MySQL05

MySQL Replication概述

集群的主要类型：
高可用集群（High Available Cluster，ＨＡ）
高可用集群是指通过特殊的软件把独立的服务器连接起来，组成一个能够提供故障切换（Ｆａｉｌ　Ｏｖｅｒ）功能的集群

如何衡量高可用：

99.53%	7天	常规系统
99.98%	8小时	可用系统
99.99%	52.6分钟	高可用系统
99.999%	5.3分钟	抗故障系统
99.9999%	32秒	容错系统

常用的集群架构：
MySQL Replication
MySQL Cluster
MySQL Group Replication （MGR） 5.7.17
MariaDB Galera Cluster
Keepalived|HeartBeat||Lvs，Haproxy等技术构建高可用集群

什么是MySQL Replication？
1、Replication可以实现将数据从一台数据库服务器（master）复制到一台到多台数据库服务器上（slave）
2、默认情况下，属于异步复制，所以无需维持长连接

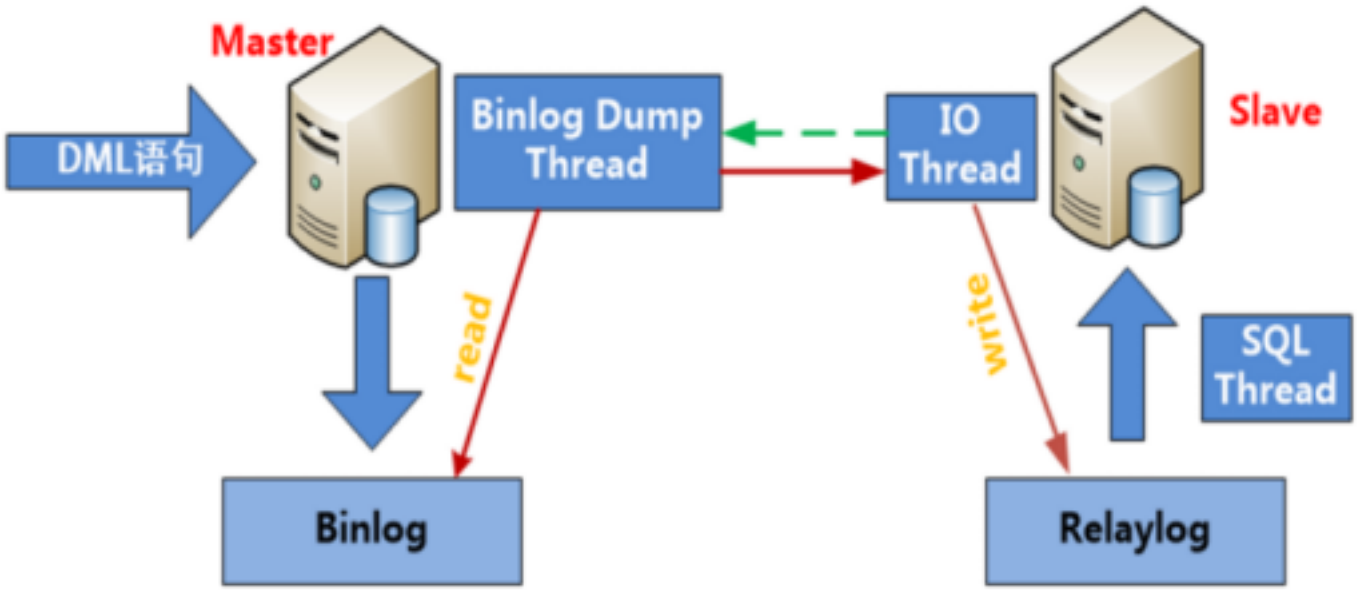
MySQL Replication的原理：
简单来说，master将数据库的改变写入二进制日志，slave同步这些二进制日志，并根据这些二进制日志进行数据重演操作，实现数据异步同步。

MySQL Replication的用途：

1、Fail Over 故障切换
2、Backup 在线热备份（机械故障）
3、High Performance 高性能

MySQL Replication的架构：
master ---> slave (双机热备)
默认情况下，master接受读写请求，slave只接受读请求以减轻master的压力
复制的过程：

1、slave端的IO线程连上master端，请求
2、master端返回给slave端，bin log文件名和位置信息
3、IO线程把master端的bin log内容依次写到slave端relay bin log里，并把master端的bin-log文件名和位置记录到master.info里。
4、salve端的sql线程，检测到relay bin log中内容更新，就会解析relay log里更新的内容，并执行这些操作



master ---> slave1 -----> slave2 (级联架构)
优点： 进一步分担读压力
缺点： slave1 出现故障，后面的所有级联slave服务器都会同步失败

/----> slave1
master (并联架构)
\----> slave2

优点：解决上面的slave1的单点故障，同时也分担读压力
缺点：间接增加master的压力（传输二进制日志压力）

master1 <------> master2 (互为主从)

优点：

从命名来看，两台master好像都能接受读、写请求，但实际上，往往运作的过程中，同一时刻只有其中一台master会接受写请求，另外一台接受读请求

M—S架构：实现双机热备（AB复制）

1、可以降低master读压力

2、可以对数据库做“热备”，热备只能解决硬件master硬件故障，软件故障等重大故障问题，但无法解决人为误操作导致的逻辑故障（例如输入错误的SQL语句把重要的记录删除了），所以常规的备份是必须。

环境准备及要求：
1、关闭防火墙和selinux
2、hosts文件中两台服务器主机名和ip地址一一对应起来
3、系统时间需要同步
4、master和slave的数据库版本保持一致
5、master:10.1.1.1 slave:10.1.1.2

具体步骤：
1、修改配置文件（master和slave）
master：
log-bin=/var/lib/mysql/mysqld-bin　　master必须开启二进制日志
server-id=1　　mysql数据库的编号，master和slave必须不一样

slave：
log-bin=/mysql56/mysqld-bin　　slave上的二进制日志可以开启也可以不开启，看具体情况
server-id=2　　mysql数据库的编号
relay-log=/mysql56/relay-log　　主从复制日志需要开启

2、初始化数据，使两边数据一致。（以master为主）

3、master端创建授权用户
mysql> grant replication slave on *.* to 'slave'@'10.1.1.%' identified by '123';
mysql> flush privileges;

4、查看master的正在写的二进制文件名和位置
mysql> flush tables with read lock;　　先加锁，防止两边数据不一致;如果业务还未上线，这个就没有必要了
Query OK, 0 rows affected (0.00 sec)
mysql> show master status;　只有打开二进制日志，这句命令才有结果，表示当前数据库的二进制日志写到什么位置

```
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000001 |      331 |              |                  |
+------------------+----------+--------------+------------------+
```

二进制文件名　　正在写入的位置

5、slave端设定复制信息

```
mysql> change master to
    -> master_host='10.1.1.20',          master ip
    -> master_user='slave',              同步用户
    -> master_password='123',      密码
    -> master_port=3306,                   端口
    -> master_log_file='mysqld-bin.000001',  主上面查到到二进制日志名
    -> master_log_pos=331;               主上面查到的位置号
```

6、启动复制线程，开始同步
mysql> start slave;
mysql> show slave status \G;

Slave_IO_Running: Yes 代表成功连接到master并且下载日志
Slave_SQL_Running: Yes 代表成功执行日志中的SQL语句

回到master端解锁：
mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)
7、测试验证
master写——>slave可以看到
slave写——>master看不到

在上述架构下实现故障迁移和恢复
故障迁移：

1、模拟master出现故障
2、查看slave同步状态并停止向master同步数据
slave > show slave status \G;
Slave_IO_Running: Connecting
Slave_SQL_Running: Yes
Last_IO_Error: error reconnecting to master 'slave@10.1.1.2:3307' - retry-time: 60 retries: 1

mysql> stop slave; <---停止

3、master故障之后，前端的应用应该把读写请求都调度给slave

```
                r/w
  X             |
master          slave
```

