

中级

- 1 哪些情况有索引，查询还是很慢
- 2 虚拟机往 SSD 上迁移，需要调整什么参数
- 3 重做日志（Redo Log）和二进制日志的区别
- 4 MySQL 是否需要开启查询缓存？并说出理由
- 5 关联查询的几种模式？
- 6 幻读和不可重复读的区别
- 7 MySQL 哪些参数需要优化
- 8 画一个 B+ 树
- 9 查看冗余索引，SQL 执行情况
- 10 模拟几种死锁场景
- 11 什么情况会导致死锁？
- 12 怎么降低死锁概率？
- 13 主库执行 delete，但是从库同步延迟非常高可能有哪些原因？
- 14 second_behind_master 原理
- 15 怎样判断主从延迟，位点具体对比参数，GITD 模式具体对比参数，主从延迟的原因有哪些？
- 16 怎么保证 MySQL 主从一致性
- 17 怎么检查主从一致
- 18 MHA 原理
- 19 如何提高 MySQL 的安全？
- 20 备份策略
- 21 mysqldump 加 --single-transaction 会把 mysql 设置成什么隔离级别
- 22 xtrabackup 备份恢复原理
- 23 MySQL 分库分表类型和中间件
- 24 如何搭建慢查询系统的
- 25 一条 select 语句的过程？
- 26 一条 update 会经历哪些过程？
- 27 MySQL 怎么实现 MVCC 的
- 28 pt-online-schema-change 原理，3 个触发器的内容，它有什么问题？
- 29 gh-ost 原理，需要满足哪些条件？
- 30 双写的好处
- 31 两阶段提交
- 32 insert buffer (change buffer) 是什么？有何意义？使用 insert buffer 需要满足哪些条件？
- 33 为什么再大的事务提交的时间也是很短的？
- 34 事务实现原理
- 35 MySQL 遇到过比较棘手的问题
- 36 自增主键到上限了会发生什么？
- 37 能想到的 Buffer 相关参数有哪些？
- 38 Redis 数据淘汰策略有哪些？说出他们的区别
- 39 Redis 如何做备份？
- 40 Redis 主从复制的原理
- 41 Redis 常见监控项
- 42 使用过哪些 Redis 工具

欢迎关注公众号“悦专栏”，专注分享数据库相关内容，包括 MySQL、Redis、ClickHouse、MongoDB 等。关注后，回复“合集”，可获取“悦专栏”历史原创干货文章。



微信搜一搜



悦专栏

中级

1 哪些情况有索引，查询还是很慢

函数操作

隐式转换

模糊查询

范围查询

2 虚拟机往 SSD 上迁移，需要调整什么参数

innodb_io_capacity: 刷新脏页的数量

3 重做日志（Redo Log）和二进制日志的区别

Redo Log 是在 InnoDB 存储引擎层产生的。而 Binlog 是在 MySQL 数据库的上层生成的，其不单单记录 InnoDB 引擎的修改，也记录其他任何存储引擎的修改；

Redo Log 是物理逻辑格式日志，记录的是每个页的修改。而 Binlog 是一种逻辑日志，记录的是对应的变更 SQL；

事务进行中，会不断地写入 Redo Log；而 Binlog 只在事务提交时写入一次。

4 MySQL 是否需要开启查询缓存？并说出理由

开启查询缓存有以下缺点：

需要前后两条 SQL 完全一样才能使用查询缓存

只要存在更新，就会清空这张表的查询缓存

因此，如果线上环境中 99% 以上都是读，就可以考虑开启查询缓存，其他情况就不建议开启查询缓存了。

5 关联查询的几种模式？

Index Nested-Loop Join(NLJ): 关联是可以用到被驱动表的索引

Block Nested-Loop Join(BNL): 被驱动表没有索引, 把一张表的数据放入 join buffer 中, 把另一张表的每一行取出来, 跟 join_buffer 中的数据做对比, 满足 join 条件的, 作为结果集的一部分返回。

