

Redis

[零 参考资料 3](#_Toc47686968)

[一 涉及的C语言知识点 3](#_Toc47686969)

[1 Pointer 3](#_Toc47686970)

[2 malloc 3](#_Toc47686971)

[3 数据类型 3](#_Toc47686972)

[4 数据结构 4](#_Toc47686973)

[4.1 struct 4](#_Toc47686974)

[4.2 typedef struct 4](#_Toc47686975)

[5 定义宏 4](#_Toc47686976)

[5.1 #define 4](#_Toc47686977)

[5.2 #ifdef … [#else … ] #endif 4](#_Toc47686978)

[5.3 #ifndef … #endif 4](#_Toc47686979)

[二 SDS 4](#_Toc47686980)

[1 概述 4](#_Toc47686981)

[2 Create 4](#_Toc47686982)

[3 Retrieve 6](#_Toc47686983)

[4 Update 6](#_Toc47686984)

[5 Delete 6](#_Toc47686985)

[三 跳跃表 6](#_Toc47686986)

[1 概述 6](#_Toc47686987)

[2 Create 7](#_Toc47686988)

[3 Retrieve 7](#_Toc47686989)

[4 Update 7](#_Toc47686990)

[5 Delete 7](#_Toc47686991)

[四 ZipList(压缩列表) 7](#_Toc47686992)

[1 概述 7](#_Toc47686993)

[1.1 7](#_Toc47686994)

[2 Create 7](#_Toc47686995)

[3 Retrieve 8](#_Toc47686996)

[4 Update 8](#_Toc47686997)

[5 Delete 8](#_Toc47686998)

[五 dict字典 8](#_Toc47686999)

[1 概述 8](#_Toc47687000)

[2 Create 8](#_Toc47687001)

[3 Retrieve 8](#_Toc47687002)

[4 Update 8](#_Toc47687003)

[5 Delete 8](#_Toc47687004)

[六 整数 8](#_Toc47687005)

[1 概述 8](#_Toc47687006)

[2 Create 9](#_Toc47687007)

[3 Retrieve 9](#_Toc47687008)

[4 Update 9](#_Toc47687009)

[5 Delete 9](#_Toc47687010)

[七 quicklist 9](#_Toc47687011)

[1 概述 9](#_Toc47687012)

[2 Create 10](#_Toc47687013)

[3 Retrieve 10](#_Toc47687014)

[4 Update 10](#_Toc47687015)

[5 Delete 10](#_Toc47687016)

[八 Stream 10](#_Toc47687017)

[1 概述 10](#_Toc47687018)

[2 Create 10](#_Toc47687019)

[3 Retrieve 10](#_Toc47687020)

[4 Update 10](#_Toc47687021)

[5 Delete 10](#_Toc47687022)

[九 其他相关源码 10](#_Toc47687023)

# 零 参考资料

《C primer plus》第六版

《Redis 5设计与源码分析》

Redis 5.0.8源码

# 涉及的C语言知识点

## Pointer

## malloc

## 数据类型

|  |  |
| --- | --- |
| 类型 | 说明 |
| int8\_t |  |
| int16\_t |  |
| int32\_t |  |
| uint8\_t |  |
| uint8\_t |  |
| uint8\_t |  |
|  |  |

