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

单机安装

https://redis.io/download

1>下载Redis

2> 进入Redis文件夹目录

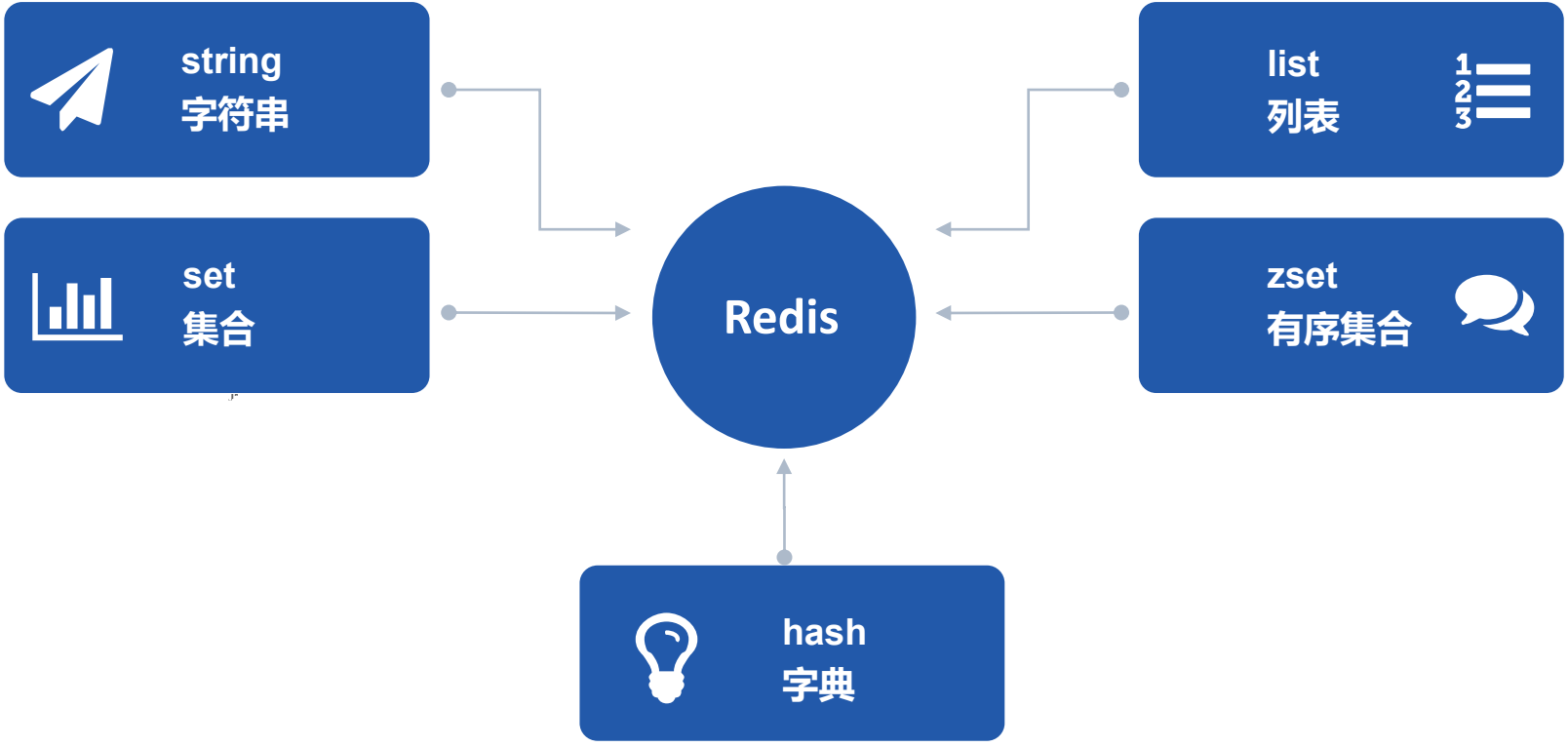
3> 执行make安装

4> src/redis-server --daemonize yes

5> src/redis-cli

Redis基础篇

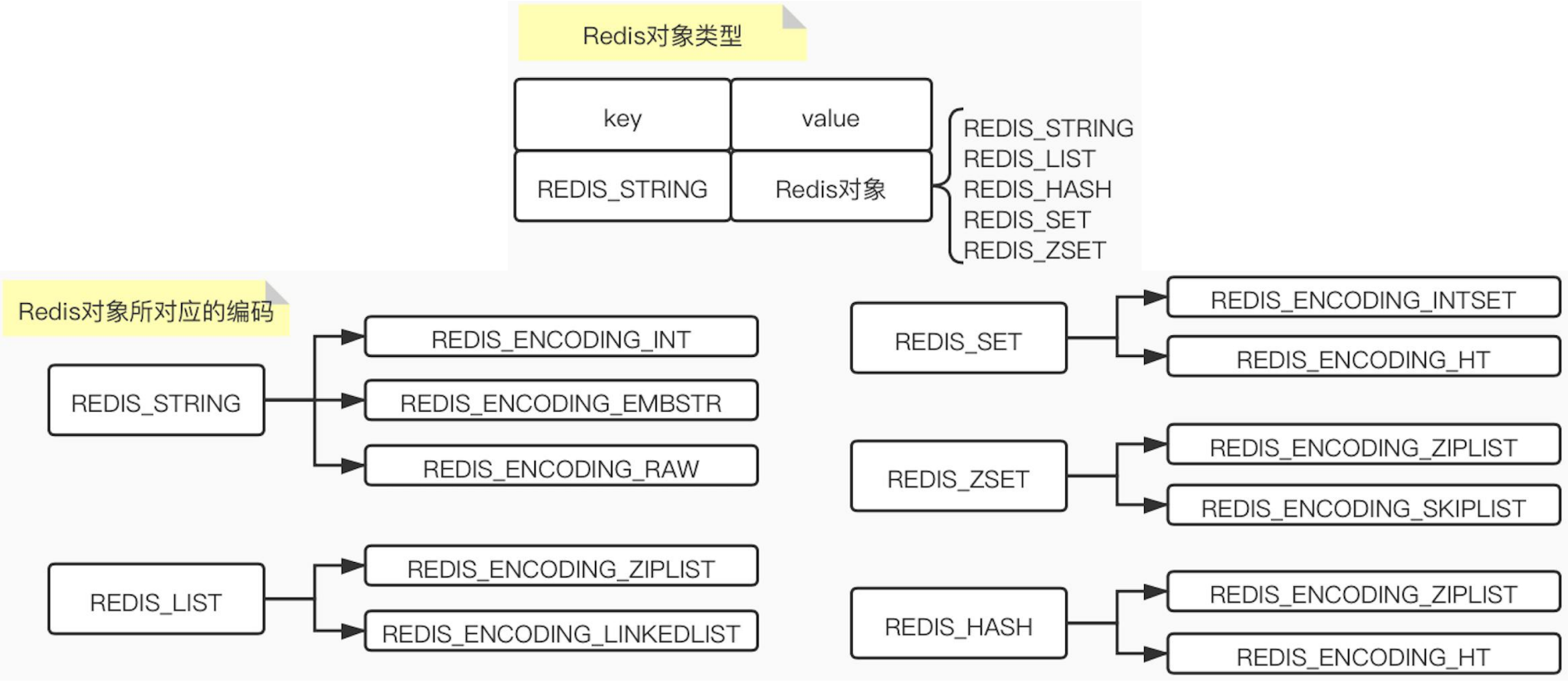
Redis支持的数据类型



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



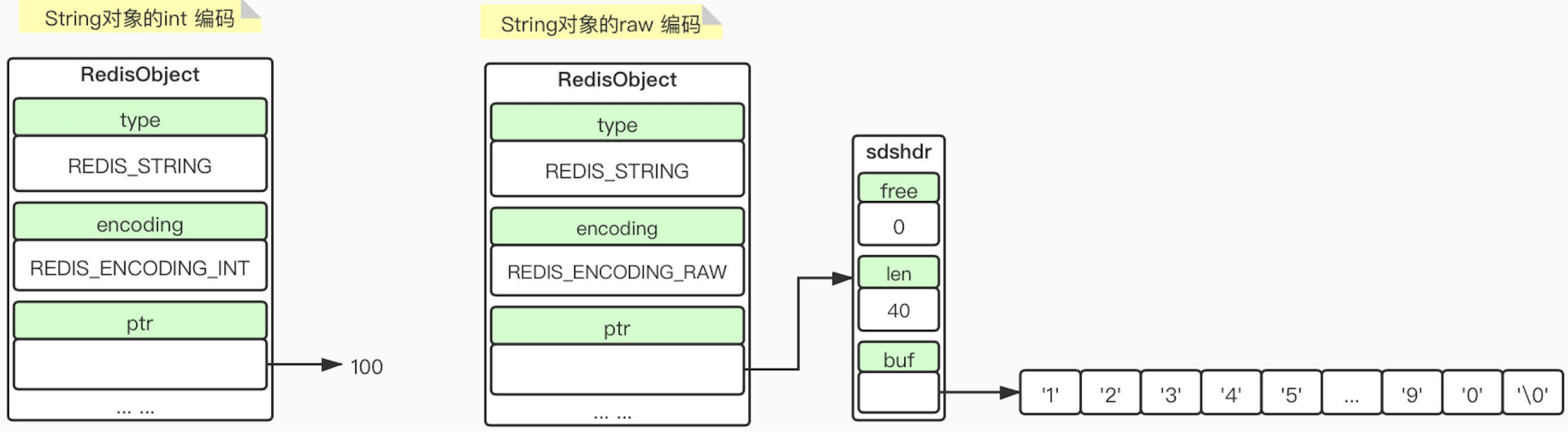