直接用客户端登录slave，对数据进行修改，模拟写操作
mysql> use db2;
mysql> insert into t1 set id=2;
mysql> insert into t1 set id=3;
mysql> use db1;
mysql> update t1 set name='test' where id=3;

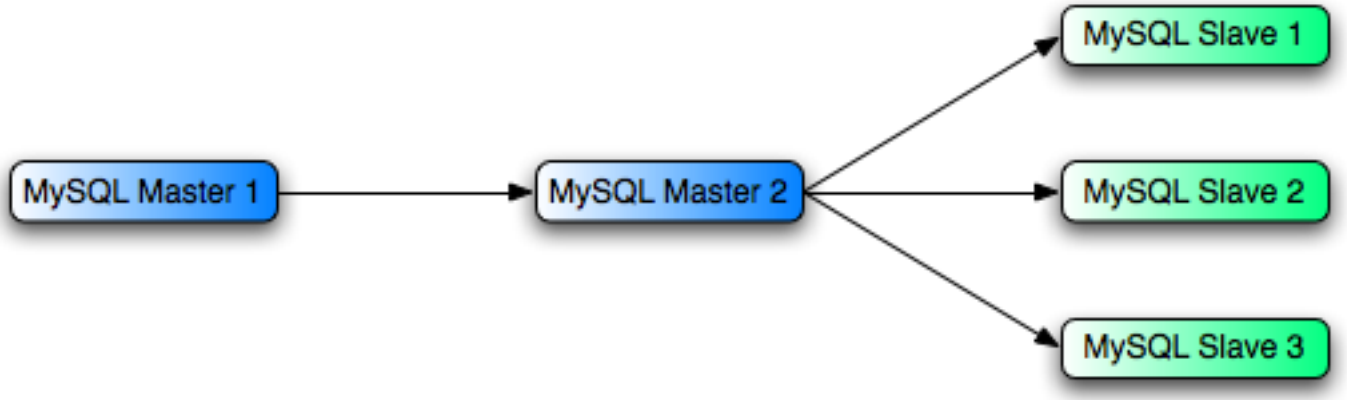
故障修复
可以肯定的是，在这个架构下，master要上线，肯定只有一个选择，就是作为原有slave的从，重新上线：
主 从
slave ---> master
情况：
假设数据已经损害了、丢失了，那么最简单的方法就是重装master数据库，把master作为slave的从，原来的slave就变成新架构的主

注意：
确保新的架构复制成功之后，回到slave服务器，把数据目录下的master.info文件删除，否则的化，下次如果slave重启数据库服务，会自动连接master；slave IO线程把master端的bin log内容依次写到slave端relay bin log里，并把master端的bin-log文件名和位置记录到master.info里

M-M 互为主从（双向复制）

master1和master2相互都作为对方的主和从
但是，同一时刻，只有其中一个节点接受写操作

```
        r/w                ro
        |                  |
      master1  <----> master2
```



注意：在架构工作正常的情况下，master1没有出现故障的情况下，尽可能不要给master1写请求的同时也给master2写请求。

故障情况：
状况1：
　　如果master2出现故障，而且仅仅是简单的故障，没有出现数据丢失，那么只需要重新启动master2，master2会把落后的数据自动同步。

状况2：
如果master2出现严重故障，数据已经丢失了，建议重新使用master1过去某个备份去搭建master2

修复步骤：
　　1）停止master1的复制线程： stop slave
　　2）重新搭建master2
　　3）重新设定master1向新的master2复制的设定，然后start slave

M-S1-S2 级联复制
master——>slave1——>slave2

打开log-slave-updates=1，让第一台传过来relay日志记录到自己的二进制日志
思路：
先搭建好主从——>然后在加入slave2

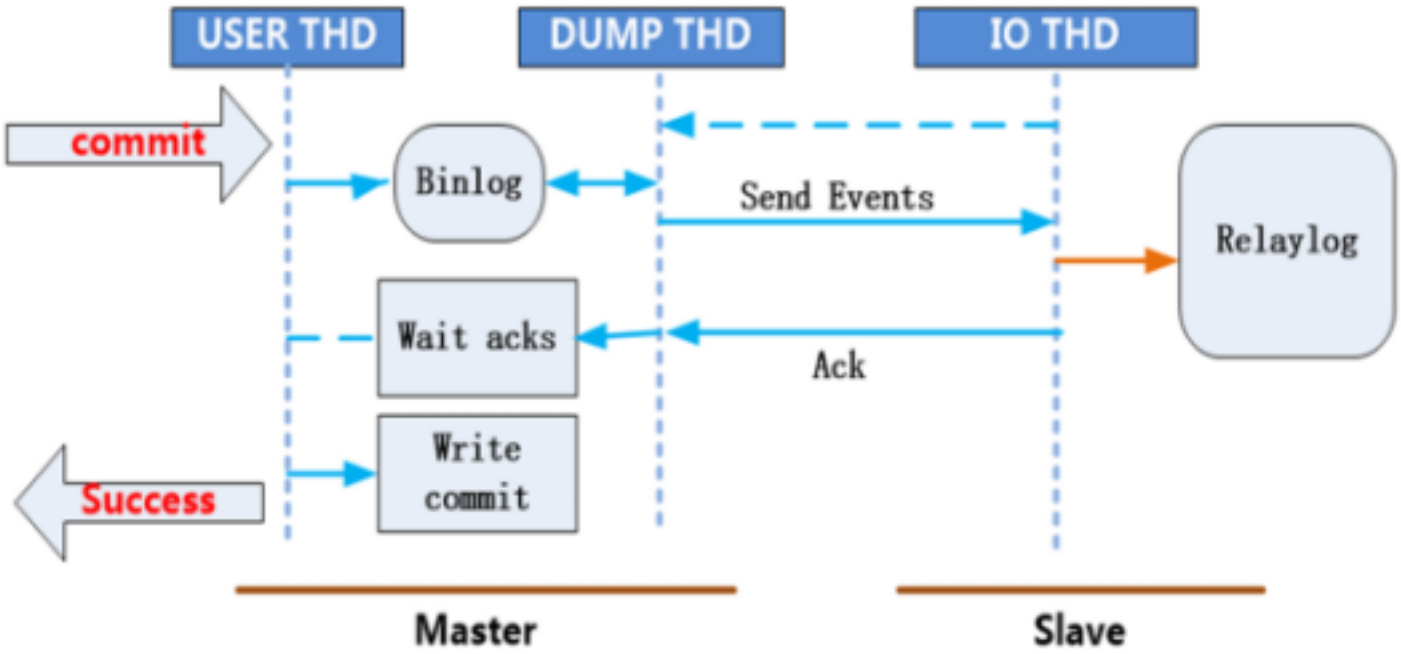
总结：

上面的复制架构默认都是异步的，也就是主库将binlog日志发送给从库，这一动作就结束了，并不会验证从库是否接受完毕。这样可以提供最佳的性能。但是同时也带来了很高的风险，当主服务器或者从服务器发生故障时，极有可能从服务器没有接到主服务器发过来的binglog日志，这样就会导致主从数据不一致，甚至导致数据丢失。为了解决该问题，mysql5.5引入了半同步复制模式。

mysql半同步复制

所谓的半同步复制就是master每commit一个事务(简单来说就是做一个改变数据的操作），要确保slave接受完主服务器发送的binlog日志文件并写入到自己的中继日志relay log里，然后会给master信号，告诉对方已经接收完毕，这样master才能把事物成功commit。这样就保证了master-slave的数据绝对的一致（但是以牺牲master的性能为代价).但等待时间也是可以调整的。

原理：



mysql半同步复制等待时间超时后(默认时间为10秒)，会自动转换成异步复制
架构：
master——>slave

步骤：
第一大步:
先要搭建好mysqlAB异步复制

第二大步:在异步基础上转成半同步复制
需要安装插件：
ls /usr/local/mysql/lib/plugin/semisync_*
/usr/local/mysql/lib/plugin/semisync_master.so　master上
/usr/local/mysql/lib/plugin/semisync_slave.so　　slave上

