# Redis 文档说明

目录

[Redis 文档说明 1](#_Toc28798)

[1.Redis简介: 2](#_Toc4273)

[2. 网站信息 3](#_Toc19506)

[3. Redis安装 3](#_Toc6286)

[3.1 Windows 3](#_Toc32078)

[3.2 Liunx 3](#_Toc24131)

[4. Redis 数据结构 3](#_Toc1973)

[4.1 动态字符串SDS P9 3](#_Toc30883)

[4.2 链表list P20 5](#_Toc10110)

[4.3 字典dict P24 6](#_Toc31148)

[4.4 跳跃表zplist P39 8](#_Toc31696)

[4.5 整数集合intset P47 9](#_Toc3210)

[4.6 压缩链表ziplist P53 10](#_Toc20830)

[5. Redis 指令 12](#_Toc10376)

[5.1 Redis对象 P61 12](#_Toc20997)

[5.2 键Key 13](#_Toc10449)

[5.3字符串对象P64 14](#_Toc18969)

[5.4列表对象P68 15](#_Toc22251)

[5.5哈希对象P71 17](#_Toc31077)

[5.6集合对象P75 18](#_Toc9809)

[5.7有序集合对象P77 19](#_Toc21745)

[5.8二进制位数组bit P363 22](#_Toc4504)

[6. 网络Socket 23](#_Toc2042)

[6.1 select 23](#_Toc19971)

[6.2 epoll 24](#_Toc13772)

[6.3 net 25](#_Toc28788)

[7. 附件 29](#_Toc19299)

[7.1参考资料 29](#_Toc14101)

# 1.Redis简介:

REmote DIctionary Server(Redis) 是一个由Salvatore Sanfilippo写的key-value存储系统。

Redis是一个开源的使用ANSI C语言编写、遵守BSD协议、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。

它通常被称为数据结构服务器，因为值（value）可以是 字符串(String), 哈希(Hash), 列表(list), 集合(sets) 和 有序集合(sorted sets)等类型。

# 网站信息

Redis 官网：[https://redis.io/](https://redis.io/" \t "https://www.runoob.com/redis/_blank)

Redis 在线测试：[http://try.redis.io/](http://try.redis.io/" \t "https://www.runoob.com/redis/_blank)

菜鸟教程：<https://www.runoob.com/redis/redis-tutorial.html>

# Redis安装

## Windows

下载地址：[https://github.com/MSOpenTech/redis/releases](https://github.com/MSOpenTech/redis/releases" \t "https://www.runoob.com/redis/_blank)

cmd执行命令：redis-server.exe redis.windows.conf

## Liunx

下载地址：[http://redis.io/download](http://redis.io/download" \t "https://www.runoob.com/redis/_blank)

$ wget http://download.redis.io/releases/redis-2.8.17.tar.gz

$ tar xzf redis-2.8.17.tar.gz

$ cd redis-2.8.17

$ make

$ cd src

$ make install

$ ./redis-server ../redis.conf

# Redis 数据结构

## 动态字符串SDS P9

### 4.1.1 动态字符串结构

struct sdshdr{

//记录buf数组中已经使用字节的数量

//等于SDS所保存的字符串的长度

int len;

//记录buf数组中未使用字节的数量

int free;

//字节数组 用于保存字符串

char buf[];

}

sdsdhr

free 0

len 5

e

R

\0

s

i

d

buf

* free属性的值为0，表示这个sds没有分配未使用空间
* len属性的值为5，表示这个SDS保存了一个五字节长度的字符串
* buf属性是一个char类型数组，数组前五个字节分别保存Redis五个字符

，而最后一个字节则保留了空字符’\0’。

### 4.1.2 动态字符串与C字符串的区别

* 常数复杂度获取字符串长度

和C字符串不同，因为SDS在len属性中记录了SDS本身的长度，所以获取

一个SDS长度的复杂度仅为O(1)

* 杜绝缓冲区溢出

C语言strcat没有分配足够多的内存，容纳src字符串中的所有内容，就会产生缓冲区溢出，而SDS中的sdscat会先扩展空间再做拼接工作

* 减少修改字符串时带来的内存重分配次数

1.空间预分配:当SDS的API对一个SDS进行修改，并且需要对SDS进行空

间扩展的时候，程序不仅会为SDS分配修改所需的空间，还会为SDS分配额外的未使用的空间。

2.惰性空间释放:当SDS的API需要缩短SDS保存的字符串时，程序并不立即使用内存重新分配来收回缩短后多出来的字节，而是使用free属性将这些字节数量记录起来，并等待将来使用。

* 二进制安全:可以保存文本数据，还可以保存任意格式的二进制数据
* 一个String类型的value最大可以存储512M

## 链表list P20

### 4.2.1链表结构

typedef struct listNode{

//前置节点

strcuct listNode \*prev;

//后置节点

strcuct listNode \*next;

//节点的值

void \*value;

}listNode;

typedef struct list{

//表头节点

listNode \*head;

//表尾节点

listNode \*tail;

//链表所包含的节点数量

unsigned long len;

//节点值复制函数

void \*(\*dup)(void \*ptr)

//节点值释放函数

void \*(\*free)(void \*ptr)

//节点值对比函数

int \*(\*match)(void \*ptr, void \*key)

}list;

NULL

listNode

listNode

listNode

list

head

value

NULL

value

value

tail

len 3

dup

free

match

### 4.2.2链表list特性总结

* 双端:链表节点带有prev和next指针，获取某个节点的前置节点和后置节点的复杂度都是O(1)。
* 无环:表头节点的prev指针和表尾节点的next指针都是指向NULL，对链表的访问以NULL为终点。
* 带表头指针和表尾指针:通过list结构的head和tail指针，获取链表的头和尾节点复杂度都是是O(1)。
* 带链表长度计数器:程序使用list结构的len属性对list特有的链表节点进行计数，获取链表中节点数量复杂度为是O(1)。
* 多态:链表节点使用void\*指针来保存节点值，并且通过list结构的dup、free、match三个属性为节点值设置类型特定函数，所以链表可以用于保存各种不同类型的值。
* list的元素个数最多为2^32-1个，也就是4294967295个

## 字典dict P24

### 4.3.1字典结构

哈希表:

typedef struct dictht{

//哈希表数组 指向dictEntry结构体指针

dictEntry \*\*table;

//哈希表大小

unsigned long size;

//哈希表大小掩码 用于计算索引值 总是等于size-1

unsigned long sizemask;

//该哈希表已有的节点数量

unsigned long used;

}dictht;

哈希表节点:

typedef struct dictEntry{

//键

void \*key;

//值

union{

void \*val;

uint64\_t u64;

int64\_t s64;

}v;

//指向下个哈希表节点 形成链表

struct dictEntry \*next;

}dictEntry;

字典:

typedef struct dict{

//类型特定函数

dictType \*type;

//私有数据

void \*privdata;

//哈希表

dictht ht[2];

//rehash 索引 当rehash不再进行时 值为-1

int rehashidx;

}

dictEntry\*[4]

dictht

0->NULL

table

size 4

dict

1->NULL

type

2->NULL

sizemask 3

used 1

3

privadata

ht[2]

NULL11

dictEntry

dictht

rehashidx

table

k1

v1

size 0

sizemask 0

used 0

typedef strcut dictType{

unsigned int (\*hashFunction)(const void \*key);

void \*(\*keyDup)(void \*privdata, const void \*key);

void \*(\*valDup)(void \*privdata, const void \*obj);

Int (\*keyCompare)(void \*privdata,const void \*key1,const void \*key2);

int (\*keyDestructor)(void \*privdata, const void \*key);

int (\*valDestructor)(void \*privdata, const void \*obj);

}dictType;

### 4.3.2字典dict总结

* 哈希算法:当要将一个新的键值对添加到字典里面，程序需要先根据键值对的键计算出哈希值和索引值，然后根据索引值，将包含新键值对的哈希表节点放到哈希表数组的指定索引上面

hash=dict->type->hashFunction(key)

Index=hash&dict->ht[x].sizemask;

