数据库

**数据库**

**---------------------------------------------------------------------------------------------------------------------**

DB , DataBase

– 数据库:依照某种数据模型进行组织并存放到存储器的数据集合

• DBMS , DataBase Management System

– 数据库管理系统:用来操纵和管理数据库的大型服务软件

• DBS , DataBase System

– 数据库系统:即DB+DBMS ,指带有数据库并整合了数据库管理软件的计算机系统

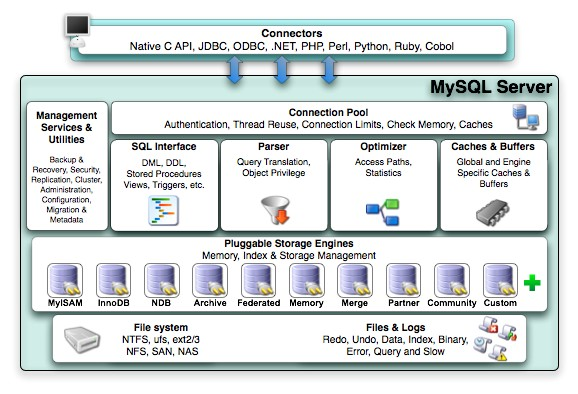
常见的数据库服务软件

|  |  |
| --- | --- |
| 类型 | 厂商 |
| Oracle Database | Oracle (甲骨文) |
| MySQL/MariaDB |
| SQL Server(不能跨平台) | Microsoft (微软) |
| Accessd |
| DB2 | IBM |
| Sybase | Sybase |
| PostgreSQL | 加州大学伯克利分校 |

**MYSQL**

**---------------------------------------------------------------------------------------------------------------------**

1. MYSQL工作原理



Mysql是由SQL接口，解析器，优化器，缓存，存储引擎组成的。

connectors：与其他编程语言中的sql 语句进行交互，如php、java等。

Management Serveices & Utilities：系统管理和控制工具。

Connection Pool (连接池)：管理缓冲用户连接，线程处理等需要缓存的需求。

SQL Interface (SQL接口)：接受用户的SQL命令，并且返回用户需要查询的结果。

Parser （解析器）：SQL命令传递到解析器的时候会被解析器验证和解析。

Optimizer (查询优化器)：SQL语句在查询之前会使用查询优化器对查询进行优化。

Cache和Buffer (查询缓存)：如果查询缓存有命中的查询结果，查询语句就可以直接去 查询缓存中取数据。

Engine (存储引擎)：存储引擎是MySql中具体的与文件打交道的子系统。处理表的处 理器

查看可用引擎类型：show engines；

| Engine | Support | Comment | Transactions | XA | Savepoints |

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

| InnoDB | YES | Supports transactions, row-level locking, and foreign keys | YES | YES | YES |

| MRG\_MYISAM | YES | Collection of identical MyISAM tables | NO | NO | NO |

| MEMORY | YES | Hash based, stored in memory, useful for temporary tables | NO | NO | NO |

| BLACKHOLE | YES | /dev/null storage engine (anything you write to it disappears) | NO |NO | NO |

| MyISAM | DEFAULT | MyISAM storage engine | NO | NO | NO |

| CSV | YES | CSV storage engine | NO | NO | NO |

| ARCHIVE | YES | Archive storage engine | NO | NO | NO |

| PERFORMANCE\_SCHEMA | YES | Performance Schema | NO | NO | NO |

| FEDERATED | NO | Federated MySQL storage engine | NULL | NULL | NULL |

设置默认存储引擎

修改配置文件/etc/my.cnf

default-storage-engine=MyISAM

#systemctl restart mysqld

MyISAM存储引擎

主要特点

– 支持表级锁

– 不支持事务、事务回滚、外键

优势是访问速度快

相关的表文件

表名 .frm 、表名 .MYI、表名 .MYD

InnoDB存储引擎

主要特点

– 支持行级锁定

– 支持事务、事务回滚、支持外键

相关表文件

表名.frm 、 表名.ibd、 ibdata1、ib\_logfile0、ib\_logfile1

MYSQL锁机制

锁粒度：表级锁、行级锁、页级锁

锁类型：读锁(共享锁):支持并发读。

写锁(互斥锁、排它锁):是独占锁,上锁期间其他线程不能读表或写表。

查看当前的锁状态

show status like ‘Table\_lock%’;

事物特性（ACID）

Atomic :原子性

– 事务的整个操作是一个整体,不可分割,要么全部成功,要么全部失败。

Consistency :一致性

– 事务操作的前后,表中的记录没有变化。

Isolation :隔离性

– 事务操作是相互隔离不受影响的。

Durability :持久性

– 数据一旦提交,不可改变,永久改变表数据

mysql> show variables like "autocommit"; // 查看提交状态

mysql> set autocommit=off; // 关闭自动提交

mysql> rollback ; // 数据回滚

mysql> commit; // 提交数据

1. 软件安装

http://dev.mysql.com/downloads/mysql/ //选择适用于当前系统的bundle集合包

mysql-community-client // 客户端应用程序

mysql-community-common // 数据库和客户端库共享文件

mysql-community-devel // 客户端应用程序的库和头文件

mysql-community-embedded // 嵌入式函数库

mysql-community-embedded-compat // 嵌入式兼容函数库

mysql-community-embedded-devel // 头文件和库文件作为 Mysql 的嵌入式库文件

mysql-community-libs //MySQL 数据库客户端应用程序的共享库

mysql-community-libs-compat // 客户端应用程序的共享兼容库

//mysql-community-mini最小化安装软件包冲突删除。

# yum -y install perl-JSON //依赖包安装

# rpm -Uvh mysql-community-\*.rpm //采用 U 升级安装,可替换冲突文件

1. 首次登陆

• 默认的数据库管理账号

– root ,允许从localhost 访问

– 首次登录密码在安装时随机生成

– 存储在错误日志文件里

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

2017-04-18T07:29:27.634784Z 1

[Note] A temporary password isgenerated for root@localhost: mtoa>Av<p6Yk

# mysql -uroot -p'mtoa>Av<p6Yk’ //部分密码有特殊字符，用‘’屏蔽

1. 修改root用户密码

mysql> set global validate\_password\_policy=0; // 只验证长度

validate\_password\_policy 验证密码策略

0 or LOW 长度

1 or MEDIUM( 默认 ) 长度;数字,小写/大写,和特殊字符

2 or STRONG 长度;数字,小写/大写和特殊字符;字典文件

mysql> set global validate\_password\_length=6; // 修改密码长度

mysql> alter user root@”localhost” identified by "123456"; // 修改登陆密码

1. 服务相关参数

/etc/my.cnf 主配置文件

/var/lib/mysql 数据库目录

/var/log/mysqld.log log文件

默认端口号 3306

进程名 mysqld

传输协议 TCP

进程所有者 mysql

进程所属组 mysql

1. 连接数据库

– mysql [ -h 服务器 -u 用户名 -p 密码 数据库 ]

1. 数据库的命名规则

– 可以使用数字 / 字母 / 下划线,但不能纯数字

– 区分大小写,具有唯一性

– 不可使用指令关键字、特殊字符

1. 数据库管理

库管理命令

– Show databases; // 显示已有的库

– Use 库名 ; // 切换库

– Select database(); // 显示当前所在的库

– Create database 库名 ; // 创建新库

– Show tables; // 显示已有的表

– Drop database 库名; // 删除库

新建指定名称的表

– CREATE TABLE 库名 . 表名 (

字段名 1 字段类型 ( 宽度 ) 约束条件 ,

字段名 2 字段类型 ( 宽度 ) 约束条件 ,

.. ..

字段名 N 字段类型 ( 宽度 ) 约束条件

);

表管理命令

– Desc 表名 ; // 查看表结构

– Select \* from 表名; // 查看表记录

– Drop table 表名 ; // 删除表

记录管理命令

– Select \* from 表名 ; // 查看表记录

– Insert into 表名 values( 值列表 );// 插入表记录

– Update 表名 set 字段 = 值 ; // 修改表记录

– Delete from 表名 ; // 删除表记录

修改表结构

– ALTER TABLE 表名 执行动作 ;

ADD 字段名 类型 ( 宽度 ) 约束条件 ;（可加 after 字段名 ;或者 first;）

Modify 字段名 类型 ( 宽度 ) 约束条件 ;（可加 after 字段名 ;或者 first;）

change 源字段名 新字段名 类型 ( 宽度 ) 约束条件;

Drop 字段名；

Rename 新表名；

查询匹配条件

数值比较：=、>=、>、<=、=、!=

字符比较：=、!=、is null、is not null

逻辑比较：OR、AND、!、()提高优先级

范围内匹配/去重：in（值列表）、not in（值列表）、

between 数字1 and 数字2、DISTINCT 字段名去重显示

模糊匹配：WHERE 字段名 LIKE ' 通配字串 '；

//通配符： \_ 匹配单个字符、% 匹配 0~N 个字符

正则匹配：WHERE 字段名 REGEXP ' 正则表达式 ‘

四则运算：+、-、\*、/、%

聚集函数：avg（字段名） 求平均值

sum（字段名） 求和

min（字段名） 统计最小值

max（字段名） 统计最大值

count（字段名） 统计数量

操作查询结果

查询结果排序： SQL 查询ORDER BY 字段名 [ asc | desc ]

查询结果分组： SQL 查询 group by 字段名

查询结果过滤： SQL 查询 HAVING 条件表达式;

SQL 查询 where 条件 HAVING 条件表达式;

SQL 查询 group by 字段名HAVING 条件表达式;

限制查询结果显示行数：

SQL 查询 LIMIT N; 显示查询结果前 N 条记录

SQL 查询 LIMIT N,M ;显示指定范围内的查询记录

SQL 查询 where 条件查询 LIMIT N ;显示查询结果前 N 条记录

SQL 查询 where 条件查询 LIMIT N , M ;显示指定范围内的查询记录

1. 常见类型

• 数值型:

数值类型 大 小 范围(有符号) 范围(无符号) 用 途

TINYINT 1 字节 -128~127 0 ~ 255 微小整数

SMALLINT 2 字节 -32768~32767 0 ~ 65535 小整数

MEDIUMINT 3 字节 -2^23 ~ 2^23-1 0 ~ 2^24 -1 中整数

INT 4 字节 -2^31 ~ 2^31-1 0 ~ 2^32-1 大整数

BIGINT 8 字节 -2^63 ~ 2^63-1 0 ~ 2^64-1 极大整数

FLOAT 4 字节 单精度浮点数

DOUBLE 8 字节 双精度浮点数

DECIMAL 对DECIMAL(M,D) ,其中 M 为有效位数、D 为小数位数, M 应大于D ,占用 M+2 字节

//整数型：UNSIGNED 修饰时,对应的字段只保存正数。关键字ZEROFILL时,填0代替空格补位。数值超过范围时报错。数值不够指定宽度时,在左边填空格补位。宽度仅是显示宽度,存数值的大小由类型决定

//浮点型：float(总宽度,小数位数)

• 字符型: 姓名、工作单位、通信住址

//字符型：定长char(字符数)，最大长度255字符

• 日期时间型: 出生日期、注册时间

时间函数：DATETIME、TIMESTAMP、DATE、YEAR、TIME

now() 获取系统当前日期和时间

year() 执行时动态获得系统日期时间

sleep(N) 休眠N秒

curdate() 获取当前的系统日期

curtime() 获取当前的系统时刻

month() 获取指定时间中的月份

date() 获取指定时间中的日期

time() 获取指定时间中的时刻

例：mysql > select time(20180506125040);

• 枚举型: 兴趣爱好、性别

从给定值集合中选择单个值, ENUM。格式: enum( 值 1, 值 2, 值 N)

从给定值集合中选择一个或多个值, SET。格式: set( 值 1, 值 2, 值 N)

1. 约束条件

Null 允许为空,默认设置

NOT NULL 不允许为空

Key 索引类型

Default 设置默认值,缺省为 NULL

1. 索引

– 通过创建唯一性索引,可以保证数据库表中每一行数据的唯一性

– 可以加快数据的检索速度

键值类型

• INDEX :普通索引

建表的时候指定索引字段

– INDEX( 字段 1), INDEX( 字段 2) .. ..

在已有的表中设置 INDEX 字段

– CREATE INDEX 索引名 ON 表名 ( 字段名 );

删除指定表的索引字段

– DROP INDEX 索引名 ON 表名 ;

查看表的索引信息

– SHOW INDEX FROM 表名 ;

• UNIQUE :唯一索引

• FULLTEXT :全文索引

• PRIMARY KEY :主键

建表的时候指定主键字段

– PRIMARY KEY( 字段名 )

在已有的表中设置 PRIMARY KEY 字段

– ALTER TABLE 表名 ADD PRIMARY KEY( 字段名 );

移除表中的 PRIMARY KEY 字段

– ALTER TABLE 表名 DROP PRIMARY KEY;

• FOREIGN KEY :外键

– 让当前表字段的值在另一个表中字段值的范围内选择。

使用条件

– 表的存储引擎必须是innodb

– 字段类型要一致

– 被参照字段必须要是索引类型的一种 (primary key)

基本用法

– FOREIGN KEY( 表 A 的字段名 )

References 表 B( 字段名 )

ON UPDATE CASCADE

ON DELETE CASCADE //在被参照表发生改变时参照表跟随改变

删除外键字段

– ALTER TABLE 表名 DROP FOREIGN KEY 约束名 ;

1. 数据导入导出

设置搜索目录

查看默认使用目录show variables like ‘secure\_file\_priv’；

（/var/lib/mysql-files/）

修改搜索目录

#vim /etc/my.cnf

secure\_file\_priv="目录路径"

#systemctl restart mysqld

数据导入

LOAD DATA INFILE “ 目录名 / 文件名”

INTO TABLE 表名

FIELDS TERMINATED BY “ 分隔符”

LINES TERMINATED BY “\n”;

数据导出

SELECT 查询 .. ..

INTO OUTFILE “ 目录名 / 文件名”

FIELDS TERMINATED BY “ 分隔符”

LINES TERMINATED BY “\n”;

1. 多表查询

多表查询：select 字段名列表 from 表 a, 表 b ; //笛卡尔集

select 字段名列表 from 表 a, 表 b where 条件;

//select t6.\*,t5.shell from t5,t6 where t5.uid = t6.uid;

where子查询：

select 字段名列表 from 表 A where 条件 ( select 字段名列表 from 表 A ) ;

select 字段名列表 from 表 A where 条件 ( select 字段名列表 from 表 A where 条件 ) ;

左连接查询：

select 字段名列表 from 表a LEFT JOIN表b ON条件表达式; //以左表为主显示

右连接查询：

select 字段名列表 from 表a RIGHT JOIN表b ON条件表达式; //以右表为主显示

复制表

• 将源表 xxx 复制为新表 yyy

CREATE TABLE yyy SELECT \* FROM xxx;

• 将指定的查询结果复制为新表 zzz

CREATE TABLE zzz SQL 查询语句 ;

• 复制源表 xxx 的结构到新表 vvv

CREATE TABLE vvv SELECT \* FROM xxx WHERE FALSE;

• 将源表 vvv 的名称改为 www

ALTER TABLE vvv RENAME TO www;

1. 用户授权及撤销

恢复MYSQL管理密码

#systemctl stop mysqld

# vim /etc/my.cnf

skip-grant-tables

#validate\_password\_policy=0 //注释密码验证策略，否则会出现错误

#validate\_password\_length=6

# systemctl restart mysqld

# mysql

mysql> update mysql.user set

authentication\_string=password("888888")

-> where

-> user="root" and host="localhost";

mysql> flush privileges;

mysql> quit

创建授权

GRANT 权限列表 .. .. ON 库名 . 表名 TO 用户名 @' 客户端地址 ' IDENTIFIED BY ' 密码 ' [ WITH GRANT OPTION ];

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

撤销授权

REVOKE 权限列表 ON 库名 . 表名 FROM 用户名 @' 客户端地址 ';

删除授权用户

drop user 用户名@‘客户端地址’；

权限列表

– all :匹配所有权限

– SELECT,UPDATE,INSERT .. ..

– SELECT,UPDATE ( 字段 1, .. .. , 字段 N)

–权限关键字

usage 无权限

SELECT 查询表记录

INSERT 插入表记录

UPDATE 更新表记录

DELETE 删除表记录

CREATE 创建库、表

DROP 删除库、表

RELOAD 有重新载入授权 必须拥有reload权限，才可以执行flush [tables | logs | privileges]

SHUTDOWN 允许关闭mysql服务 使用mysqladmin shutdown 来关闭mysql

PROCESS 允许查看用户登录数据库服务器的进程 （ show processlist; ）

FILE 导入、导出数据

REFERENCES 创建外键

INDEX 创建索引

ALTER 修改表结构

SHOW DATABASES 查看库

SUPER 关闭属于任何用户的线程

CREATE TEMPORARY TABLES 允许在create table 语句中使用 TEMPORARY关键字

LOCK TABLES 允许使用 LOCK TABLES 语句

EXECUTE 执行存在的Functions,Procedures

REPLICATION SLAVE 从主服务器读取二进制日志

REPLICATION CLIENT 允许在主/从数据库服务器上使用 show status命令

CREATE VIEW 创建视图

SHOW VIEW 查看视图

CREATE ROUTINE 创建存储过程

ALTER ROUTINE 修改存储过程

CREATE USER 创建用户

EVENT 有操作事件的权限

TRIGGER, 有操作触发器的权限

CREATE TABLESPACE 有创建表空间的权限

客户端地址

– % :匹配所有主机

– 192.168.1.% :匹配指定的一个网段

– 192.168.1.1 :匹配指定 IP 地址的单个主机

– %.tarena.com :匹配一个 DNS 区域

– svr1.tarena.com :匹配指定域名的单个主机

[ WITH GRANT OPTION ]

是否有授权权限，默认root在本机登陆具有授权权限

查看授权权限

SHOW GRANTS; //查看自己权限

SHOW GRANTS FOR 用户名 @' 客户端地址 '; //管理员查看其他用户权限

SELECT @@hostname; //查看登陆数据库主机名

SELECT user(); //查看登陆用户

重设用户密码

#mysqladmin -h主机地址 -u用户名 -p旧密码 password 新密码

SET PASSWORD=PASSWORD(' 新密码 '); //设置自己密码

SET PASSWORD FOR 用户名 @' 客户端地址 '=PASSWORD(' 新密码 '); //管理员设置其他用户密码

授权库mysql主要的几个表

– user 表,存储授权用户的访问权限

– 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

– columns\_priv 表,存储授权用户对字段的访问权限

1. 数据的备份与恢复

完全备份/恢复

物理备份：直接copy数据库/var/lib/mysql文件进行备份恢复

缺点：跨平台性差

备份时间长、冗余备份、浪费存储空间

逻辑备份：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。

mysqldump备份缺点：

效率较低，备份和还原速度慢

备份过程中，数据插入和更新操作会被挂起

# mysqldump -uroot -p123456 库名 > /路径/文件名

//-A 备份所有库

//-B 库名 多个库备份到一份备份文件

# mysql -uroot -p123456库名 < /路径/文件名

mysql> use 库名；

mysql> source /路径/文件名；

增量备份/恢复

备份完全备份后所有新产生的数据

启用mysql服务的binlog日志，实现增量备份

记录除查询之外的SEQ命名

mysql> show variables like “binlog\_format”;

//三种格式：statement（每条修改数据的sql命令都记录）

row（不记录sql语句上下文相关信息，只保存哪条记录被修改）

mixed（以上两种格式的混合）

#vim /etc/my.cnf

server\_id=50 //指定id值

log-bin //启用binlog日志

binlog\_format =”mixed”

#systemctl restart mysqld

binlog相关文件 （位置/var/lib/mysql）

主机名-bin.index 记录已有日志文件名

主机名-bin.000001 第一个二进制日志

恢复数据

通过mysqlbinlog提取历史SQL操作，管道给mysql命名执行

#mysqlbinlog 日志文件名 | mysql -u用户名 -p密码

删除指定版本之前的binlog 日志

#purge master logs to "binlog 文件名 ";

删除所有 binlog 日志,重建新日志

#reset master;

查看当前使用binlog日志

#show master status

查看日志

#mysqlbinlog [选项] binlog 日志文件名

选项

--start-datetime==“yyyy-mm-dd hh:mm:ss”

--stop-datetime==“yyyy-mm-dd hh:mm:ss”

