修改桌面: usr > share > backgrounds

庞丽静　　静静

笔记存放位置:http://github.com/yaya9998/notedir

知识多 练习时间少 命令多 理论多

08:30
 9:00 - 12:00　 40　15
 10:00 -18:00
 18:40- 20:45
comment-dayX.txt
172.40.50.117
2018/notedir/command-dayX.txt tmooc
PPT/
soft/
++

panglj@tedu.cn
Linux 系统管理

第三阶段 15 天　数据库管理　　工作岗位　DBA　8K+
DBA 基础 5 天
DBA 进阶 5 天

NoSQL　5 天

ssh　172.40.55.X

++

DBA1 day01

一搭建数据库服务器

二数据库服务的基本使用

三 MySQL 数据类型

一搭建数据库服务器

在 IP 地址是 192.168.4.50 主机 上运行 mysql 数据库服务。

服务器 CPU　内存　存储

操作系统的版本：W　L　U

rhel7

5.7

装包：源码　RPM　包的来源　软件的版本

RDBMS　NoSQL　Redis　Mongodb

主流数据库服务软件：Oracle　DB2　MySQL　MariaDB

SQL SERVER

开源软件

商业软件

跨平台

]#scp mysql-5.7.17.tar 192.168.4.50:/root/

]#tar -xf mysql-5.7.17.tar

]# ls *.rpm

]#yum -y install perl-JSON

]#rpm -Uvh mysql-community-*.rpm

]#rpm -qa | grep -i mysql

修改配置文件 /etc/my.cnf

数据库目录 /var/lib/mysql/

启动服务

systemctl start mysqld

tar -xvf xxx.tar

rm -rf

mysql-community-server-minimal-5.7.17-1.el7.x86_64.rpm

yum -y install perl-JSON

rpm -Uvh mysql-community-*.rpm

systemctl start mysqld

php java

Nginx apache tomcat

LNMP LAMP

mysql Mongodb

游戏 购物 论坛 金融

xxxx.php

MySQL-5.7

把数据存储到数据库服务器上过程

1 连接数据库服务器 （命令连接 图形工具）

]#mysql -hlocalhost -uroot -p 密码

]#grep password /var/log/mysqld.log

]#mysql -hlocalhost -uroot -p'k2T_SH0Jx%MB'

mysql> set global validate_password_policy=0;

mysql> set global validate_password_length=6;

mysql> alter user root@"localhost" identified by "123456";

mysql 数据库密码修改:

grep -i password /var/log/mysqld.log 查看数据库 root 初始密码

mysql> set global validate password policy=0; 是否检测密码

mysql> set global validate password length=6; 设置密码长度

mysql> alter user root@"localhost" identified by "123456" 修改密码为 123456

mysql -hlocalhost -uroot -p123456 -hlocalhost: 本机数据库

mysql> select database(); 显示在哪个库下

mysql> use 库名 进入某个数据库

mysql> show tables; 查看数据库中的数据表

mysql> create database 库名; 创建新库

mysql> drop database 库名; 删除数据库

mysql> show databases; 显示已有的库

mysql>show processlist; 查看列出程序列表

mysql 数据表的增删改查

增加数据表：

create table 表名.库名(列名 字段,列名 字段);

例 ： mysql> create table db1.t1(name char(15),class char(7));

desc 表名 进入数据库查看表结构

select * from 表名; 查看表记录

drop table 表名； 删除表

mysql> insert into 表名 values("liangxiang","nsd1803"),("bob","nsd1803"); 数据表中增加数据

mysql>

update 表名 set class="nsd1809"; 批量改内容

mysql> delete from 表名; 删除表

mysql> show create table 表名; 显示之前建表的命令

案例：创建一个学生库并在里面写入内容

mysql> create database 学生库;

mysql> show databases;

mysql> use 学生库;

mysql> create table 学生信息表(姓名 char(15),班级 char(7))

mysql> show tables;

mysql> desc 学生信息表;

mysql> create table 学生信息表 2(姓名 char(15),班级 char(7)) DEFAULT CHARSET=utf8; (DEFAULT CHARSET=utf8 字符集,可以显示中文)

mysql> insert into 学生信息表2 values("齐吉祥","nsd1803"),("甘钱","nsd1803"),("胡思超","nsd1803"),("梁湘","nsd1803");

mysql> select * from 学生信息表2;

重点：mysql 数据类型

数值型：体重，身高，成绩，工资

字符(char)：姓名，工作单位，住址

枚举型：兴趣爱好，性别

日期时间型：出生日期，注册时间

数值类型

整型　　存整数　　如：18

有符号的存储范围：负数

无符号的存储范围：正数　如：年龄

浮点型 float，存小数　　如：1.8

PRI

例：

有符号：

create table db1.t2(level tinyint);

insert into t2 values(127);

insert into t2 values(-128);

无符号：

create table db1.t3(level tinyint unsigned);

insert into t3 values(256);

整数.小数　　总位数

　　n　　m

1098.88　　6

字段名　　类型

　　float（m,n）

　　double（m,n）

create table db1.t4(pay double(5,2))

insert into 库名.表名　values(900);

字符类型：

定长：char（字符数）

最大长度 0～255 字符

指定宽度，不够用空格代替

变长：varchar（字符数）

按数据实际大小分配存储空间

create table t5(name char(15),email varchar(30));

insert into t5 values("bob","bob@163.com"),("lx","lx@163.com");

枚举类型: 字段的值必须在列举范围内选

enum（值 1,值 2,值 N）　单选

set （值 1,Using a pa 值 2,值 N）　多选

create table 库名.表名(name char(15),age tinyint unsigned, pay float(7,2) sex enum("boy","girl","no"),likes set("woman","money","game","eat"));

desc t6; 查看表结构

insert into t6 values("lx",21,20000,"boy","eat,game");

日期时间类型：

年 year 赋值默认要求 4 位数

日期 date 赋值默认要求 4 位数

时间 time 赋值默认要求两位时间。两位分钟，两位秒

create table t7(name char(15),syear year,birthday date,up_class time,party datetime);

insert into t7 values("lx",1992,20181120,083000,20180618220000);

select * from t7

insert into t7 values("qq",1999,20181020,090000,20180718220000);

日期时间类型 PRI： YYYYMMDDHHMMSS 赋值默认要求四位数的年，两位数的日期的时间

mysql 内置时间函数的使用

使用时间函数获取数据给日期时间类型字符段赋值

datetime 与 timestamp 的区别

数值类型宽度数字符类型宽度的区别

age int(2)

name char(2)

使用时间函数 PRI 获取数据给日期时间类型字符段赋值 ：

insert into db1.t7 values($y,$w)

函数的格式：

括号里面不需要加值：

select now();

select curdate();

括号里面需要加值：

select year(now());

select date(now());

select month(now());

select time(now());

例：mysql> select year(20191112）;

用时间函数赋值：

mysql> insert into t7 values("lxx",year(19920101),date(now()),083000,now());

datetime 与 timestamp 的区别:

1　存储的时间范围不一样

2　赋值的方式不一样

datetime 占用 8 个字节

timestamp 占用 4 个字节

create table t8(name char(15),meetting

datatime,party timestamp);

datetime 方式赋值:

insert into t8 values("lx",now(),now());

timestamp 方式赋值:

insert into t8(name,meetting) values("df",20191120205059);

insert into t8(name,party) values("df",20150919205059);

数值类型宽度数字符类型宽度的区别

age int(2) 显示宽度 2

name char(2)

create table t10(level int(2));

insert into t10 values(128);

create table t11(level int); 不设置，默认宽度 11

create table t12 （ name char(5),levle int(3) zerofill,money int(5) zerofill);

insert into t12 values("swk",18,29);

insert into t12 values("zbj",8,2132);

insert into t12 values("ts",888,21322);

insert into t12 values("fz",8888888,223546781322);

select * from t12;

作业：

1,在 51 到 55 虚拟机上创建数据库并设置数据库管理本机登录密码 123456

2，在 persondb 库下创建保存前任信息的表名叫

dogperson（字段定义表结构）

回顾

搭建 mysql 数据库服务器

装包　默认配置运行　启动服务　查看服务器运行状态

使用初始密码登录并设置登录密码

把数据存储到数据库服务器上的步骤：

1　连接服务器　mysql -hlocalhost　-u 用户名 -p 密码

2 库 ：　查看　创建　删除　切换　查看已有的表　显示当前所在的库

3 表：　查看　创建　删除

表记录的管理：　select 查　update 改　delete 删　insert 写

4 断开连接　quit　exit　\q

mysql 数据类型

数值类型：

整型　浮点型

tinyint unsigned　float

字符类型　char　varchar

日期时间　年　日期　时间　日期时间

枚举类型　enum 单选　set 多选

DBA2 day02

字段约束条件

修改表结构

mysql 键值

字段约束条件

create table t14(name char(5) not null,level int(3) zerofill default 0,monet tinyint(2) zerofill default 0);

insert into t14(name) values("natasha");

create table t15(name char(5) not null,level int(3) zerofill default 0,monet tinyint(2) zerofill default 0);

修改表结构

语法格式：

alter table 表名 执行动作

添加新字段：新字段会添加在已有字段的后面

alter table t15 add email varchar(30) default "stu@tedu.cn",add tel char(11);

alter table t15 add stu id char(9) first; 在第一列添加列

在 name 字段后面添加字段

alter table t15 add sex enum("boy","girl","no") defaulet "no" after name;

desc t15;

修改字段类型

alter table t15 modify stu_id varchar(10);

alter table t15 modify name char(10);

alter table t15 modify tel char(11) after name;

删除表中的列

mysql>alter table t15 drop stu_id,drop tel;

改字段名

mysql>alter table t15 change email maill varchar(30) default "stu@tedu.cn";

mysql>alter table t15 change mail Emaill varchar(50) default "stu@163.com";

修改表名

mysql>alter table t15 rename stuinfo;

mysql 键值

mysql 索引类型

普通索引 index （Btree B+Tree hash）

唯一索引 unique

主键　primary key

外键　foreign key

全文索引　fulltext

创建规则　　创建　　删除　查看

字典查找方法：

目录　　1 - 200

拼音　　a - z

笔画　　1 -20

部首

正文 201-5000 页

index

stuinfo ----> /var/lib/mysql/db1/stuinfo.*

name age class homeaddr

查找表中的数据

mysql>select * from db1.stuinfo where name="zhangsan";

普通索引 index

创建索引：

mysql>create table t16(name char(10),age int(2),class char(7),index(name),index(class));

查看

mysql>desc t16;

mysql>show index from t16\G

删除 class 索引

mysql>drop index class on t16;

删除后再将 class 索引添加回来

mysql>create index class on t16(class);

在已有表里创建索引名

create index 索引名 on stuinfo(name);

BTREE 二叉树

主键 primary key

一个表中只能有一个主键

设置了主键后，内容不能重复和为 null

两种创建表和主键格式

mysql>create table t17(stu_id char(9) primary key,name char(10));

mysql>create table t18(stu_id char(9),name char(10),primary key(stu_id));

添加内容

mysql>insert into t17 values("nsd180301","tom");

删出 t17 表里面的主键,表里面谁是主键删谁

mysql>alter table t17 drop primary key;

在已有表里添加主键

mysql>alter table t17 add primary key(stu_id);

复合主键 ：一个字段以上的主键

不同时重复就可以创建

创建复合主键

mysql>create table t19(cip char(15),serport smallint(2),status enum("yes","no") ， primary key(cip,serpoet));

插入内容：

mysql>insert into t19 values("1.1.1.1",22,"no"),("1.1.1.1",21,"yes");

删除复合主键

mysql>alter table t19 drop primary key;

在已有表里添加复合主键

mysql>alter table t19 add primary key(cip,serport);

auto_increment 字段值自增长 不会有重复

创建字段值自增长

mysql>create table t20(stu_id int(2) primary key auto_increment,name char(10),age tinyint(2));

添加内容

mysql>insert into t20(name,age) values("bob",19);

mysql> insert into t20 values(19,"jerry",29);

在已有表中添加字增长字段

mysql>alter table stuinfo add id int(2) primary key auto_increment first;

外键　foreign key

条件：

表的存储引擎必须是 innodb

字段类型和宽度要一致

被参考的字段要是索引类型的一种，通常是普通索引

创建外键格式：

on update cascade:同步更新

on delete cascade:同步删除

engine=innodb：存储引擎

gzb：工资表

ygb：员工表

yg ：员工

name: 姓名

bumen：　部门

mysql>create table gzb(字段列表，foreign key(字段名) references ygb(yg_id) on update cascade on delete cascade)engine=innodb;

参考表(员工表)：

mysql>create table ygb(yg_id int(2) primary key auto_increment,name char(15),bumen char(20))engine=innodb;

插入内容

mysql>insert into ygb(name,bumen) values("bob","tea"),("jack","tea");

查看：

mysql>select * from ygb;

创建工资表

mysql>create table gzb(gz_id int(2),name char(15),pay float(7,2),bumen char(20),foreign key(gz_id) references ygb(yg_id) on update cascade on delete cascade)engine=innodb;

查看表中的外键

mysql>show create table gzb

插入内容

mysql>insert into gzb values(1,"bob",20000,"eat");

mysql>select * from gzb;

插入信息

mysql>insert into gzb values()

同步更新(更改)

mysql>update ygb set yg_id=8 where name="bob";

同步删除(删除)

mysql>delete from ygb where yg_id=2;

清空数据表中 ygb 中的内容:

mysql>delete from gzb;

工资表中当前字段的内容不能重复(在字段前添加一个主键)

mysql>alter table gzb add primary key(gz_id);

删除 ygb 中的全部内容：

mysql>delete from ygb;

删除表中的外键:

1,mysql> show create table gzb;　　查询外键

2,CONSTRAINT `gzb_ibfk_1`　　里面有一条这样的内容

3,alter table gzb drop foreign key gzb_ibfk_1; 删除外键

DBA3 day03

数据导入，导出

管理表记录（增删改查）

匹配条件 1

数据导入，导出

数据导入:

导入：把系统文件的内容存储到数据库的表里，文件的内容应该有一定规律的

1，创建存储文件内容的表

2，执行导入数据的 sql 命令

格式：

load data infile "目录名/文件名"

into table 表名

[root@db50 ~]#cp /etc/passwd /var/lib/mysql-files

mysql>create table user(name char(30),password char(1),uid int(2),gid int(2),comment char(100),homedir char(150),shell char(50));

查看文件在那个目录下：

mysql>show variables like "secure_file_priv";

在数据库里面执行命令:（将文件拷贝到文件夹下）

mysql> system cp /etc/passwd /var/lib/mysql-files;

mysql> system ls /var/lib/mysql-files;

将文件导入数据库: \n：换行

格式：

mysql>load data infile "文件所在的绝对路径" into table 表名 fields terminated by "分隔符" lines terminated by "转行";

将/var/lib/mysql-files/passwd 的文件导入 user 表中

mysql>load data infile "/var/lib/mysql-files/passwd" into table user fields terminated by ":" lines terminated by "\n";

导入成功后添加行号字段：

mysql>alter table user add id int(2) primary key auto_increment first;

改以下导入文件时，搜索文件的目录（不是重点）

1，在主配置文件中/etc/my.conf 添加：

secure_file_priv=/dirdata

数据导出：

表记录存储到系统文件里

格式：

sql 查询 into outfile "目录/文件名"；

mysql>select * from user into outfile "/var/lib/mysql-files/user1.txt";

表中前三列到出来

mysql>select id,name,password from user into outfile "/var/lib/mysql-files/user2.txt";

表中前三行导出来

mysql>select id,name,password from user where

id <= 3 into outfile "/var/lib/mysql-files/user3.txt";

mysql>select id,name,password from user where id <= 3 into outfile "/var/lib/mysql-files/user4.txt";

加字段隔符

fields terminated by 换行符号

lines terminated by 字段间隔符号

mysql>select id,name,password from user where id <= 3 into outfile "/var/lib/mysql-files/user5.txt" fields terminated by ";" lines terminated by "!!!";

管理表记录（增删改查）(不加条件就是批量删改查)

增

insert into 库.表 values(字段值列表)；

删

delete from 库.表 where 条件;

改

update 库.表 set 字段名=值，字段名="值" where 条件;

查

select 字段名列表 from 库.表 where 条件;

给字段赋值:

mysql>insert into db1.user values(42,"bob","x",2000,2000,"student user","/home/bob","/bin/bash"),(43,"jeson","x",2100,2100,"student user","/home/bob","/bin/bash")；

只给 name 和 uid 两个字段字段赋值：

mysql>insert into db1.user(name,uid) values("lucy",3000)，("jack",3010)，("jerry",3020);

条件查询：

格式：

select 列名 from 表名 where id=3;

查看表中第三行的这些列:

select id,name,shell,homedir from user where id=3;

查看表中的列:

select id,name,shell,homedir from user

批量修改 password 密码为 A

update user set password="A";

修改符合条件的字段的值

uodate user set password="x" where id=1;

匹配条件

可以跟条件的: select delete update

数值比较：符号：= ,>,>= ,<,<=,=

where 字段名 符号 值

where 字段名　　符号　　字段名

例：在 user 表里找行号为 3 的值

select * from user where id=3;

找 uid=gid

select name,uid,gid,from user where uid=gid;

找 uid<10

select name,uid from user where uid<10;

修改

update user set password="F",homedir="/student" where uid<10;

字符比较　=　　！=

where 字段名　字符　"值"

select name from user where shell="/bin/bash";

select name from user where shell!="/bin/bash";

select name from user where name="mysql";

select * from user where name="mysql";

匹配查找空 is null

匹配查找非空 is not null

uid 字段是空的值

select name from user where uid is null;

uid 字段是不是空的值

select name,uid from user where uid is not null;

