MySQL学习记录

目录

[MySQL安装 3](#_Toc524007269)

[MySQL基础配置 7](#_Toc524007270)

[InnoDB体系架构 8](#_Toc524007271)

[MySQL性能相关配置 9](#_Toc524007272)

[innodb\_buffer\_pool\_size 9](#_Toc524007273)

[innodb\_log\_file\_size 9](#_Toc524007274)

[innodb\_log\_buffer\_size 10](#_Toc524007275)

[innodb\_flush\_log\_at\_trx\_commit 10](#_Toc524007276)

[sync\_binlog 11](#_Toc524007277)

[innodb\_flush\_method 12](#_Toc524007278)

[innodb\_buffer\_pool\_instances 13](#_Toc524007279)

[innodb\_thread\_concurrency 13](#_Toc524007280)

[skip\_name\_resolve 14](#_Toc524007281)

[innodb\_io\_capacity 14](#_Toc524007282)

[innodb\_io\_capacity\_max 15](#_Toc524007283)

# MySQL安装

1. 卸载MySQL

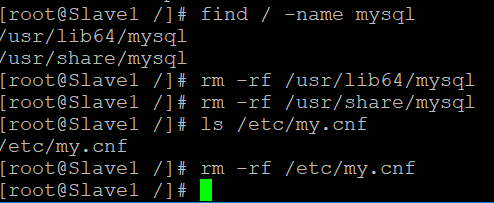
如果系统上有安装过MySQL，先执行此步骤，否则跳到下一步。

find / -name mysql

删除查询到的结果文件

删除/etc/my.cnf

删除/etc/profile 中设置的MySQL的path

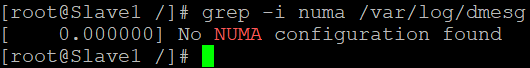


1. 安装包准备

安装好CentOS 7，官网下载安装包：2.7.26版本

1. 检查NUMA

grep -i numa /var/log/dmesg



结果为：[ 0.000000] No NUMA configuration found

否则：关闭NUMA

关闭方法：修改/etc/default/grub文件，在kernel那行追加numa=off;