--start-position==数字

--stop-position==数字

在线热备份

Xtrabackup工具

备份过程中不锁库表，适合生产环境

主要包含两个组件：

xtrabackup-->C程序，支持InnoDB/XtraDB

innobackupex-->以Perl脚本封装xtrabackup。（Innodb支持增量备份，MyISAM

只支持增量备份）

安装软件

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

# yum -y install libev-4.15-1.el6.rf.x86\_64.rpm

# rpm -ivh percona-xtrabackup-24-2.4.7-1.el7.x86\_64.rpm

innobackupex基本选项

--host 主机名

--user 用户名

--port 段口号

--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（会滚日志）

--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

# innobackupex --user root --password 123456 /backup --no-timestamp // 完全备份

#innobackupex --user root --password 123456 --databases =" 库名列表 " /fullbak --no-timestamp // 完全备份

#rm -rf /var/lib/mysql //恢复数据必须要求空的库目录

# innobackupex --user root --password 123456 --copy-back /backup // 恢复数据

#chown -R mysql:mysql /var/lib/mysql 更改权限

#systemctl restart mysqld 重启mysql服务

#innobackupex --user root --password 123456 --databases =" 库名列表” --incremental /new1 --incremental-basedir="/fullbak" --no-timestamp // 第 1 次增量备份

#innobackupex --user root --password 123456 --databases =" 库名列表” --incremental /new2 --incremental-basedir="/new1" --no-timestamp // 第 2 次增量备份

事物日志文件 /var/lib/mysql/ib\_logfile0 ib\_logfile1 ibdata1

增量LSN日志序列号 xtrabackup\_checkpoints

重新做并合并日志 --apply-log --redo-only

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

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

--increment-dir=new1db

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

--increment-dri=new2db

停止服务

#systemctl stop mysqld

清空数据库目录

#rm -rf /var/lib/mysql/\*

恢复数据

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

启动服务

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

#systemctl start mysqld

恢复某个表数据

完全备份

# innobackupex --user root --password 123456 --databases="buydb" /buydbfull2

--no-timestamp

从备份里导出表信息

# innobackupex --user root --password 123456 --databases="buydb" --apply-log

--export /buydbfull2

创建删除的表（表结构一样）

mysql> create table cc(name int);

删除表空间

mysql> alter table cc discard tablespace;

//#ls /var/lib/mysql/buydb

cc.cfg cc.exp cc.frm cc.ibd //确认表空间删除成功

拷贝表信息文件

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

修改文件归属

chown -R mysql:mysql /var/lib/mysql/buydb/cc.\*

导入表空间

差异备份/恢复（课程未讲解）

备份上次备份后所有新产生的数据

优点：提供多个恢复点，每个恢复点归档大小不会持续增长

缺点：要求完成备份和整个增量备份链才可以恢复到一个合适的点

1. 主从同步

主库Master和从库Slave，Master提供客户端访问，Slave同步Mater的数据到Slave实现数据的自动备份.

原理：

1> Master记录数据更改操作（设置sever\_id、启用binlog日志、设置binlog日志格式）

2> Slave运行两个线程

IO thread线程

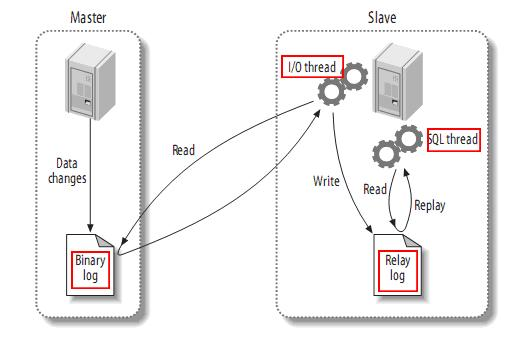
复制 master 主机 binlog 日志文件里的 SQL 到本机的 relay-log 文件里

Mater库mysql>show processlist; //显示当前工作服务列表（Binlog Dump）

SQL thread线程

执行本机 relay-log 文件里的 SQL 语句,重现 Master 的数据操作。

中继日志relay-log的命令不会记录到binlog文件里。



主库配置步骤

1. 启用binlog日志

#vim /etc/my.cnf

server\_id=50 //指定id值

log-bin //启用binlog日志

binlog\_format =”mixed”

#systemctl restart mysqld

1. 用户授权

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

1. 查看正在使用的binlog日志信息

#show master status

从库配置步骤

1. 验证授权用户

# mysql -h192.168.4.51 -urepluser -p123456

1. 指定server\_id

#vim /etc/my.cnf

server\_id=52

1. 指定主库信息

#mysql -uroot -p123456

mysql>change master to

master\_host=”192.168.4.51”,

master\_user=”repluser”,

master\_password=”123456”,

master\_log\_file=”host51-bin.000001”,

master\_log\_pos=441;

1. 查看从库状态信息

mysql>show slave status;

确保 Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

若Slave\_IO\_Running:No

解决方案：1.可以删除Slave端配置文件重新配置

2.master端，flush logs; show master status确认file及position信息;

//从库可以不启动binlog日志

------------------------------------------------------------------------------------------------------------------------------------------

数据库备份需要确保Master和Slave数据相同

Master数据备份应该需要同步的所有库

#mysqldump -u root -p123456 test > mytest.sql

Slave数据恢复，确保和Maste相同

#mysql -uroot -p123456 < mytest.sql

------------------------------------------------------------------------------------------------------------------------------------------

Slave配置文件

文件名 说明

master.info 连接主服务器信息

relay-log.info 中继日志信息

主机名 -relay-bin.xxxxxx 中继日志

主机名 -relay-bin.index 中继日志索引文件

常用配置选项

Master 服务器

选 项 用 途

binlog\_do\_db=name 设置 Master 对哪些库记日志

binlog\_ignore\_db=name 设置 Master 对哪些库不记日志

Slave 服务器

选 项 用 途

log\_slave\_updates 记录从库更新,允许链式复制( A-B-C )

relay\_log=dbsvr2-relay-bin 指定中继日志文件名

replicate\_do\_db=mysql 仅复制指定库,其他库将被忽略,此选项可设置多条(省略时复制

所有库)

replicate\_ignore\_db=test 不复制哪些库,其他库将被忽略, ignore-db与do-db只需选用其

中一种

MYSQL主从同步模式

主从同步结构模式

主+从、主+从+...+从、主从从（给从库做备份）、主主结构

主从同步复制模式

异步复制( Asynchronous replication )

– 主库在执行完客户端提交的事务后会立即将结果返给客户端,并不关心从库是否已经 接收并处理。

全同步复制( Fully synchronous replication )

– 当主库执行完一个事务,所有的从库都执行了该事务才返回给客户端。

半同步复制( Semisynchronous replication )

– 介于异步复制和全同步复制之间,主库在执行完客户端提交的事务后不是立刻返回给 客户端,而是等待至少一个从库接收到并写到 relay log 中才返回给客户端。

模式配置

mysql>show variables like "have\_dynamic\_loading"; //允许动态加载模块

命令行加载插件----用户需有 SUPER 权限

主库: mysql> INSTALL PLUGIN rpl \_ semi \_ sync \_ master SONAME 'semisync\_master.so';

从库: mysql> INSTALL PLUGIN rpl \_ semi \_ sync \_ slave SONAME 'semisync\_slave.so';

查看: mysql> SELECT PLUGIN\_NAME, PLUGIN\_STATUS FROM INFORMATION\_SCHEMA.PLUGINS WHERE PLUGIN\_NAME LIKE '%semi%';

启用半同步复制----在安装完插件后,半同步复制默认是关闭的

主: mysql> SET GLOBAL rpl\_semi\_sync\_master\_enabled = 1;

从: mysql> SET GLOBAL rpl\_semi\_sync\_slave\_enabled = 1;

查看: mysql> show variables like "rpl\_semi\_sync\_ %\_enabled";

永久配置启用半同步复制----主配置文件 /etc/my.cnf

主: plugin-load=rpl\_semi\_sync\_master=semisync\_master.so

rpl\_semi\_sync\_master\_enabled=1

从:plugin-load=rpl\_semi\_sync\_slave=semisync\_slave.so

rpl\_semi\_sync\_slave\_enabled=1

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. MYSQL性能调优

提高 MySQL 系统的性能、响应速度

– 替换有问题的硬件( CPU/ 磁盘 / 内存等)

– 服务程序的运行参数调整

– 对 SQL 查询进行优化

并发及连接控制

选 项 含 义

max\_connections 允许的最大并发连接数

connect\_timeout 等待建立连接的超时秒数,默认 10 秒,只在登录时有效

wait\_timeout 等待关闭连接的不活动超时秒数,默认 28800 秒( 8 小时)

mysql> show global status like "max\_used\_connections"; //显示历史使用最大数

mysql> show variables like "max\_connections"; //默认允许连接最大数

缓存参数控制

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

mysql> show variables like "key\_buffer\_size"; //内存存储索引

mysql> show global status like "key\_read%";

mysql> show variables like "sort\_buffer\_size";

mysql> show variables like "read%size";

mysql> show variables like "thread%size"; //查看可重用线程数

mysql> show global status like "threads\_%"; //产看当前线程重用情况

mysql> show variables like "table%"; //查看可缓存多少个打开的表

mysql> show global status like "open%tables"; //查看已打开，打开过多少表

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

SQL查询优化

性能调优思路

手段 具体操作

升级硬件 CPU 、内存、硬盘

加大网络带宽 付费加大带宽

调整mysql服务运行参数 并发连接数、连接超时时间、重复使用的线程数 ........

调整与查询相关的参数 查询缓存、索引缓存 .......

