楼兰
你的神秘技术宝藏

一、MySQL高可用集群介绍

1、数据库主从架构与分库分表

随着现在互联网的应用越来越大，数据库会频繁的成为整个应用的性能瓶颈。而我们经常使用的MySQL数据库，也会不断面临数据量太大、数据访问太频繁、数据读写速度太快等一系列的问题。所以，我们需要设计复杂的应用架构来保护孱弱的数据库，例如添加Redis缓存，增加MQ进行流量削峰等等。但是，数据库本身如果不能得到提升，这就相当于是水桶理论中的最短板。

而要提升数据库的性能，一种思路，当然是对数据库本身进行优化，例如对MySQL进行优化配置，或者干脆换成ClickHouse这一类的针对大数据的产品。另一方面就是跟微服务架构的思路一样，从单体架构升级到集群架构，这样才能真正全方位解放数据库的性能瓶颈。而我们后续要学习的分库分表就是一种非常常见的数据库集群架构管理方案。

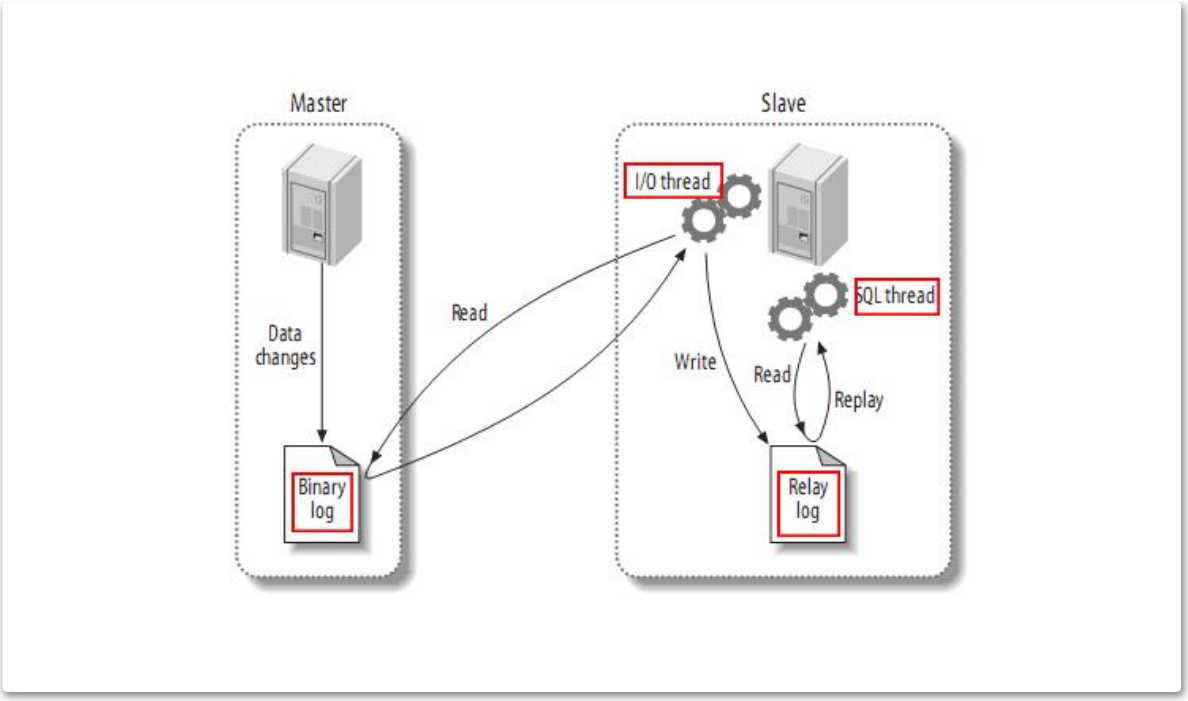
但是就像微服务架构并不是简单的将服务从单机升级为就能一样，分库分表也并不只是字面意义上的将数据分到多个库或者多个表这么简单，他也是基于数据库产品的一系列分布式解决方案。在不同的应用场景下，针对不同的数据库产品，分库分表也有不同的落地方式。而我们后续，会以最为常见的MySQL数据库以及ShardingSphere框架来了解分库分表要如何进行。

2、MySQL主从同步原理

既然要解决MySQL数据库的分布式集群化问题，那就不能不先了解MySQL自身提供的主从同步原理。这是构建MySQL集群的基础，也是后续进行分库分表的基础，更是MySQL进行生产环境部署的基础。

其实数据库的主从同步，就是为了要保证多个数据库之间的数据保持一致。最简单的方式就是使用数据库的导入导出工具，定时将主库的数据导出，再导入到从库当中。这是一种很常见，也很简单易行的数据库集群方式。也有很多的工具帮助我们来做这些事情。但是这种方式进行数据同步的实时性比较差。

而如果要保证数据能够实时同步，对于MySQL，通常就要用到他自身提供的一套通过Binlog日志在多个MySQL服务之间进行同步的集群方案。基于这种集群方案，一方面可以提高数据的安全性，另外也可以以此为基础，提供读写分离、故障转移等其他高级的功能。



即在主库上打开Binlog日志，记录对数据的每一步操作。然后在从库上打开RelayLog日志，用来记录跟主库一样的Binlog日志，并将RelayLog中的操作日志在自己数据库中进行重演。这样就能够更加实时的保证主库与从库的数据一致。

> MySQL的Binlog默认是不打开的。

他的实现过程是在从库上启动一系列IO线程，负责与主库建立TCP连接，请求主库在写入Binlog日志时，也往从库传输一份。这时，主库上会有一个IO Dump线程，负责将Binlog日志通过这些TCP连接传输给从库的IO线程。而从库为了保证日志接收的稳定性，并不会立即重演Binlog数据操作，而是先将接收到的Binlog日志写入到自己的RelayLog日志当中。然后再异步的重演RelayLog中的数据操作。

MySQL的BinLog日志能够比较实时的记录主库上的所有日志操作，因此他也被很多其他工具用来实时监控MySQL的数据变化。例如Canal框架，可以模拟一个slave节点，同步MySQL的Binlog，然后将具体的数据操作按照定制的逻辑进行转发。例如转发到Redis实现缓存一致，转发到Kafka实现数据实时流转等。而ClickHouse也支持将自己模拟成一个MySQL的从节点，接收MySQL的Binlog日志，实时同步MySQL的数据。这个功能目前还在实验阶段。

二、动手搭建MySQL主从集群

1、基础环境搭建

以下实验准备两台服务器，来搭建一个MySQL的主从集群。均安装CentOS7操作系统。 192.168.232.128将作为MySQL主节点，192.168.232.129将作为MySQL的从节点。

然后在两台服务器上均安装MySQL服务，MySQL版本采用mysql-8.0.20版本。

2、安装MySQL服务

这里强调下，我们下面的示例是带大家在Linux上搭建MySQL服务。但是在Linux上安装MySQL经常会遇到各种各样的环境问题，这些环境问题大都只能通过百度加经验的方式来解决。大家根据自己的实际情况，如果在Linux上搭建MySQL有困难的话，可以改为用Windows来安装MySQL。Windows上安装MySQL会简单很多，并且也不影响我们后续ShardingSphere的学习。

1》初始化MySQL

MySQL的安装有很多种方式，具体可以参考官网手册：https://dev.mysql.com/doc/refman/8.0/en/binary-installation.html

我们这里采用对系统环境依赖最低，出问题的可能性最小的tar包方式来安装。

上传mysql压缩包到worker2机器的root用户工作目录/root下，然后按照下面的指令，解压安装mysql

```
groupadd mysql
useradd -r -g mysql -s /bin/false mysql  #这里是创建一个mysql用户用于承载mysql服务，但是不需要登录权限。
tar -zxvf mysql-8.0.20-el7-x86_64.tar.gz #解压
ln -s mysql-8.0.20-el7-x86_64 mysql #建立软链接
cd mysql
mkdir mysql-files
chown mysql:mysql mysql-files
chmod 750 mysql-files
bin/mysqld --initialize --user=mysql #初始化mysql数据文件 注意点1
bin/mysql_ssl_rsa_setup
bin/mysqld_safe --user=mysql

cp support-files/mysql.server /etc/init.d/mysql.server
```

> **注意点：**

> 1、初始化过程中会初始化一些mysql的数据文件，经常会出现一些文件或者文件夹权限不足的问题。如果有文件权限不足的问题，需要根据他的报错信息，创建对应的文件或者文件夹，并配置对应的文件权限。
>
> 2、初始化过程如果正常完成，日志中会打印出一个root用户的默认密码。这个密码需要记录下来。

```
2020-12-10T06:05:28.948043Z 6 [Note] [MY-010454] [Server] A temporary password is generated for root@localhost: P6kigsT6Lg>=
```

2》启动mysql

```
bin/mysqld --user=mysql
```

> **注意点：**
>
> 1、这个启动过程会独占当前命令行窗口，如果要后台执行可以在后面添加一个 &。但是一般第一次启动mysql服务时，经常会出现一些错误，所以建议用独占窗口的模式跟踪下日志。
>
> Linux上安装软件经常会出现各种各样的环境问题，很难全部概括。大部分的问题，需要查百度，根据别人的经验来修改。如果安装有困难的同学，可以改为在Windows上安装MySQL，整个过程会简单很多，不会影响后续ShardingSpehre的学习。

3、连接MySQL

MySQL服务启动完成后，默认是只能从本机登录，远程是无法访问的。所以需要用root用户登录下，配置远程访问的权限。

```
cd /root/mysql
bin/mysql -uroot -p #然后用之前记录的默认密码登录
```

> **注意点：**
>
> 1、如果有同学遇到 **ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)** 这个报错信息，可以参照下面的配置，修改下/etc/my.cnf配置文件，来配置下sock连接文件的地址。主要是下面client部分。

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0
# Settings user and group are ignored when systemd is used.
# If you need to run mysqld under a different user or group,
# customize your systemd unit file for mariadb according to the
# instructions in http://fedoraproject.org/wiki/Systemd

[mysqld_safe]
log-error=/var/log/mariadb/mariadb.log
pid-file=/var/run/mariadb/mariadb.pid

