链滴

MySQL 数据恢复

作者：XinyiZhang
原文链接：https://ld246.com/article/1601260540592
来源网站：链滴

直接恢复

直接恢复是使用备份文件做全量恢复,这是最常见的场景.

mysqldump 备份全量恢复

```
gzip -d backup.sql.gz | mysql -u<user> -h<host> -P<port> -p
```

xtrabackup 备份全量恢复

```
# 步骤一：解压（如果没有压缩可以忽略这一步）
innobackupex --decompress <备份文件所在目录>

# 步骤二：应用日志
innobackupex --apply-log <备份文件所在目录>

# 步骤三：复制备份文件到数据目录
innobackupex --datadir=<MySQL数据目录> --copy-back <备份文件所在目录>
```

基于时间点恢复

新建测试表

```
chengqm-3306>>show create table mytest.mytest \G;
*************************** 1. row ***************************
       Table: mytest
Create Table: CREATE TABLE `mytest` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `ctime` datetime DEFAULT NULL,
```

```
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

每秒插入一条数据

```
while true; do mysql -S /tmp/mysql.sock -e 'insert into mytest.mytest(ctime)values(now())';dat
;sleep 1;done
```

备份

```
mysqldump --opt --single-transaction --master-data=2 --default-character-set=utf8 -S /tmp
mysql.sock -A > backup.sql
```

找出备份时的日志位置

```
head -n 25 backup.sql | grep 'CHANGE MASTER TO MASTER_LOG_FILE'
-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000032', MASTER_LOG_POS=39654;
```

假设要恢复到 2019-08-09 11:01:54 这个时间点,我们从 binlog 中查找从 39654 到 019-08-09 11:01
54 的日志

```
mysqlbinlog --start-position=39654 --stop-datetime='2019-08-09 11:01:54' /data/mysql_log
mysql_test/mysql-bin.000032 > backup_inc.sql
tail -n 20 backup_inc.sql
```

当前数据条目数

```
-- 2019-08-09 11:01:54之前的数据条数
select count(*) from mytest.mytest where ctime < '2019-08-09 11:01:54';
+----------+
| count(*) |
+----------+
|      161 |
+----------+
1 row in set (0.00 sec)

-- 所有数据条数
chengqm-3306>>select count(*) from mytest.mytest;
+----------+
| count(*) |
+----------+
|      180 |
+----------+
1 row in set (0.00 sec)
```

然后执行恢复

```
# 全量恢复
mysql -S /tmp/mysql.sock < backup.sql

# 应用增量日志
mysql -S /tmp/mysql.sock < backup_inc.sql
```

检查数据

```
select count(*) from mytest.mytest;
+----------+
| count(*) |
+----------+
|      161 |
+----------+
1 row in set (0.00 sec)

chengqm-3306>>select * from mytest.mytest order by id desc limit 5;
+-----+---------------------+
| id  | ctime               |
+-----+---------------------+
| 161 | 2019-08-09 11:01:53 |
| 160 | 2019-08-09 11:01:52 |
| 159 | 2019-08-09 11:01:51 |
| 158 | 2019-08-09 11:01:50 |
| 157 | 2019-08-09 11:01:49 |
+-----+---------------------+
5 rows in set (0.00 sec)
```

恢复一个表

从 mysqldump 备份恢复一个表

```
# 提取某个库的所有数据
sed -n '/^-- Current Database: `mytest`/,/^-- Current Database:/p' backup.sql > backup_myte
t.sql

# 从库备份文件中提取建表语句
sed -e'/./{H;$!d;}' -e 'x;/CREATE TABLE `mytest`/!d;q' backup_mytest.sql > mytest_table_create.
ql

# 从库备份文件中提取插入数据语句
grep -i 'INSERT INTO `mytest`' backup_mytest.sql > mytest_table_insert.sql

# 恢复表结构到 mytest 库
mysql -u<user> -p mytest < mytest_table_create.sql

# 恢复表数据到 mytest.mytest 表
mysql -u<user> -p mytest <  mytest_table_insert.sql
```

从 xtrabackup 备份恢复一个表

MyISAM 表

假设从备份文件中恢复表 mytest.t_myisam.从备份文件中找到 t_myisam.frm, t_myisam.MYD, t_myisam.MYI 这 3 个文件,复制到对应的数据目录中,并授权.进入 MySQL.检查表情况

```
show tables;
+------------------+
| Tables_in_mytest |
+------------------+
```

