零声教育 Mark 老师 QQ: 2548898954

Redis

> Redis 是**Re**mote **Di**ctionary **S**ervice 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，评论ID列表，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)

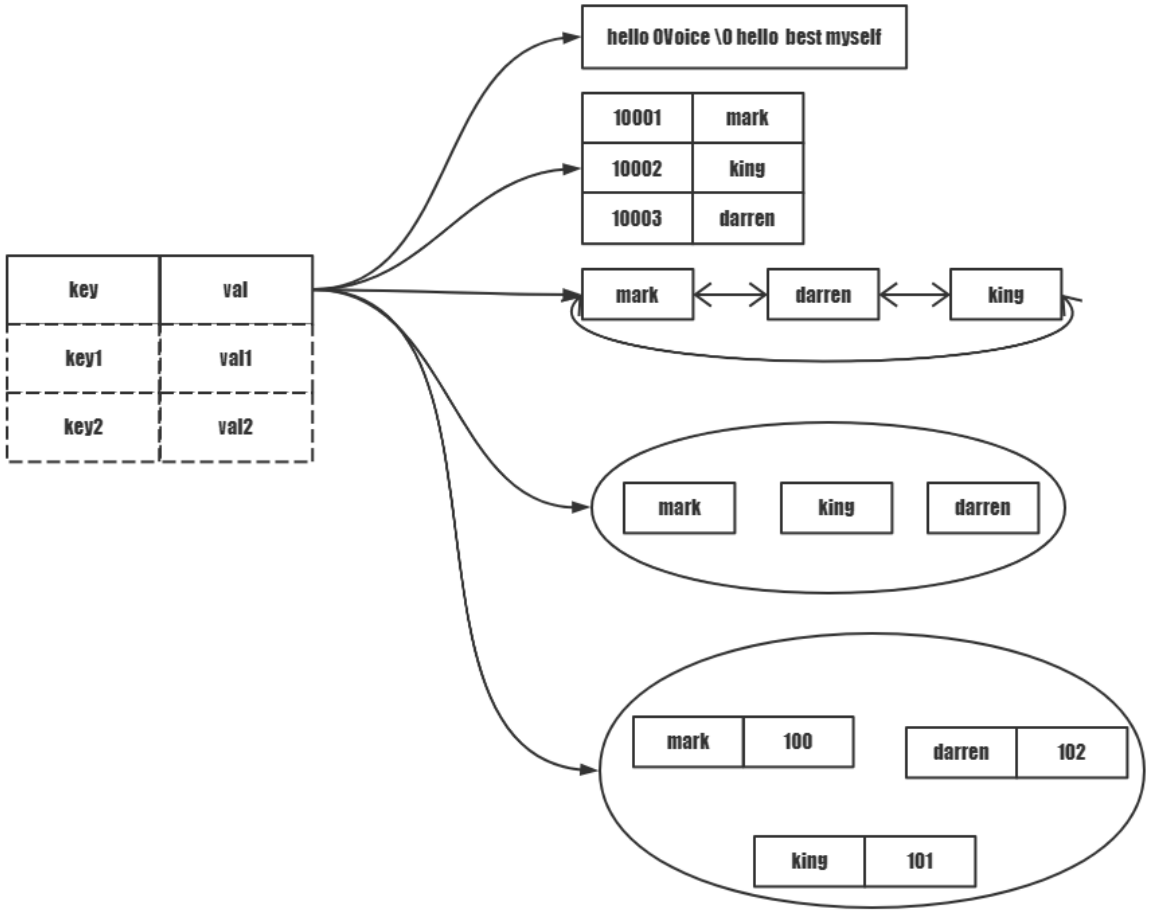
安装编译

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

