Table of Contents

Introduction 1.1
0.概述 1.2
0.1 dble 简介与整体架构 1.2.1
0.2 dble对MyCat做的增强 1.2.2
0.3 快速开始 1.2.3
0.4 docker镜像快速开始 1.2.4
0.5 docker-compose快速开始 1.2.5
0.6 数据拆分简介 1.2.6
0.7 docker-compose快速开始dble高可用 1.2.7
1.配置文件 1.3
1.1 cluster.cnf 1.3.1
1.2 bootstrap.cnf 1.3.2
1.3 user.xml 1.3.3
1.4 db.xml 1.3.4
1.5 sharding.xml 1.3.5
1.6 log4j2.xml 1.3.6
1.7 全局序列配置 1.3.7
1.7.1 MySQL offset-step方式 1.3.7.1
1.7.2 时间戳方式(类Snowflake) 1.3.7.2
1.7.3 分布式时间戳方式(类Snowflake) 1.3.7.3
1.7.4 分布式offset-step方式 1.3.7.4
1.8 cache配置 1.3.8
1.8.1 cache配置 1.3.8.1
1.8.2 ehcache配置 1.3.8.2
1.9 自定义拆分算法 1.3.9
1.10 配置文件变更记录 1.3.10
1.11 自定义告警 1.3.11
1.12 自定义全局表一致性检查 1.3.12
2.功能描述 1.4
2.0 管理端元数据库 1.4.1
2.1 管理端命令 1.4.2
2.1.1 select命令 1.4.2.1
2.1.2 set命令 1.4.2.2
2.1.3 show命令 1.4.2.3
2.1.4 switch命令 1.4.2.4
2.1.5 kill命令 1.4.2.5
2.1.6 stop命令 1.4.2.6
2.1.7 reload命令 1.4.2.7
2.1.8 rollback命令 1.4.2.8
2.1.9 offline命令 1.4.2.9
2.1.10 online命令 1.4.2.10
2.1.11 file命令 1.4.2.11
2.1.12 log命令 1.4.2.12
2.1.13 配置检查命令 1.4.2.13
2.1.14 pause & resume 命令 1.4.2.14
2.1.15 慢查询日志相关命令 1.4.2.15
2.1.16 创建/删除物理库命令 1.4.2.16
2.1.17 check @@metadata命令 1.4.2.17
2.1.18 release @@metadata命令 1.4.2.18
2.1.19 split命令 1.4.2.19
2.1.20 flow_control 命令 1.4.2.20
2.1.21 刷新连接池命令 1.4.2.21
2.2 全局序列 1.4.3
2.2.1 MySQL offset-step方式 1.4.3.1
2.2.2 时间戳方式 1.4.3.2
2.2.3 分布式时间戳方式 1.4.3.3
2.2.4 分布式offset-step方式 1.4.3.4
2.3 读写分离 1.4.4
2.4 注解 1.4.5

2.5 分布式事务 1.4.6
2.5.1 XA事务概述 1.4.6.1
2.5.2 XA事务的提交以及回滚 1.4.6.2
2.5.3 XA事务的后续补偿以及日志清理 1.4.6.3
2.5.4 XA事务的记录 1.4.6.4
2.5.5 一般分布式事务概述 1.4.6.5
2.6 连接池管理 1.4.7
2.7 内存管理 1.4.8
2.8 集群同步协调&状态管理 1.4.9
2.9 grpc 告警 1.4.10
2.10 表meta数据管理 1.4.11
2.10.1 Meta信息初始化 1.4.11.1
2.10.2 Meta信息维护 1.4.11.2
2.10.3 一致性检测 1.4.11.3
2.10.4 View Meta 1.4.11.4
2.11 统计管理 1.4.12
2.11.1 查询条件统计 1.4.12.1
2.11.2 表状态统计 1.4.12.2
2.11.3 用户状态统计 1.4.12.3
2.11.4 命令统计 1.4.12.4
2.11.5 heartbeat统计 1.4.12.5
2.11.6 网络读写统计 1.4.12.6
2.12 故障切换 1.4.13
2.13 前后端连接检查 1.4.14
2.14 ER表 1.4.15
2.15 global表 1.4.16
2.16 缓存的使用 1.4.17
2.17 执行计划 1.4.18
2.18 性能观测和调整 1.4.19
2.19 智能计算reload 1.4.20
2.20 慢查询日志 1.4.21
2.21 单条SQL性能trace 1.4.22
2.22 KILL @@DDL_LOCK 1.4.23
2.23 外部高可用联动 1.4.24
2.23.1 外部后端MYSQL-HA连接 1.4.24.1
2.23.2 命令的使用说明 1.4.24.2
2.23.3 命令的实现细节 1.4.24.3
2.23.4 简单的HA交互使用案例 1.4.24.4
2.24 超时控制 1.4.25
2.25 流量控制 1.4.26
2.26 client_found_rows权能标志 1.4.27
3.语法兼容 1.5
3.1 DDL 1.5.1
3.1.1 DDL&Table Syntax 1.5.1.1
3.1.2 DDL&View Syntax 1.5.1.2
3.1.3 DDL&Index Syntax 1.5.1.3
3.1.4 DDL透传 1.5.1.4
3.1.5 DDL&Database_Syntax 1.5.1.5
3.1.6 ONLINE DDL 1.5.1.6
3.2 DML 1.5.2
3.2.1 INSERT 1.5.2.1
3.2.2 REPLACE 1.5.2.2
3.2.3 DELETE 1.5.2.3
3.2.4 UPDATE 1.5.2.4
3.2.5 SELECT 1.5.2.5
3.2.6 SELECT JOIN syntax 1.5.2.6
3.2.7 SELECT UNION Syntax 1.5.2.7
3.2.8 SELECT Subquery Syntax 1.5.2.8
3.2.9 LOAD DATA 1.5.2.9
3.2.10 不支持的语句 1.5.2.10
3.3 Prepared SQL Syntax 1.5.3
3.4 Transactional, Savepoint and Locking Statements 1.5.4

3.4.1 Lock&unlock 1.5.4.1
3.4.2 XA 事务语法 1.5.4.2
3.4.3 一般事务语法 1.5.4.3
3.4.4 SET TRANSACTION Syntax 1.5.4.4
3.4.5 SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax 1.5.4.5
3.5 DAL 1.5.5
3.5.1 SET 1.5.5.1
3.5.2 SHOW 1.5.5.2
3.5.3 KILL 1.5.5.3
3.6 存储过程支持方式 1.5.6
3.7 Utility Statements 1.5.7
3.8 Hint 1.5.8
3.9 其他不支持语句 1.5.9
3.10 函数与操作符支持列表(alpha版本) 1.5.10
3.11 导入导出方式 1.5.11
4.协议兼容 1.6
4.1 基本包 1.6.1
4.2 连接建立 1.6.2
4.3 文本协议 1.6.3
4.4 二进制协议 (Prepared Statements) 1.6.4
4.5 服务响应包 1.6.5
5.已知限制 1.7
5.1 druid引发的限制 1.7.1
5.2 其他已知限制 1.7.2
6.与MySQL Server的差异化描述 1.8
6.1 事务中遇到主键冲突需要显式回滚 1.8.1
6.2 INSERT不能显式指定自增序列 1.8.2
6.3 增加"show all tables" 1.8.3
6.4 去除了增删改的message信息 1.8.4
6.5 information_schema等库的支持 1.8.5
7.开发者须知 1.9
7.1 SQL开发编写原则 1.9.1
7.2 dble连接Demo 1.9.2
7.3 其他注意事项 1.9.3
8.配置示例 1.10
8.1 时间戳方式全局序列的配置 1.10.1
8.2 MySQL-offset-step 方式全局序列的配置 1.10.2
9.sysbench压测dble示例 1.11
9.1 测试环境及架构 1.11.1
9.2 修改dble配置 1.11.2
9.3 使用sysbench进行压测 1.11.3
A.Faq 1.12
A.1 ErrorCode 1.12.1
max Connections 1.12.1.1
Out Of Memory Error 1.12.1.2
The Problem Of Hint 1.12.1.3
NestLoop Parameters Lead To Temptable Exception 1.12.1.4
Can't Get Variables From ShardingNode 1.12.1.5
Port already in use:1984 1.12.1.6
Sharding Column Cannot Be Null 1.12.1.7
A.2 原理解释 1.12.2
How To Use Explain To Resolve The Distribution Rules Of Group Gy 1.12.2.1
Hash And ConsistentHashing And Jumpstringhash 1.12.2.2
A.3 使用说明 1.12.3
ToBeContinued2 1.12.3.1

dble 中文技术参考手册

目录

参考 gitbook 左侧目录区 或 SUMMARY.md

PDF下载

PDF下载

中文公开课

dble中文公开课

官方技术支持

- 代码库 github: github.com/actiontech/dble
- 自动化测试库 github: github.com/actiontech/dble-test-suite
- 文档库 github: github.com/actiontech/dble-docs-cn
- 文档库 github pages: actiontech.github.io/dble-docs-cn
- 网站：DBLE官方网站
- QQ group: 669663113
- 开源社区微信公众号

注意

本分支上的手册适用于3.20.10.0版dble，其他版本的文档请参考对应tag分支或者release版文档。

提示

如果您使用了dble，请告诉我们。

联系我们

如果想获得dble 的商业支持, 您可以联系我们:

- 全国支持: 400-820-6580
- 华北地区: 86-13718877200, 王先生
- 华南地区: 86-18503063188, 曹先生
- 华东地区: 86-18930110869, 梁先生
- 西南地区: 86-13540040119, 洪先生

0 概览

- 0.1 dble 简介与整体架构
- 0.2 dble对MyCat做的增强
- 0.3 快速开始
- 0.4 docker镜像快速开始
- 0.5 docker-compose快速开始
- 0.6 数据拆分简介
- 0.7 docker-compose快速开始dble高可用

0.1 dble 简介与整体架构

0.1.1 dble简介

dble是上海爱可生信息技术股份有限公司基于mysql的高可扩展性的分布式中间件，存在以下几个优势特性：

- 数据水平拆分 随着业务的发展，您可以使用dble来替换原始的单个MySQL实例。
- 兼容**Mysql** 与MySQL协议兼容，在大多数情况下，您可以用它替换MySQL来为你的应用程序提供新的存储，而无需更改任何代码。
- 高可用性 dble服务器可以用作集群，业务不会受到单节点故障的影响。
- **SQL**支持 支持SQL 92标准和MySQL方言。我们支持复杂的SQL查询，如group by，order by，distinct，join，union，sub-query等等。
- 复杂查询优化 优化复杂查询，包括但不限于全局表连接分片表，ER关系表，子查询，简化选择项等。
- 分布式事务支持 使用两阶段提交的分布式事务。您可以为了性能选择普通模式或者为了数据安全采用XA模式。当然，XA模式依赖于MySQL-5.7的XA Transaction，MySQL节点的高可用性和数据的可靠性。

0.1.2 dble由来

- dble 是基于开源项目MyCat的，在此对于MyCat的贡献者们致以由衷的感谢。
- 对我们来说，专注于MySQL是一个更好的选择。 所以我们取消了对其他数据库的支持，对兼容性，复杂查询和分布式事务的行为进行了深入的改进/优化。 当然，还修复了一些bugs。详情可见dble对MyCat做的改进

0.1.3 dble内部架构

0.2 dble对MyCat做的增强

建议大家收看免费课程来初步认识dble的改进项

0.2.1 缺陷修复（目前mycat社区不活跃，不再追踪mycat的bug）

- 由于对堆外内存的使用不当，导致高并发操作时对同一片内存可能发生“double free"，从而造成JVM异常，服务崩溃。 #4
- XA事务漏洞：包乱序导致客户端崩溃 #21
- where关键字写错时，会忽视后面的where条件，会得到错误的结果，比如select * from customer wher id=1;#126
- 对于一些隐式分布式事务，例如insert into table values(节点1)，(节点2)；原生mycat直接下发，这样当某个节点错误时，会造成该SQL执行了一部分
- 权限黑名单针对同一条sql只在第一次生效。#92
- 聚合/排序的支持度非常有限，而且在很多场景下还存在结果不正确、执行异常等问题 #43,#31,#44
- 针对between A and B语法，hash拆分算法计算出来的范围有误#23
- 开启全局表一致性检查时，对全局表的处理存在诸多问题，例如不能alter table、insert...on duplicate...时不更新时间戳、update...in ()报错等#24, #25,#26 ,#5
- 多值插入时，全局序列生成重复值 #1
- ER表在一个事务内被隔离,不能正确插入子表数据#13
- sharding-join结果集不正确#17
  详情及其他修正请见 修复列表

0.2.2 实现改进

- 对某些标准SQL语法支持不够好的方面作了改进，例如对create table if not exists...、alter table add/drop [primary] key...等语法的支持
- 对整体内部IO结构进行了大幅的改造和调优:参见dble的IO结构 或者见本节结尾附录2
- 禁止普通用户连接管理端口进行管理操作，增强安全性 #56
- 对全局序列做了如下改进
  - 删去无工程意义的本地文件方式
  - 改进数据库方式、ZK方式，使获取的序列号更加准确
  - 改进时间戳方式和ZK ID生成器方式，消除并发低时的数据分布倾斜问题
  - 修复了数据库方式全局序列中线程安全的问题#489
  - 移除自定义语法 限制：全局序列值不能显式指定
    - 原来：insert into table1(id,name) values(next value for MYCATSEQ_GLOBAL,‘test’);
    - 现在1：insert into table1(name) values(‘test’);
    - 现在2：insert into table1 values(‘test’);
    - 注意时间戳方式需要该字段是bigint
- 改进对ER表的支持，智能处理连接隔离，解决同一事务内不可以同时写入父子表的问题，并优化ER表的执行计划
- 系统通过智能判断，对于一些没有显式配置但实际符合ER条件的表视作ER表同样处理
- 在中间件内进行智能解析与判断，使用正确的schema，替换有缺陷的checkSQLschema 参数
- conf/index_to_charset.properties的内容固化到代码 。
- 对于前端按照用户限制连接数，限制总连接数
- 改进原本的SQL统计，增加UPDATE/DELETE/INSERT

0.2.3 功能增强

- 提供了更强大的查询解析树，取代ShareJoin，使跨节点的语法支持度更广(join,union,subquery)，执行效率更高，同时聚合/排序也有了较大改进
- 提供科学的元数据管理机制，更好的支持show、desc等管理命令，支持不指定columns的insert语句 #7
- 元数据自动检查
  - 启动时对元数据进行一致性检查
  - 配置定时任务，对元数据进行一致性检查
- 提供更详实的执行计划，更准确的反映SQL语句的执行过程
  - 举例： mysql> explain select * from sharding_two_node a inner join sharding_four_node b on a.id =b.id;

```
+---------------+----------+-------------------------------------------------------------------------------------------+
| SHARDING_NODE | TYPE     | SQL/REF                                                                                   |
+---------------+----------+-------------------------------------------------------------------------------------------+
| dn1.0         | BASE SQL | select `a`.`id`,`a`.`c_char`,`a`.`ts`,`a`.`si` from  `sharding_two_node` `a` ORDER BY `a`.`id` ASC |
| dn2.0         | BASE SQL | select `a`.`id`,`a`.`c_char`,`a`.`ts`,`a`.`si` from  `sharding_two_node` `a` ORDER BY `a`.`id` ASC |
| dn1.1         | BASE SQL | select `b`.`id`,`b`.`c_flag`,`b`.`c_decimal` from  `sharding_four_node` `b` ORDER BY `b`.`id` ASC  |
| dn2.1         | BASE SQL | select `b`.`id`,`b`.`c_flag`,`b`.`c_decimal` from  `sharding_four_node` `b` ORDER BY `b`.`id` ASC  |
| dn3.0         | BASE SQL | select `b`.`id`,`b`.`c_flag`,`b`.`c_decimal` from  `sharding_four_node` `b` ORDER BY `b`.`id` ASC  |
| dn4.0         | BASE SQL | select `b`.`id`,`b`.`c_flag`,`b`.`c_decimal` from  `sharding_four_node` `b` ORDER BY `b`.`id` ASC  |
| merge.1       | MERGE    | dn1.0, dn2.0                                                                                              |
| merge.2       | MERGE    | dn1.1, dn2.1, dn3.0, dn4.0                                                                                |
| join.1        | JOIN     | merge.1, merge.2                                                                                          |
+---------------+----------+-------------------------------------------------------------------------------------------+
9 rows in set (0.00 sec)
```

- set 系统变量语句的改进
- set charset/names 语句的支持
- 分布式事务:XA实现方式的异常处理的改进
- 大小写敏感支持
- 支持DUAL
- 支持单次请求多语句(部分客户端有使用,C和C++常见)
- 不断丰富的路由规则优化和条件优化
- 升级Druid,跟进最新的解析器
- 升级fastjson,避免安全问题
- 智能判断reload连接变更，热更新连接池
- 对MySQL协议及GUI工具/Driver更友好的支持
- 增加更多的管理端命令,满足更多运维需要
- 缓存支持使用RocksDB
- 增加慢查询日志功能，兼容mysqldumpslow 和 pt-query-digest

- Trace功能，用于分析单条查询的性能瓶颈
- 大小写敏感依赖于后端MySQL
- 支持Prepared SQL Statement Syntax
- 支持以下子查询
  - The Subquery as Scalar Operand
  - Comparisons Using Subqueries
  - Subqueries with ANY, IN, or SOME
  - Subqueries with ALL
  - Subqueries with EXISTS or NOT EXISTS
  - Derived Tables (Subqueries in the FROM Clause)
- 支持dble层面的 View
- 支持MySQL8.0 默认登陆验证插件
- 提供自定义告警接口
- 支持自定义拆分算法
- 支持自增列可以设置为非主键列
- 可以观察执行中的DDL
- 提供配置预检查功能
- 流量暂停和恢复功能

0.2.4 功能裁减

- 仅保留枚举、范围、HASH、日期等分片算法，对这几个算法进行了可用性的改进，使之更加贴合实际应用，项目需要时可以按需提供
- 移除异构数据库支持
- 禁止某些不支持的功能，这些功能客户端调用时不会报错，但结果并非用户想要的结果，例如无效的set语句
- 移除目前实现有问题的第一结点库内分表模式
- 移除writeType参数，等效于原来writeType = 1
- 移除handleDistributedTransactions 选项，直接支持分布式事务

0.2.5 附录

一些功能改进的详细描述，见开源数据库中间件DBLE对MyCAT的增强与改进

0.2.6 附录2

Nio处理器（处理线程）内部为挂载不定数量个连接，并且循环响应每个连接的请求

在数据处理和数据接收进行线程分割之后(dble 2.18.02)，使得dble可以并发响应挂载在同一个NIO连接器(同一个processor线程)上的请求

e.g.

恰好我们存在场景连接1，2同时有请求过来，旧版本需要循环处理连接1，2的任务，在连接1的任务处理完成之前，连接2的任务无法进行处理

新的IO结构中，连接1的数据被接收完毕之后，NIO线程就可以接收连接2的数据，并且此时连接1的数据已经在executor线程池进行处理中，连接1，2之间的任务执行变成并行

0.3 快速开始

0.3.1 关于本节

- 本节内容为您介绍如何使用dble安装包快速部署并启动一个dble服务，并简单了解dble的使用和管理

0.3.2 安装准备

以下部分将被需要作为dble启动的基础支撑

- 两个启动的MySQL实例
  dble是通过连接mysql数据库实例来进行数据的存储，所以请至少准备两个个正在运行的mysql实例，
  假设您的机器上存在两个MySQL实例：
  A:$url=ip1:3306,$user=test,$password=testPsw
  B:$url=ip2:3306,$user=test,$password=testPsw
  请正确配置/etc/hosts，保证此MySQL实例可以正确访问，否则之后可能会报错 "NO ROUTE TO HOST"。
- JVM环境
  dble是使用java开发的，所以需要启动dble您先需要在机器上安装java版本1.8或以上，并且确保JAVA_HOME参数被正确的设置

0.3.3 下载并安装

- 通过此连接下载最新版本的安装包https://github.com/actiontech/dble/releases
- 解压并安装dble到指定文件夹中

```
mkdir -p $working_dir
cd $working_dir
tar -xvf actiontech-dble-$version.tar.gz
cd $working_dir/dble/conf
mv cluster_template.cnf cluster.cnf
mv bootstrap_template.cnf bootstrap.cnf
mv db_template.xml db.xml
mv user_template.xml user.xml
mv sharding_template.xml sharding.xml
```

0.3.4 dble的初始化配置

- 修改db.xml，找到其中的instanceM1 和instanceM2，将您对应的数据库信息进行替换

```
<dbInstance name="instanceM1" url="ip1:3306" user="your_user" password="your_psw" maxCon="1000" minCon="10"
                    primary="true" readWeight="1" id="xx1">
        </dbInstance>
```

```
<dbInstance name="instanceM2" url="ip2:3306" user="your_user" password="your_psw" maxCon="1000" minCon="10"
                    primary="true">
        </dbInstance>
```

0.3.5 启动并连接

- 启动命令

```
cd  $working_dir/dble
bin/dble start
```

- 如果启动失败请使用此命令查看失败的详细原因 tail -f logs/wrapper.log
- 使用mysql客户端直接连接dble管理端口，默认密码654321 mysql -p -P9066 -h 127.0.0.1 -u man1
- 您可以使用mysql一样的方式执行以下语句用于在实例上建立虚拟结点对应的schema

```
 create database @@shardingnode='dn$1-6';
```

另外此端口还可以执行一些其他命令

- 使用mysql客户端直接连接dble服务，默认密码123456 mysql -p -P8066 -h 127.0.0.1 -u root
- 您可以使用mysql一样的方式执行以下语句

```
use testdb;
drop table if exists tb_enum_sharding;
create table if not exists tb_enum_sharding (
  id int not null,
  code int not null,
  content varchar(250) not null,
  primary key(id)
)engine=innodb charset=utf8;
insert into tb_enum_sharding values(1,10000,'1'),(2,10010,'2'),(3,10000,'3'),(4,10010,'4');
```

0.4 快速开始(docker)

0.4.1 关于本节

- 本节内容为您介绍如何通过dockerhub上的dble镜像快速启动一个dble的demon

0.4.2 安装准备

- 安装docker
- 安装mysql连接工具，用于进行连接测试观察结果

0.4.3 安装过程

按照顺序依次执行以下docker命令：

> docker network create -o "com.docker.network.bridge.name"="dble-net" --subnet 172.18.0.0/16 dble-net
>
> docker run --name backend-mysql1 --ip 172.18.0.2 -e MYSQL_ROOT_PASSWORD=123456 -p 33061:3306 --network=dble-net -d mysql:5.7 --server-id=1
>
> docker run --name backend-mysql2 --ip 172.18.0.3 -e MYSQL_ROOT_PASSWORD=123456 -p 33062:3306 --network=dble-net -d mysql:5.7 --server-id=2
>
> sleep 30
>
> docker run -d -i -t --name dble-server --ip 172.18.0.5 -p 8066:8066 -p 9066:9066 --network=dble-net actiontech/dble:latest

通过以上命令依次创建一个docker网络，两个分别映射到主机33061,33062的mysql服务，一个将服务端和管理端映射到主机8066和9066端口的服务 服务将在约一分钟之后被启动，这是由于为了进行快速的启动需要对于mysql和dble的配置进行一些初始化

0.4.4 连接并使用

使用准备好的mysql连接工具连接主机的8066或者9066端口，在docker的默认配置中
8066 端口(服务端口能够执行SQL语句)的用户为 root/123456
9066 端口(管理端口能够执行管理语句)的用户为 man1/654321
此例子中准备了一些表格并提前进行了表格创建，若需要连接更多的表格配置详情和使用方法
请在dble-server容器中查阅/opt/dble/conf/shardingxml文件

在虚拟机已经有安装mysql客户端的状态下，可以使用以下默认命令进行连接

```
#连接dble sql服务端口
mysql -P8066 -u root -p123456 -h 127.0.0.1
#连接dble 控制管理端口
mysql -P9066 -u man1 -p654321 -h 127.0.0.1
#连接后端mysql1
mysql -P33061 -u root -p123456 -h 127.0.0.1
#连接后端mysql2
mysql -P33062 -u root -p123456 -h 127.0.0.1
```

0.4.5 环境清理

使用完成或者进行环境重建的时候可以使用以下命令进行环境的清空

```
docker stop backend-mysql1
docker stop backend-mysql2
docker stop dble-server
docker rm backend-mysql1
docker rm backend-mysql2
docker rm dble-server
docker network rm dble-net
```

0.4.6 docker-compose 版本快速启动

docker-compose启动需要先从github项目下载对应的配置文件

```
wget https://raw.githubusercontent.com/actiontech/dble/master/docker-images/docker-compose.yml
```

通过使用docker-compose的配置脚本直接启动两个mysql镜像以及一个dble镜像，在配置文件存放目录执行

```
docker-compose up
```

同样的，默认状态下启动dble的状态和quick-start一致，dble-server容器会启动两个端口8066/9066对外提供服务 可以使用默认用户root/123456连接8066端口进行测试，同时两个后端mysql开端口于本机33061/33062，默认的用户也是root/123456

在使用或者测试完毕之后，在配置文件存放目录下使用以下指令方便回收对应资源

```
docker-compose stop
docker-compose rm
```

0.4.7 尝试使用本地配置启动docker-compose

注意
本小结的内容需要用户在充分了解并掌握dble配置和结构的状态下进行,作为一种快速启动特定配置dble以供测试或调试使用 首次了解并使用dble的用户可以跳过此节

本地配置启动dble的原理是通过volumes配置的映射将本地的配置目录映射到docker容器中，之后初始化的时候在初始化脚本中将本地配置目录中的文件复制到对应的dble/conf目录中，之后再进行初始化和dble启动

首先需要在docker-compose.yml的最后一段dble-server容器配置中添加以下内容

```
        volumes:
           - /opt/test/conf:/opt/self_conf
```

此命令将本地的/opt/test/conf目录映射到目标容器/opt/self_conf目录中去 之后调整dble-server容器的启动命令

```
        command: ["/opt/dble/bin/wait.sh", "backend-mysql1:3306","--","/opt/self_conf/docker_init_start.sh"]
```

将原本调用的/opt/dble/bin/docker_init_start.sh修改为本次初始化准备使用的self_conf目录下的初始化脚本
这里对于本地启动的初始化脚本只给出如下一些建议

- 首先将需要修改的配置文件放置到/opt/dble/conf目录下
- 调用/opt/dble/bin/dble start启动dble服务
- 调用脚本/opt/dble/bin/wait-for-it.sh 脚本监听dble 8066 服务，等待dble服务启动完成
- 调用mysql命令对于dble的后端数据库以及初始数据进行初始化

0.5 快速开始(docker-compose)

0.5.1 关于本节

- 本节内容为你介绍如何快速使用dble的docker-compose文件来启动一个dble的quick start
  以及一个按照自定义的配置和sql脚本来启动dble quick start的用例

0.5.2 安装准备

- 安装docker
- 安装docker-compose
- 安装mysql连接工具，用于进行连接测试观察结果

0.5.3 安装过程

docker-compose启动需要先从github项目下载对应的配置文件

```
wget https://raw.githubusercontent.com/actiontech/dble/master/docker-images/docker-compose.yml
```

通过使用docker-compose的配置脚本直接启动两个mysql镜像以及一个dble镜像，在配置文件存放目录执行

```
docker-compose up
```

同样的，默认状态下启动dble的状态和quick-start一致，dble-server容器会启动两个端口8066/9066对外提供服务 可以使用默认用户root/123456连接8066端口进行测试，同时两个后端mysql开端口于本机33061/33062，默认的用户也是root/123456

在使用或者测试完毕之后，在配置文件存放目录下使用以下指令方便回收对应资源

```
docker-compose stop
docker-compose rm
```

0.5.4 连接并使用

使用准备好的mysql连接工具连接主机的8066或者9066端口，在docker的默认配置中
8066 端口(服务端口能够执行SQL语句)的用户为 root/123456
9066 端口(管理端口能够执行管理语句)的用户为 man1/654321
此例子中准备了表格并提前进行了表格创建，若需要连接更多的表格配置详情和使用方法
请在dble-server容器中查阅/opt/dble/conf/sharding.xml文件

0.5.5 使用自定义配置启动dble

在这里介绍下如何使用自定义的dble本地配置来启动docker-compose中的dble
首先，简单描述下默认docker-compose.yml执行的过程，dble-compose挨个启动容器，并且在最终启动dble-server的时候调用。 存在于镜像actiontech/dble:latest目录下的/opt/dble/bin/wait-for-it.sh脚本等待指定的（默认为mysql1容器的mysql服务端口）TCP端口启动， 待到指定TCP端口启动之后调用初始化脚本/opt/dble/bin/docker_init_start.sh对于dble-server这个容器进行初始化（替换配置文件，启动dble，执行sql文件）
下面是默认的docker_init_start.sh脚本

```
#!/bin/sh

echo "dble init&start in docker"

sh /opt/dble/bin/dble start
sh /opt/dble/bin/wait-for-it.sh 127.0.0.1:8066
mysql -P9066 -u man1 -h 127.0.0.1 -p654321 -e "create database @@shardingNode ='dn1,dn2,dn3,dn4'"
mysql -P8066 -u root -h 127.0.0.1 -p123456 -e "source /opt/dble/conf/testdb.sql" testdb

echo "dble init finish"

/bin/bash
```

可以轻易的看出脚本的内容非常简单，启动dble->等待dble服务启动->执行dble管理命令create database在后端mysql数据库中创建database->执行初始化sql脚本在dble中创建表和插入初始化数据
所以当需要使用非默认的情况进行启动dble-server容器时需要一以下几个基本步骤

- 需要按照以上的模块化步骤编写一个自己的初始化sh文件，然后照着需要的步骤对于dble进行启动和初始化
- 在dble-server的启动配置中添加volumes项，将linux本地的配置文件和初始化文件存放目录挂载至dble-server容器内部(注意dble本地放在/opt/dble下)
- 修改dble-server的启动命令，将初始化脚本修改为自定义的容器内部的初始化脚本

e.g.

docker-compose.yml修改如下部分

```
 dble-server:
        image: actiontech/dble:latest
        container_name: dble-server
        hostname: dble-server
        privileged: true
        stdin_open: true
        tty: true
        volumes:
            - ./:/opt/init/
        command: ["/opt/dble/bin/wait-for-it.sh", "backend-mysql1:3306","--","/opt/init/customized_script.sh"]
        ports:
            - "8066:8066"
            - "9066:9066"
        depends_on:
            - "mysql1"
            - "mysql2"
        networks:
            net:
              ipv4_address: 172.18.0.5
```

对应本地目录./存在以下文件可以自定义

```
cluster.cnf  bootstrap.cnf db.xml user.xml sharding.xml init.sql customized_script.sh
```

脚本customized_script.sh中的内容为

```
#!/bin/sh

echo "dble init&start in docker"

cp cluster_template.cnf /opt/dble/conf/
cp bootstrap_template.cnf /opt/dble/conf/
cp db_template.xml /opt/dble/conf/
cp user_template.xml /opt/dble/conf/
cp sharding_template.xml /opt/dble/conf/


sh /opt/dble/bin/dble start
sh /opt/dble/bin/wait-for-it.sh 127.0.0.1:8066
mysql -P9066 -u man1 -h 127.0.0.1 -p654321 -e "create database @@shardingNode ='dn1,dn2,dn3,dn4'"
mysql -P8066 -u root -h 127.0.0.1 -p123456 -e "source /opt/init/init.sql" testdb

echo "dble init finish"

/bin/bash
```

0.5.6 环境清理

使用完成或者进行环境重建的时候使用docker-compose stop/down来进行环境的清理

0.6 数据拆分简介

0.6.1 数据拆分简介

- 在dble中将表格按照数据分片的大致方式将表格的归为三个种类
  - 全局表：设置为全局表的表格将会在每个mysql节点上存在一个实体，并且在每个表格实体里面存放的都是全量的数据，一般用作字典表之类的数据量小、和多表关联或者是作为数据字典的表格。
  - 拆分表：设置为拆分表的表格会根据具体选择的拆分算法类型将其中的数据按照一定的规则分别存放到不同的mysql节点中去，每个节点存放一部分的数据。一般用以存放超大数据量的业务类表格，通过对于业务类表格的水平数据拆分实现性能的优化。
  - 非拆分表：设置为非拆分表的表格会在指定的mysql节点上单独存在一个表格实体，在此表格中存放全量数据。一般用于存放数据压力不大的业务类表格，类似冷门功能的业务数据表。

0.6.2 规划拆分方案

- 系统开发之前，应该对业务特点进行深度分析，对数据规模进行较准确的评估，根据表的数据量、数据特点和表与表之间的关系，决定哪些表需要分片。数据节点的数量应该根据数据量和访问性能要求合理配置，过多的节点数量不仅浪费资源，而且增加运维复杂度，有时不仅不能提升性能,还会降低性能。
- 对于小表（数据量不大，如千万以下），尽量不要配置成拆分表。如果表比较独立，与其他的表基本上不进行join运算，可以作为非拆分表处理，性能不能满足要求的时候可以通过配置读写分离机制来提高性能。如果需要与拆分表进行join运算，可以配置成全局表。
- 关于每个实例的规格，选择的根据有两个因素:
  - 需要存储的最大磁盘空间，需要通过拆分算法计算，根据存储数据量最大的节点来计算，并预估未来2到3年的数据增长;
  - 估算一个实例需要的最大 QPS和TPS，要根据最慢的节点估算。
- QPS、TPS与实际的SQL语句、数据量、数据结构和数据节点的规格有关，根据经验来估计的话，很可能偏差较大。系统建设之前，应该配置同等或者接近的环境，进行针对性的性能测试，从而做出准确的判断。

0.6.3 数据拆分表格的配置方法

- 数据拆分配置包括节点配置和拆分规则配置。节点配置决定了数据的物理存储方式，由主机和节点组成，主机代表具体的数据库实例，节点代表实例中的数据库。拆分规则决定了数据在不同的分区上读写的规则，由拆分算法和应用拆分算法的逻辑库、表、字段组成。
- 数据写入时，系统对指定的字段值应用拆分算法得到目标节点，然后将数据写入目标节点。数据读取时，系统对查询条件应用拆分算法得到数据源节点，从源节点获取数据，在中间层进行结果合并之后返回请求方，对于不同的查询条件，源节点可能有一个或多个。

0.7 docker-compose快速开始dble高可用

0.7.1 关于本节

- 本节内容为你介绍如何快速使用docker-compose文件来启动两组mgr(六个mysql实例)、一个dble的quick start以及验证dble高可用效果

0.7.2 安装准备

- 安装docker
- 安装docker-compose
- 安装mysql连接工具，用于进行连接测试观察结果

0.7.3 安装过程

注意：可以通过 `wget https://raw.githubusercontent.com/actiontech/dble/master/docker-images/docker-compose-mgr.yml` 方式下载**docker-compose-mgr.yml**文件后，直接跳至第**4**点（执行**docker-compose up**命令，启动容器）

1. 下载dble项目，找到docker-images目录:

```
docker-images
     ├─Dockerfile-Mgr            构建actiontech/dble-mgr:latest镜像
     ├─docker-compose-mgr.yml    搭建六个mysql实例(两组mgr)和dble-server容器
     └─quick-start
            ├─start-mgr.sh            初始化两组mgr
            ├─docker_init_start.sh    执行start-mgr.sh 、启动dble、初始数据
            ├─db_mgr.xml              配置6个mysql实例
            ├─bootstrap.cnf           将useOuterHa设置为false
            └─...
```

2. 将bootstrap.cnf中的useOuterHa参数配置为false

```
-DuseOuterHa=false
```

3. 执行Dockerfile-Mgr文件，构建actiontech/dble-mgr:latest镜像

```
[root@localhost]docker-images# docker build -f="Dockerfile-Mgr" -t="actiontech/dble-mgr:latest" .
[root@localhost]docker-images# docker images
REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
actiontech/dble-mgr   latest              d773d83fe832        46 hours ago        711MB
```

4. 执行docker-compose-mgr.yml文件，启动六个mysql(两组mgr)和dble-server容器

```
[root@localhost]~# cd docker-images
[root@localhost]docker-images# docker-compose -f docker-compose-mgr.yml up -d
[root@localhost]docker-images# docker-compose -f docker-compose-mgr.yml ps
  Name                  Command               State                Ports
-------------------------------------------------------------------------------------
dble-server   /opt/dble/bin/wait-for-it. ...   Up      0.0.0.0:8066->8066/tcp,0.0.0.0:9066->9066/tcp
mgr-a-1       docker-entrypoint.sh mysql ...   Up      0.0.0.0:3307->3306/tcp,33060/tcp
mgr-a-2       docker-entrypoint.sh mysql ...   Up      0.0.0.0:3308->3306/tcp,33060/tcp
mgr-a-3       docker-entrypoint.sh mysql ...   Up      0.0.0.0:3309->3306/tcp,33060/tcp
mgr-b-1       docker-entrypoint.sh mysql ...   Up      0.0.0.0:3310->3306/tcp,33060/tcp
mgr-b-2       docker-entrypoint.sh mysql ...   Up      0.0.0.0:3311->3306/tcp,33060/tcp
mgr-b-3       docker-entrypoint.sh mysql ...   Up      0.0.0.0:3312->3306/tcp,33060/tcp
```

5. 查看两组mgr关系：

```
[root@localhost]docker-images# docker exec mgr-a-1 mysql -h127.0.0.1 -p3306 -uroot -p123456 \
-e "SHOW STATUS LIKE 'group_replication_primary_member';" \
-e "SELECT * FROM performance_schema.replication_group_members;"

+----------------------------------+--------------------------------------+
| Variable_name                    | Value                                |
+----------------------------------+--------------------------------------+
| group_replication_primary_member | 72da84d7-0c4b-11eb-9f0e-0242ac120002 |
+----------------------------------+--------------------------------------+
+---------------------------+--------------------------------------+-------------+-------------+--------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
+---------------------------+--------------------------------------+-------------+-------------+--------------+
| group_replication_applier | 72da84d7-0c4b-11eb-9f0e-0242ac120002 | mgr-a-1     |        3306 | ONLINE       |
| group_replication_applier | 7314efdd-0c4b-11eb-ba28-0242ac120004 | mgr-a-3     |        3306 | ONLINE       |
| group_replication_applier | 733b00fe-0c4b-11eb-bbea-0242ac120003 | mgr-a-2     |        3306 | ONLINE       |
+---------------------------+--------------------------------------+-------------+-------------+--------------+

[root@localhost]docker-images# docker exec mgr-b-1 mysql -h127.0.0.1 -p3306 -uroot -p123456 \
-e "SHOW STATUS LIKE 'group_replication_primary_member';" \
-e "SELECT * FROM performance_schema.replication_group_members;"

+----------------------------------+--------------------------------------+
| Variable_name                    | Value                                |
+----------------------------------+--------------------------------------+
| group_replication_primary_member | 728c327d-0c4b-11eb-9300-0242ac120005 |
+----------------------------------+--------------------------------------+
+---------------------------+--------------------------------------+-------------+-------------+--------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
+---------------------------+--------------------------------------+-------------+-------------+--------------+
| group_replication_applier | 728c327d-0c4b-11eb-9300-0242ac120005 | mgr-b-1     |        3306 | ONLINE       |
| group_replication_applier | 732c5b3b-0c4b-11eb-9eb1-0242ac120007 | mgr-b-3     |        3306 | ONLINE       |
| group_replication_applier | 733c6350-0c4b-11eb-b0fb-0242ac120006 | mgr-b-2     |        3306 | ONLINE       |
+---------------------------+--------------------------------------+-------------+-------------+--------------+
```

6. 观察dble日志, 进入dble-server容器查看/opt/dble/logs/下的相关日志

```
[root@localhost]docker-images# docker exec -it dble-server bash
[root@dble-server /]# less /opt/dble/logs/wrapper.log
[root@dble-server /]# less /opt/dble/logs/dble.log

#bootstrap.cnf配置中useOuterHa参数配置为false时，才会有该日志生成
[root@dble-server /]# less /opt/dble/logs/custom_mysql_ha.log
```

0.7.5 验证高可用效果

1. 前提bootstrap.cnf配置文件中将useOuterHa设置为false(可参考高可用切换接口)
2. 进入dble-server容器，查看/opt/dble/conf/db.xml配置如下：

```
[root@localhost]docker-images# docker exec -it dble-server bash
[root@dble-server /]# cat /opt/dble/conf/db.xml

<dbGroup name="dbGroup1" rwSplitMode="2" delayThreshold="100">
   <heartbeat>show slave status</heartbeat>
   <dbInstance name="instanceM1" url="172.18.0.2:3306" user="root" password="123456" maxCon="300" minCon="10"
               primary="true" readWeight="1" id="xx1">
   </dbInstance>
   <dbInstance name="instanceS1" url="172.18.0.3:3306" user="root" password="123456" maxCon="1000" minCon="10" readWeight="2">
       <property name="testOnCreate">false</property>
   </dbInstance>
   <dbInstance name="instanceS2" url="172.18.0.4:3306" user="root" password="123456" maxCon="1000" minCon="10" readWeight="2">
       <property name="testOnCreate">false</property>
   </dbInstance>
</dbGroup>

[root@dble-server /]# exit
```

3. 停用mgr-a-1的主实例，进入dble-server容器中查看变化：custom_mysql_ha.log中出现172.18.0.2:3066...is not alive、db.xml中dbGroup1组的主实例为instanceS1：

```
[root@localhost]docker-images# docker-compose -f docker-compose-mgr.yml stop mgr-a-1
[root@localhost]docker-images# docker exec -it dble-server bash
[root@dble-server /]# less /opt/dble/logs/custom_mysql_ha.log
...
2020-10-12 07:05:08 [DBLEDbGroupsCheck] [INFO] DbInstance 172.18.0.2:3306 in dbGroup1 is not alive!
2020-10-12 07:05:08 [DBLEDbGroupsCheck] [INFO] DbInstance 172.18.0.3:3306 in dbGroup1 is normal!
2020-10-12 07:05:08 [DBLEDbGroupsCheck] [INFO] DbInstance 172.18.0.4:3306 in dbGroup1 is normal!
2020-10-12 07:05:08 [DBLEDbGroupsCheck] [INFO] DbInstance 172.18.0.5:3306 in dbGroup2 is normal!
2020-10-12 07:05:08 [DBLEDbGroupsCheck] [INFO] DbInstance 172.18.0.6:3306 in dbGroup2 is normal!
2020-10-12 07:05:08 [DBLEDbGroupsCheck] [INFO] DbInstance 172.18.0.7:3306 in dbGroup2 is normal!
...

[root@dble-server /]# cat /opt/dble/conf/db.xml

<dbGroup name="dbGroup1" rwSplitMode="2" delayThreshold="100">
   <heartbeat>show slave status</heartbeat>
   <dbInstance name="instanceM1" url="172.18.0.2:3306" user="root" password="123456" maxCon="300" minCon="10"
                readWeight="1" id="xx1">
   </dbInstance>
   <dbInstance name="instanceS1" url="172.18.0.3:3306" user="root" password="123456" maxCon="1000" minCon="10" readWeight="2" primary="true">
       <property name="testOnCreate">false</property>
   </dbInstance>
   <dbInstance name="instanceS2" url="172.18.0.4:3306" user="root" password="123456" maxCon="1000" minCon="10" readWeight="2">
       <property name="testOnCreate">false</property>
   </dbInstance>
</dbGroup>
```

4. 结束

```
[root@dble-server /]# exit
[root@localhost]docker-images#  docker-compose -f docker-compose-mgr.yml down
```

1.dble配置文件基础

- 配置文档列表以及相关对应功能
  - cluster.cnf:集群参数配置
  - bootstrap.cnf:实例参数配置，包括JVM启动参数，dble性能，定时任务，端口等
  - user.xml:dble 用户配置
  - db.xml：数据库相关配置
  - sharding.xml：数据拆分相关配置
  - log4j.xml：log4j2.xml，配置日志参数
  - 全局序列：全局序列相关配置
  - cache配置：配置缓存参数
  - 自定义拆分算法
  - 自定义告警
  - 自定义全局表一致性检查
- 重要日志及文件：
  - /logs/wrapper.log：启动日志，如果dble启动失败，将会有日志出现在这个文件中
  - /logs/dble.log：dble日志，日志记录并反馈dble执行过程中的重要信息
- 配置文件变更记录：
  - 配置文件变更记录

dble 在3.20.07.0 版本做了配置的重构。历史变更请参考2.20.04.0的变更

2. 配置升级

可以通过升级工具dble_update_config将配置从2.20.04.0 升级到3.20.07.0，如果是更早版本，建议先升级到2.20.04.0

升级工具用法：

```
dble_update_config [-i=read_dir] [-o=write_dir] [-p=rootPath]
```

read_dir/write_dir: 如果不指定，缺省值为当前目录，建议指定或者提前备份配置 rootPath:如果集群模式是zk, 那么缺省值为 `/dble` , 如果集群模式是ucore, 缺省值为 `universe/dble`

工具将会读取文件:
myid.properties
wrapper.conf
server.xml
schema.xml
rule.xml
log4j2.xml
cacheservice.properties(option)
sequence_distributed_conf.properties for type3 (option)
sequence_time_conf.properties for type2 (option)

然后写出文件:
cluster.cnf
bootstrap.cnf
user.xml
db.xml
sharding.xml
log4j2.xml
cacheservice.properties(option)

3. 重构后的配置概览图：

db.xml
变更：
1 datahost->dbGroup
2 不再使用writehost和readhost， 而是统一命名为dbInstance， 增加primary的boolean属性来区分，各dbInstance平级
3 maxCon,minCon的属性配置下沉到实例， 其他连接池属性使用dbInstance下的property来声明，默认可不配置。
4.writehost/readHost的属性host改成name
5.移除 tempReadHostAvailable功能。默认为读可持续
6. 池的心跳处理，周期及配置放在dbGroup级别，各个周期可不同
7.balance->rwSplitMode
8.keepOrig->disableHA
9. slaveThreshold->delayThreshold
10.weight->readWeight
备注：
当前池级别的心跳用处
1.快速拦截流量，可配置敏感程度
2.延迟检测，需要特殊心跳设置
3.查看是否read only，需要特殊心跳设置
dbInstance
name（原host）
*id
url
user
password
primary
minCon
maxCon
* usingDecrypt
* readWeight
* disabled
write 1:1
read n:1
dbGroup
name
* rwSplitMode
* disableHA
* delayThreshold
* heartbeat
1:n
1:1
sharding.xml
childTable
name
joinColumn
parentColumn
* incrementColumn
* sqlMaxLimit
N;1
shardingTable
name
shardingNode
shardingColumn
function
* incrementColumn
* sqlMaxLimit
* sqlRequiredSharding
N : 1
function
name
+ property
m:n
globalTable
name
shardingNode
* checkClass
* cron
* sqlMaxLimit
m:n
singleTable
name
shardingNode
* sqlMaxLimit
1:n
变更：
1.rule.xml删除，function保留，拆分表直接增加shardingColumn列来标识
2. table拆分三类，globalTable，singleTable和sharding table
3.cacheKey功能废弃
4.shardingTable/childTable属性名字由xxKey改为xxColumn
5.废弃globalCheck,.其他两个配置已经能表达含义了,globalCheckClass->checkClass
6.ruleRequired更名为sqlRequiredSharding
7.needAddLimit更名为sqlMaxLimit，默认继承schema，如果-1表示不限制
shardingNode
name
dbGroup
database
1:1
schema
name
* shardingNode
* sqlMaxLimit
m:n
user.xml
变更：
1.用户分三类，管理用户，读写分离用户以及分库用户
2.whitehosts较简单，所以冗余在用户内部更合适。
3.增加租户字段，可选，登陆时候检查连接字符串
4.管理用户 审计时需要多个（未来审计）
readonly表示只监控，不配默认可以运维
rwSplitUser
name
* tenant
dbGroup
password
* usingDecrypt
* whiteIPs
* blacklist
* maxCon
managerUser
name
password
* usingDecrypt
* whiteIPs
* readonly
* maxCon
shardingUser
name
* tenant
schemas
password
* usingDecrypt
* whiteIPs
* blacklist
* maxCon
* readonly
* privileges系列
blacklist
name
略
变更
1.连接池有效性心跳移到连接池配置中
2.连接池内部连接的有效性委托连接池处理
3.组级别的配置和myid.properties合并， 更名cluster.cnf
3.1cluster裂变为clusterEnable和clusterMode
3.2ipAddress->clusterIP,port->clusterPort,clusterId->clusterId,clusterHa->needSyncHa
3.3 myid下沉到bootstrap.cnf的instanceName
3.4 serverID沉到bootstrap.cnf的serverID
4.实例级别的配置都放在bootstrap.cnf里，被wrapper.conf引用，wrapper.conf 放在bin下
bootstrap.cnf还包含JVM参数
5.部分可动态变更的参数， 放在 bootstrap.dynamic.cnf,读的时候覆盖bootstrap.cnf
cluster.cnf
Group Parameters整组运维/不可变动
showBinlogStatusTimeout，
sequenceHandlerType
原myid.properties的集群部分，见上面备注
原自增序列的时间戳方式部分配置，参见全局序列章节
增加rootPath
bootstrap.cnf
Instance Parameters实例重启变动生效
* 后端连接socket 配置
* 前端连接socket 配置
* Session预留内存配置
* 堆外内存管理
系统服务基本参数（除了在组配置中的）
* 表结构一致性检查
* processor内部前后端连接检查
* 普通事务日志相关
* 视图相关参数
* XA 事务
* 队列大小参数
* 使用Nest Loop优化
* load data 相关配置
* 统计管理
连接缺省值
* 功能性配置
fakeMySQLVersion
useOuterHa
DBLE_HOME->homePath
增加instanceName对应原myid.properties的myid，并且取代serverNodeId原配置
增加instanceId对应自增序列的实例id配置
增加serverId用于告警
bootstrap.dynamic.cnf
Dynamic Parameters
通过运维命令变动，未来支持多值和落盘
* 慢查询日志相关配置
* 流量控制相关参数
* 告警开关
enableAlert
enableSlowLog
flushSlowLogPeriod
flushSlowLogSize
sqlSlowTime
enableFlowControl
flowControlStartThreshold
flowControlStopThreshold

1.1 cluster.cnf

本配置文件为key=value的形式，用于配置在一个dble集群中各个dble实例必须一致的一些参数，key/value值大小写敏感。
如果没有此文件启动dble，将不具有集群功能（配置同步，DDL同步等），并且部分参数使用默认值。

注意：本配置文件如果需要修改,需要多个dble节点停下来，修改配置，重新初始化集群元数据(zk可删除原有元数据),然后挨个启动dble

配置名称	配置内容&示例	可选项/默认值	详细描述
clusterEnable	是否开启集群功能	非必需项，可配置true/false。默认值为false	标记是否开启集群功能，如果开启，需要配置 clusterIP，clusterPort，rootPath，clusterID
clusterMode	集群调度中心的模式	可配置zk/ucore。无默认值，clusterEnable开启后不配此项，会导致启动报错	zk表示集群调度中心使用zookeeper，ucore表示使用爱可生商业集群调度中心
clusterIP	集群调度中心的IP	无默认值，clusterEnable开启后不配此项，会导致启动报错	clusterMode为zk时，配置zk完整地址，例如10.186.19.aa:2281,10.186.60.bb:2281；clusterMode为core时，配置ucore的ip地址，可为逗号隔开的集群IP地址
clusterPort	集群调度中心的端口	无默认值	clusterMode为zk时，此项可以空缺；clusterMode为ucore时，配置ucore的端口号。
rootPath	集群调度中心的根目录	无默认值，clusterEnable开启后不配此项，会导致启动报错	集群调度中心的根目录，按需配置
clusterId	本dble集群的名称	无默认值，clusterEnable开启后不配此项，会导致启动报错	本dble集群的名称，多个使用同一实例名称的dble会被视为在同一集群下
needSyncHa	ha接口是否需要集群同步	可配置true/false。默认值为false	当使用高可用接口时，是否需要使用集群同步，当配置为true时，实例级别的参数useOuterHa会被置为true
showBinlogStatusTimeout	拉取一致性binlog线的超时时间	可配置正整数，默认60000 ,单位毫秒	拉取一致性binlog线的超时时间
sequenceHandlerType	全局序列处理器的方式	可配置1~4，默认值2	在初始化的时候根据这个配置选择不同的序列生成器进行加载 1，MySQL offset-step序列方式， sequence信息存储在数据库中 2，时间戳方式(类Snowflake) 3，分布式time序列(类Snowflake) 4，分布式offset-step序列
sequenceStartTime	时间相关的全局序列的起始时间	非必需项，默认2010-11-04 09:42:54，修改需要保持这个格式	仅当sequenceHandlerType为2或3时候这个值有意义
sequenceInstanceByZk	分布式time序列是否使用zk来生成唯一实例id	非必需项，可以配置true/false,默认true	仅当sequenceHandlerType为3，并且clusterMode为zk时，这个值有意义

1.1.1 不使用的例子

不配置 或者

```
clusterEnable=false
#showBinlogStatusTimeout=60000
sequenceHandlerType=2
#sequenceStartTime=2010-11-04 09:42:54
#sequenceInstanceByZk=true
```

1.1.2 使用ZK的例子

```
clusterEnable=true
clusterMode=zk
clusterIP=10.186.19.aa:2281,10.186.60.bb:2281
rootPath=/dble
clusterId=cluster-1
#needSyncHa=false
#showBinlogStatusTimeout=60000
sequenceHandlerType=2
#sequenceStartTime=2010-11-04 09:42:54
#sequenceInstanceByZk=true
```

1.1.3 使用ucore的例子

```
clusterEnable=true
clusterMode=ucore
clusterIP=10.186.19.aa,10.186.60.bb
clusterPort=5700
rootPath=universe/dble
clusterId=cluster-1
#needSyncHa=false
#showBinlogStatusTimeout=60000
sequenceHandlerType=2
#sequenceStartTime=2010-11-04 09:42:54
#sequenceInstanceByZk=true
```

1.2 bootstrap.cnf

本配置文件为dble实例启动时候加载的配置文件。默认使用wrapper.cnf启动的外置参数文件的格式，即使不使用wrapper启动时，也会加载此文件。

1.2.1 jvm启动参数

本文件第一部分为JVM启动参数,可以根据需要修改。

```
-agentlib:jdwp=transport=dt_socket,server=y,address=8088,suspend=n
-server
-XX:+AggressiveOpts
-Dfile.encoding=UTF-8
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=1984
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.host=127.0.0.1
-Xmx4G
-Xms1G
-XX:MaxDirectMemorySize=2G
```

大部分以上的配置都没有特殊的意义，仅仅是一般的JVM配置，关于JVM调优的部分需要以现实情况进行操作，在此仅介绍以个特殊情况

MaxDirectMemorySize上限值为81917M，约等于79G。需要根据机器的情况进行提前适配，不然会导致服务无法正常启动 具体的细节为需要大于bufferPoolPageNumber*bufferPoolPageSize，这两个选项在第二部分中配置

bufferPoolPageNumber 的默认配置是20 × 机器CPU线程数(注意这里I5和I7的CPU可能会返回不同的结果)，上限值为32767 ×1024
bufferPoolPageSize 的默认配置是 4 × 512 ×1024，上限值为32767 ×1024

以下为建议值：
dble总内存=0.6 可用物理内存*(*剔除操作系统,驱动等的占用*)*
Xmx = 0.4 dble总内存
MaxDirectMemorySize = 0.6 * dble总内存

另外，在启动参数中的bufferPoolPageNumber 和bufferPoolPageSize 受MaxDirectMemorySize影响。
建议 和bufferPoolPageSize设置为2M，bufferPoolPageNumber 设置为 取整(MaxDirectMemorySize * 0.8 /bufferPoolPageSize)

1.2.2 dble系统参数

本文件第二部分为dble系统参数,配置格式遵照启动参数的格式：-Dkey=value， 注意如果是通过JSW启动的dble服务，需要复合JSW对配置文件的要求，也就是不能有空格，具体参见 wrapper.app.parameter_file
具体系统参数含义参见以下表格。

模块	配置名称	配置内容	默认值/单位	详细作用原理或应用	配置范围
	homePath	基本目录 慢查询日志（slowlogs）、视图记录日志（viewConf）、xa的tm日志（xalogs）、load data临时文件（temp）等存放路径的父目录	默认为当前目录	其他一些类似于事务或者视图存储的的根路径	有效路径, 推荐默认
	instanceName	实例名称	无默认值，不配置会报错	集群配置时的唯一标识，xa事务时的实例标识	集群内唯一值
	instanceId	实例id	无默认值	集群配置时的唯一标识， 全局序列时的唯一标识	仅当sequenceHandlerType=2 或者 3时候有意义. 当sequenceHandlerType=2，合法值范围为0~1023 当sequenceHandlerType=3，合法值范围为0~,511
	serverId	服务器名称	默认值为服务器IP	dble所在机器的名称	用于告警时候报告服务器名称
	bindIp	服务IP	默认 "0.0.0.0"	在服务初始化的时候作为侦听的IP	有效IP地址, 推荐默认
	serverPort	服务端口	默认8066	在服务初始化的时候作为服务侦听的端口	机器空闲端口

系统服务基本参数	managerPort	控制端口	默认9066	在服务初始化的时候作为控制侦听的	机器空闲端口
	maxCon	控制最大连接数	默认0	默认不做限制。若maxCon大于0,建立的连接数大于maxCon之后,建立连接会失败.注意当各个用户的maxcon总和值大于此值时，以当前值为准。 全局maxCon不作用于manager用户	正整数
	processors	NIO前端处理器的数量	默认java虚拟机核数,单位个	进行前端网络IO吞吐的线程数	正整数
	backendProcessors	NIO后端处理器的数量	默认java虚拟机核数,单位个	进行后端网络IO吞吐的线程数	正整数
	processorExecutor	前端业务处理线程池数量	默认(单核为2,否则等于宿主机核数)	进行前端具体业务处理的线程池大小，负责解析路由下发	正整数
	backendProcessorExecutor	后端业务处理线程池数量	默认(单核为2,否则等于宿主机核数)	进行后端具体业务处理的线程池大小，负责回收结果集并合并	正整数
	complexExecutor	复杂查询后端业务线程池数量	默认(单核为2,否则等于宿主机核数，宿主机核数大于8时，数量为8	负责复杂查询或者子命令结果集的回收	正整数
	writeToBackendExecutor	广播下发SQL时候批量处理的线程池梳理	默认(单核为2,否则等于宿主机核数)	负责广播下发SQL时候批量处理	正整数
	fakeMySQLVersion	dble模拟mysql版本号	默认NULL	模拟成正常的MySql版本在进行前端协议交互的时候能够使用到	MYSQL版本号
	serverBacklog	前端tcp连接 backlog	默认2048	前端tcp连接 backlog	正整数
	usePerformanceMode	是否启用性能模式	默认0/单位无	开启之后Dble会大量占用CPU资源，并提供更高的性能体现,慎用	1-是0-否
	useOuterHa	是否启用外部高可用联动	默认为true，若此时不设置外部高可用，将不做切换	如关闭此功能并且dble部署方式为单机，将使用默认的切换方式，详情请见切换相关章节	true/false
连接缺省值	charset	字符集	utf8mb4	服务启动后的默认字符集于所有字符集相关的部分，包括前端连接和后端连接	有效字符集
	maxPacketSize	包大小限制	默认 4×1024×1024	限制请求的包大小，启动时候dble会拉取并尝试同步(此值+1024)到每个dbInstance，如果同步失败，就取配置值与各个dbInstance中最小的那个值-1024.留出1024的冗余用于对SQL改写或者上下文同步的支持	正整数
	txIsolation	隔离级别	默认 3	执行具体SQL的时候会比较前后端连接， 启动时候dble会会拉取并尝试同步此值到每个dbInstance，如果同步失败	1-READ_UNCOMMITTED 2-READ_COMMITTED

				或者session级别重新设置该值，session在SQL下发之前，会执行session级别的隔离级别set	3-REPEATABLE_READ 4-SERIALIZABLE
	autocommit	是否自动提交	默认 1，自动提交	启动时候dble会会拉取并尝试同步此值到每个dbInstance 如果同步失败或者session级别重新设置该值，执行具体SQL的时候会比较，如果不一致将会执行session级别的set	0/1
功能性配置	useCompression	是否启用数据压缩	默认 0否	使用mysql压缩协议	1 - 是 0 - 否
	capClientFoundRows	是否开启Client_Found_Rows权能标识	默认 false，关闭	dble开启Client_Found_Rows权能标识	true - 开启 false - 关闭
	usingAIO	是否启用AIO	默认0	在初始化服务的时候将会作为判断启用AIO或是NIO的依据	1 - 是 0 - 否
线程使用率统计	useThreadUsageStat	开启线程使用率统计	默认0/单位无	开启之后能在管理端通过管理命令show @@thread_used查看各个部分的线程使用情况	1-是0-否
	useCostTimeStat	是否启用查询耗时统计	默认0/单位无	开启之后以一定的比例统计查询过程中的各个步骤的耗时情况，可以使用BTraceCostTime.jav进行观测,也可在管理端使用show @@cost_time观察	1-是0-否
	maxCostStatSize		默认100	show @@cost_time结果最近保留的行数	
	costSamplePercent	查询采样百分比	默认1/单位%	在耗时采样统计中实际采样百分比为costSamplePercent	
一致性检查	checkTableConsistency	表格一致性检查	默认0	如果值为1，那么在服务初始化的时候会启动一个定时任务，在 定时任务会检查DB是不是存在，表格是不是存在，表结构是否一致	1-是,0-否
	checkTableConsistencyPeriod	表格一致性检查周期	默认30×60×1000 ,单位毫秒	表格一致性检查周期	正整数
processor内部前后端连接检查	sqlExecuteTimeout	后端连接执行超时时间	默认 300 ,单位秒	如果超过这个时间没有完毕，就直接关闭连接	正整数
	idleTimeout	前端连接无响应超时时间	默认 10 × 60 × 1000 ,单位毫秒	在processor定时连接检查时，发现前端连接上一次的读写距今超过阀值，会直接关闭连接	正整数
	processorCheckPeriod	processor定时任务检查周期	1000 ,单位毫秒	根据此配置定时的检查在processor中的前后端连接的状态	正整数
	backSocketSoRcvbuf	后端套接字接收缓冲区大小	1024×1024×4 ,单位字节	在创建后端管道的时候作为buffer大小使用	正整数

后端连接socket 配置	backSocketSoSndbuf	后端套接字发送缓冲区大小	1024×1024 ,单位字节	在创建后端管道的时候作为buffer大小使用	正整数
	backSocketNoDelay	后端Nagle算法是否禁用	默认1/单位无	在创建后端管道的时候禁用延迟加载，会影响网络包的情况 详见相关资料	1-是, 0-否
前端连接socket 配置	frontSocketSoRcvbuf	前端套接字接受缓冲区大小	1024 × 1024 ,单位字节	在读取网络传输信息的时候作为每次缓冲的大小使用	正整数
	frontSocketSoSndbuf	前端套接字发送缓冲区大小	1024×1024×4 ,单位字节	在创建前端管道的时候作为buffer大小使用	正整数
	frontSocketNoDelay	前端Nagle算法是否禁用	默认1	在创建前端管道的时候禁用延迟加载 相关资料	1-是,0-否
Session预留内存配置	orderMemSize	session中的复杂查询order预留内存	默认4,单位M	在session初始化的时候创建内存分配对象，在复杂查询order by的时候使用到	正整数
	otherMemSize	session中的复杂查询其他预留内存	默认4,单位M	在session初始化的时候创建内存分配对象，在复杂查询subQuery以及distinctd的时候使用	正整数
	joinMemSize	session中的复杂查询join预留内存	默认4，单位M	在session初始化的时候创建内存分配对象，在复杂查询join使用到	正整数
堆外内存管理	bufferPoolChunkSize	内存池中分配的最小粒度	默认4096 ,单位字节	内存池中分配的最小粒度，需要的大小除以此粒度，向上取整	
	bufferPoolPageNumber	预分配内存池页数量	默认 0.8 × MaxDirectMemorySize / bufferPoolPageSize(default 2M)	在初始化的时候通过和bufferPoolPageSize的相乘确定缓冲池最后的大小，内存配置建议见1.4_wrapper.conf	
	bufferPoolPageSize	预分配内存池页大小	默认512×1024×4 ,单位字节	在初始化的时候通过和bufferPoolPageNumbe的相乘确定缓冲池最后的大小, 注意：虚拟机参数***MaxDirectMemorySize(***见***1.4_wrapper.conf)***需要大于***bufferPoolPageNumber * bufferPoolPageSize***，否则会触发***OOM***	
	mappedFileSize	文件映射区单个文件最大体积	默认1024×1024×64 ,单位字节	在初始化的时候此参数确定文件映射区最大容量,参见内存管理章节	
	bufferUsagePercent	是否清理大结果集阈值	默认80, 单位百分号	定时任务resultSetMapClear使用，周期clearBigSQLResultSetMapMs，定时清理统计的结果集，当定时任务执行时发现结果集统计超过阀值，触发清理结果集的行为	0-100
	useSqlStat	是否启用SQL统计	默认1/单位无	启用之后会对于下发的查询进行SQL的统计，分别按照用户、表格、查询条件进行存放在内存中 并且开启之后会随之开启recycleSqlStat定时任务以固	1-是0-否

	clearBigSQLResultSetMapMs	定期大结果清理时间	默认600×1000 ,单位毫秒	定时任务resultSetMapClear的执行周期，定时清理记录的查询结果集	正整数
	sqlRecordCount	慢查询记录阈值	默认10 ,单位条	在定时任务recycleSqlStat中会进行sql记录的清理,当发现记录的慢查询SQL数量超过阀值时，会仅保留阀值数量个元素	正整数
	maxResultSet	大结果集阈值	默认512×1024 ,单位字节	当查询的结果集超过这个阀值时，查询的SQL和查询结果集的大小才会被记录到结果集统计里面	正整数
普通事务日志相关	recordTxn	事务log记录	默认0	在初始化服务的时候会注册一个类，其作用就是将事务的log写到一个指定的文件中	1-是，0-否
	transactionLogBaseDir	事务log目录	默认当前路径/txlogs	当开启日志log记录时，记录文件会被存放在对应目录下	绝对路径
	transactionLogBaseName	事务log文件名称	默认server-tx	事务记录存储文件的文件名	符合运行系统 文件的命名规范
	transactionRotateSize	事务日志单个文件大小。	默认16 ,单位M		正整数
XA 事务	xaRecoveryLogBaseDir	xa的tm日志路径	dble目录/tmlogs/	此日志涉及到XA事务状态的记录，并且在Dble意外重启之后需要从里面获取重启之前的xa事务状态，切勿自行修改	绝对路径
	xaRecoveryLogBaseName	xa的tm日志名称	tmlog		符合运行系统 文件的命名规范
	xaSessionCheckPeriod	XA定时任务执行周期	默认1000， 单位ms	在server开始的时候会注册一个定时任务以此参数为执行周期 （注：定时任务必定会被注册） 如果有尝试多次没有成功提交的session在之后的定时任务会被重复提交	正整数
	xaLogCleanPeriod	定时XAlog清除周期	默认1000， 单位ms	在server开始的时候会根据这个周期注册一个定时任务 （注：定时任务必定会被注册） 定时清XA log，主要是将已经回滚和提交成功的部分从记录中删除	正整数
	xaRetryCount	后台重试XA次数	默认0	后台定时任务重试XA次数，0为无限重试，达到设定次数后，停止重试	正整数
视图相关参数	viewPersistenceConfBaseDir	视图记录本地文件路径	dble目录/viewConf	用于存放视图本地记录文件的文件路径,集群配置时无意义	绝对路径
	viewPersistenceConfBaseName	视图记录本地文件名	viewJson	视图记录的文件文件名,集群配置时无意义	符合运行系统 文件的命名规范
				当行数大于此值而又没有	

队列大小参数	joinQueueSize	join时,左右结点的暂存数据行数的队列大小	1024	及时被消费者消费掉，将会阻塞，目的是防止接收数据量太大，堆积在内存中	正整数
	mergeQueueSize	merge时,左右结点的暂存数据行数的队列大小	1024	当行数大于此值而又没有及时被消费者消费掉，将会阻塞，目的是防止接收数据量太大，堆积在内存中	正整数
	orderByQueueSize	排序时，时,左右结点的暂存数据行数的队列大小	1024	当行数大于此值而又没有及时被消费者消费掉，将会阻塞，目的是防止接收数据量太大，堆积在内存中	正整数
使用Nest Loop优化	useJoinStrategy	是否使用nest loop 优化	默认不使用	开启之后会尝试判断join两边的where来重新调整查询SQL下发的顺序	true 开启 false 不开启
	nestLoopConnSize	临时表阈值	默认4	若临时表行数大于这两个值乘积，则报告错误	正整数
	nestLoopRowsSize	临时表阈值	默认2000		
慢查询日志相关配置	enableSlowLog	慢查询日志开关	默认为0，关闭	慢查询日志开关	0或者1
	slowLogBaseDir	慢查询日志存储文件夹	dble根目录/slowlogs	慢查询日志存储文件夹	文件夹路径
	slowLogBaseName	慢查询日志存储文件名前缀	slow-query	慢查询日志存储文件名前缀(后缀名是.log)	合法文件名
	flushSlowLogPeriod	日志刷盘周期，单位秒	1	日志刷盘周期，每隔这个周期，会强制将内存数据刷入磁盘	正整数
	flushSlowLogSize	日志刷盘条数阈值	1000	日志刷盘条数阈值，内存中每次写出这么多条日志，会强制刷盘1次	正整数
	sqlSlowTime	慢日志时间阈值，单位毫秒	100	慢日志时间阈值，大于此时间的查询会记录下来	正整数或者0
load data 相关配置	maxCharsPerColumn	每列所允许最大字符数	默认为65535	每列所允许最大字符数	正整数
	maxRowSizeToFile	需要持久化的最大行数	默认为10000	当load data的数据行数超过阈值后，会将数据保存在文件中以防OOM	正整数
流量控制相关参数	enableFlowControl	是否启用流量控制，true/false	默认为false	具体流量控制请参见相关功能描述章节	true/false
	flowControlStartThreshold	流量控制触发队列长度阈值	默认为4096	当部分连接的写出队列超出阈值时触发流量控制	正整数
	flowControlStopThreshold	流量控制取消队列长度阈值	默认为256	当流量控制的连接写出队列长度小于阈值取消流量控制	正整数

1.2.3 bootstrap.dynamic.cnf

有些参数是可以通过管理命令热生效的，这部分参数会写到bootstrap.dynamic.cnf中，当dble重启时候，会用内部的值替换bootstrap.cnf中对应的值，这些参数是：

enableAlert
enableSlowLog
flushSlowLogPeriod
flushSlowLogSize
sqlSlowTime
enableFlowControl
flowControlStartThreshold
flowControlStopThreshold

1.2.4 配置实例

```
#encoding=UTF-8
-agentlib:jdwp=transport=dt_socket,server=y,address=8088,suspend=n
-server
-XX:+AggressiveOpts
-Dfile.encoding=UTF-8
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=1984
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.host=127.0.0.1
-Xmx4G
-Xms1G
-XX:MaxDirectMemorySize=2G
#  base config
-DhomePath=.
-DinstanceName=1
# valid for sequenceHandlerType=2 or 3
-DinstanceId=1
-DserverId=xxx1
#-DbindIp=0.0.0.0
#-DserverPort=8066
#-DmanagerPort=9066
#-DmaxCon=1024
#-Dprocessors=4
#-DbackendProcessors=12
#-DprocessorExecutor=4
#-DbackendProcessorExecutor=12
#-DcomplexExecutor=8
#-DwriteToBackendExecutor=4


-DfakeMySQLVersion=5.7.11

# serverBacklog size,default 2048
-DserverBacklog=2048

#-DusePerformanceMode=0
# if need out HA
-DuseOuterHa=true

# connection
#-Dcharset=utf8mb4
-DmaxPacketSize=167772160
-DtxIsolation=2
#-Dautocommit=1

# option
#-DuseCompression=1
-DusingAIO=0

-DuseThreadUsageStat=1
#  query time cost statistics
#-DuseCostTimeStat=0
#-DmaxCostStatSize=100
#-DcostSamplePercent=1


# consistency
#  check the consistency of table structure between nodes,default not
-DcheckTableConsistency=0
#  check period, he default period is 60000 milliseconds
-DcheckTableConsistencyPeriod=60000

#  processor check conn
-DprocessorCheckPeriod=1000
-DsqlExecuteTimeout=3000
-DidleTimeout=1800000


#-DbackSocket unit:bytes
#-DbackSocketSoRcvbuf=4194304
#-DbackSocketSoSndbuf=1048576
#-DbackSocketNoDelay=1

#  frontSocket
#-DfrontSocketSoRcvbuf=1048576
#-DfrontSocketSoSndbuf=4194304
#-DfrontSocketNoDelay=1


#  query memory used for per session,unit is M
-DotherMemSize=4
-DorderMemSize=4
-DjoinMemSize=4


#  off Heap unit:bytes
-DbufferPoolChunkSize=32767
-DbufferPoolPageNumber=512
-DbufferPoolPageSize=2097152
#-DmappedFileSize=2097152


#  sql statistics
#  1 means use SQL statistics, 0 means not
-DuseSqlStat=1
#-DbufferUsagePercent=80
-DclearBigSQLResultSetMapMs=600000
#-DsqlRecordCount=10
#-DmaxResultSet=524288


#  transaction log
#  1 enable record the transaction log, 0 disable ,the unit of transactionRotateSize is M
-DrecordTxn=0
#-DtransactionLogBaseDir=/txlogs
```

```
#-DtransactionLogBaseName=server-tx
#-DtransactionRotateSize=16
#  XA transaction
#  use XA transaction ,if the mysql service crash,the unfinished XA commit/rollback will retry for several times , it is the check period for ,default is 1000 milliseconds
-DxaSessionCheckPeriod=1000
#  use XA transaction ,the finished XA log will removed. the default period is 1000 milliseconds
-DxaLogCleanPeriod=1000
#  XA Recovery Log path
# -DxaRecoveryLogBaseDir=/tmlogs/
#  XA Recovery Log name
#-DxaRecoveryLogBaseName=tmlog
#  XA Retry count, retry times in backend, 0 means always retry until success
#-DxaRetryCount=0

#-DviewPersistenceConfBaseDir=/viewPath
#-viewPersistenceConfBaseName=viewJson

#  for join tmp results
#-DmergeQueueSize=1024
#-DorderByQueueSize=1024
#-DjoinQueueSize=1024


#  true is use JoinStrategy, default false
#-DuseJoinStrategy=true
-DnestLoopConnSize=4
-DnestLoopRowsSize=2000


#  if enable the slow query log
-DenableSlowLog=1
#  the slow query log location
#-DslowLogBaseDir=./slowlogs
#-DslowLogBaseName=slow-query
#  the max period for flushing the slow query log from memory to disk  after last time , unit is second
-DflushSlowLogPeriod=1
#  the max records for flushing the slow query log from memory to disk after last time
-DflushSlowLogSize=1000
#  the threshold for judging if the query is slow , unit is millisecond
-DsqlSlowTime=100

#  used for load data,maxCharsPerColumn means max chars length for per column when load data
#-DmaxCharsPerColumn=65535
#  used for load data, because dble need save to disk if loading file contains large size
#-DmaxRowSizeToFile=10000

#enableFlowControl=false
#flowControlStartThreshold=4096
#flowControlStopThreshold=256
```

