解决的问题

数据分布

负载均衡

备份

高可用性和容错性//几台机器挂掉也没关系

# 重点：

主服务器在二进制日志中跟踪所有对数据库的更改

从服务器使用I/O线程接受主服务器的二进制日志中保存的更新。将其拷贝到它的中继日志中。

slave使用SQL线程重做中继日志中的事件。

三个库的userid如果用默认的主键会冲突，可以用序列解决。当然mycat有自带的解决方案。

从库出问题可以用change master to重新指定主库的位置.

GTID不用指定二进制日志的偏移量。

GTID是全局事务ID，保证为每一个在主上提交的事务在复制集群中可以生成一个唯一的ID。

SwitchType=”2”通过心跳检测语句自动切换。

Grant语句，change master to语句

vim /etc/my.cnf

log-bin=mysql-bin 开启二进制日志，mysql-bin是二进制日志文件名

server-id 唯一的值，定义不要重复

/usr/local/mysql/var 二进制文件在这里被找到。

File Position

mysql-bin.000003 154 将来写操作，值就会增大。

000003写满了就开000004

reset master 就会从000001开始写

从服务器：

可以注释掉bin-log

增加relay\_log = mysql\_relay-log

在主服务上:grant replication slave on \*.\* to slave@192.168.199.152 identified

by ‘123456’

授权从服务器192.168.199.152通过密码123456的给所有库的所有表

在从服务器上配置master

stop slave

151一开始作为主

change master to matser\_host='192.168.199.151',

master\_port=3306,

master\_user='slave',

master\_password='123456',

master\_log\_file='mysql-bin.000001',

master\_log\_pos=452;

start slave

show slave status\G看Slave\_IO\_Running,Slave\_SQL\_Running都为yes就成功。

mysql双主热备

将刚才的从服务器也作为master。二进制，中继日志blablabla...

将之前注释掉的log\_bin打开.

log\_slave\_updates=1 将relay\_log又写入bin\_log中。

在从服务器上grant replication slave on \*.\* to master@192.168.199.151 identified by

'123456';

在主服务器上 配置relay\_log=mysql-relay-bin

log\_slave\_updates =1

然后配置change master to....

总结 在两个主服务上都要配置relay\_log和log\_bin。都还要grant和change master....和log\_slave\_updates

Mycat

支持常见sql语法

支持读写分离，支持主从，心跳故障自动切换

配置server.xml和schema.xml就可以了

server.xml 连接mycat

<user name=’root’>

<property name=”password”>123456</property>

<property name=”schemas”>shop</property>

<property name=”readonly”>true</property>//如果配置了就只能查

</user>

通过用户名root密码123456连接shop库（是一个映射，如这个例子映射imooc\_shop）

Schema.xml

Schema标签 的name=shop和上面配置的名字要一致，用来管理它。

dataNode=”dn1”的名字配置以后，就交给对应的datanode标签进行管理。

//checkSQLschema 为true带表名就自动去除

sqlMaxlimit在SQL语句后自动加上limit 100

Datanode标签 name=”dn1”

database=”imooc\_shop”这个是实际的数据库名称。

datahost=”localhost1”这个又对应了下面的一个datahost标签。

DataHost标签 name=”localhost1”对应了上面的一个datanode。

Heartbeat指定了一个sql语句如show slave status，执行结果有问题就代表死掉，切换。

Balance=”3”负载均衡类型。所有读请求随机分发到wirtehost对应的readhost。writeHost不负载读操作。

writeType=”0”所有写操作发到第一个wirtehost挂了就切换到第二个。

SwitchType=”2”通过心跳检测语句自动切换。

SlaveThreshold=”100”检测从数据库的数据不能太久超过100秒。

writeHost标签对应要写的主机。host=”hostM1”

url=”192.168.199.151:3306” user=”mycat” password=”123456”

然后又要去配置151的grant insert,update,delete,select on \*.\* to [mycat@192.168.199.100](mailto:mycat@192.168.199.100) identified by “123456”;

