redis底层数据结构主要有：

1. 简单动态字符串
2. 链表
3. 字典
4. 跳跃表
5. 整数集合
6. 压缩列表
7. 对象

1. 简单动态字符串（SDS）

如SET msg "Redis"

2.2 SDS 的定义

Redis 中定义动态字符串的结构：

```
/*
 * 保存字符串对象的结构
 */
struct sdshdr {

    // buf 中已占用空间的长度
    int len;

    // buf 中剩余可用空间的长度
    int free;

    // 数据空间
    char buf[];
};
```

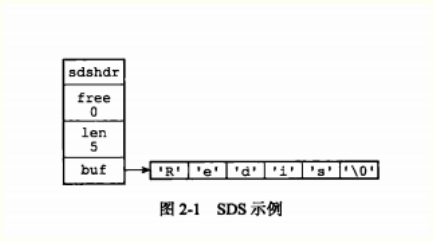


图 2-1 SDS 示例

1、len 变量，用于记录buf 中已经使用的空间长度（这里指出Redis 的长度为5）

2、free 变量，用于记录buf 中还空余的空间（初次分配空间，一般没有空余，在对字符串修改的时候，会有剩余空间出现）

3、buf 字符数组，用于记录我们的字符串（记录Redis）

当在字符串拼接时，预先会检查SDS空间是否足够，如果不够，会先拓展SDS的空间，然后再执行拼接操作。

2. 链表

链表提供高效了节点重排能力，以及顺序性的节点访问方式，并且可以通过增删节点来灵活地调整链表的长度。

链表键的底层实现之一就是链表。当一个列表键包含了数量较多的元素，又或者列表中包含的元素都是较长的字符串时，redis就会使用链表作为列表键的底层实现。

3.2 链表的数据结构

每个链表节点使用一个 **listNode**结构表示（adlist.h/listNode）：

```
typedef struct listNode{
      struct listNode *prev;
      struct listNode * next;
      void * value;
}
```

多个链表节点组成的双端链表：

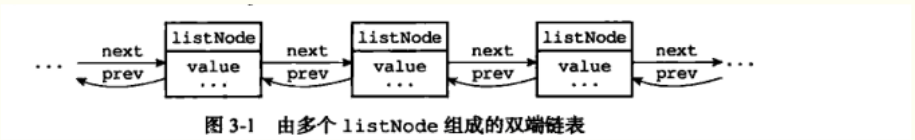


图 3-1　由多个 listNode 组成的双端链表

我们可以通过直接操作**list** 来操作链表会更加方便：

、

3. 字典

又称符号表，关联数组或映射，是一种用于保存键值对的抽象数据结构。

在字典中，一个键(key)可以和一个值进行关联，字典中的每个键都是独一无二的。其实就是hash表

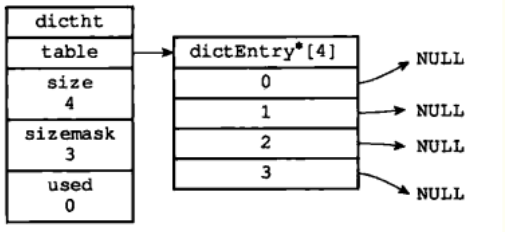
4.2.1 哈希表

Redis 字典所使用的哈希表由 dict.h/dictht 结构定义：

```
typedef struct dictht {
   //哈希表数组
   dictEntry **table;
   //哈希表大小
   unsigned long size;

   //哈希表大小掩码，用于计算索引值
   unsigned long sizemask;
   //该哈希表已有节点的数量
   unsigned long used;
}
```

一个空的字典的结构图如下：



我们可以看到，在结构中存有指向dictEntry 数组的指针，而我们用来存储数据的空间既是dictEntry

4. 跳跃表

跳跃表是一种有序数据结构，它通过在每个节点中维持多个指向其他节点的指针，从而达到快速访问节点的目的。跳跃表是一种随机化的数据，跳跃表以有序的方式在层次化的链表中保存元素，效率和平衡树媲美--查找、删除、添加等操作都可以在对数期望时间下完成，并且比起平衡树来说，跳跃表的实现要简单直观得多。

redis只在两个地方用到了跳跃表，一个是实现有序集合键，另外一个是在集群节点中用作内部数据结构。

5.2 跳跃表的定义

我们先来看一下一整个跳跃表的完整结构：

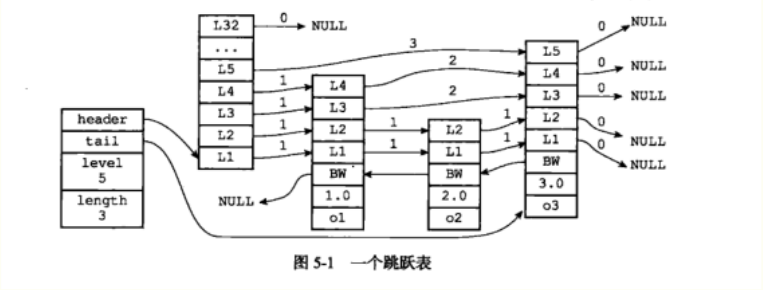


图 5-1 一个跳跃表

Redis 的跳跃表 主要由两部分组成：zskiplist（链表）和zskiplistNode （节点）

5.2.1 zskiplistNode（节点） 数据结构：

```
typedef struct zskiplistNode{
     //层
     struct zskiplistLevel{
        //前进指针
        struct zskiplistNode *forward;
       //跨度
        unsigned int span;
    } level[];
   //后退指针
    struct zskiplistNode *backward;
   //分值
    double score;
   //成员对象
    robj *obj;
}
```

