SQL to Redis Implementation

Author: Yunpei Zhou

Status: **Draft** > In Review > Approved > Implemented

Freshness: 2021/12/19

Expected Deadline: 2021/01/31

Project: https://github.com/wuzhoupei/mtor/tree/master

Objective

实现一个简易版工具，将单个 MySQL 数据库迁移到 Redis 中

Non Goal

1、外键语句及其关系

2、除日期时间、数值、字符串类型之外的数据类型

3、多个数据库备份

4、增量全量进行代码合并，目前为两份单独的代码，只是进行了文档合并

Scope

Background

总体来讲，是为了了解 tidb-tools 中的组件，其中最好复现的是 lightning，原因如下（主次排名不分先后）：

- 开发者本身并不了解 MySQL 中的语句、语法
- MySQL 备份之后的.sql 文件只包含 建库、建表、插入 三种语句，结构足够简单易于理解，甚至不需要考虑很多数据之间的关系，比如 删：是否有该数据、删除后如何处理
- Lightning 适用于第一次将 MySQL 数据全量迁移到 TiDB 中，方式便是通过 .sql 文件进行导入

然而 TiDB 底层是 rocksdb，但是现阶段开发者只接触了 Redis，于是将 TiDB 换为 Redis。

经过调查，发现 TiDB-DM 可以进行先全量后增量的迁移过程，于是本项目进行了增量方面的补充。

Design Details

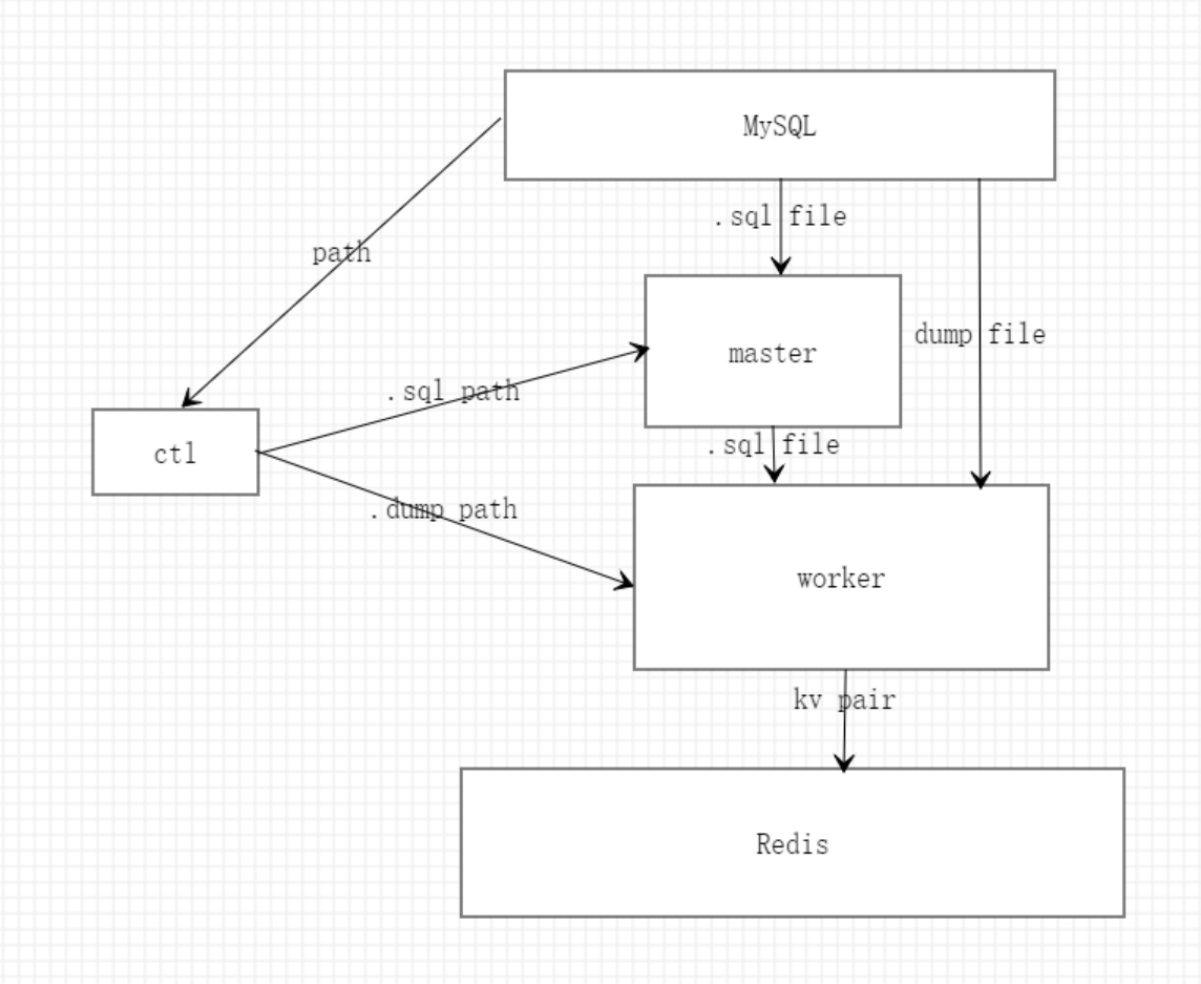
一、全量迁移

包	组件及作用
ctl	mtor-ctl.go 用于启动
CMD	CMD.go 用于操作 redis 函数的封装
master	master.go 用于分配任务
worker	worker-DDL.go 用于处理 DDL 语句 worker-DML.go 用于处理 DML 语句
codec	codec.go 用于封装编码解码函数

部分约定：

- redis 操作全部调用 redigo 包，并将所有 redis 操作封装于 CMD 包中，提供外调接口
- 由于 .sql 文件中，DDL DML 排版顺序上相互隔离，所以可以直接顺序处理，不考虑互相穿插情况，流程中默认本条件