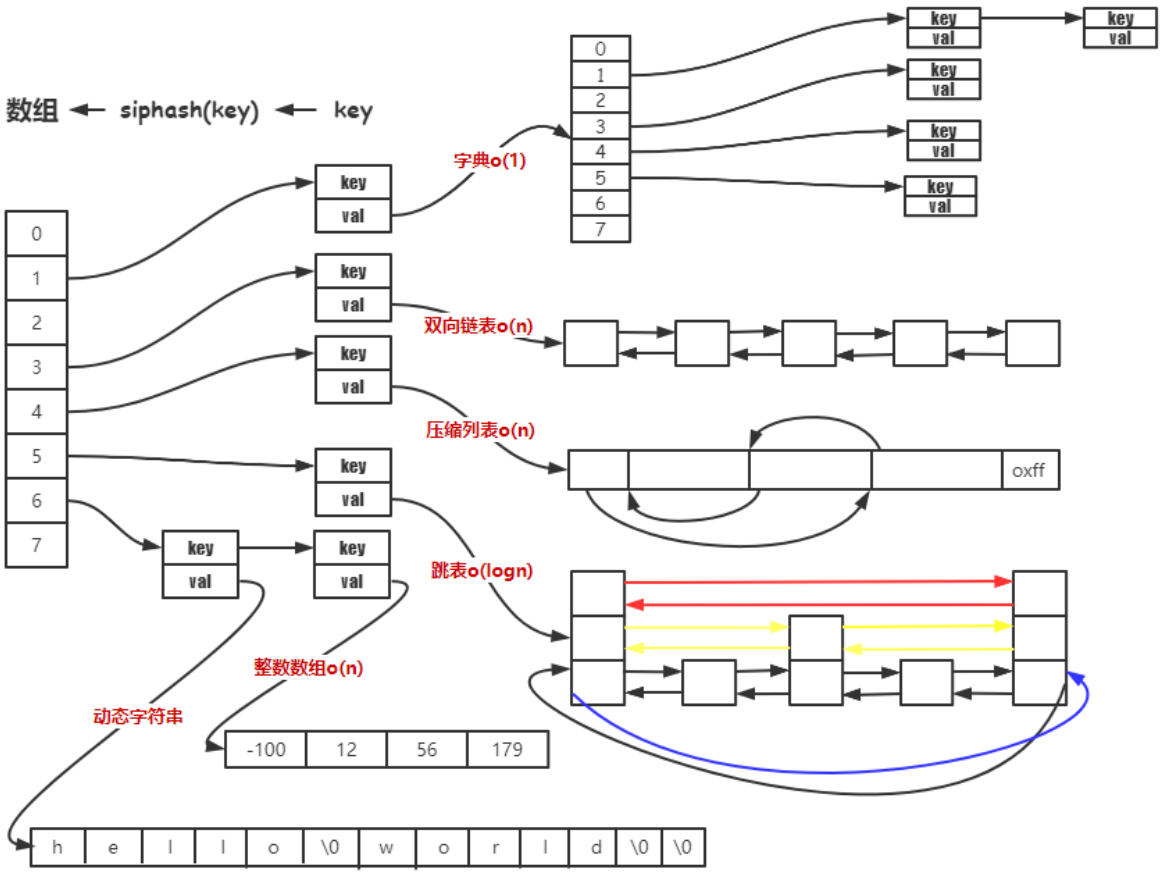
启动

```
mkdir redis-data
# 把redis文件夹下 redis.conf 拷贝到 redis-data
# 修改 redis.conf
# requirepass 修改密码 123456
# daemonize yes
cd redis-data
redis-server redis.conf
# 通过 redis-cli 访问 redis-server
redis-cli -h 127.0.0.1 -a 123456
```