1.3 user.xml 配置

1.3.1 整体XML结构

- managerUser (可多值,至少一个)
- shardingUser (可多值)
- rwSplitUser (可多值)
- blacklist(可多值)

1.3.2 managerUser(管理用户配置)

配置名称	配置内容	可选项/默认值	详细作用原理或应用
name	用户名	符合mysql用户名规范的字符串	用户唯一标识，用于登录校验
password	密码		用户密码校验
usingDecrypt	是否启用加密	可配置true/false，默认false	启用加密password项配置通过执行脚本encrypt.sh 0:{user}:{password}的结果进行配置 举例: encrypt.sh 0:xxx:123456 fP/nl3XPXrSfWjpQzit5lIOrRU1QRXuLTYtATUG0fGW2k5kdXUhKL5zf02hE6nGjdnSWrufVkJPUZpbQ2qX9uQ== 配置项: password fP/nl3XPXrSfWjpQzit5lIOrRU1QRXuLTYtATUG0fGW2k5kdXUhKL5zf02hE6nGjdnSWrufVkJPUZpbQ2qX9uQ== user xxx 登录项: -u root -p123456
whiteIPs	可登录的ip的白名单	可选项	可以参考本配置文件中的whiteIPs内容
readOnly	是否是只读管理用户	可配置true/false，默认false	只读用户不能进行运维管理操作，只能进行show或者select
maxCon	负载限制，默认不做限制	正整数	用户的连接数限制,会在用户验证登录的时候进行校验，默认0，表示不做限制。 特别的，管理用户不受系统级别的maxCon的限制

1.3.3 shardingUser(分库用户配置)

配置名称	配置内容	可选项/默认值	详细作用原理或应用
name	用户名	符合mysql用户名规范的字符串	用户唯一标识，用于登录校验
password	密码		用户密码校验
usingDecrypt	是否启用加密	可配置true/false，默认false	启用加密password项配置通过执行脚本encrypt.sh 0:{user}:{password}的结果进行配置 举例： encrypt.sh 0:xxx:123456 fP/nl3XPXrSfWjpQzit5lIOrRU1QRXuLTYtATUG0fGW2k5kdXUhKL5zf02hE6nGjdnSWrufVkJPUZpbQ2qX9uQ== 配置项： password fP/nl3XPXrSfWjpQzit5lIOrRU1QRXuLTYtATUG0fGW2k5kdXUhKL5zf02hE6nGjdnSWrufVkJPUZpbQ2qX9uQ== user xxx 登录项： -u root -p123456
whiteIPs	可登录的ip的白名单	可选项	可以参考本配置文件中的whiteIPs内容
readOnly	是否是只读分库用户	可配置true/false，默认false	只读用户不能进行DML操作，只能进行show或者select
tenant	租户名	可选配置。和用户名相当于整个用户列表的联合主键	可以参考本配置文件中的tenant内容
schemas	该用户可以访问的schema列表	可配置多值，用逗号隔开	该用户可以访问的schema列表，schema参见sharding.xml中的schma名称
maxCon	负载限制，默认不做限制	正整数	用户的连接数限制,会在用户验证登录的时候进行校验，默认0，表示不做限制。 特别的，当系统级别的maxCon已经到达上限之后，本用户的maxCon会失效，不能新建连接
blacklist	blacklist的名称	可选配置	可以参考本配置文件中的blacklist内容
privileges	子元素，具体table的增删改查权限	可选配置	可以参考本配置文件中的privileges内容

1.3.3.1 user.privileges.schema

user.privileges 下的schema的dml权限，可配置多值

配置名称	配置内容	可选项/默认值	详细作用原理或应用
name	schema名称		用以标识对应schema
dml	dml权限	0000	权限判断，每一位分别表示INSERT UPDATE SELECT DELETE四种权限 1- 拥有权限 0-没有权限 例如拥有所有权限为1111
table	子元素	可配置多个	如果没有配置，则table继承schema的权限

1.3.3.2 user.privileges.schema.table

配置名称	配置内容	可选项/默认值	详细作用原理或应用
name	表格名称		在权限判断的时候作为key值
dml	dml权限	0000	权限判断，每一位分别表示INSERT UPDATE SELECT DELETE四种权限 1- 拥有权限 0-没有权限 例如拥有所有权限为1111

1.3.4 rwSplitUser(读写用户配置)

配置名称	配置内容	可选项/默认值	详细作用原理或应用
name	用户名	符合mysql用户名规范的字符串	用户唯一标识，用于登录校验
password	密码		用户密码校验
usingDecrypt	是否启用加密	可配置true/false，默认false	启用加密password项配置通过执行脚本encrypt.sh 0:{user}:{password}的结果进行配置 举例： encrypt.sh 0:xxx:123456 fP/nl3XPXrSfWjpQzit5lIOrRU1QRXuLTYtATUG0fGW2k5kdXUhKL5zf02hE6nGjdnSWrufVkJPUZpbQ2qX9uQ== 配置项： password fP/nl3XPXrSfWjpQzit5lIOrRU1QRXuLTYtATUG0fGW2k5kdXUhKL5zf02hE6nGjdnSWrufVkJPUZpbQ2qX9uQ== user xxx 登录项： -u root -p123456
whiteIPs	可登录的ip的白名单	可选项	可以参考本配置文件中的whiteIPs内容
tenant	租户名	可选配置。和用户名相当于整个用户列表的联合主键	可以参考本配置文件中的tenant内容
dbGroup	该用户对应的数据库组dbGroup	单值配置	对应db.xml中的dbGroup名称
maxCon	负载限制，默认不做限制	正整数	用户的连接数限制,会在用户验证登录的时候进行校验，默认0，表示不做限制。 特别的，当系统级别的maxCon已经到达上限之后，本用户的maxCon会失效，不能新建连接
blacklist	blacklist的名称	可选配置	可以参考本配置文件中的blacklist内容

1.3.5 blacklist(黑名单配置)

配置名称	配置内容	配置范围/可选项	详细作用原理或应用
name	blacklist的名称		用于被用户引用，标记黑名单
property	子元素，可以有多个	详细的黑名单校验规则	如果开启黑名单校验具体的校验规则将有所有property来确定

1.3.5.1 blacklist.property(详细的黑名单配置)

形式为

```
<property name="selectHavingAlwayTrueCheck">true</property>
```

下面表格来描述key和value的含义。

配置名称	配置内容	默认值	可选项	详细作用原理或应用	备注
`selectHavingAlwayTrueCheck`	是否允许复杂select having 条件结果恒为真	true	true - 允许 false -禁止	1、having部分的结果为真 2、SQL语句需要以注释结尾(不能有换行之类的) 3、条件部分不是简单SQL（单个条件、含有简单数值对等或大小比较、直接是真假值的表达式等） 符合以上三个条件的查询在校验的时候会被阻止， 举例：select * from test having id = id and hujh = hujh /*lxxddfsgdfsfdqwesfct*/;	暂不具有实际意义
`selectWhereAlwayTrueCheck`	是否允许复杂select where 条件结果恒为真	true	true - 允许 false -禁止	1、where部分的结果为真 2、SQL语句需要以注释结尾(不能有换行之类的) 3、条件部分不是简单SQL（单个条件、含有简单数值对等或大小比较、直接是真假值的表达式等） 符合以上三个条件的查询在校验的时候会被阻止， 举例：select * from test where id = id and hujh = hujh/*lxxddfsgdfsfdqwesfct*/;	暂不具有实际意义
			true - 允许		

doPrivilegedAllow	druid内部权限控制使用	false	true - 允许 false -禁止	druid内部函数调用flag，在dble中没有作用	暂不具有实际意义
wrapAllow	是否允许调用 isWrapFor和unwrap方法	true	true - 允许 false -禁止	druid内部函数调用flag，在dble中没有作用	暂不具有实际意义
metadataAllow	是否允许调用getmetadata方法	true	true - 允许 false -禁止	druid内部函数调用flag，在dble中没有作用	暂不具有实际意义
completeInsertValuesCheck	在dble依赖的1.0.31版本中没有效果	false	true - 允许 false -禁止	druid内部函数调用flag，在dble中没有作用	暂不具有实际意义
mergeAllow	是否允许merge语句 (在mysql中不支持)	true	true - 允许 false -禁止	会校验是否是merge语句	在Dble中没有效果
conditionLikeTrueAllow	是否允许like之后包含永真条件	true	true - 允许 false -禁止	会根据SQL里面的内容进行判断，如果发现有like '%'就会抛出异常	可用
conditionDoubleConstAllow	是否允许连续两个常量判断	false	true - 允许 false -禁止	会根据SQL里面的内容进行判断，如果发现有两个常量判断抛出异常 select * from suntest asdf where 1 = 1 and 2 = 1;	可用
conditionAndAlwayFalseAllow	是否允许and连接的语句存在 恒为false的条件	false	true - 允许 false -禁止	会根据where之后and跟随的条件进行判断，如果发现恒为假的情况会抛出异常 举例:select * from suntest where id = 567 and 1 ！ = 1;	可用
conditionAndAlwayTrueAllow	是否允许and连接的语句存在 恒为true的条件	false	true - 允许 false -禁止	会根据where之后and跟随的条件进行判断，如果发现恒为真的情况会抛出异常 举例:select * from suntest where id = 567 and 1 = 1;	可用
selectAllColumnAllow	是否允许查询所有列	true	true - 允许 false -禁止	会根据查询sql进行判断，如果发现直接查询*会有这个异常， 但是如果带有别名(x.*)则不在此列	存在问题
multiStatementAllow	是否允许一次提交多条sql	false	true - 允许 false -禁止	会检查sql的数量，如果超过一则抛出异常	Dble不支持
constArithmeticAllow	是否允许常量运算	true	true - 允许 false -禁止	在SQL中如果发现包含有常量运算会抛出异常 select * from suntest asdf where id = 2 -1;	可用
alterTableAllow	是否允许alter table 语句	true	true - 允许 false -禁止	在执行alter table之前会纯粹根据SQL进行语句的校验 ，发现SQL是alter table语句会抛出异常返回错误信息	可用
commitAllow	是否允许commit语句	true	true - 允许 false -禁止	在执行commit之前会纯粹根据SQL进行语句的校验， 发现commit语句会抛出异常返回错误信息	可用
createTableAllow	是否允许create table 语句	ture	true - 允许 false -禁止	在执行create table之前会纯粹根据SQL进行语句的校验， 发现SQL是create table语句会抛出异常返回错误信息	可用
deleteAllow	是否允许delete语句	true	true - 允许 false -禁止	在执行delete之前会纯粹根据SQL进行语句的校验，发现delete语句会抛出异常 返回错误信息	可用
dropTableAllow	是否允许drop table 语句	ture	true - 允许 false -禁止	在执行drop table之前会纯粹根据SQL进行语句的校验 ，发现SQL是drop table语句会抛出异常返回错误信	可用

				息	
insertAllow	是否允许insert	true	true - 允许 false -禁止	在执行insert之前会纯粹根据SQL进行语句的校验，发现insert语句会抛出异常 返回错误信息	可用
intersectAllow	是否支持intersect	true	true - 允许 false -禁止	在执行intersect之前会纯粹根据SQL进行语句的校验 ，发现intersect语句会抛出异常返回错误信息	可用
lockTableAllow	是否允许lock tables语句	true	true - 允许 false -禁止	在执行lock tables之前会纯粹根据SQL进行语句的校验 ，发现lock语句会抛出异常返回错误信息	可用
minusAllow	是否支持minus语句	true	true - 允许 false -禁止	在执行minus之前会纯粹根据SQL进行语句的校验 ，发现minus语句会抛出异常返回错误信息	可用
callAllow	是否允许call语句	true	true - 允许 false -禁止	在执行query之前会纯粹根据SQL进行语句的校验， 发现SQL是CALL语句会抛出异常返回错误信息	可用
selectIntoOutfileAllow	是否允许SELECT ... INTO OUTFILE	false	true - 允许 false -禁止	在执行query之前会纯粹根据SQL进行语句的校验，发现SQL是 SELECT ... INTO OUTFILE句会抛出异常返回错误信息	Dble本身不支持
selectIntoAllow	是否允许select into 语句	true	true - 允许 false -禁止	在执行query之前会纯粹根据SQL进行语句的校验， 发现SQL是select into语句会抛出异常返回错误信息	Dble本身不支持
selelctAllow	是否允许select语句	true	true - 允许 false -禁止	在执行query之前会纯粹根据SQL进行语句的校验， 发现SQL是select语句会抛出异常返回错误信息	可用
renameTableAllow	是否允许rename table 语句	true	true - 允许 false -禁止	在执行rename table之前会纯粹根据SQL进行语句的校验 ，发现SQL是rename table语句会抛出异常返回错误信息	可用
replaceAllow	是否允许replace语句	true	true - 允许 false -禁止	在执行replace之前会纯粹根据SQL进行语句的校验， 发现replace语句会抛出异常返回错误信息	可用
rollbackAllow	是否允许rollback	true	true - 允许 false -禁止	在执行rollback之前会纯粹根据SQL进行语句的校验， 发现rollback语句会抛出异常返回错误信息	可用 Dble慎用
setAllow	是否允许set语句	true	true - 允许 false -禁止	在执行set之前会纯粹根据SQL进行语句的校验， 发现set语句会抛出异常返回错误信息	可用
describeAllow	是否支持describe语句	true	true - 允许 false -禁止	在执行SQL之前会纯粹根据SQL进行语句的校验， 发现describe语句会抛出异常返回错误信息	可用
limitZeroAllow	是否允许出现limit 0的情况	false	true - 允许 false -禁止	在执行SQL之前会纯粹根据SQL进行语句的校验， 发现limit 0语句会抛出异常返回错误信息	可用
showAllow	是否允许show语句	true	true - 允许 false -禁止	在执行SQL之前会纯粹根据SQL进行语句的校验， 发现show语句会抛出异常返回错误信息	可用
hintAllow	是否允许sql 包含hint	true	true - 允许 false -禁止	在执行sql之前会纯粹根据SQL进行语句的校验 ，发现SQL是包含hint语句会抛出异常返回错误信息	可用

				会抛出异常返回错误信息	
`commentAllow`	是否允许在SQL中存在注释	true	true - 允许 false -禁止	在执行SQL之前会纯粹根据SQL进行语句的校验，发现注释语句会抛出异常 返回错误信息	可用
`mustParameterized`	是否必须参数化	false	true - 是 false -否	在执行SQL之前会纯粹根据SQL进行语句的校验， 发现类似 name = 'sdfasdf' ,id = 1语句会抛出异常返回错误信息	可用
`conditionOpXorAllow`	是否允许SQL中使用关系符XOR	false	true - 允许 false -禁止	在执行SQL之前会纯粹根据SQL进行语句的校验， 发现类似运算符语句会抛出异常返回错误信息	可用
`conditionOpBitwseAllow`	查询条件中是否允许有"&"、 "~"、"|"、"^"运算符。	true	true - 允许 false -禁止	在执行SQL之前会纯粹根据SQL进行语句的校验， 发现类似运算符语句会抛出异常返回错误信息	可用
`startTransactionAllow`	是否允许START TRANSACTION	true	true - 允许 false -禁止	在执行START TRANSACTION之前会纯粹根据SQL进行语句的校验， 发现START TRANSACTION语句会抛出异常返回错误信息 注:现阶段如果开启黑名单检查begin无法通过校验， 这个是由于在druid中不支持的缘故	可用
`truncateAllow`	是否允许truncate语句	true	true - 允许 false -禁止	在执行truncate之前会纯粹根据SQL进行语句的校验， 发现truncate语句会抛出异常 返回错误信息	可用
`updateAllow`	是否允许update语句	true	true - 允许 false -禁止	在执行update之前会纯粹根据SQL进行语句的校验，发现update语句会抛出异常 返回错误信息	可用
`useAllow`	是否允许use语句	true	true - 允许 false -禁止	在执行use之前会纯粹根据SQL进行语句的校验，发现use语句会抛出异常 返回错误信息	可用
`blockAllow`	是否允许语句块	`true`	true - 允许 false -禁止	在解析SQL阶段会判断SQL是否属于SQL语句块，如果是的话就会抛出错误 举例：BEGIN; select * from suntest;END;//	可用
`deleteWhereNoneCheck`	是否允许delete语句没有where条件	false	true - 启用 false -不启用	如果发现delete语句没有限定条件会抛出异常 举例：delete from suntest;	可用
`updateWhereNoneCheck`	是否允许update语句没有where条件	false	true - 启用 false -不启用	如果发现update语句没有限定条件会抛出异常 举例：update suntest set name = '33';	可用
`deleteWhereAlwayTrueCheck`	是否允许delete语句存在恒真条件	true	true - 启用 false -不启用	如果解析发现delete语句存在恒真条件,并且满足sql以注释结尾 ，并且where条件不是简单条件的，会抛出异常 举例：delete from suntest where id = id and name= name /*sdfaasdf*/;	暂不具有实际意义
`updateWhereAlayTrueCheck`	是否允许delete语句存在恒真条件	true	true - 启用 false -不启用	如果解析发现delete语句存在恒真条件，并且满足sql以注释结尾 ，并且where条件不是简单条件的，会抛出异常 举例：update suntest set name = '33' where id = id,name = name /*sdfsdf*/;	暂不具有实际意义
				如果进行校验，则不允许	

`selectIntersectCheck`	是否进行intersect check	true	true - 进行 false -不进行	当且仅当left sql的from不是空并且right from为空的时候不能通过校验 举例:select * from sbtest1 where name = 'ff' INTERSECT; select * from dual;	暂不具有实际意义
`selectExceptCheck`	是否进行except check	true	true - 进行 false -不进行	如果进行校验，则不允许except语句， 当且仅当left sql的from不是空并且right from为空的时候不能通过校验 举例： select * from sbtest1 where name = 'ff' except; select * from dual;	暂不具有实际意义
`selectMinusCheck`	是否进行MINUS check	true	true - 进行 false -不进行	如果进行校验，则不允许MINUS语句， 当且仅当left sql的from不是空并且right from为空的时候不能通过校验 举例： select * from sbtest1 where namec = 'fff' minus select * from dual;	暂不具有实际意义
`selectUnionCheck`	是否进行union check	true	true - 进行 false -不进行	如果进行校验，则不允许unoin语句 举例： select * from sbtest1 unoin select * from suntest;	可用
`caseConditionConstAllow`	是否允许复杂查询中外部是一个常量	false	true - 允许 false -禁止	是否允许复杂查询中外部是一个常量， 如果子查询外部对应的是常量那么就在SQL检查的时候抛出异常 举例:delete from suntest where id = 123 and 'name' = (select case 'fname' whe dsome' else 'good' end from xtest) /*sdfaasdf*/;	暂不具有实际意义
`strictSyntaxCheck`	是否启用严格语法检查	true	true - 是 false - 否	是否进行严格的语法检测，Druid SQL Parser在某些场景不 能覆盖所有的SQL语法，属于调试级别的参数，在正常的使用中不建议更改	暂不具有实际意义
`schemaCheck`	检测是否使用了禁用的schema	true	true - 启用 false -禁止	这个需要配合drui的配置模式使用，在dble此功能无法被使用	暂不具有实际意义
`tableCheck`	检测是否使用了禁用的table	true	true - 启用 false -禁止	这个需要配合drui的配置模式使用，在dble此功能无法被使用	暂不具有实际意义
`functionCheck`	检测是否使用了禁用的function	true	true - 启用 false -禁止	这个需要配合drui的配置模式使用，在dble此功能无法被使用	暂不具有实际意义
`objectCheck`	检测是否使用了禁用的object	true	true - 启用 false -禁止	这个需要配合drui的配置模式使用，在dble此功能无法被使用	暂不具有实际意义
`variantCheck`	检测是否使用了禁用的变量	true	true - 启用 false -禁止	这个需要配合drui的配置模式使用，在dble此功能无法被使用	暂不具有实际意义

1.3.6 tenant（租户配置）

dble支持两种方式的设置

1. 用户:租户 这种方式以:分隔开用户和租户，一起作为登录的用户 如:

```
mysql -u用户:租户 -p -h
```

```
DriverManager.getConnection("jdbc:mysql://127.0.0.1:8066", "root2:tenant1", "123456");
```

2. JDBC-connectionAttributes 在 `connectionAttributes` 中添加tenant指定租户名称 如:

```
DriverManager.getConnection("jdbc:mysql://127.0.0.1:8066?connectionAttributes=tenant:tenant1", "root2", "123456");
```

以上两种方式中，若同时设置了*1/2*两种方式，则采用方式*1*的配置

1.3.7 whiteIPs（IP白名单）

默认不限制，值为IP，多个用逗号隔开

格式：

支持用户输入多ip，如192.168.1.2,192.168.2.22
支持用户输入IP段，如192.168.1.10-192.168.1.100
支持用户输入通配符，如192.168.1.%
支持用户输入IP/CIDR格式，如192.168.1.1/20

以上格式同样适合*IPV4/IPV6*

注：

- 管理员用户一旦配置该项，默认允许本机（127.0.0.1、0:0:0:0:0:0:0:1）登陆
- IPV6格式中不支持IPv4映射

1.3.8 完整例子

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dble:user SYSTEM "user.dtd">
<dble:user xmlns:dble="http://dble.cloud/">
    <managerUser name="man1" password="654321" whiteIPs="127.0.0.1,0:0:0:0:0:0:0:1" readOnly="false"/>
    <managerUser name="user" usingDecrypt="true" readOnly="true" password="AqEkFEuIFAX6g2TJQnp4cJ2r7Yc0Z4/KBsZqKhT8qSz18Aj91e8lxO49BKQElC6OFfW4c38pCYa8QGFTub7pnw==" />

    <shardingUser name="root" password="123456" schemas="testdb" readOnly="false" blacklist="blacklist1" maxCon="20"/>
    <shardingUser name="root2" password="123456" schemas="testdb,testdb2" maxCon="20" tenant="tenant1">
        <privileges check="true">
            <schema name="testdb" dml="0110">
                <table name="tb01" dml="0000"/>
                <table name="tb02" dml="1111"/>
            </schema>
        </privileges>
    </shardingUser>
    <!--rwSplitUser not work for now-->
    <rwSplitUser name="rwsu1" password="123456" dbGroup="dbGroup1" blacklist="blacklist1"
                 maxCon="20"/>
    <blacklist name="blacklist1">
        <property name="selelctAllow">true</property>
    </blacklist>
</dble:user>
```

1.4 db.xml

db.xml包含具体的数据库组和实例配置,可以配置多组，每组可以配置多个实例

1.4.1 dbGroup配置

- dbGroup

配置名称	配置内容&示例	可选项/默认值	详细描述
name	节点名称	必需项，无默认值	dbGroup的唯一标识，不允许重复
rwSplitMode	读操作的负载均衡模式	必需项，无默认值，候选值0/1/2	在进行读负载均衡的时候会根据这个配置进行 0：不做均衡，直接分发到主实例，从实例将被忽略,不会尝试建立连接池，但会有心跳连接 1：读操作在所有从实例中均衡，当所有从实例都不可用时，下发语句会报错，而之前的版本是会发往主实例中的。 2：读操作在所有实例中均衡。具体拓扑结构见负载均衡相关章节
delayThreshold	指定主从延迟阀值	默认-1，表示无延迟	1:在进行读取负载均衡的时候会根据最近一次的心跳状态以及读库和主库的延迟进行判断，如果延迟超过delayThreshold配置，则认为此节点不适合进行读取，依赖于心跳为show slave status 2:此配置会影响到进行读负载均衡的时候延迟检测的开启，如果delayThreshold=-1那么读负载均衡选取的时候不会进行延迟检测
disableHA	是否禁用该组的高可用切换	可配置true/false，默认false	禁用该组的高可用切换，可能在某些场景下需要
heartbeat	子元素,心跳语句	必选项	该配置会在服务启动时设置的心跳任务里面被使用到，用于进行mysql实例状态的判断. 该配置有以下几种建议值: 1.普通心跳只是用于探活，建议使用select 1 2.使用 select @@read_only 探测结点可用性以及可写性 3.使用show slave status,可以探活，检查复制是否正常，以及延迟检测。如果Seconds_Behind_Master返回的状态有延迟，那么会被记入mysql实例的主从延迟中，影响读请求的路由分发，延迟超过指定限制读写分离会变为只读主库。
dbInstance	子元素，表示组下的实例，可配置多个	空	具体的物理节点配置

- heartbeat

配置名称	配置内容&示例	可选项/默认值	详细描述
timeout	heartbeat子元素,心跳超时阈值，单位：秒	默认0	心跳超时阈值。前置知识：dble会按照heartbeatPeriodMillis 的间隔向 dbInstance 发送心跳 心跳发起时候检会查上次心跳是否为不正常的心跳,如果上次心跳尚未返回，并且距离最近的正常心跳的时间大于timeout，则标记该结点不可达。 例如:心跳周期(heartbeatPeriodMillis)2秒，第一次心跳正常，2s后的第二次心跳未返回，4s后第三次心跳发起时候发现上次不正常，不会真的再次下发，而是会根据4s和timeout的大小来确定该节点是否真正超时(该节点使用时候才会真正用到） 如果未超时，则什么也不做，继续下一个周期。如果超时了，则尝试杀掉超时的连接，无论是否杀成功，都会在下一个周期换一个连接继续做心跳，极端情况下会消耗很多连接 2. 心跳连续返回失败后，dble使用该结点时，会检查距离第一次失败的时间差，如果大于timeout，则报该结点不可达。
errorRetryCount	heartbeat子元素,心跳失败后的尝试次数	默认1，0表示不重试	心跳失败/心跳连接被关闭后,开始重试errorRetryCount次。 1. 心跳失败重试期间，连接池状态为error状态，一旦重试成功，则标记回OK。(目的：防止网络抖动或者连接异常断开场景) 2. 心跳连接被关闭重试期间，连接池状态为ok状态，只有都重试失败后才会置为error状态 3. 重试期间超时，按照超时逻辑处理。

- dbInstance

配置名称	配置内容&示例	可选项/默认值	详细描述
name	写节点名称	空	节点名称作为标识
url	写节点地址ip:port	空	被分成IP和PORT用于连接数据库
user	写节点用户	空	用于连接数据库
password	写节点用户密码	空	用于连接数据库
usingDecrypt	是否启用加密password	候选值false/true,默认值false	如果设置为true，password属性值应该为用工具encrypt.sh加密串 1:{name}:{user}:{password} 得到的串
minCon	空闲时保有最小连接数	必需项，无默认值	后端连接池中空闲时维持的最小连接数量
maxCon	最大连接数	否	后端连接池最大允许的连接数量
readWeight	节点权重(负载均衡时候使用)	默认0	负载均衡过程中会查看所有节点的权重是否相等，如果不相等，那么就会根据权重来配置压力
primary	主实例需要配置成true	false	用来标识主节点
disabled	标记改实例不可用	false	高可用切换时候可能会用到
property	连接池属性	空	具体的后端连接池属性

- property

  后端连接池属性

举例如下:

```
<?xml version="1.0"?>
<!DOCTYPE dble:db SYSTEM "db.dtd">
<dble:db xmlns:dble="http://dble.cloud/">
    <dbGroup name="dbGroup1" rwSplitMode="2" delayThreshold="100">
        <heartbeat>show slave status</heartbeat>
        <dbInstance name="instanceM1" url="ip1:3306" user="your_user" password="your_psw" maxCon="1000" minCon="10"
                    primary="true" readWeight="1" id="xx1">
            <property name="lifo">true</property>
        </dbInstance>
        <dbInstance name="instanceS1" url="ip2:3306" user="your_user" password="your_psw" maxCon="1000" minCon="10"  readWeight="2" disabled="true">
            <property name="testOnCreate">false</property>
        </dbInstance>
    </dbGroup>

    <dbGroup name="dbGroup2" rwSplitMode="0" delayThreshold="100" disableHA="true">
        <heartbeat errorRetryCount="1" timeout="10">show slave status</heartbeat>
        <dbInstance name="instanceM2" url="ip4:3306" user="your_user" password="your_psw" maxCon="1000" minCon="10"
                    primary="true">
            <property name="testOnCreate">true</property>
        </dbInstance>

        <!-- can have multi read instances -->
        <dbInstance name="instanceS2" url="ip5:3306" user="your_user" password="your_psw" maxCon="1000" minCon="10" usingDecrypt="true">
            <property name="testOnCreate">true</property>
        </dbInstance>
    </dbGroup>
</dble:db>
```

1.4.2 MySQL用户权限说明

权限项目	作用
SELECT	数据查询权限
INSERT	新增数据权限
UPDATE	更新数据权限
DELETE	删除数据权限
FILE	load data等数据导出导入的权限
CREATE	建库(管理端)，建表，建索引权限
DROP	删除表的权限
ALTER	变更表结构的权限
LOCK TABLES	lock tables的权限
ALTER ROUTINE	(hint)变更/删除存储过程的权限
CREATE ROUTINE	(hint)创建存储过程的权限
EXECUTE	(hint)执行存储过程的权限
INDEX	建立/删除索引权限
SUPER	用于KILL功能
SHOW DATABASES	部分GUI工具会使用INFORMATION_SCHEMA SCHEMATA表
PROCESS	用于管理端show processlist 功能
REPLICATION CLIENT	可能场景: 1.用于配置了主从关系的数据结点,使用了读写分离 2.使用show slave status作为心跳 3.需要使用show @@binlog_status功能
REFERENCES	外键约束(纯语法支持，无使用意义)

1.5 sharding.xml 配置

1.5.1 整体XML结构

- schema (虚拟schema，可配置多个)
- shardingNode (虚拟分片，可配置多个)
- function (拆分算法，可配置多个)

1.5.2 schema

1.5.3 schema配置

- schema

配置名称	配置内容	可选项/默认值	详细描述
name	schema名称		schema的唯一标识，不允许重复
shardingNode	涉及的数据点	缺省无，最多一个	未配置则只加载xml中的table子节点; 若配置后：1.table子节点的配置会覆盖schema中的配置，2.物理schema下存在的,并且不在配置内的table被视为single node table
sqlMaxLimit	最大返回结果集限制	默认-1	当且仅当查询的SQL符合下列条件时，产生效果 1 整个查询属于单表查询，包括简单查询以及(order/group/聚合函数) 2 表格对应的shema需要有对应的sqlMaxLimit配置
shardingTable	子属性，拆分表，详见shardingTable配置选项	可配置多个	每个表格的详细配置信息
globalTable	子属性，全局表，详见globalTable配置选项	可配置多个	每个表格的详细配置信息
singleTable	子属性，单节点表，详见singleTable配置选项	可配置多个	每个表格的详细配置信息

- shardingTable

配置名称	配置内容	可选项/默认值	详细描述
name	表格名称	必须项	表名，可以配置多个使用’,’分割
shardingNode	表格涉及的数据节点	必须项	可使用通配符: xxx$n0-n1 此种格式指定xxxn0, ..., xxxnm, ..., xxxn1作为该表的数据节点。 dn$1-6 等价于 dn1,dn2, dn3,dn4,dn5,dn6
shardingColumn	拆分列名	必须项	标记表格的拆分列名
function	使用的拆分规则	必须项	引用function节中的拆分规则名称那个
incrementColumn	表格自增列	可选项，默认无	指定表格自增列，指定自增列的表格会使用全局序列
sqlMaxLimit	最大返回结果集限制	可选项，继承父级schema的sqlMaxLimit	与schemas 中的对应配置效果相同，但是覆盖schema中配置
sqlRequiredSharding	是否要求sql中包含拆分列条件	默认false	如果是true，sql中未包含拆分列条件，返回报错
childTable	关联子表信息，详见childTable选项	可配置多个	路由是通过父子关系进行ER关联

- childTable

配置名称	配置内容	可选项/默认值	详细描述
name	表格名称	必需项	表名，可以配置多个使用’,’分割
joinColumn	指定同父表进行join操作时的join键	必需项	子表和父表关联的字段
parentColumn	指定进行join操作时父表中的join键	必需项	如果父表为非子表，在父表中该字段必须与其拆分规则/拆分键有对等关系。
incrementColumn	表格自增列	默认空	显式指定表格自增列
sqlMaxLimit	最大返回结果集限制	可选项，继承父级schema的sqlMaxLimit	与schemas 中的对应配置效果相同，但是覆盖schema中配置
childTable	关联子表信息，详见childTable选项	可配置多个	路由是通过父子关系进行ER关联

- globalTable

配置名称	配置内容	可选项/默认值	详细描述
name	表格名称	必须项	表名，可以配置多个使用’,’分割
shardingNode	表格涉及的数据节点	必须项	可使用通配符: xxx$n0-n1 此种格式指定xxxn0, ..., xxxnm, ..., xxxn1作为该表的数据节点。 dn$1-6 等价于 dn1,dn2, dn3,dn4,dn5,dn6
sqlMaxLimit	最大返回结果集限制	可选项，继承父级schema的sqlMaxLimit	与schemas 中的对应配置效果相同，但是覆盖schema中配置
checkClass	全局表检查类	可选项	全局表检查自定义类名或者是缩写 dble自带CHECKSUM和COUNT两种默认实现
cron	全局表一致性检查周期	0 0 0 * * ?	quartz定时任务时间设置 详见：http://www.quartz-scheduler.org/api/2.4.0-SNAPSHOT/org/quartz/CronScheduleBuilder.html

- singleTable

配置名称	配置内容	可选项/默认值	详细描述
name	表格名称	必须项	表名，可以配置多个使用’,’分割
shardingNode	表格涉及的数据节点	必需项	唯一的shardingNode结点，配置多个会报错
sqlMaxLimit	最大返回结果集限制	可选项，继承父级schema的sqlMaxLimit	与schemas 中的对应配置效果相同，但是覆盖schema中配置

1.5.4 shardingNode配置

- shardingNode

配置名称	配置内容&示例	详细描述
name	数据节点名称，唯一，例如"dn,dn$0-5"	作为数据节点的标识以及键,节点个数的计算方法为：从值出发,以‘,’（逗号）分隔字符串,如果其中有连续几项拥有相同的字符串前缀X(不能为空)并且后续其他几位为连续的数字时（比如0到5）,可以以"X$0-5"来省略表示,个数为：以逗号分隔的字符串个数加上包含$的连续个数。name的个数必须等于database与dbGroup的个数之积。
database	shardingNode对应的存在于mysql物理实例中的schema,可以配置单个或多个使用，例如"db,db$0-5"	所使用的详细数据库节点,节点个数的计算方法为：从值出发,以‘,’（逗号）分隔字符串,如果其中有连续几项拥有相同的字符串前缀X(不能为空)并且后续其他几位为连续的数字时（比如0到5）,可以以"X$0-5"来省略表示,个数为：以逗号分隔的字符串个数加上包含$的连续个数。
dbGroup	shardingNode对应的数据库组，参考db.xml中的dbGroup名称,可以配置单个或多个使用,例如"dh,dh$0-5"	用于关联对应的Host节点,节点个数的计算方法为：从值出发，以‘,’（逗号）分隔字符串，如果其中有连续几项拥有相同的字符串前缀X(不能为空)并且后续其他几位为连续的数字时（比如0到5）,可以以"X$0-5"来省略表示,个数为：以逗号分隔的字符串个数加上包含$的连续个数。

例如：

```
<shardingNode name="dn1" dbGroup="localhost1" database="db1" />
```

name, dbGroup, database均可用如下格式在单个配置中配置多个节点： xxx$n0-n1, xxx, 这种格式的意义为：xxxn0， ... , xxxnm， ... ,xxxn1, xxx, 其中

```
n0 < nm < n1。
```

例如： 配置

```
<shardingNode name="dn1$0-19" dbGroup="localhost1$0-9" database="db1$0-1" />
```

等同于:

```
<shardingNode name="dn10" dbGroup="localhost10" database="db10" />
<shardingNode name="dn11" dbGroup="localhost10" database="db11" />
<shardingNode name="dn12" dbGroup="localhost11" database="db10" />
<shardingNode name="dn13" dbGroup="localhost11" database="db11" />
   ...
<shardingNode name="dn119" dbGroup="localhost19" database="db11" />
```

注意：如果是使用通配符的配置，那么shardingNode(name)的通配符展开个数必须等于dbGroup通配符展开个数与database通配符展开个数之积,上例中,name的个数为20,dbGroup的个数为10,database的个数为2; 又例如

```
<shardingNode name="dn,dn1$0-19,dnx" dbGroup="localhost,localhost1$0-9" database="db1$0-1" />
```

中,name的个数为22,dbGroup的个数为11,database的个数为2。

注意：若出现两个不同的shardingNode拥有同样的database以及dbGroup，在配置检查的时候会报错(包括从通配符批量生成的shardingNode)

1.5.5 function配置

拆分算法定义有如下形式：

name: 定义分区算法名， 在分区规则定义中被引用。 class：指定分区算法实现类。 每一种分区算法要求的参数个数， 类型各不相同，property name部分用于指定相应分区算法的参数。这部

分请参考各分区算法描述。

- function

配置名称	配置内容	说明
name	函数的名称	在分区规则定义中被引用
class	拆分算法	只能是Enum,NumberRange,Hash,StringHash,Date,PatternRange,jumpStringHash之一
property	根据具体的function代码示例的属性进行配置参数	可以配置多个属性

举例:

```
<function name="rang-long""  class="com.actiontech.dble.route.function.AutoPartitionByLong">
         <property name="mapFile">auto-sharding-long.txt</property>
         ...
</function>
```

支持的分区算法: 目前，已支持的分区算法有: hash, stringhash, enum, numberrange, patternrange, date，jumpstringhash.

1.5.5.1.hash分区算法

function的 class属性设置为“hash”或者“com.actiontech.dble.route.function.PartitionByLong"的分区规则应用该算法。具体配置如下:

```
<function name="hashLong" class="hash">
      <property name="partitionCount">C1[,C2, ...Cn]</property>
      <property name="partitionLength">L1[,L2, ...Ln]</property>
</function>
```

partitionCount:指定分区的区间数， 具体为 C1 [+C2 + ... + Cn].

partitionLength:指定各区间长度， 具体区间划分为 [0, L1), [L1, 2*L1), ..., [(C1-1)*L1, C1*L1), [C1*L1+L2, C1*L1+2*L2), ... 其中，每一个区间对应一个数据节点。

例如，配置F1:

```
<property name="partitionCount">2,3</property>
<property name="partitionLength">100,50</property>
```

将划分如下的分区:
[0 , 100) [100, 200) [200, 250) [250, 300) [300, 350)

再如,配置F2:

```
<property name="partitionCount">2</property>
<property name="partitionLength">1000</property>
```

将划分如下的分区: [0 , 1000) [1000, 2000)

根据具体配置， 模的基数M有如下计算公式: C1*L1* + ... + *Cn* Ln. 上面的例子中F1 的M值为350，F2的M值为2000。 在进行分片查找时， 将分区字段key和M值进行求模运算: value = key mod M 得到的value值再从区间分布中找到自己数据节点的序号。 例如，当配置为F1,key =805 时,value = 105,那么从5个区间内发现对应的数据节点的序号为1(从0开始)。

结点的个数N 记为 C1 [+C2 + ... + Cn]. 上面的例子中F1 的N值为5，F2的N值为2。

注意事项:

1. M不能大于2880。2880的原因是这样的:2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 30, 32, 36, 40, 45, 48, 60, 64, 72, 80, 90, 96, 120, 144, 160, 180, 192, 240, 288, 320, 360, 480, 576, 720, 960, 1440是2880的约数,这样预分片扩容方便。
2. N必须要等于逻辑表的shardingNode属性指定的shardingNode数量之和，如shardingNode="dn1,dn2,dn3,dn4"中，N必须等于4。
3. Cn和Ln的个数必须相等。
4. 分区字段必须为整型字段，如果是其他类型，要求值可转化为数字。
5. 当partitionLength为1时，hash分区算法退化为求模算法，M及N均为partitionCount的值。
6. NULL作为分片列的值的时候数据的结果恒落在0号节点（第一个节点上）

1.5.5.2.stringhash分区算法

class属性设置为“stringhash”或者“com.actiontech.dble.route.function.PartitionByString"的分区规则应用该算法。具体配置如下:

```
<function name="hashSting" class="stringhash">
       <property name="partitionCount">C1[,C2, ...Cn]</property>
       <property name="partitionLength">L1[,L2, ...Ln]</property>
       <property name="hashSlice">l:r</property>
</function>
```

partitionCount，**partitionLength**的具体意义参看hash分区算法。 **hashSlice**: 指定参与hash值计算的key的子串。字符串从0开始索引计数。

子串截取有如下计算步骤和格式:

步骤1. 子串索引区间确定

格式	条件	区间
n	n>=0	(0,n)
n	n<0	(n,0)
:r		(0,r)
l:		(l,0)
:		(0:0)
l:r		(l, r)

步骤2. 子串索引区间修订

a.左边界l修订

第一次修订			第二次修订	
修订前	条件	修订后	条件	修订后
l	l>=0	l		l
l	l<0	l=l+length	l<0	l=0

b.右边界r修订

第一次修订			第二次修订	
修订前	条件	修订后	条件	修订后
r	r>0	r	r>length	r=length
r	r<=0	r=r+length		r

注：*length*是分区字段实际串的长度.

步骤3. 结果确定

条件	结果子串	**hash**值
l<r	索引在[l, r)范围的子串	通过子串计算的hash值
l>=r	空字串	0

这个子串截取算法只是看起来比较复杂。用简洁但不太准确的语言可描述为：区间边界为负值的，从分区字段串尾部开始计数；区间边界为正值的，从分区字段串首部开始计数；然后做一些纠错处理。

该算法与hash分区算法类似，但针对的分区字段为字符串类型。在进行分片查找时， 要先对分区字段求hash值，然后将得到的hash值做hash分区算法的分区查找运算。

注意事项:

1. 该分区算法和hash分区算法有同样的限制(注意事项3除外）
2. 分区字段为字符串类型。

算法解析:
stringhash 按照用户定义的起点和终点去截取分片索引字段中的部分字符，根据当中每个字符的二进制 unicode 值换算出一个长整型数值，然后就直接调用内置 hash 算法求解分片路由：先求模得到逻辑分片号，再根据逻辑分片号直接映射到物理分片。

- 用户需要定义 partitionLength[] 和 partitionCount[] 两个数组和 hashSlice 二元组。
- 在 DBLE 的启动阶段，点乘两个数组得到模数，也是逻辑分片的数量。
- 并且根据两个数组的叉乘，得到各个逻辑分片到物理分片的映射表（物理分片数量由 partitionCount[] 数组的元素值之和）。
- 此外根据 hashSlice 二元组，约定把分片索引值中的第 4 字符到第 5 字符（字符串以 0 开始编号，编号 3 到编号 4 等于第 4 字符到第 5 字符）字符串用于 “字符串->整型”的转换。
- 在 DBLE 的运行过程中，用户访问使用这个算法的表时，WHERE 子句中的分片索引值会被提取出来，取当中的第 4 个字符到第 5 字符，送入下一步。
- 设置一个初始值为 0 的累计值，逐个取字符，把累计值乘以 31，再把这个字符的 unicode 值当成长整型加入到累计值中，如此类推直至处理完截取出来的所有字符，此时的累计值就能够代表用户的分片索引值，完成了 “字符串->整型” 的转换。
- 对上一步的累计值进行求模，得到逻辑分片号。
- 再根据逻辑分片号，查映射表，直接得到物理分片号。

1.5.5.3.enum分区算法

class属性设置为“enum”或者“com.actiontech.dble.route.function.PartitionByFileMap"的分区规则应用该算法。具体配置如下：

```
<function name="enum" class="enum">
       <property name="mapFile">partition.txt</property>
       <property name="defaultNode">0</property>
       <property name="type">0</property>
</function>
```

mapFile:指定配置文件名。其格式将在下面做详细说明。 **defaultNode**：指定默认节点号。默认值为-1， 不指定默认节点。 **type**：指定配置文件中key的类型。0：整型； 其它：字符串。

配置文件格式如下： a. type值为0时，

```
#comment //comment this line will be skiped
int1=node0
int2=node1
...
```

b. type值为非0时，

```
#comment //comment this line will be skiped
string1=node0
string2=node1
...
```

在进行分片查找时，分片字段的枚举值对应的数据节点既是目的节点。如果不能在配置映射中找到枚举值对应的数据节点：如果配置了defaultNode，则defaultNode既是目的数据节点， 否则出错。

注意事项：

1. 不包含“=”的行将被跳过.
2. nodex为数据节点索引号。
3. 重复的枚举值的分区数据节点以最后一个配置为准。
4. 分片字段为该枚举类型。
5. 分片字段为NULL时，数据落在defaultNode节点上，若此时defaultNode没有配置，则会报错;当真实存在于mysql的字段值为not null的时候,报错 "Sharding column can't be null when the table in MySQL column is not null"

1.5.5.4.numberrange分区算法

class属性设置为“numberrange”或者“com.actiontech.dble.route.function.AutoPartitionByLong"的分区规则应用该算法。具体配置如下：

```
<function name="rangeLong" class="numberrange">
       <property name="mapFile">partition.txt</property>
       <property name="defaultNode">0</property>
</function>
```

mapFile:指定配置文件名。其格式将在下面做详细说明。 **defaultNode**：指定默认节点号。默认值为-1， 不指定默认节点。

配置文件格式如下： #comment //comment this line will be skiped start1-end1=node1 start2-end2=node2 ...

该分区算法相当于定义了区间序列 [start1, end1], [start2, end2], ... 每一个区间对应一个数据节点。 在进行分片查找时，分片字段的值落在的区间对应的数据节点即是目的节点。 如果不能在配置映射中找到分片字段的值所落的区间时，分两种情况：1.配置了defaultNode，则defaultNode是目的数据节点；2.没配defaultNode，出错。 注意事项：

1. 不包含“=”的行将被跳过。
2. nodex为数据节点索引号。
3. 如果区间存在重合， 在对重合部分的分片字段值进行分片查找时在配置文件中最先定义的区间对应的数据节点为目的节点。
4. 分片字段为整型。
5. 分片字段为NULL时，数据落在defaultNode节点上，若此时defaultNode没有配置，则会报错;当真实存在于mysql的字段值为not null的时候,报错 "Sharding column can't be null when the table in MySQL column is not null"

1.5.5.5.patternrange分区算法

class属性设置为“patternrange”或者“com.actiontech.dble.route.function.PartitionByPattern"的分区规则应用该算法。具体配置如下：

```
<function name="pattern" class="patternrange">
       <property name="mapFile">partition.txt</property>
       <property name="patternValue">1024</property>
       <property name="defaultNode">0</property>
</function>
```

mapFile:指定配置文件名。其格式将在下面做详细说明。 **patternValue**:指定模值， 默认为1024。 **defaultNode**：指定默认节点号。默认值为-1， 不指定默认节点。

配置文件格式如下： #comment //comment this line will be skiped start1-end1=node1 start1-end2=node2 ...

该分区算法类似于numberrange分区算法。但在进行分片查找时先对分片字段值进行模patternValue运算，然后将得到的模数进行等同于numberrange分区算法的分区查找。 注意事项：

1. 不包含“=”的行将被跳过。
2. nodex为数据节点索引号。
3. 如果区间存在重合， 在对重合部分的分片字段值进行分片查找时在配置文件中最先定义的区间对应的数据节点为目的节点。
4. 分片字段的内容必须可以转化为整数。如果不能转化为整数：如果配置了defaultNode， 目的数据节点为defaultNode； 否则， 出错。
5. 分片字段为NULL时，数据落在defaultNode节点上，若此时defaultNode没有配置，则会报错;当真实存在于mysql的字段值为not null的时候,报错 "Sharding column can't be null when the table in

MySQL column is not null"

1.5.5.6.date分区算法

class属性设置为“date”或者“com.actiontech.dble.route.function.PartitionByDate"的分区规则应用该算法。具体配置如下：

```
<function name="partbydate" class="date">
       <property name="dateFormat">yyyy-MM-dd</property>
       <property name="sBeginDate">2015-01-01</property>
       [<property name="sEndDate">2015-01-31</property>]
       <property name="sPartionDay">10</property>
       <property name="defaultNode">0</property>
</function>
```

dateFormat:指定日期的格式。 **sBeginDate**：指定日期的开始时间。 **sEndDate**：指定日期的结束时间。该性质可以不配置或配置为空("")。 **sPartionDay**：指定分区的间隔，单位是天。 **defaultNode**：指定默认节点号。默认值为-1， 不指定默认节点。

该算法有两种工作模式： 模式**1**：不配置sEndDate或者将sEndDate配置为"" 在这种模式下， 该算法将时间以sBeginDate为开始， 以sPartionDay为间隔，进行区间划分， 每个区间对应一个数据节点。 在进行区间查找时，分区字段值落在的区间对应的数据节点即是目的数据节点。如果分区字段值小于sBeginDate：如果配置了defaultNode，则数据会被路由至defaultNode；如果没配置defaultNode,则会报错。 模式**2**：配置sEndDate且不为"" 在这种模式下， 该算法将时间以sBeginDate为开始， 以sPartionDay为间隔，以sEndDate为终点， 进行区间划分，划分为N个区间， 每个区间对应一个数据节点。 在进行区间查找时： 如果分区字段值小于等于sEndDate，则计算过程和结果等同于模式1。如果分区字段值大sEndDate，则分片查找公式为：index=((key - sBeginDate)/sPartionDay)%N, 其中key为分片字段值， index为映射到的数据节点索引。这等同于一个环形映射。 如果分区字段值小于sBeginDate，则会检查是否设置了defaultNode。如果设置了，数据会路由至defaultNode;否则报错。

注意事项:

1. 分片字段必须是符合dateFormat的日期字符串。
2. 区间划分不以日历时间为准，无法对应到日历时间。内部进行区间划分时，会将sPartionDay转化为以86400000毫秒为一天进行运算。
3. 在模式2的情况下， 如果(sEndDate - sBeginDate)不是sPartionDay的整数倍，则索引号为0的数据节点承载更多的数据。
4. 分片字段为NULL时，数据落在defaultNode节点上，若此时defaultNode没有配置，则会报错;当真实存在于mysql的字段值为not null的时候,报错 "Sharding column can't be null when the table in MySQL column is not null"

1.5.5.7.跳增字符串算法

class属性设置为"jumpstringhash"或者"com.actiontech.dble.route.function.PartitionByJumpConsistentHash"的分区规则应用该算法，具体配置如下

```
<function name="jumphash"
              class="jumpStringHash">
        <property name="partitionCount">2</property>
        <property name="hashSlice">0:2</property>
</function>
```

partitionCount:分片数量 **hashSlice**:分片截取长度 该算法来自于Google的一篇文章A Fast, Minimal Memory, Consistent Hash Algorithm其核心思想是通过概率分布的方法将一个hash值在每个节点分布的概率变成1/n，并且可以通过更简便的方法可以计算得出，而且分布也更加均匀

注意事项:

1. 分片字段值为NULL时，数据恒落在0号节点之上;当真实存在于mysql的字段值为not null的时候,报错 "Sharding column can't be null when the table in MySQL column is not null"

1.5.6 完整配置举例

举例：

```
<?xml version="1.0"?>
<!--
  ~ Copyright (C) 2016-2020 ActionTech.
  ~ License: http://www.gnu.org/licenses/gpl.html GPL version 2 or higher.
  -->

<!DOCTYPE dble:sharding SYSTEM "sharding.dtd">
<dble:sharding xmlns:dble="http://dble.cloud/" version="4.0">

    <schema name="testdb"  sqlMaxLimit="100">
        <shardingTable name="tb_enum_sharding" shardingNode="dn1,dn2" sqlMaxLimit="200" function="func_enum" shardingColumn="code"/>
        <shardingTable name="tb_range_sharding" shardingNode="dn1,dn2,dn3" function="func_range" shardingColumn="id"/>
        <!--er tables-->
        <shardingTable name="tb_hash_sharding" shardingNode="dn1,dn2" function="func_common_hash" shardingColumn="id"/>
        <shardingTable name="tb_hash_sharding_er1" shardingNode="dn1,dn2" function="func_common_hash" shardingColumn="id"/>
        <shardingTable name="tb_hash_sharding_er2" shardingNode="dn1,dn2" function="func_common_hash" shardingColumn="id2"/>
        <shardingTable name="tb_hash_sharding_er3" shardingNode="dn1,dn2" function="func_common_hash" shardingColumn="id" incrementColumn="id2"/>

        <shardingTable name="tb_uneven_hash" shardingNode="dn1,dn2,dn3" function="func_uneven_hash" shardingColumn="id"/>

        <shardingTable name="tb_mod" shardingNode="dn1,dn2,dn3,dn4" function="func_mod" shardingColumn="id" sqlRequiredSharding="true"/>

        <shardingTable name="tb_jump_hash" shardingNode="dn1,dn2" function="func_jumpHash" shardingColumn="code"/>

        <shardingTable name="tb_hash_string" shardingNode="dn1,dn2,dn3,dn4" function="func_hashString" shardingColumn="code"/>

        <shardingTable name="tb_date" shardingNode="dn1,dn2,dn3,dn4" function="func_date" shardingColumn="create_date"/>

        <shardingTable name="tb_pattern" shardingNode="dn1,dn2" function="func_pattern" shardingColumn="id"/>
        <!--global  tables-->
        <globalTable name="tb_global1" shardingNode="dn1,dn2" sqlMaxLimit="103" />
        <globalTable name="tb_global2" shardingNode="dn1,dn2,dn3,dn4" cron="0 0 0 * * ?" checkClass="CHECKSUM"/>
        <!--single node table-->
        <singleTable name="tb_single" shardingNode="dn6" sqlMaxLimit="105"/>
        <!--er tables-->
        <shardingTable name="tb_parent" shardingNode="dn1,dn2" function="func_common_hash" shardingColumn="id">
            <childTable name="tb_child1" joinColumn="child1_id" parentColumn="id" sqlMaxLimit="201">
```

```
                <childTable name="tb_grandson1" joinColumn="grandson1_id" parentColumn="child1_id"/>
                <childTable name="tb_grandson2" joinColumn="grandson2_id" parentColumn="child1_id2"/>
            </childTable>
            <childTable name="tb_child2" joinColumn="child2_id" parentColumn="id"/>
            <childTable name="tb_child3" joinColumn="child3_id" parentColumn="id2"/>
        </shardingTable>
    </schema>
    <!-- sharding testdb2 route to database named dn5 in localhost2  -->
    <schema name="testdb2" shardingNode="dn5"/>
    <shardingNode name="dn1" dbGroup="dbGroup1" database="db_1"/>
    <shardingNode name="dn2" dbGroup="dbGroup2" database="db_2"/>
    <shardingNode name="dn3" dbGroup="dbGroup1" database="db_3"/>
    <shardingNode name="dn4" dbGroup="dbGroup2" database="db_4"/>
    <shardingNode name="dn5" dbGroup="dbGroup1" database="db_5"/>
    <shardingNode name="dn6" dbGroup="dbGroup2" database="db_6"/>
    <!-- enum partition -->
    <function name="func_enum" class="Enum">
        <property name="mapFile">partition-enum.txt</property>
        <property name="defaultNode">0</property><!--the default is -1,means unexpected value will report error-->
        <property name="type">0</property><!--0 means key is a number, 1 means key is a string-->
    </function>
    <!-- number range partition -->
    <function name="func_range" class="NumberRange">
        <property name="mapFile">partition-number-range.txt</property>
        <property name="defaultNode">0</property><!--he default is -1,means unexpected value will report error-->
    </function>
    <!-- Hash partition,when partitionLength=1, it is a mod partition, MAX(sum(count*length[i]) must not more then 2880-->
    <function name="func_common_hash" class="Hash">
        <property name="partitionCount">2</property>
        <property name="partitionLength">512</property>
    </function>
    <!-- Hash partition,when partitionLength=1, it is a mod partition, MAX(sum(count*length[i]) must not more then 2880-->
    <function name="func_uneven_hash" class="Hash">
        <property name="partitionCount">2,1</property>
        <property name="partitionLength">256,512</property>
    </function>
    <!-- eg:  mod 4 -->
    <function name="func_mod" class="Hash">
        <property name="partitionCount">4</property>
        <property name="partitionLength">1</property>
    </function>
    <!-- jumpStringHash partition for string-->
    <function name="func_jumpHash" class="jumpStringHash">
        <property name="partitionCount">2</property>
        <property name="hashSlice">0:2</property>
    </function>
    <!-- Hash partition for string-->
    <function name="func_hashString" class="StringHash">
        <property name="partitionCount">4</property>
        <property name="partitionLength">256</property>
        <property name="hashSlice">0:2</property>
        <!--<property name="hashSlice">-4:0</property> -->
    </function>
    <!-- date partition  4 case:
    1.set sEndDate and defaultNode: input <sBeginDate ,router to defaultNode; input>sEndDate ,mod the period
    2.set sEndDate, but no defaultNode:input <sBeginDate report error; input>sEndDate ,mod the period
    3.set defaultNode without sEndDate: input <sBeginDate router to defaultNode;input>sBeginDate + (node size)*sPartionDay-1 will report error(expected is defaultNode,but can't control now)
    4.sEndDate and defaultNode are all not set: input <sBeginDate report error;input>sBeginDate + (node size)*sPartionDay-1 will report error
     -->
    <function name="func_date" class="Date">
        <property name="dateFormat">yyyy-MM-dd</property>
        <property name="sBeginDate">2015-01-01</property>
        <property name="sEndDate">2015-01-31</property> <!--if not set sEndDate,then in fact ,the sEndDate = sBeginDate+ (node size)*sPartionDay-1 -->
        <property name="sPartionDay">10</property>
        <property name="defaultNode">0</property><!--the default is -1-->
    </function>
    <!-- pattern partition : mapFile must contains all value of 0~patternValue-1,key and value must be Continuous increase-->
    <function name="func_pattern" class="PatternRange">
        <property name="mapFile">partition-pattern.txt</property>
        <property name="patternValue">1024</property>
        <property name="defaultNode">0</property><!--contains string which is not number,router to default node-->
    </function>
</dble:sharding>
```

1.6 log4j2.xml

1.6.1 配置详述

Dble中的整体配置和一般java项目的log4j2.xml没有什么区别

1.6.1.1 日志滚动删除配置

日志滚动删除是通过DefaultRolloverStrategy的配置对于RollingRandomAccessFile的RolloverStrategy内容进行重载

```
<DefaultRolloverStrategy max="100">
      <Delete basePath="logs" maxDepth="2">
          <IfFileName glob="*/dble-*.log.gz">
              <IfLastModified age="2d">
                  <IfAny>
                      <IfAccumulatedFileSize exceeds="1 GB" />
                      <IfAccumulatedFileCount exceeds="10" />
                  </IfAny>
              </IfLastModified>
          </IfFileName>
      </Delete>
  </DefaultRolloverStrategy>
```

上例中参数说明如下:
basePath：基准路径，日志的统计归类和删除会在此目录下进行
maxDepth： 最大路径深度，在basePath路径下maxDepth深度的日志文件都会被扫描，譬如.../logs/2018-01-01 .../logs/2018-01-02 此类路径也都会被扫描和统计
glob： 日志格式，在所有扫描到的文件中符合此命名规范的文件都被认为是日志文件
age： 举例最后一次修改文件的时间，只有修改时间超过限值的文件才会被考虑删除
IfAccumulatedFileSize： 触发文件大小，符合上述条件的文件大小总量达到触发值则触发清理
IfAccumulatedFileCount： 触发文件数量，符合上述条件的文件数量达到触发值则触发清理

例子详述：在此例中代表会监控logs目录下2层目录深度内所有命名符合"*/dble-*.log.gz"并且最后修改时间已经超出2天的文件，当有符合上述条件的文件大小到达1 GB或者文件数量到达10的时候会触发清理

1.6.2 配置实例

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN" packages="com.actiontech.dble.log">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d [%-5p][%t] %m %throwable{full} (%C:%F:%L) %n"/>
        </Console>

        <RollingRandomAccessFile name="RollingFile" fileName="logs/dble.log"
                                 filePattern="logs/$${date:yyyy-MM}/dble-%d{MM-dd}-%i.log.gz">
            <PatternLayout>
                <Pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %5p [%t] (%l) - %m%n</Pattern>
            </PatternLayout>
            <Policies>
                <OnStartupTriggeringPolicy/>
                <SizeBasedTriggeringPolicy size="250 MB"/>
                <TimeBasedTriggeringPolicy/>
            </Policies>
            <DefaultRolloverStrategy max="100">
                <Delete basePath="logs" maxDepth="2">
                    <IfFileName glob="*/dble-*.log.gz">
                        <IfLastModified age="2d">
                            <IfAny>
                                <IfAccumulatedFileSize exceeds="1 GB" />
                                <IfAccumulatedFileCount exceeds="10" />
                            </IfAny>
                        </IfLastModified>
                    </IfFileName>
                </Delete>
            </DefaultRolloverStrategy>
        </RollingRandomAccessFile>

    </Appenders>
    <Loggers>
        <asyncRoot level="debug" includeLocation="true">
            <AppenderRef ref="Console"/>
            <AppenderRef ref="RollingFile"/>
        </asyncRoot>
    </Loggers>
</Configuration>
```

1.7 全局序列

在分库分表的情况下，数据库自增主键无法保证自增主键的全局唯一。为此，dble提供了全局序列，并且针对不同应用场景提供了多种实现方式。
要应用全局序列，首先需要在cluster.cnf 中配置：

```
sequenceHandlerType=n
```

根据sequenceHandlerType的类型配置具体的实现：

- 1:MySQL offset-step方式
- 2:时间戳方式(类Snowflake)
- 3:分布式时间戳方式(类Snowflake)
- 4:分布式offset-step方式

其次，使用全局序列的表要在sharding.xml中配置表格并指定其自增列。
自增列的指定逻辑**:** 以显式声明的以incrementColumn中的指定为准

```
//有显式指定incrementColumn=pid，则pid作为自增列
<shardingTable name="table1" shardingNode="dn1,dn2" function="func_common_hash" shardingColumn="id" incrementColumn="pid"/>
```

配置之后的使用示例：

```
insert into table1(name) values('test');
insert into table1 set name = 'test';
```

每一个全局序列的具体实现配置将在各个小节进行详细说明；其功能将在2.2 全局序列进行详细描述。

与**MYSQL**差异
在MySQL中要求自增列必须含有唯一键，在dble中对于表格的自增列的唯一属性不做要求，但是在dble进行插入数据操作的时候不允许用户手动插入自增列的数值，自增列只能由dble自己生成的ID进行填充
并且存在以下情形和mysql的行为不一致

```
table1拥有列aid,bid,cid,did 其中bid为在dble中的自增列
insert into table1 values(1,2,3)
和以下sql的效果相等
insert into table1 set aid = 1,cid = 2,did = 3
```

特别提醒 在dble中全局序列只在生成的时候确保其唯一性，在之后的过程中用户被允许使用update或者replace语句进行更新自增字段（特例：自增字段同时也是分片字段是除外)。 这给予用户提供了充足的修改空间但同时要求用户在更新自增字段的时候有足够的谨慎和了解

1.7.1 MySQL-offset-step 方式

1.7.1.1 MySQL-offset-step序列配置

mysql序列由文件sequence_db_conf.properties进行配置。具体配置有如下格式:

#this is comment

`schema1`.`table1`=node1

`schema1`.`table2`=node1

`schema2`.`table1`=node2

...

schemaX：使用全局序列的dble表所属的dble库名。

tableX： 使用全局序列dble的dble表名。

nodeX：实现序列功能的数据节点名。此节点具体配置见下节。

1.7.1.2 数据节点配置

mysql序列的实现依赖一些存储函数。因此在序列可用之前必须在数据节点**nodeX**上创建这些存储函数，并针对使用此节点的表（**`schemaX`.`tableX`**)初始化序列初始值。

数据节点**nodeX**上创建必要的表和存储函数

具体步骤:

a. 用mysql客户端登录到**nodeX**上

mysql ...

b. 切换到**nodeX**上的后端存储数据库db(参见1.5 sharding.xml)

use db;

c. 执行创建脚本dbsql.sql（dbseq.sql的内容详见1.7.1.3节）

source .../dbseq.sql;

d. 初始化相应序列初始值

INSERT INTO DBLE_SEQUENCE VALUES ('**schemaX**`.`**tableX**`', 1, 1);

...

1.7.1.3 dbseq.sql内容

dbseq.sql的内容如下：

```
DROP TABLE IF EXISTS DBLE_SEQUENCE;
CREATE TABLE DBLE_SEQUENCE ( name VARCHAR(64) NOT NULL, current_value BIGINT(20) NOT NULL, increment INT NOT NULL DEFAULT 1, PRIMARY KEY (name) ) ENGINE=InnoDB;

-- ----------------------------
-- Function structure for `dble_seq_currval`
-- ----------------------------
DROP FUNCTION IF EXISTS `dble_seq_currval`;
DELIMITER ;;
CREATE FUNCTION `dble_seq_currval`(seq_name VARCHAR(64)) RETURNS varchar(64) CHARSET latin1
DETERMINISTIC
BEGIN
DECLARE retval VARCHAR(64);
SET retval="-1,0";
SELECT concat(CAST(current_value AS CHAR),",",CAST(increment AS CHAR) ) INTO retval FROM DBLE_SEQUENCE WHERE name = seq_name;
RETURN retval ;
END
;;
DELIMITER ;

-- ----------------------------
-- Function structure for `dble_seq_nextval`
-- ----------------------------
DROP FUNCTION IF EXISTS `dble_seq_nextval`;
DELIMITER ;;
CREATE FUNCTION `dble_seq_nextval`(seq_name VARCHAR(64)) RETURNS varchar(64) CHARSET latin1
DETERMINISTIC
BEGIN
DECLARE retval VARCHAR(64);
DECLARE val BIGINT;
DECLARE inc INT;
DECLARE seq_lock INT;
set val = -1;
set inc = 0;
SET seq_lock = -1;
SELECT GET_LOCK(seq_name, 15) into seq_lock;
if seq_lock = 1 then
SELECT current_value + increment, increment INTO val, inc FROM DBLE_SEQUENCE WHERE name = seq_name for update;
if val != -1 then
UPDATE DBLE_SEQUENCE SET current_value = val WHERE name = seq_name;
end if;
SELECT RELEASE_LOCK(seq_name) into seq_lock;
end if;
SELECT concat(CAST((val - inc + 1) as CHAR),",",CAST(inc as CHAR)) INTO retval;
RETURN retval;
END
;;
DELIMITER ;

-- ----------------------------
-- Function structure for `dble_seq_setvals`
-- ----------------------------
DROP FUNCTION IF EXISTS `dble_seq_nextvals`;
DELIMITER ;;
CREATE FUNCTION `dble_seq_nextvals`(seq_name VARCHAR(64), count INT) RETURNS VARCHAR(64) CHARSET latin1
DETERMINISTIC
BEGIN
DECLARE retval VARCHAR(64);
DECLARE val BIGINT;
DECLARE seq_lock INT;
SET val = -1;
SET seq_lock = -1;
SELECT GET_LOCK(seq_name, 15) into seq_lock;
if seq_lock = 1 then
SELECT current_value + count INTO val FROM DBLE_SEQUENCE WHERE name = seq_name for update;
IF val != -1 THEN
UPDATE DBLE_SEQUENCE SET current_value = val WHERE name = seq_name;
END IF;
SELECT RELEASE_LOCK(seq_name) into seq_lock;
end if;
SELECT CONCAT(CAST((val - count + 1) as CHAR), ",", CAST(count as CHAR)) INTO retval;
RETURN retval;
END
;;
DELIMITER ;

-- ----------------------------
-- Function structure for `dble_seq_setval`
-- ----------------------------
DROP FUNCTION IF EXISTS `dble_seq_setval`;
DELIMITER ;;
CREATE FUNCTION `dble_seq_setval`(seq_name VARCHAR(64), value BIGINT) RETURNS varchar(64) CHARSET latin1
DETERMINISTIC
BEGIN
DECLARE retval VARCHAR(64);
DECLARE inc INT;
SET inc = 0;
SELECT increment INTO inc FROM DBLE_SEQUENCE WHERE name = seq_name;
UPDATE DBLE_SEQUENCE SET current_value = value WHERE name = seq_name;
SELECT concat(CAST(value as CHAR),",",CAST(inc as CHAR)) INTO retval;
RETURN retval;
END
;;
DELIMITER ;
```

1.7.2 时间戳方式

时间戳方式由bootstrap.cnf 和cluster.cnf文件进行配置。具体配置如下：

bootstrap.cnf的**instanceId**：指定instance id值， 必须为[0,1023]之间的整数。

cluster.cnf的**sequenceStartTime**：指定开始时间戳，格式必须为 YYYY-MM-dd HH:mm:ss，默认开始时间 2010-10-04 09:42:54。

注意事项：**bootstrap.cnf**的**instanceId**的配置必须使该dble实例在dble集群中唯一。
另外，使用这种方式需要对应字段为bigint来保证63位

配置示例

见时间戳方式全局序列的配置

1.7.3 分布式时间戳方式

分布式时间戳方式由由bootstrap.cnf 和cluster.cnf文件进行配置。具体配置格式如下：

bootstrap.cnf的**instanceId**：指定instance id值， 必须为[0,511]之间的整数，当**cluster.cnf**的**sequenceInstanceByZk**为true时id由zk生产。

cluster.cnf的**sequenceStartTime**：指定开始时间戳，格式必须为 YYYY-MM-dd HH:mm:ss，默认开始时间 2010-10-04 09:42:54。。

注意事项：

1. 当cluster.cnf的sequenceInstanceByZk的值配置为true时，必须配置zookeeper服务器(参见1.8 myid.properties)。
2. **bootstrap.cnf**的**instanceId**，保证dble在集群中唯一。
3. 使用这种方式需要对应字段为bigint来保证63位。

1.7.4 分布式offset-step方式

分布式offset-step方式由文件sequence_conf.properties进行配置。具体格式如下：

this is comment
`**schema1**`.`**table1**`.MINID=1001
`**schema1**`.`**table1**`.MAXID=2000
`**schema1**`.`**table1**`.CURID=1000

`**schema2**`.`**table2**`.MINID=1001
`**schema2**`.`**table2**`.MAXID=20000
`**schema2**`.`**table2**`.CURID=1000

schemaX：使用mysql序列的dble表所属的dble库名。

tableX：使用mysql序列的dble表名。

注意事项：

1. 每一个zk序列均需要指定当前区间内最小值MINID，当前区间内最大值MAXID，当前区间内当前值CURID。这些值仅在初始配置时有效。
2. 初始配置时MINID要比CURID大1，否则序列值从MINID+1开始。
3. 值（MAXID - MINID + 1）为每次从zookeeper服务器获取的序列值数量。
4. 配置dble使用zookeeper服务器，具体见相关配置1.8 myid.properties。

1.8.1 cache 配置

1.8.1.1 dble的cache使用

dble的cache使用有如下三类：

- SQLRouteCache：从前端连接收到的SQL以及对应的路由结果 内容： shema_user_SQL -> 具体路由结果RouteResult
- ER_SQL2PARENTID：父子表辅助查询SQL以及对应的路由结果。在插入ER子表的时候需要根据子表joinKey(父表parentKey)计算它应该插入的结点，所以需要辅助路由来查询，然后将辅助路由及对应的结果缓存下来以备下次查询。内容为：schema:select * from 父表 where parentKey = (value of joinKey) -> 对应数据shardingNode

1.8.1.2 dble的cache实现

dble的cache实现有如下几种：

- ehcache, 用ehcache缓存作为cache实现。
- leveldb， 用leveldb数据库作为cache实现。
- mapdb， 用MapDB数据库引擎作为cache实现。
- rocksdb，用RocksDB数据库引擎作为cache实现。

1.8.1.3 dble的cache配置

dble的cache配置分为总配置和实现配置。总配置由文件cacheservice.properties进行设定。实现配置由各个实现具体指定，具体详见各个实现的分章节说明。

总配置有如下格式：

设置缓存类型:

factory.**cache_type**=**cache_type**

设置分类缓存的具体值，key为缓存池名字，value是类型，最大容量，以及失效时间

A.SQL路由缓存

pool.SQLRouteCache=**type**,**max_size**,**expire_seconds**

B.ER表子表路由缓存

pool.ER_SQL2PARENTID=**type**,**max_size**,**expire_seconds**

1.8.1.4 cache配置说明

总配置文件中各配置项说明：

a. 以#开头的行为注释，被忽略。 空行被忽略。

b. factory.**cache_type**=**cache_type**是cache的总开关。**cache_type**指定cache类型，具体可以为：ehcache，leveldb、mapdb 或者rocksdb。如果要用cache功能，必须配置该配置项。这个配置项可以指定多个， 每行仅能指定一个。每一个指定一个cache实现。

例如：

配置,

```
factory.encache=ehcache
pool.SQLRouteCache=encache,10000,1800
pool.ER_SQL2PARENTID=encache,1000,1800
```

中的**type**就必须是ehcache。而配置：

```
factory.encache=ehcache
factory.leveldb=leveldb
pool.SQLRouteCache=encache,10000,1800
pool.ER_SQL2PARENTID=leveldb,1000,1800
```

中的**type**可以为encache或者leveldb。

c. pool.SQLRouteCache=**type**,**max_size**,**expire_seconds**和pool.ER_SQL2PARENTID=**type**,**max_size**,**expire_seconds**分别配置SQLRouteCache和ER_SQL2PARENTID的缓存功能。这两个配置项可以配置也可以不配置， 不配值则不使用相应的缓存功能。**type**指定缓存类型，必须是已配置的缓存实现类型；**max_size**指定缓存的最大大小，单位是字节；**expire_seconds**指定缓存项的生命周期，单位是秒。

d. **default**缓存用于缓存没有为其指定特定缓存的表的分区键值到数据所在节点的映射。

1.8.1.5 注意事项

- 使用 RocksDB作为 cache 实现时，需要在dble目录下手工创建 rocksdb 目录，否则dble启动失败。

1.8.2 ehcache存配置

要用ehcache实现缓存，必须在cacheservice.properties中配置使用ehcache，具体请参看上一节内容。

1.8.2.1 ehcache版本

目前，dble使用的是2.6.11.

