

# | Redis使用与原理



# Redis简介

## 什么是Redis

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster。

Redis是开源（BSD许可）的，数据结构存储于内存中，被用来作为数据库，缓存和消息代理。它支持多种数据结构，例如字符串（string），哈希（hash），列表（list），集合（set），带范围查询的排序集合（zset），位图（bitmap），hyperloglog，带有半径查询和流的地理空间索引。Redis具有内置的复制，Lua脚本，LRU逐出，事务和不同级别的磁盘持久性，并通过Redis Sentinel和Redis Cluster自动分区提供高可用性。

# Redis安装篇

<https://redis.io/download>

1> 下载Redis

2> 进入Redis文件夹目录

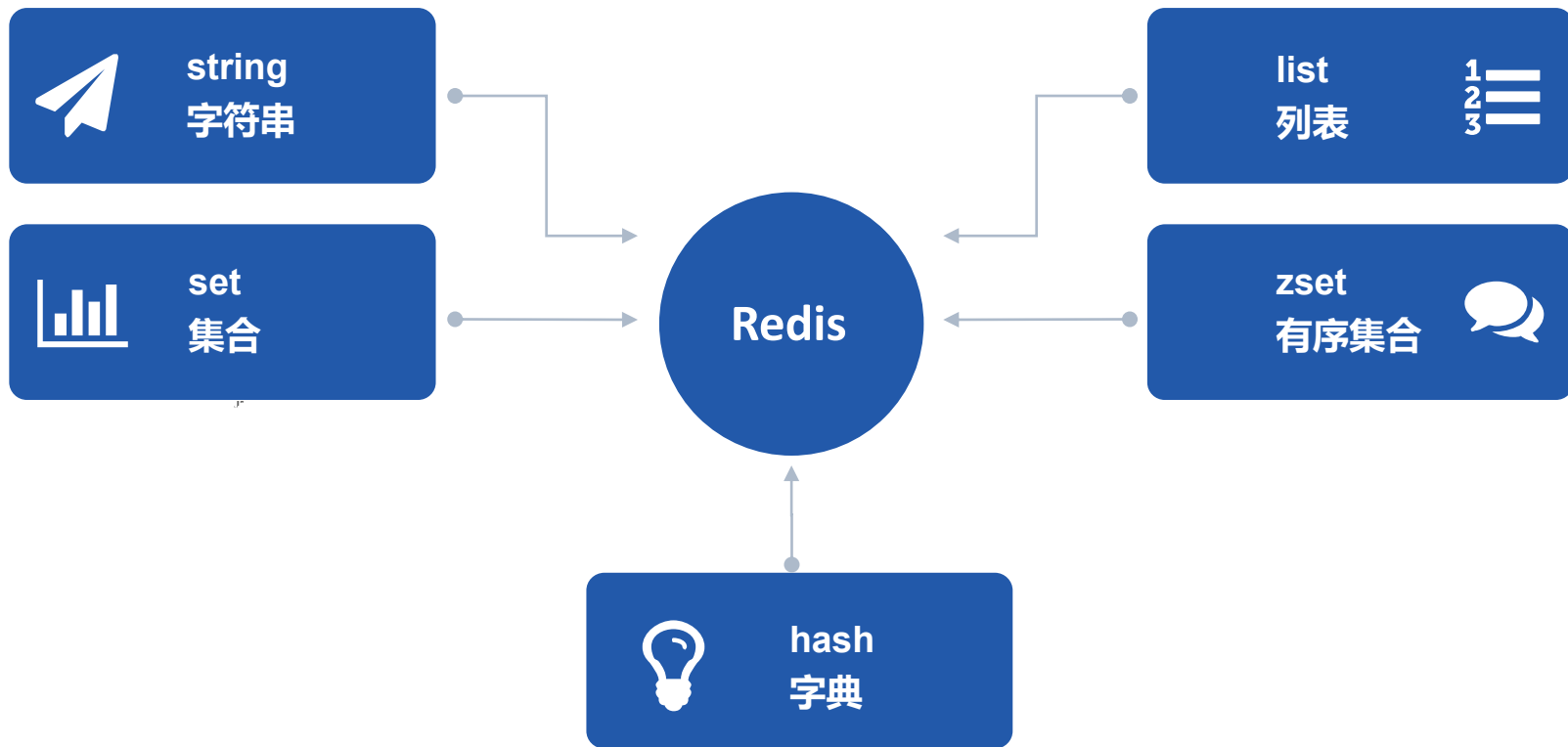
3> 执行make安装

4> `src/redis-server --daemonize yes`

5> `src/redis-cli`

# Redis基础篇

## Redis支持的数据类型



# Redis对象源码

```
redis.h
/*
 * redis对象
 */
typedef struct redisObject {
    // 对象的类型 (取值范围: REDIS_STRING, REDIS_LIST, REDIS_HASH, REDIS_SET, REDIS_ZSET)
    unsigned type:4;

    // 对象的编码 (取值范围: REDIS_ENCODING_INT, REDIS_ENCODING_EMBSTR, REDIS_ENCODING_RAW, REDIS_ENCODING_HT,
    REDIS_ENCODING_LINKEDLIST, REDIS_ENCODING_ZIPLIST, REDIS_ENCODING_INTSET, REDIS_ENCODING_SKIPLIST)
    unsigned encoding:4;

    // 指向底层实现数据结构的指针
    void *ptr;

    unsigned notused:2;    /* Not used */
    unsigned lru:22;       /* lru time (relative to server.lruclock) */
    int refcount;
} robj;
```

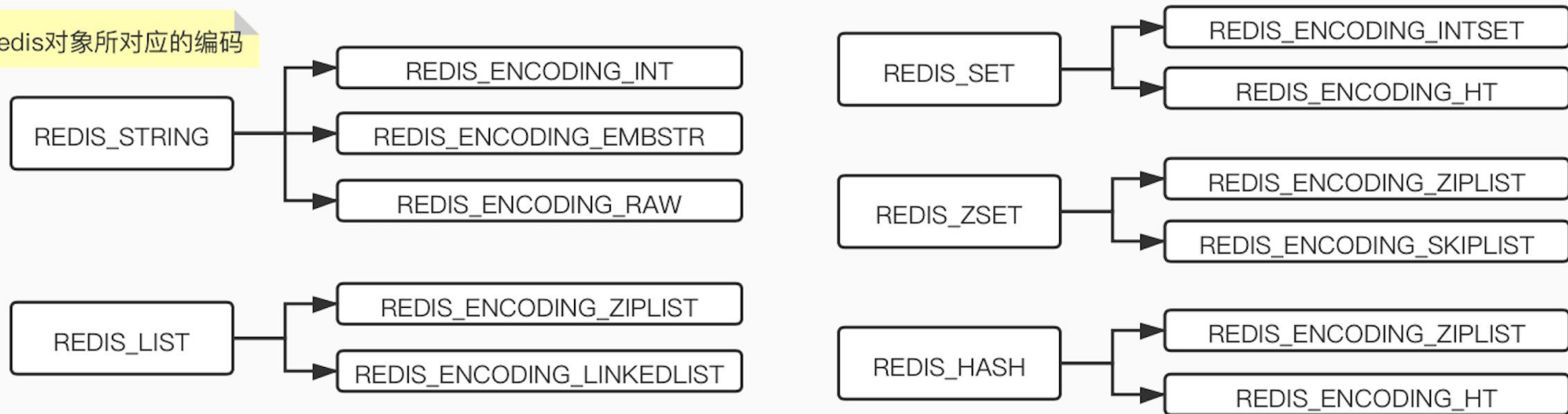


# Redis对象类型——type&encoding

Redis对象类型



Redis对象所对应的编码



## 编码与数据结构实现——encoding & ptr

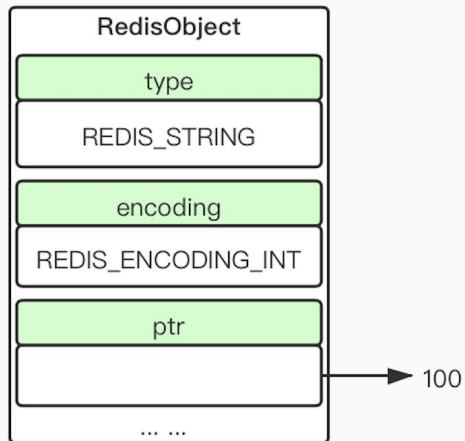
encoding常量	编码所对应的底层数据结构
REDIS_ENCODING_INT	long类型的整数
REDIS_ENCODING_EMBSTR	embstr编码的简单动态字符串
REDIS_ENCODING_RAW	简单动态字符串
REDIS_ENCODING_HT	字典
REDIS_ENCODING_LINKEDLIST	双向链表
REDIS_ENCODING_ZIPLIST	压缩列表
REDIS_ENCODING_INTSET	整数集合
REDIS_ENCODING_SKIPLIST	跳表和字典

