# 概述

# 安装

# 架构

## 整体架构

## SQLEngine

## 管控模块

## 存储模块

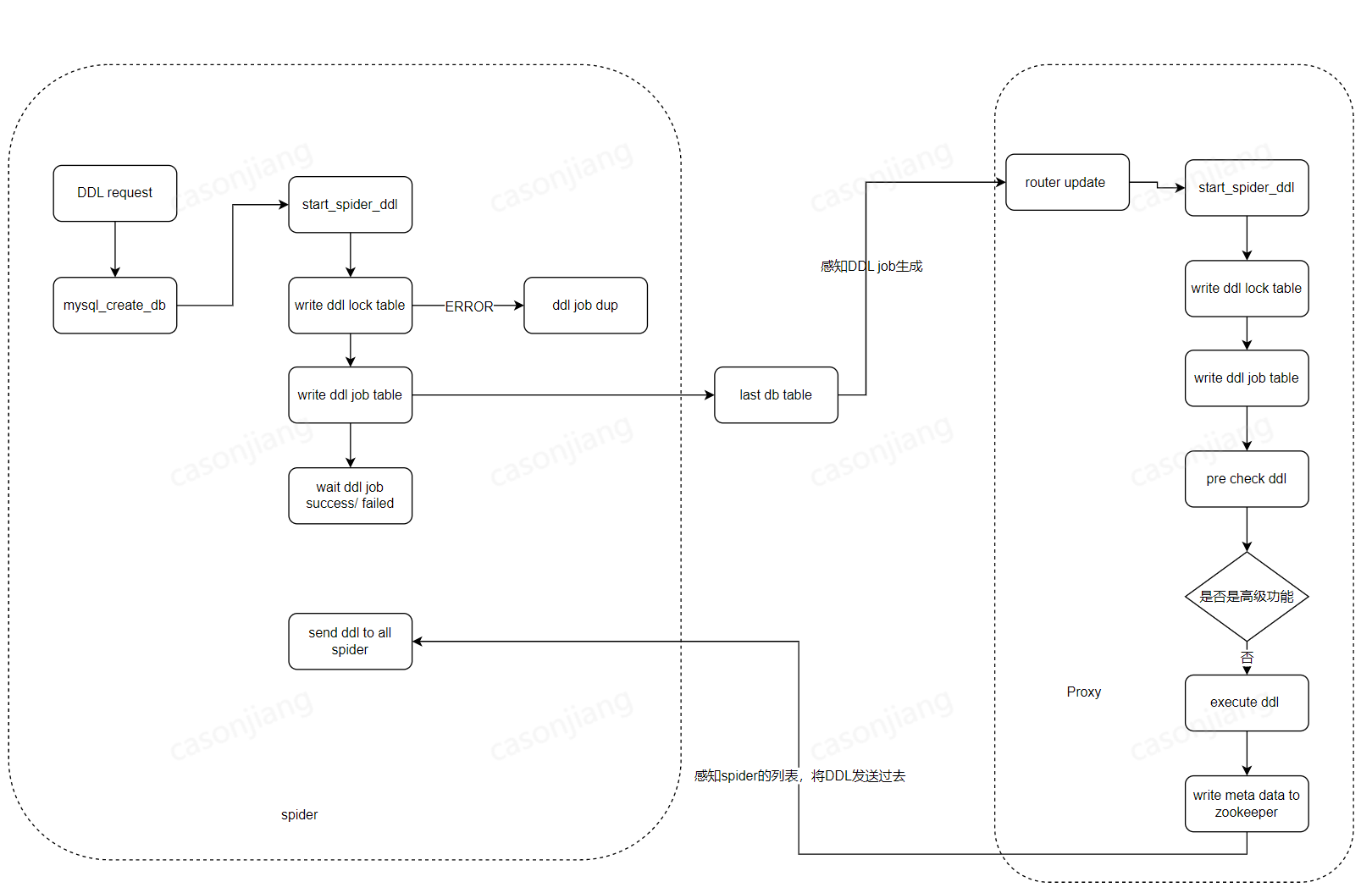
# 原理

## DDL

### DDL数据字典同步

spider的任务会写到DB中，让伴生的router update进程去执行。

说明：这里是基于TDSQL2.0的proxy基础上的设计（为了直接能够对接2.0的proxy），现在的实现方案修改为Spider写zk，然后其他Spider感知到DDL job后，在本地写元数据。



#### DDL流程

##### 入口函数

mysql\_create\_db

mysql\_alter\_db

mysql\_rm\_db

mysql\_alter\_table

mysql\_xxx

判断是否使用proxy的逻辑执行DDL，因为spider 需要执行业务的DDL，也需要执行其他spider发过来的更新元数据的SQL（使用hint的语法）。

使用db\_type判断spider引擎的表并且没有为hint的语句，需要proxy去执行。

bool mysql\_alter\_table(THD \*thd, const char \*new\_db, const char \*new\_name,

HA\_CREATE\_INFO \*create\_info,

TABLE\_LIST \*table\_list,

Alter\_info \*alter\_info) {

DBUG\_TRACE;

​

if (create\_info->db\_type->db\_type == 100

// 100 为spider 表，hint是创建元数据的SQL。

&& thd && thd->lex->opt\_hints\_global->find\_by\_name("ddl") == nullptr) {

// start proxy ddl

}

##### DDL job的任务交互数据

#### 设计方案

##### 语法和引擎支持

##### 权限问题

执行DDL的链接支持IP透传的功能，为了防止超级用户创建出来的表和高级功能，其他用户用不了的问题。

1、spider要支持ip透传的功能，才能进行鉴权。https://git.woa.com/TXSQL/MySQL-8.0/issues/3

2、spider的client要支持传递sha(password)的密码形式，因为spider 并不知道用户密码。下面是对应的文档。

native\_password\_authenticate

https://piaohua.github.io/post/mysql/20210404-mysql-pluggable-authentication/

3、鉴权的时候，需要将sha(password)保存到THD上，在DDL链接的时候使用。

##### spider的列表

1、上报zookeeper的版本信息，先上报为临时节点。

get /tdsqlzk/group\_\*\*/supervisor/proxys/register@proxy/proxy@9.30.\*.\*\_1234

{"set":"group\_1666752955\_433021689","single\_backend":0,"start\_time":"2023-02-24 21:50:32","version":"\"proxy-2.0.20-8-R751D001\""}

2、监听register@proxy目录的变化，delete/create的事件，将spider的ip\_port，版本，写入cache中，自定义一个cache结构。

