TDengine 用户手册

(适用版本 1.6)

版权申明

目 录

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的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处理产品 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。

TDengine 提供的是时序数据处理的全栈技术解决方案，包含了时序数据库、消息队列、缓存、流式计算、订阅等系列功能，从而不用再集成 Kafka,Redis,Hadoop,Spark,Hbase 等软件，大幅降低研发和运维成本。在时序空间大数据处理上，TDengine 具有如下优势：

- **快十倍的插入和查询**：定义了创新的时序数据存储结构，通过采用无锁设计和多核技术，TDengine 让数据插入和查询的速度比现有通用数据库提高了十倍以上。
- **超融合**：将大数据处理所需要的数据库、消息队列、缓存、流式计算等功能融合一起，应用无需再集成这些功能的软件，大幅降低应用开发难度。
- **更高的水平扩展能力**：通过先进的集群设计，保证了系统处理能力的水平扩展，而且让数据库不再依赖昂贵的硬件和存储设备，不存在任何单点瓶颈和故障。
- **极低的资源消耗**：整个完整安装包才 1.5M，计算资源不到通用方案的 1/5。通过列式存储和先进的压缩算法，存储空间不到传统数据库的 1/10。

- **零学习成本**：使用标准的 SQL 语法，并支持 JDBC，ODBC，REST 接口，应用 API 与 MySQL 高度相似，让学习成本几乎为零。
- **零运维管理成本**：追求极致的用户体验，将复杂的运维工作完全智能化。无需分库分表，数据备份、数据恢复完全自动；扩容、升级、IDC 机房迁移轻松完成。

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整体成本降至现有的 1/5。同样的硬件资源，TDengine 能将系统处理能力和容量增加五倍以上。但需要指出的是，因充分利用了时序数据的特点，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 运行在 64 位 Linux 系统之上，也可直接安装并运行在国内外主流公有云平台(阿里、腾讯、亚马逊 AWS)上提供的系统镜像之上(例如：Ubuntu 16.04 及 Centos 7 等)。目前 TDengine 使用 systemd 来启动和停止服务后台程序，请用 which 命令来检测系统中是否存在 systemd.

2.1 安装 TDengine

安装步骤如下：

1. **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 taos-xxxxxxxxx.tar.gz
   ```

   其中 xxxxxxx 需要替换为实际版本的字符串。

2. **执行 install.sh 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.sh：主安装脚本，用于安装服务端及客户端程序
   install_client.sh：仅安装客户端程序的脚本
   taos.tar.gz：主安装包
   code：示例代码及部分脚本
   driver：TDengine 客户端 driver
   release_note：版本更新内容总结
   ```

   运行 install.sh 进行安装。

   安装过程中，会提示安装的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意一节点的 IP 地址；如果不是，则会新创建一个集群。

2.2 启动 TDengine 服务

安装后，可以根据应用场景，编辑配置文件/etc/taos/taos.cfg 来修改配置参数（配置参数详见**错误!未找到引用源。**章），然后在 Linux 终端执行如下命令启动服务：

```
sudo systemctl start taosd
```

启动后，可以在 Linux 终端执行 TDengine 命令行程序(Command Line Interface, CLI) taos 来验证是否一切正常。

2.3 TDengine 命令行程序 taos

在 Linux 终端执行命令行程序 taos 后，应该可以看到如下类似信息：

```
Welcome to the TDengine shell, server version: 0.9.2, client version:0.9.2
Copyright (c) 2017 by TAOS Data, Inc. All rights reserved.
```

如果看到上述信息，表示一切正常。如果失败，则会打印错误消息出来（请参考 FAQ 来解决终端链接服务端失败的问题）

用户可以通过命令行程序 taos 对 TDengine 进行全面的管理（创建数据库、创建数据表、插入数据、执行查询等），所有的操作均通过标准 SQL 语句完成。DBA 也可以从 CLI 进行增删用户账号、增删集群节点，查看节点状态等操作。示例如下：

```
taos> create database testdb;
Query OK,1 row(s) are affected (0.001723 s)

taos> use testdb;
Database changed.

taos> create table meter1(ts timestamp, speed int);
Query OK,1 row(s) are affected (0.002123 s)

taos> insert into meter1 values(now, 100);
Query OK,1 row(s) are affected (0.001746 s)

taos> select * from meter1;
         ts            |   speed  |
==================================
 17-08-04 09:58:03:907|       100|
Query OK,1 row(s) in set (0.001718 s)
```

命令行参数：taos 支持命令行参数，部分重要的如下：

- -c 指定配置文件所在目录路径。

- -d 指定所要连接的数据库。
- -f 无需进入 Shell 即可运行的 SQL 脚本。
- -h 数据库的服务 IP 地址（可选参数，集群中任何一台机器的 public IP）
- -p 手动输入登录密码密码。如不使用此参数，则自动使用 root 账户的初始密码 taosdata 登录。
- -P 手动指定所要连接的服务端 TCP/IP 的端口号。
- -r 将查询结果集中的时间以无符号长整型输出。
- -s 无需进入 Shell 运行的临时 SQL 命令，命令需用引号括起。运行多条语句时，各语句以分号分隔。
- -t 设置客户端时间戳。
- -u 指定登录用户名，缺省的 DBA 用户名是 root。
- -V 打印客户端版本号到命令行。
- -? 列出所有支持的命令行参数

Tips:

- 使用上下光标键查看历史输入的命令。
- 修改用户密码：在 shell 中使用 alter user 命令，缺省密码为 taosdata
- ctrl+c 终止正在进行中的查询。
- 执行 RESET QUERY CACHE 可清除本地缓存的元数据。
- 批量执行 SQL 语句。可以将一系列的 shell 命令（以；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 SQL 语句。

3 TDengine 数据模型

在典型的物联网场景中，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，其采集的数据类似如下的表格：

Device ID	Time Stamp	Value 1	Value 2	Value 3	Tag 1	Tag 2
D1001	1538548685000	10.3	219	0.31	Red	Tesla
D1002	1538548686000	10.2	220	0.23	Blue	BMW
D1003	1538548686500	11.5	221	0.35	Black	Honda
D1004	1538548687000	13.4	223	0.29	Red	Volvo
D1001	1538548695000	12.6	218	0.33	Red	Tesla
D1003	1538548696600	11.8	221	0.28	Black	Honda
设备ID	时间戳	采集量			静态标签	

TDengine 采用的仍然是传统的关系型数据库的模型。用户需要根据应用场景，创建一到多个库，然后在每个库里创建多张表，创建表时需要定义 Schema。

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。这种设计能保证一个采集点的数据在存储介质上是一块一块连续的，大幅减少随机读取操作，成数量级的提升读取和查询速度。而且由于不同数据采集设备产生数据的过程完全独立，每个设备只产生属于自己的数据，一张表也就只有一个写入者。这样每个表就可以采用无锁方式来写，写入速度就能大幅提升。同时，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。

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 value1，value2，value3)，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。数据是用列式存储方式保存。

对于同一类型的采集点，为保证 Schema 的一致性，而且为便于聚合统计操作，可以先定义超级表 STable(详见第 6 章)，然后再定义表。每个采集点往往还有静态标签信息(如上表中的 Tag 1，Tag 2)，比如设备型号、颜色等，这些静态信息不会保存在存储采集数据的数据节点中，而是通过超级表保存在元数据节点中。这些静态标签信息将作为过滤条件，用于采集点之间的数据聚合统计操作。

不同的数据采集点往往具有不同的数据特征，包括数据采集频率高低，数据保留时间长短，备份数目，单个字段大小等等。为让各种场景下 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。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、数据备份的份数、cache 大小、文件块大小、是否压缩等多种参数(详见第 11.1 章)。

TDengine 对库的数量、超级表的数量以及表的数量没有做任何限制，而且其多少不会对性能产生影响，应用按照自己的场景创建即可。

与 NoSQL 的各种引擎相比，由于应用需要事先设计定义 schema，插入数据的灵活性降低。但对于物联网、金融等典型的时序数据场景，业务特点决定了 schema 会很少变更，因此事先设计 schema 的设计就不成问题，而换来的是，TDengine 采用结构化数据来进行处理的方式将让查询、分析的性能成数量级的提升。

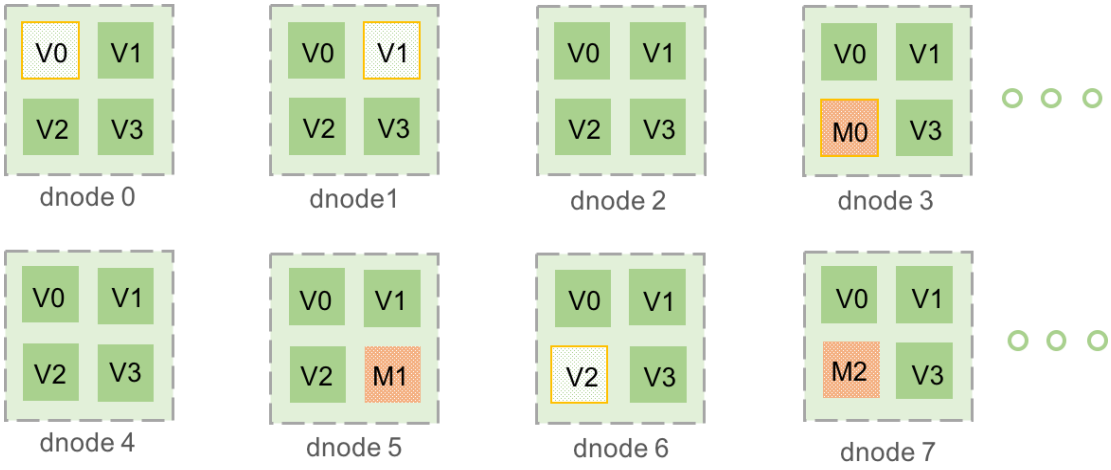
4 TDengine 系统结构

4.1 集群与主要逻辑单元

TDengine 的设计原则是是基于单个硬件、软件系统不可靠、未来发生故障无法百分之百的避免的假设，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，基因中就是完全去中心化的，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。在这种设计下，系统迁移扩容的工作非常简单，系统的运维工作量和成本显著下降，只要数据副本数大于一，无论是硬软件的升级、还是 IDC 的迁移等都无需停止集群的服务，系统的正常运行都可以得到保证。

如下图以八个物理节点的系统作为示例，对 TDengine 系统的基本概念进行介绍。



物理节点(dnode)： dnode 是一台运行 taosd 服务的物理节点，是集群中的一物理服务器或云平台上的一虚拟机。为安全以及通讯效率，一个物理节点可配置两张网卡，或两个 IP 地址。其中一张网卡用于集群内部通讯，其 IP 地址为 **privateIp**，另外一张网卡用于与集群外部应用的通讯，其 IP 地址为 **publicIp**。在一些云平台（如阿里云），对外的 IP 地址是映射过来的，因此 publicIp 还有一个对应的内部 IP 地址 **internalIp**（与 privateIp 不同）。对于只有一个 IP 地址的物理节点，publicIp，privateIp 以及 internalIp 都是同一个地址，没有任何区别。一个 dnode 上有而且只有一个 taosd 实例运行。

虚拟数据节点(vnode)：vnode 是在物理节点 dnode 之上的在 taosd 内部虚拟化的基础逻辑单元，时序数据写入、存储、查询等操作逻辑都在虚拟节点中进行(图中 V)，采集的时序数据逻辑上存储在 vnode 上。一个 vnode 包含固定数量的表。当创建一张新表时，系统会检查是否需要创建新的 vnode。一个物理节点上能创建的 vnode 的数量取决于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。

虚拟数据节点组(vgroup)：vgroup 是一组位于不同物理节点的 vnode 组成的一个虚拟数据节点组(如上图 dnode0 中的 V0，dnode1 中的 V1，dnode6 中的 V2 属于同一个虚拟节点组)，根据系统配置的副本 replica 数量来确定 vgroup 中的虚拟节点数。归属于同一个 vgroup 的虚拟节点采取 master/slave 的方式进行管理。写操作只能在 master 上进行，系统采用异步复制的方式将数据同步到 slave，这样确保了一份数据在多个物理节点上有拷贝。如果 master 节点发生故障，vgroup 中其他节点监测到后，将重新选举 vgroup 里的 master，新的 master 能继续处理数据请求，从而保证系统运行的可靠性。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 个物理节点。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，获得同样的数据高可靠性。

虚拟管理节点(mnode)：mnode 是管理节点，负责所有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，虚拟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(最多不超过 5 个） mnode，它们自动构建成为一个管理节点集群(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。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，每个 dnode 通过内部消息交互自动获取整个集群中所有 mnode 的 IP 地址。

taosc：taosc 是 TDengine 给应用提供的驱动程序（driver），负责处理与集群的接口交互，内嵌于 JDBC、ODBC driver 中，或者 C 语言连接库里。应用都是通过 taosc 而不是直接连接集群中的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虚拟节点；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，ODBC，C/C++接口而言，这个模块是在应用所处的计算机上运行，但消耗的资源很小。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。

