# MySQL整理

- update命令
- -U参数，防止误操作
- delete命令
- 删除表
- rename命令

MySQL基础操作
- 建库
- 建表
- 往表里插入数据

数据类型
- 数值类型
- 字符串类型
- 日期和时间类型

索引
- 建立主键索引的方法
- 建立普通索引的方法
- 删除索引
- 创建前缀索引
- 联合索引
- 唯一索引
- 建表语句加入联合索引：例如某sns产品生产正式建表语句
- 创建索引总结
- explain测试查询是否走索引
- 企业案例：利用explain优化SQL语句
- 使用profile功能查询优化

MySQL备份常用方法
- 逻辑备份介绍
- 物理备份介绍

MySQL数据库备份与恢复
- mysqldump工作原理
- 备份库
- 备份表
- 备份数据库表结构
- 刷新binlog
- mysqldump关键参数
- 生产场景不同引擎mysqldump备份
- 恢复数据库

MySQL生产环境备份
- 全量备份
- 增量备份
- 企业场景全量和增量的频率是怎么做的？
- MySQL的mysqldump备份什么时候派上用场？

# 数据库介绍

数据库：存放数据的仓库，按照一定的数据结构（数据结构是指数据的组织形式或数据之间的联系）来组织、存储的。
**当今数据库模型种类：关系型数据库和非关系型数据库。**

## 关系型数据库

是把复杂的数据结构归结为简单的二元关系（即二维表格形式）；在关系型数据库中，对数据的操作几乎全部建立在一个或多个关系表格上，通过对这些关联的表格分类、合并、连接或选取等运算来实现数据的管理。

**关系型数据库小结：**

1. 关系型数据库就是二维表格 。
2. 市场占有量较大的为MySQL和Oracle数据库，互联网运维最常用的是MySQL。
3. 通过SQL结构化查询语言来存取、管理数据。
4. 保持数据一致性方面很强，ACID理论。

> 特点：读写更多的是和磁盘打交道，数据一致性，安全性高
> 缺点：速度慢

**常见关系型数据库**

- **Oracle：**应用范围：大企业、政府、金融、证券；Oracle11g、Oracle12c。
- **MariaDB：**是数据库管理系统是MySQL数据库的一个分支，主要由开源社区维护。它的目的是完全兼容MySQL数据库，MariaDB基于事务的Maria存储引擎。
- **SQL Server：**微软开发的大型关系型数据库系统，具有使用方便可伸缩性好与相关软件集

成程度高等优点，缺点是只能在Windows系统下运行。

- **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。其体积小、速度快、总体拥有成本低，尤其是开放源码。

# 非关系型数据库

**NoSQL（Not Only SQL），非关系型数据库。**
针对特定场景，以高性能和使用便利为目的的功能特异化的数据库产品。
随着互联网web2.0网站的兴起，传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的SNS类型的web2.0纯动态网站已经显得力不从心，暴露了很多难以克服的问题，而非关系型的数据库则由于其本身的特点得到了非常迅速的发展。NoSQL数据库的产生就是为了解决大规模数据集合多重数据种类带来的挑战，尤其是大数据应用难题。

**NoSQL数据库在以下的这几种情况下比较适用：**

1. 数据模型比较简单；
2. 需要灵活性更强的IT系统；
3. 对数据库性能要求较高；
4. 不需要高度的数据一致性；
5. 对于给定key，比较容易映射复杂值的环境。

**NoSQL小结：**

1. NoSQL不是否定关系型数据库，而是作为关系型数据库的一个重要补充。
2. NoSQL为了高性能、高并发而生，以高效、高性能为目的，凡是和效率性能无关的因素都尽可能抛弃，忽略影响高性能、高并发的功能。
3. NoSQL典型产品memcached（纯内存），redis（持久化缓存），mongodb（文档型数据库）。

# 非关系型数据库种类

**1. 键值（key-Value）存储数据库**
这一类数据库主要会使用到一个哈希表，这个表中有一个特定的键和一个指针指向特定的数据。Key/value模型对于IT系统来说的优势在于简单、易部署。但是如果DBA只对部分值进行查询或更新的时候，Key/value就显得效率低下了。例：Tokyo Cabinet/Tyrant, Redis, Voldemort, Oracle BDB。

**2. 列存储（Columu-Oriented）数据库**
这部分数据库通常是用来应对分布式存储的海量数据。键仍然存在，但是它们的特点是指向了多个列。这些列是由列家族来安排的。如：Cassandra, HBase, Riak。

**3. 面向文档（Document-Oriented）的数据库**
文档型数据库的灵感是来自于Lotus Notes办公软件的，而且它同第一种键值存储相类似。该类型的数据模型是版本化的文档，半结构化的文档以特定的格式存储，比如JSON。文档型数据库可以看作是键值数据库的升级版，允许之间嵌套键值。而且文档型数据库比键值数据库的查询效率更高。如：CouchDB, MongoDb。国内也有文档型数据库SequoiaDB，已经开源。

**4. 图形（Graph）数据库**

图形结构的数据库同其他行列以及刚性结构的SQL数据库不同，它是使用灵活的图形模型，并且能够扩展到多个服务器上。NoSQL数据库没有标准的查询语言(SQL)，因此进行数据库查询需要制定数据模型。许多NoSQL数据库都有REST式的数据接口或者查询API。如：Neo4J, InfoGrid, Infinite Graph。

| 分类 | Example 举例 | 典型应用场景 | 数据模型 | 优点 | 缺点 |
| --- | --- | --- | --- | --- | --- |
| 键值(key-value) | Tokyo Cabinet/Tyrant **Redis**,Oracle BDB,Voldmort | 内容缓存，主要用于处理大量数据的高负载，也用于一些日志系统等。 | key 执行 value 的键值对，通常用 hash table 来实现。 | 查找速度快 | 数据无结构化，通常只被当做字符串或二进制数据。 |
| 列存储数据库 | Cassandra,Hbase,Riak | 分布式的文件系统 | 以列簇式存储，将同一列数据存在一起。 | 查看速度快，可扩展性强，更容易进行分布式扩展。 | 功能相对局限 |
| 文档性数据库 | CouchDB,**MongoDB** | WEB 应用(与 key-value 类似，value 是结构化的，不同的时数据库能够了解 value 的内容)。 | key-value 对应的键值对，value 为结构化数据。 | 数据结构要求不严格，表结构可变，不需要像关系型数据库一眼更需要预先定义表结构。 | 查询性能不高，而且缺乏统一的查询语法。 |
| 图形数据库 | Neo4J,InfoGrid,Infinite Graph | 社交网络，推荐系统等。专注于构建关系图谱。 | 图结构 | 利用图结构相关算法。如最短路径寻址、N 度关系查找等。 | 很多时候需要对整个图做计算才能的出需要的信息，且这种结构不太好做分布式的集群方案。 |

**常见非关系型数据库**

- **memcached（key-value）：**
  memcached是一个开源的、高性能的、具有分布式内存对象的缓存系统。它通过在内存中缓存数据和对象来减少读取数据库的次数，从而提高动态、数据库驱动网站的速度。

  > ①key-value型数据库
  > ②纯内存数据库，重启就丢数据
  > ③持久化产品memcachedb（sina开发）

- **Redis：**
  Redis是一个key-value存储系统。和Memcached类似，它支持存储的value类型相对更多，包括string(字符串)、list(链表)、set(集合)、zset(sorted set –有序集合)和hash（哈希类型）。这些数据类型都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，redis支持各种不同方式的排序。与memcached一样，为了保证效率，数据都是缓存在内存中。区别的是redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了master-slave(主从)同步。

> ①支持内存缓存，这个功能相当于memcached
> ②支持持久化存储，这个功能相当于memcachedb
> ③数据类型更丰富，比其他key-value数据库功能更多
> ④支持主从集群，分布式集群
> ⑤支持队列等特殊功能

- **MongoDB：**
  MongoDB介于关系型数据库和非关系型数据之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似json的bson格式，因此可以存储比较复杂的数据类型。Mongo最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

# 为什么选择MySQL数据库

毫无疑问，绝大多数的使用Linux操作系统的大中小互联网网站都在使用MySQL作为其后端的数据库存储，从大型的BAT门户，到电商平台，分类门户等无一例外都使用MySQL数据库。那么，MySQL数据库到底有哪些优势和特点：

1. MySQL性能卓越，服务稳定，很少出现异常宕机。
2. MySQL开放源代码且无版权制约，自主性及使用成本低。
3. MySQL历史悠久，社区及用户非常活跃，遇到bug、问题，易寻求帮助。
4. MySQL软件体积小，安装使用简单，并且易于维护；安装和维护成本低。
5. MySQL品牌口碑效应，使得企业无需考虑就直接使用之，LAMP、LNMP流行架构都可。
6. MySQL支持多种操作系统，提供多种API接口，支持多种开发语言，特别对流行的PHP语言有很好的支持。
   **注：** `API` 叫做应用程序接口，又称为应用编程接口，就是软件系统不同组成部分衔接的约定。

# 企业场景MySQL版本选择

1. 稳定版：选择开源的社区版的稳定版（GA版）。
2. 产品线：可以选择5.1或5.5；互联网公司主流5.5，其次是5.1和5.6 。
3. 选择MySQL数据库GA版发布后6个月以上的GA版 。
4. 要选择前后几个月没有大的BUG修复的版本，而不是大量修复BUG的集中版本 。
5. 最好向后较长时间没有更新发布的版本。
6. 要考虑开发人员开发程序使用的版本是否兼容你选的版本。
7. 作为内部开发测试数据库环境，跑大概3-6个月的时间 。
8. 优先企业非核心业务采用新版本的数据库GA版本软件。
9. 向DBA高手请教，或者在技术氛围好的群里和大家一起交流，使用真正的高手们用过的好用的GA版本产品 。
10. 经过上述工序之后，若是没有重要的功能BUG或性能瓶颈，则可以开始考虑作为任何业务数据服务的后端数据库软件。

# MySQL的安装

1. yum/rpm 简单、快，无法定制
2. 编译安装，`./configure;make;make install`
   复杂、慢，可定制
3. 使用二进制包
   解压就能用（就像绿色软件，无需安装）
   简单、快，不好定制

> **建议**
> 针对MySQL，第一条产品线4.0-5.1系列yum安装
> MySQL 5.5以上：编译安装(./cmake或gmake;gmake install)或二进制包

# 使用二进制包安装

一. 下载二进制包，本次使用mysql-5.6.28版本演示

```
1. [root@db01 ~]# cd /home/lzyi/tools/
2. [root@db01 tools]# wget http://mirrors.sohu.com/mysql/MySQL-5.6/mysql-5.6.28-linux-glibc2.5-i686.tar.gz
```

二. 解压二进制包

```
1. [root@db01 tools]# mkdir /application
2. [root@db01 tools]# tar xf mysql-5.6.28-linux-glibc2.5-i686.tar.gz -C /application/
3. [root@db01 tools]# mv /application/mysql-5.6.28-linux-glibc2.5-i686 /application/mysql-5.6.28
4. [root@db01 tools]# ln -s /application/mysql-5.6.28/ /application/mysql
```

三. 创建用户mysql

```
1. [root@db01 tools]# useradd mysql -M -s /sbin/nologin
```

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

```
[root@db01 tools]# /application/mysql/scripts/mysql_install_db --basedir=/application/mysql --datadir=/application/mysql/data/ --user=mysql
/application/mysql/bin/my_print_defaults: error while loading shared libraries: libstdc++.so.6: cannot open shared object file: No such file or directory
FATAL ERROR: Neither host 'db01' nor 'localhost' could be looked up with /application/mysql/bin/resolveip
Please configure the 'hostname' command to return a correct hostname.
If you want to solve this at a later stage, restart this script with the --force option

注：发现报错，需安装libstdc++.so.6和libaio.so.1库
[root@db01 tools]# yum install libstdc++.so.6 libaio.so.1 -y

再次运行初始化脚本，初始化数据库
[root@db01 tools]# /application/mysql/scripts/mysql_install_db --basedir=/application/mysql --datadir=/application/mysql/data --user=mysql
2016-01-01 01:11:42 2178 [Note] InnoDB: Shutdown completed; log sequence number 1626017
OK
.....................................................
2016-01-01 01:11:44 2201 [Note] InnoDB: Shutdown completed; log sequence number 1626027
OK
以上两个OK表示成功
```

五. 设置目录属主和属组权限

```
[root@db01 tools]# chown -R mysql.mysql /application/
[root@db01 tools]# ll /application/
total 4
lrwxrwxrwx.  1 mysql mysql   26 Jan  1 00:31 mysql -> /application/mysql-5.6.28/
drwxr-xr-x. 13 mysql mysql 4096 Jan  1 00:42 mysql-5.6.28
```

六. 修改配置文件

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copy到`/etc/`下

```
[root@db01 tools]# cp /application/mysql/support-files/my-default.cnf /etc/my.cnf
cp: overwrite `/etc/my.cnf'? yes
修改[mysqld]标签下，添加basedir和datadir
[root@db01 tools]# vim /etc/my.cnf
basedir = /application/mysql
datadir = /application/mysql/data
```

七. 修改启动脚本，使用mysql官方提供的启动脚本

```
[root@db01 tools]# cp /application/mysql/support-files/mysql.server /etc/init.d/mysqld
启动MySQL并检查进程
[root@db01 tools]# /etc/init.d/mysqld start
Starting MySQL.... SUCCESS!
[root@db01 tools]# ps -ef|grep mysql
root       2786      1  0 01:43 pts/0    00:00:00 /bin/sh /application/mysql/bin/mysqld_safe --datadir=/application/mysql/data --pid-file=/application/mysql/data/db01.pid
mysql      2923   2786  4 01:43 pts/0    00:00:00 /application/mysql/bin/mysqld --basedir=/application/mysql --datadir=/application/mysql/data --plugin-dir=/application/mysql/lib/plugin --user=mysql --log-error=/application/mysql/data/db01.err --pid-file=/application/mysql/data/db01.pid
root       2953   1725  1 01:44 pts/0    00:00:00 grep mysql
[root@db01 tools]# netstat -lntp|grep mysql
tcp    0  0 :::3306      :::*      LISTEN      2923/mysqld
```

八. 设置开机自启

第1种方法：使用`chkconfig`管理

```
$ chkconfig --add mysqld
$ chkconfig mysqld on
```

第2种方法：使用`/etc/rc.local`文件管理，是为了工作中统一管理，防止多处启动，也可归档，建议使用此法。

```
$ cat>>/etc/rc.local<<EOF
#####start mysql####
/etc/init.d/mysqld start
EOF
```

九. 添加至环境变量

第1种方法：添加至`/etc/profile`全局变量文件中，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。

```
$ echo "export PATH=/application/mysql/bin:$PATH">>/etc/profile
$ source /etc/profile 或 . /etc/profile
```

第2种方法：批量添加软连接至PATH变量路径下，这种方法比较新颖，但是可以达到效果而不用将300M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。

```
$ ln -s /application/mysql/bin/* /usr/local/sbin
$ which mysqld
/usr/local/sbin/mysqld
```

十. 配置man帮助，编辑`/etc/man.config`文件

```
$ vim /etc/man.config
......略......
48 MANPATH /application/mysql/man  #此为添加项
......略......
```

十一. 配置头文件，可通过创建软链接实现

```
$ ln -s /application/mysql/include /usr/include/mysql
```

十二. 配置库文件，将`lib`库路径添加到`ld.so.conf`文件中或者`ld.so.conf.d`目录下

```
$ echo "/application/mysql/lib" >/etc/ld.so.conf.d/mysql-x86_64.conf
$ ldconfig
```

# 源码安装MySQL

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-5.6.28.tar.gz**安装包作为演示， 版本安装方式都一样，小版本可以忽略。

一. 安装软件依赖包**ncurses-devel**和**libaio-devel**，前者提供字符终端处理库，后者提供ATO，即异步IO所需库，直接通过`yum`安装

```
$ yum install ncurses-devel libaio-devel -y
```

二. 安装**cmake**工具，MySQL 5.5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：
法1：使用epel和阿里源，直接yum安装

```
$ wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-6.repo  #epel源
或:
$ wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-6.repo #阿里CentOS源
$ yum install cmake -y
```

法2：源码安装

```
$ rz
$ tar xf cmake-3.2.2.tar.gz
$ cd cmake-3.2.2
$ ./configure
$ gmake && gmake install
```

### 三. 创建mysql用户

```
useradd mysql -M -s /sbin/nologin
```

### 四. 下载、解压软件包

```
[root@db03 ~]# cd /home/lzyi/tools/
[root@db03 tools]# wget http://mirrors.sohu.com/mysql/MySQL-5.6/mysql-5.6.28.tar.gz
[root@db03 tools]# tar xf MySQL-5.6/mysql-5.6.28.tar.gz
```

### 五. 编译MySQL

```
[root@db03 tools]# cd mysql-5.6.28
[root@db03 mysql-5.6.28]# cmake . -DCMAKE_INSTALL_PREFIX=/application/mysql-5.6.28 \
-DMYSQL_DATADIR=/application/mysql-5.6.28/data \
-DMYSQL_UNIX_ADDR=/application/mysql-5.6.28/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 \
-DWITH_FAST_MUTEXES=1 \
-DWITH_ZLIB=bundled \
-DENABLED_LOCAL_INFILE=1 \
-DWITH_EMBEDDED_SERVER=1 \
-DWITH_DEBUG=0

出现以下信息则编译成功：
-- Build files have been written to: /home/lzyi/tools/mysql-5.6.28

[root@db03 mysql-5.6.28]# make && make install
[root@db03 mysql-5.6.28]# ln -s /application/mysql-5.6.28/ /application/mysql
```

注意：重新运行编译时，需要删除CMakeCache.txt文件，清除旧的对象文件和缓存信息，如下：

```
[root@db03 mysql-5.6.28]# find / -type f -name "CMakeCache.txt"
/home/lzyi/tools/mysql-5.6.28/CMakeCache.txt
[root@db03 mysql-5.6.28]# find / -type f -name "CMakeCache.txt"|xargs rm -f
[root@db03 mysql-5.6.28]# make clean
```

编译的详细参数参考

六. 设置软件目录属主和属组权限

```
[root@db03 mysql-5.6.28]# chown -R mysql.mysql /application/
```

七. 使用初始化脚本初始化数据库

```
[root@db03 mysql-5.6.28]# /application/mysql/scripts/mysql_install_db --basedir=/application/mysql/ --datadir=/application/mysql/data/ --user=mysql

输出以下信息表示则成功：
2016-01-01 04:03:29 48310 [Note] InnoDB: Shutdown completed; log sequence number 1625977
OK
.....................略...........
2016-01-01 04:03:31 48332 [Note] InnoDB: Shutdown completed; log sequence number 1625987
OK
.....................略...........
```

八. 修改配置文件

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copy到 `/etc/` 下

```
[root@db03 mysql-5.6.28]# cp /application/mysql/support-files/my-default.cnf /etc/my.cnf
cp: overwrite `/etc/my.cnf'? yes
修改[mysqld]标签下，添加basedir和datadir
[root@db03 mysql-5.6.28]# vim /etc/my.cnf
basedir = /application/mysql
datadir = /application/mysql/data
```

九. 修改启动脚本 ，使用mysql官方提供的启动脚本

```
[root@db03 mysql-5.6.28]# cp /application/mysql/support-files/mysql.server /etc/init.d/mysqld
启动MySQL并检查进程
[root@db03 mysql-5.6.28]# /etc/init.d/mysqld start
Starting MySQL.... SUCCESS!
[root@db03 mysql-5.6.28]# netstat -lntp|grep mysql
tcp    0  0 :::3306      :::*       LISTEN      48514/mysqld
```

十. 设置开机自启
第1种方法：使用**chkconfig**管理

```
$ chkconfig --add mysqld
$ chkconfig mysqld on
```

第2种方法：使用**/etc/rc.local**文件管理，是为了工作中统一管理，防止多处启动，也可归档，建议使用此法。

```
$ cat>>/etc/rc.local<<EOF
#####start mysql####
/etc/init.d/mysqld start
EOF
```

十一. 添加至环境变量
第1种方法：添加至`/etc/profile`全局变量文件中，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。

```
$ echo "export PATH=/application/mysql/bin:$PATH">>/etc/profile
$ source /etc/profile 或 . /etc/profile
```

第2种方法：批量添加软连接至PATH变量路径下，这种方法比较新颖，但是可以达到效果而不用将300M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。

```
$ ln -s /application/mysql/bin/* /usr/local/sbin
$ which mysqld
/usr/local/sbin/mysqld
```

# yum安装MySQL

这种方式最简单，用于简单的测试、不重要的业务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
```

# 制作MySQL源码rpm包

一. 安装fpm打包工具，fpm工具使用ruby语言写的，要使用fpm，得安装ruby环境和gem命令（gem命令是从rubygem仓库安装软件类似yum从yum仓库安装软件）

```
$ yum -y install ruby rubygems ruby-devel # 安装ruby环境
$ gem source list  # 查看当前源
$ gem sources -a http://mirrors.aliyun.com/rubygems/  # 添加国内阿里源
$ gem sources --remove http://rubygems.org/ # 移除国外源
$ gem install fpm # 安装fpm工具
$ fpm --help  # 查看帮助
```

二. 准备rpm的post-install脚本进行安装之后的初始化工作

```
$ vim mysql_setup.sh
#!/bin/bash
#Date:         04:17  2016-01-01
#Author:       Created by lzyi
#Mail:         623913455@qq.com
#Function:     This scripts function is Initialization mysql
#Version:      1.0

ln -s /application/mysql-5.6.28/ /application.mysql

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

\cp /application/mysql/support-files/my-default.cnf /etc/my.cnf && \
sed -i 's#basedir.*#basedir = /application/mysql#g' /etc/my.cnf && \
sed -i 's#datadir.*#datadir = /application/mysql/data#g' /etc/my.cnf


useradd mysql -M -s /sbin/nologin && \
chown mysql.mysql -R /application/mysql*

echo "/application/mysql/lib" >/etc/ld.so.conf.d/mysql-x86_64.conf &&
ldconfig

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

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

/etc/init.d/mysqld start
```

三. 使用fpm对MySQL程序进行打包

```
$ fpm -f -s dir -t rpm -n mysql-install -v 1.0 --description 'mysql-se
rver installed by lzyi' -d 'ncurses-devel,libaio-devel'--post-install
/server/scripts/mysql_setup.sh /application/mysql-5.6.28
```

这里**-f**表示存在覆盖，**-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**指定依赖软件包，**–post-install**指定安装完之后要执行的脚本，最后接多个要打包的文件【不支持软连接】。

# MySQL多实例

> **什么是MySQL多实例**
> 简单说就是在一台机器上开启多个不同的服务端口（如：3306/3307），运行多个MySQL服务进程，这些服务进程通过不同的socket监听不同的服务端口来提供各自的服务。
> 这些MySQL多实例公用一套MySQL安装程序，使用不同（亦可相同）的my.cnf配置文件、启动程序、数据文件。在提供服务时，多实例MySQL在逻辑上看来是各自独立的，多个实例的自身是根据配置文件的对应设定值，来取得服务器的相关的硬件资源多少。

比喻：MySQL多实例相当于房子的多个卧室一样，每个实例可以看多一间卧室，整个服务器就是一套房子，服务器的硬件资源（CPU、mem、Disk）可以看做房子的卫生间、厨房、厅一样，是房子的公用资源，北漂的小伙伴蜗居在一起，休息在自己的卧室，出来活动肯定要共用上述公共资源。这样想就应该明白MySQL多实例。 补充：其实很多服务器可以配置多实例的，在门户网站用的更广泛，例如nginx就可以多实例，Apache、Haproxy、redis、memcahce，都可以多实例。

## MySQL多实例的作用与问题

- **①有效利用服务器资源**
  当单个服务器资源有剩余时，可以充分利用剩余的资源提供更多的服务，且可以实现资源的逻辑隔离。
- **②节约服务器资源**
  当公司资金紧张，但是数据库又需要各自尽量独立的提供服务，而且，需要主从复制等技术时，多实例就再好不过了。
- **③资源互相抢占问题**
  当某个数据库实例并发很高或者有SQL慢查询时，整个实例会消耗整个更多的内存、CPU、磁盘I/O资源，导致服务器上的其他的数据库实例提供服务的质量一起下降。这就相当于大家住在一个房子的不同卧室，早上起来上班，都要刷牙、洗脸等，卫生间就会长期占用，其他人就要等待一样的道理。

## MySQL多实例生产应用场合

- **1.资金紧张形公司的选择**
  当公司业务访问量不太大，又舍不得花钱，但又希望不同业务的数据库服务各自尽量独立的提供服务互相不受影响，而且需要主从复制等技术提供备份或读写分离服务时，多实例就再好不过了。如：可通过3台服务器部署6-9个实例，交叉做主从同步备份及读写分离，实现6-9台服务器才有的效果。这里要强调的是：所谓的尽量独立是相对的。
- **2.并发访问不是特别大的业务**
  当公司业务访问量不是太大的时候，服务器的资源基本都是浪费的，这时就很适合多实例的应用，如果对SQL语句优化做的比较好，MySQL多实例是一个很值得使用的技术，即使

并发很大，合理分配好系统资源以及搭配好服务，也不会有大问题

- **3.门户网站应用MySQL多实例场景**
  门户网站通常都会使用多实例，配置硬件好的服务器，可节省IDC机柜空间，跑多实例也会减少硬件资源不满的浪费，如：百度搜索引擎的数据库就是多实例，一般是从库，例如某部门：IBM 48核CPU，内存96G，跑3-4个实例；sina网也是用的多实例，内存48G左右。
  门户网站使用多实例的目的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

## 多实例相关面试题

> 问题1：你们的数据库是多实例，跑几个实例？CPU、内存、磁盘类型、RAID类型分别如何配置？
> 答：一般在1-4个实例之间居多，1-2个最多，因为大业务占用的机器比较多。机器是R510居多，CPU是E5210,48G内存，磁盘12*300G的SAS盘，做RAID10
>
> 问题2：是编译安装DB？还是二进制的多？还是什么方式多？
> 答：编译安装之后，做成RPM包，统一使用。

# MySQL多实例配置方案

第1种：多配置文件、多启动程序部署方案【老男孩推荐， 官方手册有】
通过配置多个配置文件及多个启动程序来实现多实例的方案。
**优点：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。**
配置方式：

```
[root@db03 data]# tree
.
├── 3306
│   ├── data        《=3306实例的数据文件
│   ├── my.cnf      《=3306实例的配置文件
│   └── mysql       《=3306实例的启动文件
└── 3307
    ├── data        《=3307实例的数据文件
    ├── my.cnf      《=3307实例的配置文件
    └── mysql       《=3307实例的启动文件

4 directories, 4 files
配置文件、启动程序、数据文件都是独立的文件。
```

第2种：单一配置文件，单一启动程序【官方推荐】

```
my.cnf配置文件样例（mysql手册提到的方法）
[mysqld_multi]
mysqld      = /usr/bin/mysqld-safe
mysqladmin  = /usr/bin/mysqladmin
user        = mysql

[mysqld1]
socket      = /var/lib/mysql/mysql.sock
port        = 3306
pid-file      = /var/lib/mysql/mysql.pid
datadir      = /var/lib/mysql/
user        = mysql

[mysqld2]
socket      = /mnt/data/db01/mysql.sock
port        = 3307
pid-file      = /mnt/data/db01/mysql.pid
datadir      = /mnt/data/db01/
user        = mysql
skip-name-resolve
server-id=10
default-storage-engine=innodb
innodb_buffer_pool_size=512M
innodb_additional_men_pool=10M
default_character_set=utf8
character_set_server=utf8
#read-only
relay-log-space-limit=3G
expire_logs_day=20
```

启动程序命令：`mysql_multi --config-file=/data/mysql/my_multi.cnf start 1,2`
**此方案缺点：耦合性太高，一个配置文件，不好管理。工作开发和运维的统一原则：降低耦合度。**

# MySQL多实例配置

MySQL多实例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 、环境变量、库文件、 帮助都已搞定。

- **多实例配置文件my.cnf**

```
[client]
port            = 3306
socket          = /data/3306/mysql.sock

[mysql]
no-auto-rehash

```

```
[mysqld]
user    = mysql      #MySQL管理用户
port    = 3306       #MySQL实例端口，不同实例端口不同
socket  = /data/3306/mysql.sock #指定socket文件位置
basedir = /application/mysql    #MySQL程序安装目录
datadir = /data/3306/data       #MySQL数据目录，不同实例目录不同
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       #最大错误连接数
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    #默认表引擎类型是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             #长请求时间为1S
#log_long_format
#log-error = /data/3306/error.log       #错误日志位置
#log-slow-queries = /data/3306/slow.log #慢查询日志位置
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 #relaylog信息文件
binlog_cache_size = 1M          #binlog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，只对MyISAM引擎有效
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
# 以下MyISAM相关选项未开启
#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      #忽略表名大小写
skip-name-resolve               #跳过名称解析
slave-skip-errors = 1032,1062   #主从复制，从库错误代码忽略列表
```

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

server-id = 1                      #主从复制server-id不能一样

innodb_additional_mem_pool_size = 4M  #innodb附加内存池大小
innodb_buffer_pool_size = 32M      #缓冲池大小
innodb_data_file_path = ibdata1:128M:autoextend #innodb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_flush_log_at_trx_commit = 2 #提交时刷新日志，0，1，2
innodb_log_buffer_size = 2M        #日志缓冲大小
innodb_log_file_size = 4M          #日志文件大小
innodb_log_files_in_group = 3      #一组日志文件数量
innodb_max_dirty_pages_pct = 90    #最大dirtypages（脏页）百分比
innodb_lock_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/mysqld.pid             #PID文件位置
```