3、提供api拉取全量的spider的ip port

##### 高级功能DDL元数据格式

该信息应该包含高级功能的定义，创建用户，是解决新安装的spider的，需要重放该高级功能到spier本地。

格式：{"type": function//，user、client ip，sql}

存储位置：存储到最后一个set的meta\_history\_table里面

##### Spider的DDL转发

spider根据router update需要的字段，decode成json字符串，并且写到lock table和ddl job table。这里需要注意需要跟proxy写入的字段一样。

create table/drop table任务json格式

rename的任务的json格式

alter table的任务的json格式

##### Spider的DDL任务通知

1. 如果是spider当前进程执行时，可以使用条件变量唤醒。

说明：proxy之前和router update是两个进程，使用文件来通知。

1. 如果是其他的spider的去执行DDL，则需要异步线程唤醒。

说明：proxy有个异步线程，定期扫描未完成任务的状态，通知。

##### 数据字典

### auto\_id

#### 整体设计

1、在table创建变更时创建变更CAutoID（port from proxy）

2、在获取和设置auto id的地方改为使用CAutoID

3、监听被CAutoID修改的alter\_seqid zk node对本地seq做alter/clean

#### DDL

auto id创建删除随同Schema/Table DDL进行修改:

AlterSchema(AlterSchema)

CreateTable(TdCreateTableDdl)

DropTable(TdCreateTableDdl)

RenameTable(TdRenameTableDdl)

在以上AlterDdlOperator实现中分别的对CAutoID进行创建和删除(或 rename)

#### DML

主要修改在Partition\_helper::get\_auto\_increment\_first\_field和Partition\_helper::set\_auto\_increment\_if\_higher中处理获取auto id和rebase auto id分区处调用CAutoID进行get和set操作。

#### ZK

主要监听/alter\_id处理auto id rebase逻辑在多个proxy的协作，CAutoID会触发修改：

Delete: clean seq清理CAutoID 中预cache id

Create/Update: alter seq修改start值

### sequence

#### 整体设计

1、spider本地支持创建seq（port txsql/mysql-8.0）

2、通过新的ddl operator在创建和删除seq时同步在所有spider创建删除spider本地seq（即目前seq定义信息以spider local的seq的形式存在）

3、因为各个spider都有sequence符号解析可以工作，也可以拿到当前执行的seq定义信息(e.g. step/ cache/ ..)，在Item\_func中切换使用CAutoID获取和修改seq

4、监听zk处理/alter\_seqid处理alter seq\_id操作

5、为了兼容2.0升级中间状态需要考虑写定义到"/rt"

6、后续转使用新的SP同步机制可靠同步到spider

https://iwiki.woa.com/pages/viewpage.action?pageId=4008231020

#### DDL

* SQLCOM\_CREATE\_SEQ时make\_create\_seq\_job将step inc等定义信息持久化
* SQLCOM\_DROP\_SEQ时复用make\_drop\_table\_job
* 增加AlterDdlOperator的新实现TdCreateSequenceDdl负责处理create /drop seq, 注册时指定SPIDER\_ROUTE，将 create/drop seq直接同步到所有spider( before flex's MR)

1）在 PreCheck中查local检查exist逻辑(目前外层parser语法其实未支持 if not exist option)，同时加了类似table的一致性检查

2）在ExecuteDdl中对所有spider执行/\*local\*/创建 seq

3）在WriteMetaDataTable中还是会写入或删除ZK

seq定义信息到路由rt路径(这个目前只有兼容性作用? 2.0 + 2.5 混部灰度?)

start信息到atler\_seqid路径

4）增加 AlterDdlOperator的新实现TdAlterSequenceDdl(TODO:)

* SQLCOM\_ALTER\_SEQ时支持alter sequence语法

#### DML

dml执行时， 如无意外本地有seq定义信息，可以被正常执行到表达式部分thd\_seq\_set\_val/thd\_seq\_next\_val时改为使用CAutoID获取和修改seq

#### ZK

主要监听/alter\_id处理auto id rebase逻辑在多个proxy的协作，CAutoID会触发修改：

Delete: clean seq清理CAutoID中预cache id

Create/Update: alter seq修改start值

## DML

## DQL

## DCL

### 用户管理

### 变量

#### Spider和DB的变量同步

##### 背景

因为spider 本身是基于Mysql的开发的，自身也有一整套参数。在spider 上执行show variables 会展示spider自身的参数设置，而不是DB的。

##### 方案

在spider 内部进行异步同步DB的变量，目前是从最后一个分片将变量拉出来，然后判断跟spider 是否一致，然后回放到spider中，然而并不是所有的变量都会进行同步。

新增三个参数控制变量的同步：

* tdsql\_sync\_vars\_list

默认为default或者空。会将DB的下面的变量同步default\_sync\_vars\_list 给spider。如果想要去除某个变量的同步，需要设置tdsql\_sync\_vars\_blacklist 黑名单去除即可。

static TdVector<TdString> default\_sync\_vars\_list = {

"transaction\_isolation",

"autocommit",

"sql\_select\_limit",

"character\_set\_results",

"character\_set\_client",

"character\_set\_connection",

"character\_set\_database",

"character\_set\_server",

"collation\_connection",

"collation\_database",

"collation\_server",

"default\_collation\_for\_utf8mb4",

"sql\_mode",

"max\_prepared\_stmt\_count",

"net\_write\_timeout",

"net\_read\_timeout",

"net\_buffer\_length",

"net\_retry\_count",

"sql\_require\_primary\_key",

"sql\_safe\_updates",

"low\_priority\_updates",

"group\_concat\_max\_len"};

当 tdsql\_sync\_vars\_list 为\* 或者all的时候，会同步db的变量到spider上面。

当 tdsql\_sync\_vars\_list 为var1;var2 时，会同步var1 var2加上default\_sync\_vars\_list 的变量。

* tdsql\_sync\_vars\_blacklist 黑名单机制，不会同步设置的变量。

默认为default 为空。会禁止下面default\_black\_sync\_vars\_list的变量进行同步

static TdVector<TdString> default\_black\_sync\_vars\_list = {

"innodb\_buffer\_pool\_size", "default\_storage\_engine", "log\_error\_verbosity"};

当 tdsql\_sync\_vars\_blacklist 为\* 或者all的时候，会禁止所有的变量进行同步。

当 tdsql\_sync\_vars\_blacklist 为val1;var2 的时候，会禁止default\_black\_sync\_vars\_list 加上var1 var2的变量同步。

* tdsql\_sync\_vars\_interval 同步间隔，默认为30s。单位为s。

**注意：**

此功能只在tdsql\_mode为tdsql的模式下生效，此功能合并到主分支以后，会导致部分变量没有办法设置，因为后台会异步同步db的变量，可以通过赤兔修改db的变量，让spider进行生效。只会影响global 级别，session 级别的并不受影响。

##### TODO

现在只是将部分变量同步给了spider。在spider上执行show variables 依旧会读取spider的变量，也会有部分变量不一致。

* 需要区分出来spider的变量和DB的变量设置。
* spider上执行show variables 可能要发送到db去了。
* 支持类似/\*proxy\*/show variables 查询spider内部变量的形式。

#### Spider维护set session变量

##### 背景

常见的set names 等等一些session 变量在设置以后，只在spider上面生效，但是对于跟后端的链接未生效。

期望在spider设置的session变量，能够在后端db连接上同样生效；

##### 方案

THD上维护一个map，记录客户端设置的session变量；

spider的后端connection上也维护一个map，记录session变量；

spider接受到前端的set session var命令，会更新THD上的变量map同时map的version加1；

每次后端connection执行sql之前，比较下thd的session变量和connection的version是否一致，如果不一致就在后端连接上执行sql重新设置下session变量；

