root，密码（password）为 taosdata。

注意：使用 JDBC-JNI 的 driver，taos-jdbcdriver 驱动包时需要依赖系统对应的本地函数库。

- libtaos.so
  在 Linux 系统中成功安装 TDengine 后，依赖的本地函数库 libtaos.so 文件会被自动拷贝至 /usr/lib/libtaos.so，该目录包含在 Linux 自动扫描路径上，无需单独指定。
- taos.dll
  在 Windows 系统中安装完客户端之后，驱动包依赖的 taos.dll 文件会自动拷贝到系统默认搜索路径 C:/Windows/System32 下，同样无需要单独指定。

> 在 Windows 环境开发时需要安装 TDengine 对应的 [windows 客户端][14]，Linux 服务器安装完 TDengine 之后默认已安装 client，也可以单独安装 [Linux 客户端][15] 连接远程 TDengine Server。

JDBC-JNI 的使用请参见视频教程。

TDengine 的 JDBC URL 规范格式为：
`jdbc:[TAOS|TAOS-RS]://[host_name]:[port]/[database_name]?[user={user}|&password={password}|&charset={charset}|&cfgdir={config_dir}|&locale={locale}|&timezone={timezone}]`

url中的配置参数如下：

- user：登录 TDengine 用户名，默认值 root。
- password：用户登录密码，默认值 taosdata。
- cfgdir：客户端配置文件目录路径，Linux OS 上默认值 /etc/taos ，Windows OS 上默认值 C:/TDengine/cfg。

- charset：客户端使用的字符集，默认值为系统字符集。
- locale：客户端语言环境，默认值系统当前 locale。
- timezone：客户端使用的时区，默认值为系统当前时区。

8.3.2.4.2. 指定URL和Properties获取连接

除了通过指定的 URL 获取连接，还可以使用 Properties 指定建立连接时的参数，如下所示：

```
public Connection getConn() throws Exception{
  Class.forName("com.taosdata.jdbc.TSDBDriver");
  // Class.forName("com.taosdata.jdbc.rs.RestfulDriver");
  String jdbcUrl = "jdbc:TAOS://taosdemo.com:6030/test?
user=root&password=taosdata";
  // String jdbcUrl = "jdbc:TAOS-RS://taosdemo.com:6041/test?
user=root&password=taosdata";
  Properties connProps = new Properties();
  connProps.setProperty(TSDBDriver.PROPERTY_KEY_CHARSET, "UTF-8");
  connProps.setProperty(TSDBDriver.PROPERTY_KEY_LOCALE, "en_US.UTF-8");
  connProps.setProperty(TSDBDriver.PROPERTY_KEY_TIME_ZONE, "UTC-8");
  Connection conn = DriverManager.getConnection(jdbcUrl, connProps);
  return conn;
}
```

以上示例，建立一个到 hostname 为 taosdemo.com，端口为 6030，数据库名为 test 的连接。注释为使用 JDBC-RESTful 时的方法。这个连接在 url 中指定了用户名(user)为 root，密码（password）为 taosdata，并在 connProps 中指定了使用的字符集、语言环境、时区等信息。

properties 中的配置参数如下：

- TSDBDriver.PROPERTY_KEY_USER：登录 TDengine 用户名，默认值 root。
- TSDBDriver.PROPERTY_KEY_PASSWORD：用户登录密码，默认值 taosdata。
- TSDBDriver.PROPERTY_KEY_CONFIG_DIR：客户端配置文件目录路径，Linux OS 上默认值 /etc/taos ，Windows OS 上默认值 C:/TDengine/cfg。
- TSDBDriver.PROPERTY_KEY_CHARSET：客户端使用的字符集，默认值为系统字符集。
- TSDBDriver.PROPERTY_KEY_LOCALE：客户端语言环境，默认值系统当前 locale。
- TSDBDriver.PROPERTY_KEY_TIME_ZONE：客户端使用的时区，默认值为系统当前时区。

8.3.2.4.3. 使用客户端配置文件建立连接

当使用 JDBC-JNI 连接 TDengine 集群时，可以使用客户端配置文件，在客户端配置文件中指定集群的 firstEp、secondEp参数。如下所示：

1. 在 Java 应用中不指定 hostname 和 port

```
public Connection getConn() throws Exception{
  Class.forName("com.taosdata.jdbc.TSDBDriver");
  String jdbcUrl = "jdbc:TAOS://:/test?user=root&password=taosdata";
  Properties connProps = new Properties();
  connProps.setProperty(TSDBDriver.PROPERTY_KEY_CHARSET, "UTF-8");
  connProps.setProperty(TSDBDriver.PROPERTY_KEY_LOCALE, "en_US.UTF-8");
  connProps.setProperty(TSDBDriver.PROPERTY_KEY_TIME_ZONE, "UTC-8");
  Connection conn = DriverManager.getConnection(jdbcUrl, connProps);
  return conn;
}
```

2. 在配置文件中指定 firstEp 和 secondEp

```
## first fully qualified domain name (FQDN) for TDengine system
firstEp               cluster_node1:6030

## second fully qualified domain name (FQDN) for TDengine system, for cluster only
secondEp              cluster_node2:6030

## default system charset
## charset               UTF-8

## system locale
## locale                en_US.UTF-8
```

以上示例，jdbc 会使用客户端的配置文件，建立到 hostname 为 cluster_node1、端口为 6030、数据库名为 test 的连接。当集群中 firstEp 节点失效时，JDBC 会尝试使用 secondEp 连接集群。
TDengine 中，只要保证 firstEp 和 secondEp 中一个节点有效，就可以正常建立到集群的连接。

> 注意：这里的配置文件指的是调用 JDBC Connector 的应用程序所在机器上的配置文件，Linux OS 上默认值 /etc/taos/taos.cfg ，Windows OS 上默认值 C://TDengine/cfg/taos.cfg。

8.3.2.4.4. 配置参数的优先级

通过以上 3 种方式获取连接，如果配置参数在 url、Properties、客户端配置文件中有重复，则参数的优先级由高到低分别如下：

1. JDBC URL 参数，如上所述，可以在 JDBC URL 的参数中指定。
2. Properties connProps
3. 客户端配置文件 taos.cfg

例如：在 url 中指定了 password 为 taosdata，在 Properties 中指定了 password 为 taosdemo，那么，JDBC 会使用 url 中的 password 建立连接。

> 更多详细配置请参考客户端配置

8.3.2.5. 创建数据库和表

```
Statement stmt = conn.createStatement();

// create database
stmt.executeUpdate("create database if not exists db");

// use database
stmt.executeUpdate("use db");

// create table
stmt.executeUpdate("create table if not exists tb (ts timestamp, temperature int, humidity float)");
```

> 注意：如果不使用 use db 指定数据库，则后续对表的操作都需要增加数据库名称作为前缀，如 db.tb。

8.3.2.6. 插入数据

```
// insert data
int affectedRows = stmt.executeUpdate("insert into tb values(now, 23, 10.3) (now + 1s, 20, 9.3)");

System.out.println("insert " + affectedRows + " rows.");
```

> now 为系统内部函数，默认为客户端所在计算机当前时间。
> now + 1s 代表客户端当前时间往后加 1 秒，数字后面代表时间单位：a(毫秒)，s(秒)，m(分)，h(小时)，d(天)，w(周)，n(月)，y(年)。

8.3.2.7. 查询数据

```
// query data
ResultSet resultSet = stmt.executeQuery("select * from tb");

Timestamp ts = null;
int temperature = 0;
float humidity = 0;
while(resultSet.next()){

    ts = resultSet.getTimestamp(1);
    temperature = resultSet.getInt(2);
    humidity = resultSet.getFloat("humidity");

    System.out.printf("%s, %d, %s\n", ts, temperature, humidity);
}
```

> 查询和操作关系型数据库一致，使用下标获取返回字段内容时从 1 开始，建议使用字段名称获取。

8.3.2.8. 处理异常

在报错后，通过SQLException可以获取到错误的信息和错误码：

```
try (Statement statement = connection.createStatement()) {
    // executeQuery
    ResultSet resultSet = statement.executeQuery(sql);
    // print result
    printResult(resultSet);
} catch (SQLException e) {
    System.out.println("ERROR Message: " + e.getMessage());
    System.out.println("ERROR Code: " + e.getErrorCode());
    e.printStackTrace();
}
```

JDBC连接器可能报错的错误码包括3种：JDBC driver本身的报错（错误码在0x2301到0x2350之间），JNI方法的报错（错误码在0x2351到0x2400之间），TDengine其他功能模块的报错。
具体的错误码请参考：

- https://github.com/taosdata/TDengine/blob/develop/src/connector/jdbc/src/main/java/com/taosdata/jdbc/TSDBErrorNumbers.java
- https://github.com/taosdata/TDengine/blob/develop/src/inc/taoserror.h

8.3.2.9. 通过参数绑定写入数据

从 2.1.2.0 版本开始，TDengine 的 **JDBC-JNI** 实现大幅改进了参数绑定方式对数据写入（INSERT）场景的支持。采用这种方式写入数据时，能避免 SQL 语法解析的资源消耗，从而在很多情况下显著提升写入性能。（注意：**JDBC-RESTful** 实现并不提供参数绑定这种使用方式。）

```
Statement stmt = conn.createStatement();
Random r = new Random();

// INSERT 语句中，VALUES 部分允许指定具体的数据列；如果采取自动建表，则 TAGS 部分需要设定全部 TAGS 列的参数值：
TSDBPreparedStatement s = (TSDBPreparedStatement) conn.prepareStatement("insert into ? using weather_test tags (?, ?) (ts, c1, c2) values(?, ?, ?)");

// 设定数据表名：
s.setTableName("w1");
// 设定 TAGS 取值：
s.setTagInt(0, r.nextInt(10));
s.setTagString(1, "Beijing");

int numOfRows = 10;

// VALUES 部分以逐列的方式进行设置：
ArrayList<Long> ts = new ArrayList<>();
for (int i = 0; i < numOfRows; i++){
    ts.add(System.currentTimeMillis() + i);
}
s.setTimestamp(0, ts);
```

```
21
22 ArrayList<Integer> s1 = new ArrayList<>();
23 for (int i = 0; i < numOfRows; i++){
24     s1.add(r.nextInt(100));
25 }
26 s.setInt(1, s1);
27
28 ArrayList<String> s2 = new ArrayList<>();
29 for (int i = 0; i < numOfRows; i++){
30     s2.add("test" + r.nextInt(100));
31 }
32 s.setString(2, s2, 10);
33
34 // AddBatch 之后，缓存并未清空。为避免混乱，并不推荐在 ExecuteBatch 之前再次绑定新一批的数
   据：
35 s.columnDataAddBatch();
36 // 执行绑定数据后的语句：
37 s.columnDataExecuteBatch();
38 // 执行语句后清空缓存。在清空之后，可以复用当前的对象，绑定新的一批数据（可以是新表名、新 TAGS
   值、新 VALUES 值）：
39 s.columnDataClearBatch();
40 // 执行完毕，释放资源：
41 s.columnDataCloseBatch();
```

用于设定 TAGS 取值的方法总共有：

```
1  public void setTagNull(int index, int type)
2  public void setTagBoolean(int index, boolean value)
3  public void setTagInt(int index, int value)
4  public void setTagByte(int index, byte value)
5  public void setTagShort(int index, short value)
6  public void setTagLong(int index, long value)
7  public void setTagTimestamp(int index, long value)
8  public void setTagFloat(int index, float value)
9  public void setTagDouble(int index, double value)
10 public void setTagString(int index, String value)
11 public void setTagNString(int index, String value)
```

用于设定 VALUES 数据列的取值的方法总共有：

```
public void setInt(int columnIndex, ArrayList<Integer> list) throws SQLException
public void setFloat(int columnIndex, ArrayList<Float> list) throws SQLException
public void setTimestamp(int columnIndex, ArrayList<Long> list) throws SQLException
public void setLong(int columnIndex, ArrayList<Long> list) throws SQLException
public void setDouble(int columnIndex, ArrayList<Double> list) throws SQLException
public void setBoolean(int columnIndex, ArrayList<Boolean> list) throws SQLException
public void setByte(int columnIndex, ArrayList<Byte> list) throws SQLException
public void setShort(int columnIndex, ArrayList<Short> list) throws SQLException
public void setString(int columnIndex, ArrayList<String> list, int size) throws SQLException
public void setNString(int columnIndex, ArrayList<String> list, int size) throws SQLException
```

其中 setString 和 setNString 都要求用户在 size 参数里声明表定义中对应列的列宽。

8.3.2.10. 订阅

8.3.2.10.1. 创建

```
TSDBSubscribe sub = ((TSDBConnection)conn).subscribe("topic", "select * from meters", false);
```

subscribe 方法的三个参数含义如下：

- topic：订阅的主题（即名称），此参数是订阅的唯一标识
- sql：订阅的查询语句，此语句只能是 select 语句，只应查询原始数据，只能按时间正序查询数据
- restart：如果订阅已经存在，是重新开始，还是继续之前的订阅

如上面的例子将使用 SQL 语句 select * from meters 创建一个名为 topic 的订阅，如果这个订阅已经存在，将继续之前的查询进度，而不是从头开始消费所有的数据。

8.3.2.10.2. 消费数据

```
int total = 0;
while(true) {
    TSDBResultSet rs = sub.consume();
    int count = 0;
    while(rs.next()) {
        count++;
    }
    total += count;
    System.out.printf("%d rows consumed, total %d\n", count, total);
    Thread.sleep(1000);
}
```

consume 方法返回一个结果集，其中包含从上次 consume 到目前为止的所有新数据。请务必按需选择合理的调用 consume 的频率（如例子中的 Thread.sleep(1000)），否则会给服务端造成不必要的压力。

8.3.2.10.3. 关闭订阅

```
sub.close(true);
```

close 方法关闭一个订阅。如果其参数为 true 表示保留订阅进度信息，后续可以创建同名订阅继续消费数据；如为 false 则不保留订阅进度。

8.3.2.11. 关闭资源

```
resultSet.close();
stmt.close();
conn.close();
```

> 注意务必要将 connection 进行关闭，否则会出现连接泄露。

8.3.3. 与连接池使用

HikariCP

- 引入相应 HikariCP maven 依赖：

```
<dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>HikariCP</artifactId>
    <version>3.4.1</version>
</dependency>
```

- 使用示例如下：

```
 public static void main(String[] args) throws SQLException {
    HikariConfig config = new HikariConfig();
    // jdbc properties
    config.setJdbcUrl("jdbc:TAOS://127.0.0.1:6030/log");
    config.setUsername("root");
    config.setPassword("taosdata");
    // connection pool configurations
    config.setMinimumIdle(10);           //minimum number of idle connection
    config.setMaximumPoolSize(10);      //maximum number of connection in the pool
    config.setConnectionTimeout(30000); //maximum wait milliseconds for get connection from pool
    config.setMaxLifetime(0);       // maximum life time for each connection
    config.setIdleTimeout(0);       // max idle time for recycle idle connection
    config.setConnectionTestQuery("select server_status()"); //validation query

    HikariDataSource ds = new HikariDataSource(config); //create datasource

    Connection  connection = ds.getConnection(); // get connection
```

```
    Statement statement = connection.createStatement(); // get statement

    //query or insert
    // ...

    connection.close(); // put back to conneciton pool
}
```

> 通过 HikariDataSource.getConnection() 获取连接后，使用完成后需要调用 close() 方法，实际上它并不会关闭连接，只是放回连接池中。
> 更多 HikariCP 使用问题请查看官方说明。

Druid

- 引入相应 Druid maven 依赖：

```
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.1.20</version>
</dependency>
```

- 使用示例如下：

```
public static void main(String[] args) throws Exception {

    DruidDataSource dataSource = new DruidDataSource();
    // jdbc properties
    dataSource.setDriverClassName("com.taosdata.jdbc.TSDBDriver");
    dataSource.setUrl(url);
    dataSource.setUsername("root");
    dataSource.setPassword("taosdata");
    // pool configurations
    dataSource.setInitialSize(10);
    dataSource.setMinIdle(10);
    dataSource.setMaxActive(10);
    dataSource.setMaxWait(30000);
    dataSource.setValidationQuery("select server_status()");

    Connection  connection = dataSource.getConnection(); // get connection
    Statement statement = connection.createStatement(); // get statement
    //query or insert
    // ...

    connection.close(); // put back to conneciton pool
}
```

> 更多 druid 使用问题请查看官方说明。

注意事项：

- TDengine `v1.6.4.1` 版本开始提供了一个专门用于心跳检测的函数 `select server_status()`，所以在使用连接池时推荐使用 `select server_status()` 进行 Validation Query。

如下所示，`select server_status()` 执行成功会返回 `1`。

```
taos> select server_status();
server_status()|
================
1              |
Query OK, 1 row(s) in set (0.000141s)
```

8.3.4. 在框架中使用

- Spring JdbcTemplate 中使用 taos-jdbcdriver，可参考 SpringJdbcTemplate
- Springboot + Mybatis 中使用，可参考 springbootdemo

8.3.5. 常见问题

- java.lang.UnsatisfiedLinkError: no taos in java.library.path

  原因：程序没有找到依赖的本地函数库 taos。

  解决方法：Windows 下可以将 C:\TDengine\driver\taos.dll 拷贝到 C:\Windows\System32\ 目录下，Linux 下将建立如下软链 `ln -s /usr/local/taos/driver/libtaos.so.x.x.x.x /usr/lib/libtaos.so` 即可。

- java.lang.UnsatisfiedLinkError: taos.dll Can't load AMD 64 bit on a IA 32-bit platform

  原因：目前 TDengine 只支持 64 位 JDK。

  解决方法：重新安装 64 位 JDK。

- 其它问题请参考 Issues

8.4. Python Connector

Python连接器的使用参见视频教程

安装：参见下面具体步骤

示例程序：位于install_directory/examples/python

8.4.1. 安装

Python连接器支持的系统有：Linux 64/Windows x64

安装前准备：

- 已安装好TDengine应用驱动，请参考安装连接器驱动步骤
- 已安装python 2.7 or >= 3.4
- 已安装pip

8.4.2. Python连接器安装

Linux

用户可以在源代码的src/connector/python（或者tar.gz的/connector/python）文件夹下找到connector安装包。用户可以通过pip命令安装：

```
pip install src/connector/python/
```

或

```
pip3 install src/connector/python/
```

Windows

在已安装Windows TDengine 客户端的情况下， 将文件"C:\TDengine\driver\taos.dll" 拷贝到"C:\Windows\system32" 目录下, 然后进入Windows *cmd* 命令行界面

```
cd C:\TDengine\connector\python
python -m pip install .
```

- 如果机器上没有pip命令，用户可将src/connector/python下的taos文件夹拷贝到应用程序的目录使用。对于windows 客户端，安装TDengine windows 客户端后，将C:\TDengine\driver\taos.dll拷贝到C:\windows\system32目录下即可。

8.4.3. 示例程序

示例程序源码位于install_directory/examples/Python，有：
read_example.py　　Python示例源程序

用户可以参考read_example.py这个程序来设计用户自己的写入、查询程序。

在安装了对应的应用驱动后，通过import taos引入taos类。主要步骤如下：

- 通过taos.connect获取TDengineConnection对象，这个对象可以一个程序只申请一个，在多线程中共享。
- 通过TDengineConnection对象的 .cursor()方法获取一个新的游标对象，这个游标对象必须保证每个线程独享。
- 通过游标对象的execute()方法，执行写入或查询的SQL语句。

- 如果执行的是写入语句，execute返回的是成功写入的行数信息affected rows。
- 如果执行的是查询语句，则execute执行成功后，需要通过fetchall方法去拉取结果集。 具体方法可以参考示例代码。

8.4.4. 安装验证

运行如下指令：

```
cd {install_directory}/examples/python/PYTHONConnectorChecker/`
python3 PythonChecker.py –host <fqdn>
```

验证通过将打印出成功信息。

8.4.5. Python连接器的使用

8.4.5.1. 代码示例

- 导入TDengine客户端模块

```
import taos
```

- 获取连接并获取游标对象

```
conn = taos.connect(host="127.0.0.1", user="root", password="taosdata",
config="/etc/taos")
c1 = conn.cursor()
```

- *host* 是TDengine 服务端所有IP, *config* 为客户端配置文件所在目录
- 写入数据

```
import datetime

# 创建数据库
c1.execute('create database db')
c1.execute('use db')
# 建表
c1.execute('create table tb (ts timestamp, temperature int, humidity float)')
# 插入数据
start_time = datetime.datetime(2019, 11, 1)
affected_rows = c1.execute('insert into tb values (\'%s\', 0, 0.0)' %start_time)
# 批量插入数据
time_interval = datetime.timedelta(seconds=60)
sqlcmd = ['insert into tb values']
for irow in range(1,11):
```

```
  start_time += time_interval
  sqlcmd.append('(\'%s\', %d, %f)' %(start_time, irow, irow*1.2))
affected_rows = c1.execute(' '.join(sqlcmd))
```

- 查询数据

```
c1.execute('select * from tb')
# 拉取查询结果
data = c1.fetchall()
# 返回的结果是一个列表，每一行构成列表的一个元素
numOfRows = c1.rowcount
numOfCols = len(c1.description)
for irow in range(numOfRows):
  print("Row%d: ts=%s, temperature=%d, humidity=%f" %(irow, data[irow][0], data[irow][1],data[irow][2]))

# 直接使用cursor 循环拉取查询结果
c1.execute('select * from tb')
for data in c1:
  print("ts=%s, temperature=%d, humidity=%f" %(data[0], data[1],data[2]))
```

- 创建订阅

```
# 创建一个主题为 'test' 消费周期为1000毫秒的订阅
# 第一个参数为 True 表示重新开始订阅，如为 False 且之前创建过主题为 'test' 的订阅，则表示继续消费此订阅的数据，而不是重新开始消费所有数据
sub = conn.subscribe(True, "test", "select * from tb;", 1000)
```

- 消费订阅的数据

```
data = sub.consume()
for d in data:
    print(d)
```

- 取消订阅

```
sub.close()
```

- 关闭连接

```
c1.close()
conn.close()
```

8.4.5.2. 关于纳秒 (nanosecond) 在 Python 连接器中的说明

由于目前 Python 对 nanosecond 支持的不完善(参见链接 1. 2.)，目前的实现方式是在 nanosecond 精度时返回整数，而不是 ms 和 us 返回的 datetime 类型，应用开发者需要自行处理，建议使用 pandas 的 to_datetime()。未来如果 Python 正式完整支持了纳秒，涛思数据可能会修改相关接口。

1. https://stackoverflow.com/questions/10611328/parsing-datetime-strings-containing-nanoseconds
2. https://www.python.org/dev/peps/pep-0564/

8.4.5.3. 帮助信息

用户可通过python的帮助信息直接查看模块的使用信息，或者参考tests/examples/python中的示例程序。以下为部分常用类和方法：

- *TDengineConnection* 类

  参考python中help(taos.TDengineConnection)。
  这个类对应客户端和TDengine建立的一个连接。在客户端多线程的场景下，推荐每个线程申请一个独立的连接实例，而不建议多线程共享一个连接。

- *TDengineCursor* 类

  参考python中help(taos.TDengineCursor)。
  这个类对应客户端进行的写入、查询操作。在客户端多线程的场景下，这个游标实例必须保持线程独享，不能跨线程共享使用，否则会导致返回结果出现错误。

- *connect* 方法

  用于生成taos.TDengineConnection的实例。

8.4.6. Python客户端使用示例代码

在tests/examples/python中，我们提供了一个示例Python程序read_example.py，可以参考这个程序来设计用户自己的写入、查询程序。在安装了对应的客户端后，通过import taos引入taos类。主要步骤如下

- 通过taos.connect获取TDengineConnection对象，这个对象可以一个程序只申请一个，在多线程中共享。
- 通过TDengineConnection对象的 .cursor()方法获取一个新的游标对象，这个游标对象必须保证每个线程独享。
- 通过游标对象的execute()方法，执行写入或查询的SQL语句。
- 如果执行的是写入语句，execute返回的是成功写入的行数信息affected rows。
- 如果执行的是查询语句，则execute执行成功后，需要通过fetchall方法去拉取结果集。
  具体方法可以参考示例代码。

8.5. RESTful Connector

为支持各种不同类型平台的开发，TDengine 提供符合 REST 设计标准的 API，即 RESTful API。为最大程度降低学习成本，不同于其他数据库 RESTful API 的设计方法，TDengine 直接通过 HTTP POST 请求 BODY 中包含的 SQL 语句来操作数据库，仅需要一个 URL。RESTful 连接器的使用参见视频教程。

注意：与标准连接器的一个区别是，RESTful 接口是无状态的，因此 `USE db_name` 指令没有效果，所有对表名、超级表名的引用都需要指定数据库名前缀。

8.5.1. 安装

RESTful接口不依赖于任何TDengine的库，因此客户端不需要安装任何TDengine的库，只要客户端的开发语言支持HTTP协议即可。

8.5.2. 验证

在已经安装TDengine服务器端的情况下，可以按照如下方式进行验证。

下面以Ubuntu环境中使用curl工具（确认已经安装）来验证RESTful接口的正常。

下面示例是列出所有的数据库，请把h1.taosdata.com和6041（缺省值）替换为实际运行的TDengine服务fqdn和端口号：

```
curl -H 'Authorization: Basic cm9vdDp0YW9zZGF0YQ==' -d 'show databases;'
h1.taosdata.com:6041/rest/sql
```

返回值结果如下表示验证通过：

```
{
  "status": "succ",
  "head":
["name","created_time","ntables","vgroups","replica","quorum","days","keep1,keep2,keep(D)","cache(MB)","blocks","minrows","maxrows","wallevel","fsync","comp","precision","status"],
  "data": [
    ["log","2020-09-02 17:23:00.039",4,1,1,1,10,"30,30,30",1,3,100,4096,1,3000,2,"us","ready"],
    ],
  "rows": 1
}
```

8.5.3. RESTful连接器的使用

8.5.3.1. HTTP请求格式

```
http://<fqdn>:<port>/rest/sql
```

参数说明：

- fqnd: 集群中的任一台主机FQDN或IP地址
- port: 配置文件中httpPort配置项，缺省为6041

例如：http://h1.taos.com:6041/rest/sql 是指向地址为h1.taos.com:6041的url。

HTTP请求的Header里需带有身份认证信息，TDengine支持Basic认证与自定义认证两种机制，后续版本将提供标准安全的数字签名机制来做身份验证。

- 自定义身份认证信息如下所示（稍后介绍）

```
Authorization: Taosd <TOKEN>
```

- Basic身份认证信息如下所示

```
Authorization: Basic <TOKEN>
```

HTTP请求的BODY里就是一个完整的SQL语句，SQL语句中的数据表应提供数据库前缀，例如<db-name>.<tb-name>。如果表名不带数据库前缀，系统会返回错误。因为HTTP模块只是一个简单的转发，没有当前DB的概念。

使用curl通过自定义身份认证方式来发起一个HTTP Request，语法如下：

```
curl -H 'Authorization: Basic <TOKEN>' -d '<SQL>' <ip>:<PORT>/rest/sql
```

或者

```
curl -u username:password -d '<SQL>' <ip>:<PORT>/rest/sql
```

其中，TOKEN为{username}:{password}经过Base64编码之后的字符串，例如root:taosdata编码后为cm9vdDp0YW9zZGF0YQ==

8.5.4. HTTP返回格式

返回值为JSON格式，如下：

```
{
    "status": "succ",
    "head": ["ts","current", …],
    "column_meta": [["ts",9,8],["current",6,4], …],
    "data": [
        ["2018-10-03 14:38:05.000", 10.3, …],
        ["2018-10-03 14:38:15.000", 12.6, …]
    ],
    "rows": 2
}
```

说明：

- status: 告知操作结果是成功还是失败。
- head: 表的定义，如果不返回结果集，则仅有一列“affected_rows”。（从 2.0.17.0 版本开始，建议不要依赖 head 返回值来判断数据列类型，而推荐使用 column_meta。在未来版本中，有可能会从返回值中去掉 head 这一项。）
- column_meta: 从 2.0.17.0 版本开始，返回值中增加这一项来说明 data 里每一列的数据类型。具体每个列会用三个值来说明，分别为：列名、列类型、类型长度。例如`["current",6,4]`表示列名为“current”；列类型为 6，也即 float 类型；类型长度为 4，也即对应 4 个字节表示的 float。如果列类型为 binary 或 nchar，则类型长度表示该列最多可以保存的内容长度，而不是本次返回值中的具体数据长度。当列类型是 nchar 的时候，其类型长度表示可以保存的 unicode 字符数量，而不是 bytes。
- data: 具体返回的数据，一行一行的呈现，如果不返回结果集，那么就仅有[[affected_rows]]。data 中每一行的数据列顺序，与 column_meta 中描述数据列的顺序完全一致。
- rows: 表明总共多少行数据。

column_meta 中的列类型说明：

- 1：BOOL
- 2：TINYINT
- 3：SMALLINT
- 4：INT
- 5：BIGINT
- 6：FLOAT
- 7：DOUBLE
- 8：BINARY
- 9：TIMESTAMP
- 10：NCHAR

8.5.5. 自定义授权码

HTTP请求中需要带有授权码`<TOKEN>`，用于身份识别。授权码通常由管理员提供，可简单的通过发送`HTTP GET`请求来获取授权码，操作如下：

```
curl http://<fqnd>:<port>/rest/login/<username>/<password>
```

其中，fqdn是TDengine数据库的fqdn或ip地址，port是TDengine服务的端口号，username为数据库用户名，password为数据库密码，返回值为JSON格式，各字段含义如下：

- status：请求结果的标志位
- code：返回值代码
- desc：授权码

获取授权码示例：

```
curl http://192.168.0.1:6041/rest/login/root/taosdata
```

返回值：

```
{
  "status": "succ",
  "code": 0,
  "desc": "/KfeAzX/f9na8qdtNZmtONryp201ma04bEl8LcvLUd7a8qdtNZmtONryp201ma04"
}
```

8.5.6. 使用示例

- 在demo库里查询表d1001的所有记录：

```
curl -H 'Authorization: Basic cm9vdDp0YW9zZGF0YQ==' -d 'select * from demo.d1001'
192.168.0.1:6041/rest/sql
```

返回值：

```
{
    "status": "succ",
    "head": ["ts","current","voltage","phase"],
    "column_meta": [["ts",9,8],["current",6,4],["voltage",4,4],["phase",6,4]],
    "data": [
        ["2018-10-03 14:38:05.000",10.3,219,0.31],
        ["2018-10-03 14:38:15.000",12.6,218,0.33]
    ],
    "rows": 2
}
```

- 创建库demo：

```
curl -H 'Authorization: Basic cm9vdDp0YW9zZGF0YQ==' -d 'create database demo'
192.168.0.1:6041/rest/sql
```

返回值：

```
{
    "status": "succ",
    "head": ["affected_rows"],
    "column_meta": [["affected_rows",4,4]],
    "data": [[1]],
    "rows": 1
}
```