执行 grub2-mkconfig -o /etc/grub2.cfg 使grub配置生效,重启。

1. 限制/etc/security/limits.conf文件

添加：

mysql hard nproc 65535

mysql soft nproc 65535

mysql hard nofile 65535

mysql soft nofile 65535

或者

\* hard nproc 65535

\* soft nproc 65535

\* hard nofile 65535

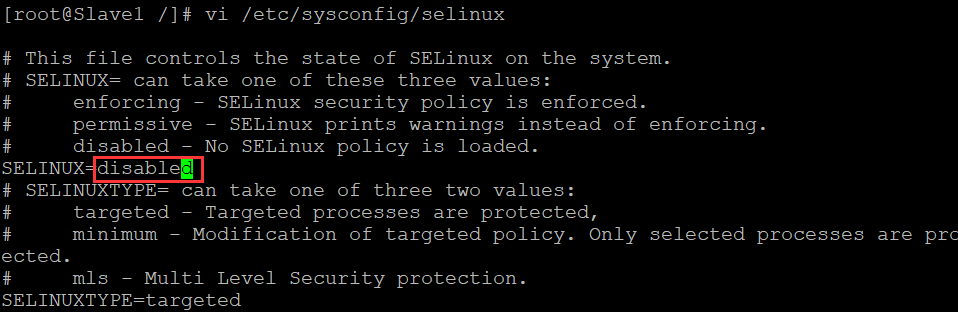
\* soft nofile 65535

\* 代表所有用户

1. 禁用selinux

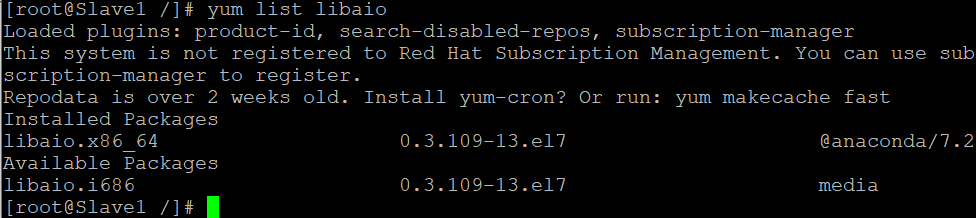
修改/etc/sysconfig/selinux

把SELINUX值修改成disabled，以禁用selinux



1. 安装libaio 库

查看是否已安装：yum list libaio



否则安装：yum install libaio

1. 解压

安装包上传至/usr/local

解压：tar -xzvf mysql-5.7.23-el7-x86\_64.tar.gz

修改文件名：mv mysql-5.7.23-el7-x86\_64 mysql

1. 创建用户mysql

指定home目录/usr/local/mysql ,并且是不能登陆，-M代表不创建home目录groupadd mysql

useradd -M -g mysql -s /sbin/nologin -d /usr/local/mysql mysql

1. 创建软连接

ln命令创建指向安装目录的符号链接，这样可以访问/usr/local/mysql

cd /usr/local

ln -s /usr/local/mysql mysql

1. 设置环境变量

echo "export PATH=$PATH:/usr/local/mysql/bin" >> /etc/profile

使环境变量生效

source /etc/profile

1. 创建基本目录

mkdir -p /data/mysql

cd /data/mysql

mkdir {data,logs,tmp}

chown -R mysql:mysql /data/mysql/

1. 创建/etc/my.cnf

[client]

default-character-set=utf8

socket = /data/mysql/tmp/mysql.sock

[mysqld]

datadir = /data/mysql/data

tmpdir = /data/mysql/tmp

socket = /data/mysql/tmp/mysql.sock

pid-file = /data/mysql/tmp/mysql.pid

log-error = /data/mysql/error.log

character-set-server = utf8

default-storage-engine = INNODB

secure\_file\_priv = /usr/local/mysql/mysql-files/

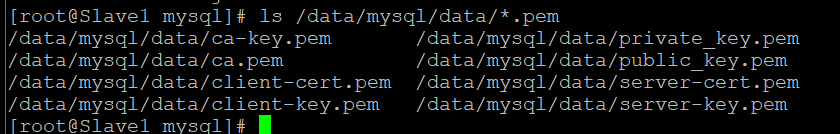
lower\_case\_table\_names = 1

1. 初始化MySQL

/usr/local/mysql/bin/mysqld --defaults-file=/etc/my.cnf --initialize --user=mysql

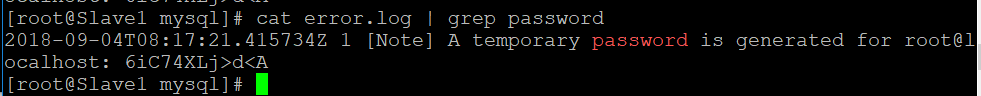
bin/mysql\_ssl\_rsa\_setup --datadir=/data/mysql/data

ls /data/mysql/data 有\*.pem文件



1. 查看临时密码

cat error.log|grep password



1. 创建启动脚本

cd /usr/local/mysql/

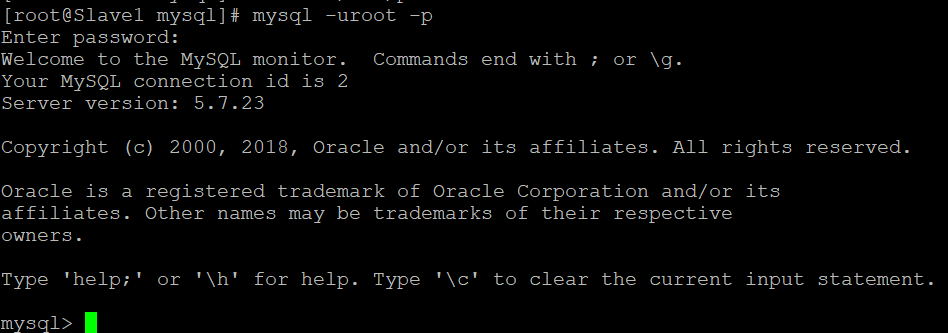
cp support-files/mysql.server /etc/init.d/mysql

1. 启动和关闭

/etc/init.d/mysql start | stop | restart

1. 以临时密码登陆mysql客户端

mysql –uroot –p



1. 更改密码

mysql>alter user user() identified by 'dtauser';

Query OK, 0 rows affected (0.01 sec)

1. 创建dtauser用户

create user dtauser identified by ‘dtauser’;

1. 赋予权限

grant all privileges on \*.\* to 'dtauser'@‘%’