编码与数据结构实现——encoding&ptr

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

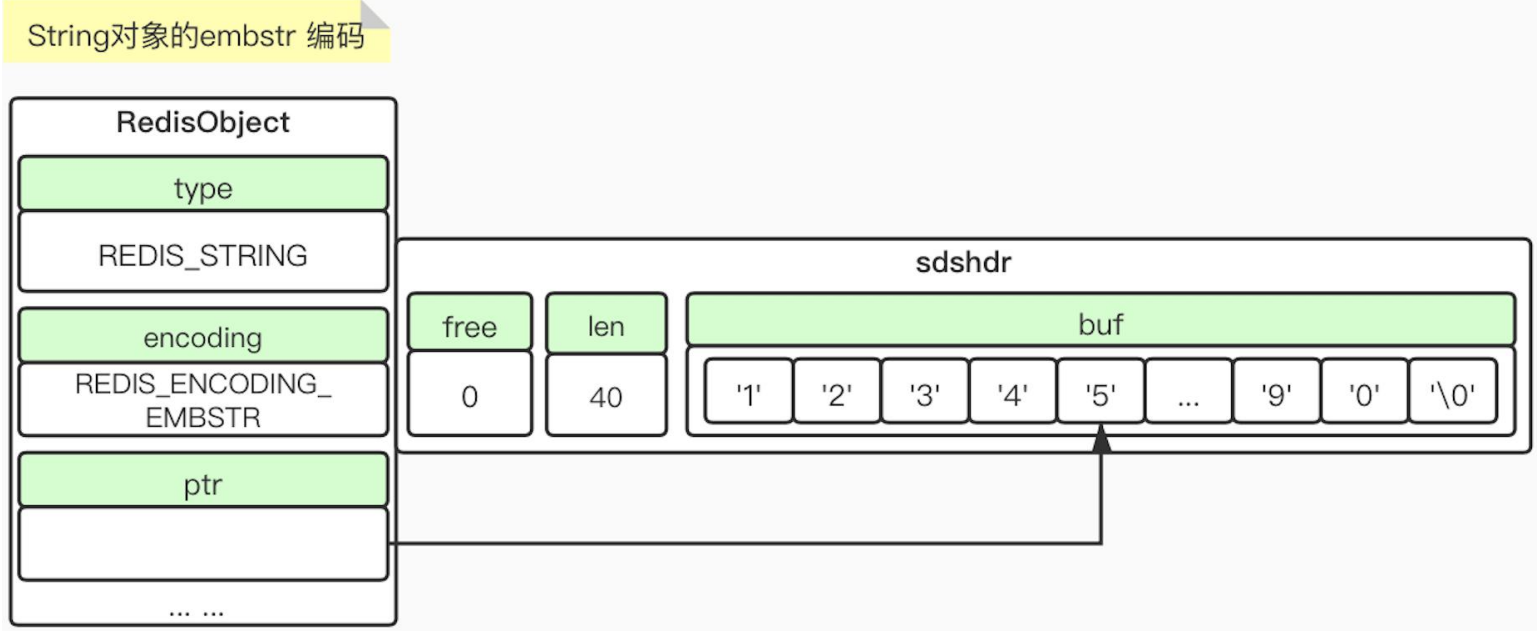
string类型

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



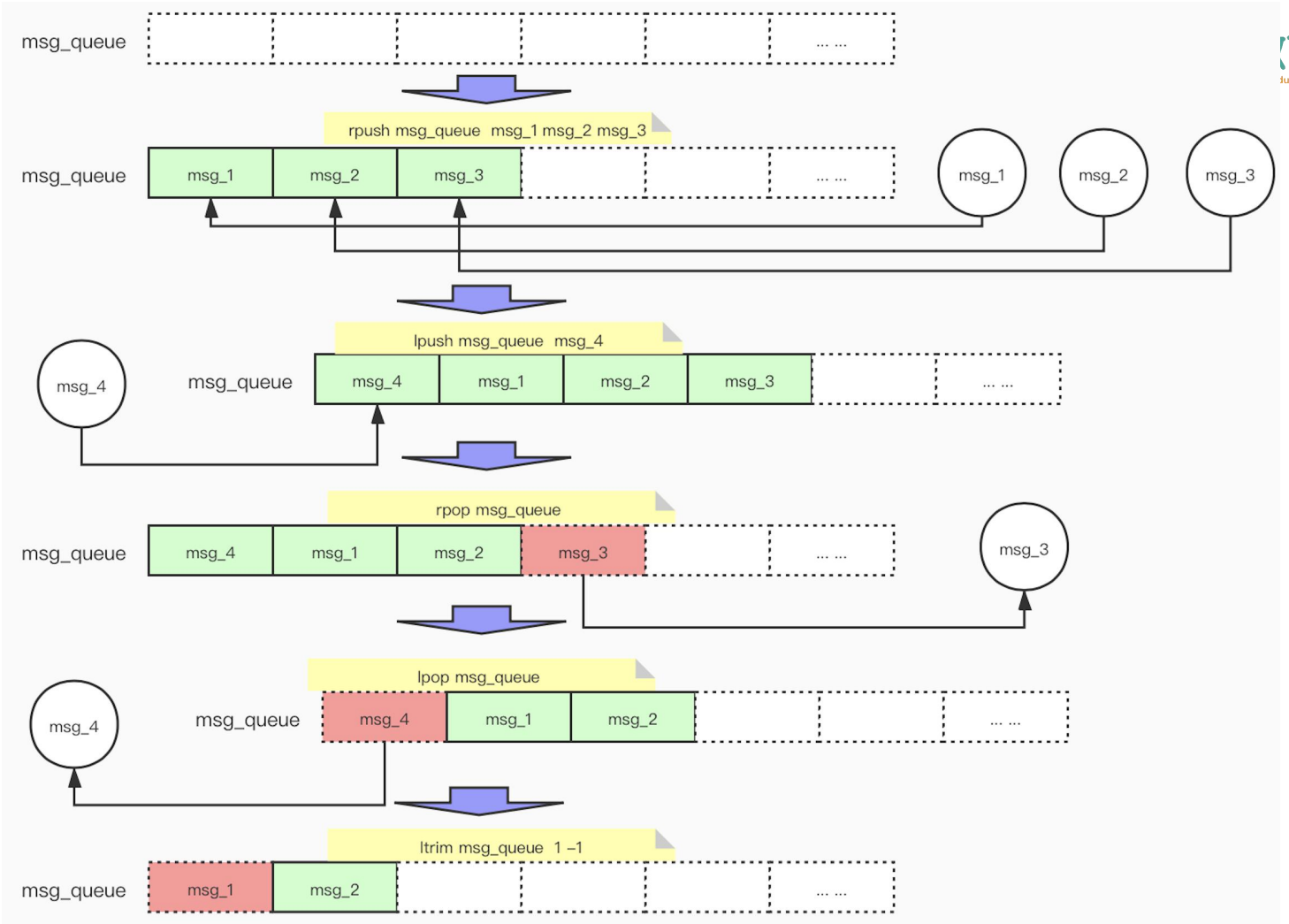
字符串内部实现——embstr编码



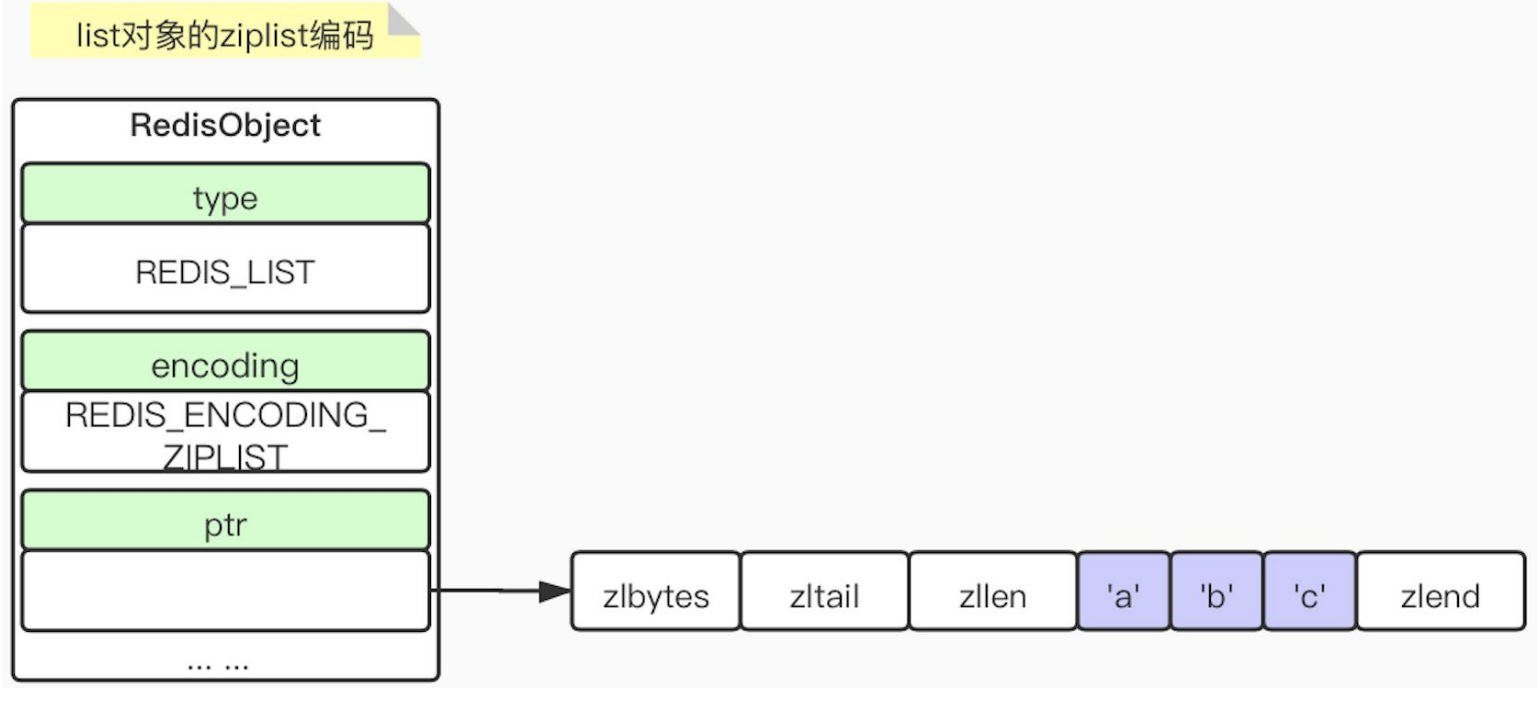
list类型

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。
ltrm key startIndex endIndex	下标同上。