启用慢查询日志 slow-query-log

网络架构不合理 调整网络架构

常用日志种类及选项

类型 用途 配置

错误日志 记录启动/运行/停止过程中的错误消息 log-error[=name]

查询日志 记录客户端连接和查询操作 general-log

general-log-file=

慢查询日志 记录耗时较长或不使用索引的查询操作 slow-query-log

slow-query-log-file=

long-query-time=

关于查询的缓存

mysql> show variables like "query\_cache%"; //查看缓存的大小

mysql> show global status like "qcache%"; //查看当前的查询缓存统计

查询日志：记录所有字形过的操作命令

配置：

#vim /etc/my.cnf

genral-log

general-log-file=

-----------------------------------------------------------------------------------------------------------------------

慢查询日志：记录耗时较长或不使用索引慢查询日志的查询操作

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

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

配置：

#vim /etc/my.cnf

slow-query-log

log-queries-not-using-indexes=1

查看慢查询日志

#mysqldumpslow /var/lib/mysql/mysql-slow.log

～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～

1. Mysql数据读写分离

利用代理服务器maxscale，客户端的访问区分数据的读、写请求给不同的数据库服务器处理。

1>搭建主从结构

主库：启用binlog日志、用户授权、查看日志信息

从库:验证授权、指定server id、指定主库信息、查看从库状态

2>搭建代理服务

装包(mysql-proxy mycat maxscale)

# rpm -ivh maxscale-2.1.2-1.rhel.7.x86\_64.rpm

修改配置

# vim /etc/maxscale.cnf

[maxscale]

threads=auto

[server1]

type=server

address=192.168.4.10

port=3306

protocol=MySQLBackend

[server2]

type=server

address=192.168.4.20

port=3306

protocol=MySQLBackend

[MySQL Monitor]

type=monitor

module=mysqlmon

servers=server1, server2

user=scalemon

passwd=123456

monitor\_interval=10000

[Read-Write Service]

type=service

router=readwritesplit

servers=server1, server2

user=maxscale

passwd=123456

max\_slave\_connections=100%

[MaxAdmin Service]

type=service

router=cli

[Read-Write Listener]

type=listener

service=Read-Write Service

protocol=MySQLClient

port=4006

[MaxAdmin Listener]

type=listener

service=MaxAdmin Service

protocol=maxscaled

socket=default

port=4009

数据库服务器添加授权用户

mysql> grant replication slave, replication client on \*.\* toscalemon@'%' identified by “123456”; // 创建监控用户

mysql> grant select on mysql.\* to maxscale@'%' identified by“123456”; // 创建路由用户

mysql> grant all on \*.\* to student@'%' identified by“123456”; // 创建访问数据用户

启动服务

# maxscale --config=/etc/maxscale.cnf

查看服务信息

代理机连接管理端口：#maxadmin -uadmin -pmariadb -p 端口

MaxScale > list servers

连接代理服务器：#mysql -h192.168.4.100 -P4406 -uuser -p123456

//默认连接访问从库，在写入命令的瞬间切换到主库，写入完成切回从库

1. 多实例

一台服务器运行多个数据库服务（节约运维成本，提高硬件利用率）

配置步骤

– 安装支持多实例服务的软件包

# tar -zxvf mysql-5.7.20-linux-glibc2.12-x86\_64.tar.gz

# mv mysql-5.7.20-linux-glibc2.12-x86\_64 /usr/local/mysql

# vim /etc/profile

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

# . /etc/profile

– 修改主配置文件(每个实例要有独立的数据库目录、监听端口号、实例名称、sock文件）

#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 ,X 表示实例名称

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

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

– 初始化授权库

# ./mysqld --user=mysql --basedir= 软件安装目录 --datadir= 数据库目录 –- initialize

– 启动服务

# mysqld\_multi start实例编号

# ./mysqld\_multi --user=root --password=密码 stop 实例编号 // 停止实例进程

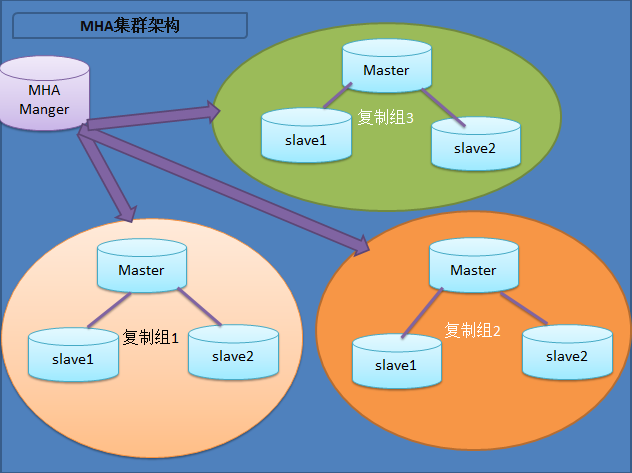
– 客户端访问

# ./mysql -uroot –p 初始密码 -S sock 文件

mysql> alter user user() identified by “ 新密码 ";

# ./mysql –uroot –p 新密码 -S sock 文件

1. MHA（Master High Availability）集群



主备模式，当Master主机宕机，Slave自动接替主角色的主机提供服务给客户端。然后将所有其他的 slave重新指向新的 master.

//将宕机的主机加入时需要重新配配置为Slave

MHA组成：

MHA Manager (管理节点)---->>MHA Node (数据节点)---->>SQL节点

– ( 1 )从宕机崩溃的 master 保存二进制日志事件( binlog events)

– ( 2 )识别含有最新更新的 slave

– ( 3 )应用差异的中继日志( relay log )到其他的 slave

– ( 4 )应用从 master 保存的二进制日志事件( binlog events )

– ( 5 )提升一个 slave 为新的 master ;

– ( 6 )使其他的 slave 连接新的 master 进行复制;

环境部署

-----------------------------------------------------------------------------------------------------------------

数据节点之间，管理主机到数据节点配置无密码登陆

# ssh-key-gen

# ssh-copy-id root@192.168.4.X

安装软件包

所有主机安装：

#yum -y install perl-\*.rpm //安装perl编程依赖包

perl-Config-Tiny-2.14-7.el7.noarch.rpm

perl-Mail-Sender-0.8.23-1.el7.noarch.rpm

perl-MIME-Types-1.38-2.el7.noarch.rpm

perl-Email-Date-Format-1.002-15.el7.noarch.rpm

perl-Mail-Sendmail-0.79-21.el7.art.noarch.rpm

perl-Parallel-ForkManager-1.18-2.el7.noarch.rpm

perl-Log-Dispatch-2.41-1.el7.1.noarch.rpm

perl-MIME-Lite-3.030-1.el7.noarch.rpm

# yum -y install perl-DBD-mysql //依赖包

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

管理主机上安装mha-manager包

# yum -y install perl-ExtUtils-\* perl-CPAN-\* //依赖包安装

#tar -zxf mha4mysql-manager-0.56.tar.gz

#cd mha4mysql-manager-0.56

# perl Makefile.pl

# make

# make install

master51数据库服务器配置文件(52~55做相应相似设置以51为主库)

#vim /etc/my.cnf

[mysqld]

plugin-load ="rpl\_semi\_sync\_master=semisync\_master.so;rpl\_semi\_sync\_slave=semisyn

c\_slave.so" //部分服务器开半同步复制

rpl-semi-sync-master-enabled = 1

rpl-semi-sync-slave-enabled = 1

server\_id=51

log-bin=master51

binlog-format="mixed"

# systemctl restart mysqld

mysql> set global relay\_log\_purge=off;// 不自动删除本机的中继日志文件

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

"123456";

52～55

mysql>change master to master\_host="192.168.4.53"master\_user="repluser",

master\_password="123456",master\_log\_file="db53-bin.000006",master\_log\_pos=154;

mysql>start slave;

mysql> set global relay\_log\_purge=off;// 不自动删除本机的中继日志文件

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

管理节点主机配置文件

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

#mkdir /etc/mha/

[root@host114 mha4mysql-manager-0.56]# cp

samples/conf/app1.cnf /etc/mha/

# vim /etc/mha/app1.cnf

[server default]

manager\_workdir=/etc/mha

manager\_log=/etc/mha/manager.log

master\_ip\_failover\_script=/etc/mha/master\_ip\_failover // 自动failover 时候的切换脚本,需要部署修改

ssh\_user=root

ssh\_port=22

repl\_user=repluser // 主从同步用户名

repl\_password=123456 // 主从同步密码

user=root // 连接数据库服务器用户名

password=123456 // 密码

[server1]

hostname=192.168.4.51

candidate\_master=1

[server2]

hostname=192.168.4.52

candidate\_master=1 // 设置为候选 master

[server3]

hostname=192.168.4.53

candidate\_master=1 // 设置为候选 master

[server4]

hostname=192.168.4.54

no\_master=1 // 不竞选 master

[server5]

hostname=192.168.4.55

no\_master=1 // 不竞选 master

测试 ssh 密钥对认证登陆

# masterha\_check\_ssh --conf= 管理节点主机主配置文件

# masterha\_check\_ssh --conf=/etc/mha/app1.cnf

测试主从同步状态

#masterha\_check\_repl --conf= 管理节点主机主配置文件

masterha\_check\_repl --conf=/etc/mha/app1.cnf

master主机设置vip地址

# ifconfig eth0:1 192.168.4.100/24

启动

chmod +x /etc/mha/master\_ip\_failover //添加执行权限

masterha\_manager --conf=/etc/mha/app1.cnf --remove\_dead\_master\_conf --ignore\_last\_failover（终端持续运行状态）

# masterha\_check\_status --conf=/etc/mha\_manager/app1.cnf //查看状态

-----------------------------------------------------------------------------------------------------------------------

1. 视图

虚拟表，内容与真实的表相似，包含一系列带有名称的列和行数据。

视图并不在数据库中以存储的数据的形式存在。

行和列的数据来自定义视图时查询所引用的基表,并且在具体引用视图时动态生成。

更新视图的数据,就是更新基表的数据，更新基表数据,视图的数据也会跟着改变。

优点

简单：使用视图的用户完全不需要关心视图中的数据是通过什么查询得到的。视图中的

数据对用户来说已经是过滤好的符合条件的结果集。

安全：用户只能看到视图中的数据。

数据独立：一旦视图的结构确定了,可以屏蔽表结构变化对用户的影响。

使用视图的限制

不能在视图上创建索引

在视图的 FROM 子句中不能使用子查询

部分情形中的视图不可更新

– 包含以下关键字的 SQL 语句:聚合函数 (SUM 、 MIN、 MAX 、 COUNT 等 ) 、 DISTINCT 、 GROUP BY、 HAVING 、 UNION 或 UNION ALL

– 常量视图

– JOIN

– FROM 一个不能更新的视图

– WHERE 子句的子查询引用了 FROM 子句中的表

– 使用了临时表,视图是不可更新

-----------------------------------------------------------------------------------------------------------------------基本使用

创建视图

create view 视图名称 as SQL查询；

create view 视图名称(字段名列表) as SQL查询;