#
# include all files from the config directory
#
!includedir /etc/my.cnf.d

[client]
port=3306
socket=/var/lib/mysql/mysql.sock
```

登录进去后，需要配置远程登录权限：

```
alter user 'root'@'localhost' identified by '123456'; #修改root用户的密码
use mysql;
update user set host='%' where user='root';
flush privileges;
```

这样，Linux机器上的MySQL服务就搭建完成了。可以使用navicat等连接工具远程访问MySQL服务了。

> 然后如果有同学安装MySQL确实有问题的话，推荐大家可以使用宝塔面板。https://www.bt.cn/ 。使用这个工具可以图形化安装以及管理MySQL，非常方便。
>
> 另外，对于熟悉Docker和K8s的同学，可以用这些虚拟化的方式来搭建，也非常简单高效。

这里需要注意下的是，搭建主从集群的多个服务，有两个必要的条件。

1、MySQL版本必须一致。

2、集群中各个服务器的时间需要同步。

3、搭建主从集群

接下来在这两个MySQL服务基础上，搭建一个主从集群。

1》配置master主服务

首先，配置主节点的mysql配置文件： /etc/my.cnf(没有的话就手动创建一个)

这一步需要对master进行配置，主要是需要打开binlog日志，以及指定severId。我们打开MySQL主服务的my.cnf文件，在文件中一行server-id以及一个关闭域名解析的配置。然后重启服务。

```
[mysqld]
server-id=47
#开启binlog
log_bin=master-bin
log_bin-index=master-bin.index
skip-name-resolve
# 设置连接端口
port=3306
# 设置mysql的安装目录
basedir=/usr/local/mysql
# 设置mysql数据库的数据的存放目录
datadir=/usr/local/mysql/mysql-files
# 允许最大连接数
max_connections=200
# 允许连接失败的次数。
max_connect_errors=10
# 服务端使用的字符集默认为UTF8
character-set-server=utf8
# 创建新表时将使用的默认存储引擎
default-storage-engine=INNODB
# 默认使用“mysql_native_password”插件认证
#mysql_native_password
default_authentication_plugin=mysql_native_password
```

> 配置说明：主要需要修改的是以下几个属性：
>
> server-id：服务节点的唯一标识。需要给集群中的每个服务分配一个单独的ID。
>
> log_bin：打开Binlog日志记录，并指定文件名。

> log_bin-index：Binlog日志文件

重启MySQL服务， `service mysqld restart`

然后，我们需要给root用户分配一个replication slave的权限。

```
#登录主数据库
mysql -u root -p
GRANT REPLICATION SLAVE ON *.* TO 'root'@'%';
flush privileges;
#查看主节点同步状态：
show master status;
```

> 在实际生产环境中，通常不会直接使用root用户，而会创建一个拥有全部权限的用户来负责主从同步。

```
mysql> show master status;
+------------------+----------+--------------+------------------+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------+
| master-bin.000004 |      156 |              |                  |                   |
+------------------+----------+--------------+------------------+-------------------+
1 row in set (0.00 sec)
```

这个指令结果中的File和Position记录的是当前日志的binlog文件以及文件中的索引。

而后面的Binlog_Do_DB和Binlog_Ignore_DB这两个字段是表示需要记录binlog文件的库以及不需要记录binlog文件的库。目前我们没有进行配置，就表示是针对全库记录日志。这两个字段如何进行配置，会在后面进行介绍。

> 开启binlog后，数据库中的所有操作都会被记录到datadir当中，以一组轮询文件的方式循环记录。而指令查到的File和Position就是当前日志的文件和位置。而在后面配置从服务时，就需要通过这个File和Position通知从服务从哪个地方开始记录binLog。

```
[root@worker1 mysql-files]# ll
总用量 179660
-rw-r----- 1 mysql mysql       56 12月  30 14:24 auto.cnf
-rw------- 1 mysql mysql     1676 12月  30 14:24 ca-key.pem
-rw-r--r-- 1 mysql mysql     1112 12月  30 14:24 ca.pem
-rw-r--r-- 1 mysql mysql     1112 12月  30 14:24 client-cert.pem
-rw------- 1 mysql mysql     1676 12月  30 14:24 client-key.pem
-rw-r----- 1 mysql mysql   196608 12月  30 14:33 #ib_16384_0.dblwr
-rw-r----- 1 mysql mysql  8585216 12月  30 14:24 #ib_16384_1.dblwr
-rw-r----- 1 mysql mysql     3768 12月  30 14:32 ib_buffer_pool
-rw-r----- 1 mysql mysql 12582912 12月  30 14:33 ibdata1
-rw-r----- 1 mysql mysql 50331648 12月  30 14:33 ib_logfile0
-rw-r----- 1 mysql mysql 50331648 12月  30 14:24 ib_logfile1
-rw-r----- 1 mysql mysql 12582912 12月  30 14:32 ibtmp1
drwxr-x--- 2 mysql mysql      187 12月  30 14:32 #innodb_temp
-rw-r----- 1 mysql mysql      179 12月  30 14:32 master-bin.000001
-rw-r----- 1 mysql mysql     1829 12月  30 14:33 master-bin.000002
-rw-r----- 1 mysql mysql       40 12月  30 14:32 master-bin.index
drwxr-x--- 2 mysql mysql      143 12月  30 14:24 mysql
-rw-r----- 1 mysql mysql 28311552 12月  30 14:33 mysql.ibd
drwxr-x--- 2 mysql mysql     8192 12月  30 14:24 performance_schema
-rw------- 1 mysql mysql     1676 12月  30 14:24 private_key.pem
-rw-r--r-- 1 mysql mysql      452 12月  30 14:24 public_key.pem
-rw-r--r-- 1 mysql mysql     1112 12月  30 14:24 server-cert.pem
-rw------- 1 mysql mysql     1680 12月  30 14:24 server-key.pem
drwxr-x--- 2 mysql mysql       28 12月  30 14:24 sys
-rw-r----- 1 mysql mysql 10485760 12月  30 14:33 undo_001
-rw-r----- 1 mysql mysql 10485760 12月  30 14:33 undo_002
-rw-r----- 1 mysql mysql     2200 12月  30 14:32 worker1.err
-rw-r----- 1 mysql mysql        5 12月  30 14:32 worker1.pid
[root@worker1 mysql-files]# pwd
/usr/local/mysql/mysql-files
```

2》配置slave从服务

下一步，我们来配置从服务mysqls。 我们打开mysqls的配置文件my.cnf，修改配置文件：

```
[mysqld]
#主库和从库需要不一致
server-id=48
#打开MySQL中继日志
relay-log-index=slave-relay-bin.index
relay-log=slave-relay-bin
#打开从服务二进制日志
log-bin=mysql-bin
#使得更新的数据写进二进制日志中
log-slave-updates=1
# 设置3306端口
port=3306
# 设置mysql的安装目录
basedir=/usr/local/mysql
# 设置mysql数据库的数据的存放目录
datadir=/usr/local/mysql/mysql-files
# 允许最大连接数
max_connections=200
# 允许连接失败的次数。
max_connect_errors=10
# 服务端使用的字符集默认为UTF8
character-set-server=utf8
```

```
# 创建新表时将使用的默认存储引擎
default-storage-engine=INNODB
# 默认使用“mysql_native_password”插件认证
#mysql_native_password
default_authentication_plugin=mysql_native_password
```

> 配置说明：主要需要关注的几个属性：
>
> server-id：服务节点的唯一标识
>
> relay-log：打开从服务的relay-log日志。
>
> log-bin：打开从服务的bin-log日志记录。

然后我们启动mysqls的服务，并设置他的主节点同步状态。

```
#登录从服务
mysql -u root -p;
#设置同步主节点：
CHANGE MASTER TO
MASTER_HOST='192.168.232.128',
MASTER_PORT=3306,
MASTER_USER='root',
MASTER_PASSWORD='root',
MASTER_LOG_FILE='master-bin.000004',
MASTER_LOG_POS=156,
GET_MASTER_PUBLIC_KEY=1;
#开启slave
start slave;
#查看主从同步状态
show slave status;
或者用 show slave status \G; 这样查看比较简洁
```

> 注意，CHANGE MASTER指令中需要指定的MASTER_LOG_FILE和MASTER_LOG_POS必须与主服务中查到的保持一致。
>
> 并且后续如果要检查主从架构是否成功，也可以通过检查主服务与从服务之间的File和Position这两个属性是否一致来确定。

```
mysql> show slave status \G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 127.0.0.1
                  Master_User: root
                  Master_Port: 3307
                Connect_Retry: 60
              Master_Log_File: master-bin.000004
          Read_Master_Log_Pos: 156
               Relay_Log_File: slave-relay-bin.000006
                Relay_Log_Pos: 373
        Relay_Master_Log_File: master-bin.000004
             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
```

我们重点关注其中红色方框的两个属性，与主节点保持一致，就表示这个主从同步搭建是成功的。

> 从这个指令的结果能够看到，有很多Replicate_开头的属性，这些属性指定了两个服务之间要同步哪些数据库、哪些表的配置。只是在我们这个示例中全都没有进行配置，就标识是全库进行同步。后面我们会补充如何配置需要同步的库和表。

3》主从集群测试

测试时，我们先用showdatabases，查看下两个MySQL服务中的数据库情况

主服务

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| masterdemo         |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

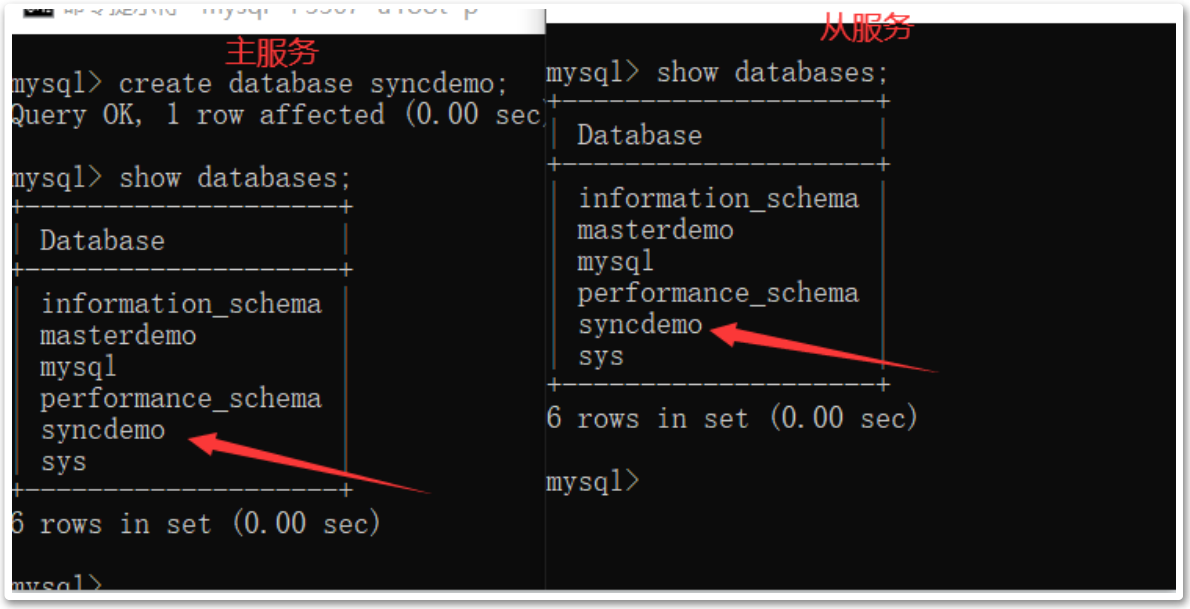
从服务

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| masterdemo         |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql>
```

