Redis 笔记三

作者：matthewhan
原文链接：https://ld246.com/article/1626059978962
来源网站：链滴

Q：万金油的 String 不一定好用

场景：

开发一个图片存储系统，要求这个系统能快速地记录图片 ID 和图片在存储系统中保存时的 ID（可以接叫作图片存储对象 ID）。同时，还要能够根据图片 ID 快速查找到图片存储对象 ID。

用 10 位数来表示图片 ID 和图片存储对象 ID，例如，图片 ID 为 1101000051，它在存储系统中对的 ID 号是 3301000051。

初始设计：

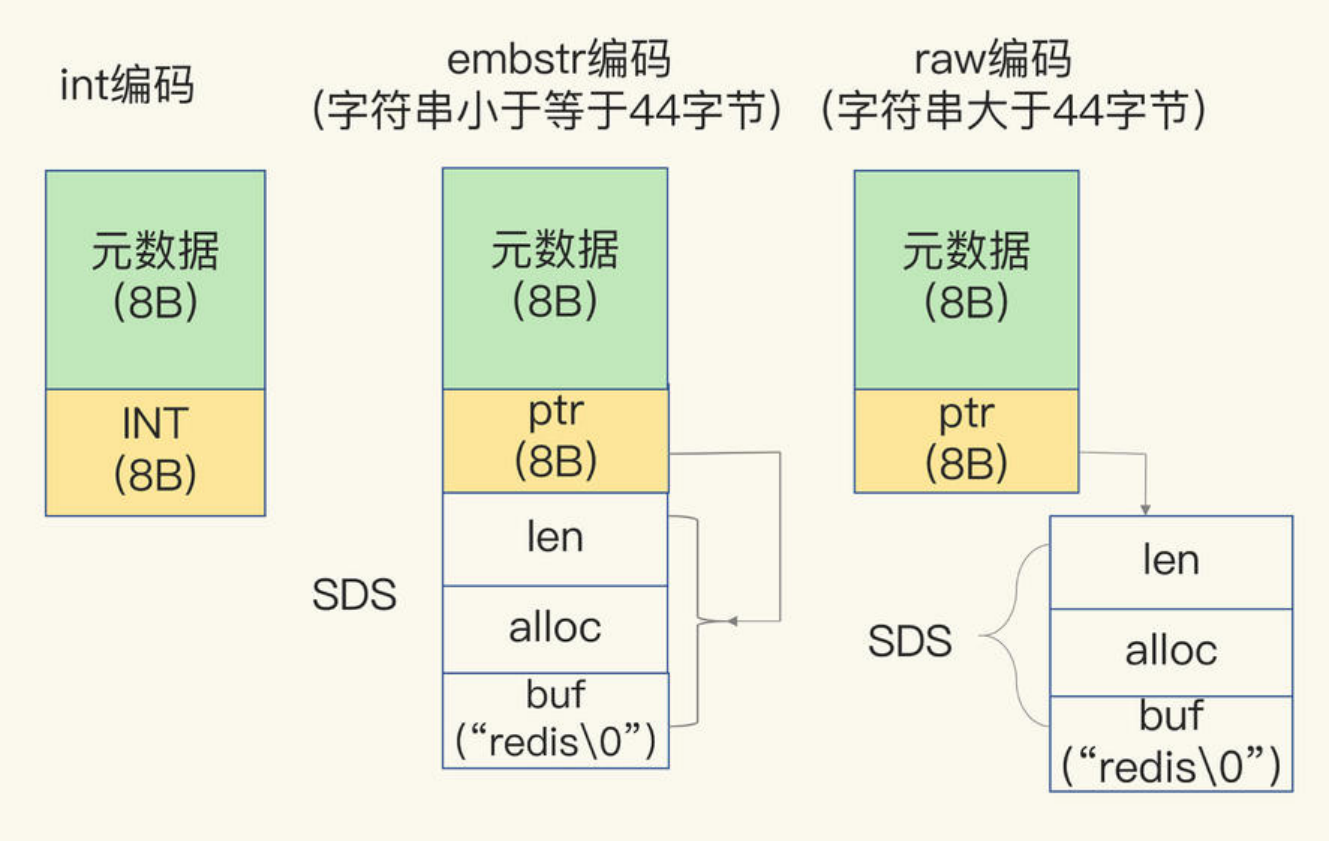
图片 ID 和图片存储对象 ID 正好一一对应，是典型的“键 - 单值”模式。所谓的“单值”，就是指键对中的值就是一个值，而不是一个集合，这和 String 类型提供的“一个键对应一个值的数据”的保形式刚好契合。