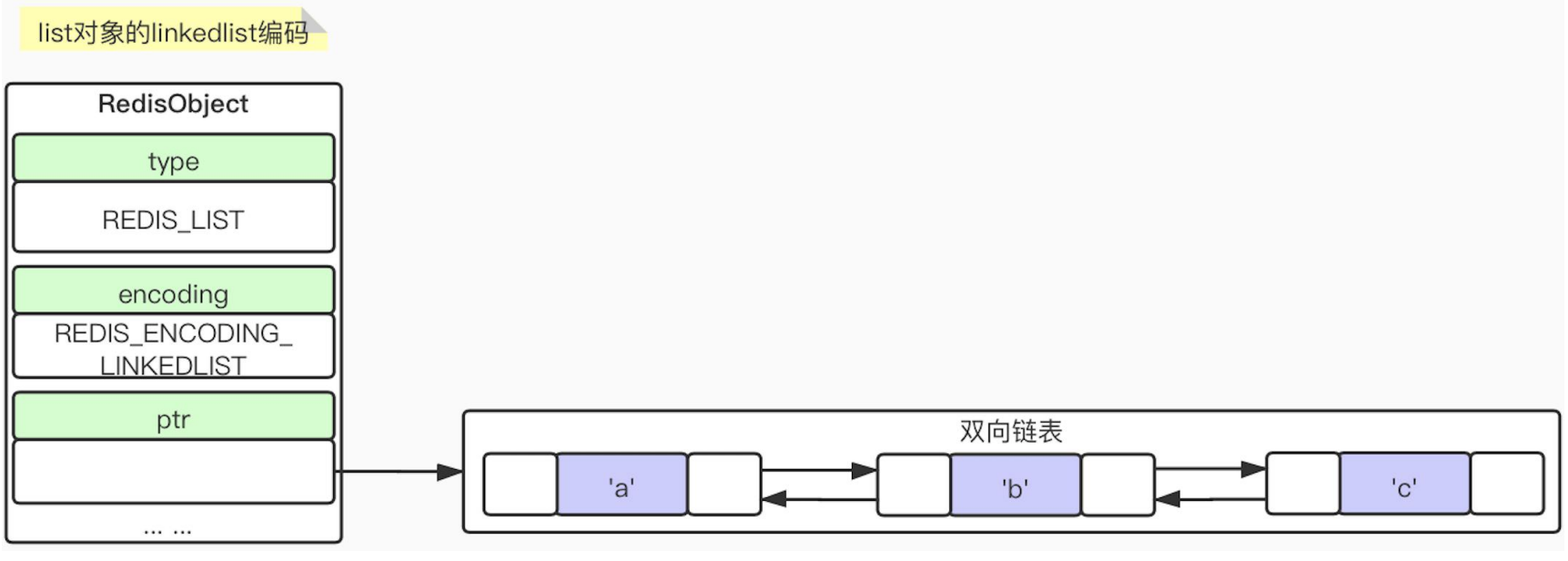
列表操作


msg_queue
... ...
rpush msg_queue msg_1 msg_2 msg_3
msg_queue
msg_1
msg_2
msg_3
... ...
msg_1
msg_2
msg_3
lpush msg_queue msg_4
msg_4
msg_queue
msg_4
msg_1
msg_2
msg_3
... ...
rpop msg_queue
msg_queue
msg_4
msg_1
msg_2
msg_3
... ...
msg_3
lpop msg_queue
msg_4
msg_queue
msg_4
msg_1
msg_2
... ...
ltrim msg_queue 1 -1
msg_queue
msg_1
msg_2
... ...

