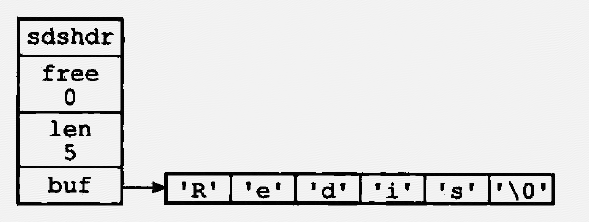
Redis设计与实现

## 第一部分：数据结构与对象

### 第二章：简单动态字符串（Simple Dynamic String）

Redis的默认字符串表示为SDS，即简单动态字符串。结构图为：



Len记录了SDS中记录的字符串的长度：查询长度的时间复杂度为O(1)。

buf记录了SDS中存放的数据；buf中的实际长度为len+free+1byte。

free表示了SDS中未使用的空间。实现了空间预分配和惰性空间释放两种策略。

1.空间预分配：

优化了SDS的字符串增长操作，减少了字符串增长带来的内存重分配次数。

2.惰性空间释放：

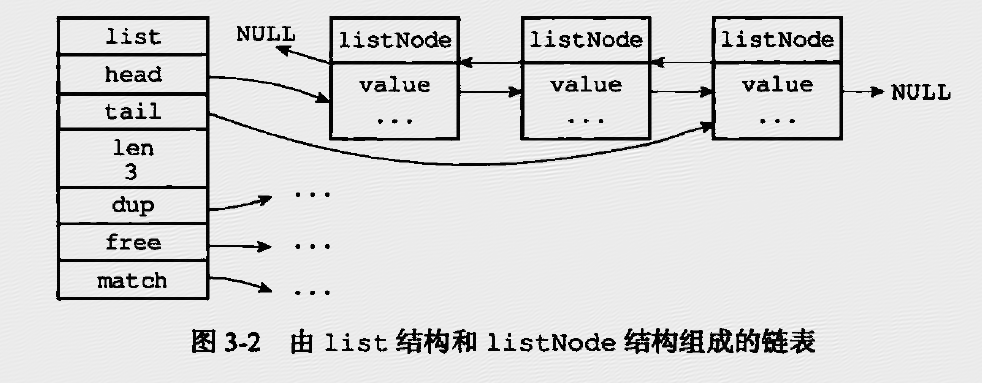
优化了SDS的字符串缩短操作，减少了字符串缩减带来的内存重分配次数。

### 第三章：链表 list

<http://redisdoc.com/list/index.html>

链表提供了高效的节点重排能力，以及顺序性的节点访问方式，并且可以通过增删节点灵活的调整链表的长度。

链表在redis中应用举例：列表键的底层实现之一就是链表：当一个列表键包含了数量比较多的元素，或者列表中包含的元素都是比较长的字符串时，redis就会使用链表作为列表键的底层实现。发布与订阅，慢查询，监视器等功能也用到了链表，redis服务器使用链表保存多个客户端的状态信息，以及使用链表来构建客户端输出缓冲区。



Redis链表的实现特点：

1.双端：链表节点带有prev和next指针，获取某个节点的前置节点和后置节点复杂度都是O(1)

2.无环：表头节点的prev和表位节点的next指针都指向null，所以对链表的访问都是nulll为终点

3.带表头指针和表尾指针：通过list结构的head和tail指针，获取链表的表头节点和表尾节点的复杂度都为O(1)

4.带链表长度计数器：程序可以使用list的len属性直接获取链表长度，复杂度O(1)

