链滴

【Redis】Redis 的常用命令

作者：HR-HR

原文链接：https://ld246.com/article/1661956968490

来源网站：链滴

Redis常见命令

启用redis：redis-cli -h [ip] -p [6379] -a [password]

名词解释

原子操作(命令)：可以理解为只有这条命令执行完毕才会进行下一个命令操作，有点阻塞的意思。

位图：这里的位图并不是图形图像中的“位图(像素图片)”，而是一种数据储存概念，即“位图数据构”。

例如假设我们想记录用户本周7天内签到记录，可以使用字符串“0101101”：第1位表示第1天，0示未签到，1表示已签到。

通用命令

keys命令

计算redis所有的键：keys *

keys pu* (找出以pu开头的所有key的集合)

keys pu? (找出以pu开头，长度为3的key的集合)

keys pu[a-g]* (找出以pu开头，后面为a-g范围内的字母，*结尾并不限制的key的集合)

key相关命令

计算key的总数：dbsize

判断一个key是否存在：exists key

删除key：del key1 key2

设置key在5秒以后过期：expire key 5 (对于字符串类型比较特别，如果调用了set方法重新赋值，那原来的过期时间会自动删除)

查看key的剩余过期时间：ttl key (如果返回-2则表示key不存在，如果返回-1则表示该key存在且永不过期)

删除key的过期时间(key不再会过期)：persist key

查看key的类型：type key (返回类型可能为：string、hash、list、set、zset、none)

Object命令

1、查看某个key在redis内部中记录的编码类型：object encoding key

int：整数

raw：一般字符串

embstr：在2.8版本里不存在、3.0版本里一般字符串小于39字节字符串、3.2版本里小于44字节的字串

zipmap：比较小的嘻哈表

hashtable：任意哈希表

ziplist：节约大小并且较小的列

linkedlist：任意列表

intset：值储存数字的小集合

hashtable：任意集合

ziplist：比较小的有序集合

skiplist：任何有序集合

2、查看某个key空闲时间(没有被读取或写入)：object idletime key

返回空闲秒数

3、查看引用所储存值的次数：object refcount key

暂时没明白含义

type命令

查看某个key对外的类型：type key

返回值有：none (key不存在)、string (字符串)、hash (哈希表)、list (列表)、set (集合)、zset (有序

)

字符串(string)类型操作

设置(新增)key

不管key是否存在，都设置(新增)key：set key value

key不存在，才设置：setnx key value | set key value nx

key xx存在，才设置：set key value xx

key存在的情况下，设置值value和过期时间(假设8秒)：set key value ex 8 xx

设置key的值并且设置过期时间的另外一种写法：setex key 8 value nx|xx

整型(整数)运算

自增1：incr key (若key不存在，自增后key=1)

自减1：decr key (若key不存在，自减后key=-1)

自增n：incrby n (若key不存在，增加后key=n)

自减n：decrby n (若key不存在，自减呼key=-n)

浮点数运算

自增浮点数(例如3.5)：incrbyfloat key 3.5

批量获取或设置

批量获取：mget key1 key2 key3

批量设置：mset key1 value1 key2 value2 key3 value3

获取和修改value相关

设置key新的newvalue，并且返回旧的value：getset key newvalue

追加value到旧的value尾部：append key value (返回追加后的长度)

注意：这里是追加到尾部，并不是增加，假设key当前值为38，append key 9 执行以后，key的值为89)

返回字符串的长度：strlen key (注意使用UTF-8编码一个汉字为3个字节、使用GB2312一个汉字为2字节)

获取某个范围的值(例如获取下标0-5范围内的字符)：getrange key 0 5

注意：返回前6个字符，因为第1个字符下标为0，下标为5其实是第6个字符。若取值范围超出则会忽

超出部分。

设置某个下标的值(例如设置下标5的值)：setrange key 5 value (将下标5的值改为value，并返回修之后key的长度)

注意：如果设置下标超出范围，则会将中间空白的范围自动填充\x00。\x00表示不可见的字符。

哈希(hash)类型操作

设置(新增)hash

设置key的value：hset key value

获得key的value：hget key

删除key的value：hedel key

当key的field不存在时，添加并设置：hsetnx key field value

注意：哈希值操作中，不存在hset key field value nx|xx这个操作

批量设置或获取

设置key的多对field和value：hmset key field1 value1 field2 value2

获得key的多个value：hmget key field1 field2

返回key的所有field和value：hgetall key

返回key的所有valus：hvals key

返回key的所有field：hkeys key

判断子项(field)是否存在

判断key的field是否存在：hexists key field``

返回key拥有的field数量：hlen key

子项(field)自增

让field自增(若不存在则创建)：hincrby k field n

让field浮点数自增：hincrbyfloat key field float

列表(list)类型操作

设置(新增)

