TDSQL MySQL版

常见问题

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

常见问题
InnoDB 引擎相关
TDStore 引擎相关

常见问题
InnoDB 引擎相关

最近更新时间：2024-12-17 10:48:42

如何选择实例规格?

- 使用 TDSQL MySQL 版做功能性测试，且对性能没有特别要求：2个分片，每个分片配置为：内存/磁盘：2GB/25GB。
- 业务发展初期，总数据规模较小但增长快的选型：2个分片，每个分片配置为：内存/磁盘：16GB/200GB。
- 业务发展稳定，根据业务实际情况选型：4个分片，每个分片配置等于：当前业务峰值 * 增长率 / 4。
  更多关于实例规格，请参见 TDSQL MySQL 版实例及分片配置 。

TDSQL MySQL 版与传统 MySQL 语法之间的区别有哪些?

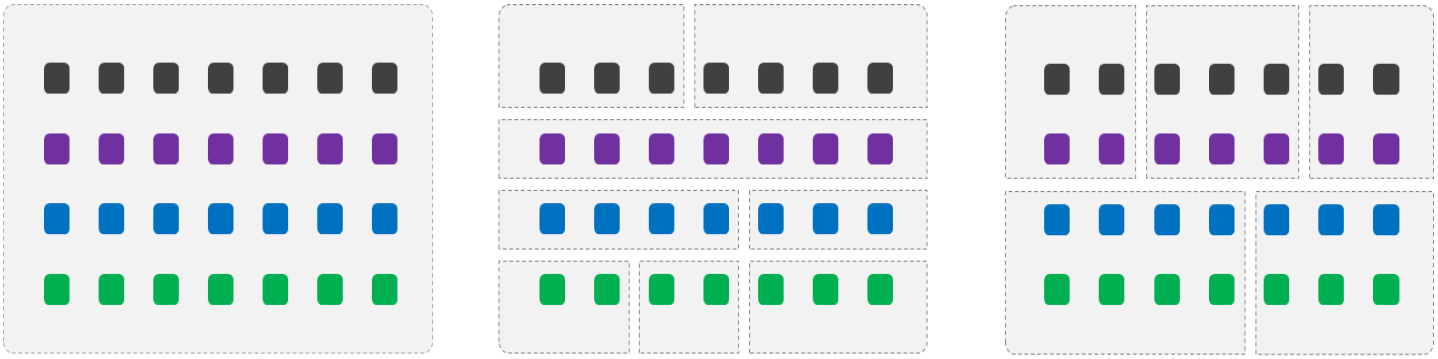
TDSQL MySQL 版目前版本不能通过命令行进行用户权限相关的设置，需要登录 控制台 进行操作。
TDSQL MySQL 版目前版本暂不支持自定义函数、视图、触发器、外键等特性。
对 MySQL 的语法兼容详情，请参见 使用限制 。

分表键有何作用?

- 使用分表，在执行操作 select 时，建议带上 shardkey 字段，proxy 根据该字段的 hash 值直接将 SQL 请求路由至对应的数据库实例进行处理；否则就需要发送给集群中所有的数据库实例执行，然后 proxy 根据数据库返回的结果集进行聚合，影响执行效率。
- 使用分表，在执行操作 insert/replace 时，字段必须包含 shardkey，否则会拒绝执行该 SQL，因为 proxy 不知道将该 SQL 发往哪个后端数据库。
- 使用分表，在执行操作 delete/update 时，为安全考虑，执行该类 SQL 时，必须带有 where 条件，否则拒绝执行该 SQL 命令。

如何选择分表键?

分表键是在水平拆分过程中用于生成拆分规则的数据表字段，必须在建表时指定好。TDSQL MySQL版 建议分表键尽可能找到数据表中的数据在业务逻辑上的主体，并确定大部分（或核心的）数据库操作都围绕这个主体的数据进行，方可使用该主体对应的字段作为拆分键，进行分表（该分表方案名为 Group-Shard）。如下图所示：



按组分表方案可以确保不同分表的某些关联数据和复杂的业务逻辑运算，可以聚合到一个物理分片内。例如，某电商平台订单表和用户表都是基于用户维度（UserID）拆分，平台就很容易通过联合查询（不会存在跨节点 join，或分布式事务）快速计算某个用户近期产生了多少订单。

