# MYSQL高级部分

# MYSQL介绍

## 主要配置文件

### 二进制日志log-bin

主从复制

### 错误日志log-error

默认是关闭的，记录严重的警告和错误信息，每次启动和关闭的详细信息等

### 查询日志log

默认关闭，记录查询的sql语句，如果开启会减低mysql的整体性能，因为记录日志也是需要消耗资源

### 数据文件

两系统：  
windows：D：\devSoft\MySQLServer5.5\data目录下可以挑选很多库

Linux:看看当前系统中的全部库再进去  
默认路径：/var/lib/mysql

Frm文件：存放表结构

Myd文件：存放表结构  
myi文件：存放表索引

四层结构：1.连接层2.服务层3.引擎层4.存储层

### 如何配置

## 存储引擎简介

1. 如何用命令查看  
   #查看你的mysql现在已提供什么存储引擎  
   show engines；  
   #看你的mysql当前默认的存储引擎  
   show variables like ‘%storage\_engine%’；

## SQL性能下降原因

性能下降SQL慢

执行时间长

等待时间长

原因：1.查询语句写的烂2.索引失效3.关联查询太多（设计缺陷不得已的需求）4.服务器调优及各个参数设置（缓冲、线程数等）

常见通用的Join查询

|  |  |
| --- | --- |
| A,b相交求全部a | Select<select\_list> from table a left join table b ON A.key=B.key |
| A,B相交求排除B,A独有的部分 | Select<select\_list> from tableA A left join TableB B ON A.key=B.key Where B.key is NuLL |
| A,B相交求全部B | Select <select\_list> from tableA A right join tableB B on A.key=B.key |
| A,B相交求两者的交集 | Select <select\_list> from tableA A inner join  TableB B ON A.key=B.key |
| A,B相交求排除掉全部A，B独有的部分 | Select <select\_list> from TableA A Right Join  TableB B ON A.key=B.key Where A.key IS NUll |
| 全连接 A,B相交求全部 | Select <select\_list> From TableA A full outer  Join TableB B ON A.key=B.key |
| A,B相交求去除两者相交的部分 | Select <select\_list> From TableA A Full outer  Join TableB B ON A.key=B.key Where A.keys IS Null ON B.keys IS NUll |

### 索引简介

MySQL官方对索引的定义为：索引是帮助MySQL高效获取数据的数据结构。可以得到索引的本质：索引是数据结构，可以理解为排好序的快速查找数据结构  
索引的目的在于提高查询效率，可以类比字典

索引的分类：1.单值索引：即一个索引只包含单个列，一个表可以有多个单列索引  
2.唯一索引：索引列的值必须唯一，但是允许有空值

1. 复合索引：即一个索引包含多个列

基本语法：

创建：1.create [unique] index indexName ON mytable(columnname(length));  
2.alter mytable ADD [unique] index [indexName] ON(columnname(length))

删除：DROP INDEX [indexName] ON mytable

查看：SHOW INDEX FROM table\_name\G

使用ALTER命令

有四种方式来添加数据表的索引：  
ALTER TABLE tbl\_name ADD PRIMARY KEY (column\_list)该语句添加一个主键，这意味者索引值必须是唯一的，且不能为null  
ALTER TABLE tbl\_name ADD UNIQUE index\_name(column\_list)；这条·语句创建索引值必须是唯一的（除了NULL外，NULL可能会出现多次）  
ALTER TABLE tbl\_name ADD INDEX index\_name(column\_list);添加普通索引，索引可出现多次  
ALTER TABLE tbl\_name ADD FULLTEXT index\_name(column\_list) 该语句指定了索引为FULLTEXT，用于全文索引

## Mysql索引结构

### BTREE索引

检索原理：

Hash索引

Full-text全文索引

R-Tree索引

### 那些情况需要创建索引：

1.主键自动建立唯一索引  
2.频繁作为查询条件的字段应该创建索引

1. 查询中与其他表关联的字段，外键关系建立索引
2. 频繁更新的字段不适合创建索引—因为每次更新不单单是更新了记录还会更新索
3. Where条件里用不到的字段不创建索引
4. 单键/组合索引的选择问题，who？（在高并发下倾向创建组合索引）
5. 查询中排序的字段，排序字段若通过索引去访问将大大提高排序速度
6. 查询中统计或者分组字段