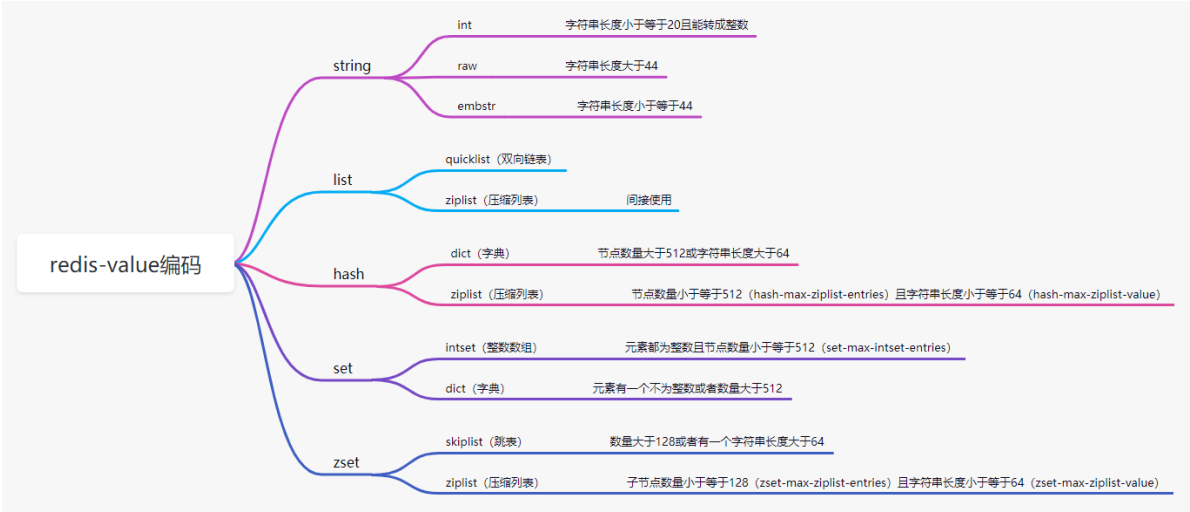
认识Redis



redis存储结构（KV）



redis中value编码



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

```
# 设置 key 的 value 值
SET key val
# 获取 key 的 value
GET key
# 执行原子加一的操作
INCR key
# 执行原子加一个整数的操作
INCRBY key increment

# 执行原子减一的操作
DECR key
# 执行原子减一个整数的操作
DECRBY key decrement

# 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
SETNX key value
# 删除 key val 键值对
DEL key

# 设置或者清空key的value(字符串)在offset处的bit值。
SETBIT key offset value
# 返回key对应的string在offset处的bit值
GETBIT key offset
# 统计字符串被设置为1的bit数.
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

```
SET role:10001 '{["name"]:"mark",["sex"]:"male",["age"]:30}'
GET role:10001
```

累加器

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

分布式锁

```
# 加锁
setnx lock 1
# 释放锁
del lock
#  1. 排他功能  2. 加锁行为定义  3. 释放行为定义
```

位运算

```
# 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
setbit sign:10001:202106 1 1
# 计算 2021年6月份 的签到情况
bitcount sign:10001:202106
# 获取 2021年6月份 第二天的签到情况 1 已签到 0 没有签到
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

```
# 从队列的左侧入队一个或多个元素
LPUSH key value [value ...]
# 从队列的左侧弹出一个元素
LPOP key
# 从队列的右侧入队一个或多个元素
RPUSH key value [value ...]
# 从队列的右侧弹出一个元素
RPOP key
# 返回从队列的 start 和 end 之间的元素  0, 1 2
LRANGE key start end
# 从存于 key 的列表里移除前 count 次出现的值为 value 的元素
```

```
LREM key count value
# 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
BRPOP key timeout  # 超时时间 + 延时队列
```

存储结构

```
/* Minimum ziplist size in bytes for attempting compression. */
#define MIN_COMPRESS_BYTES 48

/* quicklistNode is a 32 byte struct describing a ziplist for a quicklist.
 * We use bit fields keep the quicklistNode at 32 bytes.
 * count: 16 bits, max 65536 (max zl bytes is 65k, so max count actually < 32k).
 * encoding: 2 bits, RAW=1, LZF=2.
 * container: 2 bits, NONE=1, ZIPLIST=2.
 * recompress: 1 bit, bool, true if node is temporary decompressed for usage.
 * attempted_compress: 1 bit, boolean, used for verifying during testing.
 * extra: 10 bits, free for future use; pads out the remainder of 32 bits */
typedef struct quicklistNode {
    struct quicklistNode *prev;
    struct quicklistNode *next;
    unsigned char *zl;
    unsigned int sz;             /* ziplist size in bytes */
    unsigned int count : 16;     /* count of items in ziplist */
    unsigned int encoding : 2;   /* RAW==1 or LZF==2 */
    unsigned int container : 2;  /* NONE==1 or ZIPLIST==2 */
    unsigned int recompress : 1; /* was this node previous compressed? */
    unsigned int attempted_compress : 1; /* node can't compress; too small */
    unsigned int extra : 10; /* more bits to steal for future usage */
} quicklistNode;

typedef struct quicklist {
    quicklistNode *head;
    quicklistNode *tail;
    unsigned long count;        /* total count of all entries in all ziplists */
    unsigned long len;          /* number of quicklistNodes */
    int fill : QL_FILL_BITS;              /* fill factor for individual nodes */
    unsigned int compress : QL_COMP_BITS; /* depth of end nodes not to compress;0=off */
    unsigned int bookmark_count: QL_BM_BITS;
    quicklistBookmark bookmarks[];
} quicklist;
```

