#########################数据库高级########################

http://bbs.csdn.net/topics/390620630

mysql> show variables like '%dir%';  
+-----------------------------------------+----------------------------+  
| Variable\_name                           | Value                      |  
+-----------------------------------------+----------------------------+  
| basedir                                 | /usr                       |  
| binlog\_direct\_non\_transactional\_updates | OFF                        |  
| character\_sets\_dir                      | /usr/share/mysql/charsets/ |  
| datadir                                 | /var/lib/mysql/            |  
| ignore\_db\_dirs                          |                            |  
| innodb\_data\_home\_dir                    |                            |  
| innodb\_log\_group\_home\_dir               | ./                         |  
| innodb\_max\_dirty\_pages\_pct              | 75                         |  
| innodb\_max\_dirty\_pages\_pct\_lwm          | 0                          |  
| innodb\_undo\_directory                   | .                          |  
| lc\_messages\_dir                         | /usr/share/mysql/          |  
| plugin\_dir                              | /usr/lib/mysql/plugin/     |  
| slave\_load\_tmpdir                       | /tmp                       |  
| tmpdir                                  | /tmp                       |  
+-----------------------------------------+----------------------------+

数据库文件默认在：cd /usr/share/mysql  
配置文件默认在：/etc/my.cnf

———————————–

数据库目录：/var/lib/mysql/  
配置文件：/usr/share/mysql(mysql.server命令及配置文件)  
相关命令：/usr/bin(mysqladmin、mysqldump等命令)(\*mysql的一种安全启动方式：/usr/bin/mysqld\_safe –user=root &)  
启动脚本：/etc/rc.d/init.d/

首先你可以使用以下的命令来寻找MySQL  
root@stuhome /]# find / -name “mysql” -print  
一般来说mysql是放在/usr/local/mysql/下的。  
然后在其bin目录下有个mysql\_config文件，vi之，你会看见这么一句：  
ldata=’/usr/local/mysql/var’

rpm安装默认目录：  
数据文件：/var/lib/mysql/  
配置文件模板：/usr/share/mysql  
mysql客户端工具目录：/usr/bin  
日志目录：/var/log/  
pid，sock文件目录：/tmp/

注：和其他数据库相比，MySql优点与众不同，它的架构可以在多种不同场景中应用并发挥良好的作用。主要体现在存储引擎的架构上。插件式的存储引擎将查询处理和其他的系统任务以及数据的存储提取分离。这种架构可以根据业务的需求选择合适的存储引擎。

数据库层次：

* 连接层
* 服务层
* 引擎层
* 存储层

SQL查询语句执行顺序：from -> join on -> where -> group by -> having ->select -> order by -> limit

* having 是在分组后对数据进行过滤，where 是在分组前对数据进行过滤
* having后面可以使用分组函数(统计函数)，where后面不可以使用分组函数
* where 是对分组前记录的条件，如果某行记录没有满足where字句的条件，那么这行记录不会参加分组；而having是对分组后数据的约束

## **SQL JOINS**：

* 1. 内连接：select <select\_list> from TableA A inner join TableB B on A.key = B.key

表A 和 表B公共的部分

* 1. 左连接：select <select\_list> from TableA A left join TableB B on A.key = B.key

表A和表B的共有加表A的其他部分，简称A表的所有

3、右连接: select <select\_list> from TableA A right join TableB B on A.key = B.key

4、左外连接select <select\_list> from TableA A left join TableB B on A.key = B.key where B.key is null

表A除去表A和表B共有的部分

5、右外连接select <select\_list> from TableA A right join TableB B on A.key = B.key where A.key id null

6、全连接：select <select\_list> from TableA A full outer join TableB B on A.key = B.key

7、select <select\_list> from TableA A full outer join TableB B on A.key = B.key where A.key is null or B.key is null

Oracle数据库支持full join，mysql是不支持full join的，但仍然可以同过左外连接+ union+右外连接实现

SELECT \* FROM t1

LEFT JOIN t2 ON t1.id = t2.id

UNION

SELECT \* FROM t1

RIGHT JOIN t2 ON t1.id = t2.id

## 索引：

mysql官方定义：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

**可以简单理解为“排好序的快速查找数据结构”**

数据本身之外，数据库还维护了一个满足特定查找算法的数据结构，这些数据结构以某种方式指向数据，这样就可以在这些数据结构的基础上实现高级查找算法，这种数据结构就是索引

一般来说索引本身也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储在磁盘上

优势：提高数据检索的效率，降低数据库的io成本

通过索引对数据进行排序，降低数据排序的成本，降低了cpu的消耗

劣势：实际上索引也是一张表，该表保存了主键与索引字段，并指向实体表的记录，所以索引也是要占用空间的

虽然索引大大提高了查询速度同时却降低了更新表的速度，如对表的insert,update,delete.因为更新表，Mysql不仅要保存数据，还要保存一下索引文件每次更新添加了索引列的字段。

