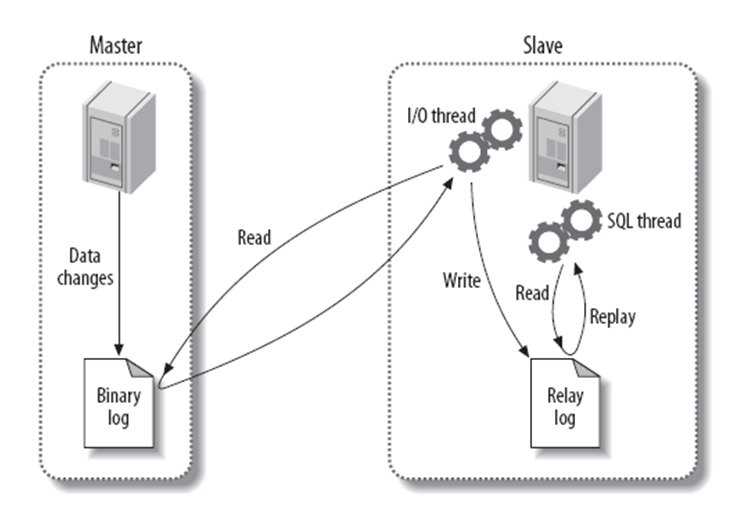
# MySQL Semi-Synchronous Replication

[mnstory.net](http://mnstory.net)

半同步复制(Semi-Synchronous Replication)是相对于同步复制和异步复制而言的折中方案，当两台MySQL数据库需要同步数据，基本的复制思路如图：



master对外提供写服务，并开启binary log功能，然后slave节点启动一个IO线程来从master的binary log同步并将其写入本节点的relay log中，slave再启动一个SQL线程，从relay log中读取日志记录并插入本节点的数据库。

一切原理都是那么简单。

## binary log

### 三种记录模式

1. SQL模式

我们很可能一下子就想到，记录的是SQL语句，任何在master端的写入操作我都同步一份到slave，这样传输的数据量并不大，而且比较符合人类思维，但是有几个潜在的问题需要考虑：

1. 有的函数和特定环境相关，如何保证复制后不同环境下数据是一致的？

例如时间生成函数可能在不同机器上不同，例如uuid等不同设备产生的结果不同，例如使用存储过程操作了本地某些文件并没有同步到对端。

1. slave和master可能因为人为干预后就不再一致。

例如，slave里面如果删除了一条记录再插入一条同样的记录，那用auto id产生的字段都会和master不一致。

1. ROW模式

行模式记录的是表里发生改变行的数据，相对于SQL记录而言，获取源数据的逻辑更靠后，对程序来说处理更简单，不需要考虑和特定环境相关的函数产生不同结果的问题，也不太存在人为干预导致数据不一致问题（可覆盖过来）。

但是行模式同样存在自己的问题，比如，我想在表里面插入一新列，那这个表里所有的的行都要记录一遍到binary log，那日志量可谓惊人。

1. MIX模式

现在人们还没想到第三种原理上不同的日志记录模式，但是肯定会有人想到，既然SQL和ROW模式各有各的优缺点，我们能否取长补短？

于是出现了MIX模式。

MIX模式简单点说，就是针对某一次更改，在SQL和ROW模式中挑一个最优的方式来记录，例如，alert table，我们尽可能用SQL模式，避免产生太多的日志，而insert一行数据更倾向于ROW模式，因为此时ROW模式产生的数据量和SQL相当，并且不需要再次执行SQL解析。

binary log使用什么模式记录日志，可以在配置里面设置，例如下面使用的行模式：

# cat /etc/my.cnf | grep log

log-bin=binlog.bin

binlog-format=ROW

binlog-row-image=minimal

sync-binlog=1

relay log和binary log内部格式一样，配置为：

# cat /etc/my.cnf | grep relay-log

relay-log-info-repository=TABLE

relay-log=relay-log

relay-log-index=relay-log.index

relay-log-purge=1

relay-log-recovery=1

binary log可以设置自己的日志记录模式，但是relay log不能，因为relay log完全是从binary-log同步过来的，没有自主选择权。

### 日志相关操作

目前产生了有哪些日志，我们可以通过mysql命令行查看：

master>show binary logs;

