Part 17 其他数据库日志

1. MySQL支持的日志

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。
- 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

- 降低数据库性能
- 占用大量磁盘空间

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 举例

某电商平台中，用户使用微信支付时，网络系统出错，钱已经汇出，但记录没有保存。用户转而用支付宝支付，系统留下记录。事后，用户反馈付款两次，我们通过通用查询日志发现用户在用微信支付时网络故障，而后等到微信的回调通知发送到平台，发现已经有一条该订单的记录，只能覆盖。因此该订单只有一条付款记录。

从上述例子我们知道通用查询日志的用处。

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)
```

3.3 启动日志

- 方式一：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。

- 方式二：临时性方式

  ```
  # 开启通用查询日志
  SET GLOBAL general_log=on;
  # 设置日志文件保存位置
  SET GLOBAL general_log_file='path/filename';

  # 关闭通用查询日志
  SET GLOBAL general_log=off;
  # 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Mac OSX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。

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

3.5.1 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。

3.5.2 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。

```
# 通用查询日志的目录，然后可以手动删除
SHOW VARIABLES LIKE 'general_log%';

# 删除之后要重新生成查询日志文件，一定要开启通用日志
mysqladmin -uroot -p flush-logs
```

4. 错误日志(error log)

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。

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。

```
[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.'
```

官网提示补充操作：

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

5. 二进制日志(bin log)

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）。

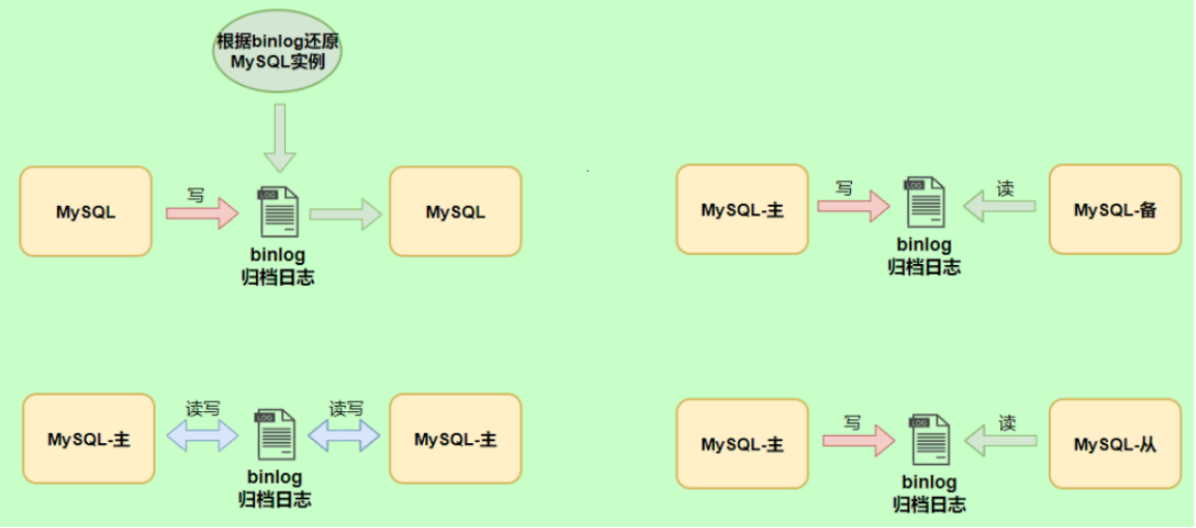
它以`事件形式`记录并保存在`二进制文件`中。通过这些信息，我们可以再现数据更新操作的全过程。

> 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。

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。

```
show variables like '%log_bin%';
```

5.2 日志参数设置

5.2.1 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
```

- 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；86400 1天；259200 3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：

```
show variables like '%log_bin%';
```

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
```

> 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。

5.2.2 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mysql 8 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 ;

# session级别
SET sql_log_bin= 0 ;
```

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
SHOW BINARY LOGS;
```

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，借助mysqlbinlog命令工具了。

```
# 查看二进制日志文件内容
mysqlbinlog  "/var/lib/mysql/lqhdb-binlog.000001"

# 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'
```

