23.MySQL-备份恢复 √

一.数据损坏的类型

1.1 物理损坏

> 磁盘损坏，核心数据文件被删除或者损坏
> 解决方法：主从复制，高可用

1.2 逻辑损坏

> drop delete truncate update
> 解决方法：备份,binlog,binlog2sql,延时从库

1.3 备份恢复的DBA职责

> a.设计能备份和恢复的策略
> b.备份的定期检查
> c.定期的恢复演练
> d.数据损坏的恢复（讲究效率和一致性）
> f.迁移，升级，主从

1.4 备份工具

a.物理备份

1.Percona Xtrabackup 简称PXB,XBK,Xbackup（社区版 免费）8.0.12+
2.MySQL Enterprise Backup 简称MEB （企业版 收费）
3.8.0新特性clone plugin(8.0.17)

b.逻辑备份

1.Mysqldump 简称MDP （mysql自带工具） 备份效率比较的慢，适用于少量数据
2.Mydumper&Myloader（一套工具）
3.逻辑导入导出 load data/mysqlimport

二.===逻辑备份工具===

1.Mysqldump 简称MDP （mysql自带客户端工具）

1.1 备份内容

> 备份的结果: SQL语句,Create database, create table , insert into 语句

1.2 备份过程

> 1.首先隔离级别设定为RR级别 获取一致性快照
> 2.进行FTWRL Flush tables with read lock; ---> MDL锁
> Flush tables ---> close tables,阻止所有commit
> read lock ---> all read lock 开启读锁
> 3.对元数据进行备份
> show create database .. ---> 建库语句
> show create table .. ---> 建表语句
> 4.解锁 unlock tables; 只会解锁innodb的表，不会解锁myisam的表
> 5.备份数据行会根据不同的存储引擎进行不同的操作
> InnoDB可以进行快照备份（热备）拼接成insert
> 非innoDB表，锁表备份

1.3 适合的场景

> 单节点数据量,100G以内,单表2000w以内.可以使用MDP
> 恢复时间一般是备份时间的3-5倍.

1.4参数应用

A.连接参数

B.备份参数

1.-A 全库备份

```
[root@db01 ~]# mysqldump -uroot -p123 -S /tmp/mysql.sock -A >/tmp/full.s
```

2. -B 单库或多库备份

```
[root@db01 ~]# mysqldump -uroot -p123 -S /tmp/mysql.sock -B oldboy test
```

3.备份单表或者多表

```
#注意 恢复数据时应该先建立数据库
[root@db01 ~]# mysqldump -uroot -p123 -S /tmp/mysql.sock  test t100w   >
                                                          库名 表名
```

C.其他备份参数 (mysqldump --help解释)

4. --master-data=2

> 会将position号和文件名追加在导入的文件中
> 功能：
> 1.自动记录binlog信息（以注释的形式,保存备份开始时间点的binlog的状态信息 ）
> 2.自动进行GRL锁(FTWRL) 阻塞DML语句
> 加了--single-transaction 会减少全局锁表时间.

```
mysqldump -uroot -p123  -S /tmp/mysql.sock -A --master-data=2  >/tmp/ful
```

加上 --master-data=2的效果

```
-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000003', MASTER_LOG_POS=21106881;
```

5.--single-transaction （结合 --master-data=2 一起使用）

1.针对有mvcc机制的存储引擎的innodb进行一致性快照备份（热备）
2.备份期间如果有DDL会导致备份失败

> master-data可以自动加锁
> （1）在不加--single-transaction ，启动所有表的温备份，所有表都锁定
> （2）加上--single-transaction ,对innodb进行快照备份,对非innodb表可以实现自动锁表功能

6.--max_allowed_packet=128M 允许最大数据包大小

应用场景：
1.添加在客户端：备份数据（从服务端拉取数据）
2.添加在客户端：向服务端存储数据

7.--set-gtid-purged=auto

默认等于auto(等价on)，关闭off

使用场景:

```
使用场景:
1. --set-gtid-purged=OFF,可以使用在日常备份参数中.
mysqldump -uroot -p -A -R -E --triggers --master-data=2 --single- trans
action --set-gtid-purged=OFF >/data/backup/full.sql
2. auto , on:在构建主从复制环境时需要的参数配置
mysqldump -uroot -p -A -R -E --triggers --master-data=2 --single- trans
action --set-gtid-purged=ON >/data/backup/full.sql
```

8.-R -E --triggers

备份时,同时将 存储过程及函数\事件\触发器 等高级对象备份走.

9. -F

在备份开始时，刷新一个新binlog日志

D.标准化备份语句

```
mysqldump -uroot -p -A --master-data=2 --single-transaction -R -E --tri
gz包解压
gunzip + gz包
```

1.5 案例:通过mysqldump全备+binlog实现PIT（point ）数据恢复

环境背景： 小型的业务数据库，50G，每天23:00全备，定期binlog异地备份。

故障场景： 周三下午2点，开发Navicat连接数据库实例错误，导致生产数据被误删除（DROP）

恢复思路:

1. 挂维护页。
2. 检查备份、日志可用。
3. 如果只是部分损坏，建议找一个应急库进行恢复
   a. 全备恢复
   b. 日志截取并恢复
4. 恢复后数据校验 （业务测试部门验证）
5. 立即备份（停机冷备）
6. 恢复架构系统
7. 撤维护页，恢复业务

模拟故障:

```
1. 模拟测试数据
mysql> create database pit;
mysql> use pit
mysql> create table t1 (id int);
mysql> insert into t1 values(1),(2),(3);
mysql> commit;
2. 全备
[root@db01 tmp]# mysqldump -uroot -p -A --master-data=2 --single-transa
3. 模拟周三白天的操作
mysql> use pit
mysql> insert into t1 values(11),(22),(33);
mysql> commit;
4. 模拟周三下午2:00,删库操作
mysql> drop database pit;
```

恢复过程:

1.恢复全备

```
mysql>source /tmp/full_2300.sql  (文件中写有set sql_log_bin=0;)
```

2. 截取二进制日志

2.1 找终点

先找当前使用的日志文件

`mysql>show master status;`

分析日志可以在mysql中用到 parge less 进行查看日志的分页显示

方法一：

先pager less

再 show binlog events in 'mysql-bin.000002'

方法二：

mysql>先 pager grep -i "drop database pit" -B 10

再 show binlog events in 'mysql-bin.000002'

确定终点是21107812

```
| mysql-bin.000003 | 21107781 | Xid           |       51 |   21107812 | COMMIT /* xid=2860 */
| mysql-bin.000003 | 21107812 | Gtid          |       51 |   21107889 | SET @@SESSION.GTID_NEXT= '1bc7779a-48f0-11eb-bdae-000c29ef43a9:69'
| mysql-bin.000003 | 21107889 | Query         |       51 |   21107990 | drop database pit /* xid=2862 */
```

2.2 找起点

[root@db01 tmp]# grep "\-- CHANGE MASTER TO" /tmp/full_2300.sql

-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000003', MASTER_LOG_POS=21107531;

确定的起点是21107531

2.3 截取日志

mysqlbinlog --skip-gtids --start-position=21107812 --stop-position=21107531 /data/3306/binlog/mysql-bin.000003 >/tmp/bin.sql

3.恢复binlog二进制日志

mysql>set sql_log_bin=0;

mysql>source /tmp/bin.sql;

mysql>set sql_log_bin=1;

1.6 注意事项：

> 1、mysqldump在备份和恢复时都需要mysql实例启动为前提。
> 2、一般数据量级100G以内，大约15-45分钟可以备份成功，但恢复时间通常需要5-10倍时间，数据量级很大很大的时候（PB、EB）
> 3、mysqldump是覆盖形式恢复的方法
>
> 一般我们认为，在同数据量级，物理备份要比逻辑备份速度快.
> 逻辑备份的优势:
> 1、可读性强
> 2、压缩比很高

2.Mydumper&Myloader（一套工具）

2.1Mydumper介绍

> MySQL自身的mysqldump工具支持单线程工作，依次一个个导出多个表，没有一个并行的机制，这就使得它无法迅速的备份数据。
> mydumper作为一个实用工具，能够良好支持多线程工作，可以并行的多线程的从表中读入数据并同时写到不同的文件里，这使得它在处理速度方面快于传统的mysqldump。
> 其特征之一是在处理过程中需要对列表加以锁定，因此如果我们需要在工作时段执行备份工作，那么会引起DML阻塞。
> 但一般现在的MySQL都有主从，备份也大部分在从上进行，所以锁的问题可以不用考虑。这样，mydumper能更好的完成备份任务。

2.2Mydumper特性

多线程备份
因为是多线程逻辑备份，备份后会生成多个备份文件
备份时对MyISAM表施加FTWRL(FLUSH TABLES WITH READ LOCK),会阻塞DML语句
保证备份数据的一致性
支持文件压缩
支持导出binlog
支持多线程恢复
支持以守护进程模式工作，定时快照和连续二进制日志
支持将备份文件切块

2.3 Mydumper工作原理

mydumper主要流程概括
1、主线程 FLUSH TABLES WITH READ LOCK, 施加全局只读锁，以阻止DML语句写入，保证数据的一致性
2、读取当前时间点的二进制日志文件名和日志写入的位置并记录在metadata文件中，以供即使点恢复使用
3、N个（线程数可以指定，默认是4）dump线程 START TRANSACTION WITH CONSISTENT SNAPSHOT; 开启读一致的事务
4、dump non-InnoDB tables, 首先导出非事务引擎的表
5、主线程 UNLOCK TABLES 非 事务引擎备份完后，释放全局只读锁
6、dump InnoDB tables, 基于 事务导出InnoDB表
7、事务结束

