1.存储引擎的 InnoDB 与 MyISAM 的区别，优缺点，使用场景？

- **InnoDB和MyISAM存储引擎区别：**

	Innodb	Myisam
存储文件	.frm 表定义文件 .ibd 数据文件和索引文件	.frm 表定义文件 .myd 数据文件 .myi 索引文件
锁	表锁、行锁	表锁
事务	支持	不支持
CRDU	读、写	读多
count	扫表	专门存储的地方 （加where也扫表）
索引结构	B+ Tree	B+ Tree
外键	支持	不支持

存储引擎的选型：

InnoDB：支持事务处理，支持外键，支持崩溃修复能力和并发控制。如果需要对事务的完整性要求比较高（比如银行），要求实现并发控制（比如售票），那选择 InnoDB 有很大的优势。如果需要频繁的更新、删除操作的数据库，也可以选择 InnoDB，因为支持事务的提交（commit）和回滚（rollback）。

MyISAM：插入数据快，空间和内存使用比较低。如果表主要是用于插入新记录和读出记录，那么选择 MyISAM 能实现处理高效率。如果应用的完整性、并发性要求比 较低，也可以使用。

MEMORY：所有的数据都在内存中，数据的处理速度快，但是安全性不高。如果需要很快的读写速度，对数据的安全性要求较低，不需要持久保存，可以选择 MEMOEY。它对表的大小有要求，不能建立太大的表。所以，这类数据库只使用在相对较小的数据库表。

注意，同一个数据库也可以使用多种存储引擎的表。如果一个表要求比较高的事务处理，可以选择 InnoDB。这个数据库中可以将查询要求比较高的

表选择 MyISAM 存储。如果该数据库需要一个用于查询的临时表，可以选择 MEMORY 存储引擎。

2、说说 MySQL 优化之道？

1. 首先需要使用【慢查询日志】功能，去获取所有查询时间比较长的 SQL 语句

2. 其次【查看执行计划】查看有问题的 SQL 的执行计划 explain

3. 最后可以使用【show profile[s]】 查看有问题的 SQL 的性能使用情况

4. 设计中间表，一般针对于统计分析功能，或者实时性不高的需求（OLTP、OLAP）为减少关联查询，创建合理的冗余字段（考虑数据库的三范式和查询性能的取舍，创建冗余字段还需要注意数据一致性问题）

5. 对于字段太多的大表，考虑拆表（比如一个表有 100 多个字段） 人和身份证对于表中经常不被使用的字段或者存储数据比较多的字段，考虑拆表（比如商品表中会存储商品介绍，此时可以将商品介绍字段单独拆解到另一个表中，使用商品 ID 关联）

6. 索引优化

where 字段 、组合索引 （最左前缀）、 索引下推 （非选择行 不加锁）、索引覆盖（不回表）、on 两边、排序 、分组统计 、不要用 * 、 LIMIT 优化、小结果集关联大结果集

3. UndoLog 和 RedoLog 的区别和联系？

Redo log Buffer 重做日志缓冲

原子性，持久性和一致性主要是通过 redo log、undo log 和 Force Log at

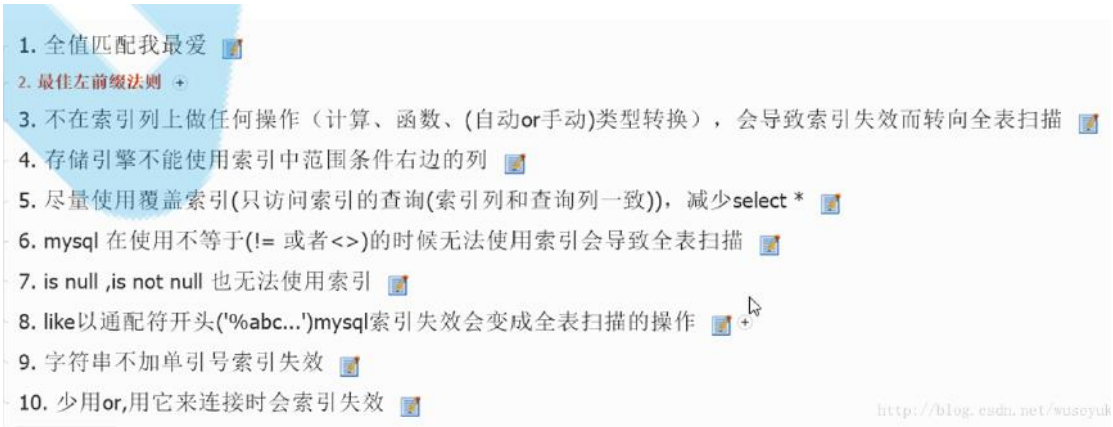
Commit 机制机制来完成的。redo log 用于在崩溃时恢复数据，undo log 用于对事务的影响进行撤销，也可以用于多版本控制。而 Force Log at Commit 机制保证事务提交后 redo log 日志都已经持久化。

