MySQL日志

一、MySQL日志

1.1、日志类型

- **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。（除二进制日志外，其他日志都是 文本文件）
- **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。
- **重做日志（redo log）**：记录数据库中所有对数据进行修改的操作，它用于确保数据库在系统故障后能够将事务的操作恢复到已提交的状态。
- **撤销日志（undo log）**：记录数据库中数据修改的日志，用于撤销尚未提交的事务对数据的修改。它的主要作用是在事务回滚时恢复数据到事务开始前的状态。

说明：默认情况下，所有日志创建于 MySQL数据目录 中

1.2、日志优缺点

优点：

- 便于记录数据库运行中的诊断信息，包括错误、警告、注释等信息,便于问题的排查
- 在数据复制、数据恢复、操作审计以及确保数据永久性和一致性方面（redo日志）有着不可替代作用

缺点：

- 日志功能会 降低MySQL数据库的性能。例如，在查询非常频繁的MySOL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。

二、慢查询日志（slow query log △）

可以再第09章中进行补全

三、通用查询日志（general query log）

3.1 说明

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。

3.2 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。
我们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。

3.3 相关操作

1.查看当前状态

查看设置后情况：

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

```
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，在需要的时候开启日志。
- 说明2:通用查询日志文件的名称是 atguigu01.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。

2.启动日志

- 永久方式：修改配置文件的方式

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）

- 临时性方式

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

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

3.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Mac OSX系统中，可以使用文本文件查看器或者vi等工具查看

日志示例：

```
/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。

4.停止日志

- 永久方式

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。

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

- 临时方式

使用SET语句停止MySQL通用查询日志功能：

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

5.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：

```
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。

- 重新生成：

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
```

四、错误日志

4.1 说明

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。

4.2 相关命令

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** 。（相较于通用查询日志，其更加明确地可以看出错误信息，而不需要在一堆内容中找错误信息）

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。

2.查看错误日志

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。

3. 删除/刷新错误日志

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。

直接删除：

```
[root@atguigu01 log]# mysqladmin -uroot -p flush-logs
Enter password:
```

报错：

```
mysqladmin: refresh failed; error: 'Could not open file '/var/log/mysqld.log' for
error logging.'
```

官方解决：

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
```

需要执行操作：

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

原因：

对于flush-logs指令：

- 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。

五、二进制日志（binlog）

5.1 说明

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。

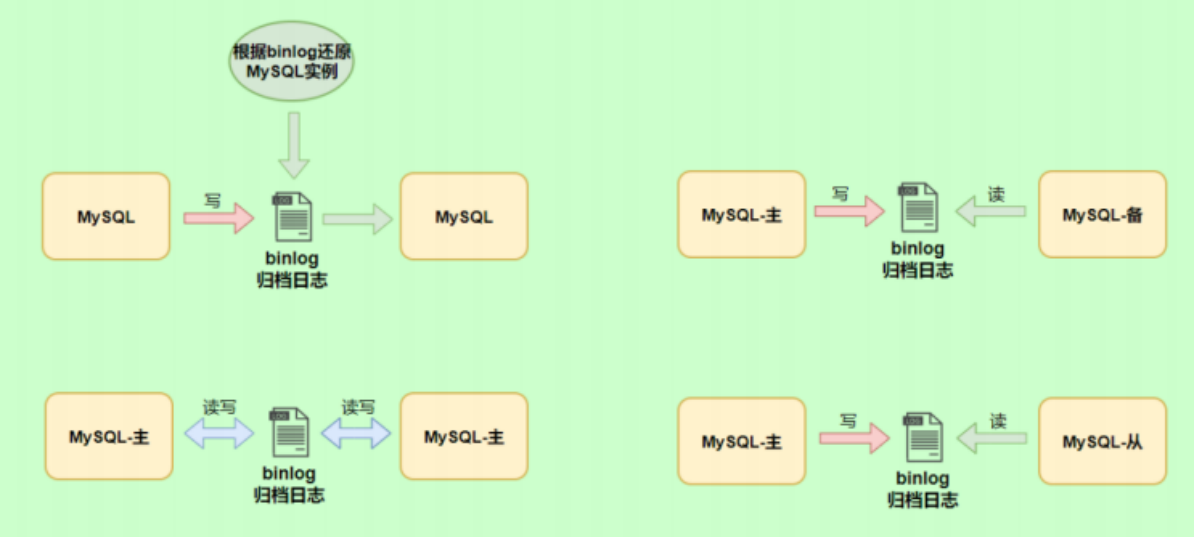
binlog即binary log，二进制日志文件，也叫作变更日志（update log）。它记录了数据库所有执行的DDL 和 DML 等数据库更新事件的语句，但是不包含没有修改任何数据的语句（如数据查询语句select、show等）

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）。

5.2 使用场景

- 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来闱步数据，保证数据一致性。



5.3 相关操作

1.查看默认情况

通过命令`show variables like '%log_bin%';`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)
```