备份所生成的文件
目录中包含一个metadata文件
记录了备份数据库在备份时间点的二进制日志文件名，日志的写入位置，
如果是在从库进行备份，还会记录备份时同步至主库的二进制日志文件及写入位置
每个表有两个备份文件：
database.table-schema.sql 表结构文件
database.table.sql 表数据文件
如果对表文件分片，将生成多个备份数据文件，可以指定行数或指定大小分片

2.4 参数介绍

mydumper
-B, --database 要备份的数据库，不指定则备份所有库
-T, --tables-list 需要备份的表，名字用逗号隔开
-o, --outputdir 备份文件输出的目录
-s, --statement-size 生成的insert语句的字节数，默认1000000
-r, --rows 将表按行分块时，指定的块行数，指定这个选项会关闭 --chunk-filesize
-F, --chunk-filesize 将表按大小分块时，指定的块大小，单位是 MB
-c, --compress 压缩输出文件
-e, --build-empty-files 如果表数据是空，还是产生一个空文件（默认无数据则只有表结构文件）
-x, --regex 是同正则表达式匹配 'db.table'
-i, --ignore-engines 忽略的存储引擎，用都厚分割
-m, --no-schemas 不备份表结构
-k, --no-locks 不使用临时共享只读锁，使用这个选项会造成数据不一致
--less-locking 减少对InnoDB表的锁施加时间（这种模式的机制下文详解）
-l, --long-query-guard 设定阻塞备份的长查询超时时间，单位是秒，默认是60秒（超时后默认mydumper将会退出）
--kill-long-queries 杀掉长查询 (不退出)
-b, --binlogs 导出binlog (使用的版本取消了这个参数)
-D, --daemon 启用守护进程模式，守护进程模式以某个间隔不间断对数据库进行备份
-I, --snapshot-interval dump快照间隔时间，默认60s，需要在daemon模式下
-L, --logfile 使用的日志文件名(mydumper所产生的日志), 默认使用标准输出
--tz-utc 跨时区是使用的选项，不解释了
--skip-tz-utc 同上
--use-savepoints 使用savepoints来减少采集metadata所造成的锁时间，需要SUPER 权限
--success-on-1146 Not increment error count and Warning instead of Critical in case of table doesn't exist

> –h, ––host 连接的主机名
> –u, ––user 备份所使用的用户
> –p, ––pass 密码
> –P, ––port 端口
> –S, ––socket 使用socket通信时的socket文件
> –t, ––threads 开启的备份线程数，默认是4
> –C, ––compress–protocol 压缩与mysql通信的数据
> –V, ––version 显示版本号
> –v, ––verbose 输出信息模式, 0 = silent, 1 = errors, 2 = warnings, 3= info, 默认为 2
>
> # myloader
> –d, ––directory 备份文件的文件夹
> –q, ––queries–per–transaction 每次事物执行的查询数量，默认是1000
> –o, ––overwrite–tables 如果要恢复的表存在，则先drop掉该表，使用该参数，需要备份时候要备份表结构
> –B, ––database 需要还原的数据库
> –e, ––enable–binlog 启用还原数据的二进制日志
> –h, ––host 主机
> –u, ––user 还原的用户
> –p, ––pass 密码
> –P, ––port 端口
> –S, ––socket socket文件
> –t, ––threads 还原所使用的线程数，默认是4
> –C, ––compress–protocol 压缩协议
> –V, ––version 显示版本
> –v, ––verbose 输出模式, 0 = silent, 1 = errors, 2 = warnings， 3 = info, 默认为2

2.5 Mydumper安装

Bash | Copy

```
[root@db01 opt]# yum -y install mydumper-0.9.5-2.el7.x86_64.rpm
```

2.6 Mydumper的使用

2.6.1 Mydumper备份全库

Bash | Copy

```
mydumper -u root -p 123  -S /tmp/mysql.sock -o /data/backup
```

备份目录下结构是 metadata文件 + 多张表独立进行备份（每一张表都有两个文件组成）
metadata文件存放 备份时间 使用的那个日志文件 pos位置点 GTID号

```
[root@db01 backup]# cat metadata
Started dump at: 2021-04-28 23:19:55
SHOW MASTER STATUS:
        Log: binlog.000003
        Pos: 768
        GTID:

Finished dump at: 2021-04-28 23:19:56
[root@db01 backup]#
```

```
-rw-r--r-- 1 root root    319 4月  28 23:19 world.cry-schema.sql
-rw-r--r-- 1 root root    197 4月  28 23:19 world.cry.sql
```

world.cry–schema.sql 表结构文件，存放建表语句

```
[root@db01 backup]# cat world.cry-schema.sql
/*!40101 SET NAMES binary*/;
/*!40014 SET FOREIGN_KEY_CHECKS=0*/;

/*!40103 SET TIME_ZONE='+00:00' */;
CREATE TABLE `cry` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(20) NOT NULL,
  `sal` int NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
[root@db01 backup]#
```

1 建表语句

world.cry.sql 表数据文件，存放insert插入的sql语句

```
[root@db01 backup]# cat world.cry.sql
/*!40101 SET NAMES binary*/;
/*!40014 SET FOREIGN_KEY_CHECKS=0*/;
/*!40103 SET TIME_ZONE='+00:00' */;
INSERT INTO `cry` VALUES
(1,"a",3000),
(2,"b",3000),
(3,"c",3000),
(4,"d",4000),
(5,"e",1000);
[root@db01 backup]#
```

2.6.2 Mydumper备份单库

Bash | Copy

```
mydumper -u root -p 123  -B oldguo -S /tmp/mysql.sock   -o /data/backup
```

2.6.3 Mydumper备份多张表

Bash | Copy

```
 mydumper -u root -p 123  -B oldguo -T t1,stu1  -S /tmp/mysql.sock   -o
```

```
[root@db01 backup]# mydumper -u root -p 123  -B oldguo -T t1,stu1  -S /tmp/mysql.sock   -o /data/backup
[root@db01 backup]# ll
总用量 20
-rw-r--r-- 1 root root 133 4月  29 00:48 metadata
-rw-r--r-- 1 root root 130 4月  29 00:48 oldguo-schema-create.sql
-rw-r--r-- 1 root root 276 4月  29 00:48 oldguo.stu1-schema.sql
-rw-r--r-- 1 root root 340 4月  29 00:48 oldguo.t1-schema.sql
-rw-r--r-- 1 root root 180 4月  29 00:48 oldguo.t1.sql
[root@db01 backup]#
```

2.6.4 Mydumper备份单表数据，不备份表结构

Bash | Copy

```
mydumper -u root -p 123  -B oldguo -T t1 -m  -S /tmp/mysql.sock   -o /da
```

```
[root@db01 backup]# mydumper -u root -p 123  -B oldguo -T t1 -m  -S /tmp/mysql.sock   -o /data/backup
[root@db01 backup]# ll
总用量 8
-rw-r--r-- 1 root root 133 4月  29 00:48 metadata
-rw-r--r-- 1 root root 180 4月  29 00:48 oldguo.t1.sql
[root@db01 backup]#
```

2.6.5 Mydumper备份单表数据并进行压缩

Bash | Copy

```
mydumper -u root -p 123  -B oldguo -c（压缩协议 默认gzip）  -S /tmp/mysql.s
```

```
[root@db01 backup]# mydumper -u root -p 123  -B oldguo -c -S /tmp/mysql.sock   -o /data/backup
[root@db01 backup]# ll
总用量 56
-rw-r--r-- 1 root root 133 4月  29 00:45 metadata
-rw-r--r-- 1 root root 209 4月  29 00:45 oldguo.a-schema.sql.gz
-rw-r--r-- 1 root root 165 4月  29 00:45 oldguo.a.sql.gz
-rw-r--r-- 1 root root 229 4月  29 00:45 oldguo.b-schema.sql.gz
-rw-r--r-- 1 root root 173 4月  29 00:45 oldguo.b.sql.gz
-rw-r--r-- 1 root root 133 4月  29 00:45 oldguo-schema-create.sql.gz
-rw-r--r-- 1 root root 257 4月  29 00:45 oldguo.stu1-schema.sql.gz
-rw-r--r-- 1 root root 257 4月  29 00:45 oldguo.stu2-schema.sql.gz
-rw-r--r-- 1 root root 357 4月  29 00:45 oldguo.stu3-schema.sql.gz
-rw-r--r-- 1 root root 444 4月  29 00:45 oldguo.stu5-schema.sql.gz
-rw-r--r-- 1 root root 459 4月  29 00:45 oldguo.stu6-schema.sql.gz
-rw-r--r-- 1 root root 256 4月  29 00:45 oldguo.stu-schema.sql.gz
-rw-r--r-- 1 root root 251 4月  29 00:45 oldguo.t1-schema.sql.gz
-rw-r--r-- 1 root root 158 4月  29 00:45 oldguo.t1.sql.gz
[root@db01 backup]#
```

2.7 Myloader 恢复

2.7.1 针对库

Bash | Copy

```bash
myloader -u root -p '123' -B test -d /data/backup/
```

2.7.2 针对表

Bash | Copy

```bash
myloader -u root -p '123' -B test -o tableA -d /data/backup/
```