## 数据结构

### struct

### typedef struct

## 定义宏

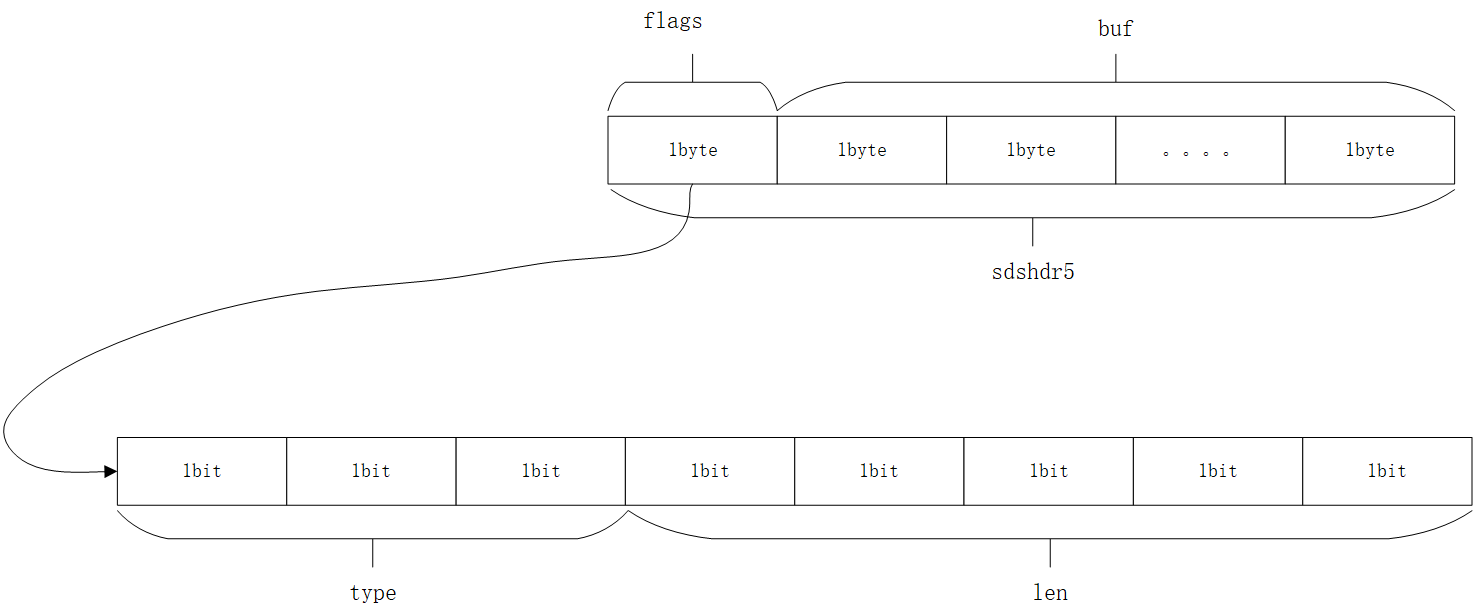
### #define

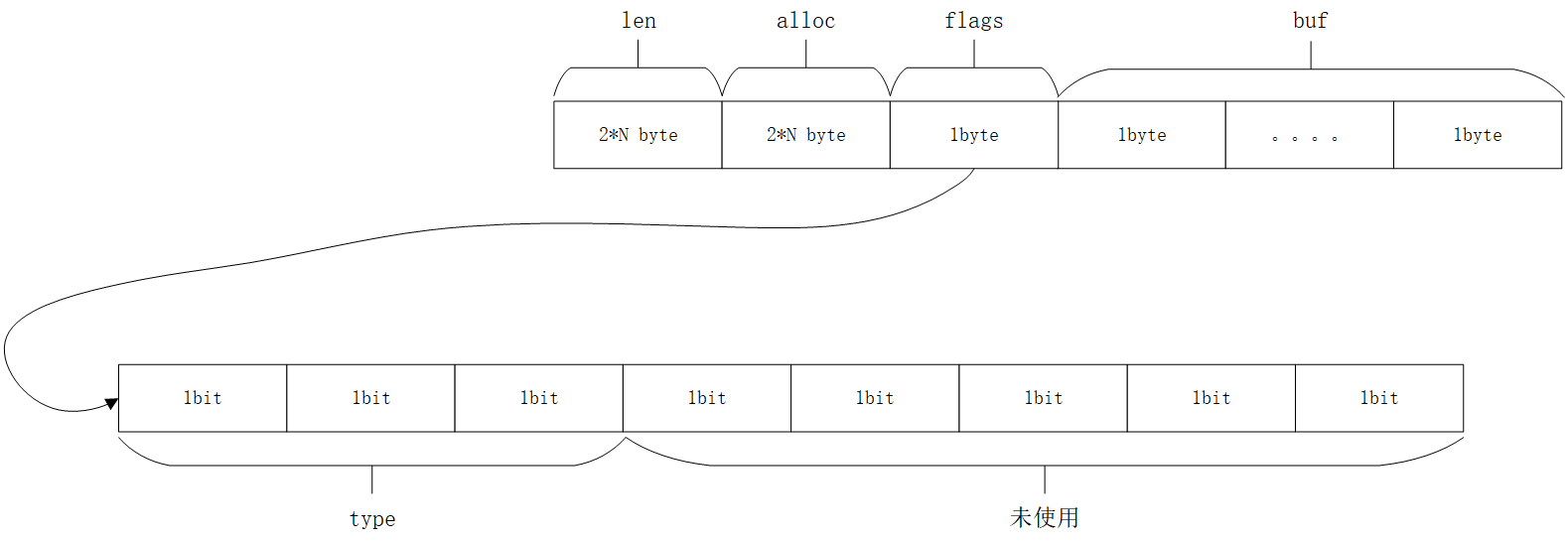
### #ifdef … [#else … ] #endif

### #ifndef … #endif

# SDS

## 概述





## Create

1. /\*\*
2. \* 创建一个新的SDS String
3. \* size\_t  是部分C/C++标准在stddef.h中定义的。这个类型足以用来表示对象的大小。size\_t的真实类型与操作系统有关
4. \*         32位上大部分是typedef   unsigned int size\_t;
5. \*         64位上大部分是typedef  unsigned long size\_t;
6. \* @param init 初始化内容
7. \* @param initlen 初始化大小
8. \*/
9. sds sdsnewlen(**const** **void** \*init, **size\_t** initlen) {
10. **void** \*sh;
11. //声明sds数据部分
12. sds s;
13. //判断初始化长度对应的实际数据结构类型
14. **char** type = sdsReqType(initlen);
15. //如果初始长度小于32bit，则转为SDS\_TYPE\_8
16. **if** (type == SDS\_TYPE\_5 && initlen == 0) type = SDS\_TYPE\_8;
17. //根据type获取对应的数据结构头部数据长度
18. **int** hdrlen = sdsHdrSize(type);
19. unsigned **char** \*fp; /\* flags pointer. \*/
21. /\*\*
22. \* 分配sds对象
23. \* 这里分配的形式是分配一个指定长度的连续空间
24. \* sh是指向分配到的空间的起始地址
25. \* 这里之所以要加 1 ，是因为c的字符串尾部都有个\0，为了减少sds和c字符串之间转换的成本，所以预留了这部分位置
26. \*/
27. sh = s\_malloc(hdrlen+initlen+1);
28. **if** (init==SDS\_NOINIT)
29. init = NULL;
30. **else** **if** (!init)
31. /\*\*
32. \* 将sds对象数据初始化为0
33. \* \*memset(void \*buffer 对象指针, int c赋值, int count 位数)，即从\*buffer的起始地址开始，将后续的count位数赋值位c
34. \*/
35. memset(sh, 0, hdrlen+initlen+1);