**新增黑名单变量;**

tdsql\_set\_var\_blacklist

session变量黑名单，黑名单内的变量，在spider进行set的时候，不会透传到db上。

格式是: var1,var2,var3

默认是空，允许透传所有变量；

**注意事项：**

如果黑名单是空，默认透传所有session变量；

##### 实现

在Server层代码set\_var.cc、set\_var.h中增加set 变量的相关逻辑，这里主要是将变量保存到thd中。

在Spider层代码spd\_db\_conn.cc中增加Spider->DB变量设置的操作，将thd存储的变量传递给DB层。

##### TODO

1、spider代码里面自己维护了几个常见的session变量，如字符集、timezone、隔离级别，这部分代码可以去掉。

2、set session变量报错，可能是这个变量只在spider存在，db上不支持这个变量。

需要配置一下 tdsql\_set\_var\_blacklist ，把这个变量屏蔽掉，不让透传到db；

## 数据字典

### Spider数据字典

#### 表类型

catalogs: Catalog information.

character\_sets: Information about available character sets.

check\_constraints: Information about CHECK constraints defined on tables. See Section 13.1.20.6, “CHECK Constraints”.

collations: Information about collations for each character set.

column\_statistics: Histogram statistics for column values. See Section 8.9.6, “Optimizer Statistics”.

column\_type\_elements: Information about types used by columns.

columns: Information about columns in tables.

dd\_properties: A table that identifies data dictionary properties, such as its version. The server uses this to determine whether the data dictionary must be upgraded to a newer version.

events: Information about Event Scheduler events. See Section 25.4, “Using the Event Scheduler”. If the server is started with the --skip-grant-tables option, the event scheduler is disabled and events registered in the table do not run. See Section 25.4.2, “Event Scheduler Configuration”.

foreign\_keys, foreign\_key\_column\_usage: Information about foreign keys.

index\_column\_usage: Information about columns used by indexes.

index\_partitions: Information about partitions used by indexes.

index\_stats: Used to store dynamic index statistics generated when ANALYZE TABLE is executed.

indexes: Information about table indexes.

innodb\_ddl\_log: Stores DDL logs for crash-safe DDL operations.

parameter\_type\_elements: Information about stored procedure and function parameters, and about return values for stored functions.

parameters: Information about stored procedures and functions. See Section 25.2, “Using Stored Routines”.

resource\_groups: Information about resource groups. See Section 5.1.16, “Resource Groups”.

routines: Information about stored procedures and functions. See Section 25.2, “Using Stored Routines”.

schemata: Information about schemata. In MySQL, a schema is a database, so this table provides information about databases.

st\_spatial\_reference\_systems: Information about available spatial reference systems for spatial data.

table\_partition\_values: Information about values used by table partitions.

table\_partitions: Information about partitions used by tables.

table\_stats: Information about dynamic table statistics generated when ANALYZE TABLE is executed.

tables: Information about tables in databases.

tablespace\_files: Information about files used by tablespaces.

tablespaces: Information about active tablespaces.

triggers: Information about triggers.

view\_routine\_usage: Information about dependencies between views and stored functions used by them.

view\_table\_usage: Used to track dependencies between views and their underlying tables.

#### 查看数据字典表

需要debug包的mysql。

SET SESSION debug='+d,skip\_dd\_table\_access\_check';

select \* from mysql.table\_partitions;

设置skip\_dd\_table\_access\_check可以访问元数据，需要更新到txsql，更新到后端db。

MySQL INFORMATION\_SCHEMA库提供了对数据局元数据、统计信息、以及有关MySQL server的访问信息（例如：数据库名或表名，字段的数据类型和访问权限等）。

#### 存储

目前所有的元数据表使用innodb表存储。

#### 作用

数据字典信息内容就包括表结构、数据库名或表名、字段的数据类型、视图、索引、表字段信息、存储过程、触发器等内容。在访问表的时，会生成table，table shard的等等数据结构。

### 数据字典同步

## 元数据同步

### 基本表

#### 表元数据回放

#### 表元数据一致性校验

目的：主要分析Spider中表的字段类型和DB的字段类型发生不一致后，会出现的异常情况。如Spider对于table t1的类型是varchar，但是对于DB而言，table t1的类型是int。

#### 表元数据补偿

表元数据不一致的情况下，需要补偿。

### 高级特性元数据回放

# 操作

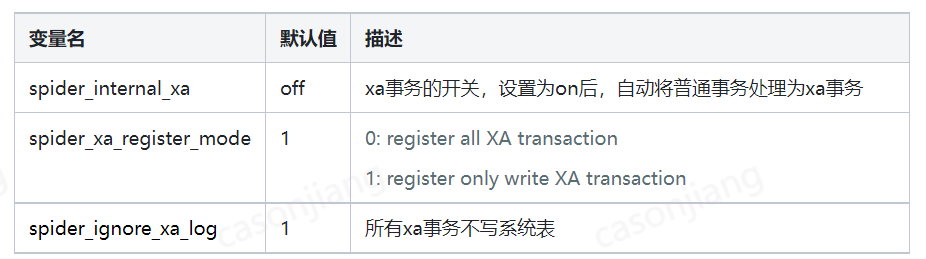
# 事务与并发

和2.0的proxy类似，spider分布式事务的实现机制用的也是XA事务。但是实现细节上有些差异，比如事务id的格式、全局事务状态的存储。

为了能够和2.0的保持兼容，需要对spider进行改造，让spider能够无缝衔接现有2.0的架构。

## 现状

**相关变量：**



**相关的表：**



**原生spider的xa存在的问题：**

1、xa事务记录的日志信息太多了。

规划：这部分代码逻辑全部删除掉，参照2.0的设计，仅仅在xa prepare完成后，写入gtid\_log\_t;

2、xa的日志spider原来是存储在本地的，这个当spider挂掉后，无法保证事务的一致性；

规划：参照2.0，gtid\_log\_t是随机存放在后端某个set中。

**XA事务id**

spider的xa事务id直接使用的是xid，trx->xid，这个id是server层生成的。

## XA事务

### XA事务流程

事务流程按照MySQL+Spider的实现：

#### 事务开启

* 客户端发送begin或者autocommit = 1
* 在向远端转发sql时，spider根据事务状态和涉及后端的分片数目决定是否开启xa事务，没有显示begin和只是单分片不需要开启xa事务，需要则生成xa start + gtid发送给远端
* 然后才是用户的业务sql

#### 事务提交

* 客户端显式commit，或者隐式提交（autocommit=1等）
* 当判断到当前有开启xa事务时，如果发现只是单分片，则进行xa one phase提交
* 如果跨分片则先进行xa end + xa prepare，重构主要就是删除了这些持久化xa状态到本地的系统表的逻辑，改为用2.0写到远端存储节点的实现

#### 事务回滚

* 客户显式rollback, 或者隐式rollback
* 这儿主要关注的是跨分片的xa事务，不跨分片不需要spider额外处理，如果需要则进行xa end + xa rollback