1.8.2.2 ehcache配置

ehcache的配置通过文件ehcache.xml进行。

具体的缓存存储策略和配置请参看http://www.ehcache.org/documentation/ehcache-2.6.x-documentation.pdf。

例如：

```
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="ehcache.xsd" maxEntriesLocalHeap="100000000" maxBytesLocalDisk="50G" updateCheck="false">
               <defaultCache maxElementsInMemory="1000000" eternal="false" overflowToDisk="false" diskSpoolBufferSizeMB="30" maxElementsOnDisk="10000000" diskPersistent="false" diskExpiryThreadI
ntervalSeconds="120" memoryStoreEvictionPolicy="LRU"/>
</ehcache>
```

需要特殊说明是：

1.dble仅用ehcache配置的defaultCache级别创建cache。

2.maxEntriesLocalHeap

该属性指定允许存储元素的最大条数。如果设定了该值且不为0，则缓存大小限制为缓存的条数限制，具体限制由cacheservice.properties中**max_size**指定，参见上一节内容。如果没有设定该参数，则缓存大小限制仍为大小限制，具体限制由cacheservice.properties中**max_size**指定，参见上一节内容。

3.timeToIdleSeconds

该属性指定一个元素在不被请求的情况下允许在缓存中存在的最长时间。它将被cacheservice.properties中**expire_seconds**取代。

4.dble将defaultCache配置应用到每一个创建的cache。

1.9 自定义拆分算法

1.9.1 工作原理

1.9.1.1 函数的加载

路由函数的加载发生在dble启动或重载时。

1. dble读取sharding.xml时，根据用户配置的标签的class属性
2. dble通过Java的反射机制，从$DBLE_HOME/lib的jar包中，找到对应的jar（里的class文件），加载同名的类并创建对象
3. dble会逐个扫描中的标签，并根据name属性来调用路由函数的对应setter，以此完成赋值过程——例如，如果用户配置了2，那么dble就会尝试找路由函数中叫做setPartitionCount()的方法，并将字符串“2”传给它
4. dble调用路由函数的selfCheck()方法，执行函数编写者制定的检查动作，例如检查赋值得到的变量值是否有问题
5. dble调用路由函数的init()方法，执行函数编写者制定的准备动作，例如创建后面要用到的一些中间变量

1.9.1.2 路由计算

路由函数接受用户SQL中的分片字段的值，计算出这个值对应的数据记录应该在哪个编号的数据分片（逻辑分片）上，DBLE从而知道把这个SQL准确发到这些分片上。

1.9.1.3 参数查询

用户通过管理端口（默认9066），通过SHOW @@ALGORITHM WHERE SCHEMA=? AND TABLE=?来查询表上的路由算法时，dble调用路由算法的getAllProperties()方法，直接从内存中获取路由信息的配置。

```
mysql> show @@algorithm where schema=testdb and table=seqtest;
+-----------------+----------------------------------------------------+
| KEY             | VALUE                                              |
+-----------------+----------------------------------------------------+
| TYPE            | SHARDING TABLE                                     |
| COLUMN          | ID                                                 |
| CLASS           | com.actiontech.dble.route.function.PartitionByLong |
| partitionCount  | 2                                                  |
| partitionLength | 1                                                  |
+-----------------+----------------------------------------------------+
5 rows in set (0.05 sec)
```

1.9.2 开发和部署

1.9.2.1 开发

开发时，理论上只需要引入AbstractPartitionAlgorithm抽象类和RuleAlgorithm接口及它们的依赖类就可以了。但实际上AbstractPartitionAlgorithm抽象类依赖了TableConfig类，由此开启了环游世界的依赖之旅。因此，现实的操作还是引用整个DBLE项目的源代码会比较直接方便。

开发一个新的路由函数时，必须给这个路由函数的开发新建项目，然后再引用DBLE项目（项目引用项目的方式）。而不应该直接打开DBLE的项目，然后在DBLE的项目里面直接新建源代码来直接开发（内嵌开发方式）。通过遵循这个做法，会有以下好处：

1. 路由函数可以独立打包，直接去看路由函数的jar包版本就能够确认函数版本；而把路由函数嵌到DBLE里的话，就很容易出现DBLE版本一样，但不清楚里面的函数是什么版本的窘况
2. 路由函数的递进可以更加自由，如果DBLE的AbstractPartitionAlgorithm抽象类和RuleAlgorithm接口没有变动，同一版本的路由函数可以延续使用好几个版本的DBLE，而不需要每次DBLE释放新版就得去重编译
3. 可以让路由函数中的受保护代码免受DBLE自身的开源协议影响

1.9.2.2 部署

完成开发之后，成品打包成jar包进行发布，而不要直接发布class和依赖的library（其他项目的jar包或class文件）。

让DBLE使用上新的路由函数的过程：

1. 将成品jar包放入$DBLE_HOME/lib目录中
2. 调整jar包的所有者权限（chown）和文件权限（chmod），使之与其他$DBLE_HOME/lib目录里的jar包一样
3. 按照原来的思路配置sharding.xml，但需要注意标签的class属性必须要填写新的路由函数类的完全限定名（Fully Qualified Name），例如net.john.dble.route.functions.NewFunction
4. 配置逻辑表之类的必要信息，重启DBLE后，自动生效。

1.9.3 接口规范

每个路由函数本质上就是一个继承了AbstractPartitionAlgorithm抽象类，并且实现了RuleAlgorithm接口的一个类。下面以内置的com.actiontech.dble.route.function.PartitionByLong为例，介绍实现一个路由函数类所需要做的最小工作（必要工作）。

1.9.3.1 配置项setters

在sharding.xml中，我们需要配置partitionCount和partitionLength两个配置项。

```
<function name="hashmod" class="com">
  <property name="partitionCount">4</property>
  <property name="partitionLength">1</property>
</function>
```

为了让dble在函数加载过程中，能够认出这里的partitionCount（值为4）和partitionLength（值为1），因此PartitionByLong类中，就必须有属性设置方法（setter）setPartitionCount()和setPartitionLength()。而因为sharding.xml是个文本型的XML文件，所以这些函数的传入参数就只能是一个String，数据类型转换和预处理的动作就由这些setter来处理了。

```
public void setPartitionCount(String partitionCount) {
  this.count = toIntArray(partitionCount);
    /* 参考本文的getAllProperties()的说明 */
  propertiesMap.put("partitionCount", partitionCount);
}

public void setPartitionLength(String partitionLength) {
  this.length = toIntArray(partitionLength);
    /* 参考本文的getAllProperties()的说明 */
  propertiesMap.put("partitionLength", partitionLength);
}
```

1.9.3.2 selfCheck()

在函数加载过程中，完成了配置项赋值之后，dble会调用这个路由函数对象的selfCheck()方法，让这个对象自我检查刚才读进来的配置项的值，放在一起是不是有问题。如果有问题的话，路由函数编写者在这时候，可以通过抛出RuntimeException来进行报错，并终止dble使用这个函数，当然，由于RuntimeException的霸道，dble自己也会因此而报错退出。

由于selfCheck()是RuleAlgorithm接口的要求，而且AbstractPartitionAlgorithm抽象类没又实现它，对于想偷懒或者没有必要进行这个检查的人来说，还是需要自行定义一个空的同名方法来实现它。

```
@Override
public void selfCheck() {
}
```

1.9.3.3 init()

在函数加载过程的最后，dble调用这个路由函数对象的init()方法，让这个对象完成一些内部的初始化工作。

在我们的例子PartitionByLong里，通过init()方法准备了PartitionUtil对象，其中有一个哈希值的范围与逻辑分片号对应的数组，这样在后面的路由计算时就能通过查数组来加速得到结果。

```
@Override
public void init() {
  partitionUtil = new PartitionUtil(count, length);

  initHashCode();
}
```

1.9.3.4 calculate()和calculateRange()

dble执行用户SQL时，根据用户SQL的不同，调用calculate()或calculateRange()来确定用户的SQL应该发到哪个数据分片上去。

从IPO（Input-Process-Output）来分析，calculate()和calculateRange()的工作原理是一样的：

- Input：用户SQL中的分片字段值

- Output：用户SQL应该要发往的数据分片的编号
- Process：Input与Output转换的计算过程，由函数开发者编写

calculate()和calculateRange()的使用场景不同，导致它们存在着一些微小的差异。

函数名	调用场景	**Input**	**Output**
calculate()	用户SQL里分片字段的值是单值的情况，例如 ... WHERE sharding_key = 1	1个String	1个Integer
calculateRange()	用户SQL里分片字段的值是连续范围，例如 ... WHERE sharding_key BETWEEN 1 AND 5	2个String	Integer数组

```
@Override
public Integer calculate(String columnValue) {
  try {
    if (columnValue == null || columnValue.equalsIgnoreCase("NULL")) {
      return 0;
    }
    long key = Long.parseLong(columnValue);
    return calculate(key);
  } catch (NumberFormatException e) {
    throw new IllegalArgumentException("columnValue:" + columnValue + " Please eliminate any quote and non number within it.", e);
  }
}

@Override
public Integer[] calculateRange(String beginValue, String endValue) {
  long begin = 0;
  long end = 0;
  try {
    begin = Long.parseLong(beginValue);
    end = Long.parseLong(endValue);
  } catch (NumberFormatException e) {
    return new Integer[0];
  }
  int partitionLength = partitionUtil.getPartitionLength();
  if (end - begin >= partitionLength || begin > end) { //TODO: optimize begin > end
    return new Integer[0];
  }
  Integer beginNode = calculate(begin);
  Integer endNode = calculate(end);

  if (endNode > beginNode || (endNode.equals(beginNode) && partitionUtil.isSingleNode(begin, end))) {
    int len = endNode - beginNode + 1;
    Integer[] re = new Integer[len];

    for (int i = 0; i < len; i++) {
      re[i] = beginNode + i;
    }
    return re;
  } else {
    int split = partitionUtil.getSegmentLength() - beginNode;
    int len = split + endNode + 1;
    if (endNode.equals(beginNode)) {
      //remove duplicate
      len--;
    }
    Integer[] re = new Integer[len];
    for (int i = 0; i < split; i++) {
      re[i] = beginNode + i;
    }
    for (int i = split; i < len; i++) {
      re[i] = i - split;
    }
    return re;
  }
}
```

1.9.3.5 getAllProperties()

当用户找dble要路由函数的配置信息时，dble通过访问路由函数的getAllProperties()来获得一个<配置项, 配置值>的哈希表，然后将里面的内容逐项返回给用户。

getAllProperties()是RuleAlgorithm接口所规定要实现的，但为了简化编写新的路由函数的工作，在AbstractPartitionAlgorithm抽象类里，定义了propertiesMap这个私有变量，并且把“将propertiesMap交出去”作为了实现了getAllProperties()方法的默认实现。一般来说，这个默认的实现能满足需求，而新路由函数编写者只需要在配置项setters处理用户配置时，将<配置项, 配置值>给put()进propertiesMap里就好了。

```
@Override
public Map<String, String> getAllProperties() {
  return propertiesMap;
}
```

1.9.4 内置路由函数的缩写与类名对照表

DBLE内置的路由函数都位于com.actiontech.dble.route.function命名空间。但实际配置sharding.xml的时候，却不用写那么长的完全限定名，这其实都是XMLRuleLoader类做了转换，因此实现了简写。下面就是7个内置函数的类名和它们的简写。

简写名	完整类名
date	PartitionByDate
enum	PartitionByFileMap
hash	PartitionByLong
jumpstringhash	PartitionByJumpConsistentHash
numberrange	AutoPartitionByLong
patternrange	PartitionByPattern
stringhash	PartitionByString

1.9.5 IntelliJ IDEA中的实践

1.9.5.0 前提

1. 安装java开发环境
2. 准备好dble的最新的release版本jar包，以下是2.19.05.0 安装包下载链接：https://github.com/actiontech/dble/releases/download/2.19.05.0%2Ftag/actiontech-dble-2.19.05.0.tar.gz，解压后在lib目录中可以找到dble的对应版本的jar包。

1.9.5.1 创建 java 项目

创建简单的java项目，点击红框中Project

点击Project后弹出框中默认选中java，若没有选中，手动选中后一路点击next，填写恰当的项目名即可。

1.9.5.2 引入 dble jar 包

右击新建的java项目，新建lib目录

将 dble 的jar包复制到lib目录下。

将lib中的dble jar 包添加为项目的依赖。

点击加号之后选中文件系统中的lib目录后加入即可

1.9.5.3 自定义分片算法类

在新建项目中自定义算法类，具体方式请参照上文。

1.9.5.4 新建Artifacts

选中Project Strcucture后，点击Artifacts，在弹出的对话框中直接点击OK。

1.9.5.5 编译**jar**包

选择菜单栏中Build下的Build Artifacts，在弹出的对话框中选中自定义的Artifact后build。

编译完成之后，在项目路径下会生成一个out文件夹，在artifact子文件夹下可以找到生成的jar包。

1.10 配置文件版本变更

dble 在3.20.07.0 版本做了配置的重构。历史变更请参考2.20.04.0的变更

重构后的配置概览图:

1.11 自定义告警模块

- 告警工作原理
  - 告警模块的加载
  - 告警发送和解除
  - 告警Interface Alert详解
  - 告警发送对象详解
- 自定义告警的开发和部署
  - 自定义告警的开发
  - 自定义告警的使用
  - 自定义告警示例以及示例文件

1.11.1 工作原理

1.11.1.1 告警模块的加载

告警模块加载仅在**dble**启动的时候进行

1. dble启动，通过一系列加载之后，开始加载告警模块
2. 由对象AlertGeneralConfig读取告警配置文件dble_alert.properties
3. 根据读取到配置文件中的alert配置项进行对象反射，获取到用户自定义对象的实例
4. 当dble运行过程中发生告警或者告警解除事件，将告警任务放入任务队列中
5. 由一个dble线程循环消费队列中的内容，调用用户自定义对象对于告警任务进行发送

1.11.1.2 告警发送和解除

在dble的告警模块中，将系统中的告警分成以下两种类型

- 可自动解决告警
- 不可自动解决告警

其中“可自动解决告警”为dble在运行过程中遭遇的偶发错误导致的告警，当错误被修正或者重试成功时告警被解除。例如：后端连接数到达最大值，XA事务提交单次失败等
而“不可自动解决告警”则为dble不会重试也不能解决的错误，譬如：启动无法读取xa日志记录文件等

同时在dble中还将告警分成以下两种类型

- dble内部发生的错误告警
- dble和外部交互发生的错误告警

其中“dble内部发生的告警”指的是在dble对于内部任务在执行过程中发生的异常或者其他现象，比如写文件失败，kill后端连接失败等等 而“dble和外部交互发生的错误告警”则指的是dble和外部具体节点交互时候的错误，譬如无法连接某后端节点，或者某后端节点心跳失败等等

根据以上两种分类，在**dble**的告警系统中存在着以下的四种告警信息

- 可自动解决-dble内部告警
- 不可自动解决-dble内部告警

- 可自动解决-dble与外部交互告警
- 不可自动解决-dble与外部交互告警

针对以上的四种告警的类别，我们可以发现，在**dble**的告警周期中，需要有以下四个逻辑上的方法:

- dble与外部交互告警发送方法
- dble内部告警发送方法
- dble与外部交互告警解除发送方法
- dble内部告警解除发送方法

1.11.1.3 告警Interface Alert详解

```
public interface Alert {
    void alertSelf(ClusterAlertBean bean);
    void alert(ClusterAlertBean bean);
    boolean alertResolve(ClusterAlertBean bean);
    boolean alertSelfResolve(ClusterAlertBean bean);
    void alertConfigCheck() throws Exception;
}
```

以上是整个告警接口的定义，可以看到除了一个比较特殊的方法alertConfigCheck（检查告警配置）其他的几个方法基本可以说是成对出现

- alert告警发送 --- dble与外部交互告警
- alertSelf告警发送 --- dble内部告警发送
- alertResolve告警解决发送 --- dble与外部告警解除发送
- alertSelfResolve内部告警解除发送 --- dble内部告警解除发送

四个信息发送的方法分别对应到上文中对于几个dble告警系统中的方法需求
并在此解释下alert和alertSelf方法的区别：alert方法的调用输入中为ClusterAlertBean设定了alertComponentType和alertComponentId两个字段，而alertSelf方法则没有，设定适当的DBLEserver标识的动作交给Alert对象来完成 当然在alertResolve和alertSelfResolve之间的区别也是同样的

1.11.1.4 告警发送对象详解

```
public class ClusterAlertBean {
    String code;                //告警的具体代码
    String level;               //告警发生的具体级别
    String desc;                //告警发送的详细描述
    String sourceComponentType; //告警发生的问题源头组件类型
    String sourceComponentId;   //告警发送的问题源头组件ID
    String alertComponentType;  //具体发送告警的组件类型
    String alertComponentId;    //具体发送告警的组件ID
    String serverId;            //告警发生的服务器ID
    long timestampUnix;         //告警发送的时候戳
    long resolveTimestampUnix;  //告警解除的时间，仅有解除的告警拥有此字段的值
     Map<String, String> labels; //告警的额外附加信息，补充信息
    }
```

以上是在dble中对于告警对象的定义，每个字段的含义都在注释中进行了说明，在自定义告警发送的时候只需要对于告警对象中用户所关心的信息进行发送和处理即可

1.11.2 自定义告警的开发和部署

1.11.2.1 自定义告警的开发

从上文中对于整个dble中告警系统的介绍，可以大致的将dble中的告警进行如下的总结:

- dble会在启动的时候根据配置文件中的信息加载一个Alert的对象
- dble的告警由dble内部的代码进行固定的触发，并将告警的任务发送到一个指定的告警队列中
- dble内部会有一个线程循环调用Alert的实现对象对于告警进行发送
- dble中的告警发送分为四种情况“内部告警”“外部告警”“内部告警解决”“外部告警解决”

按照上文中对于dble内部告警机制和原理的解释，可以整理出对于自定义告警的基本开发过程 ------创建一个Alert接口的实现，并且按照需要的方式将告警的信息发送到指定的地方

为了达成这个目的，可以按照以下的步骤进行逐一的开发，以下提供一个基于IDEA逐步操作的图文步骤，以一个通过邮件发送告警信息为例

1 创建一个**java**项目

最简单的java项目就能完成这个告警自定义的开发

2 将需要的依赖包**copy**到项目中，并且添加到**lib**

在此例中由于仅需额外依赖一个java的邮件发送包mail，所以这里仅导入了两个jar包作为依赖

3.1 编写**Alert**实现类初始化方法

```
public MailAlert() {
    //init the mail data and read config file
    properties = AlertGeneralConfig.getInstance().getProperties();
}
```

在Alert实现类MailAlert中，可以通过调用AlertGeneralConfig.getInstance().getProperties();获取到dble启动时候加载到的告警配置文件(dble_alert.properties)中的内容 所以，在这个地方如果自定义的告警需要使用到各种参数，都可以直接写在文件dble_alert.properties的配置中，而在Alert实现类里面按照key值进行获取即可

3.2 编写**Alert**的配置检查

```
@Override
public void alertConfigCheck() throws ConfigException {
    //check if the config is correct
    if (properties.getProperty(MAIL_SENDER) == null
            || properties.getProperty(SENDER_PASSWORD) == null
            || properties.getProperty(MAIL_SERVER) == null
            || properties.getProperty(MAIL_RECEIVE) == null) {
        throw new ConfigException("alert check error, for some config is missing");
    }
}
```

在此例中由于是基于邮件进行告警发送，所以启动的时候需要在dble_alert.properties文件中配置MAIL_SENDER，SENDER_PASSWORD，MAIL_SERVER，MAIL_RECEIVE四个选项 当配置不能满足自定义的这些选项的时候则抛出异常，dble启动加载的时候会打印对应的异常日志，并且切换为内置默认告警启动
上文中

```
properties.getProperty(MAIL_RECEIVE)
```

的这种方式则是从配置中获取自定义配置项的方法

3.3 编写发送告警主体方法

由于此例中发送告警统一为邮件发送，包括告警和告警解决仅存在邮件内容中的差异，所以在这里仅用一个邮件发送方法send进行实现

```
private boolean sendMail(boolean isResolve, ClusterAlertBean clusterAlertBean) {
    try {
        Properties props = new Properties();
        props.setProperty("mail.debug", "true");
        props.setProperty("mail.smtp.auth", "true");
        props.setProperty("mail.host", properties.getProperty(MAIL_SERVER));
        props.setProperty("mail.transport.protocol", "smtp");

        MailSSLSocketFactory sf = new MailSSLSocketFactory();
        sf.setTrustAllHosts(true);
        props.put("mail.smtp.ssl.enable", "true");
        props.put("mail.smtp.ssl.socketFactory", sf);

        Session session = Session.getInstance(props);

        Message msg = new MimeMessage(session);
        msg.setSubject("DBLE告警 " + (isResolve ? "RESOLVE\n" : "ALERT\n"));
        StringBuilder builder = new StringBuilder();
        builder.append(groupMailMsg(clusterAlertBean, isResolve));
        msg.setText(builder.toString());
        msg.setFrom(new InternetAddress(properties.getProperty(MAIL_SENDER)));

        Transport transport = session.getTransport();
        transport.connect(properties.getProperty(MAIL_SERVER), properties.getProperty(MAIL_SENDER), properties.getProperty(SENDER_PASSWORD));

        transport.sendMessage(msg, new Address[]{new InternetAddress(properties.getProperty(MAIL_RECEIVE))});
        transport.close();
        //send EMAIL SUCCESS return TRUE
        return true;
    } catch (Exception e) {
        e.printStackTrace();
    }
    //send fail reutrn false
        return false;
    }

      private String groupMailMsg(ClusterAlertBean clusterAlertBean, boolean isResolve) {
            StringBuffer sb = new StringBuffer("Alert mail:\n");
            sb.append("         Alert type:" + clusterAlertBean.getCode() + " " + (isResolve ? "RESOLVE\n" : "ALERT\n"));
            sb.append("         Alert message:" + clusterAlertBean.getDesc() + "\n");
            sb.append("         Alert componnent:" + clusterAlertBean.getAlertComponentType() + "\n");
            sb.append("         Alert componnentID:" + clusterAlertBean.getAlertComponentId() + "\n");
            sb.append("         Alert source:" + clusterAlertBean.getAlertComponentId() + "\n");
            sb.append("         Alert server:" + clusterAlertBean.getServerId() + "\n");
            sb.append("         Alert time:" + TimeStamp2Date(clusterAlertBean.getTimestampUnix()) + "\n");
            String detail = "|";
            if (clusterAlertBean.getLabels() != null) {
                for (Map.Entry<String, String> entry : clusterAlertBean.getLabels().entrySet()) {
                    detail += entry.getKey() + ":" + entry.getValue();
                }
            }
            sb.append("         Other detail:" + detail + "|\n");
            return sb.toString();
        }
```

具体如何使用java发送邮件我在此不进行赘述了，大概上面这段代码的原理就是从配置properties获取邮件配置的项，然后发送邮件到指定邮箱即可 而具体的groupMailMsg就是将告警发送过来的对象clusterAlertBean中的数据组织成邮件的文本内容，完全可以根据实际需要进行组织

3.4 实现具体的几个告警发送的方法

```
@Override
    public void alertSelf(ClusterAlertBean clusterAlertBean) {
        alert(clusterAlertBean.setAlertComponentType(COMPONNENT_TYPE).setAlertComponentId(properties.getProperty(COMPONNENT_ID)));
    }

    @Override
    public void alert(ClusterAlertBean clusterAlertBean) {
        clusterAlertBean.setSourceComponentType(COMPONNENT_TYPE).
                setSourceComponentId(properties.getProperty(COMPONNENT_ID)).
                setServerId(properties.getProperty(SERVER_ID)).
                setTimestampUnix(System.currentTimeMillis() * 1000000);
        sendMail(false, clusterAlertBean);
    }

    @Override
    public boolean alertResolve(ClusterAlertBean clusterAlertBean) {
        clusterAlertBean.setSourceComponentType(COMPONNENT_TYPE).
                setSourceComponentId(properties.getProperty(COMPONNENT_ID)).
                setServerId(properties.getProperty(SERVER_ID)).
                setTimestampUnix(System.currentTimeMillis() * 1000000);
        return sendMail(true, clusterAlertBean);
    }

    @Override
    public boolean alertSelfResolve(ClusterAlertBean clusterAlertBean) {
        return alertResolve(clusterAlertBean.setAlertComponentType(COMPONNENT_TYPE).setAlertComponentId(properties.getProperty(COMPONNENT_ID)));
    }
```

以上方法实现就特别简单了，一般情况下都可以按照以上的示例进行实现，基本上就是补全部分缺失的信息，调用发送的方法即可

3.5 打包成**jar**进行使用

通过idea File->Project Structure..创建打包方式

确认打包的内容，注意的点是不要将dble的jar包也当作依赖打包到项目的jar包中去，当自定义中使用的jar包和dble的依赖jar包发生重复的时候，也请不要把重复的依赖打包进去

通过IDEA的Build->Build Artifacts..对于项目进行打包，能在项目新生成的out文件夹下找到对应的jar包

1.11.2.2 自定义告警的使用

有了上面打包出来的jar包之后就是该如何进行使用了，大致的步骤只有三个

- 将打包完成的jar包放入dble安装目录的lib目录下
- 配置文件dble_alert.properties使得自定义的文件能被加载到,将类名配置到alert配置项中去
- 重启dble使得加载生效 在这里同样是衔接上文的案例，文件复制到lib下在此进行跳过 之后修改dble_alert.properties配置文件,大致如下图所示

```
alert=com.actiontech.addtionAlert.MailAlert
mail_sender=123456798@qq.com
sender_pass=qwertyuiop
mail_server=smtp.qq.com
mail_receive=yyyyyyyyyy@actionsky.com
server_id=dble-server-001
componnent_id=DBLE-FOR-XXX-01
```

之后重启dble之后就会加载com.actiontech.addtionAlert.MailAlert这个类进行告警的发送了

1.11.2.3 自定义告警示例以及示例文件

在这里将上文中的实例以文件的形式给出，方便用户进行参考 示例代码下载 示例jar包下载

1.11.3 附录：dble项目中各个告警CODE的含义

告警代码	解释	告警分类
DBLE_WRITE_TEMP_RESULT_FAIL	写出中间结果集到文件失败	内部/非自解决
DBLE_XA_RECOVER_FAIL	XA事务恢复失败	内部/非自解决
XA_READ_XA_STREAM_FAIL	读取XA事务记录文件失败	内部/非自解决
DBLE_XA_READ_DECODE_FAIL	解析XA事务记录文件失败	内部/非自解决
DBLE_XA_READ_IO_FAIL	读取XA事务记录文件失败	内部/非自解决
DBLE_XA_WRITE_IO_FAIL	写XA事务记录文件失败	内部/非自解决
DBLE_XA_WRITE_CHECK_POINT_FAIL	XA写出检查点信息失败	内部/自解决
DBLE_XA_BACKGROUND_RETRY_FAIL	XA后台重试提交失败	内部/自解决
DBLE_REACH_MAX_CON	节点连接数到达配置最大值，获取连接失败	内部/自解决
DBLE_TABLE_NOT_CONSISTENT_IN_SHARDINGS	表结构在不同数据节点中不一致	内部/自解决
DBLE_TABLE_NOT_CONSISTENT_IN_MEMORY	表结构在数据节点中和dble内存中不一致	内部/自解决
DBLE_GLOBAL_TABLE_COLUMN_LOST	全局表检查列不存在	内部/自解决
DBLE_CREATE_CONN_FAIL	创建连接到后端mysql的连接失败	外部/自解决
DBLE_DB_INSTANCE_CAN_NOT_REACH	后端连接创建不可达	外部/非自解决
DBLE_KILL_BACKEND_CONN_FAIL	Kill后端连接执行失败	内部/非自解决
DBLE_NIOREACTOR_UNKNOWN_EXCEPTION	NIO意外报错	内部/非自解决
DBLE_NIOREACTOR_UNKNOWN_THROWABLE	NIO意外严重错误	内部/非自解决
DBLE_NIOCONNECTOR_UNKNOWN_EXCEPTION	NIO后端连接创建器意外错误	内部/非自解决
DBLE_TABLE_LACK	配置中的表格未被创建	内部/自解决
DBLE_GET_TABLE_META_FAIL	获取表格的创建语句失败	内部/自解决
DBLE_TEST_CONN_FAIL	测试后端连接可达性失败	内部/非自解决
DBLE_HEARTBEAT_FAIL	心跳后端节点失败	外部/自解决
DBLE_SHARDING_NODE_LACK	缺少可用的shardingNode节点	外部/自解决

自定义全局表检查

- 全局表一致性检查逻辑
- 全局表自定义方法详述
  - getCountSQL
  - getFetchCols
  - resultEquals
  - failResponse
  - resultResponse
- 自定义全局表检查操作步骤
- 自定义全局表检查配置

背景

全局表是dble中一种特殊类型的表格，一般来说认为在一个全局表table_a所有分布的节点上，table_a因同时满足以下两个条件：

- 拥有相同的表格结构
- 拥有相同的表格数据

但事实上在系统和dble的运行过程中，可能由于一些不可避免分布式事务方面的误差，导致在长时间运行之后，不同节点上面的table_a上面的数据不一致
为了及时的发现问题并不再造成更进一步的错误，dble中采用定时进行表格数据检查的方式对于table_a中的数据一致性进行检查，并及时把检查的结果通知到运维人员

工作原理大致逻辑

全局表检查工作逻辑

全局表检查的大致逻辑如下图所示：

上图中着色的部分允许接收用户的自定义，在下一小节中会对于每个步骤进行详细的说明
整体上来说全局表检查的工作原理分为以下几个步骤：

- 加载表格检查配置，在启动或者是reload阶段将配置加载到定时任务管理器CronScheduler中

- 当根据配置的触发条件正常触发时执行GlobalCheckJob开始任务
- 在GlobalCheckJob中根据表格配置计算表格配置并创建具体的SQL检查任务
  - 此时会调用checker中的方法返回执行的具体SQL语句以及需要取得的结果级列名
  - 按照shardingNode的结构将SQL任务提前构造完毕
- 触发SQL执行，逐个执行构造完成的SQL任务，下发SQL到MySQL进行执行
- 等待所有的SQL执行结果都返回(成功或者失败)
- SQL执行结果返回，回调方法checkResults进行结果集检查
- 根据checker中的结果集比较方法对于SQL执行的结果进行比较
- 调用回馈方法进行结果回馈，当SQL执行结果有超过一个版本(存在不一致)时调用失败接口failResponse，当SQL执行结果只有一个版本(所有正常返回的结果一致)时调用resultResponse方法

全局表检查方法详解

1.执行**SQL**定义**String getCountSQL(String dbName, String tName)**

功能： 返回全局表检查需要对于表格执行的SQL内容
输入： SQL执行的MySQL中database的名称，所检查的表格的名称
输出： 检查具体需要执行的SQL
举例：

```
public String getCountSQL(String dbName, String tName) {
      //假如需要对于对应的table名字求checksum
      return "checksum table " + tName;
  }
```

2.结果集定义 **getFetchCols()**

功能： 返回SQL执行完毕需要使用的结果集中的列名
输入： 无
输出： 需要收集的列名list
举例：

```
public String[] getFetchCols() {
   //checksum返回结果，我们只关心Checksum字段的返回值
   // mysql> checksum table suntest;
   //+-------------+----------+
   //| Table     | Checksum |
   //+-------------+----------+
   //| db1.suntest |1290812451|
   //+-------------+----------+
   //所以return的内容只需要一个Checksum的列名即可
      return new String[]{"Checksum"};
  }
```

3.SQL结果比较方法 **boolean resultEquals(result1,result2)**

功能： 用于判断两个不同节点的返回结果是否一致
输入： 不同节点的两个节点result，result1,result2

```
SQLQueryResult<List<Map<String, String>>> result

result
     |
     ------ row(List)
                    |
              -------Key-Value(Field-Value)
例如checksum
result
     |
     ------ row(List<1> checksum table suntest只有一行返回结果)
                    |
           ------Key-Value(checksum - 1290812451 getFetchCols只取了一列)
```

输出： 需要收集的列名list
举例：

```
public boolean resultEquals(SQLQueryResult<List<Map<String, String>>> or, SQLQueryResult<List<Map<String, String>>> cr) {
        //因为checksum只有一行，并且即使表不存在也会有一行结果集
        //所以直接取结果集的第一行即可
        Map<String, String> oresult = or.getResult().get(0);
        Map<String, String> cresult = cr.getResult().get(0);
        //直接对比Map中checksum的值是不是一致即可
        return (oresult.get("Checksum") == null && cresult.get("Checksum") == null) ||
                (oresult.get("Checksum") != null && cresult.get("Checksum") != null &&
                        oresult.get("Checksum").equals(cresult.get("Checksum")));
    }
```

4.失败行为接口 **failResponse(resultList)**

功能： 检查失败的通知/响应/其他自定义行为
输入： 检查结果列表
输出： 无
举例：

```
public void failResponse(List<SQLQueryResult<List<Map<String, String>>>> res) {
        //简单的情况下直接在日志中打印出对应信息
        //如果有需要可以自行实现发邮件/发短信/发接口给告警系统等等
        String errorMsg = "Global Consistency Check fail for table :" + schema + "-" + tableName;
        System.out.println(errorMsg);
        for (SQLQueryResult<List<Map<String, String>>> r : res) {
            System.out.println("Checksum is : " + r.getResult().get(0).get("Checksum"));
        }
    }
```

5.其他结果通知 **void resultResponse(errorList)**

功能： 检查结果的通知/响应/其他自定义行为
输入： 检查(报错)结果列表
输出： 无
举例：

```
public void resultResponse(List<SQLQueryResult<List<Map<String, String>>>> elist) {
        //输入参数是检查过程中SQL执行报错的list，因为SQL自定义
        //不同的检查SQL对于SQL报错的处理不同，具体报错应该别忽视
        //或者应该视作不一致，由用户自己进行定义
        String tableId = schema + "." + tableName;

        if (elist.size() == 0) {
            System.out.println("Global Consistency Check success for table :" + schema + "-" + tableName);
        } else {
            System.out.println("Global Consistency Check fail for table :" + schema + "-" + tableName);
            StringBuilder sb = new StringBuilder("Error when check Global Consistency, Table ");
            sb.append(tableName).append(" shardingNode ");
            for (SQLQueryResult<List<Map<String, String>>> r : elist) {
                System.out.println("error node is : " + r.getTableName() + "-" + r.getShardingNode());
                sb.append(r.getShardingNode()).append(",");
            }
            sb.setLength(sb.length() - 1);
        }
    }
```

自定义全局表检查的开发及使用

检查的自定义步骤

1 创建一个**java**项目

2 将需要的依赖包**copy**到项目中，并且添加到**lib**

3 按照上一节的介绍逐个实现**5**个自定义方法

4 打包成**jar**进行使用

5 示例文件及**jar**包

示例代码下载

自定义检查的配置

当前的全局表检查定义为schema.table级别，需要对于每个需要进行全局表一致性检查的表格进行配置，配置的下放带来一些繁琐的工作，但是却提供了一个重要的特性，用户可以根据不同全局表格的需要，或者是业务上面的特性，给与不同的全局表格不同的校验方式

注意***:***检查方式的修改仅在***reload***或者重启之后生效

举例：

```
<!--dble内置CHECKSUM检查方式-->
<globalTable name="tb_global1" shardingNode="dn1,dn2" cron = "0 * * * * ?" globalCheckClass="CHECKSUM"/>

<!--dble内置COUNT检查方式-->
<globalTable name="tb_global2" shardingNode="dn1,dn2" cron = "0 * * * * ?" globalCheckClass="COUNT"/>

<!--上文中我们自定义的CustomizeTest类的检查方式-->
<globalTable name="tb_global3" shardingNode="dn1,dn2" cron = "0 * * * * ?" globalCheckClass="CustomizeTest"/>
```

自定义的jar包和其他dble内的自定义功能一样，将jar包放置于algorithm或者lib目录下就会在启动的时候被dble加载到，但是由于java中的类加载方式，如果由更新jar包内容和新增jar包的情况下，请先重启dble进程

注意：当修改自定义***jar***包的时候请重启***dble***，此时***reload***可能无法得到预期的结果

2.功能描述

- 2.0 管理端元数据库
- 2.1 管理端命令
  - 2.1.1 select命令
  - 2.1.2 set命令
  - 2.1.3 show命令
  - 2.1.4 switch命令
  - 2.1.5 kill命令
  - 2.1.6 stop命令
  - 2.1.7 reload命令
  - 2.1.8 rollback命令
  - 2.1.9 offline命令
  - 2.1.10 online命令
  - 2.1.11 file命令
  - 2.1.12 log命令
  - 2.1.13 配置检查命令
  - 2.1.14 pause & resume 命令
  - 2.1.15 慢查询日志相关命令
  - 2.1.16 创建/删除物理库命令
  - 2.1.17 check @@metadata命令
  - 2.1.18 release @@reload_metadata命令
  - 2.1.19 split命令
  - 2.1.20 flow_control 命令
  - 2.1.21 刷新连接池命令
- 2.2 全局序列
  - 2.2.1 MySQL offset-step方式
  - 2.2.2 时间戳方式
  - 2.2.3 分布式时间戳方式
  - 2.2.4 分布式offset-step方式
- 2.3 读写分离
- 2.4 注解
- 2.5 分布式事务
  - 2.5.1 XA事务概述
  - 2.5.2 XA事务的提交以及回滚
  - 2.5.3 XA事务的后续补偿以及日志清理
  - 2.5.4 XA事务的记录
  - 2.5.5 一般分布式事务概述
- 2.6 连接池管理
- 2.7 内存管理
- 2.8 集群同步协调&状态管理
- 2.9 grpc 告警
- 2.10 表meta数据管理
  - 2.10.1 Meta信息初始化
  - 2.10.2 Meta信息维护
  - 2.10.3 一致性检测
  - 2.10.4 View Meta
- 2.11 统计管理
  - 2.11.1 查询条件统计
  - 2.11.2 表状态统计
  - 2.11.3 用户状态统计
  - 2.11.4 命令统计
  - 2.11.5 heartbeat统计
  - 2.11.6 网络读写统计
- 2.12 故障切换
- 2.13 前后端连接检查
- 2.14 ER表
- 2.15 global表
- 2.16 缓存的使用
- 2.17 执行计划
- 2.18 性能观测和调整
- 2.19 智能计算reload
- 2.20 慢查询日志
- 2.21 单条SQL性能trace
- 2.22 KILL @@DDL_LOCK
- 2.23 外部高可用联动
- 2.24 超时控制
- 2.25 流量控制
- 2.26 client_found_rows权能标志

2.0 管理端元数据库dble_information

2.0.0 简介

dble_information 提供了一系列表格来描述dble内部一些元数据，可以通过管理端口连接dble之后，use dble_information之后查询内部的元数据信息。
以下是这些表格的详述:
支持表格的投影(select)
支持表格的选择(where)
支持表格的连接(join)
支持表格的非关联where子查询
支持聚合运算
支持排序
支持标量函数
支持 use dble_information
支持 show tables [like]
支持 desc|describe table xxx
支持 show databases; 注意和show @@database不同
部分表格支持INSERT/UPDATE/DELETE

2.0.1 dble_information下的表

2.0.1.0 version

- 表名：version
- 含义：dble版本号
- 字段：

列名	主键	注释
version	true	版本号

- 数据行：

```
- dble的版本号
```

2.0.1.1 dble_variables

- 表名：dble_variables
- 含义：全局设置
- 字段：

列名	主键	注释
variable_name	true	变量名
variable_value		变量值
comment		说明
read_only		是否只读

- 数据行：

```
- version_comment:  版本信息
- isOnline: 在线状态
- heap_memory_max:  堆内存的最大限制(mb)
- direct_memory_max: 通过-XX:MaxDirectMemorySize设置的值

附加项：show @@sysparam中的所有配置
```

2.0.1.2 dble_status

- 表名：dble_status
- 含义：全局状态
- 字段：

列名	主键	注释
variable_name	true	变量名
variable_value		变量值
comment		说明

- 数据行：

```
- uptime: dble启动的时间长度(秒)
- current_timestamp: dble系统的当前时间
- startup_timestamp: dble系统的启动时间
- heap_memory_max: 堆内存的最大限制
- heap_memory_used: 堆内存的使用量
- heap_memory_total: 堆内存的总量
- config_reload_timestamp: 上次config加载时间
- config_reload_rollback_timestamp: 上次config的rollback时间
- direct_memory_max: 通过-XX:MaxDirectMemorySize设置的值
- direct_memory_pool_size: 内存池的大小，等于bufferpoolpagesize和bufferpoolpagenumber的乘积
- direct_memory_pool_used: 已经使用的内存池中的directmemory内存
- questions: 请求数
- transactions: 事务数
```

2.0.1.3 dble_thread_pool

- 表名：dble_thread_pool
- 含义：线程池使用情况
- 字段:

列名	主键	注释
name	true	线程池名称
size		线程池大小
active_count		活跃的线程数
waiting_task_count		等待消费的任务数(如果值较大, 需增大处理线程数)

- 数据行： 略

2.0.1.4 dble_processor

- 表名：dble_processor
- 含义：processor信息
- 字段:

列名	主键	注释
name	true	名称
type		类型(session/backend)
conn_count		其负责处理的连接数
conn_net_in		网络接收流量 (线程不安全)
conn_net_out		网络发送流量 (线程不安全)

- 数据行：略

2.0.1.5 dble_sharding_node

- 表名：dble_sharding_node
- 含义：sharding_node配置及状态
- 字段:

列名	主键	注释
name	true	名称
db_group		db_group名
db_schema		db_schema名
pause		是否暂停流量

- 数据行：略

2.0.1.6 dble_db_group

- 表名：dble_db_group
- 含义：db_group配置及状态
- 字段:

列名	主键	注释
name	true	名称
heartbeat_stmt		心跳用的sql语句
heartbeat_timeout		心跳超时时间(秒)
heartbeat_retry		心跳重试次数
rw_split_mode		读写分离模式
delay_threshold		延迟时间
disable_ha		是否停用高可用
active		是否落盘

- 数据行：略

2.0.1.7 dble_db_instance

- 表名：dble_db_instance
- 含义：db_instance配置及状态
- 字段：

列名	主键	注释
name	true	名称
db_group	true	db_group的主键
addr		地址
port		端口
primary		是否是主节点
user		账号
password_encrypt		加密后的密码
encrypt_configured		是否启用加密
active_conn_count		当前活动的后端连接数量
idle_conn_count		当前空闲的后端连接数量
read_conn_request		获取读连接的次数
write_conn_request		获取写连接的次数
disabled		是否被标记为disabled
last_heartbeat_ack_timestamp		上次收到心跳回复时间戳
last_heartbeat_ack		init/ok/error/timeout
heartbeat_status		idle/checking
heartbeat_failure_in_last_5min		过去5分钟, 心跳失败的次数
min_conn_count		最小后端连接数量
max_conn_count		最大后端连接数量
read_weight		读负载
id		id
connection_timeout		获取连接的超时时间
connection_heartbeat_timeout		空闲连接检测后的超时时间
test_on_create		连接创建后是否检测有效性
test_on_borrow		连接被借出后是否检测有效性
test_on_return		连接被返回时是否检测有效性
test_while_idle		连接空闲时是否检测有效性
time_between_eviction_runs_millis		扩缩容线程的检测周期
evictor_shutdown_timeout_millis		扩缩容线程停止的超时时间
idle_timeout		连接空闲多久之后被回收
heartbeat_period_millis		连接池的心跳周期

- 数据行：略

2.0.1.8 dble_schema

- 表名：dble_schema
- 含义：schema配置
- 字段：

列名	主键	注释
name	true	版本号
sharding_node		sharding_node表的主键
sql_max_limit		最大返回结果集限制

- 数据行：略

2.0.1.9 session_variables

- 表名：session_variables
- 含义：前端连接变量
- 字段：

列名	主键	注释
session_conn_id	true	前端连接id
variable_name	true	变量名
variable_value		变量名值
variable_type		变量类型(sys/user)

- 数据行：

```
已知变量 ：
- tx_read_only：只读事务(mysql8.0)
- transaction_read_only：只读事务(mysql5.7)
- character_set_client：字符集
- collation_connection：字符集
- character_set_results：字符集
- tx_isolation_level：隔离级别(mysql8.0)
- transaction_isolation：隔离级别 (mysql5.7)
- autocommit：自动提交
- 其他被特别设置的变量
```

2.0.1.10 session_connections

- 表名：session_connections
- 含义：前端连接
- 字段：

列名	主键	注释
session_conn_id	true	前端连接id
remote_addr		远端地址
remote_port		远端端口
local_port		本地端口
processor_id		负责处理连接的处理器id
user		登录用户名
tenant		租户
schema		当前schema? (分库分表/读写分离 功能不同)
sql		最后运行的sql(如果长度大于1024个字符，将会被截断为1024)
sql_execute_time		已完成sql的响应时间, 或未完成的sql的持续时间(单位为ms)（由于实现方式的原因，可能出现正负20ms的误差）
sql_start_timestamp		sql的开始时间戳
sql_stage		运行的当前阶段,结束时会变成finished
conn_net_in		网络接收流量
conn_net_out		网络发送流量
conn_estab_time		连接建立时长(毫秒)
conn_recv_buffer		接收缓冲区大小(字节) (若值较大, 说明正在接收较大的数据包, 或者接收的数据包没有及时被消费)
conn_send_task_queue		网络发送任务队列(个) (若值较大, 说明没有线程对数据包进行及时发送)
in_transaction		该连接是否在事务中
entry_id		入口id

- 数据行：略

2.0.1.11 backend_variables

- 表名：backend_variables
- 含义：后端连接变量
- 字段：

列名	主键	注释
backend_conn_id	true	后端连接id
variable_name	true	变量名
variable_value		变量值
variable_type		变量类型(sys/user)

- 数据行：

```
已知行:
- tx_read_only, 只读事务(mysql8.0)
- transaction_read_only, 只读事务(mysql5.7)
- character_set_client, 字符集
- collation_connection, 字符集
- character_set_results, 字符集
- tx_isolation_level, 隔离级别(mysql8.0)
- transaction_isolation, 隔离级别(mysql5.7)
- autocommit, 自动提交
- 其他被特别设置的变量
```

2.0.1.12 backend_connections

- 表名：backend_connections
- 含义：后端连接
- 字段：

列名	主键	注释
backend_conn_id	true	后端连接id
db_group_name		db组
db_instance_name		db实例
remote_addr		远端地址
remote_port		远端端口
remote_processlist_id		远端的mysql线程id
local_port		本地端口
processor_id		负责处理连接的处理器id
user		登录用户名
schema		当前schema? (分库分表/读写分离 功能不同)
session_conn_id		与之对应的前端连接id, 未使用时可为空
sql		最后运行的sql(如果长度大于1024个字符，将会被截断为1024)
sql_execute_time		响应时间或者未完成sql持续时间（由于实现方式的原因，可能出现正负20ms的误差）
mark_as_expired_timestamp		该连接被标记为退休的时间, 退休连接将不返还连接池, 当前任务结束后即关闭
conn_net_in		网络接收流量
conn_net_out		网络发送流量
conn_estab_time		连接建立时长(秒)
borrowed_from_pool		是否从连接池中取出使用(这列可以按照当前连接池属性扩展为多列)
conn_recv_buffer		接收缓冲区大小(字节) (若值较大, 说明正在接收较大的数据包, 或者接收的数据包没有及时被消费)
conn_send_task_queue		网络发送任务队列(个) (若值较大, 说明没有线程对数据包进行及时发送)
used_for_heartbeat		该连接是否被用于心跳检测
conn_closing		该连接是否正在被关闭
xa_status		该连接的xa状态
in_transaction		该连接是否在事务中

- 数据行：略

2.0.1.13 dble_table系列

2.0.1.13.0 dble_table

- 表名：dble_table
- 含义：table基本信息
- 字段：

列名	主键	注释
id	true	自增序列（配置中的table前缀为C、schema配置默认的mysql节点中的table的前缀为M）
name		名称
schema		schema名称
max_limit		最大返回结果集限制
type		global/single/sharding/child/no sharding

- 逻辑主键
  name、schema
- 数据行：略

2.0.1.13.1 dble_global_table

- 表名：dble_global_table
- 含义：全局表信息
- 字段：

列名	主键	注释
id	true	dble_table的id字段
check		是否开启一致性检查
checkClass		开启一致性检查的class
cron		开启一致性检查的定时任务

- 数据行：略

2.0.1.13.2 dble_sharding_table

- 表名：dble_sharding_table
- 含义：分片表信息
- 字段：

列名	主键	注释
id	true	dble_table的id字段
increment_column		全局序列
sharding_column		拆分列
sql_required_sharding		sqlRequiredSharding配置
algorithm_name		拆分算法的名称

- 数据行：略

2.0.1.13.3 dble_child_table

- 表名：dble_child_table
- 含义：分片子表信息
- 字段：

列名	主键	注释
id	true	dble_table的id字段
parent_id		父表的id
increment_column		全局序列
join_column		关联列
paren_column		父表的列

- 数据行：略

2.0.1.13.4 dble_table_sharding_node

- 表名：dble_table_sharding_node
- 含义：表和sharding_node关联信息
- 字段：

列名	主键	注释
id	true	dble_table的id字段
sharding_node	true	分片节点
order		sharding_node的顺序(对拆分表意义重大，从0开始计数)

- 数据行：略

2.0.1.14 dble_algorithm

- 表名：dble_algorithm
- 含义：拆分算法的配置
- 字段：

列名	主键	注释
name	true	名称
key	true	属性名称
value		属性值
is_file		mapfile的显示格式（true：file，false：content）有些算法由于使用了mapfile，可能导致属性过多，这种情况最多展示1024字节，超过则显示文件名称

- 数据行：略

2.0.1.15 dble_entry系列

2.0.1.15.0 dble_entry

- 表名：dble_entry
- 含义：登录入口表(目前是用户或者用户+租户的模式)
- 字段：

列名	主键	注释
id	true	自增序列
type		入口类型(username/conn_attr), 通过用户名识别或通过连接属性识别
user_type		是否为管理用户/读写分离用户/sharding用户
username		用户名
password_encrypt		密码
encrypt_configured		原本密码是否加密
conn_attr_key		连接属性键，目前支持tenant或者空
conn_attr_value		连接属性值
white_ips		白名单
readonly		是否只读 （读写分离用户不支持，填写-）
max_conn_count		最大连接数限制
blacklist		黑名单

- 数据行：略

2.0.1.15.1 dble_entry_schema

- 表名：dble_entry_schema
- 含义：分库用户对应的schema的关系表
- 字段：

列名	主键	注释
id	true	dble_entry表的id
schema	true	虚拟schema的名字

- 数据行：略

2.0.1.15.2 dble_rw_split_entry

- 表名：dble_rw_split_entry
- 含义：分库用户对应的schema的关系表
- 字段：

列名	主键	注释
id	true	自增序列（需要实现）
type		入口类型(username/conn_attr), 通过用户名识别或通过连接属性识别
username		用户名
password_encrypt		密码
encrypt_configured		是否启用加密
conn_attr_key		连接属性键， 目前支持tenant或者空
conn_attr_value		连接属性值
white_ips		白名单
max_conn_count		最大连接数限制
blacklist		黑名单
db_group		对应的db_group的名字

- 逻辑主键
  username、conn_attr_key、conn_attr_value
- 数据行：略

2.0.1.15.3 dble_entry_table_privilege

- 表名：dble_entry_table_privilege
- 含义：分库用户的对于表的privilege权限
- 字段：

列名	主键	注释
id	true	dble_entry表的id
schema	true	对应的schema的名字
table	true	对应的table的名字
exist_metas		对应table的元数据在dble中是否存在
insert		是否允许insert
update		是否允许update
select		是否允许select
delete		是否允许delete
is_effective		配置是否生效

- 数据行：略

2.0.1.16 dble_blacklist

- 表名：dble_blacklist
- 含义：黑名单信息
- 字段：

列名	主键	注释
name	true	名称
property_key	true	黑名单属性key
property_value		黑名单属性value
user_configured		是否是用户配置的

- 数据行：略

2.0.1.17 processlist

- 表名：processlist
- 含义：查看前端连接和后端连接对应关系(若前端连接没有对应的后端连接，显示NULL)
- 字段：

列名	主键	注释
front_id	true	前端连接id
sharding_node		前端连接下发操作的 sharding node
db_instance		对应后端的实例name
mysql_id	true	后端连接对应的 mysql 线程id
user		用户名
front_host		客户端主机名
mysql_db		后端连接默认数据库，来自于 mysql 'show processlist' 字段 db
command		mysql线程正在执行的指令类型，来自于 mysql 'show processlist' 字段 command
time		mysql线程处于当前state的时间，来自于 mysql 'show processlist' 字段 time
state		mysql线程执行状态，来自于 mysql 'show processlist' 字段 state
info		mysql线程执行语句，来自于 mysql 'show processlist' 字段 info

- 数据行：略(类似show @@processlist)

2.0.1.18 dble_thread_usage

- 表名：dble_thread_usage
- 含义：线程使用率
- 字段：

列名	主键	注释
thread_name	true	线程名称
last_quarter_min		最近15s平均使用率
last_minute		最近1min平均使用率
last_five_minute		最近5min平均使用率

- 数据行：略(类似show @@thread_used;)

2.0.1.19 dble_reload_status

- 表名：dble_reload_status
- 含义：最近的reload信息
- 字段：

列名	主键	注释
index	true	reload对应的编号，能与日志中的[RL]日志编号相对应
cluster		当前dble使用的集群方式
reload_type		最近的reload的类型 reload_matadata/reload_all/manager_insert/manager_update/mamager_delete
reload_status		最近一次reload的执行状状态 not_reloading/self_reload/meta_reload/waiting_others
last_reload_start		起始时间
last_reload_end		结束时间
trigger_type		触发类型local_command/cluster_notify
end_type		结束原因

- 数据行：略(类似show @@reload_status)

2.0.1.20 dble_xa_session

- 表名：dble_xa_session
- 含义：后端重试xa事务信息
- 字段：

列名	主键	注释
front_id	true	前端连接id
xa_id		xa事务id
xa_state		xa事务状态
sharding_node	true	xa提交失败的sharding_node名称,需要展开成多行

- 数据行：略(show @@session.xa)

2.0.1.21 dble_ddl_lock

- 表名：dble_ddl_lock
- 含义：当前dble内部未释放的ddl
- 字段：

列名	主键	注释
schema	true	schema名称
table	true	table名称
sql		ddl sql语句

- 数据行：略(类似show @@ddl)

2.0.2 支持INSERT/UPDATE/DELETE的语法&表格

为满足读写分离的场景，对部分配置表格支持增删改功能

2.0.2.0 INSERT Syntax

```
INSERT
    [INTO] tbl_name
    [(col_name [, col_name] ...)]
    {VALUES | VALUE} (value_list) [, (value_list)] ...

INSERT
    [INTO] tbl_name
    SET assignment_list
```

注意：

- 不支持 INSERT ...SELECT ,LOW_PRIORITY,DELAYED,HIGH_PRIORITY,IGNORE,ON DUPLICATE KEY UPDATE,PARTITION

2.0.2.1 UPDATE Syntax

```
UPDATE table_reference
    SET assignment_list
    WHERE where_condition
value:
    {expr | DEFAULT}
assignment:
    col_name = value
assignment_list:
    assignment [, assignment]
```

注意：

- 不支持多表更新
- 不支持子查询
- 不支持修改主键（物理主键、逻辑主键）
- 不支持LOW_PRIORITY，IGNORE，ORDER BY，LIMIT ,PARTITION
- 不支持不带条件（where）
- 不支持别名

2.0.2.2 DELETE Syntax

```
DELETE FROM tbl_name WHERE where_condition
```

注意：

- 不支持多表删除
- 不支持子查询
- 不支持LOW_PRIORITY，IGNORE，ORDER BY，LIMIT ,PARTITION
- 不支持不带条件（where）
- 不支持别名

2.0.2.3 支持INSERT/UPDATE/DELETE的表格

2.0.2.3.0 dble_db_group

- 不可写列：active

2.0.2.3.1 dble_db_instance

- 不可写列：active_conn_count、idle_conn_count、read_conn_request、write_conn_request、last_heartbeat_ack_timestamp、last_heartbeat_ack、heartbeat_status、heartbeat_failure_in_last_5min

2.0.2.3.2 dble_rw_split_entry

- 不可写列：id、blacklist、type

2.1 管理端命令集

- 2.1.1 select命令
- 2.1.2 set命令
- 2.1.3 show命令
- 2.1.4 switch命令
- 2.1.5 kill命令
- 2.1.6 stop命令
- 2.1.7 reload命令
- 2.1.8 rollback命令
- 2.1.9 offline命令
- 2.1.10 online命令
- 2.1.11 file命令
- 2.1.12 log命令
- 2.1.13 配置检查命令
- 2.1.14 pause & resume 命令
- 2.1.15 慢查询日志相关命令
- 2.1.16 create database命令
- 2.1.17 check @@metadata命令
- 2.1.18 release @@reload_metadata 命令
- 2.1.19 split 命令
- 2.1.20 flow_control 命令
- 2.1.21 刷新连接池命令

2.1.1 select 命令

2.1.1.1 select @@VERSION_COMMENT

select @@VERSION_COMMENT;

描述：查询dble的版本信息；
例：

```
MySQL [(none)]> select @@VERSION_COMMENT;
+-------------------------+
| @@VERSION_COMMENT       |
+-------------------------+
| dble Server (ActionTech) |
+-------------------------+
1 row in set (0.02 sec)
```

列描述：
略

2.1.1.2 select @@SESSION.TX_READ_ONLY / select @@SESSION.Transaction_READ_ONLY

select @@SESSION.TX_READ_ONLY;

select @@SESSION.Transaction_READ_ONLY

描述：为了支持驱动连接管理端时下发的上下文
结果：返回管理用户是否readonly

2.1.1.3 select @@max_allowed_packet

select @@max_allowed_packet;

描述：无实际意义，仅为了支持驱动连接管理端时下发的上下文
结果：定值，永远返回1048576

2.1.1.4 select TIMEDIFF(NOW(), UTC_TIMESTAMP())

select TIMEDIFF(NOW(), UTC_TIMESTAMP())

描述：无实际意义，仅为了支持驱动连接管理端时下发的上下文
结果：定值，永远返回00:00:00

2.1.2 set xxx

set xxx;

其中，xxx为要设置的变量。
描述：无实际意义，仅为了支持驱动连接管理端时下发的上下文
结果：永远返回OK

2.1.3 show命令

2.1.3.1 show @@time.current

show @@time.current;

描述：展示系统当前时间

结果：略

2.1.3.2 show @@time.startup

show @@time.startup;

描述：展示系统启动时间

结果：略

2.1.3.3 show @@version

show @@version;

描述：展示dble版本

结果：略

2.1.3.4 show @@server

show @@server;

描述：dble的当前信息

例：

```
mysql> show @@server;
+------------+-------------+--------------+------------+---------------------+---------------+---------+--------+
| UPTIME     | USED_MEMORY | TOTAL_MEMORY | MAX_MEMORY | RELOAD_TIME         | ROLLBACK_TIME | CHARSET | STATUS |
+------------+-------------+--------------+------------+---------------------+---------------+---------+--------+
| 1h 4m 47s  |    17414592 |     87031808 | 1840250880 | 2017/10/17 16:42:09 |          -1   | utf8    | ON     |
+------------+-------------+--------------+------------+---------------------+---------------+---------+--------+
1 row in set (0.05 sec)
```

列描述：

```
UPTIME: 服务已经启动时间
USED_MEMORY: 已使用堆内存
TOTAL_MEMORY: 总共的堆内存
MAX_MEMORY: 最大可用堆内存
RELOAD_TIME: 上次config加载时间
ROLLBACK_TIME: 上次config的rollback时间
CHARSET: 字符集
STATUS: 在线状态
```

2.1.3.5 show @@threadpool

show @@threadpool;

描述：展示当前线程池信息

例：

```
mysql> show @@threadpool;
+----------------------+-----------+--------------+-----------------+----------------+------------+
| NAME                 | POOL_SIZE | ACTIVE_COUNT | TASK_QUEUE_SIZE | COMPLETED_TASK | TOTAL_TASK |
+----------------------+-----------+--------------+-----------------+----------------+------------+
| Timer                |         1 |            0 |               0 |          22596 |      22596 |
| BusinessExecutor     |         8 |            1 |               0 |            216 |        217 |
| complexQueryExecutor |         0 |            0 |               0 |              0 |          0 |
+----------------------+-----------+--------------+-----------------+----------------+------------+
3 rows in set (0.03 sec)
```

列描述：

```
NAME: 线程池名称
POOL_SIZE: 线程池大小
ACTIVE_COUNT: 活动数量
TASK_QUEUE_SIZE: 队列中的数量
COMPLETED_TASK: 已完成的任务数量
TOTAL_TASK: 总共任务数量
```

2.1.3.6 show @@database

show @@database;

描述：展示配置的schema名字

结果：略

2.1.3.7 show @@shardingnode

show @@shardingnode;

描述：展示配置的所有shardingnode信息

例：

```
mysql> show @@shardingnode;
+------+----------------+---------------+--------+------+------+---------+---------------+
| NAME | DB_GROUP        | SCHEMA_EXISTS | ACTIVE | IDLE | SIZE | EXECUTE | RECOVERY_TIME |
+------+----------------+---------------+--------+------+------+---------+---------------+
| dn1  | dh1/dble_test  | true          |      0 |    0 | 1000 |      34 |            -1 |
| dn2  | dh2/dble_test  | true          |      0 |    0 | 1000 |      34 |            -1 |
| dn3  | dh1/dble2_test | false         |      0 |    0 | 1000 |      26 |            -1 |
| dn4  | dh2/dble2_test | true          |      0 |    0 | 1000 |      26 |            -1 |
| dn5  | dh1/nosharding | true          |      0 |    0 | 1000 |       9 |            -1 |
+------+----------------+---------------+--------+------+------+---------+---------------+
5 rows in set (0.09 sec)
```

列描述：

```
NAME: 名称
DB_GROUP: dbGroupName/实际schema
SCHEMA_EXISTS: 对应后端物理库是否存在，true为存在，false为不存在。
ACTIVE: 当前活动的后端连接数量
IDLE: 当前空闲的后端连接数量(空闲容量维护疑似bug)
SIZE: maxCon容量
EXECUTE: 有过活动的后端连接数量统计
RECOVERY_TIME: 恢复心跳还需要秒数(stop @@heartbeat 中设置)
```

如果要查看某个schema相关的shardingnode信息，执行：

show @@shardingnode where schema=xxx;

其中，xxx为要查看的schema的名字。

2.1.3.8 show @@dbinstance

show @@dbinstance

描述：展示配置的所有dbinstance信息

例：

```
mysql> show @@dbinstance;
+------------+--------+--------------+------+------+--------+------+------+---------+-----------+------------+----------+
| DB_GROUP   | NAME   | HOST         | PORT | W/R  | ACTIVE | IDLE | SIZE | EXECUTE | READ_LOAD | WRITE_LOAD | DISABLED |
+------------+--------+--------------+------+------+--------+------+------+---------+-----------+------------+----------+
| localhost2 | hostS1 | 10.18x.2x.63 | 3307 | W    |      1 |    9 |  100 |      11 |         0 |          0 |     true |
| localhost1 | hostM1 | 10.18x.2x.64 | 3306 | W    |      1 |    9 |  100 |      17 |         0 |          0 |    false |
+------------+--------+--------------+------+------+--------+------+------+---------+-----------+------------+----------+
2 rows in set (0.09 sec)
```

列描述

```
DB_GROUP:dbinstance所属DB_GROUP
NAME: dbinstance名称
HOST: host名
PORT: 端口
W/R: 读写结点标识
ACTIVE: 当前活动的后端连接数量,按照dbinstance统计
IDLE: 当前空闲的后端连接数量,按照dbinstance统计(空闲容量维护疑似bug)
SIZE: maxCon容量
EXECUTE: 有过活动的后端连接数量统计,按照dbinstance统计
READ_LOAD: 读负载 (请求次数)
WRITE_LOAD: 写负载 (请求次数)
DISABLED: db.xml中dbinstance中的配置 (2.19.09.0以前的版本没有此列，disabled为true的结点不显示)
```

如果要查看某个shardingnode对应的dbinstance信息，执行：

show @@dbinstance where shardingnode=xxx;

其中，xxx为要查看的shardingnode的名字。

2.1.3.9 show @@dbinstance.synstatus

show @@dbinstance.synstatus;

描述：展示当前各dbinstance的同步信息
前提条件: heartbeat 配置了 show slave status（参见db.xml）

例：

```
mysql> show @@dbinstance.synstatus \G
*************************** 1. row ***************************
             DB_GROUP: dbGroup2
                 NAME: instanceM3
                 HOST: 111.231.25.141
                 PORT: 30309
          MASTER_HOST: mysql3
          MASTER_PORT: 3306
          MASTER_USER: replicator
SECONDS_BEHIND_MASTER: 0
     SLAVE_IO_RUNNING: Yes
    SLAVE_SQL_RUNNING: Yes
       SLAVE_IO_STATE: Waiting for master to send event
        CONNECT_RETRY: 10
        LAST_IO_ERROR:
1 row in set (0.00 sec)
```

列描述：

```
DB_GROUP:dbinstance所属DB_GROUP
NAME: dbinstance名称
HOST: 主机名/ip
PORT: 端口
```

其余列含义参见mysql中show slave status的命令。

2.1.3.10 show @@dbinstance.syndetail where name=?

show @@dbinstance.syndetail where name=xxx;

其中，xxx为要查看的dbinstance的名字。

描述：展示24小时内各dbinstance的历次同步信息

例：

```
mysql> show @@dbinstance.syndetail WHERE name =hostM2;
+------------+--------+--------------+------+--------------+-------------+-------------+---------------------+-----------------------+
| DB_GROUP   | NAME   | HOST         | PORT | MASTER_HOST  | MASTER_PORT | MASTER_USER | TIME                | SECONDS_BEHIND_MASTER |
+------------+--------+--------------+------+--------------+-------------+-------------+---------------------+-----------------------+
| localhost2 | hostM2 | 10.18x.2x.64 | 3320 | 10.18x.2x.62 |        3320 | qrep        | 2017-10-17 18:31:27 |                    -1 |
| localhost2 | hostM2 | 10.18x.2x.64 | 3320 | 10.18x.2x.62 |        3320 | qrep        | 2017-10-17 18:31:57 |                    -1 |
| localhost2 | hostM2 | 10.18x.2x.64 | 3320 | 10.18x.2x.62 |        3320 | qrep        | 2017-10-17 18:32:27 |                    -1 |
| localhost2 | hostM2 | 10.18x.2x.64 | 3320 | 10.18x.2x.62 |        3320 | qrep        | 2017-10-17 18:32:57 |                    -1 |
+------------+--------+--------------+------+--------------+-------------+-------------+---------------------+-----------------------+
4 row in set (0.05 sec)
```

列描述:

```
DB_GROUP:dbinstance所属DB_GROUP
NAME: dbinstance名称
HOST: 主机名/ip
PORT: 端口
```

其余列含义参见mysql中show slave status的命令。

2.1.3.11 show @@datasource.cluster

show @@datasource.cluster;

描述：此功能在2.20.04.0 版本已经废除。

2.1.3.12 show @@processor

show @@processor;

描述：展示dble实例的processor信息

例：

```
mysql> show @@processor\G
*************************** 1. row ***************************
        NAME: frontProcessor0
      NET_IN: 0
     NET_OUT: 0
 REACT_COUNT: 0
     R_QUEUE: 0
     W_QUEUE: 0
 FREE_BUFFER: 1072169008
TOTAL_BUFFER: 1073741824
  BU_PERCENT: 0
    BU_WARNS: 0
    FC_COUNT: 0
    BC_COUNT: 0
*************************** 2. row ***************************
        NAME: frontProcessor1
      NET_IN: 0
     NET_OUT: 267
 REACT_COUNT: 0
     R_QUEUE: 0
     W_QUEUE: 0
 FREE_BUFFER: 1072169008
TOTAL_BUFFER: 1073741824
  BU_PERCENT: 0
    BU_WARNS: 0
    FC_COUNT: 0
    BC_COUNT: 0
*************************** 3. row ***************************
        NAME: frontProcessor2
      NET_IN: 0
     NET_OUT: 150
 REACT_COUNT: 0
     R_QUEUE: 0
     W_QUEUE: 0
 FREE_BUFFER: 1072169008
TOTAL_BUFFER: 1073741824
  BU_PERCENT: 0
    BU_WARNS: 0
    FC_COUNT: 0
    BC_COUNT: 0
*************************** 4. row ***************************
        NAME: frontProcessor3
      NET_IN: 0
     NET_OUT: 1548
 REACT_COUNT: 0
     R_QUEUE: 0
     W_QUEUE: 0
 FREE_BUFFER: 1072169008
TOTAL_BUFFER: 1073741824
  BU_PERCENT: 0
    BU_WARNS: 0
    FC_COUNT: 0
    BC_COUNT: 0
...
```

列描述：

```
NAME: 名称
NET_IN: 接收流量
NET_OUT: 发送流量
REACT_COUNT: 固定值0
R_QUEUE: 固定值0
W_QUEUE: 写队列大小
FREE_BUFFER: BufferPool free大小
TOTAL_BUFFER: BufferPool 总大小
BU_PERCENT: BufferPool使用率百分比
BU_WARNS: 固定值0
FC_COUNT: 前端连接数量
BC_COUNT: 后端连接数量
```

2.1.3.13 show @@command

show @@command;

描述：processor对各个类型的数据包的分类统计信息

例：

```
mysql> show @@command;

+------------+---------+-------+--------------+--------------+------------+------+------+------+-------+
| PROCESSOR  | INIT_DB | QUERY | STMT_PREPARE | STMT_EXECUTE | STMT_CLOSE | PING | KILL | QUIT | OTHER |
+------------+---------+-------+--------------+--------------+------------+------+------+------+-------+
| Processor0 |       0 |     0 |            0 |            0 |          0 |    0 |    0 |    0 |     0 |
| Processor1 |       0 |     0 |            0 |            0 |          0 |    0 |    0 |    0 |     0 |
| Processor2 |       0 |     6 |            0 |            0 |          0 |    0 |    0 |    0 |     0 |
| Processor3 |       0 |     4 |            0 |            0 |          0 |    0 |    0 |    0 |     0 |
+------------+---------+-------+--------------+--------------+------------+------+------+------+-------+
4 rows in set (0.00 sec)
```

列描述：

```
PROCESSOR: processor名称
INIT_DB: COM_INIT_DB
QUERY: COM_QUERY
STMT_PREPARE: COM_STMT_PREPARE
STMT_EXECUTE: COM_STMT_EXECUTE
STMT_CLOSE: COM_STMT_CLOSE
PING: COM_PING
KILL: COM_PROCESS_KILL
QUIT: COM_QUIT
OTHER: 其余
```

2.1.3.14 show @@connection where processor=? and front_id=? and host=? and user=?

show @@connection where processor=? and front_id=? and host=? and user=?;

描述：查询前端连接信息，可通过processor，front_id，host和user进行过滤筛选，条件可以任意组合搭配。

例：

```
mysql> show @@connection where processor='frontProcessor4' \G
*************************** 1. row ***************************
            PROCESSOR: frontProcessor4
             FRONT_ID: 4
                 HOST: 192.168.2.190
                 PORT: 9066
           LOCAL_PORT: 52082
                 USER: man1
               SCHEMA:
 CHARACTER_SET_CLIENT: utf8mb4
 COLLATION_CONNECTION: utf8mb4_general_ci
CHARACTER_SET_RESULTS: utf8mb4
               NET_IN: 1438
              NET_OUT: 10925
        ALIVE_TIME(S): 526
          RECV_BUFFER: 32767
           SEND_QUEUE: 0
   TX_ISOLATION_LEVEL:
           AUTOCOMMIT:
        SYS_VARIABLES:
       USER_VARIABLES:
1 row in set (0.01 sec)
```

