

# Redis

Redis 是 **Remote Dictionary Service** 的简称；也是远程字典服务；

Redis 是内存数据库，**KV** 数据库，数据结构数据库；

Redis 应用非常广泛，如 Twitter、暴雪娱乐、Github、Stack Overflow、腾讯、阿里巴巴、京东、华为、新浪微博等，很多中小型公司也在使用；

Redis 命令查看：<http://redis.cn/commands.html>

## 应用

- 记录朋友圈点赞数、评论数和点击数 (hash)
- 记录朋友圈说说列表 (排序)，便于快速显示朋友圈 (list)
- 记录文章的标题、摘要、作者和封面，用于列表页展示 (hash)
- 记录朋友圈的点赞用户ID列表 (list)，评论ID列表 (list)，用于显示和去重计数 (zset)
- 缓存热点数据，减少数据库压力 (hash)
- 如果朋友圈说说 ID 是整数 id，可使用 redis 来分配朋友圈说说 id (计数器) (string)
- 通过集合 (set) 的交并差集运算来实现记录好友关系 (set)
- 游戏业务中，每局战绩存储 (list)

## 安装编译

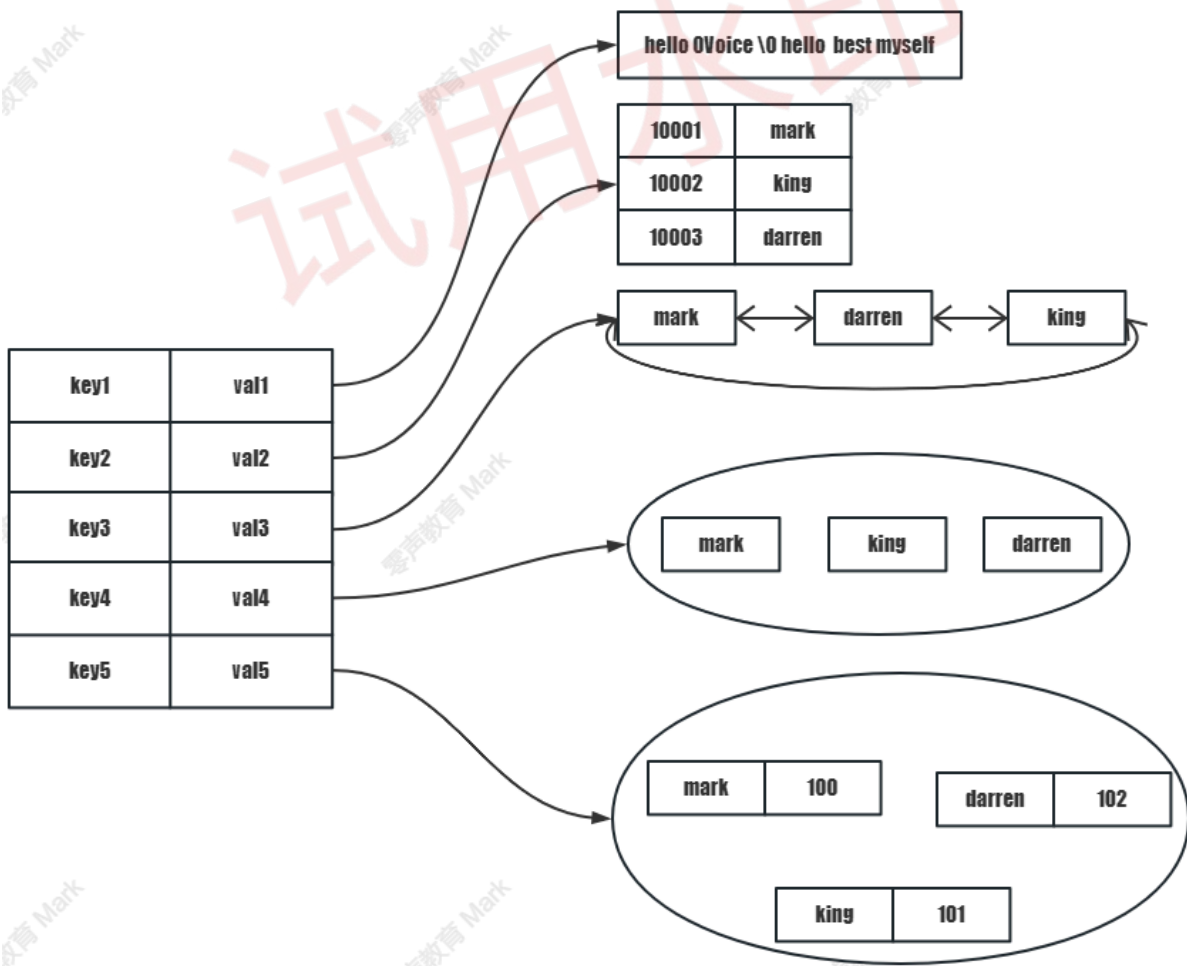
```
1 git clone https://gitee.com/mirrors/redis.git -b 6.2
2 cd redis
3 make
4 make test
5 make install
6 # 默认安装在 /usr/local/bin
7 # redis-server 是服务端程序
8 # redis-cli 是客户端程序
```