刚开始，我们保存了 1 亿张图片，大约用了 6.4GB 的内存。但是，随着图片数据量的不断增加，我的 Redis 内存使用量也在增加，结果就遇到了大内存 Redis 实例因为生成 RDB 而响应变慢的问题。显然，String 类型并不是一种好的选择，我们还需要进一步寻找能节省内存开销的数据类型方案。

内存使用量大的原因

String 类型并不是适用于所有场合的，它有一个明显的短板，就是它保存数据时所消耗的内存空间较。

其实，除了记录实际数据，String 类型还需要额外的内存空间记录数据长度、空间使用等信息，这些息也叫作元数据。



反正就是除了本身的应存的数据之外，还保留很多其他的元数据，所以占用内存较大。

如何优化

在保存单值的键值对时，可以采用基于 Hash 类型的二级编码方法。这里说的二级编码，就是把一个值的数据拆分成两部分，前一部分作为 Hash 集合的 key，后一部分作为 Hash 集合的 value，这样来，我们就可以把单值数据保存到 Hash 集合中了。

以图片 ID 1101000060 和图片存储对象 ID 3302000080 为例，我们可以把图片 ID 的前 7 位（1101 00）作为 Hash 类型的键，把图片 ID 的最后 3 位（060）和图片存储对象 ID 分别作为 Hash 类型中的 key 和 value。

二级编码一定要把图片 ID 的前 7 位作为 Hash 类型的键，把最后 3 位作为 Hash 类型值中的 key 吗其实，二级编码方法中采用的 ID 长度是有讲究的。

Redis Hash 类型的两种底层实现结构，分别是压缩列表和哈希表。那么，Hash 类型底层结构什么时使用压缩列表，什么时候使用哈希表呢？其实，Hash 类型设置了用压缩列表保存数据时的两个阈值一旦超过了阈值，Hash 类型就会用哈希表来保存数据了。

这两个阈值分别对应以下两个配置项：

- **hash-max-ziplist-entries**：表示用压缩列表保存时哈希集合中的最大元素个数。
- **hash-max-ziplist-value**：表示用压缩列表保存时哈希集合中单个元素的最大长度。

如果我们往 Hash 集合中写入的元素个数超过了 hash-max-ziplist-entries，或者写入的单个元素大超过了 hash-max-ziplist-value，Redis 就会自动把 Hash 类型的实现结构由压缩列表转为哈希表。旦从压缩列表转为了哈希表，Hash 类型就会一直用哈希表进行保存，而不会再转回压缩列表了。在省内存空间方面，哈希表就没有压缩列表那么高效了。

为了能充分使用压缩列表的精简内存布局，我们一般要控制保存在 Hash 集合中的元素个数。所以，刚才的二级编码中，我们只用图片 ID 最后 3 位作为 Hash 集合的 key，也就保证了 Hash 集合的元个数不超过 1000，同时，我们把 hash-max-ziplist-entries 设置为 1000，这样一来，Hash 集合就以一直使用压缩列表来节省内存空间了。

个人总结

这篇就是说了 String 在大量的键值对方面内存容量上有点拉胯，可以抽取出这些 key 的共性，再搞个集合类型作为 value。当然如果 key 各个都不太一样就不太好搞了。