- 由于 redis 是键值数据库，所以需要对 MySQL 的数据进行编码，具体编码部分放于本部分末尾，流程中默认本条件
- 本项目默认一个 .sql 文件对应一个数据库，不存在 .sql 文件中包含多个数据库情况

迁移流程：



1、使用 mtor-ctl.go 调用 CMD 包，连接 Redis

2、mtor-ctl.go 将 dump path 传入 work-DML，方便操作时导入 .dump 文件；然后传入 .sql path 并运行 master.go

3、master 遍历 .sql 文件，将 .sql 文件按行转为 string 数组，分为两部分进行：DDL、DML，每部分均将整个数组传入（为了避免时空损耗，本处传递地址即可）

4、DDL、DML 操作，本处使用识别关键字的方式进行语句判断（基于 .sql 文件无误）：

1) DDL

   a) master 读取 .sql 文件后，将建库删库语句分别作为任务，依次分配给 worker-DDL 处理，等待结束。建库时我们在库内部存储本库的名称，对应的寻库方式便是遍历所有库，判断库名是否吻合。删库则是先找库，然后删除整个库。(理论上不会存在删库情况，因为是初始化的全量迁移，必然是空库然后导入新库)

   b) master 读取 .sql 文件后（事实上，这里不需要重新读取文件，直接顺序处理即可，不过重新读取也 ok)，将建表语句分别作为任务，依次分配给 worker-DDL，等待结束。这里分为建表和建表内列名两部分，建表时，本库存储表名和 id 对应关系；建

列时，同样存储名称和 id 对应关系。本处由于表都在一个数据库中，不会出现切换库的操作，同时 kv 存储方式使得表结构体解耦，可以考虑多线程。

c) DDL 部分结束

2) DML

a) master 读取 .sql 文件后，将不同 .dump 文件作为不同任务，依次分配给 worker-DML 处理，等待结束。此处进行行存编码时，需要得知每个值的类型，所以我们对于每个 dump 文件，先根据其表提取对应类型，提前处理，然后再进行编码导入。同时底层归并了一些类型，便于编码存储。本处同样是因为不需要切换库，同时 kv 存储方式使得内部结构解耦，可以考虑多线程。可以直接按照 .dump 文件进行任务分割，也可按照数据量进行更小粒度的分割，不过这样会导致工程量上升，但是在数据量差别较大时，是十分值得一试的。

