

# 04\_MySQL数据写入过程 + MVCC + 三种日志

时间：2023年1月18日17:38:35

建议的配置文件：

- [02\\_MySQL建议的配置文件](#)
- [在线翻译\\_有道](#)

## 一、MySQL三种日志（InnoDB）

### 1. 官方文档介绍

- UndoLog
  - <https://dev.mysql.com/doc/refman/5.7/en/innodb-undo-logs.html>
- RedoLog (MySQL5.7+InnoDB)
  - <https://dev.mysql.com/doc/refman/5.7/en/innodb-redo-log.html>

重做日志是一种基于磁盘的数据结构，用于在崩溃恢复期间纠正未完成事务写入的数据。在正常操作期间，重做日志会对由SQL语句或低级API调用引起的更改表数据的请求进行编码。在初始化期间和接受连接之前，在意外关闭之前未完成更新数据文件的修改将自动重放。

默认情况下，重做日志在磁盘上由ib\_logfile0和ib\_logfile1两个文件物理表示。MySQL以循环的方式写入重做日志文件。重做日志中的数据按照受影响的记录进行编码；这些数据统称为重做。数据在重做日志中的传递用一个不断增加的LSN值表示。

- Binlog
  - <https://dev.mysql.com/doc/refman/5.7/en/binary-log.html>
  - Binlog有三个作用
    - ① 为数据库高可用提供主从数据同步支持
    - ② 参与处理数据库宕机数据恢复过程
    - ③ 用于用户手动恢复数据
  - 使用binlog恢复数据
    - <https://dev.mysql.com/doc/refman/5.7/en/point-in-time-recovery-binlog.html>
    - <https://dev.mysql.com/doc/refman/5.7/en/innodb-recovery.html>

### 2. UndoLog

撤销表空间包含撤销日志，撤销日志是记录的集合，其中包含关于如何撤销事务对聚集索引记录的最新更改的信息。

```
-- ./
SHOW VARIABLES LIKE '%innodb_undo_directory%';
-- 16384
show variables like 'innodb_page_size';
-- 128
SHOW VARIABLES LIKE '%innodb_rollback_segments%';
-- 2
SHOW VARIABLES LIKE '%innodb_undo_tablespaces%';
```

- Undo TableSpace (属于系统表空间)
  - <https://dev.mysql.com/doc/refman/8.0/en/innodb-undo-tablespaces.html>
    - `innodb_undo_directory=.`

默认的undo表空间创建在`innodb_undo_directory`变量定义的位置上。如果`innodb_undo_directory`变量未定义，则在`data`目录中创建默认的undo表空间。缺省的undo表空间数据文件名为“undo\_002”。对应数据字典中定义的undo表空间名称为`innodb_undo_001`和`innodb_undo_002`。

- 从MySQL 8.0.14开始，可以在运行时使用SQL创建额外的undo表空间。

#### 移动Undo表空间

使用`CREATE Undo TABLESPACE`语法创建的Undo表空间可以在服务器脱机时移动到任何目录。由`innodb_directories`变量定义的目录。由`innodb_data_home_dir`、`innodb_data_home_dir`和`datadir`定义的目录会自动添加到`innodb_directories`值中，而不管`innodb_directories`是否定义。在启动时扫描这些目录及其子目录以查找undo表空间文件。一个被移动到这些目录的文件在启动时被发现，并被假定为被移动的撤销表空间文件。

初始化MySQL实例时创建的默认undo表空间(`innodb_undo_001`和`innodb_undo_002`)位于`innodb_undo_directory`变量定义的目录中。如果`innodb_undo_directory`变量未定义，则默认undo表空间位于`data`目录。如果默认的undo表空间在服务器脱机时被移动，服务器启动时必须将`innodb_undo_directory`变量配置为新目录。

- undo日志的I/O模式使undo表空间成为SSD存储的良好候选者。

### 删除Undo表空间

从MySQL 8.0.14开始，使用CREATE undo TABLESPACE语法创建的undo表空间可以在运行时使用DROP undo TABALESPACE语法删除。

undo表空间在被删除之前必须为空。要清空undo表空间，必须首先使用ALTER undo tablespace语法将undo表空间标记为非活动的，以便该表空间不再用于为新事务分配回滚段。

### 撤销表空间

在撤销表空间被标记为非活动后，当前使用撤销表空间中的回滚段的事务被允许结束，在这些事务完成之前启动的任何事务也被允许结束。事务完成后，清除系统释放undo表空间中的回滚段，undo表空间被截断为初始大小。(截断undo表空间时使用相同的过程。参见截断撤销表空间。)一旦undo表空间为空，就可以删除它。

### 删除UNDO表空间

#### 请注意

或者，undo表空间可以保持空状态，如果需要，稍后通过发出ALTER undo tablespace tablespace\_name SET ACTIVE语句重新激活。

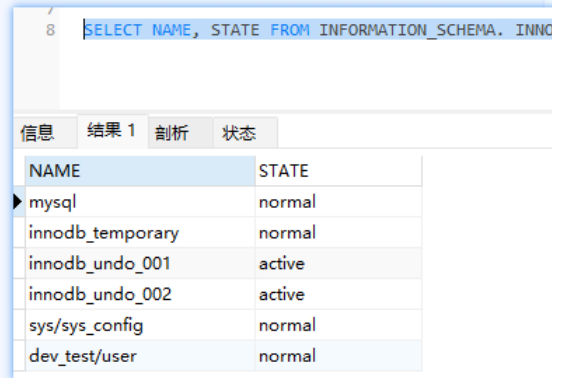
undo表空间的状态可以通过查询Information Schema INNODB\_TABLESPACES表来监控。

```
SELECT NAME, STATE FROM INFORMATION_SCHEMA.INNODB_TABLESPACES  
WHERE NAME LIKE 'tablespace_name';
```

未激活状态表示撤销表空间中的回滚段不再被新事务使用。空状态表示undo表空间是空的，可以被删除，或者可以使用ALTER undo tablespace tablespace\_name SET active语句再次激活。试图删除非空的undo表空间将返回错误。

MySQL实例初始化时默认创建的undo表空间(innodb\_undo\_001和innodb\_undo\_002)不能删除。但是，可以使用ALTER UNDO TABLESPACE tablespace\_name SET inactive语句使其变为非活动状态。在一个默认的撤销表空间被取消活动之前，必须有一个撤销表空间来代替它。在任何时候都需要至少两个活动的undo表空间来支持自动截断undo表空间。

以上翻译结果来自有道神经网络翻译 (YNMT) - 通用领域



```
8 SELECT NAME, STATE FROM INFORMATION_SCHEMA.INNODB_TABLESPACES  
WHERE NAME LIKE 'innodb_undo_%';
```

| 信息 | 结果 1             | 剖析 | 状态     |
|----|------------------|----|--------|
|    | NAME             |    | STATE  |
|    | mysql            |    | normal |
|    | innodb_temporary |    | normal |
|    | innodb_undo_001  |    | active |
|    | innodb_undo_002  |    | active |
|    | sys/sys_config   |    | normal |
|    | dev_test/user    |    | normal |



