第17章_其他数据库日志

讲师：尚硅谷-宋红康（江湖人称：康师傅）

官网：http://www.atguigu.com

我们在讲解数据库事务时，讲过两种日志：`重做日志`、`回滚日志`。

对于线上数据库应用系统，突然遭遇`数据库宕机`怎么办?在这种情况下，`定位宕机的原因`就非常关键。可以查看数据库的`错误日志`。因为日志中记录了数据库运行中的诊断信息，包括了错误、警告和注释等信息。比如:从日志中发现某个连接中的SQL操作发生了死循环，导致内存不足，被系统强行终止了。明确了原因，处理起来也就轻松了，系统很快就恢复了运行。

除了发现错误，日志在数据复制、数据恢复、操作审计，以及确保数据的永久性和一致性等方面，都有着不可替代的作用。

千万不要小看日志。很多看似奇怪的问题，答案往往就藏在日志里。很多情况下，只有通过查看日志才能发现问题的原因，真正解决问题。所以，一定要学会查看日志，养成检查日志的习惯，对提升你的数据库应用开发能力至关重要。

MySQL8.0 官网日志地址

1. MySQL支持的日志

1.1 日志类型

MySQL有不同类型的日志文件，用来存储不同类型的日志，分为`二进制日志`、`错误日志`、`通用查询日志`和`慢查询日志`，这也是常用的4种。MySQL8又新增两种支持的日志：`中继日志`和`数据定义语句日志`。使用这些日志文件，可以查看MySQL内部发生的事情。

这6类日志分别为：

- **慢查询日志：**记录所有执行时间超过long_query_time的所有查询，方便我们对查询进行优化。
- **通用查询日志：**记录所有连接的起始时间和终止时间，以及连接发送给数据库服务器的所有指令，对我们复原操作的实际场景、发现问题，甚至是对数据库操作的审计都有很大的帮助。
- **错误日志：**记录MySQL服务的启动、运行或停止MySQL服务时出现的问题，方便我们了解服务器的状态，从而对服务器进行维护。
- **二进制日志：**记录所有更改数据的语句，可以用于主从服务器之间的数据同步，以及服务器遇到故障时数据的无损失恢复。
- **中继日志：**用于主从服务器架构中，从服务器用来存放主服务器二进制日志内容的一个中间文件。从服务器通过读取中继日志的内容，来同步主服务器上的操作。
- **数据定义语句日志：**记录数据定义语句执行的元数据操作。

除二进制日志外，其他日志都是`文本文件`。默认情况下，所有日志创建于`MySQL数据目录`中。

1.2 日志的弊端

- 日志功能会`降低MySQL数据库的性能`。例如，在查询非常频繁的MySQL数据库系统中，如果开启了通用查询日志和慢查询日志，MySQL数据库会花费很多时间记录日志。
- 日志会`占用大量的磁盘空间`。对于用户量非常大、操作非常频繁的数据库，日志文件需要的存储空间设置比数据库文件需要的存储空间还要大。

2. 慢查询日志(slow query log)

前面章节《第09章_性能分析工具的使用》已经详细讲述。

3. 通用查询日志(general query log)

通用查询日志用来`记录用户的所有操作`，包括启动和关闭MySQL服务、所有用户的连接开始时间和截止时间、发给MySQL数据库服务器的所有SQL指令等。当我们的数据发生异常时，**查看通用查询日志，还原操作时的具体场景**，可以帮助我们准确定位问题。

3.1 问题场景

在电商系统中，购买商品并且使用微信支付完成以后，却发现支付中心的记录并没有新增，此时用户再次使用支付宝支付，就会出现`重复支付`的问题。但是当去数据库中查询数据的时候，会发现只有一条记录存在。那么此时给到的现象就是只有一条支付记录，但是用户却支付了两次。

对系统进行了仔细检查，没有发现数据问题，因为用户编号和订单编号以及第三方流水号都是对的。可是用户确实支付了两次，这个时候，我们想到了检查通用查询日志，看看当天到底发生了什么。

查看之后，发现: 1月1日下午2点，用户使用微信支付完以后，但是由于网络故障，支付中心没有及时收到微信支付的回调通知，导致当时没有写入数据。1月1日下午2点30，用户又使用支付宝支付，此时记录更新到支付中心。1月1日晚上9点，微信的回调通知过来了，但是支付中心已经存在了支付宝的记录，所以只能覆盖记录了。

由于网络的原因导致了重复支付。至于解决问题的方案就很多了，这里省略。

可以看到通用查询日志可以帮助我们了解操作发生的具体时间和操作的细节，对找出异常发生的原因极其关键。

3.2 查看当前状态

```
mysql> SHOW VARIABLES LIKE '%general%';
+------------------+-------------------------------+
| Variable_name    | Value                         |
+------------------+-------------------------------+
 general_log       | OFF                           | #通用查询日志处于关闭状态
| general_log_file | /var/lib/mysql/atguigu01.log | #通用查询日志文件的名称是
atguigu01.log
+------------------+-------------------------------+
2 rows in set (0.03 sec)
```

说明1：系统变量general_log的值是OFF，即通用查询日志处于关闭状态。在MySQL中，这个参数的`默认值是关闭的`。因为一旦开启记录通用查询日志，MySQL 会记录所有的连接起止和相关的SQL操作，这样会消耗系统资源并且占用磁盘空间。我们可以通过手动修改变量的值，在`要的时候开启日志`。

说明2:通用查询日志文件的名称是主机.log(hadoop102.log)。存储路径是`/var/lib/mysql/`，默认也是数据路径。这样我们就知道在哪里可以查看通用查询日志的内容了

3.3 启动日志

方式1：永久性方式

修改`my.cnf`或者`my.ini`配置文件来设置。在[mysqld]组下加入log选项，并重启MySQL服务。格式如下：

```
[mysqld]
general_log=ON
general_log_file=[path[filename]]   # 日志文件所在目录路径，filename为日志文件名
```

如果不指定目录和文件名，通用查询日志将默认存储在MySQL数据目录中的hostname.log文件中，hostname表示主机名。

方式2：临时性方式

使用SET语句停止MySQL通用查询日志功能:

```
SET GLOBAL general_log=on;  # 开启通用查询日志
```

```
SET GLOBAL general_log_file='path/filename';    # 设置日志文件保存位置
```

对应的，关闭操作SQL命令如下：

```
SET GLOBAL general_log=off;     # 关闭通用查询日志
```

查看设置后情况：

```
SHOW VARIABLES LIKE 'general_log%';
```

3.4 查看日志

通用查询日志是以`文本文件`的形式存储在文件系统中的，可以使用`文本编辑器`直接打开日志文件。每台MySQL服务器的通用查询日志内容是不同的。

- 在Windows操作系统中，使用文本文件查看器；
- 在Linux系统中，可以使用vi工具或者gedit工具查看；
- 在MacOSX系统中，可以使用文本文件查看器或者vi等工具查看。

从`SHOW VARIABLES LIKE 'general_log%';`结果中可以看到通用查询日志的位置。

通过通用查询日志，可以了解用户对MySQL进行的操作。比如，MySQL启动信息和用户root连接服务器和执行查询表的记录。

```
/usr/sbin/mysqld, Version: 8.0.26 (MySQL Community Server -GPL). started with:
Tcp port: 3306 Unix socket: /var/lib/mysql/mysql.sock
Time                 Id Command    Argument
2022-01-04T07:44:58.052890Z     10 Query        SHOW VARIABLES LIKE '%general%'
2022-01-04T07:45:15.666672Z     10 Query        SHOW VARIABLES LIKE 'general_log%'
2022-01-04T07:45:28.970765Z     10 Query        select  * from student
2022-01-04T07:47:38.706804Z     11 Connect      root@localhost  on  using Socket
2022-01-04T07:47:38.707435Z     11 Query        select @@version_comment limit 1
2022-01-04T07:48:21.384886Z     12 Connect      root@172.16.210.1  on  using TCP/IP
```

```
2022-01-04T07:48:21.385253Z	   12 Query	SET NAMES utf8
2022-01-04T07:48:21.385640Z	   12 Query	USE `atguigu12`
2022-01-04T07:48:21.386179Z	   12 Query	SHOW FULL TABLES WHERE Table_Type !=
'VIEW'
2022-01-04T07:48:23.901778Z	   13 Connect	root@172.16.210.1 on using TCP/IP
2022-01-04T07:48:23.902128Z	   13 Query	SET NAMES utf8
2022-01-04T07:48:23.905179Z	   13 Query	USE `atguigu`
2022-01-04T07:48:23.905825Z	   13 Query	SHOW FULL TABLES WHERE Table_Type != 'VIEW'
2022-01-04T07:48:32.163833Z	   14 Connect	root@172.16.210.1 on using TCP/IP
2022-01-04T07:48:32.164451Z	   14 Query	SET NAMES utf8
2022-01-04T07:48:32.164840Z	   14 Query	USE `atguigu`
2022-01-04T07:48:40.006687Z	   14 Query	select * from account
```

在通用查询日志里面，我们可以清楚地看到，什么时候开启了新的客户端登陆数据库，登录之后做了什么SQL操作，针对的是哪个数据表等信息。

3.5 停止日志

方式1：永久性方式

修改`my.cnf`或者`my.ini`文件，把[mysqld]组下的`general_log`值设置为`OFF`或者把general_log一项注释掉。修改保存后，再重启`MySQL服务`，即可生效。

举例1：

```
[mysql]
general_log=OFF
```

举例2：

```
[mysqld]
#general_log=ON
```

方式2：临时性方式

使用SET语句停止MySQL通用查询日志功能：

```
SET GLOBAL general_log=off;
```

查询通用日志功能：

```
SHOW VARIABLES LIKE 'general_log%';
```

3.6 删除\刷新日志

如果数据的使用非常频繁，那么通用查询日志会占用服务器非常大的磁盘空间。数据管理员可以删除很长时间之前的查询日志，以保证MySQL服务器上的硬盘空间。

手动删除文件

```
SHOW VARIABLES LIKE 'general_log%';
```

可以看出，通用查询日志的目录默认为MySQL数据目录。在该目录下手动删除通用查询日志atguigu01.log。

使用如下命令重新生成查询日志文件，具体命令如下。刷新MySQL数据目录，发现创建了新的日志文件。前提一定要开启通用日志。

```
mysqladmin -uroot -p flush-logs
```

如果希望备份旧的通用查询日志，就必须先将旧的日志文件复制出来或者改名，然后执行上面的mysqladmin命令。正确流程如下：

```
cd mysql-data-directory #输入自己的通用日志文件所在目录
mv mysql.general.log mysql.general.log.old #指名就的文件名 以及新的文件名
mysqladmin -uroot -p flush-logs
```

4. 错误日志(error log)

错误日志记录了MySQL服务器启动、停止运行的时间，以及系统启动、运行和停止过程中的诊断信息，包括`错误`、`警告`和`提示`等。

通过错误日志可以查看系统的运行状态，便于即时发现故障、修复故障。如果MysQL服务`出现异常`，错误日志是发现问题、解决故障的`首选`。

4.1 启动日志

在MySQL数据库中，错误日志功能是`默认开启`的。而且，错误日志`无法被禁止`。

默认情况下，错误日志存储在MySQL数据库的数据文件夹下，名称默认为`mysqld.log`（Linux系统）或`hostname.err`（mac系统）。如果需要制定文件名，则需要在my.cnf或者my.ini中做如下配置：

```
[mysqld]
log-error=[path/[filename]] #path为日志文件所在的目录路径，filename为日志文件名
```

修改配置项后，需要重启MySQL服务以生效。

4.2 查看日志

MySQL错误日志是以文本文件形式存储的，可以使用文本编辑器直接查看。

查询错误日志的存储路径：

```
mysql> SHOW VARIABLES LIKE 'log_err%';
+----------------------------+----------------------------------------+
| Variable_name              | Value                                  |
+----------------------------+----------------------------------------+
| log_error                  | /var/log/mysqld.log                    |
| log_error_services         | log_filter_internal; log_sink_internal |
| log_error_suppression_list |                                        |
| log_error_verbosity        | 2                                      |
+----------------------------+----------------------------------------+
4 rows in set (0.01 sec)
```