b) DML 部分结束

ps：单线程时，worker-DDL、worker-DML 均按照描述，串行即可。

本工具中，会将所有的 SQL 关系，全部存入对应的数据库中。而对于 SQL 关系转化为 kv pair，规则如下：

库：

Key : _DBName

Value : {database name}

表：

Key : _ti{table id} // 根据 id 查表名

Value : _tn{table name}

Key : _tn{table name} // 根据表名查 id

Value : _ti{table id}

列：

Key : _ti{table id}_ci{column id} // 根据列 id 查列名

Value : _cc{column can nil}_ct{column type}_cn{column name}

Key : _ti{table id}_cn{column name} // 根据列名查列 id

Value : _cc{column can nil}_ct{column type}_cn{column id}

键：

Key : _ti{table id}_P_cn{column name} // 是否为主键

Value : 1

Key : _ti{table id}_U_cn{column name} // 是否为唯一约束

Value : 1

此上编码（目前已重新设计）成立。原因为，只有最后一段编码不是数字，单独识别即可。

行存：

Key : t{table id}_r{row id}

Value : _i_{int}

_f_{double}

_s_{string}

_d_{YYYY-MM-DD}

_t_{HH:MM:SS}

_y_{YYYY}

_dt_{YYYY-MM-DD HH:MM:SS}

_ds_{YYYYMMDD HHMMSS}

索引：

Key : t{table id}_ci{colums id}_ce{columns element}_r{row id}

Value : 空

行存和索引中，元素按照此格式直接进行拼接，'_' 字符代表 '\x00'。问题在于，两个字母还是有点占内存，但是为了可读性，就先这样。

至于更细则的类型划分，因为会转为 string 存储，所以无论长短，直接用 int32、[]byte 存储后转换为 string，或者转换回去，至于查询原始类型，已经在列名中存储了。

二、增量迁移

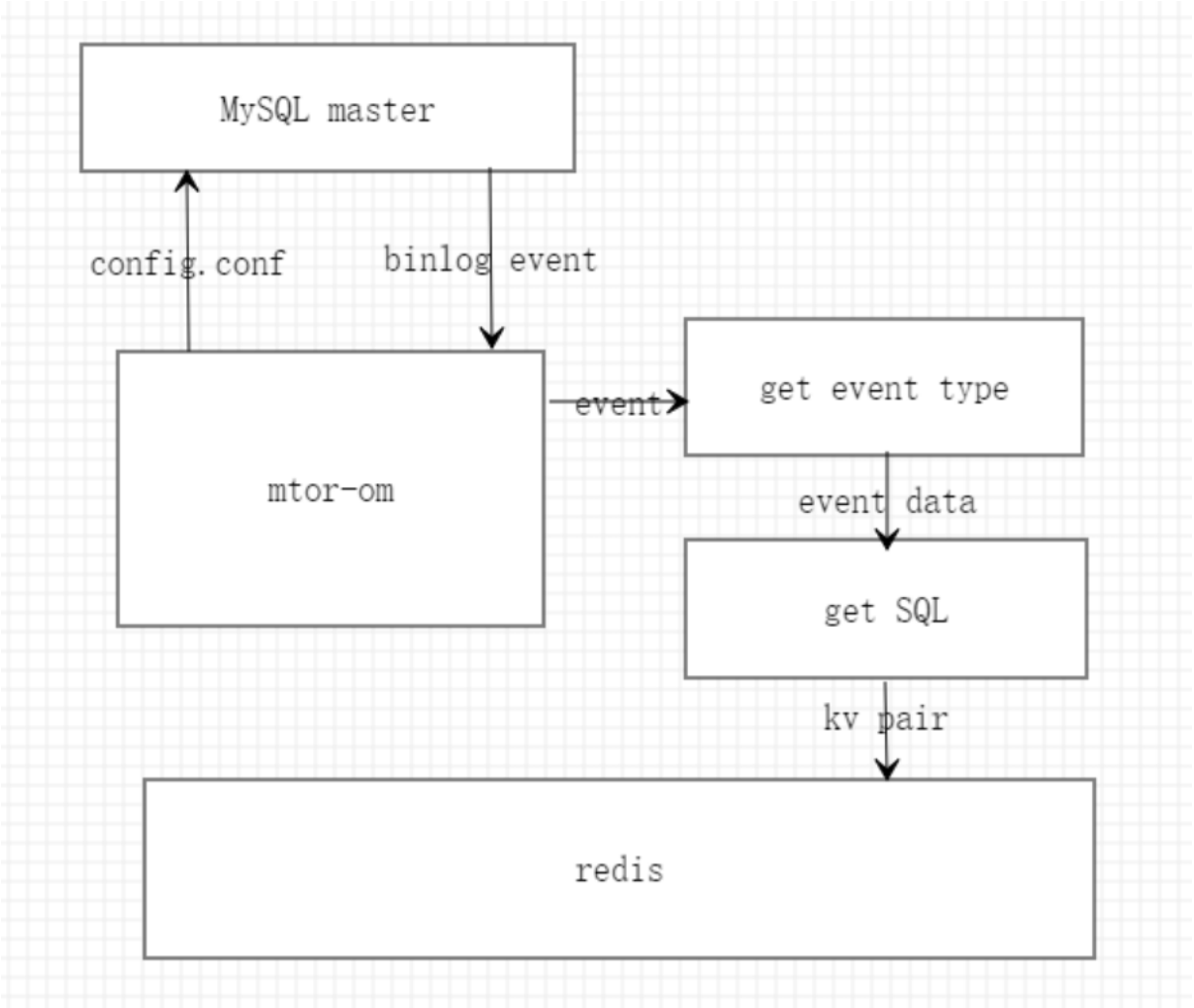
包名	作用
include	部分开源包源码
global	config.conf 配置文件 global.go 配置文件读取、全局函数和变量定义
linkDB	linkDB.go 伪装为 Slave 连接 MySQL getEvent.go 不断拉取 event

