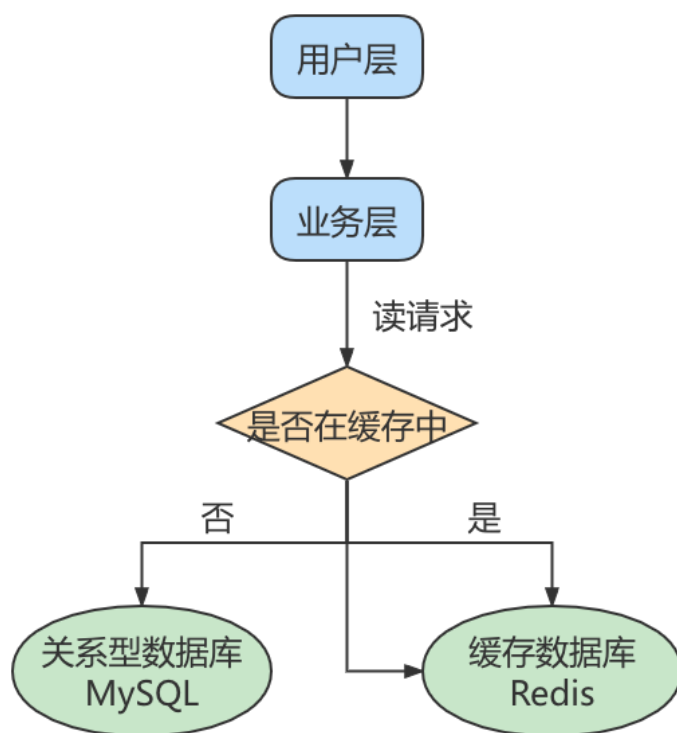


# 第18章\_主从复制

## 1. 主从复制概述

### 1.1 如何提升数据库并发能力



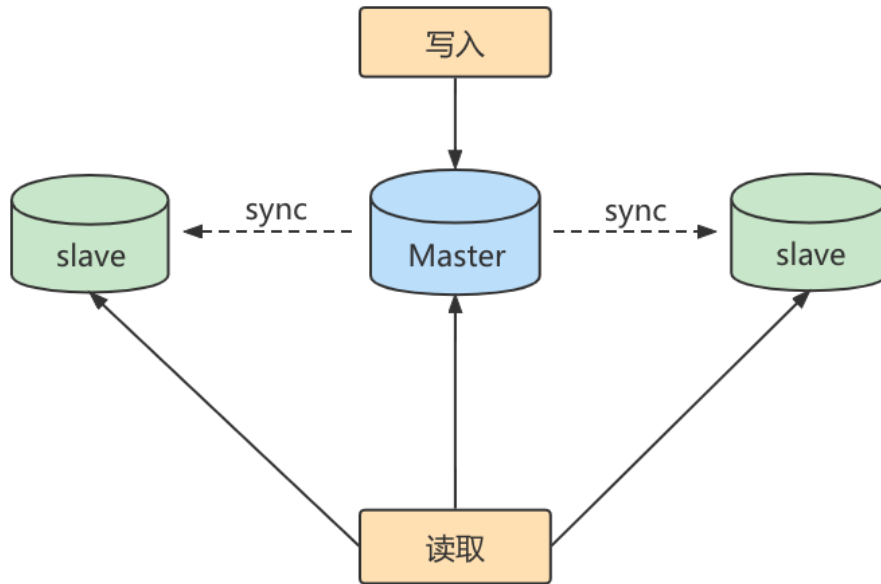
此外，一般应用对数据库而言都是“**读多写少**”，也就说对数据库读取数据的压力比较大，有一个思路就是采用数据库集群的方案，做 **主从架构**、进行 **读写分离**，这样同样可以提升数据库的并发处理能力。但并不是所有的应用都需要对数据库进行主从架构的设置，毕竟设置架构本身是有成本的。

如果我们的目的在于提升数据库高并发访问的效率，那么首先考虑的是如何 **优化SQL和索引**，这种方式简单有效；其次才是采用 **缓存的策略**，比如使用 Redis将热点数据保存在内存数据库中，提升读取的效率；最后才是对数据库采用 **主从架构**，进行读写分离。