### 第四章：字典 map

<http://redisdoc.com/hash/index.html>

字典，又称为符号表（symbol table），关联数组（associative array）或映射（map），是用于保存键值对的抽象数据结构。

字典在redis中应用广泛，举例：

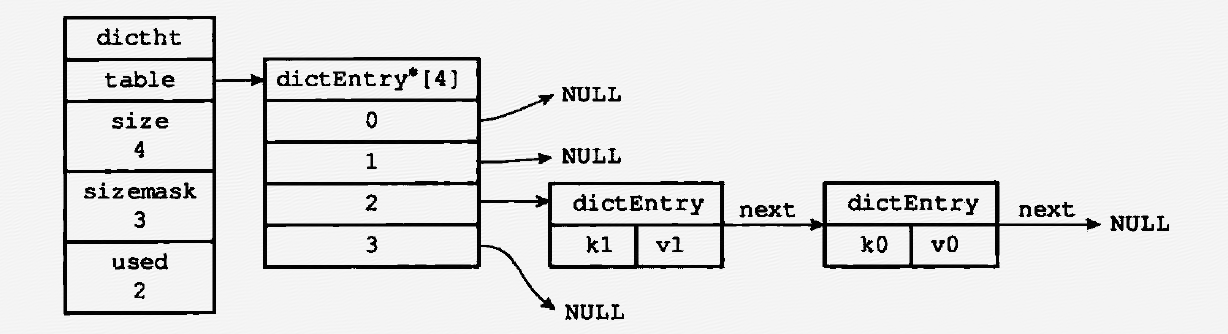
1).redis的数据库就是使用字典来作为底层实现的，对数据库的增删改查操作也是构建在对字典的操作之上

2).字典还是哈希键的底层实现之一，当一个哈希键包含的键值对比较多，或者键值对中的元素都是比较长的字符串时，redis就会使用字典作为哈希键的底层实现。

4.1 字典的实现

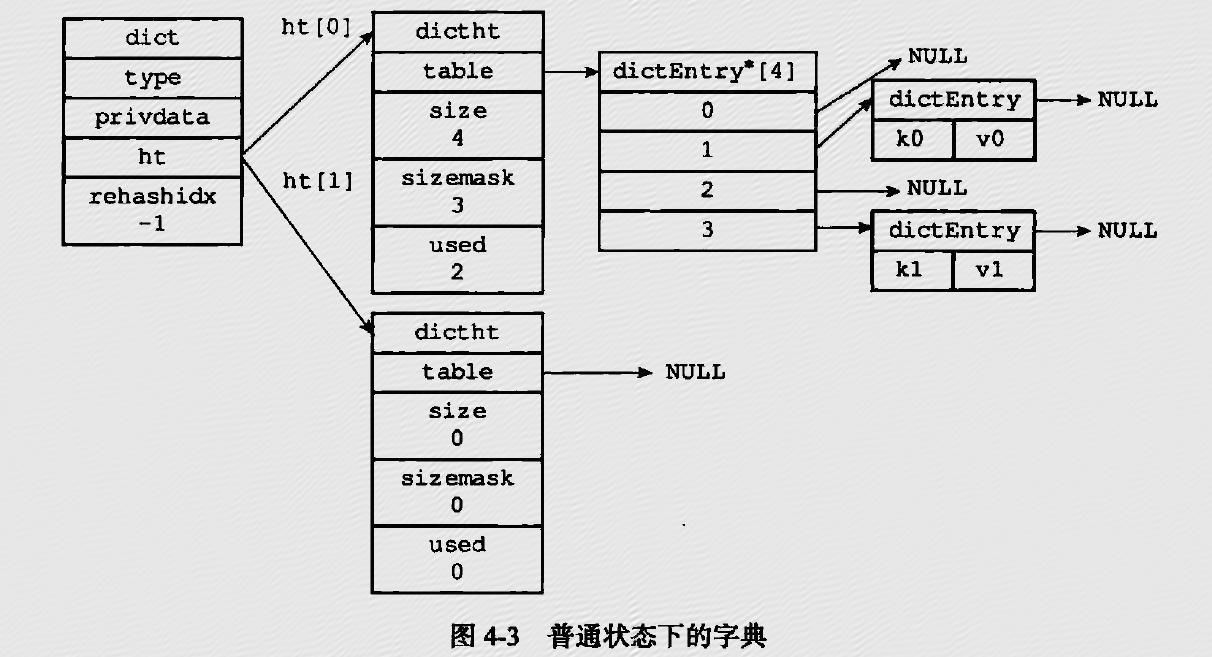
Redis的字典使用哈希表作为底层实现，一个哈希表里面可以有多个哈希表节点，每个哈希表节点就保存了字典中的一个键值对。

