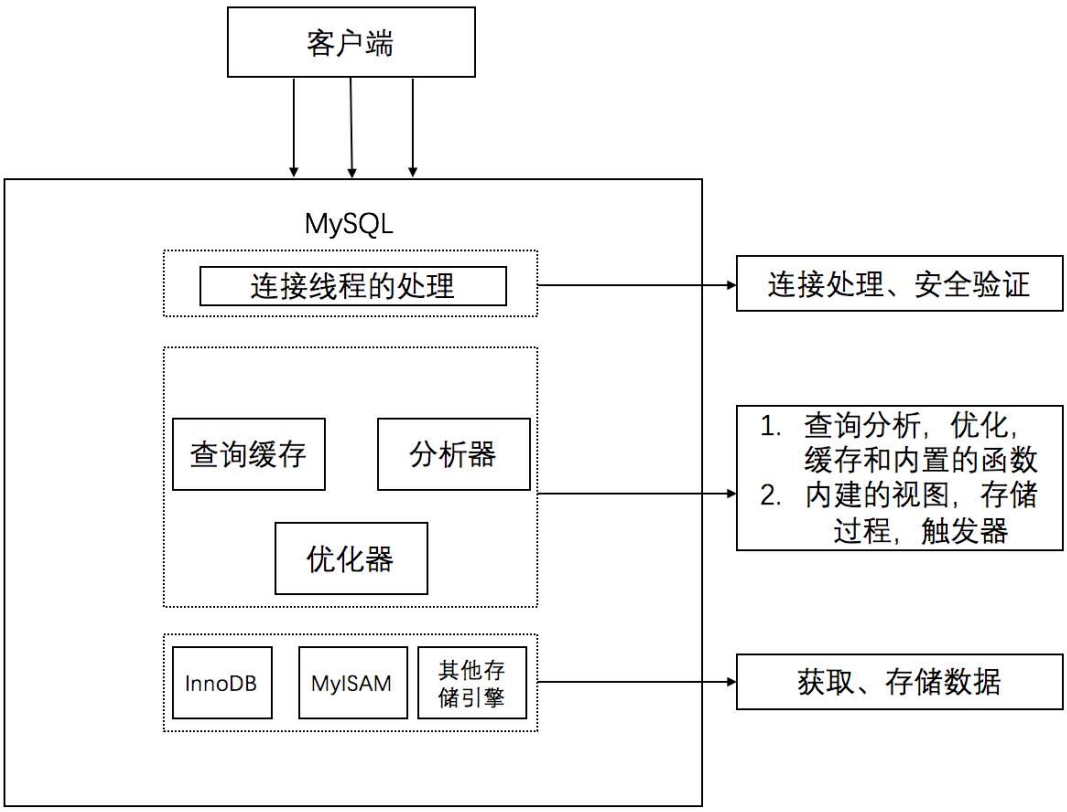
校招面试中常见的 MySQL 考察难点和热点

基本架构

> MySQL是典型的三层架构模式，在平常使用中对MySQL问题排查和优化，也应该针对具体问题，从对应的层解决问题

- 服务层：经典的C/S架构,主要是处理连接和安全验证。
- 核心层：处理MySQL核心业务。
    - 查询分析，优化，缓存和内置函数 。
    - 内建的视图，存储过程，触发器。
- 存储引擎层：存储引擎负责数据的存储和提取。核心层通过存储引擎的 API 与存储引擎通信,来遮蔽不同存储引擎的差异,使得差异对上层透明化。



MySQL调优必备18个参数

连接数 、会话数和线程数

- max_connections

  max_connections参数用来设置最大连接（用户）数。每个连接MySQL的用户均算作一个连接，

  max_connections的默认值为100。

  MySQL无论如何都会为管理员保留一个用于登录的连接，因此实际MySQL最大可连接数为

  max_connections+1。

  max_connections 最大为16384。

  该参数设置过小的话，会出现”Too many connections”错误。

  查看max_connections参数的方法：

```
mysql> show variables like "max_connections";
```


修改最大可连接数的方法有：

```
# 方法一：即时生效，无需重启MySQL，重启失效
1.root登录MySQL：
$ mysql -uroot -p
2.查看当前的Max_connections参数值：
mysql> SELECT @@MAX_CONNECTIONS AS 'Max Connections';
3.设置该参数的值：
mysql> set global max_connections = 200;
# 方法二：需要重启MySQL，重启不失效，修改my.conf
max_connections=200
# 方法三：编译MySQL的时候，设置默认最大连接数
1.打开MySQL源码，进入SQL目录，修改mysqld.cc文件：
{"max_connections", OPT_MAX_CONNECTIONS,
"The number of simultaneous clients allowed.", (gptr*)
&max_connections,
(gptr*) &max_connections, 0, GET_ULONG, REQUIRED_ARG, 100, 1,
16384, 0, 1,
0},
2. 编译三部曲
$ ./configure
$ make
$ make install
```