### 1.2 主从复制的作用

主从同步设计不仅可以提高数据库的吞吐量，还有以下 3 个方面的作用。

**第1个作用：读写分离。**



第2个作用就是数据备份。

第3个作用是具有高可用性。

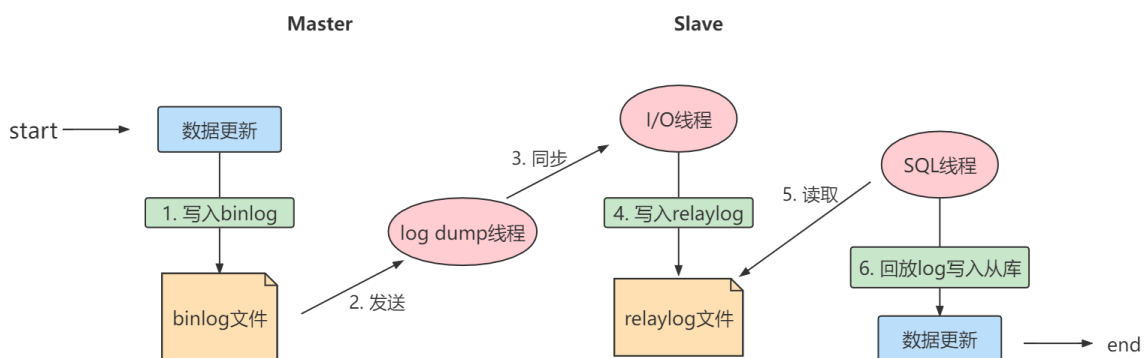
## 2. 主从复制的原理

Slave 会从 Master 读取 binlog 来进行数据同步。

### 2.1 原理剖析

#### 三个线程

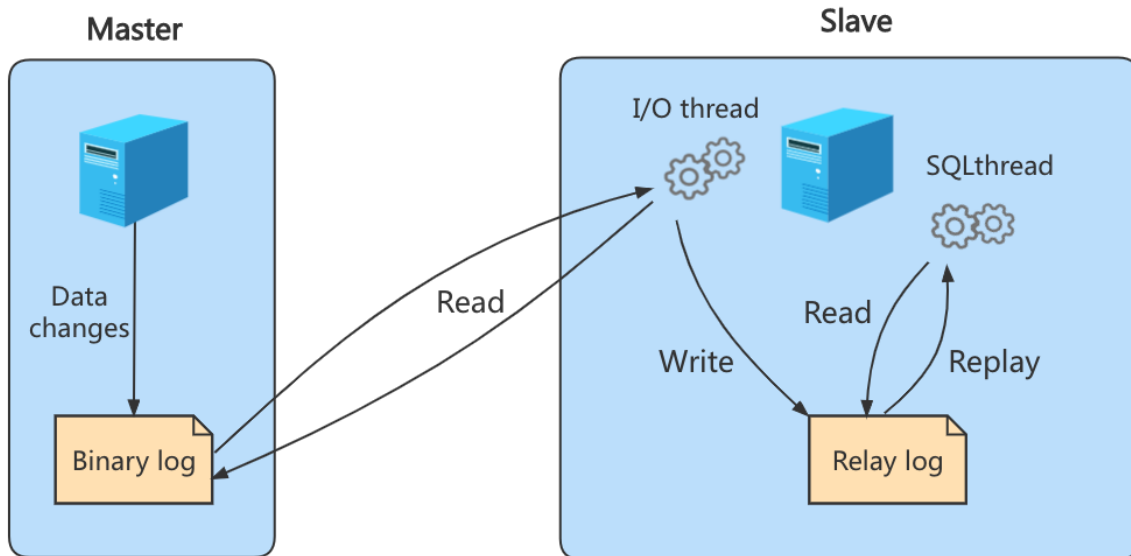
实际上主从同步的原理就是基于 binlog 进行数据同步的。在主从复制过程中，会基于 3 个线程 来操作，一个主库线程，两个从库线程。



**二进制日志转储线程**（Binlog dump thread）是一个主库线程。当从库线程连接的时候，主库可以将二进制日志发送给从库，当主库读取事件（Event）的时候，会在 Binlog 上 **加锁**，读取完成之后，再将锁释放掉。

从库 **I/O 线程** 会连接到主库，向主库发送请求更新 Binlog。这时从库的 I/O 线程就可以读取到主库的二进制日志转储线程发送的 Binlog 更新部分，并且拷贝到本地的中继日志（Relay log）。

从库 **SQL 线程** 会读取从库中的中继日志，并且执行日志中的事件，将从库中的数据与主库保持同步。



### 复制三步骤

步骤1: **Master** 将写操作记录到二进制日志 ( **binlog** ) 。

步骤2: **Slave** 将 **Master** 的binary log events拷贝到它的中继日志 ( **relay log** ) ；

步骤3: **Slave** 重做中继日志中的事件，将改变应用到自己的数据库中。MySQL复制是异步的且串行化的，而且重启后从 **接入点** 开始复制。

### 复制的问题

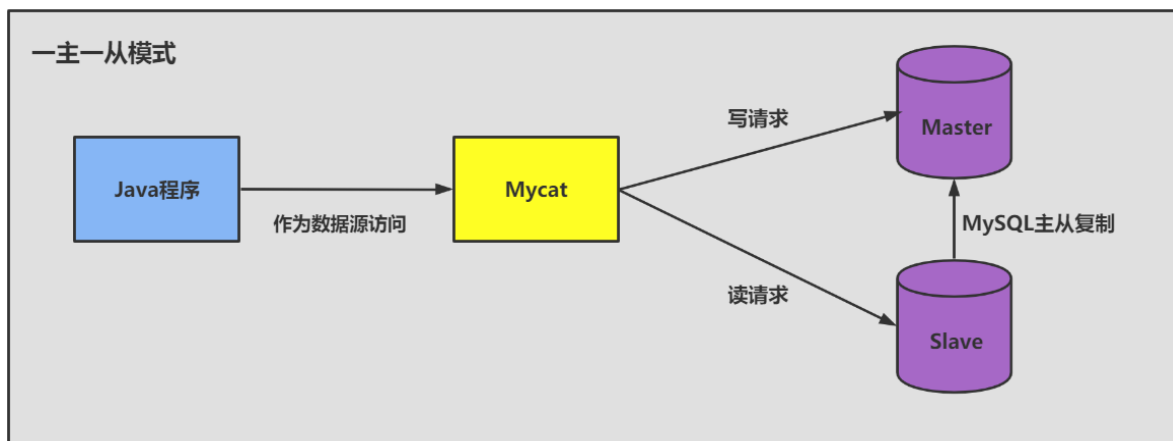
复制的最大问题: **延时**

## 2.2 复制的基本原则

- 每个 **Slave** 只有一个 **Master**
- 每个 **Slave** 只能有一个唯一的服务器ID
- 每个 **Master** 可以有多个 **Slave**

## 3. 一主一从架构搭建

一台 **主机** 用于处理所有 **写请求**，一台 **从机** 负责所有 **读请求**，架构图如下：



## 3.1 准备工作

- 1、准备 2台 CentOS 虚拟机
- 2、每台虚拟机上需要安装好MySQL (可以是MySQL8.0)

说明：前面我们讲过如何克隆一台CentOS。大家可以在一台CentOS上安装好MySQL，进而通过克隆的方式复制出1台包含MySQL的虚拟机。

注意：克隆的方式需要修改新克隆出来主机的：① MAC地址 ② hostname ③ IP 地址 ④ UUID。

此外，克隆的方式生成的虚拟机（包含MySQL Server），则克隆的虚拟机MySQL Server的UUID相同，必须修改，否则在有些场景会报错。比如：`show slave status\G`，报如下的错误：

```
Last_I0_Error: Fatal error: The slave I/O thread stops because master and slave have equal MySQL server UUIDs; these UUIDs must be different for replication to work.
```

