链滴

redis 学习总结

作者：jiangqiang

原文链接：https://ld246.com/article/1662892365745

来源网站：链滴

redis学习总结

数据类型

通用命令

del key 删除指定key的value
keys * 查看数据库中所有的key
exists key 判断key是否存在
type key获取key的类型
expire key seconds/milliseconds 为指定key设置有效期
ttl key 获取key剩余的时效 返回3种结果，一是在有效期返回有效时长，而是不在有效期返回-2，是既没有有效期又存在返回-1

persist key 设置key的时效为永久
rename key newkey 为key进行重命名
Sort key 对集合类型排序
select index 切换数据库
move key db 移动数据到指定的库
dbsize(查看库中的key总数)
flushdb(删除当前库的数据)
flushall(删除所有库的数据)

ps -ef | grep redis   查看redis进程

启动redis命令：进入到redis-server存在的目录执行./redis-server /usr/local/redis/conf/redis-6379 conf   表示后台启动，以配置文件的方式

cat redis.conf | grep -v "#" | grep -v "^$" > redis-6379.conf 表示复制redis.conf文件到当前目录，并且去除注释和换行

./redis-cli -p 6379 ： 指定连接端口号的redis服务器

kill -9 进程号   表示杀死进程

slaveof 真机的ip 端口：表示在从机下连接主机（搭建集群）

启动redis并连接主机：进入到redis-server存在的目录执行./redis-server /usr/local/redis/conf/redi-6379.conf --slaveof 主机ip 主机端口
sed "s/6379/6380/g" redis-jiqun-6379.conf > redis-jiqun-6380.conf：表示复制文件并将文件中6379替换为6380

String数据类型

set key value：添加一组键值对
get key ： 根据key获取value
strlen key ： 统计该key对应的value的字符个数
append key value ： 表示给存在的key的value追加数据，注意当key不存在会新增key
mset key1 value1 key2 value2...：批量添加多个键值对
mget key1 key2 key3...：一次获取多个key的值，建议使用批量操作，效率更高
incr key   ：增量1，注意key的value不能是字母类型，可以是数字类型

decr key ： 减量1
incrby key 值 ： 增加指定的整数值
decrby key 值 ： 减少指定的整数值
incrbyfloat key 值 ： 表示增加指定的小数值
decrbyfloat key 值 ： 表示减少指定的小数值
set EXPIRE key ： EXPIRE 将key的生存时间设置为ttl秒
set PEXPIRE key ： PEXPIRE 将key的生成时间设置为ttl毫秒
set EXPIREAT key ： EXPIREAT 将key的过期时间设置为timestamp所代表的的秒数的时间戳
set PEXPIREAT key ： PEXPIREAT 将key的过期时间设置为timestamp所代表的的毫秒数的时间戳

注意incr和decr的数据值有上限Long.MAX_VALUE 9223372036854775807 超过上限报错

应用场景

海选投票，设计主播各大平台的粉丝数量自动更新问题

hash数据类型

因为string类型存储对象数据增删改不太灵活，所以就有了hash类型，针对hash类型来讲 key都是符串类型，value就是个hash类型(小key+value)

所以在redis中 hash底层采用的是哈希表实现

哈希表的实现（数组 + 拉链法 | 数组 + 寻址法）

hset key field value ：添加数据（hset user username zhangsan）key和filed存在就覆盖
hget key field ：获取数据
hgetall key ：获取所有的field以及对应的value值
hdel key field1 field... ：删除某个field以及value
hmset key field1 value1 field2 value2... ：添加或者修改多个数据
hmget key field1 field2... ：获取多个数据
heln key ：获取hash中value的键值对个数
hexists key field ：判断hash中是否存在某个field
hkeys key ：获取hash中所有的field
hvals key ：获取hash中所有的value
hincrby key field 值 ：增加指定的值
hincrbyfloat key filed 值 ：增加指定的小数值
hdecrby key field 值 ：减少指定的值
hdecrbyfloat key filed 值 ：减少指定的小数值
hsetnx key field value ：如果key中的field存在就什么都不做,否则添加新的field以及对应的value

注意事项：key和field不可重复，但是value可以重复

应用场景

购物车设计，充值券抢购

注意：String和hash的区别?