- max_connect_errors

  max_connect_errors是一个MySQL中与安全有关的计数器值，它负责阻止过多尝试失败的客户端以防止暴力破解密码的情况。max_connect_errors的值与性能并无太大关系。

由于出现某台host连接错误次数等于max_connect_errors ， 日志中会出现类似blocked because of many connection errors 的信息,解决方法如下：

```
1.执行mysqladmin flush-hosts或者重启 MySQL服务，将错误计数器清零
2.my.cnf修改max_connect_errors的值，可以适当大些
```

- thread_concurrency

是在特定场合下才能使用的，这个变量是针对Solaris系统的，如果设置这个变量的话，mysqld就会调用 `thr_setconcurrency()`.这个函数使应用程序给同一时间运行的线程系统提供期望的线程数目。

CPU核数的2倍，比如有一个双核的CPU，那么 `thread_concurrency` 的应该为4；2个双核的cpu， **thread_concurrency**的值应为8。

数据包和缓存

- max_allowed_packet

  这个参数是限制server允许通信的最大数据包大小，有时候可能因为这个参数设置过小，比较大的insert或者update操作会失败，所以参数应该设置大一些

- key_buffer_size

  关键词缓冲区大小，缓存MyISAM索引块 ，决定索引处理速度，读取索引处理。

根据增大 `Key_reads / Uptime` 来优化这个参数查看 `Key_reads / Uptime` 的方法：

```
$ mysqladmin ext -ri10 | grep Key_reads
```

- thread_cache_size

  此参数用来缓存空闲的线程，以至不被销毁，如果线程缓存中有空闲线程，这时候如果建立新连接，MYSQL就会很快的响应连接请求。

使用 show status查看当前mysql连接情况：

```
mysql>SHOW STATUS WHERE Variable_name LIKE '%Thread%';
```

Threads_cached:代表当前此时此刻线程缓存中有多少空闲线程。
Threads_connected:代表当前已建立连接的数量，因为一个连接就需要一个线程，所以也可以看成当前被使用的线程数。
Threads_created:代表从最近一次服务启动，已创建线程的数量。
Threads_running:代表当前激活的（非睡眠状态）线程数。并不是代表正在使用的线程数，有时候连接已建立，但是连接处于sleep状态，这里相对应的线程也是sleep状态。

建议thread_cache_size设置成与threads_connected一样 。

- sort_buffer_size

  每个连接需要使用 buffer 时分配的内存大小，不是越大越好。例 : 1000个连接 , 一个1MB，会占用1GB内存，200WX1MB=20GB

- join_buffer_size

  为了减少参与 join 的 " 被驱动表 " 的读取次数以提高性能 , 需要使用到join buffer来协助完成 join 操作当 join buffer 太小，MySQL 不会将该buffer存入磁盘文件而是先将join buffer中的结果与需求join的表进行操作，然后清空 join buffer 中的数据，继续将剩余的结果集写入次buffer中，如此往复，这势必会造成被驱动表需要被多次读取，成倍增加IO访问，降低效率 。

- query_cache_size

  查询缓存大小，再查询时返回缓存，缓存期间表必须没有被更改，否则缓存失效，多写入操作的话设置大了会影响写入效率。

mySQL用于查询的缓存的内存被分成一个个变长数据块，用来存储类型，大小，数据等信息。 当服务器启动的时候，会初始化缓存需要的内存，是一个完整的空闲块。当查询结果需要缓存的时候，先从空闲块中申请一个数据块大于参数query_cache_min_res_unit的配置，即使缓存数据很小，申请数据块也是这个，因为查询开始返回结果的时候就分配空间，此时无法预知结果多大。

配内存块需要先锁住空间块，所以操作很慢，MySQL会尽量避免这个操作，选择尽可能小的内存块，如果不够，继续申请，如果存储完时有空余则释放多余的。 缓存存放在一个引用表中，通过一个哈希值引用，这个哈希值包括查询本身，数据库，客户端协议的版本等，任何字符上的不同，例如空格，注释都会导致缓存不命中。

