数据类型使用

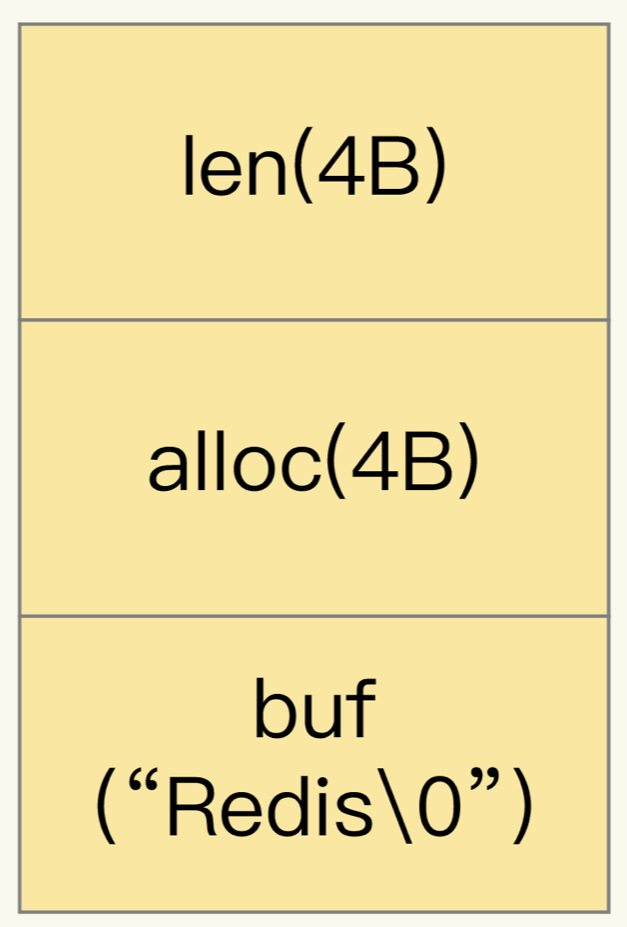
一、String

String类型除了记录实际数据，还需要额外的内存空间记录数据长度、空间使用等信息，这些信息也叫作元数据。当实际保存的数据较小时，元数据的空间开销就显得比较大。

1.1 String类型保存数据

- [ ] 当保存64位有符号整数时，String类型会把它保存为一个8字节的Long类型整数，这种保存方式通常也叫作int编码方式。

- [ ] 当保存的数据中包含字符时，String类型就会用简单动态字符串（Simple Dynamic String，SDS）结构体来保存，如下图所示：