### XA事务id的生成

按照2.0的现状，在开始事务的时候，就要生成xa事务id

proxy的现状：

Global\_txn \*GTM::start\_txn(Global\_txn::Iso\_level isolevel,

network\_mysqld\_con \*client\_con,

bool internal) {

Global\_txn \*gtxn = NULL;

uint64\_t seqno = Seq\_mgr::instance()->get\_next\_seqno();

​

Gtid gtid(internal, m\_gwid, m\_randno, seqno, current\_hour(), Invalid\_partition\_no);

gtxn = new Global\_txn(gtid, isolevel);

return gtxn;

}

2.0的xa事务id的格式，m\_partition\_no是在：



目前spider也参照这个写的gtid：

m\_spider\_gtid = std::unique\_ptr<tdsql::Gtid>(new tdsql::Gtid(true, server\_id, rand\_no, query\_id, (uint32\_t)now, partition\_no));

### gtid\_log\_t的写入

#### 账号

需要使用系统账户去写入t\_gtid\_log记录，普通账户不一定有权限

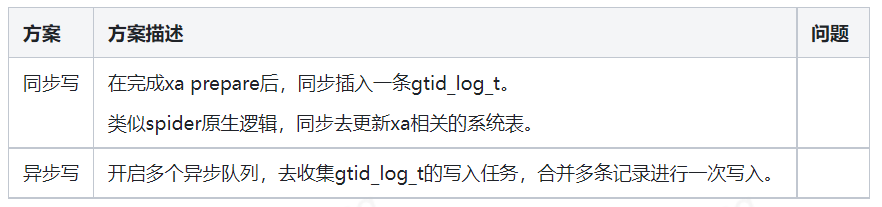
具体实现：

直接配置文件中读取后端set的系统账户+密码。

说明：在TDSQL2.0并t\_gtid\_log并不需要存储系统账户信息。

#### 异步/同步写入

方案选型：将任务插入异步队列后，同步等待任务完成。



xa事务应该在所有子事务prepare成功之后，写入一条t\_gtid\_log，写入成功之后再进行xa commit。

##### 同步写

简单的逻辑是，直接串行同步写入gtid\_log\_t；

但是这样存在问题：所有的xa事务都会额外多一次与后端db的交互。

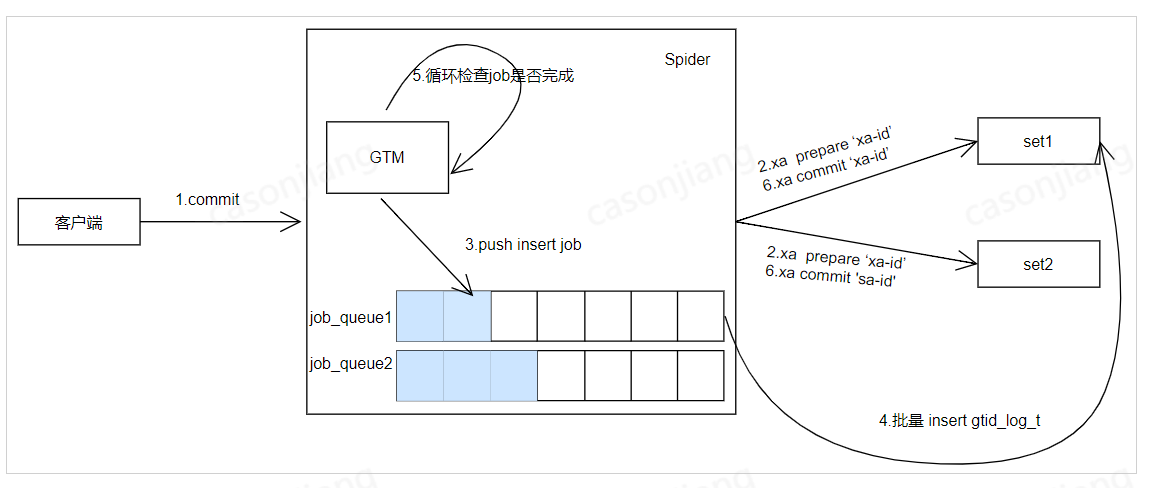
##### 异步写

如果异步写入，由一个单独的线程去批量刷入gtid\_log\_t，可以减少db交互次数。

这样在并发压力较高时，可以增加系统的吞吐能力，但是会增加单个事务的处理时间。

###### 2.0异步/batch写入

写入流程：



用户线程xa prepare后将对应的gtid投递到后端的写gtid工作队列，**根据gtid的part\_no哈希到对应的远端set**。

工作队列一次将收集的gtid拼接成一个大的sql写入到对应的远端set的xa.gtid\_log\_t系统表，完成后通过job注册的callback处理返回值和通知用户线程继续xa commit。

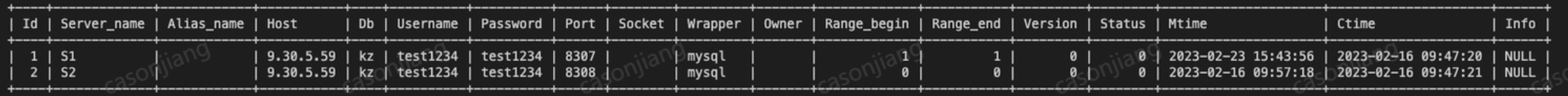
###### 2.5异步/batch写入

复用2.0的batch写gtid逻辑，将xa\_xx.cc以源码的形式引入到2.5中通过编译，同时进行相应的调整：

1、partition\_no维护在partition的handler层，用于生成xa start的gtid，gtid其他字段参照2.0的定义生成

2、2.5的路由信息进行job的分发和远端连接的读写

2.5当前的路由信息表：



xa prepare '53b3e7b-d4-16-63f73e98-1'这个part\_no = 1根据路由信息range\_begin && range\_end将被分发到写入set1的队列中。