修改 uid 值，当 uid 为空

update user set uid=4008 where uid is null;

修改 gid 字段为空值

select * from user where id=3;

update user set gid=null where id =3;

逻辑比较（有多个条件的时候）

逻辑与　and　多个条件必须同时成立

逻辑或　or　多个条件某一个条件成立即可

逻辑非　!，not　取反

select name,uid from user where name="root" and uid=0 and shell="/bin/bash";

select name,uid from user where name="root" or uid=7 and shell="/sbin/nologin";

优先级：()

select name,uid from user where name="root" and (uid=1 or uid=3);

范围内匹配

select name from user where name in ("apache","mysql","adm","bin");

select name from user where uid in (10,20,30,40);

select name from user where shell not in ("/bin/bash","/sbin/nologin");

uid 在 10 和 20 之间

select name,uid from user where uid between 10 and 20;

distinct 不显示字段的重复值

格式: select distinct 字段名 from user；

select distinct shell from user;

select distinct shell from user where uid <= 10;

模糊查询 like

匹配三个字符(三个下划线，不能空格)

select name from user where name like '___';

以 a 开头 t 结尾，中间 0 个或多个

select name from user where name like 'a%t';

名字不能少于 4 个字符

select name from user where name like '__%__';

正则匹配

where 字段名 regexp '正则表达式';

^ 匹配行首

$ 匹配行尾

. 匹配任意单个字符

*

[]

匹配任意四个字符

select name,uid from user where uid regexp '....';

匹配必须是四个字符

select name,uid from user where uid regexp '^....$';

以 t 结尾的

select name from user where name regexp 't$'

查询带 0-9 的名字:

mysql> insert into user(name) values("haha1"),("haha5"),("hah3a"),("ha6ha");

mysql> select name from user where name regexp '[0-9]';

四则计算 +(加) -(减) *(乘) /(除) %(取余)

alter table user add age tinyint(2) unsigned default

18 after name;

select name,age,2018 - age as syear from user where name="root";

age 字段批量加 1

update user set age=age+1;

显示每个字段值算平均值(字段类型是数值类型才行)

select name,uid,gid,(uid+gid)/2 pjz from user where uid>=10 and uid<=30;

聚集函数(后面可加条件)

select avg(uid) from user; 输出字段值的平均值

select sum(uid) from user; 求和

select min(uid) from user; 输出字段值的最小值

select max(uid) from user; 输出字段值的最大值

select count(id) from user; 统计字段值的个数

select name from user where shell="/bin/bash";

select count(name) from user where shell="/bin/bash";

select count(*) from user;

select count(id),count(name) from user;

select shell from user where uid>=10 and uid<=

前十行的用户名显示出来

select name from user where id>=10;

查询结果的行数 limit 数字

sql 查询 limit 起始行,行数；

显示结果前 3 行

select name from user where uid>=10 and uid<=1000 limit 3;

从第四行开始显示，一共显示 3 行

select * from user limit 3,3;

从第 10 行开始显示，一共显示 5 行

select * from user limit 9,5;

倒序排列

order by 字段名 (desc？)

大到小加上 desc，小到大不加 desc

uid 号从大到小

select id,name,shell from user where uid <=10 order by uid desc;

uid 号从小到大排：

select * from user where uid<=10 order by uid;

分组 group by 字段名;

select shell from user where uid>=10 and uid<=1000 group by shell;

过滤数据 having 条件;

select name from user;

select name from user having name="jerry";

select id,name from user where name like '%' having name is null;

select shell from user where uid>=10 and uid<=1000 group by shell having shell="sync";

mysql 存储引擎

mysql>show engines; 显示当前数据库服务器支持的存储引擎

修改/etc/my.cnf 配置文件

[mysqld]

.....

default-storage-engine=myisam

重启服务

常用的存储引擎

myisam

特点：

1 个表用 3 个文件存储

t1.frm　t1,MYD　t1.MYI

表结构　　数据　　索引

不支持外键和事务和事务回滚

支持表级锁（给一张表加锁）

执行查询操作多的表，适合使用 myisam 存储引擎，这样可以节省系统资源

事务：一次 sql 操作从开始到结束的过程

事务回滚：在事务执行过程中任意一步操作失败，恢复所有的操作

innodb

特点：

1 个表使用 2 个文件存储

t2.frm　t2.ibd

表结构　　数据+索引

支持外键和事务和事务回滚

支持行级锁（只给被访问的行加锁）

执行写操作多的表，适合使用 innodb 存储引擎，这样可以加大并发访问量

事务：一次 sql 操作从开始到结束的过程

事务回滚：在事务执行过程中任意一步操作失败，恢复所有的操作

典型的事务：银行的转账或汇款

加锁的目的： 解决并发访问冲突问题的

锁粒度：行锁 表锁 页级锁

锁类型：

select 读锁（共享锁）

insert、update、delete 写锁（排它锁 或互斥锁）

执行查询造作多的表，适合使用 myisam 存储引擎，这样可以节省系统资源

innodb t1 t2 ...

myisam a1 a2 a3 ...

事务日志

ibdata1 未提交的 sql 命令

LSN 日志序列号

ib_logfile0 以及提交的 sql27/*!*/;

insert into db55.t1 values(399)

命令

ib_logfile1

事务特性：

原子性：

一致性：事务操作的前后，表中的记录没有变化

隔离性：事务操作是相互隔离不 27/*!*/;

insert into db55.t1 values(399)

受影响的
持久性：数据一旦提交，不可改变，永久改变表数据

查看自动提交功能：
show variables like "%auto%";
关闭自动提交功能：
mysql>set autocommit ss= off;
mysql>rollback; 数据回滚
mysql>commit; 提交永久保存

DBA4 day04
多表查询
安装图形管理工具
用户授权及权限撤销

多表查询
复制表：功能 1：备份表
功能 2：快速键表
语法格式：
mysql>create table 库.表 sql 查询;

理解：67 87
教学
成绩表 学号 姓名 课程名 成绩
信息表 学号 姓名 年龄 地址

select 姓名,成绩 from 成绩表 where 姓名 in (select 姓名 from 信息表 where 地址="北京");

将 db1 下的 user 表备份到 db4 库中的 t1 表里

mysql>create table db4.t1 select * from db1.user;

将 db1 下的 user 表中的 id,name 字段的前 10 行备份到 db4 库中的 t2 表里

mysql>create table db4.t2 select id,name from db1.user limit 10;

将 db1 库中的 user 表结构复制到 db4 中的 t3 表里

create table db4.t3 select * from db1.user where 1 =2;

多表查询:

单表查询格式:(进库中查询)

select 字段名列表 from 表 where 条件;

where 嵌套查询:把内层的查询结果作为外层查询的查询条件

语法格式：内层查询的表和外层查询的表可以是同一个库下的相同表，也可以是不同一个库下的不同表

select 字段名列表 from 表名 27/*!*/;

insert into db55.t1 values(399)

where 条件 (select 字段名列表 from 表名 where 条件);

查询 db1 库中 user 表里的 name,uid 字段列表

select name,uid from db1.user

查询 db1 库中 user 表里的 uid 平均值

select avg(uid) from db1.user;

把 user 表中的 uid 号小于平均值的用户列出来

select name,uid from db1.user where uid < (select avg(uid) from db1.user);

desc mysql.user;

select host,user from mysql.user;

select host,user from mysql.user where host="localhost";

查询名字是本地的 db4.t1 里面没有 host 字段名列表

select name from db4.t1 where name in (select user from mysql.user where host="localhost");

连接查询：交叉连接　自然连接　内连接 外连接

多表查询

格式 1：select 字段名列表 from 表 a,表 b;

迪　　卡　　尔　　集

10　　3

10 * 3 = 30

select * from t5,t6 where t5.uid = t6.uid;

select t5.name,t6.name from t5,t6 where t5.uid = t6.uid;

左连接：当条件成立时，以左表为主显示的查询结果

语法格式：

slect 字段名列表 from 表名 1 left join 表名 2 on 条件

右连接：当条件成立时，以右表为主显示的查询结果

语法格式：

slect 字段名列表 from 表名 1 right join 表名 2 on 条件

创建 t7 表

create table t7 select name,uid,shell from db1.user limit 5;

创建 t8 表

create table t8 select name,uid,shell from db1.user limit 8;

将 t7 表与 t8 表相同的显示出来(左连接)

mysql>select * from t7 left join t8 on t7.uid =

t8.uid;

mysql>select t7.* from t7 left join t8 on t7.uid = t8.uid;

将 t7 表与 t8 表不同的显示出来(右连接)

mysql>select * from t7 right join t8 on t7.uid = t8.uid;

在数据库上安装图形管理工具

phpMyAdmin-2.11.11-all-languages.tar.gz

部署软件运行环境　httpd　hph

装包

yum -y install httpd php php-mysql

起服务

systemctl start httpd

systemctl enable httpd

写网页

vim /var/www/html/abc.php

<?php

phpinfo();

?>

测试

firefox http://ip 地址/abc.php

解压

tar -xf phpMyAdmin-2.11.11-all-languages.tar.gz -C /var/www/html

cd /var/www/html

mv phpMyAdmin-2.11.11-all-languages/ phpMyAdmin

递归修改目录的所有者，所属组

chown -R apache:apache phpMyAdmin/

拷贝主配置文件,改名字并修改配置文件

cd phpMyAdmin/

cp config.sample.inc.php config.inc.php

vim config.inc.php

17 行　$cfg['blowfish_secret'] = '';　在单引号后面随便加内容比如：lx123

31 行　$cfg['Servers'][$i]['host'] = 'localhost';　不需要改默认就是 localhost

真机访问测试:

firefox http://192.168.4.50/phpMyAdmin

登入名称：root

密码：　123456

点击执行

用户授权及权限撤销

用户授权：在数据库服务器上添加可以连接的用户，并且可以设置添加用户的访问权限

默认只有数据库管理员在本机能够访问数据库服务器

默认数据库管理源在本机登录有授权权限。

管理员 root 用户密码管理：

重置本机登录密码

mysqladmin -hlocalhost -uroot -p password "新密码"

Enter password: 输入当前登录密码

恢复本机登录密码

当前登录密码忘记了，需要修改配置文件来恢复密码：

1，停止 mysql 服务

systemctl stop mysqld

2，改配置

vim /etc/my.cnf

[mysql]

skip-grant-tables

#validate_password_policy=0

#validate_password_length=6

3,起服务

systemctl startDBA4 day04 mysqld

4,直接登录

mysql

5,看表结构,找到 authentication_string

desc mysql.user

6,修改密码

update mysql.user set authentication_string=password("新密码") where user="root" and host="localhost";

6,刷新

flush privileges;

7,退出

quit;

8，停止 mysql 服务

systemctl stop mysqld

9，改配置

vim /etc/my.cnf

[mysql]

#skip-grant-tables

validate_password_policy=0

validate_password_length=6

10,起服务

systemctl restart mysqld

11,用新密码登录

mysql -uroot -p 新密码

用户授权命令

mysql>grant 权限列表 on 数据库名 to 用户名@"客户端地址" identified by "密码";

指定用户有授权权限格式：

mysql>grant 权限列表 on 数据库名 to 用户名@"客户端地址" identified by "密码" with grant option;

在客户端上验证用户权限

which mysql 没有就用 yum 装 mariadb

mysql -hip 地址 -u 用户名 -p 密码

mysql>select @@hostname;　显示当前连接机器的名称

mysql>show grants;　查看访问权限

mysql>select user();　查看哪个用户连的

改授权表中的记录了，添加 insert 权限

mysql>update mysql.table_priv set Table_priv=""select,insert" where user="用户名";

mysql>flush privileges;　刷新

在 50 服务器上添加连接用户 admin

grant all on *.* to admin@"192.168.4.51" identified by "123456" with grant option;

mysql -h192.168.4.50 -uadmin -p123456　50 服务器上登录

mysql>select @@hostname;　显示当前连接数据库服务器的名称

mysql>show grants;　查看访问权限

mysql>select user();　查看哪个用户连的

mysql> select user,host from user;　查看哪些用户

mysql>show grants for 用户名@"客户机地址";有 root 权限查看某个用户访问权限

mysql> select * from 库名.表名 where user="用户名" \G;　查看用户的授权权限(只知道用户名)

mysql> select user from mysql.user;　查看授权用户

grant all on *.* 数据库中全部授权

grant select,insert,update,delete on db1.* 给 db1 库中的所有内容授权增删改查

grant select,update(name,id) on db1.user 给 db1 中的 user 表中的 name 和 id 字段名列表授权查和改

用户名@"localhost" 本地主机

用户名@"ip 地址" DBA4 day04 某一个 ip 地址

用户名@"192.168.4.%" 网段

用户名@"%" 所有网段主机

给指定网段授权

mysql>grant all on webdb.* to webuser@"192.168.4.%" identified by "123456";

给所有主机添加授权

mysql>grant select,update(name,uid) on db1.user to yaya@"%" identified by "123456";

授权信息存储在数据库服务器的授权库 mysql 库里，使用不同的表存储授权信息 columns_priv db tables_priv user

use mysql;

show tables;

columns_priv 表　　存储授权用户对字段的访问权限

db 表　　存储授权用户对数据库的访问权限

tables_priv 表　　存储授权用户对表的访问权限

user 表　　存储授权用户的访问权限

mysql> select user from mysql.user;

mysql> select * from mysql.user where user="admin" \G;

select * from mysql.db where

select host,user,db from mysql.db;

select * from mysql.user where user="webuser" \G;

改授权表中的记录了，添加 insert 权限

mysql>update mysql.table_priv set Table_priv=""select,insert" where user="yaya";

mysql>flush privileges;　　刷新

show grants for yaya@"%";

权限撤销

撤销的时用户的权限

对数据库有过授权才可以撤销权限

格式:

mysql>revoke 权限列表 on 库名.表名 from 用户名@"客户端地址"

例:

单独撤授权用户的授权权限

revoke GRANT OPTION on *.* from admin@"192.168.4.51";

查看

show grants for admin@"192.168.4.51";

撤销 delete 权限

revoke delete on *.* from admin@"192.168.4.51";

查看

show grants for admin@"192.168.4.51";

删除授权用户

drop user admin@"192.168.4.51"

管理员 root 修改授权用户的连接密码

mysql>set password 用户名@"客户端地址"=password("新密码");

授权用户登录后修改登录密码：

mysql>set password=password("新密码")

DBA5 day05

数据库的备份和恢复

完全备份/恢复

备份　mysqldump 命令

恢复　mysql 命令

数据备份的目的：

误操作（管理员，使用者）

硬件损坏（磁盘）

数据备份方式

物理备份：直接拷贝库和表对应的系统文件

备份整个库：cp -rp /var/lib/mysql /opt/mysqldir.bak

备份单个库：cp -rp /var/lib/mysql/mysql /opt/mysql.bak

备份单个表：cp -rp /var/lib/mysql/bbsdb/t1.* /opt

tar 备份：tar -zcvf /opt/mysql.tar.gz /var/lib/mysql/*

删除: rm -rf /var/lib/mysql/mysql

恢复到数据库里：cp -r /opt/mysql.bak /var/lib/mysql/mysql

递归改归属：chown -R mysql:mysql /var/lib/mysql/mysql

重启数据库: systemctl restart mysql

逻辑备份(生产环境中的)：执行备份操作时，根据备份的库，表产生对应的 sql 命令，把 sql 命令存储到指定的文件里.