执行结果中可以看到错误日志文件是mysqld.log，位于MySQL默认的数据目录下。

下面我们查看一下错误日志的内容。

```
2024-06-22T11:35:12.106692Z 0 [System] [MY-015017] [Server] MySQL Server Initialization - start.
2024-06-22T11:35:12.108513Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld (mysqld 8.4.0) initializing of server in progress as process 60962
2024-06-22T11:35:12.116132Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2024-06-22T11:35:12.342871Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2024-06-22T11:35:12.965572Z 6 [Note] [MY-010454] [Server] A temporary password is generated for root@localhost: sfuGuerEj7(8
2024-06-22T11:35:15.002191Z 0 [System] [MY-015018] [Server] MySQL Server Initialization - end.
2024-06-22T11:38:31.667788Z 0 [System] [MY-015015] [Server] MySQL Server - start.
2024-06-22T11:38:31.929101Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.4.0) starting as process 61076
2024-06-22T11:38:31.939919Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2024-06-22T11:38:32.220536Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2024-06-22T11:38:32.474523Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
2024-06-22T11:38:32.474637Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connections are now supported for this channel.
2024-06-22T11:38:32.514692Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.4.0'  socket: '/var/lib/mysql/mysql.sock'  port: 3306  MySQL Community Server - GPL.
2024-06-22T11:38:32.768225Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqlx.sock
2024-06-24T11:59:40.802225Z 0 [System] [MY-015015] [Server] MySQL Server - start.
2024-06-24T11:59:41.760595Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.4.0) starting as process 2957
2024-06-24T11:59:42.059737Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2024-06-24T11:59:43.798048Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2024-06-24T11:59:44.133084Z 0 [System] [MY-010229] [Server] Starting XA crash recovery...
2024-06-24T11:59:44.204113Z 0 [System] [MY-010232] [Server] XA crash recovery finished.
2024-06-24T11:59:44.296448Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
2024-06-24T11:59:44.296494Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connections are now supported for this channel.
2024-06-24T11:59:44.329007Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.4.0'  socket: '/var/lib/mysql/mysql.sock'  port: 3306  MySQL Community Server - GPL.
```

```
2024-06-24T11:59:44.583113Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqlx.sock
2024-06-29T07:01:31.759818Z 0 [System] [MY-013172] [Server] Received SHUTDOWN from user <via user signal>. Shutting down mysqld (Version: 8.4.0).
2024-06-29T07:01:32.670072Z 0 [System] [MY-010910] [Server] /usr/sbin/mysqld: Shutdown complete (mysqld 8.4.0)  MySQL Community Server - GPL.
2024-06-29T07:01:32.670092Z 0 [System] [MY-015016] [Server] MySQL Server - end.
2024-06-29T07:02:02.664560Z 0 [System] [MY-015015] [Server] MySQL Server - start.
2024-06-29T07:02:03.140000Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.4.0) starting as process 1796
2024-06-29T07:02:03.305380Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2024-06-29T07:02:04.418471Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2024-06-29T07:02:04.748141Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
2024-06-29T07:02:04.748181Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connections are now supported for this channel.
2024-06-29T07:02:04.776579Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.4.0'  socket: '/var/lib/mysql/mysql.sock'  port: 3306  MySQL Community Server - GPL.
//..
```

可以看到，错误日志文件中记录了服务器启动的时间，以及存储引擎InnoDB启动和停止等，我们在做初始化时候生成的数据库初始密码也是记录在error.log中。

4.3 删除\刷新日志

对于很久以前的错误日志，数据库管理员查看这些错误日志的可能性不大，可以将这些错误日志删除，以保证MySQL服务器上的`硬盘空间`。MySQL的错误日志是以文本文件的形式存储在文件系统中的，可以`直接删除`。

- 第1步(方式1)：删除操作

```
rm -f /var/lib/mysql/mysql.log
```

在运行状态下删除错误日志文件后，MySQL并不会自动创建日志文件

- 第1步(方式2)：重命名文件

```
mv /var/lib/mysql/mysql.log /var/lib/mysql/mysql.log.old
```

- 第2步：重建日志

```
mysqladmin -uroot -p flush-logs
```

可能报错

```
[root@atguigu01 log]# mysqladmin -uroot -p flush-logs
Enter password:
mysqladmin: refresh failed; error: 'Could not open file '/var/log/mysqld.log'
for error logging.'
```

官网提示：

> **Note**
>
> For the server to recreate a given log file after you have renamed the file externally, the file location must be writable by the server. This may not always be the case. For example, on Linux, the server might write the error log as `/var/log/mysqld.log`, where `/var/log` is owned by `root` and not writable by **mysqld**. In this case, log-flushing operations fail to create a new log file.
>
> To handle this situation, you must manually create the new log file with the proper ownership after renaming the original log file. For example, execute these commands as `root`:
>
> ```
> mv /var/log/mysqld.log /var/log/mysqld.log.old
> install -omysql -gmysql -m0644 /dev/null /var/log/mysqld.log
> ```

补充操作：

```
install -omysql -gmysql -m0644 /dev/null /var/log/mysqld.log
```

flush-logs指令操作:

- MySQL 5.5.7以前的版本，flush-logs将错误日志文件重命名为filename.err_old，并创建新的日志文件。
- 从MySQL 5.5.7开始，flush-logs只是重新打开日志文件，并不做日志备份和创建的操作。
- 如果日志文件不存在，MySQL启动或者执行flush-logs时会自动创建新的日志文件。重新创建错误日志，大小为0字节。

4.4 MySQL8.0新特性

MySQL8.0里对错误日志的改进。MySQL8.0的错误日志可以理解为一个全新的日志，在这个版本里，接受了来自社区的广泛批评意见，在这些意见和建议的基础上生成了新的日志。

下面这些是来自社区的意见:

- 默认情况下内容过于冗长
- 遗漏了有用的信息
- 难以过滤某些信息
- 没有标识错误信息的子系统源
- 没有错误代码，解析消息需要识别错误
- 引导消息可能会丢失
- 固定格式

针对这些意见，MySQL做了如下改变:

- 采用组件架构，通过不同的组件执行日志的写入和过滤功能
- 写入错误日志的全部信息都具有唯一的错误代码从10000开始

- 增加了一个新的消息分类《system》用于在错误日志中始终可见的非错误但服务器状态更改事件的消息。增
- 加了额外的附加信息，例如关机时的版本信息，谁发起的关机等等
- 两种过滤方式，Internal和Dragnet
- 三种写入形式，经典、JSON和syseventlog

> 小结:
>
> 通常情况下，管理员不需要查看错误日志。但是，MySQL服务器发生异常时，管理员可以从错误日志中找到发生异常的时间、原因，然后根据这些信息来解决异常。

5. 二进制日志(bin log)

binlog可以说是MySQL中比较`重要`的日志了，在日常开发及运维过程中，经常会遇到。

binlog即binarylog，二进制日志文件，也叫作变更日志（update log）。它记录了数据库所有执行的`DDL`和`DML`等数据库更新事件的语句，但是不包含没有修改任何数据的语句（如数据查询语句select、show等）。

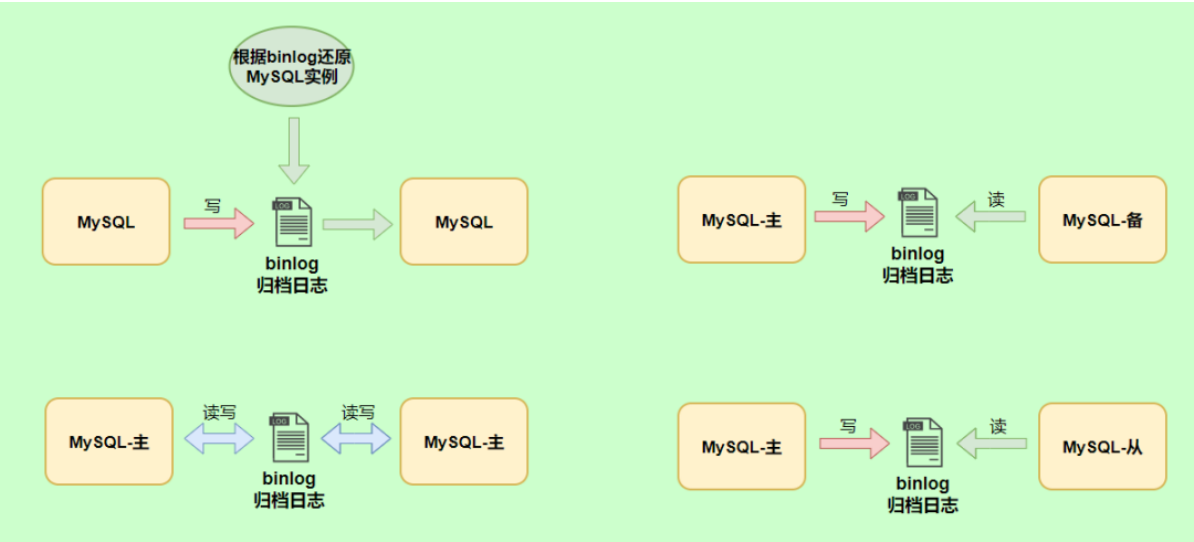
它以`事件形式`记录并保存在`二进制文件`中。通过这些信息，我们可以再现数据更新操作的全过程。

> 如果想要记录所有语句（例如，为了识别有问题的查询），需要使用通用查询日志。

binlog主要应用场景：

- 一是用于`数据恢复`，如果MySQL数据库意外停止，可以通过二进制日志文件来查看用户执行了哪些操作，对数据库服务器文件做了哪些修改，然后根据二进制日志文件中的记录来恢复数据库服务器。
- 二是用于`数据复制`，由于日志的延续性和时效性，master把它的二进制日志传递给slaves来达到master-slave数据一致的目的。

可以说MySQL数据库的**数据备份、主备、主主、主从**都离不开binlog，需要依靠binlog来同步数据，保证数据一致性。



5.1 查看默认情况

查看记录二进制日志是否开启：在MySQL8中默认情况下，二进制文件是开启的。

```
mysql> show variables like '%log_bin%';
+---------------------------------+-----------------------------+
| Variable_name                   | Value                       |
+---------------------------------+-----------------------------+
| log_bin                         | ON                          |
| log_bin_basename                | /var/lib/mysql/binlog       |
| log_bin_index                   | /var/lib/mysql/binlog.index |
| log_bin_trust_function_creators | OFF                         |
| log_bin_use_v1_row_events       | OFF                         |
| sql_log_bin                     | ON                          |
+---------------------------------+-----------------------------+
6 rows in set (0.00 sec)
```

`log_bin_basename`：是binlog日志的基本文件名，后面会追加标识来表示每一个文件

`log_bin_index`:是binlog文件的索引文件，这个文件管理了所有的binlog文件的目录

`log_bin_trust_function_creators`：限制存储过程，前面我们已经讲过了，这是因为二进制日志的一个重要功能是用于主从复制，而存储函数有可能导致主从的数据不一致。所以当开启二进制日志后，需要限制存储函数的创建、修改、调用

`log_bin_use_v1_row_events`：此只读系统变量已弃用。ON表示使用版本1二进制日志行，OFF表示使用版本2二进制日志行(MysQL 5.6的默认值为2)。

每次服务重启，都会新创建一个binlog：

```
-rw-r-----. 1 mysql mysql 139017509 12月 22 09:38 binlog.000012
-rw-r-----. 1 mysql mysql       179 12月 22 18:41 binlog.000013
-rw-r-----. 1 mysql mysql 508001388 12月 22 20:16 binlog.000014
-rw-r-----. 1 mysql mysql       474 12月 23 15:15 binlog.000015
-rw-r-----. 1 mysql mysql   3584595 12月 24 14:54 binlog.000016
-rw-r-----. 1 mysql mysql       156 12月 25 09:29 binlog.000017
-rw-r-----. 1 mysql mysql  65285264 12月 26 18:24 binlog.000018
-rw-r-----. 1 mysql mysql       605 12月 28 11:24 binlog.000019
-rw-r-----. 1 mysql mysql 315060323 12月 29 12:09 binlog.000020
-rw-r-----. 1 mysql mysql      4295 12月 30 11:55 binlog.000021
-rw-r-----. 1 mysql mysql       179 12月 30 14:20 binlog.000022
-rw-r-----. 1 mysql mysql       156 12月 30 14:26 binlog.000023
-rw-r-----. 1 mysql mysql       156 12月 30 14:52 binlog.000024
-rw-r-----. 1 mysql mysql       156 12月 30 15:13 binlog.000025
-rw-r-----. 1 mysql mysql       156 12月 30 16:06 binlog.000026
-rw-r-----. 1 mysql mysql       156 12月 30 16:27 binlog.000027
-rw-r-----. 1 mysql mysql    388723 12月 31 19:02 binlog.000028
```

