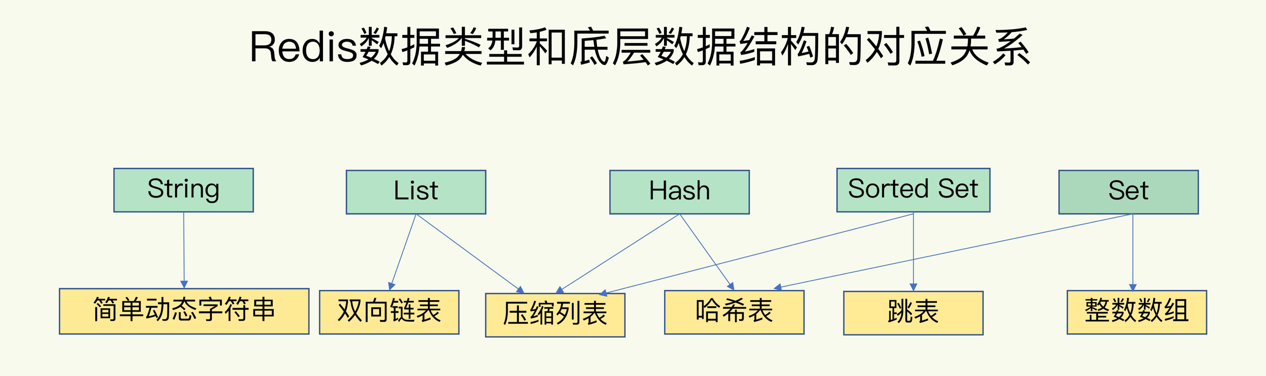
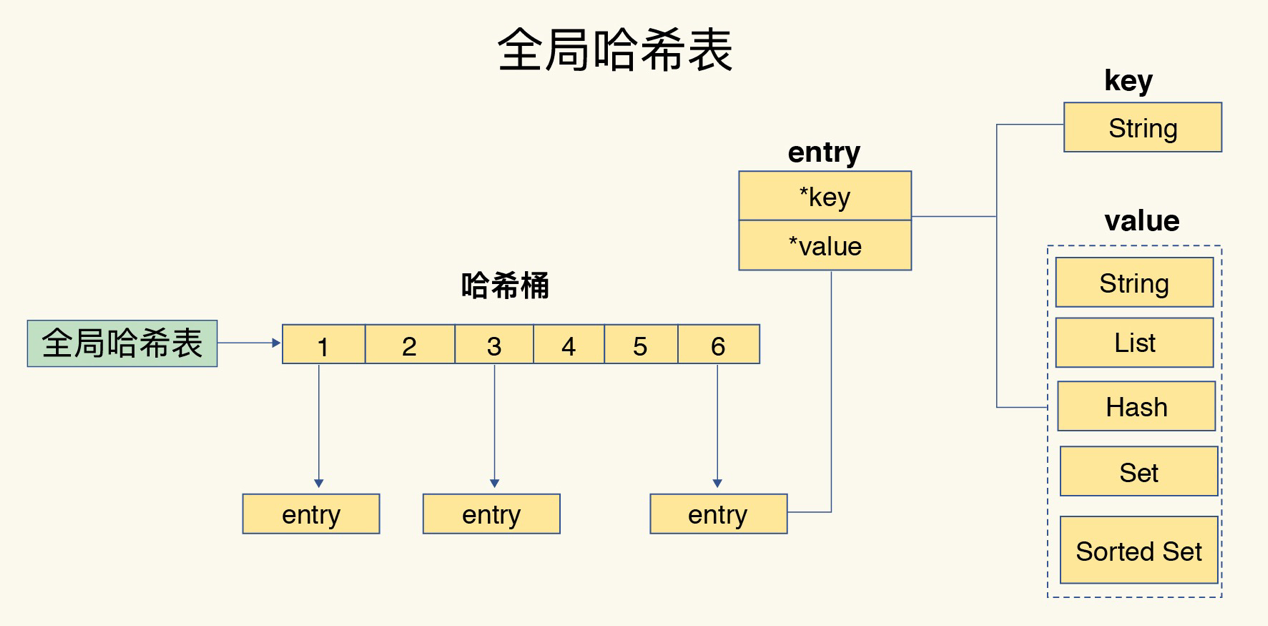
数据结构