```
| mytest          |
| t_myisam        |
+-----------------+
2 rows in set (0.00 sec)

chengqm-3306>>check table t_myisam;
+-----------------+-------+----------+----------+
| Table           | Op    | Msg_type | Msg_text |
+-----------------+-------+----------+----------+
| mytest.t_myisam | check | status   | OK       |
+-----------------+-------+----------+----------+
1 row in set (0.00 sec)
```

Innodb 表

假设从备份文件中恢复表 mytest.t_innodb,恢复前提是设置了 innodb_file_per_table = on

- 起一个新实例
- 在实例上建一个和原来一模一样的表
- 执行 alter table t_innodb discard tablespace; 删除表空间,这个操作会把 t_innodb.ibd 删除
- 从备份文件中找到 t_innodb.ibd 这个文件,复制到对应的数据目录,并授权
- 执行 alter table t_innodb IMPORT tablespace; 加载表空间
- 执行 flush table t_innodb;check table t_innodb; 检查表
- 使用 mysqldump 导出数据,然后再导入到要恢复的数据库

跳过误操作SQL

使用备份文件恢复跳过

不开启 GTID

使用备份文件恢复的步骤和基于时间点恢复的操作差不多,区别在于多一个查找 binlog 操作.举个例子我这里建立了两个表 a 和 b,每分钟插入一条数据，然后做全量备份,再删除表 b,现在要跳过这条 SQL.

删除表 b 后的数据库状态

```
show tables;
+-----------------+
| Tables_in_mytest |
+-----------------+
| a               |
+-----------------+
1 row in set (0.00 sec)
```

找出备份时的日志位置

```
head -n 25 backup.sql | grep 'CHANGE MASTER TO MASTER_LOG_FILE'
-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000034', MASTER_LOG_POS=38414;
```

找出执行了 drop table 语句的 pos 位置

```
mysqlbinlog -vv /data/mysql_log/mysql_test/mysql-bin.000034 | grep -i -B 3 'drop table `b`';
# at 120629
#190818 19:48:30 server id 83  end_log_pos 120747 CRC32 0x6dd6ab2a     Query    thread_id
29488    exec_time=0    error_code=0
SET TIMESTAMP=1566128910/*!*/;
DROP TABLE `b` /* generated by server */
```

从 binglog 中提取跳过这条语句的其他记录

```
# 第一条的 start-position 为备份文件的 pos 位置，stop-position 为 drop 语句的开始位置
mysqlbinlog -vv --start-position=38414 --stop-position=120629 /data/mysql_log/mysql_test
mysql-bin.000034 > backup_inc_1.sql

# 第二条的 start-position 为 drop 语句的结束位置
mysqlbinlog -vv --start-position=120747 /data/mysql_log/mysql_test/mysql-bin.000034 > ba
kup_inc_2.sql
```

恢复备份文件

```
mysql -S /tmp/mysql.sock < backup.sql
```

全量恢复后状态

```
show tables;
+------------------+
| Tables_in_mytest |
+------------------+
| a                |
| b                |
+------------------+
2 rows in set (0.00 sec)

chgnqm-3306>>select count(*) from a;
+----------+
| count(*) |
+----------+
|       71 |
+----------+
1 row in set (0.00 sec)
```

恢复增量数据

```
mysql -S /tmp/mysql.sock < backup_inc_1.sql
mysql -S /tmp/mysql.sock < backup_inc_2.sql
```

恢复后状态,可以看到已经跳过了 drop 语句

```
show tables;
+------------------+
| Tables_in_mytest |
+------------------+
| a                |
| b                |
```

```
+------------------+
2 rows in set (0.00 sec)

select count(*) from a;
+----------+
| count(*) |
+----------+
|      274 |
+----------+
1 row in set (0.00 sec)
```

开启 GTID

- 找出备份时的日志位置
- 找出执行了 drop table 语句的 GTID 值
- 导出备份时日志位置到最新的 binglog 日志
- 恢复备份文件
- 跳过这个 GTID

```
SET SESSION GTID_NEXT='对应的 GTID 值';
BEGIN; COMMIT;
SET SESSION GTID_NEXT = AUTOMATIC;
```

- 应用步骤 3 得到的增量 binlog 日志

使用延迟库跳过

不开启 GTID

使用延迟库恢复的关键操作在于 start slave until.我在测试环境搭建了两个 MySQL 节点,节点二延迟60秒,新建 a,b 两个表,每秒插入一条数据模拟业务数据插入.

