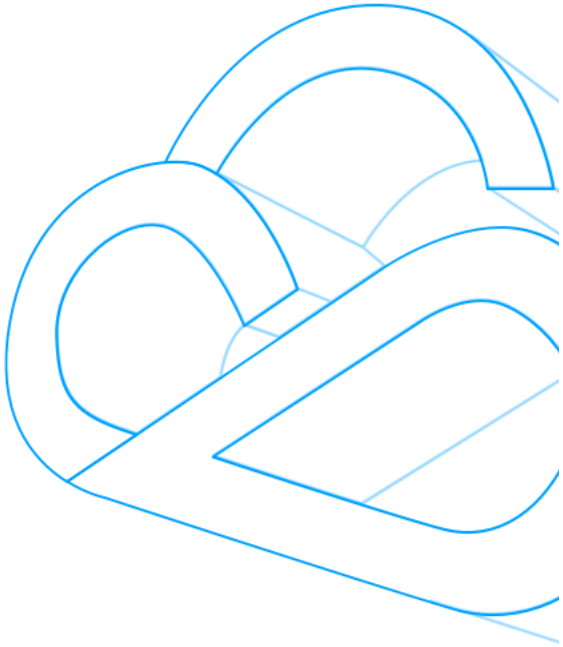
腾讯云



腾讯云数据库 TDSQL MySQL 版
分布式 V10.3.22.8/集中式 V8.0.22.8
日志参考

文档版本：
发布日期：

腾讯云计算（北京）有限责任公司

版权声明

商标声明

免责声明

本文档旨在向客户介绍本文档撰写时，腾讯云相关产品、服务的当时的整体概况，部分产品或服务在后续可能因技术调整或项目设计等任何原因，导致其服务内容、标准等有所调整。因此，本文档仅供参考，腾讯云不对其准确性、适用性或完整性等做任何保证。您所购买、使用的腾讯云产品、服务的种类、内容、服务标准等，应以您和腾讯云之间签署的合同约定为准，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。

修订记录

文档版本	发布日期	修订人	修订内容

目录

前言

文档目的

本文档用于帮助用户掌握云产品的操作方法与注意事项。

目标读者

本文档主要适用于如下对象群体：

- 客户
- 交付 PM
- 交付技术架构师
- 交付工程师
- 产品交付架构师
- 研发工程师
- 运维工程师

符号约定

本文档中可能采用的符号约定如下：

符号	说明
说明:	表示是正文的附加信息，是对正文的强调和补充。
注意:	表示有低度的潜在风险，主要是用户必读或较关键信息，若用户忽略注意消息，可能会因误操作而带来一定的不良后果或者无法成功操作。
警告:	表示有中度的潜在风险，例如用户应注意的高危操作，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。
禁止:	表示有高度潜在危险，例如用户应注意的禁用操作，如果不能避免，会导致系统崩溃、数据丢失且无法修复等严重问题。

1 概述

日志记录了系统的运行情况和功能处理情况。运维工程师可通过日志查看系统状态、实现对故障的排查定位，集群健康度的检查等。
TDSQL 分组件记录日志，包括如下主要组件：

- LVS
- ZooKeeper
- Proxy
- DB
- Monitor
- Scheduler/Manager
- OC_Agent
- Agent
- OSS
- MC
- Analysis
- Collector
- OnlineDDL
- CloudDBA
- 赤兔
- HDFS
- 多源同步组件

本文将通过介绍各运行日志的日志含义、日志存储位置、日志命名、日志示例，日志清理方式等，帮助维护工程师解读日志，快速定位问题。

- 日志含义：描述某日志记录的信息或该日志的具体用途。
- 日志命名：描述某日志的具体命名规则。
- 日志存储位置：描述日志的默认存储位置。
- 日志示例：描述某日志的内容构成，包含哪些信息要素。
- 日志清理方式：组件会生产大量日志或占用较多存储空间。用户可以配置自动化清理方案，也可以手动清除已释放空间。

级别名称	级别标志	级别含义
DEBUG	0	记录系统的工作过程，包括软件工作过程中各操作的日志信息以及相应的参数。
INFO	1	记录系统运行的工作流程以及工作流程中涉及的关键步骤。

WARN	2	记录系统运行过程中，系统可能出现的问题或异常。
ERROR	3	记录系统运行过程中发生的错误。

2 LVS 日志

日志含义

LVS 组件日志包含 lvsmanager 日志和 Keepalived 日志。

- lvsmanager 用于控制 LVS 路由信息更新等动作。lvsmanager 日志主要辅助诊断 LVS 相关功能配置是否缺失等问题。
- Keepalived 主要用于控制 VIP 漂移等检测功能，Keepalived 日志可以定位 LVS 是否做过 VIP 切换等动作。

日志命名

- Lvsmanager 日志：sys_lvsmanager.log.*
- Keepalived 日志：log

日志存储路径

- Lvsmanager 日志：/data/application/lvsmanager/log/
- Keepalived 日志：/data/application/keepalived/log/

日志格式

[日志打印时间][日志打印级别][进程 ID][对应代码文件][对应代码行号][现成对应的映射地址][线程 ID][具体日志信息]

日志示例

```
[2023-11-13 23:59:59 516911] DEBUG 14395,report.cpp:24:run,tid:0xfffe2494ee40,14509,Report thread is running.
```

日志清理

自动化清理方法

修改各节点参数文件 vi /data/application/lvsmanager/conf/lvsmanager.xml

```
…
<Log>
        <lvsmanager name="../log/sys_lvsmanager.log" log_size="536870912" log_level="0" defaultreserver="547483648"/>
</Log>
…
```

- name：表示输出日志的地址和文件名字。
- log_size：表示单个日志文件的大小。
- log_level：表示打印日志的级别。

手动清理方法

- 删除方法：按需清理．按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

3 ZooKeeper 日志

日志含义

ZooKeeper 在 TDSQL 中主要负责相关元数据的管理，比如表结构、字段信息、set 等。
ZooKeeper 日志包含事务日志、快照日志和 log4j 日志。

- 运行日志：记录 ZooKeeper 集群服务器运行日志。
- 事务日志：ZooKeeper 系统在正常运行过程中，针对所有更新操作，在返回客户端“更新成功”的响应前，ZooKeeper 会保证已经将本次更新操作的事务日志已经写到磁盘上，只有这样，整个更新操作才会生效。
- 快照日志：ZooKeeper 会定期生成一个快照，记录所有节点的数据状态。
- 告警日志：ZooKeeper 的主系统日志文件，记录 ZooKeeper 进程的本身相关信息，进程启停，报错日志，状态等。

日志命名

- 运行日志：zookeeper.out
- 事务日志：log.*
- 快照日志：snapshot.*
- 告警日志：Zookeeper.log

日志相关配置文件

- **log4j.properties**
  log4j.properties 是 ZooKeeper 的日志配置文件，它定义了日志的输出格式、级别、输出路径等。
  配置示例：
  示例中设置了日志级别为 INFO，并定义了两个 Appender，一个输出到控制台，一个输出到文件。同时，还设置了日志文件的最大大小和备份数量。

```
properties
# 设置日志级别为 INFO
log4j.rootLogger=INFO, stdout, zookeeper
# 定义输出到控制台的 Appender
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ISO8601} [zookeeper-%X{id}] %-
5p %c{1}:%L - %m%n
```

```
# 定义输出到文件的 Appender
log4j.appender.zookeeper=org.apache.log4j.RollingFileAppender
log4j.appender.zookeeper.File=${zookeeper.log.dir}/zookeeper.out
log4j.appender.zookeeper.MaxFileSize=10MB
log4j.appender.zookeeper.MaxBackupIndex=10
log4j.appender.zookeeper.layout=org.apache.log4j.PatternLayout
log4j.appender.zookeeper.layout.ConversionPattern=%d{ISO8601} [zookeeper-%X{id}] %-
5p %c{1}:%L - %m%n
```

- **zoo.cfg**

  zoo.cfg 是 ZooKeeper 的配置文件，通过修改 zoo.cfg 来改变 ZooKeeper 的运行参数，包括日志的输出路径。

  配置示例：

```
tickTime=2000
initLimit=10
syncLimit=5
dataDir=../data
dataLogDir=../log
clientPort=2118
server.100=tdsql_s187f88t1_zk1:2338:2558
server.90=tdsql_s187f88t1_zk2:2338:2558
server.80=tdsql_s187f88t1_zk3:2338:2558
maxClientCnxns=2000
autopurge.snapRetainCount=50
autopurge.purgeInterval=1
tdsqlACL=WEAK
admin.serverPort=9033
4lw.commands.whitelist=*
```

参数	说明
clientPort	zkcli.sh 客户端连接端口号。默认为 2181。
initLimit	允许 follower （相对于 leader 而言的“客户端”）连接 并同步到 leader 的初始化连接时间，它以 tickTime 的倍数来表示。当超过设置倍数的 tickTime 时间，则连接失败。
syncLimit	leader 与 follower 之间发送消息，请求 和 应答 时间长度。如果 follower 在设置的时间内不能与 leader 进行通信，那么此 follower 将被丢弃。
dataDir	ZooKeeper 将存储内存数据库快照的位置，以及数据库更新的事务日志（除非另有指定）。
dataLogDir	数据日志目录，表示机器将事务日志写入 dataLogDir 而不是 dataDir。建议使用专用日志设备，有助于避免日志记录和快照之间的竞争。

autopurge.snapRetainCount	启用后，ZooKeeper 自动清除功能将分别保留 dataDir 和 dataLogDir 中的 autopurge.snapRetainCount 最新快照以及相应的事务日志，并删除其余部分。默认为 50。最小值为 3。

日志默认存储路径

- 运行日志：/data/application/zookeeper/logs/
- 事务日志：/data/application/zookeeper/log/version-2/
- 快照日志：/data/application/zookeeper/data/version-2/
- 告警日志：/data/application/zookeeper/logs/

日志示例

- 运行日志

```
Removing file: Dec 31, 2023 8:51:50 PM  ../log/version-2/log.3008cb062
```

字段名	解释
Removing file	执行操作。
Dec 31, 2023 8:51:50 PM	时间。
../log/version-2/log.3008cb062	文件。

- 告警日志

```
2023-12-15 21:10:32.430 [myid:100] - INFO  [main:FileTxnSnapLog@124] - zookeeper.snapshot.trust.empty : false
```

字段名	解释
2023-12-15 21:10:32.430	时间。
[myid:100]	myid 标识。
INFO	日志级别。
[main:FileTxnSnapLog@124] - zookeeper.snapshot.trust.empty : false	具体信息。

日志清理

自动化清理方法

通过配置文件/data/application/zookeeper/conf/zoo.cfg 中 autopurge.snapRetainCount 参数控制系统保存的最大日志文件数，默认为 50。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