8.5.7. 其他用法

8.5.7.1. 结果集采用Unix时间戳

HTTP请求URL采用sqlt时，返回结果集的时间戳将采用Unix时间戳格式表示，例如

```
curl -H 'Authorization: Basic cm9vdDp0YW9zZGF0YQ==' -d 'select * from demo.d1001' 192.168.0.1:6041/rest/sqlt
```

返回值：

```
{
    "status": "succ",
    "head": ["ts","current","voltage","phase"],
    "column_meta": [["ts",9,8],["current",6,4],["voltage",4,4],["phase",6,4]],
    "data": [
        [1538548685000,10.3,219,0.31],
        [1538548695000,12.6,218,0.33]
    ],
    "rows": 2
}
```

8.5.7.2. 结果集采用UTC时间字符串

HTTP请求URL采用sqlutc时，返回结果集的时间戳将采用UTC时间字符串表示，例如

```
  curl -H 'Authorization: Basic cm9vdDp0YW9zZGF0YQ==' -d 'select * from demo.t1' 192.168.0.1:6041/rest/sqlutc
```

返回值：

```
{
    "status": "succ",
    "head": ["ts","current","voltage","phase"],
    "column_meta": [["ts",9,8],["current",6,4],["voltage",4,4],["phase",6,4]],
    "data": [
        ["2018-10-03T14:38:05.000+0800",10.3,219,0.31],
        ["2018-10-03T14:38:15.000+0800",12.6,218,0.33]
    ],
    "rows": 2
}
```

8.5.8. 重要配置项

下面仅列出一些与RESTful接口有关的配置参数，其他系统参数请看配置文件里的说明。（注意：配置修改后，需要重启taosd服务才能生效）

- 对外提供RESTful服务的端口号，默认绑定到 6041（实际取值是 serverPort + 11，因此可以通过修改 serverPort 参数的设置来修改）
- httpMaxThreads: 启动的线程数量，默认为2（2.0.17.0版本开始，默认值改为CPU核数的一半向下取整）
- restfulRowLimit: 返回结果集（JSON格式）的最大条数，默认值为10240
- httpEnableCompress: 是否支持压缩，默认不支持，目前TDengine仅支持gzip压缩格式
- httpDebugFlag: 日志开关，默认131。131：仅错误和报警信息，135：调试信息，143：非常详细的调试信息，默认131

8.6. CSharp Connector

C#连接器支持的系统有：Linux 64/Windows x64/Windows x86

8.6.1. 安装准备

- 应用驱动安装请参考安装连接器驱动步骤。
- 接口文件TDengineDrivercs.cs和参考程序示例TDengineTest.cs均位于Windows客户端install_directory/examples/C#目录下。
- 在Windows系统上，C#应用程序可以使用TDengine的原生C接口来执行所有数据库操作，后续版本将提供ORM（Dapper）框架驱动。

8.6.2. 示例程序

示例程序源码位于install_directory/examples/C#，有：

TDengineTest.cs　　C#示例源程序

8.6.3. 安装验证

运行install_directory/examples/C#/C#Checker/C#Checker.exe

```
cd {install_directory}/examples/C#/C#Checker
csc /optimize *.cs
C#Checker.exe -h <fqdn>
```

8.6.4. C#连接器的使用

在Windows系统上，C#应用程序可以使用TDengine的C#连接器接口来执行所有数据库的操作。使用的具体步骤如下所示：

1. 将接口文件TDengineDrivercs.cs加入到应用程序所在的项目空间中。
2. 用户可以参考TDengineTest.cs来定义数据库连接参数，以及如何执行数据插入、查询等操作。

此接口需要用到taos.dll文件，所以在执行应用程序前，拷贝Windows客户端install_directory/driver目录中的taos.dll文件到项目最后生成.exe可执行文件所在的文件夹。之后运行exe文件，即可访问TDengine数据库并做插入、查询等操作。

注意：

1. TDengine V2.0.3.0之后同时支持32位和64位Windows系统，所以C#项目在生成.exe文件时，“解决方案”/“项目”的“平台”请选择对应的“X86” 或“x64”。
2. 此接口目前已经在Visual Studio 2015/2017中验证过，其它VS版本尚待验证。

8.6.5. 第三方驱动

Maikebing.Data.Taos是一个TDengine的ADO.Net提供器，支持linux，windows。该开发包由热心贡献者麦壳饼@@maikebing提供，具体请参考

```
//接口下载
https://github.com/maikebing/Maikebing.EntityFrameworkCore.Taos
//用法说明
https://www.taosdata.com/blog/2020/11/02/1901.html
```

8.7. Go Connector

8.7.1. 安装准备

Go连接器支持的系统有：

CPU类型	x64（64bit）			aarch64	aarch32
OS类型	Linux	Win64	Win32	Linux	Linux
支持与否	支持	支持	支持	支持	开发中

安装前准备：

- 已安装好TDengine应用驱动，参考安装连接器驱动步骤。

8.7.2. 示例程序

使用 Go 连接器的示例代码请参考 https://github.com/taosdata/TDengine/tree/develop/tests/examples/go 以及视频教程。

示例程序源码也位于安装目录下的 examples/go/taosdemo.go 文件中。

提示：建议Go版本是1.13及以上，并开启模块支持：

```
go env -w GO111MODULE=on
go env -w GOPROXY=https://goproxy.io,direct
```

在taosdemo.go所在目录下进行编译和执行：

```
go mod init *demo*
go build   ./demo -h fqdn -p serverPort
```

8.7.3. Go连接器的使用

TDengine提供了GO驱动程序包`taosSql`.`taosSql`实现了GO语言的内置接口`database/sql/driver`。用户只需按如下方式引入包就可以在应用程序中访问TDengine。

```
import (
  "database/sql"
  _ "github.com/taosdata/driver-go/taosSql"
)
```

提示：下划线与双引号之间必须有一个空格。

8.7.4. 常用API

- `sql.Open(DRIVER_NAME string, dataSourceName string) *DB`

  该API用来打开DB，返回一个类型为*DB的对象，一般情况下，DRIVER_NAME设置为字符串`taosSql`，dataSourceName设置为字符串`user:password@/tcp(host:port)/dbname`，如果客户想要用多个goroutine并发访问TDengine, 那么需要在各个goroutine中分别创建一个sql.Open对象并用之访问TDengine

  注意： 该API成功创建的时候，并没有做权限等检查，只有在真正执行Query或者Exec的时候才能真正的去创建连接，并同时检查user/password/host/port是不是合法。另外，由于整个驱动程序大部分实现都下沉到taosSql所依赖的libtaos动态库中。所以，sql.Open本身特别轻量。

- `func (db *DB) Exec(query string, args ...interface{}) (Result, error)`

  sql.Open内置的方法，用来执行非查询相关SQL

- `func (db *DB) Query(query string, args ...interface{}) (*Rows, error)`

  sql.Open内置的方法，用来执行查询语句

- `func (db *DB) Prepare(query string) (*Stmt, error)`

  sql.Open内置的方法，Prepare creates a prepared statement for later queries or executions.

- `func (s *Stmt) Exec(args ...interface{}) (Result, error)`

  sql.Open内置的方法，executes a prepared statement with the given arguments and returns a Result summarizing the effect of the statement.

- `func (s *Stmt) Query(args ...interface{}) (*Rows, error)`

  sql.Open内置的方法，Query executes a prepared query statement with the given arguments and returns the query results as a *Rows.

- `func (s *Stmt) Close() error`

  sql.Open内置的方法，Close closes the statement.

8.7.5. 其他代码示例

Consume Messages from Kafka 是一个通过 Go 语言实现消费 Kafka 队列写入 TDengine 的示例程序，也可以作为通过 Go 连接 TDengine 的写法参考。

8.8. Node.js Connector

Node.js连接器支持的系统有：

CPU类型	x64（64bit）			aarch64	aarch32
OS类型	Linux	Win64	Win32	Linux	Linux
支持与否	支持	支持	支持	支持	支持

Node.js连接器的使用参见视频教程。

8.8.1. 安装准备

- 应用驱动安装请参考安装连接器驱动步骤。

8.8.2. 安装Node.js连接器

用户可以通过npm来进行安装，也可以通过源代码*src/connector/nodejs/* 来进行安装。具体安装步骤如下：

首先，通过npm安装node.js 连接器。

```
npm install td2.0-connector
```

我们建议用户使用npm 安装node.js连接器。如果您没有安装npm，可以将*src/connector/nodejs/*拷贝到您的nodejs 项目目录下。

我们使用node-gyp和TDengine服务端进行交互。安装node.js连接器之前，还需要根据具体操作系统来安装下文提到的一些依赖工具。

8.8.3. Linux

- `python` (建议`v2.7`, `v3.x.x` 目前还不支持)
- `node` 2.0.6支持v12.x和v10.x，2.0.5及更早版本支持v10.x版本，其他版本可能存在包兼容性的问题。
- `make`
- c语言编译器比如GCC

8.8.4. Windows

8.8.4.1. 安装方法1

使用微软的windows-build-tools在cmd 命令行界面执行`npm install --global --production windows-build-tools` 即可安装所有的必备工具。

8.8.4.2. 安装方法2

手动安装以下工具：

- 安装Visual Studio相关：Visual Studio Build 工具 或者 Visual Studio 2017 Community
- 安装 Python 2.7(`v3.x.x` 暂不支持) 并执行 `npm config set python python2.7`
- 进入cmd命令行界面，`npm config set msvs_version 2017`

如果以上步骤不能成功执行，可以参考微软的node.js用户手册Microsoft's Node.js Guidelines for Windows。

如果在Windows 10 ARM 上使用ARM64 Node.js，还需添加 "Visual C++ compilers and libraries for ARM64" 和 "Visual C++ ATL for ARM64"。

8.8.5. 示例程序

示例程序源码位于install_directory/examples/nodejs，有：

Node-example.js　　node.js示例源程序
Node-example-raw.js

8.8.6. 安装验证

在安装好TDengine客户端后，使用nodejsChecker.js程序能够验证当前环境是否支持nodejs方式访问Tdengine。

验证方法：

1. 新建安装验证目录，例如：`~/tdengine-test`，拷贝github上nodejsChecker.js源程序。下载地址：（https://github.com/taosdata/TDengine/tree/develop/tests/examples/nodejs/nodejsChecker.js）。
2. 在命令行中执行以下命令：

```
npm init -y
npm install td2.0-connector
node nodejsChecker.js host=localhost
```

3. 执行以上步骤后，在命令行会输出nodejs连接Tdengine实例，并执行简答插入和查询的结果。

8.8.7. Node.js连接器的使用

以下是Node.js 连接器的一些基本使用方法，详细的使用方法可参考TDengine Node.js connector。

8.8.7.1. 建立连接

使用node.js连接器时，必须先`require td2.0-connector`，然后使用 `taos.connect` 函数建立到服务端的连接。例如如下代码：

```
const taos = require('td2.0-connector');
var conn = taos.connect({host:"taosdemo.com", user:"root", password:"taosdata",
config:"/etc/taos",port:6030})
var cursor = conn.cursor(); // Initializing a new cursor
```

建立了一个到hostname为taosdemo.com，端口为6030（Tdengine的默认端口号）的连接。连接指定了用户名（root）和密码（taosdata）。taos.connect 函数必须提供的参数是`host`，其它参数在没有提供的情况下会使用如下的默认值。taos.connect返回了`cursor` 对象，使用cursor来执行sql语句。

8.8.7.2. 执行SQL和插入数据

对于DDL语句（例如create database、create table、use等），可以使用cursor的execute方法。代码如下：

```
cursor.execute('create database if not exists test;')
```

以上代码创建了一个名称为test的数据库。对于DDL语句，一般没有返回值，cursor的execute返回值为0。

对于Insert语句，代码如下：

```
var affectRows = cursor.execute('insert into test.weather values(now, 22.3, 34);')
```

execute方法的返回值为该语句影响的行数，上面的sql向test库的weather表中，插入了一条数据，则返回值affectRows为1。

TDengine目前还不支持update和delete语句。

8.8.7.3. 查询

可通过 `cursor.query` 函数来查询数据库。

```
var query = cursor.query('show databases;')
```

查询的结果可以通过 `query.execute()` 函数获取并打印出来。

```
var promise = query.execute();
promise.then(function(result) {
  result.pretty();
});
```

格式化查询语句还可以使用`query`的`bind`方法。如下面的示例：`query`会自动将提供的数值填入查询语句的`?`里。

```
var query = cursor.query('select * from meterinfo.meters where ts <= ? and areaid = ?;').bind(new Date(), 5);
query.execute().then(function(result) {
  result.pretty();
})
```

如果在query语句里提供第二个参数并设为true也可以立即获取查询结果。如下：

```
var promise = cursor.query('select * from meterinfo.meters where v1 = 30;', true)
promise.then(function(result) {
  result.pretty();
})
```

8.8.7.4. 关闭连接

在完成插入、查询等操作后，要关闭连接。代码如下：

```
conn.close();
```

8.8.7.5. 异步函数

异步查询数据库的操作和上面类似，只需要在cursor.execute, TaosQuery.execute等函数后面加上_a。

```
var promise1 = cursor.query('select count(*), avg(v1), avg(v2) from meter1;').execute_a()
var promise2 = cursor.query('select count(*), avg(v1), avg(v2) from meter2;').execute_a();
promise1.then(function(result) {
  result.pretty();
})
promise2.then(function(result) {
  result.pretty();
})
```

8.8.8. 示例

node-example.js提供了一个使用NodeJS 连接器建表，插入天气数据并查询插入的数据的代码示例。

node-example-raw.js同样是一个使用NodeJS 连接器建表，插入天气数据并查询插入的数据的代码示例，但和上面不同的是，该示例只使用cursor。

9. 与其他工具的连接

9.1. Grafana

TDengine 能够与开源数据可视化系统 Grafana快速集成搭建数据监测报警系统，整个过程无需任何代码开发，TDengine 中数据表中内容可以在仪表盘(DashBoard)上进行可视化展现。

9.1.1. 安装Grafana

目前 TDengine 支持 Grafana 6.2 以上的版本。用户可以根据当前的操作系统，到 Grafana 官网下载安装包，并执行安装。下载地址如下：https://grafana.com/grafana/download。

9.1.2. 配置Grafana

TDengine 的 Grafana 插件在安装包的 /usr/local/taos/connector/grafanaplugin 目录下。

以 CentOS 7.2 操作系统为例，将 grafanaplugin 目录拷贝到 /var/lib/grafana/plugins 目录下，重新启动 grafana 即可。

```
sudo cp -rf /usr/local/taos/connector/grafanaplugin
/var/lib/grafana/plugins/tdengine
```

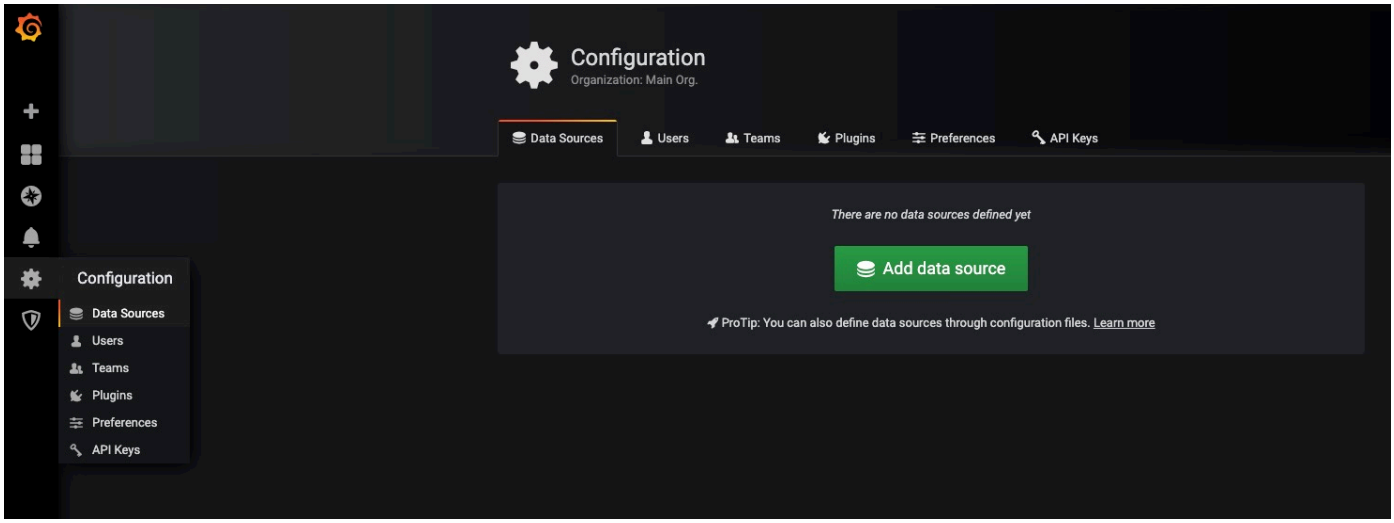
Grafana 8.x 版本会对插件进行签名检查，因此还需要在 grafana.ini 文件中增加如下行，才能正确使用插件：

```
[plugins]
enable_alpha = true
allow_loading_unsigned_plugins = taosdata-tdengine-datasource
```

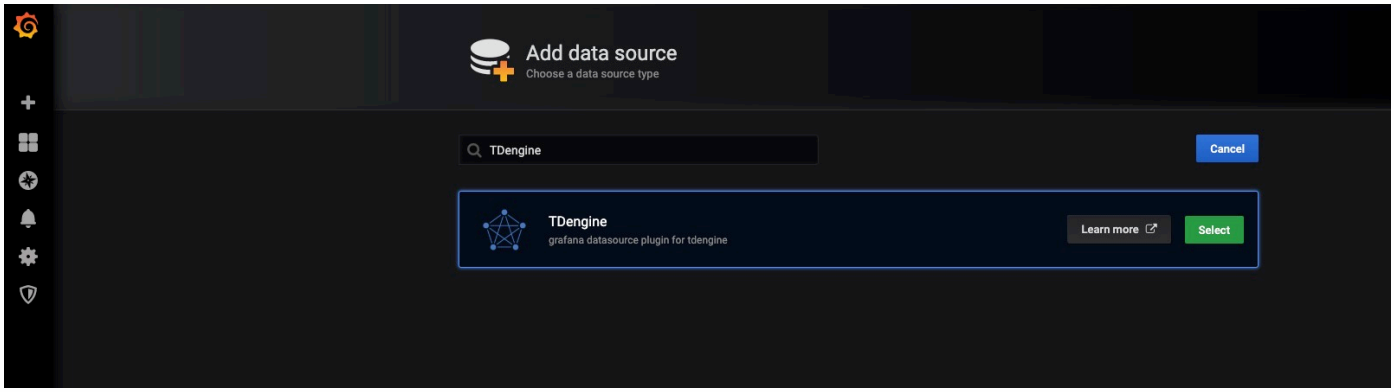
9.1.3. 使用 Grafana

9.1.3.1. 配置数据源

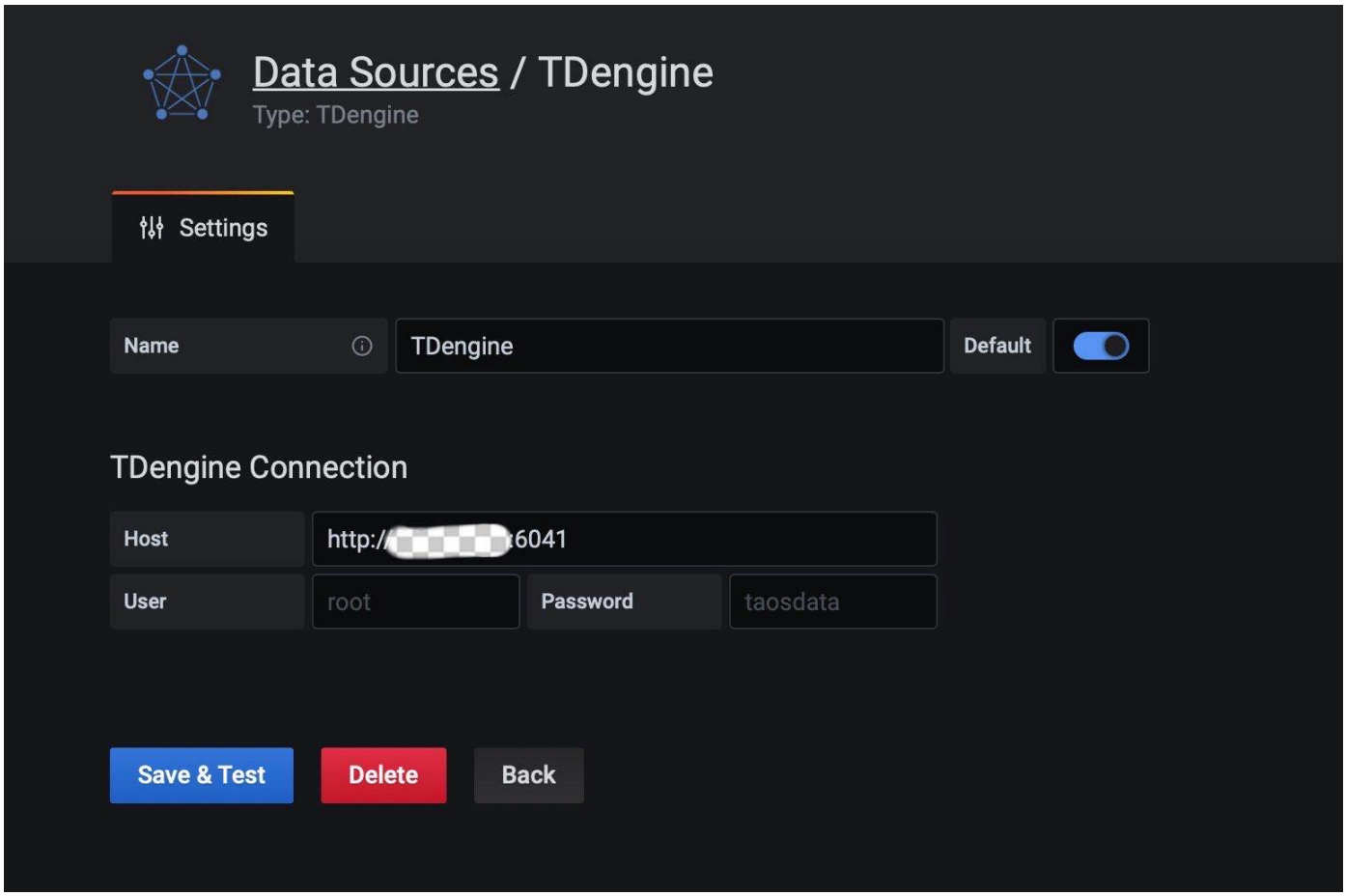
用户可以直接通过 localhost:3000 的网址，登录 Grafana 服务器（用户名/密码：admin/admin），通过左侧 `Configuration -> Data Sources` 可以添加数据源，如下图所示：



点击 `Add data source` 可进入新增数据源页面，在查询框中输入 TDengine 可选择添加，如下图所示：

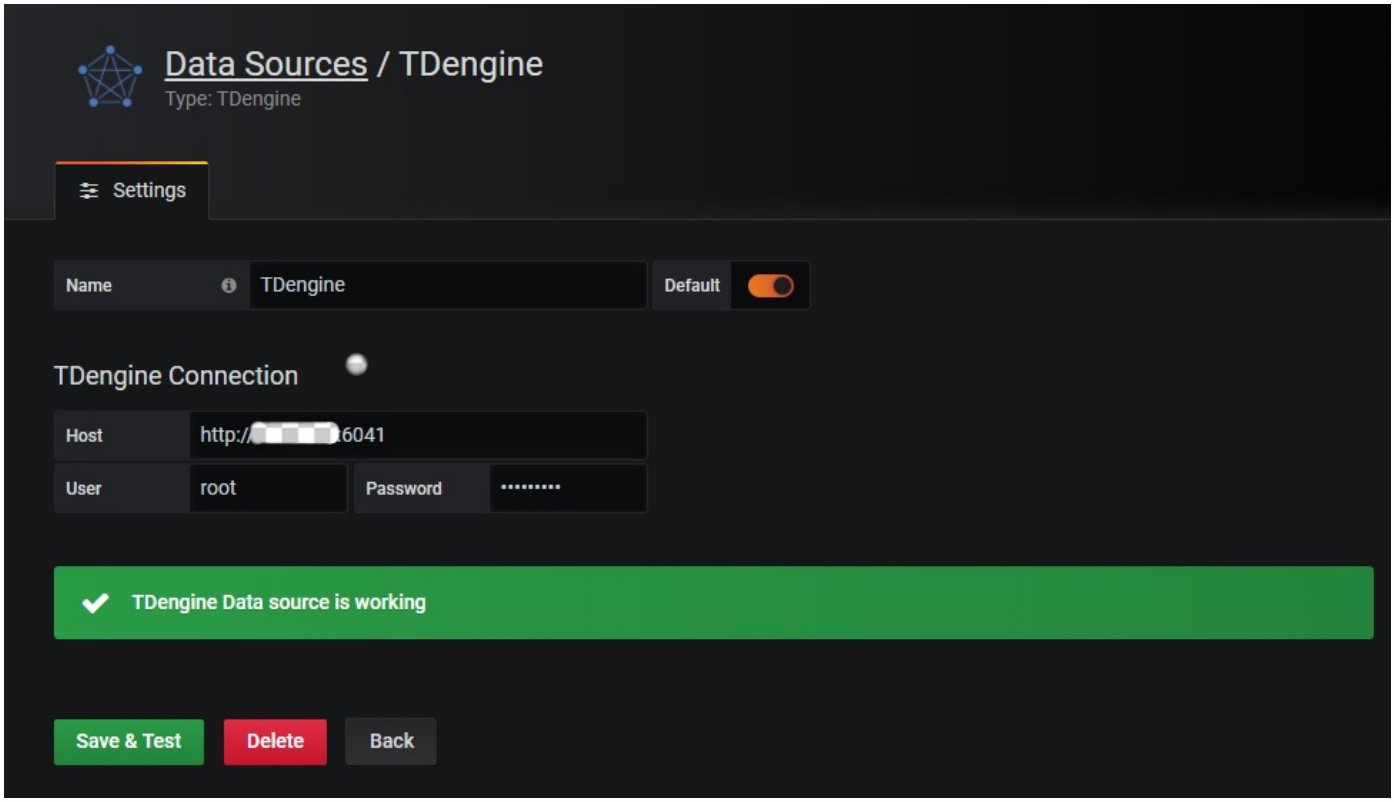


进入数据源配置页面，按照默认提示修改相应配置即可：



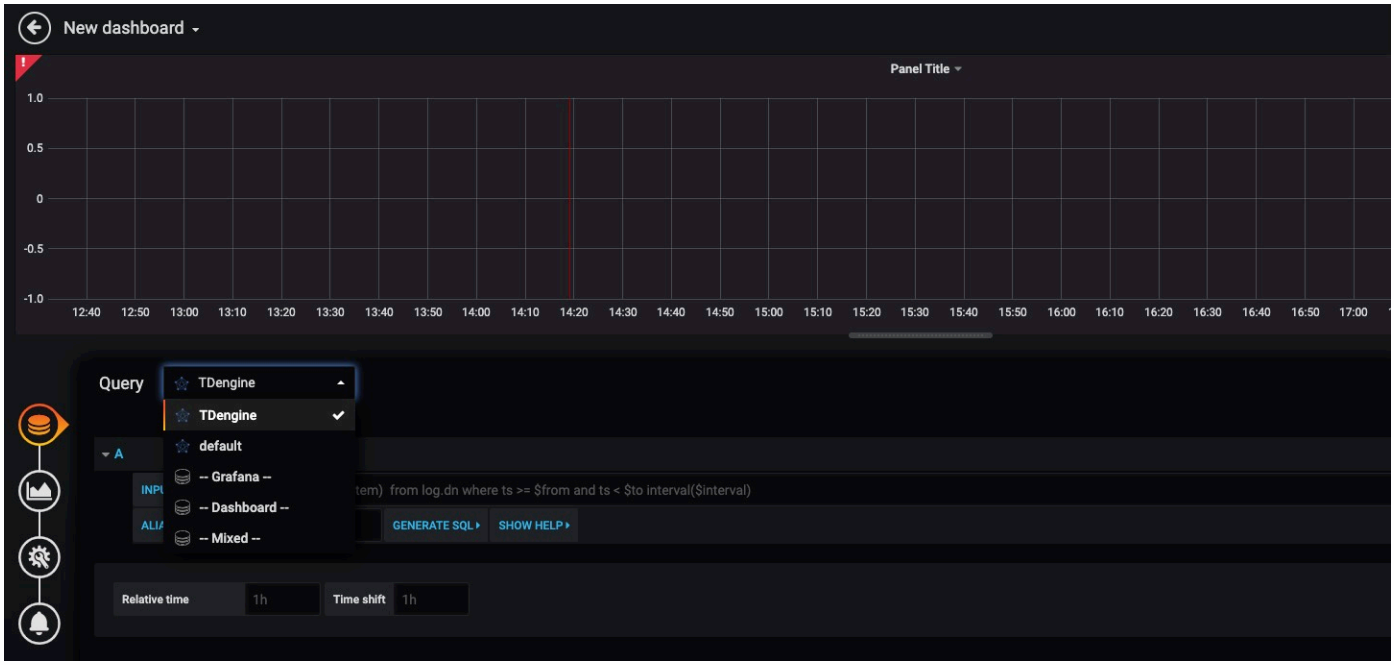
- Host： TDengine 集群的中任意一台服务器的 IP 地址与 TDengine RESTful 接口的端口号(6041)，默认 http://localhost:6041 。
- User：TDengine 用户名。
- Password：TDengine 用户密码。