+----------------+-----------+

| Log\_name | File\_size |

+----------------+-----------+

| binglog.000001 | 440656239 |

+----------------+-----------+

1 row in set (0.00 sec)

如果这里的日志比较多，我们还可以清除一些，例如，删除所有日志直到序号03：

master>purge binary logs to 'binlog.000003';

ERROR 1373 (HY000): Target log not found in binlog index

再例如，清除'2014-01-01 14:14:14'以前的日志：

master>purge binary logs before '2014-01-01 14:14:14';

Query OK, 0 rows affected, 1 warning (0.08 sec)

哦，这里发现一个warning，可以通过下面命令看WARING详情：

master>show warnings;

+---------+------+-------------------------------------------------------------------------+

| Level | Code | Message |

+---------+------+-------------------------------------------------------------------------+

| Warning | 1868 | file ./binlog.000001 was not purged because it is the active log file. |

+---------+------+-------------------------------------------------------------------------+

1 row in set (0.00 sec)

如果想知道文件binlog.000001里面的具体内容，可以通过mysqlbinlog来查看(同样适用于relay log)：

# mysqlbinlog /var/lib/mysql/binlog.000001

# at 7522

#161116 18:48:26 server id 1 end\_log\_pos 7575 CRC32 0x4fce539c Update\_rows: table id 70 flags: STMT\_END\_F

BINLOG '

ejksWBMBAAAAUAAAAGIdAAAAAEYAAAAAAAEABmNpbmRlcgAIc2VydmljZXMADRISEgEDDw8PAwEP

DxIOAAAA/QL9Av0C/QL9AgDvHuvhLdM=

ejksWB8BAAAANQAAAJcdAAAAAEYAAAAAAAEAAgANEAACAf4KAAAA/Jma4KwaZbICAJxTzk8=

'/\*!\*/;

上面是ROW模式记录的日志，这日志并没有SQL模式的那么容易看懂，通过MySQL命令也能查看，而且打印格式更可读一些：

mysql> show binlog events in 'binlog.000009' limit 6;

+---------------+-------+----------------+-------------+----------------------------------------------------+

| Log\_name | Pos | Event\_type | End\_log\_pos | Info |

+---------------+-------+----------------+-------------+----------------------------------------------------+

| binlog.000009 | 120 | Previous\_gtids | 191 | 0c25d2e4-a5ba-11e6-8b89-0cc47aaba96e:1-5722078 |

| binlog.000009 | 191 | Gtid | 239 | SET @@SESSION.GTID\_NEXT= '0c25d2e4-a5ba-11e6-8b89' |

| binlog.000009 | 239 | Query | 313 | BEGIN |

| binlog.000009 | 313 | Table\_map | 389 | table\_id: 894 (cinder.volume\_metadata) |

| binlog.000009 | 389 | Update\_rows | 451 | table\_id: 894 flags: STMT\_END\_F |

| binlog.000009 | 451 | Xid | 482 | COMMIT /\* xid=150492246 \*/ |

+---------------+-------+----------------+-------------+----------------------------------------------------+

## 同步、异步和半同步模式

几乎所有人都能想到同步和异步两种日志同步模式，这里，同步和异步指的是，当master提交一条更新事务时，如果要等到slave返回，那就是同步模式，如果不管slave直接返回，那就是异步模式（这里的返回，我认为是等待slave的IO线程返回即可，具体取决于实现，如果要等待SQL线程也做完再返回，也有他的道理）。

### 同步模式

同步的优点是，master和slave之间是强一致的，因为slave写入失败，master不会真正commit这条记录，但是缺点是，master每次都要等到slave返回，那这个性能，的确不敢恭维，特别是有多个slave的时候。

