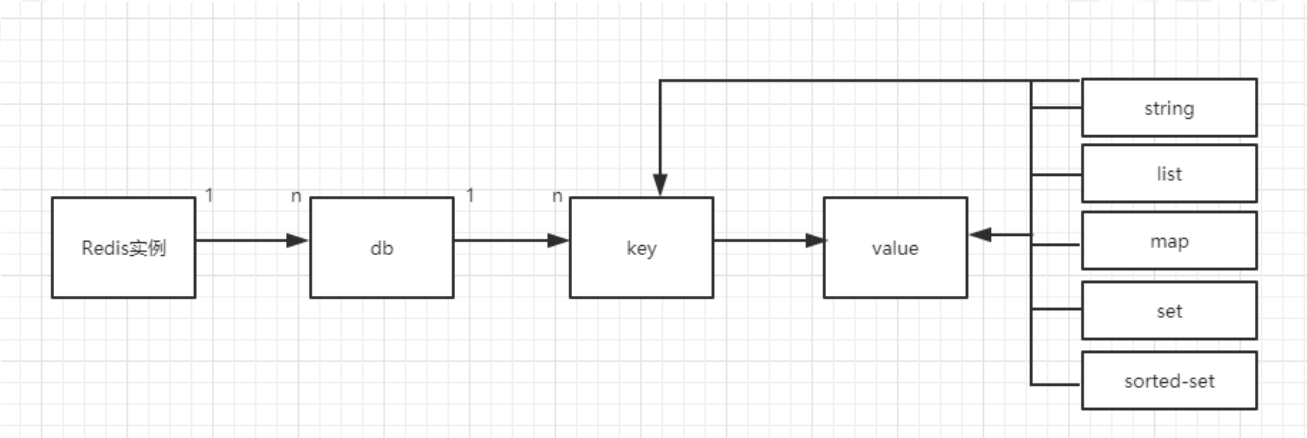
Redis的魅力

缓存大致可以分为两类，一种是应用内缓存，比如Map(简单的数据结构)，以及EH Cache(Java第三方库)，另一种就是缓存组件，比如Memached，Redis；Redis（remote dictionary server）是一个基于KEY-VALUE的高性能的存储系统，通过提供多种键值数据类型来适应不同场景下的缓存与存储需求

存储结构

大家一定对字典类型的数据结构非常熟悉，比如map ，通过key value的方式存储的结构。 redis的全称是remote dictionary server(远程字典服务器)，它以字典结构存储数据，并允许其他应用通过TCP协议读写字典中的内容。数据结构如下



Redis的安装

redis约定次版本号（第一个小数点后的数字）为偶数版本是稳定版，如2.8、3.0， 奇数版本为非稳定版，生产环境需要使用稳定版；目前最新版本为Redis4.0.9, 我们本次课仍然以3.2作为演示版本

安装配置

\1. 下载redis的安装包

\2. tar -zxvf 解压

\3. cd 到解压后的目录

\4. 执行make 完成编译

可能会遇到的错误

```
1. 需要安装tcl  yum install tcl 、 yum install gcc
2. error: jemalloc/jemalloc.h: No such file or directory
说关于分配器allocator， 如果有MALLOC  这个 环境变量， 会有用这个环境变量的 去建立Redis。
而且libc 并不是默认的 分配器， 默认的是 jemalloc，因为 jemalloc 被证明 有更少的 fragmentation
problems 比libc。
但是如果你又没有jemalloc 而只有 libc 当然 make 出错。 所以加这么一个参数。
解决办法
make MALLOC=libc
```

\5. make test 测试编译状态

\6. make install {PREFIX=/path}

启动停止redis

安装完redis后的下一步就是怎么去启动和访问，我们首先先了解一下Redis包含哪些可执行文件

Redis-server	**Redis服务器**
Redis-cli	Redis命令行客户端
Redis-benchmark	Redis性能测试工具
Redis-check-aof	Aof文件修复工具
Redis-check-dump	Rdb文件检查工具
Redis-sentinel	Sentinel服务器（2.8以后）