1. 开启远程登录

user mysql

update user set host=’%’ where user=’dtauser’;

1. 错误解决

cd /data/mysql

cat error.log | grep ERROR如有以下：

2018-08-17T02:54:27.307176Z 0 [ERROR] SSL error: Unable to get private key from 'server-key.pem'

则执行：chmod go+r server-key.pem即可

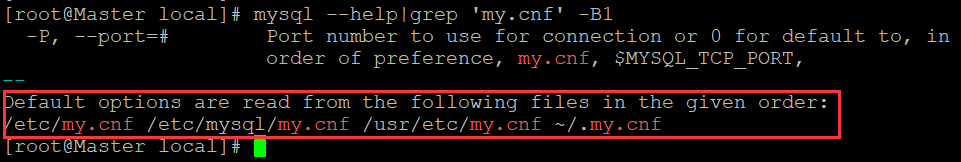
# MySQL基础配置

MySQL配置存放在my.cnf中。

MySQL启动时通过指定--defaults-file参数来指定使用的配置文件，如果没有指定，则读取默认目录下的配置文件。如果默认目录下没有配置文件，则使用默认配置启动MySQL。

可以使用如下命令查看MySQL启动时读取配置文件的默认目录：

mysql --help|grep 'my.cnf' -B1



下面列出部分基础的配置

## client

* default-character-set：编码，如default-character-set = utf8
* socket：用于本地连接的socket套接字，如：socket = /data/mysql/tmp/mysql.sock

## mysqld

* port：指定MySQL监听端口，如，port = 3306
* socket：为MySQL客户端程序和服务器之间的本地通讯指定一个套接字文件，如：socket = /data/mysql/tmp/mysql.sock
* pid-file：pid文件所在目录，如：pid-file = /data/mysql/tmp/mysql.pid
* basedir：指定MySQL根目录（安装目录），如：basedir=/usr/local/mysql
* datadir：指定数据文件存储目录，如：datadir = /data/mysql/data
* tmpdir ：临时文件存储目录，如：tmpdir = /data/mysql/tmp
* character-set-server：服务端默认编码，如：character-set-server=utf8
* collation\_server：服务端默认比对规则，如：collation\_server = utf8\_general\_ci
* user：MySQL启动用户，如：user = mysql
* default-storage-engine：默认存储引擎，如：default-storage-engine = InnoDB
* secure\_file\_priv：MySQL与本地文件交互路劲，如：secure\_file\_priv = /usr/local/mysql/mysql-files/

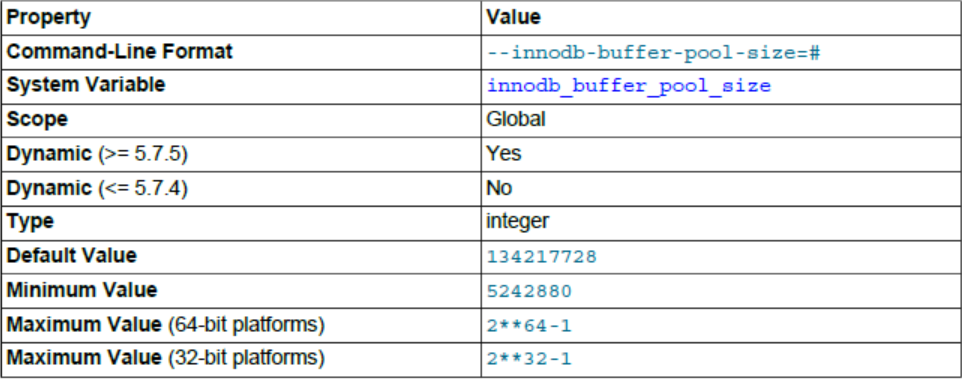
# InnoDB体系架构



# MySQL性能相关配置

下面列举一些与InnoDB相关且影响性能的配置。

## innodb\_buffer\_pool\_size



InnoDB缓冲池大小。

InnoDB缓冲池作用：

* 数据缓存 – 这绝对是它的最重要的目的
* 索引缓存 – 这使用是的同一个缓冲池
* 缓冲 – 更改的数据(通常称为脏数据)在被刷新到硬盘之前先存放到缓冲
* 存储内部结构 – 一些结构如自适应哈希索引或者行锁也都存储在InnoDB缓冲池

