07 掌握数据备份与恢复是很有必要的

更新时间：2020-03-16 09:45:13

> 人不可有傲气，但不可无傲骨。——徐悲鸿

在我们的日常生活中，备份数据是常有的事，包括了照片、歌曲、文章等等。其目的就是当这些数据由于某些原因找不到的时候，也不至于丢失。但是，对于 MySQL 来说，备份数据就显得格外重要了，毕竟数据是 MySQL 存在的基础。

通常来说，数据的备份与恢复也是 DBA 的本职工作，但是，也有很多时候 DBA 不会帮你做这些事（数据量太少、创业型公司没有专职的 DBA 等等），这就需要你去负责这些工作了。甚至，你会慢慢发现，数据备份与恢复是一件很有意思的事，因为这件事确实是“无孔不入”。下面，我将会围绕这一节的标题“备份”、“恢复”来做讲解。

1 浅谈数据备份与恢复

既然说到数据是需要备份的，那么，数据为什么需要备份呢 ？又怎样去制定备份的策略呢 ？数据备份以后，怎样验证 MySQL 中的数据已经备份且不存在错误呢 ？

好吧，先停一停，问题先说到这里。你是不是也已经感觉到了数据的备份与恢复不是想象中的那么简单，不仅仅是复制与粘贴的过程。是的，这里面确实是有很多问题值得思考。那么，带着这些疑问，开启我们的路程吧。

1.1 为什么数据是需要备份的呢 ？

其实这个问题可以换一种问法：在哪些情况下可能会造成数据的丢失呢？乍一看，这个问题多简单呀，磁盘坏了，机房着火了等等都可能会造成数据丢失呀。这个回答当然是没问题的，但是没有经过系统的梳理，也就自然无法避免数据再次丢失的情况。理清楚这个问题其实是非常重要的，因为你只有找到了问题所在，才能去想办法避免问题的再次发生，也就是及时的做到“亡羊补牢”。

生产环境中，造成数据丢失的原因纷繁复杂，总结下来，有这样的几类：

- 硬件故障：多半是出于硬件老化引起的磁盘故障
- 软件故障：例如软件不兼容，内核版本升级造成的 MySQL 服务器错误
- 不可抗力：这是由于自然灾害导致，例如：洪水、地震等造成的服务器损坏
- 人为原因：人为导致的数据丢失又可以再细分为两类：
    - 恶意攻击：常见的场景是恶意的 SQL 注入删表删库
    - 误操作：敲错了字母、脑袋一懵都可能引起数据的误删除

从以上总结的各种情况可以知道，人为原因造成的数据丢失是最容易（相对来说）避免的。为了防止 SQL 注入，你可以在代码层面做一些检查，同时，很多 ORM 框架也帮助你完成了这一步骤。而对于误操作，也有两种方式去做到尽量避免。下面，我来给出示例说明。

```
-- 可以在执行语句之前加上 # 符号，代表注释语句，即使敲回车也不会真正去执行
-- 在检查确认无误后再把 # 去掉，执行删除语句
mysql> # DELETE FROM worker WHERE id = 11;

-- DELETE 语句也可以加上 LIMIT，限制受影响的行记录数
-- 在删除数据之前，先确定下影响的行数，再用 LIMIT 去限制，MySQL 就一定不会多删除数据
mysql> DELETE FROM worker WHERE id > 11 LIMIT 1;
Query OK, 1 row affected (0.03 sec)
```

1.2 怎样制定备份策略

粗略来看，这其实是一个比较简单的问题，核心点在于确定数据的重要程度。重要的数据当然要经常备份，而且要多备份；普通的数据则可以放宽限制。也就是说，备份数据的策略是与实际的应用场景相绑定的，需要认真思考以下的几个问题：

- 数据的重要性分级：如果是对账型的数据，当然应该是比较高的级别；如果只是报表型的数据，级别可以定的低一些。到底怎样分级，还需要结合你对业务的理解程度
- 最多能够容忍丢失多少数据：这项要求一般以时间为标准，如果定义为 1 小时，则至少每小时都需要做一次备份
- 恢复数据的时长：如果是线上用户数据，那么，数据恢复应该是越快越好，这就需要备份的数据有良好的数据结构，易于读取，快速恢复