重做日志： Redo Log 如果要存储数据则先存储数据的日志 ， 一旦内存崩了 则可以从日志找重做日志保证了数据的可靠性，InnoDB 采用了 Write Ahead Log（预写日志）策略，即当事务提交时，先写重做日志，然后再择时将脏页写入磁盘。如果发生宕机导致数据丢失，就通过重做日志进行数据恢复，数据库崩溃重启后需要从 redo log 中把未落盘的脏页数据恢复出来，重新写入磁盘，保证用户的数据不丢失。当然，在崩溃恢复中还需要回滚没有提交的事务。由于回滚操作需要 undo 日志的支持，undo 日志的完整性和可靠性需要 redo 日志来保证，所以崩溃恢复先做 redo 恢复数据，然后做 undo 回滚。

4. MySQL 索引的数据结构是什么，及为什么使用这种数据结构？

B+树：所有数据在叶子节点，排序直接输出 hash

5. 索引失效的场景有哪些？



6. 什么是死锁和死锁的排查和解决？

MySQL 默认会主动探知死锁，并回滚某一个影响最小的事务。等另一事务执行完成之后，再重新执行该事务。

7. RC 和 RR 的实现原理及区别和使用场景？

MVCC 使得数据库读不会对数据加锁，普通的 SELECT 请求不会加锁，提高了数据库的并发处理能力。借助 MVCC，数据库可以实现 READ COMMITTED，REPEATABLE READ 等隔离级别，用户可以查看当前数据的前一个或者前几个历史版本，保证了 ACID 中的 I 特性（隔离性)。

- 快照读，读取的是记录的可见版本 (有可能是历史版本)，不用加锁。(select)

- 当前读，读取的是记录的最新版本，并且当前读返回的记录，都会加上锁，保证其他事务不会再并发修改这条记录。

8. 分库与分表带来的分布式困境与应对之策？

当【表的数量】达到了几百上千张表时，众多的业务模块都访问这个数据库，压力会比较大，考虑对其进行分库。

当【表的数据】达到了几千万级别，在做很多操作都比较吃力,所以，考虑对其进行分库或者分表

分库分表需要解决的问题：

1.分布式事务问题

本地事务：ACID

分布式事务：根据百度百科的定义，CAP 定理又称 CAP 原则，指的是在一个分布式系统中，Consistency（一致性）、 Availability（可用性）、Partition tolerance（分区容错性)。一致性是强一致性。CAP 理论最多只能同时满足两个。

BASE：基本可用+软状态+最终一致性

2.多个库中表的主键冲突：redis incr 命令，snowflake 算法跨库 join 问题建立全局表（每个库都有一个相同的表） 代码表 E-R 分片（将有 ER 关系的记录都存储到一个库中）