另一种查询命令

```
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）

```
# 将指定的binlog日志文件，分成有效事件行的方式返回
show binlog events in 'atguigu-bin.000002';

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

#指定查询mysql-bin.088802这个文件
show binlog events in 'atguigu-bin. 008002'\G;

#指定查询mysql-bin. 080802这个文件，从pos点:391开始查起:
show binlog events in 'atguigu-bin.008802' from 391\G;

#指定查询mysql-bin.000802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.000882' from 391 limit 5\G

#指定查询 mysql-bin.880002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.088882' from 391 limit 2,5\G;
```

binlog格式查看，一共三种格式：

```
show variables like 'binlog_format';
```

- 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。

5.4 使用日志恢复数据

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。

```
#可以生成新的binLog文件，不然这个文件边恢复边变大是不行的。
flush logs;
# 显示有哪些binLog 文件
show binary logs;
```

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。

5.5.1 PURGE MASTER LOGS：删除指定日志文件

```
# 删除指定文件名之前的所有日志文件
PURGE {MASTER | BINARY} LOGS TO ‘指定日志文件名’
# 删除指定日期之前的所有日志文件
PURGE {MASTER | BINARY} LOGS BEFORE ‘指定日期’
```

5.5.2 RESET MASTER:删除所有二进制日志文件

