MySQL数据备份（增量备份）

内容来自《MySQL 5.7 Reference Manual Including MySQL Cluster NDB 7.5》

**Full Versus Incremental Backups**

**完整备份对比增量备份**

A full backup includes all data managed by a MySQL server at a given point in time. An incremental backup consists of the changes made to the data during a given time span (from one point in time to another). MySQL has different ways to perform full backups, such as those described earlier in this section.

完整备份包括了给定时间点上MySQL服务器管理的所有数据。而增量备份包含的是给定时间跨度上的数据变化结果（从一个时间点到另一个时间点）。MySQL有几种不同的方式执行完全备份，比如此前本章描述的相关部分。

Incremental backups are made possible by enabling the server's binary log, which the server uses to record data changes.

使能增量备份功能，需要打开服务器的二进制日志功能，MySQL服务器用二进制日志来记录数据的更改变化情况。

**Full Versus Point-in-Time (Incremental) Recovery**

**完全恢复对比时间点（增量）恢复**

A full recovery restores all data from a full backup. This restores the server instance to the state that it had when the backup was made. If that state is not sufficiently current, a full recovery can be followed by recovery of incremental backups made since the full backup, to bring the server to a more up-to-date state.

完全恢复可以用于完整备份情况下的所有数据的恢复。可以将服务器实例恢复到创建备份时的状态。如果对于当前状态，完全恢复之后的状态仍然不足以满足需求，那么在做完完全恢复之后，可以采用完全备份之后增量备份文件进行增量恢复，以此可以是服务器达到最新的状态。

Incremental recovery is recovery of changes made during a given time span. This is also called point-intime recovery because it makes a server's state current up to a given time. Point-in-time recovery is based on the binary log and typically follows a full recovery from the backup files that restores the server to its state when the backup was made. Then the data changes written in the binary log files are applied as incremental recovery to redo data modifications and bring the server up to the desired point in time.

增量恢复主要是将给定时间跨度上的数据更改状态进行恢复。其也被称为时间点恢复，因为增量恢复可以是服务器状态变更到一个指定时间点的状态。时间点恢复操作基于二进制日志文件，而且需要在做完完全恢复之后，才能做增量恢复操作，完全备份之后的数据改变情况会写入到二进制日志文件中，此后，二进制日志文件会被用于增量恢复，以此来实现重做数据修改操作，并且可以将数据库服务器恢复到指定的时间点。

**Making Incremental Backups by Enabling the Binary Log**

**通过打开二进制日志功能来使能增量备份**

MySQL supports incremental backups: You must start the server with the --log-bin option to enable binary logging; see Section 6.4.4, “The Binary Log”. The binary log files provide you with the information you need to replicate changes to the database that are made subsequent to the point at which you performed a backup. At the moment you want to make an incremental backup (containing all changes that happened since the last full or incremental backup), you should rotate the binary log by using FLUSH LOGS. This done, you need to copy to the backup location all binary logs which range from the one of the moment of the last full or incremental backup to the last but one. These binary logs are the incremental backup; at restore time, you apply them as explained in Section 8.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”. The next time you do a full backup, you should also rotate the binary log using FLUSH LOGS or mysqldump --flush-logs. See Section 5.5.4, “mysqldump — A Database Backup

Program”.

MySQL支持增量备份：需要在启动服务的时候在命令行中明确指定**--log-bin**选项，告诉服务器打开二进制日志功能；具体参看Manual Section 6.4.4, “The Binary Log”部分。二进制日志文件给你提供了重现数据库更改的信息，因为执行数据库备份操作之后的数据库变更信息都会被记录在二进制日志文件中。当你想做增量备份的时候（包含最后一次完全备份或者增量备份操作之后发生的所有数据库更改操作），需要使用**FLUSH LOGS**命令执行二进制日志文件轮转操作。执行完这步操作之后，需要拷贝自从最后一次完全备份或者增量备份之后到倒数第二次增量备份之间的所有二进制文件到备份路径下。这些二进制日志文件即是增量备份；在恢复的时候，就需要应用这些二进制文件来实现，如Section 8.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”。下次你做完整备份的时候，也需要使用**FLUSH LOGS**命令或者**mysqldump --flush-logs**来实现手动日志轮转操作。具体参看Section 5.5.4, “mysqldump —— A Database Backup Program”。

**8.3.1 Establishing a Backup Policy**

**8.3.1 建立备份策略**

To be useful, backups must be scheduled regularly. A full backup (a snapshot of the data at a point in time) can be done in MySQL with several tools. For example, MySQL Enterprise Backup can perform a physical backup of an entire instance, with optimizations to minimize overhead and avoid disruption when backing up InnoDB data files; mysqldump provides online logical backup. This discussion uses mysqldump.

要想达到备份的作用，那么备份操作需要规律性的定期执行。MySQL中有多种工具可以实现一次完全备份操作（某个时间点的数据快照）。比如MySQL Enterprise Backup可以实现全部实例的物理备份，通过优化可以减少系统资源开销，并且可以避免备份InnoDB数据库文文件的时候数据损坏的情况。

Assume that we make a full backup of all our InnoDB tables in all databases using the following command on Sunday at 1 p.m., when load is low:

假定我们在周六的下午1点钟，系统负载低的时候，使用如下命令执行一次数据库中所有的InnoDB表的完全备份操作：

shell> **mysqldump --single-transaction --all-databases > backup\_sunday\_1\_PM.sql**

The resulting .sql file produced by mysqldump contains a set of SQL INSERT statements that can be used to reload the dumped tables at a later time.

mysqldump命令产生的.sql结果文件中，包含了一组INSERT语句，通过这些INSERT语句，可以在随后重新装载备份表中的数据。

This backup operation acquires a global read lock on all tables at the beginning of the dump (using FLUSH TABLES WITH READ LOCK). As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the FLUSH statement is issued, the backup operation may stall until those statements finish. After that, the dump becomes lock-free and does not disturb reads and writes on the tables.

备份操作在开始之前，会给所有表加上全局读取锁（使用FLUSH TABLES WITH READ LOCK）。一旦锁被加上，读取二进制日志文件，然后释放锁。如果当发起FLUSH语句的时候，有需要长时间运行的update语句执行，那么备份操作会暂停，等到update语句执行完成之后继续进行备份操作。此后，dump操作变成无锁状态，也就不会干扰到表的读写操作了。

It was assumed earlier that the tables to back up are InnoDB tables, so --single-transaction uses a consistent read and guarantees that data seen by mysqldump does not change. (Changes made by other clients to InnoDB tables are not seen by the mysqldump process.) If the backup operation includes nontransactional tables, consistency requires that they do not change during the backup. For example, for the MyISAM tables in the mysql database, there must be no administrative changes to MySQL accounts during the backup.

Full backups are necessary, but it is not always convenient to create them. They produce large backup files and take time to generate. They are not optimal in the sense that each successive full backup includes all data, even that part that has not changed since the previous full backup. It is more efficient to make an initial full backup, and then to make incremental backups. The incremental backups are smaller and take less time to produce. The tradeoff is that, at recovery time, you cannot restore your data just by reloading the full backup. You must also process the incremental backups to recover the incremental changes.