* 键冲突及其解决:当有2个或2个以上数量的键被分配到了一个索引上面时，我们称这些键发生了冲突；Redis使用链地址法来解决冲突，每个哈希表节点都有一个next指针，多个哈希表节点可以用next指针构成一个单向链表，被分配到同一个索引上的多个节点可以用这个单向链表连接起来
* rehash:随着操作的不断执行，哈希表保存的键值对会逐渐的增多或者减少，为了让哈希表的负载因子(load factor)维持在一个合理的范围之内，当哈希表保存的键值对数量太多或则太少时，需要对哈希表的大小进行相应的扩展或则收缩。
* 渐进式rehash:扩展或则收缩哈希表需要将ht[0]里面的所有键值对rehash到ht[1]里面，但是这个rehash动作不是一次性、集中式的完成的，而是分成多次，渐进式完成的，因为哈希表里保存的键值对数量百万 千万对时，那么一次性将这些键值对rehash到ht[1]的话，庞大的计算量可能导致服务器在一段时间内停止服务；在渐进式rehash期间，字典的删除、查找、更新等操作会在2个哈希表ht[0]和ht[1]上进行,新添加的键值对一律保存在ht[1]里面，直到rehash结束才会全部操作到ht[0]上进行；
* 键值对个数最多为2^32-1个，也就是4294967295个

## 跳跃表zplist P39

### 4.4.1跳跃表节点结构

typedef struct zskiplistNode{

//后退指针

strcut zskiplistNode \*backward;

//分值

double score;

//成员对象

robj \*obj;

//层

struct zskiplistLevel{

//前进指针

strcut zskiplistNode \*farward;

//跨度

unsigned int span；

}level[]

}

* 层
* 跨度
* 前进指针
* 后退指针
* 分值和成员

### 4.4.2跳跃表结构

typedef struct zskiplist{

//表头节点和表尾节点

struct skiplistNode \*header, \*tail;

//表节点的数量

unsigned long length;

//表中层数最大的节点层数

int level;

}zskiplist;

## 整数集合intset P47

### 4.5.1 整数集合结构

typedef struct intset{

//编码方式

uint32\_t encoding;

//集合包含元素的数量

uint32\_t length;

//保存元素的数组

int8\_t contents[];

}intset;

intset

encoding

INTSET\_ENC\_INT16

length 5

-6370

-5

14632

233

contents

18

* length:记录整数集合包含的元素数量，也是contents数组的长度。
* contents:整数集合的每个元素都是contents数组的一个数组项(item)，各个项在数组中按值的大小从小到大有序排列，并且在数组中不包含任何重复相。
* encoding: INTSET\_ENC\_INT16 INTSET\_ENC\_INT32 INTSET\_ENC\_INT64

如果encoding属性值为INTSET\_ENC\_INT16，那么contents就是一个int16\_t类型的数组，数组里的每个项都是int16\_t类型的整数值(-32768 ~ 32767)。

如果encoding属性值为INTSET\_ENC\_INT32，那么contents就是一个int32\_t类型的数组，数组里的每个项都是int32\_t类型的整数值(-2147483648 ~ 2147483647)。

如果encoding属性值为INTSET\_ENC\_INT64，那么contents就是一个int64\_t类型的数组，数组里的每个项都是int64\_t类型的整数值(-9223372036854775808 ~ 9223372036854775807)。

### 4.5.2整数集合intset总结

* 升级:每当我们要将一个新元素添加到整数集合里面时，并且新元素的类型比整数集合现有所有元素类型都要长，整数集合需要先进行升级，然后才能将新元素添加到整数集合里面；如INTSET\_ENC\_INT16->INTSET\_ENC\_INT32
* 升级优势:提升整数集合的灵活性和尽可能的节约内存
* 降级:整数集合不支持降级操作；一旦对数组进行了升级操作，编码就会一直保持升级后的状态。
* 元素个数最多为2^32-1个，也就是4294967295个

## 压缩链表ziplist P53

### 4.6.1 压缩链表结构

entry2

entry1

zllen

zltail

zlend

entryN

zlbytes

|  |  |  |  |
| --- | --- | --- | --- |
| **属性** | **类型** | **长度** | **用途** |
| zlbytes | uint32\_t | 4字节 | 记录整个压缩列表占用的内存字节数；在对压缩列表计算zlend的位置时使用 |
| zltail | uint32\_t | 4字节 | 记录压缩列表表尾节点距离压缩列表的起始地址多少字节；通过这个偏移量，程序无须遍历整个压缩列表就可以确定表尾节点的地址 |
| zllen | uint16\_t | 2字节 | 记录了压缩列表包含的节点数量：当这个属性的值小于UINT16\_MAX(65535)时，这个属性的值就是压缩列表包含的节点数量；当这个值等于UINT16\_MAX时，节点的真实数量需要遍历整个压缩列表计算得出 |
| entryX | 列表节点 | 不定 | 压缩列表包含的各个节点，节点的长度有节点保存的内容决定 |
| zlend | uint8\_t | 1字节 | 特殊值0xFF，用于标记压缩列表的末端 |

### 4.6.2 压缩链表节点entry结构

content

encoding

previous\_entry\_length

* previous\_entry\_length:记录了压缩列表中前一个节点的长度。长度可以是1字节或则5字节；如果前一节点的长度小于254字节，那么previous\_entry\_length属性长度为1字节：前一节点的长度就保存在这一字节里面；如果前一节点的长度大于等于254字节，那么previous\_entry\_length属性长度为5字节：其中属性的第一个字节会被设置为OxFE(254)，而之后的四字节则用于保存前一节点的长度。
* encoding：记录了节点的content属性所保存数据的类型及其长度

|  |  |  |
| --- | --- | --- |
| **编码** | **编码长度** | **content属性保存的值** |
| **字节数组编码** | | |
| 00bbbbbb | 1字节 | 长度小于等于63字节的字节数组(2^6-1) |
| 01bbbbbb xxxxxxxx | 2字节 | 长度小于等于16383字节的字节数组(2^14-1) |
| 10\_\_\_\_\_\_ aaaaaaaa  bbbbbbbb cccccccc  dddddddd | 5字节 | 长度小于等于4294967295字节的字节数组(2^32-1) |
| **整数编码** | | |
| 11000000 | 1字节 | int16\_t类型的整数 |
| 11010000 | 1字节 | int32\_t类型的整数 |
| 11100000 | 1字节 | int64\_t类型的整数 |
| 11110000 | 1字节 | 24位有符号整数 |
| 11111110 | 1字节 | 8位有符号整数 |
| 1111xxxx | 1字节 | 使用这一编码的节点没有相应的content属性，因为编码本身的xxxx四个位已保存一个介于0到12之间的值，所以它无须乎content属性 |

* content：负责保存节点的值，节点值可以是一个字节数组或则整数，值的类型和长度由节点的encoding属性决定

### 4.6.3 压缩链表总结

* 连锁更新：如果前一个节点的长度小于254字节，那么length属性需要用1字节长的空间来保存这个长度值；如果前一节点的长度大于等于254字节，那么length属性需要用到5字节长的空间来保存这个长度值；

当在一个压缩列表中，有多个连续的、长度介于250字节到253字节之间的 节点e1到eN，这时我们将一个长度大于等于254字节的新节点new设置为压缩列表的表头节点，那么new将成为前置节点，而针对后面的e1到eN需要连续更新前节点长度由1->5，引发连锁更新，除此之外删除节点也可能引发连锁更新。

# Redis 指令

## Redis对象 P61

### 5.1.1 对象结构及其说明

Redis使用对象来表示数据库中的键和值，每次当我们在Redis的数据库中新创建一个键值对时，我们至少创建2个对象，一个对象用作键值对的键(键对象)，另外一个对象用作键值对的值(值对象)

typedef strcut redisObject{

//类型

unsigned type:4;

//编码

unsigned encoding:4;

//指向底层实现数据结构的指针

void \*ptr

//...