结果列描述：

```
PROCESSOR: PROCESSOR名称
FRONT_ID: 前端连接ID
HOST: 客户端host
PORT: 本地端口(流量或者管理)
LOCAL_PORT: 客户端端口
USER: 用户
SCHEMA: 所在的schema
CHARACTER_SET_CLIENT: 字符集信息
COLLATION_CONNECTION: 字符集信息
CHARACTER_SET_RESULTS : 字符集信息
NET_IN: 接收流量
NET_OUT: 发送流量
ALIVE_TIME(S): 连接建立时长
RECV_BUFFER: 接收缓冲区大小(字节)
SEND_QUEUE: 发送缓冲区队列大小
TX_ISOLATION_LEVEL: 隔离级别
AUTOCOMMIT: 略
SYS_VARIABLES: 系统变量
USER_VARIABLES: 用户变量
```

2.1.3.15 show @@cache

show @@cache;

描述：展示cache信息

例：

```
mysql> show @@cache;
+-----------------+-------+------+--------+------+------+-------------+----------+
| CACHE           | MAX   | CUR  | ACCESS | HIT  | PUT  | LAST_ACCESS | LAST_PUT |
+-----------------+-------+------+--------+------+------+-------------+----------+
| ER_SQL2PARENTID |  1000 |    0 |      0 |    0 |    0 |             |          |
| SQLRouteCache   | 10000 |    0 |      0 |    0 |    0 |             |          |
+-----------------+-------+------+--------+------+------+-------------+----------+
2 rows in set (0.09 sec)
```

列描述:

```
CACHE: cache名
MAX: 最大容量
CUR: 当前容量
ACCESS: 缓存查询次数
HIT: 命中次数
PUT: 加入缓存计数器
LAST_ACCESS:上一次查询时间戳(格式为yyyy/mm/dd hh:mm:ss)
LAST_INPUT: 上一次加入缓存时间戳(格式为yyyy/mm/dd hh:mm:ss)
```

2.1.3.16 show @@backend where processor=? and backend_id=? and mysqlid=? and host=? and port=?

show @@backend where processor=? and backend_id=? and mysqlid=? and host=? and port=?;

描述：查询活动的后端连接信息，可与show @@session结合使用。该命令可通过processor，backend_id，mysqlid，host和port进行过滤筛选，条件可以任意组合搭配。

例：

```
mysql> show @@backend where processor='backendProcessor9' and host='172.18.0.3' \G
*************************** 1. row ***************************
            processor: backendProcessor9
           BACKEND_ID: 29
              MYSQLID: 26
                 HOST: 172.18.0.3
                 PORT: 3306
       LOCAL_TCP_PORT: 34848
               NET_IN: 93
              NET_OUT: 85
       ACTIVE_TIME(S): 699
               CLOSED: false
                STATE: IDLE
           SEND_QUEUE: 0
               SCHEMA: NULL
 CHARACTER_SET_CLIENT: utf8mb4
 COLLATION_CONNECTION: utf8mb4_general_ci
CHARACTER_SET_RESULTS: utf8mb4
   TX_ISOLATION_LEVEL: 2
           AUTOCOMMIT: true
        SYS_VARIABLES:
       USER_VARIABLES:
             XA_STATUS: 0
             DEAD_TIME:
   USED_FOR_HEARTBEAT: false
1 row in set (0.01 sec)
```

列描述：

```
processor: processor名称
BACKEND_ID: 后端连接ID
MYSQLID: mysql线程id(对应节点上的show processlist里的MYSQLID)
HOST: 主机名
PORT: 端口
LOCAL_TCP_PORT: tcp连接的本地端口
NET_IN: 接收流量大小
NET_OUT: 发送流量大小
ACTIVE_TIME(S): 连接建立时间（单位秒）
CLOSED: 是否被关闭
BORROWED: 是否正在使用
SEND_QUEUE: 发送缓冲队列大小
SCHEMA: schema上下文
CHARACTER_SET_CLIENT: 字符集信息
COLLATION_CONNECTION: 字符集信息
CHARACTER_SET_RESULTS: 字符集信息
TX_ISOLATION_LEVEL: 隔离级别（新建未使用过的连接为-1，表示未初始化）
AUTOCOMMIT: 是否自动提交
SYS_VARIABLES: 系统变量
USER_VARIABLES: 用户变量
XA_STATUS: xa状态
DEAD_TIME:  连接池被回收的时间，连接在完成任务后也会关闭回收
USED_FOR_HEARTBEAT: 是否被用于心跳
```

2.1.3.17 show @@session

show @@session;

描述：展示当前活动前端session的后端连接信息

例：

```
mysql>  show @@session ;
+----------+----------+-------------------------------------------------------------+
| FRONT_ID | DN_COUNT | DN_LIST                                                     |
+----------+----------+-------------------------------------------------------------+
| 2        | 2        | MySQLConnection [backendId=59, lastTime=1508233042917 [,... ]   |
+----------+----------+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

列描述：

```
FRONT_ID: 前端连接ID
DN_COUNT: 后端连接个数:
DN_LIST: 后端连接的详情,
```

DN_LIST例如：

```
MySQLConnection [backendId=59, lastTime=1508233042917, user=xxxx , schema=dble_test, old schema=dble_test, borrowed=true, fromSlaveDB=false, mysqlId=23201,character_set_client=utf8,character_set
_results=utf8,collation_connec tion=utf8_general_ci, txIsolation=3, autocommit=false, attachment=dn2{select * f rom sharding_two_node LIMIT 100}.0, respHandler=com.actiontech.dble.backend.mysq l
.nio.handler.MultiNodeQueryHandler@181db802, host=10.18x.2x.64, port=3320, stat usSync=com.actiontech.dble.backend.mysql.nio.MySQLConnection$StatusSync@d7da548, writeQueue=0, modifiedSQLExecuted
=false] MySQLConnection [id=58, lastTime=1508233042917, user=qrep, schema=dble_test, old schema=dble_test, borrowed=true, fromSlaveDB=false, threadId=11112,character_se t_client=utf8,character_s
et_results=utf8,collation_connection=utf8_general_ci, t xIsolation=3, autocommit=false, attachment=dn1{select * from sharding_two_node L IMIT 100}.0, respHandler=com.actiontech.dble.backend.mysq
l.nio.handler.MultiNode QueryHandler@181db802, host=10.18x.2x.63, port=3320, statusSync=com.actiontech.d ble.backend.mysql.nio.MySQLConnection$StatusSync@5882b3d, writeQueue=0, modified SQLExecu
ted=false]
```

含义：

```
backendId: 后端连接id
lastTime：上次读写时间戳
user: 后端连接对应的用户
schema: 后端连接对应的schema
old_schema: 上次连接被使用时的schema
borrowed: 被使用
fromSlaveDB: 是否是从数据库
mysqlId: 对应后端连接在数据库内的线程id（show processlist）
charset系列: 字符集
txIsolation: 隔离级别
autocommit: 自动提交
attachment: 路由结果集
respHandler: 收到回复时处理的handler类
host: host ip
port: 端口号
statusSync: 同步上下文的类
writeQueue: 写队列
modifiedSQLExecuted: 是否是增删改
```

2.1.3.18 show @@connection.sql

show @@connection.sql;

描述：当前活动session的前端的SQL信息

例：

```
mysql> show @@connection.sql;
+----------+-----------------+------+--------+---------------------+--------------+-----------------------+-----------+
| FRONT_ID | HOST            | USER | SCHEMA | START_TIME          | EXECUTE_TIME | SQL                   | STAGE     |
+----------+-----------------+------+--------+---------------------+--------------+-----------------------+-----------+
|    1     | 0:0:0:0:0:0:0:1 | man  | NULL   | 2017/10/17 17:00:58 |          139 | show @@connection.sql | Read SQL  |
+----------+-----------------+------+--------+---------------------+--------------+-----------------------+-----------+
1 row in set (0.13 sec)
```

列描述:

```
FRONT_ID: 前端连接ID
HOST: 客户端host
USER: 用户
SCHEMA: 所在的schema
START_TIME: 上次接收请求时间戳
EXECUTE_TIME: 响应时间或者未完成SQL持续时间（由于实现方式的原因，可能出现正负20ms的误差）
SQL：如果长度大于1024个字符，将会被截断为1024
STAGE: 运行的当前阶段,结束时会变成finished
```

2.1.3.19 show @@sql

show @@sql;

描述：展示用户近期执行完的50条sql语句(多余的每5秒清理一次)

例：

```
mysql> show @@sql;
+------+------+---------------------+--------------+-----------------------------------------+
| ID   | USER | START_TIME          | EXECUTE_TIME | SQL                                     |
+------+------+---------------------+--------------+-----------------------------------------+
|    1 | root | 2017/10/17 17:37:22 |          381 | select * from sharding_two_node LIMIT 100 |
+------+------+---------------------+--------------+-----------------------------------------+
1 row in set (0.02 sec)
```

列描述:

```
ID: 行号
USER: 用户
START_TIME: 上次接收请求时间戳
EXECUTE_TIME: 响应时间
SQL：略
```

如果需要在查询后重置统计，执行：

show @@sql true;

2.1.3.20 show @@sql.high

show @@sql.high;

描述：展示各个用户的高频sql(容量1024，超过会被定期清理，清理周期5秒)

例：

```
mysql> show @@sql.high;
+------+------+-----------+----------+----------+----------+--------------+---------------------+----------------------------------------+
| ID   | USER | FREQUENCY | AVG_TIME | MAX_TIME | MIN_TIME | EXECUTE_TIME | LAST_TIME           | SQL                                    |
+------+------+-----------+----------+----------+----------+--------------+---------------------+----------------------------------------+
|    1 | root |         1 |      381 |      381 |      381 |          381 | 2017/10/17 17:37:23 | SELECT * FROM sharding_two_node LIMIT ? |
+------+------+-----------+----------+----------+----------+--------------+---------------------+----------------------------------------+
1 row in set (0.06 sec)
```

列描述：

```
ID: 行号
USER: 用户
FREQUENCY: sql曾被执行次数
AVG_TIME: 平均执行耗时
MAX_TIME: 最大执行耗时
MIN_TIME: 最小执行耗时
EXECUTE_TIME: 最近一次执行耗时
LAST_TIME: 最近一次执行时间戳
SQL：略
```

如果需要在查询后重置统计，执行：

show @@sql.high true;

2.1.3.21 show @@sql.slow

show @@sql.slow;

描述：展示执行时间超过给定阈值(默认100毫秒，可通过reload修改)的sql(默认10条，可以通过设置系统参数sqlRecordCount修改，多余的每5秒清理一次）

例：

```
mysql> show @@sql.slow;
+------+---------------------+--------------+-----------------------------------------+
| USER | START_TIME          | EXECUTE_TIME | SQL                                     |
+------+---------------------+--------------+-----------------------------------------+
| root | 2017/10/17 17:37:22 |          381 | select * from sharding_two_node LIMIT 100 |
+------+---------------------+--------------+-----------------------------------------+
1 row in set (0.07 sec)
```

列描述：

```
USER: 用户
START_TIME:上次接收请求时间戳
EXECUTE_TIME:响应时间
SQL：略
```

如果需要在查询后重置统计，执行：

show @@sql.slow true;

2.1.3.22 show @@sql.resultset

show @@sql.resultset;

描述：展示结果集大小超过某个阈值(默认512K，可以通过maxResultSet配置) 的sql，结果集统计信息

例：

```
mysql>  show @@sql.resultset;
+------+------+-----------+-------------------------------+----------------+
| ID   | USER | FREQUENCY | SQL                           | RESULTSET_SIZE |
+------+------+-----------+-------------------------------+----------------+
|    1 | root |         1 | SELECT * FROM sharding_two_node | 1048576        |
+------+------+-----------+-------------------------------+----------------+
1 row in set (0.05 sec)
```

列描述：

```
ID:行号
USER: 用户
FREQUENCY:sql曾被执行次数
SQL: 略
RESULTSET_SIZE:结果集的大小
```

2.1.3.23 show @@sql.sum

show @@sql.sum;

描述：展示用户的sql执行情况, 是否带.user结果是一样的.带参数true，表示查询结束后清空已经缓存的结果

例：

```
mysql> show @@sql.sum;
+------+------+------+------+------+------+--------+---------+--------------+--------------+---------------------+
| ID   | USER | R    | W    | R%   | MAX  | NET_IN | NET_OUT | TIME_COUNT   | TTL_COUNT    | LAST_TIME           |
+------+------+------+------+------+------+--------+---------+--------------+--------------+---------------------+
|    1 | root |    1 |    0 | 1.00 | 1    |     41 |     840 | [0, 0, 1, 0] | [0, 0, 1, 0] | 2017/10/17 17:37:23 |
+------+------+------+------+------+------+--------+---------+--------------+--------------+---------------------+
1 row in set (0.26 sec)
```

列描述:

```
ID:行号
USER:用户
R:读的次数
W:写的次数，恒为零(因为未实现统计)
R%:因为W为0，此值恒为100%
MAX:最大并发数
NET_IN:网络流入量
NET_OUT:网络流出量
TIME_COUNT:query在四个时间区间的个数分布，四个区间分别是前一天22-06 夜间,06-13 上午,13-18下午,18-22 晚间
TTL_COUNT:query耗时在四个时间级别内的个数分布，四个区间分别是10毫秒内,10 - 200毫秒内,1秒内,超过 1秒
LAST_TIME:上次SQL执行时间戳
```

如果需要在查询后重置统计，执行：

show @@sql.sum true;

2.1.3.24 show @@sql.sum.user

等同于：
show @@sql.sum;

如果需要在查询后重置统计，执行：

show @@sql.sum.user true;

2.1.3.25 show @@sql.sum.table

show @@sql.sum.table;

描述：展示各个表的读写情况

例：

```
mysql> show @@sql.sum.table;
+------+-------------------+------+------+------+-----------+-----------+---------------------+
| ID   | TABLE             | R    | W    | R%   | RELATABLE | RELACOUNT | LAST_TIME           |
+------+-------------------+------+------+------+-----------+-----------+---------------------+
|    1 | sharding_two_node |    1 |    0 | 1.00 | NULL      | NULL      | 2017/10/17 17:37:23 |
+------+-------------------+------+------+------+-----------+-----------+---------------------+
1 row in set (0.06 sec)
```

列描述:

```
ID:行号
TABLE:表名(注:解析器实现很简单,可能有bug)
R:读的次数
W:写的次数，恒为零(因为未实现统计)
R%:因为W为0，此值恒为100%
RELATABLE:关联表的名称(目前拆分表关联查询都使用查询计划树,此值为NULL)
RELACOUNT:关联表的个数(目前拆分表关联查询都使用查询计划树,此值为NULL)
LAST_TIME:上次SQL执行时间戳
```

如果需要在查询后重置统计，执行：

show @@sql.sum.table true;

2.1.3.26 show @@heartbeat

show @@heartbeat;

描述：展示dbinstance的heartbeat信息

例：

```
mysql> show @@heartbeat;
+--------+--------------+------+---------+-------+--------+---------+--------------+------------------+-------+-------------+
| NAME   | HOST         | PORT | RS_CODE | RETRY | STATUS | TIMEOUT | EXECUTE_TIME | LAST_ACTIVE_TIME | STOP  | RS_MESSAGE  |
+--------+--------------+------+---------+-------+--------+---------+--------------+------------------+-------+-------------+
| hostM1 | 10.18x.2x.63 | 3320 |      OK |     0 | idle   |       0 | 8,8,8        | NULL             | false | NULL        |
| hostM2 | 10.18x.2x.64 | 3320 |      OK |     0 | idle   |       0 | 9,9,9        | NULL             | false | NULL        |
+--------+--------------+------+---------+-------+--------+---------+--------------+------------------+-------+-------------+
2 rows in set (0.07 sec)
```

列描述：

```
NAME:dbGroup名称
HOST:主机名/IP
PORT:端口
RS_CODE:状态码，有以下四种状态：INIT, OK, ERROR, TIMEOUT
RETRY:重试错误次数
STATUS:checking/idle
TIMEOUT:心跳超时阈值(来源于db.xml中heartbeat子元素timeout的值，默认为0)
EXECUTE_TIME:最近3个时段的平均响应时间，默认1，10，30分钟
LAST_ACTIVE_TIME:上次收到心跳回复时间戳
STOP:是否stop,和stop命令相关
RS_MESSAGE:心跳失败信息，当RS_CODE为INIT, OK, TIMEOUT时，message为null，只有当RS_CODE为ERROR时，message才会显示最近一次心跳失败的信息
```

2.1.3.27 show @@heartbeat.detail where name=？

show @@heartbeat.detail where name=xxx;

其中，xxx为要查询的dbinstance的名字。

描述：展示指定dbinstance的heartbeat的详细信息

前提条件:至少发生过一次心跳语句（与shardingNodeHeartbeatPeriod相关）

例:

```
mysql> show @@heartbeat.detail  where name='hostM1';
+--------+--------------+------+---------------------+--------------+
| NAME   | HOST         | PORT | TIME                | EXECUTE_TIME |
+--------+--------------+------+---------------------+--------------+
| hostM1 | 10.18x.2x.63 | 3320 | 2017-10-17 17:31:58 | 7            |
| hostM1 | 10.18x.2x.63 | 3320 | 2017-10-17 17:32:59 | 9            |
+--------+--------------+------+---------------------+--------------+
2 row in set (0.00 sec)
```

列描述:

```
NAME:dbGroup名称
HOST:主机名/IP
PORT:端口
TIME:收到心跳时间戳
EXECUTE_TIME:心跳执行耗时(毫秒)
```

2.1.3.28 show @@sysparam

show @@sysparam;

描述：展示sysconfig参数配置
结果：略

2.1.3.29 show @@syslog limit=?

show @@syslog limit=N;

其中，N为整数。

描述: 按时间从新到旧展示dble.log中的N条记录内容。
结果：略

2.1.3.30 show @@white

show @@white;

描述：展示配置的白名单信息

例：

```
mysql> show @@white;
+-----------------+------+
| IP              | USER |
+-----------------+------+
| 0:0:0:0:0:0:0:1 | root |
| 127.0.0.1       | root |
| 0:0:0:0:0:0:0:1 | test |
| 127.0.0.1       | test |
+-----------------+------+
4 rows in set (0.00 sec)
```

列描述：
略

2.1.3.31 show @@directmemory

show @@directmemory;

描述：堆外内存使用总览
结果集举例：

```
+---------------------+------------------------+------------------------+
| DIRECT_MEMORY_MAXED | DIRECT_MEMORY_POOL_SIZE | DIRECT_MEMORY_POOL_USED |
+---------------------+------------------------+------------------------+
| 3GB                 | 1024MB                 | 44KB                   |
+---------------------+------------------------+------------------------+
1 row in set (0.16 sec)
```

结果列描述：

```
DIRECT_MEMORY_MAXED:通过-XX:MaxDirectMemorySize设置的值
DIRECT_MEMORY_POOL_SIZE: 内存池的大小，等于bufferPoolPageSize和bufferPoolPageNumber的乘积
DIRECT_MEMORY_POOL_USED：已经使用的内存池中的DirectMemory内存
```

2.1.3.32 show @@command.count

show @@command.count;

描述：查询当前系统的查询数；

结果：略

2.1.3.33 show @@connection.count

show @@connection.count;

描述：查询当前的前端链接数；

结果：略

2.1.3.34 show @@backend.statistics

show @@backend.statistics;

描述：查询系统中进程的后端数据源信息；

例：

```
MySQL [(none)]> show @@backend.statistics;
+---------------+---------+------------+----------+
| HOST          | PORT    | ACTIVE     | TOTAL    |
+---------------+---------+------------+----------+
| 192.168.2.177 | 3307    | 0          | 10       |
| 192.168.2.177 | 3308    | 0          | 10       |
+---------------+---------+------------+----------+
2 rows in set (0.02 sec)
```

列描述：

```
HOST：数据源的ip
PORT：数据源的端口
ACTIVE：数据源正在被使用的链接数
TOTAL ：活的后端链接数。
```

2.1.3.35 show @@backend.old

show @@backend.old;

描述：reload @@config_all之后待回收的活动的后端链接信息
结果格式：同show @@backend

2.1.3.36 show @@binlog.status

show @@binlog.status;

描述：对被分库分表（sharding.xml）使用的mysql节点拉一条一致性的binlog线。

例：

```
mysql> show @@binlog.status;
+-------------------+------------------+----------+--------------+------------------+--------------------------------------------+
| Url               | File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set                          |
+-------------------+------------------+----------+--------------+------------------+--------------------------------------------+
| 10.18x.2x.63:3320 | mysql-bin.000024 | 14128    |              |                  | 7ad71aab-de94-11e5-9488-3a935460da28:1-67646 |
| 10.18x.2x.64:3320 | mysql-bin.000049 | 604440   |              |                  | ba8f8b5c-debf-11e5-a87b-26b8a61f9012:1-91    |
+-------------------+------------------+----------+--------------+------------------+--------------------------------------------+
2 rows in set (0.11 sec)
```

列描述：

```
Url: 后端节点的连接Url值
其余列：等同于在对应结点上执行show master status的结果。
```

2.1.3.37 show @@help

show @@help;

描述：展示帮助信息
结果：略

2.1.3.38 show @@sql.large

show @@sql.large;

描述：展示各个用户的结果集超过10000行的sql(容量为10,多的会被定时清理，清理周期5秒)

例：

```
mysql> show @@sql.large;
+------+-------+---------------------+--------------+---------------------------------------+
| USER | ROWS  | START_TIME          | EXECUTE_TIME | SQL                                   |
+------+-------+---------------------+--------------+---------------------------------------+
| root | 20000 |2017/10/17 17:37:23  | 381          | SELECT * FROM sharding_two_node LIMIT ? |
+------+-------+---------------------+--------------+---------------------------------------+
1 row in set (0.06 sec)
```

列描述:

```
USER: 用户
ROWS: 该查询的行数
START_TIME:上次接收请求时间戳 EXECUTE_TIME:响应时间
SQL：略
```

如果需要在查询后重置统计，执行：

show @@sql.large true;

2.1.3.39 show @@sql.condition

show @@sql.condition;

描述：查询条件统计，需要配合reload @@query_cf 使用，前者设置了table&column后，运行此语句后展示sql查询条件统计信息.（最多100000条，超出后不再统计）
比如select *from sharding_two_node where id =0;* 和*select* from sharding_two_node where id =1;

例：

```
mysql>  show @@sql.condition;
+------+--------------------------------+-------+-------+
| ID   | KEY                            | VALUE | COUNT |
+------+--------------------------------+-------+-------+
|    2 | sharding_two_node.id           | 0     |     1 |
|    3 | sharding_two_node.id           | 1     |     2 |
|    2 | sharding_two_node.id.valuekey  | size  |     2 |
|    3 | sharding_two_node.id.valuecount | total |     3 |
+------+--------------------------------+-------+-------+
4 rows in set (0.05 sec)
```

列描述:

```
ID: 行号
KEY: schema.table 最后两行为schema.table.valuekey 和 schema.table.valuecount
VALUE: 对应key的value值
COUNT: 查询的次数
```

2.1.3.40 show @@cost_time;

show @@cost_time;

描述：查询query耗时统计的结果,需要在bootstrap.cnf中开启useCostTimeStat选项之后才会有统计结果

例：

```
mysql>  show @@cost_time;
+--------------+--------------------------------+----------------------------------+
| OVER_ALL(us) | FRONT_PREPARE                  | BACKEND_EXECUTE                  |
+--------------+--------------------------------+----------------------------------+
|        71496 | Id:9,Time:53135;Id:12,Time:54056 | Id:9,Time:16924;Id:12,Time:16006  |
|        15316 | Id:17,Time:2301;Id:11,Time:3196  | Id:17,Time:10691;Id:11,Time:11397 |
+--------------+--------------------------------+----------------------------------+
2 rows in set (0.05 sec)
```

列描述:

```
OVER_ALL: 总耗时
FRONT_PREPARE: 前端连接以及dble中的耗时
BACKEND_EXECUTE: 后端连接执行耗时
```

2.1.3.41 show @@shardingNodes where schema=? and table=?;

show @@shardingNodes

描述：查询某具体表格的节点信息

例：

```
mysql> show @@shardingNodes where schema=testdb and table=seqtest;
+------+----------+---------------+------+----------------+------+----------+
| NAME | SEQUENCE | HOST          | PORT | PHYSICAL_SCHEMA | USER | PASSWORD |
+------+----------+---------------+------+----------------+------+----------+
| dn1  | 0        | 10.186.24.113 | 3309 | db1            | root | 123456   |
| dn2  | 1        | 10.186.24.113 | 3309 | db2            | root | 123456   |
+------+----------+---------------+------+----------------+------+----------+
2 rows in set (0.05 sec)
```

列描述:

```
NAME:节点名称
SEQUENCE:节点编号
HOST:节点所在的IP
PORT：节点对应的服务端口
PHYSICAL_SCHEMA：节点所对应的物理库
USER：节点连接的用户
PASSWORD：节点连接的密码
```

2.1.3.42 show @@algorithm where schema=? and table=?;

show @@algorithm

描述：查询某具体表格的分片算法信息，由于不同算法会有不同的分片参数以及辅助文件及数据，所以不同算法表格的输出分片事项都不相同

例：

```
mysql> show @@algorithm where schema=testdb and table=seqtest;
+-----------------+-----------------------------------------------------+
| KEY             | VALUE                                               |
+-----------------+-----------------------------------------------------+
| TYPE            | SHARDING TABLE                                      |
| COLUMN          | ID                                                  |
| CLASS           | com.actiontech.dble.route.function.PartitionByLong |
| partitionCount  | 2                                                   |
| partitionLength | 1                                                   |
+-----------------+-----------------------------------------------------+
5 rows in set (0.05 sec)
```

行描述:

```
KEY:分片事项
VALUE：详细信息
```

2.1.3.43 show @@thread_used;

show @@thread_used;

描述：查看各个主要业务处理线程的使用状况

例:

```
mysql> show @@thread_used;
+-------------------------+------------------+-------------+------------------+
| THREAD_NAME             | LAST_QUARTER_MIN | LAST_MINUTE | LAST_FIVE_MINUTE |
+-------------------------+------------------+-------------+------------------+
| BusinessExecutor3       | 0%               | 0%          | 0%               |
| $_NIO_REACTOR_BACKEND-2 | 0%               | 0%          | 0%               |
| BusinessExecutor1       | 0%               | 0%          | 0%               |
| $_NIO_REACTOR_BACKEND-3 | 0%               | 0%          | 0%               |
| $_NIO_REACTOR_BACKEND-0 | 0%               | 0%          | 0%               |
| $_NIO_REACTOR_FRONT-0   | 0%               | 0%          | 0%               |
| $_NIO_REACTOR_BACKEND-1 | 0%               | 0%          | 0%               |
+-------------------------+------------------+-------------+------------------+
7 rows in set (0.00 sec)
```

行描述:

```
THREAD_NAME:线程名称
LAST_QUARTER_MIN：最近15秒使用率
LAST_MINUTE：最近一分钟使用率
LAST_FIVE_MINUTE：最近五分钟使用率
```

2.1.3.44 show @@ddl;

show @@ddl;

描述：查看正在执行，没有在dble内部释放锁的DDL

例:

```
mysql> show @@ddl;
+--------+--------------------+-------------------------------------------------+
| Schema | Table              | Sql                                             |
+----------------------------+-------------------------------------------------+
| testdb | sharding_two_node  | alter table sharding_two_node add column id2 int  |
| mytest | sharding_four_node | drop table sharding_four_node                   |
+--------+--------------------+-------------------------------------------------+
2 rows in set (0.00 sec)
```

行描述:

```
Schema:Schema名称
Table：Table名称
Sql：ddl sql语句
```

2.1.3.45 show @@processlist;

show @@processlist;

描述：查看前端连接和后端连接对应关系，若前端连接没有对应的后端连接，显示NULL。

例:

```
mysql> show @@processlist;
+----------+--------------+---------+------+-----------------+------+---------+------+-------+------+
| Front_Id | shardingNode | MysqlId | User | Front_Host      | db   | Command | Time | State | Info |
+----------+--------------+---------+------+-----------------+------+---------+------+-------+------+
|        1 | dn2          |    2303 | root | 127.0.0.1:33222 | db2  | Sleep   |   17 |       | NULL |
|        2 | NULL         |    NULL | man1 | 127.0.0.1:34882 | NULL | NULL    |    0 |       | NULL |
|        3 | dn3          |    2259 | root | 127.0.0.1:33226 | db1  | Sleep   |    4 |       | NULL |
|        3 | dn2          |    2308 | root | 127.0.0.1:33226 | db2  | Sleep   |    4 |       | NULL |
|        3 | dn1          |    2304 | root | 127.0.0.1:33226 | db1  | Sleep   |    4 |       | NULL |
+----------+--------------+---------+------+-----------------+------+---------+------+-------+------+
5 rows in set (0.05 sec)
```

行描述:

```
Front_Id：前端连接ID
shardingNode：前端连接下发操作的 shardingNode
MysqlId：后端连接对应的 mysql 线程ID
User：用户名
Front_Host：客户端主机名
db：后端连接默认数据库，来自于 mysql 'show processlist' 字段 db
Command：mysql线程正在执行的指令类型，来自于 mysql 'show processlist' 字段 Command
Time：mysql线程处于当前state的时间，来自于  mysql 'show processlist' 字段 Time
State：mysql线程执行状态，来自于 mysql 'show processlist' 字段 State
Info：mysql线程执行语句，来自于 mysql 'show processlist' 字段 Info
```

2.1.3.46 show @@session.xa;

show @@session.xa;

描述：查看后台重试的xa事务信息。

例：

```
mysql> show @@session.xa;
+----------+--------------------+------------------------+------------------+
| FRONT_ID | XA_ID              | XA_STATE               | SHARDING_NODES   |
+----------+--------------------+------------------------+------------------+
| 1        | 'Dble_Server.1.1'  | TX_COMMIT_FAILED_STATE  | dn1,dn3          |
+----------+--------------------+------------------------+------------------+
1 rows in set (0.00 sec)
```

行描述:

```
FRONT_ID：前端连接ID
XA_ID：xa事务id
XA_STATE：xa事务状态
SHARDING_NODES：xa提交失败的shardingNode名称
```

2.1.3.47 show @@reload_status

show @@reload_status

描述：查看dble中最近的reload信息

举例

```
+-------+----------+-------------+---------------+---------------------+---------------------+---------------+------------+
| INDEX | CLUSTER  | RELOAD_TYPE | RELOAD_STATUS | LAST_RELOAD_START   | LAST_RELOAD_END     | TRIGGER_TYPE  | END_TYPE   |
+-------+----------+-------------+---------------+---------------------+---------------------+---------------+------------+
|     0 |No Cluster| RELOAD_ALL  | NOT_RELOADING | 2020/06/19 14:28:04 | 2020/06/19 14:28:05 | LOCAL_COMMAND | RELOAD_END |
+-------+----------+-------------+---------------+---------------------+---------------------+---------------+------------+
```

行描述:

```
INDEX:reload对应的编号，能与日志中的[RL]日志编号相对应
CLUSTER:当前dble使用的集群方式
RELOAD_TYPE:最近的reload的类型 RELOAD_ALL/RELOAD_META/MANAGER_INSERT/MANAGER_UPDATE/MANAGER_DELETE
RELOAD_STATUS:最近一次reload的执行状状态not_reloading/self_reload/meta_reload/waiting_others
LAST_RELOAD_START：起始时间
LAST_RELOAD_END：结束时间
TRIGGER_TYPE：触发类型  LOCAL_COMMAND/CLUSTER_NOTIFY
END_TYPE：结束原因 RELOAD_END/INTERRUPUTED
```

此命令被用于配合命令release @@reload_metadata

2.1.3.48 show @@user

show @@user

描述：查看dble 所有用户

举例

```
mysql> show @@user;
+----------+---------+----------+----------+
| Username | Manager | Readonly | Max_con  |
+----------+---------+----------+----------+
| man1     | Y       | N        | no limit |
| root     | N       | N        | no limit |
| user     | N       | N        | no limit |
+----------+---------+----------+----------+
3 rows in set (0.03 sec)
```

行描述:

```
Username：用户名
Manager：是否是管理用户
Readonly：是否是只读用户
Max_con：最大连接数
```

2.1.3.49 show @@user.privilege

show @@user.privilege

描述：查看dble 用户的权限信息，不包含管理用户

举例

```
mysql> show @@user.privilege;
+----------+---------+-------+--------+--------+--------+--------+
| Username | Schema  | Table | INSERT | UPDATE | SELECT | DELETE |
+----------+---------+-------+--------+--------+--------+--------+
| root     | testdb1 | *     | Y      | Y      | Y      | Y      |
| root     | testdb  | *     | Y      | Y      | Y      | Y      |
| user     | testdb  | *     | N      | Y      | Y      | N      |
+----------+---------+-------+--------+--------+--------+--------+
3 rows in set (0.01 sec)
```

行描述:

```
Username：用户名
Schema：用户授权逻辑库
Table：用户显式指定dml权限的表名，未指定的其他表使用*表示
INSERT：插入权限位
UPDATE：更新权限位
SELECT：查询权限位
DELETE：删除权限位
```

2.1.3.50 show @@data_distribution where table ='schema.table'

show @@data_distribution where table ='schema.table'

描述：查看某个表在各个节点上的数据分布情况

举例

```
+---------------+-------+
| SHARDING_NODE | COUNT |
+---------------+-------
| dn1           | 100   |
| dn2           | 101   |
| dn3           | 98    |
| dn4           | 104   |
+---------------+-------+
4 rows in set (0.09 sec)
```

行描述:

```
SHARDING_NODE：数据结点名字
COUNT：数据量
```

2.1.3.51 show @@Questions

show @@Questions

描述：查看自启动之后SQL服务端口执行的QUERY和Transaction数量

举例：

```
mysql> show @@Questions;
+-----------+--------------+
| Questions | Transactions |
+-----------+--------------+
|         0 |            0 |
+-----------+--------------+
```

行描述:

```
Questions:收到的查询的数量
Transactions：执行事务的数量，非事务查询算单语句事务
```

2.1.3.52 show @@connection_pool

show @@connection_pool

描述：查看后端连接池的各种属性

举例：

```
mysql> show @@connection_pool;
+----------+-------------+-----------------------------+--------+
| DB_GROUP | DB_INSTANCE | PROPERTY                    | VALUE  |
+----------+-------------+-----------------------------+--------+
| dbGroup1 |  instanceM1 | minCon                      |      2 |
| dbGroup1 |  instanceM1 | maxCon                      |      4 |
| dbGroup1 |  instanceM1 | testOnCreate                |  false |
| dbGroup1 |  instanceM1 | testOnBorrow                |  false |
| dbGroup1 |  instanceM1 | testOnReturn                |  false |
| dbGroup1 |  instanceM1 | testWhileIdle               |  false |
| dbGroup1 |  instanceM1 | connectionHeartbeatTimeout  |     20 |
| dbGroup1 |  instanceM1 | connectionTimeout           |  10000 |
| dbGroup1 |  instanceM1 | heartbeatPeriodMillis       |  10000 |
| dbGroup1 |  instanceM1 | idleTimeout                 | 600000 |
| dbGroup1 |  instanceM1 | evictorShutdownTimeoutMillis |  10000 |
| dbGroup1 |  instanceM1 | timeBetweenEvictionRunsMillis |  30000 |
+----------+-------------+-----------------------------+--------+
12 rows in set (0.01 sec)
```

行描述:

```
DB_GROUP：dbinstance所属DB_GROUP
DB_INSTANCE：dbinstance名
PROPERTY：属性名
VALUE：属性值
```

2.1.3.52 show @@cap_client_found_rows

show @@cap_client_found_rows

描述：查看cap_client_found_rows权能标志

举例：

```
mysql> show @@cap_client_found_rows;
+-------------------------+
| @@cap_client_found_rows |
+-------------------------+
| 0                       |
+-------------------------+
1 row in set (0.02 sec)
```

值描述:

```
0-关闭，1-开启
```

2.1.4 switch命令

本小节部分自2.20.04.0 版本起废弃

2.1.5 kill命令

2.1.5.1 kill @@connection;

kill @@connection id1,id2,...;

其中，idx为 前端连接id值，可以通过show @@connection 获取。

描述：关闭存在的前端链接 ，对不存在的前端链接，不会报错

结果：返回OK，关闭的前端链接数。

2.1.5.2 kill @@xa_session;

kill @@xa_session id1,id2,...;

其中，idx为 session id值，可以通过show @@session.xa 获取。

描述：取消指定session后台重试xa事务，对不存在的session，不会报错

结果：返回OK，取消的session数量。

2.1.5.3 KILL @@DDL_LOCK where schema=? and table=?

KILL @@DDL_LOCK where schema=? and table=?;

描述：释放指定schema下table的ddl锁，详细描述可参考 2.22 KILL @@DDL_LOCK

结果：返回OK。

2.1.6 stop命令

2.1.6.1 stop @@heartbeat

stop @@heartbeat keys:dbGroup；

其中：keys:dbGroup名字列表，可以是多个，用逗号隔开的key；该命令停止的是每个dbGroup的写实例。 **key**:可以直接是dbGroup，也可以是dbGroup$0-n （如dbGroup$0-2实际会输出dbGroup[0],dbGroup[1],dbGroup[2]，疑似BUG）；
value: 应当是个整数，单位毫秒
描述：设置dbGroup名为key的host停止heartbeat n秒
结果：返回OK

注意：未作异常处理，慎用

2.1.7 reload 命令

2.1.7.1 reload @@config

reload @@config;

2.19.09.0(不含**)**版本以前请参考对应章节文档，**2.19.09.0**版本之后功能完全等同于**reload @@config_all**

2.1.7.2 reload @@config_all

reload @@config_all [-s] [-f] [-r];

描述：重新加载所有配置，涉及user.xml，db.xml，sharding.xml 内容，

-s 跳过测试后端链接，默认不加此参数会对配置中的后端链接进行测试，测试失败将返回ERROR；
-f 关闭所有变更的dbGroup（加-r参数所有dbGroup会被视为变更）相关的处于事务中的前端链接,如果无此参数默认仅将相应后端链接放入旧链接池。
-r 不做智能判断,将所有后端连接池全部重新加载一遍。不加此参数时，将对新配置进行智能判断，只会对增删改的连接池做变更，不影响未作变更的连接池
更多细节 参考 2.19 智能计算reload_all
结果：OK 或者ERROR

相关影响：当执行此命令时，当有以下情况发生时，涉及到的表的meta信息会被重载，否则保持原有表meta信息。

- 有新增的表
- 有删除的表
- 表的shardingNode或者类别发生变更
- 表的shardingNode对应的物理节点或者对应的dbGroup/dbInstance发生变更
- 有新增的schema
- 有删除的schema
- schema 的默认shardingNode属性发生变更
- schema的shardingNode对应的物理节点或者对应的dbGroup/dbInstance发生变更。

另外，如果包含-r参数则不做上述判断，全部重新加载meta数据。
如果不包含-r但是包含-s参数,则对metadata是否需要重新加载的计算时，忽略所有dbGroup/dbInstance的变更

注意,不能在配置变更中体现的的某些变化是无法重新加载metadata的，举例#1002
一个带有默认shardingNode的schema尝试通过删除配置将拆分表或者global表变成非拆分表是不符合规范的。应当避免这种操作。

注意：如果使用默认的切换方式(即单实例部署并且system的outerHA属性为false), 需要做配置的重载时,需要人工保证流程是标准的,否则可能导致切换功能故障,具体请参看相关章节2.12 故障切换。

2.1.7.3 reload @@metadata

reload @@metadata;

描述：重新加载所有元数据信息。
结果：返回OK。

支持过滤表达式

reload @@metadata where schema=? [and table=?]

描述：重新加载指定schema中所有表或指定表的元数据信息。
结果：返回OK。

reload @@metadata where table in ('schema1.table1' ,'schema2.table2','schema1.table3',...)

描述：重新加载schema1中table1,table3和schema2中table2的元数据信息。
结果：返回OK。

2.1.7.4 reload @@sqlslow=N;

reload @@sqlslow=N;

描述：设定用户分析统计的slow sql时间阈值到N毫秒； 结果：OK

2.1.7.5 reload @@user_stat

reload @@user_stat;

描述：重置用户状态统计结果。
影响的命令:
show @@sql;
show @@sql.sum;
show @@sql.slow;
show @@sql.high;
show @@sql.large;
show @@sql.resultset;

结果：OK

2.1.7.6 reload @@query_cf

reload @@query_cf[=table&column];

其中，table为要统计的目标表的表名，column为目标表中目标列的列名。
描述：重设show @@sql.conditiont要统计的条件。
结果：OK

如果要清除查询条件统计表列的设置执行命令：

reload @@query_cf;

或者

reload @@query_cf=NULL;

2.1.8 rollback命令

2.1.8.1 rollback @@config

rollback @@config;

描述：针对reload @@config 或者reload @@config_all 操作，回滚为原有配置信息, 注意可能与配置文件不同，是内存级别的修改。

结果：OK或者ERROR

2.1.9 offline命令

offline;

描述：设置dble处于离线状态, 之后外部对dble的ping命令，select user，心跳都会返回错误。
问题：其他查询未处理，合理性需要考量
结果：OK

2.1.10 online命令

online;

描述：设置dble处于在线状态，与offline相反
结果：OK

2.1.11 file命令

2.1.11.1 file @@list

描述：列出conf目录下文件
例：

```
MySQL [(none)]> file @@list;
+-------------------------------------------------------+
| DATA                                                  |
+-------------------------------------------------------+
| 1 : wrapper.conf  time:2019-10-17 11:13               |
| 2 : myid.properties  time:2019-10-17 11:11            |
+-------------------------------------------------------+
2 rows in set (0.00 sec)
```

2.1.11.2 file @@show filename

其中，filename为目标文件的文件名
描述：展示配置文件filename的内容 结果：单列，其中每行为conf/filename的每行内容
结果：略

2.1.11.3 file @@upload filename content

描述：加载content写到文件fileName中，fileName前后必须以空格隔开，xml文件会做校验。
结果：单行单列，返回加载是否成功

2.1.12 log命令

log @@[file=logname limit=numberOfRow key=keyword regex=regex]

描述：展示logs下文件logname的信息。格式错误导致参数列表为空，展示所有文件的信息。

- logname：不指定默认为dble.log
- rowLimit：以逗号为分隔符，不指定默认为0:10000
- keyWord :筛选日志中包含key,如果需要筛选的key中包含空格，允许使用'key word'形式
- regexStr :筛选符合正则表达式的结果 ，注意regexStr中不允许包含空格，如有需要使用\s进行替代

结果：单列，按行展示logs下文件logname的信息。

2.1.13 配置检查命令dryrun

背景；当我们修改配置之后并且reload之前，可以通过dryrun来检查配置的正确性。

举例，当这样一个sharding.xml 准备reload时，先观察dryrun结果： sharding.xml:

```
<?xml version="1.0"?>
<!DOCTYPE dble:sharding SYSTEM "sharding.dtd">
<dble:sharding xmlns:dble="http://dble.cloud/">
    <schema name="testdb" sqlMaxLimit="100" >
        <shardingTable name="sharding_two_node" shardingNode="dn1,dn2" shardingColumn="id" function="two-long" />
        <shardingTable name="sharding_two_node2" shardingNode="dn1,dn2" shardingColumn="id" function="two-long" />
        <shardingTable name="sharding_two_node3" shardingNode="dn1,dn2" shardingColumn="id" function="two-long" />
        <shardingTable name="sharding_four_node" shardingNode="dn1,dn2,dn3,dn4" shardingColumn="id" function="rule_simple"  />
        <globalTable name="test_table"shardingNode="dn$1-2"/>
        <shardingTable name="a_test" shardingNode="dn1,dn2,dn3,dn4" shardingColumn="id" function="rule_simple"  />
        <shardingTable name="a_order" shardingNode="dn1,dn2,dn3,dn4" shardingColumn="id" function="rule_simple"  />
        <shardingTable name="test_shard" shardingNode="dn1,dn2,dn3,dn4" shardingColumn="id" function="rule_simple" />
        <globalTable name="test_global" shardingNode="dn1,dn2,dn3,dn4"/>
        <shardingTable name="sbtest1" shardingNode="dn1,dn2,dn3,dn4" shardingColumn="id" function="rule_simple"  />
    </schema>
    <schema name="nosharding_test" sqlMaxLimit="100" shardingNode="dn5">
    </schema>
    <shardingNode name="dn1" dbGroup="dh1" database="ares_test" />
    <shardingNode name="dn2" dbGroup="dh2" database="dble_test" />
    <shardingNode name="dn3" dbGroup="dh1" database="mycat_test" />
    <shardingNode name="dn4" dbGroup="dh2" database="mycat_test" />
    <shardingNode name="dn5" dbGroup="dh1" database="nosharding" />
    <shardingNode name="dn8" dbGroup="dh1" database="xxxoooxxx" />

<function name="rule_simple" class="Hash">
        <property name="partitionCount">4</property>
        <property name="partitionLength">1</property>
    </function>
</dble:sharding>
<function class="Hash" name="two-long">
        <property name="partitionCount">2</property>
        <property name="partitionLength">1</property>
    </function>
```

dryrun结果如下：

```
mysql> dryrun;
+------+---------+----------------------------+
| TYPE | LEVEL   | DETAIL                     |
+------+---------+----------------------------+
| Xml  | WARNING | shardingNode dn9 is useless |
| Xml  | WARNING | shardingNode dn8 is useless |
+------+---------+----------------------------+
2 rows in set (0.58 sec)
```

列名含义：
TYPE: 错误类型，比如XML表示xml配置错误，BACKEND表示后端连接错误
LEVAL:错误级别：分为WARNNING 和ERROR表,一般来说WARNNING错误不影响启动和使用，但需要注意。
DETAIL :错误详情

2.1.14 shardingNode级别的流量暂停和恢复功能

背景：当我们做部分shardingNode的拆分或者是数据重分片时，有时希望不影响到其他无关的业务所涉及到的dbGroup和shardingNode，所以希望只停下部分shardingNode的流量。

2.1.14.0 功能描述

pause功能会在dble不停止的状态下停止对于指定后端节点的流量，在暂停期间的所有涉及到节点的路由结果会被挂起（为防止挂起的连接过多，queue和wait_limit参数会控制挂起连接的数量和时间），直到恢复命令resume被执行之后,之前被挂起的查询才会继续进行

pause的执行不一定成功，pause命令将会在指定的timeout的时间内等待需要暂停的流量上的所有事务或者正在执行的sql结束，当目标节点上来自dble的sql或者事务一直无法在指定时间内结束的时候，本次暂停会返回失败。

pause（暂停）是一种dble server级别的全局状态，暂停节点不可追加或者逐步恢复，只能一起暂停指定的一个或几个shardingNode并一起恢复流量，并且暂停期间执行reload命令不会恢复对应的流量状态。但是暂停操作不会被记录在文件之中，若发生dble重启，则暂停状态会被重置。
这里推荐谨慎的使用暂停并选择影响的范围，推荐逐个变动暂停，reload，恢复，再进行下一个变动

2.1.14.1 暂停流量：

`pause @@shardingNode = 'dn1,dn2' and timeout = 10 ,queue = 10,wait_limit = 10;`

参数描述:
timeout:这个参数是等待涉及到的事务完成的时间，如果在到达timeout之后，仍然有事务未完成，本次pause失败，单位秒。
queue:这个参数表示暂停期间的阻塞前端连接的数量，超过此数量时，前端连接建立将会发生错误。
wait_limit:是针对被阻塞的每个单个的前端的时间限制，如果被阻塞了超过wait_limit的时间，将会返回错误，单位秒。

2.1.14.2 恢复流量：

`RESUME;` 返回正常的OK 或者错误“No shardingNode paused”

2.1.14.3 查看当前暂停状态：

`show @@pause;`

```
mysql> show @@pause;
+---------------------+
| PAUSE_SHARDING_NODE |
+---------------------+
| dn1                 |
| dn2                 |
+---------------------+
2 rows in set (0.15 sec)
```

另外，商业版本支持集群操作，开源版本后续也将支持

2.1.15 慢日志相关命令

慢日志相关的命令:

2.1.15.1 查询慢查询日志的开启状态

```
mysql> show @@slow_query_log;
+------------------+
| @@slow_query_log |
+------------------+
| 0                |
+------------------+
1 row in set (0.00 sec)
```

2.1.15.2 开启慢查询日志

```
mysql> enable @@slow_query_log;
Query OK, 1 row affected (0.09 sec)
enable slow_query_log success
```

2.1.15.3 关闭慢查询日志

```
mysql> disable @@slow_query_log;
Query OK, 1 row affected (0.03 sec)
disable slow_query_log success
```

2.1.15.4 查看慢查询日志统计阈值

```
mysql> show @@slow_query.time;
+-------------------+
| @@slow_query.time |
+-------------------+
| 100               |
+-------------------+
1 row in set (0.00 sec)
```

2.1.15.5 修改慢查询日志统计阈值

```
mysql> reload @@slow_query.time=200;
Query OK, 1 row affected (0.10 sec)
reload @@slow_query.time success

mysql> show @@slow_query.time;
+-------------------+
| @@slow_query.time |
+-------------------+
| 200               |
+-------------------+
1 row in set (0.00 sec)
```

2.1.15.6 查看慢查询日志刷盘周期

```
mysql> show @@slow_query.flushperiod;
+--------------------------+
| @@slow_query.flushperiod |
+--------------------------+
| 1                        |
+--------------------------+
1 row in set (0.00 sec)
```

2.1.15.7 修改慢查询日志刷盘周期

```
mysql> reload @@slow_query.flushperiod=2;
Query OK, 1 row affected (0.05 sec)
reload @@slow_query.flushPeriod success

mysql> show @@slow_query.flushperiod;
+--------------------------+
| @@slow_query.flushperiod |
+--------------------------+
| 2                        |
+--------------------------+
1 row in set (0.00 sec)
```

2.1.15.8 查看慢查询日志刷盘条数阈值

```
mysql> show @@slow_query.flushsize;
+------------------------+
| @@slow_query.flushsize |
+------------------------+
| 1000                   |
+------------------------+
1 row in set (0.01 sec)
```

2.1.15.9 修改慢查询日志刷盘条数阈值

```
mysql> reload @@slow_query.flushsize=1100;
Query OK, 1 row affected (0.03 sec)
reload @@slow_query.flushSize success

mysql> show @@slow_query.flushsize;
+------------------------+
| @@slow_query.flushsize |
+------------------------+
| 1100                   |
+------------------------+
1 row in set (0.00 sec)
```

2.1.15.10 查看某个连接的当前执行状态

show @@connection.sql.status where FRONT_ID= ?; 此功能需要开启慢日志才有效，当对应的连接当前query已经执行完毕时，执行此命令的结果与 `trace` 功能相同。 如果query正在执行，本结果将试图展示query执行到哪一个步骤了。 例如，广播查询

```
mysql> show @@connection.sql.status where FRONT_ID= 1;
+--------------------------+-------------+-------------+--------------+---------------+----------------------------+
| OPERATION                | START(ms)   | END(ms)     | DURATION(ms) | SHARDING_NODE | SQL/REF                    |
+--------------------------+-------------+-------------+--------------+---------------+----------------------------+
|                 Read_SQL | 0.0         | 0.082598    | 0.082598     | -             | -                          |
|                Parse_SQL | 0.082598    | 0.676424    | 0.593826     | -             | -                          |
|        Route_Calculation | 0.676424    | 0.895382    | 0.218958     | -             | -                          |
| Prepare_to_Push/Optimize | 0.895382    | 6743.838628 | 6742.943246  | -             | -                          |
|              Execute_SQL | 6743.838628 | 6753.488422 | 9.649794     | dn1           | select * from sharding_4_t1 |
|              Execute_SQL | 6743.838628 | 6751.472835 | 7.634207     | dn3           | select * from sharding_4_t1 |
|              Execute_SQL | 6743.838628 | 6750.981646 | 7.143018     | dn4           | select * from sharding_4_t1 |
|              Execute_SQL | 6743.838628 | 6753.31394  | 9.475312     | dn2           | select * from sharding_4_t1 |
|             Fetch_result | 6753.488422 | 6754.383316 | 0.894894     | dn1           | select * from sharding_4_t1 |
|             Fetch_result | 6751.472835 | 6751.656604 | 0.183769     | dn3           | select * from sharding_4_t1 |
|             Fetch_result | 6750.981646 | 6751.188385 | 0.206739     | dn4           | select * from sharding_4_t1 |
|             Fetch_result | 6753.31394  | 6754.286055 | 0.972115     | dn2           | select * from sharding_4_t1 |
|          Write_to_Client | 6750.981646 | unfinished  | unknown      | -             | -                          |
+--------------------------+-------------+-------------+--------------+---------------+----------------------------+
13 rows in set (0.04 sec)
```

再比如join

```
mysql> show @@connection.sql.status where FRONT_ID= 1;
+--------------------------+-------------+-------------+--------------+-------------------+-------------------------------------------------------------------------------------+
| OPERATION                | START(ms)   | END(ms)     | DURATION(ms) | SHARDING_NODE     | SQL/REF                                                                             |
+--------------------------+-------------+-------------+--------------+-------------------+-------------------------------------------------------------------------------------+
|                 Read_SQL | 0.0         | 0.039588    | 0.039588     | -                 | -                                                                                   |
|                Parse_SQL | 0.039588    | 0.756578    | 0.71699      | -                 | -                                                                                   |
|        Route_Calculation | 0.756578    | 1.5547      | 0.798122     | -                 | -                                                                                   |
| Prepare_to_Push/Optimize | 1.5547      | 3.428551    | 1.873851     | -                 | -                                                                                   |
|              Execute_SQL | 3.428551    | 2362.10579  | 2358.677239  | dn1_0             | select `a`.`age`,`a`.`id`,`a`.`name` from  `sharding_4_t1` `a` ORDER BY `a`.`id` ASC |
|             Fetch_result | 2362.10579  | unfinished  | unknown      | dn1_0             | select `a`.`age`,`a`.`id`,`a`.`name` from  `sharding_4_t1` `a` ORDER BY `a`.`id` ASC |
|              Execute_SQL | 3.428551    | 2362.122407 | 2358.693856  | dn2_0             | select `a`.`age`,`a`.`id`,`a`.`name` from  `sharding_4_t1` `a` ORDER BY `a`.`id` ASC |
|             Fetch_result | 2362.122407 | unfinished  | unknown      | dn2_0             | select `a`.`age`,`a`.`id`,`a`.`name` from  `sharding_4_t1` `a` ORDER BY `a`.`id` ASC |
|              Execute_SQL | 3.428551    | 2362.307153 | 2358.878602  | dn3_0             | select `a`.`age`,`a`.`id`,`a`.`name` from  `sharding_4_t1` `a` ORDER BY `a`.`id` ASC |
|             Fetch_result | 2362.307153 | unfinished  | unknown      | dn3_0             | select `a`.`age`,`a`.`id`,`a`.`name` from  `sharding_4_t1` `a` ORDER BY `a`.`id` ASC |
|              Execute_SQL | 3.428551    | 2364.523615 | 2361.095064  | dn4_0             | select `a`.`age`,`a`.`id`,`a`.`name` from  `sharding_4_t1` `a` ORDER BY `a`.`id` ASC |
|             Fetch_result | 2364.523615 | unfinished  | unknown      | dn4_0             | select `a`.`age`,`a`.`id`,`a`.`name` from  `sharding_4_t1` `a` ORDER BY `a`.`id` ASC |
|          MERGE_AND_ORDER | 2362.639012 | unfinished  | unknown      | merge_and_order_1 | dn1_0; dn2_0; dn3_0; dn4_0                                                          |
|            SHUFFLE_FIELD | 4178.383366 | unfinished  | unknown      | shuffle_field_1   | merge_and_order_1                                                                   |
|              Execute_SQL | 3.428551    | 2365.71371  | 2362.285159  | dn1_1             | select `b`.`id` from  `sharding_2_t1` `b` ORDER BY `b`.`id` ASC                     |
|             Fetch_result | 2365.71371  | unfinished  | unknown      | dn1_1             | select `b`.`id` from  `sharding_2_t1` `b` ORDER BY `b`.`id` ASC                     |
|              Execute_SQL | 3.428551    | 2365.952707 | 2362.524156  | dn2_1             | select `b`.`id` from  `sharding_2_t1` `b` ORDER BY `b`.`id` ASC                     |
|             Fetch_result | 2365.952707 | unfinished  | unknown      | dn2_1             | select `b`.`id` from  `sharding_2_t1` `b` ORDER BY `b`.`id` ASC                     |
|          MERGE_AND_ORDER | 2366.164823 | unfinished  | unknown      | merge_and_order_2 | dn1_1; dn2_1                                                                        |
|            SHUFFLE_FIELD | not started | unfinished  | unknown      | -                 | -                                                                                   |
|                     JOIN | not started | unfinished  | unknown      | -                 | -                                                                                   |
|            SHUFFLE_FIELD | not started | unfinished  | unknown      | -                 | -                                                                                   |
|          Write_to_Client | not started | unfinished  | unknown      | -                 | -                                                                                   |
+--------------------------+-------------+-------------+--------------+-------------------+-------------------------------------------------------------------------------------+
23 rows in set (0.04 sec)
```

2.1.16 创建/删除物理数据库命令

2.1.16.1 创建物理数据库

用于dble启动后发现有些shardingNode对应的物理库还未建立，可以使用后端命令一次性建立。
命令格式:
`create database @@shardingNode ='dn.....'`

shardingNode值支持 `dn$1-4` 这种形式。

当所包含的shardingNode至少有一个不在配置文件当中时，返回错误：shardingNode $Name does not exists.

否则会对所有 shardingNode 执行 create database if not exists $databaseName，执行完成之后返回OK。

2.1.16.2 删除物理数据库

用于删除某些shardingNode对应的物理库，可以使用后端命令一次性删除。
命令格式:
`drop database @@shardingNode ='dn.....'`

shardingNode值支持 `dn$1-4` 这种形式。

当所包含的shardingNode至少有一个不在配置文件当中时，返回错误：shardingNode $Name does not exists.

否则会对所有 shardingNode 执行 drop database if exists $databaseName，执行完成之后返回OK。若在执行过程中发生错误，会将show @@shardingNode里面的SCHEMA_EXISTS置为false，需要用户人工确认下是否物理库已删除成功。

check系列命令

2.1.17.0 check @@metadata 命令

用于检查meta信息是否存在以及加载的时间。
命令格式:

- 第一种形式 `check @@metadata`
  - 返回上一次 reload @@metadata 的datatime（或者上一次 reload @@config_all的datatime 或者启动时加载meta的datatime）
- 第二种形式 `check full @@metadata` ,并且支持以下过滤条件:
  - `where schema=? and table=?`
  - `where schema=?`
  - `where reload_time='yyyy-MM-dd HH:mm:ss'` , `where reload_time>='yyyy-MM-dd HH:mm:ss'` , `where reload_time<='yyyy-MM-dd HH:mm:ss'`
  - `where reload_time is null`
  - `where consistent_in_sharding_nodes=0`
  - `where consistent_in_sharding_nodes = 1`
  - `where consistent_in_memory=0`
  - `where consistent_in_memory = 1`
  - If no where, retrun all results.
- `check full @@metadata` 结果集如下:

schema	table	reload_time	table_structure	consistent_in_sharding_nodes	consistent_in_memory
schema	table	2018-09-18 11:01:04	CREATE TABLE table`(.....	1	1

column **table_structure** 和 `show create table` 命令结果的形式一样
column **consistent_in_sharding_nodes** 表示不同分片之间的一致性，0为不一致，1为一致
column **consistent_in_memory** 表示内存中meta与实际后端结点的一致性，0为不一致，1为一致

当table_structure列为null时，consistent_in_sharding_nodes列和consistent_in_memory列没有意义。
当consistent_in_sharding_nodes为0时，consistent_in_memory没有意义。

2.1.17.1 check @@global schema = '' [and table = '']

用于进行手动全局表检查的命令，当即触发一次特定范围的全局表检查，并且将检查结果作为返回值进行展示。
结果如下所示：

```
mysql> check @@global schema = 'testdb';
+--------+------------+----------------------------+-------------------+
| SCHEMA | TABLE      | DISTINCT_CONSISTENCY_NUMBER | ERROR_NODE_NUMBER |
+--------+------------+----------------------------+-------------------+
| testdb | tb_global1 |                          0 |                 0 |
+--------+------------+----------------------------+-------------------+
```

SCHEMA: 所检查的SCHEMA名字
TABLE: 所检查的TABLE名字
DISTINCT_CONSISTENCY_NUMBER： 返回有几个不同的检查结果的版本
ERROR_NODE_NUMBER： 在检查过程中有几个节点执行SQL报错
具体最终结果是否符合用户的预期，还是要根据表格的检查SQL和返回信息来进行判断，但一般情况下认为当DISTINCT_CONSISTENCY_NUMBER的值大于1的情况下，表格中的内容一定不一致

2.1.18 release 命令

2.1.18.1 release @@reload_metadata

描述：打断正在进行中的reload_metadata步骤。当reload/reload config_all/reload metadata/rollback config进行到最后一个步骤时，都会进行reload metadata
就经验而言这个步骤存在hang死的风险，所以当出现reload metadata步骤迟迟未返回的情况，可以使用这个命令进行终止，使得dble能够继续提供服务 结果：OK 或者ERROR

注意：

- 此命令会导致config可能和table meta不匹配的情况，若执行此命令，请务必在后面追加命令reload @@metadata更新最新的meta信息
- 当一个reload过程被打断时，执行reload的进程会返回编号为5999的SQL错误，请慎重对待此错误CODE，这个错误代表着“新的配置被应用，meta信息在更新的过程中终止”
- 在执行运维操作的过程中可能会需要查看dble当前的reload状态，可通过具体命令show @@reload_status进行查看

2.1.19 split 命令

背景

在进行POC时，现场人员进行数据导入时经常遇到各种问题，比较典型的是dble在导入文件时，对部分sql语句的不支持。另外，未分片的历史数据通过dble导入，旧数据会路由分片，在数据量较大时，耗时会比较长，在此过程中出现错误的话也很难排查。 基于以上原因，2.19.09.0版本提供工具对mysqldump导出的源数据文件按照后端分片节点进行分割。分割后的数据文件可以在每个后端分片执行导入，适配数据按分片导入的需求。

dump文件语句处理

1. create database：会将逻辑数据库转换为物理库。
2. ddl语句：根据表的分片节点写入到对应后端节点的dump文件中，对于自增列，会将自增列的数据类型修改为bigint。
3. insert：全局序列列值会被dble替换为全局序列，再按照拆分列根据拆分算法路由到对应后端节点的dump文件中。
4. 一些属性设置的语句会根据最近一次解析的ddl来决定下发到具体的后端节点的dump文件中。
5. 会跳过对视图的处理
6. 会跳过对子表的处理

使用方法

命令

在管理端口执行以下命令：

```
mysql > split src dest [-sschema] [-r500] [-w500] [-l10000] [--ignore]

- src：表示原始dump文件名
- dest：表示生成的dump文件存放的目录
- -s：表示默认逻辑数据库名，当dump文件中不包含schema的相关语句时，会默认导出到该schema
- -r：表示设置读文件队列大小，默认500
- -w：表示设置写文件队列大小，默认500
- -l：表示split后一条insert中最多包含的values，只针对分片表,默认4000
- --ignore：insert时，忽略已存在的数据
```

生成的分片文件以格式：源文件名-shardingNode名-时间戳.dump，最新的文件时间戳最大。

例如：我的原始dump文件是 /tmp/mysql_dump.sql ，我想切分以后输出到/tmp/dump/目录下： 命令就是：

```
split /tmp/mysql_dump.sql /tmp/dump/
```

日志

默认情况下，split过程中生成的日志打印到在dble.log中，提供配置让split命令产生的日志单独存放，若需要开启，则需修改log4j.xml文件。

```
<Configuration status="WARN">
  <Appenders>
    <!-- 将下面的此段配置追加至已安装dble的log4j.xml中的Appenders下 -->
    <RollingFile name="DumpFileLog" fileName="logs/dump.log"
                 filePattern="logs/$${date:yyyy-MM}/dump-%d{MM-dd}-%i.log.gz">
      <PatternLayout>
        <Pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %5p [%t] (%l) - %m%n</Pattern>
      </PatternLayout>
      <Policies>
        <OnStartupTriggeringPolicy/>
        <SizeBasedTriggeringPolicy size="250 MB"/>
        <TimeBasedTriggeringPolicy/>
      </Policies>
      <DefaultRolloverStrategy max="10"/>
    </RollingFile>
  </Appenders>
  <Loggers>
    <!-- 将下面的此段配置追加至已安装dble的log4j.xml中的Loggers下，可通过调整level为debug来调整性能 -->
    <Logger name="dumpFileLog" level="info" additivity="false" includeLocation="false" >
      <AppenderRef ref="DumpFileLog" />
      <AppenderRef ref="RollingFile"/>
    </Logger>
  </Loggers>
</Configuration>
```

可通过日志中的 “dump file has bean read d%”关键字来查看解析进度。

可开启日志的debug级别来调整性能

任务停止

执行dump file任务的管理连接不受 idletimeout 参数的限制。用户可以通过kill @@connection id 方式杀掉管理连接从而停止dump file的任务的执行。

使用限制

1. 数据导入之后需要运维检查下数据完整性。
2. 对于使用全局序列的表，表原先全局序列中的值会被擦除，替换成全局序列，需要注意。
3. 暂时不支持子表的dump操作。

2.1.20 flow_control 命令

2.1.20.1 查询流量控制当前配置状态

```
mysql> flow_control @@show;
+---------------------+--------------------+------------------+
| FLOW_CONTROL_ENABLE | FLOW_CONTROL_START | FLOW_CONTROL_END |
+---------------------+--------------------+------------------+
| false               |               4096 |              256 |
+---------------------+--------------------+------------------+
1 row in set (0.00 sec)
```

列描述：

- FLOW_CONTROL_ENABLE 是否有开启流量控制功能
- FLOW_CONTROL_START 流量控制功能触发值
- FLOW_CONTROL_END 流量控制功能取消值

2.1.20.2 修改流量控制当前配置状态

flow_control @@set [enableFlowControl = true/false] [flowControlStart = ?] [flowControlEnd = ?]

