**Redis**

▶**缓存原理&设计**

◇缓存的读写模式

•Cache Aside Pattern

读的时候，先读缓存，缓存没有的话，就读数据库，然后取出数据后放入缓存，同时返回 响应。更新的时候，先更新数据库，然后再删除缓存。

•Read/Write Through Pattern

Read-Through（穿透读模式/直读模式）：应用程序读缓存，缓存没有，由缓存回源到数据库， 并写入缓存。

Write-Through（穿透写模式/直写模式）：应用程序写缓存，缓存写数据库。

•Write Behind Caching Pattern

缓存通过异步的方式将数据批量或合并后更新到DB中

◇缓存问题

•缓存穿透

缓存穿透是指在高并发下查询key不存在的数据，会穿过缓存查询数据库。导致数据库压 力过大而宕机

•解决方案

使用布隆过滤器

对查询结果为空的情况也进行缓存

•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

•解决方案

○key的失效期分散开 不同的key设置不同的有效期

○设置二级缓存（数据不一定一致）

○限流

○集群

○用分布式锁控制访问的线程

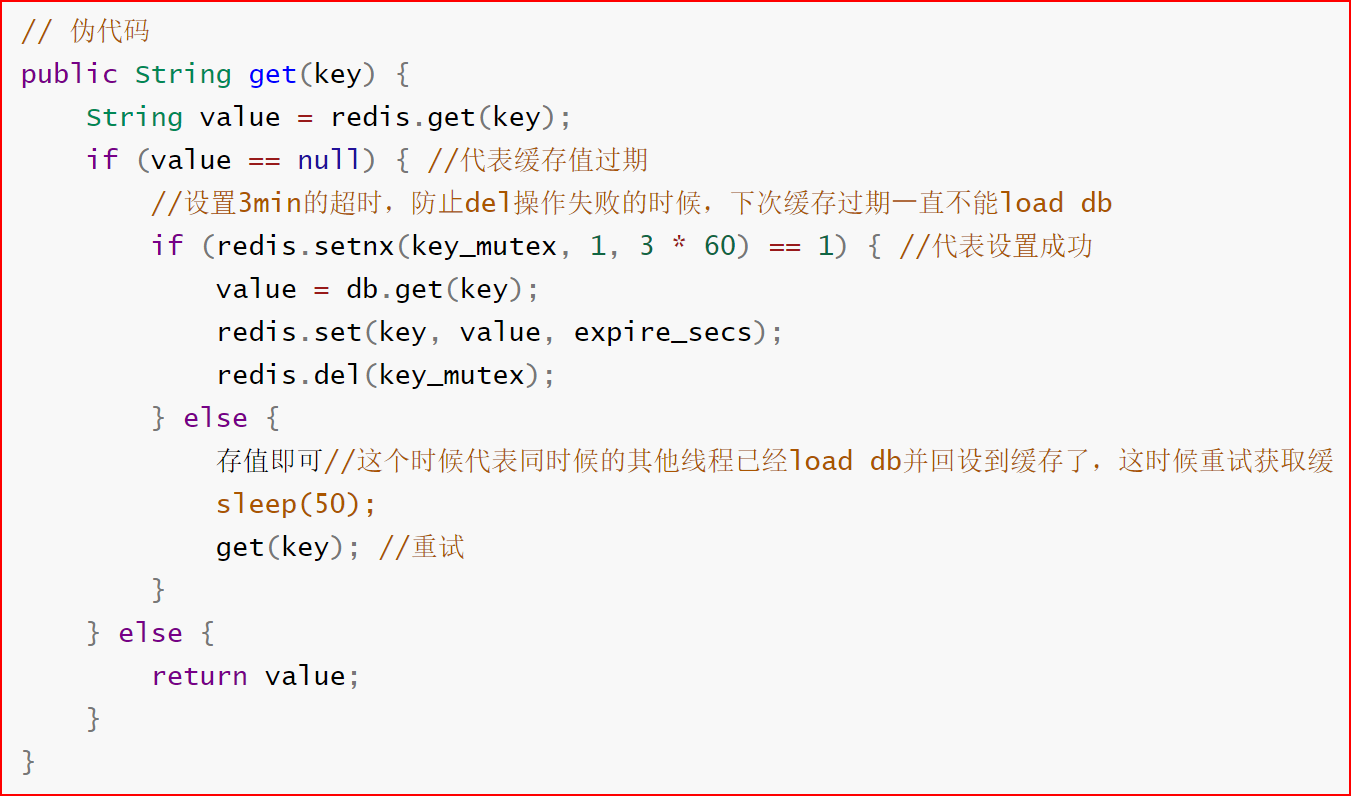


•缓存击穿

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

•解决方案

用分布式锁控制访问的线程



不设超时时间

•Hot Key

当有大量的请求(几十万)访问某个Redis某个key时，由于流量集中达到网络上限，从而导 致这个redis的服务器宕机。造成缓存击穿

•解决方案

变分布式缓存为本地缓存

利用对热点数据访问的限流熔断保护措施

▶**数据类型**

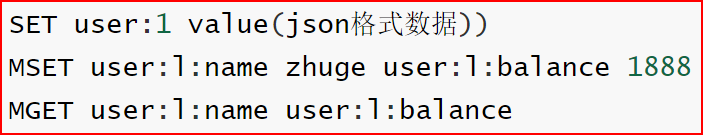
◇key的类型是字符串

◇value的数据类型有:string 、list 、set 、sort set、hash

•string字符串类型

使用场景

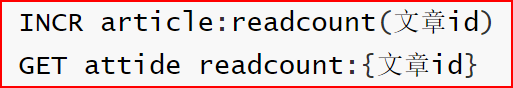
○普通的赋值



Redis Mset 命令用于同时设置一个或多个 key-value 对

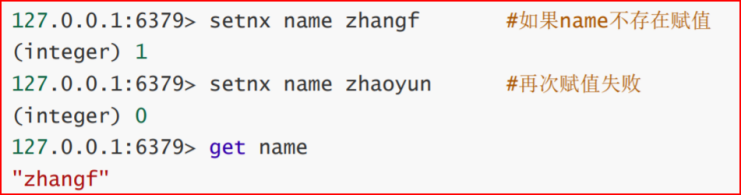
○incr用于乐观锁