## 启动

```
1 mkdir redis-data
2 # 把redis文件夹下 redis.conf 拷贝到 redis-data
3 # 修改 redis.conf
4 # requirepass 修改密码 123456
5 # daemonize yes
6 cd redis-data
7 redis-server redis.conf
8 # 通过 redis-cli 访问 redis-server
9 redis-cli -h 127.0.0.1 -a 123456
```

# 认识 Redis

type	K	V
string	K1 ⇒ K2 ⇒	V1 V2
list	K3 ⇒	ele1 ↔ ele2 ↔ ele3
hash	K4 ⇒	(f1,v1) (f2,v2) (f3,v3)
set	k5 ⇒	[m1,m2,m3,m4]
zset	k6 ⇒	[(c1,m1),(c2,m2),(c3,m3)]



`string` 是一个安全的二进制字符串;

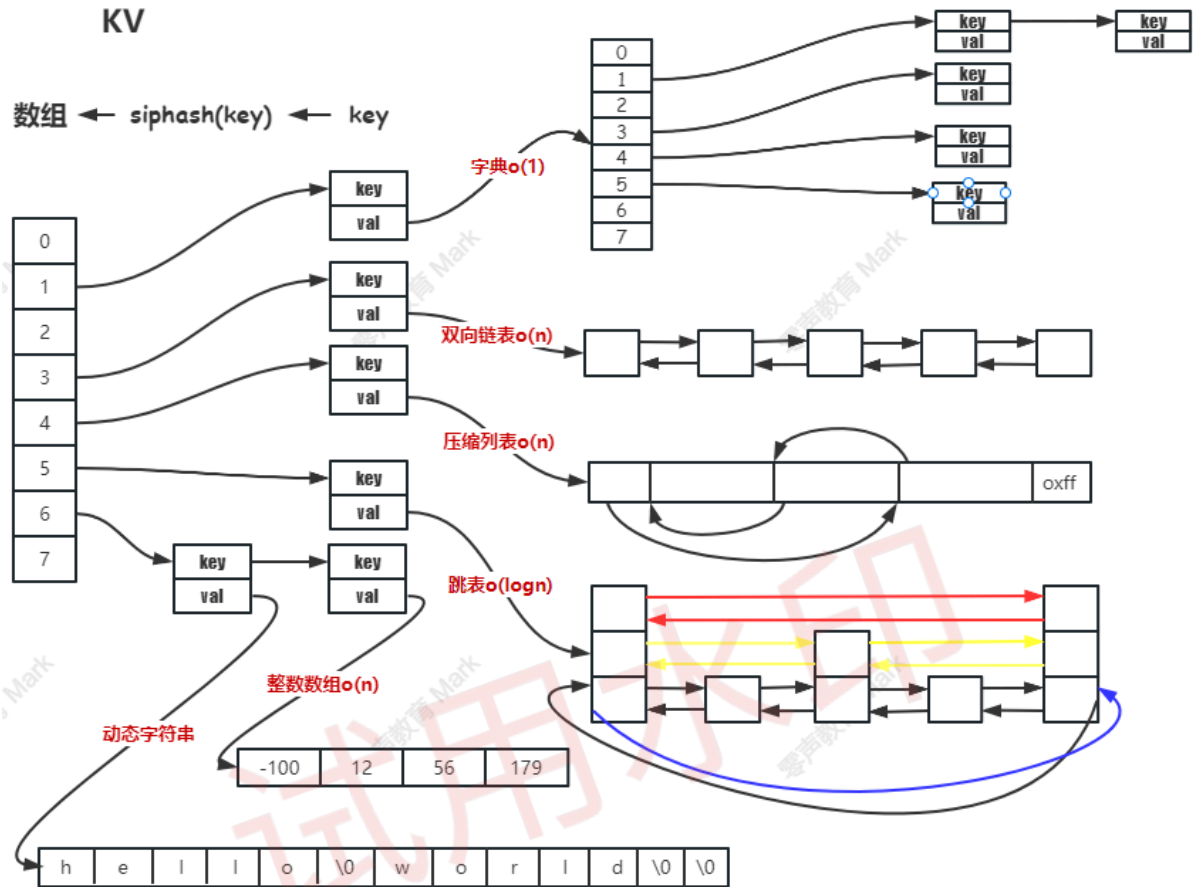
双端队列 (链表) `list`: 有序 (插入有序);