当新增时list并不存在，则默认新增该list

获取

返回列表总长度：llen listkey

获取索引对应的值：lindex listkey index

获取某范围内索引对应的值：lrange listkey n m （若0 -1，则表示从0到最后一个）

设置某个索引对应的值：lset listkey index newValue

删除

删除count个列表中某个值：lrem listkey count value

注意：

若 count=0，则遍历列表，删除所有value

若 count>0，则从左开始遍历，删除最多count个符合的value

若 count<0，则从右开始遍历，删除最多count个符合的value

增加或弹出

从右侧(结尾)插入值：rpush listkey value1 value2

从左侧(开头)插入值：lpush listkey value1 value2

从右侧弹出值：rpop listkey

从左侧弹出值：lpop listkey

在指定的值 前|后 插入新的值：linsert listkey before|after value newValue （会遍历整个列，发现合value就插入一次）

按照索引范围修剪(保留)列表：ltrim listkey start end

lpop阻塞(等待有可用值)版本：blpop listkey timeout (timeout为阻塞超时时间，若timeout=0表永远不阻塞)

rpop阻塞(等待有可用值)版本：brpop listkey timeout (timeout为阻塞超时时间，若timeout=0表永远不阻塞)

实用口诀

lpush + lpop = stack(栈 先进后出)　永远在列表最前面插入和弹出

lpush + rpop = queue(队列 先进先出)　在最前面插入，最后面弹出

lpush + ltrim = capped collection　在最前面插入，修剪队列 -> 控制列的数量

lpush + brpop = message queue　在最前面插入，最后面阻塞弹出 -> 实现消息队列

集合(set)类型操作

注意：集合中没有get set 这些函数、集合里的内容不允许有重复。

新增或删除

向集合key中添加集合内容element：sadd key element (若element存在则添加失败)

删除集合key中的某个元素：srem key element

获取

获取集合中元素的个数：scard key

判断某元素是否在集合内容中：sismember key value

随机获取集合中的count个元素：srandmember key count

随机弹出(删除)集合中1个元素：spop key

获取集合全部元素：smembers key

两个集合间的操作

获取重合的集合内容(交集)：sinter follow1 follow2

获取差异的集合内容(差集)：sdiff follow1 follow2

获取全部的集合内容(合集)：sunion follow1 follow2

有序集合(set)类型操作

概念

有序集合的“值”的构成：score分数值(可重复) + element值(不可重复)

注意

无论string、hash、list、set，他们赋值通常都是 field + value，但是有序集合的顺序和他们相反：score + element。

新增

添加有序集合的对值：zadd key score1 element1 score2 element2 (score可重复，element不可复)

注意：该执行复杂程度是O(logN)

删除

删除有序集合的值：zrem key element1 element2 ...

获取相关

获取有序集合内容总个数：zcard key

获取元素的分数：zscore key element

按照score来自增score

增加或减少某元素的分数：zincrby key count element (element是唯一不可重复的)

按照score来获取排名

获取某element在集合中的排名(根据score的值升序排列，即从小到大)：zrank key element

某范围内(按照score或按照索引)的相关操作，默认均为按score升序

获取某个范围内集合的值：zrange key start end withscores (0第一个,-1最后一个)

注意：该执行复杂程度是O(log(n)+m)，其中n为有序集合中元素总个数，m为start到end之间元素总个数

获取指定分数范围内的所有值：zrangebyscore key minScore maxScore

获取指定分数范围内值的个数：zcount key minScore maxScore

删除指点排名内的升序元素：zremrangebyrank key start end

删除指定分数内的升序元素：zremrangebyscore key minScore maxScore

降序相关操作

降序 rev 开头，从高到低

zrevrank、zrevrange、zrevrangebyscore

两个有序集合相关操作

两个有序集合交集排名：zinterstore

两个有序集合合集排名：zunionstore

缓存更新策略

1、LRU/LFU/FIFO算法剔除：

例如maxmemory-policy最大内存策略配置

优点：维护成本低、缺点：数据一致性最差

2、超时剔除：

例如给key设置expire过期时间

优点：维护成本低、缺点：数据一致性较差

3、主动更新：

开发控制生命周期

优点：数据一致强、缺点：维护成本高

实际项目中的策略：

1、若数据一致性要求不高，采用最大内存和淘汰策略。

2、若对数据一致性要求高，采用超时剔除和主动更新结合，还要加上最大内存和淘汰策略。

问：选择了主动更新为啥还要加上超时剔除?