对外服务地址：TDengine 集群可以容纳单台、多台甚至几千台物理节点。应用只需要向集群中任何一个物理节点的 publicIp 发起连接即可。通过命令行 CLI 启动应用 taos 时，可以通过选项-h 配置 publicIp。

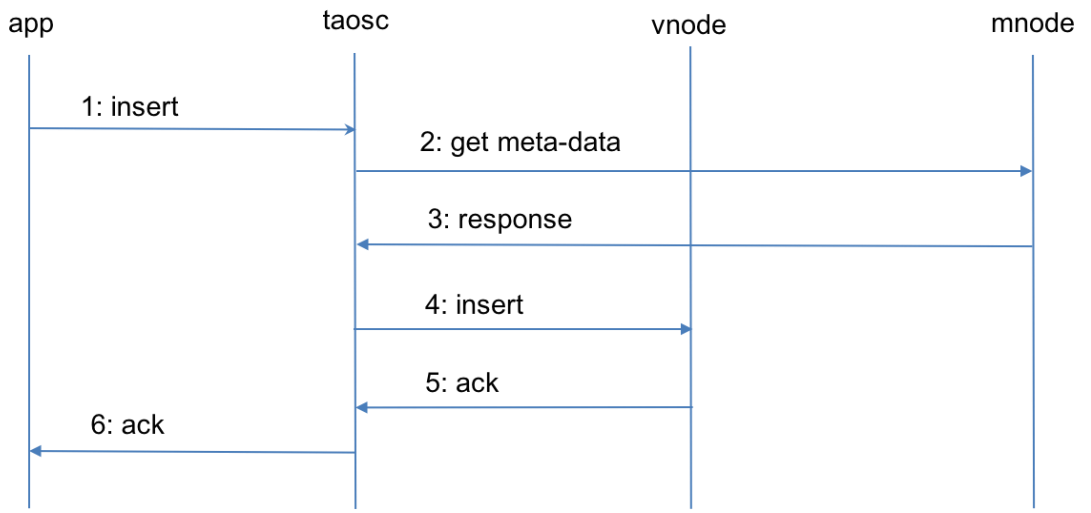
masterIp/secondIp：masterIp/secondIp 是该 dnode 所属的集群中某个 dnode 的 privateIp,可以通过这个 ip 与集群进行交互。每一个 dnode 都需要配置一个 masterIp。dnode 启动后，将对配置的 masterIp 发起加入集群的连接请求。masterIp 是已经创建的集群中的任何一个节点的 privateIp，对于集群中的第一个节点，就是它自己的 privateIp。为保证连接成功，每个 dnode 还可配置一个备用的 secondIp，该 IP 地址也是已创建的集群中的任何一个其他节点的 privateIp。如果一个节点连接 masterIp 失败,它将试图链接 secondIp。

dnode 启动后，会获知集群的 mnode IP 列表，并且定时向 mnode 发送状态信息。

vnode 与 mnode 只是逻辑上的划分，都是执行程序 taosd 里的不同线程，无需安装不同的软件，做任何特殊的配置。最小的系统配置就是一个物理节点,vnode,mnode 和 taosc 都存在而且都正常运行，但单一节点无法保证系统的高可靠。

4.2 一典型的操作流程

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。



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，看是有保存有该表的 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 IP 地址，如果副本数为 N，就有 N 组 vnode ID/IP)。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 IP 地址，因此会直接向配置的集群对外服务的 IP 地址发起请求。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 的 IP 地址列表（如果有多个 dnodes，mnode 的 IP 地址可以有多个），这样 taosc 会重新向新的 mnode 的 IP 地址发出获取 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/IP 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 会缓存住 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，对于应用来说吴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。

4.3 数据分区

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 将一个物理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 可存储多张表的数据，一个 vnode 可容纳的表的数目由配置参数 tables 指定，缺省为 1000。设计上，一个 vnode 里所有的表都属于同一个 DB。因此一个数据库 DB 需要的 vnode 或 vgroup 的个数等于：数据库表的数目/tables。

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。

参数 tables 的设置需要考虑具体场景，创建 DB 时，可以个性化指定该参数。该参数不宜过大，也不宜过小。过小，极端情况，就是每个数据采集点一个 vnode，这样导致系统数据文件过多。过大，虚拟化带来的优势就会丧失。给定集群计算资源的情况下，整个系统 vnode 的个数应该是 CPU 核的数目的 8 倍以上。

4.4 负载均衡

每个 dnode(物理节点)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。负载均衡操作结束后，应用也无需重启，将自动连接新的 vnode。

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 如果副本数大于一，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 的副本数大于一，系统也将自动在其他 dnode 上创建新的 mnode，以保证 mnode 的副本数。

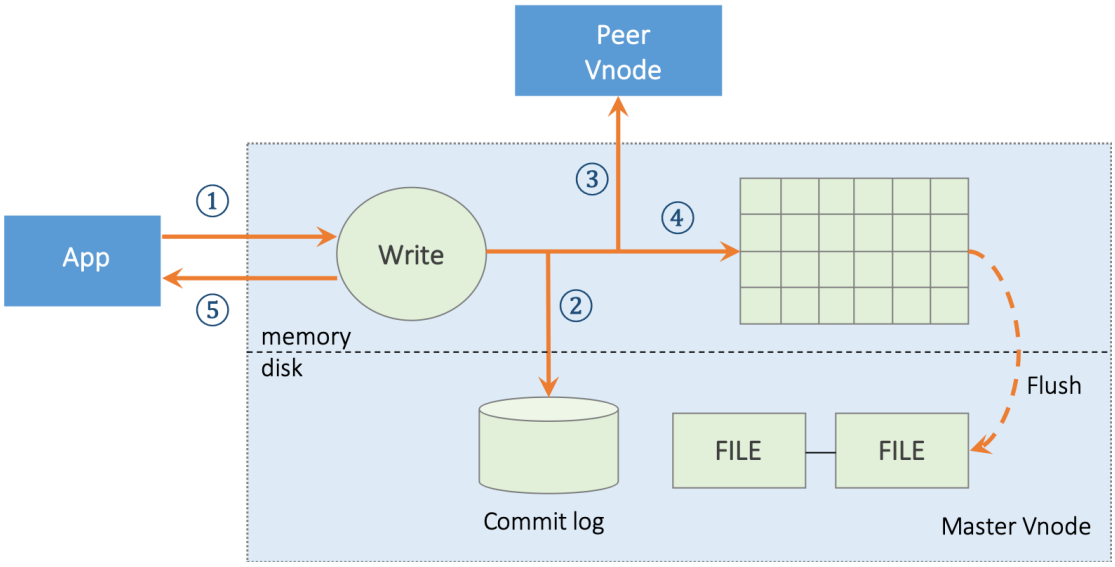
4.5 数据写入与同步流程

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。

4.5.1 Master Vnode 写入流程

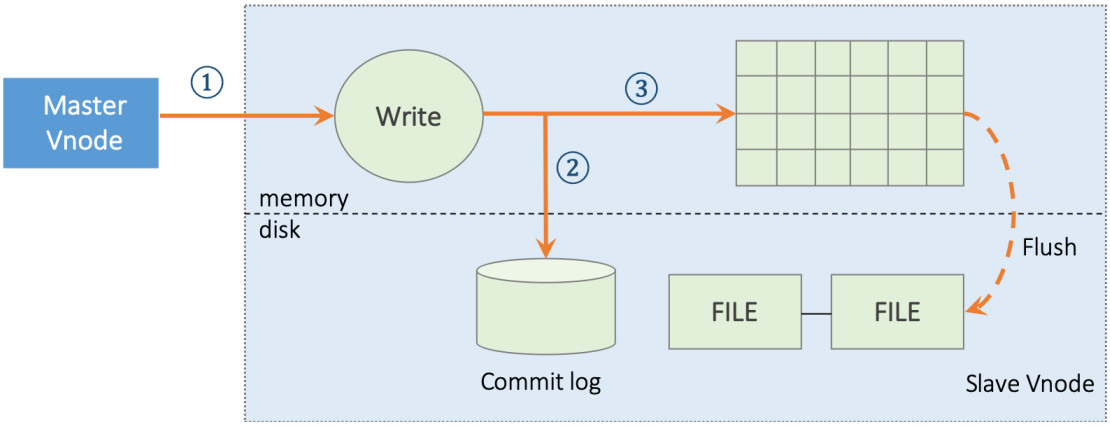
Master Vnode 遵循下面的写入流程：



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 clog 打开（设置为 1），vnode 将把该请求的原始数据包写入数据库日志(Write Ahead Log)submitxx.log，以保证 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；
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 slave vnodes，该转发包带有上一次写入的版本号(last version)
4. 写入内存，按照列式存储方式写入；
5. Master vnode 返回确认信息给应用，表示写入成功。

4.5.2 Slave Vnode 写入流程

对于 slave vnode，写入流程是：



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 clog 打开，vnode 将把该请求的原始数据包写入日志(WAL)；
3. 写入内存，按照列式存储方式写入。

与 Master vnode 相比，Slave 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。

4.5.3 异步复制、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，可以让一个虚拟节点组内，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。

另外一方面，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，mater 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。

4.5.4 数据同步和主从选择

Vnode 会保持一个数据版本号(Last 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。每次写入一条记录，这个版本号将增一。

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 链接，并互相交换 status，其中包括 last version 和自己的角色。通过 status 的交换，系统进行如下的选主流程：

- 如果某节点声称自己是 Master，那么其他节点就将自己角色设为 slave
- 如果 Master 节点的 last version 不是最高的，立即将自己角色设为未定，而且通过 status 发送给其他虚拟节点
- 如果所有虚拟节点都在线，但角色都是未定，那么版本最高的 vnode 选为主，如果版本最高的不止一个 vnode,那就选择列表中的第一个版本最高的为 master.
- 如果有虚拟节点离线，而且在线的虚拟节点的角色都是未定，那么必须等待离线的节点上线才能进行选举操作。

- 如果 Master 离线，slave 发现还有超过半数的虚拟节点仍然在线，那么具有最高版本的 slave 选为主，如果最高版本不止一个，那么选择列表中的第一个为主。
- 任何时候，如果超过半数的虚拟节点不在线，系统处于无主状态，无法提供服务。

当收到一条记录插入请求，Master vnode 转发给 Slave vnode 时，会把它当前的 Last Version 带上。无论是收到新纪录转发包，还是收到 status 消息，Slave 将把自己的 last version 与 Master 的 last version 进行对比。如果不一致，Slave 将马上启动同步过程。

同步进行中，Master 会继续会提供写入服务，不仅把不一致的历史数据发送给 slave, 还会把最新记录转发给 slave， slave 最终与 master 完全一致。为避免同步消耗过多资源，而影响正常的数据插入流程，系统对同时进行的同步数量进行了限制。超过限定的数量，新的同步将被放入一个队列，等有资源时，才会启动。

4.6 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。

TDengine 将内存池按块划分进行管理，数据在内存块里按照列式存储。每个 vnode 有自己独立的内存池，vnode 之间完全隔离。一个 vnode 的内存池是在 vnode 创建时按块分配好的，而且每个内存块按照先进先出的原则进行管理。一张表所需要的内存块是从 vnode 的内存池中进行分配的，块的大小由系统配置参数 cache 决定。每张表最大内存块的数目由配置参数 tblocks 决定，每张表平均的内存块的个数由配置参数 ablocks 决定。因此对于一个 vnode，总的内存大小为：cache*ablocks*tables。内存块参数 cache 不宜过小，一个 cache block 需要能存储至少几十条以上记录，才会有效率。参数 ablocks 最小为 2，保证每张表平均至少能分配两个内存块。

4.7 持久化存储

TDengine 中有两种机制让缓存中的数据写入硬盘进行持久化存储：

1. **时间驱动的落盘**：TDengine 服务会定时将 vnode 缓存中的数据写入到硬盘上，默认为一个小时落一次盘。落盘间隔可在配置文件 taos.cfg 中通过参数 commime 配置。

2．**数据驱动的落盘**：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。数据驱动的落盘会刷新定时落盘的时间。

TDengine 在数据落盘时会打开新的数据库日志文件，在落盘后则会删除老的数据库日志文件，避免日志文件无限制的增长。

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，将被系统将自动删除，释放存储空间。

给定 days 与 keep 两个参数，一个 vnode 总的数据文件数为：keep/days。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，这两个参数在系统初始化之前配置，一但设置后，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 rows（每块记录条数）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一定的 threshold，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。