1、在master上安装这个插件
master> install plugin rpl_semi_sync_master soname 'semisync_master.so';
Query OK, 0 rows affected (0.00 sec)

删除插件的方法：
mysql > uninstall plugin rpl_semi_sync_master;
master> show global variables like 'rpl_semi_sync%'; --安装OK后，主上会多几个参数
+-----------------------------------+-------+
| Variable_name　　| Value |
+-----------------------------------+-------+
| rpl_semi_sync_master_enabled　　| OFF　| --是否启用master的半同步复制
| rpl_semi_sync_master_timeout　　| 10000 | --默认主等待从返回信息的超时间时间，10秒。动态可调
| rpl_semi_sync_master_trace_level　| 32　|　用于开启半同步复制模式时的调试级别，默认是32
| rpl_semi_sync_master_wait_no_slave | ON　|　　--是否允许每个事物的提交都要等待slave的信号.on为每一个事物都等待，off则表示slave追赶上后，也不会开启半同步模式，需要手动开启
+-----------------------------------+-------+
官方手册里都有解释：5.1.1 Server Option and Variable Reference
rpl_semi_sync_master_trace_level
Default　32
The semisynchronous replication debug trace level on the master. Four levels are defined:
1 = general level (for example, time function failures)　一般等级
16 = detail level (more verbose information)　　更详细的信息
32 = net wait level (more information about network waits)　　网络等待等级
64 = function level (information about function entry and exit)　函数等级（有关函数输入和退出的信息）

2、在slave上安装插件
slave> install plugin rpl_semi_sync_slave soname 'semisync_slave.so';
Query OK, 0 rows affected (0.03 sec)

slave> show global variables like 'rpl_semi_sync%';
+-------------------------------+-------+
| Variable_name | Value |
+-------------------------------+-------+
| rpl_semi_sync_slave_enabled | OFF | slave是否启用半同步复制
| rpl_semi_sync_slave_trace_level | 32 |
+-------------------------------+-------+

3，master上激活半同步复制
master> set global rpl_semi_sync_master_enabled =on;
Query OK, 0 rows affected (0.00 sec)

4，slave上激活半同步复制
slave> set global rpl_semi_sync_slave_enabled=on;
slave> stop slave IO_THREAD;
slave> start slave IO_THREAD;

5，在master查看状态
master > show global status like 'rpl_semi_sync%';
+--+-------+
| Variable_name | Value |
+--+-------+
| Rpl_semi_sync_master_clients | 1 | --有一个从服务器启用半同步复制
| Rpl_semi_sync_master_net_avg_wait_time | 0 | --master等待slave回复的平均等待时间。单位毫秒
| Rpl_semi_sync_master_net_wait_time | 0 | --master总的等待时间。单位毫秒
| Rpl_semi_sync_master_net_waits | 0 | --master等待slave回复的总的等待次数
| Rpl_semi_sync_master_no_times | 0 | --master关闭半同步复制的次数
| Rpl_semi_sync_master_no_tx | 0 | --表示从服务器确认的不成功提交的数量
| Rpl_semi_sync_master_status | ON | --标记master现在是否是半同步复制状态

| Rpl_semi_sync_master_timefunc_failures | 0 | --master调用时间（如gettimeofday())失败的次数
| Rpl_semi_sync_master_tx_avg_wait_time | 0 | --master花在每个事务上的平均等待时间
| Rpl_semi_sync_master_tx_wait_time | 0 | --master花在事物上总的等待时间
| Rpl_semi_sync_master_tx_waits | 0 | --master事物等待次数
| Rpl_semi_sync_master_wait_pos_backtraverse | 0 | --后来的先到了，而先来的还没有到的次数
| Rpl_semi_sync_master_wait_sessions | 0 | --当前有多少个session因为slave回复而造成等待
| Rpl_semi_sync_master_yes_tx | 0 | --表示从服务器确认的成功提交数量
+--+-------+

6，在slave上查看状态就只有下面一条信息
slave > show global status like 'rpl_semi_sync%';
+---------------------------+-------+
| Variable_name | Value |
+---------------------------+-------+
| Rpl_semi_sync_slave_status | ON |
+---------------------------+-------+

第三大步：测试
工作原理：当slave从库的IO_Thread 线程将binlog日志接受完毕后，要给master一个确认，如果超过10s未收到slave的接收确认信号，那么就会自动转换为传统的异步复制模式。
正常情况下，master插入一条记录，查看slave是否有成功返回
master > insert into a values (3);
Query OK, 1 row affected (0.01 sec)

master > show global status like 'rpl_semi_sync%_yes_tx';
+----------------------------+-------+
| Variable_name | Value |
+----------------------------+-------+
| Rpl_semi_sync_master_yes_tx | 1 | --表示这次事物成功从slave返回一次确认信号
+----------------------------+-------+

模拟故障：当slave挂掉后，master这边更改操作
service stop mysql

或者直接停止slave的IO_thread线程
stop slave io_thread;
master> insert into a values (4);
Query OK, 1 row affected (10.00 sec)　　--这次插入一个值需要等待10秒（默认的等待时间)

master> insert into a values (5);
Query OK, 1 row affected (0.01 sec)　　--现在自动转成了原来的异步模式（类似oracle DG里的最大性能模式)

再次把slave启动，看到半同步复制没启来，是异步模式
重新按下面的步骤把同步模式再启起来就可以了
slave> set global rpl_semi_sync_slave_enabled=on;
slave> stop slave IO_THREAD;
slave> start slave IO_THREAD;

或者可以将该参数写入到配置文件中：
master：rpl_semi_sync_master_enabled=1
slave：rpl_semi_sync_slave_enabled=1

结果：master需要等到slave确认后才能提交，如果等不到确认消息，master等待10s种后自动变成异步同步;slave启起来后，master上改变的数据还是会自动复制过来，数据又回到一致

等待时间可以动态调整：
mysql> set global rpl_semi_sync_master_timeout=3600000;
mysql> show global variables like 'rpl_semi_sync%';

Variable_name	Value
rpl_semi_sync_master_enabled	ON
rpl_semi_sync_master_timeout	3600000
rpl_semi_sync_master_trace_level	32
rpl_semi_sync_master_wait_no_slave	ON

mysql开源管理工具 ——>maatkit --perl写的

maatkit-7540.tar.gz

--在mysql AB的slave上安装（只需要在slave上安装，包含下面的步骤都是在slave上做的)
tar xf maatkit-7540.tar.gz -C /usr/src/
cd /usr/src/maatkit-7540/

安装方法README文件里有写
perl Makefile.PL --如果不成功，需要安装perl有关的多个包，可以yum install perl* ；如果安装报错，检查是否安装mysql-share包
make install

[root@vm2 maatkit-7540]# ls bin/ --这些命令，就是各个管理工具
mk-archiver mk-purge-logs
mk-checksum-filter mk-query-advisor
mk-config-diff mk-query-digest
mk-deadlock-logger mk-query-profiler
mk-duplicate-key-checker mk-show-grants
mk-error-log mk-slave-delay
mk-fifo-split mk-slave-find
mk-find mk-slave-move
mk-heartbeat mk-slave-prefetch
mk-index-usage mk-slave-restart
mk-kill mk-table-checksum
mk-loadavg mk-table-sync
mk-log-player mk-table-usage
mk-merge-mqd-results mk-tcp-model
mk-parallel-dump mk-upgrade
mk-parallel-restore mk-variable-advisor
mk-profile-compact mk-visual-explain

--使用--help查看一个命令的使用方法
mk-slave-delay --help 启动和停止slave服务器使其滞后master
mk-slave-delay starts and stops a slave server as needed to make it lag behind the master. The SLAVE-HOST and MASTER-HOST use DSN syntax, and values are copied from the SLAVE-HOST to the MASTER-HOST if omitted. For more details, please use the --help option, or try 'perldoc /usr/bin/mk-slave-delay' for complete documentation.