参数说明：

- log_bin: bin_log日志是否开启
- log_bin_basename：默认情况下存储的位置（其会生成很多个binlog文件，如binlog00001，每一个文件由大小上限，会自己换文件存储）
- log_bin_index: 为binlog日志建立的索引
- log_bin_trust_function_creators:是否信任函数（防止主从调用函数不一致性，如now函数，主机和从机通过now函数获取的时间是不一样的）
- log_bin_usr_v1_row_events: /
- sql_log_bin:支持把变更行为的语句进行记录

2.日志参数设置

- 永久方式

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
```

- 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
- 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 30天--14400 4小时;864001天;2592003天;
- 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3.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/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)
```

设置带文件夹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：

1、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
```

2、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送故障时，可以使用日志文件恢复过程。

- 临时方法

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)
```

重启MySQL服务之后，新的二进制日志文件将出现在如下文件夹下

```
mysql> show variables like '%log_bin%';
+---------------------------------+-----------------------------------------+
| Variable_name                   | Value                                   |
+---------------------------------+-----------------------------------------+
| log_bin                         | ON                                      |
| log_bin_basename                | /var/lib/mysql/binlog/atguigu-bin       |
| log_bin_index                   | /var/lib/mysql/binlog/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)
```

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。

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,@@session.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//8AAQAAAAblvKDkuIkG5LiA54+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,@@session.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
### 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）

示例：

```
#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.000002' 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.000002' 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.000002' 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格式通过命令show variables like 'binlog_format';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row level 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：row level 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。

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

具体过程（△ 可以实操一下）：

- 先通过flush logs，生成一个新的binlog文件，与之前需要恢复的binlog文件分开
- 通过时间方式或position的方式去进行数据的恢复

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

- PURGE MASTER LOGS：删除指定日志文件

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1：使用PURGE MASTER LOGS语句删除创建时间比binlog.0005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.000085";
```

（3）显示二进制日志文件列表

```
SHOW BINARY LOGS;
```

此时比binlog.000005早的所有日志文件都已经被删除了。

举例2:

- RESET MASTER：删除所有的二进制日志文件

使用 RESET MASTER语句，清空所有的binlog日志。MVSOL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!

