12-有一亿个keys要统计，应该用哪种集合？

你好，我是蒋德钧。

在Web和移动应用的业务场景中，我们经常需要保存这样一种信息：一个key对应了一个数据集合。我举几个例子。

- 手机App中的每天的用户登录信息：一天对应一系列用户ID或移动设备ID；
- 电商网站上商品的用户评论列表：一个商品对应了一系列的评论；
- 用户在手机App上的签到打卡信息：一天对应一系列用户的签到记录；
- 应用网站上的网页访问信息：一个网页对应一系列的访问点击。

我们知道，Redis集合类型的特点就是一个键对应一系列的数据，所以非常适合用来存取这些数据。但是，在这些场景中，除了记录信息，我们往往还需要对集合中的数据进行统计，例如：

- 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。

通常情况下，我们面临的用户数量以及访问量都是巨大的，比如百万、千万级别的用户数量，或者千万级别、甚至亿级别的访问信息。所以，我们必须要选择能够非常高效地统计大量数据（例如亿级）的集合类型。

要想选择合适的集合，我们就得了解常用的集合统计模式。这节课，我就给你介绍集合类型常见的四种统计模式，包括聚合统计、排序统计、二值状态统计和基数统计。我会以刚刚提到的这四个场景为例，和你聊聊在这些统计模式下，什么集合类型能够更快速地完成统计，而且还节省内存空间。掌握了今天的内容，之后再遇到集合元素统计问题时，你就能很快地选出合适的集合类型了。