散列表 **hash**: 对顺序不关注, *field* 是唯一的;

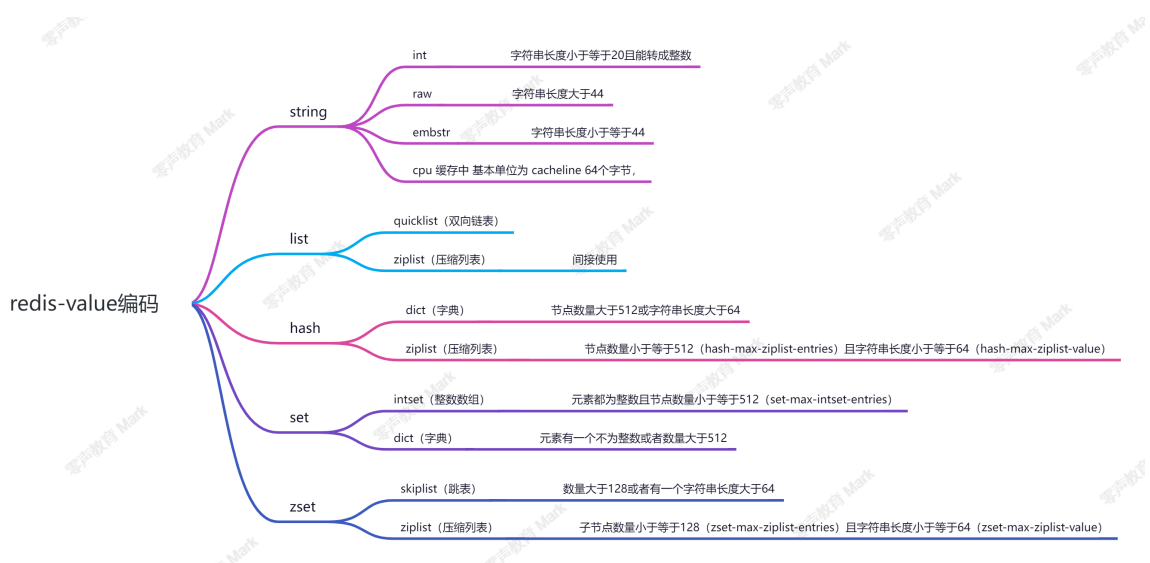
无序集合 **set**: 对顺序不关注, 里面的值都是唯一的;

有序集合 **zset**: 对顺序是关注的, 里面的值是唯一的; 根据 *member* 来确定唯一; 根据 *score* 来确定有序;