9. Redis 的事务原理是什么？

Redis 的事务是通过 MULTI 、EXEC 、DISCARD 和 WATCH 这四个命令来完成的。

Redis 的单个命令都是原子性的，所以这里需要确保事务性的对象是命令集合。

Redis 将命令集合序列化并确保处于同一事务的命令集合连续且不被打断的执行

Redis 不支持回滚操作。

Redis 不支持事务回滚（为什么呢）

1、大多数事务失败是因为语法错误或者类型错误，这两种错误，在开发阶段都是可以预见的

2、Redis 为了性能方面就忽略了事务回滚。 （回滚记录历史版本）

Redis 中使用 lua 的好处

1、减少网络开销，在 Lua 脚本中可以把多个命令放在同一个脚本中运行

2、原子操作，redis 会将整个脚本作为一个整体执行，中间不会被其他命令插入。换句话说，编写脚本

的过程中无需担心会出现竞态条件

3、复用性，客户端发送的脚本会永远存储在 redis 中，这意味着其他客户端可以复用这一脚本来完成同样的逻辑

EVAL 命令通过执行 redis 的 eval 命令，可以运行一段 lua 脚本。命令说明：

script 参数：是一段 Lua 脚本程序，它会被运行在 Redis 服务器上下文中，这段脚本不必(也不应该)定义为一个 Lua 函数。

numkeys 参数：用于指定键名参数的个数。key [key ...]参数： 从 EVAL 的第三个参数开始算起，使用了 numkeys 个键（key），表示在脚本中所用到的那些 Redis 键(key)，这些键名参数可以在 Lua 中通过全局变量 KEYS 数组,用 1 为基址的形式访问(KEYS[1] ， KEYS[2] ,以此类推)。arg [arg ...]参数：可以在 Lua 中通过全局变量 ARGV 数组访问，访问的形式和 KEYS 变量类似(

ARGV[1] 、 ARGV[2] ，诸如此类)。

redis.call()：返回值就是 redis 命令执行的返回值如果出错，则返回错误信息，不继续执行

redis.pcall()：返回值就是 redis 命令执行的返回值如果出错，则记录错误信息，继续执行注意事项在脚本中，使用 return 语句将返回值返回给客户端，如果没有 return，则返回 nil

SCRIPT 命令

SCRIPT FLUSH ：清除所有脚本缓存

SCRIPT EXISTS ：根据给定的脚本校验和，检查指定的脚本是否存在于脚本缓存

SCRIPT LOAD ：将一个脚本装入脚本缓存，返回 SHA1 摘要，但并不立即运行它

SCRIPT KILL ：杀死当前正在运行的脚本

redis-cli -eval：直接执行 lua 脚本，利用 Redis 整合 Lua，主要是

为了性能以及事务的原子性。因为 redis 帮我们提供的事务功能太差。

10. Redis 链表实现结构是怎样的？

类型	编码	OBJECT ENCODING命令输出	对象
REDIS_STRING	REDIS_ENCODING_INT	“int”	使用整数值实现的字符串对象
REDIS_STRING	REDIS_ENCODING_EMBSTR	“embstr”	使用embstr编码的简单动态字符串实现的字符串对象
REDIS_STRING	REDIS_ENCODING_RAW	“raw”	使用简单动态字符串实现的字符串对象
REDIS_LIST	REDIS_ENCODING_ZIPLIST	“ziplist”	使用压缩列表实现的列表对象
REDIS_LIST	REDIS_ENCODING_LINKEDLIST	“linkedlist”	使用双端链表实现的列表对象
REDIS_HASH	REDIS_ENCODING_ZIPLIST	“ziplist”	使用压缩列表实现的哈希对象
REDIS_HASH	REDIS_ENCODING_HT	“hashtable”	使用字典实现的哈希对象
REDIS_SET	REDIS_ENCODING_INTSET	“intset”	使用整数集合实现的集合对象
REDIS_SET	REDIS_ENCODING_HT	“hashtable”	使用字典实现的集合对象
REDIS_ZSET	REDIS_ENCODING_ZIPLIST	“ziplist”	使用压缩列表实现的有序集合对象
REDIS_ZSET	REDIS_ENCODING_SKIPLIST	“skiplist”	使用跳跃表和字典实现的有序集合对象

11.如何提高链表的查询性能？

跳表

12. 为什么 Redis 是单线程运行的性能还特别高？

数据结构 Io 多路复用，起子进程初始化

13. 介绍 Redis 下的分布式锁实现原理、优势劣势和使用场景

利用 Redis 的单线程特性对共享资源进行串行化处理存在问题

单机：无法保证高可用

主--从：无法保证数据的强一致性，在主机宕机时会造成锁的重复获得

分布式锁的实现方式，基于 Redis 的 set 实现分布式锁，基于 zookeeper 临时节点的分布式锁

	Redis	zookeeper	etcd
一致性算法	无	paxos (ZAB)	raft
CAP	AP	CP	CP
高可用	主从集群	n+1 (n至少为2)	n+1
接口类型	客户端	客户端	http/grpc
实现	setNX	createEphemeral	restful API