incr：递增数字，可用于实现乐观锁 watch(事务)



○setnx用于分布式锁

setnx:



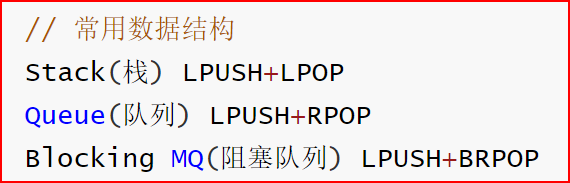
Set：



•list列表类型

使用场景

作为栈或队列使用



可用于各种列表，比如用户列表、商品列表、评论列表等。



特点

存储有序、可重复的元素

获取头部或尾部附近的记录是极快的

list的元素个数最多为2^32-1个（40亿）

•set集合类型

使用场景

集合的操作，并集、交集等

通过spop进行随机抽奖

**微信抽奖：**

1）点击参与抽奖加入集合

SADD key (userID)

2）查看参与抽奖所有用户

SMEMBERS key

3）抽取count名中奖者

SRANDMEMBER key [count] /SPOP key [count]

**微信微博点赞,收藏,标签**

1）点赞

SADD like:{消息ID} {用户ID}

2）取消点赞

SREM like:{消息ID} {用户ID}

3）检查用户是否点过赞

SISMEMBER like:{消息ID} {用户ID}

4）获取点赞的用户列表

SMEMBERS like:{消息ID}

5）获取点赞用户数

SCARD like:{消息ID}

**集合操作实现微博微信关注模型**

1）诸葛老师关注的人:

zhugeSet-> {yangguo, sima, luban}

2）杨过老师关注的人:

yangguoset {zhuge, sima, luban.guojia}

3）司马老师关注的人:

simaSet-> {zhuge, yangguo, guojia, luban, xuny}

4）我和杨过老师共同关注:

SINTER zhugeSet yangguoSet–> {sima, luban}

5）我关注的人也关注他(杨过老师):

SISMEMBER simaSet yangquoSISMEMBER lubanSet yangquo

6）我可能认识的人:

SDIFF yangquoSet zhugeSet->{zhuge, guojia}

特点

无序、唯一元素

集合中最大的成员数为 2^32 - 1

•sortedset有序集合类型

使用场景

排行榜



特点

元素本身是无序不重复的

每个元素关联一个分数(score)