## redis 存储结构 (KV)



## redis 中 value 编码



redis 内存数据库;

1. 到底是要运行速度快, 还是要存储效率高;
2. 数据量少的时候, 存储效率高为主;

3. 数据量多的时候，运行速度快；

## string

字符数组，该字符串是动态字符串 *raw*，字符串长度小于1M 时，加倍扩容；超过 1M 每次只多扩 1M；字符串最大长度为 512M；

注意：*redis* 字符串是二进制安全字符串；可以存储图片，二进制协议等二进制数据；

## 基础命令

```
1  # 设置 key 的 value 值
2  SET key val
3  # 获取 key 的 value
4  GET key
5  # 执行原子加一的操作
6  INCR key
7  # 执行原子加一个整数的操作
8  INCRBY key increment
9
10 # 执行原子减一的操作
11 DECR key
12 # 执行原子减一个整数的操作
13 DECRBY key decrement
14
15 # 如果key不存在，这种情况下等同SET命令。 当key存在时，什么也不做
16 # set Not exist ok 这个命令是否执行了 0,1 是不是操作结果是不是成功
17 SETNX key value
18 # 删除 key val 键值对
19 DEL key
20
21 # 设置或者清空key的value(字符串)在offset处的bit值。 setbit embstr raw int
22 # 动态字符串 能够节约内存
23 SETBIT key offset value
24 # 返回key对应的string在offset处的bit值
25 GETBIT key offset
26 # 统计字符串被设置为1的bit数。
27 BITCOUNT key
```

## 存储结构

字符串长度小于等于 20 且能转成整数，则使用 `int` 存储；

字符串长度小于等于 44，则使用 `embstr` 存储；

字符串长度大于 44，则使用 `raw` 存储；

## 应用

### 对象存储

```
1 SET role:10001 '{"name":"mark","sex":"male","age":30}'
2 SET role:10002 '{"name":"darren","sex":"male","age":30}'
3 # 极少修改，对象属性字段很少改变的时候
4 GET role:10001
```

```
5 # key 如何来设置
6 # 1. 有意义的字段 role 有多行
7 # 2. role:10001 redis 客户端 role:10001:recharge role:10001:activity:10001
```

## 累加器

```
1 # 统计阅读数 累计加1
2 incr reads
3 # 累计加100
4 incrby reads 100
```

## 分布式锁

```
1 # 加锁 加锁 和 解析 redis 实现是 非公平锁 etcd zk 用来实现公平锁
2 # 阻塞等待 阻塞连接的方式
3 # 介绍简单的原理: 事务
4 setnx lock 1 # 不存在才能设置 定义加锁行为 占用锁
5 setnx lock uuid # expire 30 过期
6 set lock uuid nx ex 30
7
8 # 释放锁
9 del lock
10 if (get(lock) == uuid)
11     del(lock);
```

## 位运算

```
1 # 猜测一下 string 是用的 int 类型 还是 string 类型
2 # 月签到功能 10001 用户id 202106 2021年6月份的签到 6月份的第1天
3 setbit sign:10001:202106 1 1
4 # 计算 2021年6月份 的签到情况
5 bitcount sign:10001:202106
6 # 获取 2021年6月份 第二天的签到情况 1 已签到 0 没有签到
7 getbit sign:10001:202106 2
```

## list

双向链表实现，列表首尾操作（删除和增加）时间复杂度  $O(1)$ ；查找中间元素时间复杂度为  $O(n)$ ；

列表中数据是否压缩的依据：

1. 元素长度小于 48，不压缩；
2. 元素压缩前后长度差不超过 8，不压缩；

## 基础命令

```
1 # 从队列的左侧入队一个或多个元素
2 LPUSH key value [value ...]
3 # 从队列的左侧弹出一个元素
4 LPOP key
5 # 从队列的右侧入队一个或多个元素
6 RPUSH key value [value ...]
```

```

7  # 从队列的右侧弹出一个元素
8  RPOP key
9  # 返回从队列的 start 和 end 之间的元素 0, 1 2 负索引
10 LRange key start end
11 # 从存于 key 的列表里移除前 count 次出现的值为 value 的元素
12 # list 没有去重功能      hash set zset
13 LREM key count value
14 # 它是 RPOP 的阻塞版本, 因为这个命令会在给定list无法弹出任何元素的时候阻塞连接
15 BRPOP key timeout # 超时时间 + 延时队列