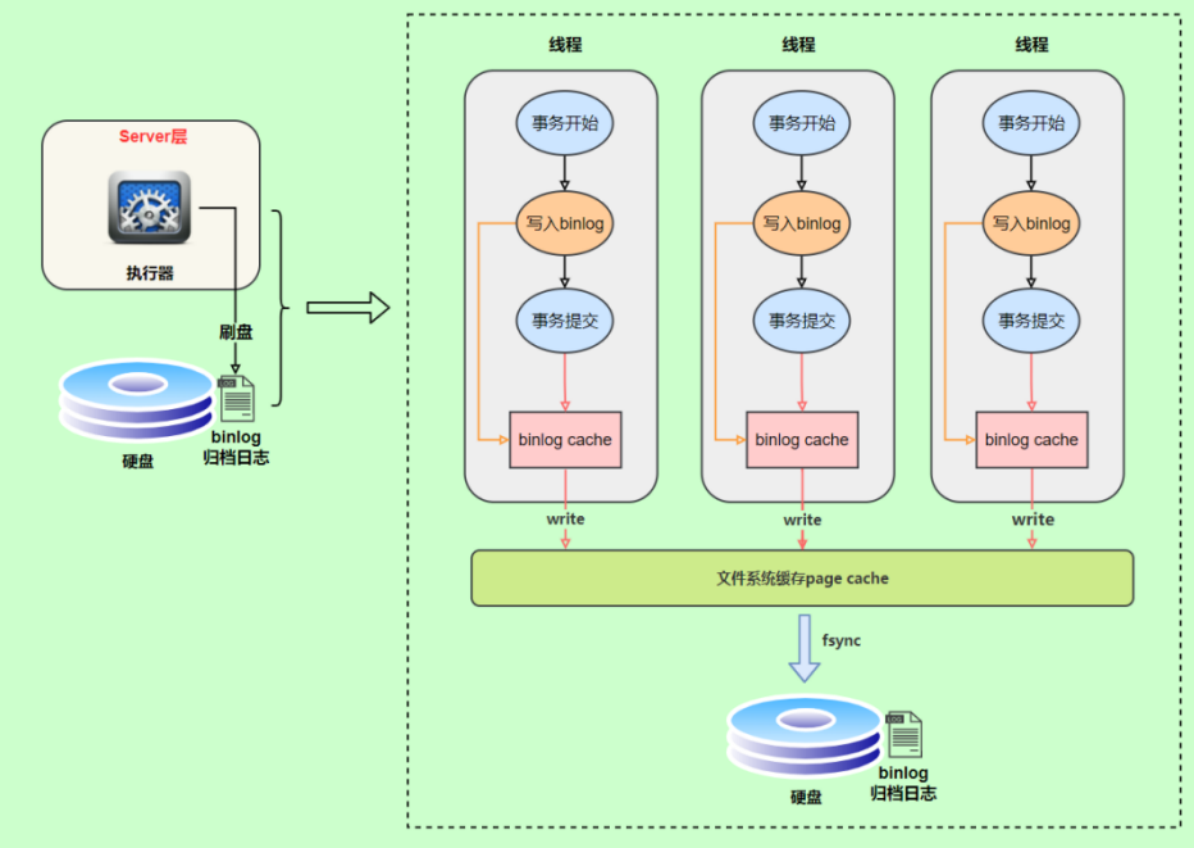
6.其他场景

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。

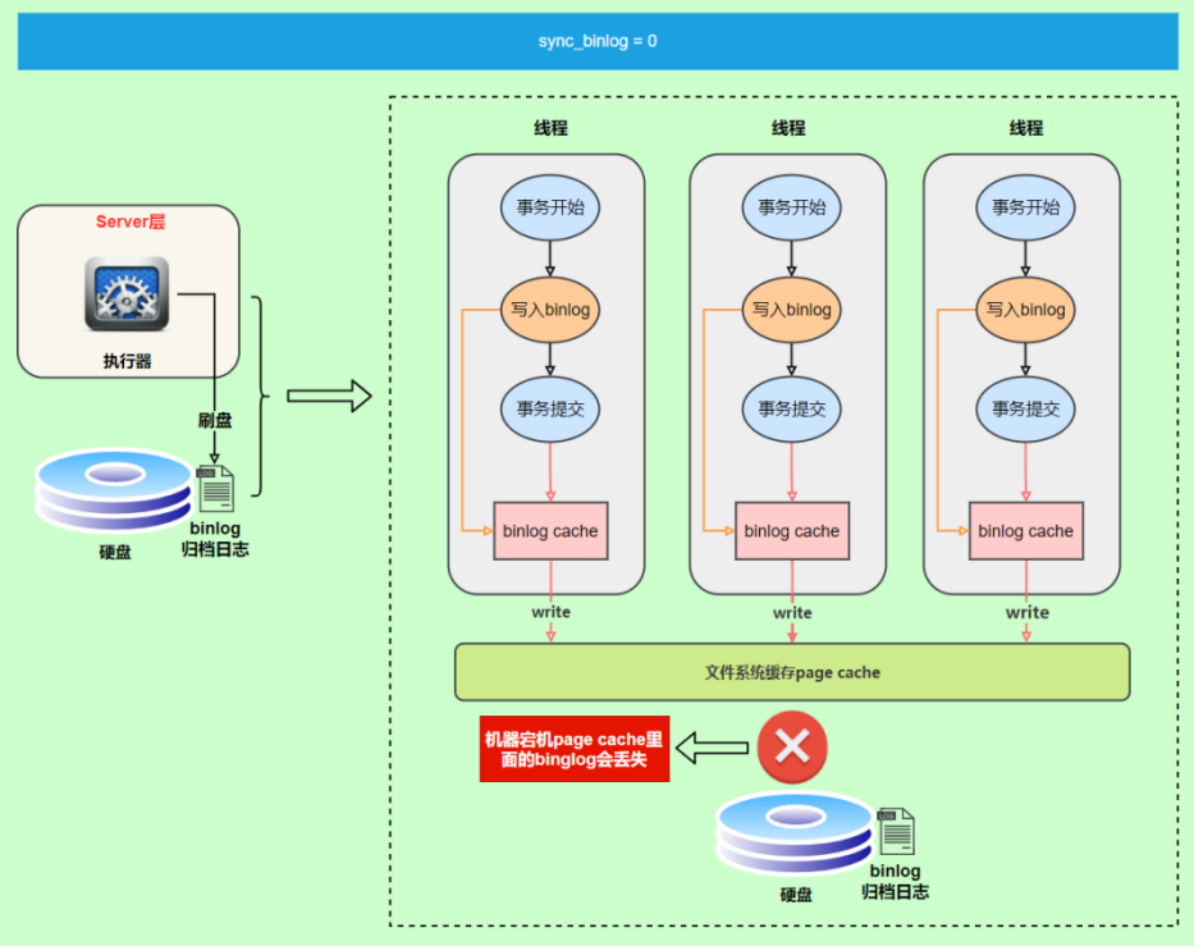
5.4 写入机制

binlog的写入时机也非常简单，事务执行过程中，先把日志（更改行为的操作）写到binlog cache（内存中的缓存），事务提交的时候，再把binlog cache先写入到文件系统缓存Page cache中，再以一定的时机同步到磁盘中的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binlogcache。
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:



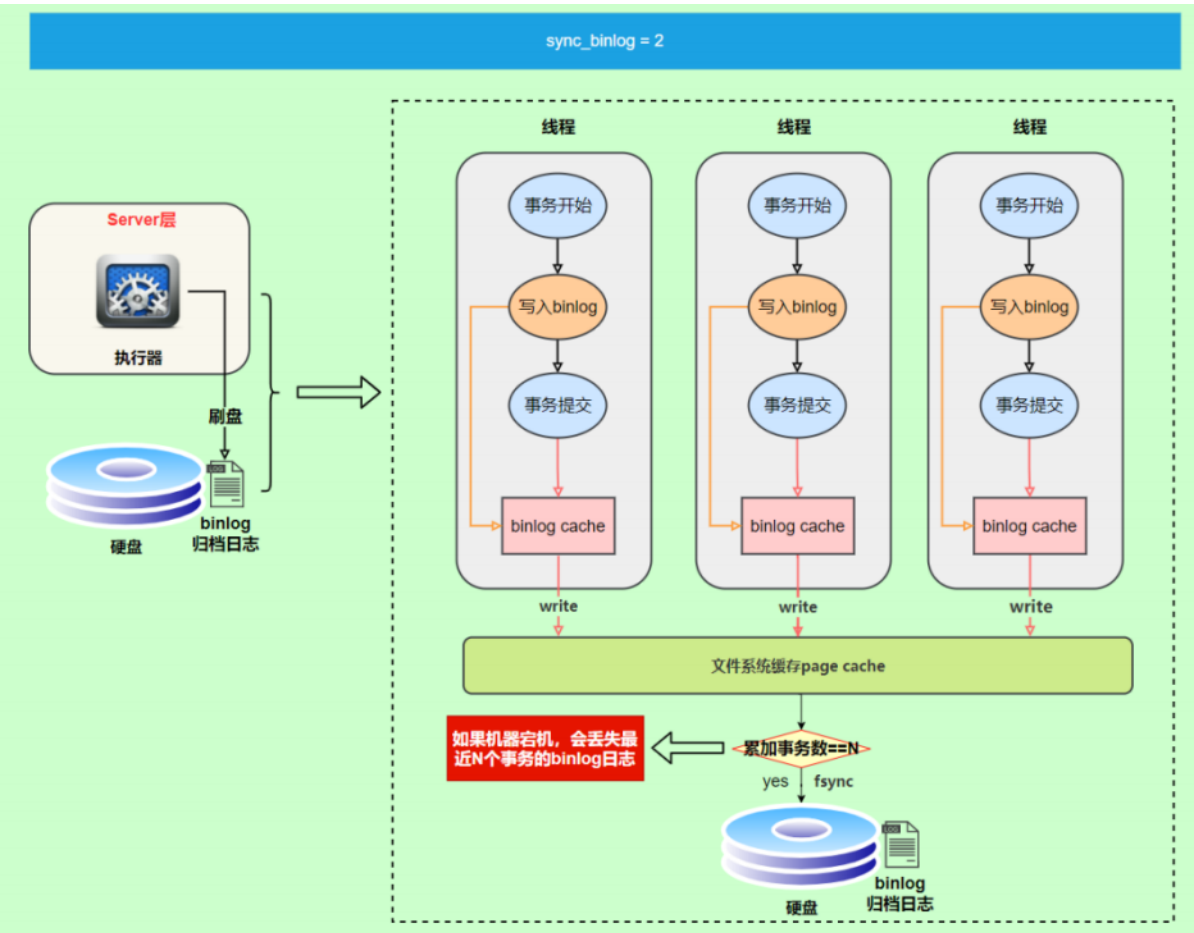
- 上图的 write，是指把日志写入到文件系统的 page cache，并没有把数据持久化到磁盘，所以速度比较快
- 上图的 fsync，才是将数据持久化到磁盘的操作，其是操作系统层面，由操作系统觉得什么时候将内容写到磁盘中的binlog文件中。

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 binglog 会丢失（即mysql事务提交后，将binlog日志写入到了文件系统缓存中后，操作系统宕机会出现page cache里面的binlog丢失。对于Mysql未提交事务，不存在binlog的丢失的情况，因为事务没有提交，binlog也没有写到文件系统缓存page cache中）。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（太频繁，影响性能，因为要进行磁盘的IO）。