可按分数排序，分数可重复

•hash类型（散列表）

使用场景

对象的存储



特点

一个 string 类型的 field 和 value 的映射表，它提供了字段和字段值的映射。

每个 hash 可以存储 2^32 - 1 键值对（40多亿）。

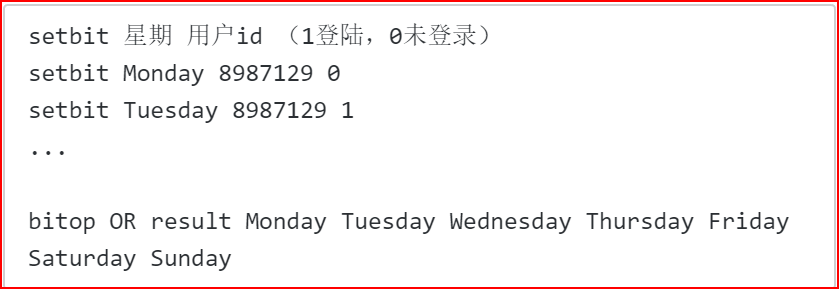
•bitmap位图类型

使用场景

用户签到



统计活跃用户



查询用户在线状态

布隆过滤器

通过一个bit位来表示某个元素对应的值或者状态,其中的key就是对应元素本身

•geo地理位置类型

使用场景

记录地理位置

计算距离

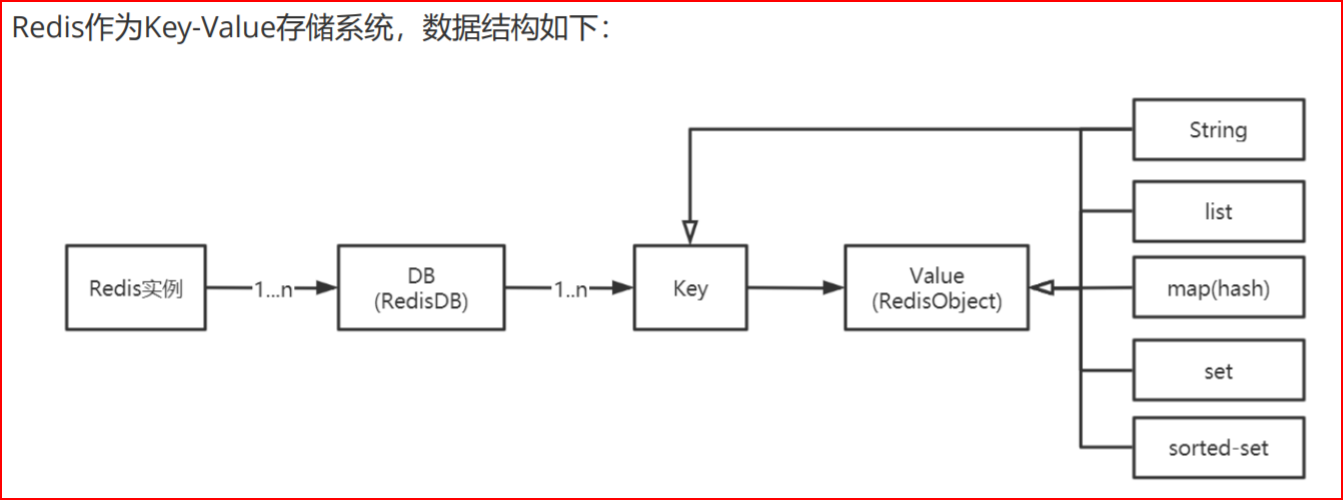
查找"附近的人"

•stream数据流类型

使用场景

可持久化的消息队列。

▶**底层数据结构**



◇RedisDB结构

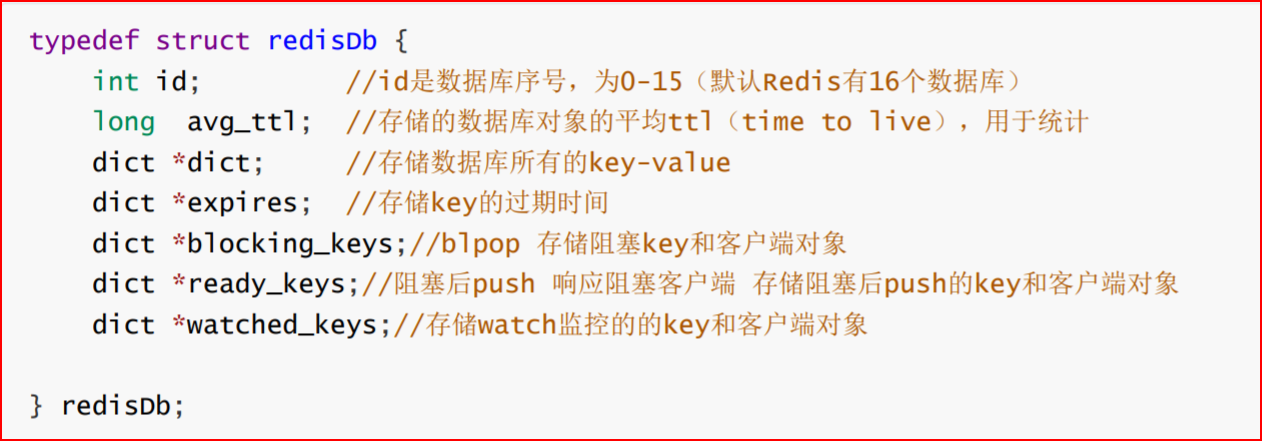
Redis中存在“数据库”的概念，该结构由redis.h中的redisDb定义。