```

## 存储结构

```

1  /* Minimum ziplist size in bytes for attempting compression. */
2  #define MIN_COMPRESS_BYTES 48
3
4  /* quicklistNode is a 32 byte struct describing a ziplist for a quicklist.
5   * we use bit fields keep the quicklistNode at 32 bytes.
6   * count: 16 bits, max 65536 (max zl bytes is 65k, so max count actually <
7   * 32k).
8   * encoding: 2 bits, RAW=1, LZF=2.
9   * container: 2 bits, NONE=1, ZIPLIST=2.
10  * recompress: 1 bit, bool, true if node is temporary decompressed for
11  * usage.
12  * attempted_compress: 1 bit, boolean, used for verifying during testing.
13  * extra: 10 bits, free for future use; pads out the remainder of 32 bits */
14  typedef struct quicklistNode {
15      struct quicklistNode *prev;
16      struct quicklistNode *next;
17      unsigned char *zl;
18      unsigned int sz; /* ziplist size in bytes */
19      unsigned int count : 16; /* count of items in ziplist */
20      unsigned int encoding : 2; /* RAW==1 or LZF==2 */
21      unsigned int container : 2; /* NONE==1 or ZIPLIST==2 */
22      unsigned int recompress : 1; /* was this node previous compressed? */
23      unsigned int attempted_compress : 1; /* node can't compress; too small
24      */
25      unsigned int extra : 10; /* more bits to steal for future usage */
26  } quicklistNode;
27
28  typedef struct quicklist {
29      quicklistNode *head;
30      quicklistNode *tail;
31      unsigned long count; /* total count of all entries in all
32      ziplists */
33      unsigned long len; /* number of quicklistNodes */
34      int fill : QL_FILL_BITS; /* fill factor for individual
35      nodes */
36      unsigned int compress : QL_COMP_BITS; /* depth of end nodes not to
37      compress;0=off */
38      unsigned int bookmark_count: QL_BM_BITS;
39      quicklistBookmark bookmarks[];
40  } quicklist;

```

## 应用

### 栈 (先进后出 FILO)

```
1 LPUSH + LPOP
2 # 或者
3 RPUSH + RPOP
```

### 队列 (先进先出 FIFO)

```
1 LPUSH + RPOP
2 # 或者
3 RPUSH + LPOP
```

### 阻塞队列 (blocking queue)

```
1 LPUSH + BRPOP
2 # 或者
3 RPUSH + BLPOP
```

### 异步消息队列

操作与队列一样，但是在不同系统间；生成者和消费者；

### 获取固定窗口记录 (战绩)

```
1 # 在某些业务场景下，需要获取固定数量的记录；比如获取最近50条战绩；这些记录需要按照插入的先
   后顺序返回；
2 lpush says '{"name":"零声教育【Mark老师】", ["text"]:"祝大家儿童节快乐!",
   ["picture":["url://image-20210601172741434.jpg", "url://image-
   20210601172741435.jpg"], timestamp = 1231231230}'
3 lpush says '{"name":"零声教育【King老师】", ["text"]:"祝大家儿童节快乐!",
   ["picture":["url://image-20210601172742434.jpg", "url://image-
   20210601172741436.jpg"], timestamp = 1231231231}'
4 lpush says '{"name":"零声教育【Darren老师】", ["text"]:"祝大家儿童节快乐!",
   ["picture":["url://image-20210601172743434.jpg", "url://image-
   20210601172741437.jpg"], timestamp = 1231231232}'
5 lpush says '{"name":"零声教育【Mark老师】", ["text"]:"一切只为渴望更优秀的你",
   ["picture":["url://image-20210601172744434.jpg", "url://image-
   20210601172741438.jpg"], timestamp = 1231231233}'
6 lpush says '{"name":"零声教育【Darren老师】", ["text"]:"hello 0voice! hello
   to better self", ["picture":["url://image-20210601172745439.jpg",
   "url://image-20210601172741435.jpg"], timestamp = 1231231234}'
7 lpush says '{"name":"零声教育【King老师】", ["text"]:"2021届学员真牛逼!",
   ["picture":["url://image-20210601172745434.jpg", "url://image-
   20210601172741440.jpg"], timestamp = 1231231235}'
8 # 裁剪最近5条记录    战绩    近50条
9 ltrim says 0 4
10 lrange says 0 -1
```