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。



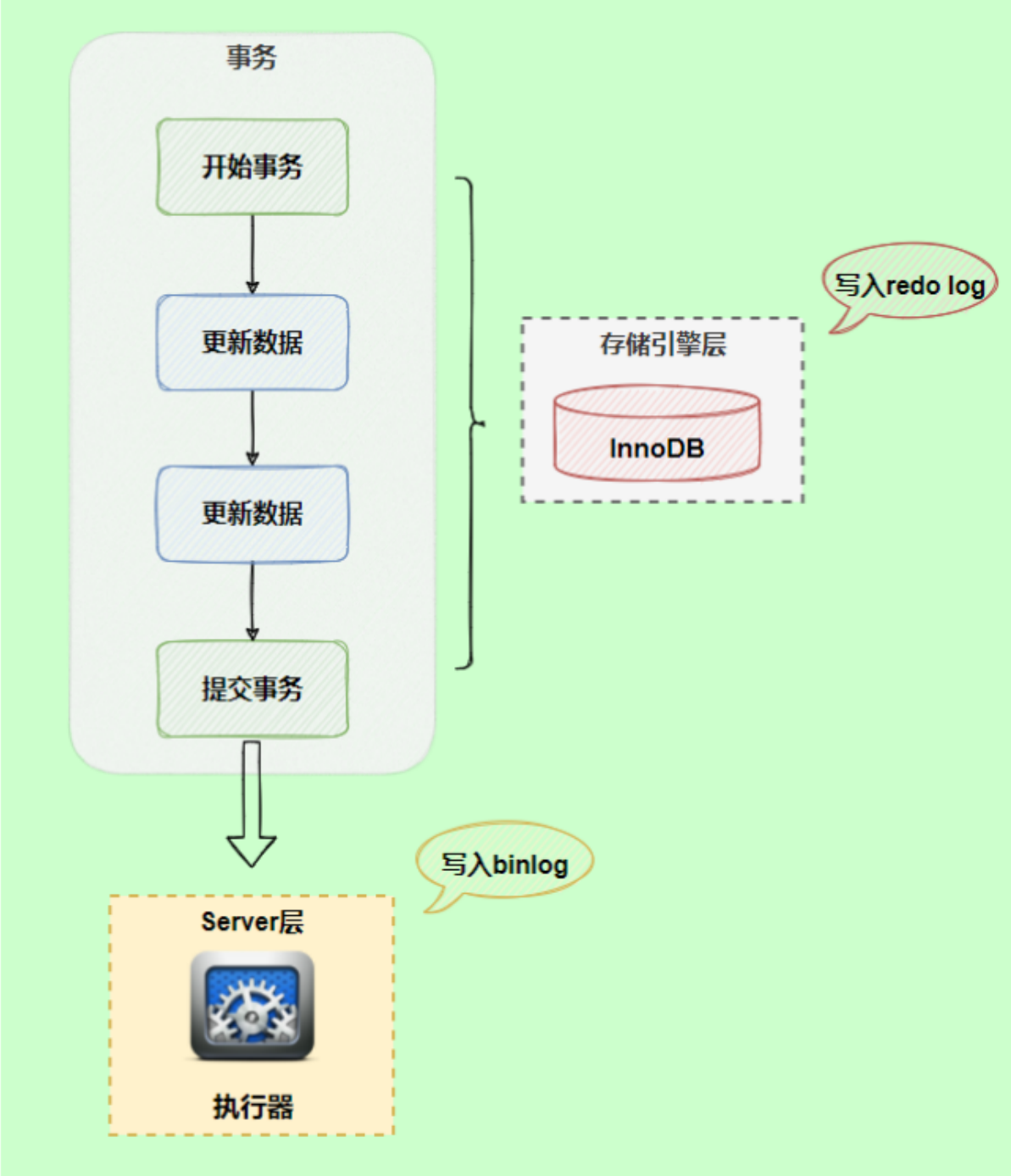
在出现IO瓶颈的场景里，将sync_binlog设置成一个比较大的值，可以提升性能。同样的，如果机器宕机，会丢失最近N个事务的binlog日志。

5.5 关于redolog和binlog

- 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 MySQL Server 层。
- 虽然它们都属于持久化的保证，但是则重点不同，
    - 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。