聚合统计

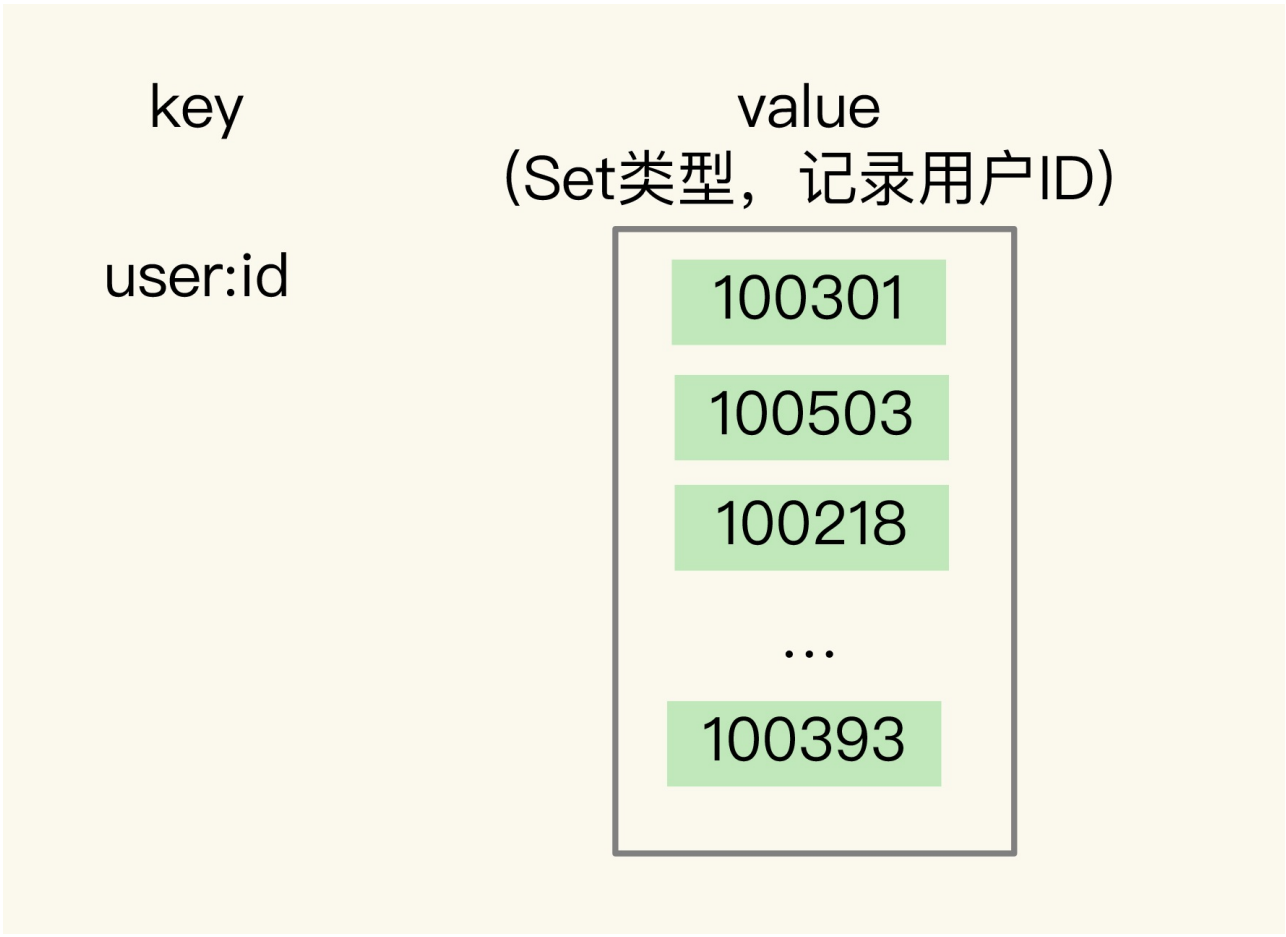
我们先来看集合元素统计的第一个场景：聚合统计。

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（并集统计）。

在刚才提到的场景中，统计手机App每天的新增用户数和第二天的留存用户数，正好对应了聚合统计。

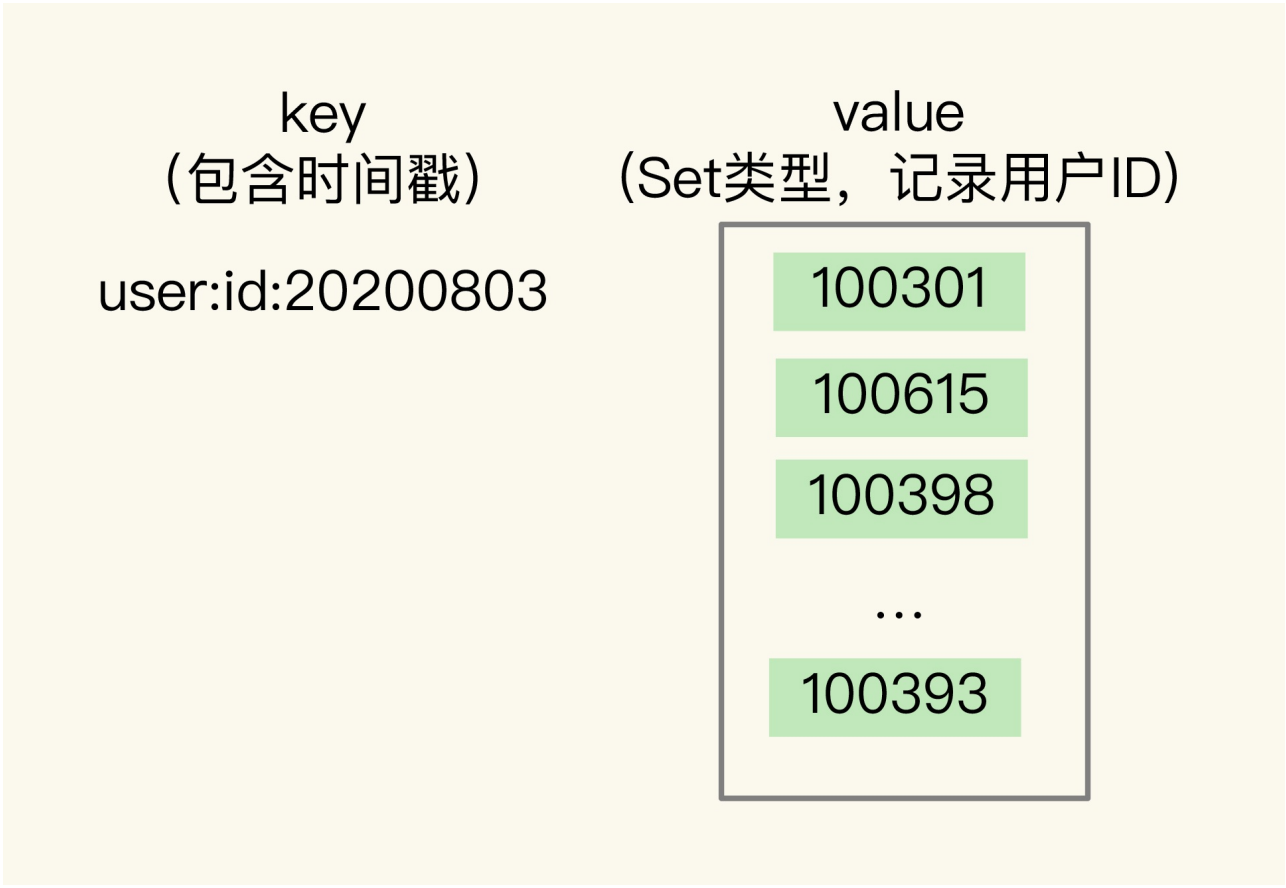
要完成这个统计任务，我们可以用一个集合记录所有登录过App的用户ID，同时，用另一个集合记录每一天登录过App的用户ID。然后，再对这两个集合做聚合统计。我们来看下具体的操作。