点击 `Save & Test` 进行测试，成功会有如下提示：



9.1.3.2. 创建 Dashboard

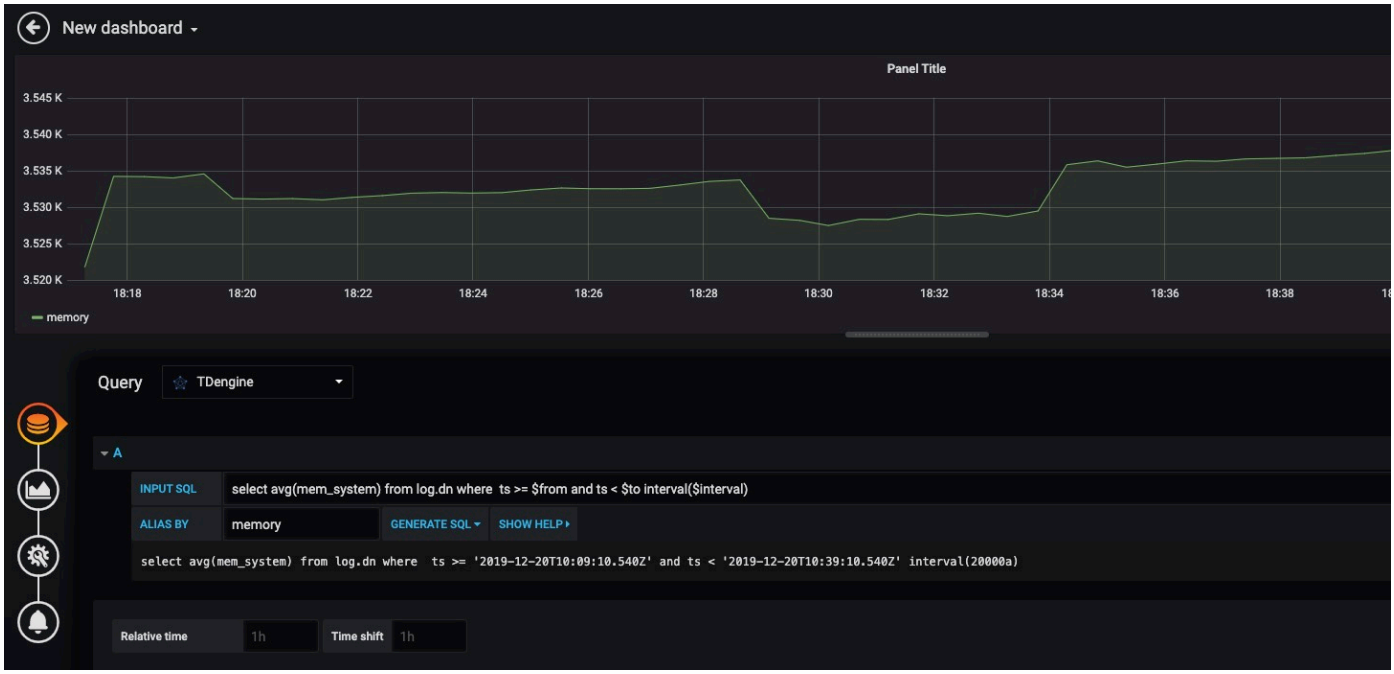
回到主界面创建 Dashboard，点击 Add Query 进入面板查询页面：



如上图所示，在 Query 中选中 `TDengine` 数据源，在下方查询框可输入相应 sql 进行查询，具体说明如下：

- INPUT SQL：输入要查询的语句（该 SQL 语句的结果集应为两列多行），例如：`select avg(mem_system) from log.dn where  ts >= $from and ts < $to interval($interval)`，其中，from、to 和 interval 为 TDengine插件的内置变量，表示从Grafana插件面板获取的查询范围和时间间隔。除了内置变量外，也支持可以使用自定义模板变量。
- ALIAS BY：可设置当前查询别名。
- GENERATE SQL： 点击该按钮会自动替换相应变量，并生成最终执行的语句。

按照默认提示查询当前 TDengine 部署所在服务器指定间隔系统内存平均使用量如下：

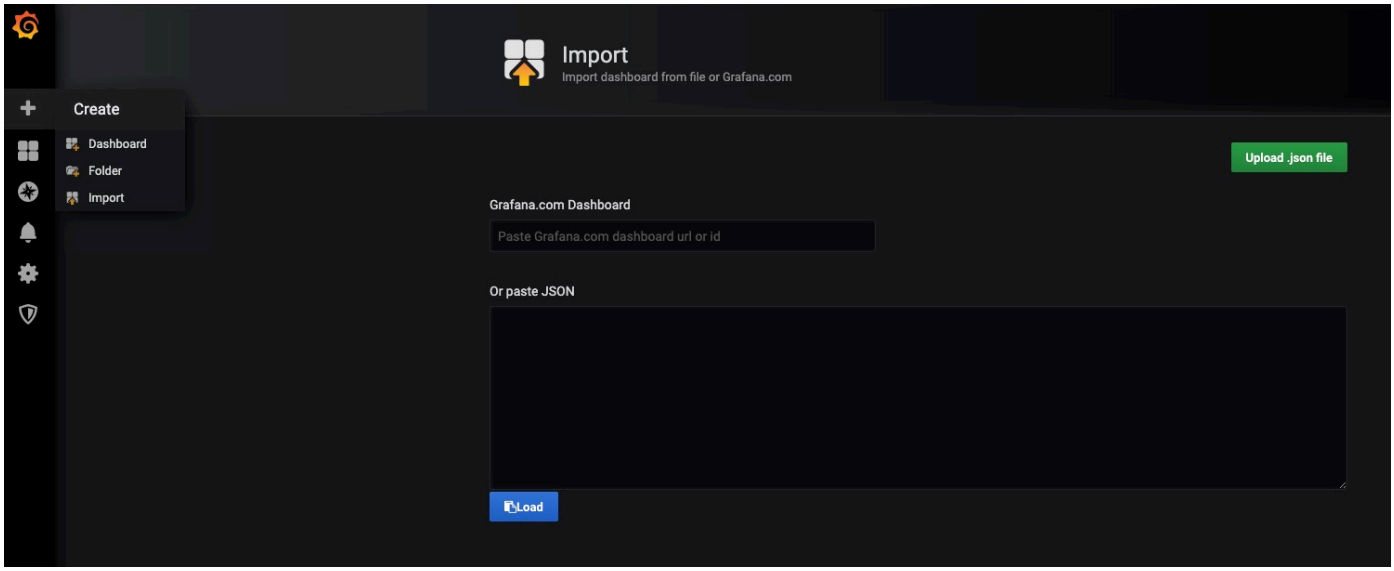


> 关于如何使用Grafana创建相应的监测界面以及更多有关使用Grafana的信息，请参考Grafana官方的文档。

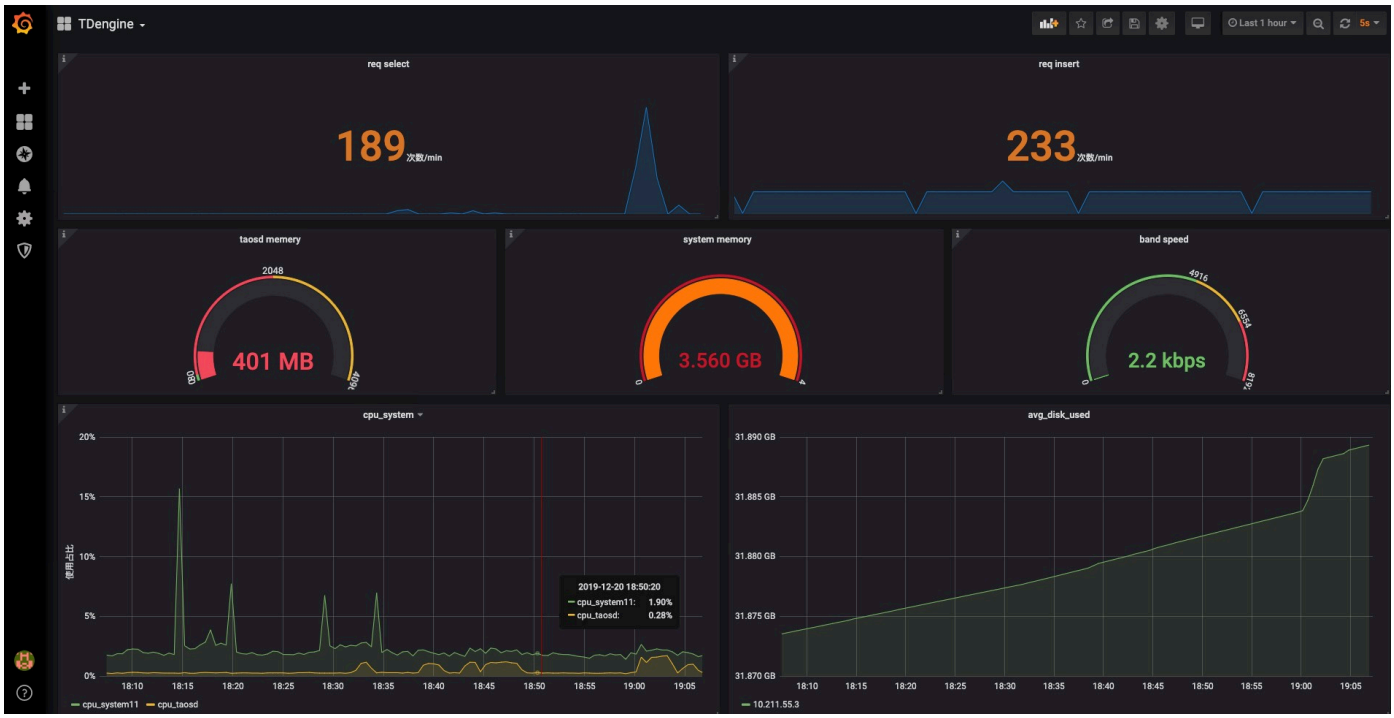
9.1.3.3. 导入 Dashboard

在 Grafana 插件目录 /usr/local/taos/connector/grafanaplugin/dashboard 下提供了一个 `tdengine-grafana.json` 可导入的 dashboard。

点击左侧 `Import` 按钮，并上传 `tdengine-grafana.json` 文件：



导入完成之后可看到如下效果：



9.2. MATLAB

MATLAB 可以通过安装包内提供的 JDBC Driver 直接连接到 TDengine 获取数据到本地工作空间。

9.2.1. MATLAB 的 JDBC 接口适配

MATLAB 的适配有下面几个步骤，下面以 Windows 10 上适配 MATLAB2021a 为例：

- 将 TDengine 客户端安装路径下的 `\TDengine\connector\jdbc的驱动程序taos-jdbcdriver-2.0.25-dist.jar` 拷贝到 `${matlab_root}\MATLAB\R2021a\java\jar\toolbox`。
- 将 TDengine 安装包内的 `taos.lib` 文件拷贝至 `${matlab_root_dir}\MATLAB\R2021\lib\win64`。
- 将新添加的驱动 jar 包加入 MATLAB 的 classpath。在 `${matlab_root_dir}\MATLAB\R2021a\toolbox\local\classpath.txt` 文件中添加下面一行：

```
$matlabroot/java/jar/toolbox/taos-jdbcdriver-2.0.25-dist.jar
```

- 在 `${user_home}\AppData\Roaming\MathWorks\MATLAB\R2021a\` 下添加一个文件 `javalibrarypath.txt`，并在该文件中添加 taos.dll 的路径，比如您的 taos.dll 是在安装时拷贝到了 `C:\Windows\System32` 下，那么就应该在 `javalibrarypath.txt` 中添加如下一行：

```
C:\Windows\System32
```

9.2.2. 在 MATLAB 中连接 TDengine 获取数据

在成功进行了上述配置后，打开 MATLAB。

- 创建一个连接：

```
conn = database('test', 'root', 'taosdata', 'com.taosdata.jdbc.TSDBDriver',
'jdbc:TSDB://192.168.1.94:6030/')
```

- 执行一次查询：

```
sql0 = ['select * from tb']
data = select(conn, sql0);
```

- 插入一条记录：

```
sql1 = ['insert into tb values (now, 1)']
exec(conn, sql1)
```

更多例子细节请参考安装包内 `examples\Matlab\TDengineDemo.m` 文件。

9.3. R

R语言支持通过JDBC接口来连接TDengine数据库。首先需要安装R语言的JDBC包。启动R语言环境，然后执行以下命令安装R语言的JDBC支持库：

```
install.packages('RJDBC', repos='http://cran.us.r-project.org')
```

安装完成以后，通过执行`library('RJDBC')`命令加载 *RJDBC* 包：

然后加载TDengine的JDBC驱动：

```
drv<-JDBC("com.taosdata.jdbc.TSDBDriver","JDBCDriver-2.0.0-dist.jar",
identifier.quote="\"")
```

如果执行成功，不会出现任何错误信息。之后通过以下命令尝试连接数据库：

```
conn<-dbConnect(drv,"jdbc:TSDB://192.168.0.1:0/?
user=root&password=taosdata","root","taosdata")
```

注意将上述命令中的IP地址替换成正确的IP地址。如果没有任务错误的信息，则连接数据库成功，否则需要根据错误提示调整连接的命令。TDengine支持以下的 *RJDBC* 包中函数：

- dbWriteTable(conn, "test", iris, overwrite=FALSE, append=TRUE)：将数据框iris写入表test中，overwrite必须设置为false，append必须设为TRUE,且数据框iris要与表test的结构一致。
- dbGetQuery(conn, "select count(*) from test")：查询语句。
- dbSendUpdate(conn, "use db")：执行任何非查询sql语句。例如dbSendUpdate(conn, "use db")， 写入数据dbSendUpdate(conn, "insert into t1 values(now, 99)")等。
- dbReadTable(conn, "test")：读取表test中数据。
- dbDisconnect(conn)：关闭连接。
- dbRemoveTable(conn, "test")：删除表test。

TDengine客户端暂不支持如下函数：

- dbExistsTable(conn, "test")：是否存在表test。
- dbListTables(conn)：显示连接中的所有表。

10. TDengine 集群安装、管理

多个TDengine服务器，也就是多个taosd的运行实例可以组成一个集群，以保证TDengine的高可靠运行，并提供水平扩展能力。要了解TDengine 2.0的集群管理，需要对集群的基本概念有所了解，请看《TDengine整体架构》一章。而且在安装集群之前，建议先按照《立即开始》一章安装并体验单节点功能。

集群的每个数据节点是由End Point来唯一标识的，End Point是由FQDN(Fully Qualified Domain Name)外加Port组成，比如 h1.taosdata.com:6030。一般FQDN就是服务器的hostname，可通过Linux命令`hostname -f`获取（如何配置FQDN，请参考：一篇文章说清楚TDengine的FQDN）。端口是这个数据节点对外服务的端口号，缺省是6030，但可以通过taos.cfg里配置参数serverPort进行修改。一个物理节点可能配置了多个hostname, TDengine会自动获取第一个，但也可以通过taos.cfg里配置参数fqdn进行指定。如果习惯IP地址直接访问，可以将参数fqdn设置为本节点的IP地址。

TDengine的集群管理极其简单，除添加和删除节点需要人工干预之外，其他全部是自动完成，最大程度的降低了运维的工作量。本章对集群管理的操作做详细的描述。

关于集群搭建请参考视频教程。

10.1. 准备工作

第零步：规划集群所有物理节点的FQDN，将规划好的FQDN分别添加到每个物理节点的/etc/hostname；修改每个物理节点的/etc/hosts，将所有集群物理节点的IP与FQDN的对应添加好。【如部署了DNS，请联系网络管理员在DNS上做好相关配置】

第一步：如果搭建集群的物理节点中，存有之前的测试数据、装过1.X的版本，或者装过其他版本的TDengine，请先将其删除，并清空所有数据（如果需要保留原有数据，请联系涛思交付团队进行旧版本升级、数据迁移），具体步骤请参考博客《TDengine多种安装包的安装和卸载》。
注意1：因为FQDN的信息会写进文件，如果之前没有配置或者更改FQDN，且启动了TDengine。请一定在确保数据无用或者备份的前提下，清理一下之前的数据（`rm -rf /var/lib/taos/*`）；
注意2：客户端也需要配置，确保它可以正确解析每个节点的FQDN配置，不管是通过DNS服务，还是 Host 文件。

第二步：建议关闭所有物理节点的防火墙，至少保证端口：6030 - 6042的TCP和UDP端口都是开放的。**强烈建议**先关闭防火墙，集群搭建完毕之后，再来配置端口；

第三步：在所有物理节点安装TDengine，且版本必须是一致的，**但不要启动taosd**。安装时，提示输入是否要加入一个已经存在的TDengine集群时，第一个物理节点直接回车创建新集群，后续物理节点则输入该集群任何一个在线的物理节点的FQDN:端口号(默认6030)；

第四步：检查所有数据节点，以及应用程序所在物理节点的网络设置：

1. 每个物理节点上执行命令`hostname -f`，查看和确认所有节点的hostname是不相同的(应用驱动所在节点无需做此项检查)；
2. 每个物理节点上执行`ping host`，其中host是其他物理节点的hostname，看能否ping通其它物理节点；如果不能ping通，需要检查网络设置，或/etc/hosts文件(Windows系统默认路径为C:\Windows\system32\drivers\etc\hosts)，或DNS的配置。如果无法ping通，是无法组成集群的；

3. 从应用运行的物理节点，ping taosd运行的数据节点，如果无法ping通，应用是无法连接taosd的，请检查应用所在物理节点的DNS设置或hosts文件；
4. 每个数据节点的End Point就是输出的hostname外加端口号，比如`h1.taosdata.com:6030`。

第五步：修改TDengine的配置文件（所有节点的文件/etc/taos/taos.cfg都需要修改）。假设准备启动的第一个数据节点End Point为 h1.taosdata.com:6030，其与集群配置相关参数如下：

```
// firstEp 是每个数据节点首次启动后连接的第一个数据节点
firstEp                 h1.taosdata.com:6030

// 必须配置为本数据节点的FQDN，如果本机只有一个hostname，可注释掉本项
fqdn                    h1.taosdata.com

// 配置本数据节点的端口号，缺省是6030
serverPort              6030

// 副本数为偶数的时候，需要配置，请参考《Arbitrator的使用》的部分
arbitrator              ha.taosdata.com:6042
```

一定要修改的参数是firstEp和fqdn。在每个数据节点，firstEp需全部配置成一样，但**fqdn一定要配置成其所在数据节点的值**。其他参数可不做任何修改，除非你很清楚为什么要修改。

加入到集群中的数据节点dnode，涉及集群相关的下表11项参数必须完全相同，否则不能成功加入到集群中。

#	配置参数名称	含义
1	numOfMnodes	系统中管理节点个数
2	mnodeEqualVnodeNum	一个mnode等同于vnode消耗的个数
3	offlineThreshold	dnode离线阈值，超过该时间将导致Dnode离线
4	statusInterval	dnode向mnode报告状态时长
5	arbitrator	系统中裁决器的End Point
6	timezone	时区
7	balance	是否启动负载均衡
8	maxTablesPerVnode	每个vnode中能够创建的最大表个数
9	maxVgroupsPerDb	每个DB中能够使用的最大vgroup个数

备注：在 2.0.19.0 及更早的版本中，除以上 9 项参数外，dnode 加入集群时，还会要求 locale 和 charset 参数的取值也一致。

10.2. 启动第一个数据节点

按照《立即开始》里的指示，启动第一个数据节点，例如h1.taosdata.com，然后执行taos, 启动taos shell，从shell里执行命令"show dnodes;"，如下所示：

```
Welcome to the TDengine shell from Linux, Client Version:2.0.0.0
Copyright (c) 2017 by TAOS Data, Inc. All rights reserved.

taos> show dnodes;
 id |       end_point    | vnodes | cores | status | role |      create_time
|
================================================================================
===
  1 |  h1.taos.com:6030  |      0 |     2 |  ready |  any | 2020-07-31
03:49:29.202 |
Query OK, 1 row(s) in set (0.006385s)

taos>
```

上述命令里，可以看到这个刚启动的这个数据节点的End Point是：h1.taos.com:6030，就是这个新集群的firstEP。

10.3. 启动后续数据节点

将后续的数据节点添加到现有集群，具体有以下几步：

1. 按照《立即开始》一章的方法在每个物理节点启动taosd；（注意：每个物理节点都需要在 taos.cfg 文件中将 firstEP 参数配置为新集群首个节点的 End Point——在本例中是 h1.taos.com:6030）
2. 在第一个数据节点，使用CLI程序taos，登录进TDengine系统，执行命令：

   ```
   CREATE DNODE "h2.taos.com:6030";
   ```

   将新数据节点的End Point (准备工作中第四步获知的) 添加进集群的EP列表。**"fqdn:port"需要用双引号引起来**，否则出错。请注意将示例的“h2.taos.com:6030" 替换为这个新数据节点的End Point。
3. 然后执行命令

   ```
   SHOW DNODES;
   ```

   查看新节点是否被成功加入。如果该被加入的数据节点处于离线状态，请做两个检查：
   - 查看该数据节点的taosd是否正常工作，如果没有正常运行，需要先检查为什么
   - 查看该数据节点taosd日志文件taosdlog.0里前面几行日志(一般在/var/log/taos目录)，看日志里输出的该数据节点fqdn以及端口号是否为刚添加的End Point。如果不一致，需要将正确的End Point添加进去。

按照上述步骤可以源源不断的将新的数据节点加入到集群。

提示：

- 任何已经加入集群在线的数据节点，都可以作为后续待加入节点的 firstEP。
- firstEp 这个参数仅仅在该数据节点首次加入集群时有作用，加入集群后，该数据节点会保存最新的 mnode 的 End Point 列表，不再依赖这个参数。

    - 接下来，配置文件中的 firstEp 参数就主要在客户端连接的时候使用了，例如 taos shell 如果不加参数，会默认连接由 firstEp 指定的节点。
- 两个没有配置 firstEp 参数的数据节点 dnode 启动后，会独立运行起来。这个时候，无法将其中一个数据节点加入到另外一个数据节点，形成集群。**无法将两个独立的集群合并成为新的集群**。

10.4. 数据节点管理

上面已经介绍如何从零开始搭建集群。集群组建完后，还可以随时添加新的数据节点进行扩容，或删除数据节点，并检查集群当前状态。

10.4.1. 添加数据节点

执行CLI程序taos，使用root账号登录进系统，执行：

```
CREATE DNODE "fqdn:port";
```

将新数据节点的End Point添加进集群的EP列表。**"fqdn:port"需要用双引号引起来**，否则出错。一个数据节点对外服务的fqdn和port可以通过配置文件taos.cfg进行配置，缺省是自动获取。【强烈不建议用自动获取方式来配置FQDN，可能导致生成的数据节点的End Point不是所期望的】

10.4.2. 删除数据节点

执行CLI程序taos，使用root账号登录进TDengine系统，执行：

```
DROP DNODE "fqdn:port | dnodeID";
```

通过"fqdn:port"或"dnodeID"来指定一个具体的节点都是可以的。其中fqdn是被删除的节点的FQDN，port是其对外服务器的端口号；dnodeID可以通过SHOW DNODES获得。

【注意】

- 一个数据节点一旦被drop之后，不能重新加入集群。需要将此节点重新部署（清空数据文件夹）。集群在完成drop dnode操作之前，会将该dnode的数据迁移走。
- 请注意 drop dnode 和 停止taosd进程是两个不同的概念，不要混淆：因为删除dnode之前要执行迁移数据的操作，因此被删除的dnode必须保持在线状态。待删除操作结束之后，才能停止taosd进程。
- 一个数据节点被drop之后，其他节点都会感知到这个dnodeID的删除操作，任何集群中的节点都不会再接收此dnodeID的请求。
- dnodeID是集群自动分配的，不得人工指定。它在生成时是递增的，不会重复。

10.4.3. 手动迁移数据节点

手动将某个vnode迁移到指定的dnode。

执行CLI程序taos，使用root账号登录进TDengine系统，执行：

```
ALTER DNODE <source-dnodeId> BALANCE "VNODE:<vgId>-DNODE:<dest-dnodeId>";
```

其中：source-dnodeId是源dnodeId，也就是待迁移的vnode所在的dnodeID；vgId可以通过SHOW VGROUPS获得，列表的第一列；dest-dnodeId是目标dnodeId。

【注意】

- 只有在集群的自动负载均衡选项关闭时(balance设置为0)，才允许手动迁移。
- 只有处于正常工作状态的vnode才能被迁移：master/slave，当处于offline/unsynced/syncing状态时，是不能迁移的。
- 迁移前，务必核实目标dnode的资源足够：CPU、内存、硬盘。

10.4.4. 查看数据节点

执行CLI程序taos，使用root账号登录进TDengine系统，执行：

```
SHOW DNODES;
```

它将列出集群中所有的dnode，每个dnode的ID，end_point(fqdn:port)，状态(ready, offline等），vnode数目，还未使用的vnode数目等信息。在添加或删除一个数据节点后，可以使用该命令查看。

10.4.5. 查看虚拟节点组

为充分利用多核技术，并提供scalability，数据需要分片处理。因此TDengine会将一个DB的数据切分成多份，存放在多个vnode里。这些vnode可能分布在多个数据节点dnode里，这样就实现了水平扩展。一个vnode仅仅属于一个DB，但一个DB可以有多个vnode。vnode的是mnode根据当前系统资源的情况，自动进行分配的，无需任何人工干预。

执行CLI程序taos，使用root账号登录进TDengine系统，执行：

```
SHOW VGROUPS;
```

10.5. vnode的高可用性

TDengine通过多副本的机制来提供系统的高可用性，包括vnode和mnode的高可用性。

vnode的副本数是与DB关联的，一个集群里可以有多个DB，根据运营的需求，每个DB可以配置不同的副本数。创建数据库时，通过参数replica 指定副本数（缺省为1）。如果副本数为1，系统的可靠性无法保证，只要数据所在的节点宕机，就将无法提供服务。集群的节点数必须大于等于副本数，否则创建表时将返回错误"more dnodes are needed"。比如下面的命令将创建副本数为3的数据库demo：

```
1 CREATE DATABASE demo replica 3;
```

一个DB里的数据会被切片分到多个vnode group，vnode group里的vnode数目就是DB的副本数，同一个vnode group里各vnode的数据是完全一致的。为保证高可用性，vnode group里的vnode一定要分布在不同的数据节点dnode里（实际部署时，需要在不同的物理机上），只要一个vgroup里超过半数的vnode处于工作状态，这个vgroup就能正常的对外服务。

一个数据节点dnode里可能有多个DB的数据，因此一个dnode离线时，可能会影响到多个DB。如果一个vnode group里的一半或一半以上的vnode不工作，那么该vnode group就无法对外服务，无法插入或读取数据，这样会影响到它所属的DB的一部分表的读写操作。

因为vnode的引入，无法简单地给出结论：“集群中过半数据节点dnode工作，集群就应该工作”。但是对于简单的情形，很好下结论。比如副本数为3，只有三个dnode，那如果仅有一个节点不工作，整个集群还是可以正常工作的，但如果有两个数据节点不工作，那整个集群就无法正常工作了。

10.6. Mnode的高可用性

TDengine集群是由mnode (taosd的一个模块，管理节点) 负责管理的，为保证mnode的高可用，可以配置多个mnode副本，副本数由系统配置参数numOfMnodes决定，有效范围为1-3。为保证元数据的强一致性，mnode副本之间是通过同步的方式进行数据复制的。

一个集群有多个数据节点dnode，但一个dnode至多运行一个mnode实例。多个dnode情况下，哪个dnode可以作为mnode呢？这是完全由系统根据整个系统资源情况，自动指定的。用户可通过CLI程序taos，在TDengine的console里，执行如下命令：

```
1 SHOW MNODES;
```

来查看mnode列表，该列表将列出mnode所处的dnode的End Point和角色(master, slave, unsynced 或offline)。当集群中第一个数据节点启动时，该数据节点一定会运行一个mnode实例，否则该数据节点dnode无法正常工作，因为一个系统是必须有至少一个mnode的。如果numOfMnodes配置为2，启动第二个dnode时，该dnode也将运行一个mnode实例。

为保证mnode服务的高可用性，numOfMnodes必须设置为2或更大。因为mnode保存的元数据必须是强一致的，如果numOfMnodes大于2，复制参数quorum自动设为2，也就是说，至少要保证有两个副本写入数据成功，才通知客户端应用写入成功。

注意：一个TDengine高可用系统，无论是vnode还是mnode, 都必须配置多个副本。

10.7. 负载均衡

有三种情况，将触发负载均衡，而且都无需人工干预。

- 当一个新数据节点添加进集群时，系统将自动触发负载均衡，一些节点上的数据将被自动转移到新数据节点上，无需任何人工干预。
- 当一个数据节点从集群中移除时，系统将自动把该数据节点上的数据转移到其他数据节点，无需任何人工干预。
- 如果一个数据节点过热（数据量过大），系统将自动进行负载均衡，将该数据节点的一些vnode自动挪到其他节点。

当上述三种情况发生时，系统将启动各个数据节点的负载计算，从而决定如何挪动。

【提示】负载均衡由参数balance控制，它决定是否启动自动负载均衡。

10.8. 数据节点离线处理

如果一个数据节点离线，TDengine集群将自动检测到。有如下两种情况：

- 该数据节点离线超过一定时间（taos.cfg里配置参数offlineThreshold控制时长），系统将自动把该数据节点删除，产生系统报警信息，触发负载均衡流程。如果该被删除的数据节点重新上线时，它将无法加入集群，需要系统管理员重新将其添加进集群才会开始工作。
- 离线后，在offlineThreshold的时长内重新上线，系统将自动启动数据恢复流程，等数据完全恢复后，该节点将开始正常工作。

注意：如果一个虚拟节点组（包括mnode组）里所归属的每个数据节点都处于离线或unsynced状态，必须等该虚拟节点组里的所有数据节点都上线、都能交换状态信息后，才能选出Master，该虚拟节点组才能对外提供服务。比如整个集群有3个数据节点，副本数为3，如果3个数据节点都宕机，然后2个数据节点重启，是无法工作的，只有等3个数据节点都重启成功，才能对外服务。

10.9. Arbitrator的使用

如果副本数为偶数，当一个 vnode group 里一半或超过一半的 vnode 不工作时，是无法从中选出 master 的。同理，一半或超过一半的 mnode 不工作时，是无法选出 mnode 的 master 的，因为存在“split brain”问题。为解决这个问题，TDengine 引入了 Arbitrator 的概念。Arbitrator 模拟一个 vnode 或 mnode 在工作，但只简单的负责网络连接，不处理任何数据插入或访问。只要包含 Arbitrator 在内，超过半数的 vnode 或 mnode 工作，那么该 vnode group 或 mnode 组就可以正常的提供数据插入或查询服务。比如对于副本数为 2 的情形，如果一个节点 A 离线，但另外一个节点 B 正常，而且能连接到 Arbitrator，那么节点 B 就能正常工作。

总之，在目前版本下，TDengine 建议在双副本环境要配置 Arbitrator，以提升系统的可用性。

Arbitrator 的执行程序名为 tarbitrator。该程序对系统资源几乎没有要求，只需要保证有网络连接，找任何一台 Linux 服务器运行它即可。以下简要描述安装配置的步骤：

1. 请点击 安装包下载，在 TDengine Arbitrator Linux 一节中，选择合适的版本下载并安装。
2. 该应用的命令行参数 `-p` 可以指定其对外服务的端口号，缺省是 6042。
3. 修改每个 taosd 实例的配置文件，在 taos.cfg 里将参数 arbitrator 设置为 tarbitrator 程序所对应的 End Point。（如果该参数配置了，当副本数为偶数时，系统将自动连接配置的 Arbitrator。如果副本数为奇数，即使配置了 Arbitrator，系统也不会去建立连接。）
4. 在配置文件中配置了的 Arbitrator，会出现在 `SHOW DNODES;` 指令的返回结果中，对应的 role 列的值会是“arb”。