6 幻读和不可重复读的区别

幻读: 针对 insert

不可重复读: 针对 delete/update

7 MySQL哪些参数需要优化

数据安全相关的

log-bin

开启binlog

binlog_format

日志格式

log_slave_updates

innodb_flush_log_at_trx_commit

innodb的redo日志刷新方式, 对innodb的影响会很大

sync_binlog =1

binlog的落盘时机

性能相关的

innodb_buffer_pool_size

一般可以设置内存的 50%-80%

innodb_file_per_table

采用独立表空间

transaction_isolation

隔离级别

query_cache_size = 0

query_cache_type = 0

#关闭QC

max_connections

最大连接数

使用相关的

read_only

只读

lower_case_table_names

表名区分大小写

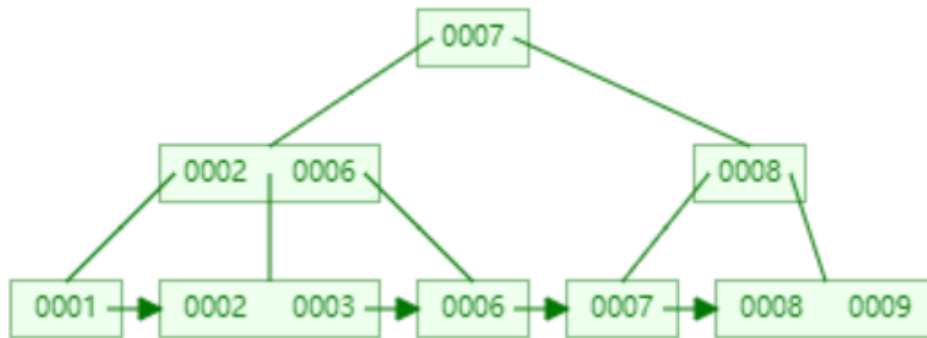
interactive_timeout = 600

当一个连接超过十分钟不活跃就把它kill 掉

wait_timeout = 600

server_id

8 画一个 B+ 树



9 查看冗余索引，SQL 执行情况

冗余索引

```
select * from sys.schema_redundant_indexes\G
```

统计无用索引

```
select * from sys.schema_unused_indexes;
```

统计全表扫描

```
select * from sys.schema_tables_with_full_table_scans limit 4;
```

统计DML锁

```
select * from sys.schema_table_lock_waits limit 4\G
```

10 模拟几种死锁场景

同一张表下的死锁

不同表下的死锁

间隙锁下的死锁

insert 语句的死锁：3 个 会话写入同一行数据

11 什么情况会导致死锁？

死锁是指两个或者多个事务在同一资源上相互占用，并请求锁定对方占用的资源，从而导致恶性循环的现象。

一下几种常见情况会导致死锁：

不同线程并发访问同一张表的多行数据，并按顺序访问导致死锁；

不同线程并发访问多张表时，未按顺序访问导致死锁；

RR 隔离级别下，由于间隙锁导致死锁。

12 怎么降低死锁概率？

更新 SQL 的 where 条件尽量用索引；

基于 primary 或 unique key 更新数据；

减少范围更新，尤其非主键、非唯一索引上的范围更新；

加锁顺序一致，尽可能一次性锁定所有需要行；

将 RR 隔离级别调整为 RC 隔离级别。

13 主库执行 delete，但是从库同步延迟非常高可能有哪些原因？

没有索引

表没主键

14 second_behind_master 原理

对于主从库机器时间不一致的情况，在 I/O 线程第一次启动时，会计算主从机器之间的时间差，在后续计算复制延迟时，会把这个时间差减掉。

如果 I/O 和 SQL 线程同时为 YES，且 SQL 线程没有做任何事情，此时直接判定复制延迟结果为 0，不会走公式计算。

如果 SQL 线程为 Yes，且还存在着 I/O 线程已经读取的 relay log 未应用完成的，则会走公式计算延迟时间，而不管 I/O 线程是否正在运行，但当 SQL 线程重放完成了所有 relay log 时，如果 I/O 线程不为 Yes，直接判定复制延迟结果为 NULL。

