一、mysqldump 备份结合 binlog 日志恢复

MySQL 备份一般采取全库备份加日志备份的方式，例如每天执行一次全备份，每小时执行一次二进制日志备份。这样在 MySQL 故障后可以使用全备份和日志备份将数据恢复到最后一个二进制日志备份前的任意位置或时间。

1、binlog 介绍

mysql 的二进制日志记录着该数据库的所有增删改的操作日志(前提是要在自己的服务器上开启 binlog)，还包括了这些操作的执行时间。为了显示这些二进制内容，我们可以使用 mysqlbinlog 命令来查看。

Binlog 的用途

1：主从同步

2：恢复数据库

开启 binary log 功能

通过编辑 my.cnf 中的 log-bin 选项可以开启二进制日志；形式如下：

log-bin [=DIR/[filename]]（配置文件中只写 log_bin 不写后面的文件名和路径时，默认存放在 /usr/local/mysql/data 目录下，文件名为主机名-bin.000001…命名）

其中，DIR 参数指定二进制文件的存储路径；filename 参数指定二级制文件的文件名，其形式为 filename.number，number 的形式为 000001、000002 等。每次重启 mysql 服务或运行 mysql> flush logs;都会生成一个新的二进制日志文件，这些日志文件的 number 会不断地递增。除了生成上述的文件外还会生成一个名为 filename.index 的文件。这个文件中存储所有二进制日志文件的清单又称为二进制文件的索引

配置保存以后重启 mysql 的服务器，用 mysql> show variables like 'log_bin';查看 bin-log 是否开启，如图：

```
mysql> show variables like 'log_bin';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | ON    |
+---------------+-------+
```

查看产生的 binary log 注：查看 binlog 内容是为了恢复数据

bin-log 因为是二进制文件，不能通过文件内容查看命令直接打开查看，mysql 提供两种方式查看方式，在介绍之前，我们先对数据库进行一下增删改的操作，否则 log 里边数据有点空。

#mysql -uroot -p -e "reset master"// 清空所有的二进制文件，从 00001 开始

#mysql -uroot -p -e "create database test"

#mysql -uroot -p -e "use test;create table tb1(id int primary key auto_increment,name varchar(20))"

#mysql -uroot -p -e "insert into test.tb1(name) values('lisi')"

#mysql -uroot -p -e "insert into test.tb1(name) values('zhangsan')"

重新开始一个新的日志文件

#mysql -uroot -p -e "flush logs"

#mysql -uroot -p -e "delete from test.tb1 where id=2"

#mysql -uroot -p -e "insert into test.tb1(name) values('tom')"

mysql -uroot -p -e "select * from test.tb1"

Enter password:

+----+------+

| id | name |

+----+------+

```
|    1 | lisi |
|    3 | tom    |
+----+------+
```

查看 MySQL Server 上的二进制日志

```
mysql> show binary logs;
+------------------+-----------+
| Log_name                | File_size |
+------------------+-----------+
| mysql-bin.000001 |          1087 |
| mysql-bin.000002 |           673 |
+------------------+-----------+
```

查看二进制日志信息的命令：

语法格式：SHOW BINLOG EVENTS[IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]

查看二进制日志中的事件

```
mysql> show binlog events;
```

默认显示可找到的第一个二进制日志文件中的事件，包含了日志文件名、事件的开始位置、事件类型、结束位置、信息等内容

```
+------------------+------+---------------+-----------+-------------+-----------------------------------------------------------------------------------------
------+
| Log_name              | Pos    | Event_type         | Server_id | End_log_pos | Info
|
+------------------+------+---------------+-----------+-------------+-----------------------------------------------------------------------------------------
------+
| mysql-bin.000001 |      4 | Format_desc       |             1 |             123 | Server ver: 5.7.13-log, Binlog ver:
4             |                  //此事件为格式描述事件
| mysql-bin.000001 |   123 | Previous_gtids |             1 |             154 |
|
| mysql-bin.000001 |   154 | Anonymous_Gtid |             1 |             219 | SET @@SESSION.GTID_NEXT=
'ANONYMOUS'                                                 |
| mysql-bin.000001 |   219 | Query                |             1 |             313 | create database test
|                                  //为查询事件
| mysql-bin.000001 |   313 | Anonymous_Gtid |             1 |             378 | SET @@SESSION.GTID_NEXT=
'ANONYMOUS'                                                 |
| mysql-bin.000001 |   378 | Query                |             1 |             520 | use `test`; create table tb1(id int
primary key auto_increment,name varchar(20)) |
| mysql-bin.000001 |   520 | Anonymous_Gtid |             1 |             585 | SET @@SESSION.GTID_NEXT=
'ANONYMOUS'                                                 |
| mysql-bin.000001 |   585 | Query                |             1 |             653 | BEGIN
|                                  //为查询事件，事务开始
| mysql-bin.000001 |   653 | Table_map          |             1 |             702 | table_id: 110 (test.tb1)
|                                  //为表映射事件
| mysql-bin.000001 |   702 | Write_rows         |             1 |             747 | table_id: 110 flags: STMT_END_F
|                                  //为我们执行的 insert 事件
```

| mysql-bin.000001 | 747 | Xid | 1 | 778 | COMMIT /* xid=2052 */
| //Xid 时间是自动提交事务的动作
mysql-bin.000001	778	Anonymous_Gtid	1	843	SET @@SESSION.GTID_NEXT= 'ANONYMOUS'
mysql-bin.000001	843	Query	1	911	BEGIN
mysql-bin.000001	911	Table_map	1	960	table_id: 110 (test.tb1)
mysql-bin.000001	960	Write_rows	1	1009	table_id: 110 flags: STMT_END_F
mysql-bin.000001	1009	Xid	1	1040	COMMIT /* xid=2055 */
mysql-bin.000001	1040	Rotate	1	1087	mysql-bin.000002;pos=4
//为日志轮换事件，是我们执行 flush logs 开启新日志文件引起的。

查看指定的二进制日志中的事件

```
mysql> show binlog events in 'mysql-bin.000002';
+------------------+-----+---------------+-----------+-------------+-------------------------------------+
| Log_name         | Pos | Event_type    | Server_id | End_log_pos | Info                                |
+------------------+-----+---------------+-----------+-------------+-------------------------------------+
| mysql-bin.000002 |   4 | Format_desc   |         1 |         123 | Server ver: 5.7.13-log, Binlog ver: 4 |
| mysql-bin.000002 | 123 | Previous_gtids |         1 |         154 |                                     |
| mysql-bin.000002 | 154 | Anonymous_Gtid |         1 |         219 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS' |
| mysql-bin.000002 | 219 | Query         |         1 |         287 | BEGIN                               |
| mysql-bin.000002 | 287 | Table_map     |         1 |         336 | table_id: 110 (test.tb1)            |
| mysql-bin.000002 | 336 | Delete_rows   |         1 |         385 | table_id: 110 flags: STMT_END_F     |
| mysql-bin.000002 | 385 | Xid           |         1 |         416 | COMMIT /* xid=2068 */               |
| mysql-bin.000002 | 416 | Anonymous_Gtid |         1 |         481 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS' |
| mysql-bin.000002 | 481 | Query         |         1 |         549 | BEGIN                               |
| mysql-bin.000002 | 549 | Table_map     |         1 |         598 | table_id: 110 (test.tb1)            |
| mysql-bin.000002 | 598 | Write_rows    |         1 |         642 | table_id: 110 flags: STMT_END_F     |
| mysql-bin.000002 | 642 | Xid           |         1 |         673 | COMMIT /* xid=2071 */               |
```