查看集群 Arbitrator 的状态【2.0.14.0 以后支持】

```
SHOW DNODES;
```

TDengine 企业版也会提供专门的 Arbitrator 安装包，例如（以 x64 平台为例）：

- TDengine-enterprise-arbitrator-2.x.x.x-Linux-x64.tar.gz

10.9.0.1. Arbitrator安装

1. 从涛思交付团队获得服务器的tar.gz安装包，如TDengine-enterprise-arbitrator-2.0.14.0-Linux-x64.tar.gz

2. 解压缩软件包

将软件包放置在当前用户可读写的任意目录下，然后执行下面的命令：

```
tar -xzvf TDengine-enterprise-arbitrator-xxxxxxxxx.tar.gz
```

其中xxxxxxxxx需要替换为实际版本的字符串。

3. 执行安装脚本

解压软件包之后，会在解压目录下看到以下文件(目录)：

install_arbi.sh：主安装脚本，用于安装服务端及客户端程序
bin：可执行程序
Init.d：支持程序
运行`install_arbi.sh`进行安装。

10.9.0.2. Arbitrator启动

安装成功后，用户可使用`systemctl`命令来启动Arbitrator的服务进程：

```
systemctl start tarbitratord
```

检查服务是否正常工作：

```
systemctl status tarbitratord
```

10.9.0.3. 数据节点的相关配置

集群内的所有数据节点，taos.cfg必须做以下配置：

```
arbitrator     <hostname of arbitrator>:6042
```

修改配置后须重启taosd服务，在taos shell里面执行 `show variables`查看arbitrator的配置信息。

最后，修改DNS服务器，或在各数据节点编辑/etc/hosts，配置arbitrator的域名解析：

```
<IP address of arbitrator>    <hostname of arbitrator>
```

以上步骤完成后，如集群与arbitrator配置无误，可在taos shell里面执行`SHOW DNODES;`查看arbitrator的状态。

11. TDengine的运营与运维

11.1. 容量规划

使用 TDengine 来搭建一个物联网大数据平台，计算资源、存储资源需要根据业务场景进行规划。下面分别讨论系统运行所需要的内存、CPU 以及硬盘空间。

11.1.1. 内存需求

每个 Database 可以创建固定数目的 vgroup，默认与 CPU 核数相同，可通过 maxVgroupsPerDb 配置；vgroup 中的每个副本会是一个 vnode；每个 vnode 会占用固定大小的内存（大小与数据库的配置参数 blocks 和 cache 有关)；每个 Table 会占用与标签总长度有关的内存；此外，系统会有一些固定的内存开销。因此，每个 DB 需要的系统内存可通过如下公式计算：

```
Database Memory Size = maxVgroupsPerDb * (blocks * cache + 10MB) + numOfTables * (tagSizePerTable + 0.5KB)
```

示例：假设是 4 核机器，cache 是缺省大小 16M, blocks 是缺省值 6，并且一个 DB 中有 10 万张表，标签总长度是 256 字节，则这个 DB 总的内存需求为：4 * (16 * 6 + 10) + 100000 * (0.25 + 0.5) / 1000 = 499M。

在实际的系统运维中，我们通常会更关心 TDengine 服务进程（taosd）会占用的内存量。

```
taosd 内存总量 = vnode 内存 + mnode 内存 + 查询内存
```

其中：

1. “vnode 内存”指的是集群中所有的 Database 存储分摊到当前 taosd 节点上所占用的内存资源。可以按上文“Database Memory Size”计算公式估算每个 DB 的内存占用量进行加总，再按集群中总共的 TDengine 节点数做平均（如果设置为多副本，则还需要乘以对应的副本倍数）。
2. “mnode 内存”指的是集群中管理节点所占用的资源。如果一个 taosd 节点上分布有 mnode 管理节点，则内存消耗还需要增加“0.2KB * 集群中数据表总数”。
3. “查询内存”指的是服务端处理查询请求时所需要占用的内存。单条查询语句至少会占用“0.2KB * 查询涉及的数据表总数”的内存量。

注意：以上内存估算方法，主要讲解了系统的“必须内存需求”，而不是“内存总数上限”。在实际运行的生产环境中，由于操作系统缓存、资源管理调度等方面的原因，内存规划应当在估算结果的基础上保留一定冗余，以维持系统状态和系统性能的稳定性。并且，生产环境通常会配置系统资源的监控工具，以便及时发现硬件资源的紧缺情况。

最后，如果内存充裕，可以考虑加大 Blocks 的配置，这样更多数据将保存在内存里，提高查询速度。

11.1.1.1. 客户端内存需求

客户端应用采用 taosc 客户端驱动连接服务端，会有内存需求的开销。

客户端的内存开销主要由写入过程中的 SQL 语句、表的元数据信息缓存、以及结构性开销构成。系统最大容纳的表数量为 N（每个通过超级表创建的表的 meta data 开销约 256 字节），最大并行写入线程数量 T，最大 SQL 语句长度 S（通常是 1 Mbytes）。由此可以进行客户端内存开销的估算（单位 MBytes）：

```
M = (T * S * 3 + (N / 4096) + 100)
```

举例如下：用户最大并发写入线程数 100，子表数总数 10,000,000，那么客户端的内存最低要求是：

```
100 * 3 + (10000000 / 4096) + 100 = 2741 (MBytes)
```

即配置 3 GBytes 内存是最低要求。

11.1.2. CPU 需求

CPU 的需求取决于如下两方面：

- **数据插入** TDengine 单核每秒能至少处理一万个插入请求。每个插入请求可以带多条记录，一次插入一条记录与插入 10 条记录，消耗的计算资源差别很小。因此每次插入，条数越大，插入效率越高。如果一个插入请求带 200 条以上记录，单核就能达到每秒插入 100 万条记录的速度。但对前端数据采集的要求越高，因为需要缓存记录，然后一批插入。
- **查询需求** TDengine 提供高效的查询，但是每个场景的查询差异很大，查询频次变化也很大，难以给出客观数字。需要用户针对自己的场景，写一些查询语句，才能确定。

因此仅对数据插入而言，CPU 是可以估算出来的，但查询所耗的计算资源无法估算。在实际运营过程中，不建议 CPU 使用率超过 50%，超过后，需要增加新的节点，以获得更多计算资源。

11.1.3. 存储需求

TDengine 相对于通用数据库，有超高的压缩比，在绝大多数场景下，TDengine 的压缩比不会低于 5 倍，有的场合，压缩比可达到 10 倍以上，取决于实际场景的数据特征。压缩前的原始数据大小可通过如下方式计算：

```
Raw DataSize = numOfTables * rowSizePerTable * rowsPerTable
```

示例：1000 万台智能电表，每台电表每 15 分钟采集一次数据，每次采集的数据 128 字节，那么一年的原始数据量是：10000000 * 128 * 24 * 60 / 15 * 365 = 44.8512T。TDengine大概需要消耗 44.851 / 5 = 8.97024T 空间。

用户可以通过参数 keep，设置数据在磁盘中的最大保存时长。为进一步减少存储成本，TDengine 还提供多级存储，最冷的数据可以存放在最廉价的存储介质上，应用的访问不用做任何调整，只是读取速度降低了。

为提高速度，可以配置多块硬盘，这样可以并发写入或读取数据。需要提醒的是，TDengine 采取多副本的方式提供数据的高可靠，因此不再需要采用昂贵的磁盘阵列。

11.1.4. 物理机或虚拟机台数

根据上面的内存、CPU、存储的预估，就可以知道整个系统需要多少核、多少内存、多少存储空间。如果数据副本数不为 1，总需求量需要再乘以副本数。

因为 TDengine 具有很好的水平扩展能力，根据总量，再根据单个物理机或虚拟机的资源，就可以轻松决定需要购置多少台物理机或虚拟机了。

立即计算 CPU、内存、存储，请参见：资源估算方法。

11.2. 容错和灾备

11.2.1. 容错

TDengine支持**WAL**（Write Ahead Log）机制，实现数据的容错能力，保证数据的高可用。

TDengine接收到应用的请求数据包时，先将请求的原始数据包写入数据库日志文件，等数据成功写入数据库数据文件后，再删除相应的WAL。这样保证了TDengine能够在断电等因素导致的服务重启时从数据库日志文件中恢复数据，避免数据的丢失。

涉及的系统配置参数有两个：

- walLevel：WAL级别，0：不写wal; 1：写wal, 但不执行fsync; 2：写wal, 而且执行fsync。
- fsync：当walLevel设置为2时，执行fsync的周期。设置为0，表示每次写入，立即执行fsync。

如果要100%的保证数据不丢失，需要将walLevel设置为2，fsync设置为0。这时写入速度将会下降。但如果应用侧启动的写数据的线程数达到一定的数量(超过50)，那么写入数据的性能也会很不错，只会比fsync设置为3000毫秒下降30%左右。

11.2.2. 灾备

TDengine的集群通过多个副本的机制，来提供系统的高可用性，实现灾备能力。

TDengine集群是由mnode负责管理的，为保证mnode的高可靠，可以配置多个mnode副本，副本数由系统配置参数numOfMnodes决定，为了支持高可靠，需要设置大于1。为保证元数据的强一致性，mnode副本之间通过同步方式进行数据复制，保证了元数据的强一致性。

TDengine集群中的时序数据的副本数是与数据库关联的，一个集群里可以有多个数据库，每个数据库可以配置不同的副本数。创建数据库时，通过参数replica 指定副本数。为了支持高可靠，需要设置副本数大于1。

TDengine集群的节点数必须大于等于副本数，否则创建表时将报错。

当TDengine集群中的节点部署在不同的物理机上，并设置多个副本数时，就实现了系统的高可靠性，无需再使用其他软件或工具。TDengine企业版还可以将副本部署在不同机房，从而实现异地容灾。

11.3. 服务端配置

TDengine系统后台服务由taosd提供，可以在配置文件taos.cfg里修改配置参数，以满足不同场景的需求。配置文件的缺省位置在/etc/taos目录，可以通过taosd命令行执行参数-c指定配置文件目录。比如taosd -c /home/user来指定配置文件位于/home/user这个目录。

另外可以使用 “-C” 显示当前服务器配置参数：

```
taosd -C
```

下面仅仅列出一些重要的配置参数，更多的参数请看配置文件里的说明。各个参数的详细介绍及作用请看前述章节，而且这些参数的缺省配置都是可以工作的，一般无需设置。**注意：配置文件参数修改后，需要重启*taosd*服务，或客户端应用才能生效。**

#	配置参数名称	内部	S|C	单位	含义	取值范围	缺省值	备注
1	firstEP		SC		taosd启动时，主动连接的集群中首个dnode的end point		localhost:6030	
2	secondEP	YES	SC		taosd启动时，如果firstEp连接不上，尝试连接集群中第二个dnode的end point		无	
3	fqdn		SC		数据节点的FQDN。如果习惯IP地址访问，可设置为该节点的IP地址。		缺省为操作系统配置的第一个hostname。	这个参数值的长度需要控制在 96 个字符以内。
4	serverPort		SC		taosd启动后，对外服务的端口号		6030	RESTful服务使用的端口号是在此基础上+11，即默认值为6041。
5	logDir		SC		日志文件目录，客户端和服务器的运行日志将写入该目录		/var/log/taos	
6	scriptDir	YES	S					
7	dataDir		S		数据文件目录，所有的数据文件都将写入该目录		/var/lib/taos	
8	arbitrator		S		系统中裁决器的end point		空	
9	numOfThreadsPerCore		SC		每个CPU核生成的队列消费者线程数		1.0	

					量		
10	ratioOfQueryThreads		S		设置查询线程的最大数量	0：表示只有1个查询线程；1：表示最大和CPU核数相等的查询线程；2：表示最大建立2倍CPU核数的查询线程。	1	该值可以为小数，即0.5表示最大建立CPU核数一半的查询线程。
11	numOfMnodes		S		系统中管理节点个数		3	
12	vnodeBak		S		删除vnode时是否备份vnode目录	0：否，1：是	1	
13	telemetryReporting		S		是否允许TDengine 采集和上报基本使用信息	0：不允许；1：允许	1	
14	balance		S		是否启动负载均衡	0，1	1	
15	balanceInterval	YES	S	秒	管理节点在正常运行状态下，检查负载均衡的时间间隔	1-30000	300	
16	role		S		dnode的可选角色	0：any（既可作为mnode，也可分配vnode）；1：mgmt（只能作为mnode，不能分配vnode）；2：dnode（不能作为mnode，只能分配vnode）	0	
17	maxTmerCtrl		SC	个	定时器个数	8-2048	512	
18	monitorInterval		S	秒	监控数据库记录系统参数（CPU/内存）的时间间隔	1-600	30	
19	offlineThreshold		S	秒	dnode离线阈值，超过该时间将导致dnode离线	5-7200000	86400*10（10天）	
20	rpcTimer		SC	毫秒	rpc重试时长	100-3000	300	
21	rpcMaxTime		SC	秒	rpc等待应答最大时长	100-7200	600	
22	statusInterval		S	秒	dnode向mnode报告状态间隔	1-10	1	
23	shellActivityTimer		SC	秒	shell客户端向mnode发送心跳间隔	1-120	3	
24	tableMetaKeepTimer		S	秒	表的元数据cache时长	1-8640000	7200	
25	minSlidingTime		S	毫秒	最小滑动窗口时长	10-1000000	10	支持us补值后，这个值就是1us了。

26	minIntervalTime		**S**	毫秒	时间窗口最小值	1-1000000	10	
27	stream		**S**		是否启用连续查询（流计算功能）	0：不允许；1：允许	1	
28	maxStreamCompDelay		**S**	毫秒	连续查询启动最大延迟	10-1000000000	20000	为避免多个stream同时执行占用太多系统资源，程序中对stream的执行时间人为增加了一些随机的延时。maxFirstStreamCompDelay 是stream第一次执行前最少要等待的时间。streamCompDelayRatio 是延迟时间的计算系数，它乘以查询的 interval 后为延迟时间基准。maxStreamCompDelay是延迟时间基准的上限。实际延迟时间为一个不超过延迟时间基准的随机值。stream某次计算失败后需要重试，retryStreamCompDelay是重试的等待时间基准。实际重试等待时间为不超过等待时间基准的随机值。
29	maxFirstStreamCompDelay		**S**	毫秒	第一次连续查询启动最大延迟	10-1000000000	10000	
30	retryStreamCompDelay		**S**	毫秒	连续查询重试等待间隔	10-1000000000	10	
31	streamCompDelayRatio		**S**		连续查询的延迟时间计算系数	0.1-0.9	0.1	
32	maxVgroupsPerDb		**S**		每个DB中 能够使用的最大vnode个数	0-8192		
33	maxTablesPerVnode		**S**		每个vnode中能够创建的最大表个数		1000000	
34	minTablesPerVnode	YES	**S**		每个vnode中必须创建的最小表个数		100	
35	tableIncStepPerVnode	YES	**S**		每个vnode中超过最小表数后递增步长		1000	
36	cache		**S**	MB	内存块的大小		16	
37	blocks		**S**		每个vnode（tsdb）中有多少cache大小的内存块。因此一个vnode的用的内存大小粗略为（cache * blocks）		6	
38	days		**S**	天	数据文件存储数据的时间跨度		10	
39	keep		**S**	天	数据保留的天数		3650	
40	minRows		**S**		文件块中记录的最小条数		100	
41	maxRows		**S**		文件块中记录的最大条数		4096	
42	quorum		**S**		异步写入成功所需应答之法定数	1-3	1	
43	comp		**S**		文件压缩标志位	0：关闭，1:一阶段压缩，2:两阶	2	

						段压缩		
44	walLevel		S		WAL级别	1：写wal, 但不执行fsync; 2：写wal, 而且执行fsync	1	
45	fsync		S	毫秒	当wal设置为2时，执行fsync的周期	最小为0，表示每次写入，立即执行fsync；最大为180000（三分钟）	3000	
46	replica		S		副本个数	1-3	1	
47	mqttHostName	YES	S		mqtt uri			mqtt://username:password@hostname:1883/taos/
48	mqttPort	YES	S		mqtt client name			1883
49	mqttTopic	YES	S					/test
50	compressMsgSize		S	bytes	客户端与服务器之间进行消息通讯过程中，对通讯的消息进行压缩的阈值。如果要压缩消息，建议设置为64330字节，即大于64330字节的消息体才进行压缩。	`0`表示对所有的消息均进行压缩 >0: 超过该值的消息才进行压缩 -1: 不压缩	-1	
51	maxSQLLength		C	bytes	单条SQL语句允许的最长限制	65480-1048576	65380	
52	maxNumOfOrderedRes		SC		支持超级表时间排序允许的最多记录数限制		10万	
53	timezone		SC		时区		从系统中动态获取当前的时区设置	
54	locale		SC		系统区位信息及编码格式		系统中动态获取，如果自动获取失败，需要用户在配置文件设置或通过API设置	
55	charset		SC		字符集编码		系统中动态获取，如果自动获取失败，需要用户在配置文件设置或通过API设置	
56	maxShellConns		S		一个dnode容许的连接数	10-50000000	5000	
57	maxConnections		S		一个数据库连接所容许的dnode连接数	1-100000	5000	实际测试下来，如果默认没有配，选 50 个 worker thread 会产生 Network unavailable

58	minimalLogDirGB		**SC**	GB	当日志文件夹的磁盘大小小于该值时，停止写日志		0.1	
59	minimalTmpDirGB		**SC**	GB	当日志文件夹的磁盘大小小于该值时，停止写临时文件		0.1	
60	minimalDataDirGB		**S**	GB	当日志文件夹的磁盘大小小于该值时，停止写时序数据		0.1	
61	mnodeEqualVnodeNum		**S**		一个mnode等同于vnode消耗的个数		4	
62	http		**S**		服务器内部的http服务开关。	0：关闭http服务，1：激活http服务。	1	
63	mqtt	YES	**S**		服务器内部的mqtt服务开关。	0：关闭mqtt服务，1：激活mqtt服务。	0	
64	monitor		**S**		服务器内部的系统监控开关。监控主要负责收集物理节点的负载状况，包括CPU、内存、硬盘、网络带宽、HTTP请求量的监控记录，记录信息存储在LOG库中。	0：关闭监控服务，1：激活监控服务。	0	
65	httpEnableRecordSql		**S**		内部使用，记录通过RESTFul接口，产生的SQL调用		0	生成的文件（httpnote.0/httpnote.1），与服务端日志所在目录相同。
66	httpMaxThreads		**S**		RESTFul接口的线程数		2	
67	telegrafUseFieldNum	YES						
68	restfulRowLimit		**S**		RESTFul接口单次返回的记录条数		10240	最大10,000,000
69	numOfLogLines		**SC**		单个日志文件允许的最大行数。		10,000,000	
70	asyncLog		**SC**		日志写入模式	0：同步、1：异步	1	
71	logKeepDays		**SC**	天	日志文件的最长保存时间		0	大于0时，日志文件会被重命名为taosdlog.xxx，其中xxx为日志文件最后修改的时间戳。
72	debugFlag		**SC**		运行日志开关	131（输出错误和警告日志），135（输出错误、警告和调试日志），143（输出错误、警告、调试	131或135（不同模块有不同的默认值）	

			和跟踪日志）			
73	mDebugFlag	**S**	管理模块的日志开关	同上	135	
74	dDebugFlag	**SC**	dnode模块的日志开关	同上	135	
75	sDebugFlag	**SC**	sync模块的日志开关	同上	135	
76	wDebugFlag	**SC**	wal模块的日志开关	同上	135	
77	sdbDebugFlag	**SC**	sdb模块的日志开关	同上	135	
78	rpcDebugFlag	**SC**	rpc模块的日志开关	同上		
79	tmrDebugFlag	**SC**	定时器模块的日志开关	同上		
80	cDebugFlag	**C**	client模块的日志开关	同上		
81	jniDebugFlag	**C**	jni模块的日志开关	同上		
82	odbcDebugFlag	**C**	odbc模块的日志开关	同上		
83	uDebugFlag	**SC**	共用功能模块的日志开关	同上		
84	httpDebugFlag	**S**	http模块的日志开关	同上		
85	mqttDebugFlag	**S**	mqtt模块的日志开关	同上		
86	monitorDebugFlag	**S**	监控模块的日志开关	同上		
87	qDebugFlag	**SC**	查询模块的日志开关	同上		
88	vDebugFlag	**SC**	vnode模块的日志开关	同上		
89	tsdbDebugFlag	**S**	TSDB模块的日志开关	同上		
90	cqDebugFlag	**SC**	连续查询模块的日志开关	同上		
91	tscEnableRecordSql	**C**	是否记录客户端sql语句到文件	0：否，1：是	0	生成的文件（tscnote-xxxx.0/tscnote-xxx.1，xxxx是pid），与客户端日志所在目录相同。
92	enableCoreFile	**SC**	是否开启服务crash时生成core文件	0：否，1：是	1	不同的启动方式，生成core文件的目录如下：1、systemctl start taosd启动：生成的core在根目录下；2、手动启动，就在taosd执行目录下。
93	gitinfo	YES **SC**		1		
94	gitinfoofInternal	YES **SC**		2		
95	Buildinfo	YES **SC**		3		
96	version	YES **SC**		4		

97							
98	maxBinaryDisplayWidth		**C**		Taos shell中 binary 和 nchar字段的显示宽度上限，超过此限制的部分将被隐藏	5 -	30	实际上限按以下规则计算：如果字段值的长度大于maxBinaryDisplayWidth，则显示上限为 **字段名长度** 和 **maxBinaryDisplayWidth** 的较大者。否则，上限为 **字段名长度** 和 **字段值长度** 的较大者。可在 shell 中通过命令 set max_binary_display_width nn动态修改此选项
99	queryBufferSize		**S**	MB	为所有并发查询占用保留的内存大小。			计算规则可以根据实际应用可能的最大并发数和表的数字相乘，再乘 170 。（2.0.15 以前的版本中，此参数的单位是字节）
100	ratioOfQueryCores		**S**		设置查询线程的最大数量。			最小值0 表示只有1个查询线程；最大值2表示最大建立2倍CPU核数的查询线程。默认为1，表示最大和CPU核数相等的查询线程。该值可以为小数，即0.5表示最大建立CPU核数一半的查询线程。
101	update		**S**		允许更新已存在的数据行	0 ∣ 1	0	从 2.0.8.0 版本开始
102	cacheLast		**S**		是否在内存中缓存子表的最近数据	0：关闭；1：缓存子表最近一行数据；2：缓存子表每一列的最近的非NULL值；3：同时打开缓存最近行和列功能。	0	2.1.2.0 版本之前、2.0.20.7 版本之前在 taos.cfg 文件中不支持此参数。
103	numOfCommitThreads	YES	**S**		设置写入线程的最大数量			
104	maxWildCardsLength		**C**	bytes	设定 LIKE 算子的通配符字符串允许的最大长度	0-16384	100	2.1.6.1 版本新增。

注意：对于端口，TDengine会使用从serverPort起13个连续的TCP和UDP端口号，请务必在防火墙打开。因此如果是缺省配置，需要打开从6030到6042共13个端口，而且必须TCP和UDP都打开。（详细的端口情况请参见 TDengine 2.0 端口说明）

不同应用场景的数据往往具有不同的数据特征，比如保留天数、副本数、采集频次、记录大小、采集点的数量、压缩等都可完全不同。为获得在存储上的最高效率，TDengine提供如下存储相关的系统配置参数（既可以作为 create database 指令的参数，也可以写在 taos.cfg 配置文件中用来设定创建新数据库时所采用的默认值）：

#	配置参数名称	单位	含义	取值范围	缺省值
1	days	天	一个数据文件存储数据的时间跨度		10
2	keep	天	（可通过 alter database 修改，在企业版中可以分别设置多级存储的不同级别介质的保留天数，详情请参照 SQL 语句的使用说明。）数据库中数据保留的天数。	3650	
3	cache	MB	内存块的大小		16
4	blocks		（可通过 alter database 修改）每个 VNODE（TSDB）中有多少个 cache 大小的内存块。因此一个 VNODE 使用的内存大小粗略为（cache * blocks）。		6
5	quorum		（可通过 alter database 修改）多副本环境下指令执行的确认数要求	1-2	1
6	minRows		文件块中记录的最小条数		100
7	maxRows		文件块中记录的最大条数		4096
8	comp		（可通过 alter database 修改）文件压缩标志位	0：关闭，1:一阶段压缩，2:两阶段压缩	2
9	walLevel		（作为 database 的参数时名为 wal；在 taos.cfg 中作为参数时需要写作 walLevel）WAL级别	1：写wal，但不执行 fsync；2：写wal, 而且执行fsync	1
10	fsync	毫秒	当wal设置为2时，执行fsync的周期。设置为0，表示每次写入，立即执行fsync。		3000
11	replica		（可通过 alter database 修改）副本个数	1-3	1
12	precision		时间戳精度标识（2.1.2.0 版本之前、2.0.20.7 版本之前在 taos.cfg 文件中不支持此参数。）（从 2.1.5.0 版本开始，新增对纳秒时间精度的支持）	ms 表示毫秒，us 表示微秒，ns 表示纳秒	ms
13	update		是否允许更新	0：不允许；1：允许	0
14	cacheLast		（可通过 alter database 修改）是否在内存中缓存子表的最近数据（从 2.1.2.0 版本开始此参数支持 0 ~ 3 的取值范围，在此之前取值只能是 [0, 1]；而 2.0.11.0 之前的版本在 SQL 指令中不支持此参数。）（2.1.2.0 版本之前、2.0.20.7 版本之前在 taos.cfg 文件中不支持此参数。）	0：关闭；1：缓存子表最近一行数据；2：缓存子表每一列的最近的非NULL值；3：同时打开缓存最近行和列功能	0

对于一个应用场景，可能有多种数据特征的数据并存，最佳的设计是将具有相同数据特征的表放在一个库里，这样一个应用有多个库，而每个库可以配置不同的存储参数，从而保证系统有最优的性能。TDengine允许应用在创建库时指定上述存储参数，如果指定，该参数就将覆盖对应的系统配置参数。举例，有下述SQL：

```
CREATE DATABASE demo DAYS 10 CACHE 32 BLOCKS 8 REPLICA 3 UPDATE 1;
```

该SQL创建了一个库demo, 每个数据文件存储10天数据，内存块为32兆字节，每个VNODE占用8个内存块，副本数为3，允许更新，而其他参数与系统配置完全一致。

一个数据库创建成功后，仅部分参数可以修改并实时生效，其余参数不能修改：

参数名	能否修改	范围	修改语法示例
name			
create time			
ntables			
vgroups			
replica	**YES**	在线dnode数目为1：1-1；2：1-2；>=3：1-3	ALTER DATABASE REPLICA *n*
quorum	**YES**	1-2	ALTER DATABASE QUORUM *n*
days			
keep	**YES**	days-365000	ALTER DATABASE KEEP *n*
cache			
blocks	**YES**	3-1000	ALTER DATABASE BLOCKS *n*
minrows			
maxrows			
wal			
fsync			
comp	**YES**	0-2	ALTER DATABASE COMP *n*
precision			
status			
update			
cachelast	**YES**	0 ∣ 1 ∣ 2 ∣ 3	ALTER DATABASE CACHELAST *n*

说明：在 2.1.3.0 版本之前，通过 ALTER DATABASE 语句修改这些参数后，需要重启服务器才能生效。

TDengine集群中加入一个新的dnode时，涉及集群相关的一些参数必须与已有集群的配置相同，否则不能成功加入到集群中。会进行校验的参数如下：

- numOfMnodes：系统中管理节点个数。默认值：3。（2.0 版本从 2.0.20.11 开始、2.1 及以上版本从 2.1.6.0 开始，numOfMnodes 默认值改为 1。）
- mnodeEqualVnodeNum: 一个mnode等同于vnode消耗的个数。默认值：4。
- offlineThreshold: dnode离线阈值，超过该时间将导致该dnode从集群中删除。单位为秒，默认值：

86400*10（即10天）。
- statusInterval: dnode向mnode报告状态时长。单位为秒，默认值：1。
- maxTablesPerVnode: 每个vnode中能够创建的最大表个数。默认值：1000000。
- maxVgroupsPerDb: 每个数据库中能够使用的最大vgroup个数。
- arbitrator: 系统中裁决器的end point，缺省为空。
- timezone、locale、charset 的配置见客户端配置。（2.0.20.0 及以上的版本里，集群中加入新节点已不要求 locale 和 charset 参数取值一致）
- balance：是否启用负载均衡。0：否，1：是。默认值：1。
- flowctrl：是否启用非阻塞流控。0：否，1：是。默认值：1。
- slaveQuery：是否启用 slave vnode 参与查询。0：否，1：是。默认值：1。
- adjustMaster：是否启用 vnode master 负载均衡。0：否，1：是。默认值：1。

为方便调试，可通过SQL语句临时调整每个dnode的日志配置，系统重启后会失效：

```
ALTER DNODE <dnode_id> <config>
```

- dnode_id: 可以通过SQL语句"SHOW DNODES"命令获取
- config: 要调整的日志参数，在如下列表中取值

> resetlog 截断旧日志文件，创建一个新日志文件
> debugFlag ＜131 | 135 | 143＞设置debugFlag为131、135或者143

例如：

```
    alter dnode 1 debugFlag 135;
```

11.4. 客户端及应用驱动配置

TDengine系统的前台交互客户端应用程序为taos，以及应用驱动，它与taosd共享同一个配置文件taos.cfg。运行taos时，使用参数-c指定配置文件目录，如taos -c /home/cfg，表示使用/home/cfg/目录下的taos.cfg配置文件中的参数，缺省目录是/etc/taos。更多taos的使用方法请见帮助信息 `taos --help`。本节主要说明 taos 客户端应用在配置文件 taos.cfg 文件中使用到的参数。

2.0.10.0 之后版本支持命令行以下参数显示当前客户端参数的配置

```
taos -C  或  taos --dump-config
```

客户端及应用驱动配置参数列表及解释

- firstEp: taos启动时，主动连接的集群中第一个taosd实例的end point, 缺省值为 localhost:6030。
- secondEp: taos 启动时，如果 firstEp 连不上，将尝试连接 secondEp。
- locale：系统区位信息及编码格式。

  默认值：系统中动态获取，如果自动获取失败，需要用户在配置文件设置或通过API设置。

  TDengine为存储中文、日文、韩文等非ASCII编码的宽字符，提供一种专门的字段类型nchar。写入nchar字段的数据将统一采用UCS4-LE格式进行编码并发送到服务器。需要注意的是，编码正确性是客户端来保证。因此，如果用户想要正常使用nchar字段来存储诸如中文、日文、韩文等非ASCII字符，需要正确设置客户端的编码格式。

客户端的输入的字符均采用操作系统当前默认的编码格式，在Linux系统上多为UTF-8，部分中文系统编码则可能是GB18030或GBK等。在docker环境中默认的编码是POSIX。在中文版Windows系统中，编码则是CP936。客户端需要确保正确设置自己所使用的字符集，即客户端运行的操作系统当前编码字符集，才能保证nchar中的数据正确转换为UCS4-LE编码格式。