## string 类型

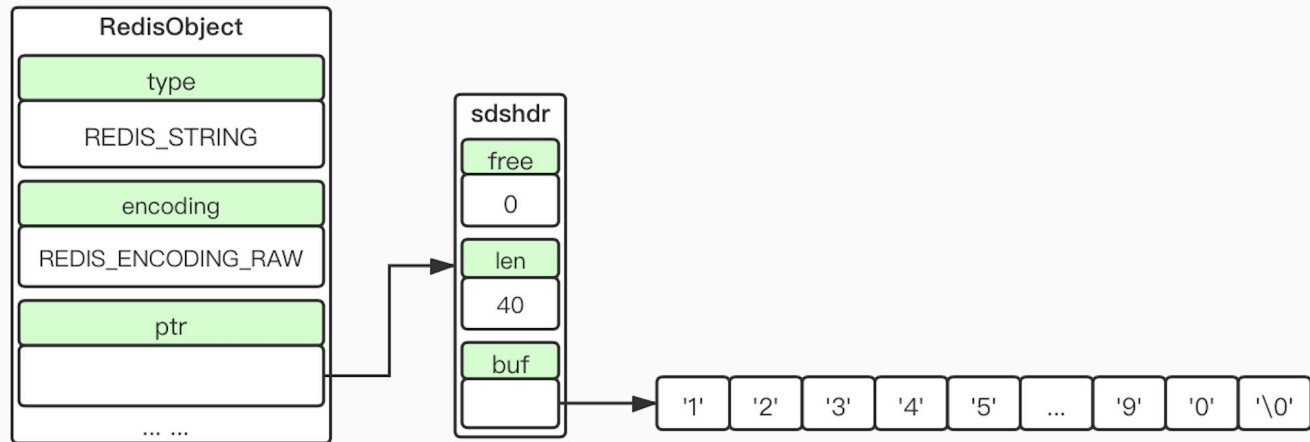
命令行	含义
set key value	赋值key的值为value
get key	获取key的value值
del key	删除key
expire key seconds	设置key在seconds秒后过期
setex key value	如果key存在，则将值更新为value
ttl key	查看key还有多久过期
setnx key value	如果key不存在，才新增key和value
strlen key	计算指定key的值的长度
incr key	加1
incrby key numbers	指定增加值，numbers可以是负值
mset key1 value1 key2 value2 ...	批量添加
mget key1 key2 key3 ...	批量获取