只是简单的备份数据还并不完整，因为我们还不能确定备份的数据是否一定是对的、一定是可用的，毕竟备份数据只是一份（多份）文件。需要：

- 定期的做还原测试：可以准备测试机器，定期的使用备份文件做还原测试，以检验备份文件的可用性
- 对数据逻辑进行测试：使用备份文件进行数据恢复后，只能证明数据是可用的。但是，还需要保证这些数据是合理的，是正确的。需要使用测试代码对数据的逻辑进行校验

备份数据可以再分为两种类型：

- 全量备份：它是说对所有的库和表进行备份，优点是始终保持当前时间的最新备份，恢复更快。缺点是备份数据量大，需要花费更多的时间，对 MySQL 服务器系统的压力也较大
- 增量备份：它是说去备份每天的增量日志（Binlog），优点是备份时间短，MySQL 负载压力小。缺点是恢复数据慢（由于需要重放用户的操作日志）

好的，到这里，我们已经搞清楚了 MySQL 数据备份与恢复的思想。也就是从理论层面认识了备份与恢复，接下来，我去讲解下数据备份与恢复的实践操作。

2 备份数据的方法

可用于 MySQL 备份数据的方法与工具有很多，而且需要考虑不同的存储引擎是否都适用，为此又出现了逻辑备份和物理备份的概念。在 MySQL 中，逻辑备份对各种存储引擎都可以使用同样的方法来备份；而物理备份则不同，不同的存储引擎有不同的备份方法。所以，为了追求简单且通用的目标，我们可以选择使用 mysqldump 工具（由 MySQL 提供，安装了 MySQL，也就自动的安装了 mysqldump）来做数据备份。

2.1 前置准备工作

首先，我们先去做一些简单的前置准备工作：创建库和表，并插入一些测试性的数据。创建库和表的语句如下所示（只是作为示例，任意发挥即可）：

```
-- 创建 imooc_mysql 库（如果当前已经存在了，会报错）
CREATE DATABASE imooc_mysql;

-- 创建 ad_unit 表
CREATE TABLE `ad_unit` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '自增主键',
  `user_id` bigint(20) NOT NULL DEFAULT '0' COMMENT '标记当前记录所属用户',
  `unit_name` varchar(48) NOT NULL COMMENT '推广单元名称',
  `unit_status` tinyint(4) NOT NULL DEFAULT '0' COMMENT '推广单元状态: 0-正常, 1-失效',
  `position_type` tinyint(4) NOT NULL DEFAULT '0' COMMENT '广告位类型(1,2,3)',
  `budget` bigint(20) NOT NULL COMMENT '预算',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=14 DEFAULT CHARSET=utf8 COMMENT='广告-推广单元表';

-- 创建 worker 表
CREATE TABLE `worker` (
  `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT COMMENT 'id',
  `type` char(64) NOT NULL DEFAULT '' COMMENT '员工类型',
  `name` char(64) NOT NULL,
  `salary` bigint(20) unsigned DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=15 DEFAULT CHARSET=utf8 COMMENT='员工表';
```