说明：与TDSQL2.0类似，这里使用part\_no进行分发set的计算。

1. 重新实现job回调callback逻辑，将之前的邮箱通知用户线程修改为bthread\_cond

### 代码

#### Server

1. 设置分区号（sql/partitioning/partition\_base.cc）
2. MySQL Server层XA事务

Sql\_cmd\_xa\_recover::execute

Sql\_cmd\_xa\_init::check\_xa\_init\_privilege：新增接口

Sql\_cmd\_xa\_init::execute：新增接口

Sql\_cmd\_gtid\_state::check\_gtid\_state\_privilege：新增接口

Sql\_cmd\_gtid\_state::execute：新增接口

Sql\_cmd\_xa\_boost::check\_xa\_boost\_privilege：新增接口

Sql\_cmd\_xa\_boost::execute：新增接口

Sql\_cmd\_xa\_lockwait::check\_xa\_lockwait\_privilege：新增接口

Sql\_cmd\_xa\_lockwait::execute：新增接口

Sql\_cmd\_xa\_show::check\_xa\_show\_privilege：新增接口

Sql\_cmd\_xa\_show::execute

1. TDSQL分布式事务

xa\_deadlock.cc：死锁检测

xa\_gt.cc：GTID操作接口

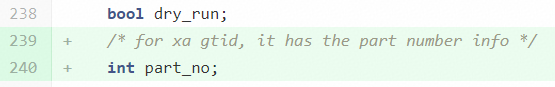
xa\_health.cc：锁超时

xa\_log.cc：写日志（解决2PC弊端）

xa\_tm.cc：事务控制接口

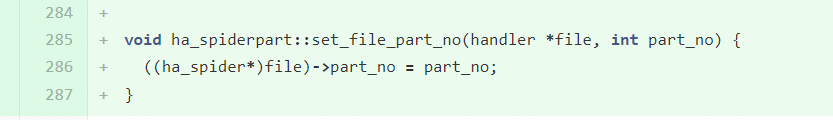
#### Spider

1. 初始化ha\_spider（ha\_spider.cc），存储分区号，这个是与GTID相关的（最后一位即为part\_no，默认第一个）

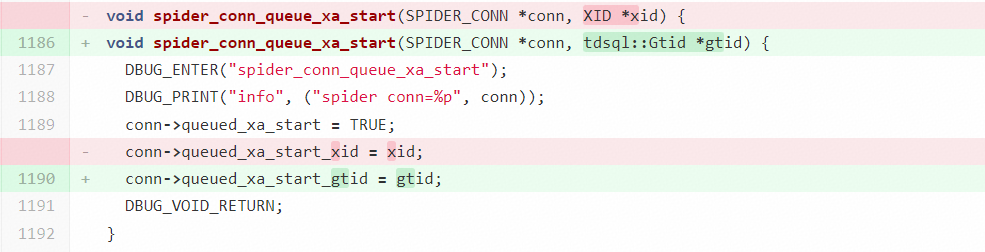


在spider\_internal\_start\_trx接口中增加参数part\_no，用于执行xa start的时候生成GTID使用。

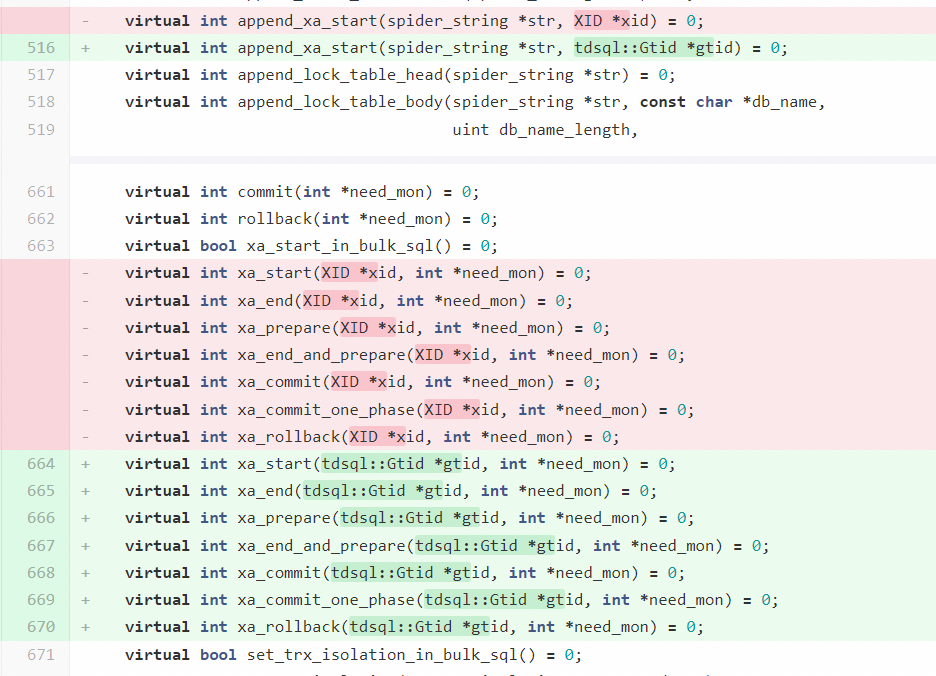
1. 增加设置Spider分区号的接口（ha\_spiderpart.cc）



1. 修改Spider connection相关属性信息（spd\_conn.cc），修改xid->gtid：

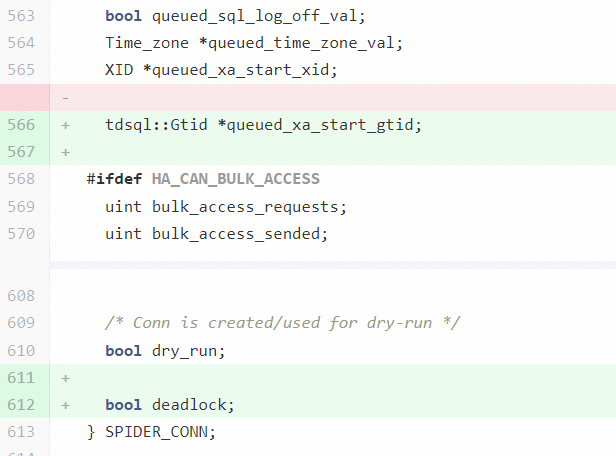


同时修改XID相关接口（spd\_db\_include.h）：

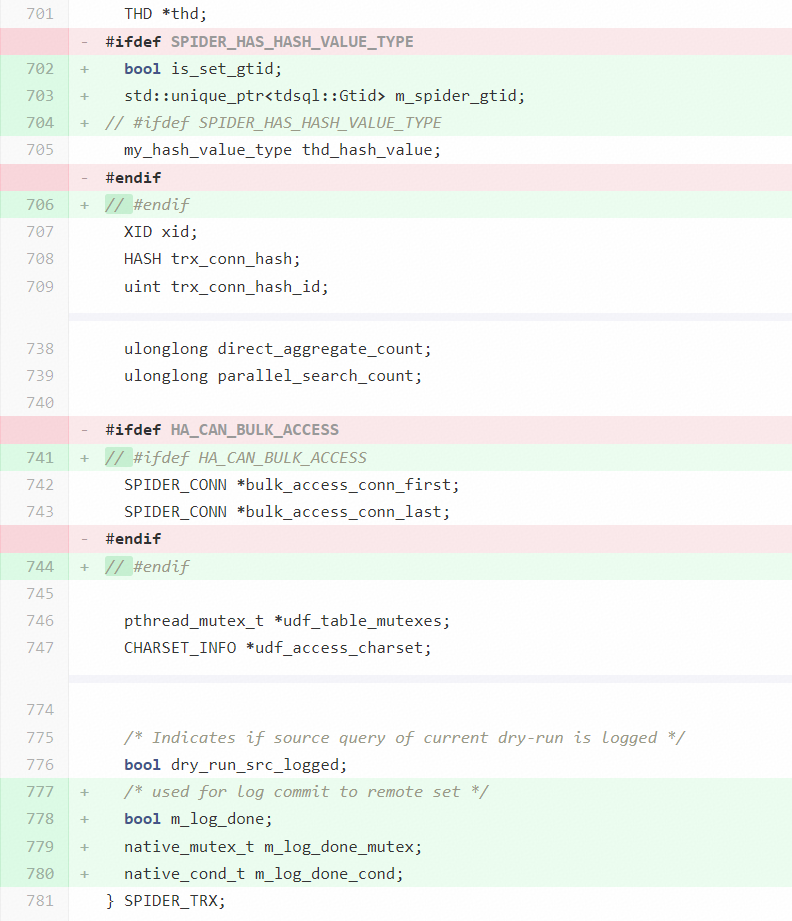


修改是否已经设置gts的标识（spd\_include.h）：

1. 修改Spider连接结构体st\_spider\_conn



1. 修改Spider事务结构体st\_spider\_transaction



1. 修改与后端DB连接的事务接口：

spider\_db\_conn\_queue\_action（spd\_db\_conn.cc）：

spider\_db\_errorno（spd\_db\_conn.cc）

spider\_db\_append\_xid\_str -> spider\_db\_append\_gtid\_str

spider\_db\_xa\_end

-> spider\_db\_mysql::xa\_end（spd\_db\_mysql.cc）：处理gts，下发DB执行

spider\_db\_xa\_prepare

-> spider\_db\_mysql::xa\_prepare（spd\_db\_mysql.cc）

spider\_db\_xa\_end\_and\_prepare

-> spider\_db\_mysql::xa\_end\_and\_prepare

spider\_db\_xa\_commit

-> spider\_db\_mysql::xa\_commit

spider\_db\_xa\_commit\_one\_phase

-> spider\_db\_mysql::xa\_commit\_one\_phase

spider\_db\_xa\_rollback