修改MySQL Server 的UUID方式：

```
vim /var/lib/mysql/auto.cnf

systemctl restart mysqld
```

## 3.2 主机配置文件

建议mysql版本一致且后台以服务运行，主从所有配置项都配置在 `[mysqld]` 节点下，且都是小写字母。

具体参数配置如下：

- 必选

```
#[必须]主服务器唯一ID
server-id=1

#[必须]启用二进制日志,指名路径。比如：自己本地的路径/log/mysqlbin
log-bin=atguigu-bin
```

- 可选

```
#[可选] 0（默认）表示读写（主机），1表示只读（从机）
read-only=0

#设置日志文件保留的时长，单位是秒
binlog_expire_logs_seconds=6000

#控制单个二进制日志大小。此参数的最大和默认值是1GB
max_binlog_size=200M

#[可选]设置不要复制的数据库
binlog-ignore-db=test

#[可选]设置需要复制的数据库,默认全部记录。比如：binlog-do-db=atguigu_master_slave
binlog-do-db=需要复制的主数据库名字

#[可选]设置binlog格式
binlog_format=STATEMENT
```

## binlog格式设置:

格式1: **STATEMENT模式** (基于SQL语句的复制(statement-based replication, SBR))

```
binlog_format=STATEMENT
```

每一条会修改数据的sql语句会记录到binlog中。这是默认的binlog格式。

- SBR 的优点:
  - 历史悠久, 技术成熟
  - 不需要记录每一行的变化, 减少了binlog日志量, 文件较小
  - binlog中包含了所有数据库更改信息, 可以据此来审核数据库的安全等情况
  - binlog可以用于实时的还原, 而不仅仅用于复制
  - 主从版本可以不一样, 从服务器版本可以比主服务器版本高
- SBR 的缺点:
  - 不是所有的UPDATE语句都能被复制, 尤其是包含不确定操作的时候
- 使用以下函数的语句也无法被复制: LOAD\_FILE()、UUID()、USER()、FOUND\_ROWS()、SYSDATE() (除非启动时启用了 --sysdate-is-now 选项)
  - INSERT ... SELECT 会产生比 RBR 更多的行级锁
  - 复制需要进行全表扫描(WHERE 语句中没有使用到索引)的 UPDATE 时, 需要比 RBR 请求更多的行级锁
  - 对于有 AUTO\_INCREMENT 字段的 InnoDB表而言, INSERT 语句会阻塞其他 INSERT 语句
  - 对于一些复杂的语句, 在从服务器上的耗资源情况会更严重, 而 RBR 模式下, 只会对那个发生变化的记录产生影响
  - 执行复杂语句如果出错的话, 会消耗更多资源
  - 数据表必须几乎和主服务器保持一致才行, 否则可能会导致复制出错

## ② ROW模式 (基于行的复制(row-based replication, RBR))

```
binlog_format=ROW
```

5.1.5版本的MySQL才开始支持, 不记录每条sql语句的上下文信息, 仅记录哪条数据被修改了, 修改成什么样了。

- RBR 的优点:
  - 任何情况都可以被复制, 这对复制来说是最 **安全可靠** 的。(比如: 不会出现某些特定情况下的存储过程、function、trigger的调用和触发无法被正确复制的问题)
  - 多数情况下, 从服务器上的表如果有主键的话, 复制就会快了很多
  - 复制以下几种语句时的行锁更少: INSERT ... SELECT、包含 AUTO\_INCREMENT 字段的 INSERT、没有附带条件或者并没有修改很多记录的 UPDATE 或 DELETE 语句
  - 执行 INSERT, UPDATE, DELETE 语句时锁更少
  - 从服务器上采用 **多线程** 来执行复制成为可能
- RBR 的缺点:
  - binlog 大了很多
  - 复杂的回滚时 binlog 中会包含大量的数据
  - 主服务器上执行 UPDATE 语句时, 所有发生变化的记录都会写到 binlog 中, 而 SBR 只会写一次, 这会导致频繁发生 binlog 的并发写问题
  - 无法从 binlog 中看到都复制了些什么语句

## ③ MIXED模式 (混合模式复制(mixed-based replication, MBR))

```
binlog_format=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。

在Mixed模式下，一般的语句修改使用statement格式保存binlog。如一些函数，statement无法完成主从复制的操作，则采用row格式保存binlog。

MySQL会根据执行的每一条具体的sql语句来区分对待记录的日志形式，也就是在Statement和Row之间选择一种。

### 3.3 从机配置文件

要求主从所有配置项都配置在 `my.cnf` 的 `[mysqld]` 栏位下，且都是小写字母。

- 必选

```
#[必须]从服务器唯一ID
server-id=2
```

- 可选

```
#[可选]启用中继日志
relay-log=mysql-relay
```

重启后台mysql服务，使配置生效。

注意：主从机都关闭防火墙

```
service iptables stop #CentOS 6
```

```
systemctl stop firewalld.service #CentOS 7
```

### 3.4 主机：建立账户并授权

#在主机MySQL里执行授权主从复制的命令

```
GRANT REPLICATION SLAVE ON *.* TO 'slave1'@'从机器数据库IP' IDENTIFIED BY 'abc123';
#5.5,5.7
```

**注意：如果使用的是MySQL8，需要如下的方式建立账户，并授权slave：**

```
CREATE USER 'slave1'@'%' IDENTIFIED BY '123456';
```

```
GRANT REPLICATION SLAVE ON *.* TO 'slave1'@'%' ;
```

#此语句必须执行。否则见下面。

```
ALTER USER 'slave1'@'%' IDENTIFIED WITH mysql_native_password BY '123456';
```