```
reset master;
```

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。

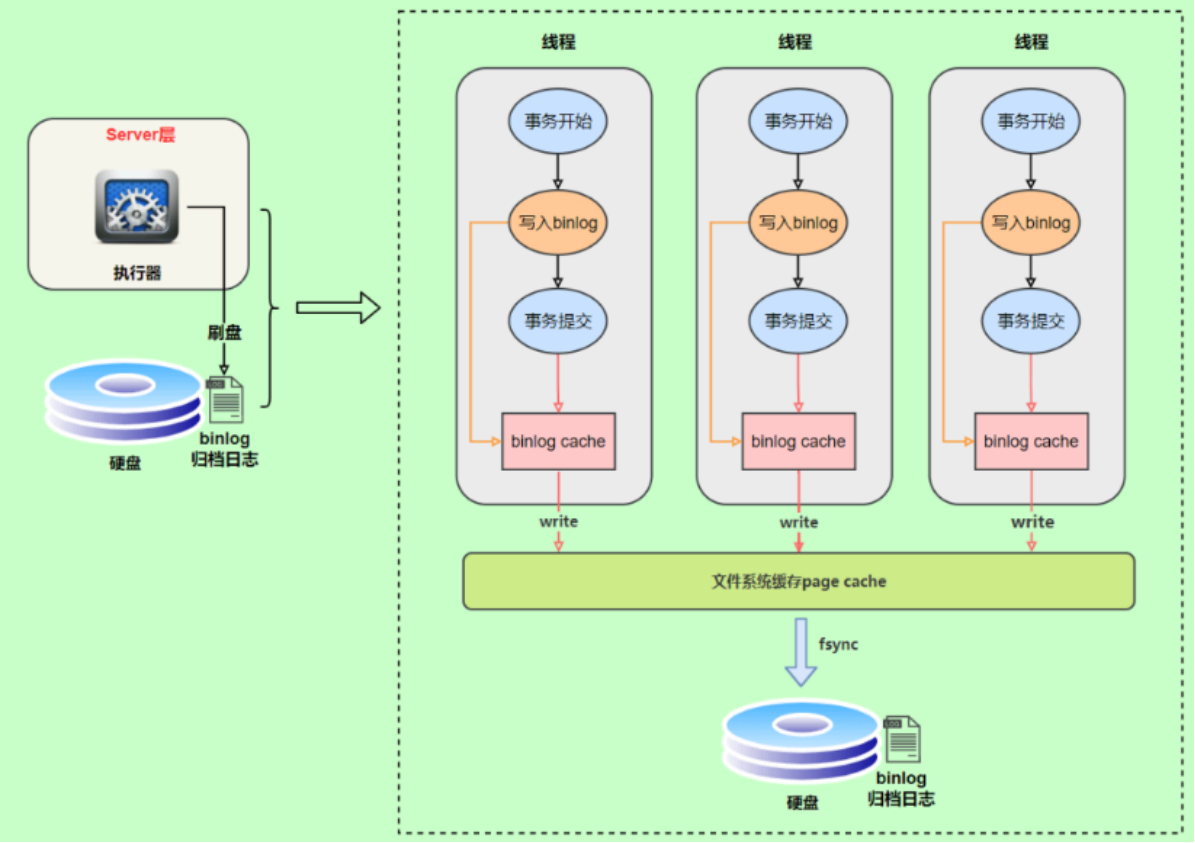
在这种情况下，一个有效的解决办法是`配置主从数据库服务器`，甚至是`一主多从`的架构，把二进制日志文件的内容通过中继日志，同步到从数据库服务器中，这样就可以有效避免数据库故障导致的数据异常等问题。

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binlog cache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，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：