### 那些情况不需要创建索引

1. 表记录太少
2. 经常增删改的表：提高了查询速度，同时却会降低更新表的速度，如对表进行INSERT，UPDATE和DELETE
3. 数据重复且分布均匀的表字段，因此应该只为最经常查询和最经常排序的数据列建立索引。注意，如果某个数据列包含许多重复的内容，为它建立索引就没有太大的实际效果

## 性能分析

MYSQL常见瓶颈： CPU：CPU在饱和的时候一般发生在数据装入内存或从磁盘上读取数据时候

IO：磁盘I/O瓶颈发生在装入数据远大于内存容量的时候

服务器硬件的性能瓶颈：top，free，iostat和vmstat来查看系统的性能状态

Explain：

怎么用：  
explain+sql语句

能干嘛：  
表的读取顺序

数据读取操作的操作类型

那些索引可以使用

那些索引被实际使用

表之间的引用

每张表有多少行被优化器查询

执行计划包含的信息：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Id | Select\_type | table | type | Possible\_keys | key | Key\_len | ref | row |  |

### Id:

select查询的序列号，包含一组数字，表示查询中执行select子句或操作表的顺序

三种情况：id相同，执行顺序由上至下

id不相同，如果是子查询，id的序号会递增，id值越大优先级越高，越优先被执行

id相同不同，同时存在：id如果相同，可以认为是一组，从上往下执行顺序：在所有组中，id值越大，优先级越高，越先执行

衍生=DERIVED

### Select-type

有哪些：1.simple2.primary3.subquery4.DERIVED5.UNION6.UNION RESULT

查询的类型：主要用于区别普通查询，联合查询、子查询等的复杂查询

1. simple：简单的select查询，查询中不包含子查询或者UNION
2. PRIMARY：查询中若包含任何复杂的子部分，最外层查询则被标记为
3. SUBQUERY：在select或WHER列表中包含了子查询
4. DERIVED：在FROM列表中包含的子查询被标记为DERIVED（衍生）MySQL会递归执行这些子查询，把结果放在临时表里
5. UNION：若第二个select出现在UNION之后，则被标记为UNION；若UNION包含在FROm子句的子查询中，外层SELECT将被标记为：DERIVED
6. UNION RESULT 从UNION表获取结果的SELECT

### Type

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ALL | Index | Range | Ref | Eq\_REF | Const,system | NULL |
|  |  |  |  |  |  |  |

访问类型排列：

Type显示的是访问类型，是较为重要的一个指标，结果值从最好到最坏依次是：

System>const>eq\_ref>ref>fulltest>ref\_or\_null>index\_merge>unique\_subquery>index\_subquery>range>index>ALL

System>const>eq\_ref>ref>range>index>ALL

一般来说，得保证至少达到range级别，最好能达到ref

显示查询使用了何种类型：

从最好到最差依次是：

System>const>eq\_ref>range>index>ALL

System：表只有一行记录（等于系统表），这是const类型的特列，平时不会出现，这个也可以忽略不计

Const：表示通过索引一次就找到了，const用于比较primary key或者unique索引。因为只匹配一行数据，所以很快如将主键置于where列表中，MySQL就能将该查询转换为一个常量

Eq\_ref:唯一性索引扫描，对于每个索引键，表中只有一条记录与之匹配。常见于主键或唯一索引扫描

Ref：非唯一性索引扫描，返回某个单独值的所有行，本质上也是一种索引访问，他返回所有匹配某个单独值的行，然而，他可能会找到多个符合条件的行，所以他应该属于查找和扫描的混合体

Range：只检索给定范围的1行，使用一个索引来选择行。Key列显示使用了那个索引

一般就是在你的where语句中出现了between、<、>、IN等的查询

这种范围扫描索引扫描比全表扫描要好，因为它只需要哦开始于索引的某一点，而结束于另一点，不用扫描全部索引

Index：Full Index Scan，index与ALL区别为index类型只遍历索引树，这通常比ALL快，因为索引文件通常比数据文件小。（也就是说虽然all和Index都是读全表，但index是从索引中读取的，而all是从硬盘中读的）