- **多实例启动脚本**

```
#!/bin/sh
#init
port=3306                  #不同实例端口不同
mysql_user="root"          #mysql用户
mysql_pwd="123456"         #mysql用户密码，使用mysqladmin关闭，需正确密码
CmdPath="/application/mysql/bin"       #mysql命令路径
mysql_sock="/data/${port}/mysql.sock" #socket文件位置
#startup function
function_start_mysql()    #使用进程来检测状态更准确，防止强制停止导致socket遗留问题
{
    if [ `netstat -lntp|grep ${port}|wc -l` -ne 1 ];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 [ `netstat -lntp|grep ${port}|wc -l` -ne 1 ];then
       printf "MySQL is stopped...\n"
       exit
    else
       printf "Stoping MySQL...\n"
       ${CmdPath}/mysqladmin -u ${mysql_user} -p${mysql_pwd} -S /dat
a/${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: $0 {start|stop|restart}\n"
esac
```

一. 目录文件准备

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 -p /data/{3306,3307}/data
```

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：

```
[root@db03 ~]# tree /data/
/data/
├── 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的登录密码，所以必须授予**700**的权限，防止密码泄露

```
$ find /data -type f -name "mysql"|xargs chmod 700
$ find /data -type f -name "mysql"|xargs ls -l
-rwx------ 1 root root 1334 Jan  2 01:48 /data/3306/mysql
-rwx------ 1 root root 1334 Jan  2 01:49 /data/3307/mysql
```

## 二.初始化数据库

```
# 初始化3306实例数据
$ /application/mysql/scripts/mysql_install_db --basedir=/application/mysql/ --datadir=/data/3306/data/ --user=mysql
# 初始化3307实例数据
$ /application/mysql/scripts/mysql_install_db --basedir=/application/mysql/ --datadir=/data/3307/data/ --user=mysql
```

## 三.启动MySQL服务

```
$ /data/3307/mysql start
$ /data/3307/mysql start
```

## 四.检查mysqld服务

```
$ netstat -lntp|grep mysqld
或
$ ss -lntp|grep mysqld
```

## 五.添加至开机自启脚本

```
$ cat >>/etc/rc.local<<EOF
####start mysql####
/data/3306/mysql start
/data/3307/mysql start
EOF
```

六.快速添加一个多实例

```
$ mkdir -p /data/3308/data
$ \cp /data/3306/my.cnf  /data/3308/
$ \cp /data/3306/mysql  /data/3308/
$ sed -i 's/3306/3308/g' /data/3308/my.cnf
$ sed -i 's/server-id = 1/server-id = 8/g' /data/3308/my.cnf
$ sed -i 's/3306/3308/g' /data/3308/mysql
$ chown -R mysql.mysql /data/3308
$ chmod 700 /data/3308/mysql
$ cd /application/mysql/scripts
$ ./mysql_install_db --datadir=/data/3308/data --basedir=/applicatio
n/mysql --user=mysql
$ chown -R mysql:mysql /data/3308
$ /data/3308/mysql start
$ sleep 5
$ netstat -lntp|grep 3308
```

# 多实例启动、登录、关闭命令

一.启动MySQL，多实例的启动实质使用**mysqld_safe**指定**my.cnf**配置文件

```
$ /application/mysql/bin/mysqld_safe --defaults-file=/data/3306/my.cnf
&
```

二.本地登录MySQL，需指定socket登录

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

远程登录，首先要在MySQL里授权，然后使用 ip:port 登录

```
$ mysql -uroot -p123456 -P 3306 -h 172.16.1.53
```

三.关闭多实例MySQL，不能全部关闭，所以选择使用**mysqladmin**指定 `socket` 来关闭，这就是**/data/3306/mysql stop**的实质：

```
$ mysqladmin -uroot -p123456 -S /data/3306/mysql.sock shutdown
```

# FAQ

- 使用 `/data/3306/mysql` 脚本关闭时关闭不了，显示 `error: 'Access denied for user 'root'@'localhost' (using password: YES)'` 。
  **原因：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。

**原因：需要grant授权远程登录，user+host_ip才是一条验证条目。**

# MySQL启动、关闭命令的实质

启动mysql的过程：

```
$ /etc/init.d/mysqld start
$ ps -ef|grep mysql|grep -v grep
root 1307 1 0 17:08 ? 00:00:00 /bin/sh /application/mysql/bin/mysqld_safe -datadir=/application/mysql/data -pid-file=/application/mysql/data/db01.pid
mysql 1562 1307 0 17:08 ? 00:00:00 /application/mysql/bin/mysqld -basedir=/application/mysql -datadir=/application/mysql/data -plugin-dir=/application/mysql/lib/plugin -user=mysql -log-error=/application/mysql/data/db01.err -pid-file=/application/mysql/data/db01.pid -socket=/tmp/mysql.sock -port=3306
```

启动命令实质是执行mysql命令脚本中：

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

关闭命令实质是执行mysql命令脚本中：

```
${CmdPath}/mysqladmin -u ${mysql_user} -p${mysql_pwd} -S /data/${port}/mysql.sock shutdown
```

启动3306实例命令：`/data/3306/mysql start`
实质是执行：`mysqld_safe --defaults-file=/data/3306/my.cnf 2>&1 > /dev/null &`

停止3306实例命令：`/data/3306/mysql stop`
实质是执行：`mysqladmin -uroot -poldboy123 -S /data/3306/mysql.sock shutdown`

`/etc/init.d/mysqld start`相当于`mysqld_safe 2>&1 > /dev/null &`
`/etc/init.d/mysqld stop`相当于`kill pidnum`

**强调：**尽量不要野蛮粗鲁杀死数据库，生产高并发环境可能会引起数据丢失。
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

为了安全，不要在命令行输入mysql用户密码，可交互式输入密码，也可使用如下方法：
HISTCONTROL=ignorespace 《==历史记录会忽略记录命令行头部加空格的命令

# 设置MySQL命令提示符

方法一：临时生效

```
mysql> prompt \u@root \r:\m:\s->
mysql> prompt (\u@\h) [\d]>_            d表示显示当前库
PROMPT set to '(\u@\h) [\d]>_ '
(root@localhost) [mysql]>
```

方法二：永久生效 my.cnf中增加：

```
[mysql]
prompt=\u@oldboy \r:\m:\s->
即可显示登录mysql的用户、时间。
```

对Linux初学者的建议： http://oldboy.blog.51cto.com/2561410/1566703

# MySQL基础管理

## MySQL基础安全控制

1. 启动程序设置700，属主和用户组为mysql。
2. 为MySQL超级用户root设置密码。
3. 如果要求严格可以删除root用户，创建其它管理用户，如：admin。
4. 登录时尽量不要在命令行暴露密码，备份脚本中如果有密码，给设置700，属主和属组为mysql或root。
5. 删除默认存在的test库及无用的用户，用户只保留
   | root | 127.0.0.1 |
   | root | localhost |
6. 授权用户对应的主机不要用 “%”，权限不要给all，最小化授权，从库只给select。
7. 不要一个用户管所有的库，尽量专库专用户。
8. 清理mysql操作日志文件 ~/.mysql_history，可写定时脚本清理。
9. 禁止开发获取到web连接的密码，禁止开发连接操作生产对外的库。
10. phpmyadmin安全（见其它文档说明）。
11. 服务器禁止设置外网IP。
12. 放SQL注入（WEB），php.ini。

**其它参考以下博文：**
http://www.jb51.net/article/52257.htm
http://blog.sina.com.cn/s/blog_6d491a2f0100rook.html
http://blog.csdn.net/kobeyan/article/details/7591214

## 设置、修改密码，删除多于库、用户

```
$ mysqladmin password 123456 -S /data/3306/mysql.sock
$ mysqladmin password 123456 -S /data/3307/mysql.sock
$ mysqladmin password 123456 -S /data/3308/mysql.sock
$ mysql -uroot -p123456 -S /data/3306/mysql.sock
mysql> drop database test;
mysql> use mysql;
mysql> drop user root@'::1';
mysql> drop user root@'db02';
mysql> drop user ''@'db02';
mysql> drop user ''@'localhost';
# 以下两个库实例亦执行同样操作
$ mysql -uroot -p123456 -S /data/3307/mysql.sock
$ mysql -uroot -p123456 -S /data/3308/mysql.sock

# 替换启动脚本中的密码
$ sed -i '13 s#oldboy#123456#g' /data/3306/mysql
$ sed -i '13 s#oldboy#123456#g' /data/3307/mysql
$ sed -i '13 s#oldboy#123456#g' /data/3308/mysql
```

**修改密码几种方法**

- ①在shell命令行使用`mysqladmin`修改
  设置密码方法：`mysqladmin -uroot password '123456'`
  修改密码方法：`mysqladmin -uroot -p123456 password '123456'`
  多实例设置和修改密码都要加`-S`指定`socket`。
- ②使用update的方式修改密码，必须刷新权限表

```
mysql> update mysql.user SET password=PASSWORD("123456") WHERE user='root' and host='localhost';
mysql> flush privileges;
```

- ③使用变量，修改已登录用户的密码

```
mysql> set password=password('123456')
```

# 丢失MySQL的root用户密码，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MySQL。
**步骤：**
①杀死mysql进程`kill pid`，不要用`pkill`或`kill mysqld`。
②使用`--skip-grant-tables`参数启动MySQL

```
# 单实例
$ mysql_safe --skip-grant-tables -user=mysql &
# 多实例
$ mysql_safe --defaults-file=/data/3306/my.cnf --skip-grant-tables &
```

③登录mysql

```
# 单实例
$ mysql
# 多实例
$ mysql -S /data/3306/mysql.sock
```

④使用update修改密码，完毕后刷新权限表

```
mysql> update mysql.user SET password=PASSWORD("123456") WHERE user='root' and host='localhost';
mysql> flush privileges;
```

⑤退出mysql，关闭MySQL进程

```
# 单实例
$ /etc/init.d/mysqld stop
# 多实例关闭前，得修改mysql启动脚本中的密码
$ sed -i '13 s#123456#123456#g' /data/3306/mysql
$ /data/3306/mysql stop
```

⑥再次启动MySQL后就可使用新密码登录mysql了

# SQL结构化语言

SQL语句常见分类：

| 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：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) | 针对数据库里的表的数据进行操作、记录 |

**对于高级运维来说，需要熟练掌握的是DDL和DCL ，需要了解DML。**

> **注意**
> 凡事看帮助，要形成潜意识看帮助的习惯，使用SQL语言前，不明之处请help。

附：**数据库权限总览**

| 权限 | 权限级别 | 权限说明 |
| --- | --- | --- |
| SELECT | 表 | 查询权限 |
| INSERT | 表 | 插入权限 |
| UPDATE | 表 | 更新权限 |
| DELETE | 表 | 删除权限 |
| CREATE | 数据库、表或索引 | 创建数据库、表或索引权限 |
| DROP | 数据库或表 | 删除数据库、表或用户 |
| GRANT OPTION | 数据库、表或保存的程序 | 赋予权限选项 |
| RELOAD | 服务器管理 | 执行flush-privileges、reload等命令的权限 |
| SHUTDOWN | 服务器管理 | 关闭数据库权限 |
| PROCESS | 服务器管理 | 查看进程权限 |
| FILE | 服务器主机上的文件访问 | 文件访问权限 |
| REFERENCES | 数据库或表 | 建立约束 |
| INDEX | 表 | 索引权限 |
| ALTER | 表 | 更改表，如：添加字段、索引等 |
| SHOW DATABASES | 服务器管理 | 查看数据库权限 |
| SUPER | 服务器管理 | 执行kill线程权限 |
| CREATE TEMPORARY TABLES | 服务器管理 | 创建临时表权限 |
| LOCK TABLES | 服务器管理 | 锁表权限 |
| EXECUTE | 存储过程 | 执行存储过程权限 |
| REPLICATION SLAVE | 服务器管理 | 复制权限 |

| | | |
|---|---|---|
| REPLICATION CLIENT | 服务器管理 | 复制权限 |
| CREATE VIEW | 视图 | 创建视图权限 |
| SHOW VIEW | 视图 | 查看视图权限 |
| CREATE ROUTINE | 存储过程 | 创建存储过程权限 |
| ALTER ROUTINE | 存储过程 | 更改存储过程权限 |
| CREATE USER | 服务器管理 | 创建用户权限 |
| EVENT | 数据库 | 事件调度器，删除、创建、修改事件权限 |
| TRIGGER | 表 | 创建、删除触发器权限 |
| CREATE TABLESPACE | 表 | 创建表空间 |

# create命令

**①创建数据库**，具体语法请`help`：

```
mysql> help create database;
Name: 'CREATE DATABASE'
Description:
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。
```

例子：

```
mysql> create database bbs; #创建库，默认使用使用系统数据库的字符集
mysql> create database bbs_gbk DEFAULT CHARACTER SET gbk COLLATE gbk_chinese_ci;              #创建gbk字符集数据库
mysql> create database bbs_utf8 DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;            #创建utf8字符集数据库
mysql> show create database bbs_utf8\G   #查看建库语句
*************************** 1. row ***************************
       Database: bbs_utf8
Create Database: CREATE DATABASE `bbs_utf8` /*!40100 DEFAULT CHARACTER SET utf8 */
1 row in set (0.00 sec)
```

字符集和collate校对字符集可通过`show`查询：

```
mysql> show character set;
```

**②创建表**，具体语法请`help`：

```
mysql> help create table;
```

例子：

```
mysql> use bbs_utf8;  #切换到bbs_utf8库，使用use dbname切换数据库
Database changed
mysql> create table student(
 id int(4) not null AUTO_INCREMENT,
 name char(20) not null,
 age tinyint(2)  NOT NULL default '0',
 dept varchar(16)  default NULL,
 primary key(id),
 KEY index_name(name)
 );
```

**创建用户**，具体语法请`help`：

```
mysql> help create user;
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';  #诶，有例子
```

# show命令

`show`命令可以查看很多信息，具体请`help show`：

```
mysql> help show;
mysql> help show databases;
Name: 'SHOW DATABASES'
Description:
Syntax:
SHOW {DATABASES | SCHEMAS}
    [LIKE 'pattern' | WHERE expr]
注：查看所有数据库，也可以通过like模糊查询或者where指定条件查询
```

例：显示数据库

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| bbs_utf8           |
| mysql              |
| oldboy             |
| performance_schema |
+--------------------+
5 rows in set (0.07 sec)

mysql> show databases like '%b%';
+----------------+
| Database (%b%) |
+----------------+
| bbs_utf8       |
| oldboy         |
+----------------+
2 rows in set (0.01 sec)

mysql> show databases where 'bbs_utf8';
Empty set, 1 warning (0.00 sec)
```

例子：显示表

```
mysql> show tables;
+--------------------+
| Tables_in_bbs_utf8 |
+--------------------+
| student            |
+--------------------+
1 row in set (0.00 sec)
```

## drop命令

`drop`用来删除，具体语法请`help drop`

```
mysql> help drop;
mysql> help drop database; #删除数据库
Syntax:
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name

mysql> help drop user;      #删除用户
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

mysql> help drop table;     #删除表
Syntax:
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
#删除整张表

mysql> help drop index;     #删除索引
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，即相当于：alter table table_name drop {index|key} index_name
```

**两种删除用户的方法：**

1. `drop user "user"@"主机域"`
2. `delete from mysql.user where user='root' and host = 'oldboy';`#此法是对于大写、特殊字符的用户

# alter命令

`alter`用于修改，具体语法请`help alter`

```
mysql> help alter;

mysql> help alter database;
Name: 'ALTER DATABASE'
Description:
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.

mysql> help alter table;
Syntax:
ALTER [ONLINE | OFFLINE] [IGNORE] TABLE tbl_name
    [alter_specification [, alter_specification] ...]
    [partition_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。
```

例子：增删改表的字段
命令语法：`alter table` 表名 `add` 字段 类型 其它；

```
mysql> alter table student add sex char(4);
Query OK, 5 rows affected (0.11 sec)
Records: 5  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    |                |
| sex   | char(4)     | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

# 在指定位置增加
mysql> alter table student add shouji varchar(20) after age;
Query OK, 5 rows affected (0.12 sec)
Records: 5  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       |                |
| shouji | varchar(20) | YES  |     | NULL    |                |
| dept   | varchar(16) | YES  |     | NULL    |                |
| sex    | char(4)     | YES  |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)

mysql> alter table student add qq varchar(20) first;
Query OK, 5 rows affected (0.16 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> desc student;
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| qq     | varchar(20) | YES  |     | NULL    |                |
| id     | int(4)      | NO   | PRI | NULL    | auto_increment |
| name   | char(20)    | NO   | MUL | NULL    |                |
| age    | tinyint(2)  | NO   |     | 0       |                |
| shouji | varchar(20) | YES  |     | NULL    |                |
| dept   | varchar(16) | YES  |     | NULL    |                |
| sex    | char(4)     | YES  |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

**删除字段**

```
mysql> alter table student drop sex;
Query OK, 5 rows affected (0.16 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> alter table student drop shouji,drop qq;
Query OK, 5 rows affected (0.13 sec)
Records: 5  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 (0.00 sec)
```

**修改一个字段**

```
alter table user MODIFY new1 VARCHAR(10);   #修改一个字段的类型
alter table user CHANGE new1 new4 int;   #修改一个字段的名称，此时一定要重新指定该字段的类型
```

# 生产场景多个复杂增删改多字段

```
1、增加1个字段：
ALTER TABLE `etiantian` ADD `FIRSTPHOTO_URL` varchar(255) default NULL COMMENT '第一张图片URL'

2、增2个字段：
ALTER TABLE `basic` ADD `adhtml_top`  varchar(1024) default NULL COMMENT '顶部广告html' ,
                                                ADD `adhtml_right` varchar(1024) default NULL COMMENT '右侧广告html' ;

3、改变字段：
alter table ett_ambiguity change ambiguity_state  ambiguity_state tinyint  comment '状态，默认1=正常，0=失效';
ALTER TABLE `ett_photo`
MODIFY COLUMN `PHOTO_DESCRIPTION` varchar(512) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL COMMENT '描述' AFTER PHOTO_TITLE`;

4、修改字段类型：
mysql> alter table test modify age char(4) after name;
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0

5、修改字段名称
mysql> alter table test change age oldboyage char(4) after name;
Query OK, 6 rows affected (0.01 sec)
Records: 6  Duplicates: 0  Warnings: 0
提示：工作中添加字段需求来自开发，运维或DBA拿着开发给的语句执行。
```

# grant命令

`grant`命令用于授权，具体语法请`help grant`

```
mysql> help grant;
简单来说就是：grant 权限 on 数据库.表 to 用户@主机 identified by 密码 with grant option
```

**给mysql中库授权用户的两种方法**
第一种方法：

```
1. CREATE USER '用户'@'主机' IDENTIFIED BY '密码';
2. GRANT ALL ON dbname.* TO '用户'@'主机';
```

以上两条命令相当于下面一条命令，即第二种方法：

```
1. GRANT ALL ON  dbname.* TO '用户'@'主机' IDENTIFIED BY '密码';
```

例子：

```
1. mysql> grant select,update,delete,insert on bbs_utf8.* to bbs@'172.1
   6.1.%' identified by '123456';
2. Query OK, 0 rows affected (0.01 sec)
3. mysql> flush privileges;  #记得要刷新权限表
4.
5. mysql> show grants for bbs@'172.16.1.%';  #查看用户权限
6. +-------------------------------------------------------------------
   ---------------------------------------+
7. | Grants for bbs@172.16.1.%
   |
8. +-------------------------------------------------------------------
   ---------------------------------------+
9. | GRANT USAGE ON *.* TO 'bbs'@'172.16.1.%' IDENTIFIED BY PASSWORD '*6B
   B4837EB74329105EE4568DDA7DC67ED2CA2AD9' |
10. | GRANT SELECT, INSERT, UPDATE, DELETE ON `bbs_utf8`.* TO 'bbs'@'172.1
    6.1.%'                                  |
11. +-------------------------------------------------------------------
    ---------------------------------------+
12. 2 rows in set (0.00 sec)
```

# revoke命令

`revoke`用于撤销用户权限，具体语法请`help revoke`

```
mysql> help revoke;
Name: 'REVOKE'
Description:
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] ...
# 例子：REVOKE INSERT ON *.* FROM 'jeffrey'@'localhost';
# 简单说就是revoke 权限 on 数据库.表 from 用户@主机
```

## commit语句和rollback语句

开启事务后有效，提交和回滚。

## insert命令

`insert`用于插入数据，具体语法请`help insert`
命令语法：`insert into` <表名> [(<字段名1>[,...<字段名n>])] `values` (值1)[,(值n)]

```
mysql> help insert;
Name: 'INSERT'
Description:
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] ... ]
```

**往表里插入数据的不同的语法例子**

a. 按规矩指定所有的列名，并且每列都插入值

```
mysql> insert into test(id,name) values(1,'oldboy');
```

b. 由于id列为自增的，所以可只在name列插入值

```
mysql> insert into test(name) values('oldgirl');
```

c. 如果不指定列，就要按规矩为每列都插入恰当的值

```
mysql> insert into test values(3,'inca');
```

d. 批量插入数据方法，提升效率

```
mysql> insert into test values(4,'zuma'),(5,'kaka');
```

例子：

```
mysql> insert into student(id,name,age,dept) values(1,'lzyi',23,'173cm'),(2,'lz',24,'175cm'); #批量插入
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> insert into student values(3,'lb',22,'176cm');
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。

## select命令

`select`用于查询，具体语法请`help select`
命令语法：`select` <字段1,字段2,...> `from` <表名> `where` <表达式>
其中，`select,from,where`是不能随便改的，是关键字，支持大小写。

```
mysql> help select;
Name: 'SELECT'
Description:
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]]
```

例子：查询整张表的内容

```
mysql> select * from student;
+----+------+-----+-------+
| id | name | age | dept  |
+----+------+-----+-------+
|  1 | lzyi |  23 | 173cm |
|  2 | lz   |  24 | 175cm |
|  3 | lb   |  22 | 176cm |
+----+------+-----+-------+
3 rows in set (0.00 sec)
```

例子：通过`limit`关键字限制显示

```
mysql> select * from student limit 2;
+----+------+-----+-------+
| id | name | age | dept  |
+----+------+-----+-------+
|  1 | lzyi |  23 | 173cm |
|  2 | lz   |  24 | 175cm |
+----+------+-----+-------+
2 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 | lb      |  22 | 176cm |
|  4 | laoyu   |  23 | 173cm |
|  5 | leishen |  22 | 174cm |
+----+---------+-----+-------+
3 rows in set (0.00 sec)
```

例子：通过`where`指定条件查询，`order by`排序

```
mysql> select name,age,dept from student where age < 23 order by age;
+---------+-----+-------+
| name    | age | dept  |
+---------+-----+-------+
| hah     |  21 | 172cm |
| lb      |  22 | 176cm |
| leishen |  22 | 174cm |
+---------+-----+-------+
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。

例子：导出结果至本地系统文件

```
mysql> select * from student into outfile '/tmp/student.txt';
Query OK, 6 rows affected (0.00 sec)

mysql> system cat /tmp/student.txt; #使用system跳出mysql执行命令
1   lzyi    23  173cm
2   lz      24  175cm
3   lb      22  176cm
4   laoyu   23  173cm
5   leishen 22  174cm
6   hah     21  172cm
```

**也可使用tee命令**

`tee`命令会从标准输入设备读取数据，将其内容输出到标准输出设备，同时保存成文件。

```
mysql> tee /tmp/test.txt   #tee会记录以下所有操作
Logging to file '/tmp/test.txt'
mysql> select * from student;
+----+---------+-----+-------+
| id | name    | age | dept  |
+----+---------+-----+-------+
|  1 | lzyi    |  23 | 173cm |
|  2 | lz      |  24 | 175cm |
|  3 | lb      |  22 | 176cm |
|  4 | laoyu   |  23 | 173cm |
|  5 | leishen |  22 | 174cm |
+----+---------+-----+-------+
5 rows in set (0.00 sec)

mysql> system cat /tmp/test.txt
mysql> select * from student;
+----+---------+-----+-------+
| id | name    | age | dept  |
+----+---------+-----+-------+
|  1 | lzyi    |  23 | 173cm |
|  2 | lz      |  24 | 175cm |
|  3 | lb      |  22 | 176cm |
|  4 | laoyu   |  23 | 173cm |
|  5 | leishen |  22 | 174cm |
+----+---------+-----+-------+
5 rows in set (0.00 sec)
```

# update命令

`update`用于更新，具体语法请`help update`

命令语法：`update` 表名 `set` 字段=新值, .... `where` 条件（一定要注意条件）

```
mysql> help update;
Name: 'UPDATE'
Description:
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。

例子：将`hah`的`age`改为33

```
mysql> update student set age=33 where name='hah';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from student where name = 'hah';
+----+------+-----+-------+
| id | name | age | dept  |
+----+------+-----+-------+
|  6 | hah  |  33 | 172cm |
+----+------+-----+-------+
1 row in set (0.00 sec)
```

严重的案例：
不带条件更改所有表的记录，只能用备份的数据恢复

# -U参数，防止误操作

```
 -U, --safe-updates  Only allow UPDATE and DELETE that uses keys.
# 只允许更新和删除使用密码
```

例子：

```
$ mysql -uroot -p623913 -S /data/3306/mysql.sock -U
mysql> update student set age = 0;
ERROR 1175 (HY000): You are using safe update mode and you tried to up
date a table without a WHERE that uses a KEY column
# 看，有-U保护，误操作被阻止了。
```

DBA防止自己出错的设置：
http://oldboy.blog.51cto.com/2561410/1321061

# delete命令

`delete`用于删除，具体语法请`help delete`

```
mysql> help delete;
Name: 'DELETE'
Description:
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]
```

**注：**`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。

例子：

```
mysql> delete from student where name='hah';
Query OK, 1 row affected (0.00 sec)

mysql> select * from student;
+----+---------+-----+-------+
| id | name    | age | dept  |
+----+---------+-----+-------+
|  1 | lzyi    |  23 | 173cm |
|  2 | lz      |  24 | 175cm |
|  3 | lb      |  22 | 176cm |
|  4 | laoyu   |  23 | 173cm |
|  5 | leishen |  22 | 174cm |
+----+---------+-----+-------+
5 rows in set (0.00 sec)
```

# 删除表

删除表的两种方法：`truncate` 和 `delete`

**①truncate**

```
mysql> help truncate table;
Name: 'TRUNCATE TABLE'
Description:
Syntax:
TRUNCATE [TABLE] tbl_name

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

**②delete**

命令语法：`delete from` 表名 `where` 表达式

```
mysql> help delete;

mysql> delete from student;
Query OK, 5 rows affected (0.08 sec)

mysql> select * from student;
Empty set (0.00 sec)   # 已为空

mysql> delete from test where name='wahaha'; #删除指定字段
```

**③truncate和delete的区别**

```
① mysql> delete from test;
② mysql> truncate table test;
```

- `truncate table test`是物理删除，速度更快，直接清空对应数据的物理文件
- `delete from test`是逻辑清除，速度慢，按行删除

# rename命令

`rename`用于改名，具体语法请`help rename`

```
mysql> help rename;
   ALTER EVENT
   ALTER TABLE
   RENAME TABLE
   RENAME USER
```

例子：

```
mysql> show tables;
+--------------------+
| Tables_in_bbs_utf8 |
+--------------------+
| student            |
+--------------------+
1 row in set (0.00 sec)
命令语法：rename table 原表名 to 新表名
mysql> rename table student to stu;
Query OK, 0 rows affected (0.08 sec)

mysql> show tables;
+--------------------+
| Tables_in_bbs_utf8 |
+--------------------+
| stu                |
+--------------------+
1 row in set (0.00 sec)
```

**`alter`亦可改表名**

```
mysql> alter table stu rename to test;
Query OK, 0 rows affected (0.10 sec)

mysql> show tables;
+--------------------+
| Tables_in_bbs_utf8 |
+--------------------+
| test               |
+--------------------+
1 row in set (0.00 sec)
```

# MySQL基础操作

数据库服务器–>数据库（多个实例）–>多个库–>多个表–>多个字段行列（数据）

## 建库

```
mysql> create database bbs;
Query OK, 1 row affected (0.00 sec)
```

## 建表

```
mysql> use bbs;
Database changed
mysql> create table student(
id int(4) not null,
name char(20) not null,
age tinyint(2)  NOT NULL default '0',
dept varchar(16)  default NULL
);

mysql> show create table student\G  #查看建表信息
*************************** 1. row ***************************
       Table: student
Create Table: CREATE TABLE `student` (
  `id` int(4) NOT NULL,
  `name` char(20) NOT NULL,
  `age` tinyint(2) NOT NULL DEFAULT '0',
  `dept` varchar(16) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

mysql> desc student;   #查看表结构
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(4)      | NO   |     | NULL    |       |
| name  | char(20)    | NO   |     | NULL    |       |
| age   | tinyint(2)  | NO   |     | 0       |       |
| dept  | varchar(16) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

## 往表里插入数据