数据备份策略

完全备份：

备份所有数据 ： 一台服务器所有的数据

一个库的所有数据

一张表的所有数据

只备份新产生的数据：差异备份：备份完全备份后所有新产生的数据

* 增量备份：备份上次备份后所有新产生的数据

生产环境下对数据做备份的方法

1、写个周期性计划任务（crond）

执行备份脚本（shell、python、go、ruby、perl）

2、搭建 mysql 主从结构实现数据的自动备份

使用脚本对数据做备份要考虑的问题

1、备份频率 大概多长时间备份一次(以周为单位)

2、备份策略 完全+增量 or 完全+差异

3、备份时间 一般凌晨备份(数据库服务访问量少的时候)

4、备份文件名 日期_xx.sql （以.sql 结尾）

5、存储空间 lv+raid

t1 表　　数据　　23：00　　文件名　　备份量

周一　　完全备份　　10 条　　1.sql　　10 条

周二　　差异备份　　2 条　　2.sql　　2 条

周三　　差异备份　　5　　3.sql　　7

周四　　差异　　3　　4.sql　　10

周五　　差异　　1　　5.sql　　11

周六　　差异　　4　　6.sql　　15

周七　　差异　　9　　7.sql　　24

t1 表　　数据　　23：00　　文件名　　备份量

周一　　完全备份　　10 条　　1.sql　　10 条

周二　　增量备份　　2 条　　2.sql　　2 条

周三　　增量　　5　　3.sql　　5

周四　　增量　　3　　4.sql　　3

周五　　增量　　1　　5.sql　　1

周六 增量 4
6.sql 4
周七 增量 9
7.sql 9

完全备份/恢复

备份 mysqldump 命令

格式：(密码可以不输，回车后在输入密码)

mysqldump -u 用户名 -p 密码 源库名 > 路径/xxx.sql

源库名的表示

--all-databases 或 -A 所有库

数据库名 单个库

数据库名 表名 单张表

-B 数据库 1 数据库 2 备份多个库

注意事项:无论备份还是恢复,都要验证用户及权限

例：

创建一个存放备份的文件夹

mkdir /mybakdata

备份单个库

mysqldump -uroot -p123456 db1 > /mybakdata/db1.sql

备份单张表

mysqldump -uroot -p123456 db1 user > /mybakdata/db1 user.sql

备份所有库

mysqldump -uroot -p123456 -A > /mybakdata/all.sql

备份多个库

mysqldump -uroot -p123456 -B db1 db2 > /mybakdata/twodb.sql

恢复 mysql 命令 完全恢复数据

格式：(密码可以不输，回车后在输入密码)

mysql -u 用户名 -p 密码 库名 < 路径 /xxx.sql

例：

恢复单个库(先在数据里创建一个 db1 的库,然后进行数据恢复)

mysql -uroot -p123465 db1 < /mybakdata/db1.sql

恢复单张表(先在数据里创建一个 db1 的库和 user 表，然后进行数据恢复)

mysql -uroot -p123465 db1 user < /mybakdata/db1_user.sql

恢复所有库

mysql -uroot -p123465 < /mybakdata/all.sql

恢复多个库

mysql -uroot -p123465 < /mybakdata/twodb.sql

另外一种数据恢复方式

mysql -uroot -p123456

mysql>create databases userdb;

mysql>quit;

mysql -uroot -p123456 userdb <

/mybakdata/db1_user.sql

mysql -uroot -p123456

mysql>create database db5;

mysql>use db5;

mysql>source /mybakdata/db1_user.sql;

例:

每周一半夜 23 点把 db1 库里的所有数据全部备份到系统的/mybakdb1 目录中。

vim /shell/allbakdb1.sh

```
#!/bin/bash
if [ ! -e /mydatadb1 ];then
    mkdir /mydatadb1
fi
day=`date +%F`
mysqldump -uroot -p123456 db1 > /mydatadb1/${day}-db1.sql
```

chmod +x /shell/allbakdb1.sh

/shell/allbakdb1.sh &> /dev/null

cat /mydatadb1/

crontab -e

00 23 * * 1 /shell/allbakdb1.sh

缺点:

备份和恢复时都会锁表

增量备份/恢复

一、启用 mysql 服务的 binlog 日志(二进制日志)，实现增量备份/恢复

记录除查询之外的 sql 命令

select、desc、show

查看日志当前记录格式

mysql> show variables like "binlog_format";

binlog 日志三种记录格式 ：

statement :每一条修改数据的 sql 命令都会记录在 binlog 日志中。

row: 不记录 sql 语句上下文相关信息,仅保存哪条记录被修改。

mixed: 是以上两种格式的混合使用。

修改日志记录格式

临时修改

mysql> set binlog_format="statement/row/mixed";

修改配置文件

vim /etc/my.cnf

[mysqld]

....

server_id=50(主机名)

log-bin
binlog_format="mixed/statement/row"
重启服务
systemctl restart mysqld
查看日志:
mysqlbinlog /var/lib/mysql/db50-bin.000001

自定义 binlog 日志文件存储目录和名字
mkdir /logdir
chown mysql /logdir
修改配置文件
vim /etc/my.cnf
[mysqld]
....
server_id=50(主机名)
#log-bin
log-bin=/logdir/lx（名字随便起）
systemctl restart mysqld

binlog 日志文件记录命令方式
偏移量　　在 mysqlbinlog /logdir/lx.000001 里面看
开头（start）
at 1280　　#写上开头偏移量 1280
结尾（stop）

/*!*/;

at 1385　　#写上结尾偏移量 1385

时间点　　在 mysqlbinlog /logdir/lx.000001 里面看

开头（start）

at 622

#180622 15:40:27　　# 写 上 时 间 2018-06-22 15:40:27

结尾（stop）

COMMIT

/*!*/;

at 807

#180622 15:40:34 server id 50　end_log_pos 872 CRC32 0x264883c9　　Anonymous_GTID　　last_committed=3　　sequence_number=4

SET @@SESSION.GTID_NEXT= 'ANONYMOUS'/*!*/;

at 872

#180622 15:40:34　　# 写 上 时 间 2018-06-22 15:40:34

执行 binlog 日志文件里的 sql 命令恢复数据

格式：

mysqlbinlog 选项 日志文件名 | mysql -u 用户名 -p 密码

选项：　--start-position=数值　　--stop-position=数值

　　--start-datetime="yyyy-mm-dd　hh:mm:ss" --stop-　datetime="yyyy-mm-dd hh:mm:ss"

例:

mysqlbinlog --start-position=1280 --stop-position=1385 /logdir/lx.000001 | mysql -uroot -p123456

这样可以把 insert 命令在执行一次，查询数据表会增加一个值

例：

mysqlbinlog --start-datetime="2018-06-22 15:40:27" --stop-datetime="2018-06-22 15:40:34" /logdir/lx.000001 | mysql -uroot -p123456

手动生成新的 binlog 日志文件的方法（默认 500M 会生成新的）

1、在登录状态下执行刷新命令

mysql>flush logs;

2、在命令行下

mysql -uroot -p123456 -e "flush logs"

3、重启服务(生产中不能用)

systemctl restart mysqld

4、在命令行下

mysqldump -uroot -p123456 --flush-logs 库名 > 路径/文件名.sql

查看状态:

mysql>show master status;

删除已有的 binlog 日志

1、mysql>purge master logs to "日志名"

例：

把指定编号 5 之前的日志删了

mysql>purge master logs to "lx.000005";

2、删除当前所有日志文件重新生成第一个日志文件

mysql>reset master;

3、系统命令删（不建议这样删）

rm -rf 日志名

二、安装第三方软件，提供 innobackupex 命令作增量备份/恢复

XtraBackup 工具

备份过程中不锁库表,适合生产环境

主要含两个组件

xtrabackup : C 程序,支持 InnoDB/XtraDB

innobackupex :以 Perl 脚本封装 xtrabacku
p ,还支持 MyISAM

安装 xtrabackup

1、安装提供命令的软件包

yum -y install perl-Digest-MD5 perl-DBD-mysql

rpm -ivh libev-4.15-1.el6.rf.x86_64.rpm

rpm -ivh percona-xtrabackup-24-2.4.7-1.el7.x86_64.rpm

rpm -ql percona-xtrabackup-24

innobackupex<选项>

--host 主机名

--user 用户名

--port 端口号

--password 密 码

--databases 数据库名

--no-timestamp 不用日期命名备份文件存储的子目录名

--redo-only 日志合并

--apply-log 准备还原 (回滚日志)

--copy-back 恢复数据

--incremental 目录名 增量备份

--incremental-basedir=目录名 增量备份时,指定上一次备份数据存储的目录名

--incremental-dir= 目录名 准备恢复数据时,指定增量备份数据存储的目录名

--export 导出表信息

import 导入表空间

完全备份:(存过备份数据的目录不能在用了,需要备份的话需要新的目录)

备份所有库 (/allbak 这个目录可以不用事先创建)

innobackupex --user root --password 123456 /allbak --no-timestamp

显示 180622 17:54:45 completed OK! 这个则备份成功了

ls /allbak

注意： 备份库或表的时候需要备份 mysql sys performancr_schema 这三个系统库，恢复的时候有用

备份指定库的所有数据

格式:

innobackupex --user root --password 123456 --databases="mysql sys performance schema 库名" /buydballbak --no-timestamp

例

innobackupex --user root --password 123456 --databases="mysql sys performance schema buydb" /buydballbak --no-timestamp

备份多个库的所有数据

格式:

innobackupex --user root --password 123456 --databases="mysql sys performance schema 库名 1 库名 2" /buydballbak --no-timestamp

例

innobackupex --user root --password 123456 --databases="mysql sys performance schema buydb mysql" /towdballbak --no-timestamp

备份指定表的所有数据

格式：

innobackupex --user root --password 123456 --databases="mysql sys performance schema 库名.表名" /allbaktab --no-timestamp

例

innobackupex --user root --password 123456 --databases="mysql sys performance schema buydb.a" /allbaktab --no-timestamp

备份不同库中的不同表的所有数据

innobackupex --user root --password 123456 --databases="mysql sys performance schema 库名 1.表名 1 库名 2.表名 2" /haha --no-timestamp

例

innobackupex --user root --password 123456 --databases="mysql sys performancebinlog 日志文件 schema buydb.a paydb.b" /haha --no-timestamp

实验:

备份数据库：（起名为/buydbfull）

innobackupex --user root --password 123456 --databases="mysql sys performance schema buydb" /buydb_full --no-timestamp

完全恢复

数据恢复：数据库中的目录是空的

rm -rf /var/lib/mysql/*

步骤：

1、重做日志　　--apply-log

innobackupex --user root --password 123456 --apply-log /buydb_full

清空数据库目录：

rm -rf /var/lib/mysql/*

停止数据库服务

systemctl stop mysqld

2、恢复数据　　--copy-back

innobackupex --user root --password 123456 --copy-back /buydb_full

更改文件归属

chown -R mysql:mysql /var/lib/mysql

3、重启数据库服务

systemctl start mysqld

进入数据库

mysql -uroot -p123456

查看有哪些库

mysql>show databases;

mysql>select * from buydb.a;

innobackupex 增量备份、恢复

增量备份

1、做首次备份（完全备份）

innobackupex --user root --password 123456 --databases="mysql sys performance schema buydb" /fullbuydb --no-timestamp

查看一下

ls /fullbuydb

写一些新数据

insert into buydb.a values(201),(202),(203);

--incremental 目录名 增量备份

--incremental-basedir=目录名 增量备份时,指定上一次备份数据存储的目录名

innobackupex --user root --password 123456 --databases="mysql sys performance schema buydb" --incremental /new1dir --incremental-basedir=/fullbuydb --no-timestamp

查看一下

ls /new1dir

写新数据，做增量备份

mysql>insert into buydb.a values(301),(302),(303);

innobackupex --user root --password 123456

--databases="mysql sys performance schema buydb" --incremental /new2dir --incremental-basedir=/new1dir --no-timestamp

查看一下

ls /new2dir

innodb 和 xtrdb

cd /var/lib/mysql

事务日志文件

ib_logfile1

ib_logfile0

ibtmp1 记录未提交的

LSN 日志序列号

cat /fullbuydb/xtrabackup_checkpoints

增量恢复（数据库目录为空）

所需选项

--incremental-dir=增量备份目录名

--apply-log 准备还原

--redo-only 日志合并

重做并合并日志

innobackupex --user root --password 123456 --apply-log --redo-only /fullbuydb

查看一下

cat /fullbuydb/xtrabackup_checkpoints

innobackupex --user root --password 123456 --apply-log --redo-only /fullbuydb --incremental-dir=/new1dir

查看一下

cat /new1dir/xtrabackup_checkpoints

innobackupex --user root --password 123456 --apply-log --redo-only /fullbuydb --incremental-dir=/new2dir

查看一下

cat /new2dir/xtrabackup_checkpoints

清空数据库目录

rm -rf /var/lib/mysql/*

停止服务

systemctl stop mysqld

恢复数据

--user root --password 123456 可以不写

innobackupex --user root --password 123456 --copy-back /fullbuydb

如果报错，再次看下/var/lib/mysql/这个文件夹是否为空

递归改归属

chown -R mysql:mysql /var/lib/mysql

启动服务

systemctl start mysqld

连接查看数据

mysql -uroot -p123456

mysql>show databases;

在生产环境下使用 innobackupex 备份数据

计划任务 crontab -e

00 23 * * 1 /shell/allbak.sh ----> 完全备份 innobackupex

00 23 * * 1-7 /shell/newdata.sh --> 增量备份 innobackupex

在 innobackupex 完全备份数据中，恢复某个表的数据

用到的选项：

--export 导出表信息

删除表空间

mysql> alter table 库名.表名 discard tablespace;

导入表空间

mysql> alter table 库名.表名 import tablespace;

ls /buydbfull2/buydb/

a.frm a.ibd db.opt

其中的 a.ibd 是表空间文件

首先要在 vim /etc/my.cnf 配置文件中

default-storage-engine=innodb

然后重启服务

systemctl restart mysqld

完全备份

innobackupex --user root --password 123456 --databases="buydb" /buydbfull2 --no-timestamp

mysql> drop table b;

恢复步骤：

从备份数据里导出表信息

innobackupex --user root --password 123456 --datebases="buydb" --apply-log --export /buydbfull2

创建删除的表（表结构要和删除的时候相同）

mysql> create table b(id int(2));

删除表空间文件

mysql> alter table buydb.b discard tablespace;

把导出的表信息文件拷贝到数据库下

cp /buydbfull2/buydb/b.{cfg,exp,ibd} /var/lib/mysql/buydb

cd /var/lib/mysql/buydb

ls

chown mysql:mysql b.*

导入表空间

mysql> alter table buydb.b import tablespace;

查看数据

select * from buydb.b;

导完了就可以删文件了

rm -rf /var/lib/mysql/buydb/b.{cfg,exp}

备份方式： 物理 逻辑

备份策略： 完全 差异 增量

##

DBA 进阶课程

day1

mysql 主从同步

mysql 主从同步模式

mysql 主从同步介绍：

角色分为 2 种：

数据库服务　做主 master 库：被客户端存储数据访问的库

数据库服务　做从 slave 库：同步主库的数据到本机

mysql 主从同步作用：

实现数据的自动备份

配置 mysql 主从同步结构

客户端 192.168.4.254

##

主从同步

重要实验：

准备 2 台数据库服务器　192.168.4.51(master) 、192.168.4.52(slave)

主库配置步骤：192.168.4.51

1、启动 binlog 日志

vim /etc/my.cnf

[mysqld]

server_id=51

log-bin=master51

binlog-format="mixed"

2、重启数据库服务并查看文件

systemctl restart mysqld

ls /var/lib/mysql/master51.*

3、用户授权

mysql -uroot -p123456

mysql> grant replication slave on *.* to repluser@"192.168.4.52" identified by "123qqq...A";

4、查看正在使用的 binlog 日志信息

mysql> show master status;

从库配置步骤：192.168.4.52

1、验证主库授权用户

mysql -h192.168.4.51 -urepluser -p123qqq...A

mysql> quit;

2、指定 server_id

重要:指定的 server id 一定不能和其他指定的 server id 相同

vim /etc/my.cnf

[mysqld]

server_id=52

重启服务

systemctl restart mysqld

3、指定主库信息

mysql -uroot -p123456

mysql> show slave status\G;　　没指定之前查的时候是空的

master51.000001（在 192.168.4.51(主库)上查的 mysql> show master status;)

452　(在 192.168.4.51(主库)上查的　mysql> show master status;)

第一次指定就用 change，之后就是修改

mysql>change master to master_host="192.168.4.51",主库 ip

master user="repluser",　　主库授权的用户名

master password="123qqq...A",　　主库授权的用户名密码

master_log_file="master51.000001", binlog日志
master_log_pos=452; 偏移量

4、查看从库状态信息

mysql>start slave; 起一下 slave 进程

mysql> show slave status\G; 每一行占一列显示

找到这两行同时是 YES 状态就可以了

Slave_IO_Running: Yes

Slave_SQL_Running: Yes

主从同步的工作原理：

cd /var/lib/mysql

ls （有这四个文件就可以了）

master.info

relay-log.info

db52-relay-bin.xxxxxx (xxxxxx:会以 000001 的方式显示)

db52-relay-bin.index

IO 线程：

Last_IO_Error:

SQL 线程：

Last_SQL_Error:

弄错了，要重新弄，删文件(从库中操作)

cd /var/lib/mysql

rm -rf master.info

rm -rf relay-log.info

rm -rf db52-relay-bin.*

rm -rf master53.*

然后从第 3 步(从库)指定主库信息开始做

mysql>show processlist;　　查看列出程序列表(主库中查看)

客户端（真机）验证主从同步配置

1、在主库上添加访问数据时连接的用户

mysql>create database bbsdb;

mysql>grant all on bbsdb.* to jim@"%" identified by "123qqq...A";

2、客户端（真机）访问

mysql -h 主库 IP -u 主库上授权的用户名　-p 主库上授权的用户密码

MySQL [(none)]>show databases;

建表插入记录

MySQL [(none)]> create table bbsdb.t1(id int(3));

MySQL [(none)]> insert into bbsdb.t1 values(101),(202),(303),(404);

查看是否有数据

MySQL [(none)]> select * from bbsdb.t1;

在在主库(192.168.4.51)和从库(192.168.4.52)中查看是否是同样的数据

用 root 用户登录

mysql> select * from bbsdb.t1;　　主库上操作

mysql> select * from bbsdb.t1;　　从库上操作

create database

##

让当前从库临时不同步主库上的数据

mysql>stop slave;

在从库中修改主库中的信息：

先暂停

mysql>stop slave;

哪一项错了就改哪一项(不知道是 to 还是 tom)

mysql>change master tom 选项="值",选项="值";

改完后要让他生效

mysql>start slave;

主从配置常用参数：

主库　/etc/my.cnf

[mysql]

二选一:

binlog_do_db=db2,bbsdb　　只允许同步的库

binlog_ignore_db=db9　　只不 create database 允许同步的库

从库　/etc/my.cnf(重点记住)

[mysql]

log_slave_updates　　级联复制(在配置文件中加)

二选一:

replicate_do_db=db3,db5　只同步的库（db3,db5：库名）

replicate_ignore_db=db1,db7,db9 只不允许同步的库（db：库名)

主从同步模式

1、主从同步结构模式

一主一从

一主多从　把 50 主机配置为 51 的从库

从配置步骤：192.168.4.50

1、验证主库授权用户

2、指定 server_id

3、指定主库信息

4、查看从库状态信息

2、主从从 把主机 53 配置为 52 从库

配置主库 192.168.4.52

启用 binlog 日志

用户授权

配置从库：192.168.4.53

1、验证主库授权用户

2、指定 server_id

3、指定主库信息

4、查看从库状态信息 create database

注意：从库的 sql 线程执行本机中继日志文件里的 sql 命令，不会记录在本机的 binlog 日志文件里

3、主主结构（互为主从）课后练习

mysql 主从同步复制模式

异步复制

完全复制

半同步复制

查看是否支持动态加载模块

mysql>show variables like "have_dynamic_loading";

主库安装的模块：

mysql>INSTALL PLUGIN rpl semi sync master SONAME'semisync_master.so';

查看系统库下的表，模块是否安装成功

mysql>

SELECT PLUGIN_NAME , PLUGIN_STATUS

FROM INFORMATION_SCHEMA.PLUGINS

WHERE

PLUGIN_NAME LIKE '%semi%';

从库安装的模块

mysql>INSTALL PLUGIN rpl semi sync slave SONAME'semisync_slave.so';

然后从第 3 步(从库)指定主库信息开始做

启用半同步复制

启用半同步复制模式

主库

mysql>SET GLOBAL

rpl_semi_sync_master_enabled = 1;

从库

mysql>SET GLOBAL rpl semi create databasesync_slave_enabled = 1;

查看本机的半同步复制状态

mysql>show variables like "rpl_semi_sync_%_enabled";

修改配置文件/etc/my.cnf 让安装模块和启用的模式永久生效。

主库

vim /etc/my.cnf 然后从第 3 步(从库)指定主库信息开始做

[mysqld]

plugin-load=rpl semi sync master=semisync master.so

rpl_semi_sync_master_enabled=1

:wq

从库

vim /etc/my.cnf

[mysqld]

plugin-load=rpl semi sync slave=semisync slave .so

rpl_semi_sync_slave_enabled=1

既做主又做从

vim /etc/my.cnf

[mysqld]

plugin-load = "rpl_semi_sync_master=semisync_master.so;
rpl_semi_sync_slave=semisync_slave.so"

rpl-semi-sync-master-enabled = 1

rpl-semi-sync-slave-enab 公测 led = 1

++++++++++++++++++++++++++++create database++++++++++++++++++++++++++++

报错问题：（在从库中）

mysql> change master to master_host="192.168.4.52",master_user="repluser",
master_password="123qqq...A",
master_log_file="master51.000001",master_log_pos=1035;

ERROR 3021 (HY000): This operation cannot be performed with a running slave io thread; run STOP SLAVE IO_THREAD FOR CHANNEL '' first.

