1. 压缩列表
2. 概述

压缩列表是列表键和哈希键的底层实现之一，所谓的列表键指的是该键的值时列表，同理哈希键表示该键的值是哈希。当一个列表键只包含少量的列表项，并且每个列表项要么是小整数值，要么是短字符串，则使用压缩列表作为列表键的实现。

1. 压缩列表的构成

压缩列表是Redis为了节约内存开发的，由一系列特殊编码的连续内存块组成的顺序型结构，一个压缩列表可以包含多个节点，每个节点可保存一个字节数组或者一个整数值。压缩列表中包含了列表占用的内存字节数，记录尾结点的位置，包含的节点数量，各个节点的内容以及一个标记列表末端的字节。

1. 压缩列表节点的构成

每个压缩列表节点保存一个字节数组或者一个整数值，每个节点都由前一个节点的长度，节点的编码，节点的内容三部分组成。

1. 前一个节点的长度

以字节为单位，记录了压缩列表中前一个节点的长度，如果前一个节点的长度小于254字节，则使用1个字节记录长度，否则使用5个字节记录长度，第一个字节设置为0xFE，后面的4个字节记录实际长度。我们可以根据这个长度来计算出前一个节点的起始位置，也就可以从表尾向表头进行遍历操作。

1. 编码

记录了节点的内容属性所保存数据的类型和长度。以00, 01或者10开头的是字节数组编码，表示内容中保存着字节数组；以11开头的是整数编码。

1. 内容

节点的内容属性负责保存节点的值，节点值可以是一个字节数组或者整数。

1. 对象
2. 概述

我们之前介绍了很多Redis中的数据结构，Redis没有直接使用这些数据结构来实现键值对数据库，而是基于这些结构实现了一个对象系统，这个系统包含了字符串对象，列表对象，哈希对象，集合对象和有序集合对象这五种类型的对象。每个对象都至少有一种底层实现方式，可以针对不同的使用场景为对象设置不同的实现。

对象系统还实现了基于计数计数的内存回收机制，当程序不再使用某个对象时，这个对象占用的内存就会被自动释放。还通过引用计数计数实现了对象共享机制，节约内存，还有对象带有访问时间记录信息，用于计算数据库键的空转时长，优先删除这些键。

1. 对象的类型和编码

Redis使用对象来表示数据库中的键和值，每次创建一个键值对时，至少会创建两个对象，一个是键对象，一个是值对象，键对象永远是字符串对象，而值对象则可以是五种对象之一。Redis中的每个对象都由一个redisObject结构表示，该结构中保存着三个属性：对象类型属性，对象编码属性，指向底层实现数据结构的指针。

1. 对象类型属性

但我们称呼一个数据库键为字符串键时，表示这个键对应的值为字符串对象。TYPE命令也是这样，对一个数据库键执行TYPE命令时，返回的结果为该键对应的值对象类型。

1. 编码和底层实现

对象ptr指针指向对象的底层实现数据结构，这些数据结构由编码属性指定，即这个对象使用了什么数据结构实现的。

有以下类型的编码：

REDIS\_ENCODING\_INT: long类型的整数

REDIS\_ENCODING\_EMBSTR: embstr编码的简单动态字符串

REDIS\_ENCODING\_RAW: 简单动态字符串

REDIS\_ENCODING\_HT: 字典

REDIS\_ENCODING\_LINKEDLIST: 双端链表

REDIS\_ENCODING\_ZIPLIST: 压缩链表

REDIS\_ENCODING\_INTSET: 整数集合

REDIS\_ENCODING\_SKIPLIST: 跳跃表和字典

每种类型的对象至少使用了两种不同的编码：

字符串对象（STRING）：INT, EMBSTR, RAW

列表(LIST)：压缩列表，双端链表

哈希表(HASH)：压缩列表，字典

集合(SET)：整数集合，字典

有序集合(ZSET)：压缩列表，跳跃表和字典

使用OBJECT ENCODING命令可以查看一个数据库键的值对象的编码。

1. 字符串对象

字符串对象对应的编码可以是INT, EMBSTR, RAW，用long long类型表示的浮点数也是作为字符串对象保存的。

1. INT

如果一个字符串对象保存的是整数值，且这个整数值也可以使用long类型来表示，则会将该整数值保存到字符串对象结构中的ptr属性里面，并设置编码为INT。

1. EMBSTR

保存的是一个字符串值，且长度小于等于39个字节，则使用embstr编码。这个编码专用于保存短字符串，与RAW一样，都使用redisObject结构和sdshdr结构来表示字符串对象，但是raw编码会调用两次内存分配函数分别创建两个结构，而embstr调用一次内存分配一块连续空间。这样在释放空间时只需要一次操作，且由于使用的是连续内存，能够更好地利用缓存。