☐ 逐句对照

```
[root@datax mysql]# ll
总用量 98M
-rw-r-----. 1 polkitd input 56 1月 17 19:40 auto.cnf
-rw-r-----. 1 polkitd input 2.9M 1月 17 19:40 binlog.000001
-rw-r-----. 1 polkitd input 912 1月 17 20:48 binlog.000002
-rw-r-----. 1 polkitd input 32 1月 17 19:40 binlog.index
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 ca-key.pem
-rw-r-----. 1 polkitd input 1.1K 1月 17 19:40 ca.pem
-rw-r-----. 1 polkitd input 1.1K 1月 17 19:40 client-cert.pem
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 client-key.pem
drwxr-x---. 2 polkitd input 21 1月 17 19:40 dev_test
-rw-r-----. 1 polkitd input 192K 1月 17 20:48 #ib_16384_0.dblwr
-rw-r-----. 1 polkitd input 8.2M 1月 17 19:40 #ib_16384_1.dblwr
-rw-r-----. 1 polkitd input 5.5K 1月 17 19:40 ib_buffer_pool
-rw-r-----. 1 polkitd input 12M 1月 17 20:48 ibdata1
-rw-r-----. 1 polkitd input 12M 1月 17 19:40 ibtmp1
drwxr-x---. 2 polkitd input 4.0K 1月 17 19:40 #innodb_redo
drwxr-x---. 2 polkitd input 187 1月 17 19:40 #innodb_temp
drwxr-x---. 2 polkitd input 143 1月 17 19:40 mysql
-rw-r-----. 1 polkitd input 30M 1月 17 20:48 mysql.ibd
lrwxrwxrwx. 1 polkitd input 27 1月 17 19:40 mysql.sock ->
drwxr-x---. 2 polkitd input 8.0K 1月 17 19:40 performance_schema
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 private_key.pem
-rw-r-----. 1 polkitd input 452 1月 17 19:40 public_key.pem
-rw-r-----. 1 polkitd input 1.1K 1月 17 19:40 server-cert.pem
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 server-key.pem
drwxr-x---. 2 polkitd input 28 1月 17 19:40 sys
-rw-r-----. 1 polkitd input 16M 1月 17 20:48 undo_001
-rw-r-----. 1 polkitd input 16M 1月 17 20:48 undo_002

[root@datax mysql]# pwd
/data/mysql
[root@datax mysql]# cd dev_test/
[root@datax dev_test]# ll
总用量 112K
-rw-r-----. 1 polkitd input 112K 1月 17 20:48 xxx.ibd

[root@datax dev_test]#
```

ibdata中原先存储着所有表  
↓  
表具体数据被单独提取到具体  
↓  
undolog被单独提取收到具体  
↓  
8.0.20后 double write buf

undolog专属文件

innodb\_file\_per\_table=1 (默认值)

```
14 -- 2
15 SHOW VARIABLES LIKE '%innodb'
信息 结果 1 剖析 状态
Variable_name
innodb_undo_tablespaces
```

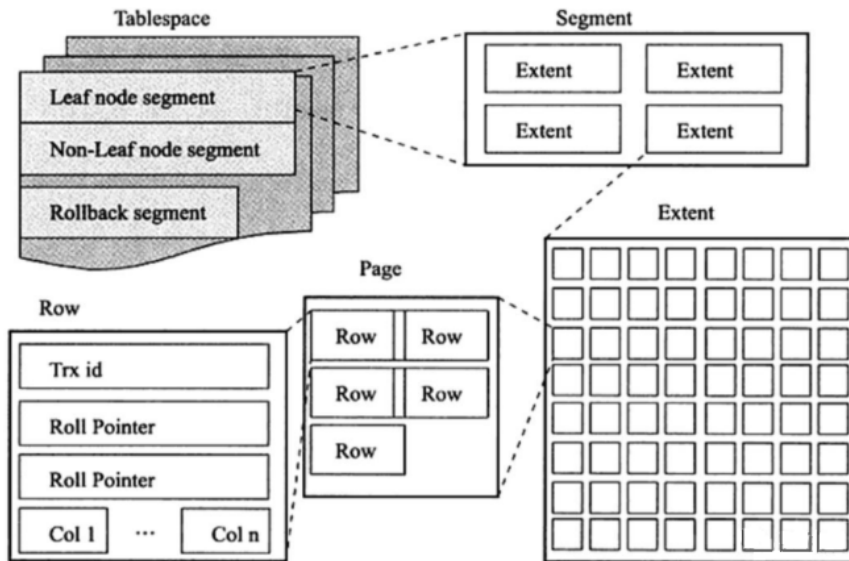
■ 表空间大小 (问题: 表空间是什么?)

在MySQL 8.0.23之前, undo表空间的初始大小取决于innodb\_page\_size的值。默认页时, 初始undo表空间文件大小为10MiB。当页面大小为4KB、8KB、32KB、64KB时, 文件大小分别为7MiB、8MiB、20MiB、40MiB。从MySQL 8.0.23开始, 初始undo表16MiB。通过截断操作创建新的undo表空间时, 初始大小可能不同。在这种情况下, 如小大于16MB, 并且上一个文件扩展名发生在最后一秒内, 那么新的undo表空间将以innodb\_max\_undo\_log\_size变量定义的大小的四分之一创建。

➔ + 自动扩展 +

在MySQL 8.0.23之前, undo表空间一次扩展四个区段。从MySQL 8.0.23开始, undo:了16MB。为了处理快速增长, 如果前一个文件扩展名发生在0.1秒之前, 则文件扩展名倍。扩展大小可以翻倍多次, 最大达到256MB。如果先前的文件扩展名发生在0.1秒之前减少一半(也可以出现多次), 最小为16MB。如果AUTOEXTEND\_SIZE选项是为undo表:么它将被AUTOEXTEND\_SIZE设置和由上面描述的逻辑确定的扩展大小中的较大值所扩。AUTOEXTEND\_SIZE选项的信息, 请参见第15.6.3.9节 “表空间AUTOEXTEND\_SIZE配

● 表空间是什么?



