学习一个服务的过程：

1、 此服务器的概述：名字，功能，特点，端口号

2、 安装

3、 配置文件的位置

4、 服务启动关闭脚本，查看端口

5、 此服务的使用方法

6、 修改配置文件，实战举例

7、 排错（从下到上，从内到外）

本节所讲内容：

• 调优思路：

• 0.硬件优化

• 1.数据库设计与规划--以后再修改很麻烦，估计数据量，使用什么存储引擎

• 2.数据的应用--怎样取数据，sql 语句的优化

• 3.磁盘 io 优化

• 4.操作系统的优化--内核、tcp 连接数量

• 5.mysql 服务优化--内存的使用，磁盘的使用

• 6.my.cnf 内参数的优化：

• 7 分库分表思路和优劣

0.硬件优化

CPU—— 64 位、高主频、高缓存，高并行处理能力

内存——大内存、主频高，尽量不要用 SWAP

硬盘——15000转、RAID5、raid10 。 SSD

网络——标配的千兆网卡，10G网卡，bond0，msyql服务器尽可能和使用它的web服务器在同一局域网内，尽量避免诸如防火墙策略等不必要的开销。

1.数据库设计与规划（架构上的优化)

纵向拆解: 专机专用

例：现在公司一台服务器同时负责 web、ftp、数据库等多个角色。 R720 dell 内存 ：768G

纵向拆解后就是：数据库服务器专机专用，避免额外的服务可能导致的性能下降和不稳定性。



横向拆解: 主从同步、负载均衡、高可用性集群，当单个 mysql 数据库无法满足日益增加的需求时，可以考虑在数据库这个逻辑层面增加多台服务器，以达到稳定、高效的效果。

2、查询优化 如果你想玩dba

a>建表时表结构要合理，每个表不宜过大；在任何情况下均应使用最精确的类型。例如，如果ID列用int是一个好主意，而用text类型则是个蠢办法；TIME列酌情使用DATE或者DATETIME。

b>索引，建立合适的索引。

c>查询时尽量减少逻辑运算（与运算、或运算、大于小于某值的运算）；

d>减少不当的查询语句，不要查询应用中不需要的列，比如说select \* from 等操作。

e>减小事务包的大小；

f>将多个小的查询适当合并成一个大的查询，减少每次建立/关闭查询时的开销；

g>将某些过于复杂的查询拆解成多个小查询，和上一条恰好相反

h>建立和优化存储过程来代替大量的外部程序交互。

3，磁盘 io 规划，io相关的技术

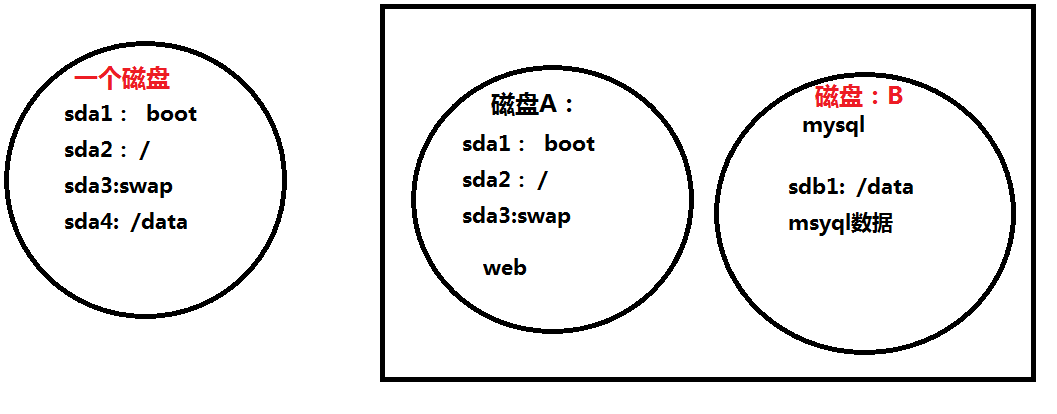
raid 技术：raid0或raid10

SSD

15000转、RAID5、raid10 。 SSD

swap 分区：最好使用 raid0 或 SSD

磁盘分区：将数据库目录放到一个分区上或一个磁盘上的物理分区. 存储数据的硬盘或分区和系统所在的硬盘分开。

****

设置主从时，由于binlog日志频繁记录操作，开销非常大，需要把binlog日志放到其它硬盘分区上：

#vim /etc/my.cnf

[mysqld]

