Redis（一）基础入门

作者：wlgzs-sjl
原文链接：https://ld246.com/article/1578883757437
来源网站：链滴

<h2 id="一-Redis基础">一、Redis 基础</h2>
<h3 id="1-Redis介绍">1、Redis 介绍</h3>
redis 是一种基于键值对（key-value）数据库，其中 value 可以为 string、hash、list、 set、zset 等多种数据结构，可以满足很多应用场景。还提供了过期，发布订阅，事务，流水线，等附加功能。
流水线: Redis 的流水线功能允许客户端一次将多个命令请求发送给服务器，并将被执行的多个命令请求的结果在一个命令回复中全部返回给客户端，使用这个功能可以有效地减少客户端在执行多个命令时需要与服务器进行通信的次数。
<h3 id="2-Redis特性">2、Redis 特性</h3>

速度快，数据放在内存中，官方给出的读写性能 10 万/S，{实测：读 7-9w/s,写 5-8w/s}； (注与机器性能也有关)

基于内存，对比关系型数据库基于硬盘来说数据读写快很多；

ANSI C 语言编写，与 OS 接近，能快速调用内核函数；

单线程，避免了多线程竞争带来的资源消耗和避免了上下文的切换频繁；

键值对的数据结构服务器；

丰富的数据类型提供了丰富的功能；

简单稳定：单线程；

持久化：发生断电或机器故障，数据可能会丢失，持久化到硬盘；

主从复制：实现多个相同数据的 redis 副本；

高可用机制：哨兵机制实现高可用，保证 redis 节点故障发现和自动转移；在分布式环境下 Redi 3.0 之后提供了 Redis-Cluster 集群，能够灵活的进行水平扩容与缩容；

客户端语言多，支持多语言：java、php、python、c/ c++、nodejs 等；

<h3 id="3-使用场景">3、使用场景</h3>
1、缓存：合理使用缓存加快数据访问速度，降低后端数据源压力；
2、排行榜：按照热度排名，按照发布时间排行，主要用到列表和有序集合；
3、计数器应用：视频网站播放数，网站浏览数，使用 redis 计数；

4、社交网络：赞、踩、粉丝、下拉刷新；
5、消息队列：发布和订阅；
6、分布式锁：分布式架构保证线程安全问题，防止超卖、重复卖问题产生；
<h2 id="二-重要的指令使用">二、重要的指令使用</h2>
1. 查看所有键：keys*； -- 返回所有键（keys 还能用来搜索，比如 keys h：搜索所有以 h 开头的键）；
2. 键总数：dbsize； -- 返回键数量，如果存在大量键，线上禁止使用此令；
3. 检查键是否存在：exists key； -- 存在返回 1，不存在返回 0；
4. 删除键：del key；-- 返回删除键个数，删除不存在键返回 0；
5. 查看键存活时间：ttl key；-- 返回键剩余过期时间，不存在返回 -2；
6. 键的数据结构类型：type key；-- 返回该 key 的数据类型,键不存在返回 none；
7. 切换库：select 15;-- 切换到第 16 个库；
8. 设置过期时间：expire key seconds； -- 成功返回 1，失败返回 0；
9. 去掉过期时间：persist key； -- 成功返回 1；
<h2 id="三-基本数据类型">三、基本数据类型</h2>
<h3 id="1-字符串-String-">1、字符串（String）</h3>
1.1、字符串类型：
实际上可以是字符串（包括 XML JSON）、还有数字（整形浮点数）、二进制（图片音频视频，单个值最大不能超过 512MB；
1.2、常用命令：
set key value -- 向 Redis 写入 String 类型值； （注：redis2.6 之后允 set 命令携带参数，例如：ex nx，代替 setex setnx 作用）；
setnx key value -- 不存在该键时，返回 1 设置成功；存在的该键时，回 0 设置失败；
setex key value seconds -- 原子操作，允许设置键时同时设置过期时；
mset key1 value1 key2 value2 -- 批量设值，允许一次请求设置多个，批量获取 mget； （注：使用批量操作=一次网络请求 + 多次 redis 内部查询）；
country key1 key2 -- 获取多个值，不存在的返回 nil；
1.3、计数：
incr key -- 该 key 的 value 必须为整数，value++，非整数返回错误，存在该键从 0 自增返回 1；
incrby key 2 -- 该 key 的 value 为整数时，value+2；
decr key -- 该 key 的 value 必须为整数，value--；
decrby key 2 -- 该 key 的 value 为整数时，value-2；
incrbyfloat key 0.1 -- 该 key 的 value 为浮点型时，value+0.1；
1.4、append 追加指令：
set name zero; append name 123 -- 追加后成为 zero123；
1.5、字符串长度：
strlen key -- 返回该 key 的字节数，每个中文占 3 个字节；
1.6、截取字符串：
getrange key2 4 -- 截取该 key 的 value 的 2-4 位置并返回（包含第位）；
1.7、内部编码：
object encoding key -- 查看该 key 的内部编码；
int：当该 key 的 value 值为整型时 object encoding key 返回 int；
embstr：当该 key 的 value 值小于等于 39 字节串 object encoding ke 返回 embstr；
raw：当该 key 的 value 值大于 39 字节的字符串 object encoding key