较大的缓冲池在获取同样的表中的数据时可减少磁盘I/O。

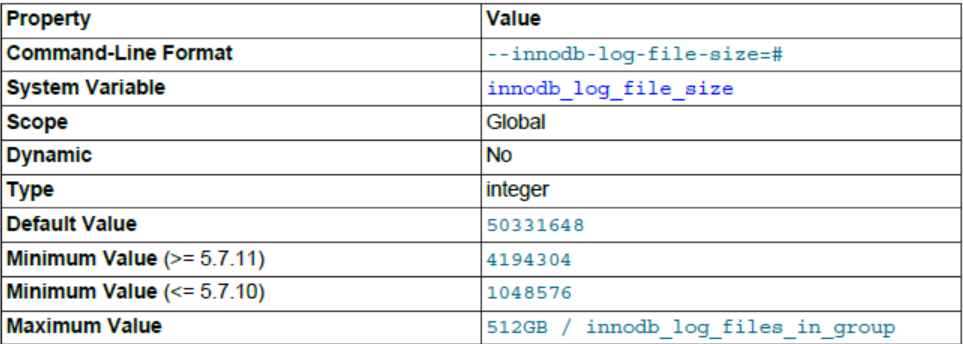
缓冲池大小的大小等于innodb\_buffer\_pool\_chunk\_size\*innodb\_buffer\_pool\_instances的倍数。如果修改的不等，则缓冲池自动调整。

在独立的服务器中，缓冲池大小一般设置为服务器总可用内存的80%。共享服务器需跟据实际情况而定。

## Key\_buffer\_size

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。一般设置服务器总内存的5%。

## innodb\_log\_file\_size

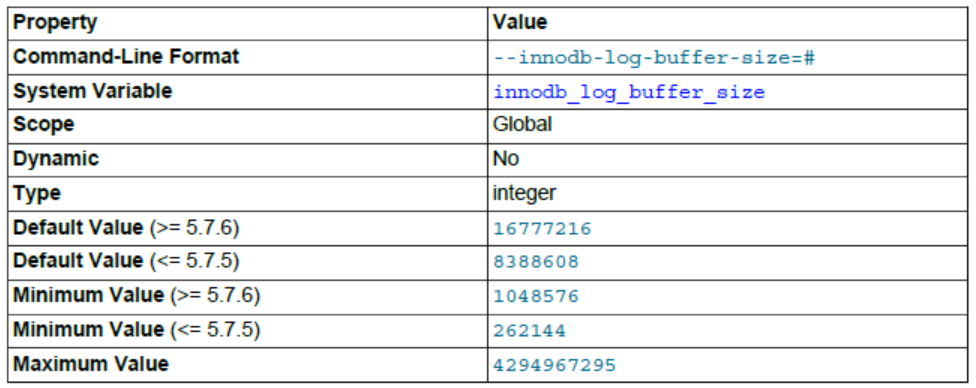


重做日志文件大小。

MySQL在写入数据时，并非将数据直接写入到磁盘文件，而是先将数据写到日志文件，更新的记录也缓存到InnoDB缓冲池中的脏页里面。过一段时间之后，才将数据刷新到磁盘。更改写到事务日志比写到磁盘快。当回滚时，由于数据未写到磁盘，只需擦除InnoDB中的脏页并回滚事务日志即可。已提交的事务，且未写入到磁盘的，如果服务器挂掉，在服务器重启之后，InnoDB会找到最后更新到磁盘的数据，根据事务日志将已提交的事务都写入到磁盘。设置合理的innodb\_log\_file\_size大小，有利于提高性能。

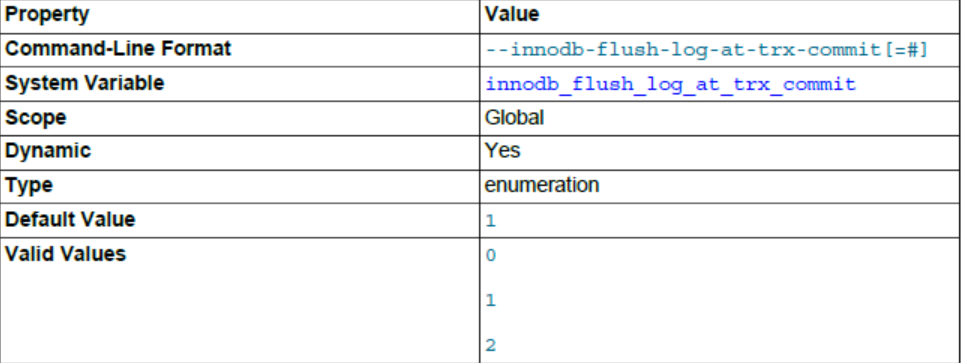
主要有两个规则：小日志文件使写入速度更慢，崩溃恢复速度更快；大日志文件使写入更快，崩溃恢复速度更慢。小日志文件很快就会写满，需要频繁的刷新到磁盘，在产生大量的写操作的时候，性能会下降。大日志文件写入数据快，但是在崩溃恢复时需要读取大量的重做日志文件，恢复很慢。