一些典型选择拆分键的应用场景如下：

- 面向用户的互联网应用，是围绕用户维度来做各种操作，那么业务逻辑主体就是用户，可使用用户对应的字段作为拆分键。
- 电商应用或 O2O 应用，是围绕卖家/买家维度来进行各种操作，那么业务逻辑主体就是卖家/买家，可使用卖家/买家对应的字段作为拆分键。但请注意，某些情况下几个超大卖家占到绝大多数交易额，会导致某几个分片的负载和压力明显高于其他分片。
- 游戏类的应用，是围绕玩家维度来做各种操作，那么业务逻辑主体就是玩家，可使用玩家对应的字段作为拆分键。
- 物联网方面的应用，则是基于物联信息进行操作，那么业务逻辑主体就是传感器/SIM 卡，可使用传感器、独立设备、SIM 卡的 IMEI 作为对应的字段作为拆分键。
- 税务/工商类/社保的应用，主要是基于纳税人/法定代表人/居民的信息来开展前台业务，那么业务逻辑主体就是纳税人/法定代表人，可使用纳税人/法定代表人对应的字段作为拆分键。
  以此类推，其它类型的应用场景，大多也能找到合适的业务逻辑主体作为拆分键的选择。但需要注意在选择分表键时有一定限制，详情参见 分表键选择限制 。

分表键能否更换?

一旦选择好分区字段（shardkey），就不能轻易更改。若需要修改一个表的分区字段，只能新建一个表。
若需要修改一个分表某一行中的 shardkey 值，需要先 insert 再 delete。直接操作 update 不能修改分区字段的值。

如何连接实例?

您可通过内网地址和外网地址连接 TDSQL MySQL 版，详细操作步骤可参见 连接实例 。

可以限制用户登录有效期吗?

创建的用户不支持限制登录有效期。

Navicat 无法登录?

TDSQL MySQL 版数据库默认使用内网连接，如果您需要使用 Navicat 进行外网连接，请参考 开启外网地址 开启数据库的外网地址连接。

如何设置只从从机读取数据?

您可在 TDSQL MySQL 版控制台 创建账号时，账号类型选择“只从备机读取”。

如何创建备份?

您可通过 TDSQL MySQL 版控制台自定义备份，详情请参见 备份方式 。

如何设置实例参数?

您可登录 TDSQL MySQL 版控制台 ，单击实例 ID 进入实例管理页面，在**参数配置**页修改参数。

磁盘空间组成有哪些?

磁盘由以下2部分空间组成：

- 数据空间：您的数据所占用的空间。
- 系统文件空间：系统表空间文件、redolog 和 undolog、临时文件等。

> **说明：**
> binlog 文件空间为腾讯云免费赠送，不会占用您购买的磁盘空间容量。

支持灾备/只读实例吗?

支持，详情可参见 灾备/只读实例 。

是否支持缩减分片数量?

暂不支持缩减分片数量。

高版本能否降级为低版本?

暂不支持高版本无法降级为低版本。

性能检测主要功能有哪些?

资源分析、系统状态、表空间分布、冗余索引检测、死锁诊断、锁等待诊断、长会话诊断、慢查询分析、DB 状态检查等。详情可参见 性能检测 。

是否支持重启实例?

支持，您可登录 TDSQL MySQL 版控制台，在实例列表重启实例。

如何设置读写分离?

TDSQL MySQL 版支持下列几种模式的读写分离：

- 由 只读账号 发送的请求会根据配置的属性发给备机。
- 通过增加 slave 注释标记，将指定的 SQL 发往备机，即在 SQL 中添加 `/*slave*/` 这样的标记，该 SQL 会发送给备机。

> **说明：**
> 支持 `/*slave:slaveonly*/` 、 `/*slave:20*/` 、 `/*slave:slaveonly,20*/` 这几种形式，数值表示 slave 应该满足的延迟，slaveonly 表示在没有符合条件的 slave 时，不会将查询发送给主节点。

如何导入导出数据?

TDSQL MySQL 版支持通过 DTS 迁移数据 和通过 工具导入导出数据 。

三权分立是什么意思?

三权分立指将传统数据库系统 DBA 的角色分解为安全管理员、审计管理员、数据管理员三个相互独立的角色，三个角色之间相互制约，消除系统中的超级权限，从系统角色设计上解决了数据安全问题。

支持哪些格式的两级分区?

TDSQL MySQL 版支持 range 和 list 格式的两级分区，具体建表语法和 MySQL 分区语法类似。详情可参见 两级分区 。

支持的数据类型有哪些?

TDSQL MySQL 版支持 MySQL 的所有数据类型，包括数字，字符，日期，空间类型，JSON。详情请参见 兼容性 。

最大建表数是多少?