## 字符串内部实现——int 编码 & raw 编码

String对象的int 编码

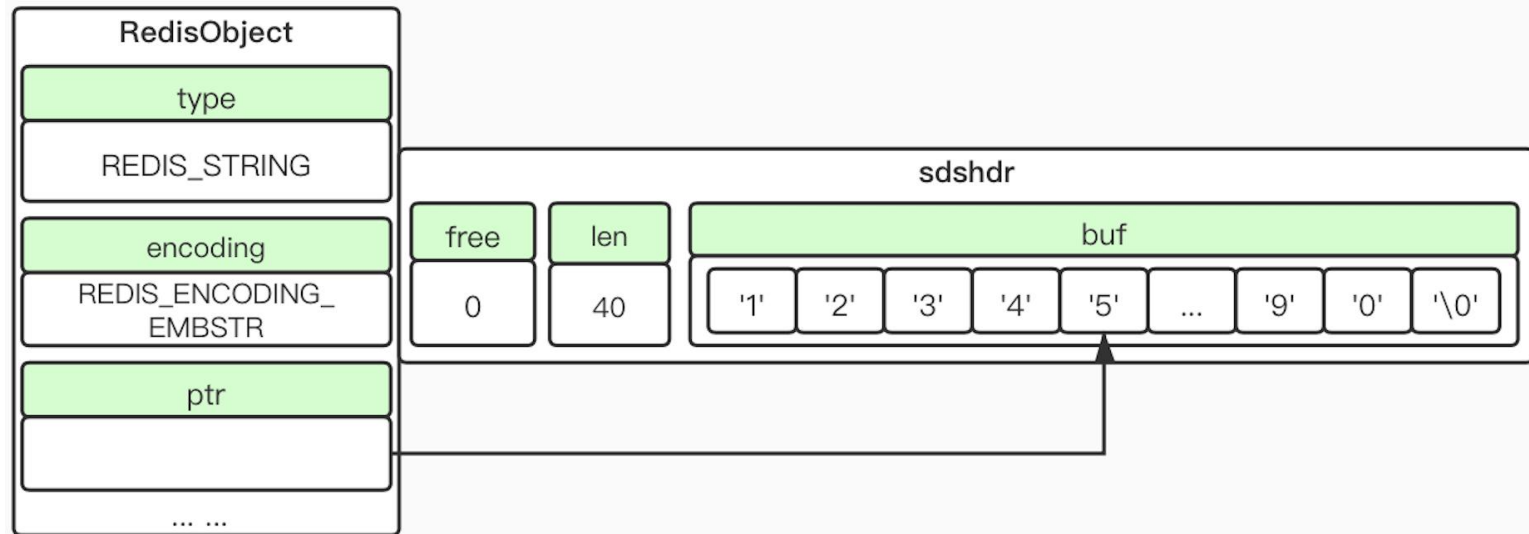


String对象的raw 编码



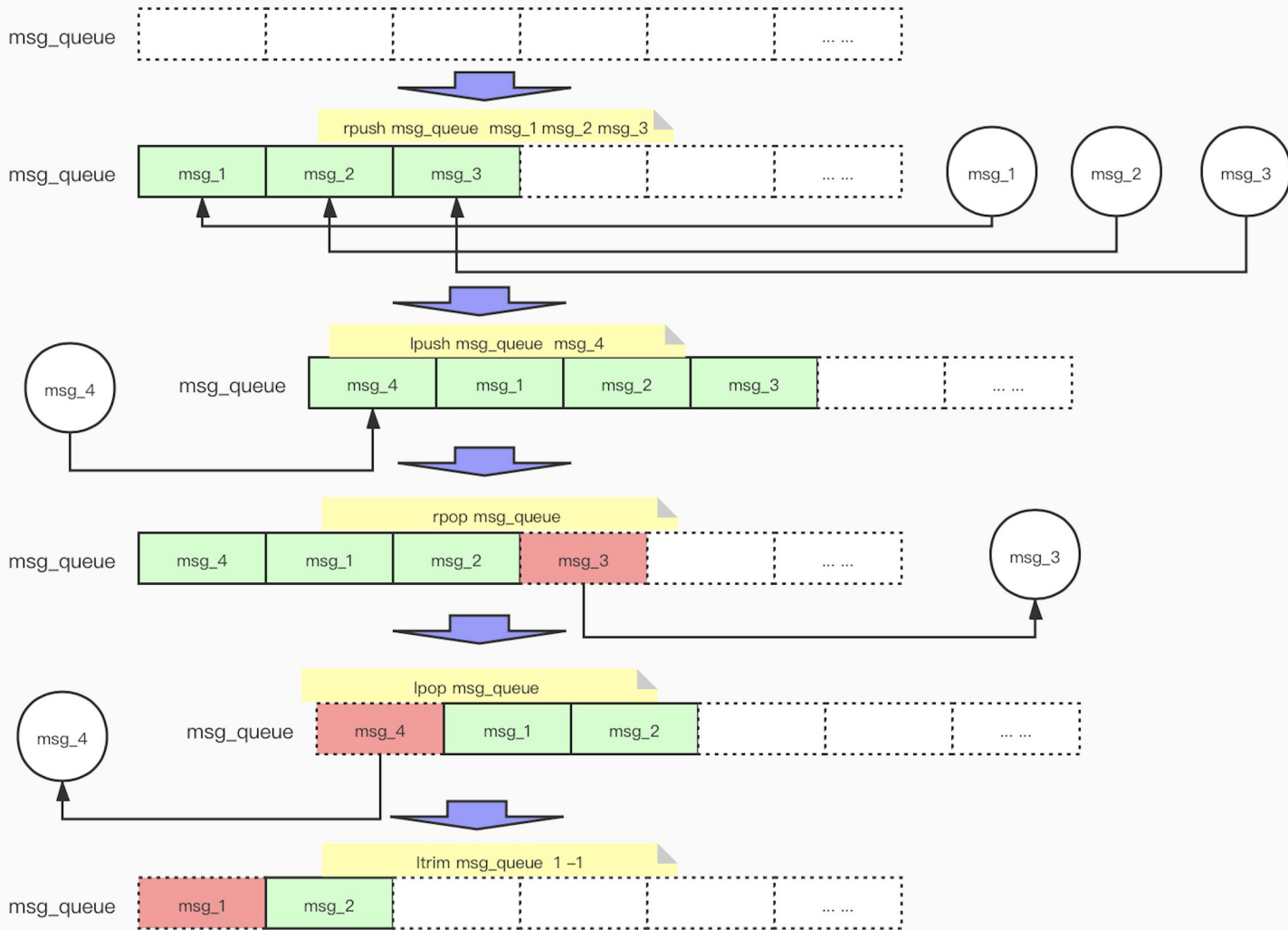
## 字符串内部实现——embstr编码

String对象的embstr 编码



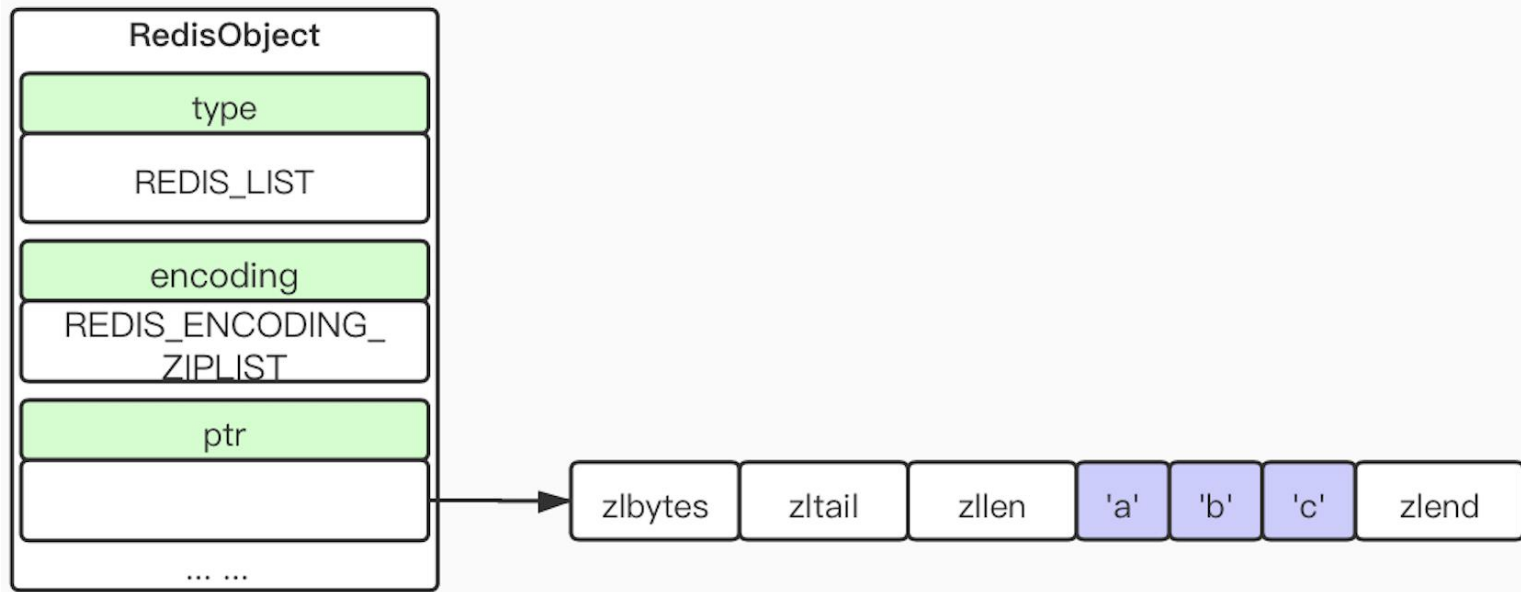
命令行	含义
lpush key value1 value2	左侧插入value
rpush key value1 value2	右侧插入value
lpop key	左侧弹出value
rpop key	右侧弹出value
llen key	查看key的长度
lindex key index	查看列表中某个index对应的value值
setnx key value	如果key不存在，才新增key和value
lrange key startIndex endIndex	查看指定元素，下标从0开始，-1为倒数第一个。
ltrim key startIndex endIndex	下标同上。