我们常用的命令是redis-server和redis-cli

\1. 直接启动

redis-server ../redis.conf

服务器启动后默认使用的是6379的端口，通过--port可以自定义端口； 6379在手机键盘上MERZ对应，MERZ是一名意大利歌女的名字

Redis-server --port 6380

以守护进程的方式启动，需要修改redis.conf配置文件中daemonize yes

\2. 停止redis

redis-cli SHUTDOWN

考虑到redis有可能正在将内存的数据同步到硬盘中，强行终止redis进程可能会导致数据丢失，正确停止redis的方式应该是向Redis发送SHUTDOW命令

当redis收到SHUTDOWN命令后，会先断开所有客户端连接，然后根据配置执行持久化，最终完成退出

数据类型

字符串类型

字符串类型是redis中最基本的数据类型，它能存储任何形式的字符串，包括二进制数据。你可以用它存储用户的邮箱、json化的对象甚至是图片。一个字符类型键允许存储的最大容量是512M

内部数据结构

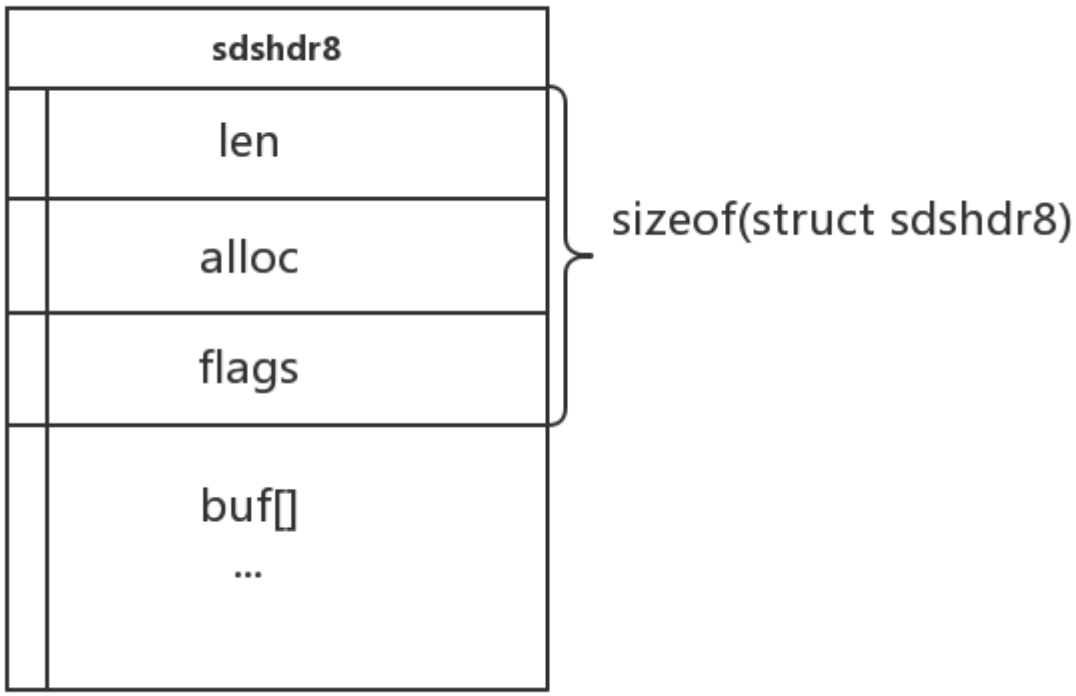
在Redis内部，String类型通过 int、SDS(simple dynamic string)作为结构存储，int用来存放整型数据，sds存放字节/字符串和浮点型数据。在C的标准字符串结构下进行了封装，用来提升基本操作的性能，同时也充分利用已有的C的标准库，简化实现逻辑。我们可以在redis的源码中【sds.h】中看到sds的结构如下；

typedef char *sds;

redis3.2分支引入了五种sdshdr类型，目的是为了满足不同长度字符串可以使用不同大小的Header，从而节省内存，每次在创建一个sds时根据sds的实际长度判断应该选择什么类型的sdshdr，不同类型的sdshdr占用的内存空间不同。这样细分一下可以省去很多不必要的内存开销，下面是3.2的sdshdr定义

```
`struct __attribute__ ((__packed__)) sdshdr8 { 8表示字符串最大长度是2^8-1 (长度为255) `` uint8_t len;//表示当前sds的长度(单位是字节)``  uint8_t alloc; //表示已为sds分配的内存大小(单位是字节)``  unsigned char flags; //用一个字节表示当前sdshdr的类型，因为有sdshdr有五种类型，所以至少需要3位来表示000:sdshdr5，001:sdshdr8，010:sdshdr16，011:sdshdr32，100:sdshdr64。高5位用不到所以都为0。``  char buf[];//sds实际存放的位置``};`
```