另外，教你了怎么采用压缩列表能够更大程度的节省空间。然后这个何时采用压缩列表何时采用其他数据结构，配置项可配的。

另外，压缩列表是一块连续内存，对 CPU cache 也友好，CPU 命中率也不错，所以读取速度也非常。

Q：有一亿个keys要统计，应该用哪种集合

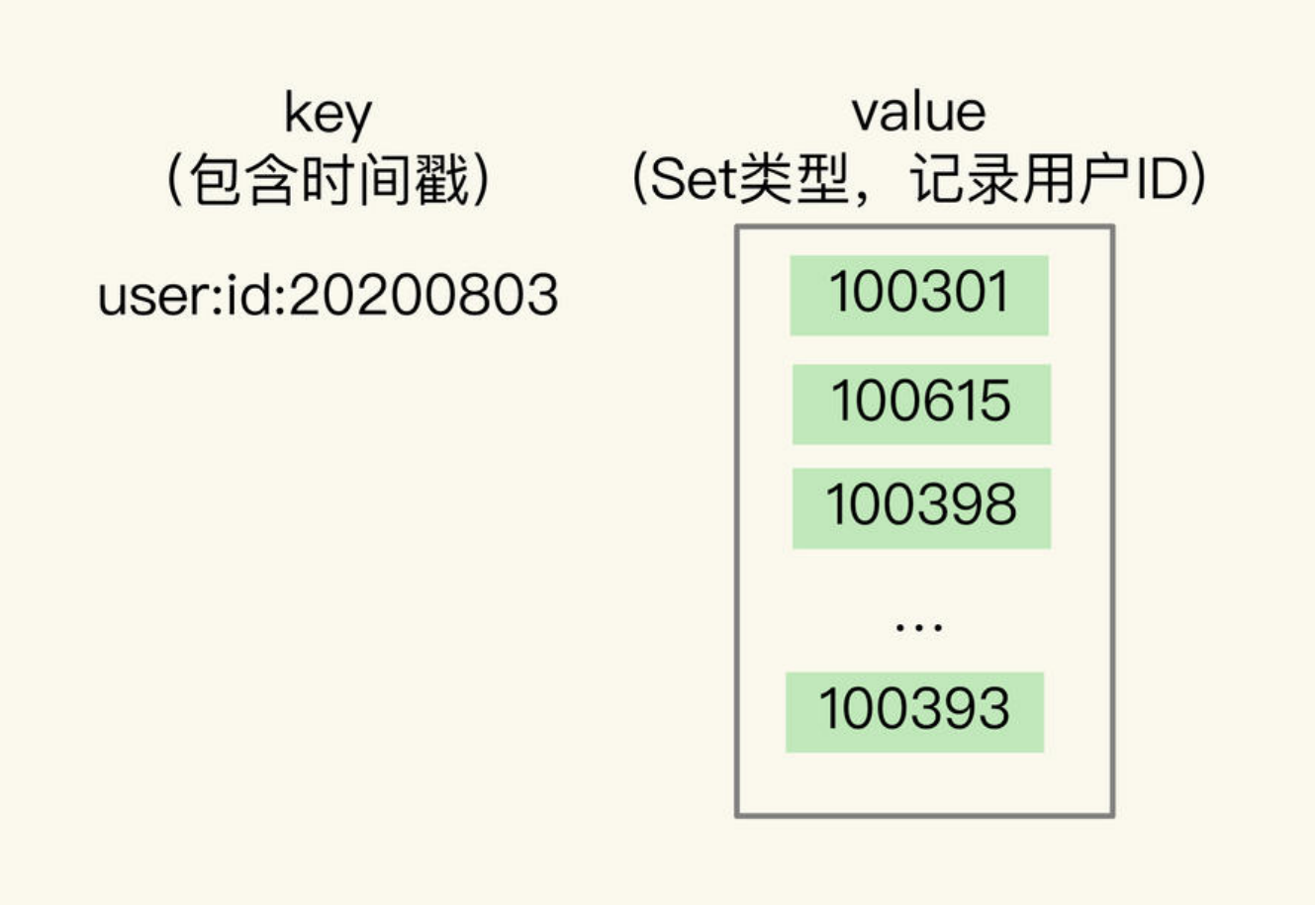
- 在移动应用中，需要统计每天的新增用户数和第二天的留存用户数；
- 在电商网站的商品评论中，需要统计评论列表中的最新评论；
- 在签到打卡中，需要统计一个月内连续打卡的用户数；
- 在网页访问记录中，需要统计独立访客（Unique Visitor，UV）量。

聚合统计

统计手机 App 每天的新增用户数和第二天的留存用户数

所谓的聚合统计，就是指统计多个集合元素的聚合结果，包括：统计多个集合的共有元素（交集统计）；把两个集合相比，统计其中一个集合独有的元素（差集统计）；统计多个集合的所有元素（并集统）。

我们还需要把每一天登录的用户 ID，记录到一个新集合中，我们把这个集合叫作每日用户 Set，它有个特点：key 是 user:id 以及当天日期，例如 user:id:20200803；value 是 Set 集合，记录当天登录用户 ID。



当要计算 8 月 4 日的留存用户时，我们只需要再计算 user:id:20200803 和 user:id:20200804 两个 St 的交集，就可以得到同时在这两个集合中的用户 ID 了，这些就是在 8 月 3 日登录，并且在 8 月 4 留存的用户。

执行的命令如下：

```
SINTERSTORE user:id:rem user:id:20200803 user:id:20200804
```

Set 的差集、并集和交集的计算复杂度较高，在数据量较大的情况下，如果直接执行这些计算，会导致 Redis 实例阻塞。所以，我给你分享一个小建议：**你可以从主从集群中选择一个从库，让它专门负责合计算，或者是把数据读取到客户端，在客户端来完成聚合统计**，这样就可以规避阻塞主库实例和其从库实例的风险了。