记录所有登录过App的用户ID还是比较简单的，我们可以直接使用Set类型，把key设置为user280680，表示记录的是用户ID，value就是一个Set集合，里面是所有登录过App的用户ID，我们可以把这个Set叫作累计用户Set，如下图所示：



需要注意的是，累计用户Set中没有日期信息，我们是不能直接统计每天的新增用户的。所以，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，它有两个特点：

1. key是user280680以及当天日期，例如user280680:20200803；
2. value是Set集合，记录当天登录的用户ID。



在统计每天的新增用户时，我们只用计算每日用户Set和累计用户Set的差集就行。

我借助一个具体的例子来解释一下。

假设我们的手机App在2020年8月3日上线，那么，8月3日前是没有用户的。此时，累计用户Set是空集，当天登录的用户ID会被记录到 key为user280680:20200803的Set中。所以，user280680:20200803这个Set中的用户就是当天的新增用户。

然后，我们计算累计用户Set和user280680:20200803 Set的并集结果，结果保存在user280680这个累计用户Set中，如下所示：

```
SUNIONSTORE  user280680  user280680  user280680:20200803
```

此时，user280680这个累计用户Set中就有了8月3日的用户ID。等到8月4日再统计时，我们把8月4日登录的用户ID记录到user280680:20200804 的Set中。接下来，我们执行SDIFFSTORE命令计算累计用户Set和user280680:20200804 Set的差集，结果保存在key为user:new的Set中，如下所示：

```
SDIFFSTORE  user:new  user280680:20200804 user280680
```

可以看到，这个差集中的用户ID在user280680:20200804 的Set中存在，但是不在累计用户Set中。所以，user:new这个Set中记录的就是8月4日的新增用户。

当要计算8月4日的留存用户时，我们只需要再计算user280680:20200803 和 user280680:20200804两个Set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日留存的用户。执行的命令如下：

```
SINTERSTORE user280680:rem user280680:20200803 user280680:20200804
```