string应该的是数据的整体性，追求的是读为主，而hash使用了field把属性隔离开，追求的是更新作更加灵活，所以修改较多使用hash，只是存储使用string

list数据类型

对于Redis中List数据类型底层的实现原理是链表(双向)

lpush key value1 value2...　：从左边推入元素（list结构中可以允许添加重复元素）
rpush key value1 value2...　：从右边推入元素
lrange key start end：从key中取元素，从start开始，end结束
lindex key index：根据索引找key对应value的元素
llen key　：获取list中所有的元素个数
lpop key　：从左边删除（弹出）元素
rpop key　：从右边删除（弹出）元素
blpop key timeout　：从key中左边弹出元素，有就弹出，没有等待指定时间，超时还是没有获取到
brpop key timeout　：从key中右边弹出元素，有就弹出，没有等待指定时间，超时还是没有获取到

应用场景

朋友圈点赞记录

set数据类型

1.新的存储需求：存储大量的数据，在查询方面提供更高的效率(list很鸡肋的是结构是链表)

2.需要的存储结构：能够保存大量的数据，高效的内部存储机制，便于查询

3.set类型：与hash存储结构完全相同，仅存储键，不存储值（nil),并且值式不允许重复的

链表增删效率比较低，set跟hash结构完全相同，不相同的地方在于键不允许重复，value也不允许复，且无序

sadd key value1 value2...：添加元素
smembers key　：获取所有元素
srem key value...：删除元素
scard key　：获取set中所有元素的个数
sismember key value：判断集合中是否包含某个元素
spop key：从key中随机取出一个value
sinter key1 key2...：求交集（取出集合重合的内容）
sunion key2 key2...　：求并集（相同的只合并一次）
sdiff key1 key2...：求差集（谁在前就保留谁的数据）
sinterstore key3 key1 key2...　：表示将后面的交集数据存到key3集合中
sunionstore key3 key1 key2...：表示将后面的并集数据存到key3集合中
sdiffstore key3 key1 key2...　：表示将后面的差集集数据存到key3集合中
smove key3 key value　：表示将key集合中的value移动到key3集合中

应用场景

热点推荐，权限控制，统计网站的PV（网站访问量）UV（活跃用户）IP（独立访问的ip）等，黑白单问题

sorted_set数据类型

sorted_set底层存储还是基于set结构的，因此数据不能重复，如果重复添加相同的数据，score值将反复覆盖，保留最后一次修改的结果

zadd key score1 memeber1...　：存储数据（zadd scores 94 qiangqiang）
zrange key start end：取出元素（从低到高）zrange scores 0 -1

zrevrange scores start end：取出元素（从高到低）zrevrange scores 0 -1
后面加上withscores可以看到详细信息，即显示排名又显示数据
zrem key memeber ：删除某个元素
zscore key member ：获取某个member的数据
zincrby key increment member：对某个member的score进行增量或者减量(zincrby movies 1 aa)
zrangebyscore key min max withscores ：按照条件获取数据(min和max表示score范围)后面还以跟分页
zremrangebyrank key start stop：按照条件进行删除（zremrangebyrank scores 0 -1）
zinterstore destination numkeys key1 key2...：交集（注意会进行求和操作）
zunionstore destination numkeys key1 key2...：并集
zinterstore s5 3 s1 s2 s3 aggregate max：在取出并集的时候取出最大的值存到s5

应用场景

热门排行的数据获取，基于一些会员临时体验 或者会员到期提醒，权重记录

redis持久化

redis作为缓存，所有数据真正保存到mysql，为什么redis需要持久化?

解释：redis中存储了大量的数据，如果我们的redis突然宕机，客户端发起的请求都会去冲击我们的ysql，很容易造成mysql宕机，做了redis持久化，在宕机之后，重启会读取磁盘中的数据保证不用询数据库，当然在redis读取大量的数据恢复->需要缓存预热

持久化方式--RDB（快照，持久化的内容是数据）

RDB启动的方式

1、save指令（在每次操作后执行save命令，不建议使用save指令，底层是将每个命令放到队列中依执行，会造成长时间线程阻塞）

2、bgsave（后台进程启动，会将bgsave前面的数据进行持久化，发送bgsave后会发送指令调用fok函数生成子进程，创建rdb文件进行持久化）

3、redis.conf配置文件中进行配置 save second changes （指定时间内多少个key发生变化就持化）底层还是bgsave，持久化的时候可以开启LZF压缩以及数据校验，但是校验可能存在10%时间耗