### 异步模式

异步的优点是master和slave之间没有太多相关性，各自干各自的，不存在影响整体性能一说，多台slave也是如此，但缺点是，master和slave之间可能有日志同步不一致，当master突然挂了，slave里面的日志可能并不是当前最新的，存在丢数据的情况，再者，同步模式下，slave提供读功能完全没有问题，但是异步模式下slave提供的读功能需要考虑是否能接受延迟或者想办法规避延迟。

### 半同步模式

同日志记录的SQL模式和ROW模式一样，聪明的人们在同步和异步之外并没有想到第三种模式，于是来了一个各取其长的模式，叫半同步模式。

目前MySQL默认支持的是异步模式(同步是第三方支持的)，也就是说，master干master的，slave干slave的，我们互不影响，但是可能slave和master数据不同步，而半同步模式采用了同步模式的方法，要求至少有一个slave返回OK后，master才返回给用户，所以针对有多个slave的场景，是非常实用的，但是，细细想来，如果只有一个slave，这不就是同步模型吗？这挂羊头卖狗肉的活也堪称半同步？

事实上，为了避免被骂没干什么实事，半同步还有一个机制，就是当一定timeout内(rpl\_semi\_sync\_master\_timeout)没有slave返回，自动将半同步模式切换回异步模式，过一段时间slave追上master了，又自动将模式切换为半同步，有点像电视剧里生孩子的场景，数据和性能，保大还是保小，你看着办：

# grep Semi-sync /var/log/mysqld.log

2016-09-27 09:31:18 145773 [Note] Semi-sync replication switched OFF.

2016-09-27 10:07:06 145773 [Note] Semi-sync replication switched OFF.

2016-09-27 10:07:59 145773 [Note] Semi-sync replication switched ON with slave (server\_id: 2) at (mysqlsf.000001, 508650)

2016-09-27 16:14:37 145773 [Note] Semi-sync replication switched OFF.

2016-09-27 16:14:47 145773 [Note] Semi-sync replication switched ON with slave (server\_id: 2) at (mysqlsf.000001, 73834284)

上面说的rpl\_semi\_sync\_master\_timeout，是master等待slave多久超时(单位ms)，默认是10秒还没有slave返回就切换为异步模式，内网环境好，推荐修改小一点，例如3s：

mysql> show global variables like '%semi%';

+------------------------------------+-------+

| Variable\_name | Value |

+------------------------------------+-------+

| rpl\_semi\_sync\_master\_enabled | ON |

| rpl\_semi\_sync\_master\_timeout | 10000 |

| rpl\_semi\_sync\_master\_trace\_level | 32 |

| rpl\_semi\_sync\_master\_wait\_no\_slave | ON |

| rpl\_semi\_sync\_slave\_enabled | ON |

| rpl\_semi\_sync\_slave\_trace\_level | 32 |

+------------------------------------+-------+

6 rows in set (0.00 sec)

出现这种二选一的艰难决策，显然在数据复制方面，目前的手段并不是很高明，同志们还需努力。

## 启用半同步

如何启用的文章网上很多，但有强迫症的人，不说一点什么，感觉人生都不那么完整了。

启用半同步机制，非常简单，因为这是不是后妈生的，默认自带，先看看你的插件目录有没有这两个文件：

# ls -lh /usr/lib64/mysql/plugin/semisync\_\*

-rwxr-xr-x 1 root root 519K Nov 21 2014 /usr/lib64/mysql/plugin/semisync\_master.so\*

-rwxr-xr-x 1 root root 288K Nov 21 2014 /usr/lib64/mysql/plugin/semisync\_slave.so\*

当然，从MySQL命令里看是否加载了插件更准确些：

mysql> show plugins;

+----------------------------+----------+--------------------+--------------------+-------------+

| Name | Status | Type | Library | License |

+----------------------------+----------+--------------------+--------------------+-------------+

| binlog | ACTIVE | STORAGE ENGINE | NULL | PROPRIETARY |

...