然后我们在主服务器上创建一个数据库

```
mysql> create database syncdemo;
Query OK, 1 row affected (0.00 sec)
```

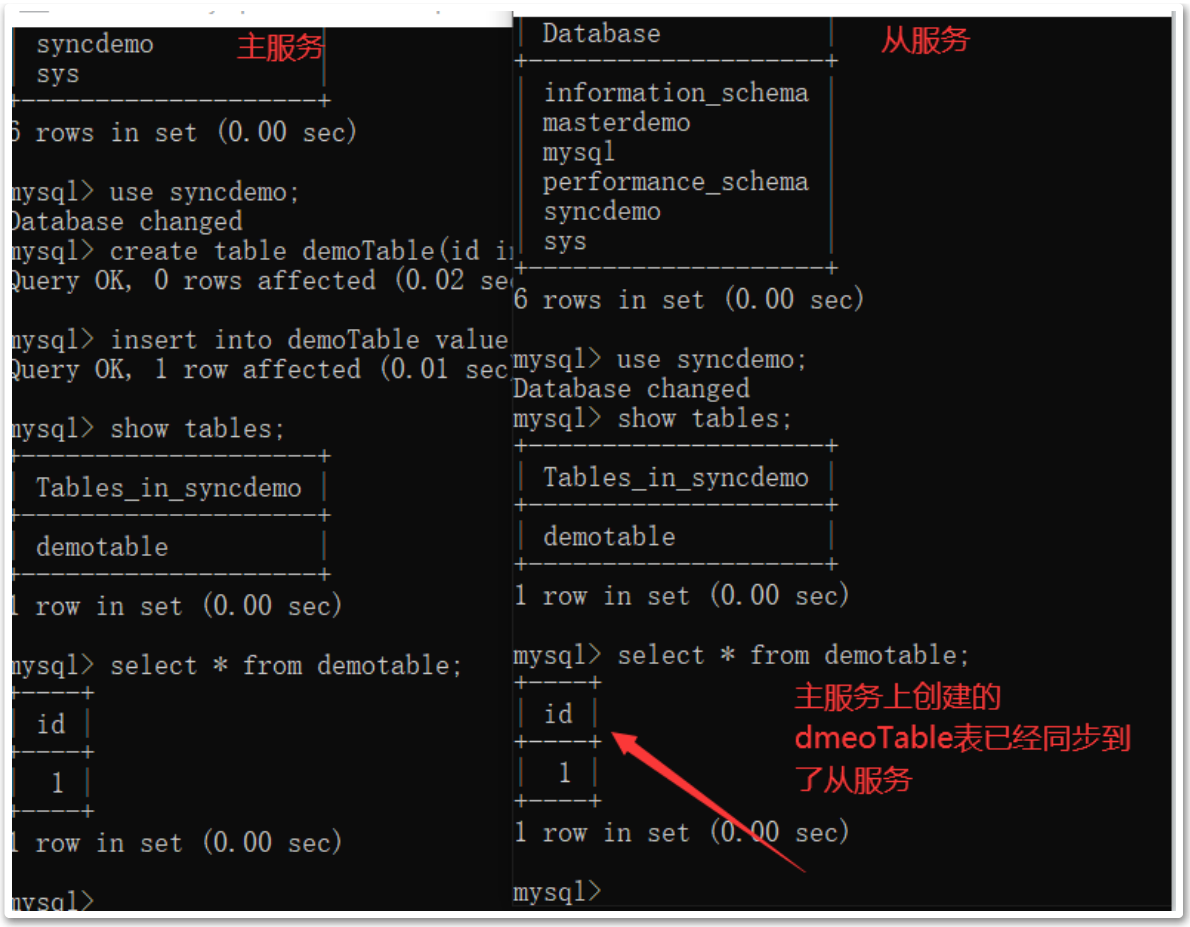
然后我们再用show databases，来看下这个syncdemo的数据库是不是已经同步到了从服务。



接下来我们继续在syncdemo这个数据库中创建一个表，并插入一条数据。

```
mysql> use syncdemo;
Database changed
mysql> create table demoTable(id int not null);
Query OK, 0 rows affected (0.02 sec)

mysql> insert into demoTable value(1);
Query OK, 1 row affected (0.01 sec)
```

然后我们也同样到主服务与从服务上都来查一下这个demoTable是否同步到了从服务。



从上面的实验过程看到，我们在主服务中进行的数据操作，就都已经同步到了从服务上。这样，我们一个主从集群就搭建完成了。

> 另外，这个主从架构是有可能失败的，如果在slave从服务上查看slave状态，发现Slave_SQL_Running=no，就表示主从同步失败了。这有可能是因为在从数据库上进行了写操作，与同步过来的SQL操作冲突了，也有可能是slave从服务重启后有事务回滚了。
>
> 如果是因为slave从服务事务回滚的原因，可以按照以下方式重启主从同步：

```
mysql> stop slave ;
mysql> set GLOBAL SQL_SLAVE_SKIP_COUNTER=1;
mysql> start slave ;
```

> 而另一种解决方式就是重新记录主节点的binlog文件消息

```
mysql> stop slave ;
mysql> change master to .....
mysql> start slave ;
```

> 但是这种方式要注意binlog的文件和位置，如果修改后和之前的同步接不上，那就会丢失部分数据。所以不太常用。

4》全库同步与部分同步：

在完成这个基本的MySQL主从集群后，我们还可以进行后续的实验：

之前提到，我们目前配置的主从同步是针对全库配置的，而实际环境中，一般并不需要针对全库做备份，而只需要对一些特别重要的库或者表来进行同步。那如何针对库和表做同步配置呢?

首先在Master端：在my.cnf中，可以通过以下这些属性指定需要针对哪些库或者哪些表记录binlog

```
#需要同步的二进制数据库名
binlog-do-db=masterdemo
#只保留7天的二进制日志，以防磁盘被日志占满(可选)
expire-logs-days  = 7
#不备份的数据库
binlog-ignore-db=information_schema
binlog-ignore-db=performation_schema
binlog-ignore-db=sys
```

然后在Slave端：在my.cnf中，需要配置备份库与主服务的库的对应关系。

```
#如果salve库名称与master库名相同，使用本配置
replicate-do-db = masterdemo
#如果master库名[mastdemo]与salve库名[mastdemo01]不同，使用以下配置[需要做映射]
replicate-rewrite-db = masterdemo -> masterdemo01
#如果不是要全部同步[默认全部同步]，则指定需要同步的表
replicate-wild-do-table=masterdemo01.t_dict
replicate-wild-do-table=masterdemo01.t_num
```

配置完成了之后，在show master status指令中，就可以看到Binlog_Do_DB和Binlog_Ignore_DB两个参数的作用了。

5》GTID同步集群

上面我们搭建的集群方式，是基于Binlog日志记录点的方式来搭建的，这也是最为传统的MySQL集群搭建方式。而在这个实验中，可以看到有一个Executed_Grid_Set列，暂时还没有用上。实际上，这就是另外一种搭建主从同步的方式，即GTID搭建方式。这种模式是从MySQL5.6版本引入的。

GTID的本质也是基于Binlog来实现主从同步，只是他会基于一个全局的事务ID来标识同步进度。GTID即全局事务ID，全局唯一并且趋势递增，他可以保证为每一个在主节点上提交的事务在复制集群中可以生成一个唯一的ID 。

在基于GTID的复制中，首先从服务器会告诉主服务器已经在从服务器执行完了哪些事务的GTID值，然后主库会有把所有没有在从库上执行的事务，发送到从库上进行执行，并且使用GTID的复制可以保证同一个事务只在指定的从库上执行一次，这样可以避免由于偏移量的问题造成数据不一致。

他的搭建方式跟我们上面的主从架构整体搭建方式差不多。只是需要在my.cnf中修改一些配置。

在主节点上：

```
gtid_mode=on
enforce_gtid_consistency=on
log_bin=on
server_id=单独设置一个
binlog_format=row
```

在从节点上：

```
gtid_mode=on
enforce_gtid_consistency=on
log_slave_updates=1
server_id=单独设置一个
```

然后分别重启主服务和从服务，就可以开启GTID同步复制方式。

4、集群扩容与MySQL数据迁移

我们现在已经搭建成功了一主一从的MySQL集群架构，那要扩展到一主多从的集群架构，其实就比较简单了，只需要增加一个binlog复制就行了。

但是如果我们的集群是已经运行过一段时间，这时候如果要扩展新的从节点就有一个问题，之前的数据没办法从binlog来恢复了。这时候在扩展新的slave节点时，就需要增加一个数据复制的操作。

MySQL的数据备份恢复操作相对比较简单，可以通过SQL语句直接来完成。具体操作可以使用mysql的bin目录下的mysqldump工具。

```
mysqldump -u root -p --all-databases > backup.sql
#输入密码
```

通过这个指令，就可以将整个数据库的所有数据导出成backup.sql，然后把这个backup.sql分发到新的MySQL服务器上，并执行下面的指令将数据全部导入到新的MySQL服务中。

```
mysql -u root -p < backup.sql
#输入密码
```