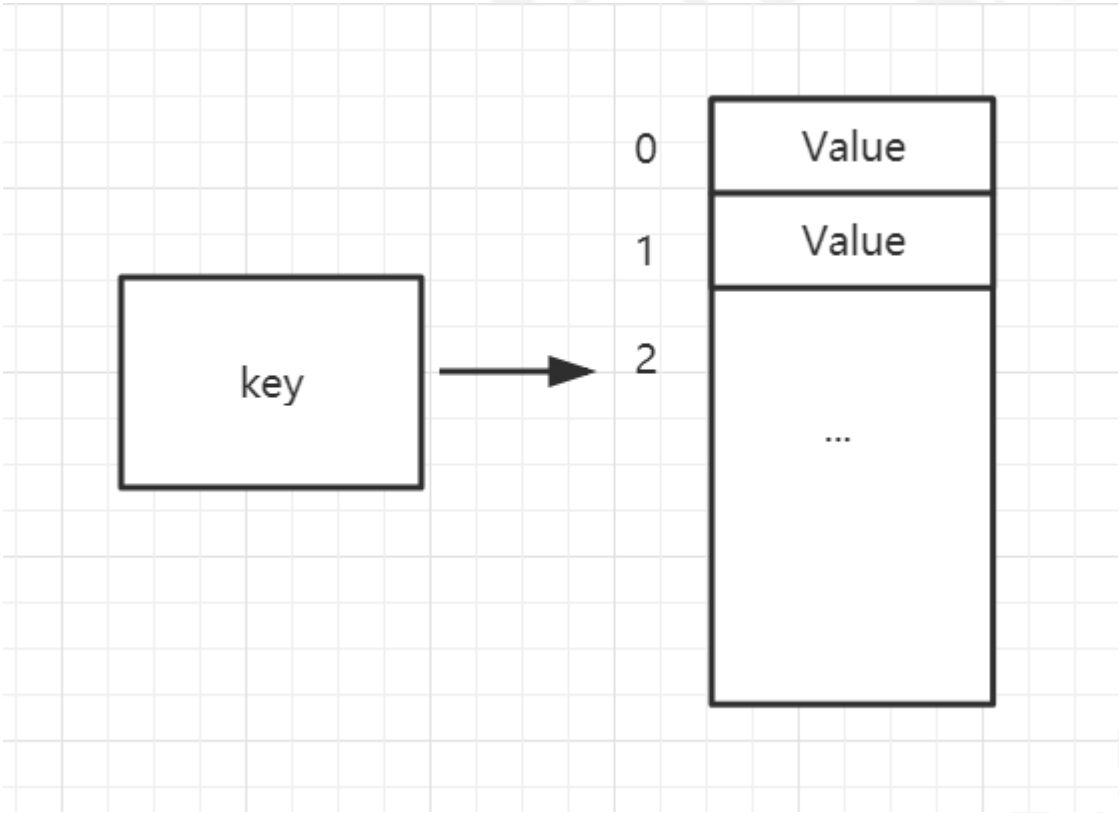
sdshdr8的内存布局



列表类型

列表类型(list)可以存储一个有序的字符串列表，常用的操作是向列表两端添加元素或者获得列表的某一个片段。

列表类型内部使用双向链表实现，所以向列表两端添加元素的时间复杂度为O(1), 获取越接近两端的元素速度就越快。这意味着即使是一个有几千万个元素的列表，获取头部或尾部的10条记录也是很快的