TableSpace  
Segment  
Page  
Row

数据的具体存储格式

```
[root@datax mysql]# ll
总用量 98M
-rw-r----- 1 polkitd input 56 1月 17 19:40 auto.cnf
-rw-r----- 1 polkitd input 2.9M 1月 17 19:40 binlog.000001
-rw-r----- 1 polkitd input 912 1月 17 20:48 binlog.000002
-rw-r----- 1 polkitd input 32 1月 17 19:40 binlog.index
-rw-r----- 1 polkitd input 1.7K 1月 17 19:40 ca-key.pem
-rw-r----- 1 polkitd input 1.1K 1月 17 19:40 ca.pem
-rw-r----- 1 polkitd input 1.7K 1月 17 19:40 client-cert.pem
-rw-r----- 1 polkitd input 1.7K 1月 17 19:40 client-key.pem
drwxr-x--- 2 polkitd input 21 1月 17 19:40 dev_test
-rw-r----- 1 polkitd input 192K 1月 17 20:48 #ib_16384_0.dblwr
-rw-r----- 1 polkitd input 8.2M 1月 17 19:40 #ib_16384_1.dblwr
-rw-r----- 1 polkitd input 5.5K 1月 17 19:40 ib_buffer_pool
-rw-r----- 1 polkitd input 12M 1月 17 20:48 ibdata1
-rw-r----- 1 polkitd input 12M 1月 17 19:40 ibtmp1
drwxr-x--- 2 polkitd input 4.0K 1月 17 19:40 #innodb_redo
drwxr-x--- 2 polkitd input 187 1月 17 19:40 #innodb_temp
drwxr-x--- 2 polkitd input 143 1月 17 19:40 mysql
-rw-r----- 1 polkitd input 36M 1月 17 20:48 mysql.ibd
lrwxrwxrwx 1 polkitd input 27 1月 17 19:40 mysql.sock -> /var/run/mysql/mysql.sock
drwxr-x--- 2 polkitd input 8.0K 1月 17 19:40 performance_schema
-rw-r----- 1 polkitd input 1.7K 1月 17 19:40 private_key.pem
-rw-r----- 1 polkitd input 452 1月 17 19:40 public_key.pem
-rw-r----- 1 polkitd input 1.1K 1月 17 19:40 server-cert.pem
-rw-r----- 1 polkitd input 1.7K 1月 17 19:40 server-key.pem
drwxr-x--- 2 polkitd input 28 1月 17 19:40 sys
-rw-r----- 1 polkitd input 16M 1月 17 20:48 undo_001
-rw-r----- 1 polkitd input 16M 1月 17 20:48 undo_002
[root@datax mysql]# pwd
/data/mysql
[root@datax mysql]#
```

```
7  -- ibdata1:12M:autoextend
8
9  show variables like 'innodb_data_file_path';
--
Variable_name | Value
innodb_data_file_path | ibdata1:12M:autoextend
```

```

[root@datax mysql]#
[root@datax mysql]# ll
总用量 98M
-rw-r-----. 1 polkitd input 56 1月 17 19:40 auto.cnf
-rw-r-----. 1 polkitd input 2.9M 1月 17 19:40 binlog.000001
-rw-r-----. 1 polkitd input 912 1月 17 20:48 binlog.000002
-rw-r-----. 1 polkitd input 32 1月 17 19:40 binlog.index
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 ca-key.pem
-rw-r-----. 1 polkitd input 1.1K 1月 17 19:40 ca.pem
-rw-r-----. 1 polkitd input 1.1K 1月 17 19:40 client-cert.pem
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 client-key.pem
drwxr-x---. 2 polkitd input 21 1月 17 19:40 dev_test
-rw-r-----. 1 polkitd input 192K 1月 17 20:48 #ib_16384_0.dblwr
-rw-r-----. 1 polkitd input 8.2M 1月 17 19:40 #ib_16384_1.dblwr
-rw-r-----. 1 polkitd input 5.5K 1月 17 19:40 ib_buffer_pool
-rw-r-----. 1 polkitd input 12M 1月 17 20:48 ibdata1
-rw-r-----. 1 polkitd input 12M 1月 17 19:40 ibtmp1
drwxr-x---. 2 polkitd input 4.0K 1月 17 19:40 #innodb_redo
drwxr-x---. 2 polkitd input 187 1月 17 19:40 #innodb_temp
drwxr-x---. 2 polkitd input 143 1月 17 19:40 mysql
-rw-r-----. 1 polkitd input 30M 1月 17 20:48 mysql.ibd
lrwxrwxrwx. 1 polkitd input 27 1月 17 19:40 mysql.sock ->
drwxr-x---. 2 polkitd input 8.0K 1月 17 19:40 performance_schema
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 private_key.pem
-rw-r-----. 1 polkitd input 452 1月 17 19:40 public_key.pem
-rw-r-----. 1 polkitd input 1.1K 1月 17 19:40 server-cert.pem
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 server-key.pem
drwxr-x---. 2 polkitd input 28 1月 17 19:40 sys
-rw-r-----. 1 polkitd input 16M 1月 17 20:48 undo_001
-rw-r-----. 1 polkitd input 16M 1月 17 20:48 undo_002
[root@datax mysql]# pwd
/data/mysql
[root@datax mysql]# cd dev_test/
[root@datax dev_test]# ll
总用量 112K
-rw-r-----. 1 polkitd input 112K 1月 17 20:48 xxx.ibd
[root@datax dev_test]#

```

临时表数据文件

```

8 -- ibdata1:12M:autoextend
9 show variables like 'innodb_data_file_path';
10
11 -- ibtmp1:12M:autoextend
12 show variables like 'innodb_temp_data_file_path';

```

| Variable_name              | Value                 |
|----------------------------|-----------------------|
| innodb_temp_data_file_path | ibtmp1:12M:autoextend |

- innodb\_file\_per\_table=1, 所有各个表数据将会从ibdata1中单独提取出来存储在自己的表文件中, 否则所有数据都会存在ibdata1中。

- MySQL5.6+都是开启的

```

[root@datax mysql]# pwd
/data/mysql
[root@datax mysql]# cd dev_test/
[root@datax dev_test]# ll
总用量 112K
-rw-r-----. 1 polkitd input 112K 1月 17 20:48 xxx.ibd
[root@datax dev_test]#

```

数据库名

表名.idb (innodbData)

xxx.ibd

## 二、MySQL数据写入过程

### 1. 写入过程

- 事务开始
- 解析SQL将相关表数据加载到BufferPool
- 记录数据旧值+计算后得出的数据新值写入undo log
  - 增加操作: 没有旧值, 新值将会直接写入undo log, 如果事务正常提交, 该值作为下一次事务链中的旧值
  - 更新操作: 记录旧值, 将新记录写入undolog, 将新记录的回滚指针指向旧记录
  - 删除操作: 记录旧值, 同时更新旧值记录header部分的删除标记
  - 提示: 事务回滚时, 会删除事务链中的当前事务的所有版本, 原有事务链指向会被更新
- 将数据的具体更改记录到redo log buffer中, 标记当前事务为运行状态
  - 可根据参数innodb\_flush\_log\_at\_trx\_commit调节日志落盘时机 (默认=1, 事务提交, 同步落盘, 如果=0, 1s刷一次)
- 修改BufferPool中的数据
  - 异步执行刷写任务: 当某个Page修改/新增的数据超过一定阈值后, 会被刷写到硬盘中



- Double write机制，解决InnoDB页大小和OS页大小不同可能带来的写入异常，数据库重启失败的问题

- 增加操作：数据在BufferPool中添加
- 更新操作：数据在BufferPool中更新指定记录
- 删除操作：数据在BufferPool中为指定记录添加删除标记（刷新到磁盘后，会由指定Purge对待删除数据进行定期清理）

## ● 事务提交

- 把记录的binlog日志追加写入硬盘binlog文件，一次事务binlog提交必须完全写入binlog文件后，下一个事务才允许写入。

- 对比binlog的一次性写入，redo log则不同，redo log允许多事务并发写入具体对数据值的修改。

因为二进制日志只在提交的时候一次性写入，所以二进制日志中的记录方式和提交顺序有关，且一次提交对记录。而redo log中是记录的物理页的修改，redo log file中同一个事务可能多次记录，最后一个提交的事务记录有未提交的事务记录。例如事务T1，可能在redo log中记录了T1-1,T1-2,T1-3, T1\* 共4个操作，其中T1\*表示时的日志记录，所以对应的数据页最终状态是T1\* 对应的操作结果。而且redo log是并发写入的，不同事务之