All：FULL Table Scan，将遍历全表以找到匹配的行

备注：一般来说，得保证查询至少达到range级别，最好能达到ref

### Possible\_keys

显示科能应用在这张表中的索引，一个或多个

查询涉及到的字段上若存在索引，则该索引将被列出，但不一定被查询实际使用

### Key

实际使用的索引。如果为NULL，则没有使用索引

查询中若使用了覆盖索引，则该索引仅出现在key列表中

### Key\_len

表示索引中使用的字节数，可通过该列计算查询中使用的索引的长度。在不损失精确性的情况下，长度越短越好

Key\_len显示的值为索引字段的最大可能长度，并非实际使用长度，则key\_len是根据表定义计算而得，不是通过表内检索出的

### Ref：

显示索引的哪一列被使用了，如果可能的话，是一个常数。那些列或常量被用于查询索引列上的值

### Rows

根据表统计信息及索引选用情况，大致估算出找到所需的记录所需要读取的行数

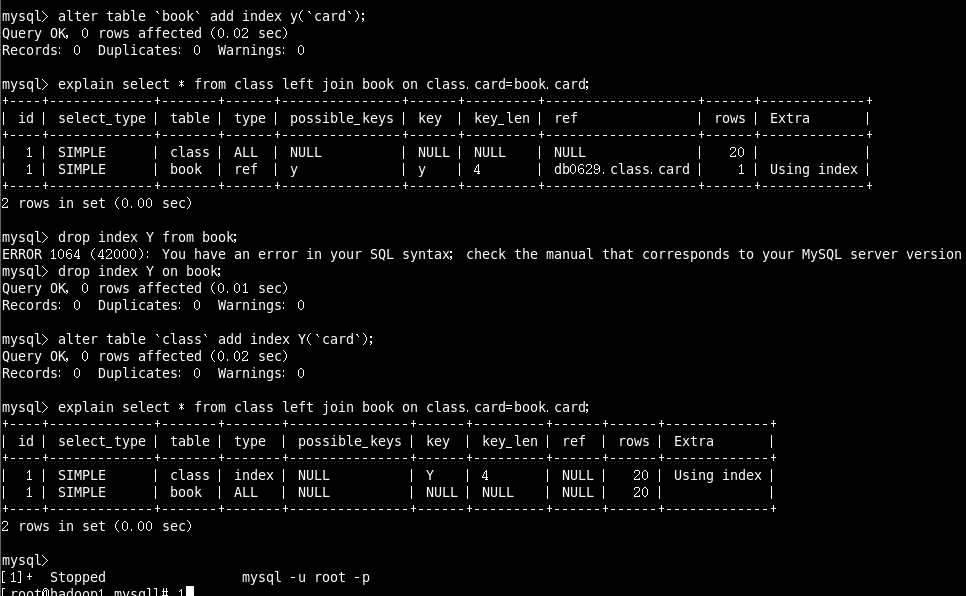
Extra：包含不适合在其他列中显示但十分重要的额外信息

1. Using filesort：说明mysql会对数据使用一个外部的索引排序，而不是按照表内的索引顺序进行读取，Mysql中无法利用索引完成的排序操作成为“文件排序”
2. Using temporary：使用了临时表保存中间结果，MySQL在对查询结果排序时使用临时表。常见于order by和分组查询group by
3. Using index：表示相应的select操作中使用了覆盖索引（Covering Index，避免访问了表的数据行，效率不错。如果同时出现using where，表明索引被引用来执行索引键值的查找，如果没有同时出现using where，表名索引用来读取数据而非执行查找动作

覆盖索引：理解方式1：就是select 的数据列只用从索引中就能够取得，不必读取数据行，MySQL可以利用索引返回select列表中的字段，而不必根据索引再次读取数据文件，换句话说查询列要被所建的索引覆盖

1. Using where :表明使用了where过滤
2. Using join buffer:表示使用了连接缓存；
3. Impossible where ：where子句的值总是false，不能用来获取任何元组
4. Select tables optimized away：在没有GROUP BY子句的情况下，基于索引优化MIN/MAX操作或者对于MyISAM存储引擎优化COUNT（\*）操作，不必等到执行阶段再进行计算，查询执行计划生成的阶段即完成优化
5. Distinct：优化distinct操作，在找到第一匹配的元组后即停止找同样值的动作

## 索引量表优化案例



Left join条件用于确定如何从右表搜索行，左边一定都有

所以右边是我们的关键点，一定要建立索引

Join语句的优化：

尽可能减少join语句中的NestedLoop的循环总次数：“永远用小于结果集驱动大的结果集”