应用

栈（先进后出 FILO）

```
LPUSH + LPOP
# 或者
RPUSH + RPOP
```

队列（先进先出 FIFO）

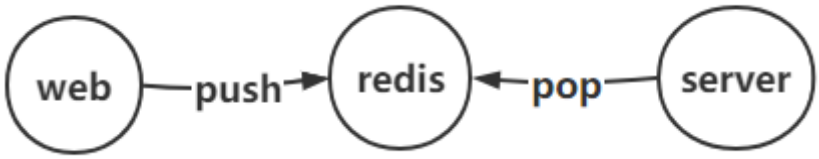
```
LPUSH + RPOP
# 或者
RPUSH + LPOP
```

阻塞队列（blocking queue）

```
LPUSH + BRPOP
# 或者
RPUSH + BLPOP
```

异步消息队列

> 操作与队列一样，但是在不同系统间；



获取固定窗口记录（战绩）

```
# 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；
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}'
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}'
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}'
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}'
lpush says '{["name"]:"零声教育【Darren老师】", ["text"]:"hello OVoice! hello to better self", ["picture"]:["url://image-20210601172745439.jpg", "url://image-20210601172741435.jpg"], timestamp = 1231231234}'
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}'
# 裁剪最近5条记录    战绩   近50条
ltrim says 0 4
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；

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

```
#  获取 key 对应 hash 中的 field 对应的值
HGET key field
#  设置 key 对应 hash 中的 field 对应的值
HSET key field value
#  设置多个hash键值对
HMSET key field1 value1 field2 value2 ... fieldn valuen
#  获取多个field的值
HMGET key field1 field2 ... fieldn
#  给 key 对应 hash 中的 field 对应的值加一个整数值
HINCRBY key field increment
#  获取 key 对应的 hash 有多少个键值对
HLEN key
#  删除 key 对应的 hash 的键值对，该键为field
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；