实际项目中需要保证命令的原子性，所以一般用 lua 脚本 或者使用 pipeline 命令；

```
1  -- redis lua脚本
2  local record = KEYS[1]
3  redis.call("LPUSH", "says", record)
4  redis.call("LTRIM", "says", 0, 4)
```

## hash

散列表，在很多高级语言当中包含这种数据结构；c++ `unordered_map` 通过 key 快速索引 value；

## 基础命令

```
1  # 获取 key 对应 hash 中的 field 对应的值
2  HGET key field
3  # 设置 key 对应 hash 中的 field 对应的值
4  HSET key field value
5  # 设置多个hash键值对
6  HMSET key field1 value1 field2 value2 ... fieldn valuen
7  # 获取多个field的值
8  HMGET key field1 field2 ... fieldn
9  # 给 key 对应 hash 中的 field 对应的值加一个整数值
10 HINCRBY key field increment
11 # 获取 key 对应的 hash 有多少个键值对
12 HLEN key
13 # 删除 key 对应的 hash 的键值对，该键为field
14 HDEL key field
```

## 存储结构

节点数量大于 512 (`hash-max-ziplist-entries`) 或所有字符串长度大于 64 (`hash-max-ziplist-value`)，则使用 `dict` 实现；

节点数量小于等于 512 且有一个字符串长度小于 64，则使用 `ziplist` 实现；

## 应用

### 存储对象

```
1  hmset hash:10001 name mark age 18 sex male
2  # 与 string 比较
3  set hash:10001 '{"name":"mark","sex":"male","age":18}'
4  # 假设现在修改 mark的年龄为19岁
5
6  # hash:
7      hset hash:10001 age 19
8  # string:
9      get hash:10001
10     # 将得到的字符串调用json解密，取出字段，修改 age 值
11     # 再调用json加密
12     set hash:10001 '{"name":"mark","sex":"male","age":19}'
```





```
1 # 将用户id作为 key
2 # 商品id作为 field
3 # 商品数量作为 value
4 # 注意：这些物品是按照我们添加顺序来显示的；
5
6 # 添加商品：
7     hmset MyCart:10001 40001 1 cost 5099 desc "戴尔笔记本14-3400"
8     lpush MyItem:10001 40001
9 # 增加数量：
10    hincrby MyCart:10001 40001 1
11    hincrby MyCart:10001 40001 -1 // 减少数量1
12 # 显示所有物品数量：
13    hlen MyCart:10001
```

```
14 # 删除商品:
15     hdel MyCart:10001 40001
16     lrem MyItem:10001 1 40001
17 # 获取所有物品:
18     lrange MyItem:10001
19     # 40001 40002 40003
20     hget MyCart:10001 40001
21     hget MyCart:10001 40002
22     hget MyCart:10001 40003
```

## set

集合；用来存储唯一性字段，不要求有序；

存储不需要有序，操作（交并差集的时候排序）？

## 基础命令

```
1 # 添加一个或多个指定的member元素到集合的 key中
2 SADD key member [member ...]
3 # 计算集合元素个数
4 SCARD key
5
6 # SMEMBERS key
7 SMEMBERS key
8 # 返回成员 member 是否是存储的集合 key的成员
9 SISMEMBER key member
10
11 # 随机返回key集合中的一个或者多个元素，不删除这些元素
12 SRANDMEMBER key [count]
13 # 从存储在key的集合中移除并返回一个或多个随机元素
14 SPOP key [count]
15
16 # 返回一个集合与给定集合的差集的元素
17 SDIFF key [key ...]
18 # 返回指定所有的集合的成员的交集
19 SINTER key [key ...]
20 # 返回给定的多个集合的并集中的所有成员
21 SUNION key [key ...]
```