任何时候，如果 SQL 线程不为 YES，直接判定复制延迟为 NULL。

当计算出现负数，直接归零。

官方文档解释：<https://dev.mysql.com/doc/refman/8.0/en/show-replica-status.html>

statement 使用 UUID 导致 主从数据不一致

15 怎样判断主从延迟，位点具体对比参数，GITD 模式具体对比参数，主从延迟的原因有哪些？

a、seconds_behind_master 是否等于 0

b、对比位点：如果 Master_Log_File 和 Relay_Master_Log_File、Read_Master_Log_Pos 和 Exec_Master_Log_Pos 这两组值完全相同，就表示接收到的日志已经同步完成。

c、对比 GTID：Auto_Position=1，表示这对主备关系使用了 GTID 协议；Retrieved_Gtid_Set，是备库收到的所有日志的 GTID 集合；Executed_Gtid_Set，是备库所有已经执行完成的 GTID 集合。

延迟原因：

a、主从机器性能差异

b、备库压力大

c、大事务

d、备库的并行复制能力

16 怎么保证 MySQL 主从一致性

Binlog 格式设置为 Row

不在从库进行更新操作

开启双一

```
innodb_flush_log_at_trx_commit = 1  
sync_binlog = 1
```

开启半同步复制

17 怎么检查主从一致

pt-table-checksum

可以指定对应的表

pt-table-sync 修复不一致

18 MHA 原理

1. 从宕机崩溃的 master 保存二进制日志事件（保存 binlog）
2. 通过对比 slave 之间 I/O 线程读取 masterbinlog 的位置，选取最接近的 slave 作为 latestslave。（选 latestslave）
3. 其他 slave 通过与 latest slave 对比生成差异中继日志。（对比 relay log）
4. 在 latest slave 上应用从 master 保存的 binlog，同时将 latest slave 提升为 master。（切换）
5. 最后在其他 slave 上应用相应的差异中继日志并开始从新的 master 开始复制。（其他 slave 接上新从）

19 如何提高 MySQL 的安全?

- 1、禁止 root 账号远程登录
 - 2、权限最小化
 - 3、MySQL 只对内网开放
 - 4、防止 SQL 注入
 - 5、明文密码采用加密方式存储
- 等等

20 备份策略

开放型问题，比如每日全备，增量备份通过备份 Binlog，并增加异地备份和延迟备份等。

21 mysqldump 加 --single-transaction 会把 mysql 设置成什么隔离级别

RR

22 xtrabackup 备份恢复原理

xtrabackup 备份原理：

- 1、复制已有的 redo log，然后监听 redo log 变化并持续复制
- 2、复制事务引擎数据文件
- 3、加锁：全局读锁
- 4、备份非事务引擎数据文件及其他文件
- 5、获取 binlog 位点信息等元数据
- 6、停止复制 redo log
- 7、解锁：全局读锁
- 8、复制 buffer pool dump
- 9、备份完成

xtrabackup 恢复原理：

恢复过程类似 mysqld crash后，做一次 crash recover。

恢复的目的是把备份集中的数据恢复到一个一致性位点。

InnoDB 的 ibd 文件拷贝是在 FTWRL 前做的，拷贝出来的不同 ibd 文件最后更新时间点是不一样的，这种状态的 ibd 文件是不能直接用的，但是 redo log 是从备份开始一直持续拷贝的，最后的 redo 日志点是在持有 FTWRL 后取得的，所以最终通过 redo 应用后的 ibd 数据时间点也是和 FTWRL 一致的。