36. //如果sds对象分配失败，则直接返回NULL
37. **if** (sh == NULL) **return** NULL;
38. //通过sh加上头部数据长度，s指向SDS内容数据的起始地址，获得SDS数据对象
39. s = (**char**\*)sh+hdrlen;
40. //s地址向后移动一位指向flag所在地址，减获得flags值
41. fp = ((unsigned **char**\*)s)-1;
42. /\*\*
43. \* 根据数据结构类型初始化数据头
44. \* SDS\_HDR\_VAR(T,s) struct sdshdr##T \*sh = (void\*)((s)-(sizeof(struct sdshdr##T)));
45. \* SDS\_HDR\_VAR宏 作用是获取指定sds对象的起始地址
46. \*/
47. **switch**(type) {
48. //因为type不可能是SDS\_TYPE\_5，这里忽略
49. **case** SDS\_TYPE\_5: {
50. \*fp = type | (initlen << SDS\_TYPE\_BITS);
51. **break**;
52. }
53. **case** SDS\_TYPE\_8: {
54. SDS\_HDR\_VAR(8,s);
55. sh->len = initlen;
56. sh->alloc = initlen;
57. \*fp = type;
58. **break**;
59. }
60. **case** SDS\_TYPE\_16: {
61. SDS\_HDR\_VAR(16,s);
62. sh->len = initlen;
63. sh->alloc = initlen;
64. \*fp = type;
65. **break**;
66. }
67. **case** SDS\_TYPE\_32: {
68. SDS\_HDR\_VAR(32,s);
69. sh->len = initlen;
70. sh->alloc = initlen;
71. \*fp = type;
72. **break**;
73. }
74. **case** SDS\_TYPE\_64: {
75. SDS\_HDR\_VAR(64,s);
76. sh->len = initlen;
77. sh->alloc = initlen;
78. \*fp = type;
79. **break**;
80. }
81. }
82. //初始化成功，则开始赋值内容数据
83. **if** (initlen && init)
84. //从SDS数据对象的起始地址开始，赋值内容
85. memcpy(s, init, initlen);
86. //在数据最后，加上\0
87. s[initlen] = '\0';
88. //返回sds内容数据部分
89. **return** s;
90. }