一、键和值的结构组织

Redis 使用了一个哈希表来保存所有键值对。

一个哈希表，其实就是一个数组，数组的每个元素称为一个哈希桶。哈希桶中的元素保存的并不是值本身，而是指向具体值的指针。

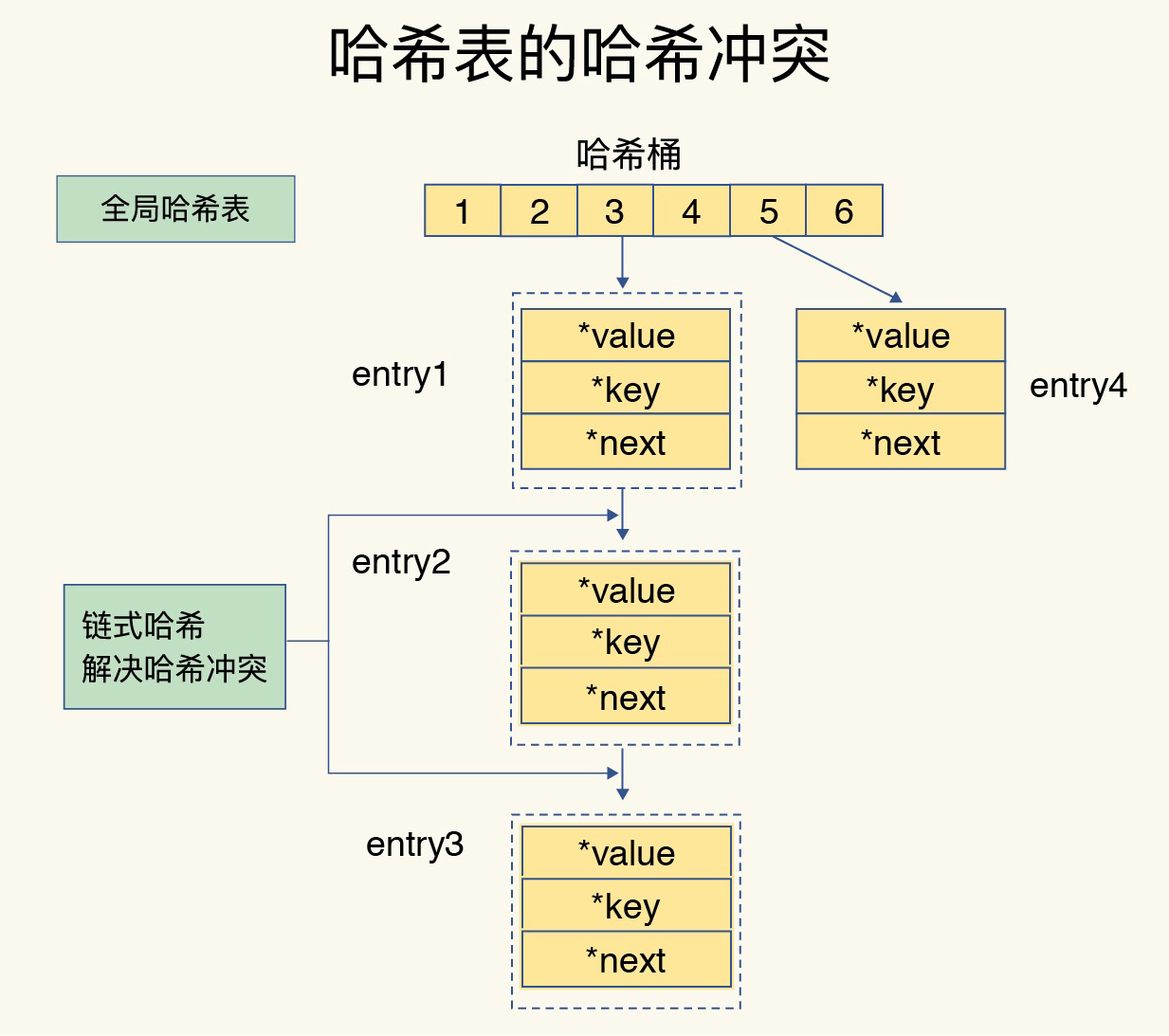


二、哈希冲突

两个 key 的哈希值和哈希桶计算对应关系时，正好落在了同一个哈希桶中。

2.1 链式哈希

同一个哈希桶中的多个元素用一个链表来保存，它们之间依次用指针连接。



哈希冲突链上的元素只能通过指针逐一查找再操作。

2.2 rehash 操作

rehash 也就是增加现有的哈希桶数量，让逐渐增多的 entry 元素能在更多的桶之间分散保存，减少单个桶中的元素数量，从而减少单个桶中的冲突。

为了使 rehash 操作更高效，Redis 默认使用了两个全局哈希表：哈希表 1 和哈希表 2。一开始，当你刚插入数据时，默认使用哈希表 1，此时的哈希表 2 并没有被分配空间。随着数据逐步增多，Redis 开始执行 rehash，这个过程分为三步：