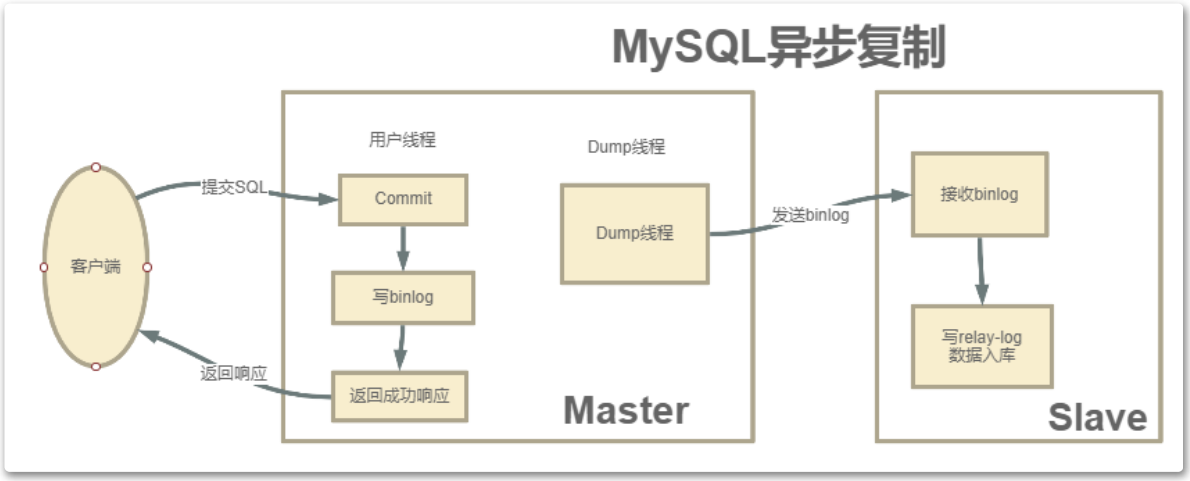
这样新的MySQL服务就已经有了所有的历史数据，然后就可以再按照上面的步骤，配置Slave从服务的数据同步了。

5、搭建半同步复制

1》理解半同步复制

到现在为止，我们已经可以搭建MySQL的主从集群，互主集群，但是我们这个集群有一个隐患，就是有可能会丢数据。这是为什么呢？这要从MySQL主从数据复制分析起。

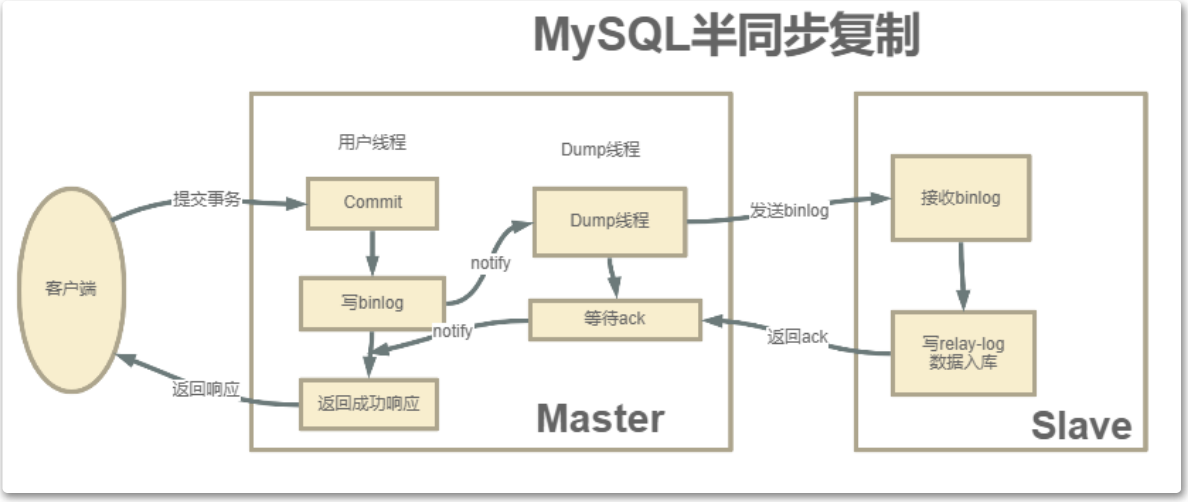
MySQL主从集群默认采用的是一种异步复制的机制。主服务在执行用户提交的事务后，写入binlog日志，然后就给客户端返回一个成功的响应了。而binlog会由一个dump线程异步发送给Slave从服务。



由于这个发送binlog的过程是异步的。主服务在向客户端反馈执行结果时，是不知道binlog是否同步成功了的。这时候如果主服务宕机了，而从服务还没有备份到新执行的binlog，那就有可能会丢数据。

那怎么解决这个问题呢，这就要靠MySQL的半同步复制机制来保证数据安全。

半同步复制机制是一种介于异步复制和全同步复制之前的机制。主库在执行完客户端提交的事务后，并不是立即返回客户端响应，而是等待至少一个从库接收并写到relay log中，才会返回给客户端。MySQL在等待确认时，默认会等10秒，如果超过10秒没有收到ack，就会降级成为异步复制。



这种半同步复制相比异步复制，能够有效的提高数据的安全性。但是这种安全性也不是绝对的，他只保证事务提交后的binlog至少传输到了一个从库，并且并不保证从库应用这个事务的binlog是成功的。另一方面，半同步复制机制也会造成一定程度的延迟，这个延迟时间最少是一个TCP/IP请求往返的时间。整个服务的性能是会有所下降的。而当从服务出现问题时，主服务需要等待的时间就会更长，要等到从服务的服务恢复或者请求超时才能给用户响应。

2》搭建半同步复制集群

半同步复制需要基于特定的扩展模块来实现。而mysql从5.5版本开始，往上的版本都默认自带了这个模块。这个模块包含在mysql安装目录下的lib/plugin目录下的semisync_master.so和semisync_slave.so两个文件中。需要在主服务上安装semisync_master模块，在从服务上安装semisync_slave模块。

```
[root@worker1 plugin]# pwd
/usr/local/mysql/lib/plugin
[root@worker1 plugin]# ll
总用量 90952
-rwxr-xr-x 1 7161 31415   166168 3月   26 2020 adt_null.so
-rwxr-xr-x 1 7161 31415   639696 3月   26 2020 authentication_ldap_sasl_client.so
-rwxr-xr-x 1 7161 31415   113784 3月   26 2020 auth_socket.so
-rwxr-xr-x 1 7161 31415   271816 3月   26 2020 component_audit_api_message_emit.so
-rwxr-xr-x 1 7161 31415   192920 3月   26 2020 component_log_filter_dragnet.so
-rwxr-xr-x 1 7161 31415    36968 3月   26 2020 component_log_sink_json.so
-rwxr-xr-x 1 7161 31415   146024 3月   26 2020 component_log_sink_syseventlog.so
-rwxr-xr-x 1 7161 31415   566016 3月   26 2020 component_mysqlbackup.so
-rwxr-xr-x 1 7161 31415   510288 3月   26 2020 component_validate_password.so
-rwxr-xr-x 1 7161 31415  1311200 3月   26 2020 connection_control.so
-rwxr-xr-x 1 7161 31415  3183232 3月   26 2020 ddl_rewriter.so
drwxr-xr-x 2 7161 31415     4096 3月   26 2020 debug
-rwxr-xr-x 1 7161 31415 60459336 3月   26 2020 group_replication.so
-rwxr-xr-x 1 7161 31415   633896 3月   26 2020 ha_example.so
-rwxr-xr-x 1 7161 31415  1303856 3月   26 2020 ha_mock.so
-rwxr-xr-x 1 7161 31415  1229744 3月   26 2020 innodb_engine.so
-rwxr-xr-x 1 7161 31415  2648040 3月   26 2020 keyring_file.so
-rwxr-xr-x 1 7161 31415   518872 3月   26 2020 keyring_udf.so
-rwxr-xr-x 1 7161 31415  1208848 3月   26 2020 libmemcached.so
-rwxr-xr-x 1 7161 31415  9064128 3月   26 2020 libpluginmecab.so
-rwxr-xr-x 1 7161 31415    91000 3月   26 2020 locking_service.so
-rwxr-xr-x 1 7161 31415   115064 3月   26 2020 mypluglib.so
-rwxr-xr-x 1 7161 31415  2675360 3月   26 2020 mysql_clone.so
-rwxr-xr-x 1 7161 31415   112048 3月   26 2020 mysql_no_login.so
-rwxr-xr-x 1 7161 31415   114176 3月   26 2020 rewrite_example.so
-rwxr-xr-x 1 7161 31415  1570552 3月   26 2020 rewriter.so
-rwxr-xr-x 1 7161 31415  1646120 3月   26 2020 semisync_master.so
-rwxr-xr-x 1 7161 31415   750408 3月   26 2020 semisync_slave.so
-rwxr-xr-x 1 7161 31415   438736 3月   26 2020 validate_password.so
-rwxr-xr-x 1 7161 31415  1340600 3月   26 2020 version_token.so
```

首先我们登陆主服务，安装semisync_master模块：

```
mysql> install plugin rpl_semi_sync_master soname 'semisync_master.so';
Query OK, 0 rows affected (0.01 sec)

mysql> show global variables like 'rpl_semi%';
+-----------------------------------------------+------------+
| Variable_name                                 | Value      |
+-----------------------------------------------+------------+
| rpl_semi_sync_master_enabled                  | OFF        |
| rpl_semi_sync_master_timeout                  | 10000      |
| rpl_semi_sync_master_trace_level              | 32         |
| rpl_semi_sync_master_wait_for_slave_count     | 1          |
| rpl_semi_sync_master_wait_no_slave            | ON         |
| rpl_semi_sync_master_wait_point               | AFTER_SYNC |
+-----------------------------------------------+------------+
6 rows in set, 1 warning (0.02 sec)

mysql> set global rpl_semi_sync_master_enabled=ON;
Query OK, 0 rows affected (0.00 sec)
```