该命令还包含其他选项以便灵活查看

```
mysql> show binlog events in 'mysql-bin.000002' from 219 limit 1,3;     // limit 1，1 为偏移量
+------------------+-----+-------------+-----------+-------------+--------------------------------+
| Log_name         | Pos | Event_type  | Server_id | End_log_pos | Info
|
+------------------+-----+-------------+-----------+-------------+--------------------------------+
| mysql-bin.000002 | 287 | Table_map   |         1 |         336 | table_id: 110 (test.tb1)
|
| mysql-bin.000002 | 336 | Delete_rows |         1 |         385 | table_id: 110 flags:
STMT_END_F |
| mysql-bin.000002 | 385 | Xid         |         1 |         416 | COMMIT /* xid=2068
*/         |
```

SHOW BINARY LOGS 等价于 SHOW MASTER LOGS

PURGE BINARY LOGS 用于删除二进制日志，如：

PURGE BINARY LOGS TO 'mysql-bin.00010'; //把这个文件之前的其他文件都删除掉

PURGE BINARY LOGS BEFORE '2016-08-28 22:46:26';//把指定时间之前的二进制文件删除了

RESET MASTER 与 RESET SLAVE

前者清空 index 文件中列出的所有二进制日志，重置 index 文件为空，并创建一个新的二进制日志文件，一般用于 MASTER 首次启动时。后者使 SLAVE 忘记其在 MASTER 二进制日志文件中的复制位置，它会删除 master.info、relay-log.info 和所有中继日志文件并开始一个新的中继日志文件，以便于开始一个干净的复制。在使用 RESET SLAVE 前需先关闭 SLAVE 复制线程。

上述方式可以查看到服务器上存在的二进制日志文件及文件中的事件，但是想查看到文件中具体的内容并应于恢复场景还得借助 mysqlbinlog 这个工具。

语法格式： mysqlbinlog [options] log_file ...

输出内容会因日志文件的格式以及 mysqlbinlog 工具使用的选项不同而略不同。

mysqlbinlog 的可用选项可参考 man 手册。

二进制日志文件的格式包含行模式、语句模式和混合模式（也即有服务器决定在什么情况下记录什么类型的日志），基于语句的日志中事件信息包含执行的语句等，基于行的日志中事件信息包含的是行的变化信息等。混合模式的日志中两种类型的事件信息都会记录。

为了便于查看记录了行变化信息的事件在当时具体执行了什么样的 SQL 语句可以使用 mysqlbinlog 工具的-v（--verbose）选项，该选项会将行事件重构成被注释掉的伪 SQL 语句，如果想看到更详细的信息可以将该选项给两次如-vv，这样可以包含一些数据类型和元信息的注释内容，如

先切换到 binlog 所在的目录下

#mysqlbinlog mysql-bin.000001

#mysqlbinlog -v mysql-bin.000001

#mysqlbinlog -vv mysql-bin.000001

另外 mysqlbinlog 和可以通过--read-from-remote-server 选项从远程服务器读取二进制日志文件，这时需要一些而外的连接参数，如-h，-P，-p，-u 等，这些参数仅在指定了--read-from-remote-server 后有效。

无论是本地二进制日志文件还是远程服务器上的二进制日志文件，无论是行模式、语句模式还是混合模式的二进制日志文件，被 mysqlbinlog 工具解析后都可直接应用与 MySQL Server

进行基于时间点、位置或数据库的恢复。
下面我们就来演示如何使用 binlog 恢复之前删除数据（id=2 那条记录）
注意：在实际生产环境中，如果遇到需要恢复数据库的情况，不要让用户能访问到数据库，以避免新的数据插入进来，以及在主从的环境下，关闭主从。
查看 binlog 文件，从中找出 delete from test.tb1 where id=2

```
# cd /usr/local/mysql/data/
# mysqlbinlog    -v mysql-bin.000002
```

显示结果

```
# at 219
#160913 20:59:51 server id 1    end_log_pos 287 CRC32 0x1a97741b        Query     thread_id=42
      exec_time=0    error_code=0
SET TIMESTAMP=1473771591/*!*/;
SET @@session.pseudo_thread_id=42/*!*/;
SET                  @@session.foreign_key_checks=1,                  @@session.sql_auto_is_null=0,
@@session.unique_checks=1, @@session.autocommit=1/*!*/;
SET @@session.sql_mode=1075838976/*!*/;
SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/;
/*!\C utf8 *//*!*/;
SET
@@session.character_set_client=33,@@session.collation_connection=33,@@session.collation_
server=33/*!*/;
SET @@session.lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
BEGIN
/*!*/;
# at 287
#160913 20:59:51 server id 1    end_log_pos 336 CRC32 0x930ab248        Table_map:  `test`.`tb1`
mapped to number 125
# at 336
#160910 23:17:43 server id 1    end_log_pos 385 CRC32 0xdede3eb7    Delete_rows: table id 110
flags: STMT_END_F
BINLOG '
FyTUVxMBAAAAMQAAAFABAAAAAG4AAAAAAAEABHRlc3QAA3RiMQACAw8CPAAC2t7UdQ==
FyTUVyABAAAAMQAAAIEBAAAAAG4AAAAAAAEAAgAC//wCAAAACHpoYW5nc2Futz7e3g==
'/*!*/;
### DELETE FROM `test`.`tb1`
### WHERE
###     @1=2
###     @2='zhangsan'
# at 385
#160910 23:17:43 server id 1    end_log_pos 416 CRC32 0x7881c9da    Xid = 2068
COMMIT/*!*/;
```

从中可以看出 delete 事件发生 position 是 287，事件结束 position 是 416

恢复流程：直接用 bin-log 日志将数据库恢复到删除位置 287 前,然后跳过故障点,再进行恢复下面所有的操作,命令如下

由于之前没有做过全库备份，所以要使用所有 binlog 日志恢复，所以生产环境中需要很长时间恢复，导出相关 binlog 文件

#mysqlbinlog /usr/local/mysql/data/mysql-bin.000001 > /opt/mysql-bin.000001.sql

#mysqlbinlog --stop-position=287 /usr/local/mysql/data/mysql-bin.000002 > /opt/287.sql

#mysqlbinlog --start-position=416 /usr/local/mysql/data/mysql-bin.000002 > /opt/416.sql

删除 test 数据库

mysql>drop database test;

利用 binlog 恢复数据

#mysql -uroot -p123456< /opt/mysql-bin.000001.sql

#mysql -uroot -p123456< /opt/287.sql

#mysql -uroot -p123456< /opt/416.sql

恢复完成后，我们检查下表的数据是否完整

mysql> select * from test.tb1;

```
+----+----------+
| id | name     |
+----+----------+
|  1 | lisi     |
|  2 | zhangsan |
|  3 | tom      |
+----+----------+
```