datadir=/data/ 放在独立的硬盘上SSD

socket=/var/lib/mysql/mysql.sock

user=mysql

# Disabling symbolic-links is recommended to prevent assorted security risks

symbolic-links=0 #在原配置文件中，添加以下内容：

log-bin=/data/mysqllog #启用二进制日志，默认存在/var/lib/mysql 下面

server-id=1 #本机数据库ID 标示。

binlog-do-db=db #可以被从服务器复制的库。二进制需要同步的数据库名

4.操作系统的优化

网卡 bonding 技术

设置tcp 连接数量限制，优化系统打开文件的最大限制。

使用64位操作系统，64位系统可以分给单个进程更多的内存。 计算更快 。

禁用不必要启动的服务

文件系统调优，给数据仓库一个单独的文件系统，推荐使用XFS，一般效率更高、更可靠。

ext3 不错。 ext4 只是一个过渡的文件系统。

可以考虑在挂载分区时启用 noatime 选项。 #不记录访问时间

最小化原则:

1） 安装系统最小化

2） 开启程序服务最小化原则

3） 操作最小化原则

4） 登录最小化原则

5） 权限最小化

例：关文件系统atime选项：

[root@xuegod63 ~]# vim /etc/fstab #在挂载项中添加noatime选项就可以了。

UUID=46cb104c-e4dc-4f84-8afc-552f21279c65 /boot

t4 defaults,noatime 1 2

[root@xuegod63 ~]# mount #查看添加前mount挂载选项

/dev/sda1 on /boot type ext4 (rw) ：

使设置立即生效，可运行：

[root@xuegod63 ~]# mount -o remount /boot/

[root@xuegod63 ~]# mount

。。。

/dev/sda1 on /boot type ext4 (rw,noatime)

这样以后系统在读此分区下的文件时，将不会再修改atime属性。

说明：测试效果，结果没有太大的意义。

5. mysql 服务优化（数据库服务的优化)

保持每个表都不要太大，可以对大表做横切和纵切：比如说我要取得某 ID 的 lastlogin， 完全可以做一张只有“ID和 “lastlog”的小表，而非几十、几百列数据的并排大表。