```
mysql> insert into student values(4,'l',34,'179cm'),(5,'wen',27,'178cm');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from student;
+----+------+-----+-------+
| id | name | age | dept  |
+----+------+-----+-------+
|  1 | lzyi |  23 | 172cm |
|  2 | lzy  |  33 | 172cm |
|  3 | lz   |  13 | 150cm |
|  4 | l    |  34 | 179cm |
|  5 | wen  |  27 | 178cm |
+----+------+-----+-------+
5 rows in set (0.00 sec)
```

**强调补充**：平时登录网站发帖、发博文，实质上都是调用web网站的程序连接MySQL数据库，通过上述的insert语句把帖子、博文数据存入数据库的。

# 数据类型

## 数值类型

Mysql支持所有标准SQL中的数值类型，其中包括严格数据类型(INTEGER,SMALLINT,DECIMAL,NUMBERIC)，以及近似数值数据类型(FLOAT,REAL,DOUBLE PRESISION),并在此基础上进行扩展。
扩展后增加了TINYINT,MEDIUMINT,BIGINT这3种长度不同的整形，并增加了BIT类型，用来存放位数据。

| 整数类型 | 字节 | 有符号区间 | 无符号区间 |
|---|---|---|---|
| TINYINT | 1 | -128~127 | 0~255 |
| SMALLINT | 2 | -32768~32767 | 0~65535 |
| MEDIUMINT | 3 | -8388608~8388607 | 0~1677215 |
| INT/INTEGER | 4 | -2147483648~2147483647 | 0~4294967295 |
| BIGINT | 8 | -9223372036854775808~9223372036854775807 | 0~18446744073709551615 |

## 字符串类型

MySQL 提供了8个基本的字符串类型,分别:CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、ENUM 各SET等多种字符串类型。
可以存储的范围从简单的一个字符到巨大的文本块或二进制字符串数据。

| 字符串类型 | 字节大小 | 描述及存储需求 |
|---|---|---|
| CHAR(M) | M个字节，0-255字节 | 定长字符串 |
| VARCHAR(M) | 0-255字节 | 变长字符串 |
| TINYBLOB | 0-255字节 | 不超过 255 个字符的二进制字符串 |
| TINYTEXT | 0-255字节 | 短文本字符串 |
| BLOB | 0-65535字节 | 二进制形式的长文本数据 |
| TEXT | 0-65535字节 | 长文本数据 |
| MEDIUMBLOB | 0-16 777 215字节 | 二进制形式的中等长度文本数据 |
| MEDIUMTEXT | 0-16 777 215字节 | 中等长度文本数据 |
| LOGNGBLOB | 0-4 294 967 295字节 | 二进制形式的极大文本数据 |
| LONGTEXT | 0-4 294 967 295字节 | 极大文本数据 |
| VARBINARY(M) | | 允许长度0-M个字节的定长字节符串，值的长度+1个字节 |
| BINARY(M) | M | 允许长度0-M个字节的定长字节符串 |

| 列类型 | 存储需求 |
|---|---|
| CHAR(*M*) | *M*个字节，0 <= *M* <= 255 |
| VARCHAR(*M*) | *L*+1个字节，其中*L* <= *M* 且0 <= *M* <= 65535(参见下面的注释) |
| BINARY(*M*) | *M*个字节，0 <= *M* <= 255 |
| VARBINARY(*M*) | *L*+1个字节，其中*L* <= *M* 且0 <= *M* <= 255 |
| TINYBLOB, TINYTEXT | *L*+1个字节，其中$L < 2^8$ |
| BLOB, TEXT | *L*+2个字节，其中$L < 2^{16}$ |
| MEDIUMBLOB, MEDIUMTEXT | *L*+3个字节，其中$L < 2^{24}$ |
| LONGBLOB, LONGTEXT | *L*+4个字节，其中$L < 2^{32}$ |
| ENUM('*value1*','*value2*',...) | 1或2个字节，取决于枚举值的个数(最多65,535个值) |
| SET('*value1*','*value2*',...) | 1、2、3、4或者8个字节，取决于set成员的数目(最多64个成员) |

1. INT[(M)]型：正常大小整数类型
2. CHAR(M)型：定长字符串类型，当存储时，总是用空格填满右边到指定的长度
3. VARCHAR型：变长字符串类型
   详情见MySQL 5.1手册

**小结**：

1. char定长，不够的用空格补全，浪费存储空间，查询速度快，多数系统表字段默认使用。
2. varchar变长，查询速度慢。

VARCHAR、BLOB和TEXT类是变长类型。每个类型的存储需求取决于列值的实际长度(用前面的表中的L表示)，而不是该类型的最大可能的大小。例如，VARCHAR(10)列可以容纳最大长度为10的字符串。实际存储需求是字符串(L)的长度，加上一个记录字符串长度的字节。对于字符串'abcd'，L是4，存储需要5个字节。

# 日期和时间类型

在处理日期和时间类型的值时，MySQL 带有 5 个不同的数据类型可供选择。它们可以被分成简单的日期、时间类型，和混合日期、时间类型。
根据要求的精度，子类型在每个分类型中都可以使用，并且 MySQL 带有内置功能可以把多样化的输入格式变为一个标准格式。

| 类型 | 大小(字节) | 范围 | 格式 | 用途 |
| --- | --- | --- | --- | --- |
| DATE | 4 | 1000-01-01/9999-12-31 | YYYY-MM-DD | 日期值 |
| TIME | 3 | '-838:59:59'/'838:59:59' | HH:MM:SS | 时间值或持续时间 |
| YEAR | 1 | 1901/2155 | YYYY | 年份值 |
| DATETIME | 8 | 1000-01-01 00:00:00/9999-12-31 23:59:59 | YYYY-MM-DD HH:MM:SS | 混合日期和时间值 |
| TIMESTAMP | 4 | 1970-01-01 00:00:00/2037年某时 | YYYYMMDD HHMMSS | 混合日期和时间值，时间戳 |

# 索引

**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，以索引列为查询条件时可以加快数据表的查询速度。 这是优化的重要内容之一。
比喻：ext[2-4]文件系统的inode、block

查询数据库，按主键查询是最快的，每个表只能有一个主键列，但是可以有多个普通索引列。主键列要求列的所有内容必须唯一，而普通索引列不要求内容唯一。

**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。

# 建立主键索引的方法

**①在建表时，可以增加建立主键索引的语句**

```
create table student(
id int(4) not null AUTO_INCREMENT,   #自增
name char(20) not null,
age tinyint(2)  NOT NULL default '0',
dept varchar(16)  default NULL,
primary key(id),                     #主键索引
KEY index_name(name)                 #普通索引
);

mysql> desc student;                 #查看表结构
+-------+-------------+------+-----+---------+----------------+
| 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建表后通过alter命令增加主键索引（不这样干）**

```
a. 主键列不能重复创建，必须先删除上面配置
mysql> alter table student drop primary key;

b. 利用alter命令修改id列为自增主键列
mysql> alter table student change id id int primary key auto_incremen
t;
mysql> desc student;
```

# 建立普通索引的方法

**①在建表时，可以增加建立普通索引的语句**

```
create table student(
id int(4) not null AUTO_INCREMENT,    #自增
name char(20) not null,
age tinyint(2)  NOT NULL default '0',
dept varchar(16)  default NULL,
primary key(id),                      #主键索引
KEY index_name(name)                  #name字段是普通索引
);

然后通过：
mysql> show index from student\G;     #查看索引
```

**②通过alter或者create增加索引**

通过alter增加索引：

```
a. 先删除索引
mysql> alter table student drop index index_name; #删除索引
b. 增加索引
mysql> alter table student add index index_name(name); #alter添加索引
```

通过create增加索引

```
mysql> create index index_age on student(age);
Query OK, 0 rows affected (0.17 sec)
Records: 0  Duplicates: 0  Warnings: 0

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_age
  Seq_in_index: 1
  Column_name: age
    Collation: A
  Cardinality: 0
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
2 rows in set (0.00 sec)
```

# 删除索引

### ①通过alter删除索引

```
mysql> alter table student drop index index_age;
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

### ②通过drop删除索引

```
mysql> drop index index_dept on student;
Query OK, 0 rows affected (0.17 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   |     | NULL    |                |
| age   | tinyint(2)  | NO   |     | 0       |                |
| dept  | varchar(16) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

> 生产场景：
> 数据量很大的时候，不适合建立索引，会影响用户访问，曾经400-500万条记录的表，建立索引，花了90-180秒。尽量选在业务低谷时建立索引。

# 创建前缀索引

当遇到表中比较大的列时，列内容的前N个字符在所有内容中已经接近唯一时，这是可以对列的前N个字符建立索引，而无需对整个列建立索引，这样节省创建索引占用的系统空间，及降低读取和更新维护索引消耗的系统资源。

```
mysql> create index index_dept on student(dept(8));
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
#这个语句等同于alter table student add index index_dept(dept(8));

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  | MUL | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

**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: 8
       Packed: NULL
         Null: YES
   Index_type: BTREE
      Comment:
Index_comment:
3 rows in set (0.00 sec)
```

# 联合索引

若查询数据的条件是多列时，我们可以为多个查询的列创建联合索引，甚至可以为多列的前N个字符列创建联合索引。

**联合索引有前缀特性**：

> 三个字段：
> a，b，c
> 只有一下组合才可以走索引：
> a
> ab
> abc

index(a,b,c)仅a，ab，abc三个查询条件列可以走索引。b,bc,ac,c等无法使用索引了。
尽量把最常用作为查询条件的列，放在第一位置。

> 由于这个特性，我们设置索引的时候需要注意：
> 1、经常用于索引的字段放在前面。
> 2、多个字段组合，唯一值是对字段的组合，所以唯一值增多。

**实践：**

```
先删除以前的索引
mysql> alter table student drop index index_name;
Query OK, 0 rows affected (0.12 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> drop index index_dept on student;
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> create index index_name_dept on student(name,dept);
Query OK, 0 rows affected (0.11 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 (0.00 sec)

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_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: NULL
       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)
```

```
mysql> alter table student drop index index_name_dept;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

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:
1 row in set (0.00 sec)

mysql> create index index_name_dept on student(name(8),dept(10));
Query OK, 0 rows affected (0.24 sec)
Records: 0  Duplicates: 0  Warnings: 0

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_dept
 Seq_in_index: 1
  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: 10
       Packed: NULL
         Null: YES
   Index_type: BTREE
      Comment:
Index_comment:
3 rows in set (0.00 sec)
```

**尽量在唯一值多的大表上建立索引，如何统计表记录的唯一值了？**

```
mysql> select * from student;
+----+------+-----+-------+
| id | name | age | dept  |
+----+------+-----+-------+
|  1 | lzyi |  23 | 172cm |
|  2 | lzy  |  33 | 172cm |
|  3 | lz   |  13 | 150cm |
+----+------+-----+-------+
3 rows in set (0.00 sec)

mysql> select distinct id from student;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
+----+
3 rows in set (0.00 sec)

mysql> select count(distinct id) from student;
+--------------------+
| count(distinct id) |
+--------------------+
|                  3 |
+--------------------+
1 row in set (0.00 sec)
```

# 唯一索引

创建唯一索引（非主键），属于非重点，需时学习。

```
mysql> create unique index uni_ind_name on student(name);
Query OK, 0 rows affected (0.04 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   | UNI | NULL    |                |
| age   | tinyint(2)  | NO   |     | 0       |                |
| dept  | varchar(16) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

# 建表语句加入联合索引：例如某sns产品生产正式建表语句

```
use sns;
set names gbk;
CREATE TABLE `subject_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 '素材的主键',
  `subject_title` varchar(255) NOT NULL COMMENT '素材的名称',
  `edit_user_nick` 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(1) 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_primary_key`(32)),  #括号内的32表示对前32个字符做前缀索引。
  KEY `IDX_SUBJECT_TITLE` (`subject_title`(32))
  KEY `index_nick_type` (`edit_user_nick`(32),`subject_type`)  #联合索引，此行为新加的，用于给大家讲解的。实际表语句内没有此行。
) ENGINE=InnoDB AUTO_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。
```

# 创建索引总结

**①索引列的创建及生效条件**：

> 问题1：既然索引可以加快查询速度，那么就给所有的列建索引吧？
> 解答：因为索引不但占用系统空间，而且更新数据时还需要维护索引数据的，因此，索引是一把双刃剑，并不是越多越好，例如：数十到几百行的小表上无需建立索引，插入更新频繁，读取比较少的表要少建立索引。
>
> 问题2：需要在哪些列上创建索引才能加快查询速度呢？
> 解答：select user,host from mysql.user where password=....，索引一定要创建在where后的条件列上，而不是select后的选择数据的列上。另外，我们要尽量选择在唯一值多的大表上的列建立索引，例如，男女性别列唯一值少，不适合建立索引。

**②创建索引命令总结**

```
# 创建主键索引：
alter table student change id id-num int primary key auto_increment;
# 删除主键索引：
alter table student drop primary key;
# 创建普通索引：
alter table student add index index_dept(dept(8));
# 根据列的前N个字符创建索引：
create index index_dept on student(dept(8));
# 根据多个列创建联合索引：
create index ind_name_dept on student(name,dept);
# 根据多个列的前N个字符创建联合索引：
create index ind_name_dept on student(name(8),dept(10));
# 创建唯一索引：
create unique index uni_ind_name on student(name);
# 删除普通索引与唯一索引：
alter table student drop index index_dept;
drop index index_dept on student;
```

**③创建索引知识总结**

1. 索引类似书籍的目录，会加快数据查询的速度。
2. 要在表的列（字段）上创建索引。
3. 索引会加快查询速度，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. 索引类型有 BTREE（默认）和哈希（适合做缓存（内存数据库））等 。

# explain测试查询是否走索引

```
mysql> help explain;
Name: 'EXPLAIN'
Description:
Syntax:
{EXPLAIN | DESCRIBE | DESC}
    tbl_name [col_name | wild]

{EXPLAIN | DESCRIBE | DESC}
    [explain_type] SELECT select_options

explain_type:
    EXTENDED
  | PARTITIONS
```

测试数据：

```
mysql> select * from student;
+----+------+-----+-------+
| id | name | age | dept  |
+----+------+-----+-------+
|  1 | lzyi |  23 | 172cm |
|  2 | lzy  |  33 | 172cm |
|  3 | lz   |  13 | 150cm |
|  4 | l    |  34 | 179cm |
|  5 | wen  |  27 | 178cm |
+----+------+-----+-------+
5 rows in set (0.00 sec)
```

测试1：
通过`explain`可以看`select`查询是否走索引

```
mysql> explain select * from student where name='lzyi'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: student
         type: ref
possible_keys: index_name_dept
          key: index_name_dept   #所走的索引
      key_len: 24
          ref: const
         rows: 1                 #遍历的行数
        Extra: Using where
1 row in set (0.00 sec)
```

测试2：

```
mysql> explain select * from student where name like 'lzy%'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: student
         type: range
possible_keys: index_name_dept
          key: index_name_dept
      key_len: 24
          ref: NULL
         rows: 2
        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3：

```
mysql> explain select * from student where name like '%zy'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: student
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 5
        Extra: Using where
1 row in set (0.00 sec)
# 模糊查询%号在前面，不会走索引。
```

# 企业案例：利用explain优化SQL语句

**①抓慢SQL**

A. 救火，紧急且重要，现场抓慢SQL语句。

```
mysql> show full processlist;  #登录数据库现场抓，隔五秒抓一次
或者，在命令行抓，建议在命令行操作，不要进入MySQL抓
$ mysql -uroot -p123456 -S /data/3306/mysql.sock -e "show full process
list\G"|egrep -vi "sleep"
```

B. 未雨绸缪，重要不紧急，分析慢查询日志

在`my.cnf`文件的`mysqld`设置启用慢查询记录的参数记录慢查询语句：

```
$ vim /data/3306/my.cnf
[mysqld]
long_query_time = 1                      #超过2秒，记录到log里
log-slow-queries = /data/3306/slow.log #没有走索引的语句，记录到log里
log_queries_not_using_indexes            #log文件
```

**②使用explain检查索引执行情况**

```
explain select * from student where name='lzyi'\G
explain select * from student where name like 'lzy%'\G
explain select SQL_NO_CACHE * from test where name='lzyi'\G
```

**③对需要建索引的条件列建立索引**
首先统计where条件字段中的唯一值数量，然后建立索引或联合索引（与开发、DBA合作完成）。
注意：生产场景，大表不能高峰期建立索引，例如：300万记录情况下。

**④使用分析慢查询SQL的工具–mysqlsla（每天早晨发邮件）**
mysqlsla切割MySQL慢查询日志，去重分析后发给相关人员。
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：

```
1、mv，reload进程或cp，>清空
2、定时任务
mv /data/3306/slow.log /opt/$(date +%F)_slow.log
mysqladmin -uroot -poldboy -S /data/3306/mysql.sock flush-logs
mysqlsla /backup/$(date +%F)_slow.log >/backup/new_$(date +%F)_slow.log
mail -s "$(date +%F)_slow.log" 623913455@qq.com </backup/new_$(date +%F)_slow.log
```

⑤注意日常优化
DBA，总监，开发，CTO联合解决

# 使用profile功能查询优化

profile博文

# MySQL备份常用方法

MySQL备份的常用方式有逻辑备份和物理备份（oracle也是如此）。

## 逻辑备份介绍

MySQL的逻辑备份其实就是使用MySQL自带的mysqldump命令或者其它工具把MySQL数据备份成SQL语句存储，在恢复时通过执行MySQL恢复命令（或source等）将存储的SQL语句还原到MySQL数据库中。
**补充：**增量备份binlog日志文件即可，如何增量恢复binlog日志？mysqlbinlog工具可以把binlog日志转换成SQL语句，然后通过mysql恢复命令（或source等）将SQL语句还原到mysql数据库。

常用工具为mysql自带的mysqldump命令：

```
1. mysqldump -uroot -p623913 -S /data/3306/mysql.sock -F -B oldboy|gzip
   >/backup/mysqlbak_$(date +%F).sql.gz
```

恢复增量的方法：

```
1. mysqlbinlog /data/3306/mysql-bin.000006 >bin.sql
2. mysql -uroot -p623913 -S /data/3306/mysql.sock oldboy<bin.sql
```

## 物理备份介绍

MySQL的物理备份方法之一是使用cp,rsync,tar,scp等复制工具把MySQL数据文件复制成多份，由于在备份及期间数据仍在写入数据库，所以，直接复制的方式备份会引起数据丢失。另外在恢复数据库时，对新数据库的路径、配置也有要求，一般要和原库的路径保持一致（版本、路径、配置尽可能一样）。
为了确保备份期间的数据一致性，可以选择人工停库或锁库后再进行物理复制数据，在生产环境中一般是不允许的，除非是可以申请停机或锁表时间。
基本可用性和数据一致性的选择问题。
一般在进行大规模数据库迁移，停库然后物理迁移，是最有效率的方案。
除了在Linux命令行通过命令直接复制MySQL数据文件之外，还有第三方开源或商业备份工具：xtrabackup。对于oracle数据来说，可以通过rman工具来物理全备及增量备份。

**物理备份：**
1、停库或锁表，然后打包拷贝。
2、第三方工具xtrabackup。

# MySQL数据库备份与恢复

运维工作简单的概括就两件事：**第一个是保护公司数据，第二个是网站7*24小时提供服务。**
`mysqldump --default-character-set=gbk` #指定字符集备份
mysqldump的–no-data或 -d 参数不导数据，只导表结构

**小结：**

1. 备份数据使用`-B`参数，会在备份数据中增加建库及use库的语句。
2. 备份数据使用`-B`参数，后面可以直接接多个库名。
3. 用`gzip`对备份的数据压缩。

4. debug时可以用`--compact`减少输出，但不用于生产。
5. 指定字符集备份用`--default-character-set=gbk`（一般不用）。

# mysqldump工作原理

利用`mysqldump`命令备份数据的过程，实际就是把数据从`mysql`库里以逻辑的`sql`语句的形式直接输出或者生成备份文件的过程。

# 备份库

`-B`参数是关键，表示接多个库，并且增加`use dbname`和`create database dbname`的信息(常用)。
-B参数说明：

```
-B, --databases
     Dump several databases. Note the difference in usage; in this case no tables are given. All name arguments are regarded as database names. 'USE db_name;' will be included in the output.

说明：该参数用于导出若干个数据库，在备份结果中会加入create database ‘dbname’和use dbname; -B后的参数都将被作为数据库名，该参数比较常用。当-B后的数据库列全时，同-A参数。
```

**分库备份**
法一：命令拼接备份多个库

```
$ mysql -uroot -p623913 -S /data/3306/mysql.sock -e "show databases;"|egrep -vi "Data|mysql|_schema"|sed -r 's#(^.*$)#mysqldump -uroot -p623913 -S /data/3306/mysql.sock --events -B -x -F \1|gzip >/backup/\1_$(date +%F).sql.gz#g'|bash
$ ls -lrt /backup/
-rw-r--r-- 1 root root    928 Jan  3 00:44 bbs_2016-01-03.sql.gz
-rw-r--r-- 1 root root    847 Jan  3 00:44 bbs_utf8_2016-01-03.sql.gz
-rw-r--r-- 1 root root   1647 Jan  3 00:44 oldboy_2016-01-03.sql.gz
```

法二：shell脚本备份多个库

```
$ vim mysqldump.sh
#!/bin/bash
USER=root
PASSWD=623913
SOCKET=/data/3306/mysql.sock
LOGIN="mysql -u$USER -p$PASSWD -S $SOCKET"
DUMPCMD="mysqldump -u$USER -p$PASSWD -S $SOCKET -x -B -F -R"
DATABASE=$($LOGIN -e "show databases;"|egrep -v "Data|mysql|_schema")
DIR=/backup
for dbname in $DATABASE
do
   [ ! -d $DIR ] && mkdir -p $DIR
   $DUMPCMD $dbname|gzip >$DIR/${dbname}_$(date +%F).sql.gz
done

$ sh mysqldump.sh
-rw-r--r-- 1 root root    930 Jan  3 00:57 bbs_2016-01-03.sql.gz
-rw-r--r-- 1 root root    849 Jan  3 00:57 bbs_utf8_2016-01-03.sql.gz
-rw-r--r-- 1 root root   1649 Jan  3 00:57 oldboy_2016-01-03.sql.gz
```

**分库备份的意义：**
有时一个企业的数据库里会有多个库，例如（www,bbs,blog），但是出问题时很可能是某一个库，如果在备份时把所有的库都备份成一个数据文件的话，恢复某一个库的数据时就比较麻烦。且有些库大，有些库小。

# 备份表

- 备份单个表：
  语法：mysqldump -u用户名 -p密码 数据库名 表名 >备份文件名
  例：`mysqldump -uroot -p623913 oldboy student >/backup/table1.sql`
  不能加-B参数了，因为oldboy库后面就是student表。
- 备份多个表：
  语法：mysqldump -u用户名 -p密码 数据库名 表名1 表名2 >备份文件名
  例：`mysqldump -uroot -p623913 oldboy student course >/backup/table2.sql`

**企业需求：**一个库里有大表、小表，有时可能需要只恢复某一个小表，上述的多表备份文件很难拆开，就会像没有分库那样导致某一个小表很麻烦。
那么如何进行分表备份？和分库的思想一样，每执行一条语句备份一个表，生产不同的数据文件，如下：

```
mysqldump -uroot -p623913 oldboy test >oldboy_test.sql
mysqldump -uroot -p623913 oldboy student >oldboy_student.sql
……….
```

上述命令放入一个脚本就是分表备份脚本了，当然很土。

**分表备份缺点：**文件多，很碎
1、备一个完整全备，再做一个分库分表备份。
2、脚本批量恢复多个SQL文件。

> **面试题：多个库或多个表备份到一块了，如何恢复单个库或表？**
> 解答：
> a. 第三方测试库，导入倒库里，然后把需要的备份出来，最后恢复到正式库里。
> b. 单表：grep 表名 bak.sql >表名.sql
> 单库：循环过滤库里所有表，grep 表名 bak.sql >表名.sql（多个表数据）
> c. 实现分库分表备份

- **分表备份脚本**

```
$ vim mysqldump.sh
#!/bin/bash
USER=root
PASSWD=623913
SOCKET=/data/3306/mysql.sock
LOGIN="mysql -u$USER -p$PASSWD -S $SOCKET"
DBDUMP="mysqldump -u$USER -p$PASSWD -S $SOCKET -x -B -F -R"
DATABASE=$($LOGIN -e "show databases;"|egrep -v "Data|mysql|_schema")
TBDUMP="mysqldump -u$USER -p$PASSWD -S $SOCKET"
DIR=/backup
for dbname in $DATABASE
do
   [ ! -d $DIR ] && mkdir -p $DIR
   for tbname in $($LOGIN -e "show tables from $dbname;"|sed '1d')
   do
     [ ! -d $DIR/$dbname ] && mkdir -p $DIR/$dbname
     $TBDUMP $dbname $tbname|gzip >$DIR/${dbname}/${tbname}_$(date
+%F).sql.gz
   done
   # $DBDUMP $dbname|gzip >$DIR/${dbname}_$(date +%F).sql.gz
done

$ sh mysqldump
$ tree /backup/
/backup/
├── bbs
│   ├── student_2016-01-03.sql.gz
│   └── test_2016-01-03.sql.gz
├── bbs_utf8
│   ├── student_2016-01-03.sql.gz
│   └── test_2016-01-03.sql.gz
└── oldboy
    ├── course_2016-01-03.sql.gz
    ├── sc_2016-01-03.sql.gz
    ├── student_2016-01-03.sql.gz
    ├── test_2016-01-03.sql.gz
    └── tl_2016-01-03.sql.gz

3 directories, 9 files
```

# 备份数据库表结构

利用`mysqldump -d`参数只备份表结构：

```
$ mysqldump -uroot -p623913 -S /data/3306/mysql.sock -B -d bbs >/backup/b.sql
$ egrep -v "#|\*|--|^$" /backup/b.sql
USE `bbs`;
DROP TABLE IF EXISTS `student`;
CREATE TABLE `student` (
  `id` int(4) NOT NULL AUTO_INCREMENT,
  `name` char(20) NOT NULL,
  `age` tinyint(2) NOT NULL DEFAULT '0',
  `dept` varchar(16) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `index_name_dept` (`name`(8),`dept`(10))
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8;
DROP TABLE IF EXISTS `test`;
CREATE TABLE `test` (
  `id` int(4) NOT NULL,
  `name` char(20) NOT NULL,
  `age` tinyint(2) NOT NULL DEFAULT '0',
  `dept` varchar(16) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

**如果只希望导出数据则用`-t`参数**

```
-t, --no-create-info
             Don't write table creation info.
```

`-T, --tab=name`语句和数据分离，数据为文本。

```
-T, --tab=name
           Create tab-separated textfile for each table to given path.
(Create .sql and .txt files.) NOTE: This only works if mysqldump is ru
n on the same machine as the mysqld server.
```

**小结：mysqldump的参数说明**

1. -B备份多个库（并添加create和use库的语句）#重点
2. -d只备份库表结构
3. -t只备份数据
4. -T分离库表和数据成不同的文件，数据是文本，非SQL语句。

# 刷新binlog

mysqldump用于定时对某一时刻的数据全备，例：0点进行备份bak.sql.gz。

增量备份：当有数据写入到数据库时，还会同时把更新的sql语句写入到对应的文件里，这个文件就叫做binlog文件。

**10点丢失数据需要恢复：**

1. 0点时刻备份的bak.sql.gz数据还原到数据库，这时数据恢复到0点；
2. 0点至10点数据，就要从binlog里恢复。

binlog文件生效需要在配置文件加上一个参数：log-bin

**binlog日志切割：**确定全备和增量的临界点
`-F` 刷新 `binlog` 日志，生成新文件，将来增量恢复从这个文件开始。

`--master-data` 在备份语句里添加 `change master` 语句及 `binlog` 文件及位置点信息；`--master-data` 不同值的作用：
值为1：可执行的 `change master` 语句；
值为2：注释的 `--change master` 语句。
`--master-data` 除了增量恢复确定临界点外，做主从复制时作用更大。
**注：**刷新 `binlog` 日志就是切割 `binlog` 日志。

# mysqldump关键参数

具体请 `mysqldump --help`

1. `-B` 指定多个库，增加建库语句和 `use` 语句
2. `--compact` 去掉注释，适合调试输出，生产不适用
3. `-A` 备份所有库
4. `-F` 刷新 `binlog` 日志，生产新文件，将来增量恢复从这个文件开始
5. `--master-data` 增加binlog日志文件名及对应的位置点（及 `CHANGE MASTER` 语句）；`--master-data=1` 不注释，`--master-data=2` 注释。
6. `-x, --lock-all-tables`
   Locks all tables across all databases. This is achieved by taking a global read lock for the duration of the whole dump. Automatically turns –single-transaction and –lock-tables off.
7. `-l, --lock-tables` `Lock all tables for read.`
8. `-d` 只备份库表结构，无数据
9. `-t` 只备份数据，无库表结构
10. `-T` 库表和数据分离不同文件，数据文本形式
11. `--single-transaction` 适合 `innodb` 事务数据库备份
    `innodb` 表在备份时，通常启用选项 `--single-transaction` 来保证备份的一致性，实际上他的工作原理是设定本次会话的隔离级别为：`REPEATABLE READ`，以确保本次会话（dump）时，不会看到其他会话已经提交了的数据。
12. `-q, --quick` Don’t buffer query, dump directly to stdout. (Defaults to on; use –skip-quick to disable.) -q,–quick

# 生产场景不同引擎mysqldump备份

**myisam引擎企业生产备份命令（适合所有引擎或混合引擎）**

```
$ mysqldump -uroot -p623913 -A -B -F -R --master-data=2 -x --events|gzip >/backup/all.sql.gz
提示：-F也可以不用，与--master-data有些重复。
```

**innodb引擎企业生产备份命令（推荐使用）**