Redis字典所使用的哈希表由dict.h/dictht结构定义



上图中table是一个数组，由dictEntry结构组成，size表示哈希表大小，即table数组大小，used表示已有节点(键值对)的数量。dictEntry包含了k，v，还有一个next，next指向了下一个hash值相等的键值对，形成链表来解决键冲突的问题，注意，为了性能考虑，程序总是将新节点放在该链表的表头位置，排在其他节点的前面。

Redis中的字典由dict.h/dict结构表示：



Redis中的字典如上图表示，type属性和pirvdata属性是针对不同类型的键值对，为创建多态字典而设置的。ht属性是一个包含两个项的数组，每一项都是一个dictht哈希表，字典只使用ht[0]哈希表，ht[1]哈希表只会在对ht[0]进行rehash时使用。Rehashidx记录了rehash目前的进度，如果没有进行rehash，那它的值为-1.

在对哈希表进行扩展和收缩操作时，程序需要将现有hash表所有键值对rehash，并且这个rehash不是一次性完成的，渐进式的。

### 第五章：跳跃表

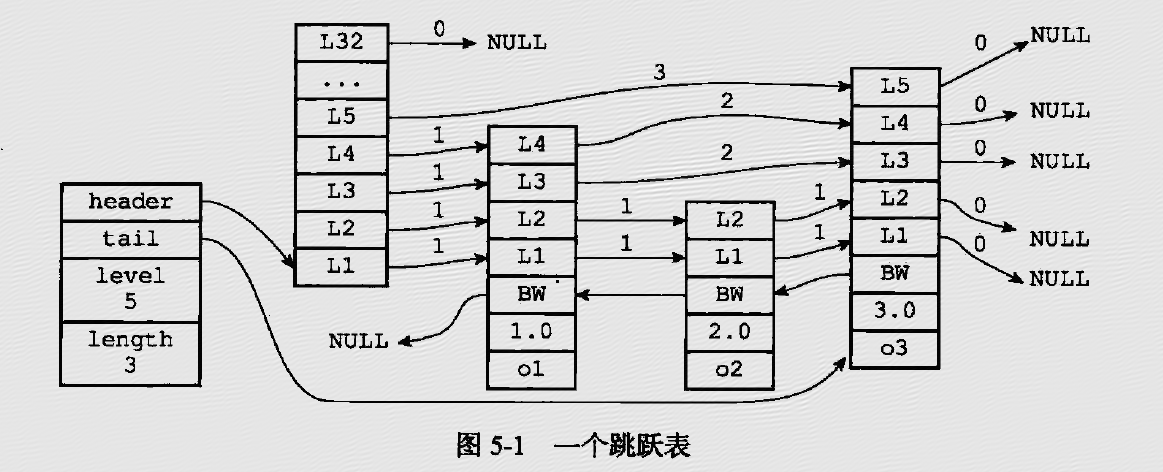
<http://redisdoc.com/sorted_set/index.html>

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，达到快速访问节点的目的。

redis使用跳跃表作为有序集合键的底层实现之一，如果一个有序集合包含的元素数量比较多，又或者有序集合中元素的成员是比较长的字符串，redis就会使用跳跃表来作为有序集合键的底层实现。

Redis只在两个地方用到了跳跃表：一是实现有序集合，二是在集群节点中用作内部数据结构。

跳跃表由redis.h/zskiplistNode和redis.h/zskiplist两个结构定义，zskiplistNode表示跳跃表节点，zskiplist用于保存跳跃表节点的相关信息。跳跃表示例：