假设此前要备份的表都是InnoDB表，所以使用--single-transaction选项实现一致性读取，确保mysqldump执行备份操作的过程中表是没有被更改的。（其他客户端对InnoDB表的更改不会被mysqldump进程看到。）如果备份操作中包含了非事务型的表，那么在备份过程中，不会改变其一致性。比如MySQL数据库中的MyISAM表，要求在数据库备份操作过程中，不能有管理性的更改操作。完全备份是有必要的，但是创建完全备份并总是很方便的。创建完全备份会产生很大的备份文件，而且会花费时间来生成这个备份文件。而且每一个连续的完全备份中都包含所有的数据，这也不是最优的情况，因为即便自从上次备份之后没有更改的数据也会被完全备份操作备份。更高效的做法应该是，初始做一个完全备份，此后在完全备份的基础上做增量备份。增量备份产生的文件更小，而且耗费的时间也更短。所做的取舍就是，在恢复的时候，无法像恢复完全备份产生的备份文件那样重新装载数据来实现恢复操作。必须对增量备份文件进行必要的处理，来恢复增量的修改操作。

To make incremental backups, we need to save the incremental changes. In MySQL, these changes are represented in the binary log, so the MySQL server should always be started with the --log-bin option to enable that log. With binary logging enabled, the server writes each data change into a file while it updates data. Looking at the data directory of a MySQL server that was started with the --log-bin option and that has been running for some days, we find these MySQL binary log files:

为了创建增量备份，我们需要保存增量的修改操作。在MySQL中，这些更改操作都记录在二进制日志文件中，要启动二进制日志功能，可以在启动MySQL服务器的时候，加上--log-bin选项。通过打开二进制日志功能，服务器会将更新数据时每一个数据的变化都写入到这个二进制日志文件中。下面看下启动MySQL服务器时加了--log-bin选项之后，运行了几天的服务器，产生的二进制文件如下所示：

-rw-rw---- 1 guilhem guilhem 1277324 Nov 10 23:59 gbichot2-bin.000001

-rw-rw---- 1 guilhem guilhem 4 Nov 10 23:59 gbichot2-bin.000002

-rw-rw---- 1 guilhem guilhem 79 Nov 11 11:06 gbichot2-bin.000003

-rw-rw---- 1 guilhem guilhem 508 Nov 11 11:08 gbichot2-bin.000004

-rw-rw---- 1 guilhem guilhem 220047446 Nov 12 16:47 gbichot2-bin.000005

-rw-rw---- 1 guilhem guilhem 998412 Nov 14 10:08 gbichot2-bin.000006

-rw-rw---- 1 guilhem guilhem 361 Nov 14 10:07 gbichot2-bin.index

Each time it restarts, the MySQL server creates a new binary log file using the next number in the sequence. While the server is running, you can also tell it to close the current binary log file and begin a new one manually by issuing a FLUSH LOGS SQL statement or with a mysqladmin flush-logs command. mysqldump also has an option to flush the logs. The .index file in the data directory contains the list of all MySQL binary logs in the directory.

每次重启的时候，MySQL服务器都会创建一个新的二进制文件，并且使用数字按序递增的方式命名日志文件名。服务器运行期间，也可以通过提交FLUSH LOGS语句或者在命令行中执行mysqladmin flush-logs命令，来通知服务器关闭当前的二进制日志文件，并创建一个新的二进制文件。mysqldump也有一个选项可以刷新二进制日志文件。二进制日志文件所在目录中的.inde文件包含了当前目录中的所有二进制文件的列表。

The MySQL binary logs are important for recovery because they form the set of incremental backups. If you make sure to flush the logs when you make your full backup, the binary log files created afterward contain all the data changes made since the backup. Let's modify the previous mysqldump command a bit so that it flushes the MySQL binary logs at the moment of the full backup, and so that the dump file contains the name of the new current binary log:

MySQL的二进制日志文件对于数据库的恢复来说是很重要的，因为它们形成了一组增量备份集合。如果你确定在完全备份之后刷新了二进制日志文件，那么二进制日志文件就会在完全备份之后被创建，并且包含完全备份之后的所有数据更改操作。我们修改一下之前的mysqldump命令，使其在完全备份的时候能够刷新MySQL二进制日志文件，这样，产生的备份文件中就有当前新的二进制日志文件了：

shell> **mysqldump --single-transaction --flush-logs --master-data=2 \**

**--all-databases > backup\_sunday\_1\_PM.sql**

After executing this command, the data directory contains a new binary log file, gbichot2-bin.000007, because the --flush-logs option causes the server to flush its logs. The --master-data option causes mysqldump to write binary log information to its output, so the resulting .sql dump file includes these lines:

执行完这个命令之后，日志数据目录中就包含了一个新的二进制日志文件，文件名为：gbichot2-bin.000007，因为--flush-logs选项使服务器刷新其日志文件。--master-data选项使mysqldump将二进制日志信息写入到标准输出中，所以产生的备份文件.sql中包含了如下几行：

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

-- CHANGE MASTER TO MASTER\_LOG\_FILE='gbichot2-bin.000007',MASTER\_LOG\_POS=4;

Because the mysqldump command made a full backup, those lines mean two things:

由于mysqldump命令创建完全备份，所以这些行代表两个含义：

• The dump file contains all changes made before any changes written to the gbichot2-bin.000007 binary log file or newer.  
 备份文件中包含了写入gbichot2-bin.000007或者更新的二进制日志文件之前的所有数据变更操作。

• All data changes logged after the backup are not present in the dump file, but are present in the gbichot2-bin.000007 binary log file or newer.

备份操作之后的所有被记录的数据更改操作不会被记录到这个备份文件中，但是会记录在gbichot2-bin.000007或者更新的二进制日志文件中。

On Monday at 1 p.m., we can create an incremental backup by flushing the logs to begin a new binary log file. For example, executing a mysqladmin flush-logs command creates gbichot2-bin.000008.

在周一下午1点，可以通过刷新日志文件的方式创建一个增量备份二进制日志文件。比如，执行mysqladmin flush-logs命令创建一个名为gbichot2-bin.000008的二进制日志文件。

All changes between the Sunday 1 p.m. full backup and Monday 1 p.m. will be in the gbichot2-bin.000007 file. This incremental backup is important, so it is a good idea to copy it to a safe place. (For example, back it up on tape or DVD, or copy it to another machine.) On Tuesday at 1 p.m., execute another mysqladmin flush-logs command. All changes between Monday 1 p.m. and Tuesday 1 p.m. will be in the gbichot2-bin.000008 file (which also should be copied somewhere safe).