索引分类：

单值索引：即一个索引只包含单了列，一个表可以有多个单列索引

唯一索引：索引列的值必须唯一，但允许有空值

复合索引：即一个索引包含多个列

语法：

创建：create index indexName on mytable(columnname);

alter mytabel add index [indexName] on (columnname);

删除：drop index [indexName] on mytable

查看：show index from table\_name

### 2.1、B树结构

Btree又叫多路平衡树，一颗m叉的Btree特性如下

* 树中每个节点最多包含m个孩子
* 除根节点与子节点外，每个节点至少又【ceil(m/2)】个孩子
* 若根节点不是叶子节点，则至少有两个孩子
* 所有的叶子节点都在同一层
* 每个非叶子节点又n个key与n+1个指针组成，其中ceil(m/2)<=n<=m-1

### B+树数据结构

B + tree为B树的变种，B+树和B树的区别为：

n叉B+树最多含有n个key,而B树最多含有n-1个key

B+树的叶子节点保存所有的key信息，依key大小排序

所有的非叶子节点看作是key的索引部分

## 视图

视图：视图是一种虚拟存在的表。视图并不在数据库中实际存在，行和列数据来自 定义视图的查询中使用的表，并且在引用视图时动态生成。

视图的优势主要包括以下几项：

* 简单：使用视图的用户完全不需要关心后面对应的表的结构、关联条件和筛选条件，对用户来说已经是过滤好的复合条件的结果集。
* 安全：使用视图的用户只能访问他们被允许查询的结果集，我们对表的权限管理并不能限制到某个行某个列，但是通过视图就可以简单的实现。
* 数据独立：一旦视图的结构确定了，可以屏蔽表结构变化对用户的影响，源表增加列对视图没有影响;源表修改列名，则可以通过修改视图来解决；不会造成对访问者的影响。
* 我们可以面向视图进行增删改查，这样会导致原表被操作

创建视图：create view dept2\_view as select \* from dept2;

修改视图：alter view dept2\_view as select dname,age from dept2;

删除视图：drop view if exists dept2\_view;

查看视图：show create view dept2\_view;

### 存储过程和函数

存储过程和函数是事先经过编译并存储在数据库中的一段sql语句的集合，调用存储过程和函数可以简化应用开发人员的很多工作，减少数据在数据库和应用服务器之间的传输，对于提高数据处理的效率是有好处的。

存储过程和函数的区别在于函数必须有返回值，而存储过程没有

创建存储过程：创建带in模式参数的存储过程

create procedure myp2(in name varchar(20))

begin

select e.ename, d.dname from emp e join dept d on e.deptno = de.deptno where e.ename = name;

end;

存储过程的调用：call myp2(‘smith’)

知识小贴士：delimiter $

该关键字用来声明sql语句的分隔符，告诉mysql解释器，该段命令是否已经结束了，mysql是否可以执行了。默认情况下 delimiter是分号

查看存储过程：

* 查询db\_name数据库中的所有的存储过程

select name from mysql.proc where db=’bjpowernode’;

* 查询存储过程的状态信息

show procedure status;

* 查询某个存储过程的定义

show create procedure myp4

* 删除存储过程 drop procedure if exists myp4;

语法：

存储过程是可以编程的，意味着可以使用变量，表达式，控制结构，来完成比较复杂的功能

* declare:通过declare可以定义一个局部变量，该变量的作用范围只能在begin … end块中

create procedure pro\_test2()

begin

declare num int default 5;

select num+ 10;

end;

* 使用set赋值，可以赋值常量或者表达式：set var\_name = expr
* 也可以通过select…into 方式进行赋值操作
* 传递参数 create procedure procedure\_name(in/out/inout 参数名 参数类型)

小知识：@description:这种变量要在变量名称前面加上‘@‘叫做用户变量，代表整个会话过程他都是有作用的。

两个@@的话是系统变量

* repeat结构：

有条件的循环控制语句，当满足条件的时候退出循环，while是满足条件的时候才执行

* loop

游标：

游标是用来存储查询结果集的数据类型，在存储过程和函数中可以使用游标对结果集进行循环处理，游标的使用包括声明，open，fetch和close

**触发器：**

触发器是与表有关的数据库对象，指在insert/update/delete之前或之后，触发并执行触发器中定义的sql语句集合，触发器的这种特性可以协助应用在数据库端确保数据的完整性，日志记录，数据校验等操作

使用old和new 来引用触发器中发生变化的记录内容，这与其他数据是相似的

mysql只支持行级触发器不支持语句级触发器

* 创建触发器

create trigger triger\_name

before/after insert/update/delete

on tb1\_name

[for each row] –行级触发器

begin

end;

通过触发器记录dept2表的数据变更日志，包含增加，修改，删除

存储引擎:

和大多数的数据库不同，mysql中有一个存储引擎的概念，针对不同的存储需求可以选择最优的存储引擎。

存储引擎就是存储数据，建立索引，更新查询数据等等技术的实现方式。存储引擎是基于表的，而不是基于库的，所以说存储引擎也可以说是表类型。

查看存储引擎：show engines

查看默认的存储引擎：show variables like ‘%storage\_engine%’;

Mysql默认存储引擎：InnoDB:支持事务安全，行锁（适合高并发），支持外键

MyISAM：不支持事务，支持表锁，不支持外键

创建表时在后面填写这两种类型

on delte restict --🡪 删除主表数据时，如果有关联记录，不删除

on update cascade --🡪 更新主表时，如果子表有关联记录，更新子记录

### 5、优化SQL

查看sql执行频率

* 查看当前session中所有统计参数的值show status like‘com\_\_\_\_\_\_\_’
* 查询全局的信息 show global status like‘com\_\_\_\_\_\_\_’
* 查询Innodb中的sql语句的操作数量:show global status like ‘Innodb\_rows\_%’

定位低效率执行SQL

可以通过以下两种方式定位执行效率较低的SQL语句

* 慢查询日志
* show processlist
* explain分析执行计划：

通过以上步骤查询到效率低的SQL语句后，可以通过explain或者desc命令获取mysql如何执行select语句的信息，包括select语句执行过程中如何连接和连接的顺序

查询sql语句执行计划

explain select \* from tb\_item where id = 1;

* explain之id

id字段是select查询的序列号，是一组数字，表示查询中执行select子句或者是操作表的顺序，id有三种情况

1. id相同表示加载表的顺序是从上到下
2. id不同id值越大，优先级越高，越先被执行
3. id有相同也有不同，同时存在。id相同的可以认为是一组，从上往下顺序执行,在所有的组中，id的值越大，优先级越高越先执行

explain之select\_type

表示select的类型常见的取值，如下表所示

* simple: 简单的select查询，查询中不包含子查询或者union
* primary:查询中若包含任何复杂的子查询，最外层查询标记为该标识
* subquery: 在select或where列表中包含了子查询
* derived: 在from列表中包含的子查询，把结果放在临时表中
* union: 若第二个select出现在union之后，则标记为union;若union包含在from子句之后子句的子查询中，外层select将被标记为：derived
* union result: 从union表获取结果集的select

explain之table:展示这一行数据是关于那一张表的

explain之type:type显示的是访问类型，是较为重要的一个指标，可取值为：

* null : mysql不访问任何表，索引，直接返回结果

select now();

* system:表只要一行记录，这是const类型的特列，一般不会出现
* 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。
* eq\_ref：类似ref,区别在于使用的是唯一索引，使用主键的关联查询，关联查询出的记录只有一条。常见与主键或唯一索引扫描。
* ref:非唯一性索引扫描，返回匹配某个单独值所有行。返回所有匹配某个单独值的所有行（多个）
* rang: 只检索给定返回的行，使用一个索引来选择行。where之后出现between ,<>,in等操作

explain之key:

possible\_keys: 显示可能应用在这张表的索引，一个或多个

key: 实际使用的索引，如果为null,则没有使用索引

key\_len: 表示索引中使用的字节数，该值为索引字段最大可能长度，并非实际使用长度，在不损失精确性的前提下，长度越短越好

* index: index与all的区别为index类型这是遍历了索引树，通常比all快，all是遍历数据文件
* all: 将遍历全表已找到最匹配行

explain之rows: 扫描行的数量

explain之extra: 其他的额外的执行计划信息，在该列显示

* 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进行读取，称为”文件排序”，效率低
* 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 group by
* using index: 表示相应的select操作使用了覆盖索引，避免访问表的数据行，效率不错

show profile分析sql:  
 查看当前mysql是否支持profile: select @@have\_profiling

* 默认profiling是关闭的，可以通过set语句在Session级别开始

set profiling = 1;

* show profiles 查看每一个操作的耗时
* 通过 show profile for query query\_id语句可以查看该sql执行过程中每个线程的状态和消耗时间

trace分析优化器执行计划

通过trace文件能够进一步了解为什么优化器选择A计划，而不是B计划

打开trace: set optimizer\_trace=”enabled=on”,end\_markers\_in\_json=on;

set optimizer\_trace\_max\_mem\_size=100000;

执行sql语句 select \* from dept2 where deptno < 50;