}robj;

* 类型:指令 type key

|  |  |  |
| --- | --- | --- |
| **对象** | **对象type属性的值** | **type命令的输出** |
| 字符串对象 | REDIS\_STRING | “string” |
| 列表对象 | REDIS\_LIST | “list” |
| 哈希对象 | REDIS\_HASH | “hash” |
| 集合对象 | REDIS\_SET | “set” |
| 有序集合对象 | REDIS\_ZSET | “zset” |

* 编码和底层实现：指令 object encoding key

|  |  |  |
| --- | --- | --- |
| **对象所使用的底层数据结构** | **编码常量** | **object encoding命令输出** |
| 整数 | REDIS\_ENCODING\_INT | “int” |
| embstr简单动态字符串SDS | REDIS\_ENCODING\_EMBSTR | “embstr” |
| 简单动态字符串 | REDIS\_ENCODING\_RAW | “raw” |
| 字典 | REDIS\_ENCODING\_HT | “hashtable” |
| 双端链表 | REDIS\_ENCODING\_LINKEDLIST | “linkedlist” |
| 压缩列表 | REDIS\_ENCODING\_ZIPLIST | “ziplist” |
| 整数集合 | REDIS\_ENCODING\_INTSET | “intset” |
| 跳表和字典 | REDIS\_ENCODING\_SKIPLIST | “skiplist” |

* 类型和编码对象

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **类型** | **编码** | **对象** |
| 1 | REDIS\_STRING | REDIS\_ENCODING\_INT | 使用整数值实现的字符串对象 |
| REDIS\_STRING | REDIS\_ENCODING\_EMBSTR | 使用embstr编码实现的字符串对象 |
| REDIS\_STRING | REDIS\_ENCODING\_RAW | 使用简单动态字符串实现字符串对象 |
| 2 | REDIS\_LIST | REDIS\_ENCODING\_ZIPLIST | 使用压缩列表实现的列表对象 |
| REDIS\_LIST | REDIS\_ENCODING\_LINKEDLIST | 使用双端链表实现的列表对象 |
| REDIS\_LIST | quicklist | 3.2版本之后有quicklist取代 |
| 3 | REDIS\_HASH | REDIS\_ENCODING\_ZIPLIST | 使用压缩列表实现的哈希对象 |
| REDIS\_HASH | REDIS\_ENCODING\_HT | 使用字典实现的哈希对象 |
| 4 | REDIS\_SET | REDIS\_ENCODING\_INTSET | 使用整数集合实现的集合对象 |
| REDIS\_SET | REDIS\_ENCODING\_HT | 使用字典实现的集合对象 |
| 5 | REDIS\_ZSET | REDIS\_ENCODING\_ZIPLIST | 使用压缩列表实现的有序集合对象 |
| REDIS\_ZSET | REDIS\_ENCODING\_SKIPLIST | 使用跳跃表和字典实现的有序集合对象 |

## 5.2 键Key

|  |  |
| --- | --- |
| **命令** | **描述** |
| [DEL key](https://www.runoob.com/redis/keys-del.html) | 使该命令用于在key存在时删除 key |
| [DUMP key](https://www.runoob.com/redis/keys-dump.html) | 序列化给定key，并返回被序列化的值 |
| [EXISTS key](https://www.runoob.com/redis/keys-exists.html) | 检查给定key是否存在 |
| [EXPIRE key](https://www.runoob.com/redis/keys-expire.html) seconds | 为给定key设置过期时间，以秒计 |
| [PEXPIRE key milliseconds](https://www.runoob.com/redis/keys-pexpire.html) | 设置key的过期时间以毫秒计 |
| [EXPIREAT key timestamp](https://www.runoob.com/redis/keys-expireat.html) | EXPIREAT的作用和EXPIRE类似，都用于为key设置过期时间。不同在于EXPIREAT命令接受的时间参数是 UNIX 时间戳(unix timestamp) |
| [PEXPIREAT key milliseconds-timestamp](https://www.runoob.com/redis/keys-pexpireat.html) | 设置key过期时间的时间戳(unix timestamp) 以毫秒计 |
| [KEYS pattern](https://www.runoob.com/redis/keys-keys.html) | 查找所有符合给定模式( pattern)的 key: KEYS runoob\* |
| [MOVE key db](https://www.runoob.com/redis/keys-move.html) | 将当前数据库的key移动到给定的数据库 db 当中 |
| [PERSIST key](https://www.runoob.com/redis/keys-persist.html) | 移除key的过期时间，key将持久保持 |
| [PTTL key](https://www.runoob.com/redis/keys-pttl.html) | 以毫秒为单位返回key的剩余的过期时间 |
| [TTL key](https://www.runoob.com/redis/keys-ttl.html) | 以秒为单位，返回给定key的剩余生存时间(TTL, time to live) |
| [RANDOMKEY](https://www.runoob.com/redis/keys-randomkey.html) | 从当前数据库中随机返回一个 key |
| [RENAME key newkey](https://www.runoob.com/redis/keys-rename.html) | 修改key的名称 |
| [RENAMENX key newkey](https://www.runoob.com/redis/keys-renamenx.html) | 仅当newkey不存在时，将 key 改名为 newkey |
| [TYPE key](https://www.runoob.com/redis/keys-type.html) | 返回key所储存的值的类型 |

## 5.3字符串对象P64

|  |  |  |  |
| --- | --- | --- | --- |
| **命令及描述** | **int编码实现方法** | **embstr编码的实现方法** | **raw编码的实现方法** |
| [SET key value](https://www.runoob.com/redis/strings-set.html) | 使用int编码保存值 | 使用embstr编码保存值 | 使用raw编码保存值 |
| [GET key](https://www.runoob.com/redis/strings-get.html) | 拷贝对象所保存的整数值,将这个拷贝转换成字符串，然后向客户端返回这个字符串值 | 直接向客户端返回字符串 | 直接向客户端返回字符串 |
| [APPEND key value](https://www.runoob.com/redis/strings-append.html) | 将对象转换成raw编码，然后按照raw编码的方式执行此操作 | 将对象转换成raw编码，然后按照raw编码的方式执行此操作 | 调用sdscatlen函数,将给定的字符串追加到现有字符串的末尾 |
| [INCRBYFLOAT key increment](https://www.runoob.com/redis/strings-incrbyfloat.html) | 取出整数数值将其转换成long double类型的浮点数，对这个浮点数进行加法计算,然后将得出的浮点数结果保存 | 取出字符串并尝试转换成long double类型的浮点数,对这个浮点数进行加法计算结果保存起来。如果字符串值不能被转换成浮点数，那么向客户端返回一个错误 | 取出字符串并尝试转换成long double类型的浮点数,对这个浮点数进行加法计算结果保存起来。如果字符串值不能被转换成浮点数，那么向客户端返回一个错误 |
| [INCRBY key increment](https://www.runoob.com/redis/strings-incrby.html) | 对整数值进行加法计算，得出的结果会作为整数被保存起来 | embstr编码不能执行此命令，向客户端返回一个错误 | raw编码不能执行此命令，向客户端返回一个错误 |
| [DECRBY key decrement](https://www.runoob.com/redis/strings-decrby.html) | 对整数值进行减法计算，得出的结果会作为整数被保存起来 | embstr编码不能执行此命令，向客户端返回一个错误 | raw编码不能执行此命令，向客户端返回一个错误 |
| [STRLEN key](https://www.runoob.com/redis/strings-strlen.html) | 拷贝对象所保存的整数值，将这个拷贝转换成字符串，计算并返回这个字符串的长度 | 将对象转换成raw编码，然后按照raw编码的方式执行此命令 | 调用sdslen函数，返回字符串的长度 |
| [SETRANGE key offset value](https://www.runoob.com/redis/strings-setrange.html) | 将对象转换成raw编码，然后按照raw编码的方式执行此命令 | 将对象转换成raw编码，然后按照raw编码的方式执行此命令 | 将字符串特定的索引上的值设置为给定的字符串 |
| [GETRANGE key start end](https://www.runoob.com/redis/strings-getrange.html) | 拷贝对象所保存的整数值，将这个拷贝转换成字符串值，然后取出并返回字符串指定索引上的字符 | 直接取出并返回字符串指定索引上的字符 | 直接取出并返回字符串指定索引上的字符 |
| [SETEX key seconds value](https://www.runoob.com/redis/strings-setex.html)  将值value关联到key，并将 key的过期时间设为seconds (以秒为单位)  [PSETEX key milliseconds value](https://www.runoob.com/redis/strings-psetex.html)  这个命令和SETEX命令相似，但它以毫秒为单位设置key的生存时间  [SETNX key value](https://www.runoob.com/redis/strings-setnx.html)  只有在key不存在时设置key的值 |  |  |  |
| [GETSET key value](https://www.runoob.com/redis/strings-getset.html) 将给定key的值设为value，并返回key的旧值(old value) |  |  |  |
| [SETBIT key offset value](https://www.runoob.com/redis/strings-setbit.html) 对 key 所储存的字符串值，设置或清除指定偏移量上的位(bit)  [GETBIT key offset](https://www.runoob.com/redis/strings-getbit.html) 对 key 所储存的字符串值，获取指定偏移量上的位(bit) |  |  |  |