4 Proxy 日志

4.1 Proxy 日志概述

Proxy 是 TDSQL 的核心组件之一，主要提供了 SQL 转写、SQL 分发、权鉴、结果聚合、结果过滤等功能。

Proxy 日志进程

打印 Proxy 日志的有两个进程，一个是 mysql-proxy 进程，一个是 router_update 进程，可通过 ps 命令查看。

进程名称	进程描述	包含配置模块
mysql-proxy	网关的主进程，负责处理用户的 SQL 请求。	Gateway
router_update	网关的辅助进程，负责从决策集群（ZK）获取路由等信息，监听决策集群的变更信息，并实时更新相关路由信息。	Server

- 查看 Proxy 日志进程

```
[root@VM-0-2-tencentos ~]# ps -ef | grep 15001
root      89575 529024  0 15:56 pts/0    00:00:00 grep --color=auto 15001
tdsql    125651      1  0 11 月 10 ?     00:00:00 ./router_update
/data/tdsql_run/15001/gateway/conf/instance_15001.cnf
tdsql    125652 125651  0 11 月 10 ?     00:28:03 ./router_update
/data/tdsql_run/15001/gateway/conf/instance_15001.cnf
tdsql    125742      1  0 11 月 10 ?     00:00:16 ./mysql-proxy
/data/tdsql_run/15001/gateway/conf/instance_15001.cnf
tdsql    125743 125742  6 11 月 10 ?     19:00:37 ./mysql-proxy
/data/tdsql_run/15001/gateway/conf/instance_15001.cnf
```

- 进程启停命令

```
su - tdsql
cd/data/tdsql_run/${PORT}/gateway/bin
./start.shinstance_${PORT}#启动
./stop.shinstance_${PORT}#停止
./restart.shinstance_${PORT}#重启
```

Proxy 日志分类

- 网关系统日志：sys_instance.*
- 路由日志：route_instance.*
- 接口名称日志：interf_ instance.*
- SQL 语句日志：sql_.*
- 慢查询日志：slow_sql.*

4.2 网关系统日志 sys_instance.*

日志含义

记录 Proxy 进程本身执行过程中打印的日志信息。

日志命名

网关系统日志：sys_instance.*

日志默认存储路径

/data/tdsql_run/port/gateway/log/

日志示例

```
[2023-11-22 08:45:04 937009] ERROR 0,/data/landun/workspace/proxy/src/proxy/proxy-no-
shard.cpp:701:no_shard_read_query_result,tid:0x7f59ef3fe700,con:0x7f59eca6d000,in
log_sql,con:0x7f59eca6d000 qstat.query_statut is not ok sql
```

日志基础释义：

字段名	解释
[2023-11-22 08:45:04 937009]	时间戳，表示日志记录的时间
ERROR	日志级别，表示该日志记录的级别为错误
0	错误码，表示该错误的错误码为 0
/data/landun/workspace/proxy/src/proxy/proxy-no-shard.cpp:701:no_shard_read_query_result	日志记录的位置，表示该错误发生在 proxy-no-shard.cpp 文件的第 701 行 no_shard_read_query_result 函数中
tid:0x7f59ef3fe700	线程 ID，表示该错误发生的线程 ID 为 0x7f59ef3fe700
con:0x7f59eca6d000	连接 ID，表示该错误发生的连接 ID 为 0x7f59eca6d000
connection health check:client is closed!	错误信息，表示该错误是由于客户端关闭连接导致的连接健康检查失败
sock: { fd: 200, src address: { address name: 172.21.16.2:53297}, dst address: { address name: 172.21.16.17:15001}, socket_type: 1, last_packet_id:	与该错误相关的套接字信息，包括套接字文件描述符、源地址、目标地址、套接字类型和最后一个数据包的 ID 等信息

日志清理

自动化清理方法

1. 修改各节点配置文件（请用实际端口号替换以下路径）。

vi /data/tdsql_run/{port}/gateway/conf/instance_port.cnf

```
…
<clean time="14" size="10" thresh="15" interval="3" speed="32" />
…
```

参数	说明
time	**说明**：清除日志，当超过设置天数时，日志将被自动删除。 **取值范围**：0~140 **默认值**：14 **单位**：天。
size	**说明**：清除日志大小，当超过设置值时，日志将被自动删除。优先按照超过设置天数的时间顺序删除日志。 **取值范围**：0~10000 **默认值**：10 **单位**：G。
thresh	**说明**：阈值设置，如果超过这个大小，加速清理。 **取值范围**: 0~2147483647 **默认值**：15 **单位**: GB
interval	**说明**：用于控制日志清理的周期。 **取值范围**：0 到 2147483647

	默认值：3 **单位**：秒
speed	**说明**：用于控制日志清理的速度。 **取值范围**：1 到 2147483647 **默认值**：32 **单位**：M/S

2. 依次重启对应节点 Proxy。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

4.3 路由日志 route_instance.*

日志含义

route-update 进程本身执行的日志，主要记录从决策集群（ZK）获取路由等信息，监听决策集群的变更信息，并实时更新相关路由信息等相关记录。

日志命名

路由日志：route_instance.*

日志默认存储路径

/data/tdsql_run/port/gateway/log/

日志示例

```
[2023-11-22 16:53:08 611043] ERROR 0,/data/landun/workspace/proxy/router_src/proxy-router-util.cpp:29:delfile,tid:0x7f85b97f9700,slowdelfile file:../log/sql_instance_15001.2023-11-22.18 success
```

日志基础释义：

字段名	解释
[2023-11-22 16:53:08 611043]	时间戳，表示日志记录的时间
ERROR	日志级别，表示该日志记录的级别为 ERROR
0	错误码，表示该错误的错误码为 0
/data/landun/workspace/proxy/router_src/proxy-router-util.cpp:29:delfile	日志记录的位置，表示该日志发生在 proxy-router-util.cpp 文件的第 29 行 delfile 函数中
tid:0x7f85b97f9700	线程 ID，表示该日志发生的线程 ID 为 0x7f85b97f9700
get setrun node success, value: {"cgroup_cpu":"soft","degrade_flag":0,"history_ids":[{"id":"set_1665302135_1","ti	详细信息，从 ZK 获取的详细信息

mestamp":1665302135,"trace_id":"ac1510111684221689878100424541","type":0}],"kpstatus":0,"master":{"alive":"0","city":"default","election":true,"hb_err":"0","idc":"idc2","idc_weight":"100","losthbtime":"0","name":"172.21.16.17_4001","sqlasyn":"0","trace_id":"ac151011168422168987810 0424541","weight":"1","zone":"none"},"password":"22ngmWj5@%UF0#8aH","proxy":[{"name":"172.21.16.12_15001","trace_id":"ac151011168422168987810042454 1"},{"name":"172.21.16.13_15001","trace_id":"ac151011168422168987810042454 1"},{"name":"172.21.16.17_15001","trace_id":"ac151011168422168987810042454 1"},{"name":"172.21.16.7_15001","trace_id":"ac151011168422168987810042454 1"}],"read_only":"0","resource_info":{"cpu":100,"data_disk":30000,"log_disk":30000,"mem":1000,"trace_id":"ac151011168422168987 8100424541"},"set":"set_1665302135_1","slave":[{"alive":"-1","city":"default","election":true,"hb_err":"1","idc":"idc1","idc_weight":"100","losthbtime":"1684773775","name":"172.21.16.12_4001","sqlasyn":"0","trace_id":"ac151 011168422168987810042454 1","weight":"1","zone":"none"}],"specid":32768,"status":0,"trace_id":"ac151011168422168987810 00424541","uniqueid":"unique_1665302135_1","user":"tdsqlsys_normal","wait_slaves_run":1}	
path: /tdsqlzk/sets/set@set_1665302135_1/setrun@set_1665302135_1	zk 的路径信息，表示在 /tdsqlzk/sets/set@set_1665302135_1/setrun@set_1665302135_1 路径下获取的

注意：
不同的日志记录格式可能会有所不同，字段的含义和顺序也可能会有所不同。

日志清理

自动化清理方法

1. 修改各节点配置文件（请用实际端口号替换以下路径）。

vi /data/tdsql_run/{port}/gateway/conf/instance_port.cnf

```
…
<clean time="14" size="10" thresh="15" interval="3" speed="32" />
…
```

参数	说明
time	**说明**：清除日志，当超过设置天数时，日志将被自动删除。 **取值范围**：0~140 **默认值**：14 **单位**：天。
size	**说明**：清除日志大小，当超过设置值时，日志将被自动删除。优先按照超过设置天数的时间顺序删除日志。 **取值范围**：0~10000 **默认值**：10 **单位**：G。
thresh	**说明**：阈值设置，如果超过这个大小，加速清理。 **取值范围**: 0~2147483647 **默认值**：15 **单位**: GB
interval	**说明**：用于控制日志清理的周期。 **取值范围**：0 到 2147483647 **默认值**：3 **单位**：秒
speed	**说明**：用于控制日志清理的速度。 **取值范围**：1 到 2147483647 **默认值**：32 **单位**：M/S

2. 依次重启对应节点 Proxy。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

4.4 接口日志 interf_ instance.*

日志含义

接口日志主要存储了由客户端传输给服务器的原始 SQL 信息，包括客户端、后端服务器等。日志格式以&和空格为分隔符。

日志命名

接口日志：interf_ instance.*

日志默认存储路径

/data/tdsql_run/port/gateway/log/

日志示例

```
[2021-08-12 15:55:58 386568] INFO
topic=group_1623641961_38567&tid=57332&con=0x7ff118c3e000&qid=4751805-1628054013-934461&splited=0&clientIP=10.xx.xx.xx:xx&proxyHost=10.xx.xx.xx:xx&sql_size=166&sql_type=3&sub_sql_type=0&sql=select name, sub_system_code, system_code from tsp_mutex where name%3D'ksys_plrenw_health_check_mutex' and sub_system_code%3D'dept_bat' and system_code%3D'dept' for update&db=core_acc&user=appuser&10.xx.xx.xx:xx=2&backend=10.xx.xx.xx:xx&autocommit=0&new_connnum=0&conn_tc=0&select_result_sum=1&affect_num=0&hold_conns=1&resultcode=0&mc_request=0&timecost=0
```