①buf：字节数组，保存实际数据。为了表示字节数组的结束，Redis会自动在数组最后加一个“\0”，这就会额外占用1个字节的开销。

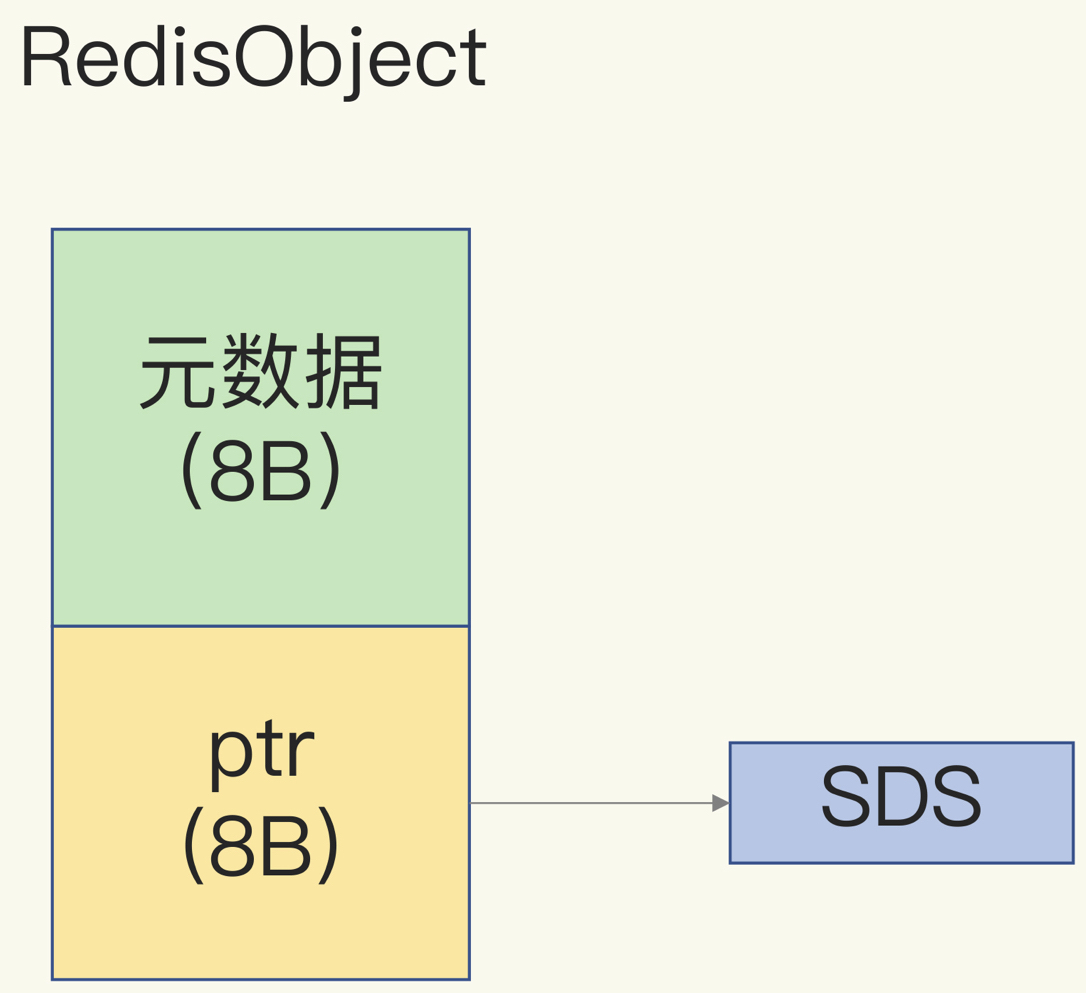
②len：占4个字节，表示buf的已用长度。

③alloc：也占个4字节，表示buf的实际分配长度，一般大于len。

另外，对于String类型来说，除了SDS的额外开销，还有一个来自于RedisObject结构体的开销。

因为Redis的数据类型有很多，而且，不同数据类型都有些相同的元数据要记录（比如最后一次访问的时间、被引用的次数等），所以，Redis会用一个RedisObject结构体来统一记录这些元数据，同时指向实际数据。