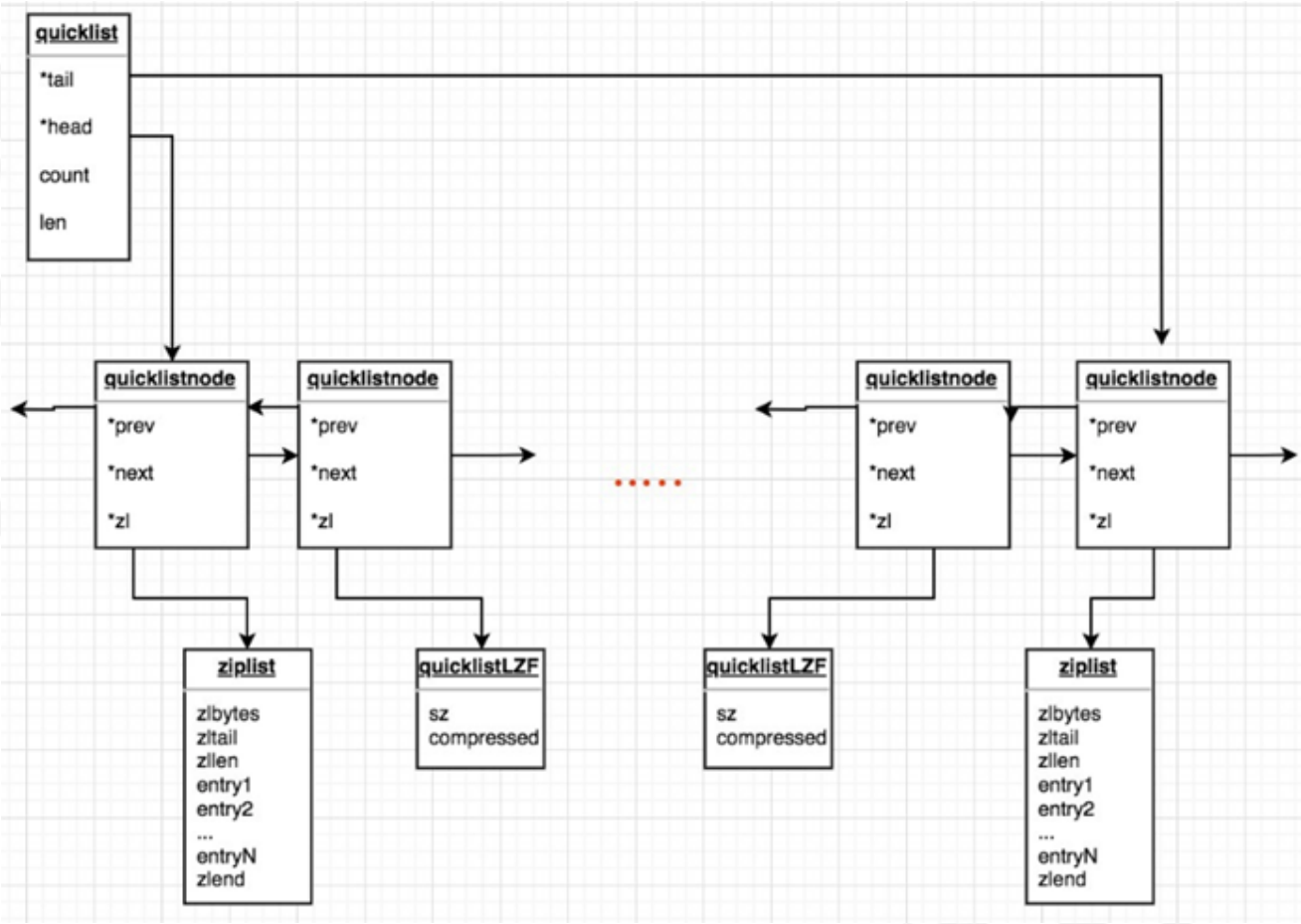
内部数据结构

redis3.2之前，List类型的value对象内部以linkedlist或者ziplist来实现, 当list的元素个数和单个元素的长度比较小的时候，Redis会采用ziplist（压缩列表）来实现来减少内存占用。否则就会采用linkedlist（双向链表）结构。