TDSQL MySQL 版默认支持最大建表数量为5000。

如何添加过滤条件查询数据?

使用 -c 模式连接数据库。

```
jyc@10.0.3.17 : pen 04:25:49> /*sets:allsets*/select count(1) from employees_int;
+----------+------------------+
| count(1) | info             |
+----------+------------------+
|        0 | set_1605865782_5 |
|        0 | set_1605865530_3 |
|        0 | set_1606708730_7 |
|        0 | set_1610700868_11 |
|        0 | set_1605865449_1 |
+----------+------------------+
5 rows in set (0.01 sec)

jyc@10.0.3.17 : pen 04:26:03>
jyc@10.0.3.17 : pen 04:26:03> /*sets:set_1605865449_1*/ select * from employees_int;
Empty set (0.01 sec)

jyc@10.0.3.17 : pen 04:26:18>
```

执行第一个命令，可以看到各个 set。

```
/*sets:allsets*/select count(1) from employees_int;
```

执行第二个命令，看到某个 set 上的数据。

```
/*sets:set_1605865449_1*/ select * from employees_int;
```

支持广播表和单表的 join 操作吗?

TDSQL MySQL 版广播表和单表的 join 操作如下：

- 支持单表（普通表）与单表（普通表）join，相当于单机 join，性能无损失。
- 支持广播表与分表 join，相当于单机 join，性能无损失。
- 支持广播表与广播表 join，相当于单机 join，性能无损失。
- 支持单表（普通表）与分表进行 join 操作。
  详情可参见 JOIN 和子查询 。

数据是均匀分布的吗?

TDSQL MySQL 版的数据是均匀分布的，通过 HASH 算法保证数据相对均匀地分散在不同的物理设备中。
您也可以将同年度的数据放到一个分片中，需要您不指定 shardkey，创建的表属于单表，默认落在第一个分片上。单表的创建方式您可以参考 建单表 进行操作。
详情可参见 两级分区 。

为什么 select last_insert_id() 无法获取自增 ID?

TDSQL MySQL 版目前 select last_insert_id() 只能跟 shard 表和广播表的自增字段一起使用，不支持 noshard 表。详情可参见 全局唯一字段 进行查看。

GTID 能否关闭?

TDSQL MySQL 版 GTID 不支持关闭。
MySQL 是基于 GTID 做的主从复制 enforce_gtid_consistency 用于保证主从数据一致，关闭将会导致主从同步失败，不支持设置为 OFF，GTID 是为了保证您的事务正常运行，所以不支持关闭。

支持窗口函数吗?

暂不支持窗口函数。

主键长度限制是多少?

主键默认长度不能超过767字节，同时也不建议主键设置过长。

不同事务隔离级别对性能有影响吗?

不同隔离级别，执行效率不同，隔离级别最高的是 Serializable 级别，最低的是 Read uncommitted 级别，级别越高，执行效率就越低。

版权所有：腾讯云计算（北京）有限责任公司

TDStore 引擎相关

最近更新时间：2025-04-22 15:40:02

TDStore 兼容什么数据库协议?

TDStore 兼容 MySQL8.0 协议，用户可以将其视为一个 MySQL 8.0 实例来使用，但有个别受限的操作，具体请参考 使用说明 。

TDStore 是否需要分片键（ShardKey）?

TDStore 无需定义分片键，其建表语法与原生 MySQL 保持一致。TDStore 分片机制基于 MySQL 原生的分区表，大多数情况下一级 Hash 分区表足以覆盖需求，Hash 分区打散在所有的数据节点上，均衡写入压力。

TDStore 的内部架构原理是怎样的?

请参考 TDStore 引擎介绍 。

TDStore 引擎与原生 MySQL 在读写性能上有何差异?

TDStore 是一款原生分布式数据库。它通过 Raft 协议在数据单元的副本之间保持一致性，默认情况下每个数据单元有三个副本，并且这些数据均匀分布在所有数据节点上。这种数据分布策略极大地提升了处理大量写入操作的效率，尤其在写多读少的业务场景中表现出色。
与 MySQL 的 InnoDB 存储引擎相比，TDStore 能够实现3~9倍的数据压缩率，这不仅有效减少了存储空间的需求，还可能提高 I/O 性能。因此，TDStore 特别适合那些对写入性能有较高要求的场景。

TDStore 是否提供了读写分离能力?

