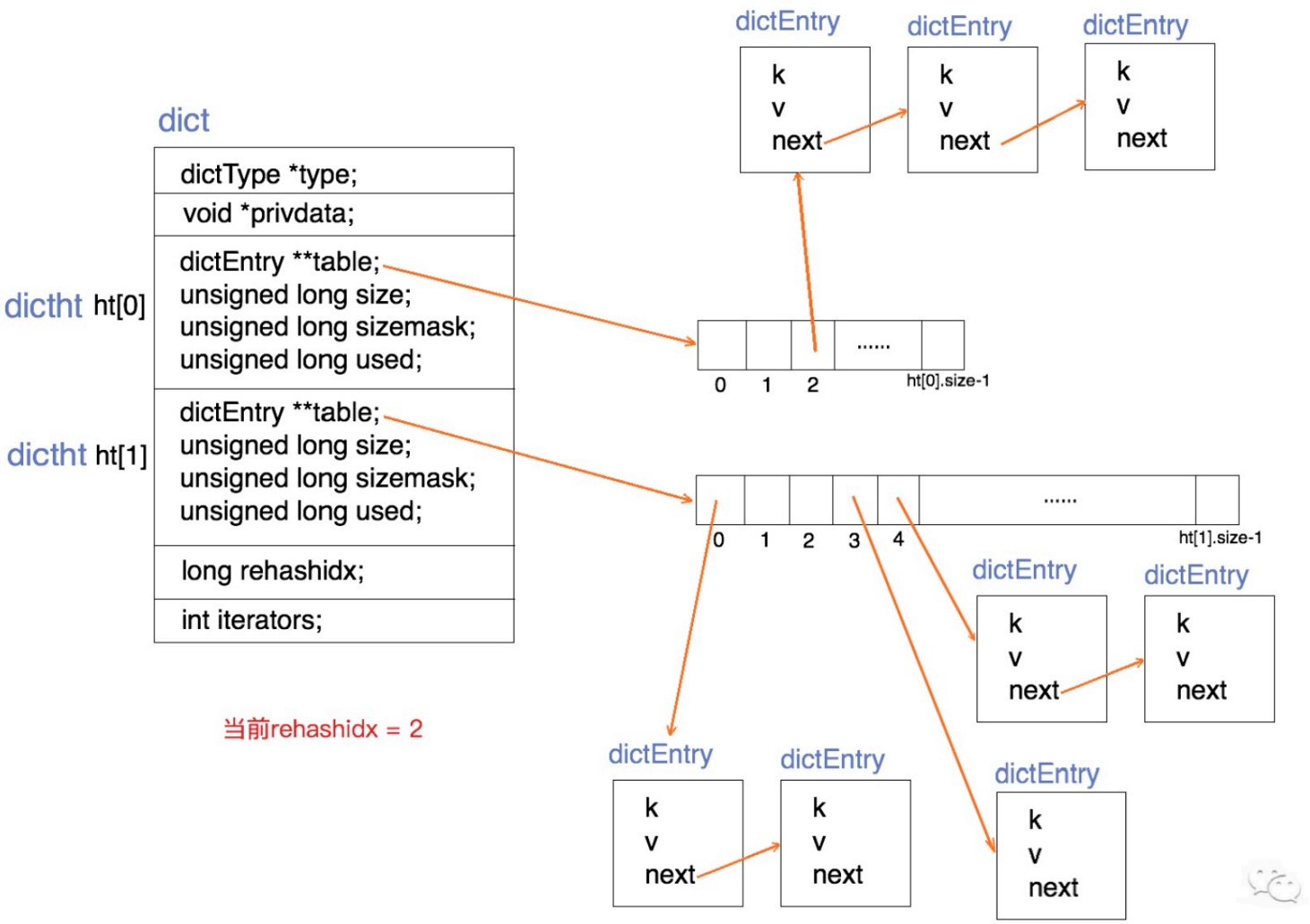
Redis底层数据结构

dict (字典)