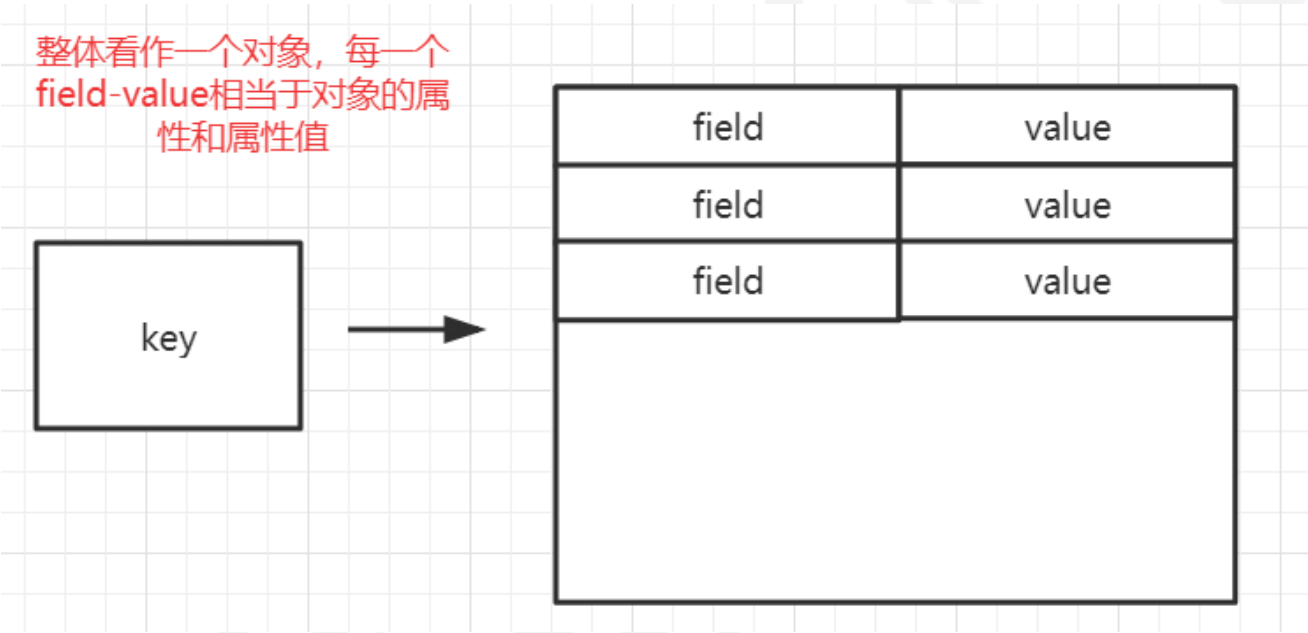
redis3.2之后，采用的一种叫quicklist的数据结构来存储list，列表的底层都由quicklist实现。

这两种存储方式都有优缺点，双向链表在链表两端进行push和pop操作，在插入节点上复杂度比较低，但是内存开销比较大； ziplist存储在一段连续的内存上，所以存储效率很高，但是插入和删除都需要频繁申请和释放内存；

quicklist仍然是一个双向链表，只是列表的每个节点都是一个ziplist，其实就是linkedlist和ziplist的结合，quicklist中每个节点ziplist都能够存储多个数据元素，在源码中的文件为【quicklist.c】，在源码第一行中有解释为：A doubly linked list of ziplists意思为一个由ziplist组成的双向链表；



hash类型



数据结构

map提供两种结构来存储，一种是hashtable、另一种是前面讲的ziplist，数据量小的时候用ziplist. 在redis中，哈希表分为三层，分别是，源码地址【**dict.h**】

dictEntry

管理一个key-value，同时保留同一个桶中相邻元素的指针，用来维护哈希桶的内部链；

```
typedef struct dictEntry {
    void *key;
    union {  //因为value有多种类型，所以value用了union来存储
        void *val;
        uint64_t u64;
        int64_t s64;
        double d;
    } v;
    struct dictEntry *next;//下一个节点的地址，用来处理碰撞，所有分配到同一索引的元素通过next指针链接起来形成链表key和v都可以保存多种类型的数据
} dictEntry;
```

dictht

实现一个hash表会使用一个buckets存放dictEntry的地址，一般情况下通过hash(key)%len得到的值就是buckets的索引，这个值决定了我们要将此dictEntry节点放入buckets的哪个索引里,这个buckets实际上就是我们说的hash表。dict.h的dictht结构中table存放的就是buckets的地址

```
typedef struct dictht {
    dictEntry **table;//buckets的地址
    unsigned long size;//buckets的大小,总保持为 2^n
    unsigned long sizemask;//掩码，用来计算hash值对应的buckets索引
    unsigned long used;//当前dictht有多少个dictEntry节点
} dictht;
```

dict

dictht实际上就是hash表的核心，但是只有一个dictht还不够，比如rehash、遍历hash等操作，所以redis定义了一个叫dict的结构以支持字典的各种操作，当dictht需要扩容/缩容时，用来管理dictht的迁移，以下是它的数据结构,源码在

```
typedef struct dict {
    dictType *type;//dictType里存放的是一堆工具函数的函数指针，
    void *privdata;//保存type中的某些函数需要作为参数的数据
    dictht ht[2];//两个dictht，ht[0]平时用，ht[1] rehash时用
    long rehashidx; //当前rehash到buckets的哪个索引，-1时表示非rehash状态
    int iterators; //安全迭代器的计数。
} dict;
```

