52|算法实战（一）：剖析Redis常用数据类型对应的数据结构

到此为止，专栏前三部分我们全部讲完了。从今天开始，我们就正式进入实战篇的部分。这部分我主要通过一些开源项目、经典系统，真枪实弹地教你，如何将数据结构和算法应用到项目中。所以这部分的内容，更多的是知识点的回顾，相对于基础篇、高级篇的内容，其实这部分会更加容易看懂。

不过，我希望你不要只是看懂就完了。你要多举一反三地思考，自己接触过的开源项目、基础框架、中间件中，都用过哪些数据结构和算法。你也可以想一想，在自己做的项目中，有哪些可以用学过的数据结构和算法进一步优化。这样的学习效果才会更好。

好了，今天我就带你一块儿看下，经典数据库**Redis**中的常用数据类型，底层都是用哪种数据结构实现的？

Redis数据库介绍

Redis是一种键值（Key-Value）数据库。相对于关系型数据库（比如MySQL），Redis也被叫作非关系型数据库。

像MySQL这样的关系型数据库，表的结构比较复杂，会包含很多字段，可以通过SQL语句，来实现非常复杂的查询需求。而Redis中只包含“键”和“值”两部分，只能通过“键”来查询“值”。正是因为这样简单的存储结构，也让Redis的读写效率非常高。

除此之外，Redis主要是作为内存数据库来使用，也就是说，数据是存储在内存中的。尽管它经常被用作内存数据库，但是，它也支持将数据存储在硬盘中。这一点，我们后面会介绍。

Redis中，键的数据类型是字符串，但是为了丰富数据存储的方式，方便开发者使用，值的数据类型有很多，常用的数据类型有这样几种，它们分别是字符串、列表、字典、集合、有序集合。

“字符串（string）”这种数据类型非常简单，对应到数据结构里，就是字符串。你应该非常熟悉，这里我就不多介绍了。我们着重看下，其他四种比较复杂点的数据类型，看看它们底层都依赖了哪些数据结构。

列表（list）

我们先来看列表。列表这种数据类型支持存储一组数据。这种数据类型对应两种实现方法，一种是压缩列表（ziplist），另一种是双向循环链表。

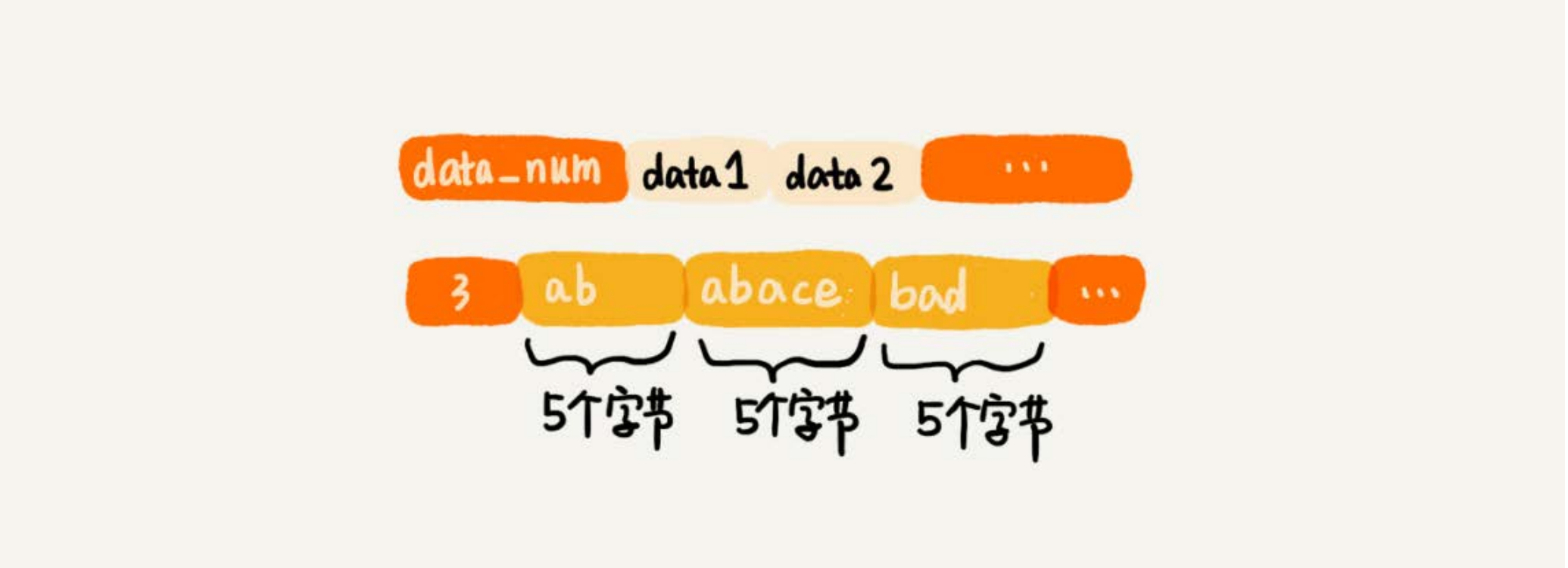
当列表中存储的数据量比较小的时候，列表就可以采用压缩列表的方式实现。具体需要同时满足下面两个条件：