返回 raw；
 1.8、应用场景：
1、做对象缓存，对于对象进行序列化操作，然后通过 set 存入 Redis；
2、计数器，当需要统计一个网站访问量时可以使用 incr key；
3、分布式锁，2.6 之后通过 set key value ex 10 nx 命令，2.6 之前通过 setnx key value + expire key 10 + lua 语言保证原子性来实现；
4、存普通字符串，当程序中需要一些字符串一直存在时放入 DB 和定义 static 都不合适，可使用 redis 来保存；
<h3 id="2-哈希-Hash-">2、哈希（Hash）</h3>
 2.1、哈希类型：
Hash 类型是一个 string 类型的 field 和 value 的映射表，hash 特适合用于存储对象。
 2.2、常用命令：
hset key field value -- 设值，成功返回 1，失败返回 0；
hget keyfield -- 返回该 key 对应的 field 与 value；
hdel keyfield -- 删除该 key 中的 field；
hlen keyfield -- 计算这个 key 存在多少个 field；
hmset key field1 value1 field2 value2 -- 批量设值，成功返回 OK；
hmget key field1 field2 -- 返回查询的 field 对应的 value；
hexists key field -- 判断该 key 中是否存在该 field；
field:hkeys key -- 返回该 key 的所有 field；
hvals key -- 返回该 key 的所有 value 值；
hgetall key； -- 获取该 key 的所有 field 和 value 值；
hincrby key field 1 -- 该 key 的该 filed 的 value 加 1（field 的 value 须为整型）；
hincrbyfloat key field 0.1 -- 该 key 的该 filed 加 0.1（ield 的 value 须为浮点型）；
 2.3、内部编码：
ziplist< 压缩列表 >：当 field 个数少且没有大的 value 时，内部编码为 ziplist；
hashtable< 哈希表 >：当 value 大于 64 字节，内部编码有 ziplist 变 hashtable；
 2.4、应用场景：
做缓存：比如将关系型数据表转成 redis 存储，之前是通过序列化存入 string 类型中，命令为 et 服务名_唯一标识 序列化后的文件，例子如下：
set user_id_1{“userId:1”,” userName:zero”}
使用 hash 后的存储方式为：
hmset user_id_1 userId 1 userName zero
（注：HASH 类型是稀疏，每个键可以有不同的 filed, 若用 redis 模拟做关系复杂查询开发因难维护成本高，不建议存储在 redis 之后进行复杂的条件查询）；
 2.5、三种方案实现用户信息存储优缺点：
1、原生：
set userId 1
set userName zero
优点：简单直观，每个键对应一个值；
缺点：键数过多，占用内存多，用户信息过于分散，不用于生产环境；
2、将对象序列化存入 redis：
set user:1 serialize(userInfo)
优点：编程简单，若使用序列化合理内存使用率高；
缺点：序列化与反序列化有一定开销，更新属性时需要把 userInfo 全取出来，进行反序列化，新后再序列化到 redis；
3、使用 hash 类型：
hmset user:1 userId 1 userName zero
优点：简单直观，使用合理可减少内存空间消耗；
缺点：要控制 ziplist 与 hashtable 两种编码转换，且 hashtable 会消耗更多内存；
总结：对于更新不多的情况下，可以使用序列化，对于 VALUE 值不大于 64 字节可