man mk-slave-delay　man文档

mk-slave-delay --delay 1m --interval 15s --run-time 10m slavehost
--delay：延迟1m复制
--interval：间隔15s去检查slave状态
--quiet：运行时不显示输出信息
--run-time：运行时间，默认一直运行

mysql AB(无论同步或异步)正在运行OK的情况下，使用下面的命令在slave上运行;做之间建议把时间同步一下

mk-slave-delay --defaults-file=/usr/local/mysql/etc/my.cnf --delay=1m --interval=15s --user=root --password=123 --quiet localhost &
表示延时1分钟，才会应用SQL线程；这里是测试所以才使用很小的时间，实际情况可以调成1小时或2小时；间隔15s去检测slave是否需要被关闭或者启动

测试：

在master上随便插入几条数据

然后在slave上发现没有马上同步过来
slave > show slave status\G;　--查看状态会发现SQL线程状态为NO

Slave_IO_Running: Yes
Slave_SQL_Running: NO

大概等1分钟，就会自动延时同步过来了；
--注意:日志已经传到slave的relay-bin log里了，但由SQL线程延时去解析

问题1：
如果现在一个小公司mysql数据库已经跑了一年，现在要搭建mysqlAB复制，你检查了主库后，发现它这一年都没有使用二进制日志，请问如何做复制？

1、打开主库二进制日志，然后重启

2、全备主库，然后恢复到备库
3、搭建AB复制，指定从备份完成的那个日志position开始复制

问题2：
如果一个lamp架构在深圳机房在运行，如何尽量无影响的把这个lamp迁移到广州的机房

1、在广州那边先搭建lamp，其中web内容可以和深圳的这边做rsync远程实时同步，mysql做双主
2、改DNS的A记录，把IP由深圳的换成广州的
3、过一段时间（等深圳服务器没有连接了），就可以关闭深圳服务器了。
    stop slave;

基于GTIDs的Replication

基于GTIDs的MySQL Replication
什么是GTIDs以及有什么特点？
1、GTIDs（Global transaction identifiers）全局事务标识符，是mysql 5.6新加入的一项技术
2、当使用GTIDs时，每一个事务都可以被识别并且跟踪
3、添加新的slave或者当发生故障需要将master身份或者角色迁移到slave上时，都无需考虑是哪一个二进制日志以及哪个position值，极大简化了相关操作
4、GTIDs是完全基于事务的，因此不支持MYISAM存储引擎
5、GTID由source_id和transaction_id组成：
1>source_id来自于server_uuid,可以在auto.cnf中看到
2>transation_id是一个序列数字，自动生成

使用GTIDs的限制条件有哪些？
1、不支持非事务引擎（MYisam），因为可能会导致多个gtid分配给同一个事务
2、create table ... select 语句不支持（主库语法报错）
3、create/drop temporary table 语句不支持
4、必须使用enforce-gtid-consistency参数
5、sql-slave-skip-counter不支持(传统的跳过错误方式)
6、GTID复制环境中必须要求统一开启和GTID或者关闭GTID
7、在mysql 5.6.7之前，使用mysql_upgrade命令会出现问题

GTID的生命周期包含以下部分：
1. A transaction is executed and committed on the master.

This transaction is assigned a GTID using the master's UUID and the smallest nonzero transaction sequence number not yet used on this server; the GTID is written to the master's binary log (immediately preceding the transaction itself in the log).

2. After the binary log data is transmitted to the slave and stored in the slave's relay log, the slave reads the GTID and sets the value of its gtid_next system variable as this GTID. This tells the slave that the next transaction must be logged using this GTID.It is important to note that the slave sets gtid_next in a session context.

3. The slave verifies that this GTID has not already been used to log a transaction in its own binary log. If this GTID has not been used, the slave then writes the GTID, applies the transaction, and writes the transaction to its binary log. By reading and checking the transaction's GTID first, before processing the transaction itself, the slave guarantees not only that no previous transaction having this GTID has been applied on the slave, but also that no other session has already read this GTID but has not yet committed the associated transaction. In other words, multiple clients are not permitted to apply the same transaction concurrently.

4. Because gtid_next is not empty, the slave does not attempt to generate a GTID for this transaction but instead writes the GTID stored in this variable—that is, the GTID obtained from the master—immediately preceding the transaction in its binary log.

总结：有了GTID大大的简化了复制的过程，降低了维护的难度

配置基于GTIDs的Replication

在生产环境中，大多数情况下使用的MySQL5.6基本上都是从5.5或者更低的版本升级而来，这就意味着之前的mysql replication方案是基于传统的方式部署，并且已经在运行，因此，接下来我们就利用已有的环境升级至基于GITDs的Replication

步骤：

1、将master和slave服务器都设置为read-only

mysql>set @@global.read_only=ON;

2、停止两台服务器的mysql服务

3、开启GTIDs

注意:

1、开启GITDs需要在master和slave上都配置gtid-mode，log-bin，log-slave-updates，enforce-gtid-consistency（该参数在5.6.9之前是--disable-gtid-unsafe-

statement）
2、其次，slave还需要增加skip-slave-start参数,目的是启动的时候，先不要把slave起来，需要做一些配置
master:
[mysqld]
gtid-mode=on
log-bin
log-slave-updates
enforce-gtid-consistency

slave:
[mysqld]
gtid-mode=on
log-bin
log-slave-updates
enforce-gtid-consistency
skip-slave-start

4、重新配置slave
mysql>change master to
master_host='10.1.1.1',
master_port=3306,
master_user='slave',
master_password=123,
master_auto_position=1;

mysql> start slave;

5、关闭read-only模式
mysql> set @@global.read_only=OFF;

keepalived+mysql

keepalived是什么

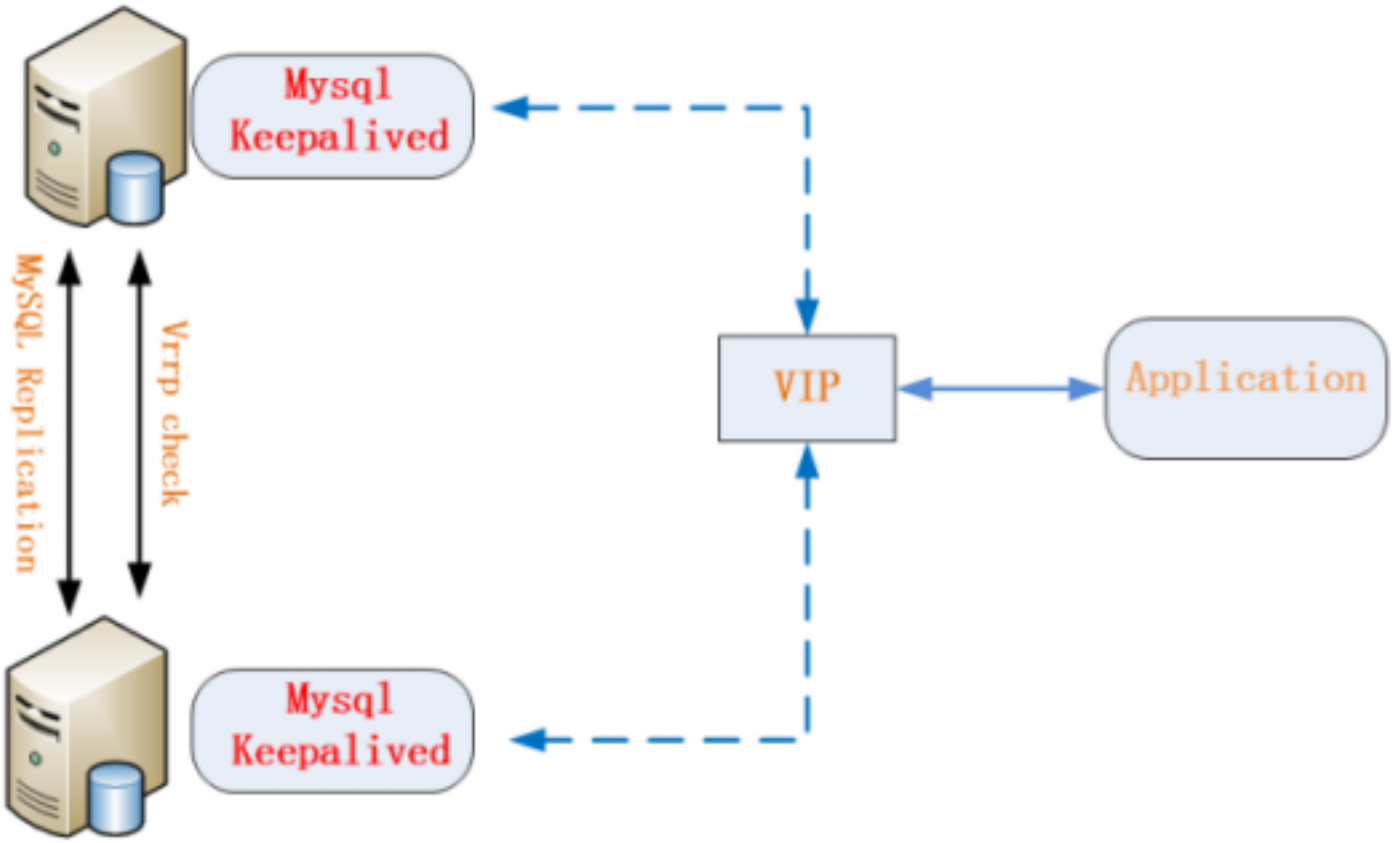
keepalived是集群管理中保证集群高可用的一个服务软件，其功能类似于heartbeat，用来防止单点故障。(HA)