5.2 日志参数设置

方式1：永久性方式

修改MySQL的`my.cnf`或`my.ini`文件可以设置二进制日志的相关参数：

```
[mysqld]
#启用二进制日志
log-bin=atguigu-bin
binlog_expire_logs_seconds=600
max_binlog_size=100M
```

> 提示:
>
> 1. log-bin=mysql-bin #打开日志(主机需要打开)，这个mysql-bin也可以自定义，这里也可以加上路径，如: /home/www/mysql_bin_log/mysql-bin
> 2. binlog_expire_logs_seconds:此参数控制二进制日志文件保留的时长，单位是秒，默认2592000 3(天-- 14400 4小时; 86400 1天; 259200 3天;
> 3. max_binlog_size:控制单个二进制日志大小，当前日志文件大小超过此变量时，执行切换动作。此参数的`最大和默认值是1GB`，该设置并不能严格控制Binlog的大小，尤其是Binlog比较靠近最大值而又遇到一个比较大事务时，为了保证事务的完整性，可能不做切换日志的动作，只能将该事务的所有SQL都记录进当前日志，直到事务结束。一般情况下可采取默认值

重新启动MySQL服务，查询二进制日志的信息，执行结果:

```
mysql> show variables like '%log_bin%';
+---------------------------------+-----------------------------+
| Variable_name                   | Value                       |
+---------------------------------+-----------------------------+
| log_bin                         | ON                          |
| log_bin_basename                | /var/lib/mysql/binlog       |
| log_bin_index                   | /var/lib/mysql/binlog.index |
| log_bin_trust_function_creators | OFF                         |
| log_bin_use_v1_row_events       | OFF                         |
| sql_log_bin                     | ON                          |
+---------------------------------+-----------------------------+
6 rows in set (0.01 sec)
```

设置带文件夹的bin-log日志存放目录

如果想改变日志文件的目录和名称，可以对my.cnf或my.ini中的log_bin参数修改如下:

```
[mysqld]
log-bin="/var/lib/mysql/binlog/atguigu-bin"
```

注意：新建的文件夹需要使用mysql用户，使用下面的命令即可。

```
chown -R -v mysql:mysql binlog
```

重启MySQL服务之后，新的二进制日志文件将出现在/var/lib/mysql/binlog/文件夹下面:

```
mysql>  show variables like '%log_bin%';
+---------------------------------+-----------------------------------+
| Variable_name                   | Value                             |
+---------------------------------+-----------------------------------+
| log_bin                         | ON                                |
| log_bin_basename                | /var/lib/mysql/atguigu-bin        |
| log_bin_index                   | /var/lib/mysql/atguigu-bin.index  |
| log_bin_trust_function_creators | OFF                               |
| log_bin_use_v1_row_events       | OFF                               |
| sql_log_bin                     | ON                                |
+---------------------------------+-----------------------------------+
6 rows in set (0.00 sec)
```

> 提示:

> `数据库文件最好不要与日志文件放在同一个磁盘上！`这样，当数据库文件所在的磁盘发生故障时，可以使用日志文件恢复数据。

方式2：临时性方式

如果不希望通过修改配置文件并重启的方式设置二进制日志的话，还可以使用如下指令，需要注意的是在mysql8中只有`会话级别`的设置，没有了global级别的设置。

```
# global级别
mysql> set global sql_log_bin=0;
ERROR 1228 (HY000): Variable 'sql_log_bin' is a SESSION variable and can`t be used with SET GLOBAL

# session级别
mysql> SET sql_log_bin=0;
Query OK, 0 rows affected (0.01秒)
```

5.3 查看日志

当MySQL创建二进制日志文件时，先创建一个以“filename”为名称、以“.index”为后缀的文件，再创建一个以“filename”为名称、以“.000001”为后缀的文件。

MySQL服务`重新启动一次`，以“.000001”为后缀的文件就会增加一个，并且后缀名按1递增。即日志文件的个数与MySQL服务启动的次数相同；如果日志长度超过了`max_binlog_size`的上限（默认是1GB），就会创建一个新的日志文件。

查看当前的二进制日志文件列表及大小。指令如下：

```
mysql> SHOW BINARY LOGS;
+--------------------+-----------+-----------+
| Log_name           | File_size | Encrypted |
+--------------------+-----------+-----------+
| atguigu-bin.000001 | 156       | No        |
+--------------------+-----------+-----------+
1 行于数据集 (0.02秒)
```

所有对数据库的修改都会记录在binglog中。但binlog是二进制文件，无法直接查看，想要更直观的观测它就要借助`mysqlbinlog`命令工具了。指令如下:在查看执行，先执行两条SQL语句，如下

```
insert into student(id,name,class) values(18,'Jerry','四班');
update student set name = 'Tom' where id = 15;
```

开始查看binlog

```
[root@hadoop102 mysql]# mysqlbinlog "/var/lib/mysql/atguigu-bin.000002"
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=1*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#230722 16:14:20 server id 1  end_log_pos 125 CRC32 0xfbe10f64  Start: binlog v 4, server v 8.0.25 created 230722 16:14:20 at startup
# Warning: this binlog is either in use or was not closed properly.
ROLLBACK/*!*/;
BINLOG '
```

```
'/*!*/;
# at 547
#230722 16:19:02 server id 1  end_log_pos 637 CRC32 0x0e4d6052  Query
thread_id=8 exec_time=0 error_code=0
SET TIMESTAMP=1690013942/*!*/;
BEGIN
//......
# at 776
#230722 16:19:02 server id 1  end_log_pos 807 CRC32 0x6f80cb79  Xid = 15
COMMIT/*!*/;
SET @@SESSION.GTID_NEXT= 'AUTOMATIC' /* added by mysqlbinlog */ /*!*/;
DELIMITER ;
# End of log file
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=0*/;
```

执行结果可以看到，这是一个简单的日志文件，日志中记录了用户的一些操作，这里并没有出现具体的SQL语句，这是因为binlog关键字后面的内容是经过编码后的`二进制日志`。

这里一个update语句包含如下事件

- Query事件负责开始一个事务(BEGIN)
- Table_map事件负责映射需要的表.
- Update_rows事件负责写入数据
- Xid事件负责结束事务

下面命令将行事件以`伪SQL的形式`表现出来

```
mysqlbinlog -v "/var/lib/mysql/binlog/atguigu-bin.000002"
#220105 9:16:37 server id 1 end_log_pos 324 CRC32 0x6b31978b Query
thread_id=10 exec_time=0 error_code=0
SET TIMESTAMP=1641345397/*!*/;
SET @@session.pseudo_thread_id=10/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=0,
@@session.unique_checks=1, @@session.autocommit=1/*!*/;
SET @@session.sql_mode=1168113696/*!*/;
SET @@session.auto_increment_increment=1,
@@session.auto_increment_offset=1/*!*/; /*!\C utf8mb3 *//*!*/;
SET
@@session.character_set_client=33,@@session.collation_connection=33,@@sessio
n.collatio n_server=255/*!*/;
SET @@session.lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
/*!80011 SET @@session.default_collation_for_utf8mb4=255*//*!*/;
BEGIN
/*!*/;
# at 324
#220105 9:16:37 server id 1 end_log_pos 391 CRC32 0x74f89890 Table_map:
`atguigu14`.`student` mapped to number 85
# at 391
#220105 9:16:37 server id 1 end_log_pos 470 CRC32 0xc9920491 Update_rows:
table id 85 flags: STMT_END_F
```

```
BINLOG
'dfHUYRMBAAAAQwAAAIcBAAAAAFUAAAAAAAEACWF0Z3VpZ3UxNAAHc3R1ZGVudAADAw8PBDwAHgAG
AQEAAgEhkJj4dA==dfHUYR8BAAAATwAAANYBAAAAAFUAAAAAAAEAAgAD//8AAQAAAAblvKDkuIkG
5LiA54+tAAEAAAAL 5byg5LiJX2JhY2sG5LiA54+tkQSSyQ==''/*!*/;
### UPDATE `atguigu`.`student`
### WHERE
### @1=1
### @2='张三'
### @3='一班'
### SET
### @1=1
### @2='张三 _back'
### @3='一班'
# at 470
#220105 9:16:37 server id 1 end_log_pos 501 CRC32 0xca01d30f Xid = 15
COMMIT/*!*/;
```

前面的命令同时显示binlog格式的语句，使用如下命令不显示它

```
mysqlbinlog -v --base64-output=DECODE-ROWS "/var/lib/mysql/binlog/atguigu-
bin.000002"
#220105 9:16:37 server id 1 end_log_pos 324 CRC32 0x6b31978b Query
thread_id=10 exec_time=0 error_code=0
SET TIMESTAMP=1641345397/*!*/;
SET @@session.pseudo_thread_id=10/*!*/;
SET @@session.foreign_key_checks=1,
@@session.sql_auto_is_null=0,@@session.unique_checks=1,
@@session.autocommit=1/*!*/;
SET @@session.sql_mode=1168113696/*!*/;
SET @@session.auto_increment_increment=1,
@@session.auto_increment_offset=1/*!*/;
/*!\C utf8mb3 *//*!*/;
SET
@@session.character_set_client=33,@@session.collation_connection=33,@@sessio
n.collation_server=255/*!*/;
SET @@session.lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
/*!80011 SET @@session.default_collation_for_utf8mb4=255*//*!*/;
BEGIN
/*!*/;
# at 324
#220105 9:16:37 server id 1 end_log_pos 391 CRC32 0x74f89890 Table_map:
`atguigu14`.`student` mapped to number 85
# at 391
#220105 9:16:37 server id 1 end_log_pos 470 CRC32 0xc9920491 Update_rows:
table id 85 flags: STMT_END_F
### UPDATE `atguigu14`.`student`
### WHERE
### @1=1
### @2='张三 '
### @3='一班 '
### SET
### @1=1
### @2='张三 _back'
```

```
### @3='一班 '
# at 470
#220105 9:16:37 server id 1 end_log_pos 501 CRC32 0xca01d30f Xid = 15
```

关于mysqlbinlog工具的使用技巧还有很多，例如只解析对某个库的操作或者某个时间段内的操作等。简单分享几个常用的语句，更多操作可以参考官方文档。

```
#可查看参数帮助
mysqlbinlog --no-defaults --help

#查看最后 100行
mysqlbinlog --no-defaults --base64-output=decode-rows -vv atguigu-bin.000002 |tail -100

#根据 position查找
mysqlbinlog --no-defaults --base64-output=decode-rows -vv atguigu-bin.000002 |grep -A 20 '4939002'
```

上面这种办法读取出binlog日志的全文内容比较多，不容易分辨查看到pos点信息，下面介绍一种更为方便的查询命令：

```
mysql> show binlog events [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count];
```

- `IN 'log_name'`：指定要查询的binlog文件名（不指定就是第一个binlog文件）
- `FROM pos`：指定从哪个pos起始点开始查起（不指定就是从整个文件首个pos点开始算）
- `LIMIT [offset]`：偏移量(不指定就是0)
- `row_count`:查询总条数（不指定就是所有行）

```
mysql> show binlog events in 'atguigu-bin.000002';
+-------------------+-----+----------------+-----------+-------------+-----------------------------------+
| Log_name          | Pos | Event_type     | Server_id | End_log_pos | Info                              |
+-------------------+-----+----------------+-----------+-------------+-----------------------------------+
| atguigu-bin.000002 |   4 | Format_desc    |         1 |         125 | Server ver: 8.0.25, Binlog ver: 4 |
| atguigu-bin.000002 | 125 | Previous_gtids |         1 |         156 |                                   |
| atguigu-bin.000002 | 156 | Anonymous_Gtid |         1 |         235 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS' |
| atguigu-bin.000002 | 235 | Query          |         1 |         316 | BEGIN                             |
| atguigu-bin.000002 | 316 | Table_map      |         1 |         384 | table_id: 92 (atguigudb3.student) |
| atguigu-bin.000002 | 384 | Write_rows     |         1 |         437 | table_id: 92 flags: STMT_END_F    |
| atguigu-bin.000002 | 437 | Xid            |         1 |         468 | COMMIT /* xid=14 */               |
| atguigu-bin.000002 | 468 | Anonymous_Gtid |         1 |         547 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS' |
```

```
| atguigu-bin.000002 | 547 | Query          |         1 |         637 |
BEGIN                                  |
| atguigu-bin.000002 | 637 | Table_map      |         1 |         705 |
table_id: 92 (atguigudb3.student)      |
| atguigu-bin.000002 | 705 | Update_rows    |         1 |         776 |
table_id: 92 flags: STMT_END_F         |
| atguigu-bin.000002 | 776 | Xid            |         1 |         807 |
COMMIT /* xid=15 */                    |
+--------------------+-----+----------------+-----------+-------------+-----
----------------------------------+
```