23 MySQL 分库分表类型和中间件

水平分割：将一张表的数据拆分到不同的数据库中

垂直分割：将不同的表放到不同的数据库中

全局表

取模分片

范围分片

分库分表中间件

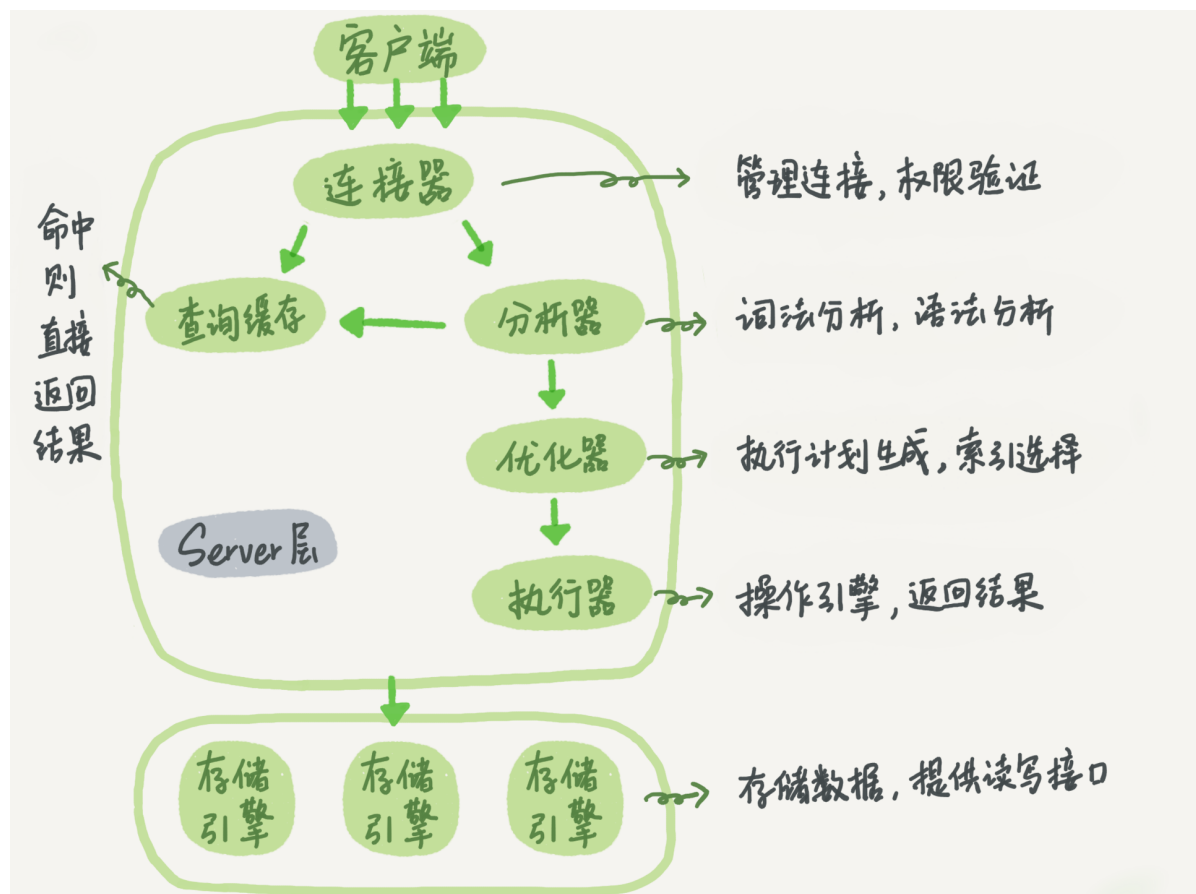
MyCAT、DBLE、Atlas、MySQL Route、TDDL 等

24 如何搭建慢查询系统的

1 Filebeat + Kafka + Logstash + Elasticsearch + Kibana

2 ClickTail + ClickHouse + grafana

25 一条 select 语句的过程？



26 一条 update 会经历哪些过程？

执行器查找数据，如果数据页在内存中，则直接返回给执行器；如果数据页不在内存中，则从磁盘读入内存，然后再返回。

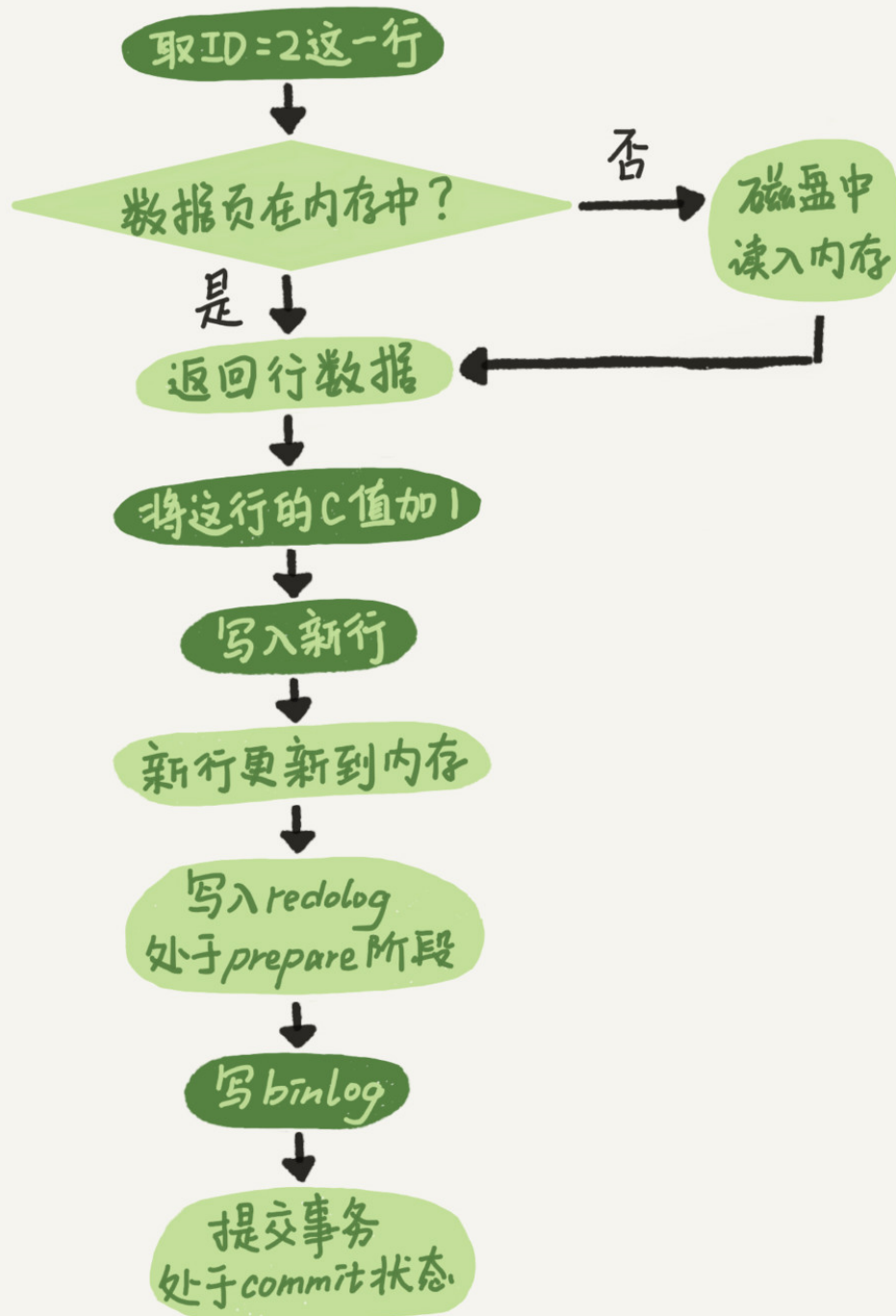
执行器拿到数据后，进行修改，得到新的数据，调用引擎接口写入新数据

引擎将新数据更新到内存，同时记录 redo log，此时 redo log 处于 prepare 状态，告知执行器执行完了，随时可以提交

执行器生成binlog，并让 binlog 落盘

执行器调用引擎的提交事务接口，把 刚刚写入 redo log 改成 commit 状态

更新完成



27 MySQL怎么实现 MVCC 的

MVCC 只在 RC 和 RR 两个隔离级别下工作。

InnoDB 每一行数据都有一个隐藏的回滚指针，用于指向该行修改前的最后一个历史版本（存放在 UNDO log 中）

MVCC 最大的好处是读不加锁，读写不冲突。极大增加了 MySQL 的并发性。

28 pt-online-schema-change 原理，3 个触发器的内容，它有什么问题？

原理

1. 创建一个和要执行 alter 操作的表一样的新的空表结构(是alter之前的结构)
2. 在新表执行alter table 语句
3. 在原表中创建触发器3个触发器分别对应insert,update,delete操作
4. 以一定块大小从原表拷贝数据到临时表，拷贝过程中通过原表上的触发器在原表进行的写操作都会更新到新建的临时表
5. Rename 原表到old表中，在把临时表Rename为原表
6. 如果有参考该表的外键，根据alter-foreign-keys-method参数的值，检测外键相关的表，做相应设置的处理
7. 默认最后将旧原表删除