一个RedisObject包含了8字节的元数据和一个8字节指针，这个指针再进一步指向具体数据类型的实际数据所在。

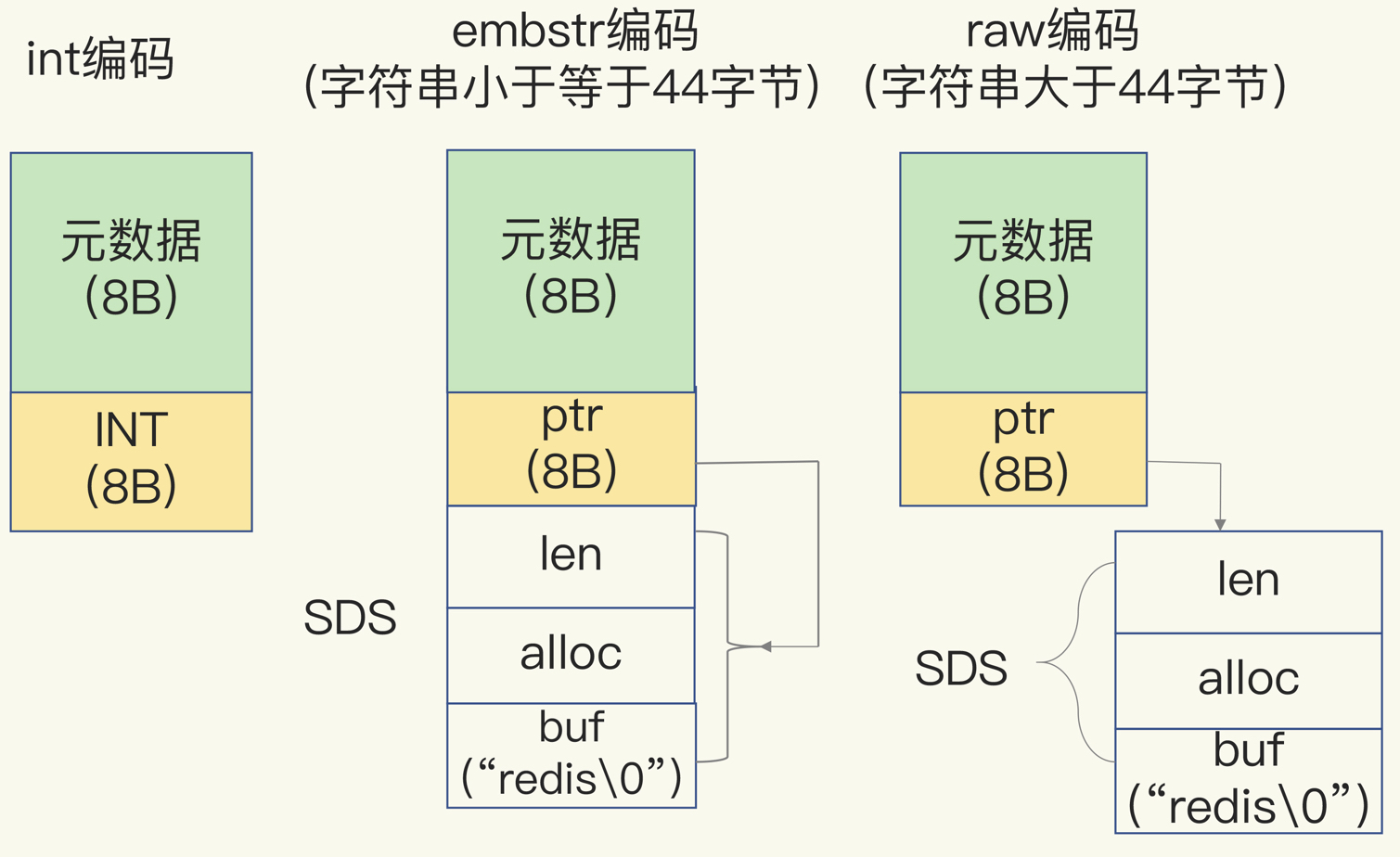


为了节省内存空间，Redis还对Long类型整数和SDS的内存布局做了专门的设计。

- [ ] 当保存的是Long类型整数时，RedisObject中的指针就直接赋值为整数数据了，这样就不用额外的指针再指向整数了，节省了指针的空间开销。

- [ ] 当保存的是字符串数据，并且字符串小于等于44字节时，RedisObject中的元数据、指针和SDS是一块连续的内存区域，这样就可以避免内存碎片。这种布局方式也被称为embstr编码方式。

- [ ] 当字符串大于44字节时，SDS的数据量就开始变多了，Redis就不再把SDS和RedisObject布局在一起了，而是会给SDS分配独立的空间，并用指针指向SDS结构。这种布局方式被称为raw编码模式。