# 列表操作



## 列表的内部实现——ziplist编码

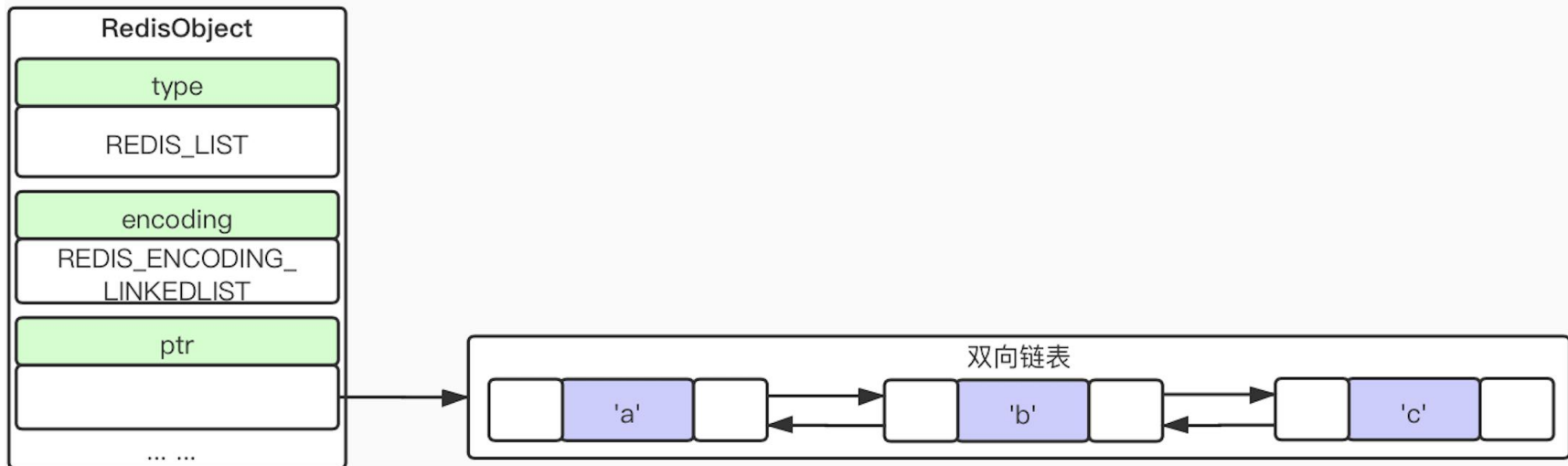
list对象的ziplist编码





## 列表的内部实现——linkedlist编码

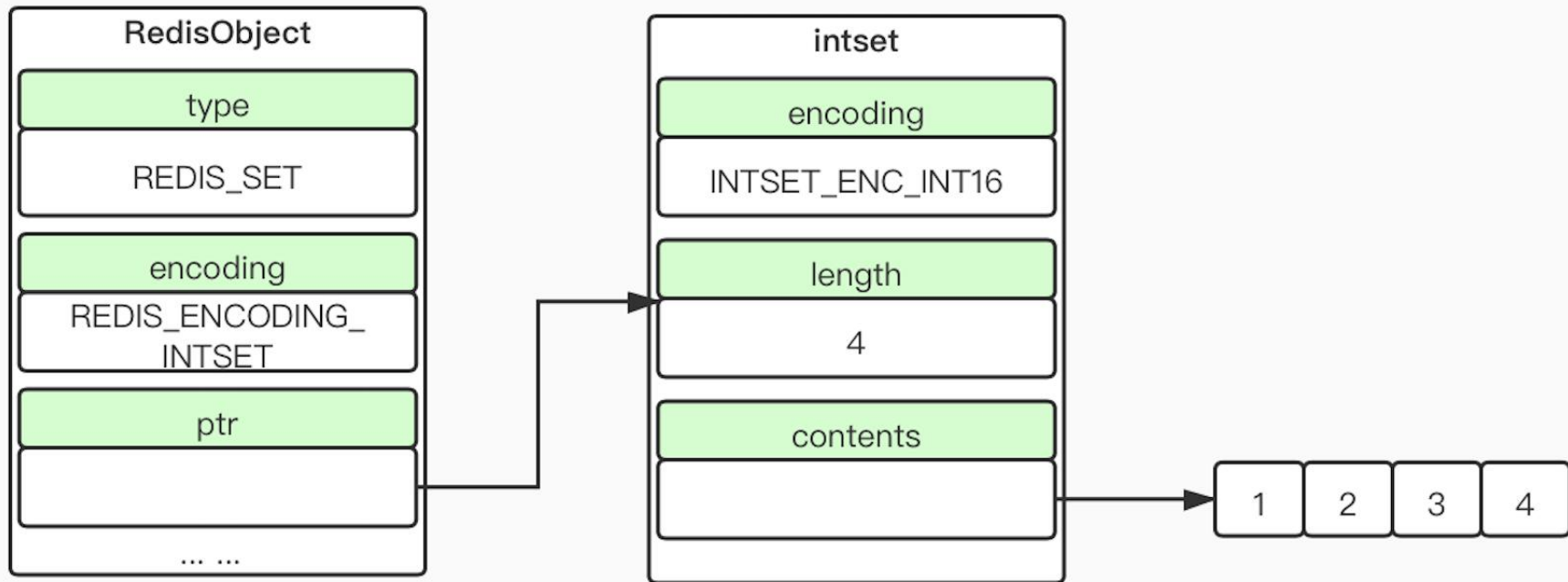
list对象的linkedlist编码



命令行	含义
sadd key value1 value2	添加元素到集合中
smembers key	查看集合中的所有元素
sismember key value	查看value是否在集合中
scard key	查询集合的长度
spop key	取出集合中的一个元素
del key	删除集合