- [ ] 给哈希表 2 分配更大的空间，例如是当前哈希表 1 大小的两倍；

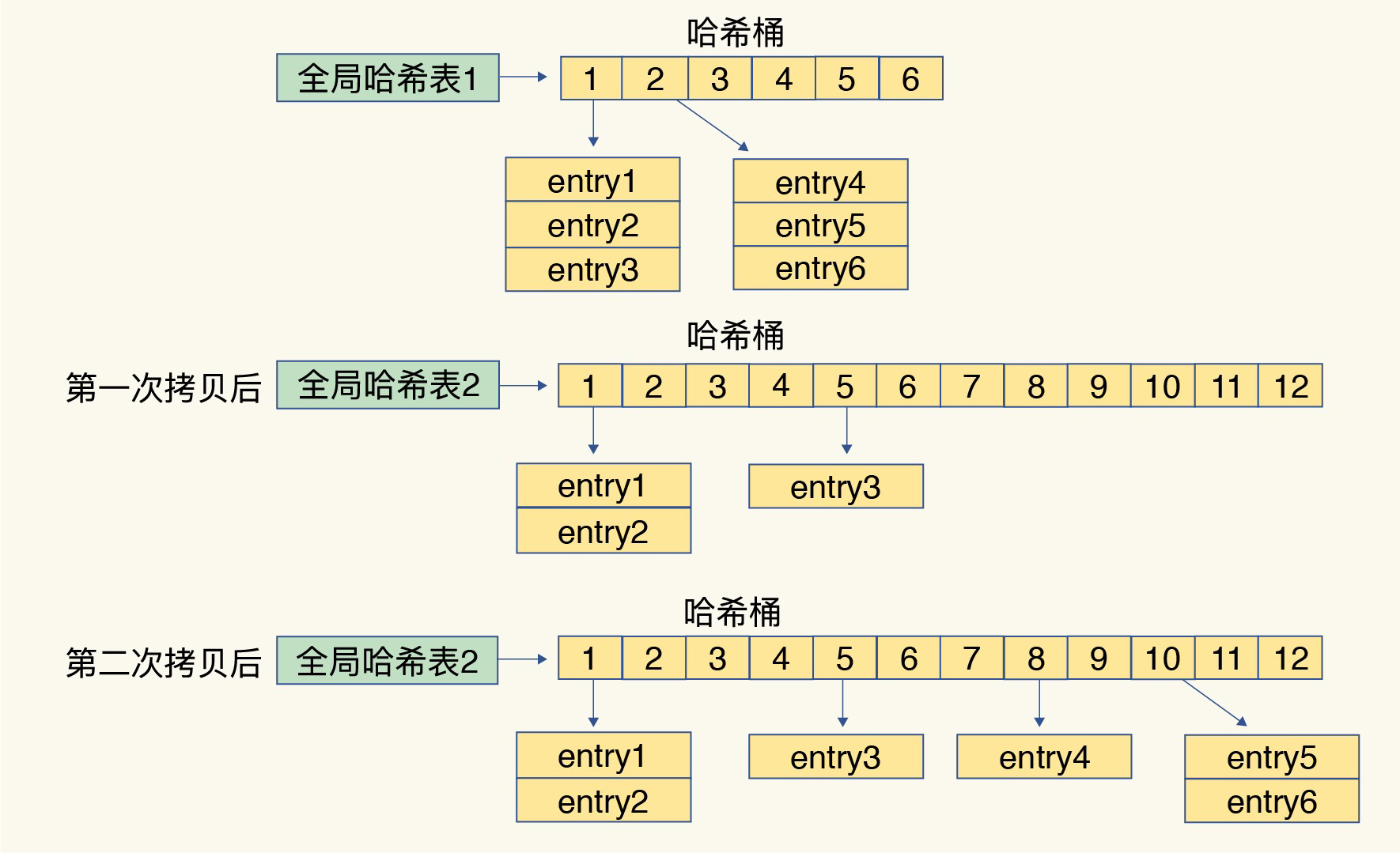
- [ ] 把哈希表 1 中的数据重新映射并拷贝到哈希表 2 中；

- [ ] 释放哈希表 1 的空间。

从哈希表 1 切换到哈希表 2，用增大的哈希表 2 保存更多数据，而原来的哈希表 1 留作下一次 rehash 扩容备用。

2.3 渐进式 rehash

第二步拷贝数据时，Redis 仍然正常处理客户端请求，每处理一个请求时，从哈希表 1 中的第一个索引位置开始，顺带着将这个索引位置上的所有 entries 拷贝到哈希表 2 中；等处理下一个请求时，再顺带拷贝哈希表 1 中的下一个索引位置的 entries。