在 Linux 中 locale 的命名规则为: <语言>_<地区>.<字符集编码> 如：zh_CN.UTF-8，zh代表中文，CN代表大陆地区，UTF-8表示字符集。字符集编码为客户端正确解析本地字符串提供编码转换的说明。Linux系统与 Mac OSX 系统可以通过设置locale来确定系统的字符编码，由于Windows使用的locale中不是POSIX标准的locale格式，因此在Windows下需要采用另一个配置参数charset来指定字符编码。在Linux 系统中也可以使用charset来指定字符编码。

- charset：字符集编码。

  默认值：系统中动态获取，如果自动获取失败，需要用户在配置文件设置或通过API设置。

  如果配置文件中不设置charset，在Linux系统中，taos在启动时候，自动读取系统当前的locale信息，并从locale信息中解析提取charset编码格式。如果自动读取locale信息失败，则尝试读取charset配置，如果读取charset配置也失败，则中断启动过程。

  在Linux系统中，locale信息包含了字符编码信息，因此正确设置了Linux系统locale以后可以不用再单独设置charset。例如：

  ```
  locale zh_CN.UTF-8
  ```

  在Windows系统中，无法从locale获取系统当前编码。如果无法从配置文件中读取字符串编码信息，taos默认设置为字符编码为CP936。其等效在配置文件中添加如下配置：

  ```
  charset CP936
  ```

  如果需要调整字符编码，请查阅当前操作系统使用的编码，并在配置文件中正确设置。

  在Linux系统中，如果用户同时设置了locale和字符集编码charset，并且locale和charset的不一致，后设置的值将覆盖前面设置的值。

  ```
  locale zh_CN.UTF-8
  charset GBK
  ```

  则charset的有效值是GBK。

  ```
  charset GBK
  locale zh_CN.UTF-8
  ```

  charset的有效值是UTF-8。

  日志的配置参数，与server 的配置参数完全一样。

- timezone

  默认值：从系统中动态获取当前的时区设置

  客户端运行系统所在的时区。为应对多时区的数据写入和查询问题，TDengine 采用 Unix 时间戳(Unix Timestamp)来记录和存储时间戳。Unix 时间戳的特点决定了任一时刻不论在任何时区，产生的时间戳均一致。需要注意的是，Unix时间戳是在客户端完成转换和记录。为了确保客户端其他形式的时间转换为正确的 Unix 时间戳，需要设置正确的时区。

在Linux系统中，客户端会自动读取系统设置的时区信息。用户也可以采用多种方式在配置文件设置时区。例如：

```
timezone UTC-8
timezone GMT-8
timezone Asia/Shanghai
```

均是合法的设置东八区时区的格式。

时区的设置对于查询和写入SQL语句中非Unix时间戳的内容（时间戳字符串、关键词now的解析）产生影响。例如：

```
SELECT count(*) FROM table_name WHERE TS<'2019-04-11 12:01:08';
```

在东八区，SQL语句等效于

```
SELECT count(*) FROM table_name WHERE TS<1554955268000;
```

在UTC时区，SQL语句等效于

```
SELECT count(*) FROM table_name WHERE TS<1554984068000;
```

为了避免使用字符串时间格式带来的不确定性，也可以直接使用Unix时间戳。此外，还可以在SQL语句中使用带有时区的时间戳字符串，例如：RFC3339格式的时间戳字符串，2013-04-12T15:52:01.123+08:00或者ISO-8601格式时间戳字符串2013-04-12T15:52:01.123+0800。上述两个字符串转化为Unix时间戳不受系统所在时区的影响。

启动taos时，也可以从命令行指定一个taosd实例的end point，否则就从taos.cfg读取。

- maxBinaryDisplayWidth

  Shell中 binary 和 nchar 字段的显示宽度上限，超过此限制的部分将被隐藏。默认值：30。可在 taos shell 中通过命令 set max_binary_display_width nn 动态修改此选项。

11.5. 用户管理

系统管理员可以在CLI界面里添加、删除用户，也可以修改密码。CLI里SQL语法如下：

```
CREATE USER <user_name> PASS <'password'>;
```

创建用户，并指定用户名和密码，密码需要用单引号引起来，单引号为英文半角

```
DROP USER <user_name>;
```

删除用户，限root用户使用

```
ALTER USER <user_name> PASS <'password'>;
```

修改用户密码，为避免被转换为小写，密码需要用单引号引用，单引号为英文半角

```
ALTER USER <user_name> PRIVILEGE <write|read>;
```

修改用户权限为：write 或 read，不需要添加单引号

说明：系统内共有 super/write/read 三种权限级别，但目前不允许通过 alter 指令把 super 权限赋予用户。

```
SHOW USERS;
```

显示所有用户

注意：SQL 语法中，< >表示需要用户输入的部分，但请不要输入< >本身。

11.6. 数据导入

TDengine提供多种方便的数据导入功能，一种按脚本文件导入，一种按数据文件导入，一种是taosdump工具导入本身导出的文件。

按脚本文件导入

TDengine的shell支持source filename命令，用于批量运行文件中的SQL语句。用户可将建库、建表、写数据等SQL命令写在同一个文件中，每条命令单独一行，在shell中运行source命令，即可按顺序批量运行文件中的SQL语句。以'#'开头的SQL语句被认为是注释，shell将自动忽略。

按数据文件导入

TDengine也支持在shell对已存在的表从CSV文件中进行数据导入。CSV文件只属于一张表且CSV文件中的数据格式需与要导入表的结构相同，在导入的时候，其语法如下：

```
insert into tb1 file 'path/data.csv';
```

注意：如果CSV文件首行存在描述信息，请手动删除后再导入。如某列为空，填NULL，无引号。

例如，现在存在一个子表d1001, 其表结构如下：

```
taos> DESCRIBE d1001
             Field              |        Type         |   Length    |    Note    |
================================================================================
 ts                             | TIMESTAMP           |           8 |            |
 current                        | FLOAT               |           4 |            |
 voltage                        | INT                 |           4 |            |
 phase                          | FLOAT               |           4 |            |
 location                       | BINARY              |          64 | TAG        |
 groupid                        | INT                 |           4 | TAG        |
```

要导入的data.csv的格式如下：

```
'2018-10-04 06:38:05.000',10.30000,219,0.31000
'2018-10-05 06:38:15.000',12.60000,218,0.33000
'2018-10-06 06:38:16.800',13.30000,221,0.32000
'2018-10-07 06:38:05.000',13.30000,219,0.33000
'2018-10-08 06:38:05.000',14.30000,219,0.34000
'2018-10-09 06:38:05.000',15.30000,219,0.35000
'2018-10-10 06:38:05.000',16.30000,219,0.31000
'2018-10-11 06:38:05.000',17.30000,219,0.32000
'2018-10-12 06:38:05.000',18.30000,219,0.31000
```

那么可以用如下命令导入数据：

```
taos> insert into d1001 file '~/data.csv';
Query OK, 9 row(s) affected (0.004763s)
```

taosdump工具导入

TDengine提供了方便的数据库导入导出工具taosdump。用户可以将taosdump从一个系统导出的数据，导入到其他系统中。具体使用方法，请参见博客：TDengine DUMP工具使用指南。

11.7. 数据导出

为方便数据导出，TDengine提供了两种导出方式，分别是按表导出和用taosdump导出。

按表导出CSV文件

如果用户需要导出一个表或一个STable中的数据，可在taos shell中运行：

```
select * from <tb_name> >> data.csv;
```

这样，表tb_name中的数据就会按照CSV格式导出到文件data.csv中。

用taosdump导出数据

利用taosdump，用户可以根据需要选择导出所有数据库、一个数据库或者数据库中的一张表，所有数据或一时间段的数据，甚至仅仅表的定义。

具体使用方法，请参见博客：TDengine DUMP工具使用指南。

11.8. 系统连接、任务查询管理

系统管理员可以从CLI查询系统的连接、正在进行的查询、流式计算，并且可以关闭连接、停止正在进行的查询和流式计算。CLI里SQL语法如下：

```
SHOW CONNECTIONS;
```

显示数据库的连接，其中一列显示ip:port, 为连接的IP地址和端口号。

```
KILL CONNECTION <connection-id>;
```

强制关闭数据库连接，其中的connection-id是SHOW CONNECTIONS中显示的第一列的数字。

```
SHOW QUERIES;
```

显示数据查询，其中第一列显示的以冒号隔开的两个数字为query-id，为发起该query应用连接的connection-id和查询次数。

```
KILL QUERY <query-id>;
```

强制关闭数据查询，其中query-id是SHOW QUERIES中显示的 connection-id:query-no字串，如“105:2”，拷贝粘贴即可。

```
SHOW STREAMS;
```

显示流式计算，其中第一列显示的以冒号隔开的两个数字为stream-id, 为启动该stream应用连接的connection-id和发起stream的次数。

```
KILL STREAM <stream-id>;
```

强制关闭流式计算，其中的中stream-id是SHOW STREAMS中显示的connection-id:stream-no字串，如103:2，拷贝粘贴即可。

11.9. 系统监控

TDengine启动后，会自动创建一个监测数据库log，并自动将服务器的CPU、内存、硬盘空间、带宽、请求数、磁盘读写速度、慢查询等信息定时写入该数据库。TDengine还将重要的系统操作（比如登录、创建、删除数据库等）日志以及各种错误报警信息记录下来存放在log库里。系统管理员可以从CLI直接查看这个数据库，也可以在WEB通过图形化界面查看这些监测信息。

这些监测信息的采集缺省是打开的，但可以修改配置文件里的选项enableMonitor将其关闭或打开。

11.10. 性能优化

因数据行 update、表删除、数据过期等原因，TDengine 的磁盘存储文件有可能出现数据碎片，影响查询操作的性能表现。从 2.1.3.0 版本开始，新增 SQL 指令 COMPACT 来启动碎片重整过程：

```
COMPACT VNODES IN (vg_id1, vg_id2, ...)
```

COMPACT 命令对指定的一个或多个 VGroup 启动碎片重整，系统会通过任务队列尽快安排重整操作的具体执行。COMPACT 指令所需的 VGroup id，可以通过 `SHOW VGROUPS;` 指令的输出结果获取；而且在 `SHOW VGROUPS;` 中会有一个 compacting 列，值为 2 时表示对应的 VGroup 处于排队等待进行重整的状态，值为 1 时表示正在进行碎片重整，为 0 时则表示并没有处于重整状态（未要求进行重整或已经完成重整）。

需要注意的是，碎片重整操作会大幅消耗磁盘 I/O。因此在重整进行期间，有可能会影响节点的写入和查询性能，甚至在极端情况下导致短时间的阻写。

11.11. 文件目录结构

安装TDengine后，默认会在操作系统中生成下列目录或文件：

目录/文件	说明
/usr/local/taos/bin	TDengine可执行文件目录。其中的执行文件都会软链接到/usr/bin目录下。
/usr/local/taos/connector	TDengine各种连接器目录。
/usr/local/taos/driver	TDengine动态链接库目录。会软链接到/usr/lib目录下。
/usr/local/taos/examples	TDengine各种语言应用示例目录。
/usr/local/taos/include	TDengine对外提供的C语言接口的头文件。
/etc/taos/taos.cfg	TDengine默认[配置文件]
/var/lib/taos	TDengine默认数据文件目录。可通过[配置文件]修改位置。
/var/log/taos	TDengine默认日志文件目录。可通过[配置文件]修改位置。

可执行文件

TDengine的所有可执行文件默认存放在 */usr/local/taos/bin* 目录下。其中包括：

- *taosd*：TDengine服务端可执行文件
- *taos*：TDengine Shell可执行文件
- *taosdump*：数据导入导出工具
- *taosdemo*：TDengine测试工具
- remove.sh：卸载TDengine的脚本，请谨慎执行，链接到/usr/bin目录下的**rmtaos**命令。会删除TDengine的安装目录/usr/local/taos，但会保留/etc/taos、/var/lib/taos、/var/log/taos。

您可以通过修改系统配置文件taos.cfg来配置不同的数据目录和日志目录。

11.12. TDengine 的启动、停止、卸载

TDengine 使用 Linux 系统的 systemd/systemctl/service 来管理系统的启动和、停止、重启操作。TDengine 的服务进程是 taosd，默认情况下 TDengine 在系统启动后将自动启动。DBA 可以通过 systemd/systemctl/service 手动操作停止、启动、重新启动服务。

以 systemctl 为例，命令如下：

- 启动服务进程：`systemctl start taosd`
- 停止服务进程：`systemctl stop taosd`
- 重启服务进程：`systemctl restart taosd`
- 查看服务状态：`systemctl status taosd`

如果服务进程处于活动状态，则 status 指令会显示如下的相关信息：

```
......

Active: active (running)

......
```

如果后台服务进程处于停止状态，则 status 指令会显示如下的相关信息：

```
......

Active: inactive (dead)

......
```

卸载 TDengine，只需要执行如下命令：

```
rmtaos
```

警告：执行该命令后，TDengine 程序将被完全删除，务必谨慎使用。

11.13. TDengine参数限制与保留关键字

名称命名规则

1. 合法字符：英文字符、数字和下划线
2. 允许英文字符或下划线开头，不允许以数字开头
3. 不区分大小写

密码合法字符集

`[a-zA-Z0-9!?$%^&*()_–+={[}]:;@~#|<,>.?/]`

去掉了 ‘“`\ (单双引号、撇号、反斜杠、空格)

- 数据库名：不能包含“.”以及特殊字符，不能超过 32 个字符
- 表名：不能包含“.”以及特殊字符，与所属数据库名一起，不能超过 192 个字符，每行数据最大长度 16k 个字符
- 表的列名：不能包含特殊字符，不能超过 64 个字符
- 数据库名、表名、列名，都不能以数字开头，合法的可用字符集是“英文字符、数字和下划线”
- 表的列数：不能超过 1024 列，最少需要 2 列，第一列必须是时间戳
- 记录的最大长度：包括时间戳 8 byte，不能超过 16KB（每个 BINARY/NCHAR 类型的列还会额外占用 2 个

byte 的存储位置）

- 单条 SQL 语句默认最大字符串长度：65480 byte，但可通过系统配置参数 maxSQLLength 修改，最长可配置为 1048576 byte
- 数据库副本数：不能超过 3
- 用户名：不能超过 23 个 byte
- 用户密码：不能超过 15 个 byte
- 标签(Tags)数量：不能超过 128 个，可以 0 个
- 标签的总长度：不能超过 16K byte
- 记录条数：仅受存储空间限制
- 表的个数：仅受节点个数限制
- 库的个数：仅受节点个数限制
- 单个库上虚拟节点个数：不能超过 64 个
- 库的数目，超级表的数目、表的数目，系统不做限制，仅受系统资源限制
- SELECT 语句的查询结果，最多允许返回 1024 列（语句中的函数调用可能也会占用一些列空间），超限时需要显式指定较少的返回数据列，以避免语句执行报错。

目前 TDengine 有将近 200 个内部保留关键字，这些关键字无论大小写均不可以用作库名、表名、STable 名、数据列名及标签列名等。这些关键字列表如下：

关键字列表

ABORT	CREATE	IGNORE	NULL	STAR
ACCOUNT	CTIME	IMMEDIATE	OF	STATE
ACCOUNTS	DATABASE	IMPORT	OFFSET	STATEMENT
ADD	DATABASES	IN	OR	STATE_WINDOW
AFTER	DAYS	INITIALLY	ORDER	STORAGE
ALL	DBS	INSERT	PARTITIONS	STREAM
ALTER	DEFERRED	INSTEAD	PASS	STREAMS
AND	DELIMITERS	INT	PLUS	STRING
AS	DESC	INTEGER	PPS	SYNCDB
ASC	DESCRIBE	INTERVAL	PRECISION	TABLE
ATTACH	DETACH	INTO	PREV	TABLES
BEFORE	DISTINCT	IS	PRIVILEGE	TAG
BEGIN	DIVIDE	ISNULL	QTIME	TAGS
BETWEEN	DNODE	JOIN	QUERIES	TBNAME
BIGINT	DNODES	KEEP	QUERY	TIMES
BINARY	DOT	KEY	QUORUM	TIMESTAMP
BITAND	DOUBLE	KILL	RAISE	TINYINT
BITNOT	DROP	LE	REM	TOPIC
BITOR	EACH	LIKE	REPLACE	TOPICS

BLOCKS	END	LIMIT	REPLICA	TRIGGER
BOOL	EQ	LINEAR	RESET	TSERIES
BY	EXISTS	LOCAL	RESTRICT	UMINUS
CACHE	EXPLAIN	LP	ROW	UNION
CACHELAST	FAIL	LSHIFT	RP	UNSIGNED
CASCADE	FILE	LT	RSHIFT	UPDATE
CHANGE	FILL	MATCH	SCORES	UPLUS
CLUSTER	FLOAT	MAXROWS	SELECT	USE
COLON	FOR	MINROWS	SEMI	USER
COLUMN	FROM	MINUS	SESSION	USERS
COMMA	FSYNC	MNODES	SET	USING
COMP	GE	MODIFY	SHOW	VALUES
COMPACT	GLOB	MODULES	SLASH	VARIABLE
CONCAT	GRANTS	NCHAR	SLIDING	VARIABLES
CONFLICT	GROUP	NE	SLIMIT	VGROUPS
CONNECTION	GT	NONE	SMALLINT	VIEW
CONNECTIONS	HAVING	NOT	SOFFSET	VNODES
CONNS	ID	NOTNULL	STABLE	WAL
COPY	IF	NOW	STABLES	WHERE

11.14. 诊断及其他

11.14.0.1. 网络连接诊断

当出现客户端应用无法访问服务端时，需要确认客户端与服务端之间网络的各端口连通情况，以便有针对性地排除故障。

目前网络连接诊断支持在：Linux 与 Linux，Linux 与 Windows 之间进行诊断测试。

诊断步骤：

1. 如拟诊断的端口范围与服务器 taosd 实例的端口范围相同，须先停掉 taosd 实例
2. 服务端命令行输入：`taos -n server -P <port>` 以服务端身份启动对端口 port 为基准端口的监听
3. 客户端命令行输入：`taos -n client -h <fqdn of server> -P <port>` 以客户端身份启动对指定的服务器、指定的端口发送测试包

服务端运行正常的话会输出以下信息：

```
# taos -n server -P 6000
12/21 14:50:13.522509 0x7f536f455200 UTL work as server, host:172.27.0.7 startPort:6000 endPort:6011 pkgLen:1000

12/21 14:50:13.522659 0x7f5352242700 UTL TCP server at port:6000 is listening
12/21 14:50:13.522727 0x7f5351240700 UTL TCP server at port:6001 is listening
...
...
...
12/21 14:50:13.523954 0x7f5342fed700 UTL TCP server at port:6011 is listening
12/21 14:50:13.523989 0x7f53437ee700 UTL UDP server at port:6010 is listening
12/21 14:50:13.524019 0x7f53427ec700 UTL UDP server at port:6011 is listening
12/21 14:50:22.192849 0x7f5352242700 UTL TCP: read:1000 bytes from 172.27.0.8 at 6000
12/21 14:50:22.192993 0x7f5352242700 UTL TCP: write:1000 bytes to 172.27.0.8 at 6000
12/21 14:50:22.237082 0x7f5351a41700 UTL UDP: recv:1000 bytes from 172.27.0.8 at 6000
12/21 14:50:22.237203 0x7f5351a41700 UTL UDP: send:1000 bytes to 172.27.0.8 at 6000
12/21 14:50:22.237450 0x7f5351240700 UTL TCP: read:1000 bytes from 172.27.0.8 at 6001
12/21 14:50:22.237576 0x7f5351240700 UTL TCP: write:1000 bytes to 172.27.0.8 at 6001
12/21 14:50:22.281038 0x7f5350a3f700 UTL UDP: recv:1000 bytes from 172.27.0.8 at 6001
12/21 14:50:22.281141 0x7f5350a3f700 UTL UDP: send:1000 bytes to 172.27.0.8 at 6001
...
...
...
12/21 14:50:22.677443 0x7f5342fed700 UTL TCP: read:1000 bytes from 172.27.0.8 at 6011
12/21 14:50:22.677576 0x7f5342fed700 UTL TCP: write:1000 bytes to 172.27.0.8 at 6011
12/21 14:50:22.721144 0x7f53427ec700 UTL UDP: recv:1000 bytes from 172.27.0.8 at 6011
12/21 14:50:22.721261 0x7f53427ec700 UTL UDP: send:1000 bytes to 172.27.0.8 at 6011
```

客户端运行正常会输出以下信息：

```
# taos -n client -h 172.27.0.7 -P 6000
12/21 14:50:22.192434 0x7fc95d859200 UTL work as client, host:172.27.0.7 startPort:6000 endPort:6011 pkgLen:1000

12/21 14:50:22.192472 0x7fc95d859200 UTL server ip:172.27.0.7 is resolved from host:172.27.0.7
12/21 14:50:22.236869 0x7fc95d859200 UTL successed to test TCP port:6000
12/21 14:50:22.237215 0x7fc95d859200 UTL successed to test UDP port:6000
...
...
...
12/21 14:50:22.676891 0x7fc95d859200 UTL successed to test TCP port:6010
12/21 14:50:22.677240 0x7fc95d859200 UTL successed to test UDP port:6010
12/21 14:50:22.720893 0x7fc95d859200 UTL successed to test TCP port:6011
12/21 14:50:22.721274 0x7fc95d859200 UTL successed to test UDP port:6011
```

仔细阅读打印出来的错误信息，可以帮助管理员找到原因，以解决问题。

11.14.0.2. 启动状态及RPC诊断

`taos -n startup -h <fqdn of server>`

判断 taosd 服务端是否成功启动，是数据库管理员经常遇到的一种情形。特别当若干台服务器组成集群时，判断每个服务端实例是否成功启动就会是一个重要问题。除检索 taosd 服务端日志文件进行问题定位、分析外，还可以通过 `taos -n startup -h <fqdn of server>` 来诊断一个 taosd 进程的启动状态。

针对多台服务器组成的集群，当服务启动过程耗时较长时，可通过该命令行来诊断每台服务器的 taosd 实例的启动状态，以准确定位问题。

`taos -n rpc -h <fqdn of server>`

该命令用来诊断已经启动的 taosd 实例的端口是否可正常访问。如果 taosd 程序异常或者失去响应，可以通过 `taos -n rpc -h <fqdn of server>` 来发起一个与指定 fqdn 的 rpc 通信，看看 taosd 是否能收到，以此来判定是网络问题还是 taosd 程序异常问题。

11.14.0.3. sync 及 arbitrator 诊断

```
taos -n sync -P 6040 -h <fqdn of server>
taos -n sync -P 6042 -h <fqdn of server>
```

用来诊断 sync 端口是否工作正常，判断服务端 sync 模块是否成功工作。另外，-P 6042 用来诊断 arbitrator 是否配置正常，判断指定服务器的 arbitrator 是否能正常工作。

11.14.0.4. 网络速度诊断

`taos -n speed -h <fqdn of server> -P 6030 -N 10 -l 10000000 -S TCP`

从 2.1.7.0 版本开始，taos 工具新提供了一个网络速度诊断的模式，可以对一个正在运行中的 taosd 实例或者 `taos -n server` 方式模拟的一个服务端实例，以非压缩传输的方式进行网络测速。这个模式下可供调整的参数如下：

-n：设为“speed”时，表示对网络速度进行诊断。
-h：所要连接的服务端的 FQDN 或 ip 地址。如果不设置这一项，会使用本机 taos.cfg 文件中 FQDN 参数的设置作为默认值。
-P：所连接服务端的网络端口。默认值为 6030。
-N：诊断过程中使用的网络包总数。最小值是 1、最大值是 10000，默认值为 100。
-l：单个网络包的大小（单位：字节）。最小值是 1024、最大值是 1024*1024*1024，默认值为 1000。
-S：网络封包的类型。可以是 TCP 或 UDP，默认值为 TCP。

11.14.0.5. 服务端日志

taosd 服务端日志文件标志位 debugflag 默认为 131，在 debug 时往往需要将其提升到 135 或 143 。

一旦设定为 135 或 143，日志文件增长很快，特别是写入、查询请求量较大时，增长速度惊人。如合并保存日志，很容易把日志内的关键信息（如配置信息、错误信息等）冲掉。为此，服务端将重要信息日志与其他日志分开存放：

- taosinfo　存放重要信息日志
- taosdlog　存放其他日志

其中，taosinfo 日志文件最大长度由 numOfLogLines 来进行配置，一个 taosd 实例最多保留两个文件。

taosd 服务端日志采用异步落盘写入机制，优点是可以避免硬盘写入压力太大，对性能造成很大影响。缺点是，在极端情况下，存在少量日志行数丢失的可能。

11.15. 数据迁移

如果某个 2.0 实例的数据需要迁移到一个新的环境，例如：将生产环境数据复制到 UAT 环境，可以很简单地通过配置 dnodeEPS.json 实现了。

特别提示：仅将数据文件复制到新服务器，直接启动 taosd 服务端实例会失败，切记！

迁移步骤如下：

1. 规划好新系统各节点 fqdn 及 IP 地址，以及与旧系统各节点的对应关系
2. 先登录旧系统，show dnodes 记录下各个 dnode 的 ID 对应的 End Point 及 IP
3. 将旧系统各节点的数据文件复制到新系统对应的节点
4. 将旧系统各节点的 taos.cfg 复制到新系统对应的各节点，并做好相应修改：firstEP/fqdn/dataDir/logDir…
5. 编辑新系统各节点数据文件夹里 dnodeEps.json，将其修改为旧系统的 dnode ID 对应的新系统各节点的 fqdn 和 port
6. 启动新系统，迁移完毕

如系统内找不到 dnodeEps.json，则需要联系涛思售后团队，协助迁移。

12. TAOS SQL

本文档说明 TAOS SQL 支持的语法规则、主要查询功能、支持的 SQL 查询函数，以及常用技巧等内容。阅读本文档需要读者具有基本的 SQL 语言的基础。

TAOS SQL 是用户对 TDengine 进行数据写入和查询的主要工具。TAOS SQL 为了便于用户快速上手，在一定程度上提供类似于标准 SQL 类似的风格和模式。严格意义上，TAOS SQL 并不是也不试图提供 SQL 标准的语法。此外，由于 TDengine 针对的时序性结构化数据不提供删除功能，因此在 TAO SQL 中不提供数据删除的相关功能。

TAOS SQL 不支持关键字的缩写，例如 DESCRIBE 不能缩写为 DESC。

本章节 SQL 语法遵循如下约定：

- < > 里的内容是用户需要输入的，但不要输入 <> 本身
- [ ] 表示内容为可选项，但不能输入 [] 本身
- | 表示多选一，选择其中一个即可，但不能输入 | 本身
- … 表示前面的项可重复多个

为更好地说明 SQL 语法的规则及其特点，本文假设存在一个数据集。以智能电表(meters)为例，假设每个智能电表采集电流、电压、相位三个量。其建模如下：

```
taos> DESCRIBE meters;
             Field              |        Type         |   Length    |    Note    |
================================================================================
 ts                             | TIMESTAMP           |           8 |            |
 current                        | FLOAT               |           4 |            |
 voltage                        | INT                 |           4 |            |
 phase                          | FLOAT               |           4 |            |
 location                       | BINARY              |          64 | TAG        |
 groupid                        | INT                 |           4 | TAG        |
```

数据集包含 4 个智能电表的数据，按照 TDengine 的建模规则，对应 4 个子表，其名称分别是 d1001, d1002, d1003, d1004。

12.1. 支持的数据类型

使用 TDengine，最重要的是时间戳。创建并插入记录、查询历史记录的时候，均需要指定时间戳。时间戳有如下规则：

- 时间格式为 `YYYY-MM-DD HH:mm:ss.MS`，默认时间分辨率为毫秒。比如：`2017-08-12 18:25:58.128`
- 内部函数 now 是客户端的当前时间
- 插入记录时，如果时间戳为 now，插入数据时使用提交这条记录的客户端的当前时间
- Epoch Time：时间戳也可以是一个长整数，表示从格林威治时间 1970-01-01 00:00:00.000 (UTC/GMT) 开始的毫秒数（相应地，如果所在 Database 的时间精度设置为“微秒”，则长整型格式的时间戳含义也就对应于从格林威

治时间 1970-01-01 00:00:00.000 (UTC/GMT) 开始的微秒数；纳秒精度的逻辑也是类似的。）
- 时间可以加减，比如 now-2h，表明查询时刻向前推 2 个小时（最近 2 小时）。数字后面的时间单位可以是 b(纳秒)、u(微秒)、a(毫秒)、s(秒)、m(分)、h(小时)、d(天)、w(周)。 比如 `select * from t1 where ts > now-2w and ts <= now-1w`，表示查询两周前整整一周的数据。在指定降频操作（down sampling）的时间窗口（interval）时，时间单位还可以使用 n(自然月) 和 y(自然年)。

TDengine 缺省的时间戳是毫秒精度，但通过在 CREATE DATABASE 时传递的 PRECISION 参数就可以支持微秒和纳秒。（从 2.1.5.0 版本开始支持纳秒精度）

在TDengine中，普通表的数据模型中可使用以下 10 种数据类型。

#	类型	Bytes	说明
1	TIMESTAMP	8	时间戳。缺省精度毫秒，可支持微秒和纳秒。从格林威治时间 1970-01-01 00:00:00.000 (UTC/GMT) 开始，计时不能早于该时间。（从 2.0.18.0 版本开始，已经去除了这一时间范围限制）（从 2.1.5.0 版本开始支持纳秒精度）
2	INT	4	整型，范围 [-2^31+1, 2^31-1], -2^31 用作 NULL
3	BIGINT	8	长整型，范围 [-2^63+1, 2^63-1], -2^63 用于 NULL
4	FLOAT	4	浮点型，有效位数 6-7，范围 [-3.4E38, 3.4E38]
5	DOUBLE	8	双精度浮点型，有效位数 15-16，范围 [-1.7E308, 1.7E308]
6	BINARY	自定义	记录单字节字符串，建议只用于处理 ASCII 可见字符，中文等多字节字符需使用 nchar。理论上，最长可以有 16374 字节，但由于每行数据最多 16K 字节，实际上限一般小于理论值。binary 仅支持字符串输入，字符串两端需使用单引号引用。使用时须指定大小，如 binary(20) 定义了最长为 20 个单字节字符的字符串，每个字符占 1 byte 的存储空间，总共固定占用 20 bytes 的空间，此时如果用户字符串超出 20 字节将会报错。对于字符串内的单引号，可以用转义字符反斜线加单引号来表示，即 \'。
7	SMALLINT	2	短整型， 范围 [-32767, 32767], -32768 用于 NULL
8	TINYINT	1	单字节整型，范围 [-127, 127], -128 用于 NULL
9	BOOL	1	布尔型，{true, false}
10	NCHAR	自定义	记录包含多字节字符在内的字符串，如中文字符。每个 nchar 字符占用 4 bytes 的存储空间。字符串两端使用单引号引用，字符串内的单引号需用转义字符 \'。nchar 使用时须指定字符串大小，类型为 nchar(10) 的列表示此列的字符串最多存储 10 个 nchar 字符，会固定占用 40 bytes 的空间。如果用户字符串长度超出声明长度，将会报错。

