# 老男孩linux运维实战网络培训-周斌-MySQL

老男孩老师教学与培训核心思想：重目标、重思路、重方法、重实践、重习惯、重总结。

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

本文内容来自《老男孩linux运维实战培训》学生—周斌

如有转载，请务必保留本文链接及本内容版权信息。

欢迎广大运维同仁一起交流linux/unix网站运维技术!

QQ:3234548

E-mail:zbxm2000@126.com

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

老男孩linux运维实战培训中心

**咨询QQ:41117397 41117483 70271111**

电话: 01060747396 18911718229 18600338340

官方群 08群384467551 07群145178854(标明51CTO)  
#####################################################

MySQL数据库整理

*目录*

[老男孩linux运维实战网络培训-周斌-MySQL 1](#_Toc439085038)

[1 MySQL介绍 5](#_Toc439085039)

[1.1什么是MySQL 5](#_Toc439085040)

[1.2 MySQL发展历程 5](#_Toc439085041)

[2 MySQL安装 7](#_Toc439085042)

[2.1 MySQL之二进制包安装 7](#_Toc439085043)

[2.2 MySQL之源码包编译安装 10](#_Toc439085044)

[2.3 MySQL之yum源直接安装 12](#_Toc439085045)

[2.4 MySQL之源码rpm打包 13](#_Toc439085046)

[3 MySQL多实例 14](#_Toc439085047)

[3.1多实例介绍 14](#_Toc439085048)

[3.2多实例应用场景 14](#_Toc439085049)

[3.3多实例相关面试题 14](#_Toc439085050)

[3.4多实例配置思路 15](#_Toc439085051)

[3.5多实例具体配置步骤 15](#_Toc439085052)

[3.6多实例启动、登录、关闭命令 29](#_Toc439085053)

[3.7 FAQ 29](#_Toc439085054)

[4 MySQL主从复制 29](#_Toc439085055)

[4.1文件级别异机同步（共享）方案 30](#_Toc439085056)

[4.2 MySQL主从复制架构 30](#_Toc439085057)

[4.3主从复制原理 31](#_Toc439085058)

[4.4主从复制配置 32](#_Toc439085059)

[4.5主从复制配置FAQ 37](#_Toc439085060)

[4.6 MySQL主从从配置 37](#_Toc439085061)

[4.7 MySQL主主模式配置 38](#_Toc439085062)

[4.8主从复制企业应用场景 39](#_Toc439085063)

[4.9 MySQL读写分离方案 41](#_Toc439085064)

[4.10主从库不同步的原因 42](#_Toc439085065)

[4.11监测主从同步 42](#_Toc439085066)

[4.12数据库集群的授权 44](#_Toc439085067)

[4.13 MySQL高可用之角色切换原理 46](#_Toc439085068)

[5 MySQL程序相关 48](#_Toc439085069)

[5.1数据库启动命令 48](#_Toc439085070)

[5.2数据库关闭命令 48](#_Toc439085071)

[5.3设置MySQL命令提示符 49](#_Toc439085072)

[5.4修改密码的几种方法 50](#_Toc439085073)

[5.5忘记root密码如何修改 50](#_Toc439085074)

[5.6如何做好MySQL安全措施 51](#_Toc439085075)

[6 MySQL SQL语句基础 52](#_Toc439085076)

[6.1 DDL数据定义语言 52](#_Toc439085077)

[6.2 DCL数据控制语言 67](#_Toc439085078)

[6.3 DML数据操作语言 69](#_Toc439085079)

[6.4 MySQL 索引 76](#_Toc439085080)

[7 MySQL字符集 90](#_Toc439085081)

[7.1什么是字符集 90](#_Toc439085082)

[7.2 MySQL字符集查看 91](#_Toc439085083)

[7.3 MySQL字符集乱码原因 91](#_Toc439085084)

[7.4如何更改数据库表字符集 93](#_Toc439085085)

[8 MySQL日志系统 96](#_Toc439085086)

[8.1错误日志 96](#_Toc439085087)

[8.2普通查询日志 97](#_Toc439085088)

[8.3慢查询日志 97](#_Toc439085089)

[8.4二进制日志 98](#_Toc439085090)

[8.5binlog的三种模式 99](#_Toc439085091)

[8.6关于日志切割 101](#_Toc439085092)

[9全量备份以及增量备份 103](#_Toc439085093)

[9.1增量工具mysqlbinlog主要用法 103](#_Toc439085094)

[9.2企业备份方案 103](#_Toc439085095)

[10 MySQL 引擎 108](#_Toc439085096)

[10.1什么是数据库引擎 108](#_Toc439085097)

[10.2 MySQL引擎的类型 109](#_Toc439085098)

[10.3 MyISAM 引擎 110](#_Toc439085099)

[10.4 Innodb 引擎 112](#_Toc439085100)

[10.5修改MySQL数据引擎 114](#_Toc439085101)

[10.6 MySQL 事务 114](#_Toc439085102)

[10.7关闭自动提交 115](#_Toc439085103)

**1 MySQL介绍**

**1.1什么是MySQL**

MySQL是一个关系型数据库管理系统,由瑞典MySQL AB公司开发,目前属于Oracle旗下公司。MySQL最流行的关系型数据库管理系统,在WEB应用方面MySQL是最好的 RDBMS(Relational Database Management System,关系数据库管理系统）应用软件之一。MySQL是一种关联数据库管理系统,关联数据库将数据保存在不同的表中,而不是将所有数据放在一个大仓库内,这样就增加了速度并提高了灵活性。MySQL所使用的SQL语言是用于访问数据库的最常用标准化语言。MySQL软件采用了双授权政策（本词条“授权政策”),它分为社区版和商业版,由于其体积小、速度快、总体拥有成本低,尤其是开放源码这一特点,一般中小型网站的开发都选择MySQL作为网站数据库。由于其社区版的性能卓越,搭配PHP和Apache可组成良好的开发环境。

**1.2 MySQL发展历程**

MySQL的历史最早可以追溯到1979年,有一个人叫Monty Widenius,为一个叫TcX的小公司打工,并用BASIC设计了一个报表工具,可以在4M主频和16KB内存的计算机上运行。过了不久,又将此工具使用C语言重写,移植到Unix平台,当时,它只是一个很底层的面向报表的存储引擎。这个工具叫做Unireg。

1985年,瑞典的几位志同道合小伙子（以David Axmark为首）成立了一家公司,这就是MySQL AB的前身。这个公司最初并不是为了开发数据库产品,而是在实现他们想法的过程中,需要一个数据库。他们希望能够使用开源的产品。但在当时并没有一个合适的选择,没办法,那就自己开发吧。

在最初,他们只是自己设计了一个利用索引顺序存取数据的方法,也就是ISAM (Indexed Sequential Access Method)存储引擎核心算法的前身,利用ISAM结合mSQL来实现他们的应用需求。在早期,他们主要是为瑞典的一些大型零售商提供数据仓库服务。在系统使用过程中,随着数据量越来越大,系统复杂度越来越高,ISAM和mSQL的组合逐渐不堪重负。在分析性能瓶颈之后,他们发现问题出在mSQL上面。不得已,他们抛弃了mSQL,重新开发了一套功能类似的数据存储引擎,这就是ISAM存储引擎。大家可能已经注意到他们当时的主要客户是数据仓库,应该也容易理解为什么直至现在,MySQL最擅长的是查询性能,而不是事务处理（需要借助第三方存储引擎)。

1990年,TcX的customer中开始有人要求要为它的API提供SQL支持,当时,有人想到了直接使用商用数据库算了,但是Monty觉得商用数据库的速度难令人满意。于是,他直接借助于mSQL的代码,将它集成到自己的存储引擎中。但不巧的是,效果并不太好。于是,Monty雄心大起,决心自己重写一个SQL支持。

1996年,MySQL 1.0发布,在小范围内使用。到了 96年10月,MySQL 3.11.1发布了,没有2.x版本。最开始,只提供了 Solaris下的二进制版本。一个月后,Linux版本出现了。此时的MySQL还非常简陋,除了在一个表上做一些Insert,Update, Delete和Select操作之外,没有其他更多的功能。

紧接下来的两年里,MySQL依次移植到各个平台下。它发布时,采用的许可策略,有些与众不同:允许免费商用,但是不能将MySQL与自己的产品绑定在一起发布。如果想一起发布,就必须使用特殊许可,意味着要花银子。当然,商业支持也是需要花银子的。其它的,随用户怎么用都可以。这种特殊许可为MySQL带来了一些收入,从而为它的持续发展打下了良好的基础。

1999-2000年,有一家公司在瑞典成立了,叫MySQL AB。雇了几个人,与Sleepycat合作,开发出了Berkeley DB引擎,因为BDB支持事务处理,所以,MySQL从此开始支持事务处理了。

在2000年的时候,MySQL公布了自己的源代码,并采用GPL(GNU General Public License)许可协议,正式进入开源世界。

2000年4月，MySQL对旧的存储引擎进行了整理,命名为MyISAM。

2001年,HeikikiTuuri向MySQL提出建议,希望能集成他们的存储引擎InnoDB,这个引擎同样支持事务处理,还支持行级锁。所以在2001年发布的3.23版本的时候,该版本已经支持大多数的基本的SQL操作,而且还集成了MyISAM和InnoDB存储引擎。MySQL与InnoDB的正式结合版本是4.0。

2004年10月,发布了经典的4.1版本。2005年10月,又发布了里程碑的一个版本,MySQL 5.0在5.0中加入了游标,存储过程,触发器,视图和事务的支持。在5.0之后的版本里,MySQL明确地表现出迈向高性能数据库的发展步伐。

2008年1月16号MySQL被Sun公司收购。

2009年04月20日Oracle收购Sun公司,MySQL转入Oracle门下。

2010 年04月22发布 MySQL 5.5,MySQLcluster 7.1。

**2 MySQL安装**

MySQL有多种安装方式,有二进制包直接安装,源码包编译安装,yum源直接安装,以及基于源码包制作的rpm包安装。

**2.1 MySQL之二进制包安装**

### 2.1.1准备安装包

从官网下载二进制包,选择社区MySQL 5.5版本,这里是下载链接：

http://cdn.mysql.com/Downloads/MySQL-5.5/mysql-5.5.45-linux2.6-x86\_64.tar.gz

### 2.1.2解压到指定规范目录并创建软连接

软件包存放目录:/server/tools/

程序安装目录:/application/

# cd /server/tools

# tar xf mysql-5.5.45-linux2.6-x86\_64.tar.gz

# mv mysql-5.5.45-linux2.6-x86\_64 /application/mysql-5.5.45

# ln -s /application/mysql-5.5.45 /application/mysql

### 2.1.3准备数据目录

用于存放MySQL数据库文件**。**

# mkdir /data/ -p

### 2.1.4创建mysql用户并授权

# useradd mysql -s /sbin/nologin -M

# chown -R mysql:mysql /mydata/data /application/mysql-5.5.45

### 2.1.5运行初始化脚本

初始化脚本的作用是初始化mysql这个数据库,里面包含了一些数据库信息和授权表等。

# /application/mysql/scripts/mysql\_install\_db -basedir=/application/mysql --user=mysql --datadir=/data

### 2.1.6提供配置文件

MySQL会检查位置的配置文件,首先检查/etc/my.cnf,所以我们将mysql提供的配置文件cp到/etc下：

# cp /application/mysql/support-files/my-large.cnf /etc/my.cnf

修改[mysqld]标签下,添加datadir和basedir。

# vim /etc/my.cnf

[mysqld]

basedir = /application/mysql

datadir = /data

### 2.1.7提供mysql启动脚本

使用MySQL官方提供的启动脚本:

# cp /application/mysql/support-files/mysql.server /etc/init.d/mysqld

### 2.1.8添加开机启动

方法一：使用chkconfig管理

# chkconfig --add mysqld

# chkconfig mysqld on

方法二：使用rc.local管理

# cat >>/etc/rc.local<< EOF

###start mysqld###

/etc/init.d/mysqld start

EOF

观点:这里选择添加到rc.local,规范将所有自己添加的服务都放到rc.local中管理,防止多处启动,也可归档。

### 2.1.9添加至环境变量

方法一：添加至/etc/profile中

# echo "export PATH=/application/mysql/bin:$PATH" >>/etc/profile

# source /etc/profile

这种方法最为简便,但是写入脚本执行,无法直接生效,与脚本执行环境有关,子shell中设置的环境变量无法传递到父shell中。

方法二：添加批量添加软连接

# ln -s /application/mysql/bin/\* /usr/local/sbin/ #by sunwj

这种方法比较新颖,但是可以达到效果而不用将300MB的bin目录复制到PATH变量对应的路径中,而且这种方法可以直接用于写脚本。

### 2.1.10配置man帮助

编辑/etc/man.config,添加 MANPATH /application/mysql/man

# vim /etc/man.config

...省略...

MANPATH /application/mysql/man

...省略...

### 2.1.11配置头文件

这可以通过简单的创建链接实现：

# ln -s /applicaiton/mysql/include /usr/include/mysql

### 2.1.12配置库文件

将lib库添加到ld.so.conf中或者ld.so.conf.d文件夹下:

# echo '/application/mysql/lib' > /etc/ld.so.conf.d/mysql.conf

# ldconfig

**2.2 MySQL之源码包编译安装**

编译安装时,我们可以定制功能,设置参数,自由度高,这里我们使用mysql-5.5.32.tar.gz安装包作为演示,5.5版本都安装方式一样,小版本可以忽略。

### 2.2.1准备安装包

通过官方下载mysql-5.5.32.tar.gz或者最新5.5版本

### 2.2.2安装依赖包

这里需要两个包,ncurses-devel和libaio-devel,前者提供字符终端处理库,后者提供AIO,即异步IO所需库,直接通过yum安装:

# yum install ncurses-devel libaio-devel -y

### 2.2.3安装cmake

由于MySQL5.5以后版本都需要使用cmake进行编译安装,所以需要安装cmake工具,方法有两种:

方法一:使用源码安装:

# tar xf cmake-2.8.8.tar.gz cd cmake-2.8.8

# ./configure

# gmake

# gmake install

方法二:使用epel源,yum安装

# wget -0 /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-6.repo

# yum install cmake -y

### 2.2.4建立mysql用户

# useradd mysql -s /sbin/nologin -M

### 2.2.5解压并编译mysql

这里的编译参数指定各种文件或者文件夹的位置,默认字符集utf8,支持的字符集,mysql存储表引擎等。

# tar xf mysql-5.5.32.tar.gz

# cd mysql-5.5.32

# cmake . -DCMAKE\_INSTALL\_PREFIX=/application/mysql-5.5.32 \

-DMYSQL\_DATADIR=/application/mysql-5.5.32/data \

-DMYSQL\_UNIX\_ADDR=/application/mysql-5.5.32/tmp/mysql.sock \

-DDEFAULT\_CHARSET=utf8 \

-DDEFAULT\_COLLATION=utf8\_general\_ci \

-DEXTRA\_CHARSETS=gbk,gb2312,utf8,ascii \

-DENABLED\_LOCAL\_INFILE=ON \

-DWITH\_INNOBASE\_STORAGE\_ENGINE=1 \

-DWITH\_FEDERATED\_STORAGE\_ENGINE=1 \

-DWITH\_BLACKHOLE\_STORAGE\_ENGINE=1 \

-DWITHOUT\_EXAMPLE\_STORAGE\_ENGINE=1 \

-DWITHOUT\_PARTITION\_STORAGE\_ENGINE=1 \

-DWITH\_FAST\_MUTEXES=1 \

-DWITH\_ZLIB=bundled \

-DENABLED\_LOCAL\_INFILE=1 \

-DWITH\_READLINE=1 \

-DWITH\_EMBEDDED\_SERVER=1 \

-DWITH\_DEBUG=0

# make

# make install

### 2.2.6其他配置见MySQL二进制包安装过程

此时相当于tar xf mysql-5.5.45-linux2.6-x86\_64.tar.gz完成

**2.3 MySQL之yum源直接安装**

这种方式最简单,用于简单的测试或者临时搭建一个数据库,或者安装一个mysql客户端:

# yum install mysql mysql-devel mysql-server -y

**2.4 MySQL之源码rpm打包**

这里使用简单的rpm打包工具fpm进行打包。

### 2.4.1安装fpm

#安装ruby

yum -y install ruby rubygems ruby-devel

#添加gem源

gem source -a http://mirrors.aliyun.com/rubygems/

gem source -r http://rubygems.org/

gem source -r https://ruby.taobao.org/

#从gem源安装

fpm gem install fpm

### 2.4.2准备fpm的post-install脚本进行安装完初始化工作

脚本名:post-mysql.sh

#!/bin/bash

In -s /application/mysql-5.5.32 /application/mysql

[ `echo $PATH|grep" /application/mysql/bin"|wc -l` -ne 1 ] && \

ln -s /application/mysql/bin/\* /usr/local/sbin/

\cp /application/mysql/my.cnf /etc/my.cnf

useradd mysql -s /sbin/nologin -M

chown -R mysql.mysql /application/mysql-5.5.32

echo "/application/mysql/lib">/etc/ld.so.conf.d/mysql.conf && ldconfig

sed -i '/^\<MANPATH\>/a MANPATH /application/mysql/man' /etc/man.config

/etc/init.d/mysqld start

echo -e "###startup mysqld##\n/etc/init.d/mysqld start" >> /etc.rc.local

### 2.4.3使用fpm进行打包

# fpm -f -s dir -t rpm -n mysql-alone -v 1.0 --description "mysql-serve r installed by jackz" --post-install /server/src/post-mysql.sh /application/mysql-5.5.32/ /etc/init.d/mysqld