解决：

mysql> stop slave;

++

day2

mysql 性能优化 create database

mysql 数据读写分离

MySQL 性能调优

MySQL 优化：

数据库服务器处理客户端的连接请求慢，可能是由于那些原因导致的？

1、网络带宽，网络接口的流量

2、服务器硬件的配置，查看服务器硬件资源的使用情况 CPU 内存 存储

I\O 影响服务器存储速度

3、提供数据服务软件版本低

查看服务运行时的参数设置：/etc/my.cnf

mysql 撇值文件祥解

mysql> show variables; 查看数据库中的所有变量值

mysql> show variables like "%关键字%"; 模糊查询

例

mysql> show variables like "%time%";

create database

修改变量值

命令行修改（临时修改）

set [global] 变量名=值；

永久修改

vim /etc/my.cnf

[mysqld]

变量名=值

重用的参数有哪些？

1、查看并发连接数

mysql> show variables like "max_connections";

设置并发连接数：

mysql>set global max_connections = 500;

有过的最大连接数量/并发连接数=0.85　这个时正常的

Max_used_connections/max_connections=0.85

100/x=0.85　公式

0.85*100%=85%create database

mysql>flush status;　把当前的值清零

查看 thread

mysql> show variables like "%thread%";

thread_cache_size = 9 可以重复使用的线程的数量

缓冲区、线程数量、开表数量

mysql> show variables like "%关键字%"; 可用模糊查询

选 项	含 义
key buffer-size	用于 MyISAM 引擎的关键索引缓存大小
sort buffer size	为每个要排序的线程分配此大小的缓存空间
read buffer size	为顺序读取表记录保留的缓存大小
thread cache size	允许保存在缓存中被重用的线程数量
table open cache	为所有线程缓存的打开的表的数量

优化 SQL 查询

记录慢查询

选 项	含 义
slow-query-log	启用慢查询
slow-query-log-file	指定慢查询日志文件
long-query-time	超过指定秒数(默认 10 秒)的查询才被记录

create database

log-queries-not-using-indexes 记录未使用索引的查询

慢查询日志做统计的命令

mysqldumpslow

调整服务配置

vim /etc/my.cnf

[mysqld]

slow_query_log=1

slow_query_log_file=mysql-slow.log

long_query_time=5

log_queries_not_using_indexes=1

service mysql restart 启动 mysqld

create database

mysql> show variables like "%query_cache%";

query_cache_type = 0 | 1 | 2

selct sql_in_cache 字段名列表 from 表;

query_cache_wlock_invalidate | OFF

当对 myisam 存储引擎的表，查询的时候，若此时有客户端对表执行写操作，mysql 服务不会从缓存里查找数据返回给客户端，而是等写操作完成后，重新从表里查找数据返回给客户端

查询缓存统计信息

mysql> show global status like "qcache%";

Qcache_hits 0 在查询缓存里找到的

Qcache inserts 0 接收到的查询请求总数

Qcache lowmem prunes 0 查询缓存低的时候

Qcache_not_cached 1

create database

程序员编写的访问数据库服务数据的 sql 命令复杂导致处理速度慢。

在数据库服务器上启用慢查询日志，记录超过指定时间显示查询结果的 sql 命令。

binlog 日志

默认开启

错误日志

用 途 记录启动 / 运行 / 停止过程中的错误消息

配 置 log-error[=name]

查询日志 记录所有的 sql 命令

用 途 记录客户端连接和查询操作

启用日志： /etc/my.cnf

[mysqld]

general-log

重启数据库服务

systemctl restart mysqld

在这个目录下会有一个　主机名.log 结尾的文件

ls /var/lib/mysql

10.log

慢查询日志　只记录超时时间显示查询结果的 sql 命令

用 途　记录耗时较长或不使用索引的查询操作

启用日志：/etc/my.cnf

[mysqld]　启用慢查询

slow-query-log

log queries not using indexes=1　记录未使用索引的查询

重启数据库服务

systemctl restart mysqld

mysql 数据读写分离

数据读写分离：把后端访问数据库服务时的查询请求和写数据的请求给不同的数据库服务器处理。create database

人肉分离：

++

部署 mysql 数据读写分离架构

1　配置 mysql 主从同步

要求：把 20 配置为 10 的从库

配置主库 10

启 binlog 日志

用户授权

查看日志信息

配置从库 20

验证授权

指定 server_id

指定主库信息

查看从库状态

客户端验证

客户端验证主从同步配置

在主库添加访问数据的连接用户并设置密码

mysql>create database db12;

mysql>grant all on db12.* to yaya@"%" identified by "123456";

客户端连接主库，执行 sql 命令

mysql -h192.168.4.10 -uyaya -p123456

mysql>建库 建表 插入记录

mysql>

mysql>create table db12.1(id int);

mysql>insert into db12.1 values(110),(112);

在从库本机也能看到同样的数据

mysql -uroot -p123456

mysql>select * from db12.1

mysql 中间件:mysql-proxy　mycat　maxscale

##
读写分离

在主机 100 上部署代理服务，实现数据读写分离

1、 装包

rpm -ivh maxscale-2.1.2-1.rhel.7.x86_64.rpm

查一下

rpm -qc maxscale

ls /etc/maxscale.cnf

备份一下

cp /etc/maxscale.cnf /etc/maxscale.cnf.bak

2、修改主配置文件,并根据配置文件的设置在数据库服务器上添加对应的授权用户

vim /etc/maxs 修改主配置文件,并根据配置文件的设置在数据库服务器上添加对应的授权用户 cale.cnf

10 行　　threads=auto　　(本来默认 1)

18 行　 有两台数据库服务器

　　[server1]

type=server
address=192.168.4.10
port=3306
protocol=MySQLBackend

[server2]
type=server
address=192.168.4.20
port=3306
protocol=MySQLBackend

39 行　servers=server1, server2　(逗号空格)

40 行　user=abc　用户名密码随便起，但要记住

41 行　passwd=123456

53 行到 59 行注释

67 行　server=server1, server2　(逗号空格)

68 行　user=abcd　用户名密码随便起，但要记住

69 行　passwd=123456

86 行到 90 行注释

102 行下面添加　port=4099

保存退出

在主库 10 里

abc 用户

mysql> grant replication slave,replication client on *.* to

abc@"%" identified by "123456"; // 创建监控用户

abcd 用户

mysql> grant select on mysql.* to abcd@'%' identified by

"123456"; // 创建路由用户

查看一下

mysql>select user from mysql.user where user in ("abc","abcd");

在 100 上测试是否能连上数据库（测试授权）

mysql -h192.168.4.10 -uabc -p123456

mysql -h192.168.4.20 -uabc -p123456

mysql -h192.168.4.10 -uabcd -p123456

mysql -h192.168.4.20 -uabcd -p123456

3、启动服务

maxscale -f /etc/maxscale.cnf

查进程

ps -C maxscale

看端口

netstat -utnlp | grep maxscale

4、测试配置

1)在主机 100 上连接管理服务查看监控信息

maxadmin -uadmin -pmariadb -P4099

MaxScale>list servers　　列出所有服务器列表

2)在客户端连接 100 主机，测试访问数据时能否实现数据读写分离

ping -c 2 192.168.4.100

which mysql

mysql -h192.168.4.100 -P4006 -uyaya -p123456

客户端 client

mysql -h192.168.4.100 -u 用户名 -p 密码

##

day03

上午:

mysql 多实例

做之前先停 mysqld 服务并设为开机不自起
ststemctl stop mysqld
systemctl disable mysqld

步骤:
1、安装支持多实例服务的软件包
mysql-5.7.20-linux-glibc2.12-x86_64.tar.gz

tar -zxf mysql-5.7.20-linux-glibc2.12-x86_64.tar.gz

ls mysql-5.7.20-linux-glibc2.12-x86_64

mv mysql-5.7.20-linux-glibc2.12-x86_64 /usr/local/mysql

打开环境变量配置文件

vim /etc/profile

在最下面添加

export PATH=/usr/local/mysql/bin:$PATH

保存退出

source /etc/profile

2、修改主配置文件

mv /etc/my.cnf /etc/my.cnf.bak

vim /etc/my.cnf

[mysqld_multi] // 启用多实例

mysqld = /usr/local/mysql/bin/mysqld_safe // 指定进程文件的路径

mysqladmin = /usr/local/mysql/bin/mysqladmin // 指定管理命令路径

user = root // 指定调用进程的用户

[mysqld1] // 实例进程名称 ,1 表示实例名称 , 如 [mysqld1]

port=3307 // 端口号

datadir=/dir3307 // 数据库目录 ,要手动创建

socket=/dir3307/mysqld3307.sock // 指定 sock 文件的路径和名称.名称可以随便起，但要记住

pid-file=/dir3307/mysqld.pid //进程 pid 号文件位

置
log-error = /dir3307/mysqld.err // 错误日志位置

[mysqld2]
port=3308
datadir=/dir3308
socket=/dir3308/mysqld3308.sock
pid-file=/dir3308/mysqld.pid
log-error=/dir3308/mysqld.err
保存退出

3、根据配置文件做相应设置

mkdir /dir3307 /dir3308

chown mysql:mysql /dir3307 /dir3308

4、初始化授权库
cd /usr/local/mysql/bin
mysqld --user=mysql --basedir= 软件安装目录 --datadir= 数据库目录 – initialize

即:
cd /usr/local/mysql/bin

mysqld --user=mysql --basedir=/usr/local/mysql --datadir=/dir3307 --initialize

最下面会有一个初始密码　　N_ou2q9Y.bk!

mysqld　--user=mysql　--basedir=/usr/local/mysql --datadir=/dir3308 --initialize

最下面会有一个初始密码　axF:<z?tY0+Y

5、启动服务

启动服务

mysqld_multi start 1

查看端口

ss -antpu | grep :3307

启动服务

mysqld_multi start 2

查看端口

ss -antpu | grep :3308

查看进程

ps -C mysqld

端口

ss -antpu | grep mysqld

ps aux | grep mysqld

6、客户端访问

连接本机的数据库服务

cd /usr/local/mysql/bin

mysql -uroot –p'初始密码' -S sock 文件

即：

用初始密码登录

mysql -uroot – p'N ou2q9Y.bk!' -S /dir3307/mysqld3307.sock

改密码:

mysql> alter user root@"localhost" identified by "123456"

mysql>quit;

用新密码登录

mysql -uroot – p123456 -S /dir3307/mysqld3307.sock

mysql>show databases;

然后就建库 建表了

用初始密码登录

mysql -uroot – p'axF:<z?tY0+Y' -S /dir3308/mysqld3308.sock

改密码:

mysql> alter user root@"localhost" identified by "123456"

mysql>quit;

用新密码登录

mysql -uroot – p123456 -S /dir3308/mysqld3308.sock

mysql>show databases;

然后就建库　建表了

7、停止指定的多实例服务(起的话查看第 5 步启动服务)

mysqld_multi --user=root --password=
密码 stop 实例编号

即

停 3307
mysqld_multi --user=root --password=
1234123456 stop 1

停 3308
mysqld_multi --user=root --password=
123456 stop 2

##
##################
下午：

部署 mysql 高可用集群（第三方软件 MHA+主从同步）

集群：使用多台服务器提供相同的服务（如 51-55）

高可用集群：主备模式 当主角色的主机宕机后，备用的主机自动接替主角色的主机提供服务给客户端。

client mysql -h192.168.4.52 -uadmin -p123456

56 监控服务

vip 地址 192.168.4.100

vip

主 备用主 备用主

51 52 53 54 55 56

master slave slave slave slave 监控

一、配置所有数据节点主机之间可以互相以 ssh 密钥对方式认证登陆

ssh-keygen

ssh-copy-id root@ip 地址

二、配置 manager56 主机 无密码 ssh 登录 所有数据节点主机

ssh-keygen

ssh-copy-id root@ip 地址

三、配置主从同步,要求如下：

51 主库 开半同步复制

52 从库（备用主库） 开半同步复制

53 从库（备用主库） 开半同步复制

54 从库 不做备用主库所以不用开半同步复制

55 从库 不做备用主库所以不用开半同步复制

master51 配置：

]# vim /etc/my.cnf

[mysqld]

plugin-load = "rpl_semi_sync_master=semisync_master.so ; rpl_semi_sync_slave=semisync_slave.so"

rpl-semi-sync-master-enabled = 1

rpl-semi-sync-slave-enabled = 1

server_id=51

log-bin=master51

binlog-format="mixed"

:wq

]# systemctl restart mysqld

]# ls /var/lib/mysql/master51.*

]# mysql -uroot -p123456

mysql> grant replication slave on *.* to

repluser@"%"   identified by "123456";
mysql> set global relay_log_purge=off;
mysql> show master status;
mysql> quit;

备用 master52 的配置
]# vim   /etc/my.cnf
[mysqld]
plugin-load =
"rpl_semi_sync_master=semisync_master.so;
rpl_semi_sync_slave=semisync_slave.so"
rpl-semi-sync-master-enabled = 1
rpl-semi-sync-slave-enabled = 1

server_id=52
log-bin=master52
binlog-format="mixed"

]# systemctl restart mysqld
]# ls /var/lib/mysql/master52.*
]# mysql   -uroot   -p123456
mysql> grant replication slave on *.* to
repluser@"%"   identified by "123456";
mysql> set global relay_log_purge=off;
mysql> change master to
    -> master_host="192.168.4.51",
    -> master_user="repluser",

-> master_password="123456",

-> master_log_file="master51.000001",

-> master_log_pos=441;

mysql> start slave;

mysql> show slave status\G;

]# mysql -uroot -p123456 -e "show slave status\G" grep -i YES

备用 master53 的配置

]# vim /etc/my.cnf

[mysqld]

plugin-load = "rpl_semi_sync_master=semisync_master.so;

rpl_semi_sync_slave=semisync_slave.so"

rpl-semi-sync-master-enabled = 1

rpl-semi-sync-slave-enabled = 1

server_id=53

log-bin=master53

binlog-format="mixed"

:wq

]# systemctl restart mysqld

]# ls /var/lib/mysql/master53.*

]# mysql -uroot -p123456

mysql> grant replication slave on *.* to repluser@"%" identified by "123456";

mysql> set global relay_log_purge=off;

```
mysql> change master to
    -> master_host="192.168.4.51",
    -> master_user="repluser",
    -> master_password="123456",
    -> master_log_file="master51.000001",
    -> master_log_pos=441;
mysql> start slave;
mysql> show slave status\G;
]# mysql -uroot -p123456 -e "show slave status\G"
| grep -i yes
```

配置从服务器 54

```
]# vim /etc/my.cnf
[mysqld]
server_id=54
:wq
]# systemctl   restart mysqld
]# mysql -uroot -p123456
mysql> set global relay_log_purge=off;
mysql> change master to
    -> master_host="192.168.4.51",
    -> master_user="repluser",
    -> master_password="123456",
    -> master_log_file="master51.000001",
    -> master_log_pos=441;
mysql> start slave;
mysql> show slave status\G;
```

]# mysql -uroot -p123456 -e "show slave status\G" | grep -i yes

配置从服务器 55

]# vim /etc/my.cnf

[mysqld]

server_id=55

:wq

]# systemctl restart mysqld

]# mysql -uroot -p123456

mysql> set global relay_log_purge=off;

mysql> change master to

    -> master_host="192.168.4.51",

    -> master_user="repluser",

    -> master_password="123456",

    -> master_log_file="master51.000001",

    -> master_log_pos=441;

mysql> start slave;

mysql> show slave status\G;