上面这条语句可以将指定的binlog日志文件，分成有效事件行的方式返回，并可使用limit指定pos点的起始偏移，查询条数。其它举例：

```
#a、查询第一个最早的binlog日志:
show binlog events\G ;

#b、指定查询mysql-bin.000002这个文件
show binlog events in 'atguigu-bin.000002'\G;

#c、指定查询mysql-bin.000002这个文件，从pos点:391开始查起:
show binlog events in 'atguigu-bin . 088882' from 391\G;

#d、指定查询mysql-bin.000002这个文件，从pos点:391开始查起，查询5条（即5条语句
show binlog events in 'atguigu-bin.080802' from 391 limit 5\G;

#e、指定查询mysql-bin.000002这个文件，从pos点:391开始查起，偏移2行（即中间跳过2个）查询
5条（即5条语句)。
show binlog events in 'atguigu-bin.088002 ' from 391 limit 2,5\G;
```

上面我们讲了这么多都是基于binlog的默认格式，binlog格式查看

```
mysql> show variables like 'binlog_format';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | ROW   |
+---------------+-------+
1 行于数据集 (0.02 秒)
```

除此之外，binlog还有2种格式，分别是`Statement`和`Mixed`

- **Statement**

  每一条会修改数据的sql都会记录在binlog中。

  优点：不需要记录每一行的变化，减少了binlog日志量，节约了IO，提高性能。

- **Row**

  5.1.5版本的MySQL才开始支持rowlevel的复制，它不记录sql语句上下文相关信息，仅保存哪条记录被修改。

  优点：rowlevel的日志内容会非常清楚的记录下每一行数据修改的细节。而且不会出现某些特定情况下的存储过程，或function，以及trigger的调用和触发无法被正确复制的问题。

- **Mixed**

从5.1.8版本开始，MySQL提供了Mixed格式，实际上就是Statement与Row的结合。

详细情况，下章讲解。

5.4 使用日志恢复数据

如果MySQL服务器启用了二进制日志，在数据库出现意外丢失数据时，可以使用MySQLbinlog工具从指定的时间点开始（例如，最后一次备份）直到现在或另一个指定的时间点的日志中恢复数据。

mysqlbinlog恢复数据的语法如下：

```
mysqlbinlog [option] filename|mysql -uuser -ppass;
```

这个命令可以这样理解：使用mysqlbinlog命令来读取filename中的内容，然后使用mysql命令将这些内容恢复到数据库中。

- `filename`：是日志文件名。
- `option`：可选项，比较重要的两对option参数是--start-date、--stop-date和--start-position、--stop-position。
  - `--start-date和 --stop-date`：可以指定恢复数据库的起始时间点和结束时间点。
  - `--start-position和--stop-position`：可以指定恢复数据的开始位置和结束位置。

> 注意：使用mysqlbinlog命令进行恢复操作时，必须是编号小的先恢复，例如atguigu-bin.000001必
> 须在atguigu-bin.000002之前恢复。

案例

现在对student表有以下操作

```
mysql> use atguigudb3;
Database changed
# 查询数据
mysql> select * from student;
+----+---------+--------+
| id | name    | class  |
+----+---------+--------+
|  1 | 张三2   | 一班   |
|  3 | 李四1   | 一班   |
|  6 | Jane    | 一班   |
|  8 | 王五    | 二班   |
| 15 | Tom     | 二班   |
| 18 | Jerry   | 四班   |
| 20 | 钱七    | 三班   |
+----+---------+--------+
7 rows in set (0.00 sec)

#插入数据
mysql> insert into student(id,name,class) values(21,'aaa','No.1');
Query OK, 1 row affected (0.00 sec)

mysql> insert into student(id,name,class) values(22,'aaa','No.1');
Query OK, 1 row affected (0.00 sec)

```

```
mysql> insert into student(id,name,class) values(23,'aaa','No.1');
Query OK, 1 row affected (0.00 sec)

mysql> select * from student;
+----+---------+--------+
| id | name    | class  |
+----+---------+--------+
|  1 | 张三2   | 一班   |
|  3 | 李四1   | 一班   |
|  6 | Jane    | 一班   |
|  8 | 王五    | 二班   |
| 15 | Tom     | 二班   |
| 18 | Jerry   | 四班   |
| 20 | 钱七    | 三班   |
| 21 | aaa     | No1    |
| 22 | aaa     | No1    |
| 23 | aaa     | No1    |
+----+---------+--------+
10 rows in set (0.01 sec)
# 删除数据
mysql> delete from student where id = 21;
Query OK, 1 row affected (0.01 sec)
# 修改数据
mysql> update student set name='bbb' where id=22;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>  select * from student;
+----+---------+--------+
| id | name    | class  |
+----+---------+--------+
|  1 | 张三2   | 一班   |
|  3 | 李四1   | 一班   |
|  6 | Jane    | 一班   |
|  8 | 王五    | 二班   |
| 15 | Tom     | 二班   |
| 18 | Jerry   | 四班   |
| 20 | 钱七    | 三班   |
| 22 | bbb     | No1    |
| 23 | aaa     | No1    |
+----+---------+--------+
9 rows in set (0.00 sec)
```

上面是对student表做的插入，更新和删除操作。现在不小心删除了所有新增的数据，现在目标就是恢复后面的新增，更新和删除的数据。当前表中数据如下。

误操作

```
mysql> delete from student where id > 21;
Query OK, 2 rows affected (0.01 sec)

mysql>  select * from student;
+----+---------+--------+
| id | name    | class  |
+----+---------+--------+
```

```
|  1 | 张三2    | 一班   |
|  3 | 李四1    | 一班   |
|  6 | Jane     | 一班   |
|  8 | 王五     | 二班   |
| 15 | Tom      | 二班   |
| 18 | Jerry    | 四班   |
| 20 | 钱七     | 三班   |
+----+---------+--------+
7 rows in set (0.00 sec)
```

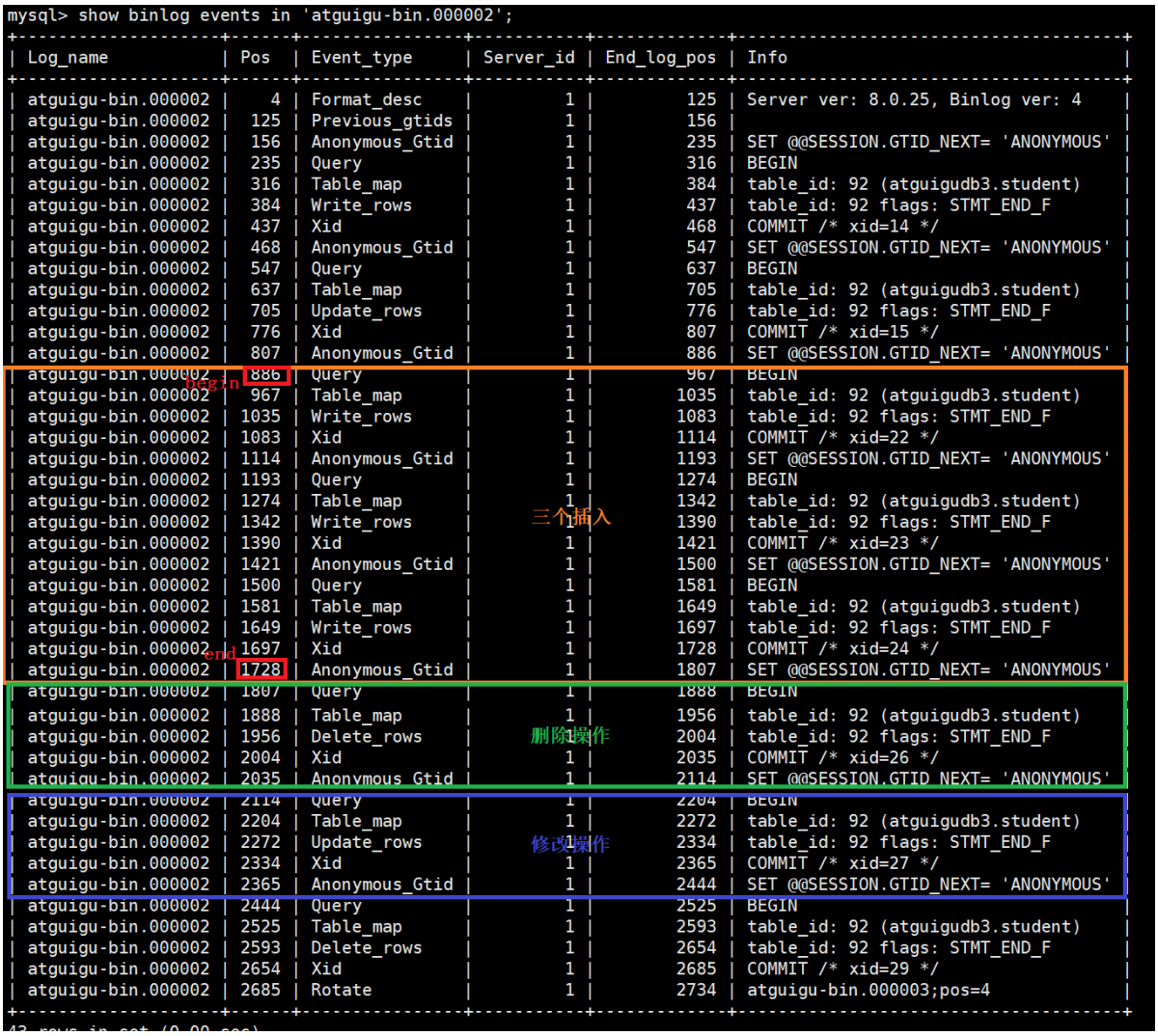
尝试恢复

```
# 查看当前数据库的bin_log日志
mysql> show binary logs;
+--------------------+-----------+-----------+
| Log_name           | File_size | Encrypted |
+--------------------+-----------+-----------+
| atguigu-bin.000001 |       179 | No        |
| atguigu-bin.000002 |      2685 | No        |
+--------------------+-----------+-----------+
2 rows in set (0.00 sec)

#新增binlog【利用日志恢复本质上依旧是对表进行增删改，仍然会产生bin_log日志。所以我们应该新增一个bin_log日志，避免对用于恢复的日志00002产生影响】
mysql> flush logs;
Query OK, 0 rows affected (0.01 sec)

mysql> show binary logs;
+--------------------+-----------+-----------+
| Log_name           | File_size | Encrypted |
+--------------------+-----------+-----------+
| atguigu-bin.000001 |       179 | No        |
| atguigu-bin.000002 |      2734 | No        |
| atguigu-bin.000003 |       156 | No        |
+--------------------+-----------+-----------+
3 rows in set (0.00 sec)
```