当redis 服务器初始化时，会预先分配 16 个数据库

所有数据库保存到结构 redisServer 的一个成员 redisServer.db 数组中

redisClient中存在一个名叫db的指针指向当前使用的数据库

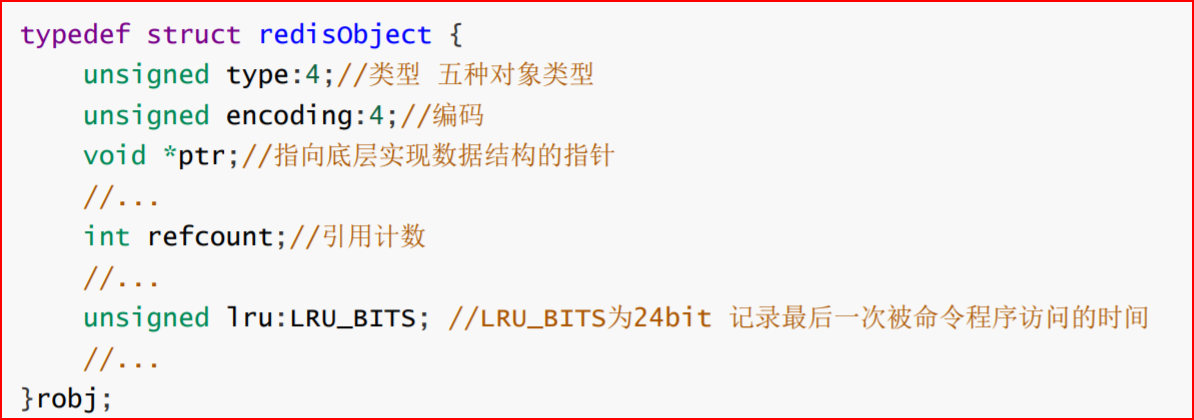
**RedisDB结构体源码**：



◇RedisObject结构

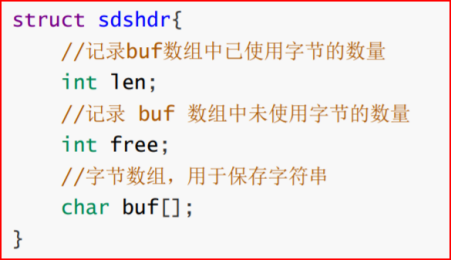
Value是一个对象 包含字符串对象，列表对象，哈希对象，集合对象和有序集合对象

**结构信息概览：**



•7种type

○字符串



Redis 使用了 SDS(Simple Dynamic String)。用于存储字符串和整型数据。

SDS 由于记录了长度，在可能造成缓冲区溢出时，自动重新分配内存，杜绝缓冲区溢 出。

应用场景

存储字符串和整型数据、存储key、AOF缓冲区和用户输入缓冲。

○跳跃表

将有序链表中的部分节点分层，每一层都是一个有序链表

特点

每层都是一个有序链表

查找次数近似于层数（1/2）

底层包含所有元素

空间复杂度 O(n) 扩充了一倍

应用场景

有序集合的实现

○字典

字典dict又称散列表（hash），是用来存储键值对的一种数据结构。

实现

字典(dict)、Hash表(dictht)、Hash表节点(dictEntry)。

○压缩列表

一系列特殊编码的连续内存块组成的顺序型数据结构。

应用场景

sorted-set和hash元素个数少且是小整数或短字符串（直接使用）

list用快速链表(quicklist)数据结构存储，而快速链表是双向列表与压缩列表的组 合。（间接使用）

特点

节省空间

实现

一个字节数组，可以包含多个节点（entry）。每个节点可以保存一个字节数组或 一个整数。

○整数集合

一个有序的（整数升序）、存储整数的连续存储结构

○快速列表

quicklist是一个双向链表，链表中的每个节点是一个ziplist结构。quicklist中的每个节 点ziplist都能够存储多个数据元素。

（在Redis3.2之前，Redis采用双向链表（adlist）和压缩列表（ziplist）实现。）在Redis3.2 以后结合adlist和ziplist的优势Redis设计出了quicklist

应用场景

列表(List)的底层实现、发布与订阅、慢查询、监视器等功能。

○流对象

消息、生产者、消费者和消费组构成。

▶**缓存过期和淘汰策略**

◇Redis的过期策略

•定时删除

在设置键的过期时间的同时，创建一个定时器，让定时器在键的过期时间来临时，立即执 行对键的删除 操作。

•惰性删除

在key被访问时如果发现它已经失效，那么就删除它。

◇定期+惰性都没有删除过期的key怎么办

•内存淘汰机制

○LRU

volatile-lru 从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰

allkeys-lru 从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰

LRU (Least recently used) 最近最少使用，算法根据数据的历史访问记录来进行淘汰数 据，其核心思想：

1. 新数据插入到链表头部；
2. 每当缓存命中（即缓存数据被访问），则将数据移到链表头部；
3. 当链表满的时候，将链表尾部的数据丢弃。
4. 在Java中可以使用LinkHashMap（哈希链表）去实现LRU