- 列表中保存的单个数据（有可能是字符串类型的）小于64字节；
- 列表中数据个数少于512个。

关于压缩列表，我这里稍微解释一下。它并不是基础数据结构，而是Redis自己设计的一种数据存储结构。它有点儿类似数组，通过一片连续的内存空间，来存储数据。不过，它跟数组不同的一点是，它允许存储的数据大小不同。具体的存储结构也非常简单，你可以看我下面画的这幅图。

现在，我们来看看，压缩列表中的“压缩”两个字该如何理解？

听到“压缩”两个字，直观的反应就是节省内存。之所以说这种存储结构节省内存，是相较于数组的存储思路而言的。我们知道，数组要求每个元素的大小相同，如果我们要存储不同长度的字符串，那我们就需要用最大长度的字符串大小作为元素的大小（假设是20个字节）。那当我们存储小于20个字节长度的字符串的时候，便会浪费部分存储空间。听起来有点儿拗口，我画个图解释一下。



压缩列表这种存储结构，一方面比较节省内存，另一方面可以支持不同类型数据的存储。而且，因为数据存储在一片连续的内存空间，通过键来获取值为列表类型的数据，读取的效率也非常高。

当列表中存储的数据量比较大的时候，也就是不能同时满足刚刚讲的两个条件的时候，列表就要通过双向循环链表来实现了。

在链表里，我们已经讲过双向循环链表这种数据结构了，如果不记得了，你可以先回去复习一下。这里我们着重看一下Redis中双向链表的编码实现方式。

Redis的这种双向链表的实现方式，非常值得借鉴。它额外定义一个list结构体，来组织链表的首、尾指针，还有长度等信息。这样，在使用的时候就会非常方便。

```
// 以下是C语言代码，因为Redis是用C语言实现的。
typedef struct listnode {
  struct listNode *prev;
  struct listNode *next;
  void *value;
} listNode;


typedef struct list {
  listNode *head;
  listNode *tail;
  unsigned long len;
  // ....省略其他定义
} list;
```

字典（hash）

字典类型用来存储一组数据对。每个数据对又包含键值两部分。字典类型也有两种实现方式。一种是我们刚刚讲到的压缩列表，另一种是散列表。

同样，只有当存储的数据量比较小的情况下，Redis才使用压缩列表来实现字典类型。具体需要满足两个条件：

- 字典中保存的键和值的大小都要小于64字节；
- 字典中键值对的个数要小于512个。

当不能同时满足上面两个条件的时候，Redis就使用散列表来实现字典类型。Redis使用MurmurHash2这种运行速度快、随机性好的哈希算法作为哈希函数。对于哈希冲突问题，Redis使用链表法来解决。除此之外，Redis还支持散列表的动态扩容、缩容。

当数据动态增加之后，散列表的装载因子会不停地变大。为了避免散列表性能的下降，当装载因子大于1的时候，Redis会触发扩容，将散列表扩大为原来大小的2倍左右（具体值需要计算才能得到，如果感兴趣，你可以去阅读源码）。