所有在周日下午1点完全备份之后到周一下午1点之间的数据修改操作都会被记录到二进制日志文件gbichot2-bin.000007中。这个增量备份很重要，所以最好将其拷贝到一个安全存储设备中。（比如，将其备份到磁带或者DVD光盘中，或者将其拷贝到其他服务器中。）在周二下午1点再次执行mysqladmin flush-logs命令。那么所有在周一下午1点到周二下午1点期间的数据更改操作都会记录到二进制日志文件gbichot2-bin.000008中（其也应该被拷贝到一个安全的存储设备中）。

The MySQL binary logs take up disk space. To free up space, purge them from time to time. One way to do this is by deleting the binary logs that are no longer needed, such as when we make a full backup:

MySQL二进制日志文件会占用一部分存储空间。要释放这部分存储空间，需要不时清除它们。其中一种方式是删除不再需要的二进制日志文件，比如在做了完全备份操作之后，就可以删除此前的增量备份二进制日志文件了：

shell> **mysqldump --single-transaction --flush-logs --master-data=2 \**

**--all-databases --delete-master-logs > backup\_sunday\_1\_PM.sql**

**Note**

**注意**

Deleting the MySQL binary logs with mysqldump --delete-master-logs can

be dangerous if your server is a replication master server, because slave servers might not yet fully have processed the contents of the binary log. The description for the PURGE BINARY LOGS statement explains what should be verified before deleting the MySQL binary logs. See Section 14.4.1.1, “PURGE BINARY LOGS Syntax”.

如果你的服务器是主从复制的MySQL服务器，那么使用mysqldump --delete-master-logs命令删除MySQL二进制日志文件就是一个很危险的操作，因为从服务器还没有彻底的处理完二进制日志文件中的就内容。对于PURGE BINARY LOGS语句的描述解释了在删除MySQL二进制日志文件之前应该验证哪些内容。具体参看Section 14.4.1.1, “PURGE BINARY LOGS Syntax”。

**8.3.2 Using Backups for Recovery**

**8.3.2 利用备份文件进行恢复操作**

Now, suppose that we have a catastrophic crash on Wednesday at 8 a.m. that requires recovery from backups. To recover, first we restore the last full backup we have (the one from Sunday 1 p.m.). The full backup file is just a set of SQL statements, so restoring it is very easy:

现在，假设在周三早上8点中出现了灾难性崩溃，此时需要从备份中恢复数据库。为此，我们需要首先恢复最近一次的完整性备份（也就是周日下午1点钟做的完整备份）。完整性备份文件只是一个SQL语句集，所以，完整性恢复很简单，如下所示：

shell> **mysql < backup\_sunday\_1\_PM.sql**

At this point, the data is restored to its state as of Sunday 1 p.m.. To restore the changes made since then, we must use the incremental backups; that is, the gbichot2-bin.000007 and gbichot2-bin.000008 binary log files. Fetch the files if necessary from where they were backed up, and then process their contents like this:

此时，数据库恢复到周日下午1点钟的状态。要恢复从周日下午1点之后所做的更改，就必须要依赖于增量备份文件了；也就是gbichot2-bin.000007和gbichot2-bin.000008这两个二进制日志文件。找到这两个备份文件并执行如下操作：

shell> **mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql**

We now have recovered the data to its state as of Tuesday 1 p.m., but still are missing the changes from that date to the date of the crash. To not lose them, we would have needed to have the MySQL server store its MySQL binary logs into a safe location (RAID disks, SAN, ...) different from the place where it stores its data files, so that these logs were not on the destroyed disk. (That is, we can start the server with a --log-bin option that specifies a location on a different physical device from the one on which the data directory resides. That way, the logs are safe even if the device containing the directory is lost.) If we had done this, we would have the gbichot2-bin.000009 file (and any subsequent files) at hand, and we could apply them using mysqlbinlog and mysql to restore the most recent data changes with no loss up to the moment of the crash:

现在我们已经将数据库中的数据恢复到了周二的下午1点钟的状态，但是从周二下午1点钟到崩溃这段时间的数据库更改，仍然处于缺失状态。要想不丢失这些数据，我们需要将MySQL服务器上存储的二进制日志文件存储在一个相对安全的地方（比如RAID磁盘，SAN存储等等），而不是将这些二进制日志文件与数据文件存放在一起，这样一来，这些二进制日志文件就不会因为磁盘故障等原因而丢失了。（也就是说，我们可以在启动MySQL服务器的时候，通过--log-bin选项，给二进制日志文件指定一个不同于数据文件存储位置的其他安全的物理存储设备，这样一来，即便包含数据文件的路径从其设备中丢失，也不会影响到二进制日志文件）如果我们是这么做的，那么我们手头就会有gbichot2-bin.000009这个二进制文件（以及其他随后产生的二进制日志文件），我们可以通过mysqlbinlog和mysql命令，操作二进制日志文件将最近的数据更改操作（周二下午1点钟到周三早上8点钟数据库崩溃之前的这段时间的更改操作）操作恢复到崩溃的那个时刻。

shell> **mysqlbinlog gbichot2-bin.000009 ... | mysql**

For more information about using mysqlbinlog to process binary log files, see Section 8.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

关于使用mysqlbinlog处理二进制日志文件的更多信息，详见Section 8.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

**8.3.3 Backup Strategy Summary**

**8.3.3 备份策略总结**

In case of an operating system crash or power failure, InnoDB itself does all the job of recovering data. But to make sure that you can sleep well, observe the following guidelines:

要是操作系统崩溃或者电源故障，InnoDB引擎本身会做所有的数据恢复操作。但是为了确保你能睡一个安稳觉，所以，建议遵从如下指南：

• Always run the MySQL server with the --log-bin option, or even --log-bin=*log\_name*, where the log file name is located on some safe media different from the drive on which the data directory is located. If you have such safe media, this technique can also be good for disk load balancing (which results in a performance improvement).

当运行MySQL服务器的时候，总是加上--log-bin选项，或者更进一步，使用--log-bin=*log\_name*选项详细指明二进制日志文件的存储路径以及文件名，这个选项可以将二进制日志文件的存储路径指定到一个不同于数据文件存储路径的其他安全的存储设备上。如果你拥有这样的一个安全存储设备，这对于磁盘的负载均衡也是有益处的（可以在一定程度上提高磁盘性能，“PS：应该是将频繁写硬盘操作的日志记录文件与数据文件分离，降低数据存储设备的非业务IO请求操作，以此提高数据记录设备的业务IO请求性能。by 祁志伟”）

• Make periodic full backups, using the mysqldump command shown earlier in Section 8.3.1, “Establishing a Backup Policy”, that makes an online, nonblocking backup.

周期性的执行完整性备份操作，使用此前Section 8.3.1, “Establishing a Backup Policy”介绍的mysqldump命令可以创建一个在线的非阻塞的完整性备份操作。

• Make periodic incremental backups by flushing the logs with FLUSH LOGS or mysqladmin flushlogs.

通过周期性的执行FLUSH LOGS指令或者执行mysqladmin flushlogs命令，实现周期性的增量备份。

**8.4 Using mysqldump for Backups**

**8.4 使用mysqldump进行备份操作**

This section describes how to use mysqldump to produce dump files, and how to reload dump files. A dump file can be used in several ways:

这部分主要描述如何使用mysqldump命令生成备份文件，以及如何重新装载备份文件到数据库中。生成的备份文件可以有以下几种方式应用：

• As a backup to enable data recovery in case of data loss.

用于防止数据丢失，备份文件可以用于恢复数据。

• As a source of data for setting up replication slaves.

用于设置从服务器，作为从服务器的数据源。

• As a source of data for experimentation:

用于实验测试数据的数据源。

• To make a copy of a database that you can use without changing the original data.