- 本的记录会穿插写入到redo log文件中，例如可能redo log的记录方式如下：T1-1,T1-2,T2-1,T2-2,T2,T1-3,T1

- 可根据参数sync\_binlog调节多少次事务执行之后，binlog落盘（默认=1，事务提交，同步落盘）

- binlog写入硬盘完成后，会再次更新Redo Log，更新当前事务状态为已完成，默认此时缓冲区日志开始同步写入硬盘。

- 默认innodb\_flush\_log\_at\_trx\_commit=1，此时开始同步落盘redo log缓冲区数据。

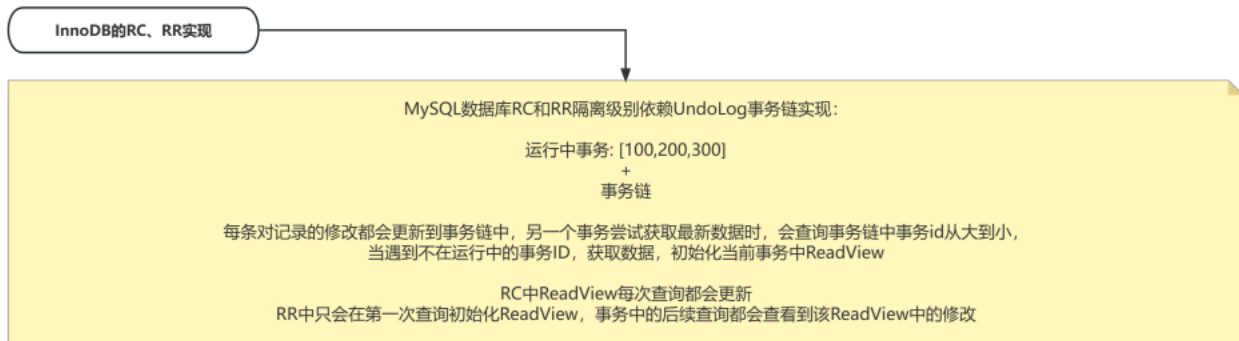
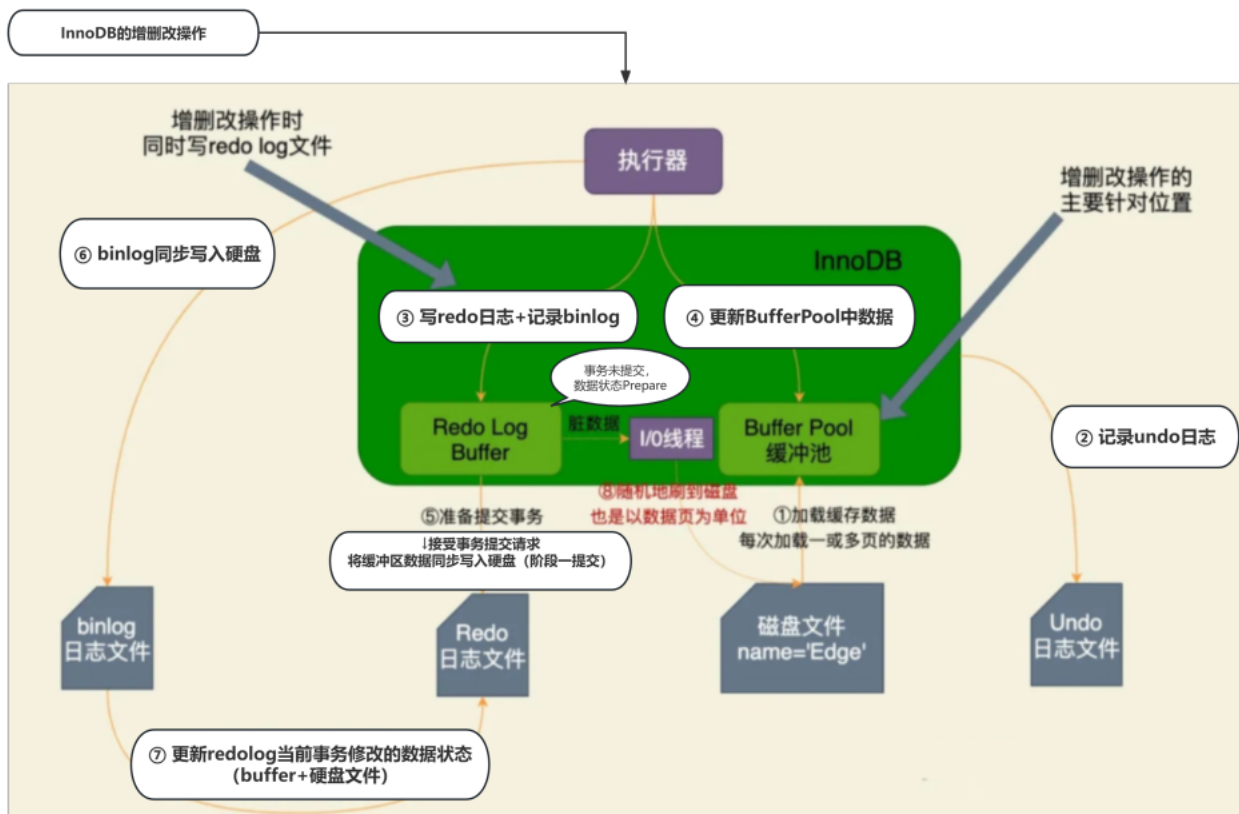
- binlog写入硬盘后，redo log写入硬盘后；