> 这三行指令中，第一行是通过扩展库来安装半同步复制模块，需要指定扩展库的文件名。

> 第二行查看系统全局参数，rpl_semi_sync_master_timeout就是半同步复制时等待应答的最长等待时间，默认是10秒，可以根据情况自行调整。
>
> 第三行则是打开半同步复制的开关。
>
> 在第二行查看系统参数时，最后的一个参数rpl_semi_sync_master_wait_point其实表示一种半同步复制的方式。
>
> 半同步复制有两种方式，一种是我们现在看到的这种默认的AFTER_SYNC方式。这种方式下，主库把日志写入binlog，并且复制给从库，然后开始等待从库的响应。从库返回成功后，主库再提交事务，接着给客户端返回一个成功响应。
>
> 而另一种方式是叫做AFTER_COMMIT方式。他不是默认的。这种方式，在主库写入binlog后，等待binlog复制到从库，主库就提交自己的本地事务，再等待从库返回给自己一个成功响应，然后主库再给客户端返回响应。

然后我们登陆从服务，安装smeisync_slave模块

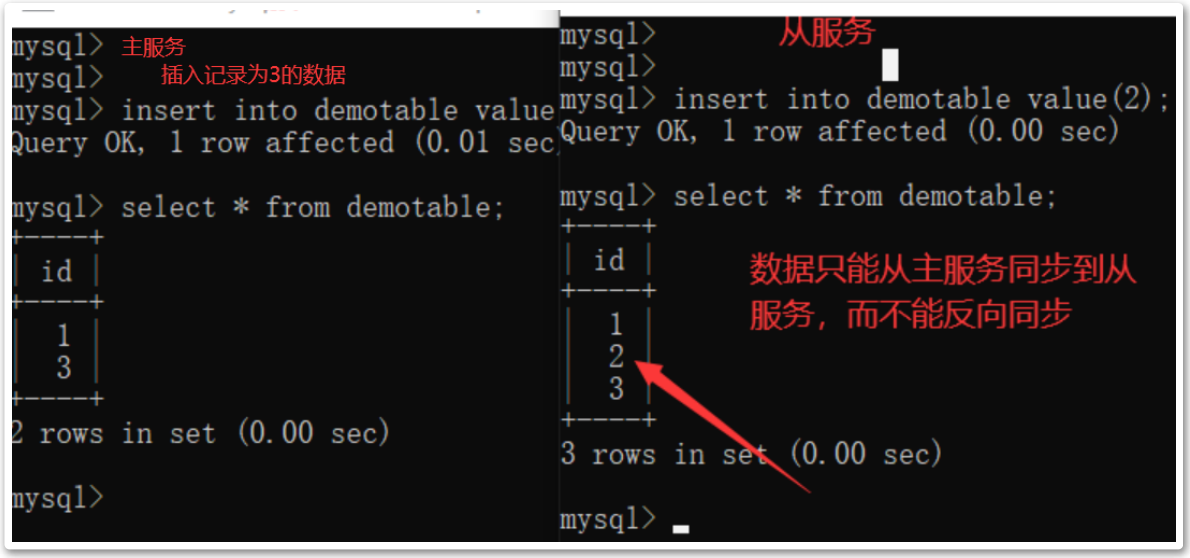
```
mysql> install plugin rpl_semi_sync_slave soname 'semisync_slave.so';
Query OK, 0 rows affected (0.01 sec)

mysql> show global variables like 'rpl_semi%';
+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| rpl_semi_sync_slave_enabled     | OFF   |
| rpl_semi_sync_slave_trace_level | 32    |
+---------------------------------+-------+
2 rows in set, 1 warning (0.01 sec)

mysql> set global rpl_semi_sync_slave_enabled = on;
Query OK, 0 rows affected (0.00 sec)

mysql> show global variables like 'rpl_semi%';
+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| rpl_semi_sync_slave_enabled     | ON    |
| rpl_semi_sync_slave_trace_level | 32    |
+---------------------------------+-------+
2 rows in set, 1 warning (0.00 sec)
```

```
24
25 mysql> stop slave;
26 Query OK, 0 rows affected (0.01 sec)
27
28 mysql> start slave;
29 Query OK, 0 rows affected (0.01 sec)
```

> slave端的安装过程基本差不多，不过要注意下安装完slave端的半同步插件后，需要重启下slave服务。

6、主从集群与读写分离

我们要注意，目前我们的这个MySQL主从集群是单向的，也就是只能从主服务同步到从服务，而从服务的数据表更是无法同步到主服务的。



所以，在这种架构下，为了保证数据一致，通常会需要保证数据只在主服务上写，而从服务只进行数据读取。这个功能，就是大名鼎鼎的读写分离。但是这里要注意下，mysql主从本身是无法提供读写分离的服务的，需要由业务自己来实现。这也是我们后面要学的ShardingSphere的一个重要功能。

> 到这里可以看到，在MySQL主从架构中，是需要严格限制从服务的数据写入的，一旦从服务有数据写入，就会造成数据不一致。并且从服务在执行事务期间还很容易造成数据同步失败。
>
> 如果需要限制用户写数据，我们可以在从服务中将read_only参数的值设为1(`set global read_only=1;`)。这样就可以限制用户写入数据。
> 但是这个属性有两个需要注意的地方：

> 1、read_only=1设置的只读模式，不会影响slave同步复制的功能。 所以在MySQL slave库中设定了read_only=1后，通过 "show slave status\G" 命令查看salve状态，可以看到salve仍然会读取master上的日志，并且在slave库中应用日志，保证主从数据库同步一致；
>
> 2、read_only=1设置的只读模式， 限定的是普通用户进行数据修改的操作，但不会限定具有super权限的用户的数据修改操作。 在MySQL中设置read_only=1后，普通的应用用户进行insert、update、delete等会产生数据变化的DML操作时，都会报出数据库处于只读模式不能发生数据变化的错误，但具有super权限的用户，例如在本地或远程通过root用户登录到数据库，还是可以进行数据变化的DML操作； 如果需要限定super权限的用户写数据，可以设置super_read_only=0。另外 **如果要想连super权限用户的写操作也禁止，就使用"flush tables with read lock;"，这样设置也会阻止主从同步复制！**

7、扩展更复杂的集群结构

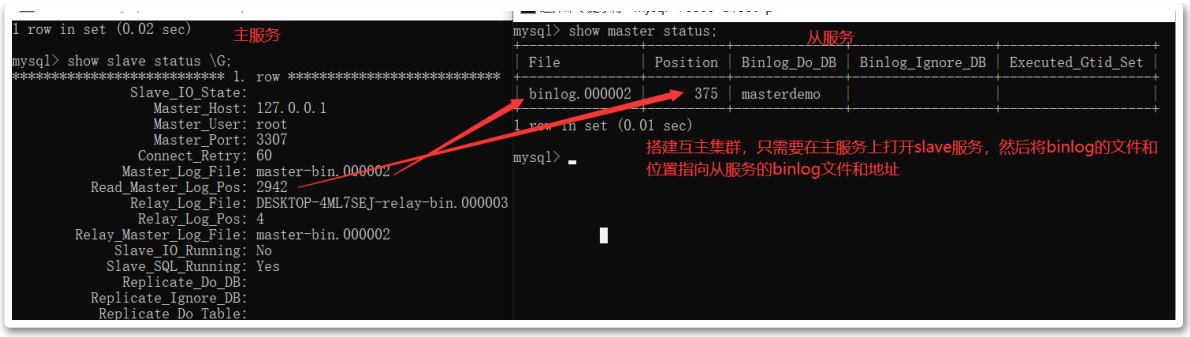
我们到这里搭建出了一个一主一从的MySQL主从同步集群，具有了数据同步的基础功能。而在生产环境中，通常会以此为基础，根据业务情况以及负载情况，搭建更大更复杂的集群。

例如**为了进一步提高整个集群的读能力，可以扩展出一主多从。**而为了减轻主节点进行数据同步的压力，可以继续扩展出多级从的主从集群。

而**为了提高这个集群的写能力，可以搭建互主集群**，即两个服务互为主从。这样不管写到哪个服务上，集群内的数据都是同步的。这样就可以用一个集群来分担写数据的压力。

以此为基础，可以扩展出多主多从的集群，全方位提升集群的数据读写能力。甚至，我们也可以扩展出环形的主从集群，实现MySQL多活部署。

搭建互主集群只需要按照上面的方式，在主服务上打开一个slave进程，并且指向slave节点的binlog当前文件地址和位置。



另外，在我们搭建的这个主从集群中，有一个比较隐藏的问题，就是这样的主从复制之间会有延迟。这在复杂集群中，做了读写分离后，会更容易体现出来。即数据往主服务写，而读数据在从服务读。这时候这个主从复制延迟就有可能造成主库上刚插入了数据但是从库查不到。当然，这在我们目前的这个集群中是很难出现的，但是在大型集群中会很容易出现。

出现这个问题的根本在于：面向业务的主服务数据都是多线程并发写入的，而从服务是单个线程慢慢拉取binlog，这中间就会有个效率差。所以解决这个问题的关键是要让从服务也用多线程并行复制binlog数据。

MySQL自5.7版本后就已经支持并行复制了。可以在从服务上设置slave_parallel_workers为一个大于0的数，然后把slave_parallel_type参数设置为LOGICAL_CLOCK，这就可以了。

```
mysql> show global variables like 'slave_parallel%';
+------------------------+----------+
| Variable_name          | Value    |
+------------------------+----------+
| slave_parallel_type    | DATABASE |
| slave_parallel_workers | 0        |
+------------------------+----------+
2 rows in set (0.01 sec)
```

三、了解MySQL的其他高可用方案

我们之前的MySQL服务集群，都是使用MySQL自身的功能来搭建的集群。但是这样的集群，不具备高可用的功能。即如果是MySQL主服务挂了，从服务是没办法自动切换成主服务的。而如果要实现MySQL的高可用，需要借助一些第三方工具来实现。

> 这一部分方案只需要了解即可，因为一方面，这些高可用方案通常都是运维需要考虑的事情，作为开发人员没有必要花费太多的时间精力，偶尔需要用到的时候能够用起来就够了。另一方面，随着业界技术的不断推进，也会冒出更多的新方案。例如ShardingSphere的5.x版本的目标实际上就是将ShardingSphere由一个数据库中间件升级成一个独立的数据库平台，而将MySQL、PostGreSql甚至是RocksDB这些组件作为数据库的后端支撑。等到新版本成熟时，又会冒出更多新的高可用方案。