当查询中有一些不确定的数据时，是不会缓存的，比方说now(),current_date()，自定义函数，存储函数，用户变量，字查询等。所以这样的查询也就不会命中缓存，但是还会去检测缓存的，因为查询缓存在解析SQL之前，所以MySQL并不知道查询中是否包含该类函数，只是不缓存，自然不会命中。
缓存存放在一个引用表中，通过一个哈希值引用，这个哈希值包括查询本身，数据库，客户端协议的版本等，任何字符上的不同，例如空格，注释都会导致缓存不命中。

- read_buffer_size

  这个参数是MySQL读入缓冲区的大小，将对表进行顺序扫描的请求将分配一个读入缓冲区，mysql会为它分配一段内存缓冲区，read_buffer_size变量控制这一缓冲区的大小，如果对表的顺序扫描非常频繁，并你认为频繁扫描进行的太慢，可以通过增加该变量值以及内存缓冲区大小提高其性能， `read_buffer_size` 变量控制这一提高表的顺序扫描的效率 数据文件顺序。

- read_rnd_buffer_size

  这个参数是MySQL的随机读缓冲区大小，当按任意顺序读取行时（列如按照排序顺序）将分配一个随机读取缓冲区，进行排序查询时，MySQL会首先扫描一遍该缓冲，以避免磁盘搜索，提高查询速度，如果需要大量数据可适当的调整该值，但MySQL会为每个客户连接分配该缓冲区所以尽量适当设置该值，以免内存开销过大。表的随机的顺序缓冲 提高读取的效率。从排序好的数据中读取行时，行数据从缓冲区读取的大小，会提升order by性能 注意：MySQL会为每个客户端申请这个缓冲区，并发过大时，设置过大影响内存开销。

- myisam_sort_buffer_size

  MyISAM表发生变化时，重新排序所需的缓存。

- innodb_buffer_pool_size

  InnoDB 使用缓存保存索引，保存原始数据的缓存大小，可以有效减少读取数据所需的磁盘IO。

日志和事务

- innodb_log_file_size

  数据日志文件大小，大的值可以提高性能，但增加了恢复故障数据库的时间（恢复故障数据库时需要读取数据日志文件，当日志过大会导致时间过长）。

- innodb_log_buffer_size

  日志文件缓存，增大该文件可以提高性能，但增大了忽然宕机后损失数据的风险（日志文件在缓存中，还没来得及存进硬盘就断电了）。

- innodb_flush_log_at_trx_commit

  执行事务的时候，会往InnoDB存储引擎的日志缓存插入事务日志，写数据前先写日志（预写日志方式）设置为0,实时写入；当设置为1时，缓存实时写入磁盘；2时，缓存实时写入文件，每秒文件实时写入磁盘。

- innodb_lock_wait_timeout

回滚前（当一个事务被撤销时），一个InnoDB事务，应该等待一个锁被批准多久，当InnoDB无法检测死锁时，这个值就有用了。

软件优化

1. 选择合适的引擎

MyISAM 索引顺序访问方法，支持全文索引，非事务安全，不支持外键，会加表级锁存在三个文件：

- FRM 存放表结构
- MYD 存放数据
- MYI 存放索引

InnoDB 事务型存储引擎，加行锁，支持回滚，崩溃恢复，ACID事务控制，表和索引放在一个表空间里头，表空间多个文件。

例：

```
update tableset age=3 where name like "%jeff%";
//会锁表
```


2. 正确使用索引

给合适的列表建立索引，给where子句，连接子句建立索引，而不是select选择列表索引值应该不相同，唯一值时效果最好，大量重复效果很差使用短索引，指定前缀长度char(50)的前20，30值唯一例。文件名索引缓存一定（小）时，存的索引多，消耗IO更小，能提高查找速度最左前缀n列索引，最左列的值匹配，更快。

like查询，索引会失效，尽量少用like。百万、千万数据时，用like Sphinx开源方案结合MySQL不能滥用索引。

- 索引占用空间。
- 更新数据，索引必须更新，时间长，尽量不要用在长期不用的字段上建立索引。
- SQL执行一个查询语句，增加查询优化的时间。

3. 避免使用SELECT

- 返回结果过多，降低查询的速度。

- 过多的返回结果，会增大服务器返回给APP端的数据传输量。例：网络传输速度面，弱网络环境下，容易造成请求失效。

4. 字段尽量设置为NOT NULL

> “” 和 NULL问题
>
> {“name”:”myf”} {“name”:””} {“hobby”:空array}
>
> NULL占空间
>
> 例：安卓需要判断””还是NULL
>
> Java和OC都是强类型，会造成APP闪退