创建了库和表之后，再去插入一些测试数据，语句如下：

```
-- 插入测试数据到 ad_unit 表
INSERT INTO `ad_unit` (`id`, `user_id`, `unit_name`, `unit_status`, `position_type`, `budget`)
VALUES
 (1, 1001, '推广单元-01', 0, 1, 1200),
 (2, 1001, '推广单元-02', 0, 1, 1500),
 (3, 1001, '推广单元-03', 0, 2, 1700),
 (4, 1001, '推广单元-04', 1, 1, 2500),
 (5, 1002, '推广单元-05', 0, 1, 2000),
 (6, 1002, '推广单元-06', 0, 3, 1000),
 (7, 1003, '推广单元-07', 0, 1, 3400),
 (8, 1003, '推广单元-08', 0, 2, 2100),
 (9, 1004, '推广单元-09', 0, 1, 1600),
 (10, 1004, '推广单元-10', 0, 1, 1100),
 (11, 1004, '推广单元-11', 0, 2, 3500),
 (12, 1004, '推广单元-12', 0, 3, 1900),
 (13, 1004, '推广单元-13', 0, 3, 3200);

-- 插入测试数据到 worker 表
INSERT INTO `worker` (`id`, `type`, `name`, `salary`)
VALUES
 (1, 'A', 'tom', 1800),
 (2, 'B', 'jack', 2100),
 (3, 'C', 'pony', NULL),
 (4, 'B', 'tony', 3600),
 (5, 'B', 'marry', 1900),
 (6, 'C', 'tack', 1200),
 (7, 'A', 'tick', NULL),
 (8, 'B', 'clock', 2000),
 (9, 'C', 'noah', 1500),
 (10, 'C', 'jarvis', 1800);
```

好棒！目前为止，库、表、数据都已经有了，可以开始正式学习 mysqldump 的使用方法了。这里建议大家一定要动手去做尝试，学习和理解知识点的最好方式就是实践。

2.2 mysqldump 工具的常见备份方法

既然 mysqldump 是一个工具，它大概率会提供一些选项的功能。经过查询资料，确定了我的这个想法，不过可怕的是：它的选项真多！但是，你也可以猜到，实际上常用的选项也就那么几个。我们先来看一看它的基本语法（在 shell 或是 cmd 中执行）：

```
 ~ mysqldump -help
Usage: mysqldump [OPTIONS] database [tables]
OR     mysqldump [OPTIONS] --databases [OPTIONS] DB1 [DB2 DB3...]
OR     mysqldump [OPTIONS] --all-databases [OPTIONS]
For more options, use mysqldump --help
```

其中，OPTIONS 是选项的意思，我们可以按需指定。–databases 用于指定一个或多个库；–all-databases 则代表备份所有的库。同时，最后一句打印的内容也提示了，更加详细的选项可以使用 mysqldump --help 查看。接下来，我们就使用 mysqldump 工具来做数据备份。

首先，我们来看一个最简单的，导出所有的数据库：

```
 ~ mysqldump -uroot -proot --all-databases > ~/QinyiZhang/mysql_backup_data/all_databases.sql
```

需要注意，-u 和 -p 指的是用户名和密码，这个需要与你设定的保持一致；> 后面是设定的导出文件保存位置。经过总结、查看导出文件，我们可以得出如下结论：

- mysqldump 默认会创建 SQL 格式的转储文件来备份数据库；

- mysqldump 默认会生成 INSERT 语句来备份数据；
- mysqldump 不仅备份了数据，同时也包含了库和表的定义语句；
- mysqldump 也会备份系统库，但是只有 mysql（系统库）。

搞清楚了 mysqldump 工具的基本语法，更多的用法只是各种选项的组合。下面，我们来一起看看更多常见的用法，同时，也就搞清楚了常见的选项代表怎样的含义（继续之前，可以再去看看导出的文件）。

```
# 导出 imooc_mysql 库的所有数据，如果有更多的库，空格隔开即可
mysqldump -uroot -proot --databases imooc_mysql > ~/QinyiZhang/mysql_backup_data/imooc_mysql.sql

# 导出 imooc_mysql 库中的 worker 表数据，如果有更多的表，空格隔开即可
# 需要注意，导出指定表只能针对一个库，且导出指定表的 SQL 中没有创建库的语句，只有删除表、创建表、插入数据
mysqldump -uroot -proot --databases imooc_mysql --tables worker > ~/QinyiZhang/mysql_backup_data/imooc_mysql_worker.sql

# 导出 imooc_mysql 库中 worker 表 id > 5 的数据（字段是数字类型）
# 这也被称为条件导出，如果有多个表的条件相同，则可以同时导出多个表
mysqldump -uroot -proot --databases imooc_mysql --tables worker --where='id > 5' > ~/QinyiZhang/mysql_backup_data/imooc_mysql_worker_id_greater_5.sql

# 导出 imooc_mysql 库中 worker 表 type = B 的数据（字段是字符类型）
mysqldump -uroot -proot --databases imooc_mysql --tables worker --where="type = 'B'" > ~/QinyiZhang/mysql_backup_data/imooc_mysql_worker_type_equal_B.sql

# 只导出 imooc_mysql 库中 worker 表结构，不导出数据
mysqldump -uroot -proot --no-data --databases imooc_mysql --tables worker > ~/QinyiZhang/mysql_backup_data/imooc_mysql_worker_only_table.sql
```