```
flush privileges;
```

注意：在从机执行show slave status\G时报错：

```
Last_IO_Error: error connecting to master 'slave1@192.168.1.150:3306' - retry-time: 60 retries: 1
message: Authentication plugin 'caching_sha2_password' reported error: Authentication requires
secure connection.
```

查询Master的状态，并记录下File和Position的值。

```
show master status;
```

```
mysql> show master status;
+-----+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+-----+-----+-----+-----+-----+
| mysql-bin.000007 |      154 | testdb       | mysql             |                   |
+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

- 记录下File和Position的值

注意：执行完此步骤后 **不要再操作主服务器MySQL**，防止主服务器状态值变化。

### 3.5 从机：配置需要复制的主机

**步骤1：**从机上复制主机的命令

```
CHANGE MASTER TO
MASTER_HOST='主机的IP地址',
MASTER_USER='主机用户名',
MASTER_PASSWORD='主机用户名的密码',
MASTER_LOG_FILE='mysql-bin.具体数字',
MASTER_LOG_POS=具体值;
```

举例：

```
CHANGE MASTER TO
MASTER_HOST='192.168.1.150', MASTER_USER='slave1', MASTER_PASSWORD='123456', MASTER_LOG_F
ILE='atguigu-bin.000007', MASTER_LOG_POS=154;
```

```
mysql> CHANGE MASTER TO MASTER_HOST='192.168.140.128',
-> MASTER_USER='slave',
-> MASTER_PASSWORD='123123',
-> MASTER_LOG_FILE='mysql-bin.000007', MASTER_LOG_POS=154;
Query OK, 0 rows affected, 2 warnings (0.00 sec)
```

```
mysql> CHANGE MASTER TO MASTER_HOST='192.168.124.3',
-> MASTER_USER='zhangsan',
-> MASTER_PASSWORD='123456',
-> MASTER_LOG_FILE='mysqlbin.000012', MASTER_LOG_POS=4386;
ERROR 1198 (HY000): This operation cannot be performed with a running slave; run STOP SLAVE first
mysql> stop slave;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CHANGE MASTER TO MASTER_HOST='192.168.124.3',
-> MASTER_USER='zhangsan',
-> MASTER_PASSWORD='123456',
-> MASTER_LOG_FILE='mysqlbin.000012', MASTER_LOG_POS=4386;
Query OK, 0 rows affected (0.01 sec)
mysql>
```

如果之前做过同步，请先停止

**步骤2：**

```
#启动slave同步
START SLAVE;
```

```
mysql> start slave;
Query OK, 0 rows affected (0.00 sec)
```

如果报错：

```
mysql> start slave;
ERROR 1872 (HY000): Slave failed to initialize relay log info structure from the repository
mysql> reset slave;
Query OK, 0 rows affected, 1 warning (0.03 sec)
```

可以执行如下操作，删除之前的relay\_log信息。然后重新执行 CHANGE MASTER TO ...语句即可。

```
mysql> reset slave; #删除SLAVE数据库的relaylog日志文件，并重新启用新的relaylog文件
```

接着，查看同步状态：

```
SHOW SLAVE STATUS\G;
```

```
mysql> SHOW SLAVE STATUS\G;
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
      Master_Host: 172.16.116.1
      Master_User: slave01
      Master_Port: 3306
      Connect_Retry: 60
      Master_Log_File: logbin.000001
      Read_Master_Log_Pos: 154
      Relay_Log_File: mysql-relay.000002
      Relay_Log_Pos: 317
      Relay_Master_Log_File: logbin.000001
      Slave_IO_Running: Yes
      Slave_SQL_Running: Yes
      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: 154
      Relay_Log_Space: 520
      Until_Condition: None
      Until_Log_File:
      Until_Log_Pos: 0
      Master_SSL_Allowed: No
      Master_SSL_CA_File:
```

这两个都是yes，说明同步配置成功了

上面两个参数都是Yes，则说明主从配置成功！

显式如下的情况，就是不正确的。可能错误的原因有：

1. 网络不通
2. 账户密码错误
3. 防火墙
4. mysql配置文件问题
5. 连接服务器时语法
6. 主服务器mysql权限



```

mysql> show slave status \G
***** 1. row *****
Slave_IO_State: Connecting to master
Master_Host: 192.168.1.110
Master_User: slave
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.000001
Read_Master_Log_Pos: 1513
Relay_Log_File: mysql-relay.000001
Relay_Log_Pos: 4
Relay_Master_Log_File: mysql-bin.000001
Slave_IO_Running: Connecting
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:

```

### 3.6 测试

主机新建库、新建表、insert记录，从机复制：

```

CREATE DATABASE atguigu_master_slave;

CREATE TABLE mytbl(id INT,NAME VARCHAR(16));

INSERT INTO mytbl VALUES(1, 'zhang3');

INSERT INTO mytbl VALUES(2,@hostname);

```

### 3.7 停止主从同步

- 停止主从同步命令：

```
stop slave;
```

- 如何重新配置主从

如果停止从服务器复制功能，再使用需要重新配置主从。否则会报错如下：

```

-> MASTER_LOG_FILE='mysql-bin.000003',MASTER_LOG_POS=722;
ERROR 3021 (HY000): This operation cannot be performed with a running slave io thread; run STOP SLAVE
IO_THREAD FOR CHANNEL '' first.
mysql> stop slave;
Query OK, 0 rows affected (0.00 sec)

