1. 简述redis

redis是由ANSI C语言开发的, 没有任何依赖, 支持网络, 性能优先, 可持久化的缓存数据库



在互联网应用上这5中数据结构都应用到了

string: 缓存、限流、计数器、分布式锁、分布式Session

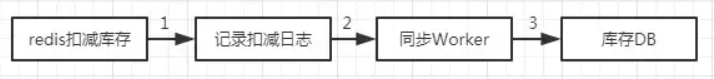
list: 关注人列表、简单的队列

hash: 存储用户信息、主页访问量、组合查询

set: 赞、踩、标签

zset: 排行榜

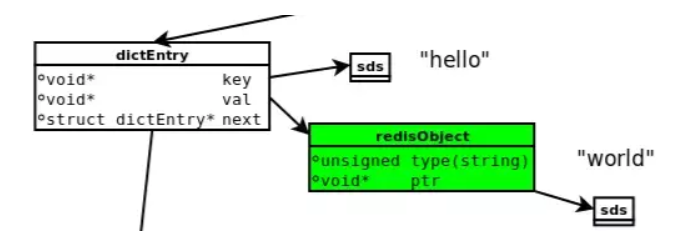
比如在电商的促销环节下, 会通过一些特殊的设计保证系统的稳定性, 如扣减库存



在同步worker时需要考虑并发处理和同步处理, 从这可以看出redis是高效稳定的

1. redis的redisObject

当执行set hello world的时候, 他的数据模型为:



dictEntry: redis给每个键值对分配一个dictEntry, 里面有key和val的指针, next指向下一个

dictEntry, 从而形成一个链表, 这个指针可以把多个hash值相同的键值对连接在 一起, 从而解决hash冲突的问题(链地址法)

sds: key键 hello 是 SDS(Simple dynamic string) 简单动态字符串类型

redisObject: val值 world是存在redisObject类型中, redis的5中类型都是以redisObject存 储的, type字段表明了这个value是什么类型的, ptr字段则指向对象的存储地址

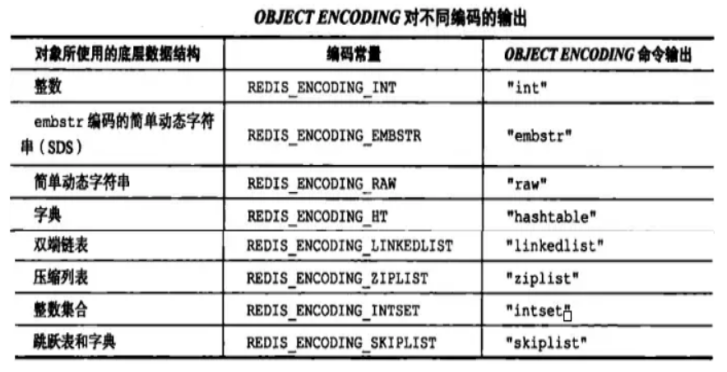
redisObject是redis的核心, redis的对象类型, 内部编码, 内存回收, 共享对象等功能都是需要redisObject支持的 这样的好处是: **针对不同的场景, 对5中基本类型设置多种不同的数据结构, 从而优化对象在不同场景下使用效率**

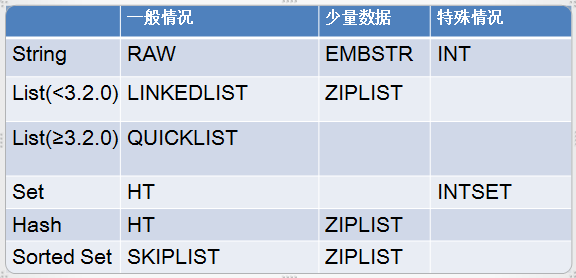
dictEntry, sds, redisObject 都是需要内存分配器 (如jemalloc)分配内存进行存储的

jemalloc是redis的默认内存分配器, 它在减小内存碎片方面做得比较好

在64位操作系统中, jemalloc将内存空间划分为 **小、大、巨大**这三个范围, 每个范围又分配了很多小的内存块单位, 当redis进行存储操作的时候, 会选择最合适的内存块进行数据的存储