1. RAW

如果保存的是一个字符串值，且长度大于39个字节，则使用一个SDS来保存该字符串值，设置编码为RAW。

1. 编码的转换

int和embstr都会转换为raw编码，如下所示：

对于int编码，如果该对象保存的不再是整数值，而是一个字符串值，则编码转换为raw；

对于embstr编码，Redis没有编写任何修改程序，因此它是只读的，当我们修改embstr编码的字符串对象时，会先将编码转换为raw，再执行修改。

1. 字符串命令

set: 保存键对应的值

get：返回键对应的值

append: 追加字符串到末尾

incrbyfloat: 对浮点数进行加法运算

incrby: 对整数进行加法运算

decrby: 对整数进行减法运算

strlen: 计算字符串的长度

1. 列表对象

列表对象的编码可以是ziplist和linkedlist。

1. ziplist

使用压缩列表作为底层实现，每个压缩列表节点保存了一个列表元素。

1. linkedlist

使用双端链表作为底层实现，每个节点保存了一个字符串对象，每个字符串对象保存了一个列表元素。

1. 编码转换

当列表对象同时满足以下两个条件时，使用ziplist编码：

1. 列表对象保存的所有字符串元素的长度都小于64字节；
2. 列表对象保存的元素数量小于512个；

不能满足条件的使用linkedlist编码，即字符串元素长度大于或等于64字节，又或者数量大于或等于512个。只要有一个条件不满足，对象的编码转换操作就会执行，原本保存在压缩列表里的所有列表元素都会被转移并保存到双端链表中，编码也会变为linkedlist。可以看到，ziplist适合的是短字符串，以及元素数量少的情况，而linkedlist适合的是长字符串，元素数量多的情况。

1. 列表命令

列表命令都以L开头：

LPUSH: 将新元素加入表头；

RPUSH: 将新元素加入表尾；

LPOP: 定位表头节点，删除表头节点；

RPOP: 定位表尾节点，删除表尾节点；

LINDEX: 定位指定节点，返回节点保存的值；

LLEN: 返回列表的长度；

LINSERT: 插入元素；

1. 哈希对象

哈希对象的编码可以是ziplist或者hashtable。

1. ziplist

使用压缩列表作为底层实现，每当有新的键值对要加入哈希对象中，先将保存了键的压缩列表节点推入到压缩列表节点，再将保存了值的压缩列表节点推入压缩列表表尾，键值对总是连在一起。

1. hashtable

使用字典作为底层实现，哈心对象中的一个键值对使用一个字典键值对来保存，字典的每个键都是一个字符串对象，每个值也是一个字符串对象。

1. 编码转换

当哈希对象同时满足以下两个条件，使用ziplist编码：

1. 所有键和值的字符串长度小于64字节；
2. 键值对数量小于512个

不能满足的使用hashtable编码。可以看到，ziplist适合的是短字符串，以及元素数量少的情况，而hashtable适合的是长字符串，元素数量多的情况。

1. 哈希命令

哈希命令都以H开头：

HSET: 保存键值对；

HGET: 查找指定键的值；

HEXISTS: 查找指定键对应的节点；

HDEL: 删除指定键的键值对；

HLEN: 返回所有键值对数量；

HGETALL: 遍历返回所有键值对；

1. 集合对象

集合对象的编码可以使用intset 或者 hashtable。

1. intset

使用整数集合作为底层实现，集合对象包含的所有元素都被保存在整数集合中。

1. hashtable

使用字典作为底层实现，字典的每个键都是一个字符串对象，每个字符串对象包含了一个集合元素，而字典的值都被设置为了NULL。

1. 编码的转换

当集合对象可以同时满足以下条件时，使用intset编码：

1. 集合对象保存的所有元素都是整数值；
2. 集合对象保存的元素数量不超过512个；

不能满足这两个条件的集合对象使用hashtable编码，可以看到，保存整数就用intset编码，元素数量较少也使用intset编码，hashtable对应的是数量多，保存字符串的情况。