触发器内容

insert = replace into

update=delete ignore + replace into

delete =delete ignore

copy rows = insert ignore into

限制

1. 原表必须要有主键或者唯一索引（不含NULL）
2. 原表上不能有触发器存在
3. 使用前需保证有足够的磁盘容量，因为复制原表需要一倍的空间

问题

增加唯一索引，如果有重复数据，会导致数据丢失；或者其他你遇到过的问题。

29 gh-ost 原理，需要满足哪些条件？

原理

- 1、gh-ost 首先连接到主库上，根据 alter 语句创建幽灵表
- 2、然后作为一个备库连接到其中一个真正的备库或者主库上(根据具体的参数来定)，一边在主库上拷贝已有的数据到幽灵表，一边从备库上拉取增量数据的 binlog，然后不断的把 binlog 应用回主库。
- 3、等待全部数据同步完成，进行 cut-over 幽灵表和原表切换。

操作过程会生成两个中间状态的表

_b_ghc 记录执行过程

_b_gho 幽灵表

限制

1. 源表必须要有主键或者唯一索引

2.不支持外键

3.不支持触发器

4.不支持虚拟列

5.不支持 5.7 point类型的列

6. 5.7 JSON列不能是主键

7.不能存在另外一个table名字一样，只是大小写有区别

8.不支持多源复制

9.不支持M-M 双写

10.不支持FEDERATED engine

30 双写的好处

双写

存储引擎缓冲池数据页默认为 16 KB，而文件系统一页大小为 4KB，所以在进行刷盘操作时，可能导致一部分刷新，导致写失效，而重启时，磁盘就不完整

在重做日志前，用户需要一个页的副本，当写入失效发生时，先通过页的副本来还原该页，再进行重做，这就是double write

双写的好处

增加了数据页的可靠性

在应用重做日志前，用户需要一个页的副本，当写入失效发生时，先通过页的副本来还原该页，再进行重做。

思考

磁盘写入时，都是以 512 字节为单位的，不能保证 MySQL 数据页面 16KB 的一次性原子写入。可能产生页面断裂的问题，如果能保证 16 KB 数据的原子写入，就可以取消两次写了。