巧妙地把一次性大量拷贝的开销，分摊到了多次处理请求的过程中，避免了耗时操作，保证了数据的快速访问。

三、底层数据结构



3.1 整数数组和双向链表

它们的操作特征都是顺序读写，也就是通过数组下标或者链表的指针逐个元素访问，操作复杂度基本是 O(N)，操作效率比较低。

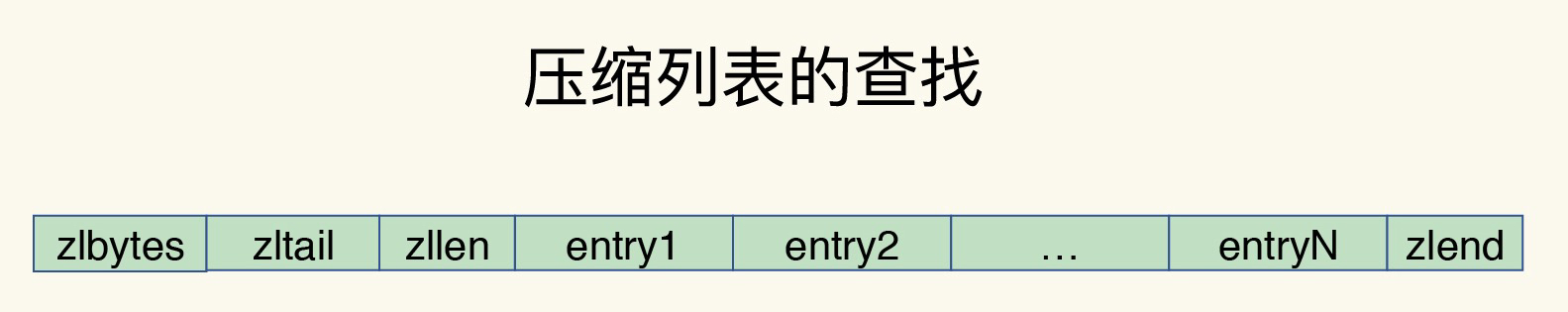
集合set采用有序数组条件：

- [ ] 存储的数据都是整数；

- [ ] 存储的数据元素个数不超过512个

3.2 压缩列表

类似于一个数组，数组中的每一个元素都对应保存一个数据。和数组不同的是，压缩列表在表头有三个字段 zlbytes、zltail 和 zllen，分别表示列表长度、列表尾的偏移量和列表中的 entry 个数；压缩列表在表尾还有一个 zlend，表示列表结束。