```
localhost:3306 -> localhost:3307(delay 600)
```

当前节点二状态

```
show slave status \G;
...
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000039
          Read_Master_Log_Pos: 15524
               Relay_Log_File: mysql-relay-bin.000002
                Relay_Log_Pos: 22845
        Relay_Master_Log_File: mysql-bin.000038
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
...
       Seconds_Behind_Master: 600
...
```

当前节点二表

```
show tables;
+------------------+
| Tables_in_mytest |
+------------------+
| a                |
| b                |
+------------------+
```

在节点一删除表 b

```
drop table b;
Query OK, 0 rows affected (0.00 sec)

chengqm-3306>>show tables;
+------------------+
| Tables_in_mytest |
+------------------+
| a                |
+------------------+
1 row in set (0.00 sec)
```

接下来就是跳过这条 SQL 的操作步骤

延迟库停止同步

```
stop slave;
```

找出执行了 drop table 语句的前一句的 pos 位置

```
mysqlbinlog -vv /data/mysql_log/mysql_test/mysql-bin.000039 | grep -i -B 10 'drop table `b`';
...
# at 35134
#190819 11:40:25 server id 83  end_log_pos 35199 CRC32 0x02771167     Anonymous_GTID
ast_committed=132    sequence_number=133    rbr_only=no
SET @@SESSION.GTID_NEXT= 'ANONYMOUS'/*!*/;
# at 35199
#190819 11:40:25 server id 83  end_log_pos 35317 CRC32 0x50a018aa     Query    thread_id=
7155    exec_time=0    error_code=0
use `mytest`/*!*/;
SET TIMESTAMP=1566186025/*!*/;
DROP TABLE `b` /* generated by server */
```

延迟库同步到要跳过的 SQL 前一条

```
change master to master_delay=0;
start slave until master_log_file='mysql-bin.000039',master_log_pos=35134;
```

查看状态看到已经同步到对应节点

```
show slave status \G;
...
                  Master_Port: 3306
                Connect_Retry: 60
```

```
              Master_Log_File: mysql-bin.000039
          Read_Master_Log_Pos: 65792
...
            Slave_IO_Running: Yes
           Slave_SQL_Running: No
         Exec_Master_Log_Pos: 35134
...
              Until_Log_File: mysql-bin.000039
               Until_Log_Pos: 35134
```

跳过一条 SQL 后开始同步

```
set global sql_slave_skip_counter=1;
start slave;
```

查看同步状态,删除表 b 的语句已经被跳过

```
how slave status \G;
...
            Slave_IO_Running: Yes
           Slave_SQL_Running: Yes
...
1 row in set (0.00 sec)

show tables;
+------------------+
| Tables_in_mytest |
+------------------+
| a                |
| b                |
+------------------+
2 rows in set (0.00 sec)
```

开启 GTID

- 停止同步
- 找出执行了 drop table 语句的 GTID
- 执行这个 GTID 的事务

```
SET SESSION GTID_NEXT='对应的 GTID 值';
BEGIN; COMMIT;
SET SESSION GTID_NEXT = AUTOMATIC;
```

- 继续同步

闪回

闪回操作就是反向操作,比如执行了 delete from a where id=1,闪回就会执行对应的插入操作 insert into a (id,...) values(1,...),用于误操作数据,只对 DML 语句有效,且要求 binlog 格式设为 ROW.

binlog2sql

安装

```
wget https://github.com/danfengcao/binlog2sql/archive/master.zip -O binlog2sql.zip
unzip binlog2sql.zip
cd binlog2sql-master/

# 安装依赖
pip install -r requirements.txt
```

生成回滚SQL

```
python binlog2sql/binlog2sql.py --flashback \
-h<host> -P<port> -u<user> -p'<password>' -d<dbname> -t<table_name>\
--start-file='<binlog_file>' \
--start-datetime='<start_time>' \
--stop-datetime='<stop_time>' > ./flashback.sql

python binlog2sql/binlog2sql.py --flashback \
-h<host> -P<port> -u<user> -p'<password>' -d<dbname> -t<table_name> \
--start-file='<binlog_file>' \
--start-position=<start_pos> \
--stop-position=<stop_pos> > ./flashback.sql
```

MyFlash

安装

```
# 依赖(centos)
yum install gcc*  pkg-config glib2 libgnomeui-devel -y

# 下载文件
wget https://github.com/Meituan-Dianping/MyFlash/archive/master.zip -O MyFlash.zip
unzip MyFlash.zip
cd MyFlash-master

# 编译安装
gcc -w  `pkg-config --cflags --libs glib-2.0` source/binlogParseGlib.c  -o binary/flashback
mv binary /usr/local/MyFlash
ln -s /usr/local/MyFlash/flashback /usr/bin/flashback
```

使用

生成回滚语句

```
flashback --databaseNames=<dbname> --binlogFileNames=<binlog_file> --start-position=
start_pos> --stop-position=<stop_pos>
```

执行后会生成 binlog_output_base.flashback 文件,需要用 mysqlbinlog 解析出来再使用

```
mysqlbinlog -vv binlog_output_base.flashback | mysql -u<user> -p
```

限制

- binlog 格式必须为 row,且 binlog_row_image=full
- 仅支持5.6与5.7
- 只能回滚 DML(增、删、改)