```
MySQL [(none)]> flow_control @@set enableFlowControl = true flowControlStart= 100 flowControlEND = 30;
Query OK, 0 rows affected (0.02 sec)
```

通过此命令可以修改对应内存中生效的值，不会影响配置文件，对应关系如下：

- enableFlowControl： bootstrap.cnf中enableFlowControl参数
- flowControlStart：bootstrap.cnf中flowControlStartThreshold参数
- flowControlEnd：bootstrap.cnf中flowControlStopThreshold参数

注：以上命令三个参数顺序相关

2.1.20.3 查看当前流量控制中的连接

```
MySQL [(none)]> flow_control @@list;
+------------------+---------------+-----------------------------------+------------------+
| CONNECTION_TYPE  | CONNECTION_ID | CONNECTION_INFO                   | WRITE_QUEUE_SIZE |
+------------------+---------------+-----------------------------------+------------------+
| ServerConnection |             2 | 127.0.0.1:60772/testdb user = root |             2920 |
+------------------+---------------+-----------------------------------+------------------+
```

列描述：

- CONNECTION_TYPE 连接的类型MySQLConnection/ServerConnection
- CONNECTION_ID 连接在dble中的ID信息，可以通过ID查找日志
- CONNECTION_INFO 连接详细信息，使用端口，IP地址，用户，MySQL中的连接ID等
- WRITE_QUEUE_SIZE 当前连接的写队列长度

2.1.21 刷新后端连接池

2.1.21.1 介绍&背景

当在后端mysql做全局变量/权限等的变更时，需要dble的连接池进行一次重新建立，新建的连接才会有新的属性。

2.1.21.2 命令

fresh conn [forced] where dbGroup ='groupName' [and dbInstance='instanceName'];

- 不包含forced时，则空闲的后端连接直接丢弃，而正在使用的后端连接等归还后丢弃
- 包含forced时，则无论空闲还是正在使用的后端连接都直接丢弃；另外正在使用的后端连接对应的前端连接也会全部断开
- 不指定dbInstance时，则dbGroup下所有的dbInstance都将刷新
- 指定dbInstance时，多个dbInstance以逗号隔开，则刷新dbGroup下指定的dbInstance

2.2 全局序列

在分库分表的情况下，数据库自增主键无法保证自增主键的全局唯一。
为此，dble提供了全局序列，并且针对不同应用场景提供了多种实现方式。 要应用全局序列，需要在cluster.cnf中配置：

```
sequenceHandlerType=n
```

其中sequenceHandlerType包括如下种类：

- 1：MySQL offset-step方式
- 2：时间戳方式
- 3：分布式时间戳方式
- 4：分布式offset-step方式

全局序列用于给自增列赋值，写入数据时字段值由系统自动生成，不可以再指定值，使用示例：

```
/*id为主键，配置为自增长*/
insert into table1(name) values('test');
```

2.2.1 MySQL offset-step方式

MySQL offset-step方式利用了数据库的并发和并发更新互斥的特性。
在数据库中建立一张表，保存序列的当前值和步长，使用该序列时，系统从表中读取序列的当前值，加上步长，返回给客户端的同时更新表中的当前值。 线程获取序列的执行步骤如下：

1. 检查是否有请求序列未用的缓存的序列值，如果没有，执行步骤2；否则，执行步骤4。
2. 到相应的数据节点上执行 SELECT dble_seq_nextval(\'**seqName**\'); 其中， **seqName**为请求序列的序列名。此步骤获取的序列值数依赖于该序列的步长**increment**，为区间[current_value, current_value+increment)中的值。
3. 缓存步骤2得到的序列值。
4. 返回缓存的序列值中第一个未用的序列值。

以上提到的部分配置内容参见1.7.1 offset-step方式

2.2.2 时间戳方式

这种方式下，全局序列在dble服务实例本地产生，只能生成全局唯一的ID，不能实现连续自增。
使用这种方式需要对应字段为bigint来保证63位(63位的原因是Java没有无符号整数类型，所以最高位恒为0，保证全局序列是个正数)
序列值是63bits的整数。整数的位模式如下：

a.29bits	b.10bits	c.12bits	d.12bits

其中，

- a - e为从高位到低位。
- a为系统当前时间戳的低41位中的高29位。
- b为10位instance id，使用bootstrap.cnf的instanceId
- c为12位自增长值
- d为系统当前时间戳的低41位中的低12位。

注意事项：

1. bootstrap.cnf的instanceId的最大值均为1023。
2. 每毫秒时间内允许的最多序列值为4095。为了保证序列值的唯一性，在毫秒时间内请求超过4095个序列值时系统会进行等待到下一毫秒开始。
3. 因为java没有无符号整数,实际使用41位时间戳相对于1288834974657L(2010年左右)的偏移量。
4. 相对于偏移量的处理过够后的41位时间戳可供使用69年。

2.2.3 分布式时间戳方式

本方式提供一个基于Zookeeper(也可以本地配置）的分布式ID生成器，可以生成全局唯一的63位二进制ID。
PS:63位的原因是Java没有无符号整数类型，所以最高位恒为0，保证全局序列是个正数

2.2.3.1 位模式

序列值是63bits的整数。整数的位模式如下:

a.9bits	b.9bits	c.6bits	d.39bits

其中:

- a - e为从高位到低位。
- a为线程id的低9位值。
- b为9位实例 id值(根据配置， 此值为bootstrap.cnf的instanceId值或者从zookeeper服务器获取的值, 参见1.7.3 分布式time序列)。
- c为6位自增长值
- d为系统当前时间戳的低39位值(可以使用17年)。

2.2.3.2 退化的分布式时间序列

如果cluster.cnf中的sequenceInstanceByZk值不为true, 序列的维护仅依赖于单实例（bootstrap.cnf的instanceId值的维护），此时序列类似于时间戳方式(参见2.2.2 时间戳方式)。

2.2.3.3 分布式时间序列

如果cluster.cnf中的sequenceInstanceByZk值为true，序列的维护用zookeeper的临时自增节点来维持。每次生成全局序列时,向zk申请一个临时自增节点,通过计算自增节点数 % 32 获取INSTANCEID.

2.2.4 分布式offset-step方式

分布式offset-step方式利用zookeeper的分布式锁功能进行实现。

线程在获取序列值时执行如下步骤:

1. 根据序列名(**`schemaX`.`tableX`**)获取相应的分布式锁。
2. 从zookeeper服务器上获取未用序列值的最小值min。
3. 根据配置(参见1.7.4 分布式offset-step序列),计算能获取的最大未用序列值max。
4. 用max+1更新zookeeper服务器上的未用序列值的最小值min。
5. 释放相应的分布式锁。

2.3 读写分离

3.20.10.0版本dble支持单纯的读写分离，可以和分库分表功能分开单独使用。3.20.10.0之前的版本，分库分表也支持读写分离，兼容该功能。

2.3.1 读写分离配置

2.3.1.1 单纯使用读写分离功能的配置

若想启用dble的读写分离，仅需在 user.xml 文件中配置 rwSplitUser并指定对应的dbGroup即可。dbGroup的配置参考db.xml的章节。这里需要注意的是三种用户配置的顺序是固定的。user.xml的配置请参考user.xml章节。

```
<dble:user xmlns:dble="http://dble.cloud/" version="4.0">
    <managerUser name="man1" password="654321" maxCon="100"/>
    <shardingUser name="root" password="123456" schemas="testdb" readOnly="false" maxCon="20"/>
    <rwSplitUser name="rwsu1" password="123456" dbGroup="rwGroup" maxCon="20"/>
</dble:user>
```

配置注意事项：

1. 当在user.xml中仅配置rwSplitUser时，dble不会加载sharding.xml配置文件，即dble不具备分库分表的功能。
2. 当同时开启dble读写分离和分库分表的功能，分库分表引用的dbGroup和读写分离引用的dbGroup必须相互独立。rwSplitUser引用的dbGroup，仅需在db.xml中定义即可。shardingUser引用的dbGroup，需要被配置的schemas对应的sharding.xml中的shardingNode所引用。
3. 多个rwSplitUser可以引用同一个dbGroup。
4. 被读写分离或者分库分表使用的dbGroup内的instance才会有心跳和连接池；未被有效使用的dbGroup内的instance只有心跳，不会初始化连接池。

2.3.1.2 分库分表中读写分离的配置

分库分表中的读写分离，配置好db.xml和sharding.xml即可，具体参考db.xml和sharding.xml的章节

2.3.2 负载均衡

dble通过配置多个dbInstance为读操作提供负载均衡，注意的是rwSplitMode配置不为0，详细请参见db.xml章节。负载均衡规则如下：

1. 确定参与读写分离的dbInstance集合
2. 负载均衡算法

2.3.2.1 确定参与读写分离的**dbInstance**集合

该算法在每次连接获取时提供可用的dbInstances实例集。可用的dbInstance是指心跳正常的dbInstance，这里需要注意的是show slave status和其他心跳语句是有区别的，以该语句作为心跳语句，心跳正常只是基本前提。dble会根据最近一次的心跳返回结果判断读库和主库的延迟，如果延迟超过delayThreshold配置，则不会将此节点加入到dbInstances实例集中，如果delayThreshold=-1那么不会进行延迟检测。

- 写节点(primary="true")可用
  - rwSplitMod配置为2，则写节点有资格参到读写分离，将写节点加入到dbInstances实例集
  - 读节点(primary没配置或者 primary="false")
    - 节点可用且需要进行延迟检测，检查延迟是否在阈值内再决定是否加入到dbInstances实例集
    - 节点可用且不需要进行延迟检测，直接加入到dbInstances实例集
- 写节点异常
  - 检查读节点是否可用，与上面读节点的检测机制一致

2.3.2.2 负载均衡算法

该算法在dbInstance集合中选择一个dbInstance实例来获取连接。

- dbInstance集合为空，前端报错。
- dbInstance集合非空
  - 每个dbInstance有权重设置(readWeight参数), 但不都是等值权重, 依权重随机选择。
  - 每个dbInstance无权重设置或所有权重等值, 则等权随机选择。此种情况只是上面情况的特例。

2.3.2.3 写节点是否参与均衡与**dbGroup**的**rwSplitMode**属性有关，具体见下图

2.3.3 读写分离支持语句类型

在事务中所有语句都会发主，在非事务中则根据语句类型进行负载均衡。

2.3.3.1 纯读写分离支持的语句类型

1. ddl
2. dml
3. prepared statement协议
4. 函数，存储过程

2.3.3.2 分库分表支持的语句类型

1. 可进行负载均衡的SQL语句为 select 或者 show。

2.3.4 读写分离功能限制

1. druid 解析器限制 - 不支持set语句中存在特殊字符；
2. druid 解析器限制 - set session transaction read write, isolation level repeatable read中，逗号后的语句不生效；
3. 只读事务的支持；
4. 不支持set transaction read write；
5. select 语句现在的逻辑是都进行负载，还没有进行细节的区分，比如有些语句需要强制发主，如系统函数，系统表，系统变量；
6. select ... into 或者 load data中存在用户变量，通过dble再次查询该变量，变量值不对；
7. 临时表，预编译语句和当前连接存在关联，使用会有问题；
8. client连接dble时，若指定不存在的schema默认库，dble不会报错；
9. 在会话中，删除正在使用的库，mysql会将当前库置为null，dble依然保留；
10. prepared statament协议未测试，谨慎使用；
11. set 语句目前只支持会话级别系统变量和用户变量的设置，若需要设置密码等可以使用hint的方式设置或去后端节点去设置；
12. 读写分离会打破原先的隔离级别；

2.4 注解/Hint

Hint, 即注解。 我们定义为:在要执行的SQL语句前添加额外的一段由注解组织的代码，这样SQL就能按照编写者的意图执行，这段代码称之为“注解”。
注解的作用如下：

- 指定路由，比如强制读写分离。
- 帮助dble支持一些不能实现的语句，如单节点内存储过程的创建和调用，如insert…select…；

原理解释：分布式环境下执行SQL语句的流程是先进行SQL解析处理，计算出路由信息后，然后到对应的物理库上去执行；若传入的SQL语句无法解析，则不会去执行。注解则可以告诉解析器按照注解内的SQL（称之为注解SQL）去进行解析处理，解析出路由信息后，将真正要执行的SQL语句（称之为原始SQL）发送到对应的物理库上去执行。

2.4.1 Hint语法

Hint语法有三种形式：

1. /*!dble:type=....*/
2. /*#dble:type=...*/
3. /* */(只适用于读写分离功能)

```
"/*#dble: */" for mybatis and "/*!dble: */"  for mysql
```

具体语法介绍请见：Hint

其中，type有4种值可选：shardingnode，db_type，sql，db_instance_url。每一种值的功能和形式详见各个部分的具体说明。

2.4.2 类型shardingnode

1. 形式
   shardingnode=node
   其中，node为单个数据节点名,不能为多值(node定义参见配置1.2 schemal.xml)。
2. 功能
   为不方便路由或者不能路由的的语句指定具体的目的数据节点。

2.4.3 类型db_type

1. 形式
   db_type=master或者db_type=slave
2. 功能
   帮助实现正确的业务逻辑，强制读写分离。
3. 注意事项
   delete, insert, replace, update, ddl语句不能使用db_type=slave进行注解。

2.4.4 类型sql

1. 形式
   sql=**sql_statement**
2. 功能
   用**sql_statement**的路由结果集作为实际sql语句的执行数据节点。支持存储过程。

2.4.5 类型db_instance_url

1. 形式
   类型db_instance_url=**ip:port**
2. 功能
   在读写分离场景下，可直接下发到相应的mysql实例
3. 注意事项
   运行delete, insert, replace, update, ddl语句时需考虑mysql节点的read_only属性

2.4.6 注意事项

写注解需要注意如下事项：

- dble的注解和MySQL原生注解含义不同, 想通过MySQL原生注解来设置变量或者指定索引是无法得到预期结果的。如#1169
- 使用select语句作为注解SQL，不要使用delete/update/insert 等语句。 delete/update/insert 等语句虽然也能用在注解中，但这些语句在SQL处理中有一些额外的逻辑判断，会降低性能，不建议使用；
- 注解SQL 本身禁用表关联语句，注解目的是路由计算，如果本身写得过于复杂，会影响路由计算；
- 使用hint做DDL需要额外执行reload @@metadata
- 使用hint做session级别的系统变量和环境变量可能不会生效，请慎用
- 使用注解并不额外增加的执行时间；从解析复杂度以及性能考虑，注解SQL应尽量用最简单的SQL 语句，如select id from tab_a where id=’10000’；
- 能不用注解也能够解决的场景，尽量不用注解。
- 在读写分离场景下，语句为/ *xxx* /的注释(纯注释，不包含sql)，uproxy会将该语句发往slave，dble会将该语句发往master。

2.5 分布式事务

- 2.5.1 XA事务概述
- 2.5.2 XA事务的提交以及回滚
- 2.5.3 XA事务的后续补偿以及日志清理
- 2.5.4 XA事务的记录
- 2.5.5 一般分布式事务概述

2.5.1 XA事务概述

2.5.1.1 XA事务概述

普通事务只能在单节点内部保证事务的完整性，如果事务要在不同的节点上执行，无法保证数据一致性，具有跨节点的数据一致性要求的场景需要使用2阶段协议实现分布式事务。 由于以上普通事务的缺陷所以Dble引入了Mysql的XA事务来解决这个问题，MySQL5.7之前版本的XA事务存在一些问题，启用需要MySQL 5.7版本，否则无法保证数据不丢失。dble提供的分布式事务采用两阶段提交协议，目前还是弱XA事务，不能绝对保证跨节点数据的强一致性。

1. 事务开始前需要设置手动提交：set autocommit=0;
2. 使用命令开启XA 事务：set xa=on;
3. 执行相应的SQL 语句。
4. 对事务提交或者回滚，事务结束：commit/rollback;

分布式事务的性能开销比较大，所以只推荐在全局表的事务以及其他一些对一致性要求比较高的场景中使用。一般情况下尽量不要在同一个事务中运行跨节点的SQL语句。 dble支持大事务，但一方面大事务会造成事务执行时间上升，事务信息规模扩大，导致系统性能下降；另一方面，大事务也容易造成潜在的事务数据不一致问题。因此，大事务一定要谨慎使用，一般建议单个事务的 SQL 语句不要超过100条。

以下提供一个JDBC使用XA事务的实例，注意，此例子为最简单的demo，切勿直接在生产环境上使用。
生产环境需要一些更健壮的代码，比如使用连接池(连接重用)时，在放回连接池之前需要将系统变量设置回初始值。

```
public class XaDemo {

    public static final String URL = "jdbc:mysql://localhost:8066/testdb";
                                    //在这里也可以使用jdbc:mysql://127.0.0.1:8066?sessionVariables=xa=1
                                    //进行替代，在这种情况下不需要执行set xa = 1
    public static final String USER = "root";
    public static final String PASSWORD = "123456";

    public static void main(String[] args){
        try {
            //1.加载驱动程序
            Class.forName("com.mysql.jdbc.Driver");
            //2. 获得数据库连接
            Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
            //3.操作数据库，实现增删改查
            Statement stmt = conn.createStatement();
            stmt.execute("set xa = 1");
            //开始一个xa事务
            stmt.execute("begin");
            try {
                //执行相关数据操作的时候需要对于可能出现的错误进行catch
                //并在错误出现的时候对于整个事务进行rollback
                stmt.execute("insert into xa_test set id = 11,name = '3333'");
                stmt.execute("insert into xa_test set id = 22,name = '333'");
                stmt.execute("insert into xa_test set id = 3,name = '33'");
                //数据执行完成提交
                stmt.execute("commit");
            }catch (Exception e){
                System.out.println(" error "+e);
                //如果在数据操作的时候出现错误，将整个事务的操作回滚
                stmt.execute("rollback");
            }finally {
                stmt.close();
                conn.close();
            }

        }catch(Exception e){

        }
    }
}
```

2.5.1.2 XA事务的基础

xa事务基于mysql5.7的xa事务的特性，其流程和特性由以下图所示：

2.5.2 XA事务的提交以及回滚

2.5.2.1 XA事务的提交

在Dble中采用二段提交的方式对于XA事务进行提交 具体的逻辑可见下图

整体来说的处理原则如下：

1. 将XA事务的提交分为END PREPARE COMMIT三个部分
2. 如果在PREPARE下发之前有节点失败或报错，关闭所有后端连接放弃事务数据
3. 如果在PREPARE下发过程中发生失败，则回滚事务，所有节点下发ROLLBACK
4. 如果在COMMIT节点发生失败，则尝试重新下发，几次尝试未果将事务交给定时任务来继续重试

2.5.2.2 XA事务的回滚

相对来说回滚的逻辑就容易的多，直接在所有节点下发rollback。如果失败，直接关闭失败的连接。

2.5.2.3 XA事务重试机制

2.19.03.0之前的dble版本对于失败的事务，策略是将事务交给定时任务后台进行无限重试。2.19.03.0版本对这一过程进行了可配置化，并提供管理命令控制这一过程。

2.5.2.3.1 配置

2.19.03.0版本可以通过bootstrap.cnf文件中的xaRetryCount属性配置xa后台重试策略：

1. 当 xaRetryCount 等于0时，后台无限重试
2. 当 xaRetryCount 大于0时，后台尝试次数达到xaRetryCount后，重试停止

   若重试失败，会发出一条告警，重试成功后自动解决相应告警。

2.5.2.3.2 相关命令

2.19.03.0版本新增两个管理命令

1. 通过 `show @@session.xa` 查看后台重试xa事务信息
2. 通过 `kill @@xa_session id1,id2...` 取消指定session后台尝试xa事务

2.5.3 XA事务的后续补偿以及日志清理

2.5.3.1 XA事务的补偿逻辑

由于XA事务是通过多次提交来达成最终的提交和回滚的，所以就会出现由于服务下线或者其他原因导致的提交或是回滚任务被执行了一半 在这种情况发生的时候dble通过XA事务记录的日志进行日志的处理以及补偿 详细通过在dble重启的时候在重启之前对于XA日志进行读取，并根据里面的内容对于XA事务进行回滚或者补偿 具体的补偿流程如下图所示：

注：XA事务的补偿不保证所有事务都能在补偿期间正确提交，若补偿期间依然因为SQL执行造成失败，只会在dble普通日志中体现或者告警（如果有配置告警功能）

2.5.3.2 XA事务的日志清理逻辑（xaLogClean）

xa事务的日志只保存最近一段时间的，所以需要有定期的清理已经正确提交或者是正确回滚的日志数据 会根据bootstrap.cnf中的配置信息xaLogCleanPeriod定时将没有后续作用的数据做清除

2.5.3.3 XA事务定期连接检查（xaSessionCheck）

在dble提供服务期间如果发生commit(狭义commit,特指XA事务中prepare之后的commit动作)失败或者是rollback失败，那么失败的事务将被存储到内存队列中 并且进行定时的提交或回滚，直到事务被正确的提交或回滚 这是由于在xa事务的逻辑中prepare全部完成之后的事务已经都被成功持久化，仅需要提交即可

2.5.4 XA事务的记录

2.5.4.1 XA 事务过程中记录的内容

由于在Dble中采用两段提交的分布式事务，所以使用XA事务的时候对于DBLE本身就拥有了状态。状态就需要有文件或者其他方式的记录，其中关于XA事务细节的记录主要是记录以下几个部分

```
1. 事务ID
2. 事务状态
3. 事务中每个节点的连接host
4. 事务中每个节点的连接端口
5. 事务中每个节点连接最后的事务状态
6. 事务中每个节点连接的过期状态(没有实际作用)
7. 事务中每个节点连接对应的后端数据库
```

这里举例一个记录的实例

```
{
    "id": "'Dble_Server.1.15'",
    "state": "8",
    "participants": [
        {
            "host": "10.186.24.37",
            "port": "3308",
            "p_state": "8",
            "expires": 0,
            "schema": "db3"
        },
        {
            "host": "10.186.24.37",
            "port": "3306",
            "p_state": "8",
            "expires": 0,
            "schema": "db2"
        },
        {
            "host": "10.186.24.37",
            "port": "3308",
            "p_state": "8",
            "expires": 0,
            "schema": "db2"
        },
        {
            "host": "10.186.24.37",
            "port": "3306",
            "p_state": "8",
            "expires": 0,
            "schema": "db1"
        }
    ]
}
```

2.5.4.2 XA事务中status的标识字典

status	状态	解释
0	`TX_INITIALIZE_STATE`	XA事务处于初始化状态
1	`TX_STARTED_STATE`	XA事务处于开始状态，在事务开始直到提交或者回滚之前 XA事务的状态一直会保持此状态
2	`TX_ENDED_STATE`	XA END下发成功状态
3	`TX_PREPARED_STATE`	XA PREPARED成功状态
4	`TX_PREPARE_UNCONNECT_STATE`	XA PREPARED下发过程中连接被断开
5	`TX_COMMIT_FAILED_STATE`	XA COMMIT 下发失败
6	`TX_ROLLBACK_FAILED_STATE`	XA ROLLBACK 失败
7	TX_CONN_QUIT	后端mysql连接失败
8	`TX_COMMITTED_STATE`	XA 事务提交成功
9	`TX_ROLLBACKED_STATE`	XA 事务回滚成功
10	`TX_COMMITTING_STATE`	XA 事务正在提交
11	`TX_ROLLBACKING_STATE`	XA 事务正在回滚
12	`TX_PREPARING_STATE`	XA 事务正在下发prepare

2.5.4.3 XA事务记录的存储方式

一、本地文件方式

顾名思义在这种方式下，xa事务的状态将以本地记录文件的方式被存放到对应的文件中，具体的路径和文件名配置是

配置与bootstrap.cnf中的{xaRecoveryLogBaseDir}/{XaRecoveryLogBaseName}.log默认条件下文件会被储存在./tmlogs/tmlogs-1.log

一般只在Dble单机状态下使用本地文件方式，使用集群时本地文件的方式将在集群状态下造成不可预知的错误

二、**ZK**存储方式

ZK存储方式不需要额外的配置，当Dble使用ZK配置时，自动默认XA事务记录的存储方式也会是ZK存储

具体的XA事务记录的内容保持不变，记录在dble/{clusterId}/XALOG/{myid} 的Key值中

2.5.5 一般分布式事务概述

在分布式事务中整体的逻辑和mysql的事务逻辑类似，通过长期持有的连接来进行，每个前端连接frontconnection对应一个session，在dble的每个session中有对应的事务状态以及session所持有的后端连接集合target，在非事务状态下或者是autocommit状态下每次后端连接被使用完毕之后就会被移除target并释放回空闲连接池，但是在事务开启的状态下，在SQL执行完毕的时候connection会在target中长期储存，直到session发起commit或者是rollback。

综上我们可以看到，Dble中的普通分布式事务其实就是后端mysql事务的集合，并且这个事务是没有文件记录的，由于mysql的事务特性在事务发生的过程中若断开连接等同于放弃事务，所以可能出现在commit的过程中由于各种意外情况导致事务的部分提交，例如连接后端四个节点dn1,dn2,dn3,dn4在提交commit进行依次下发的时候dn1,dn2,dn3都提交成功，但是dn4由于网络意外提交失败，导致了预期执行的部分内容丢失，并且由于dn1,dn2.dn3已经提交成功无法进行数据回滚，只能进行人工的数据补偿。

其逻辑图如下：

:

2.6 连接池管理

在dble中每个后端MySQL节点由 PhysicalDbInstance 表示，PhysicalDbInstance中维护了两类连接：

1. 大部分业务使用的连接由连接池管理
2. 独立于连接池之外的连接，这类连接主要有两类：MySQL实例的心跳连接和用于OneTimeJob的一次性任务的连接，这种连接是一次性的，用完即关，次数也比较少。

2.6.1 dble后端连接池管理

后端连接池使用 CopyOnWriteArrayList 存储该MySQL实例的全量连接，通过连接的state状态来维护连接的初始化，借出，空闲，心跳，移除状态。连接池初始化之后会维护一个evictor线程来维持连接池的扩缩容以及空闲连接的有效性。结构如下图所示：

2.6.1.1 连接获取

业务向连接池请求获取后端连接，会遍历连接池中的全量连接直到找到第一个空闲连接。若连接池中当时没有空闲连接，则线程会进入超时等待队列，在超时时间内未获取到连接，前端报错。

2.6.1.2 连接释放

后端连接处理完业务之后，状态会被置为空闲，并且处理线程会唤醒在超时等待队列中的线程来重新获取连接。

2.6.1.3 连接池扩缩容

连接池初始化之后会维护一个evictor线程来维持连接池的扩缩容以及空闲连接的有效性，evictor线程是一个定时任务。

扩容：当空闲连接数小于minCon时，维持连接池中的空闲连接在minCon的数量上，每次扩容的连接数量通过以下公式计算：min(配置的最小空闲连接数 - 当前连接池中空闲连接数， 配置的最大连接数 - 连接池中的总连接数) - 正在创建的连接数，若数量大于0，则创建该数量的连接。

缩容：当空闲连接数大于minCon时，维持连接池中的空闲连接在minCon的数量上，每次关闭的连接数量通过以下公式计算：（连接池中的最小连接数 - 配置的最小连接） > 0 && 连接达到idleTimeout。

2.6.1.4 连接有效性检测

在连接的不同阶段，提供对连接有效性的检测手段。

- testOnCreate为true，在连接被创建后，会发送ping命令探测连接有效性，若在connectionHeartbeatTimeout没有收到结果，会关闭连接。
- testOnBorrow为true，在连接被借出后，会发送ping命令探测连接有效性，若在connectionHeartbeatTimeout没有收到结果，会关闭连接。
- testOnReturn为true，在连接被返回后，会发送ping命令探测连接有效性，若在connectionHeartbeatTimeout没有收到结果，会关闭连接。
- testWhileIdle为true，对所有空闲连接，发送ping命令探测连接有效性，若在connectionHeartbeatTimeout没有收到结果，会关闭连接。

2.6.2 连接状态管理

由上文我们知道，后端连接池使用 CopyOnWriteArrayList 存储该MySQL实例的全量连接，通过连接的state状态来维护连接的初始化，借出，空闲，心跳，移除状态。下面是连接状态的跃迁图：

2.6.3 连接池属性

属性名	默认值	单位	含义
testOnCreate	false	无	连接创建后是否检测有效性
testOnBorrow	false	无	连接被借出后是否检测有效性
testOnReturn	false	无	连接被返回时是否检测有效性
testWhileIdle	false	无	连接空闲时是否检测有效性
connectionTimeout	30000 (30s)	毫秒	获取连接的超时时间
connectionHeartbeatTimeout	20	毫秒	空闲连接检测后的超时时间
timeBetweenEvictionRunsMillis	30000 (30s)	毫秒	扩缩容线程的检测周期
idleTimeout	600000 (10 minute)	毫秒	连接空闲多久之后被回收
heartbeatPeriodMillis	10000 (10s)	毫秒	连接池的心跳周期
evictorShutdownTimeoutMillis	10000 (10s)	毫秒	扩缩容线程停止的超时时间

2.6.4 dble后端连接池的心跳管理

dble后端MySQL节点的心跳管理是通过定时任务来完成的，检测周期由heartbeatPeriodMillis来控制。dble会对每一个后端MySQL节点持有一个长连接，定期发送心跳语句，根据返回结果的不同将连接池标记为不同的状态。若心跳异常，会影响evictor线程的扩缩容。

2.6.4.1 心跳周期内的不同阶段

我们可以将每个心跳周期简单划分为两个阶段：

- 检测阶段：心跳检测发起到收到回复的阶段
- 空闲阶段：心跳返回后到下一个心跳检测发起的阶段

2.6.4.2 心跳状态

dble的心跳状态有三种：

- ok状态：一次正常的心跳返回
- timeout状态：最近的一次心跳在HeartbeatTimeout时间段内没有收到响应
- error状态：心跳语句返回错误或者心跳连接异常都可能导致此种状态，dble里面会有重试机制来预防网络抖动等网络方面的异常。

2.6.4.2 心跳重试

此处的心跳重试分为两种情况：第一种是心跳语句返回错误导致的重试，第二种是心跳连接关闭导致的重试。

- 对于心跳语句返回失败，dble会立即将连接池状态置为error状态，随即会发送errorRetryCount次心跳，若有一次心跳正常，心跳恢复成ok状态。
- 对于心跳连接关闭引起的失败，dble会在接下来的时间立即发送errorRetryCount次心跳，若有一次心跳正常，则停止重试，但如果都失败，则将连接池状态置为error状态

2.7 内存管理

2.7.1 内存结构概览:

- On-Heap 大小由JVM 参数Xms ,Xmx 决定，就是正常服务需要的内存，由jvm自动分配和回收。
- Off-Heap大小由JVM 参数 XX:MaxDirectMemorySize 大小确定。
- DirectByteBufferPool 大小 = bufferPoolPageNumber*bufferPoolPageSize

  bufferPoolPageNumber和bufferPoolPageSize可在bootstrap.cnf 配置， bufferPoolPageSize默认为2M, bufferPoolPageNumber默认为Java虚拟机的可用的处理器数量*20
- 映射文件区不在JVM之内

  大小计算方法求得值tmpMin= Min(物理内存的一半， free内存)

  可在映射区保存的文件的个数=(向下取整(tmpMin/mappedFileSize))

  真实使用值= 可在映射区保存的文件的个数*mappedFileSize (mappedFileSize默认64M)

  其中mappedFileSize可由bootstrap.cnf 指定

  注：这里是通过单例模式，一次性确定映射文件区的大小，如果运行过程中，内存被其他进程占用，这里可能存在风险

2.7.2 DirectByteBufferPool

BufferPool结构概览:

单个bufferPoolPage的内部结构:

1.用途**:**

1.1 网络读写时使用

1.2 中间结果集暂存时用于写数据前的缓存**buffer**

2.初始化:

按照参数，初始化为bufferPoolPageNumber个页，每个页大小为bufferPoolPageSize

3.内存分配

3.1 分配大小

如果不指定分配大小，则默认分配一个最小单元(最小单元由bufferPoolChunkSize决定，默认大小为 4k,请最好设为bufferPoolPageSize 的约数，否则最后一个会造成浪费)。

如果指定分配大小，则分配放得下分配大小的最小单元的整数倍(向上取整) 。

总之，大小为 M*bufferPoolChunkSize

3.2 分配方式

遍历 缓冲池从N+1页到bufferPoolPageNumber-1页(上次分配过的记为第N页)

```
对单页加锁在每个页中从头寻找未被使用的连续M个最小单元
```

如果没找到，再从第0页找到第N页

以上成功后更新上次分配页，标记分配的单元

如果找不到可存放的单页(比如大于bufferPoolPageSize)，直接分配On-Heap内存

4.内存回收

4.1 如果是**On-Heap**内存

直接clear，等GC回收

4.2 如果是**Off-Heap**内存

遍历所有页，找到对应页

对单页加锁，到对应页的对应块的位置，标记为未使用

2.7.3 处理中间结果集过大时内存使用

目前dble 对每个session的内存管理如下：

- Join内存管理，最大上限为4M，目前写死，用于管理join操作暂存的数据
- Order内存管理，最大上限为4M，目前写死，用于管理排序操作暂存的数据
- Other内存管理，最大上限为4M，目前写死，用于管理distinct group ，nestloop操作暂存的数据

每一个复杂查询中，当子查询单元需要在中间件当中暂存数据的时候,数据会存在Heap内存当中，但如果当前存储使用的内存大于4M，则需要写内存映射文件。
如果内存映射文件个数达到上限（参见概览中的可在映射区保存的文件的个数），则会去写硬盘。
写文件的时候，当单个文件大于mappedFileSize时，会将文件拆分。

注1:内存映射文件总大小在第一次使用时就确定，有风险
注2:写硬盘时候的缓冲区是从DirectByteBufferPool申请的chunk大小的

2.8 集群同步协调&状态管理

2.8.1 概述

大多数时候，dble结点是无状态的，所以可以用常用的高可用/负载均衡软件来接入各个结点。
这里不讨论各个负载均衡软件的使用。
主要讨论一下某些情况下需要同步状态的操作和细节。
注：本部分内容需要额外部署zookeeper用于管理集群的状态和同步。

2.8.2 相关配置

2.8.2.1 cluster.conf

以ZK为例

```
# 开启集群模式
clusterEnable=true
# 集群模式元数据中心为zk
clusterMode=zk
# zk地址
clusterIP=10.186.19.aa:2281,10.186.60.bb:2281
#zk为dble提供的根目录
rootPath=/dble
#本组dble的组名
clusterId=cluster-1
# 是否需要同步Ha状态
#needSyncHa=false
# 拉取一致性binlog线的超时时间
#showBinlogStatusTimeout=60000
#自增序列类型
sequenceHandlerType=2
#自增序列默认开始时间
#sequenceStartTime=2010-11-04 09:42:54
#自增序列类型为3时，instanceId是否由ZK生成
#sequenceInstanceByZk=true
```

2.8.2.2 bootstrap.conf

instanceName 实例名称，用于发起集群任务或者汇报自己完成集群任务的标记 instanceId （自增序列使用，根据类型范围为0到1023 或者0到511）

2.8.3 初始化状态

方式1.

通过执行脚本init_zk_data.sh方式将某个结点的配置文件等数据写入ZK,所有结点启动时都从ZK拉取配置数据。

方式2.

在第一个结点第一次启动时自动将自己的配置文件写入ZK,其他结点启动时从ZK拉取。
第一个结点的判定：用分布式锁抢占的方式，未抢占到结点会阻塞等待直到获取到分布式锁，如果此时初始化标记已经被设置，则从ZK拉取配置，否则将自己本地配置写入。

2.8.4 状态同步

A.DDL

做DDL时候会在执行的某个节点成功后，将消息推给ZK，ZK负责通知其他结点做变更。流程如下图：

注:

1. 新结点启动时，加载元数据前会检测是否有其他结点在做DDL变更，如果有，则等待。
   否则加分布式锁防止加载元数据期间其他结点做DDL变更, 直到元数据加载完释放分布式锁 。
1. 回收指的是：每个节点ddl完成后会告知分布式协调器自己已经完成，并检查如果所有结点均完成，则删除schema.table 结点。此操作为原子操作。
1. 发起结点故障：如果执行DDL的结点故障下线，其他结点会侦听到此消息，保证解开对应结点的tablemeta锁，并记录故障告警（如果配置了告警通道），需要运维人工介入修改ZK对应ddl结点的状态，检查各个结点meta数据状态，可能需要reload metadata。
1. 逻辑上不应该有某个监听节点上加载meta失败的情况，如果发生了，告警处理

   (人工介入对应结点的meta是不正确的,需要reload meta)
1. 注:view目前是异步模式,可能存在某个间隙view修改成功，查询仍旧拿到旧版的view结构。

B.reload @@config/ reload @@config_all/ rollback @@config

执行流程如下图:

注：如果在部分结点失败，则会返回错误及错误原因以及结点名。 设计影响面：db.xml，sharding.xml, userxml ,sequence_conf.properties 和sequence_db_conf.properties

C.拉取一致性的binlog线

目的:

获得后端数据库实例的一致性binlog位置。由于两阶段提交的第二阶段执行在各结点无法保证时序性和同步性，所以直接下发show master status获取binlog可能会造成不一致。

实现方式:

如下图，当前端收到show @@binlog.status 语句时，遍历当前所有活动session查看状态 。
若session处于绿色区域，则在进入红色区域前等待知道show @@binlog.status结果返回
若存在session处于红色区域，则需要等待所有红色区域的session返回结果走出红色区域后下发show master status。

超时处理:

此处有可能有死锁发生。
场景:session1 正在更新tableA，处于绿色区域，session2下发有关于tableA 的DDL，等待 metaLock解锁，处于红色区域.session3 下发show @@binlog.status.
此时session1 等待session3, session2等待session1,session3等待session2.
因此引入超时机制。如果session3 等待超过showBinlogStatusTimeout(默认60s，可配置)，自动放弃等待，环状锁解除。

集群协调:

1.收到请求后同步通知ZK，先等待本身结点准备工作结束，之后zk通知其他结点处理。
2.所有结点遍历各自的活动的session，进入红色区域的等待处理完成，绿色区域的暂停进入红色区域。
3.结点将准备好/超时将状态上报给ZK
4.主节点等待所有结点状态上报完成之后，判断是否可以执行任务，若是,则执行show @@binlog.status并返回结果，否则报告本次执行失败。
5.主节点通过ZK通知各结点继续之前的任务

集群超时处理:

若有结点超时未准备好，主节点会报超时错误，并通过ZK通知各结点继续之前的任务。

故障处理:

主节点执行过程中故障下线，其他结点会感知，保证自己结点一定时间后自动解锁继续原有任务。
ZK状态需要人工干预。人工等待所有结点超时之后，手动删除/修改zk上的状态信息以便下次执行时不出问题。

D.View管理

在使用集群模式是，使用zk进行view视图信息的管理，使得整个dble集群能够进行视图信息的同步 由于视图只用于数据查询且不会造成数据异常的属性，在视图同步时采用异步同步的方法

注 在zk上的view数据信息如下形式 key schema.table value {"serverId":"create_Server_id", "createSql":"view_create_sql"}

E.online状态

dble启动时候会在online的目录下注册自己的信息，如果此时正在做DDL，会阻塞一段时间，防止表结构元数据不一致。
功能:
1.作为集群协调时候检查哪些dble完成了对应任务的标准 2.dble故障后同集群的其他实例会发现状态并处理完收尾的状态，主要是DDL状态残留，拉取一致性binlog线或者暂停流量的残余。

F.高可用命令同步

当启用needSyncHa时候，此选项才生效

F.1 disable工作流程:

1.申请分布式锁
2.本地执行disable
3.将信息写入集群
4.等待所有结点完成disable
5.完成并清理
6.释放分布式锁

F.2 enable工作流程:

1.申请分布式锁
2.本地执行enable
3.将信息写入集群
4.释放分布式锁
5.其他订阅结点异步完成enable

F.3 switch工作流程:

1.申请分布式锁
2.本地执行switch
3.将信息写入集群
4.释放分布式锁
5.其他订阅结点异步完成switch

G.暂停流量（一般用于扩容迁移等）

G.1 pause流程

1.申请分布式锁 2.通知其他结点 3.本结点停流量 4.等待其他结点完成停流量 或者超时 5.返回暂停成功或者失败

G.2 resume流程

1.本结点恢复流量 2.通知其他结点 3.等待其他结点完成恢复流量 4.释放分布式锁

2.8.5 XA日志管理

在使用集群模式时，未完成的XA日志会存放在zookeeper上，更加安全，防止某台机器硬盘物理损坏导致日志丢失（此处可能会有并发高吞吐引发的性能及其他问题，待测试）。

2.8.6 ZK整体目录结构

```
rootPath(配置在cluster.cnf中的)
    clusterId(配置在cluster.cnf中的)
        conf
            inited
        status
            operator
                instanceName(临时的，key为bootstrap.cnf配置内容。功能:reload响应id)
        sharding(sharding.xml的json信息)
        db(db.xml的json信息)
        user(user.xml的json信息)
        migration(用于暂停流量)
            pause
                instanceName(临时的，key为bootstrap.cnf配置内容。功能:响应结点)
            resume
                instanceName(临时的，key为bootstrap.cnf配置内容。功能:响应结点)
        sequences
            instanceid //zk 分布式方式
            incr_sequence//批量步长方式
                table_name
            common(sequence_conf.properties 或者sequence_db_conf.properties)
    binlog_pause
        status (发生时建立，结束后回收)
            instanceName(临时的，key为bootstrap.cnf配置内容。功能:响应结点)
    lock
        syncMeta.lock(临时的,启动时防止元数据变更)
        confInit.lock
        confChange.lock
        binlogStatus.lock
        ddl_lock/schema.table
        view_lock/`schema`.`table`
        dbGroup_locks/groupName
        pause_node.lock
    online
        instanceName(临时的，key为bootstrap.cnf配置内容。)
    ddl
        schema.table（ddl运行时）
            instanceName(临时的，key为bootstrap.cnf配置内容。功能:响应结点)
        schema.table2
    xalog
        node1
        node2
    view
        schema:view
        operator （订阅目录）
            schema.view:(update/delete,注意:create当作update处理)
                instanceName(临时的，key为bootstrap.cnf配置内容。功能:响应结点)

    dbGroups
        dbGroup_status
            groupName
        dbGroup_response
            instanceName(临时的，key为bootstrap.cnf配置内容。功能:响应结点)
```

2.8.7 全局序列

类twitter snowflake 方式，ZK完成的工作是生成每个节点的instanceID。
类offset-step 方式，ZK完成的工作是存储当前的Step值。

2.8.8 附录

2.9 Grpc告警功能

2.9.1 告警功能概述

Dble拥有和商业项目ucore进行告警对接的功能，当dble触发某些重要的报错信息时，会通过ucore提供的grpc接口将对应的告警信息发送到ucore告警中，免去了运维人员在日志文件中的大量搜索，能够直观展示在页面上。

2.9.2 告警配置依赖

依赖**myid.properties** 的告警的基础信息

名称	内容	默认值	详细作用原理或应用	实例/全局属性
url	grpc告警的url	myid.properties 里的ipAddress	在发送grpc的时候作为IP地址使用	实例
port	告警端口	myid.properties 里的port	grpc发送的目的端口	实例
serverId	服务器ID	$ushard-id(ip1,ip2) ,其中$ushard-id 是myid.properties 里的myid	接口参数	实例
componentId	组件ID	$ushard-id 即myid.properties 里myid	接口参数	实例
componentType	组件类型	ushard	接口参数	实例

2.10 meta数据管理

Meta数据的管理包含以下部分：

- 2.10.1 Meta信息初始化
- 2.10.2 Meta信息维护
- 2.10.3 一致性检测
- 2.10.4 View Meta

2.10.1 Meta信息初始化

dble在启动时会逐个同步抓取相关表，视图的信息，并对分区表做一致性检测。

2.10.1.1 表信息的初始化

表信息是从后端数据节点抓取的。dble在启动时会根据schema，table，数据节点的配置（参见1.2 schemal.xml）对逐个数据节点执行：

show tables;

show create table ...

获取后端表的实际创建信息并做解析以获取需要的信息。

2.10.1.2 视图信息的初始化

视图的信息是属于Dble中间件的状态信息，是为数不多的Dble自有的状态之一，存在两种形式的初始化

1 非集群模式状态下通过本地文件进行view的存储和初始化

2 集群状态下通过ZK进行view的存储和初始化

关于其他view meta的实现以及使用细节详见2.10.4 view meta

2.10.1.3 分区表的一致性规则

分区表分布在多个实例上的后端表的表结构不必完全相同，但最好完全形同。多个后端表被认为一致的规则为:

- 有相同的列;
- 有相同的主键;
- 有相同的唯一键;
- 有同样的索引;
- 有相同的键。

只要这五个方面相同就认为后端的表具有一致性，一致性检测通过。

2.10.2 Meta信息维护

dble在每次执行某些类型ddl语句之后都会更新相关表，视图的元信息。目前，更新元信息的ddl语句类型如下：

- create table语句
- drop table语句
- alter table语句
- truncate table语句
- create index语句
- drop index语句

根据是否配置zookeeper服务器服务(参见1.8 myid.properties)，Meta信息的维护逻辑分如下两种情况:

1. 不用zookeeper服务
   在此种情况下，由于仅有一个dble运行实例，本地更新已是全部信息，不必做进一步的维护逻辑。
2. 利用zookeeper服务
   此种情况下的更新逻辑为:
   a. 在启动时每一个dble实例都向zookeeper服务器注册监听事件，监听系统中表元信息的改变。
   b. 当某一个dble实例更新了某些表的元信息之后，它负责向zookeepr服务器广播更新事件。
   c. 其他dble实例在监听到更新事件后更新自己维护的元信息。

2.10.3 一致性检测

由于Dble对于分片的应用和处理可以看到，对于某张具体的分片表，dble默认在具体数据库节点上的所有表格的实体拥有同样的结构，由于可能在Dble运行的过程中的种种问题导致了数据库节点的表格结构不一致的情况，在Dble中有对于各个分片表的表格结构进行定期检查确认的机制。

分片表结构一致性检查的定时任务为tableStructureCheckTask，其周期为checkTableConsistencyPeriod默认值30×60×1000（30分钟），任务开关为checkTableConsistency默认为0（关闭），以上配置均配置在bootstrap.cnf中

大致的表格一致性检查任务逻辑如下：

- 循环配置信息中的schema以及table
- 检查对应数据库和表格的meta信息是否存在
- 对于每个有效的节点下发SQL “show create table”
- 根据收到的建表语句创建新的meta，并通过Set进行去重
- 判断有几个不同的meta，若meta数量超过一个则进行在日志中打印告警信息
- 将新老的meta进行对比，如果发现新老meta之间的meta也不一致则在日志中打印告警信息

整体的流程图如下：

全局表的一致性检查

全局表在Dble的使用中被认为每个节点都存在全量的数据，所以在全局表的检查中不光检查了全局表的表格结构，还在需要另外检查表格的数据一致性,功能开启由全局表具体配置来决定。

2.10.4 view meta

2.10.4.1 view 种类

- mysql中的view
- dble中的view

2.10.4.2 view meta概述

在Dble 2.18.11.0 版本中新增了对view的支持，作为一个中间件支持view采用的方法是和复杂查询类似的逻辑，将VIEW创建的语句中的select部分进行解析，并将解析所得到的解析树进行存储，当有query调用到视图的时候使用解析树进行局部的替代还原成完整的查询SQL，从而达到view实现的效果。

在MySQL中的View meta信息是一种持久化的存储。Dble中的view实现方式也类似，这样Dble中view的实现会给Dble本身带来状态，这个状态需要Dble自身进行保存和维护，所以在view中采取了和XA事务日志相似的方式，并且以视图创建SQL的形式进行以下两种方式的存储:

- 本地文件存储
- ZK存储

在每次发生重启的时候Dble会在初始化meta的时候，通过读取文件或者是ZK中的信息将创建视图的SQL进行重新的解析，最终以解析树的形式保存到meta中去。

在Dble 2.19.10.0版本中新增了对mysql view创建的支持，mysql view会被dble直接下发到后端mysql中执行。 不过，创建mysql view有着各种限制，首先view 所属的schema必须采取以下配置，即垂直分片的方式:

```
<schema name="schema2" sqlMaxLimit="100" shardingNode="dn5">
</schema>
```

view中涉及的表必须是shardingNode dn5 中的表。对于mysql view，dble则没有持久化，但是依然会有相对应的view meta。mysql view 支持通过dble修改，也支持在后端节点手动修改并通过reload方式加载到dble。

2.10.4.2 view meta保存

本地文件存储

本地文件的存储中，对应的信息以JSON的格式存放在本地文件中，文件的存储路径以及文件名称通过bootstrap.cnf中的viewPersistenceConfBaseDir和viewPersistenceConfBaseName两个参数进行配置（默认存储于./viewConf/viewJson中）,具体的文件内容举例如下

```
[{
    "schema": "testdb",
    "list": [{
        "name": "view_test",
        "sql": "create view view_test as select * from a_test"
    }, {
        "name": "vt2",
        "sql": "create or replace view vt2 as select * from suntest"
    }, {
        "name": "suntest",
        "sql": "create view suntest as select * from sbtest"
    }]
}]
```

ZK K/V存储

在集群状态下Dble的各个内部状态需要同步，包括view的创建删除和修改，具体的key值会存储在ZK BASE_PATH/view下，并使用key值schema_name:view_name，在view中使用json的格式进行create sql和修改的serverID的存储，以下给出一个举例:

```
{
    "serverId":"10010",
    "createSql":"create view view_test as select * from a_test"
}
```

此JSON字符串当作value存储在 /..../view/testdb:view_test

2.11 统计管理

统计管理包含以下几个部分

- 2.11.1 查询条件统计
- 2.11.2 表状态统计
- 2.11.3 用户状态统计
- 2.11.4 命令统计
- 2.11.5 heartbeat统计
- 2.11.6 网络读写统计

2.11.1 查询条件统计

查询条件统计是在处理查询结果时进行的。统计的是关于某个表中关于某列的条件。
dble系统每次只能进行关于一个表中的一个列的条件统计。
要进行此类统计首先必须开启useSqlStat，其次要设置要统计的表和列，详见下节。

2.11.1.1 查询条件统计表列设置

查询条件统计表列的设置命令：
reload @@query_cf=**table&column**;

其中**table**为要统计的目标表的表名，**column**为目标表中目标列的列名。
如果要清除查询条件统计表列的设置执行命令：

```
reload @@query_cf;
```

2.11.1.2 查看查询条件统计结果

要查看查询条件统计结果执行如下命令：

```
show @@sql.condition;
```

2.11.2 表状态统计

表状态统计是在处理查询结果时进行的。要进行此类统计必须开启useSqlStat。

2.11.2.1 统计内容

表状态主要统计如下内容:

- 读操作的次数
- 写操作的次数
- 表关系，即在同一个语句中出现多个表，以第一个表作为操作的主表，其他表是和主表有关系的表。
- 对主表最后一次操作的时刻。

2.11.2.2 统计结果查看

表状态的统计结果可以通过如下命令查看:

```
show @@sql.sum.table;
```

如果要重置表状态统计，执行如下命令:

```
show @@sql.sum.table true;
```

2.11.3 用户状态统计

用户状态统计是在处理查询结果时进行的。要进行此类统计必须开启useSqlStat。

2.11.3.1 统计内容

用户状态统计如下内容:

1. 每个用户的最大并发
2. 每个用户的前N条最慢SQL的sql语句，语句执行的开始时间。N通过sqlRecordCount进行设置，默认值为10。慢SQL的标准为执行时间大于等于T的sql语句。
   通过命令:
   reload @@sqlslow=t;
   进行设置。t为整数，单位为毫秒。
3. 网络读写字节数
4. 每用户最后执行的50条语句
5. 每用户结果集大于10000行的前10条select语句
6. 每用户频度最大的前1024条sql
7. 每用户结果集大小（单位：字节）大于M的sql语句。M通过maxResultSet进行设置。

2.11.3.2 用户状态统计服务的命令

- show @@sql;
- show @@sql.high;
- show @@sql.large;
- show @@sql.resultset;
- show @@sql.slow;
- show @@sql.sum.user;

以上命令的使用请参看2.1 管理端命令集。

2.11.3.3 用户状态统计重置

要清空用户状态统计结果而重新进行统计，执行如下命令:

```
reload @@user_stat;
```

2.11.4 命令统计

命令统计是分类统计dble执行的命令计数，在执行各个命令类的命令时进行统计。统计的命令类如下：

1. initDB
2. query
3. stmtPrepare
4. stmtSendLongData
5. stmtReset
6. stmtExecute
7. stmtClose
8. ping
9. kill
10. quit
11. heartbeat
12. other，除以上命令类命令之外的所有命令。

2.11.4.1 命令统计结果查看

命令统计结果查看执行如下命令：

- show @@command;
- show @@command.count; 具体命令的使用请参看2.1 管理端命令集

2.11.5 heartbeat统计

heartbeat统计后端mysql实例的heartbeat状态信息，在对后端mysql实例进行heartbeat检测时进行统计。

2.11.5.1 统计内容

heartbeat统计每一次从heartbeat查询发送到查询结果接收之间的时间差，同步状态。

2.11.5.2 统计结果查看

heartbeat统计的查看执行如下命令：

- show @@heartbeat;
- show @@heartbeat.detail where name=xxx; 其中，xxx为dbinstance名字。
- show @@dbinstance.synstatus;
- show @@dbinstance.syndetail where name=xxx;其中，xxx为dbinstance名字。

以上命令的使用请参看参看2.1 管理端命令集

2.11.6 网络读写统计

dble的每一个前后端在进行网络通信时对读写的数据量进行统计。

2.11.6.1 统计内容

- 网络读字节数
- 网络写字节数
- 最后一次进行读/写的时刻

2.11.6.2 服务的命令

- show @@connection;
- show @@backend;
- show @@connection.sql;

以上命令的使用请参看2.1 管理端命令集

2.12 故障切换

版本2.20.04.0起对切换功能进行了重定义,废除原有的切换功能，请知悉。
新的MySQL高可用切换分为两类:
一个是单实例部署的dble会内置一个自带的高可用切换的python3脚本，跟随dble启动和停止,需要设置bootstrap.cnf中system的useOuterHa 参数为false。
另一个是支持第三方的高可用切换接口功能，支持集群部署。当然，单实例dble也可以采用这种方式，需要设置system的useOuterHa 参数为true.
这种方式情况下，如果没有真的配置第三方高可用切换组件，则什么也不会发生。 具体请参考高可用切换接口

2.12.1 前提条件

- 单实例部署的dble，设置bootstrap.cnf中system的useOuterHa 参数为false。
- 安装好相应的python3环境，详情参考本章"自定义python脚本"部分

2.12.2 工作方式

dble启动时跟随dble启动python3脚本进程，脚本会读取db.xml内部的配置，检查后端数据库结点的状态，发生异常后会执行切换命令来通知dble

dble提供以下运维命令：

```
show @@custom_mysql_ha
```

用于查看或者监控当前进程是否存活

当python脚本意外终止时， 可以手工运行命令来启动：

```
enable @@custom_mysql_ha
```

当然，也可以手动关闭该脚本

```
disable @@custom_mysql_ha
```

2.12.2 注意事项

2.12.2.1 此功能目前只支持在linux环境下使用。

2.12.2.2 reload注意事项

做reload @@config 之前需要修改配置文件，这时候如果不停止python脚本，可能会读到中间状态的文件，所以我们建议的标准流程如下：
1. `disable @@custom_mysql_ha` 关闭切换功能
2.修改配置文件
3. `reload @@config` 重新加载配置
4. `enable @@custom_mysql_ha` 开启切换功能

2.12.2.3 自定义python脚本

custom_mysql_ha.py 脚本在dble安装目录的bin目录下,使用python3编写,您可以根据实际情况自行修改。
为了使它能正常工作，需要做以下的准备工作：

1、安装**Python3,**确认**python3**是否已安装可通过如下命令

```
/usr/local/bin/python3 --version
/usr/local/bin/pip3 --version
```

2、安装**mysqlclient**及依赖
CentOS 依赖

```
yum install mysql-devel
```

or Ubuntu 依赖

```
apt-get install libmysqlclient-dev
```

然后

```
pip3 install mysqlclient
```

3、**six**

```
pip3 install six
```

or

```
pip3 install six -i http://pypi.douban.com/simple --trusted-host pypi.douban.com
```

4、**coloredlogs**

```
pip3 install coloredlogs
```

or

```
pip3 install coloredlogs -i http://pypi.douban.com/simple --trusted-host pypi.douban.com
```

5、rsa

```
pip3 install rsa
```

or

```
pip3 install rsa -i http://pypi.douban.com/simple --trusted-host pypi.douban.com
```

2.13 前后端连接检查

2.13.1 前端连接空闲检查

- 根据在bootstrap.cnf中配置的processorCheckPeriod进行定时的前后端连接检查
- 根据配置在bootstrap.cnf中的idleTimeout判断前端连接是否存在空闲时间超限的现象
- 如果发现空闲时间超过限度则关闭连接

2.13.2 后端连接SQL超时检查

- 根据在bootstrap.cnf中配置的processorCheckPeriod进行定时的前后端连接检查
- 根据配置sqlExecuteTimeout检查所有正在执行的后端连接，是否有执行时间超限的情况
- 关闭所有执行时间超过限度的非DDL后端连接

2.14 ER 拆分

2.14.1 普通ER拆分

当我们需要两个表要进行join的时候,由于数据被分配到不同的节点上，普通的nest loop 方式可能效率会很低。
如果我们能把需要join的表按照统一规则划分到相同的区上，就能大概率的解决这一问题。
举个简单的例子如下：

这样两张表具有外键关系，我们可以仿照聚簇的方式按照对应列来进行拆分，这样就可以不跨库实现join了。
如下图:

要实现这样的ER功能，需要如下配置。

```
<shardingTable name="sales" shardingNode="dn1,dn2" function="sharding" shardingColumn="id">
<childTable name="sales_detail" joinKey="sales_detail_pos_num" parentKey="sales_pos_num"/>
</table>
```

2.14.2 智能的ER关系

当我们有多个不同的表时，上面的配置方式有点难以使用了。
这种情况下，如果2张或者多张表在mycat上的分片规则相同并且具体分片也相同，即使没有配置ER关系，也会当作ER关系来处理。
举例如下配置（片段）：

```
<!--schema片段-->
<shardingTable name="tableA" shardingNode="dn1,dn2" function="hash_function" shardingColumn="id_a" />
<shardingTable name="tableB" shardingNode="dn1,dn2" function="hash_function" shardingColumn="id_b" />
<shardingTable name="tableC" shardingNode="dn2,dn1" function="hash_function" shardingColumn="id_c" />
<shardingTable name="tableD" shardingNode="dn3,dn4" function="hash_function" shardingColumn="id_a" />
<shardingTable name="tableE" shardingNode="dn1,dn2" function="hash_function" shardingColumn="id_a" />
<shardingTable name="tableF" shardingNode="dn1,dn2" function="enum_par" shardingColumn="id_a" />

<!--rfunction片段-->
<function name="enum_par"
   class="io.mycat.route.function.PartitionByFileMap">
   <property name="mapFile">partition-hash-int.txt</property>
</function>
<function name="hash_function" class="io.mycat.route.function.PartitionByLong">
   <property name="partitionCount">2</property>
   <property name="partitionLength">512</property>
</function>
```

最终会得出这样一个映射关系，识别分组会根据数据分布结点 和function的唯一性将表分为几组，同一组的才会有ER关系。

table名	拆分列	数据分布结点	**function**	识别分组
tableA	id_a	dn1,dn2	hash_function	1
tableB	id_b	dn1,dn2	hash_function	1
tableC	id_c	dn2,dn1	hash_function	2
tableD	id_a	dn3,dn4	hash_function	3
tableE	id_a	dn1,dn2	hash_function	1
tableF	id_a	dn1,dn2	enum_par	4

即，ER关系集合为：

PS：此处略去了schema，实际实现需要标识schema防止重复，ER关系是到列的，如果关联关系不是上述表对应的列，也不会视为ER.

2.15 global 表

在一些业务系统中，存在着类似字典表的表格，它们与业务表之间可能有关系，这种关系，可以理解为“标签”，而不应理解为通常的“主从关系”。
这些表具有以下几个特性:

- 变动不频繁
- 数据量总体变化不大
- 数据规模不大，很少有超过数十万条记录。

鉴于此，dble 定义了一种特殊的表，称之为“全局表”，全局表具有以下特性：

- 全局表的插入、更新操作会实时在所有节点上执行，保持各个分片的数据一致性
- 全局表的查询操作，只从一个节点获取
- 全局表可以跟任何一个表进行JOIN 操作

将字典表或者符合字典表特性的一些表定义为全局表，某种程度上部分解决了数据JOIN的难题。
举例如下:

对于数据量不大的字典表（例:超市商品）,在多个分片上都有一份同样的副本

相关JOIN语句可以直接下发给各个结点，直接合并结果集就行.

JOIN 例子(伪SQL):

```
SELECT 日期,商品名,COUNT(*)  AS 订单量
FROM 商品表
JOIN 销售详单 USING(商品ID)
WHERE 日期范围(跨结点)
GROUP  BY 日期,商品名。
```

2.16 cache 的使用

cache 的配置请参见1.6节的内容。

2.16.1 路由缓存

路由缓存里键值对是[sql, 路由节点]

配置的值里KEY是pool.SQLRouteCache
VALUE 是逗号隔开的三个值，分别为cachefactory的name,容量，超时时间。

2.16.2 ER子表计算缓存

路由缓存里键值对是[子表对应的joinkey, 路由节点]

KEY是pool.ER_SQL2PARENTID

VALUE是逗号隔开的三个值，分别为cachefactory的name,容量，超时时间

2.17 执行计划

2.17.1 执行计划的意义

对执行计划进行分析，可以了解中间件和节点是否对SQL语句生成了最优的执行计划，是否有优化的空间，从而为SQL优化提供重要的参考信息。

2.17.2 执行计划的分类

dble的执行计划分为两个层次：dble层的执行计划与节点层的执行计划。
dble层的执行计划: 在SQL语句执行前，dble会根据SQL语句的基本信息，判断该SQL语句应该在哪些节点上执行，将SQL改写成在节点上执行的具体形式，并决定采用何种策略进行数据合并与计算等。
节点层的执行计划: 就是原生的MySQL执行计划。

2.17.3 dble层的执行计划

dble用EXPLAIN指令来查看dble层的执行计划。如： explain select id,accountno from account where userid=2;
EXPLAIN指令的执行结果包括语句下发的节点，实际下发的SQL语句和数据的合并操作的信息。这些信息是系统静态分析产生的，并没有真正的执行语句。

另外，复杂查询的查询计划也会有所反映，可以通过计划来优化查询语句
如

```
mysql> explain select * from sharding_two_node a inner join sharding_four_node b on a.id =b.id;

+---------------+----------+-------------------------------------------------------------------------------------------+
| SHARDING_NODE | TYPE     | SQL/REF                                                                                   |
+---------------+----------+-------------------------------------------------------------------------------------------+
| dn1.0         | BASE SQL | select `a`.`id`,`a`.`c_char`,`a`.`ts`,`a`.`si` from  `sharding_two_node` `a` ORDER BY `a`.`id` ASC |
| dn2.0         | BASE SQL | select `a`.`id`,`a`.`c_char`,`a`.`ts`,`a`.`si` from  `sharding_two_node` `a` ORDER BY `a`.`id` ASC |
| dn1.1         | BASE SQL | select `b`.`id`,`b`.`c_flag`,`b`.`c_decimal` from  `sharding_four_node` `b` ORDER BY `b`.`id` ASC  |
| dn2.1         | BASE SQL | select `b`.`id`,`b`.`c_flag`,`b`.`c_decimal` from  `sharding_four_node` `b` ORDER BY `b`.`id` ASC  |
| dn3.0         | BASE SQL | select `b`.`id`,`b`.`c_flag`,`b`.`c_decimal` from  `sharding_four_node` `b` ORDER BY `b`.`id` ASC  |
| dn4.0         | BASE SQL | select `b`.`id`,`b`.`c_flag`,`b`.`c_decimal` from  `sharding_four_node` `b` ORDER BY `b`.`id` ASC  |
| merge.1       | MERGE    | dn1.0, dn2.0                                                                                              |
| merge.2       | MERGE    | dn1.1, dn2.1, dn3.0, dn4.0                                                                                |
| join.1        | JOIN     | merge.1, merge.2                                                                                          |
+---------------+----------+-------------------------------------------------------------------------------------------+
9 rows in set (0.00 sec)
```

再举例如:

```
mysql> explain select id from single union all select b.si from sharding_four_node a inner join  sharding_two_node b on a.id =b.id
+-----------------+-----------+-------------------------------------------------------------------------+
| SHARDING_NODE   | TYPE      | SQL/REF                                                                 |
+-----------------+-----------+-------------------------------------------------------------------------+
| dn1.0           | BASE SQL  | select `single`.`id` from  `single`                                     |
| dn1.1           | BASE SQL  | select `a`.`id` from  `sharding_four_node` `a` ORDER BY `a`.`id` ASC    |
| dn2.0           | BASE SQL  | select `a`.`id` from  `sharding_four_node` `a` ORDER BY `a`.`id` ASC    |
| dn3.0           | BASE SQL  | select `a`.`id` from  `sharding_four_node` `a` ORDER BY `a`.`id` ASC    |
| dn4.0           | BASE SQL  | select `a`.`id` from  `sharding_four_node` `a` ORDER BY `a`.`id` ASC    |
| dn1.2           | BASE SQL  | select `b`.`si`,`b`.`id` from  `sharding_two_node` `b` ORDER BY `b`.`id` ASC |
| dn2.1           | BASE SQL  | select `b`.`si`,`b`.`id` from  `sharding_two_node` `b` ORDER BY `b`.`id` ASC |
| merge.2         | MERGE     | dn1.1, dn2.0, dn3.0, dn4.0                                              |
| merge.3         | MERGE     | dn1.2, dn2.1                                                            |
| join.1          | JOIN      | merge.2, merge.3                                                        |
| merge.1         | MERGE     | dn1.0                                                                   |
| union_all.1     | UNION_ALL | join.1, merge.1                                                         |
+-----------------+-----------+-------------------------------------------------------------------------+
12 rows in set (0.01 sec)
```

2.17.4 节点层的执行计划

通过EXPLAIN2命令可查看指定节点上的执行计划。如:
explain2 shardingnode=dn1 sql=select id,accountno from account where userid=2;
explain2会将sql语句加上explain下发到指定的shardingnode执行，并把节点上explain的结果返回调用者。

2.18 性能观测以及调试概览

- Btrace脚本性能观察(观察查询过程中每个不同阶段的耗时)
- manager命令show @@thread_used观察(观察不同线程的负载情况)

2.18.1 Btrace脚本观察

2.18.1.1 观察

Dble源码中自带观测脚本,BTraceCostTime.java文件是专用的Btrace观测脚本,当前脚本适用Btrace v.1.3
Btrace 相关资料https://github.com/btraceio/btrace
开启性能观察的统计需要配置bootstrap.cnf中的两个参数useCostTimeStat和costSamplePercent，分别是启用耗时监控的标志位以及耗时监控的采样百分比，useCostTimeStat = 1的状态下耗时监控被开启，并且以costSamplePercent的概率进行统计，默认的采样率是1%，采样率过低有可能导致btrace脚本输出没有统计结果的情况，采样率过高则会影响性能本身。 脚本中包含几个统计点如下图所示：

通过执行btrace监控正在运行的dble，可以获得类似以下的结果图：

```
profiling:
  Block                          Invocations  SelfTime.Total  SelfTime.Avg  SelfTime.Min  SelfTime.Max  WallTime.Total  WallTime.Avg  WallTime.Min  WallTime.Max
  request->1.startProcess               9073      -638142734        -70334      -1051058        493260       202952071         22368         10565        493260
  request->2.endParse                   9073       234134936         25805         13206        523393       437087007         48174         23771       1016653
  request->3.endRoute                   9073       404389553         44570         20123        121474       841476560         92745         43894       1075553
  request->4.resFromBack                9073          592398            65       -649019       1602901      4805691043        529669        261612       1602901
  request->5.startExecuteBackend        9073       -56808823         -6261      -1749483       2020297      5047581273        556329        350530       2020297
  request->6.response                   9073        59150286          6519          3045        366620      5107315454        562913        353575       2386917
```

以上输出值有效信息为Block,Invocations,WallTime. , 而*SelfTime.*由于btrace的原因 不准确，可以忽略。
其中通过记录一个查询在每个记录点的时间点来观察是否存在某个中间步骤耗时过长，并进行针对性的优化和调整

2.18.1.2 节点描述

- **0** : 逻辑时间0，来自客户端的请求首次被dble接收到的时间点
- **1.startProcess** : 开始处理前端请求的时间节点
- **2.endParse** : SQL被解析完成的时间节点
- **3.endRoute** : SQL被路由完成的时间节点
- **3.05 readyToDeliver** : SQL准备开始下发的时间点
- **4.resFromBack** : 前端请求首次由任意个后端连接返回信息
- **4.X.resFromBack** : 前端请求后端连接的首个返回包的采样时间点,例：4.3.resFromBack指的是第三个后端连接首次返回包给dble的时间点，具体的后端连接的编号为首次返回的顺序，即首个有网络包返回的后端连接编号为1

- **5.startExecuteBackend** : 前端请求后端连接首个包进入处理阶段的时间点
- **5.X.startExecuteBackend** : 前端请求后端连接的首个包被处理的采样时间点，例：5.3.startExecuteBackend指的是第三个后端连接首次进入到后端数据处理的阶段，具体的编号为首次进入处理阶段的顺序
- **5.1.allBackendConnReceive** ：在复杂查询的情况下开始merage的时间点
- **6.response** : 给前端开始最终返回包的时间点

注意**:Btrace**关于性能跟踪的脚本设计专为单一**SQL**模式下使用，在多个**SQL**混合查询的情况下，由于每个**SQL**涉及的采样点可能不同，会出现数据上的异常，甚至是节点顺序和时间点倒挂的现象

2.18.1.3 调整策略

- 1-0的耗时增长：需增大bootstrap.cnf中processorExecutor的值 （其中0比表示dble开始读到某条语句时的时间，即初始值：0，所以1-0的时间就是request->1.startProcess的输出值，不需要进行计算）
- 4-3的耗时增长：需增大bootstrap.cnf中backendProcessors的值
- 5-4的耗时增长：需增大bootstrap.cnf中backendBusinessExecutor的值

2.18.2 Manager命令观察

2.18.2.1 观察

Dble 在18.02.0版本中新添加了manager端口的性能观测命令，可以通过命令查看各个线程的负载情况，需要配合在配置文件bootstrap.cnf中的新增参数useThreadUsageStat进行使用
使用命令show @@thread_used会返回各个dble中关键线程最近时间的负载情况，如下示例

```
mysql> show @@thread_used;
+-------------------------+------------------+-------------+------------------+
| THREAD_NAME             | LAST_QUARTER_MIN | LAST_MINUTE | LAST_FIVE_MINUTE |
+-------------------------+------------------+-------------+------------------+
| backendBusinessExecutor2 | 0%               | 0%          | 0%               |
| backendBusinessExecutor1 | 0%               | 0%          | 0%               |
| backendBusinessExecutor0 | 0%               | 0%          | 0%               |
| BusinessExecutor3        | 0%               | 0%          | 0%               |
| $_NIO_REACTOR_BACKEND-2  | 0%               | 0%          | 0%               |
| BusinessExecutor1        | 0%               | 0%          | 0%               |
| $_NIO_REACTOR_BACKEND-3  | 0%               | 0%          | 0%               |
| BusinessExecutor2        | 12%              | 3%          | 3%               |
| $_NIO_REACTOR_BACKEND-0  | 0%               | 0%          | 0%               |
| $_NIO_REACTOR_FRONT-0    | 0%               | 0%          | 0%               |
| $_NIO_REACTOR_BACKEND-1  | 0%               | 0%          | 0%               |
| BusinessExecutor0        | 0%               | 0%          | 0%               |
+-------------------------+------------------+-------------+------------------+
12 rows in set (0.00 sec)
```

- BussinessExecutor**X**
  面向前端的业务处理线程，主要处理前端请求的解析，sql解析路由，下发查询到mysql实例等
- backendBusinessExecutor**X**
  面向后端的业务处理线程，主要处理后端mysql查询结果的返回解析，结果聚合，并发回结果到client
- $_NIO_REACTOR_FRONT_**X**
  前端请求接受线程,负责请求的接收和读取,之后把数据处理交给BussinessExecutor进行
- $_NIO_REACTOR_BACKEND_**X**
  后端请求接收线程，负责mysql返回信息的接收和读取，之后把数据交给backendBusinessExecutor处理

2.18.2.2 调整策略

当输出的统计结果中，有一个或者多个类型的线程使用率过高（经验值为超过80%），可以适当调整对应的处理线程的数量

- processors 参数控制前端_NIO_REACTOR_FRONT_**X**的数量
- backendProcessors 参数控制后端_NIO_REACTOR_BACKEND_**X**的数量
- backendProcessorExecutor 参数控制backendBusinessExecutor**X**数量
- processorExecutor 参数控制BussinessExecutor**X**数量

2.19 智能计算reload

我们对reload @@config_all 做了重构，增加了对dbGroup的变化计算，使得reload行为对整个系统的影响变到最小。

变更的dbGroup，等效于删除旧的dbGroup，新增新的dbGroup

2.19.1 默认reload @@config_all

连接池行为描述:

2.19.1.1 不变的dbGroup

此种情况下,dbGroup连接不发生变化，如果关联的schmea发生变更,在需要使用时候进行连接新建或者偷取并同步上下文的方式进行更新。

2.19.1.2 新增的dbGroup

建立新的连接池供使用

2.19.1.3 删除的dbGroup

遍历当前连接池，如果没有事务正在使用连接，则回收，否则放回后端待回收连接池 （在show backend中任可以看到放入回收池的时间），等事务结束时候连接被关闭

2.19.2 reload @@config_all -f

强制回收所有正在使用的链接
连接池行为描述:

2.19.2.1 不变的dbGroup

此种情况下,dbGroup连接不发生变化，正在使用的连接会被回收

2.19.2.2 新增的dbGroup

建立新的连接池供使用

2.19.2.3 删除的dbGroup

遍历如果当前连接池，如果没有事务正在使用连接，则回收，否则关闭对应的前端连接以及相关的后端连接

2.19.3 reload @@config_all -r

不计算dbGroup的变化，相当于原本所有的连接池会被删除，然后新建所有的连接池

遍历旧连接池，如果没有事务正在使用连接，则回收，否则放回后端待回收连接池 （在show backend中任可以看到放入回收池的时间），等事务结束时候连接被关闭

2.19.4 reload @@config_all -s

跳过dbGroup连接检查（与节点建立连接，连接成功，检测通过）

2.20 慢查询日志

类似于MySQL的慢查询日志，可以全局开启并设置，记录dble server运行过程当中的慢查询日志，日志格式兼容MySQL慢查询分析工具(已测试过MySQL官方工具mysqldumpslow和Percona的pt-query-digest)
此外，开启了慢查询日志工具之后，也可以查询某个连接的当前SQL的执行状态，相关命令为：show @@connection.sql.status where FRONT_ID= ?;

2.20.1 在bootstrap.cnf里增加了6个参数，用于启动时候控制慢查询日志的行为

```
<!-- 是否开启慢查询日志 -->
-DenableSlowLog=1
<!-- 慢查询日志保存文件目录 -->
-DslowLogBaseDir=./slowlogs
<!-- 慢查询日志保存文件前缀名称 -->
-DslowLogBaseName=slow-query
<!-- 日志两次刷盘之间的最大周期，单位是秒 -->
-DflushSlowLogPeriod=1
<!-- 日志两次刷盘之间内存中的最大条数阈值 -->
-DflushSlowLogSize=1000
<!-- 慢查询统计阈值，大于此值会被认为是慢查询 , 单位是毫秒 -->
-DsqlSlowTime=100
```

2.20.2 管理端口增加命令，用于运行过程中动态修改慢查询日志统计行为

```
enable @@slow_query_log; -- 开启慢查询日志
show @@slow_query_log; -- 查询慢查询日志的开启状态
disable @@slow_query_log; -- 关闭慢查询日志
show @@slow_query_log; -- 再次查询慢查询日志的开启状态

show @@slow_query.time;  -- 查看慢查询日志统计阈值
reload @@slow_query.time=200;  -- 修改慢查询日志统计阈值

show @@slow_query.flushperiod; -- 查看慢查询日志刷盘周期
reload @@slow_query.flushperiod=2;  -- 修改慢查询日志刷盘周期

show @@slow_query.flushsize;-- 查看慢查询日志刷盘条数阈值
reload @@slow_query.flushsize=1100;  -- 修改慢查询日志刷盘条数阈值
```

2.20.3 支持慢查询日志分析工具：MySQL的 mysqldumpslow 工具和Percona 的 pt-query-digest 工具

慢查询日志大概是这样的：

```
/FAKE_PATH/mysqld, Version: FAKE_VERSION. started with:
Tcp port: 3320  Unix socket: FAKE_SOCK
Time                 Id Command    Argument
# Time: 2018-08-23T17:40:10.149000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.132709  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000350  Prepare_Push: 0.116678  dn1_First_Result_Fetch: 0.013686  dn1_Last_Result_Fetch: 0.001422  Write_C
lient: 0.001995
SET timestamp=1535017210149;
select * from sharding_two_node where id =1;
# Time: 2018-08-23T17:40:10.200000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.035600  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000062  Prepare_Push: 0.006733  dn2_First_Result_Fetch: 0.012524  dn1_First_Result_Fetch: 0.010971  dn2_La
st_Result_Fetch: 0.015368  dn1_Last_Result_Fetch: 0.005119  Write_Client: 0.017834
SET timestamp=1535017210200;
select * from sharding_two_node;
# Time: 2018-08-23T17:40:10.282000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.045337  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000166  Prepare_Push: 0.003941  dn1_First_Result_Fetch: 0.039652  dn1_Last_Result_Fetch: 0.000300  Write_C
lient: 0.001578
SET timestamp=1535017210282;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:10.315000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.031232  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.005467  Prepare_Push: 0.001989  dn2_First_Result_Fetch: 0.020240  dn2_Last_Result_Fetch: 0.001900  Write_C
lient: 0.003536
SET timestamp=1535017210315;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:10.432000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.116672  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.013625  Prepare_Push: 0.024767  dn2_First_Result_Fetch: 0.056395  dn1_First_Result_Fetch: 0.026420  dn2_La
st_Result_Fetch: 0.000743  dn1_Last_Result_Fetch: 0.001700  Write_Client: 0.051861
SET timestamp=1535017210432;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:10.772000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.338569  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000082  Prepare_Push: 0.258365  dn1_0_First_Result_Fetch: 0.047494  dn1_0_Last_Result_Fetch: 0.029018  dn2
_0_First_Result_Fetch: 0.042964  dn2_0_Last_Result_Fetch: 0.033525  Write_Client: 0.009385
SET timestamp=1535017210772;
select count(*) from sharding_two_node;
# Time: 2018-08-23T17:40:10.821000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.046745  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000059  Prepare_Push: 0.025401  dn1_0_First_Result_Fetch: 0.011755  dn1_0_Last_Result_Fetch: 0.001180  Gen
erate_New_Query: 0.001706  dn1_1_First_Result_Fetch: 0.004224  dn1_1_Last_Result_Fetch: 0.001213  Write_Client: 0.001384
SET timestamp=1535017210821;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:12.061000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.036952  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.001111  Prepare_Push: 0.001132  dn1_First_Result_Fetch: 0.034266  dn1_Last_Result_Fetch: 0.000084  Write_C
lient: 0.000443
SET timestamp=1535017212061;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:12.091000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.028213  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000666  Prepare_Push: 0.001206  dn2_First_Result_Fetch: 0.025991  dn2_Last_Result_Fetch: 0.000101  Write_C
lient: 0.000349
SET timestamp=1535017212091;
```