每一个redis对象由redisObject结构表示, 其中的ptr指针指向了底层实现的数据结构, 而数据结构是由encoding属性决定的

type对应encoding图



1. redis的string

字符串对象底层实现可以是 int、ember、raw ember编码是通过调用一次内存分配器函数来分配一块连续的空间, raw需要调用两次

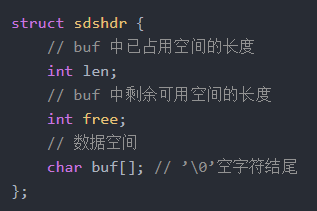
int编码和ember编码在一定情况下会转换为raw编码

ember: <= 39个字节

int : 8个字节的长整形

raw: 大于39个字节的字符串

简单动态字符串 (SDS), 这种结构像Java中的ArrayList<Character> 长度是可变的

SDS源码

get: sdsrange --- O(n)

set: sdscpy --- O(n)

len: sdslen --- O(1)

常数复杂度获取字符串长度: SDS对象中有len属性, 所以获得SDS对象的长度时间复杂度是 O(1)

预分配空间: 如果对SDS进行修改可能有两种情况

1 SDS的长度(len) 小于1MB, 那么程序分配和len属性一样大小的空间, 这时free 属性和len属性值相同

举个例子: 如SDS将len属性改为15字节, 程序也会分配15个字节给free, 这

时buf数组的实际长度变为 15+15+1 = 31字节 这里的1是 ‘\0’的空字符的字节

2 SDS的长度(len) 大于1MB, 程序会分配1MB的可用空间

举个例子: 如SDS将len属性改为10MB, 程序会分配1MB的空间给free, 这时 buf数组的实际长度变为 10MB+1MB+1byte

惰性释放空间: 当执行截取字符串操作 (sdstrim) 后, SDS不会立马释放多出来的空间,

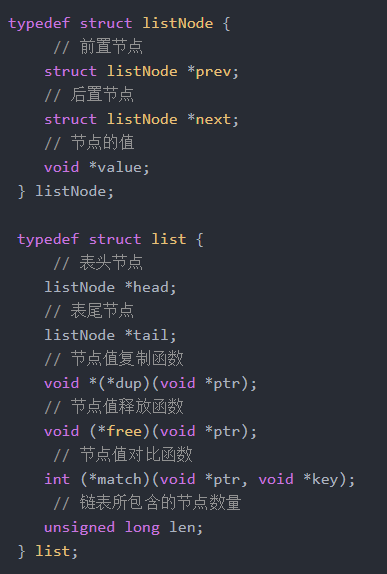
如果下次再进行拼接字符串操作, 且拼接没有刚才释放的空间大, 那些未释放的空间就派 上了用场, **惰性释放空间避免了特定情况下字符串重新分配内存的的操作**

杜绝缓冲区溢出: 使用C语言字符串操作时, 如果字符串长度增加 (strcat操作), 而忘记重新分 配内存, 则很容易造成缓冲区溢出, 由于SDS记录了长度, 相应的操作在可能造成缓冲区 溢出时会自动重新分配内存, 从而杜绝缓冲区溢出

1. redis的list

list对象底层实现是quicklist(快速队列, 是ziplist压缩队列 和 linkedlist 双向列表的组合)

list支持两边插入和弹出, 可以获得指定位置(范围)的元素, 可以充当数组, 队列, 栈



list源码

rpush: listAddNodeHead --- O(1)

lpush: listAddNodeTail --- O(1)

push: listInsertNode --- O(1)

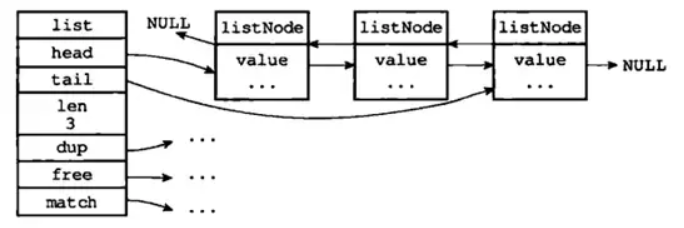
index : listIndex --- O(N)

pop: ListFirst/listLast --- O(1)

llen: listLength --- O(N)

linkedList (双端列表)