4.8 多级存储

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，即根据数据文件的新老程度存储在不

同的存储介质上。比如最新的数据存储在 SSD 上，超过一周的数据存储在本地硬盘上，超过 4 周的数据存储在网络存储设备上，这样来降低存储成本，而又保证高效的访问数据。数据在不同存储介质上的移动是由系统自动完成的，对应用是完全透明的。数据的分级存储也是通过系统参数 dataDir 来配置。

dataDir 的配置格式如下：

```
dataDir  data_path  [tier_level]
```

其中 data_path 为挂载点的文件夹路径，tier_level 为介质存储等级。介质存储等级越高，盛放数据文件越老。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。TDengine 最多支持 3 级存储，所以 tier_level 的取值为 0、1 和 2。在配置 dataDir 时，必须存在且只有一个挂载路径不指定 tier_level，称之为特殊挂载盘（路径）。该挂载路径默认为 0 级存储介质，且包含特殊文件链接，不可被移除，否则会对写入的数据产生毁灭性影响。

假设一物理节点有六个可挂载的硬盘/mnt/disk1、/mnt/disk2、…、/mnt/disk6，其中 disk1 和 disk2 需要被指定为 0 级存储介质，disk3 和 disk4 为 1 级存储介质，disk5 和 disk6 为 2 级存储介质。disk1 为特殊挂载盘，则可在/etc/taos/taos.cfg 中做如下配置：

```
dataDir /mnt/disk1/taos
dataDir /mnt/disk2/taos 0
dataDir /mnt/disk3/taos 1
dataDir /mnt/disk4/taos 1
dataDir /mnt/disk5/taos 2
dataDir /mnt/disk6/taos 2
```

挂载的盘也可以是非本地的网络盘，只要系统能访问即可。

5 TDengine 基本 SQL 语法

TDengine 提供类似 SQL 语法，用户可以通过在控制台程序中使用 SQL 语句操纵数据库，也可以通过应用程序接口（Application Programming Interface, API）来执行 SQL 语句。

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

5.1 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
- 关键字 now 是服务器的当前时间。在集群环境中，该时间是写入主节点的时间。
- 插入记录时，如果时间戳为 0，插入数据时使用服务器当前时间。
- Epoch Time：时间戳也可以是一个长整数，表示从 1970-01-01 08:00:00.000 开始的毫秒数。
- 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：a(毫秒)，s(秒)，m(分)，h(小时)，d(天)，w(周)，n(月)，y(年)。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。

5.2 数值类型

在 TDengine 中，普通表的数据模型中可使用以下 10 种数据类型。

序号	类型	长度(Bytes)	说明
1	TIMESTAMP	8	时间戳。最小精度毫秒。从格林威治时间 1970-01-01 08:00:00.000 开始，计时不能早于该时间。用于表第一列，是该表的主键。
2	INT	4	整型，范围 [-2^31+1, 2^31-1], -2^31 被用作 Null 值
3	BIGINT	8	长整型，范围 [-2^59, 2^59]
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	用户定义	用于记录字符串，最长不能超过 504 bytes。binary 仅支持字符串输入，字符串两端使用单引号引用，否则英文全部自动转化为小写。使用时须指定大小，如 binary(20)定义了最长为 20 个字符的字符串，每个字符占 1byte 的存储空间。如果用户字符串超出 20 字节，将被自动截断。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]
8	TINYINT	1	单字节整型，范围 [-127, 127]
9	BOOL	1	布尔型，{true, false}
10	NCHAR	用户定义	用于记录非 ASCII 字符串，如中文字符。每个 nchar 字符占用 4bytes 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 40bytes 的空间。如用户字符串长度超出声明长度，则将被自动截断。

Tips：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。

5.3 数据库管理

- CREATE DATABASE [if not exists] <db-name> **[REPLICA <replica>] [DAYS <days>] [KEEP <keep>]**
  创建数据库。包括三个可选项：REPLICA，数据库同步备份的副本数（也可以理解为数据备份的节点数）。缺省值为 1，没有备份；DAYS，该数据库中时序数据切片的时间周期，缺省值为 10 天；KEEP，该数据库的数据保留多长天数，缺省是 3650 天（10 年），数据库自动删除超过时限的数据。更多参数将在第 11.1 章详细说明。

- USE <db-name>
  使用/切换数据库

- DROP database [if exists] <db-name>
  删除数据库。所包含的全部数据表将被删除，谨慎使用

- ALTER database <db-name> REPLICA <replica>
  修改数据库。当前版本仅支持修改 REPLICA 选项。

- SHOW DATABASES
  显示系统所有数据库

5.4 表管理

TDengine 中的表(TABLE)由一个时间戳主键和若干个数据列组成，每行为一条记录。表管理语句如下：

- CREATE TABLE [if not exists] <tb_name> (<field_name> TIMESTAMP, field_name1 field_type,…)
  创建数据表。
  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）表的每行长度不能超过 4096 字节；
  3）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。

- CREATE TABLE <tb_name> USING <stable_name> TAGS (<tag_valu e>,…)
  使用超级表 STable 创建表，关于 STable 的介绍，请参考后续第 6 章

- CREATE TABLE <tb_name> AS SELECT <select_cluase>
  按照指定的时间段连续查询及计算，并将查询结果写入表 tb_name。具体内容请参考后续章节

- DROP TABLE [if exists] <table_name>
  删除数据表

- SHOW TABLES [LIKE table_name]
  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。

  通配符匹配：
  1）'%' (百分号)匹配 0 到任意个字符；
  2）'_'下划线匹配一个字符。

- DESCRIBE <table_name>
  获取表的结构信息

- ALTER TABLE <tb_name> ADD COLUMN <field_name data_type>
  表增加列

- ALTER TABLE <tb_name> DROP COLUMN <field_name>
  表删除列

  说明：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

Tips： SQL 语句中操作的当前数据库（通过 use db_name 的方式指定）中的表不需要指定表所属数据库。如果要操作非当前数据库中的表，需要采用"库名"."表名"的方式。例如：demo.tb1，是指数据库 demo 中的表 tb1。

5.5 数据写入

- INSERT INTO <tb_name> VALUES (field_value, ...);
  向表 tb_name 中插入一条记录

- INSERT INTO <tb_name> (field1_name,) VALUES(field1_value,…)
  向表 tb_name 中插入一条记录，数据对应到指定的列。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。

- INSERT INTO <tb_name> VALUES (field1_value1,field2_value1, …) (field1_value2,field2_value2, …)…;
  向表 tb_name 中插入多条记录

- INSERT INTO <tb_name> (field1_name,…) VALUES(field1_value1,…) (field1_value2,…)
  向表 tb_name 中按指定的列插入多条记录

- INSERT INTO <tb1_name> VALUES (field1_value1, …)(field1_value2, …)… <tb2_name> VALUES (field1_value1, …)(field1_value2, …)…;
  同时向表 tb1_name 和 tb2_name 中分别插入多条记录

- INSERT INTO <tb1_name> (field1_name,…) VALUES(field1_value1,…) (field1_value2,…) <tb2_name> (field1_name,…) VALUES (field1_value1,…) (field1_value2,…)
  同时向表 tb1_name 和 tb2_name 中按列分别插入多条记录

注意：对同一张表，插入的新记录的时间戳必须递增，否则会跳过插入该条记录。如果时间戳为 0，系统将自动使用服务器当前时间作为该记录的时间戳。

IMPORT：如果需要将时间戳小于最后一条记录时间的记录写入到数据库中，可使用 IMPORT 替代 INSERT 命令，IMPORT 的语法与 INSERT 完全一样。如果同时 IMPORT 多条记录，需要保证一批记录是按时间戳排序好的。IMPORT 命令可用于导入历史数据。TDengine 对 Import 的处理效率不高，因此如果乱序导入数据的比例过高，TDengine 插入数据的效率将大幅降低，而且会有一定的数据碎片化。

5.6 数据查询

查询语法：

- SELECT {* | expr_list} FROM table_name [WHERE [condition1] AND [condition2] OR …] [LIMIT limit [,OFFSET offset]] [ORDER BY ts ASC|DESC][>> export_file]

- SELECT function_list FROM tb_name [WHERE where_condition] [LIMIT limit [, OFFSET offset]] [>> export_file]

注意事项：

1．可以使用*返回所有列，或指定列名。可以对数值列进行四则运算，可以给输出的列取列名。
2．Where 语句可以使用各种逻辑判断来过滤数值，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。
3．参数 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。
4．通过”>>"输出结果可以导出到指定文件

5．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
%	match with any char sequences	**binary** **nchar**
_	match with a single char	**binary** **nchar**

- 同时进行多个字段的范围过滤需要使用关键词 AND 进行连接不同的查询条件，暂不支持 OR 连接的查询条件。
- 针对某一字段的过滤只支持单一区间的过滤条件。例如：value>20 and value<30 是合法的过滤条件，而 Value<20 AND value<>5 是非法的过滤条件。

查询操作示例：

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))
```

- 查询表 tb1 中最近 1 小时的所有记录：

```
SELECT * FROM t1 WHERE ts>=NOW-1h
```

- 查询表 tb1 中从 2018-06-01 08:00:00.000 到 2018-06-02 08:00:00.000 时间范围，并且 clo3 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
```

5.7 SQL 函数

5.7.1 聚合函数

TDengine 支持针对数据的聚合查询。提供支持的聚合和提取函数如下表：

count
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。

语法：
```
SELECT COUNT([*|<field_name>]) FROM <tb_name> [WHERE clause]
```

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。

AVG
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。

语法：
```
SELECT AVG(<field_name>) FROM <tb_name> [WHERE clause]
```

wavg
功能说明：统计表/超级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。

语法：
```
SELECT WAVG(<field_name>) FROM <tb_name> WHERE clause
```

SUM
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。

语法：
```
SELECT SUM(<field_name>) FROM <tb_name> [WHERE clause]
```

STDDEV
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。

语法：
```
SELECT STDDEV(<field_name>) FROM <tb_name> [WHERE clause]
```

LEASTSQUARES
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。
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。

语法：
```
SELECT LEASTSQUARES(<field_name>) FROM <tb_name> [WHERE clause]
```

说明：自变量是时间戳，因变量是该列的值

5.7.2 选择函数

MIN
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。

语法：
```
SELECT MIN(<field_name>) FROM <tb_name>|<stb_name> [WHERE clause]
```

MAX
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。

语法：
```
SELECT MAX(<field_name>) FROM <tb_name>|<stb_name> [WHERE clause]
```

FIRST
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。

语法：

```
SELECT FIRST(<field_name>) FROM <tb_name>|<stb_name> [WHERE clause]
```

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。

LAST

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。

语法：

```
SELECT LAST(<field_name>) FROM <tb_name>|<stb_name> [WHERE clause]
```

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。

3）如果结果集中所有列全部为 NULL 值，则不返回结果。

TOP

功能说明：统计表/超级表中某列的值最大 *k* 个非 NULL 值。若多于 k 个列值并列最大，则返回时间戳小的。

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。

语法：

```
SELECT TOP(<field_name>, K) FROM <tb_name>|<stb_name> [WHERE clause]
```

说明：

1）*k* 值取值范围 1≤*k*≤20。

2）系统同时返回该记录关联的时间戳列。

BOTTOM

功能说明：统计表/超级表中某列的值最小 *k* 个非 NULL 值。若多于 k 个列值并列最小，则返回时间戳小的

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。

语法：

```
SELECT BOTTOM(<field_name>, K) FROM <tb_name>|<stb_name> [WHERE
clause]
```

说明：
1）*k* 值取值范围 1≤*k*≤20。
2）系统同时返回该记录关联的时间戳列。

PERCENTILE
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。

语法：
```
SELECT PERCENTILE(<field_name>, K) FROM <tb_name> [WHERE clause]
```

说明：*k* 值取值范围 0≤*k*≤100，为 0 的时候等同于 MIN，为 100 的时候等同于 MAX.