日志基础释义：

字段名	解释
[2021-08-12 15:55:58 386568]	日志记录时间。
INFO	日志级别，支持 INFO、WARNING、ERROR。
topic=group_1623641961_38567	对应的实例。
tid=57332	线程 ID。
con=0x7ff118c3e000	对应的是 proxy 处理过程中连接对象的地址，和 tid 一起用于定位用户连接生命周期内的所有操作。
qid=4751805-1628054013-934461	Proxy 生成的事务 ID。
splited=0	● 0：表示原始查询。 ● 1：表示 proxy 改写后的查询 。
clientIP=10.xx.xx.xx:xx	客户端的 IP 和端口。

proxyHost=10.xx.xx.xx:xx	连接 proxy 的 IP 和端口。
sql_size=166	传输的 SQL 长度（字节）。
sql_type=3	用户在握手后发送的请求类型，常用的有 3（COM_QUERY），22 和 23（对应 PREPARE 和 EXECUTE）。
sub_sql_type=0	子查询类型，一般都是 0（select）。
sql=select name, sub_system_code, system_code from tsp_mutex where name%3D'ksys_plrenw_health_check_mutex' and sub_system_code%3D'dept_bat' and system_code%3D'dept' for update	执行的 SQL 语句。
db=core_acc	数据库名称。
user=appuser	数据库用户。
10.xx.xx.xx:xx=2	数据库的 IP 和端口。
backend=10.xx.xx.xx:xx	后端数据库的 IP 和端口。
autocommit=0	是否自动提交。
new_connnum=0	新建连接的数量
conn_tc=0	新建连接的花费的时间。
select_result_sum=1	查询得到的结果集总量。
affect_num=0	操作影响的行数。
hold_conns=1	请求结束后，仍然占用的连接数量。非事物请求，应该是 0。
resultcode=0	错误码。
mc_request=0	是否开启全局一致性读。
timecost=0	SQL 耗时。

日志清理

自动化清理方法

1. 修改各节点配置文件（请用实际端口号替换以下路径）。

vi /data/tdsql_run/{port}/gateway/conf/instance_port.cnf

```
…
<clean time="14" size="10" thresh="15" interval="3" speed="32" />
…
```

参数	说明
time	**说明**：清除日志，当超过设置天数时，日志将被自动删除。 **取值范围**：0~140 **默认值**：14 **单位**：天。

size	**说明**：清除日志大小，当超过设置值时，日志将被自动删除。优先按照超过设置天数的时间顺序删除日志。 **取值范围**：0~10000 **默认值**：10 **单位**：G。
thresh	**说明**：阈值设置，如果超过这个大小，加速清理。 **取值范围**: 0~2147483647 **默认值**：15 **单位**: GB
interval	**说明**：用于控制日志清理的周期。 **取值范围**：0 到 2147483647 **默认值**：3 **单位**：秒
speed	**说明**：用于控制日志清理的速度。 **取值范围**：1 到 2147483647 **默认值**：32 **单位**：M/S

2. 依次重启对应节点 Proxy。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

4.5 SQL 语句日志 sql_.*

日志含义

SQL 通过 Proxy 层改写，分发到各个 DB 节点执行，并取出数据。SQL 日志中记录了在 set 中具体执行的 SQL。日志以空格为分隔符，以:为赋值符号。

日志命名

SQL 语句日志：sql_.*

日志默认存储路径

/data/tdsql_run/port/gateway/log/

日志示例

```
[2021-08-11 23:59:59 661067] DEBUG tid:64744 con:0x7f8595b23800 qid:8624534-1628419255-1298688 user:ksrcbadmin C:132.xx.xx.xx:xx G:132.xx.xx.xx:xx S:132.xx.xx.xx:xx timecost:0.979(ms) inj_id:1 sql:3,387 "/*2:8624534-1628419255-1298688*/select name, sub_system_code, system_code from tsp_mutex where name='ksys_plrenw_health_check_mutex' and sub_system_code='dept_bat' and system_code='dept' for update"
```

日志基础释义：

字段名	解释
[2021-08-11 23:59:59 661067]	日志记录时间。
DEBUG	日志级别，支持 DEBUG、WARNING、REEOR。
tid:64744	线程 iID。
con:0x7f8595b23800	对应的是 proxy 处理过程中连接对象的地址，和 tid 一起用于定位用户连接生命周期内的所有操作 。
qid:8624534-1628419255-1298688	proxy 生成的 sql id
user:xxxxxx	用户名。
C:132.xx.xx.xx:xx	客户端连接 IP 和端口。
G:132.xx.xx.xx:xx	网关连接 IP 和端口。
S:132.xx.xx.xx:xx	数据库服务器 IP 和端口 。
timecost:0.979(ms)	语句耗时。
inj_id:1	这个是 proxy 拆分 sql 语句之后，标记这个语句是属于原始语句生成的，还是 proxy 自己补充的（比如 xa start） 。

sql:3,387 "/*2:8624534-1628419255-1298688*/select name, sub_system_code, system_code from tsp_mutex where name='ksys_plrenw_health_check_mutex' and sub_system_code='dept_bat' and system_code='dept' for update"	执行的 SQL 语句。

日志清理

自动化清理方法

1. 修改各节点配置文件（请用实际端口号替换以下路径）。

vi /data/tdsql_run/{port}/gateway/conf/instance_port.cnf

```
…
<clean time="14" size="10" thresh="15" interval="3" speed="32" />
…
```

参数	说明
time	**说明**：清除日志，当超过设置天数时，日志将被自动删除。 **取值范围**：0~140 **默认值**：14 **单位**：天。
size	**说明**：清除日志大小，当超过设置值时，日志将被自动删除。优先按照超过设置天数的时间顺序删除日志。 **取值范围**：0~10000 **默认值**：10 **单位**：G。
thresh	**说明**：阈值设置，如果超过这个大小，加速清理。 **取值范围**: 0~2147483647 **默认值**：15 **单位**: GB
interval	**说明**：用于控制日志清理的周期。 **取值范围**：0 到 2147483647 **默认值**：3 **单位**：秒
speed	**说明**：用于控制日志清理的速度。 **取值范围**：1 到 2147483647 **默认值**：32 **单位**：M/S

2. 依次重启对应节点 Proxy。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

4.6 慢查询日志 slow_sql.*

日志含义

慢查询日志主要存储 DDL 语句及响应时间超过 1s 的 DML 语句。

日志命名

慢查询日志：slow_sql.*

日志默认存储路径

/data/tdsql_run/port/gateway/log/

日志示例

```
# Time: 210910 16:33:17 432697
# User@Host: mc_test[mc_test] @  [10.xx.xx.xx:xx]
# Backend_host: 10.xx.xx.xx:xx,10.xx.xx.xx:xx
# Thread_id: 1  Schema: test  QC_hit: DEBUG
# Query_time: 0.214000 Lock_time: 0.000000  Rows_sent: 0  Rows_examined: 0
/*4194305-1631262696-7*/create table t1(acct_id bigint(20) primary key, ammount bigint(20))
shardkey=acct_id;
```

日志基础释义：

字段名	解释
Time: 210910 16:33:17 432697	时间戳，SQL 执行的时间。
User@Host: mc_test[mc_test] @ [10.xx.xx.xx:xx]	用户名，mysql 用户比较特殊，是用户+host 来进行权限控制。
Backend_host: 10.xx.xx.xx:xx,10.xx.xx.xx:xx	后台执行的 DB 节点地址 。
Thread_id: 1	线程 ID。
Schema: test	数据库名 。
QC_hit: DEBUG	查询级别。
Query_time: 0.214000	SQL 执行消耗的时间。
Lock_time: 0.000000	SQL 执行过程中获取锁消耗的时间。
Rows_sent: 0	推送数据行数。
Rows_examined: 0	扫描数据行数 。
/*4194305-1631262696-7*/create	具体执行。

table t1(acct_id bigint(20) primary key, ammount bigint(20)) shardkey=acct_id;	

日志清理

自动化清理方法

1. 修改各节点配置文件（请用实际端口号替换以下路径）。

vi /data/tdsql_run/{port}/gateway/conf/instance_port.cnf

```
…
<clean time="14" size="10" thresh="15" interval="3" speed="32" />
…
```

参数	说明
time	**说明**：清除日志，当超过设置天数时，日志将被自动删除。 **取值范围**：0~140 **默认值**：14 **单位**：天。
size	**说明**：清除日志大小，当超过设置值时，日志将被自动删除。优先按照超过设置天数的时间顺序删除日志。 **取值范围**：0~10000 **默认值**：10 **单位**：G。
thresh	**说明**：阈值设置，如果超过这个大小，加速清理。 **取值范围**: 0~2147483647 **默认值**：15 **单位**: GB
interval	**说明**：用于控制日志清理的周期。 **取值范围**：0 到 2147483647 **默认值**：3 **单位**：秒
speed	**说明**：用于控制日志清理的速度。 **取值范围**：1 到 2147483647 **默认值**：32 **单位**：M/S

2. 依次重启对应节点 Proxy。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

5 DB 日志

5.1 DB 日志概述

DB 日志分类

DB 日志系统分为四种：错误日志（Error log）、慢日志（Slow log）、查询日志（General query log）和二进制日志（Binary log）。

- 错误日志：记录了 MySQL 启动、运行或停止期间出现的问题。这些信息可以帮助管理员定位问题并找出可能的解决方案。
- 查询日志：记录了 MySQL 服务器接收到的所有查询和命令。这可以帮助你了解数据库的使用情况，但因为它记录了所有的查询，所以可能会占用大量的磁盘空间。
- 慢查询日志：MySQL 中一种非常重要的日志类型，用于记录执行时间较长的 SQL 查询。通过记录和分析慢查询日志，可以找出哪些查询导致数据库性能下降，从而优化数据库性能。
- 二进制日志：MySQL 的核心日志文件，记录了所有的数据修改事件，如插入、更新和删除操作。它主要用于数据库的恢复和管理，但因为其包含了大量的数据修改事件，所以它的体积可能会非常大。

DB 进程

进程名称	进程描述	包含配置模块
mysqld_safe	DB 守护进程	mysqld_safe
mysqld	DB 服务进程	mysqld

DB 日志进程启停命令：

```
su - tdsql
cd/data/tdsql_run/${PORT}/mysql-server-8.0.24/install/
./startmysql_cgroup.sh${PORT}      #启动
./stopmysql.sh${PORT}              #停止
./restartmysql_cgroup.sh${PORT}    #重启
```

5.2 错误日志 mysqld.err.*

日志含义

在 MySQL 服务器运行过程中，如果发生了错误或警告，MySQL 服务器会将相关信息记录到 mysqld.err 文件中。这些信息包括错误代码、错误消息、错误发生的时间、错误发生的位置等。通过查看 mysqld.err 文件，可以了解 MySQL 服务器的运行状态和发生的错误，以便及时排查和解决问题。如服务启动失败，硬盘空间满，oom，实例 crash 等。这些信息可以帮助管理员定位问题并找出可能的解决方案。

日志命名

错误日志：mysqld.err.*

日志默认存储路径

/data1/tdengine/log/｛port｝/dblogs/

日志相关配置文件

在配置文件/data/tdsql_run/$PORT/$VERSION/etc/my_$PORT.cnf 中设置 log_error 参数，指定错误日志的文件路径。

例如：log_error = /path/to/error.log

日志示例

```
[ERROR] 2023-04-25 10:00:00 mysqld: [ERROR] InnoDB: cannot read page 1 in file "ibdata1"
```

日志基础释义：

字段名	解释
[ERROR]	日志级别。
2023-04-25 10:00:00	日志打印时间戳。
mysqld:	代表 Mysqld server 层的报错。
[ERROR] InnoDB	代表引擎层的报错。
cannot read page 1 in file "ibdata1"	具体报错信息。