3.逻辑导入导出 load data/mysqlimport

3.1 功能

> 1. 大数据量的录入
> 2. 异构迁移
>
> 数据源： MySQL 、异构平台导出的、造数工具生成文本类的文件。

3.2 模拟环境

Bash | Copy

```
  0.创建表
  create table test3(id int unsigned not null primary key auto_increme
nt,test varchar(100),test2 varchar(100));
  1.插入数据
  insert into test3(test,test2)
  values('a string','100.20'),('a string containing a , comma','102.2
0'),
  ('a string containing a " quote','102.20'),
  ('a string containing a '', quote and comma','102.20');
  2.查看创建完成的模拟表
  mysql> select * from test3;
+----+-------------------------------------------+--------+
| id | test                                      | test2  |
+----+-------------------------------------------+--------+
|  1 | a string                                  | 100.20 |
|  2 | a string containing a , comma             | 102.20 |
|  3 | a string containing a " quote             | 102.20 |
|  4 | a string containing a '', quote and comma | 102.20 |
+----+-------------------------------------------+--------+
```

3.3 语法

```
mysql> help load data;
Name: 'LOAD DATA'
Description:
Syntax:
LOAD DATA
    [LOW_PRIORITY | CONCURRENT] [LOCAL]
    INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
    [IGNORE number {LINES | ROWS}]
    [(col_name_or_user_var
        [, col_name_or_user_var] ...)]
    [SET col_name={expr | DEFAULT},
        [, col_name={expr | DEFAULT}] ...]

常用的语句: load data infile 'file.txt' into table tb_name;

load data语句加载的数据源可以是mysqldump导出的纯文本数据文件,
也可以是使用SELECT … INTO OUTFILE '/path/xx.txt';语句生成的单表纯文本数据文件,
或者其他的方式生成的txt（只 要生成的纯文本数据列按指定分隔符分割的纯文本数据文件即可
```

3.4 load data语句实列–常用的必选子句（导出+导入）

如果文本文件中的数据字段与表结构中的字段定义顺序相同，则直接使用如下语句载入即可

1.导出test3表数据

```
mysql> select * from test3 into outfile "/tmp/test3.txt";
ERROR 1290 (HY000): The MySQL server is running with the --secure-file-p
导出前要设置安全路径参数，否则报错
```

修改配置文件

```
[mysqld]
user=mysql
basedir=/usr/local/mysql
datadir=/data/3306/data
socket=/tmp/mysql.sock
transaction_isolation='read-committed'
autocommit=0
secure-file-priv=/tmp
[mysql]
socket=/tmp/mysql.sock
```

```
mysql> select * from test3 into outfile "/tmp/test3.txt"; (每次导出会生成新
Query OK, 4 rows affected (0.00 sec)
```

2. 查看导出的数据

```
[root@db01 tmp]# cat test3.txt
5 a string  100.20
6 a string containing a , comma 102.20
7 a string containing a " quote 102.20
8 a string containing a ", quote and comma  102.20
```

3.导入数据到test4

模拟生成一个表结构和test3一样的test4表

```
mysql> create table test4 like test3;
```

导入/tmp/test3.txt文件数据到test4表中

```
mysql> load data infile '/tmp/test3.txt' into table test4;
Query OK, 4 rows affected (0.00 sec)
Records: 4  Deleted: 0  Skipped: 0  Warnings: 0

mysql> select * from test4;
+----+-----------------------------------------+--------+
| id | test                                    | test2  |
+----+-----------------------------------------+--------+
|  5 | a string                                | 100.20 |
|  6 | a string containing a , comma           | 102.20 |
|  7 | a string containing a " quote           | 102.20 |
|  8 | a string containing a ", quote and comma | 102.20 |
+----+-----------------------------------------+--------+
```

3.5 load data语句实列–设置字段顺序导出导入

如果文本文件中的数据字段与表结构中的字段定义顺序不同，则使用如下语句指定载入表中的字段顺序

导出文本，导出文本时不使用select *，而是使用具体的字段，把顺序稍微调整一下

1.导出test3表指定表结构字段定义顺序

```
mysql> select id,test2,test from test3 into outfile "/tmp/test3.txt";
```

2.查看导出的文件内容

```
[root@db01 tmp]# cat test3.txt  列信息会根据我们指定的进行排序导出 id,test2,t
5 100.20  a string
6 102.20  a string containing a , comma
7 102.20  a string containing a " quote
8 102.20  a string containing a ", quote and comma
```

3.导入数据时，要查看到入的表的表结构 ，根据表结构进行指定字段顺序导入

```
load data infile '/tmp/test3.txt' into table test3(id,test2,test);
```

3.6 load data语句的关键字

3.6.1 远程导出导入（LOCAL关键字）----不常用

> 说明： 如果要载入的文本文件不在mysql server数据库本身的本地磁盘，客户端也不是从mysql
> server本机登录的，则需要使用local关键字，指定mysql server从client host本地加载该文件，需要mysql server端使用local_infile=true(或者设置为1，不设置时默认为1)启动，以及

客户端连接mysql server时也使用local_infile=true(或者设置为1，不指定时默认为1)连接才能使用，server和client必须都开启这个参数才能使用local关键字，任意一个关闭都不能使用.

```
# 登录到数据库，重新导出表数据到文本，并发送到10.0.0.52服务器
system rm -f /tmp/test3.txt;
select * from test3 into outfile "/tmp/test3.txt";
system scp /tmp/test3.txt 10.0.0.52:/tmp/test3.txt

# 登录到10.0.0.52服务器，远程连接10.0.0.51数据库
mysql -uroot -p123 h10.0.0.51
mysql> use oldguo
mysql> system ls -lh /tmp/test3.txt;
-rw-r--r-- 1 root root 146 May 3 11:11 /tmp/test3.txt
mysql> system cat /tmp/test3.txt;
2 a string 100.20
4 a string containing a , comma 102.20
6 a string containing a " quote 102.20
8 a string containing a ", quote and comma 102.20
mysql> show variables like '%local%';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| local_infile | ON |
+---------------+-------+
mysql> set global local_infile=OFF; #关闭server端的local_infile参数
mysql> truncate test3;
mysql> load data local infile '/tmp/test3.txt' into table test3; #执行导入数据时报错了
ERROR 1148 (42000): The used command is not allowed with this MySQL version
mysql> set global local_infile=ON; #重新打开server端的local_infile参数
mysql> load data local infile '/tmp/test3.txt' into table test3; #导入成功
Records: 4 Deleted: 0 Skipped: 0 Warnings: 0
mysql> select * from test3; #查看数据，可以看到数据已成功导入表
+----+-----------------------------------------+--------+
| id | test | test2 |
+----+-----------------------------------------+--------+
| 2 | a string | 100.20 |
| 4 | a string containing a , comma | 102.20 |
| 6 | a string containing a " quote | 102.20 |
| 8 | a string containing a ", quote and comma | 102.20 |
+----+-----------------------------------------+--------+
4 rows in set (0.00 sec)

# 对于客户端连接server时使用local_infile=0参数，在执行导入数据时也会报相同的错误：
mysql -uroot -p123 -h10.0.0.51 --local-infile=0
```

3.6.2 导出导入主键冲突问题（REPLACE与IGNORE关键字）

REPLACE和IGNORE关键字控制对唯一键值冲突行的处理：
如果指定了REPLACE关键字，则输入行将覆盖现有行。换句话说，与主键或唯一索引冲突的数据行将被执
行覆盖写入，如果同时使用了local关键字，则与没有使用local关键字行为相同
如果指定了IGNORE关键字，则与唯一键值冲突的数据行将被丢弃，如果同时使用了local关键字，则与
没有使用local关键字行为相同

3.6.3 fields关键字应用

①字段(列)分隔符应用(TERMINATED) 常用

1.导出指定分割符

> load工具默认分割符是\t(tab键)，使用子句 fields terminated by 'string' 指定，其中string代表指定的字段分隔符。

举例说明

```
Bash | Copy
select * from test3 into outfile "/tmp/test3.txt" FIELDS TERMINATED BY '

逗号为分割符
[root@db01 tmp]# cat test3.txt
5,a string,100.20
6,a string containing a \, comma,102.20
7,a string containing a " quote,102.20
8,a string containing a "\, quote and comma,102.20
```

2.导入指定分割符

举例说明

将linux系统中/etc/passwd目录下的信息导入到数据库中

我们先查看/etc/passwd目录下的信息是以 （: ）冒号为分隔符的

```
Bash | Copy
[root@db01 tmp]# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:999:998:User for polkitd:/:/sbin/nologin
tss:x:59:59:Account used by the trousers package to sandbox the tcsd da
abrt:x:173:173::/etc/abrt:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
mysql:x:1000:1000::/home/mysql:/bin/bash
```

在数据库中创建相对应列的表

```
Bash | Copy
mysql> create table passwd (user varchar(200), passwd varchar(200), u v
archar(200), g varchar(200), users varchar(200), homedir varchar(200),
bash varchar(200));
```