```
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:12.132000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.040365  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000223  Prepare_Push: 0.001172  dn2_First_Result_Fetch: 0.019852  dn1_First_Result_Fetch: 0.019810  dn2_La
st_Result_Fetch: 0.000901  dn1_Last_Result_Fetch: 0.000780  Write_Client: 0.019160
SET timestamp=1535017212132;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:12.145000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.012196  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000115  Prepare_Push: 0.001403  dn1_0_First_Result_Fetch: 0.006714  dn1_0_Last_Result_Fetch: 0.002561  dn2
_0_First_Result_Fetch: 0.006787  dn2_0_Last_Result_Fetch: 0.001806  Write_Client: 0.002280
SET timestamp=1535017212145;
select count(*) from sharding_two_node;
# Time: 2018-08-23T17:40:12.164000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.016979  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000224  Prepare_Push: 0.002236  dn1_0_First_Result_Fetch: 0.006678  dn1_0_Last_Result_Fetch: 0.000703  Gen
erate_New_Query: 0.000866  dn1_1_First_Result_Fetch: 0.004532  dn1_1_Last_Result_Fetch: 0.000879  Write_Client: 0.001002
SET timestamp=1535017212164;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:13.134000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.010213  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000145  Prepare_Push: 0.001520  dn1_First_Result_Fetch: 0.007996  dn1_Last_Result_Fetch: 0.000201  Write_C
lient: 0.000551
SET timestamp=1535017213134;
select * from sharding_two_node where id =1;
# Time: 2018-08-23T17:40:13.153000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.014257  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000080  Prepare_Push: 0.002394  dn2_First_Result_Fetch: 0.008839  dn1_First_Result_Fetch: 0.008837  dn2_La
st_Result_Fetch: 0.001424  dn1_Last_Result_Fetch: 0.002407  Write_Client: 0.002945
SET timestamp=1535017213153;
select * from sharding_two_node;
# Time: 2018-08-23T17:40:13.212000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.029822  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000063  Prepare_Push: 0.001128  dn1_First_Result_Fetch: 0.028277  dn1_Last_Result_Fetch: 0.000109  Write_C
lient: 0.000355
SET timestamp=1535017213212;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:13.240000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.027695  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000067  Prepare_Push: 0.000682  dn2_First_Result_Fetch: 0.026582  dn2_Last_Result_Fetch: 0.000078  Write_C
lient: 0.000364
SET timestamp=1535017213240;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:13.321000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.076093  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000231  Prepare_Push: 0.001334  dn2_First_Result_Fetch: 0.035072  dn1_First_Result_Fetch: 0.035074  dn2_La
st_Result_Fetch: 0.018756  dn1_Last_Result_Fetch: 0.001263  Write_Client: 0.039457
SET timestamp=1535017213321;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:13.348000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.026278  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000335  Prepare_Push: 0.001249  dn1_0_First_Result_Fetch: 0.011028  dn1_0_Last_Result_Fetch: 0.009279  dn2
_0_First_Result_Fetch: 0.019200  dn2_0_Last_Result_Fetch: 0.003441  Write_Client: 0.004600
SET timestamp=1535017213348;
select count(*) from sharding_two_node;
# Time: 2018-08-23T17:40:13.381000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.029152  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000662  Prepare_Push: 0.003189  dn1_0_First_Result_Fetch: 0.014453  dn1_0_Last_Result_Fetch: 0.001013  Gen
erate_New_Query: 0.000911  dn1_1_First_Result_Fetch: 0.005703  dn1_1_Last_Result_Fetch: 0.001483  Write_Client: 0.002114
SET timestamp=1535017213381;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:14.163000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.012540  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000197  Prepare_Push: 0.001303  dn2_First_Result_Fetch: 0.006452  dn1_First_Result_Fetch: 0.007858  dn2_La
st_Result_Fetch: 0.004065  dn1_Last_Result_Fetch: 0.002960  Write_Client: 0.004588
SET timestamp=1535017214163;
select * from sharding_two_node;
# Time: 2018-08-23T17:40:14.220000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.027587  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000051  Prepare_Push: 0.000744  dn1_First_Result_Fetch: 0.026441  dn1_Last_Result_Fetch: 0.000104  Write_C
lient: 0.000350
SET timestamp=1535017214220;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:14.253000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.031984  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000070  Prepare_Push: 0.001144  dn2_First_Result_Fetch: 0.030202  dn2_Last_Result_Fetch: 0.000182  Write_C
lient: 0.000568
SET timestamp=1535017214253;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:14.292000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.037327  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000272  Prepare_Push: 0.001316  dn2_First_Result_Fetch: 0.014299  dn1_First_Result_Fetch: 0.014331  dn2_La
st_Result_Fetch: 0.001148  dn1_Last_Result_Fetch: 0.000753  Write_Client: 0.021440
SET timestamp=1535017214292;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:14.303000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.010244  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000050  Prepare_Push: 0.001101  dn1_0_First_Result_Fetch: 0.004540  dn1_0_Last_Result_Fetch: 0.002781  dn2
_0_First_Result_Fetch: 0.004708  dn2_0_Last_Result_Fetch: 0.002592  Write_Client: 0.002092
SET timestamp=1535017214303;
select count(*) from sharding_two_node;
# Time: 2018-08-23T17:40:14.327000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.021078  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000109  Prepare_Push: 0.002098  dn1_0_First_Result_Fetch: 0.006720  dn1_0_Last_Result_Fetch: 0.000748  Gen
erate_New_Query: 0.001158  dn1_1_First_Result_Fetch: 0.008043  dn1_1_Last_Result_Fetch: 0.001147  Write_Client: 0.001269
SET timestamp=1535017214327;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:15.254000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.010569  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000076  Prepare_Push: 0.001050  dn1_First_Result_Fetch: 0.008330  dn1_Last_Result_Fetch: 0.000146  Write_C
lient: 0.001113
SET timestamp=1535017215254;
select * from sharding_two_node where id =1;
# Time: 2018-08-23T17:40:15.321000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.024216  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000081  Prepare_Push: 0.001295  dn1_First_Result_Fetch: 0.021938  dn1_Last_Result_Fetch: 0.000422  Write_C
lient: 0.000902
```

```
SET timestamp=1535017215321;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:15.351000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.027796  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000666  Prepare_Push: 0.000760  dn2_First_Result_Fetch: 0.025984  dn2_Last_Result_Fetch: 0.000094  Write_C
lient: 0.000386
SET timestamp=1535017215351;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:15.392000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.039805  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000217  Prepare_Push: 0.000804  dn2_First_Result_Fetch: 0.017410  dn1_First_Result_Fetch: 0.017468  dn2_La
st_Result_Fetch: 0.001490  dn1_Last_Result_Fetch: 0.001223  Write_Client: 0.021374
SET timestamp=1535017215392;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:15.410000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.017384  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000152  Prepare_Push: 0.001183  dn1_0_First_Result_Fetch: 0.005037  dn1_0_Last_Result_Fetch: 0.007164  dn2
_0_First_Result_Fetch: 0.008156  dn2_0_Last_Result_Fetch: 0.004962  Write_Client: 0.004043
SET timestamp=1535017215410;
select count(*) from sharding_two_node;
# Time: 2018-08-23T17:40:15.434000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.021341  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000318  Prepare_Push: 0.002764  dn1_0_First_Result_Fetch: 0.010897  dn1_0_Last_Result_Fetch: 0.000544  Gen
erate_New_Query: 0.000798  dn1_1_First_Result_Fetch: 0.004506  dn1_1_Last_Result_Fetch: 0.000790  Write_Client: 0.000845
SET timestamp=1535017215434;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:16.322000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.030106  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000217  Prepare_Push: 0.001253  dn1_First_Result_Fetch: 0.028330  dn1_Last_Result_Fetch: 0.000086  Write_C
lient: 0.000306
SET timestamp=1535017216322;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:16.353000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.030005  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.001306  Prepare_Push: 0.001004  dn2_First_Result_Fetch: 0.027242  dn2_Last_Result_Fetch: 0.000140  Write_C
lient: 0.000453
SET timestamp=1535017216353;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:16.403000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.049615  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.001459  Prepare_Push: 0.000830  dn2_First_Result_Fetch: 0.024286  dn1_First_Result_Fetch: 0.025469  dn2_La
st_Result_Fetch: 0.001726  dn1_Last_Result_Fetch: 0.000853  Write_Client: 0.023039
SET timestamp=1535017216403;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:16.526000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.121702  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000337  Prepare_Push: 0.000889  dn1_0_First_Result_Fetch: 0.009370  dn1_0_Last_Result_Fetch: 0.002010  dn2
_0_First_Result_Fetch: 0.009160  dn2_0_Last_Result_Fetch: 0.001779  Write_Client: 0.109753
SET timestamp=1535017216526;
select count(*) from sharding_two_node;
# Time: 2018-08-23T17:40:16.560000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.030306  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.001534  Prepare_Push: 0.001759  dn1_0_First_Result_Fetch: 0.011846  dn1_0_Last_Result_Fetch: 0.001663  Gen
erate_New_Query: 0.003223  dn1_1_First_Result_Fetch: 0.006428  dn1_1_Last_Result_Fetch: 0.002601  Write_Client: 0.002291
SET timestamp=1535017216560;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:17.325000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.017545  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.006231  Prepare_Push: 0.002335  dn1_First_Result_Fetch: 0.008121  dn1_Last_Result_Fetch: 0.000277  Write_C
lient: 0.000857
SET timestamp=1535017217325;
select * from sharding_two_node where id =1;
# Time: 2018-08-23T17:40:17.390000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.026216  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000854  Prepare_Push: 0.000904  dn1_First_Result_Fetch: 0.024157  dn1_Last_Result_Fetch: 0.000081  Write_C
lient: 0.000301
SET timestamp=1535017217390;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:17.411000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.020095  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000699  Prepare_Push: 0.000711  dn2_First_Result_Fetch: 0.017634  dn2_Last_Result_Fetch: 0.000132  Write_C
lient: 0.001051
SET timestamp=1535017217411;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:17.491000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.078505  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.001702  Prepare_Push: 0.000763  dn2_First_Result_Fetch: 0.018547  dn1_First_Result_Fetch: 0.018482  dn2_La
st_Result_Fetch: 0.036637  dn1_Last_Result_Fetch: 0.000566  Write_Client: 0.057558
SET timestamp=1535017217491;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:17.518000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.026112  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000686  Prepare_Push: 0.000872  dn1_0_First_Result_Fetch: 0.007054  dn1_0_Last_Result_Fetch: 0.001072  dn2
_0_First_Result_Fetch: 0.005839  dn2_0_Last_Result_Fetch: 0.017248  Write_Client: 0.016586
SET timestamp=1535017217518;
select count(*) from sharding_two_node;
# Time: 2018-08-23T17:40:17.558000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.038199  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000966  Prepare_Push: 0.005189  dn1_0_First_Result_Fetch: 0.013137  dn1_0_Last_Result_Fetch: 0.001134  Gen
erate_New_Query: 0.003973  dn1_1_First_Result_Fetch: 0.010228  dn1_1_Last_Result_Fetch: 0.003564  Write_Client: 0.002115
SET timestamp=1535017217558;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:18.353000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.019048  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.003008  Prepare_Push: 0.000844  dn2_First_Result_Fetch: 0.006415  dn1_First_Result_Fetch: 0.009082  dn2_La
st_Result_Fetch: 0.000323  dn1_Last_Result_Fetch: 0.005902  Write_Client: 0.008781
SET timestamp=1535017218353;
select * from sharding_two_node;
# Time: 2018-08-23T17:40:18.410000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.025498  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000060  Prepare_Push: 0.000696  dn1_First_Result_Fetch: 0.024394  dn1_Last_Result_Fetch: 0.000084  Write_C
lient: 0.000348
SET timestamp=1535017218410;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:18.430000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.018794  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000047  Prepare_Push: 0.001301  dn2_First_Result_Fetch: 0.017073  dn2_Last_Result_Fetch: 0.000099  Write_C
```

```
lient: 0.000373
SET timestamp=1535017218430;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:18.471000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.039810  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000052  Prepare_Push: 0.000661  dn2_First_Result_Fetch: 0.019799  dn1_First_Result_Fetch: 0.019923  dn2_La
st_Result_Fetch: 0.000698  dn1_Last_Result_Fetch: 0.000814  Write_Client: 0.019298
SET timestamp=1535017218471;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:18.484000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.012214  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000047  Prepare_Push: 0.001782  dn1_0_First_Result_Fetch: 0.007109  dn1_0_Last_Result_Fetch: 0.001544  dn2
_0_First_Result_Fetch: 0.005518  dn2_0_Last_Result_Fetch: 0.001470  Write_Client: 0.003568
SET timestamp=1535017218484;
select count(*) from sharding_two_node;
# Time: 2018-08-23T17:40:18.507000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.019695  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000046  Prepare_Push: 0.001448  dn1_0_First_Result_Fetch: 0.006244  dn1_0_Last_Result_Fetch: 0.000988  Gen
erate_New_Query: 0.001564  dn1_1_First_Result_Fetch: 0.007080  dn1_1_Last_Result_Fetch: 0.001306  Write_Client: 0.001137
SET timestamp=1535017218507;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:19.351000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.020937  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000059  Prepare_Push: 0.000800  dn1_First_Result_Fetch: 0.019607  dn1_Last_Result_Fetch: 0.000169  Write_C
lient: 0.000472
SET timestamp=1535017219351;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:19.370000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.018011  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.001184  Prepare_Push: 0.000583  dn2_First_Result_Fetch: 0.015894  dn2_Last_Result_Fetch: 0.000129  Write_C
lient: 0.000351
SET timestamp=1535017219370;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:19.412000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.041319  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000689  Prepare_Push: 0.000573  dn2_First_Result_Fetch: 0.017735  dn1_First_Result_Fetch: 0.017876  dn2_La
st_Result_Fetch: 0.000601  dn1_Last_Result_Fetch: 0.000806  Write_Client: 0.022322
SET timestamp=1535017219412;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:19.423000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.010063  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000200  Prepare_Push: 0.001136  dn1_0_First_Result_Fetch: 0.006820  dn1_0_Last_Result_Fetch: 0.000694  dn2
_0_First_Result_Fetch: 0.003519  dn2_0_Last_Result_Fetch: 0.003944  Write_Client: 0.001443
SET timestamp=1535017219423;
select count(*) from sharding_two_node;
# Time: 2018-08-23T17:40:19.454000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.027592  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000182  Prepare_Push: 0.012798  dn1_0_First_Result_Fetch: 0.005960  dn1_0_Last_Result_Fetch: 0.000530  Gen
erate_New_Query: 0.000811  dn1_1_First_Result_Fetch: 0.005659  dn1_1_Last_Result_Fetch: 0.000926  Write_Client: 0.001101
SET timestamp=1535017219454;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:20.312000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.025903  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.001470  Prepare_Push: 0.000887  dn1_First_Result_Fetch: 0.022114  dn1_Last_Result_Fetch: 0.000197  Write_C
lient: 0.001433
SET timestamp=1535017220312;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:20.342000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.028503  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.005643  Prepare_Push: 0.001172  dn2_First_Result_Fetch: 0.021342  dn2_Last_Result_Fetch: 0.000074  Write_C
lient: 0.000346
SET timestamp=1535017220342;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:20.381000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.037424  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000641  Prepare_Push: 0.000959  dn2_First_Result_Fetch: 0.015139  dn1_First_Result_Fetch: 0.015238  dn2_La
st_Result_Fetch: 0.000795  dn1_Last_Result_Fetch: 0.000956  Write_Client: 0.020685
SET timestamp=1535017220381;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:20.408000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.016143  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000122  Prepare_Push: 0.001979  dn1_0_First_Result_Fetch: 0.004408  dn1_0_Last_Result_Fetch: 0.000484  Gen
erate_New_Query: 0.000965  dn1_1_First_Result_Fetch: 0.006059  dn1_1_Last_Result_Fetch: 0.001553  Write_Client: 0.000755
SET timestamp=1535017220408;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:21.214000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.023376  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000073  Prepare_Push: 0.001306  dn1_First_Result_Fetch: 0.021694  dn1_Last_Result_Fetch: 0.000081  Write_C
lient: 0.000302
SET timestamp=1535017221214;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:21.241000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.025408  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000083  Prepare_Push: 0.001029  dn2_First_Result_Fetch: 0.023856  dn2_Last_Result_Fetch: 0.000122  Write_C
lient: 0.000440
SET timestamp=1535017221241;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:21.281000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.038482  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000087  Prepare_Push: 0.000871  dn2_First_Result_Fetch: 0.016690  dn1_First_Result_Fetch: 0.016708  dn2_La
st_Result_Fetch: 0.000579  dn1_Last_Result_Fetch: 0.000891  Write_Client: 0.020835
SET timestamp=1535017221281;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:21.293000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.011657  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000615  Prepare_Push: 0.001320  dn1_0_First_Result_Fetch: 0.006906  dn1_0_Last_Result_Fetch: 0.001589  dn2
_0_First_Result_Fetch: 0.005105  dn2_0_Last_Result_Fetch: 0.001548  Write_Client: 0.003341
SET timestamp=1535017221293;
select count(*) from sharding_two_node;
# Time: 2018-08-23T17:40:21.312000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.017169  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000635  Prepare_Push: 0.001609  dn1_0_First_Result_Fetch: 0.006997  dn1_0_Last_Result_Fetch: 0.000728  Gen
erate_New_Query: 0.001037  dn1_1_First_Result_Fetch: 0.004816  dn1_1_Last_Result_Fetch: 0.000709  Write_Client: 0.000703
SET timestamp=1535017221312;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:22.150000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
```

```
# Query_time: 0.026153  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000180  Prepare_Push: 0.000771  dn1_First_Result_Fetch: 0.024940  dn1_Last_Result_Fetch: 0.000061  Write_C
lient: 0.000261
SET timestamp=1535017222150;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:22.170000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.019181  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000180  Prepare_Push: 0.000642  dn2_First_Result_Fetch: 0.018060  dn2_Last_Result_Fetch: 0.000088  Write_C
lient: 0.000299
SET timestamp=1535017222170;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:22.220000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.049834  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000261  Prepare_Push: 0.000735  dn2_First_Result_Fetch: 0.019862  dn1_First_Result_Fetch: 0.019807  dn2_La
st_Result_Fetch: 0.000418  dn1_Last_Result_Fetch: 0.000655  Write_Client: 0.029031
SET timestamp=1535017222220;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:22.240000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.019128  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000934  Prepare_Push: 0.002731  dn1_0_First_Result_Fetch: 0.013430  dn1_0_Last_Result_Fetch: 0.000521  dn2
_0_First_Result_Fetch: 0.003296  dn2_0_Last_Result_Fetch: 0.001243  Write_Client: 0.011072
SET timestamp=1535017222240;
select count(*) from sharding_two_node;
# Time: 2018-08-23T17:40:22.270000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.028479  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.003233  Prepare_Push: 0.004986  dn1_0_First_Result_Fetch: 0.009870  dn1_0_Last_Result_Fetch: 0.001172  Gen
erate_New_Query: 0.001590  dn1_1_First_Result_Fetch: 0.006060  dn1_1_Last_Result_Fetch: 0.000771  Write_Client: 0.000700
SET timestamp=1535017222270;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:23.097000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.053956  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000056  Prepare_Push: 0.001034  dn1_First_Result_Fetch: 0.052343  dn1_Last_Result_Fetch: 0.000174  Write_C
lient: 0.000523
SET timestamp=1535017223097;
select * from sharding_two_node where id =1;
# Time: 2018-08-23T17:40:23.110000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.010839  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000499  Prepare_Push: 0.000680  dn2_First_Result_Fetch: 0.006349  dn1_First_Result_Fetch: 0.009082  dn2_La
st_Result_Fetch: 0.000270  dn1_Last_Result_Fetch: 0.000333  Write_Client: 0.003311
SET timestamp=1535017223110;
select * from sharding_two_node;
# Time: 2018-08-23T17:40:23.181000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.027573  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000109  Prepare_Push: 0.000980  dn2_First_Result_Fetch: 0.026156  dn2_Last_Result_Fetch: 0.000086  Write_C
lient: 0.000328
SET timestamp=1535017223181;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:23.231000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.049380  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.002435  Prepare_Push: 0.000670  dn2_First_Result_Fetch: 0.025278  dn1_First_Result_Fetch: 0.025242  dn2_La
st_Result_Fetch: 0.000392  dn1_Last_Result_Fetch: 0.000629  Write_Client: 0.021032
SET timestamp=1535017223231;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:23.268000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.025207  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000060  Prepare_Push: 0.001492  dn1_0_First_Result_Fetch: 0.007693  dn1_0_Last_Result_Fetch: 0.000752  Gen
erate_New_Query: 0.001946  dn1_1_First_Result_Fetch: 0.008776  dn1_1_Last_Result_Fetch: 0.005040  Write_Client: 0.001884
SET timestamp=1535017223268;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:24.121000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.027104  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.001558  Prepare_Push: 0.001107  dn1_First_Result_Fetch: 0.024084  dn1_Last_Result_Fetch: 0.000085  Write_C
lient: 0.000356
SET timestamp=1535017224121;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:24.141000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.019191  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000072  Prepare_Push: 0.000673  dn2_First_Result_Fetch: 0.017923  dn2_Last_Result_Fetch: 0.000092  Write_C
lient: 0.000522
SET timestamp=1535017224141;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:24.182000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.039883  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000483  Prepare_Push: 0.000584  dn2_First_Result_Fetch: 0.017241  dn1_First_Result_Fetch: 0.017320  dn2_La
st_Result_Fetch: 0.000603  dn1_Last_Result_Fetch: 0.000767  Write_Client: 0.021575
SET timestamp=1535017224182;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:24.196000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.012406  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000073  Prepare_Push: 0.000958  dn1_0_First_Result_Fetch: 0.008102  dn1_0_Last_Result_Fetch: 0.001255  dn2
_0_First_Result_Fetch: 0.007566  dn2_0_Last_Result_Fetch: 0.001772  Write_Client: 0.002300
SET timestamp=1535017224196;
select count(*) from sharding_two_node;
# Time: 2018-08-23T17:40:24.218000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.021238  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000864  Prepare_Push: 0.001143  dn1_0_First_Result_Fetch: 0.010305  dn1_0_Last_Result_Fetch: 0.000532  Gen
erate_New_Query: 0.001852  dn1_1_First_Result_Fetch: 0.005359  dn1_1_Last_Result_Fetch: 0.000618  Write_Client: 0.000661
SET timestamp=1535017224218;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:25.093000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.029579  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000106  Prepare_Push: 0.000882  dn1_First_Result_Fetch: 0.028241  dn1_Last_Result_Fetch: 0.000069  Write_C
lient: 0.000351
SET timestamp=1535017225093;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:25.121000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.027422  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.001867  Prepare_Push: 0.001330  dn2_First_Result_Fetch: 0.023887  dn2_Last_Result_Fetch: 0.000102  Write_C
lient: 0.000338
SET timestamp=1535017225121;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:25.161000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.038859  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000050  Prepare_Push: 0.000753  dn2_First_Result_Fetch: 0.019091  dn1_First_Result_Fetch: 0.019189  dn2_La
st_Result_Fetch: 0.000582  dn1_Last_Result_Fetch: 0.000560  Write_Client: 0.018965
SET timestamp=1535017225161;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:25.191000Z
```

```
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.016379  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000043  Prepare_Push: 0.001276  dn1_0_First_Result_Fetch: 0.007469  dn1_0_Last_Result_Fetch: 0.000678  Gen
erate_New_Query: 0.001327  dn1_1_First_Result_Fetch: 0.003927  dn1_1_Last_Result_Fetch: 0.000787  Write_Client: 0.000893
SET timestamp=1535017225191;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:26.026000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.029878  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000162  Prepare_Push: 0.000916  dn1_First_Result_Fetch: 0.028497  dn1_Last_Result_Fetch: 0.000084  Write_C
lient: 0.000303
SET timestamp=1535017226026;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:26.051000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.024231  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.001105  Prepare_Push: 0.000469  dn2_First_Result_Fetch: 0.022188  dn2_Last_Result_Fetch: 0.000100  Write_C
lient: 0.000470
SET timestamp=1535017226051;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:26.091000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.039762  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.001669  Prepare_Push: 0.001915  dn2_First_Result_Fetch: 0.018107  dn1_First_Result_Fetch: 0.018187  dn2_La
st_Result_Fetch: 0.000633  dn1_Last_Result_Fetch: 0.000832  Write_Client: 0.018071
SET timestamp=1535017226091;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:26.105000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.012664  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000261  Prepare_Push: 0.000935  dn1_0_First_Result_Fetch: 0.007328  dn1_0_Last_Result_Fetch: 0.000733  dn2
_0_First_Result_Fetch: 0.006229  dn2_0_Last_Result_Fetch: 0.002592  Write_Client: 0.003554
SET timestamp=1535017226105;
select count(*) from sharding_two_node;
# Time: 2018-08-23T17:40:26.134000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.028335  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000046  Prepare_Push: 0.003442  dn1_0_First_Result_Fetch: 0.009563  dn1_0_Last_Result_Fetch: 0.001069  Gen
erate_New_Query: 0.001856  dn1_1_First_Result_Fetch: 0.010875  dn1_1_Last_Result_Fetch: 0.000798  Write_Client: 0.000712
SET timestamp=1535017226134;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
# Time: 2018-08-23T17:40:26.859000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.014882  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000067  Prepare_Push: 0.001351  dn1_First_Result_Fetch: 0.013084  dn1_Last_Result_Fetch: 0.000137  Write_C
lient: 0.000381
SET timestamp=1535017226859;
select * from sharding_two_node where id =1;
# Time: 2018-08-23T17:40:26.874000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.010509  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000066  Prepare_Push: 0.001761  dn2_First_Result_Fetch: 0.006921  dn1_First_Result_Fetch: 0.008256  dn2_La
st_Result_Fetch: 0.000279  dn1_Last_Result_Fetch: 0.000211  Write_Client: 0.001761
SET timestamp=1535017226874;
select * from sharding_two_node;
# Time: 2018-08-23T17:40:26.931000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.028690  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000237  Prepare_Push: 0.001126  dn1_First_Result_Fetch: 0.026194  dn1_Last_Result_Fetch: 0.000640  Write_C
lient: 0.001133
SET timestamp=1535017226931;
delete from sharding_two_node where id =15;
# Time: 2018-08-23T17:40:26.951000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.018818  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000342  Prepare_Push: 0.001671  dn2_First_Result_Fetch: 0.016482  dn2_Last_Result_Fetch: 0.000063  Write_C
lient: 0.000323
SET timestamp=1535017226951;
delete from sharding_two_node where id =519;
# Time: 2018-08-23T17:40:26.991000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.039399  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000082  Prepare_Push: 0.000706  dn2_First_Result_Fetch: 0.019233  dn1_First_Result_Fetch: 0.019167  dn2_La
st_Result_Fetch: 0.000426  dn1_Last_Result_Fetch: 0.000739  Write_Client: 0.019444
SET timestamp=1535017226991;
insert into sharding_two_node values(15,'15',15),(519,'519',519);
# Time: 2018-08-23T17:40:27.032000Z
# User@Host: root[root] @  [0:0:0:0:0:0:0:1]  Id:   2
# Query_time: 0.029495  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0  Read_SQL: 0.000064  Prepare_Push: 0.001349  dn1_0_First_Result_Fetch: 0.005745  dn1_0_Last_Result_Fetch: 0.000632  Gen
erate_New_Query: 0.001056  dn1_1_First_Result_Fetch: 0.018101  dn1_1_Last_Result_Fetch: 0.002282  Write_Client: 0.000863
SET timestamp=1535017227032;
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
```

2.20.3.1 mysqldumpslow 结果:

```
Reading mysql slow query log from /tmp/slow3.log
Count: 17  Time=0.05s (0s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@[0:0:0:0:0:0:0:1]
  insert into sharding_two_node values(N,'S',N),(N,'S',N)

Count: 13  Time=0.05s (0s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@[0:0:0:0:0:0:0:1]
  select count(*) from sharding_two_node

Count: 6  Time=0.04s (0s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@[0:0:0:0:0:0:0:1]
  select * from sharding_two_node where id =N

Count: 33  Time=0.03s (0s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@[0:0:0:0:0:0:0:1]
  delete from sharding_two_node where id =N

Count: 17  Time=0.03s (0s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@[0:0:0:0:0:0:0:1]
  select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=N)

Count: 6  Time=0.02s (0s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@[0:0:0:0:0:0:0:1]
  select * from sharding_two_node
```

2.20.3.2 pt-query-digest 结果:

```
# 710ms user time, 70ms system time, 23.35M rss, 68.36M vsz
# Current date: Thu Aug 23 17:48:25 2018
# Hostname: 10-186-24-63
# Files: /tmp/slow_query4.log
# Overall: 92 total, 6 unique, 5.41 QPS, 0.18x concurrency _______________
# Time range: 2018-08-23T17:40:10 to 2018-08-23T17:40:27
```

```
# Attribute          total     min     max     avg     95%  stddev  median
# ============     ======= ======= ======= ======= ======= ======= =======
# Exec time             3s    10ms   339ms    34ms    75ms    37ms    27ms
# Lock time              0       0       0       0       0       0       0
# Rows sent              0       0       0       0       0       0       0
# Rows examine           0       0       0       0       0       0       0
# Query size         4.91k      31      94   54.64   92.72   20.87   42.48
# Generate New        0.03    0.00    0.00    0.00    0.00    0.00    0.00
# Prepare Push        0.56    0.00    0.26    0.01    0.01    0.03    0.00
# Read SQL            0.07    0.00    0.01    0.00    0.00    0.00    0.00
# Write Client        0.70    0.00    0.11    0.01    0.02    0.01    0.00
# dn1 0 First         0.29    0.00    0.05    0.01    0.01    0.01    0.01
# dn1 0 Last R        0.07    0.00    0.03    0.00    0.01    0.01    0.00
# dn1 1 First         0.12    0.00    0.02    0.01    0.01    0.00    0.01
# dn1 1 Last R        0.03    0.00    0.01    0.00    0.00    0.00    0.00
# dn1 First Re        0.93    0.01    0.05    0.02    0.03    0.01    0.02
# dn1 Last Res        0.04    0.00    0.01    0.00    0.00    0.00    0.00
# dn2 0 First         0.13    0.00    0.04    0.01    0.02    0.01    0.01
# dn2 0 Last R        0.08    0.00    0.03    0.01    0.02    0.01    0.00
# dn2 First Re        0.80    0.01    0.06    0.02    0.03    0.01    0.02
# dn2 Last Res        0.09    0.00    0.04    0.00    0.01    0.01    0.00

# Profile
# Rank Query ID                           Response time Calls R/Call V/M   I
# ==== ================================== ============= ===== ====== ===== =
#    1 0x13233F8ADA41C6E2D889AEE0C2B...  0.8815 28.1%    33 0.0267  0.00 DELETE sharding_two_node
#    2 0xF68D16B46E487184E8FD3BB3912...  0.8525 27.1%    17 0.0501  0.01 INSERT sharding_two_node
#    3 0xB46D813C53609C853F7CBA6D2DB...  0.6306 20.1%    13 0.0485  0.15 SELECT sharding_two_node
#    4 0x3FB41587E746A475282C1ED2606...  0.4335 13.8%    17 0.0255  0.00 SELECT sharding_two_node
#    5 0x04CDF91DDFC4E1DD7A22E312C72...  0.2399  7.6%     6 0.0400  0.05 SELECT sharding_two_node
# MISC 0xMISC                             0.1028  3.3%     6 0.0171   0.0 <1 ITEMS>

# Query 1: 2.06 QPS, 0.06x concurrency, ID 0x13233F8ADA41C6E2D889AEE0C2BC6CB5 at byte 943
# Scores: V/M = 0.00
# Time range: 2018-08-23T17:40:10 to 2018-08-23T17:40:26
# Attribute    pct   total     min     max     avg     95%  stddev  median
# ============ === ======= ======= ======= ======= ======= ======= =======
# Count         35      33
# Exec time     28   882ms    18ms    45ms    27ms    31ms     5ms    27ms
# Lock time      0       0       0       0       0       0       0       0
# Rows sent      0       0       0       0       0       0       0       0
# Rows examine   0       0       0       0       0       0       0       0
# Query size    27   1.37k      42      43   42.52   42.48    0.50   42.48
# Prepare Push   6    0.04    0.00    0.00    0.00    0.00    0.00    0.00
# Read SQL      35    0.03    0.00    0.01    0.00    0.00    0.00    0.00
# Write Client   2    0.02    0.00    0.00    0.00    0.00    0.00    0.00
# dn1 First Re  45    0.42    0.02    0.04    0.03    0.03    0.01    0.03
# dn1 Last Res   7    0.00    0.00    0.00    0.00    0.00    0.00    0.00
# dn2 First Re  47    0.38    0.02    0.03    0.02    0.03    0.00    0.02
# dn2 Last Res   3    0.00    0.00    0.00    0.00    0.00    0.00    0.00
# String:
# Hosts        0:0:0:0:0:0:0:1
# Users        root
# Query_time distribution
#   1us
#  10us
# 100us
#   1ms
#  10ms  ################################################################
# 100ms
#    1s
#  10s+
# Tables
#    SHOW TABLE STATUS LIKE 'sharding_two_node'\G
#    SHOW CREATE TABLE `sharding_two_node`\G
delete from sharding_two_node where id =15\G
# Converted for EXPLAIN
# EXPLAIN /*!50100 PARTITIONS*/
select * from  sharding_two_node where id =15\G

# Query 2: 1.06 QPS, 0.05x concurrency, ID 0xF68D16B46E487184E8FD3BB3912A3658 at byte 1690
# Scores: V/M = 0.01
# Time range: 2018-08-23T17:40:10 to 2018-08-23T17:40:26
# Attribute    pct   total     min     max     avg     95%  stddev  median
# ============ === ======= ======= ======= ======= ======= ======= =======
# Count         18      17
# Exec time     27   853ms    37ms   117ms    50ms    78ms    21ms    40ms
# Lock time      0       0       0       0       0       0       0       0
# Rows sent      0       0       0       0       0       0       0       0
# Rows examine   0       0       0       0       0       0       0       0
# Query size    21   1.06k      64      64      64      64       0      64
# Prepare Push   7    0.04    0.00    0.02    0.00    0.00    0.01    0.00
# Read SQL      32    0.02    0.00    0.01    0.00    0.00    0.00    0.00
# Write Client  63    0.45    0.02    0.06    0.03    0.05    0.01    0.02
# dn1 First Re  37    0.35    0.01    0.04    0.02    0.03    0.00    0.02
# dn1 Last Res  40    0.01    0.00    0.00    0.00    0.00    0.00    0.00
# dn2 First Re  46    0.37    0.01    0.06    0.02    0.03    0.01    0.02
# dn2 Last Res  72    0.07    0.00    0.04    0.00    0.02    0.01    0.00
# String:
# Hosts        0:0:0:0:0:0:0:1
# Users        root
# Query_time distribution
#   1us
#  10us
# 100us
#   1ms
#  10ms  ################################################################
# 100ms  ####
#    1s
#  10s+
# Tables
#    SHOW TABLE STATUS LIKE 'sharding_two_node'\G
#    SHOW CREATE TABLE `sharding_two_node`\G
insert into sharding_two_node values(15,'15',15) /*... omitted ...*/\G

# Query 3: 0.81 QPS, 0.04x concurrency, ID 0xB46D813C53609C853F7CBA6D2DB4047C at byte 2152
# Scores: V/M = 0.15
```

```
# Time range: 2018-08-23T17:40:10 to 2018-08-23T17:40:26
# Attribute    pct   total     min     max     avg     95%  stddev  median
# ============ === ======= ======= ======= ======= ======= ======= =======
# Count         14      13
# Exec time     20   631ms    10ms   339ms    49ms   116ms    84ms    12ms
# Lock time      0       0       0       0       0       0       0       0
# Rows sent      0       0       0       0       0       0       0       0
# Rows examine   0       0       0       0       0       0       0       0
# Query size     9     494      38      38      38      38       0      38
# Prepare Push  48    0.27    0.00    0.26    0.02    0.00    0.07    0.00
# Read SQL       5    0.00    0.00    0.00    0.00    0.00    0.00    0.00
# Write Client  24    0.17    0.00    0.11    0.01    0.02    0.03    0.00
# dn1 0 First   48    0.14    0.00    0.05    0.01    0.01    0.01    0.01
# dn1 0 Last R  80    0.06    0.00    0.03    0.00    0.01    0.01    0.00
# dn2 0 First  100    0.13    0.00    0.04    0.01    0.02    0.01    0.01
# dn2 0 Last R 100    0.08    0.00    0.03    0.01    0.02    0.01    0.00
# String:
# Hosts        0:0:0:0:0:0:0:1
# Users        root
# Query_time distribution
#   1us
#  10us
# 100us
#   1ms
#  10ms  ################################################################
# 100ms  ###########
#    1s
#  10s+
# Tables
#    SHOW TABLE STATUS LIKE 'sharding_two_node'\G
#    SHOW CREATE TABLE `sharding_two_node`\G
# EXPLAIN /*!50100 PARTITIONS*/
select count(*) from sharding_two_node\G

# Query 4: 1 QPS, 0.03x concurrency, ID 0x3FB41587E746A475282C1ED2606795FB at byte 2596
# Scores: V/M = 0.00
# Time range: 2018-08-23T17:40:10 to 2018-08-23T17:40:27
# Attribute    pct   total     min     max     avg     95%  stddev  median
# ============ === ======= ======= ======= ======= ======= ======= =======
# Count         18      17
# Exec time     13   434ms    16ms    47ms    26ms    38ms     8ms    26ms
# Lock time      0       0       0       0       0       0       0       0
# Rows sent      0       0       0       0       0       0       0       0
# Rows examine   0       0       0       0       0       0       0       0
# Query size    31   1.56k      94      94      94      94       0      94
# Generate New 100    0.03    0.00    0.00    0.00    0.00    0.00    0.00
# Prepare Push  13    0.07    0.00    0.03    0.00    0.01    0.01    0.00
# Read SQL      12    0.01    0.00    0.00    0.00    0.00    0.00    0.00
# Write Client   2    0.02    0.00    0.00    0.00    0.00    0.00    0.00
# dn1 0 First   51    0.15    0.00    0.01    0.01    0.01    0.00    0.01
# dn1 0 Last R  19    0.01    0.00    0.00    0.00    0.00    0.00    0.00
# dn1 1 First  100    0.12    0.00    0.02    0.01    0.01    0.00    0.01
# dn1 1 Last R 100    0.03    0.00    0.01    0.00    0.00    0.00    0.00
# String:
# Hosts        0:0:0:0:0:0:0:1
# Users        root
# Query_time distribution
#   1us
#  10us
# 100us
#   1ms
#  10ms  ################################################################
# 100ms
#    1s
#  10s+
# Tables
#    SHOW TABLE STATUS LIKE 'sharding_two_node'\G
#    SHOW CREATE TABLE `sharding_two_node`\G
# EXPLAIN /*!50100 PARTITIONS*/
select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1)\G

# Query 5: 0.38 QPS, 0.01x concurrency, ID 0x04CDF91DDFC4E1DD7A22E312C72C268D at byte 0
# Scores: V/M = 0.05
# Time range: 2018-08-23T17:40:10 to 2018-08-23T17:40:26
# Attribute    pct   total     min     max     avg     95%  stddev  median
# ============ === ======= ======= ======= ======= ======= ======= =======
# Count          6       6
# Exec time      7   240ms    10ms   133ms    40ms   128ms    43ms    35ms
# Lock time      0       0       0       0       0       0       0       0
# Rows sent      0       0       0       0       0       0       0       0
# Rows examine   0       0       0       0       0       0       0       0
# Query size     5     258      43      43      43      43       0      43
# Prepare Push  22    0.12    0.00    0.12    0.02    0.12    0.04    0.00
# Read SQL       9    0.01    0.00    0.01    0.00    0.01    0.00    0.00
# Write Client   0    0.01    0.00    0.00    0.00    0.00    0.00    0.00
# dn1 First Re  11    0.10    0.01    0.05    0.02    0.05    0.02    0.01
# dn1 Last Res   6    0.00    0.00    0.00    0.00    0.00    0.00    0.00
# String:
# Hosts        0:0:0:0:0:0:0:1
# Users        root
# Query_time distribution
#   1us
#  10us
# 100us
#   1ms
#  10ms  ################################################################
# 100ms  ############
#    1s
#  10s+
# Tables
#    SHOW TABLE STATUS LIKE 'sharding_two_node'\G
#    SHOW CREATE TABLE `sharding_two_node`\G
# EXPLAIN /*!50100 PARTITIONS*/
select * from sharding_two_node where id =1\G
```

2.20.3.3 限制

当mysqldumpslow版本为5.6或更早的版本时，在解析慢日志的过程中，由于mysqldumpslow只能识别'190428 10:28:16'格式的Time字段，而dble却使用了'2019-04-28T10:28:16.515000Z'字段格式，因此解析生成的慢日志会出错。详情参见issue:https://github.com/actiontech/dble/issues/908

2.20.4 附：慢查询日志格式解析

2.20.4.1 MySQL 慢查询日志格式

先放一段正常记录的MySQL慢日志

```
/usr/local/mysql5.7.11/bin/mysqld-debug, Version: 5.7.11-debug-log (MySQL Community Server - Debug (GPL)). started with:

Tcp port: 3320  Unix socket: /tmp/mysql_3320.sock

Time                 Id Command    Argument

# Time: 2018-05-15T10:53:23.798040Z

# User@Host: action[action] @  [192.168.2.206]  Id:   436

# Query_time: 296.145816  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0

use test;

SET timestamp=1526381603;

drop table sharding_two_node;

# Time: 2018-05-15T11:32:25.549290Z

# User@Host: action[action] @  [192.168.2.206]  Id:   451

# Query_time: 129.555883  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0

use nosharding;

SET timestamp=1526383945;

drop table test4;

# Time: 2018-05-15T11:32:25.550190Z

# User@Host: action[action] @  [192.168.2.206]  Id:   454

# Query_time: 84.316518  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0

SET timestamp=1526383945;

insert into test4 values(1,'1');

# Time: 2018-05-15T11:37:01.079214Z

# User@Host: action[action] @  [192.168.2.206]  Id:   483

# Query_time: 49.571983  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0

SET timestamp=1526384221;

drop table test3;

# Time: 2018-07-11T05:28:34.161405Z

# User@Host: action[action] @  [192.168.2.206]  Id: 16421

# Query_time: 10.035706  Lock_time: 0.000000 Rows_sent: 1  Rows_examined: 0

use test;

SET timestamp=1531286914;

insert into test4 values(1,'1');
```

我们从MySQL源代码上来分析慢日志的格式：

1.1 文件头：（包含版本信息等）

/usr/local/mysql5.7.11/bin/mysqld-debug, Version: 5.7.11-debug-log (MySQL Community Server - Debug (GPL)). started with:

Tcp port: 3320 Unix socket: /tmp/mysql_3320.sock

Time Id Command Argument

如果MySQL实例没有专门设置参数log-short-format，则会有time行和session信息行

1.2 time行

例如：

`# Time: 2018-05-15T10:53:23.798040Z`

1.3 session信息行

例如：

`# User@Host: action[action] @ [192.168.2.206] Id: 436`

1.4 执行时间行

注意用两个空格分隔不同键值对，用单个空格区分key和value

`# Query_time: 296.145816 Lock_time: 0.000000 Rows_sent: 0 Rows_examined: 0`

1.5 如果有**Database changed**，则有转换**schema**行：

`use nosharding;`

1.6 set行**(**注：**mysqldumpslow**官方工具只处理了**SET timestamp=)**

举例

`SET timestamp=1526383945;`

1.6.1 last_insert_id

如果设置了

`stmt_depends_on_first_successful_insert_id_in_prev_stmt` （含义待调查）

会有last_insert_id=某个值

1.6.2 insert_id

如果没有设置log-short-format，并且

`auto_inc_intervals_in_cur_stmt_for_binlog.nb_elements()` （含义待调查）

会有last_id=某个值

1.6.3 timestamp=

设置时间戳

1.7 命令行

例如：

`# administrator command`

如果是is_command（含义待调查），则有此行

1.8 SQL语句行

例如

`insert into test4 values(1,'1');`

2.20.4.2 dble 慢日志格式

为了兼容mysqldumpslow和pt-query-digest 工具，dble的慢日志格式如下：

2.1 文件头

```
/FAKE_PATH/mysqld, Version: FAKE_VERSION. started with:

Tcp port: 3320  Unix socket: FAKE_SOCK

Time                 Id Command    Argument
```

2.2 time行

因为java8目前只能获取到毫秒级别的绝对时间戳，所以时间戳后三位为0，与mysql不同， 例如：

`# Time: 2018-08-23T17:40:10.149000Z`

2.3 session信息行

例如：

`# User@Host: root[root] @ [0:0:0:0:0:0:0:1] Id: 2`

2.4 执行时间行

注意用两个空格分隔不同键值对，用单个空格区分key和value.

如图，这是dble的内部流程，根据此图：

我们在兼容MySQL的情况下增加了如下字段：

Read_SQL: 接受到SQL字节数据开始到将字节数据转为可供解析的SQL字符串 的耗时，即T1-T0

Prepare_Push：解析/路由/尝试优化及其他准备工作耗时，与Read_SQL是线性顺序的,即T2-T1

{$shardingnodeName}_First_Result_Fetch： 某个shardingnodeName收到的第一个数据包，相对于Prepare_Push结束时的耗时 ,即T3-T2 。另外：不同的shardingnodeName之间是并行的

{$shardingnodeName}_Last_Result_Fetch： 某个shardingnodeName收到的最后一个数据包，相对于自己的{$shardingnodeName}__First_Result_Fetch的耗时,即T5-T3

Write_Client：可以开始返回客户端（收到一个后端返回数据包）到最后一个数据包准备写出的时间，如果是复杂查询，可能包含合并等操作的结果,即T6-T4

另外，MySQL原有的Query_time为dble开始读取SQL到准备写回最后一个数据包的时间间隔,即T6-T0
Lock_time，Rows_sent，Rows_examined用0填充
例如：

```
# Query_time: 0.116672 Lock_time: 0.000000 Rows_sent: 0 Rows_examined: 0 Read_SQL: 0.013625 Prepare_Push: 0.024767 dn2_First_Result_Fetch: 0.056395 dn1_First_Result_Fetch: 0.026420 dn2_Last_Result_Fetch: 0.000743 dn1_Last_Result_Fetch: 0.001700 Write_Client: 0.051861
```

2.5 set行(只处理了SET timestamp=)

例如：

```
SET timestamp=1535017210432;
```

2.6 SQL语句行

例如

```
insert into sharding_two_node values(15,'15',15),(519,'519',519);
```

2.21 单条SQL性能trace

这个功能与MySQL的profile功能类似，用于统计一条查询在dble内部的各个阶段的耗时情况，用来发现性能瓶颈，改进SQL或者dble配置，达到提高性能的目的。 此功能为session 级别。

1. 可以通过此命令检查**trace**功能是否开启

```
mysql> select @@trace;
+---------+
| @@trace |
+---------+
| 0       |
+---------+
1 row in set (0.02 sec)
```

2. 开启 **trace** 功能

```
mysql> set trace =1;
Query OK, 0 rows affected (0.09 sec)

mysql> select @@trace;
+---------+
| @@trace |
+---------+
| 1       |
+---------+
1 row in set (0.00 sec)
```

3. 单节点查询的**trace**结果举例

```
mysql> select * from sharding_two_node where id =1;
+----+--------+-----------+
| id | c_flag | c_decimal |
+----+--------+-----------+
|  1 | 1      |    1.0000 |
+----+--------+-----------+
1 row in set (0.02 sec)

mysql> show trace;
+-------------------+-----------+-----------+--------------+---------------+-------------------------------------------+
| OPERATION         | START(ms) | END(ms)   | DURATION(ms) | SHARDING_NODE | SQL/REF                                   |
+-------------------+-----------+-----------+--------------+---------------+-------------------------------------------+
| Read SQL          | 0.0       | 0.1085    | 0.1085       | -             | -                                         |
| Parse SQL         | 0.1085    | 0.49607   | 0.38757      | -             | -                                         |
| Route Calculation | 0.49607   | 1.274142  | 0.778072     | -             | -                                         |
| Prepare to Push   | 1.274142  | 1.560543  | 0.286401     | -             | -                                         |
| Execute SQL       | 1.560543  | 18.711851 | 17.151308    | dn1           | select * from sharding_two_node where id =1 |
| Fetch result      | 18.711851 | 18.978213 | 0.266362     | dn1           | select * from sharding_two_node where id =1 |
| Write to Client   | 18.711851 | 19.276344 | 0.564493     | -             | -                                         |
| Over All          | 0.0       | 19.276344 | 19.276344    | -             | -                                         |
+-------------------+-----------+-----------+--------------+---------------+-------------------------------------------+
8 rows in set (0.00 sec)
```

4. 多节点查询的**trace**结果举例

```
mysql> select * from sharding_two_node ;
+-----+--------+-----------+
| id  | c_flag | c_decimal |
+-----+--------+-----------+
| 513 | 513    |  513.0000 |
| 514 | 514    |  514.0000 |
| 515 | 515    |  515.0000 |
| 516 | 516    |  516.0000 |
|   1 | 1      |    1.0000 |
|   2 | 2      |    2.0000 |
|   3 | 3      |    3.0000 |
|   4 | 4      |    4.0000 |
|   5 | 5      |    5.0000 |
|   7 | 7      |    7.0000 |
|   8 | 8      |    8.0000 |
|   9 | 9      |    9.0000 |
|  10 | 10     |   10.0000 |
|  11 | 11     |   11.0000 |
|  12 | 12     |   12.0000 |
+-----+--------+-----------+
15 rows in set (0.01 sec)

mysql> show trace;
+-------------------+-----------+----------+--------------+---------------+----------------------------------------+
| OPERATION         | START(ms) | END(ms)  | DURATION(ms) | SHARDING_NODE | SQL/REF                                |
+-------------------+-----------+----------+--------------+---------------+----------------------------------------+
| Read SQL          | 0.0       | 0.079175 | 0.079175     | -             | -                                      |
| Parse SQL         | 0.079175  | 0.637315 | 0.55814      | -             | -                                      |
| Route Calculation | 0.637315  | 1.046389 | 0.409074     | -             | -                                      |
| Prepare to Push   | 1.046389  | 1.465238 | 0.418849     | -             | -                                      |
| Execute SQL       | 1.465238  | 8.141409 | 6.676171     | dn1           | SELECT * FROM sharding_two_node LIMIT 100 |
| Execute SQL       | 1.465238  | 7.59109  | 6.125852     | dn2           | SELECT * FROM sharding_two_node LIMIT 100 |
| Fetch result      | 8.141409  | 8.817824 | 0.676415     | dn1           | SELECT * FROM sharding_two_node LIMIT 100 |
| Fetch result      | 7.59109   | 8.366718 | 0.775628     | dn2           | SELECT * FROM sharding_two_node LIMIT 100 |
| Write to Client   | 7.59109   | 9.324157 | 1.733067     | -             | -                                      |
| Over All          | 0.0       | 9.324157 | 9.324157     | -             | -                                      |
+-------------------+-----------+----------+--------------+---------------+----------------------------------------+
10 rows in set (0.00 sec)
```

5. 多节点写入的**SQL** 的**trace**结果举例**,**事实上**,**这是一个隐式分布式事务

```
mysql> insert into sharding_two_node values(15,'15',15),(519,'519',519);
Query OK, 2 rows affected (0.06 sec)

mysql> show trace;
+--------------------------------+-----------+-----------+--------------+---------------+-------------------------------------------------------+
| OPERATION                      | START(ms) | END(ms)   | DURATION(ms) | SHARDING_NODE | SQL/REF                                                |
+--------------------------------+-----------+-----------+--------------+---------------+-------------------------------------------------------+
| Read SQL                       | 0.0       | 0.131959  | 0.131959     | -             | -                                                      |
| Parse SQL                      | 0.131959  | 0.601637  | 0.469678     | -             | -                                                      |
| Route Calculation              | 0.601637  | 0.825479  | 0.223842     | -             | -                                                      |
| Prepare to Push                | 0.825479  | 1.025374  | 0.199895     | -             | -                                                      |
| Execute SQL                    | 1.025374  | 27.095675 | 26.070301    | dn1           | INSERT INTO sharding_two_node VALUES (15, '15', 15)    |
| Execute SQL                    | 1.025374  | 25.023911 | 23.998537    | dn2           | INSERT INTO sharding_two_node VALUES (519, '519', 519) |
| Fetch result                   | 27.095675 | 27.405046 | 0.309371     | dn1           | INSERT INTO sharding_two_node VALUES (15, '15', 15)    |
| Fetch result                   | 25.023911 | 26.478398 | 1.454487     | dn2           | INSERT INTO sharding_two_node VALUES (519, '519', 519) |
| Distributed Transaction Prepare | 27.405046 | 27.736411 | 0.331365     | -             | -                                                      |
| Distributed Transaction Commit  | 27.736411 | 57.426311 | 29.6899      | -             | -                                                      |
| Write to Client                | 25.023911 | 57.428266 | 32.404355    | -             | -                                                      |
| Over All                       | 0.0       | 57.428266 | 57.428266    | -             | -                                                      |
+--------------------------------+-----------+-----------+--------------+---------------+-------------------------------------------------------+
12 rows in set (0.00 sec)
```

6. 复杂查询的**trace**结果举例

```
mysql> select count(*) from sharding_two_node;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.01 sec)

mysql> show trace;
+-----------------------+-----------+-----------+--------------+-----------------+-----------------------------------------------------------------------+
| OPERATION             | START(ms) | END(ms)   | DURATION(ms) | SHARDING_NODE   | SQL/REF                                                                |
+-----------------------+-----------+-----------+--------------+-----------------+-----------------------------------------------------------------------+
| Read SQL              | 0.0       | 0.08553   | 0.08553      | -               | -                                                                      |
| Parse SQL             | 0.08553   | 0.56987   | 0.48434      | -               | -                                                                      |
| Try Route Calculation | 0.56987   | 0.71698   | 0.14711      | -               | -                                                                      |
| Try to Optimize       | 0.71698   | 1.237487  | 0.520507     | -               | -                                                                      |
| Execute SQL           | 1.237487  | 9.091029  | 7.853542     | dn1.0           | select COUNT(*) as `_$COUNT$_rpda_0` from  `sharding_two_node` LIMIT 100 |
| Fetch result          | 9.091029  | 10.186782 | 1.095753     | dn1.0           | select COUNT(*) as `_$COUNT$_rpda_0` from  `sharding_two_node` LIMIT 100 |
| Execute SQL           | 1.237487  | 8.348635  | 7.111148     | dn2.0           | select COUNT(*) as `_$COUNT$_rpda_0` from  `sharding_two_node` LIMIT 100 |
| Fetch result          | 8.348635  | 9.342241  | 0.993606     | dn2.0           | select COUNT(*) as `_$COUNT$_rpda_0` from  `sharding_two_node` LIMIT 100 |
| MERGE                 | 8.721543  | 10.289905 | 1.568362     | merge.1         | dn1.0; dn2.0                                                           |
| ORDERED_GROUP         | 8.726919  | 10.424309 | 1.69739      | ordered_group.1 | merge.1                                                                |
| LIMIT                 | 9.020162  | 10.499574 | 1.479412     | limit.1         | ordered_group.1                                                        |
| SHUFFLE_FIELD         | 9.023584  | 10.501529 | 1.477945     | shuffle_field.1 | limit.1                                                                |
| Write to Client       | 9.072457  | 11.52055  | 2.448093     | -               | -                                                                      |
| Over All              | 0.0       | 11.52055  | 11.52055     | -               | -                                                                      |
+-----------------------+-----------+-----------+--------------+-----------------+-----------------------------------------------------------------------+
14 rows in set (0.03 sec)
```

7. 子查询的**trace**结果举例

```
mysql> select count(*) from sharding_two_node where id =(select id from sharding_two_node where id=1);
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.03 sec)

mysql> show trace;
+-----------------------+-----------+-----------+--------------+--------------------+-----------------------
```

2.22 KILL @@DDL_LOCK

背景

在运维dble集群的过程中，有时会遇到 " There is other session is doing DDL " 或者 "xxx is doing DDL " 等问题，这些问题会导致某些表无法被操作，无法进行 reload 等。此时，需要人工干预保证集群运行正常。

逻辑判断

在 dble 集群中，可以通过 ZK 中某些键值对推断正在执行 DDL 的节点状态，比如通过键 universe/dble/{cluster-id}/ddl/{schema.table} 的值内容推断。该值理论上只会被发起者节点所修改，其他节点会订阅该键值，该值的内容为json格式，其中有一个 status 字段：

1. status 为 INIT 时，发起动作的节点会锁住本节点对应 table 的 table meta并获取当前key的分布式锁。其他的dble节点订阅到该值后会锁住本节点上对应 table 的 table meta；
2. 待发起节点执行DDL之后，会根据 DDL 执行结果更新当前键的值，此时根据 DDL 执行结果 status 分为两种情形：SUCCESS 和 FAILED。其他节点订阅到 status 为 SUCCESS 时，才会真正执行对应的DDL操作，否则会取消执行DDL并释放本节点对应table 的 table meta 锁；
3. 每个节点都会插入一条 universe/dble/{cluster-id}/ddl/{schema.table}/{dble-id}:SUCCESS 数据。

到此，除发起者的节点执行已经结束，但是发起者节点会等待其他节点响应，只有全部节点都汇报执行完成之后，当前ddl才算完成。

发起者节点通过什么来判断其他节点都响应完成？根据以下两个键的内容：

- universe/dble/{cluster-id}/ddl/{schema.table}/{dble-id}
- universe/dble/{cluster-id}/online/下属结点

所有执行当前DDL操作的节点都会在universe/dble/{cluster-id}/ddl/{schema.table}/ 下面插入记录，发起节点通过判断 online 节点是否在 universe/dble/{cluster-id}/ddl/{schema.table}/ 下面留下记录判断 DDL 是否被该节点所应用。只有两者相符合时，ddl才算真正意义上的完成。

此时，发起者会释放 table meta 锁 和 分布式锁，并且删除 `universe/dble/{cluster-id}/ddl/{schema.table}` kv 树。

kill ddl_lock

当前指令只会释放对应ddl在当前节点中所持有的元数据锁，但是不会影响执行该ddl的线程的状态。 若出现上述问题，可以参考 上面如何判断其他节点都响应完成的方式找出哪些没有响应，在这些节点和发起节点上执行当前命令，并手动删除universe/dble/{cluster-id}/ddl/{schema.table} kv 树。

注意事项

1. 需要注意kill指令的执行顺序：先在各个从节点上执行该指令，最后再在主节点上执行。
2. 如果不是特殊因素，此命令请不要随意使用。

2.23 外部高可用联动

- 2.23.1 外部后端MYSQL-HA连接
- 2.23.2 命令的使用说明
- 2.23.3 命令的实现细节
- 2.23.4 简单的HA交互使用案例

2.23.1 外部后端MYSQL-HA连接

dble在版本2.19.09.0开始提供对应的外部接口，基于以下几个基础要点进行设计：

- 外部HA组件能够通过一定的方式和dble进行信息的主动沟通（包括脚本，网络通信，命令等方式）
- 外部HA组件能够做出对与MYSQL可用性的明确判断（包括哪些节点停流量，哪些节点主从切换，哪些节点启动流量）

在基于以上的两个条件的情况下，基于提供外部接口通用性以及易用性的考虑，决定通过管理端用户命令的方式添加对应的交互接口，具体的提供以下的三个交互接口：

- 停止流量命令(dbGroup @@disable)
- 启动流量命令(dbGroup @@enable)
- 主从切换命令(dbGroup @@swtich)

其次为了方便进行管理和观测，提供了一个当前切换状态的查询命令:

- 切换状态查询命令(dbGroup @@events)

外部HA组件可以通过 mysql命令、脚本调用mysql命令、或者程序的DB连接执行mysql命令的方式进行交互。
同时此功能设计考虑到zk集群用户的需求，当用户使用zk集群状态下的dble进行高可用切换时，会有集群的联动机制，外部HA组件只需要通知其中一个节点即可。

以下提供一个外部HA和dble交互设计图：

更加详细的情况请查询以下文档:
命令的使用说明
命令的实现细节
简单的HA交互使用案例

2.23.2 dbGroup命令的具体使用方法和解释

外部ha启用参数

当项目准备使用dble的外部ha联动方式来完成高可用切换或者同步的时候，需要提前进行如下配置

bootstrap.cnf 中的对应参数

```
-DuseOuterHa=true
```

注意:本参数的调整需要重启*dble*服务

cluster.cnf 的对应参数

needSyncHa = true

- 当此参数启用时，集群状态的dble将会在集群中同步自身的dbGroup的状态
- 此配置会强制将bootstrap.cnf 中的useOuterHa设置为true
- 当bootstrap.cnf 中的useOuterHa但cluster.cnf的needSyncHa不为true时，dble可以执行高可用切换的所有指令，但是其行为退化为单机dble，需要人工进行集群中多个dble的状态同步

注意：此参数的调整需要重启集群内的所有*dble*服务

关于dble中对于后端MySQL状态的解释

dbInstance的状态“disabled/enable”仅表示dble层面对于具体每个MySQL后端节点是否允许有流量的标识，和具体的MySQL存活状态无关

dbGroup @@disable

命令细节：
dbGroup @@disable name = 'dbGroup_name' [instance = 'instance_name']

- 其中的dbGroup_name指的是在db.xml中配置的dbGroup的名称，而instance_name指的是在一个dbInstance的name
- 当此命令不指定instance = '..'的内容时，默认将此dbGroup下所有dbInstance的状态置为disabled
- 被标记为disabled的节点无法提供正常的查询，即使对应的mysql真实的存活着
- 具体的dbInstance的disable状态可以通过管理端命令show @@dbInstance进行查询
- 若当前dble服务尚存连接被disable的连接，在命令执行过程中会被全部关闭，包括正在新建过程中的连接，可能会导致少量的查询报错

dbGroup @@enable

命令细节：
dbGroup @@enable name = 'dbGroup_name' [instance = 'instance_name']

- 其中的dbGroup_name指的是在db.xml中配置的dbGroup的名称，而instance_name指的是在一个dbInstance的name
- 当此命令不指定instance = '..'的内容时，默认将此dbGroup下所有dbInstance的状态置为enable

dbGroup @@switch

命令细节： dbGroup @@switch name = 'dbGroup_name' master = 'instance_name'

- 其中的dbGroup_name指的是在db.xml中配置的dbGroup的名称，而instance_name指的是在一个dbInstance的name
- 此命令name和master内容都为必填，在缺少任意元素的状态下会报错
- 此命令的作用会导致dbGroup下的标记为primary的dbInstance发生重置，也就是写节点会发生变化
- 此命令不会导致所有dbInstance节点的disable状态变化，但如果命令使得一个dbInstance从primary退化成为非primary，此dbInstance上的所有既有连接都会被关闭，以确保新的写请求不会被写入到错误的dbInstance上面去，这可能导致一些前端连接的查询报错和事务失败

2.23.3 高可用联动命令的逻辑细节

简述

在dble高可用协同的几个接口中基本逻辑都是相似的，只是去更新dbGroup的属性，只不过在细节上每个命令都有一些自身的特殊行为

- disable命令在执行过程中会断开当前所有连接，并且在zk集群状态下会要求其他节点同步响应
- enable命令没有任何附加行为
- switch命令会在切换过程中断开旧primary dbInstance的所有连接

dbGroup @@disable

dble在单机情况下大致逻辑如下：

- 检查命令参数是否符合条件
- 更新dbInstance的状态
- 关闭所有已经存在的连接
- 同步更新配置文件，将dbGroup的最新状态落盘
- 返回OK信息

在集群状态下的逻辑有所不同：

- 检查命令参数是否符合条件
- 更新dbInstance的状态
- 关闭所有已经存在的连接
- 异步更新配置文件，将dbGroup的最新状态落盘
- 将最新的dbGroup的状态信息同步到zk上
- 写出一个zk上面的key任务，等待其他节点响应
- 其他节点响应任务，更新自身写出响应结果

- 发起dble检查所有响应结果，等待所有节点响应完成
- 返回最终的执行结果

dbGroup @@enable

dble在单机情况下的逻辑大致如下：

- 检查命令参数是否符合条件
- 更新dbInstance的状态
- 同步更新配置文件，将dbGroup的最新状态落盘
- 返回OK信息

在ZK集群的状况下逻辑大致如下：

- 检查命令参数是否符合条件
- 更新ddbInstance的状态
- 关闭所有已经存在的连接
- 异步更新配置文件，将dbGroup的最新状态落盘
- 将最新的dbGroup的状态信息同步到zk上
- 写出一个zk上面的key任务，等待其他节点响应
- 其他节点响应任务，更新自身写出响应结果
- 发起dble检查所有响应结果，等待所有节点响应完成
- 返回最终的执行结果

dbGroup @@switch

dble在单机情况下的其逻辑大致如下：

- 检查命令参数是否符合条件
- 更新dbInstance的状态
- 关闭旧primary dbInstance的所有连接
- 同步更新配置文件，将dbGroup的最新状态落盘
- 返回OK信息

在ZK集群的状况下逻辑大致如下：

- 检查命令参数是否符合条件
- 更新dbInstance的状态
- 关闭所有已经存在的连接
- 异步更新配置文件，将dbGroup的最新状态落盘
- 将最新的dbGroup的状态信息同步到zk上
- 写出一个zk上面的key任务，等待其他节点响应
- 其他节点响应任务，更新自身写出响应结果
- 发起dble检查所有响应结果，等待所有节点响应完成
- 返回最终的执行结果

开始收到命令
检查参数
合法性
是
否
获取不到锁
获取全局dbGroup锁
获取锁
zk
变更dbInstance标志
否
完成
删除旧主dbInstance
所有旧有连接
将最新的状态同步到
zk，并对于文件进行
异步落盘
同步最新状态
报错，给出提
示信息
dble-master
返回客户端信
息
订阅更新dbGroup最
新状态
根据最新的dbGroup
状态更新自身信息

2.23.4 mha-dble高可用联动实例

基础介绍

服务结构图

整体的服务结构如下图所示，dble将一个一组两从的MySQL视作一个读写分离的dbGroup，然后通过MHA对于此MySQL进行复制的管理：

下图是对于各个服务在每个docker容器上面的分布图：

实验过程

- 搭建一个MySQL主从(一主两从)的基础环境
- 搭建MHA监管MySQL复制组的环境
- 搭建dble环境使用这MySQL组进行启动
- 手动kill MySQL主实例让MHA托管的高可用组发生切换
- 验证dble和MySQL的高可用切换过程

实验目的

本实验原则上存在两个目的：

- 验证dble在2.19.09.0版本提供的高可用接口功能
- 搭建简单的MHA和dble高可用的环境范例

前期环境准备

前期准备环境：
三个docker容器，安装并部署mysql实例，并让三个mysql之间形成一主两从的复制关系(操作略)

操作过程

mha环境搭建

- 在每个docker容器中开启ssh服务

```
/usr/sbin/sshd -D
```

- 给每个容器之间创建免密登录,去到所有容器中执行，包括需要给自身容器创建

```
ssh-keygen -t rsa
ssh-copy-id -i ~/.ssh/id_rsa.pub root@other1
ssh-copy-id -i ~/.ssh/id_rsa.pub root@other2
ssh-copy-id -i ~/.ssh/id_rsa.pub root@self
```

- 确保/usr/bin/mysqlbinlog和/usr/bin/mysql文件存在，若不存在，请使用yum安装mysql即可
- 创建所有节点上的MHA工作目录

```
mkdir /etc/masterha/app1 -p
mkdir /var/log/masterha/app2 -p
mkdir /var/log/masterha/app1 -p
```

- 给所有mysql节点统一复制权限

```
grant all on *.* to root@'%' identified by '123456' with grant option;
grant replication slave on *.* to repl@'%' identified by 'repl';
```

- 下载并安装MHA的rpm包

```
从项目的文档页面能够直接找到对应的下载地址
https://github.com/yoshinorim/mha4mysql-manager/wiki/Downloads
下载一个MHA Manager 0.56 rpm RHEL6
以及一个MHA Node 0.56 rpm RHEL6
并将下载完成的rpm包上传到对应的容器中
注意使用yum localinstall 命令对于rpm包进行安装，这样在安装过程中可以由yum来进行依赖的处理
Node的包需要先于Manager的包安装，不然会有依赖方面的报错
在所有容器上安装完成这两个rpm
当yum无法正确安装rpm依赖，请更新yum源到阿里云的yum
```

切换脚本准备

mha这个项目是使用perl写的，同时mha这个项目给用户提供了大量的自定义插入接口，可以允许用户在一些关键步骤进行自定义的操作响应以及通知，在自动高可用切换过程中，mha提供了一个切换脚本的入口master_ip_failover_script脚本配置
详细的参数介绍可以参考官网的解释https://github.com/yoshinorim/mha4mysql-manager/wiki/Parameters#master_ip_failover_script
大体上来说这个脚本的三种命令会在以下几个时间节点得到调用

- 启动检查HA的状态，调用脚本master_ip_failover的status命令
- 发现MySQL master节点失效，调用master_ip_failover的stop命令，并输入失效master的信息
- 在新选择的master节点补偿数据完成，并进行写恢复设置read_only=0，调用master_ip_failover脚本的start命令

在本次实验中，通过自定义的脚本行为来进行mha和dble之间的事件交互，所以我们对于原有的样例脚本进行以下的修改

- 原有MySQL master节点失效时，将dbGroup上面对应写节点的状态修改成disable，这要求在stop命令中对于dble发送dbGroup @@disable命令
- 当新的MySQL master被选出来并且上线时，在脚本start阶段通过调用dbGroup @@switch命令将新的master节点切换成为dbGroup中的主节点

按照以上的逻辑对于修改完成的master_ip_failover脚本如下(注意给创建的脚步提供执行权限)

```
#!/usr/bin/env perl

#  Copyright (C) 2011 DeNA Co.,Ltd.
#
#  This program is free software; you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
#  the Free Software Foundation; either version 2 of the License, or
#  (at your option) any later version.
#
#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public License
#   along with this program; if not, write to the Free Software
#  Foundation, Inc.,
#  51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA

## Note: This is a sample script and is not complete. Modify the script based on your environment.

use strict;
use warnings FATAL => 'all';

use Getopt::Long;
use MHA::DBHelper;

my (
  $command,        $ssh_user,         $orig_master_host,
  $orig_master_ip, $orig_master_port, $new_master_host,
  $new_master_ip,  $new_master_port,  $new_master_user,
  $new_master_password
);
GetOptions(
  'command=s'             => \$command,
  'ssh_user=s'            => \$ssh_user,
```

```
  'orig_master_host=s'    => \$orig_master_host,
  'orig_master_ip=s'      => \$orig_master_ip,
  'orig_master_port=i'    => \$orig_master_port,
  'new_master_host=s'     => \$new_master_host,
  'new_master_ip=s'       => \$new_master_ip,
  'new_master_port=i'     => \$new_master_port,
  'new_master_user=s'     => \$new_master_user,
  'new_master_password=s' => \$new_master_password,
);

exit &main();

sub main {
  if ( $command eq "stop" || $command eq "stopssh" ) {

    # $orig_master_host, $orig_master_ip, $orig_master_port are passed.
    # If you manage master ip address at global catalog database,
    # invalidate orig_master_ip here.
    my $exit_code = 1;
    eval {

      # dbGroup @@disable name = "dbGroup1" instance='$orig_master_host'
      # 调用对应的disable命令，使得部分节点不可写
      $orig_master_host =~tr/./_/;
      system "mysql -P9066 -u man1 -p654321 -h 172.20.134.1 -e \"dbGroup \@\@disable name = 'dbGroup1' instance='".$orig_master_host."'\"";
      $exit_code = 0;
    };
    if ($@) {
      warn "Got Error: $@\n";
      exit $exit_code;
    }
    exit $exit_code;
  }
  elsif ( $command eq "start" ) {

    # all arguments are passed.
    # If you manage master ip address at global catalog database,
    # activate new_master_ip here.
    # You can also grant write access (create user, set read_only=0, etc) here.
    my $exit_code = 10;
    eval {
      my $new_master_handler = new MHA::DBHelper();

      # args: hostname, port, user, password, raise_error_or_not
      $new_master_handler->connect( $new_master_ip, $new_master_port,
        $new_master_user, $new_master_password, 1 );

      ## Set read_only=0 on the new master
      $new_master_handler->disable_log_bin_local();
      print "Set read_only=0 on the new master.\n";
      $new_master_handler->disable_read_only();

      ## Creating an app user on the new master
      print "Creating app user on the new master..\n";
      $new_master_handler->enable_log_bin_local();
      $new_master_handler->disconnect();

      ## try to switch the dbGroup master into new master
      ## 调用dbGroup switch的命令，将新的new_master_host节点提升
      $new_master_host =~tr/./_/;
      system "mysql -P9066 -u man1 -p654321 -h 172.20.134.1 -e \"dbGroup \@\@switch name = 'dbGroup1' master='".$new_master_host."'\"";

      $exit_code = 0;
    };
    if ($@) {
      warn $@;

      # If you want to continue failover, exit 10.
      exit $exit_code;
    }
    exit $exit_code;
  }
  elsif ( $command eq "status" ) {
    # test for start command
    exit 0;
  }
  else {
    &usage();
    exit 1;
  }
}

sub usage {
  print
"Usage: master_ip_failover --command=start|stop|stopssh|status --orig_master_host=host --orig_master_ip=ip --orig_master_port=port --new_master_host=host --new_master_ip=ip --new_master_port=por
t\n";
}
```

将脚本存放到指定目录/etc/masterha/app1目录下

并且创建MHA最终使用的 app1.conf配置文件在/etc/masterha/app1目录，具体创建的配置文件内容如下所示：

```
#mha manager工作目录
manager_workdir = /var/log/masterha/app1
manager_log = /var/log/masterha/app1/app1.log
remote_workdir = /var/log/masterha/app2
master_ip_failover_script=/etc/masterha/app1/master_ip_failover
# master_ip_online_change_script=/etc/masterha/app1/master_ip_online_change
# MySQL管理帐号和密码
user=root
password=123456
# 系统ssh用户
ssh_user=root

# 复制帐号和密码
repl_user=repl
repl_password= repl

# 监控间隔(秒)
ping_interval=1
manager_log=/var/log/masterha/app1/manager.log

[server1]
hostname=172.20.134.1
master_binlog_dir = /opt/3306/
port=3306


[server2]
# 每个机器上面的mysql实例的信息
hostname=172.20.134.2
master_binlog_dir = /opt/3306/
candidate_master=1
check_repl_delay=0
port=3306

[server3]
# 每个机器上面的mysql实例的信息
hostname=172.20.134.3
master_binlog_dir = /opt/3306/
candidate_master=1
check_repl_delay=0
port=3306
```

最终通过以下命令对于MHA的监听线程进行启动

```
nohup masterha_manager --conf=/etc/masterha/app1/app1.conf >> /var/log/masterha/app1/manager.log 2>&1 &
```

dble配置

配置dble在这里是比较简单的内容，我们仅使用默认的配置即可，从项目的release页面下载最新的2.19.09.0安装包，解压并将conf文件下的文件配置好