日志清理

自动化清理方法

系统自动清理 4 天前的错误日志。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

5.3 慢日志 host_name.*

日志含义

慢查询日志是 MySQL 中一种非常重要的日志类型，用于记录执行时间较长的 SQL 查询。通过记录和分析慢查询日志，可以找出哪些查询导致数据库性能下降，从而优化数据库性能。

日志命名

慢日志：host_name.*

日志默认存储路径

/data1/tdengine/data/｛port｝/dbdata_raw/data/

日志相关配置文件

在配置文件/data/tdsql_run/$PORT/$VERSION/etc/my_$PORT.cnf 中设置 slow_query_log 参数为 1，并设置 slow_query_log_file 参数指定慢查询日志的文件路径。同时，需要设置 long_query_time 参数指定慢查询的时间阈值。

```
slow_query_log = 1
slow_query_log_file = /path/to/slow-query.log  #指定慢查询日志的文件路径
long_query_time = 2                    # 单位为秒，表示记录执行时间超过 2 秒的查询
```

日志示例

```
[2023-04-25 10:30:00] Query 456 from localhost @ mydatabase user 'root' IDENTIFIED BY 'password' took 5.2 seconds to execute
SELECT * FROM mytable WHERE id = 1;
```

日志基础释义：

字段名	解释
[2023-04-25 10:30:00]	日志时间。
Query	慢查询的类型。
456	processlist Id。
from localhost @ mydatabase user 'root' IDENTIFIED BY 'password'	用户认证信息。

took 5.2	花费时间。
SELECT * FROM mytable WHERE id = 1;	具体执 SQL。

日志清理

自动化清理方法

系统自动清理 4 天前的日志。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

5.4 二进制日志 binlog.*

日志含义

binlog 是 MySQL 数据库中的二进制日志文件，用于记录数据库的所有修改操作，包括增、删、改等操作。binlog 文件是 MySQL 数据库的核心组成部分，可以用于数据恢复、主从复制等功能。在主从复制中，主服务器会将自己的 binlog 文件发送给从服务器，从服务器会将这些日志文件应用到自己的数据库中，以保证数据的一致性。

日志命名

二进制日志：binlog.*

日志默认存储路径

/data1/tdengine/log/｛port｝/dblogs/bin/

日志相关配置文件

开启查询日志的方法是在配置文件/data/tdsql_run/$PORT/$VERSION/etc/my_$PORT.cnf 中设置 general_log 参数为 1，并设置 general_log_file 参数指定查询日志的文件路径。

```
log_bin = 1
log_bin_basename = /path/to/binlog/mysql-bin.log #指定二进制日志的文件路径
```

解析 Binlog

解析 binlog 时，常用下面三种方法：

```
mysqlbinlog -v binlog.00000N  > v.sql
mysqlbinlog -vv binlog.00000N  > vv.sql
mysqlbinlog -vv --base64-output=decode-rows binlog.00000N > vvbase64.sql 说明：
```

参数说明：

- -v：解析出行信息。
- -vv：在-v 的基础上增加了列数据类型的注释信息。
- --base64-output=decode-rows：该参数会屏蔽掉解析后文件中的二进制内容。
- **-v 示例：**

```
# at 381
```

```
#220226  7:23:28 server id 420450214  end_log_pos 451   Table_map: `sysdb`.`statustableforhb`
mapped to number 97
# at 451
#220226  7:23:28 server id 420450214  end_log_pos 553   Update_rows: table id 97 flags:
STMT_END_F

BINLOG '
8GQZYhOmjw8ZRgAAAMMBAAAAAGEAAAAAAAEABXN5c2RiABBzdGF0dXN0YWJsZW
ZvcmhiAAT+DwMR
Bf48PAAAAAEBAAIBUw==
8GQZYh+mjw8ZZgAAACkCAAAAAGEAAAAAAAEAAgAE//8ADDdmYjM4OWZmYjcwMA
wxNzIuMjcuMjUu
MTWmDwAAYhlk7QAMN2ZiMzg5ZmZiNzAwDDE3Mi4yNy4yNS4xNaYPAABiGWTw
'/*!*/;
### UPDATE `sysdb`.`statustableforhb`
### WHERE
###   @1='7fb389ffb700'
###   @2='172.27.25.15'
###   @3=4006
###   @4=1645831405
### SET
###   @1='7fb389ffb700'
###   @2='172.27.25.15'
###   @3=4006
###   @4=1645831408
```

- **-vv 示例：**

```
# at 381
#220226  7:23:28 server id 420450214  end_log_pos 451   Table_map: `sysdb`.`statustableforhb`
mapped to number 97
# at 451
#220226  7:23:28 server id 420450214  end_log_pos 553   Update_rows: table id 97 flags:
STMT_END_F

BINLOG '
8GQZYhOmjw8ZRgAAAMMBAAAAAGEAAAAAAAEABXN5c2RiABBzdGF0dXN0YWJsZW
ZvcmhiAAT+DwMR
Bf48PAAAAAEBAAIBUw==
8GQZYh+mjw8ZZgAAACkCAAAAAGEAAAAAAAEAAgAE//8ADDdmYjM4OWZmYjcwMA
wxNzIuMjcuMjUu
MTWmDwAAYhlk7QAMN2ZiMzg5ZmZiNzAwDDE3Mi4yNy4yNS4xNaYPAABiGWTw
'/*!*/;
### UPDATE `sysdb`.`statustableforhb`
```

```
### WHERE
###   @1='7fb389ffb700' /* STRING(60) meta=65084 nullable=0 is_null=0 */
###   @2='172.27.25.15' /* VARSTRING(60) meta=60 nullable=0 is_null=0 */
###   @3=4006 /* INT meta=0 nullable=0 is_null=0 */
###   @4=1645831405 /* TIMESTAMP(0) meta=0 nullable=0 is_null=0 */
### SET
###   @1='7fb389ffb700' /* STRING(60) meta=65084 nullable=0 is_null=0 */
###   @2='172.27.25.15' /* VARSTRING(60) meta=60 nullable=0 is_null=0 */
###   @3=4006 /* INT meta=0 nullable=0 is_null=0 */
###   @4=1645831408 /* TIMESTAMP(0) meta=0 nullable=0 is_null=0 */
```

- **--base64-output=decode-rows 示例：**

```
# at 381
#220226  7:23:28 server id 420450214  end_log_pos 451   Table_map: `sysdb`.`statustableforhb`
mapped to number 97
# at 451
#220226  7:23:28 server id 420450214  end_log_pos 553   Update_rows: table id 97 flags:
STMT_END_F
### UPDATE `sysdb`.`statustableforhb`
### WHERE
###   @1='7fb389ffb700' /* STRING(60) meta=65084 nullable=0 is_null=0 */
###   @2='172.27.25.15' /* VARSTRING(60) meta=60 nullable=0 is_null=0 */
###   @3=4006 /* INT meta=0 nullable=0 is_null=0 */
###   @4=1645831405 /* TIMESTAMP(0) meta=0 nullable=0 is_null=0 */
### SET
###   @1='7fb389ffb700' /* STRING(60) meta=65084 nullable=0 is_null=0 */
###   @2='172.27.25.15' /* VARSTRING(60) meta=60 nullable=0 is_null=0 */
###   @3=4006 /* INT meta=0 nullable=0 is_null=0 */
###   @4=1645831408 /* TIMESTAMP(0) meta=0 nullable=0 is_null=0 */
```

日志示例

以下是一个 binlog 解析后的内容（截取出的一部分）：

```
...
# at 30303595
#220228 18:43:03 server id 419884197  end_log_pos 30303662  Query   thread_id=112
exec_time=0 error_code=0
SET TIMESTAMP=1646044983/*!*/;
/*!\C utf8mb3 *//*!*/;
SET
@@session.character_set_client=33,@@session.collation_connection=83,@@session.collation_serv
er=83/*!*/;
BEGIN
/*!*/;
```

...

日志基础释义：

字段名	解释
at 30303595	binlog 中 event 事件发生时的位置信息。
220228 18:43:03	binlog 中 event 事件发生的时间。
server id 419884197	服务器的 id 值，可以通过 show variables like '%server_id%';获取。
end_log_pos 30303662	binlog 中事件结束的时间。
Query	代表这是一个查询类型的事件。
thread_id	线程 ID。
exec_time	执行时间。
error_code	返回的错误码是 0。

```
     1 # at 30303662
     2 #220228 18:43:03 server id 419884197  end_log_pos 30303732  Table_map:
`sysdb`.`statustableforhb` mapped to number 97
     3 # at 30303732
     4 #220228 18:43:03 server id 419884197  end_log_pos 30303832  Update_rows: table id 97
flags: STMT_END_F
     5 ### UPDATE `sysdb`.`statustableforhb`
     6 ### WHERE
     7 ###   @1='7f805bfff700'
     8 ###   @2='172.27.25.6'
     9 ###   @3=4005
    10 ###   @4=1646044980
    11 ### SET
    12 ###   @1='7f805bfff700'
    13 ###   @2='172.27.25.6'
    14 ###   @3=4005
    15 ###   @4=1646044983
    16 # at 30303832
    17 #220228 18:43:03 server id 419884197  end_log_pos 30303859  Xid = 39768241
    18 COMMIT/*!*/;
    19 # at 30303859
    20 #220228 18:43:04 server id 419884197  end_log_pos 30303941  GTID
last_committed=89731    sequence_number=89735   rbr_only=yes    original_committe
d_timestamp=1646044984169729   immediate_commit_timestamp=1646044984186009
transaction_length=346
    21 /*!50718 SET TRANSACTION ISOLATION LEVEL READ COMMITTED*//*!*/;
    22 # original_commit_timestamp=1646044984169729 (2022-02-28 18:43:04.169729 CST)
    23 # immediate_commit_timestamp=1646044984186009 (2022-02-28 18:43:04.186009 CST)
    24 /*!80001 SET @@session.original_commit_timestamp=1646044984169729*//*!*/;
    25 /*!80014 SET @@session.original_server_version=80024*//*!*/;
    26 /*!80014 SET @@session.immediate_server_version=80024*//*!*/;
    27 SET @@SESSION.GTID_NEXT= 'd42b4f09-954d-11ec-9cf5-52540038f10e:479200'/*!*/;
~
```

此部分重点内容说明：

行数	字段名	解释
2	Table_map	于 ROW 格式的 Binlog 在记录 DML 语句的数据时，总会先写入一个 Table_map 事件，用于记录表结构相关元数据信息，比如数据库名称，表名称，表的字段类型，表的字段元数据等。
17	Xid = 39768241	在事务提交时，不管是 STATEMENT 还是 ROW 格式

		的 binlog，都会在末尾添加一个 XID_EVENT 事件代表事务的结束。该事件记录了该事务的 ID，当进行崩溃恢复时，根据事务在 binlog 中的提交情况来决定是否提交存储引擎中状态为 prepared 的事务。
20	last_committed、sequence_number、rbr_only	多线程复制的相关参数。