当数据动态减少之后，为了节省内存，当装载因子小于0.1的时候，Redis就会触发缩容，缩小为字典中数据个数的大约2倍大小（这个值也是计算得到的，如果感兴趣，你也可以去阅读源码）。

我们前面讲过，扩容缩容要做大量的数据搬移和哈希值的重新计算，所以比较耗时。针对这个问题，Redis使用我们在散列表（中）讲的渐进式扩容缩容策略，将数据的搬移分批进行，避免了大量数据一次性搬移导致的服务停顿。

集合（set）

集合这种数据类型用来存储一组不重复的数据。这种数据类型也有两种实现方法，一种是基于有序数组，另一种是基于散列表。

当要存储的数据，同时满足下面这样两个条件的时候，Redis就采用有序数组，来实现集合这种数据类型。

- 存储的数据都是整数；
- 存储的数据元素个数不超过512个。

当不能同时满足这两个条件的时候，Redis就使用散列表来存储集合中的数据。

有序集合（sortedset）

有序集合这种数据类型，我们在跳表里已经详细讲过了。它用来存储一组数据，并且每个数据会附带一个得分。通过得分的大小，我们将数据组织成跳表这样的数据结构，以支持快速地按照得分值、得分区间获取数据。

实际上，跟Redis的其他数据类型一样，有序集合也并不仅仅只有跳表这一种实现方式。当数据量比较小的时候，Redis会用压缩列表来实现有序集合。具体点说就是，使用压缩列表来实现有序集合的前提，有这样两个：

- 所有数据的大小都要小于64字节；
- 元素个数要小于128个。

数据结构持久化

尽管Redis经常会被用作内存数据库，但是，它也支持数据落盘，也就是将内存中的数据存储到硬盘中。这样，当机器断电的时候，存储在Redis中的数据也不会丢失。在机器重新启动之后，Redis只需要再将存储在硬盘中的数据，重新读取到内存，就可以继续工作了。

刚刚我们讲到，Redis的数据格式由“键”和“值”两部分组成。而“值”又支持很多数据类型，比如字符串、列表、字典、集合、有序集合。像字典、集合等类型，底层用到了散列表，散列表中有指针的概念，而指针指向的是内存中的存储地址。 那Redis是如何将这样一个跟具体内存地址有关的数据结构存储到磁盘中的呢？

实际上，Redis遇到的这个问题并不特殊，很多场景中都会遇到。我们把它叫作数据结构的持久化问题，或者对象的持久化问题。这里的“持久化”，你可以笼统地可以理解为“存储到磁盘”。

如何将数据结构持久化到硬盘？我们主要有两种解决思路。

第一种是清除原有的存储结构，只将数据存储到磁盘中。当我们需要从磁盘还原数据到内存的时候，再重新将数据组织成原来的数据结构。实际上，Redis采用的就是这种持久化思路。

不过，这种方式也有一定的弊端。那就是数据从硬盘还原到内存的过程，会耗用比较多的时间。比如，我们现在要将散列表中的数据存储到磁盘。当我们从磁盘中，取出数据重新构建散列表的时候，需要重新计算每个数据的哈希值。如果磁盘中存储的是几GB的数据，那重构数据结构的耗时就不可忽视了。

第二种方式是保留原来的存储格式，将数据按照原有的格式存储在磁盘中。我们拿散列表这样的数据结构来举例。我们可以将散列表的大小、每个数据被散列到的槽的编号等信息，都保存在磁盘中。有了这些信息，我们从磁盘中将数据还原到内存中的时候，就可以避免重新计算哈希值。

总结引申

今天，我们学习了Redis中常用数据类型底层依赖的数据结构，总结一下大概有这五种：压缩列表（可以看作一种特殊的数组）、有序数组、链表、散列表、跳表。实际上，Redis就是这些常用数据结构的封装。

你有没有发现，有了数据结构和算法的基础之后，再去阅读Redis的源码，理解起来就容易多了？很多原来觉得很深奥的设计思想，是不是就都会觉得顺理成章了呢？

还是那句话，夯实基础很重要。同样是看源码，有些人只能看个热闹，了解一些皮毛，无法形成自己的知识结构，不能化为己用，过不几天就忘了。而有些人基础很好，不但能知其然，还能知其所以然，从而真正理解作者设计的动机。这样不但能有助于我们理解所用的开源软件，还能为我们自己创新添砖加瓦。