列表的内部实现——ziplist编码



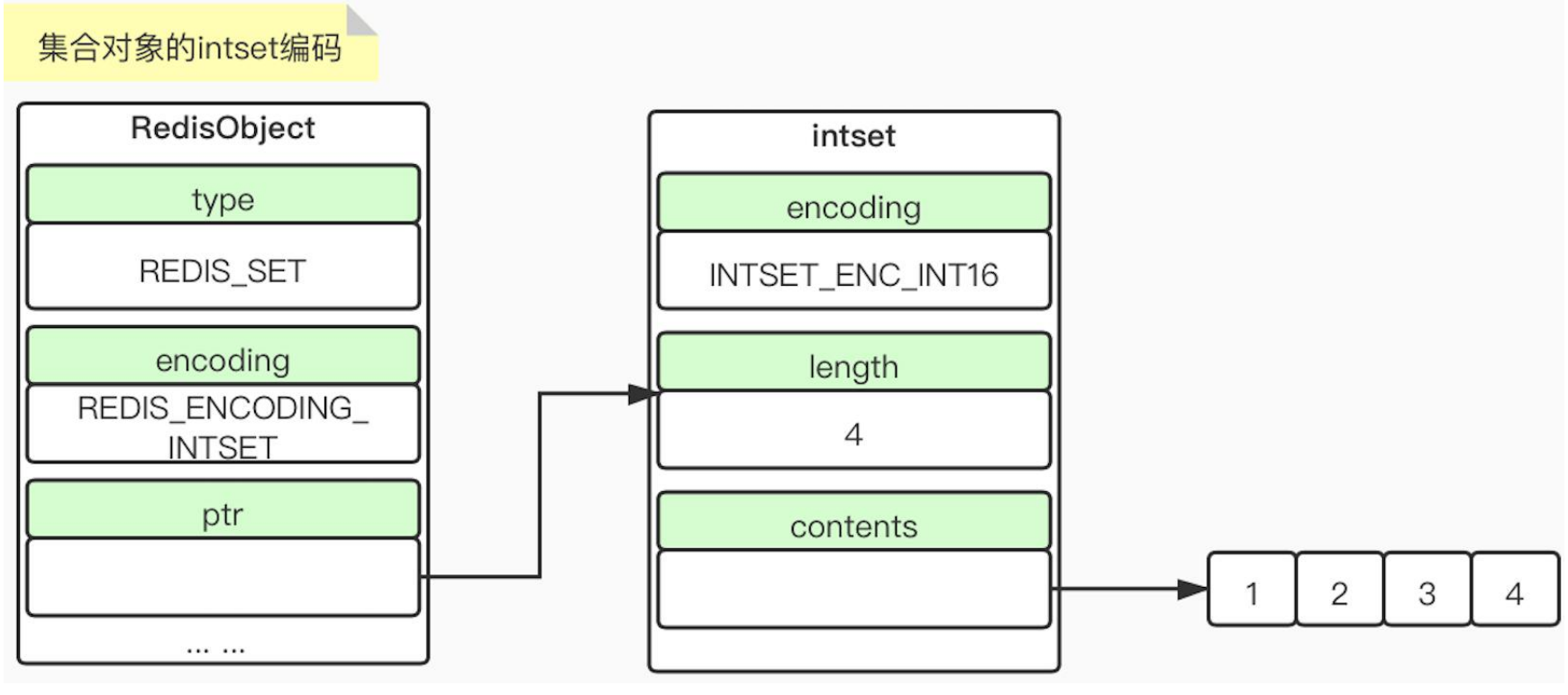
列表的内部实现——linkedlist编码



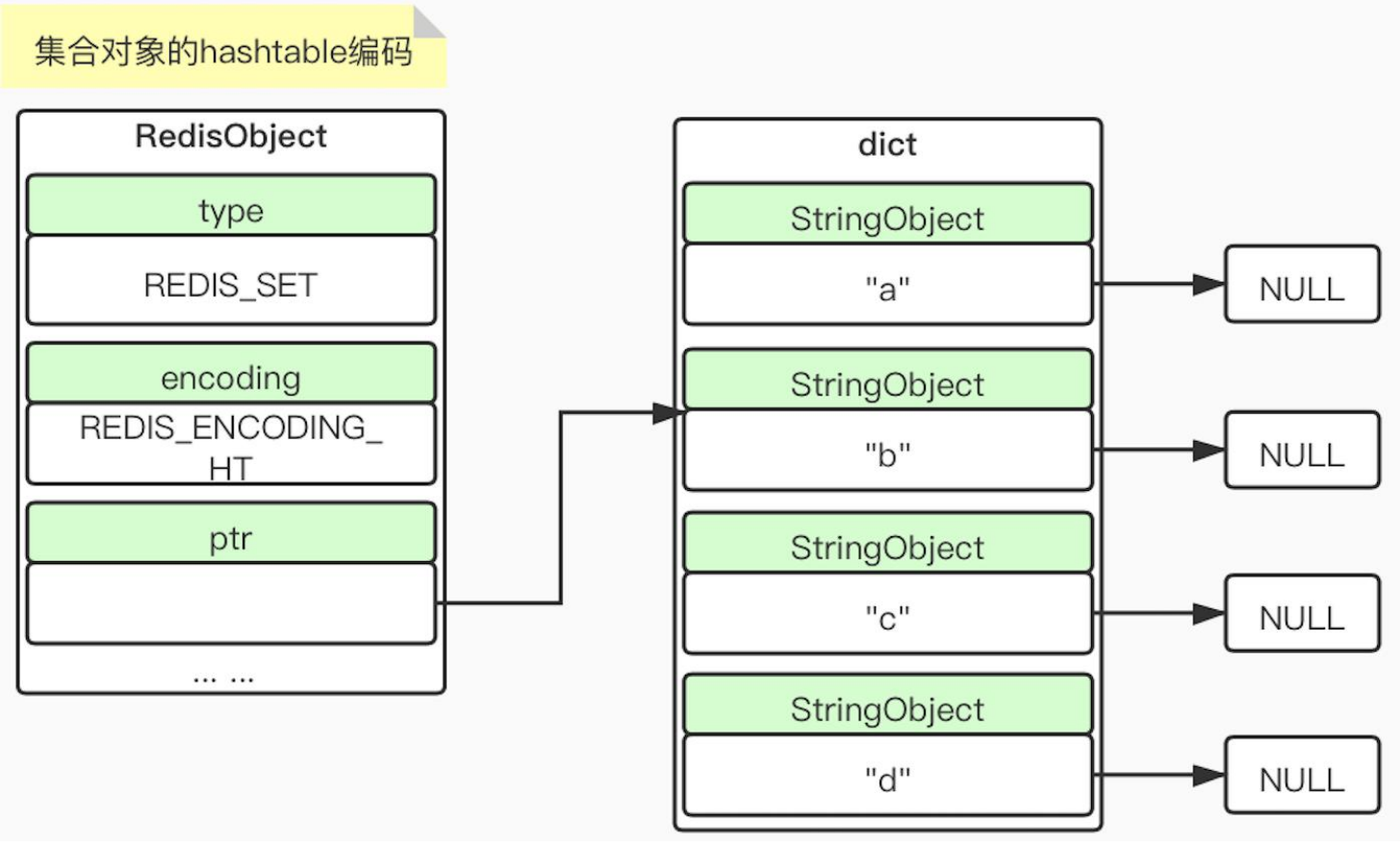
set类型

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



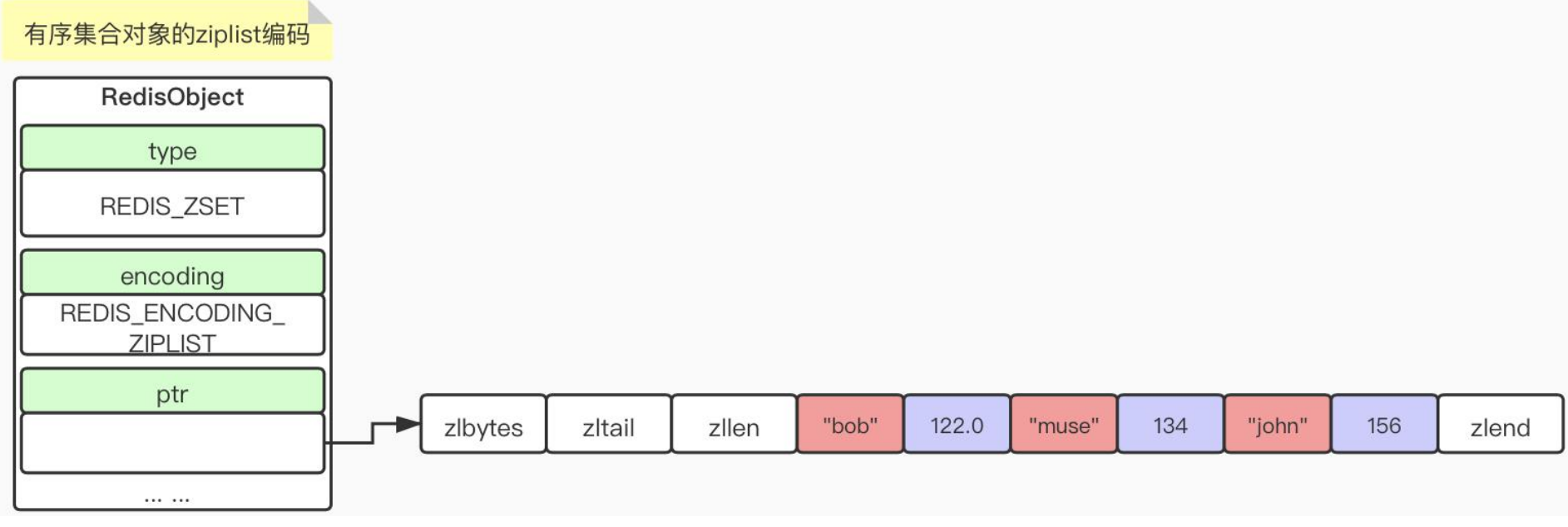
集合类型内部实现——hashtable编码



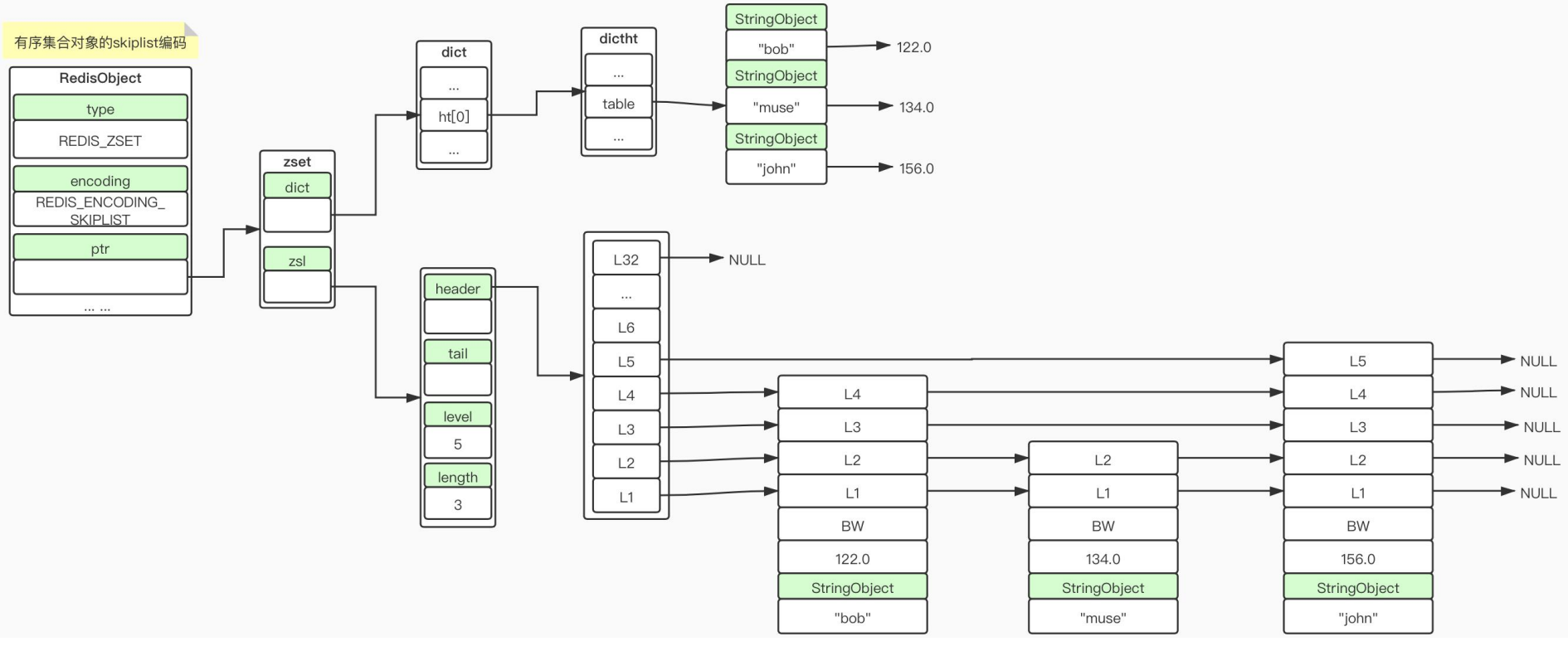
zset类型

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