位置恢复show binlog events in



Log_name	Pos	Event_type	Server_id	End_log_pos	Info
atguigu-bin.000002	4	Format_desc	1	125	Server ver: 8.0.25, Binlog ver: 4
atguigu-bin.000002	125	Previous_gtids	1	156	
atguigu-bin.000002	156	Anonymous_Gtid	1	235	SET @@SESSION.GTID_NEXT= 'ANONYMOUS'
atguigu-bin.000002	235	Query	1	316	BEGIN
atguigu-bin.000002	316	Table_map	1	384	table_id: 92 (atguigudb3.student)
atguigu-bin.000002	384	Write_rows	1	437	table_id: 92 flags: STMT_END_F
atguigu-bin.000002	437	Xid	1	468	COMMIT /* xid=14 */
atguigu-bin.000002	468	Anonymous_Gtid	1	547	SET @@SESSION.GTID_NEXT= 'ANONYMOUS'
atguigu-bin.000002	547	Query	1	637	BEGIN
atguigu-bin.000002	637	Table_map	1	705	table_id: 92 (atguigudb3.student)
atguigu-bin.000002	705	Update_rows	1	776	table_id: 92 flags: STMT_END_F
atguigu-bin.000002	776	Xid	1	807	COMMIT /* xid=15 */
atguigu-bin.000002	807	Anonymous_Gtid	1	886	SET @@SESSION.GTID_NEXT= 'ANONYMOUS'
atguigu-bin.000002	886	Query	1	967	BEGIN
atguigu-bin.000002	967	Table_map	1	1035	table_id: 92 (atguigudb3.student)
atguigu-bin.000002	1035	Write_rows	1	1083	table_id: 92 flags: STMT_END_F
atguigu-bin.000002	1083	Xid	1	1114	COMMIT /* xid=22 */
atguigu-bin.000002	1114	Anonymous_Gtid	1	1193	SET @@SESSION.GTID_NEXT= 'ANONYMOUS'
atguigu-bin.000002	1193	Query	1	1274	BEGIN
atguigu-bin.000002	1274	Table_map	1	1342	table_id: 92 (atguigudb3.student)
atguigu-bin.000002	1342	Write_rows	1	1390	table_id: 92 flags: STMT_END_F
atguigu-bin.000002	1390	Xid	1	1421	COMMIT /* xid=23 */
atguigu-bin.000002	1421	Anonymous_Gtid	1	1500	SET @@SESSION.GTID_NEXT= 'ANONYMOUS'
atguigu-bin.000002	1500	Query	1	1581	BEGIN
atguigu-bin.000002	1581	Table_map	1	1649	table_id: 92 (atguigudb3.student)
atguigu-bin.000002	1649	Write_rows	1	1697	table_id: 92 flags: STMT_END_F
atguigu-bin.000002	1697	Xid	1	1728	COMMIT /* xid=24 */
atguigu-bin.000002	1728	Anonymous_Gtid	1	1807	SET @@SESSION.GTID_NEXT= 'ANONYMOUS'
atguigu-bin.000002	1807	Query	1	1888	BEGIN
atguigu-bin.000002	1888	Table_map	1	1956	table_id: 92 (atguigudb3.student)
atguigu-bin.000002	1956	Delete_rows	1	2004	table_id: 92 flags: STMT_END_F
atguigu-bin.000002	2004	Xid	1	2035	COMMIT /* xid=26 */
atguigu-bin.000002	2035	Anonymous_Gtid	1	2114	SET @@SESSION.GTID_NEXT= 'ANONYMOUS'
atguigu-bin.000002	2114	Query	1	2204	BEGIN
atguigu-bin.000002	2204	Table_map	1	2272	table_id: 92 (atguigudb3.student)
atguigu-bin.000002	2272	Update_rows	1	2334	table_id: 92 flags: STMT_END_F
atguigu-bin.000002	2334	Xid	1	2365	COMMIT /* xid=27 */
atguigu-bin.000002	2365	Anonymous_Gtid	1	2444	SET @@SESSION.GTID_NEXT= 'ANONYMOUS'
atguigu-bin.000002	2444	Query	1	2525	BEGIN
atguigu-bin.000002	2525	Table_map	1	2593	table_id: 92 (atguigudb3.student)
atguigu-bin.000002	2593	Delete_rows	1	2654	table_id: 92 flags: STMT_END_F
atguigu-bin.000002	2654	Xid	1	2685	COMMIT /* xid=29 */
atguigu-bin.000002	2685	Rotate	1	2734	atguigu-bin.000003;pos=4

查看日志发现，在备份数据后首先执行的是插入数据操作，在Info信息中xid的值分别是22、23、24 (紧邻的三项)

思路：先恢复3个insert22 23 24，再恢复delete 26，最后update 27

步骤1:恢复插入的数据

```
#语法
[root@centos7-mysql-1 mysql]# /usr/bin/mysqlbinlog --start-position=初位置 --stop-position=初位置 --database=数据库名 /var/lib/mysql/日志文件 | /usr/bin/mysql -uroot -p密码 -v 数据库名
#实战
[root@hadoop102 mysql]# /usr/bin/mysqlbinlog --start-position=886 --stop-position=1728 --database=atguigudb3 /var/lib/mysql/atguigu-bin.000002 | /usr/bin/mysql -uroot -proot -v atguigudb3;
```

执行完进行查看，发现这三个插入操作已经被恢复

```
mysql>  select * from student;
+----+---------+--------+
| id | name    | class  |
+----+---------+--------+
|  1 | 张三2   | 一班   |
|  3 | 李四1   | 一班   |
|  6 | Jane    | 一班   |
|  8 | 王五    | 二班   |
| 15 | Tom     | 二班   |
| 18 | Jerry   | 四班   |
| 20 | 钱七    | 三班   |
| 21 | aaa     | No1    |
| 22 | aaa     | No1    |
| 23 | aaa     | No1    |
+----+---------+--------+
10 rows in set (0.00 sec)
```

步骤2:恢复删除的数据

```
[root@hadoop102 mysql]# /usr/bin/mysqlbinlog --start-position=1807 --stop-position=2035 --database=atguigudb3 /var/lib/mysql/atguigu-bin.000002 | /usr/bin/mysql -uroot -proot -v atguigudb3;
```

查看结果

```
mysql>  select * from student;
+----+---------+--------+
| id | name    | class  |
+----+---------+--------+
|  1 | 张三2   | 一班   |
|  3 | 李四1   | 一班   |
|  6 | Jane    | 一班   |
|  8 | 王五    | 二班   |
| 15 | Tom     | 二班   |
| 18 | Jerry   | 四班   |
| 20 | 钱七    | 三班   |
| 22 | aaa     | No1    |
| 23 | aaa     | No1    |
+----+---------+--------+
9 rows in set (0.00 sec)
```

步骤3:恢复修改的数据的数据

```
[root@hadoop102 mysql]# /usr/bin/mysqlbinlog --start-position=2114 --stop-position=2365 --database=atguigudb3 /var/lib/mysql/atguigu-bin.000002 | /usr/bin/mysql -uroot -proot -v atguigudb3;
```

```
mysql> select * from student;
+----+---------+--------+
| id | name    | class  |
+----+---------+--------+
|  1 | 张三2   | 一班   |
|  3 | 李四1   | 一班   |
|  6 | Jane    | 一班   |
|  8 | 王五    | 二班   |
| 15 | Tom     | 二班   |
| 18 | Jerry   | 四班   |
| 20 | 钱七    | 三班   |
| 22 | bbb     | No1    |
| 23 | aaa     | No1    |
+----+---------+--------+
9 rows in set (0.00 sec)
```

可以看到最终结果和删除数据之前的结果一样，利用binlog实现了数据恢复。

当然也可以使用日期恢复，命令格式如下：

```
/usr/bin/mysqlbinlog --start-datetime="2023-07-22 16:14:20” --stop-datetime="2023-07-22 16:19:02"  --database=atguigudb3 /var/lib/mysql/binlog/atguigu-bin.000002 | /usr/bin/mysql -uroot root  -v atguigudb3
```

另外，有时候可能出现一个事务A执行时间过短，几秒内执行完成。此时我们找到下一个事务B的开始时间和结束时间。只要恢复的时间在B结束时间之前就可以只恢复A不恢复B【只要时间不完全包含一个事务，那么此事务就不会进行恢复~】

```
# at 4
#230722 16:14:20 server id 1  end_log_pos 125 CRC32 0xfbe10f64  Start: binlog v 4, server v 8.0.25 created 230722 16:14:20 at startup
# Warning: this binlog is either in use or was not closed properly.
ROLLBACK/*!*/;
BINLOG '
'/*!*/;
# at 547
#230722 16:19:02 server id 1  end_log_pos 637 CRC32 0x0e4d6052  Query thread_id=8 exec_time=0 error_code=0
SET TIMESTAMP=1690013942/*!*/;
BEGIN
//......
# at 776
#230722 16:19:02 server id 1  end_log_pos 807 CRC32 0x6f80cb79  Xid = 15
COMMIT/*!*/;
```

```
SET @@SESSION.GTID_NEXT= 'AUTOMATIC' /* added by mysqlbinlog */ /*!*/;
DELIMITER ;
# End of log file
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
```

mysqlbinlog命令对于意外操作非常有效，比如因操作不当误删了数据表

5.5 删除二进制日志

MySQL的二进制文件可以配置自动删除，同时MySQL也提供了安全的手动删除二进制文件的方法。`PURGE MASTER LOGS`只删除指定部分的二进制日志文件，`RESET MASTER`删除所有的二进制日志文件。具体如下：

1. PURGE MASTER LOGS：删除指定日志文件

PURGE MASTER LOGS语法如下：

```
PURGE {MASTER | BINARY} LOGS TO ‘指定日志文件名’
PURGE {MASTER | BINARY} LOGS BEFORE ‘指定日期’
```

举例:使用PURGE MASTER LOGS语句删除创建时间比binlog.000005早的所有日志

1. 多次重新启动MySQL服务，便于生成多个日志文件。然后用SHOW语句显示二进制日志文件列表

```
SHOW BINARY LOGS;
```

2. 执行PURGE MASTER LOGS语句删除创建时间比binlog.000005早的所有日志

```
PURGE MASTER LOGS TO "binlog.000005"; # 不会删除005
```

3. 显示二进制日志文件列表

```
SHOW BINARY LOGS ;
```

比binlog.000005早的所有日志文件都已经被删除了。

举例:使用PURGE MASTER LOGS语句删除2020年10月25号前创建的所有日志文件。具体步骤如下:

1. 显示二进制日志文件列表

```
SHOW BINARY LOGS;
```

2. 执行mysqlbinlog命令查看二进制日志文件binlog.000005的内容

```
mysqlbinlog --no-defaults "/var/lib/mysql/atguigu-bin.000005"
```

结果可以看出20230805为日志创建的时间，即2023年08月05日。

3. 使用PURGE MASTER LOGs语句删除2022年1月05日前创建的所有日志文件

```
PURGE MASTER LOGS before "20220105";
```

4. 显示二进制日志文件列表

```
SHOW BINARY LOGS ;
```

2023年08月05号之前的二进制日志文件都已经被删除，最后一个没有删除，是因为当前在用，还未记录最后的时间，所以未被删除。

2.RESET MASTER:删除所有二进制日志文件

使用`RESET MASTER`语句，清空所有的binlog日志。MySQL会重新创建二进制文件，新的日志文件扩展名将重新从000001开始编号。`慎用！`

举例:使用RESET MASTER语句删除所有日志文件。

1. 重启MySQL服务若干次，执行SHOW语句显示二进制日志文件列表。

```
SHOW BINARY LOGS;
```

2. 执行RESET MASTER语句，删除所有日志文件

```
RESET MASTER;
```

执行完该语句后，原来的所有二进制日志已经全部被删除。

5.6 其它场景

二进制日志可以通过数据库的`全量备份`和二进制日志中保存的`增量信息`，完成数据库的`无损失恢复`。但是，如果遇到数据量大、数据库和数据表很多（比如分库分表的应用）的场景，用二进制日志进行数据恢复，是很有挑战性的，因为起止位置不容易管理。