导入数据

```
0.将导入数据存放在安全路径下
[root@db01 ~]# cp /etc/passwd /tmp 因为我们设置的安全路径是/tmp路径下

1.指定字符串导入数据到mysql中
mysql> load data infile '/tmp/passwd' into table passwd FIELDS TERMINAT
Query OK, 24 rows affected (0.00 sec)
Records: 24  Deleted: 0  Skipped: 0  Warnings: 0

2.查看
mysql> select * from passwd;
+----------------+--------+-------+-------+-------------------------
| user           | passwd | u     | g     | users
+----------------+--------+-------+-------+-------------------------
| root           | x      | 0     | 0     | root
| bin            | x      | 1     | 1     | bin
| daemon         | x      | 2     | 2     | daemon
| adm            | x      | 3     | 4     | adm
| lp             | x      | 4     | 7     | lp
| sync           | x      | 5     | 0     | sync
| shutdown       | x      | 6     | 0     | shutdown
| halt           | x      | 7     | 0     | halt
| mail           | x      | 8     | 12    | mail
| operator       | x      | 11    | 0     | operator
| games          | x      | 12    | 100   | games
| ftp            | x      | 14    | 50    | FTP User
| nobody         | x      | 99    | 99    | Nobody
| systemd-network | x      | 192   | 192   | systemd Network Management
| dbus           | x      | 81    | 81    | System message bus
| polkitd        | x      | 999   | 998   | User for polkitd
| tss            | x      | 59    | 59    | Account used by the trouse
| abrt           | x      | 173   | 173   |
| sshd           | x      | 74    | 74    | Privilege-separated SSH
| postfix        | x      | 89    | 89    |
| rpc            | x      | 32    | 32    | Rpcbind Daemon
| rpcuser        | x      | 29    | 29    | RPC Service User
| nfsnobody      | x      | 65534 | 65534 | Anonymous NFS User
| mysql          | x      | 1000  | 1000  |
+----------------+--------+-------+-------+-------------------------
```

② 字段引用符(ENCLOSED) ----常用

介绍

> 如果加optionally选项则只用在char、varchar和text等字符型字段上，数值类型会忽略使用引用符，如果不指定该子句，则默认不使用引用符，
> 使用子句fields [optionally] enclosed by 'char'指定，其中char代表指定的字段引用符

举例说明

导出数据

```
  mysql>  select * from test3 into outfile "/tmp/test3.txt" FIELDS ENCL
Query OK, 4 rows affected (0.01 sec)

[root@db01 tmp]# cat test3.txt
"5" "a string"  "100.20"
"6" "a string containing a , comma" "102.20"
"7" "a string containing a \" quote"  "102.20"
"8" "a string containing a \", quote and comma" "102.20"

联合使用 指定分割符+引用符
mysql>  select * from test3 into outfile "/tmp/test3.txt" FIELDS  TERMI
[root@db01 tmp]# cat test3.txt
"5","a string","100.20"
"6","a string containing a , comma","102.20"
"7","a string containing a \" quote","102.20"
"8","a string containing a \", quote and comma","102.20"
```

③ 转义字符 （ escaped ）

介绍

> 默认为反斜杠，使用子句 fields escaped by 'char' 指定，其中char代表指定的转义字符

举例说明

将数据中的转译字符用点号代替

```
mysql> select * from test3 into outfile "/tmp/test3.txt" fields escaped

[root@db01 tmp]# cat test3.txt
5 a string  100..20
6 a string containing a , comma 102..20
7 a string containing a " quote 102..20
8 a string containing a ", quote and comma  102..20
```

3.6.4 LINES 关键字应用

① 行前缀字符串

介绍

> 通过子句LINES STARTING BY 'xxx'; 改变每行前缀字符串

举例说明

导出

```
导出后的数据前缀加上xxx
mysql> select * from test3 into outfile "/tmp/test3.txt" LINES STARTING

[root@db01 tmp]# cat test3.txt
xxx5  a string  100.20
xxx6  a string containing a , comma 102.20
xxx7  a string containing a " quote 102.20
xxx8  a string containing a ", quote and comma  102.20
```

导入

```
导入数据前缀是xxx,导入mysql中取消前缀
清空表中数据
mysql> truncate test3;

mysql> load data infile "/tmp/test3.txt" into table test3 LINES STARTIN
Query OK, 4 rows affected (0.00 sec)
Records: 4  Deleted: 0  Skipped: 0  Warnings: 0

mysql> select * from test3;
+----+-----------------------------------------+--------+
| id | test                                    | test2  |
+----+-----------------------------------------+--------+
|  5 | a string                                | 100.20 |
|  6 | a string containing a , comma           | 102.20 |
|  7 | a string containing a " quote           | 102.20 |
|  8 | a string containing a ", quote and comma | 102.20 |
+----+-----------------------------------------+--------+
```

② 行结束符

介绍

> linux下默认为\n，windows下默认为\r .使用子句lines terminated by 'string' 指定，其中string代表指定的换行符

举例说明

Bash | Copy

```
# 指定换行符为\r\n导出数据
system rm -f /tmp/test3.txt;
select * from test3 into outfile "/tmp/test3.txt" lines terminated by
'\r\n';
# 现在，把数据重新导入表，从下面的结果中可以看到，导入表中的数据正确
truncate test3;
load data infile "/tmp/test3.txt" into table test3 lines terminated b
y '\r\n';

select * from test3;
+----+-----------------------------------------+--------+
| id | test | test2 |
+----+-----------------------------------------+--------+
| 2 | a string | 100.20 |
| 4 | a string containing a , comma | 102.20 |
| 6 | a string containing a " quote | 102.20 |
| 8 | a string containing a ", quote and comma | 102.20 |
| 10 | \t | 102.20 |
| 14 | \t | 102.20 |
+----+-----------------------------------------+--------+
```

3.6.5 FIELDS和LINES注意事项

> MySQL中反斜杠是SQL语句中特殊字符的转义字符，因此在sql语句中碰到特殊字符时，您必须指定一个或者两个反斜杠来为特殊字符转义（如在mysql中或者一些其他程序中，\n代表换行符，\t代表制表符，\代表转义符，那么需要使用\t来转义制表符，\n来转义换行符，\来转义转义符本身，这样才能正确写入数据库或者生成导出的数据文本,使用FIELDS ESCAPED BY子句指定转义符.
>
> 特殊字符列表如下
> \0 ASCII NUL (X'00') 字符
> \b 退格字符
> \n 换行符
> \r 回车符
> \t 制表符
> \Z ASCII 26 (Control+Z)
> \N NULL值，如果转义符值为空，则会直接导出null字符串作为数据，这在导入时将把null作为数据
>
> 导入
> 例如： 数据中包含了ENCLOSED BY '"'子句指定字段引用符号，则与字段引用符号相同数据字符也会
> 被自动添加一个反斜杠进行转义（如果转义符指定为空，则可能会导致数据在导入时无法正确解析）。
> 例如:
> select * from test3 into outfile "/tmp/test3.txt" FIELDS OPTIONALLY
> enclosed BY '"';
> system cat /tmp/test3.txt;
> 2 "a string" "100.20"
> 4 "a string containing a , comma" "102.20"
> 6 "a string containing a \" quote" "102.20"
> 8 "a string containing a \", quote and comma" "102.20" # 可以看到与字段引用符相同的符号数据被转义了

3.6.6 IGNORE number {LINES | ROWS}子句

介绍

> 忽略输入文件中的前number行数据，使用子句ignore number lines指定忽略文本的前number行，在某些情况下生成的文本（如：mysql –e "select ...." > xx.txt中）带有字段名称，在导入时会 把这一行字段名称也当作数据，所以需要忽略掉这行字段名称

举例说明

```
模拟环境
[root@db01 tmp]# cat test3.txt
id      test            test2       导入数据不导入这一行
xxx5  a string  100.20
xxx6  a string containing a , comma 102.20
xxx7  a string containing a " quote 102.20
xxx8  a string containing a ", quote and comma  102.20

mysql> load data infile "/tmp/test3.txt" into table test3  ignore 1 lir
Query OK, 4 rows affected (0.00 sec)
Records: 4  Deleted: 0  Skipped: 0  Warnings: 0

mysql> select * from test3;
+----+-----------------------------------------+--------+
| id | test                                    | test2  |
+----+-----------------------------------------+--------+
|  5 | a string                                | 100.20 |
|  6 | a string containing a , comma           | 102.20 |
|  7 | a string containing a " quote           | 102.20 |
|  8 | a string containing a ", quote and comma | 102.20 |
+----+-----------------------------------------+--------+
```

3.6.7 SET col_name = expr,...子句

介绍

> 将列做一定的数值转换后再加载，使用子句set col_name = expr,.. 指定，要注意：col_name必须为表中真实的列名，expr可以是任意的表达式或者子查询，只要返回的数据结果值能对应上表中的字段数据定义类型即可，注意，非set语句生成的列名，必须使用括号括起来，否则报语法错误。

举例说明

```
1.首先因为版本的问题，sql_mode限制空行应该有null,多了一列我们插入的test3列 所以我
查看默认sql_mode
 mysql> select @@sql_mode;
+-------------------------------------------------------------------------
| @@sql_mode
+-------------------------------------------------------------------------
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,E
+-------------------------------------------------------------------------
修改sql_mode
set global sql_mode='NO_ZERO_IN_DATE,NO_ZERO_DATE';
重新登入查看
mysql> select @@sql_mode;
+-----------------------------+
| @@sql_mode                  |
+-----------------------------+
| NO_ZERO_IN_DATE,NO_ZERO_DATE |
+-----------------------------+
2.或者使用local关键字强制进行截断最后一个字段的null值列进行导入
需要进行配置文件的修改，在服务器标签和客户端标签下都加上secure-file-priv=/tmp
重启数据库生效
3.导入数据库信息，并对信息进行函数运算
mysql> load data local infile "/tmp/test3.txt" into table test3 (id,tes
Query OK, 5 rows affected, 3 warnings (0.00 sec)
Records: 5  Deleted: 0  Skipped: 0  Warnings: 3

mysql> select * from test3;
+----+-----------------------------------------+--------+-------+
| id | test                                    | test2  | test3 |
+----+-----------------------------------------+--------+-------+
|  1 | NULL                                    | NULL   | 10    |
|  5 | a string                                | 100.20 | 15    |
|  6 | a string containing a , comma           | 102.20 | 16    |
|  7 | a string containing a " quote           | 102.20 | 17    |
|  8 | a string containing a ", quote and comma | 102.20 | 18    |
+----+-----------------------------------------+--------+-------+
```