如上图所示，最左边是一个zskiplist结构，该结构包含属性：

1. Header：指向跳跃表的表头节点

2. Tail：指向跳跃表的表尾节点

3. Level：记录目前跳跃表中层数最大的那个节点的层数（表头节点的层数不计算在内）

4. Length：跳跃表的长度，即跳跃表目前包含的节点数量（表头节点不计算在内）

位于zskiplist结构右方的是四个zskiplistNode结构，该结构包含属性：

1. 层（level）：节点用L1、L2、L3等字样标记节点各个层。每个层都带有两个属性:前进指针和跨度。前进指针用于访问表尾方向的其他节点，跨度则记录了前进指针所指向节点与当前节点的举例

2. 后退指针（backward）：节点中用BW字样标记节点的后退指针，它指向位于当前节点的前一个节点。后退指针在程序从表尾向表头遍历时使用。

3. 分值（score）：各个节点中的1.0，2.0和3.0是节点锁保存的分值。在跳跃表中，节点按各自所保存的分值从小到大排列。

4. 成员对象（obj）：各个节点中的o1、o2和o3是节点所保存的成员对象。

注意表头节点和其他节点的构造是一样的，表头节点也有后退指针，分值和成员对象，不过都不会用到，上图省略了。

在同一个跳跃表中，多个节点可以包含相同的分值，但是每个节点的成员对象必须是唯一的。

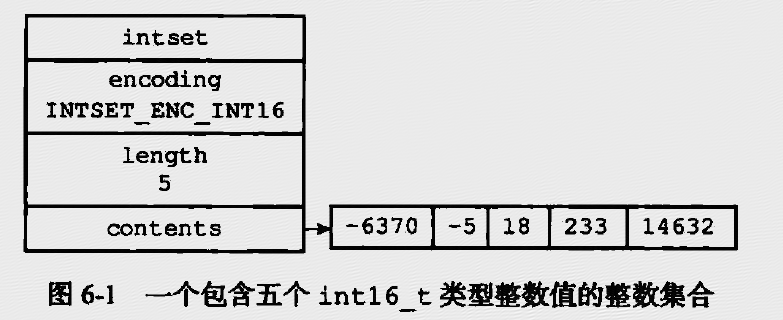
跳跃表中的节点按照分值大小进行排序，当分值相同时，节点按照成员对象的大小进行排序。

### 第六章：整数集合

<http://redisdoc.com/set/index.html>

整数集合（intset）是集合键的底层实现之一，当一个集合只包含整数值元素，并且这个集合的元素数量不多时，redis就会使用整数集合作为集合键的底层实现。

每个intset.h/intset结构表示一个整数集合。



1. Contents数组是整数集合的底层实现：整数集合的每个元素都是contents数组的一个数据项（item），各个项在数组中从小到大有序排列，并且数组中不包含任何重复项。

2. length属性记录了整数集合包含的元素数量，即contents数组长度。

每当有新元素添加到整数集合里面，并且新元素的类型比整数集合现有所有元素的类型都要长时，整数集合需要先进行升级（upgrade），然后才能将新元素添加到整数集合里面。升级整数集合并添加新元素共分为三步进行：

1. 根据新元素类型，扩展整数集合底层数组的空间大小，并为新元素分配空间

2. 将底层数组现有的所有元素都转换成与新元素相同的类型，并将类型转换后的元素放置到正确的位上，而且在放置元素的过程中，需要继续维持底层数组的有序性质不变。

3. 将新元素添加到底层数组里面。

整数集合的升级策略带来的好处：

1. 提升整数集合的灵活性。

随意将不同类型（int16\_t,int32\_t,int64\_t）的整数添加到集合，而不必担心类型错误。

2. 尽可能的节约内存。

如果数组要保存int16\_t,int32\_t,int64\_t三种类型的值，最简单的做法是直接使用int64\_t作为整数集合的底层实现。不过这样的话会浪费内存。整数集合现在的做法是既可以让集合能同时保存三种不同类型的值，又可以确保升级操作只会在有需要的时候（要存入的整数比现有的类型都要长）进行，尽量节省内存。