dict是基于哈希表的算法，类似于Map。它与其他hash表不同的地方在于它内部的两个hash表以及它的重hash过程（又称增量式重hash）。



- 事实上，除了查找，插入和删除都会出发重hash的过程。
- dictFind 过程：计算出key的hash值，并首先在ht[0]上查找，找到就立刻返回，找不到的话看是否在重hash过程中，如果没有重hash就返回 null，如果在重hash过程中则会在ht[1]中查找
- rehash过程：每一次调用rehash的时候会将rehash迁移的位置向未迁移的桶向前推1步，如果下一个桶没有则会推进1*10步，此次结束。最终ht[0]上无数据时则迁移完成，此时将ht[0]变成

ht[1],并重新置空ht[1]。

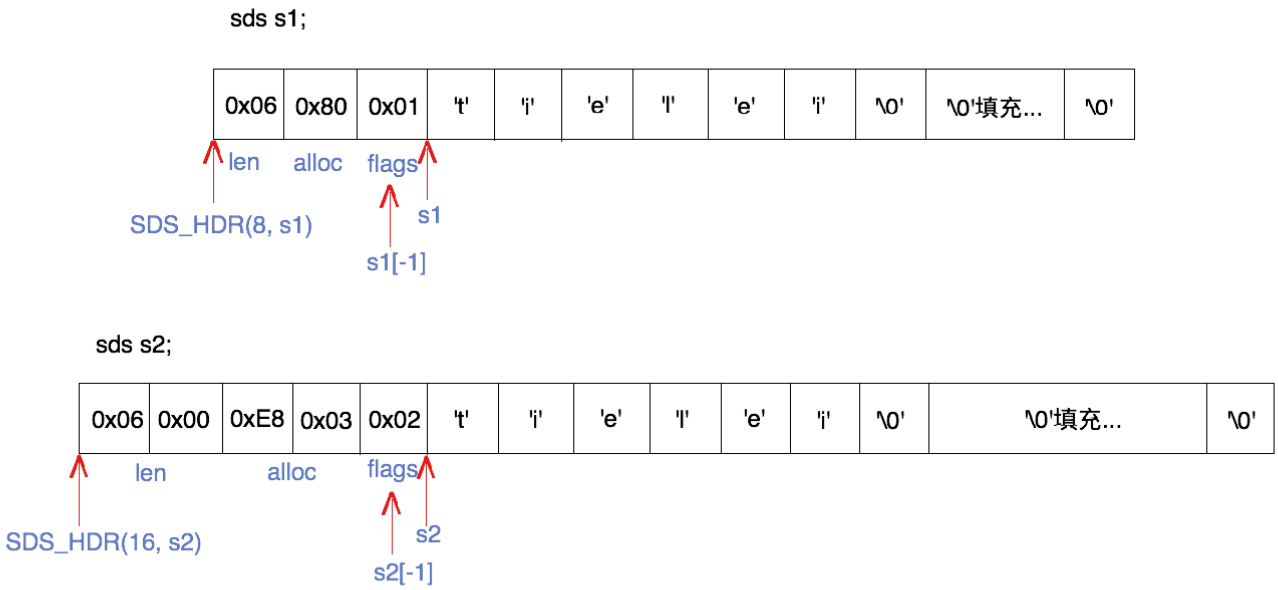
- dict add 与 delete操作类似，都会先判断是否在rehash过程中，如果在rehash过程中在ht[0]中找key是否存在，否则，在ht[0]与ht[1]中查找。

sds (Simple Dynamic String)

```
结构:  header + char 数组

    一个header包含: length(长度), alloc(最大容量),flag(标识那种类型header)。
    char数组通常要比length大 1,用来存放NULL结束标识。

    存在五种header类型:
        define SDS_TYPE_5 0;
        define SDS_TYPE_8 1;
        define SDS_TYPE_16 2;
        define SDS_TYPE_32 3;
        define SDS_TYPE_64 4;
```



- 获取header的方法就是先将s1,s2指针左移1位，定位到字符串header的类型。0x01 == SDS_TYPE_8，因此s1的header类型是sdshdr8。0x02 == SDS_TYPE_16，因此s2的header类型是sdshdr16。