## 存储结构

元素都为整数且节点数量小于等于 512 (*set-max-intset-entries*)，则使用整数数组存储；

元素当中有一个不是整数或者节点数量大于 512，则使用字典存储；

## 应用

### 抽奖

```
1 # 添加抽奖用户
2     sadd Award:1 10001 10002 10003 10004 10005 10006
3     sadd Award:1 10009
4 # 查看所有抽奖用户
```

```
5 smembers Award:1
6 # 抽取多名幸运用户
7 srandmember Award:1 10
8 # 如果抽取一等奖1名，二等奖2名，三等奖3名，该如何操作？
```

## 共同关注

```
1 sadd follow:A mark king darren mole vico
2 sadd follow:C mark king darren
3 sinter follow:A follow:C
```

## 推荐好友

```
1 sadd follow:A mark king darren mole vico
2 sadd follow:C mark king darren
3 # C可能认识的人:
4 sdiff follow:A follow:C
```

## zset

有序集合；用来实现排行榜；它是一个有序唯一；

## 基础命令

```
1 # 添加到键为key有序集合（sorted set）里面
2 ZADD key [NX|XX] [CH] [INCR] score member [score member ...]
3 # 从键为key有序集合中删除 member 的键值对
4 ZREM key member [member ...]
5 # 返回有序集key中，成员member的score值
6 ZSCORE key member
7 # 为有序集key的成员member的score值加上增量increment
8 ZINCRBY key increment member
9 # 返回key的有序集元素个数
10 ZCARD key
11 # 返回有序集key中成员member的排名
12 ZRANK key member
13 # 返回存储在有序集合key中的指定范围的元素 order by id limit 1,100
14 ZRANGE key start stop [WITHSCORES]
15 # 返回有序集key中，指定区间内的成员(逆序)
16 ZREVRANGE key start stop [WITHSCORES]
```

## 存储结构

节点数量大于 128 或者有一个字符串长度大于 64，则使用跳表（skiplist）；

节点数量小于等于 128（zset-max-ziplist-entries）且所有字符串长度小于等于 64（zset-max-ziplist-value），则使用 ziplist 存储；

数据少的时候，节省空间； $O(n)$

数量多的时候，访问性能； $O(1)$  or  $O(\log_2\{n\})$

## 应用

### 百度热榜

百度搜索 >

换一换

传承发展中华优秀传统文化

1 村民擅自组织民间划龙舟活动被拘留 热

2 女子称撑破试穿裙子无奈买下 热

3 超级工程背后的超级智慧

4 章莹颖父亲的心碎直播

5 毕业典礼比心时校长手指被强制掰弯

6 校方称曝光大叔女子将被开除系不实 热

7 婆婆不开空调致孕妇中暑送医

8 中国时隔7年重启对越南送电

9 美军侦察机在东海高强度活动

```
1  # 点击新闻:
2     zincrby hot:20230612 1 10001
3     zincrby hot:20230612 1 10002
4     zincrby hot:20230612 1 10003
5     zincrby hot:20230612 1 10004
6     zincrby hot:20230612 1 10005
7     zincrby hot:20230612 1 10006
8     zincrby hot:20230612 1 10007
9     zincrby hot:20230612 1 10008
10    zincrby hot:20230612 1 10009
11    zincrby hot:20230612 1 10010
12
13  # 获取排行榜:
14    zrevrange hot:20230612 0 9 withscores
```

### 延时队列

将消息序列化成一个字符串作为 `zset` 的 `member`；这个消息的到期处理时间作为 `score`，然后用多个线程轮询 `zset` 获取到期的任务进行处理。