当你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。不过，我要提醒你一下，这里有一个潜在的风险。

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。

排序统计

接下来，我们再来聊一聊应对集合元素排序需求的方法。我以在电商网站上提供最新评论列表的场景为例，

进行讲解。

最新评论列表包含了所有评论中的最新留言，**这就要求集合类型能对元素保序**，也就是说，集合中的元素可以按序排列，这种对元素保序的集合类型叫作有序集合。

在Redis常用的4个集合类型中（List、Hash、Set、Sorted Set），List和Sorted Set就属于有序集合。

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。

看起来好像都可以满足需求，我们该怎么选择呢？

我先说说用List的情况。每个商品对应一个List，这个List包含了对这个商品的所有评论，而且会按照评论时间保存这些评论，每来一个新评论，就用LPUSH命令把它插入List的队头。

在只有一页评论的时候，我们可以很清晰地看到最新的评论，但是，在实际应用中，网站一般会分页显示最新的评论列表，一旦涉及到分页操作，List就可能会出现问题了。

假设当前的评论List是{A, B, C, D, E, F}（其中，A是最新的评论，以此类推，F是最早的评论），在展示第一页的3个评论时，我们可以用下面的命令，得到最新的三条评论A、B、C：

```
LRANGE product1 0 2
1) "A"
2) "B"
3) "C"
```

然后，再用下面的命令获取第二页的3个评论，也就是D、E、F。

```
LRANGE product1 3 5
1) "D"
2) "E"
3) "F"
```

但是，如果在展示第二页前，又产生了一个新评论G，评论G就会被LPUSH命令插入到评论List的队头，评论List就变成了{G, A, B, C, D, E, F}。此时，再用刚才的命令获取第二页评论时，就会发现，评论C又被展示出来了，也就是C、D、E。

```
LRANGE product1 3 5
1) "C"
2) "D"
3) "E"
```

之所以会这样，关键原因就在于，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。

和List相比，Sorted Set就不存在这个问题，因为它是根据元素的实际权重来排序和获取数据的。

我们可以按评论时间的先后给每条评论设置一个权重值，然后再把评论保存到Sorted Set中。Sorted Set的ZRANGEBYSCORE命令就可以按权重排序后返回元素。这样的话，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。

二值状态统计

现在，我们再来分析下第三个场景：二值状态统计。这里的二值状态就是指集合元素的取值就只有0和1两种。在签到打卡的场景中，我们只用记录签到（1）或未签到（0），所以它就是非常典型的二值状态，

在签到统计时，每个用户一天的签到用1个bit位就能表示，一个月（假设是31天）的签到情况用31个bit位就可以，而一年的签到也只需要用365个bit位，根本不用太复杂的集合类型。这个时候，我们就可以选择Bitmap。这是Redis提供的扩展数据类型。我来给你解释一下它的实现原理。

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。你可以把Bitmap看作是一个bit数组。

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，用来统计这个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？我还是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，就可以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 2
```

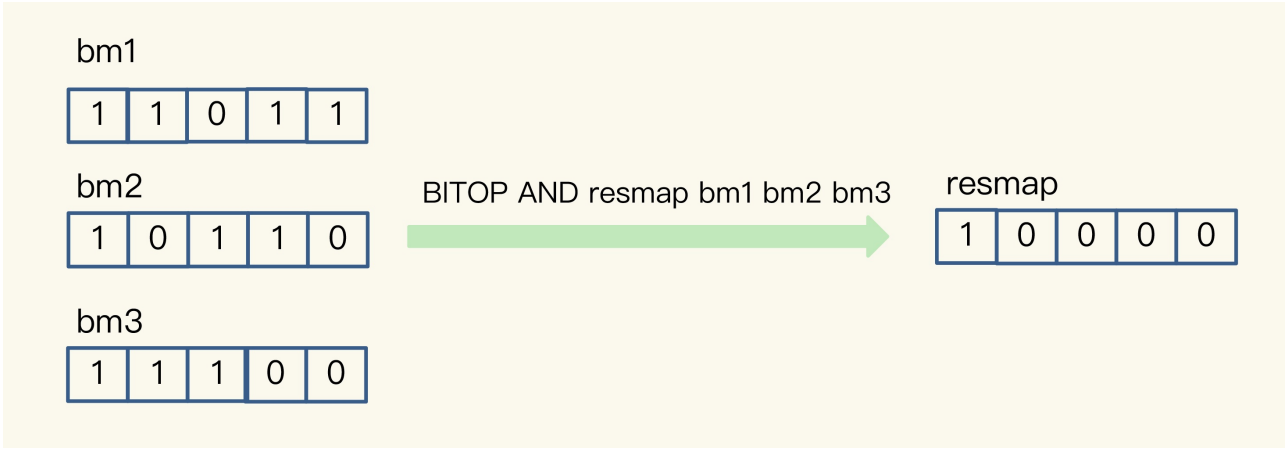
第三步，统计该用户在8月份的签到次数。