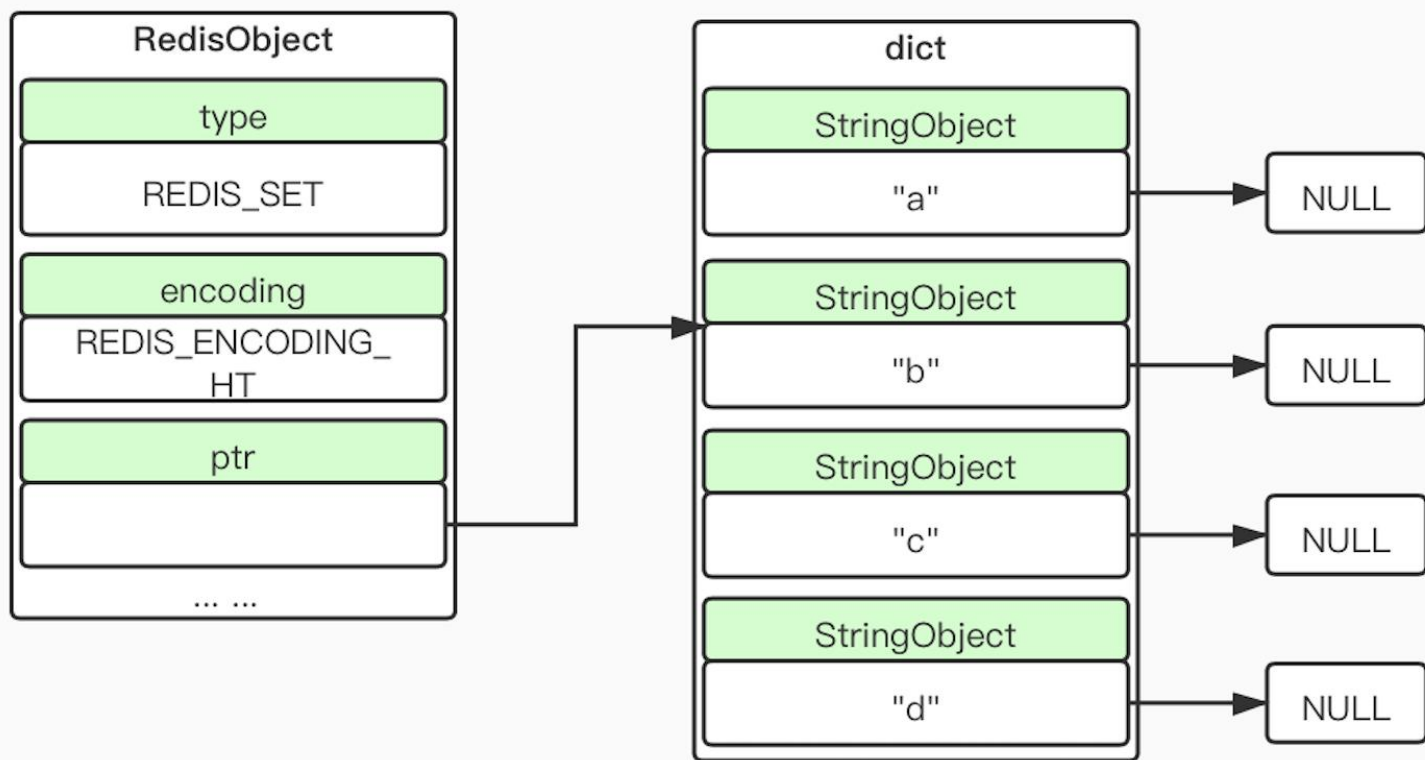
## 集合类型内部实现——intset编码

### 集合对象的intset编码



## 集合类型内部实现——hashtable编码

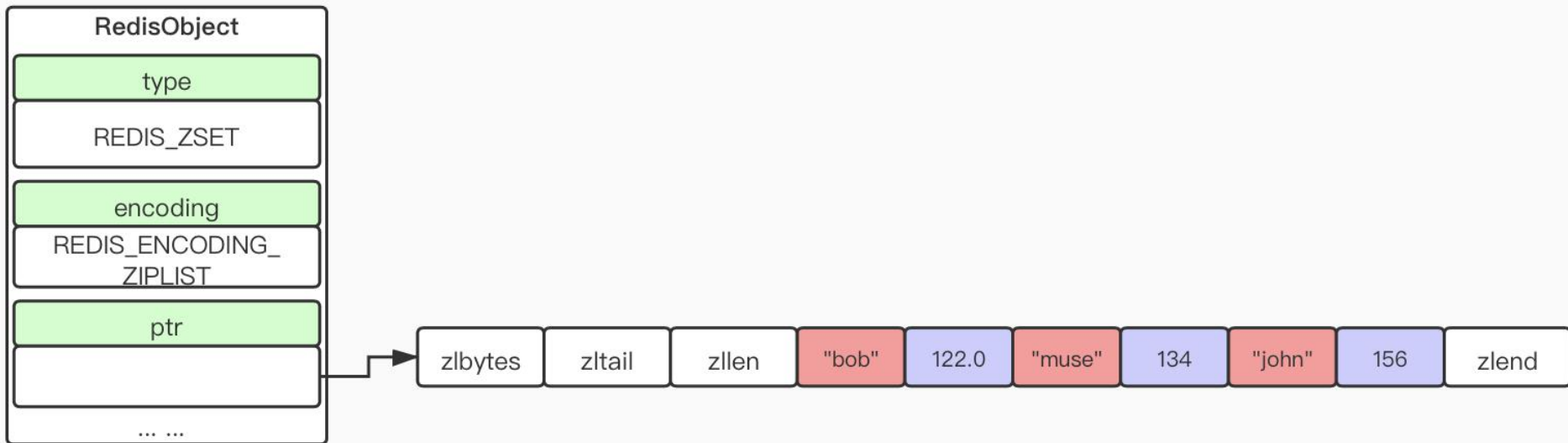
集合对象的hashtable编码



命令行	含义
<code>zadd key value1 score1 value2 score2</code>	添加元素到有序集合中
<code>zscore key value</code>	查看key的score值，输出 $\text{score} \geq \text{负无穷}$ ， $\text{score} \leq \text{正无穷}$ 的所有元素
<code>zrange key 0 -1</code>	正序输出
<code>zrangebyscore key -inf +inf</code>	正序输出
<code>zrevrange key 0 -1</code>	倒序输出
<code>zcard key</code>	查看key中的元素个数
<code>zrangebyscore key indexStart endStart</code>	获得key中 $\text{score} \geq \text{indexStart}$ 且 $\text{score} \leq \text{endStart}$ 的元素，正序排列
<code>zrevrangebyscore key indexStart endStart</code>	同上，倒序排列
<code>zrem key value</code>	删除key中的元素value

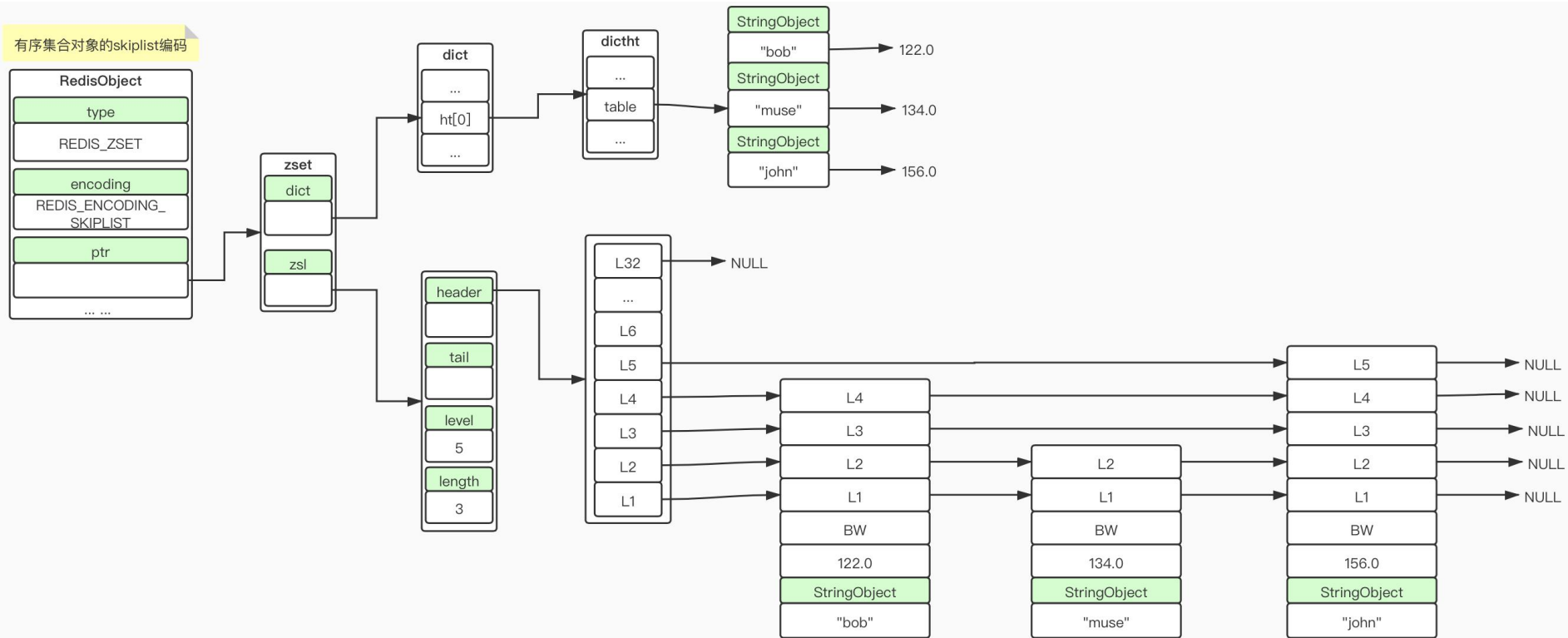
## 有序集合类型内部实现——ziplist编码

有序集合对象的ziplist编码



# 有序集合类型内部实现——skiplist编码

有序集合对象的skiplist编码

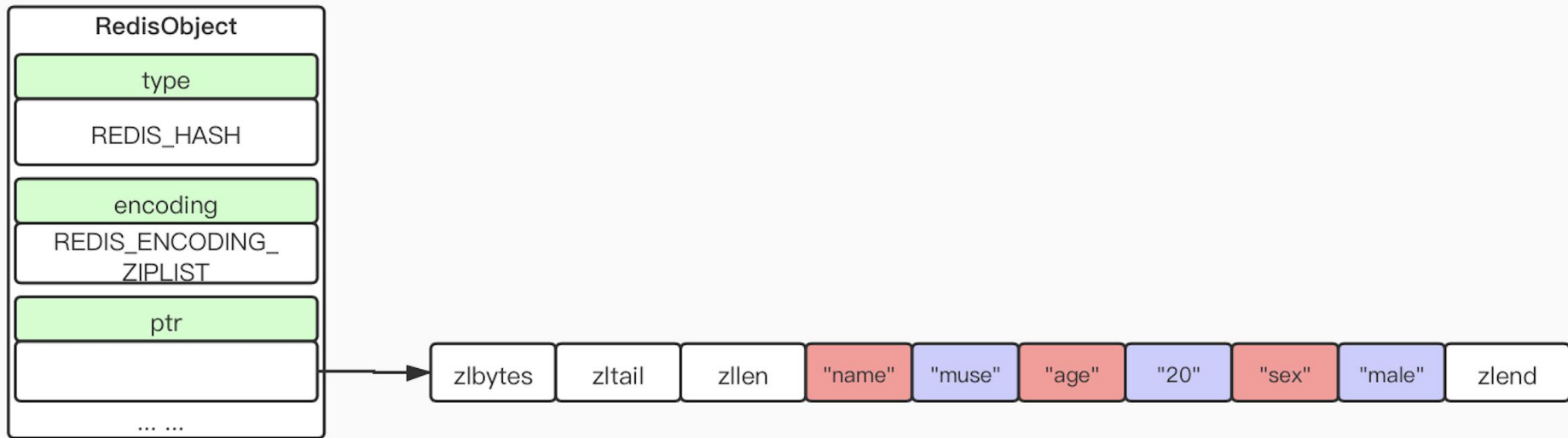


命令行	含义
hset key name value	添加属性元素name和value到key中
hget key name	查看key的name值
hmset key name1 value1 name2 value2	批量添加key的属性元素
hmget key name1 name2	批量获取key的属性元素
hlen key	获得key的属性元素个数
hgetall key	查询key中的所有元素