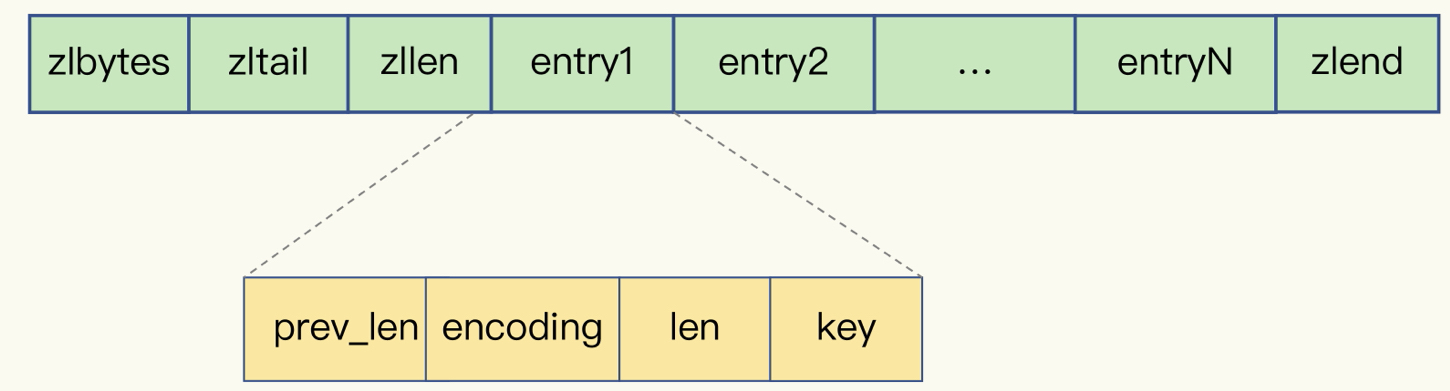
Redis会使用一个全局哈希表保存所有键值对，哈希表的每一项是一个dictEntry的结构体，用来指向一个键值对。dictEntry结构中有三个8字节的指针，分别指向key、value以及下一个dictEntry，三个指针共24字节。

jemalloc在分配内存时，会根据我们申请的字节数N，找一个比N大，但是最接近N的2的幂次数作为分配的空间，这样可以减少频繁分配的次数。

二、压缩列表（ziplist）

2.1 构成

表头有三个字段zlbytes、zltail和zllen，分别表示列表长度、列表尾的偏移量，以及列表中的entry个数。压缩列表尾还有一个zlend，表示列表结束。



①prev\_len，表示前一个entry的长度。prev\_len有两种取值情况：1字节或5字节。取值1字节时，表示上一个entry的长度小于254字节。虽然1字节的值能表示的数值范围是0到255，但是压缩列表中zlend的取值默认是255，因此，就默认用255表示整个压缩列表的结束，其他表示长度的地方就不能再用255这个值了。所以，当上一个entry长度小于254字节时，prev\_len取值为1字节，否则，就取值为5字节。

②len：表示自身长度，4字节；

③encoding：表示编码方式，1字节；

④content：保存实际数据。

Redis基于压缩列表实现了List、Hash和Sorted Set这样的集合类型，这样做的最大好处就是节省了dictEntry的开销。

三、用集合类型保存单值的键值对

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，就是把一个单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。

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。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。

一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。在节省内存空间方面，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。

四、统计模式

4.1 聚合统计

当需要对多个集合进行聚合计算时，Set类型会是一个非常不错的选择。但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，可以从主从集群中选择一个从库，让它专门负责聚合计算，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他从库实例的风险了。

4.2 排序统计

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。

- [ ] List是通过元素在List中的位置来排序的，当有一个新元素插入时，原先的元素在List中的位置都后移了一位，比如说原来在第1位的元素现在排在了第2位。所以，对比新元素插入前后，List相同位置上的元素就会发生变化，用LRANGE读取时，就会读到旧元素。

- [ ] Sorted Set的ZRANGEBYSCORE命令就可以按权重排序后返回元素。即使集合中的元素频繁更新，Sorted Set也能通过ZRANGEBYSCORE命令准确地获取到按序排列的数据。

假设越新的评论权重越大，目前最新评论的权重是N，我们执行下面的命令时，就可以获得最新的10条评论：

ZRANGEBYSCORE comments N-9 N

所以，在面对需要展示最新列表、排行榜等场景时，如果数据更新频繁或者需要分页显示，建议你优先考虑使用Sorted Set。

4.3 二值状态统计

二值状态是指集合元素的取值就只有0和1两种。

4.3.1 Bitmap（Redis提供的扩展数据类型）

Bitmap本身是用String类型作为底层数据结构实现的一种统计二值状态的数据类型。String类型是会保存为二进制的字节数组，所以，Redis就把字节数组的每个bit位利用起来，用来表示一个元素的二值状态。