这里-f表不存在覆盖,-s打包源类型,-t指定包格式,-n指定包名,-v指定版本,--description指定描述,-d指定依赖,这里没有,--post-install指定安装完成脚本,最后接多个要打包的文件【不支持软连接】。

**3 MySQL多实例**

**3.1多实例介绍**

MySQL多实例就是在一台物理机上运行多个MySQL服务进程,监听多个端口,来分别提供MySQL服务。由于进程空间是相互独立的,而数据文件又是不同目录,所以多实例是逻辑上隔离的,充分利用了服务器的硬件资源,但是如果负载过大,会存在资源抢占的等问题。

**3.2多实例应用场景**

⮚资金紧张型公司

⮚并发量不是很大的业务

⮚门户网站

百度搜索引擎的数据库就是多实例的,一般是从库,例如某部门的例子,IBM48核CPU,内存96G,跑3-4个实例;新浪也是用的多实例,内存48G左右,门户网站使用多实例的目的就是配硬件好的服务器,节省IDC机柜空间,同时跑多实例让硬件资源不浪费。

**3.3多实例相关面试题**

1）你们数据库是多实例，跑几个实例？CPU、内存、磁盘类型、RAID类型分别如何配置？

一般在1-4个实例之间,1-2个最多,因为大业务占用的机器不较多,机器是R510居多,CPU是E5210,48G内存,磁盘12\*300GSAS盘,做RAID10。

2）新浪的MYSQL是编译安装的还是二进制安装的？或者什么方式多？

编译安装之后,做成RPM包,统一使用。

**3.4多实例配置思路**

### 3.4.1多实例多配置文件,多启动程序【老男孩推荐，官方手册有】

多个配置文件,耦合度低,使用不同的脚本分别启动,逻辑上分离。

### 3.4.2单一配置文件,单一启动程序【官方推荐】

一个配置文件,耦合度太高,不好管理。

**3.5多实例具体配置步骤**

### 3.5.1配置前环境及文件准备

**3.5.1.1基础环境**

这里可以使用二进制包安装、也可以使用源码包安装,进行到初始化阶段之前即可。用户mysql、环境变量、库文件、man帮助都己搞定。

**3.5.1.2多实例配置文件my.cnf**

[client]

port = 3306

socket = /data/3306/mysql.sock

[mysql]

no-auto-rehash

[mysqld]

user = mysql

port = 3306

socket = /data/3306/mysql.sock

basedir = /application/mysql

datadir = /data/3306/data

open\_files\_limit = 1024

back\_log = 600

max\_connections = 800

max\_connect\_errors = 3000

table\_cache = 614

external-locking = FALSE

max\_allowed\_packet = 8M

sort\_buffer\_size = 1M

join\_buffer\_size = 1M

thread\_cache\_size = 100

thread\_concurrency = 2

query\_cache\_size = 2M

query\_cache\_limit = 1M

query\_cache\_min\_res\_unit = 2k

#default\_table\_type = InnoDB

thread\_stack = 192K

#transaction\_isolation = READ-COMMITTED

tmp\_table\_size = 2M

max\_heap\_table\_size = 2M

long\_query\_time = 1

#log\_long\_format

#log-error = /data/3306/error.log

#log-slow-queries = /data/3306/slow.log

pid-file = /data/3306/mysql.pid

log-bin = /data/3306/mysql-bin

relay-log = /data/3306/relay-bin

relay-log-info-file = /data/3306/relay-log.info

binlog\_cache\_size = 1M

max\_binlog\_cache\_size = 1M

max\_binlog\_size = 2M

expire\_logs\_days = 7

key\_buffer\_size = 16M

read\_buffer\_size = 1M

read\_rnd\_buffer\_size = 1M

bulk\_insert\_buffer\_size = 1M

#myisam\_sort\_buffer\_size = 1M

#myisam\_max\_sort\_file\_size = 10G

#myisam\_max\_extra\_sort\_file\_size = 10G

#myisam\_repair\_threads = 1

#myisam\_recover

lower\_case\_table\_names = 1

skip-name-resolve

slave-skip-errors = 1032,1062

replicate-ignore-db=mysql

server-id = 1

innodb\_additional\_mem\_pool\_size = 4M

innodb\_buffer\_pool\_size = 32M

innodb\_data\_file\_path = ibdata1:128M:autoextend

innodb\_file\_io\_threads = 4

innodb\_thread\_concurrency = 8

innodb\_flush\_log\_at\_trx\_commit = 2

innodb\_log\_buffer\_size = 2M

innodb\_log\_file\_size = 4M

innodb\_log\_files\_in\_group = 3

innodb\_max\_dirty\_pages\_pct = 90

innodb\_lock\_wait\_timeout = 120

innodb\_file\_per\_table = 0

[mysqldump]

quick

max\_allowed\_packet = 2M

[mysqld\_safe]

log-error=/data/3306/mysql\_oldboy3306.err

pid-file=/data/3306/mysqld.pid #pid文件

[mysql]

no-auot-rehash

[mysqld]

user = mysql

port = 3306 #mysql实例端口，可修改

socket = /data/3306/mysql.sock #指定 socket 文件位置

basedir = /application/mysql #安装程序目录

datadir = /data/3306/data #mysql数据目录

open\_files\_limit = 1024 #文件描述符限制

back\_log = 600 #系统预处理等待队列连接数量

max**\_**connections = 800 #最大连接数

max\_connect\_errors = 3000 #最大错误数

table\_cache = 614 #表缓存

external-locking = FALSE #外部锁

max\_allowed\_packet = 8M #最大允许的包大小

sort\_buffer\_size = 1M #排序缓冲大小

join\_buffer\_size = 1M #联结缓冲大小

thread\_cache\_size = 100 #线程缓存数量

thread\_concurrency = 2 #线程并发数

query\_cache\_size = 2M #请求缓存大小

query\_cache\_limit = 1M #请求缓存限制大小

query\_cache\_min\_res\_unit = 2k #请求缓存最小保留单元大小

#default\_table\_type = InnoDB

thread\_stack = 192K #线程栈大小

#transaction\_isolation = READ-COMMITTED

tmp\_table\_size = 2M #临时表大小

max\_heap\_table\_size = 2M #最大堆表大小

long\_query\_time = 1 #长请求时间 1s

#log\_long\_format

#log-error = /data/3306/error.log #错误日志

#log-slow-queries = /data/3306/slow.log #慢查询日志

pid-file = /data/3306/mysql.pid #pid 文件位置

log-bin = /data/3306/mysql-bin #binlog 开启及位置

relay-log = /data/3306/relay-bin #relaylog 开启及位置

relay-log-info-file = /data/3306/relay-log.info #信息文件

binlog\_cache\_size = 1M #缓存大小

max\_binlog\_cache\_size = 1M #最大 binlog 缓存大小

max\_binlog\_size = 2M #最大 binlog 大小

expire\_logs\_days = 7 #日志超期时间

key\_buffer\_size = 16M #索引缓冲大小

read\_buffer\_size = 1M #读缓冲大小

read\_rnd\_buffer\_size = 1M #排序缓冲读大小

bulk\_insert\_buffer\_size = 1M #块插入缓冲大小

#myisam\_sort\_buffer\_size = 1M #MyISAM相关选项未开启

#myisam\_max\_sort\_file\_size = 10G

#myisam\_max\_extra\_sort\_file\_size = 10G

#myisam\_repair\_threads = 1

#myisam\_recover

lower\_case\_table\_names = 1 #忽略表名大小写

skip-name-resolve #跳过名称解析

slave-skip-errors = 1032,1062 #主从同步，从库错误忽略代码列表

replicate-ignore-db=mysql #复制忽略数据库：mysql

server-id = 1 #设置server-id,主从复制不能一样

innodb\_additional\_mem\_pool\_size = 4M #附加内存池大小

innodb\_buffer\_pool\_size = 32M #缓冲池大小

innodb\_data\_file\_path = ibdatal: 128M:autoextend #数据文件

innodb\_file\_io\_threads = 4 #文件 io 线程数

innodb\_thread\_concurrency = 8 #并发线程数

innodb—flushjog\_at\_trx\_comrrrLt = 2 #提交时刷新日志,0,1,2

innodb\_log\_buffer\_size = 2M #日志缓存大小

innodb jog\_file\_size = 4M #日志文件大小

innodb\_log\_files\_in\_group = 3 #一组日志文件数

innodb\_max\_dirty\_pages\_pct = 90 #dirtypages百分比

innodb jock\_wait\_timeout = 120 #锁等待超时时间

innodb\_file\_per\_table = 0 # 是否一表一数据文件

[mysqldump]

quick #快速复制

max\_allowed\_packet = 2M #最大允许数据包大小

[mysqld\_safe]

log-error=/data/3306/mysql\_oldboy3306.err #错误日志文件

pid-file=/data/3306/mysql.pid #pid文件

**3.5.1.3多实例启动脚本mysql**

#!/bin/sh

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

#this scripts is created by oldboy at 2007-06-09

#oldboy QQ:31333741

#site:http://www.etiantian.org

#blog:http://oldboy.blog.51cto.com

#oldboy trainning QQ group: 208160987 226199307 44246017

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

#init

port=3306

mysql\_user="root"

mysql\_pwd="oldboy"

CmdPath="/application/mysql/bin"

mysql\_sock="/data/${port}/mysql.sock"

#startup function

function\_start\_mysql()

{

if [ ! -e "$mysql\_sock" ];then

printf "Starting MySQL...\n"

/bin/sh ${CmdPath}/mysqld\_safe --defaults-file=/data/${port}/my.cnf 2>&1 > /dev/null &

else

printf "MySQL is running...\n"

exit

fi

}

#stop function

function\_stop\_mysql()

{

if [ ! -e "$mysql\_sock" ];then

printf "MySQL is stopped...\n"

exit

else

printf "Stoping MySQL...\n"

${CmdPath}/mysqladmin -u ${mysql\_user} -p${mysql\_pwd} -S /data/${port}/mysql.sock shutdown

fi

}

#restart function

function\_restart\_mysql()

{

printf "Restarting MySQL...\n"

function\_stop\_mysql

sleep 2

function\_start\_mysql

}

case $1 in

start)

function\_start\_mysql

;;

stop)

function\_stop\_mysql

;;

restart)

function\_restart\_mysql

;;

\*)

printf "Usage: /data/${port}/mysql {start|stop|restart}\n"

esac

!/bin/bash

#vars define

port=3306 #不同实例可更改

mysql\_user="root"

mysql\_pwd="123456" #关闭时使用mysqladmin关闭，需要正确密码

CmdPath="/application/mysql/bin"

mysql\_sock="/data/${port}/mysql.sock"

#startup function

function\_start\_mysql() #使用进程来检测状态更加准确，防止强制停止导致socket遗留问题

{

if [ `netstat -tuplnlgrep ${port}lwc -1` -ne 1 ];then

printf "Starting MySQl..."

/bin/sh ${CmdPath}/mysqld\_safe --defaults-file=/data/${port}/my.cnf 2>&1 > /dev/null &

while true #每300ms检测一次，直到mysqld启动完成

do

sleep 0.3

if [ 'netstat -tuplnlgrep ${port}Iwc -1` -ne 1 ];then

printf "."

else

print "ok!\n"

break

fi

done

else

printf "MySQL is running...\n"

exit

fi

}

#stop function

function\_stop\_mysql()

{

if [ ! -e "$mysql\_sock" ];then

printf "MySQL is stopped...\n"

exit

else

printf "Stoping MySQl...\n"

${CmdPath}/mysqladmin -u ${mysql\_user} -p${mysql\_pwd} -S /data/${port}/mysql.sock shutdown

fi

}

#restart function

function\_restart\_mysql()

{

printf "Restarting MySQL...

function\_stop\_mysql

sleep 2

function\_start\_mysql

}

case $1 in

start)

function\_start\_mysql

;;

stop)

function\_stop\_mysql

;;

restart)

function\_restart\_mysql

\*)

printf "Usage: /data/${port}/mysql {start|stop|restart}\n

esac

### 3.5.2多实例初始化配置

**3.5.2.1目录文件准备**

这里以两个实例进行配置,分别为3306和3307,首先安排目录结构,这里以/data/作为总目录,/data/{3306,3307}作为实例目录,/data/{3306,3307}/data分别作为多实例的数据目录:

# mkdir /data/{3306,3307}/data -p

将3306和3307对应的my.cnf配置文件和mysql启动脚本放入各自的目录下,注意修改配置信息,server-id不可相同,这里为了方便，使用端口号最为server-id:

# tree 3306 3307

3306

|-- data

|-- my.cnf

|-- mysql

3307

|-- data

|-- my.cnf

`-- mysql

初始化过程需要相应的目录权限,由于mysql脚本中包含mysql登录密码,所以必须授予700权限,防止密码泄露:

# chown -R mysql:mysql /data/

# chmod 700 /data/{3306,3307}/mysql

**3.5.2.2初始化**

#初始化3306数据

/application/mysql/scripts/mysql\_install\_db -- basedir=/application/mysql/ datadir=/data/3306/data/ --user=mysql

#初始化3307数据

/application/mysql/scripts/mysql\_install\_db --basedir=/application/mysql/ datadir=/data/3307/data/ --user=mysql

**3.5.2.3启动mysql服务测试**

/data/3306/mysql start

/data/3307/mysql start

**3.5.2.4检査mysqld服务**

# netstat -lntup|grep mysqld

**3.5.2.5添加到启动脚本中**

cat >>/etc/rc.local <<EOF

/data/3306/mysql start

/data/3307/mysql start

EOF

### 3.5.3添加一个多实例

**3.5.3.1创建目录**

mkdir -p /data/3308/data

**3.5.3.2拷贝文件**

\cp /data/3306/my.cnf /data/3308/

\cp /data/3306/mysql /data/3308/

**3.5.3.3替换参数**

sed -i 's/3306/3308/g' /data/3308/{my.cnf,mysql}

**3.5.3.4授权**

chown -R mysql:mysql /data/3308

chmod 700 /data/3308/mysql

**3.5.3.5初始化**

cd /application/mysql/scripts

./mysql\_install\_db --datadir=/data/3308/data --basedir=/application/mysql --user=mysql

**3.5.3.6启动并查看**

/data/3308/mysql start

netstat -lntup|grep 3308

### 3.5.4添加多实例脚本

#!/bin/bash

#FileName:add\_mysql\_instance.sh

#Description:

#Author:jackz

#Datatime:2015-12-24-14:39:13

#Usage :add\_mysql\_instance.sh

[ $# -ne 1 ] && {

echo "Usage:`basename $0 PORT`"

exit 10

}

tmpfile=`mktemp -t "tmp.XXXX"`

basedir="/application/mysql/"

port=$1

password="123456"

[ -f /data/${port} ] && echo "MYSQL INSTANCE ${port} exists." || \

mkdir /data/${port}/data -p

[ -f /data/3306/my.cnf ] && /bin/cp /data/3306/my.cnf /data/${port}/ || \

exit 2

[ -f /data/3306/mysql ] && /bin/cp /data/3306/mysql /data/${port}/ || \

exit 3

sed -i 's#3306#'"${port}"'#g' /data/${port}/my.cnf /data/${port}/mysql && \

chmod 700 /data/${port}/mysql

chown -R mysql.mysql /data/${port}

#initialize mysql

/application/mysql/scripts/mysql\_install\_db --basedir=${basedir}\

--datadir=/data/${port}/data\

--user=mysql 2>/dev/null|grep -i "ok" >>${tmpfile}

[ `grep -i "ok" ${tmpfile}|wc -l` -eq 2 ] && \

echo "initialize mysql successfully..." || { \

echo "initialize mysql failed..."

exit 4

}

sed -i 's#server-id=.\*$#server-id = '"${port}"'#g' /data/${port}/my.cnf

/data/${port}/mysql start

sleep 15

[ `lsof -i :${port}|grep "mysqld"|wc -l` -eq 1 ] && \

echo "start successfully..." || { \

echo "start mysql failed..."

exit 5

}

#set mysql root password

/application/mysql/bin/mysqladmin -S /data/${port}/mysql.sock -uroot password ${password}

sed -i 's#mysql\_pwd=.\*$#mysql\_pwd='"$password"'#g' /data/${port}/mysql

rm $tmpfile -f

**3.6多实例启动、登录、关闭命令**

### 3.6.1启动

多实例的登录使用mysqld\_safe指定my.cnf配置文件:

/application/mysql/bin/mysqld\_safe --defaults-file=/data/3306/my.cnf &

### 3.6.2登录

⯎如果是本地,使用socket登录:

mysql -uroot -p123456 -S /data/3306/mysql.sock

⯎如果是远程登录,首先grant授权,然后使用ip:port登录:

mysql -uroot -pl23456 -P 3306 -h 192.168.16.31

**3.6.3关闭**

由于是多实例,不能全部关闭,所以选择使用mysqladmin指定socket来关闭

# mysqladmin -uroot -pl23456 -S /data/3306/mysql.sock shutdown

**3.7 FAQ**

⯎使用/data/3306/mysql脚本关闭的时候关闭不了,出现error:'Access denied for user 'root'@'localhost' (using password: YES)'提示

由于mysqladmin需要使用权限关闭socket,所以需要修改mysql脚本中的密码。

⯎本机使用IP:PORT无法登录

需要grant授权远程登录,user+host\_ip才是一条验证条目。

**4 MySQL主从复制**

**4.1文件级别异机同步（共享）方案**

⮚scp/sftp/nc命令可以实现远程数据同步

⮚搭建ftp/http/svn/nfs服务器,客户端实现数据共享

⮚搭建samba文件共享服务器,同上

⮚利用 rsync/csync2/uirion 软件实现

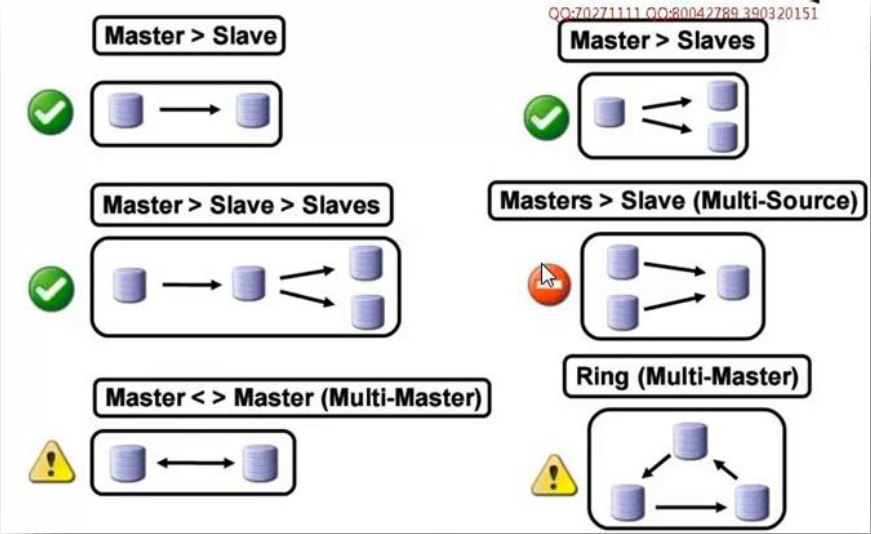
union可实现双向同步，csync2可实现多机同步。

⮚使用mysql、mongodb容器实现同步,如docker的共享数据卷

⮚程序双写

⮚DRBD+Heartbeat

**4.2 MySQL主从复制架构**



⮚主从模式,或者一主多从模式,属于两层结构,后端从库数量控制在3~5个,否则会影响性能,造成主从复制延迟等问题

⮚主从从模式,A+B+C,当主库宕机后,从库B提升为主

⮚多主一从模式,存在数据冲突,不推荐

⮚互为主从模式,可以通过程序实现双写

⮚多主环形模式,可以是实现多写

**4.3主从复制原理**

**4.3.1前提**

⮚主库需要开启log-bin,用于记录数据库更新SQL语句

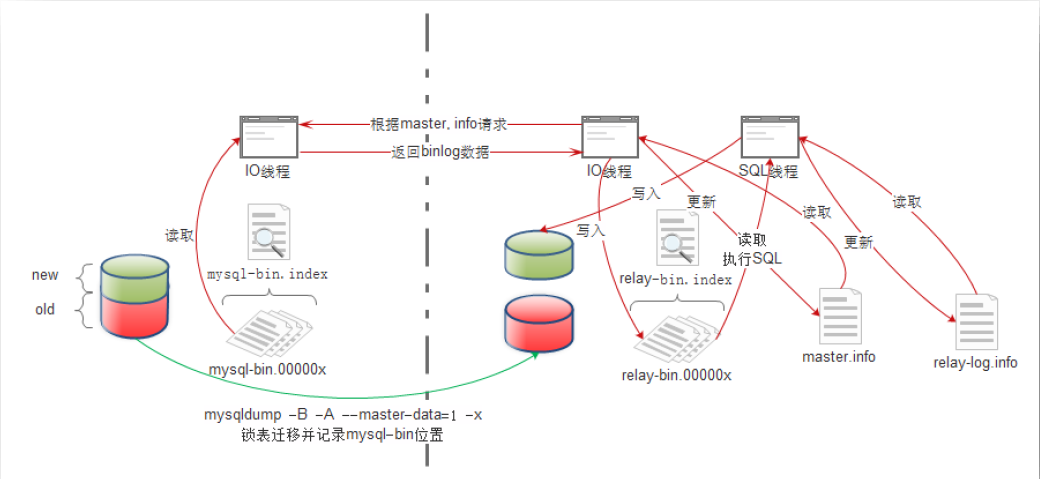
⮚主库授权同步用户

⮚从库需要开启relay-bin,用于存放主库复制过来的SQL语句

⮚从库设置主库信息

⮚主从库都需设置server-id且不能相同

**4.3.2主从复制过程**



1.从库IO线程读取master.info,根据主库信息以及binlog位置发起连接

2.主库dump线程根据请求,读取binlog文件+位置开始后一部分内容,发送给从库,数据包大小可以在my.cnf中配置

3.从库IO线程将内容写到relay-log文件中,并将位置信息写入master.info中

4.从库SQL线程实时监测relay-log文件,执行sql语句,写入数据库;同时更新relay-log.info文件,记录SQL线程执行的位置

5.循环往复以上步骤,完成主从复制。

**4.4主从复制配置**

**4.4.1 主库开启log-bin**

mysql>\! grep "log-bin" /data/3306/my.cnf

log-bin=/data/3306/mysql-bin 🡨配置文件中开启log-bin

mysql>show variables like "log\_bin"; 🡨查看log-bin状态

+----------------+------+

| Variable\_name | Value|

+----------------+------+

| log-bin | ON |

+----------------+------+

1 row in set (0.00 sec)

**4.4.2 创建同步用户并授权replication slave**

mysql> grant replication slave on \*.\* to rep@"192.168.16.%" identified by "123456";

Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;

Query OK, 0 rows affected (0.00 sec)

**4.4.3 从库开启relay-log**

# grep "relay" /data/3307/my.cnf

relay-log = /data/3307/relay-bin 🡨开启relay-log

relay-log-info-file = /data/3307/relay-log.info 🡨指定relay-log-info文件

**4.4.4 设置主从server-id,不能相同**

# grep server-id /data/{3306,3307}/my.cnf

/data/3306/my.cnf:server-id = 1

/data/3307/my.cnf:server-id = 3

#如果相同,会导致无法同步,如果单单主从还好,会报错,如果是主从从,即A-B-C结构,A 和C的server-id相同,也是无法同步的,而且不会报错。

**4.4.5 mysqldump导出数据**

方法一：直接mysqldump -x参数锁表

# mysqldump -S /data/3306/mysql.sock -uroot -p123456 --events -A -B -x -F --master-data=1 > /server/backup/mysql.sql

#参数说明：-A备份所有库表,-B备份多个数据库,带创库语句,方便恢复,-x锁表,--master-data=1记录binlog日志文件名以及位置,-F刷新bin-log日志

方法二：在MySQL命令行中使用命令锁表，然后手动记录binlog文件及位置

mysql> flush table with read lock;

Query OK, 0 rows affected (0.03 sec)

mysql> show master status;

+------------------+----------+--------------+-----------------+

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB|

+------------------+----------+--------------+-----------------+

| mysql-bin.000008 | 107 | | |

+------------------+----------+--------------+-----------------+

1 row in set (0.00 sec)

然后通过mysqldump备份，结束后解锁

mysql> unlock tables;

Query OK, 0 rows affected (0.00 sec)

注：锁表后不能退出MySQL命令行，否则锁表会失效。

**4.4.6导入数据到从库**

#导入数据库之前确保关闭从库复制

# mysql -S /data/3307/mysql.sock -uroot -pl23456 -e "stop slave;"

# mysql -S /data/3307/mysql.sock -uroot -pl23456 </server/backup/mysq1.sql

#如果数据库很大,可以考虑使用物理备份工具,比如cp,tar或者xtrabackup工具

**4.4.7从库上设置主库信息**

#mysql -S /data/3307/mysql.sock -uroot -p123456 -e "change master to

master\_host='192.168.16.102',🡨主库 IP

master\_port=3306,🡨主库端口

master\_user='rep',🡨主库同步用户

master\_password='123456';"🡨主库同步用户密码

#由于有master-data=l,所以导入数据时就有设置binlog位置的语句;否则需要在这里添上change master to MASTER\_LOG\_FILE='mysql-bin.0000012',MASTER\_LOG\_POS=267;

**4.4.8开启主从同步**

# mysql -S /data/3307/mysql.sock -uroot -p123456 -e "start slave;"

**4.4.9査看主动同步状态**

# mysql -S /data/3307/mysql.sock -uroot -pl23456 -e "show slave status \G"

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Slave\_IO\_State:Waiting for master to send event

Master\_Host: 192.168.16.102 #🡨主库 IP

Master\_User: rep #🡨同步用户

Master\_Port: 3306 #🡨端口

Connect\_Retry: 60 #🡨连接重试超时

Master\_Log\_File: mysql-bin.000008 #🡨主库bin-log文件名

Read\_Master\_Log\_Pos: 107 #🡨主库 bin-log 位置

Relay\_Log\_File: relay-bin.000010 #🡨从库中继日志文件名

Relay\_Log\_Pos: 253 #🡨从库中继日志位置

Relay\_Master\_Log\_File: mysql-bin.000008

Slave\_IO\_Running: Yes #🡨线程状态

Slave\_SQL\_Running: Yes #🡨SQL线程状态

Replicate\_Do\_DB:

Replicate\_Ignore\_DB: mysql

Replicate\_Do\_Table:

Replicate\_Ignore\_Table:

Replicate\_Wild\_Do\_Table:

Replicate\_Wild\_Ignore\_Table:

Last\_Errno: 0 #🡨错误数量

Last\_Error: #🡨错误类型

Skip\_Counter: 0

Exec\_Master\_Log\_Pos: 107

Relay\_Log\_Space: 446

Until\_Condition: None Until\_Log\_File:

Until\_Log\_Pos: 0

Master\_SSL\_Allowed: No

Master\_SSL\_CA\_File:

Master\_SSL\_CA\_Path:

Master\_SSL\_Cert:

Master\_SSL\_Cipher:

Master\_SSL\_Key:

Seconds\_Behind\_Master: 0 #🡨从库延迟主库时间 Master\_SSL\_Verify\_Server\_Cert: No

Last\_IO\_Errno: 0 #🡨10错误数量

Last\_I0\_ErT〇r: #🡨10 错误类型

Last\_SQL\_Errno: 0 #🡨SQL 错误数量

Last\_SQL\_Error: #🡨SQL 错误类型

Replicate\_Ignore\_Server\_Ids:

Master\_Server\_Id: 1 #🡨主库server-id

#两个yes说明主从同步成功

主从同步配置总结：

⮚准备两台数据库或者单台多实例,初始化并配置好

⮚配置my.cnf主库配置log-bin和server-id,从库配置relay-log和relay-info-file,修改配置文件重启生效

⮚主库添加同步用户rep并授权replication slave权限

⮚使用mysqldump -x --master-date等参数锁表备份

⮚恢复数据到从库中

⮚从库设置主库信息

⮚从库开启同步开关

⮚检查同步状态

**4.5主从复制配置FAQ**

**4.5.1Q: I/O线程报错怎么办？**

A: I/O线程报错一般是因为CHANGE MASTER TO的信息配置有误,比如连接验证错误,binlog文件错误等,检查并重新配置即可。

**4.5.2Q: SQL线程报错怎么办?**

A: SQL线程报错一般是因为binlog记录点有问题,位置提前或者落后导致SQL语句执行失败而报错,此时可以重新导入备份以及binlog位置;或者根据错误选择跳过或者忽略错误等操作。

**4.5.3Q:提示数据库已经存在,冲突怎么办?**

A:如果出现数据库冲突,原因有很多,比如设置主库信息出现问题,或者人为修改从库数据等,此时可以通过以下两种方法解决:

方法一：MySQL命令行中配置

stop slave;

set global sql\_slave\_skip\_counter = 1;

start slave;

方法二：修改配置文件my.cnf跳过错误

slave-skip-errors = 1007,1008,1032,1062

**4.6 MySQL主从从配置**

现假设有三个MySQL数据库,组成A(3306)🡪B(3307)🡪(3308)的主从从结构,其中A是B的主库,B是C的主库。

**4.6.1配置A+B主从复制**

方法同主从复制

**4.6.2确保ABC三库server-id不同**

# grep server-id /data/{3306,3307,3308}/my.cnf

/data/3306/my.cnf:server-id = 1

/data/3307/my.cnf:server-id = 3

/data/3308/my.cnf:server-id = 3308

**4.6.3 B数据库上开启从库log-bin**

# grep -E "log-bin|log-slave-updates" /data/B307/my.cnf

log-bin = /data/3307/mysql-bin

log-slave-updates

注：由于B是从库,必须开启log-slave-updates才能将主库同步过来的SQL语句记录到B库的bin-log文件中。

**4.6.4配置B+C主从复制**

方法同主从复制

**4.7 MySQL主主模式配置**

方法和主从从类似,不过A🡨🡪B中两者都需要开启log-bin以及log-slave-updates

注：主主双写时需要对id进行特殊处理,这里有两种方法:

**4.7.1通过序列服务器,当需要插入数据时,先去序列服务器取序号,然后再插入**

**4.7.2 A库写 1,3,5,7...;B库写 2,4,6,8...**

#修改A库配置文件：

auto\_increment\_increment = 2 #自增步长为2

auto\_increment\_offset = 1 #设置起始偏移为1, id为1,3,5,7...

#修改B库配置文件：

auto\_increment\_increment = 2 #步长2

auto\_increment\_offset = 2 #设置起始偏移为2, id为2,4,6,8...

