**Tasktool 工具使用说明**

1. **概述**

tasktool是一个连接tidb数据库的客户端工具，使用tasktool支持的语法可以灵活地完成拆批、跑批任务。

2. **工具使用场景**

1. 没有批量框架支持的情况下运行拆批、跑批任务
2. 没有合适客户端情况下，作为数据库交互工具
3. 批量框架无法对单个作业设置并发的情况下，作为单个作业的执行工具
4. 批量生成测试数据

3. **安装运行**

3.1 **平台及环境要求**

|  |  |  |
| --- | --- | --- |
| **项目** | **需求** | **备注** |
| 操作系统 | 无特殊要求 | 支持跨平台 |
| JDK/JRE | 1.8及以上 |  |
| jdbc驱动 | 无特殊要求 | 内置mysql jdbc 8.0.22  可根据实际情况设置所需版本 |

3.2 **安装步骤**

只有一个tasktool.jar文件，下载即可使用

**[tasktool.jar]**

3.3 **运行程序**

* 交互式执行

|  |
| --- |
| Shell #作为TiDB客户端，交互式运行 java -jar tasktool.jar  ======================================================================== Welcome to the tasktool. Commands end with ;  tasktool version 1.0.0 Mailto: shponyxie@163.com  use .help; to get help  tasktool> .logon 127.0.0.1:4000 user "root" password ""; driver: mysql-connector-java-8.0.22 (Revision: d64b664fa93e81296a377de031b8123a67e6def2) logon to 127.0.0.1:4000 successfully! tasktool> |

* 生成加密密码

|  |
| --- |
| Shell #生成加密明文密码（密文密码将用在批量任务脚本中) java -jar tasktool.jar -e "pswd123\\@" encoded password:gwxtyKmmoMSvYW+G34GLOA== |

* 校验批量脚本语法

|  |
| --- |
| Shell #校验job1.tsk脚本语法是否正确 java -jar tasktool.jar -c job1.tsk syntax of file job1.tsk is ok! |

* 运行批量脚本

|  |
| --- |
| Shell #非交互模式执行job1.tsk，应用可根据程序退出码判断是否执行成功 java -jar tasktool.jar -f job1.tsk driver: mysql-connector-java-8.0.22 (Revision: d64b664fa93e81296a377de031b8123a67e6def2) logon to 127.0.0.1:4000 successfully! --> select count(\*) from information\_schema.tables  2023-11-07T15:33:27.938071 elapsed: 0.026 seconds, affected rows: 1 +----------+ | count(\*) | +----------+ | 798 | +----------+ logoff from database successfully! Program finished with exit code: 0 |

4. **脚本编写语法参考**

4.1 **命令说明**

脚本由多个命令组成，每个命令

* 用点号开头
* 用分号结束
* 复合命令.IF 和 .FOR 中的子命令**不使用分号结束**

4.2 **命令清单**

4.2.1 **.help 显示帮助**

语法：.help;

4.2.2 **.logon 登录数据库**

语法： .logon ip1:port1/port2, ip2:port1/port2 user "xxx" password "PASSWORD";

如果没有ELB可以配置多个TiDB主机和端口，同一主机多个端口间用斜杠分隔，多个主机间用逗号分隔，若配置多个实例，并发运行sql时会按round robin方式进行负载均衡

* 用户名用双引号、单引号、倒引号包围
* 密码，如果需要解密则必须用倒引号包围，否则用双引号、单引号包围
* 密码支持从环境变量读取，如从环境变量PSWD中读取： password "{env:PSWD}"

4.2.3 **.logoff 登出数据库**

语法： .logoff;

4.2.4 **.os 调用操作系统命令**

语法：.os {{ 命令 }};

调用操作系统命令，命令用双花括号包围 如： .os {{ df -h }}; 调用操作系统命令后，可通过 OS\_CODE 来判断外部命令是否执行成功

|  |
| --- |
| Shell .os {{ rm /tmp/lock }}; .if os\_code != 0 .then .exit 1 .end if; |