14. Redis-Cluster 和 Redis 主从+哨兵的区别，你认为哪个更好，为什么

Redis-Cluster

架构细节:

(1)所有的 redis 主节点彼此互联(PING-PONG 机制),内部使用二进制协议优化传输速度和带宽.

(2)节点的 fail 是通过集群中超过半数的节点检测失效时才生效.

(3)客户端与 redis 节点直连,不需要中间 proxy 层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可

(4)redis-cluster 把所有的物理节点映射到[0-16383]slot 上,cluster 负责维护 node<->slot<->value，Redis 集群中内置了 16384 个哈希槽，当需要在 Redis 集群中放置一个 key-value 时,redis 先对 key 使用 crc16 算法算出一个结果，然后把结果对 16384 求余数，这样每个 key 都会对应

一个编号在

0-16383 之间的哈希槽，redis 会根据节点数量大致均等的将哈希槽映射到不同的节点

1、主节点投票，如果超过半数的主都认为某主 down 了，则该主就 down 了 （主选择单数）

2、主节点投票，选出挂了的主的从升级为主

集群挂了的情况：

1、半数的主挂了，不能投票生效，则集群挂了

2、挂了的主机的从也挂了，造成 slot 槽分配不连续（16384 不能完全分配），集群就挂了

哨兵进程的作用

监控(Monitoring):哨兵(sentinel) 会不断地检查你的 Master 和 Slave 是否运作正常。

提醒(Notification)： 当被监控的某个 Redis 节点出现问题时, 哨兵(sentinel) 可以通过 API 向管理员或者其他应用程序发送通知。

自动故障迁移(Automatic failover)：当一个 Master 不能正常工作时，哨兵(sentinel) 会开始一次自动故障迁移操作

15. Redis 的主从同步机制是什么?

Redis 的主从同步，分为**全量同步**和**增量同步**。

只有从机第一次连接上主机是全量同步。

断线重连有可能触发全量同步也有可能是增量同步（ master 判断 runid 是否一致）。除此之外的情况都是增量同步

同步快照阶段： Master 创建并发送快照 RDB 给 Slave ， Slave 载入并解析快照。Master 同时将此阶段所产生的新的写命令存储到缓冲区。

同步写缓冲阶段： Master 向 Slave 同步存储在缓冲区的写操作命令。

同步增量阶段： Master 向 Slave 同步写操作命令。

增量同步

Redis 增量同步主要指** Slave 完成初始化后开始正常工作时，Master 发生的写操作同步到 Slave`的过程。

通常情况下， Master 每执行一个写命令就会向 Slave 发送相同的写命令，然后 Slave 接收并执行。

16. 什么情况下会造成缓存穿透，如何解决？

缓存穿透：一般的缓存系统，都是按照 key 去缓存查询，如果不存在对应的 value，就应该去后端系统查找（比如 DB）。如果 key 对应的 value 是一定不存在的，并且对该 key 并发请求量很大，就会对后端系统造成很大的压力。

也就是说，对不存在的 key 进行高并发访问，导致数据库压力瞬间增大，这就叫做【缓存穿透】。

解决方案：对查询结果为空的情况也进行缓存，缓存时间设置短一点，或者该 key 对应的数据 insert 了之后清理缓存。

缓存雪崩：当缓存服务器重启或者大量缓存集中在某一个时间段失效，这样在失效的时候，也会给后端系统(比如 DB)带来很大压力。突然间大量的 key 失效了或 redis 重启，大量访问数据库