```
$ mysqldump -uroot -p623913 -A -B -F -R --master-data=2 --events --single-transaction|gzip >/backup/all.sql.gz
提示：-F也可以不用，与--master-data有些重复。
```

**`--master-data`的作用**

1. 使用`--master-data=2`进行备份文件会增加如下内容：适合普通备份、增量备份
   `-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000020',MASTER_LOG_POS=1191;`
2. 使用`--master-data=1`进行备份文件会增加如下内容：更适合主从复制
   `CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000020',MASTER_LOG_POS=1191;`

> **数据库备份补充**
> ①mysqldump是逻辑备份
> 缺点：效率不是特别高
> 优点：简单、方便、可靠、迁移。
> 适用于数据量不是特别大的场景，50G以内的数据。
> ②超过50G数据可选方案：
> 1.xtrabackup物理备份工具：全备和增量
> 2.物理备份方案：从库停止IO线程，打包(tar)，cp。
> ③什么时候会使用备份的数据？
> 1.恢复数据到测试库时；
> 2.认为通过SQL语句将数据删除的时候；
> 3.做主从复制的时候。

# 恢复数据库

**利用source命令恢复数据库**

```
mysql -uroot -p623913 -S /data/3306/mysql.sock #登录数据库
mysql> use bbs;                 #切换库
mysql> source /backup/bbs.sql   #使用source命令接脚本文件路径，默认是登录mysql前的目录。
```

**针对压缩的备份数据恢复**

方法1：

```
gzip -d /backup/mysql_bak.sql.gz
mysql -uroot -p623913 </backup/mysql_bak.sql
不删除源备份文件：
gzip -cd 01.sql.gz >02.sql
```

方法2：

```
gunzip < b_bak.sql.gz >/backup/mysql_bak.sql
mysql -uroot -p623913 </backup/mysql_bak.sql
或：
gunzip < b_bak.sql.gz | mysql -uroot -p623913
```

# MySQL生产环境备份

## 全量备份

全量数据就是数据库中所有的数据，全量备份就是把数据库中所有的数据进行备份。
例：备份所有库

```
mysqldump -uroot -p623913 -S /data/3306/mysql.sock -F -B -A|gzip >/bac
kup/mysqlbak_$(date +%F).sql.gz
```

备份一个库：

```
mysqldump -uroot -p623913 -S /data/3306/mysql.sock -F -B oldboy|gzip
>/backup/mysqlbak_$(date +%F).sql.gz
```

## 增量备份

增量数据是从上次全量备份之后更新的数据，对MySQL来说，binlog日志就是MySQL的增量数据。

**1、按天全备情况**

| 周一 0 点全备 | 周二 0 点全备 | |
|---|---|---|
| 01.sql.gz | 02.sql.gz | |
| 周一的增量备份 | 周二的增量数据 | |
| mysql-bin.000024<br>mysql-bin.000024<br>mysql-bin.000024<br>……<br>mysql-bin.index | mysql-bin.000037<br>mysql-bin.000038<br>mysql-bin.000039<br>……<br>mysql-bin.index | |

优点：恢复时间短，维护成本低。
缺点：占用空间多，占用系统资源多，经常锁表影响用户体验。

**2、按周全备情况**

| | | |
|---|---|---|
| 周六 0 点全量备份 | | |
| 01.sql.gz | | |
| 周一增量备份 | 周二增量数据 | 周三增量数据 |
| mysql-bin.000024<br>mysql-bin.000024<br>mysql-bin.000024<br>……..<br>mysql-bin.index | mysql-bin.000037<br>mysql-bin.000038<br>mysql-bin.000039<br>……<br>mysql-bin.index | mysql-bin.000040<br>mysql-bin.000041<br>mysql-bin.000042<br>……<br>mysql-bin.index |

优点：占用空间小，占用 系统资源少，无需锁表或锁表次数少，用户体验好一些。
缺点：维护成本高，恢复麻烦，时间长。

# 企业场景全量和增量的频率是怎么做的？

**企业备份方案**
**①按天备份–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binlog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：** 恢复时间短；维护成本低。
**缺点：** 占用空间大；占用资源多；经常影响到用户体验。

**②按周备份–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。

**关于备份频率:**

1. 中小公司，全量一般是每天一次，业务流量低谷执行全部，备份时会锁表。
   增量备：
   a. 定时推binlog增量，例如每分钟推一次增量。
   rsync -avz /data/3306/mysql-bin.000* rsync_backup@10.0.0.41::backup –password-file=/etc/rsync.password
   b. 再在其它远程实时读binlog，（read from remote server）。
2. 大公司周备，每周六00点一次全量，周日至下周六00点前都是增量。
   优点：节省备份时间，减少备份压力。
   缺点：增量的binlog文件副本太多，还原很麻烦。
3. 一主多从环境，主从复制本身就是实时远程备份，可以解决服务器物理故障。
4. 一主多从环境，可采取一个从库服务器上专门用mysqldump，cp，tar，xtrabackup做备份，延迟同步

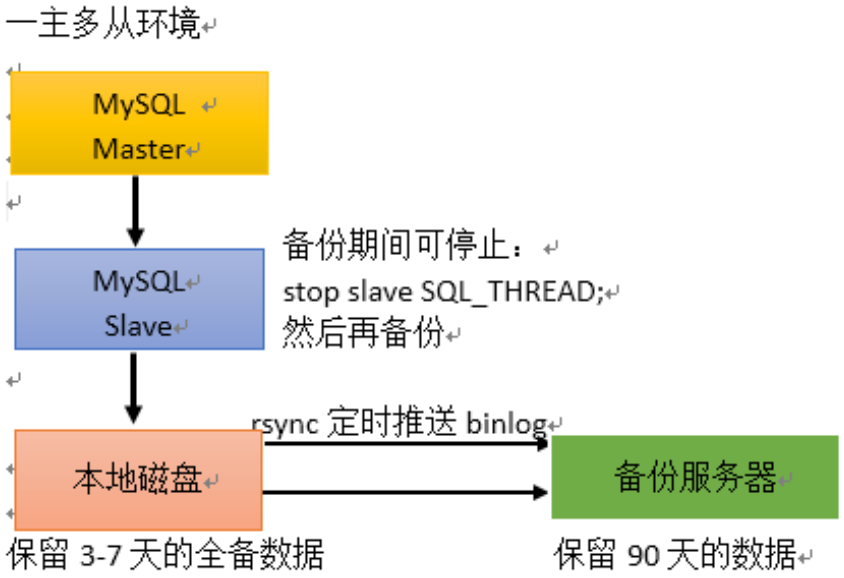
# MySQL的mysqldump备份什么时候派上用场？

1. 迁移或者升级数据库时。
2. 增加从库的时候。

3. 因为硬件或特殊异常情况，主库或从库宕机，主从可以互相切换，无需备份。
4. 人为的DDL，DML语句，主从库没办法了，所有库都会执行。此时需要备份。
5. 跨机房灾备，需要备份到异地。

单台数据库做增量：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，尽量做主从复制。

# MySQL主从环境备份




**说明：**一主多从架构，从库上停SQL线程做全备到本地，然后通过同步工具发送到远端。本地保留全备3-7日数据；备份服务器保留90天的数据以及周六的数据不删。

> **小结：**
> 1.一般由人为（或程序）逻辑的方式在数据库执行的SQL语句等误操作，才需要增量恢复，因为此时，所有从库也执行了误操作语句。
> 2.物理故障，则直接切换到从库。

# MySQL增量备份必备条件

MySQL增量备份必备条件：开启binlog日志功能，log-bin参数如下

```
$ grep log-bin /data/3306/my.cnf
log-bin = /data/3306/mysql-bin
```

**注意：** 主库和备份的从库都要开启binlog记录功能。
**小结：** 增量恢复的条件是存在一份全备加上全备之后，到出问题的时间段的所有增量binlog文件。

# 增量恢复总结

1. 条件：全备份以及全备之后的binlog日志。
2. 关键点：G点，确定全备份之后时刻binlog文件及位置。

```
$ mysqldump -F
$ ls -lrt mysql-bin.log >dump.log
$ cat master.info
mysql> show mater status;
```

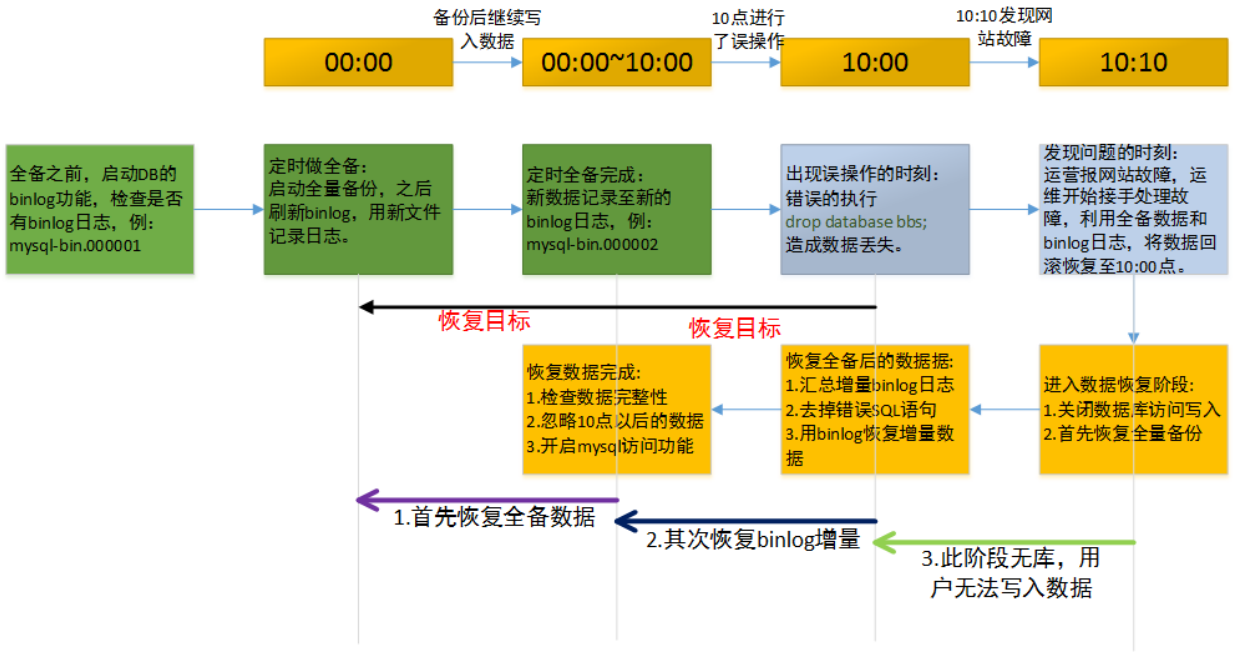
3. 恢复时候尽量对外停止服务。
   满足条件的数据堆在一起，写好恢复语句，进行恢复。
4. 补充：备份恢复时可以将 `sql_log_bin` 关闭，恢复完再开启。
5. 未雨绸缪，多事前下功夫，而不是事后补救。

**恢复时如果不能停止对外服务，则**

1. 停止一个从库，然后在主库刷新 `binlog` ，把 `mysql-bin.000014` 恢复成 `bin.sql` ，然后去掉错误的 `drop` 语句。
2. 把全备 `bak_2014-11-06.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。但在 `mysql-bin.000015` 中。
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，所以**
>
> 1. 停止一个从库，然后再主库刷新 `binlog` ，把 `mysql-bin.000014` 恢复成 `bin.sql` ，然后去掉错误的 `drop` 语句
> 2. 把全备 `bak_2014-11-06.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BINLOG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

# MySQL备份恢复流程




**过程**

1. 备份数据库全备，并且获得所有 `binlog` 。
2. 0点： `mysql_backup_2015-02-13.sql.gz` 。
3. 0点备份后——10点出问题前：用户会正常写入数据。
4. 10点出问题的时刻： `drop database oldboy;`
5. 10:10监控和运营人员发现问题，报告给运维。
6. 检查，发现确实网站不正常，通过日志或者页面提示大概找到问题在哪，同时问下有数据库权限的人员是否做过什么操作。结果有人回答：10点左右，执行了一个 `drop` 。
7. 恢复阶段：保留现场，备份全备及所有 `binlog` ，防止数据被二次破坏。
8. 精确的找到全备份时刻之后的所有日志， `mysqlbinlog -d oldboy mysql-bin*` 转成 `bin.sql` ，然后删除掉出问题的SQL语句。
9. 恢复全备 `mysql <mysql_backup_2015-02-13.sql.gz`
   恢复增量 `mysql <bin.sql`
   测试检查恢复结果。
10. 对外提供服务，完毕。

# 重点问题结论

①数据库里如果有多个库。

```
1.  -d,--database=name
2.              List entries for just this database(locallog only)
```

从binlog文件里，过滤出指定库的binlog语句。

```
1.  mysqlbinlog -d oldboy mysql-bin.000014 >oldboy_bin.sql
```

②如果是某个表被删，恢复单表，把 `binlog` 语句分出指定表的思路：

1. 把原来指定`oldboy`库导出表结构，恢复到测试库，然后把`oldboy_bin.sql`语句恢复到测试库，然后再用`mysqldump`导出需要的单个表，恢复到已经恢复了全备的正式库上。
2. `oldboy_bin.sql`最小按库分，那么可以`grep “表名”`把`oldboy_bin.sql`所有对于该表的记录过滤出来。恢复到已经恢复了全备的正式库。

③如果是重要的库出问题，那么最好停库或禁止库被应用服务器写入，然后再恢复（iptables出来）。如果通过`host`解析的，注释解析文件记录，用户中心（接口停掉）。

④多个`binlog`文件要按顺序恢复
方法一：

```
1. mysqlbinlog -d oldboy 01 02 03 04 >bin.sql
2. 或者for循环读取，顺序很重要。
```

方法二：

```
1. mysqlbinlog -d oldboy >bin.sql后
2. grep 要恢复的表 bin.sql >a.sql
```

# MySQL命令小结

```
1. mysql> show processlist;          #查看数据库里正在执行的SQL语句，看不全
2. mysql> show full processlist;     #查看正在执行的完整SQL语句，完整显示
3. mysql> set global key_buffer_size = 1024*1024*32;    #不重启数据库调整数据库参数，直接生效，重启失效
4. mysql> show variables;            #查看数据库的参数信息，例：my.cnf里参数的生效情况
5. mysql> show variables like ‘%log_bin%’;
6. mysql> kill ID;                   #杀掉线程的命令，ID为线程号
7. mysql> show status;               #查看当前会话的数据库状态信息
8. mysql> show global status;        #查看整个数据库运行状态信息，很重要，要分析并要做好监控
9. mysql> show engine innnodb status;  #显示innodb引擎的性能状态（早起版本show innnodb status;）
```

> 计算一天之内数据库有多少个insert、delete，有没有好方法？
>
> 1. 定时任务每天0点，`show global status;`按天取出对比。
> 2. 按天分析binlog日志，获取数据库不同语句的频率。
>
> mysqldump命令

```
mysqladmin password 123456              #设置密码
mysqladmin -uroot -p123456 passowrd 654321  #修改密码
mysqladmin -uroot -p123456 -S /data/3306/mysql.sock status
mysqladmin -uroot -p123456 -S /data/3306/mysql.sock -i 1 status
mysqladmin -uroot -p123456 flush-logs
mysqladmin -uroot -p123456 processlist
mysqladmin -uroot -p123456 extended-status
mysqladmin -uroot -p123456 processlist -i 1       #实时跟踪
mysqladmin -uroot -p123456 shuwdown;      #关闭mysql服务进程
mysqladmin -uroot -p123456 variables;     #等于show variables;
watch mysqladmin -uroot -p123456 -S /data/3306/mysql.sock processlist
# 每两秒显示

注意：所有多实例数据库都要指定socket
```

**命令使用小技巧**

> 例子：mysql线程中”大海捞针“
> 平时登录数据库show processlist; 发现结果经常超长，找自己所要的信息比较困难，而且SQL显示不全。如果直接执行show full processlist那更是瞬间滚了N屏。找到有问题的SQL语句非常困难。

**解决之法：**

```
mysql -uroot -p623913 -S /data/3306/mysql.sock -e “show full processlist;”|grep -v sleep
```

过滤当前执行的SQL语句完整内容，这条命令很有用。后面还可以加iconv等中文转码，根据需求过滤响应的内容，此命令屡试不爽。

# mysqlbinlog命令

`mysqlbinlog`工具的作用是解析`mysql`的二进制`binlog`日志内容，把二进制日志解析为可以在`mysql`数据库里执行的`SQL`语句。

`mysql`的`binlog`日志作用是用来记录`mysql`内部增删改等对`mysql`数据库有更新的内容的记录（对数据库的改动），对数据查询的语句如：`show，select`开头的语句，不会被binlog日志记录。
**作用：** 数据库的主从复制，及数据的增量恢复。

**必须开启log-bin功能才能生成binlog日志**

```
$ grep log-bin /data/3306/my.cnf
log-bin = /data/3306/mysql-bin
```

在MySQL数据库中，关于`binlog`日志：依靠足够长度的`binlog`日志和定期的全备，我们可以恢复任何时间点的单表数据。

**利用mysqlbinlog命令的-d参数解析指定库的binlog日志**

```
$ mysqlbinlog -d oldboy /data/3306/mysql-bin.000003
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=1*/;
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYP
E=0*/;
DELIMITER /*!*/;
# at 4
#151217 10:53:20 server id 1  end_log_pos 107   Start: binlog v 4, ser
ver v 5.5.32-log created 151217 10:53:20 at startup
# Warning: this binlog is either in use or was not closed properly.
```

指定位置截取：精确

```
$ mysqlbinlog /data/3306/mysql-bin.000006 --start-position=365 --stop-
position=456 -r pos.sql

指定开始位置，不指定结束位置，那结束位置是？
$ mysqlbinlog /data/3306/mysql-bin.000006 --start-position=365 -r po
s.sql
指定结束位置，不指定开始位置，请问开始位置是？
$ mysqlbinlog /data/3306/mysql-bin.000006 --stop-position=465 -r pos.s
ql
```

按时间截取：模糊、不准

```
$ mysqlbinlog /data/3306/mysql-bin.000006 --start-datetime='2015-10-14
16:23:12' --stop-datetime='2015-12-14 16:23:12' -r time.sql

指定开始时间，不指定结束时间，请问结束时间是？
$ mysqlbinlog /data/3306/mysql-bin.000006 --start-datetime='2015-10-14
16:23:12' -r time.sql
指定结束时间，不指定开始时间，开始时间是？
$ mysqlbinlog /data/3306/mysql-bin.000006 --stop-datetime='2015-12-14
16:23:12' -r time.sql
```

**mysqlbinlog命令小结**

1. 把binlog日志解析为sql语句（包含位置和时间点）
2. `-d`参数根据指定库拆分binlog（拆分单表binlog可通过SQL关键字过滤）
3. 通过位置参数截取部分binlog：`--start-position=365 --stop- position=465`，精确定位取部分内容。
4. 通过时间参数截取部分binlog：`--start-datetime='2015-10-14 16:23:12' --stop-`

`datetime='2015-12-14 16:23:12'`，模糊取部分内容，会丢数据。

5. `-r`文件名，相当于重定向`>filename`。
6. 解析ROW级别binlog日志的方法。
   `mysqlbinlog --base64-output=decode-rows-v mysql-bin.000016`
   `mysqlbinlog --base64-output="decode-rows" --verbose mysql-bin.000016`

# mysql sleep线程过多的企业问题案例

首先得找出sleep线程

```
mysql> show processlist;  #查看线程
```

解决办法：

```
mysql> show variables like '%_timeout%';
mysql> set global wait_timeout = 60;
mysql> set global interactive_timeout = 60;
```

或者在配置文件里修改：

```
[mysqld]
interactive_timeout = 120  #此参数设置后wait_timeout自动生效
wait_timeout = 120
```

其它方法：

1. PHP程序中，不是要持久链接，即使用mysql_connect而不是pconnect（JAVA调整连接池）。
2. PHP程序执行完毕，应该显示调用mysql_close。
3. 逐步分析mysql的SQl查询及慢查询日志，找到查询过慢的SQL，将其优化。

# 不重启数据库修改数据参数的方法

```
第一步：
mysql> set global key_buffer_size = 1024*1024*32;    #即时生效，重启失效

第二步：配置文件也要改
$ vim /etc/my.cnf
[mysqld]
key_buffer_size = 32M
```

**此参数说明：** `key_buffer_size`是对`MyISAM`表性能影响最大的一个参数，它指定索引缓冲区的大小，决定索引处理的速度，尤其是索引读的速度。

`mysqlreport`：可详细的展示MySQL参数的配置情况。

# 字符集

字符集是一套符号和编码。校对规则是在字符集内用于比较字符的一套规则。

## 数据库字符乱码

**乱码的原因**

1. 环境本身不对，插入了错误数据，很难解决
2. 环境对的时候插入了正确数据，客户端环境破坏了，查看时乱码了。

**数据库不乱码的前提**
（ 中文推荐utf8，不乱码的重要保障 ）

1. Linux客户端字符集（ UTF8 ）
   CRT、xshell等
2. Linux服务端字符集（ UTF-8 ）

```
$ cat /etc/sysconfig/i18n
LANG="en_US.UTF-8"
SYSFONT="latarcyrheb-sun16"
$ echo $LANG
en_US.UTF-8
```

3. 数据库 客户端字符集（ UTF8 ）
   法一：set names 字符集 # 临时生效
   法二：

```
set character_set_client = gbk;
set character_set_results = gbk;
set character_set_connection = gbk;

以上相当于：set names gbk;
```

法三：`mysql -uroot -p623913 -S /data/3306/mysql.sock --default-character-set=gbk`
法四：在配置文件/etc/my.cnf里下添加：

```
[client]
default-character-set=gbk
```

4. 数据库 服务端字符集（UTF8）
   法一：在my.cnf文件中添加以下字段之后重启mysql进程

```
[mysqld]
character-set-server=gbk    #适合5.5及以上
default-character-set=gbk   #适合5.1及以前
```

以上改变的是：

```
character_set_database   | utf8
character_set_server     | utf8
```

法二：数据库编译的时候指定字符集

```
-DDEFAULT_CHARSET=utf8
-DDEFAULT_COLLATION=utf8_general_ci
-DEXTRA_CHARSETS=gbk,gb2312,utf8,ascii
```

5. 数据库具体库的字符集
   指定字符集建库：
   `create database oldboy_utf8 default character set utf8 collate utf8_general_ci;`
6. 表的字符集
   指定字符集建表：

```
create table `student` (
  `id` int(4) not null auto_increment,
  `name` char(20) not null,
  primary key (`id`)
) engine=InnoDB auto_increment=10 default charset=utf8
```

7. PHP、IAVA程序字符集

# 在数据库中执行sql语句的方法

1. 操作习惯：尽量不在MySQL命令行直接插入数据（SSH客户端影响），SQL语句文件形式
2. SQL文件的格式统一：“UTF-8没有签名”
3. 导入文件形式方式：sql文件里`set names utf8`，然后source执行sql文件
   a. 可在mysql命令行中用source执行sql文件

b. 命令方式导入数据 `mysql -uroot -p623913 oldboy<test.sql`

c. `mysql -uroot -p623913 oldboy --default-character-set=utf8<test.sql`

# MySQL如何选择合适的字符集

1. 如果处理各种各样的文字，发布到不同语言国家地区，应选Unicode字符集，对mysql来说就是utf8（每个汉字三字节），如果应用处理英文，仅有少量汉字UTF8更好。
2. 如只需支持中文，并且数据量很大，性能要求也很高，可选GBK（定长，每个汉字占双字节，英文也占双字节），若需大量运算，比较排序等，定长字符集，更快，性能高。
3. 处理移动互联网业务，可能需要使用utf8mb4字符集。

没有特别需求，请选择utf8，很多开源程序都会有多种字符集版本。

不同字符集参数的含义：

```
mysql> show variables like "characte%";
+--------------------------+---------
| Variable_name            | Value
+--------------------------+---------
| character_set_client     | utf8    #客户端字符集
| character_set_connection | utf8    #连接字符集
| character_set_database   | utf8    #数据库字符集，配置文件指定或建库建表指定
| character_set_filesystem | binary  #文件系统字符集
| character_set_results    | utf8    #客户端返回结果字符集
| character_set_server     | utf8    #服务器字符集，配置文件指定或建库建表指定
| character_set_system     | utf8
| character_sets_dir       | /application/mysql-5.5.32/share/charsets/|
+--------------------------+--------
8 rows in set (0.00 sec)
```

# 如何更改生产MySQL数据库库表的字符集

**数据字符集修改步骤：**

对于已有的数据库想修改字符集不能直接通过 `alter database character set *` 或 `alter table tablename character set *`，这两个命令都没有更新已有记录的字符集，而只是对新创建的表或记录生效。

已有记录的字符集的调整，必须先将数据导出，经过修改字符集后重新导入后才可完成。

**修改数据库的库默认编码：**

`alter database dbname charset [character setting]`

下面方法只能适应新数据，库里的老数据字符集还是没改。

1. 更改oldboy库的字符集

`alter database oldboy character set gbk collate = gbk_chinese_ci;`

2. 更改test表的字符集
   `alter table test character set gbk;`

**模拟将latin1字符集的数据库修改成utf8字符集的实际过程：**

1. 导出表结构
   `mysqldump -uroot -p623913 --default-character-set =latin1 -d dbname >alltable.sql`
   `--default-character-set =utf8`表示以utf8字符集进行连接，-d只导出表结构
2. 编辑表结构语句，在alltable.sql将所有latin1字符串改成utf8。
3. 确保数据库不再更新，导出所有数据（不带表结构）
   `mysqldump -uroot -p623913 --quick --no-create-info --extended-insert --default-character-dbname >alldata.sql`

   > 参数说明：
   > –quick：用于转储大的表，强制mysqldump从服务器一次一行的检索数据而不是行，并输出前cache到内存中。
   > –no-create-info：不创建create table语句。
   > –extended-insert：使用包括几个values列表的多行insert语法，这样文件也小，导入数据时会非常快。
   > –default-character-set =latin1：按照原有字符集导出数据，这样导出的文件中，所有数据是可见的，不会保存成乱码。

4. 修改my.cnf配置调整客户端及服务端字符集，重启生效。
5. 通过utf8建库（可选）
   删除原库，然后建库`create database dbname default charset utf8;`
6. 导入表结构（更改过字符集的表结构）
   `mysql -uroot -p623913 dbname <alltable.sql` #单实例
   `mysql -uroot -p623913 -S /data/3306/mysql.sock dbname <alltable.sql` #多实例
7. 导入数据
   `mysql -uroot -p623913 dbname <alldata.sql` #单实例
   `mysql -uroot -p623913 -S /data/3306/mysql.sock dbname <alldata.sql` #多实例
   **注意：**选择目标字符集时，最好大于等于源字符集（字库更大），否则可能产生不被支持的数据。

**生产环境修改方法：**

1. mysqldump备份（数据量大先导出表结构，sed替换），再导出数据备份。
2. 在表结构中把字符集改了，例：sed替换。
3. 修改系统配置调整字符集生效。
4. mysql把表结构还原，再把数据还原。
5. 开发人员把程序的字符集调整好。

**字符集修改问题**
对新数据生效，老数据不行：
`alter table test character set = gbk collate gbk_chinese_ci;`
`alter database oldboy character set latin1 collate = latin1_awedish_ci;`

====更改字符集总的思想=====

1. 数据库不要更新，导出所有数据
2. 把导出的数据进行字符集替换（替换表和库）
3. 修改my.cnf，更改MySQL客户端、服务端字符集，重启生效
4. 导入更改过字符集的数据，包括表结构语句，提供服务。
5. ssh客户端，以及程序更改为对应字符集。

# MySQL数据库服务常用日志文件知识

## 误日志（error log）介绍与调整

①错误日志（error log）介绍
MySQL的错误日志记录MySQL服务进程mysqld在启动、关闭或运行过程中遇到的错误信息。
②错误日志实践
法一：在配置文件中调整方法，当然可以在启动时加入启动参数

```
[mysql_safe]
log-error=/data/3306/mysql_oldboy3306.err
```

法二：在启动命令加入`--log-error=/data/3306/mysql_oldboy.err`

```
mysql_safe --defaults-file=/data/3306/my.cnf --log-error=/data/3306/my
sql_oldboy.err
```

**错误排查思路**
1.先把日志文件备份并清空启动一下mysql服务然后再查看日志文件报有什么错误。

```
$ cat mysql_oldboy3306.err
```

2.然后mysql 3306目录下面所有文件都加上属主并是递归-R。

```
$ chown -R mysql *
```

3.然后查看一下有没有这个mysql ID。

```
$ id mysql
uid=666(mysql) gid=666(mysql) groups=666(mysql)
```

4.重启mysql服务

```
$ . /mysql start
```

5.再查看服务有没有启动

```
$ lsof -i :3306
```

# 普通查询日志（general query log）介绍与调整

①普通查询日志（general query log）介绍
普通查询日志记录客户端连接信息和执行的SQL语句信息。
②普通查询日志（general query log）调整

```
mysql> show variables like 'general_log%';
+------------------+--------------------------+
| Variable_name    | Value                    |
+------------------+--------------------------+
| general_log      | OFF  #默认不开启,一般不开启 |
| general_log_file | /data/3306/data/db01.log |
+------------------+--------------------------+
2 rows in set (0.00 sec)

mysql> set global general_log = on;   #开启普通日志，临时生效
Query OK, 0 rows affected (0.13 sec)

mysql> show variables like 'general_log%';
+------------------+--------------------------+
| Variable_name    | Value                    |
+------------------+--------------------------+
| general_log      | ON                       |
| general_log_file | /data/3306/data/db01.log |
+------------------+--------------------------+
2 rows in set (0.00 sec)
```

永久生效：在my.cnf中加入如下