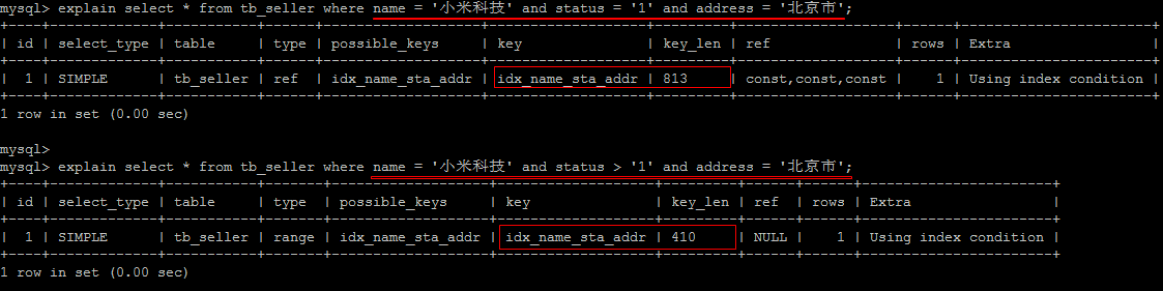
select \* from information\_schema.OPTIMIZER\_TRACE;

**索引的使用：**

索引是数据库优化最常用也是最重要的手段之一

1. 复合索引的情况下，全值匹配，对所有列都指定具体值
2. 最左前缀法则：如果索引了多列，要遵守最左前缀法则，指定是查询从索引的最左前列开始，并且不跳过索引中列
3. 范围查询，右边的列，索引失效，不能使用索引

根据前面的两个字段name ， status 查询是走索引的， 但是最后一个条件address 没有用到索引。



1. 不要再索引列上进行运算操作，索引将失效
2. 字符串不加单引号，造成索引失效
3. 尽量使用覆盖索引，避免select \*

覆盖索引：索引列完全包含查询列

注：using index: 使用覆盖索引的时候就会出现

using where:再查找使用索引的情况下，需要回表查询所需的数据

using index condition:查找使用了索引，但是需要回表查询数据

using index; using where 查找使用了索引，但是需要的数据都在索引列能找到，所以不需要回表查询

1. 用or分割开的条件，如果or前的条件中的列有索引，而后面的列中没有索引，那么涉及的索引都不会被使用到。
2. 以%开头的like模糊查询，索引失效

如果仅仅是尾部模糊匹配，索引不会失效。如果头部模糊匹配，索引失效

解决方案：通过覆盖索引来解决：根据主键来查也走索引

1. 如果mysql评估使用索引比全表更慢，则不使用索引：比如查询的内部的数据在表中占一大部分，则不如全表扫描
2. is null, is not null 有时索引失效，具体看null 和 not null在数据库占的量。
3. in 走索引，not in索引失效
4. 单列索引和复合索引

尽量使用复合索引，而少使用单列索引

如果是几个单列索引，数据库会选择一个最优的索引（辨识度最高的索引）来使用，并不会使用全部索引

查看索引使用情况

show status like ‘Handler\_read%’;

show global status like ‘Handler\_read%’;

**大批量插入数据优化：**

1. 主键顺序插入：

因为InnoDB类型的表是按照主键的顺序保存的，所以将导入的数据按照主键的顺序排列，可以有效提高导入数据的效率，如果InnoDB没有主键，那么系统会默认创建一个内部列作为主键，所以如果可以给表创建一个主键，将可以利用这点，来提高导入数据的效率。

1. 关闭唯一性校验

在导入数据前执行：set unique\_checks=0,关闭唯一性校验，在导入结束后执行set unique\_checks=1，恢复唯一性校验，可以提高导入的效率

1. 手动提交事务

如果应用使用自动提交的方式，建议在导入前执行set autocommit = 0 关闭自动提交，导入结束后再失效 set autocommit=1 打开自动提交，也可以提高导入的效率

1. 如果需要同时对一张表插入很多行数据时，应该尽量使用多个值表的insert语句，这种方式将大大的缩减客户端和数据库之间的连接，关闭等消耗。使得效率比分开执行的单个insert语句快。

原始方式：insert into tb\_test values(1,’Tom’);

insert into tb\_test values(3,’Cat’);

insert into tb\_test values(2,’Jerry’);

优化后的方案为：

insert into tb\_test values(2,’Jerry’),(2,’Tom’),(3,’Cat’)

1. 在事务中进行数据插入

start transation;

insert into tb\_test values(1,’Tom’);

insert into tb\_test values(3,’Cat’);

nsert into tb\_test values(2,’Jerry’);

commit;

优化order by语句

两种排序方式

第一种是通过对返回数据进行排序，也是通常说的filesort排序，所有不是通过索引直接返回排序结果的排序都叫FileSort排序

第二种是通过有序索引顺序扫描直接返回有序数据，这种情况即为using index 不需要额外排序，操作效率高 即查询的列都包含在覆盖索引内

多字段排序：通过order by进行多字段排序，要么都是升序，要么都是降序，排序顺序需要和索引顺序对应

总结：了解mysql的排序方式，优化目标就清晰了：尽量减少额外的排序，索引直接返回有序数据。where条件和Order by使用相同的索引。并且order by的顺序和索引顺序相同，并且order by的字段都是升序，或者都是降序，否则肯定需要额外的操作，这样就会出现fileSort