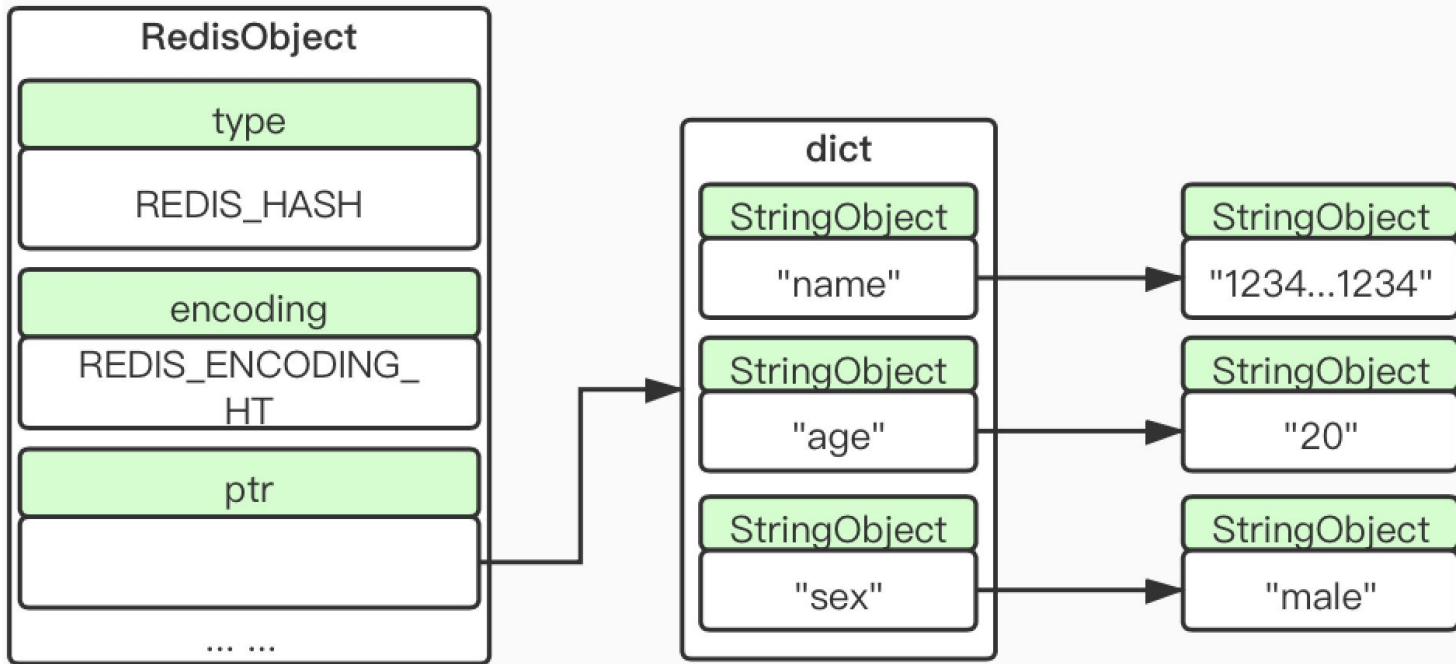


## 哈希类型内部实现——ziplist编码

hash对象的ziplist 编码

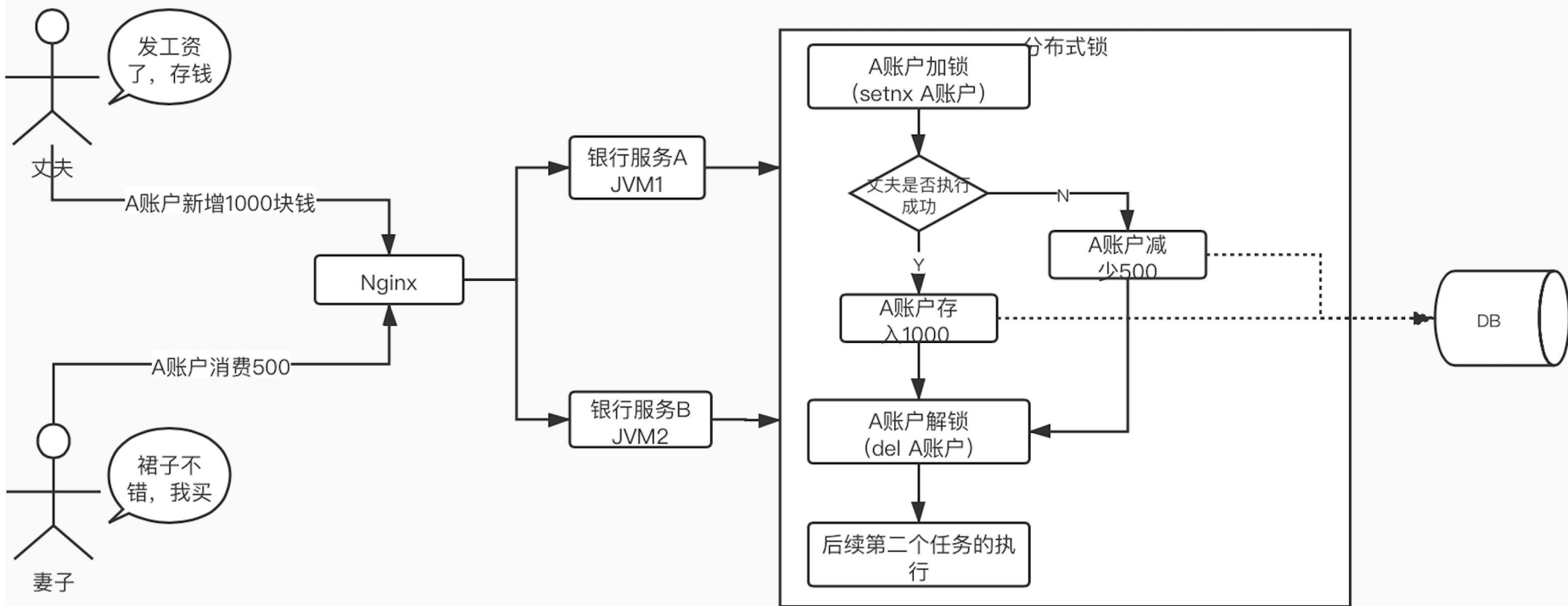


hash对象的hashtable 编码

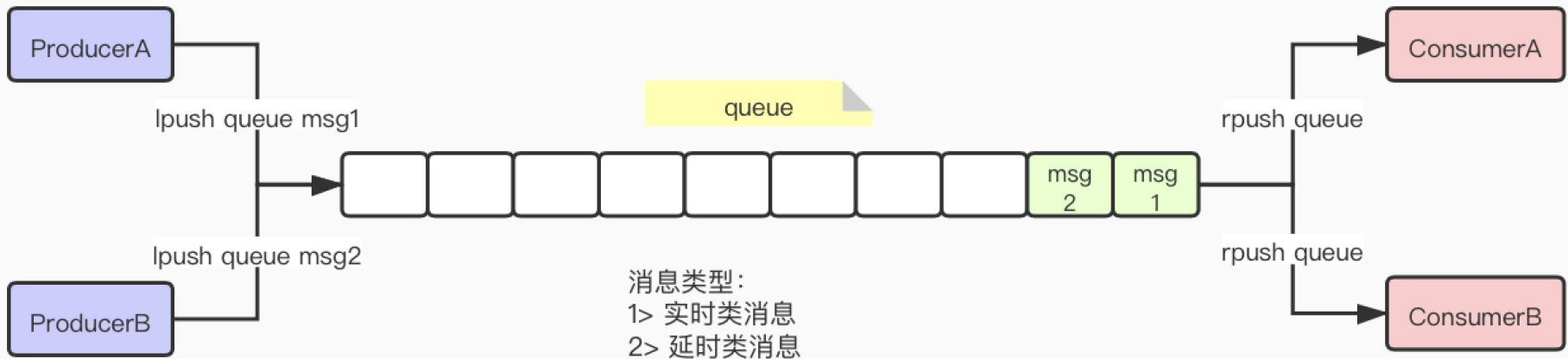


# Redis应用

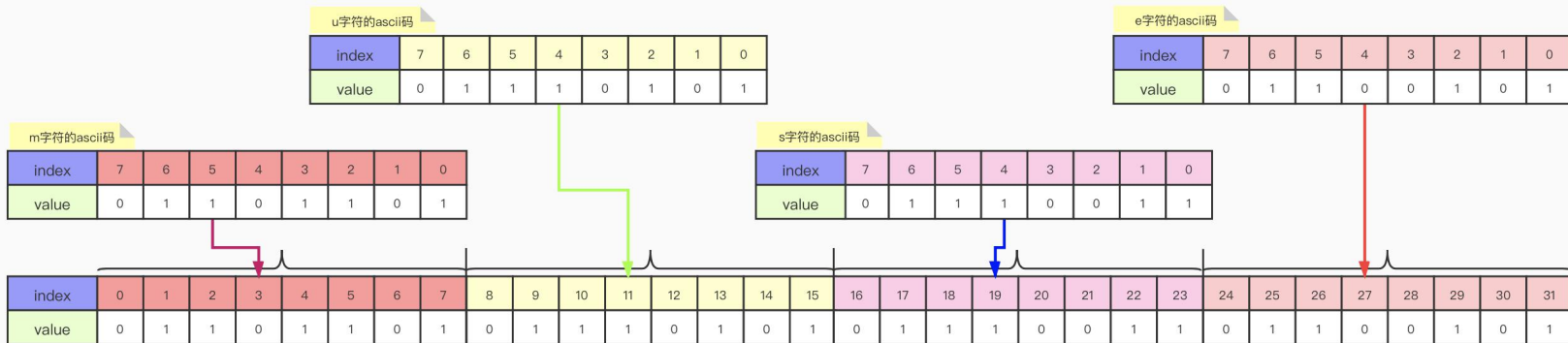
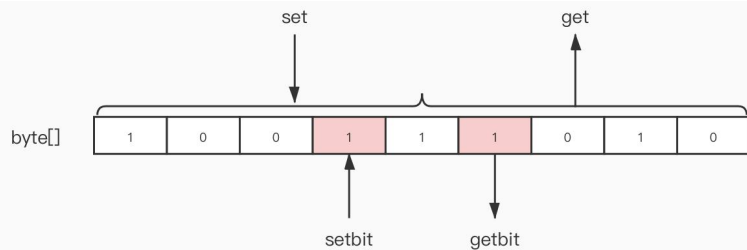
# 分布式锁



# 消息队列

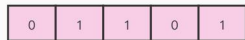
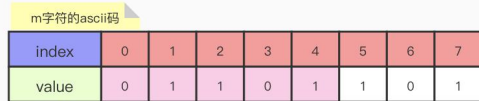


# 位图



127.0.0.1:6379> BITFIELD bitmap get u5 0  
1) (integer) 13

从index为0开始, 获取5位无符号数



$$2^0 + 2^2 + 2^3 = 1 + 4 + 8 = 13$$



# Redis的持久化

## RDB相关参数

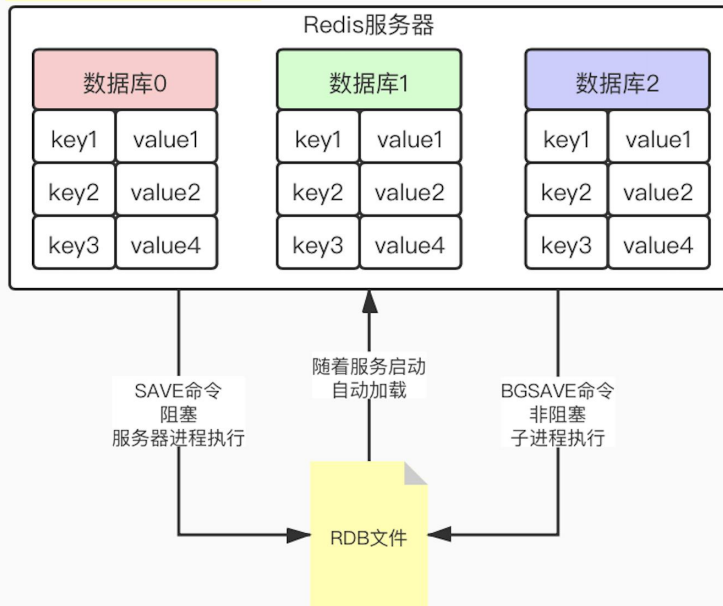
```
redis.h
/*-----