```
general_log = on
general_log_file = /data/3306/data/MySQL_oldboy.log
```

注意：高并发场景企业里普通日志是不开启的（默认关闭），主要因为IO性能问题。特别需要时，才临时开启。

# 慢查询日志（show query log）介绍与调整

①慢查询日志（show query log）介绍
慢查询日志记录执行时间超出指定值（long_query_timeout）
②慢查询日志调整
配置参数记录慢查询语句：

```
vim my.cnf----------------------------------------
long_query_time = 1                       #超过2秒，记录到log里
log-slow-queries = /data/3306/slow.log #没有走索引的语句，记录到log里
log_queries_not_using_indexes           #log文件
```

慢查询的设置，对于数据库SQL的优化非常重要

**利用慢查询优化的一套解决方案**
1.1.开启慢查询，在my.cnf中配置

```
long_query_time = 1                       #超过2秒，记录到log里
log-slow-queries = /data/3306/slow.log  #没有走索引的语句，记录到log里
log_queries_not_using_indexes           #log文件
```

1.2.慢查询日志切割脚本

```
$ vim cut_slow_log.sh
#!/bin/sh
cd /data/3306/ && \
/bin/mv slow.log slow.log.$(date +%F)&& \
mysqladmin -uroot -p623913 -S /data/3306/mysql.sock flush-log
[root@db01 scripts]# echo "00 00 * * * /bin/sh /server/scripts/cut_slo
w_log.sh &>/dev/null" >>/var/spool/cron/root
```

2.explain索引分析
3.对需要建索引的条件列建立索引。
生产场景，大表不能高峰期建立索引，例：300万记录情景下。
4.使用`mysqlsla`工具分析慢查询，切割MySQL慢查询日志，去重分析后发给相关人员信箱

# 二进制日志（binary log）介绍与调整

①二进制日志（binary log）：记录数据被修改的相关信息
②二进制日志（binary log）调整

```
mysql> show variables like '%log_bin%';
+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| log_bin                         | ON    | #记录Binlog
| log_bin_trust_function_creators | OFF   |
| sql_log_bin                     | ON    | #临时不记录Binlog（增量恢复）
+---------------------------------+-------+
3 rows in set (0.00 sec)
```

```
$ grep log-bin /data/3306/my.cnf
log-bin = /data/3306/mysql-bin
```

### log-bin作用

1. 记录更改的二进制形式SQL语句
2. 主从复制
3. 数据增量备份

### 临时不记录Binlog（增量恢复）

```
mysql> set session sql_log_bin = OFF;   #不要启用
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like '%log_bin%';
+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| log_bin                         | ON    |
| log_bin_trust_function_creators | OFF   |
| sql_log_bin                     | OFF   |
+---------------------------------+-------+
3 rows in set (0.01 sec)

mysql> create database abcd;
Query OK, 1 row affected (0.00 sec)
```

```
$ mysqlbinlog /data/3306/mysql-bin.000007|grep abcd
                    # 什么都没有
```

# MySQL binlog的三种模式及设置方法

## Row Level

日志中会记录成每一行数据被修改的形式，然后在slave端再对相同的数据进行修改。

- **优点**
  在row level模式下，binlog中可以不记录执行的sql语句的上下文相关的信息，仅仅只需要记录那一条记录被修改了，修改成什么样了。所以row level的日志内容会非常清除的记录下每一行数据修改的细节，非常容易理解。而且不会出现某些特定情况下的存储过程、function，及trigger的调用和触发无法被正确复制的问题。
  **优点总括：**记录每一条数据的执行细节；最大程度上让主从保持一致。
- **缺点**
  row level下，所有执行的语句当记录到日志中的时候，都将以每行记录的修改来记录，这样可能会产生大量的日志内容，如：有这样一条update语句：update product set owner_member_id = 'b' where owner_member_id = 'a'，执行之后，日志中记录的不是这条update语句所对应的事件（MySQL以事件的形式来记录bin-log日志），而是这条语句所更新的每一条记录的变化情况，这样就记录成很多条记录被更新的很多个事件。自然，bin-log日志的量就会很大。尤其是当执行alter table之类的语句的时候，产生的日志量是惊人的。因为MySQL对于alter table之类表结构变更语句的处理方式使整个表的每一条记录都需要变动，实际就是重建了整个表，改表的每一条记录都会被记录到日志中。

# Statement Level（默认）

每一条被修改数据的SQL都会记录到master的bin-log中，slave在复制的时候SQL进程会解析成和原来master端执行过的相同的SQL来再次执行。

- **优点**
  statement level下的优点首先就是解决了row level下的缺点，不需要记录每一行数据的变化，减少bin-log日志量，节约IO，提高性能。因为他只需要记录在Master上所有执行语句的细节，及执行语句时候的上下文的信息。
- **缺点**
  由于它是记录的执行语句，所以，为了让这些语句在slave端也能正确执行，那么它还必须记录每条语句在执行时的一些相关信息，也就是上下文信息，以保证所有语句在slave端被执行时能够得到和在Master端执行时相同的结果。另外就是，由于MySQL现在发展比较快，很多的新功能不断的加入，使MySQL的复制遇到了不小的挑战，自然复制的时候涉及到越复杂的内容，bug也就越容易出现。在statement level下，目前一经发现的就有不少情况会造成MySQL的复制出现问题，如：sleep() 函数在有些版本中就不能正确复制，在存储过程中使用了last_insert_id() 函数，可能回事slave和master上得到不一致的id等等，由于row level是基于每一行来记录的变化，所以不会出现类似的问题。

# Mixed

实际就是前两种模式的结合。在Mixed模式下，MySQL会根据执行的每一条具体的SQL语句来区分对待记录的日志形式，也就是statement和row之间选择一种。新版中的statement level还是和以前一样，仅仅记录执行的语句。而新版MySQL对row level模式也被做了优化，并不是所有的修改都会以row level来记录，像遇到表结构变更的时候就会以statement模式来记录，如果SQL语句确实就是update或delete等修改数据的语句，那么还是会记录所有行的变更。

# 企业场景如何选择binlog的模式

1. 互联网公司，使用MySQL的功能相对较少（存储过程、触发器、函数）
   选择默认的语句模式：Statement Level（默认）
2. 公司如果用到使用MySQL的特殊功能（存储过程、触发器、函数）
   则选择Mixed模式
3. 公司如果用到使用MySQL的特殊功能（存储过程、触发器、函数），又希望数据最大化一致，此时最好Row Level模式

# 修改bin-log模式的例子

①设置bin-log模式为row

```
mysql> set global binlog_format = ‘row’;
```

再在配置文件中参数如下

```
log-bin=mysql-bin
binlog_format=”ROW”
```

②重启数据库查看修改效果

```
mysql> show global variables like “binlog_format%”;
```

③批量插入及更新数据

```
mysql> update test set name=”test”;
mysql> select * from test;
```

④检验ROW模式下binlog日志记录效果

```
$ mysqlbinlog --base64-output=decode-rows -v /data/3306/mysql-bin.000007
或：
$ mysqlbinlog --base64-output="decode-rows" --verbose /data/3306/mysql-bin.000007
```

# 删除二进制日志(binlog)

随着时间往后二进制日志会变的多、大，因此，有必要执行删除二进制日志，一般会在配置文件里加入以下参数实现自动清理工作：
①RESETMASTER;
作用：删除所有binlog日志，新日志编号从头开始。

```
mysql> RESETMASTER;

mysql> system ls -l /data3306/mysql-bin*
```

②PURGE MASTER LOGS TO‘mysql-bin.000005’;
作用：删除mysql-bin.000005之前所有日志，不包括000005自身。

```
mysql> PURGE MASTER LOGS TO‘mysql-bin.000005’;
```

③PURGE MASTER LOGS BEFORE ‘2015-02-01 24:35:21’ ;
作用：删除2015-02-01 24:35:21之前产生的所有日志

④删除二进制日志

```
$ grep expire /data/3306/my.cnf
expire_logs_days = 7
```

此参数含义：使MySQL自动删除7天的二进制日志，因此，使用`mysqldump`备份的频度，应该在7天之内进行一次全备，这样数据会完整。如果数据量很大，也可以一周做一次全备，但是，对于中心公司来说，这不是一个号的备份方案，因为，增量日志越多，意味着恢复时更麻烦、耗时。

**完全备份+增量备份=完整备份**

# 企业Linux运维场景数据同步方案

文件级别同步方案：

1. 普通同步：scp/sftp/ftp/samba/rsync/csync2/union
2. 实时同步：nfs/inotify/sersync/lsyncd
3. 文件系统同步：drbd

第三方同步软件：实时的（实时的不一定是同步的，可能异步的）
DB自带的同步功能：mysql replication，oracle dataguard
redis主从复制，mongodb

运维思想实现：程序双写

## 普通文件（磁盘上的文件）的同步方法

一、文件级别的以及同步方案：

1. scp、sftp、nc命令可以实现远程数据同步。
2. 搭建FTP、http、SVN、nfs服务器，然后在客户端上也可以把数据同步到服务器。
3. 搭建Samba文件共享服务，然后在客户端上也可以把数据同步到服务器
   参考文档
4. 利用rsync、csync2、union等均可以实现数据同步
   提示：union可实现双向同步，虽然csync2可实现多机同步
   以上文件同步方式如果结合定时任务或者inotify、sersync等功能，可以实现定时以及实时的数据同步。
5. 扩展思想：文件级别也可以利用mysql，mongodb等软件作为容器实现。
6. 扩展思想：程序向两个服务器同时写数据，双鞋就是一个同步机制。
   特点：简单、方便、效率和文件系统级别要差一些，但是被同步的节点可以提供访问。
   软件的自身同步机制（mysql，Oracle，mongdb，ttserver，redis…），文件放到数据库，同步到从库，再到文件拿出来。
7. DRBD

# 文件系统级别的异机同步方案

1. drbd基于文件系统同步，相当于网络RAID1，可以同步几乎任何业务数据。
   mysql数据库的官方推荐drbd同步数据，所有单点服务例如：nfs、mfs（drbd），mysql等都可以用drbd做复制，效率很高；缺点：备机服务不可用。
2. 数据库同步方案
   a、自身同步机制：mysql replication，mysql主从复制（逻辑的SQL重写）物理复制方法——>drbd（从库不提供读写）。
   Oracle dataguard（物理的磁盘块，逻辑的SQL语句重写），9i从库不提供读写的，11g的从库实现了read-only。
   b、第三方DRBD，参考文档

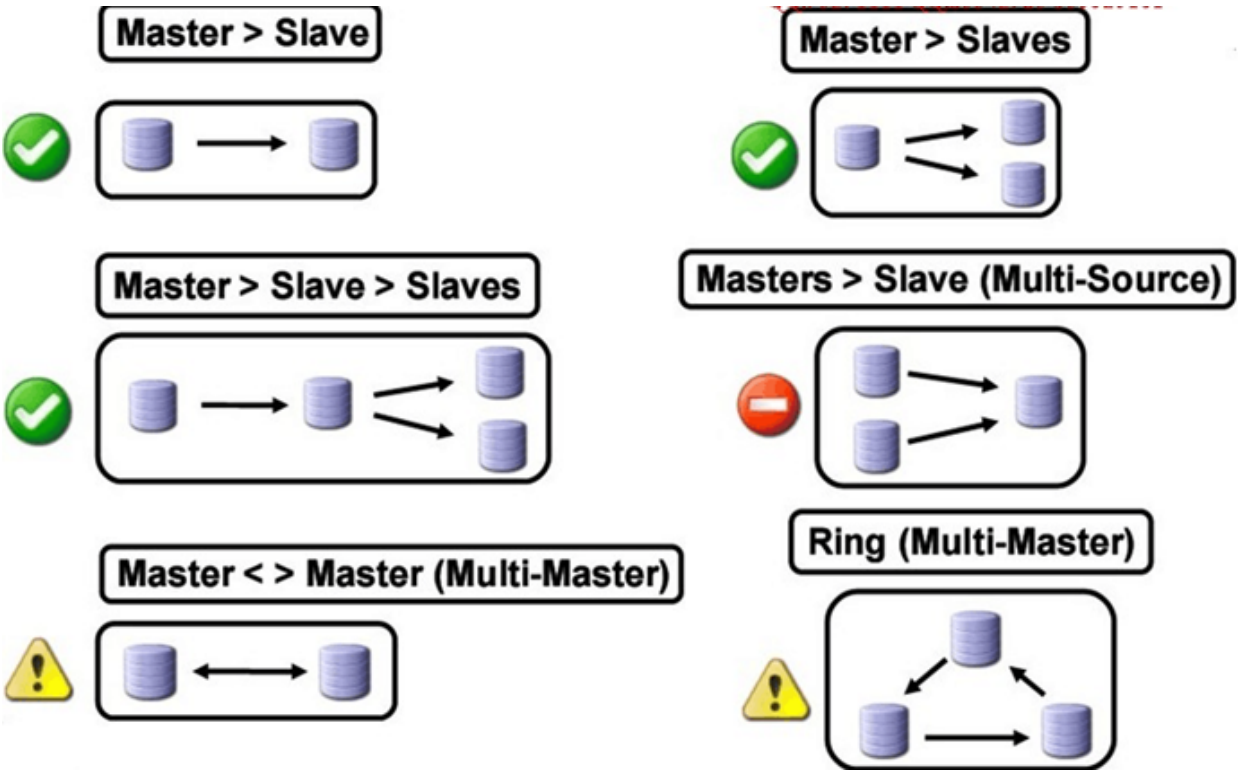
# MySQL主从复制

MySQL的主从复制方案，和scp、rsync等文件级别同步是类似的，都是数据的传输。只不过MySQL无需借助第三方工具，而是其自带的复制功能，有一点不同，MySQL的主从复制并不是数据库磁盘上的文件直接拷贝复制，而是通过逻辑的Binlog日志复制到同步的数据本地然后读取里面的SQL语句应用到数据库的过程。

MySQL数据库支持单向、双向、链式级联等不同场景的复制。在复制过程中，一台服务器充当主服务器（Master），而一个或多个其它的服务器充当从服务器（Slave）。

复制可以是单向：M—>S，也可以是双向M<—>M，当然也可以多M环状同步等。

如果设置了链式级联复制，那么，从（Slave）服务器本身除了充当从服务器外，也会同时充当其下面从服务器的主服务器。链式级联复制类似A–>B–>C–>D的复制形式。

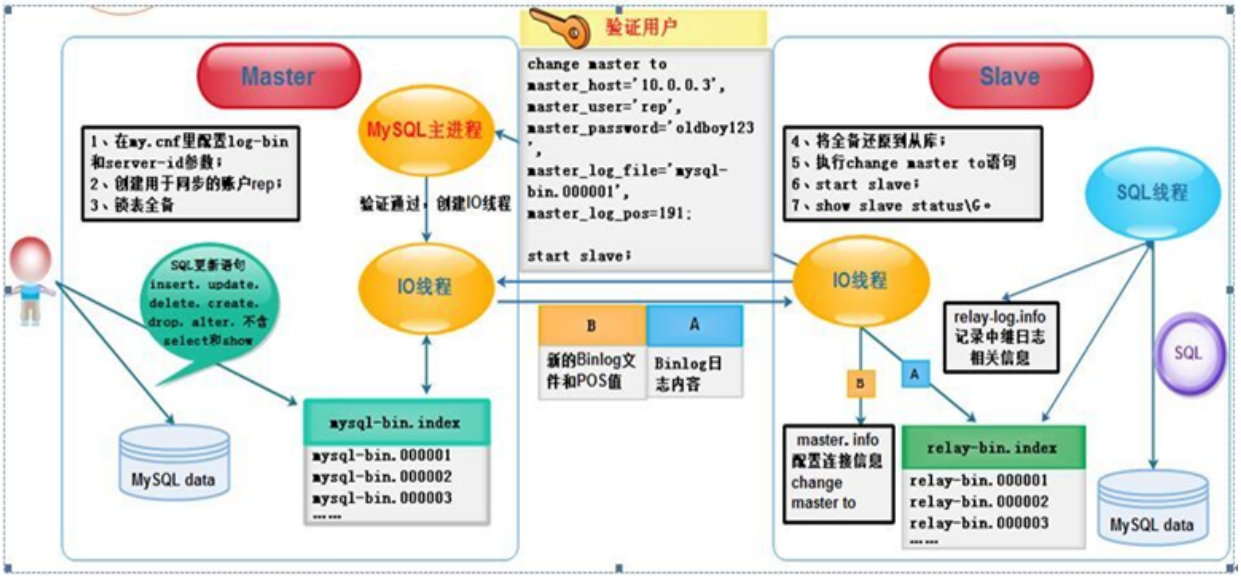



# MySQL主从复制原理

MySQL的主从复制是一个异步的复制过程（虽然一般情况下感觉是实时同步的），数据库数据从一个MySQL数据库（称之为Master）复制到另一台MySQL数据库（称之为Slave）。在master和slave之间实现整个主从复制的过程是由三个线程参与完成的。期中有两个线程（SQL线程和IO线程）在slave端，另外一个线程（IO线程）在Master端。

要实现MySQL的主从复制，首先须打开Master端的Binlog（mysql-bin.xxx）功能，否则就无法实现主从复制。因为整个复制过程实际上就是slave从Master端获取Binlog日志，然后再在slave自身上以相同顺序执行获取的Binlog日志中所记录的各种操作。

MySQL replication的复制原理逻辑图：



**MySQL主从复制原理过程详细描述**

1. Slave服务器上执行start slave，开启主从复制开关。
2. 此时，slave服务器的IO线程会通过在Master上授权的复制用户权限请求连接Master服务器，并请求从指定Binlog日志文件的指定位置（日志文件名和位置就是在配置主从复制服务时执行change master命令时指定的）之后发送Binlog日志内容。
3. Master服务器接收到来自Slave服务器的IO线程的请求后，Master服务器上负责复制的IO线程根据slave服务器的IO线程请求的信息读取指定Binlog日志文件指定位置之后的Binlog日志信息，然后返回给Slave端的IO线程。返回的信息中除了Binlog日志内容外，还有本次返回日志内容后在Master服务器端的新的Binlog文件名称及在Binlog中的下一个指定更新位置。
4. 当slave服务器的IO线程获取到来自Master服务器IO线程发送日志内容及日志文件及位置点后，将Binlog日志内容一次写入到slave端自身的Relay Log（即中继日志）文件（MySQL-relay-bin.xxx）的最末端，并将新的Binlog文件名和位置记录到master-info文件中，以便下一次读取Master端新Binlog日志时能够告诉Master服务器需要从新Binlog日志的哪个文件哪个位置开始请求新的Binlog日志内容。
5. slave服务器端的SQL线程会实时的检测本地Relay Log中新增加的日志内容，然后及时的把Log文件中的内容解析成在Master端曾经执行的SQL语句的内容，并在自身slave服务器上按语句的顺序执行应用这些SQL语句，应用完毕后清理应用过的日志。
6. 经过上面的过程，可确保在master端和slave端执行了同样的SQL语句。当复制状态正常的情况下，master端和slave端数据是完全一样的，MySQL的同步机制是有一些特殊情况的，多数情况下，不用担心。

**主从复制条件**

1. 开启 `binlog` 功能
2. 主库建立账户
3. 从库配置 `master.info（change master）`
4. `start slave` 复制开关

**知识点**

1. 3个线程，主库IO，从库IO和SQL，及作用
2. master.info作用
3. relay-log作用
4. 异步复制
5. binlog作用

**小结**

1. 主从复制是异步的、逻辑的SQL语句级的复制。PXC
2. 同步时，主库有一个IO线程，从库有两个线程，IO和SQL线程。
3. 实现主从复制的必要条件，主库要开启Binlog功能。
4. Binlog文件只记录数据库有更改的SQL语句（来自主数据库内容的变更）。

# 主从复制实战

①检查是否启用多实例和binlog功能，多实例 `server-id` 不能一样

```
$ [root@db02 ~]# ss -lntup|grep 330
tcp LISTEN  0  600    *:3306      *:*    users:(("mysqld",2089,12))
tcp LISTEN  0  600    *:3307      *:*    users:(("mysqld",2806,11))
tcp LISTEN  0  600    *:3308      *:*    users:(("mysqld",3547,12))

$ egrep "server-id|log-bin" /data/{3306,3307,3308}/my.cnf
/data/3306/my.cnf:log-bin = /data/3306/mysql-bin
/data/3306/my.cnf:server-id = 1
/data/3307/my.cnf:#log-bin = /data/3307/mysql-bin
/data/3307/my.cnf:server-id = 3
/data/3308/my.cnf:log-bin = /data/3308/mysql-bin
/data/3308/my.cnf:server-id = 8
```

②登录主库，建立用于从库复制账户及授权

```
$ mysql -uroot -poldboy123 -S /data/3306/mysql.sock
mysql> grant replication slave on *.* to rep@'172.16.1.%' identified b
y 'oldboy';
Query OK, 0 rows affected (0.01 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

mysql> select user,host from mysql.user;
+------+------------+
| user | host       |
+------+------------+
| root | 127.0.0.1  |
| rep  | 172.16.1.% |
| root | localhost  |
+------+------------+
3 rows in set (0.00 sec)
```

③锁表，锁表期间不可退出mysql，只能新建SSH连接

```
mysql> flush table with read lock;
Query OK, 0 rows affected (0.00 sec)

mysql> show master status;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000008 |    20152 |              |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

④主库全量备份

```
$ mysqldump -uroot -poldboy123 -S /data/3306/mysql.sock --events -A -B|gzip >/backup/rep_bak_$(date +%F).sql.gz
在此条命令中加上 -x 参数亦可锁表
```

⑤解表

```
mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)
```

⑥将主库备份数据导入从库

```
$ gzip -d /backup/rep_bak_2015-12-25.sql.gz
$ cd /backup/
$ mysql -uroot -poldboy456 -S /data/3307/mysql.sock <rep_bak_2015-12-25.sql
```

⑦登录从库配置主从复制参数并启动主从复制

```
$ mysql -uroot -poldboy456 -S /data/3307/mysql.sock

mysql> CHANGE MASTER TO
    -> MASTER_HOST='172.16.1.52',          # 主库IP
    -> MASTER_PORT=3306,                   # 主库端口
    -> MASTER_USER='rep',                  # 主库上建立的用于复制的账户rep
    -> MASTER_PASSWORD='oldboy',           # rep的密码
    -> MASTER_LOG_FILE='mysql-bin.000008',# 是show master status;时查看到的二进制日志文件名称，不能多空格
    -> MASTER_LOG_POS=20152;               # show master status时看到的二进制日志偏移量，不能多空格
Query OK, 0 rows affected (0.13 sec)
提示：如果在备份时加入了"--master-data=1"，则在恢复时就会直接执行带Binlog文件及位置点的CHANGE MASTER语句，则此处的CHANGE MASTER执行不需要指定的Binlog文件及位置点了。

mysql> start slave;                        # 开启主从复制
```

**附：无需锁表备份的方案**

在生产环境时，一般会每天备份一份完整数据，在备份时锁表备份并且记录下备份时的`Binlog`对应的文件及位置点，这样在实现主从复制时，就无需先锁表备份了，直接用夜里的全备即完成主从同步的配置。另外，若没有全备，那最好等到夜里进行全备，也可能需要申请停机时间。因为锁表期间，会影响业务。小规模就不用了直接夜里`mysqldump`锁表备份即可，当然务必记录`Binlog`的位置或者增加`--master-data=1`。

**无需锁表的mysqldump备份命令**

myisam引擎企业生产备份命令

```
mysqldump -uroot -p623913 -A -B -F --master-data=2 -x --events|gzip >/opt/all.sql.gz
```

innodb引擎企业生产备份命令：推荐使用

```
mysqldump -uroot -p623913 -A -B -F --master-data=2 --events --single-transaction|gzip >/opt/all.sql.gz
```

`--master-data`参数的作用：

使用–master-data=2进行备份文件会增加如下内容：适合普通备份增量恢复

```
-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000005', MASTER_LOG_POS=107;
```

使用–master-data=1进行备份文件会增加如下内容：更适合主从复制

```
CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000005', MASTER_LOG_POS=107;
```

**问题小结**

1. mysql5.5授权时用`GRANT REPLICATION CLIENT ON *.* TO 'rep'@'10.0.0.%' IDENTIFIED BY PASSWORD '123456';`在主从复制的时候状态会一直显示Connecting ，所以请选择`GRANT REPLICATION SLAVE ON *.* TO 'rep'@'10.0.0.%' IDENTIFIED BY PASSWORD '123456';`
   错误显示：

```
    Slave_IO_Running: Connecting
   Slave_SQL_Running: Yes
       Last_IO_Error: Master command COM_REGISTER_SLAVE failed: Access denied for user 'rep'@'172.16.1.%' (using password: YES) (Errno: 1045)
```

2. 如果`change master`里面的东西设置错了，修改`master info`不生效~请执行`reset slave all;`
3. `Last_IO_Error: error connecting to master 'rep@172.16.1.52:3306' - retry-time: 60 retries: 86400`
   密码错误，密码为当时grant授权时的密码，不是登录密码

# MySQL主从同步配置步骤

1. 两台数据库环境，或单台多实例环境，能都正常启动、登录。
2. 配置`my.cnf`文件，主库配置`log-bin`和`server-id`参数，从库配置`server-id`，不能喝主库及其它从库相同，一般不开启从库log-bin功能。注意：配置参数后要重启生效。
3. 登录主库增加用于从库连接主库同步的账户，例：rep，并授权`replication slave`同步的权限。
4. 登录主库。整库锁表`flush table with read lock;`（窗口关闭后即失效，超时参数到了也失效）；然后`show master status;`查看`Binlog`的位置状态。
5. 新开窗口，Linux命令行备份或导出原有的数据库数据，并拷贝到从库所在的服务器目录。如果数据量很大，并且允许停机，可以停机打包，而不用mysqldump。
6. 解锁主库：`unlock tables;`
7. 把主库导出的原有数据恢复到从库。
8. 根据主库的`show master status;`查看到的`Binlog`的位置状态，在从库执行`change master to …`语句。
9. 从库开启同步开关：`start slave;`
10. 从库`show slave status\G;`检查同步状态，并在主库进行更新测试。

**补充**以下两者的区别

```
flush tables with read lock
flush table with read lock
```

> **做主从同步小技巧**
> 写脚本定时半夜锁表备份，白天再做主从复制。无需熬夜守着，快速部署MySQL主从复制。

**小结**
**主库上**

1. 开`binlog`功能
2. 确保所有实例`server-id`不同
3. 授权复制的用户rep
4. 锁表，查看`binlog`文件及位置点`(--master-data=2)`。
5. 新开窗口导出全备
6. 解锁，开放用户写入功能

**从库上**

1. 确保所有实例`server-id`不同
2. 把主库的全备导入到从库。
3. 找位置点，配置`master.info`，登录MySQL执行以下操作

```
CHANGE MASTER TO
MASTER_HOST='172.16.1.52',
MASTER_PORT=3306,
MASTER_USER='rep',
MASTER_PASSWORD='oldboy',
MASTER_LOG_FILE='mysql-bin.000008',
MASTER_LOG_POS=20152;
```

4. 开启主从复制`start slave;`查看状态`show slave status\G`
   测试。

# 主从复制的线程状态

```
$ mysql -uroot -poldboy456 -S /data/3307/mysql.sock -e "show slave sta
tus\G"|egrep -i "running|Behind"
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
        Seconds_Behind_Master: 0
```

1. `Slave_IO_Running: Yes`这个是IO线程状态，`IO`线程负责从从库去主库读取binlog日志，并写入从库的中继日志`relay-log`中，状态为Yes表示IO线程工作正常。
2. `Slave_SQL_Running: Yes`这个是SQL线程状态，SQL线程负责读取中继日志`relay-log`中的数据并转换为SQL语句应用到从数据库中，状态为Yes表示IO线程工作正常。
3. `Seconds_Behind_Master: 0`这个是从库和主库比，复制延迟的秒数，这个参数很重要，但企业里更精确的判断主从延迟的方法为：在主库写时间戳，然后从库读取时间戳和当前

数据库时间比较，来认定是否延迟。

提示：有关**show slave status;** 结果的说明，请看mysql手册。

# 主库IO线程工作状态

| 主库 IO 线程工作状态 | 解释说明 |
| --- | --- |
| Sending binlog event to slave | 线程已经从二进制 binlog 日志读取了一个时间并且正将它发送到从服务器。 |
| Finished reading one binlog; switching to next binlog | 线程已经读完二进制 binlog 日志文件，并且正打开下一个要发送到从服务器到的 binlog 日志文件。 |
| Has sent all binlog to slave; waiting for binlog to be updated | 线程已经从 binlog 日志读取所有更新并已经发送到了从数据库服务器。线程现在为空闲状态，等待由主服务器上二进制 binlog 日志中的新事件更新。 |
| Waiting to finalize termination | 线程停止时发生的一个很简单的状态 |

# 主从复制故障集

1. 主库`show master status;`没有返回状态结果

```
1. mysql> show master status;
2. Empty set (0.00 sec)
```

原因：主库`binlog`功能没开或没生效。

2. 服务无法启动故障

```
1. $ /data/3306/mysql start
2. MySQL is running...
3. $ ps -ef|grep mysql
4. root     1636   1595  0  11:47  pts/0    00:00:00   grep --color=auto m
   ysql
```

原因：启动脚本里做了对`mysql.sock`是否存在做了判断，如果存在则输出`MySQL is running...`，是个小bug，可自行修改启动脚本

```
1. [root@db02 ~]# vim /data/3306/mysql
2.     if [ ! -e "$mysql_sock" ];then
3.       printf "Starting MySQL...\n"
```