5.2. zskiplist 数据结构：

```
typedef struct zskiplist {
    //表头节点和表尾节点
    structz skiplistNode *header,*tail;
    //表中节点数量
    unsigned long length;
    //表中层数最大的节点的层数
    int level;

}zskiplist;
```

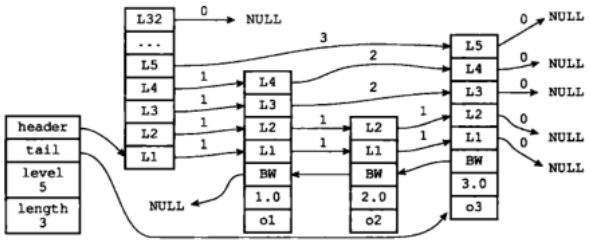


图 5-9　带有 zskiplist 结构的跳跃表

从结构图中我们可以清晰的看到，header，tail分别指向跳跃表的头结点和尾节点。level 用于记录最大的层数，length 用于记录我们的节点数量。

5.3 总结

- 跳跃表是有序集合的底层实现之一
- 主要有zskiplist 和zskiplistNode两个结构组成
- 每个跳跃表节点的层高都是1至32之间的随机数
- 在同一个跳跃表中，多个节点可以包含相同的分值，但每个节点的对象必须是唯一的
- 节点按照分值的大小从大到小排序，如果分值相同，则按成员对象大小排序

6. 整数集合

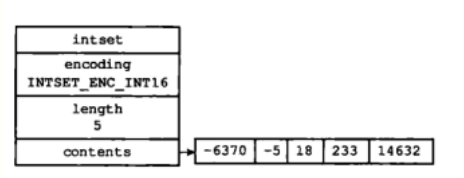
在redis中对整数集合的定义是：整数集合是集合键的底层实现之一，当一个集合中只包含整数，且这个集合的元素数量不多时，redis就会使用整数集合insert作为集合的底层实现。

即就是整数集合是一个特殊集合，里面存储的数据只能够是整数，并且数据量不能过大。

```
typedef struct intset{
    //编码方式
    uint32_t enconding;
   // 集合包含的元素数量
    uint32_t length;
    //保存元素的数组
    int8_t contents[];

}
```

我们观察一下一个完成的整数集合结构图：



1、encoding：用于定义整数集合的编码方式

2、length：用于记录整数集合中变量的数量

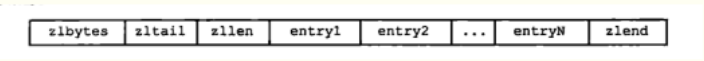
3、contents：用于保存元素的数组，虽然我们在数据结构图中看到，intset将数组定义为int8_t，但实际上数组保存的元素类型取决于encoding

7. 压缩列表

压缩列表是列表键和哈希键的底层实现之一。当一个列表键只含有少量列表项，并且每个列表项要么就是小整数，要么就是长度比较短的字符串，那么redis就会使用压缩列表作为列表键的底层实现。

7.2 压缩列表的构成

一个压缩列表的组成如下：



1、zlbytes:用于记录整个压缩列表占用的内存字节数

2、zltail：记录要列表尾节点距离压缩列表的起始地址有多少字节

3、zllen：记录了压缩列表包含的节点数量。

4、entryX：要说列表包含的各个节点

5、zlend：用于标记压缩列表的末端

属性	类型	长度	用　途
zlbytes	uint32_t	4字节	记录整个压缩列表占用的内存字节数：在对压缩列表进行内存重分配，或者计算 zlend 的位置时使用
zltail	uint32_t	4字节	记录压缩列表表尾节点距离压缩列表的起始地址有多少字节：通过这个偏移量，程序无须遍历整个压缩列表就可以确定表尾节点的地址
zllen	uint16_t	2字节	记录了压缩列表包含的节点数量：当这个属性的值小于 UINT16_MAX（65535）时，这个属性的值就是压缩列表包含节点的数量；当这个值等于 UINT16_MAX 时，节点的真实数量需要遍历整个压缩列表才能计算得出
entryX	列表节点	不定	压缩列表包含的各个节点，节点的长度由节点保存的内容决定
zlend	uint8_t	1字节	特殊值 0xFF（十进制 255），用于标记压缩列表的末端