4.2.5 **.print 消息打印命令**

语法：.print {{ 消息 }};

4.2.6 **.sql get 读取数据命令**

语法：.sql get (with retry errorlist? )? {{ SQL语句 }};

执行SQL语句，并打印结果，SQL语句用双花括号包围，只能是单条SQL语句

* SQL语句必须是查询语句
* 语句是否执行成功可通过 error\_code是否为0来判断
* 记录数可通过activity\_count访问 如：.if activity\_count > 0 .then ..... .else ...... .end if;
* 如果语句中带了retry子句, 则
* 重试次数由.SET RETRY 命令设置，如果没有设置则不会重试
* 若errorlist为空，只要出错就会重试
* 若errorlist不为空，则只有错误码在errorlist里面时会重试否则不重试

4.2.7 **.sql do 执行SQL命令**

语法：.sql do? (with retry errorlist? )? {{ SQL语句 }};

执行SQL语句，do关键字可以省略，SQL语句用双花括号包围，只能是单条SQL语句。

* SQL语句主要是DML，也可是查询语句
* 语句是否执行成功可通过 error\_code是否为0来判断
* 可通过activity\_count判断受影响的记录数 如：.if activity\_count > 0 .then ..... .else ...... .end if;
* 如果语句中带了retry子句, 则
* 重试次数由.SET RETRY 命令设置，如果没有设置则不会重试
* 若errorlist为空，只要出错就会重试
* 若errorlist不为空，则只有错误码在errorlist里面时会重试否则不重试

4.2.8 **.sql batch 批量拆分命令**

语法：.sql batch reset? (id | id =)? batchID procid 'xxxxxxxx'

on? tablename (each n)?

(BY columnName (GROUP BY groupByExpr )?)?

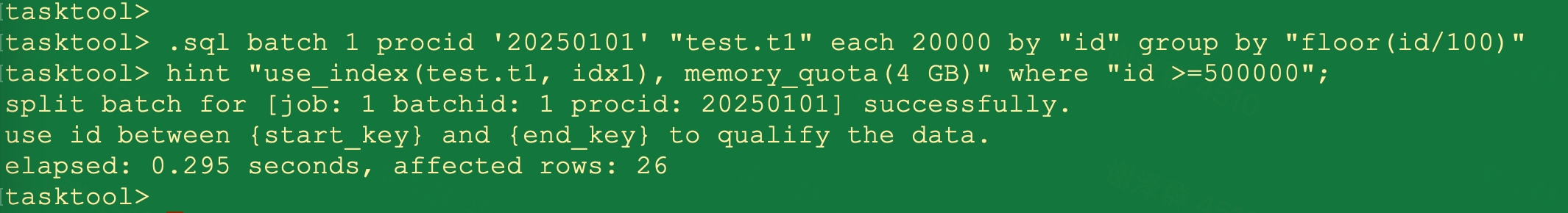
('HINT' hintStatement)?

('WHERE' whereStatement)?

对指定的表进行拆批，拆批结果保存在.set batch\_table语句设置的资料表中，拆批结果是 .sql run batch命令的输入

* reset： 若设置，则拆批前先检查是否已经拆过批，若已经拆过则清理原拆批结果
* Batchid: 若一个脚本中涉及多个拆批语句，则每个拆批语句设置一个唯一编号
* Procid: 脚本对应的处理标识，譬如 业务日期
* Each n: 每n条记录一批（实际拆批结果不一定等于n）
* BY columnName: 拆批字段（最多两个字段，若不设置，则根据表类型自行选择，非聚簇表使用\_tidb\_rowid,聚簇表则使用主键的前两个字段
* Group By: 拆批时先按Group By表达式进行数据收敛，提升拆批语句性能
* Hint: 拆批语句用到的SQL Hint， 如 Memory\_quota(4 GB)
* Where: 拆批数据过滤条件（非全表拆批）

|  |
| --- |
| SQL .sql batch 1 procid '20250101' `test.t1`; -- 默认1万条记录一批 .sql batch reset id = 1 procid '20250501' on "test.t1" each 20000;  .sql batch 1 procid '20250101' "test.t1" each 20000 by "id" group by "floor(id/100)";  .sql batch reset 1 procid '20250101' "test.t1" each 20000 by "id" group by "floor(id/100)"  hint "use\_index(test.t1,idx1), memory\_quota(4 GB)"; |



4.2.9 **.sql run batch 执行批量命令**

语法：.sql run batch id batchID procid 'xxxxxxxx'

(.with cmd\_set\*)?