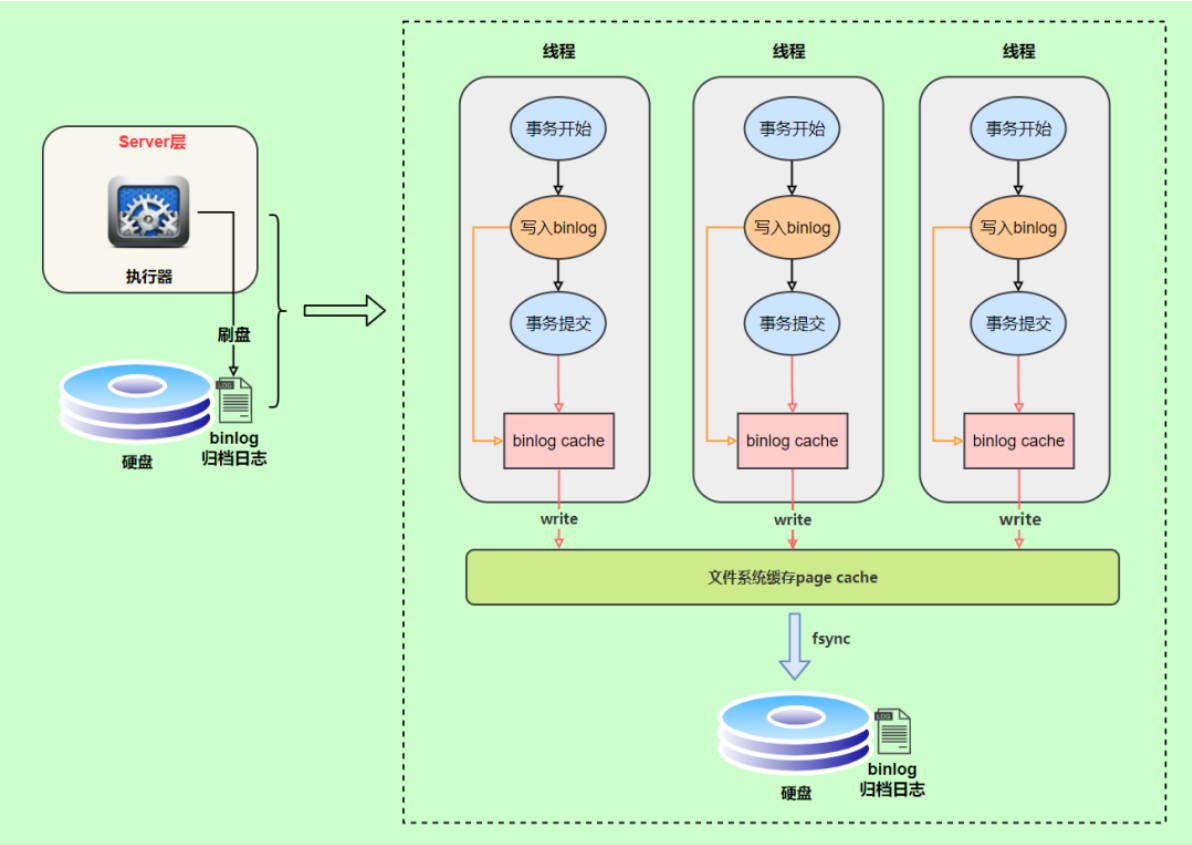
在这种情况下，一个有效的解决办法是`配置主从数据库服务器`，甚至是`一主多从`的架构，把二进制日志文件的内容通过中继日志，同步到从数据库服务器中，这样就可以有效避免数据库故障导致的数据异常等问题。

6. 再谈二进制日志(binlog)

6.1 写入机制

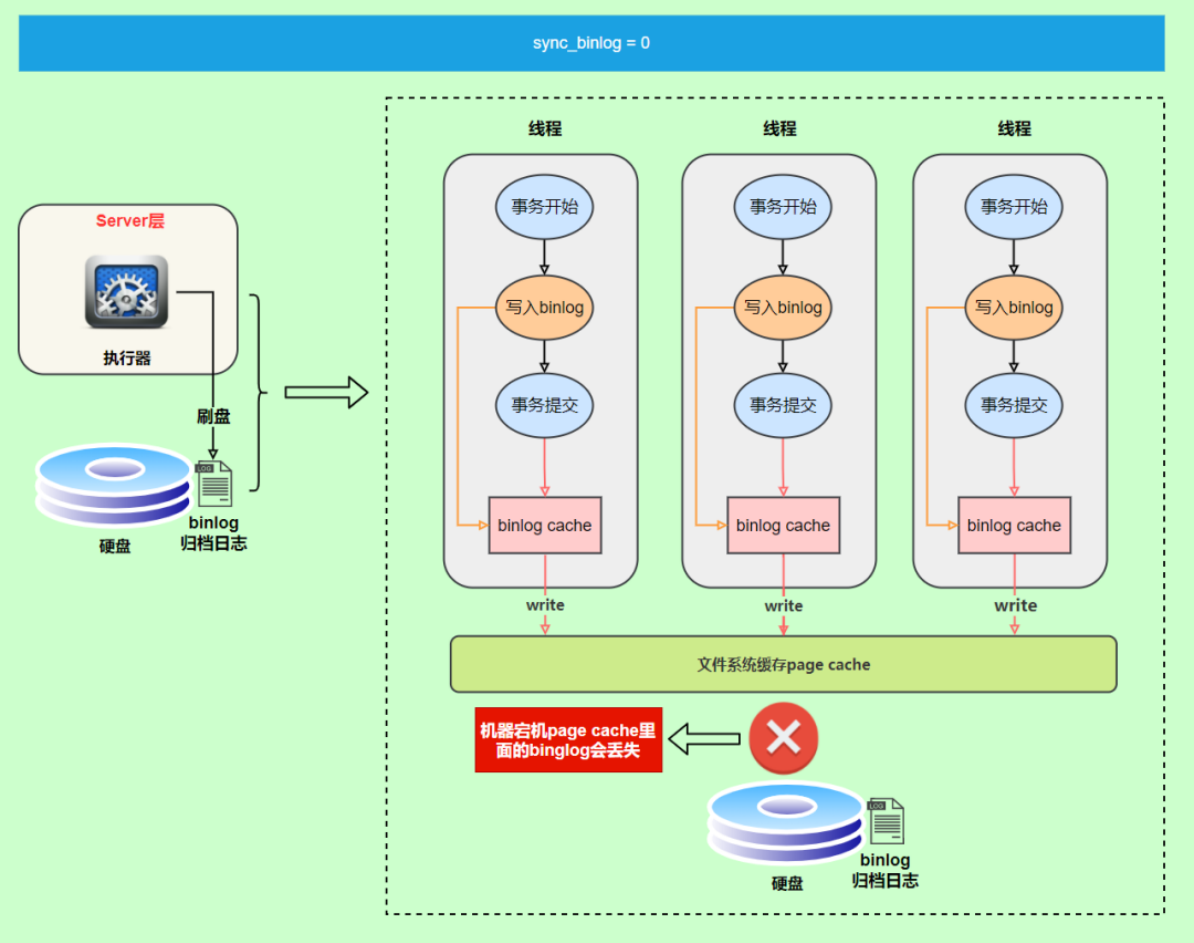
binlog的写入时机也非常简单，事务执行过程中，先把日志写到`binlog cache`，事务提交的时候，再把binlogcache写到binlog文件中。因为**一个事务的binlog不能被拆开，无论这个事务多大，也要确保一次性写入**，所以系统会给每个线程分配一个块内存作为binlog cache。

我们可以通过`binlog_cache_size`参数控制单个线程binlog cache大小，如果存储内容超过了这个参数，就要暂存到磁盘(Swap)。binlog日志刷盘流程如下:



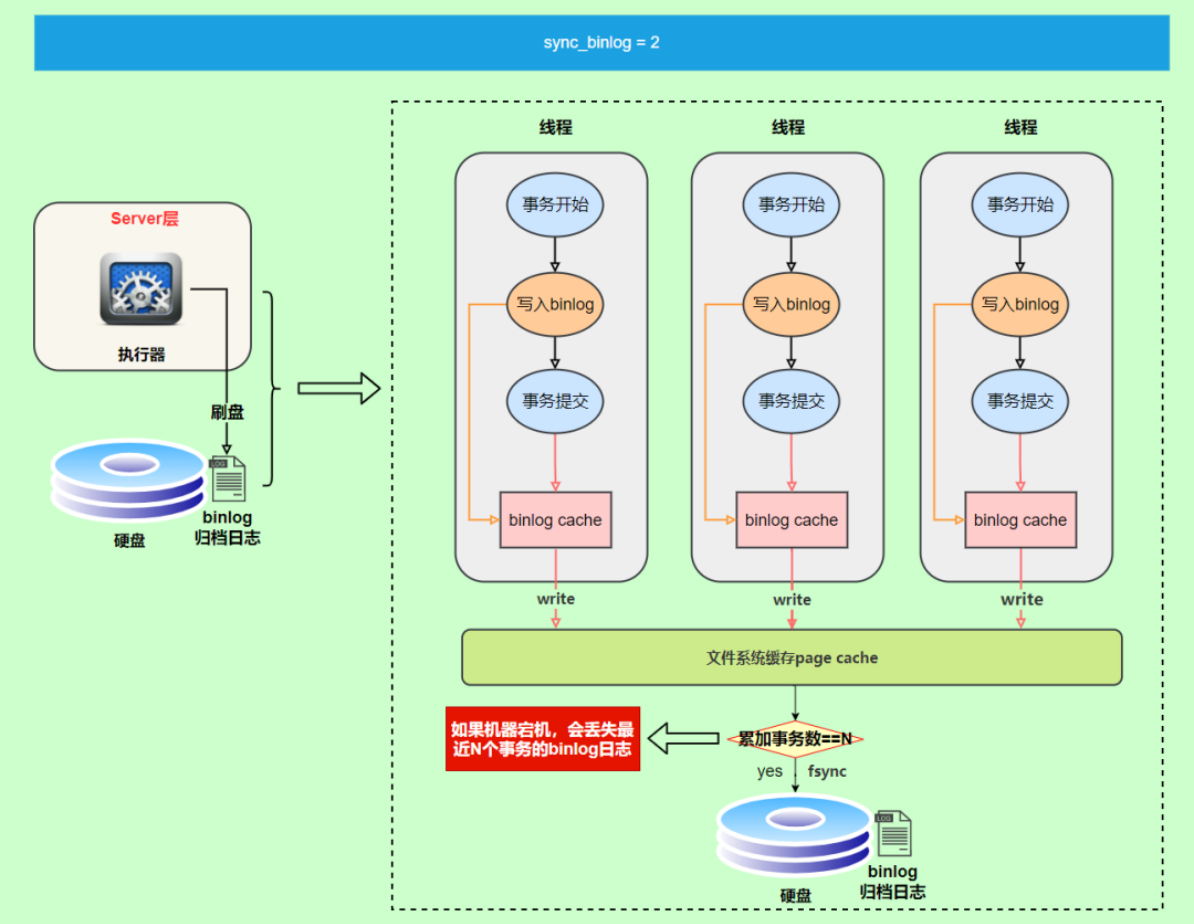
> - 上图的write，是指把日志写入到文件系统的page cache，并没有把数据持久化到磁盘，所以速度比较快
> - 上图的fsync，才是将数据持久化到磁盘的操作

write和fsync的时机，可以由参数`sync_binlog`控制，默认是`0`。为0的时候，表示每次提交事务都只write，由系统自行判断什么时候执行fsync。虽然性能得到提升，但是机器宕机，page cache里面的binlog会丢失。如下图:



为了安全起见，可以设置为1，表示每次提交事务都会执行fsync，就如同`redo log刷盘流程`一样。

最后还有一种折中方式，可以设置为N(N>1)，表示每次提交事务都write，但累积N个事务后才fsync。



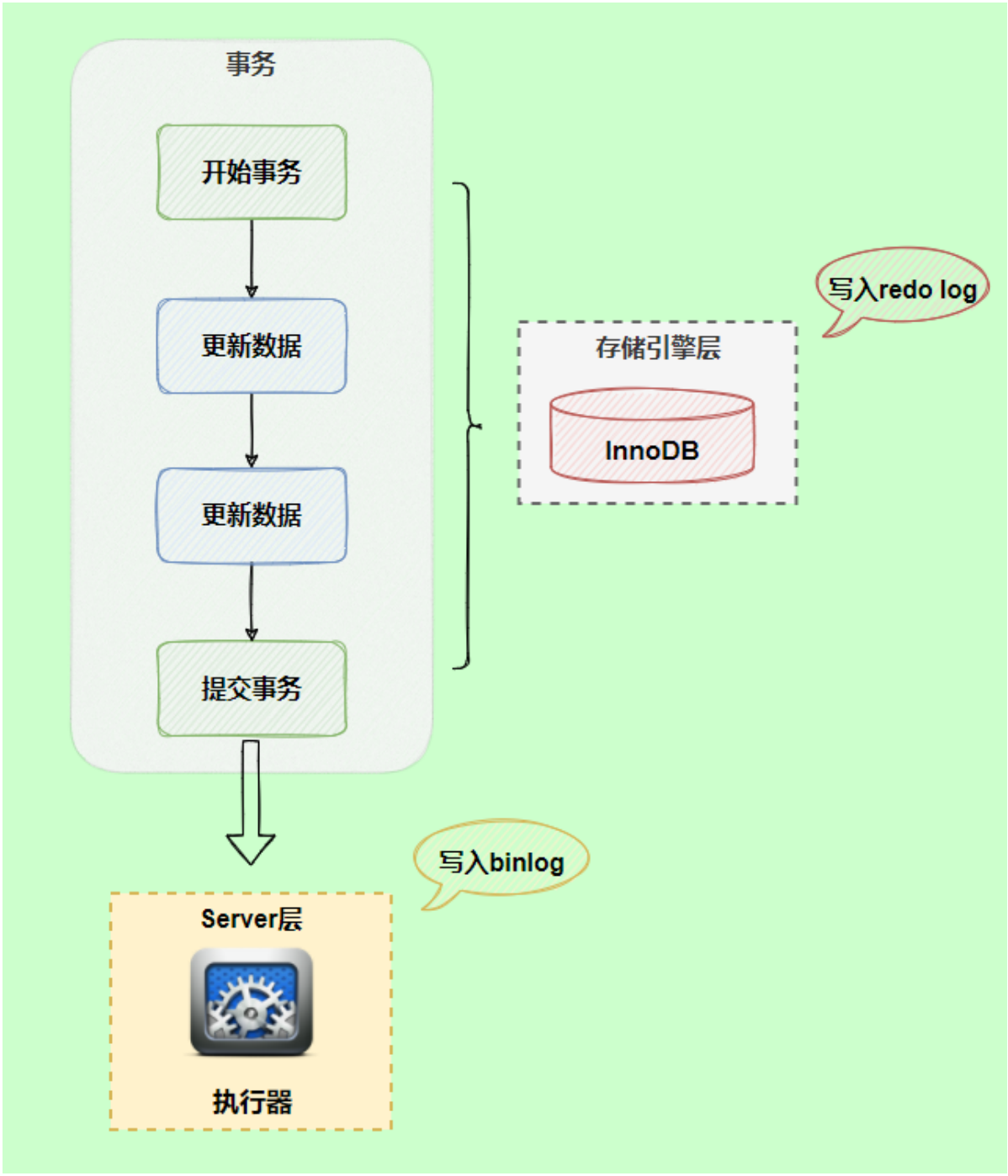
在出现IO瓶颈的场景里，将sync_binlog设置成一个比较大的值，可以提升性能。同样的，如果机器宕机，会丢失最近N个事务的binlog日志。

6.2 binlog与redolog对比

- redo log它是`物理日志`，记录内容是“在某个数据页上做了什么修改”，属于InnoDB存储引擎层产生的。
- 而binlog是`逻辑日志`，记录内容是语句的原始逻辑，类似于“给ID=2这一行的c字段加1”，属于MySQLServer层。
- 虽然它们都属于持久化的保证，但是侧重点不同：
  - redo log让InnoDB存储引擎拥有了崩溃恢复能力
  - binlog 保证了MySQL集群架构的数据一致性

6.3 两阶段提交

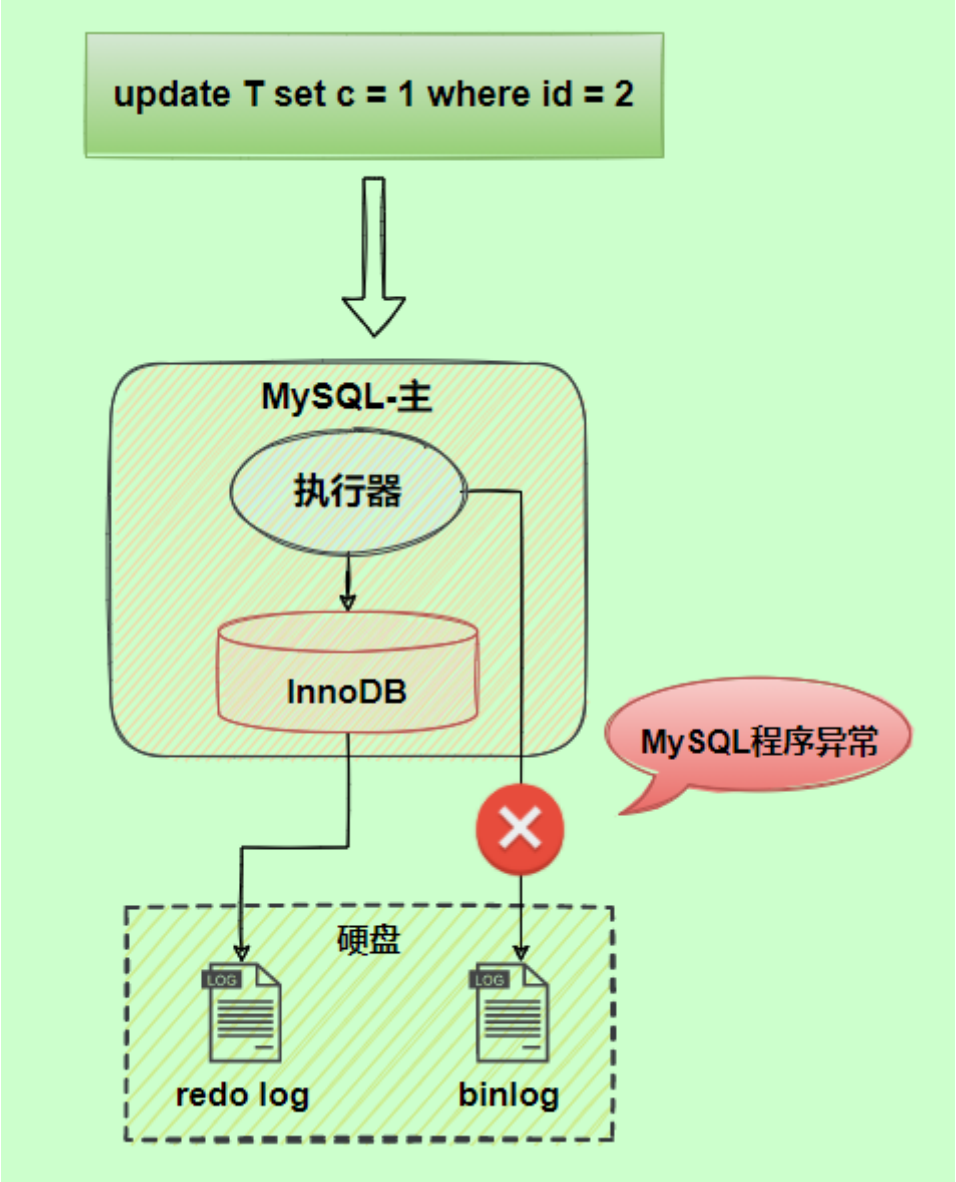
在执行更新语句过程，会记录redolog与binlog两块日志，以基本的事务为单位，redolog在事务执行过程中可以不断写入，而binlog只有在提交事务时才写入，所以redolog与binlog的`写入时机`不一样。



redo log与bin log两份日志之间的逻辑不一致，会出现什么问题?

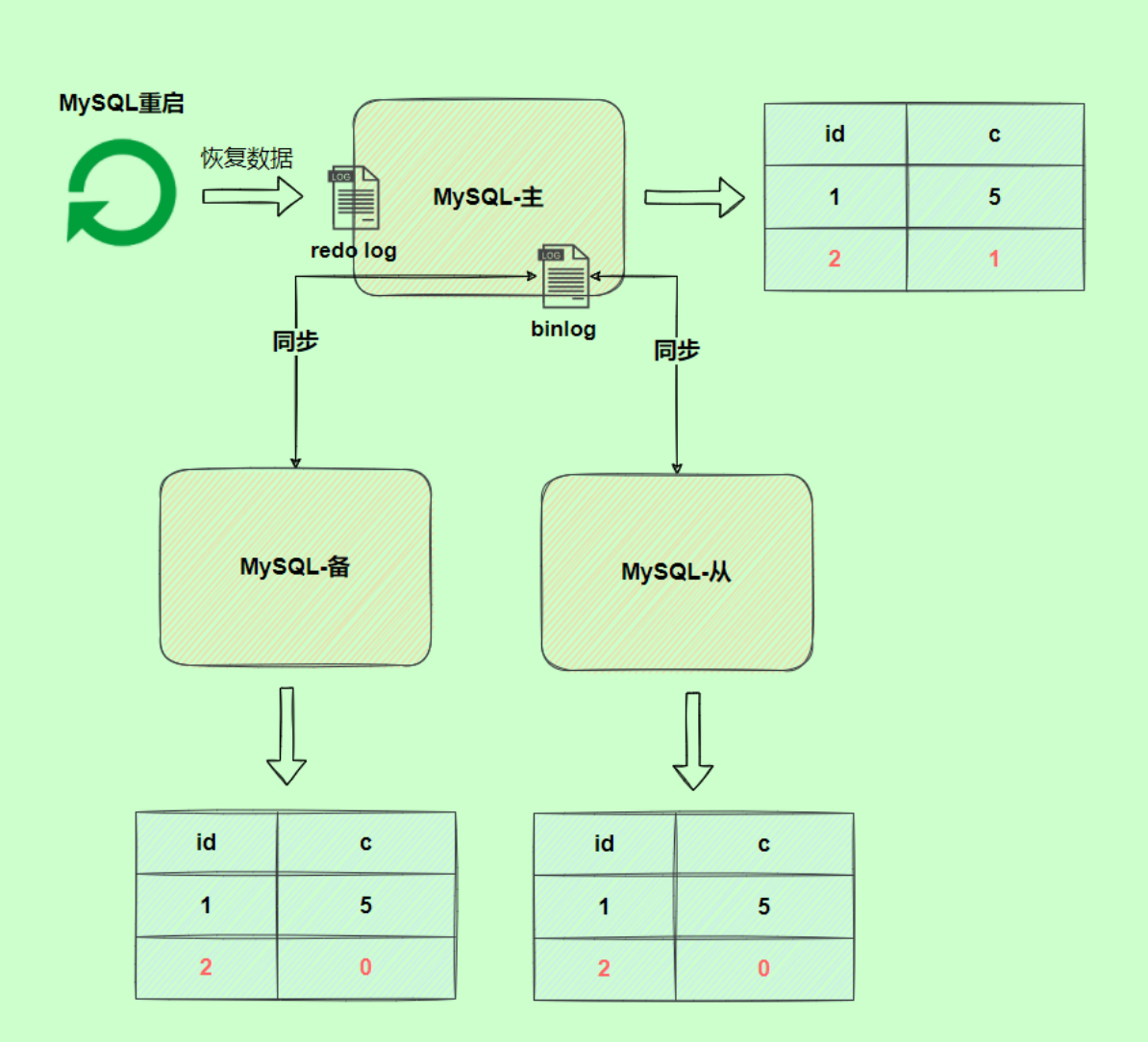
以update语句为例，假设id=2的记录，字段c值是0，把字段c值更新成1，SQL语句为update T set c=1 where id=2。

假设执行过程中写完redo log日志后，binlog日志写期间发生了异常，会出现什么情况呢?