Ok 完整的都恢复过来了

mysqlbinlog 选项示例

常见的选项有以下几个：

--start-datetime

从二进制日志中读取指定时间戳或者本地计算机时间之后的日志事件。

--stop-datetime

从二进制日志中读取指定时间戳或者本地计算机时间之前的日志事件。

--start-position

从二进制日志中读取指定 position 事件位置作为开始。

--stop-position

从二进制日志中读取指定 position 事件位置作为事件截至。

2、mysqldump 介绍

mysqldump 是 mysql 用于备份和数据转移的一个工具。它主要产生一系列的 SQL 语句，可以封装到文件，该文件包含有所有重建你的数据库所需要的 SQL 命令如 CREATE DATABASE，CREATE TABLE，INSERT 等等。可以用来实现轻量级的快速迁移或恢复数据库。

mysqldump 是将数据表导成 SQL 脚本文件，在不同的 MySQL 版本之间升级时相对比较合适，这也是最常用的备份方法。

mysqldump 一般在数据量很小的时候（几个 G）可以用于备份。当数据量比较大的情况下，就不建议用 mysqldump 工具进行备份了。

数据库的导出

导出对象说明：

mysqldump 可以针对单个表、多个表、单个数据库、多个数据库、所有数据库进行导出的操

作

mysqldump [options] db_name [tbl_name ...] //导出指定数据库或单个表

mysqldump [options] --databases db_name ... //导出多个数据库

#mysqldump [options] --all-databases //导出所有

导出数据库 test

mysqldump -uroot -p --flush-logs test > /opt/test.sql //--flush-logs 这个选项就会完整备份的时候重新开启一个新 binlog

数据库的导入

mysql -uroot -p test < /opt/test.sql

在前面我们介绍了 mysql 的 binlog 和 mysqldump 工具，下面我们来学习如何实现 mysqldump 全库备份+binlog 的数据恢复

环境准备与备份还原：

检查开启 binlog

先创建一些原始数据

```
mysql> reset master;
mysql> create database test_db;
mysql> use test_db;
mysql> create table tb1(id int primary key auto_increment,name varchar(20));
mysql> insert into tb1(name) values('tom1');
mysql> insert into tb1(name) values('tom2');
mysql> commit;
mysql> select * from tb1;
+----+------+
| id | name |
+----+------+
|  1 | tom1 |
|  2 | tom2 |
+----+------+
```

方案：mysqldump 全库备份+binlog 还原

1、**mysqldump 备份方案：**

每周一凌晨 1 点全库备份

2、备份步骤

（1）**创建备份目录**

mkdir /opt/mysqlbackup

mkdir /opt/mysqlbackup/daily

（2）全库备份

这里我们模拟周一的完整备份数据库任务

#mysqldump -uroot -p --flush-logs test_db > /opt/mysqlbackup/test_db_2016_09_12.sql（test_db_`date +%Y%m%d_%H%M%S`）

[root@localhost data]# ls -l /opt/mysqlbackup/

-rw-r--r--. 1 root root 1871 Sep 13 21:06 test_db_2016_09_12.sql

备份 mysqldump 全库备份之前的 binlog 日志文（注：生产环境中可能不只一个 binlog 文件）

cp /usr/local/mysql/data/mysql-bin.000001 /opt/mysqlbackup/daily/

mysql -uroot -p -e "purge binary logs to 'mysql-bin.000002'"

模拟下操作失误,将数据修改错误了。

```
mysql> use test_db;
mysql> delete from tb1 where id=1;
mysql> commit;
mysql> insert into tb1(name) values('tom3');
mysql> commit;
```

备份自 mysqldump 之后的 binlog 日志文件

```
cp /usr/local/mysql/data/mysql-bin.000002 /opt/mysqlbackup/daily/
```

上面的模拟的误操作是删除了 id=1 的记录

（3）现在我们使用 mysqldump 的全库备份和 binlog 来恢复数据。

使用 mysqldump 的备份进行全库恢复

```
# mysql -uroot -p test_db < /opt/mysqlbackup/test_db_2016_09_12.sql
```

查询一下数据

```
[root@localhost ~]# mysql -uroot -p -e "select * from test_db.tb1"
Enter password:
+----+------+
| id | name |
+----+------+
|  1 | tom1 |
|  2 | tom2 |
+----+------+
```

从显示结果可以看到使用 mysqldump 备份将数据还原到了备份时的状态，刚才删除的数据（id=2）恢复回来了，但备份后产生的数据却丢失了所以还得利用 binlog 进一步还原

因为删除是在全库备份后发生的，而 mysqldump 全库备份时使用--flush-logs 选项，所以只需要分析全库备份后的 binlog 即 mysql-bin.000002。

```
mysql> show binary logs;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mysql-bin.000002 |      1853 |
+------------------+-----------+
```

查看 mysql-bin.000002 中的事件，可以看到有删除事件

```
mysql> show binlog events in 'mysql-bin.000002';
| mysql-bin.000002 |  219 | Query       |         1 |         294 | BEGIN
|
| mysql-bin.000002 |  294 | Table_map   |         1 |         346 | table_id: 118
(test_db.tb1)
|
| mysql-bin.000002 |  346 | Delete_rows |         1 |         391 | table_id: 118
flags:                                                              STMT_END_F
|
| mysql-bin.000002 |  391 | Xid         |         1 |         422 | COMMIT /*
xid=2739 */
```

使用 mysqlbinlog 命令可以查看备份的 binlog 文件的详细事件。

恢复流程：我们直接用 bin-log 日志将数据库恢复到删除位置前,然后跳过故障点,再进行恢复删除后的所有操作。

mysqlbinlog -v /opt/mysqlbackup/daily/mysql-bin.000002

我们先用 mysqlbinlog 命令找到 delete 那条语句的位置

at 219

#160911 17:19:55 server id 1 end_log_pos 294 CRC32 0x84590493 Query thread_id=66 exec_time=0 error_code=0

SET TIMESTAMP=1473585595/*!*/;

SET @@session.pseudo_thread_id=66/*!*/;

SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=0, @@session.unique_checks=1, @@session.autocommit=1/*!*/;

SET @@session.sql_mode=1075838976/*!*/;

SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/;

/*!\C utf8 *//*!*/;

SET @@session.character_set_client=33,@@session.collation_connection=33,@@session.collation_server=33/*!*/;

SET @@session.lc_time_names=0/*!*/;

SET @@session.collation_database=DEFAULT/*!*/;

BEGIN

/*!*/;

at 294

#160911 17:19:55 server id 1 end_log_pos 346 CRC32 0x5cdccf9e Table_map: `test_db`.`tb1` mapped to number 118

at 346

#160911 17:19:55 server id 1 end_log_pos 391 CRC32 0x320c4935 Delete_rows: table id 118 flags: STMT_END_F

BINLOG '

uyHVVxMBAAAANAAAAFoBAAAAAHYAAAAAAAEAB3Rlc3RfZGIAA3RiMQACAw8CPAACns/cXA==

uyHVVyABAAAALQAAAIcBAAAAAHYAAAAAAAEAAgAC//wBAAAABHRvbTE1SQwy

'/*!*/;

DELETE FROM `test_db`.`tb1`

WHERE

@1=1

@2='tom1'

at 391

#160911 17:19:55 server id 1 end_log_pos 422 CRC32 0x5e4a6699 Xid = 2739

COMMIT/*!*/;

通过 mysqlbinlog 命令所显示的结果可以看到误操作 delete 的开始 postion 为 219，结束 position 是 422。

从二进制日志中读取指定 position=219 事件位置作为截至，即把数据恢复到 delete 删除前

mysqlbinlog --stop-position=219 /opt/mysqlbackup/daily/mysql-bin.000002 | mysql -u root -p

从二进制日志中读取指定 position=422 事件位置作为开始，即跳过删除事件，恢复删除事件

之后对数据的正常操作

#mysqlbinlog --start-position=422 /opt/mysqlbackup/daily/mysql-bin.000002 | mysql -u root -p

查看恢复结果：

```
# mysql -uroot -p -e "select * from test_db.tb1"
Enter password:
+----+------+
| id | name |
+----+------+
|  1 | tom1 |
|  2 | tom2 |
|  3 | tom3 |
+----+------+
```