有序集合类型内部实现——ziplist编码



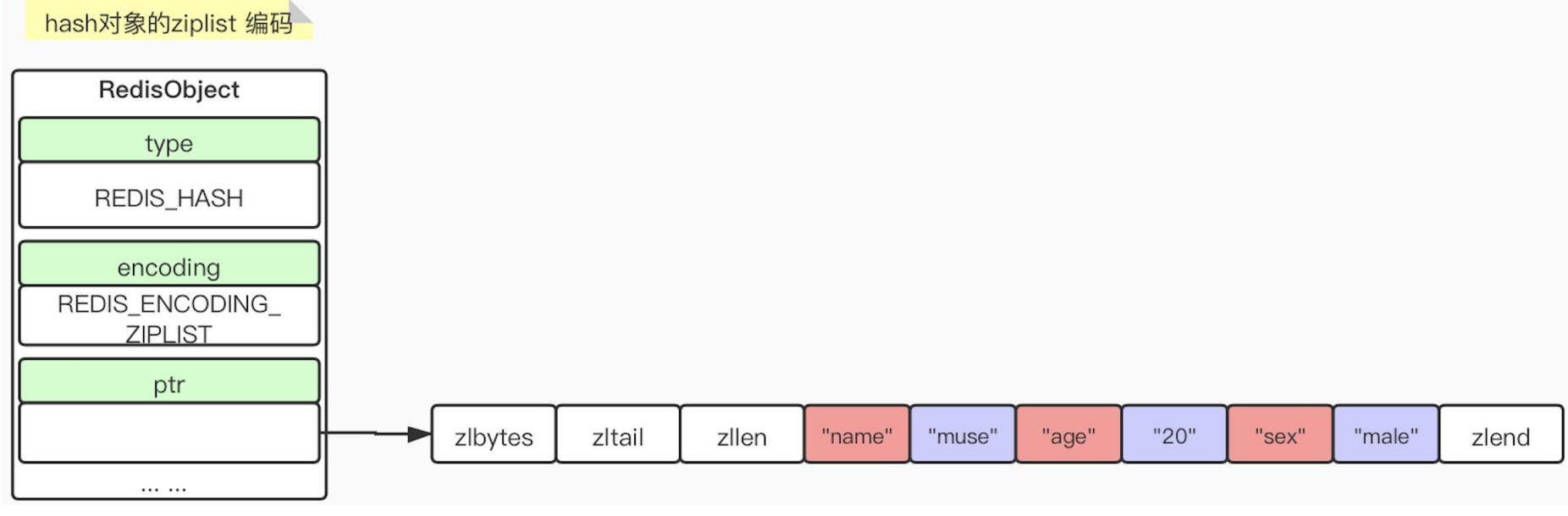
有序集合类型内部实现——skiplist编码



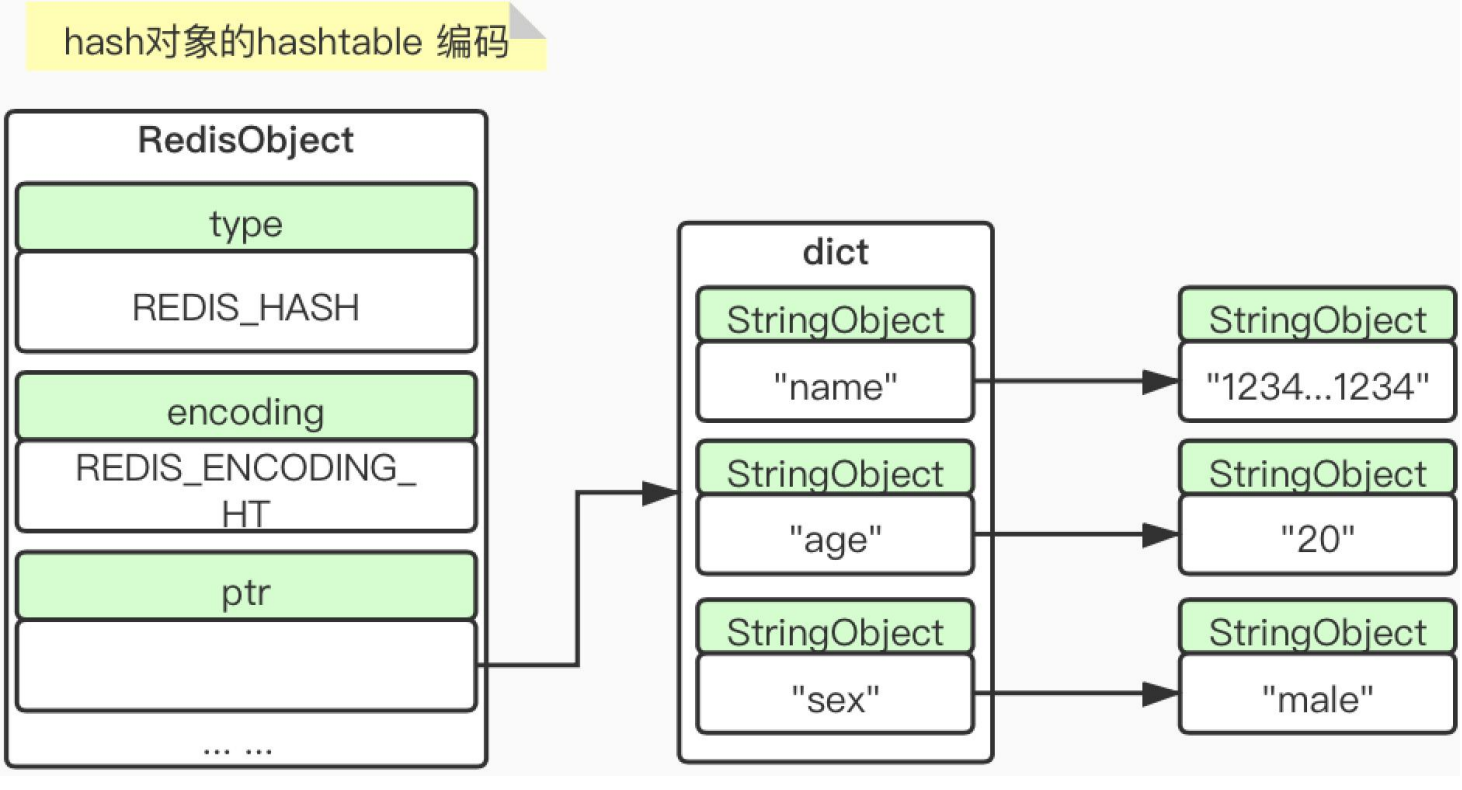
hash类型

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

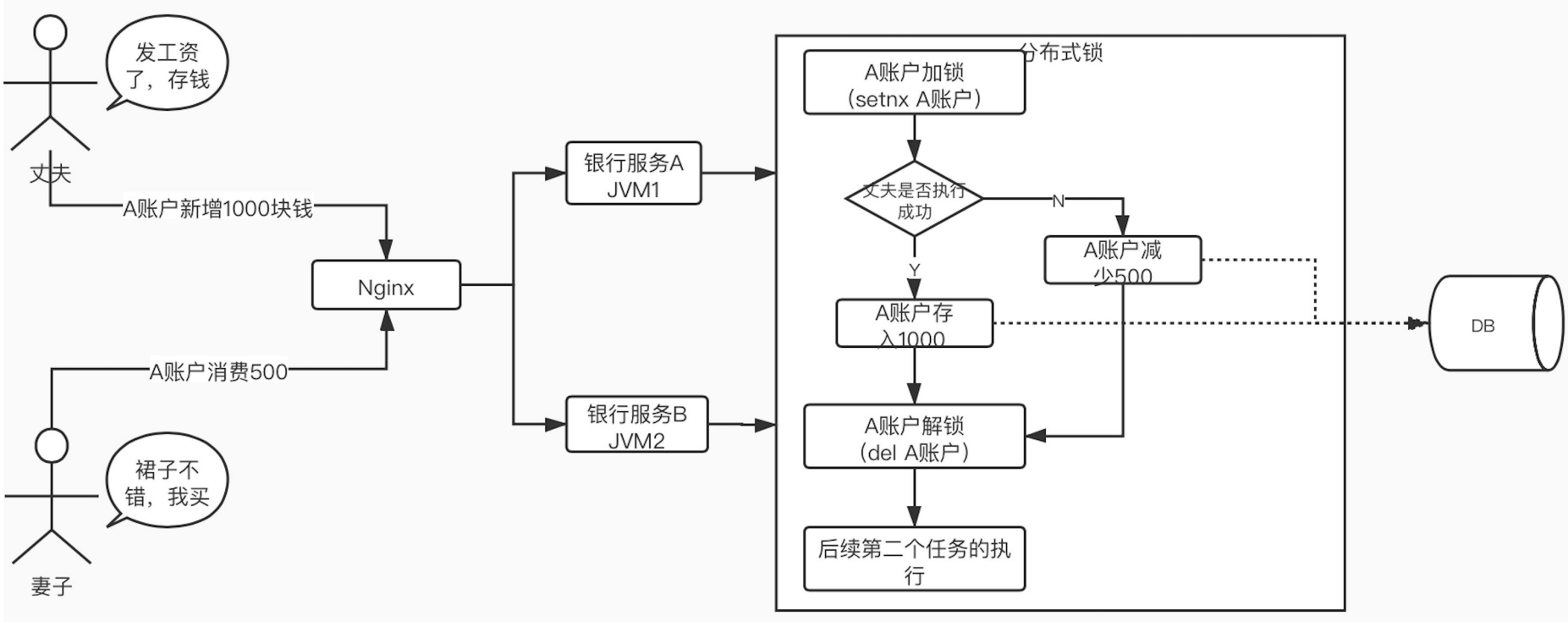


哈希类型内部实现——hashtable编码

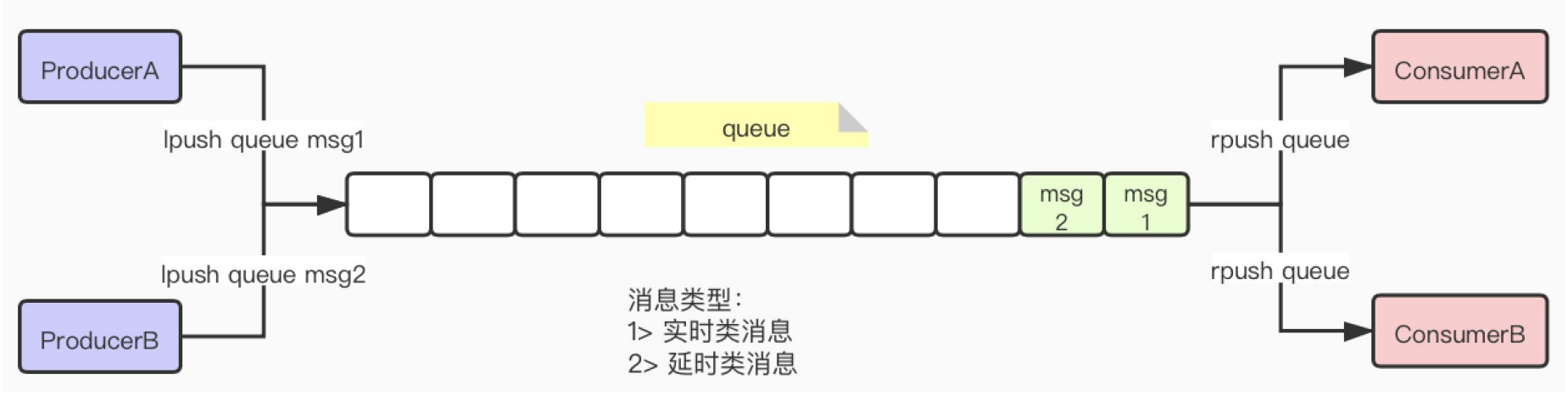


Redis应用

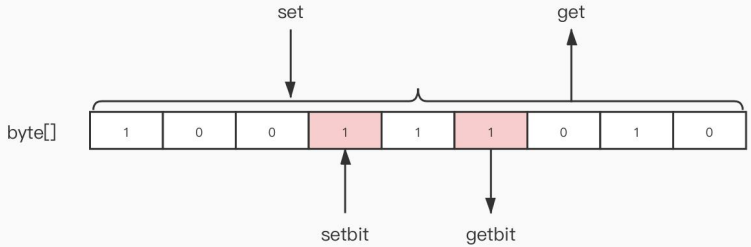
分布式锁


发工资了，存钱
丈夫
A账户新增1000块钱
A账户消费500
裙子不错，我买
妻子
Nginx
银行服务A JVM1
银行服务B JVM2
分布式锁
A账户加锁（setnx A账户）
丈夫是否执行成功
N
Y
A账户减少500
A账户存入1000
A账户解锁（del A账户）
后续第二个任务的执行
DB

消息队列


ProducerA
lpush queue msg1
lpush queue msg2
ProducerB
queue
msg 2
msg 1
rpush queue
rpush queue
ConsumerA
ConsumerB
消息类型:
1> 实时类消息
2> 延时类消息