```
mv cluster_template.cnf cluster.cnf
mv bootstrap_template.cnf bootstrap.cnf
mv db_template.xml db.xml
mv user_template.xml user.xml
mv sharding_template.xml sharding.xml
```

并按照当前的配置需求对于其中的db.xml以进行如下的配置调整

db.xml

```
    <dbGroup name="dbGroup1"  delayThreshold="100">
        <heartbeat>show slave status</heartbeat>
        <dbInstance name="172_20_134_1" url="172.20.134.1:3306" password="123456" user="root" disabled="false" id="udp-1" primary="true" />
        <dbInstance name="172_20_134_2" url="172.20.134.2:3306" password="123456" user="root" disabled="false" id="udp-3"  />
        <dbInstance name="172_20_134_3" url="172.20.134.3:3306" password="123456" user="root" disabled="false" id="udp-2"  />
    </dbGroup>
```

启动dble，并通过管理用户man1查看基础状态下的dble后端节点状态

```
MySQL [(none)]> show @@dbinstance;
+-----------+--------------+--------------+------+------+--------+------+------+---------+-----------+------------+----------+
| DB_GROUP  | NAME         | HOST         | PORT | W/R  | ACTIVE | IDLE | SIZE | EXECUTE | READ_LOAD | WRITE_LOAD | DISABLED |
+-----------+--------------+--------------+------+------+--------+------+------+---------+-----------+------------+----------+
| dbGroup1  | 172_20_134_1 | 172.20.134.1 | 3306 | W    |      1 |    0 | 1000 |       1 |         0 |          0 | false    |
| dbGroup1  | 172_20_134_3 | 172.20.134.3 | 3306 | R    |      1 |    0 | 1000 |       0 |         0 |          0 | false    |
| dbGroup1  | 172_20_134_2 | 172.20.134.2 | 3306 | R    |      1 |    0 | 1000 |       0 |         0 |          0 | false    |
+-----------+--------------+--------------+------+------+--------+------+------+---------+-----------+------------+----------+
```

可见在初始状态下，节点的使用属性和配置文件一致，172.20.134.1节点作为写节点，并且所有节点的使用状态都是“可用”

最终效果

- 通过ps命令在172.20.134.1容器中找到对应的mysqld进程
- 通过kill -9 命令对于172.20.134.1容器中的mysqld进程进行关闭
- 重新通过命令检查dble中此时的后端节点状态，发现如下的执行结果

```
MySQL [(none)]> show @@dbinstance;
+-----------+--------------+--------------+------+------+--------+------+------+---------+-----------+------------+----------+
| DB_GROUP  | NAME         | HOST         | PORT | W/R  | ACTIVE | IDLE | SIZE | EXECUTE | READ_LOAD | WRITE_LOAD | DISABLED |
+-----------+--------------+--------------+------+------+--------+------+------+---------+-----------+------------+----------+
| dbGroup1  | 172_20_134_2 | 172.20.134.2 | 3306 | W    |      1 |    0 | 1000 |       0 |         0 |          0 | false    |
| dbGroup1  | 172_20_134_3 | 172.20.134.3 | 3306 | R    |      1 |    0 | 1000 |       0 |         0 |          0 | false    |
| dbGroup1  | 172_20_134_1 | 172.20.134.1 | 3306 | R    |      0 |    0 | 1000 |       0 |         0 |          0 | true     |
+-----------+--------------+--------------+------+------+--------+------+------+---------+-----------+------------+----------+
```

- 检查当前dble的配置文件，发现配置文件中的后端节点位置发生扭转,172_20_134_2变成新的写节点

```
<dbGroup name="dbGroup1"  delayThreshold="100">
        <heartbeat>show slave status</heartbeat>
        <dbInstance name="172_20_134_1" url="172.20.134.1:3306" password="123456" user="root" disabled="false" id="udp-1" />
        <dbInstance name="172_20_134_2" url="172.20.134.2:3306" password="123456" user="root" disabled="false" id="udp-3"  />
        <dbInstance name="172_20_134_3" url="172.20.134.3:3306" password="123456" user="root" disabled="false" id="udp-2" primary="true"  />
    </dbGroup>
```

- 检查存在于172.20.134.3上mysql从的状态，发现其复制指向已经发生切换，切换为新主172.20.134.2，与dble中最新的配置文件相符

```
MySQL [(none)]> show slave status\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 172.20.134.2
                  Master_User: repl
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000001
          Read_Master_Log_Pos: 2113
               Relay_Log_File: udp3-relay-bin.000002
                Relay_Log_Pos: 320
        Relay_Master_Log_File: mysql-bin.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
.........................
```

2.24 超时（连接/执行）控制

概述

在dble中连接分为两种，前后端连接，分别对应客户端连接dble的网络连接和dble中连接mysql的网络连接
大致的情况如下图所示：

由于网络TCP连接的特性，dble中需要对于这两种连接都有超时控制，具体在dble中将连接的超时控制分为空闲超时和执行超时两种情况。当超时发生的时候，dble会通过切断超时连接的方法进行控制。

超时的检查和实现

在配置文件bootstrap.cnf中存在三个配置项

- sqlExecuteTimeout(后端连接执行超时)
- idleTimeout（前端连接空闲超时）
- processorCheckPeriod（超时检查周期）

以下描述具体超时检查的实现逻辑

- dble按照配置的processorCheckPeriod作为周期去周期性检查所有的连接
- 检查所有后端连接，若满足以下条件就关闭后端连接
  - 此后端连接已经被某个前端连接借走（正在执行或持有）
  - 此后端连接距离上一次收发包超过sqlExecuteTimeout
  - 此后端连接没有在执行DDL或者执行xa事务
- 检查所有前端连接，若满足以下条件就关闭前端连接
  - 此前端连接正在执行xa事务，并不处于commit失败补偿/rollback失败补偿
  - 此前端连接距离上一次收发包超过idleTimeout

综上总结为超时关闭连接的逻辑可以描述为：

- 后端连接非DDL,XA事务的情况下被前端连接借走，并且超过sqlExecuteTimeout没有收发包（包括普通事务执行间隔，后端连接被长期持有的时间超时）
- 前端连接非XA事务特殊阶段，超过idleTimeout没有收发包（包括在loaddata大文件过程中发生的长时间空档）

SQL执行超时注意事项

- 部分语言或者框架的连接池会长期持有连接，并且没有设定或者定时发送网络包，可能会导致空闲的连接自动断开，然后应用在取用到对应断开连接的时候可能会报错
- 当开启一个事务但是长期不执行SQL会导致后端连接被判断定为执行超时，当应用再次在连接上执行内容的时候，会给出后端连接已关闭的报错信息
- 当前端连接执行一个大文件内容load data的时候，由于后端可能执行较慢，前端连接存在超过idleTimeout导致连接关闭的情况，当有类似数据导入计划的时候，考虑暂时放宽idleTimeout的限制
- 连接的超时检查和关闭是通过一个循环来实现的，所以并不是一个实时检查的值，最坏的情况对于超时的检查会迟到一个processorCheckPeriod周，当应用对于SQL超时需要一个严格的时间设定时，请谨慎使用

2.25 dble流量控制

背景

在之前的dble版本中，当进行大文件load以及大结果集查询的过程中，都有可能由于数据的发送不及时造成数据在dble内存中堆积，当条件足够的时候甚至有可能造成dble服务的OOM，进而影响服务的稳定运行
在本版2.20.04中由社区开发者@ssxlulu提供了对于这部分流量控制的实现，通过连接级别的写队列长度，进行数据加载/获取的负反馈调节，从而实现在数据load和大结果集查询过程中的内存使用情况稳定。 具体的issue详情请见

原理

dble中的流量控制通过连接级别的写出队列进行负反馈调节，具体在生效的时候分成两种具体的形式：

- load data过程中后端连接写出队列过长负反馈
- Query过程中前端连接写出队列过长负反馈 下图中分别描述了两种负反馈具体实现的逻辑：

注：流量控制功能的生效级别为连接级别，不同连接之间的队列长度不会互相影响

使用举例

本功能为默认关闭功能，需要在bootstrap.cnf中进行相关配置进行显式开启，或者通过管理端的辅助命令进行实时的调整。 在bootstrap.cnf中使用下列参数使得功能开启并生效：

```
enableFlowControl(是否开启流量控制开关)
flowControlStartThreshold(写队列上限阈值，写队列超限时开启流量控制)
flowControlStopThreshold(写队列下限阈值，写队列低于阈值时取消流量控制)
```

附带三个管理端命令可以修改当前的流量控制参数：

- flow_control @@show （展示当前流控配置参数信息）
- flow_control @@set [enableFlowControl = true/false] [flowControlStart = ?] [flowControlEnd = ?]（修改配置参数）
- flow_control @@list （展示当前正在被流量控制的连接）

2.26 开启/关闭client_found_rows权能标志

2.26.1 介绍&背景

在客户端与服务器在初次连接的时候，服务端发送初始化握手包时会带上自己所支持的权能标志；客户端接收后会对服务器发送的权能标志进行筛选，保留自身所支持权能标志且返回给服务器，从而保证服务器与客户端通讯的兼容性。因此权能标志是在初次连接确定，不能动态修改。
在DBLE中后端连接都是使用连接池预先建立好的，导致与前端请求不同导致行为不一样；因此在dble管理端中新增了对client_found_rows权能标志更改和后端连接池的刷新。

2.26.1.1 client_found_rows的作用

若初次连接handshake协议中启用client_found_rows权能标志，表示在DML等操作时结果集里返回发现行(found rows)，而不是影响行(affect rows)。

2.26.1.2 client_found_rows值设定：

MYSQL客户端：默认关闭client_found_row；（则返回结果集里为affect rows)
JDBC：默认开启client_found_rows；（则返回结果集里为found rows）

2.26.1.3 JDBC中useAffectedRows与client_found_row的关系

useAffectedRows=true 即关闭client_found_rows
useAffectedRows=false(默认) 即开启client_found_rows

2.26.2 具体

2.26.2.1 在bootstrap.cnf里增加参数

```
#是否开启client_found_rows权能标志，默认关闭
-DcapClientFoundRows=false
```

2.26.2.2 管理端口增加命令

```
show @@cap_client_found_rows; -- 查询client_found_row权能标志开启状态 0-关闭 1-开启
disable @@cap_client_found_rows; -- 关闭client_found_row权能标志
enable @@cap_client_found_rows; -- 开启client_found_row权能标志
```

注意： 如果在不停dble服务的情况下，更改该权能标志，为了保证与后端连接的mysql的该权能标志一致，(强调)需要刷新连接池；否则insert的结果集不正确

2.26.3 验证

step1. 启动dble(默认关闭client_found_rows)

step2. 管理用户身份开启client_found_rows权能标志

```
mysql -uman1 -h192.xx.xx.xx -P9066 -p654321
enable @@cap_client_found_rows;
```

step3. 用户身份尝试登录，提示客户端与dble内存中的client_found_rows权能标志不一致

```
mysql -uroot -h192.xx.xx.xx -P8066 -p123456
ERROR 1045 (HY000): The client requested CLIENT_FOUND_ROWS capabilities does not match, in the manager use show @@cap_client_found_rows check latest status.
```

3.语法兼容

- 3.1 DDL
  - 3.1.1 DDL&Table Syntax
  - 3.1.2 DDL&View Syntax
  - 3.1.3 DDL&Index Syntax
  - 3.1.4 DDL透传
  - 3.1.5 DDL&Database Syntax
  - 3.1.6 ONLINE DDL
- 3.2 DML
  - 3.2.1 INSERT
  - 3.2.2 REPLACE
  - 3.2.3 DELETE
  - 3.2.4 UPDATE
  - 3.2.5 SELECT
  - 3.2.6 SELECT JOIN syntax
  - 3.2.7 SELECT UNION Syntax
  - 3.2.8 SELECT Subquery Syntax
  - 3.2.9 LOAD DATA
  - 3.2.10 不支持的语句
- 3.3 Prepared SQL Syntax
- 3.4 Transactional and Locking Statements
  - 3.4.1 Lock&unlock
  - 3.4.2 XA 事务语法
  - 3.4.3 一般事务语法
  - 3.4.4 SET TRANSACTION Syntax
- 3.5 DAL
  - 3.5.1 SET
  - 3.5.2 SHOW
  - 3.5.3 KILL
- 3.6 存储过程支持方式
- 3.7 Utility Statements
- 3.8 Hint
- 3.9 其他不支持语句
- 3.10 函数与操作符支持列表(alpha版本)
- 3.11 导入导出方式

3.1 DDL

DDL 包含以下几部分内容

注意，当DDL执行时，涉及到的相同的表上的DML和DDL操作将会报错。

- 3.1.1 DDL&Table Syntax
- 3.1.2 DDL&View Syntax
- 3.1.3 DDL&Index Syntax
- 3.1.4 DDL透传

3.1.1 TABLE DDL

3.1.1.1 CREATE TABLE Syntax

```
CREATE TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
    [partition_options]

create_definition:
    col_name column_definition

column_definition:
    data_type [NOT NULL | NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
      [COMMENT 'string']

data_type:
    BIT[(length)]
  | TINYINT[(length)] [UNSIGNED] [ZEROFILL]
  | SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  | MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  | INT[(length)] [UNSIGNED] [ZEROFILL]
  | INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  | BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]
  | NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]
  | DATE
  | TIME[(fsp)]
  | TIMESTAMP[(fsp)]
  | DATETIME[(fsp)]
  | YEAR
  | CHAR[(length)]
  | VARCHAR(length)
  | BINARY[(length)]
  | VARBINARY(length)
  | TINYBLOB
  | BLOB
  | MEDIUMBLOB
  | LONGBLOB
  | TINYTEXT
  | TEXT
  | MEDIUMTEXT
  | LONGTEXT
  | ENUM(value1,value2,value3,...)

table_options:
    table_option [[,] table_option] ...

table_option:
    ENGINE [=] engine_name
  | [DEFAULT] CHARACTER SET [=] charset_name
  | CHECKSUM [=] {0 | 1}
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'string'
  | CONNECTION [=] 'connect_string'
  | KEY_BLOCK_SIZE [=] value
  | MAX_ROWS [=] value
  | MIN_ROWS [=] value
  | PASSWORD [=] 'string'
  | ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
  | STATS_AUTO_RECALC [=] {DEFAULT|0|1}
  | STATS_PERSISTENT [=] {DEFAULT|0|1}
partition_options:
      {[LINEAR] HASH(expr)
       | PARTITION BY [linear] KEY (column_list)
       | RANGE{(expr) | COLUMNS(column_list)}
       | LIST{(expr) | COLUMNS(column_list)}
       }
    [(partition_definition [, partition_definition] ...)]
```

注意：

- engine_name仅能为忽略大小写的“InnoDB“
- 不建议含有枚举类型的表作为分片表，比如表结构：CREATE TABLE `test` (`id` enum('1','2','3') DEFAULT '1')。因为此种表在插入id列时，既可以使用枚举值插入，也可以使用枚举值的下标，'1'的下标是1，以此类推。若用户以枚举值进行分片，但是插入时确使用枚举值下标，因为dble不会将下标转换为枚举值，所以分片会出现问题，详细可参考issue：https://github.com/actiontech/dble/issues/816。

例：

```
create table if not exists test(
id bigint primary key AUTO_INCREMENT,
col1 int not null default 5,
col2 int null COMMENT 'info for col1',
col3 varchar(20) not null,
col4 varchar(20) unique key
);

create table test(
id int primary key,
col_bit    BIT(1),
col_tinyint TINYINT(2) UNSIGNED ZEROFILL,
col_smallint SMALLINT(3) UNSIGNED ZEROFILL,
col_mediumint MEDIUMINT(4) UNSIGNED ZEROFILL,
col_int INT(5) UNSIGNED ZEROFILL,
col_integer INTEGER(6) UNSIGNED ZEROFILL,
col_bigint BIGINT(7) UNSIGNED ZEROFILL,
col_real REAL(8,1) UNSIGNED ZEROFILL,
col_double DOUBLE(9,2) UNSIGNED ZEROFILL,
col_float FLOAT(10,3) UNSIGNED ZEROFILL,
col_decimal DECIMAL(11,4) UNSIGNED ZEROFILL,
col_numeric NUMERIC(12,5) UNSIGNED ZEROFILL,
col_date DATE,
col_time TIME(3),
col_timestamp TIMESTAMP(4),
col_datetime DATETIME(5),
col_year YEAR,
col_char CHAR(10) ,
col_varcgar VARCHAR(20) ,
col_binary BINARY(30),
col_varbinary VARBINARY(40),
col_tinyblob TINYBLOB,
col_blob BLOB,
col_mediumblob MEDIUMBLOB,
col_longblob LONGBLOB,
col_tinytext TINYTEXT ,
col_text TEXT ,
col_mediumtext MEDIUMTEXT ,
col_longtext LONGTEXT ,
col_enum ENUM('a','b','c')
);
```

或者

```
create table test(
id int primary key,
col1 varchar(20)
)ENGINE = innodb
 AVG_ROW_LENGTH = 20
 DEFAULT CHARACTER SET = utf8
 CHECKSUM = 1
 DEFAULT COLLATE = utf8_general_ci
 COMMENT = 'info of table test'
 CONNECTION = '111111'
 DELAY_KEY_WRITE = 1
 INSERT_METHOD = LAST
 KEY_BLOCK_SIZE = 65536
 MAX_ROWS = 3
 MIN_ROWS = 2
 PACK_KEYS = 1
 ROW_FORMAT = DEFAULT;
```

3.1.1.2 ALTER TABLE Syntax

```
ALTER [IGNORE] TABLE tbl_name
    [alter_specification [, alter_specification] ...]

alter_specification:
  | ADD [COLUMN] col_name column_definition
        [FIRST | AFTER col_name ]
  | ADD [COLUMN] (col_name column_definition,...)
  | ADD {INDEX | KEY} [index_name]
  | CHANGE [COLUMN] old_col_name new_col_name column_definition
        [FIRST|AFTER col_name]
  | MODIFY [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | DROP [COLUMN] col_name
  | DROP {INDEX | KEY} index_name
  | ADD [INDEX|KEY] [index_name] (index_col_name,...)
  | DROP {INDEX|KEY} index_name
  | ADD PRIMARY KEY (index_col_name,...)
  | DROP PRIMARY KEY
  | ALTER [COLUMN] col_name
        {SET DEFAULT {literal | (expr)} | DROP DEFAULT}
  | COMMENT [=] 'string'
```

例:

```
alter table test add column col5 int not null default 1 first,add column col6 int after col4;
alter table test change column col1 col1_new int after col3;
alter table test modify column col1_new varchar(20) after id;
alter table test drop column col6;
alter table test add key idx_col4(col4);
alter table test add index idx_col4(col4);
alter table test drop key idx_col4;
alter table test drop index idx_col4;
alter table test drop primary key;
alter table test add primary key (id);
alter table test alter column col set default 0;
alter table test alter column col drop default;
alter table test comment = 'string';
```

3.1.1.3 DROP TABLE Syntax

```
DROP TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
```

例:

```
drop table if exists test cascade;
drop table test restrict;
```

3.1.1.4 TRUNCATE TABLE Syntax

```
TRUNCATE [TABLE] tbl_name
```

例:

```
truncate table test;
```

3.1.2 VIEW DDL

Syntax

create view :

```
CREATE [OR REPLACE] VIEW
        view_name [(column_list)]
        AS select_statement
```

alter view :

```
alter  VIEW
        view_name [(column_list)]
        AS select_statement
```

drop view:

```
DROP VIEW [IF EXISTS] view_name [, view_name]
```

show create view :

```
SHOW CREATE VIEW view_name;
```

3.1.3 INDEX DDL

3.1.3.1 CREATE INDEX Syntax

```
CREATE [UNIQUE|FULLTEXT] INDEX index_name
    [index_type]
    ON tbl_name (index_col_name,...)

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}
```

例:

```
create unique index idx1 using btree on test(col1);
create index idx2 using hash on test(col2);
create fulltext index idx3 on test(col4);
create fulltext index idx4 on test(col4(10));
```

3.1.3.2 DROP INDEX Syntax

```
DROP INDEX index_name ON tbl_name
```

例:

```
drop index idx1 on test;
```

3.1.4 DDL透传

除了前面章节中的DDL语句，dble在非注解方式方式下不支持mysql中的其它DDL语句(eg. ALTER EVENT)。
但通过如下的注解方式，dble支持所有的mysql中的DDL语句：

```
/*!dble:sql=select ... from tbx where id=M*/ ddl statement;
```

其中， tbx为分区表， id为分区列，M为分区列的某个值。

例如：

```
MySQL [TESTDB]> /*!dble:sql=select * from a_test where id=2*/CREATE PROCEDURE account_count()

      BEGIN   SELECT 'Number of accounts:', COUNT(*) FROM mysql.user;

END//
```

3.1.5 DATABASE DDL

3.1.5.1 CREATE DATABASE Syntax

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification] ...

create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
  | DEFAULT ENCRYPTION [=] {'Y' | 'N'}
```

注意：

- schema一定要在shema.xml中配置
- create_specification 中的选项不生效
- 只具有语法意义，没有实际语义的作用，需要建库请用管理端口对应的命令

3.1.6 ONLINE DDL

3.1.6.1 背景

1. 在3.20.04.0或之前版本的dble中执行ddl，需要加表锁，若是在集群中，也会通知其他dble加上表锁，加锁期间执行对应表的SQL会报错。在ddl执行结束后dble会下发show create table 得到建表语句，得到建表语句后会解析重新生成元数据，集群中的其他dble也会执行此操作。
2. 但一部分SQL其实对dble不造成影响，对dble造不造成影响的判断依据在于是否需要更改dble中表的元数据，目前dble中表的元数据只关心表的列名称，列类型，类是否为null。若这样的sql在mysql侧也是onlineDDL的，可以无需加锁，直接下发执行，例如增加索引的操作。

3.1.6.2 ONLINE DDL种类

以下穷举了MySQL 8.0中所有的online ddl以及是否在dble中支持online ddl模式。

- Index Operations
- Primary Key Operations
- Column Operations
- Generated Column Operations
- Foreign Key Operations
- Table Operations
- Tablespace Operations
- Partitioning Operations

序号	类别	操作	语句	直接下发?	备注
1	索引操作	创建或者新增二级索引	CREATE INDEX *name* ON *table* (*col_list*);ALTER TABLE *tbl_name* ADD INDEX *name* (*col_list*);	是	涉及到表的事务结束后才会执行该操作。操作期间，支持表的并发读写。
2		删除索引	DROP INDEX *name* ON *table*;ALTER TABLE *tbl_name* DROP INDEX *name*;	是	涉及到表的事务结束后才会执行该操作。操作期间，支持表的并发读写。
3		索引重命名	ALTER TABLE *tbl_name* RENAME INDEX *old_index_name* TO *new_index_name*, ALGORITHM=INPLACE, LOCK=NONE;	否	mysql 5.6 该操作不是online ddl
4		添加 FULLTEXT 索引	CREATE FULLTEXT INDEX *name* ON table(*column*);	否	有限制，具体参见mysql官方文档
5		添加 SPATIAL 索引	CREATE TABLE geom (g GEOMETRY NOT NULL); ALTER TABLE geom ADD SPATIAL INDEX(g), ALGORITHM=INPLACE, LOCK=SHARED;	否	有限制，具体参见mysql官方文档
6		修改索引类型	ALTER TABLE *tbl_name* DROP INDEX i1, ADD INDEX i1(*key_part,...*) USING BTREE, ALGORITHM=INSTANT;	是	
7	主键操作	新增主键	ALTER TABLE *tbl_name* ADD PRIMARY KEY (*column*), ALGORITHM=INPLACE, LOCK=NONE;	否	主键操作过程中都需要重建表数据，代价较昂贵
8		删除主键	ALTER TABLE *tbl_name* DROP PRIMARY KEY, ALGORITHM=COPY;	否	该操作只允许COPY操作，过程中不支持并发dml
9		修改主键	ALTER TABLE *tbl_name* DROP PRIMARY KEY, ADD PRIMARY KEY (*column*), ALGORITHM=INPLACE, LOCK=NONE;	否	主键的确定在初期就应该确认好，避免在后期修改
10	列操作	新增列	ALTER TABLE *tbl_name* ADD COLUMN *column_name column_definition*, ALGORITHM=INSTANT;	否	dble需要重新生成表元数据
11		删除列	ALTER TABLE *tbl_name* DROP COLUMN *column_name*, ALGORITHM=INPLACE, LOCK=NONE;	否	dble需要重新生成表元数据
12		重命名列	ALTER TABLE *tbl* CHANGE *old_col_name new_col_name data_type*, ALGORITHM=INPLACE, LOCK=NONE;	否	dble需要重新生成表元数据
13		列重排序	ALTER TABLE *tbl_name* MODIFY COLUMN *col_name column_definition* FIRST, ALGORITHM=INPLACE, LOCK=NONE;	否	

14		修改列数据类型	ALTER TABLE *tbl_name* CHANGE c1 c1 BIGINT, ALGORITHM=COPY;	否	执行期间，不允许dml
15		加长 VARCHAR 长度	ALTER TABLE *tbl_name* CHANGE COLUMN c1 c1 VARCHAR(255), ALGORITHM=INPLACE, LOCK=NONE;	否	涉及到dble的表元数据
16		给列设置默认值	ALTER TABLE *tbl_name* ALTER COLUMN *col* SET DEFAULT *literal*, ALGORITHM=INSTANT;	是	
17		删除列的默认值	ALTER TABLE *tbl* ALTER COLUMN *col* DROP DEFAULT, ALGORITHM=INSTANT;	是	
18		修改自增值	ALTER TABLE *table* AUTO_INCREMENT=*next_value*, ALGORITHM=INPLACE, LOCK=NONE;	否	若是分片节点，给每个节点设置成相同的自增值没有意义
19		列可以为NULL	ALTER TABLE tbl_name MODIFY COLUMN *column_name data_type* NULL, ALGORITHM=INPLACE, LOCK=NONE;	否	涉及到dble的表元数据
20		列不可以为NULL	ALTER TABLE *tbl_name* MODIFY COLUMN *column_name data_type* NOT NULL, ALGORITHM=INPLACE, LOCK=NONE;	否	涉及到dble的表元数据
21		修改 `ENUM` 或 `SET`列的定义	CREATE TABLE t1 (c1 ENUM('a', 'b', 'c')); ALTER TABLE t1 MODIFY COLUMN c1 ENUM('a', 'b', 'c', 'd'), ALGORITHM=INSTANT;	否	不常用
22	外键操作	新增外键	ALTER TABLE *tbl1* ADD CONSTRAINT *fk_name* FOREIGN KEY *index* (*col1*) REFERENCES *tbl2*(*col2*) *referential_actions*;	否	dble中的表会跨多实例，表与表之间建立外键无实际意义
23		删除外键	ALTER TABLE *tbl* DROP FOREIGN KEY *fk_name*;		
24	Generated列操作	新增 Generated 列	ALTER TABLE t1 ADD COLUMN (c2 INT GENERATED ALWAYS AS (c1 + 1) STORED), ALGORITHM=COPY;	否	相当于新增一列
25		修改 Generated 列顺序	ALTER TABLE t1 MODIFY COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) STORED FIRST, ALGORITHM=COPY;		
26		删除 Generated 列	ALTER TABLE t1 DROP COLUMN c2, ALGORITHM=INPLACE, LOCK=NONE;		
27		新增 VIRTUAL 列	ALTER TABLE t1 ADD COLUMN (c2 INT GENERATED ALWAYS AS (c1 + 1) VIRTUAL), ALGORITHM=INSTANT;		
28		修改 VIRTUAL 列顺序	ALTER TABLE t1 MODIFY COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) VIRTUAL FIRST, ALGORITHM=COPY;		
29		删除 VIRTUAL 列	ALTER TABLE t1 DROP COLUMN c2, ALGORITHM=INSTANT;		
30	表空间操作			否	更偏向MySQL运维测
31	表分区操作			否	dble 分库分表的功能不涉及表分区，不需要支持

3.1.6.3 限制

1. dble 使用阿里的druid作为解析器，因此有些online ddl语句的 “ALGORITHM=INPLACE, LOCK=NONE” 不被支持，详情参考：https://github.com/alibaba/druid/issues/3750

3.2 DML

DML 包含以下内容

- 3.2.1 INSERT
- 3.2.2 REPLACE
- 3.2.3 DELETE
- 3.2.4 UPDATE
- 3.2.5 SELECT
- 3.2.6 SELECT JOIN syntax
- 3.2.7 SELECT UNION Syntax
- 3.2.8 SELECT Subquery Syntax
- 3.2.9 LOAD DATA
- 3.2.10 不支持的语句

3.2.1 INSERT

3.2.1.1 Syntax

INSERT [INTO] tbl_name

[(col_name,...)]

{VALUES | VALUE} ({expr },...),(...),...

[ON DUPLICATE KEY UPDATE

col_name=expr

[, col_name=expr] ...]

OR

INSERT [INTO] tbl_name

SET col_name={expr | DEFAULT}, ...

[ON DUPLICATE KEY UPDATE

col_name=expr [, col_name=expr] ...]

3.2.1.2 举例

```
insert into test (col1,col3) values(1,'cust1'),(2,'cust2');
insert into test (col1,col3) values(default,'cust3');
insert into test set col1=4,col3='cust4';
insert into test set col1=default,col3='cust5';
insert into test (col1,col3) values(default,cast(now() as char));
```

3.2.1.3 限制

- 在插入ER关系的子表时，每个语句只允许插入一个ROW
- 全局序列在插入时不允许指定值，全部由dble序列生成
- 对于含有枚举类型的分片表，比如表结构：CREATE TABLE `test` (`id` enum('1','2','3') DEFAULT '1')，在插入id列时，既可以使用枚举值插入，也可以使用枚举值的下标，'1'的下标是1，以此类推。若用户以枚举值进行分片，但是插入时确使用枚举值下标，因为dble不会将下标转换为枚举值，所以分片会出现问题，详细可参考issue：https://github.com/actiontech/dble/issues/816。
- 存在特例，当insert/replace... select 语句满足以下条件时，dble会在确保数据安全性的情况下对于SQL进行下发执行
  - 当插入目标是单节点表时，要求所有数据来源的表格都有明确的路由信息路由到同一节点
  - 当插入目标是全局表时，要求所有的数据来源表格都是全局表，并且路由范围能够覆盖插入目标
  - 当插入目标是分片表时，要求分片列的数据直接来源自拥有同样分片逻辑的分片表，并且对于select子查询中间的其他表格，要求能够子查询部分路由结果能整体下发，并且逻辑上无错误

3.2.2 REPLACE

3.2.2.1 Syntax

REPLACE

[INTO] tbl_name [(col_name [, col_name] ...)]

{VALUES | VALUE} (value_list) [, (value_list)] ...

OR

REPLACE

[INTO] tbl_name SET assignment_list

3.2.2.2 举例

```
REPLACE INTO test VALUES (1, 'Old', '2014-08-20 18:47:00');
REPLACE INTO test set id = 1, type= 'Old',create_date =  '2014-08-20 18:47:00';
```

3.2.2.3 限制

- 由于replace的语义为如果存在则替换，如果不存在则新增，所以在使用表格自增主键的时候
- 如果对于自增表格使用replace且ID不存在，那么就会插入一条指定ID的数据，并不会自动生成ID
- 存在特例，当insert/replace语句满足以下条件时，dble会在确保数据安全性的情况下对于SQL进行下发执行
  - 当插入目标是单节点表时，要求所有数据来源的表格都有明确的路由信息路由到同一节点
  - 当插入目标是全局表时，要求所有的数据来源表格都是全局表，并且路由范围能够覆盖插入目标
  - 当插入目标是分片表时，要求分片列的数据直接来源自拥有同样分片逻辑的分片表，并且对于select子查询中间的其他表格，要求能够子查询部分路由结果能整体下发，并且逻辑上无错误

3.2.3 DELETE

3.2.3.1 Syntax

DELETE [IGNORE]

FROM tbl_name [WHERE where_condition]

3.2.3.2 举例

```
delete from test where id>5;
```

3.2.3.3 限制

- 原则上Delete语句中的where_condition部分只允许出现简单的条件，不能支持计算表达式以及子查询
- 原则上不支持多表Join 的DELETE
- 存在特例，当DELETE满足以下判断条件时，部分复杂DELETE语句会在确保数据正确的情况下被下发执行
  - 包括update/delete语句操作多表的时候，【操作的全是全局表，并拥有同样的分片范围，并且where条件中不含有子查询】或 【所有操作的表都有条件显式路由到同一个节点，并且where条件不含有子查询】
  - 包括update/delete语句操作单表，但是where条件中包含子查询时，【被操作的表格是单节点表，where条件中的所有表格都有条件能路由到同一个节点】或 【被操作的表格是全局表，其余所有表格也都是全局表，并且涉及范围都能覆盖被操作的表格】

3.2.4 UPDATE

3.2.4.1 Syntax

UPDATE **table_reference**

SET **col_name1**={**expr1**} [, **col_name2**={**expr2**}] ...

[WHERE **where_condition**]

3.2.4.2 举例

```
UPDATE  test SET VALUE =1 where id=5;
```

3.2.4.3 限制

- 原则上UPDATE语句中的where_condition部分只允许出现简单的条件，不能支持计算表达式以及子查询
- 原则上不支持多表Join 的UPDATE
- 存在特例，当update满足以下判断条件时，部分复杂DELETE语句会在确保数据正确的情况下被下发执行
  - 包括update/delete语句操作多表的时候，【操作的全是全局表，并拥有同样的分片范围，并且where条件中不含有子查询】或 【所有操作的表都有条件显式路由到同一个节点，并且where条件不含有子查询】
  - 包括update/delete语句操作单表，但是where条件中包含子查询时，【被操作的表格是单节点表，where条件中的所有表格都有条件能路由到同一个节点】或 【被操作的表格是全局表，其余所有表格也都是全局表，并且涉及范围都能覆盖被操作的表格】

3.2.5 SELECT

3.2.5.1 Syntax

SELECT

[ALL | DISTINCT | DISTINCTROW]

select_expr

[, select_expr ...]

[FROM table_references [WHERE where_condition]

[GROUP BY {col_name | expr | position} [ASC | DESC], ...]

[HAVING where_condition] [ORDER BY {col_name | expr | position} [ASC | DESC], ...]

[LIMIT {[offset,] row_count | row_count OFFSET offset}]

3.2.5.2 举例

```
select id,col1,col3 from test where id=3;
select distinct col1,col3 from test where id>=3;
select count(*),max(id),col1 from test group by col1 desc having(count(*)>1) order by col1 desc;
select id,col1,col3 from test order by id limit 2 offset 2;
select id,col1,col3 from test order by id limit 2,2;
select 1+1,'test',id,col1*1.1,now() from test limit 3;
select current_date,current_timestamp;
```

3.2.6 JOIN Syntax:

table_references:

```
table_reference [, table_reference] ...
```

table_reference:

```
table_factor | join_table
```

table_factor:

```
tbl_name [[AS] alias]
| table_subquery [AS] alias
| ( table_references )
```

join_table:

```
table_reference [INNER | CROSS] JOIN table_factor [join_condition]
| table_reference STRAIGHT_JOIN table_factor
| table_reference STRAIGHT_JOIN table_factor ON conditional_expr
| table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_condition
| table_reference NATURAL [{LEFT|RIGHT} [OUTER]] JOIN table_factor
```

join_condition:

```
ON conditional_expr
| USING (column_list)
```

3.2.7 UNION Syntax:

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
```

3.2.8 Subquery

3.2.8.1 The Subquery as Scalar Operand

For example :

```
SELECT (SELECT s2 FROM t1);
SELECT (SELECT s1 FROM t2) FROM t1;
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

3.2.8.2 Comparisons Using Subqueries

The most common use of a subquery is in the form:

```
non_subquery_operand comparison_operator (subquery)
```

Where comparison_operator is one of these operators:

```
=    >    <    >=    <=    <>    !=    <=>
```

MySQL also permits this construct:

```
non_subquery_operand LIKE (subquery)
```

3.2.8.3 Subqueries with ANY, IN, or SOME

Syntax:

```
operand comparison_operator ANY (subquery)
operand IN (subquery)
operand comparison_operator SOME (subquery)
```

Where comparison_operator is one of these operators:

```
=    >    <    >=    <=    <>    !=
```

3.2.8.4 Subqueries with ALL

Syntax:

```
operand comparison_operator ALL (subquery)
```

3.2.8.5 Subqueries with EXISTS or NOT EXISTS

For example:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

Not support Correlated Subqueries for now.

3.2.8.6 Derived Tables (Subqueries in the FROM Clause)

```
SELECT ... FROM (subquery) [AS] tbl_name ...
```

3.2.9 LOAD DATA

3.2.9.1 Syntax

LOAD DATA

[LOCAL]

INFILE 'file_name' INTO TABLE tbl_name

[CHARACTER SET charset_name]

[{FIELDS | COLUMNS}

[TERMINATED BY 'string']

[[OPTIONALLY] ENCLOSED BY 'char']

[ESCAPED BY 'char']]

[LINES [STARTING BY 'string']

[TERMINATED BY 'string']]

3.2.9.2 举例

```
load data infile 'data.txt' into table test_table CHARACTER SET 'utf8' FIELDS  TERMINATED by ',';
```

3.2.9.3 原理

dble解析MySQL协议之后，会根据数据路由拆分文件,每满足maxRowSizeToFile(可通过bootstrap.cnf配置)就写到文件中，再通过 load data local infile 的方式导入到后端结点。
所以，这里local_infile这个参数会影响到load data的正确性.
可参考#1085

3.2.9.4 限制

- 存在BUG导致在dble中CHARACTER SET charset_name必填
- 在mysql中如果插入的数据不符合规范会插入部分数据，相对来说dble的load data对正确性有更高的要求，一个错误的发生会导致整体操作的回滚
- 在ENCLOSED BY的时候存在BUG，使用的的时候会导致转义的数据无法被正确转义存储到数据库中
- 由于当前解析设定的限制，loaddata默认每列最大字节数65535，可通过bootstrap.cnf中的maxCharsPerColumn配置修改
- 使用load data导入数据时,若导入表是分片表,应保证导入文件中分片键数据符合分片规则的要求,否则dble将会报错.详细参考issue:https://github.com/actiontech/dble/issues/770
- load data语句需要严格按照语法书写,由于druid解析器的缘故,若语法或关键字错误,druid会解析出错误的语句,从而导致结果错误.详细请参照issue:https://github.com/actiontech/dble/issues/1248
- load data语句读字段的时候，如果遇到行结束符，会认为是本行结束了，需要注意:https://github.com/actiontech/dble/issues/1507
- load data中使用用户变量后，再次查询用户变量的值是不正确的，可参考issue：https://github.com/actiontech/dble/issues/1761

3.2.10 不支持的DML语句

DO Syntax

HANDLER Syntax

LOAD XML Syntax

3.3 PREPARE SQL Syntax

3.3.1 PREPARE Syntax

```
PREPARE stmt_name FROM preparable_stmt
```

例:

```
prepare stmt1 from "select * from a_test where id=?";
```

3.3.2 EXECUTE Syntax

```
EXECUTE stmt_name
[USING @var_name [, @var_name] ...]
```

例:

```
SET @a = 1;
EXECUTE stmt1 USING @a;
```

3.3.3 DEALLOCATE PREPARE Syntax

```
{DEALLOCATE | DROP} PREPARE stmt_name
```

例:

```
DROP PREPARE stmt1;
```

3.4 Transactional and Locking Statements

Transactional and Locking Statements 包含以下内容

- 3.4.1 Lock&unlock
- 3.4.2 XA 事务语法
- 3.4.3 一般事务语法
- 3.4.4 SET TRANSACTION Syntax

3.4.1 Lock&unlock

3.4.1.1 Syntax

LOCK TABLES tbl_name [[AS] alias] **lock_type**

lock_type: READ | WRITE

UNLOCK TABLES

3.4.1.2 举例

```
lock tables test_table read;
unlock tables;
```

3.4.1.3 限制

1. 当前session加锁后访问其他表可能不会被阻止或者报错。
2. 加写锁后，复杂查询可能不会返回正确结果。
3. 如果在xa事务中使用 lock table 语句，dble会将隐式提交并将xa事务关闭。

3.4.2 XA 事务语法

3.4.2.1 Syntax

开启**XA**

set xa = {0|1}

开启事务

START TRANSACTION;

BEGIN

SET autocommit = {0 | 1}

提交事务

COMMIT

回滚事务

ROLLBACK

3.4.2.2 限制

1. 在Dble中一旦开启了SQL黑名单检查，不能使用BEGIN开启事务(druid解析器不支持)。
2. 使用 lock table 语句后会将 xa 事务关闭。

3.4.3 一般事务语法

3.4.3.1 Syntax

开启事务

START TRANSACTION;

BEGIN

SET autocommit = {0 | 1}

提交事务

COMMIT

回滚事务

ROLLBACK

3.4.3.2 限制

- 在Dble中一旦开启了SQL黑名单检查，不能使用BEGIN开启事务(druid解析器不支持)
- 2PC实现的分布式事务(非xa方式)可能会出现commit时部分提交的情况,如需保障最终一致性，需要开启XA

3.4.4 SET TRANSACTION Syntax

SET SESSION TRANSACTION ISOLATION LEVEL level

level:
REPEATABLE READ
| READ COMMITTED
| READ UNCOMMITTED
| SERIALIZABLE

SET @@SESSION.TX_ISOLATION = 'level_str'

level_str: REPEATABLE-READ
| READ-COMMITTED
| READ-UNCOMMITTED
| SERIALIZABLE

注：因为隔离级别不加session关键字语义不同，暂不支持

3.4.5 SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax

3.4.5.1 Syntax

SAVEPOINT *identifier*

ROLLBACK [WORK] TO [SAVEPOINT] *identifier*

RELEASE SAVEPOINT *identifier*

3.4.5.1 举例

```
# start transaction
set autocommit = 0;

# savepoint
savepoint s0;
insert into test value(1);
savepoint s1;
insert into test value(2);
savepoint s2;
insert into test value(3);

# rollback to
rollback to s0

# release
release savepoint s0
```

3.4.5.2 限制

1. 在mysql中,在事务外也可以定义savepoint,但是这些savepoint是没有意义的.因此在dble中savepoint强制在事务中使用,否则报错.
2. ROLLBACK TO [SAVEPOINT] *identifier* 语句暂不支持 work 可选项.

3.5 DAL

DAL主要包含以下内容

- 3.5.1 SET
- 3.5.2 SHOW
- 3.5.3 KILL

3.5.1 SET语句

3.5.1.1 XA

```
set xa=value

value:
  0
 | off
 | false
 | 1
 | on
 | true
```

例：

```
set xa=1;
```

注意事项：XA设置不能在多变量设置语句中使用。

3.5.1.2 AUTOCOMMIT

```
set autocommit=value

value:
  0
 | off
 | false
 | 1
 | on
 | true
```

例：

```
set autocommit=1;
```

注意事项：AUTOCOMMIT设置不能在多变量设置语句中使用。

3.5.1.3 NAMES

```
SET NAMES {'charset_name' [COLLATE 'collation_name'] | DEFAULT}
```

例：

```
set names utf8;

set names utf8 collate utf8_general_ci;

set names default;
```

3.5.1.4 CHARSET

```
SET {CHARACTER SET | CHARSET}
{'charset_name' | DEFAULT}
```

例：

```
set CHARACTER SET utf8;
```

3.5.1.5 COLLATION_CONNECTION/CHAESET_SET_X

```
SET COLLATION_CONNECTION='collation_name'

SET CHARSET_SET_CLIENT='charset_name',

SET CHARSET_SET_RESULTS='charset_name'  其中，'charset_name' 可以为NULL。

SET CHARSET_SET_CONNECTION='charset_name'
```

例：

```
set collation_connection=utf8_general_ci;

set CHARSET_SET_CLIENT=utf8;

set CHARSET_SET_RESULTS=utf8;

set CHARSET_SET_CONNECTION=utf8;
```

3.5.1.6 TRANSACTION ACCESS MODE

```
SET SESSION { TX_READ_ONLY |  TRANSACTION_READ_ONLY}=value

value:
  0
 | off
 | false
 | 1
 | on
 | true
```

例:

```
set session @@tx_read_only=1;
```

3.5.1.7 TRANSACTION ISOLATION LEVEL

```
SET SESSION {TRANSACTION_ISOLATION | TX_ISOLATION}=level

level:

READ-UNCOMMITTED | READ-COMMITTED | REPEATABLE-READ | SERIALIZABLE
```

例:

```
SET SESSION TX_ISOLATION=READ-COMMITTED;
```

3.5.1.8 USER/SYSTEM VARIABLE

```
SET variable_assignment[, variable_assignment ] ...

variable_assignment:

  @user_var_name = expr

| SESSION system_var_name = expr

| system_var_name = expr

| @@system_var_name = expr

| @@session.system_var_name = expr
```

注意事项:

1. 不能设置全局系统变量。
2. 支持的系统变量为:

```
audit_log_current_session
audit_log_filter_id
auto_increment_increment
auto_increment_offset
autocommit
big_tables
binlog_direct_non_transactional_updates
binlog_error_action
binlog_format
binlog_row_image
binlog_rows_query_log_events
binlogging_impossible_mode
block_encryption_mode
bulk_insert_buffer_size
character_set_client
character_set_connection
character_set_database
character_set_filesystem
character_set_results
character_set_server
collation_connection
collation_database
collation_server
completion_type
debug
debug_sync
default_storage_engine
default_tmp_storage_engine
default_week_format
disconnect_on_expired_password
div_precision_increment
end_markers_in_json
eq_range_index_dive_limit
error_count
explicit_defaults_for_timestamp
external_user
foreign_key_checks
group_concat_max_len
gtid_next
gtid_owned
identity
innodb_create_intrinsic
innodb_ft_user_stopword_table
innodb_lock_wait_timeout
innodb_optimize_point_storage
innodb_strict_mode
innodb_support_xa
innodb_table_locks
innodb_tmpdir
insert_id
```

```
interactive_timeout
join_buffer_size
keep_files_on_create
last_insert_id
lc_messages
lc_time_names
lock_wait_timeout
long_query_time
low_priority_updates
max_allowed_packet
max_delayed_threads
max_error_count
max_execution_time
max_heap_table_size
max_insert_delayed_threads
max_join_size
max_length_for_sort_data
max_seeks_for_key
max_sort_length
max_sp_recursion_depth
max_statement_time
max_user_connections
min_examined_row_limit
myisam_repair_threads
myisam_sort_buffer_size
myisam_stats_method
ndb-allow-copying-alter-table
ndb_autoincrement_prefetch_sz
ndb-blob-read-batch-bytes
ndb-blob-write-batch-bytes
ndb_deferred_constraints
ndb_force_send
ndb_fully_replicated
ndb_index_stat_enable
ndb_index_stat_option
ndb_join_pushdown
ndb_log_bin
ndb_log_bin
ndb_table_no_logging
ndb_table_temporary
ndb_use_copying_alter_table
ndb_use_exact_count
ndb_use_transactions
ndbinfo_max_bytes
ndbinfo_max_rows
ndbinfo_show_hidden
ndbinfo_table_prefix
net_buffer_length
net_read_timeout
net_retry_count
net_write_timeout
new
old_alter_table
old_passwords
optimizer_prune_level
optimizer_search_depth
optimizer_switch
optimizer_trace
optimizer_trace_features
optimizer_trace_limit
optimizer_trace_max_mem_size
optimizer_trace_offset
parser_max_mem_size
preload_buffer_size
profiling
profiling_history_size
proxy_user
pseudo_slave_mode
pseudo_thread_id
query_alloc_block_size
query_cache_type
query_cache_wlock_invalidate
query_prealloc_size
rand_seed1
rand_seed2
range_alloc_block_size
range_optimizer_max_mem_size
rbr_exec_mode
read_buffer_size
read_rnd_buffer_size
session_track_gtids
session_track_schema
session_track_state_change
session_track_system_variables
show_old_temporals
sort_buffer_size
sql_auto_is_null
sql_big_selects
sql_buffer_result
sql_log_bin
sql_log_off
sql_mode
sql_notes
sql_quote_show_create
sql_safe_updates
sql_select_limit
sql_warnings
storage_engine
thread_pool_high_priority_connection
thread_pool_prio_kickup_timer
time_zone
timestamp
tmp_table_size
transaction_alloc_block_size
transaction_allow_batching
transaction_prealloc_size
transaction_write_set_extraction
```

```
tx_isolation
tx_read_only
unique_checks
updatable_views_with_limit
version_tokens_session
version_tokens_session_number
wait_timeout
warning_count
```

例：

```
set @a=20;

SET SESSION sql_mode = 'TRADITIONAL';

SET sql_mode = 'TRADITIONAL';
```

1. insert_id 在使用过程中可能会在另一个前端连接中被重置而导致主键冲突的问题。sql_auto_is_null 和 insert_id 是联合使用的，使用时也有此限制。详情参见 issue：https://github.com/actiontech/dble/issues/1252.

3.5.1.9 TRACE

用于观察单条SQL的性能，打开此开关后，可以执行需要观察性能的查询语句, 然后执行 `show trace` 来观察最后结果。
可以用 `select @@trace` 观察当前的开启状态。 详情请见 单条SQL性能trace

```
set trace=value

value:
  0
  | off
  | false
  | 1
  | on
  | true
```

例：

```
set trace=1;
```

3.5.2 SHOW语句

3.5.2.1 dble劫持的SHOW

- SHOW DATABASES
  将sharding.xml 中的所有schema展示出来 。
- SHOW CREATE DATABASE [IF NOT EXISTS] schema
  将sharding.xml 中的指定schema的创建语句展示，创建语句为dble伪造，无实际意义 。
- SHOW [FULL|ALL] TABLES [FROM db_name] [LIKE 'pattern'| WHERE expr]
  当schmea没有配置默认节点时，将schema下配置的tables直接展示出来。
  当schema有默认节点时，将语句转发至默认节点，然后将结果集与schema下配置的tables做一个去重合并，再返回给客户端。
- SHOW ALL TABLES [FROM db_name] [LIKE 'pattern'| WHERE expr]
  dble自有命令，与SHOW FULL TABLES 返回结果集类似，不同之处是Table_type这列分为了 SHARDING TABLE，sharding table， GLOBAL TABLE 。参见6.Differernce_from_MySQL_Server.md。
- SHOW [FULL] {COLUMNS | FIELDS} FROM tbl_name [{FROM|IN} db_name] [LIKE 'pattern' | WHERE expr]
  将逻辑schema转为物理schema之后下发到表所在的任意节点。
- SHOW { INDEX | INDEXES | KEYS} {FROM | IN} tbl_name [{FROM | IN} db_name] [WHERE expr]
  将逻辑schema转为物理schema之后下发到表所在的任意节点。
- SHOW CREATE TABLE tbl_name
  将逻辑schema转为物理schema之后下发到表所在的任意节点。
- SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern' | WHERE expr]
  随机转发到任意节点，收到结果集后，用本地变量进行覆盖（global 不正确？）
- SHOW CREATE VIEW view_name
  将dble层面的view展示出来
- SHOW CHARSET
  将之转为show character set 之后透传转发
- SHOW TABLE STATUS [{FROM | IN} db_name] [LIKE 'pattern' | WHERE expr]
  只是为了支持SQLyog，其中name列逻辑和show tables一致，其他列均为伪造。
- SHOW TRACE
  观察trace结果，详情请见 单条SQL性能trace

注意事项：

所有以上命令的explain结果可能不准确。

例：

```
show databases;
show full tables;
show columns from a_test;
show index from a_test;
show create table a_test;
show variables;
show charset;
```

3.5.2.2 dble透传的SHOW

除了dble劫持的特定SHOW语句外，其它SHOW语句都透传，这些语句与mysql语法相同。

例：

```
SHOW CHARACTER SET;
SHOW CHARACTER SET like 'utf8';
SHOW CHARACTER SET where maxlen=2;
```

3.5.3 KILL

3.5.3.1 KILL conn_id

其中，conn_id为 前端连接id值，可以通过运维命令show @@connection 获取。

3.5.3.1.1 举例

```
kill 1;
```

3.5.3.1.2 限制

- 在Kill自身连接的时候只会向自身写入OK包，不会有其他操作
- 如果Kill的连接在XA事务的提交或者回滚状态，不会直接关闭后端连接，会仅关闭前端连接
- 后端连接的关闭通过向MYSQL节点发送 KILL processlist_id 来完成

3.5.3.2 KILL query conn_id

其中，conn_id为 前端连接id值，可以通过运维命令show @@connection 获取。

3.5.3.2.1 举例

```
kill query 1;
```

3.5.3.2.3 说明

- dble 中kill query的实现是将正在执行语句的后端连接与前端连接相割离的方式来实现。
- 后端未执行完成的语句，取决与mysql自身的机制。

3.5.3.2.2 限制

- 对于ddl语句，不保证一致性
- 对于未开启事务的dml操作不保证一致性

3.6 存储过程支持方式

3.6.1 Syntax

Create procedure

/Hint/ CREATE [DEFINER = { user | CURRENT_USER }]

PROCEDURE sp_name ([proc_parameter[,...]])

[characteristic ...] routine_body

/Hint/ CREATE

[DEFINER = { user | CURRENT_USER }]

FUNCTION sp_name ([func_parameter[,...]])

RETURNS type [characteristic ...] routine_body

drop procedure

/Hint/ DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name

call procedure

[/Hint/] CALL sp_name([parameter[,...]])

[/Hint/] CALL sp_name[()]

3.6.2 举例

```
删除存储过程:
/*!dble:sql=select 1 from account */drop procedure if exists proc_arc;
创建存储过程:
/*!dble:sql=select 1 from account */create procedure proc_arc(userid1 int)
begin
  insert into account_arc select * from account where userid=userid1;
  update account set arc_flag=true,arc_time=now() where userid=userid1;
end;
调用存储过程:
/*!dble:sql=select 1 from account */call proc_arc(1);
```

3.6.3 限制

- dble支持存储过程和自定义函数的透传，存储过程的开发完全使用MySQL的语法，开发、调试与部署的方法同单机MySQL相同。存储过程和自定义函数需要在所有节点上创建，节点扩容的时候也需要考虑存储过程和自定义函数的迁移。
- 存储过程和自定义函数是直接发送到节点上执行，中间件不参与运算，因此要慎重使用，需要保证过程的内部不出现跨节点运算。
- 存储过程调用时，要在调用语句之前增加注解，系统根据注解透传到节点运行，存储过程的执行路径以及执行结果的正确性由开发者保证。对于只是写入数据，不返回结果的存储过程，需要注意避免重复写入数据。对于返回结果的存储过程，需要特别注意返回结果的正确性。dble不会对存储过程的结果进行汇聚运算，只能由应用端自行完成。

3.7 Utility Statements

3.7.1 USE

```
USE db_name
```

例:

```
use TESTDB;
```

3.7.2 EXPLAIN

```
EXPLAIN explainable_stmt

 SELECT statement
| DELETE statement
| INSERT statement
| REPLACE statement
| UPDATE statement
```

注意事项:

1. INSERT中表不能为自增序列表
   例:

   ```
   explain SELECT select * from a_test where id=1;
   ```

2. 在dble中，EXPLAIN 不等价于于DESC

3.7.3 EXPLAIN2

```
EXPLAIN2 shardingNode=node_name sql=sql_stmt
```

例:

```
explain2 shardingNode=dn2 sql=select * from a_test where id=1;
```

3.7.4 DESC

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

例:

```
DESC a_test id;
```

注意: 在dble中，EXPLAIN 不等价于于DESC

3.8 Hint

3.8.1 分库分表-Syntax

```
/* { ! | #}dble: {sql=SELECT select_expr  FROM  table_references  WHERE  where_condition

                  |shardingNode=shardingNode_name

                  |db_type={slave|master}}

*/   ordinary_sql
```

3.8.2 读写分离-Syntax

```
/* { ! | #}dble: {db_type={slave|master}}

                  |db_instance_url={ip:port}}

*/   ordinary_sql
/* master */  ordinary_sql
/* uproxy_dest: ip:port */  ordinary_sql
```

注意：

- 在不影响sql语句完整性的前提下，`/* master */` 和 `/* uproxy_dest: ip:port */` 可放在sql语句的首尾和中间

3.8.3 举例

```
/*!dble:sql=select 1 from sbtest */ call p_show_time();
/*!dble:shardingNode=dn1*/ update sbtest set name = 'test';
/*!dble:db_type=master*/ select count(*) from sbtest;
/*!dble:db_instance_url=127.0.0.1:3307*/ select count(*) from sbtest;
/*#dble:sql=select 1 from sbtest */ call p_show_time();
/*#dble:shardingNode=dn1*/ update sbtest set name = 'test';
/*#dble:db_type=master*/ select count(*) from sbtest;
/*#dble:db_instance_url=127.0.0.1:3307*/ select count(*) from sbtest;
select /* master */ * from sbtest;
show processlist /* uproxy_dest: 127.0.0.1:3307 */
```

3.9 其他不支持语句

- Compound-Statement Syntax
- Replication Statements
- DDL：
  - 不支持针对database的操作语句，包括alter database、drop database。create database 在管理端口支持，流量端口遇到会判断schema已经配置后返回ok，否则报错。
  - 不支持create table时的一些table option，如DATA DIRECTORY、ALGORITHM等，table option在alter table时也不能修改
  - 不支持ALTER TABLE ... LOCK ...
  - 不支持ALTER TABLE ... ORDER BY ...
  - 不支持create table ... like ...和create table ... select ...
  - 库名、表名不可修改，拆分字段的名称和类型都不可以变更
  - 不支持外键关联
  - 不支持临时表
  - 不支持分布式级别的存储过程和自定义函数
  - 不支持触发器
- DML：
  - 对于INSERT... VALUES(expr)，不支持expr中含有子查询
  - 不支持INSERT DELAYED...
  - 不支持INSERT... SELECT...
  - 不支持不包含拆分字段的INSERT语句
  - 不支持HANDLER语句
  - 不支持UPDATE多张表
  - 不支持修改拆分字段的值
  - 不支持DELETE ... ORDER BY ... LIMIT ...
  - 不支持DELETE多张表
  - 不支持DELETE/UPDATE ...LIMIT路由到一个分片表的多个节点
  - 不支持DO语句
- 查询：
  - 不支持select ... use/ignore index ...
  - 不支持select ... group by ... with rollup
  - 不支持select ... for update | lock in share mode 正确语义
  - 不支持select ... into outfile ...
  - 不支持Row Subqueries
  - 不支持select ... union [all] select ... order by ...，可写成(select ...) union [all] (select ...) order by ...
  - 不支持session变量赋值与查询，如set @rowid=0;select @rowid:=@rowid+1,id from user;
- 管理语句：
  - 不支持用户管理及权限管理语句
  - 不支持表维护语句，包括ANALYZE/CHECK/CHECKSUM/OPTIMIZE/REPAIR TABLE
  - 不支持INSTALL/UNINSTALL PLUGIN语句
  - 不支持BINLOG语句
  - 不支持CACHE INDEX/ LOAD INDEX INTO CACHE语句
  - 不支持除FLUSH TABLES [WITH READ LOCK]以外的其他FLUSH语句，FLUSH TABLE也仅语法支持无实际意义
  - 不支持RESET语句
  - 不支持大部分的运维SHOW语句，如SHOW PROFILES、SHOW ERRORS等

3.10 函数与操作符支持列表(alpha版本)

3.10.0 注意：

1. 如果能保证SQL会将函数整体下发给某个后端结点，函数支持度没有意义，支持度将托管于MySQL.
2. 在2.18.09.0 版本之前（含），涉及到中文字符集的字符串相关函数有bug。
3. 除了聚合函数经过充分测试之外，其余函数只进行了简单的测试，并没有全覆盖测试，请使用前充分测试 。
4. 未在列表中的函数，一概不支持。

3.10.1 Operators

Name	Description	Support
AND, &&	Logical AND	Y
=	Assign a value (as part of a SET statement, or as part of the SET clause in an UPDATE statement)	Y
:=	Assign a value	N
BETWEEN ... AND ...	Check whether a value is within a range of values	Y
BINARY	Cast a string to a binary string	N
&	Bitwise AND	Y
~	Bitwise inversion	Y
|	Bitwise OR	Y
^	Bitwise XOR	Y
CASE	Case operator	Y
DIV	Integer division	Y
/	Division operator	Y
=	Equal operator	Y
<=>	NULL-safe equal to operator	Y
>	Greater than operator	Y
>=	Greater than or equal operator	Y
IS	Test a value against a boolean	Y
IS NOT	Test a value against a boolean	Y
IS NOT NULL	NOT NULL value test	Y
IS NULL	NULL value test	Y
->	Return value from JSON column after evaluating path; equivalent to JSON_EXTRACT().	N
->>	Return value from JSON column after evaluating path and unquoting the result; equivalent to JSON_UNQUOTE(JSON_EXTRACT()).	N
<<	Left shift	Y
<	Less than operator	Y
<=	Less than or equal operator	Y
LIKE	Simple pattern matching	Y
-	Minus operator	Y
%, MOD	Modulo operator	Y
NOT, !	Negates value	Y
NOT BETWEEN ... AND ...	Check whether a value is not within a range of values	Y
!=, <>	Not equal operator	Y
NOT LIKE	Negation of simple pattern matching	Y
NOT REGEXP	Negation of REGEXP	Y
||, OR	Logical OR	Y
+	Addition operator	Y
REGEXP	Whether string matches regular expression	Y
>>	Right shift	Y
RLIKE	Whether string matches regular expression	N
SOUNDS LIKE	Compare sounds	N
*	Multiplication operator	N
-	Change the sign of the argument	Y
XOR	Logical XOR	Y
COALESCE()	Return the first non-NULL argument	Y
GREATEST()	Return the largest argument	Y
IN()	Check whether a value is within a set of values	Y
INTERVAL()	Return the index of the argument that is less than the first argument	Y
ISNULL()	Test whether the argument is NULL	Y
LEAST()	Return the smallest argument	Y
STRCMP()	Compare two strings	Y

3.10.2 Control Flow Functions

Name	Description	Support
CASE	Case operator	Y
IF()	If/else construct	Y
IFNULL()	Null if/else construct	Y
NULLIF()	Return NULL if expr1 = expr2	Y

3.10.3 String Functions

Name	Description	Support
ASCII()	Return numeric value of left-most character	Y
BIN()	Return a string containing binary representation of a number	N
BIT_LENGTH()	Return length of argument in bits	Y
CHAR()	Return the character for each integer passed	Y
CHAR_LENGTH()	Return number of characters in argument	Y
CHARACTER_LENGTH()	Synonym for CHAR_LENGTH()	Y
CONCAT()	Return concatenated string	Y
CONCAT_WS()	Return concatenate with separator	Y
ELT()	Return string at index number	Y
EXPORT_SET()	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string	N
FIELD()	Return the index (position) of the first argument in the subsequent arguments	Y
FIND_IN_SET()	Return the index position of the first argument within the second argument	Y
FORMAT()	Return a number formatted to specified number of decimal places	Y
FROM_BASE64()	Decode base64 encoded string and return result	N
HEX()	Return a hexadecimal representation of a decimal or string value	Y
INSERT()	Insert a substring at the specified position up to the specified number of characters	Y
INSTR()	Return the index of the first occurrence of substring	Y
LCASE()	Synonym for LOWER()	Y
LEFT()	Return the leftmost number of characters as specified	Y
LENGTH()	Return the length of a string in bytes	Y
LIKE	Simple pattern matching	Y
LOAD_FILE()	Load the named file	N
LOCATE()	Return the position of the first occurrence of substring	Y
LOWER()	Return the argument in lowercase	Y
LPAD()	Return the string argument, left-padded with the specified string	Y
LTRIM()	Remove leading spaces	Y
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set	Y
MATCH	Perform full-text search	N
MID()	Return a substring starting from the specified position	N
NOT LIKE	Negation of simple pattern matching	Y
NOT REGEXP	Negation of REGEXP	Y
OCT()	Return a string containing octal representation of a number	N
OCTET_LENGTH()	Synonym for LENGTH()	N
ORD()	Return character code for leftmost character of the argument	Y
POSITION()	Synonym for LOCATE()	N
QUOTE()	Escape the argument for use in an SQL statement	Y
REGEXP	Whether string matches regular expression	Y
REPEAT()	Repeat a string the specified number of times	Y
REPLACE()	Replace occurrences of a specified string	Y
REVERSE()	Reverse the characters in a string	Y
RIGHT()	Return the specified rightmost number of characters	Y
RLIKE	Whether string matches regular expression	N
RPAD()	Append string the specified number of times	Y
RTRIM()	Remove trailing spaces	Y

SOUNDEX()	Return a soundex string	Y
SOUNDS LIKE	Compare sounds	Y
SPACE()	Return a string of the specified number of spaces	Y
STRCMP()	Compare two strings	Y
SUBSTR()	Return the substring as specified	Y
SUBSTRING()	Return the substring as specified	Y
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter	Y
TO_BASE64()	Return the argument converted to a base-64 string	N
TRIM()	Remove leading and trailing spaces	Y
UCASE()	Synonym for UPPER()	Y
UNHEX()	Return a string containing hex representation of a number	Y
UPPER()	Convert to uppercase	Y
WEIGHT_STRING()	Return the weight string for a string	N

3.10.4 Numeric Functions and Operators

Name	Description	Support
ABS()	Return the absolute value	Y
ACOS()	Return the arc cosine	Y
ASIN()	Return the arc sine	Y
ATAN()	Return the arc tangent	Y
ATAN2(), ATAN()	Return the arc tangent of the two arguments	Y
CEIL()	Return the smallest integer value not less than the argument	Y
CEILING()	Return the smallest integer value not less than the argument	Y
CONV()	Convert numbers between different number bases	Y
COS()	Return the cosine	Y
COT()	Return the cotangent	Y
CRC32()	Compute a cyclic redundancy check value	Y
DEGREES()	Convert radians to degrees	Y
DIV	Integer division	Y
/	Division operator	Y
EXP()	Raise to the power of	Y
FLOOR()	Return the largest integer value not greater than the argument	Y
LN()	Return the natural logarithm of the argument	Y
LOG()	Return the natural logarithm of the first argument	Y
LOG10()	Return the base-10 logarithm of the argument	Y
LOG2()	Return the base-2 logarithm of the argument	Y
-	Minus operator	Y
MOD()	Return the remainder	Y
%, MOD	Modulo operator	Y
PI()	Return the value of pi	Y
+	Addition operator	Y
POW()	Return the argument raised to the specified power	Y
POWER()	Return the argument raised to the specified power	Y
RADIANS()	Return argument converted to radians	Y
RAND()	Return a random floating-point value	Y
ROUND()	Round the argument	Y
SIGN()	Return the sign of the argument	Y
SIN()	Return the sine of the argument	Y
SQRT()	Return the square root of the argument	Y
TAN()	Return the tangent of the argument	Y
*	Multiplication operator	Y
TRUNCATE()	Truncate to specified number of decimal places	Y
-	Change the sign of the argument	Y

3.10.5 Date and Time Functions

Name	Description	Support
ADDDATE()	Add time values (intervals) to a date value	Y
ADDTIME()	Add time	Y
CONVERT_TZ()	Convert from one time zone to another	N
CURDATE()	Return the current date	Y
CURRENT_DATE()	Synonyms for CURDATE()	Y
CURRENT_TIME()	Synonyms for CURTIME()	Y
CURRENT_TIMESTAMP()	Synonyms for NOW()	Y
CURTIME()	Return the current time	Y
DATE()	Extract the date part of a date or datetime expression	Y
DATE_ADD()	Add time values (intervals) to a date value	Y
DATE_FORMAT()	Format date as specified	Y
DATE_SUB()	Subtract a time value (interval) from a date	Y
DATEDIFF()	Subtract two dates	Y
DAY()	Synonym for DAYOFMONTH()	N
DAYNAME()	Return the name of the weekday	Y
DAYOFMONTH()	Return the day of the month (0-31)	Y
DAYOFWEEK()	Return the weekday index of the argument	Y
DAYOFYEAR()	Return the day of the year (1-366)	Y
EXTRACT()	Extract part of a date	Y
FROM_DAYS()	Convert a day number to a date	Y
FROM_UNIXTIME()	Format Unix timestamp as a date	Y
GET_FORMAT()	Return a date format string	Y
HOUR()	Extract the hour	Y
LAST_DAY	Return the last day of the month for the argument	N
LOCALTIME()	Synonym for NOW()	Y
LOCALTIMESTAMP()	Synonym for NOW()	Y
MAKEDATE()	Create a date from the year and day of year	Y
MAKETIME()	Create time from hour, minute, second	Y
MICROSECOND()	Return the microseconds from argument	Y
MINUTE()	Return the minute from the argument	Y
MONTH()	Return the month from the date passed	Y
MONTHNAME()	Return the name of the month	Y
NOW()	Return the current date and time	Y
PERIOD_ADD()	Add a period to a year-month	Y
PERIOD_DIFF()	Return the number of months between periods	Y
QUARTER()	Return the quarter from a date argument	Y
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format	Y
SECOND()	Return the second (0-59)	Y
STR_TO_DATE()	Convert a string to a date	Y
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments	Y
SUBTIME()	Subtract times	Y
SYSDATE()	Return the time at which the function executes	Y
TIME()	Extract the time portion of the expression passed	Y
TIME_FORMAT()	Format as time	Y
TIME_TO_SEC()	Return the argument converted to seconds	Y
TIMEDIFF()	Subtract time	Y
TIMESTAMP()	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments	N
TIMESTAMPADD()	Add an interval to a datetime expression	Y
TIMESTAMPDIFF()	Subtract an interval from a datetime expression	Y
TO_DAYS()	Return the date argument converted to days	Y
TO_SECONDS()	Return the date or datetime argument converted to seconds since Year 0	Y
UNIX_TIMESTAMP()	Return a Unix timestamp	Y
UTC_DATE()	Return the current UTC date	Y

UTC_TIME()	Return the current UTC time	Y
UTC_TIMESTAMP()	Return the current UTC date and time	Y
WEEK()	Return the week number	Y
WEEKDAY()	Return the weekday index	Y
WEEKOFYEAR()	Return the calendar week of the date (1-53)	Y
YEAR()	Return the year	Y
YEARWEEK()	Return the year and week	Y
CURRENT_DATE	Synonyms for CURDATE()	N
CURRENT_TIME	Synonyms for CURTIME()	N
CURRENT_TIMESTAMP	Synonyms for NOW()	N
LOCALTIME	Synonym for NOW()	N
LOCALTIMESTAMP()	Synonym for NOW()	N

3.10.6 Cast Functions and Operators

Name	Description	Support
BINARY	Cast a string to a binary string	N
CAST()	Cast a value as a certain type	Y
CONVERT()	Cast a value as a certain type	Y

3.10.6.1 CAST 不支持以下类型或语法

BINARY
CHAR[(N)] [charset_info] 包含charset_info时
JSON
SIGNED [INTEGER] 包含INTEGER时(druid解析问题)
UNSIGNED [INTEGER] 包含INTEGER时 (druid解析问题)

3.10.6.2 CONVERT 不支持以下类型或语法

BINARY
CHAR[(N)] [charset_info] 包含charset_info时
JSON
SIGNED [INTEGER] 包含INTEGER时(druid解析问题)
UNSIGNED [INTEGER] 包含INTEGER时 (druid解析问题)

3.10.7 Bit Functions and Operators

Name	Description	Support
BIT_COUNT()	Return the number of bits that are set	Y
&	Bitwise AND	Y
~	Bitwise inversion	Y
|	Bitwise OR	Y
^	Bitwise XOR	Y
<<	Left shift	Y
>>	Right shift	Y

3.10.8 Aggregate (GROUP BY) Functions

Name	Description	Support
AVG()	Return the average value of the argument	Y
BIT_AND()	Return bitwise AND	Y
BIT_OR()	Return bitwise OR	Y
BIT_XOR()	Return bitwise XOR	Y
COUNT()	Return a count of the number of rows returned	Y
COUNT(DISTINCT)	Return the count of a number of different values	Y
GROUP_CONCAT()	Return a concatenated string	Y
JSON_ARRAYAGG()	Return result set as a single JSON array	N
JSON_OBJECTAGG()	Return result set as a single JSON object	N
MAX()	Return the maximum value	Y
MIN()	Return the minimum value	Y
STD()	Return the population standard deviation	Y
STDDEV()	Return the population standard deviation	Y
STDDEV_POP()	Return the population standard deviation	Y
STDDEV_SAMP()	Return the sample standard deviation	Y
SUM()	Return the sum	Y
VAR_POP()	Return the population standard variance	Y
VAR_SAMP()	Return the sample variance	Y
VARIANCE()	Return the population standard variance	Y

备注: STD 和VARIANCE相关函数，因为分布式计算的局限性，精度会有一些问题，见 `STD()` / `STDDEV()` / `STDDEV_POP()` / `STDDEV_SAMP()` / `VAR_POP()` / `VAR_SAMP()` / `VARIANCE()` result precision is not correct
在AVG\SUM等相关计算函数的时候，由于dble的数据来源于MySQL的查询返回，
当MySQL默认返回的数据精度不够时，可能出现最终查询的结果和MySQL的计算结果有少许差异的情况,见
for data type float, dble and mysql may get different results

3.10.9 Full-Text Search Functions

not supported

3.10.10 XML Functions

not supported

3.10.11 Encryption and Compression Functions

not supported

3.10.12 Information Functions

not supported

3.10.13 Spatial Analysis Functions

not supported

3.10.14 JSON Functions

not supported

3.10.15 Functions Used with Global Transaction IDs

not supported

3.10.16 MySQL Enterprise Encryption Functions

not supported

3.10.17 Miscellaneous Functions

not supported

参考资料：**MySQL5.7** 函数文档

导入导出方式的支持

支持工具

1. workbench
2. dbeaver
3. mysqldump
4. navicat
5. 导入数据也可以使用mysql中的source和load data

注意点

1. 若使用mysqldump导出时，请按照以下格式进行导出，否则可能出现错误，因为有些 mysqldump 参数dble不支持。

```
./mysqldump -h127.0.0.1 -utest -P3306 -p111111 --default-character-set=utf8mb4 --master-data=2 --single-transaction --set-gtid-purged=off --hex-blob --databases schema1 > export.sql
```

1. 导入时，脚本中若存在非注释性的视图相关语句，需要注释掉或删除。
2. 导出时，因为dble对视图相关的一些语句不支持，因此尽量确保导出的dble中不存在视图。

4 协议兼容

- 4.1 基本包
- 4.2 连接建立
- 4.3 文本协议
- 4.4 二进制协议 (Prepared Statements)
- 4.5 服务响应包

4.1 基本包

- 标准包：支持
- 大包（16M以上）：支持
- 压缩包：支持, 需要全局配置，参见bootstrap.cnf
- 压缩后的大包：不支持

4.2 连接建立

4.2.1 Authentication Plugin

a.空(即默认,等同于mysql_native_password 或者8.0以后的caching_sha2_password)

b.mysql_native_password

c.caching_sha2_password

4.2.2 Capabilities

大部分权能标志位在建立连接之后是不能修改的。 因为dble的连接模型是预先建立后端连接池，所以在建立时候已经初始化了权能标志位。之后如果新的前端连接建立连接时候设置了不同的权能标志位，就无法生效传递给后端连接。
比如jdbc连接dble设置了CLIENT_FOUND_ROWS 相关的属性，是无法实际生效的。
又比如，dble设置了IGNORE_SPACE，所以不做任何操作的情况下 select @@sql_mode 会永远包含IGNORE_SPACE
这样带来的限制是，某些和函数同名的词会当成保留字来处理。参见MySQL文档 以及相关issue-972
当然，IGNORE_SPACE又是个特例，它可以通过session级别的sql_mode进行设置。

这里列举一下dble建立连接权能标志位的使用情况。

如:

名称	标记值	描述	后端连接设置值	模拟服务端权能位
CLIENT_LONG_PASSWORD	1	Use the improved version of Old Password Authentication.Assumed to be set since 4.1.1.	Y	Y
CLIENT_FOUND_ROWS	2	Send found rows instead of affected rows in EOF_Packet	Y	Y
CLIENT_LONG_FLAG	4	Get all column flags.	Y	Y
CONNECT_WITH_DB	8	Database (schema) name can be specified on connect in Handshake Response Packet.	Y	Y
CLIENT_NO_SCHEMA	16	Don't allow database.table.column.	N	N
CLIENT_COMPRESS	32	Compression protocol supported	bootstrap.cnf的useCompression选项控制	
CLIENT_ODBC	64	Special handling of ODBC behavior.No special behavior since 3.22.	Y	Y
CLIENT_LOCAL_FILES	128	Can use LOAD DATA LOCAL.	Y	Y
CLIENT_IGNORE_SPACE	256	Ignore spaces before '('.	Y	Y
CLIENT_PROTOCOL_41	512	New 4.1 protocol	Y	Y
CLIENT_INTERACTIVE	1024	This is an interactive client.	Y	Y
CLIENT_SSL	2048	Use SSL encryption for the session	N	N
CLIENT_IGNORE_SIGPIPE	4096	Client only flag.Not used.	Y	Y
CLIENT_TRANSACTIONS	8192	Client knows about transactions	Y	Y
CLIENT_RESERVED	16384	DEPRECATED:Old flag for 4.1 protocol.	N	N
CLIENT_RESERVED2	32768	DEPRECATED:Old flag for 4.1 authentication.	Y	Y
CLIENT_MULTI_STATEMENTS	65536	Enable/disable multi-stmt support	Y	Y
CLIENT_MULTI_RESULTS	131072	Enable/disable multi-results	Y	Y
CLIENT_PS_MULTI_RESULTS	262144	Multi-results and OUT parameters in PS-protocol	N	N
CLIENT_PLUGIN_AUTH	524288	Client supports plugin authentication.	N	Y
CLIENT_CONNECT_ATTRS	1048576	Client supports connection attributes.	N	N
CLIENT_PLUGIN_AUTH_LENENC_CLIENT_DATA	2097152	Enable authentication response packet to be larger than 255 bytes.	N	N
CLIENT_CAN_HANDLE_EXPIRED_PASSWORDS	4194304	Don't close the connection for a user account with expired password.	N	N
CLIENT_SESSION_TRACK	8388608	Capable of handling server state change information.	N	N
CLIENT_DEPRECATE_EOF	16777216	Client no longer needs EOF_Packet and will use OK_Packet instead.	N	N
CLIENT_SSL_VERIFY_SERVER_CERT	1UL << 30	Verify server certificate	N	N
CLIENT_REMEMBER_OPTIONS	1UL << 31	Don't reset the options after an unsuccessful connect.	N	N

权能标志位定义：

参考 https://dev.mysql.com/doc/dev/mysql-server/8.0.13/group__group__cs__capabilities__flags.html

4.3 文本协议

4.3.1 Supported

- COM_INIT_DB
  Specifies the default schema for the connection.
- COM_PING
  Sends a packet containing one byte to check that the connection is active.
- COM_QUERY
  Sends the server an SQL statement to be executed immediately.Support Multi-Statement.
- COM_QUIT
  Client tells the server that the connection should be terminated.
- COM_SET_OPTION
  Enables or disables server option.
- COM_CHANGE_USER
  Resets the connection and re-authenticates with the given credentials.
- COM_RESET_CONNECTION
  Resets a connection without re-authentication.
  - 关闭后端连接(rollback & unlock)
  - 事务状态情况
  - 用户变量清空
  - 系统变量恢复成系统默认值
  - prepare清空
  - 上下文(字符集，隔离级别)恢复成为默认值
  - LAST_INSERT_ID 置零
- COM_FIELD_LIST
  MySQL Doc said that it is deprecated from 5.7.11 . But some tools are still use it, like OGG or MariaDB client.