]# mysql -uroot -p123456 -e "show slave status\G" | grep -i yes

在客户端测试主从同步配置

在主库 51 上添加访问数据的授权用户

]# mysql -uroot -p123456

mysql> grant all on gamedb.* to admin@"%" identified by "123456";

mysql> create database gamedb;

mysql> create table gamedb.t1 (id int);

mysql> insert into gamedb.t1 values(999);

mysql> insert into gamedb.t1 values(999);

mysql> select * from gamedb.t1;

在客户端使用授权用户连接从库 52-55,也能看到同样的库表及记录

]# mysql -h 从库 IP 地址 -uadmin -p123456

mysql> select * from gamedb.t1;

在所有主机上安装 perl 软件包 （51~56）

]# cd mha-soft-student

]# yum -y install perl-*.rpm

在所有主机上安装 mha_node 软件包 （51~56）

yum -y install perl-DBD-mysql

rpm -ivh mha4mysql-node-0.56-0.el6.noarch.rpm

只在管理 "主机 56" 上安装 mha_manager 软件包

]# yum -y install perl-ExtUtils-* perl-CPAN*

]#tar -zxvf mha4mysql-manager-0.56.tar.gz

]#cd mha4mysql-manager-0.56

]#perl Makefile.PL

]#make

]#make install

检查配置环境

在主机 51 52 53 检查是否有同步数据的用户 repluser

mysql> show grants for repluser@"%" ;

在主机 51~55 做如下授权

]# mysql -uroot -p123456

mysql> grant all on *.* to root@"%" identified by "123456";

在 56 上操作

拷贝命令

]# cd mha-soft-student

cp mha4mysql-manager-0.56/bin/* /usr/local/bin/

创建工作目录 和主配置文件

]# mkdir /etc/mha_manager/

cp mha4mysql-manager-0.56/samples/conf/app1.cnf /etc/mha_manager/

vim /etc/mha_manager/app1.cnf

[server default]

manager_workdir=/etc/mha_manager

manager_log=/etc/mha_manager/manager.log

master_ip_failover_script=/etc/mha_manager/master_ip_failover

ssh_user=root

ssh_port=22

repl_user=repluser

repl_password=123qqq...A

user=root

password=123aaa...A

[server1]

hostname=192.168.4.51

candidate_master=1

[server2]

hostname=192.168.4.52

candidate_master=1

[server3]

hostname=192.168.4.53

candidate_master=1

[server4]

hostname=192.168.4.54

no_master=1

[server5]

hostname=192.168.4.55

no_master=1

保存退出

创建故障切换脚本

cp mha4mysql-manager-0.56/samples/scripts/master_ip_failover /usr/local/bin/

cp mha4mysql-manager-0.56/samples/scripts/master_ip_failover /etc/mha_manager/

vim /etc/mha_manager/master_ip_failover

#!/usr/bin/env perl

Copyright (C) 2011 DeNA Co.,Ltd.

#

This program is free software; you can redistribute it and/or modify

it under the terms of the GNU General Public License as published by：

the Free Software Foundation; either version 2

of the License, or

(at your option) any later version.

#

This program is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

GNU General Public License for more details.

#

You should have received a copy of the GNU General Public License

along with this program; if not, write to the Free Software

Foundation, Inc.,

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Note: This is a sample script and is not complete. Modify the script based on your environment.

use strict;

use warnings FATAL => 'all';

use Getopt::Long;

use MHA::DBHelper;

```
my (
  $command,                    $ssh_user,
$orig_master_host,
  $orig_master_ip,             $orig_master_port,
$new_master_host,
  $new_master_ip,              $new_master_port,
$new_master_user,
  $new_master_password
);

my $vip = '192.168.4.60/24';   # Virtual IP
my $key = "1";
my $ssh_start_vip = "/sbin/ifconfig eth0:$key
$vip";
my $ssh_stop_vip = "/sbin/ifconfig eth0:$key
down";

GetOptions(
  'command=s'          => \$command,
  'ssh_user=s'         => \$ssh_user,
  'orig_master_host=s'                  =>
\$orig_master_host,
  'orig_master_ip=s'   => \$orig_master_ip,
  'orig_master_port=i' => \$orig_master_port,
  'new_master_host=s'                   =>
\$new_master_host,
  'new_master_ip=s'    => \$new_master_ip,
```

```
  'new_master_port=i'                    =>
\$new_master_port,
  'new_master_user=s'                    =>
\$new_master_user,
  'new_master_password=s'                =>
\$new_master_password,
);


exit &main();


sub main {
  if ( $command eq "stop" || $command eq
"stopssh" ) {


    #      $orig_master_host,      $orig_master_ip,
$orig_master_port are passed.
    # If you manage master ip address at global
catalog database,
    # invalidate orig_master_ip here.
    my $exit_code = 1;
    eval {


      # updating global catalog, etc
      &stop_vip();
      $exit_code = 0;
    };
    if ($@) {
      warn "Got Error: $@\n";
```

```
        exit $exit_code;
      }
      exit $exit_code;
    }
    elsif ( $command eq "start" ) {


      # all arguments are passed.
      # If you manage master ip address at global catalog database,
      # activate new_master_ip here.
      # You can also grant write access (create user, set read_only=0, etc) here.
      my $exit_code = 10;
      eval {
        my $new_master_handler = new MHA::DBHelper();


        # args: hostname, port, user, password, raise_error_or_not
        $new_master_handler->connect( $new_master_ip, $new_master_port,
          $new_master_user, $new_master_password, 1 );


        ## Set read_only=0 on the new master

$new_master_handler->disable_log_bin_local();
        print "Set read_only=0 on the new
```

```
master.\n";
        $new_master_handler->disable_read_only();

        ## Creating an app user on the new master
        print "Creating app user on the new
master..\n";

$new_master_handler->enable_log_bin_local();
        $new_master_handler->disconnect();

        ## Update master ip on the catalog
database, etc
        &start_vip();
        $exit_code = 0;
      };
      if ($@) {
        warn $@;

        # If you want to continue failover, exit 10.
        exit $exit_code;
      }
      exit $exit_code;
    }
    elsif ( $command eq "status" ) {

      # do nothing
      exit 0;
```

```
  }
  else {
    &usage();
    exit 1;
  }
}
sub start_vip() {
    `ssh $ssh_user\@$new_master_host \"$ssh_start_vip \"`;
}
sub stop_vip() {
    return 0 unless ($ssh_user);
    `ssh $ssh_user\@$orig_master_host \"$ssh_stop_vip \"`;
}

sub usage {
  print
"Usage: master_ip_failover --command=start|stop|stopssh|status --orig_master_host=host --orig_master_ip=ip --orig_master_port=port --new_master_host=host --new_master_ip=ip --new_master_port=port\n";
}
```

保存退出

```
cd /etc/mha_manager
```

加个执行权限

chmod +x master_ip_failover

验证配置

验证 ssh 免密码登录 数据节点主机

]# masterha check ssh --conf=/etc/mha_manager/app1.cnf

验证 数据节点的主从同步配置（要不调用故障切换脚本）

]#

masterha check repl --conf=/etc/mha_manager/app1.cnf

测试高可用集群配置

在主库上手动部署 vip 地址 192.168.4.60

]# ifconfig eth0:1 192.168.4.60/24

]# ifconfig eth0:1

启动管理服务，并查看服务状态

]# masterha_manager --conf=/etc/mha/app1.cnf --remove_dead_master_conf --ignore_last_failover

开一个新端口

]# masterha check status --conf=/etc/mha_manager/app1.cnf

测试故障转移

在主库 51 上执行]# systemctl stop mysqld 要修

复
先起数据库服务

进 51 把 51 设为 52 的从库

进 56 的配置文件/etc/mha/app1.cnf 添加 server1
执行 masterha_manager --conf=/etc/mha/app1.cnf
--remove_dead_master_conf --ignore_last_failover

开一个新端口
]# masterha check status
--conf=/etc/mha_manager/app1.cnf

在 52 本机查看是否获取 vip 地址
]# ip addr show | grep 192.168.4

客户端连接 vip 地址 ， 访问数据服务
]#mysql -h192.168.4.100 -uadmin -p123456

##
##################

day 04

mysql>create database db9;
mysql> use db9

```
mysql> create table user(
    -> name char(20),
    -> password char(1),
    -> uid int(2),
    -> gid int(2),
    -> comment char(150),
    -> homedir char(150),
    -> shell char(30)
    -> );
mysql> system cp /etc/passwd /var/lib/mysql-files/
mysql> load data infile "/var/lib/mysql-files/passwd" into table user fields terminated by ":" lines terminated by "\n";
mysql> alter table user add id int(2) primary key auto_increment first;
```

视图： 虚拟表(假表)

视图的基本使用：

创建

create view 视图名称 as sql 查询;

mysql> create view t1 as select name,shell from user where uid<=20;

create view 视图名称(字段名列表) as sql 查询;

字段名列表要和表结构的一样

create view t4(user,stu id,stu g) as select name,uid,gid from user limit 3;

create view v5 as select name,id,shell,password from user;

查看创建视图的具体命令

show create view 视图名;

mysql> show create view t11\G;

查看 （那些表时视图表，那些表时物理表）

mysql> show table status where comment="view"\G;

查看当前库下所有表的状态信息

show table status;

查看视图的数据时 file:///usr/share/doc/HTML/index.html 哪个基表里的

mysql> show create view t1;

使用视图

查询记录

select 字段名列表 from 视图名 where 条件;

插入记录

insert into 视图名 (字段名列表) values(字段值列表);

更新记录

update 视图名 set 字段名 = 值 where 条件;

删除记录

delete from 视图名 where 条件;

删除视图

drop view 视图名

视图进阶

创建视图完全格式：

设置字段名

创建视图时必须给视图中的字段定义别名的情况

mysql> create table user2 select name,uid,shell from user limit 5;

mysql>create table info select name,uid,shell from user limit 10;

查看视图重复的

mysql> select * from user2 left join info on user2.uid=info.uid;

格式:

mysql>create view v3 as select a.name as aname,b.name as bname,a.uid as auid,b.uid as buid from user2 a left join info b on a.uid=b.uid;

mysql>create view v22 as select user2.name as uname,info.name as iname from user2 left join info on user2.uid=info.uid;

mysql> select * from v22;

```
+--------+--------+
| uname  | iname  |
+--------+--------+
| root   | root   |
| bin    | bin    |
| daemon | daemon |
| adm    | adm    |
| lp     | lp     |
+--------+--------+
5 rows in set (0.00 sec
```

重写已有的视图

mysql> create or replace view v22 as select * from user;

选项

with check option 作用：

定义对视图表里的数据做操作时的限制方式。

local 的方式满足视图本身的限制

cascaded 同时满足基表的限制

create table user2 select name,uid,shell from user;

select * from user2;

创建视图 v1:

create view v1 as select * from user2 where uid <=30;

select * from v1;

创建视图 v2:

create view v1 as select * from v1 where uid >=20 with check option;

查看

select * from v2;

查看视图

show create view v2;

修改操作

不满足自己

update v2 set uid=19 where name="mysql";

不满足基本

update v2 set uid=39 where name="mysql";

同时满足

update v2 set uid=29 where name="mysql";

创建 v3 视图

create view v3 as select * from v1 where uid>=20 with local check option;

show create view v3;

select * from v3;

不满足条件（条件是：创建视图 v3 的时候　uid>=20）

update v3 set uid=10 where name="mysql";

update v3 set uid=30 where name="mysql";

select * from v3 where name="mysql";

select * from v3;

select * from v1;

select * from v1 where name="mysql";

select * from user2;

存储过程的优点“：

提高性能
可减轻网络负担
可以防止对表的直接访问
避免重复的 sql 操作

创建存储过程
语法格式：

```
delimiter //   把命令结尾改为//
create procedure  名称()
begin
        功能代码   每行代码要以分号结尾
    .....
end
//          结束存储过程
delimiter;      把命令结尾改为回来为分号
```

例：

```
mysql> delimiter //
mysql> create procedure say()
    -> begin
    -> select * from db9.user;
    -> select * from mysql.user where user="root";
    -> end
        -> //
```

Query OK, 0 rows affected (0.04 sec)

mysql> delimiter ;

调用存储（名称后面加不加括号都可以）
格式：
mysql> call 名称()

例

mysql>call say();
mysql>call say;

查看存储过程方法：
方法 1：
mysql> show procedure status;

方法 2：
mysql> select db,name,type from mysql.proc where name=“ 存储过程名 ";

mysql 存储过程
mysql>desc mysql.proc;
mysql>select db,name,type from mysql.proc;

mysql>select db,name,type from mysql.proc where name="say";

查看 say1 存储过程怎么写的

mysql>select * from mysql.proc where name="say1"\G;

删除存储过程：

mysql>drop procedure 名称;

例

mysql>drop procedure say1;

编写功能代码时，可以使用变量 条件判断 流程控制（if 循环） 算术计算 sql 命令

例

满足以下要求:

- 存储过程名称为 p1
- 功能显示 user 表中 shell 是 /bin/bash 的用户个数
- 调用存储过程 p1

```
delimiter //
create procedure p1()
```

```
begin
select count(name) from db9.user where shell="/bin/bash";
end
//
delimiter ;

mysql>call p1();
```

变量

会话变量

会话变量和全局变量叫系统变量 使用 set 命令定义;

全局变量

全局变量的修改会影响到整个服务器,但是对会话变量的修改,只会影响到当前的会话。

用户变量（自定义变量）

在客户端连接到数据库服务的整个过程中都是有效的。当当前连接断开后所有用户变量失效。

定义 set

@ 变量名 = 值;

输出 select @ 变量名;

局部变量

存储过程中的 begin/end 。其有效范围仅限于该语句块中,语句块执行完毕后,变量失效。

declare 专门用来定义局部变量。

注意:局部变量 和 参数变量 调用时 变量名前不需要加 @

mysql> show global variables; // 查看全局变量

mysql> show session variables; // 查看会话变量

mysql> set session sort buffer size = 40000; // 设置会话变量

Mysql> show global variables like “% 关键字 %”; // 查看全局变量

查看 say1 存储过程怎么写的

格式

mysql>select * from mysql.proc where name=“ 存储过程名 "\G;

例

mysql>select * from mysql.proc where

name="say1"\G;

自定义变量：

mysql>set @name="值"；

例

mysql>set @name="yaya";
mysql>set @x=1;
select max(uid) from db9.user;
select max(uid) into @y from db9.user;
select @y;

查看

mysql>select @name
mysql> select @y;

局部变量（最好在存储过程中写）

mysql>declare @y=1;　错误的

例

delimiter //
create procedure p2()
begin

```
declare x int default 77;
declare y char(10);
set y="yaya";
select x;
select y;
end
//
delimiter ;

mysql>call p2();

mysql>select * from mysql.proc where name="p2"\G;



delimiter //
create procedure p3()
begin
declare x int default 77;
select x;
select max(uid) into x from db9.user;
select x;
end
//
delimiter ;
```

mysql>call p3();

mysql>select * from mysql.proc where name="p3"\G;

变量

算术运算

符号	描述	例子
+	加法运算	SET @var1=2+2; 4
-	减法运算	SET @var2=3-2; 1
*	乘法运算	SET @var3=3*2 ; 6
/	除法运算	SET @var4=10/3; 3.333333333
DIV	整除运算	SET @var5=10 DIV 3; 3
%	取模(取余)	SET @var6=10%3 ; 1

mysql> set @z=1+2;select @z;

mysql> set @x=1; set @y=2; set @z=@x*@y; select @z;

mysql> set @x=1; set @y=2; set @z=@x-@y; select @z;

mysql> set @x=1; set @y=2; set @z=@x/@y;

```
select @z;



delimiter //
create procedure p4()
begin
declare x int;
declare y int;
declare z int;
select count(shell) into x from db9.user where shell="/bin/bash";
select count(shell) into y from db9.user where shell="/sbin/nologin";
set z=x+y;
select z;
end
//
delimiter ;


mysql>call p4();


mysql>select * from mysql.proc where name="p4"\G;
```

变量

算术计算

存储过程参数类型

in　输入类型参数（默认值）

out　输出类型参数

inout　输入输出型参数

注意:此三中类型的变量在存储过程中调用时不需要加 @ 符号 !!!