```

重新配置主从，需要在从机上执行：

```

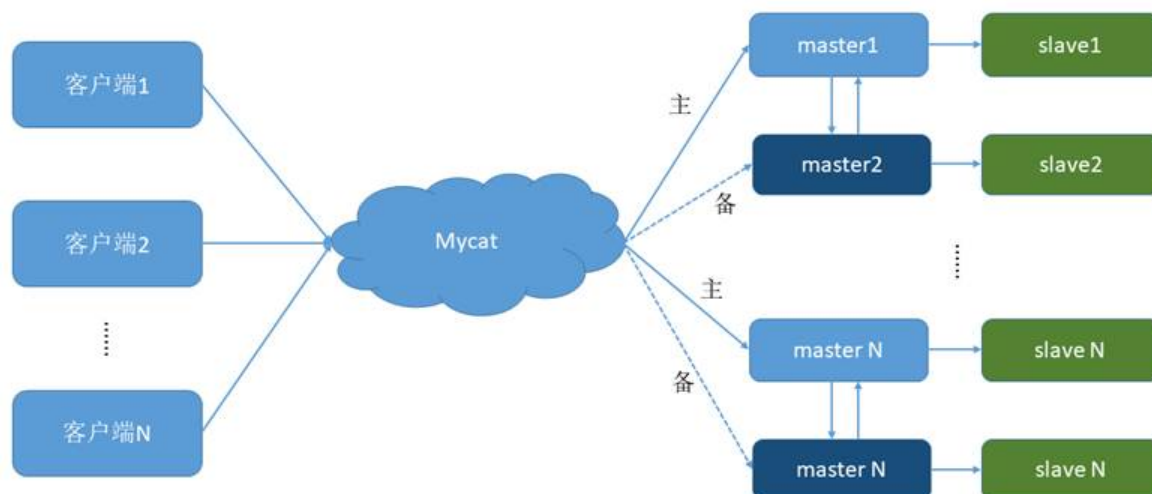
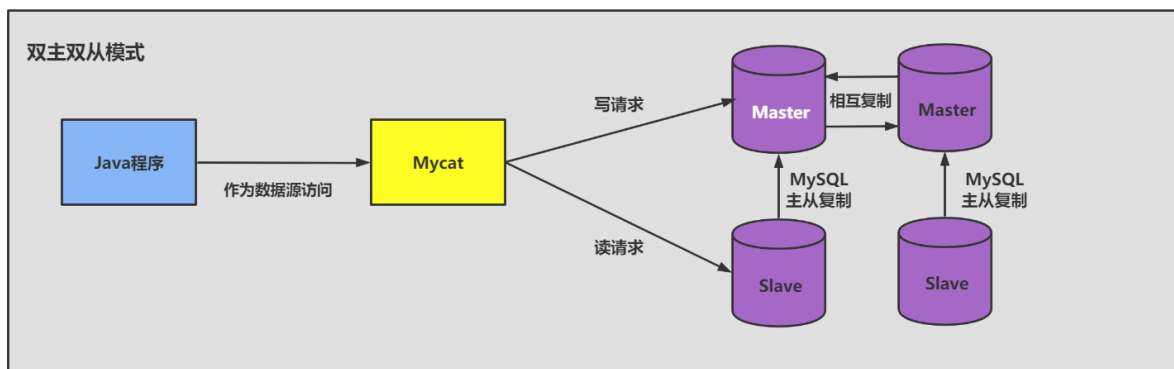
stop slave;

reset master; #删除Master中所有的binlog文件，并将日志索引文件清空，重新开始所有新的日志文件(慎用)

```

### 3.8 后续

**搭建主从复制：双主双从**



## 4. 同步数据一致性问题

### 主从同步的要求：

- 读库和写库的数据一致(最终一致)；
- 写数据必须写到写库；
- 读数据必须到读库(不一定)；

### 4.1 理解主从延迟问题

进行主从同步的内容是二进制日志，它是一个文件，在进行网络传输的过程中就一定会存在主从延迟（比如 500ms），这样就可能造成用户在从库上读取的数据不是最新的数据，也就是主从同步中的数据不一致性问题。

### 4.2 主从延迟问题原因

在网络正常的时候，日志从主库传给从库所需的时间是很短的，即 $T_2 - T_1$ 的值是非常小的。即，网络正常情况下，主备延迟的主要来源是备库接收完binlog和执行完这个事务之间的时间差。

**主备延迟最直接的表现是，从库消费中继日志（relay log）的速度，比主库生产binlog的速度要慢。造成原因：**

- 1、从库的机器性能比主库要差
- 2、从库的压力大
- 3、大事务的执行

**举例1：**一次性用delete语句删除太多数据

结论：后续再删除数据的时候，要控制每个事务删除的数据量，分成多次删除。

**举例2：**一次性用insert...select插入太多数据

**举例3：**大表DDL

比如在主库对一张500W的表添加一个字段耗费了10分钟，那么从节点上也会耗费10分钟。

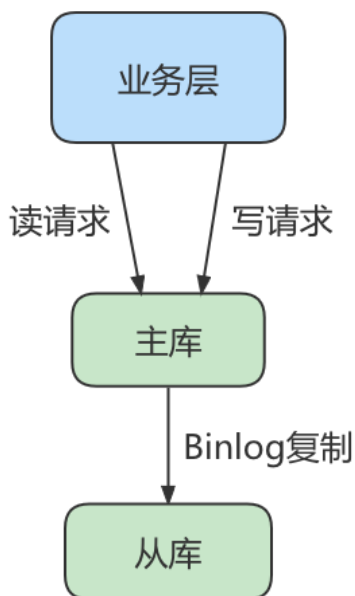
## 4.3 如何减少主从延迟

若想要减少主从延迟的时间，可以采取下面的办法：

1. 降低多线程大事务并发的概率，优化业务逻辑
2. 优化SQL，避免慢SQL，减少批量操作，建议写脚本以update-sleep这样的形式完成。
3. 提高从库机器的配置，减少主库写binlog和从库读binlog的效率差。
4. 尽量采用短的链路，也就是主库和从库服务器的距离尽量要短，提升端口带宽，减少binlog传输的网络延时。
5. 实时性要求的业务读强制走主库，从库只做灾备，备份。