## 5.4列表对象P68

|  |  |  |
| --- | --- | --- |
| **命令及描述** | **ziplist编码的实现方式** | **linkedlist编码的实现方式** |
| [LPUSH key value1 [value2]](https://www.runoob.com/redis/lists-lpush.html" \t "https://www.runoob.com/redis/_blank)  [LPUSHX key value](https://www.runoob.com/redis/lists-lpushx.html" \t "https://www.runoob.com/redis/_blank)  将一个值插入到已存在的列表头部 | 调用ziplistPush函数，将新元素推入到压缩列表头的表头 | 调用listAddNodeHead函数，将新元素推入到双端链表的表头 |
| [RPUSH key value1 [value2]](https://www.runoob.com/redis/lists-rpush.html" \t "https://www.runoob.com/redis/_blank)  [RPUSHX key value](https://www.runoob.com/redis/lists-rpushx.html" \t "https://www.runoob.com/redis/_blank) 为已存在的列表添加值 | 调用ziplistPush函数，将新元素推入到压缩列表头的表尾 | 调用listAddNodeTail函数，将新元素推入到双端链表的表尾 |
| [LPOP key](https://www.runoob.com/redis/lists-lpop.html" \t "https://www.runoob.com/redis/_blank)  [BLPOP key1 [key2 ] timeout](https://www.runoob.com/redis/lists-blpop.html" \t "https://www.runoob.com/redis/_blank)  移出并获取列表的第一个元素， 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止 | 调用ziplistIndex函数定位压缩列表的头节点，在向用户返回节点保存的元素之后，调用ziplistDelete函数删除表头节点 | 调用listFirst函数定位双端链表的表头节点，在向用户返回节点保存的元素之后，调用listDelNode函数删除表头节点 |
| [RPOP key](https://www.runoob.com/redis/lists-rpop.html" \t "https://www.runoob.com/redis/_blank)  [BRPOP key1 [key2 ] timeout](https://www.runoob.com/redis/lists-brpop.html" \t "https://www.runoob.com/redis/_blank)  移出并获取列表的最后一个元素，如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止 | 调用ziplistIndex函数定位压缩列表的尾节点，在向用户返回节点保存的元素之后，调用ziplistDelete函数删除表尾节点 | 调用listLast函数定位双端链表的表尾节点，在向用户返回节点保存的元素之后，调用listDelNode函数删除表尾节点 |
| [LINDEX key index](https://www.runoob.com/redis/lists-lindex.html" \t "https://www.runoob.com/redis/_blank) | 调用ziplistIndex函数定位压缩列表的指定节点，然后返回节点所保存的元素 | 调用listIndex函数定位双端链表的指定节点，然后返回节点所保存的元素 |
| [LLEN key](https://www.runoob.com/redis/lists-llen.html" \t "https://www.runoob.com/redis/_blank) | 调用ziplistLen函数返回压缩列表的长度 | 调用listLength函数返回双端列表的长度 |
| [LINSERT key BEFORE|AFTER existuing\_val value](https://www.runoob.com/redis/lists-linsert.html" \t "https://www.runoob.com/redis/_blank)  在列表的元素前或者后插入元素 | 插入新节点到压缩列表的头部或则尾部时，使用ziplistPush函数；插入新节点到压缩列表的中间位置时，使用ziplistInsert函数 | 调用listInsertNode函数，将新节点插入到双端链表的指定位置 |
| [LREM key count value](https://www.runoob.com/redis/lists-lrem.html" \t "https://www.runoob.com/redis/_blank) 移除多少个列表元素 | 遍历压缩列表节点，并调用ziplistDelete函数删除包含了给定元素的节点 | 遍历双端链表节点，并调用listDelNode函数删除包含了给定元素的节点 |
| [LTRIM key start stop](https://www.runoob.com/redis/lists-ltrim.html" \t "https://www.runoob.com/redis/_blank) | 调用ziplistDeleteRange函数，删除压缩列表中所有不在指定索引范围内的节点 | 遍历双端链表节点，并调用listDelNode函数删除所有不在指定索引范围内的节点 |
| [LSET key index value](https://www.runoob.com/redis/lists-lset.html" \t "https://www.runoob.com/redis/_blank)  通过索引设置列表元素的值 | 调用ziplistDelete函数，先删除压缩列表指定索引上的现有节点，然后调用ziplistInsert函数，将一个包含给定元素的新节点插入到相同的索引上面 | 调用listIndex函数，定位到双端链表指定索引上的节点，然后通过赋值操作更新节点的值 |
| [LRANGE key start stop](https://www.runoob.com/redis/lists-lrange.html" \t "https://www.runoob.com/redis/_blank) 获取列表指定范围内的元素 |  |  |
| [RPOPLPUSH source destination](https://www.runoob.com/redis/lists-rpoplpush.html" \t "https://www.runoob.com/redis/_blank) 移除列表的最后一个元素，并将该元素添加到另一个列表并返回  [BRPOPLPUSH source destination timeout](https://www.runoob.com/redis/lists-brpoplpush.html" \t "https://www.runoob.com/redis/_blank)  阻塞加超时时间 |  |  |

## 5.5哈希对象P71

|  |  |  |
| --- | --- | --- |
| **命令及描述** | **ziplist编码的实现方式** | **Hashtable编码的实现方式** |
| [HSET key field value](https://www.runoob.com/redis/hashes-hset.html)  将哈希表key中的字段field的值设为value  [HSETNX key field value](https://www.runoob.com/redis/hashes-hsetnx.html) 只有在字段field不存在时，设置哈希表字段的值 | 首先调用ziplistPush函数，将键推入到压缩列表的尾部，然后再次调用ziplistPush函数，将值推入到压缩列表的尾部 | 调用dictAdd函数，将新节点添加到字典里面 |
| [HGET key field](https://www.runoob.com/redis/hashes-hget.html) | 首先调用ziplistFind函数，在压缩列表中查找指定键所对应的节点，然后调用ziplistNext函数，将指针移动到键节点旁边的节点，最后返回值节点 | 调用dictFind函数，在字典中查找给定键，然后调用dictGetVal函数返回 |
| [HEXISTS key field](https://www.runoob.com/redis/hashes-hexists.html) | 调用ziplistFind函数，在压缩列表中查找指定键所对应的节点，如果找到的话说明键值存在 否则不存在 | 调用dictFind函数，在字典中查找给定键，如果找到的话说明键值存在 否则不存在 |
| [HDEL key field1 [field2]](https://www.runoob.com/redis/hashes-hdel.html)  删除一个或多个哈希表字段 | 调用ziplistFind函数，在压缩列表中查找指定键所对应的节点，然后将相应的键节点以及键节点旁边的值节点全部删除 | 调用dictDelete函数，将指定的键所对应的键和值从字典中删除 |
| [HLEN key](https://www.runoob.com/redis/hashes-hlen.html) | 调用ziplistLen函数，取得压缩列表包含的节点总数，将这个总数除以2，就是键值对的数量 | 调用dictSize函数，返回字典包含的键值对数量 |
| [HGETALL key](https://www.runoob.com/redis/hashes-hgetall.html) | 遍历整个压缩列表，用ziplistGet函数返回所有的键和值 | 遍历整个字典，用dictGetKey函数返回字典的键，用dictGetVal函数返回字典的制值 |
| [HINCRBY key field increment](https://www.runoob.com/redis/hashes-hincrby.html) 为哈希表 key 中的指定字段的整数值加上增量 increment  [HINCRBYFLOAT key field increment](https://www.runoob.com/redis/hashes-hincrbyfloat.html) 为哈希表 key 中的指定字段的浮点数值加上增量 increment |  |  |
| [HKEYS key](https://www.runoob.com/redis/hashes-hkeys.html) 获取所有哈希表中的字段  [HVALS key](https://www.runoob.com/redis/hashes-hvals.html) 获取哈希表中所有值 |  |  |
| [HMSET key field1 value1 [field2 value2 ]](https://www.runoob.com/redis/hashes-hmset.html) 同时将多个 field-value (域-值)对设置到哈希表 key 中  [HMGET key field1 [field2]](https://www.runoob.com/redis/hashes-hmget.html) 获取所有给定字段的值 |  |  |
| HSCAN key cursor [MATCH pattern] [COUNT count] 迭代哈希表中的键值对 |  |  |