指定存储过程参数的格式：类型　　参数名　参数的类型（宽度）

create procedure p6(参数 1，参数 2)

begin

.....

end

##

in 类型参数：

p5()　括号里面可以添加参数

mysql>call p4();　括号里面需要给表里面有的值

delimiter //

create procedure p5(in username char(20))

begin

select name from db9.user where

```
name=username;
end
//
delimiter ;

mysql>call p5();   错误操作
mysql>call p5("rpc");
mysql>call p5("lisi");

mysql>set @name="lisi";
mysql>call p5(@name);

mysql>select * from mysql.proc where name="p5"\G;



##############################################################
out 类型参数

delimiter //
create procedure p6(out num int(2) )
begin
select num;
set num=7;
select num;
select count(name) into num from db9.user where
```

```
shell!="/bin/bash";
select num;
end
//
delimiter ;
```

mysql>call p6(); 只要参数里面有值，括号里不能为空
mysql>call p6(7); 执行后报错

```
set @x=1;
```
mysql>call p6(@x);

mysql>select * from mysql.proc where name="p6"\G;

##
inout 类型参数

```
delimiter //
create procedure p7(inout num int(2) )
begin
select num;
set num=7;
select num;
select count(name) into num from db9.user where
```

```
shell!="/bin/bash";
select num;
end
//
delimiter ;

set @x=9;     赋值
mysql> call p7(@x);       查看


mysql>select * from mysql.proc where name="p7"\G;

##############################################################
```

条件判断 与查询命令 select 的条件判断语法相同

此处的条件判断是给流程+

OR 、 AND 、 !	逻辑或、逻辑与、逻辑非
IN .. 、 NOT IN ..	在 .. 范围内、不在 .. 范围内
IS NULL	字段的值为空
IS NOT NULL	字段的值不为空
LIKE	模糊匹配
REGEXP	正则匹配

条件判断：

```
delimiter //
create procedure p8(in num int(2) )
begin
   if num <= 10 then
       select * from db9.user where id <= num;
   end if;
end
//
delimiter ;

mysql> call p8(3);
```

也可以用变量赋值

```
mysql> set @x=7;
mysql> call p8(@x);

mysql>select * from mysql.proc where name="p8"\G;

##############################################
```

加一个判断在执行

```
delimiter //
create procedure p9(in num int(2) )
```

```
begin
   if num is null then
       select * from db9.user where id = 2;
    else
       select * from db9.user where id <= num;
   end if;
end
//
delimiter ;
```

```
mysql> call p9(5);
```

```
mysql>select * from mysql.proc where name="p9"\G;
```

```
##############################################
#########
```

循环结构

条件式循环　　反复测试条件,只要成立就执行命令序列

while 循环

```
##############################################
#########
```

uid 号是偶数的用户名和 uid 输出一遍

```
delimiter //
create procedure p11()
begin
   declare i int(2);
   declare j int(2);
 select count(id) into i from db9.user;
   set j=1;
   while j <= i do
        if j % 2 = 0 then
           select * from db9.user where id = j;
      end if;
     set j=j+1;
     end while;
end
//
delimiter ;
```

调用

```
mysql>call p10();
```

```
##############################################
###############
```

loop 死循环 无条件执行

```
delimiter //
create procedure p12()
begin
declare j int(2);
```

```
   set j=1;
    loop
    select j;
       set j=j+1;
 end loop;
end
//
delimiter ;

mysql>call p12();

##############################################
#############

until 条件判断,不成立时成立循环
                                成立时，跳出循环
until j=6      此处不用加分号

delimiter //
create procedure p13()
begin
declare j int(2);
   set j=1;
    repeat
    select j;
       set j=j+1;
       until j=6
```

```
 end repeat;
end
//
delimiter ;

mysql>call p13();
```

试验： until j=1 时，死循环

```
delimiter //
create procedure p14()
begin
declare j int(2);
   set j=1;
    repeat
    select j;
       set j=j+1;
       until j=1
 end repeat;
end
//
delimiter ;

mysql>call p14();
```

试验： until j=2 时，输出第一个值

```
delimiter //
create procedure p15()
begin
declare j int(2);
   set j=1;
    repeat
    select j;
       set j=j+1;
       until j=2
 end repeat;
end
//
delimiter ;
```

```
mysql>call p15();
```

##
##########

控制循环结构的执行

循环结构控制语句 , 控制循环结构的执行。

LEAVE 标签名 // 跳出循环（结束循环的执行）

ITERATE 标签名 //放弃本次循环,执行下一次循环

用 while 循环输出数字 1-10

```
delimiter //
create procedure p16()
begin
declare i int(2);
   set i=1;
     while i <= 10 do
        select i;
        set i=i+1;
  end while;
end
//
delimiter ;
```

```
mysql>call p16();
```

```
LEAVE loab1;     跳出整个循环
ITERATE loab1;  放弃本次循环,执行下一次循环
```

```
delimiter //
create procedure p18()
begin
declare i int(2);
   set i=1;
        loab1 :loop
```

```
        select i;
        set i=i+1;
LEAVE loab1;
  end loop;
end
//
delimiter ;

mysql>call p18();
```

ITERATE loab1; 放弃本次循环,执行下一次循环(死循环)

```
delimiter //
create procedure p19()
begin
declare i int(2);
   set i=1;
        loab1 :loop
        select i;
        set i=i+1;
ITERATE loab1;
  end loop;
end
//
delimiter ;
```

```
mysql>call p19();
```

只输出 1、2 的值之后就卡住了

```
delimiter //
create procedure p20()
begin
declare i int(2);
   set i=1;
        loab1 :loop
       if i = 3 then
         ITERATE loab1;
        end if;
        select i;
        set i=i+1;
ITERATE loab1;
 end loop;
end
//
delimiter ;

mysql>call p20();
```

换了位置：（死循环）

```
delimiter //
create procedure p21()
begin
declare i int(2);
   set i=1;
        loab1 :loop
        select i;
       if i = 3 then
         ITERATE loab1;
        end if;
        set i=i+1;
ITERATE loab1;
 end loop;
end
//
delimiter ;

mysql>call p21();
```

##
用 ITERATE 循环输出数字 1-10

```
delimiter //
```

```
create procedure p23()
begin
declare i int(2);
   set i= 0;
        loab1:while i < 10 do
        set i = i+1;
       if i = 7 then
         ITERATE loab1;
        end if;
        select i;
 end while;
end
//
delimiter ;

mysql>call p23();
```

##
##########################

day 05

上午

分库分表

解决但数据库服务器的访问压力和存储压力

解决单表过大的问题

##

实验：

分库分表

56 上做 mycat　　54 和 55 时数据库服务器

54 上:

]# cd /var/lib/mysql

]# ls

]# rm -rf 55-relay-bin.* master.info relay-log.info

]# systemctl restart mysqld

]# mysql -uroot -p123456

留下四个初始的库其他的库都删了

mysql> show slave status\G;　测验一下查询不到就对了

mysql> create database db1;　创建数据库 db1

mysql> use db1;

mysql> show tables;

mysql> grant all on *.* to root@"%" identified by "123456";　　用户授权

mysql> show databases;

mysql> quit;

]# vim /etc/my.cnf

[mysqld]

lower_case_table_names = 1

]# systemctl restart mysqld

55 上

]# cd /var/lib/mysql

]# ls

]# rm -rf 54-relay-bin.* master.info relay-log.info

]# systemctl restart mysqld

]# mysql -uroot -p123456

留下四个初始的库其他的库都删了

mysql> show slave status\G;　测验一下查询不到就对了

mysql> create database db2;　创建数据库 db1

mysql> use db2;

mysql> show tables;

mysql> grant all on *.* to root@"%" identified by "123456";　用户授权

mysql> show databases;

mysql> quit;

]# vim /etc/my.cnf

[mysqld]

lower_case_table_names = 1

]# systemctl restart mysqld

56 上

]# masterha stop --conf=/etc/mha_manager/app1.cnf

]# systemctl stop mysqld

]# ping -c 3 192.168.4.54 能通

]# ping -c 3 192.168.4.55 能通

配置数据分片数据 56 步骤

1 装包

]# rpm -qa | grep -i jdk # 安装系统自带的即可

]# tar -zxf Mycat-server-1.4-beta-20150604171601-linux.tar.gz 解压

]#mv mycat/ /usr/local

2 改配置

]#cd /usr/local/mycat/conf/

]#vim rule.xml

]#vim server.xml 设置连接 mycat 服务的账号 、密码等

]# cp schema.xml schema.xml.bak 备份重要文件

]#vim schema.xml 配置 mycat 使用的真实数据库和表

2.1 定义连接 mycat 服务的用户和密码及虚拟数据库名称

]#vim server.xml

2.2 对那些表做数据分片及使用的分片规则

]#vim schema.xml 逻辑表名 使用的分片规则 存储到哪个数据库服务器 dn1 dn2

指定 dn1 存储数据的库名 db1

指定 dn2 存储数据的库名 db2

指定 dn1 对应的数据库服务器 ip 地址

指定 dn2 对应的数据库服务器 ip 地址

根据配置在指定的数据库服务器上创建存储数据库的库和连接用户

两台数据库服务器

]#vim schema.xml

行 7 删掉 dn3

行 11 删掉 dn3

行 15 删掉 dn3

行 37 和 38 改称这样

<dataNode name="dn1" dataHost="c1" database="db1" />

<dataNode name="dn2" dataHost="c2" database="db2" />

行 39 注释掉 <!-- 内容 -->

行 40 注释掉 <!-- 内容 -->

行 43

<dataHost name="c1" maxCon="1000" minCon="10" balance="0"

行 47

<writeHost host="hostM1" url="192.168.4.55:3306" user="root"

行 52-53 注释掉 <!-- 内容 -->

复制 43-55 行 粘贴在 56 行之后

<dataHost name="c2" maxCon="1000" minCon="10" balance="0"

writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">

<heartbeat>select user()</heartbeat>

<!-- can have multi write hosts -->

<writeHost host="hostM1" url="192.168.4.54:3306" user="root"

]# cd /var/lib/mysql password="123456">

<!-- can have multi read hosts -->

</writeHost>

<!--<writeHost host="hostS1" url="localhost:3316" user="root"

password="123456" /> -->

<!-- <writeHost host="hostM2" url="localhost:3316" user="root" password="123456"/> -->

</dataHost>

3 启动或停止服务（56 上）

]#cd /usr/local/mycat/bin/

]# /usr/local/mycat/bin/mycat start|stop

]# ls /usr/local/mycat/logs/

]# cat /usr/local/mycat/logs/mycat.pid 会看到一个 pid 值

]# ps -P 2610 查看 pid 值

]# ss -antpu | grep :8066 查看端口是否启动

]# ss -antpu | grep java 查看

4 查看服务信息

]# ss -antpu | grep :8066

5 客户端测试配置

]# which mysql

]# mysql -h192.168.4.56 -P8066 -utest -ptest

show databases;

use TESTDB;

show tables;

desc employee;

create table employee(id int not null primary key,name varchar(100),age int(2),sharding_id int not null);

insert into employee(id,name,age,sharding_id) values(1,"bob",21,10000),(2,"lucy",18,10010),(3,"boba",21,10000),(4,"lucyf",18,10010);

select * from employee;

在 55 和 54 上分别查看一下

select * from employee;

##

DBA 基础

mysql 服务基本使用

mysql 服务 常用的

主从同步

数据读写分离

mysql 高可用

多实例

视图 存储过程

##
########

nosql

在主机 50 上部署内存存储数据库服务器 redis

装软件 redis-4.0.8.tar.gz

配置步骤

装包

]# tar -zxf redis-4.0.8.tar.gz

]# ls

]# cd redis-4.0.8/

]# yum -y install gcc gcc-c++

]# make

]# make install

]# cd utils/

]# ls

]# ./install_server.sh　一直回车

启动服务

服务默认启动的

服务的查看状态 启动　停止

]# /etc/init.d/redis_6379 status|start|stop

查看端口

]# ss -antpu | grep :6379

查看进程

]# ps -C redis-server

访问 redis 存储数据

]# redis-cli

127.0.0.1:6379> ping

PONG

输入 ping　　回复 PONG 就可以

存数据

127.0.0.1:6379> set name bob

取数据

127.0.0.1:6379> get name

退出

127.0.0.1:6379> quit

存放数据位置

]# ls /var/lib/redis/6379/

在主机 51 上部署内存存储数据库服务器 redis

装软件 redis-4.0.8.tar.gz

配置步骤
装包
]# tar -zxf redis-4.0.8.tar.gz
]# ls
]# cd redis-4.0.8/
]# yum -y install   gcc gcc-c++
]# make
]# make install
]# cd utils/
]# ls
]# ./install_server.sh   调用脚本一直回车

启动服务
服务默认启动的
服务的查看状态 启动   停止

]# /etc/init.d/redis_6379 status|start|stop

查看端口

]# ss -antpu | grep :6379

查看进程

]# ps -C redis-server

访问 redis 存储数据

]# redis-cli

127.0.0.1:6379> ping

PONG

输入 ping　　回复 PONG 就可以

存数据

127.0.0.1:6379> set name bob

取数据

127.0.0.1:6379> get name

退出

127.0.0.1:6379> quit

存放数据位置

]# ls /var/lib/redis/6379/

redis

常用操作指令

Set keyname keyvalue // 存储

get keyname // 获取
Select 数据库编号 0-15 // 切换库
Keys * // 打印所以变量
Keys a? // 打印指定变量
Exists keyname // 测试是否存在
ttl keyname // 查看生存时间

ttl name
(integer) -2 -2 代表过期

type keyname // 查看类型
move keyname dbname // 移动变量
expire keyname 10 // 设置有效时间
del keyname // 删除变量
flushall // 删除所有变量
save // 保存变量
shutdown // 关闭服务

##
##################

问题（复习）：

执行新注册的帐号

```
delimiter //
create procedure getnum(in riqi date)
begin

select count(login_name) from rge_tab where
注册时间>riqi;

end
//
delimiter ;

mysql>call getnum(2018070201000);

##########################################
####################
innobackupex 做完全备份 db1 库 a 表  b 表

/db1allbak

模拟数据丢失 删除表

从备份文件中导出表信息  a.exp a.cnf a.frm a.ibd
                         b.exp b.cnf b.frm b.ibd

创建删除的表（表结构要和删除时的相同）
b.frm b.ibd
```

删除表空间文件

把导出的表信息文件拷贝到数据库目录下
*.exp *.cnf *.ibd

导入表信息

导完之后删掉这两个文件
rm -rf *.exp *.cnf

查看数据是否恢复成功

select * from 表名;

##
###########

为什么要做数据读写分离

 装包
修改配置文件
maxscale.cnf

指定服务运行的线程数量

指定数据库服务器[server 数字]

指定监控的数据库服务器[mysql_monitor]

定义读写分离服务[read_writer_services]

定义管理服务使用的端口号

定义读写分离使用的端口号

在数据库服务器上添加有对应访问权限的用户
添上这两个用户 scalemon 和 maxscal

启动 maxscal 服务并查看端口和进程名

客户端连接 maxscal 服务访问数据

##
##########################

binlog 日志

安装开源的 percona 软件提供的 innobackupex 命令做增量备份

首先要有一次备份（通常时完全备份）
--datebases="库名"不写就是有多少库备份多少库
innobackipex --user 用户名 --password 密码

--datebases="库名" 目录名 --no-timestamp

增量备份

innobackipex --user 用户名 --password 密码 --datebases="库名" --incremental /nwe1dir --incremental-basedir= 上一次备份目录名 --no-timestamp

##

多表查询什么时候用？

嵌套查询

把内层的查询条件作为外层的查询条件

select 字段名列表 from 表名 where 条件 (select 字段名列表 from 表名 where 条件);

select 字段名列表 from 表名列表 where 条件;

表名.字段名,表名.*

left join 左连接

right join 右连接

左连接(查询的结果更直观)

select 字段名列表 from 表名 A left join 表名 B on 条件;

右左连接

select 字段名列表 from 表名 A right join 表名 B on 条件;

##

变量

show variables; 列出所有的变量

mysql 在生产环境中的优化

show variables lile "%cache%";

set [global] 变量名=值;

内存相关变量

mysql 帮助手册 主配置文件参数详细说明 my.cnf

##

高可用集群的配置步骤：

配置一主多从（保证有三台机器做主库并开半同步复制）

全同步模式

异步同步模式

半同步模式

配置 mha 集群：

配置数据节点 51-55 安装 MHA_node

配置管理节点 56 安装 MHA_node MHA_manager

修改管理主机的主配置文件

app1.cnf

管理主机的配置[server default]

定义数据库服务器[server 数字]

测试配置文件：ssh 无密码连接

启动服务

测试 mha 集群

把当前的主库宕机或停止数据库服务器，此时客户端依然可以访问 vip 地址连接数据库服务器

把宕机的数据库服务器添加到集群里

手动配置当前主库到从库

修改 app1.cnf 宕机的数据库的信息

测试配置并启动管理服务

##

配置 php 支持 Redis

修改配置文件

]# vim /etc/php.ini

728 行 extension_dir = "/usr/lib64/php/modules/"

730 行 extension = "redis.so"

重启服务

]# systemctl restart php-fpm

修改 redis 的主配置文件:(修改 ip 和端口号)

]# cp /etc/redis/6379.conf /root/6379.conf.bak 将默认配置文件拷贝保存

]# vim /etc/redis/6379.conf

70 行 bind 192.168.4.50 从物理接口访问

93 行 port 6350

]# ss antpu | grep redis ip 地址没有变

]# /etc/init.d/redis_6379 stop 停服务

]# /etc/init.d/redis_6379 start 起服务

]# ss -antpu | grep redis ip 地址就变成了修改后的地址了

]# redis-cli -h 192.168.4.50 -p 6350 登录

]# redis-cli -h 192.168.4.50 -p 6350 shutdown 停掉(命令可以写到脚本中)

]# vim /etc/init.d/redis_6379 命令写到脚本中

8 行 REDISPORT="6350"

43 行 $CLIEXEC -h 192.168.4.50 -p $REDISPORT shutdown

保存退出

测试一下

]# /etc/init.d/redis 6379 start[ERR] Sorry, can't connect to node 192.168.4.57:6358 用脚本起

]# /etc/init.d/redis_6379 stop 用脚本停

]# ss -ntplu | grep redis 查看是否启用

内存管理 内存清除策略

volatile-lru 最近最少使用 (针对设置了过期时间的 key)

allkeys-lru 删除最少使用的 key

volatile-random 在设置了过期的 key 里随机移除

allkeys-random 随机移除 key

volatile-ttl (minor TTL) 移除最近过期的 key

noeviction 不删除 写满时报错

选项默认设置

maxmemory <bytes> // 最大内存

maxmemory-policy noeviction // 定义使用的策略

maxmemory-samples 5 // 选取模板数据的个数(针对 lru 和 ttl 策略)