从上面显示可以看出数据恢复到正常状态

生产环境中 Mysql 数据库的备份是周期性重复的操作，所以通常是要编写脚本实现，通过 crond 计划任务周期性执行备份脚本

mysqldump 备份方案：

周日凌晨 1 点全库备份

周一到周六凌晨每隔 4 个小时增量备份一次

设置 crontab 任务，每天执行备份脚本

```
# crontab –e
#每个星期日凌晨 1:00 执行完全备份脚本
0 1 * * 0 /root/mysqlfullbackup.sh >/dev/null 2>&1
#周一到周六每隔 4 个小时增量备份一次
0 */4 * * 1-6 /root/mysqldailybackup.sh >/dev/null 2>&1
```

mysqlfullbackup.sh 脚本内容：

```
[root@localhost ~]# cat mysqlfullbackup.sh
#!/bin/sh
# Name:mysqlFullBackup.sh
# 定义数据库目录
mysqlDir=/usr/local/mysql
# 定义用于备份数据库的用户名和密码
user=root
userpwd=123456
dbname=test_db
# 定义备份目录
databackupdir=/opt/mysqlbackup
[ ! -d $databackupdir ] && mkdir $databackupdir
# 定义邮件正文文件
emailfile=$databackupdir/email.txt
# 定义邮件地址
email=root@localhost.localdomain
# 定义备份日志文件
logfile=$databackupdir/mysqlbackup.log
DATE=`date -I`
```

```
echo "" > $emailfile
echo $(date +"%y-%m-%d %H:%M:%S") >> $emailfile
cd $databackupdir
# 定义备份文件名
dumpfile=mysql_$DATE.sql
gzdumpfile=mysql_$DATE.sql.tar.gz

# 使用 mysqldump 备份数据库，请根据具体情况设置参数
$mysqlDir/bin/mysqldump -u$user -p$userpwd --flush-logs -x $dbname > $dumpfile
//-x--lock-all-tables

# 压缩备份文件
if [ $? -eq 0 ]; then
   tar czf $gzdumpfile $dumpfile >> $emailfile 2>&1
   echo "BackupFileName:$gzdumpfile" >> $emailfile
   echo "DataBase Backup Success!" >> $emailfile
   rm -f $dumpfile
else
   echo "DataBase Backup Fail!" >> $emailfile
fi
# 写日志文件
echo "-------------------------------------------------------" >> $logfile
cat $emailfile >> $logfile
# 发送邮件通知
cat $emailfile | mail -s "MySQL Backup" $email
```

mysqldailybackup.sh 脚本内容：

```
[root@localhost ~]# cat mysqldailybackup.sh
#!/bin/sh
# Name:mysqlDailyBackup.sh
# 定义数据库目录和数据目录
mysqldir=/usr/local/mysql
datadir=$mysqldir/data
# 定义用于备份数据库的用户名和密码
user=root
userpwd=123456
# 定义备份目录，每日备份文件备份到$dataBackupDir/daily
databackupdir=/opt/mysqlbackup
dailybackupdir=$databackupdir/daily
[ ! -d $dailybackupdir ] && mkdir -p $databackupdir/daily
# 定义邮件正文文件
emailfile=$databackupdir/email.txt
# 定义邮件地址
```

```
email=root@localhost.localdomain
# 定义日志文件
logfile=$databackupdir/mysqlbackup.log
echo "" > $emailfile
echo $(date +"%y-%m-%d %H:%M:%S") >> $emailfile
#
# 刷新日志，使数据库使用新的二进制日志文件
$mysqldir/bin/mysqladmin -u$user -p$userpwd flush-logs
cd $datadir
# 得到二进制日志列表
filelist=`cat mysql-bin.index`
icounter=0
for file in $filelist
do
   icounter=`expr $icounter + 1`        // = let icounter++
done
nextnum=0
ifile=0
for file in $filelist
do
   binlogname=`basename $file`
   nextnum=`expr $nextnum + 1`
# 跳过最后一个二进制日志（数据库当前使用的二进制日志文件）
   if [ $nextnum -eq $icounter ]; then
      echo "Skip lastest!" > /dev/null
   else
      dest=$dailybackupdir/$binlogname
# 跳过已经备份的二进制日志文件
      if [ -e $dest ]; then
         echo "Skip exist $binlogname!" > /dev/null
      else
# 备份日志文件到备份目录
         cp $binlogname $dailybackupdir
         if [ $? -eq 0 ]; then
            ifile=`expr $ifile + 1`
            echo "$binlogname backup success!" >> $emailfile
         fi
      fi
   fi
done
if [ $ifile -eq 0 ];then
   echo "No Binlog Backup!" >> $emailfile
else
   echo "Backup $ifile File(s)." >> $emailfile
```

```
    echo "Backup MySQL Binlog OK!" >> $emailfile
fi
# 发送邮件通知
cat $emailfile | mail -s "MySQL Backup" $email
# 写日志文件
echo "------------------------------------------------------------" >> $logfile
cat $emailfile >> $logfile
```

二、使用 xtrabackup 进行 MySQL 数据库备份

前面介绍 mysqldump 备份方式是采用逻辑备份，其最大的缺陷就是备份和恢复速度都慢，对于一个小于 50G 的数据库而言，这个速度还是能接受的，但如果数据库非常大，那再使用 mysqldump 备份就不太适合了。

这时就需要一种好用又高效的工具，xtrabackup 就是其中一款，号称免费版的 InnoDB HotBackup。

Xtrabackup 实现是物理备份，而且是物理热备

目前主流的有两个工具可以实现物理热备：ibbackup 和 xtrabackup；ibbackup 是商业软件，需要授权，非常昂贵。而 xtrabackup 功能比 ibbackup 还要强大，但却是开源的。因此我们这里就来介绍 xtrabackup 的使用。

Xtrabackup 提供了两种命令行工具：

xtrabackup：专用于备份 InnoDB 和 XtraDB 引擎的数据；

innobackupex：这是一个 perl 脚本，在执行过程中会调用 xtrabackup 命令，这样用该命令即可以实现备份 InnoDB，也可以备份 MyISAM 引擎的对象。

Xtrabackup 是由 percona 提供的 mysql 数据库备份工具，特点：

(1)备份过程快速、可靠；

(2)备份过程不会打断正在执行的事务；

(3)能够基于压缩等功能节约磁盘空间和流量；

(4)自动实现备份检验；

(5)还原速度快。

官方链接地址：http://www.percona.com/software/percona-xtrabackup；可以下载源码编译安装，也可以下载适合的 RPM 包或使用 yum 进行安装或者下载二进制源码包。

安装 xtrabackup

1)下载 xtrabackup

wget https://www.percona.com/downloads/XtraBackup/Percona-XtraBackup-2.4.4/binary/tarball/percona-xtrabackup-2.4.4-Linux-x86_64.tar.gz

2)解压

```
# tar zxf percona-xtrabackup-2.4.4-Linux-x86_64.tar.gz
```

3)进入解压目录

```
# cd percona-xtrabackup-2.4.4-Linux-x86_64/
```

4)复制 bin 下的所有程序到/usr/bin

```
[root@localhost percona-xtrabackup-2.4.4-Linux-x86_64]# cp bin/* /usr/bin/
```

Xtrabackup 中主要包含两个工具：

xtrabackup：是用于热备份 innodb, xtradb 表中数据的工具，支持在线热备份，可以在不加锁的情况下备份 Innodb 数据表，不过此工具不能操作 Myisam 引擎表；

innobackupex：是将 xtrabackup 进行封装的 perl 脚本，能同时处理 Innodb 和 Myisam，但在处理 Myisam 时需要加一个读锁。