使用 hash 类型；
<h3 id="3-列表-List-">3、列表（List）</h3>
 3.1、列表类型：
用来存储多个有序的字符串，一个列表最多可存 2 的 32 次方减 1 个元素。因为有序，可以通索引下标获取元素或某个范围内元素列表，列表元素可以重复
 3.2、常用命令：
写命令：
rpush key value1 value2 -- 从右到左插入列表值；
lpush key value1 value2 -- 从左到右插入列表值；
linsertkey before value1 value2 -- 在 value1 之前插入 value2；
linsertkey after value1 value2 -- 在 value1 之后插入 value2；</p

lpop key -- 弹出该 key 列表中最左边第一个元素并删除；
rpop key -- 弹出该 key 列表中最右边第一个元素并删除；
lrem key count value -- 从左边开始，删除 count 个 value 元素；
ltrim key start end -- 只保留下标 start 到 end 的元素；
lset key index value -- 把下标为 index 的值替换成 value；
 读命令：
lrange key start end -- 从左 start 下标开始截取到 end 下标，end 为负数时代表倒数第几位（例：-1 为该列表倒数第一个）；
lindex keyindex -- 从左开始，返回第 index 下标的元素；
llen key -- 返回该列表的长度；
 3.3、应用场景：
特殊情况下的缓存：结合 hash 类型实现一对多缓存；
栈：先进后出；
队列：先进先出；
<h3 id="4-集合-Set-">4、集合（Set）</h3>
 4.1、集合类型：
与列表不一样的是不允许有重复元素，且集合是无序，一个集合最多可存 2 的 32 次方减 1 个素，除了支持增删改查，还支持集合交集、并集、差集；
 4.2、常用命令：
sadd key value1 value2 -- 向该 key 插入新的 value 值，返回插入的 va ue 的个数（若元素已存在，则重复无效，返回 0）；
smembers key -- 返回该 key 所有元素，返回结果为无序；
srem key value -- 删除该 key 中的 value 元素，成功返回 1，不存在回 0；
scard key -- 计算元素个数；
sismember keyvalue -- 判断 value 元素是否在该集合存在，存在返回，不存在 0；
srandmember keycount -- 在该集合中随机返回 count 个元素；
spop keycount -- 随机返回 count 个元素,并将返回的元素从集合中删；
集合的交集/并集（集合合并去重）/差集：
初始化集合：
sadduserId_1 zero 27 boy
sadduserId_2 fame 32 boy
sadduserId_3 alan29 boy
sinter userId_1 userId_2 userId_3 -- 求三个集合的交集，返回 boy
sunion userId_1 userId_2 userId_3 -- 三个集合合并（并集），去重oy；
sdiffuserId_1 userId_2 -- 俩个集合的差集，第一个集合和第二集合比

， 返回第一个集合跟第二个集合不同的值；
□□将交集(jj)、并集(bj)、差集(cj)的结果保存：
□□□□sinterstore user_jj userId_1 userId_2 userId_3 -- 求三个集合的交， 返回 boy 并将该值保存在 user_jj；
□□□□sunionstore user_bj userId_1 userId_2 userId_3 -- 三个集合合并（集），去重 boy 并将结果保存在 user_bj 中；
□□□□sdiffstore user_cj userId_1 userId_2 -- 俩个集合的差集，第一个集和第 二集合比较，返回第一个集合跟第二个集合不同的值并保存在 user_cj 中；
□ 4.3、应用场景：
□□标签： sadduserId_1_fav 健身 篮球 游泳（给用户添加爱好标签）；