-> spider\_db\_mysql::xa\_rollback

1. 事务接口（spd\_trx.cc）

spider\_internal\_xa\_prepare：

spider\_commit：

spider\_rollback：

## TDSQL全局一致性读

### 计算层

新增参数：

mc\_enabled // 是否创建meta cluster后台线程获取gts

meta\_cluster\_iplist // meta cluster的ip端口信息

eg: x.x.x.x:12345,x.x.x.x:123456 支持域名方式

meta\_cluster\_thread\_nums // 后台线程数

meta\_cluster\_timeout // 获取超时时间，单位秒

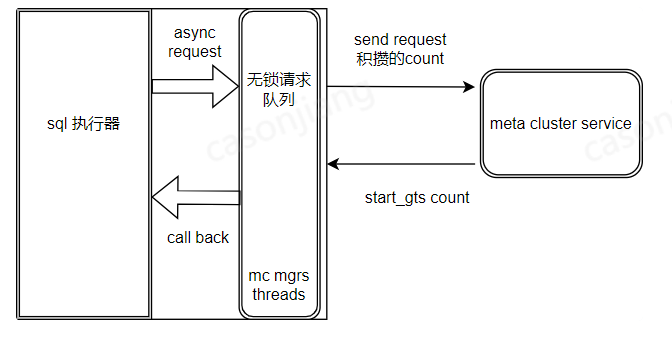
consistent\_read\_optimize // 是否开启全局一致性读

计算层要支持全局一致性读，提供给存储层两个gts, 一个是标记数据版本的信息的commit gts，另一个就是读取数据的read view gts。这个又根据隔离级别 有所不同：

* RR 只在xa start (事务开始的时候设置一个）
* RC 事务中每个查询都需要

在计算层逻辑中增加读取gts的网络通信时间，如果不进行优化，极大的影响性能

2.0 proxy采用了异步batch的方式进行申请：



### 方案

主要分3步：添加攒批mc客户端，**rr隔离级别在xa start和xa commit上加上gts，rc隔离级别在查询时加gts**。

#### 移植2.0 mc\_mgrs

用于攒批gts请求, 主要就是mc\_gts.cc/h依赖的无锁数据结构以头文件源码引入。

#### RR隔离级别

需要在事务开始xa start和xa commit上带上gts

在SPIDER\_TRX中增加了两个gts（标记数据版本信息的commit gts，读取数据的read view gts），和对应的condition（用于回调通知）

/\* used for consistent read \*/

uint64\_t m\_start\_gts;

native\_mutex\_t m\_start\_gts\_mutex;

native\_cond\_t m\_start\_gts\_cond;

uint64\_t m\_end\_gts;

native\_mutex\_t m\_end\_gts\_mutex;

native\_cond\_t m\_end\_gts\_cond;

##### xa start

判断需要start gts的条件：开启一致性读和事务隔离级别为RR。

在开启xa事务的时候申请（async\_request)，在发送xa start时等待并拼接对应的sql发送到存储节点。

##### xa commit

条件：开启一致性读都需要申请。

在所有分片xa prepare完成后写gtid\_log前申请，（目前未把commit gts写入gtid log，考虑异步update?)，这样申请gts和写gtid log可以并行。

#### RC隔离级别

在thd的lex中新增变量记录查询需要的gts：

// global consistent read

uint64\_t query\_gts;

native\_mutex\_t query\_gts\_mutex;

native\_cond\_t query\_gts\_cond;

##### xa commit

和RR级别一样。

##### select

对于每个用户的查询请求判断是否需要gts：开启了全局一致性读和事务隔离级别是RC，是select请求则申请gts。

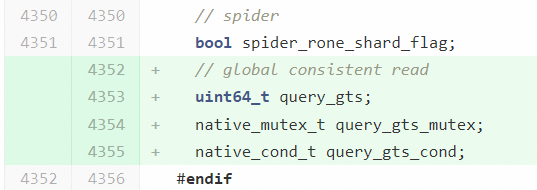
在发送查询请求时，判断是否需要查询的gts, 如果需要等待gts 获取并拼接到sql末尾发送给存储执行。

说明：这里是将gts（读取的版本信息）发送到DB（select … from tdsql\_withgts xxx），由DB负责去判断读取具体版本的数据。

### 代码

#### Server

1. 在lex中增加一致性读gts的变量（sql\_lex.h、sql\_lex.cc）：

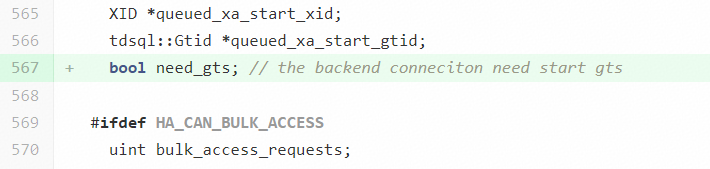


1. 设置初始化MC的接口（sql/tdsql/xa/mc\_gts.cc/ init\_meta\_cluster），与MC建立连接
2. 在mysqld启动流程中初始化MC（sql/mysqld.cc）