位图



u字符的ascii码

index	7	6	5	4	3	2	1	0
value	0	1	1	1	0	1	0	1

e字符的ascii码

index	7	6	5	4	3	2	1	0
value	0	1	1	0	0	1	0	1

m字符的ascii码

index	7	6	5	4	3	2	1	0
value	0	1	1	0	1	1	0	1

s字符的ascii码

index	7	6	5	4	3	2	1	0
value	0	1	1	1	0	0	1	1

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
value	0	1	1	0	1	1	0	1	0	1	1	1	0	1	0	1	0	1	1	1	0	0	1	1	0	1	1	0	0	1	0	1

```
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

m字符的ascii码

index	0	1	2	3	4	5	6	7
value	0	1	1	0	1	1	0	1

0	1	1	0	1

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

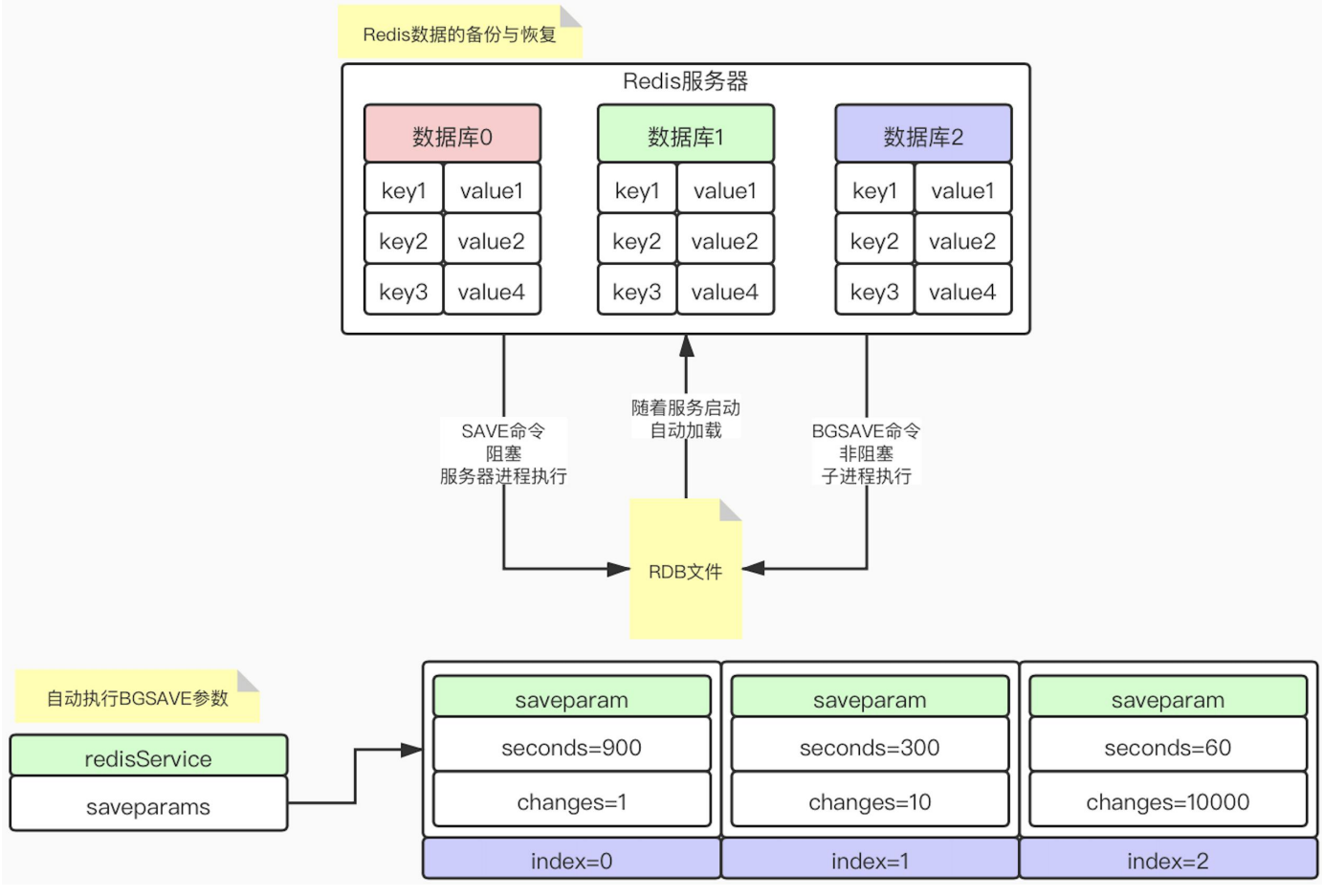
Redis的持久化

RDB相关参数

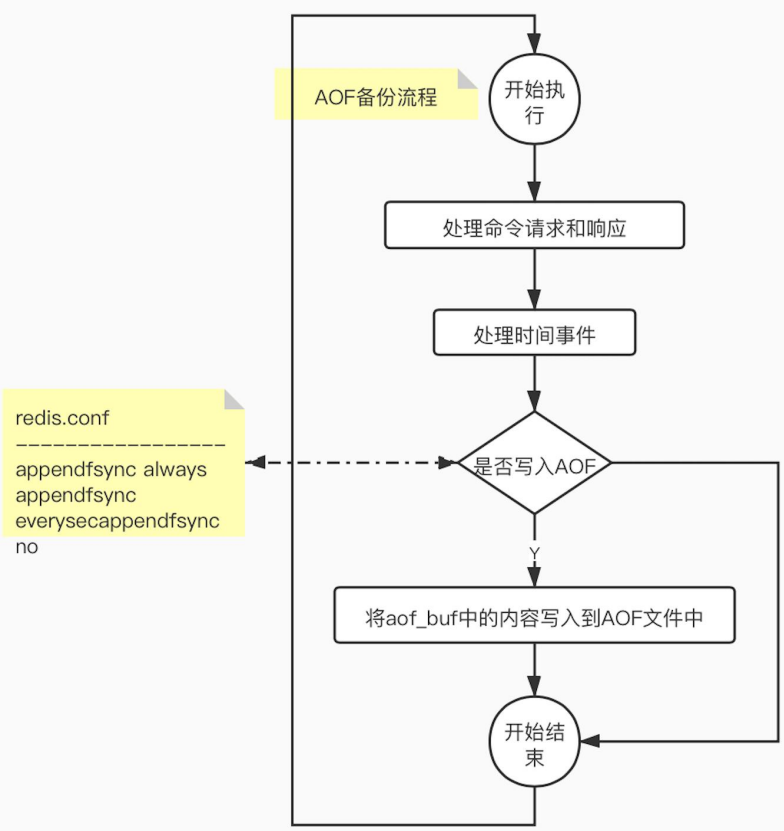
```
redis.h
/*-----------------------------------------------------------------------------
 * Global server state
 *----------------------------------------------------------------------------*/