结构比较像Java中linkedList

linkedList结构图

从linkedList图中可以看到redis 双端列表list的节点带有 prev指针、next指针、head指针

和tail指针 **获取前节点、后节点、头部节点、尾部节点他们的复杂度都是O(1), len的复杂度也是O(1)**

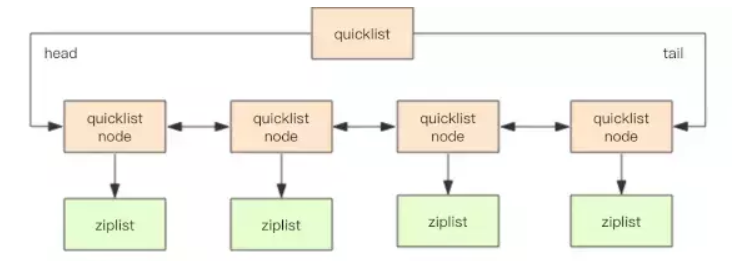
双端列表与压缩列表相比, 压缩列表可以节省空间, 但是增删操作的复杂度太高, 当节点较少时可以使用压缩列表, 当节点很多时, 还是使用双端列表好

ziplist (压缩列表)

当一个列表包含少量元素时, 并且是小数整数或比较短的字符串时, 那么redis就用 ziplist 作为

list的底层实现

ziplist是为了节约内存而开发的, 是由一系列特殊编码的连续内存块 (而不像双端列表每个节点都是指针) 组成的顺序型数据结构, 具体结构比较复杂, 在新版本的redis中 使用quicklist替代了linkedList以及ziplist



quickList结构图

quickList是 linkedList和zipList的混合体, 它将linkedList按断切分, 每一段使用zipList来紧凑储存, 多个zipList之间使用双向指针串起来, 因为链表的附加空间太高, 每一个prev和next指针都要占16个字节(64位系统8个字节), 另外每个节点的内存都是独立分配的, 这样会加剧内存的碎片化, 影响内存的管理效率

quickList的默认压缩深度为0, 也就是说不压缩, 为了支持快速的pop/push操作, quickList的首尾两个zipList不压缩, 此时深度为1, 为了进一步的节省空间, redis还会对zipList进行压缩储存, **使用LZF压缩算法**

1. redis的hash

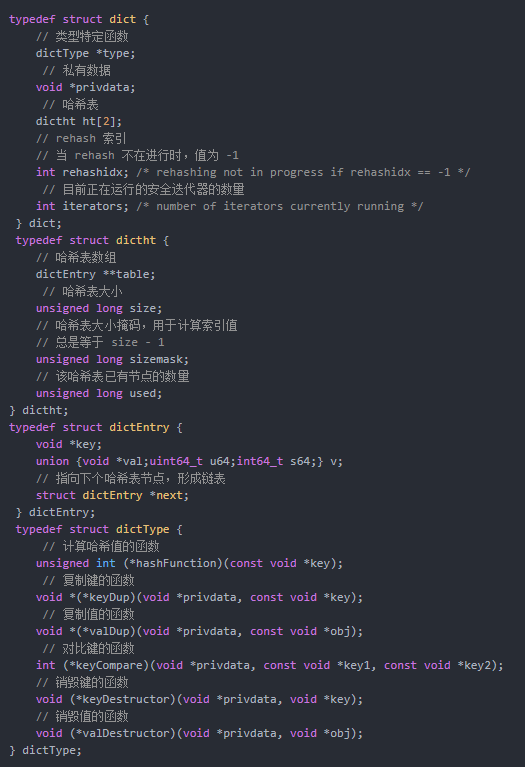
hash的底层对象可以是zipList (压缩列表), 也可以是hashTable (哈希表)

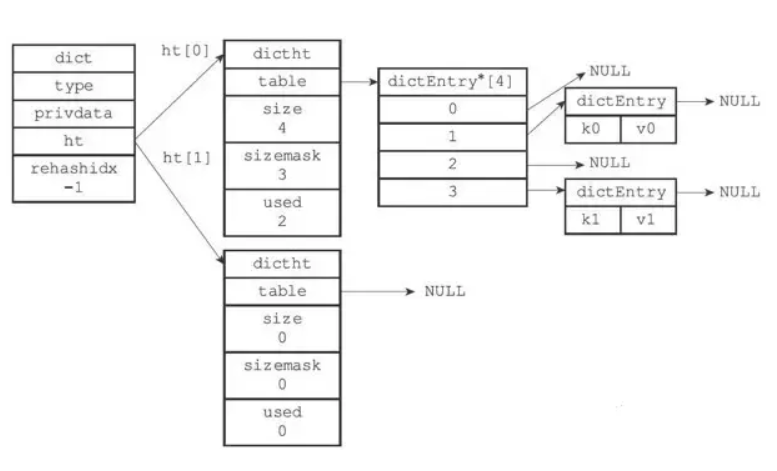
当满足这两个条件时, 才会使用zipList:

1 哈希表中的数据小于512个

2 哈希表中的每一个键值对的长都小于64个字节

hashTable哈希表可以实现O(1)复杂度的读写操作, 一次效率非常高

hashTable源码



hashTable结构图

hash的结构类似于JDK7之前的hashMap, 当有两个或两个以上的键被分到哈希数组的同一个索引上, 会产生hash冲突, redis用链地址法来解决键冲突 ,每个哈希表节点都有一个next指针,

多个hash表节点使用next指针构成单项链表, 链地址法就是将哈希冲突的对象组织成一个单项链表, 放在hash值对应的槽上

redis 的hash 使用 hashTable作为底层实现的话, 每个hash都会有两个hash表, 一个平时使

用, 另一个仅在rehash (从新散列) 时使用, 随着对hash表的操作, key会逐渐的增多或者减少,

为了让hash表的负载因子维持在一个合理的范围内, redis会对hash表的大小进行扩张或收缩

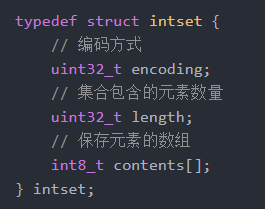
(rehash), 也就是将 ht [0] 的所有的键值进行多次的, 渐进式的rehash 到 ht [1] 里

1. redis的set

set集合的底层实现可以是intset (整数集合) 或hashTable (哈希表)

intset (整数集合) 当一个集合只包含整数的时候, 并且元素不多的时候会使用 intset, 作为set

集合的底层实现



intset源码

sadd: intsetAdd --- O(1)

smembers: intsetGetO(1) --- O(N)

srem: intsetRemove --- O(N)

slen: intsetlen --- O(1)

intset底层实现为有序, 无重复数组保存元素, intset这个结构里的整数数组的类型可以是16位

的, 32位, 64位 如果这个集合里的元素都是16位长度的, 添加一个17为长度的元素, 那么这

整个集合会从16位升级成32位的数组, **升级可以提升intset的灵活性, 还可以节约内存, 但是 不可逆 (32位 可以变为64位, 但不能变成16位)**

1. redis的zset

zset的底层实现可以是zipList (压缩列表)也可以是skipList (跳跃表)

当一个有序集合的元素比较多或者成员是比较长的字符串时, redis就会使用skipList (跳跃表) 作为zset的底层实现



skipList源码

zadd: zslinsert

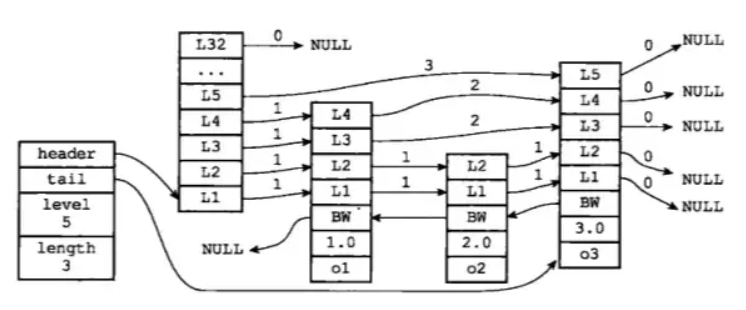
平均O(logN), 最坏O(N)

zrem: zsldelete

平均O(logN), 最坏O(N)

zrank: zslGetRank

平均O(logN), 最坏O(N)



skipList结构图

skipList的查找复杂度是logN, 可以和平衡二叉树相当, 实现起来比较简单

skipList是一种有序数据结构, 他通过在某个节点中维护多个指向其他节点的指针, 从而达到快速访问的目的