'.begin'

statement+

'.end'

根据batchid 和 procid 读取未成功执行的拆批任务，并执行批量任务。

.with cmd\_set： 初始化操作（参考for语句）

**注意：**

1. **语句中不要另外再使用事务，跑批语句会更新任务表状态，状态表的更新和业务数据的更新包含在同一个内置事务中，只要SQL语句有异常会自动回滚。**
2. **命令执行后，需要使用 .if all success 来判断是否全部执行成功,不能使用 .if error\_code<> 0来判断语句是否执行成功**

|  |
| --- |
| SQL .sql run batch 1 procid '{args:1}' .with .set concurrency 2 -- 设置并发 .set database {{use test }} .begin  .sql {{insert into t1\_copy   select \* from t1  where \_tidb\_rowid between {start\_key} and {end\_key}  }} .end; .if all success .then .quit 0 .else .quit 1 .end if; |

4.2.10 **.try 命令**

语法： .try

statments

.exception

statments

.end try

try语句将一组命令包装在一起，期间一个SQL语句有异常既跳到exception模块，使用try语句可以简化对SQL结果的检测，try语句中不支持.sql run batch语句。

|  |
| --- |
| SQL .try  .set jobid 1  .set database {{use test }}  .sql batch reset 1 procid '{args:1}' 'test.t1' each 20000 where 'id>=950000'  .sql {{truncate table t1\_copy }}  .sql {{split table t1\_copy between (0) and (9223372036854775807) regions 4 }} .exception  .quit 1  .end try; |

4.2.11 **.set 设置运行参数命令**

* .set LOG '='? (SUMMARY | DETAIL);

设置log模式，默认SUMMARY，若设置为Detail，将打印for循环语句中的SQL

* .set DRIVER '='? "JDBC驱动jar包";

设置tasktool使用的mysql jdbc驱动版本，默认使用mysql 8.0.22。

.set driver = "/home/etl/MyJars/mysql-connect-java-5.1.49.jar";

* .set CONCURRENCY '='? n1 (WATCH ('FILE' | 'FILE =')?  
   "并发监控文件名(含路径) EACH n2 SECS)?

设置for语句的并发数 n1，默认for循环不并发执行

如果指定了并发监控文件，则执行过程中每隔n2秒从文件获取新的并发数 (文件第一行，一个合法整数) 时间间隔不小于20秒，手工跑数时可根据系统负载在不重启任务的情况下动态调整并发数

* .set ERROR LIMIT '='? n;

设置for循环语句中最大错误数 n，默认为0，若并行执行for语句，最终的错误数可能超出此限制

* .set RETRY n INTERVAL n1 (, n2 )? SECS

设置失败语句重试次数，如果interval只设置n1个则每次间隔n1秒后重试，如果设置了n1和n2，则每次间隔n1+ random(n2)秒后重试，总共最大重试n次

* .set JOBID '='? n;

设置当前作业编号，若一个作业对应一个脚本，通过此命令设置作业编号，JOBID在拆批和跑批语句中被用到

* .set BATCH TABLE '='? tableName

设置拆批跑批语句用到的资料表，默认表名：batch\_task

* .set DATABASE {{ SQL 语句 }};

设置数据参数，可用来设置一些session级别的变量,多放在for循环语句的初始化部分 如：

.set database {{ set session tidb\_mem\_quota\_query = 4096000000 }};

4.2.12 **.if 流程控制命令**

|  |
| --- |
| Shell 语法： .if condition .then  command1  command2  ... ... .else  command1  command2  ... ... .end if; |