## ● 事务提交结束

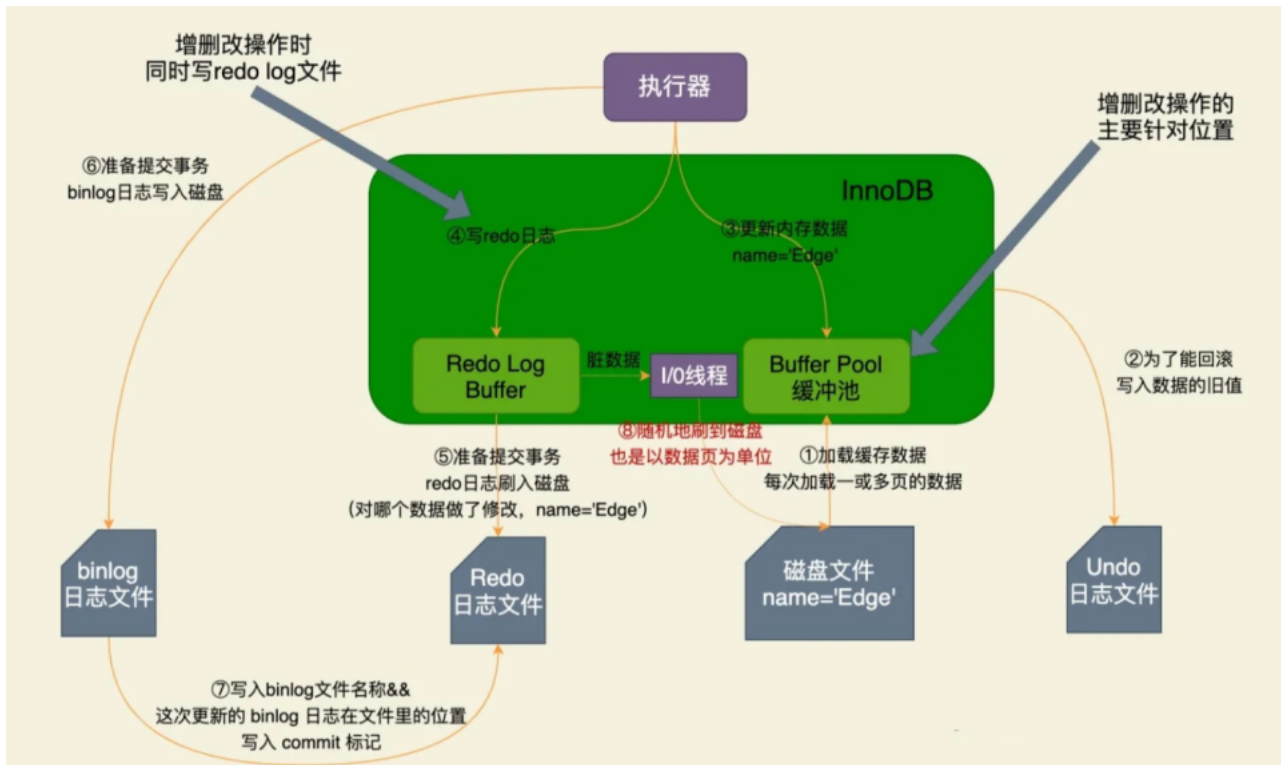
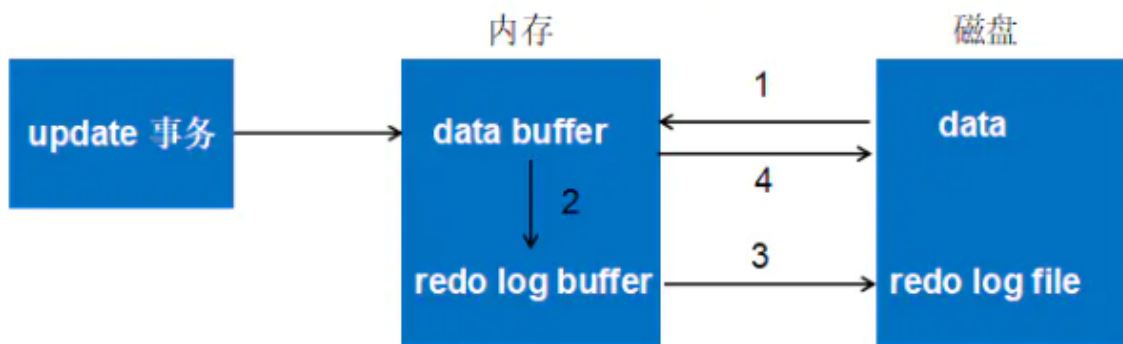
```
Every 1.0s: ls -alh          Wed Jan 18 15:33:05 2023
总用量 114M
dwar-x-x. 0 polkit root  4.0K 1月  18 15:30 .
dwar-x-x. 0 root      56 1月  18 15:30 .
-rw-r----- 1 polkit input 56 1月  18 14:46 auto.cnf
-rw-r----- 1 polkit input 2.9K 1月  18 14:46 binlog.000001
-rw-r----- 1 polkit input 1.5K 1月  18 15:10 binlog.000002
-rw-r----- 1 polkit input 180 1月  18 15:30 binlog.000003
-rw-r----- 1 polkit input 157 1月  18 15:30 binlog.000004
-rw-r----- 1 polkit input 64 1月  18 15:30 binlog.index
-rw-r----- 1 polkit input 1.7K 1月  18 14:46 ca-key.pem
-rw-r----- 1 polkit input 1.1K 1月  18 14:46 ca.pem
-rw-r----- 1 polkit input 1.1K 1月  18 14:46 client-cert.pem
-rw-r----- 1 polkit input 1.7K 1月  18 14:46 client-key.pem
dwar-x-x. 2 polkit input 22 1月  18 14:46 dev.txt
-rw-r----- 1 polkit input 102K 1月  18 15:33 #sh_16384_0_dblwr
-rw-r----- 1 polkit input 8.2M 1月  18 15:33 #sh_16384_1_dblwr
-rw-r----- 1 polkit input 10K 1月  18 15:30 ib_buffer_pool
-rw-r----- 1 polkit input 12M 1月  18 15:31 iddata1
-rw-r----- 1 polkit input 12M 1月  18 15:30 idtmp1
dwar-x-x. 2 polkit input 4.5K 1月  18 15:33 innodb_redo
dwar-x-x. 2 polkit input 187 1月  18 15:30 innodb_temp
-rw-r----- 1 polkit input 143 1月  18 14:46 mysql
-rw-r----- 1 polkit input 50K 1月  18 15:30 mysql.ibd
lnxxxxxx. 1 polkit input 27 1月  18 15:30 mysql.sock -> /var/run/mysqld/mysqld.sock
dwar-x-x. 2 polkit input 8.0K 1月  18 14:46 performance_schema
-rw-r----- 1 polkit input 1.7K 1月  18 14:46 private_key.pem
-rw-r----- 1 polkit input 452 1月  18 14:46 public_key.pem
-rw-r----- 1 polkit input 1.1K 1月  18 14:46 server-cert.pem
-rw-r----- 1 polkit input 1.7K 1月  18 14:46 server-key.pem
dwar-x-x. 2 polkit input 20 1月  18 14:46 sys
-rw-r----- 1 polkit input 16K 1月  18 15:33 undo.001
-rw-r----- 1 polkit input 32K 1月  18 15:33 undo.002
```

```
Every 1.0s: ls -alh          Wed Jan 18 15:33:05 2023
总用量 805M
dwar-x-x. 2 polkit input  22 1月  18 14:46 .
dwar-x-x. 0 polkit root  4.0K 1月  18 15:30 .
-rw-r----- 1 polkit input 804K 1月  18 15:33 user.ibd

一个事务修改的数据会在OS空闲时刷写到硬盘中，即使事务没有提交（数据异步落盘，InnoDB使用的是WAL：Write Ahead Logging）
↓
事务如果回滚，会根据undo log找到需要回滚的数据，其中新增的数据不会立刻删除，会被purge system定期清理
↓
事务如果提交，undo log更新当前事务状态为已完成即可（数据已经被异步刷新到硬盘中）
↓
undo log必须要在BufferPool中数据修改前记录并完成落盘
```