4.3.1.1 Multi-Statement

- Supported
  - DML:select/insert/update/replace/delete
  - DDL
  - OTHER
    - BEGIN;
    - COMMIT;
    - LOCK TABLE
    - UNLOCK TABLES
    - START
    - KILL
    - USE
    - ROLLBACK
    - MYSQL_CMD_COMMENT
    - MYSQL_COMMENT
    - SELECT VERSION_COMMENT (SELECT @@VERSION_COMMENT)
    - SELECT DATABASE（ select database() ）
    - SELECT USER（select user()）
    - SELECT VERSION (select version())
    - SELECT SESSION_INCREMENT(select @@session.auto_increment_increment)
    - SELECT SESSION_ISOLATION(select @@session.tx_isolation)
    - SELECT LAST_INSERT_ID(select last_insert_id(#) as `id`)
    - SELECT IDENTITY(select @@identity)
    - SELECT SESSION_TX_READ_ONLY（select @@session.tx_read_only）
- Not Supported
  - EXPLAIN
  - EXPLAIN2
  - DESCRIBE
  - SET
  - SHOW DATABASES/TABLES/TABLE_STATUS/COLUMNS/INDEX/CREATE_TABLE/VARIABLES/CREATE_VIEW/CHARSET
  - HELP
  - LOAD_DATA_INFILE_SQL
  - CREATE_VIEW
  - REPLACE_VIEW
  - ALTER_VIEW
  - DROP_VIEW

4.3.2 Not Supported

- COM_DEBUG
  Forces the server to dump debug information to stdout
- COM_STATISTICS
  Get internal server statistics.
- COM_CREATE_DB
- COM_DROP_DB

4.3.3 Internal

- COM_SLEEP

Used inside the server only.

- COM_CONNECT an internal command in the server.
- COM_TIME an internal command in the server.
- COM_DAEMON an internal command in the server.
- COM_DELAYED_INSERT an internal command in the server.

4.3.4 Deprecated

- COM_PROCESS_INFO
  Deprecated from 5.7.11.
- COM_PROCESS_KILL
  Deprecated from 5.7.11.
- COM_SHUTDOWN
  Deprecated from 5.7.9.
- COM_REFRESH
  Deprecated from 5.7.11.

4.4 二进制协议 (Prepared Statements)

4.4.1 Supported

- COM_STMT_CLOSE
  Closes a previously prepared statement.
- COM_STMT_EXECUTE
  Executes a previously prepared statement.
- COM_STMT_RESET
  Resets a prepared statement on client and server to state after preparing.
- COM_STMT_SEND_LONG_DATA
  When data for a specific column is big, it can be sent separately.

4.4.2 Faked

- COM_STMT_PREPARE
  Prepares a statement on the server
  COM_STMT_PREPARE response is faked. Parameters and numbers are faked.

4.4.3 Not Supported

- COM_STMT_FETCH
  Fetches rows from a prepared statement

4.4.4 MySQL Flow

注意点：

- 可通过url方式指定useCursorFetch=true，开启分批从server查询数据。
- 但是jdbc中默认fetchSize为0，jdbc中必须fetchSize > 0才会发送fetch包分批查询数据，否则和普通prepareStatement没有区别。

4.4.5 Dble Flow

4.4.6 额外限制

在使用mariadb-java-client版本1.5.0及以上版本连接dble的时候，executeBatch使用的时候存在注意事项，需要禁用driver的useBatchMultiSend选项，否则会出现不固定的错误
原因是在mariadb-java-client版本1.5.0及以上版本中添加了一个新的优化功能useBatchMultiSend，具体的优化在executeBatch的时候driver在未收到上一条数据执行结果的情况下提前下发后续的执行语句，这个优化与当前版本的dble内部实现不兼容
大致情况如下图所示：

原文详情https://mariadb.com/kb/en/library/option-batchmultisend-description/

4.5 服务响应包

兼容支持:

- EOF_Packet
- ERR_Packet
- OK_Packet
- LOCAL_INFILE Packet
- PACKET_LOCAL_INFILE
- PACKET_RESULTSET

5. 已知限制

- 5.1 druid引发的限制
- 5.2 其他已知限制

5.1 druid引发的限制

1. INSERT ... VALUE ... 和 INSERT ... VALUES ...这个两个语句中，VALUE[S]关键字必须写正确，否则会出现解析正常，但结果错误。
   这个是由于druid的bug引起的。当VALUES[S]关键字错误时，druid会把这个错误的关键字当作别名处理。
   已向druid提issue2218 。
   dble相关issue dble_issue_379.
2. 当开启黑名单时，`SHOW ALL TABLES` 等dble的自定义语句可能不支持
3. 其他
   参见 dble_issue_788

5.2 其他已知限制

1. 显式配置的父子表，在子表插入数据时，不能多值插入
   原因 : 子表插入数据时,如有parentkey不是父表的拆分列。需要去对应父表的拆分结点中反向查询路由规则，如果此时是多值插入，就会变成多值反查，查询功能较难完成，即使完成，性能也会很差。
   相关issue : https://github.com/actiontech/dble/issues/12
2. 不建议使用JDBC的rewriteBatchedStatements=true
   原因 :
   insert : 多条简单insert拼接成一条长的insert.. values(),(),一个com_query包 ，对于dble来说，可能引发分布式事务，降低性能和数据一致性。
3. 使用JDBC的useServerPrepStmts=true会降低性能
   原因 : dble是将前端的Binary Protocol 转为Text Protocol，收到结果集之后再反向转回协议，所以需要额外的工作，降低了性能。
4. lock/unlock 实现不完整
   相关issue : https://github.com/actiontech/dble/issues/38
5. 不支持在schema.xml里配置复合主键（3.20.07.0移除此配置）
   原因 : schema.xml里的主键作用为主键路由和全局序列，这两者暂不支持多主键。
   相关issue : https://github.com/actiontech/dble/issues/70
6. 不保证主键唯一性 由于dble不要求配置在table中的主键在创建表格时一定为主键，并且由于dble后端分布式存储的方式, dble不对于主键做唯一性检查，并且允许用户对于非分片列的主键字段进行任意更新
7. 并发更新多行数据/全局表数据可能导致死锁超时
   原因 : 并发情况下，分布式下发sql可能乱序。
   相关issue : https://github.com/actiontech/dble/issues/85
8. 防火墙导致无响应
   原因 : 防火墙可能drop包，java层面的tcp_keeplive无法指定时间。
   相关issue : https://github.com/actiontech/dble/issues/87
9. 方差/标准差精度问题
   原因 : 方差计算方式导致
   相关issue : https://github.com/actiontech/dble/issues/100
10. `order by lock in share mode/for update` , lock clause is ignored
    原因 : 无法支持。
    相关issue : https://github.com/actiontech/dble/issues/127
11. 不支持 _charset_name ‘string’ `_charset_name+b'val'`
    相关issue : https://github.com/actiontech/dble/issues/262
    相关issue : https://github.com/actiontech/dble/issues/267
12. 未能正确支持 set sql_select_limit
    相关issue : https://github.com/actiontech/dble/issues/331
13. 日期拆分算法中，sEndDate如果不配置，default node就无用
    原因 : 算法设计问题。
    相关issue : https://github.com/actiontech/dble/issues/357
14. selece @@sql_mode 始终包含IGNORE_SPACE
    原因 : 后端权能标志位设置导致，参见4.2节内容。
    相关issue : https://github.com/actiontech/dble/issues/364
15. replace ... into
    由于replace的语义为如果存在则替换，如果不存在则新增，所以在使用表格自增主键的时候 如果对于自增表格使用replace且ID不存在，那么就会插入一条指定ID的数据，并不会自动生成ID
16. kill 语句杀自身session，直接返回ok，不会有任何实质性操作
17. 由于2.19.01版本在rule/schema/server.xml中加入了version字段造成的BUG，导致在2.19.01使用zk集群进行同步化操作的时候要求version字段不能为空或者不填，此问题在后续版本会进行修复
18. 若mysql节点上设置了 `set global local_infile = 0` ,dble load data指令执行报错
    原因 : dble会将load data指令转换为 `load data local infile ...` 指令下发至各个后端mysql，所以各个节点 `local_infile` 参数需要开启。
    相关issue : https://github.com/actiontech/dble/issues/1111
19. 不能正确支持 set @@sql_auto_is_null=on; 原因 : set @@sql_auto_is_null=on 的语义是用新生成的自增序列取代null,在dble层不做实现。
    相关issue : https://github.com/actiontech/dble/issues/978
20. 不能正确支持mariadb-java-client版本1.5.0及以上版本的useBatchMultiSend优化 原因： mariadb-java-client版本1.5.0及以上版本修改了prepare执行的时候的交互逻辑 相关issue ： https://github.com/actiontech/dble/issues/1244
21. 复杂查询会透传难以理解的报错，建议结合explain语句分析 原因： 需要枚举所有错误来做替换.成本高，收益低 相关issue： https://github.com/actiontech/dble/issues/1449

6. 与MySQL Server的差异化描述

这里主要描述一些与MySQL Server不同的行为，这些行为不是bug，是分布式场景下的一些正常的行为表现，但与已知的MySQL行为不一致。

- 6.1 事务中遇到主键冲突需要显式回滚
- 6.2 INSERT不能显式指定自增序列
- 6.3 增加"show all tables"
- 6.4 去除了增删改的message信息
- 6.5 information_schema等库的支持

6.1 事务中遇到主键冲突需要显式回滚，MySQL不需要

具体表现如下:

MYSQL行为:

```
[test_yhq]>select * from char_columns_4;
+----+--------+
| id | c_char |
+----+--------+
|  1 | xx |
|  4 | z  |
+----+--------+
2 rows in set (0.02 sec)
[test_yhq]>begin;
Query OK, 0 rows affected (0.01 sec)

[test_yhq]>insert into char_columns_4 values(1,'yy');
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
[test_yhq]>insert into char_columns_4 values(2,'yy');
Query OK, 1 row affected (0.00 sec)

[test_yhq]>commit;
Query OK, 0 rows affected (0.02 sec)
```

dble行为:

```
[testdb]>select * from sharding_four_node order by id;
+----+--------+-----------+
| id | c_flag | c_decimal |
+----+--------+-----------+
|  1 | 1_1    |    1.0000 |
|  2 | 2      |    2.0000 |
|  3 | 3      |    3.0000 |
+----+--------+-----------+
3 rows in set (0.28 sec)

begin;
Query OK, 0 rows affected (0.01 sec)

[testdb]>insert into sharding_four_node values(1,'1',1.0);
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
[testdb]>insert into sharding_four_node values(13,'13',13.0);
ERROR 1003 (HY000): Transaction error, need to rollback.Reason:[ errNo:1062 Duplicate entry '1' for key 'PRIMARY']
[testdb]>commit;
ERROR 1003 (HY000): Transaction error, need to rollback.Reason:[ errNo:1062 Duplicate entry '1' for key 'PRIMARY']
```

6.2 INSERT自增序列表由dble生成，不能显式指定自增列，MySQL可以。

具体表现如下:

MySQL行为:

```
desc mysql_autoinc;
+--------+------------+------+-----+---------+----------------+
| Field  | Type       | Null | Key | Default | Extra          |
+--------+------------+------+-----+---------+----------------+
| c_char | char(255)  | YES  |     | NULL    |                |
| id     | bigint(20) | NO   | PRI | NULL    | auto_increment |
+--------+------------+------+-----+---------+----------------+
2 rows in set (0.02 sec)

[test_yhq]>insert into mysql_autoinc values('1',1);
Query OK, 1 row affected (0.01 sec)
```

dble行为

```
desc sharding_four_node_autoinc;
+--------+------------+------+-----+---------+----------------+
| Field  | Type       | Null | Key | Default | Extra          |
+--------+------------+------+-----+---------+----------------+
| c_char | char(255)  | YES  |     | NULL    |                |
| id     | bigint(20) | NO   | PRI | NULL    | auto_increment |
+--------+------------+------+-----+---------+----------------+
2 rows in set (0.01 sec)
[testdb]>insert into sharding_four_node_autoinc values('2',2);
ERROR 1064 (HY000): In insert Syntax, you can't set value for Autoincrement column!
```

6.3 ADD "show all tables"

The optional ALL modifier causes SHOW TABLES to display a second output column with values of BASE TABLE for a table ,VIEW for a view, SHARDING TABLE for a sharding table and GLOBAL TABLE for a global table.

具体表现如下:

```
[testdb]>show all tables;
+----------------------------+----------------+
| Tables in testdb           | Table_type     |
+----------------------------+----------------+
| global_four_node           | GLOBAL TABLE   |
| global_four_node_autoinc   | GLOBAL TABLE   |
| global_two_node            | GLOBAL TABLE   |
| sbtest1                    | SHARDING TABLE |
| sharding_four_node         | SHARDING TABLE |
| sharding_four_node2        | SHARDING TABLE |
| sharding_four_node_autoinc | SHARDING TABLE |
| sharding_two_node          | SHARDING TABLE |
| single                     | SHARDING TABLE |
| customer                   | BASE TABLE     |
| district                   | BASE TABLE     |
+----------------------------+----------------+
11 rows in set (0.02 sec)
```

6.4 去除了增删改的message 信息

具体表现如下:

MySQL行为:

```
mysql> insert into sharding_two_node values(9,'9',9.0),(10,'10',10.0);
Query OK, 2 rows affected (0.24 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

dble行为:

```
mysql> insert into sharding_two_node values(11,'11',11.0),(12,'12',12.0);
Query OK, 2 rows affected (0.49 sec)
```

6.5 information_schema等库的支持

背景

用户使用 Navicat Premium 12 连接dble时，Navicat Premium 12会查询 information_schema，mysql 等数据库中系统表的数据。
dble 需要支持查询这些系统表的语句，保证这些 driver 的正常使用。

Navicat Premium12

1. SELECT SCHEMA_NAME, DEFAULT_CHARACTER_SET_NAME, DEFAULT_COLLATION_NAME FROM information_schema.SCHEMATA;

该语句查询当前 mysql 实例中的所有 scheme的名称，character set 以及 collation。该语句会影响Navicat Premium 12的使用。
dble中的scheme是逻辑上的，可以通过 SchemeConfig 获取所有 scheme 的名称，scheme 的 character set 以及 collation 使用默认返回。
问题1：dble中的scheme是逻辑上的，会对应多个shardingNode，会出现shardingNode的 character set 以及 collation 与 默认character set 和 collation 不同,需要运维安装MySQL时候保证。
问题2：当前dble中处理 SCHEMATA 系统表时，只是对表名做了判断，并未对查询字段做检验。

以下语句不影响driver的使用

对此种语句的处理是根据请求字段，伪造一个行数为0的报文返回。

```
1. SELECT TABLE_SCHEMA, TABLE_NAME, TABLE_TYPE
    FROM information_schema.TABLES WHERE TABLE_SCHEMA = 'testdb'
    ORDER BY TABLE_SCHEMA, TABLE_TYPE
2. SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, COLUMN_TYPE
    FROM information_schema.COLUMNS
    WHERE TABLE_SCHEMA = 'testdb'
    ORDER BY TABLE_SCHEMA, TABLE_NAME
3. SELECT DISTINCT ROUTINE_SCHEMA, ROUTINE_NAME, PARAMS.PARAMETER
    FROM information_schema.ROUTINES LEFT JOIN
    ( SELECT SPECIFIC_SCHEMA, SPECIFIC_NAME,
        GROUP_CONCAT(CONCAT(DATA_TYPE, ' ', PARAMETER_NAME) ORDER BY ORDINAL_POSITION SEPARATOR ', ') PARAMETER, ROUTINE_TYPE
    FROM information_schema.PARAMETERS GROUP BY SPECIFIC_SCHEMA, SPECIFIC_NAME, ROUTINE_TYPE
    )PARAMS
    ON ROUTINES.ROUTINE_SCHEMA = PARAMS.SPECIFIC_SCHEMA AND
    ROUTINES.ROUTINE_NAME = PARAMS.SPECIFIC_NAME AND
    ROUTINES.ROUTINE_TYPE = PARAMS.ROUTINE_TYPE
    WHERE ROUTINE_SCHEMA = 'testdb' ORDER BY ROUTINE_SCHEMA
4. SELECT TABLE_NAME, CHECK_OPTION, IS_UPDATABLE, SECURITY_TYPE, DEFINER
    FROM information_schema.VIEWS
    WHERE TABLE_SCHEMA = 'testdb' ORDER BY TABLE_NAME ASC
5. SELECT * FROM information_schema.ROUTINES
    WHERE ROUTINE_SCHEMA = 'testdb' ORDER BY ROUTINE_NAME
6. SELECT EVENT_CATALOG, EVENT_SCHEMA, EVENT_NAME, DEFINER, TIME_ZONE,
    EVENT_DEFINITION, EVENT_BODY, EVENT_TYPE, SQL_MODE, STATUS, EXECUTE_AT,
    INTERVAL_VALUE, INTERVAL_FIELD, STARTS, ENDS, ON_COMPLETION, CREATED,
    LAST_ALTERED, LAST_EXECUTED, ORIGINATOR, CHARACTER_SET_CLIENT,
    COLLATION_CONNECTION, DATABASE_COLLATION, EVENT_COMMENT
    FROM information_schema.EVENTS WHERE EVENT_SCHEMA = 'testdb'
    ORDER BY EVENT_NAME ASC
7. SELECT COUNT(*) FROM information_schema.TABLES
    WHERE TABLE_SCHEMA = 'testdb' UNION
   SELECT COUNT(*)
    FROM information_schema.COLUMNS
    WHERE TABLE_SCHEMA = 'testdb' UNION
   SELECT COUNT(*) FROM information_schema.ROUTINES WHERE ROUTINE_SCHEMA = 'testdb'
```

7 开发者须知

- 7.1 SQL开发编写原则
- 7.2 dble连接Demo
- 7.3 其他注意事项

7.1 SQL开发编写原则

- 分布式数据库处理的是分布式关系运算，其SQL的优化方法与单机关系数据库有所不同，侧重考虑的分布式环境中的网络开销。分布式执行计划中，应该尽量将SQL中的运算下推到底层各个节点执行，避免跨节点运算，从而减少网络开销、提升SQL执行效率。

1）执行计划

访问数据时的一组有序的操作步骤集合，称为执行计划。dble的执行计划分为两个层次：dble层的执行计划与节点层的执行计划。对执行计划进行分析，可以了解中间件和节点是否对SQL语句生成了最优的执行计划，是否有优化的空间，从而为SQL优化提供重要的参考信息。 在SQL语句执行前，dble会根据SQL语句的基本信息，判断该SQL语句应该在哪些节点上执行，将SQL改写成在节点上执行的具体形式，并决定采用何种策略进行数据合并与计算等，这就是 dble层的执行计划。节点层的执行计划就是原生的MySQL执行计划。 dble用EXPLAIN指令来查看dble层的执行计划。如:

```
explain select id,accountno from account where userid=2;
```

EXPLAIN指令的执行结果包括语句下发的节点，实际下发的SQL语句和数据的合并操作的信息。这些信息是系统静态分析产生的，并没有真正的执行语句。 通过EXPLAIN2命令可查看指定节点上的执行计划。如:

```
explain2 shardingNode=dn1 sql=select id,accountno from account where userid=2;
```

explain2会将sql语句加上explain下发到指定的shardingNode执行，并把节点上explain的结果返回调用者。

2）SQL优化

对于SQL优化有以下原则:

- SQL语句中尽可能带有拆分字段，并且拆分字段过滤条件的取值范围越小，越有助于提高查询速度。在数据写入时，必须给拆分字段赋值。
- 拆分字段的查询条件尽可能是等值条件。
- 如果拆分字段的条件是IN子句，则IN后面的值的数目应尽可能少。特别注意，随着业务增长，某些IN子句的条件会随之增长。
- 如果SQL语句不带有拆分字段，那么DISTINCT、GROUP BY和ORDER BY在同一个SQL语句中尽量只出现一种。
- 数据查询时，应该尽量减少节点返回的结果数量，这样能够使消耗的网络带宽最小，使查询性能能够达到最优状态。

3）跨节点查询

跨节点查询通常发生在下列语句中:

- 涉及多个分片的分布式 Join;
- 涉及多个分片的聚合函数;
- 复杂子查询。

这些语句可以通过以下策略来优化:

- 合理调整查询语句，使查询条件中包含拆分字段的等值条件，这样就能够将语句下推到单一节点执行。
- 参与Join的表配置相同的拆分规则，查询时将拆分字段作为Join的关联条件，这样Join操作就可以在节点内完成。
- 将参与Join的表中数据量较小的表配置成全局表，通过数据冗余避免跨节点。
- 如果一定需要跨节点Join，尽量对驱动表添加更多的过滤条件，从而使参与跨节点Join的数据量尽可能的少。
- 如果查询涉及到了多个节点，则尽量不要使用limit a,b这样的子句
- GROUP子句尽量包含拆分字段。
- 对语句进行改写，将复杂子查询分解为多条语句来执行。
- 子查询尽可能改写成Join的形式。

7.2 dble连接Demo

开发框架连接

ibatis

利用ibatis连接dble时，连接方式与MySQL相同。下面是一个简单示例。 JDBC配置:

```
jdbc.driverClass=com.mysql.jdbc.Driver
jdbc.jdbcUrl=jdbc: mysql://127.0.0.1:8066/TESTDB?useUnicode=true&characterEncoding=utf-8
jdbc.user=root
jdbc.password=123456
```

映射文件配置:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.mapper.UserMapper">
  <insert id="saveUser" parameterType="com.bean.User">
    insert into user(id,name,phone,birthday)
    values (0,#{name},#{phone},#{birthday})
    <selectKey keyProperty="id" order="after" resultType="int">
      select last_insert_id() as id
    </selectKey>
  </insert>
  <delete id="deleteUserById" parameterType="java.lang.String">
    delete from user where id=#{id}
  </delete>
  <update id="updateUser" parameterType="com.bean.User">
    update user set name=#{name},phone=#{phone},birthday=#{birthday} where id=#{id}
  </update>
  <update id="updateUsers">
    /\*!dble:sql=select * from user;\*/update users set usercount=(select count(\*) from user),ts=now()
  </update>
  <select id="getUserById" parameterType="java.lang.String" resultType="com.bean.User">
    select \* from user where id=#{id}
  </select>
  <select id="getUsers" resultType="com.bean.User">
    select \* from user
  </select>
```

语句select last_insert_id() as id可用来获取新写入记录的ID。 updateUsers方法用到了 dble的注解，由于ibatis中的符号#具有特殊含义，因此注解中不能含有#。

hibernate

利用hibernate连接dble时，连接方式与MySQL相同。下面是一个简单示例。 hibernate.cfg.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc: mysql://192.168.58.51:8066/testdb?useUnicode=true&characterEncoding=utf-8</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">123456</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
  <property name="hibernate.format_sql">true</property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <mapping resource="com/actiontech/test/News.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

News.hbm.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="com.actiontech.test.News" table="news_table">
    <id name="id" type="java.lang.Integer">
      <column name="id" />
    </id>
    <property name="title" type="java.lang.String">
      <column name="title" />
    </property>
    <property name="content" type="java.lang.String">
      <column name="content" />
    </property>
  </class>
</hibernate-mapping>
```

News.java:

```
package com.actiontech.test;
public class News {
  private Integer id;
  private String title;
  private String content;
  public Integer getId() {
    return id;
  }
  public void setId(Integer id) {
    this.id = id;
  }
  public String getTitle() {
    return title;
  }
  public void setTitle(String title) {
    this.title = title;
  }
```

public String getContent() { return content; } public void setContent(String content) { this.content = content; } }</pre> NewsManager.java:

```
package com.actiontech.test;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
public class NewsManager {
  public static void main(String[] args)
    throws Exception {
    Configuration config = new Configuration().configure();
    SessionFactory factory = config.buildSessionFactory();
    Session session = factory.openSession();
    Transaction transaction = session.beginTransaction();
    News news = new News();
    news.setId(10);
    news.setTitle("dble示例");
    news.setContent("Hibernate 连接dble的第一个例子");
    session.save(news);
    transaction.commit();
    session.close();
    factory.close();
  }
}
```

dble虽然支持Hibernate但不建议使用Hibernate，因为Hibernate无法控制SQL的生成，无法做到对查询SQL的优化，大数量下可能会出现性能问题。

JDBC

利用JDBC连接dble时，连接方式与MySQL相同。下面是一个简单示例：

```
package com.actiontech.test;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.atomic.AtomicLong;
public class SingleMixEngine {
  public static void main(String[] args) throws Exception {
```

```
    Class.forName("com.mysql.jdbc.Driver");
    Properties props = new Properties();
    props.setProperty("user", "root");
    props.setProperty("password", "123456");
    SingleMixEngine engine = new SingleMixEngine();
    engine.execute(props,"jdbc:mysql://192.168.58.51:8066/testdb");
  }
  final AtomicLong tmAl = new AtomicLong();
  final String tableName="news_table";
  public void execute(Properties props,String url) {
    CountDownLatch cdl = new CountDownLatch(1);
    long start = System.currentTimeMillis();
    for (int i = 0; i < 1; i++) {
      TestThread insertThread = new TestThread(props,cdl, url);
      Thread t = new Thread(insertThread);
      t.start();
      System.out.println("Test start");
    }
    try {
      cdl.await();
      long end = System.currentTimeMillis();
      System.out.println("Test end,total cost:" + (end-start) + "ms");
    } catch (Exception e) {
    }
  }

  class TestThread implements Runnable {
    Properties props;
    private CountDownLatch countDownLatch;
    String url;
    public TestThread(Properties props,CountDownLatch cdl,String url) {
      this.props = props;
      this.countDownLatch = cdl;
      this.url = url;
    }
    public void run() {
      Connection connection = null;
      PreparedStatement ps = null;
      Statement st = null;
      try {
        connection = DriverManager.getConnection(url,props);
        connection.setAutoCommit(true);
        st = connection.createStatement();
        String dropSql = "drop table if exists " + tableName;
        System.out.println("Execute SQL:\n\t"+dropSql);
        st.execute(dropSql);

        String createSql = "create table " + tableName + "(id int,title varchar(20),content varchar(50))";
        System.out.println("Execute SQL:\n\t"+createSql);
        st.execute(createSql);

        String insertSql = "insert into " + tableName + " (id,title,content) values(?,?,?)";
        System.out.println("Prepared SQL:\n\t"+insertSql);
        ps = connection.prepareStatement(insertSql);
        for (int i = 1; i <= 3; i++) {
          ps.setInt(1,i);
          ps.setString(2, "测试"+i);
          ps.setString(3, "这是第"+i+"条测试数据");
          ps.execute();
          System.out.println("Insert data:\t"+i+","+"测试"+i+","+"这是第"+i+"条测试数据");
        }

        String querySQL = "select * from " + tableName + " order by id";
        System.out.println("Execute SQL:\n\t"+querySQL);
        ResultSet rs = st.executeQuery(querySQL);
        int colcount = rs.getMetaData().getColumnCount();
        System.out.println("Current Data:");
        while(rs.next()){
          for(int i=1;i<=colcount;i++){
            System.out.print("\t"+rs.getString(i));
          }
          System.out.println();
        }

        String updateSql = "update " + tableName + " set title='test1' where id=1";
        System.out.println("Execute SQL:\n\t"+updateSql);
        st.execute(updateSql);
        rs = st.executeQuery(querySQL);
        System.out.println("Current Data:");
        while(rs.next()){
          for(int i=1;i<=colcount;i++){
            System.out.print("\t"+rs.getString(i));
          }
          System.out.println();
        }

        String deleteSql = "delete from " + tableName + " where id=2";
        System.out.println("Execute SQL:\n\t"+deleteSql);
        st.execute(deleteSql);
        rs = st.executeQuery(querySQL);
        System.out.println("Current Data:");
        while(rs.next()){
          for(int i=1;i<=colcount;i++){
            System.out.print("\t"+rs.getString(i));
          }
          System.out.println();
        }

        String createIndexSql = "create index idx_1 on " + tableName + "(title)";
        System.out.println("Execute SQL:\n\t"+createIndexSql);
        st.execute(createIndexSql);

        String dropIndexSql = "drop index idx_1 on " + tableName;
        System.out.println("Execute SQL:\n\t"+dropIndexSql);
        st.execute(dropIndexSql);
      } catch (Exception e) {
        System.out.println(new java.util.Date().toString());
```

```
        e.printStackTrace();
      } finally {
        if (ps != null)
          try {
            ps.close();
          } catch (SQLException e1) {
            e1.printStackTrace();
          }
        if (connection != null)
          try {
            connection.close();
          } catch (SQLException e1) {
            e1.printStackTrace();
          }
        this.countDownLatch.countDown();
      }
    }
  }
}
```

7.3 其他注意事项

- 在应用分布式数据库时，首先要考虑清楚，是否真的需要分片，在所有环境下分片都是不得已的选择，必然会增加研发、运维、管理的复杂度，现在有很多技术例如分区、复制、缓存等都能提高系统的处理速度和吞吐量，也许我们应用这些技术能够实现技术目标而无需分片
- 数据分片之前，请确保已经对单机数据库做了优化，包括系统架构、硬件、MySQL版本、数据读写模式等，数据分片之后这些优化依然非常有意义
- 配置拆分规则时，还应该同时考虑运维的复杂性，以及未来系统的扩容。
- 数据量较大时，DDL语句会耗时较多，分布式环境下这个问题会更加突出，因此应尽量在业务空闲时进行DDL操作。

8 配置示例

- 8.1 时间戳方式全局序列的配置
- 8.2 MySQL-offset-step 方式全局序列的配置

8.1 时间戳方式全局序列的配置

配置表tb的id列为时间戳方式全局序列，并按id列分片

1）cluster.conf

```
sequenceHandlerType=2
sequenceStartTime=2010-10-01 09:42:54
...
```

2）bootstrap.conf

```
instanceId=1
...
```

3）user.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dble:user SYSTEM "user.dtd">
<dble:user xmlns:dble="http://dble.cloud/">
    <managerUser name="test" password="test"/>
    <shardingUser name="abc" password="abc" schemas="myschema" maxCon="1000000">
    </shardingUser>

</dble:user>
```

4）db.xml

```
<?xml version="1.0"?>
<!DOCTYPE dble:db SYSTEM "db.dtd">
<dble:db xmlns:dble="http://dble.cloud/" >
    <dbGroup name="host_1" rwSplitMode="0" delayThreshold="100">
        <heartbeat >select USER()</heartbeat>
        <dbInstance name="hostM1" url="172.100.10.101:3306" user="test1" password="test1" maxCon="1000" minCon="1000" primary="true" />
    </dbGroup>
    <dbGroup name="host_2" rwSplitMode="0" delayThreshold="100">
        <heartbeat >select USER()</heartbeat>
        <dbInstance name="hostM2" url="172.100.10.102:3306" user="test1" password="test1" maxCon="1000" minCon="1000" primary="true" />
    </dbGroup>
</dble:db>
```

5）sharding.xml

```
<?xml version="1.0"?>
<!DOCTYPE dble:sharding SYSTEM "sharding.dtd">
<dble:sharding xmlns:dble="http://dble.cloud/">
    <schema name="myschema" shardingNode="dn1">
    <shardingTable name="sbtest1"  shardingNode="dn1,dn2" function="mod" shardingColumn="id" incrementColumn="id" />
    </schema>
    <shardingNode name="dn1" dbGroup="host_1" database="dble"/>
    <shardingNode name="dn2" dbGroup="host_2" database="dble"/>
    <function name="mod" class="Hash">
                <property name="partitionCount">2</property>
                <property name="partitionLength">1</property>
        </function>
</dble:sharding>
```

6）实验

```mysql mysql -utest -p111111 -h127.0.0.1 -P8066 -Dmyschema mysql> drop table if exists sbtest1; Query OK, 0 rows affected (0.05 sec) mysql> create table sbtest1(id bigint(20), k int unsigned not null default '0', primary key(id)); Query OK, 0 rows affected (0.05 sec)

mysql> insert into sbtest1 values(2); Query OK, 1 row affected (0.11 sec)

mysql> select * from sbtest1; 查看序列

注意事项，自增列的数据类型要是bigint
```

8.2 MySQL-offset-step 方式全局序列的配置

配置表sbtest1的id列为MySQL-offset-step方式全局序列，并按id列分片

1）cluster.conf

```
sequenceHandlerType=1
...
```

2）user.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dble:user SYSTEM "user.dtd">
<dble:user xmlns:dble="http://dble.cloud/" >
    <managerUser name="test" password="test"/>

    <shardingUser name="abc" password="abc" schemas="myschema" maxCon="1000000">
    </shardingUser>

</dble:user>
```

3）db.xml

```
<?xml version="1.0"?>
<!DOCTYPE dble:db SYSTEM "db.dtd">
<dble:db xmlns:dble="http://dble.cloud/" >
    <dbGroup name="host_1" rwSplitMode="0" delayThreshold="100">
        <heartbeat >select USER()</heartbeat>
        <dbInstance name="hostM1" url="172.100.10.101:3306" user="test1" password="test1" maxCon="1000" minCon="1000" primary="true" />
    </dbGroup>
    <dbGroup name="host_2" rwSplitMode="0" delayThreshold="100">
        <heartbeat >select USER()</heartbeat>
        <dbInstance name="hostM2" url="172.100.10.102:3306" user="test1" password="test1" maxCon="1000" minCon="1000" primary="true" />
    </dbGroup>
</dble:db>
```

4）sharding.xml

```
<?xml version="1.0"?>
<!DOCTYPE dble:sharding SYSTEM "sharding.dtd">
<dble:sharding xmlns:dble="http://dble.cloud/" >
    <schema name="myschema" shardingNode="dn1">
    <shardingTable name="sbtest1"  shardingNode="dn1,dn2" function="mod" shardingColumn="id" incrementColumn="id" />
    </schema>
    <shardingNode name="dn1" dbGroup="host_1" database="dble"/>
    <shardingNode name="dn2" dbGroup="host_2" database="dble"/>

    <function name="mod" class="Hash">
                <property name="partitionCount">2</property>
                <property name="partitionLength">1</property>
        </function>
</dble:sharding>
```

5）sequence_db_conf.properties

```
#sequence stored in shardingNode
`myschema`.`sbtest1`=dn1
```

myschema, sbtest1, dn1均为在sharding.xml配置的值

在dn1分片对应的后端host_1/dble上执行dble安装目录下的conf/dbseq.sql(路径根据情况自行修改)。

```
mysql -h172.100.10.101 -utest1 -ptest1 -Ddble
mysql>source conf/dbseq.sql
```

在上述sql文件执行成功后向创建的表DBLE_SEQUENCE插入自增相关的配置数据：

```
mysql -h172.100.10.101 -utest1 -ptest1 -Ddble
mysql>INSERT INTO DBLE_SEQUENCE VALUES ('`mytest`.`sbtest1`', 16, 1);
```

DBLE_SEQUENCE列说明：

- name：在sequence_db_conf.properties中配置的逻辑数据库和表名
- current_value：全局序列的当前值
- increment：每次取出多少值用于全局序列，注意全局序列递增的步长固定是1

6）实验

登录dble业务端口创建设置了全局序列并以其分片的表：

```
mysql -utest -p111111 -h127.0.0.1 -P8066 -Dmyschema
mysql> drop table if exists sbtest1;
Query OK, 0 rows affected (0.05 sec)
mysql> create table sbtest1(id int, k int unsigned not null default '0', primary key(id));
Query OK, 0 rows affected (0.05 sec)

mysql> insert into sbtest1 values(2);
Query OK, 1 row affected (0.11 sec)

mysql> select * from sbtest1;
+----+---+
| id | k |
+----+---+
| 17 | 2 |
+----+---+
1 row in set (0.01 sec)
```

从上面的sql可以看到，在设置DBLE_SEQUENCE表时，current_value设置的是16，在insert后变为了17。

配置要点：

- sequence_db_conf.properties：

```
`myschema`.`sbtest1`=dn1
```

- 在sequence_db_conf.properties配置的后端分片dn1对应的后端数据库上执行dbseq.sql， 并插入全局序列表对应的记录

9 sysbench压测dble示例

- 9.1 测试环境及架构
- 9.2 修改dble配置
- 9.3 使用sysbench进行压测

9.1 测试环境及架构

版本信息

- Sysbench version: 1.0
- Dble version: 5.6.23-dble-2.19.11.0-2d7c4911b7a4fecaa9eb0299f49c32ec11e97c42-20200228124218
- MySQL version: 5.7.25

测试架构

- sysbench独立服务器（172.20.134.1）
- dble独立服务器（172.20.134.2）
- 3台mysql独立服务器（172.20.134.3，172.20.134.4，172.20.134.5）

9.2 修改dble配置

说明： 此配置仅为示例配置，并非调优配置，请根据运行环境自行调优，调优步骤参考：2.18 性能观测以及调试概览

1. bootstrap.cnf

调整如下参数 -Dprocessors=10 -DbackendProcessors=10 -DprocessorExecutor=8 -DbackendProcessorExecutor=6 -DsqlExecuteTimeout=3000000

2.user.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dble:user SYSTEM "user.dtd">
<dble:user xmlns:dble="http://dble.cloud/" version="4.0">
    <managerUser name="root" password="111111" />
    <shardingUser name="test" password="111111" schemas="sbtest" maxCon="1000000">
    </shardingUser>
</dble:user>
```

3.db.xml

```
<?xml version="1.0"?>
<!DOCTYPE dble:db SYSTEM "db.dtd">
<dble:db xmlns:dble="http://dble.cloud/" version="4.0">
    <dbGroup name="host_1" rwSplitMode="0" delayThreshold="-1">
        <heartbeat >select USER()</heartbeat>
        <dbInstance name="hostM1" url="172.20.134.3:3306" user="test1" password="test1" maxCon="1000" minCon="100" primary="true"/>
    </dbGroup>
    <dbGroup name="host_2" rwSplitMode="0" delayThreshold="-1">
        <heartbeat >select USER()</heartbeat>
        <dbInstance name="hostM1" url="172.20.134.4:3306" user="test1" password="test1" maxCon="1000" minCon="100" primary="true"/>
    </dbGroup>
    <dbGroup name="host_3" rwSplitMode="0" delayThreshold="-1">
        <heartbeat >select USER()</heartbeat>
        <dbInstance name="hostM1" url="172.20.134.5:3306" user="test1" password="test1" maxCon="1000" minCon="100" primary="true"/>
    </dbGroup>
</dble:db>
```

4.sharding.xml

```
<?xml version="1.0"?>
<!DOCTYPE dble:sharding SYSTEM "sharding.dtd">
<dble:sharding xmlns:dble="http://dble.cloud/" version="4.0">
<schema name="sbtest">
    <shardingTable name="sbtest1" shardingNode="dn$1-9" function="hash-sysbench" shardingColumn="id" />
</schema>
<shardingNode name="dn$1-3" dbGroup="host_1" database="dbledb$1-3" />
<shardingNode name="dn$4-6" dbGroup="host_2" database="dbledb$4-6" />
<shardingNode name="dn$7-9" dbGroup="host_3" database="dbledb$7-9" />
<function name="hash-sysbench" class="Hash">
        <property name="partitionCount">9</property>
        <property name="partitionLength">1</property>
    </function>
</dble:sharding>
```

5.在后端mysql节点创建相应物理库

- 172.20.134.3:3306 创建库：dbledb1, dbledb2, dbledb3
- 172.20.134.4:3306 创建库：dbledb4, dbledb5, dbledb6
- 172.20.134.5:33066 创建库：dbledb7, dbledb8, dbledb9

9.3 使用sysbench进行压测

说明：
如果在压测中遇到问题请参考：how about a sysbench-testing-quick-start; 如果在此参考中未包括您遇到的问题请在issue下添加评论，将问题及配置做出说明。
注意：
使用sysbench压测语句，在高并发下会出现主键冲突的报错。

```
FATAL: mysql_drv_query() returned error 1062 (Duplicate entry '49823' for key 'PRIMARY') for query 'INSERT INTO sbtest1 (id, k, c, pad) VALUES (49823, 58210, '27111667985-11552069038-79242882109
-05602914209-02374993639-32242662584-65155028223-08319627673-44873060047-22215118936', '07405724915-32799061660-96650146042-59717172693-66753749407')'
```

解决方式见如上链接。

数据清理：

> /usr/share/sysbench/oltp_read_write.lua --mysql-db=sbtest --mysql-host=172.20.134.2 --mysql-port=8066 --mysql-user=test1 --mysql-password=test1 --auto_inc=off --tables=1 --table-size=100000 --threads=4 --time=30 --report-interval=1 --max-requests=0 --percentile=95 --db-ps-mode=disable --skip-trx=on cleanup

数据准备：

> /usr/share/sysbench/oltp_read_write.lua --mysql-db=sbtest --mysql-host=172.20.134.2 --mysql-port=8066 --mysql-user=test1 --mysql-password=test1 --auto_inc=off --tables=1 --table-size=100000 --threads=4 --time=30 --report-interval=1 --max-requests=0 --percentile=95 --db-ps-mode=disable --skip-trx=on prepare

执行压测：

> /usr/share/sysbench/oltp_read_write.lua --mysql-db=sbtest --mysql-host=172.20.134.2 --mysql-port=8066 --mysql-user=test1 --mysql-password=test1 --auto_inc=off --tables=1 --table-size=100000 --threads=4 --time=30 --report-interval=1 --max-requests=0 --percentile=95 --db-ps-mode=disable --skip-trx=on run

- A.1 ErrorCode
- A.2 原理解释
- A.3 使用说明

- Max Connections
- Out Of Memory Error
- The Problem Of Hint
- NestLoop Parameters Lead To Temptable Exception
- Can't Get Variables From ShardingNode
- Port Already In Use 1984
- Sharding Column Cannot Be Null

dble-MaxConnections

Issue

- [Err] 3009 - java.io.IOException: the max activeConnnections size can not be max than maxconnections.

Resolution

1. 检查是否使用了过多的短链，导致频繁建立连接，链接池不够用
   > 注意：不要过多的使用短链，容易消耗dble资源
2. maxCon配置过小，调大maxCon

配置名称	配置内容	默认值/单位	作用原理或应用
maxCon	控制最大连接数	默认1024	大于此连接数之后,建立连接会失败.注意当各个用户的maxcon总和值大于此值时，以当前值为准 全局maxCon不作用于manager用户

Root Cause

配置的链接池数量不够用，导致报错。

Relevant Content

数据库连接池

1. **maxWait**
   从连接池获取连接的超时等待时间，单位毫秒。
   > 注意：maxActive不要配置过大，虽然业务量飙升后还能处理更多的请求，但其实连接数的增多在很多场景下反而会减低吞吐量
2. **connectionProperties**
   可以配置 connectTimeout 和 socketTimeout，单位都是毫秒。 connectTimeout 配置建立 TCP 连接的超时时间；socketTimeout 配置发送请求后等待响应的超时时间。
   > 注意：不设置这两项超时时间，服务会有高的风险。
   > 例如网络异常下 socket 没有办法检测到网络错误，如果没有设置 socket 网络超时，连接就会一直等待 DB 返回结果，造成新的请求都无法获取到连接。
3. **maxActive**
   最大连接池数量，允许的最大同时使用中的连接数。
   > 注意：当业务出现大流量涌入时，连接池耗尽，maxWait未配置或者配置为 0 时将无限等待，导致等待队列越来越长，表现为业务接口大量超时，实际吞吐量越低。

dble-OutOfMemoryError

Setting

load data语句，一共有17G的数据，每条语句4K
load data相关参数值均为默认

Issue

- INFO | jvm 1 | 2019/06/28 14:55:37 | Exception in thread "backendBusinessExecutor17" java.lang.OutOfMemoryError: GC overhead limit exceeded.

Resolution

1. 调小maxRowSizeToFile参数值，减少占用内存量

配置名称	配置内容	默认值	详细作用原理或应用
maxCharsPerColumn	每列所允许最□字符数	默认为65535	每□所允许最□字符数
maxRowSizeToFile	需要持久化的最□□数	默认为10000	当load data的数据□数超过阈值后，会将数据保存在□件中以防OOM

1. 调大wrapper.conf中的Xmx值

> 注意：发生OOM，可以关注一下这个值的大小是否配置合适

Root Cause

1. maxRowSizeToFile参数设置过大，对于机器来说超出了承载的数据量，导致内存溢出
2. 配置文件中Xmx设置过小，内存不够

> 注意：On-Heap □□由JVM 参数Xms ,Xmx 决定，就是正常服务需要的内存，由jvm□动分配和回收。

Relevant Content

load data相关配置

设置load data相关参数时，不要过度的调大，防止OOM

JVM配置

以下为建议值：

1. dble总内存=0.6 可□物理内存(刨除操作系统,驱动等的占□)
2. Xmx = 0.4 dble总内存
3. MaxDirectMemorySize = 0.6 * dble总内存

堆内存分配

1. JVM初始分配的内存由-Xms指定，默认是物理内存的1/64；
2. JVM最大分配的内存由-Xmx指定，默认是物理内存的1/4。
3. 默认空余堆内存小于40%时，JVM就会增大堆直到-Xmx的最大限制；空余堆内存大于70%时，JVM会减少堆直到-Xms的最小限制。

dble-TheProblemOfHint

Setting

- mysql> */#dble:sql=select 1 from rp_cre_data_mobile_track_cmcc /* call update_track();
- 预期：表为分片表，sql语句应该下发到所有节点
- 结果：只有一个节点在执行

Issue

- mysql> show @@processlist;

Front_Id	shardingNode	BconnID	user	Front_Host	db	Command	Time	State	Info
33	NULL	NULL	root	略	NULL	NULL	0	updating	NULL
34	NULL	NULL	root	略	NULL	NULL	0	updating	NULL
35	NULL	NULL	root	略	NULL	NULL	0	updating	NULL
41	dn9	9372	root	略	db9	Query	0	updating	NULL
42	NULL	NULL	root	略	NULL	NULL	0	updating	NULL
43	NULL	NULL	root	略	NULL	NULL	0	updating	NULL
30	NULL	NULL	admin	略	NULL	NULL	0	updating	NULL

Resolution

1. 将注解方式： */#dble:type=..../*
   改为： */!dble:type=..../*
2. 或者在mysql client端，加上 -c 选项

> 注意：mysql --help
> -c, --comments
> Preserve comments. Send comments to the server. The default is --skip-comments (discard comments), enable with --comments.

Root Cause

1. */#dble:type=..../* 这种注释方式是mysql的标准注释
2. 如果不加 -c 选项，默认注释会被skip

Relevant Content

hint作用

1. 指定路由，比如强制读写分离。
2. 帮助dble支持一些不能实现的语句，如单节点内存储过程的创建和调用，如insert…select…；

Hint语法

Hint语法有三种形式：

1. /*!dble:type=....*/
2. /*#dble:type=...*/
3. /* */(只适用于读写分离功能)

type有4种值可选：shardingNode，db_type，sql，db_instance_url。
type详情请见： https://actiontech.github.io/dble-docs-cn/2.Function/2.04_hint.html

Hint注意事项

- 使用select语句作为注解SQL，不要使用delete/update/insert 等语句。delete/update/insert 等语句虽然也能用在注解中，但这些语句在SQL处理中有一些额外的逻辑判断，会降低性能，不建议使用
- 注解SQL 禁用表关联语句
- 使用hint做DDL需要额外执行reload @@metadata
- 使用hint做session级别的系统变量和环境变量可能不会生效，请慎用
- dble的注解和MySQL原生注解含义不同, 想通过MySQL原生注解来设置变量或者指定索引是无法得到预期结果的。如#1169
- 使用注解并不额外增加的执行时间；从解析复杂度以及性能考虑，注解SQL应尽量用最简单的SQL 语句，如select id from tab_a where id='10000'；
- 能不用注解也能够解决的场景，尽量不用注解

dble-NestLoop Parameters Lead To Temptable Exception

Setting

- 开启了NestLoop优化，设置NestLoop值的范围为4
- 两个表join关联：student表和class表
- student表结构：id列、name列、class_name列，主键id
- class表结构：id列，class_name列、teacher_name列
- `SELECT class.teacher_name FROM student LEFT JOIN class on student.class_name=class.class_name WHERE student.name="张三";`

Issue

- com.actiontech.dble.plan.common.exception.TempTableException: temptable too much rows,[rows size is 5].

Resolution

- 调大NestLoop中的默认参数值，如下：

配置名称	配置内容	默认值/单位	详细作用原理或应用
useJoinStrategy	是否使用nestLoop优化	默认不使用	开启之后会尝试判断join两边的where来重新调整查询SQL下发的顺序
nestLoopConnSize	临时表阈值	默认4	若临时表数于这两个值乘积，则报告错误
nestLoopRowsSize	临时表阈值	默认2000	若临时表数于这两个值乘积，则报告错误

- 或者关闭NestLoop，不使用其优化，如下：
  - `<property name="useJoinStrategy">false</property>`
- 为了更直观，本例提前调小参数值，引发场景复现

Root Cause

- 使用了NestLoop优化，但是依据NestLoop规则选择出的表数据量太大，超出了NestLoop默认范围
- NestLoop优化规范

> 注意：
>
> - 对于两表join时，NestLoop选择小表作为驱动表 （同样适用于多join）
> - 判断依据：是否有where条件，有where条件的作为小表
> - 当两个表都有where条件或者都没有where条件，NestLoop无法判断，不起作用
> - 在不确定哪个表为小表时，不建议开启NestLoop

- 针对本例SQL做详细说明：如下
- `SELECT class.teacher_name FROM student LEFT JOIN class on student.class_name=class.class_name WHERE student.name="张三";`

根据dble中NestLoop优化规则可知：

1. SQL中使用了student表和class表
2. where条件限制指定于student表，所以NestLoop可以根据规则选择出student表作为驱动表（小表）
3. 实际SQL中student表的数据量较大，超出了NestLoop值的范围，引发报错

Relevant Content

MySQL的多表连接之**NestLoop**介绍

1. NestLoop:
   - 对于被连接的数据子集较小的情况，Nested Loop是个较好的选择。
   - Nested Loop就是扫描一个表（外表），每读到一条记录，就根据Join字段上的索引去另一张表（内表）里面查找，若Join字段上没有索引查询优化器一般就不会选择 Nested Loop。
   - 在Nested Loop中，内表（一般是带索引的大表）被外表（也叫“驱动表”，一般为小表——不紧相对其它表为小表，而且记录数的绝对值也较小，不要求有索引）驱动，外表返回的每一行都要在内表中检索找到与它匹配的行，因此整个查询返回的结果集不能太大。
2. NestLoop优缺点

类别	使用条件	相关资源	特点	缺点
NestLoop	任何条件	CPU、磁盘I/O	当有高选择性索引或进行限制性搜索时效率比较高，能够快速返回第一次的搜索结果。	当索引丢失或者查询条件限制不够时，效率很低；当表记录数多时，效率低。

dble-Can't get variables from shardingNode

Setting

- db.xml 片段

```
<dbGroup name="localhost1" rwSplitMode="0" delayThreshold="100">
<heartbeat>show slave status</heartbeat>
<dbInstance host="hostM1" url="localhost:3306" user="root"  password="nE7jA%5m" maxCon="1000" minCon="10" primary="true" >  </dbInstance>
</dbGroup>
<dbGroup name="localhost2" rwSplitMode="0" delayThreshold="100">
<heartbeat>show slave status</heartbeat>
<dbInstance host="hostM2" url="localhost:3306" user="root"  password="nE7jA%5m" maxCon="1000" minCon="10" primary="true">  </dbInstance>  </dbGroup>
```

Issue

- 查看dble启动日志:

```
Running dble-server...
wrapper | --> Wrapper Started as Console
wrapper | Launching a JVM...
jvm 1   | Wrapper (Version 3.2.3)
http://wrapper.tanukisoftware.org
jvm 1   | Copyright 1999-2006 Tanuki Software, Inc. All Rights Reserved.
jvm 1   |
jvm 1   | java.io.IOException:Can't get variables from shardingNode  ...
wrapper | <-- Wrapper Stopped
```

Resolution

1. 检查mysql版本及是否正常启动，不支持mysql5.1，请升级mysql, 如果启动正常见下一步;
2. db.xml中的root用户能否通过配置文件的信息连接mysql，连接成功见下一步;
3. 检查root用户权限;
4. 连接mysql，并执行show variables命令，未执行成功见下一步;
5. 修改配置文件中dbGroup指定的后端数据库密码，更新配置文件，dble正常启动

Root Cause

- 通过db.xml 配置信息成功连接mysql后，并不能执行show variables
- 由于mysql 5.7 初始化之后，首次使用随机密码登陆，没有修改密码，无法对数据库进行操作

Relevant Content

1. 安装好mysql5.7后，第一次初始化数据库
2. 随机密码登录mysql，首次登录后，mysql要求必须修改默认密码，否则不能执行任何其他数据库操作，这样体现了不断增强的Mysql安全性。
3. 第一次登陆后必须更改密码:
    - mysql> show databases;
    - ERROR 1820 (HY000): You must reset your password using ALTER USER statement before executing this statement.
    - mysql > set password = password('xxxxxx');

dble-Port already in use:1984

Issue

- wrapper.log-Error1

```
STATUS | wrapper | 2019/07/23 16:37:06 | --> Wrapper Started as Daemon
STATUS | wrapper | 2019/07/23 16:37:06 | Launching a JVM...
INFO | jvm 1 | 2019/07/23 16:37:06 | OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=64M; support was removed in 8.0
INFO | jvm 1 | 2019/07/23 16:37:07 | 错误: 代理抛出异常错误: java.rmi.server.ExportException: Port already in use: 1984; nested exception is:
INFO | jvm 1 | 2019/07/23 16:37:07 | java.net.BindException: Address already in use (Bind failed)
INFO | jvm 1 | 2019/07/23 16:37:07 | sun.management.AgentConfigurationError: java.rmi.server.ExportException: Port already in use: 1984;
```

- wrapper.log-Error2

```
STATUS | wrapper  | 2019/07/26 16:12:48 | --> Wrapper Started as Daemon
STATUS | wrapper  | 2019/07/26 16:12:49 | Launching a JVM...
INFO   | jvm 1    | 2019/07/26 16:12:49 | Wrapper (Version 3.2.3) http://wrapper.tanukisoftware.org
INFO   | jvm 1    | 2019/07/26 16:12:49 |   Copyright 1999-2006 Tanuki Software, Inc.  All Rights Reserved.
INFO   | jvm 1    | 2019/07/26 16:12:49 |
INFO   | jvm 1    | 2019/07/26 16:12:51 | java.net.BindException: Address already in use
INFO   | jvm 1    | 2019/07/26 16:12:51 |       at sun.nio.ch.Net.bind0(Native Method)
INFO   | jvm 1    | 2019/07/26 16:12:51 |       at sun.nio.ch.Net.bind(Net.java:433)
INFO   | jvm 1    | 2019/07/26 16:12:51 |       at sun.nio.ch.Net.bind(Net.java:425)
INFO   | jvm 1    | 2019/07/26 16:12:51 |       at sun.nio.ch.ServerSocketChannelImpl.bind(ServerSocketChannelImpl.java:223)
INFO   | jvm 1    | 2019/07/26 16:12:51 |       at com.actiontech.dble.net.NIOAcceptor.<init>(NIOAcceptor.java:46)
```

Resolution

- 根据Error1：
  修改配置文件wrapper.conf：修改被占用端口1984
  -Dcom.sun.management.jmxremote.port=1984
- 根据Error2：
  netstat -nap 查看程序运行的pid（8066和9066依然存在）
  kill -9 pid 杀掉进程
- 启动dble成功

Root Cause

- 已经启动过一个开启jmx服务的java程序后，再启动dble会报这个错；
- dble启动过程中会占用三个端口：业务端口，管理端口，jvm对外提供jmx服务端口；
- jmx可通过jconsole连接上jvm，观测jvm的运行状态。

Relevant Content

1. JVM
   JVM是一种使用软件模拟出来的计算机，它用于执行Java程序，有一套非常严格的技术规范，是Java跨平台的依赖基础。
   Java虚拟机有自己想象中的硬件，如处理器，堆栈，寄存器等，还有相应的指令系统它允许Java程序就好像一台计算机允许c或c++程序一样。

2. JMX
   所谓JMX，是Java Management Extensions的缩写，是Java管理系统的一个标准、一个规范，也是一个接口，一个框架。
   它和JPA、JMS是一样的，就是通过将监控和管理涉及到的各个方面的问题和解决办法放到一起，统一设计，以便向外提供服务，以供使用者调用。

3. Jconsole
   Jconsole是JDK自带的监控工具，它用于连接正在运行的本地或者远程的JVM，对运行在java应用程序的资源消耗和性能进行监控，并画出大量的图表，提供强大的可视化界面。
   自身占用的服务器内存很小，甚至可以说几乎不消耗。

dble-Sharding column can't be null

Setting

- sharding.xml部分配置如下：

```
<sharingTable shadingColumn="number" ... >
...
<function name="rangeLong" class="NumberRange">
  <property name="mapFile">partition.txt</property>
  <property name="defaultNode">0</property>
</function>
```

- create table account (id int(10),number int(10) not null,name varchar(20) not null);
- insert into account (id,number,name) values (1,NULL,'aaa');

Issue

ERROR 1064 (HY000): Sharding column can't be null when the table in MySQL column is not null

Resolution

- number列和name列的插入值不为NULL；
- 或者修改number列为允许插入NULL值；
  ALTER TABLE `account` MODIFY `number` VARCHAR (20);
- 注意：上一步的前提是：
  在blacklist中开启参数alterTableAllow;

```
<blacklist name="bk1">
    <property name="alterTableAllow">true</property>
</blacklist>
```

并修改sharding-by-range中的拆分列，dble不允许对分片键或ER键进行alter，会造成无法分片；

```
<sharingTable shadingColumn="id" ... >
```

- alter列值为允许插入空值后，再将拆分列修改为原值。

Root Cause

1. 在MySQL中执行相同insert：
   报错：ERROR 1048 (23000): Column 'number' cannot be null
2. desc查看表结构：number列和name列均定义为非空列，不允许插入空值。

Field	Type	Null	Key	Default	Extra
id	int(10)	YES		NULL	
number	int(10)	NO		NULL	
name	varchar(20)	NO		NULL	

- How To Use Explain To Resolve The Distribution Rules Of Group Gy
- Hash And ConsistentHashing And Jumpstringhash

dble-How To Use Explain To Resolve The Distribution Rules Of Group Gy

Questions

一张数据表做了分表。 如果查询里有group by分组统计，运行原理是按范围去各分表查询出数据后，再到中间件里进行分组统计的吗?

Conclusions

- 会在中间件中做数据重聚合
  1. 利用explain工具查看sql的执行过程
  2. dble 在explain上做了大量改善，相比mycat能提供更详实的执行计划，更准确的反映SQL语句的执行过程

For Example

1. 配置好配置文件：

sharding.xml：

```
<shardingTable name="eee" shardingNode="dn1,dn2"  function="hashLong" shardingColumn="id"/>
...
<function name="hashLong" class="Hash">
       <property name="partitionCount">2</property>
       <property name="partitionLength">128</property>
    </function>
```

1. 在dble client创建表 eee 并插入数据： mysql> select * from eee;

   | id | name | | -- | -- | | 1 | 上海 | | 2 | 广州 | | 3 | 杭州 | | 4 | 北京 | | 5 | 北京 | | 130 | 北京 | | 131 | 北京 | | 132 | 上海 | | 133 | 上海 | | 134 | 上海 |

mysql> select name,count(name) from eee group by name;

name	COUNT(name)
上海	4
北京	4
广州	1
杭州	1

1. 利用explain工具查看sql的执行过程

SHARDING_NODE	TYPE	SQL/REF
dn1_0	BASE SQL	select `eee.name` ,COUNT(name) as `_$COUNT$_rpda_0` from `eee` GROUP BY `eee.name` ASC
dn2_0	BASE SQL	select `eee.name` ,COUNT(name) as `_$COUNT$_rpda_0` from `eee` GROUP BY `eee.name` ASC
merge_1	MERGE	dn1_0; dn2_0
aggregate_1	AGGREGATE	merge_1
shuffle_field_1	SHUFFLE_FIELD	aggregate_1

Instructions

由explain的结果可知：

1. dble将sql语句下发到对应shardingnode执行
2. 将对应shardingnode数据结果进行merge
3. 对merge后的数据进行group by聚合
4. SHUFFLE_FIELD进行整理，达到用户预期的结果

   > 注意：普通用户可以不关注SHUFFLE_FIELD

Relevant Content

dble的内部功能层

1. 在dble内部，包括了三个部分：面向app的连接层，内部功能层，面向myslq的连接池。
2. 内部功能层实现：前端请求接收，处理过后由后端协议层发出，将数据返回给用户。

   内部功能涉及到了简单查询和复杂查询：

   - 简单查询：直接下发单个/多个节点
   - 复杂查询：dble内部需要进行排序、聚合、join、group by，结果集计算
   - 详细介绍：https://opensource.actionsky.com/dble-lesson-one/

dble内部各线程池简介

1. 后端IO接收线程 处理来自MySQL连接的网络包（sql的执行结果、查询的结果集）。
2. 后端业务处理线程 简单查询的后端MySQL返回处理，并将结果转发反馈给客户端。
3. 复杂查询处理线程 处理复杂查询MySQL返回结果，包括结果集的聚合，对比，排序，去重，子查询语句下发等。

> PS：dble中仅此线程池不限线程数量

dble-Hash And ConsistentHashing And Jumpstringhash

Questions

- dble中的hash和一致性hash是否相同
- 为什么没有一致性hash

Conclusions

- dble中的hash并非一致性hash
- 从一致性hash的性能和均衡性来看，已被跳增一致性hash取代

Instructions

dble-hash

- 配置如下：
- sharding.xml

```
<function name="hashLong" class="hash">
<property name="partitionCount">1,2</property>
<property name="partitionLength">10,20</property>
</function>
```

- 或者

```
<function name="hashLong" class="hash">
<property name="partitionCount">4</property>
<property name="partitionLength">10</property>
</function>
```

> 注意： partitionCount：指定分区的区间数
> partitionLength：指定各区间的长度
> 模值（M）为最后一个区间段的末尾值（C1L1 + ... + CnLn）

对于dble-hash来说，该算法可以均匀地将数据分布到各个节点。

例如：
Count=2,Length=2 -> [0,2)[2,4) -> 模值=4
key分别取值：1,2,3,4,5,6,7,8
数据分布情况如下：

node1	node2
1,4,5,8	2,3,6,7

当增加一个节点，使Count=3,Length=2 -> [0,2)[2,4)[4,6) -> 模值=6 key分别取值：1,2,3,4,5,6,7,8
数据分布情况如下：

node1	node2	node3
1,6,7	2,3,8	4,5

从这个例子中我们可以看出：当节点数增加时，大多数旧的数据都需要重新分布，而重新分布的成本就是需要在count数发生变化的时候，进行数据迁移，大多数的数据都需要重新移动。

summary

因此，对于这种算法，当node数发生变化（增加、移除）后，大多数旧的数据都需要重新分布，并进行数据迁移。

Consistent Hashing

一致性哈希，将整个哈希值空间组织成一个虚拟的圆环，整个空间按顺时针方向组织。例如我们有NodeA、Node B、Node C、Node D四个节点，有数据A、数据B、数据C、数据D四个数据对象，根据一致性哈希算法，数据A会被分布到Node A上，B被分布到Node B上，C被分布到Node C上，D被分布到Node D上:

> 数据A ——> NodeA
> 数据B ——> NodeB
> 数据C ——> NodeC
> 数据D ——> NodeD

现假设Node C不幸宕机，此时数据A、B、D不会受到影响，只有数据C被重分布到Node D。 如果在系统中增加一台服务器Node X:

> 数据A ——> NodeA
> 数据B ——> NodeB
> 新增 ——> NodeX
> 数据C ——> NodeC
> 数据D ——> NodeD

数据A、B、D不受影响，只有数据C需要重分布到新的Node X 。也就是说旧数据之间不会发生数据的变动。

虽然一致性Hash算法解决了节点变化导致的数据迁移问题，但是数据项分布的均匀性不够好。 一致性哈希算法分布不均匀的原因是因为：将node进行哈希后，这些值并没有均匀地落在环上，因此，这些节点所管辖的范围（每个节点实际占据环上的区间大小不）并不均匀，最终导致了数据分布的不均匀。

因此，为使得每个节点在环上所“管辖”更加均匀，一致性哈希算法引入了虚拟节点机制，即对每一个服务节点计算多个哈希，每个计算结果位置都放置一个服务节点，称为虚拟节点。

例如：
我们有NodeA、Node B；可以为每台服务器计算三个虚拟节点，于是可以分别计算 “Node A#1”、“Node A#2”、“Node A#3”、“Node B#1”、“Node B#2”、“Node B#3”的哈希值，形成六个虚拟节点。
同时数据定位算法不变，只是多了一步虚拟节点到实际节点的映射，例如定位到“Node A#1”、“Node A#2”、“Node A#3”三个虚拟节点的数据均定位到Node A上,“Node B#1”、“Node B#2”、“Node B#3”三个虚拟节点的数据均定位到Node B上，用来解决分布不均的问题。

> Node A#1 ——> NodeA
> Node A#2 ——> NodeA
> Node A#3 ——> NodeA
> Node B#1 ——> NodeB
> Node B#2 ——> NodeB
> Node B#3 ——> NodeB

因此，通过增加虚节点的方法，使得每个节点在环上所“管辖”更加均匀。这样就既保证了在节点变化时，尽可能小的影响数据分布的变化，而同时又保证了数据分布的均匀。也就是靠增加“节点数量”加强管辖区间的均匀。

summary

对于一致性hash算法来说，当node数发生变化（增加、移除）后，旧的数据之间没有发生变动，只是需要将部分旧数据重新分布到新的节点。

Jumpstringhash

- 配置如下：
- sharding.xml

```
<function name="jumphash" class="jumpStringHash">
<property name="partitionCount">2</property>
<property name="hashSlice">0:2</property>
</function>
```

> 注意： partitionCount：分片数量 hashSlice：分片截取长度

该算法该算法来自于Google的一篇文章A Fast, Minimal Memory, Consistent Hash Algorithm，核心思想是通过概率分布的方法将一个hash值在每个节点分布的概率变成1/n，并且可以通过更简便的方法可以计算得出，并且分布也更加均匀。
设计目标是把对象均匀地分布在所有节点中（平衡性）；当节点数量变化时，只需要把一些对象从旧节点移动到新节点，不需要做其它移动（单调性）。

根据论文原理，可以这样说明：

- 假设我们有四个节点，比作为四个桶，分别是：0,1,2,3。
- 数据落入第一个桶(0)的概率是1；落入第二个桶(1)的概率是1/(n+1)，也就是1/2；落入第三个桶(2)的概率是1/(n+1)，也就是1/3；落入第四个桶(3)的概率是1/(n+1)，也就是1/4。
- 由此，一般规律是：数据落入每个桶中的概率，有占比 n/(n+1) 的结果保持不变，而有 1/(n+1) 跳变为 n+1。
- 而每个数据都是落入至index（max）中，比如数据a有概率分布至0,1,2,3四个桶中，那么最终会落入至3内。

当增加一个桶时：由0,1,2,3变为0,1,2,3,4，需要重新分布的数据仅是每个桶内有概率分布到新桶内的数据，旧数据之间不会发生变化。

summary

由于Jumpstringhash采用的是概率分布的结果，因此计算相对于一致性hash较简单。

> - Jumpstringhash相比于一致性hash算法来说，占用内存更小，计算更快，数据分布更均匀；
> - Jumpstringhash和一致性hash算法，对于节点变动的情况下，都是将部分旧数据重新分布到新节点上，旧数据之间不会发生变动；
> - Jumpstringhash和一致性hash算法相比于dble-hash来说，节点变动的情况下，旧数据之间不会发生变动；
> - dble-hash相比于Jumpstringhash和一致性hash算法来说，最大的优势在于计算方便，可以人为的快速计算出结果。

- ToBeContinued2

待更新