日志清理

自动化清理方法

日志盘超过 50%会自动清理。

手动清理方法

- 清理方法：确认复制关系正常，无主从延迟，通过 purge 命令清理相关日志。
- 影响：若复制关系异常，清除主上的二进制日志，则无法正常恢复复制关系。

5.5 Relaylog 日志 relay.*

日志含义

relaylog 是 MySQL 数据库中的中继日志文件，用于在主从复制中传递和存储主服务器的 binlog 文件。当从服务器连接到主服务器时，主服务器会将自己的 binlog 文件发送给从服务器，从服务器会将这些日志文件存储到自己的 relaylog 文件中。然后，从服务器会将 relaylog 文件中的日志应用到自己的数据库中，以保证数据的一致性。relaylog 文件只在从服务器上存在，不会在主服务器上存在。

日志命名

relaylog 日志：relay.*

日志默认存储路径

/$LOGPATH/$PORT/dblogs/relay/relay.log

日志相关配置文件

在配置文件/data/tdsql_run/$PORT/$VERSION/etc/my_$PORT.cnf 中设置 relay-log 参数指定 relaylog 日志的文件路径。

```
relay-log = /data1/tdengine/log/4008/dblogs/relay/relay.log
```

日志示例

```
# at 6730669
#230611  0:00:54 server id 269243903  end_log_pos 6730743        Query  thread_id=3500015
          exec_time=0    error_code=0
SET TIMESTAMP=1686412854/*!*/;
BEGIN
```

字段名	解释
at 6730669	日志 position 号。表示在日志中的位置。
230611 0:00:54	event 时间。
end_log_pos 6730743	event 结束 position 号。
Query	event 类型。

thread_id=3500015	线程 ID。
exec_time=0	执行完成时间与主库执行时间差。
error_code=0	错误编码。
SET TIMESTAMP=1686412854/*!*/;	时间戳。
BEGIN	event 内容。

日志清理

自动化清理方法

从库执行完 relaylog 后自动清理。

手动清理方法

- 清理方法：确认复制关系正常，无主从延迟，通过 purge 命令清理相关日志。
- 影响：若复制关系异常，清除主上的二进制日志，则无法正常恢复复制关系。

5.6 查询日志 General query log

日志含义

查询日志记录了 MySQL 服务器接收到的所有查询和命令。这可以帮助你了解数据库的使用情况，但因为它记录了所有的查询，所以可能会占用大量的磁盘空间。

日志命名

查询日志： General query log

日志默认存储路径

/data1/tdengine/data/port/dbdata_raw/data/

日志相关配置文件

开启查询日志的方法是在配置文件/data/tdsql_run/$PORT/$VERSION/etc/my_$PORT.cnf 中设置 general_log 参数为 1，并设置 general_log_file 参数指定查询日志的文件路径。

```
general_log = 1
general_log_file = /path/to/query.log  #指定查询日志的文件路径
```

日志示例

```
2024-01-05T11:00:20.269208+08:00          99 Query     show variables where Variable_name in ('sqlasync_after_sync' , 'sqlasyn')
```

字段名	解释
2024-01-05T11:00:20.269208+08:00	时间
99	线程号

Query	操作类型
show variables where Variable_name in ('sqlasync_after_sync' , 'sqlasyn')	具体操作

日志清理

自动化清理方法

默认不开启。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

6 Monitor 日志

日志含义

- dstat 日志：实时监控 CPU、内存、磁盘、网络等方面的性能。
- iostat 日志：监控 IO 使用情况。
- iotop 日志：监控各个进程 IO 使用情况。
- meminfo 日志：监控内存使用情况。
- top 日志：实时记录系统中各个进程的资源占用状况。
- vmstat 日志：记录内核线程、虚拟内存、磁盘、陷阱和 CPU 活动的统计信息。

日志命名

- dstat 日志：dstatlog.*
- iostat 日志：iostatlog.*
- iotop 日志：iotoplog.*
- meminfo 日志：meminfo.*
- top 日志：toplog.*
- vmstat 日志：vmstat.log.*

日志默认存储路径

- dstat 日志：/data/monitorlog/dstatlog/
- iostat 日志：/data/monitorlog/iostatlog/
- iotop 日志：/data/monitorlog/iotoplog/
- meminfo 日志：/data/monitorlog/meminfo/
- top 日志：/data/monitorlog/toplog/
- vmstat 日志：/data/monitorlog/vmstat/

日志格式

相关日志格式同 Linux 系统命令（Dstat、iostat、iotop、meminfo、top、vmstat）。

日志清理

仅支持手动清理。

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

7 Scheduler/Manager 日志

日志含义

- Manager 系统日志：记录 Manager 进程运行详细信息。
- Scheduler 系统日志：记录 Scheduler 进程运行详细信息。

日志命名

- Manager 系统日志：sys_manager.*
- Scheduler 系统日志：sys_scheduler.*

日志默认存储路径

/data/scheduler_log/

日志示例

sys_manager 和 sys_scheduler 日志格式相同：

```
[2023-12-20 00:00:02 071774] INFO
384993,func_dealzk.cpp:669:GetZnodeJson,tid:0x7f25e6a5da00,384993,get
/tdsqlzk/manager/manager_mode content {"no_proxy":0,"value":1}
```

日志基础释义：

字段名	解释
[2023-12-20 00:00:02 071774]	日志打印时间。
INFO	日志级别。
384993	线程 ID。
func_dealzk.cpp	涉及到的源码文件。
669	具体在文件中的行数。
GetZnodeJson	涉及到的函数。
tid:0x7f25e6a5da00	线程在内存中的映射地址。
384993	线程 ID。
get /tdsqlzk/manager/manager_mode content {"no_proxy":0,"value":1}	具体操作内容。

日志清理

自动化清理方法

1. 方式一：crontab 方式。
   a. 以 tdsql 用户登录依次登录 Manager/Scheduler 进程所在节点。
   b. 执行以下命令，查询系统已配置的自动化清理信息。

```
[root@VM-16-14-centos ~]# cat /etc/crontab |grep scheduler_log
30 4 * * * tdsql (find /data/scheduler_log -type f -mtime +7 |xargs rm -f &>/dev/null)
```

2. 方式二：修改配置文件。
   a. 以 tdsql 用户登录依次登录 Manager/Scheduler 进程所在节点。
   b. 修改配置文件
   vi /data/application/scheduler/conf/scheduler.xml

```
<scheduler>
    <Log>
     <log_info name="/data/scheduler_log/sys_scheduler" log_size="1000000000" log_level="0"
resv_days="7"/>
     <log_stat name="/data/scheduler_log/stat_scheduler" log_size="1000000000" log_level="0"
resv_days="7"/>
    </Log>
…
 <manager>
    <Log>
     <log_info name="/data/scheduler_log/sys_manager" log_size="1000000000" log_level="0"
resv_days="7"/>
     <log_stat name="/data/scheduler_log/stat_manager" log_size="1000000000" log_level="0"
resv_days="7"/>
    </Log>
…
```

参数	说明
name	日志存储路径。
log_size	单个日志文件大小，单位为 Byte。
log_level	日志级别。
resv_days	日志保留天数，单位为天。

3. 修改完成后，请重启进程。

```
cd /data/application/scheduler/bin
./restart.sh
```

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

8 MetaCluster 日志

日志含义

记录 MetaCluster 进程运行详细信息。

日志命名

运行日志：mc-host-*

日志默认存储路径

/data/application/TDSQL_MetaCluster/sbin/logs

日志示例

- mc-host-*日志示例

```
[2023/12/29 07:24:05.016 +08:00] [INFO] [console_api.go:77] ["Receive console service api request"] [method=GetBasicStats] [caller=local-web-client]
```

日志基础释义：

字段名	解释
[2023/12/29 07:24:05.016 +08:00]	时间。
[INFO]	日志级别。
[console_api.go:77]	源代码文件:行号。
["Receive console service api request"] [method=GetBasicStats] [caller=local-web-client]	日志详情。

日志清理

自动化清理方法

1. 使用 tdsql 用户依次登录各 MC 节点，修改配置文件。

vi /data/application/TDSQL_MetaCluster/conf/config-mc-host-1.toml

```
…
# Log file sub config
[log.file]
filename = "/data/application/TDSQL_MetaCluster/sbin/logs/mc-host-1.log"
# max log file size in MB
max-size = 512
# max log file retaining days, 0 is for never delete
max-days = 0
# maximum number of old log files to retain, 0 is for unlimited
max-backups = 0
…
```

2. 所有 MC 节点配置完成后，使用 tdsql 用户依次登录各 MC 节点，依次重启 MetaCluster。

```
cd /data/application/TDSQL_MetaCluster/sbin
./stop-meta-cluster.sh all
/bin/bash /data/tools/check_mc_alive.sh >/dev/null
```

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

9 OC_Agent 日志

9.1 OCAgent 日志 agent_main.log

日志含义

记录 OCAgent 进程的启动、停止、报错相关信息。

日志命名

OCAgent 日志：agent_main.log

日志默认存储路径

/data/oc_agent/log

日志示例

```
[2023-12-15 21:10:16] DEBUG: create process_thread succ.
```

日志基础释义：

字段名	解释
[2023-12-15 21:10:16]	日志时间。
DEBUG	日志级别。
create process_thread succ.	具体日志内容。

日志清理

自动化清理方法

1. 以 tdsql 依次登录所有节点，修改以下配置信息。
2. 修改配置文件/data/oc_agent/conf/oc_agent.xml。

```
<log log_file="../log/agent_main" log_module="agent_main" log_maxsize="50000000" log_num="2"/>
```

参数	说明
log_file	日志文件路径。
log_module	无需修改，表示配置 OCAgent 模块日志信息。
log_maxsize	单日志文件大小。单位 Byte。
log_num	保留日志文件个数。超过日志文件数，系统将定时删除。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

9.2 心跳上报日志 oc_sshd*

日志含义

记录 sshd 连接信息，通过 sshd 连接的 ip、port、执行的命令，以及命令返回的结果。

日志命名

心跳上报日志：oc_sshd*

日志默认存储路径

/data/oc_agent/log

日志示例

```
[2023-12-20 00:37:50 334299] INFO 384728,async_log.cpp:28:run,tid:0x7f7ed96af700,384732,acceptcb: Connection from 172.x.x.x port 60443 on 0.0.0.0 port 8966
```

日志基础释义：

字段名	解释
[2023-12-20 00:37:50 334299]	日志时间。
INFO	日志级别.
384728	线程 ID。
async_log.cpp	源码文件。
28	行号。
Run	运行的函数。
tid:0x7f7ed96af700	线程在内存中的映射地址。
acceptcb: Connection from 172.x.x.x port 60443 on 0.0.0.0 port 8966	具体打印的信息。