#创建数据库的时候使用自增主键id

create database oldboy; #创建数据库old\_dboy

use oldboy; #使用数据库

CREATE TABLE `tl` (

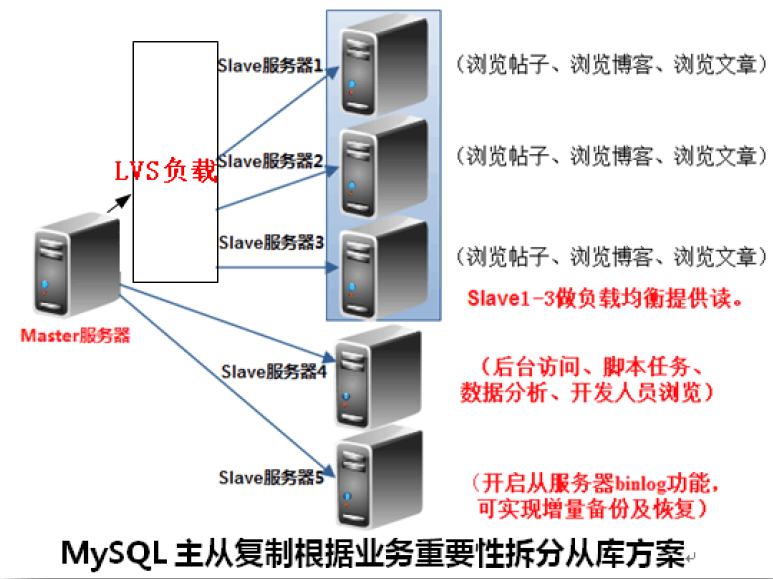
`id` bigint(12) NOT NULL auto\_increment,

`name` varchar(12) NOT NULL,

PRIMARY KEY (`id`)

）；

**4.8主从复制企业应用场景**



一主多从方案，建议从库数量3-5个，否则会出现从库延迟大，不同步的问题;然后划分不同的从库不同的业务:

⮚ Sl、S2、S3提供对外提供读服务,前端增加LVS实现负载均衡;

⮚ S4提供测试数据分析等,内部使用;

⮚ S5提供任何业务,只作为备份服务器使用。

⮚ Ml为存在单点故障,可以选择增加一台备主

前端可以增加MySQL代理软件实现读写分离;也可以使用程序实现读写分离。

至于主从的高可用架构：

⮚ 两台主库之间只做简单的keepalived,会丢数据

⮚ 两台主库之间使用Heartbeat+Drbd+MySQL

⮚ 主从之间使用MMM架构,增加一台监控服务器,主库一个VIP,从库多个VIP提供服务

⮚ 主从之间使用MHA架构,增加一台监控服务器,实现高可用。

主从复制架构的数据备份策略：

使用从库做备份。

备份时，通过执行命令：stop slave sql\_thread;;只停止SQL线程,也就是停止执行relay-log中的sql语句,相当于"锁表"。但是IO线程依然保留工作状态,依旧可以源源不断地从主库更新bin-log文件到从库的relay-log中,但不执行。

备份方式可以采取mysqldump做全量备份,配合binlog的增量备份,可以恢复到任意时间点的数据。

如果50G以下数据量可以使用mysqldump备份;

如果大于50G,建议使用物理备份,cp,tar或者xtrabackup工具。

全量备份完成后,通过start slave sql\_thread;开启SQL线程,SQL线程会把relay-log中的SQL更新到数据库中。

**4.9 MySQL读写分离方案**

**4.9.1通过程序实现读写分离（性能,效率最佳,推荐,公司用得最多)**

程序员根据请求语句,连接不同的数据库。

比如 update delete insert 选择 dbl;select 选择 db2

**4.9.2通过开源的软件实现读写分离**

mysql-proxy,Amoeba等代理软件实现读写分离

**4.9.3大型门户独立开发DAL层综合软件**

**4.10主从库不同步的原因**

**4.10.1从库太多导致复制延迟**

优化：建议从库数量3-5个为宜

**4.10.2从库硬件比主库硬件差**

优化：提升硬件性能

**4.10.3慢SQL语句过多**

优化:SQL语句执行时间太长,需要SQL语句优化

**4.10.4主从复制的设计问题**

优化:主从复制单线程,可以通过多线程IO方案解决;另外MySQL5.6.3支持多线程IO复制。

http://www.oschina.net/question/54100\_68329

**4.10.5主从库之间的网络延迟**

优化：尽量链路短，提升端口带宽

**4.10.6主库读写压力大**

优化：前端加buffer和缓存。

**4.11监测主从同步**

监控的主要的参数：

# mysql -S /data/3307/mysql.sock -uroot -pl23456 -e "show slave status \G"|grep -iE "Behind|Running”|sed -rn 's/^\s\*(.\*)$/\1/gp'

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

Seconds\_Behind\_Master: 0

由于各种原因,这里的延迟时间不是太准确,所以我们可以通过插入时间函数来判断主从复制延迟。

主库上：

mysql> create database monitor;

Query OK, 1 row affected (0.00 sec)

mysql> grant select,insert,update,delete on monitor.\* to monitor@'192.168.16.%' identified by '123456';

mysql> use monitor

Database changed

mysql>create table time( id int(2) auto\_increment,time int(10),primary key(id));

Query OK, 0 rows affected (0.01 sec)

mysql> desc time;

+--------+-------+-------+------+----------+---------------+

| Field | Type | Null | Key | Default | Extra |

+--------+-------+-------+------+----------+---------------+

| id | int(2) | NO | PRI | NULL | auto\_increment|

| time | int(10)| YES | | NULL | |

+--------+-------+-------+------+----------+---------------+

2 rows in set (0.00 sec)

监控机上:

SHELL> mysql -umonitor -pl23456 -h 192.168.16.102 -P3306 -e "use monitor;insert into time(time) values(UNIX\_TIMESTAMP());"

SHELL> mysql -umonitor -pl23456 -h 192.168.16.102 -P3306 -e "use monitor;select time from time where id = 1;"|grep "[0-9]"

1439460039

SHELL> mysql -umonitor -pl23456 -h 192.168.16.102 -P3306 -e "use monitor;select time from time where id = 1;"|grep "[0-9]"

1439460039

查看b主库binlog信息:

# mysqlbinlog mysql-bin.000011 |grep -Cl UNIX\_TIME

SET TIMESTAMP=1439460039/\*!\*/;

insert into time(time) values(UNIX\_TIMESTAMP())

/\*!\*/;

虽然bin-log是语句级别的形式,语句以及函数都己经记录了,但是同时binlog也记录了语句的上下文信息,所以通过时间函数检测是否同步的方法是不可取的,也许可以通过更改binlog信息来使函数执行的时间不一样,从而获取同步延迟。

才疏学浅,方法待定。

**4.12数据库集群的授权**

读写分离集群:如何授权web用户访问？

方法1:传统方法,使用两个用户;或者使用两个验证条目

主库：

用户：web\_w 密码：123 3306 10.0.0.7

权限：insert,update,select,delete

从库：

用户：web\_r 密码：123 3306 10.0.0.8

权限：select

方法2:通过权限控制,利用主从复制的特点,主同步到从,从可修改;或者通过参数控制忽略mysql库复制

主库：

用户：web 密码：123 3306 10.0.0.7 oldboy

权限：insert,update,select,delete

从库：

用户：web 密码：123 3306 10.0.0.8 oldboy

权限：select

配置方法：

a.主库创建完用户和权限,从库上revoke收回写的权限。

b.忽略授权库同步

binlog-ignore-db=mysql 🡨这个是关键，binlog更新忽略mysql数据库 replicate-ignore-db=mysql 🡨单配复制忽略数据库不生效

方法3:设置从库read-only,参数控制web用户读写权限。

主库从库：

用户：web 密码：123 3306 10.0.0.7 oldboy

权限：insert,update,select,delete

配置方法：

从库配置文件my.cnf中添加read-only一行即可。

read-only参数更新的条件:

a.具有SUPER权限的用户可以更新,比如root

b.具有REPLICATION SLAVE权限的用户可以更新,比如主从复制所用的rep用户

**4.13 MySQL高可用之角色切换原理**

情景假设:

主库master宕机。

M同步VIP，所有从库都以VIP为主库IP

M-->S1

>S2

>S3

>S4

>S5

**4.13.1太子,接班人如何选择**

1.半同步从库（MySQL自带半同步插件或者谷歌半同步插件)

**原理**：S1和主库做半同步,主库插入数据后,同时写入到S1,成功后返回,所以S1和M的数据是实时同步的,所以选S1作为太子。

**优点**：两台库同时写入数据。

**缺点**：写入会慢,网络不稳定,主库持续等待。

解决措施：

⮚连不上S1的时候,会自动转为异步。

⮚设置超时时间10秒,超过时间转为异步。

⮚S1网络,硬件要好,不提供服务,干等接管。

涉及到的两个插件库文件：

# Is /application/mysql/lib/plugin/semisync\_\*

/application/mysql/lib/plugin/semisync\_master.so —主库半同步插件 /application/mysql/lib/plugin/semisync\_slave.so —从库半同步插件

2.选择S1空载,只做同步的从库（百度,500台服务器)。

3.皇帝驾崩现选（耽误事，容易被篡位）。

**4.13.2主库宕机的两种情况**

1.如果主库可以ssh连接

binlog数据没丢,要把主库的binlog补全到所有从库。（半同步binlog补全不需要。）

a.提升S1为Ml的操作。

1)调配置read-only。授权用户权限select修改,变成可写,开启binlog。

如果做了双主同步,就直接切换即可,啥都不用做。

2)删除master.info 以及 relay-log\*

3)登录数据库reset master

4)重启数据库

b.所有从库:CHANGE MASTER TO MASTER\_LOG\_FILE=' ',MASTER\_L0G\_P0S=' ';

2.如果主库连不上:

a.半同步从库提升主库,半同步数据补全到所有从库（通过中继日志）。

半同步从库提升主库操作需要上述步骤1-a,所有从库CHANGE MASTER同1-b

b.S1啥也不干的方法

提升啥也不干的S1从库为主库操作需要上述步骤1-a,所有从库CHANGE MASTER同1-b

c.主库宕机,没有指定从库为主库。

选主的方法：

1)登录所有从库show processlist\G;确保IO和SQL都读取了自己的binlog并应用到了数据库。

2)登录所有从库分别查看master.info,看谁更新得更快。

如果相同文件,选择P0S最大的;如果不同文件,选择编号最大的为主库,补全（通过中继日志）其它从库

3)切换角色

提升选择的S1为主库Ml操作需要上述步骤1-a,所有从库CHANGE MASTER同1-b。

**5 MySQL程序相关**

**5.1数据库启动命令**

一般是通过调用脚本来启动,比如官方提供的mysql.server脚本,其中使用了mysqld\_safe的命令:

# /bin/sh mysqld\_safe --defaults-file=/data/3306/my.cnf &

注：对于单实例,会寻找默认的my.cnf位置,比如/etc/my.cnf;对于多实例,由于配置文件不同,我们启动时需要通过--defaults-file指定配置文件。

**5.2数据库关闭命令**

如果是单实例MySQL数据库,直接使用官方提供的脚本关闭即可,脚本中包含如下命令:

# kill -0 $mysqld\_pid 2

经过搜索,kill -0不发送任何信号,但是系统会进行错误检查。所以经常用来检查一个进程是否存在,存在返回0;不存在返回1。

然后脚本中使用kill $mysqld\_pid命令杀掉mysqld进程,kill默认信号为15,SIGTERM。

如果是多实例,也可以通过这种方法杀掉mysql进程,只需要通过pid文件获取 mysqld 的pid即可。

当然我们也可以通过mysqladmin指定套接字发送shutdown命令来关闭mysql服务：

# mysqladmin -S /data/3306/mysql.sock -uroot -pl23456 shutdown

为了安全的杀死mysqld,我们也可以使用USR2信号来关闭

# kill -s USR2 'cat /data/3306/mysqld.pid'

注:千万不可使用kill -9来终止mysqld服务,会导致数据丢失:pkill的方法也慎用,多实例pkill会杀掉所有mysqld进程。

**更详细、更安全的关闭**mysqld**方式:**<http://imysql.com/2014/08/13/mysql-faq-howto-shutdown-mysqld-fulgraceful.shtml>

野蛮粗鲁杀死数据库导致故障企业案例：

<http://oldboy.blog.51cto.com/2561410/1431161>

[http://oldboy.blog.51cto.com/2561410/1431172](http://oldboy.blog.51cto.com/Z561410/1431172)

**5.3设置MySQL命令提示符**

方法一：

临时生效:

mysql> prompt \u@oldboy \r:\m:\s->;

方法二：

永久生效：

[mysql]

prompt=\\u@oldboy \\r:\\m:\\s->

**5.4修改密码的几种方法**

**5.4.1在shell命令行使用mysqladmin修改**

设置密码方法：

# mysqladmin -uroot password '123456';

更新密码方法：

# mysqladmin -uroot -p'123456' password '123456';

多实例方法：-S指定sock

**5.4.2使用update的方式,修改密码;使用完需要刷新权限**

update mysql.user set password=PASSWORD('123456') where user = 'root' and host = 'localhost';

flush privileges;

**5.4.3使用变量,修改已登录用户的密码**

set password=password('123456');

**5.5忘记root密码如何修改**

解决方法:通过--skip-grant-tables参数忽略授权表即可免密码登录

单实例:

# /etc/init.d/mysqld stop

# mysqld\_safe --skip-grant-tables &

# mysql

update mysql.user set password=PASSWORD('newpasswd') where user='root' and host='localhost';

flush privileges;

多实例:

# killall mysqld

# mysqld\_safe --defaults-file=/data/3306/my.cnf --skip-grant-tables &

# mysql -uroot -p -S /data/3306/my.cnf

update mysql.user set password=PASSWORD('newpasswd') where user='root' and host='localhost';

flush privileges;

注意:一旦刷新权限,--skip-grant-tables就失效了

**5.6如何做好MySQL安全措施**

**5.6.1为root设置比较复杂的密码**

可以设置为随机生成的密码,然后使用密码管理工具管理密码。

**5.6.2删除无用的mysql库内的用户账号**

方法一：

drop user root@‘127.0.0.1'; 🡨此种方法无法删除含有大写字母或者特殊字符的条目

方法二：

delete from mysql.user where user='root' and host='127.0.0.1'

#当我们设置密码的时候,默认设置的是root@localhost这个条目的密码,而root@127.0.0.1依旧是空密码,所以必须删除这些无用的验证条目。

**5.6.3授权尽量范围小,不要用％全部授权**

尽量使用IP段来限定允许主机段,可以使用192.168.16.%,也可以使用192.168.0/255. 255.255.0

**5.6.4删除root用户,建立拥有grant权限的用户system**

grant all privileges on \*.\* to system@'localhost' identified by '123456' with grant option;

**6 MySQL SQL语句基础**

SQL语句最常见的分类一般就3类：

|  |  |  |
| --- | --- | --- |
| 类型 | 解释 | 作用 |
| DDL(Data Definition Language) | 数据定义语言（CREATE,ALTER, DROP) | 管理基础数据,例如库、表 |
| DCL (Data Control Language) | 数据控制语言(GRANT,REVOKE,COMMIT,ROLLBACK) | 用户授权,权限回收,数据提交,回滚等 |
| DML (Data Manipulation Language) | 数据操作语言（SELECT, INSERT,DELETE,UPDATE) | 针对数据库里的表,记录操作 |

对于高级运维来说,需要熟练掌握的是DDL以及DCL，DML需要了解。

**6.1 DDL数据定义语言**

**6.1.1创建数据库**

Syntax:

CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db\_name

[create\_specification] ...

create\_specification:

[DEFAULT] CHARACTER SET [=] charset\_name

| [DEFAULT] COLLATE [=] collation\_name

说明:character set指定字符集;collate指定校对字符。如果不指定字符集,默认是使用系统数据库的字符集。

举例：创建一个oldboy数据库,指定字符集为GBK。

mysql>create database oldboy character set gbk collate gbk\_chinese\_ci; #🡨建库语句

Query OK, 1 row affected (0.00 sec)

mysql>show databases like 'oldboy'; #🡨查看数据库语句

+-------------------+

| Database (oldboy) |

+-------------------+

| oldboy |

+-------------------+

1 row in set (0.00 sec

mysql> show create database oldboy\G #🡨查看建库语句，注意此处没分号

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Database: oldboy

Create Database: CREATE DATABASE `oldboy` /\*!40100 DEFAULT CHARACTER SET gbk \*/

1 row in set (0.00 sec)

字符集以及collate校对字符集可以通过show查询：

mysql> show character set;

+----------+-----------------------------+---------------------+--------+

| Charset | Description | Default collation | Maxlen |

+----------+-----------------------------+---------------------+--------+

| big5 | Big5 Traditional Chinese | big5\_chinese\_ci | 2 |

| dec8 | DEC West European | dec8\_swedish\_ci | 1 |

| cp850 | DOS West European | cp850\_general\_ci | 1 |

| hp8 | HP West European | hp8\_english\_ci | 1 |

| koi8r | KOI8-R Relcom Russian | koi8r\_general\_ci | 1 |

| latin1 | cp1252 West European | latin1\_swedish\_ci | 1 |

| latin2 | ISO 8859-2 Central European | latin2\_general\_ci | 1 |

| swe7 | 7bit Swedish | swe7\_swedish\_ci | 1 |

| ascii | US ASCII | ascii\_general\_ci | 1 |

| ujis | EUC-JP Japanese | ujis\_japanese\_ci | 3 |

| sjis | Shift-JIS Japanese | sjis\_japanese\_ci | 2 |

| hebrew | ISO 8859-8 Hebrew | hebrew\_general\_ci | 1 |

| tis620 | TIS620 Thai | tis620\_thai\_ci | 1 |

| euckr | EUC-KR Korean | euckr\_korean\_ci | 2 |

| koi8u | KOI8-U Ukrainian | koi8u\_general\_ci | 1 |

| gb2312 | GB2312 Simplified Chinese | gb2312\_chinese\_ci | 2 |

| greek | ISO 8859-7 Greek | greek\_general\_ci | 1 |

| cp1250 | Windows Central European | cp1250\_general\_ci | 1 |

| gbk | GBK Simplified Chinese | gbk\_chinese\_ci | 2 |

| latin5 | ISO 8859-9 Turkish | latin5\_turkish\_ci | 1 |

| armscii8 | ARMSCII-8 Armenian | armscii8\_general\_ci | 1 |

| utf8 | UTF-8 Unicode | utf8\_general\_ci | 3 |

| ucs2 | UCS-2 Unicode | ucs2\_general\_ci | 2 |

| cp866 | DOS Russian | cp866\_general\_ci | 1 |

| keybcs2 | DOS Kamenicky Czech-Slovak | keybcs2\_general\_ci | 1 |

| macce | Mac Central European | macce\_general\_ci | 1 |

| macroman | Mac West European | macroman\_general\_ci | 1 |

| cp852 | DOS Central European | cp852\_general\_ci | 1 |

| latin7 | ISO 8859-13 Baltic | latin7\_general\_ci | 1 |

| utf8mb4 | UTF-8 Unicode | utf8mb4\_general\_ci | 4 |

| cp1251 | Windows Cyrillic | cp1251\_general\_ci | 1 |

| utf16 | UTF-16 Unicode | utf16\_general\_ci | 4 |

| cp1256 | Windows Arabic | cp1256\_general\_ci | 1 |

| cp1257 | Windows Baltic | cp1257\_general\_ci | 1 |

| utf32 | UTF-32 Unicode | utf32\_general\_ci | 4 |

| binary | Binary pseudo charset | binary | 1 |

| geostd8 | GEOSTD8 Georgian | geostd8\_general\_ci | 1 |

| cp932 | SJIS for Windows Japanese | cp932\_japanese\_ci | 2 |

| eucjpms | UJIS for Windows Japanese | eucjpms\_japanese\_ci | 3 |

+----------+-----------------------------+---------------------+--------+

39 rows in set (0.02 sec)

**6.1.2显示数据库**

Syntax:

SHOW {DATABASES | SCHEMAS}

[LIKE 'pattern' | WHERE expr]

#查看所有数据库,也可以通过like或者where模糊查询或者指定条件。

举例：

mysql> show databases like '%oldboy%'; #🡨%为通配符，相当于所有

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| mysql |

| oldboy |

| performance\_schema |

| test |

+--------------------+

5 rows in set (0.00 sec)

**6.1.3删除数据库**

Syntax:

DROP {DATABASE | SCHEMA} [IF EXISTS] db\_name

#删除数据库会删除整个数据库的所有数据，所以一定要谨慎。

举例：

mysql> drop database oldboy;

Query OK, 0 rows affected (0.00 sec)

**6.1.4切换数据库**

Syntax:

USE db\_name

#use db告诉MySQL使用哪个数据库作为默认数据库,方便进行后续的操作。

举例：

mysql> use test;

Database changed

**6.1.5创建表**

Syntax:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name

(create\_definition,...)