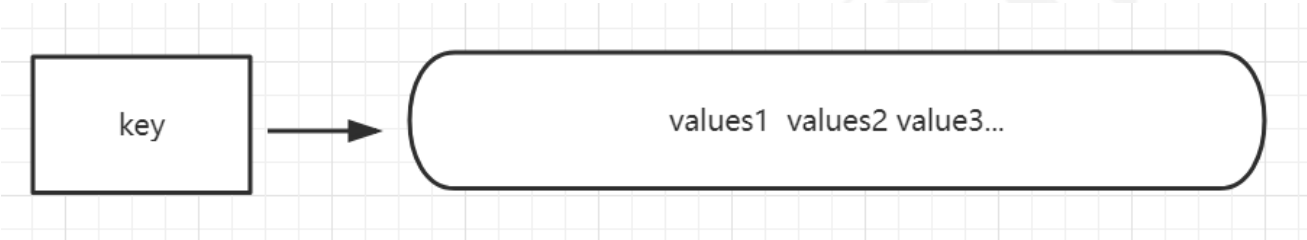
比如我们要讲一个数据存储到hash表中，那么会先通过murmur计算key对应的hashcode，然后根据hashcode取模得到bucket的位置，再插入到链表中

img

集合类型

集合类型中，每个元素都是不同的，也就是不能有重复数据，同时集合类型中的数据是无序的。一个集合类型键可以存储至多232-1个 。集合类型和列表类型的最大的区别是有序性和唯一性

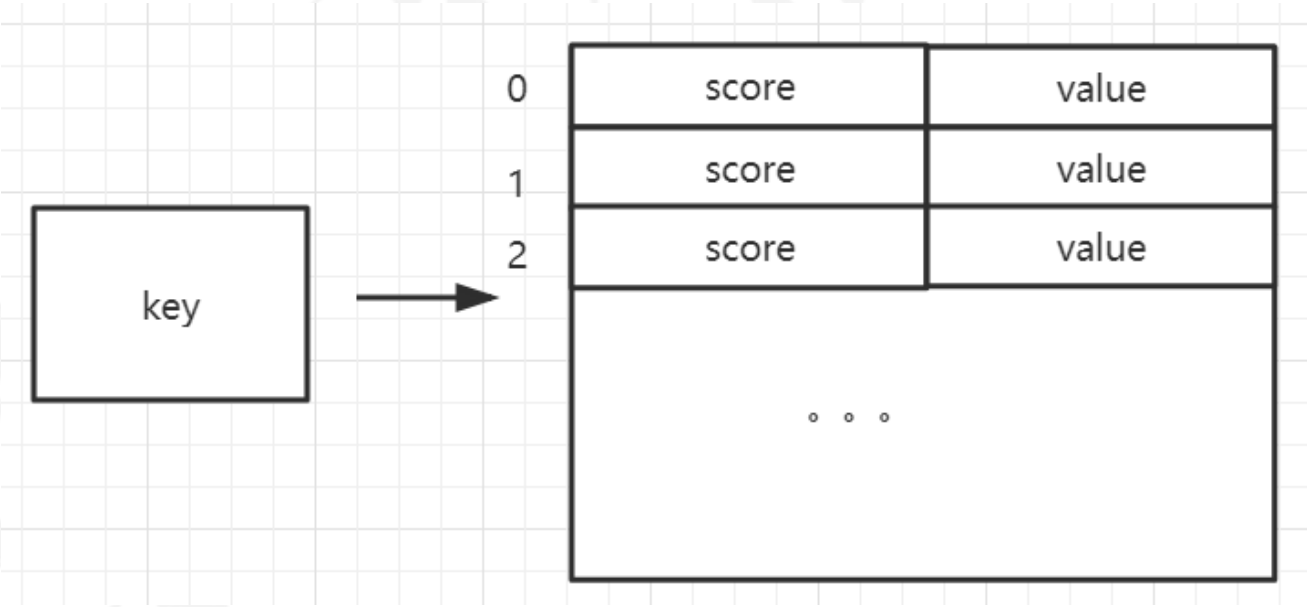
集合类型的常用操作是向集合中加入或删除元素、判断某个元素是否存在。由于集合类型在redis内部是使用的值为空的散列表(hash table)，所以这些操作的时间复杂度都是O(1).