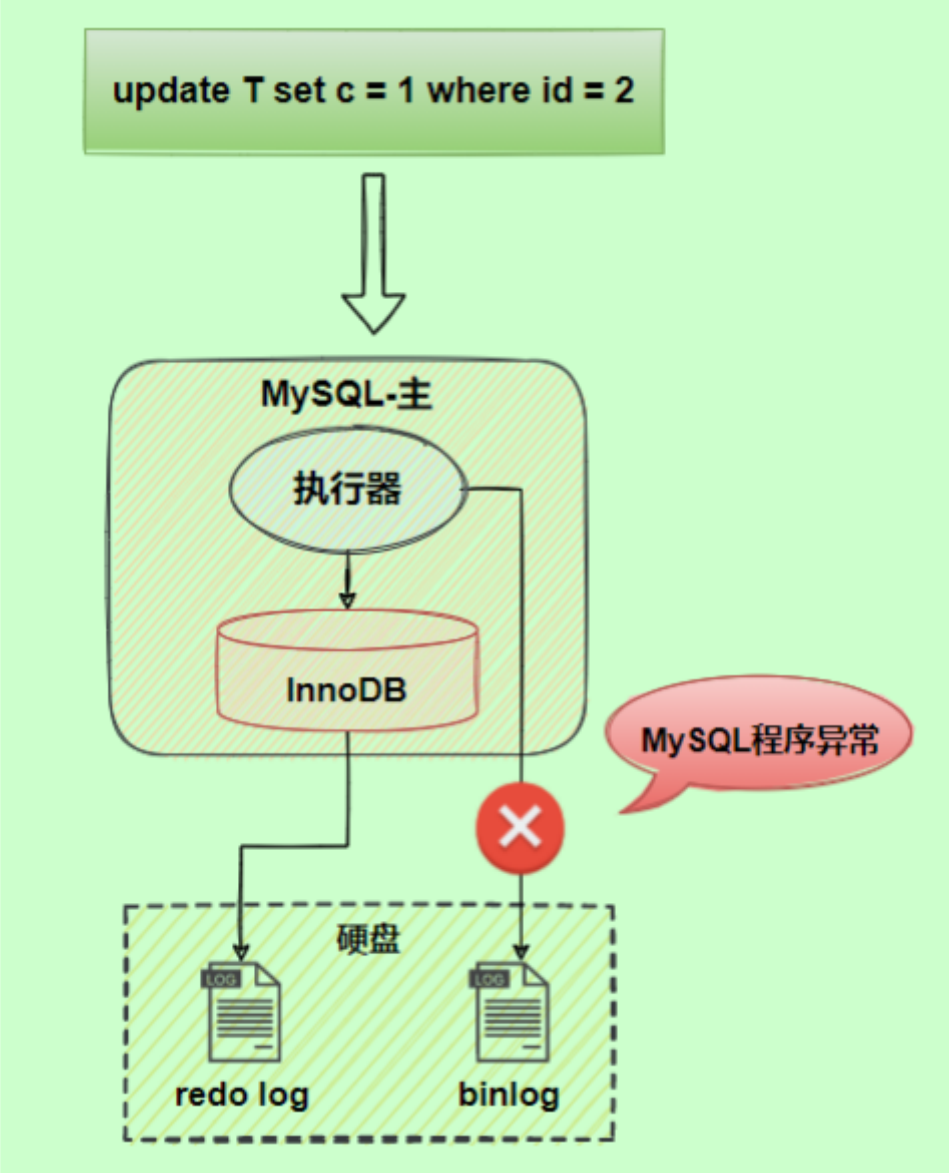
5.6 两阶段提交

在执行更新语句过程，会记录redo log与binlog两块日志，以基本的事务为单位，redo log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redo log与binlog的 写入时机 不一样。

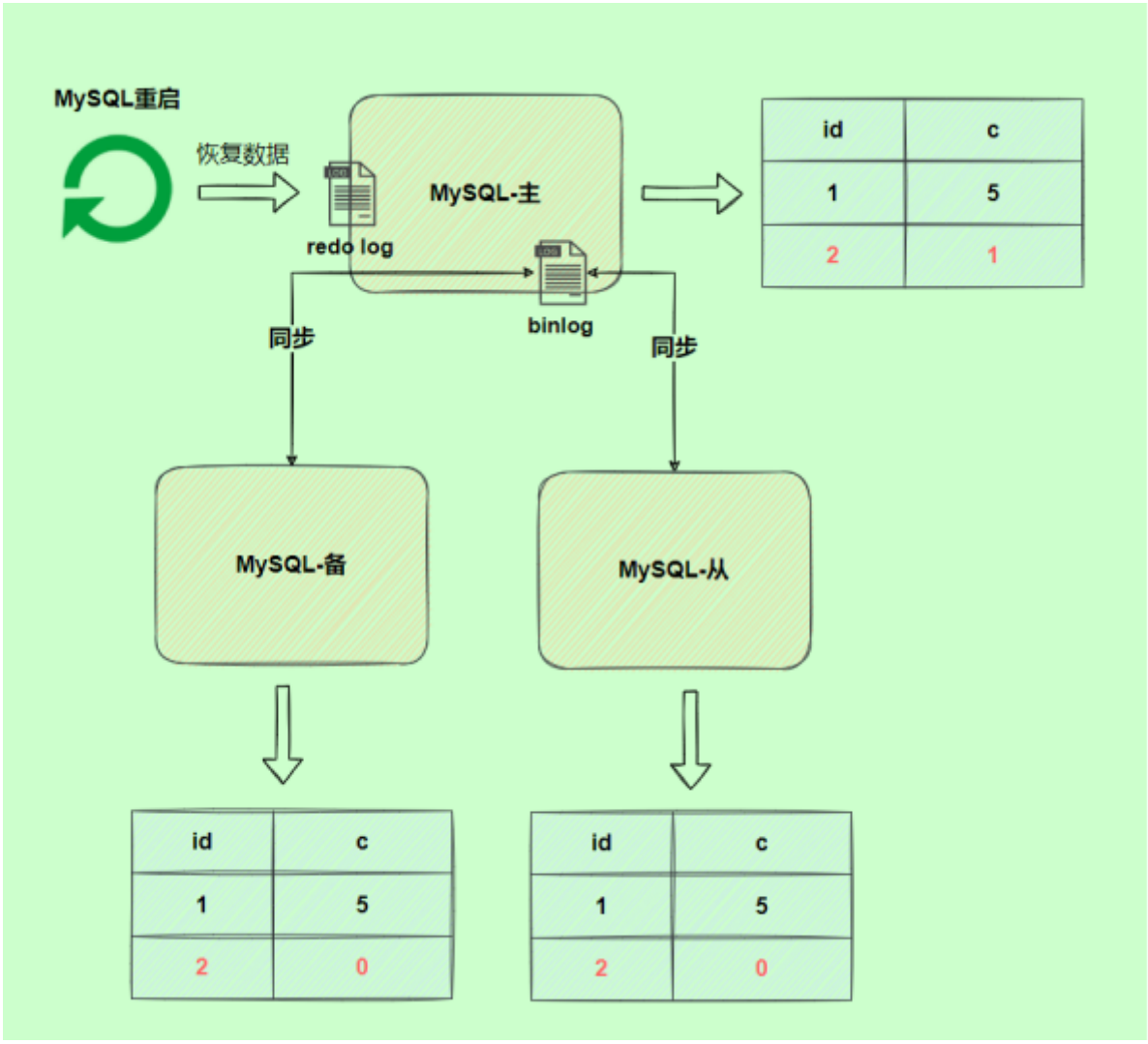


redo log与binlog两份日志之间的逻辑不一致，会出现什么问题?