访问 redis 服务

]# ps -C redis

]# ss -utnlp | grep redis

]# redis-cli // 连接本机 redis 服务

]# vim /etc/redis/6379.conf redis 服务

常用配置选项

port 6379 // 端口
bind 127.0.0.1 //IP 地址
tcp-backlog 511 //tcp 连接总数
timeout 0 // 连接超时时间
tcp-keepalive 300 // 长连接时间
daemonize yes // 守护进程方式运行
databases 16 // 数据库个数
logfile /var/log/redis_6379.log //pid 文件
maxclients 10000 // 并发连接数量
dir /var/lib/redis/6379 // 数据库目录

设置连接密码为 123456
501 行 requirepass 123456 设置密码
]# /etc/init.d/redis_6379 stop 停服务
]# /etc/init.d/redis_6379 start 起服务
方式一
]# redis-cli -h 192.168.4.50 -p 6350 -a 123456 直接输密码
登录后验证
192.168.4.50:6350> ping
PONG
方式二
]# redis-cli -h 192.168.4.50 -p 6350
192.168.4.50:6350> auth 123456
OK
192.168.4.50:6350> ping
PONG

]# vim /etc/init.d/redis_6379　　修改配置文件

43 行　　$CLIEXEC -h 192.168.4.50 -a 123456 -p $REDISPORT shutdown

]# /etc/init.d/redis_6379 stop　停服务

]# /etc/init.d/redis_6379 start 起服务

两个知识点

搭建 Redis 服务器

服务的常用配置 ip 地址 端口 数据库位置 数据库个数

内存清除策略 设置连接密码

管理数据的常用命令

set　存

get　取

del　删

move　移动变量

flushall　删除所有变量

type　查看类型

ttl　查看生存时间

select　切换库

lnmp+Redis

##

day02

配置 Redis 集群步骤

1　准备 6 台 redis 服务器（使用默认配置运行即可）

]# yum -y install gcc gcc-c++

]# tar -zxvf redis-4.0.8.tar.gz

]# cd redis-4.0.8/

]# make

]# make install

]# ./utils/install_server.sh　(一直回车)

2 配置 redis 集群

2.1 在每台 redis 服务器上做集群配置，然后重启 redis 服务

]# netstat -utnlp | grep redis　查看是否开启 redis 服务

51-56 上操作

]# /etc/init.d/redis_6379 stop　停服务

修改配置文件配置

]# vim /etc/redis/6379.conf

修改 redis 的主配置文件:(修改 ip 和端口号)

]# /etc/init.d/redis_6379 stop　停服务

]# vim /etc/redis/6379.conf

70 行　　bind 192.168.4.50　从物理接口访问

93 行　　port 6350

829 行　cluster-node-timeout 5000　// 请求超时 5 秒

137 行　daemonize yes　　// 守护进程方式运行

815 行　cluster-enabled yes　// 开启集群

823 行　cluster-config-file nodes.conf　// 集群的配置文件不要使用默认的名称

501 行 密码(可以不做)

]# /etc/init.d/redis_6379 start　开启服务

验证

]# ss -antpu | grep redis　　查看是否启用

]# ls /var/lib/redis/6379

51 上

]# echo $PATH

]# ls redis-4.0.8/src/redis-trib.rb

]# cp redis-4.0.8/src/redis-trib.rb /usr/local/sbin/

2.2 创建集群

2.2.1 安装 ruby 脚本运行软件

]# yum -y install ruby rubygems

忽略依赖关系装

]# rpm -ivh --nodeps

ruby-devel-2.0.0.648-30.el7.x86_64.rpm

]# gem install redis-3.2.1.gem

2.2.2 安装 ruby 脚本拷贝到系统命令路径下

]# cp redis-4.0.8/src/redis-trib.rb /usr/local/sbin/

2.2.3 创建集群，保证 redis 数据库里没有数据

]# redis-trib.rb create --replicas 1 192.168.4.51:6351 192.168.4.52:6352 192.168.4.53:6353 192.168.4.54:6354 192.168.4.55:6355 192.168.4.56:6356

[OK] All 16384 slots covered. 正确输出

使用 ruby 脚本查看集群信息

]# cat /var/lib/redis/6379/nodes-6351.conf

或者

任意一台主机访问本机的 redis 服务,查看即可

redis-cli -c -h IP 地址 -p 端口

> cluster nodes # 查看本机信息

> cluster info # 查看集群信息

如果 redis 数据库里有数据，进入 redis 数据库

>flushall

>shutdown

>exit

]# rm -rf /var/lib/redis/6379/*

]# redis-cli -h ip -p 端口 shutdown

>key * 查看一下

>exit

]# /etc/init.d/redis_6379 start　开启服务

]# ss -antpu | grep redis　　查看是否启用

]# /etc/init.d/redis_6379 stop　　停止服务

##
##########################

day 03

测试集群 dis-trib.rb chec

在客户端访问任意一台主库 master 角色的 redis 服务器，查询数据或存储数据

50 机:

]# ping -c 2 192.168.4.5X　(X:52-56)

]# redis-cli -c -h 192.168.4.51 -p 6351　(-c:集群)

> set name bob

> set name lx

> set age 19

> keys *

> exit

51 和 52 上

]# redis-cli -c -h 192.168.4.52 -p 6352

> keys *

> get name

]# redis-cli -c -h 192.168.4.51 -p 6351

> keys *

> get age

把某个 master 角色主机的 redis 服务停止，对应的从库会自动升级成主库（master）

停掉 52(主库)

]# redis-cli -h 192.168.4.52 -p 6352 shutdown

用脚本启动 52

]# /etc/init.d/redis_6379 start

再次查询(52 变成了从库)

]# redis-trib.rb check 192.168.4.51:6351

解决方法：

可以给升级为主库的从库添加多个从库(至少两个以上)

问题

]# redis-cli -h 192.168.4.50 -p 6350

Could not connect to Redis at 192.168.4.50:6350:

Connection refused

Could not connect to Redis at 192.168.4.50:6350: Connection refuseddis-trib.rb chec

解决办法

]# rm -rf /var/lib/redis/6379/*

]# redis-cli -h 192.168.4.50 -p 6350 shutdown

]# /etc/init.d/redis_6379 start

]# ss -antpu | grep value is not an integer or out of rangeredis

]# redis-cli -h 192.168.4.50 -p 6350

> keys *

> flushall

OK

> quit

或者

]# killall redis-server

]# /etc/init.d/redis_6379 start

]# /etc/init.d/redis_6379 status

]# redis-cli -h 192.168.4.70 -p 6370

问题

]# /etc/init.d/redis_6379 status

cat: /var/run/redis_6379.pid: 没有那个文件或目录

Redis is running (

解决办法

]# /etc/init.d/redis_6379 start

]# killall redis-server

]# vim /etc/redis/6379.conf 改配置文件

]# /etc/init.d/redis_6379 start

]# ss -antpu | grep redis

问题

]# redis-trib.rb add-node --slave 192.168.4.57:6358 192.168.4.51:6351

.....

[ERR] Sorry, can't connect to node 192.168.4.57:6358

解决办法

]# redis-cli -h 192.168.4.57 -p 6357

> keys *

(empty list or set)

> shutdown

> quit

]# /etc/init.d/redis_6379 start

]# /etc/init.d/redis_6379 status

]# redis-trib.rb add-node --slave 192.168.4.57:6357 192.168.4.51:6351

问题

]# redis-trib.rb add-node 192.168.4.50:6350 192.168.4.51:6351

[ERR] Node 192.168.4.50:6350 is not empty. Either the node already knows other nodes (check with CLUSTER NODES) or contains some key in database 0.

解决办法：

在 50 上

]# redis-cli -h 192.168.4.50 -p 6350

> CLUSTER reset

> CLUSTER info

cluster_state:fail　　这个为 fail 的时候就可以了

在 51 上再次执行添加命令

]# redis-trib.rb add-node 192.168.4.50:6350 192.168.4.51:6351

查看主从配置

]# redis-trib.rb check 192.168.4.51:6351

问题

]# /etc/init.d/redis_6379 start

/var/run/redis_6379.pid 95bde :0@0 slave,noaddr c8f519ac53c506ab2207994366f2ac3465dfd86d 15308exists, process is already running or crashed

解决办法

]# ls /var/run/redis_6379.pid

/var/run/redis_6379.pid

]# cat /var/run/redis_6379.pid

7524

]# rm -rf /var/run/redis_6379.pid

]# /etc/init.d/redis_6379 start

Starting Redis server...

]# /etc/init.d/redis_6379 status

Redis is running (8990)

]# ss -antpu | grep redis

]# rm -rf /var/lib/redis/6379/*

]# /etc/init.d/redis_6379 stop

]# /etc/init.d/redis_6379 start

]# redis-cli -h 192.168.4.70 -p 6370

95bde :0@0 slave,noaddr
c8f519ac53c506ab2207994366f2ac3465dfd86d
15308

++++++++++++++++++++++++++++++++++++
++++++++++++

管理集群

redis-cli 命令

redis-cli -h 查看命令帮助

-h IP 地址

-p 端口

-c 访问集群

]# redis-cli -h 192.168.4.51 -c -p 6351

> cluster nodes # 查看本机信息

> cluster info # 查看集群信息

redis-trib.rb 脚本

语法格式

Redis-trib.rb 选项 参数 (参数: ip:端口号)

选项

add-node 添加新节点

check 对节点主机做检查

reshard 对节点主机重新分片

add-node --slredis-trib.rb add-node 192.168.4.50:6350 192.168.4.51:6351ave 添加从节点主机

del-node 删除节点主机

在 51 上检测主从

]# redis-trib.rb check 192.168.4.51:6351 (后面可以连任意的主机 ip 和端口)

管理集群 redis-cli

向集群里添加系统 redis 服务器 50 和 57

运行 redis 服务 清楚已有的数据 做启用集群配置后重启服务

添加 mastre 角色主机

]# redis-trib.rb add-node 192.168.4.50:6350 192.168.4.51:6351

]# redis-trib.rb check 192.168.4.51:6351 查看主从库

]# redis-trib.rb reshard 192.168.4.51:6351

1 从当前的集群中拿出多少个 hash slots 4096 (16384/4)

2 新主库的 id 值

3 all 从当前

4 yes

添加 salve 角色主机 95bde :0@0 slave,noaddr c8f519ac53c506ab2207994366f2ac3465dfd86d 15308

57 上运行 redis 服务 清楚已有的数据 做启用集群配置后重启服务

]# redis-trib.rb add-node --slave 192.168.4.57:6357 192.168.4.51:6351

]# redis-trib.rb check 192.168.4.51:6351 查看主从库

移除节点(主从库)

移除主节点(主库)

1) 删除槽位

]# redis-trib.rb reshard 192.168.4.51:6351

4096　　移除的槽数

1 移出多少个 hash slots

2 主库的 id　　移动到哪个主节点上(给谁)

3 主库的 id　　从哪个主节点上移除（从哪拿）

4 done　　手动输入 done

5 yes　　输入 yes

2) 移除主节点

]# redis-trib.rb del-node 192.168.4.51:6351 被移除节

点主机 id

移除从节点(slave)

从节点主机没有槽位范围,直接执行移除命令即可

1) 移除主节点

]# redis-trib.rb del-node 192.168.4.51:6351 被移除

主机的 ID

##

答疑

表级锁　　锁表　myisam

行级锁　　给每一行加锁　　innodb

关系型数据库软件　　oracle　sql server　db2
mysql postgrpsql

非关系型数据库软件 memcached　redis　mongodb

##
##########################

redis 主从复制

redis 持久化

数据类型

redis 主从复制

支持模式

一主一从、一主多从、主从从

主从复制工作原理

1、Slave 向 maste 发送 sync 同步命令

2、Master 启动后台存盘进程,同时收集所有修改数据命令

3、Master 执行完后台存盘进程后,传送整个数据文件到 slave

4、Slave 接收数据文件后,将其存盘并加载到内存中完成首次完全同步

5、后续有新数据产生时, master 继续将新的所以收集到的修改命令依次传给 slave ,完成同步

主从复制缺点

网络繁忙,会产生数据同步延时问题

系统繁忙,会产生数据同步延时问题

配置主从复制

一主一从模式：

配置从库

> info replication // 查看主从配置信息

> slaveof 主库 IP 地址 主库端口号

如何那当前的从库恢复成主库

> slaveof on one

设置了连接的密码：

redis=cli -h ip 地址 -p 端口号 -a 密码

数据正常同步

]# vim /etc/redis/6379.conf

282 行 slaveof 192.168.4.50 6350

289 行 masterauth 123456

哨兵模式

主库宕机后,从库自动升级为主库

在 slave 主机编辑 sentinel.conf 文件

在 slave 主机运行哨兵程序

]# vim /etc/sentinel.conf

sentinel monitor redis50 192.168.4.50 6350 1

主 机 名 ip 地址 端口 票数

主机名:自定义

IP 地址: master 主机的 IP 地址

端口: master 主机 redis 服务使用的端口

票数:主库宕机后，票数大于 1 的主机被升级为主库

##
##########################

配置从库

> info replication // 查看主从配置信息

> slaveof 主库 IP 地址 主库端口号

修改 redis 的主配置文件:(修改 ip 和端口号)

]# /etc/init.d/redis_6379 stop 停服务

修改配置文件配置

]# vim /etc/redis/6379.conf

70 行 bind 192.168.4.50 从物理接口访问(改 ip)

93 行 port 6350 改端口

829 行 cluster-node-timeout 5000 // 请求超时 5 秒

137 行 daemonize yes // 守护进程方式运行

823 行 cluster-config-file nodes.conf // 集群的配置文件不要使用默认的名称

501 行 密码(可以不做)

]# /etc/init.d/redis_6379 start 开启服务

验证

]# ss -antpu | grep redis 查看是否启用

]# ls /var/lib/redis/6379

配置哨兵模式

实现效果：redis 服务器中 主库宕机后从库自动成为主库

> slaveof on one 主库宕了后，手动将从库切换成主库(临时的)

50 上

关闭服务并修改配置文件后开启服务

]# /etc/init.d/redis_6379 stop

vim /etc/redis/6379.conf

#requirepass 123456 去掉注释

#masterauth 123456 去掉注释

cluster-enabled yes 加上注释？

]# /etc/init.d/redis_6379 start

]# /etc/init.d/redis_6379 status

57 上做哨兵模式

]# /etc/init.d/redis_6379 stop

]# vim /etc/redis/6379.conf

#masterauth 123456 去掉注释

slaveof 192.168.4.50 6350 添加这个

]# /etc/init.d/redis_6379 start

]# rm -rf /etc/sentinel.conf

]# vim /etc/sentinel.conf

sentinel monitor redis50 192.168.4.50 6350 1

主 库 主 机 名 ip 地址 端口 票数

sentinel auth-pass redis50 123456 加上连接密码(设了密码)

]# redis-sentinel /etc/sentinel.conf 开启服务

50 上

关闭服务

/etc/init.d/redis_6379 stop

57 上查看从库就成为主库了

~]# redis-cli -h 192.168.4.57 -p 6357

> info replication

>role:master

把 57 和 50 恢复成独立的数据库服务器

50 上

关闭服务

]# /etc/init.d/redis_6379 stop

57 上

]# /etc/init.d/redis_6379 stop

]# vim /etc/redis/6379.conf
#masterauth 123456　　加上注释
]# /etc/init.d/redis_6379 start

50 上
开启服务
]# /etc/init.d/redis_6379 start

重新配置从库（57 上）
配置从库
> info replication // 查看主从配置信息
> slaveof 主库 IP 地址　主库端口号

rm -rf /etc/sentinel.conf
vim /etc/sentinel.conf
sentinel monitor redis50 192.168.4.50 6350　1
redis-sentinel /etc/sentinel.conf

50 上
关闭服务
/etc/init.d/redis_6379 stop

57 上

]# redis-cli -h 192.168.4.57 -p 6357

> info replication

>role:master

##

redis 持久化

的方式　AOF 和 RDB

RDB　数据持久化方式之一

在指定时间间隔内,将内存中的数据集快照写入硬盘

术语叫 Snapshot 快照

恢复时,将快照文件直接读到内存里

备份 dump.rdb 文件到其他位置

]# cp 数据库目录 /dump.rdb　　备份目录

恢复数据

把备份的 dump.rdb 文件拷贝回数据库目录 , 重启 redis 服务

cp 备份目录 /dump.rdb　数据库目录 /

/etc/redid/redis_ 端口 start

RDB 优点

持久化时, Redis 服务会创建一个子进程来进行持久化,会先将数据写入到一个临时文件中,待持久化过

程都结束了,再用这个临时文件替换上次持久化好的
文件;整个过程中主进程不做任何 IO 操作,这就确保
了极高的性能　如果要进程大规模数据恢复,且对数据完整行要求不
是非常高,使用 RDB 比 AOF 更高效

RDB 的缺点
意外宕机,最后一次持久化的数据会丢失

AOF
只追加操作的文件
记录 redis 服务所有写操作
不断的将新的写操作,追加到文件的末尾。
使用 cat 命令可以查看文件内容

启用 aof 文件对数据做持久化步骤:
默认没起
1、启用 50 主机上的 aof 文件对数据做持久化
]# vim /etc/redis/6379.conf
appendonly yes　　改为 yes

2、连接 50 主机 存储数据然后查看 aof 文件的内容

3、拷贝 aof 文件到系统的/root 目录下

4、删除 50 主机上的所有数据，使用备份的 aof 文件恢复

数据

5、连接 50 书记，查看数据是否恢复成功

修复 AOF 文件,把文件恢复到最后一次的正确操作

]# redis-check-aof --fix appendonly.aof

日志重写（日志文件会不断增大）,何时会触发日志重写?

redis 会记录上次重写时 AOF 文件的大小,默认配置是当 aof 文件是上次 rewrite 后大小的 1 倍且文件大于 64M 时触发。

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

##

redis 数据类型

string

list

hash

搭建 redis 服务器

redis 服务的基本使用

lnmp+redis

redis 集群（解决访问节点故障 同时实现数据的自动备份）

管理集群：向集群里添加和删除（master slave）

部署 redis 主从同步结构

从主库上启用哨兵模式

redis 服务数据持久化 RDB AOF

redis 数据类型 字符 set get setbit setcount decr incr incrby

list 列表

定义

输出值 lrange namegrp 0 -1

删除值 lpop namegrp 删除第一个 rpop namegrp 删除最后个

添加新值 LPUSH namegrp qq

hash 表

让定义变量可以存储多个 key/values key/values

> hgetall site 输出值和列名

> hkeys site 输出列名

> hmget site yahu qq lx baidu 360 sina 输出值

> hset site sina www.sina.com.cn 添加列名和值

> hmset site yahu www.yahu.com qq www.qq.com lx www.lx.com 连续添加值和列名

> type site 查看类型

> hdel site 360 yahu 连续删除列名

##
##########################

在 50 主机上部署 mongodb 数据库服务

配置步骤：

1、 装包

]# mkdir /usr/local/mongodb

]# tar -zxf mongodb-linux-x86_64-rhel70-3.6.3.tgz

]# cp -r mongodb-linux-x86_64-rhel70-3.6.3/bin /usr/local/mongodb/

]# cd /usr/local/mongodb/

]# mkdir etc log

]# mkdir -p data/db　　这个目录名称必须是 data/db

2、修改配置文件

]# cd bin

]# ./mongod --help　查看帮助

]# cd ..