[table\_options]

[partition\_options]

Or:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name

[(create\_definition,...)]

[table\_options]

[partition\_options]

select\_statement

Or:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name

{ LIKE old\_tbl\_name | (LIKE old\_tbl\_name) }

create\_definition:

col\_name column\_definition

| [CONSTRAINT [symbol]] PRIMARY KEY [index\_type] (index\_col\_name,...)

[index\_option] ...

| {INDEX|KEY} [index\_name] [index\_type] (index\_col\_name,...)

[index\_option] ...

| [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]

[index\_name] [index\_type] (index\_col\_name,...)

[index\_option] ...

| {FULLTEXT|SPATIAL} [INDEX|KEY] [index\_name] (index\_col\_name,...)

[index\_option] ...

| [CONSTRAINT [symbol]] FOREIGN KEY

[index\_name] (index\_col\_name,...) reference\_definition

| CHECK (expr)

column\_definition:

data\_type [NOT NULL | NULL] [DEFAULT default\_value]

[AUTO\_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]

[COMMENT 'string']

[COLUMN\_FORMAT {FIXED|DYNAMIC|DEFAULT}]

[STORAGE {DISK|MEMORY|DEFAULT}]

[reference\_definition]

data\_type:

BIT[(length)]

| TINYINT[(length)] [UNSIGNED] [ZEROFILL]

| SMALLINT[(length)] [UNSIGNED] [ZEROFILL]

| MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]

| INT[(length)] [UNSIGNED] [ZEROFILL]

| INTEGER[(length)] [UNSIGNED] [ZEROFILL]

| BIGINT[(length)] [UNSIGNED] [ZEROFILL]

| REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]

| DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]

| FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]

| DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]

| NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]

| DATE

| TIME

| TIMESTAMP

| DATETIME

| YEAR

| CHAR[(length)]

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| VARCHAR(length)

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| BINARY[(length)]

| VARBINARY(length)

| TINYBLOB

| BLOB

| MEDIUMBLOB

| LONGBLOB

| TINYTEXT [BINARY]

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| TEXT [BINARY]

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| MEDIUMTEXT [BINARY]

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| LONGTEXT [BINARY]

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| ENUM(value1,value2,value3,...)

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| SET(value1,value2,value3,...)

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| spatial\_type

index\_col\_name:

col\_name [(length)] [ASC | DESC]

index\_type:

USING {BTREE | HASH}

index\_option:

KEY\_BLOCK\_SIZE [=] value

| index\_type

| WITH PARSER parser\_name

| COMMENT 'string'

reference\_definition:

REFERENCES tbl\_name (index\_col\_name,...)

[MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]

[ON DELETE reference\_option]

[ON UPDATE reference\_option]

reference\_option:

RESTRICT | CASCADE | SET NULL | NO ACTION

table\_options:

table\_option [[,] table\_option] ...

table\_option:

ENGINE [=] engine\_name

| AUTO\_INCREMENT [=] value

| AVG\_ROW\_LENGTH [=] value

| [DEFAULT] CHARACTER SET [=] charset\_name

| CHECKSUM [=] {0 | 1}

| [DEFAULT] COLLATE [=] collation\_name

| COMMENT [=] 'string'

| CONNECTION [=] 'connect\_string'

| DATA DIRECTORY [=] 'absolute path to directory'

| DELAY\_KEY\_WRITE [=] {0 | 1}

| INDEX DIRECTORY [=] 'absolute path to directory'

| INSERT\_METHOD [=] { NO | FIRST | LAST }

| KEY\_BLOCK\_SIZE [=] value

| MAX\_ROWS [=] value

| MIN\_ROWS [=] value

| PACK\_KEYS [=] {0 | 1 | DEFAULT}

| PASSWORD [=] 'string'

| ROW\_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}

| TABLESPACE tablespace\_name [STORAGE {DISK|MEMORY|DEFAULT}]

| UNION [=] (tbl\_name[,tbl\_name]...)

partition\_options:

PARTITION BY

{ [LINEAR] HASH(expr)

| [LINEAR] KEY [ALGORITHM={1|2}] (column\_list)

| RANGE{(expr) | COLUMNS(column\_list)}

| LIST{(expr) | COLUMNS(column\_list)} }

[PARTITIONS num]

[SUBPARTITION BY

{ [LINEAR] HASH(expr)

| [LINEAR] KEY [ALGORITHM={1|2}] (column\_list) }

[SUBPARTITIONS num]

]

[(partition\_definition [, partition\_definition] ...)]

partition\_definition:

PARTITION partition\_name

[VALUES

{LESS THAN {(expr | value\_list) | MAXVALUE}

|

IN (value\_list)}]

[[STORAGE] ENGINE [=] engine\_name]

[COMMENT [=] 'comment\_text' ]

[DATA DIRECTORY [=] 'data\_dir']

[INDEX DIRECTORY [=] 'index\_dir']

[MAX\_ROWS [=] max\_number\_of\_rows]

[MIN\_ROWS [=] min\_number\_of\_rows]

[TABLESPACE [=] tablespace\_name]

[NODEGROUP [=] node\_group\_id]

[(subpartition\_definition [, subpartition\_definition] ...)]

subpartition\_definition:

SUBPARTITION logical\_name

[[STORAGE] ENGINE [=] engine\_name]

[COMMENT [=] 'comment\_text' ]

[DATA DIRECTORY [=] 'data\_dir']

[INDEX DIRECTORY [=] 'index\_dir']

[MAX\_ROWS [=] max\_number\_of\_rows]

[MIN\_ROWS [=] min\_number\_of\_rows]

[TABLESPACE [=] tablespace\_name]

[NODEGROUP [=] node\_group\_id]

select\_statement:

[IGNORE | REPLACE] [AS] SELECT ... (Some valid select statement)

#有点多,主要是一些细节,，比如数据类型都罗列出来了,其实建表很简单,需要什么填什么即可。

举例1:

mysql> create table tb1(id int(2) auto\_increment,name varchar(10),age tinyint(1),primary key(id));

Query OK, 0 rows affected (0.08 sec)

Query OK, 0 rows affected (0.01 sec)

mysql> desc tb1;

+-------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+-------+-------------+------+-----+---------+----------------+

| id | int(2) | NO | PRI | NULL | auto\_increment |

| name | varchar(10) | YES | | NULL | |

| age | tinyint(1) | YES | | NULL | |

+-------+-------------+------+-----+---------+----------------+

3 rows in set (0.00 sec)

举例2:生产场景

use sns;

set names utf8;

CREATE TABLE 'subject\_comment\_manager' (

'subject\_comment\_manager\_id' bigint(lZ) NOT NULL auto\_increment COMM ENT '主键'，

'subject\_type' tinyint(2) NOT NULL COMMENT '素材类型'， 'subject\_pKmary\_key' varchar(255) NOT NULL COMMENT '素材的主键'， 'subject\_title' varchar(255) NOT NULL COMMENT '素材的名称'， 'edit\_user\_irLck' varchar(64) default NULL COMMENT '修改人’， 'edit\_user\_time' timestamp NULL default NULL COMMENT '修改时间'， 'edit\_comment' varchar(255) default NULL COMMENT '修改的理由'，

'state' tinyint(l) NOT NULL default '1' COMMENT '0 代表关闭，1 代表正常'， PRIMARY KEY ('subject\_comment\_manager\_id'),

KEY 'IDX\_PRIMARYKEY' ('subject\_pKmary\_key'(32)), #<==括号内的 32 表示

对前32个字符做前缀索引。

KEY 'IDX\_SUBJECT\_TITLE' C'subject\_title'(B2))

)ENGINE=InnoDB AUT0\_INCREMENT=1 DEFAULT CHARSET=utf8；

主键自增长，数据类型合适非空，对于每一个字段都要加备注，指定主键，指定前缀普通所以， 指定表存储引擎，指定自增长步长，默认字符集。

**6.1.6drop 相关**

help查看关于drop相关的命令:

DROP DATABASE

DROP EVENT

DROP FUNCTION

DROP FUNCTION UDF

DROP INDEX

DROP LOGFILE GROUP

DROP PROCEDURE

DROP SERVER

DROP TABLE

DROP TABLESPACE

DROP TRIGGER

DROP USER

DROP VIEW

标红的即接触过的,分别是删除数据库,删除索引,删除表,删除用户。

**6.1.5.1 drop index**

Syntax:

DROP [ONLINE|OFFLINE] INDEX index\_name ON tbl\_name

DROP INDEX drops the index named index\_name from the table tbl\_name.

This statement is mapped to an ALTER TABLE statement to drop the index.

See [HELP ALTER TABLE].

#指定表名索引名删除索引;这个语句映射到alter table删除索引的语句,也就是相当于:alter table tbl\_name drop {index I key} index\_name

举例等后面讲了索引再来补充

**6.1.5.2 drop table**

Syntax:

DROP [TEMPORARY] TABLE [IF EXISTS]

tbl\_name [, tbl\_name] ...

[RESTRICT | CASCADE]

#删除整张表

**6.1.5.3 drop user**

Syntax:

DROP USER user [, user] ...

#MySQL中的user并非只是用户名，而是user@host一整个条目。

**6.1.7 alter 相关**

查看 help alter:

ALTER DATABASE

ALTER EVENT

ALTER FUNCTION

ALTER LOGFILE GROUP

ALTER PROCEDURE

ALTER SERVER

ALTER TABLE

ALTER TABLESPACE

ALTER VIEW

#目前仅了解标红的，一个修改数据库，一个修改表

**6.1.7.1 alter database**

Syntax:

ALTER {DATABASE | SCHEMA} [db\_name]

alter\_specification ...

ALTER {DATABASE | SCHEMA} db\_name

UPGRADE DATA DIRECTORY NAME

alter\_specification:

[DEFAULT] CHARACTER SET [=] charset\_name

| [DEFAULT] COLLATE [=] collation\_name

# alter database可以修改数据库属性,主要是修改字符集。语法2是升级低版本的数据库。

#### 6.1.7.2 alter table

Syntax:

ALTER [ONLINE | OFFLINE] [IGNORE] TABLE tbl\_name

[alter\_specification [, alter\_specification] ...]

[partition\_options]

alter\_specification:

table\_options

| ADD [COLUMN] col\_name column\_definition 🡨增加字段

[FIRST | AFTER col\_name ]

| ADD [COLUMN] (col\_name column\_definition,...)

| ADD {INDEX|KEY} [index\_name] 🡨增加索引

[index\_type] (index\_col\_name,...) [index\_option] ...

| ADD [CONSTRAINT [symbol]] PRIMARY KEY

[index\_type] (index\_col\_name,...) [index\_option] ...

| ADD [CONSTRAINT [symbol]]

UNIQUE [INDEX|KEY] [index\_name]

[index\_type] (index\_col\_name,...) [index\_option] ...

| ADD FULLTEXT [INDEX|KEY] [index\_name] 🡨增加全文索引

(index\_col\_name,...) [index\_option] ...

| ADD SPATIAL [INDEX|KEY] [index\_name]

(index\_col\_name,...) [index\_option] ...

| ADD [CONSTRAINT [symbol]]

FOREIGN KEY [index\_name] (index\_col\_name,...)

reference\_definition

| ALTER [COLUMN] col\_name {SET DEFAULT literal | DROP DEFAULT}

| CHANGE [COLUMN] old\_col\_name new\_col\_name column\_definition

[FIRST|AFTER col\_name] 🡨修改字段名及类型

| MODIFY [COLUMN] col\_name column\_definition

[FIRST | AFTER col\_name] 🡨修改字段定义

| DROP [COLUMN] col\_name🡨删除字段

| DROP PRIMARY KEY🡨删除主键

| DROP {INDEX|KEY} index\_name🡨删除索引

| DROP FOREIGN KEY fk\_symbol

| MAX\_ROWS = rows

| DISABLE KEYS

| ENABLE KEYS

| RENAME [TO|AS] new\_tbl\_name

| ORDER BY col\_name [, col\_name] ... 🡨排序

| CONVERT TO CHARACTER SET charset\_name [COLLATE collation\_name] 🡨转换字符集

| [DEFAULT] CHARACTER SET [=] charset\_name [COLLATE [=] collation\_name]

| DISCARD TABLESPACE

| IMPORT TABLESPACE

| FORCE

| ADD PARTITION (partition\_definition)

| DROP PARTITION partition\_names

| TRUNCATE PARTITION {partition\_names | ALL}

| COALESCE PARTITION number

| REORGANIZE PARTITION [partition\_names INTO (partition\_definitions)]

| ANALYZE PARTITION {partition\_names | ALL}

| CHECK PARTITION {partition\_names | ALL}

| OPTIMIZE PARTITION {partition\_names | ALL}

| REBUILD PARTITION {partition\_names | ALL}

| REPAIR PARTITION {partition\_names | ALL}

| PARTITION BY partitioning\_expression

| REMOVE PARTITIONING

index\_col\_name:

col\_name [(length)] [ASC | DESC]

index\_type:

USING {BTREE | HASH}

index\_option:

KEY\_BLOCK\_SIZE [=] value

| index\_type

| WITH PARSER parser\_name

| COMMENT 'string'

table\_options:

table\_option [[,] table\_option] ... (see CREATE TABLE options)

partition\_options:

(see CREATE TABLE options)

#alter table的内容比较多,我们用的比较多的是添加删除字段、索引、引擎、字符集等

**6.2 DCL数据控制语言**

**6.2.1 grant授权语句**

Syntax:

GRANT

priv\_type [(column\_list)]

[, priv\_type [(column\_list)]] ...

ON [object\_type] priv\_level

TO user\_specification [, user\_specification] ...

[REQUIRE {NONE | ssl\_option [[AND] ssl\_option] ...}]

[WITH with\_option ...]

GRANT PROXY ON user\_specification

TO user\_specification [, user\_specification] ...

[WITH GRANT OPTION]

object\_type:

TABLE

| FUNCTION

| PROCEDURE

priv\_level:

\*

| \*.\*

| db\_name.\*

| db\_name.tbl\_name

| tbl\_name

| db\_name.routine\_name

user\_specification:

user

[

IDENTIFIED BY [PASSWORD] 'password'

| IDENTIFIED WITH auth\_plugin [AS 'auth\_string']

]

ssl\_option:

SSL

| X509

| CIPHER 'cipher'

| ISSUER 'issuer'

| SUBJECT 'subject'

with\_option:

GRANT OPTION

| MAX\_QUERIES\_PER\_HOUR count

| MAX\_UPDATES\_PER\_HOUR count

| MAX\_CONNECTIONS\_PER\_HOUR count

| MAX\_USER\_CONNECTIONS count

example:

CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';

GRANT ALL ON dbl.\* TO 'jeffrey'@'localhost';

GRANT SELECT ON db2.invoice TO 'jeffrey'@'loocalhost';

GRANT USAGE ON \*.\* TO 'jeffrey'@'localhost' WITH MAX\_QUERIES\_PER\_HOUR 90;

#简单来说就是：grant权限on数据库.表to用户@主机identitied by密码with grant option

举例1:

mysql> grant select,insert,update,delete on monitor.\* to monitor@'192 168.16.%' identified by '123456';

Query OK, 0 rows affected (0.00 sec)

举例2:

mysql> grant all on \*.\* to test@'10.0.0.0/255.255.255.0' identified by 'test123'; #🡨10.0.0.0/255.255.255.0也可写成10.0.0.%

举例3:

mysql> grant all on \*.\* to admin@'192.168.16.%' identified by '123456' with grant option;

**6.2.2 revoke撤权语句**

Syntax:

REVOKE

priv\_type [(column\_list)]

[,priv\_type [(column\_list)]] ON [object\_type] priv\_level FROM user [, user]...

REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user]...

REVOKE PROXY ON user FROM user [, user]...

For example:

REVOKE INSERT ON \*.\* FROM 'jeffrey'@'localhost'

#简单说就是revoke权限on数据库from用户@主机

举例:

mysql> revoke drop on \*•\* from admin@’192.168.16.%’;

Query OK, 0 rows affected (0.00 sec)

**6.2.3commit语句与 rollback语句**

开启事务后有效，提交和回滚。

**6.3 DML数据操作语言**

**6.3.1 插入语句insert**

Syntax:

INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE]

[INTO] tbl\_name [(col\_name,...)]

{VALUES | VALUE} ({expr | DEFAULT},...),(...),...

[ ON DUPLICATE KEY UPDATE

col\_name=expr

[, col\_name=expr] ... ]

Or:

INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE]

[INTO] tbl\_name

SET col\_name={expr | DEFAULT}, ...

[ ON DUPLICATE KEY UPDATE

col\_name=expr

[, col\_name=expr] ... ]

Or:

INSERT [LOW\_PRIORITY | HIGH\_PRIORITY] [IGNORE]

[INTO] tbl\_name [(col\_name,...)]

SELECT ...

[ ON DUPLICATE KEY UPDATE

col\_name=expr

[, col\_name=expr] ... ]

举例1:

mysql> insert into student(id,name,age,dept) values(5,'xiedi',25,'175 cm'),(6,'liyanchao',22,'173cm');

Query OK, 2 rows affected (0.00 sec)

Records: 2 Duplicates: 0 Warnings: 0

举例2:

mysql> insert into student values(7,'xiedi2',25,'175cm');

Query OK, 1 row affected (0.00 sec)

注:多条语句合并插入有利于提升效率,因为插入需要等待锁,打开表的过程,如果分开插入,这个过程重复多次,耗费时间。

**6.3.2 查询语句select**

Syntax:

SELECT

[ALL | DISTINCT | DISTINCTROW ]

[HIGH\_PRIORITY]

[STRAIGHT\_JOIN]

[SQL\_SMALL\_RESULT] [SQL\_BIG\_RESULT] [SQL\_BUFFER\_RESULT]

[SQL\_CACHE | SQL\_NO\_CACHE] [SQL\_CALC\_FOUND\_ROWS]

select\_expr [, select\_expr ...]

[FROM table\_references

[WHERE where\_condition]

[GROUP BY {col\_name | expr | position}

[ASC | DESC], ... [WITH ROLLUP]]

[HAVING where\_condition]

[ORDER BY {col\_name | expr | position}

[ASC | DESC], ...]

[LIMIT {[offset,] row\_count | row\_count OFFSET offset}]

[PROCEDURE procedure\_name(argument\_list)]

[INTO OUTFILE 'file\_name'

[CHARACTER SET charset\_name]

export\_options

| INTO DUMPFILE 'file\_name'

| INTO var\_name [, var\_name]]

[FOR UPDATE | LOCK IN SHARE MODE]]

1)査询整张表:

mysql> select \* from student;

+----+----------+------+--------+

| id | name | age | dept |

+----+----------+------+--------+

| 1 | troy | 23 | 175cm |

| 2 | tony | 24 | 177cm |

| 3 | terry | 25 | 180cm |

| 4 | lichuang | 38 | 150cm |

| 5 | xiedi | 25 | 175cm |

| 6 | liyanchao| 22 | 173cm |

| 7 | xiedi2 | 25 | 175cm |

+----+----------+------+--------+

7 rows in set (0.00 sec)

2)通过limit关键字限制显示记录:

mysql> select \* from student limit 3;

+----+----------+------+--------+

| id | name | age | dept |

+----+----------+------+--------+

| 1 | troy | 23 | 175cm |

| 2 | tony | 24 | 177cm |

| 3 | terry | 25 | 180cm |

+----+----------+------+--------+

3 rows in set (0.00 sec)

mysql> select \* from student limit 2,3; 🡨start\_pos【起始 0】，后面代表长度

+----+----------+------+--------+

| id | name | age | dept |

+----+----------+------+--------+

| 3 | terry | 25 | 180cm |

| 4 | lichuang | 38 | 150cm |

| 5 | xiedi | 25 | 175cm |

+----+----------+------+--------+

3 rows in set (0.00 sec)

3)通过where指定条件,order by排序:

mysql> select id,name,age from student where age < 25 order by age;

+----+----------+------+--------+

| id | name | age | dept |

+----+----------+------+--------+

| 6 | liyanchao| 22 | 173cm |

| 1 | troy | 23 | 175cm |

| 2 | tony | 24 | 177cm |

+----+----------+------+--------+

3 rows in set (0.00 sec)

mysql> select id,name,age from student where age < 25 order by age desc;

+----+----------+------+--------+

| id | name | age | dept |

+----+----------+------+--------+

| 2 | tony | 24 | 177cm |

| 1 | troy | 23 | 175cm |

| 6 | liyanchao| 22 | 173cm |

+----+----------+------+--------+

3 rows in set (0.00 sec)

#注意:where后面可以指定字段,可以是比较操作符;排序默认是升序asc,可以通过desc指定为降序。

4)导出结果到本地系统文件:

mysql> select \* from student where age >24 order by age asc INTO OUTFILE '/tmp/a.txt';

Query OK, 4 rows affected (0.00 sec)

[1 root@mysql-server-01 # cat /tmp/a.txt

3 terry 25 180cm

5 xiedi 25 175cm

7 xiedi2 25 175cm

4 lichuang 38 150cm

**6.3.3 更新语句update**

Syntax:

Single-table syntax:

UPDATE [LOW\_PRIORITY] [IGNORE] table\_reference

SET col\_name1={expr1|DEFAULT} [, col\_name2={expr2|DEFAULT}] ...

[WHERE where\_condition]

[ORDER BY ...]

[LIMIT row\_count]

Multiple-table syntax:

UPDATE [LOW\_PRIORITY] [IGNORE] table\_references

SET col\_name1={expr1|DEFAULT} [, col\_name2={expr2|DEFAULT}] ...

[WHERE where\_condition]

说明:update分为单表语法和多表语法。操作是需要进行,必须指定条件,否则一个字段的值都会更改。

案例1:将lichuang age改为25

mysql> update student set age = 23 where name='lichuang'

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> select \* from student where name='lichuang';

+----+----------+------+--------+

| id | name | age | dept |

+----+----------+------+--------+

| 4 | lichuang | 23 | 150cm |

+----+----------+------+--------+

1 row in set (0.00 sec)

案例2:利用-U参数来防止误操作

[66 root@mysql-server-01 3306]# mysql -S /data/B306/mysql.sock -uroot -p123456 -U

mysql> update student set age = 0 ;

ERROR 1175 CHY000): You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column

#啊误操作了,没事有-U保护,再也不用担心了。

**6.3.4删除语句delete**

Syntax:

Single-table syntax:

DELETE [LOW\_PRIORITY] [QUICK] [IGNORE] FROM tbl\_name

[WHERE where\_condition]

[ORDER BY ...]

[LIMIT row\_count]

Multiple-table syntax:

DELETE [LOW\_PRIORITY] [QUICK] [IGNORE]

tbl\_name[.\*] [, tbl\_name[.\*]] ...

FROM table\_references

[WHERE where\_condition]

Or:

DELETE [LOW\_PRIORITY] [QUICK] [IGNORE]

FROM tbl\_name[.\*] [, tbl\_name[.\*]] ...

USING table\_references

[WHERE where\_condition]

delete根据条件删除记录;如果条件没有指定,也会删除整个表,容易误操作。

mysql> delete from student where name = 'xiedi2';

Query OK, 1 row affected (0.00 sec)

mysql> select \* from student;

+----+----------+------+--------+

| id | name | age | dept |

+----+----------+------+--------+

| 1 | troy | 23 | 175cm |

| 2 | tony | 24 | 177cm |

| 3 | terry | 25 | 180cm |

| 4 | lichuang | 38 | 150cm |

| 5 | xiedi | 25 | 175cm |

| 6 | liyanchao| 22 | 173cm |

+----+----------+------+--------+

6 row in set (0.00 sec)

**6.3.5 truncate删除表和delete删除表的区别**

1)truncate 介绍:

Syntax:

TRUNCATE [TABLE] tbl.name

TRUNCATE TABLE empties a table completely. It requires the DROP privilege.

2)truncate 和 delete 区别:

⮚truncate是物理删除,属于DDL语句;delete from属于逻辑删除,属于DML语句

⮚truncate速度快;delete速度慢

**6.3.6 改名语句rename**

Syntax:

RENAME TABLE tbl\_name TO new\_tbl\_name

[,tbl\_name2 TO new\_tbl\_name2]…

操作：

mysql> rename table student to stu;

Query OK, 0 rows affected (0.01 sec)

mysql> show tables;

+-------------------------+

| Tables\_in\_test |

+-------------------------+

| stu |

| subject\_comment\_manager |

| tb1 |

+-------------------------+

3 rows in set (0.00 sec)

**6.4 MySQL 索引**

**6.4.1索引的基本概念**

**6.4.1.1什么是索引**

索引就像书的目录一样,有了目录我们可以很快的知道这本书的基本内容和结构,数据索引也一样,它可以加快数据表的查询速度。这是mysql优化的重要内容之一。

**6.4.1.2什么是主键**

主键就相当与书本的页码,唯一,能够标识数据位置。

主键有以下两个特点：

⮚ 惟一地标识一行。

⮚ 作为一个可以被外键有效引用的对象。

### 6.4.2如何创建/删除索引

1)建表是建立主键索引和普通索引:

建表建立索引:

create table student(

id int(4) not null AUTO\_INCREMENT, #🡨自增

name char(20) not null,

age tinyintC2) NOT NULL default '0',

dept varchar(16) default NULL,

primary key(id), #🡨主键索引

KEY index\_name(name) #🡨普通索引

);

查看表结构：

mysql> desc student;

+------+-------------+----+-----+---------+--------------+

|Field | Type |Null| Key | Default | Extra |

+------+-------------+----+-----+---------+--------------+

| id | int(4) | NO | PRI | NULL |auto\_increment|

| name | char(20) | NO | MUL | NULL | |

| age | tinyint(2) | NO | | 0 | |

| dept | varchar(16) |YES | | NULL | |

+------+-------------+----+-----+---------+--------------+

4 rows in set (0.00 sec)

#PRI代表主键索引；MUL代表普通索引

查看索引:

mysql> show index from student\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: student

Non\_unique: 0

Key\_name: PRIMARY

Seq\_in\_index: 1

Column\_name: id

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null:

Index\_type: BTREE

Comment:

Index\_comment:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*2.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: student

Non\_unique: 1

Key\_name: index\_name

Seq\_in\_index: 1

Column\_name: name

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null:

Index\_type: BTREE

Comment:

Index\_comment:

2 rows in set (0.00 sec)

2)通过alter或者create增加索引:

通过alter增加索引：

mysql> alter table student add index index\_dept(dept);

Query OK, 0 rows affected (0.05 sec)

Records: 0 Duplicates: 0 Warnings: 0

通过create增加索引：

mysql> create index index\_age on student(age);

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

查看索引:

mysql> show index from student\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: student

Non\_unique: 0

Key\_name: PRIMARY

Seq\_in\_index: 1

Column\_name: id

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null:

Index.type: BTREE

Comment:

Index.comment:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*2.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: student

Non\_unique: 1

Key\_name: index\_name

Seq\_in\_index: 1

Column\_name: name

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null:

Index.type: BTREE

Comment:

Index.comment:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: student

Non\_unique: 1

Key\_name: index\_dept

Seq\_in\_index: 1

Column\_name: dept

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null: YES

Index.type: BTREE

Comment:

Index.comment:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*4.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: student

Non\_unique: 1

Key\_name: index\_age

Seq\_in\_index: 1

Column\_name: age

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null:

Index.type: BTREE

Comment:

Index.comment:

4 rows in set C0.00 sec)

3)通过alter或者drop删除索引:

通过alter删除索引：

mysql> alter table student drop index index\_dept;

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

通过drop删除索引:

mysql> drop index index\_age on student;

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

查看索引：

mysql> desc student;

+------+-------------+----+-----+---------+--------------+

|Field | Type |Null| Key | Default | Extra |

+------+-------------+----+-----+---------+--------------+

| id | int(4) | NO | PRI | NULL |auto\_increment|

| name | char(20) | NO | MUL | NULL | |

| age | tinyint(2) | NO | | 0 | |

| dept | varchar(16) |YES | | NULL | |

+------+-------------+----+-----+---------+--------------+

4 rows in set C0.00 sec)

4)创建前缀索引:

当遇到表中比较大的列时,可以使用前几列接近唯一的前n个字符作为索引,节省索引空间空间。

创建前缀索引:

mysql> alter table student add index index\_name(name(8));

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

#当然也可以通过 create index index\_name on student(name(8));来创建

查看索引：

mysql> show index from student\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: student

Non\_unique: 0

Key\_name: PRIMARY

Seq\_in\_index: 1

Column\_name: id

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null:

Index.type: BTREE

Comment:

Index.comment:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*2.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: student

Non\_unique: 1

Key\_name: index\_name

Seq\_in\_index: 1

Column\_name: name

Collation: A

Cardinality: 0

Sub\_part: 8

Packed: NULL

Null:

Index.type: BTREE

Comment:

Index\_comment:

2 rows in set C0.00 sec)

### 6.4.3联合索引

1)什么是联合索引:

以多个字段建立索引,称之为联合索引。

2)联合索引的特性:

联合索引有前缀特性：

三个字段：

ABC

只有以下列组合才会走索引：

A

A，B

A，B，C

由这个特性，我们设置索引的时候需要注意：

1.经常用于索引的字段放前面

2.多个字段组合，唯一值是多字段的组合，所以唯一值增多。

3)如何创建联合索引:

mysql> drop index index\_name on student;

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> create index index\_name\_dept on student(name(8),dept);

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> desc student;

+------+-------------+----+-----+---------+--------------+

|Field | Type |Null| Key | Default | Extra |

+------+-------------+----+-----+---------+--------------+

| id | int(4) | NO | PRI | NULL |auto\_increment|

| name | char(20) | NO | MUL | NULL | |

| age | tinyint(2) | NO | | 0 | |

| dept | varchar(16) |YES | | NULL | |

+------+-------------+----+-----+---------+--------------+

4 rows in set C0.00 sec)

mysql> show index from student\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*2.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: student

Non\_unique: 0

Key\_name: PRIMARY

Seq\_in\_index: 1

Column\_name: id

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null:

Index.type: BTREE

Comment:

Index.comment:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*2.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: student

Non\_unique: 1

Key\_name: index\_name\_dept

Seq\_in\_index: 1 #🡨联合索引标识，说明该字段位于联合索引什么位置。

Column\_name: name

Collation: A

Cardinality: 0

Sub\_part: 8

Packed: NULL

Null:

Index.type: BTREE

Comment:

Index.comment:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: student

Non\_unique: 1

Key\_name: index\_name\_dept

Seq\_in\_index: 2

Column\_name: dept

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null: YES

Index.type: BTREE

Comment:

Index\_comment:

3 rows in set (0.00 sec)

注意：我们可以看到,普通索引,前缀索引,联合索引的建立方法都是一致的,可以通过 create index i\_ndex\_name on table\_name(columnl(suffix\_bi t),...）方式建立;也可以通过 alter table table\_name add index index\_name(columnl(suffix\_bits)，...）方式创建。联合索引也是支持前缀索引的。除此之外还有唯一索引等,属于非重点,需时学习。

### 6.4.4MySQL创建索引总结

创建索引总结：

1.索引类似书籍的目录,会加快数据查询的速度。

2.要在表的字段上创建索引。

3.索引会加快查询额速度,但是会影响更新的速度,因为更新要维护索引数据。

4.索引列并不是越多越好,要在频繁查询的where后的条件上创建索引。

5.小表或者重复值很多的字段上可以不建立索引,要在大表以及重复值少的条件字段上创建索引。

6.多个字段的联合索引有前缀生效的特性。

7.当字段内容前N个字符已经接近唯一时,可以对前N个字符创建索引。

8.索引从工作方式分:有主键,唯一,普通索引

9.索引类型BTREE (默认）和哈希（适合做缓存（内存数据库））。

### 6.4.5explain测试查询是否走索引

Syntax:

{EXPLAIN | DESCRIBE | DESC}

tbl\_name [col\_name | wild]

{EXPLAIN | DESCRIBE | DESC}

[explain\_type] SELECT select\_options

explain\_type:

EXTENDED

| PARTITIONS

准备测试数据:

mysql> insert into student values(l,’troy',23,’175cm’），（2,’tony’,24,'177cm'),（3,'terry',25,'180cm');

Query OK, 3 rows affected (0.00 sec)

Records: 3 Duplicates: 0 Warnings: 0

mysql> select \* from student;