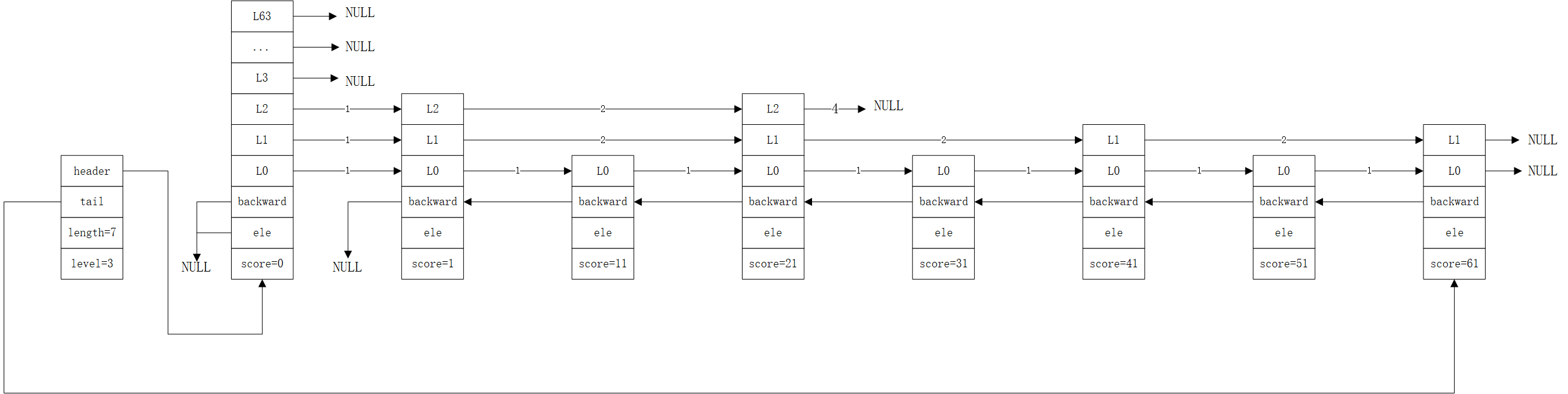
## Retrieve

## Update

## Delete

# 跳跃表

## 概述



## Create

## Retrieve

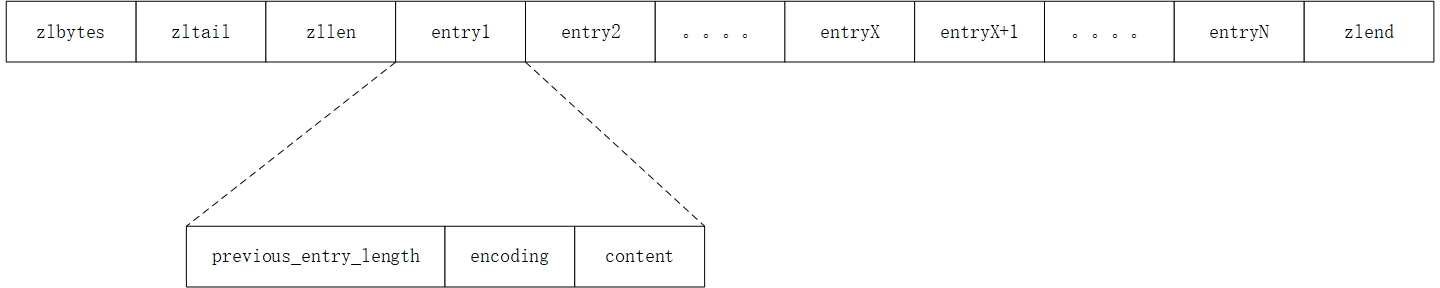
## Update

## Delete

# ZipList(压缩列表)

## 概述

### 



## Create

1. /\*\*
2. \* 创建一个新的空的ziplist
3. \* @return zl对象
4. \*/
5. unsigned **char** \*ziplistNew(**void**) {
6. //初始化大小  头部 (sizeof(uint32\_t)\*2+sizeof(uint16\_t))+ 尾部(sizeof(uint8\_t))
7. unsigned **int** bytes = ZIPLIST\_HEADER\_SIZE+ZIPLIST\_END\_SIZE;
8. //分配对应的空间
9. unsigned **char** \*zl = zmalloc(bytes);
10. /\*\*
11. \* (\*((uint32\_t\*)(zl))) = intrev32(bytes)
12. \* intrev32ifbe 将大端值表示法转为小端，if big-end
13. \*/
14. ZIPLIST\_BYTES(zl) = intrev32ifbe(bytes);
15. /\*\*
16. \* (\*((uint32\_t\*)((zl)+sizeof(uint32\_t)))) = intrev32(ZIPLIST\_HEADER\_SIZE)
17. \* zl即对象起始地址 + uint32\_t 大小 =即指向尾部偏移量值
18. \* 向尾部偏移量值 = (sizeof(uint32\_t)\*2+sizeof(uint16\_t))
19. \*/
20. ZIPLIST\_TAIL\_OFFSET(zl) = intrev32ifbe(ZIPLIST\_HEADER\_SIZE);
21. /\*\*
22. \* (\*((uint16\_t\*)((zl)+sizeof(uint32\_t)\*2)))
23. \* zl即对象起始地址 + uint32\_t的大小\*2 的到存储list大小的地址
24. \* 初始化list len =0
25. \*/
26. ZIPLIST\_LENGTH(zl) = 0;
27. //zip list 末尾固定值 255
28. zl[bytes-1] = ZIP\_END;
29. //返回ziplist
30. **return** zl;
31. }