## 5.6集合对象P75

|  |  |  |
| --- | --- | --- |
| **命令及描述** | **Intset编码的实现方式** | **Hashtable编码的实现方式** |
| [SADD key member1 [member2]](https://www.runoob.com/redis/sets-sadd.html) | 调用intsetAdd函数，将所有新元素添加到整数集合里面 | 调用dictAdd函数，将新元素为键，NULL为值，将键值对添加到字典里面 |
| [SCARD key](https://www.runoob.com/redis/sets-scard.html) 获取集合的成员数量 | 调用intsetLen函数，返回整数集合所有包含元素的数量 | 调用dictSize函数，返回字典包含的键值对数量 |
| [SISMEMBER key member](https://www.runoob.com/redis/sets-sismember.html) 判断member元素是否是集合 key 的成员 | 调用intsetFind函数，在整数集合中查找给定的元素，如果找到了说明元素存在于集合中，否则不存在 | 调用dictFind函数，在字典中查找给定键，如果找到的话说明键值存在 否则不存在 |
| [SMEMBERS key](https://www.runoob.com/redis/sets-smembers.html) 返回集合中的所有成员 | 遍历整个整数集合，使用intsetGet函数返回集合元素 | 遍历整个字典，使用dictGetKey函数返回字典键作为集合元素 |
| [SRANDMEMBER key [count]](https://www.runoob.com/redis/sets-srandmember.html) 返回集合中一个或多个随机数 | 多次[调用intsetRandom函数，从整数集合中随机返回一个元素] | 调用dictGetRandomKey函数，从字典中随机返回一个字典键 |
| [SPOP key](https://www.runoob.com/redis/sets-spop.html) 移除并返回集合中的一个随机元素 | 调用intsetRandom函数，从整数集合中随机取出一个元素，再将这个随机元素返回给客户端之后，调用intsetRemove函数，将随机元素从整数集合中删除 | 调用dictGetRandomKey函数，从字典中随机取出一个字典键，再将这个随机元素返回给客户端之后，调用dictDelete函数，从字典中删除随机元素字典键所对应的键值对 |
| [SREM key member1 [member2]](https://www.runoob.com/redis/sets-srem.html) 移除集合中一个或多个成员 | 调用intsetRemove函数，从整数集合中删除所有给定的元素 | 调用dictDelete函数，从字典中删除所有键为给定元素的键值对 |
| [SDIFF key1 [key2]](https://www.runoob.com/redis/sets-sdiff.html) 返回给定所有集合的差集  [SDIFFSTORE destination key1 [key2]](https://www.runoob.com/redis/sets-sdiffstore.html) 返回给定所有集合的差集并存储在destination中 | 差集是以key1键的内容为准则  key1 = {b,c,d}  key2 = {c}  SDIFF key1 key2 = {b,d} |  |
| [SINTER key1 [key2]](https://www.runoob.com/redis/sets-sinter.html) 返回给定所有集合的交集  [SINTERSTORE destination key1 [key2]](https://www.runoob.com/redis/sets-sinterstore.html) 返回给定所有集合的交集并存储在 destination 中 | key1 = {b,c,d}  key2 = {c}  SINTER key1 key2 = {c} |  |
| [SUNION key1 [key2]](https://www.runoob.com/redis/sets-sunion.html) 返回所有给定集合的并集  [SUNIONSTORE destination key1 [key2]](https://www.runoob.com/redis/sets-sunionstore.html) 所有给定集合的并集存储在 destination 集合中 | key1 = {b,c,d}  key2 = {c}  SUNION key1 key2 ={b,c,d} |  |
| [SMOVE source destination member](https://www.runoob.com/redis/sets-smove.html) 将 member 元素从 source 集合移动到 destination 集合 | Smove set1 set2 “1” |  |
| [SSCAN key cursor [MATCH pattern] [COUNT count]](https://www.runoob.com/redis/sets-sscan.html) 迭代集合中的元素 | sscan set1 0 match h\* count 2 |  |