+----+------+-----+------+

| id | name | age | dept |

+----+------+-----+------+

| 1 | troy | 23 | 175cm |

| 2 | tony | 24 | 177cm |

| 3 |terry | 25 | 180cm |

+----+------+-----+------+

3 rows in set (0.00 sec)

测试select查询1:

mysql> explain select \* from student where name='troy'\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 1

select\_type: SIMPLE

table: student

type: ref

possible\_keys: index\_name\_dept

key: index\_name\_dept 🡨所走索引

key\_len: 24

ref: const

rows: 1🡨遍历的行数

Extra: Using where

1 row in set (0.00 sec)

#通过explain可以看select查询是否走索引。

测试select查询2:

mysql> explain select \* from student where name like ‘tro%\*’\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

select\_type: SIMPLE

table: student

type: range

possible\_keys: index\_name\_dept

key: index\_name\_dept

key-len: 24

ref: NULL

rows: 1

Extra: Using where

1 row in set (0.00 sec)

#模糊查询％号在后面，也是会走索引的。

测试select查询3:

mysql> explain select \* from student where name like ‘%roy’\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

select\_type: SIMPLE

table: student

type: ALL

possible\_keys: NULL

key: NULL

key\_len: NULL

rows: 4

Extra: Using where

1 row in set (0.00 sec)

#模糊查询％号在前面，不会走索引。

### 6.4.6 企业案例一：利用explain优化SQL语句

1)抓慢SQL:

a.show full processlist\G (现场查看执行慢的语句)

[128 root@template 3306]# mysql -S /mysql.sock -uroot -pl23456 -e "show full processlist\G"

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Id: 17

User: root

Host: localhost

db: NULL

Command: Query

Time: 0

State: NULL

Info: show full processlistb.

b.记录慢查询日志

设置启用慢查询记录的参数到my. cnf的mysqld段中:

long\_query\_time = 1

log-slow-queries = /data/3306/slow.log

log\_queries\_not\_using\_indexes

C.切割慢查询日志

原理：首先移走现有慢查询日志,然后通过mysqladmin的flush-logs,重新生成慢查询日志;然后将该脚本定时执行。

#!/bin/bash

DATE=$(date +%F)

cd /data/3306

/bin/mv slow.log /backup/${DATE}\_slow.log

mysqladmin -uroot -pl23456 -S /data/3306/mysql.sock flush-logs

mysqlsla /backup/${DATE}\_slow.log >/backup/new\_${DATE}\_slow.log

mail -s "${DATE}"\_slow.log e-mail < /backup/new\_${DATE}\_slow.log

d.分析慢查询日志

##mysqlsla方便查看统计慢查询日志

http://blog.itpub.net/7607759/viewspace-692828/

e.通过mysqlla处理后的结果

邮件内容：

直接阅读的操作体验很不好，使用mysqlsla处理后，结果呈现如下：

Count: 23 (8.52%)

Time: 102 s total, 4.434783s avg, 3s to 7s max

95% of Time : 88s total, 4.190476s avg, 3s to 6s max

Lock Time (s) : 0 total, 0 avg, 0 to 0 max (0.00%)

95% of Lock : 0 total, 0 avg, 0 to 0 max

Rows sent : 1 avg, 1 to 1 max (0.02%)

Rows examined : 11.5Bk avg, 5.70k to 17.60k max (1.07%)

Database : jssdb

Users:

junsansi@ 192.168.1.27 : 86.96% (20) of query, 11.11% (30) of all users

jss@ 192.168.1.26 : 13.04% (B) of query, 2.96% (8) of all users Query abstract:

SELECT MIN(doc\_his\_id) AS doc\_his\_id FROM t\_\*\*\*\*\*\*\*\* WHERE doc\_his一ist eammate=N AND doc\_his\_editor\_user\_id\_encrypt=TST;

Query sample:

select min(DOC\_HIS\_ID) AS DOC\_HIS\_ID from t\_\*\*\*\*\*\*\*\* where DOC\_HIS\_IST EAMMATE=1 and DOC\_HIS\_EDITOR\_USER\_ID\_ENCRYPT='nfEACAwQEW2MICAN2';

2) explain检测索引执行情况:

根据上面抓取的慢查询，查看查询是否走索引

3)对需要建索引的列建立索引:

首先统计where条件字段中的唯一值数量，然后建立索引或者联合索引（与开发、DBA合作完成）

4)分析慢查询的工具（每天早晨发邮件):

使用mysqlsla工具+shell+mail实现自动化。

**7 MySQL字符集**

**7.1什么是字符集**

字符（Character)是各种文字和符号的总称,包括各国家文字、标点符号、图形、符号、数字等。字符集（Character set)是多个字符的集合,字符集种类较多,每个字符集包含的字符个数不同,常见字符集名称：ASCII字符集、GB2312字符集、BIG5字符集、GB18030字符集、Unicode字符集等。计算机要准确的处理各种字符集文字,需要进行字符编码,以便计算机能够识别和存储各种文字。

推荐使用UTF-8字符集,UTF-8根据英文字符还是中文字符自动选取编码长度1~3个字节。

**7.2 MySQL字符集查看**

MySQL的字符系统由多个字符集影响，通过以下命令查看MySQL字符集:

mysql> show variables like 'character\_Set\\_%';

+---------------------------+----------+

| Variable\_name | Value |

+---------------------------+----------+

|character\_set\_client | latinl |

|character\_set\_connection | latinl |

|character\_set\_database | utf8 |

|character\_set\_filesystem | binary |

|character\_set\_results | latinl |

|character\_set\_server | utf8 |

|character\_set\_system | utf8 |

+---------------------------+----------+

7 rows in set (0.00 sec)

#红色表示跟客户端相关的字符集有关;蓝色表示跟MySQL服务端有关。

#客户端、连接、返回结果字符集是是由MySQL客户端控制的,默认为latinl;服务、数据库字符集是MySQL服务端控制的,与编译选项有关,这里指定了utf8

**7.3 MySQL字符集乱码原因**

⮚MySQL客户端字符集本身不对,插入就使用了和MySQL服务端不一致的字符集,那么就插入了乱码数据,即使调整了字符集,也无法正确显示字符。

⮚MySQL客户端字符集和MySQL服务端字符集一样时插入数据,可能因为查看时客户端字符集不正确,无法正确显示,造成字符显示乱码,通过调整MySQL客户端字符集解决。

具体影响MySQL乱码的字符集的可能因素:

1.Linux客户端字符集（推荐UTF8)

SecureCRT的字符集

2.linux 服务端字符集（LANG="zh\_CN.UTF-8")

[11 root@mysql-server-01 3306]# cat /etc/sysconfig/il8n LANG="zh\_CN.UTF-8"

SYSFONT="latarcyrheb-sunl6"

3.数据库客户端字符集（UTF8)

方法一：set names utf8;临时生效

show variables like 'character\\_set\\_%';

方法二：登录时指定参数--default-character-set=utf8

方法三：多实例：在/etc/my.cnf 中配置[client]default-character-set=gbk

4.数据库服务端字符集

方法一：修改my.cnf配置文件对于版本5.5，在[mysqld]段中增加character-set-server=utf8

方法二：编译时指定

-DDEFAULT\_CHARSET=utf8 \

-DDEFAULT\_COLLATION=utf8\_general\_ci \

-DEXTRA\_CHARSETS=gbk,gb2312,utf8,ascii \

5.建数据库指定数据库的字符集

建库字符集，默认使用MySQL数据库字符集

create database oldboy character set utf8 collate utf8\_general\_ci:

alter database oldboy CHARACTER SET utf8 COLLATE utf8\_general\_ci;

6.表的字符集

表的字符集默认使用数据库的字符集，也可以建表是指定字符集。

alter table test character set = utf8 collate utf8\_general\_ci：

7.PHP/JAVA程序字符集

程序所用字符集相当MySQL客户端。

比如Discuz的安装包提供多个版本，比如UTF-8, GBK版本等。

**7.4如何更改数据库表字符集**

更改数据库表字符集涉及到原有数据,所以要分两种情况:1.原本没有数据;2源本有数据。

**7.4.1MySQL数据库没有数据**

这种情况直接使用alter语句修改数据库表字符集即可：

alter database oldboy CHARACTER SET utf8 COLLATE utf8\_general\_ci ;

alter table test character set = utf8 collate utf8\_general\_ci;

**7.4.2MySQL数据库原本有数据**

这种情况如果直接修改库表字符集,原有数据会出现问题,此时需要特殊处理:

1.mysqldump数据备份（数据量大的话,先导出表结构【-d】,sed替换）,再导出数据【-t】

2.在表结构中把字符集修改了,比如sed替换

3.修改系统配置,调整字符集 MySQL先把表结构还原,再把数据还原

练练手：通过库表结构数据分离的方式导出修改

#导出MySQL test库表结构

[31 root@mysql-server-01 data]# mysqldump -S /data/3306/mysql.sock -uroot -p123456 -B -d -x test >/server/backup/mysql\_test\_arch.sql

#导出MySQL test库表数据

mysqldump -S /data/3306/mysql.sock -uroot -pl23456 -t -x test >/server/backup/mysql\_test\_data.sql

#批量修改库表结构字符集

[65 root@mysql-server-01 backup]# sed -i 's#utf8#gbk#g' /server/backup /mysql\_test\_arch.sql

#导入MySQL test库表结构

[70 root@mysql-server-01 3308]# mysql -S /data/3308/mysql.sock -uroot -p123456 < /server/backup/mysql\_test\_arch.sql

#导入MySQL test数据

[71 root@mysql-server-01 3308]# mysql -S /data/3308/mysql.sock -uroot -p123456 test < /server/backup/mysql\_test\_data.sql

#查看库类型

mysql> show create database test\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Database: test

Create Database: CREATE DATABASE 'test' /\*!40100 DEFAULT CHARACTER SET gbk \*/

#查看表类型

mysql> show create table subject\_comment\_manager\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: subject\_comment\_manager

Create Table: CREATE TABLE 'subject\_comment\_manager' (

`subject\_comment\_manager\_id` bigint(12) NOT NULL AUTO\_INCREMENT COMM ENT '???',

`subject\_type` tinyint(2) NOT NULL COMMENT '??????',

`subject\_primary\_key` varchar(255) NOT NULL COMMENT '???????', `subject\_title` varchar(255) NOT NULL COMMENT '???????',

`edit\_user\_nick` varchar(64) DEFAULT NULL COMMENT,

'edit\_user\_time' timestamp NULL DEFAULT NULL COMMENT '?????',

`edit\_comment' varchar(255) DEFAULT NULL COMMENT '???????',

`state` tinyint(l) NOT NULL DEFAULT '1' COMMENT '0??????????????',PRIMARY KEY (`subject\_comment\_manager\_id`),

KEY `IDX—PRIMARYKEY` （`subject—primary\_key`（32)),

KEY `IDX\_SUBJECT\_TITLE` （`subject\_title`（32))

)ENGINE=InnoDB DEFAULT CHARSET=gbk

1 row in set (0.00 sec)

#客户端字符集默认为latinl,更改为gbk，再次查看

mysql> set names gbk;

Query OK, 0 rows affected (0.00 sec)

mysql> show create table subject\_comment\_manager\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1.row\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: subject\_comment\_manager

Create Table: CREATE TABLE xsubject\_comment\_manager'(

`subject\_comment\_manager\_id` bigint(12) NOT NULL AUTO\_INCREMENT COMMENT '主键',

`subject\_type` tinyint(2) NOT NULL COMMENT '素材类型'，

`subject\_primary\_key` varchar(255) NOT NULL COMMENT '素材的主',

`subject\_title` varchar(255) NOT NULL COMMENT '素材的名',

'edit\_user\_nick` varchar(64) DEFAULT NULL 'COMMENT',

`edit\_user\_time` timestamp NULL DEFAULT NULL COMMENT '间',

`edit\_comment` varchar(255) DEFAULT NULL COMMENT '理',

`state` tinyint(1) NOT NULL DEFAULT '1' COMMENT '0 代表关闭常’，

PRIMARY KEY ('subject\_comment\_manager\_id'),

KEY `IDX\_PRIMARYKEY` (`subject\_primary\_key`(32)),

KEY `IDX\_SUBJECT\_TITLE` (`subject\_title`(32)),

)ENGINE=InnoDB DEFAULT CHARSET=gbk

1 row in set (0.00 sec)

#这种方法适合大容量数据库,库表结构数据分离,这样sed替换不会处理太多行数据。

**8 MySQL日志系统**

MySQL日志主要包括:错误日志、普通查询日志、慢查询日志以及二进制日志。

**8.1错误日志**

记录着MySQL程序错误的信息,通过错误日志,我们可以找出MySQL无法正常工作的原因,比如配置文件参数错误之类的。

1)配置方法：

方法一：配置文件my.cnf中调整

[mysqld—safe]

log-error=/data/3306/mysql.err

方法二：MySQL启动参数调整

-log-error = /data/3306/mysql.err

**8.2普通查询日志**

普通查询日志会记录每一条SQL请求语句,包括错误语句,调试时可以开启,但是生产场景不要开启,因为会大大影响磁盘IO性能。

1)配置方法:

法1:临时生效

show variables like 'general.log%';

set global general\_log\_file = ON;

set global general\_log\_file=f/data/3306/data/template.log'

法2:永久生效

配置文件：

[mysqld]

general\_log = on

general\_log\_file = /data/3306/data/MySQL\_oldboy.log

**8.3慢查询日志**

慢查询日志,记录着查询时间超过设置阀值的SQL语句以及没有走索引的语句,通过慢查询日志,找到慢SQL,然后优化之,可以大大提高MySQL的性能,在日常工作中也是经常使用的工具之一。

1)配置方法:

方法：通过修改配置文件my.cnf

long\_query\_time = 1

log-slow-queries = /data/3306/slow.log

log\_queries\_not\_using\_indexes

其中long\_query\_time指定慢查询的阀值,超过这个设定的值,就会被记录到慢查询日志当中;log\_queries\_not\_using\_indexes参数将没有走索引的查询语句也记录到慢查询日志当中。关于如何利用慢查询优化,见MySQL索引优化案例一。

**8.4二进制日志**

二进制日志包括binlog日志，relaylog日志；binlog日志记录更新数据库的 SQL语句，主要有以下作用：

1.主从复制

2.增量备份

1)配置方法

方法一：临时配置binlog

show variables like ,%log\_bin%1

set session sql\_log\_bin = OFF;

set global sql\_log\_bin = OFF;

说明：临时关闭binlog可用于恢复数据时,临时关闭binlog,然后source恢复数据库,防止恢复的sql语句写入binlog文件,影响IO以及主从复制等。

方法二：设置配置文件my.cnf永久生效

#设置 binlog:

log-bin = /data/3308/mysql-bin

#设置 relaylog:

relay-log = /data/3308/relay-bin relay-log-info-file = /data/3308/relay-log.info

#从库开启binlog:

log-bin = /data/3308/mysql-bin log-slave-updates

至于具体配置主从复制见前面章节；增量备份看备份章节。

2)清除binlog的方法

reset master

比如从库提主,就可以通过这个命令清空binlog文件;但是如果是正常情况下不要使用，否则会影响主从复制。

**8.5binlog的三种模式**

**8.5.1概念**

1.Row Level

对于每一条记录生成一条sql语句。优点：记录每一行数据修改的细节,且不会出现某些特定情况下的存储过程,或function,以及trigger的调用和触发无法被正确复制的问题。缺点: 记录每一行,会产生大量binlog日志,性能下降。

2.Statement Level

对于每一条修改的语句生成一条sql语句。

优点：减少bin-log日志量,节省IO,提高性能,因为只记录了执行的语句,以及上下文的细节信息。

缺点：在MySQL特殊函数会出现不一致的问题。【MySQL默认binlog】

3.Mixed

分析SQL语句,根据情况在Statement和Row之间选择一种

**8.5.2如何配置**

通过配置文件:

#binlog\_format="STATEMENT"

#binlog\_format="ROW"

binlog\_format=nMIXEDH

临时生效

mysql> SET SESSION binlog\_format = 'STATEMENT';

mysql> SET SESSION binlog\_format = 'ROW';

mysql> SET SESSION binlog\_format = 'MIXED';

mysql> SET GLOBAL binlog\_format = 'STATEMENT';

mysql> SET GLOBAL binlog\_format = 'R0W';

mysql> SET GLOBAL binlog\_format = 'MIXED':

**8.5.3关于如何査看row模式下的binlog文件**

一般查看二进制日志可以使用mysqlbinlog工具,但是查看row模式binlog,其中有base64编码的信息,我们可以通过参数解码后查看：

mysqlbinlog --base64-output="decode-rows" --verbose mysql-bin.000001

实战演练:

1.清空binlog日志，并删除表内容：

mysql>reset master; mysql>delete from stu;

2.查看binlog文件：

# mysqlbinlog mysql-bin.000001

...省...