硬件优化

1. Linux内核用内存开缓存存放数据

写文件：文件延迟写入机制，先把文件存放到缓存，达到一定程度写进硬盘。

读文件：同时读文件到缓存，下次需要相同文件直接从缓存中取，而不是从硬盘取。

2. 增加应用缓存

本地缓存：数据防盗服务器内存的文件中。

分布式缓存：Redis, Mencache 读写性能非常高，QPS（每秒查询请求数）每秒达到1W以上；数据持久化用Redis，不持久化两者都可以。

3. 用SSD代替机械硬盘

日志和数据分开存储，日志顺序读写 - 机械硬盘，数据随机读写 - SSD

可以调参数

```
#操作系统禁用缓存，直接通过fsync方式将数据刷入机械硬盘
innodb_flush_method = O_DIRECT

# 控制MySQL中一次刷新脏页的数量，SSD io 增强，增大一次输入脏页的数量
innodb_in_capacity = 1000
```

4. SSD+SATA混合存储，FlashCache: Facebook开源在文件系统和设备驱动之间加了一层缓存，对热数据缓存

架构优化

1. 分表

水平拆分：数据分成多个表拆分后的每张表的表头相同。

垂直拆分：字段分成多个表。

插入数据、更新数据、删除数据、查询数据时：MyISAM MERGE存储引擎，多个表合成一个表，InnoDB用alter table，变成MyISAM存储引擎，然后MEGRE。

面试题：MERGE存储引擎将N个表合并，数据库中如何存储？答： 真实存储为N个表，表更大的话就需要分库了。

2. 读写分离

读是一些机器，写是一些机器，二进制文件的主从复制，延迟解决方案。数据库压力大了，可以把读和写拆开，对应主从服务器，主服务器写操作、从服务器是读操作。大多数业务是读业务。京东、淘宝大量浏览商品、挑选商品是读操作（多），购买是写操作（少）。主服务器写操作的同时，同步到从服务器，保持数据完整性——主从复制。

主从复制原理：基于主服务器的二进制日志（binlog）跟踪所有的对数据库的完整更改实现要实现主从复制，必须在主服务器上启动二进制日志主从复制是异步复制。

三个线程参与：主服务器一个线程（IO线程）、从服务器两个（IO线程和SQL线程）

主从复制过程：

- 从数据库，执行start slave开启主从复制。
- 从数据库IO线程会通过主数据库授权的用户请求连接主数据库，并请求主数据库的binlog日志的指定位置，change master命令指定日志文件位置。
- 主数据库收到IO请求，负责复制的IO线程跟据请求读取的指定binlog文件返回给从数据库的IO线程，返回的信息除了日志文件，还有本次返回的日志内容在binlog文件名称和位置。
- 从数据库获取的内容和位置（binlog），写入到（从数据库）relaylog中继日志的最末端，并将新的binlog文件名和位置记录到master-info文件，方便下次读取主数据库的binlog日志，指定位置，方便定位。
- 从数据库SQL线程，实时检测本地relaylog新增内容，解析为SQL语句，执行。

弊端：延迟

主从复制延迟解决方案：

- 定位问题：延迟瓶颈，IO压力大，升级硬件，换成SSD
- 单线程从relaylog执行MySQL语句延迟，换成MySQL5.6以上版本多线程，或者Tungsten第三方并行复制工具
- 都不行，直接分库

3. 分库

Cobar方案：阿里开源（后续无更新）

MyCat基于Cobar，MySQL通讯协议，代理服务器，无状态，容易部署，负载均衡。

原理：应用服务器传SQL语句，路由解析，转发到不同的后台数据库，结果汇总，返回
MyCat把逻辑数据库和数据表对应到物理真实的数据库、数据表，遮蔽了物理差异性。

MyCat工作流程：

- 应用服务器向MyCat发送SQL语句select * from user where id in(30, 31, 32)。
- MyCat前端通信模块与应用服务器通信，交给SQL解析模块。
- SQL解析模块解析完交给SQL路由模块。
- SQL路由模块，id取模，余数为0：db1，余数为1：db2……
- 把SQL拆解为select * from user where id in 30……交给SQL执行模块，对应db1 db2 db3。
- SQL执行模块通过后端，分别在db1 db2 db3执行语句，返回结构到数据集合合并模块，然后返回给应用服务器。

SQL慢查询分析、调参数

慢查询：指执行超过一定时间的SQL查询语句记录到慢查询日志，方便开发人员查看日志

找问题：

```
long_qeury_time定义慢查询时间
slow_query_log设置慢查询开关
slow_query_log_file设置慢查询日志文件路径
```

设置方法1：

```
set log_query_time = 1;
set slow_query_log = on;
set slow_query_log_file = '/data/slow.log'
```

设置方法2：

/etc/my.comf设置参数

分析：

explain命令进行分析，输出结构含义，官方文档。