## 5.7有序集合对象P77

|  |  |  |
| --- | --- | --- |
| **命令及描述** | **Ziplist编码的实现方式** | **Zset编码的实现方式** |
| [ZADD key score1 member1 [score2 member2]](https://www.runoob.com/redis/sorted-sets-zadd.html) 向有序集合添加一个或多个成员，或者更新已存在成员的分数 | 调用ziplistInsert函数，将成员和分值作为2个节点插入到压缩列表中 | 先调用zslInsert函数，将新元素添加到跳跃表，然后调用dictAdd函数，将新元素关联到字典 |
| [ZCARD key](https://www.runoob.com/redis/sorted-sets-zcard.html) 获取有序集合的成员数量 | 调用ziplistLen函数，取得压缩列表包含的节点总数，将这个总数除以2，就是键值对的数量 | 访问跳跃表数据结构的length属性，直接返回集合元素的数量 |
| [ZCOUNT key minscore max](https://www.runoob.com/redis/sorted-sets-zcount.html)score 计算在有序集合中指定区间分数的成员数 | 遍历压缩列表，统计分值在给定范围内的节点数量 | 遍历跳跃表，统计分值在给定范围内的节点数量 |
| [ZRANGE key start stop [WITHSCORES]](https://www.runoob.com/redis/sorted-sets-zrange.html) 通过索引区间返回有序集合指定区间内的成员  [ZRANGEBYSCORE key minScore maxScore [WITHSCORES] [LIMIT]](https://www.runoob.com/redis/sorted-sets-zrangebyscore.html) 通过分数返回有序集合指定区间内的成员 | 从表头向表尾遍历压缩列表，返回给定索引范围内的所有元素[带分值]  从表头向表尾遍历压缩列表，返回给定分值围内的所有元素[带分值] | 从表头向表尾遍历跳跃表，统计分值在给定范围内的元素[带分值]  从表头向表尾遍历跳跃表，统计给定分值内的元素[带分值] |
| [ZREVRANGE key start stop [WITHSCORES]](https://www.runoob.com/redis/sorted-sets-zrevrange.html) 返回有序集中指定区间内的成员，通过索引，分数从高到低  [ZREVRANGEBYSCORE key maxScore minScore [WITHSCORES]](https://www.runoob.com/redis/sorted-sets-zrevrangebyscore.html) 返回有序集中指定分数区间内的成员，分数从高到低排序 | 从表尾向表头遍历压缩列表，返回给定索引范围内的所有元素[带分值] | 从表尾向表头遍历跳跃表，统计分值在给定范围内的元素[带分值] |
| [ZRANK key member](https://www.runoob.com/redis/sorted-sets-zrank.html) 返回有序集合中指定成员的索引 | 从表头向表尾遍历压缩列表，查找给定的成员，沿途记录节点的数量，当找到给定成员之后，途经节点数量就是该成员所对应元素的排名 | 从表头向表尾遍历跳跃表，查找给定的成员，沿途记录节点的数量，当找到给定成员之后，途经节点数量就是该成员所对应元素的排名 |
| [ZREVRANK key member](https://www.runoob.com/redis/sorted-sets-zrevrank.html) 返回有序集合中指定成员的排名，有序集成员按分数值递减(从大到小)排序 | 从表尾向表头遍历压缩列表，查找给定的成员，沿途记录节点的数量，当找到给定成员之后，途经节点数量就是该成员所对应元素的排名 | 从表尾向表头遍历跳跃表，查找给定的成员，沿途记录节点的数量，当找到给定成员之后，途经节点数量就是该成员所对应元素的排名 |
| [ZREM key member [member ...]](https://www.runoob.com/redis/sorted-sets-zrem.html) 移除有序集合中的一个或多个成员  [ZREMRANGEBYRANK key start stop](https://www.runoob.com/redis/sorted-sets-zremrangebyrank.html) 移除有序集合中给定的排名区间的所有成员  [ZREMRANGEBYSCORE key minScore max](https://www.runoob.com/redis/sorted-sets-zremrangebyscore.html)Score 移除有序集合中给定的分数区间的所有成员 | 遍历压缩列表，删除所有包含给定成员的节点以及被删除成员节点旁边的分值节点  移除有序集合中给定的排名区间的所有成员  移除有序集合中给定的分数区间的所有成员 | 遍历跳跃表，删除所有包含给定成员的跳跃表节点，并且在字典中解除被删除元素的成员和分值的关联 |
| [ZSCORE key member](https://www.runoob.com/redis/sorted-sets-zscore.html) 返回有序集中，成员的分数值 | 遍历压缩列表，查找包含了给定成员节点，然后取出成员节点旁边的分值节点存的分值 | 直接从字典中取出给定成员的分值 |
|  |  |  |
| [ZINCRBY key increment member](https://www.runoob.com/redis/sorted-sets-zincrby.html) 有序集合中对指定成员的分数加上增量 increment | ZINCRBY zset 1 key1 |  |
| [ZLEXCOUNT key minmember max](https://www.runoob.com/redis/sorted-sets-zlexcount.html)member 在有序集合中计算指定字典区间内成员数量  member只能是数字 不能是其他字符  [ZRANGEBYLEX key minmember maxmember [LIMIT offset count]](https://www.runoob.com/redis/sorted-sets-zrangebylex.html) 通过字典区间返回有序集合的成员  [ZREMRANGEBYLEX key minmember max](https://www.runoob.com/redis/sorted-sets-zremrangebylex.html)member 移除有序集合中给定的字典区间的所有成员 | 127.0.0.1:6379>zadd lpl 1 11 2 22 3 33 4 44 5 55 6 66 7 77  127.0.0.1:6379> zlexcount lpl [22 [66  (integer) 5  不包含( 包含[  ZRANGEBYLEX lpl [22 (66  ZRANGEBYLEX lpl [22 [66  127.0.0.1:6379> ZREMRANGEBYLEX lpl [22 [22  (integer) 1 |  |
| [ZINTERSTORE destination numkeys key [key ...]](https://www.runoob.com/redis/sorted-sets-zinterstore.html) 计算给定的一个或多个有序集的交集并将结果集存储在新的有序集合 key 中  相同的member的score分值会相加 | # 有序集 mid\_test  redis 127.0.0.1:6379> ZADD mid\_test 70 "Li Lei"  redis 127.0.0.1:6379> ZADD mid\_test 70 "Han Meimei"  # 另一个有序集 fin\_test  redis 127.0.0.1:6379> ZADD fin\_test 88 "Li Lei"  redis 127.0.0.1:6379> ZADD fin\_test 75 "Han Meimei"  # 交集  redis 127.0.0.1:6379> ZINTERSTORE sum\_point 2 mid\_test fin\_test  (integer) 3  # 显示有序集内所有成员及其分数值  redis 127.0.0.1:6379> ZRANGE sum\_point 0 -1 WITHSCORES   1. "Han Meimei"   2) "145"  3) "Li Lei"  4) "158" |  |
| [ZUNIONSTORE destination numkeys key [key ...]](https://www.runoob.com/redis/sorted-sets-zunionstore.html) [WEIGHTS weight [weight ...]] [AGGREGATE SUM|MIN|MAX] 计算给定的一个或多个有序集的并集，并存储在新的key中 | ZUNIONSTORE salary 2 programmer manager WEIGHTS 2 3  WEIGHTS 2 3解析:  2:针对programmer的key元素分值\*2  3:针对manager的key元素分值\*3 |  |
| [ZSCAN key cursor [MATCH pattern] [COUNT count]](https://www.runoob.com/redis/sorted-sets-zscan.html) 迭代有序集合中的元素（包括元素成员和元素分值） | ZSCAN salary 0 match \*r\* |  |

## 5.8二进制位数组bit P363

|  |  |
| --- | --- |
| **命令及描述** | **bit实现方式** |
|  | 1）计算出offset偏移量指定的二进制所在哪个字节中  byte =[offset ÷ 8]  2）计算offset偏移量指定的二进制是byte字节的第几个二进制位 bit = (offset mod 8) + 1 |
| setbit key offset val  在指定偏移量的二进制位设置值，偏移量从0开始计数 | setbit key 3 1  二进制结果:0000 1000  setbit key 0 1 二进制结果:0000 1001 |
| getbit key offset  获取数组指定偏移量上的二进制位的值 | getbit key 3  (interger) 1 |
| bitcount key  用于统计位数组里面，值为1的二进制位的数量 | bitcount key #0000 1001  (interger) 2 |
| bittop operator result [key...]  对多个位数组进行按位与(and)、按位或(or)、按位异或(xor)运算,取反(not) | x:0000 1011  y:0000 0110  z:0000 0101  bitop and andResult x y z  #0000 0000  bitop or orResult x y z  #0000 1111  bitop xor xorResult x y z  #0000 1000 位一样的为0  bitop not notResult x  #1111 0100 |

# 网络Socket

## select

|  |  |
| --- | --- |
| **序号** | **接口** |
| 1 | int select(int maxfdp,fd\_set \*readfds,fd\_set \*writefds,fd\_set \*errorfds,struct timeval \*timeout);  **参数一:**int maxfdp是一个整数值，是指集合中所有文件描述符的范围，即所有文件描述符的最大值加1;在Windows中这个参数的值无所谓，可以设置不正确;  **参数二:**fd\_set\*readfds是指向fd\_set结构的指针，这个集合中应该包括文件描述符，我们是要监视这些文件描述符的读变化的，即我们关心是否可以从这些文件中读取数据了，如果这个集合中有一个文件可读，select就会返回一个大于0的值，表示有文件可读，如果没有可读的文件，则根据timeout参数再判断是否超时，若超出timeout的时间，select返回0，若发生错误返回负值。可以传入NULL值，表示不关心任何文件的读变化。  **参数三:**fd\_set\*writefds是指向fd\_set结构的指针，这个集合中应该包括文件描述符，我们是要监视这些文件描述符的写变化的，即我们关心是否可以向这些文件中写入数据了，如果这个集合中有一个文件可写，select就会返回一个大于0的值，表示有文件可写，如果没有可写的文件，则根据timeout参数再判断是否超时，若超出timeout的时间，select返回0，若发生错误返回负值。可以传入NULL值，表示不关心任何文件的写变化。  **参数四:**fd\_set \*errorfds同上面两个参数的意图，用来监视文件错误异常。 可传入NULL。  **参数五:**struct timeval \*timeout是select的超时时间，这个参数至关重要，它可以使select处于三种状态，第一，若将NULL以形参传入，即不传入时间结构，就是将select置于阻塞状态，一定等到监视文件描述符集合中某个文件描述符发生变化为止；第二，若将时间值设为0秒0毫秒，就变成一个纯粹的非阻塞函数，不管文件描述符是否有变化，都立刻返回继续执行，文件无变化返回0，有变化返回一个正值；第三，timeout的值大于0，这就是等待的超时时间，即select在timeout时间内阻塞，超时时间之内有事件到来就返回了，否则在超时后不管怎样一定返回，返回值同上述。  **返回值:**负值:select错误;  正值:某些文件可读写或出错  0:等待超时，没有可读写或错误的文件 |
| 2 | FD\_ZERO(fd\_set \*)  清空集合 |
| 3 | FD\_SET(int fd,fd\_set \*)  将一个给定的文件描述符加入集合之中 |
| 4 | FD\_CLR(int fd,fd\_set\*)  将一个给定的文件描述符从集合中删除 |
| 5 | FD\_ISSET(int fd,fd\_set\* )  检查集合中指定的文件描述符是否可以读写 |