keepalived工作原理

keepalived是以VRRP协议为实现基础的，VRRP全称Virtual Router Redundancy Protocol，即虚拟路由冗余协议。

虚拟路由冗余协议，可以认为是实现路由器高可用的协议，即将N台提供相同功能的路由器组成一个路由器组，这个组里面有一个master和多个backup，master上面有一个对外提供服务的vip（该路由器所在局域网内其他机器的默认路由为该vip），master会发组播，当backup收不到vrrp包时就认为master宕掉了，这时就需要根据VRRP的优先级来选举一个backup当master。这样的话就可以保证路由器的高可用了。

keepalived主要有三个模块，分别是core、check和vrrp。core模块为keepalived的核心，负责主进程的启动、维护以及全局配置文件的加载和解析。check负责健康检查，包括常见的各种检查方式。vrrp模块是来实现VRRP协议的。



在mysql互主的基础上配置keepalived(两台mysql都要安装)

```
# tar -xvf keepalived-1.2.24.tar.gz -C /usr/local/src/
# cd /usr/local/src/keepalived-1.2.24/
#./configure --prefix=/ --mandir=/usr/local/share/man/
# make
# make install
# cd /etc/keepalived/

! Configuration File for keepalived

global_defs {                                #全局定义主要设置 keepalived 的通知机制和
标识
  notification_email {
       root@localhost       #邮件地址可以多个，每行一个
  }
  notification_email_from      keepalived@localhost    #邮件的发件人
  smtp_server 127.0.0.1        #邮件服务的地址（本地）smtp协议
  smtp_connect_timeout 30  #连接smtp超时时间
  router_id test                  #标识号，路由名字
}

vrrp_instance VI_1 {
    state BACKUP                  当前主机所扮演的模式master|backup
    interface eth0                实例绑定的网卡
    virtual_router_id 51     VRID    虚拟路由标识 00-00-5e-00-01-{VRID}
    priority 100                优先级高为master，master 至少要高于 backup
    nopreempt              设置不抢占，重新启动不夺回master角色（高的优先级加）
    advert_int 1              检查间隔，默认为1秒
    authentication {          认证设置
        auth_type PASS          认证类型|验证:主备之间做身份验证  主备之间一定一致
        auth_pass 1111
    }
    virtual_ipaddress {     虚拟ip地址
        192.168.100.200
    }
}
virtual_server 192.168.100.200 3306 {    虚拟服务器所监听的端口（mysql端口）
    delay_loop 1            服务轮询的时间间隔
```

```
    lb_algo wrr   负载均衡的算法
    lb_kind DR      LVS集群模式
    persistence_timeout 50      会话保持时间（秒为单位），即以用户在50秒内被分配到同一个后端realserver
    protocol TCP            健康检查的协议 TCP还是UDP
    real_server 192.168.1.100 3306 {        实际mysql的真实ip
        weight 1  权重：0表示失效(不转发请求直到恢复正常)，默认是1
        notify_down     /usr/local/etc/keepalived/mysql.sh      检查服务器失败(down)后，要执行的脚本
        TCP_CHECK {         下面是常用的健康检查方式
            connect_timeout 10
            bingto 192.168.1.100        健康检查的IP地址
            nb_get_retry 3              重连次数
            delay_before_retry 3        延迟时间3秒
            connect_port 3306           连接端口3306
        }
    }
```

2台mysql上，启动Keepalived服务

```
# service keepalived start
```

测试2台mysql的故障转移

```
  161  modprobe ip_vs
  162  modprobe ip_vs_wrr
```

```
[root@node1 keepalived]# cat keepalived.conf
! Configuration File for keepalived

global_defs {
   notification_email {
     root@localhost.localdomain
   }
   notification_email_from root@localhost.localdomain
   smtp_server 127.0.0.1
   smtp_connect_timeout 30
```

```
   router_id test-mysql
}
vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 100
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.1.1.200
    }
}
virtual_server 10.1.1.200 3307 {
    delay_loop 1
    lb_algo wrr
    lb_kind DR
    persistence_timeout 5
    protocol TCP
    real_server 10.1.1.10 3307 {
        weight 1
        notify_down /etc/keepalived/mysql.sh
        TCP_CHECK {
            connect_timeout 1
            nb_get_retry 3
            delay_before_retry 3
            connect_port 3307
        }
    }
}


[root@node2 mysql]# cat /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
```

```
   notification_email {
     root@localhost.localdomain
   }
   notification_email_from root@localhost.localdomain
   smtp_server 127.0.0.1
   smtp_connect_timeout 30
   router_id test-mysql-ha
}
vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.1.1.200
    }
}
virtual_server 10.1.1.200 3306 {
    delay_loop 1
    lb_algo wrr
    lb_kind DR
    persistence_timeout 5
    protocol TCP
    real_server 10.1.1.20 3306 {
        weight 1
        notify_down /etc/keepalived/mysql.sh
        TCP_CHECK {
            connect_timeout 1
            nb_get_retry 3
            delay_before_retry 3
            connect_port 3306
        }
    }
}
```