由于操作 Myisam 时需要加读锁，这会堵塞线上服务的写操作，而 Innodb 没有这样的限制，所以数据库中 Innodb 表类型所占的比例越大，则越有利。

4)安装相关插件

#yum install perl-DBI perl-DBD-MySQL perl-Time-HiRes perl-IO-Socket-SSL perl-TermReadKey.x86_64 perl-Digest-MD5 –y

5)下载 percona-toolkit 并安装

#wget https://www.percona.com/downloads/percona-toolkit/2.2.19/RPM/percona-toolkit-2.2.19-1.noarch.rpm

rpm -vih percona-toolkit-2.2.19-1.noarch.rpm

下面就可以启动备份了

方案一：xtrabackup 完全备份+binlog 增量备份

1、备份

创建备份目录

mkdir -p /opt/mysqlbackup/{full,inc}

full：全备存放的目录；inc：增量备份存放的目录

1)完全备份

基本语法：# innobackupex --user=DBUSER --password=DBUSERPASS /path/to/BACKUP-DIR/

执行下面的命令进行完全备份：

innobackupex --user=root --password=123456 /opt/mysqlbackup/full

注： --defaults-file=/etc/my.cnf 指定 mysql 的配置文件 my.cfg，如果指定则必须是第一个参数。

/path/to/BACKUP-DIR/指定备份所存放的目标目录，备份过程会创建一个以当时备份时间命名的目录存放备份文件。

出现如下提示。表示成功

```
160912 11:29:57 Backup created in directory '/opt/mysqlbackup/full/2016-09-12_11-29-55'
MySQL binlog position: filename 'mysql-bin.000023', position '2378'
160912 11:29:57 [00] Writing backup-my.cnf
160912 11:29:57 [00]        ...done
160912 11:29:57 [00] Writing xtrabackup_info
160912 11:29:57 [00]        ...done
xtrabackup: Transaction log of lsn (2536475) to (2536484) was copied.
160912 11:29:57 completed OK!
```

备份后的文件：

在备份的同时，备份数据会在备份目录下创建一个以当前日期时间为名字的目录存放备份文件：

```
[root@localhost ~]# ls /opt/mysqlbackup/full/
2016-09-12_11-29-55
[root@localhost ~]# ls /opt/mysqlbackup/full/2016-09-12_11-29-55/
backup-my.cnf   ibdata1             sys                    xtrabackup_checkpoints
db1             mysql               test_db                xtrabackup_info
ib_buffer_pool  performance_schema  xtrabackup_binlog_info  xtrabackup_logfile
[root@localhost ~]#
```

各文件说明：

(1)xtrabackup_checkpoints ——备份类型（如完全或增量）、备份状态（如是否已经为 prepared 状态）和 LSN(日志序列号)范围信息；

每个 InnoDB 页(通常为 16k 大小)都会包含一个日志序列号，即 LSN。LSN 是整个数据库

系统的系统版本号，每个页面相关的 LSN 能够表明此页面最近是如何发生改变的。
(2)xtrabackup_binlog_info ——— mysql 服务器当前正在使用的二进制日志文件及至备份这一刻为止二进制日志事件的位置。
(3)xtrabackup_binlog_pos_innodb ——二进制日志文件及用于 InnoDB 或 XtraDB 表的二进制日志文件的当前 position。
(4)xtrabackup_binary ——备份中用到的 xtrabackup 的可执行文件；
(5)backup-my.cnf ——备份命令用到的配置选项信息；
在使用 innobackupex 进行备份时，还可以使用--no-timestamp 选项来阻止命令自动创建一个以时间命名的目录；如此一来，innobackupex 命令将会创建一个 BACKUP-DIR 目录来存储备份数据
注意：相关选项说明：
其中，--user 指定连接数据库的用户名，--password 指定连接数据库的密码，--defaults-file 指定数据库的配置文件，innobackupex 要从其中获取 datadir 等信息；--database 指定要备份的数据库，这里指定的数据库只对 MyISAM 表有效，对于 InnoDB 数据来说都是全备（所有数据库中的 InnoDB 数据都进行了备份，不是只备份指定的数据库，恢复时也一样）；/opt/mysqlbackup/full 是备份文件的存放位置。
注意：备份数据库的用户需要具有相应权限，如果要使用一个最小权限的用户进行备份，则可基于如下命令创建此类用户：

```
mysql> create user 'bkpuser'@'localhost' identified by '123456';
mysql> revoke all privileges,grant option from 'bkpuser'@'localhost';
mysql> grant reload,lock tables,replication client, process on *.* to 'bkpuser'@'localhost';
mysql> flush privileges;
```

至此全备完全成功，然后向 mysql 某个库插入几条数据，然后进行增量备份
对完全备份的后数据库更改进行二进制日志增量备份：
查看完全备份时 binlog 日志位置(position)：

```
[root@localhost ~]# cat /opt/mysqlbackup/full/2016-09-12_11-29-55/xtrabackup_binlog_inf
o
mysql-bin.000023        2378
```

模拟数据库修改：

```
mysql> select * from tb1;
+----+------+
| id | name |
+----+------+
|  1 | tom1 |
|  2 | tom2 |
+----+------+
2 rows in set (0.00 sec)

mysql> insert into tb1(name) values('tom3');
Query OK, 1 row affected (0.00 sec)

mysql> insert into tb1(name) values('tom4');
Query OK, 1 row affected (0.00 sec)

mysql> quit
```

2)增量备份二进制文件：
#mysqlbinlog --start-position=2378 /usr/local/mysql/data/mysql-bin.000023 > /opt/mysqlbackup/inc/`date +%F`.sql

2、还原数据库：

模拟数据库损坏：

我这里直接使用删除数据目录文件来模拟损坏。

rm -fr /usr/local/mysql/data/*

还原完全备份：

（1）准备(prepare)一个完全备份

一般情况下，在备份完成后，数据尚且不能用于恢复操作，因为备份的数据中可能会包含尚未提交的事务或已经提交但尚未同步至数据文件中的事务。因此，此时数据文件仍处理不一致状态。“准备”的主要作用正是通过回滚未提交的事务及同步已经提交的事务至数据文件也使得数据文件处于一致性状态。

在准备（prepare）过程结束后，InnoDB 表数据已经前滚到整个备份结束的点，而不是回滚到 xtrabackup 刚开始时的点。

innobakupex 命令的--apply-log 选项可用于实现上述功能。如下面的命令：

--apply-log 指明是将日志应用到数据文件上，完成之后将备份文件中的数据恢复到数据库中：

innobackupex --apply-log /opt/mysqlbackup/full/2016-09-12_11-29-55/

注：/opt/mysqlbackup/full/2016-09-12_11-29-55/备份文件所在目录名称

如果执行正确，其最后输出的几行信息通常如下：

```
InnoDB: xtrabackup: Last MySQL binlog file position 1331, file name mysql-bin.000023
InnoDB: Removed temporary tablespace data file: "ibtmp1"
InnoDB: Creating shared tablespace for temporary tables
InnoDB: Setting file './ibtmp1' size to 12 MB. Physically writing the file full; Please
 wait ...
InnoDB: File './ibtmp1' size is now 12 MB.
InnoDB: 96 redo rollback segment(s) found. 1 redo rollback segment(s) are active.
InnoDB: 32 non-redo rollback segment(s) are active.
InnoDB: 5.7.13 started; log sequence number 2536981
xtrabackup: starting shutdown with innodb_fast_shutdown = 1
InnoDB: FTS optimize thread exiting.
InnoDB: Starting shutdown...
InnoDB: Shutdown completed; log sequence number 2537000
160912 11:53:25 completed OK!
```