另外对一个有 1000 万条记录的表做更新比对 10 个 100 万记录的表做更新一般来的要慢。

折线图

存储引擎：

myisam 引擎，表级锁，表级锁开销小，影响范围大，适合读多写少的表，不支持事务。 表锁定不存在死锁 （也有例外）

innodb 引擎，行级锁，锁定行的开销要比锁定全表要大。影响范围小，适合写操作比较频繁的数据表。行级锁可能存在死锁。

MySQL的锁机制比较简单，其最显著的特点是不同的存储引擎支持不同的锁机制。

详解死锁<https://www.douban.com/note/344647957/?type=like>

http://www.jb51.net/article/100689.htm

之所以出现死锁，是因为多线程对资源的强占，你要我的，我也要你的，两人堵在路上谁都不让，所以死锁了。

#开启后会将所有的死锁记录到error\_log中 错误日志在my.cnf配置为log-error=/var/log/mysqld.log

innodb\_print\_all\_deadlocks = 1

innodb\_sort\_buffer\_size = 16M

查看数据库服务的状态，登录mysql

mysql> show status; 看系统的状态

mysql> show engine innodb status \G #显示 InnoDB 存储引擎的状态

mysql> show variables; 看变量，在 my.cnf 配置文件里定义的变量值

例如：

log\_error | /var/log/mysqld.log

查看警告信息：

mysql> show warnings; 查看最近一个 sql 语句产生的错误警告

看其他的错误信息，需要看日志/var/log/mysqld.log。

例：查看警告信息

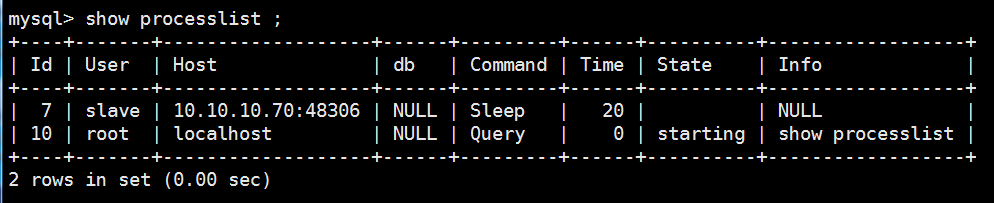
mysql> adadfs; #随便输入一些内容，回车。将看到以下一些错误信息

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'adadfs' at line 1

mysql> show warnings;

mysql> show processlist ; #显示mysql系统中正在运行的所有线程。 可以看到每个客户端正在执行的命令

#本语句TCP/IP连接的主机名称（采用host\_name:client\_port格式），以方便地判定哪个客户端正在做什么。



启用 mysql 慢查询：---分析 sql 语句，找到影响效率的 SQL

[root@xuegod63 ~]# vim /etc/my.cnf

[mysqld]

slow\_query\_log = 1 #开启慢查询日志

slow-query-log-file=/var/lib/mysql/slow.log #这个路径对 mysql 用户具有可写权限

long\_query\_time=5 #查询超过 5 秒钟的语句记录下来

log-queries-not-using-indexes =1 #没有使用索引的查询

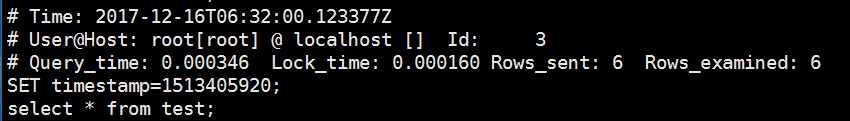
这三个设置一起使用，可以记录执行时间超过5 秒和没有使用索引的查询。请注意有关log-queries-not-using-indexes的警告。慢速查询日志都保存在/var/lib/mysql/slow.log。

Create trable test (id int,name varchar(20));

Insert into test values (1, ‘man’)

Select \* from test;

查看：



6. my.cnf 内参数的优化

优化总原则：给 mysql 的资源太少，则 mysql 施展不开：给 mysql 的资源太多，可能会拖累整个 OS。

40%资源给OS， 60%-70% 给mysql （内存和CPU）

对查询进行缓存

大多数LAMP应用都严重依赖于数据库查询，查询的大致过程如下:

PHP发出查询请求->数据库收到指令对查询语句进行分析->确定如何查询->从磁盘中加载信息->返回结果

如果反复查询，就反复执行这些。MySQL 有一个特性称为查询缓存，他可以将查询的结果保存在内存中，在很多情况下，这会极大地提高性能。不过，问题是查询缓存在默认情况下是禁用的。

启动查询缓存：

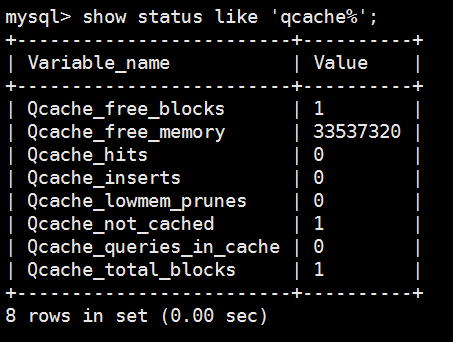
vim /etc/my.cnf 添加：

[mysqld] #在此字段中添加

query\_cache\_size = 32M

查看：查询缓存

Systemctl restart mysqld



解释如下：

变量名 说明

1. Qcache\_free\_blocks 缓存中相邻内存块的个数。数目大说明可能有碎片。

如果数目比较大，可以执行：

mysql> flush query cache;