MySQL-Cluster

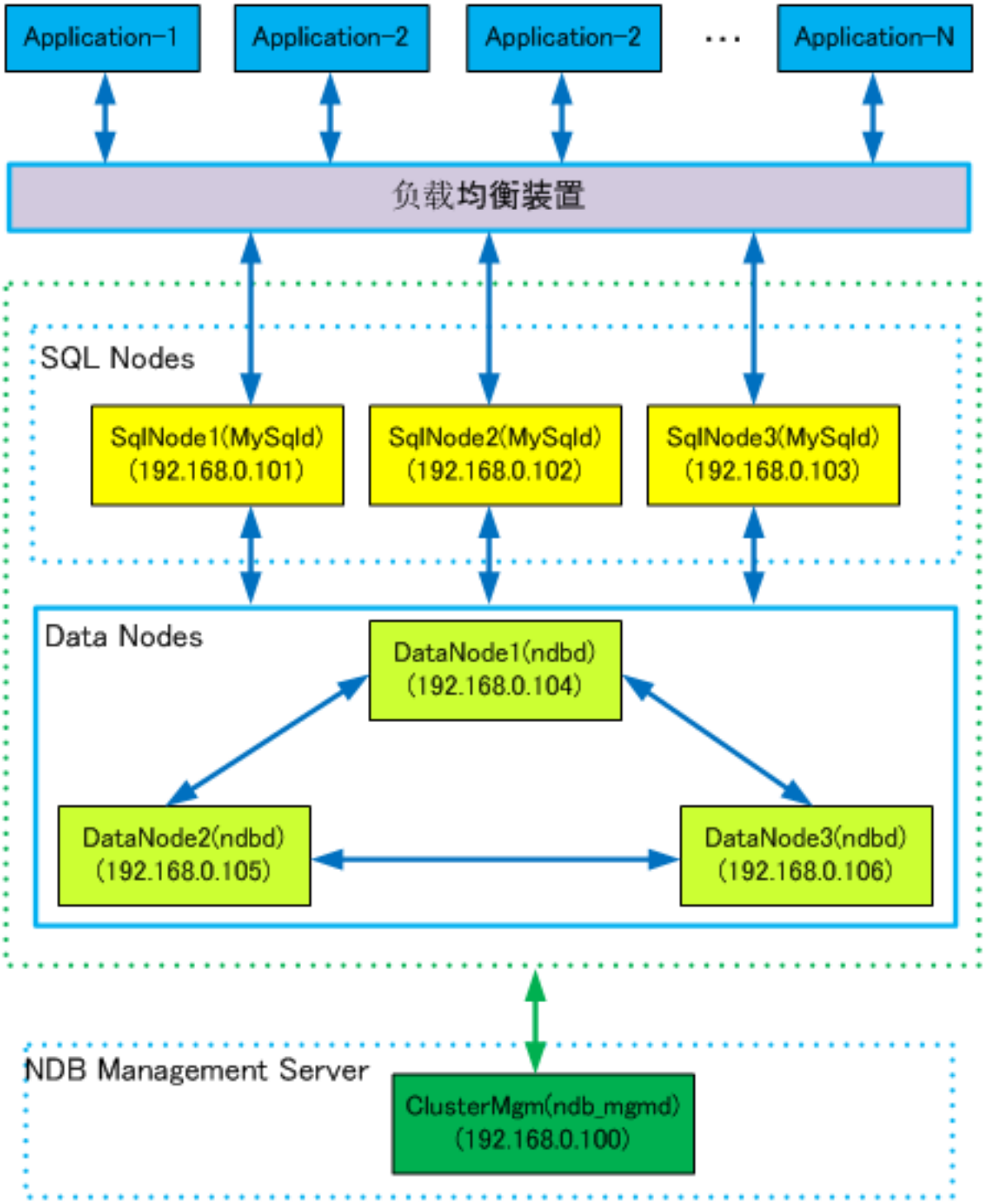
MySQL集群技术 cluster

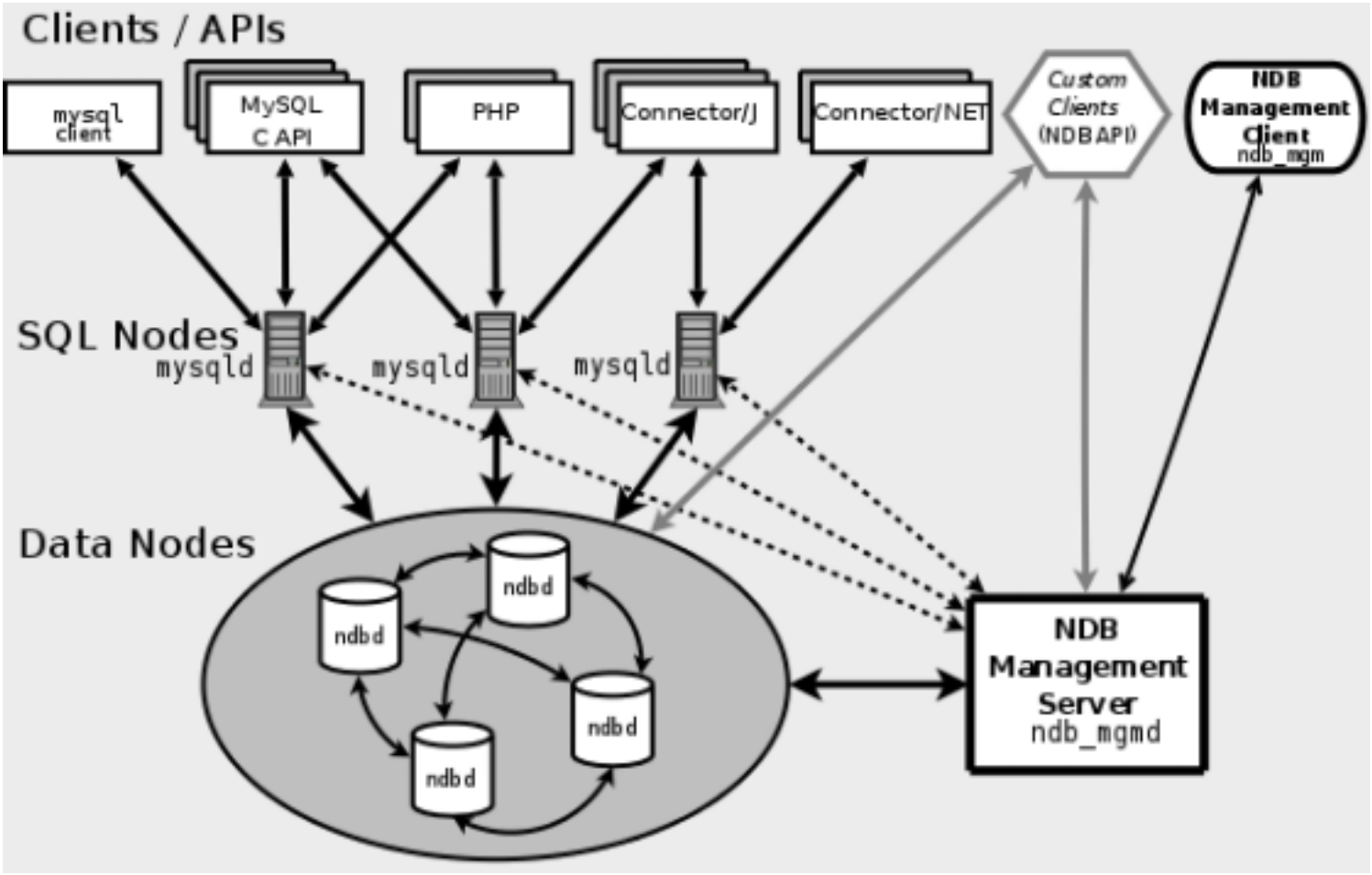
集群成员：

管理节点(MGM)　　进程名ndb_mgmd

数据节点(Data Nodes)　　进程名ndbd

SQL节点(Sql Nodes)　　进程名mysqld


Application-1
Application-2
Application-2
...
Application-N
負載均衡裝置
SQL Nodes
SqlNode1(MySqld)
(192.168.0.101)
SqlNode2(MySqld)
(192.168.0.102)
SqlNode3(MySqld)
(192.168.0.103)
Data Nodes
DataNode1(ndbd)
(192.168.0.104)
DataNode2(ndbd)
(192.168.0.105)
DataNode3(ndbd)
(192.168.0.106)
NDB Management Server
ClusterMgm(ndb_mgmd)
(192.168.0.100)



注意：

1. 所有节点均不安装和启动mysql-server软件包，仅安装集群软件包 mysql-cluster-gpl-7.4.12.tar.gz

2. Mysql Cluster采用的是NDB存储引擎，因此在建表时必须指定ENGINE为ndbcluster，这是一种内存

式的存储引擎，因此对内存要求很高。

==========

环境准备：

10.1.1.2		管理节点 mgm
10.1.1.2	10.1.1.3	SQL节点 sql1 sql1
10.1.1.4	10.1.1.5	数据节点 data1 data2