注意：可将`! -e "$mysql_sock"`改为：`netstat -lntup|grep ${port}|wc -l` -ne 1，会更好些。

# MySQL从库停止复制的故障解决

模拟错误：模拟重现故障的能力是运维人员最重要的能力。
先在从库创建一个库，然后去主库创建同名的库来模拟数据冲突。

```
mysql> create database blog;         #从库上建库
Query OK, 1 row affected (0.01 sec)

mysql> create database blog;         #主库上建库
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    6
Current database: *** NONE ***

Query OK, 1 row affected (0.00 sec)

mysql> show slave status\G           #从库上查看，发现报错
              Slave_IO_Running: Yes
             Slave_SQL_Running: No
         Seconds_Behind_Master: NULL
                   Last_Errno: 1007
                   Last_Error: Error 'Can't create database 'blog'; da
tabase exists' on query. Default database: 'blog'. Query: 'create data
base blog'
```

解决方法一

```
mysql> stop slave;          #临时停止同步开关
Query OK, 0 rows affected (0.07 sec)

mysql> set global sql_slave_skip_counter = 1; #将同步指针向下移动一个，如果
多次不同步，可以重复操作。
Query OK, 0 rows affected (0.00 sec)

mysql> start slave;          #开启同步开关
Query OK, 0 rows affected (0.00 sec)

mysql> show slave status\G
              Slave_IO_Running: Yes
             Slave_SQL_Running: Yes
```

解决方法二：提前在配置文件中加上以下参数，跳过某些不影响业务的错误

```
$ grep slave-skip /data/3308/my.cnf
slave-skip-errors = 1032,1062
1007：数据库已存在，创建数据库失败 《===============可以忽略
1008：数据库不存在，删除数据库失败 《===============可以忽略
1032：记录不存在    《============================可以忽略
1062：字段值重复，入库失败   《====================可以忽略
```

其它可能引起复制故障的问题：

1. mysql自身的原因及人为重复插入数据
2. 不同的数据库版本会引起不同步，低版本到高版本可以，但是高版本不能往低版本同步。
3. mysql的错误
4. binlog记录模式，例如：row level模式就比默认的语句模式要好

# 双主模式

## 让MySQL从库记录binlog日志方法

从库需要记录binlog的应用场景为：
当前的从库还要作为其它从库的主库，例如：级联复制、双主互为主从。

从库记录binlog日志方法：在从库my.cnf中加入以下参数，重启后生效

```
log-slave-updates      #必须要有的参数
log-bin = /data/3307/mysql-bin
expire_logs_days = 7  #相当于find /data/3307/ -type f -name “mysql-bin.000*” -mtime +7|xargs rm -f
```

## 双主实战

**双主实现方案**
a. 让表的ID自增，让主1写 `1,3,5……` ；主2写 `2,4,6……` 。
b. 不让表的ID自增，然后通过WEB端程序去seq服务器取ID，写入双主。
双主工作场景：写高并发场景，慎用

Master1

```
auto_increment_increment        = 2   #自增ID的间隔，如1,3,5….间隔为2
auto_increment_offset           = 1   #ID的初始位置
log-slave-updates
将形成1,3,5,7…..序列
```

Master2

```
auto_increment_increment        = 2  #自增ID的间隔，如2,4,6….间隔为2
auto_increment_offset           = 2  #ID的初始位置
将形成2,4,6,8…..序列
```

Master1的配置文件

```
$ cp /data/3306/my.cnf /data/3306/my.cnf.bak
$ vim /data/3306/my.cnf
在[mysqld_safe]下增加:
#========M-M M1 start=============
auto_increment_increment        = 2
auto_increment_offset           = 1
log-slave-updates
log-bin = /data/3306/mysql-bin
expire_logs_days = 7
#=======M-M M1 end================
并删除: log-bin = /data/3306/mysql-bin和expire_logs_days = 7

$ /data/3306/mysql restart
Restarting MySQL...
Stoping MySQL...
Starting MySQL...
```

Master2的配置文件

```
$ cp /data/3307/my.cnf /data/3307/my.cnf.bak
$ vim /data/3307/my.cnf
在[mysqld_safe]下增加:
#========M-M M2 start=============
auto_increment_increment        = 2
auto_increment_offset           = 2
log-slave-updates
log-bin = /data/3307/mysql-bin
expire_logs_days = 7
#=======M-M M2 end================
并删除: expire_logs_days = 7

$ /data/3307/mysql restart
Restarting MySQL...
Stoping MySQL...
Starting MySQL...
```

配置完毕：使用以下命令检查

```
mysql> show variables like "log_%";
mysql> show variables like "auto%";
```

从库数据导入主库

```
$ mysqldump -uroot -poldboy456 -S /data/3307/mysql.sock -A -B -x --events --master-data=1|gzip >/backup/rep3307_$(date +%F).sql.gz
$ ll /backup/rep3307_2015-12-26.sql.gz
-rw-r--r-- 1 root root 145410 Dec 26 16:49 /backup/rep3307_2015-12-26.sql.gz
$ gzip -d /backup/rep3307_2015-12-26.sql.gz
$ mysql -uroot -poldboy123 -S /data/3306/mysql.sock </backup/rep3307_2015-12-26.sql
$ mysql -uroot -poldboy123 -S /data/3306/mysql.sock
```

```
mysql> CHANGE MASTER TO
    -> MASTER_HOST='172.16.1.52',
    -> MASTER_PORT=3307,
    -> MASTER_USER='rep',
    -> MASTER_PASSWORD='oldboy';
Query OK, 0 rows affected (0.14 sec)

mysql> start slave;
Query OK, 0 rows affected (0.00 sec)

mysql> show slave status\G
```

配置双主互备完毕，检查

```
mysql> create database zhu1;          #Master1上创建库zhu1
Query OK, 1 row affected (0.00 sec)

mysql> show databases;                #Master2上检查
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| oldboy             |
| performance_schema |
| www                |
| zhu1               |
+--------------------+
6 rows in set (0.00 sec)

mysql> create database zhu2;        #Master2上创建库zhu2
Query OK, 1 row affected (0.00 sec)

mysql> show databases;              #Master1上检查
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| oldboy             |
| performance_schema |
| www                |
| zhu1               |
| zhu2               |
+--------------------+
7 rows in set (0.00 sec)
```

Master1上建表，插入数据

```
mysql> use oldboy;
Database changed
mysql> create table `tl` (
    -> `id` bigint(12) not null auto_increment,
    -> `name` varchar(12) not null,
    -> primary key (`id`)
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql> desc tl;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | bigint(12)  | NO   | PRI | NULL    | auto_increment |
| name  | varchar(12) | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> insert into tl(name) values('oldboy'),('oldgirl');
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from tl;
+----+---------+
| id | name    |
+----+---------+
|  1 | oldboy  |
|  3 | oldgirl |
+----+---------+
2 rows in set (0.00 sec)

mysql> insert into tl(name) values('littleboy');
Query OK, 1 row affected (0.01 sec)

mysql> select * from tl;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | oldboy    |
|  3 | oldgirl   |
|  5 | littleboy |
+----+-----------+
3 rows in set (0.00 sec)
```

Master2上插入数据

```
mysql> use oldboy;
Database changed
mysql> select * from tl;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | oldboy    |
|  3 | oldgirl   |
|  5 | littleboy |
+----+-----------+
3 rows in set (0.00 sec)

mysql> insert into tl(name) values('oldboy');
Query OK, 1 row affected (0.00 sec)

mysql> select * from tl;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | oldboy    |
|  3 | oldgirl   |
|  5 | littleboy |
|  6 | oldboy    |
+----+-----------+
4 rows in set (0.00 sec)

mysql> insert into tl(name) values('lzyi');
Query OK, 1 row affected (0.00 sec)

mysql> select * from tl;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | oldboy    |
|  3 | oldgirl   |
|  5 | littleboy |
|  6 | oldboy    |
|  8 | lzyi      |
+----+-----------+
5 rows in set (0.00 sec)
```

# MySQL主从复制企业应用场景

MySQL主从复制集群功能使得MySQL数据库支持大规模高并发读写成为可能，同时有效的保护物理服务器宕机场景的数据库备份。

# 应用场景①从服务器作为主服务器的实时数据备份

主从服务器架构的设置，可以大大的加强数据库架构的健壮性。例如：当主服务器出现问题时，我们可以人工或自动切换到从服务器继续提供服务，此时从服务器的数据和宕机时的主数据库几乎是一致的。

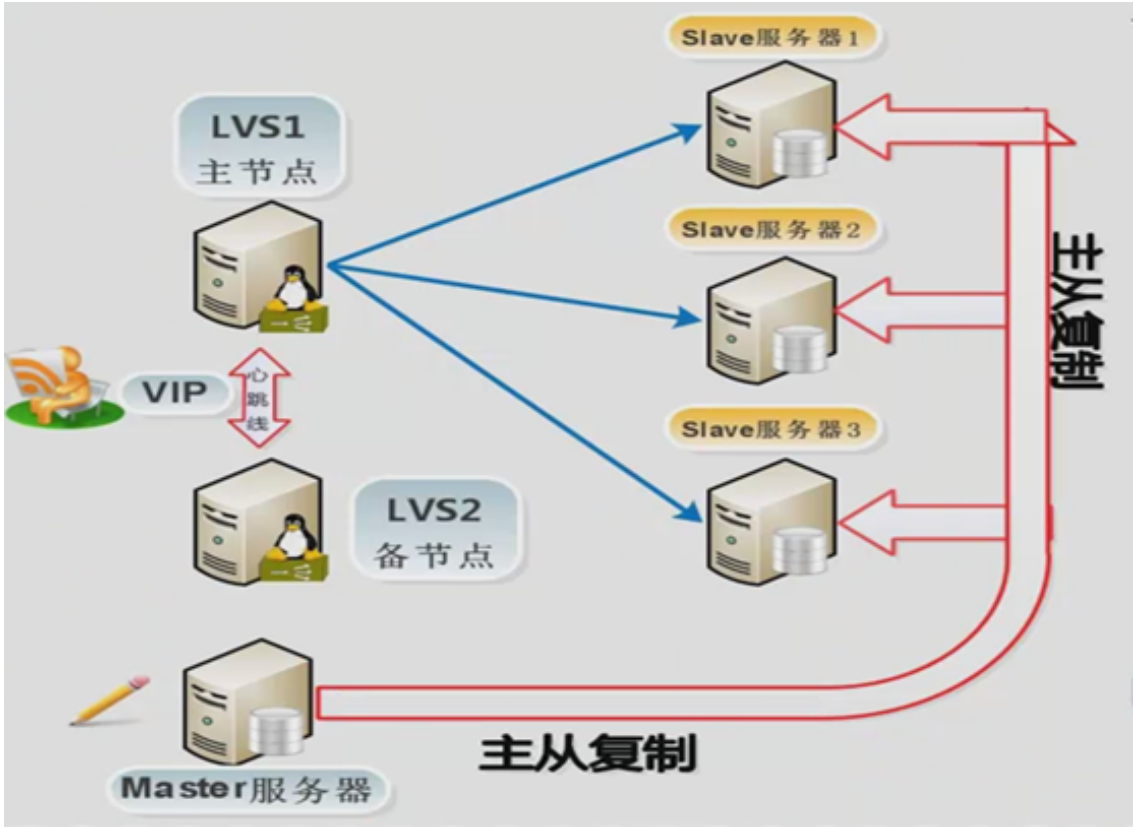
这类似我们前面课程中的NFS存储数据通过inotify+rsync同步到备份的NFS服务器非常类似，只不过MySQL的复制方案，是其自带的工具。

利用MySQL自带的复制功能做数据备份，在硬件宕机，服务故障的场景数据备份是有效的，但对于人为的执行drop、elete等语句删除数据的情况，从库的备份功能就没用了，因为从服务器也会执行删除的语句。

# 应用场景②主从服务器实现读写分离，从服务器实现负载均衡

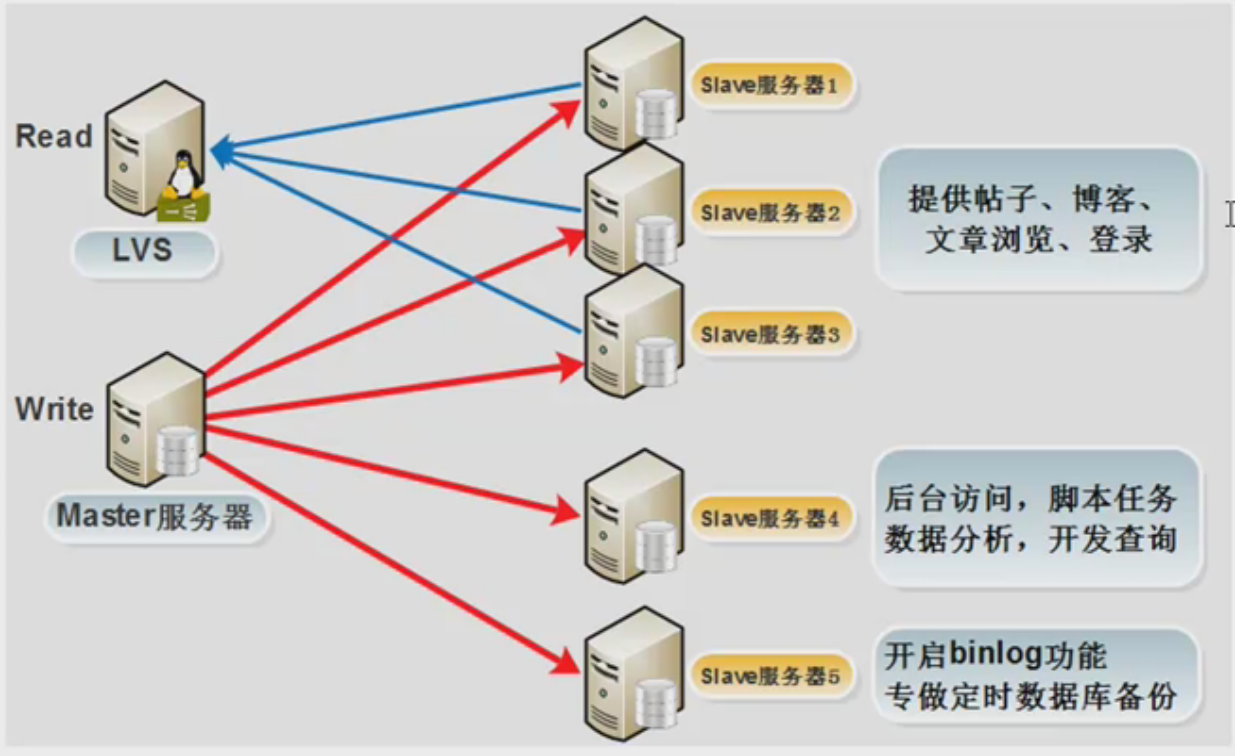
主从服务器架构可通过程序（php,java）或代理软件（mysql-proxy，amoeba）实现对用户（客户端）的请求读写分离，即让从服务器仅仅处理用户的select查询请求，降低用户查询响应时间及读写同时在主服务器带来的压力。对于更新的数据（update，insert，delete）仍然交给主服务器处理，确保主服务器和从服务器保持实时同步。

百度、淘宝、新浪等绝大多数的网站都是用户浏览页面多于用户发布内容，因此通过在从服务器只接收读请求，就可以很好的减轻主库的读压力，且从服务器可以很容易的扩展到多台，且可以加上LVS做负载均衡，效果就非常棒了，这就是传说中的数据库读写分离架构。上述架构的逻辑图如下



## 应用场景③把多个从服务器根据业务重要性进行拆分访问

可以把几个不同的从服务器，根据公司的业务进行拆分。例如：有为外部用户提供查询服务的从服务器，有内部DBA用来做数据备份的从服务器，还有为公司内部人员提供访问的后台、脚本，日志分析及开发人员使用的从服务器。这样的拆分除了减轻主服务器的压力外。是的数据库对外部用户浏览、对内部用户业务处理，及DBA人员备份等互不影响。具体看一下的简单架构

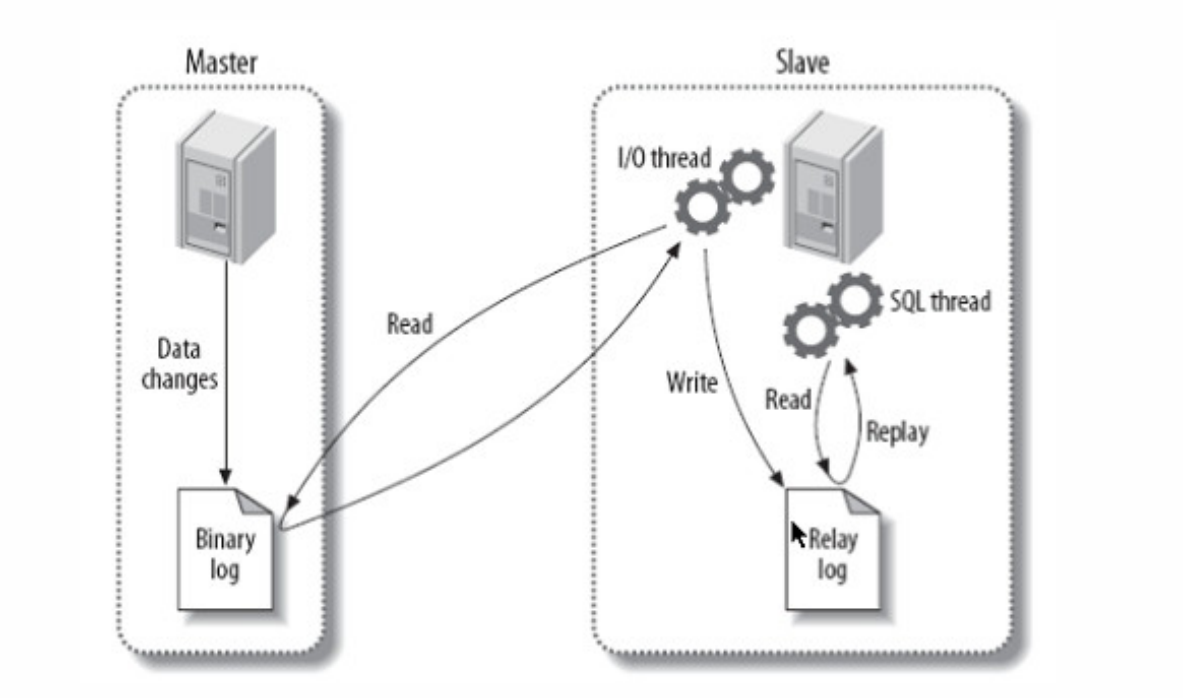



# MySQL主从半同步

## MySQL半同步复制研究背景

半同步主要针对一主多从的生产环境为产生的。MySQL主从复制默认采用的是异步模式，异步模式，从库每次从主库读取binlog日志放到自己的relaylog里，但主库不会关心从库读取多少binlog这样就就不会影响主库的性能，但是当主库宕机，但是从库binglog读取差异很大从而导致数据差异很大。实时同步这样会影响主库的性能。半同步是一个中间产品，他的机制是采用的是实时同步加异步。主库等待从库响应回答，如果超时主库就才用异步同步，当规定时间响应采用同步。系统默认是10s ，mysql5.5 本身自带半同步插件。

## 半同步逻辑图




# 半同步复制实战

基础环境：
多实例3306为主，3308为从，已建立主从复制。
一. 安装mysql半同插件，mysql5.5 已经自带这个插件
查看半同步插件的位置

```
mysql> show variables like "plugin_dir";
+---------------+---------------------------------+
| Variable_name | Value                           |
+---------------+---------------------------------+
| plugin_dir    | /application/mysql/lib/plugin/ |
+---------------+---------------------------------+
1 row in set (0.00 sec)
```

```
$ ll /application/mysql/lib/plugin/semisync_*
-rwxr-xr-x. 1 root root 172944 Dec 15 12:53 /application/mysql/lib/plugin/semisync_master.so #主库
-rwxr-xr-x. 1 root root  94066 Dec 15 12:53 /application/mysql/lib/plugin/semisync_slave.so  #从库
```

在master上安装

```
mysql> install plugin rpl_semi_sync_master soname "semisync_master.so";
Query OK, 0 rows affected (0.16 sec)
```

在slave上安装

```
mysql> install plugin rpl_semi_sync_slave soname "semisync_slave.so";
Query OK, 0 rows affected (0.05 sec)
```

二. 查看当前半同步状态
master上

```
mysql> show variables like "rpl_semi_sync%";
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| rpl_semi_sync_master_enabled      | OFF   |
| rpl_semi_sync_master_timeout      | 10000 |
| rpl_semi_sync_master_trace_level  | 32    |
| rpl_semi_sync_master_wait_no_slave | ON    |
+-----------------------------------+-------+
4 rows in set (0.00 sec)
```

slave上

```
mysql> show variables like "rpl_semi_sync%";
+----------------------------------+-------+
| Variable_name                    | Value |
+----------------------------------+-------+
| rpl_semi_sync_slave_enabled      | OFF   |
| rpl_semi_sync_slave_trace_level | 32    |
+----------------------------------+-------+
2 rows in set (0.00 sec)
```

三. 开启半同步开关
master上

```
mysql> set global rpl_semi_sync_master_enabled = 1;
Query OK, 0 rows affected (0.01 sec)

mysql> show variables like "rpl_semi_sync%";
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| rpl_semi_sync_master_enabled      | ON    |
| rpl_semi_sync_master_timeout      | 10000 |
| rpl_semi_sync_master_trace_level  | 32    |
| rpl_semi_sync_master_wait_no_slave | ON    |
+-----------------------------------+-------+
4 rows in set (0.00 sec)
```

slave上

```
mysql> set global rpl_semi_sync_slave_enabled = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like "rpl_semi_sync%";
+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| rpl_semi_sync_slave_enabled     | ON    |
| rpl_semi_sync_slave_trace_level | 32    |
+---------------------------------+-------+
2 rows in set (0.00 sec)
```

salve上要重启slave

```
mysql> stop slave IO_THREAD;
Query OK, 0 rows affected (0.06 sec)

mysql> start slave IO_THREAD;
Query OK, 0 rows affected (0.00 sec)
```

master上在检查半同步状态

```
mysql> show status like "Rpl_semi_sync%";
+--------------------------------------------+-------+
| Variable_name                              | Value |
+--------------------------------------------+-------+
| Rpl_semi_sync_master_clients               | 1     |
| Rpl_semi_sync_master_net_avg_wait_time     | 0     |
| Rpl_semi_sync_master_net_wait_time         | 0     |
| Rpl_semi_sync_master_net_waits             | 0     |
| Rpl_semi_sync_master_no_times              | 0     |
| Rpl_semi_sync_master_no_tx                 | 0     |
| Rpl_semi_sync_master_status                | ON    |
| Rpl_semi_sync_master_timefunc_failures     | 0     |
| Rpl_semi_sync_master_tx_avg_wait_time      | 0     |
| Rpl_semi_sync_master_tx_wait_time          | 0     |
| Rpl_semi_sync_master_tx_waits              | 0     |
| Rpl_semi_sync_master_wait_pos_backtraverse | 0     |
| Rpl_semi_sync_master_wait_sessions         | 0     |
| Rpl_semi_sync_master_yes_tx                | 0     |
+--------------------------------------------+-------+
14 rows in set (0.01 sec)
```

参数解释

```
| Rpl_semi_sync_master_clients               | 1     |
# 记录支持半同步的slave的个数
| Rpl_semi_sync_master_net_avg_wait_time     | 0     |
# master 等待slave回复的平均等待时间，单位微秒
| Rpl_semi_sync_master_net_wait_time         | 0     |
# master 总的等待时间
| Rpl_semi_sync_master_net_waits             | 0     |
# master 等待slave回复的的总的等待次数
| Rpl_semi_sync_master_no_times              | 0     |
# master 关闭半同步复制的次数
| Rpl_semi_sync_master_no_tx                 | 0     |
# master 没有收到slave的回复而提交的次数
| Rpl_semi_sync_master_status                | ON    |
# 标记master现在是否是半同步复制状态
| Rpl_semi_sync_master_timefunc_failures     | 0     |
# 时间函数未正常工作的次数
| Rpl_semi_sync_master_tx_avg_wait_time      | 0     |
# 开启Semi-sync，事务返回需要等待的平均时间
| Rpl_semi_sync_master_tx_wait_time          | 0     |
# 事务等待备库响应的总时间
| Rpl_semi_sync_master_tx_waits              | 0     |
# 事务等待备库响应的总次数
| Rpl_semi_sync_master_wait_pos_backtraverse | 0     |
# 改变当前等待最小二进制日志的次数
| Rpl_semi_sync_master_wait_sessions         | 0     |
# 当前有多少个session 因为slave 的回复而造成等待
| Rpl_semi_sync_master_yes_tx                | 0     |
# master 成功接收到slave的回复的次数
```

# 半同步复制测试

主库插入数据

```
mysql> use bbs;
Database changed
mysql> select * from student;
+----+------+-----+-------+
| id | name | age | dept  |
+----+------+-----+-------+
|  1 | lzyi |  23 | 172cm |
|  2 | lzy  |  33 | 172cm |
|  3 | lz   |  13 | 150cm |
|  4 | l    |  34 | 179cm |
|  5 | wen  |  27 | 178cm |
+----+------+-----+-------+
5 rows in set (0.00 sec)

mysql> insert into student values(6,'liubiao',22,'176cm');
Query OK, 1 row affected (0.00 sec)

mysql> insert into student values(7,'moge',23,'175cm'),(8,'chenp',22,'173cm');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> show status like "Rpl_semi_sync%";
+--------------------------------------------+-------+
| Variable_name                              | Value |
+--------------------------------------------+-------+
| Rpl_semi_sync_master_clients               | 1     |
| Rpl_semi_sync_master_net_avg_wait_time     | 1445  |
| Rpl_semi_sync_master_net_wait_time         | 2891  |
| Rpl_semi_sync_master_net_waits             | 2     |
| Rpl_semi_sync_master_no_times              | 0     |
| Rpl_semi_sync_master_no_tx                 | 0     |
| Rpl_semi_sync_master_status                | ON    |
| Rpl_semi_sync_master_timefunc_failures     | 0     |
| Rpl_semi_sync_master_tx_avg_wait_time      | 1533  |
| Rpl_semi_sync_master_tx_wait_time          | 1533  |
| Rpl_semi_sync_master_tx_waits              | 1     |
| Rpl_semi_sync_master_wait_pos_backtraverse | 0     |
| Rpl_semi_sync_master_wait_sessions         | 0     |
| Rpl_semi_sync_master_yes_tx                | 2     |
+--------------------------------------------+-------+
14 rows in set (0.00 sec)
```

slave关闭IO线程

```
mysql> stop slave IO_THREAD;
Query OK, 0 rows affected (0.05 sec)
```

主库再插入数据并查看状态

```
mysql> insert into student values(9,'nidage',22,'172cm');
Query OK, 1 row affected (10.00 sec)

mysql> select * from student;
+----+---------+-----+-------+
| id | name    | age | dept  |
+----+---------+-----+-------+
|  1 | lzyi    |  23 | 172cm |
|  2 | lzy     |  33 | 172cm |
|  3 | lz      |  13 | 150cm |
|  4 | l       |  34 | 179cm |
|  5 | wen     |  27 | 178cm |
|  6 | liubiao |  22 | 176cm |
|  7 | moge    |  23 | 175cm |
|  8 | chenp   |  22 | 173cm |
|  9 | nidage  |  22 | 172cm |
+----+---------+-----+-------+
9 rows in set (0.00 sec)

mysql> show status like "Rpl_semi_sync%";
+--------------------------------------------+-------+
| Variable_name                              | Value |
+--------------------------------------------+-------+
| Rpl_semi_sync_master_clients               | 1     |
| Rpl_semi_sync_master_net_avg_wait_time     | 976   |
| Rpl_semi_sync_master_net_wait_time         | 2930  |
| Rpl_semi_sync_master_net_waits             | 3     |
| Rpl_semi_sync_master_no_times              | 1     |
| Rpl_semi_sync_master_no_tx                 | 1     |
| Rpl_semi_sync_master_status                | OFF   |
| Rpl_semi_sync_master_timefunc_failures     | 0     |
| Rpl_semi_sync_master_tx_avg_wait_time      | 1533  |
| Rpl_semi_sync_master_tx_wait_time          | 1533  |
| Rpl_semi_sync_master_tx_waits              | 1     |
| Rpl_semi_sync_master_wait_pos_backtraverse | 0     |
| Rpl_semi_sync_master_wait_sessions         | 0     |
| Rpl_semi_sync_master_yes_tx                | 2     |
+--------------------------------------------+-------+
14 rows in set (0.00 sec)
```

我们再看看从库数据

```
mysql> select * from student;
+----+---------+-----+-------+
| id | name    | age | dept  |
+----+---------+-----+-------+
|  1 | lzyi    |  23 | 172cm |
|  2 | lzy     |  33 | 172cm |
|  3 | lz      |  13 | 150cm |
|  4 | l       |  34 | 179cm |
|  5 | wen     |  27 | 178cm |
|  6 | liubiao |  22 | 176cm |
|  7 | moge    |  23 | 175cm |
|  8 | chenp   |  22 | 173cm |
+----+---------+-----+-------+
8 rows in set (0.00 sec)
```

由上看出，主从数据并未同步。
当从库上开启IO线程时

```
mysql> start slave IO_THREAD;
Query OK, 0 rows affected (0.00 sec)
```