根据condition结果执行不同的流程，子命令可以嵌套.if 和 .for 命令，**子命令不用分号结尾**

* condition语句只支持单句简单判断，不支持and/or运算
* condition语法1: **操作变量** **操作符** **比较数值n**

操作变量:

error\_code 判断上一个命令是否执行返回码

os\_code 判断上一个.os 调用外部命令返回码

activity\_count 判断上一个SQL命令返回的记录数或者被影响的行数

error\_count 判断上一个for语句中错误数

操作符：> (大于) <(小于) <>(不等于) !=(不等于) >=(大于等于) <=(小于等于)

.if activity\_count != 1 .then .... .end if;

* condition语法2: all success 或者 not all success

用于判断for循环语句执行是否全部成功

.if all success .then ... .end if;

4.2.13 **.for 循环命令**

|  |
| --- |
| Shell **语法1：数据游标, SQL语句返回要遍历的结果集** .for **data** in {{ SQL 语句 }} .with  .set 语句1  .set 语句2  ... ... .do  command1  command2  ... ... .end for;  **语法2：序列，n1、n2、n3为整数，step可以省略**  **如果n1>n2 step默认为-1，如果赋值，则必须小于0** 如: from 100 to 1 step -2  **如果n1<n2 step默认为1， 如果赋值，则必须大于0** 如: from 1 to 100 step 2 .for **seq** id from n1 to n2 step n3 .with  .set 语句1  .set 语句2  ... ... .do  command1  command2  ... ... .end for;  **语法3: 常量列表，常量必须用单引号、双引号或者倒引号包围，两个引号表示引号自身** .for **list** in {{ c1,c2,c3... }} .with  .set 语句1  .set 语句2  ... ... .do  command1  command2  ... ... .end for; |

for语句循环遍历数据集，子命令可以嵌套.if 和 .for 命令，**子命令不用分号结尾**

* **.with** 子句用来做一些初始化操作，可以不设置
* 若要并发执行for语句，则需在for语句之前设置并发参数 .set concurrency n
* for语句使用新的数据库连接执行命令， 即使没有设置并发，for语句也会单独启动一个数据库连接
* 在for语句内部
* 使用 **.break** 退出最近的for循环
* 使用 **.continue** 忽略当前for循环当前行后的命令

4.2.14 **.exit 退出命令**

语法： .exit n;

以退出码 n 退出程序,调用tasktool的主调程序可根据此退出码做不同的操作

4.2.15 **变量引用**

SQL语句，print语句，logon password子句可以引用变量

* 环境变量

语法： {**env**:变量名} 变量名大小写敏感

* 命令行参数

语法： {**args**:参数序号} 序号从1开始, 大小写不敏感

|  |
| --- |
| Shell #假设跑批交易日期为20231111 java -jar tasktool.jar -f job1.tsk 20221111  脚本job1.tsk中可以通过 {args:1} 来引用交易日期参数 .sql get {{ select count(\*) from fin\_txn\_log where txn\_dt = '**{args:1}**' }}; |

* sql语句错误代码和错误描述

语法：{**sqlerr**:code} 错误码 {**sqlerr**:message} 错误描述，大小写不敏感

|  |
| --- |
| Shell .sql {{update log\_tbl set status = 'F',   msg='**{sqlerr:message}**',  errcode=**{sqlerr:code}**,  endTime = current\_timestamp()  where batchid = {batchid}  }}; |

* for语句游标中字段值

语法: {字段名} 大小写不敏感

|  |
| --- |
| Shell .for data in {{select table\_name as tblname   from information\_schema.tables limit 10  }} .do  .print {{table name: {tblname} }} .end for; |