具体步骤：

一、初始配置（所有节点）

IP、Iptables、SELinux、hostname解析

卸载已经安装了的mysql软件包

二、Installing a MySQL Cluster Binary Release on Linux

SQL nodes.

```
[root@sql1 ~]# useradd mysql
[root@sql1 ~]# tar -xf mysql-cluster-gpl-7.4.12-linux-glibc2.5-x86_64.tar.gz -C /usr/local/
[root@sql1 ~]# ln -sv /usr/local/mysql-cluster-gpl-7.4.12-linux-glibc2.5-x86_64/ /usr/local/mysql
[root@sql1 ~]# cd /usr/local/mysql
[root@sql1 ~]# chown -R mysql.mysql .
[root@sql1 mysql]# scripts/mysql_install_db --user=mysql
[root@sql1 mysql]# cp support-files/mysql.server /etc/rc.d/init.d/mysql-cluster
[root@sql1 mysql]# chmod +x /etc/rc.d/init.d/mysql-cluster
[root@sql1 mysql]# chkconfig --add mysql-cluster
[root@sql1 mysql]# chkconfig mysql-cluster on
```

Data nodes.

```
[root@data1 ~]# rsync -va sql1:/usr/local/mysql/bin/ndbd /usr/local/bin/
[root@data1 ~]# rsync -va sql1:/usr/local/mysql/bin/ndbmtd /usr/local/bin/
[root@data1 ~]# chmod a+x /usr/local/bin/ndb*
```

Management nodes.

```
[root@mgm ~]# rsync -va sql1:/usr/local/mysql/bin/ndb_mgm* /usr/local/bin/
[root@mgm ~]# chmod a+x /usr/local/bin/ndb_mgm*
[root@mgm ~]# mkdir -p /usr/local/mysql/mysql-cluster
```

三、Initial Configuration of MySQL Cluster

Configuring the data nodes and SQL nodes.

```
[root@sql1 mysql]# vim /etc/my.cnf
[mysqld]
# Options for mysqld process:
ndbcluster

[mysql_cluster]
# Options for MySQL Cluster processes:
ndb-connectstring=10.1.1.10      【管理节点】

[root@sql1 mysql]# rsync -va /etc/my.cnf sql2:/etc/
[root@sql1 mysql]# rsync -va /etc/my.cnf data1:/etc/
[root@sql1 mysql]# rsync -va /etc/my.cnf data2:/etc/
```

Configuring the management node.

```
[root@mgm ~]# cd /usr/local/mysql/mysql-cluster
[root@mgm mysql-cluster]# vim config.ini
[ndbd default]
# Options affecting ndbd processes on all data nodes:
NoOfReplicas=2          # Number of replicas
DataMemory=80M    # How much memory to allocate for data storage
IndexMemory=18M   # How much memory to allocate for index storage
                        # For DataMemory and IndexMemory, we have used
the
                        # default values. Since the "world" database takes up
                        # only about 500KB, this should be more than
enough for
                        # this example Cluster setup.

[tcp default]
# TCP/IP options:
portnumber=3306         # This the default; however, you can use any
                        # port that is free for all the hosts in the cluster
                        # Note: It is recommended that you do not specify
the port
                        # number at all and simply allow the default value to
be used
                        # instead

[ndb_mgmd]
# Management process options:
id=1
hostname=10.1.1.2       # Hostname or IP address of MGM node
datadir=/usr/local/mysql/mysql-cluster  # Directory for MGM node log files

[mysqld]
# SQL node options:
id=2
hostname=10.1.1.2

[mysqld]
# SQL node options:
id=3
hostname=10.1.1.3
```

```
[ndbd]
# Options for data node "data1":
id=4
hostname=10.1.1.4       # Hostname or IP address
datadir=/usr/local/mysql/data   # Directory for this data node's data files

[ndbd]
# Options for data node "data2":
id=5
hostname=10.1.1.5       # Hostname or IP address
datadir=/usr/local/mysql/data   # Directory for this data node's data files
```

四、Initial Startup of MySQL Cluster

1. Management nodes

启动管理节点：

```
[root@mgm ~]# ndb_mgmd -f /usr/local/mysql/mysql-cluster/config.ini
```

2. data node

启动数据节点：

```
[root@data1 ~]# ndbd
2015-02-17 10:46:23 [ndbd] INFO     -- Angel connected to '10.1.1.2:1186'
2015-02-17 10:46:23 [ndbd] INFO     -- Angel allocated nodeid: 4

[root@data2 ~]# ndbd
2015-02-17 10:47:15 [ndbd] INFO     -- Angel connected to '10.1.1.2:1186'
2015-02-17 10:47:15 [ndbd] INFO     -- Angel allocated nodeid: 5
```

~~# echo "ndbd" >> /etc/rc.local   //设置数据节点开机运行~~

3. SQL node

```
[root@sql1 mysql]# /etc/init.d/mysql-cluster start
Starting MySQL.....                        [  OK  ]
[root@sql1 mysql]# chkconfig mysql-cluster on

[root@sql2 mysql]# /etc/init.d/mysql-cluster start
Starting MySQL.....                        [  OK  ]
[root@sql2 mysql]# chkconfig mysql-cluster on
```

4. Management nodes

管理节点监控集群状态：
[root@mgm ~]# ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> show
Connected to Management Server at: 10.1.1.2:1186
Cluster Configuration

[ndbd(NDB)] 2 node(s)
id=4 @10.1.1.4 (mysql-5.6.31 ndb-7.4.12, Nodegroup: 0, *)
id=5 @10.1.1.5 (mysql-5.6.31 ndb-7.4.12, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1 @10.1.1.2 (mysql-5.6.31 ndb-7.4.12)

[mysqld(API)] 2 node(s)
id=2 @10.1.1.2 (mysql-5.6.31 ndb-7.4.12)
id=3 @10.1.1.3 (mysql-5.6.31 ndb-7.4.12)

五、测试：

1. 在所有SQL节点建立授权用户

[root@sql1 ~]# mysql
mysql> grant all on *.* to root@'%' identified by '456';
Query OK, 0 rows affected (0.04 sec)
mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

2. 从客户端连接测试

[root@client1 ~]# mysql -h 10.1.1.3 -uroot -p123

测试一：所有节点均正常，写入或查询
mysql> create table t1(id int)engine=ndbcluster;
mysql> alter table uplooking.t1 engine=ndbcluster; //改变已创建表的存储引擎

测试二：停用一个SQL节点
[root@sql1 ~]#service mysql-cluster stop

测试三：停用一个数据节点

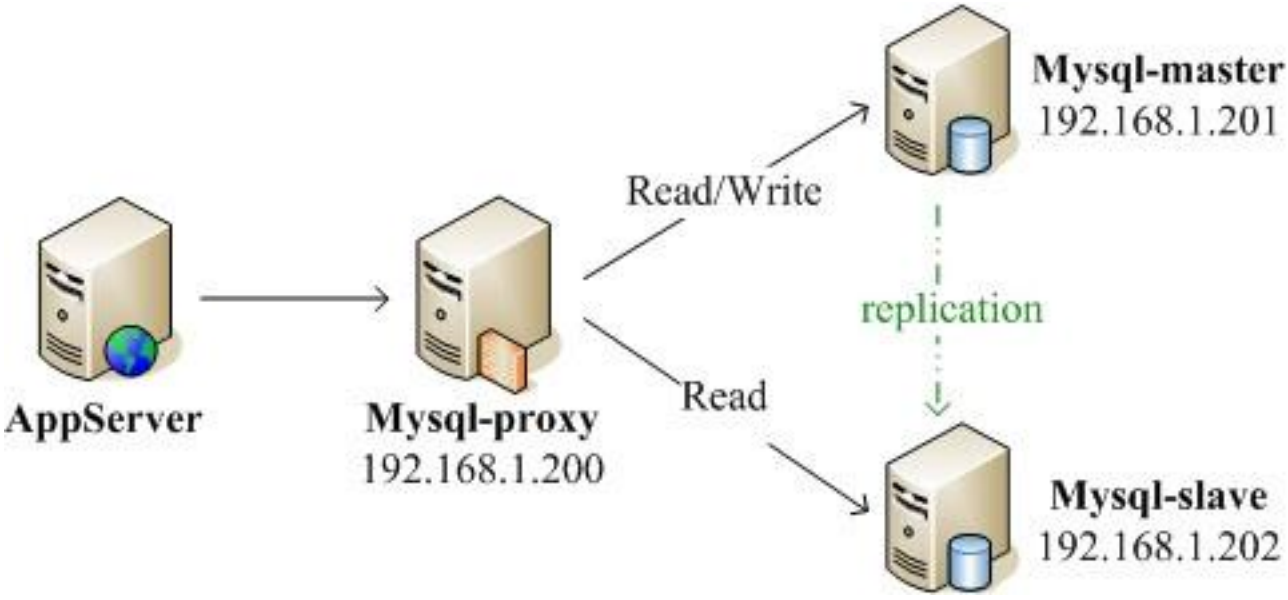
[root@mgm ~]# ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm>4 stop