```
BITCOUNT uid:sign:3000:202008
```

这样，我们就知道该用户在8月份的签到情况了，是不是很简单呢？接下来，你可以再思考一个问题：如果记录了1亿个用户10天的签到情况，你有办法统计出这10天连续签到的用户总数吗？

在介绍具体的方法之前，我们要先知道，Bitmap支持用BITOP命令对多个Bitmap按位做“与”“或”“异或”的操作，操作的结果会保存到一个新的Bitmap中。

我以按位“与”操作为例来具体解释一下。从下图中，可以看到，三个Bitmap bm1、bm2和bm3，对应bit位做“与”操作，结果保存到了一个新的Bitmap中（示例中，这个结果Bitmap的key被设为“resmap”）。



回到刚刚的问题，在统计1亿个用户连续10天的签到情况时，你可以把每天的日期作为key，每个key对应一个1亿位的Bitmap，每一个bit对应一个用户当天的签到情况。

接下来，我们对10个Bitmap做“与”操作，得到的结果也是一个Bitmap。在这个Bitmap中，只有10天都签到的用户对应的bit位上的值才会是1。最后，我们可以用BITCOUNT统计下Bitmap中的1的个数，这就是连续签到10天的用户总数了。

现在，我们可以计算一下记录了10天签到情况后的内存开销。每天使用1个1亿位的Bitmap，大约占12MB

的内存（10^8/8/1024/1024），10天的Bitmap的内存开销约为120MB，内存压力不算太大。不过，在实际应用时，最好对Bitmap设置过期时间，让Redis自动删除不再需要的签到记录，以节省内存开销。

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，因为它只用一个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

最后，我们再来看一个统计场景：基数统计。基数统计就是指统计一个集合中不重复的元素个数。对应到我们刚才介绍的场景中，就是统计网页的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类型。

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
```

用户1再来访问时，Set的去重功能就保证了不会重复记录用户1的访问次数，这样，用户1就算是一个独立访客。当你需要统计UV时，可以直接用SCARD命令，这个命令会返回一个集合中的元素个数。

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。对于一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，就会消耗很大的内存空间。

当然，你也可以用Hash类型记录UV。

例如，你可以把用户ID作为Hash集合的key，当用户访问页面时，就用HSET命令（用于设置Hash集合元素的值），对这个用户ID记录一个值“1”，表示一个独立访客，用户1访问page1后，我们就记录为1个独立访客，如下所示：

```
HSET page1:uv user1 1
```

即使用户1多次访问页面，重复执行这个HSET命令，也只会把user1的值设置为1，仍然只记为1个独立访客。当要统计UV时，我们可以用HLEN命令统计Hash集合中的所有元素个数。

但是，和Set类型相似，当页面很多时，Hash类型也会消耗很大的内存空间。那么，有什么办法既能完成统计，还能节省内存吗？

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。

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户都添加到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HyperLogLog的统计结果。

```
PFCOUNT page1:uv
```