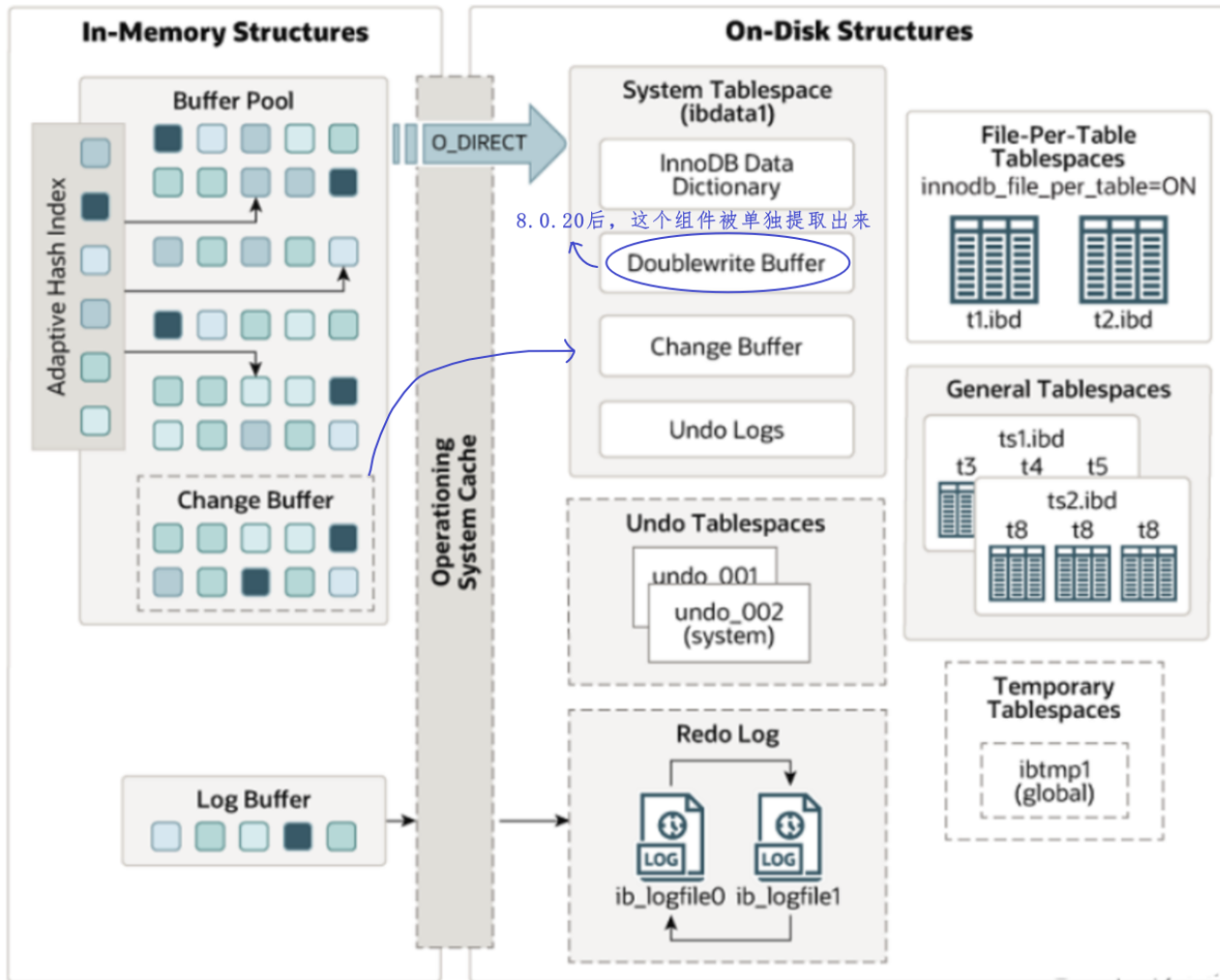






## 2. InnoDB存储引擎

- <https://dev.mysql.com/doc/refman/5.7/en/innodb-storage-engine.html>



### 3. BufferPool中的ChangeBuffer

如果一条update语句是位于普通索引（非唯一索引）上的修改，比方说要修改page500上的某条数据，那么该条修改记录会先记录到change buffer中，而不是产生缺页中断去磁盘上加载新page500。如果在之后的过程中，有新的查询需要加载该page500，当page500加载到buffer pool时，会与change buffer中的相关记录做一次合并操作。

这样可以明显节省一次IO过程，有利于写性能的提高。

```
[root@datax mysql]# ll
总用量 98M
-rw-r-----. 1 polkitd input 56 1月 17 19:40 auto.cnf
-rw-r-----. 1 polkitd input 2.9M 1月 17 19:40 binlog.000001
-rw-r-----. 1 polkitd input 912 1月 17 20:48 binlog.000002
-rw-r-----. 1 polkitd input 32 1月 17 19:40 binlog.index
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 ca-key.pem
-rw-r-----. 1 polkitd input 1.1K 1月 17 19:40 ca.pem
-rw-r-----. 1 polkitd input 1.1K 1月 17 19:40 client-cert.pem
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 client-key.pem
drwxr-x---. 2 polkitd input 21 1月 17 19:40 dev_test
-rw-r-----. 1 polkitd input 192K 1月 17 20:48 #ib_16384_0.dblwr
-rw-r-----. 1 polkitd input 8.2M 1月 17 19:40 #ib_16384_1.dblwr
-rw-r-----. 1 polkitd input 5.5K 1月 17 19:40 ib_buffer_pool
-rw-r-----. 1 polkitd input 12M 1月 17 20:48 ibdata1
-rw-r-----. 1 polkitd input 12M 1月 17 19:40 ibtmp1
drwxr-x---. 2 polkitd input 4.0K 1月 17 19:40 #innodb_redo
drwxr-x---. 2 polkitd input 187 1月 17 19:40 #innodb_temp
drwxr-x---. 2 polkitd input 143 1月 17 19:40 mysql
-rw-r-----. 1 polkitd input 30M 1月 17 20:48 mysql.ibd
lrwxrwxrwx. 1 polkitd input 27 1月 17 19:40 mysql.sock -> /var/run/mysqld/mysqld.sock
drwxr-x---. 2 polkitd input 8.0K 1月 17 19:40 performance_schema
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 private_key.pem
-rw-r-----. 1 polkitd input 452 1月 17 19:40 public_key.pem
-rw-r-----. 1 polkitd input 1.1K 1月 17 19:40 server-cert.pem
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 server-key.pem
drwxr-x---. 2 polkitd input 28 1月 17 19:40 sys
-rw-r-----. 1 polkitd input 16M 1月 17 20:48 undo_001
-rw-r-----. 1 polkitd input 16M 1月 17 20:48 undo_002
[root@datax mysql]#
```

服务关闭前，会将buffer\_pool持久化到本地

## 4. DoubleWrite机制

- 默认开启: innodb\_doublewrite参数控制

Mysql是以page为单位从磁盘上获取数据加载到内存或者从内存中写入磁盘的, page的默认大小为16KB, 而目前并不是所有的磁盘都能支持16KB的原子性写入的(很多磁盘的块大小是4KB), 所以会导致一种情况, Mysql向磁盘中写入一个page时, 如果发生意外, 磁盘上就会存在一个损坏的page, 这将导致Mysql在下次重启过程中, 无法恢复该页数据, 导致数据丢失。

所以buffer pool中的脏页在写入磁盘前, 会先写入double write buffer(会持久化到共享表空间), 这样在磁盘上的page发生缺失时, 还可以根据double write中的副本页进行恢复。