制作一个数据库的副本，这样就无需更改原始数据了。

• To test potential upgrade incompatibilities.

测试更新操作的潜在不兼容性。

mysqldump produces two types of output, depending on whether the --tab option is given:

mysqldump命令产生两种类型的输出，取决于执行命令的时候，是否制定了--tab选项：

• Without --tab, mysqldump writes SQL statements to the standard output. This output consists of CREATE statements to create dumped objects (databases, tables, stored routines, and so forth), and INSERT statements to load data into tables. The output can be saved in a file and reloaded later using mysql to recreate the dumped objects. Options are available to modify the format of the SQL statements, and to control which objects are dumped.

不使用--tab选项，那么mysqldump命令会将SQL语句写入到标准输出。这个输出中包含了创建备份的对象（数据库，表，存储过程等等）的CREATE语句，以及将数据装载入表的INSERT语句。这个输出可以保存在文件中，以便后面使用mysql命令重新装载，来重新创建备份的对象。可以通过多个选项指定SQL语句的输出形式，以及控制那些对象需要备份。

• With --tab, mysqldump produces two output files for each dumped table. The server writes one file as tab-delimited text, one line per table row. This file is named *tbl\_name*.txt in the output directory.

使用--tab选项，mysqldump命令会为每个备份的表产生两个输出文件。服务器创建一个以tab分割的文本文件，表中每行对应文本文件中的一行。这个文件以*tbl\_name*.txt的命名方式存储在输出路径中。

The server also sends a CREATE TABLE statement for the table to mysqldump, which writes it as a file named *tbl\_name*.sql in the output directory.

与此同时服务器也会发送CREATE TABLE语句到一个名为*tbl\_name*.sql的文件，并将其存储在输出路径中。

**8.4.1 Dumping Data in SQL Format with mysqldump**

**8.4.1 使用mysqldump将数据备份为SQL格式**

This section describes how to use mysqldump to create SQL-format dump files. For information about reloading such dump files, see Section 8.4.2, “Reloading SQL-Format Backups”.

这部分主要描述如何使用mysqldump命令创建SQL格式的备份文件。关于如何重新装载生成的备份文件（即数据恢复操作），参见Section 8.4.2, “Reloading SQL-Format Backups”.

By default, mysqldump writes information as SQL statements to the standard output. You can save the output in a file:

默认情况下，mysqldump命令将信息以SQL语句的形式写入到标准输出中。可以用如下命令形式将输出保存在文件中：

shell> **mysqldump [*arguments*] > *file\_name***

To dump all databases, invoke mysqldump with the --all-databases option:

为了备份所有的数据库，在调用mysqldump命令的时候，加上--all-databases选项。

shell> **mysqldump --all-databases > dump.sql**

To dump only specific databases, name them on the command line and use the --databases option:

如果只是要备份特定的数据库，使用--databases选项，并将这些要备份的数据库名称写在这个选项的后面即可：

shell> **mysqldump --databases db1 db2 db3 > dump.sql**

The --databases option causes all names on the command line to be treated as database names.

通过使用--databases选项可以将其后的所有名字都作为数据库名字来对待。

Without this option, mysqldump treats the first name as a database name and those following as table names.

如果不使用这个选项，那么mysqldump命令会将第一个名字作为数据库名字，而将其后的名字都作为表名来对待。

With --all-databases or --databases, mysqldump writes CREATE DATABASE and USE statements prior to the dump output for each database. This ensures that when the dump file is reloaded, it creates each database if it does not exist and makes it the default database so database contents are loaded into the same database from which they came. If you want to cause the dump file to force a drop of each database before recreating it, use the --add-drop-database option as well. In this case, mysqldump writes a DROP DATABASE statement preceding each CREATE DATABASE statement.

通过--all-databases选项或者--databases选项，mysqldump命令可以为每个数据库将CREATE DATABASE和 USE写入到备份输出中。这就可以保证在重新装载备份的文件时候，如果不存在这个数据库可以自动创建，并且可以作为默认的数据库，以便数据库内容可以被重新装载到与原始数据库同名的数据库中。如果要想要使备份文件创建的时候，重新创建所有的数据库，那么就需要使用--add-drop-database选项。在这种情况下，mysqldump命令会将DROP DATABASE语句加在CREATE DATABASE语句之前。

To dump a single database, name it on the command line:

如果要备份单一数据库，将其名字列在命令行中即可，如下所示：

shell> **mysqldump --databases test > dump.sql**

In the single-database case, it is permissible to omit the --databases option:

在单一数据库备份的例子中，允许忽略掉--databases选项：

shell> **mysqldump test > dump.sql**

The difference between the two preceding commands is that without --databases, the dump output contains no CREATE DATABASE or USE statements. This has several implications:

前面两条命令的差异，主要是如果没有--databases选项的话，备份输出中就不会包含CREATE DATABASE或者USE语句。这有多种含义：

• When you reload the dump file, you must specify a default database name so that the server knows which database to reload.

当你重新装载备份文件的时候，你必须手动指定要恢复的数据库名字，如此服务器才知道要恢复到哪一个数据库中。

• For reloading, you can specify a database name different from the original name, which enables you to reload the data into a different database.

对于重新装载（数据库恢复操作），也可以指定一个不同于初始名字的数据库名字，这样就可以将数据恢复到一个不同的数据库中了。

• If the database to be reloaded does not exist, you must create it first.

如果要被写入备份的数据的数据库不存在，那么就需要首先创建这个数据库，然后再执行恢复操作。

• Because the output will contain no CREATE DATABASE statement, the --add-drop-database option has no effect. If you use it, it produces no DROP DATABASE statement.

因为输出中不包含CREATE DATABASE语句，所以--add-drop-database选项就不会起作用。如果使用了这个--add-drop-database选项，其也不会产生DROP DATABASE语句。

To dump only specific tables from a database, name them on the command line following the database name:

如果只是备份数据库中特定的表，那么将这些表列在数据库名字的后面即可。如下所示：

shell> **mysqldump test t1 t3 t7 > dump.sql**

**8.4.2 Reloading SQL-Format Backups**

**8.4.2 重新装载SQL格式的备份**

To reload a dump file written by mysqldump that consists of SQL statements, use it as input to the mysql client. If the dump file was created by mysqldump with the --all-databases or --databases option, it contains CREATE DATABASE and USE statements and it is not necessary to specify a default database into which to load the data:

要重新装载mysqldump生成的包含SQL语句的备份文件，只需要将这个备份文件作为mysql命令的输入即可。如果备份文件是通过mysqldump命令加上--all-databases或者--databases选项生成的，那么备份文件中就会包含了CREATE DATABASE和USE语句，所以在恢复数据库的时候就无需指定默认数据库了。

shell> **mysql < dump.sql**

Alternatively, from within mysql, use a source command:

或者，在登录进mysql服务器之后，使用source命令也可以达到相同效果。

mysql> **source dump.sql**

If the file is a single-database dump not containing CREATE DATABASE and USE statements, create the database first (if necessary):

如果备份的是一个单一数据库，且备份文件中不包含CREATE DATABASE和USE语句，那么恢复数据库的时候就需要首先创建这个数据库（如果数据库中没有这个数据库的话）：

shell> **mysqladmin create db1**

Then specify the database name when you load the dump file:

然后在恢复数据库的时候，指定刚才创建的数据库名字即可：

shell> **mysql db1 < dump.sql**

Alternatively, from within mysql, create the database, select it as the default database, and load the dump file:

或者，可以登录进MySQL服务器，然后执行相应语句创建要恢复的数据库，然后使用这个数据库，并且通过source语句装载数据，具体如下所示：

mysql> **CREATE DATABASE IF NOT EXISTS db1;**

mysql> **USE db1;**

mysql> **source dump.sql**

**8.4.3 Dumping Data in Delimited-Text Format with mysqldump**

**8.4.3 使用mysqldump将数据备份到一个分隔文本格式的文件中**

This section describes how to use mysqldump to create delimited-text dump files. For information about reloading such dump files, see Section 8.4.4, “Reloading Delimited-Text Format Backups”.