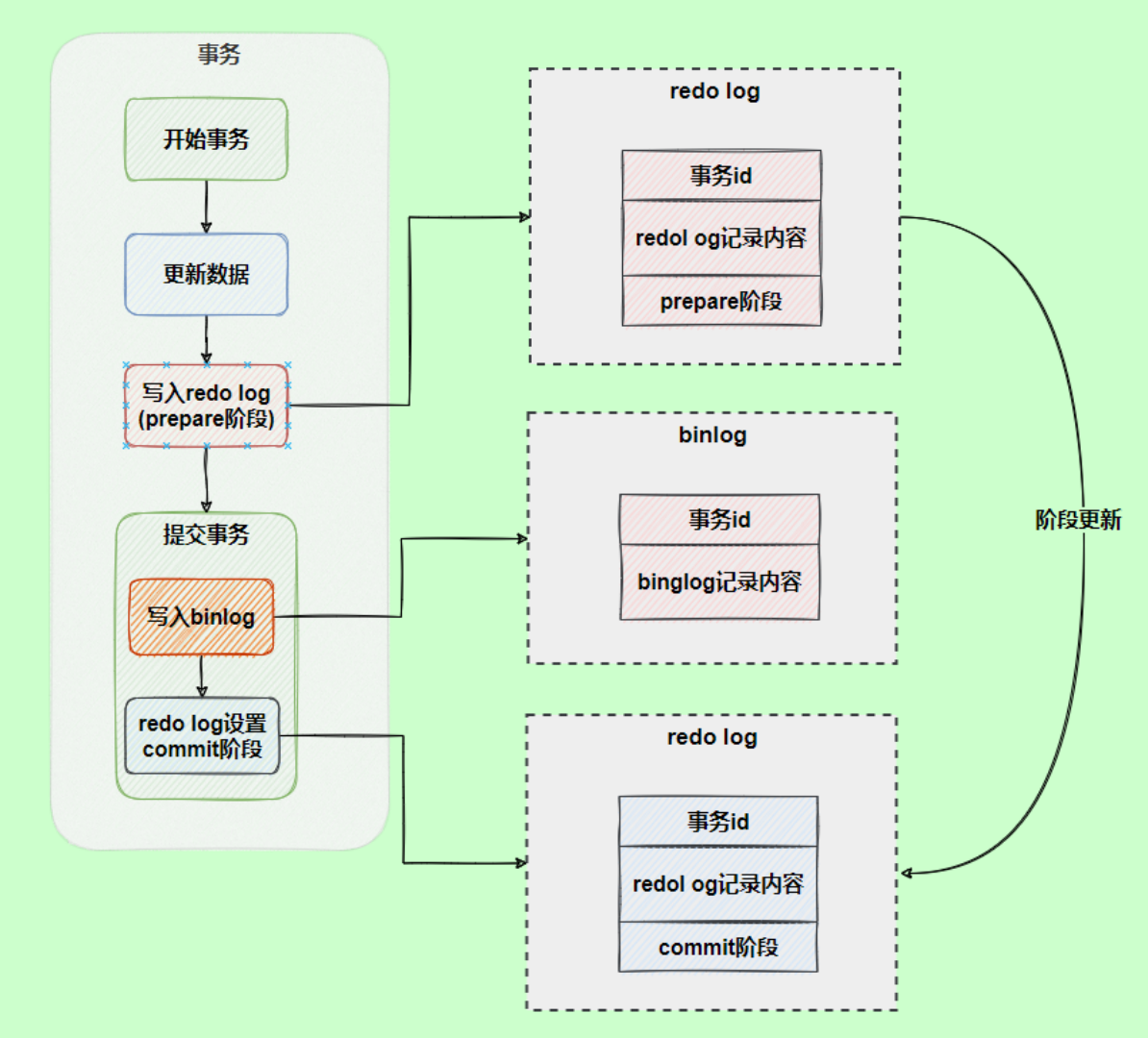


由于binlog没写完就异常，这时候binlog里面没有对应的修改记录。

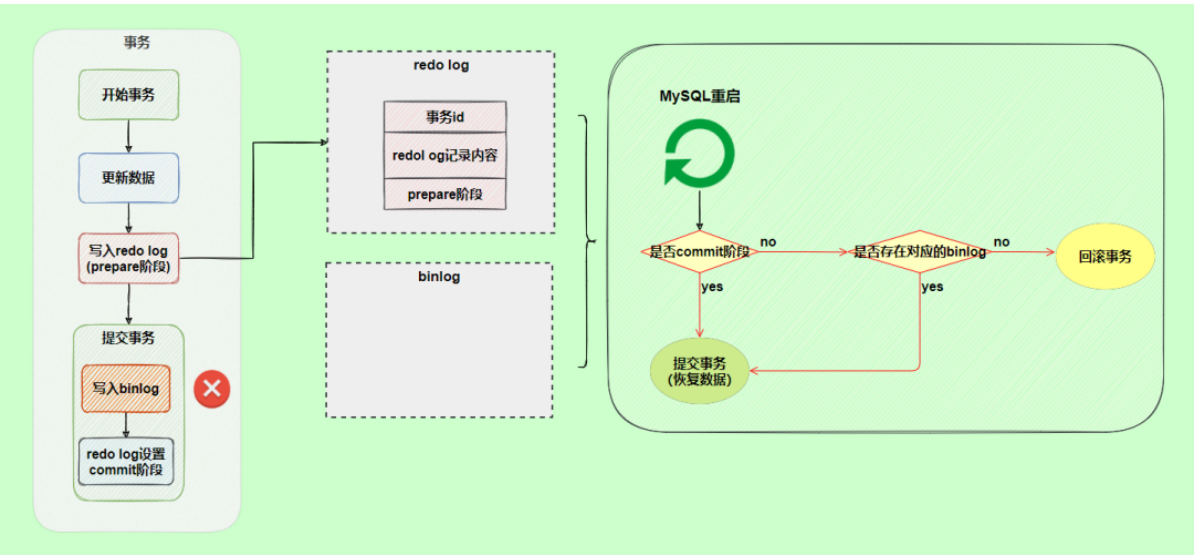
因此，之后用binlog日志恢复数据时，就会少这一次更新，恢复出来的这一行c值是0，而原库因为redo log日志恢复，这一行c值是1，最终数据不一致。



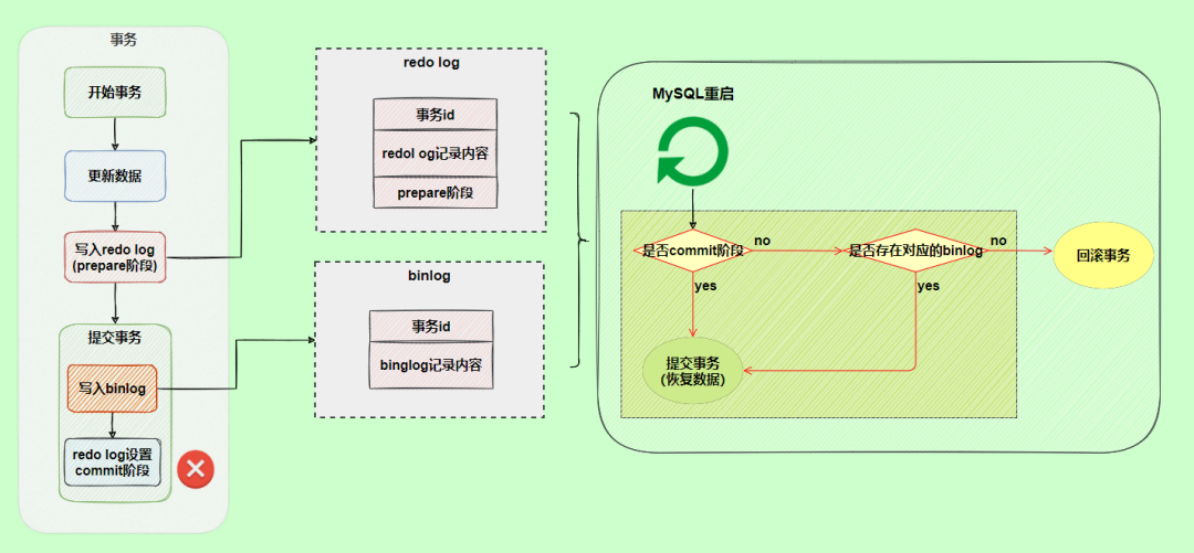
为了解决两份日志之间的逻辑一致问题，InnoDB存储引擎使用`两阶段提交`方案。原理很简单，将redo log的写入拆成了两个步骤prepare和commit，这就是`两阶段提交`。



使用 两阶段提交 后，写入binlog时发生异常也不会有影响，因为MySQL根据redo log日志恢复数据时，发现redolog还处于prepare阶段，并且没有对应binlog日志，就会回滚该事务。



另一个场景，redolog设置commit阶段发生异常，那会不会回滚事务呢？



并不会回滚事务，它会执行上图框住的逻辑，虽然redolog是处于prepare阶段，但是能通过事务id找到对应的binlog日志，所以MySQL认为是完整的，就会提交事务恢复数据。

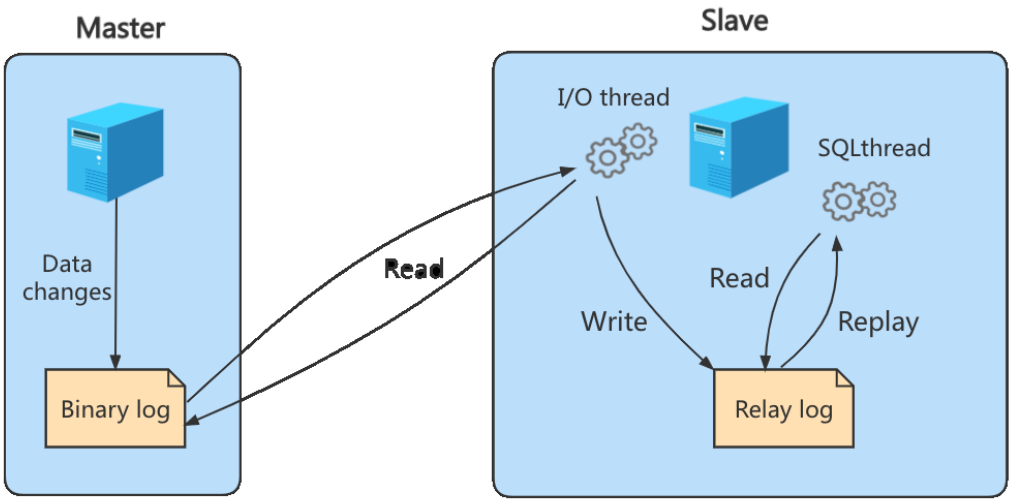
7. 中继日志(relay log)

7.1 介绍

中继日志只在主从服务器架构的从服务器上存在。从服务器为了与主服务器保持一致，要从主服务器读取二进制日志的内容，并且把读取到的信息写入`本地的日志文件`中，这个从服务器本地的日志文件就叫`中继日志`。然后，从服务器读取中继日志，并根据中继日志的内容对从服务器的数据进行更新，完成主从服务器的`数据同步`。

搭建好主从服务器之后，中继日志默认会保存在从服务器的数据目录下。

文件名的格式是：`从服务器名-relay-bin.序号`。中继日志还有一个索引文件：`从服务器名-relay-bin.index`，用来定位当前正在使用的中继日志。



7.2 查看中继日志

中继日志与二进制日志的格式相同，可以用`mysqlbinlog`工具进行查看。下面是中继日志的一个片段：

```
SET TIMESTAMP=1618558728/*!*/;
BEGIN
/*!*/;
# at 950
#210416 15:38:48 server id 1 end_log_pos 832 CRC32 0xcc16d651 Table_map:
`atguigu`.`test` mapped to number 91
# at 1000
#210416 15:38:48 server id 1 end_log_pos 872 CRC32 0x07e4047c Delete_rows:
table id 91 flags: STMT_END_F --server id 1是主服务器，意思是主服务器删了一行数据
BINLOG
'CD95YBMBAAAAMgAAAEADAAAAAFsAAAAAAAEABGRlbW8ABHRlc3QAAQMAAQEBAFHWFsw=CD95YCABA
AAAKAAAAGgDAAAAAFsAAAAAAAEAAgAB/wABAAAAfATkBw=='/*!*/;
# at 1040
```

这一段的意思是，主服务器（“serverid1”）对表atguigu.test进行了2步操作：

```
定位到表atguigu.test编号是91的记录，日志位置是832；
删除编号是91的记录，日志位置是872d。
```

7.3 恢复的典型错误

如果从服务器宕机，有的时候为了系统恢复，要重装操作系统，这样就可能会导致你的`服务器名称`与之前`不同`。而中继日志里`是包含从服务器名`的。在这种情况下，就可能导致你恢复从服务器的时候，无法从宕机前的中继日志里读取数据，以为是日志文件损坏了，其实是名称不对了。

解决的方法也很简单，把从服务器的名称改回之前的名称。