在实现“准备”的过程中，innobackupex 通常还可以使用--use-memory 选项来指定其可以使用的内存的大小，默认通常为 100M。如果有足够的内存可用，可以多划分一些内存给 prepare 的过程，以提高其完成速度。

innobackupex 命令的--copy-back 选项用于执行恢复操作，其通过复制所有数据相关的文件至 mysql 服务器 DATADIR 目录中来执行恢复过程。innobackupex 通过 backup-my.cnf 来获取 DATADIR 目录的相关信息。

（2）还原数据库语法：

innobackupex --copy-back /opt/mysqlbackup/full/2016-09-12_11-29-55/

这里的--copy-back 指明是进行数据恢复。数据恢复完成之后，需要修改相关文件的权限 mysql 数据库才能正常启动。

如果执行正确，其输出信息的最后几行通常如下：

```
160912 12:22:52 [01] Copying ./performance_schema/session_status.frm to /usr/local/mysq
l/data/performance_schema/session_status.frm
160912 12:22:52 [01]        ...done
160912 12:22:52 [01] Copying ./ib_buffer_pool to /usr/local/mysql/data/ib_buffer_pool
160912 12:22:52 [01]        ...done
160912 12:22:52 [01] Copying ./xtrabackup_info to /usr/local/mysql/data/xtrabackup_info
160912 12:22:52 [01]        ...done
160912 12:22:52 [01] Copying ./xtrabackup_binlog_pos_innodb to /usr/local/mysql/data/xt
rabackup_binlog_pos_innodb
160912 12:22:52 [01]        ...done
160912 12:22:52 [01] Copying ./ibtmp1 to /usr/local/mysql/data/ibtmp1
160912 12:22:52 [01]        ...done
160912 12:22:52 completed OK!
```

请确保如上信息的最行一行出现“completed OK!”。

修改还原后的数据目录权限:

```
[root@localhost ~]# ll /usr/local/mysql/data/
total 417832
drwxr-x---. 2 root root         49 Sep 12 12:22 db1
-rw-r-----. 1 root root        304 Sep 12 12:22 ib_buffer_pool
-rw-r-----. 1 root root   12582912 Sep 12 12:22 ibdata1
-rw-r-----. 1 root root  134217728 Sep 12 12:22 ib_logfile0
-rw-r-----. 1 root root  134217728 Sep 12 12:22 ib_logfile1
-rw-r-----. 1 root root  134217728 Sep 12 12:22 ib_logfile2
-rw-r-----. 1 root root   12582912 Sep 12 12:22 ibtmp1
drwxr-x---. 2 root root       4096 Sep 12 12:22 mysql
drwxr-x---. 2 root root       8192 Sep 12 12:22 performance_schema
drwxr-x---. 2 root root       8192 Sep 12 12:22 sys
drwxr-x---. 2 root root         47 Sep 12 12:22 test_db
-rw-r-----. 1 root root         22 Sep 12 12:22 xtrabackup_binlog_pos_innodb
-rw-r-----. 1 root root        479 Sep 12 12:22 xtrabackup_info
```

当数据恢复至 DATADIR 目录以后，还需要确保所有数据文件的属主和属组均为正确的用户，如 mysql，否则，在启动 mysqld 之前还需要事先修改数据文件的属主和属组。如：

chown -R mysql:mysql /usr/local/mysql/data/

必须重启 MySQL：

systemctl restart mysqld

验证还原后的数据：

```
mysql> select * from tb1;
+----+------+
| id | name |
+----+------+
|  1 | tom1 |
|  2 | tom2 |
+----+------+
```

（3）还原增量备份：

为了防止还原时产生大量的二进制日志，在还原时可临时关闭二进制日志后再还原：

mysql> set sql_log_bin=0;

mysql>source /opt/mysqlbackup/inc/2016-09-12.sql

或者在命令行：mysql –uroot –p < /opt/mysqlbackup/inc/2016-09-12.sql

mysqlbinlog /opt/mysqlbackup/inc/2016-09-12.sql | mysql –uroot -p

重新启动二进制日志并验证还原数据：

mysql> set sql_log_bin=1;

验证数据是否恢复回来

```
mysql> select * from test_db.tb1;
+----+------+
| id | name |
+----+------+
|  1 | tom1 |
|  2 | tom2 |
|  3 | tom3 |
|  4 | tom4 |
+----+------+
```

方案二、xtrabackup 完全备份+xtrabacup 增量备份

前面我们进行增量备份时，使用的还是老方法：备份二进制日志。其实 xtrabackup 还支持进行增量备份。

先介绍下 xtrabackup 的备份原理

在 InnoDB 内部会维护一个 redo 日志文件，我们也可以叫做事务日志文件（transaction log，事务日志）。事务日志会存储每一个 InnoDB 表数据的记录修改。当 InnoDB 启动时，InnoDB 会检查数据文件和事务日志，并执行两个步骤：它应用已经提交的事务日志到数据文件，并将修改过但没有提交的数据进行回滚操作。

xtrabackup 在启动时会记住 log sequence number（LSN），并且复制所有的数据文件。复制过程需要一些时间，所以这期间如果数据文件有改动，那么将会使数据库处于一个不

同的时间点。这时，xtrabackup 会运行一个后台进程，用于监视事务日志，并从事务日志复制最新的修改。xtrabackup 必须持续的做这个操作，是因为事务日志是会轮转重复的写入，并且事务日志可以被重用。所以 xtrabackup 自启动开始，就不停的将事务日志中每个数据文件的修改都记录下来。这就是 xtrabackup 的备份过程

所以每个 InnoDB 的页面都会包含一个 LSN 信息，每当相关的数据发生改变，相关的页面的 LSN 就会自动增长。这正是 InnoDB 表可以进行增量备份的基础。
xtraBackup 基于 InnoDB 的 crash-recovery 功能。它会复制 innodb 的 data file，由于不锁表，复制出来的数据是不一致的，在恢复的时候使用 crash-recovery，使得数据恢复一致。
当 InnoDB 启动的时候，它会先去检查 data file 和 transaction log，并且会做二步操作：
1.It applies committed transaction logentries to the data files
2.it performs an undo operation on anytransactions that modified data but did not commit.
所以在 prepare 过程中，XtraBackup 使用复制到的 transactions log 对备份出来的 innodb data file 进行 crash recovery。
测试环境准备
创建一个测试数据库，并创建一张表输入几行数据
mysql> create database test;
mysql> use test;
mysql> create table xx(id int,name varchar(20));
mysql> insert into xx values(1,'tom1');
mysql> insert into xx values(2,'tom2');

1、 xtrabacup 进行备份
执行完全备份：
备份命令：
xtrabackup --defaults-file=/etc/my.cnf --user=root --password="123456" --port=3306 --backup --target-dir=/opt/mysqlbackup/full/full_incre_$(date +%Y%m%d_%H%M%S)
部分显示信息如下图所示：

```
160912 22:11:13 Backup created in directory '/opt/mysqlbackup/full/full_incre_20160912_
221111'
MySQL binlog position: filename 'mysql-bin.000065', position '1529'
160912 22:11:13 [00] Writing backup-my.cnf
160912 22:11:13 [00]        ...done
160912 22:11:13 [00] Writing xtrabackup_info
160912 22:11:13 [00]        ...done
xtrabackup: Transaction log of lsn (2596412) to (2596421) was copied.
160912 22:11:13 completed OK!
```