这部分主要描述如何使用mysqldump创建分隔文本格式的备份文件。关于重新装载备份文件的信息，详见Section 8.4.4, “Reloading Delimited-Text Format Backups”.

If you invoke mysqldump with the --tab=*dir\_name* option, it uses *dir\_name* as the output directory and dumps tables individually in that directory using two files for each table. The table name is the base name for these files. For a table named t1, the files are named t1.sql and t1.txt. The .sql file contains a CREATE TABLE statement for the table. The .txt file contains the table data, one line per table row.

如果在执行mysqldump命令的时候指定--tab=*dir\_name*选项，其会使用作为*dir\_name*输出路径，并将每个备份的表分别使用两种文件存储在该目录中。表名作为这些文件的基名。对于表名t1，对应的两个文件分别为t1.sql和t1.txt。其中.sql文件包含了创建表的CREATE TABLE语句。.txt文件中包含了表数据，表中每行对应该文件中的一行。

The following command dumps the contents of the db1 database to files in the /tmp database:

下面的命令将数据库db1的内容备份到/tmp目录中，如下所示：

shell> **mysqldump --tab=/tmp db1**

The .txt files containing table data are written by the server, so they are owned by the system account used for running the server. The server uses SELECT ... INTO OUTFILE to write the files, so you must have the FILE privilege to perform this operation, and an error occurs if a given .txt file already exists.

.txt文件中包含了被服务器写入的表数据，所以其属于运行这个服务器的系统帐户（所有者关系）。服务器使用SELECT ... INTO OUTFILE语句写入到文件中，所以你必须要有FILE权限（MySQL的一个权限，登录数据库之后使用show privileges;命令即可查看）才可以执行这个操作，如果指定的.txt文件已经存在于目录中，那么就会报错。

The server sends the CREATE definitions for dumped tables to mysqldump, which writes them to .sql files. These files therefore are owned by the user who executes mysqldump.

在执行mysqldump的时候，服务器会将要备份的表的表定义语句写入到.sql文件中。所以，文件所有者就是执行mysqldump命令的用户。

It is best that --tab be used only for dumping a local server. If you use it with a remote server, the --tab directory must exist on both the local and remote hosts, and the .txt files will be written by the server in the remote directory (on the server host), whereas the .sql files will be written by mysqldump in the local directory (on the client host).

--tab选项最好在备份本地数据库服务器的时候使用。如果在备份远程数据库服务器的时候使用，那么就需要--tab指定的路径必须同时存在于本地和远程服务器上，而且.txt文件会被写入到远程服务器的路径中（在远程MySQL服务器主机上），而.sql文件将会被mysqldump命令写入到本地服务器的路径中（在本地客户端服务器上）。

For mysqldump --tab, the server by default writes table data to .txt files one line per row with tabs between column values, no quotation marks around column values, and newline as the line terminator.(These are the same defaults as for SELECT ... INTO OUTFILE.)

对于mysqldump --tab命令，MySQL服务器默认情况下会将表数据写入到.txt文件中，列之间用tab分隔，列值不会用引号括起来，并且使用换行符作为行结束符。（与SELECT ... INTO OUTFILE语句的效果相同。）

To enable data files to be written using a different format, mysqldump supports these options:

要想将数据文件以不同的格式写入，mysqldump支持三种选项：

• --fields-terminated-by=*str*

The string for separating column values (default: tab).

指定用于分隔列值的字符串*str*（默认是tab符）。

• --fields-enclosed-by=*char*

The character within which to enclose column values (default: no character).

指定包围列值的字符*char*（默认是不包围列值）。

• --fields-optionally-enclosed-by=*char*

The character within which to enclose non-numeric column values (default: no character).

指定包围非数字列值的字符*char*（默认不包围列值）。

• --fields-escaped-by=*char*

The character for escaping special characters (default: no escaping).

指定用于转义特殊字符的字符*cahr*（默认不转义）。

• --lines-terminated-by=*str*

The line-termination string (default: newline).

指定行结束字符串*str*（默认是换行符）。

Depending on the value you specify for any of these options, it might be necessary on the command line to quote or escape the value appropriately for your command interpreter. Alternatively, specify the value using hex notation. Suppose that you want mysqldump to quote column values within double quotation marks. To do so, specify double quote as the value for the --fields-enclosed-by option. But this character is often special to command interpreters and must be treated specially. For example, on Unix, you can quote the double quote like this:

对于你为这些选项所指定的值，需要针对你的命令解释器，对于这些值进行必要的引号包围或这转义。或者直接指定选项值的对应十六进制值作为替代。假设你想让mysqldump的输出中用双引号将列值包围起来。只需要这样做：在调用mysqldump的时候指定--fields-enclosed-by选项，并给其赋值为双引号。但是由于双引号对于命令解释器来说通常是特殊字符，所以必须特殊对待。比如，在Unix系统上，可以采用如下方式给这个选项指定值：

--fields-enclosed-by='"'

On any platform, you can specify the value in hex:

在其他平台上，可以将这个值指定为十六进制值的形式：

--fields-enclosed-by=0x22

It is common to use several of the data-formatting options together. For example, to dump tables in comma-separated values format with lines terminated by carriage-return/newline pairs (\r\n), use this command (enter it on a single line):

将多种数据格式选项一起使用是很常见的情况。比如，采用逗号分隔列值的形式，并且采用回车换行符作为行结束符（\r\n）的形式备份表，命令如下所示（在一行中输入如下命令）：

shell> **mysqldump --tab=/tmp --fields-terminated-by=,**

**--fields-enclosed-by='"' --lines-terminated-by=0x0d0a db1**

Should you use any of the data-formatting options to dump table data, you will need to specify the same format when you reload data files later, to ensure proper interpretation of the file contents.

备份表数据的时候，可以使用任何有效的数据格式化选项，同时，当你恢复数据的时候，也要明确指定备份数据时所用的数据格式化选项，来确保MySQL能够正确解析备份文件的内容。

**8.4.4 Reloading Delimited-Text Format Backups**

**8.4.4 重新装载分隔文本格式的备份文件**

For backups produced with mysqldump --tab, each table is represented in the output directory by an .sql file containing the CREATE TABLE statement for the table, and a .txt file containing the table data.