LAST_ROW
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。

语法：
```
SELECT LAST_ROW <field_name> FROM <tb_name>|<stb_name>
```

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。

5.7.3 计算函数

DIFF
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。

语法：
```
SELECT DIFF(<field_name>) FROM <tb_name> [WHERE clause]
```

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。

SPREAD
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。

语法：

```
SELECT SPREAD(<field_name>) FROM <tb_name>|<stb_name> [WHERE clause]
```

说明：

1） 可用于 TIMESTAMP 字段，此时表示记录的时间覆盖范围

四则运算

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。

语法：

```
SELECT <field_name>([+|-|*|/|%][Value|field_name])
FROM <tb_name>|<stb_name>  [WHERE clause]
```

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。

5.8 时间维度聚合

TDengine 支持按时间段进行聚合，可以将表中数据按照时间段进行切割后聚合生成结果，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。这个聚合适合于降采样 (down sample) 操作。

语法：

```
SELECT function1<field_name1>,function2<filed_name2> …
FROM <table name>|<stb_name>
WHERE <primary field_name> <[>=|=|<=|<>] val> (<[AND|OR]> …)
INTERVAL(<time range>) [FILL(NONE|VALUE|PREV|NULL)]
```

聚合时间段的长度由关键词 INTERVAL 指定，最短时间间隔 10 毫秒（10a）。聚合查询中，能够同时执行的聚合和选择函数仅限于**单个输出**的函数：count、avg、sum 、stddev、leastsquares、percentile、min、max、first、last，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）。

WHERE 语句可以指定查询的起止时间。

FILL 语句指定某一时间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：
不进行填充：NONE (默认填充模式)。
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)
NULL 填充：使用 NULL 填充数据。例如：fill(null)

PREV 填充：使用前一个非 NULL 值填充数据。例如：fill(prev)

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。

示例：温度数据表的建表语句如下：

```
create table sensor(ts timestamp, degree double, pm25 smallint)
```

针对传感器采集的数据，以 10 分钟为一个阶段，计算过去 24 小时的温度数据的平均值、最大值、温度的中位数、以及随着时间变化的温度走势拟合直线

```
SELECT AVG(degree),MAX(degree),LEASTSQUARES(degree),
PERCENTILE(degree, 50)
FROM sensor
WHERE TS>=NOW-1d
INTERVAL(10m)
```

6 超级表 STable：多表聚合

TDengine 要求每个数据采集点单独建表，这样能极大提高数据的插入/查询性能，但是导致系统中表的数量猛增，让应用对表的维护以及聚合、统计操作难度加大。为降低应用的开发难度，TDengine 引入了超级表 STable(Super Table)的概念。

6.1 STable 定义

STable 是同一类型数据采集点的抽象，是同类型采集实例（即子表）的集合，包含多张数据结构一样的子表。每个 STable 为其子表集合定义了表结构和一组标签：表结构即表中记录的数据列及其数据类型；标签名和数据类型由 STable 定义，标签值记录着每个子表的静态信息，用以对子表进行分组过滤。子表本质上就是第 7 章介绍的表，由一个时间戳主键和若干个数据列组成，每行记录着具体的数据，数据查询操作与普通表完全相同；但子表与普通表的区别在于每个子表都从属于一张超级表，并带有一组由 STable 定义的标签值。每种类型的采集设备可以定义一个 STable。数据模型定义表的每列数据的类型，如温度、压力、电压、电流、GPS 实时位置等，而标签信息属于 Meta Data，如采集设备的序列号、型号、位置等，是静态的，是表的元数据。用户在创建表（数据采集点）时指定 STable(采集类型)外，还可以指定标签的值，也可事后修改。

TDengine 扩展标准 SQL 语法用于定义 STable，使用关键词 tags 指定标签信息。语法如下：

```
CREATE TABLE <stable_name> (<field_name> TIMESTAMP, field_name1
field_type,…) TAGS(tag_name tag_type, …)
```

其中 tag_name 是标签名，tag_type 是标签的数据类型。标签可以使用时间戳之外的其他 TDengine 支持的数据类型，标签的个数最多为 6 个，名字不能与系统关键词相同，也不能与其他列名相同。如：

```
create table thermometer (ts timestamp, degree float)
tags (location binary(20), type int)
```

上述 SQL 创建了一个名为 thermometer 的 STable，带有标签 location 和标签 type。

为某个采集点创建表时，可以指定其所属的 STable 以及标签的值，语法如下：

```
CREATE TABLE <tb_name> USING <stb_name> TAGS (tag_value1,...)
```

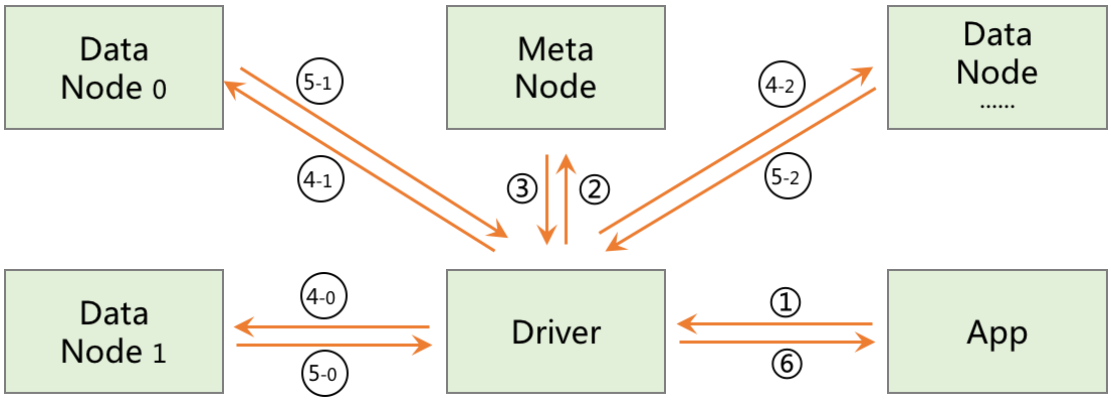
沿用上面温度计的例子，使用超级表 thermometer 建立单个温度计数据表的语句如下：

```
create table t1 using thermometer tags ('beijing', 10)
```

上述 SQL 以 thermometer 为模板，创建了名为 t1 的表，这张表的 Schema 就是 thermometer 的 Schema，但标签 location 值为'beijing'，标签 type 值为 10。

用户可以使用一个 STable 创建数量无上限的具有不同标签的表，从这个意义上理解，STable 就是若干具有相同数据模型，不同标签的表的集合，是一个超级表。与普通表一样，用户可以创建、删除、查看超级表 STable。

TDengine 还提供以 STable 为对象进行查询，大部分适用于普通表的查询操作都可运用到 STable 上，包括各种聚合和投影选择函数。除此之外，可以设置标签的过滤条件，仅对 STbale 中部分表进行聚合查询，大大简化应用的开发。其具体流程如下图所示：



1. 应用将一个查询条件发往系统；
2. Driver 将查询的过滤条件发往 Meta Node（虚拟管理节点 mnode)；
3. mnode 将符合查询过滤条件的子表的列表发回 Driver(包含每个子表对应的数据节点的 IP 地址)；
4. 这些返回的子表可能分布在多个数据节点，Driver 将计算的请求发往相应的多个数据节点；
5. 每个数据节点完成相应的聚合计算，将结果返回给 Driver；
6. Driver 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。

TDengine 对表的主键（时间戳）建立索引，暂时不提供针对数据模型中其他采集量（比如温度、压力值）的索引。每个数据采集点会采集若干数据记录，但每个采集点的标签仅仅是一条记录，因此数据标签在存储上没有冗余，且整体数据规模有限。TDengine 将标签数据与采集的动态数据完全分离存储，而且针对 STable 的标签建立了高性能内存索引结构，为标签提供全方位的快速操作支持。用户可按照需求对其进行增删改查（Create，Retrieve，Update，Delete，CRUD）操作。

STable 从属于库，一个 STable 只属于一个库，但一个库可以有一到多个 STable，一个 STable 可有多个子表。

6.2 STable 管理

- CREATE TABLE <stable_name> (<field_name> TIMESTAMP, field_name1 field_type,…) TAGS(tag_name tag_type, …)
  创建 STable，与创建表的 SQL 语法相似。但需指定 TAGS 字段的名称和类型。

  说明：
  1） TAGS 列总长度不能超过 512 bytes；
  2） TAGS 列的数据类型不能是 timestamp 和 nchar 类型；
  3） TAGS 列名不能与其他列名相同；
  4） TAGS 列名不能为预留关键字.

- SHOW STABLES
  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。

- DROP TABLE <stable_name>
  删除 STable。删除 STable 不会自动级联删除通过 STable 创建的表；而删除 STable 时要求通过该 STable 创建的表都已经被删除。

- SELECT TBNAME,[TAG_NAME,…] FROM <stable_name> WHERE <tag_name> <[=|=<|>=|<>] values..> ([AND|OR] …)
  查看属于某 STable 并满足查询条件的表。说明：TBNAME 为关键词，显示通过 STable 建立的子表表名，查询过程中可以使用针对标签的条件。

- SELECT COUNT(TBNAME) FROM <stable_name> WHERE <tag_name> <[=|=<|>=|<>] values..> ([AND|OR] …)
  统计属于某个 STable 并满足查询条件的子表的数量

6.3 写数据时自动建子表

在某些特殊场景中，用户在写数据时并不确定某个设备的表是否存在，此时可使用自动建表语法来实现写入数据时里用超级表定义的表结构自动创建不存在的子表，若该表已存在则不会建立新表。注意：自动建表语句只能自动建立子表而不能建立超级表，这就要求超级表已经被事先定义好。自动建表语法跟 insert/import 语法非常相似，唯一区别是语句中增加了超级表和标签信息。具体语法如下：

- INSERT INTO <tb_name> USING <stb_name> TAGS (<tag1_value>, ...) VALUES (field1_value1, ...) (field1_value2, ...) ...;
  向表 tb_name 中插入一条或多条记录，如果 tb_name 这张表不存在，则会用超级表 stb_name 定义的表结构以及用户指定的标签值(即 tag1_value…)来创建名为 tb_name 新表，并将用户指定的值写入表中。如果 tb_name 已经存在，则建表过程会被忽略，系统也不会检查 tb_name 的标签是否与用户指定的标签值一致，也即不会更新已存在表的标签。

- INSERT INTO <tb1_name> USING <stb1_name> TAGS (<tag1_value1>, ...) VALUES (<field1_value1>, ...) (<field1_value2>, ...) ... <tb_name2> USING <stb_name2> TAGS(<tag1_value2>, ...) VALUES (<field1_value1>, ...)(<field1_value2>,...) ...;
  向多张表 tb1_name，tb2_name 等插入一条或多条记录，并分别指定各自的超级表进行自动建表。

6.4 STable 中 TAG 管理

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，对于数值型的标签，新增加的标签的默认值是 0。

- ALTER TABLE <stable_name> ADD TAG <new_tag_name> <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 6 个。

- ALTER TABLE <stable_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。
  说明：第一列标签不能删除，至少需要为 STable 保留一个标签。

- ALTER TABLE <stable_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。

- ALTER TABLE <table_name> SET TAG <tag_name>=<new_tag_value>
  修改子表的标签值。

6.5 Stable 多表聚合

针对所有的通过 STable 创建的子表进行多表聚合查询，支持按照全部的 TAG 值进行条件过滤，并可将结果按照 TAGS 中的值进行聚合，暂不支持针对 binary 类型的模糊匹配过滤。语法如下：

```
SELECT function<field_name>,…
FROM <stable_name>
WHERE <tag_name> <[=|<=|>=|<>] values..> ([AND|OR] …)
INTERVAL (<time range>)
GROUP BY <tag_name>, <tag_name>…
ORDER BY <tag_name> <asc|desc>
SLIMIT <group_limit>
SOFFSET <group_offset>
LIMIT <record_limit>
OFFSET <record_offset>
```

说明：
超级表聚合查询，TDengine 目前支持以下聚合\选择函数：sum、count、avg、first、last、min、max、top、bottom，以及针对全部或部分列的投影操作，使用方式与单表查询的计算过程相同。暂不支持其他类型的聚合计算和四则运算。当前所有的函数及计算过程均不支持嵌套的方式进行执行。

不使用 GROUP BY 的查询将会对超级表下所有满足筛选条件的表按时间进行聚合，结果输出默认是按照时间戳单调递增输出，用户可以使用 ORDER BY _c0 ASC|DESC 选择查询结果时间戳的升降排序；使用 GROUP BY <tag_name> 的聚合查询会按照 tags 进行分组，并对每个组内的数据分别进行聚合，输出结果为各个组的聚合结果，组间的排序可以由 ORDER BY <tag_name> 语句指定，每个分组内部，时间序列是单调递增的。

使用 SLIMIT/SOFFSET 语句指定组间分页，即指定结果集中输出的最大组数以及对组起始的位置。使用 LIMIT/OFFSET 语句指定组内分页，即指定结果集中每个组内最多输出多少条记录以及记录起始的位置。

6.6 STable 使用示例

以温度传感器采集时序数据作为例，示范 STable 的使用。

在这个例子中，对每个温度计都会建立一张表，表名为温度计的 ID，温度计读数的时刻记为 ts，采集的值记为 degree。通过 tags 给每个采集器打上不同的标签，其中记录温度

计的地区和类型，以方便我们后面的查询。所有温度计的采集量都一样，因此我们用 STable 来定义表结构。

- 定义 STable 表结构并使用它创建子表
  创建 STable 语句如下：

```
CREATE TABLE thermometer (ts timestamp, degree double)
TAGS(location binary(20), type int)
```

  假设有北京，天津和上海三个地区的采集器共 4 个，温度采集器有 3 种类型，我们就可以对每个采集器建表如下：

```
CREATE TABLE therm1 USING thermometer TAGS (’beijing’, 1);
CREATE TABLE therm2 USING thermometer TAGS (’beijing’, 2);
CREATE TABLE therm3 USING thermometer TAGS (’tianjin’, 1);
CREATE TABLE therm4 USING thermometer TAGS (’shanghai’, 3);
```