优先优化NestedLoop的内层循环：

保证join语句中被驱动表上join条件字段已经被索引；

当无法保证被驱动表的join条件字段被索引且内存资源充足的前提下，不要太吝惜JoinBuffer的设置；

## 索引失效

1. 全值匹配我最爱
2. 最贱左前缀法则  
   如果索引了很多列，要遵守最最前缀法则。指的是查询从索引的最左前列开始并且不跳过索引中的列。
3. 不在索引列上做任何操作（计算、函数、（自动or手动）类型转换），会导致索引失效而转向全表扫描
4. 存储引擎不能使用索引中范围·条件右边的列
5. 尽量使用覆盖索引（只访问索引的查询（索引列和查询列一致）），减少select \*
6. Mysql在使用·不等于（！=或者<>）的时候无法使用索引会导致全表扫描
7. Is null，is not null也无法使用索引
8. Like以通配符开头（······“%abc…”）mysql索引失效会变成全表扫描的操作
9. 字符串不加单引号索引失效
10. 少用or，用它来连接时会索引失效
11. 补充：mysql中的left函数：1、LEFT()函数是一个字符串函数，它返回具有指定长度的字符串的左边部分。　　LEFT(Str,length);　　接收两个参数：　　　　str：一个字符串；　　　　length：想要截取的长度，是一个正整数；
12. 定值，范围还是排序，一般都是order by 是给个范围 group by基本上都需要进行排序，会有临时表的产生
13. 一般性建议：对于单键索引，尽量选择针对当前query过滤性更好的索引  
    2.在选择组合索引的时候，当前query中过滤性最好的字段在索引字段顺序中，位置越靠前越好，在选择·组合索引的时候，尽量选择可以能够包含当前query中的where子句中更多字段的索引，尽可能通过分析统计信息和调整query的写法来达到选择合适索引的目的

## 优化口诀

全值匹配我最爱，最左前缀要遵守；

带头大哥不能死，中间兄弟不能断；

索引列上少计算，范围之后全失效；

Like百分写最右，覆盖索引不写星；

不等空值还有or，索引失效要少用；

VAR引号不可丢，SQL高级也不难；

## 查询截取分析

### 查询优化

#### 永远小表驱动大表

Exists：

Select… …FROM table where exists（subquery）

该语法可以理解为：将主查询的数据，放到子查询中做条件验证，根据验证结果（TRUE或FALSE）来决定主查询的数据结果是否可以得以保留

提示：

Exists（subquery）只返回TRUE或FALSE，因此子查询的SELECT \*也可以是SELECT 1或其他，官方说法是实际执行时会忽略select清单，因此没有区别

Exists子查询的实际执行过程可能经过了优化而不是我们理解上的逐条对比，如果担忧效率问题，可进行实际检验以确定是否效率有问题

Exists子查询往往也可以用条件表达式、其他子查询或者join来替代，何种最优需要具体问题具体分析

#### Order by关键字优化：

* order by子句，尽量使用index方式排序，避免使用filesort方式排序
* 尽可能在索引列上完成排序操作，遵照索引建的最佳左前缀
* 如果不在索引列上，filesort有两种排序算法
* Mysql就要启动双路排序和单路排序：

1. 双路排序：Mysql4.1之前是使用双路排序，字面意思就是两次扫描磁盘，最终得到数据，读取行指针和order by列，对他们进行排序，然后扫描已经排序好的列表，按照列表中的值重新从列表中读取
2. 从磁盘取排序字段，在buffer进行排序，再从磁盘取其他字段。
3. 单路算法：从磁盘读取查询需要的所有列，按照order by列在buffer对他们进行排序，然后扫描排序后的列表进行输出，它的效率更快一些，避免了第二次读取数据。并且把随机IO变成了顺序IO，但是它会使用更多的空间

* 优化策略

1.增大sort\_buffer\_size 参数的设置

2.增大max\_length\_for\_sort\_data参数的设置

3.why

#### 提高order by的速度

1.order by时select \*是一个大忌，只query需要的字段，这点非常重要。在这里的影响是：

1.1当Query的字段大小总和小于max\_length\_for\_sort\_data而且排序字段不是TEXT|BLOB类型时，会用改进后的算法-单路排序，否则用老算法-多路排序

1.2两种算法的数据都很有可能超出sort\_buffer 的容量，超出之后，会创建tmp文件进行合并排序，导致多次I/O，但是用单路排序算法的风险会更大一些，所以要提高sort\_buffer\_size