Writehost标签里面写readHost。

readHost标签，也要用grant授权给mycat，和writeHost的属性差不多。

开发机Mysql -uroot -p123456 -h192.168.199.100 -P8066链接mycat

分库分表

在schema标签里面加入table name=”shop\_cart”和真实表名字要一致。

Primarykey=”cartid”

dataNode=”cart$1-3”

rule=”sharding-by-murmur”表进行分片的规则。

Rule.xml文件里面 tablerule标签 name=” sharding-by-murmur”

Columns标签比如userid对哪个字段进行一致性哈希处理

dataNode标签 name=”cart1” database=”shop\_cart1”

dataNode标签 name=”cart1” database=”shop\_cart2”

dataNode标签 name=”cart1” database=”shop\_cart3”

三个库的userid如果用默认的主键会冲突，可以用序列解决。当然mycat有自带的解决方案。

Sequence\_conf.proper文件

插入时候用Next value for MYCATSEQ\_CART

LVS

使用IP负载均衡技术，通过ipvs内核模块实现。

安装在负载均衡服务器上，并且虚拟出一个IP地址。

Real Server如何返回数据给用户，是重点技术，有三种机制

NAT TUN DR

NAT：

VIP配置在DR上

将受到请求的集群服务器请求报文目标地址转换成根据算法计算得出的后端主机IP地址。

然后后端主机将相应报文发送至DR

再由DR将源地址转换成VIP地址。

需要双网卡 DIP（内网）DIP和后端主机IP在同一个网络中，后端主机网关是DIP。

支持端口映射

VIP（外网）

Director容易成为瓶颈。

Route del default

Route add default gw 192.168.199.101

开启IP转发 echo “1” >/proc/sys/net/ipv4/ip\_forward

Ipvsadm -C

Ipvsadm -A -t 10.16.0.201:80 -s rr

Ipvsadm -a -t 10.16.0.201:80 -r 192.168.199.161:80 -m

Ipvsadm -a -t 10.16.0.201:80 -r 192.168.199.162:80 -m

DR模式

Client发送一个请求给VIP

VIP收到请求后会跟LVS设置的LB（负载均衡）算法选择一个LB比较合理的Real SERVER.

然后将此请求数据包的MAC地址修改为Real Server的MAC地址。

ARP协议会把这个包发送给真正的Real Server

RealServer收到这个数据包以后，首先判断目标IP是否是自己的IP地址，如果不是就丢掉这个包，如果是处理这个包。

在所有的Real Server的极其上面都绑定VIP的IP地址。

Real Server处理这个包以后，会将目标IP设置为clientIP直接发送，不经过LVS。效率高

Real Server的网关不指向DIP，指向外部路由。

一、Master&Slave是什么？

         也就是我们所说的主从复制，主机数据更新后根据配置和策略，自动同步到备机

的master/slaver机制，Master以写为主，Slave以读为主。

二、它能干嘛？

       1、读写分离；

       2、容灾恢复。

三、怎么玩？

       1、配从（库）不配主（库）；

       2、从库配置：slaveof [主库IP] [主库端口]；

           补充：每次slave与master断开后，都需要重新连接，除非你配置进redis.conf文件;

                      键入info replication 可以查看redis主从信息。

       3、修改配置文件细节操作

             - 拷贝多个redis.conf文件

https://img-blog.csdn.net/20171113205507601?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvemhhbmdndWFuZ2h1aTAwMg==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

             - 指定端口

https://img-blog.csdn.net/20171113205529424?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvemhhbmdndWFuZ2h1aTAwMg==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

             - 开启daemonize yes

https://img-blog.csdn.net/20171113205549901?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvemhhbmdndWFuZ2h1aTAwMg==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

             - Pid文件名字

https://img-blog.csdn.net/20171113205608147?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvemhhbmdndWFuZ2h1aTAwMg==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