  其中 therm1，therm2，therm3，therm4 是超级表 thermometer 四个具体的子表，也即普通的 Table。以 therm1 为例，它表示采集器 therm1 的数据，表结构完全由 thermometer 定义，标签 location=”beijing”, type=1 表示 therm1 的地区是北京，类型是第 1 类的温度计。

- 写入数据
  注意，写入数据时不能直接对 STable 操作，而是要对每张子表进行操作。我们分别向四张表 therm1，therm2， therm3， therm4 写入一条数据，写入语句如下：

```
INSERT INTO therm1 VALUES (’2018-01-01 00:00:00.000’, 20);
INSERT INTO therm2 VALUES (’2018-01-01 00:00:00.000’, 21);
INSERT INTO therm3 VALUES (’2018-01-01 00:00:00.000’, 24);
INSERT INTO therm4 VALUES (’2018-01-01 00:00:00.000’, 23);
```

- 按标签聚合查询
  查询位于北京(beijing)地区的型号为 1 的温度传感器采样值的数量 count(*)、平均温度 avg(degree)、最高温度 max(degree)、最低温度 min(degree)，并将结果按所处地域(location)和传感器类型(type)进行聚合。

```
SELECT COUNT(*), AVG(degree), MAX(degree), MIN(degree)
FROM thermometer
WHERE location=’beijing’ and type = 1
GROUP BY location
```

- 按时间周期聚合查询
  查询仅位于北京以外地区的温度传感器最近 24 小时(24h)采样值的数量 count(*)、平均温度 avg(degree)、最高温度 max(degree)和最低温度 min(degree)，将采集结果按照 10 分钟为周期进行聚合，并将结果按所处地域(location)和传感器类型(type)再次进行聚合。

```
SELECT COUNT(*), AVG(degree), MAX(degree), MIN(degree)
FROM thermometer
WHERE name<>'beijing' and ts>=now-1d
INTERVAL(10M)
GROUP BY location, type
```

7 TDengine 高级功能

7.1 流式计算

TDengine 可针对数据库中的表或 STable 提供高性能的连续流式计算(stream computing)功能。TDengine 中的流式计算采用时间驱动机制，原理上与其他流式计算引擎的滑动窗口相似，即通过指定滑动窗口的起始时间戳(forward sliding timestamp，每次计算的前向增量时间)和窗口范围(interval，聚合计算的时间范围)来进行连续的流式计算。通过指定向前滑动时间间隔(sliding)，将符合过滤条件的表(多个数据流)过去的一段时间内的数据进行流式计算，并可以将结果持久化写入新表，或者通过 API 调用直接使用。TDengine 暂不支持元组驱动的流式计算功能。

流式计算使用的 SQL 语法与普通的聚合计算完全一样，只是不能指定计算启动的时间，用户创建流式计算后，系统自动计算最近完整的时间周期并进行计算。通过 TDengine 提供的流式计算，应用可以方便地进行动态数据降维(down sample)，可以方便的基于原始数据，生成新的各种可后续查询的数据。

例如：将位于北京地区的温度传感器采样值的数量 count(*)、平均温度 avg(degree)、最高温度 max(degree)和最低温度 min(degree)每隔 2 分钟对过去的 10 分钟的数据进行聚合，写入新的表 taga，具体 SQL 如下：

```
CREATE TABLE TAGA AS
SELECT COUNT(*), AVG(degree), MAX(degree), MIN(degree)
FROM thermometer
WHERE name='beijing'
INTERVAL(10M) SLIDING(2M)
```

参数 SLIDING 用于指定滑动窗口每次滑动的时间间隔，而参数 INTERVAL 是指对多长时间的数据进行聚合计算。一般来说，sliding 的时间小于 interval 的时间长度。应用可以通过订阅 API 来实时获取新生成的数据。TDengine 提供一组流式计算 API，方便应用直接调用流式计算数据，而不用先保存进数据库，然后再查询，详见 API 章节。

注意事项：

1. sliding 参数是可选参数，如果不指定 sliding 值，系统自动设置为上一次聚合的结束时间为本次聚合计算的起始时间，即此时流式聚合计算退化为时间段连续聚合计算。

2. 聚合时间段(interval)不能小于 1 秒，聚合时间段没有上限。
3. 前向滑动窗口的时间(sliding)不能大于聚合时间段时间(interval)。
4. 如果是将流式计算的结果写入数据库中特定的表，删除该表将导致该流式计算过程自动终止。
5. 流计算可以通过在 WHERE 语句中添加对时间戳主键的限制来指定终止时间，例如 `create table strm as select count(*) from tb where ts >= now and ts < now + 1h interval(2m) sliding(1m);`这个流计算 strm 会统计从系统当前时间开始 1 小时内，每个 2 分钟区间内写入表 tb 的记录条数，统计频率为每 1 分钟计算一次。

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* 命令关闭对应的连续查询。在写回模式中，如果用户可以直接将写回的表删除，此时连续查询也会自动停止并关闭。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 TDengine 订阅服务

消息队列的 Pub/Sub 模式已经越来越流行。功能上，对于时序数据而言，TDengine 的 insert 与 pub 没有本质区别，都是插入一条新的记录。并且 TDengine 是严格按照数据的时序进行存储的，因此本质上，TDengine 里一张表的数据就是一个标准的消息队列，而且是一个格式化数据的消息队列。

为便于应用的开发，进一步降低集成和开发的成本，TDengine 提供订阅服务。与流行的 Kafka 相比，应用不是订阅一个“topic”,而是订阅数据库中的一张表或一条 SQL 语句的结果集，一旦这张表/结果集里有新的记录，应用将立即得到通知，从而获取该数据，整个流程与 Kafka 的数据订阅功能相似。

后续 API 的章节对如何使用将有详细介绍。

7.3 缓存（Caching）

缓存机制在前述章节已有描述。用户可通过函数 last 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：

```
select last_row(degree) from thermometer where location='beijing';
```

该 SQL 语句将获取所有位于北京的传感器最后记录的温度值。

8 连接器

TDengine 提供了丰富的应用程序开发接口，涉及语言包括 C/C++、JAVA、JavaScript 等，多种语言的开发接口可便于用户快速开发自己的应用。

C/C++开发接口类似于 MySQL，Java 开发接口则是 JDBC3.0 的子集。此外，TDengine 还面向 windows 平台提供 ODBC 接口。JavaScript 接口遵循 RESTful 接口使用习惯，用户可以快速上手进行开发。TDengine 后续还将提供大数据平台 Hadoop、Spark 接口。

8.1 C/C++ 连接器

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（安装后，位于/usr/local/include/taos，请在源程序中加上 #include <taos/taos.h>），连接时需要链接 TDengine 库 libtaos.so（安装后，位于 /usr/local/lib，gcc 编译时，请加上 -ltaos)。所有 API 返回-1 或 NULL 均表示失败。

8.1.1 C/C++同步 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：

1. TAOS *taos_connect(char *ip, char *user, char *pass, 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：TDengine 管理主节点的 IP 地址、用户名、密码、数据库名字和端口号。如果用户没有提供数据库名字，也可以正常连接，用户可以通过该链接，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。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。

2. void taos_close(TAOS *taos)
   关闭连接，其中 taos 是 taos_connect 的返回的指针。

3. int taos_query(TAOS *taos, char *sqlstr)
   该 API 用来执行 SQL 语句，可以是 DQL 语句也可以是 DML 语句，或者 DDL 语句。其中的 taos 参数是通过 taos_connect()的指针。返回-1 表示失败。

4. TAOS_RES *taos_use_result(TAOS *taos)
   选择相应的查询结果集。

5. TAOS_ROW taos_fetch_row(TAOS_RES *res)
   按行获取查询结果集中的数据。

6. int taos_num_fields(TAOS_RES *res)
   获取查询结果集中的列数。

7. TAOS_FIELD *taos_fetch_fields(TAOS_RES *res)
   获取查询结果集每列数据的属性（数据类型、名字、字节数），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。

8. 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。

9. void taos_init()
   初始化环境变量。如果没有调用该 API，taos_connect 将自动调用，因此一般情况下应用程序无需手动调用该 API。

10. char *taos_errstr(TAOS *taos)
    获取最近一次 API 调用失败的原因,返回值为字符串。

11. char *taos_errno(TAOS *taos)
    获取最近一次 API 调用失败的原因，返回值为错误代码。

12. int taos_options(TSDB_OPTION option, const void * arg, ...)
    设置客户端选项，目前只支持时区设置（TSDB_OPTION_TIMEZONE）和编码设置（TSDB_OPTION_LOCALE）。时区和编码默认为操作系统当前设置。

上述 12 个 API 是 C/C++接口中最重要的 API，剩余的辅助 API 请参看 taos.h 文件。上述 API 均采用同步调用的模式，在数据库操作执行完成之前，客户端应用将处于阻塞状态。单个数据库连接，在同一时刻只能有一个线程调用 API，否则可能会导致客户端应用崩溃。客户应用可以建立多个连接，进行多线程的数据写入或查询处理。

8.1.2 C/C++ 异步接口

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，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事物，从而可以提升整个应用的性能，在网络延迟严重时，优点尤为突出。

异步 API 都需要应用提供相应的回调函数，回调函数均包括如下参数：前面的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。下面是三个重要的异步 API：

1. void taos_query_a(TAOS *taos, char *sqlstr, void (*fp)(void *param, TAOS_RES *, int code), void *param);
   异步执行 SQL 查询或插入语句。taos 是调用 taos_connect 返回的数据库连接结构体，sqlstr 是需要执行的 SQL 语句，fp 是用户定义的回调函数，param 是

应用提供一个用于回调的参数。回调函数 fp 的第三个参数 code 用于指示操作是否成功，0 表示成功，-1 表示失败。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。

2. 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 是 taos_query_a 回调时返回的结果集结构体指针，fp 为回调函数。回调函数中的 param 是用户可定义的传递给回调函数的参数结构体。numOfRows 表明有 fetch 数据返回的行数(numOfRows 并不是本次查询满足查询条件的全部元组数量)。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)。

3. void taos_fetch_row_a(TAOS_RES *res, void (*fp)(void *param, TAOS_RES *, TAOS_ROW), void *param);
   异步获取一条记录。其中 res 是 taos_query_a 回调时返回的结果集结构体指针，fp 为回调函数，param 是应用提供的一个用于回调的参数。回调时，第三个参数 TAOS_ROW 指向一行记录。不同于 taos_fetch_rows_a，应用无需调用同步 API taos_fetch_row 来获取一个元组，更加简单。数据提取性能不及批量获取的 API。

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），不能执行第二个插入或查询操作。

8.1.3 C/C++ 流式计算接口

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：

1. TAOS_STREAM *taos_open_stream(TAOS *taos, char *sqlstr, void (*fp)(void *param, TAOS_RES *, TAOS_ROW row), int64_t stime, void *param)
   该 API 用来创建数据流，其中 taos 是调用 taos_connect 返回的结构体指针；sqlstr 是 SQL 查询语句(仅能使用查询语句)；fp 是用户定义的回调函数指针，每次流式计算完成后，均回调该函数，用户可在该函数内定义其内部业务逻辑；param 是应用提供的用于回调的一个参数，回调时，提供给应用；stime 是流式计算开始的时间，如果是 0，表示从现在开始，如果不为零，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）。返回值为 NULL，表示创建成功，返回值不为空，表示成功。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。

2. 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。

8.1.4 C 语言订阅 API

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

1. TAOS_SUB *taos_subscribe(char *host, char *user, char *pass, char *db, char *table, long time, int mseconds)
   该 API 用来启动订阅，需要提供的参数包含：TDengine 管理主节点的 IP 地址、用户名、密码、数据库、数据库表的名字；time 是开始订阅消息的时间，是从 1970 年 1 月 1 日起计算的毫秒数，为长整型，如果设为 0，表示从当前时间开始订阅；mseconds 为查询数据库更新的时间间隔，单位为毫秒，建议设为 1000 毫秒。返回值为一指向 TDengine_SUB 结构的指针，如果返回为空，表示失败。

2. TAOS_ROW taos_consume(TAOS_SUB *tsub)
   该 API 用来获取最新消息，应用程序一般会将其置于一个无限循环语句中。其中参数 tsub 是 taos_subscribe 的返回值。如果数据库有新的记录，该 API 将返回，返回参数是一行记录。如果没有新的记录，该 API 将阻塞。如果返回值为空，说明系统出错，需要检查系统是否还在正常运行。

3. void taos_unsubscribe(TAOS_SUB *tsub)
   该 API 用于取消订阅，参数 tsub 是 taos_subscribe 的返回值。应用程序退出时，需要调用该 API，否则有资源泄露。

4. int taos_num_subfields(TAOS_SUB *tsub)
   该 API 用来获取返回的一排数据中数据的列数

5. TAOS_FIELD *taos_fetch_subfields(TAOS_SUB *tsub)
   该 API 用来获取每列数据的属性（数据类型、名字、字节数），与 taos_num_subfileds 配合使用，可用来解析返回的一排数据。