//在视图表中不定义字段名的话,默认使用表中的字段名,若定义字段名的话,视图表中的字段名个数必须和基本中的字段个数相等。

查看视图

show table status; //查看全表和视图的状态

show table status where comment=”view”\G; //查看视图

show create view 视图名； //查看创建视图的具体命令

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；

创建视图完全格式

– CREATE

[OR REPLACE]

[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]

[DEFINER = { user | CURRENT\_USER }]

[SQL SECURITY { DEFINER | INVOKER }]

VIEW view\_name [(column\_list)]

AS select\_statement

[WITH [CASCADED | LOCAL] CHECK OPTION]

重要选项说明

OR REPLACE：创建时,若视图已存在,会替换已有的视图

ALGORITHM：定义处理视图的方式。

• MERAGE (替换方式)视图名直接使用视图的公式替换掉,把视图公式合并到了 select 中。

• TEMPTABLE (具体化方式)先得到视图的执行结果,该结果形成一个中间结果暂时存在内存中,之后,外

面的 select 语句就调用了这些中间结果。

• UNDEFINED (未定义)表示使用的是 MERAGE 替换方式。

WITH CHECK OPTION：当视图根据另一个视图定义时 , 对视图更新 / 删除 / 插入

– LOCAL 和 CASCADED 关键字决定了检查的范围。

– LOCAL 仅检查当前视图的限制。

– CASCADED 同时要满足基表的限制。

设置字段别名

视图中的字段名不可以重复 所以要定义别名

Create view 视图名 as select 表别名 . 源字段名 as 字段别名 from 源表名 表别名 left join 源表名 表别名 on 条件;

mysql> create view v2 as

select a.name as aname , b.name as bname , a.uid as auid , b.uid as buid

from user a left join info b on a.uid=b.uid;

MYSQL存储过程

数据库中保存的一系列sql命令的集合

编写存储过程时，可以使用变量，条件判断，流程控制等

存储过程，就是mysql中的脚本

优点

提高性能，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

delimiter //

create procedure()

begin

功能代码

.....

.....

end

//结束存储过程

delimiter ;

delimiter 关键字声明当前段分隔符，MySQL 默认以 ; 为分隔符,没有声明分割符,编译器会把存储过程当成 SQL 语句进行处理,则存储过程的编译过程会报错。

查看存储过程

mysql> show procedure status;

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

调用存储过程

mysql>Call存储过程名 ();

删除存储过程

mysql>drop procedure 存储过程名;

参数类型

Create procedure 名称 (

类型 参数名 数据类型 ,

类型 参数名 数据类型

)

关键字 名称 描述

in 输入参数 作用是给存储过程传值,必须在调用存储过程时赋值,在存储 过程中该参数的值不允许修改;默认类型是 in

out 输出参数 该值可在存储过程内部被改变,并可返回

inout 输入/输出参数 调用时指定,并且可被改变和返回

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 !!!

变量类型名称 描述

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;全局变量的修改会影响全局变量 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 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 session variables like “sort\_buffer\_size”; // 查看会话变量

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

mysql> set @y = 3; // 用户自定义变量,直接赋值

mysql> select max(uid) into @y from user; // 使用sql 命令查询结果赋值(user)

算数运算

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 SET @var6=10%3 ; 1

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

条件判断

数值比较

类 型 用 途

= 等于

> 、 >= 大于、大于或等于

< 、 <= 小于、小于或等于

!= 不等于

BETWEEN .. AND .. 在 .. 与 .. 之间

逻辑比较、范围、空、非空、模糊、正则

类型 用途

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

流程控制

顺序结构

if 条件测试 then

代码......

end if ;

if 条件测试 then

代码1......

else

代码2......

end if;

循环结构

while 条件判断 do

循环体

end while；

//无循环条件 （不按ctrl+c结束会一直执行）

loop

循环体

end loop；

//until条件判断，不成立时结束循环

repeat

循环体

until 条件判断

end repeat；

循环结构控制语句

LEAVE 标签名 //跳出循环

ITERATE 标签名 //放弃本次循环，执行下一次循环

分库分表

– 通过某种特定条件,将存放在一个数据库 ( 主机 ) 中的数据,分散存放到多个数据库 ( 主机 ) 中。

– 已达到分散单台设备负载的效果,即分库分表

– 数据的切分根据其切分规则的类型,分为 2 种切分模式

– 垂直分割 ( 纵向 ) 和 水平分割 ( 横向 )

纵向切分

– 把单一的表,拆分成多个表,并分散到不同的数据库( 主机 ) 上。

– 一个数据库由多个表构成,每个表对应不同的业务,可以按照业务对表进行分类,将其分布到不同的数据库 ( 主机 ) 上,实现专库专用,让不同的库 ( 主机 ) 分担不同的业务。

横向切分

– 按照表中某个字段的某种规则,把向表中写入的记录分散到多个库 ( 主机 ) 中。

– 简单来说,就是按照数据行切分,将表中的某些行存储到指定的数据库 ( 主机 ) 中。

Maycat软件

Mycat 是基于 Java 的分布式数据库系统中间层,为高并发下的分布式提供解决方案

– 支持 JDBC 形式连接

– 支持 MySQL 、 Oracle 、 Sqlserver 、 Mongodb 等

– 提供数据读写分离服务

– 可以实现数据库服务器的高可用

– 提供数据分片服务

– 基于阿里巴巴 Cobar 进行研发的开源软件

– 适合数据大量写入数据的存储需求

分片规则

• mycat 服务提供 10 种分片规则。

– 1 枚举法 sharding-by-intfile

– 2 固定分片 rule1

– 3 范围约定 auto-sharding-long

– 4 求模法 mod-long

– 5 日期列分区法 sharding-by-date

– 6 通配取模 sharding-by-pattern

– 7ASCII 码求模通配 sharding-by-prefixpattern

– 8 编程指定 sharding-by-substring

– 9 字符串拆分 hash 解析 sharding-by-stringhash

– 10 一致性 hash sharding-by-murmur

工作过程

• 当 Mycat 收到一个 SQL 时

– 会先解析这个 SQL 查找涉及到的表,然后看此表的定

– 如果有分片规则,则获取到 SQL 里分片字段的值,并

匹配分片函数,得到该 QL 对应的分片列表

– 然后将 SQL 发往这些分片去执行,最后收集和处理所

有分片返回的结果数据,并输出到客户端

配置Mycat //数据库服务器主机名、ip需要按照要求配置

1>安装JDK

# rpm -qa | grep -i jdk

java-1.8.0-openjdk-1.8.0.65-3.b17.el7.x86\_64

java-1.8.0-openjdk-headless-1.8.0.65-3.b17.el7.x86\_64

2>安装 mycat 服务软件包

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

#mv /mycat/ /usr/local/

3>修改配置文件

#vim /usr/local/mycat/conf/server.xml //设置连接mycat服务的帐号、密码等

<?xml version="1.0" encoding="UTF-8"?>

<!-- - - Licensed under the Apache License, Version 2.0 (the "License");

- you may not use this file except in compliance with the License. - You

may obtain a copy of the License at - - http://www.apache.org/licenses/LICENSE-2.0

- - Unless required by applicable law or agreed to in writing, software -

distributed under the License is distributed on an "AS IS" BASIS, - WITHOUT

WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. - See the

License for the specific language governing permissions and - limitations

under the License. -->

<!DOCTYPE mycat:server SYSTEM "server.dtd">

<mycat:server xmlns:mycat="http://org.opencloudb/">

<system>

<property name="defaultSqlParser">druidparser</property>

<!-- <property name="useCompression">1</property>-->

<!--1为开启mysql压缩协议-->

<!-- <property name="processorBufferChunk">40960</property> -->

<!--

<property name="processors">1</property>

<property name="processorExecutor">32</property>

-->

<!--默认是65535 64K 用于sql解析时最大文本长度 -->

<!--<property name="maxStringLiteralLength">65535</property>-->

<!--<property name="sequnceHandlerType">0</property>-->

<!--<property name="backSocketNoDelay">1</property>-->

<!--<property name="frontSocketNoDelay">1</property>-->

<!--<property name="processorExecutor">16</property>-->

<!--

<property name="mutiNodeLimitType">1</property>

0：开启小数量级（默认） ；1：开启亿级数据排序

<property name="mutiNodePatchSize">100</property>

亿级数量排序批量

<property name="processors">32</property> <property name="processorExecutor">32</property>

<property name="serverPort">8066</property> <property name="managerPort">9066</property>

<property name="idleTimeout">300000</property> <property name="bindIp">0.0.0.0</property>

<property name="frontWriteQueueSize">4096</property> <property name="processors">32</property> -->

</system>

<user name="test"> #连接mycat服务时使用的用户名 test

<property name="password">test</property>

#使用test用户连接mycat用户时使用的密码

<property name="schemas">TESTDB</property>

#连接上mycat服务后，可以看到的库名多个时，使用逗号分隔 （是逻辑上的库名）

</user>

<user name="user">

<property name="password">user</property>

<property name="schemas">TESTDB</property>

<property name="readOnly">true</property>

#定义只读权限，使用定义的user用户连接mycat服务后只有读记录的权限

</user>

<!-- <cluster> <node name="cobar1"> <property name="host">127.0.0.1</property>

<property name="weight">1</property> </node> </cluster> -->

<!-- <quarantine> <host name="1.2.3.4"> <property name="user">test</property>

</host> </quarantine> -->

</mycat:server>

#vim /usr/local/mycat/conf/schema.xml 配置 mycat 使用的真实数据库和表

<?xml version="1.0"?>

<!DOCTYPE mycat:schema SYSTEM "schema.dtd">

<mycat:schema xmlns:mycat="http://org.opencloudb/">

<schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100">

<!-- auto sharding by id (long) -->

<table name="travelrecord" dataNode="dn1,dn2" rule="auto-sharding-long" />

<!-- global table is auto cloned to all defined data nodes ,so can join

with any table whose sharding node is in the same data node -->

<table name="company" primaryKey="ID" type="global" dataNode="dn1,dn2" />

<table name="goods" primaryKey="ID" type="global" dataNode="dn1,dn2" />

<!-- random sharding using mod sharind rule -->

<table name="hotnews" primaryKey="ID" dataNode="dn1,dn2"

rule="mod-long" />

<!-- <table name="dual" primaryKey="ID" dataNode="dnx,dnoracle2" type="global"

needAddLimit="false"/> <table name="worker" primaryKey="ID" dataNode="jdbc\_dn1,jdbc\_dn2,jdbc\_dn3"

rule="mod-long" /> -->

<table name="employee" primaryKey="ID" dataNode="dn1,dn2"

rule="sharding-by-intfile" />

<table name="customer" primaryKey="ID" dataNode="dn1,dn2"

rule="sharding-by-intfile">

<childTable name="orders" primaryKey="ID" joinKey="customer\_id"

parentKey="id">

<childTable name="order\_items" joinKey="order\_id"

parentKey="id" />

</childTable>

<childTable name="customer\_addr" primaryKey="ID" joinKey="customer\_id"

parentKey="id" />

</table>

<!-- <table name="oc\_call" primaryKey="ID" dataNode="dn1$0-743" rule="latest-month-calldate"