答：因为万一主动更新失败，让超时剔除作为保底。当然最大内存和淘汰策略是更加基础的保底。

缓存穿透问题

名词解释：缓存穿透

通常一个合理的访问流程是：客户端请求 -> Redis缓存层 -> 若没有缓存 -> 请求真实数据库 -> 反客户端同时将数据写入缓存层

假设客户端的绝大多数请求在缓存层里找不到，被迫需要去请求真实数据库，相当于每次跳过了缓存，把这种现象称为“缓存穿透”。

造成原因

1、业务代码自身有问题：从数据库拿到的数据，其实没写入缓存层

2、遭遇恶意攻击、爬虫：大量意想不到的请求

如何发现?

1、使用Redis后，检查业务性能提高了多少，是否符合预期。

2、设定相关指标：总调度数、缓存层命中数、存储层(数据库)命中数

可以通过redis后台(如果有)，查看上述统计，观察并优化缓存命中率。

Redis实用技巧

缓存空对象

假设客户端请求的数据本身不存在(或短期内不存在)，业务代码请求储存层(数据库)之后(并没有拿到何结果)，可以给Redis内写入一个空对象，并设置过期时间。

这样短期内客户端再请求这条不存在的数据时，可以从缓存层拿到(尽管是空对象)，从而暂时性减少储层的压力。

缺点：数据短期不一致

互斥锁(分布式锁)

假设瞬间大量客户端同时请求一条相同的数据，例如第1、第2、第3...第n客户端，业务代码在处理第个客户端请求时，给这条数据请求加上一个锁lock，之后的第2、第3...第n客户端的请求会“挂起并待中”。当第1个客户端请求经过真实存储层(数据库)获取并写入缓存层后，将数据请求解锁unlock再依次批量将请求结果反馈给第2...第n个客户端。

缺点：若第1个客户端的请求一直未顺利执行完毕，则会造成后面第2...第n客户端一直处于“卡顿”，存在死锁风险。

解决方法：添加锁时增加一个过期时间，比如3秒，这样无论第1个客户端的流程是否顺利执行完毕，秒后都会解锁。

问题又来了，假设第1个客户端的执行流程没走完就进行了解锁(自动到期)，这时第2个客户端也开始行该请求，也会增加一把锁，那将来解锁环节是解的哪把锁(是第1个客户还是第2个客户的锁)?

解决方法：相对更加靠谱一些，就是在上锁时增加一个随机数，解锁时对比这个随机数来区分就是是把锁。

热点key永不过期

对于热点key，需要一直存在缓存层中。如果给key没有设置过期时间，那么会造成这个key的值一直更新，数据不一致的情况。

如果给key设置过期时间，key会在过期时间结束时被删除掉，直到下一次重新请求并写入缓存中。

如何保证缓存层里永远有一份key呢?

实现方法是：当有客户端请求这个key时，业务代码去判断key的过期剩余时间。假设剩余过期时间小8秒，先把缓存层中key的值反馈给客户端，

同时偷偷的进行一次存储层(数据库)数据请求，并将结果更新到缓存层中(依然设定有过期时间)。

缺点：逻辑过期时间增加代码维护成本。

布隆过滤器

1970年伯顿·布隆提出，2020年我依然听不懂的一个原理。

Redis键值设计原则

key名设计

1、可读性和可管理性：业务名(或数据库名) + : + 表名 + : + id，例如:mykoa:user:1

可管理性其中包含将来批量删除以XX开头的缓存数据。

2、简洁性：保证语义的前提下，控制key的长度。例如user:{uid}:friends:messages:{mid}简化为u:{uid}:fr:m:{mid}

注意：建议在redis3.0中key不要超过39字节、3.2以上版本不要超过44字节

3、不要包含特殊字符：例如空格、换行、单引号以及其他转义字符

value值设计

1、value不要过于大

string类型控制在10k以内、其他类型(hash/list/set/zset)元素个数不要超过5000

2、根据需求，采用合理的数据结构

3、设置声明周期(通过给key设置过期时间)

注意：过期时间不宜集中，容易造成缓存穿透和雪崩

Redis命令技巧

1、若数据量过大，不建议使用hgetall、lrange、smembers、zrange、sinter。可以使用hscan、sscan、zscan代替。

注意：上述讲的是获取而不是设置，推荐使用hmset这样的批量设置命令

2、禁止线上使用keys、flushall、flushdb等(可通过redis的rename机制禁掉命令)。若线上真的需要可使用scan相关代替。

3、合理使用select(按数字进行redis多数据库划分)

注意：无论如何redis终究是一个单线程，所以无论怎么划分依然是会整体相互影响。

4、不建议过多使用redis事务功能。redis不支持回滚，如果事务操作出错，无法回滚(恢复到上一个常的状态)。