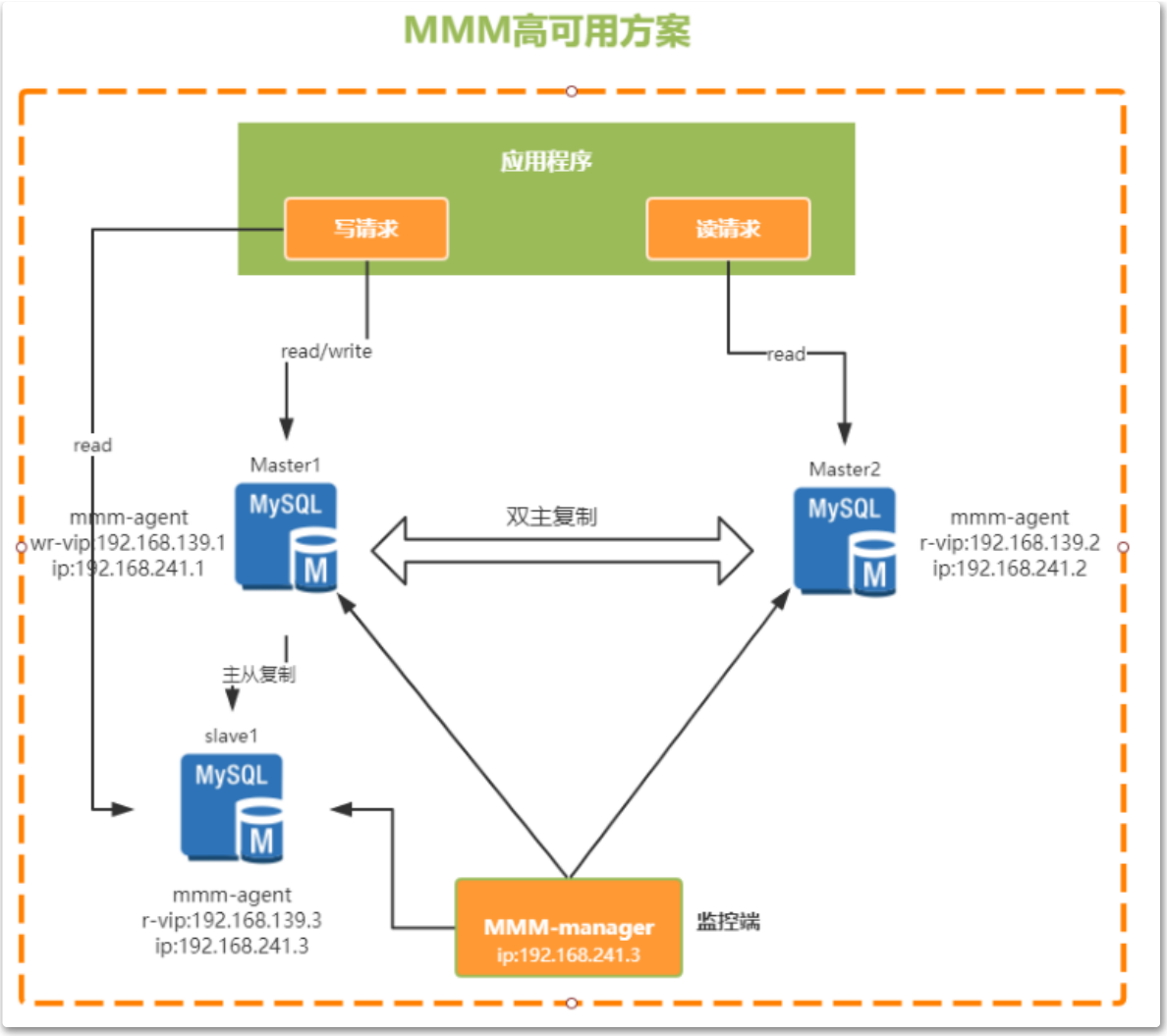
常见的MySQL集群方案有三种: MMM、MHA、MGR。这三种高可用框架都有一些共同点：

- 对主从复制集群中的Master节点进行监控
- 自动的对Master进行迁移，通过VIP。
- 重新配置集群中的其它slave对新的Master进行同步

1、MMM

MMM(Master-Master replication managerfor Mysql，Mysql主主复制管理器)是一套由Perl语言实现的脚本程序，可以对mysql集群进行监控和故障迁移。他需要两个Master，同一时间只有一个Master对外提供服务，可以说是主备模式。

他是通过一个VIP(虚拟IP)的机制来保证集群的高可用。整个集群中，在主节点上会通过一个VIP地址来提供数据读写服务，而当出现故障时，VIP就会从原来的主节点漂移到其他节点，由其他节点提供服务。



优点：

- 提供了读写VIP的配置，使读写请求都可以达到高可用
- 工具包相对比较完善，不需要额外的开发脚本
- 完成故障转移之后可以对MySQL集群进行高可用监控

缺点：

- 故障简单粗暴，容易丢失事务，建议采用半同步复制方式，减少失败的概率
- 目前MMM社区已经缺少维护，不支持基于GTID的复制

适用场景：

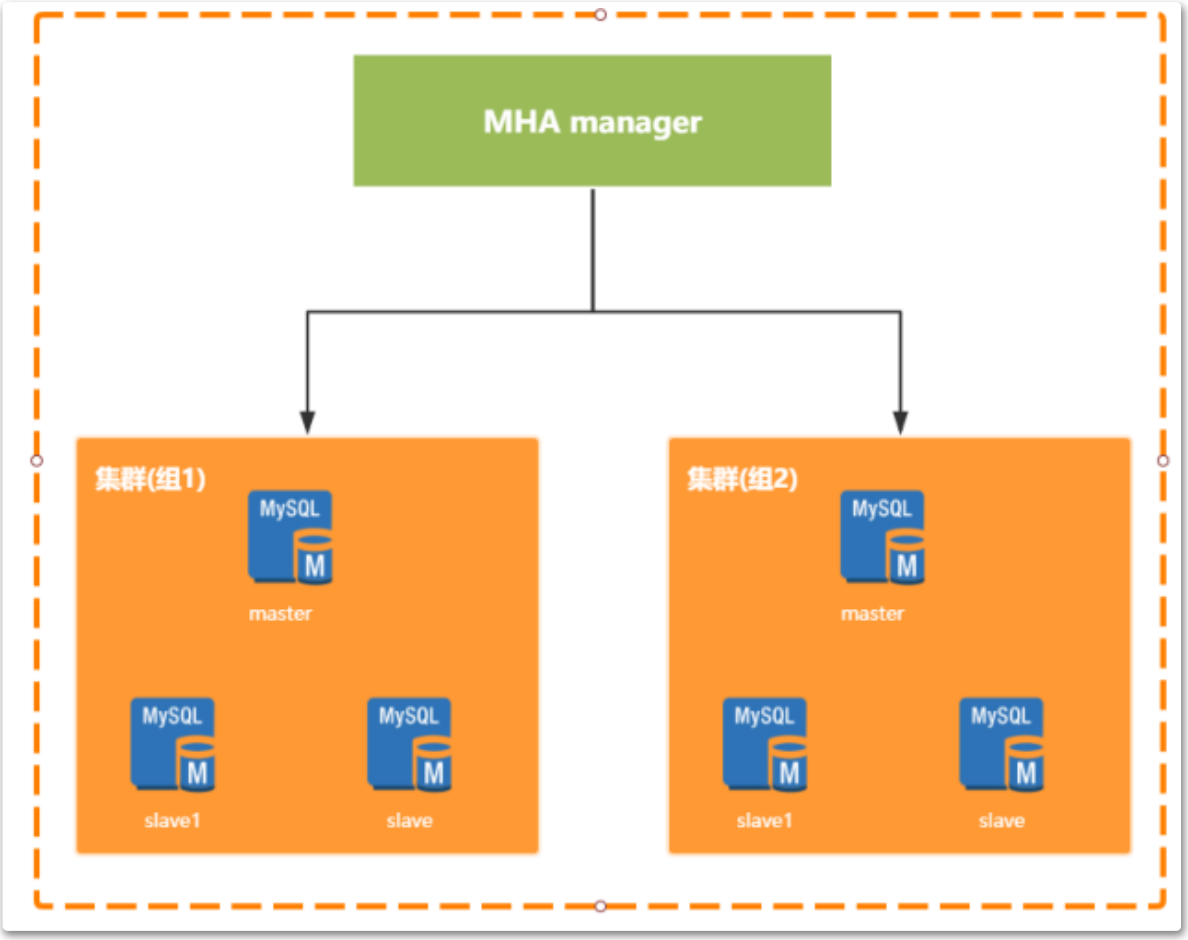
- 读写都需要高可用的
- 基于日志点的复制方式

2、MHA

Master High Availability Manager and Tools for MySQL。是由日本人开发的一个基于Perl脚本写的工具。这个工具专门用于监控主库的状态，当发现master节点故障时，会提升其中拥有新数据的slave节点成为新的master节点，在此期间，MHA会通过其他从节点获取额外的信息来避免数据一致性方面的问题。MHA还提供了mater节点的在线切换功能，即按需切换master-slave节点。MHA能够在30秒内实现故障切换，并能在故障切换过程中，最大程度的保证数据一致性。在淘宝内部，也有一个相似的TMHA产品。

MHA是需要单独部署的，分为Manager节点和Node节点，两种节点。其中Manager节点一般是单独部署的一台机器。而Node节点一般是部署在每台MySQL机器上的。 Node节点得通过解析各个MySQL的日志来进行一些操作。

Manager节点会通过探测集群里的Node节点去判断各个Node所在机器上的MySQL运行是否正常，如果发现某个Master故障了，就直接把他的一个Slave提升为Master，然后让其他Slave都挂到新的Master上去，完全透明。



优点：

- MHA除了支持日志点的复制还支持GTID的方式
- 同MMM相比，MHA会尝试从旧的Master中恢复旧的二进制日志，只是未必每次都能成功。如果希望更少的数据丢失场景，建议使用MHA架构。