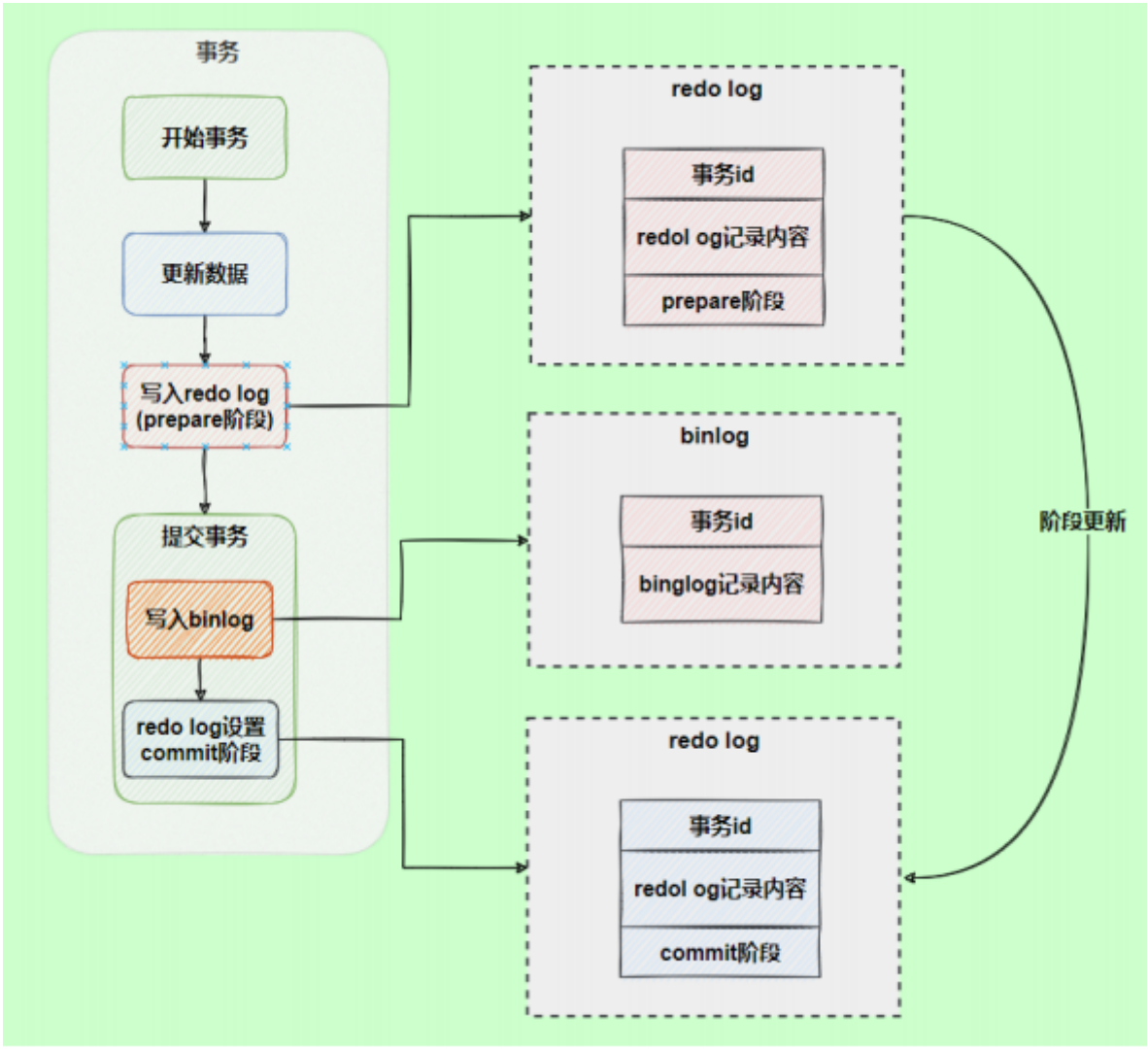
以update语句为例，假设 id=2的记录，字段c值是0，把字段c值更新成1，SQL语句为update T set c=1 where id=2。
假设执行过程中写完redo log日志后，binlog日志写期间发生了异常，会出现什么情况呢?