8.2 JAVA Connector

8.2.1 JDBC 接口

如果用户使用 Java 开发企业级应用，可选用 TDengine 提供的 JDBC Driver 来调用服务。TDengine 提供的 JDBC Driver 是标准 JDBC 规范的子集，遵循 JDBC 标准(3.0)API 规范，支持现有的各种 Java 开发框架。用户开发时，需要把驱动包

JDBCDriver-x.x.x-dist.jar 添加开发环境的依赖仓库中。

JDBCDriver-x.x.x-dist.jar 驱动程序包的在不同系统上依赖不同的动态链接库。Linux 系统上，成功安装 TDengine 后，jar 包位于/usr/local/lib/taos 目录下，该目录也包含 jar 包在 Linux 上运行所依赖的动态链接库 libtaos.so 文件。Windows 系统上，成功安装客户端后，JDBCDriver-x.x.x-dist.jar 驱动程序包位于 install_directory\driver\JDBC 目录下，其依赖的动态链接库 taos.dll 文件位于 install_directory\driver\C 目录下。

TDengine 的 JDBC Driver 遵循标准 JDBC 规范，用户可以像使用 Oracle 或 MySQL 的 JDBC Driver 一样调用其中的方法（具体使用规范，请参考 Oracle 官方的 JDBC 相关文档）。

TDengine 的 JDBC URL 格式为：

```
jdbc:TSDB://{host_ip}:{port}/{database_name}?[user={user}|&password={password}|&charset={charset}|&cfgdir={config_dir}|&locale={locale}|&timezone={timezone}]
```

其中，{}中的内容必须，[]中为可选。配置参数说明如下：

- user：登陆 TDengine 所用用户名，默认值 root
- password：用户登陆密码，默认值 taosdata
- charset：客户端使用的字符集，默认值 UTF-8
- cfgdir：客户端配置文件目录路径，默认值/etc/taos
- 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

Tips：以上配置参数出了 cfgdir 外，均可在客户端配置文件 taos.cfg 中进行配置（详见 11.4 节）。配置文件的优先级低于 JDBC URL 的优先级，也即如果 charset 同时在配置文件 taos.cfg 中配置，也在 JDBC URL 中配置，则使用 JDBC URL 中的配置值。

此外，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的差异导致 TDengine 的 Java API 并不能与 JDBC 完全一致。 TDengine 不提供针对写入数据的删除和修改的操作、不支持表间的 join 或 union 操作，因此也缺乏对该部分 API 的支持。目前 TDengine 不支持嵌套查询（即 nested query），因此对每个 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，TSDBJDBCDriver 则会自动关闭上一个 ResultSet。对于其中存在的具体操作，请参考 code/examples/JDBC/TSDBSyncSample.java 中的使用示范。

对于 TDengine 操作的报错信息，用户可使用 JDBCDriver 包里提供的枚举类 TSDBError.java 来获取 error message 和 error code 的列表。

8.2.2 Java 订阅 API

TDengine 提供源码公开的 Java 订阅接口开发包 subscribe-1.0.1-sources.jar。目前的 Java 订阅 API 支持订阅一张表，超级表或者一条 SQL 查询的结果集。结构上主要

包括三个类： Topic, Consumer, ConsumerResultSet。

- Topic 类定义订阅的对象，包括两个子类 TableTopic 和 STableTopic，分别代表对 TDengine 中单表或超级表的订阅。初始化 TableTopic/STableTopic 时，用户可以传入一个 String，该 String 可以是一个表名或者一条完整的可执行的 SQL 语句，Topic 会自动解析传入字符串，并建立订阅对象。
- Consumer 类定义订阅对象的消费者，相当于一个订阅操作的客户端。Consumer 中提供了 subscribe(Topic topic)来指定订阅对象和 poll()方法来轮询最新结果。
- Consumer 的 poll()方法会返回一个结果集 ConsumerResultSet，ConsumerResultSet 是 JDBC 标准结果集 java.sql.ResultSet 的一个实现，具体使用可参考 JDBC 标准规范。

更多详细的 API 信息请参考源码。

8.3 Python Connector

8.3.1 客户端安装

TDengine 的 python 客户端可通过 pip 命令，在解压出的安装包(windows 或 Linux）路径下运行如下命令安装相应的客户端版本：

```
pip install driver/Python/python2/
```

或

```
pip install driver/Python/python3/
```

如果机器上没有 pip 命令，用户可将 driver/python3 或 driver/python2 下的 taos 文件夹拷贝到应用程序的目录使用。

8.3.2 Python 客户端接口

在使用 TDengine 的 python 接口时，需导入 TDengine 客户端模块：

```
import taos
```

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 code/examples/python 中的示例程序。以下为部分常用类和方法：

1. TDengineConnection 类
   参考 python 中 help(taos.TDengineConnection)。
2. TDengineCursor 类
   参考 python 中 help(taos.TDengineCursor)。
3. connector 方法
   用于生成 taos.TDengineConnection 的实例。

8.4 RESTful API 接口

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。

目前版本通过简单的由用户名和密码生成的<TOKEN>来做请求的身份识别，后续版本将提供标准安全的数字签名机制来做身份验证。

8.4.1 HTTP 请求格式

```
http://<ip>:<PORT>/rest/sql
```

参数说明：
IP：集群中的任一台主机
PORT：配置文件中 httpPort 配置项，缺省为 6020

如:http://192.168.0.1:6020/rest/sql 是指向 IP 地址为 192.168.0.1 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。自定义身份认证信息如下所示（<token>稍后介绍）

```
Authorization: Taosd <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 来发起一个 HTTP Request，语法如下：

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

8.4.2 返回格式

返回值为 JSON 格式，如下：

```
{
   "status": "succ",
   "head": ["column1","column2", …],
   "data": [
     ["2017-12-12 23:44:25.730", 1],
     ["2017-12-12 22:44:25.728", 4]
   ],
   "rows": 2
}
```

说明：

- 第一行"status"告知操作结果是成功还是失败;
- 第二行"head"是表的定义，如果不返回结果集，仅有一列"affected_rows";
- 第三行是具体返回的数据，一排一排的呈现。如果不返回结果集，仅[[affected_rows]]
- 第四行"rows"表明总共多少行数据

8.4.3 获取授权码<TOKEN>

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://<ip>:6020/rest/login/<username>/<password>
```

其中，ip 是 TDengine 数据库的 IP 地址，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://192.168.0.1:6020/rest/login/root/taosdata
```

返回值：

```
{
    "status": "succ",
    "code": 0,
    "desc":
        "/KfeAzX/f9na8qdtNZmtONryp201ma04bEl8LcvLUd7a8qdtNZmtON
        ryp201ma04"
}
```

8.4.4 使用示例

- 在 demo 库里查询表 t1 的所有记录，curl 如下：

```
curl -H 'Authorization: Taosd
/KfeAzX/f9na8qdtNZmtONryp201ma04bEl8LcvLUd7a8qdtNZmtONryp201ma
04' -d 'select * from demo.t1' 192.168.0.1:6020/rest/sql
```

返回值：

```
{
    "status": "succ",
    "head": ["column1","column2","column3"],
    "data": [
      ["2017-12-12 23:44:25.730", 1, 2.3],
      ["2017-12-12 22:44:25.728", 4, 5.6]
```

```
    ],
    "rows": 2
}
```

- 创建库 demo：

```
curl -H 'Authorization: Taosd
/KfeAzX/f9na8qdtNZmtONryp201ma04bEl8LcvLUd7a8qdtNZmtONryp201ma
04' -d 'create database demo' 192.168.0.1:6020/rest/sql
```

返回值：

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

8.5 Go Connector

TDengine 提供了 GO 驱动程序“taosSql”包。taosSql 驱动包是基于 GO 的“database/sql/driver”接口的实现。用户可在安装后的 /usr/local/taos/connector/go 目录获得 GO 的客户端驱动程序。用户需将驱动包 /usr/local/taos/connector/go/src/taosSql 目录拷贝到应用程序工程的 src 目录下。然后在应用程序中导入驱动包，就可以使用“database/sql”中定义的接口访问 TDengine：

```
import (
    "database/sql"
    _ "taosSql"
)
```

taosSql 驱动包内采用 cgo 模式，调用了 TDengine 的 C/C++同步接口，与 TDengine 进行交互，因此，在数据库操作执行完成之前，客户端应用将处于阻塞状态。单个数据库连接，在同一时刻只能有一个线程调用 API。客户应用可以建立多个连接，进行多线程的数据写入或查询处理。

9 与其他工具的连接

9.1 与 Telegraf 的连接

TDengine 能够与开源数据采集系统 Telegraf 快速集成，整个过程无需任何代码开发。

9.1.1 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
```

9.1.2 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：http://ip:6020/telegraf/udb，其中 ip 为 TDengine 集群的中任意一台服务器的 IP 地址，6020 为 TDengine RESTful 接口的端口号，telegraf 为固定关键字，udb 为用于存储采集数据的数据库名称，可预先创建。
- method: "POST"
- username: 登录 TDengine 的用户名
- password: 登录 TDengine 的密码
- 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: 30，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，但对于 TDegine，该数值不能超过 50

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。

9.2 与 Grafana 连接

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。

9.2.1 安装 Grafana

目前 TDengine 支持 Grafana 5.2.4 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
```

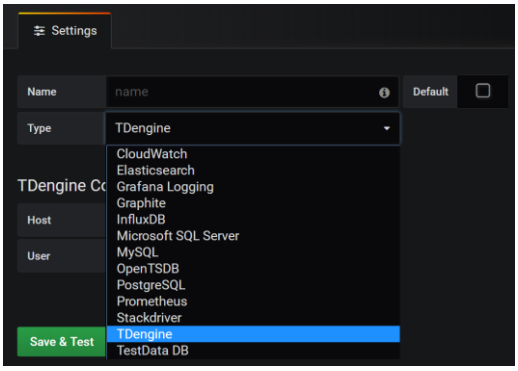
9.2.2 配置 Grafana

TDengine 的 Grafana 插件在安装包的 code/examples/grafana 目录下。

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

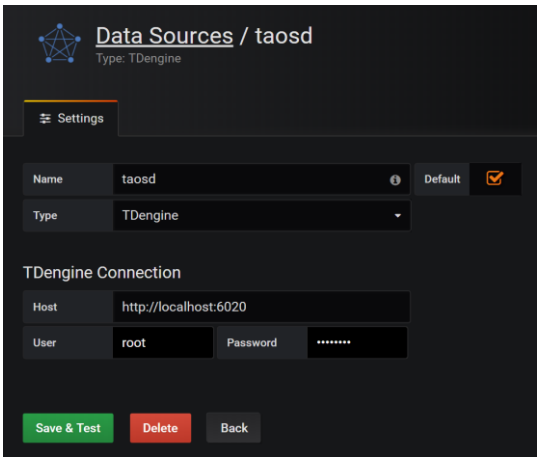
9.2.3 使用 Grafana

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)，配置 TDengine 数据源，如下图所示，此时可以在下拉列表中看到 TDengine 数据源。

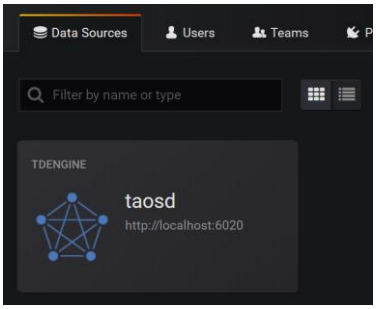


TDengine 数据源中的 HTTP 配置里面的 Host 地址要设置为 TDengine 集群的中任意一台服务器的 IP 地址与 TDengine RESTful 接口的端口号(6020)。假设 TDengine 数据库与 Grafana 部署在同一机器，那么应输入：http://localhost:6020。

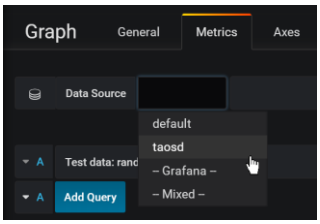
此外，还需配置登录 TDengine 的用户名与密码，然后点击下图中的 Save&Test 按钮保存。



然后，就可以在 Grafana 的数据源列表中看到刚创建好的 TDengine 的数据源：



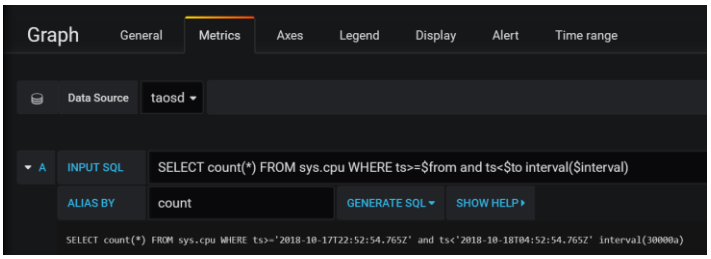
基于上面的步骤，就可以在创建 Dashboard 的时候使用 TDengine 数据源，如下图所示：



然后，可以点击 Add Query 按钮增加一个新查询。

在 INPUT SQL 输入框中输入查询 SQL 语句，该 SQL 语句的结果集应为两行多列的曲线数据，例如 SELECT count(*) FROM sys.cpu 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。

ALIAS BY 输入框为查询的别名，点击 GENERATE SQL 按钮可以获取发送给 TDengine 的 SQL 语句。如下图所示：



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。

9.3 与 MatLab 连接

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。

9.3.1 MatLab 的 JDBC 接口适配

MatLab 的适配有下面几个步骤，下面以 Windows10 上适配 MatLab2017a 为例：

1. 将 TDengine 安装包内的驱动程序 JDBCDriver-1.0.0-dist.jar 拷贝到 ${matlab_root}\MATLAB\R2017a\java\jar\toolbox
2. 将 TDengine 安装包内的 taos.lib 文件拷贝至 ${matlab_ root_dir}\MATLAB\R2017a\lib\win64
3. 将新添加的驱动 jar 包加入 MatLab 的 classpath。在${matlab_ root_dir}\MATLAB\R2017a\toolbox\local\classpath.txt 文件中添加下面一行

   $matlabroot/java/jar/toolbox/JDBCDriver-1.0.0-dist.jar
4. 在 ${user_home}\AppData\Roaming\MathWorks\MATLAB\R2017a\ 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

9.3.2 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('db', 'root', 'taosdata',
'com.taosdata.jdbc.TSDBDriver', 'jdbc:TSDB://127.0.0.1:0/')
```