缺点：

MHA需要自行开发VIP转移脚本。

MHA只监控Master的状态，未监控Slave的状态

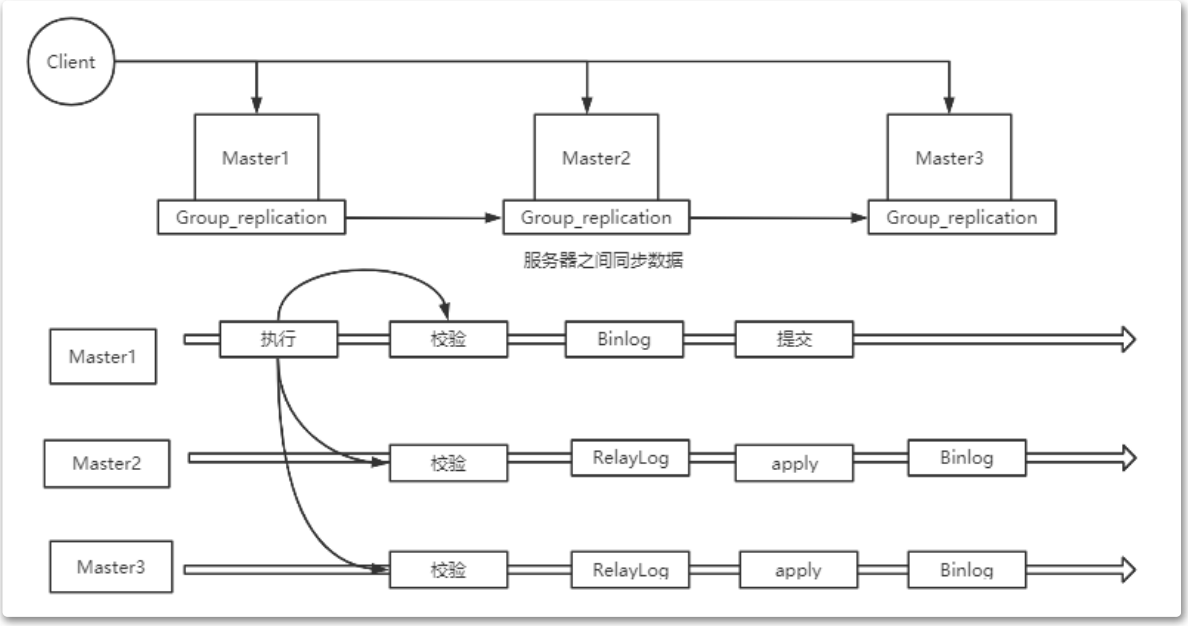
3、MGR

MGR：MySQL Group Replication。 是MySQL官方在5.7.17版本正式推出的一种组复制机制。主要是解决传统异步复制和半同步复制的数据一致性问题。

由若干个节点共同组成一个复制组，一个事务提交后，必须经过超过半数节点的决议并通过后，才可以提交。引入组复制，主要是为了解决传统异步复制和半同步复制可能产生数据不一致的问题。MGR依靠分布式一致性协议(Paxos协议的一个变体)，实现了分布式下数据的最终一致性，提供了真正的数据高可用方案(方案落地后是否可靠还有待商榷)。

支持多主模式，但官方推荐单主模式：

- 多主模式下，客户端可以随机向MySQL节点写入数据
- 单主模式下，MGR集群会选出primary节点负责写请求，primary节点与其它节点都可以进行读请求处理.



优点：

- 高一致性，基于原生复制及paxos协议的组复制技术，并以插件的方式提供，提供一致数据安全保证；

- 高容错性，只要不是大多数节点坏掉就可以继续工作，有自动检测机制，当不同节点产生资源争用冲突时，不会出现错误，按照先到者优先原则进行处理，并且内置了自动化脑裂防护机制；
- 高扩展性，节点的新增和移除都是自动的，新节点加入后，会自动从其他节点上同步状态，直到新节点和其他节点保持一致，如果某节点被移除了，其他节点自动更新组信息，自动维护新的组信息；
- 高灵活性，有单主模式和多主模式，单主模式下，会自动选主，所有更新操作都在主上进行；多主模式下，所有server都可以同时处理更新操作。

缺点:

- 仅支持InnoDB引擎，并且每张表一定要有一个主键，用于做write set的冲突检测；
- 必须打开GTID特性，二进制日志格式必须设置为ROW，用于选主与write set；主从状态信息存于表中（--master-info-repository=TABLE 、--relay-log-info-repository=TABLE），--log-slave-updates打开；
- COMMIT可能会导致失败，类似于快照事务隔离级别的失败场景
- 目前一个MGR集群最多支持9个节点
- 不支持外键于save point特性，无法做全局间的约束检测与部分事务回滚

适用的业务场景：

- 对主从延迟比较敏感
- 希望对对写服务提供高可用，又不想安装第三方软件
- 数据强一致的场景

而在目前互联网，基于云原生的思路，MySQL的服务高可用又可以有很多不同的实现方式。这里就不一一分享了。

四、分库分表方案介绍

前面我们做的一大段实验，目的是为了大家能够理解MySQL的主从集群。而主从集群的作用，在我们开发角度更大的是作为读写分离的支持，也是我们后面学习ShardingSphere的重点。我们这一部分就来介绍下分库分表。

分库分表就是业务系统将数据写请求分发到master节点，而读请求分发到slave节点的一种方案，可以大大提高整个数据库集群的性能。但是要注意，分库分表的一整套逻辑全部是由客户端自行实现的。而对于MySQL集群，数据主从同步是实现读写分离的一个必要前提条件。

1、分库分表有什么用

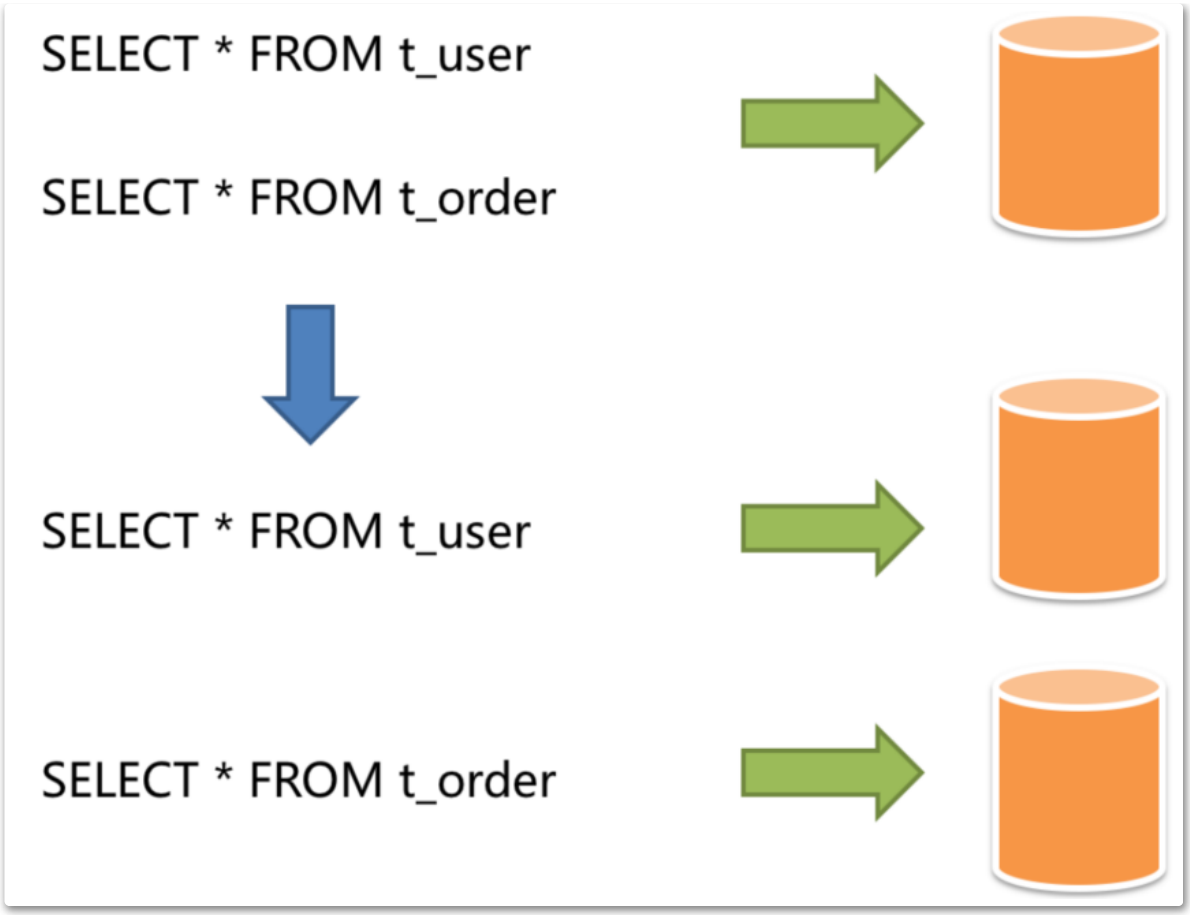
分库分表就是为了解决由于数据量过大而导致数据库性能降低的问题，将原来独立的数据库拆分成若干数据库组成
，将数据大表拆分成若干数据表组成，使得单一数据库、单一数据表的数据量变小，从而达到提升数据库性能的目
的。