数据结构

Set在的底层数据结构以intset或者hashtable来存储。当set中只包含整数型的元素时，采用intset来存储，否则，采用hashtable存储，但是对于set来说，该hashtable的value值用于为NULL。通过key来存储元素

有序集合

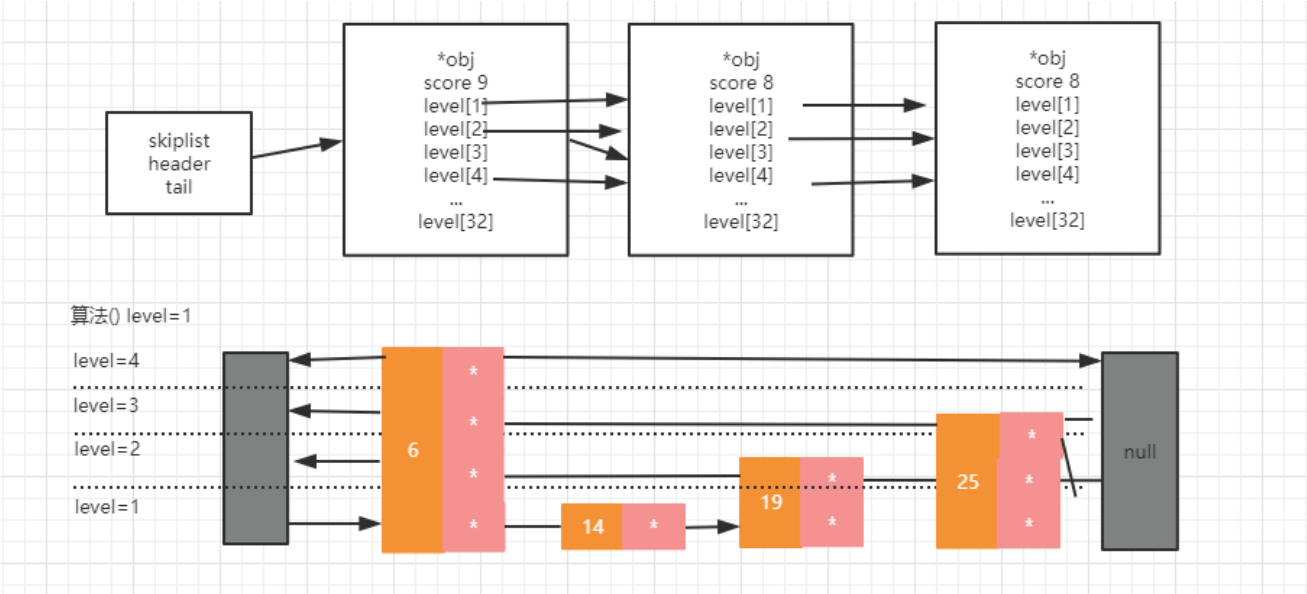


有序集合类型，顾名思义，和前面讲的集合类型的区别就是多了有序的功能

在集合类型的基础上，有序集合类型为集合中的每个元素都关联了一个分数，这使得我们不仅可以完成插入、删除和判断元素是否存在等集合类型支持的操作，还能获得分数最高(或最低)的前N个元素、获得指定分数范围内的元素等与分数有关的操作。虽然集合中每个元素都是不同的，但是他们的分数却可以相同

数据结构

zset类型的数据结构就比较复杂一点，内部是以ziplist或者skiplist+hashtable来实现，这里面最核心的一个结构就是skiplist，也就是跳跃表


skiplist
header
tail
*obj
score 9
level[1]
level[2]
level[3]
level[4]
...
level[32]
*obj
score 8
level[1]
level[2]
level[3]
level[4]
...
level[32]
*obj
score 8
level[1]
level[2]
level[3]
level[4]
...
level[32]
算法() level=1
level=4
level=3
level=2
level=1
6
14
19
25
null