关于HyperLogLog的具体实现原理，你不需要重点掌握，不会影响到你的日常使用，我就不多讲了。如果你想了解一下，课下可以看看这条链接。

不过，有一点需要你注意一下，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。

小结

这节课，我们结合统计新增用户数和留存用户数、最新评论列表、用户签到数以及网页独立访客量这4种典型场景，学习了集合类型的4种统计模式，分别是聚合统计、排序统计、二值状态统计和基数统计。为了方便你掌握，我把Set、Sorted Set、Hash、List、Bitmap、HyperLogLog的支持情况和优缺点汇总在了下面的表格里，希望你把这张表格保存下来，时不时地复习一下。

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

可以看到，Set和Sorted Set都支持多种聚合统计，不过，对于差集计算来说，只有Set支持。Bitmap也能做多个Bitmap间的聚合计算，包括与、或和异或操作。

当需要进行排序统计时，List中的元素虽然有序，但是一旦有新元素插入，原来的元素在List中的位置就会移动，那么，按位置读取的排序结果可能就不准确了。而Sorted Set本身是按照集合元素的权重排序，可以准确地按序获取结果，所以建议你优先使用它。

如果我们记录的数据只有0和1两个值的状态，Bitmap会是一个很好的选择，这主要归功于Bitmap对于一个数据只用1个bit记录，可以节省内存。

对于基数统计来说，如果集合元素量达到亿级别而且不需要精确统计时，我建议你使用HyperLogLog。

当然，Redis的应用场景非常多，这张表中的总结不一定能覆盖到所有场景。我建议你也试着自己画一张表，把你遇到的其他场景添加进去。长久积累下来，你一定能够更加灵活地把集合类型应用到合适的实践项目中。

每课一问

依照惯例，我给你留个小问题。这节课，我们学习了4种典型的统计模式，以及各种集合类型的支持情况和优缺点，我想请你聊一聊，你还遇到过其他的统计场景吗？用的是怎样的集合类型呢？

欢迎你在留言区写下你的思考和答案，和我交流讨论。如果你身边还有需要解决这些统计问题的朋友或同事，也欢迎你把今天的内容分享给他/她，我们下节课见。

精选留言：

- Kaito 2020-09-02 02:51:00
  使用Sorted Set可以实现统计一段时间内的在线用户数：用户上线时使用zadd online_users $timestamp $user_id把用户添加到Sorted Set中，使用zcount online_users $start_timestamp $end_timestamp就可以得出指定时间段内的在线用户数。

如果key是以天划分的，还可以执行zinterstore online_users_tmp 2 online_users_{date1} online_users_{date2} aggregate max，把结果存储到online_users_tmp中，然后通过zrange online_users_tmp 0 -1 withscores就可以得到这2天都在线过的用户，并且score就是这些用户最近一次的上线时间。

还有一个有意思的方式，使用Set记录数据，再使用zunionstore命令求并集。例如sadd user1 apple orange banana、sadd user2 apple banana peach记录2个用户喜欢的水果，使用zunionstore fruits_union 2 user1 user2把结果存储到fruits_union这个key中，zrange fruits_union 0 -1 withscores可以得出每种水果被喜欢的次数。

使用HyperLogLog计算UV时，补充一点，还可以使用pfcount page1:uv page2:uv page3:uv或pfmerge page_union:uv page1:uv page2:uv page3:uv得出3个页面的UV总和。

另外，需要指出老师文章描述不严谨的地方：“Set数据类型，使用SUNIONSTORE、SDIFFSTORE、SINTERSTORE做并集、差集、交集时，选择一个从库进行聚合计算”。这3个命令都会在Redis中生成一个新key，而从库默认是readonly不可写的，所以这些命令只能在主库使用。想在从库上操作，可以使用SUNION、SDIFF、SINTER，这些命令可以计算出结果，但不会生成新key。