## epoll

|  |  |
| --- | --- |
| **序号** | **接口** |
| 1 | LT（level triggered）是缺省的工作方式，并且同时支持block和no-block socket.在这种做法中，[内核](https://baike.baidu.com/item/%E5%86%85%E6%A0%B8" \t "https://baike.baidu.com/item/epoll/_blank)告诉你一个[文件描述符](https://baike.baidu.com/item/%E6%96%87%E4%BB%B6%E6%8F%8F%E8%BF%B0%E7%AC%A6" \t "https://baike.baidu.com/item/epoll/_blank)是否就绪了，然后你可以对这个就绪的fd进行IO操作。如果你不作任何操作，内核还是会继续通知你的，所以，这种模式编程出错误可能性要小一点。传统的select/poll都是这种模型的代表；  ET（edge-triggered）是高速工作方式，只支持non-block socket。在这种模式下，当描述符从未就绪变为就绪时，内核通过epoll告诉你。然后它会假设你知道文件描述符已经就绪，并且不会再为那个文件描述符发送更多的就绪通知，直到你做了某些操作导致那个文件描述符不再为就绪状态了（比如，你在发送，接收或者接收请求，或者发送接收的数据少于一定量时导致了一个EWOULDBLOCK 错误）。但是请注意，如果一直不对这个fd作IO操作（从而导致它再次变成未就绪），[内核](https://baike.baidu.com/item/%E5%86%85%E6%A0%B8" \t "https://baike.baidu.com/item/epoll/_blank)不会发送更多的通知（only once），不过在TCP协议中，ET模式的加速效用仍需要更多的benchmark确认；  ET和LT的区别就在这里体现，LT事件不会丢弃，而是只要读buffer里面有数据可以让用户读，则不断的通知你。而ET则只在事件发生之时通知。可以简单理解为LT是水平触发，而ET则为边缘触发。LT模式只要有事件未处理就会触发，而ET则只在高低电平变换时（即状态从1到0或者0到1）触发。 |
| 2 | int epoll\_create(int size)  内核会产生一个epoll 实例数据结构并返回一个文件描述符epfd，这个特殊的描述符就是epoll实例的句柄，后面的两个接口都以它为中心（即epfd形参）。size参数表示所要监视文件描述符的最大值，不过在后来的Linux版本中已经被弃用（同时，size不要传0，会报invalid argument错误）  **返回值:**返回一个文件描述符epfd |
| 3 | int epoll\_ctl(int epfd， int op， int fd， struct epoll\_event \*event)  **参数一:**epfd epoll\_create返回的文件描述符句柄  **参数二:**op  EPOLL\_CTL\_ADD：向interest list添加一个需要监视的描述符  EPOLL\_CTL\_DEL：从interest list中删除一个描述符  EPOLL\_CTL\_MOD：修改interest list中一个描述符  **参数三:**fd注册的文件描述符  **参数四:**struct epoll\_event \*event结构描述一个文件描述符的epoll行为  struct epoll\_event {  uint32\_t events; /\* 描述epoll事件 \*/  epoll\_data\_t data; /\* 见上面的结构体 \*/  };  =>data域是唯一能给出描述符信息的字段，所以在调用epoll\_ctl加入一个需要监测的描述符时，一定要在此域写入描述符相关信息  =>events域是bit mask，描述一组epoll事件，在epoll\_ctl调用中解释为：描述符所期望的epoll事件，可多选。常用的epoll事件描述如下：  EPOLLIN：描述符处于可读状态  EPOLLOUT：描述符处于可写状态  EPOLLET：将epoll event通知模式设置成edge triggered  EPOLLONESHOT：第一次进行通知，之后不再监测  EPOLLHUP：本端描述符产生一个挂断事件，默认监测事件  EPOLLRDHUP：对端描述符产生一个挂断事件  EPOLLPRI：由带外数据触发  EPOLLERR：描述符产生错误时触发，默认检测事件  **返回值:** 0:成功 -1:失败 |
| 4 | int epoll\_wait(int epfd， struct epoll\_event \*events， int maxevents， int timeout);  阻塞等待注册的事件发生，返回事件的数目，并将触发的事件写入events数组中。  **参数一:**epfd epoll\_create返回的文件描述符句柄  **参数二:**用来记录被触发的events，其大小应该和maxevents一致  **参数三:**events的数量大小  **参数四:**timeout描述在函数调用中阻塞时间上限，单位是ms  timeout = -1表示调用将一直阻塞，直到有文件描述符进入ready状态或者捕获到信号才返回；timeout = 0用于非阻塞检测是否有描述符处于ready状态，不管结果怎么样，调用都立即返回；timeout > 0表示调用将最多持续timeout时间，如果期间有检测对象变为ready状态或者捕获到信号则返回，否则直到超时  **返回值:**负值:epoll\_wait错误;  正值:某些文件可读写 返回实际监听的数量  0:等待超时,没有可读写文件 |