++
++++++++++++++++
web界面自动部署

host1 host2 host3

mysql-proxy

MySQL Proxy :
==========



1、安装mysql-proxy
rpm -ivh mysql-proxy-0.8.1-1.el6.x86_64.rpm

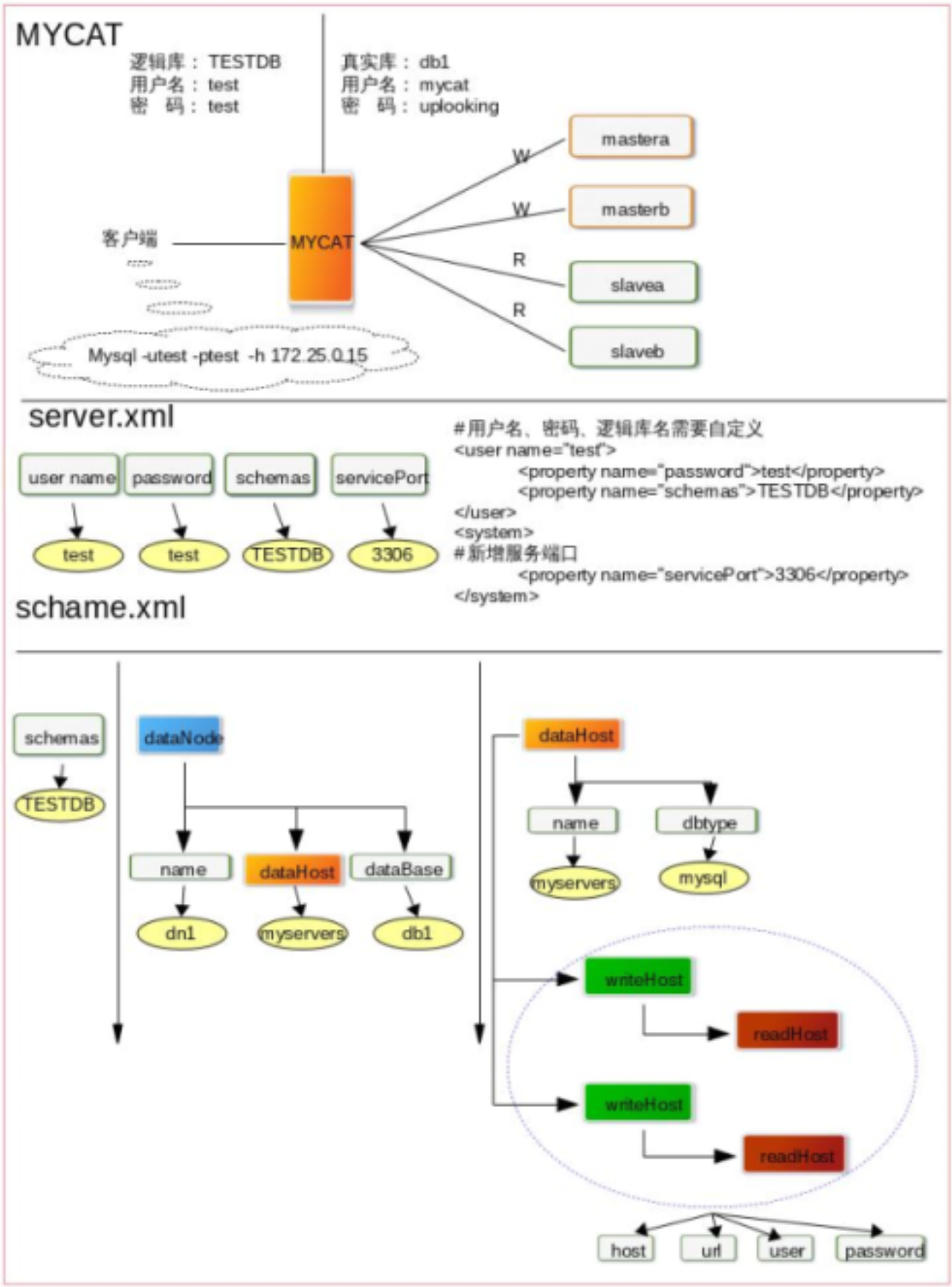
2、将服务器上的rw-splitting.lua 复制到/lib 下
cp rw-splitting.lua /lib

3、修改/etc/sysconfig/mysql-proxy
添加如下代码：[以下代码需写在一行内]
PROXY_OPTIONS="--proxy-address=10.1.1.10:3306 --proxy-read-only-backend-addresses=10.1.1.20:3307 --proxy-backend-addresses=10.1.1.20:3306 --proxy-lua-script=/lib/rw-splitting.lua --daemon"

4、设置mysql-proxy 开机启动，并启动mysql-proxy
chkconfig mysql-proxy on
service mysql-proxy start

5、授权mysql-proxy访问
mysql -p123
grant all on *.* to 'proxy'@'10.1.1.10' identified by '123';
grant all on *.* to 'proxy'@'vm1.uplook.com' identified by '123';
flush privileges;

mycat-wr



Mycat-server-1.5.1-RELEASE-20160328130228-linux.tar.gz

server.xml

```
#用户名、密码、逻辑库名需要自定义
<user name="test">
   <property name="password">test</property>  注意：此处test是密码
   <property name="schemas">TESTDB</property>
</user>
<system>
#新增服务端口
   <property name="servicePort">3306</property>
</system>
```

schema.xml

```
<schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100">
</schema>
<dataNode name="dn1" dataHost="localhost1" database="db01" />
<dataHost name="localhost1" maxCon="1000" minCon="10" balance="3"
            writeType="0" dbType="mysql" dbDriver="native" switchType="1"
slaveThreshold="100">
            <heartbeat>select user()</heartbeat>
            <!-- can have multi write hosts -->
            <writeHost host="master1" url="10.1.1.1:3306" user="mysqlproxy"
                  password="uplooking">

            <readHost host="slave1" url="10.1.1.2:3306" user="mysqlproxy"
                  password="uplooking" />
            <readHost host="slave2" url="10.1.1.3:3306" user="mysqlproxy"
                  password="uplooking" />
            </writeHost>

            <writeHost host="master2" url="10.1.1.10:3306" user="mysqlproxy"
                  password="uplooking">

            <readHost host="slave1" url="10.1.1.2:3306" user="mysqlproxy"
                  password="uplooking" />
            <readHost host="slave2" url="10.1.1.3:3306" user="mysqlproxy"
                  password="uplooking" />
            </writeHost>
```

</dataHost>
</mycat:schema>
测试

mycat的master每次之能用一个来写，以这种方法来防止两台master产生冲突。
当其中一台master宕机以后，可以自动切换。

balance 属性

负载均衡类型有 3 种:

1. balance="0", master可读可写，slave不使用，也就是不启动读写分离
2. balance="1",master可读可写，slave只读；stand by writeHost 参与 select 查询做负载均衡
3. balance="2",master可读可写，slave只读；所有读操作都随机的在 master和slave上分配。
4. balance="3",master只写，slave只读；1.4版本以后有

switchType 属性
-1 表示不自动切换
1 默认值,自动切换
2 基于 MySQL 主从同步的状态决定是否切换

更多内容 请查看《Mycat 权威指南》