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 */
  ...
}
```

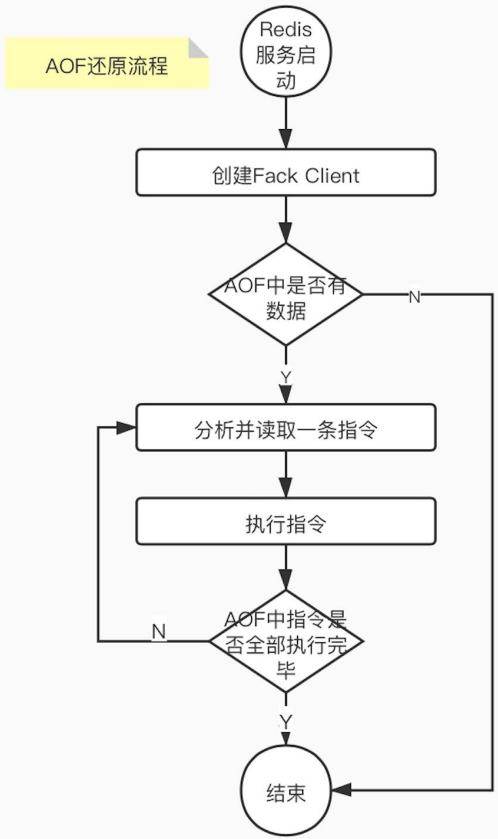
```
redis.h
struct saveparam {
    // 执行的秒数
    time_t seconds;
    // 修改的次数
    int changes;
};
```

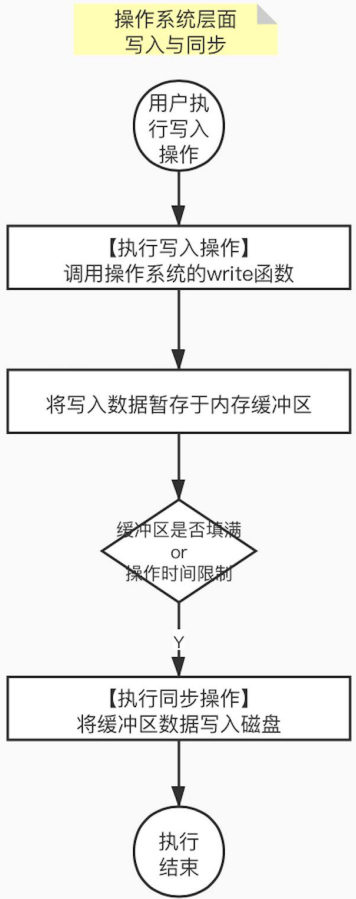
RDB


Redis数据的备份与恢复
Redis服务器
数据库0
key1
value1
key2
value2
key3
value4
数据库1
key1
value1
key2
value2
key3
value4
数据库2
key1
value1
key2
value2
key3
value4
SAVE命令
阻塞
服务器进程执行
随着服务启动
自动加载
BGSAVE命令
非阻塞
子进程执行
RDB文件
自动执行BGSAVE参数
redisService
saveparams
saveparam
seconds=900
changes=1
index=0
saveparam
seconds=300
changes=10
index=1
saveparam
seconds=60
changes=10000
index=2

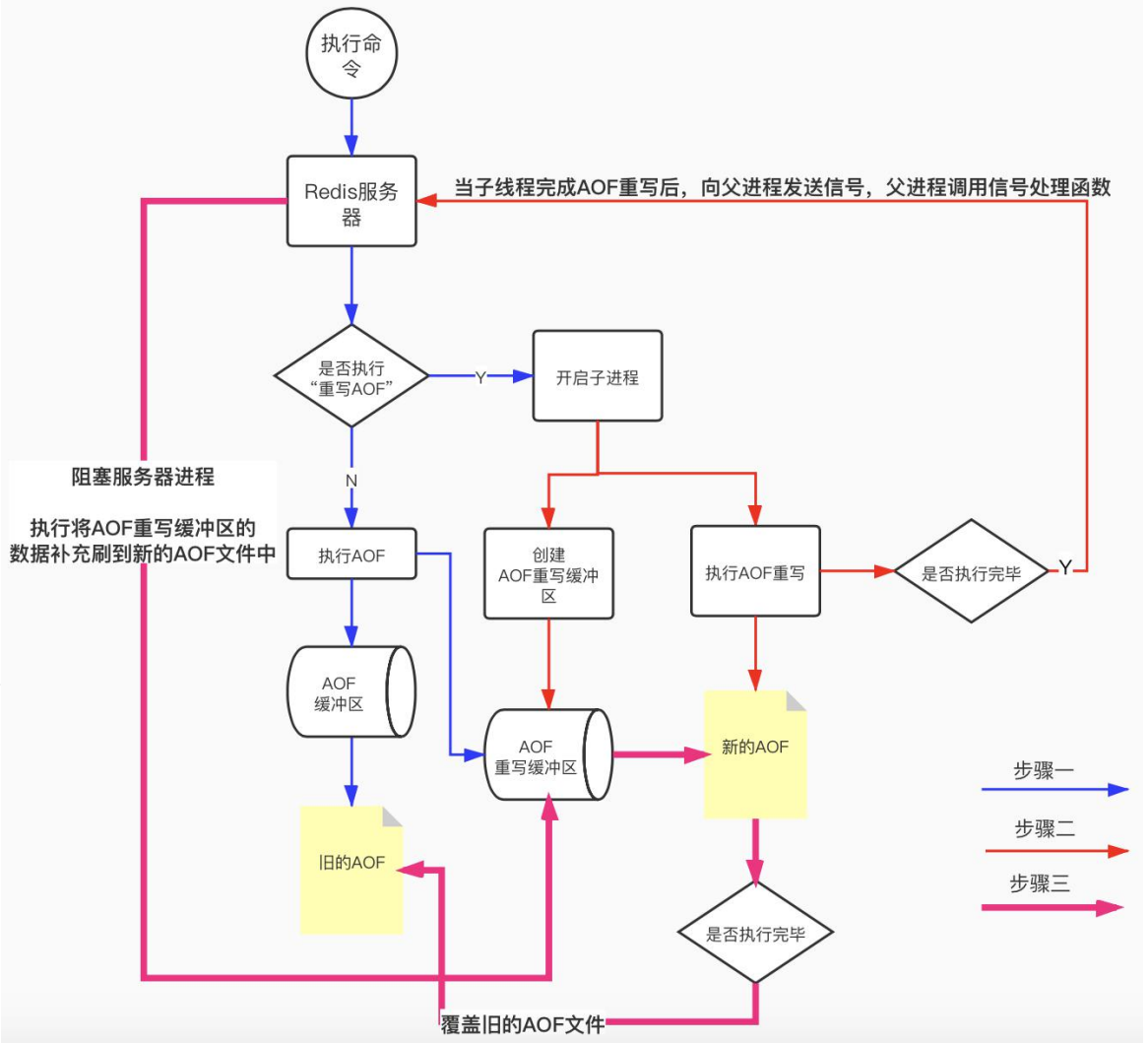
AOF


AOF备份流程
开始执行
处理命令请求和响应
处理时间事件
是否写入AOF
Y
将aof_buf中的内容写入到AOF文件中
开始结束
redis.conf
appendfsync always
appendfsync everysecappendfsync
no


AOF还原流程
Redis服务启动
创建Fack Client
AOF中是否有数据
N
Y
分析并读取一条指令
执行指令
AOF中指令是否全部执行完毕
N
Y
结束


操作系统层面
写入与同步
用户执行写入操作
【执行写入操作】
调用操作系统的write函数
将写入数据暂存于内存缓冲区
缓冲区是否填满
or
操作时间限制
Y
【执行同步操作】
将缓冲区数据写入磁盘
执行结束

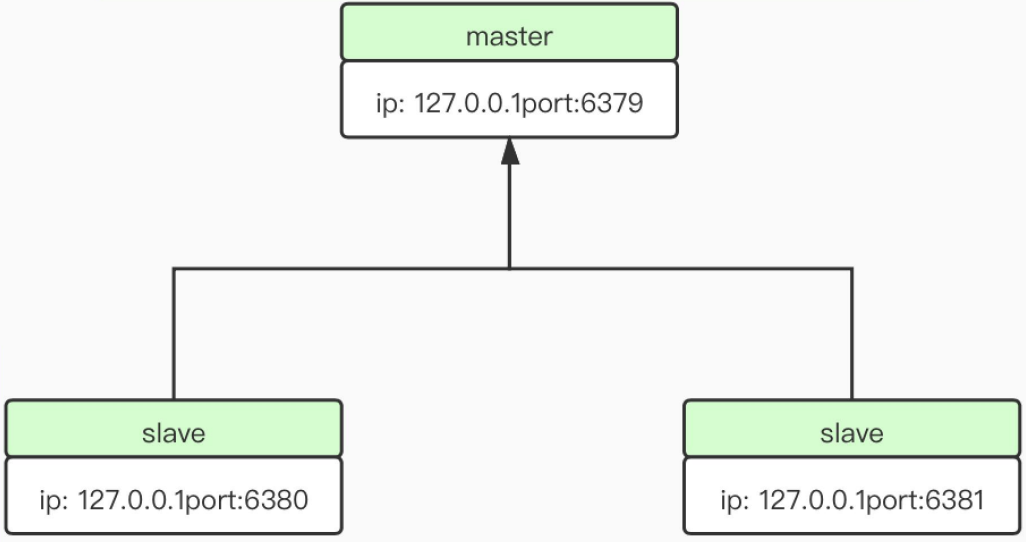
AOF重写


执行命令
Redis服务器
当子线程完成AOF重写后，向父进程发送信号，父进程调用信号处理函数
是否执行“重写AOF”
Y
开启子进程
N
阻塞服务器进程
执行将AOF重写缓冲区的数据补充刷到新的AOF文件中
执行AOF
创建AOF重写缓冲区
执行AOF重写
是否执行完毕
Y
AOF缓冲区
AOF重写缓冲区
新的AOF
旧的AOF
是否执行完毕
覆盖旧的AOF文件
步骤一
步骤二
步骤三

Redis集群搭建

集群搭建

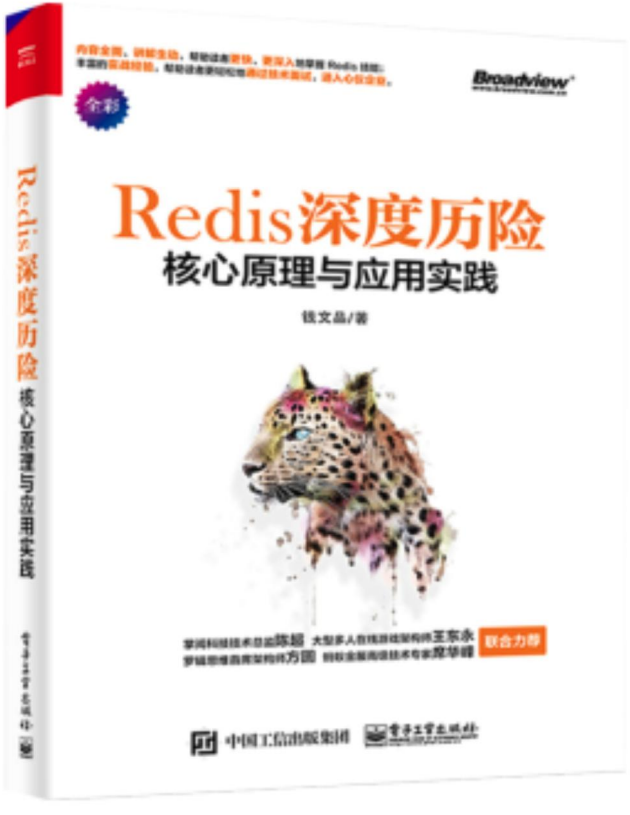
```
port 6379
bind 127.0.0.1
requirepass "myredis"
daemonize yes
logfile "6379.log"
dbfilename "dump-6379.rdb"
dir "/opt/soft/redis/data"
#如若master设置了认证密码，那么所有redis数据节点都配置上masterauth属性
masterauth "myredis"
```

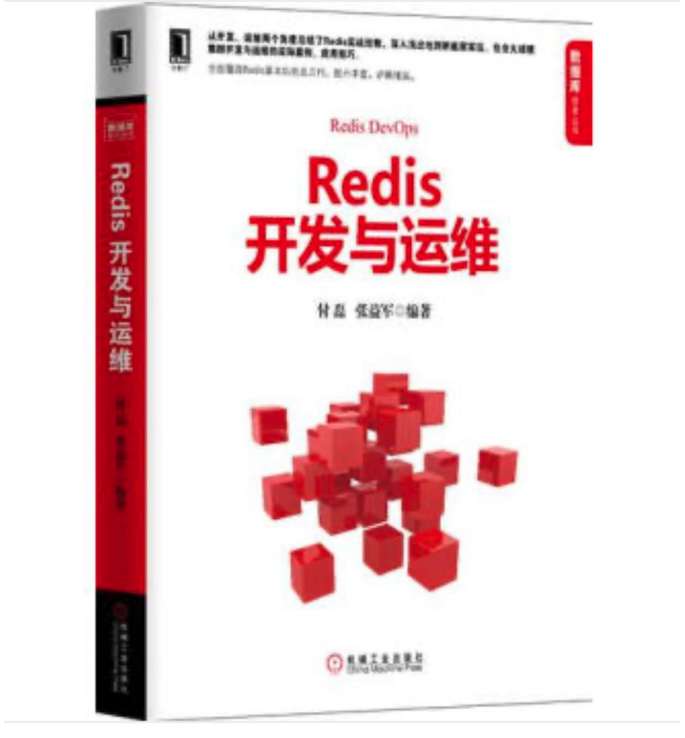



```
port 6380bind
127.0.0.1requirepass
"myredis"daemonize yeslogfile
"6380.log"dbfilename "dump-
6380.rdb"dir
"/opt/soft/redis/data"
masterauth "myredis"slaveof
127.0.0.1 6379
```

```
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


Broadview
全彩
Redis深度历险
核心原理与应用实践
钱文品/著
联合力荐
中国工信出版集团


Redis DevOps
Redis
开发与运维
付磊 张益军◎编著
机械工业出版社