/> -->

</schema>

<!-- <dataNode name="dn1$0-743" dataHost="localhost1" database="db$0-743"

/> -->

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

#定义分片使用的库，所在的物理主机 ,真正存储数据的db1库在物理主机c1上

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

#定义分片使用的库，所在的物理主机 ,真正存储数据的db2库在物理主机c2上

<!-- <dataNode name="jdbc\_dn1" dataHost="jdbchost" database="db1" /> <dataNode

name="jdbc\_dn2" dataHost="jdbchost" database="db2" /> <dataNode name="jdbc\_dn3"

dataHost="jdbchost" database="db3" /> -->

#指定c1名称主机对应的ip地址

<dataHost name="c1" 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="192.168.4.55:3306" user="admin"

password="123456">

#访问数据时 mycat服务连接数据库服务器时使用的用户名和密码

<!-- 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>

#指定c2名称主机对应的ip地址

<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="hostM2" url="192.168.4.54:3306" user="admin"

password="123456">

#访问数据时 mycat服务连接数据库服务器时使用的用户名和密码

<!-- 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>

<!-- <dataHost name="oracle1" maxCon="1000" minCon="1" balance="0" writeType="0"

dbType="oracle" dbDriver="jdbc"> <heartbeat>select 1 from dual</heartbeat>

<connectionInitSql>alter session set nls\_date\_format='yyyy-mm-dd hh24:mi:ss'</connectionInitSql>

<writeHost host="hostM1" url="jdbc:oracle:thin:@127.0.0.1:1521:nange" user="base"

password="123456" > </writeHost> </dataHost> <dataHost name="jdbchost" maxCon="1000"

minCon="1" balance="0" writeType="0" dbType="mongodb" dbDriver="jdbc"> <heartbeat>select

user()</heartbeat> <writeHost host="hostM" url="mongodb://192.168.0.99/test"

user="admin" password="123456" ></writeHost> </dataHost> <dataHost name="sparksql"

maxCon="1000" minCon="1" balance="0" dbType="spark" dbDriver="jdbc"> <heartbeat>

</heartbeat> <writeHost host="hostM1" url="jdbc:hive2://feng01:10000" user="jifeng"

password="jifeng"></writeHost> </dataHost> -->

<!-- <dataHost name="jdbchost" maxCon="1000" minCon="10" balance="0" dbType="mysql"

dbDriver="jdbc"> <heartbeat>select user()</heartbeat> <writeHost host="hostM1"

url="jdbc:mysql://localhost:3306" user="root" password="123456"> </writeHost>

</dataHost> -->

</mycat:schema>

4>数据库授权

mysql>grant all on \*.\* to admin@”%” identified by “123456”;

5>修改数据库配置文件，区分大小写

#Vim /etc/my.cnf

[mysqld]

lower\_case\_table\_names = 1 # 表名区分字母大小写

6>启动服务登陆

#sed -n ‘4，5p’ /ust/local/mycat/conf/wrapper.conf

Java Application

wrapper.java.command=java //jdk源码安装指定，RPM安装默认

#echo “export PATH=/usr/local/mycat/bin:$PATH” >> /etc /profile

#source /etc/profile

#mycat start

#mycat status //查看mycat服务状态

#netstat -ntulap | grep 8066

#mysql -hmycat服务器地址 -P8066 -utest -ptest

7>应用

mysql>show databases；

mysql>use TESTDB;

mysql>show tables；

mysql>select @@hostname;

**NOSQL**

-----------------------------------------------------------------------------------------------------------------------Not Only SQL,泛指非关系型数据库，不需要预先定义数据存储结构，表的每条记录都可以有不同的类型和结构

主流软件：

Redis、MongoDB、Memcached、CouchDB、Neo4j、FlockDB、

Redis（Remode Dictionary Server）远程字典服务器

一款高性能（Key/Values）分布式内存数据库，同时还支持list、hash、set、zset类型

支持数据持久化的NoSQL数据库服务软件，可以把内存里的数据保存到硬盘中

装包

#tar -xf redis-4.0.8.tar,gz

#cd redis-4.0.8

#make

#make install

初始化配置

#./utils/install\_server.sh

//配置运行参数，端口、主配置文件、数据库目录、pid文件、初始化完成程序默认启动

启动服务

#/etc/init.d/redis\_<portnumber> start

停止服务

#/etc/init.d/redis\_<portnumber> stop

连接Redis服务

#redis-cli //连接本机redis服务

#ps -C redis

#ss -ntulp | grep 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 \* //打印所有变量

keys a？ //打印指定变量

exits keyname //测试是否存在

ttl keyname //查看生存时间

type keyname //查看类型

move keyname dbname //以东变量

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

配置文件解析

数据单位： 1k => 1000bytes

1kb => 1024bytes

1m => 1000000bytes

1mb => 1024\*1024bytes

1g => 1000000000bytes

1gb => 1024\*1024\*1024bytes

常用配置选项

port 6379 //端口

bind 127.0.0.1 //IP地址

tcp-vacklog 511 //tcp 连接总数

timeout 0 //连接超时时间

tcp-keetalive 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

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爱你的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 //选取模版数据的个数（针对lru和ttl策略）

设置连接密码

grep -n requirepass /etc/redis/6379.conf

501:requirepass 123456

#redis-cli

127.0.0.1:6379> ping

(error) NOAUTH Authentication required.

127.0.0.1:6379> auth 123456 // 输入密码

OK

127.0.0.1:6379> ping

PONG

127.0.0.1:6379>

-----------------------------------------------------------------------------------------------------------------------部署LNMP+Redis

1>部署LNMP

#yum -y install gcc gcc-c++ pcre-devel

#tar -xf nginx-1.12.2.tar.gz

#cd nginx-1.12.2

./configure

#make

#make install

#yum -y install php-common

#rpm -ivh php-fpm-5.4.16-42.el7.x86\_64.rpm

修改nginx配置文件

#vim /usr/local/nginx/conf/nginx.conf

location ~\.php$ {

root html;

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php

include fastcgi.cnf;

}

启动服务

#systemctl restart php-fpm

#/usr/local/nginx/sbin/nginx

测试配置

#vim /usr/local/nginx/html/test.php

<?php

phpinfo();

?>

2>部署Redis

装包启服务

#tar -xf redis-4.0.8.tar.gz

#cd redis-4.0.8

#make

#make install

#./utils/install\_server.sh //初始化

#/etc/init.d/redis\_6379 start

//安装php扩展

#yum -y install autoconf

#yum -y install automake

#rpm -ivh php-devel-5.4.16-42.el7.x86\_64.rpm

#tar -xf php-redis-2.2.4.tar.gz

#cd phpredis-2.2.4/

#/usr/bin/phpize //输出php版本信息，保存在/usr/bin/php-config

Configuring for:

PHP Api Version:20100412

Zend Module Api No:20100525

Zend Extension Api No: 220100525

#./configure --with-php-config=/usr/bin/php-config

#make

#make install //生成redis.so文件，保存在/usr/lib64/php/modules

Installing shared extensions: /usr/lib64/php/modules/

#vim /etc/php.ini

extension\_dir = “/usr/lib64/php/modules/”

extension = “redis.so”

测试配置

查看是否支持模块

#php -m | grep -i redis

redis

编写测试文件

#vim /usr/local/nginx/html/redis.php

<?php

$redis = new redis();

$redis-> connect(‘127.0.0.1’，6379)；

$redis-> set(‘redistest’,’666666’);

echo $redis->get(‘redistest’);

?>

访问nginx服务

#curl http://127.0.0.1/redis.php

-----------------------------------------------------------------------------------------------------------------------创建集群

在redis主机上做如下配置，并运行redis服务

#yum -y install gcc

#tar -xf redis-4.0.8.tar.gz

#cd redis-4.0.8/

#make

#make install

#./utils/install\_server.sh

修改配置文件

#vim /etc/redis/6379.conf

bind IP 地址 // 只写物理接口 IP 地址

daemonize yes // 守护进程方式运行

port xxxx // 端口号不要使用默认的 6379

cluster-enabled yes // 开启集群

pidfile /var/run/xxx.pid #不允许相同

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

cluster-node-timeout 5000 // 请求超时 5 秒

创建集群主机

#yum -y install ruby rubygems

#rpm -ivh --nodeps ruby-devel-2.0.0.648-30.el7.x86\_64.rpm

# gem install redis-3.2.1.gem

# /opt/redis-4.0.8/src/redis-trib.rb create --replicas 1 IP:端口 ... IP:端口

//--replicas 1表示自动为每一个master节点分配一个slave节点

查看集群信息

#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

测试集群

在其中任意一台上存储数据 ,在其他任意一台上都可以获取数据。

存数据 set name jerry

取数据 get name

-----------------------------------------------------------------------------------------------------------------------管理集群

redis-cli命令

常用选项： -h ip地址

-p 端口

-c 访问集群

redis-trib.rb脚本

redis-trib.rb 选项 参数

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 --slave 添加从节点主机

del-node 删除节点主机

集群节点选举测试

停止某个主库的redis服务，对应的从库会自动升级主库

#redis-cli -h ip地址 -p端口 shutdown

添加新节点（默认新主机筛选为主库）

1>添加新主机做集群配置，并运行redis服务

2>将新主机添加到集群中

# /opt/redis-4.0.8/src/redis-trib.rb add-node 新主机IP:端口 192.168.4.11:6011

3>手动对集群进行重新分片

# ./redis-trib.rb reshard 192.168.4.11:6011

4>查看分配的槽位

./redis-trib.rb check 192.168.4.11:6011

5>添加从节点

# ./redis-trib.rb add-node --slave --master-id id值 从节点ip:端口 192.168.4.11:6011

//如果不指定主节点的 id 的话,会把新节点 随机添加为 从节点最少的主的从

移除节点

1>移除主节点

删除槽位

#redis-trib.rb reshard 192.168.4.11:6011

移除节点

#redis-trib.rb del-node 192.168.4.11:6011 被移除节点主机id

2>移除从节点

从节点主机没有槽位范围,直接执行移除命令即可

#redis-trib.rb del-node 192.168.4.51:6351 被移除主机的ID

-----------------------------------------------------------------------------------------------------------------------Redis主从复制