○LFU

LFU (Least frequently used) 最不经常使用，如果一个数据在最近一段时间内使用次数 很少，那么在将来一段时间内被使用的可能性也很小

计数器

○random

从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

○ttl

从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

○noenviction

禁止驱逐数据，不删除 默认

◇如何保证redis中存放的都是热点数据

当redis使用的内存超过了设置的最大内存时，会触发redis的数据淘汰策略，设置淘汰机制。

▶**哨兵模式**

由一个或多个sentinel实例组成sentinel集群可以监视一个或多个主服务器和多个从服务器。

当主服务器进入下线状态时，sentinel可以将该主服务器下的某一从服务器升级为主服务器继续提供 服务，从而保证redis的高可用性。

◇哨兵leader选举算法Raft

分布式系统一致性问题的协议

Raft协议描述的节点共有三种状态：Leader, Follower, Candidate。

◇Sentinel的leader选举流程

1、某Sentinel认定master客观下线后，当前Sentinel会先看看自己有没有投过票，如果自己已 经投过票给其他Sentinel，在一定时间内自己就不会成为Leader。

2、如果当前Sentinel还没投过票，那么它就成为Candidate。

3、Sentinel需要完成几件事情：

更新故障转移状态为start

当前epoch加1，相当于进入一个新term，在Sentinel中epoch就是Raft协议中的term。

向其他节点发送 is-master-down-by-addr 命令请求投票。命令会带上自己的epoch。

给自己投一票（leader、leader\_epoch）

4、当其它哨兵收到此命令时，可以同意或者拒绝它成为领导者；（通过判断epoch）

5、Candidate会不断的统计自己的票数，直到他发现认同他成为Leader的票数超过一半而且超 过它配置的quorum，这时它就成为了Leader。

6、其他Sentinel等待Leader从slave选出master后，检测到新的master正常工作后，就会去掉 客观下线的标识

◇哨兵是怎样实现修改redis配置文件的

•sentinel将选择出来的新的主机，修改redis.conf 去掉repliaof或slaveof

•sentinel自己的sentinel.conf 修改sentinel monitor mymaster 新的主机IP和端口

•如果原来的主机重新启动，则sentinel修改它的redis.conf加上repliaof 新的主机和端口或 slaveof新的主机和端口

◇故障转移

1. 它会将失效 Master 的其中一个 Slave 升级为新的 Master , 并让失效 Master 的其他 Slave 改为复制新的 Master ；

2. 当客户端试图连接失效的 Master 时，集群也会向客户端返回新 Master 的地址，使得集群 可以使用现在的 Master 替换失效 Master

3. Master 和 Slave 服务器切换后， Master 的 redis.conf 、 Slave 的 redis.conf 和sentinel.conf 的配置文件的内容都会发生相应的改变，即， Master 主服务器的 redis.conf配置文件中会多 一行 replicaof 的配置， sentinel.conf 的监控目标会随之调换

▶**Redis持久化**

◇**RDB**

•RDB（Redis DataBase），是redis默认的存储方式，RDB方式是通过快照（snapshotting）完成 的。

•触发快照的方式

○符合自定义配置的快照规则

○执行save或者bgsave命令

○执行flushall命令

○执行主从复制操作 (第一次)

•执行流程

○Redis父进程首先判断：当前是否在执行save，或bgsave/bgrewriteaof（aof文件重写命令） 的子进程，如果在执行则bgsave命令直接返回。

○父进程执行fork（调用OS函数复制主进程）操作创建子进程，这个过程中父进程是阻塞 的，Redis 不能执行来自客户端的任何命令。

○父进程fork后，bgsave命令返回”Background saving started”信息并不再阻塞父进程， 并可以响应其他命令。

○子进程创建RDB文件，根据父进程内存快照生成临时快照文件，完成后对原有文件进行 原子替换。（RDB始终完整）

○子进程发送信号给父进程表示完成，父进程更新统计信息。

○父进程fork子进程后，继续工作。

•优缺点

○优点

RDB是二进制压缩文件，占用空间小，便于传输（传给slaver） 主进程fork子进程， 可以最大化Redis性能，主进程不能太大，复制过程中主进程阻塞

○缺点

不保证数据完整性，会丢失最后一次快照以后更改的所有数据

◇**AOF**

AOF（append only file）是Redis的另一种持久化方式。Redis默认情况下是不开启的。开启AOF 持久化后 Redis 将所有对数据库进行过写入的命令（及其参数）（RESP）记录到 AOF 文件， 以 此达到记录数据库状态的目的

•原理

命令传播：Redis 将执行完的命令、命令的参数、命令的参数个数等信息发送到 AOF 程序 中。

缓存追加：AOF 程序根据接收到的命令数据，将命令转换为网络通讯协议的格式，然后将 协议内容追加 到服务器的 AOF 缓存中。

文件写入和保存：AOF 缓存中的内容被写入到 AOF 文件末尾，如果设定的 AOF 保存条件 被满足的话， fsync 函数或者fdatasync函数会被调用，将写入的内容真正地保存到磁盘中。