]# vim etc/mongodb.conf

dbpath=/usr/local/mongodb/data/db

fork=true

logpath=/usr/local/mongodb/log/mongodb.log

logappend=true

保存退出

键命令加入配置文件中做成全局变量(中间以：分隔)

]# vim /etc/profile

export PATH=/usr/local/mongodb/bin:/usr/local/mysql/bin:$PATH

测试命令,启用命令

]# source /etc/profile

3、启动服务

]# mongod -f /usr/local/mongodb/etc/mongodb.conf

about to fork child process, waiting until server is ready for connections.

forked process: 2002

child process started successfully, parent exiting

输出这个就成功了

查看这里面就会有文件

]# ls data/db

查看进程

]# ps -C mongod

查看端口

]# ss -antpu | grep :27017

4、登录 mongodb 数据库

]# mongo

退出

> exit

更改 ip 和端口号

]# killall -9 mongod

或者

]# mongod --shutdown -f /usr/local/mongodb/etc/mongodb.conf

]# vim etc/mongodb.conf

bind_ip=192.168.4.50

port=27050

起服务

]# mongod -f /usr/local/mongodb/etc/mongodb.conf

]# ss -antpu | grep mongod

登录

]# mongo --port 27050 --host 192.168.4.50

永久定义别名(etc/mongodb.conf 加个 755 的权限重起 50 主机)

]# vim /root/.bashrc

alias mstar='mongod -f /usr/local/mongodb/etc/mongodb.conf'

alias mstop='mongod --shutdown -f /usr/local/mongodb/etc/mongodb.conf'

常用命令

>show dbs 查看已有的库

>db 显示当前所在的库

>use 库名 切换库，若库不存在延时创建库

>show tables 查看库下已有集合

>db.dropDatabase() 删除当前所在的库

>db.集合名.drop() 删除集合

>db.集合名.save({","}) 创建集合，集合不存在时创建并添加文档

> db.c1.save({name:"lx",age:19,sex:"boy"})

> db.c1.find() 查看集合

> db.c1.count() 查看集合的行数

>db.集合名.findOne() 返回一条文档

>db.集合名.remove({}) 删除所有文档

>db.集合名.save({条件}) 删除匹配的文档

管理数据库

集合管理

文档管理 查看 插入 删除 修改

##

mongodb 数据类型

字符 数值 布尔 数组 空

> db.c1.save({name:"bob",age:19,sigle:true})

>
db.c1.save({name:"lucy",age:18,sigle:true,pay:null})

> db.c1.save({name:"bob",x:NumberInt(3)})

> db.c1.save({name:"tom",x:NumberInt(3.99)})

> db.c1.save({name:"jack",x:3.99})

>
db.c1.save({name:"yaya",bboy:["pyy","lyf","ssa"]})

> db.c1.find()

代码类型

格式：{x:function(){/*代码*/}}

>

```
db.c1.save({lname:"php",codeformat:function(){}
/*<?php echo "hello world" ?> */})
> db.c1.find({lname:"php"})
```

日起类型

格式：{x:new Date()}

```
>              db.c1.save({name:"haha",birthday:new
Date()})
> db.c1.find({name:"haha"})
```

对象类型

格式{x:ObjectId()}

```
> db.c1.save({name:"alice",stuid:ObjectId()})
> db.c1.find({name:"alice"})
```

内嵌类型

文档可以嵌套其他文档，被嵌套的文档作为值来处理

```
>
db.c1.save({ywzd:{p:"dmy",jg:69,v:2},nqsfc:{p:"l
x",jg:100,v:1}})
> db.c1.find()
```

正则表达式

查询时，使用正则表达时作为限定条件

格式：{x:/正则表达时/}

```
> db.c1.save({name:"qq",match:/^a/})
```

数据导入导出

数据备份恢复

数据导出　mongoexport --help

格式 1

mongoexport [--host IP 地址 --port 端口号] -d 库名 -c 集合名 -f 字段名 1,字段名 2 --type=csv > 目录名/文件名.csv

格式 2

mongoexport --host IP 地址 --port 端口号　库名 -c 集合名　-q '{条件}' -f 字段名 1,字段名 2 --type=csv > 目录名/文件名.csv

格式 3

mongoexport [--host IP 地址 --port 端口号] -d 库名 -c 集合名　[-q '{条件}' -f 字段列表]　--type=json > 目录名/文件名.json

例

json 格式：

]# mkdir /mongdbdir

]# mongoexport --host 192.168.4.50 --port 27050 -d studb -c c1 --type=json > /mongdbdir/c1.json

]# cat /mongdbdir/c1.json

csv 格式：

]# mongoexport --host 192.168.4.50 --port 27050 -d studb -c c1 -f id,name --type=csv > /mongdbdir/c1.csv

]# cat /mongdbdir/c1.csv

数据导入 mongoimport --help

格式 1

mongoimport --host IP 地址 --port 端口号 -d 库名 -c 集合名 --type=json 目录名/文件名.json

格式 2

mongoimport -host IP 地址 -port 端口号 -d 库名 -c 集合名 --type=csv [--]# killall -9 mongod

或者

]# mongod --shutdown -f /usr/local/mongodb/etc/mongodb.conf

]# vim etc/mongodb.conf

bind_ip=192.168.4.50

port=27050

起服务

]# mongod -f /usr/local/mongodb/etc/mongodb.conf

]# ss -antpu | grep mongodheaderline] [--drop] 目录名/文件名.csv

--drop 可以删除原数据后导入新数据

--headerline 忽略标题

例

json 格式：

]# mongoimport --host 192.168.4.50 --port 27050 -d bbsdb -c t1 --type=json /mongdbdir/c1.json

csv 格式：

]# mongoimport --host 192.168.4.50 --port 27050 -d bbsdb -c t2 -f id,name --type=csv /mongdbdir/c1.csv

csv 格式：

]# mongoimport --host 192.168.4.50 --port 27050 -d studb -c c1 --headerline --drop --type=csv /mongdbdir/c1.csv

创建集合并写入值

>db.c3.save({name:"yaya",password:"x",uid:88888,gid:99999,

comment:"teacher",homedir:"/home/yaya",

shell:"/bin/bash"})

导出文件:

]# mongoexport --host 192.168.4.50 --port 27050 -d studb -c c3 -f name,passwo]# killall -9 mongod

或者

]# mongod --shutdown -f /usr/local/mongodb/etc/mongodb.conf

]# vim etc/mongodb.conf

bind_ip=192.168.4.50

port=27050

起服务

]# mongod -f /usr/local/mongodb/etc/mongodb.conf

]# ss -antpu | grep mongodrd,uid,gid,comment,homedir,shell --type=csv > /mongdbdir/c3.csv

查看导出的文]# killall -9 mongod

或者

]# mongod --shutdown -f /usr/local/mongodb/etc/mongodb.conf

]# vim etc/mongodb.conf

bind_ip=192.168.4.50

port=27050

起服务

]# mongod -f /usr/local/mongodb/etc/mongodb.conf

]# ss -antpu | grep mongod 件

]# cat /mongdbdir/c3.csv

]# cd /mongdbdir/

]# cp /etc/passwd ./

]# sed -i '' c3.csv

]# sed -i '$r password' c3.csv

]# sed -i 's/:/,/g' c3.csv 将冒号替换为分号

导入数据

]# mongoimport --host 192.168.4.50 --port 27050 -d studb -c c3 --headerline --drop --type=csv /mongdbdir/c3.csv

数据备份

备份数据所有库到当前目录下的 dump 目录下

mongodump [--host ip 地址 --port 端口]

备份时指定备份的库和备份目录

mongodump [--host ip 地址 --port 端口] -d 数据库名 -c 集合名 -o 目录

查看 bson 文件内容

bsondump ./dump/bbs/t1.bson

数据恢复

格式

mongorestore --host ip 地址 --port 端口 -d 数据库名 [-c 集合名] 备份目录名

案例：备份 studb 库中的 c3 集合中的数据

备份

]# mkdir /bakmongoo

]# mongodump --host 192.168.4.50 --port 27050 -d studb -c c3 -o /bakmongo

]# ls

]# bsondump /bakmongo/studb/c3.bson

删除数据

mongo --host 192.168.4.50 --port 27050

>use studb

>db.c3.remove({})

恢复 db.c2.find()

]# mongoswitched to db haharestore --host 192.168.4.50 --port 27050 -d studb -c c3 /bakmongo/studb/c3.bson

查看数据

mongo --host 192.168.4.50 --port 27050

>use studb

>db.c3.find({})

> db.c3.count()

##

day 05

mongodb 副本集

文档管理

mongodb 副本集

也称为 mongodb 复制，指在多个服务器上存储数据副本，并实现数据同步，提高数据可用性，安全性，方便数据恢复

副本集工作过程

至少需要两个节点，其中一个时主节点，负责处理客户点请求，其余时从节点，负责复制主节点数据

常见搭配方式：一主一从、一主多从

主节点记录所有操作 poloq，从节点定期轮询主节点获取这些操作，然后对自己的数据副本执行这些操作，从而保证从节点的数据与节点一致

部署 mongodb 副本集

配置 replica sets

51-53 添加

]# killall -9 mongod

或者

]# mongod --shutdown -f /usr/local/mongodb/etc/mongodb.conf

vim /usr/local/mongodb/etc/mongodb.conf

replSet=rs1

起服务

]# mongod -f

/usr/local/mongodb/etc/mongodb.conf

]# ss -antpu | grep mongod

创建集群：(51 上)

]# mongo --port 27051 --host 192.168.4.51

>config = { _id:"rs1",members:[{ _id:0,host:"192.168.4.51:27051"},{ _id:1,host:"192.168.4.52:27052"},{ _id:2,host:"192.168.4.53:27053"}]};

db.c2.find()

创建集群

>rs.initiate(config)

查看状态信息

rs1:SECONDARY> rs.status()

查看是否时 master 库(主库)

rs1:SECONDARY> rs.isMaster()

同步数据验证，允许从库查看数据

>db.getMongo().setSlaveOk()

自动切换主库验证

>rs.isMaster()

测试

主库 51：

]# mongod --shutdown -f /usr/local/mongodb/etc/mongodb.conf

50：

mongo --host 主库 IP --port 主库端口

> 建库 建集合 插入文 db.c2.find()档

把主机 51-52 恢复成独立的数据库服务器

删除配置文件中的集群配置，重起服务

]# mongod --shutdown -f /usr/local/mongodb/etc/mongodb.conf

]# vim /usr/local/mongodb/etc/mongodb.conf

replSet=rs1 删除

起服务

]# mongod -f /usr/local/mongodb/etc/mongodb.conf

]# ss -antpu | grep mongod

文档管理(查询 插入 更新 删除)

插入

save()

格式

>db.集合名.save({key:"值"，key:"值"})

insert()

格式

>db.集合名.insert({key:"值"，key:"值"})

插入多条记录

>db.集合名.insertMany([{name:"tom",age:19},{name:"natasha",email:"natasha@163.com"}])

注意：

集合不存时创建集合，然后在插入记录

_id 字段值已存在时，修改文档字段值

_id 字段值不存在时，插入文档

>db.c5.insert({_id:17,name:"haha"})

>db.c6.save({_id:17,name:"haha"})

> db.c5.find()

> db.c6.find()

>
db.c6.insertMany([{name:"zs",age:19,pay:20000},{name:"lz",school:"nsd1803"}])

> db.c6.find()

查询

显示所有行(默认输出 20 行，输入 it 可显示后续行)

>db.集合名.find()

显示第一行

>db.集合名.findOne()

指定查询条件并指定显示的字段

>db.集合名.find({条件},{定义显示的字段})

0　不显示　　1　显示

>db.user.find({},{_id:0,name:1,shell:1})

行数显示限制

显示前 3 行

limit(数字)

>db.集合名.find().limit(3)

跳过前 2 行

skip(数字)

>db.集合名.find().skip(2)

升\降序

sort(字段名)

>db.集合名.find().sort(age:1|-1)

综合

>db.user.find({shell:"/sbin/nologin"},{ id:0,name:1,uid:1,shell:1}).skip(2).limit(2)

匹配条件

简单条件

>db.集合名.find({key:"值"})

>db.集合名.find({key:"值"，keyname:"值"})

>db.user.find({shell:"/bin/bash"})
>db.user.find({shell:"/bin/bash"，name:"root"})

范围比较
$in　　在...里
$nin　　不在...里
$or　　或
>db.user.find({uid:{$in:[1,6,9]}})
>db.user.find({uid:{$nin:[1,6,9]}})
>db.user.find({$or:[{name:"root"},{uid:1}]})

正则匹配
>db.user.find({name:/^a/})

数值比较
$lt　　<
$lte　　<=
$gt　　>
$gte　　>=
$ne　　!=
>db.user.find({uid:{$gte:10.$lte:40}},{ id:0,name:1,uid:1})
>db.user.find({uid:{$lte:5}})

匹配 null,也可以匹配没有的字段

>db.user.save({name:null,uid:null})

>db.user.find({" id":ObjectId("5afdOddbd42772e7e458fc75"),"name":null,"uid":null})

更新文档

update()

语法格式

>db.集合名.update({条件},{修改的字段})

>db.user3.find({uid:{$lte:3}},{_id:0})

>db.user3.update(uid:{$lte:3}},{password:"888"})

>db.user3.find({uid:{$lte:3}},{_id:0})

多文档更新

语法格式:默认只更新与条件匹配的第 1 行

>db.user.update({条件},{$set:{修改的字段}}，false,ture)

>db.user.update({name:"bin"},{$set:{password:"abc123456"}}，false,ture)

$set/unset

$set 条件匹配时，修改指定字段的值

>db.user.update({条件},$set:{修改的字段}）

>db.user.update({name:"bin"},{$set:{password:"A"}}）

$unset 删除与条件匹配文档的字段

>db.集合名.update({条件},{$unset:{key:values}})

>db.user.update({name:"bin"},{$unset:{password:"A"}})

$inc

$inc 条件匹配时，字段值自加或自减

>db.集合名.update({条件},{$inc:{字段名:数字}})

字段值自加 2

>db.user.update({name:"bin"},{$inc:{uid:2}})

字段值自减 1

>db.user.update({name:"bin"},{$inc:{uid:-1}})

$push/$addToSet

$push 向数组中添加新元素

>db.集合名.update({条件},{$push:{数组名:"值"}})

>db.user.insert({name:"bob",likes:["a","b","c","d","e","f"]})

>db.user.find()

>db.user.update({name:"bob"},{$push:{likes:"w"}})

>db.user.find()

$addToSet 避免重复添加

>db.集合名.update({条件},{$addToSet:{数组名:"值"}})

>db.user.update({name:"bob"},{$addToSet:{likes:"f"}})

$pop/$pull

$pop 从数组头部删除一个元素

>db.集合名.update({条件},{$pop:{数组名:"值"}})

1 删除数组尾部元素

-1 删除数组头部元素

>db.user.update({name:"bob"},{$pop:{likes:"1"}})

>db.user.update({name:"bob"},{$pop:{likes:"-1"}})

$pull 删除数组指定元素

>db.集合名.update({条件},{$pull:{数组名:"值"}})

>db.uesr.update({name:"bob"},{$pull:{likes:"b"}})

删除文档

$drop/$remove

$drop 删除集合的同时删除索引

>db.集合名.drop()

>db.user.drop()

文档更新命令总结

$set	修改文档指定字段的值
$unset	删除记录中的字段
$push	向数组内添加新元素

$pull 删除数组中的指定元素
$pop 删除数组头尾部元素
$addToSet 避免数组重复赋值
$inc 字段自加或自减

$remove() 删除文档时不删除索引
>db.集合名.remove({}) 删除所有文档
>db.集合名.remove({条件}) 删除与条件匹配的文档

>db.user.remove({uid:{$lte:10}})
>db.user.remove({})

##
##########################
答疑
生产环境中所用到的数据库
mysql redis mongodb oracle sql-server
优化的思路
软优化(配置参数)
缓存相关的参数
进公司通常要割接
]#git clone
https://github.com/MrZhangzhg/nsd2018.git
]# cd nsd2018/
]# git pull