日志清理

自动化清理方法

不支持。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

9.3 crontab 监控模块日志

crontab_monitor.log

日志含义

定时任务监控日志，包括执行的定时任务内容，及其返回信息。

日志命名

crontab 监控模块日志: crontab_monitor.log

日志默认存储路径

/data/oc_agent/log

日志示例

```
[2023-04-25 00:04:42] DEBUG:
crontab_monitor_script::crontab_monitor.sh(sh ../scripts/crontab_monitor.sh ../conf/crontab_scripts.conf ../conf/nested_scripts.conf)
returns:10000&white_list_conf(/data/oc_agent/scripts/../conf//crontab_white_list.conf) is not exists.&
```

日志基础释义：

字段名	解释
[2023-04-25 00:04:42]	时间。
DEBUG	日志级别。
crontab_monitor_script::crontab_monitor.sh(sh ../scripts/crontab_monitor.sh ../conf/crontab_scripts.conf ../conf/nested_scripts.conf) returns:10000&white_list_conf(/data/oc_agent/scripts/../conf//crontab_white_list.conf) is not exists.&	具体日志内容。

日志清理

自动化清理方法

1. 以 tdsql 依次登录所有节点，修改以下配置信息。
2. 修改配置文件。

vi /data/oc_agent/conf/oc_agent.xml

```
…
<CrontabMonitor>
   <!-- 日志规则配置文件路径，日志处理脚本，自动化清理间隔时间,单位【秒】（从处理完到下次处理） -->
   <monitor cron_conf_path="../conf/" cron_monitor_script="../scripts/crontab_monitor.sh" cron_distance_time="60"/>
…
```

参数	说明
cron_conf_path	日志规则配置文件路径。
cron_monitor_script	日志处理脚本，无需修改。
cron_distance_time	自动化清理间隔时间，单位为秒。

3. 重启 oc_agent。

```
./data/oc_agent/bin/stop_agent.sh
./data/oc_agent/bin/start_agent.sh
```

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

9.4 进程监控日志 process_monitor*

日志含义

监测当前 TDSQL 服务器 Proxy、DB 进程的运行状态日志。

日志命名

进程监控日志：process_monitor*

日志默认存储路径

/data/oc_agent/log

日志示例

```
[2023-12-16 21:01:45] DEBUG: check_process_alive cmd:
sh ../scripts/process_monitor.sh ../conf/procconf_4936.conf |grep '&' result:0&&0&0
```

日志基础释义：

字段名	解释
[2023-12-16 21:01:45]	日志打印时间。
DEBUG	日志打印级别。
check_process_alive	具体执行函数。
cmd: sh ../scripts/process_monitor.sh ../conf/procconf_4936.conf |grep	具体执行的命令。
'&' result:0&&0&0	结果。

日志清理

自动化清理方法

1. 以 tdsql 依次登录所有节点，修改以下配置信息。
2. 修改配置文件。
   vi /data/oc_agent/scripts/process_monitor.sh

```
...
FOUR_AGO=`date +"%Y%m%d" -d"4 days ago"`
LOG_FOUR_AGO="${LOG_DIR}/process_${ProcessID}.${FOUR_AGO}"
[ -f ${LOG_FOUR_AGO} ] && rm ${LOG_FOUR_AGO}
...
```

参数	说明
4 days ago	日志保留天数。默认为 4 天，超过 4 天的日志，系统将自动删除。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

10 Agent 日志

10.1 系统日志 sys_report_[DB 端口号].log.[时间]

日志含义

记录 Agent 进程的运行日志。

日志命名

系统日志：sys_report_[DB 端口号].log.[时间]

日志默认存储路径

/data/tdsql_run/[DB 端口号]/mysqlagent/log/

日志示例

```
[2023-12-20 00:00:00 493950] DEBUG 18564,mysql/mysqlop.cpp:867:getReplStatus,tid:0x7fc09bfff700,18687,agent,run sql 'show slave status for channel  'dcnconnection' ' success,get empty info
```

日志基础释义：

字段名	解释
[2023-12-20 00:00:00 493950]	时间。
DEBUG	日志级别。
18564	线程 ID。
mysql/mysqlop.cpp	源码文件。
getReplStatus	函数。
tid:0x7fc09bfff700	线程在内存映射中地址。
Agent	agent 日志标识。
run sql 'show slave status for channel 'dcnconnection' ' success,get empty info	具体打印的日志内容。

日志清理

自动化清理方法

1. 以 tdsql 依次登录所有节点，修改以下配置信息。
2. 修改/data/tdsql_run/port/mysqlagent/conf 目录下的配置文件。

```
<report name="../log/sys_report_4012.log" log_size="536870912" log_level="0"/>
  <auto_rm prefix="../log/sys_report_4012.log" days="4" size="0" speed="60"/>
```

模块	参数	说明
日志相关配置	name	存储目录
	log_size	单个文件大小
	log_level	日志级别
自动清理相关配置	auto_rm prefix	清理日志路径
	days	保留天数
	size	大小
	speed	速度

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

10.2 binlog 备份结果日志

coldbackupbinlog_result_[DB 端口号]

日志含义

记录 Binlog 备份结果，每个实例 1 个备份结果，系统每 3 秒定时备份 Binlog，并实时更新备份结果。

日志命名

Binlog 备份结果日志：coldbackupbinlog_result_[DB 端口号]

日志默认存储路径

/data/tdsql_run/[DB 端口号]/mysqlagent/log/nohup

日志示例

```
0
```

- 0：表示备份成功。
- -1：表示备份失败。

日志清理

不支持。

10.3 备份结果日志

日志含义

记录备份结果，每个实例 1 个备份结果文件。

日志命名

- 物理备份到 COS 结果文件：coldbackuptocos_result_
- 逻辑备份到 COS 结果文件：mydumperbackuptocos_result_
- 物理备份到 Local 结果文件：coldbackuptolocal_result_
- 逻辑备份到 Local 结果文件：mydumper_backuptolocal_result_
- 物理备份到 HDFS 结果文件：coldbackup_result_
- 逻辑备份到 HDFS 结果文件：：mydumper_backup_result_

日志默认存储路径

/data/tdsql_run/*/mysqlagent/log/nohup/

日志示例

```
0
```

- 0：表示备份成功。
- -1：表示备份失败。

日志清理

不支持。

11 OSS 日志

日志含义

记录了 OSS 任务的流程，和 ZooKeeper 交互的过程，拉取的 ZooKeeper 信息等。

日志命名

系统日志： sys_tdsql_oss_log.*

日志默认存储路径

/data/application/oss/log

日志示例

```
[2023-12-20 00:00:01 248313] DEBUG 0,src/comm_function.cpp:511:get_node_all_children,tid:0x7fabf5ffb700,oss::the children name is require_answer@answer_0000000030
```

日志基础释义：

字段名	解释
[2023-12-20 00:00:01 248313]	日志时间。
DEBUG	日志级别。
src/comm_function.cpp	源码文件。
511	具体行号。
get_node_all_children	具体函数。
tid:0x7fabf5ffb700	线程 id 在内存中的映射地址。
oss::the children name is require_answer@answer_0000000030	具体内容。

日志清理

自动化清理方法

1. 以 tdsql 用户依次登录所有 OSS 进程所在节点。
2. 修改配置文件。

vi /data/application/oss/conf/scheduler.xml

```
<log_clean enable="1" max_save_days="30" max_size="100000000000" hz="12"/>
```

参数	说明
enable	自动化清理开关。 ● 1：启用。

	● 0：关闭。
max_save_days	日志最大保留天数。
max_size	单个日志文件最大字节数。
hz	系统间隔 12 小时，自动检查并清理日志。

3. 待所有节点配置文件修改完成后，重启 OSS 进程。

```
cd /data/application/OSS/boot
./stop.sh
./start.sh
```

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

12 Analysis 日志

日志含义

记录 Analysis 组件的工作情况。

日志命名

- 运行日志：main.log
- 归档日志：main.log-日期.gz

日志默认存储路径

/data/application/tdsql_analysis/logs

日志示例

```
[INFO] 2023-12-20 00:01:49.785 [MysqlConnectionPhantomRefCleaner]-start PhantomReference clean current Mysql ConnectionPhantomReference map size:1
```

日志基础释义：

字段名	解释
[INFO]	日志级别。
2023-12-20 00:01:49.785	日志时间。
[MysqlConnectionPhantomRefCleaner]	具体类。
start PhantomReference clean current Mysql ConnectionPhantomReference	操作的内容。

日志清理

自动化清理方法

1. 以 tdsql 用户依次登录所有 Analysis 进程所在节点。
2. 修改配置文件。

vi /data/application/tdsql_analysis/conf/logback.xml

```
…
<file>${user.dir}/logs/main.log</file>
<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <fileNamePattern>${user.dir}/logs/main.log-%d{yyyy-MM-dd}.gz</fileNamePattern>
        <MaxHistory>30</MaxHistory>
</rollingPolicy>
…
```

3. 待所有节点配置文件修改完成后，重启进程。

/data/application/tdsql_analysis/bin/restart.sh

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

13 Collector 日志

日志含义

Collector 是集群的监控信息上报组件，该组件会把 ZooKeeper 上所有指定路径下的监控指标上报到 TDSQL 监控库中。Collector 日志用于记录 Collector 组件的工作情况。

日志命名

运行日志：main.log

日志默认存储路径

/data/application/tdsql_collector/logs

日志示例

```
[INFO] 2023-12-20 00:01:49.785 [MysqlConnectionPhantomRefCleaner]-start PhantomReference clean current Mysql ConnectionPhantomReference map size:1
```

日志基础释义：

字段名	解释
[INFO]	日志级别。
2023-12-20 00:01:49.785	日志时间。
[MysqlConnectionPhantomRefCleaner]	具体类。
start PhantomReference clean current Mysql ConnectionPhantomReference	操作的内容。

日志清理

自动化清理方法

1. 以 tdsql 用户依次登录所有 Collector 进程所在节点。
2. 修改配置文件。

vi /data/application/tdsql_collector/conf/logback.xml

```
…
   <file>logs/main.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
          <fileNamePattern>logs/main.log-%d{yyyy-MM-dd}.gz</fileNamePattern>
          <MaxHistory>30</MaxHistory>
    </rollingPolicy>
…
```

3. 重启进程。

/data/application/tdsql_collector/bin/restart.sh

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

14 OnlineDDL 日志

日志含义

记录 OnlineDDL 进程运行详细信息。

日志命名

运行任务日志：sys_ddlperformer_mgn.log*

日志默认存储路径

/data/application/onlineddl/log

日志示例

```
[2023-12-18 00:00:18 757040] DEBUG
392859,ddlmng.cpp:36:dispatchJob,tid:0x7fdc19f7a700,392864,alldispathcjobs num is 0
```