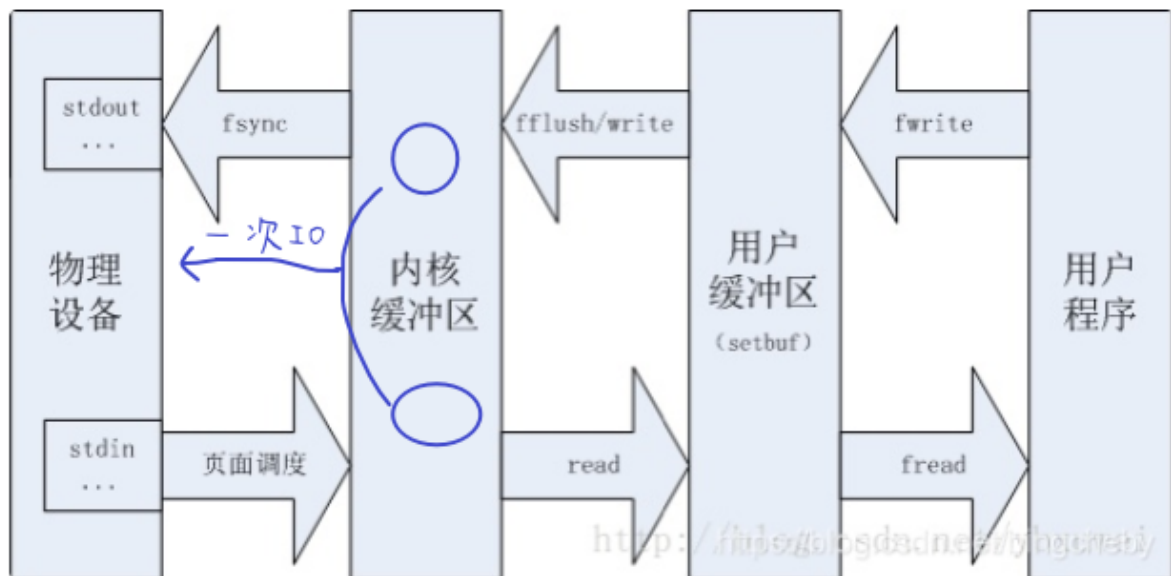
如果double write持久化时Mysql挂了, 那么Mysql会根据旧数据 + redo log 进行数据恢复。

**问题一:** 缓冲区的数据需要写入到两个文件中(double write文件+具体数据文件), 不是会多出一个磁盘IO过程吗?  
(不会)

doublewrite缓冲区是一个存储区域, InnoDB将从缓冲池中刷新的页面写入到InnoDB数据文件中相应的位置。如果有一个操作系统, 存储子系统, 或者意外的mysqld进程在写页面的过程中退出, InnoDB可以在崩溃恢复时从doublewrite缓冲区中找到一个好的页面副本。

虽然数据写入两次, 但doublewrite缓冲区并不需要两倍的I/O开销或两倍的I/O操作。数据以一个大的顺序块写入doublewrite缓冲区, 对操作系统进行一次fsync()调用(除非innodb\_flush\_method被设置为O\_DIRECT\_NO\_FSYNC)。

在MySQL 8.0.20之前, doublewrite缓冲区存储区域位于InnoDB系统表空间中。从MySQL 8.0.20开始, doublewrite缓冲区存储区域位于doublewrite文件中。



- DoubleWriteBuffer (硬盘中的文件)
  - <https://dev.mysql.com/doc/refman/8.0/en/innodb-doublewrite-buffer.html>

```
[root@datax mysql]# ll
总用量 98M
-rw-r-----. 1 polkitd input 56 1月 17 19:40 auto.cnf
-rw-r-----. 1 polkitd input 2.9M 1月 17 19:40 binlog.000001
-rw-r-----. 1 polkitd input 912 1月 17 20:48 binlog.000002
-rw-r-----. 1 polkitd input 32 1月 17 19:40 binlog.index
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 ca-key.pem
-rw-r-----. 1 polkitd input 1.1K 1月 17 19:40 ca.pem
-rw-r-----. 1 polkitd input 1.1K 1月 17 19:40 client-cert.pem
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 client-key.pem
drwxr-x---. 2 polkitd input 21 1月 17 19:40 dev_test
-rw-r-----. 1 polkitd input 192K 1月 17 20:48 #ib_16384_0.dblwr
-rw-r-----. 1 polkitd input 8.2M 1月 17 19:40 #ib_16384_1.dblwr
-rw-r-----. 1 polkitd input 5.5K 1月 17 19:40 ib_buffer_pool
-rw-r-----. 1 polkitd input 12M 1月 17 20:48 ibdata1
-rw-r-----. 1 polkitd input 12M 1月 17 19:40 ibtmp1
drwxr-x---. 2 polkitd input 4.0K 1月 17 19:40 #innodb_redo
drwxr-x---. 2 polkitd input 187 1月 17 19:40 #innodb_temp
drwxr-x---. 2 polkitd input 143 1月 17 19:40 mysql
-rw-r-----. 1 polkitd input 30M 1月 17 20:48 mysql.ibd
lrwxrwxrwx. 1 polkitd input 27 1月 17 19:40 mysql.sock -> /var/run/mysql/mysql.sock
drwxr-x---. 2 polkitd input 8.0K 1月 17 19:40 performance_schema
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 private_key.pem
-rw-r-----. 1 polkitd input 452 1月 17 19:40 public_key.pem
-rw-r-----. 1 polkitd input 1.1K 1月 17 19:40 server-cert.pem
-rw-r-----. 1 polkitd input 1.7K 1月 17 19:40 server-key.pem
drwxr-x---. 2 polkitd input 28 1月 17 19:40 sys
-rw-r-----. 1 polkitd input 16M 1月 17 20:48 undo_001
-rw-r-----. 1 polkitd input 16M 1月 17 20:48 undo_002
[root@datax mysql]#
```

8.0.20之后，DWBuffer被单独提取出提供单独的参数控制

8.0.20之前，DWBuffer是在ibdata中的一块连续区域

#### innodb\_doublewrite=1/ON

innodb\_doublewrite变量控制doublewrite缓冲区是否启用。在大多数情况下默认启用doublewrite缓冲区，将innodb\_doublewrite设置为OFF。如果您更关心性能而不是数考虑禁用doublewrite缓冲区，例如，在执行基准测试时可能会出现这种情况。

从MySQL 8.0.30开始，innodb\_doublewrite支持DETECT\_AND\_RECOVER和DETECT\_ONLY

DETECT\_AND\_RECOVER设置与ON设置相同。使用此设置，doublewrite缓冲区将完全内容将写入doublewrite缓冲区，恢复期间将访问该缓冲区以修复不完整的页写入。

使用DETECT\_ONLY设置，只有元数据被写入doublewrite缓冲区。数据库页内容不会写缓冲区，恢复也不会使用doublewrite缓冲区来修复不完整的页写入。此轻量级设置仅用页面写入。

MySQL 8.0.30以后支持动态修改innodb\_doublewrite设置，在ON、DETECT\_AND\_RECOVER、DETECT\_ONLY之间启用doublewrite缓冲区。MySQL不支持在启用doublewrite缓冲区间动态更改，反之亦然。

#### innodb\_doublewrite\_dir=空

innodb\_doublewrite\_dir变量(在MySQL 8.0.20中引入)定义了InnoDB创建doublewrite文件。如果不指定目录，则在innodb\_data\_home\_dir目录下创建doublewrite文件，如果未指定data目录。