FileSort优化：

两次扫描算法：mysql4.1之前

一次扫描算法：一次性取出满足条件的所有字段，然后再排序区sort buffer 中排序后直接输出结果集，排序时内存开销较大，但是排序效率比两次扫描算法要高

mysql通过比较系统变量max\_length\_for\_sort\_data的大小和query语句取出字段总大小，来判断使用哪种算法，如果max\_length\_for\_sort\_data更大，那么使用第二种优化之后的算法，否则使用第一种

可以适当提高sort\_buffer\_size和max\_length\_for\_sort\_data系统变量，来增大排序区的大小，提高排序的效率

show variables like ‘max\_length\_for\_sort\_data’;

show variables like ‘sort\_buffer\_size’;

**优化group by 语句**

由于group by实际上也同样会进行排序操作，而且与order by 相比，group by主要是多了排序之后的分组操作，与order by一样可以利用到索引

优化1：如果查询包含group by 但是用户想要避免排序结果的消耗，则可以执行order by null 进行排序

优化2：创建索引来提高分组效率

**优化嵌套子查询**

有些情况下，子查询可以被更高效的连接（JOIN）替代

**优化or条件**

建议使用union替换or

我们来比较下重要指标，发现主要差别是type和ref这两项

type显示的是访问类型，是较为重要的一个指标，结果值从好到坏依次是：

* system > const > eq\_ref > ref > fulltext > ref\_or\_null > index\_merge > unique\_subquery > index\_subquery > range > index > ALL
* union语句的type值为ref or语句的type值为range,可以看到这是一个明显的差距
* union语句的ref为const，or语句的type为null, const表示是常量值引用，非常快

这两项的差距就说明了union要优于or

优化分页查询：

一般分页查询时，通过创建覆盖索引能够比较好的提高性能，一个常见又非常头疼的问题是limit 2000000,10此时需要Mysql排序前200010记录，仅仅返回200000-200010的记录，查询排序的代价非常大

优化思路1：再索引上完成排序分页操作，最后根据主键关联会原表查询所需要的其他列内容

select \* from tb\_item t,(select id from tb\_item order by id limit 200000,10) a where t.id = a.id;

优化思路2：该方案适用于主键自增的表，可以把limit查询转换成某个位置的查询

select \* from tb\_item where id > 100000 limit 10;

其中主键不能断层

使用sql提示

use index:再查询语句中表后面，添加use index来提供希望Mysql去参考的索引列表就可以让Mysql不再考虑其他可用的索引

select \* from tb\_item use index(使用的索引) where name=’’

ignore index:忽略一个或多个索引

force index: 为强制使用一个特定的索引，

### 4、应用优化

1、 使用数据库连接池：

对于访问数据库来说，建立连接的代价是比较昂贵的。因为我们频繁的创建关闭连接，是比较耗费资源的，我们有必要建立连接，数据库连接池，以提高访问性能

1. 减少对mysql的访问：避免对数据进行重复检索或者增加cache层 或者使用框架（mybatis）redis数据库来缓存数据
2. 负载均衡

负载均衡是应用中使用非常普遍的一种优化方法，它的机制就是利用某种均衡算法，将固定的负载量分布到不同的服务器上，以此来降低单台服务器的负载，达到优化效果

利用mysql复制分流查询：

通过mysql的主从复制，实现读写分离，使增删改操作走主节点，查询走从节点，从而降低单台mysql服务器的读写压力

采用分布式数据库框架

分布式数据框架适合大数据量，负载高的情况，它有良好的扩展性和高可用性。通过再多台服务器之间分布数据，可以实现在多台服务器之间的负载均衡，提高访问效率

**mysql中查询缓存的优化**：

开启mysql查询缓存当执行完全相同的sql语句的时候，服务器就会直接从缓存中读取结果，当数据被修改，之前的缓存会失效

* 查看当前的mysql数据库是否支持查询缓存：

show variables like ‘have\_query\_cache’;

* 查看当前mysql是否开启了查询缓存：

show variables like ‘query\_cache\_type’;

* 查询缓存的占用大小：

show variables like ‘query\_cache\_size’;

* 查看查询缓存的状态变量

show status like ‘Qcache%’

* 开启查询缓存:

Query\_cache\_type可以是0,1,2，0代表不使用缓存，1代表使用缓存，2代表根据需要使用.

demand 或 2 查询缓存功能按需进行，显示指定SQL\_CACHE的select语句才会缓存

在/usr/my.cnf配置中增加以下配置

query\_cache\_type=1

* 查询缓存select选项

SQL\_CAHCE：如果查询结果是可缓存的,并且qeury\_cache\_type系统变量的值为on或demand,则缓存查询结果

SQL\_NO\_CACHE：服务器不使用查询缓存，它即不检查查询缓存，也不检查结果是否已缓存，也不缓存结果

* 查询缓存失效的情况

1. sql语句不一致，要想命中查询缓存，查询的sql语句必须一致

select count(\*) from tb\_item

Select count(\*) from tb\_item

1. 当查询语句中有一些不确定的条件时，则不缓存 如：now() rand() database()
2. 不使用任何表查询语句

select ‘A;