•保存模式

AOF\_FSYNC\_NO ：不保存。

AOF\_FSYNC\_EVERYSEC ：每一秒钟保存一次。（默认）

AOF\_FSYNC\_ALWAYS ：每执行一个命令保存一次。（不推荐）

•AOF重写

○解决AOF文件体积膨胀的问题

○作用

减少磁盘占用量

加速数据恢复

○BGREWRITEAOF命令实现原理

◾执行命令，同时将命令追加到AOF缓冲区和AOF重写缓冲区

◾当AOF子进程重写完成后，发送一个信号给父进程，父进程将执行AOF重写缓冲区 中的所有内容写入到新AOF文件中，新AOF文件保存的数据库状态将和服务器当 前的数据库状态一致。

◾对新的AOF文件进行改名，原子性地覆盖现有AOF文件，完成新旧两个AOF文件 替换处理完成。

▶**集群与分区**

◇分区算法

•范围

好处： 实现简单，方便迁移和扩展

缺陷： 热点数据分布不均，性能损失

•Hash

好处： 支持任何类型的key 热点分布较均匀，性能较好

缺陷： 迁移复杂，需要重新计算，扩展较差（利用一致性hash环）

•一致性hash

一致性hash是对2^32（4 294 967 296）取模，hash环

hash（服务器的IP地址） % 2^32

缺点

复杂度高

不易扩展

◇proxy端分区

•在客户端和服务器端引入一个代理或代理集群，客户端将命令发送到代理上，由代理根据算法， 将命令 路由到相应的服务器上。常见的代理有Codis（豌豆荚）和TwemProxy（Twitter）

•原理：

Codis 将所有的 key 默认划分为 1024 个槽位(slot)，它首先对客户端传过来的 key 进行 crc32 运算计算 哈希值，再将 hash 后的整数值对 1024 这个整数进行取模得到一个余数， 这个余数就是对应 key 的槽位。

Codis的槽位和分组的映射关系就保存在codis proxy当中。

•实例之间槽位同步：

codis proxy存在单点问题，需要做集群。

不同proxy之间需要同步映射关系

在Codis中使用的是Zookeeper（Etcd）来保存映射关系

Codis 将槽位关系存储在 zk 中，并且提供了一个 Dashboard 可以用来观察和修改槽位关 系，当槽位关 系变化时，Codis Proxy 会监听到变化并重新同步槽位关系，从而实现多个 Codis Proxy 之间共享相同 的槽位关系配置

•优点：

对客户端透明,与codis交互方式和redis本身交互一样

支持在线数据迁移,迁移过程对客户端透明有简单的管理和监控界面

支持高可用,无论是redis数据存储还是代理节点

自动进行数据的均衡分配

最大支持1024个redis实例,存储容量海量

高性能

•缺点：

采用自有的redis分支,不能与原版的redis保持同步

如果codis的proxy只有一个的情况下, redis的性能会下降20%左右

某些命令不支持

◇RedisCluster

•方案采用去中心化的方式，包括：sharding（分区）、replication（复制）、failover（故障转移）。

•原理

○Redis的Cluster中，一共分成16384个槽，根据key进行散列，分配到这16384个slot 中的某一个中。使用的hash算法，就是CRC16后16384取模。Redis集群中的每个node 节点负责分摊这16384个slot中的一部分。

○当动态添加或减少node节点时，需要将16384个槽做个再分配，槽中的键值也要迁移。

○node配置成主从结构，即一个master主节点，挂n个slave从节点。这时，如果主节点 失效，Redis Cluster会根据选举算法从slave节点中选择一个上升为主节点，整个集群继 续对外提供服务。

•优势

○高性能

Redis Cluster 的性能与单节点部署是同级别的。多主节点、负载均衡、读写分离

○高可用

Redis Cluster 支持标准的 主从复制配置来保障高可用和高可靠

○failover故障转移

Redis Cluster 也实现了一个类似 Raft 的共识方式，来保障整个集群的可用性

○易扩展

向 Redis Cluster 中添加新节点，或者移除节点，都是透明的，不需要停机。

水平、垂直方向都非常容易扩展。

数据分区，海量数据，数据存储

▶**Redis 主从同步**

在Redis 2.8之后使用PSYNC命令，具备完整重同步和部分重同步模式。

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）。

查看集群中槽占用情况：cluster nodes

◇**全量同步**

•从服务器连接主服务器，发送SYNC命令；

•主服务器接收到SYNC命名后，开始执行BGSAVE命令生成RDB文件并使用缓冲区记录此后执 行的所有写命令；

•主服务器BGSAVE执行完后，向所有从服务器发送快照文件，并在发送期间继续记录被执行的 写命令；

•从服务器收到快照文件后丢弃所有旧数据，载入收到的快照；

•主服务器快照发送完毕后开始向从服务器发送缓冲区中的写命令；

•从服务器完成对快照的载入，开始接收命令请求，并执行来自主服务器缓冲区的写命令；

◇**增量同步**

•增量复制的过程主要是主服务器每执行一个写命令就会向从服务器发送相同的写命令，从服务 器接收并执行收到的写命令。

◇**主从同步策略**

主从刚刚连接的时候，进行全量同步；全同步结束后，进行增量同步。

当然，如果有需要，slave 在任何时候都可以发起全量同步。redis 策略是，无论如何，首先会 尝试进行增量同步，如不成功，要求从机进行全量同步。

◇gossip 协议

redis 维护集群元数据采用的是gossip 协议，所有节点都持有一份元数据，不同的节点如果出现 了元数据的变更，就不断将元数据发送给其它的节点，让其它节点也进行元数据的变更。