#对缓存中的碎片进行整理，从而得到一个空闲块。

2. Qcache\_free\_memory 缓存中的空闲内存大小

3. Qcache\_hits 每次查询在缓存中命中时就增大。

4. Qcache\_inserts 每次插入一个查询时就增大。即没有从缓存中找到数据

5. Qcache\_lowmem\_prunes #因内存不足删除缓存次数，缓存出现内存不足并且必须要进行清理,以便为更多查询提供空间的次数。返个数字最好长时间来看；如果返个数字在不断增长，就表示可能碎片非常严重，或者缓存内存很少。

如果Qcache\_free\_blocks比较大，说明碎片严重。 如果 free\_memory 很小，说明缓存不够用了。

6. Qcache\_not\_cached # 没有进行缓存的查询的数量，通常是这些查询未被缓存或其类型不允许被缓存

7. Qcache\_queries\_in\_cache # 在当前缓存的查询（和响应）的数量。

8. Qcache\_total\_blocks #缓存中块的数量。

例：使用mysql查询缓存

[root@xuegod63 ~]# cat /etc/my.cnf

[mysqld]

datadir=/var/lib/mysql

socket=/var/lib/mysql/mysql.sock

user=mysql

# Disabling symbolic-links is recommended to prevent assorted security risks

symbolic-links=0

query\_cache\_size = 32m #至少4M以存储数据结构，可扩展。整体100G，若此服务器只运行mysql服务器。70-80G给mysql

[root@xuegod63 ~]# Systemctl restart mysqld;#重启服务

mysql> create database aa;

mysql> use aa;

mysql> create table test3 (id int, name varchar(255)) ;

mysql> insert into test3 values (1,'aaaa'), (2,'aaaa');

mysql> select \* from test3;

mysql> show status like "qcache%";

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

| Variable\_name | Value |

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

| Qcache\_free\_blocks | 1 |

| Qcache\_free\_memory | 33535112 |

| Qcache\_hits | 0 |#没有命中

| Qcache\_inserts | 1 |#第一次插入一个语句

| Qcache\_lowmem\_prunes | 0 |

| Qcache\_not\_cached | 3 |

| Qcache\_queries\_in\_cache | 1 |

| Qcache\_total\_blocks | 4 |

再查询：

mysql> select \* from test3;

mysql> show status like "qcache%";

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

| Variable\_name | Value |

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

| Qcache\_free\_blocks | 1 |

| Qcache\_free\_memory | 33535112 |

| Qcache\_hits | 1|#第二次查询时，就命中了。

| Qcache\_inserts | 1 |

Qcache\_hits/（Qcache\_inserts+ Qcache\_hits） 命中率

经过多次select，命中的次数也会增加：

mysql> show status like 'qcache%';

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

| Variable\_name | Value |

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

| Qcache\_free\_blocks | 1 |

| Qcache\_free\_memory | 33535344 |

| Qcache\_hits | 3 |#第四次查询时，就命中3次。

| Qcache\_inserts | 1 |

强制限制mysql 资源设置

您可以在mysqld中强制一些限制来确保系统负载不会导致资源耗尽的情况出现。

[root@xuegod63 ~]# vim /etc/my.cnf

[mysqld]

datadir=/var/lib/mysql

socket=/var/lib/mysql/mysql.sock

user=mysql

# Disabling symbolic-links is recommended to prevent assorted security risks

symbolic-links=0

query\_cache\_size = 32M

max\_connections=500 上限是看硬件配置

wait\_timeout=10

max\_connect\_errors = 100

参数：

第一行：最大连接数，在服务器没有崩溃之前确保只建立服务允许数目的连接。

该参数设置过小的最明显特征是出现“Too many connections”错误；

第二行：mysqld将终止等待时间（空闲时间）超过10秒的连接。在LAMP应用程序中，连接数据库的时间通常就是Web 服务器处理请求所花费的时间。有时候如果负载过重，连接会挂起，并且会占用连接表空间。如果有多个交互用户使用了到数据库的持久连接，那么应该将这个值设低一点。