eventH	eventH.go 解析 event header，进行分类
eventD*	eventD.go 各种解析 event data 函数
toRedis*	得到 SQL 语句并执行到 Redis

部分约定：

- 只识别了部分 event ：
  - FORMAT_DESCRIPTION_EVENT：binlog 文件的第一个 event，记录版本号等元数据信息
  - QUERY_EVENT: 存储 statement 类的信息，基于 statement 的 binlog 格式记录 sql 语句，在 row 模式下记录事务 begin 标签
  - XID_EVENT: 二阶段提交 xid 记录
  - TABLE_MAP_EVENT: row 模式下记录表源数据，对读取行记录提供规则参考
  - WRITE_ROWS_EVENT/DELETE_ROWS_EVENT/UPDATE_ROWS_EVENT: row 模式下记录对应行数据变化的记录
  - GTID_LOG_EVENT: 这个就是记录 GTID 事务号了，用于 5.6 版本之后基于 GTID 同步的方式
  - ROTATE_EVENT: 连接下一个 binlog 文件
- 并没有解析 SQL 语句导入到 redis 中，只考虑了如何得到 event 部分。主要是因为开发人员对 SQL 语句不太了解，解析难度较大
- 本处并未考虑并行

迁移流程：



1. 伪装为 Slave 连接到 MySQL master，此处使用 config 中预写的配置，通过 kingshard 中的连接包进行连接。得到最新的 binlog pos
2. 封装 COM_BINLOG_DUMP 通信 packet，然后持续接收 binlog event。本处可以直接调用 kingshard 包。
3. 得到 event 之后，找到对应位的值，分辨出对应的 event type，并将 event data 放入到对应的函数中处理
4. 不同函数进行处理即可。

Testing Plan

预想为：用程序随机 sql 查询语句，然后分别到 MySQL、Redis 中查询，结果分别放到两个文件中，进行对比（整体思路借鉴 OI 中对拍思想）。

Monitoring*

预先读取 binlog file 数量并存储，每隔 1s（时间自定义）将当前 binlog file name、binlog pos 输出到固定文件。封装一个进程，查询进度时，直接按照本文件进行分析即可。如果需要实时反馈进度，直接将进度查询任务常驻后台实时显示即可。也可根据进度文件进行可视化处理。

Working Plan

Item	Owner	Estimate	Status
Create a new repository		0.5d	Finished
To complete the single thread on DDL	Yunpei Zhou	1d	Finished
Find a suitable set of tests, and test DDL unit	Yunpei Zhou	0.5d	Finished
Redisign the codec and implement the DML unit	Yunpei Zhou	0.5d	Finished
Use Big test data to test the all units	Yunpei Zhou	2d	New
Change the single thread to multi-thread	Yunpei Zhou	0.5d	Finished
Test the speed	Yunpei Zhou	At any time	Doing
Add command line operations	Yunpei Zhou	1d	New

TODO

1. redis 内部存储是否有序？
2. 如何导出 log 以及导出的 log 怎么存储？

Update History

2021/12/20 - 2021/12/24

1. 讨论了为什么没有外键的问题，这个问题留到后面去探究
2. 更新了库、表、列的编码格式，重新设计了行存编码格式，引入了空字符 '\x00'，用来分隔每个元素
3. 测试了性能，1 核 2G，0.3M/s（系统内存就 2G，这个速度不是纯内存情况，应该存在磁盘读写）
4. 多线程根本没啥速度优化，性能瓶颈还是在机器上吧，一跑就崩，离谱。过程较为容易，

因为一开始就规划为多线程框架、单线程初稿，不过还是考虑不太周，用了全局变量，导致多了些改动，以及 Redis 操作的锁问题，这里我还是没太想明白

2021/12/25 - 2021/12/31

1. 由于项目过于简单，于是打算添加命令行操作界面，再之后考虑写数据正确性测试。但是服务器太拉，可能不支持再添加一个 MySQL，所以不如考虑下为什么 TiDB 没有外键