 * Global server state
 *-----*/

struct redisServer {
    ...
    // 保存saveparam的数组
    struct saveparam *saveparams; /* Save points array for RDB */
    // 修改计数器, 记录上一次成功执行SAVE或BGSAVE后, 数据进行了多少次修改 (包括写入、删除、更新等操作)。
    // set name "muse" dirty计数器+1
    // sadd name "bob" "tom" "john" "sam" dirty计数器+4
    long long dirty; /* Changes to DB from the last save */
    // 上一次执行保存的时间, 记录上一次成功执行SAVE或BGSAVE的时间
    time_t lastsave; /* Unix time of last successful save */
    ...
}
```

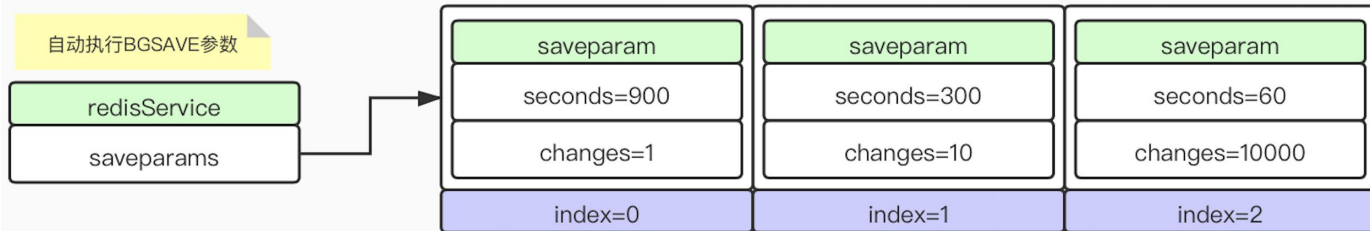
```
redis.h
struct saveparam {
    // 执行的秒数
    time_t seconds;
    // 修改的次数
    int changes;
};
```