日志基础释义：

字段名	解释
[2023-12-18 00:00:18 757040]	日志时间。
DEBUG	日志级别。
392859	线程 ID。
ddlmng.cpp	源码文件。
36	行号。
dispatchJob	函数。
tid:0x7fdc19f7a700	线程 id 在内存中的映射地址。
alldispathcjobs num is 0	具体的打印日志内容。

日志清理

自动化清理方法

1. 以 tdsql 用户依次登录所有 OnlineDDL 进程所在节点。
2. 修改配置文件。

vi /data/application/onlineddl/conf/ddlperformer.xml

```
<log log_level="0" log_dir="../log">
    <!-- 日志过滤功能，open 是否开启， reserve_days 表示日志保留时间，默认 90 天
        由于 ddl 日志很小且任务少，这里只按天清理，不考虑日志大小） -->
    <log_cleaner open="1" reserve_days="90"/>
</log>
```

3. 重启进程。

/data/application/onlineddl/bin/restart.sh

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

15 CloudDBA 日志

日志含义

记录 CloudDBA 进程运行详细信息。

日志命名

诊断日志：diagnosis*

日志默认存储路径

/data/application/clouddba/log

日志示例

```
[2023-12-20 00:00:12 119409] DEBUG 390066,/data/landun/workspace/clouddba/diagnosis/clean_log.h:118:CleanBySize,tid:0x7f74b3fcc700,390157,size:71344128 is ok,no need to clean
```

日志基础释义：

字段名	解释
[2023-12-20 00:00:12 119409]	日志时间。
DEBUG	日志级别。
390066	线程 ID。
/data/landun/workspace/clouddba/diagnosis/clean_log.h	源码文件。
118	行号。
CleanBySize	具体的函数。
tid:0x7f74b3fcc700	线程 id 在内存中的映射地址。
size:71344128 is ok,no need to clean	具体日志内容。

日志清理

自动化清理方法

1. 修改 CloudDBA 所在节点配置文件。

vi /data/application/clouddba/conf/diagnosis.conf

```
…
"SysLog":
   {
      "file" : "../log/diagnosis",
      "level" : 0,
      "size" : 1000000000,
      "clean_time": 14,
      "clean_size": 2000000000
   },
…
```

2. 重启进程时，会进行慢删除，将日志文件截断。

```
cd /data/application/clouddba/bin/
./restart.sh ../conf/diagnosis.conf
```

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

16 赤兔日志

日志含义

记录客户端访问赤兔的详细信息。

日志命名

- 赤兔的请求日志：access-[日期].log
- 赤兔当天请求日志：www.access.log
- 赤兔慢查询日志：www.log.slow

日志默认存储路径

/data/website/logs

日志示例

- 请求日志、当天请求日志示例：

```
[2023-12-15T21:40:17+08:00] 10.101.245.250 "GET /tdsqlpcloud/index.php/instance/conf?mid=/tdsqlzk/set_1702375499_127&pmid=&cluster_type=noshard&cluster=tdsql_s187f88t1 HTTP/1.1" 302 5 "http://82.157.36.211/tdsqlpcloud/index.php/instance/manage/detail?mid=/tdsqlzk/set_1702375499_127&cluster=tdsql_s187f88t1" "-"
```

日志基础释义：

字段名	解释
[2023-12-15T21:40:17+08:00]	时间。
10.101.245.250	客户端 IP。
"GET /tdsqlpcloud/index.php/instance/conf?mid=/tdsqlzk/set_1702375499_127&pmid=&cluster_type=noshard&cluster=tdsql_s187f88t1 HTTP/1.1" 302 5 "http://82.157.36.211/tdsqlpcloud/index.php/instance/manage/detail?mid=/tdsqlzk/set_1702375499_127&cluster=tdsql_s187f88t1" "-"	客户请求接口内容。

- 慢查询日志示例：

```
[02-Aug-2023 19:32:28]
[pool www] pid 178314
script filename = /data/website/tdsqlpcloud/index.php
0x00007f1cb0017720]
curl exec() /data/website/tdsqlpcloud/www/core/0SS Model.php:131
0x00007f1cb00163d0
api cal1() /data/website/tdsqlpcloud/www/models/oss/gw/0ss noshard gw mod.php:68
0x00007f1cb0015ec0
get_noshard result() /data/website/tdsqlpcloud/www/models/0ss2db mod.php:36
0x00007f1cb0014e50
import_instance() /data/website/tdsqlpcloud/www/contrbllers/instance/Manage.php:418
0x00007f1cb00149b0
get) /data/website/tdsqlpcloud/framework/core/Codelgniter.php:524
0x00007f1cb0013870
[INCLUDE OR EVAL]() /data/website/tdsqlpcloud/index.php:316
```

字段名	解释
[02-Aug-2023 19:32:28]	时间。
[pool www] pid 178314	客户端连接 pid。
script filename = /data/website/tdsqlpcloud/index.php 0x00007f1cb0017720] curl exec() /data/website/tdsqlpcloud/www/core/0SS Model.php:131 0x00007f1cb00163d0 api cal1() /data/website/tdsqlpcloud/www/models/oss/gw/0ss noshard gw mod.php:68 0x00007f1cb0015ec0 get_noshard result() /data/website/tdsqlpcloud/www/models/0ss2db mod.php:36 0x00007f1cb0014e50 import_instance() /data/website/tdsqlpcloud/www/contrbllers/instance/Manage.php:418 0x00007f1cb00149b0 get) /data/website/tdsqlpcloud/framework/core/Codelgniter.php:524 0x00007f1cb0013870 [INCLUDE OR EVAL]() /data/website/tdsqlpcloud/index.php:316	访问的堆栈。

日志清理

自动化清理方法

不支持。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

17 HDFS 日志

日志含义

HDFS 用于存储备份文件，用户可通过日志查询 HDFS 进程的详细信息。

日志命名

- 记录 datanode 进程报错和相关信息：hadoop-tdsql-datanode-[部署节点].log
- 记录当前 datanode 进程启动用户环境的 limit 值：hadoop-tdsql-datanode-[部署节点].out
- 记录 journalnode 进程报错和相关信息：hadoop-tdsql-journalnode-[部署节点].log
- 记录当前 journalnode 进程启动用户环境的 limit 值：hadoop-tdsql-journalnode-[部署节点].out
- 记录 namenode 进程报错和相关信息：hadoop-tdsql-namenode-[部署节点].log
- 记录当前 namenode 进程启动用户环境的 limit 值：hadoop-tdsql-namenode-[部署节点].out
- 记录 zkfc 进程报错和相关信息：hadoop-tdsql-zkfc-[部署节点].log
- 记录当前 zkfc 进程启动用户环境的 limit 值：hadoop-tdsql-zkfc-[部署节点].out

日志默认存储路径

/data/home/tdsql/hadoop/logs

日志示例

- hadoop-tdsql-datanode-[部署节点].log、hadoop-tdsql-journalnode-[部署节点].log、hadoop-tdsql-namenode-[部署节点].log、hadoop-tdsql-zkfc-[部署节点].log 日志输出类似，本文以 hadoop-tdsql-datanode-[部署节点].log 为例说明。

```
2023-12-16 15:50:48,887 INFO
org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to add all replicas to map for block pool BP-1749774794-127.0.0.1-1702712968852: 2ms
```

日志基础释义：

字段名	解释
2023-12-16 15:50:48,887	时间。
INFO	日志级别。
org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl	HDFS 日志相关类。
Total time to add all replicas to map for block pool BP-1749774794-127.0.0.1-1702712968852: 2ms	具体日志内容。

- hadoop-tdsql-datanode-[部署节点].out、hadoop-tdsql-journalnode-[部署节点].out 、hadoop-tdsql-namenode-[部署节点].out、hadoop-tdsql-zkfc-[部署节点].out 日志输出类似，本文以 hadoop-tdsql-datanode-[部署节点].out 为例说明。

```
core file size          (blocks, -c) unlimited
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 123513
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 600000
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 60000
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

字段名	解释
core file size	Core 文件大小限制。
data seg size	数据段大小限制，表示进程数据段的最大大小。
scheduling priority	调度优先级，表示进程的调度优先级。
file size	文件大小限制，表示单个文件的最大大小。
pending signals	待处理信号数量限制，表示进程可以排队等待处理的信号数量。
max locked memory	最大锁定内存大小限制，表示进程可以锁定的最大内存大小。
max memory size	最大内存大小限制，表示进程可以使用的最大内存大小。
open files	打开文件数量限制，表示进程可以同时打开的文件数量。
pipe size	管道大小限制，表示管道缓冲区的最大大小。
POSIX message queues	POSIX 消息队列大小限制，表示进程可以使用的最大消息队列大小。
real-time priority	实时优先级，表示进程的实时调度优先级。
stack size	栈大小限制，表示进程栈的最大大小。
cpu time	CPU 时间限制，表示进程可以使用的最大 CPU 时间。
max user processes	最大用户进程数量限制，表示用户可以创建的最大进程数量。
virtual memory	虚拟内存大小限制，表示进程可以使用的最大虚拟内存大

	小。
file locks	文件锁数量限制，表示进程可以同时持有的文件锁数量。

日志清理

自动化清理方法

不支持。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

18 多源同步组件日志

18.1 Kafka

日志含义

多源同步组件用于异构数据迁移，用户可通过日志查询多源同步组件进程的详细信息。

日志命名

- **操作日志：**
  - kafka-request.log.*（其中*代表日期和时间）：捕获与特定 HTTP 请求或类似请求相关的日志数据。通常用于调试、监控或跟踪应用程序中特定请求的性能、错误或其他重要事件。
  - server.log.* （其中*代表日期和时间）：记录 Kafka broker 的运行时信息。这包括了启动、关闭、连接、错误、警告以及其他关键的运行时事件。
  - controller.log.* （其中*代表日期和时间）：用于记录与 Kafka 集群的控制器节点相关的事件。控制器是 Kafka 集群中的一个关键组件，负责管理分区和副本的状态，以及在发生故障时进行领导者选举。
  - state-change.log.* （其中*代表日期和时间）：用于记录和追踪 Kafka 状态变化相关的日志信息。
  - kafkaServer-gc.log：记录垃圾收集（Garbage Collection, GC）的日志。垃圾收集是 Java 虚拟机（JVM）的一个过程，它会自动管理应用程序分配的内存。当对象不再被使用时，GC 进程会回收这些对象占用的内存，以便内存可以被重新利用。
- **Topic 数据：**后缀名为.log 和.index 的文件。

日志默认存储路径

- **操作日志：**/data/application/kafka/logs
- **Topic 数据：**/data/application/kafka/config/server.properties 配置文件 log.dirs 指定的路径。

日志示例

- kafka-request.log 日志示例

```
[2023-03-15 10:00:00,000] INFO [RequestSendThread controllerId=1] Controller 1 epoch 12 sends request {api_key=18,v=3,correlation_id=123,client_id=producer-1,acks=1,timeout=30000,partition_data=[{topic=test,partition_data=[{partition=0,record_set_size=1048576}]}]} to broker kafka1:9092 (id: 2 rack: null) (kafka.controller.RequestSendThread)
```