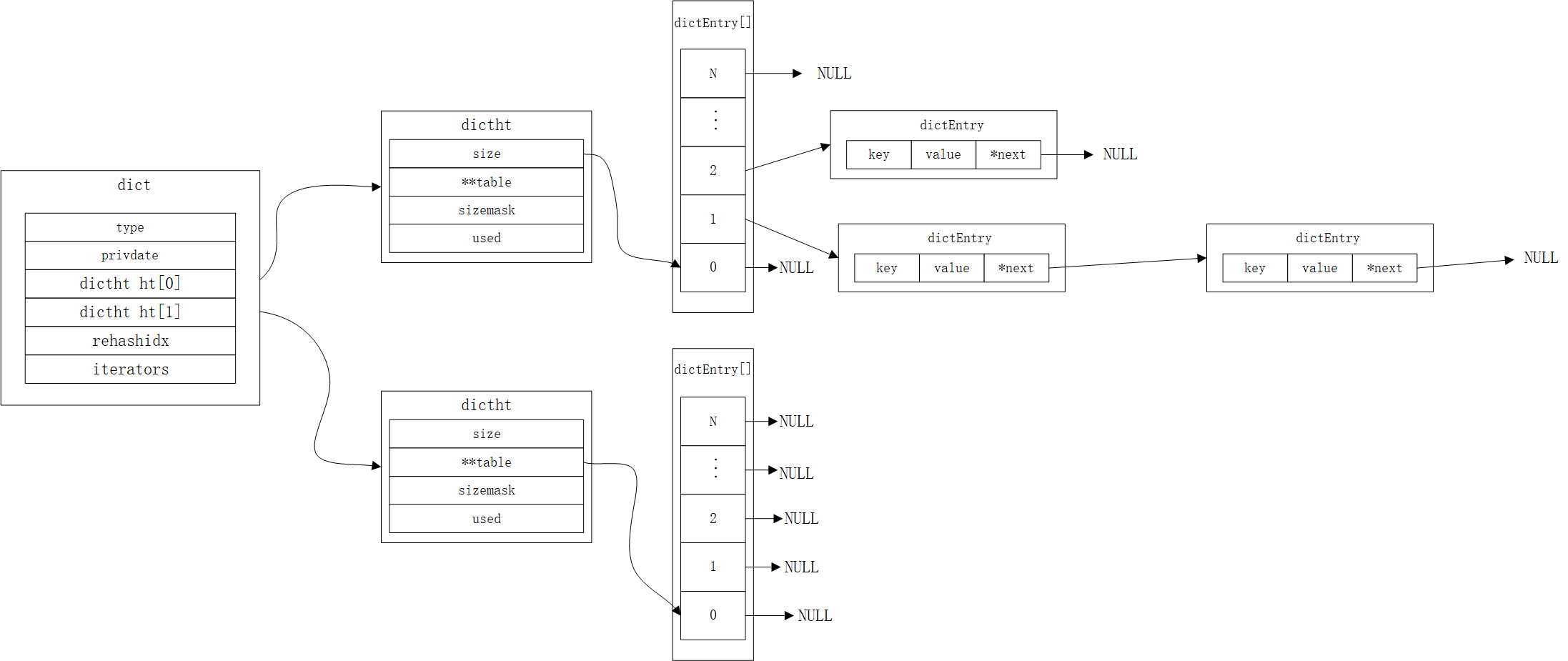
## Retrieve

## Update

## Delete

# dict字典

## 概述



## Create

## Retrieve

## Update

## Delete

# 整数

## 概述



## Create

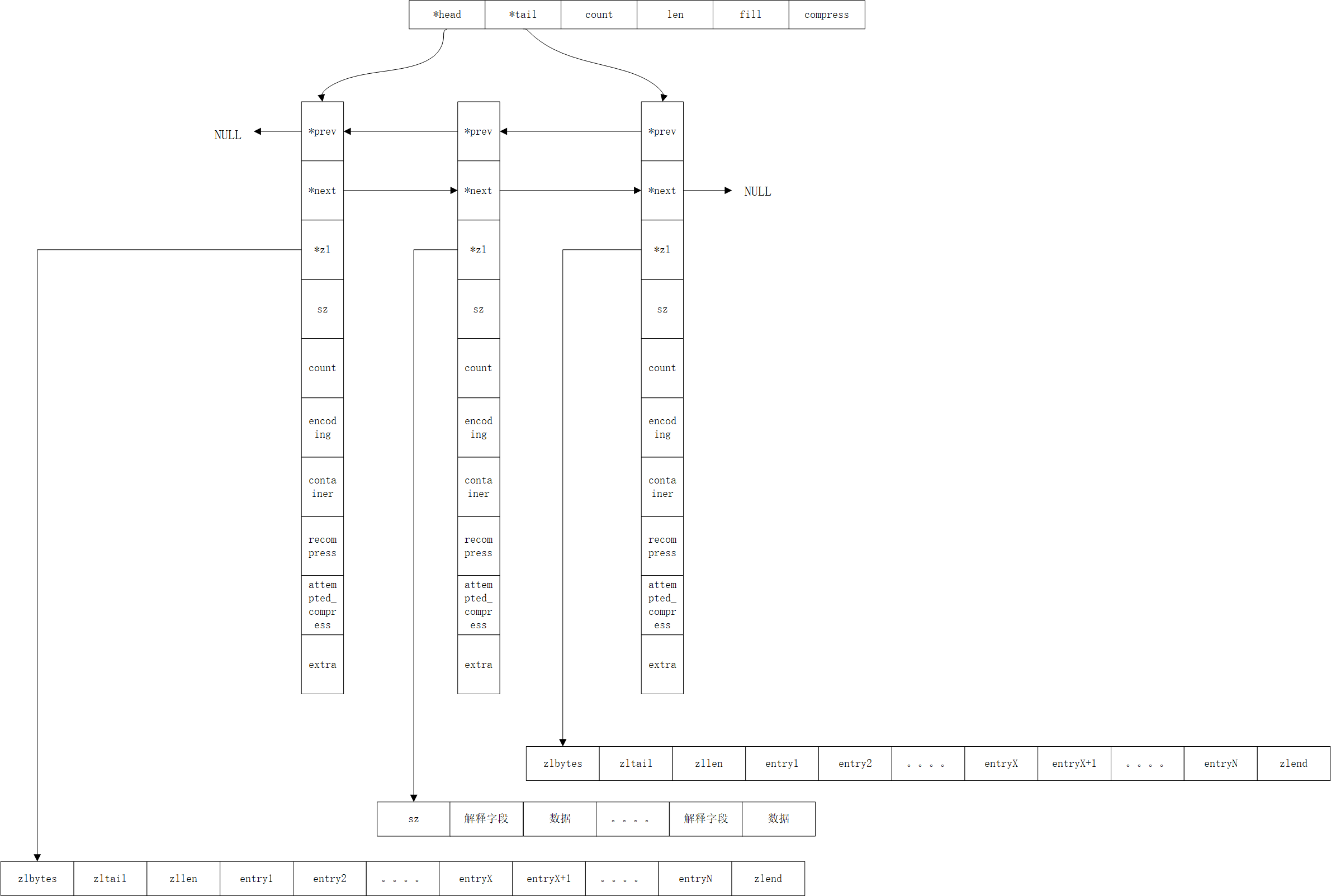
## Retrieve

## Update

## Delete

# quicklist

## 概述



## Create

## Retrieve

## Update

## Delete

# Stream

## 概述

## Create

## Retrieve

## Update

## Delete

# 其他相关源码