对于使用mysqldump --tab命令实现的备份，每个表都会在输出目录中被表示为包含CREATE TABLE语句的.sql文件，以及包含表内容的.txt文件。

To reload a table, first change location into the output directory. Then process the .sql file with mysql to create an empty table and process the .txt file to load the data into the table:

要重新装载表，首先将工作目录切换到备份的输出路径中。然后使用mysql命令处理.sql文件创建一个空的表，接着处理.txt文件装载数据到这个空表中：

shell> **mysql db1 < t1.sql**

shell> **mysqlimport db1 t1.txt**

An alternative to using mysqlimport to load the data file is to use the LOAD DATA INFILE statement from within the mysql client:

mysqlimport命令装载表数据的一个备选方案是登录到mysql服务器中，然后使用使用LOAD DATA INFILE语句：

mysql> **USE db1;**

mysql> **LOAD DATA INFILE 't1.txt' INTO TABLE t1;**

If you used any data-formatting options with mysqldump when you initially dumped the table, you must use the same options with mysqlimport or LOAD DATA INFILE to ensure proper interpretation of the data file contents:

如果你在备份数据库的时候指定了其他数据格式化选项，那么你在使用mysqlimport命令或者LOAD DATA INFILE语句重新装载数据的时候，必须要使用相同的选项，以确保服务器可以正确的解析数据文件中的内容：

shell> **mysqlimport --fields-terminated-by=, --fields-enclosed-by='"' --lines-terminated-by=0x0d0a db1 t1.txt**

Or:

或者：

mysql> **USE db1;**

mysql> **LOAD DATA INFILE 't1.txt' INTO TABLE t1**

-> **FIELDS TERMINATED BY ',' FIELDS ENCLOSED BY '"'**

-> **LINES TERMINATED BY '\r\n';**

**8.4.5 mysqldump Tips**

**8.4.5 mysqldump提示**

This section surveys techniques that enable you to use mysqldump to solve specific problems:

这部分对mysqldump能解决的特定问题进行概述：

• How to make a copy a database

怎么制作数据库副本

• How to copy a database from one server to another

怎样将数据库副本从一台服务器拷贝到另一台服务器

• How to dump stored programs (stored procedures and functions, triggers, and events)

怎样备份存储程序（包括存储过程和存储函数，触发器以及事件）

• How to dump definitions and data separately

怎样将数据库定义部分和数据部分分开备份

**8.4.5.1 Making a Copy of a Database**

**8.4.5.1 创建数据库副本**

shell> **mysqldump db1 > dump.sql**

shell> **mysqladmin create db2**

shell> **mysql db2 < dump.sql**

Do not use --databases on the mysqldump command line because that causes USE db1 to be included in the dump file, which overrides the effect of naming db2 on the mysql command line.

不要在mysqldump命令行中使用--databases选项，因为这会导致USE db1语句被包含在备份文件中，其会覆盖掉mysql命令行中指定的db2数据库。

**8.4.5.2 Copy a Database from one Server to Another**

**8.4.5.2 将数据库从一台服务器拷贝到其他服务器**

On Server 1:

在Server1上：

shell> **mysqldump --databases db1 > dump.sql**

Copy the dump file from Server 1 to Server 2.

On Server 2:

将备份文件从Server1拷贝到Server2。

在Server2上：

shell> **mysql < dump.sql**

Use of --databases with the mysqldump command line causes the dump file to include CREATE DATABASE and USE statements that create the database if it does exist and make it the default database for the reloaded data.

Alternatively, you can omit --databases from the mysqldump command. Then you will need to create the database on Server 2 (if necessary) and specify it as the default database when you reload the dump file.

On Server 1:

在mysqldump命令上使用--databases选项会导致备份文件中包含CREATE DATABASE和USE语句，这两个语句会创建数据库并将该数据库作为恢复的默认数据库使用。

或者，你在执行mysqldump命令的时候不指定--databases选项。然后需要在Server2服务器上手动创建这个数据库（如果该库不存在的话）并且将其指定为重新装载备份文件的默认数据库。、

在Server1上：

shell> **mysqldump db1 > dump.sql**

On Server 2:

在Server2上：

shell> **mysqladmin create db1**

shell> **mysql db1 < dump.sql**

You can specify a different database name in this case, so omitting

--databases from the mysqldump command enables you to dump data from one database and load it into another.

在这个案例中可以指定一个不同的数据库名字，所以在使用不带--databases选项的mysqldump命令备份数据库的时候，可以允许你将从一个数据库中备份的数据恢复到其他数据库中。

**8.4.5.3 Dumping Stored Programs**

**8.4.5.3 备份存储程序**

Several options control how mysqldump handles stored programs (stored procedures and functions, triggers, and events):

有多个选项可以控制mysqldump命令如何处理存储程序（包括存储过程和存储函数，触发器以及事件）：

• --events: Dump Event Scheduler events

--events: 备份事件调度器事件

• --routines: Dump stored procedures and functions

--routines: 备份存储过程和存储函数

• --triggers: Dump triggers for tables

--triggers: 备份表的触发器

The --triggers option is enabled by default so that when tables are dumped, they are accompanied by any triggers they have. The other options are disabled by default and must be specified explicitly to dump the corresponding objects. To disable any of these options explicitly, use its skip form: --skip-events, --skip-routines, or --skip-triggers.

默认情况下，--triggers选项是打开的，所以当备份表的时候，也会将表的所有触发器都备份起来。其他选项默认情况下是关闭的，所以在备份相关对象的时候如果有需要必须明确指定打开特定选项。如果要明确地关闭这些选项，使用其的略过模式，即不指定选项值：--skip-events, --skip-routines, 或者 --skip-triggers。

**8.4.5.4 Dumping Table Definitions and Content Separately**

**8.4.5.4 将表定义部分和表数据分开备份**

The --no-data option tells mysqldump not to dump table data, resulting in the dump file containing only statements to create the tables. Conversely, the --no-create-info option tells mysqldump to suppress CREATE statements from the output, so that the dump file contains only table data.

--no-data选项使mysqldump命令不备份表数据，会导致备份文件中只包含创建表的定义语句。反过来，--no-create-info选项使mysqldump从输出中压制表创建的CREATE语句，导致备份文件中只包含表数据。

For example, to dump table definitions and data separately for the test database, use these commands:

比如，如果要将test数据库的表定义和表数据分开定义，使用如下命令：

shell> **mysqldump --no-data test > dump-defs.sql**

shell> **mysqldump --no-create-info test > dump-data.sql**

For a definition-only dump, add the --routines and --events options to also include stored routine and event definitions:

对于一个只包含表定义的备份，备份的时候加上--routines和--events选项，那么也可以备份其中的存储过程和事件定义语句：

shell> **mysqldump --no-data --routines --events test > dump-defs.sql**

**8.4.5.5 Using mysqldump to Test for Upgrade Incompatibilities**

**8.4.5.5 使用mysqldump测试更新的不兼容性**

When contemplating a MySQL upgrade, it is prudent to install the newer version separately from your current production version. Then you can dump the database and database object definitions from the production server and load them into the new server to verify that they are handled properly. (This is also useful for testing downgrades.)