课后思考

1. 你有没有发现，在数据量比较小的情况下，Redis中的很多数据类型，比如字典、有序集合等，都是通过多种数据结构来实现的，为什么会这样设计呢？用一种固定的数据结构来实现，不是更加简单吗？
2. 我们讲到数据结构持久化有两种方法。对于二叉查找树这种数据结构，我们如何将它持久化到磁盘中呢？

欢迎留言和我分享，也欢迎点击“请朋友读”，把今天的内容分享给你的好友，和他一起讨论、学习。

精选留言：

- 在路上 2019-01-25 11:29:12
  思考题1：redis的数据结构由多种数据结构来实现，主要是出于时间和空间的考虑，当数据量小的时候通过数组下标访问最快、占用内存最小，而压缩列表只是数组的升级版；
  因为数组需要占用连续的内存空间，所以当数据量大的时候，就需要使用链表了，同时为了保证速度又需要和数组结合，也就有了散列表。

对于数据的大小和多少采用哪种数据结构，相信redis团队一定是根据大多数的开发场景而定的。

思考题2：二叉查找树的存储，我倾向于存储方式一，通过填充叶子节点形成完全二叉树，然后以数组的形式存储到硬盘，数据还原过程也是非常高效的。如果用存储方式二就比较复杂了。 [7赞]

- Jerry银银 2019-01-25 07:05:08
  发现一个功能：左滑进入上一篇，右滑进入下一篇 [6赞]

  编辑回复2019-01-25 13:04:10
  看来是没少刷专栏

- 田伟 **คิดถึง** 2019-01-25 17:50:29
  看完前边的课程，当知识点连成线和面之后，才发现原来是这么回事，再来看redis确认是豁然开朗。知识成体系之后记忆会更深刻，不过也带来了更多的思考和发现-----知识边界扩大了 [2赞]

- readyao 2019-01-26 10:48:12
  老师您好，有这样一个场景，A关注了B,这样的操作会同时写两个链表一个是A的关注列表，另一个是B的粉丝列表，比如用redis 的sortset来存储。现在要检查所有不一致的情况(比如，A的关注列表有B，但是B的粉丝列表没有A，或者A的关注列表没有B，但是B的粉丝列表有A)。这种情况有什么好的方法吗? [1赞]

- 三木子 2019-01-25 18:37:22
  有个疑问，比如对于有序集合，这个数据量可能会逐步增加，那么数据量达到阈值时就会切换成跳表吗？是数据全部移动到跳表，然后删除列表吗？ [1赞]

- 郑晨Cc 2019-01-25 13:25:33
  老师 关于redis的压缩列表有个地方不太明白
  虽然压缩列表看起来想数组 但他能像数组一样支持按照下标进行直接随机访问吗？比如我要访问下标为n的数据我启不是需要知道之前从0到n-1的所有数据的长度才能找到n，那这跟链表的时间复杂读没啥区别啊，而且还占用了连续的内存空间？ 还是说压缩列表中的每个元素的长度都记录在它的头部可以一次性的获取？ [1赞]

  作者回复2019-01-25 16:11:10
  哈哈，你说的没错。压缩列表不支持随机访问。有点类似链表。但是比较省存储空间啊。Redis一般都是通过key获取整个value的值，也就是整个压缩列表的数据，并不需要随机访问。

- 张淼 2019-01-29 09:44:19
  王老师，既然可以用压缩列表，为啥数据超过512个的时候不用单链表

- 骏彩灬星驰 2019-01-27 19:05:41

我只想说老师你优秀，很用心，我会尽量赶上来！

- wei 2019-01-26 20:36:34
  老师，如果字典保存的键和值的大小都小于 64 字节，并且键值对的个数小于 512 个，Redis 用压缩列表实现。从 [源码](https://github.com/antirez/redis/blob/unstable/src/ziplist.c) （ziplist.c ziplistFind） 来看压缩列表根据键查找值的方式，就是一个个遍历。如果有几百个键值对，这么查找比散列表快吗？