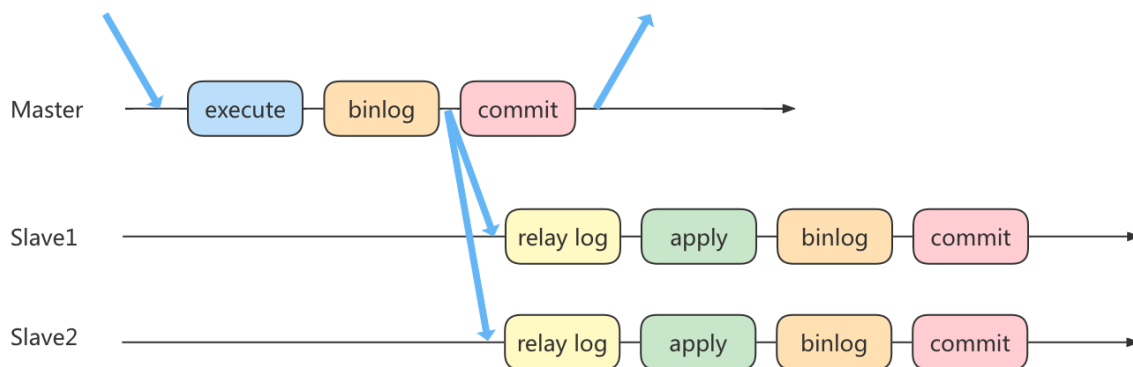
## 4.4 如何解决一致性问题

如果操作的数据存储在同一个数据库中，那么对数据进行更新的时候，可以对记录加写锁，这样在读取的时候就不会发生数据不一致的情况。但这时从库的作用就是备份，并没有起到读写分离，分担主库读压力的作用。

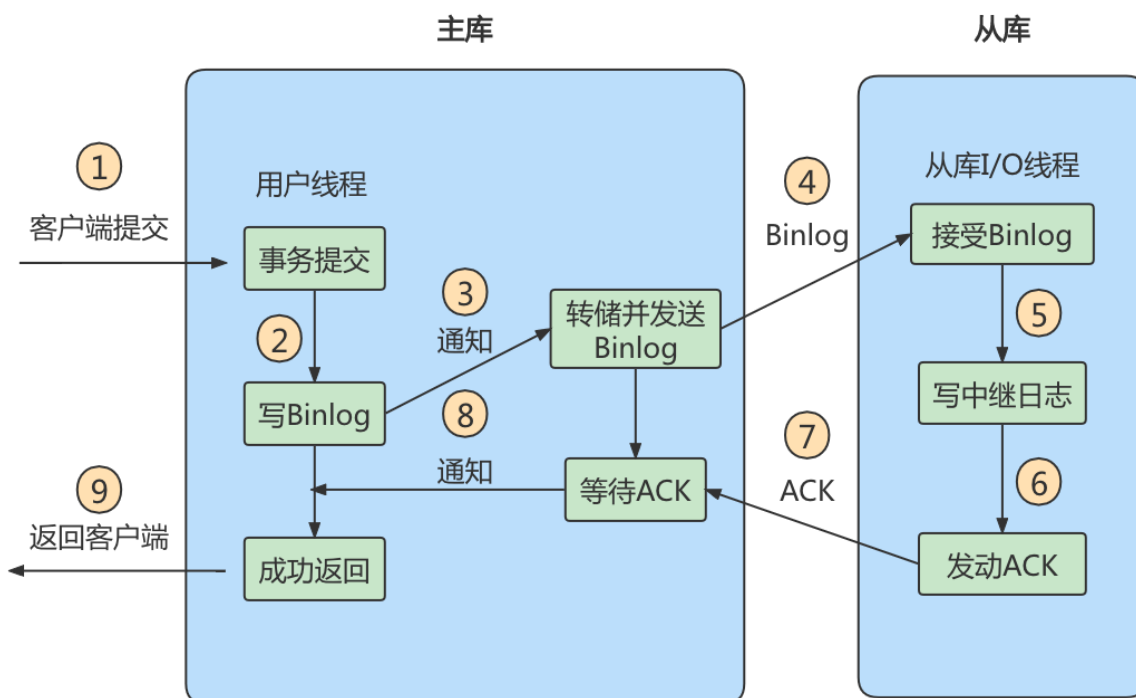


读写分离情况下，解决主从同步中数据不一致的问题，就是解决主从之间数据复制方式的问题，如果按照数据一致性从弱到强来进行划分，有以下3种复制方式。

### 方法1：异步复制



## 方法 2：半同步复制



## 方法 3：组复制

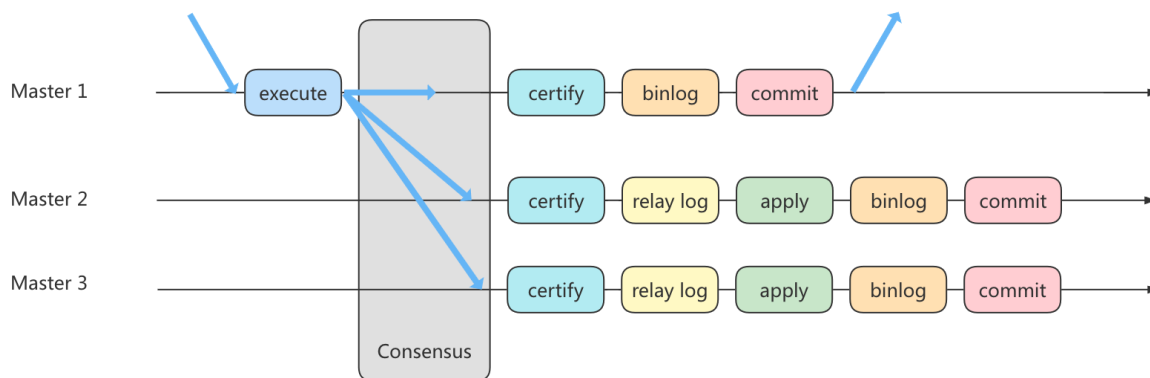
异步复制和半同步复制都无法最终保证数据的一致性问题，半同步复制是通过判断从库响应的个数来决定是否返回给客户端，虽然数据一致性相比于异步复制有提升，但仍然无法满足对数据一致性要求高的场景，比如金融领域。MGR 很好地弥补了这两种复制模式的不足。