MySQL 在使用 mysqldump 工具时会锁表，这会导致备份发生的时候，其他操作会被阻塞。为了解决阻塞且保证数据一致性的问题，mysqldump 提供了 --single-transaction 选项。使用这个选项会在导出数据之前提交一个 BEGIN SQL 语句，即把备份操作放在一个事务中去执行，不会阻塞任何应用程序且能保证导出时数据库的一致性状态。例如：我们想要使用 --single-transaction 选项再次导出 imooc_mysql 库的所有数据，可以执行：

```
mysqldump --single-transaction -uroot -proot --databases imooc_mysql > ~/QinyiZhang/mysql_backup_data/imooc_mysql_single_transaction.sql
```

最后，mysqldump 还有一个非常重要的选项：-F，它将在导出数据之后生成一个新的 binlog 文件（注意，需要开启 MySQL 的 binlog 配置）。例如，我们可以先去查看当前数据库所有 binlog 日志列表：

```
mysql> show master logs;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mysql-bin.000001 |       804 |
+------------------+-----------+
1 row in set (0.00 sec)
```

接着，使用 mysqldump -F 导出 imooc_mysql 库的所有数据（在一个事务中执行）：

```
mysqldump --single-transaction -uroot -proot --databases imooc_mysql -F > ~/QinyiZhang/mysql_backup_data/imooc_mysql_new_binlog.sql
```

成功之后，再去查看当前数据库所有 binlog 日志列表：

```
mysql> show master logs;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mysql-bin.000001 |       991 |
| mysql-bin.000002 |       154 |
+------------------+-----------+
2 rows in set (0.00 sec)
```

没错，与预期一致，生成了一个新的 binlog 日志文件。到这里呢，就已经把 mysqldump 工具的各种常见用法都介绍了，其实也就是使用各种选项的组合来达到自己的需求。理解了之后，自己动手去试一试吧。

3 恢复数据的步骤

数据已经备份完毕了，下面，我们就可以使用备份的文件做恢复了。假如，我们使用的备份方法是：

```
mysqldump --single-transaction -uroot -proot --databases imooc_mysql > ~/QinyiZhang/mysql_backup_data/imooc_mysql_single_transaction.sql
```

即得到了 imooc_mysql 库的所有数据，库和表的结构。接下来，删除 imooc_mysql 库，模拟数据丢失：

```
mysql> DROP DATABASE imooc_mysql;
Query OK, 2 rows affected (0.07 sec)
```

好的，现在关于 imooc_mysql 库的所有数据都不见了，可以使用备份文件（imooc_mysql_single_transaction.sql）进行恢复了。在 shell 或 cmd 中执行如下命令（也可以在 MySQL 的命令行中使用 SOURCE 导入）：

```
mysql -uroot -proot < ~/QinyiZhang/mysql_backup_data/imooc_mysql_single_transaction.sql
```

执行完之后，你会发现，数据又都回来了。如果要指定恢复某个库下的表数据，还需要指定对应的库。例如，想要恢复 imooc_mysql 库中的 worker 表，可以执行命令：

```
mysql -uroot -proot imooc_mysql < ~/QinyiZhang/mysql_backup_data/imooc_mysql_worker.sql
```

想必你也发现了问题：这只是最简单的一种情况，如果在备份数据之后，又执行了一些操作，还没来得及再次备份，数据不是也会丢失吗 ？是的，此时仅仅使用备份文件是不够的，需要结合 binlog 来操作。这个过程相对来说会比较麻烦，而且仅仅用文字很难解释清楚，所以，继续看示例吧。