TDengine 的企业版还额外支持以下 4 种数据类型：

#	类型	Bytes	说明
11	INT UNSIGNED	4	无符号整型，范围 [0, 2^32-2], 2^32-1用作NULL 【2.0.17.0以后支持】
12	BIGINT UNSIGNED	8	无符号长整型，范围 [0, 2^64-2], 2^64-1用于NULL【2.0.17.0以后支持】
13	SMALLINT UNSIGNED	2	无符号短整型，范围 [0, 65534], 65535用于NULL【2.0.17.0以后支持】
14	TINYINT UNSIGNED	1	单字节整型，范围 [0, 254], 255用于NULL【2.0.17.0以后支持】

Tips:

1. TDengine 对 SQL 语句中的英文字符不区分大小写，自动转化为小写执行。因此用户大小写敏感的字符串及密码，需要使用单引号将字符串引起来。
2. 注意，虽然 Binary 类型在底层存储上支持字节型的二进制字符，但不同编程语言对二进制数据的处理方式并不保证一致，因此建议在 Binary 类型中只存储 ASCII 可见字符，而避免存储不可见字符。多字节的数据，例如中文字符，则需要使用 nchar 类型进行保存。如果强行使用 Binary 类型保存中文字符，虽然有时也能正常读写，但并不带有字符集信息，很容易出现数据乱码甚至数据损坏等情况。

12.2. 数据库管理

- 创建数据库

```
CREATE DATABASE [IF NOT EXISTS] db_name [KEEP keep,keep0,keep1] [DAYS days]
[UPDATE 1];
```

说明:

1) KEEP是该数据库的数据保留多长天数，缺省是3650天(10年)，数据库会自动删除超过时限的数据；（在企业版中，KEEP 参数的格式是[KEEP keep0,keep1,keep2]；其中 keep0 是 0 级存储的保存天数，keep1 是 1 级存储的保存天数，keep2 是 2 级存储的保存天数。在 2.1.3.0 以前的版本中，KEEP 参数的顺序是 keep2,keep0,keep1 。）

2) UPDATE 标志数据库支持更新相同时间戳数据；

3) 数据库名最大长度为33；

4) 一条SQL 语句的最大长度为65480个字符；

5) 数据库还有更多与存储相关的配置参数，请参见 服务端配置 章节。

- 显示系统当前参数

```
SHOW VARIABLES;
```

- 使用数据库

```
USE db_name;
```

使用/切换数据库（在 RESTful 连接方式下无效）。

- 删除数据库

```
DROP DATABASE [IF EXISTS] db_name;
```

删除数据库。指定 Database 所包含的全部数据表将被删除，谨慎使用！

- 修改数据库参数

```
ALTER DATABASE db_name COMP 2;
```

COMP 参数是指修改数据库文件压缩标志位，缺省值为 2，取值范围为 [0, 2]。0 表示不压缩，1 表示一阶段压缩，2 表示两阶段压缩。

```
ALTER DATABASE db_name REPLICA 2;
```

REPLICA 参数是指修改数据库副本数，取值范围 [1, 3]。在集群中使用，副本数必须小于或等于 DNODE 的数目。

```
ALTER DATABASE db_name KEEP 365;
```

KEEP 参数是指修改数据文件保存的天数，缺省值为 3650，取值范围 [days, 365000]，必须大于或等于 days 参数值。

```
ALTER DATABASE db_name QUORUM 2;
```

QUORUM 参数是指数据写入成功所需要的确认数，取值范围 [1, 2]。对于异步复制，quorum 设为 1，具有 master 角色的虚拟节点自己确认即可。对于同步复制，需要至少大于等于 2。原则上，Quorum >= 1 并且 Quorum <= replica(副本数)，这个参数在启动一个同步模块实例时需要提供。

```
ALTER DATABASE db_name BLOCKS 100;
```

BLOCKS 参数是每个 VNODE (TSDB) 中有多少 cache 大小的内存块，因此一个 VNODE 的用的内存大小粗略为（cache * blocks）。取值范围 [3, 1000]。

```
ALTER DATABASE db_name CACHELAST 0;
```

CACHELAST 参数控制是否在内存中缓存子表的最近数据。缺省值为 0，取值范围 [0, 1, 2, 3]。其中 0 表示不缓存，1 表示缓存子表最近一行数据，2 表示缓存子表每一列的最近的非 NULL 值，3 表示同时打开缓存最近行和列功能。（从 2.0.11.0 版本开始支持参数值 [0, 1]，从 2.1.2.0 版本开始支持参数值 [0, 1, 2, 3]。）
说明：缓存最近行，将显著改善 LAST_ROW 函数的性能表现；缓存每列的最近非 NULL 值，将显著改善无特殊影响（WHERE、ORDER BY、GROUP BY、INTERVAL）下的 LAST 函数的性能表现。

Tips: 以上所有参数修改后都可以用show databases来确认是否修改成功。另外，从 2.1.3.0 版本开始，修改这些参数后无需重启服务器即可生效。

- 显示系统所有数据库

```
SHOW DATABASES;
```

- 显示一个数据库的创建语句

```
SHOW CREATE DATABASE db_name;
```

常用于数据库迁移。对一个已经存在的数据库，返回其创建语句；在另一个集群中执行该语句，就能得到一个设置完全相同的 Database。

12.3. 表管理

- 创建数据表

```
CREATE TABLE [IF NOT EXISTS] tb_name (timestamp_field_name TIMESTAMP, field1_name data_type1 [, field2_name data_type2 ...]);
```

说明：

1) 表的第一个字段必须是 TIMESTAMP，并且系统自动将其设为主键；

2) 表名最大长度为 192；

3) 表的每行长度不能超过 16k 个字符;（注意：每个 BINARY/NCHAR 类型的列还会额外占用 2 个字节的存储位置）

4) 子表名只能由字母、数字和下划线组成，且不能以数字开头

5) 使用数据类型 binary 或 nchar，需指定其最长的字节数，如 binary(20)，表示 20 字节；

- 以超级表为模板创建数据表

```
CREATE TABLE [IF NOT EXISTS] tb_name USING stb_name TAGS (tag_value1, ...);
```

以指定的超级表为模板，指定 TAGS 的值来创建数据表。

- 以超级表为模板创建数据表，并指定具体的 TAGS 列

```
CREATE TABLE [IF NOT EXISTS] tb_name USING stb_name (tag_name1, ...) TAGS (tag_value1, ...);
```

以指定的超级表为模板，指定一部分 TAGS 列的值来创建数据表（没被指定的 TAGS 列会设为空值）。
说明：从 2.0.17.0 版本开始支持这种方式。在之前的版本中，不允许指定 TAGS 列，而必须显式给出所有 TAGS 列的取值。

- 批量创建数据表

```
CREATE TABLE [IF NOT EXISTS] tb_name1 USING stb_name TAGS (tag_value1, ...) [IF NOT EXISTS] tb_name2 USING stb_name TAGS (tag_value2, ...) ...;
```

以更快的速度批量创建大量数据表（服务器端 2.0.14 及以上版本）。

说明：

1）批量建表方式要求数据表必须以超级表为模板。

2）在不超出 SQL 语句长度限制的前提下，单条语句中的建表数量建议控制在 1000 ~ 3000 之间，将会获得比较理想的建表速度。

- 删除数据表

```
DROP TABLE [IF EXISTS] tb_name;
```

- 显示当前数据库下的所有数据表信息

```
SHOW TABLES [LIKE tb_name_wildcar];
```

显示当前数据库下的所有数据表信息。

说明：可在 like 中使用通配符进行名称的匹配，这一通配符字符串最长不能超过 20 字节。（从 2.1.6.1 版本开始，通配符字符串的长度放宽到了 100 字节，并可以通过 taos.cfg 中的 maxWildCardsLength 参数来配置这一长度限制。但不建议使用太长的通配符字符串，将有可能严重影响 LIKE 操作的执行性能。）

通配符匹配：1）'%'（百分号）匹配0到任意个字符；2）'_'下划线匹配单个任意字符。

- 显示一个数据表的创建语句

```
SHOW CREATE TABLE tb_name;
```

常用于数据库迁移。对一个已经存在的数据表，返回其创建语句；在另一个集群中执行该语句，就能得到一个结构完全相同的数据表。

- 在线修改显示字符宽度

```
SET MAX_BINARY_DISPLAY_WIDTH <nn>;
```

如显示的内容后面以...结尾时，表示该内容已被截断，可通过本命令修改显示字符宽度以显示完整的内容。

- 获取表的结构信息

```
DESCRIBE tb_name;
```

- 表增加列

```
ALTER TABLE tb_name ADD COLUMN field_name data_type;
```

说明：

1) 列的最大个数为1024，最小个数为2；

2) 列名最大长度为64。

- 表删除列

```
ALTER TABLE tb_name DROP COLUMN field_name;
```

如果表是通过超级表创建，更改表结构的操作只能对超级表进行。同时针对超级表的结构更改对所有通过该结构创建的表生效。对于不是通过超级表创建的表，可以直接修改表结构。

- 表修改列宽

```
ALTER TABLE tb_name MODIFY COLUMN field_name data_type(length);
```

如果数据列的类型是可变长格式（BINARY 或 NCHAR），那么可以使用此指令修改其宽度（只能改大，不能改小）。（2.1.3.0 版本新增）
如果表是通过超级表创建，更改表结构的操作只能对超级表进行。同时针对超级表的结构更改对所有通过该结构创建的表生效。对于不是通过超级表创建的表，可以直接修改表结构。

12.4. 超级表STable管理

注意：在 2.0.15.0 及以后的版本中，开始支持 STABLE 保留字。也即，在本节后文的指令说明中，CREATE、DROP、ALTER 三个指令在老版本中保留字需写作 TABLE 而不是 STABLE。

- 创建超级表

```
CREATE STABLE [IF NOT EXISTS] stb_name (timestamp_field_name TIMESTAMP,
field1_name data_type1 [, field2_name data_type2 ...]) TAGS (tag1_name tag_type1,
tag2_name tag_type2 [, tag3_name tag_type3]);
```

创建 STable，与创建表的 SQL 语法相似，但需要指定 TAGS 字段的名称和类型。

说明:

1) TAGS 列的数据类型不能是 timestamp 类型；（从 2.1.3.0 版本开始，TAGS 列中支持使用 timestamp 类型，但需注意在 TAGS 中的 timestamp 列写入数据时需要提供给定值，而暂不支持四则运算，例如 `NOW + 10s` 这类表达式）

2) TAGS 列名不能与其他列名相同；

3) TAGS 列名不能为预留关键字（参见：参数限制与保留关键字 章节）；

4) TAGS 最多允许 128 个，至少 1 个，总长度不超过 16 KB。

- 删除超级表

```
DROP STABLE [IF EXISTS] stb_name;
```

删除 STable 会自动删除通过 STable 创建的子表。

- 显示当前数据库下的所有超级表信息

```
SHOW STABLES [LIKE tb_name_wildcard];
```

查看数据库内全部 STable，及其相关信息，包括 STable 的名称、创建时间、列数量、标签（TAG）数量、通过该 STable 建表的数量。

- 显示一个超级表的创建语句

```
SHOW CREATE STABLE stb_name;
```

常用于数据库迁移。对一个已经存在的超级表，返回其创建语句；在另一个集群中执行该语句，就能得到一个结构完全相同的超级表。

- 获取超级表的结构信息

```
DESCRIBE stb_name;
```

- 超级表增加列

```
ALTER STABLE stb_name ADD COLUMN field_name data_type;
```

- 超级表删除列

```
ALTER STABLE stb_name DROP COLUMN field_name;
```

- 超级表修改列宽

```
ALTER STABLE stb_name MODIFY COLUMN field_name data_type(length);
```

如果数据列的类型是可变长格式（BINARY 或 NCHAR），那么可以使用此指令修改其宽度（只能改大，不能改小）。（2.1.3.0 版本新增）

12.5. 超级表 STable 中 TAG 管理

- 添加标签

```
ALTER STABLE stb_name ADD TAG new_tag_name tag_type;
```

为 STable 增加一个新的标签，并指定新标签的类型。标签总数不能超过 128 个，总长度不超过 16k 个字符。

- 删除标签

```
ALTER STABLE stb_name DROP TAG tag_name;
```

删除超级表的一个标签，从超级表删除某个标签后，该超级表下的所有子表也会自动删除该标签。

- 修改标签名

```
ALTER STABLE stb_name CHANGE TAG old_tag_name new_tag_name;
```

修改超级表的标签名，从超级表修改某个标签名后，该超级表下的所有子表也会自动更新该标签名。

- 修改标签列宽度

```
ALTER STABLE stb_name MODIFY TAG tag_name data_type(length);
```

如果标签的类型是可变长格式（BINARY 或 NCHAR），那么可以使用此指令修改其宽度（只能改大，不能改小）。（2.1.3.0 版本新增）

- 修改子表标签值

```
ALTER TABLE tb_name SET TAG tag_name=new_tag_value;
```

说明：除了更新标签的值的操作是针对子表进行，其他所有的标签操作（添加标签、删除标签等）均只能作用于 STable，不能对单个子表操作。对 STable 添加标签以后，依托于该 STable 建立的所有表将自动增加了一个标签，所有新增标签的默认值都是 NULL。

12.6. 数据写入

12.6.1. 写入语法：

```
INSERT INTO
    tb_name
        [USING stb_name [(tag1_name, ...)] TAGS (tag1_value, ...)]
        [(field1_name, ...)]
        VALUES (field1_value, ...) [(field1_value2, ...) ...] | FILE csv_file_path
    [tb2_name
        [USING stb_name [(tag1_name, ...)] TAGS (tag1_value, ...)]
        [(field1_name, ...)]
        VALUES (field1_value, ...) [(field1_value2, ...) ...] | FILE csv_file_path
    ...];
```

12.6.2. 详细描述及示例：

- 插入一条或多条记录

指定已经创建好的数据子表的表名，并通过 VALUES 关键字提供一行或多行数据，即可向数据库写入这些数据。例如，执行如下语句可以写入一行记录：

```
INSERT INTO d1001 VALUES (NOW, 10.2, 219, 0.32);
```

或者，可以通过如下语句写入两行记录：

```
INSERT INTO d1001 VALUES ('2021-07-13 14:06:32.272', 10.2, 219, 0.32)
(1626164208000, 10.15, 217, 0.33);
```

注意：

1）在第二个例子中，两行记录的首列时间戳使用了不同格式的写法。其中字符串格式的时间戳写法不受所在 DATABASE 的时间精度设置影响；而长整形格式的时间戳写法会受到所在 DATABASE 的时间精度设置影响——例子中的时间戳在毫秒精度下可以写作 1626164208000，而如果是在微秒精度设置下就需要写为 1626164208000000，纳秒精度设置下需要写为 1626164208000000000。

2）在使用“插入多条记录”方式写入数据时，不能把第一列的时间戳取值都设为 NOW，否则会导致语句中的多条记录使用相同的时间戳，于是就可能出现相互覆盖以致这些数据行无法全部被正确保存。其原因在于，NOW 函数在执行中会被解析为所在 SQL 语句的实际执行时间，出现在同一语句中的多个 NOW 标记也就会被替换为完全相同的时间戳取值。
3）允许插入的最老记录的时间戳，是相对于当前服务器时间，减去配置的 keep 值（数据保留的天数）；允许插入的最新记录的时间戳，是相对于当前服务器时间，加上配置的 days 值（数据文件存储数据的时间跨度，单位为天）。keep 和 days 都是可以在创建数据库时指定的，缺省值分别是 3650 天和 10 天。

- **插入记录，数据对应到指定的列**
  向数据子表中插入记录时，无论插入一行还是多行，都可以让数据对应到指定的列。对于 SQL 语句中没有出现的列，数据库将自动填充为 NULL。主键（时间戳）不能为 NULL。例如：

```
INSERT INTO d1001 (ts, current, phase) VALUES ('2021-07-13 14:06:33.196', 10.27, 0.31);
```

  说明：如果不指定列，也即使用全列模式——那么在 VALUES 部分提供的数据，必须为数据表的每个列都显式地提供数据。全列模式写入速度会远快于指定列，因此建议尽可能采用全列写入方式，此时空列可以填入 NULL。

- **向多个表插入记录**
  可以在一条语句中，分别向多个表插入一条或多条记录，并且也可以在插入过程中指定列。例如：

```
INSERT INTO d1001 VALUES ('2021-07-13 14:06:34.630', 10.2, 219, 0.32) ('2021-07-13 14:06:35.779', 10.15, 217, 0.33)
            d1002 (ts, current, phase) VALUES ('2021-07-13 14:06:34.255', 10.27, 0.31) ;
```

- **插入记录时自动建表**
  如果用户在写数据时并不确定某个表是否存在，此时可以在写入数据时使用自动建表语法来创建不存在的表，若该表已存在则不会建立新表。自动建表时，要求必须以超级表为模板，并写明数据表的 TAGS 取值。例如：

```
INSERT INTO d21001 USING meters TAGS ('Beijing.Chaoyang', 2) VALUES ('2021-07-13 14:06:32.272', 10.2, 219, 0.32);
```

  也可以在自动建表时，只是指定部分 TAGS 列的取值，未被指定的 TAGS 列将置为 NULL。例如：

```
INSERT INTO d21001 USING meters (groupId) TAGS (2) VALUES ('2021-07-13 14:06:33.196', 10.15, 217, 0.33);
```

  自动建表语法也支持在一条语句中向多个表插入记录。例如：

```
INSERT INTO d21001 USING meters TAGS ('Beijing.Chaoyang', 2) VALUES ('2021-07-13 14:06:34.630', 10.2, 219, 0.32) ('2021-07-13 14:06:35.779', 10.15, 217, 0.33)
            d21002 USING meters (groupId) TAGS (2) VALUES ('2021-07-13 14:06:34.255', 10.15, 217, 0.33)
            d21003 USING meters (groupId) TAGS (2) (ts, current, phase) VALUES ('2021-07-13 14:06:34.255', 10.27, 0.31);
```

说明：在 2.0.20.5 版本之前，在使用自动建表语法并指定列时，子表的列名必须紧跟在子表名称后面，而不能如例子里那样放在 TAGS 和 VALUES 之间。从 2.0.20.5 版本开始，两种写法都可以，但不能在一条 SQL 语句中混用，否则会报语法错误。

- 插入来自文件的数据记录
  除了使用 VALUES 关键字插入一行或多行数据外，也可以把要写入的数据放在 CSV 文件中（英文逗号分隔、英文单引号括住每个值）供 SQL 指令读取。其中 CSV 文件无需表头。例如，如果 /tmp/csvfile.csv 文件的内容为：

```
'2021-07-13 14:07:34.630', '10.2', '219', '0.32'
'2021-07-13 14:07:35.779', '10.15', '217', '0.33'
```

  那么通过如下指令可以把这个文件中的数据写入子表中：

```
INSERT INTO d1001 FILE '/tmp/csvfile.csv';
```

- 插入来自文件的数据记录，并自动建表
  从 2.1.5.0 版本开始，支持在插入来自 CSV 文件的数据时，以超级表为模板来自动创建不存在的数据表。例如：

```
INSERT INTO d21001 USING meters TAGS ('Beijing.Chaoyang', 2) FILE
'/tmp/csvfile.csv';
```

  也可以在一条语句中向多个表以自动建表的方式插入记录。例如：

```
INSERT INTO d21001 USING meters TAGS ('Beijing.Chaoyang', 2) FILE
'/tmp/csvfile_21001.csv'
            d21002 USING meters (groupId) TAGS (2) FILE '/tmp/csvfile_21002.csv';
```

历史记录写入：可使用IMPORT或者INSERT命令，IMPORT的语法，功能与INSERT完全一样。

说明：针对 insert 类型的 SQL 语句，我们采用的流式解析策略，在发现后面的错误之前，前面正确的部分 SQL 仍会执行。下面的 SQL 中，INSERT 语句是无效的，但是 d1001 仍会被创建。

```
taos> CREATE TABLE meters(ts TIMESTAMP, current FLOAT, voltage INT, phase FLOAT)
TAGS(location BINARY(30), groupId INT);
Query OK, 0 row(s) affected (0.008245s)

taos> SHOW STABLES;
              name              |      created_time       | columns |  tags  |
tables    |
============================================================================================
 meters                         | 2020-08-06 17:50:27.831 |       4 |      2 |
      0 |
Query OK, 1 row(s) in set (0.001029s)

taos> SHOW TABLES;
Query OK, 0 row(s) in set (0.000946s)

taos> INSERT INTO d1001 USING meters TAGS('Beijing.Chaoyang', 2) VALUES('a');
```

```
14
15 DB error: invalid SQL: 'a' (invalid timestamp) (0.039494s)
16
17 taos> SHOW TABLES;
18            table_name           |      created_time       | columns |
   stable_name           |
19 ================================================================================
   ====================
20  d1001                          | 2020-08-06 17:52:02.097 |       4 | meters
                    |
21 Query OK, 1 row(s) in set (0.001091s)
```

12.7. 数据查询

12.7.1. 查询语法:

```
1  SELECT select_expr [, select_expr ...]
2      FROM {tb_name_list}
3      [WHERE where_condition]
4      [SESSION(ts_col, tol_val)]
5      [STATE_WINDOW(col)]
6      [INTERVAL(interval_val [, interval_offset]) [SLIDING sliding_val]]
7      [FILL(fill_mod_and_val)]
8      [GROUP BY col_list]
9      [ORDER BY col_list { DESC | ASC }]
10     [SLIMIT limit_val [SOFFSET offset_val]]
11     [LIMIT limit_val [OFFSET offset_val]]
12     [>> export_file];
```

12.7.1.1. 通配符

通配符 * 可以用于代指全部列。对于普通表，结果中只有普通列。

```
1 taos> SELECT * FROM d1001;
2            ts            |       current        |   voltage   |        phase
    |
3 ================================================================================
  ===
4  2018-10-03 14:38:05.000 |             10.30000 |         219 |
  0.31000 |
5  2018-10-03 14:38:15.000 |             12.60000 |         218 |
  0.33000 |
6  2018-10-03 14:38:16.800 |             12.30000 |         221 |
  0.31000 |
7 Query OK, 3 row(s) in set (0.001165s)
```

在针对超级表，通配符包含 *标签列* 。

```
taos> SELECT * FROM meters;
           ts            |       current        |   voltage   |        phase
  |            location            |   groupid   |
================================================================================
================================================
 2018-10-03 14:38:05.500 |             11.80000 |         221 |
 0.28000 | Beijing.Haidian                |           2 |
 2018-10-03 14:38:16.600 |             13.40000 |         223 |
 0.29000 | Beijing.Haidian                |           2 |
 2018-10-03 14:38:05.000 |             10.80000 |         223 |
 0.29000 | Beijing.Haidian                |           3 |
 2018-10-03 14:38:06.500 |             11.50000 |         221 |
 0.35000 | Beijing.Haidian                |           3 |
 2018-10-03 14:38:04.000 |             10.20000 |         220 |
 0.23000 | Beijing.Chaoyang               |           3 |
 2018-10-03 14:38:16.650 |             10.30000 |         218 |
 0.25000 | Beijing.Chaoyang               |           3 |
 2018-10-03 14:38:05.000 |             10.30000 |         219 |
 0.31000 | Beijing.Chaoyang               |           2 |
 2018-10-03 14:38:15.000 |             12.60000 |         218 |
 0.33000 | Beijing.Chaoyang               |           2 |
 2018-10-03 14:38:16.800 |             12.30000 |         221 |
 0.31000 | Beijing.Chaoyang               |           2 |
Query OK, 9 row(s) in set (0.002022s)
```

通配符支持表名前缀，以下两个SQL语句均为返回全部的列：

```
SELECT * FROM d1001;
SELECT d1001.* FROM d1001;
```

在JOIN查询中，带前缀的*和不带前缀*返回的结果有差别， *返回全部表的所有列数据（不包含标签），带前缀的通配符，则只返回该表的列数据。

```
taos> SELECT * FROM d1001, d1003 WHERE d1001.ts=d1003.ts;
           ts            | current |   voltage   |    phase     |           ts
       | current |   voltage   |    phase     |
================================================================================
================================================
 2018-10-03 14:38:05.000 | 10.30000|         219 |      0.31000 | 2018-10-03
14:38:05.000 | 10.80000|         223 |      0.29000 |
Query OK, 1 row(s) in set (0.017385s)
```

```
taos> SELECT d1001.* FROM d1001,d1003 WHERE d1001.ts = d1003.ts;
           ts            |       current        |   voltage   |        phase
  |
=================================================================================
===
 2018-10-03 14:38:05.000 |             10.30000 |         219 |
 0.31000 |
Query OK, 1 row(s) in set (0.020443s)
```

在使用SQL函数来进行查询的过程中，部分SQL函数支持通配符操作。其中的区别在于：
count(*)函数只返回一列。first、last、last_row函数则是返回全部列。

```
taos> SELECT COUNT(*) FROM d1001;
       count(*)        |
=======================
                     3 |
Query OK, 1 row(s) in set (0.001035s)
```

```
taos> SELECT FIRST(*) FROM d1001;
        first(ts)        |    first(current)    | first(voltage) |     first(phase)
    |
=================================================================================
======
 2018-10-03 14:38:05.000 |             10.30000 |            219 |
 0.31000 |
Query OK, 1 row(s) in set (0.000849s)
```

12.7.1.2. 标签列

从 2.0.14 版本开始，支持在普通表的查询中指定 *标签列*，且标签列的值会与普通列的数据一起返回。

```
taos> SELECT location, groupid, current FROM d1001 LIMIT 2;
            location            |   groupid   |       current        |
=================================================================
 Beijing.Chaoyang               |           2 |             10.30000 |
 Beijing.Chaoyang               |           2 |             12.60000 |
Query OK, 2 row(s) in set (0.003112s)
```

注意：普通表的通配符 * 中并不包含 *标签列*。

12.7.1.2.1. 获取标签列的去重取值

从 2.0.15 版本开始，支持在超级表查询标签列时，指定 DISTINCT 关键字，这样将返回指定标签列的所有不重复取值。

```
SELECT DISTINCT tag_name FROM stb_name;
```

注意：目前 DISTINCT 关键字只支持对超级表的标签列进行去重，而不能用于普通列。

12.7.1.3. 结果集列名

SELECT子句中，如果不指定返回结果集合的列名，结果集列名称默认使用SELECT子句中的表达式名称作为列名称。此外，用户可使用AS来重命名返回结果集合中列的名称。例如：

```
taos> SELECT ts, ts AS primary_key_ts FROM d1001;
           ts            |     primary_key_ts      |
==================================================
 2018-10-03 14:38:05.000 | 2018-10-03 14:38:05.000 |
 2018-10-03 14:38:15.000 | 2018-10-03 14:38:15.000 |
 2018-10-03 14:38:16.800 | 2018-10-03 14:38:16.800 |
Query OK, 3 row(s) in set (0.001191s)
```

但是针对first(*)、last(*)、last_row(*)不支持针对单列的重命名。

12.7.1.4. 隐式结果列

Select_exprs可以是表所属列的列名，也可以是基于列的函数表达式或计算式，数量的上限256个。当用户使用了interval或group by tags的子句以后，在最后返回结果中会强制返回时间戳列（第一列）和group by子句中的标签列。后续的版本中可以支持关闭group by子句中隐式列的输出，列输出完全由select子句控制。

12.7.1.5. 表（超级表）列表

FROM关键字后面可以是若干个表（超级表）列表，也可以是子查询的结果。
如果没有指定用户的当前数据库，可以在表名称之前使用数据库的名称来指定表所属的数据库。例如：power.d1001 方式来跨库使用表。

```
SELECT * FROM power.d1001;
------------------------------
USE power;
SELECT * FROM d1001;
```

12.7.1.6. 特殊功能

部分特殊的查询功能可以不使用FROM子句执行。获取当前所在的数据库 database()：

```
taos> SELECT DATABASE();
           database()           |
================================
 power                          |
Query OK, 1 row(s) in set (0.000079s)
```

如果登录的时候没有指定默认数据库，且没有使用USE命令切换数据，则返回NULL。

```
taos> SELECT DATABASE();
           database()           |
================================
 NULL                           |
Query OK, 1 row(s) in set (0.000184s)
```

获取服务器和客户端版本号：

```
taos> SELECT CLIENT_VERSION();
 client_version() |
===================
 2.0.0.0          |
Query OK, 1 row(s) in set (0.000070s)

taos> SELECT SERVER_VERSION();
 server_version() |
===================
 2.0.0.0          |
Query OK, 1 row(s) in set (0.000077s)
```

服务器状态检测语句。如果服务器正常，返回一个数字（例如 1）。如果服务器异常，返回error code。该SQL语法能兼容连接池对于TDengine状态的检查及第三方工具对于数据库服务器状态的检查。并可以避免出现使用了错误的心跳检测SQL语句导致的连接池连接丢失的问题。

```
taos> SELECT SERVER_STATUS();
 server_status() |
==================
               1 |
Query OK, 1 row(s) in set (0.000074s)

taos> SELECT SERVER_STATUS() AS status;
   status    |
==============
           1 |
Query OK, 1 row(s) in set (0.000081s)
```

12.7.1.7. TAOS SQL中特殊关键词

> TBNAME： 在超级表查询中可视为一个特殊的标签，代表查询涉及的子表名
>
> _c0: 表示表（超级表）的第一列

12.7.1.8. 小技巧

获取一个超级表所有的子表名及相关的标签信息：

```
SELECT TBNAME, location FROM meters;
```

统计超级表下辖子表数量：

```
SELECT COUNT(TBNAME) FROM meters;
```

以上两个查询均只支持在WHERE条件子句中添加针对标签（TAGS）的过滤条件。例如：

```
taos> SELECT TBNAME, location FROM meters;
             tbname             |            location            |
==================================================================
 d1004                          | Beijing.Haidian                |
 d1003                          | Beijing.Haidian                |
 d1002                          | Beijing.Chaoyang               |
 d1001                          | Beijing.Chaoyang               |
Query OK, 4 row(s) in set (0.000881s)

taos> SELECT COUNT(tbname) FROM meters WHERE groupId > 2;
     count(tbname)     |
========================
                     2 |
Query OK, 1 row(s) in set (0.001091s)
```