其中，--defaults-file 指定数据库的配置文件，如果使用该参数必须做为第一个参数；--user 指定连接数据库的用户名；--password 指定连接数据库的密码；--port 指定连接数据库的端口号；--backup 实施备份到 target-dir; --target-dir=name 备份文件的存放目录路径。innobackupex 要从其中获取 datadir 等信息；--database 指定要备份的数据库，这里指定的数据库只对 MyISAM 表和 InnoDB 表的表结构有效，对于 InnoDB 数据来说都是全备（所有数据库中的 InnoDB 数据都进行了备份，不是只备份指定的数据库，恢复时也一样）；/opt/mysqlbackup/full/是备份文件的存放位置。
查看完全备份文件
[root@localhost ~]# ls /opt/mysqlbackup/full/ -l
drwxr-x---. 8 root root 4096 Sep 12 22:11 full_incre_20160912_221111

xtrabackup 进行增量备份

先录入些数据，实现第一次增量数据：

mysql> use test;

mysql> insert into xx values(3,'tom3');

再进行增量备份 1

备份命令：

xtrabackup --defaults-file=/etc/my.cnf --user=root --password="123456" --port=3306 --backup --target-dir=/opt/mysqlbackup/inc/incre_$(date +%Y%m%d_%H%M%S) --incremental-basedir=/opt/mysqlbackup/full/full_incre_20160912_221111/

部分显示信息如下图所示：

```
160912 22:15:13 Backup created in directory '/opt/mysqlbackup/inc/incre_20160912_221510
'
MySQL binlog position: filename 'mysql-bin.000065', position '1790'
160912 22:15:13 [00] Writing backup-my.cnf
160912 22:15:13 [00]        ...done
160912 22:15:13 [00] Writing xtrabackup_info
160912 22:15:13 [00]        ...done
xtrabackup: Transaction log of lsn (2596748) to (2596757) was copied.
160912 22:15:13 completed OK!
```

其中，--incremental-basedir 指定上次完整备份或者增量备份文件的位置(即如果是第一次增量备份则指向完全备份所在目录，在执行过增量备份之后再一次进行增量备份时，其 --incremental-basedir 应该指向上一次的增量备份所在的目录)。

查看增量备份文件：

[root@localhost ~]# ls -l /opt/mysqlbackup/inc/

drwxr-x---. 8 root root 4096 Sep 12 22:15 incre_20160912_221510

注：

这里的增量备份其实只针对的是 InnoDB，对于 MyISAM 来说，还是完整备份。

向表中再插入几行数据，继续第二次增量备份

mysql> use test;

mysql> insert into xx values(4,'tom4');

mysql> commit;

进行第二次增量备份

备份命令：

xtrabackup --defaults-file=/etc/my.cnf --user=root --password="123456" --port=3306 --backup --target-dir=/opt/mysqlbackup/inc/incre_$(date +%Y%m%d_%H%M%S) --incremental-basedir=/opt/mysqlbackup/inc/incre_20160912_221510/

部分显示信息如下图所示：

```
160912 22:19:19 Backup created in directory '/opt/mysqlbackup/inc/incre_20160912_221916
'
MySQL binlog position: filename 'mysql-bin.000065', position '2051'
160912 22:19:19 [00] Writing backup-my.cnf
160912 22:19:19 [00]        ...done
160912 22:19:19 [00] Writing xtrabackup_info
160912 22:19:19 [00]        ...done
xtrabackup: Transaction log of lsn (2597100) to (2597109) was copied.
160912 22:19:19 completed OK!
```

注：第二次增量备份--incremental-basedir 指向上一次增量备份文件的位置。

查看增量备份文件

[root@localhost ~]# ls -l /opt/mysqlbackup/inc/

drwxr-x---. 8 root root 4096 Sep 12 22:15 incre_20160912_221510

drwxr-x---. 8 root root 4096 Sep 12 22:19 incre_20160912_221916

2、 xtrabacup 进行增量恢复

为了验证比对，先删除两个增量备份前表里面的数据

mysql> use test;

mysql> delete from xx where id=3;

完整备份恢复：

在进行恢复前，如果完整备份在远程主机上，首先将完整备份复制到本地主机上，如果是 tar 包，则需要先解包，解包命令为：tar －izxf xxx.tar，这里必须使用-i 参数（忽略存档中的 0 字节块（通常意味着文件结束））。

开始全备份恢复

命令如下：

xtrabackup --defaults-file=/etc/my.cnf --prepare --user=root --password="123456" --apply-log-only --target-dir=/opt/mysqlbackup/full/full_incre_20160912_221111/

部分显示信息如下图所示：

```
xtrabackup: starting shutdown with innodb_fast_shutdown = 1
InnoDB: Starting shutdown...
InnoDB: Shutdown completed; log sequence number 2596430
InnoDB: Number of pools: 1
160912 22:24:57 completed OK!
```

恢复到第一次增量的时刻

增量备份恢复的步骤和完整备份恢复的步骤基本一致，只是应用日志的过程稍有不同。增量备份恢复时，是先将所有的增量备份挨个应用到完整备份的数据文件中，然后再将完整备份中的数据恢复到数据库中。

恢复命令：

xtrabackup --defaults-file=/etc/my.cnf --prepare --user=root --password="123456" --apply-log-only --target-dir=/opt/mysqlbackup/full/full_incre_20160912_221111/--incremental-dir=/opt/mysqlbackup/inc/incre_20160912_221510/

部分显示信息如下图所示：

```
160912 22:27:23 [00] Copying /opt/mysqlbackup/inc/incre_20160912_221510//xtrabackup_binlog_info to ./xtrabackup_binlog_info
160912 22:27:23 [00]        ...done
160912 22:27:23 [00] Copying /opt/mysqlbackup/inc/incre_20160912_221510//xtrabackup_info to ./xtrabackup_info
160912 22:27:23 [00]        ...done
160912 22:27:23 completed OK!
```

恢复到第二次增量备份前面：

恢复命令：

xtrabackup --defaults-file=/etc/my.cnf --prepare --user=root --password="123456" --apply-log-only --target-dir=/opt/mysqlbackup/full/full_incre_20160912_221111/ --incremental-dir=/opt/mysqlbackup/inc/incre_20160912_221916/

部分显示信息如下图所示：

```
160912 22:29:16 [00] Copying /opt/mysqlbackup/inc/incre_20160912_221916//xtrabackup_bin
log_info to ./xtrabackup_binlog_info
160912 22:29:16 [00]        ...done
160912 22:29:16 [00] Copying /opt/mysqlbackup/inc/incre_20160912_221916//xtrabackup_inf
o to ./xtrabackup_info
160912 22:29:16 [00]        ...done
160912 22:29:16 completed OK!
```

恢复整个库

恢复命令：

xtrabackup --defaults-file=/etc/my.cnf --prepare --user=root --password="123456" --target-dir=/opt/mysqlbackup/full/full_incre_20160912_221111/

部分显示信息如下图所示：

```
InnoDB: 5.7.13 started; log sequence number 2597416
xtrabackup: starting shutdown with innodb_fast_shutdown = 1
InnoDB: FTS optimize thread exiting.
InnoDB: Starting shutdown...
InnoDB: Shutdown completed; log sequence number 2597477
160912 22:32:24 completed OK!
```