在压缩列表中，如果我们要查找定位第一个元素和最后一个元素，可以通过表头三个字段的长度直接定位，复杂度是 O(1)。

同时满足下面两个条件，采用压缩列表的方式实现：

- [ ] 保存的单个数据（有可能是字符串类型的）小于64字节；

- [ ] 数据个数针对List和Hash少于512个，针对SortSet少于128个。

Hash类型设置了用压缩列表保存数据时的两个阈值，一旦超过了阈值，Hash类型就会用哈希表来保存数据了。

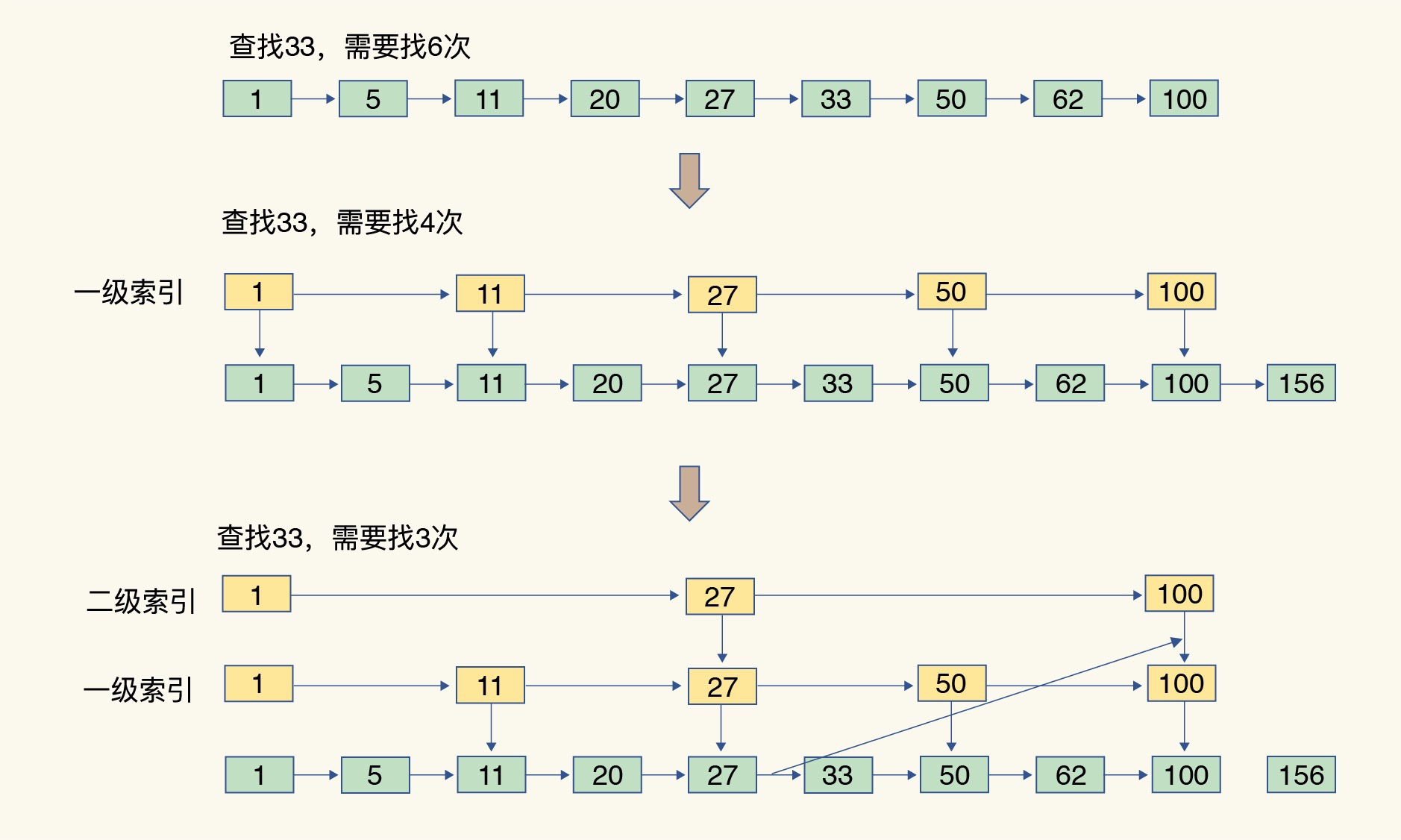
这两个阈值分别对应以下两个配置项：

hash-max-ziplist-entries：表示用压缩列表保存时哈希集合中的最大元素个数。

hash-max-ziplist-value：表示用压缩列表保存时哈希集合中单个元素的最大长度。

3.3 跳表

在链表的基础上，增加了多级索引，通过索引位置的几个跳转，实现数据的快速定位。



3.4 哈希表

- [ ] 使用MurmurHash2这种运行速度快、随机性好的哈希算法作为哈希函数。

- [ ] 对于哈希冲突问题，使用链表法来解决。

- [ ] 当数据动态增加之后，散列表的装载因子会不停地变大。为了避免散列表性能的下降，当装载因子大于1的时候，Redis会触发扩容，将散列表扩大为原来大小的2倍左右（具体值需要计算才能得到）。当数据动态减少之后，为了节省内存，当装载因子小于0.1的时候，Redis就会触发缩容，缩小为字典中数据个数的大约2倍大小（这个值也是计算得到的）。