On the production server:

当考虑MySQL更新的时候，将更新的MySQL版本与生产环境中的MySQL分开安装是明智的。然后可以将数据库和数据库对象定义部分从生产环境服务器中备份下来，并在新服务器中进行装载恢复，以此来验证新服务器是否能正确的处理这些数据。（同理，这对于MySQL服务器降级也是适用的。）

在生产环境服务器上：

shell> **mysqldump --all-databases --no-data --routines --events > dump-defs.sql**

On the upgraded server:

在更新服务器上：

shell> **mysql < dump-defs.sql**

Because the dump file does not contain table data, it can be processed quickly. This enables you to spot potential incompatibilities without waiting for lengthy data-loading operations. Look for warnings or errors while the dump file is being processed.

因为备份文件中不包含表数据，所以处理起来就很快。这可以使你快速发现不兼容的问题，而不用等待执行很久的数据装载操作。在处理备份文件的过程中找到错误和警告信息。

After you have verified that the definitions are handled properly, dump the data and try to load it into the upgraded server.

On the production server:

当你验证了这些表定义部分被正确处理之后，备份数据并尝试将其装载到更新版本的MySQL服务器中。

在生产环境中的服务器上：

shell> **mysqldump --all-databases --no-create-info > dump-data.sql**

On the upgraded server:

在更新版本的服务器上：

shell> **mysql < dump-data.sql**

Now check the table contents and run some test queries.

full backup to a more recent time.

Point-in-time recovery is based on these principles:

现在可以检查表内容并且运行一些测试查询操作。

完整性备份到最近的时间。

时间点恢复基于如下原则：

• The source of information for point-in-time recovery is the set of incremental backups represented by the binary log files generated subsequent to the full backup operation. Therefore, the server must be started with the --log-bin option to enable binary logging (see Section 6.4.4, “The Binary Log”).

时间点恢复的信息源来自于完整性备份之后所做的一组增量备份生成的二进制日志文件。所以，为了获得二进制日志文件，在启动MySQL服务器的时候，必须要加上--log-bin选项，如此MySQL服务器才能使能二进制日志记录功能（详见Section 6.4.4, “The Binary Log”）。

To restore data from the binary log, you must know the name and location of the current binary log files.

要从二进制日志文件中恢复数据，必须要知道二进制日志的名字以及当前记录的二进制日志文件的存储位置。

By default, the server creates binary log files in the data directory, but a path name can be specified with the --log-bin option to place the files in a different location. Section 6.4.4, “The Binary Log”.

默认情况下，MySQL服务器将二进制日志文件记录在数据目录中，但是可以在启动MySQL服务的时候通过--log-bin选项指定一个不同的存储位置来存放服务器生成的二进制日志文件。详见：Section 6.4.4, “The Binary Log”.

To see a listing of all binary log files, use this statement:

要查看所有的二进制日志文件，使用下面的语句即可：

mysql> **SHOW BINARY LOGS;**

To determine the name of the current binary log file, issue the following statement:

要确定当前二进制日志文件的名字，执行下面的语句：

mysql> **SHOW MASTER STATUS;**

• The mysqlbinlog utility converts the events in the binary log files from binary format to text so that they can be executed or viewed. mysqlbinlog has options for selecting sections of the binary log based on event times or position of events within the log. See Section 5.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”.

mysqlbinlog应用程序可以将二进制日志文件中的事件从二进制格式转换为文本文件，以便于其可以被执行和查看。mysqlbinlog程序提供了一些选项，可以基于日志文件中的时间事件或者事件位置选择特定的二进制日志部分。详见：Section 5.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”.

• Executing events from the binary log causes the data modifications they represent to be redone. This enables recovery of data changes for a given span of time. To execute events from the binary log, process mysqlbinlog output using the mysql client:

从二进制日志文件中执行事件，会导致其表现出来的数据更改操作被重做。这就可以使数据更改恢复到一个给定的时间跨度上。要从二进制日志文件中执行事件，需要使用mysql命令处理mysqlbinlog的输出。如下所示：

shell> **mysqlbinlog *binlog\_files* | mysql -u root -p**

• Viewing log contents can be useful when you need to determine event times or positions to select partial log contents prior to executing events. To view events from the log, send mysqlbinlog output into a paging program:

查看二进制日志文件的内容对于确定你所需要的事件时间或者事件位置，并以此来选择其中的特定部分事件进行执行是很有用的。要从二进制日志文件中查看事件，可以将mysqlbinlog的输出发送给翻页程序（比如Linux里面的less命令或者more命令）：

shell> **mysqlbinlog *binlog\_files* | more**

Alternatively, save the output in a file and view the file in a text editor:

或者，将mysqlbinlog的输出保存在一个文件中，随后在文本编辑器中查看该文本文件的内容：

shell> **mysqlbinlog *binlog\_files* > tmpfile**

shell> ... *edit tmpfile* ...

• Saving the output in a file is useful as a preliminary to executing the log contents with certain events removed, such as an accidental DROP DATABASE. You can delete from the file any statements not to be executed before executing its contents. After editing the file, execute the contents as follows:

将mysqlbinlog的输出保存在文本文件中，对于在执行恢复操作之前将其中某些事件删除，有助于剔除掉不需要执行的操作，比如偶然的删除数据库的操作DROP DATABASE。你可以在执行特定事件之前，从重定向生成的文本文件中删除掉任何不想被执行的语句。在编辑完这个文件之后，执行下面的内容：

shell> **mysql -u root -p < tmpfile**

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

如果在MySQL服务器上，有多个二进制日志文件需要处理并执行，比较安全的做法是使用同一个服务器连接进程处理所有的二进制日志文件。下面是一个证明可能不安全的做法的示例：

shell> **mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!**

shell> **mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!**

Processing binary logs this way using different connections to the server causes problems if the first log file contains a CREATE TEMPORARY TABLE statement and the second log contains a statement that uses the temporary table. When the first mysql process terminates, the server drops the temporary table. When the second mysql process attempts to use the table, the server reports “unknown table.”

这种使用不同的服务器连接进程来处理二进制日志文件的方式，可能会导致一些问题，如果第一个日志文件中包含了CREATE TEMPORARY TABLE语句，而第二个日志文件中包含了使用这个临时表的语句。那么当第一个mysql进程结束的时候，MySQL服务器就会删掉第一个日志文件中生成的临时表。当第二个mysql进程尝试使用这个临时表的时候，MySQL服务器会报出以一个“unknown table.”的错误。

To avoid problems like this, use a *single* connection to execute the contents of all binary logs that you want to process. Here is one way to do so:

要避免类似问题，需要使用单一连接进程（即同一个MySQL连接进程）来执行所有需要处理的二进制日志文件处理流程。下面就是这种做法的一种方式：

shell> **mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p**

Another approach is to write all the logs to a single file and then process the file:

其他可选做法是将所有待处理的二进制日志文件的处理输出写入到一个中间文本文件中，然后处理这个中间文本文件。如下所示：

shell> **mysqlbinlog binlog.000001 > /tmp/statements.sql**

shell> **mysqlbinlog binlog.000002 >> /tmp/statements.sql**

shell> **mysql -u root -p -e "source /tmp/statements.sql"**

When writing to a dump file while reading back from a binary log containing GTIDs (see Section 18.1.3, “Replication with Global Transaction Identifiers”), use the --skip-gtids option with mysqlbinlog, like this:

当从包含GTIDs的二进制日志文件中读取内容并写入到备份文件(从二进制日志处理结果重定向输出的.sql文件)的时候（详见Section 18.1.3, “Replication with Global Transaction Identifiers”），需要在mysqlbinlog中使用--skip-gtids选项。就像下面这样：

shell> **mysqlbinlog --skip-gtids binlog.000001 > /tmp/dump.sql**

shell> **mysqlbinlog --skip-gtids binlog.000002 >> /tmp/dump.sql**

shell> **mysql -u root -p -e "source /tmp/dump.sql"**

**8.5.1 Point-in-Time Recovery Using Event Times**

**8.5.1 使用事件事件进行时间点恢复**

To indicate the start and end times for recovery, specify the --start-datetime and --stop-datetime options for mysqlbinlog, in DATETIME format. As an example, suppose that exactly at 10:00 a.m. on April 20, 2005 an SQL statement was executed that deleted a large table. To restore the table and data, you could restore the previous night's backup, and then execute the following command:

要指明恢复的开始和结束时间点，可以在执行mysqlbinlog的时候通过--start-datetime和--stop-datetime选项来指明起始和结束时间点，时间点采用DATETIME格式。作为示例，假设一个精确时间点是2005年4月20日上午10点整，执行SQL语句删除了一个大容量表。要恢复这个表以及其中的数据，需要恢复到此前夜里的备份，然后执行如下命令：

shell> **mysqlbinlog --stop-datetime="2005-04-20 9:59:59" \**

**/var/log/mysql/bin.123456 | mysql -u root -p**

This command recovers all of the data up until the date and time given by the --stop-datetime option.

这个命令可以将--stop-datetime选项给定的终止时间点范围内容的所有数据更改操作悉数恢复。

If you did not detect the erroneous SQL statement that was entered until hours later, you will probably also want to recover the activity that occurred afterward. Based on this, you could run mysqlbinlog again with a start date and time, like so:

如果你在执行了SQL语句之后的几个小时才检测到语句的错误，也需要恢复那个操作之后的数据库活动情况。基于此，可以再次运行mysqlbinlog程序，不过这次是通过**--start-datetime**选项指定一个起始时间点，如下所示：

shell> **mysqlbinlog --start-datetime="2005-04-20 10:01:00" \**

**/var/log/mysql/bin.123456 | mysql -u root -p**

In this command, the SQL statements logged from 10:01 a.m. on will be re-executed. The combination of restoring of the previous night's dump file and the two mysqlbinlog commands restores everything up until one second before 10:00 a.m. and everything from 10:01 a.m. on.

在这个命令中，SQL语句是从上午10:01开始记录的这部分内容会被重新执行。之前夜里的备份文件以及这两个mysqlbinlog命令组合起来执行的恢复操作，可以恢复上午10:00之前1秒到上午10:01之间的所有操作。

To use this method of point-in-time recovery, you should examine the log to be sure of the exact times to specify for the commands. To display the log file contents without executing them, use this command:

要使用这个时间点恢复的方法，需要检查日志以确保给命令指定了精确的时间。

shell> **mysqlbinlog /var/log/mysql/bin.123456 > /tmp/mysql\_restore.sql**

Then open the /tmp/mysql\_restore.sql file with a text editor to examine it.

Excluding specific changes by specifying times for mysqlbinlog does not work well if multiple statements executed at the same time as the one to be excluded.

然后使用文本编辑器打开/tmp/mysql\_restore.sql文件并对其进行检查。排除特定时间上mysqlbinlog没有很好执行的特定更改，比如如果在同一时间有多重语句被执行，那么这个需要被排除。（Excluding specific changes by specifying times for mysqlbinlog does not work well if multiple statements executed at the same time as the one to be excluded.这句的翻译不太确定）

**8.5.2 Point-in-Time Recovery Using Event Positions**

**8.5.2 使用事件位置来执行时间点恢复**

Instead of specifying dates and times, the --start-position and --stop-position options for mysqlbinlog can be used for specifying log positions. They work the same as the start and stop date options, except that you specify log position numbers rather than dates. Using positions may enable you to be more precise about which part of the log to recover, especially if many transactions occurred around the same time as a damaging SQL statement. To determine the position numbers, run mysqlbinlog for a range of times near the time when the unwanted transaction was executed, but redirect the results to a text file for examination. This can be done like so:

除了可以在恢复的时候指定日期和时间之外，还可以在调用mysqlbinlog命令的时候使用--start-position和--stop-position选项来指定要执行的日志的位置。其与前面介绍的起始和结束时间点选项的工作方式相同，只不过是指定日志位置号而不是日期。使用位置可以使你更精确的控制执行哪一部分日志进行恢复操作，特别是当有一个有损害的SQL语句的同时伴随着很多事务（transactions）发生（especially if many transactions occurred around the same time as a damaging SQL statement.这一句的翻译不太肯定）。要确定位置好，执行mysqlbinlog的时候，在不想要被执行的事务的时间点附近，指定一个时间范围，但是重定向结果到一个文本文件以供检查之用。可以像如下这样操作：

shell> **mysqlbinlog --start-datetime="2005-04-20 9:55:00" \**

**--stop-datetime="2005-04-20 10:05:00" \**

**/var/log/mysql/bin.123456 > /tmp/mysql\_restore.sql**

This command creates a small text file in the /tmp directory that contains the SQL statements around the time that the deleterious SQL statement was executed. Open this file with a text editor and look for the statement that you do not want to repeat. Determine the positions in the binary log for stopping and resuming the recovery and make note of them. Positions are labeled as log\_pos followed by a number.

这个命令会在/tmp目录中创建一个包含被执行的有害SQL语句的时间点附近的SQL语句。使用文本编辑器打开这个文件，并且查找到不想要执行的语句。通过停止和恢复所执行的数据库恢复操作，以此在二进制日志文件中标注出这些不想被执行的语句的位置。位置被标记形式为log\_pos后接一个数字。

After restoring the previous backup file, use the position numbers to process the binary log file. For example, you would use commands something like these:

在恢复之前的备份文件的时候，使用位置号来处理二进制日志文件。比如，可以使用如下形式的命令：

shell> **mysqlbinlog --stop-position=368312 /var/log/mysql/bin.123456 \**

**| mysql -u root -p**

shell> **mysqlbinlog --start-position=368315 /var/log/mysql/bin.123456 \**

**| mysql -u root -p**

The first command recovers all the transactions up until the stop position given. The second command recovers all transactions from the starting position given until the end of the binary log. Because the output of mysqlbinlog includes SET TIMESTAMP statements before each SQL statement recorded, the recovered data and related MySQL logs will reflect the original times at which the transactions were executed.

第一个命令恢复指定的停止位置之前的所有事务。第二个命令恢复指定的开始位置一直到二进制日志文件结尾的所有事务。由于mysqlbinlog的输出中，每个SQL语句记录之前都包含了SET TIMESTAMP语句，所以被恢复的数据以及相关的MySQL日志将会反应被执行的事务的初始执行时间。

2018-07-22 17:04

PS: MySQL开启bin-log的配置文件