主库状态

```
mysql> show status like "Rpl_semi_sync%";
+--------------------------------------------+-------+
| Variable_name                              | Value |
+--------------------------------------------+-------+
| Rpl_semi_sync_master_clients               | 1     |
| Rpl_semi_sync_master_net_avg_wait_time     | 15610 |
| Rpl_semi_sync_master_net_wait_time         | 62440 |
| Rpl_semi_sync_master_net_waits             | 4     |
| Rpl_semi_sync_master_no_times              | 1     |
| Rpl_semi_sync_master_no_tx                 | 1     |
| Rpl_semi_sync_master_status                | ON    |
| Rpl_semi_sync_master_timefunc_failures     | 0     |
| Rpl_semi_sync_master_tx_avg_wait_time      | 1533  |
| Rpl_semi_sync_master_tx_wait_time          | 1533  |
| Rpl_semi_sync_master_tx_waits              | 1     |
| Rpl_semi_sync_master_wait_pos_backtraverse | 0     |
| Rpl_semi_sync_master_wait_sessions         | 0     |
| Rpl_semi_sync_master_yes_tx                | 2     |
+--------------------------------------------+-------+
14 rows in set (0.00 sec)
```

再在从库上查看数据

```
mysql> select * from student;
+----+---------+-----+-------+
| id | name    | age | dept  |
+----+---------+-----+-------+
|  1 | lzyi    |  23 | 172cm |
|  2 | lzy     |  33 | 172cm |
|  3 | lz      |  13 | 150cm |
|  4 | l       |  34 | 179cm |
|  5 | wen     |  27 | 178cm |
|  6 | liubiao |  22 | 176cm |
|  7 | moge    |  23 | 175cm |
|  8 | chenp   |  22 | 173cm |
|  9 | nidage  |  22 | 172cm |
+----+---------+-----+-------+
9 rows in set (0.00 sec)
```

看，主从数据已同步

# 主从复制之同步、异步、半同步复制的比较

**同步复制：**Master提交事务，直到事务在所有的slave都已提交，此时才会返回客户端，事务执行完毕。
缺点：完成一个事务可能会有很大的延迟。

**异步复制：**MySQL本身支持单向的、异步的复制。异步复制指把数据从一台机器拷贝到另一台机器有一个延时，最重要的是这意味着当应用系统的事务提交已经确认时数据并不能在同一时刻同步到从服务器上；这个延时是由网络带宽资源可用性和系统负载决定的。缺点：不能保证一些事件都能够被所有的slave接收。当slave准备好才会向master请求binlog。

**半同步复制：**半同步复制工作的机制处于同步和异步之间，Master的事务提交阻塞，只要一个Slave已收到该事务的事件且已记录。它不会等待所有的Slave都告知已收到，且它只是接收，并不用等其完全执行且提交。

默认情况，我们的支持复制是采用异步的方式进行同步的，导致我们的从服务器数据有可能会落后于主服务器，在一定程度上是不安全的。如果我们主服务器瞬间挂掉，从服务器会来不及复制数据，从而产生了半同步。
开启半同步复制功能后，主服务器只等待多个从服务器中的指定的一台从服务器复制成功，然后才进行其它写操作，使这台从服务器和主服务器上的数据完全同步，而并不管其它的从服务器，当然主服务器不会一直等待从服务器复制成功的。因为万一从服务器挂掉，那么主服务器将一直处于等待状态，而不提供写服务，这就需要我们定一个超时时间，防止等待从服务器时间太长。超过定义的时间，从服务器还没有响应，则把指定的从服务器自动降级到异步模式。

# 实现MySQL主从读写分离的方案

1. 通过程序实现读写分离（性能，效率最佳，推荐使用）
   php和JAVA程序都可以通过设置多个连接文件轻松的实现对数据库的读写分离，及当select时，就去连接读库的连接文件，当update,insert,delete时就连接写库的连接文件。
   通过程序实现读写分离的缺点就是需要开发对程序改造，对下层不透明，但这种方式更容易开发和实现，适合互联网场景。
2. 通过开源的软件实现读写分离
   MySQL-proxy，amoeba等代理软件也可以实现读写分离功能，这些软件的稳定性和功能一般，不建议生产使用。普通公司常用的还是通过客户端程序实现读写分离。
3. 大型门户独立开发DAL层综合软件
   百度、阿里等大型门户大牛众多，会花大力气开发适合自己业务的读写分离、负载均衡、监控报警、自动扩容、自动收缩等一系列功能的DAL层软件。

# MySQL主从复制集群架构的数据备份策略

1. 有主从复制了，还需要做定时全量+增量备份吗？当然需要！
   因为，如果主库有语句级的误操作，例：`drop database www;`，从库也会执行`drop database www;`，这样MySQL主从库就都删除了该数据。
2. 把从库作为数据库备份服务器时
   高并发业务场景备份时，可以选择在一台从库上备份，把从库作为数据库备份服务器时需要：
   a.选择一个不对外服务的从库，确保和主库更新最接近，专门做数据备份用。
   b.开启从库的binlog功能。
   备份时可以选择只停止SQL线程，停止应用SQL语句到数据库，IO线程保留工作状态，执行命令为`stop slave sql_thread;`，备份方式可采取mysqldump逻辑备份或者直接物理备份，例：`cp，tar(/data/)，xtrabackup`(第三方的物理备份软件)，根据总的备份数据量的多少选择，把全备和binlog发送到备份服务器上留存。

# MySQL主从复制延迟的原因及解决方案

- 问题一：一个主库的从库太多，导致复制延迟。
  建议从库数量3~5个位宜，要复制的从节点数量过多，会导致复制延迟。
- 问题二：从库硬件比主库差，导致复制延迟。
  查看master和slave的系统配置，可能会因为机器配置的问题，包括磁盘IO、CPU、内存等各方面因素造成复制的延迟，一般发生在高并发大数据写入场景。
- 问题三：慢SQL语句过多
  例如一条SQL语句，执行时间是20秒，那么从执行完毕，到从库上能查到数据也至少是20秒，这样就延迟20秒了。
  SQL语句的优化一般要作为常规工作不断的监控和优化，如果是单个SQL写入时间过长，可以修改分多次写入。通过看慢查询日志或show full processlist; 命令找出执行时间过长的查询语句或大的事务
- 问题四：主从复制的设计问题。
  例如，主从复制单线程，如果主库写并发太大，来不及传送到从库就会导致延迟。更高版本的MySQL可以支持多线程复制，门户网站则会自己开发多线程同步功能。

- 问题五：主从库之间的网络延迟。
  主从库的网卡、网线、连接的交换机等网络设备都可能成为复制的瓶颈，导致复制延迟，另外，跨公网主从复制很容易导致主从复制延迟。
- 问题六：主库读写压力大，导致复制延迟。
  主库硬件要搞好一点，架构的前端加buffer以及缓存层。
  通过read-only参数让从库只读访问，read-only参数选项可以让从服务器值允许来自从服务器线程或具有super权限的数据库用户进行更新。可以确保从服务器不接受来自用户端的非法用户更新。

**小结**

| | |
|---|---|
| 从库太多导致复制延迟 | 优化：建议从库数量 3-5 个为宜 |
| 从库硬件比主库差 | 优化：建议提升硬件性能 |
| 慢 SQL 语句过多 | 优化：SQL 语句执行时间过长，建议优化慢查询 SQL 语句 |
| 主从复制的设计问题 | 优化：主从复制 IO 是单线程的，可通过多线程 IO 方案解决；MySQL5.6.3 支持多线程 IO 复制 |
| 主从之间的网络延迟 | 优化：尽量使用短链路，提升端口带宽 |
| 主库读写压力大 | 优化：前端加 buffer 和缓存层；主从延迟不同步，不管有多延迟，只要不影响业务就没事。 |
| 业务设计缺陷导致延迟影响业务 | 优化：从库没有数据改读主库 |

# MySQL数据库集群授权

读写分离集群：如何授权web用户访问？

## 授权法一

**传统方法，使用两个用户；或者使用两个验证条目。**
主库上：
用户：web_w 密码：oldboy123 端口：3306 连接IP：10.0.0.7 连接库：oldboy
权限：SELECT，INSERT，UPDATE，DELETE

从库上：
用户：web_r 密码：oldboy123 端口：3306 连接IP：10.0.0.8 连接库：oldboy
权限：SELECT

## 授权法二

**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。**
主库和从库使用相同的用户，但授予不同的权限。
主库上对web用户授权如下：
用户：web
密码：oldboy123
端口：3306
连接IP：10.0.0.7
连接库：oldboy
权限：SELECT，INSERT，UPDATE，DELETE
命令：GRANT SELECT,INSERT,UPDATE,DELETE ON web.* TO web@'10.0.0.%' identified by 'oldboy123';

从库上对web用户授权如下：
用户：web
密码：oldboy123
端口：3306
连接IP：10.0.0.8
连接库：oldboy
权限：SELECT

提示：由于主库和从库是同步复制的，所有从库上的web用户会自动和主库的一致，即无法实现只读。
实现上述方法的方法：
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mysql同步，主库的配置参数如下：

```
1. binlog-ignore-db=mysql      #这个是关键，binlog更新忽略mysql数据库
2. replicate-ignore-db=mysql   #单配复制忽略数据库不生效
```

# 授权法三

**从库上设置read-only参数控制web用户读写权限。**
主库从库：
用户：web
密码：oldboy123
端口：3306
连接IP：10.0.0.7
连接库：oldboy
权限：SELECT，INSERT，UPDATE，DELETE
配置方法：从库配置文件my.cnf中添加read-only一行即可。

**MySQL read-only参数功能与使用实践**
测试read-only参数用于从slave中的情况

1. –read-only参数功能描述
   –read-only参数选项可让从服务器只允许来自从服务器线程或具有SUPER权限的用户的更新，可确保从服务器不接受来自用户端的更新
2. –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。
   C、来自从服务器线程可以更新
3. –read-only参数生产环境应用场景
   可以在从库slave中使用read-only参数，确保从库数据不被普通用户非法更新。

# 同步部分库

```
DO：同步少量库
binlog-do-db=db_oldboy
replicate-do-db=db_oldboy            #如需跨数据库更新并且不想复制这些更新，不应使用该选项
replicate-do-table=db_oldboy         #允许跨数据库更新。
replicate-wild-do-table=db_oldboy    #用于跨数据库更新

ignore：排除
binlog-ignore-db=mysql
replicate-ignore-db=mysql                #如需跨数据库更新并且不想复制这些更新，不应使用该选项
replicate-ignore-table=mysql             #该选项可以跨数据库进行更新。
replicate-wild-ignore-table=db_oldboy    #该选项可以跨数据库进行更新
```

结论1：只在从库上配置`replicate-ignore-db=mysql`，并不能做到从库不同步mysql库。
结论2：只在主库上配置`replicate-ignore-db=mysql`，并不能做到从库不同步mysql库。
结论3：只有在主从库上分别配置`replicate-ignore-db=mysql`，才可做到从库不同步mysql库。

总结：如果要想在从库上忽略mysql同步，方法如下：

1. 结论3：只有在主从库上分别配置`replicate-ignore-db=mysql`，才可做到从库不同步mysql库。
2. 在主库上设置`binlog-ignore-db=mysql`不记录binlog，来达到从库不同步mysql库

提示：多库、多表，分多行写。
参考资料：mysql手册—第五章：数据库管理和第六章：MySQL中的复制

特别注意：
如果服务器用`binlog-ignore-db=mysql=sales`启动，并且执行`USE prices; UPDATE sales.january SET amount-amount+1000;`该语句不写入二进制日志。

# 高可用角色切换原理

一主多从，主库down机手工恢复过程。

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
```

# 选择接班人

是否事先指定接班人——>太子，接班人如何选择:
一. 半同步从库（谷歌半同步插件 MySQL5.5是自带的）
**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，网络不稳定时，主库持续等待。**
除了实时写，还有以下解决措施：

1. 连不上S1的时候会自动转为异步。
2. 设置10秒超时，超过10秒转为异步。
3. S1网络，硬件要好，不提供服务，干等接管。

涉及的两个插件文件：

```
$ ll /application/mysql/lib/plugin/semisync_*
-rwxr-xr-x. 1 root root 172944 Dec 15 12:53 /application/mysql/lib/plu
gin/semisync_master.so  #主库半同步插件
-rwxr-xr-x. 1 root root  94066 Dec 15 12:53 /application/mysql/lib/plu
gin/semisync_slave.so   #从库半同步插件
```

二. 从库S1，啥也不干，只做同步的从库（如：百度500台服务器，可以有多余的服务器专门做同步）

三. 皇帝驾崩现选（耽误事，容易被篡位）

确定主之后，要进行角色切换，将S1提升为新主：

# 主库宕机有两种情况

一. 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）

1. 提升S1为M1的操作

1）调配置read-only。授权用户select，变成增删改查，开启binlog。如果做了双主同步，就直接切即可，啥都不用做。
2）删除master.info和relay-log*
3）登录数据库执行reset master
4）重启数据库，提升S1位M1完毕。

2. 所有从库：`CHANGE MASTER TO ,MASTER_LOG_FILE='', MASTER_LOG_POS=''`
—>S1===》M1
—>S2
—>S3
—>S4
—>S5

二. 如果主库连不上

1. 选择半同步从库提升主库，把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如下，所有从库CHANGE MASTER同1-b。
1）调配置read-only。授权用户select，变成增删改查，开启binlog。如果做了双主同步，就直接切即可，啥都不用做。
2）删除master.info和relay-log*
3）登录数据库reset master
4）重启数据库，提升S1位M1完毕。
2. 如果是S1啥也不干的方法
提升啥也不干的S1为主库M1，操作见1-a步骤，所有从库CHANGE MASTER同1-b.
3. 主库宕机，没有事先指定哪个从库为主库。

**选主的方法**
一. 确保IO和SQL都读取了自己的LOG，并应用到数据库。
①登录所有从库 show processlist;看两个线程的更新状态。
②登录从库32、33分别查看谁更新的更快（双主5从）。

二. 确保更新完毕，查看所有从库的master.info，查看谁更新的更快
看看4个从库哪个最快，经过测试没有延迟的情况POS差距很小，甚至是一致的

```
$ cat /data/3306/data/master.info
$ cat /data/3307/data/master.info
18
mysql-bin.000019
107
172.16.1.52
rep
oldboy
3306
60
0
```

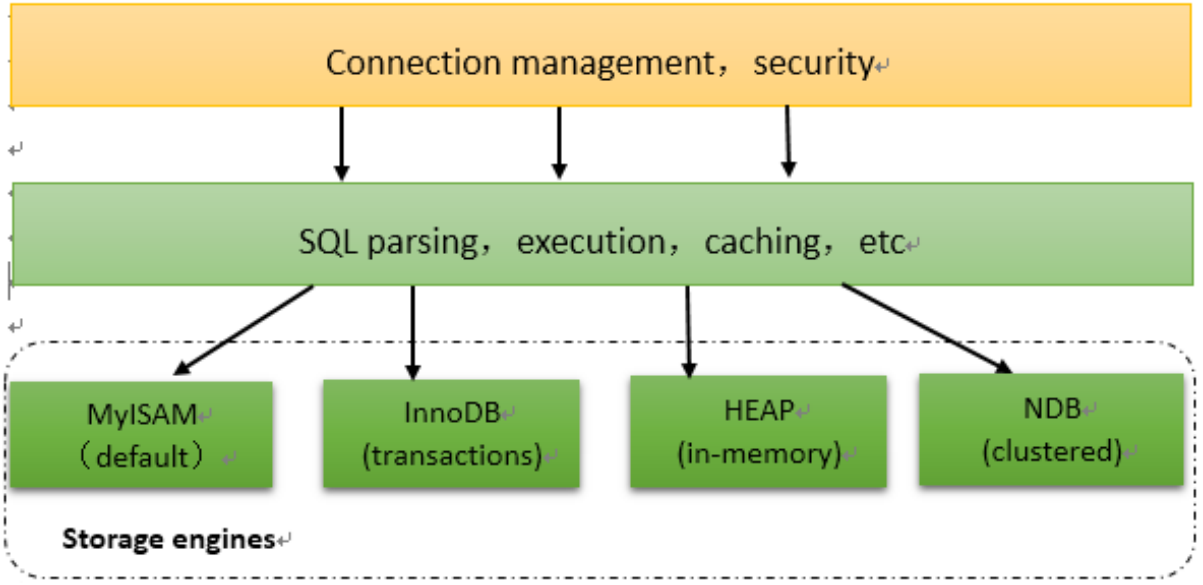
相同文件选个POS最大的作为主库，例如：32 3306。
不同文件则选择文件及位置点最大的为主库，补全（通过中继日志）其他从库。

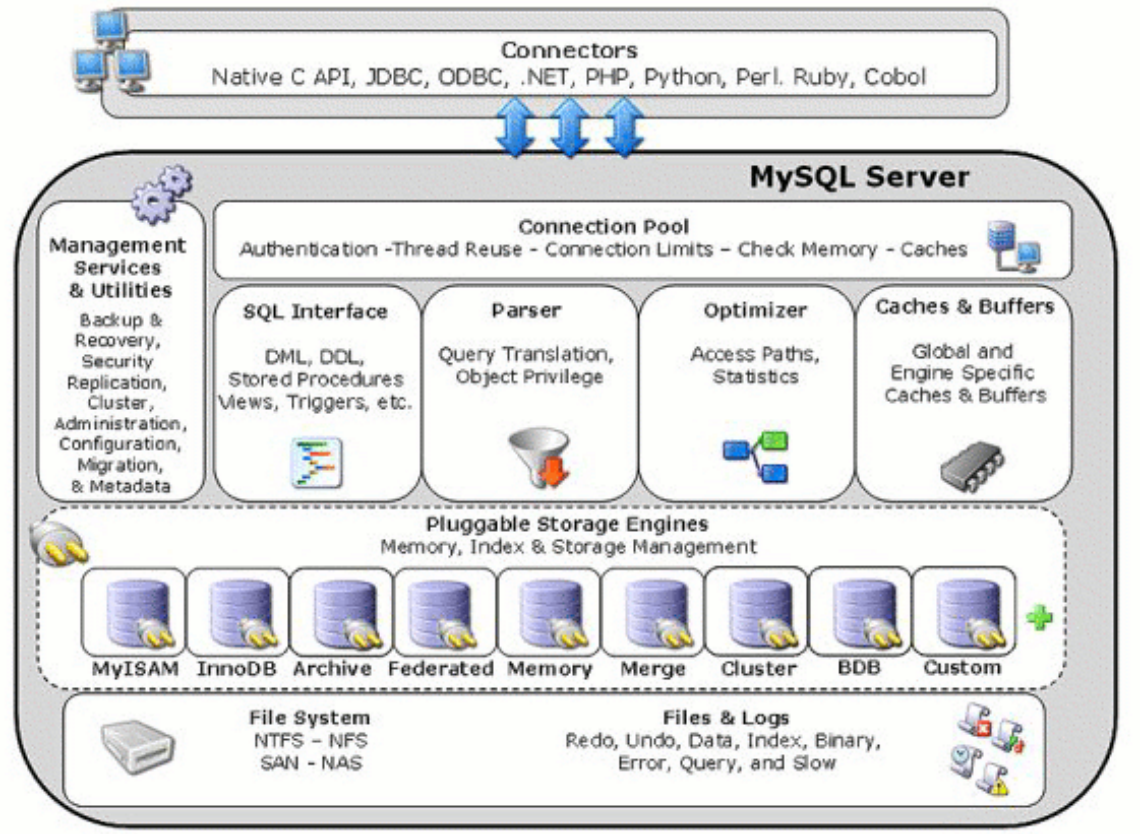
三. 切换角色：
提升选择的S1为主库M1，操作见1-a步骤，所有从库CHANGE MASTER同1-b.
补全（通过中继日志）所有其他从库，和当前准备为主的数据一致。

# MySQL引擎

**存储引擎是什么？**
先来个比喻：录制一个视频文件，可以转成不同的格式，如：MP4、AVI、WMV等，而存在我们电脑的磁盘也会存在不同类型的文件系统，如：Windows的ntfs，fat32，Linux的ext[2-4]、XFS。但是，我们从视频里看到的内容都是一样的，直观区别是：占用系统的空间大小与清晰度可能不一样。
那么，数据库表里的数据存储在数据库里及磁盘上，与上述比喻的格式特征类似，也有很多种存储方式。但是，对于用户和应用程序来说同样一张表的数据，无论用什么引擎来存储，用户看到的数据都是一样的。不同的引擎在存取、功能、占用空间、读取性能等略有区别。
MySQL常见存储引擎：MyISAM和InnoDB；目前5.5版，MyISAM和InnoDB都已经支持。

MySQL存储引擎的架构：
MySQL的存储引擎是MySQL数据库的重要组成部分，MySQL常用的表的引擎为MyISAM和InnoDB两种。每种引擎在MySQL里是通过插件的方式使用的，MySQL可同时支持多种存储引擎。
MySQL存储引擎体系结构简图：






# MyISAM引擎

MyISAM引擎是关系型数据库MySQL管理系统的默认存储引擎（MySQL 5.5.5以前），这种MySQL表存储结构从旧的ISAM代码扩展出许多有用的功能。在新版的MySQL中，InnoDB引擎由于其对事务参照完整性，及更高的并发性等优点开始逐步取代MyISAM引擎。
为什么早期选择MyIASM：硬件很小，更新少，浏览多，MyISAM更适合。

每个MyISAM表都对应硬盘上的三个文件。这三个文件有一样的文件名，只有不同的扩展名指示其类型用途：

> .frm文件保存表的定义，这个文件并不是MyISAM引擎的一部分，而是服务器的一部分；
> .MYD保存表的数据；
> .MYI是表的索引文件。
> .MYD和.MYI是MyISAM的关键点。

```
ll /data/3306/data/mysql/user*
-rw-rw----. 1 mysql mysql 10630 Dec 13 19:41 /data/3306/data/mysql/user.frm
-rw-rw----. 1 mysql mysql   380 Dec 21 17:34 /data/3306/data/mysql/user.MYD
-rw-rw----. 1 mysql mysql  2048 Dec 21 17:34 /data/3306/data/mysql/user.MYI
```

# MyISAM引擎特点

1. 不支持事务（事务是指逻辑上的一组操作，组成这组操作的各个单元，要么全成功，要么全失败）。
2. 表级锁定，数据更新时锁定整个表：其锁定机制是表级锁定，虽然可以让锁定的实现成本很小，但是同时也大大降低了其并发性能。
3. 读写互相阻塞，不仅会在写入的时候阻塞读取，MyISAM还会在读取的时候阻塞写入，但读本身并不会阻塞另外的读。
4. 只会缓存索引：MyISAM可通过key_buffer_size缓存索引，用以提高访问性能，减少磁盘IO，但是这个缓存区只会缓存索引，而不会缓存数据。

```
$ grep key_buffer /data/3306/my.cnf
key_buffer_size = 16M
```

5. 读取速度较快，占用资源相对少。
6. 不支持外键约束，但支持全文索引。
7. MyISAM引擎是MySQL5.5.5以前版本的缺省的存储引擎（is the default storage engine prior to MySQL 5.5.5）。

# MyISAM引擎适用的生产业务场景

1. 不需要事务支持的业务（如转账就不行）。
2. 一般为读数据比较多的应用，读写都频繁场景不适用，读多或写多的都适合。
3. 读写并发访问相对较低的业务（纯读写高并发也可以）（锁表机制问题）。
4. 数据修改相对较少的业务（阻塞问题）。
5. 以读为主的业务，例：数据系统表、www/blog，图片信息数据库，用户数据库，商品库等业务。
6. 对数据一致性要求不是非常高的业务（不支持事务）。
7. 硬件资源比较差的机器可用MyISAM引擎（占用资源少）。
8. 使用读写分离的MySQL的从库可使用MyISAM。

**小结：**
单一对数据库的操作都可使用MyISAM，所谓单一就是尽量纯读，或纯写（insert，update，delete）等。

# MyISAM引擎调优精要

1. 设置合适的索引（缓存机制）。
2. 调整读写优先级，根据实际需求确保重要操作更优先执行。
3. 启用延迟插入改善大批量写入性能（降低写入频率，尽可能多条数据一次性写入）。
4. 尽量顺序操作让insert数据都写入到尾部，减少阻塞。
5. 分解大的时间长的SQL操作，降低单个操作的阻塞时间。
6. 降低并发数（减少对MySQL访问），某些高并发场景通过应用进行排队队列机制Q队列。
7. 对于相对静态（更改不频繁）的数据库数据，充分利用Query Cache或memcached缓存服

务可以极大的提高访问效率，网站动态内容静态化，减少对数据库的访问。

```
query_cache_size = 256M    # 有就行，不要超过512M
query_cache_limit = 1M
query_cache_min_res_unit = 2k
```

8. MyISAM的Count只有在全表扫描是特别高效，带有其它条件的count都需要进行实际的数据访问。

```
mysql> select count(*) from oldboy.test;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)
```

9. 可以把主从同步的主库使用innodb，从库使用MyISAM引擎（不推荐）。

# InnoDB引擎

InnoDB引擎是MySQL数据库的另一个重要的存储引擎，正式为目前MySQL AB所发行新版的标准，被包含在所有二进制安装包里。和其它存储引擎相比，InnoDB引擎的优点是支持兼容ACID的事务（类似于Postgre SQL），以及参数完整性（及对外键的支持）。

Oracle公司2005年10月收购了Innobase。Inoobase采用双认证授权。它使用GNU发行，也允许其它想将InnoDB结合到商业软件的团体获得授权。

MySQL 5.5.5后数据库的默认存储引擎为InnoDB。

# InnoDB引擎特点

1. 支持事务：支持4个事务隔离级别，支持多版本读。
2. 行级锁定（更新时一般是锁定当前行）：通过索引实现，全表扫描仍然会是表锁，注意间隙锁的影响。
3. 读写阻塞与事务隔离级别相关。
4. 具有非常高效的缓冲特性：能缓存索引，也能缓存数据。
5. 整个表和主键以Cluster方式存储，组成一颗平衡树。
6. 所有Secondary Index都会保存主键信息。
7. 支持分区，表空间。类似Oracle数据库。
8. 支持外键约束，5.5以前不支持全文索引，以后支持了。
   外键参考
9. 和MyISAM引擎比，InnoDB对硬件资源要求比较高。

**小结**

Supports transactions，row-level locking and foreign keys。

> **inoodb特点：面试必答项**
> 1、Row-level locking
> 2、Full-text search indexes
> 3、Data caches
> 4、Index caches
> 5、Transactions
> 6、占用资源多
> 7、读写阻塞与事务隔离级别相关
> 8、支持外键

# InnoDB引擎适用的生产业务场景

1. 需要事务支持的业务（具有较好的事务特性）。
2. 行级锁定对高并发有很好的适应能力，但需要确保查询是通过所有完成。
3. 数据读写及更新都较为频繁的场景，如：BBS，SNS，微博，微信等。
4. 数据一致性要求较高的业务，例：充值转账，银行卡转账。
5. 硬件设备内存较大，可以利用InnoDB较好的缓存能力来提高内存利用率，尽可能减少磁盘IO。

```
$ grep -i innodb /data/3306/my.cnf
#default_table_type = InnoDB
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
```

共享表空间对应物理数据文件

```
$ ll /data/3306/data/ibdata1
-rw-rw----. 1 mysql mysql 134217728 Dec 24 10:45 /data/3306/data/ibdata1
```

独立表空间对应物理数据文件

```
innodb_file_per_table
innodb_data_home_dir = /data/xxx
```

6. 相比MyISAM引擎，innodb引擎更消耗资源，速度没有没有MyISAM引擎快。

# InnoDB引擎调优精要

1. 主键尽可能小，避免给`Secondary index`带来过大的空间负担。
2. 建立有效索引避免全表扫描，因为会使用表锁。
3. 尽可能缓存所有的索引和数据，提高响应速度，减少磁盘IO消耗。
4. 在大批量小插入的时候，尽量自己控制事务而不要使用autocommit自动提交。有开关可以控制提交方式。
5. 合理设置`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`每秒都会被刷写日志文件到磁盘，提交事务的时候不做任何操作。
6. 避免主键更新，因为这会到来大量的数据移动。

查看引擎：`mysql> show engines\G`
常用引擎：MyISAM、MEMORY、InnoDB

# 在生产环境中如何修改引擎？

`alter table test engine = MyISAM;`
`alter table test engine = InnoDB;`

```
mysql> alter table test engine = MyISAM;
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show create table test\G
*************************** 1. row ***************************
       Table: test
Create Table: CREATE TABLE `test` (
  `addr` char(4) DEFAULT NULL,
  `address` char(4) DEFAULT NULL,
  `id` int(4) DEFAULT NULL,
  `age` tinyint(2) DEFAULT NULL,
  `name` varchar(16) DEFAULT NULL,
  `shouji` varchar(16) DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=gbk
1 row in set (0.00 sec)
```

**批量修改**

```
$ mysql -uroot -p623913 -S /data/3306/mysql.sock -e "use oldboy;show tables;"|grep -iv "table"|sed -r 's#(.*)#mysql -uroot -p623913 -S /data/3306/mysql.sock -e "use oldboy;alter table \1 engine = innodb;"#g'|bash
```

**建表时的语句应加上指定引擎**