4.2.16 **注释**

* 单行注释 行首用 --
* 多行注释 /\* ... \*/

4.3 **脚本示例**

4.3.1 **准备测试环境**

demo\_prepare\_data.tsk: 创建测试表并生成测试数据

**[demo\_prepare\_data.tsk]**

|  |
| --- |
| Shell #生成交易日期为20231111一天的数据 java -jar tasktool.jar -f demo\_prepare\_data.tsk 20231111 |

|  |
| --- |
| SQL .logon 127.0.0.1:4000 user "root" password `CLPm+K3NCRYj7Mvks3gfJQ==`; --1.1 创建日志schema .sql {{ create schema if not exists tasktool; }}; .if error\_code != 0 .then .quit 11 .end if;  --1.2 创建拆批任务表 .set database {{use tasktool }}; .sql {{  create table if not exists batch\_split\_log(  job\_id int  ,batch\_dt date  ,split\_value varchar(50)  ,opt\_time datetime  ,primary key(job\_id, batch\_dt, split\_value) clustered  );  }}; .if error\_code != 0 .then .quit 12 .end if;  --1.3 创建任务明细表 .sql {{  create table if not exists batch\_task\_log(  task\_id bigint auto\_random  ,job\_id int  ,batch\_dt date  ,split\_value varchar(50)  ,batch\_rows int  ,task\_status int -- 0: init 1:running 2:failed 3:success  ,start\_key varchar(100)  ,end\_key varchar(100)  ,start\_time datetime  ,end\_time datetime  ,task\_errcode int  ,task\_errmessage varchar(256)  ,primary key(task\_id) clustered  ,key idx\_job\_batch(job\_id,batch\_dt,split\_value)  )  }}; .if error\_code != 0 .then .quit 13 .end if;  --2.1 创建数据schema .sql {{ create schema if not exists demo; }}; .if error\_code != 0 .then .quit 21 .end if;  --2.2 创建交易明细表 .sql {{use demo }}; .sql {{  create table if not exists ods\_cust\_fin\_journal(  cust\_no varchar(10)  ,txn\_dt date  ,txn\_serial decimal(10,0)  ,txn\_currency int  ,txn\_amt decimal(18,2)  ,txn\_org varchar(10)  ,key txn\_dt\_cust(txn\_dt,cust\_no)  ) shard\_row\_id\_bits = 4;  }}; .if error\_code != 0 .then .quit 22 .end if;  --2.3 创建批量结果表 .sql {{  create table if not exists ads\_cust\_fin\_summary(  cust\_no varchar(10)  ,txn\_dt date  ,total\_amt decimal(18,2)  ,total\_cnt int  ,primary key(cust\_no,txn\_dt) clustered  )  }}; .if error\_code != 0 .then .quit 23 .end if;  --3 生成测试数据 .set concurrency 5; .for data in {{ /\* 00 ~ 99 100个客户号前缀 \*/  with recursive tmp(cust\_no\_prefix) as (  select 100 as cust\_no\_prefix  union all  select cust\_no\_prefix + 1  from tmp  where cust\_no\_prefix < 199  )  select substr(cust\_no\_prefix,2) cust\_no\_prefix from tmp  }} .with  .set log summary  .set database {{use demo; }} .do  .print {{begin to generate data cust\_no prefix is: '{cust\_no\_prefix}' }}  .sql {{ set @@cte\_max\_recursion\_depth = 20000 }}  .set concurrency 10  .for data in {{ /\* 随机生成 1000 ～ 10000 记录 \*/  with recursive tmp(serial) as (  select 11000 + floor(10000\*rand()) as serial  union all  select serial - 1  from tmp  where serial > 11000  )  select \* from tmp  }}  .with  .set database {{use demo }}  .do  .sql {{  insert into ods\_cust\_fin\_journal  values(concat('{cust\_no\_prefix}','00000',substr(floor(1000000000 \* rand()),1,3))  ,'{args:1}'  ,{serial}  ,147  ,10000000 \* rand()  ,'{cust\_no\_prefix}00001')  }}  .end for .end for; .if not all success .then .quit 3 .else .quit 0 .end if; |