- 张淼 2019-01-26 14:01:16
  问题1: 压缩列表优点：访问存取快速，节省内存。但是受到操作系统空闲内存限制。越大的连续内存空间越不容易申请到。所以用了其他数据结构比如链表替代。

- 莫弹弹 2019-01-25 22:13:39
  思考题1
  redis的各个存储都有判断数据大小再选择存储结构，那么可以猜想是为了平衡时间与空间，但是优先保证空间（毕竟是内存），单一结构简单易懂，大概redis多种结构就是从单一结构优化而来的吧。
  这里有个疑问，判断条件512和128是随便写的？这有什么计算方案？

  2持久化
  普通树可以选择一个遍历方式，例如前序遍历，把树变成链表，把数据挨个保存，等需要取出来再按顺序把所有元素加到树里面。
  二叉查找树本身底层就是使用数组，整个数组写进磁盘就完事了，取出来也是连续数组，完全不用保存树结构

- 纯洁的憎恶 2019-01-25 18:46:25
  Redis这种K-Value非关系数据库是否也能够很好的支持分布式文件系统呢？

- 猫头鹰爱拿铁 2019-01-25 17:29:28
  思考题
  1. redis主要作为内存数据库来使用，数据存储在内存中。而内存相比于又便宜空间又大的磁盘而言，需要考虑存储空间相对有限和贵的问题。针对于数据量比较小的情况，尽量使用节省空间的数据结构。这个价格成本和实现成本比较是划算的。
  2. 如果用指针来表示二叉查找树的成员变量会导致持久化后物理地址失效的问题。可否借用数组的思想，存储二叉查找树节点的数据以数组的形式组织，如树中没有节点的位置在数组的对应位置使用0或null。例如二叉查找树5、3、8、2、9则存储相应的数据值5、3、8、2、0、0、9同时存储数组的长度

- JimHug 2019-01-25 10:32:21
  后边很少有跟上的

- Yakmoz 2019-01-25 10:31:31
  1. redis作为一个内存存储数据，应该会在时间复杂度不是很高的情况下，尽可能的少使用内存，基于这个原则对不同情况，采取不同策略，像set在都是整数的情况下，占用空间大小都是固定的就没必要在存储数据长度了

2. 序列化和反序列化，可以按照中序后序前序遍历树进行存储

- 小美 2019-01-25 10:12:58

  王老师两种数据结构持久化：
  redius用“清除格式，持久化数据再组织数据结构”这么看来是很消耗性能，为什么不用“保留数据结构”的方式。我理解后者只牺牲部分空间换取了更多性能。
  麻烦王老师解释下？

  作者回复2019-01-25 16:16:57

  对于Redis来说，重启并不是很经常的事情。所以并不会经常从硬盘加载数据到内存再重构成数据结构。

  实际上，两种存储格式都可以，可能Redis就是随意选择了一个而已。不要太纠结为啥选的是这个，而不是那个。

- Sharry 2019-01-25 09:32:06

  1. 对于数据量较小的情况，使用链表直接遍历的增删改查的效率比起复杂的 树/散列表/跳表 忽略了旋转，hash...等运算，没有太明显的劣势，效率反而会更高
  2. 关于树的序列化：可以使用前序遍历二叉树的规则序列化，左右子孩子为 NULL 则记为 '#'
  3. 关于树的反序列化：使用前序遍历二叉树的规则还原，遇到 '#' 则说明遍历到了叶子结点

- 大张 2019-01-25 09:26:59

  比高级篇，没那么烧脑了

- WL 2019-01-25 08:53:03

  想问老师两个问题：
  1.在列表的双向链表数据结构中，使用额外的list来存储首位节点和长度等信息后使用起来就会方便呢？
  2.redis的持久化策略中，清除原有的存储结构只将数据存在磁盘中，那这些数据具体在磁盘中是怎么存储的？是采用压缩列表的方式吗？

  作者回复2019-01-25 16:21:35

  1. 看具体应用了。这种实现思路跟我们平常直接定义链表的方式有点不一样（Node * list = null;这样子）。所以我就稍微提了一下。
  2. 这个跟数据结构和算法关系不大了。我没仔细研究过。理论上猜一下的话，就是按照一个pair（key+value）一个pair的顺序存储在文件中。pair之间用一定方法分割，方便读取。