- 可以使用 * 返回所有列，或指定列名。可以对数字列进行四则运算，可以给输出的列取列名。
  - 暂不支持含列名的四则运算表达式用于条件过滤算子（例如，不支持 `where a*2>6;`，但可以写 `where a>6/2;`）。
  - 暂不支持含列名的四则运算表达式作为 SQL 函数的应用对象（例如，不支持 `select min(2*a) from t;`，但可以写 `select 2*min(a) from t;`）。
- WHERE 语句可以使用各种逻辑判断来过滤数字值，或使用通配符来过滤字符串。
- 输出结果缺省按首列时间戳升序排序，但可以指定按降序排序(_c0 指首列时间戳)。使用 ORDER BY 对其他字段进行排序为非法操作。
- 参数 LIMIT 控制输出条数，OFFSET 指定从第几条开始输出。LIMIT/OFFSET 对结果集的执行顺序在 ORDER BY 之后。且 `LIMIT 5 OFFSET 2` 可以简写为 `LIMIT 2, 5`。
  - 在有 GROUP BY 子句的情况下，LIMIT 参数控制的是每个分组中至多允许输出的条数。
- 参数 SLIMIT 控制由 GROUP BY 指令划分的分组中，至多允许输出几个分组的数据。且 `SLIMIT 5 SOFFSET 2` 可以简写为 `SLIMIT 2, 5`。
- 通过 “>>” 输出结果可以导出到指定文件。

12.7.2. 支持的条件过滤操作

Operation	Note	Applicable Data Types
>	larger than	**timestamp** and all numeric types
<	smaller than	**timestamp** and all numeric types
>=	larger than or equal to	**timestamp** and all numeric types
<=	smaller than or equal to	**timestamp** and all numeric types
=	equal to	all types
<>	not equal to	all types
between and	within a certain range	**timestamp** and all numeric types
in	matches any value in a set	all types except first column `timestamp`
%	match with any char sequences	**binary nchar**
_	match with a single char	**binary nchar**

1. <> 算子也可以写为 != ，请注意，这个算子不能用于数据表第一列的 timestamp 字段。
2. 同时进行多个字段的范围过滤，需要使用关键词 AND 来连接不同的查询条件，暂不支持 OR 连接的不同列之间的查询过滤条件。
3. 针对单一字段的过滤，如果是时间过滤条件，则一条语句中只支持设定一个；但针对其他的（普通）列或标签列，则可以使用 `OR` 关键字进行组合条件的查询过滤。例如：`((value > 20 AND value < 30) OR (value < 12))`。
4. 从 2.0.17.0 版本开始，条件过滤开始支持 BETWEEN AND 语法，例如 `WHERE col2 BETWEEN 1.5 AND 3.25` 表示查询条件为“1.5 ≤ col2 ≤ 3.25”。
5. 从 2.1.4.0 版本开始，条件过滤开始支持 IN 算子，例如 `WHERE city IN ('Beijing', 'Shanghai')`。说明：BOOL 类型写作 `{true, false}` 或 `{0, 1}` 均可，但不能写作 0、1 之外的整数；FLOAT 和 DOUBLE 类型会受到浮点数精度影响，集合内的值在精度范围内认为和数据行的值完全相等才能匹配成功；TIMESTAMP 类型支持非主键的列。（在企业版中，IN 算子支持无符号整数类型，但需注意取值集合中也只能包含无符号整数，如果 SQL 语句中设定的取值集合包含了负数，那么对筛选结果有可能造成难以预料的影响。）

12.7.3. UNION ALL 操作符

```
SELECT ...
UNION ALL SELECT ...
[UNION ALL SELECT ...]
```

TDengine 支持 UNION ALL 操作符。也就是说，如果多个 SELECT 子句返回结果集的结构完全相同（列名、列类型、列数、顺序），那么可以通过 UNION ALL 把这些结果集合并到一起。目前只支持 UNION ALL 模式，也即在结果集的合并过程中是不去重的。

12.7.4. SQL 示例

- 对于下面的例子，表tb1用以下语句创建：

```
CREATE TABLE tb1 (ts TIMESTAMP, col1 INT, col2 FLOAT, col3 BINARY(50));
```

- 查询tb1刚过去的一个小时的所有记录：

```
SELECT * FROM tb1 WHERE ts >= NOW - 1h;
```

- 查询表tb1从2018-06-01 08:00:00.000 到2018-06-02 08:00:00.000时间范围，并且col3的字符串是'nny'结尾的记录，结果按照时间戳降序：

```
SELECT * FROM tb1 WHERE ts > '2018-06-01 08:00:00.000' AND ts <= '2018-06-02
08:00:00.000' AND col3 LIKE '%nny' ORDER BY ts DESC;
```

- 查询col1与col2的和，并取名complex, 时间大于2018-06-01 08:00:00.000, col2大于1.2，结果输出仅仅10条记录，从第5条开始：

```
SELECT (col1 + col2) AS 'complex' FROM tb1 WHERE ts > '2018-06-01 08:00:00.000'
AND col2 > 1.2 LIMIT 10 OFFSET 5;
```

- 查询过去10分钟的记录，col2的值大于3.14，并且将结果输出到文件 /home/testoutpu.csv：

```
SELECT COUNT(*) FROM tb1 WHERE ts >= NOW - 10m AND col2 > 3.14 >>
/home/testoutpu.csv;
```

12.8. SQL 函数

12.8.1. 聚合函数

TDengine支持针对数据的聚合查询。提供支持的聚合和选择函数如下：

- **COUNT**

```
SELECT COUNT([*|field_name]) FROM tb_name [WHERE clause];
```

功能说明：统计表/超级表中记录行数或某列的非空值个数。

返回结果数据类型：长整型INT64。

应用字段：应用全部字段。

适用于：表、超级表。

说明：

1）可以使用星号(*)来替代具体的字段，使用星号(*)返回全部记录数量。

2）针对同一表的（不包含NULL值）字段查询结果均相同。

3）如果统计对象是具体的列，则返回该列中非NULL值的记录数量。

示例：

```
taos> SELECT COUNT(*), COUNT(voltage) FROM meters;
    count(*)        |    count(voltage)     |
================================================
                   9 |                     9 |
Query OK, 1 row(s) in set (0.004475s)

taos> SELECT COUNT(*), COUNT(voltage) FROM d1001;
    count(*)        |    count(voltage)     |
================================================
                   3 |                     3 |
Query OK, 1 row(s) in set (0.001075s)
```

- **AVG**

```
SELECT AVG(field_name) FROM tb_name [WHERE clause];
```

功能说明：统计表/超级表中某列的平均值。

返回结果数据类型：双精度浮点数Double。

应用字段：不能应用在timestamp、binary、nchar、bool字段。

适用于：表、超级表。

示例：

```
taos> SELECT AVG(current), AVG(voltage), AVG(phase) FROM meters;
    avg(current)        |       avg(voltage)        |        avg(phase)
|
================================================================================
====
            11.466666751 |            220.444444444 |              0.293333333
|
Query OK, 1 row(s) in set (0.004135s)

taos> SELECT AVG(current), AVG(voltage), AVG(phase) FROM d1001;
    avg(current)        |       avg(voltage)        |        avg(phase)
|
================================================================================
====
            11.733333588 |            219.333333333 |              0.316666673
|
Query OK, 1 row(s) in set (0.000943s)
```

- **TWA**

```
SELECT TWA(field_name) FROM tb_name WHERE clause;
```

功能说明：时间加权平均函数。统计表中某列在一段时间内的时间加权平均。

返回结果数据类型：双精度浮点数Double。

应用字段：不能应用在timestamp、binary、nchar、bool类型字段。

适用于：表、（超级表）。

说明：从 2.1.3.0 版本开始，TWA 函数可以在由 GROUP BY 划分出单独时间线的情况下用于超级表（也即 GROUP BY tbname）。

- **IRATE**

```
SELECT IRATE(field_name) FROM tb_name WHERE clause;
```

功能说明：计算瞬时增长率。使用时间区间中最后两个样本数据来计算瞬时增长速率；如果这两个值呈递减关系，那么只取最后一个数用于计算，而不是使用二者差值。

返回结果数据类型：双精度浮点数Double。

应用字段：不能应用在timestamp、binary、nchar、bool类型字段。

适用于：表、（超级表）。

说明：（从 2.1.3.0 版本开始新增此函数）IRATE 可以在由 GROUP BY 划分出单独时间线的情况下用于超级表（也即 GROUP BY tbname）。

- **SUM**

```
SELECT SUM(field_name) FROM tb_name [WHERE clause];
```

功能说明：统计表/超级表中某列的和。

返回结果数据类型：双精度浮点数Double和长整型INT64。

应用字段：不能应用在timestamp、binary、nchar、bool类型字段。

适用于：表、超级表。

示例：

```
taos> SELECT SUM(current), SUM(voltage), SUM(phase) FROM meters;
    sum(current)        |     sum(voltage)      |        sum(phase)         |
================================================================================
            103.200000763 |                 1984 |              2.640000001 |
Query OK, 1 row(s) in set (0.001702s)

taos> SELECT SUM(current), SUM(voltage), SUM(phase) FROM d1001;
    sum(current)        |     sum(voltage)      |        sum(phase)         |
================================================================================
            35.200000763 |                 658 |              0.950000018 |
Query OK, 1 row(s) in set (0.000980s)
```

- **STDDEV**

```
SELECT STDDEV(field_name) FROM tb_name [WHERE clause];
```

功能说明：统计表中某列的均方差。

返回结果数据类型：双精度浮点数Double。

应用字段：不能应用在timestamp、binary、nchar、bool类型字段。

适用于：表。（从 2.0.15.1 版本开始，本函数也支持超级表）

示例：

```
taos> SELECT STDDEV(current) FROM d1001;
    stddev(current)      |
============================
            1.020892909 |
Query OK, 1 row(s) in set (0.000915s)
```

- **LEASTSQUARES**

```
SELECT LEASTSQUARES(field_name, start_val, step_val) FROM tb_name [WHERE clause];
```

功能说明：统计表中某列的值是主键（时间戳）的拟合直线方程。start_val是自变量初始值，step_val是自变量的步长值。

返回结果数据类型：字符串表达式（斜率, 截距）。

应用字段：不能应用在timestamp、binary、nchar、bool类型字段。

说明：自变量是时间戳，因变量是该列的值。

适用于：表。

示例：

```
taos> SELECT LEASTSQUARES(current, 1, 1) FROM d1001;
            leastsquares(current, 1, 1)             |
=====================================================
{slop:1.000000, intercept:9.733334}                 |
Query OK, 1 row(s) in set (0.000921s)
```

12.8.2. 选择函数

在使用所有的选择函数的时候，可以同时指定输出 ts 列或标签列（包括 tbname），这样就可以方便地知道被选出的值是源于哪个数据行的。

- **MIN**

```
SELECT MIN(field_name) FROM {tb_name | stb_name} [WHERE clause];
```

功能说明：统计表/超级表中某列的值最小值。

返回结果数据类型：同应用的字段。

应用字段：不能应用在timestamp、binary、nchar、bool类型字段。

适用于：表、超级表。

示例：

```
taos> SELECT MIN(current), MIN(voltage) FROM meters;
    min(current)     | min(voltage) |
======================================
            10.20000 |          218 |
Query OK, 1 row(s) in set (0.001765s)

taos> SELECT MIN(current), MIN(voltage) FROM d1001;
    min(current)     | min(voltage) |
======================================
            10.30000 |          218 |
Query OK, 1 row(s) in set (0.000950s)
```

- **MAX**

```
SELECT MAX(field_name) FROM { tb_name | stb_name } [WHERE clause];
```

功能说明：统计表/超级表中某列的值最大值。

返回结果数据类型：同应用的字段。

应用字段：不能应用在timestamp、binary、nchar、bool类型字段。

适用于：表、超级表。

示例：

```
taos> SELECT MAX(current), MAX(voltage) FROM meters;
    max(current)     | max(voltage) |
======================================
            13.40000 |          223 |
Query OK, 1 row(s) in set (0.001123s)

taos> SELECT MAX(current), MAX(voltage) FROM d1001;
    max(current)     | max(voltage) |
======================================
            12.60000 |          221 |
Query OK, 1 row(s) in set (0.000987s)
```

- **FIRST**

```
SELECT FIRST(field_name) FROM { tb_name | stb_name } [WHERE clause];
```

功能说明：统计表/超级表中某列的值最先写入的非NULL值。

返回结果数据类型：同应用的字段。

应用字段：所有字段。

适用于：表、超级表。

说明：

1）如果要返回各个列的首个（时间戳最小）非NULL值，可以使用FIRST(*);

2）如果结果集中的某列全部为NULL值，则该列的返回结果也是NULL；

3）如果结果集中所有列全部为NULL值，则不返回结果。

示例：

```
taos> SELECT FIRST(*) FROM meters;
        first(ts)        |    first(current)    | first(voltage) | first(phase)     |
==========================================================================================
2018-10-03 14:38:04.000 |             10.20000 |            220 |  0.23000 |
Query OK, 1 row(s) in set (0.004767s)

taos> SELECT FIRST(current) FROM d1002;
    first(current)    |
========================
            10.20000 |
Query OK, 1 row(s) in set (0.001023s)
```

- **LAST**

```
SELECT LAST(field_name) FROM { tb_name | stb_name } [WHERE clause];
```

功能说明：统计表/超级表中某列的值最后写入的非NULL值。

返回结果数据类型：同应用的字段。

应用字段：所有字段。

适用于：表、超级表。

说明：

1）如果要返回各个列的最后（时间戳最大）一个非NULL值，可以使用LAST(*)；

2）如果结果集中的某列全部为NULL值，则该列的返回结果也是NULL；如果结果集中所有列全部为NULL值，则不返回结果。

示例：

```
taos> SELECT LAST(*) FROM meters;
        last(ts)         |    last(current)     | last(voltage) | last(phase)      |
==========================================================================================
2018-10-03 14:38:16.800 |             12.30000 |           221 |  0.31000 |
Query OK, 1 row(s) in set (0.001452s)

taos> SELECT LAST(current) FROM d1002;
    last(current)     |
========================
            10.30000 |
Query OK, 1 row(s) in set (0.000843s)
```

- **TOP**

```
SELECT TOP(field_name, K) FROM { tb_name | stb_name } [WHERE clause];
```

功能说明： 统计表/超级表中某列的值最大 *k* 个非 NULL 值。如果多条数据取值一样，全部取用又会超出 k 条限制时，系统会从相同值中随机选取符合要求的数量返回。

返回结果数据类型：同应用的字段。

应用字段：不能应用在timestamp、binary、nchar、bool类型字段。

适用于：表、超级表。

说明：

1）*k*值取值范围1≤*k*≤100；

2）系统同时返回该记录关联的时间戳列；

3）限制：TOP函数不支持FILL子句。

示例：

```
taos> SELECT TOP(current, 3) FROM meters;
        ts            |   top(current, 3)    |
================================================
2018-10-03 14:38:15.000 |             12.60000 |
2018-10-03 14:38:16.600 |             13.40000 |
2018-10-03 14:38:16.800 |             12.30000 |
Query OK, 3 row(s) in set (0.001548s)

taos> SELECT TOP(current, 2) FROM d1001;
        ts            |   top(current, 2)    |
================================================
2018-10-03 14:38:15.000 |             12.60000 |
2018-10-03 14:38:16.800 |             12.30000 |
Query OK, 2 row(s) in set (0.000810s)
```

- **BOTTOM**

```
SELECT BOTTOM(field_name, K) FROM { tb_name | stb_name } [WHERE clause];
```

功能说明：统计表/超级表中某列的值最小 *k* 个非 NULL 值。如果多条数据取值一样，全部取用又会超出 k 条限制时，系统会从相同值中随机选取符合要求的数量返回。

返回结果数据类型：同应用的字段。

应用字段：不能应用在timestamp、binary、nchar、bool类型字段。

适用于：表、超级表。

说明：

1）*k*值取值范围1≤*k*≤100；

2）系统同时返回该记录关联的时间戳列；

3）限制：BOTTOM函数不支持FILL子句。

示例：

```
taos> SELECT BOTTOM(voltage, 2) FROM meters;
        ts            | bottom(voltage, 2) |
================================================
2018-10-03 14:38:15.000 |                218 |
2018-10-03 14:38:16.650 |                218 |
Query OK, 2 row(s) in set (0.001332s)

taos> SELECT BOTTOM(current, 2) FROM d1001;
        ts            |  bottom(current, 2)  |
================================================
2018-10-03 14:38:05.000 |             10.30000 |
2018-10-03 14:38:16.800 |             12.30000 |
Query OK, 2 row(s) in set (0.000793s)
```

- **PERCENTILE**

```
SELECT PERCENTILE(field_name, P) FROM { tb_name } [WHERE clause];
```

功能说明：统计表中某列的值百分比分位数。

返回结果数据类型： 双精度浮点数Double。

应用字段：不能应用在timestamp、binary、nchar、bool类型字段。

适用于：表。

说明：*P*值取值范围0≤*P*≤100，为0的时候等同于MIN，为100的时候等同于MAX。

示例：

```
taos> SELECT PERCENTILE(current, 20) FROM d1001;
percentile(current, 20)  |
============================
            11.100000191 |
Query OK, 1 row(s) in set (0.000787s)
```

- **APERCENTILE**

```
SELECT APERCENTILE(field_name, P) FROM { tb_name | stb_name } [WHERE clause];
```

功能说明：统计表/超级表中某列的值百分比分位数，与PERCENTILE函数相似，但是返回近似结果。

返回结果数据类型： 双精度浮点数Double。

应用字段：不能应用在timestamp、binary、nchar、bool类型字段。

适用于：表、超级表。

说明：*P*值取值范围0≤*P*≤100，为0的时候等同于MIN，为100的时候等同于MAX。推荐使用APERCENTILE函数，该函数性能远胜于PERCENTILE函数。

```
taos> SELECT APERCENTILE(current, 20) FROM d1001;
apercentile(current, 20)  |
============================
            10.300000191 |
Query OK, 1 row(s) in set (0.000645s)
```

- **LAST_ROW**

```
SELECT LAST_ROW(field_name) FROM { tb_name | stb_name };
```

功能说明：返回表/超级表的最后一条记录。

返回结果数据类型：同应用的字段。

应用字段：所有字段。

适用于：表、超级表。

说明：与LAST函数不同，LAST_ROW不支持时间范围限制，强制返回最后一条记录。

限制：LAST_ROW()不能与INTERVAL一起使用。

示例：

```
taos> SELECT LAST_ROW(current) FROM meters;
last_row(current)   |
======================
            12.30000 |
Query OK, 1 row(s) in set (0.001238s)

taos> SELECT LAST_ROW(current) FROM d1002;
last_row(current)   |
======================
            10.30000 |
Query OK, 1 row(s) in set (0.001042s)
```

- **INTERP**

```
SELECT INTERP(field_name) FROM { tb_name | stb_name } WHERE ts='timestamp' [FILL
({ VALUE | PREV | NULL | LINEAR | NEXT})];
```

功能说明：返回表/超级表的指定时间截面、指定字段的记录。

返回结果数据类型：同应用的字段。

应用字段：所有字段。

适用于：表、超级表。

说明：（从 2.0.15.0 版本开始新增此函数）INTERP 必须指定时间断面，如果该时间断面不存在直接对应的数据，那么会根据 FILL 参数的设定进行插值。其中，条件语句里面可以附带更多的筛选条件，例如标签、tbname。

示例：

```
taos> select interp(*) from meters where ts='2017-7-14 10:42:00.005' fill(prev);
       interp(ts)        | interp(f1)  | interp(f2)  | interp(f3)  |
============================================================================
 2017-07-14 10:42:00.005 |           5 |           9 |           6 |
Query OK, 1 row(s) in set (0.002912s)

taos> select interp(*) from meters where tbname in ('t1') and ts='2017-7-14
10:42:00.005' fill(prev);
       interp(ts)        | interp(f1)  | interp(f2)  | interp(f3)  |
============================================================================
 2017-07-14 10:42:00.005 |           5 |           6 |           7 |
Query OK, 1 row(s) in set (0.002005s)
```

12.8.3. 计算函数

- **DIFF**

```
SELECT DIFF(field_name) FROM tb_name [WHERE clause];
```

功能说明：统计表中某列的值与前一行对应值的差。

返回结果数据类型：同应用字段。

应用字段：不能应用在timestamp、binary、nchar、bool类型字段。

适用于：表、（超级表）。

说明：输出结果行数是范围内总行数减一，第一行没有结果输出。从 2.1.3.0 版本开始，DIFF 函数可以在由 GROUP BY 划分出单独时间线的情况下用于超级表（也即 GROUP BY tbname）。

示例：

```
taos> SELECT DIFF(current) FROM d1001;
        ts            |    diff(current)     |
================================================
2018-10-03 14:38:15.000 |              2.30000 |
2018-10-03 14:38:16.800 |             -0.30000 |
Query OK, 2 row(s) in set (0.001162s)
```

- **DERIVATIVE**

```
SELECT DERIVATIVE(field_name, time_interval, ignore_negative) FROM tb_name [WHERE
clause];
```

功能说明：统计表中某列数值的单位变化率。其中单位时间区间的长度可以通过 time_interval 参数指定，最小可以是 1 秒（1s）；ignore_negative 参数的值可以是 0 或 1，为 1 时表示忽略负值。

返回结果数据类型：双精度浮点数。

应用字段：不能应用在 timestamp、binary、nchar、bool 类型字段。

适用于：表、（超级表）。

说明：（从 2.1.3.0 版本开始新增此函数）输出结果行数是范围内总行数减一，第一行没有结果输出。DERIVATIVE 函数可以在由 GROUP BY 划分出单独时间线的情况下用于超级表（也即 GROUP BY tbname）。

- **SPREAD**

```
SELECT SPREAD(field_name) FROM { tb_name | stb_name } [WHERE clause];
```

功能说明：统计表/超级表中某列的最大值和最小值之差。

返回结果数据类型：双精度浮点数。

应用字段：不能应用在binary、nchar、bool类型字段。

适用于：表、超级表。

说明：可用于TIMESTAMP字段，此时表示记录的时间覆盖范围。

示例：

```
taos> SELECT SPREAD(voltage) FROM meters;
    spread(voltage)      |
============================
            5.000000000 |
Query OK, 1 row(s) in set (0.001792s)

taos> SELECT SPREAD(voltage) FROM d1001;
    spread(voltage)      |
============================
            3.000000000 |
Query OK, 1 row(s) in set (0.000836s)
```

- 四则运算

```
SELECT field_name [+|-|*|/|%][Value|field_name] FROM { tb_name | stb_name }
 [WHERE clause];
```

功能说明：统计表/超级表中某列或多列间的值加、减、乘、除、取余计算结果。

返回结果数据类型：双精度浮点数。

应用字段：不能应用在timestamp、binary、nchar、bool类型字段。

适用于：表、超级表。

说明：

1）支持两列或多列之间进行计算，可使用括号控制计算优先级；

2）NULL字段不参与计算，如果参与计算的某行中包含NULL，该行的计算结果为NULL。

```
taos> SELECT current + voltage * phase FROM d1001;
(current+(voltage*phase)) |
============================
             78.190000713 |
             84.540003240 |
             80.810000718 |
Query OK, 3 row(s) in set (0.001046s)
```

12.9. 按窗口切分聚合

TDengine 支持按时间段等窗口切分方式进行聚合结果查询，比如温度传感器每秒采集一次数据，但需查询每隔 10 分钟的温度平均值。这类聚合适合于降维（down sample）操作，语法如下：

```
SELECT function_list FROM tb_name
  [WHERE where_condition]
  [SESSION(ts_col, tol_val)]
  [STATE_WINDOW(col)]
  [INTERVAL(interval [, offset]) [SLIDING sliding]]
  [FILL({NONE | VALUE | PREV | NULL | LINEAR | NEXT})]

SELECT function_list FROM stb_name
  [WHERE where_condition]
  [SESSION(ts_col, tol_val)]
  [STATE_WINDOW(col)]
  [INTERVAL(interval [, offset]) [SLIDING sliding]]
  [FILL({NONE | VALUE | PREV | NULL | LINEAR | NEXT})]
  [GROUP BY tags]
```

- 在聚合查询中，function_list 位置允许使用聚合和选择函数，并要求每个函数仅输出单个结果（例如：COUNT、AVG、SUM、STDDEV、LEASTSQUARES、PERCENTILE、MIN、MAX、FIRST、LAST），而不能使用具有多行输出结果的函数（例如：TOP、BOTTOM、DIFF 以及四则运算）。
- 查询过滤、聚合等操作按照每个切分窗口为独立的单位执行。聚合查询目前支持三种窗口的划分方式：
  1. 时间窗口：聚合时间段的窗口宽度由关键词 INTERVAL 指定，最短时间间隔 10 毫秒（10a）；并且支持偏移 offset（偏移必须小于间隔），也即时间窗口划分与“UTC 时刻 0”相比的偏移量。SLIDING 语句用于指定聚合时间段的前向增量，也即每次窗口向前滑动的时长。当 SLIDING 与 INTERVAL 取值相等的时候，滑动窗口即为翻转窗口。
     - 从 2.1.5.0 版本开始，INTERVAL 语句允许的最短时间间隔调整为 1 微秒（1u），当然如果所查询的 DATABASE 的时间精度设置为毫秒级，那么允许的最短时间间隔为 1 毫秒（1a）。
     - 注意：用到 INTERVAL 语句时，除非极特殊的情况，都要求把客户端和服务端的 taos.cfg 配置文件中的 timezone 参数配置为相同的取值，以避免时间处理函数频繁进行跨时区转换而导致的严重性能影响。
  2. 状态窗口：使用整数或布尔值来标识产生记录时设备的状态量，产生的记录如果具有相同的状态量取值则归属于同一个状态窗口，数值改变后该窗口关闭。状态量所对应的列作为 STATE_WINDOW 语句的参数来指定。
  3. 会话窗口：时间戳所在的列由 SESSION 语句的 ts_col 参数指定，会话窗口根据相邻两条记录的时间戳差值来确定是否属于同一个会话——如果时间戳差异在 tol_val 以内，则认为记录仍属于同一个窗口；如果时间变化超过 tol_val，则自动开启下一个窗口。
- WHERE 语句可以指定查询的起止时间和其他过滤条件。

- FILL 语句指定某一窗口区间数据缺失的情况下的填充模式。填充模式包括以下几种：

  1. 不进行填充：NONE（默认填充模式）。
  2. VALUE 填充：固定值填充，此时需要指定填充的数值。例如：FILL(VALUE, 1.23)。
  3. PREV 填充：使用前一个非 NULL 值填充数据。例如：FILL(PREV)。
  4. NULL 填充：使用 NULL 填充数据。例如：FILL(NULL)。
  5. LINEAR 填充：根据前后距离最近的非 NULL 值做线性插值填充。例如：FILL(LINEAR)。
  6. NEXT 填充：使用下一个非 NULL 值填充数据。例如：FILL(NEXT)。

说明：

1. 使用 FILL 语句的时候可能生成大量的填充输出，务必指定查询的时间区间。针对每次查询，系统可返回不超过 1 千万条具有插值的结果。
2. 在时间维度聚合中，返回的结果中时间序列严格单调递增。
3. 如果查询对象是超级表，则聚合函数会作用于该超级表下满足值过滤条件的所有表的数据。如果查询中没有使用 GROUP BY 语句，则返回的结果按照时间序列严格单调递增；如果查询中使用了 GROUP BY 语句分组，则返回结果中每个 GROUP 内不按照时间序列严格单调递增。

时间聚合也常被用于连续查询场景，可以参考文档 连续查询(Continuous Query)。

示例： 智能电表的建表语句如下：

```
CREATE TABLE meters (ts TIMESTAMP, current FLOAT, voltage INT, phase FLOAT) TAGS
(location BINARY(64), groupId INT);
```

针对智能电表采集的数据，以 10 分钟为一个阶段，计算过去 24 小时的电流数据的平均值、最大值、电流的中位数、以及随着时间变化的电流走势拟合直线。如果没有计算值，用前一个非 NULL 值填充。使用的查询语句如下：

```
SELECT AVG(current), MAX(current), LEASTSQUARES(current, start_val, step_val),
PERCENTILE(current, 50) FROM meters
  WHERE ts>=NOW-1d
  INTERVAL(10m)
  FILL(PREV);
```

12.10. TAOS SQL 边界限制

- 数据库名最大长度为 32。
- 表名最大长度为 192，每行数据最大长度 16k 个字符（注意：数据行内每个 BINARY/NCHAR 类型的列还会额外占用 2 个字节的存储位置）。
- 列名最大长度为 64，最多允许 1024 列，最少需要 2 列，第一列必须是时间戳。
- 标签名最大长度为 64，最多允许 128 个，可以 1 个，一个表中标签值的总长度不超过 16k 个字符。
- SQL 语句最大长度 65480 个字符，但可通过系统配置参数 maxSQLLength 修改，最长可配置为 1M。
- SELECT 语句的查询结果，最多允许返回 1024 列（语句中的函数调用可能也会占用一些列空间），超限时需要显式指定较少的返回数据列，以避免语句执行报错。
- 库的数目，超级表的数目、表的数目，系统不做限制，仅受系统资源限制。

12.11. TAOS SQL其他约定

GROUP BY的限制

TAOS SQL支持对标签、TBNAME进行GROUP BY操作，也支持普通列进行GROUP BY，前提是：仅限一列且该列的唯一值小于10万个。

JOIN操作的限制

TAOS SQL支持表之间按主键时间戳来join两张表的列，暂不支持两个表之间聚合后的四则运算。

IS NOT NULL与不为空的表达式适用范围

IS NOT NULL支持所有类型的列。不为空的表达式为 <>""，仅对非数值类型的列适用。

12.12. TDengine 2.0 错误码以及对应的十进制码

状态码	模	错误码（十六进制）	错误描述	错误码（十进制）
TSDB_CODE_RPC_ACTION_IN_PROGRESS	0	0x0001	"Action in progress"	-2147483647
TSDB_CODE_RPC_AUTH_REQUIRED	0	0x0002	"Authentication required"	-2147483646
TSDB_CODE_RPC_AUTH_FAILURE	0	0x0003	"Authentication failure"	-2147483645
TSDB_CODE_RPC_REDIRECT	0	0x0004	"Redirect"	-2147483644
TSDB_CODE_RPC_NOT_READY	0	0x0005	"System not ready"	-2147483643
TSDB_CODE_RPC_ALREADY_PROCESSED	0	0x0006	"Message already processed"	-2147483642
TSDB_CODE_RPC_LAST_SESSION_NOT_FINISHED	0	0x0007	"Last session not finished"	-2147483641
TSDB_CODE_RPC_MISMATCHED_LINK_ID	0	0x0008	"Mismatched meter id"	-2147483640
TSDB_CODE_RPC_TOO_SLOW	0	0x0009	"Processing of request timed out"	-2147483639
TSDB_CODE_RPC_MAX_SESSIONS	0	0x000A	"Number of sessions reached limit"	-2147483638
TSDB_CODE_RPC_NETWORK_UNAVAIL	0	0x000B	"Unable to establish connection"	-2147483637
TSDB_CODE_RPC_APP_ERROR	0	0x000C	"Unexpected generic error in RPC"	-2147483636
TSDB_CODE_RPC_UNEXPECTED_RESPONSE	0	0x000D	"Unexpected response"	-2147483635
TSDB_CODE_RPC_INVALID_VALUE	0	0x000E	"Invalid value"	-2147483634
TSDB_CODE_RPC_INVALID_TRAN_ID	0	0x000F	"Invalid transaction id"	-2147483633
TSDB_CODE_RPC_INVALID_SESSION_ID	0	0x0010	"Invalid session id"	-2147483632
TSDB_CODE_RPC_INVALID_MSG_TYPE	0	0x0011	"Invalid message type"	-2147483631
TSDB_CODE_RPC_INVALID_RESPONSE_TYPE	0	0x0012	"Invalid response type"	-2147483630
TSDB_CODE_RPC_INVALID_TIME_STAMP	0	0x0013	"Invalid timestamp"	-2147483629