```
Create Table: CREATE TABLE `student` (
  `Sno` int(10) NOT NULL COMMENT '学号',
  `Sname` varchar(16) NOT NULL COMMENT '姓名',
  `Ssex` char(2) NOT NULL COMMENT '性别',
  `Sage` varchar(16) default NULL,
  `Sdept` varchar(16) default NULL COMMENT '学生所在系别',
  KEY `ind_sage` (`Sage`),
  KEY `ind_sno` (`Sno`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8
```

**注意：混合引擎时`key_buffer_size`参数值要调大**

# 有关MySQL引擎常见企业面试题

1. MySQL有哪些存储引擎，各自有什么特点和区别？
   MySQL 5.5：MyISAM、Memory、InnoDB、NDB
2. 生产环境中应如何选用MySQL的引擎？
   在一般的既有读又有写的业务中，建议使用Innodb引擎，一句话尽可能多的使用innodb引擎。
   纯读、纯写可用MyISAM引擎，如系统的MySQL库。
   详细见上文。
3. 不同的引擎如何备份？混合引擎如何备份。

```
# MyISAM或混合
$ mysqldump -uroot -p623913 -A -B -F -x --master-date=2 |gzip >/data/bak_$(date +%F).sql.gz
# innodb
$ mysqldump -uroot -p623913 -A -B -F --master-date=2 --single-transaction|gzip >/data/bak_$(date +%F).sql.gz
```

4. 设置my.cnf

# 事务

## 数据库事务介绍

简单地说，事务就是指逻辑上的一组SQL语句操作，组成这组操作的各个SQL语句，执行时要么全成功要么全失败。
例：小明给小红转账100元，流程如下：
①从小明的银行账户取出100元，计算式为money-100。
②把取出的100元打入小红的银行账户上，小红收到100元，money+100。
上述转账的过程对应的SQL语句：

```
update xiaoming_account set money=money-100 where name='xiaoming';
update xiaohong_account set money=money+100 where name='xiaohong';
```

上述的两条SQL操作，在事务中的操作就是要么都执行，要么都不执行。这就是事务的原子性（Atomicity）。MySQL5.5支持事务的引擎有：innodb、NDB。

# 事务的四大特性（ACID）

1. **原子性（Atomicity）**
   事务是一个不可分割的单位，事务中的所有SQL等操作要么都发生，要么都不发生。
2. **一致性（Consistency）**
   事务发生前和发生后，数据的完整性必须保持一致。
3. **隔离性（Isolation）**
   当并发访问数据库时，一个正在执行的事务在执行完毕前，对于其它的会话是不可见的，多个并发事务之间的数据是相互隔离的。之前备份的参数：`--single-transaction`
4. **持久性（Durability）**
   一个是事务一旦被提交，它对数据库中的数据改变就是永久性的。如果处理错误，事务也不允许撤销，只能通过“补偿性事务“。

# 事务的开启

数据库默认事务是自动提交的，也就是发一条SQL语句他就执行一条。若想多条SQL放在一个事务中执行，则需要使用事务进行处理。当我们开启一个事务，并且没有提交，MySQL会自动回滚事务。或者我们使用rollback命令手动回滚事务。
数据库开启事务命令：
MySQL默认是自动提交的，也就是你提交一个query，他就直接执行，可以通过：

```
set global autocommit=OFF    # 禁止自动提交
set global autocommit=ON     # 开启自动提交
rollback                     # 回滚事务
commit                       # 提交事务
```

提示：事务引擎基于表的，所以要在表上插入、更新测试事务的特性。
事务管理：
`my.cnf`
`autocommit = OFF`
`commit;`
`rollback;`

```
mysql> show variables like 'autocommit';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| autocommit    | ON    |
+---------------+-------+
1 row in set (0.00 sec)
```

**开启事务的方法**
方法一

```
mysql> start transaction;或begin;
mysql> insert into …
mysql> commit;
start transaction后，会自动忽略autocommit参数
```

方法二

```
mysql> set autocommit=off;
mysql> insert into …
mysql> commit;
```

**设置innodb_flush_log_at_trx_commit =参数**
`innodb_flush_log_at_trx_commit`参数值如下

> = 0 ：每秒 write cache & flush disk
> = 1 ：每次commit都 write cache & flush disk
> = 2 ：每次commit都 write cache 然后根据innodb_flush_log_at_timeout（默认为1s）时间 flush disk

# MySQL数据库安全权限控制管理思想

## 制度与流程控制

一. 项目开发制度流程
办公开发环境===》办公测试环境===》IDC测试环境===》IDC正式环境。通过这种较完善的项目开发制度及流程控制，尽可能的防止潜在的问题隐患发生。

二. 数据库更新流程
开发人员提交需求===》开发主管审核===》部门领导审核===》DBA（运维）审核===》DBA（运维）执行项目开发制度及流程控制的数据库更新步骤（每个步骤都要测试），最后在IDC正式环境执行。

需要说明的是：在开发人员-开始提交需求时，就可以同时抄给以上的领导及审核人员，然后，审核人员依次审核。对于特殊紧急需求，可以根据紧急程度特殊处理，这里可制定个紧急需求处理流程，比如：开发人员提交需求===》DBA（运维）审核，然后操作完再汇报给其它审核人员。
通过完善的数据库更新流程控制，可以防止很多潜在的数据丢失、破坏等问题发生。

三. DBA参与项目数据库设计
在项目开发环节上，**DBA或资深运维人员最好参与数据库设计与审核工作**，这样可以从源头降低不良的数据库设计及不良SQL语句的发生，还可以做所有语句的审核工作，包括select，但这个需要评估工作量是否允许，一般的互联网公司实施全审核比较困难。

四. 各种操作申请流程

1. 开发等人员权限申请流程。
2. 数据库更新执行流程。
3. 烂SQL语句计入KPI考核。

五. 定期对内部人员培训
定期给开发及相关人员培训，目的还是从源头上降低不良数据库设计及不良SQL语句的发生，并通过培训让大家知晓数据库性能的重要性，让大家提升开发时照顾数据库性能的意识。

1. 数据库设计规范及制度。
2. SQL语句执行优化，性能优化技巧等。
3. 数据库架构设计等内容。

# 账户权限控制

一.内部开发等人员权限分配

1. 权限申请流程要设置规范、合理，让需求不明确者知难而退。
2. 办公开发和测试环境可以放开权限，IDC测试和正式环境要严格控制数据库写权限，并且读权限和对外业务服务分离。
3. 开发人员正式环境数据库权限分配规则：给单独的不对外服务的正式从库只读权限，不能分配线上正式主库写权限。
4. 特殊人员（如领导），需要权限时，我们要问清楚他做什么，发邮件回复，注明用户名、密码、权限范围，多提醒操作注意事项，如果有可能由DBA人员代替其执行。
5. 特殊账号（all priviliges），由DBA控制，禁止在任何客户端上执行特权账号操作（如智能localhost或其它策略）。

二.WEB账户权限分配制度

1. 写库账号默认权限为select，insert，update，delete。不要给建表改表（create，alter，drop）等的权限，更不能all权限。
2. 读库账号默认权限为select（配合mysql read-only参数用），确保从库对所有非super权限是只读的。
3. 最好专库专账号，不要一个账号管理多个库。碎库特别多的小公司根据情况特殊对待处理。
4. 如果是LAMP、LNMP一体在一台服务器的环境，DB权限主机要设置为localhost，避免用root用户作为web连接用。
5. WEB和数据库分离的服务器的授权可以根据WEB服务数量多少按IP或网段来授权。
6. 安全性和管理方便，总是互相矛盾的，需要到达一个较平衡状态，如何平衡就要根据具体公司和业务来衡量了。

三.WEB账户授权实战案例

1. 生产环境主库用户的账号授权：
   `GRANT SELECT,INSERT,UPDTAE,DELETE ON blog.* TO ‘blog’@’10.0.0.%’ identified by ‘oldboy123’;`
2. 生产环境从库用户的授权：
   `GRANT SELECT ON log.* TO ‘blog’@’10.0.0.%’ identified by ‘oldboy123’;`
   说明：这里表示给10.0.0.0/24的用户blog管理blog数据库的所有表（*表示所有表），只读权限（SELECT），密码为oldboy123。
   当然从库除了做SELECT的授权外，还可以加read-only等只读参数。

四.生产环境读写分离账户设置
给开发人员的读写分离用户设置，除了IP必须要不同外，我们尽量为开发人员的使用提供方便。因此，读写分离的地址，除了IP不同外，账号、密码、端口等看起来都是一样的。这才是人性化的设计，体现了运维或DBA人员的专业。

> 主库（尽量提供写服务）：blog oldboy123 ip:10.0.0.179 port:3306
> 从库（仅提供读服务）：blog oldboy123 ip:10.0.0.180 port:3306
> 提示：两个账号的权限是不一样的。

注意：在程序架构设计上，开发人员应该设计浏览数据优先连接读库，如果读从库数据超时后跳转到主库读数据，这样从程序设计上来保证用户的浏览体验。当然也要根据主库的繁忙程度来综合考虑。具体情况都是根据业务项目需求来抉择。

开发、运维、DBA要通力配合，才能达到一个相对完美的境界。

# 数据库客户端访问控制

1. 更改默认的mysql client端口，如phpmyadmin管理端口为9999，其它客户端也一样。
2. 数据库web client端统一部署在1-2台不对外服务的WEB上，限制IP及9999端口只能从办公室内网访问。
3. 不做公网域名解析，用host实现访问（限制任何IP直接访问）或用内部IP访问。
4. phpmyadmin站点目录独立于所有其它站点根目录外，只能由指定的域名或IP地址访问。
5. 限制使用WEB连接的账号管理数据库，根据开发人员用户角色分配指定账号访问。

6. 按开发及相关人员根据职位角色分配适合的管理账号。
7. 设置指定账号访问权限层次，WEB层使用apache/nginx账户验证，数据库层使用mysql用户登录验证。
8. 统一所有数据库账号登录入口地址，禁止所有开发人员私自上传phpmyadmin等数据库管理的程序。
9. 开通VPN，跳板机，只能通过局域网内部IP管理数据库。

# 系统层控制

1. 限制或禁止开发人员SSH ROOT管理，通过SUDO细化权限，使用日志审计。
2. 对phpmyadmin端configuration等配置文件进行读写权限控制。
3. 取消非指定服务器的所有phpmyadmin WEB连接端。
4. 禁止非管理人员管理有数据库WEB Client端的服务器的权限。

# 读库分业务读写分离

细则补充：对数据库的select等大量测试、统计、备份等操作，要在不对外提供select的单独从库执行。

可以把几个不同的从服务器，更具公司的业务进行拆分。如有为外部用户提供查询服务的从服务器，有用来备份的从服务器，还有提供公司内部后台、脚本，日志分析及开发人员服务的从服务器。这样的拆分除了减轻主服务器的压力外。使得对外用户浏览、对内处理公司内部用户业务，及DBA备份业务互不影响。具体可以用下面的简单架构来说明：

主从架构生产环境从服务器分业务拆分使用案例：

> Master—>S1 ==》对外部用户提供服务器（浏览帖子、博客、文章）
> ————>S2 ==》对外部用户提供服务（浏览帖子、博客、文章）
> ————>S3 ==》对外部用户提供服务（浏览帖子、博客、文章）
> ————>S4 ==》对内部用户提供服务（后台访问、脚本任务、数据分析、开发人员浏览）
> ————>S5 ==》数据库备份服务（开启从服务器binlog功能，可实现增量备份及恢复）

# 数据库运维管理思想核心

1. 未雨绸缪，不要停留在制度上，而是实际做出来。
2. 亡羊补牢，举一反三，切记：别好了伤疤忘了疼。
3. 完备的架构设计及备份、恢复策略。
4. 定期思考、并实战模拟以上策略演练。

未雨绸缪永远比出了问题出了要好的多，出了问题补救是没办法不得已的事，最差的是很多公司，没有亡羊补牢，而是好了伤疤忘了疼，没过多久问题又发生了。

因此，在工作中要尽量做到未雨绸缪，从源头上减少故障的发生。其次，要做到亡羊补牢、举一反三，事情出现一次就不能再出现第二次。当然，完善的备份和恢复策略也是需要做的。只有把这些结合起来，才能把我们运维的工作做的更好。

# 附录1:分析mysql慢查询日志的好工具mysqlsla

mysqlsla是hackmysql.com推出的一款MySQL的日志分析工具，可以分析mysql的慢查询日志、分析慢查询非常好用，能针对库分析慢查询语句的执行频率、扫描的数据量、消耗时间等，包括执行某条sql出现的次数及在slow log数据的百分比、执行时间、等待销的时间等，而且分析出来以后还有语句范例，比mysql自己的mysqldumpslow好用。

mysqlsla不仅仅可用来处理慢查询日志，也可以用来分析其它日志比如二进制日志，普通查询日志等等，其对sql语句的抽象功能非常实用，参数设定简练易用，很好上手。

## 开启mysql慢查询日志

查看mysql慢查询是否开启

```
mysql> show variables like '%slow%';
+---------------------+--------------------------------+
| Variable_name       | Value                          |
+---------------------+--------------------------------+
| log_slow_queries    | OFF                            |
| slow_launch_time    | 2                              |
| slow_query_log      | OFF                            |
| slow_query_log_file | /data/3306/data/db02-slow.log |
+---------------------+--------------------------------+
4 rows in set (0.00 sec)

mysql> show global status like '%slow%';
+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| Slow_launch_threads | 0     |
| Slow_queries        | 0     |
+---------------------+-------+
2 rows in set (0.00 sec)
```

开启MySQL慢查询功能

```
mysql> set global slow_query_log=ON;
Query OK, 0 rows affected (0.09 sec)

mysql> show variables like '%slow%';
+---------------------+-------------------------------+
| Variable_name       | Value                         |
+---------------------+-------------------------------+
| log_slow_queries    | ON                            |
| slow_launch_time    | 2                             |
| slow_query_log      | ON                            |
| slow_query_log_file | /data/3306/data/db02-slow.log |
+---------------------+-------------------------------+
4 rows in set (0.00 sec)
```

查看mysql慢查询时间设置，默认为10秒，修改为记录5秒内的查询

```
mysql> show variables like 'long_query_time';
+-----------------+----------+
| Variable_name   | Value    |
+-----------------+----------+
| long_query_time | 10.000000|
+-----------------+----------+
1 row in set (0.00 sec)

mysql> set global long_query_time=5;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select sleep(6);   #测试mysql慢查询
+----------+
| sleep(6) |
+----------+
|        0 |
+----------+
1 row in set (6.00 sec)

mysql> show variables like '%slow%'; # 查看慢查询日志路径
+---------------------+-------------------------------+
| Variable_name       | Value                         |
+---------------------+-------------------------------+
| log_slow_queries    | ON                            |
| slow_launch_time    | 2                             |
| slow_query_log      | ON                            |
| slow_query_log_file | /data/3306/data/db02-slow.log |
+---------------------+-------------------------------+
4 rows in set (0.00 sec)

mysql> show global status like '%slow%'; #查看mysql慢查询状态
+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| Slow_launch_threads | 0     |
| Slow_queries        | 1     |
+---------------------+-------+
2 rows in set (0.00 sec)
```

查看mysql慢查询日志是否被记录

```
mysql> system cat /data/3306/data/db02-slow.log
/application/mysql-5.5.32/bin/mysqld, Version: 5.5.32-log (Source dist
ribution). started with:
Tcp port: 3306  Unix socket: /data/3306/mysql.sock
Time                 Id Command    Argument
# Time: 160106 11:40:58
# User@Host: root[root] @ localhost []
# Query_time: 6.001133  Lock_time: 0.000000 Rows_sent: 1  Rows_examine
d: 0
SET timestamp=1452051658;
select sleep(6);
```

以上为临时开启，必须修改my.cnf配置文件才可永久开启

```
$ vim /data/3306/my.cnf
[mysqld]
slow_query_log = on  #开启mysql慢查询
long_query_time = 5  #超过5秒的查询记录到log里
log-slow-queries = /data/3306/slow.log  #没有走索引的语句，记录到这里
log_queries_not_using_indexes = on #记录未使用索引的查询
```

然后保存退出，并重启MySQL服务。

```
mysql> show variables like '%slow%';
+---------------------+---------------------+
| Variable_name       | Value               |
+---------------------+---------------------+
| log_slow_queries    | ON                  |
| slow_launch_time    | 2                   |
| slow_query_log      | ON                  |
| slow_query_log_file | /data/3306/slow.log |
+---------------------+---------------------+
4 rows in set (0.00 sec)
```

# 安装mysqlsla

mysqlsla是perl编写的脚本，运行mysqlsla需要perl-DBI和per-DBD-Mysql两模块的支持，因此在运行mysqlsla前需要首先安装DBI模块和相应的数据库DBD驱动，而默认情况下linux不安装这两个模块。

```
$ yum install perl perl-devel perl-DBI perl-Time-HiRes perl-DBD-MySQL
MySQL-devel -y  #下载依赖包
$ wget http://hackmysql.com/scripts/mysqlsla-2.03.tar.gz
$ tar xf mysqlsla-2.03.tar.gz
$ cd mysqlsla-2.03
$ perl Makefile.PL  #执行perl脚本检查包依赖关系
$ make && make install  #安装
```

mysqlsla命令默认会保存在/usr/bin路径下，通常可在任意路径下直接执行。对慢查询日志文件的分析，最简化的调用方式如下：

```
mysqlsla -lt slow [SlowLogFilePath] > [ResultFilePath]
```

# 使用参数说明

1. `-log-type (-lt) type logs`:
   通过这个参数来制定log的类型，主要有slow, general, binary, msl, udl,分析slow log时通过

制定为slow.

2. `-sort`:
   制定使用什么参数来对分析结果进行排序，默认是按照t_sum来进行排序。
   t_sum按总时间排序，
   c_sum按总次数排序，
   c_sum_p: sql语句执行次数占总执行次数的百分比。
3. `-top`:
   显示sql的数量，默认是10,表示取按规则排序的前多少条
4. `-statement-filter (-sf) [+-][TYPE]`:
   过滤sql语句的类型，比如select、update、drop. [TYPE]有SELECT, CREATE, DROP, UPDATE, INSERT，例如"+SELECT,INSERT"，不出现的默认是-，即不包括。
5. `-db` :
   要处理哪个库的日志。

# 统计参数说明

1. `queries total`: 总查询次数
2. `unique`:去重后的sql数量
3. `sorted by`: 输出报表的内容排序 最重大的慢sql统计信息, 包括 平均执行时间, 等待锁时间, 结果行的总数, 扫描的行总数.
4. `Count`: sql的执行次数及占总的slow log数量的百分比.
5. `Time`: 执行时间, 包括总时间, 平均时间, 最小, 最大时间, 时间占到总慢sql时间的百分比.
6. `95% of Time`: 去除最快和最慢的sql, 覆盖率占95%的sql的执行时间.
7. `Lock Time`: 等待锁的时间.
8. `95% of Lock`: 95%的慢sql等待锁时间.
9. `Rows sent`: 结果行统计数量, 包括平均, 最小, 最大数量.
10. `Rows examined`: 扫描的行数量.
11. `Database`: 属于哪个数据库.
12. `Users`: 哪个用户,IP, 占到所有用户执行的sql百分比.
13. `Query abstract`: 抽象后的sql语句.
14. `Query sample`: sql语句.

# 简单使用

语法：

```
1. Slow log: mysqlsla -lt slow slow.log
2. General log: mysqlsla -lt general general.log
3. Binary log: mysqlbinlog bin.log | mysqlsla -lt binary -
```

例子：只取bbs库的select语句、按c_sum_p排序的前2条记录

```
$ mysqlsla -lt slow -sort c_sum_p -sf "+select" -db bbs -top 2 /data/3306/slow.log >/backup/slow_time.log
```

例子：统计慢查询文件为`/data/3306/slow.log`、数据库为bbs的所有select和update的慢查询sql，并查询次数最多的100条sql，并写到sql_time.log文件中

```
$ mysqlsla -lt slow -sf "+select,update" -top 100 -sort c_sum -db bbs /data/3306/slow.log >/backup/sql_time.log
```

# 附录2:XtraBackup MySQL物理备份工具

Xtrabackup是一个对InnoDB做数据备份的工具，支持在线热备份（备份时不影响数据读写），是商业备份工具InnoDB Hotbackup的一个很好的替代品。
Xtrabackup有两个主要的工具：`xtrabackup、innobackupex`

1. xtrabackup只能备份InnoDB和XtraDB两种数据表，而不能备份MyISAM数据表
2. innobackupex是参考了InnoDB Hotbackup的innoback脚本修改而来的。innobackupex是一个perl脚本封装，封装了xtrabackup。主要是为了方便的同时备份InnoDB和MyISAM引擎的表，但在处理myisam时需要加一个读锁。并且加入了一些使用的选项。如slave-info可以记录备份恢 复后，作为slave需要的一些信息，根据这些信息，可以很方便的利用备份来重做slave。
3. 官方文档：http://www.percona.com/docs/wiki/percona-xtrabackup:start

## Xtrabackup可以做什么

在线(热)备份整个库的InnoDB、 XtraDB表

在xtrabackup的上一次整库备份基础上做增量备份(innodb only)

以流的形式产生备份，可以直接保存到远程机器上(本机硬盘空间不足时很有用)

MySQL数据库本身提供的工具并不支持真正的增量备份，二进制日志恢复是point-in-time(时间点)的恢复而不是增量备份。Xtrabackup工具支持对InnoDB存储引擎的增量备份，工作原理如下：

1. 首先完成一个完全备份，并记录下此时检查点的LSN(Log Sequence Number)。
2. 在进程增量备份时，比较表空间中每个页的LSN是否大于上次备份时的LSN，如果是，则备份该页，同时记录当前检查点的LSN。
   首先，在logfile中找到并记录最后一个checkpoint(“last checkpoint LSN”)，然后开始从LSN的位置开始拷贝InnoDB的logfile到xtrabackup_logfile；接着，开始拷贝全部的数据文件.ibd；在拷贝全部数据文件结束之后，才停止拷贝logfile。
   因为logfile里面记录全部的数据修改情况，所以，即时在备份过程中数据文件被修改过了，恢复时仍然能够通过解析xtrabackup_logfile保持数据的一致。

# xtrabackup备份原理

XtraBackup基于InnoDB的crash-recovery功能。它会复制innodb的data file，由于不锁表，复制出来的数据是不一致的，在恢复的时候使用crash-recovery，使得数据恢复一致。

InnoDB维护了一个redo log，又称为transaction log，事务日志，它包含了innodb数据的所有改动情况。当InnoDB启动的时候，它会先去检查data file和transaction log，并且会做二步操作：

XtraBackup在备份的时候， 一页一页地复制innodb的数据，而且不锁定表，与此同时，XtraBackup还有另外一个线程监视着transactions log，一旦log发生变化，就把变化过的log pages复制走。为什么要急着复制走呢？ 因为transactions log文件大小有限，写满之后，就会从头再开始写，所以新数据可能会覆盖到旧的数据。

在prepare过程中，XtraBackup使用复制到的transactions log对备份出来的innodb data file进行crash recovery。

# 实现细节

XtraBackup以read-write模式打开innodb的数据文件，然后对其进行复制。其实它不会修改此文件。也就是说，运行 XtraBackup的用户，必须对innodb的数据文件具有读写权限。之所以采用read-write模式是因为XtraBackup采用了其内置的 innodb库来打开文件，而innodb库打开文件的时候就是rw的。

XtraBackup要从文件系统中复制大量的数据，所以它尽可能地使用posix_fadvise()，来告诉OS不要缓存读取到的数据，从 而提升性能。因为这些数据不会重用到了，OS却没有这么聪明。如果要缓存一下的话，几个G的数据，会对OS的虚拟内存造成很大的压力，其它进程，比如 mysqld很有可能被swap出去，这样系统就会受到很大影响了。

在备份innodb page的过程中，XtraBackup每次读写1MB的数据，1MB/16KB=64个page。这个不可配置。读1MB数据之 后，XtraBackup一页一页地遍历这1MB数据，使用innodb的buf_page_is_corrupted()函数检查此页的数据是否正常， 如果数据不正常，就重新读取这一页，最多重新读取10次，如果还是失败，备份就失败了，退出。在复制transactions log的时候，每次读写512KB的数据。同样不可以配置。

# xtrabackup特点与备份方式

特点：
1、无需停止数据进行InnoDB热备
2、增量备份MySQL
3、流压缩传输到其它服务器
4、能比较容易地创建主从同步
5、备份MySQL时不会增大服务器负载

备份方式：
1、热备份：读写不受影响（mysqldump–>innodb）
2、温备份：仅可以执行读操作（mysqldump–>myisam）
3、冷备份：离线备份，读写都不可用
4、逻辑备份：将数据导出至文本文件中（mysqldump）
5、物理备份：将数据文件拷贝（xtrabackup、mysqlhotcopy）
6、完整备份：备份所有数据
7、增量备份：仅备份上次完整备份或增量备份以来变化的数据
8、差异备份：仅备份上次完整备份以来变化的数据

官方总结使用XtraBackup的几个优点：
1、备份集高效、完整、可用
2、备份任务执行过程中不会阻塞事务
3、节省磁盘空间，降低网络带宽占用
4、备份集自动验证机制
5、恢复更快

## 安装

yum安装

```
yum install http://www.percona.com/downloads/percona-release/redhat/0.1-3/percona-release-0.1-3.noarch.rpm -y
或者：
wget http://www.percona.com/downloads/percona-release/redhat/0.1-3/percona-release-0.1-3.noarch.rpm
rpm -ivH percona-release-0.1-3.noarch.rpm

yum list | grep percona
yum install percona-xtrabackup -y
```

# 附录：MySQL面试题

面试题001：什么是MySQL多实例，如何配置MySQL多实例？

面试题002：如何加强MySQL数据的安全，请你给出可行的思路？

面试题003：企业中MySQL root密码忘了怎么解决，多实例密码忘了又如何解决？

面试题004：MySQL库授权oldboy用户管理bbs库的所有表，172.16.1.0整个网段访问，密码是123456，请给出详细命令(不低于2种方法)?

面试题005：delete from test和truncate table test区别？

面试题006：MySQL 的SQL语句如何优化？ 提示：注意解答问题的高度和宽度

面试题007：网站打开慢，请给出排查方法，如果是因为数据库慢导致，如何排查并解决，请分析并举例？提示：注意解答问题的高度和宽度。

面试题008：MySQL Sleep线程过多如何解决？

面试题009：key_buffer_size参数作用，如何修改这个参数不重启数据库就可以生效？

面试题010：MySQL binlog的工作模式有哪些？各有什么特点，企业中如何选择？

面试题011：工作中数据库被误操作执行了一个删除的SQL语句，你如何完整恢复丢失的数据？提示：注意解答问题的高度、宽度、实战性。

面试题012：企业中MySQLDUMP备份时使用了-A -B参数，请问此时如何进行单表单库恢复？

面试题013：详细MySQL主从同步的原理及配置主从的完整步骤。

面试题014：生产场景不停不锁主库，不夜里操作，如何白天从容批量创建从库？提示：注意解答问题的高度和宽度。

面试题015：MySQL出现同步延迟有哪些原因？如何解决？提示：以经历的生产案例框架解答。

面试题016：企业生产MySQL集群架构如何设计备份方案？你是怎么做的？你的数据库是如何备份的（请答出200服务器以上规模企业数据库备份的解决方案）。提示：注意解答问题的高度和宽度。

面试题017：MySQL主从复制故障如何解决？如何监控主从复制是否故障? 提示：注意解答问题的高度和宽度

面试题018：MySQL如何实现双向互为主从复制A<==>B?

面试题019：MySQL如何实现级联同步A–>B–>C？

面试题020：MySQL数据库如何实现读写分离，你的公司是怎么实现的（请答出200服务器以上规模企业数据库的解决方案）

面试题021：生产场景，一主多从环境，从库宕机，请问你如何恢复？

面试题022：生产场景，一主多从环境，主库宕机，请问你如何恢复（类似MHA高可用原理），注意：高度和宽度。

面试题023：什么是数据库的事务，事务有哪些特性？

面试题024：解释下有关数据库的ACID是什么意思？

面试题025：MySQL有哪些常用引擎？企业中如何批量更改表的引擎？

面试题026： MyISAM与Innodb数据库引擎有什么特点与区别，企业中如何选择？

面试题027：如何调整生产线中MySQL数据库的字符集，例如：从UTF8改成GBK，请给出完整步骤。

面试题028：请描述MySQL里中文数据乱码的背后原理，以及工作中如何防止数据库数据乱码？

面试题029：企业生产MySQL 如何优化？

面试题030 ：MySQL集群和高可用方案有哪些，再生产场景你都用过哪些方案？ 提示：注意解答问题的高度和宽度。

面试题031：你维护的企业里MySQL服务架构能说下么？

面试题032：如何分表分库备份及批量恢复（口述脚本实现过程）？

# 附录：高可用、读写分离

mysql-mmm mysql集群高可用软件（99%）
MMM项目来自 Google
MySQL-MMM官方网站
http://blog.chinaunix.net/uid-20639775-id-154606.html
http://blog.chinaunix.net/uid-20639775-id-154607.html

mysqlmtop工具
http://www.lupaworld.com/article-242608-1.html
http://www.mtop.cc/install-for-centos-redhat-mtop
http://sofar.blog.51cto.com/353572/1381563
http://sofar.blog.51cto.com/353572/1384203

MySQL数据库读写分离amoeba技术
http://docs.hexnova.com/amoeba/
https://www.centos.bz/2012/05/amoeba-for-mysql/

heartbeat+drbd
http://oldboy.blog.51cto.com/2561410/1240412
http://oldboy.blog.51cto.com/2561410/515345

Cobar是关系型数据的分布式处理系统，它可以在分布式的环境下看上去像传统数据库一样为您提供海量数据服务。http://www.cnblogs.com/carlo/archive/2013/09/25/3339649.html