□□□□□ sadd fav_健身 userId_1 userId_3（给爱好添加用户标签）；
□□社交： 通过交集、并集、差集计算；
□□查询有共同兴趣爱好的人： sinter userId_1_fav userId_3_fav（计算共爱好）；
□□智能推荐： 通过差集计算，你的好友还喜欢：比如健身；
□□规则：
□□□□sadd (常用于标签)；
□□□□spop/srandmember(随机，比如抽奖)；
□□□□sadd+sinter (用于社交，查询共同爱好的人，匹配)；
<h3 id="5-有序集合-Zset-">5、有序集合（Zset）</h3>
□ 5.1、有序集合类型：
□□有序集合常用于排行榜，如视频网站需要对用户上传视频做排行榜，或点赞数与集合有联系，能有重复的成员。
□ 5.2、常用命令：
□□zadd key score member -- 增加，不存在时返回增加成功个数，存在时回 0，score 用来排序（支持整形和双精度浮点型）；
□□zrange key start end withscores -- 查看下标 start 到 end 的 score 和 member 值；
□□zcard key -- 计算该 key 的所有元素数量；
□□zscore key member -- 查看该成员的分数；
□□zrank key member -- 升序，查看该成员在集合中的排序位置；
□□zrevrank key member -- 降序，查看该成员在集合中的排序位置；
□□zrem key member1 member2 -- 删除成员；
□□zincrby key score member -- 该成员的分数增加 score；
□□zadd key xx incr score member -- 该成员的分数增加 score；
□□zrangebyscore key score1 score2 withscores -- 返回分数 score1 到 score2 之间的所有成员（升序）；
□□zrevrangebyscore key score1 score2 withscores -- 返回分数 score1 到 score2 之间的所有成员（降序）；
□□zrangebyscore key score1 +inf withscores -- 返回大于分数 score1 所有成员（升序）；
□□zrevrangebyscore key score1 - inf withscores -- 返回小于分数 score 的所有成员（降序）；
□□zcount keysocre1 score2 -- 返回分数 score1 到 score2 之间的成员量；
□□zremrangebyrank index1 index2 -- 删除升序排序中下标 index1 和 index2 的成员；
□□zremrangebyscore key score1 score2 -- 删除分数在 score1 和 score2 之间的所有成员；
□□zremrangebyscore key socre +inf -- 删除分数大于 score 的所有成员
□有序集合的交集：
□□格式：zinterstore destination numkeys key ... [WEIGHTS weight] [AGGREGATE SUM|MIN

MAX]
□□destination:交集产生新的元素存储键名称；
□□numkeys: 要做交集计算的键个数；
□□key :元素键值；
□□weights:每个被选中的键对应值乘 weight, 默认为 1；
□□交集例子：
□□□初始化集合：
□□□□zadd user_1 1 zero 2 alan 3 fame
□□□□zadd user_2 1 zero 2 alan 7 anan
□□□□zinterstore zset_jj 2 user_1 user_2 aggregate sum
□□□□□□2：代表键合并个数；
□□□□□□aggregate sum：分数相加,可加可不加，默认就是 aggregate sum；
□□□□□□结果：zset_jj：2 zero(1+1) 4 alan(2+2)
□□□□zinterstore zset_jj_max 2 user_1 user_2 aggregate max
□□□□□□aggregate max：合并取最大分数值（还有 min 最小值）；
□□□□□□结果：zset_jj_max：1 zero 2 alan
□□□□zinterstore zset_jj_weights 2 user_1 user_2 weights 4 2 aggregate sum
□□□□1、 取出俩个集合相同的交集，结果：user_1：1 zero 2 alan
□□□□□□user_2：1 zero 2 alan
□□□□2、 将结果分数 user_1 ->zero 14=4 user_1->alan 24=8
□□□□□□得出结果：user_1：4 zero 8 alan
□□□□3、 将结果分数 user_2 –>zero 12=2 user_2 ->alan 22=4
□□□□□□得出结果：user_2：2 zero 4 alan
□□□□4、 最后相乘结果，求和：zset_jj_weights：6 zero12 alan
□□□□5、总结：将 user_1 成员分数乘 4，user_2 成员分数乘 2，取交集求和（当最后为 max 或者 min 时取最大或者最小）；
□□有序集合的并集（合并去重）：
□□□格式： zunionstore destination numkeys key ... [WEIGHTS weight] [AGGREGATE SUM|MN|MAX]；
□□□destination:交集产生新的元素存储键名称；
□□□numkeys: 要做交集计算的键个数；
□□□key :元素键值；
□□□weights:每个被选中的键对应值乘 weight, 默认为 1；
zunionstore zset_user_bj 2 user_1 user_2 weights 8 4 aggregate max
-- 与以上 zinterstore 一样，只是取并集，指令一样；
□ 5.3、应用场景：
□□排行榜：列如百度搜索排行榜、微博热搜、抖音热搜等等。
□□点赞数：通过自增指令，点赞加 1；