- [ ] 使用渐进式扩容缩容策略，将数据的搬移分批进行，避免了大量数据一次性搬移导致的服务停顿。

四、不同操作的复杂度

- [ ] 单元素操作，是指每一种集合类型对单个数据实现的增删改查操作。

- [ ] 范围操作，是指集合类型中的遍历操作，可以返回集合中的所有数据。这类操作的复杂度一般是 O(N)，比较耗时。

- [ ] 统计操作，是指集合类型对集合中所有元素个数的记录。

- [ ] 例外情况，是指某些数据结构的特殊记录，例如压缩列表和双向链表都会记录表头和表尾的偏移量。这样一来，对于 List 类型的 LPOP、RPOP、LPUSH、RPUSH 这四个操作来说，它们是在列表的头尾增删元素，这就可以通过偏移量直接定位，所以它们的复杂度也只有 O(1)，可以实现快速操作。

五、小结

- [ ] Redis 中用来保存每个键和值的全局哈希表结构，也包括了支持集合类型实现的双向链表、压缩列表、整数数组、哈希表和跳表这五大底层结构。

- [ ] Redis 能快速操作键值对，一方面是因为 O(1) 复杂度的哈希表被广泛使用，包括 String、Hash 和 Set，它们的操作复杂度基本由哈希表决定，另一方面，Sorted Set 也采用了 O(logN) 复杂度的跳表。

- [ ] 集合类型的范围操作，因为要遍历底层数据结构，复杂度通常是 O(N)。建议用其他命令来替代，例如可以用 SCAN 来代替，避免在 Redis 内部产生费时的全集合遍历操作。

- [ ] 复杂度较高的 List 类型，它的两种底层实现结构：双向链表和压缩列表的操作复杂度都是 O(N)。主要用于 FIFO 队列场景，而不是作为一个可以随机读写的集合。