RDB常见的持久化常见:比如数据容灾备份，比如主从复制(全量复制)

RDB 优缺点

1.RDB优点

RDB是一个紧凑压缩的二进制文件，存储效率较高

RDB内部存储的是redis在某个时间点的数据快照，非常适合用于数据备份，全量复制等场景

RDB恢复数据的速度要比AOF快很多

应用：服务器中每X小时执行bgsave备份，并将RDB文件拷贝到远程机器中，用于灾难恢复

2.RDB缺点

RDB方式无论是执行指令还是利用配置，无法做到实时持久化，具体较大的可能性丢失数据

bgsave指令每次运行要执行fork操作创建子进程，要牺牲掉一些性能

Redis的众多版本中未进行RDB文件格式的版本统一，有可能出现个版本服务之间数据格式无法兼容象(比如3.x和4.x的版本)

也有可能在在开启子就进行持久化，然后设置了新数据，设置完了之后服务器宕机了

持久化方式--AOF（持久化的内容是指令）

appendonly yes|no 是否开启AOF持久化

appendfsync always|everysec | no 持久化的策略

持久化原理，是当满足策略之后，会使用父进程将数据刷到缓冲区，然后父进行会调用fork函数开启进程将缓冲区的数据持久化到磁盘的aof文件

AOF重写概念： 为了节省AOF文件空间，不会持久化一些没必要的操作命令，例如对key的重复操作简单说就是将同样一个数据的若干个命令执行结果转换为最终结果数据对应的指令进行记录，AOF重是子进程在缓冲区完成的

AOF为什么要进行重写?

解释：因为当我们进行重复的操作就没有必要进行持久化，多余的操作不需要持久化

AOF重写方式:(重写主要是重写缓冲区的数据)

1.手动重写 Bgrewriteaof (类似rdb执行一条指令就重写)

2.自动重写 auto-aof-rewrite-min-size size， auto-aof-rewrite-percentage percentage

AOF触发重写机制

Redis 会记录上次重写时的AOF大小，默认配置是当AOF文件大小是上次rewrite后大小的一倍且文大于64M时触发

重写虽然可以节约大量磁盘空间，减少恢复时间。但是每次重写还是有一定的负担的，因此设定 Redi要满足一定条件才会进行重写。

auto-aof-rewrite-percentage :设置重写的基准值，文件达到100%时开始重写（文件是原来重写文件的2倍时触发)

auto-aof-rewrite-min-size:设置重写的基准值，最小文件64MB。达到这个值开始重写

例如：文件达到70MB开始重写，降到50MB，下次什么时候开始重写?100MB

系统载入时或者上次重写完毕时，Redis 会记录此时AOF大小，设为base_size,

如果Redis 的AOF当前大小>= base_size +base_size*100%(默认)且当前大小>=64mb(默认)的况下，Redis 会对AOF进行重写。

AOF重写的作用：降低磁盘占用量，提高磁盘利用率（多余的命令没有了）

提高持久化效率，提高IO性能（降低子进程的工作量）

降低数据恢复时间，提高数据恢复效率（数据量变小了）

AOF重写规则:

1.进程内已超时的数据不再写入文件

2.忽略无效指令，重写时使用进程内数据直接生成，这样新的AOF文件只保留最终数据的写入命令，del key1,hdel key2,srem key3,set key 222等

3.对统一数据的多条命令合并为一条命令

如 lpush list1 a ,lpush list1 b,lpush list1 c可以转化为lpush list1 a b c

4.为防止数据量过大造成客户端缓冲区溢出，对list,set,hash,set等类型，每条指令最多写入64个元素

AOF重写的弊端

aof重写的弊端就是如果有海量数据需要进行持久化子进程在进行持久化的时候在进行I0操作，非常耗cpu性能（redis为了解决这个问题采用管道）

Redis之管道优化AOF重写：为了提高aofrewrite效率，redis通过在父子进程间建立管道，把aofrerite期间的写命令通过管道同步给子进程，Redis管道技术可以在服务器未响应时，客户端可以继续服务器发送请求，并最终一次性获取所有服务器的响应，大大提高了AOF重写的效率。如果不使用管，当我们的缓冲区调用子进程写数据的同时线程处于阻塞状态，就不能在写数据的同时发送请求