1. 查询mysql, information\_schema 或 performance\_schema数据库中的表时，即系统数据库的内容，不会走查询缓存
2. 在存储的函数 触发器或事件的主体内执行的查询
3. 如果表更改，则使用该表的所有高速缓存查询将变为无效并从高速缓存中删除

mysql内存管理及优化

内存优化原则：

将尽量多的内存分配给mysql做缓存，但要给操作系统和其他程序预留足够的内存

排序区、连接区等缓存是分配给每个数据库会话专用的，其默认的设置要根据最大连接数合理分配，如果设置太大，不但浪费资源，而且在并发连接较高时会导致物理内存耗尽

InnoDB内存优化

InnoDB用一块内存区做IO缓存池，该缓存池不仅用来缓存InnoDB的索引块，而且也用来缓存InnoDB的数据块

InnoDB\_buffer\_pool\_size:

该变量决定了innodb存储引擎数据和索引数据的最大缓存区大小，在保证操作系统及其他程序有足够内存可用的情况下，innodb\_buffer\_pool\_size的值越大，缓存命中率就越高，访问InnoDB表需要的磁盘I/O就越少性能就越高

innodb\_log\_buffer\_size:决定了innodb重做日志缓存的大小

**mysql并发参数的调整**

max\_connections:

控制允许连接到mysql数据库的最大数量，默认值是151,如果状态量connection\_errors\_max\_connections不为零，并且一直增长，则说明不断有连接请求因数据库连接数以达到最大值而失败，这时可以考虑增大max\_connecions的值

back\_log

back\_log参数控制mysql监听TCP端口设置的积压请求栈大小，入宫mysql的连接数达到max\_connection,新的请求将会被放到堆栈中，以等待某一连接释放资源，该堆栈数量即为back\_log.如果等待连接数量超过Bxk\_log,j将不予授予连接资源

table\_open\_cache

该参数用来控制所有sql语句执行线程可打开表缓存的数量，每一个sql执行线程至少要打开一张表

thread\_cache\_size

为加快连接数据库的速度，Mysql会缓存一定数量的客户服务线程以备宠用

innodb\_lock\_wait\_timeout

**MySQL锁问题：**

锁概述：锁是计算机协调多个进程并发访问的某一资源的机制

在数据库中，除传统的计算机资源（如cpu ram i/o）的争用以外，数据也是一种供许多用户共享的资源。如何保证数据并发访问的一致性，有效性是所用数据库必须解决的一个问题，锁冲突也是影响数据库并发访问性的一个重要因素。从这个角度来说，锁对数据库而显得尤其重要，也更加复杂

锁分类：

从数据操作的粒度分：

表锁：操作时会锁定整个表

行锁：操纵时会锁定当前操作行

从对数据操作的类型分：

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（排他锁）：当前操作没有完成之前，它会阻断其他写锁和读锁

表级锁：偏向MyISAM存储引擎，开销小，加锁快，不会出现死锁，锁的粒度大，发生锁冲突的概率最高，并发度最低。

行级锁：偏向InnoDB存储引擎，开销大，加锁慢，会出现死锁，锁定粒度小，发生锁内概率最低，并发度也是最高

InnoDB 支持表级锁和行级锁

MyISAM

* MyISAM表 读锁会阻塞写，但是不会阻塞读。而写锁，则即会阻塞读，又会阻塞写
* 查看锁的争用情况：show open tables;
* 查看表的锁定情况：show status like ‘Table\_locks%’

table\_locks\_immediate指的是能够立即获得表级锁的次数，每立即获取锁，值加1

table\_locks\_waited: 指的是不能立即获得表级锁而需要等待的次数，每等待一次，该值加1

InnoDB（默认）行锁

行锁特点：偏向InnoDB存储引擎，开销大，加锁慢，会出现死锁，锁定粒度小，发生锁内概率最低，并发度也是最高

InnoDB与MyISAM的最大不同点：一是支持事务。而是采用行级锁

mysql的数据库的默认隔离级别为repeatable read

查看方式：show variable like ‘tx\_isolation’;