这一次，我们不能像之前那样备份数据，需要使用到 -F 选项，备份数据的同时，让 MySQL 生成一个新的 binlog 日志文件。执行命令：

```
mysqldump --single-transaction -uroot -proot --databases imooc_mysql -F > ~/QinyiZhang/mysql_backup_data/imooc_mysql_new_binlog.sql
```

我执行完这条命令之后，MySQL 服务器最新的 binlog 文件是 mysql-bin.000003（每个人的机器上可能都不相同），如下所示：

```
mysql> show master logs;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mysql-bin.000001 |       991 |
| mysql-bin.000002 |      6288 |
| mysql-bin.000003 |       154 |
+------------------+-----------+
```

此时，你需要知道，我们已经备份了 mysql-bin.000003 日志文件之前（即 mysql-bin.000001、mysql-bin.000002）的所有操作。接着，我们向 worker 表中插入两条数据，也就是模拟备份之后再次发生变更的情况：

```
mysql> INSERT INTO `worker` (`type`, `name`, `salary`)
    -> VALUES
    -> ('A', 'x', 1000),
    -> ('B', 'y', 1000);
Query OK, 2 rows affected (0.02 sec)
```

此时，可以知道，这些变化保存到了 mysql-bin.000003 日志文件中（可以再次执行 show master logs 语句，看到这个文件的大小已经发生了变化）。接下来，在 MySQL 命令行中执行 flush logs 命令，再次让服务器产生一个新的 binlog 日志文件。

```
-- 刷写 binlog 日志
mysql> flush logs;

-- 最新的 binlog 日志文件变成了 mysql-bin.000004
mysql> show master logs;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mysql-bin.000001 |       991 |
| mysql-bin.000002 |      6288 |
| mysql-bin.000003 |       516 |
| mysql-bin.000004 |       154 |
+------------------+-----------+
```

继续，删除 imooc_mysql 库，执行命令（此时，删除语句会记录在 mysql-bin.000004 日志文件中）：

```
mysql> DROP DATABASE imooc_mysql;
Query OK, 2 rows affected (0.04 sec)
```

最后，剩下的工作就是恢复数据了。正如我之前所说，恢复数据需要备份数据文件结合 binlog 的方式，所以，会有两个（多个）步骤。第一步，导入备份数据（与之前恢复的方式相同）：

```
mysql -uroot -proot < ~/QinyiZhang/mysql_backup_data/imooc_mysql_new_binlog.sql
```

导入成功之后，可以发现，worker 表缺失了两条最后插入的数据，而这两条数据则保存在 mysql-bin.000003 日志文件中。第二步，恢复 mysql-bin.000003 文件（可能需要 root 权限）：

```
# 使用 mysqlbinlog 工具来做 binlog 日志的恢复
□ ~ sudo mysqlbinlog /usr/local/mysql/data/mysql-bin.000003 | mysql -uroot -proot
mysql: [Warning] Using a password on the command line interface can be insecure.
Password:
```

执行成功之后，可以再去查看 worker 表，最后插入的两条数据也一并找回了。好的，到这里，数据恢复相关的思想、操作也就讲解完毕了。

4 总结

及时备份数据是一个非常好的习惯，这样才能在数据丢失时降低损失。但是，怎样做数据备份与恢复确实不是一件简单的事，需要考虑的问题非常多。那么，仔细分析业务需求，多去理解和实践，尝试各种数据备份、恢复的方法，才能够驾轻就熟，在真正遇到问题时做到临危不乱。

5 问题

在做数据恢复时，插入两条数据之后，我执行了 flush logs 命令，你知道是为什么吗 ？

mysqlbinlog 提供了 --start/stop-position、–start/stop-date 选项，你知道它们怎么使用吗 ？

6 参考资料

《高性能 MySQL（第三版）》

MySQL 官方文档：mysqldump

MySQL 官方文档：The Binary Log

MySQL 官方文档：mysqlbinlog

}

06 常常被忽略的用户与权限

08 搞清事务隔离级别，理解数据并发