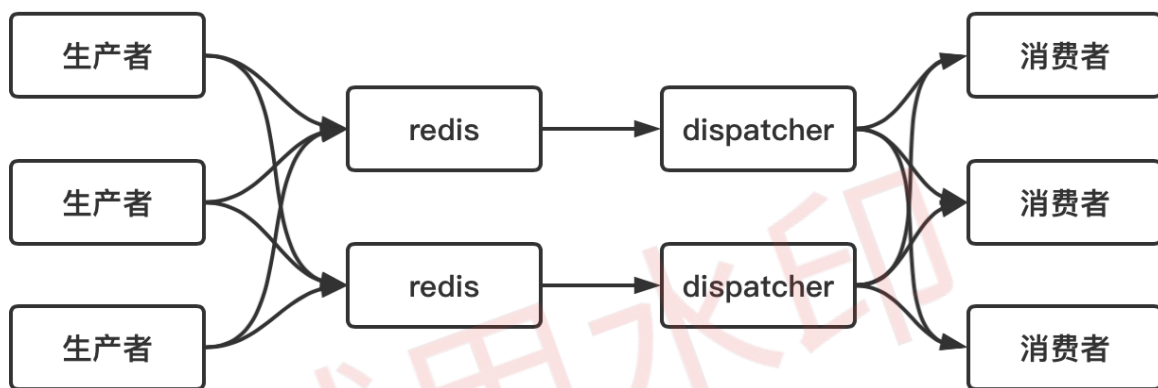
```
1  def delay(msg):
2      msg.id = str(uuid.uuid4()) #保证 member 唯一
3      value = json.dumps(msg)
4      retry_ts = time.time() + 5 # 5s后重试
5      redis.zadd("delay-queue", retry_ts, value)
6
7  # 使用连接池
8  def loop():
9      while True:
```

```

10     values = redis.zrangebyscore("delay-queue", 0, time.time(), start=0,
num=1)
11     if not values:
12         time.sleep(1)
13         continue
14     value = values[0]
15     success = redis.zrem("delay-queue", value)
16     if success:
17         msg = json.loads(value)
18         handle_msg(msg)
19
20 # 缺点: loop 是多线程竞争, 两个线程都从zrangebyscore获取到数据, 但是zrem一个成功一个失败,
21 # 优化: 为了避免多余的操作, 可以使用lua脚本原子执行这两个命令
22 # 解决: 漏斗限流

```

## 分布式定时器



生产者将定时任务 `hash` 到不同的 `redis` 实体中, 为每一个 `redis` 实体分配一个 `dispatcher` 进程, 用来定时获取 `redis` 中超时事件并发布到不同的消费者中;

## 时间窗口限流

系统限定用户的某个行为在指定的时间范围内 (动态) 只能发生 **N** 次;

```

1 # 指定用户 user_id 的某个行为 action 在特定时间内 period 只允许发生该行为最大次数
max_count
2
3 local function is_action_allowed(red, userid, action, period, max_count)
4     local key = tab_concat({"hist", userid, action}, ":")
5     local now = zv.time()
6     red:init_pipeline()
7     -- 记录行为
8     red:zadd(key, now, now)
9     -- 移除时间窗口之前的行为记录, 剩下的都是时间窗口内的记录
10    red:zremrangebyscore(key, 0, now - period * 100)
11    -- 获取时间窗口内的行为数量
12    red:zcard(key)
13    -- 设置过期时间, 避免冷用户持续占用内存 时间窗口的长度+1秒
14    red:expire(key, period + 1)
15    local res = red:commit_pipeline()
16    return res[3] <= max_count
17 end
18

```

19 # 维护一次时间窗口，将窗口外的记录全部清理掉，只保留窗口内的记录；  
20 # 缺点：记录了所有时间窗口内的数据，如果这个量很大，不适合做这样的限流；漏斗限流  
21 # 注意：如果用 **key + expire** 操作也能实现，但是实现的是熔断限流，这里是时间窗口限流的功能；

试用水印