- sdshdr5 与其它几个 header 结构不同，它不包含alloc 字段，而长度使用flags的高5位来存储。因此，它不能为字符串分配空余空间。如果字符串需要动态增长，那么它就必然要重新分配内存才行。所以说，这种类型的sds字符串更适合存储静态的短字符串（长度小于32）。
- Redis正是使用 header + char 数组的形式减少了内存碎片，而不用分成两块空间来存储。
- Redis sds，其实类似于 StringBuilder 对字符串的操作。比如：append、getRange等操作，就是对字符数组的操作，但是redis的String类型并不是用的sds，而是robj。

robj（Redis Object）

```
一个Redis Object 包含:
    type ： 数据类型 (OBJ_STRING, OBJ_LIST, OBJ_SET, OBJ_ZSET, OBJ_HASH)
    encoding ： 对象的内部表达方式
        (
        OBJ_ENCODING_RAW,
        OBJ_ENCODING_INT,
        OBJ_ENCODING_HT,
        OBJ_ENCODING_ZIPLIST,
        OBJ_ENCODING_INTSET,
        OBJ_ENCODING_SKIPLIST,
        OBJ_ENCODING_EMBSTR,
        OBJ_ENCODING_QUICKLIST
        )
    lru ： 做lru替换算法用
    refcount ： 引用计数
    ptr ： 指向真正的数据
```

- 这里需要注意的是encoding，对于同一个type还可能有不同的encoding。比如当type为String时，encoding可能为：
    - OBJ_ENCODING_RAW 表示一个原生sds。
    - OBJ_ENCODING_INT 用String表示的数字，实际为long型。
    - OBJ_ENCODING_EMBSTR 采用特殊嵌套的sds表示，用于存储长度小于等于44的字符串。
- 当用 reids 进行 set 操作时，在 redis 内部会进行编码操作：
    - 先试图将其转换为long型，在成功转换为long型之后又分为两种情况，如果数字小于10000，会使用预存的小数字；如果不能则直接使用ptr存储数字。

- 不能转换为long类型，如果字符串长度足够小（小于44）会使用OBJ_ENCODING_EMBSTR，将robj与sds重新分配在一个内存64字节长度（16字节robj + 3字节sdshdr8 + 44字节sds字符数组 + 1字节结束符）的内存中，减少内存碎片。

- String 在redis中是用 robj 来表示
- refcount 如果只剩下一个引用，则再次decrRefCount 会释放整个obj。

ziplist

> 特殊编码的双向链表，一大块内存，对值的存储采用变长的编码方式，
> 对大的值多一些字节来存储，跟普通链表的不同是，不像其每一项会占用一块内存，节省内存，以及减少内存碎片。
> 但是，每次变动都会导致内存的重新分配。

- hash 与 ziplist

  创建 hash 结构最先使用的是 ziplist，会生成两个数据项插入到 ziplist 中，上图中的数据即是使用
  ```
  hset user:100 name tielei
  hset user:100 age 20
  ```
  命令创建。
  但是随着数据的插入，hash 底层的 ziplist 会转换为 dict。