* 无索引行锁升级为表锁：

如果不通过索引条件来检索数据，那么InnoDB将对表中的所有记录加锁，实际效果跟表锁一样

间隙锁：当我们范围条件内，检索数据，并请求共享锁时，InnoDB会给符合条件的已有数据加锁，对于范围内不存在的记录 叫做间隙，也会加锁

Innodb行锁争用情况

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

总结：

尽可能让所有数据检索都能通过索引来完成，避免无索引行锁升级为表锁

合理涉及索引，尽量缩小锁的范围

尽可能减少索引条件，及索引范围，避免间隙锁

尽量控制事务大小，减少锁定资源量和时间长度

尽可能使用低级别事务隔离（但是需要业务层面满足需求）

Mysql中常用工具：

连接选项

mysql -h 127.0.0.1 -P 3306 -u root -p

myql -h127.0.0.1 -P3306 -uroot -proot

执行选项：

mysql -uroot -proot db01 -e “select \* from tb\_book”

mysqladmin

mysqladmin是一个执行管理操作的客户端程序

mysqladmin --help 指令查看帮助文档

mysqladmin -uroot -proot create ‘database’;

mysqlbinlog:

mysqlbinlog：由于服务器生成的二进制文件以二进制保存，所以如果想要检查这些文本的文本格式，就会使用到

mysqlbinlog options log-files log-files2

mysqldump:

客户端工具用来备份数据库或在不同数据库之间进行数据迁移。备份内容包含创建表，以及插入表的sql语句

mysqldump [options] db\_name [tables]

参数：

--add-drop-databse: 在数据库创建语句加上 Drop databse语句

--add-drop-table 在每个表创建前加上Drop table语句，默认开启，不开启，--skip -add-drop-table

-n, --no-create-db 不包含数据库创建语句

-t, --no-create-info 不包含数据表的创建语句

-d –no-data 不包含数据

-T, --tab=name 自动生成两个文件，一个.sql文件，创建表结构的语句

一个.txt文件，数据文件

示例： mysqldump -uroot -proot db01 tb\_book –add-drop-database –add-drop-table > tb\_book.sql

mysqldump -uroot -proot -T /tmp db01 tb\_book

mysqlimport/source

mysqlimport/source是客户端数据导入工具，用来导入mysqldump加-T参数后导出的文本文件

mysqlshow:

客户端对象查找工具，用来很快查找存在哪些数据库、数据库中的表、表中的列或索引。

查找数据库的表的数量及表中记录的数量

mysqlshow -uroot -proot - -count

查询test库中每个表的字段数，及行数

mysqlshow -uroot -proot test - - count

mysqlshow -uroot -proot test book- - i

mysql日志：

* 错误日志：错误日志是mysql中最重要的日志之一，它记录了mysqld启动和停止时，以及服务器在运行过程中发生任何严重错误时的相关信息，当数据库出现任何故障导致无法正常使用时，可以首先查看此日志，默认开启的

查看日志指令：show variables like ‘log\_error%’;

* 二进制日志：

二进制日志记录所有的DDL语句和DML语句，但是不包括数据库查询语句。此日志对于灾难时的数据恢复起着极其重要的作用，mysql的主从复制，就是通过该binlog实现的

二进制日志，默认情况下是没有开启的，需要到mysql的配置文件中开启，并配置mysql日志格式

配置文件在my.cnf

配置开启: log.bin=mysqlbin

配置二进制日志的格式

binlog\_format=statement

日志格式：

statement: 该日志格式在日志文件中记录的都是sql语句（satement）每一条对数据进行修改的sql都会记录在日志文件中，通过mysql提供的mysqlbinlog工具，可以清晰的查看到每条语句的文本。主从复制的时候，从库（slave）会将日志解析为原文本，并在从库重新执行一次

row:

该日志格式在日志文件中记录的是每一行数据变更，而不是sql语句

mixed:

这是目前mysql默认的日志格式，即混合了statement和row两种格式。默认情况下采用statement，但是在一些特殊情况下采用row来进行记录。mixed格式尽量利用两种模式的优点，而避开他们的缺点

mysqlbin.index:该文件是日志索引，记录日志的文件名

mysqlbing.000001:日志文件

查看日志内容

mysqlbinlog mysqlbing.000001;

如果日志格式是row 直接查看数据是看不懂的，需要这样查

mysqlbinlog -vv mysqlbin.000002;

日志删除：

方式一：Reset Master

方式二：purge master logs to ‘mysqlbin.\*\*\*\*\*\*’该命令将删除\*\*\*\*\*\*编号之前的所有日志

方式三：执行指令 purge master logs before ‘yyyy-mm-dd hh24:mi:ss’,该命令删除之前的所有日志

方式四：设置参数 --explain\_log\_days=# 此参数的含义是设置日志过期天数

* 查询日志：

查询日志中记录了客户端的所有操作语句，而二进制不包含查询数据的SQL语句

默认情况下，查询日志的未开启的，如果需要开启查询日志，可以设置以下配置：

general\_log=1 0表示关闭

设置日志文件名，如果没有指定 默认的文件名为 host\_name.log

general\_log\_file=file\_name

* 慢查询日志：

慢查询日志记录所有执行时间超过参数long\_query\_time设置值并且扫描记录数不小于min\_examined\_row\_limit的所有的sql语句的日志 long\_query\_time 默认为10秒 最小为0秒，精度可以到微秒