4． 集合命令

集合命令都以S开头：

SADD: 将所有新元素添加到整数集合中；

SCARD: 返回整数集合包含的元素数量；

SISMEMBER: 查找给定的元素；

SMEMBERS: 遍历整数集合，返回所有的集合元素；

SRANDMEMBERS: 随机返回一个元素；

SPOP: 随机取出元素，并删除该元素；

SREM: 删除所有指定元素；

1. 有序集合对象

有序集合的编码可以是ziplist后者skiplist。

1. ziplist

使用压缩列表作为底层实现，每个集合元素使用两个紧紧挨在一起的压缩列表节点来保存，第一个节点保存元素的成员，第二个节点则保存元素的分值。集合元素按照分值的大小从小到大进行排序。

1. skiplist

使用zset结构作为底层实现，一个zset结构同时包含一个字典和一个跳跃表。跳跃表按照分值从小到大保存了所有集合元素，每个跳跃表节点保存了一个集合元素，跳跃表节点的object属性保存了元素的成员，score属性保存了元素的分值，通过跳跃表，可以对有序集合进行范围操作；字典为有序集合创建了一个从成员到分值的映射，字典中的每个键值对都保存了成员到分值的映射，通过字典可以用O(1)复杂度找到给定成员的分值。

有序集合的每个元素的成员都是一个字符串对象，分值都是一个double类型的浮点数，虽然跳跃表和字典都保存了集合元素，但是会通过指针来共享相同元素的成员和分值。为什么使用两个结构来同时保存呢？因为要保留优点，跳表的优点是可以执行范围性操作，跳到很远的节点上，但是根据成员查找分值就不会是O(1)复杂度了，字典的优点是查询分值快，但是集合元素时无序的。

1. 编码的转换

当有序集合对象满足以下两个条件时，使用ziplist编码：

1. 元素数量小于128个；
2. 所有元素成员的长度都小于64字节；

不能满足以上两个条件的有序集合对象使用skiplist编码。

1. 有序集合命令

有序集合命令都以Z开头：

ZADD: 插入成员和分值；

ZCARD: 获取集合长度；

ZCOUNT: 统计分值在给定范围内的节点数量；

ZRANGE: 从表头向表尾遍历，返回给定索引范围内的所有元素；

ZREVRANGE: 从表尾向表头遍历，返回给定索引范围内的所有元素；

ZRANK: 从表头向表尾遍历，返回某个成员的排名，以0开头的；

ZREVRANK: 从表尾向表头遍历，返回某个成员的排名；

ZREM: 遍历删除给定元素；

ZSCORE: 遍历查找给定元素；

1. 类型检查与命令多态

Redis用于操作键的命令可以分为两类，一类可以对任何类型的键执行，如DEL, EXPIRE , RENAME, TYPE, OBJECT命令；另一类只能对特定类型的键执行，如五种类型的键命令。在执行一个类型特定的命令之前，Redis会先检查输入键的类型是否正确，再决定是否执行给定的命令。类型检查是通过redisObject结构中的type属性完成的。

Redis除了根据值对象的类型来判断键是否能够执行指定命令外，还会根据值对象的编码方式来选择正确的命令实现代码来执行命令。不同的编码实现方式会调用不同的API接口。

1. 内存回收

Redis在对象系统中构建了一个引用计数实现的内存回收机制，通过这一机制，程序可以通过跟踪对象的引用计数信息，自动释放对并进行内存回收，由redisObject结构的refcount属性记录，当引用计数为0时会自动释放。

1. 对象共享

对象的引用计数还带有对象共享的作用，在Redis中，让多个键共享同一个值对象有以下步骤：

1. 将数据库键的值指针指向一个现有的值对象；
2. 将被共享的值对象的引用计数+1

目前，Redis会在初始化服务器时创建一万个字符串对象，包括0-9999的所有整数值，需要使用时直接使用这些共享对象。怎么验证这个结论呢？我们可以创建一个值为2的键A，通过OBJECT REFCOUNT 查看它的引用计数，可以看到是2，说明一个是持有这个值对象的服务器程序，一个是共享这个值对象的键A。

目前Redis只支持对整数值的字符串对象进行共享，因为只有在共享对象和目标对象完全相同时才会将共享对象作为键的值对象，这需要验证操作，需要消耗CPU时间，因此越复杂的对象越消耗CPU时间。

（十一） 对象的空转时长

redisObject结构中中还有一个属性lru，记录了对象最后一次被命令程序访问的时间，使用object idletime命令可以打印出给定键的空转时间，使用当前时间减去lru时间即可计算出来。当然，当内存不够用时，这种属性还可以用来淘汰空转时间最长的键。