哈希符号“#”会自动添加到指定的目录名前，以避免与模式名冲突。然而，如果a'。在目录名中显式地指定了'/'前缀，哈希符号“#”没有作为目录名的前缀。

理想情况下，doublewrite目录应该放在可用的最快的存储介质上。

#### innodb\_doublewrite\_files=2

`innodb_doublewrite_files`变量定义了doublewrite文件的数量。默认情况下，为每个缓冲池实例创建一个doublewrite文件。每个doublewrite文件包含一个flush列表doublewrite文件和一个LRU列表doublewrite文件。

刷新列表双写文件用于从缓冲池刷新列表刷新的页面。flush list doublewrite文件的默认页面大小\* doublewrite页面字节。

LRU列表双写文件用于从缓冲池LRU列表刷新页面。它还包含用于单页刷新的插槽。LRU列表双写文件的默认大小是InnoDB页大小\*(双写页+(512 /缓冲池实例数))，其中512是为单页刷新预留的空间。

至少有两个doublewrite文件。doublewrite文件的最大数量是缓冲池实例数量的两倍。由`innodb_buffer_pool_instances`变量控制。) → #ib\_16384\_0.dblwr

Doublewrite文件名的格式如下:#ib\_page\_size\_file\_number.dblwr(或带有DETECT\_的.bdblwr)。例如，为一个InnoDB页面大小为16KB和一个缓冲池的MySQL实例创建以文件:

- `innodb_doublewrite_pages=4`

`innodb_doublewrite_pages`变量(在MySQL 8.0.20中引入)控制每个线程doublewrite文件的数量。如果没有指定，`innodb_doublewrite_pages`将被设置为`innodb_write_io_threads`。高级性能调优。默认值应该适合大多数用户。

- `innodb_doublewrite_batch_size=0`

`innodb_doublewrite_batch_size`变量(在MySQL 8.0.20中引入)控制批量写入doublewrite文件。此变量用于高级性能调优。默认值应该适合大多数用户。

### 三、MSQL并发控制 MVCC (MultiVersion Concurrency Control)

- 推荐阅读: [https://blog.csdn.net/weixin\\_30342639/article/details/107552255](https://blog.csdn.net/weixin_30342639/article/details/107552255)
- [MySQL MVCC.pdf](#)

在MySQL中，MVCC只在读取已提交（Read Committed）和可重复读（Repeatable Read）两个事务级别下有效。

其是通过Undo日志中的版本链和ReadView读一致性视图来实现的。

#### 1. MySQL中每行记录的隐藏字段

首先需要知道的是，在MySQL中，会默认认为我们的表后面添加三个隐藏字段：

- **DB\_ROW\_ID**：行ID，MySQL的B+树索引特性要求每个表必须要有一个主键。如果没有设置的话，会自动寻找第一个不包含NULL的唯一索引列作为主键。如果还是找不到，就会在这个DB\_ROW\_ID上自动生成一个唯一值，以此来当作主键（该列和MVCC的关系不大）；
- **DB\_TRX\_ID**：事务ID，记录的是当前事务在做INSERT或UPDATE语句操作时的事务ID（DELETE语句被当做是UPDATE语句的特殊情况，后面会进行说明）；
- **DB\_ROLL\_PTR**：回滚指针，通过它可以不同的版本串联起来，形成版本链。相当于链表的next指针。



## 2. 事务内部的读取视图 ReadView

ReadView创建时，会查找UndoLog事务版本链，找到最新的一条数据（该条数据的事务已经为提交状态），使用该条数据初始化

- 这个问题，实际上涉及到MySQL隔离级别RR下对幻读的处理；

### ReadView （一个事务内部的读取视图）

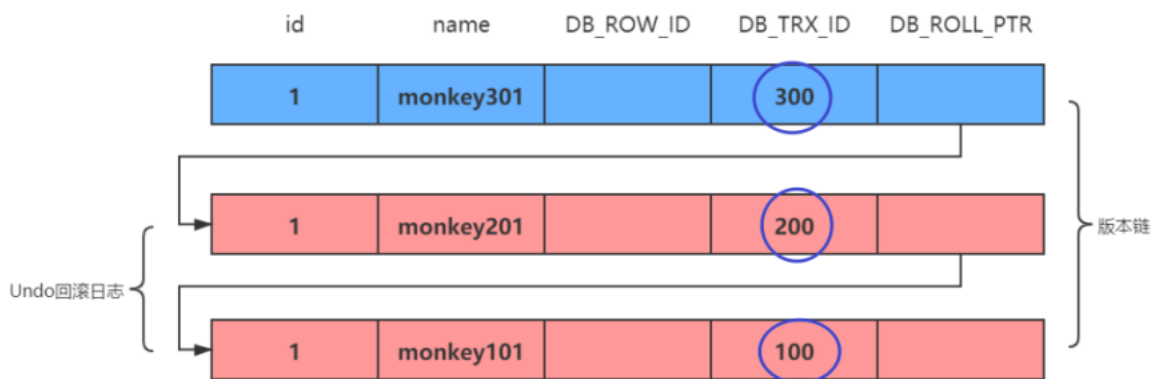
ReadView一致性视图主要是由两部分组成：**所有未提交事务的ID数组**和**已经创建的最大事务ID**组成（实际上ReadView还有其他的字段，但不影响这里对MVCC的讲解）。比如：[100,200],300。事务100和200是当前未提交的事务，而事务300是当前创建的最大事务（已经提交了）。当执行SELECT语句的时候会创建ReadView，但是在读取已提交和可重复读两个事务级别下，生成ReadView的策略是不一样的：**读取已提交级别**是每执行一次SELECT语句就会重新生成一份ReadView，而**可重复读级别**是只在第一次SELECT语句执行的时候会生成一份，后续的SELECT语句会沿用之前生成的ReadView（即使后面有更新语句的话，也会继续沿用）。

## 3. UndoLog中的事务版本链（MV：Multi Version）

### 版本链

所有版本的数据都只会存一份，然后通过回滚指针连接起来，之后就是通过一定的规则找到具体是哪个版本上的数据就行了。假设现在有一张account表，其中有id和name两个字段，那么版本链的示意图如下：

三个不同事务



三个事务同时开启：

- 100：修改name=monkey101
- 200：修改name=monkey201
- 300：修改name=monkey301，提交

如果事务内部没有select语句，那么300事务提交后，新的数据对100和200都是可见的。

但是，一旦100/200事务内部使用select语句读取某个表

**第一步**，读取数据表的所有数据，初始化该表的ReadView读一致性视图。

- 第二步**，获取到该表的Undo Log事务链，获取到的是所有事务已提交的最新数据更新ReadView读一致性视图中的数据值；
  - 隔离级别Read Committed下
    - 每次select查询表都会根据新的UndoLog事务链，生成新的ReadView一致性视图。
  - 隔离级别Repeatable Read下
    - 第一次select查询某表时会初始化该表的ReadView一致性视图，后续所有该表的查询或其他操作都在该ReadView上进行。



- 
- 从一定程度上解决了幻读问题，但是又没有完全解决。

## 四、为什么会有Redolog?

- 为什么InnoDB引擎要提供RedoLog? .zip