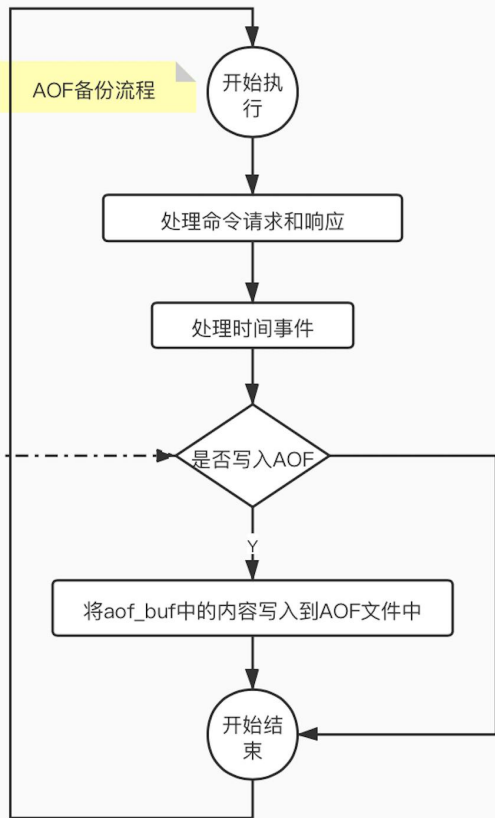
## Redis数据的备份与恢复



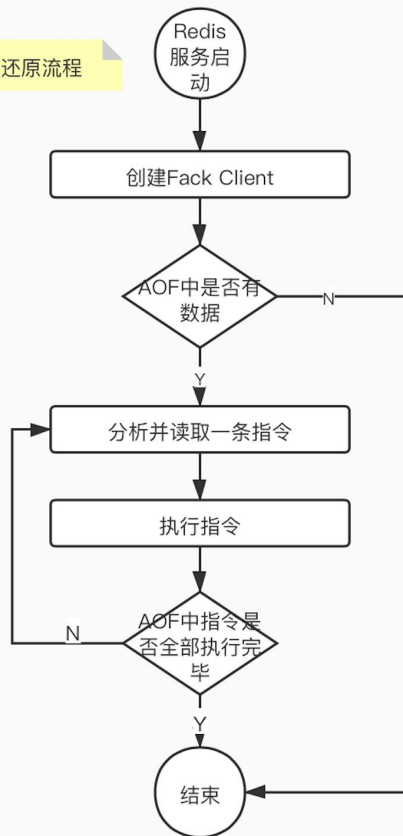
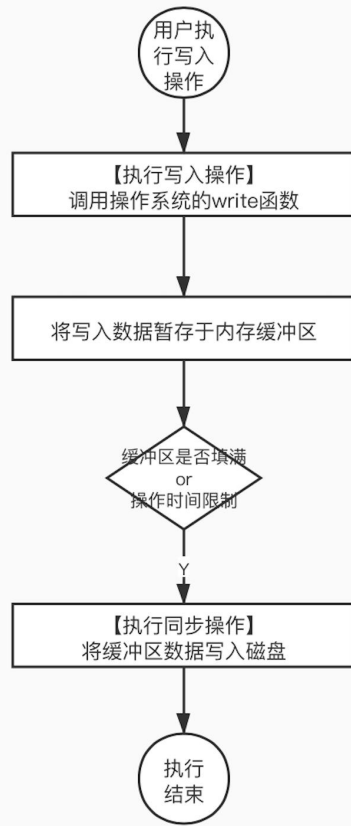
## 自动执行BGSAVE参数



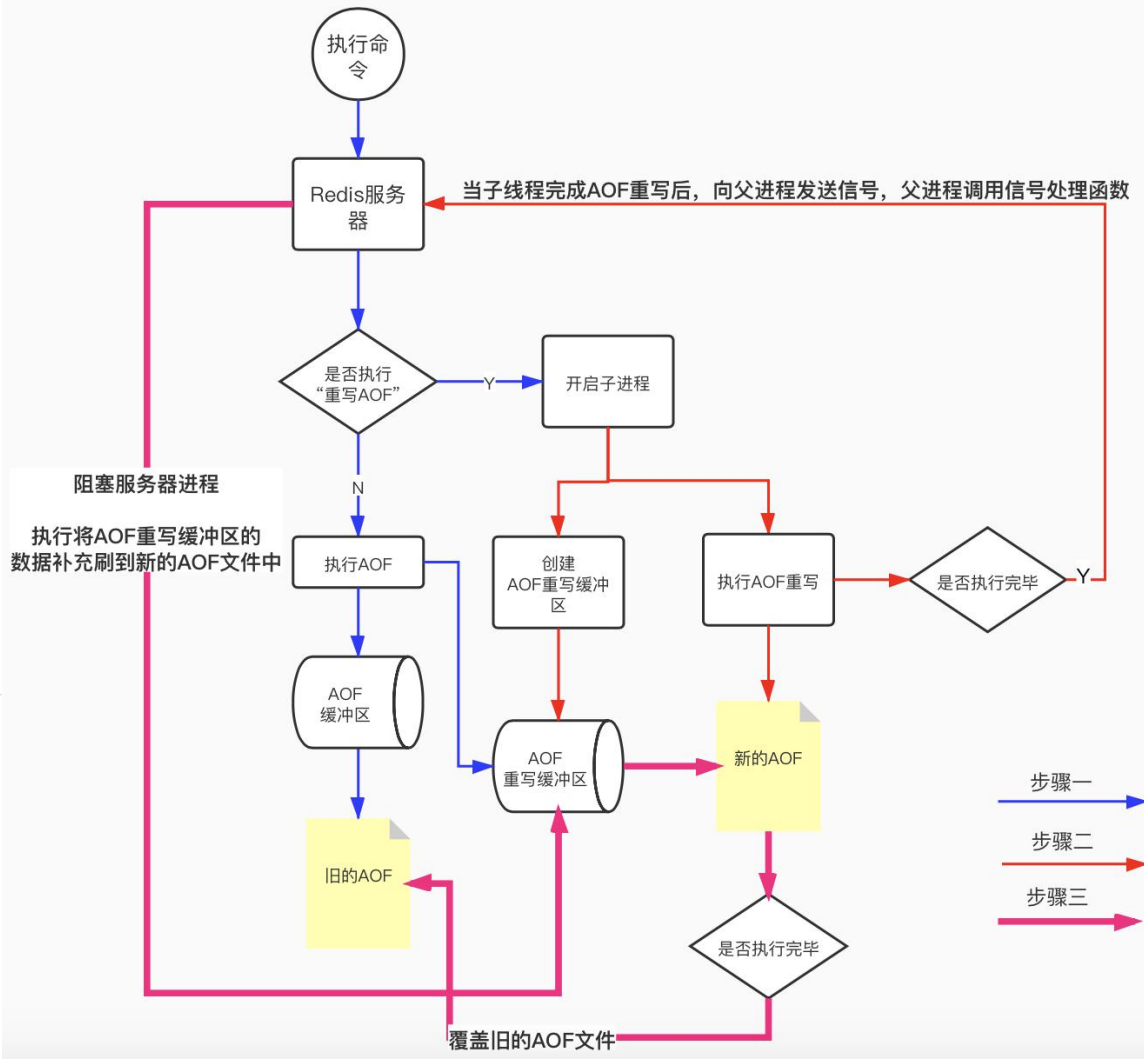
AOF备份流程



AOF还原流程

操作系统层面  
写入与同步

# AOF重写



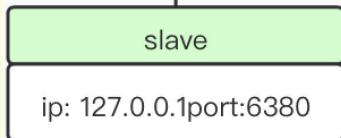


# Redis集群搭建

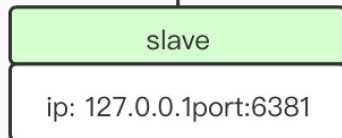
```
port 6379
bind 127.0.0.1
requirepass "myredis"
daemonize yes
logfile "6379.log"
dbfilename "dump-6379.rdb"
dir "/opt/soft/redis/data"
#如若master设置了认证密码，那么所有redis数据节点都配置上masterauth属性
masterauth "myredis"
```



```
port 6380bind
127.0.0.1requirepass
"myredis"daemonize yeslogfile
"6380.log"dbfilename "dump-
6380.rdb"dir
"/opt/soft/redis/data"
masterauth "myredis"slaveof
127.0.0.1 6379
```



```
port 6381bind
127.0.0.1requirepass
"myredis"daemonize yeslogfile
"6381.log"dbfilename "dump-
6381.rdb"dir
"/opt/soft/redis/data"
masterauth "myredis"slaveof
127.0.0.1 6379
```



## 运维实战书籍推荐