• 工作原理

– Slave 向 master 发送 sync 命令

– Master 启动后台存盘进程,同时收集所有修改数据命令

– Master 执行完后台存盘进程后,传送整个数据文件到slave 。

– 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

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

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

配置从库

#redis-cli -h 从库IP地址 -p端口

ip地址:端口 > SLAVEOF 主库IP地址 端口号 //命令行指定主库

ip地址:端口 > info replication //查看主从配置信息

反客为主（主库宕机后，手动将从库设置为主库）

#redis-cli -h 从库IP地址 -p端口

ip地址:端口 > SLAVEOF no one //设置为主库

哨兵模式（主库宕机后，从库自动升级为主库---Slave主机编辑sentinel.conf，运行哨兵程序）

# vim /etc/sentinel.conf

sentinel monitor 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

# redis-sentinel /etc/sentinel.conf

配置带验证的主从复制

配置 master 主机

– 设置连接密码 ,启动服务,连接服务

# sed -n '70p;501p' /etc/redis/6379.conf

bind 192.168.4.51

requirepass 123456 // 密码

# /etc/init.d/redis\_6379 start

# redis-cli -h 192.168.1.111 -a 123456 -p 6379

192.168.4.51:6379>

配置 slave 主机

指定主库 IP ,设置连接密码,启动服务

# sed -n '70p;282p;289p' /etc/redis/6379.conf

bind 192.168.4.52

slaveof 192.168.4.51 6379 // 主库 IP 与端口

masterauth 123456 // 主库密码

# /etc/init.d/redis\_6379 start

-----------------------------------------------------------------------------------------------------------------------

Redis 持久化 RDB/AOF

RDB（Reids DataBase）

– 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。

相关配置参数

•文件名

– dbfilename “dump.rdb” // 文件名

– save “” // 禁用 RDB

• 数据从内存保存到硬盘的频率

– save 900 1 // 900 秒内且有 1 次修改存盘

– save 300 10 //300 秒内且有 10 次修改存盘

– save 60 10000 //60 秒内且有 10000 修改存盘

• 手动立刻存盘

– > save // 阻塞写存盘

– > bgsave // 不阻塞写存盘

• 压缩

– rdbcompression yes | no

• 在存储快照后,使用 crc16 算法做数据校验

– rdbchecksum yes|no

• bgsave 出错停止写操作 , 对数据一致性要求不高设置为 no

– stop-writes-on-bgsave-error yes|no

使用RDB文件恢复数据

• 备份数据

# cp 数据库目录/dump.rdb 备份目录

• 恢复数据

#cp 备份目录/dump.rdb 数据库目录/

#/etc/redid/redis\_ 端口 start

RDB 优点 / 缺点

• 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（Append Only File）

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

相关配置参数

• 文件名

– appendfilename "appendonly.aof" // 文件名

– appendonly yes // 启用 aof ,默认 no

• AOF 文件记录,写操作的三种方式

– appendfsync always // 有新的写操作立即记录,性能差,完整性好。

– appendfsync everysec // 每秒记录一次,宕机时会丢失 1 秒的数据

– appendfsync no // 从不记录

• 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

• 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

使用 AOF 文件恢复数据

• 备份数据

cp 数据库目录/appendonly.aof 备份目录

• 恢复数据

Cp 备份目录/appendonly.aof 数据库目录 /

/etc/redid/redis\_ 端口 start

AOF 优点 / 缺点

• RDB 优点

– 可以灵活的设置同步持久化 appendfsync alwayls 或异步持久化 appendfsync verysec

– 宕机时,仅可能丢失 1 秒的数据

• RDB 的缺点

– AOF 文件的体积通常会大于 RDB 文件的体积。执行 fsync 策略时的速度可能会比 RDB 慢。

-----------------------------------------------------------------------------------------------------------------------数据类型

String字符串

• set key value [ex seconds] [px milliseconds] [nx|xx]

– 设置 key 及值,过期时间可以设置为秒或毫秒为单位

– nx 只有 key 不存在,才对 key 进行操作

– xx 只有 key 已存在,才对 key 进行操作

• setrange key offset value

– 从偏移量开始复写 key 的特定位的值

>set first "hello world"

>setrange first 6 “Redis”

• strlen key

– 统计字串长度

>strlen first

• append key value

– 字符存在则追加,不存在则创建 key 及 value

– 返回值为 key 的长度

>append myname jacob

• setbit key offset value

– 对 key 所存储字串,设置或清除特定偏移量上的位 (bit)

– Value 值可以为 1 或 0 , offset 为 0~2^32 之间

– key 不存在,则创建新 key

>setbit bit 0 1

>setbit bit 1 0

bit: 第 0 位为 1 ,第一位为 0

• bitcount key

– 统计字串中被设置为 1 的比特位数量

>setbit bits 0 1 //0001

>setbit bits 3 1 //1001

>bitcount bits //结果为2

记录网站用户上线频率,如用户 A 上线了多少天等类似的数据如用户在某天上线,则使用 setbit ,以用户名为 key ,将网站上线日为 offset ,并在该 offset 上设置 1 ,最后计算用户总上线次数时,使用 bitcount 用户名即可这样,即使网站运行 10 年,每个用户仅占用 10\*365 比特位即 456字节即可

>setbit peter 100 1 // 网站上线 100 天用户登录了一次

>setbit peter 105 1 // 网站上线 105 天用户登录了一次

>bitcount peter

• decr key

– 将 key 中的值减 1 , key 不存在则先初始化为 0 ,再减 1

>set test 10

>decr test

• decrby key decrement

– 将 key 中的值,减去 decrement

>set count 100

>decrby count 20

• get key

– 返回 key 所存储的字符串值

– 如果 key 不存在则返回特殊值 nil

– 如果 key 的值不是字串,则返回错误, get 只能处理字串

• getrange key start end

– 返回字串值中的子字串,截取范围为 start 和 end

– 负数偏移量表述从末尾计数, -1 表示最后一个字符, -2表示倒数第二个字符

>set first “hello,the world”

>getrange first -5 -1

>getrange first 0 4

• incr key

– 将 key 的值加 1 ,如果 key 不存在,则初始为 0 后再加 1

– 主要应用为计数器

>set page 20

>incr page

• incrby key increment

– 将 key 的值增加 increment

• incrbyfloat key increment

– 为 key 中所储存的值加上浮点数增量 increment

>set num 16.1

>incrbyfloat num 1.1

• mget key [key...]

– 一次获取一个或多个 key 的值,空格分隔, < 具有原子性>

• mset key value [key value ...]

– 一次设置多个 key 及值,空格分隔, < 具有原子性 >

Hash表

• Redis hash 是一个 string 类型的 field 和 value 的映射表

• 一个 key 可对应多个 field ,一个 field 对应一个 value

• 将一个对象存储为 hash 类型,较于每个字段都存储成string 类型更能节省内存

Hash表操作

• hset key field value

– 将 hash 表中 field 值设置为 value

>hset site google 'www.g.cn‘

>hset site baidu 'www.baidu.com'

• hget key filed

– 获取 hash 表中 field 的值

>hget site google

• hmset key field value [field value...]

– 同时给 hash 表中的多个 field 赋值

>hmset site google www.g.cn baidu www.baidu.com

• hmget key field [field...]

– 返回 hash 表中多个 field 的值

>hmget site google baidu

• hkeys key

– 返回 hash 表中所有 field 名称

>hmset site google www.g.cn baidu www.baidu.com

>hkeys site

• hgetall key

– 返回 hash 表中所有 field 的值

• hvals key

– 返回 hash 表中所有 filed 的值

>hvals key

• hdel key field [field...]

– 删除 hash 表中多个 field 的值,不存在则忽略

>hdel site google baidu

List 列表

Redis 的 list 是一个字符队列，先进后出，一个 key 可以有多个值

List 列表操作

• lpush key value [value...]

– 将一个或多个值 value 插入到列表 key 的表头

– Key 不存在,则创建 key

>lpush list a b c //list1 值依次为 c b a等同于 lpush list a; lpush list b; lpush list c

• lrange key start stop

– 从开始位置读取 key 的值到 stop 结束

>lrange list 0 2 // 从 0 位开始,读到 2 位为止

>lrange list 0 -1 // 从开始读到结束为止

>lrange list 0 -2 // 从开始读到倒数第 2 位值

• lpop key

– 移除并返回列表头元素数据, key 不存在则返回 nil

>lpop list // 删除表头元素,可以多次执行

• llen key

– 返回列表 key 的长度

• lindex key index

– 返回列表中第 index 个值

如 lindex key 0 ; lindex key 2; lindex key -2

• lset key index value

– 将 key 中 index 位置的值修改为 value

>lset list 3 test // 将 list 中第 3 个值修改为 test

• rpush key value [value...]

