全局表一致性的定时检测的实现

根据 Leader-us 的要求：

“增加mycat新任务，全局表定义中，需要有一个时间戳字段，每次记录的update,insert，确保 时间字段赋值，并且mycat增加定时检测逻辑，检测记录总量，以及最新时间戳的匹配，简单有效的发现全局表不一致的问题。/ 测试修复类 / 1.5&2.0 /12.9 /leader-us”

全局表一致性定时检测主要分为两个部分：

1. SQL拦截部分

主要实现对所有全局表中记录进行修改的语句进行拦截，比如：

ServerParse.INSERT，

ServerParse.UPDATE，

ServerParse.REPLACE(mycat-server不支持)

对所有对全局表的 insert, update操作进行拦截，首先判断该全局表是否存在一个记录时间戳的内部列\_mycat\_op\_time：

public class GlobalTableUtil{

 /\*\* 全局表 保存修改时间戳的字段名，用于全局表一致性检查 \*/

 public static final String GLOBAL\_TABLE\_MYCAT\_COLUMN = "\_mycat\_op\_time";

如果不存在，输出警告，哪个db的哪个全局表没有内部列：

if(innerColumnNotExist.size() > 0){

 for(SQLQueryResult<Map<String, String>> map : innerColumnNotExist){

 if(tableName.equalsIgnoreCase(map.getTableName())){

 StringBuilder warnStr = new StringBuilder();

if(map != null) warnStr.append(map.getDataNode()).append(".");

 warnStr.append(tableName).append(" inner column: ") .append(GlobalTableUtil.GLOBAL\_TABLE\_MYCAT\_COLUMN)

 .append(" is not exist.");

 LOGGER.warn(warnStr.toString());

 return sql;

 }

 }

}

然后返回原始sql. 不需要进行拦截。

如果存在一个记录时间戳的内部列，那么对该 insert或者update语句进行SQL拦截修改：

 if(sqlType == ServerParse.INSERT){

 sql = convertInsertSQL(sql, tableName);

 }

 if(sqlType == ServerParse.UPDATE){

 sql = convertUpdateSQL(sql, tableName);

 }

1.1 insert语句的拦截逻辑

对所有对全局表进行insert的sql语句，进行改写，比如下面的user是全局表：

insert into user(id,name)

valueS(1111,'dig'),

(1111, 'dig'),

(1111,'dig') ,

(1111,'dig');

会被改写成：

insert into user(id,name, \_mycat\_op\_time)

valueS(1111,'dig', 1450423751170),

(1111, 'dig', 1450423751170),

 (1111,'dig', 1450423751170) ,

(1111,'dig', 1450423751170);

其中\_mycat\_op\_time 是内部列的名称：

public static final String GLOBAL\_TABLE\_MYCAT\_COLUMN = "\_mycat\_op\_time";

而1450423751170 是在插入时在 mycat-server上生成的一个时间戳对应的long整数(对应到数据库是bigint)。然后该语句发送给所有db在其全局表中进行插入。

如果insert语句自带了内部列\_mycat\_op\_time，比如：

insert into user(id,name, \_mycat\_op\_time)

valueS(1111,'dig',13545);

那么会输出警告，并且也进行拦截改写成如下形式：

insert into user(id,name, \_mycat\_op\_time)

valueS(1111,'dig', 1450423751170)；

然后发送给所有db在其全局表中进行插入。

对mycat-server不支持的sql语句，本拦截器，不进行任何操作，直接返回原始sql。如果在拦截过程中发生任何异常，也返回原始sql语句，不进行任何修改操作。保证该拦截不会影响系统原有的健壮性。

1.2 update语句的拦截逻辑

Update语句的拦截逻辑和insert语句原理是相似的。也是判断是否有内部列。

如果没有输出警告信息，如果有则进行拦截。

对全局表 user 的如下update:

update user set name='dddd',pwd='aaa'

 where id=2

会被改写成：

update user set name='dddd',pwd='aaa', \_mycat\_op\_time=1450423751170

where id=2

如果原始sql带有\_mycat\_op\_time 那么进行警告，然后替换它的值，比如：

update user set name='dddd',pwd='aaa', \_mycat\_op\_time=1111

where id=2;

会被改写成：

update user set name='dddd',pwd='aaa', \_mycat\_op\_time=1450423751170

where id=2;