| rpl\_semi\_sync\_master | ACTIVE | REPLICATION | semisync\_master.so | PROPRIETARY |

| rpl\_semi\_sync\_slave | ACTIVE | REPLICATION | semisync\_slave.so | PROPRIETARY |

+----------------------------+----------+--------------------+--------------------+-------------+

记得在配置里面添加相应选项：

# grep sync /etc/my.cnf

sync-binlog=1

sync-master-info=1

rpl-semi-sync-slave-enabled=1

rpl-semi-sync-master-enabled=1

rpl-semi-sync-master-timeout=10000

上面有用show global variables like '%semi%'来打印半同步相关的信息，里面有rpl\_semi\_sync\_master\_enabled 为ON字段，如果为ON表示真的启用了，比配置里面配置了还要真。

看看slave线程在不在：

mysql> show processlist;

+--------+-------------+---------+--------+----------------------------------------+

| Id | User | Command | Time | State |

+--------+-------------+---------+--------+----------------------------------------+

| 20 | system user | Connect | 877466 | Waiting for master to send event |

| 21 | system user | Connect | 0 | Slave has read all relay log; waiting |

| for the slave I/O thread to update it |

| 22 | system user | Connect | 1 | Waiting for an event from Coordinator |

| 23 | system user | Connect | 3 | Waiting for an event from Coordinator |

| 628655 | root | Query | 0 | init |

+--------+-------------+---------+--------+----------------------------------------+

7 rows in set (0.00 sec)

### 性能选项

同步可能需要考虑性能问题，同步慢了，对大家都不好，对于master，我们有两个非常重要的参数：innodb\_flush\_log\_at\_trx\_commit和sync\_binlog。

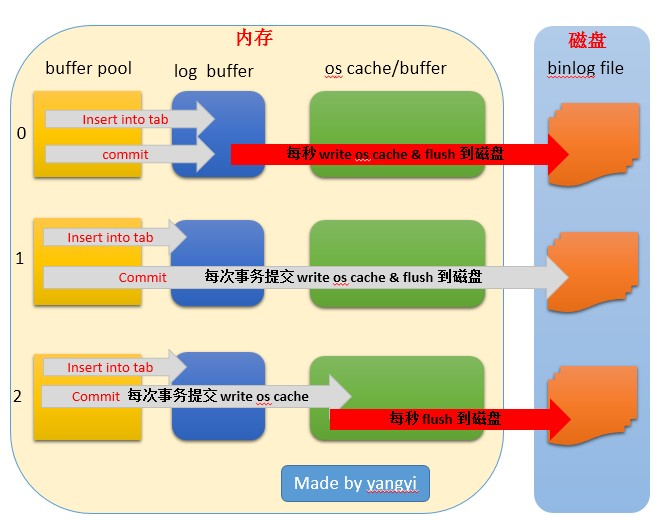
如果启动了autocommit，那么每statement会写入binary log，如果不没有启用autocommit，那么每次transaction会写入binary log。

1. **sync\_binlog**
2. 默认来说，sync\_binlog是0，数据库不会在每次写入binary log后调用 fdatasync()来确保数据已经写入存储；
3. 如果设置为1，每次写的binary log都会同步到磁盘，这么看来，最多就丢一条日志(万一这条日志写入失败了就丢了)，这种方式最安全的，但也是性能最差的；
4. 如果设置为更大的值，例如200条，性能是好了，但是可能(可能而已，因为还有其他选项交叉决定)真会丢失200条binary log。
5. **innodb\_flush\_log\_at\_trx\_commit**

innodb\_flush\_log\_at\_trx\_commit默认为0，我直接抄这位仁兄<http://blog.itpub.net/22664653/viewspace-1063134/> 的BLOG了，因为他的图画的不错，解释也很好：