## net

|  |  |
| --- | --- |
| **序号** | **接口** |
| 1 | const char \*inet\_ntop(int af, const void \*src, char \*dst, socklen\_t cnt);  **功能:**将网络二进制ip转换成[点分十进制](https://baike.baidu.com/item/%E7%82%B9%E5%88%86%E5%8D%81%E8%BF%9B%E5%88%B6" \t "https://baike.baidu.com/item/inet_ntop/_blank)IP地址  **参数一:** af  AF\_INET:Ipv4使用 配合参数二&(sa->sin\_addr)  AF\_INET6:Ipv6使用 配合参数二&(sa->sin6\_addr)  **参数三:**dst 存放[点分十进制](https://baike.baidu.com/item/%E7%82%B9%E5%88%86%E5%8D%81%E8%BF%9B%E5%88%B6" \t "https://baike.baidu.com/item/inet_ntop/_blank)IP地址  **参数四:**cnt [点分十进制](https://baike.baidu.com/item/%E7%82%B9%E5%88%86%E5%8D%81%E8%BF%9B%E5%88%B6" \t "https://baike.baidu.com/item/inet_ntop/_blank)IP地址的数组长度  **返回值:**函数成功的话返回字符串的首地址，错误返回NULL |
| 2 | int inet\_pton(int af, const char \*src, void \*dst);  **功能:**将“点分十进制” －> “二进制整数”  **返回值:**[inet\_pton](https://baike.baidu.com/item/inet_pton" \t "https://baike.baidu.com/item/inet_ntop/_blank)函数成功的话返回1，参数无效返回0，错误返回-1； |
| 3 | int fcntl(int fd, int cmd);  **功能:**针对(文件)描述符提供控制.参数fd 是被参数cmd操作(如下面的描述)的描述符  **返回值:**如果出错，所有命令都返回-1，如果成功则返回某个其他值。下列四个命令有特定返回值：F\_DUPFD、F\_GETFD、F\_GETFL、F\_GETOWN.第一个返回新的[文件描述符](https://baike.baidu.com/item/%E6%96%87%E4%BB%B6%E6%8F%8F%E8%BF%B0%E7%AC%A6" \t "https://baike.baidu.com/item/_blank)，接下来的两个返回相应标志，最后一个返回一个正的进程ID或负的[进程组](https://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B%E7%BB%84" \t "https://baike.baidu.com/item/_blank)ID  **参数一:**fd socket描述符  **参数二:**cmd  F\_GETFL 取得文件描述符状态旗标，此旗标为open（）的参数flags |
| 4 | int fcntl(int fd, int cmd, long arg);  **功能:**针对(文件)描述符提供控制.参数fd 是被参数cmd操作(如下面的描述)的描述符  **返回值:**如果出错，所有命令都返回-1，如果成功则返回某个其他值。下列四个命令有特定返回值：F\_DUPFD、F\_GETFD、F\_GETFL、F\_GETOWN.第一个返回新的[文件描述符](https://baike.baidu.com/item/%E6%96%87%E4%BB%B6%E6%8F%8F%E8%BF%B0%E7%AC%A6" \t "https://baike.baidu.com/item/_blank)，接下来的两个返回相应标志，最后一个返回一个正的进程ID或负的[进程组](https://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B%E7%BB%84" \t "https://baike.baidu.com/item/_blank)ID  **参数一:**fd socket描述符  **参数二:**cmd  F\_SETFL 设置文件描述符状态旗标，参数arg为新旗标，但只允许O\_APPEND、**O\_NONBLOCK**  和O\_ASYNC位的改变，其他位的改变将不受影响。  **参数三:**arg  **O\_NONBLOCK** 阻塞:arg &=~O\_NONBLOCK; 非阻塞:arg |= O\_NONBLOCK; |
| 5 | int setsockopt(int sockfd, int level, int optname,const void \*optval, socklen\_t optlen);  **功能:**用于任意类型、任意状态[套接口](https://baike.baidu.com/item/%E5%A5%97%E6%8E%A5%E5%8F%A3/10058888" \t "https://baike.baidu.com/item/_blank)的设置选项值  **参数二:**level：选项定义的层次；支持SOL\_SOCKET、IPPROTO\_TCP、IPPROTO\_IP和IPPROTO\_IPV6  **参数三:**optname：需设置的选项  选项名称　　　　　　　　说明　　　　　　　　　　　　　　　　　　数据类型  ========================================================================  　　　　　　　　　　　　SOL\_SOCKET  ------------------------------------------------------------------------  SO\_BROADCAST　　　　　　允许发送广播数据　　　　　　　　　　　　int  SO\_DEBUG　　　　　　　　允许调试　　　　　　　　　　　　　　　　int  SO\_DONTROUTE　　　　　　不查找路由　　　　　　　　　　　　　　　int  **SO\_ERROR　　　　　　　　获得套接字错误　　　　　　　　　　　　　int**  **SO\_KEEPALIVE　　　　　　保持连接　　　　　　　　　　　　　　　　int**  SO\_LINGER　　　　　　　 延迟关闭连接　　　　　　　　　　　　　　struct linger  SO\_OOBINLINE　　　　　　带外数据放入正常数据流　　　　　　　　　int  **SO\_RCVBUF　　　　　　　 接收缓冲区大小　　　　　　　　　　　　　int**  **SO\_SNDBUF　　　　　　　 发送缓冲区大小　　　　　　　　　　　　　int**  SO\_RCVLOWAT　　　　　　 接收缓冲区下限　　　　　　　　　　　　　int  SO\_SNDLOWAT　　　　　　 发送缓冲区下限　　　　　　　　　　　　　int  SO\_RCVTIMEO　　　　　　 接收超时　　　　　　　　　　　　　　　　struct timeval  **SO\_SNDTIMEO　　　　　　 发送超时　　　　　　　　　　　　　　　　struct timeval**  **SO\_REUSERADDR　　　　　 允许重用本地地址和端口　　　　　　　　　int**  SO\_TYPE　　　　　　　　 获得套接字类型　　　　　　　　　　　　　int  SO\_BSDCOMPAT　　　　　　与BSD系统兼容　　　　　　　　　　　　　 int  ========================================================================  　　　　　　　　　　　　IPPROTO\_IP  ------------------------------------------------------------------------  IP\_HDRINCL　　　　　　　在数据包中包含IP首部　　　　　　　　　　int  IP\_OPTINOS　　　　　　　IP首部选项　　　　　　　　　　　　　　　int  IP\_TOS　　　　　　　　　服务类型  IP\_TTL　　　　　　　　　生存时间　　　　　　　　　　　　　　　　int  ========================================================================  　　　　　　　　　　　　IPPRO\_TCP  ------------------------------------------------------------------------  TCP\_MAXSEG　　　　　　　TCP最大数据段的大小　　　　　　　　　　 int  **TCP\_NODELAY　　　　　　 不使用Nagle算法　　　　　　　　　　　　 int**  ========================================================================  **参数四:**optval：[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88" \t "https://baike.baidu.com/item/_blank)，指向存放选项待设置的新值的[缓冲区](https://baike.baidu.com/item/%E7%BC%93%E5%86%B2%E5%8C%BA" \t "https://baike.baidu.com/item/_blank)。  **参数五:**optlen：optval缓冲区长度。  **返回值:**成功执行时，返回0。失败返回-1，errno被设为以下的某个值  EBADF：sock不是有效的文件描述词  EFAULT：optval指向的内存并非有效的进程空间  EINVAL：在调用setsockopt()时，optlen无效  ENOPROTOOPT：指定的协议层不能识别选项  ENOTSOCK：sock描述的不是套接字 |
| 6 | int getsockname(int sockfd, struct sockaddr \*localaddr, socklen\_t \*addrlen);  **功能:**通过套接字描述符来获取自己的IP地址和端口  **返回值:**如果函数调用成功，则返回0，如果调用出错，则返回-1 |
| 7 | int getpeername(int sockfd, struct sockaddr \*peeraddr, socklen\_t \*addrlen);  **功能:**通过套接字描述符来获取连接对端的IP地址和端口  **返回值:**如果函数调用成功，则返回0，如果调用出错，则返回-1 |
| 8 | int getaddrinfo(const char \*node, const char \*service, const struct addrinfo \*hints, struct addrinfo \*\*res);  **功能:**将主机和服务转换到socket地址  **参数一:**node或service参数最多可能有一个为NULL。node 要么为点分式地址(IPv4-点分十进制, IPv6-十六进制) 要么为主机名。如果hints.ai\_flags包含了AI\_NUMERICHOST标志，则node必须为数字地址。  **参数二:**service为端口号或者端口名。如果不为空，为端口名，则必须可以解析，通过/etc/services。  **参数三:**  如果hints.ai\_flags指定了AI\_CANONNAME标志，则返回的结构体列表中第一个addrinfo结构体的ai\_canonname域指向了主机的正式名字  如果hints.ai\_flags指定了AI\_NUMERICHOST标志，此标志表示调用中的节点名node必须是一个数字地址字符串  适合server端使用:  如果hints.ai\_flags指定了AI\_PASSIVE标志，而且node为NULL,则返回的地址包含INADDR\_ANY或IN6ADDR\_ANY\_INIT适合用来bind将要accept连接的socket。  如果hints.ai\_flags指定了AI\_PASSIVE标志，而且node不为NULL,则忽略AI\_PASSIVE。  适合client端使用:  如果hints.ai\_flags未指定AI\_PASSIVE标志，则返回的地址适合用来connect，sendto，sendmsg。  如果hints.ai\_flags未指定AI\_PASSIVE标志，而且node为NULL, 则返回回环地址，用于本地服务。  根据node和service查找到对应的addr info。如果此信息被connect调用，则表示这是被监听对象；而监听者可以通过node里面设置。node如果是IP V6地址，必须接%eth来指定接口，则监听者就是这个local IP；如果是raw socket，则监听端口就是hints里面的 ai\_protocol。 getaddrinfo() supports the address%scope-id notation for specifying the IPv6 scope-ID.  **返回值:**成功:0 失败:非0的错误值 gai\_strerror获取错误信息  ==================================addrinfo========================================  struct addrinfo {  int ai\_flags;  int ai\_family;  int ai\_socktype;  int ai\_protocol;  socklen\_t ai\_addrlen;  struct sockaddr \*ai\_addr;  char \*ai\_canonname;  struct addrinfo \*ai\_next;  };   |  |  |  |  | | --- | --- | --- | --- | | 参数 | 取值 | 值 | 说明 | | ai\_family | AF\_INET | 2 | IPv4 | | AF\_INET6 | 23 | IPv6 | | AF\_UNSPEC | 0 | 协议无关 | | ai\_protocol | IPPROTO\_IP | 0 | IP协议 | | IPPROTO\_IPV4 | 4 | IPv4 | | IPPROTO\_IPV6 | 41 | IPv6 | | IPPROTO\_UDP | 17 | UDP | | IPPROTO\_TCP | 6 | TCP | | ai\_socktype | SOCK\_STREAM | 1 | 流 | | SOCK\_DGRAM | 2 | 数据报 | | ai\_flags | AI\_PASSIVE | 1 | 被动的，用于bind，通常用于server socket | | AI\_CANONNAME | 2 |  | | AI\_NUMERICHOST | 4 | 地址为数字串 | |
| 9 | const char \*gai\_strerror(int error);  **返回值:**指向错误描述消息字符串的指针 getaddrinfo返回非0的错误值 |
| 10 | char\*strerror(int errnum);  **功能:**通过标准错误的标号，获得错误的描述字符串 ，将单纯的错误标号转为字符串描述，方便用户查找错误 |

# 附件

## 7.1参考资料

1.《Redis设计与实践》

2.菜鸟教程