组复制技术，简称 MGR (MySQL Group Replication)。是 MySQL 在 5.7.17 版本中推出的一种新的数据复制技术，这种复制技术是基于 Paxos 协议的状态机复制。

### MGR 是如何工作的

首先我们将多个节点共同组成一个复制组，在 **执行读写 (RW) 事务** 的时候，需要通过一致性协议层 (Consensus 层) 的同意，也就是读写事务想要进行提交，必须要经过组里“大多数人” (对应 Node 节点) 的同意，大多数指的是同意的节点数量需要大于  $(N/2+1)$ ，这样才可以进行提交，而不是原发起方一个说了算。而针对 **只读 (RO) 事务** 则不需要经过组内同意，直接 COMMIT 即可。

在一个复制组内有多个节点组成，它们各自维护了自己的数据副本，并且在一致性协议层实现了原子消息和全局有序消息，从而保证组内数据的一致性。

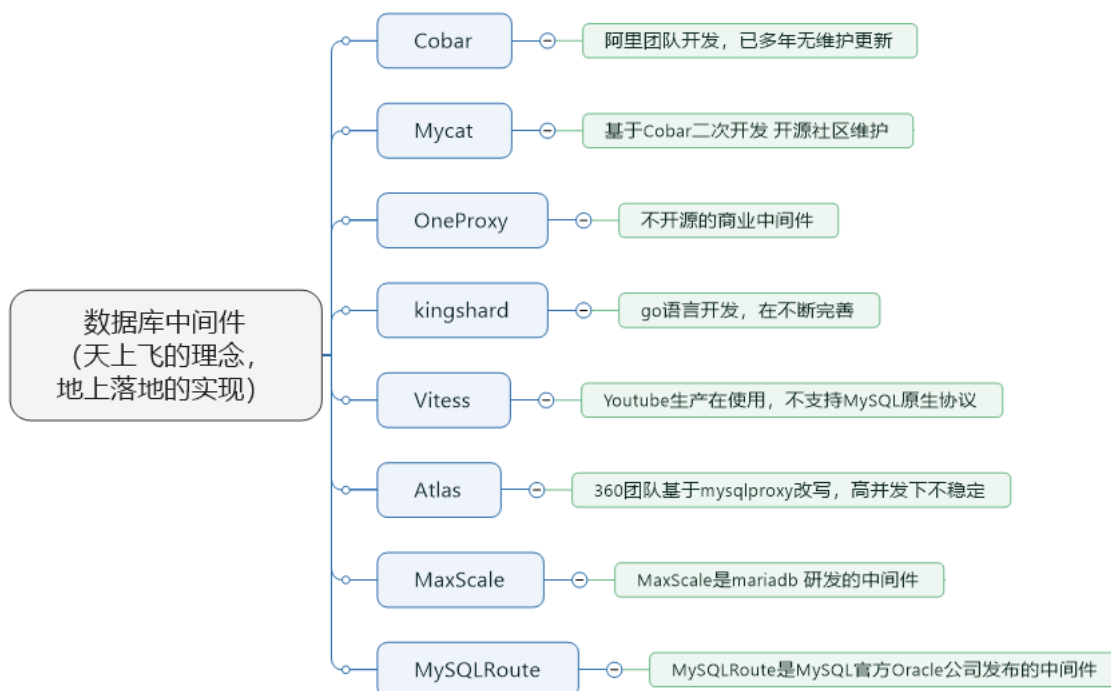


MGR 将 MySQL 带入了数据强一致性的时代，是一个划时代的创新，其中一个重要的原因就是MGR 是基于 Paxos 协议的。Paxos 算法是由 2013 年的图灵奖获得者 Leslie Lamport 于 1990 年提出的，有关这个算法的决策机制可以搜一下。事实上，Paxos 算法提出来之后就作为 **分布式一致性算法** 被广泛应用，比如 Apache 的 ZooKeeper 也是基于 Paxos 实现的。