BINLOG '

lynNVQ/sDAAAZwAAAGsAAAABAAQANS41LjMyLWxvZwAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAA

AAAAAAAAAAAAAAAAAADXKc1VEzgNAAgAEgAEBAQEEgAAVAAEGggAAAAICAgCAA==

'/\*!\*/;

...省略...

3.通过解码查看binlog文件：

# mysqlbinlog --base64-output="decode-rows" --verbose mysql-bin.00000 1

...省...

###DELETE FROM 'test'.'stu'

###WHERE

### @1=1

### @2='troy'

### @3=23

### @4='175cm'

###DELETE FROM 'test'.'stu'

###WHERE

### @1=2

### @2='tony'

### @3=24

### @4='177cm'

...省...

**8.6关于日志切割**

关键是mysqladmin 的 flush-logs 参数:

? flush-logs

Flush all logs.

由mysqladmin的帮助文件,我们知道,该命令会刷新所有的日志,包括错误日志,慢查询日志、binlog日志、以及查询日志。

这里就以慢查询日志为例:

1.首先mv重命名慢查询日志文件

2.然后通过mysqladrrrin flush-logs刷新日志，生成新的日志文件

3.通过日志分析工具分析慢查询日志，比如mysqlsla，生成报告

4.通过邮件或者短信的方式发送给相关人员

5.将该过程写入脚本，然后写入定时任务，每天00:00执行，脚本如下：

#!/bin/bash DATE=$(date +%F)

cd /data/3306

/bin/mv slow.log /backup/${DATE}\_slow.log

mysqladmin -uroot -pl23456 -S /data/3306/mysql.sock flush-logs

mysqlsla /backup/${DATE}\_slow.log >/backup/new\_${DATE}\_slow.log

mail -s "${DATE}"\_slow.log e-mail < /backup/new\_${DATE}\_slow.log

补充一下mysqladmin常用命令：

1.stuas

[1 root@mysql-server-01 ~]# mysqladmin -uroot -pl2B456 -S /data/3306/mysql.sock status

Uptime:3596 Threads:2 Questions:7 Slow queries:0 Opens:33 Flush tables:1 Open tables:26 Queries per second avg: 0.001

记录MySQL服务的运行状态，包括负载，线程，问题，慢查询，打开文件数等等

2.flush-log

用于切割日志，所有日志。

3.processlist

用于查看进程【线程】列表及信息

4.extended-status

显示扩展状态列表

5. -i 1

每1秒钟刷新一次显示

**9全量备份以及增量备份**

**9.1增量工具mysqlbinlog主要用法**

**9.1.1按库恢复**

[20 root@mysql-server-01 3306]# mysqlbinlog -d test mysql-bin.000013

说明：通过-d指定库。

**9.1.2按位置点恢复**

[ZZ root@mysql-server-01 3306]# mysqlbinlog mysql-bin.000013 --start-position=2392 --stop-position=2463

说明：通过--stcirt-position指定起始位置点；通过--stop-position指定结束位置点。

**9.1.3按时间恢复**

mysqlbinlog --start-datetime="2005-12-25 11:25:56" binlog.000003

说明：通过--start-datetime指定开始时间，使用-r选项可以将结果导入新的文件。

**9.2企业备份方案**

**9.2.1按天备份**

备份思路：

每天凌晨使用备份工具进行一次全备，然后其他时候使用监控工具，实时地将bin log推送到备份机上。

应用场景：

适合中小型企业，数据量不是太大。

优点：

a.恢复时间短

b.维护成本低

缺点：

a.占用空间大

b.占用资源多

c.经常影响到用户体验

**9.2.2按周备份**

备份思路：

每周六或者周日【流量低谷时】进行一次全备，然后每天进行一次增量备份

应用场景：

大型公司数据量很大的情况。

优点：

占用空间小、占用系统资源少、无需锁表、用户体验好一些。

缺点：

维护成本高、恢复麻烦，时间长。

**9.2.3关于全量增量的频率**

9.2.3.1中小型公司。全量一般每天一次，业务流量低估执行全备，备份时需锁表。

9.2.3.2单台数据库，如何增量。用rsync (配合定时任务频率大点，或者inotify，主从复制）把所有的binlog备份到远程服务器上，尽量做主从复制。

9.2.3.3大公司周备，每周六凌晨一次全量，下周日-周六凌晨前都是增量。

优点:节省时间，减少备份压力。

缺点:增加binlog文件副本太多,还原会很麻烦。

9.2.3.4一主五从,会有一个从库做备份,延时同步。

3) 何时需要用到全备数据？

1、迁移或者升级数据库时。

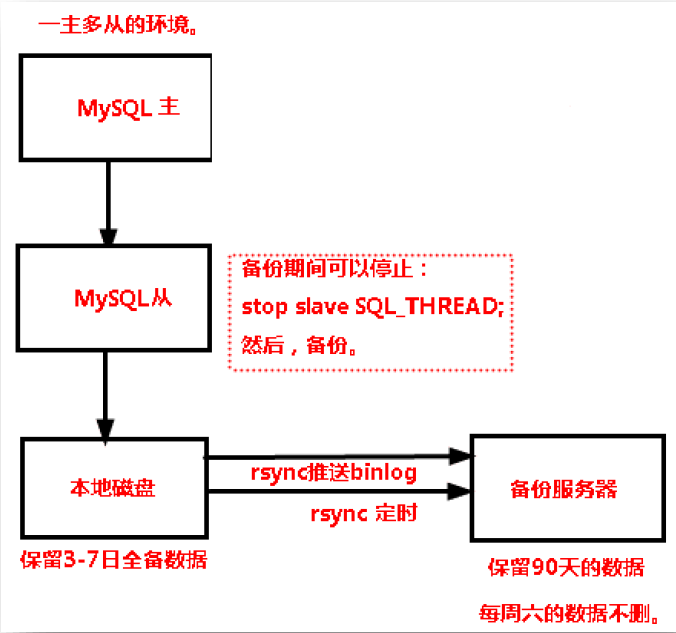
2、增加从库时。

3、因为硬件或者特殊情况。主库或者从库宕机，主从就可以互相切换。无需备份。

4、人为的DDL,DML语句，主库没办法，从库都会执行。此时需要备份。

5、跨机房灾备，需要全量备份到异地。

4)企业备份框架



6)增量恢复总结

1.条件

全备份以及全备之后的binlog

2.关键点：G点确定全备份之后时刻binlog文件及位置

Is -Irt mysql-bin.log >dump.log

3.恢复时候尽量对外停止服务

需要恢复的数据放在一起,写好恢复语句,进行恢复。

4.备份恢复的时候可以将sql\_log\_bin关闭.恢复完再打开

5.未雨绸缪，多事前下功夫,而不是事后补救。

其他方案:

停止一个从库，然后再主库刷新binlog,把mysql-bin.000014恢复成bin.sql(去掉d rop语句）

2、把全备bak\_2014-ll-06.sql及10点前的增量bin.sql恢复到从库。

3、切换到从库提供服务。

数据丢多少？ 10: 10分刷新BINLOG以后的数据。mysql\_-bin.000015

4、把mysql-bin.000015解析为sql，恢复到从库。

以上方案还是不会主键冲突问题。

1、停止一个从库，然后再主库刷新binlog，把mysql-bin.000014恢复成bin.sql(去掉d rop语句）

2、把全备bak\_2014-ll-06.sql及10点前的增量bin.sql恢复到从库。

3、数据丢多少？ 10: 10分刷新BINL0G以后的数据。mysql-bin.000015

4、停止主库，快速把mysql-bin.000015解析为sql,恢复到从库。

切换到从库提供服务

9.3企业备份流程图

9.4■分库分表备份脚本

#!/bin/bash #var define #

USERNAME="root" PASSWORD:”123456”

PORT=3306

SOCK="/data/${PORT}/mysql.sock" BACKUP\_PATH=Vserver/backup” function mysql\_cmd〇 {

CMD=$1

CMD=${CMD:-"quit;"} /application/mysql/bin/mysql -S $CMD"

}

DB\_LIST=$Cmysql\_cmd "show databases;"Igrep -vE "Database I schema Imysql

for db in $DB\_LIST do

for tb in $(mysql\_cmd "use $db;show tables;" I grep -vE ’’Tables”） do

mysqldump -S SSOCK -uSUSERNAME -pSPASSWORD --events -x $db $tbl gzip > $BACKUP\_PATH/$(date +%F)\_$db\_$tb.sql.gz done done

本说明：

这里使用mysqldump -x库名表名直接备份表。这里讲复杂的MySQL语句写成函数， 通过传参数实现mysql命令执行。

**10 MySQL** 引擎

**10.1什么是数据库引擎**

数据库引擎是用于存储、处理和保护数据的核心服务。利用数据库引擎可控制访问权限并快速处理事务,从而满足企业内大多数需要处理大量数据的应用程序的要求。使用数据库引擎创建用于联机事务处理或联机分析处理数据的关系数据库。这包括创建 用于存储数据的表和用于查看、管理和保护数据安全的数据库对象（如索引、视图和存储过程）。

**10.2** MySQL**引擎的类型**

MySQL 的引擎主要有 MylSAM、InnoDB、、Memory、NDB，下面是官方5.5版本的引擎对比：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature | MylSAM | Memory | InnoDB | Archive | NDB |
| **Storage limits** |  |  |  |  |  |
| **Transactions** |  |  |  |  |  |
| **Locking granularity** |  |  |  |  |  |
| **MVCC** |  |  |  |  |  |
| **Geospatial data type support** |  |  |  |  |  |
| **Geospatial indexing support** |  |  |  |  |  |
| **B-tree indexes** |  |  |  |  |  |
| **T-tree indexes** |  |  |  |  |  |
| **Hash indexes** |  |  |  |  |  |
| **Full-text search indexes** |  |  |  |  |  |
| **Clustered indexes** |  |  |  |  |  |
| **Data caches** |  |  |  |  |  |
| **Index caches** |  |  |  |  |  |
| **Compressed data** |  |  |  |  |  |
| **Encrypted dat** |  |  |  |  |  |
| **Cluster database support** |  |  |  |  |  |
| **Replication suppor** |  |  |  |  |  |
| **Foreign key support** |  |  |  |  |  |
| **Backup / point-in-time recover** |  |  |  |  |  |
| **Query cache support** |  |  |  |  |  |
| **Update statistics for data dictiona**  **ry** |  |  |  |  |  |
|  | | | | | |

**10.3 MyISAM 引擎**

**10.3.1 MyISAM引擎介绍**

MyISAM引擎在磁盘上的表存储会生成三个文件：.frm表定义文件,.MYI索引文件,.MYD索引文件。

**10.3.2 MylSAM引擎的特点:**

⯎不支持事务

⯎表级锁定

数据的更新会锁定整个表,这虽然可以让锁定的实现成本很小,但是也大大降低了并发的性能

⯎读写互相阻塞

写入和阻塞读取；读取会阻塞写入,但是读时不会阻塞其他读,所以一般做读写分离。

⯎只缓存MyISAM索引

通过设置key\_buffer\_size可以设置MyISAM缓存索引的大小,大大提高访问性能,减少磁盘IO,但是MyISAM不能缓存数据。

⯎读取速度快,占用资源相对较少

⯎不支持外键约束,但是支持全文索引

⯎MyISAM引擎是MySQL5.5.5前的默认引擎

10.3.3MylSAM引擎适用的生产场景

⯎不需要事务支持的业务（例如转账就不行）

⯎—般读数据比较多的应用,读写都频繁的场景不适合,读多或者写多的场景都适合。

⯎读写并发访问相对较低的业务（锁定机制导致）

⯎数据修改相对较少的业务。

⯎以读为主的业务,例如：数据库系统表、www、blog、图片信息数据库、用户信息库、商品库等业务。

⯎对数据一致性要求不是很高的业务（不支持事务）

⯎硬件资源比较差的机器（MyISAM占用资源少）

⯎使用读写分离的MySQL从库可以使用MyISAM引擎

**10.3.4 MyISAM引擎调优精要**

⯎设置合适的索引（利用索引缓存）

⯎调整读写优先级,根据实际需求确保重要操作优先执行

⯎启用延迟插入改善大批量写入性能（降低写入频率,尽量多条操作合并一次性写入）

⯎尽量顺序操作,让insert数据都写入到尾部,减少阻塞。

⯎拆分大的执行时间长的SQL操作,降低单个操作的阻塞时间。

⯎降低并发数（减少对MySQL的访问）,某些高并发的场景,可以使用排队队列机制

⯎对于相对静态的数据库文件,充分利用Query Cache或者Memcached缓存服务极大地提高访问效率,网站动态内容静态化,减少对数据库的访问。

**10.4 Innodb 引擎**

**10.4.1 Innodb引擎介绍**

事务型数据库的首选引擎,支持ACID事务,支持行级锁定。InnoDB是为处理巨大数据量时的最大性能而设计。InnoDB存储引擎完全与MySQL服务器整合,InnoDB存储引擎为在主内存中缓存数据和索引而维持它自己的缓冲池。InnoDB存储它的表&索引在一个表空间中，表空间可以包含数个文件（或原始磁盘分区）。这与MylSAM表不同,比如在MyISAM表中每个表被存在分离的文件中。

**10.4.2 Innodb引擎特点**

⯎支持事务：支持4个事务隔离级别,支持多个版本读。

⯎行级锁定（更新数据时一般是锁定当前行）：通过索引实现,全表扫描仍然会锁定整个表

⯎读写阻塞与事务隔离级别相关

⯎具有非常高效的缓存特性：能缓存索引,也能缓存数据

⯎整个表和主键以Cluster方式存储,组成一颗平衡树

⯎所有Secondary Index都会保存主键信息

⯎支持分区、表空间

⯎支持外键约束,5.5版本之前不支持全文索引,以后支持了。

⯎和MylSAM引擎相比,Innodb引擎对硬件资源要求比较高

**10.4.3 Innodb引擎调优精要**

⯎主键尽量小，避免给Secondary Index带来过大的空间负担

⯎建立有效的索引避免全表扫描，因为全表扫描也会锁表

⯎尽可能缓存所有的索引以及数据,提高响应时间,减少磁盘10消耗

⯎在批量小数据插入的时候,尽量自己控制事务,而不要使用autocommit自 动提交,有参数可以控制是否开启自动提交

⯎设置Innodb\_flush\_log\_at\_trx\_commit参数值，不要过度追求安全性

如果Innodb\_flush\_log\_at\_trx\_commit的值为0, log buffer每秒会被刷写日志文件到磁盘，提交事务的时候不做任何操作

⯎避免主键更新,因为这会带来大量的数据移动。

**10.5修改MySQL数据引擎**

**10.5.1通过shell脚本for循环批量修改表引擎**

首先通过show databases以及grep过滤出需要修改的数据库

然后use db;show tables;以及grep过滤出需要修改的表

使用两层for循环使用alter table命令批量修改表引擎

**10.5.2 通过mysql-convert-table-format命令修改表****引擎**

1.首先安装Perl依赖库

yum -y install perl-DBI perl-DBD-MySQL perl-Time-HiRes

2•使用mysql\_convert\_table\_format修改引擎

mysql\_convert\_table\_format --user=root --password=oldboyl23 --socket =/data/3306/mysql.sock --engine=MyISAM oldboy t2

**10.6 MySQL 事务**

**10.6.1事务的简介**

事务（tansaction),和现实世界的交易很类似,是指逻辑上的一组SQL操作,要么全部成功,要么全部失败。

**10.6.2事务的ACID原则**

1、A (Atomicity)原子性

原子性很容易理解,也就是说事务里的所有操作要么全部做完,要么都不做,事务成功的条件是事务里的所有操作都成功,只要有一个操作失败,整个事务就失败,需要回滚。

比如银行转账,从A账户转100元至B账户,分为两个步骤：1)从A账户取10 0元；2)存入100元至B账户。这两步要么一起完成,要么一起不完成,如果只完成第一步,第二步失败,钱会莫名其妙少了100元。

2、C (Consistency) —致性

一致性也比较容易理解,也就是说数据库要一直处于一致的状态,事务的运行不会改变数据库原本的一致性约束。

例如现有完整性约束a+b=10,如果一个事务改变了 a,那么必须得改变b,使得事务结束后依然满足a+b=10,否则事务失败。

3、I (Isolation)独立性

所谓的独立性是指并发的事务之间不会互相影响,如果一个事务要访问的数据正在被另外一个事务修改,只要另外一个事务未提交,它所访问的数据就不受未提交事务的影响。

比如现有有个交易是从A账户转100元至B账户,在这个交易还未完成的情况下,如果此时B查询自己的账户,是看不到新增加的100元的。

4、D (Durability)持久性

持久性是指一旦事务提交后,它所做的修改将会永久的保存在数据库上,即使出现宕机也不会丢失。

**10.7关闭自动提交**

MySQL默认是自动提交的，也就是提交一个QUERY，它就直接执行，我们可以通过以下操作设置为手动提交：

方法1:通过MySQL变量设置

Set global autocommit=0FF 关闭自动提交

方法2:通过MySQL配置文件my.cnf配置

autocommit=0FF

然后可以通过commit和rollback命令提交或者回滚。