1. 如果innodb\_flush\_log\_at\_trx\_commit设置为0，log buffer将每秒一次地写入log file中，并且log file的flush(刷到磁盘)操作同时进行.该模式下，在事务提交的时候，不会主动触发写入磁盘的操作。
2. 如果innodb\_flush\_log\_at\_trx\_commit设置为1，每次事务提交时MySQL都会把log buffer的数据写入log file，并且flush(刷到磁盘)中去.
3. 如果innodb\_flush\_log\_at\_trx\_commit设置为2，每次事务提交时MySQL都会把log buffer的数据写入log file.但是flush(刷到磁盘)操作并不会同时进行。该模式下,MySQL会每秒执行一次 flush(刷到磁盘)操作。

并附上盗图一张：



一般来说，如果想要数据最安全，不太在意性能，建议两者都设置为1：

For the greatest possible durability and consistency in a replication setup using InnoDB with transactions, you should useinnodb\_flush\_log\_at\_trx\_commit=1 and sync\_binlog=1 in the master my.cnf file.

## 同步状态查看

MySQL主要提供了两个命令，用于查看主从节点的同步信息。我们可以在主节点查看master的状态信息，当然，也没啥太多内容：

master>show master status\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

File: binlog.000001 #日志文件

Position: 418613282

Binlog\_Do\_DB:

Binlog\_Ignore\_DB: #不记录入日志文件的db

Executed\_Gtid\_Set: 8875511e-9a89-11e6-8292-a0369f9be84c:1-623844, a0b7791e-aa6a-11e6-aa1d-50af736f4ba0:1-3

1 row in set (0.00 sec)

我们可以在从节点看自己作为slave的信息，这个就比较丰富了：

# mysql -hslave -uroot -e "show slave status\G"

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Slave\_IO\_State: Queueing master event to the relay log

正在将master的log插入slave的relay log中。命令详解见 <http://dev.mysql.com/doc/refman/5.7/en/slave-io-thread-states.html>。

Master\_Host: 200.200.102.206

Master\_User: root

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: binlog.000007

Read\_Master\_Log\_Pos: 715424679

Relay\_Log\_File: relay-log.000019

Relay\_Log\_Pos: 715424483

Relay\_Master\_Log\_File: binlog.000007

Slave\_IO\_Running: Yes

从master的binary日志里，取最新记录，同步到slave的relay日志里，如果为No，可能是网络有问题，权限不够等。

Slave\_SQL\_Running: Yes

从slave的relay日志里，取日志记录，并执行更新操作，如果为No，可能是主从数据库不一致了，导致执行更新失败，此时Last\_SQL\_Error和Last\_SQL\_Error\_Timestamp会记录错误原因和时间。

出错后，为了不至于让数据库更糟糕，此线程会停止，而IO线程还会继续，保证最新日志同步，当人工介入，错误修复后，需stop slave再start slave重启SQL线程。

当(Master\_Log\_File == Relay\_Master\_Log\_File && Relay\_Log\_Pos == Exec\_Master\_Log\_Pos && Slave\_SQL\_Running\_State == "Wait")为true时，表示主从处于完全同步状态。

Replicate\_Do\_DB:

Replicate\_Ignore\_DB:

Replicate\_Do\_Table:

Replicate\_Ignore\_Table:

Replicate\_Wild\_Do\_Table:

Replicate\_Wild\_Ignore\_Table:

Last\_Errno: 0

Last\_Error:

Skip\_Counter: 0

Exec\_Master\_Log\_Pos: 715424277

Relay\_Log\_Space: 715425168

Until\_Condition: None

Until\_Log\_File:

Until\_Log\_Pos: 0

Master\_SSL\_Allowed: No

Master\_SSL\_CA\_File:

Master\_SSL\_CA\_Path:

Master\_SSL\_Cert:

Master\_SSL\_Cipher:

Master\_SSL\_Key:

Seconds\_Behind\_Master: 0

Seconds\_Behind\_Master 是以秒为单位衡量slave节点 的SQL 线程与slave节点的 I/O 线程之间的延迟，也可以表示当前relay日志中没有在slave节点上执行的事务数量与master节点中已提交的事务数量的关系。