Redis4.0开始支持混合持久化，Redis5.0开始默认就开启了混合持久化

混合持久化的开启 aof-use-rdb-preamble yes

注意混合持久化开启之后，文件是AOF的，但是这个文件开头是RDB后面则是AOF的内容，只要在AF的文件中看到REDIS表示就是混合持久化

两种持久化选择

占用存储空间：RDB占用相对较小，AOF相对较大

存储速度：RDB存数据很慢，AOF存指令很快

恢复速度：RDB直接恢复数据所以恢复速度快，AOF慢

数据安全性：RDB会丢失数据，AOF依据策略决定

资源消耗：RDB持久化数据所以高，AOF低

启动优先级：AOF高于RDB

redis事务

redis做事务太拉胯了，实际开发中不使用redis实现事务控制

事务的三个命令：multi类似mysql中开启事务，开启后将各种命令放入队列中，输入exec（类似mysql中提交事务）开始执行队列中的命令，如果在此期间某个命令输入错误，可以输入discard（类似msql中的回滚事务）放弃组队，也就是队列中的命令不执行

redis事务执行流程：服务器在接收命令之前会判断是否为事务状态，①如果不是事务状态，那么当

测到有MULTI命令写入就开启事务，并且创建队列，将之后的命令存入到队列中，当检测到有EXEC令时，就开始执行队列中的命令，如果存入到队列中的命令有误则不会执行队列中的命令，如果检测有DISCARD命令将销毁队列。②如果是事务状态就直接监听EXEC和DISCARD命令，就不用创建队列

使用锁解决redis事务问题

watch key1 key2... 对key添加监视锁，如果在执行exec前key发生变化那么终止事务执行

unwatch key　取消对key的监视

使用分布式锁解决redis事务问题

setnx lock-key value　使用setnx设置一个公共锁（setnx判断某个key是否存在）如果返回1表示取锁成功，0表示获取锁失败，有别的客户端在使用锁

del lock-key　操作完毕通过del释放锁

为了避免死锁（持有锁的客户端宕机了），我们需要预计业务时长设置时效，也就是给key设置时效如果网络延迟导致业务还没执行完，需要对锁（key）进行续期，例如每隔10秒去检测业务是否执行，没完就重新执行setnx lock-key

相当于在一个业务中加了同一把锁，就是java中重入锁的概念

redis内存问题

删除策略解决redis内存上限问题

数据删除策略（只针对过期数据）

1、定时删除：用处理器性能换区存储空间，也就是时间换空间，redis启动服务会读取conf文件中的erver.hz的值，默认是10，表示每秒执行10次

优点：没有内存的压力

缺点：如果此时cpu处于繁忙，然后取删除海量数据会拉低redis性能

2、惰性删除：原理是调用函数（expirelfNeeded（））实现，数据达到过期时间不做处理，等到次访问该数据，如果发现已经过期就删除数据，存储空间换取处理器性能，空间换时间

优点：节约cpu性能

缺点：内存压力很大

3、定期删除（重点）

redis会将0-16号数据库分为8个分区，current_db记录当前在那个分区，redis启动时会读取conf件中的server.hz的值，默认是10，表示每秒进行10次activeExpireCycle（）函数，也就是10次检，值越大检测时间就越短，这个函数会对每一个分区进行逐一检测，在检测第一个分区时，会随机挑W个key进行检测是否过期，过期就删除，如果删除的key的数量大于W的1/4，循环此过程，直到某轮删除的key数量小于等于W的1/4，根据current_db的值进入下一个分区继续检测

数据逐出策略（数据没有过期，就是内存满了）

策略1：检测某一个库

volatile-lru：挑选最近最少使用的数据淘汰

volatile-lfu：挑选最近使用次数最少的数据淘汰

volatile-ttl：挑选将要过期的数据淘汰

volatile-random：任意选择数据淘汰

策略2：检测全部库

allkeys-lru：挑选最近最少使用的数据淘汰

allkeys-lfu：挑选最近使用次数最少的数据淘汰

allkeys-random：任意选择数据淘汰

建议使用volatile-lru策略

redis集群（多台机器，从机连接主机）

第一种配置方式：主从复制

①在从机中使用（slaveof 主机ip 端口）连接主机

②在启动从机时后面跟上参数（--slaveof 主机ip 端口）连接主机

③在从机启动的配置文件中添加一条配置（slaveof 主机ip 端口）连接主机，可以使用slaveof no one命令断开与主机的连接，也可以在主机配置文件中配置（requirepass 密码）连接的密码，从机连就需要在配置文件中配置（masterauth 密码）