– 将 value 插入到 key 的末尾

>rpush list3 a b c //list3 值为 a b c

>rpush list3 d // 末尾插入 d

• rpop key

– 删除并返回 key 末尾的值

– >rpush list3 a b c //list3 值为 a b c

>rpush list3 d // 末尾插入 d

其他操作指令

• del key [key...] – 删除一个或多个 key

• exists key – 测试一个 key 是否存在

• expire key seconds – 设置 key 的生存周期

• persist key – 设置 key 永不过期

• ttl key – 查看 key 的生存周期

• keys 匹配 – 找符合匹配条件的 key ,特殊符号用 \ 屏蔽

>keys \* // 显示所有 key

>keys h?llo // 匹配 hello,hallo,hxllo 等

>keys h\*llo // 匹配 hllo 或 heeello 等

>keys h[ae]lo // 匹配 hello 和 hallo

• flushall – 清空所有数据

• select id – 选择数据库, id 用数字指定,默认数据库为 0

>select 0

>select 2

• move key db\_id – 将当前数据库的 key 移动到 db\_id 数据库中

>move key 1 //将key移动到1数据库中

• rename key newkey –给key改名为newkey , newkey已存在时,则覆盖其值

• renamenx key newkey – 仅当newkey不存在时,才将key改名为newkey

• sort key – 对 key 进行排序

>lpush cost 1 8 7 2 5

>sort cost // 默认对数字排序,升序

>sort cost desc // 降序

>lpush test“about”“site”“rename”

>sort test alpha // 对字符排序

>sort cost alpha limit 0 3 // 排序后提取 0-3 位数据

>sort cost alpha limit 0 3 desc

>sort cost STORE cost2 //对cost排序并保存为 cost2

• type key – 返回 key 的数据类型

-----------------------------------------------------------------------------------------------------------------------

MongoDB服务

介于关系数据库和非关系数据库之间的产品

– 一个基于分布式文件存储的数据库。

– 由 C++ 语言编写。旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

– MongoDB 将数据存储为一个文档,数据结构由键值(key=>value) 对组成。

– MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档,数组及文档数组。

软件特点

– 安装简单

– 面向文档存储,操作比较简单容易

– 支持丰富的查询表达

– 可以设置任何属性的索引

– 支持主流编程语言 RUBY|PYTHON|JAVA|PHP|C++

– 支持副本集,分片

MongoDB环境搭建

1>装包

免安装,解压后即可使用

# 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 -p etc log data/db

2>创建配置文件

手动创建服务主配置文件

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

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

logappend=true # 追加的方式记录日志信息

dbpath=/usr/local/mongodb/data/db # 数据库目录

fork=true # 守护进程方式运行

bind\_ip=192.168.4.50 //自主配置ip,默认127.0.0.1

port=27050 //自主配置端口

3>启动服务

#echo “export PATH=/usr/local/mongdb/bin/:$PATH” >> /etc/profile

#source /etc/profile

#mongd -f /usr/local/mongdb/etc/mongodb.conf

# ps -C mongod

# ss -ntulp | grep 27017

4>连接服务

#/usr/local/mongodb/bin/mongo

#/usr/local/mongodb/bin/mongo --host 主机地址 --port 端口

5>停止服务

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

MongoDB基本使用

常用管理命令

数据库管理

– show dbs 查看已有的库

– db 显示当前所在的库

– use 库名 切换库,若库不存在的话 自动延时创建库

– show collections 或 show tables 查看库下已有的集合

– db.dropDatabase() 删除当前所在的库

数据库名可以是满足以下条件的任意 UTF-8 字符串。

– 不能是空字符串( "") 。

– 不得含有 ' ' (空格 ) 、 . 、 $ 、 / 、 \ 和 \0 ( 空字符 )。

– 应全部小写。

– 最多 64 字节

集合管理

– show collections 或 show tables # 查看集合

– 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

• 合法的集合名

– 集合名不能是空字符串 "" 。

– 集合名不能含有 \0 字符(空字符 ) ,这个字符表示集合名的结尾。

– 集合名不能以 "system." 开头,这是为系统集合保留的前缀。

– 用户创建的集合名字不能含有保留字符。

文档基本管理

– db. 集合名 .find()

– db. 集合名 .count()

– db. 集合名 .insert({“name”:”jim”})

– db. 集合名 .find({ 条件 })

– 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 .remove({ 条件 }) # 删除与条件匹配的所有文档

-----------------------------------------------------------------------------------------------------------------------

基本数据类型

• 字符串 string

– UTF-8 字符串都可以表示为字符串类型的数据

– {name:” 张三” } 或 { school:“tarena”}

• 布尔 bool

– 布尔类型有两个值 true 和 false , {x:true}

• 空 null

– 用于表示空值或者不存在的字段, {x:null}

• 数值

– shell 默认使用 64 为浮点型数值。 {x : 3.14} 或 {x : 3} 。

– NumberInt ( 4 字节整数) {x:NumberInt(3)}

– NumberLong ( 8 字节整数) {x:NumberLong(3)}

• 数组 array

– 数据列表或数据集可以表示为数组

– {x : [“a“ ,“ b”,”c”]}

• 代码

– 查询和文档中可以包括任何 JavaScript 代码

– {x: function( ){/\* 代码 \*/}}

• 日期

– 日期被存储为自新纪元依赖经过的毫秒数,不存储时区

– {x:new Date( )}

• 对象

– 对象 id 是一个 12 字节的字符串,是文档的唯一标识

– {x: ObjectId() }

• 内嵌

– 文档可以嵌套其他文档,被嵌套的文档作为值来处理

– {tarena: {address:“Beijing”,tel:“888888”,perso

n:”hanshaoyun”

– }}

• 正则表达式

– 查询时,使用正则表达式作为限定条件

– {x:/ 正则表达式 /}

-----------------------------------------------------------------------------------------------------------------------数据导入导出

数据导出

• 语法格式 1

– #mongoexport [--host IP 地址 --port 端口 ] -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

• 语法格式 2

– #mongoexport --host IP地址 --port 端口 -d 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

注意:导出为 csv 格式必须使用 -f 指定字段名列表 !!!

• 语法格式 3

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

数据导入

• 语法格式 1

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

• 语法格式 2

– #mongoimport –host IP 地址 – port 端口 -d 库名 – c 集合名 --type=csv --headerline [--drop] 目录名/文件名 .csv

注意:导入数据时库和集合不存在时,会创建库和集合后导入数据反之以追加的方式导入数据到集合里,

使用--drop选项可以删除原有数据后导入新数据 --headerline 忽略标题

数据备份恢复

• 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

目录无需事先创建 备份时指定即可!!!

• 查看 bson 文件内容

#bsondump ./dump/bbs/t1.bson

数据恢复

• 语法格式

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

-----------------------------------------------------------------------------------------------------------------------MongoDB副本集

副本集概念

– MongoDB 复制是将数据同步在多个服务器的过程。

– 复制提供了数据的冗余备份,并在多个服务器上存储数据副本,提高了数据的可用性, 并可以保证数据的安全性。

– 复制还允许您从硬件故障和服务中断中恢复数据

复制原理

– mongodb 的复制至少需要两个节点。其中一个是主节点,负责处理客户端请求,其余的都是从节点,负责复制主节点上的数据。

– mongodb 各个节点常见的搭配方式为:一主一从、一主多从。

– 主节点记录在其上的所有操作 oplog ,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主节点一致。

副本集实现方式

• Master-Slave主从复制

– 实现数据同步只需要在某一台服务器启动时加上 "-master" 参数,以指明此服务器的角色是 primary ;另一台服务器加上 "-slave" 和 "-source" 参数,以指明此服务器的角色是slave。

• 主从复制的优点如下:

– 从服务器可以执行查询工作,降低主服务器访问压力。

– 在从服务器执行备份,避免备份期间锁定主服务器的数据。

– 当主服务器出现故障时,可以快速切换到从服务器,减少宕机时间。

• Replica Sets 复制集

– MongoDB 在 1.6 版本对开发了新功能 replica set,这比之前的 replication 功能要强大一 些,增加了故障自动切换和自动修复成员节点,各个 DB 之间数据完全一致,大大降低了维 护成功。使用 replica set故障切换完全自动。

– Replica Sets 的结构类似一个集群,完全可以把它当成一个集群,因为它确实与集群实现的作用是一样的:如果其中一个节点出现故障,其他节点马上会将业务接管过来而无须停机操作。

配置 Replica Sets 副本集

运行服务---指定主机所在副本集名称（副本集成员间使用西哪个通的副本集名称）

#./mongod --bind\_ip 192.168.4.61 --logpath=/var/log/mongod.log --replSet rs1

//replSet=rs1可以在配置文件配置,删除集群注释配置文件

配置节点信息

#./mongo --host 192.168.4.61

config = {

\_id:"rs1",

members:[

{\_id:0,host:“IP 地址 : 端口 "},

{\_id:1,host:“IP 地址 : 端口 "},

{\_id:2,host:“IP 地址 : 端口 "}

]};

初始化Replica Sets环境

>rs.initiate(config)

>rs.status() //查看状态信息

>rs.isMaster() //查看是否是master库

>db.getMongo().setSlaveOK() //同步数据验证，允许从库查看数据

>rs.isMaster() //自动切换主库验证，查看是否是主库

MongoDB 文档管理

插入文档

– db. 集合名 .save({ key:“ 值”, key:” 值” })

– db. 集合名 .insert({key:“ 值”, key:” 值” })

– db. 集合名 .insertMany(

[

{name:"xiaojiu“,age:19} ,

{name:"laoshi“,email:”yaya@tedu.cn”}

]) //插入多条服务

查询文档

– db. 集合名 .find()

– > db. 集合名 .findOne()

– > db. 集合名 .find ( { 条件 },{ 定义显示的字段 } )

行数显示限制

limit( 数字 ) // 显示前几行

skip( 数字 ) // 跳过前几行

sort( 字段名 ) // 排序

查询条件

简单条件

– db. 集合名 .find({key:” 值” })

– db. 集合名 .find({key:” 值”, keyname:” 值” })

范围比较

– $in 在...里

– $nin 不在...里

– $or 或

正则匹配

– > db.user.find({name: /^a/ })

数值比较

– $lt 、$lte、 $gt、 $gte、 $ne <、 <=、 >、 >=、 !=

匹配 null , 也可以匹配没有的字段

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

更新文件

– > db. 集合名 .update({ 条件 },{ 修改的字段 } )

• $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.user3.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.user3.update({name:"bin"},{$unset:{password:"A"}})

多文档更新

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

– > db.user.update({name:“bin”},{$set:{password:“abc12123”}} , false,true)

– Db. 集合名 .update({ 条件 },{$inc:{ 字段名 : 数字 }})

//$inc 条件匹配时,字段值自加或自减

– db. 集合名 .update({ 条件 },{$push:{ 数组名 :“ 值” }})

//$push 向数组中添加新元素

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

//$addToSet 避免重复添加

– db. 集合名 .update({ 条件 },{$pop:{ 数组名 : 数字 }})

//$pop 从数组头部删除一个元素

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

//$pull 删除数组指定元素

删除文档

• $drop 删除集合的同时删除索引

– db. 集合名 .drop( )

– db.user.drop( )

• 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({})