如果发现 Seconds\_Behind\_Master 的值大于 0，那就意味着可能有较高的网络延迟或服务器负载。

Master\_SSL\_Verify\_Server\_Cert: No

Last\_IO\_Errno: 0

Last\_IO\_Error:

Last\_SQL\_Errno: 0

Last\_SQL\_Error:

Replicate\_Ignore\_Server\_Ids:

Master\_Server\_Id: 1

Master\_UUID: 8875511e-9a89-11e6-8292-a0369f9be84c

Master\_Info\_File: mysql.slave\_master\_info

SQL\_Delay: 0

SQL\_Remaining\_Delay: NULL

Slave\_SQL\_Running\_State: Reading event from the relay log

这个表示slave的SQL线程正从relay日志里面读取日志来执行，如果是Wait状态，表示当前无事可做。命令详解见<http://dev.mysql.com/doc/refman/5.7/en/slave-sql-thread-states.html>。

Master\_Retry\_Count: 86400

Master\_Bind:

Last\_IO\_Error\_Timestamp:

Last\_SQL\_Error\_Timestamp:

Master\_SSL\_Crl:

Master\_SSL\_Crlpath:

Retrieved\_Gtid\_Set: 8875511e-9a89-11e6-8292-a0369f9be84c:1300-8753032

Executed\_Gtid\_Set: 8875511e-9a89-11e6-8292-a0369f9be84c:1-8753031,acdd9e1f-a23e-11e6-b4d4-50af736f4ba0:1-6

Auto\_Position: 1

从上面的slave status，可以看出很多信息来，例如：

1. IO错误1593

Slave\_IO\_Running: No

Slave\_SQL\_Running: Yes

Last\_IO\_Errno: 1593

Last\_IO\_Error: Fatal error: The slave I/O thread stops because master and slave have equal MySQL server ids; these ids must be different for replication to work (or the --replicate-same-server-id option must be used on slave but this does not always make sense; please check the manual before using it).

MYSQL的ServerID要不一样，才能正确Replication：

200.200.102.206 cat /etc/my.cnf

server-id=1

200.200.102.207 cat /etc/my.cnf

server-id=2

1. SQL错误1146

Slave\_IO\_State: Waiting for master to send event

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: No

Last\_SQL\_Errno: 1146

Last\_SQL\_Error: Error Table 'book3.no\_tbl\_on\_slave' doesn't exist' on query. Default database: 'book3'. Query: 'INSERT INTO no\_tbl\_on\_slave(id) VALUES(1)'

slave节点上没有创建数据表，需要创建再复制。

1. SQL错误1062

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: No

Seconds\_Behind\_Master: NULL

Last\_SQL\_Errno: 1062

Last\_SQL\_Error: Error Duplicate entry '1' for key 'id' on query. Default database: 'book3'. Query: 'INSERT INTO uniq\_test(id) VALUES(1),(2),(3)'

slave节点发现外键错误，我们可以跳过这个错误继续，但是最好找到根本原因，为啥会出现数据不一致，因为后面还可能会有类似错误。

## 参考资料

1. Ronald Bradford - 《Effective MySQL: Replication Techniques in Depth》
2. MySQL文档 - “15.14 InnoDB Startup Options and System Variables”

<http://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html>

1. MySQL文档 - “Chapter 18 Replication”

<http://dev.mysql.com/doc/refman/5.7/en/replication.html>

1. MySQL文档 - “18.3 Replication Solutions”

<http://dev.mysql.com/doc/refman/5.7/en/replication-solutions.html>

1. 北在南方 – “sync\_binlog innodb\_flush\_log\_at\_trx\_commit 浅析”

<http://blog.itpub.net/22664653/viewspace-1063134/>

1. 求知不倦 – “实战体验几种MySQL Cluster方案”

<http://blog.csdn.net/kingofworld/article/details/44786123>

2016/11/19