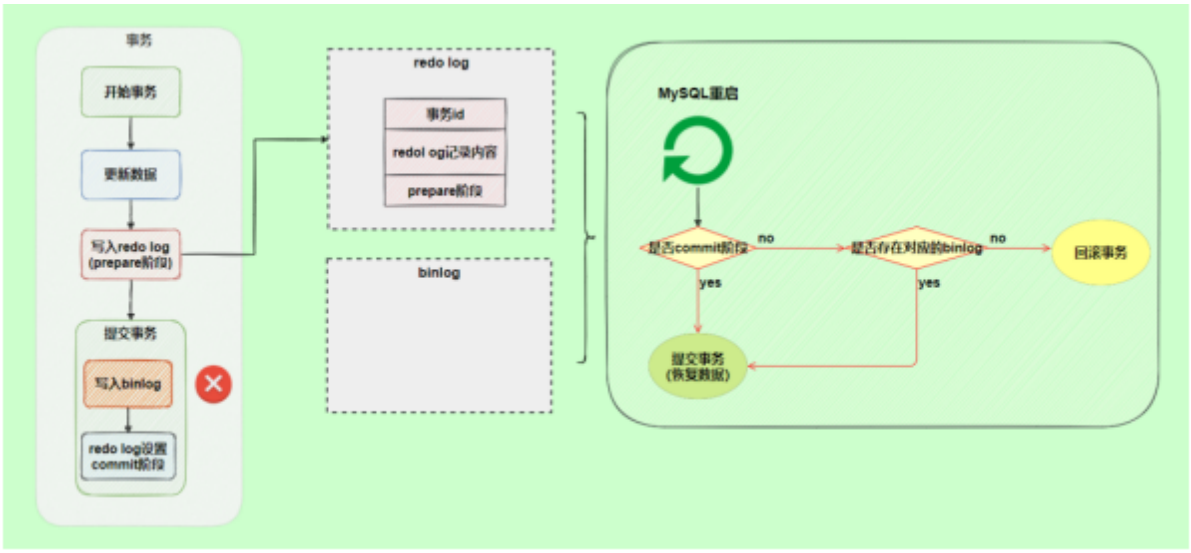
此时redo log 写入成功，表示事务已经持久化，但binlog没写完就异常，这时候binlog里面没有对应的修改记录。因此，之后用binlog日志恢复数据时（从库），就会少这一次更新，恢复出来的这一行c值是0，而原库（主库）因为redo log日志恢复，这一行c值是1，最终数据不一致



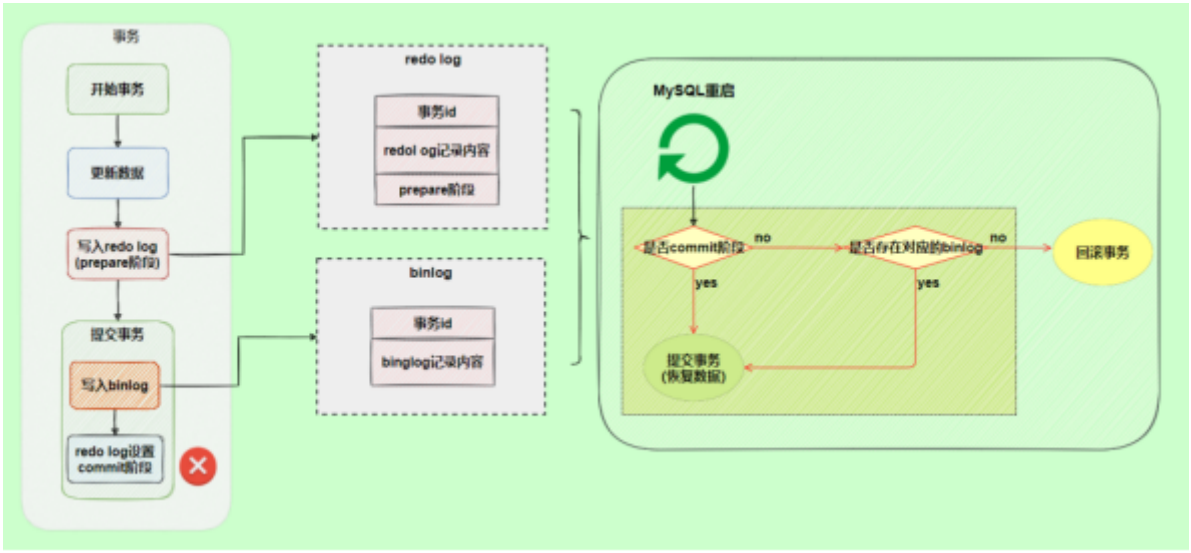
为了解决两份日志之间的逻辑一致问题，InnoDB存储引擎使用**两阶段提交**方案。原理很简单，就爱那个redo log的写入折成两个步骤prepare和commit，这就是两阶段提交（相当于多了一个判断的环节，判断binlog是否写入）



使用**两阶段提交**后，写入binlog时发生异常也不会有影响，因为MSOL根据redo log日志恢复数据时，发现redo log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，redo log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redo log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。

六、中继日志（relay log）

6.1 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。

6.2 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=
CD95YCABAAAAKAAAAGgDAAAAAFsAAAAAAAEAAgAB/wABAAAAfATkBw==
'/*!*/;
# at 1040
```

这一段的意思是，主服务器（“server id 1”）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 872。
```

6.3 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。

七、重做日志（△）

八、撤销日志（△）