主从复制数据同步的原理：当我们的从机连接主机时会发送请求数据同步指令，连接成功主机会调用fork子进程创建缓冲区，执行bgsave进行持久化，生成RDB文件通过socket发送给slave，从机接收到DB，先清空本地数据，然后执行RDB文件，这是主从第一次数据同步，也就是全量复制，数据同步束之后从机会发送命令已经恢复完成，此时如果主机继续向缓冲区刷新数据，这时从机在恢复数据时用AOF重写，所以这个时候数据同步方式是部分复制

第二种配置方式：哨兵模式

哨兵工作原理：哨兵的工作就是主从切换

主从切换经历的阶段

1、监控：监控主、从、和其他哨兵，哨兵与哨兵之间通过ping命令进行互相监控，哨兵与master和save是通过info信息进行监控

2、通知：哨兵与redis节点之间通过发布与订阅进行通知

3、故障转移：哨兵发现主机宕机了，会通知其他哨兵，其他哨兵会确认主机是否真的下线了，如果认主机确实下线了，然后此时各个哨兵之间会投票选一个哨兵作为老大，通过选出的老大去观察各个save哪个性能比较好（在线并且响应快，与原master断开时间久的，断开时间越久说明最先发现master宕机），投票选择出性能好的slave作为新的master， 然后向新的master发送slaveof no one，告诉其他slave新的master的ip和端口， 其他slave会去连接新的master主机

第三种配置方式：集群模式

QPS：表示每秒请求数

当我们的并发量变得更高，可以使用集群模式搭建，不使用主从复制搭建一个主，而是采用集群方式个主进行搭建

集群就是使用网络将若干个计算机进行连接，并提供统一的管理方式，使其对外呈现单机的服务效果就好像Nginx+tomcat集群

集群的作用

1、分散单台服务器的访问压力，实现负载均衡

2、分散单台服务器的存储压力，实现可扩展性

3、降低单台服务器宕机带来的业务灾难

集群的数据结构设计

1、通过算法计算出key应该保存的位置，调用redis的函数（ CRC16(key)%16384 ）计算key的存位置

2、将所有的存储空间计划切割为16384份，每台主机保存一部分

3、每份代表的是一个存储空间也就是slot槽，如果有新的节点（主机）添加进来，这时其他节点会出自己的一部分给新的节点，使其让新的节点具有与其他节点相同大小的空间，同理，有一台节点下了，会将空间均匀分散到其他节点，那么此时有个问题是槽换了位置不知道要存在哪个空间，这时候群内部通讯设计是每个存储空间中都会独立存储一块记录各节点的存储范围，好比目录一样，这样去找key到底存在哪个槽就只需要找两次，先随意找一块槽，然后根据计算的存储位置到目录中找。

4、将key按照计算出的结果放到对应的存储空间

注意集群方式搭建redis集群，如果主机宕机了，小弟并不会上位，而是继续当大哥，等master重新线之后master还是大哥

redis发布与订阅

redis发布与订阅是通过通道实现的，例如一个客户端发布消息到通道，另外几个客户端去订阅，必名称一样，否则不能订阅

发布：pulish 发布的通道 发布的消息

订阅：subscribe 发布的通道 （客户端会一直处于阻塞状态，有消息发布就会结束阻塞）

redis其他三种高级数据类型

GEO（和地图有关，不建议使用，直接调用第三方接口就ok）

hyperLoLog

bitmaps（位图）：将一个数据转换为二进制存储起来，位图可以理解为一个普通的字符串

set str a -》97 -》 0100 1000

存储字符串是将字符串二进制数组的形式存储在redis中，位图可以直接对二进制的数组操作，位图优势在于可以用O和1来存储布尔值，这大大降低了我们的存储空间消耗。由于这个特性，我们用位来记录签到信息，记录活跃用户等，可以达到节省空间的能力

基本存取： setbit | getbit