## innodb\_log\_buffer\_size



InnoDB 将日志写入日志磁盘文件前的缓冲大小。大的日志缓冲允许事务运行时不需要将日志保存入磁盘而只到事务被提交(commit)。 因此，如果有大的事务处理，设置大的日志缓冲可以减少磁盘I/O。

## innodb\_flush\_log\_at\_trx\_commit



默认值1：InnoDB在每次事务提交后日志立即写入操作体统缓存并刷新到磁盘。如果使用autocommit，那么每一个INSERT, UPDATE或DELETE语句都是一个事务提交。

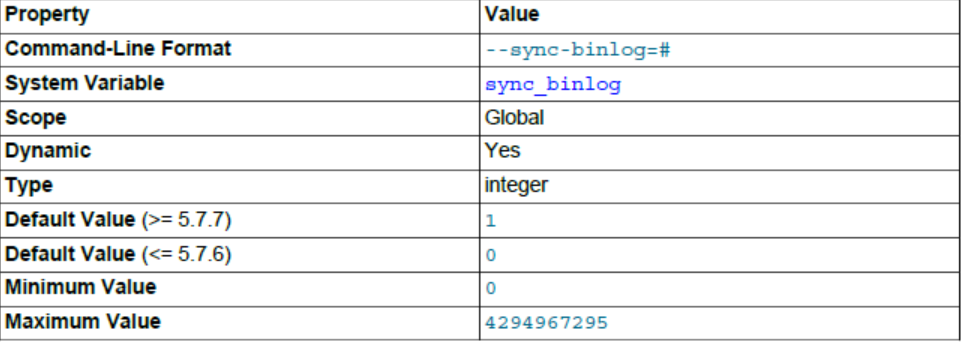
0：每秒执行一次日志写入操作系统缓存并刷新到磁盘。

2：每此事务提交日志立即写入操作系统缓存，每秒执行一次刷新到磁盘。

默认值1能够获得完整的事务ACID。0在提交事务后刷新到操作系统缓存前如果系统崩溃会导致丢失上一秒提交的事务。

操作系统缓存刷新到磁盘非常快，相对而言，使用2会提高一定性能，同时也最大程度保证了事务。

## sync\_binlog

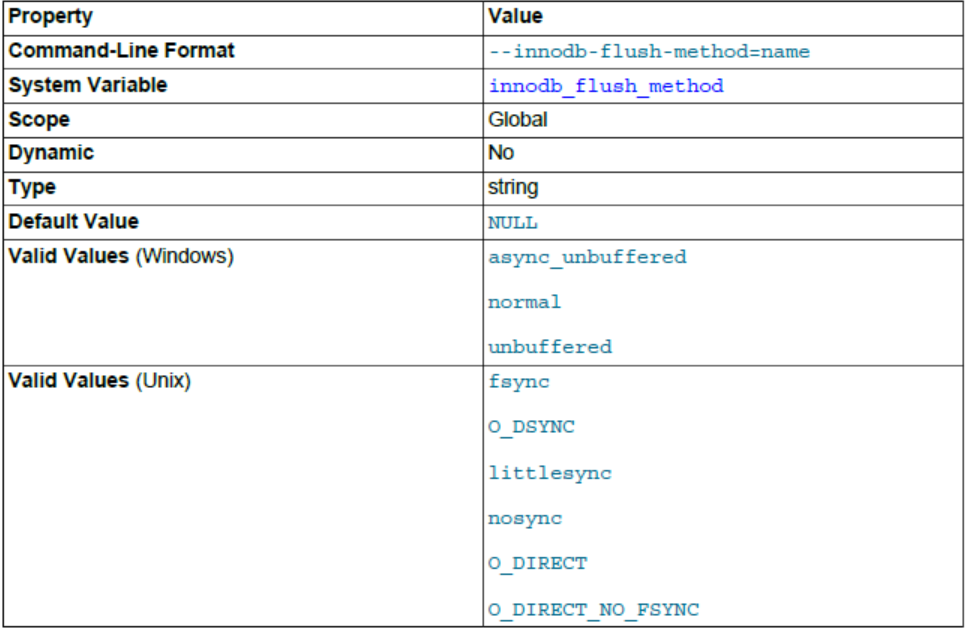


0：提供最好的性能，但是在断电或者系统崩溃时会导致已提交事务未同步到二进制日志。在没有设置从服务器且不做备份，或者在设置从服务器且做备份但是不介意当主服务器崩溃时二进制日志丢失部分事件，可以使用这个选项得到更好的性能。

1：在事务提交之前同步到二进制日志，即写入磁盘。当设置了从服务器，在意主从服务器一致性，且希望在系统崩溃后能及时回复到某个事件点（通过使用最新的一致性备份和二进制日志将数据库恢复到特定时间点）则innodb\_flush\_log\_at\_trx\_commit=1且考虑sync\_binlog=1

N：值为非0和1的其他值，表示当有n个二进制日志提交之后再同步到磁盘。系统崩溃时会导致数据丢失。值越大性能越好，但是丢失数据的风险越大。

## innodb\_flush\_method



设置刷新数据到InnoDB数据文件和日志文件的方式，影响I/O吞吐量。

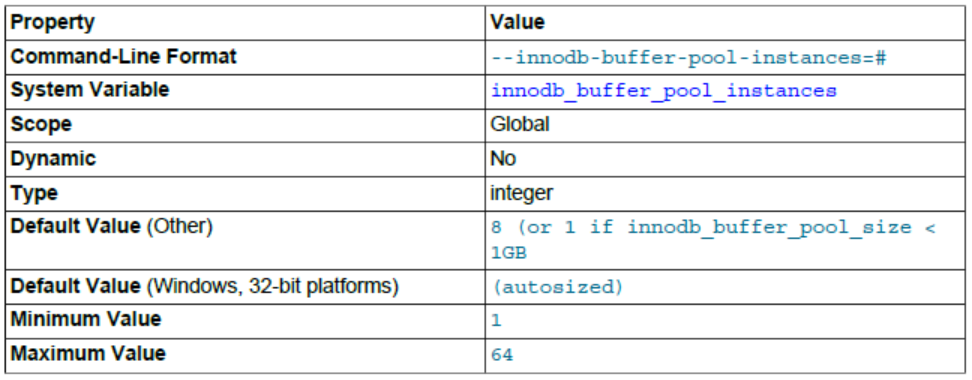
默认值为null时，在累Unix平台使用的是fsysc，在Windows平台使用的是async\_unbuffered。

类Unix选项：

* fsync：使用fsync()来刷新数据和日志文件
* O\_DSYNC：使用O\_SYNC来打开和刷新日志文件，使用fsync()来刷新数据文件
* Littlesync，nosync：内部新能测试用，目前不支持
* O\_DIRECT：使用O\_DIRECT来打开数据文件，使用fsync()来刷新数据文件和日志文件
* O\_DIRECT\_NO\_FSYNC：

How each setting affects performance depends on hardware configuration and workload. Benchmark your particular configuration to decide which setting to use, or whether to keep the default setting. Examine the InnoDB\_data\_fsyncs status variable to see the overall number of fsync() calls for each setting. The mix of read and write operations in your workload can affect how a setting performs. For example, on a system with a hardware RAID controller and battery-backed write cache, O\_DIRECT can help to avoid double buffering between the InnoDB buffer pool and the operating system file system cache. On some systems where InnoDB data and log files are located on a SAN, the default value or O\_DSYNC might be faster for a read-heavy workload with mostly SELECT statements. Always test this parameter with hardware and workload that reflect your production environment.

