

```
返回满足给定 pattern 的所有key
         语法 keys pattern
        删除对应 key 的数据
               DEL key
         确认一个 key 是否存在
                 exists key
         设置 key 的生存时间
                 EXPIRE key seconds
                           查看 key 生于的生存时间
                 PERSIST key
                              清除生存时间
                 PEXPIRE key milliseconds
                                     生存时间设置单位为: 毫秒
          重命名 key
          语法 rename oldkey newkey
         显示指定key的数据类型
                type key
                    1.定时删除:几乎不用
                    2.惰性删除:查询Key的时候删除
                    3.主动删除:redis.conf中配置主动删除策略
       过期删除策略
                    Redis采用的过期策略:惰性删除+主动删除
                                       过期key对RDB没有任何影响
                    持久化对过期key的处理
                                      过期key对AOF没有任何影响
                    noeviction:返回错误,不会驱逐任何键(默认策略,不建议使用)
                    allkeys-lru:通过LRU算法驱逐最久没有使用的键
                    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-random:加入键的时候如果过限,从所有key随机删除
                    volatile-random:从已设置过期时间的数据集(server.db[i].expires)中任意
        内存淘汰策略
                    选择数据淘汰
                    volatile-ttl:从配置了过期时间的键中驱逐马上就要过期的键
                    volatile-lfu:从所有配置了过期时间的键中驱逐使用频率最少的键
                    allkeys-lfu:从数据集(server.db[i].dict)中任意选择数据淘汰
                                          LRU 数据淘汰机制是这样的:在数据集中随机挑选几个键值对,取出其中 Iru
                    LRU(Least recently used)算法
                                          最常见的实现是使用一个链表保存缓存数据
       redis 宕机后数据会消失,持久化可以帮助 redis 重启后快速恢复数据,而不
       是为了存储数据
原因
                       全称 Redis DataBase,是 redis 默认的存储方式,通过快照
                       (snapshotting) 完成
               特点
                       只关注这一刻的数据,不关注过程
                                                 save "" # 不使用RDB存储 不能主从
                                                 save 900 1 # 表示15分钟(900秒钟)内至少1个键被更改则进行快照。
                                                 save 300 10 # 表示5分钟(300秒)内至少10个键被更改则进行快照。
                             符合自定义配置的快照规则
                                                 save 60 10000 # 表示1分钟内至少10000个键被更改则进行快照。
                                          该命令会阻塞当前Redis服务器,执行save命令期间,Redis不能处理其他命
                                          令,直到RDB过程完成为止
                             执行save命令
               触发快照的方式
                                            执行该命令时,Redis会在后台异步进行快照操作,快照同时还可以响应客户
                                           端请求。基本上 Redis 内部所有的RDB操作都是采用 bgsave 命令
                             执行bgsave命令
                             执行flushall命令
                             执行主从复制操作 (第一次)
                               有其他子进程正在执
行,直接返回
                                                父进程
                                                              信号通知父进程
                                                                            1. Redis父进程首先判断:当前是否在执行save,或bgsave/bgrewriteaof
                                                 fork
                                                              子进程
                                                                             (aof文件重写命令)的子进程,如果在执行则bgsave命令直接返回。
                                                                            2. 父进程执行fork (调用OS函数复制主进程) 操作创建子进程, 这个复制过
       RDB
                                                                            程中父进程是阻塞的,Redis不能执行来自客户端的任何命令。
                                                                            3. 父进程fork后,bgsave命令返回"Background saving started"信息并
                                               响应其它命令
                                                             生成RDB文件
                                                                            不再阻塞父进程,并可以响应其他命令。
               执行流程 (原理)
                                                                            4. 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)
                                                                            5. 子进程发送信号给父进程表示完成,父进程更新统计信息。
                                                                            6. 父进程fork子进程后,继续工作。
                          1、头部5字节固定为 "REDIS" 字符串
                         2、4字节"RDB"版本号(不是Redis版本号),当前为9,填充后为0009
                         3、辅助字段,以key-value的形式
                         4、存储数据库号码
                         5、字典大小
               文件结构
                         6、过期key
                         7、主要数据,以key-value的形式存储
                         8、结束标志
                         9、校验和,就是看文件是否损坏,或者是否被修改。
                                恢复速度快
               优缺点
                                快照生成时间慢,消耗 CPU
                        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会记录过程,RDB只管结果
                                            命令传播
                                            缓存追加
               AOF原理:同步命令到AOF文件的三个步骤
                                           文件写入和保存
方式
                            AOF_FSYNC_NO:不保存
                            AOF_FSYNC_EVERYSEC:每一秒钟保存一次。(默认)
               AOF保存模式
                            AOF_FSYNC_ALWAYS:每执行一个命令保存一次。(不推荐)
                         Redis可以在AOF体积变得过大时,自动地在后台(Fork子进程)对AOF进行重
               AOF重写
                                      在redis.conf文件中配置触发方式
                          重写触发方式
                                      执行bgrewriteaof命令
               AOF的载入与数据还原
                         恢复慢文件大
                         如果把混合持久化打开, aof rewrite的时候就直接把rdb的内容写到aof文件
                         开头。 RDB的头+AOF的身体=appendonly.aof
       RDB和AOF混合持久化
                     RDB存某个时刻的数据快照,采用二进制压缩存储,AOF存操作命令,采用
                     文本存储
                     RDB性能高、AOF性能较低
                     RDB在配置触发状态会丢失最后一次快照以后更改的所有数据,AOF设置为
                     每秒保存一次,则最多丢2秒的数据
       RDB与AOF对比
                     Redis以主服务器模式运行,RDB不会保存过期键值对数据,Redis以从服务
                     器模式运行,RDB会保存过期键值对,当主服务器向从服务器同步时,再清
                     空过期键值对。AOF写入文件时,对过期的key会追加一条del命令,当执行
                     AOF重写时,会忽略过期key和del命令。
                 作为内存数据库使用时,rdb+aof方式数据不容易丢
                  有原始数据源时,每次启动时都从原始数据源中初始化,则不用开启持久化
                  (数据量较小)
       应用场景
                 作为缓存服务器时,使用rdb方式,性能高
                  在数据还原时:有rdb+aof则还原aof,因为RDB会造成文件的丢失,AOF相
                对数据要完整。只有rdb,则还原rdb
```