•优缺点

○优点

元数据的更新比较分散，不是集中在一个地方，降低了压力；

○缺点

元数据的更新有延时，可能导致集群中的一些操作会有一些滞后。

•gossip 协议包含多种消息

○meet

某个节点在内部发送了一个gossip meet 消息给新加入的节点，通知那个节点去加入 我们的集群。然后新节点就会加入到集群的通信中

○ping

每个节点都会频繁给其它节点发送 ping，其中包含自己的状态还有自己维护的集群元 数据，互相通过 ping 交换元数据。

○pong

ping 和 meet消息的返回响应，包含自己的状态和其它信息，也用于信息广播和更新。

○fail

某个节点判断另一个节点 fail 之后，就发送 fail 给其它节点，通知其它节点说这个 节点已宕机

▶**扩展功能**

◇发布与订阅

Redis的发布订阅机制包括三个部分，publisher，subscriber和Channel。

发布者和订阅者都是Redis客户端，Channel则为Redis服务器端。

发布者将消息发送到某个的频道，订阅了这个频道的订阅者就能接收到这条消息。

•使用场景

哨兵模式

哨兵通过发布与订阅的方式与Redis主从服务器进行通信。

Redisson框架使用

在Redisson分布式锁释放的时候，是使用发布与订阅的方式通知的

◇事务

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

•事务命令

multi：用于标记事务块的开始,Redis会将后续的命令逐个放入队列中，然后使用exec原子 化地执行这个命令队列

exec：执行命令队列

discard：清除命令队列

watch：监视key

unwatch：清除监视key

•Redis不支持事务回滚

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

为了性能方面就忽略了事务回滚

◇慢查询

慢查询定位

slowlog get 可以获得执行较慢的redis命令

◇RedLock（Redis Distributed Lock;即使用redis实现的分布式锁）

•原理

在多个redis实例上创建锁，n / 2+ 1，必须在大多数redis节点上都成功创建锁，才能算这 个整体的RedLock加锁成功

•存在的问题

○故障重启后带来的锁的安全性的问题

我们一共有 A、B、C 这三个节点。

1. 客户端 1 在 A，B 上加锁成功。C 上加锁失败。

2. 这时节点 B 崩溃重启了，但是由于持久化策略导致客户端 1 在 B 上的锁没有持 久化下来。

3. 客户端 2 发起申请同一把锁的操作，在 B，C 上加锁成功。

4. 这个时候就又出现同一把锁，同时被客户端 1 和客户端 2 所持有了。

○解决

延迟重启：一个节点崩溃后，不要立即重启它，而是等待一定的时间后再重启。

○一个严重依赖系统时钟的分布式锁问题

1. 客户端 1 从 Redis 节点 A, B, C 成功获取了锁。由于网络问题，无法访问 D 和 E。

2. 节点 C 上的时钟发生了向前跳跃，导致它上面维护的锁过期了。

3. 客户端 2 从 Redis 节点 C, D, E 成功获取了同一个资源的锁。由于网络问题，无 法访问 A 和 B。

4. 现在，客户端 1 和客户端 2 都认为自己持有了锁。

▶**常见问题**

•keys \* 慎用

数据量达到几百万，keys这个指令就会导致 Redis 服务卡顿

解决

scan 0 match user\_token\* count 5

scan 游标 MATCH <返回和给定模式相匹配的元素> count 每次迭代所返回的元素数量

•为什么Redis是单线程的，却能有很好的性能

纯内存操作

单线程操作，避免了频繁的上下文切换

合理高效的数据结构

多路 I/O 复用机制

单个线程中通过记录跟踪每一个socket的状态来管理多个I/O流，这里“多路”指的是多 个网络连接socket，“复用”指的是复用同一个线程。采用多路 I/O 复用技术可以让单个线 程高效的处理多个连接请求