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。
- 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。

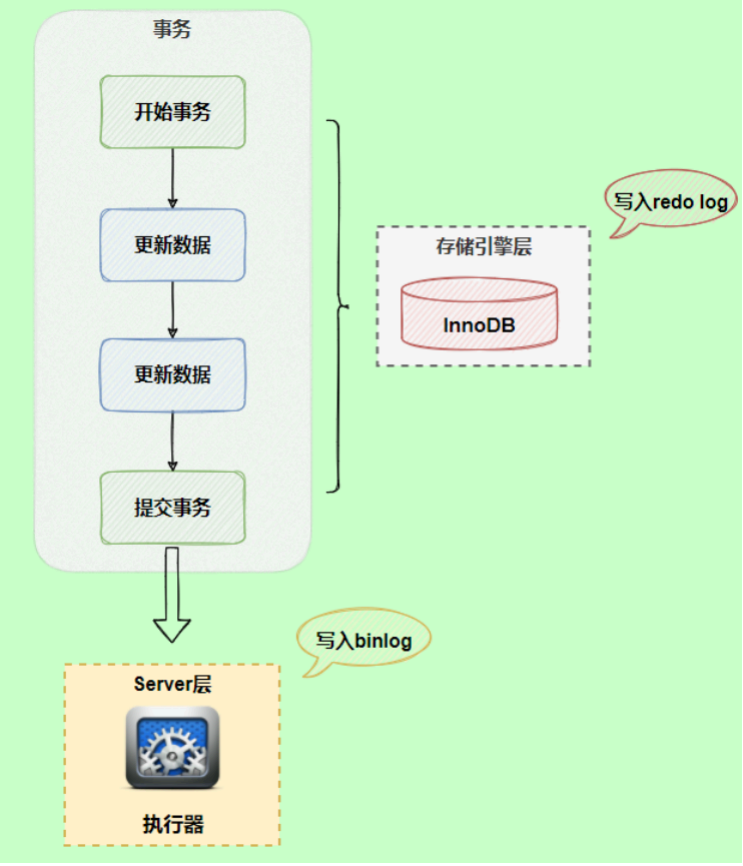
6.2 binlog与redolog对比

- 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。

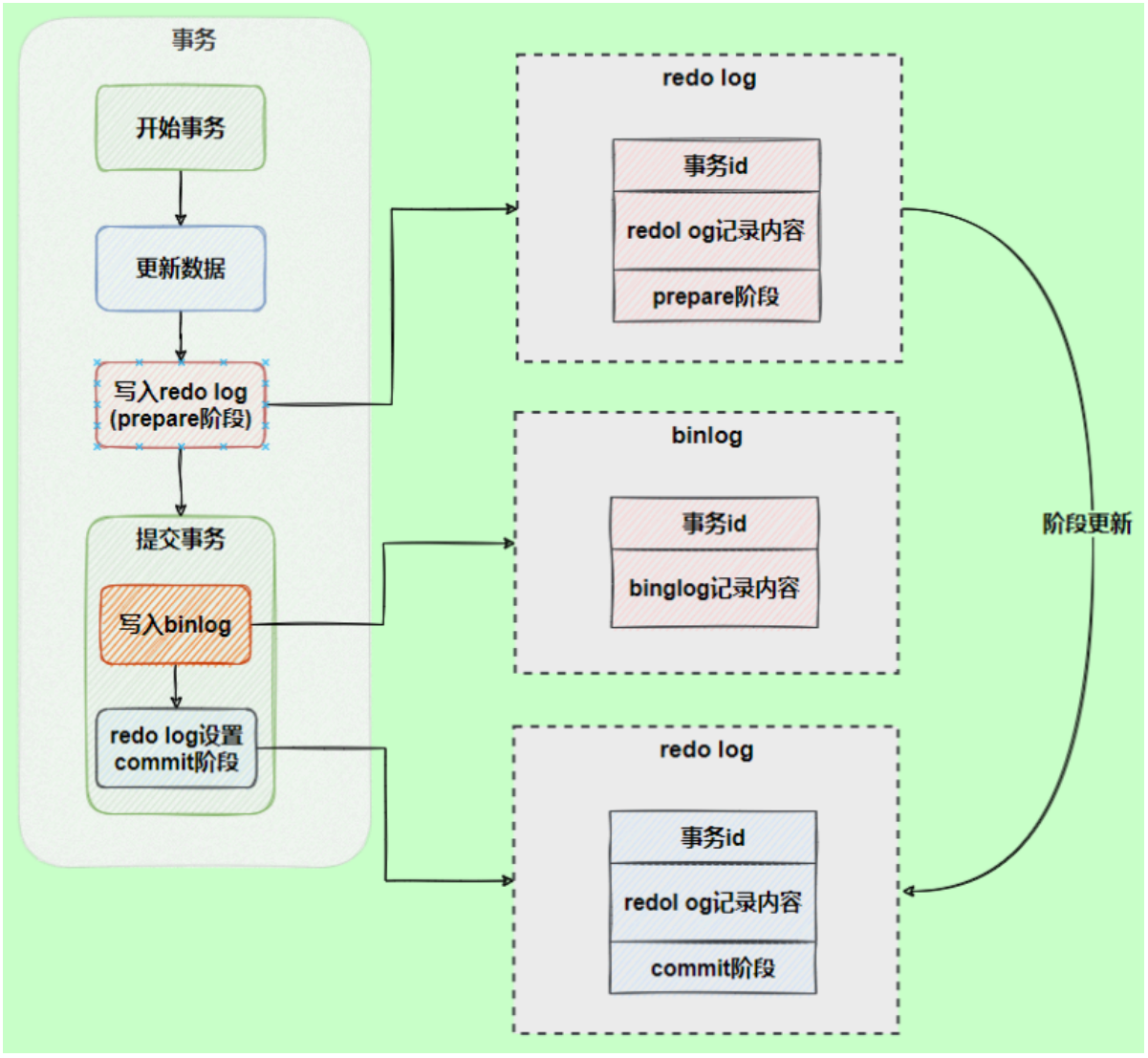
6.3 两阶段提交

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。

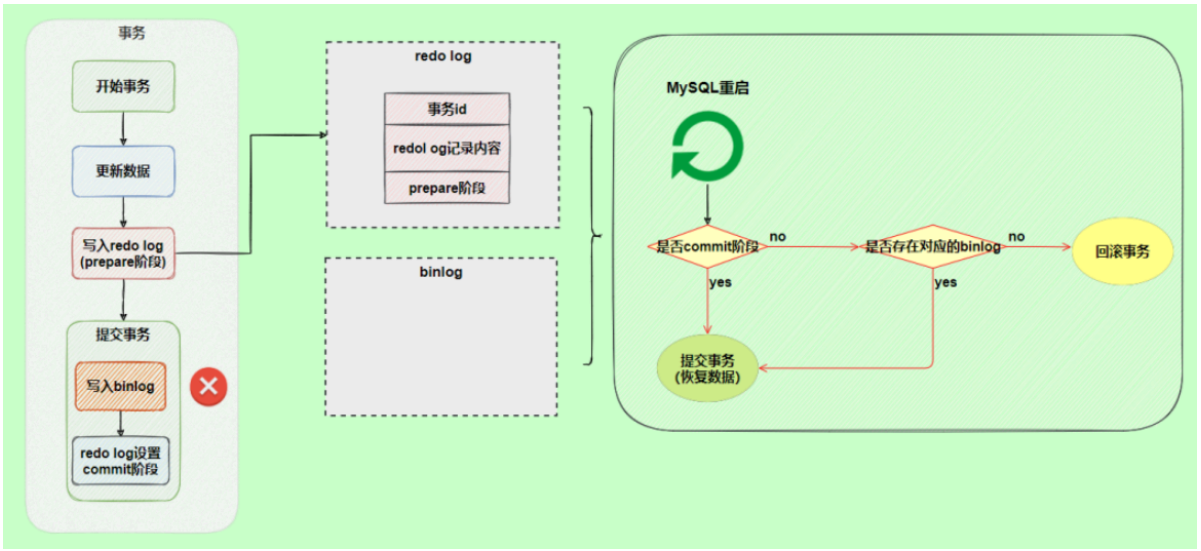


