redis利用Hash存储节约内存

Instagram可以说是网拍App的始祖级应用，也是当前最火热的拍照App之一，Instagram的照片数量已经达到3亿，而在Instagram里，我们需要知道每一张照片的作者是谁，下面就是Instagram团队如何使用Redis来解决这个问题并进行内存优化的。

首先，这个通过图片ID反查用户UID的应用有以下几点需求：

1. 查询速度要足够快
2. 数据要能全部放到内存里，最好是一台EC2的high-memory机型就能存储（17GB或者34GB的，68GB的太浪费了）
3. 要合适Instagram现有的架构（Instagram对Redis有一定的使用经验，比如这个应用）
4. 支持持久化，这样在服务器重启后不需要再预热

Instagram的开发者首先否定了数据库存储的方案，他们保持了KISS原则（Keep It Simple and Stupid）,因为这个应用根本用不到数据库的update功能、事务功能和关联查询等等牛X功能，所以不必为这些用不到的功能去选择维护一个数据库。

于是他们选择了Redis，Redis是一个支持持久化的内存数据库，所有的数据都被存储在内存中（忘掉VM吧），而最简单的实现就是使用Redis的String结构来做一个key-value存储就行了。像这样：

1. SET media:1155315 939
2. GET media:1155315
3. > 939

其中1155315是图片ID，939是用户ID，我们将每一张图片ID作为key，用户uid作为value来存成key-value对。然后他们进行了测试，将数据按上面的方法存储，1,000,000数据会用掉70MB内存，300,000,000张照片就会用掉21GB的内存。对比预算的17GB还是超支了。其实这里我们可以看到一个优化，我们可以将key值存成纯数字，经过实验，内存占用会降到50MB，总的内存占用是15GB，是满足需求的，但是Instagram后面的改进仍然有必要。

于是Instagram的开发者向Redis的开发者之一Pieter Noordhuis询问优化方案，得到的回复是使用Hash结构。具体的做法就是将数据分段，每一段使用一个Hash结构存储，由于Hash结构会在单个Hash元素在不足一定数量时进行压缩存储，所以可以大量节约内存。这一点在上面的String结构里是不存在的。而这个一定数量是由配置文件中的hash-zipmao-max-entries参数来控制的。经过开发者们的实验，将hash-zipmap-max-entries设置为1000时，性能比较好，超过1000后HSET命令就会导致CPU消耗变得非常大。

于是他们改变了方案，将数据存成如下结构：

1. HSET "mediabucket:1155" "1155315" "939"
2. HGET "mediabucket:1155" "1155315"
3. > "939"

通过取7位的图片ID的前四位为Hash结构的key值，保证了每个Hash内部只包含3位的key，也就是1000个。

再做一次实验，结果是每1,000,000个key只消耗了16MB。总内存使用也降到了5GB，满足了应用需求。

同样的，这里我们还是可以再进行优化，首先是将Hash结构的key值变成纯数字，其次是将Hash结构中的key值变成三位数，如下所示。经过实验，内存占用量会降到10MB，总内存占用为3GB。

1. HSET "1155" "315" "939"
2. HGET "1155" "315"
3. > "939"

优化无止境，只要肯琢磨。希望你再使用存储产品时也能如此爱惜内存。

附上String和Hash存储的比较，存储对象User String存储方式：

1. SET media:1155315 939
2. GET media:1155315
3. > 939

String结构存储该对象

|  |  |  |  |
| --- | --- | --- | --- |
| 存储量 | 使用内存( KB ) | 使用时间( 毫秒 ) | 使用cpu |
| 100 | 30.72 | 2983 |  |
| 100 | 30.72 | 1224 |  |
| 100 | 40.96 | 2638 |  |
| 100 | 40.96 | 1543 |  |
| 100 | 40.96 | 3335 |  |
| 4487 | 1934.62 | 21760 | 0.05 |
| 4487 | 1934.59 | 21732 | 0.05 |

Hash结构存储该对象

1. HSET "mediabucket:1155" "1155315" "939"
2. HGET "mediabucket:1155" "1155315"
3. > "939"

|  |  |  |  |
| --- | --- | --- | --- |
| 存储量 | 使用内存( KB ) | 使用时间（毫秒） | 使用CPU |
| 100 | 367.76 | 454 |  |
| 100 | 37.37 | 458 |  |
| 100 | 50.50 | 461 |  |
| 100 | 40.44 | 467 |  |
| 100 | 35.50 | 489 |  |
| 4487 | 1805.1 | 21729 | 0.06 |
| 4487 | 1844.23 | 21712 | 0.05 |
| 4487 | 1844.23 | 21778 |  |

Hash结构继续优化，类似：

1. HSET "1155" "315" "939"
2. HGET "1155" "315"
3. > "939"

|  |  |  |  |
| --- | --- | --- | --- |
| 存储量 | 使用内存（KB） | 使用时间（毫秒） | 使用cpu |
| 100 | 367.76 | 454 |  |
| 100 | 37.37 | 458 |  |
| 100 | 50.50 | 461 |  |
| 100 | 40.44 | 467 |  |
| 100 | 35.50 | 489 |  |
| 4487 | 1803.29 | 21879 | 0.06 |
| 4487 | 1842.43 | 21931 | 0.05 |