#### Group by关键字优化

Group by实质是先排序后进行分组，遵照索引建的最佳左前缀

当无法使用索引列，增大max\_length\_for\_sort\_data参数的设置+增大sort\_buffer\_size参数的设置

Where高于having，能卸写在where限定的条件就不要去having限定了

### 慢查询日志

1. Mysql的慢查询日志是Mysql提供的一种日志记录，它用来记录在MySQL中响应时间超过阈值的语句，具体指运行时间超过long\_query\_time的SQL。则会被记录到慢查询日志中
2. 具体指运行时间超过long\_query\_time值的SQL，则会被记录到慢查询日志中。Long\_query\_time的默认值为10，意思是运行10秒以上的语句
3. 由他来查看哪些SQL超过了我们的最大忍耐时间值。比如一条sql执行超过5秒钟，我们就算慢sql，希望能收集超过5秒的sql，结合之前explain进行全面分析
4. 默认情况下，MySQL数据库没有开启慢查询日志，需要我们手动来设置这个参数
5. 当然，一般，如果不是调优需要的话，一般不建议开启该参数，因为开启慢查询日志或多或少带来一定的性能影响。慢查询日志文件支持将日志记录写入文件
6. 使用set global slow\_query\_log=1开启了慢查询日志只对当前数据库生效，如果mysql重启后则会失效
7. 查看当前多少秒算慢：show variables like ‘long\_query\_time%’；
8. 设置慢的阈值时间：set global long\_query\_time=3；
9. 为什么设置后看不出变化？需要重新连接或新开一个会话才能看到修改至。Show variables like ‘long\_query\_time%’； show global variables like‘long\_query\_time’；
10. 得到返回记录集最多的10个sql  
    mysqldumpslow -s r -t 10 /var/lib/mysql/虚拟机名-slow.log
11. 得到访问次数最多的10个sql  
    mysqldumpslow -s c -t 10 /var/lib/mysql/虚拟机名-slow.log
12. 得到按照时间排序的前10条里面含有左连接的查询语句

Musqldumpslow -s t -t 10 -g “left join” /var/lib/mysql/虚拟机名-slow.log

1. 另外建议在使用这些命令时结合|和home使用，否则有可能出现爆屏情况  
   mysqldumpslow -s r -t 10 /var/lib/mysql/虚拟机名-slow.log | more
2. 查看mysqldumpslow的帮助信息 1.s:是表示按照何种方式排序2.c：访问次数3l：锁定时间4.r：返回记录5.t：查询时间6.al：平均锁定时间7.ar：=-平均返回记录数8.t：即为返回前面多少条的数据； 9.g：后边搭配一个正则匹配模式，大小写不敏感的

### SHOW Profile

1. 是什么

是mysql提供可以用来分析当前会话中语句执行的资源消耗情况。可以用于SQL的调优的出测量

默认情况下，参数处于关闭状态，并保存最近15次的运行结果

分析步骤：

是否支持，看看当前的mysql版本是否支持

开启功能，默认是关闭，使用前需要开启

运行sql

查看结果，show profiles；

诊断sql，show profile cpu，block io for query 上一步前面的问题SQL数字号码

| ALL –显示所有的开销信息

| BLOCK IO –显示块IO相关开销

| CONTEXT SWITCHES –上下文切换相关开销

| CPU –显示CPU相关开销信息

| IPC –显示发送和接收相关开销信息

| MEMORY –显示内存相关内开销信息

| PAGE FAULTS –显示页面错误相关开销信息1

| SOURCE –显示和Source\_function,Source\_file,Source\_line相关的开销信息

| SWAPS –显示交换次数相关开销信息

日常开发需要注意的结论

Converting HEAP to MyISAM 查询结果太大，内存都不够用了往磁盘上搬了

Creating tmp table 创建临时表 –1.拷贝数据到临时表2.用完再删除

Copying to tmp table on disk 把内存中临时表复制到磁盘，危险

Locked

### 全局查询日志

配置启用

命令

Set global general\_log=1;

Set global log\_output=’TABLE’;

此后，你所编写的sql语句，将会记录到mysql库里的general\_log表，可以用下面的命令查看

Select \* from mysql.general\_log;