第三行：如果一个主机在连接到服务器时有问题，并重试很多次后放弃，那么这个主机就会被锁定，直到执行：

mysql> FLUSH HOSTS;

Query OK, 0 rows affected (0.00 sec)

之后才能运行。默认情况下，10 次失败就足以导致锁定了。将这个值修改为100 会给服务器足够的时间来从问题中恢复。如果重试100 次都无法建立连接，那么使用再高的值也不会有太多帮助，可能它根本就无法连接。

例：

[root@xuegod63 ~]# cat /etc/my.cnf 在配置文件中添加红色内容：

[mysqld]

datadir=/var/lib/mysql

socket=/var/lib/mysql/mysql.sock

user=mysql

# Disabling symbolic-links is recommended to prevent assorted security risks

symbolic-links=0

query\_cache\_size = 32M

max\_connections=500

wait\_timeout=10

max\_connect\_errors = 100

[root@xuegod63 ~]# service mysqld restart

验证：

mysql> show status like 'max\_used\_connections';

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

| Variable\_name | Value |

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

| Max\_used\_connections | 1 |#当前有一个mysql

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

1 row in set (0.00 sec)

再另一个客户端打开一个mysql连接，执行一下查询，可以看到有两个：

mysql> show status like 'max\_used\_connections';

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

| Variable\_name | Value |

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

| Max\_used\_connections | 2 |

总结：

mysql有超过100个可以调节的设置，要记住那么多基本是不可能的，但是幸运的是你只需要记住很少一部分你就可以基本满足你的需求了，我们还可以通过“SHOW STATUS”命令来查看mysql是否按照我们的期望在运行。

表高速缓存：

数据库中的每个表存储在一个文件中，要读取文件的内容，你必须先打开文件，然后再读取。为了加快从文件中读取数据的过程，mysqld 对这些打开文件进行了缓存，其最大数目由 /etc/my.cnf中的 table\_cache 指定

例如：

[root@xuegod63 ~]# cat /etc/my.cnf

[mysqld]

datadir=/var/lib/mysql

……

table\_open\_cache=23 #缓存23个表

mysql> show global status like 'open%\_tables';

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

| Variable\_name | Value |

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

| Open\_tables | 1 |

| Opened\_tables | 114 |

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

Open\_tables 表示打开表的数量，Opened\_tables表示打开过的表数量，如果Opened\_tables数量过大，说明配置中 table\_cache(5.1.3之后这个值叫做table\_open\_cache)值可能太小.

table\_cache 的值在 2G 内存以下的机器中的值默认从 256 到 512个。

对于有 1G 内存的机器，推荐值是 128-256。

关键字缓冲区

key\_buffer\_size指定索引缓冲区的大小，它决定索引处理的速度，尤其是索引读的速度。

例：关键字缓存 , 缓存来缓存索引

[root@xuegod63 ~]# cat /etc/my.cnf

[mysqld]

datadir=/var/lib/mysql

……

key\_buffer\_size=512M

#只跑一个mysql服务。结合所有缓存，mysql整体使用的缓存可以使用物理内存的80%

[root@xuegod63 ~]# systemctl restart mysqld

查看：

mysql> show status like '%key\_read%';

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

| Variable\_name | Value |

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

| Key\_read\_requests | 0 |

| Key\_reads | 0 |

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

Key\_reads 代表命中磁盘的请求个数，Key\_read\_requests 是总数， 命中磁盘的读请求数除以读请求总数就是不中比率。

命中率：(1-(Key\_reads / Key\_read\_requests ) )\*100

如果每 1,000 个请求中命中磁盘的数目超过 1 个，就应该考虑增大关键字缓冲区了。

例：

mysql> create database mk;

mysql> use mk;

mysql> create table t3 (id int,name varchar(24));

mysql> insert into t3 values (1,'aaaaa'),(2,'aaaaa'),(3,'aaaaa');

mysql> create index idx\_name on t3(name);

mysql> show status like '%key\_read%';