还有全物理 REDO，将一个页面的写操作都拆成一个个小的物理写入。这种情况就不会出现写断裂了。

31 两阶段提交

事务提交时，分成 prepare 和 commit 两个阶段

redo log prepare: write

binlog: write

redo log prepare: fsync

binlog: fsync

redo log commit: write

flush 阶段

sync 阶段

commit 阶段

32 insert buffer (change buffer) 是什么？有何意义？使用insert buffer需要满足哪些条件？

InnoDB 从 1.0.x 版本开始引入了 Change Buffer，可以算是对 Insert Buffer 的升级

对于非聚集索引（非唯一索引）的插入或更新操作，不是每一次直接插入索引页中，

而是先判断插入的非聚集索引页是否在缓存池中

如果在，则直接插入；

如果不在，则先放入一个 Insert buffer 对象中。

然后再以一定频率和情况进行 insert buffer 和辅助索引叶子结点的 merge操作

这时通常能将多个插入合并到一个操作中

大大提高了对于非聚集索引插入的性能。

33 为什么再大的事务提交的时间也是很短的？

即使事务还没提交，InnoDB 存储引擎仍然每秒会将重做日志缓冲中的内容刷新到重做日志文件。因此再大的事务提交的时间也是很短。

34 事务实现原理

原子性：使用 undo log，从而达到回滚

持久性：使用 redo log，从而达到故障后恢复

隔离性：使用锁以及MVCC,运用的优化思想有读写分离，读读并行，读写并行

一致性：通过回滚，以及恢复，和在并发环境下的隔离做到一致性

35 MySQL 遇到过比较棘手的问题

开放型问题，尽量选择比较难的问题。

36 自增主键到上限了会发生什么？

表的自增 id 达到上限后，再申请时它的值就不会改变，进而导致继续插入数据时报主键冲突的错误。

row_id 达到上限后，则会归 0 再重新递增，如果出现相同的 row_id，后写的数据会覆盖之前的数据。

37 能想到的 Buffer 相关参数有哪些？

innodb_buffer_pool_size

key_buffer_size

innodb_change_buffering

innodb_log_buffer_size

binlog_cache_size

thread_cache_size

table_open_cache

38 Redis 数据淘汰策略有哪些?说出他们的区别

当 Redis 内存达到 maxmemory 上线时，会触发相应的溢出控制策略。有maxmemory-policy 参数控制，Redis 支持 6 中策略，如下：

noeviction：默认策略，不会删除任何数据，拒绝所有写入操作并返回客户端错误信息

volatile-lru：根据LRU 算法删除设置了超时属性的建，直到腾出足够空间为止。如果没有可删除的键对象，回退到noeviction 策略。

allkeys-lru：根据 LRU 算法删除键，不管数据有没有设置超时属性，直到腾出足够空间为止。

allkeys-random：随机删除所有键，直到腾出足够空间为止。

volatile-random：随机删除过期建，直到腾出足够空间为止。

volatile-ttl：根据键值对象的 ttl 属性，删除最近将要过期的数据，如果没有，回退到noeviction。

39 Redis 如何做备份?

Redis 支持两种持久化机制：RDB 和 AOF

RDB：是把当前进程数据生成快照保存到硬盘的过程，触发 RDB 持久化过程分为手动触发和自动触发。手动触发分别对应 save 和 bgsave

自动触发：

1) save 相关配置，save m n。表示 m 秒内数据集存在 n 次修改时，自动触发bgsave。

2) 从节点执行全量复制，主节点会自动执行 bgsave

3) 执行 debug reload 命令重新加载 Redis 时，也会自动触发 save 操作

4) 执行 shutdown 命令时，如果没有开启 AOF持久化功能则自动执行 bgsave。

AOF:以独立的日志方式记录每次写命令，重启时再重新执行 AOF文件中的命令达到恢复数据的目的。

AOF 的主要作用是解决了数据持久化的实时性。

40 Redis 主从复制的原理

1) 保存主节点信息

执行 slaveof 后从节点只保存主节点的地址信息便直接返回。

2) 定时任务检测主节点

从节点内部通过每秒运行的定时任务维护复制相关逻辑，当定时任务发现存在新的主节点后，会尝试与该节点建立网络连接。

3) 发送 ping 命令

连接建立成功后从节点发送ping 请求进行首次通信，ping 请求主要目的如下：

检测主从之间网络套接字是否可用。

检测主节点当前是否可接受处理命令

如果发送ping 命令后，从节点没有收到主节点的pong 恢复或者超时，比如网络超时或者主节点正在阻塞无法响应命令，从节点会断开复制连接，下次定时任务会发起重连。

4) 权限验证

如果主节点设置了requirepass参数，则需要密码验证，从节点必须配置masterauth参数保证与主节点相同的密码才能通过验证；如果验证失败复制将终止，从节点重新发起复制流程。

5) 同步数据集

主从复制连接正常通信后，对于首次建立复制的场景，主节点会把持有的数据全部发送给从节点，这部分操作是耗时最长的步骤。Redis在2.8版本以后采用新复制命令psync进行数据同步，原来的sync命令依然支持，保证新旧版本的兼容性。新版同步划分两种情况：全量同步和部分同步。

6) 命令持续复制

当主节点把当前的数据同步给从节点后，便完成了复制的建立流程。接下来主节点会持续地把写命令发送给从节点，保证主从数据一致性。

41 Redis 常见监控项

1、连接检测

连接失败检测

插入检测

2、变量检测

readonly

maxmemory

maxmemory-policy

连接数检测

内存比检测

3、主从复制检测

角色检测

复制状态检测

延迟检测

4、访问量

read

write

qps

5、内存

maxmemory

used maxmemory

内存碎片率

6、key

evicted_keys (运行以来删除的key)

key 数量

命中率

7、其他

连接数

从库延迟

慢查询的数量

42 使用过哪些 Redis 工具

redis-faina

redis-shake