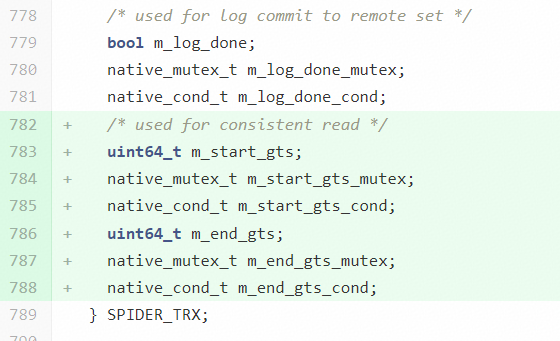
#### Spider

1. 在Spider的中增加一致性读gts的变量（storage/spider/spd\_include.h）

st\_spider\_conn连接类增加是否需要gts的标志：

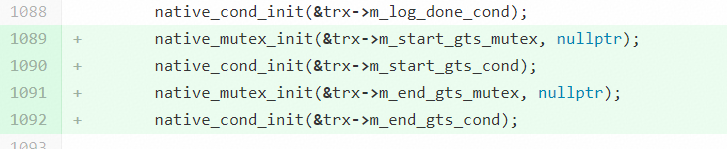


st\_spider\_transaction事务相关类增加gts变量：

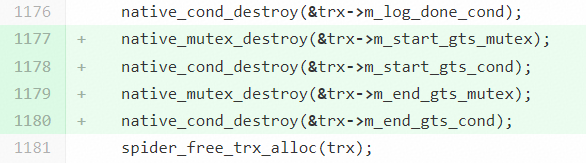


1. 修改Spider原生的事务接口，增加gts的操作

spider\_get\_trx：获取事务结构体，完成Spider一致性读相关变量初始化



spider\_free\_trx：释放事务，完成Spider一致性读相关变量的清理

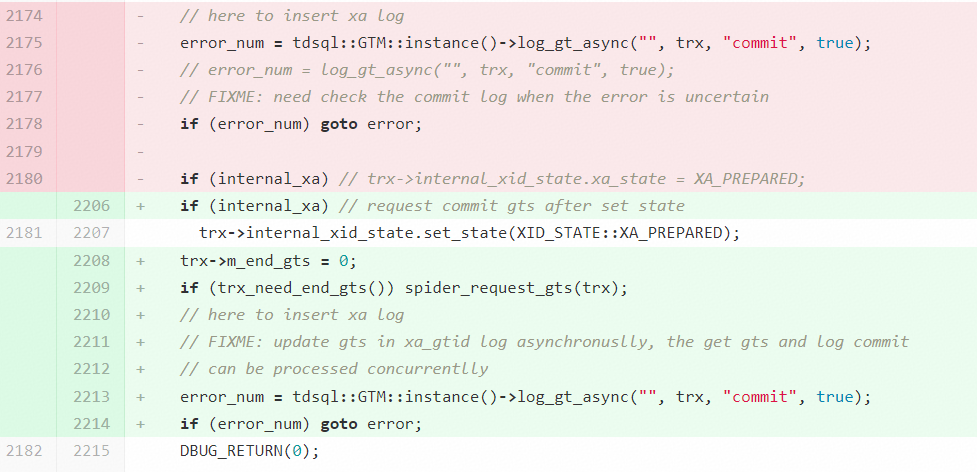


trx\_need\_start\_gts：新增接口，判断是否开启gts

trx\_need\_end\_gts：新增接口，判断是否结束gts

spider\_internal\_xa\_prepare：XA PREAPRE接口，是否需要申请gts

-> spider\_request\_gts：新增接口，异步获取gts



spider\_internal\_start\_trx（storage/spider/spd\_trx.cc）：XA START接口

->spider\_request\_gts：在xa start的时候申请gts

-> spider\_request\_gts：新增接口，异步获取gts

->spider\_conn\_queue\_xa\_start（storage/spider/spd\_conn.cc）：设置xa start标志

1. 设置select接口操作：

spider\_db\_conn\_queue\_action：与DB建链的时候设置一些控制sql

-> spider\_db\_mysql\_util::append\_xa\_start（storage/spider/spd\_db\_mysql.cc）：具体处理如果需要gts的情况下，xa start

spider\_db\_query：执行select操作的时候，是否携带gts

spider\_db\_errorno：DB返回错误码处理

### 问题

RC隔离级别因为每个查询都申请了gts，性能会有下降。

# 数据分布

# 复制/一致性

# 导入导出

# 备份恢复

# 兼容性

## MySQL兼容性

## Oracle兼容性

### 开启Oracle模式

#### DB支持ORACLE模式

* 全局开启ORACLE的模式

直接设置DB的全局oracle的模式。如在所有的set上，执行set global sql\_mode=oracle; 或者在赤兔上或者使用agent的工具，设置所有的db。（赤兔上目前sql\_mode应该不支持oracl的模式，需要使用agent的工具设置）

spider 会默认从最后一个set读取当前db是否支持oracle模式，然后设置db的oracle 以后，会将db的sql\_mode同步回来。

设置完DB的oracle 以后，在spider 节点上执行show global variables like 'sql\_mode'; 存在30s的延迟时间。

* session 开启ORACLE的模式

直接在spider 上执行set sql\_mode=oracle；即可，sql\_mode的oracle的设置会同步到DB去，保证全链路都是oracle的模式。

#### DB不支持oracle的模式

* 全局开启ORACLE的模式

由于DB不支持oracle 模式，需要在spider的各个节点通过jmysql.sh ${port} 设置全局的oracle 模式。spider的运行目录的install目录中jmysql.sh 登录到spider。执行set global sql\_mode=oracle 开启全局模式。

* session 开启ORACLE的模式

直接在spider 上执行set sql\_mode=oracle；由于DB 不支持oracle的模式，改sql\_mode不会在DB生效。只在spider生效。

### PL/SQL异常处理

### 语法调整

### 内置函数

### PL/SQL创建存储过程

### PL/SQL创建存储函数

### 新增类型

### NULL兼容

### show create table

### Prepared Statements

### begin

### PL/SQL游标

### PL/SQL循环

### PL/SQL变量

# 扩展性

# 高并发

## 协程框架

## 连接优化

### 连接保持

### 连接池改造

#### 现状

目前spider使用连接池维护后端的链接，每次请求时，从链接池中取一个链接进行使用。

#### 问题

1、在启动的时候，使用内部用户链接到后端，会导致mysql的权限控制放到计算节点进行，同时存在一个超级用户在计算节点中。

2、链接池无法动态扩展。

3、事务场景中，需要更多的连接进行处理。

说明：使用连接池可以提高并发效率，但是如果是增加了鉴权、动态扩展、事务这些要求，那么就会造成很多问题，反而不如简单非池化的好处理。

#### 方案

1、透传前端连接的IP、sha(password)和port；

2、去掉连接池，每个前端连接独立拥有自己的后端db连接；

##### 透传前端IP方案设计

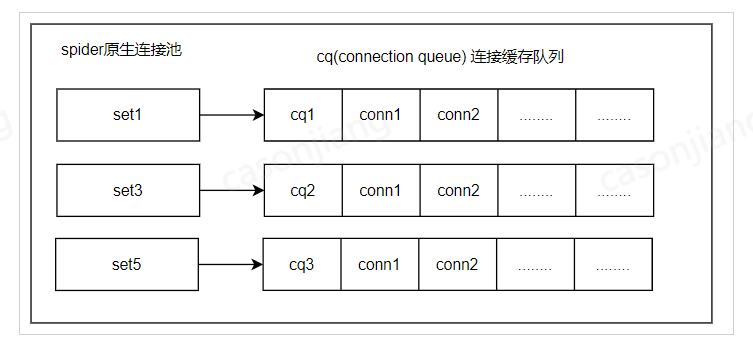
主要解决问题1，spider原来不论前端是什么用户访问，都是使用mysql.servers中配置的超级用户去访问后端。