编码启用

永远不要在生产环境开启这个功能

# MYSQL的锁机制

## 锁的分类

从对数据操作的类型（读/写）分

读锁（共享锁）：针对同一份数据，多个读操作可以同时进行而不会互相影响

写锁：（排它锁）：当前写操作没有完成前，它会阻断其他写锁和读锁

从对数据操作的粒度分

表锁

偏向MyISAM存储引擎，开销小，加锁块；无死锁；锁定粒度大，发生锁重读的概率最高，并发度最低

手动添加表锁

Lock table 表名字 read（write），表名字2read（write），其他；

查看表上加过的锁

Show open tables；

释放表锁

Unlock tables；

案例总结：

MyISAM在执行查询语句（SELECT）前，会自动给涉及的所有表加读锁，在执行增删改查操作前，会自动给涉及的表加写锁

MySQL的表级锁有两种模式

表共享读锁（Table Read Lock）

表独占写锁（Table Write Lock）

读MyISAM表的读操作（加读锁），不会阻塞其他进程对同一表的读请求，但会阻塞对同一表的读请求，但会阻塞对同一表的写请求。只有当读锁释放后，才会执行其他进程的写操作

对MyISAM表的写操作（加写锁），会阻塞其他进程对同一表的读和写操作，只有当写锁释放后，才会执行其他进程的读写操作

简而言之，就是读锁会阻塞写，但是不会阻塞读，而写锁会把读和写都阻塞.

如何分析表锁定

Show status like ‘table%’;

这里有两个状态变量记录MySQL内部表级锁定的情况，两个变量说明如下：

Table\_locks\_immeriate:产生标记锁定的次数，表示可以立即获取锁的查询次数，每立即获取锁增加1；

Table\_locks\_waited:出现表级锁争用而发生等待的次数（不能立即获取锁的次数，每等待一次锁值加1），此值高则说明存在着较严重的表级锁争用情况；

此外，MyISAM的读写锁调度是写优先，这也是muisam不适合做写为主表的引擎。因为写锁后，其他线程不能做任何操作，大量的更新会使查询很难的到锁，从而造成永远阻塞

行锁

偏向InnoDB存储引擎，开销大，加锁慢，会出现死锁；锁定粒度最小，会发生锁冲突的概率最低，并发度也最高

InnoDB与MyISAM的最大不同有两点：一是支持事物，而是采用行级锁

## 间隙锁

### 什么是间隙锁

当我们用范围条件而不是相等条件检索数据，并请求共享或排他锁时，INNODB会给符合条件的已有数据记录的索引项加锁；对应键值在条件范围内但并不存在的记录，叫做‘间隙’

INNODB也会对这个“间隙”加锁，这种锁机制就是所谓的间隙锁

危害：

因为Query执行过程中通过过范围查找的话，它会锁定整个范围内所有的索引键值，即使这个键值并不存在

间隙锁有一个比较致命的弱点，就是当锁定一个范围键值之后，即使某些不存在的键值也会被无辜的锁定，而造成在锁定的时候无法插入键值范围内的任何数据。在某些场景下可能会对性能造成很大的危害

### 如何锁定一行

Begin；

Select \* from test\_innodb\_lock where a=8 for update；

Commit；

行锁的分析

通过检查InnoDB\_row\_lock状态变量来分析系统上的行锁的争夺情况

Show status like ‘innodb\_row\_lock%’;

对各个状态量的说明如下：

Innodb\_row\_lock\_current\_waits：当前正在等待锁定的数量

Innodb\_row\_lock\_time :从系统启动到现在锁定总时间长度；

Innodb\_row\_lock\_time\_avg：每次等待所化平均时间

Innodb\_row\_lock\_time\_max：从系统启动到现在等待最长的一次所花的时间；

Innodb\_row\_lock\_waits：系统启动后到现在总共等待的次数

# 主从复制

## 复制的基本原理：

Slave会从master读取binlog来进行数据同步

Mysql的复制分成三步：

master将改变记录到二进制文件日志。这些记录过程叫做二进制日志事件，binary log events；；

slave将master的binary log events拷贝到它的中继日志（relay log）；

slave重做中继日志中的事件，将改变应用到自己的数据库中。MySQL复制是异步的且串行化的

### 复制的基本原则：

每个slave只有一个master

每个slave只能有一个唯一的服务器ID

每个master可以有多个slave

### 复制的最大问题：

延时

### 一主一从常见配置：

Mysql版本一直且后台以服务运行

主从都配置在【mysqld】结点下，都是小写

主机修改my.ini配置文件

从机修改my.cnf配置文件