3.7 批量load和dump

3.7.1使用mysqldump批量导出

mysqldump批量导出会用到的参数

> 语法：`mysqldump -u username -p'xxx' -T target_dir db_name tb_name [option];`
> 其中option参数是以下几种可选参数：
> --fields-terminated-by 'string' 字段分隔符
> --fields-enclosed-by 'char' 字段引用符
> --fields-optionally-enclosed-by 'char' 字段引用符，只在char,varchar,text等字段类型上生效
> --fields-escaped-by 'char' 转义字符
> --lines-terminated-by 'string' 记录结束符，即换行符

mysqldump批量导出举例说明

第一步

设置安全路径

配置文件中设置安全路径

Bash | Copy

```
[mysqld]
secure-file-priv=/tmp
```

第二步

在安全路径下创建导出信息存放目录

Bash | Copy

```
mkdir /tmp/dump
chown mysql.mysql /tmp/dump -R
```

第三步

使用mysqldump批量导出mysql中的oldguo库

Bash | Copy

```
[root@db01 tmp]#
mysqldump -uroot -p123 --master-data=2 --single-transaction  -R -E --tr
命令        用户    密码                                          同时备份存储过程

mysqldump: [Warning] Using a password on the command line interface car

--
-- Position to start replication or point-in-time recovery from
--

-- CHANGE MASTER TO MASTER_LOG_FILE='binlog.000006', MASTER_LOG_POS=156

--
-- Dumping events for database 'oldguo'
--


--
-- Dumping routines for database 'oldguo'
--


```

第四步

查看路径下导出的信息

文件以.sql结尾的是表结构（建表语句），以.txt结尾的是表中数据

Bash | Copy

```
文件以.sql结尾的是表结构（建表语句），以.txt结尾的是表中数据
[root@db01 dump]# ll
总用量 64
-rw-r--r-- 1 root  root  1348 5月   1 00:00 a.sql
-rw-r----- 1 mysql mysql   54 5月   1 00:00 a.txt
-rw-r--r-- 1 root  root  1408 5月   1 00:00 b.sql
-rw-r----- 1 mysql mysql   88 5月   1 00:00 b.txt
-rw-r--r-- 1 root  root  1555 5月   1 00:00 passwd.sql
-rw-r----- 1 mysql mysql    0 5月   1 00:00 passwd.txt
-rw-r--r-- 1 root  root  1381 5月   1 00:00 stu1.sql
-rw-r----- 1 mysql mysql    0 5月   1 00:00 stu1.txt
-rw-r--r-- 1 root  root  1381 5月   1 00:00 stu2.sql
-rw-r----- 1 mysql mysql    0 5月   1 00:00 stu2.txt
-rw-r--r-- 1 root  root  1577 5月   1 00:00 stu3.sql
-rw-r----- 1 mysql mysql    0 5月   1 00:00 stu3.txt
-rw-r--r-- 1 root  root  1735 5月   1 00:00 stu5.sql
-rw-r----- 1 mysql mysql    0 5月   1 00:00 stu5.txt
-rw-r--r-- 1 root  root  1755 5月   1 00:00 stu6.sql
-rw-r----- 1 mysql mysql    0 5月   1 00:00 stu6.txt
-rw-r--r-- 1 root  root  1378 5月   1 00:00 stu.sql
-rw-r----- 1 mysql mysql    0 5月   1 00:00 stu.txt
-rw-r--r-- 1 root  root  1441 5月   1 00:00 t1.sql
-rw-r----- 1 mysql mysql   69 5月   1 00:00 t1.txt
-rw-r--r-- 1 root  root  1493 5月   1 00:00 test3.sql
-rw-r----- 1 mysql mysql    0 5月   1 00:00 test3.txt
-rw-r--r-- 1 root  root  1457 5月   1 00:00 test4.sql
-rw-r----- 1 mysql mysql  172 5月   1 00:00 test4.txt

```

3.7.2 使用mysqlimport批量导入（批量load）

语法：`mysqlimport [options] db_name textfile1 ...`

参数：

--fields-terminated-by=name 指定字段分隔符

--fields-enclosed-by=name 指定字段引用符

--fields-optionally-enclosed-by=name 指定字段引用符，但只在char、varchar、text字段上使用引用符

--fields-escaped-by=name 指定转义字符

--lines-terminated-by=name 指定行记录结束符（换行符）

--ignore-liens=number 忽略前几行

--low-priority 碰到有其他线程update操作操作的表与导入操作表相同时，延迟执行导入操作

-i, --ignore 如果碰到唯一键冲突就忽略冲突行导入

-r, --replace 如果碰到唯一键冲突就覆盖冲突行导入

-L, --local 从客户端主机加载数据文本文件

-C, --compress 在C/S模型之间使用压缩传输数据

-c, --columns=name 指定需要导入哪些列，与load data语句中一样需要指定表定义中真实的列名，有多个列名时使用逗号分隔

--default-character-set=name 设置使用该选项指定的字符集来解析文本文件中的内容

-h, --host 指定导入server的主机IP

-p, --password[=name] 指定导入server的用户密码

-P, --port=# 指定导入server的监听端口

--use-threads=# 指定多少个线程并发执行load data语句（实测单表时指定多线程时要比单线程要快，由于数据量小，测试出来的差别并不大，官方并没有说明是基于什么级别的并发，只写了一句：Load files in parallel using N threads，推测可能是基于类似mydumper的并发，但是多表导入时指定多线程就明显比单线程要快很多）

-u, --user=name 指定导入server的用户名

-d, --delete 指定导入操作之前先把表清空（实测重复导入时加了这个选项之后可以正常执行，通过解析binlog发现，发现binlog中记录的第二次和第一次导入的语句完全相同是，第二次导入时如果发现表中有冲突数据，就先执行的不带where条件的delete，所有表先delete掉，然后再执行load data语句导入数据，另外，当与replace一起使用时，忽略replace选项）

mysqlimport批量导入举例说明

第一步

模拟删除库操作，导入表结构

```
0.模拟删除库操作（oldboy库）
mysql> drop database oldguo;
1.将所有oldboy库导出的表结构文件找出
[root@db01 dump]# ll *.sql
-rw-r--r-- 1 root root 1348 5月   1 00:00 a.sql
-rw-r--r-- 1 root root 1408 5月   1 00:00 b.sql
-rw-r--r-- 1 root root 1555 5月   1 00:00 passwd.sql
-rw-r--r-- 1 root root 1381 5月   1 00:00 stu1.sql
-rw-r--r-- 1 root root 1381 5月   1 00:00 stu2.sql
-rw-r--r-- 1 root root 1577 5月   1 00:00 stu3.sql
-rw-r--r-- 1 root root 1735 5月   1 00:00 stu5.sql
-rw-r--r-- 1 root root 1755 5月   1 00:00 stu6.sql
-rw-r--r-- 1 root root 1378 5月   1 00:00 stu.sql
-rw-r--r-- 1 root root 1441 5月   1 00:00 t1.sql
-rw-r--r-- 1 root root 1493 5月   1 00:00 test3.sql
-rw-r--r-- 1 root root 1457 5月   1 00:00 test4.sql
2.通过文本或者别的方式进行编辑
source /tmp/dump/a.sql
source  /tmp/dump/b.sql
source  /tmp/dump/passwd.sql
source  /tmp/dump/stu1.sql
source  /tmp/dump/stu2.sql
source  /tmp/dump/stu3.sql
source  /tmp/dump/stu5.sql
source  /tmp/dump/stu6.sql
source  /tmp/dump/stu.sql
source  /tmp/dump/t1.sql
source  /tmp/dump/test3.sql
source  /tmp/dump/test4.sql
3.登陆数据库，创建oldboy数据库
mysql> create database oldboy charset=utf8mb4;
4.进行表结构导入
mysql> source /tmp/dump/a.sql
mysql> source  /tmp/dump/b.sql
mysql> source  /tmp/dump/passwd.sql
mysql> source  /tmp/dump/stu1.sql
mysql> source  /tmp/dump/stu2.sql
mysql> source  /tmp/dump/stu3.sql
mysql> source  /tmp/dump/stu5.sql
mysql> source  /tmp/dump/stu6.sql
mysql> source  /tmp/dump/stu.sql
mysql> source  /tmp/dump/t1.sql
mysql> source  /tmp/dump/test3.sql
mysql> source  /tmp/dump/test4.sql
```