然后将语句发送给所有的全局表进行执行。

这样的话，如果有哪个表上的insert,update执行失败，那么内部列\_mycat\_op\_time 的最大值，以及全局表的记录总数就会不一致。Delete语句也一样，只是无需拦截。下面的检查机制就是根据这个原理来操作的。

1. 一致性的定时检测

在MycatServer的startup中引入一个定时检查任务：

timer.schedule(glableTableConsistencyCheck(), 0L, 1000 \* 1000L);

 // 全局表一致性检查任务

 private TimerTask glableTableConsistencyCheck() {

 return new TimerTask() {

 @Override

 public void run() {

 timerExecutor.execute(new Runnable() {

 @Override

 public void run() {

 GlobalTableUtil.consistencyCheck();

 }

 });

 }

 };

其实现在GlobalTableUtil 类中：

该类首先获得所有的全局表：

static {

getGlobalTable(); // 初始化 globalTableMap

}

其实现，参见代码。

GlobalTableUtil.consistencyCheck() 的实现，主要思路是，首先根据所有的全局表，找到对应的PhysicalDBNode，然后找到对应的PhysicalDatasource，然后对PhysicalDatasource中的所有db进行三项检测：

* 1. 检测全局表的内部列是否存在

checker.checkInnerColumnExist();

检测的实现是通过一个SQLJob来异步操作的，对应的SQL语句为：

select count(\*) as inner\_col\_exist from information\_schema.columns where column\_name=' \_mycat\_op\_time' and table\_name='user' and table\_schema='db1';

如果返回的inner\_col\_exist 大于0,那么就表示存在内部列，如果等于0，那么就表示不存在内部列。

如果PhysicalDatasource上某个db的全局表没有内部列，那么将这些db记录在一个list中，然后在 SQL 拦截过程中进行判断，如果是全局表，但是没有内部列，那么就输出警告，不对SQL进行拦截改写，因为该全局表没有内部列，无需改写SQL。在第一项检测完成之后，才能进行第二项检测。

* 1. 检测全局表的记录总数

checker.checkRecordCout();

检查过程是类似的，都是通过SQLjob来完成的，只是对应的语句不一样：

select count(\*) as record\_count from user; (假设user表为全局表)

* 1. 检测全局表的时间戳的最大值

checker.checkMaxTimeStamp();

检查过程是类似的，都是通过SQLjob来完成的，只是对应的语句不一样：

select max(\_mycat\_op\_time) as max\_timestamp from user (假设user表为全局表)

三项检查完成之后，就获得了如下所示的结果：

全局表的记录总数(user表为全局表，并且系统有三个db)：

db1. user.record\_count: 43546565

db2. user.record\_count: 43546565

db3. user.record\_count: 43546565

全局表的最大时间戳：

db1. user.max\_timestamp: 1450578802241

db2. user.max\_timestamp: 1450578802241

db3. user.max\_timestamp: 1450578802241

然后前端，比如 mycat-eye 就可以将该结果显示出来。目前直接在log中输出，也可以考虑引入像H2这样的Java实现的嵌入式数据库来记录该结果。**H2**实现为仅仅一个jar包，十分适合作为mycat-server层面的一个非文件存储方式。有一些信息如果存在在文件中，查询起来不太方便，比如上面的检测结果就是如此。

实际的SQLJob的执行，主要参照了原有的heartbeat的实现，主要在下面两个类中：

MySQLConsistencyChecker

MySQLConsistencyHelper

具体可以参考代码，和heartbeat的实现基本是一样的。

每一次定时检查，会对所有全局表进行上述三项检测。

总结成一句：

SQL的拦截实现记录全局表被修改时的时间戳；定时任务实现对全局表记录总数和时间戳最大值的获取。

1. 如何使用全局表一致性检测
2. 在所有全局表中增加一个 bigint 的内部列，列名为 \_mycat\_op\_time，(alter table t add column \_mycat\_op\_time bigint [not null default 0]); 同时建议在该列上建立索引(alter table t add index \_op\_idx(\_mycat\_op\_time))
3. 在对全局表进行crud时，最好将内部列当作不存在一样，也就是最好不要对内部列update,insert等操作，不然会在Log中进行警告：不用操作内部列;
4. 因为全局表多了一个内部列，所以在对全局表进行insert时，必须携带列名，也就是insert into t(id,name) values(xx,xx)，不能使用insert into t values(xx,xx); 因为会报错：列数不对。这是唯一的一个小问题。未来可能会fix掉。(该限制已经被fix掉了。但还是建议使用带列名的insert语句。)