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。

9.4 R Connector

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-1.0.0-dis
t.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&pass
word=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 Windows 客户端及程序接口

10.1 客户端安装

在 Windows 操作系统下，TDengine 提供 32 位和 64 位的 Windows 客户端，客户端安装应用程序为.exe 文件，运行该文件即可安装，默认安装路径为 C:\TDengine。Windows 的客户端可运行在主流的 32/64 位 Windows 平台之上，客户端目录结构如下：

```
+---cfg
+---connector
|   +---excel
|   \---grafana
|       \---tdengine
|           +---css
|           +---img
|           \---partials
+---driver
|   +---C
|   +---C#
|   +---JDBC
|   +---ODBC
|   \---Python
|       +---python2
|       |   +---build
|       |   |   \---lib
|       |   |       \---taos
|       |   +---dist
|       |   +---taos
|       |   \---taos.egg-info
|       \---python3
|           +---build
|           |   \---lib
|           |       \---taos
|           +---dist
|           +---taos
|           \---taos.egg-info
+---examples
|   +---C
|   +---C#
|   +---JDBC
|   +---Matlab
|   +---ODBC
|   +---Python
|   \---R
\---log
```

其中，最常用的文件列出如下：

1. CLI 可执行文件：install_directory/taos.exe
2. 配置文件：install_directory/cfg/taos.cfg
3. ODBC 驱动程序目录：install_directory/driver/odbc
4. JDBC 驱动程序目录：install_directory/driver/jdbc
5. Windows 开发包：install_directory/driver/C
6. .NET 接口文件：install_directory/driver/C#
7. Python 接口文件：install_directory/driver/python
8. 各接口的示例代码：install_directory/examples

在开始菜单中搜索 cmd 程序，通过命令行方式执行 taos.exe 即可打开 TDengine 的 CLI 程序。

10.2 C++接口

TDengine 在 Window 系统上提供的 API 与 Linux 系统是相同的， 应用程序使用时，需要包含 TDengine 头文件 taos.h，连接时需要链接 TDengine 库 taos.lib，运行时将

taos.dll 放到可执行文件目录下。具体 API 及其用法请参考 6.1、6.2 和 6.3 节。

10.3 ODBC 接口

应用程序可以使用 ODBC 接口来执行所有数据库的操作，提供 taosodbc.dll 动态库和 taosodbc.lib 静态库两个文件。通过 ODBC 访问数据库时，既可注册到系统 ODBC 管理器执行，也可以直接连接到开发项目中。

1. 注册到系统 ODBC 管理器的方式
   a. 检查 Windows 版本（Windows7 以上）；
   b. 将 odbc/odbc.reg 文件中的 dll 路径改为解压后的 odbc/taosodbc.dll 路径，然后运行 odbc.reg 文件将信息写入到注册表中；
   c. 运行 ODBC 管理器，新增数据源，必须添加 IP 地址、用户名、密码字段；
   d. 测试 ODBC，新增名为 demo 的数据源后，检查 odbc/odbcTest.exe 的运行情况。
2. 直接连接到开发项目的方式
   a. 在工程中添加对 odbc/taosodbc.dll 和 odbc/taosodbc.lib 的依赖；
   b. 取消对 odbc32.dll 和 odbccp32.lib 的依赖；
   c. 将 odbc/taosodbc.dll 放到可执行文件目录。
3. 由于时序空间数据库与传统的 RDBMS 是有区别的，因此仅提供了 ODBC Driver 规范中的一个子集，支持的函数列表包括：
   - SQLAllocHandle
   - SQLFreeHandle
   - SQLAllocStmt
   - SQLFreeStmt
   - SQLAllocEnv
   - SQLFreeEnv
   - SQLAllocConnect
   - SQLFreeConnect
   - SQLConnect
   - SQLDisconnect
   - SQLDriverConnect
   - SQLError
   - SQLPrepare
   - SQLExecute
   - SQLNumResultCols
   - SQLDescribeCol
   - SQLBindCol
   - SQLExecDirect
   - SQLFetch
   - SQLGetData

10.4 JDBC 接口

在 Windows 系统上，应用程序可以使用 JDBC 接口来操纵数据库。使用 JDBC 接口的步骤如下：

1. 将 JDBC 驱动程序(JDBCDriver-1.0.0-dist.jar)放置到当前的 CLASS_PATH 中;
2. 将 Windows 开发包(taos.dll)放置到 system32 目录下。

详细的 API 接口及用法请参考 8.2 节。

10.5 .NET 接口

在 Windows 系统上，.NET 应用程序可以使用 TDengine 的.NET 接口来执行所有数据库的操作。.NET 接口文件 TDengineDrivercs.cs 和参考程序示例 TDengineTest.cs 均位于 Windows 客户端 install_directory/C#目录下。使用.NET 接口的步骤如下所示：

1. 将.NET 接口文件 TDengineDrivercs.cs 加入到应用程序所在.NET 项目中。
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；
3. 此.NET 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/client 目录中的 taos.dll 文件到.NET 项目最后生成.exe 可执行文件所在文件夹。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. taos.dll 文件使用 x64 平台编译，所以.NET 项目在生成.exe 文件时，“解决方案”/“项目”的“平台”请均选择“x64”。
2. 此.NET 接口目前已经在 Visual Studio 2015/2017 中验证过，其它 VS 版本尚待验证。

11 TDengine 系统管理

TDengine 提供了较为完善的管理工具来管理系统，包括用户账号管理、库管理、表管理、节点管理、系统连接/任务管理、系统实时监测、数据导入导出等。这些管理既可以通过传统的命令行(Command Line Interface, CLI)进行，也可通过 Web 网页形式的可视化(Graphic User Interface, GUI)工具进行，管理员可根据使用习惯选择熟悉的管理数据库的工具。

11.1 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。

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。**注意：配置修改后，需要重启 taosd 服务才能生效。**

- mgmtShellPort：管理节点与客户端通信使用的 TCP/UDP 端口号（默认值是 6030）。此端口号在内向后连续的 5 个端口都会被 UDP 通信占用，即 UDP 占用 [6030-6034]，同时 TCP 通信也会使用端口[6030]。
- vnodeShellPort：数据节点与客户端通信使用的 TCP/UDP 端口号（默认值是 6035）。此端口号在内向后连续的 5 个端口都会被 UDP 通信占用，即 UDP 占用 [6035-6039]，同时 TCP 通信也会使用端口[6035]
- mgmtVnodePort：管理节点与数据节点通信使用的 TCP/UDP 端口号（默认值是 6040）。此端口号在内向后连续的 5 个端口都会被 UDP 通信占用，即 UDP 占用 [6040-6044]，同时 TCP 通信也会使用端口[6040]
- vnodeVnodePort：数据节点之间同步数据使用 TCP，端口号[6045]
- mgmtMgmtPort：管理节点之间通信使用 UDP，端口号[6050]
- mgmtSyncPort：管理节点之间同步数据使用 TCP，端口号[6050]
- httpPort：数据节点对外提供 RESTful 服务使用 TCP，端口号[6020]
- nginxPort：网页服务使用 TCP， 端口号[6060]
- publicIp：节点对外给应用提供服务的 IP 地址，对云平台，此为公网 IP 地址
- internalIp：对于阿里等云平台，publicIp 在内部映射对应的私有 IP 地址
- privateIp：物理节点用于集群节点内部通讯的 IP 地址
- masterIp：集群任意一物理节点的 privateIp
- secondIp：集群任意一物理节点的 privateIp
- dataDir：数据文件目录，缺省是/var/lib/taos
- logDir：日志文件目录，缺省是/var/log/taos
- offlineThreshold：判定服务器完全离线的时间阈值，单位为秒
- maxUsers：集群中用户的最大数量
- maxDbs：集群中数据库的最大数量
- maxTables：集群中数据表的最大数量
- maxDnodes：集群中物理节点的最大数量
- maxVGroups：集群中虚拟节点组的最大数量
- enableMicrosecond：时间戳为微秒的标志位（默认为毫秒）
- numOfLogLines：日志文件的最大行数
- enableMonitor：系统监测标志位，0：关闭，1：打开
- debugFlag：系统 debug 日志开关，131：仅错误和报警信息，135：所有

注意：对于 publicIp，internalIp，privateIp，如果不做任何配置，系统将自动取服务器的第一个可用的 IP 地址

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：

- days：一个数据文件覆盖的时间长度，单位为天
- keep：数据库中数据保留的天数
- rows：文件块中记录条数
- comp：文件压缩标志位，0：关闭，1:一阶段压缩，2:两阶段压缩
- ctime：数据从写入内存到写入硬盘的最长时间间隔，单位为秒
- clog：数据提交日志(WAL)的标志位，0 为关闭，1 为打开

- tables：每个 vnode 允许创建表的最大数目
- cache：内存块的大小（字节数）
- tblocks：每张表最大的内存块数
- ablocks：每张表平均的内存块数
- precision：时间戳为微秒的标志位，ms 表示毫秒，us 表示微秒