第二步

导入表中数据就是以.txt结尾的文件

```
[root@db01 dump]# mysqlimport -uroot -p123 oldboy /tmp/dump/*.txt --fie
mysqlimport: [Warning] Using a password on the command line interface c
oldboy.a: Records: 6  Deleted: 0  Skipped: 0  Warnings: 0
oldboy.b: Records: 4  Deleted: 0  Skipped: 0  Warnings: 0
oldboy.passwd: Records: 0  Deleted: 0  Skipped: 0  Warnings: 0
oldboy.stu1: Records: 0  Deleted: 0  Skipped: 0  Warnings: 0
oldboy.stu2: Records: 0  Deleted: 0  Skipped: 0  Warnings: 0
oldboy.stu3: Records: 0  Deleted: 0  Skipped: 0  Warnings: 0
oldboy.stu5: Records: 0  Deleted: 0  Skipped: 0  Warnings: 0
oldboy.stu6: Records: 0  Deleted: 0  Skipped: 0  Warnings: 0
oldboy.stu: Records: 0  Deleted: 0  Skipped: 0  Warnings: 0
oldboy.t1: Records: 5  Deleted: 0  Skipped: 0  Warnings: 0
oldboy.test3: Records: 0  Deleted: 0  Skipped: 0  Warnings: 0
oldboy.test4: Records: 4  Deleted: 0  Skipped: 0  Warnings: 0
```

mysqlimport用法演示示例

```
## 单表
mysqlimport -uroot -p123 -h10.0.0.51 world /data/backup/city.txt
## 多表
mysqlimport -uroot -p123 -h10.0.0.51 --replace world /data/backup/*.txt
## 多表导入时可以使用参数--use-threads指定多个线程
mysqlimport -uroot -p123 -h10.0.0.51 --replace --use-threads=8 world /da
```

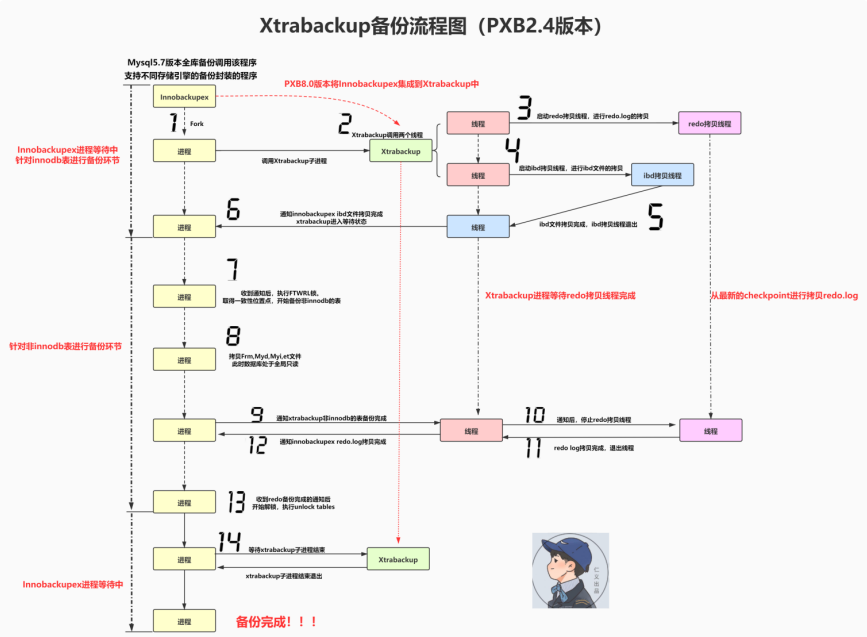
三.===物理备份工具===

1. Percona Xtrabackup 简称PXB,XBK,Xbackup（社区版免费）

1.0 原理图

pxb核心进程是Xtrabackup进程

PXB 8.0版本将innobackupex集成到了xtrabackup中



1.1介绍

> 物理备份工具. 备份恢复更快.
> 支持全备和增量备份

1.2 全备备份原理

8.0之前和8.0之后的区别在于使用不同的锁技术

> a. 8.0 之前
> 预备: 初始化进程和线程,分配专用内存.
> 备份:
> 0. 连接目标数据库,获取数据库信息.
> a. 当前LSN号码记录 ---> show engine InnoDB stutus ; 主要: Last checkpoint at NO.
> b. non-InnoDB 数据 :frm , csv , MYI ,MYI ----> FTWRL
> c. unlock tables;
> d. Copy Innodb 数据: ibd ibdata1 undo tmp ,可以允许DML
> e. 备份期间的redo截取和备份,并且记录LSN号.
> 结束:
> 记录binlog的当前位置点,将所有备份信息记录至专用日志文件中.
> b. 8.0 之前
> 预备: 初始化进程和线程,分配专用内存.
> 备份:
> 0. 连接目标数据库,获取数据库信息.

> a. 当前LSN号码记录 ---> show engine InnoDB stutus ; 主要: Last checkpoint at NO.
> b. LOCK INSTANCE FOR BACKUP; UNLOCK INSTANCE;
> c. Copy Innodb 数据: ibd ibdata1 undo tmp ,可以允许DML
> d. 备份期间的redo截取和备份,并且记录LSN号.
> 结束:
> 记录binlog的当前位置点,将所有备份信息记录至专用日志文件中.

1.3 增量备份原理

> a. 第一次增量备份,使用全备作为"参照物",把变化的数据页和备份过程中产生的redo保存.
> b. 往后所有的增量,都基于上一次备份作为参照物.把变化的数据页和备份过程中产生的redo保存.
> c. 增量备份只针对InnoDB表有效,所以在8.0之前,针对非InnoDB表,都是全备.

1.4 恢复过程

> 例如:备份策略为,FULL+inc1+inc2....
> a. prepare 全备 (CR)
> 应用redo前滚
> 应用undo回滚(省略)
> b. 合并所有增量到全备并且prepare
> 应用redo前滚
> 应用undo回滚(除了最后一次增量,这步省略)
> c. 合并后的全备prepare
> d. 恢复备份

1.5 PXB的版本兼容性(PXB本身不是oracle的产品)

> MySQL 5.6 ,5.7 : PXB 2.4版本
> MySQL 8.0.11 ~ 8.0.19 : PXB 8.0 稳定版.
> MySQL 8.0.20 : PXB 8.0.12+

1.6 全量备份（服务器端工具）

1.安装

`yum -y install percona-xtrabackup-80-8.0.13-1.el7.x86_64`

2.创建备份目录

mkdir -p /data/backup

3.修改配置文件

`vim /etc/my.cnf`

`[client]`

`socket=/tmp/mysql.sock`

4.全量备份（默认全备）

`xtrabackup --defaults-file=/etc/my.cnf --socket=/tmp/mysql.sock --user=root --password=123 --backup --target-dir=/data/backup/full`

Bash | Copy

```
1  或者：使用参数--datadir替换掉参数--defaults-file.
2  # xtrabackup --host=10.0.0.51 --user=root --password=123 --port=3306 --
   datadir=/data/crm/ --backup --target-dir=/data/backup/
```

1.7 全量备份后备份路径下的备份结构

```
-rw-r----- 1 root root      475 5月   6 02:11 backup-my.cnf
-rw-r----- 1 root root      156 5月   6 02:11 binlog.000010
-rw-r----- 1 root root       16 5月   6 02:11 binlog.index
-rw-r----- 1 root root     3680 5月   6 02:11 ib_buffer_pool
-rw-r----- 1 root root 12582912 5月   6 02:11 ibdata1
drwxr-x--- 2 root root       42 5月   6 02:11 mydb
drwxr-x--- 2 root root      143 5月   6 02:11 mysql
-rw-r----- 1 root root 25165824 5月   6 02:11 mysql.ibd
drwxr-x--- 2 root root       81 5月   6 02:11 oldboy
drwxr-x--- 2 root root     8192 5月   6 02:11 performance_schema
drwxr-x--- 2 root root       28 5月   6 02:11 sys
-rw-r----- 1 root root 11534336 5月   6 02:11 undo_001
-rw-r----- 1 root root 11534336 5月   6 02:11 undo_002
drwxr-x--- 2 root root      310 5月   6 02:11 world
-rw-r----- 1 root root       18 5月   6 02:11 xtrabackup_binlog_info
-rw-r----- 1 root root       95 5月   6 02:11 xtrabackup_checkpoints
-rw-r----- 1 root root      530 5月   6 02:11 xtrabackup_info
-rw-r----- 1 root root     2560 5月   6 02:11 xtrabackup_logfile
-rw-r----- 1 root root       39 5月   6 02:11 xtrabackup_tablespaces
```

1.8 模拟全量备份数据的恢复

1.模拟破坏

`[root@db01 ~]# pkill mysqld`

`[root@db01 ~]# rm -rf /data/3306/data/*`

`[root@db01 ~]# rm -rf /data/3306/logs/*`

`[root@db01 ~]# rm -rf /data/3306/binlog/*`

2.b 准备：(CR) 说明： 模拟CR过程，将redo前滚，undo回滚，让备份数据是一致状态

`[root@db01 ~]# xtrabackup --defaults-file=/etc/my.cnf --socket=/tmp/mysql.sock --user=root --password=123 --prepare --target-dir=/data/backup/full`

3.查看备份目录下生成的xbk文件，backup_type=full-prepared 表示已经准备完成

```
[root@db01 full]# cat xtrabackup_checkpoints
backup_type = full-prepared
from_lsn = 0
to_lsn = 21560235
last_lsn = 21560245
flushed_lsn = 0
[root@db01 full]#
```

4.恢复数据

`xtrabackup --defaults-file=/etc/my.cnf --socket=/tmp/mysql.sock --user=root --password=123 --copy-back --target-dir=/data/backup/full`