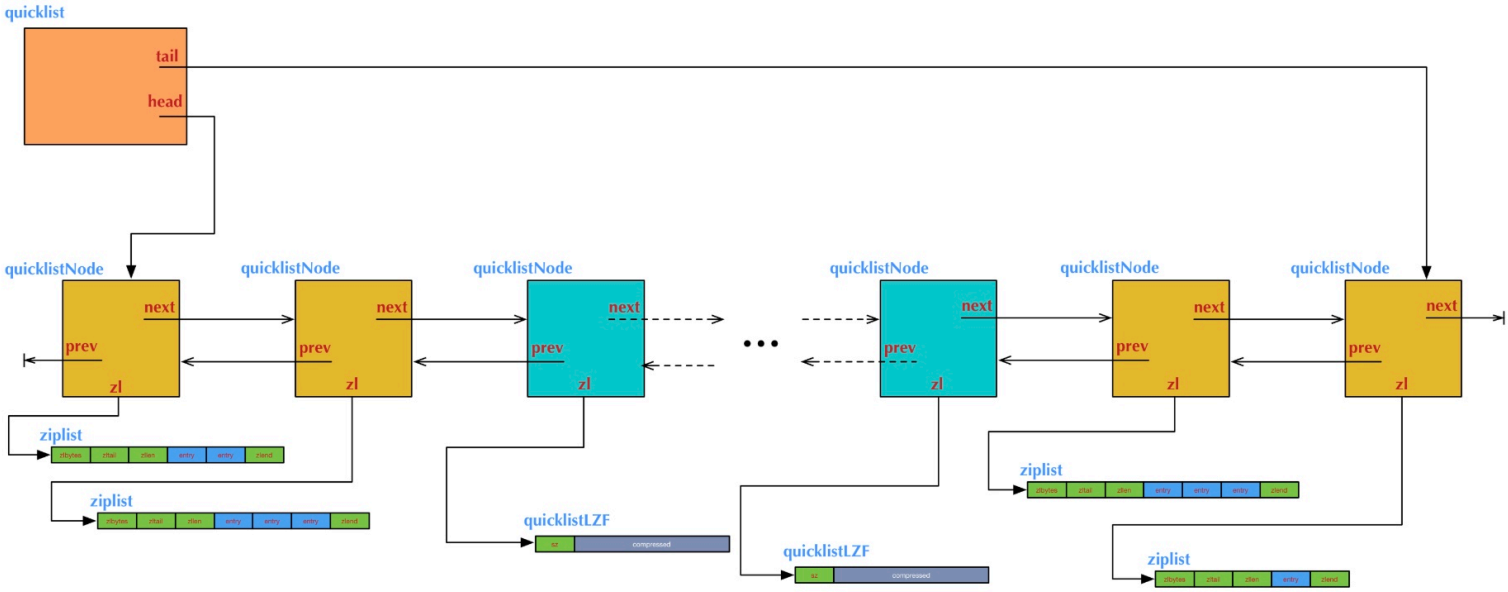
  - 当 ziplist 数据项超过1024时，会转换为 dict。
  - 或者当 ziplist 插入的 value 长度超过64。

quicklist 实现list的数据结构

> redis 对外暴露的上层list，底层实现就是quicklist。quicklist是一个双向链表，并且是一个ziplist

双向链表。
双向链表 + ziplist。

- quicklist 这样设计是为了空间与效率的折中。
    - 双向链表内存开销大，包含指针以及每个节点的内存。同时节点多了也相对来说容易产生内存碎片。
    - ziplist 是一整块的内存，但是不利于修改操作，每次都会重新分配内存。
- 那么一个 quicklist 节点，包含多少 ziplist 合适呢?
  主要取决于适用场景。redis 默认配置为 `list-max-ziplist-size -2`，代表的是为 8kb 的 ziplist，这个配置值为 -1至-5，代表从 4kb～64kb。
- 当列表很长的时候 quicklist 还支持对中间的内容进行压缩，以节省内存空间。redis可设置两端不压缩的个数， `list-compress-depth 0` 是 redis 的默认值，表示不压缩，设置为1两段默认1个节点不压缩，以此类推···