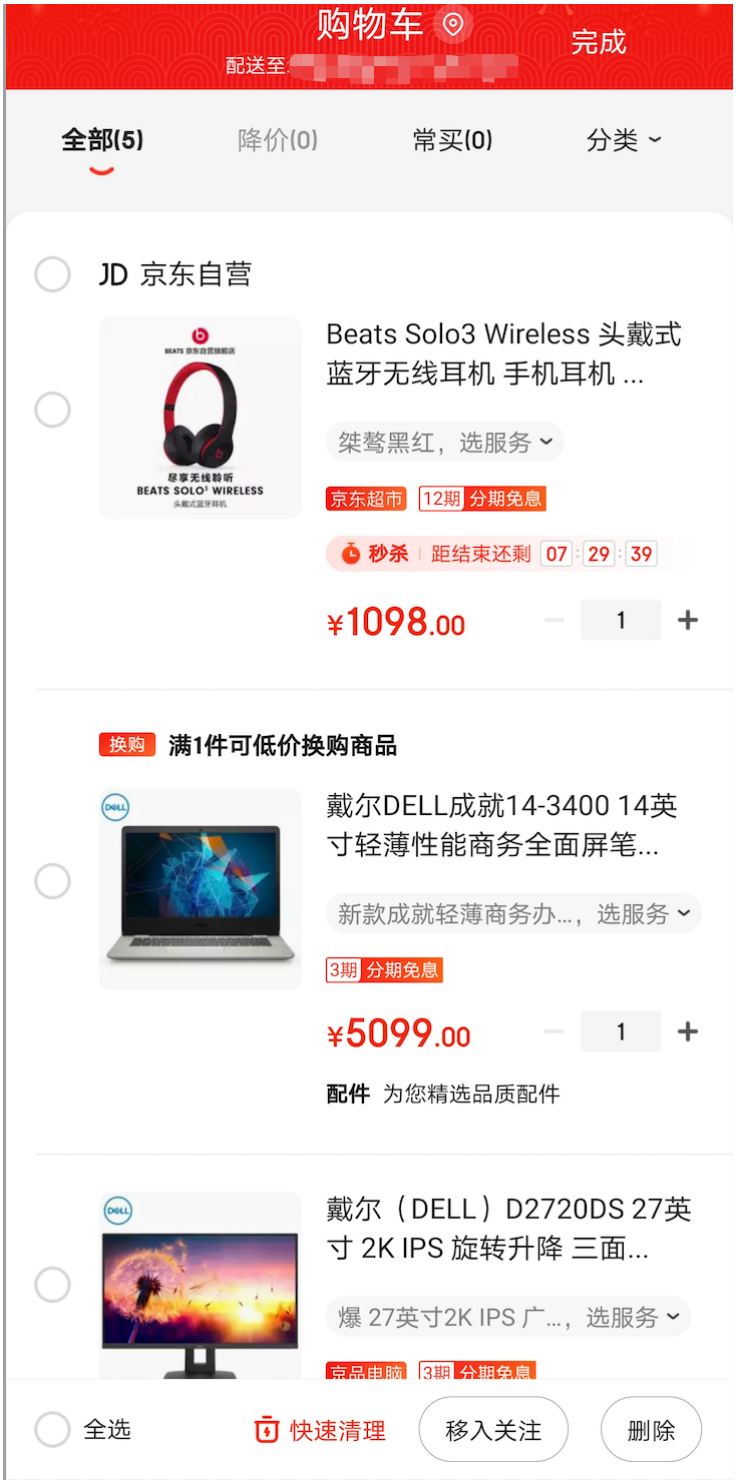
应用

存储对象

```
hmset hash:10001 name mark age 18 sex male
# 与 string 比较
set hash:10001 '{["name"]:"mark",["sex"]:"male",["age"]:18}'
# 假设现在修改 mark的年龄为19岁

# hash:
    hset hash:10001 age 19
# string:
    get role:10001
    # 将得到的字符串调用json解密，取出字段，修改 age 值
    # 再调用json加密
    set role:10001 '{["name"]:"mark",["sex"]:"male",["age"]:19}'
```

购物车




```
# 将用户id作为 key
# 商品id作为 field
# 商品数量作为 value
# 注意：这些物品是按照我们添加顺序来显示的；

# 添加商品：
    hset MyCart:10001 40001 1
    lpush MyItem:10001 40001
# 增加数量：
    hincrby MyCart:10001 40001 1
    hincrby MyCart:10001 40001 -1  // 减少数量1
# 显示所有物品数量：
    hlen MyCart:10001
# 删除商品：
    hdel MyCart:10001 40001
```

```
    lrem MyItem:10001 1 40001
# 获取所有物品：
    lrange MyItem:10001
    # 40001 40002 40003
    hget MyCart:10001 40001
    hget MyCart:10001 40002
    hget MyCart:10001 40003
```

set

> 集合；用来存储唯一性字段，不要求有序；

基础命令

```
# 添加一个或多个指定的member元素到集合的 key中
SADD key member [member ...]
# 计算集合元素个数
SCARD key

# SMEMBERS key
SMEMBERS key
# 返回成员 member 是否是存储的集合 key的成员
SISMEMBER key member

# 随机返回key集合中的一个或者多个元素，不删除这些元素
SRANDMEMBER key [count]
# 从存储在key的集合中移除并返回一个或多个随机元素
SPOP key [count]

# 返回一个集合与给定集合的差集的元素
SDIFF key [key ...]
# 返回指定所有的集合的成员的交集
SINTER key [key ...]
# 返回给定的多个集合的并集中的所有成员
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

```
# 添加抽奖用户
    sadd Award:1 10001 10002 10003 10004 10005 10006
    sadd Award:1 10009
# 查看所有抽奖用户
    smembers Award:1
# 抽取多名幸运用户
    srandmember Award:1 10
# 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