5.修改权限并启动数据库

`[root@db01 data]# chown -R mysql.mysql /data/*`

`[root@db01 data]# /etc/init.d/mysqld start`

1.9 增量备份

基于全备，备份数据页的变化，不是一个完整ibd文件。

====全备+增量备份操作======

1.9.0 首先规划备份目录，将全备和增量备份的目录分开

```
全量备份的目录为： mkdir -p /data/backup/full
增量备份的目录为： mkdir -p /data/backup/inc
```

1.9.1 模拟全备环境

```
0.创建库
mysql> create database pxe;
1.创建表
create table zb(id int ,k varchar(20)) charset=utf8mb4 engine=innodb;
2.表中插入数据
insert into zb values(1,'a');
3.查看表
mysql> select * from zb;
+------+------+
| id   | k    |
+------+------+
|    1 | a    |
+------+------+
```

1.9.2 全备操作

```
xtrabackup --defaults-file=/etc/my.cnf  --socket=/tmp/mysql.sock --user=
```

1.9.3 模拟增量备份环境

```
0.插入数据到pxe.zb表中，模拟全备后增量操作
mysql> insert into zb values(2,'b');

mysql> select * from zb;
+------+------+
| id   | k    |
+------+------+
|    1 | a    |
|    2 | b    |
+------+------+
```

1.9.4 增量备份操作

```
xtrabackup --defaults-file=/etc/my.cnf  --socket=/tmp/mysql.sock --user
=root --password=123   --backup --parallel=4 --target-dir=/data/backup/
inc --incremental-basedir=/data/backup/full
```

备份　并发数（取决于cpu）
设置基目录（基于全备）

====全备+增量恢复　操作======

1.9.5 准备全备份的日志

```
 xtrabackup --prepare --apply-log-only --target-dir=/data/backup/full
```

1.9.6 准备增量备份的日志(将增量合并到全备)

```
xtrabackup --prepare --apply-log-only --target-dir=/data/backup/full --i
```

1.9.7 全备份准备

```
xtrabackup --prepare --target-dir=/data/backup/full
```

1.9.8 拷回数据

```
Bash | Copy
1  xtrabackup --user=root --password=123  --datadir=/data/3306/data --copy-
```

1.9.9 修改权限

```
Bash | Copy
1  chown -R mysql:mysql /data/
```

1.9.10重启数据库

```
Bash | Copy
1  /etc/init.d/mysqld restart
```

2. MySQL Enterprise Backup 简称MEB （企业版 收费）

2.1 下载地址

https://edelivery.oracle.com/ <https://edelivery.oracle.com/>

2.2 全备及恢复

> mysqlbackup --backup_dir=/data/backup backup
> mysqlbackup --backup-dir=/data/backup/ apply-log
> mysqlbackup --backup-dir=/data/backup/ copy-back

2.3 单文件

> mysqlbackup --backup-image=/data/backup/img/bak.img --backup-dir=/data/backup/tmp backup-to-image

2.4 增量:

> mysqlbackup --incremental=optimistic --incremental-base=history:last_backup --backup-dir=/data/back/incr1 --backup-image=incremental_image1.bi backup-to-image
> mysqlbackup --incremental=optimistic --incremental-base=dir:/back/full1 --backup-dir=/back/incr1 --backup-image=incremental_image1.bi backup-to-image
> mysqlbackup --incremental=optimistic --incremental-base=dir:/back/incr1 --backup-dir=/back/incr2 --backup-image=incremental_image2.bi backup-to-image

2.5 恢复:

方法一:

> 使用copy-back-and-apply-log恢复全量
> 使用copy-back-and-apply-log恢复增量

方法二:

> 前滚全量
> mysqlbackup --backup-dir=/full-backup/ apply-log
> 前滚增量
> mysqlbackup --incremental-backup-dir=/incr-backup —backup-dir=/full-backup apply-incremental-backup

3. Mysql8.0（8.0.17）新特性clone plugin 克隆

3.1 介绍

> **1.本地克隆：**
> **启动克隆操作的MySQL服务器实例中的数据，克隆到同服务器或同节点上的一个目录里**
> **2.远程克隆：**
> **默认情况下，远程克隆操作会删除接受者(recipient)数据目录中的数据，并将其替换为捐赠者(donor)的克隆数据。您也可以将数据克隆到接受者的其他目录，以避免删除现有数据。(可选)**

3.2 原理

> # PAGE COPY
> 这里有两个动作
> 开启redo archiving功能，从当前点开始存储新增的redo log，这样从当前点开始所有的增量修改都不会丢失。同时上一步在page track的page被发送到目标端。确保当前点之前所做的变更一定发送到目标端。
> 关于redo archiving，实际上这是官方早就存在的功能，主要用于官方的企业级备份工具，但这里clone利用了该特性来维持增量修改产生的redo。
> 在开始前会做一次checkpoint， 开启一个后台线程log_archiver_thread()来做日志归档。当有新的写入时(notify_about_advanced_write_lsn)也会通知他去archive。
> 当 arch_log_sys处于活跃状态时，他会控制日志写入以避免未归档的日志被覆盖(log_writer_wait_on_archiver), 注意如果log_writer等待时间过长的话， archive任务会被中断掉.
> # Redo Copy
> 停止Redo Archiving, 所有归档的日志被发送到目标端，这些日志包含了从page copy阶段开始到现在的所有日志，另外可能还需要记下当前的复制点，例如最后一个事务提交时的binlog位点或者gtid信息，在系统页中可以找到。
> # Done
> 目标端重启实例，通过crash recovery将redo log应用上去。

3.3 限制

> 克隆插件受以下限制：
> *克隆操作期间不允许使用DDL。在选择数据源时应考虑这一限制。一种解决方法是使用专用的捐赠者实例，它可以容纳在克隆数据时被阻止的DDL操作。允许并发DML。
> *不能从其他MySQL服务器版本克隆实例。捐赠者和接收者必须具有相同的MySQL服务器版本。例如，不能在MySQL5.7和MySQL8.0\之间进行克隆。只有MySQL 8.0.17及更高版本才支持克隆插件。
> *一次只能克隆一个MySQL实例。不支持在单个克隆操作中克隆多个MySQL实例。
> *远程克隆操作不支持由指定的X协议端口*克隆插件不支持MySQL服务器配置的克隆。
> *克隆插件不支持二进制日志的克隆。*克隆插件只克隆存储在“InnoDB”中的数据。未克隆其他存储引擎数据。
> *不支持通过MySQL路由器连接到捐赠方MySQL服务器实例。
> *本地克隆操作不支持克隆使用绝对路径创建的常规表空间。克隆的表空间文件与源表空间文件具有相同的路径将导致冲突。

3.4 应用

====本地克隆应用（备份操作）====

1. 加载插件 mysql_clone.so

插件存放路径在 `/usr/local/mysql/lib/plugin`

```
加载插件两种方式
第一种:mysql> INSTALL PLUGIN clone SONAME 'mysql_clone.so';
第二种：配置文件
[mysqld]
plugin-load-add=mysql_clone.so clone=FORCE_PLUS_PERMANENT
查看
mysql>  SELECT PLUGIN_NAME, PLUGIN_STATUS FROM INFORMATION_SCHEMA.PLUGI
+-------------+---------------+
| PLUGIN_NAME | PLUGIN_STATUS |
+-------------+---------------+
| clone       | ACTIVE        |
+-------------+---------------+
```

2. 创建克隆专用用户

```
mysql> CREATE USER clone_user@'%' IDENTIFIED with mysql_native_password
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT BACKUP_ADMIN ON *.* TO 'clone_user';
Query OK, 0 rows affected (0.00 sec)
```

3. 创建备份目录且授权

```
[root@db01 3306]# mkdir -p /data/test/
[root@db01 3306]# chown -R mysql.mysql /data/
```

4. 本地克隆（备份操作）

```
0.克隆用户登陆数据库
[root@db01 plugin]# mysql -uclone_user  -ppassword
1.发起克隆操作
mysql> CLONE LOCAL DATA DIRECTORY = '/data/test/clonedir';  (/data/test/
Query OK, 0 rows affected (3.51 sec)
我们也可以把这两部结合起来，使用mysql -e参数 （写到脚本，任务计划都可）
[root@db01 plugin]# mysql -uclone_user  -ppassword -e  "CLONE LOCAL DATA
```