             - Log文件名字

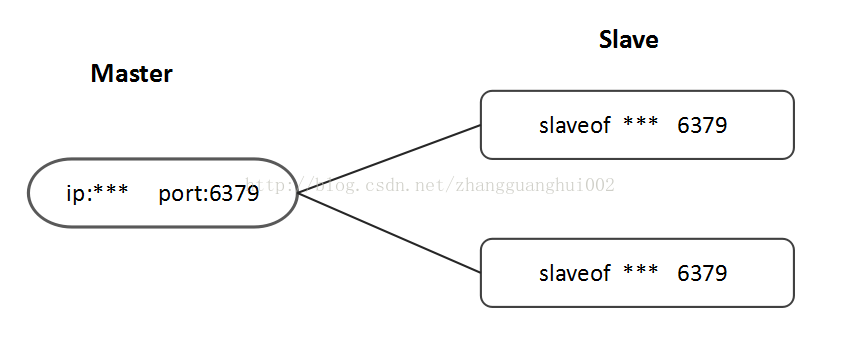
https://img-blog.csdn.net/20171113205626714?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvemhhbmdndWFuZ2h1aTAwMg==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

             - Dump.rdb名字

https://img-blog.csdn.net/20171113205641755?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvemhhbmdndWFuZ2h1aTAwMg==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

        4、常用3招

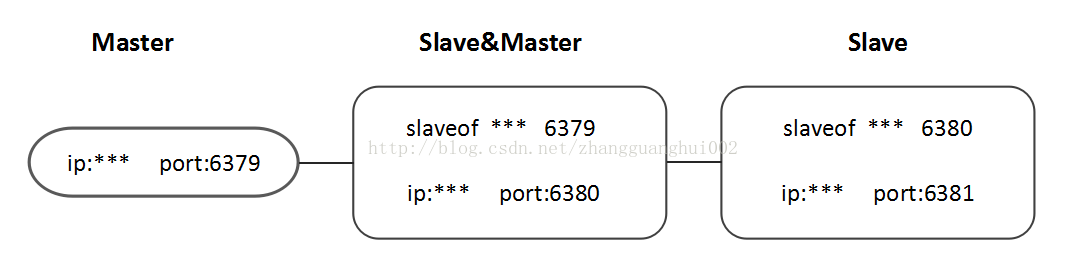
              - 一主二仆



                  一个Master，两个Slave，Slave只能读不能写；当Slave与Master断开后需要重新slave of连接才可建立之

           前的主从关系；Master挂掉后，Master关系依然存在，Master重启即可恢复。

           - 薪火相传



                 上一个Slave可以是下一个Slave的Master，Slave同样可以接收其他slaves的连接和同步请求，那么该slave作为了

           链条中下一个slave的Master，如此可以有效减轻Master的写压力。如果slave中途变更转向，会清除之前的数据，重新

           建立最新的。

           - 反客为主

                       当Master挂掉后，Slave可键入命令 slaveof no one使当前redis停止与其他Master redis数据同步，转成

              Master redis。

四、复制原理

       1、Slave启动成功连接到master后会发送一个sync命令；

       2、Master接到命令启动后的存盘进程，同时收集所有接收到的用于修改数据集命令，在后台进程执行完毕之后，master

            将传送整个数据文件到slave，以完成一次完全同步；

       3、全量复制：而slave服务在数据库文件数据后，将其存盘并加载到内存中；

       4、增量复制：Master继续将新的所有收集到的修改命令依次传给slave，完成同步；

       5、但是只要是重新连接master，一次完全同步（全量复制）将被自动执行。

五、哨兵模式（sentinel）

       反客为主的自动版，能够后台监控Master库是否故障，如果故障了根据投票数自动将slave库转换为主库。一组sentinel能

       同时监控多个Master。

       使用步骤：

       1、在Master对应redis.conf同目录下新建sentinel.conf文件，名字绝对不能错；

       2、配置哨兵，在sentinel.conf文件中填入内容：

             sentinel monitor 被监控数据库名字（自己起名字） ip port 1

             说明：上面最后一个数字1，表示主机挂掉后slave投票看让谁接替成为主机，得票数多少后成为主机。

      3、启动哨兵模式：

            命令键入：redis-sentinel  /myredis/sentinel.conf

           注：上述sentinel.conf路径按各自实际情况配置

六、复制的缺点

            延时，由于所有的写操作都是在Master上操作，然后同步更新到Slave上，所以从Master同步到Slave机器有一定

       的延迟，当系统很繁忙的时候，延迟问题会更加严重，Slave机器数量的增加也会使得这个问题更加严重。