这样实现确实很简单，但是缺点也很明显：

* 需要在spider中配置一个超级系统用户，这个用户可以访问所有后端db；
* 权限没有区分开，前端用户连接是一个用户，后端访问又是超级用户；

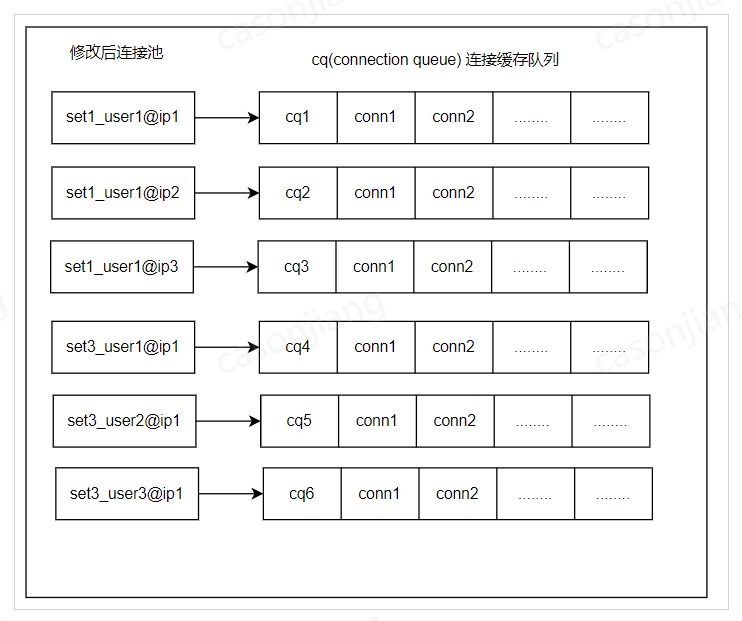
我们希望能使用前端连接进来的用户，继续去访问后端db，这样能做到统一用户，所以需要透传前端用户到后端。

当前现状，spider有set维度的连接池：



前端用户访问对应的set，会从set对应的connection queue中（pop）取出一个连接，用完了之后，再put放回去。

**要做到用户维度的连接池，我们需要给conn再加一个标志，表示是某个前端用户的连接**。



**主要改动：**

原生spider的连接池是一个hash表，hashkey的格式是“0setname”，需要修改为“0setname\_user@ip”.

**主要代码：**

1、连接池

SPIDER\_CONN\_POOL spd\_connect\_pools;

只有一个连接池，所有spider的表共用这一个；

2、一个事务需要多次访问同一个set，怎么保证每次用的是同一个链接? this->conns[link\_idx] = spider\_get\_conn()

第一次获得后端set的连接之后，就会记录下来，后面每次请求会继续使用这个连接。

3、一个事务涉及到一个set上的多个分区，怎么保证多次访问这多个分区，使用的是同一个链接？

my\_hash\_insert(&trx->trx\_conn\_hash, (uchar \*)conn)

同一个事务使用的连接，会和trx→trx\_conn\_hash进行绑定，多个分区虽然会开启多个ha\_spider，但是每个ha\_spider对应到后端同一个set会是同一个链接。

**遗留问题：**

1、原生mtr中，基本都是用的root@localhost这种方式去连接，改成透传的话，连接后端db就会失败。

所以为了兼容mtr，在代码中加了个判断，如果是tdsql\_mtr\_mode=1，代表是mtr模式，还是使用原来的spider连接池方式（按照set维度去拿连接，也不用透传ip）;

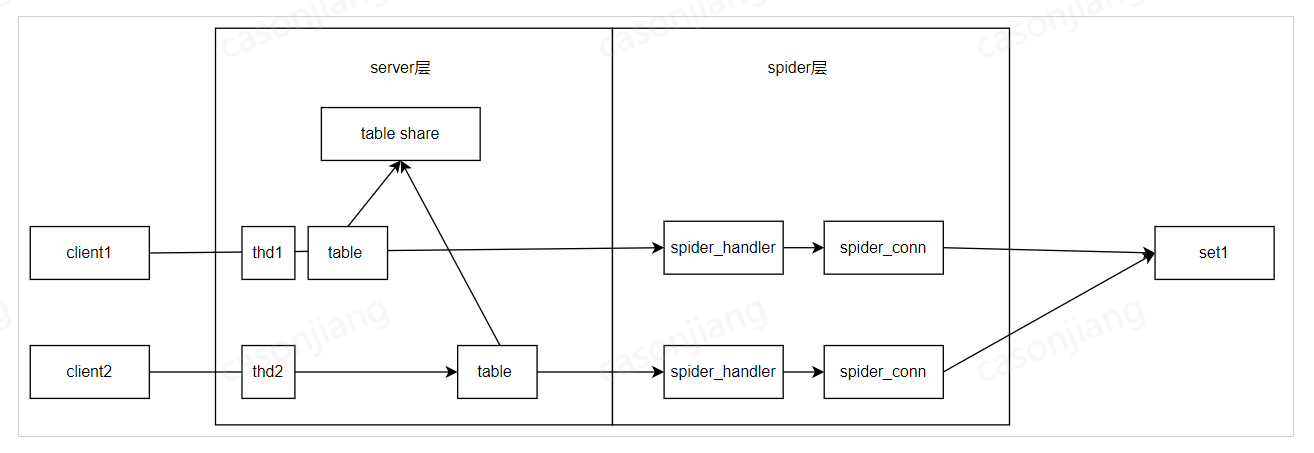
todo：这部分代码逻辑，完全是为了跑mtr。正常做法应该是把mtr修改一遍，用正规的用户去连接spider和后端db；

2、jmysql脚本使用受限不可以去访问后端db的数据了。jmysql使用的是agent@localhost用户，改成透传后，这个用户是不可以访问后端db的。

todo：后续应该是要替换成new\_kp用户，保持jmysql的超级用户功能。

##### 去掉连接池的方案设计

原生spider的connection关系



* table→file对应spider的handler接口；
* 每一次执行sql，都有open/close table操作，对应从table\_cache中取出table和归还table；
* table\_cache是全局的，默认配置的是32个instance；
* 所以事务中的sql，即使访问同一张表，获取的handler不一定是同一个；
* spider并没有把conn绑定到handler上，而是绑定到了trx上;

**方案设计：**

去掉spider的连接池，将connection绑定到thd上

1、创建连接后，需要将conn放入thd中

2、获取连接时，优先从thd中获取，如果没有就新建

3、thd析构的时候，需要析构conn

4、连接异常的时候，free\_conn的同时，需要将conn从thd中移除

**新增参数：**

* tdsql\_bind\_conn\_to\_thd

控制是否将connection绑定到thd，默认为true；

false：使用spider的原生连接池的模式，一个事务执行结束，thd将connection归还到spider的连接池；

# 高可用

# 数据压缩

# 数据迁移