读写分离通常用于解决以下两个问题：

1. 在传统的主从架构中，通过读写分离可以充分利用备库的资源。
2. 针对具有明显 AP（可用性优先）和 TP（一致性优先）业务场景的应用，读写分离可以避免两者之间的相互影响。

然而，TDStore 的架构与传统的主从架构（如 InnoDB）有所不同。在 TDStore 中，数据均匀分布在所有节点上，因此每个节点的 CPU、IO 等资源都可以得到充分利用，无需依赖读写分离来利用备库资源。
对于一些需要将 AP 读取 SQL 与 TP 场景进行隔离的情况，我们计划在后续版本中提供相应的支持。

TDStore 是否支持只读账号和只读节点?

TDStore 暂不支持只读账号和只读节点。在 TDStore 对等节点结构中，读写请求可以分散到不同的节点上，从而充分利用各节点的资源，TDStore 产品架构请参见 实例架构 。
我们将在后续版本中添加对只读节点的支持，敬请期待。

TDStore 支持几种表类型? 是否有单表、广播表、分区表?

在 TDStore 中，目前主要支持普通表，包含带分区和不带分区两种。

- 对于带分区的表，不同分区的数据可以分散到不同节点上；
- 对于不带分区的表，如果数据量较大，复制组（Replication Group，RG）也会分裂，并均衡分散到各个节点上。

我们将在后续版本中添加对广播表的支持，敬请期待。

在 TDStore 中，是否需要使用分区表?

在单机版中，分区表主要用于通过分区裁剪提升 SQL 性能和通过 `drop partition` 的方式定期清理数据。而在 TDStore 分布式场景下，使用分区表的好处还包括利用多个节点的写入能力，这对于大数据量的处理尤为重要。
当面临大规模数据迁移时，建议预先将大表改造成基于 Hash 的分区表。这样做可以利用 TDStore 多节点的能力来加速数据导入过程。
如果没有预先分区，而是创建了一个单表，那么在数据导入初期，所有的写入操作都会集中在一个数据节点上，这可能导致 I/O 瓶颈。TDStore 提供了自动分裂和数据迁移的功能，但如果表一开始没有分区，这个过程可能会比较缓慢，并且在分裂和迁移期间，副本均衡也会带来额外的 I/O 开销。
通过创建分区表，可以最大限度地发挥 TDStore 分布式数据库的能力。这种改造的成本很低，只需要修改建表语句，而不需要对业务代码进行任何其他适配。例如，如果您的 TDStore 实例包含30个节点，创建一个包含30个分区的一级 Hash 分区表，TDStore 会在每个节点上创建一个主副本，从而实现副本均衡。同时，业务的增量数据会均匀分布在所有节点上，每个节点的压力也会相对均衡。
总之，为了充分利用 TDStore 的分布式特性并避免潜在的性能瓶颈，创建分区表是一种推荐的最佳实践。

TDStore 在读取场景中是否存在性能问题? 如何确定数据分片的位置?

在 TDStore 分布式数据库中，读取性能可能会受到数据分片和查询方式的影响。以下是两种常见的情况：

- 带分区键的查询：如果查询中包含了分区键，TDStore 能够直接将查询路由到包含该分区键的具体数据分片上。这种方式非常高效，因为它避免了不必要的数据遍历，直接定位到了正确的数据节点。
- 不带分区键的查询：当查询没有指定分区键时，TDStore 需要通过二级索引来确定数据的位置。这种情况下，系统会在包含该表数据的节点上进行遍历查询，这可能会导致性能上的轻微下降，因为需要检查更多的数据。

TDStore 最大支持容量是多少?

TDStore 的最大支持容量理论上是无限的。随着业务需求的增长，您可以通过增加更多的节点来扩展数据库的容量，以适应不断增长的数据存储和处理需求。目前，公有云上已经支持部署包含数十个节点的 TDStore 实例。
同时，TDStore 还提供了可视化界面，用于便捷地进行水平扩容和缩容操作。同时 TDStore 内置了自动搬迁和容量均衡功能，能够在节点间自动调整数据分布，确保系统的性能和存储效率始终处于最佳状态，且无需人工干预。

基于 LSM-tree 的 TDStore 查询性能是否低于原生 MySQL?