例如：微服务架构中，每个服务都分配一个独立的数据库，这就是分库。而对一些业务日志表，按月拆分成不同的表，这就是分表。

2、分库分表的方式

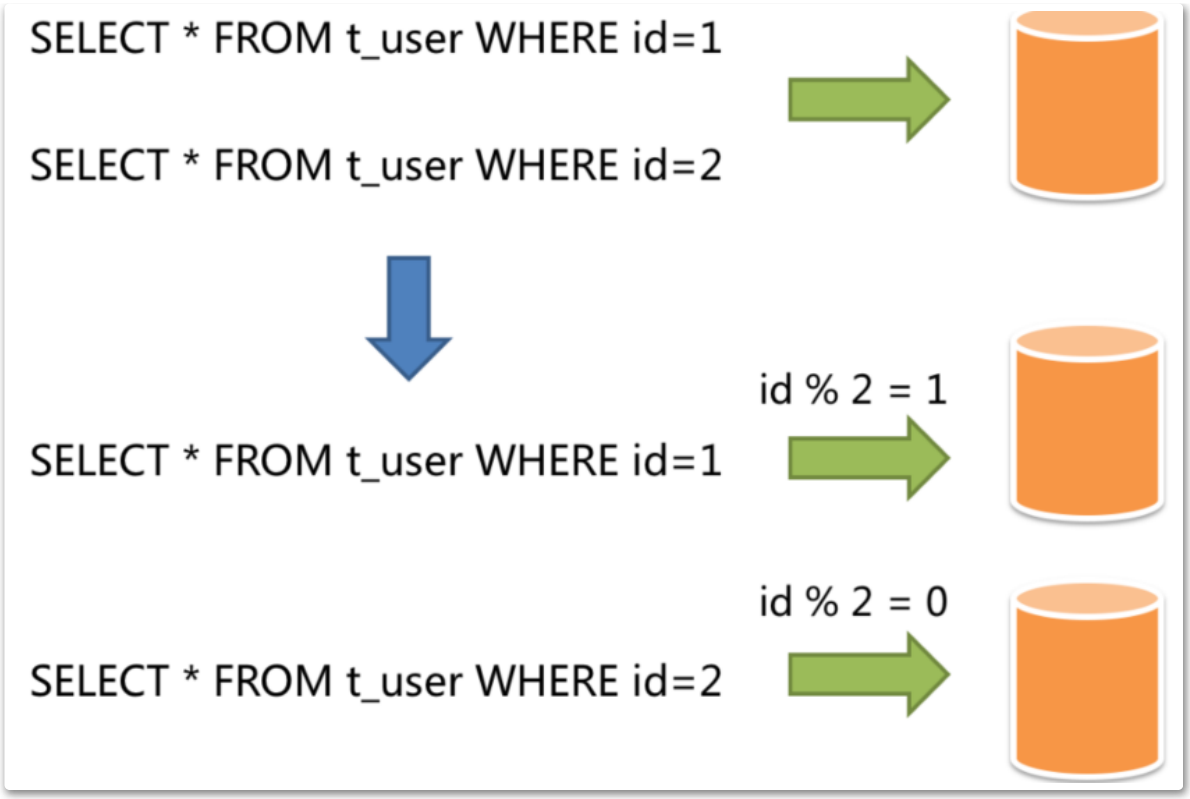
分库分表包含分库和分表 两个部分，而这两个部分可以统称为数据分片，其目的都是将数据拆分成不同的存储单元。另外，从分拆的角度上，可以分为垂直分片和水平分片。

- 垂直分片： 按照业务来对数据进行分片，又称为纵向分片。他的核心理念就是转库专用。在拆分之前，一个数据库由多个数据表组成，每个表对应不同的业务。而拆分之后，则是按照业务将表进行归类，分布到不同的数据库或表中，从而将压力分散至不同的数据库或表。例如，下图将用户表和订单表垂直分片到不同的数据库：



垂直分片往往需要对架构和设计进行调整。通常来讲，是来不及应对业务需求快速变化的。而且，他也无法真正的解决单点数据库的性能瓶颈。垂直分片可以缓解数据量和访问量带来的问题，但无法根治。如果垂直分片之后，表中的数据量依然超过单节点所能承载的阈值，则需要水平分片来进一步处理。

- 水平分片：又称横向分片。相对于垂直分片，它不再将数据根据业务逻辑分类，而是通过某个字段(或某几个字段)，根据某种规则将数据分散至多个库或表中，每个分片仅包含数据的一部分。例如，像下图根据主键机构分片。



常用的分片策略有：

取余\取模 ： 优点 均匀存放数据，缺点 扩容非常麻烦

按照范围分片： 比较好扩容， 数据分布不够均匀

按照时间分片： 比较容易将热点数据区分出来。

按照枚举值分片： 例如按地区分片

按照目标字段前缀指定进行分区：自定义业务规则分片

水平分片从理论上突破了单机数据量处理的瓶颈，并且扩展相对自由，是分库分表的标准解决方案。

一般来说，在系统设计阶段就应该根据业务耦合松紧来确定垂直分库，垂直分表方案，在数据量及访问压力不是特别大的情况，首先考虑缓存、读写分离、索引技术等方案。若数据量极大，且持续增长，再考虑水平分库水平分表方案。

> 扩展问题： 如何设计一个不需要数据迁移的取模分片扩容方案?

3、分库分表要解决哪些问题

之前说过，分库分表其实并不只是字面意义上的拆分数据，他还有一系列的问题需要解决。虽然数据分片解决了性能、可用性以及单点备份恢复等问题，但是分布式的架构在获得收益的同时，也引入了非常多新的问题。而这些，都是一个成熟的分库分表方案需要考虑的问题。

- 事务一致性问题

  原本单机数据库有很好的事务机制能够帮我们保证数据一致性。但是分库分表后，由于数据分布在不同库甚至不同服务器，不可避免会带来分布式事务问题。
- 跨节点关联查询问题

  在没有分库时，我们可以进行很容易的进行跨表的关联查询。但是在分库后，表被分散到了不同的数据库，就无法进行关联查询了。

  这时就需要将关联查询拆分成多次查询，然后将获得的结果进行拼装。
- 跨节点分页、排序函数

  跨节点多库进行查询时，limit分页、order by排序等问题，就变得比较复杂了。需要先在不同的分片节点中将数据
  进行排序并返回，然后将不同分片返回的结果集进行汇总和再次排序。

  这时非常容易出现内存崩溃的问题。
- 主键避重问题

  在分库分表环境中，由于表中数据同时存在不同数据库中，主键值平时使用的自增长将无用武之地，某个分区数据
  库生成的ID无法保证全局唯一。因此需要单独设计全局主键，以避免跨库主键重复问题。
- 公共表处理

  实际的应用场景中，参数表、数据字典表等都是数据量较小，变动少，而且属于高频联合查询的依赖表。这一类表一般就需要在每个数据库中都保存一份，并且所有对公共表的操作都要分发到所有的分库去执行。
- 运维工作量

  面对散乱的分库分表之后的数据，应用开发工程师和数据库管理员对数据库的操作都变得非常繁重。对于每一次数据读写操作，他们都需要知道要往哪个具体的数据库的分表去操作，这也是其中重要的挑战之一。

4、什么时候需要分库分表?

在阿里巴巴公布的开发手册中，建议MySQL单表记录如果达到500W这个级别，或者单表容量达到2GB，一般就建议进行分库分表。而考虑到分库分表需要对数据进行再平衡，所以如果要使用分库分表，就要在系统设计之初就详细考虑好分库分表的方案，这里要分两种情况。

一般对于用户数据这一类后期增长比较缓慢的数据，一般可以按照三年左右的业务量来预估使用人数，按照标准预设好分库分表的方案。

而对于业务数据这一类增长快速且稳定的数据，一般则需要按照预估量的两倍左右预设分库分表方案。并且由于分库分表的后期扩容是非常麻烦的，所以在进行分库分表时，尽量根据情况，多分一些表。最好是计算一下数据增量，永远不用增加更多的表。

另外，在设计分库分表方案时，要尽量兼顾业务场景和数据分布。在支持业务场景的前提下，尽量保证数据能够分得更均匀。

最后，一旦用到了分库分表，就会表现为对数据查询业务的灵活性有一定的影响，例如如果按userId进行分片，那按age来进行查询，就必然会增加很多麻烦。如果再要进行排序、分页、聚合等操作，很容易就扛不住了。这时候，都要尽量在分库分表的同时，再补充设计一个降级方案，例如将数据转存一份到ES，ES可以实现更灵活的大数据聚合查询。

5、常见的分库分表组件

由于分库分表之后，数据被分散在不同的数据库、服务器。因此，对数据的操作也就无法通过常规方式完成，并且
它还带来了一系列的问题。好在，这些问题不是所有都需要我们在应用层面上解决，市面上有很多中间件可供我们
选择，我们来了解一下它。

- shardingsphere 官网地址：https://shardingsphere.apache.org/document/current/cn/overview/

Sharding-JDBC是当当网研发的开源分布式数据库中间件，他是一套开源的分布式数据库中间件解决方案组成的生态圈，它由Sharding-JDBC、Sharding-Proxy和Sharding-Sidecar（计划中）这3款相互独立的产品组成。 他们均提供标准化的数据分片、分布式事务和
数据库治理功能，可适用于如Java同构、异构语言、容器、云原生等各种多样化的应用场景。

这也是我们这次学习的重点。

- mycat 官网地址： http://www.mycat.org.cn/

基于阿里开源的Cobar产品而研发，Cobar的稳定性、可靠性、优秀的架构和性能以及众多成熟的使用案例使得MYCAT一开始就拥有一个很好的起点，站在巨人的肩膀上，我们能看到更远。业界优秀的开源项目和创新思路被广泛融入到MYCAT的基因中，使得MYCAT在很多方面都领先于目前其他一些同类的开源项目，甚至超越某些商业产品。

MyCAT虽然是从阿里的技术体系中出来的，但是跟阿里其实没什么关系。

- DBLE 官网地址：https://opensource.actionsky.com/

该网站包含几个重要产品。其中分布式中间件可以认为是MyCAT的一个增强版，专注于MySQL的集群化管理。另外还有数据传输组件和分布式事务框架组件可供选择。

五、课程内容总结

今天课程的内容，主要是给大家介绍MySQL的分布式集群化的主要思路和解决方案。但是，如果换成其他的数据库产品，比如Oracle、SqlServer、DB2等其他数据库，那么肯定又需要有不同的解决方案。所以，对于这次课程的内容，重点在于领会思想。

其实，分库分表和主从集群，这些方案背后的核心思想都是水平扩展，从单机往集群化发展。这种思想其实我们并不陌生，微服务体系也就是干的这个事情。只不过，由于数据库是一个重状态的组件，因此，在将服务进行集群化的过程中，也需要考虑状态如何分布。而这个事情，MySQL在做，其他的数据组件也都在做，比如Redis、MongoDB等等。对于今天课程的内容，你不妨同之前学过的Redis、MongoDB等其他组件进行数据同步的思路进行下横向的总结、对比，这对于你以后去理解其他没有接触过的新产品比如HBase、Clickhouse等会有帮助的。

并且，分布式已经是现在软件的主流，我们在以后肯定会要面临各种各样的分布式问题，数据库集群、服务集群、缓存集群、图片等静态资源集群、计算集群 等等。当面临各种各样的复杂场景，如何构建一个稳定高效的集群化方案，这是一个需要日积月累，不断沉淀的问题。而这次的MySQL集群化以及后面的ShardingSphere分库分表，就是一个很好的参考。

> 有道云笔记分享链接：
>
> 文档：VIP01-MySQL主从架构及读写分离实战.md
> 链接：http://note.youdao.com/noteshare?id=64e8c9ae0f01a59ca9b7f967e771a73b&sub=34BC895B60B34639A440191A44D795C9