## 5. 知识延伸

在主从架构的配置中，如果想要采取读写分离的策略，我们可以 **自己编写程序**，也可以通过 **第三方的中间件** 来实现。

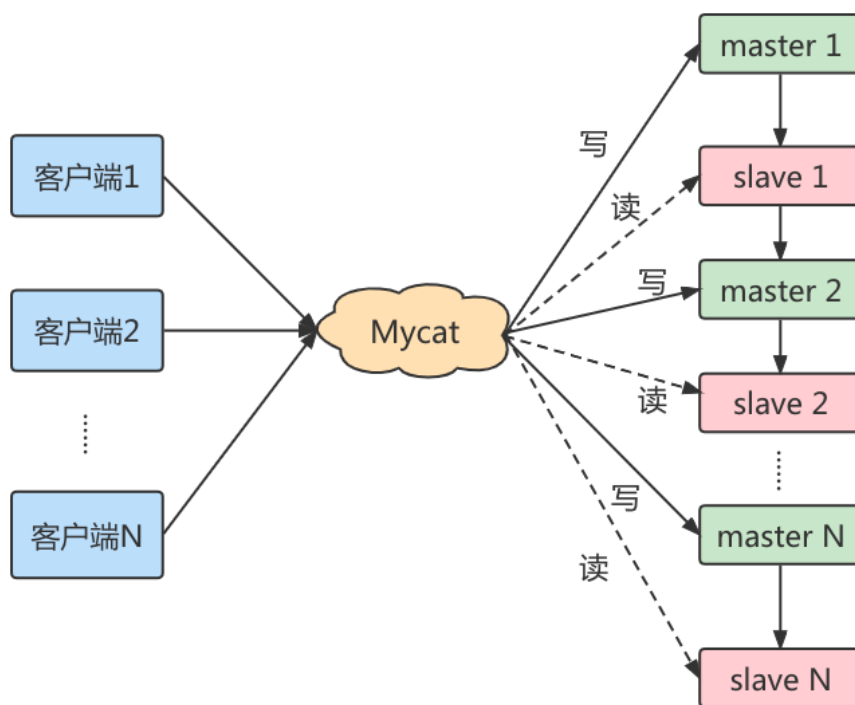
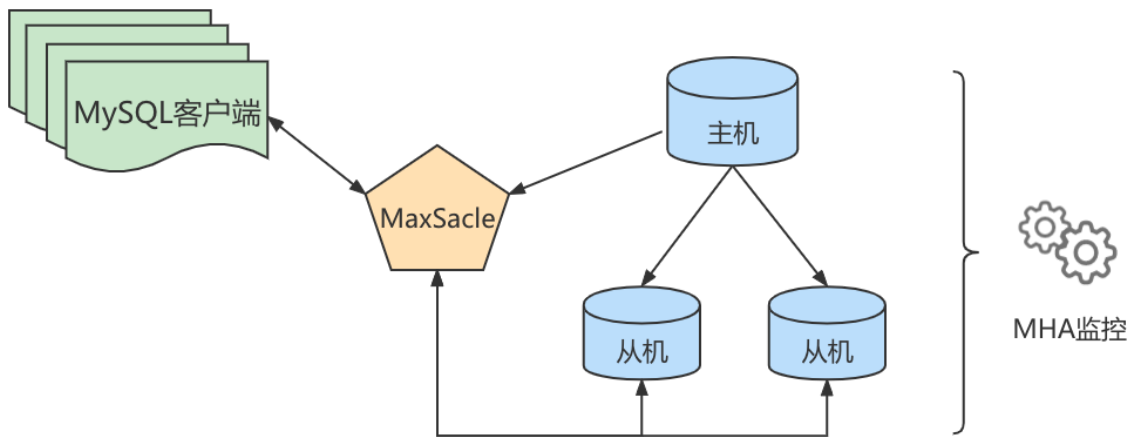
- 自己编写程序的好处就在于比较自主，我们可以自己判断哪些查询在从库上来执行，针对实时性要求高的需求，我们还可以考虑哪些查询可以在主库上执行。同时，程序直接连接数据库，减少了中间件层，相当于减少了性能损耗。
- 采用中间件的方法有很明显的优势，**功能强大**，**使用简单**。但因为在客户端和数据库之间增加了中间件层会有一些 **性能损耗**，同时商业中间件也是有使用成本的。我们也可以考虑采取一些优秀的开源工具。



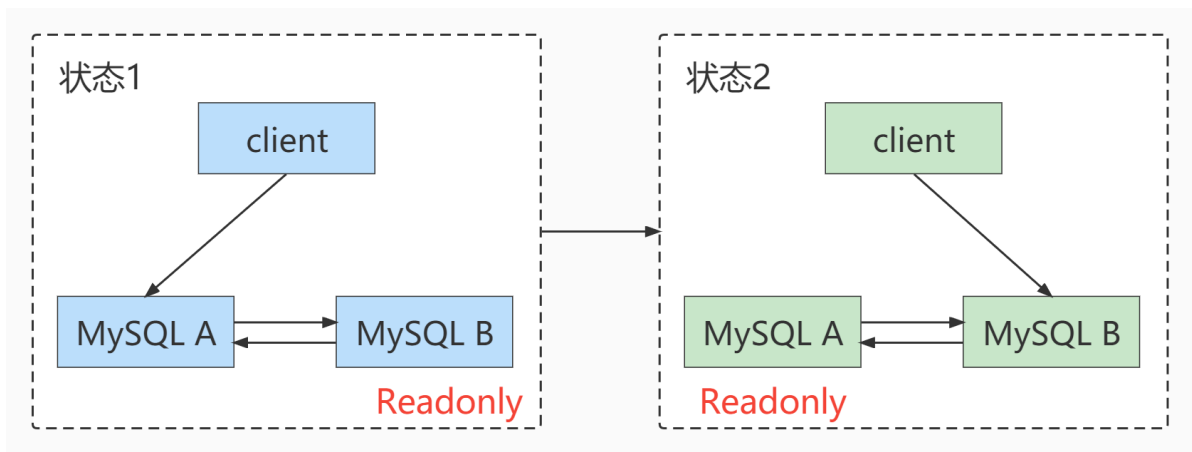
① **Cobar** 属于阿里B2B事业群，始于2008年，在阿里服役3年多，接管3000+个MySQL数据库的 schema,集群日处理在线SQL请求50亿次以上。由于Cobar发起人的离职，Cobar停止维护。

② **Mycat** 是开源社区在阿里cobar基础上进行二次开发，解决了cobar存在的问题，并且加入了许多新的功能在其中。青出于蓝而胜于蓝。

- ③ **OneProxy** 基于MySQL官方的proxy思想利用c语言进行开发的，OneProxy是一款商业 **收费** 的中间件。舍弃了一些功能，专注在 **性能和稳定性上**。
- ④ **kingshard** 由小团队用go语言开发，还需要发展，需要不断完善。
- ⑤ **Vitess** 是Youtube生产在使用，架构很复杂。不支持MySQL原生协议，使用 **需要大量改造成本**。
- ⑥ **Atlas** 是360团队基于mysql proxy改写，功能还需完善，高并发下不稳定。
- ⑦ **MaxScale** 是mariadb（MySQL原作者维护的一个版本）研发的中间件
- ⑧ **MySQLRoute** 是MySQL官方Oracle公司发布的中间件



主备切换:



- 主动切换
- 被动切换
- 如何判断主库出问题了？如何解决过程中的数据不一致性问题？