解决方案:1、 key 的失效期分散开 不同的 key 设置不同的有效期 2、设置二级缓存 3、高可用缓存击穿对于一些设置了过期时间的 key，如果这些

key 可能会在某些时间点被超高并发地访问，是一种非常“热点”的数据。这个时候，需要考虑一个问题：缓存被“击穿”的问题，这个和缓存雪崩的区别在于这里针对某一 key 缓存，前者则是很多 key。缓存在某个时间点过期的时候，恰好在这个时间点对这个 Key 有大量的并发请求过来，这些请求发现缓存过期一般都会从后端 DB 加载数据并回设到缓存，这个时候大并发的请求可能会瞬间把后端 DB 压垮。

解决方案：用分布式锁控制访问的线程，使用 redis 的 setnx 互斥锁先进行判断，这样其他线程就处于等待状态，保证不会有大并发操作去操作数据库。

数据写：数据不一致的根源 ： 数据源不一样

如何解决：强一致性很难，追求最终一致性，互联网业务数据处理的特点

高吞吐量 低延迟 数据敏感性低于金融业时序控制是否可行？

先更新数据库再更新缓存或者先更新缓存再更新数据库本质上不是一个原子操作，所以时序控制不可行

保证数据的最终一致性(延时双删)：

1、先更新数据库同时删除缓存项(key)，等读的时候再填充缓存

2、2 秒后再删除一次缓存项(key)

3、设置缓存过期时间 Expired Time 比如 10 秒 或 1 小时

4、将缓存删除失败记录到日志中，利用脚本提取失败记录再次删除（缓存失效期过长 7*24）

16. MySQL 里有 2000w 数据，Redis 中只存 20w 的数据，如何保证 Redis 中的数据都是热点数据？

LRU

17. MySQL 与 MongoDB 之间最基本的差别是什么?

传统的 RDBMS 其实使用 Table 的格式将数据逻辑地存储在一张二维的表中，其中不包括任何复杂的数据结构，但是由于 MongoDB 支持嵌入文档、数组和哈希等多种复杂数据结构的使用，所以它最终将所有的数据以 BSON 的数据格式存储起来。

RDBMS 和 MongoDB 中的概念都有着相互对应的关系，数据库、表、行和索引的概念在两中数据库中都非常相似，唯独最后的 JOIN 和 Embedded Document 或者 Reference 有着巨大的差别。这一点差别其实也影响了在使用 MongoDB 时对集合（Collection）Schema 的设计，如果我们在 MongoDB 中遵循了与 RDBMS 中相同的思想对 Collection 进行设计，那么就不可避免的使用很多的 “JOIN” 语句，而 MongoDB 是不支持 “JOIN” 的，在应用内做这种查询的性能非常非常差，在这时使用嵌入式的文档其实就可以解决这种问题了，嵌入式的文档虽然可能会造成很多的数据冗余导致我们在更新时会很痛苦，但是查询时确实非常迅速。

17. MongoDB 的适用场景有哪些？

默认采用 WiredTiger 存储引擎、

Transport Layer 业务层

Transport Layer 是处理请求的基本单位。Mongo 有专门的 listener 线程，每次有连接进来，listener 会创建一个新的线程 conn 负责与客户端交互，它把具体的查询请求交给 network 线程，真正到数据库里查询由 TaskExecutor 来进行。

WiredTiger 的写操作会默认写入 Cache ，并持久化到 WAL (Write Ahead Log)，每 60s 或 Log 文件达到 2G 做一次 checkpoint ，产生快照文件 Journaling

为了在数据库宕机保证 MongoDB 中数据的持久性，MongoDB 使用了 Write Ahead Logging 向磁盘上的 journal 文件预先进行写入；除了 journal 日志，MongoDB 还使用检查点（Checkpoint）来保证数据的一致性，当数据库发生宕机时，我们就需要 Checkpoint 和 journal 文件协作完成数据的恢复工作：

一致性 WiredTiger 使用 Copy on Write 管理修改操作。修改先放在 cache 中，并持久化，不直接作用在原 leaf page，而是写入新分配的 page，每次 checkpoint 产生新 page。