redo log与binlog两份日志之间的逻辑不一致，会出现什么问题?

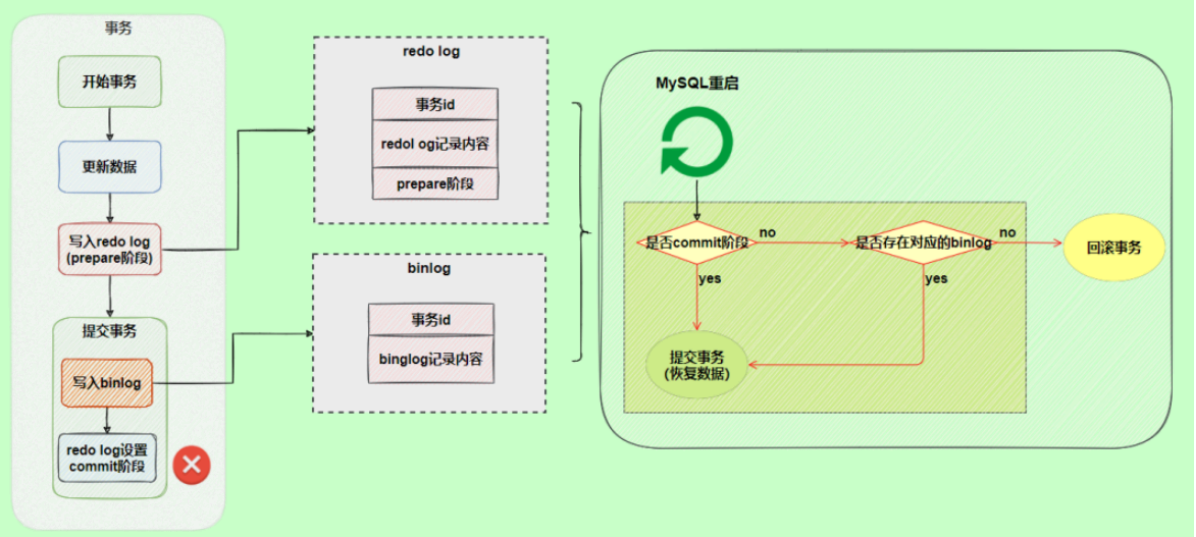
为了解决两份日志之间的逻辑一致问题，InnoDB存储引擎使用**两阶段提交**方案。原理很简单，将redo log的写入拆成了两个步骤prepare和commit，这就是**两阶段提交。**



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，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。

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。

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。