排序统计

电商网站上提供最新评论列表

List 是按照元素进入 List 的顺序进行排序的，而 Sorted Set 可以根据元素的权重来排序，我们可以自己来决定每个元素的权重值。比如说，我们可以根据元素插入 Sorted Set 的时间确定权重值，先插入的元素权重小，后插入的元素权重大。

这样会出现翻页过程中，新插入元素，导致第二页出现重复数据的情况（其实这种情况因被允许，我很多网站其实都是存在该现象）

如果不想出现这样的情况，就用 Sorted Set，因为他的取的权重还是老的，所以不会出现新的元素除非你又获取了新的元素，拿到的新的权重，不然就是老的权重。

ZRANGEBYSCORE comments N-9 N

二值状态统计

签到打卡的场景

Bitmap 提供了 GETBIT/SETBIT 操作，使用一个偏移值 offset 对 bit 数组的某一个 bit 位进行读和写不过，需要注意的是，Bitmap 的偏移量是从 0 开始算的，也就是说 offset 的最小值是 0。当使用 SETBIT 对一个 bit 位进行写操作时，这个 bit 位会被设置为 1。Bitmap 还提供了 BITCOUNT 操作，用统计这个 bit 数组中所有“1”的个数。

那么，具体该怎么用 Bitmap 进行签到统计呢?

借助一个具体的例子来说明。假设我们要统计 ID 3000 的用户在 2020 年 8 月份的签到情况，就可按照下面的步骤进行操作。

第一步，执行下面的命令，记录该用户 8 月 3 号已签到。

SETBIT uid:sign:3000:202008 2 1

第二步，检查该用户 8 月 3 日是否签到。

GETBIT uid:sign:3000:202008 3

第三步，统计该用户在 8 月份的签到次数。

BITCOUNT uid:sign:3000:202008

所以，如果只需要统计数据的二值状态，例如商品有没有、用户在不在等，就可以使用 Bitmap，因它只用一个 bit 位就能表示 0 或 1。在记录海量数据时，Bitmap 能够有效地节省内存空间。

基数统计

统计网页的 UV

网页 UV 的统计有个独特的地方，就是需要去重，一个用户一天内的多次访问只能算作一次。在 Redis 的集合类型中，Set 类型默认支持去重，所以看到有去重需求时，我们可能第一时间就会想到用 Set 型。

我们来结合一个例子看一看用 Set 的情况。有一个用户 user1 访问 page1 时，你把这个信息加到 Set 中：

SADD page1:uv user1

但是，如果 page1 非常火爆，UV 达到了千万，这个时候，一个 Set 就要记录千万个用户 ID。对于个搞大促的电商网站而言，这样的页面可能有成千上万个，如果每个页面都用这样的一个 Set，就会耗很大的内存空间。

这时候，就要用到 Redis 提供的 HyperLogLog 了。HyperLogLog 是一种用于统计基数的数据集合型，它的最大优势就在于，当集合元素数量非常多时，它计算基数所需的空间总是固定的，而且还很。

在统计 UV 时，你可以用 PFADD 命令（用于向 HyperLogLog 中添加新元素）把访问页面的每个用都添加到 HyperLogLog 中。

PFADD page1:uv user1 user2 user3 user4 user5

接下来，就可以用 PFCOUNT 命令直接获得 page1 的 UV 值了，这个命令的作用就是返回 HyperLoLog 的统计结果。

PFCOUNT page1:uv

HyperLogLog 的统计规则是基于概率完成的，所以它给出的统计结果是有一定误差的，标准误算率 0.81%。这也就意味着，你使用 HyperLogLog 统计的 UV 是 100 万，但实际的 UV 可能是 101 万虽然误差率不算大，但是，如果你需要精确统计结果的话，最好还是继续用 Set 或 Hash 类型。

数据类型	聚合统计	排序统计	二值状态统计	基数统计
Set	支持差集、交集、并集计算	不支持	不支持	精确统计，大数据量时，效率低，内存开销大
Sorted Set	支持交集、并集计算	支持		
Hash	不支持	不支持		
List	不支持	支持		不支持，元素没有去重
Bitmap	与、或、异或计算	不支持	支持，大数据量时，效率高，省内存	精确统计，大数据量时，内存开销大于HyperLogLog
HyperLogLog	不支持	不支持	不支持	概率统计，大数据量时，非常节省内存

[poll1626059944882]