4.3.2 **拆批跑批**

demo\_job\_txn\_summary.tsk 统计每个客户每日交易笔数和交易量

demo\_job\_txn\_summary.tsk

|  |
| --- |
| Shell #批量日期20231111 java -jar tasktool.jar -f demo\_job\_txn\_summary.tsk 20231111 |

|  |
| --- |
| SQL .logon 127.0.0.1:4000 user "root" password `CLPm+K3NCRYj7Mvks3gfJQ==`;  --1 根据客户号前缀，拆分100个批次,5个并发进行拆批, 每个批次1000个客户 .set concurrency 5; .for data in {{ /\* 00 ~ 99 100个客户号前缀 \*/  with recursive tmp(cust\_no\_prefix) as (  select 100 as cust\_no\_prefix  union all  select cust\_no\_prefix + 1  from tmp  where cust\_no\_prefix < 199  )  select substr(cust\_no\_prefix,2) cust\_no\_prefix from tmp  }} .with  .set database {{use demo; }}  .set database {{set session tidb\_mem\_quota\_query = 4096000000 }} .do  .sql get {{ select 1 from tasktool.batch\_split\_log  where job\_id = 1  and batch\_dt = '{args:1}'  and split\_value = '{cust\_no\_prefix}'  }}  .if activity\_count != 0 .then .continue .end if /\* 此前缀已经拆过批，跳过 \*/  .sql {{ begin }}  .sql {{ insert into tasktool.batch\_task\_log  ( job\_id  ,batch\_dt  ,split\_value  ,batch\_rows  ,task\_status  ,start\_key  ,end\_key  )  SELECT 1  ,'{args:1}'  ,'{cust\_no\_prefix}'  ,count(\*)  ,0  ,min(cust\_no)  ,max(cust\_no)  from (  select cust\_no, rank() over(order by cust\_no) rk  from demo.ods\_cust\_fin\_journal  where txn\_dt = '{args:1}'  and cust\_no like '{cust\_no\_prefix}%'  ) t  group by floor((rk-1)/1000) /\* 1000 一个任务 \*/  }}  .sql {{ insert into tasktool.batch\_split\_log(job\_id,batch\_dt,split\_value,opt\_time)  values(1,'{args:1}','{cust\_no\_prefix}',current\_timestamp())  }}  .sql {{ commit }}  .if error\_code <> 0 .then .quit 1 .end if  .end for;  --2 汇总客户交易信息  /\* 如有需要，打散结果表 \*/ .sql {{ split table demo.ads\_cust\_fin\_summary index `primary` by ('0'),('2'),('4'),('6') }}; /\* sleep 10秒，等待region调度 \*/ .sql {{ select sleep(10) }};  .set concurrency 10; .for data in {{  select task\_id, start\_key, end\_key  from tasktool.batch\_task\_log  where job\_id = 1  and batch\_dt = '{args:1}'  and task\_status in (0,1,2)  }} .do  .sql {{ update tasktool.batch\_task\_log  set task\_status = 1  ,start\_time = current\_timestamp()  ,end\_time = null  ,task\_errcode = null  ,task\_errmessage = null  where task\_id = {task\_id}  }}  .sql {{ replace into demo.ads\_cust\_fin\_summary(  cust\_no  ,txn\_dt  ,total\_amt  ,total\_cnt  )  select  cust\_no  ,txn\_dt  ,sum(txn\_amt)  ,count(\*)  from demo.ods\_cust\_fin\_journal  where txn\_dt = '{args:1}'  and cust\_no between '{start\_key}' and '{end\_key}'  group by cust\_no, txn\_dt  }}  .if error\_code != 0 .then /\* 修改task为失败状态 \*/  .sql {{ update tasktool.batch\_task\_log  set task\_status = 2  ,end\_time = current\_timestamp()  ,task\_errcode = {sqlerr:code}  ,task\_errmessage = '{sqlerr:message}'  where task\_id = {task\_id}  }}  .else /\* 修改task为成功状态 \*/  .sql {{ update tasktool.batch\_task\_log  set task\_status = 3  ,end\_time = current\_timestamp()  ,task\_errcode = 0  ,task\_errmessage = 'success'  where task\_id = {task\_id}  }}  .end if .end for;  /\* 如果有子任务失败，退出码 2 否则退出码 0 \*/ .if not all success .then .quit 2 .else .quit 0 .end if; |