整数集合只支持升级，不支持降级操作。

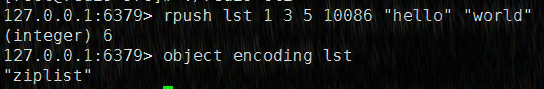
### 第七章：压缩列表

<http://redisdoc.com/list/index.html>

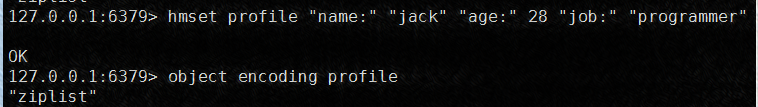
<http://redisdoc.com/hash/index.html>

压缩列表（ziplist）是列表键和哈希键的底层实现之一。

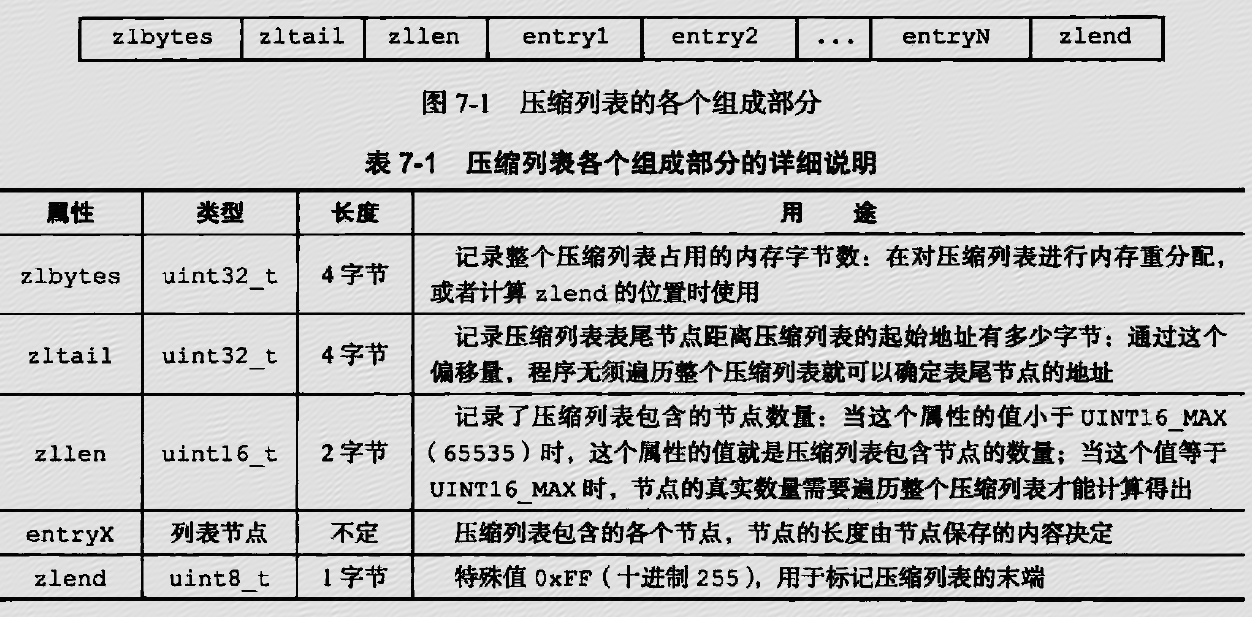
当一个列表键只包含少量列表项，并且每个列表项要么是小整数值，要么是长度比较短的字符串，那么redis就会使用压缩列表作为列表键的底层实现：举例



当一个哈希键只包含少量键值对，并且每个键值对的键和值要么是小整数值，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哈希键的底层实现。举例



压缩列表是redis为了节约内存而开发的，有一系列特殊编码的连续内存块组成的顺序型数据结构。一个压缩列表可以包含任意多个节点（entry），每个节点可以保存一个字节数组或者一个整数值。下图为压缩列表的组成。



### 第八章：对象