18.什么是副本集?有哪几种有什么区别

有仲裁和无仲裁，主从不能自动切，副本集可以自动切

19. ObjectId 的组成?

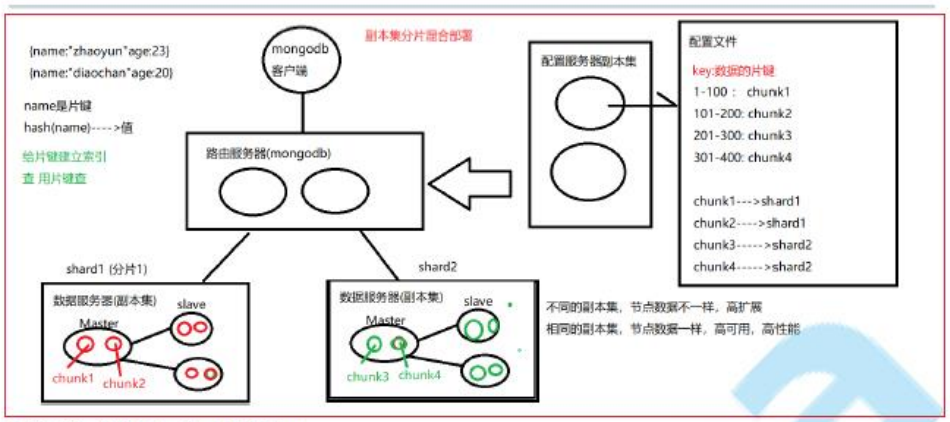
12 个字节（4 字节时间戳+3 字节机器 ID+2 字节进程 ID+3 字节计数器）

20. MongoDB 中的索引是什么?如何使用

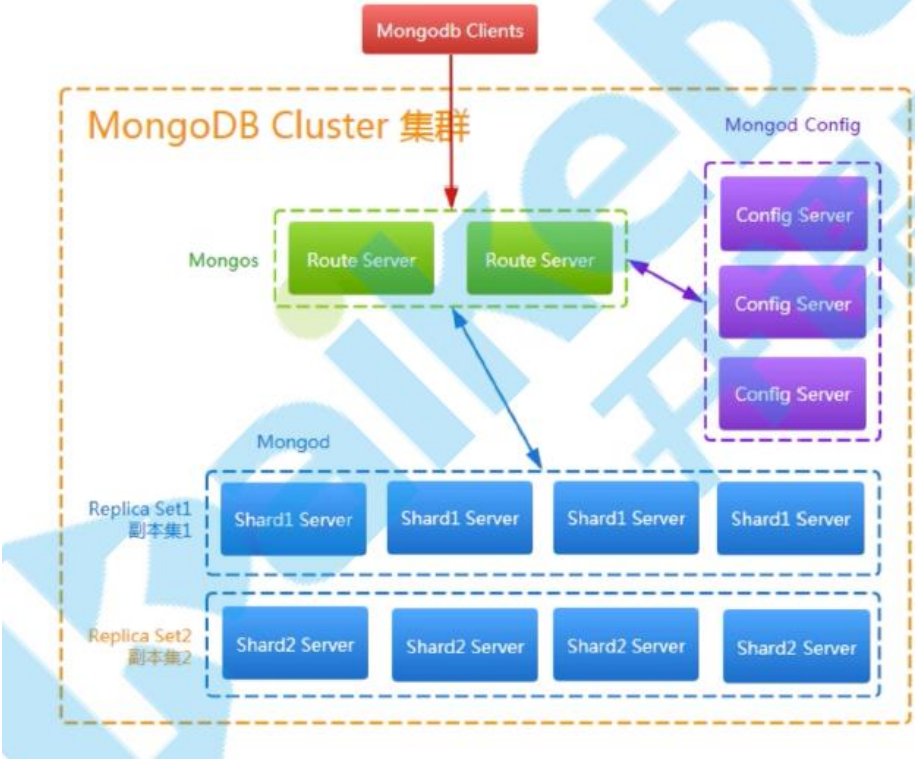
索引是特殊的数据结构，索引存储在一个易于遍历读取的数据集合中，索引是对数据库表中一列或多列的值进行排序的一种结构

db.collection.createIndex() 方法

21.MongoDB 中的分片是如何实现的？



副本集与分片混合部署方式如图：



22. 什么是 MongoDB 的混合集群，有什么优势

三高（高可用（主从），高扩展（分片），高性能（读写分离））