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。Bitmap的偏移量是从0开始算的，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，用来统计这个bit数组中所有“1”的个数。

4.4 基数统计

基数统计是指统计一个集合中不重复的元素个数。

4.4.1 HyperLogLog

HyperLogLog是一种用于统计基数的数据集合类型，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，而且还很小。

在Redis中，每个 HyperLogLog只需要花费 12 KB 内存，就可以计算接近 2^64 个元素的基数。你看，和元素越多就越耗费内存的Set和Hash类型相比，HyperLogLog就非常节省空间。

注：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，标准误算率是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。

4.5 小结



五、GEO

GEO可以记录经纬度形式的地理位置信息，被广泛地应用在LBS服务中。

5.1 底层结构

用Sorted Set来实现。

5.2 编码方法

基本原理就是“二分区间，区间编码”。

即对二维地图做区间划分，以及对区间进行编码。一组经纬度落在某个区间后，就用区间的编码值来表示，并把编码值作为Sorted Set元素的权重分数。

5.3 操作GEO类型

六、扩展数据类型实现途径

- [ ] 基于现有的数据类型，通过数据编码或是实现新的操作的方式，来实现扩展数据类型，例如基于Sorted Set和GeoHash编码实现GEO，以及基于String和位操作实现Bitmap

- [ ] 开发自定义的数据类型，具体的操作是增加新数据类型的定义，实现创建和释放函数，实现新数据类型支持的命令操作

七、时间序列数据

7.1 读写特点

- [ ] 插入数据快

- [ ] 查询模式多

7.2 基于Hash和Sorted Set保存

- [ ] Redis内在的数据类型，代码成熟和性能稳定

- [ ] Hash类型可以实现对单键的快速查询，满足了时间序列数据的单键查询需求

- [ ] Sorted Set能够根据元素的权重分数来排序。可以把时间戳作为Sorted Set集合的元素分数，把时间点上记录的数据作为元素本身。

保证写入Hash和Sorted Set是一个原子性的操作：

- [ ] MULTI命令：表示一系列原子性操作的开始。收到这个命令后，Redis就知道，接下来再收到的命令需要放到一个内部队列中，后续一起执行，保证原子性。

- [ ] EXEC命令：表示一系列原子性操作的结束。一旦Redis收到了这个命令，就表示所有要保证原子性的命令操作都已经发送完成了。此时，Redis开始执行刚才放到内部队列中的所有命令操作。

7.3 基于RedisTimeSeries模块

Sorted Set只支持范围查询，无法直接进行聚合计算，所以，只能先把时间范围内的数据取回到客户端，然后在客户端自行完成聚合计算。但会导致大量数据在Redis实例和客户端间频繁传输，这会和其他操作命令竞争网络资源，导致其他操作变慢。

RedisTimeSeries是Redis的一个扩展模块。它专门面向时间序列数据提供了数据类型和访问接口，并且支持在Redis实例上直接对数据进行按时间范围的聚合计算。

因为RedisTimeSeries不属于Redis的内建功能模块，在使用时，我们需要先把它的源码单独编译成动态链接库redistimeseries.so，再使用loadmodule命令进行加载，如下所示：

loadmodule redistimeseries.so

当用于时间序列数据存取时，RedisTimeSeries的操作主要有5个：

• 用TS.CREATE命令创建时间序列数据集合；

• 用TS.ADD命令插入数据；

• 用TS.GET命令读取最新数据；

• 用TS.MGET命令按标签过滤查询数据集合；

• 用TS.RANGE支持聚合计算的范围查询。

7.4 小结

- [ ] 部署环境中网络带宽高、Redis实例内存大，可以优先考虑第一种方案；

- [ ] 部署环境中网络、内存资源有限，而且数据量大，聚合计算频繁，需要按数据集合属性查询，可以优先考虑第二种方案。