与 MySQL 的 B+ 树索引相比，LSM-tree（Log-Structured Merge-tree）在写入性能上有显著优势，但在读取性能上可能会有所牺牲。
然而，TDStore 作为一款分布式数据库，提供了多种机制来提升查询性能：

1. 垂直/水平扩容：TDStore 可以通过增加更多的节点来扩展数据库的处理能力，从而提高每秒查询率（QPS）。这是单机 MySQL 无法实现的，因为单机 MySQL 的性能受限于单个服务器的硬件资源。
2. 优化策略：即使在单节点上，TDStore 也采用了一系列优化策略来提高读取性能：
   - Leveling Compaction：TDStore 将所有数据（包括主键和索引）存储在一个大的、有序的键值对空间中，这些数据在物理磁盘上对应多个 SST 文件，分为 L0 到 L6 共七层。Leveling Compaction 策略确保了除了 L0 层之外的每一层中键的唯一性，这有助于加快查询速度。L0层较为特殊，允许文件间存在范围重叠，但 TDStore 会限制 L0 层的文件数量，通常不超过四个。当需要访问数据时，TDStore 首先检查内存 memtable，如果数据不在 memtable 中，则按层次顺序检查磁盘上的 SST 文件。由于 L1 到 L6 层的键是唯一的，因此每层只需检查一个 SST 文件即可确定目标数据是否存在。
   - Bloom Filter：在查找数据时，TDStore 使用布隆过滤器来快速排除那些不可能包含目标键的 SST 文件，这样可以避免不必要的磁盘查找，节省资源。
   - Block Cache：TDStore 利用块缓存来存储热点数据，减少对磁盘的 I/O 操作，进一步提高读取性能。

综上所述，尽管基于 LSM-tree 的 TDStore 在读取性能上可能不如优化过的 MySQL 实例，但通过分布式架构和一系列优化措施，TDStore 仍然能够提供高效的读写性能，特别是在大规模数据处理和高并发访问的场景中。

InnoDB 与 TDStore 在大事务主备延迟问题上是否有相同的表现?

InnoDB 和 TDStore 在处理大事务主备延迟问题上表现出不同的特性：

- InnoDB：InnoDB 使用二进制日志（binlog）来同步主备数据。在高并发和大数据量的场景下，大事务可能会导致主备之间的延迟，因为 binlog 的复制和重放过程可能会很耗时。
- TDStore：TDStore 是一个基于 Raft 协议的分布式数据库，它通过 Raft 日志实时同步节点间的数据。在 Raft 协议中，Leader 节点会在接收到客户端请求后，先将请求作为一个新的日志条目添加到本地日志，然后复制到 Follower 节点。只有当日志条目被复制到多数派节点并被标记为可提交状态后，Leader 节点才会响应客户端。这种设计有效减少了大事务可能导致的延迟。

  在 TDStore 中，主备节点之间的 apply 操作几乎不会有延迟。唯一的潜在延迟是 Follower 节点需要等待 Leader 节点发送下一个日志条目（或心跳）来获取可以提交的索引。然而，这个时间间隔通常非常短，可以忽略不计。

此外，TDStore 对事务的大小有限制，尤其是对于删除操作，建议将大事务拆分成多个小事务执行。这有助于避免单个事务占用过多资源，减少对系统性能的影响。
综上所述，与 InnoDB 相比，TDStore 的 Raft 协议同步机制在大事务处理上提供了更低的主备延迟，特别是在高并发和大数据量的场景下。

如果事务最终 Rollback，已经复制到 Follower 的日志会怎么处理?

在 Raft 协议中，如果一个事务最终需要回滚（rollback），已经复制到 follower 节点的日志条目将不会被应用到状态机中，也就是说，它们不会影响到底层的数据存储。这是因为这些日志条目被视为 "未提交"。
Raft 协议通过以下机制来确保 "未提交" 的日志条目不会被错误地应用：

- 维护 commitIndex：Raft 协议使用一个名为 commitIndex 的变量来跟踪已提交日志条目的最大索引。只有当日志条目的索引大于 commitIndex 时，它才会被认为是已提交的。如果事务回滚，commitIndex 将保持不变，这样就防止了 "未提交" 的日志条目被应用到状态机。

- 日志截断：在某些情况下，例如发生故障切换时，新选举出的 leader 节点可能需要截断其日志以确保集群的一致性。新 leader 节点会删除日志中的 "未提交" 条目，并将这些更改同步到 follower 节点。这样，与回滚事务相关的日志条目就会从整个集群中移除。

通过这些机制，Raft 协议保证了即使在事务回滚的情况下，也能维护集群的数据一致性和完整性。

TDStore 的 Compaction 对性能有何影响?