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

| Variable\_name | Value |

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

| Key\_read\_requests | 2|

| Key\_reads | 0 |

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

mysql> select name from t3;

mysql> show status like '%key\_read%';

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

| Variable\_name | Value |

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

| Key\_read\_requests | 4|

| Key\_reads | 1 |

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

总结：

1、看机器配置，指三大件：cpu、内存、硬盘

2、看mysql配置参数

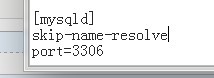
3、查看mysql行状态

4、查看mysql的慢查询

依次解决了以上问题之后，再来查找程序方面的问题

#不进行域名反解析,注意由此带来的权限/授权问题。关闭mysql 的dns 反查功能。这样速度就快了!

skip-name-resolve



该选项就能禁用DNS 解析，连接速度会快很多。不过，这样的话就不能在MySQL 的授权表中使用主机名了而只能用ip 格式。

#索引缓存,根据内存大小而定,如果是独立的db服务器,可以设置高达80%的内存总量

key\_buffer\_size= 512M

#打开表缓存总个数,可以避免频繁的打开数据表产生的开销

table\_open\_cache = 20

query\_cache\_size = 128M

max\_connections=10000# 最大连接数 内存

#设置超时时间,能避免长连接

wait\_timeout=60

#记录慢查询,然后对慢查询一一优化单位：秒

slow-queries- log-file= /var/lib/mysql/slow.log

long\_query\_time = 5

**扩展mysql优化之-------分库分表**

1 基本思想之什么是分库分表？

从字面上简单理解，就是把原本存储于一个库的数据分块存储到多个库上，把原本存储于一个表的数据分块存储到多个表上。

2 基本思想之为什么要分库分表？

数据库中的数据量不一定是可控的，在未进行分库分表的情况下，随着时间和业务的发展，库中的表会越来越多，表中的数据量也会越来越大，相应地，数据操作，增删改查的开销也会越来越大；另外，一台服务器的资源（CPU、磁盘、内存、IO等）是有限的，最终数据库所能承载的数据量、数据处理能力都将遭遇瓶颈,。

3 分库分表的实施策略。

如果你的单机性能很低了，那可以尝试分库。分库，业务透明，在物理实现上分成多个服务器，不同的分库在不同服务器上。分区可以把表分到不同的硬盘上，但不能分配到不同服务器上。一台机器的性能是有限制的，用分库可以解决单台服务器性能不够，或者成本过高问题。

当分区之后，表还是很大，处理不过来，这时候可以用分库。

orderid,userid,ordertime,.....

userid%4=0，用分库1

userid%4=1，用分库2

userid%4=2, 用分库3

userid%4=3，用分库4

上面这个就是一个简单的分库路由，根据userid选择分库，即不同的服务器

4 分库分表存在的问题。

4.1 事务问题。

在执行分库分表之后，由于数据存储到了不同的库上，数据库事务管理出现了困难。如果依赖数据库本身的分布式事务管理功能去执行事务，将付出高昂的性能代价；如果由应用程序去协助控制，形成程序逻辑上的事务，又会造成编程方面的负担。

4.2 跨库跨表的join问题。

在执行了分库分表之后，难以避免会将原本逻辑关联性很强的数据划分到不同的表、不同的库上，这时，表的关联操作将受到限制，我们无法join位于不同分库的表，也无法join分表粒度不同的表，结果原本一次查询能够完成的业务，可能需要多次查询才能完成。

4.3 额外的数据管理负担和数据运算压力。

额外的数据管理负担，最显而易见的就是数据的定位问题和数据的增删改查的重复执行问题，这些都可以通过应用程序解决，但必然引起额外的逻辑运算，例如，对于一个记录用户成绩的用户数据表userTable，业务要求查出成绩最好的100位，在进行分表之前，只需一个order by语句就可以搞定，但是在进行分表之后，将需要n个order by语句，分别查出每一个分表的前100名用户数据，然后再对这些数据进行合并计算，才能得出结果。

Mysql5.7 官方文档

https://dev.mysql.com/doc/refman/5.7/en/