然后停止 mysql 数据库：

[root@localhost ~]# systemctl stop mysqld

开始 rsync 数据文件：

cd /opt/mysqlbackup/full/full_incre_20160912_221111/

#rsync -rvt --exclude 'xtrabackup_checkpoints' --exclude 'xtrabackup_logfile' ./ /usr/local/mysql/data/

当数据恢复至 DATADIR 目录以后，还需要确保所有数据文件的属主和属组均为正确的用户，如 mysql，否则，在启动 mysqld 之前还需要事先修改数据文件的属主和属组。

授予 mysql 访问权限：

chown -R mysql:mysql /usr/local/mysql/data/

启动 mysql 服务：

systemctl start mysqld

验证

登录 mysql，看到以前在备份之后删除的数据已经通过 2 次增量备份恢复过来了，如下所示：

[root@localhost ~]# mysql -uroot -p123456 -e "select * from test.xx"

```
+------+------+
| id   | name |
+------+------+
|    1 | tom1 |
|    2 | tom2 |
|    3 | tom3 |
|    4 | tom4 |
+------+------+
```

方案三：innobackupex 全库备份+innobackupex 增量备份

测试环境准备

创建一个测试数据库，并创建一张表输入几行数据

mysql> create database test2;

mysql> use test2;

mysql> create table yy(id int,name varchar(20));

mysql> insert into yy values(1,'kim1');

mysql> insert into yy values(2,'kim2');

1、innobackupex 先做完全备份

命令如下：

innobackupex --defaults-file=/etc/my.cnf --user=root --password="123456" /opt/mysqlbackup/full/full_incre_$(date +%Y%m%d_%H%M%S) --no-timestamp

部分显示信息如下图所示：

```
160912 23:52:39 Backup created in directory '/opt/mysqlbackup/full/full_incre
_20160912_235237'
MySQL binlog position: filename 'mysql-bin.000066', position '2681'
160912 23:52:39 [00] Writing backup-my.cnf
160912 23:52:39 [00]        ...done
160912 23:52:39 [00] Writing xtrabackup_info
160912 23:52:39 [00]        ...done
xtrabackup: Transaction log of lsn (2611370) to (2611379) was copied.
160912 23:52:39 completed OK!
```

查看完全备份文件

ll /opt/mysqlbackup/full/

drwxr-x---. 10 root root 4096 Sep 12 23:52 full_incre_20160912_235237

innobackupex 做增量备份

做第一次增量备份

先录入增量数据

mysql> use test2;

mysql> insert into yy values(3,'kim3');

再进行增量备份，命令如下：

innobackupex --incremental /opt/mysqlbackup/inc/incre_$(date +%Y%m%d_%H%M%S) --incremental-basedir=/opt/mysqlbackup/full/full_incre_20160912_235237/ --user=root --password="123456" --no-timestamp

部分显示信息如下图所示：

```
160912 23:56:38 Backup created in directory '/opt/mysqlbackup/inc/incre_20160
912_235636'
MySQL binlog position: filename 'mysql-bin.000066', position '2944'
160912 23:56:38 [00] Writing backup-my.cnf
160912 23:56:38 [00]        ...done
160912 23:56:38 [00] Writing xtrabackup_info
160912 23:56:38 [00]        ...done
xtrabackup: Transaction log of lsn (2611724) to (2611733) was copied.
160912 23:56:39 completed OK!
```

查看增量备份文件

ll /opt/mysqlbackup/inc/

drwxr-x---. 10 root root 4096 Sep 12 23:56 incre_20160912_235636

基于全备和第一个增量备份来做第二次增量备份

先录入增量数据录入

mysql> use test2;

mysql> insert into yy values(4,'kim4');

开始进行第二次增量备份，备份命令：

innobackupex --incremental /opt/mysqlbackup/inc/incre_$(date +%Y%m%d_%H%M%S) --incremental-basedir=/opt/mysqlbackup/inc/incre_20160912_235636/ --user=root --password="123456" --no-timestamp

部分显示信息如下图所示：

```
160912 23:59:44 Backup created in directory '/opt/mysqlbackup/inc/incre_20160
912_235942'
MySQL binlog position: filename 'mysql-bin.000066', position '3207'
160912 23:59:44 [00] Writing backup-my.cnf
160912 23:59:44 [00]        ...done
160912 23:59:44 [00] Writing xtrabackup_info
160912 23:59:44 [00]        ...done
xtrabackup: Transaction log of lsn (2612062) to (2612071) was copied.
160912 23:59:44 completed OK!
```

查看增量备份文件

ll /opt/mysqlbackup/inc/

drwxr-x---. 10 root root 4096 Sep 12 23:56 incre_20160912_235636

drwxr-x---. 10 root root 4096 Sep 12 23:59 incre_20160912_235942

2、innobackupex 做增量恢复

先删除两次增量数据，用来查看验证恢复结果

mysql> use test2;

mysql> delete from yy where id=3;

开始做恢复，恢复全备份

命令如下：

innobackupex --apply-log --redo-only /opt/mysqlbackup/full/full_incre_20160912_235237/

部分显示信息如下图所示：

```
InnoDB: xtrabackup: Last MySQL binlog file position 2681, file name mysql-bin
.000066

xtrabackup: starting shutdown with innodb_fast_shutdown = 1
InnoDB: Starting shutdown...
InnoDB: Shutdown completed; log sequence number 2611388
InnoDB: Number of pools: 1
160913 00:05:15 completed OK!
```

--redo-only　用于准备增量备份内容把数据合并到全备份目录，配合 incremental-dir 增量备份目录使用

基于全备份进行第一次增量备份的恢复

命令如下：

innobackupex --apply-log --redo-only /opt/mysqlbackup/full/full_incre_20160912_235237/ --incremental-dir=/opt/mysqlbackup/inc/incre_20160912_235636/

基于全备份和第一次增量备份，恢复第二次增量备份

命令如下：

innobackupex --apply-log --redo-only /opt/mysqlbackup/full/full_incre_20160912_235237/ --incremental-dir=/opt/mysqlbackup/inc/incre_20160912_235942/

恢复整个数据库

停止数据库

systemctl stop mysqld

清空数据目录下所有文件

mkdir -p /tmp/mysqldatabak

mv /usr/local/mysql/data/* /tmp/mysqldatabak/

将恢复好的数据按照配置文件的需求拷贝到相应目录

innobackupex --defaults-file=/etc/my.cnf --user=root --password="123456" --copy-back /opt/mysqlbackup/full/full_incre_20160912_235237/

当数据恢复至 DATADIR 目录以后，还需要确保所有数据文件的属主和属组均为正确的用户，如 mysql，否则，在启动 mysqld 之前还需要事先修改数据文件的属主和属组。

赋予 mysql 账号权限

chown -R mysql:mysql /usr/local/mysql/data/

启动 mysql 服务

systemctl start mysqld

登录 mysql 界面，查看数据是否已经恢复，如下所示：

```
mysql> use test2;
mysql> select * from yy;
+------+------+
| id     | name |
+------+------+
|       1 | kim1 |
|       2 | kim2 |
|       3 | kim3 |
|       4 | kim4 |
+------+------+
```

附：Xtrabackup 的“流”及“备份压缩”功能

Xtrabackup 对备份的数据文件支持“流”功能，即可以将备份的数据通过 STDOUT 传输给 tar 程序进行归档，而不是默认的直接保存至某备份目录中。要使用此功能，仅需要使用--stream 选项即可。如：

innobackupex --user=root --password="123456" --stream=tar /opt/mysqlbackup/full/ | gzip >/opt/mysqlbackup/full/full_`date +%F_%H%M%S`.tar.gz