5. 查看克隆过程中的状态是否成功，和日志观测

```
使用mysql管理用户登陆
mysql> SELECT STAGE, STATE, END_TIME FROM performance_schema.clone_prog
+-----------+-------------+----------------------------+
| STAGE     | STATE       | END_TIME                   |
+-----------+-------------+----------------------------+
| DROP DATA | Completed   | 2021-05-07 08:44:13.158655 |
| FILE COPY | Completed   | 2021-05-07 08:44:14.800145 |
| PAGE COPY | Completed   | 2021-05-07 08:44:14.802611 |
| REDO COPY | Completed   | 2021-05-07 08:44:14.803463 |
| FILE SYNC | Completed   | 2021-05-07 08:44:16.641496 |
| RESTART   | Not Started | NULL                       | 远程克隆
| RECOVERY  | Not Started | NULL                       | 远程克隆
+-----------+-------------+----------------------------+

日志观测
mysql -uroot -p123
set global log_error_verbosity=3; 日志信息的详细程度，数字越大越详细（默认数字
通过 tail -f /data/3306/data/db01.err  监控克隆用户进行的本地克隆操作详细过程

2021-05-07T01:00:33.838550Z 18 [Note] [MY-013457] [InnoDB] Clone Begin
2021-05-07T01:00:33.838603Z 18 [Note] [MY-013457] [InnoDB] Clone Apply
2021-05-07T01:00:33.838612Z 18 [Note] [MY-013457] [InnoDB] Clone Apply
2021-05-07T01:00:33.838717Z 18 [Note] [MY-013458] [InnoDB] Clone set st
2021-05-07T01:00:33.838727Z 18 [Note] [MY-013458] [InnoDB] Clone Master
2021-05-07T01:00:33.838735Z 18 [Note] [MY-013458] [InnoDB] Clone State
2021-05-07T01:00:33.838741Z 18 [Note] [MY-013458] [InnoDB] Clone State
2021-05-07T01:00:33.838775Z 18 [Note] [MY-011845] [InnoDB] Clone Start
2021-05-07T01:00:33.838874Z 18 [Note] [MY-013458] [InnoDB] Clone State
2021-05-07T01:00:33.839175Z 18 [Note] [MY-013458] [InnoDB] Clone Apply
2021-05-07T01:00:33.839189Z 18 [Note] [MY-013458] [InnoDB] Clone Apply
2021-05-07T01:00:33.839214Z 18 [Note] [MY-011978] [InnoDB] Clone estima
2021-05-07T01:00:33.844810Z 18 [Note] [MY-013458] [InnoDB] Stage progre
2021-05-07T01:00:33.845542Z 18 [Note] [MY-013458] [InnoDB] Stage progre
2021-05-07T01:00:33.857761Z 18 [Note] [MY-013458] [InnoDB] Stage progre
2021-05-07T01:00:33.858257Z 18 [Note] [MY-013458] [InnoDB] Stage progre
2021-05-07T01:00:33.870583Z 18 [Note] [MY-013272] [Clone] Plugin Clone
2021-05-07T01:00:33.870619Z 18 [Note] [MY-013458] [InnoDB] Clone set st
2021-05-07T01:00:33.870630Z 18 [Note] [MY-013458] [InnoDB] Clone Master
2021-05-07T01:00:33.870637Z 18 [Note] [MY-013458] [InnoDB] Clone State
2021-05-07T01:00:33.870643Z 18 [Note] [MY-013458] [InnoDB] Clone State
2021-05-07T01:00:33.870663Z 18 [Note] [MY-011840] [InnoDB] Clone Start
2021-05-07T01:00:33.870675Z 18 [Note] [MY-011846] [InnoDB] Clone Stop
2021-05-07T01:00:33.870681Z 18 [Note] [MY-013458] [InnoDB] Clone State
2021-05-07T01:00:33.871965Z 18 [Note] [MY-013458] [InnoDB] Clone Apply
2021-05-07T01:00:33.871988Z 18 [Note] [MY-013458] [InnoDB] Clone Apply
2021-05-07T01:00:33.872005Z 18 [Note] [MY-013272] [Clone] Plugin Clone
2021-05-07T01:00:33.872015Z 18 [Note] [MY-013458] [InnoDB] Clone set st
2021-05-07T01:00:33.872021Z 18 [Note] [MY-013458] [InnoDB] Clone Master
2021-05-07T01:00:33.872027Z 18 [Note] [MY-013458] [InnoDB] Clone State
2021-05-07T01:00:33.872033Z 18 [Note] [MY-013458] [InnoDB] Clone State
2021-05-07T01:00:33.873230Z 18 [Note] [MY-011841] [InnoDB] Clone Stop
2021-05-07T01:00:33.873258Z 18 [Note] [MY-013458] [InnoDB] Clone State
2021-05-07T01:00:33.874183Z 18 [Note] [MY-013458] [InnoDB] Clone Apply
2021-05-07T01:00:33.874234Z 18 [Note] [MY-013458] [InnoDB] Clone Apply
2021-05-07T01:00:33.874345Z 18 [Note] [MY-013458] [InnoDB] Stage progre
2021-05-07T01:00:33.874373Z 18 [Note] [MY-013458] [InnoDB] Stage progre
2021-05-07T01:00:33.874385Z 18 [Note] [MY-013458] [InnoDB] Stage progre
2021-05-07T01:00:33.874400Z 18 [Note] [MY-013272] [Clone] Plugin Clone
2021-05-07T01:00:33.874425Z 18 [Note] [MY-013458] [InnoDB] Clone set st
2021-05-07T01:00:33.874432Z 18 [Note] [MY-013458] [InnoDB] Clone Master
2021-05-07T01:00:33.874481Z 18 [Note] [MY-013458] [InnoDB] Clone State
2021-05-07T01:00:33.874488Z 18 [Note] [MY-013458] [InnoDB] Clone State
2021-05-07T01:00:33.875305Z 18 [Note] [MY-013458] [InnoDB] Clone Apply
2021-05-07T01:00:33.875354Z 18 [Note] [MY-013458] [InnoDB] Clone Apply
2021-05-07T01:00:34.231805Z 18 [Note] [MY-013458] [InnoDB] Clone Apply
2021-05-07T01:00:34.384871Z 18 [Note] [MY-013458] [InnoDB] Clone Apply
2021-05-07T01:00:34.385004Z 18 [Note] [MY-013457] [InnoDB] Clone Apply
2021-05-07T01:00:34.385028Z 18 [Note] [MY-013457] [InnoDB] Clone End Ma

```

6.启动新实例(指定新备份的数据目录路径)

```
mysqld_safe --datadir=/data/test/clonedir --port=3333 --socket=/tmp/mysc
也可以设置一个配置文件 使用--defaul-file=参数指定配置文件启动
```

====远程克隆应用====

1.环境准备

准备两台虚拟机，一台db01 ,一台db02（可以链接克隆db01）

克隆完db02进行修改

1.修改db02的主机名和ip地址

2.删除db02mysql数据目录下的auto.cnf ,重新启动数据库实列，进行自动生成不同的uuid

3.配置文件将server_id修改（server_id=2）与db01不同

最后将两天虚拟机的数据库实列的日志进行清空

mysql> reset master;

2.加载克隆插件

在两台虚拟机节点上分别加载克隆功能插件

```
mysql> INSTALL PLUGIN clone SONAME 'mysql_clone.so';
```

3.创建克隆用户

捐赠者（db01）创建克隆用户

```
create user test1@'%' identified with mysql_native_password by '123';
grant backup_admin on *.* to test1@'%';
```

接收者（db02）创建克隆用户

```
create user test2@'%' identified with mysql_native_password by '123';
grant clone_admin on *.* to test2@'%';
```

4.开始克隆(在目标端db02节点上操作)

```
首先目标端（db02）使用mysql的root用户配置白名单列表
mysql> SET GLOBAL clone_valid_donor_list='10.0.0.51:3306';
其次目标端（db02）使用mysql的克隆用户test2进行克隆操作
mysql> CLONE INSTANCE FROM test1@'10.0.0.51':3306 IDENTIFIED BY '123';
```

5..监控克隆过程

```
在目标端db02节点上使用root用户监控查看
mysql>  SELECT STAGE, STATE, END_TIME FROM performance_schema.clone_pro
+-----------+-----------+----------------------------+
| STAGE     | STATE     | END_TIME                   |
+-----------+-----------+----------------------------+
| DROP DATA | Completed | 2021-05-08 10:58:48.075987 |
| FILE COPY | Completed | 2021-05-08 10:58:48.856430 |
| PAGE COPY | Completed | 2021-05-08 10:58:48.958887 |
| REDO COPY | Completed | 2021-05-08 10:58:49.059793 |
| FILE SYNC | Completed | 2021-05-08 10:58:50.294881 |
| RESTART   | Completed | 2021-05-08 10:58:53.969160 |
| RECOVERY  | Completed | 2021-05-08 10:58:54.941010 |
+-----------+-----------+----------------------------+
```

四.备份策略及备份检查

1.备份策略之备份工具的选择

1.1 逻辑备份

①mysqldump

使用场景：针对单节点数据量较小 或者 集群中拆分多节点（拆分后的每个节点数据量就不大了）

②mydumper

使用场景：数据量级别小，表个数比较多且表数量级不大

③replication

使用场景：延时从库

④mysqlbinlog

使用场景：先本地备份，再异地备份到其他节点上

⑤工具navicat

工具中有备份按钮

⑥ binlog2sql

辅助备份的工具

1.2 物理备份

① PXB(最主流的)

② MEB(大公司)

③ clone-plugin

④ 存储镜像

⑤ cp冷备

2.备份策略之备份时间的选择

凌晨！！！

3.备份策略之备份方案的选择

①针对数据量小的逻辑备份策略 使用每天全备+binlog

②针对数据量大的物理备份策略 使用每周全备+inc（增量备份）+binlog

③针对数据量极大策略 每月全备+inc（增量备份）

4.备份策略之恢复方案的选择

① 部分数据损坏

从全备+增量备份中提取+部分binlog 或者延时从库

② 大量数据损坏

全备+增量备份+binlog

5.备份策略之检查与演练

Bash | Copy

```
定期备份检查和恢复演练
a. 逻辑备份 ： 检查备份文件头部 、 抽查备份文件中核心的业务表存在性、数据量。
b. 物理备份 ： 备份目录下的日志。备份的大小。
c. 恢复演练 ： 按季度。
```

%E5%A4%87%E4%BB%BD%E6%81%A2%E5%A4%8D%20%E2%88%9A%20%7C%20%E4%B8%80.%E6%95%B0%E6%8D%AE%E6%8D%9F