•批量操作

原生命令：例如mget、mset

非原生命令：pipeline

两者不同

1.原生是原子操作，pipeline是非原子操作。

2.pipeline可以打包不同的命令，原生做不到

3.pipeline需要客户端和服务端同时支持

•Redis性能调优

○避免耗时指令

通过机制严格控制Hash、Set、Sorted Set的大小

将排序、并集、交集等操作放在调用端执行

禁止使用KEYS命令,使用SCAN类的命令进行分批的，游标式的遍历

开启自动记录耗时较长的命令的功能 （SLOWLOG GET count）

○避免延迟

◾网络引发的延迟

◽使用 unix 进程间通信，服务不是很大，考虑单机部署

◽使用 multi-key 指令合并多个指令，减少请求数，如果有可能的话

◽使用 transaction、script 合并 requests 以及 responses

◽使用 pipeline 合并 response

◾数据持久化引发的延迟

Redis的数据持久化工作本身就会带来延迟，需要根据数据的安全级别和性能要求制 定合理的持久化策略

◾Swap引发的延迟

当Linux将Redis所用的内存分页移至swap空间时，将会阻塞Redis进程，导致Redis 出现不正常的延迟。/proc/<pid>/smaps文件中会保存进程的swap记录，通过查看这 个文件，能够判断Redis的延迟是否由Swap产生。

◾数据淘汰引发的延迟

当同一秒内有大量key过期时，也会引发Redis的延迟。在使用时应尽量将key的失 效时间错开；或者用del的异步版本；或者给 keys 的过期时间设置一个随机波动量

•Redis缓存和MySQL数据⼀致性

○延时双删策略

具体的步骤

先删除缓存

再写数据库

用分布式锁控制访问的线程



休眠500毫秒

再次删除缓存

○异步更新缓存保证缓存和mysql数据⼀致

canal

•Redis哨兵选举故障转移与集群选举故障转移区别

○哨兵选举与故障转移

主要针对非集群模式下主从模式，由哨兵负责。哨兵发现主下线后，哨兵发起选举，其余 哨兵负责投票，选出一个哨兵作为故障转移的Leader, 由这个leader哨兵完成故障转移

○集群选举与故障转移

主要针对集群模式下的主从，由主的从负责。一个从发现主下线后，主发起选举，其他有 资格的主（有槽的）负责投票，选出一个从作为新主，新主完成故障转移包括槽重新分配。

•使用过Redis分布式锁么，它是什么回事？

jedis.set(String key, String value, String nx, String expx, int time)，这个set()方法一共有五个形参：

示例：set key value NX PX timeout

•假如Redis里面有1亿个key，其中有10w个key是以某个固定的已知的前缀开头(比如 pre)的，如 果将它们全部找出来？

使用keys指令可以扫出指定模式的key列表：keys pre\*

•如果这个redis正在给线上的业务提供服务，那使用keys指令会有什么问题？

这个时候你要回答redis关键的一个特性：redis的单线程的。keys指令会导致线程阻塞一段时间， 线上服务会停顿，直到指令执行完毕，服务才能恢复。

这个时候可以使用scan指令，scan指令可以无阻塞的提取出指定模式的key列表，但是会有一 定的重复概率，在客户端做一次去重就可以了，但是整体所花费的时间会比直接用keys指令长。

•使用过Redis做异步队列么，你是怎么用的？

一般使用list结构作为队列，rpush生产消息，lpop消费消息。当lpop没有消息的时候，要适当 sleep 一会再重试。

可不可以不用sleep呢？list还有个指令叫blpop，在没有消息的时候，它会阻塞住直到消息到来。

能不能生产一次消费多次呢？使用pub/sub主题订阅者模式，可以实现1:N的消息队列。

pub/sub有什么缺点？在消费者下线的情况下，生产的消息会丢失，得使用专业的消息队列如 rabbitmq等。redis中pub/sub缺陷

•redis如何实现延时队列？

使用有序集合，拿时间戳作为score，消息内容作为key调用zadd来生产消息，消费者用 zrangebyscore指令获取N秒之前的数据轮询进行处理。

•如果有大量的key需要设置同一时间过期，一般需要注意什么

如果大量的key过期时间设置的过于集中，到过期的那个时间点，redis可能会出现短暂的卡顿 现象。一般需要在时间上加一个随机值，使得过期时间分散一些。

•Pipeline有什么好处，为什么要用pipeline？

可以将多次IO往返的时间缩减为一次，前提是pipeline执行的指令之间没有因果相关性。使用 redis-benchmark进行压测的时候可以发现影响redis的QPS峰值的一个重要因素是pipeline批次 指令的数目。

•基于缓存（Redis等）实现分布式锁

1. 使用命令介绍：

（1）SETNX

SETNX key val：当且仅当key不存在时，set一个key为val的字符串，返回1；若key存在， 则什么都不做，返回0。

（2）expire

expire key timeout：为key设置一个超时时间，单位为second，超过这个时间锁会自动释放， 避免死锁。

（3）delete

delete key：删除key

在使用Redis实现分布式锁的时候，主要就会使用到这三个命令。

2. 实现思想：

（1）获取锁的时候，使用setnx加锁，并使用expire命令为锁添加一个超时时间，超过该时间 则自动释放锁，锁的value值为一个随机生成的UUID，通过此在释放锁的时候进行判断。

（2）获取锁的时候还设置一个获取的超时时间，若超过这个时间则放弃获取锁。

（3）释放锁的时候，通过UUID判断是不是该锁，若是该锁，则执行delete进行锁释放。