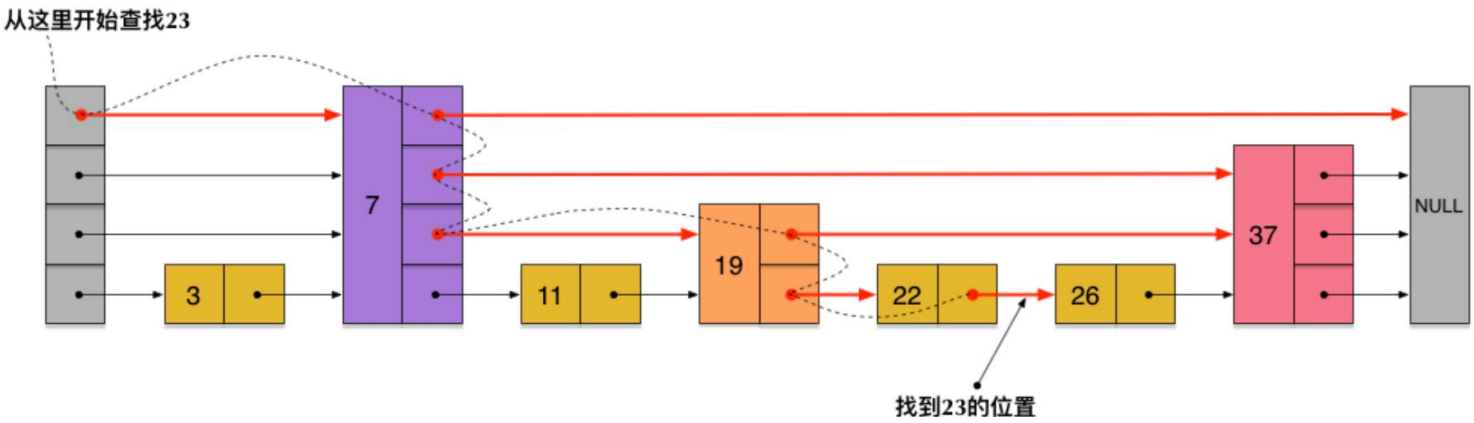
redis list数据结构图。

- list push操作

  如果插入的位置节点中 ziplist 还可以插入，那么插入，否则去看相邻节点的 ziplist 是否可以插入，如果还不能，则会新建节点 quicklist 节点

skiplist 实现sorted set的数据结构

`skiplist 数据结构为层式链表。查找复杂度 logN`



- 关于层数其实类似于二叉树，关于跳表每次新增数据维护高度是根据

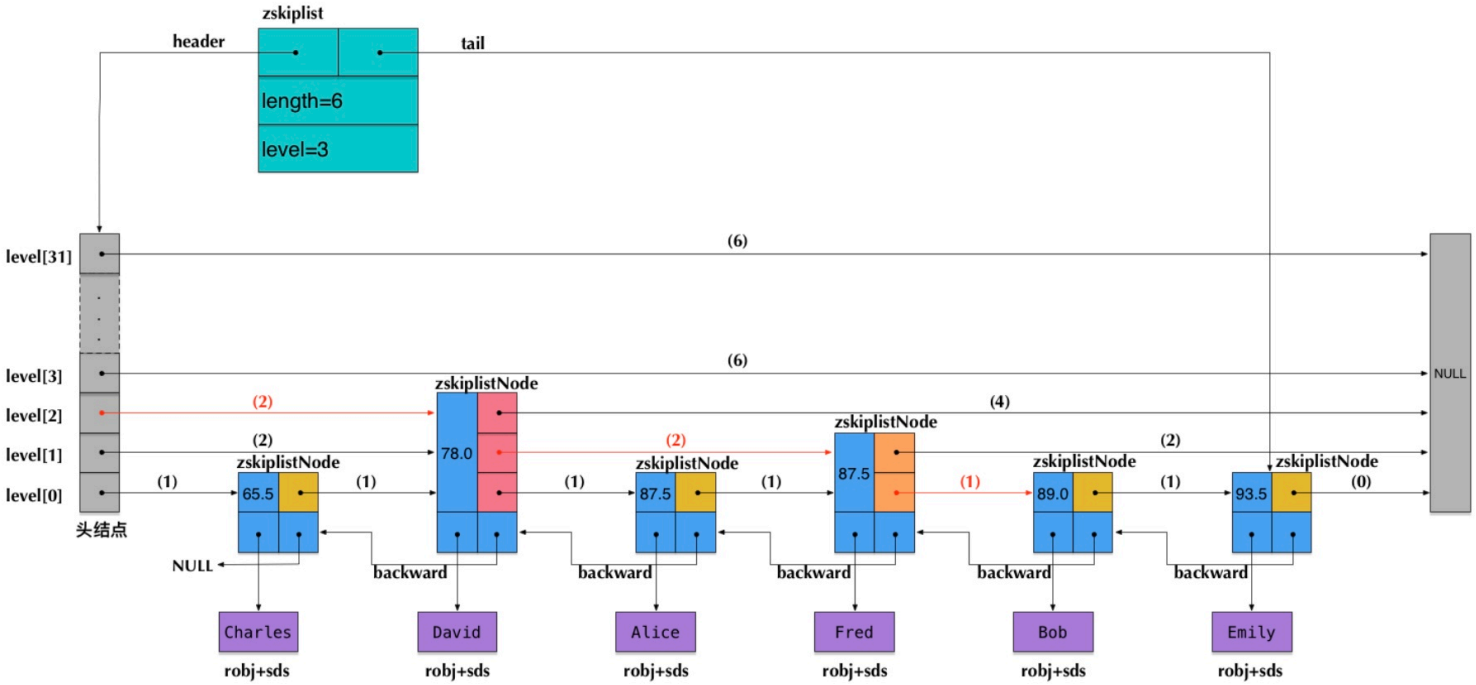
```
randomLevel()
level := 1
// random()返回一个[0...1)的随机数
while random() < p and level < MaxLevel do
level := level + 1
return level
```