最后需要提醒一下：

1、如果是在集群模式使用多个key聚合计算的命令，一定要注意，因为这些key可能分布在不同的实例上，多个实例之间是无法做聚合运算的，这样操作可能会直接报错或者得到的结果是错误的！

2、当数据量非常大时，使用这些统计命令，因为复杂度较高，可能会有阻塞Redis的风险，建议把这些统计数据与在线业务数据拆分开，实例单独部署，防止在做统计操作时影响到在线业务。 [29赞]

- Darren 2020-09-02 10:44:02

老师说的大部分场景都没用到过。。。。。
我们有这么一种场景：
在多实例下，定时任务就不能使用@Schedule使用，必须使用分布式定时调度，我们自研的分布式调度系统支持MQ和Http两种模式，同时支持一次性的调用和Cron表达是式形式的多次调用。
在MQ模式下（暂时不支持Cron的调用），分布式调度系统作为MQ的消费者消费需要调度的任务，同时消息中会有所使用的资源，调度系统有对应的资源上线，也可以做资源限制，没有可用资源时，消息不调度（不投递）等待之前任务资源的释放，不投递时消息就在Zset中保存着，当然不同的类型在不同的Zset中，当有对用的资源类型释放后，会有专门的MQ确认消息，告诉任务调度系统，某种类型的资源已经释放，然后从对应type的Zset中获取排队中优先级最高的消息，进行资源匹配，如果可以匹配，则进行消息发送。
当然http也是类似的，只是http不做资源管理，业务方自己掌控资源及调用频次，http请求的调用时调度系统自己发起的，引入quartz，在时间到达后，通过Http发送调用。
[3赞]

- 土豆哪里挖 2020-09-02 17:09:45
在集群的情况下，聚合统计就没法用了吧，毕竟不是同一个实例了 [1赞]

- 慎独明强 2020-09-02 08:07:35
有个疑问，统计亿级用户连续10天登录的场景，每天用一个bitmap的key，来存储每个用户的登录情况，将10个bitmap的key进行与运算来统计连续10天登录的用户，这个是怎么保证10个bitmap相同位是同一个用户的登录情况呢？ [1赞]

- 吕 2020-09-03 21:21:29
  我在生产环境下曾经用过bitmap来做，但是由于userId初始值太大了，导致前面的位没有对应用户，但也要占用空间，每天一个key，虽然用户数量不大，但是占用的位数却很多，所以用了两天就放弃了，因为我们的redis内存较小，也想过hash处理一下userId，尽量能够从1开始连续，但是hash以后又怕冲突，所以不知道怎么来处理了

- Wangxi 2020-09-03 10:51:40
  不是很懂 key=user280680 value 是一个set set里面又是用户id。 key不是已经是userId了么。为啥value里面还要存那么多userId干什么

- 波哥威武 2020-09-02 23:17:21
  现在大数据情况下都是通过实时流方式统计pvuv，不太会基于redis，基于存在即合理，老师能分析下相关优劣吗，我个人的想法，一个是在大量pvuv对redis的后端读写压力，还有复杂的统计结果redis也需要复杂的数据结构设计去实现，最后是业务和分析任务解耦。

- Lemon 2020-09-02 19:58:35
  受益良多

- 张三 2020-09-02 12:29:50
  精彩

- Anthony 2020-09-02 09:34:01
  感觉第一个聚合统计这种场景一般频率不会太高，一般都是用在运营统计上，可以直接在mysql的从库上去统计，而不需要在redis上维护复杂的数据结构

- 海拉鲁 2020-09-02 07:06:55
  之前做过利用redis一个统计最近200个客户触达率的方案，借助list+lua
  具体是用0代表触达，1代表未触达，不断丢入队列中。需要统计是lrang key 0 -1 取出全部元素，计算0的比例就是触达率了。
  这样不需要每次都计算一次触达率，而是按需提供，也能保证最新。应该不是很有共性的需求，是我们对用户特定需求的一个尝试