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 容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 replica 3 days 10 cache 16000 ablocks 4
```

该 SQL 创建了一个库 demo，它的副本数为 3，每个数据文件保留 10 天数据，内存块为 16000 字节，一张表平均有 4 个内存块，而其他参数与系统配置完全一致。

11.2 客户端配置

TDengine 系统的前台交互客户端应用程序为 taos，目前支持 Linux shell 和 Windows cmd。命令行运行客户端 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。更多 taos 的使用方法请见第 2.3 章。本节主要讲解 taos 客户端应用在配置文件 taos.cfg 文件中使用到的参数。

客户端配置参数列表及解释

- masterIP：客户端默认发起请求的 master 服务节点的 IP 地址
- secondIP：如果客户端连接 masterIP 失败，则尝试连接 secondIP
- charset：指明客户端所使用的字符集，默认值为 UTF-8。TDengine 存储 nchar 类型数据时使用的是 unicode 存储，因此客户端需要告知服务自己所使用的字符集，也即客户端所在系统的字符集。一般而言，中文 Windows 系统建议使用 GB18030；Linux 系统建议使用默认值 UTF-8
- locale：设置系统语言环境。Linux 上客户端与服务端共享
- defaultUser：默认登录用户，默认值 root
- defaultPass：默认登录密码，默认值 taosdata

TCP/UDP 端口，以及日志的配置参数，与 server 的配置参数完全一样。启动 taos 时，用户可以从命令行指定 IP 地址、端口号，用户名和密码，否则就从 taos.cfg 读取。

11.3 集群管理

在任意一台物理节点上，安装并启动 taosd 服务后，TDengine 将读取配置文件 taos.cfg，获取集群的 masterIp。如果没有配置集群的 masterIp，系统将 masterIp 设为服务器的本地 privateIp。如果 taosd 服务检测到自己的 privateIp 就是 masterIp，该节点将自动创建集群，并成为该集群的第一台服务器。如果启动节点的 privateIp 不是所配置的 masterIp，该节点将立即向 masterIp 的服务器发起请求加入集群。TDengine 集群根据用户已经设置的集群 IP 地址来决定是否接受加入集群的请求。如果没有在配置列表

中设置该节点的 IP，将拒绝来自该 IP 地址的节点加入请求。如果已经配置该 IP 地址，管理节点则接受其请求，并启动该节点的服务。节点的管理主要有如下几个操作：

- **增加物理节点：**
  使用 root 账号进入 TDengine CLI 控制台中，执行以下命令增加物理节点：
  ```
  create dnode ip_addr
  ```
  ip_addr 是被添加物理节点的私有 IP 地址。

- **删除物理节点：**
  使用 root 账号进入 TDengine CLI 控制台中，执行
  ```
  drop dnode ip_addr
  ```
  ip_addr 是被删除的节点 IP 地址。

- **查看节点：**
  使用 root 账号进入 TDengine CLI 控制台中，执行
  ```
  show dnodes
  ```
  集群中增加或删除的物理节点可以通过该命令进行查看。

集群中最先启动的 taosd 服务会初始化并建立集群， **请遵循如下步骤将后续服务器加入到现有集群：**

1. 后续新服务器安装 TDengine 软件包过程中，提示是否需要加入已经存在的集群时，输入第一台或现有集群中任意一台服务器的 IP 地址。
2. 使用 root 账号通过 shell 登录现有集群，将后续新服务器的 IP 地址使用上节中“增加物理节点”的方法手动添加到集群中。
3. 使用上节中”查看节点“的方法查看是否添加成功

对于步骤一，如果已经安装 TDengine，可以将 taos.cfg 里的 masterIp 配置为现有集群中的任意一台服务器 IP 地址。为避免 masterIp 机器宕机导致服务不可用，可在配置文件中指定 secondIp，secondIp 也是现有集群中任意一台服务器 IP 地址。修改后，需要重启 taosd 服务。

示例： 假设一计划的集群有三个节点，IP 地址分别为 192.168.0.1, 192.168.0.2 与 192.168.0.3。可以按照下面的步骤创建该集群：

1. 在 192.168.0.1 上安装 TDengine，不用做任何特殊配置，启动 TDengine;
2. 在 192.158.0.2 上安装 TDengine，当安装询问现有集群的 IP 地址时，输入 192.168.0.1，然后启动 TDengine。如果安装时忘记输入 masterIP,可以修改配置文件 taos.cfg，将参数 masterIp 设为 192.168.0.1，然后重新启动；
3. 在 Linux 命令行窗口执行“taos –h 192.168.0.1”，登录进 TDengine，然后执行命令：“create dnode 192.168.0.2;” 将 192.168.0.2 添加进集群。添加万后，可在 TDengine shell 里执行“show dnodes”检查是否添加成功；
4. 仿照第二和第三步，将 192.168.0.3 或更多的节点添加进集群。

警告：从集群中删除物理节点，需要首先执行 drop dnode 操作，在这个过程中系统自动将该物理节点的数据迁移到其他的节点。没有执行 drop dnode 操作的情况下，直接断开

该节点的网络连接或关闭服务器，(replica 为 1 的情况下)将有可能导致数据丢失。

11.4 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，并制定用户名和密码，密码需要用单引号引起来

- 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

- SHOW USERS
  显示所有用户

Note：TDengine 不提供针对单个用户的权限设置

11.5 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 ip:port 字串，如"192.168.0.1:42198"，拷贝粘贴即可。

- SHOW QUERIES
  显示数据查询，其中一列显示 ip:port:id，为发起该 query 应用的 IP 地址，端口号，以及系统分配的 ID。

- KILL QUERY <query-id>
  强制关闭数据查询，其中 query-id 是 SHOW QUERIES 中显示的 ip:port:id 字串，如"192.168.0.1:42198:11"，拷贝粘贴即可。

- SHOW STREAMS
  显示流式计算，其中一列显示 ip:port:id，为启动该 stream 的 IP 地址、端口和系统分配的 ID。

- 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 ip:port:id 字串，如"192.168.0.1:42198:18"，拷贝粘贴即可。

11.6 运行状态查询

TDengine 集群创建后，会自动创建一个监测数据库 SYS，并自动将集群内各节点的 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 SYS 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。

11.7 数据导入

TDengine 提供两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。

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。每个 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。其语法如下

- import/insert into tb1 file a.csv b.csv tb2 c.csv …

11.8 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 shell 中运行

- select * from tb >> a.csv

这样，表 tb 中的数据就会按照 CSV 格式导出到文件 a.csv 中。

用 taosdump 导出数据

TDengine 提供了方便的数据库导出工具 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。其用法如下：

- 导出数据库中的一张或多张表：taosdump [OPTION…] dbname tbname …
- 导出一个或多个数据库： taosdump [OPTION…] --databases dbname…
- 导出所有数据库（不含监控数据库）：taosdump [OPTION…] --all-databases

用户可通过运行 taosdump --help 获得更详细的用法说明

11.9 Web 管理工具

在浏览器输入 http://privateIp:6060，即可进入管理平台登录入口，录入用户名和密码，验证通过后即可开始对系统的维护和管理。其中 privateIP 是集群任意一个节点用于集群内部通讯的 IP 地址。

Web 页面除提供运行维护和管理功能外，还提供 SQL 命令行输入的功能，用户可以直接输入 SQL 命令操纵数据库，其使用方式与 CLI 管理工具完全相同。

使用 WEB 管理工具，因为 TDengine 安装包里已经自带 Ngix,只需要有浏览器，无需安装任何其他软件。

11.10 TDengine 文件和目录结构

TDengine 安装成功后，在系统中建立如下目录与文件：

- 执行文件目录：/usr/local/bin/taos
  该目录包含 TDengine 的可执行文件及网页配置文件，分别是：
  taosd：TDengine 后台服务，开机自动启动
  taos：命令行程序
  taosdump：数据导出程序
  remove.sh：TDengine 系统的卸载脚本，将删除 TDengine 所有程序和数据
  web：网页配置文件夹

- 配置文件目录：/etc/taos
  该目录包含 TDengine 的配置文件 taos.cfg；

- 数据文件目录：/var/lib/taos
  该目录包含 TDengine 存储的数据文件，所有的数据都存储在这些文件中。**请不要尝试删除或移动该目录下的文件，否则将导致数据丢失的灾难性后果。**

- 日志文件目录：/var/log/taos
  该目录包含 TDengine 产生的日志信息。日志文件大小、输出内容均可通过配置文件进行控制。日志文件是解决系统运行过程中出现问题的重要依据，系统管理员可以通过阅读分析日志，定位并排除系统故障。

用户可以通过修改配置文件 taos.cfg 中相应配置项 dataDir 和 logDir,调整文件目录、数据文件目录、日志文件目录的位置。

11.11 TDengine 的启动、停止、卸载

TDengine 使用 Linux 系统的 systemd 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 手动操作停止、启动、重新启动服务，命令如下：

- 启动服务进程：sudo systemctl start taosd
- 停止服务进程：sudo systemctl stop taosd
- 重启服务进程：sudo systemctl restart taosd

查看服务状态：sudo systemctl status taosd
如果服务进程处于活动状态，则系统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，则系统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 程序及所有存储的数据将被完全删除，务必谨慎使用。

11.12 TDengine 参数限制与保留关键字

- 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 32 个字符
- 表的列名：不能包含特殊字符，不能超过 32 个字符
- 表的列数：不能超过 250 列
- 记录的最大长度：包括时间戳 8 byte，不能超过 2KB
- 单条 SQL 语句最大字符串长度：64KB
- 数据库副本数：不能超过 3
- 用户名：不能超过 20 个 byte
- 用户密码：不能超过 15 个 byte
- 标签(Tags)数量：不能超过 6 个
- 标签的总长度：不能超过 512 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
- 库的格式：仅受节点个数限制
- 单个物理节点上虚拟节点个数：不能超过 64 个

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：

ABLOCKS	CONNECTION	GT	MINUS	SHOW
ABORT	CONNECTIONS	ID	MNODES	SLASH
ACCOUNT	COPY	IF	MODULES	SLIDING

ACCOUNTS	COUNT	IGNORE	NCHAR	SMALLINT
ADD	CREATE	IMMEDIATE	NE	SPREAD
AFTER	CTIME	IMPORT	NONE	STAR
ALL	DATABASE	IN	NOT	STATEMENT
ALTER	DATABASES	INITIALLY	NOTNULL	STDDEV
AND	DAYS	INSERT	NOW	STREAM
AS	DEFERRED	INSTEAD	OF	STREAMS
ASC	DELIMITERS	INTEGER	OFFSET	STRING
ATTACH	DESC	INTERVAL	OR	SUM
AVG	DESCRIBE	INTO	ORDER	TABLE
BEFORE	DETACH	IP	PASS	TABLES
BEGIN	DIFF	IS	PERCENTILE	TAG
BETWEEN	DIVIDE	ISNULL	PLUS	TAGS
BIGINT	DNODE	JOIN	PRAGMA	TBLOCKS
BINARY	DNODES	KEEP	PREV	TBNAME
BITAND	DOT	KEY	PRIVILEGE	TIMES
BITNOT	DOUBLE	KILL	QUERIES	TIMESTAMP
BITOR	DROP	LAST	QUERY	TINYINT
BOOL	EACH	LE	RAISE	TOP
BOTTOM	END	LEASTSQUARES	REM	TRIGGER
BY	EQ	LIKE	REPLACE	UMINUS
CACHE	EXISTS	LIMIT	REPLICA	UPLUS
CASCADE	EXPLAIN	LINEAR	RESET	USE
CHANGE	FAIL	LOCAL	RESTRICT	USER
CLOG	FILL	LP	ROW	USERS
CLUSTER	FIRST	LSHIFT	ROWS	USING
COLON	FLOAT	LT	RP	VALUES
COLUMN	FOR	MATCH	RSHIFT	VARIABLE
COMMA	FROM	MAX	SCORES	VGROUPS
COMP	GE	METRIC	SELECT	VIEW
CONCAT	GLOB	METRICS	SEMI	WAVG
CONFIGS	GRANTS	MIN	SET	WHERE
CONFLICT	GROUP			

12 常见问题及反馈

12.1 常见问题列表

1) 遇到错误"failed to connect to server"，我怎么办？

客户端遇到链接故障，请按照下面的步骤进行检查：

1. 在服务器，执行 systemctl status taosd 检查 taosd 运行状态。如果没有运行，启动 taos
2. 确认客户端连接时指定了正确的服务器 IP 地址
3. ping 服务器 IP，如果没有反应，请检查你的网络
4. 检查防火墙设置，确认 TCP/UDP 端口 6030-6039 是打开的
5. 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/lib/taos 里，并且/usr/local/lib/taos 在系统库函数搜索路径 LD_LIBRARY_PATH 里
6. 对于 windows 上的 JDBC, ODBC, Python, Go 等连接，确保 driver/c/taos.dll 在你的系统搜索目录里 (建议 taos.dll 放在目录 C:\Windows\System32)
7. 如果仍不能排除连接故障，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 端口链接是否工作：nc {hostIP} {port}

2) **TDengine 支持 validation query 吗？**

目前不支持 validation query，但是 TDengine 自带的监控库，'sys'，可以在某些情况下用做 validation query 的对象。您可以使用'describe sys'来作为 validation query，注意这个 query 会返回一个非空结果集。

3) TDengine 中可以删除或修改单条数据记录吗？

TDengine 的设计考虑了时序数据的特性，不对单条数据开放删除或修改操作。但是对于整张表或者其中某个列（时间戳主键不能被删除），或者整个数据库是可以删除的。此外，TDengine 提供了用户可配置的建库参数"keep"，专门用来控制数据文件的最长存储时间。keep 的默认值是 3650 天，即 10 年，配置后数据库中超过最长存储时间的数据会被系统自动删除来释放空间。

4) 如何删除超级表（STable）？与删除一张表（table）有什么区别？

删除 STable 的 SQL 语法与删除一个 table 没有任何区别。但删除 STable 的前提是，在这个 STable 下没有任何普通 table 了。目前 TDengine 还不支持，直接删除 STable 并连带删除其拥有的表。

5) 如何建立 1000 列的表

对于一张表，TDengine 目前最多支持 250 列。超过 250 列，建议分表存储。为保证查询效率，一张表不建议超过 64 列。

6) 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 shell 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);
```

7) 执行长的 sql 语句，没有语法错误，为什么系统返回了"invalid SQL"的错误？

对于 TDengine，单条 sql 语句的大小不能超过 64KB，超过这个长度，就会返回该错误。

8) 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记录。

12.2 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：

- /var/log/taos
- /etc/taos

附上必要的问题描述，以及发生该问题的执行操作，出现问题的表征及大概的时间。

请将上述信息邮件到 support@taosdata.com

反馈问题的时候，如果能够将系统调试状态下产生的运行日志反馈给技术团队，将有助于问题更快地解决。如果需要获取系统调试状态下运行日志，请在控制台程序中执行以下命令：

- **alter dnode** ip_addr debugflag 135

请将 ip_addr 替换为节点实际的 IP 地址，重新执行应用直到问题发生，然后将日志目录内全部内容打包并邮件给技术团队。最后，务必将 debug 的恢复为 131，否则系统将产生大量不必要的日志，影响运行性能。