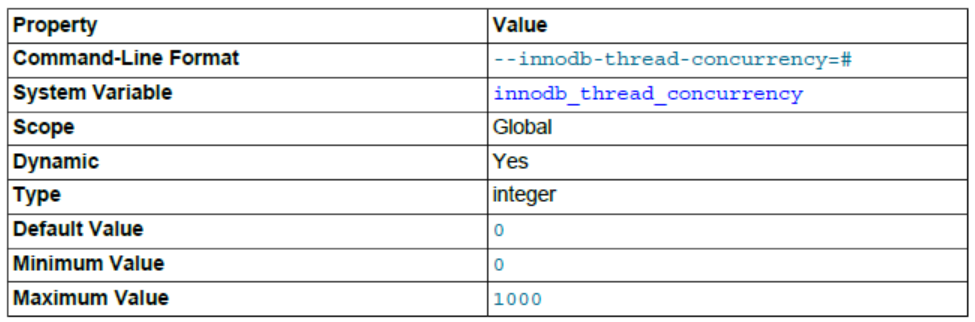
## innodb\_buffer\_pool\_instances



InnoDB缓冲池实例个数，即将缓冲池平分为指定的个数。可以提高并发性能。该设置不会影响单个查询响应时间，只有在高并发负载的情况下才会有用。

该配置只有在innodb\_buffer\_pool\_size大于1G的时候才会生效。为了得到更好的效率，设置合理的个数使每个缓冲池实例大小大于1G。

## innodb\_thread\_concurrency



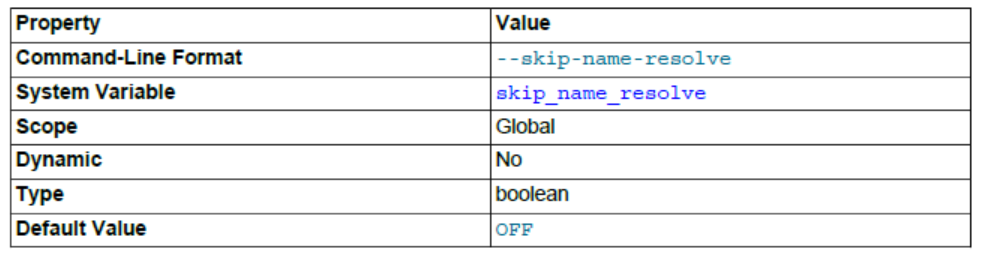
该参数用来控制并发执行的线程数。默认为0，即关闭并发控制，此时InnoDB会尽可能多的创建所需要的线程。当系统CPU或者IO出现饱和时，可以使用该参数来提高查询性能。

InnoDB会保持操作系统并发线程数在InnoDB内小于或等于该变量指定值。一旦达到上限，剩余的线程将会放在一个FIFO的队列里面等待执行。

以下是关于该值的几点参考：

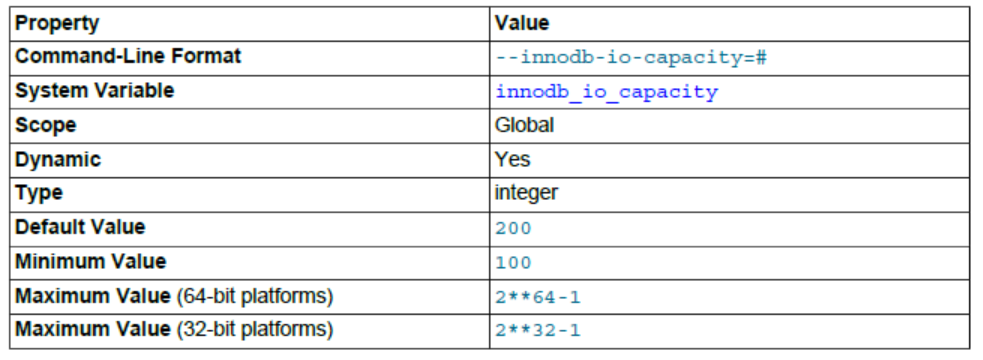
* If the number of concurrent user threads for a workload is less than 64, set innodb\_thread\_concurrency=0.
* If your workload is consistently heavy or occasionally spikes, start by setting innodb\_thread\_concurrency=128 and then lowering the value to 96, 80, 64, and so on, until you find the number of threads that provides the best performance. For example, suppose your system typically has 40 to 50 users, but periodically the number increases to 60, 70, or even 200. You find that performance is stable at 80 concurrent users but starts to show a regression above this number. In this case, you would set innodb\_thread\_concurrency=80 to avoid impacting performance.
* If you do not want InnoDB to use more than a certain number of virtual CPUs for user threads (20 virtual CPUs, for example), set innodb\_thread\_concurrency to this number (or possibly lower, depending on performance results). If your goal is to isolate MySQL from other applications, you may consider binding the mysqld process exclusively to the virtual CPUs. Be aware, however, that exclusive binding could result in non-optimal hardware usage if the mysqld process is not consistently busy. In this case, you might bind the mysqld process to the virtual CPUs but also allow other applications to use some or all of the virtual CPUs. innodb\_thread\_concurrency values that are too high can cause performance regression due to increased contention on system internals and resources.
* In some cases, the optimal innodb\_thread\_concurrency setting can be smaller than the number of virtual CPUs.
* Monitor and analyze your system regularly. Changes to workload, number of users, or computing environment may require that you adjust the innodb\_thread\_concurrency setting.