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

推荐好友

```
sadd follow:A mark king darren mole vico
sadd follow:C mark king darren
# C可能认识的人：
    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

```
# 添加到键为key有序集合（sorted set）里面
ZADD key [NX|XX] [CH] [INCR] score member [score member ...]
# 从键为key有序集合中删除 member 的键值对
ZREM key member [member ...]
# 返回有序集key中，成员member的score值
ZSCORE key member
# 为有序集key的成员member的score值加上增量increment
ZINCRBY key increment member
# 返回key的有序集元素个数
ZCARD key
# 返回有序集key中成员member的排名
ZRANK key member
# 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
ZRANGE key start stop [WITHSCORES]
# 返回有序集key中，指定区间内的成员(逆序)
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） o(logn)

应用

百度热榜


```
# 点击新闻：
    zincrby hot:20210601 1 10001
    zincrby hot:20210601 1 10002
    zincrby hot:20210601 1 10003
    zincrby hot:20210601 1 10004
    zincrby hot:20210601 1 10005
    zincrby hot:20210601 1 10006
    zincrby hot:20210601 1 10007
    zincrby hot:20210601 1 10008
    zincrby hot:20210601 1 10009
    zincrby hot:20210601 1 10010

# 获取排行榜：
    zrevrange hot:20210601 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。

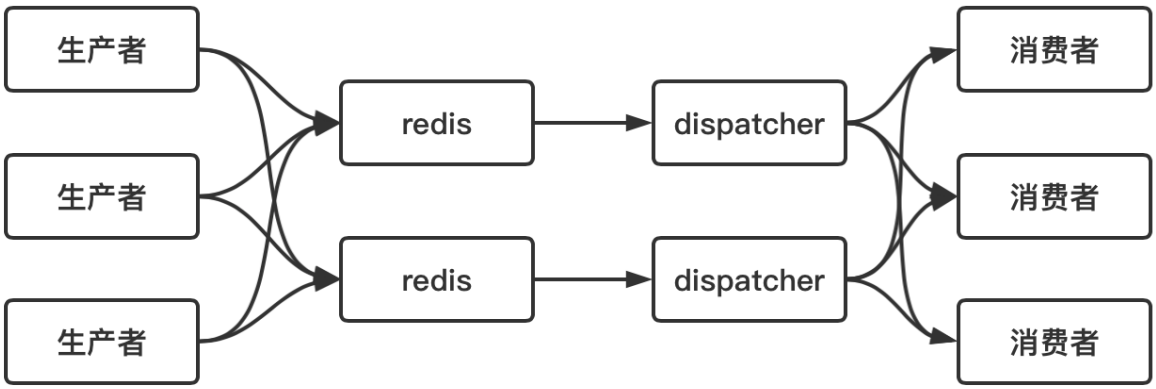
```
def delay(msg):
    msg.id = str(uuid.uuid4()) #保证 member 唯一
    value = json.dumps(msg)
    retry_ts = time.time() + 5 # 5s后重试
    redis.zadd("delay-queue", retry_ts, value)

```

```python
# 使用连接池
def loop():
    while True:
        values = redis.zrangebyscore("delay-queue", 0, time.time(), start=0, num=1)
        if not values:
            time.sleep(1)
            continue
        value = values[0]
        success = redis.zrem("delay-queue", value)
        if success:
            msg = json.loads(value)
            handle_msg(msg)

# 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，
# 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
# 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；

```lua
# 指定用户 user_id 的某个行为 action 在特定时间内 period 只允许发生做多的次数 max_count

local function is_action_allowed(red, userid, action, period, max_count)
    local key = tab_concat({"hist", userid, action}, ":")
    local now = zv.time()
    red:init_pipeline()
    -- 记录行为
    red:zadd(key, now, now)
    -- 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
    red:zremrangebyscore(key, 0, now - period *100)
    -- 获取时间窗口内的行为数量
    red:zcard(key)
    -- 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
    red:expire(key, period + 1)
    local res = red:commit_pipeline()
    return res[3] <= max_count
```

```
end



# 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；
# 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
# 注意：如果用 key + expire 操作也能实现，但是实现的是熔断，维护时间窗口是限流的功能；
```