TDStore 中的 Compaction 过程主要包括两个动作：读取上一层文件，进行 sort-merge 操作，然后将结果写入下一层或下几层的文件。这个过程主要消耗 CPU 和 I/O 资源。因此，只要系统有足够的资源，Compaction 本身通常不会对业务性能产生明显的影响，甚至可能没有影响。
此外，由于 TDStore 采用天然的分布式架构，它可以利用所有节点的资源来进行 Compaction，这与传统的主从架构不同，在主从架构中，通常只有主库的资源被用于处理读写操作（不考虑读写分离的情况）。因此，TDStore 的分布式特性有效地弥补了可能需要为 Compaction 准备额外资源的问题。
关于 Compaction 对性能影响的担心，很大程度上是因为早期 RocksDB 实现中相对简单的 Compaction 策略，这留下了一些关于 Compaction 影响的 "传说"。然而，随着版本的不断迭代，Compaction 策略已经进行了大量的优化，使得对性能的影响更加可控。

TDStore 目前的灾备能力支持情况如何?

TDStore 具备多样化的服务高可用技术，包括实例内的多副本容灾和实例间的物理备库容灾。

- **实例内的多副本容灾**
  - **同机房三副本**：同机房内三副本组成一个实例，能够防范少数派节点故障，无法防范机房级故障。
  - **同城三机房三副本**：面向具备一个城市三个机房的场景。同城三个机房组成一个实例（每个机房是一个可用区），机房间网络延迟一般在0.5 ~ 2ms之间。能够防范少数派节点故障、单机房故障，无法防范城市级故障。
- **实例间的物理备库容灾**：当前已具备同城灾备和跨城灾备能力。在两个完全独立的实例之间提供数据同步以及切换主备关系的能力。当主库出现计划内、外的不可用情况时，备库可以接管服务。

> **注意：**
> 1. 如果应用程序是处理关键业务，对业务连续性有很高的要求，那么**应用程序应该具备与数据库相同的容灾等级**。这是因为即使数据库能够在发生故障时快速恢复，如果应用程序本身没有相应的容灾能力，那么业务仍然会受到影响。
> 2. 实例间的物理备库容灾，选择该容灾方案需要注意：**主备实例之间的数据同步是准实时的**。有两种切换模式：Switchover（计划内切换），Failover（故障切换）。Switchover 可以做到无损，Failover 是有损的（故障强切场景，通常可能会出现5秒内的数据损失，取决于备实例同步实际落后的情况）。

TDStore 是否支持 JSON?

JSON 是支持的，也支持 JSON_ARRAYAGG，JSON_OBJECTAGG 两种聚合函数。（目前跟 MySQL 一致）

TDStore 是否支持外键和全局索引?

TDStore 不支持外键和全局索引。如果需要确认迁移实例是否涉及这些功能，请联系技术支持获取扫描工具进行判断。

TDStore 支持公网/外网访问吗?

出于安全和性能考虑，TDStore 实例目前仅支持从私有网络 VPC 内网访问。

TDStore 中自增字段为何会出现跳跃现象?

在 TDStore 数据库中，自增字段暂时只保证全局唯一，不保证全局自增。
为了提高自增字段值的分配效率，TDStore 采用了分片缓存机制。例如，如果有三个计算节点，它们可能会分别缓存一段连续的自增值：

- A 节点缓存自增范围1-100；
- B 节点缓存自增范围101-200；
- C 节点缓存自增范围201-300。

当 A 节点的缓存值用尽后，它将获取下一个可用的自增范围，比如301-400。这种机制确保了即使在分布式环境下，自增字段的值也能快速分配，但同时也可能导致自增值出现跳跃，因为不同节点之间的缓存值是不连续的。

使用轻量应用服务器 Lighthouse 如何连接 TDStore 数据库?

轻量应用服务器 Lighthouse 使用腾讯云自动分配的私有网络 VPC 进行网络隔离，默认情况下内网不与云数据库等其他处于私有网络 VPC 中的腾讯云资源内网互通，需通过关联云联网实现。请参考 Lighthouse 内网连通性说明 、内网互联 。

系统提示"实例版本校验错误；请升级内核至最新版后重试"，需要怎么做?

TDStore 引擎内核在不断迭代升级中，如果您遇到上述系统提示，请 提交工单 联系腾讯云工程师为您升级引擎内核。

TDStore 是否支持海外区域服务?

TDStore 目前暂不提供海外服务。海外地域的上线正在规划中，敬请期待。

版权所有：腾讯云计算（北京）有限责任公司