。

在 redis 中 p 与 MaxLevel 的取值如下：

```
p = 1/4; MaxLevel = 32
```

- redis 中sorted set的实现不仅仅是用了 skiplist，还使用了 dict、ziplist。
- redis 中 skiplist 的可能结构：



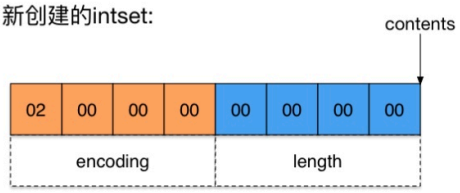
当查找 score = 89 的元素会跨越红色的指针，并将span的值累加就可得到Bob的排名。也可通过累加值取出指定排名。

关于redis 中的 sorted set 实现:
-> 数据量小时会使用 ziplist 来实现;
->当数据对超过 128 ，或者插入的数据长度超过 64 时会转换为 zset。zset 包含一个 dict 与 skiplist。dict 用来定位key -> score，而 skiplist 用来定位 score -> key。

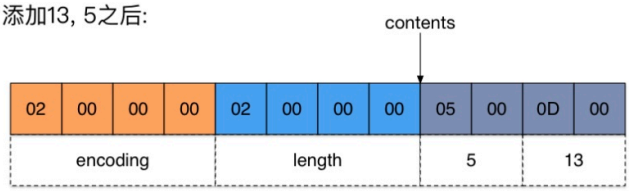
intset

与ziplist类似，内存连续的一整块内存，intset是一个有序的数据集合，可以使用二分查找迅速判断某元素在不在集合中。

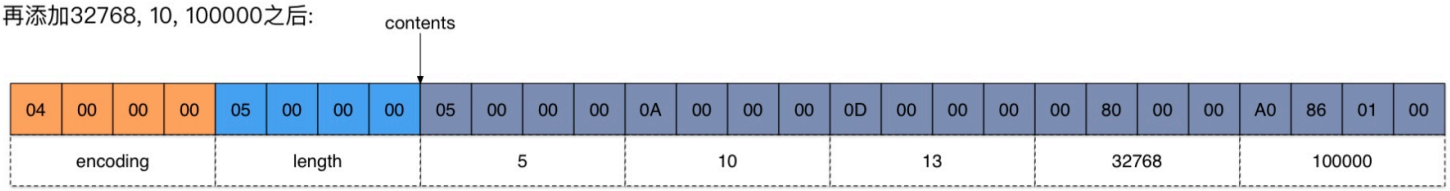
新创建的intset:



添加13, 5之后:



再添加32768, 10, 100000之后:



- 当较小整数的时候占用较小的字节，当较大数插入是会将其转换为大字节，他不像 ziplist 一样有每项数据有自己的字节长度。
- 关于 redis 中 set 的实现；其实是 intset 与 dict 的组合。转换为dict的情况：

```
-> 当添加非数字元素时，会转换为dict;
-> 当超过64bit位所能表示的最大整数(2^64~2^64-1);
-> 当元素超过 `set-max-intset-entries` 所设置的值时;
```