TSDB_CODE_COM_OPS_NOT_SUPPORT	0	0x0100	"Operation not supported"	-2147483392
TSDB_CODE_COM_MEMORY_CORRUPTED	0	0x0101	"Memory corrupted"	-2147483391
TSDB_CODE_COM_OUT_OF_MEMORY	0	0x0102	"Out of memory"	-2147483390
TSDB_CODE_COM_INVALID_CFG_MSG	0	0x0103	"Invalid config message"	-2147483389
TSDB_CODE_COM_FILE_CORRUPTED	0	0x0104	"Data file corrupted"	-2147483388
TSDB_CODE_TSC_INVALID_OPERATION	0	0x0200	"Invalid SQL statement"	-2147483136
TSDB_CODE_TSC_INVALID_QHANDLE	0	0x0201	"Invalid qhandle"	-2147483135
TSDB_CODE_TSC_INVALID_TIME_STAMP	0	0x0202	"Invalid combination of client/service time"	-2147483134
TSDB_CODE_TSC_INVALID_VALUE	0	0x0203	"Invalid value in client"	-2147483133
TSDB_CODE_TSC_INVALID_VERSION	0	0x0204	"Invalid client version"	-2147483132
TSDB_CODE_TSC_INVALID_IE	0	0x0205	"Invalid client ie"	-2147483131
TSDB_CODE_TSC_INVALID_FQDN	0	0x0206	"Invalid host name"	-2147483130
TSDB_CODE_TSC_INVALID_USER_LENGTH	0	0x0207	"Invalid user name"	-2147483129
TSDB_CODE_TSC_INVALID_PASS_LENGTH	0	0x0208	"Invalid password"	-2147483128
TSDB_CODE_TSC_INVALID_DB_LENGTH	0	0x0209	"Database name too long"	-2147483127
TSDB_CODE_TSC_INVALID_TABLE_ID_LENGTH	0	0x020A	"Table name too long"	-2147483126
TSDB_CODE_TSC_INVALID_CONNECTION	0	0x020B	"Invalid connection"	-2147483125
TSDB_CODE_TSC_OUT_OF_MEMORY	0	0x020C	"System out of memory"	-2147483124
TSDB_CODE_TSC_NO_DISKSPACE	0	0x020D	"System out of disk space"	-2147483123
TSDB_CODE_TSC_QUERY_CACHE_ERASED	0	0x020E	"Query cache erased"	-2147483122
TSDB_CODE_TSC_QUERY_CANCELLED	0	0x020F	"Query terminated"	-2147483121
TSDB_CODE_TSC_SORTED_RES_TOO_MANY	0	0x0210	"Result set too large to be sorted"	-2147483120
TSDB_CODE_TSC_APP_ERROR	0	0x0211	"Application error"	-2147483119
TSDB_CODE_TSC_ACTION_IN_PROGRESS	0	0x0212	"Action in progress"	-2147483118
TSDB_CODE_TSC_DISCONNECTED	0	0x0213	"Disconnected from service"	-2147483117
TSDB_CODE_TSC_NO_WRITE_AUTH	0	0x0214	"No write permission"	-2147483116
TSDB_CODE_MND_MSG_NOT_PROCESSED	0	0x0300	"Message not processed"	-2147482880
TSDB_CODE_MND_ACTION_IN_PROGRESS	0	0x0301	"Message is progressing"	-2147482879
TSDB_CODE_MND_ACTION_NEED_REPROCESSED	0	0x0302	"Messag need to be reprocessed"	-2147482878
TSDB_CODE_MND_NO_RIGHTS	0	0x0303	"Insufficient privilege for operation"	-2147482877
TSDB_CODE_MND_APP_ERROR	0	0x0304	"Unexpected generic error in mnode"	-2147482876
TSDB_CODE_MND_INVALID_CONNECTION	0	0x0305	"Invalid message connection"	-2147482875
TSDB_CODE_MND_INVALID_MSG_VERSION	0	0x0306	"Incompatible protocol version"	-2147482874
TSDB_CODE_MND_INVALID_MSG_LEN	0	0x0307	"Invalid message length"	-2147482873

TSDB_CODE_MND_INVALID_MSG_TYPE	0	0x0308	"Invalid message type"	-2147482872
TSDB_CODE_MND_TOO_MANY_SHELL_CONNS	0	0x0309	"Too many connections"	-2147482871
TSDB_CODE_MND_OUT_OF_MEMORY	0	0x030A	"Out of memory in mnode"	-2147482870
TSDB_CODE_MND_INVALID_SHOWOBJ	0	0x030B	"Data expired"	-2147482869
TSDB_CODE_MND_INVALID_QUERY_ID	0	0x030C	"Invalid query id"	-2147482868
TSDB_CODE_MND_INVALID_STREAM_ID	0	0x030D	"Invalid stream id"	-2147482867
TSDB_CODE_MND_INVALID_CONN_ID	0	0x030E	"Invalid connection id"	-2147482866
TSDB_CODE_MND_SDB_OBJ_ALREADY_THERE	0	0x0320	"Object already there"	-2147482848
TSDB_CODE_MND_SDB_ERROR	0	0x0321	"Unexpected generic error in sdb"	-2147482847
TSDB_CODE_MND_SDB_INVALID_TABLE_TYPE	0	0x0322	"Invalid table type"	-2147482846
TSDB_CODE_MND_SDB_OBJ_NOT_THERE	0	0x0323	"Object not there"	-2147482845
TSDB_CODE_MND_SDB_INVAID_META_ROW	0	0x0324	"Invalid meta row"	-2147482844
TSDB_CODE_MND_SDB_INVAID_KEY_TYPE	0	0x0325	"Invalid key type"	-2147482843
TSDB_CODE_MND_DNODE_ALREADY_EXIST	0	0x0330	"DNode already exists"	-2147482832
TSDB_CODE_MND_DNODE_NOT_EXIST	0	0x0331	"DNode does not exist"	-2147482831
TSDB_CODE_MND_VGROUP_NOT_EXIST	0	0x0332	"VGroup does not exist"	-2147482830
TSDB_CODE_MND_NO_REMOVE_MASTER	0	0x0333	"Master DNode cannot be removed"	-2147482829
TSDB_CODE_MND_NO_ENOUGH_DNODES	0	0x0334	"Out of DNodes"	-2147482828
TSDB_CODE_MND_CLUSTER_CFG_INCONSISTENT	0	0x0335	"Cluster cfg inconsistent"	-2147482827
TSDB_CODE_MND_INVALID_DNODE_CFG_OPTION	0	0x0336	"Invalid dnode cfg option"	-2147482826
TSDB_CODE_MND_BALANCE_ENABLED	0	0x0337	"Balance already enabled"	-2147482825
TSDB_CODE_MND_VGROUP_NOT_IN_DNODE	0	0x0338	"Vgroup not in dnode"	-2147482824
TSDB_CODE_MND_VGROUP_ALREADY_IN_DNODE	0	0x0339	"Vgroup already in dnode"	-2147482823
TSDB_CODE_MND_DNODE_NOT_FREE	0	0x033A	"Dnode not avaliable"	-2147482822
TSDB_CODE_MND_INVALID_CLUSTER_ID	0	0x033B	"Cluster id not match"	-2147482821
TSDB_CODE_MND_NOT_READY	0	0x033C	"Cluster not ready"	-2147482820
TSDB_CODE_MND_ACCT_ALREADY_EXIST	0	0x0340	"Account already exists"	-2147482816
TSDB_CODE_MND_INVALID_ACCT	0	0x0341	"Invalid account"	-2147482815
TSDB_CODE_MND_INVALID_ACCT_OPTION	0	0x0342	"Invalid account options"	-2147482814
TSDB_CODE_MND_USER_ALREADY_EXIST	0	0x0350	"User already exists"	-2147482800
TSDB_CODE_MND_INVALID_USER	0	0x0351	"Invalid user"	-2147482799
TSDB_CODE_MND_INVALID_USER_FORMAT	0	0x0352	"Invalid user format"	-2147482798
TSDB_CODE_MND_INVALID_PASS_FORMAT	0	0x0353	"Invalid password format"	-2147482797
TSDB_CODE_MND_NO_USER_FROM_CONN	0	0x0354	"Can not get user from conn"	-2147482796

TSDB_CODE_MND_TOO_MANY_USERS	0	0x0355	"Too many users"	-2147482795
TSDB_CODE_MND_TABLE_ALREADY_EXIST	0	0x0360	"Table already exists"	-2147482784
TSDB_CODE_MND_INVALID_TABLE_ID	0	0x0361	"Table name too long"	-2147482783
TSDB_CODE_MND_INVALID_TABLE_NAME	0	0x0362	"Table does not exist"	-2147482782
TSDB_CODE_MND_INVALID_TABLE_TYPE	0	0x0363	"Invalid table type in tsdb"	-2147482781
TSDB_CODE_MND_TOO_MANY_TAGS	0	0x0364	"Too many tags"	-2147482780
TSDB_CODE_MND_TOO_MANY_TIMESERIES	0	0x0366	"Too many time series"	-2147482778
TSDB_CODE_MND_NOT_SUPER_TABLE	0	0x0367	"Not super table"	-2147482777
TSDB_CODE_MND_COL_NAME_TOO_LONG	0	0x0368	"Tag name too long"	-2147482776
TSDB_CODE_MND_TAG_ALREAY_EXIST	0	0x0369	"Tag already exists"	-2147482775
TSDB_CODE_MND_TAG_NOT_EXIST	0	0x036A	"Tag does not exist"	-2147482774
TSDB_CODE_MND_FIELD_ALREAY_EXIST	0	0x036B	"Field already exists"	-2147482773
TSDB_CODE_MND_FIELD_NOT_EXIST	0	0x036C	"Field does not exist"	-2147482772
TSDB_CODE_MND_INVALID_STABLE_NAME	0	0x036D	"Super table does not exist"	-2147482771
TSDB_CODE_MND_DB_NOT_SELECTED	0	0x0380	"Database not specified or available"	-2147482752
TSDB_CODE_MND_DB_ALREADY_EXIST	0	0x0381	"Database already exists"	-2147482751
TSDB_CODE_MND_INVALID_DB_OPTION	0	0x0382	"Invalid database options"	-2147482750
TSDB_CODE_MND_INVALID_DB	0	0x0383	"Invalid database name"	-2147482749
TSDB_CODE_MND_MONITOR_DB_FORBIDDEN	0	0x0384	"Cannot delete monitor database"	-2147482748
TSDB_CODE_MND_TOO_MANY_DATABASES	0	0x0385	"Too many databases for account"	-2147482747
TSDB_CODE_MND_DB_IN_DROPPING	0	0x0386	"Database not available"	-2147482746
TSDB_CODE_DND_MSG_NOT_PROCESSED	0	0x0400	"Message not processed"	-2147482624
TSDB_CODE_DND_OUT_OF_MEMORY	0	0x0401	"Dnode out of memory"	-2147482623
TSDB_CODE_DND_NO_WRITE_ACCESS	0	0x0402	"No permission for disk files in dnode"	-2147482622
TSDB_CODE_DND_INVALID_MSG_LEN	0	0x0403	"Invalid message length"	-2147482621
TSDB_CODE_VND_ACTION_IN_PROGRESS	0	0x0500	"Action in progress"	-2147482368
TSDB_CODE_VND_MSG_NOT_PROCESSED	0	0x0501	"Message not processed"	-2147482367
TSDB_CODE_VND_ACTION_NEED_REPROCESSED	0	0x0502	"Action need to be reprocessed"	-2147482366
TSDB_CODE_VND_INVALID_VGROUP_ID	0	0x0503	"Invalid Vgroup ID"	-2147482365
TSDB_CODE_VND_INIT_FAILED	0	0x0504	"Vnode initialization failed"	-2147482364
TSDB_CODE_VND_NO_DISKSPACE	0	0x0505	"System out of disk space"	-2147482363
TSDB_CODE_VND_NO_DISK_PERMISSIONS	0	0x0506	"No write permission for disk files"	-2147482362
TSDB_CODE_VND_NO_SUCH_FILE_OR_DIR	0	0x0507	"Missing data file"	-2147482361
TSDB_CODE_VND_OUT_OF_MEMORY	0	0x0508	"Out of memory"	-2147482360

TSDB_CODE_VND_APP_ERROR	0	0x0509	"Unexpected generic error in vnode"	-2147482359
TSDB_CODE_VND_INVALID_STATUS	0	0x0510	"Database not ready"	-2147482352
TSDB_CODE_VND_NOT_SYNCED	0	0x0511	"Database suspended"	-2147482351
TSDB_CODE_VND_NO_WRITE_AUTH	0	0x0512	"Write operation denied"	-2147482350
TSDB_CODE_TDB_INVALID_TABLE_ID	0	0x0600	"Invalid table ID"	-2147482112
TSDB_CODE_TDB_INVALID_TABLE_TYPE	0	0x0601	"Invalid table type"	-2147482111
TSDB_CODE_TDB_IVD_TB_SCHEMA_VERSION	0	0x0602	"Invalid table schema version"	-2147482110
TSDB_CODE_TDB_TABLE_ALREADY_EXIST	0	0x0603	"Table already exists"	-2147482109
TSDB_CODE_TDB_INVALID_CONFIG	0	0x0604	"Invalid configuration"	-2147482108
TSDB_CODE_TDB_INIT_FAILED	0	0x0605	"Tsdb init failed"	-2147482107
TSDB_CODE_TDB_NO_DISKSPACE	0	0x0606	"No diskspace for tsdb"	-2147482106
TSDB_CODE_TDB_NO_DISK_PERMISSIONS	0	0x0607	"No permission for disk files"	-2147482105
TSDB_CODE_TDB_FILE_CORRUPTED	0	0x0608	"Data file(s) corrupted"	-2147482104
TSDB_CODE_TDB_OUT_OF_MEMORY	0	0x0609	"Out of memory"	-2147482103
TSDB_CODE_TDB_TAG_VER_OUT_OF_DATE	0	0x060A	"Tag too old"	-2147482102
TSDB_CODE_TDB_TIMESTAMP_OUT_OF_RANGE	0	0x060B	"Timestamp data out of range"	-2147482101
TSDB_CODE_TDB_SUBMIT_MSG_MSSED_UP	0	0x060C	"Submit message is messed up"	-2147482100
TSDB_CODE_TDB_INVALID_ACTION	0	0x060D	"Invalid operation"	-2147482099
TSDB_CODE_TDB_INVALID_CREATE_TB_MSG	0	0x060E	"Invalid creation of table"	-2147482098
TSDB_CODE_TDB_NO_TABLE_DATA_IN_MEM	0	0x060F	"No table data in memory skiplist"	-2147482097
TSDB_CODE_TDB_FILE_ALREADY_EXISTS	0	0x0610	"File already exists"	-2147482096
TSDB_CODE_TDB_TABLE_RECONFIGURE	0	0x0611	"Need to reconfigure table"	-2147482095
TSDB_CODE_TDB_IVD_CREATE_TABLE_INFO	0	0x0612	"Invalid information to create table"	-2147482094
TSDB_CODE_QRY_INVALID_QHANDLE	0	0x0700	"Invalid handle"	-2147481856
TSDB_CODE_QRY_INVALID_MSG	0	0x0701	"Invalid message"	-2147481855
TSDB_CODE_QRY_NO_DISKSPACE	0	0x0702	"No diskspace for query"	-2147481854
TSDB_CODE_QRY_OUT_OF_MEMORY	0	0x0703	"System out of memory"	-2147481853
TSDB_CODE_QRY_APP_ERROR	0	0x0704	"Unexpected generic error in query"	-2147481852
TSDB_CODE_QRY_DUP_JOIN_KEY	0	0x0705	"Duplicated join key"	-2147481851
TSDB_CODE_QRY_EXCEED_TAGS_LIMIT	0	0x0706	"Tag conditon too many"	-2147481850
TSDB_CODE_QRY_NOT_READY	0	0x0707	"Query not ready"	-2147481849
TSDB_CODE_QRY_HAS_RSP	0	0x0708	"Query should response"	-2147481848
TSDB_CODE_GRANT_EXPIRED	0	0x0800	"License expired"	-2147481600
TSDB_CODE_GRANT_DNODE_LIMITED	0	0x0801	"DNode creation limited by licence"	-2147481599

TSDB_CODE_GRANT_ACCT_LIMITED	0	0x0802	"Account creation limited by license"	-2147481598
TSDB_CODE_GRANT_TIMESERIES_LIMITED	0	0x0803	"Table creation limited by license"	-2147481597
TSDB_CODE_GRANT_DB_LIMITED	0	0x0804	"DB creation limited by license"	-2147481596
TSDB_CODE_GRANT_USER_LIMITED	0	0x0805	"User creation limited by license"	-2147481595
TSDB_CODE_GRANT_CONN_LIMITED	0	0x0806	"Conn creation limited by license"	-2147481594
TSDB_CODE_GRANT_STREAM_LIMITED	0	0x0807	"Stream creation limited by license"	-2147481593
TSDB_CODE_GRANT_SPEED_LIMITED	0	0x0808	"Write speed limited by license"	-2147481592
TSDB_CODE_GRANT_STORAGE_LIMITED	0	0x0809	"Storage capacity limited by license"	-2147481591
TSDB_CODE_GRANT_QUERYTIME_LIMITED	0	0x080A	"Query time limited by license"	-2147481590
TSDB_CODE_GRANT_CPU_LIMITED	0	0x080B	"CPU cores limited by license"	-2147481589
TSDB_CODE_SYN_INVALID_CONFIG	0	0x0900	"Invalid Sync Configuration"	-2147481344
TSDB_CODE_SYN_NOT_ENABLED	0	0x0901	"Sync module not enabled"	-2147481343
TSDB_CODE_WAL_APP_ERROR	0	0x1000	"Unexpected generic error in wal"	-2147479552

13. 常见问题及反馈

13.1. 常见问题列表

1. TDengine2.0之前的版本升级到2.0及以上的版本应该注意什么？☆☆☆

2.0版在之前版本的基础上，进行了完全的重构，配置文件和数据文件是不兼容的。在升级之前务必进行如下操作：

1. 删除配置文件，执行 `sudo rm -rf /etc/taos/taos.cfg`
2. 删除日志文件，执行 `sudo rm -rf /var/log/taos/`
3. 确保数据已经不再需要的前提下，删除数据文件，执行 `sudo rm -rf /var/lib/taos/`
4. 安装最新稳定版本的 TDengine
5. 如果需要迁移数据或者数据文件损坏，请联系涛思数据官方技术支持团队，进行协助解决

2. Windows平台下JDBCDriver找不到动态链接库，怎么办？

请看为此问题撰写的技术博客。

3. 创建数据表时提示more dnodes are needed

请看为此问题撰写的技术博客。

4. 如何让TDengine crash时生成core文件？

请看为此问题撰写的技术博客。

5. 遇到错误"Unable to establish connection", 我怎么办？

客户端遇到连接故障，请按照下面的步骤进行检查：

1. 检查网络环境
   - 云服务器：检查云服务器的安全组是否打开TCP/UDP 端口6030-6042的访问权限
   - 本地虚拟机：检查网络能否ping通，尽量避免使用`localhost` 作为hostname
   - 公司服务器：如果为NAT网络环境，请务必检查服务器能否将消息返回值客户端
2. 确保客户端与服务端版本号是完全一致的，开源社区版和企业版也不能混用
3. 在服务器，执行 `systemctl status taosd` 检查*taosd*运行状态。如果没有运行，启动*taosd*
4. 确认客户端连接时指定了正确的服务器FQDN (Fully Qualified Domain Name —— 可在服务器上执行Linux命令 hostname -f获得），FQDN配置参考：一篇文章说清楚TDengine的FQDN。
5. ping服务器FQDN，如果没有反应，请检查你的网络，DNS设置，或客户端所在计算机的系统hosts文件。如果部署的是TDengine集群，客户端需要能ping通所有集群节点的FQDN。
6. 检查防火墙设置（Ubuntu 使用 ufw status，CentOS 使用 firewall-cmd --list-port），确认TCP/UDP 端口6030-6042 是打开的
7. 对于Linux上的JDBC（ODBC, Python, Go等接口类似）连接, 确保*libtaos.so*在目录*/usr/local/taos/driver*里, 并且*/usr/local/taos/driver*在系统库函数搜索路径*LD_LIBRARY_PATH*里

8. 对于Windows上的JDBC, ODBC, Python, Go等连接，确保*C:\TDengine\driver\taos.dll*在你的系统库函数搜索目录里 (建议*taos.dll*放在目录 *C:\Windows\System32*)
9. 如果仍不能排除连接故障
   - Linux 系统请使用命令行工具nc来分别判断指定端口的TCP和UDP连接是否通畅
     检查UDP端口连接是否工作：`nc -vuz {hostIP} {port}`
     检查服务器侧TCP端口连接是否工作：`nc -l {port}`
     检查客户端侧TCP端口连接是否工作：`nc {hostIP} {port}`
   - Windows 系统请使用 PowerShell 命令 Net-TestConnection -ComputerName {fqdn} -Port {port} 检测服务段端口是否访问
0. 也可以使用taos程序内嵌的网络连通检测功能，来验证服务器和客户端之间指定的端口连接是否通畅（包括TCP和UDP）：TDengine 内嵌网络检测工具使用指南。

6. 遇到错误“Unexpected generic error in RPC”或者"TDengine Error: Unable to resolve FQDN"，我怎么办?

产生这个错误，是由于客户端或数据节点无法解析FQDN(Fully Qualified Domain Name)导致。对于TAOS Shell或客户端应用，请做如下检查：

1. 请检查连接的服务器的FQDN是否正确，FQDN配置参考：一篇文章说清楚TDengine的FQDN
2. 如果网络配置有DNS server，请检查是否正常工作
3. 如果网络没有配置DNS server，请检查客户端所在机器的hosts文件，查看该FQDN是否配置，并是否有正确的IP地址
4. 如果网络配置OK，从客户端所在机器，你需要能Ping该连接的FQDN，否则客户端是无法连接服务器的

7. 虽然语法正确，为什么我还是得到 "Invalid SQL"错误

如果你确认语法正确，2.0之前版本，请检查SQL语句长度是否超过64K。如果超过，也会返回这个错误。

8. 是否支持validation queries?

9. 我可以删除或更新一条记录吗?

TDengine 目前尚不支持删除功能，未来根据用户需求可能会支持。

从 2.0.8.0 开始，TDengine 支持更新已经写入数据的功能。使用更新功能需要在创建数据库时使用 UPDATE 1 参数，之后可以使用 INSERT INTO 命令更新已经写入的相同时间戳数据。UPDATE 参数不支持 ALTER DATABASE 命令修改。没有使用 UPDATE 1 参数创建的数据库，写入相同时间戳的数据不会修改之前的数据，也不会报错。

另需注意，在 UPDATE 设置为 0 时，后发送的相同时间戳的数据会被直接丢弃，但并不会报错，而且仍然会被计入 affected rows （所以不能利用 INSERT 指令的返回信息进行时间戳查重）。这样设计的主要原因是，TDengine 把写入的数据看做一个数据流，无论时间戳是否出现冲突，TDengine 都认为产生数据的原始设备真实地产生了这样的数据。UPDATE 参数只是控制这样的流数据在进行持久化时要怎样处理——UPDATE 为 0 时，表示先写入的数据覆盖后写入的数据；而 UPDATE 为 1 时，表示后写入的数据覆盖先写入的数据。这种覆盖关系如何选择，取决于对数据的后续使用和统计中，希望以先还是后生成的数据为准。

10. 我怎么创建超过1024列的表?

使用2.0及其以上版本，默认支持1024列；2.0之前的版本，TDengine最大允许创建250列的表。但是如果确实超过限值，建议按照数据特性，逻辑地将这个宽表分解成几个小表。

11. 最有效的写入数据的方法是什么?

批量插入。每条写入语句可以一张表同时插入多条记录，也可以同时插入多张表的多条记录。

12. Windows系统下插入的nchar类数据中的汉字被解析成了乱码如何解决?

Windows下插入nchar类的数据中如果有中文，请先确认系统的地区设置成了中国（在Control Panel里可以设置），这时cmd中的taos客户端应该已经可以正常工作了；如果是在IDE里开发Java应用，比如Eclipse， Intellij，请确认IDE里的文件编码为GBK（这是Java默认的编码类型），然后在生成Connection时，初始化客户端的配置，具体语句如下：

```
Class.forName("com.taosdata.jdbc.TSDBDriver");
Properties properties = new Properties();
properties.setProperty(TSDBDriver.LOCALE_KEY, "UTF-8");
Connection = DriverManager.getConnection(url, properties);
```

13. TDengine GO windows驱动的如何编译?

请看为此问题撰写的技术博客

14. JDBC报错：the excuted SQL is not a DML or a DDL?

请更新至最新的JDBC驱动

```
<dependency>
  <groupId>com.taosdata.jdbc</groupId>
  <artifactId>taos-jdbcdriver</artifactId>
  <version>2.0.27</version>
</dependency>
```

15. taos connect failed, reason: invalid timestamp

常见原因是服务器和客户端时间没有校准，可以通过和时间服务器同步的方式（Linux 下使用 ntpdate 命令，Windows 在系统时间设置中选择自动同步）校准。

16. 表名显示不全

由于 taos shell 在终端中显示宽度有限，有可能比较长的表名显示不全，如果按照显示的不全的表名进行相关操作会发生 Table does not exist 错误。解决方法可以是通过修改 taos.cfg 文件中的设置项 maxBinaryDisplayWidth，或者直接输入命令 set max_binary_display_width 100。或者在命令结尾使用 \G 参数来调整结果的显示方式。

17. 如何进行数据迁移?

TDengine是根据hostname唯一标志一台机器的，在数据文件从机器A移动机器B时，注意如下两件事：

- 2.0.0.0 至 2.0.6.x 的版本，重新配置机器B的hostname为机器A的hostname。
- 2.0.7.0 及以后的版本，到/var/lib/taos/dnode下，修复dnodeEps.json的dnodeId对应的FQDN，重启。确保机器内所有机器的此文件是完全相同的。
- 1.x 和 2.x 版本的存储结构不兼容，需要使用迁移工具或者自己开发应用导出导入数据。

18. 如何在命令行程序 taos 中临时调整日志级别

为了调试方便，从 2.0.16 版本开始，命令行程序 taos 新增了与日志记录相关的两条指令：

```
ALTER LOCAL flag_name flag_value;
```

其含义是，在当前的命令行程序下，修改一个特定模块的日志记录级别（只对当前命令行程序有效，如果 taos 命令行程序重启，则需要重新设置）：

- flag_name 的取值可以是：debugFlag，cDebugFlag，tmrDebugFlag，uDebugFlag，rpcDebugFlag
- flag_value 的取值可以是：131（输出错误和警告日志），135（ 输出错误、警告和调试日志），143（ 输出错误、警告、调试和跟踪日志）

```
ALTER LOCAL RESETLOG;
```

其含义是，清空本机所有由客户端生成的日志文件。

19. 时间戳的时区信息是怎样处理的?

TDengine 中时间戳的时区总是由客户端进行处理，而与服务端无关。具体来说，客户端会对 SQL 语句中的时间戳进行时区转换，转为 UTC 时区（即 Unix 时间戳——Unix Timestamp）再交由服务端进行写入和查询；在读取数据时，服务端也是采用 UTC 时区提供原始数据，客户端收到后再根据本地设置，把时间戳转换为本地系统所要求的时区进行显示。

客户端在处理时间戳字符串时，会采取如下逻辑：

1. 在未做特殊设置的情况下，客户端默认使用所在操作系统的时区设置。
2. 如果在 taos.cfg 中设置了 timezone 参数，则客户端会以这个配置文件中的设置为准。
3. 如果在 C/C++/Java/Python 等各种编程语言的 Connector Driver 中，在建立数据库连接时显式指定了 timezone，那么会以这个指定的时区设置为准。例如 Java Connector 的 JDBC URL 中就有 timezone 参数。
4. 在书写 SQL 语句时，也可以直接使用 Unix 时间戳（例如 `1554984068000`）或带有时区的时间戳字符串，也即以 RFC 3339 格式（例如 `2013-04-12T15:52:01.123+08:00`）或 ISO-8601 格式（例如 `2013-04-12T15:52:01.123+0800`）来书写时间戳，此时这些时间戳的取值将不再受其他时区设置的影响。

20. TDengine 都会用到哪些网络端口?

在 TDengine 2.0 版本中，会用到以下这些网络端口（以默认端口 6030 为前提进行说明，如果修改了配置文件中的设置，那么这里列举的端口都会出现变化），管理员可以参考这里的信息调整防火墙设置：

协议	默认端口	用途说明	修改方法
TCP	6030	客户端与服务端之间通讯。	由配置文件设置 serverPort 决定。
TCP	6035	多节点集群的节点间通讯。	随 serverPort 端口变化。
TCP	6040	多节点集群的节点间数据同步。	随 serverPort 端口变化。
TCP	6041	客户端与服务端之间的 RESTful 通讯。	随 serverPort 端口变化。
TCP	6042	Arbitrator 的服务端口。	因 Arbitrator 启动参数设置变化。
TCP	6060	企业版内 Monitor 服务的网络端口。	
UDP	6030-6034	客户端与服务端之间通讯。	随 serverPort 端口变化。
UDP	6035-6039	多节点集群的节点间通讯。	随 serverPort 端口变化。

13.2. 问题反馈

如果 FAQ 中的信息不能够帮到您，需要 TDengine 技术团队的技术支持与协助，请将以下两个目录中内容打包：

1. /var/log/taos （如果没有修改过默认路径）
2. /etc/taos

附上必要的问题描述，包括使用的 TDengine 版本信息、平台环境信息、发生该问题的执行操作、出现问题的表征及大概的时间，在 GitHub 提交Issue。

为了保证有足够的debug信息，如果问题能够重复，请修改/etc/taos/taos.cfg文件，最后面添加一行“debugFlag 135"(不带引号本身），然后重启taosd, 重复问题，然后再递交。也可以通过如下SQL语句，临时设置taosd的日志级别。

```
alter dnode <dnode_id> debugFlag 135;
```

但系统正常运行时，请一定将debugFlag设置为131，否则会产生大量的日志信息，降低系统效率。