## skip\_name\_resolve



该参数用来配置连接时DNS解析，正常情况下DNS解析非常快，但是当DNS解析失败时服务为会出现“unauthenticated connections”导致所有请求都慢下来。因此，打开此开关，避免基于主机名的授权。直接使用IP地址。

## innodb\_io\_capacity

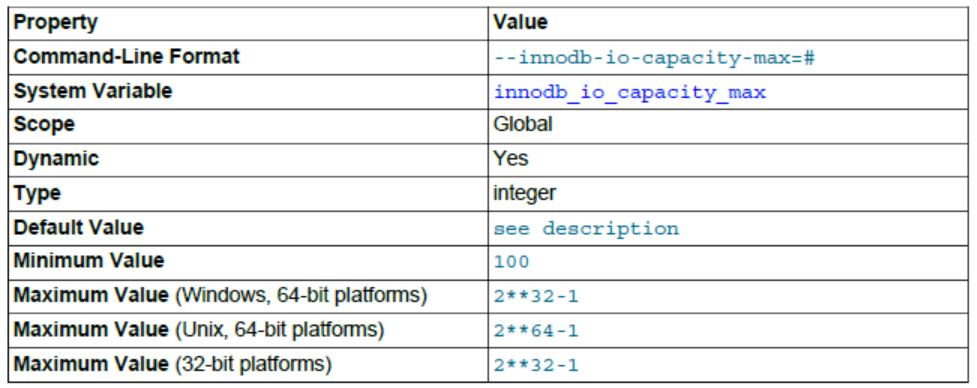


用来当刷新脏数据时，控制MySQL每秒执行的写IO量。innodb\_io\_capacity限制的是所有缓冲池实例的总量，当刷新脏页时，这个量会平分到每个实例。

innodb\_io\_capacity可设置为接近操作系统能处理的最大IO数。对于低端的SSD，默认值200就可以了，对于高端以及总线连接的SSD可以将值设为1000，而5400RPM或者是7200RPM的硬盘，则将值设为100。

innodb\_io\_capacity可以设置为任何低于innodb\_io\_capacity\_max的值。虽然可以设置为较大的值，但实际中，较大的值并不会带来多少好处。通常不推荐设置为20000及以上的值，除非对于当前的负载该配置不满足。

## innodb\_io\_capacity\_max



在紧急情况下，控制当刷新脏数据时MySQL每秒执行的写IO量。

在启动时指定了innodb\_io\_capacity而未指定innodb\_io\_capacity\_max，则为innodb\_io\_capacity的两倍或者最小值2000。

一般情况innodb\_io\_capacity\_max为innodb\_io\_capacity的两倍。

在写密集型系统中，可以测量设备存储设置的随机吞吐量，然后innodb\_io\_capacity\_max设置为设备所能达到的最大IOPS。innodb\_io\_capacity设置为innodb\_io\_capacity\_max的50%~75%。

When configuring innodb\_io\_capacity\_max, twice the innodb\_io\_capacity is often a good starting point. The default value of 2000 is intended for workloads that use a solid-state disk (SSD) or more than one regular disk drive. A setting of 2000 is likely too high for workloads that do not use SSD or multiple disk drives, and could allow too much flushing. For a single regular disk drive, a setting between 200 and 400 is recommended. For a high-end, bus-attached SSD, consider a higher setting such as 2500. As with the innodb\_io\_capacity setting, keep the setting as low as practical, but not so low that InnoDB cannot sufficiently extend beyond the innodb\_io\_capacity limit, if necessary.