慢查询日志默认是关闭的。可以通过两个参数来控制查询日志

slow\_query\_log=1

指定慢查询日志文件名

slow\_query\_log\_file=slow\_query.log

该选项用来配置查询的时间限制，超过这个时间将认为慢查询，将需要进行日志记录，默认10秒

long\_query\_time=10

tail -f slow\_query.log 实时追加尾部的内容

mysqldumpslow slow\_query.log 查看慢查询日志

### mysql复制

复制是指将主数据的DDL和DML操作通过二进制日志传到从库服务器中，然后从库上对这些日志重新执行，从而使得从库和主库的数据保持同步

mysql支持一台主库同时向多台从库进行复制，从库同时也可以作为其他从服务器的主库，实现链状复制

复制分为三步

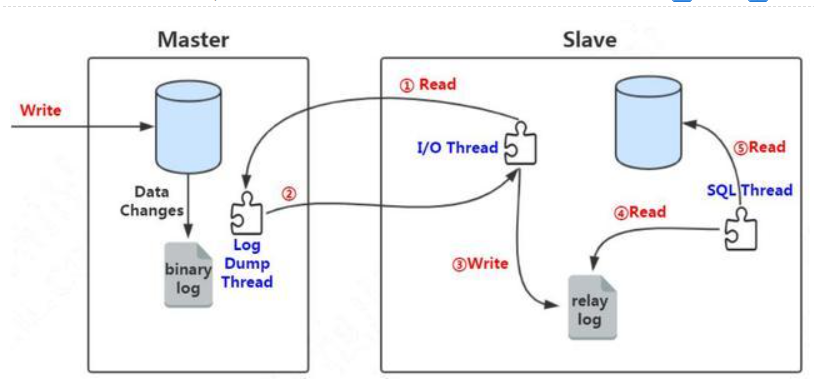
Master主库在提交事务时会把数据变更作为Events记录在二进制文件binlog中

主库推送二进制日志文件binlog的日志时间到从库的中继日志

Relay log

Salave重做中继日志事件，将改变反映它自己的数据

主从复制是怎么实现的呢?更新语句会记录 binlog，它是一种逻辑日志。有了这个 binlog，从服务器会获取主服务器的 binlog 文件，然后解析里面的 SQL 语句，在从服务器上面执行一遍，保持主从的数据一致。



主从复制是怎么实现的呢?更新语句会记录 binlog，它是一种逻辑日志。有了这个 binlog，从服务器会获取主服务器的 binlog 文件，然后解析里面的 SQL 语句，在从服务器上面执行一遍，保持主从的数据一致。

这里面涉及到三个线程，连接到 master 获取 binlog，并且解析 binlog 写入中继日 志，这个线程叫做 I/O 线程。Master 节点上有一个 log dump 线程，是用来发送 binlog 给 slave 的。从库的 SQL 线程，是用来读取 relay log，把数据写入到数据库的。

做了主从复制的方案之后，我们只把数据写入 master 节点，而读的请求可以分担到 slave 节点。我们把这种方案叫做读写分离。

读写分离可以一定程度地减轻数据库服务器的访问压力，但是需要特别注意主从数 据一致性的问题。如果我们在 master 写入了，马上到 slave 查询，而这个时候 slave 的 数据还没有同步过来，怎么办? 所以，基于主从复制的原理，我们需要弄明白，主从复制到底慢在哪里?

复制优势：

主库出现问题，可以快速切换到从库提供服务

可以在从库中执行查询操作，从库中更新，实现读写分离，降低主库的访问压力

可以在从库中执行备份，以避免备份期间影响主库的服务

主从之间是通过mysql的replication（复制）来保证数据的一致性，相对集群的数据同步方式来讲是异步的，因为异步，所以主从之间复制数据可能会有一点微小的延时，就会出现不一致。

集群最大的优点就是数据实时同步，高可用，每个节点的数据都是同步一致的，不像主从，有时会出现数据不一致，而高可用，任何一个节点宕机都不会影响业务。

但是缺点就是性能，写的性能，每次写操作，都会在所有节点之间进行同步，有失有得，损失了一点性能，保证了高可用和数据一致。

集群是一组相互独立的、通过高速网络互联的计算机，它们构成了一个组，并以单一系统的模式加以管理。一个客户与集群相互作用时，集群像是一个独立的服务器。集群配置是用于提高可用性和可缩放性。

一种基于网络的计算机处理技术，与集中式相对应。由于个人计算机的性能得到极大的提高及其使用的普及，使处理能力分布到网络上的所有计算机成为可能。分布式计算是和集中式计算相对立的概念，分布式计算的数据可以分布在很大区域。

负载均衡是一种方法，常用的是nginx反向代理，其实负载均衡是可以理解为搭建集群的一部分，属于集群服务器的前置系统。比如，一个门户网站，请求流量过高的时候，为了保障服务器的正常稳定运行，由nginx来均摊请求到后台服务器。尽量保障每台服务器承受同等的负载。

集群是解决高可用的，而分布式是解决高性能、高并发的