```
[2023-03-15 10:05:00,000] INFO [SocketServer brokerId=1] Received produce request from client producer-1 with correlation id 123 for partition test-0 (kafka.network.Processor)
[2023-03-15 10:10:00,000] INFO [SocketServer brokerId=1] Received fetch request from client consumer-1 with correlation id 234 for partition test-0 offset 1048576 (kafka.network.Processor)
```

在这个示例中，你可以看到：

- 控制器发送了一个请求，这个请求是由生产者客户端 producer-1 发起的，请求将消息发送到主题 test 的分区 0。
- Kafka broker 接收到了来自 producer-1 的生产请求，请求发送消息到 test-0 分区。
- Kafka broker 接收到了来自 consumer-1 的拉取请求

- kafkaServer-gc.log 日志示例

```
2023-03-15T10:00:00.000+0000: 1234.567: [GC (Allocation Failure) [PSYoungGen: 512K->64K(1024K)] 1024K->576K(2048K), 0.0017654 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
2023-03-15T10:05:00.000+0000: 1239.567: [Full GC (Ergonomics) [PSYoungGen: 0K->0K(1024K)] [ParOldGen: 512K->482K(1024K)] 512K->482K(2048K), [Metaspace: 485K->485K(1056768K)], 0.0056789 secs] [Times: user=0.01 sys=0.00, real=0.01 secs]
```

在这个示例中，你可以看到两个 GC 事件：

1. 一个年轻代（Young Generation）的 Minor GC，由于分配失败（Allocation Failure）触发。GC 前年轻代使用了 512K，GC 后减少到 64K，整个堆从 1024K 减少到 576K。这个 GC 事件的暂停时间非常短，只有 0.0017654 秒。
2. 一个 Full GC，由于 JVM 的自动调优（Ergonomics）触发。在这次 GC 中，年轻代的内存没有变化，老年代（Old Generation）从 512K 减少到 482K。整个堆的使用从 512K 减少到 482K。这次 GC 的暂停时间是 0.0056789 秒。

- controller.log 日志示例

```
[2023-03-15 10:00:00,000] INFO [Controller id=1] Completed loading metadata from ZK (kafka.controller.KafkaController)
[2023-03-15 10:05:00,000] INFO [Controller id=1] Broker 2 started with log4j (kafka.controller.ControllerEventManager$ControllerEventThread)
[2023-03-15 10:10:00,000] INFO [Controller id=1] New leader and ISR for partition topic1-0 is {"leader":3,"leader_epoch":42,"isr":[3,1,2]} (state.change.logger)
```

在这个示例中，你可以看到控制器（ID 为 1）正在记录来自 ZooKeeper 的元数据加载、broker 的启动，以及分区 topic1-0 的新领导者和 ISR（In-Sync Replicas）信息。

- server.log 日志示例

```
[2023-03-15 10:00:00,000] INFO Kafka version: 2.8.0 (org.apache.kafka.common.utils.AppInfoParser)
[2023-03-15 10:00:00,000] INFO Starting Kafka server with broker.id=1 (kafka.server.KafkaServer)
[2023-03-15 10:00:05,000] INFO [Kafka Server 1], started (kafka.server.KafkaServer)
[2023-03-15 10:00:10,000] WARN [ReplicaFetcher replicaId=1, leaderId=0, fetcherId=0] Error when processing fetch request for partition test-0 with offset 1048576 (kafka.server.ReplicaFetcherThread)
```

```
[2023-03-15 10:00:15,000] ERROR [Controller id=1 epoch=4] Controller moved to broker 1 (state.change.logger)
```

在这个示例中，你可以看到：

- Kafka 版本信息和 broker 启动信息。
- Kafka 服务启动完成的确认信息。
- 一个警告信息，表明在处理名为 test-0 的分区的抓取请求时遇到了问题。
- 一个错误信息，表明控制器已经迁移到了 broker 1。

- state-change.log 日志示例

```
[2023-03-15 10:00:00,000] INFO [Controller id=1] Elected as the controller, brokerId=1 (kafka.controller.KafkaController)
[2023-03-15 10:05:00,000] INFO [Controller id=1] Changed partition test-0 from ReplicaAssignment(replicas=1,2,3) to ReplicaAssignment(replicas=2,3,1) (kafka.controller.KafkaController)
[2023-03-15 10:10:00,000] INFO [Controller id=1] Broker 2 is the new leader for partition test-0 (kafka.controller.KafkaController)
[2023-03-15 10:15:00,000] INFO [Controller id=1] Removed replica 3 from ISR for partition test-0 due to it being offline (kafka.controller.KafkaController)
```

在这个示例中，你可以看到：

- 当前 broker（brokerId=1）被选举为控制器。
- 控制器更改了主题 test 分区 0 的副本分配。
- 控制器宣布 broker 2 成为主题 test 分区 0 的新领导者。
- 控制器从分区 test-0 的 ISR 集合中移除了副本 3，因为它离线了。

日志清理

自动化清理方法

- **基于时间的日志自动化清理策略：**
  a. 以 tdsql 用户登录依次登录多源同步进程所在节点。
  b. 配置文件。
  vi/data/application/kafka/config/server.properties

```
…
log.retention.hours=168  #日志保留时间，默认 7 天
log.retention.check.interval.ms=300000 #每 5 分钟检查一次是否有 segment 超过了 7 天
…
```

- **基于文件大小的日志自动化清理策略：**
  a. 以 tdsql 用户登录依次登录多源同步进程所在节点。
  b. 配置文件。
  vi/data/application/kafka/config/server.properties

```
log.retention.bytes=214748364800  #200GB
```

说明：

系统支持同时指定 log.retention.bytes 和 log.retention.hours 来混合指定保留规则。
当日志的大小超过了 log.retention.bytes 就清除老的 segment；当某个 segment 的保留时间超过了规定的值同样将其清除。

手动清理方法

- 删除方法：针对/data/application/kafka/logs 路径下的日志，按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

18.2 binlogconsumer

日志含义

多源同步组件用于异构数据迁移，用户可通过日志查询多源同步组件进程的详细信息。

日志命名

- 消费者 sql 回放日志：sys_binlogconsumer.*
- consumer 守护进程管理日志：sys_binlogconsumer_mgn.log.*
- 全量数据导出导入操作日志: sys_oncesynctable.log.*

日志默认存储路径

/data/application/consumer/log/

日志示例

```
[2024-01-05 00:01:41 849646] INFO
31886,supervisethread.cpp:93:dealResetJobNode,tid:0x7f9828dff700,32033,get
/tdsqlzk/agent/syncjobs/setsyncjobresetjob@0000000283, -101: no node
```

字段名	解释
[2024-01-08 15:03:29,311]	时间。
INFO	日志级别。
supervisethread.cpp:93:dealResetJobNode	源代码文件:行数:函数名
tid:0x7f9828dff700	线程号。
get /tdsqlzk/agent/syncjobs/setsyncjobresetjob@0000000283, -101: no node	日志详情。

日志清理

自动化清理方法

- 归档日志文件：系统默认将/data/application/consumer/log/下的超过 10 分钟的日志文件压缩并归档至/data/consumer_log_bak 下。
- 归档数据文件：系统默认将/data/application/consumer/data 下的超过 10 分钟的数据文件压缩并归档至/data/consumer_log_bak 下。
- 清理备份目录：系统自动清理/data/consumer_log_bak 下超过 20 天的日志及数据文件。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

18.3 binlogproducter

日志含义

多源同步组件用于异构数据迁移，用户可通过日志查询多源同步组件进程的详细信息。

日志命名

sys_binlogtokafka_$dbport.log.*

日志默认存储路径

/data/tdsql_run/$dbport/mysqlagent/log

日志示例

```
[2024-01-09 00:00:04 694275] INFO 29659,binlogdealthread.cpp:932:outputNewEventComm,tid:0x7f9f737fc700,29669,row binlog queue is full, loop and wait, sleep 100ms.
```

字段名	解释
[2024-01-08 15:03:29,311]	时间
INFO	日志级别
binlogdealthread.cpp:932:outputNewEventComm	源代码文件:行数:函数名
tid:0x7f9f737fc700	线程号
row binlog queue is full, loop and wait, sleep 100ms.	日志详情

日志清理

自动化清理方法

系统默自动清理超过 4 小时的日志文件。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

18.4 oncesynctable

日志含义

多源同步组件用于异构数据迁移，用户可通过日志查询多源同步组件进程的详细信息。

日志命名

sys_oncesynctable.log

日志默认存储路径

/data/application/consumer/log/

日志示例

```
[2024-01-08 00:1:29 840046] INFO 33826,autooncesynctable.cpp:2906:checkMysqlTableNeedAllDump,tid:0x7f9238dff700,32133,check need all dump trans rule failed
```

字段名	解释
[2024-01-08 00:1:29 840046]	时间。
INFO	日志级别。
autooncesynctable.cpp:2906	源代码文件:行数:函数名
tid:0x7f9238dff700	线程号
check need all dump trans rule failed	日志内容。

日志清理

自动化清理方法

- 归档日志文件：系统默认将/data/application/consumer/log/下的超过 10 分钟的日志文件压缩并归档至/data/consumer_log_bak 下。
- 归档数据文件：系统默认将/data/application/consumer/data 下的超过 10 分钟的数据文件压缩并归档至/data/consumer_log_bak 下。
- 清理备份目录：系统自动清理/data/consumer_log_bak 下超过 20 天的日志及数据文件。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。

18.5 binlogconsumermgn

日志含义

多源同步组件用于异构数据迁移，用户可通过日志查询多源同步组件进程的详细信息。

日志命名

sys_binlogconsumer_mgn.log.*

日志默认存储路径

/data/application/consumer/log/

日志示例

```
[2022-11-20 02:33:14 797694] INFO 57619,mgn.cpp:103:checkMgnInfo,tid:0x7f22d4ff9700,57654,mgnNode:consumermgnalive@10.134.89.96,weightSum:5
```

字段名	解释
[2024-01-08 15:03:29,311]	时间。
INFO	日志级别。
mgn.cpp:103:checkMgnInfo	源代码文件:行数:函数名
tid:0x7f22d4ff9700	线程号
mgnNode:consumermgnalive@10.134.89.96,weightSum:5	日志内容。

日志清理

自动化清理方法

- 归档日志文件：系统默认将/data/application/consumer/log/下的超过 10 分钟的日志文件压缩并归档至/data/consumer_log_bak 下。
- 归档数据文件：系统默认将/data/application/consumer/data 下的超过 10 分钟的数据文件压缩并归档至/data/consumer_log_bak 下。
- 清理备份目录：系统自动清理/data/consumer_log_bak 下超过 20 天的日志及数据文件。

手动清理方法

- 删除方法：按时间顺序排序，从最早的日志删起。
- 影响：无法定位删除的日志对应日期的问题。