redis满载压测

- 原因：6G内存的reids在一个月内扩容到24G，且存在重复数据，鉴于此情况，需要对redis进行一次压测，看能否更改为其他策略，如先进先出策略或淘汰最少使用策略。
- 前期调研：调研发现，redis的内存回收策略可以更改，默认是**noeviction；**
-

  redis内存回收策略：

  - **volatile-lru**：采用最近使用最少的淘汰策略，Redis将回收那些超时的（仅仅是超时的）键值对，也就是它只淘汰那些超时的键值对。
  - **allkeys-lru**：采用最近最少使用的淘汰策略，Redis将对所有（不仅仅是超时的）的键值对采用最近最少使用的淘汰策略。
  - **volatile-lfu**：采用最近最不常用的淘汰策略，所谓最近最不常用，也就是一定时期内被访问次数最少的。Redis将回收超时的键值对。
  - **allkeys-lfu**：采用最近最不常用的淘汰策略，Redis将对所有的键值对采用最近最不常用的淘汰策略。
  - **volatile-random**：采用随机淘汰策略删除超时的键值对。
  - **allkeys-random**：采用随机淘汰策略删除所有的键值对，这个策略不常用。
  - **volatile-ttl**：采用删除存活时间最短的键值对策略。
  -

  noeviction：不淘汰任何键值对，当内存满时，如果进行读操作，例如get命令，它将正常工作，而做写操作，它将返回错误，也就是说，当Redis采用这个策略内存达到最大的时候，它就只能读不能写了。

参考：https://www.jianshu.com/p/677930ffbff0

压测方案：

- 将redis的最大内存调整为1mb，通过shell脚本写入数据；

```
查看redis内存相关配置
info memory
设置redis最大内存
config set maxmemory 1mb
查看是否设置成功
config get maxmemory
```

1. 测试是否是LRU（最近最少使用）淘汰策略：分时间段对不同前缀的数据进行调用，再进行写入数据操作，然后查看最早被调用的数据是否存在；
2. 测试是否是先进先出淘汰策略：对满载的redis直接写入数据，看最早写入的数据是否存在；
3. 测试是否是LFU（最近不经常使用）淘汰策略：对数据进行访问，不同前缀的数据访问频率不同，再写入数据，查看被调用频率最少的数据是否存在；

压测过程：

1、回收策略：noeviction (不淘汰任何键值对)

shell脚本：

```
#/bin/bash
for((i=1;i<10000;i++))do
redis-cli set $i $i
echo "get $i"|redis-cli
done;
```

结果：从4097之后开始报OOM

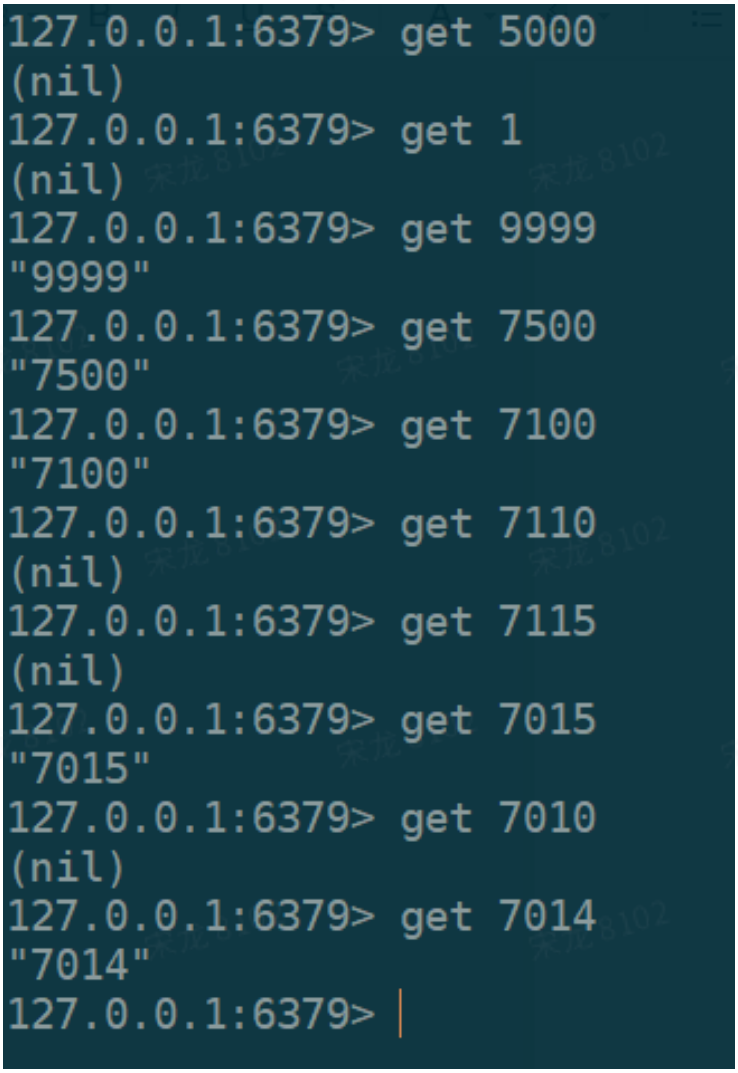
```
OK
"4089"
OK
"4090"
OK
"4091"
OK
"4092"
OK
"4093"
OK
"4094"
OK
"4095"
OK
"4096"
OK
"4097"
(error) OOM command not allowed when used memory > 'maxmemory'.
(nil)
(error) OOM command not allowed when used memory > 'maxmemory'.
(nil)
(error) OOM command not allowed when used memory > 'maxmemory'.
(nil)
(error) OOM command not allowed when used memory > 'maxmemory'.
(nil)
(error) OOM command not allowed when used memory > 'maxmemory'.
(nil)
(error) OOM command not allowed when used memory > 'maxmemory'.
(nil)
(error) OOM command not allowed when used memory > 'maxmemory'.
(nil)
(error) OOM command not allowed when used memory > 'maxmemory'.
(nil)
(error) OOM command not allowed when used memory > 'maxmemory'.
(nil)
(error) OOM command not allowed when used memory > 'maxmemory'.
(nil)
(error) OOM command not allowed when used memory > 'maxmemory'.
(nil)
(error) OOM command not allowed when used memory > 'maxmemory'.
(nil)
```

2、回收策略：allkeys-lru（最近最少使用的淘汰）

shell脚本：

```
#/bin/bash
for((i=1;i<10000;i++))do
redis-cli set $i $i
echo "get $i"|redis-cli结果：没有出现报错情况，脚本执行完之后在redis里执行get操作，发现redis里只有7014-9999这段数据，说明redis执行了最近最少使用的淘汰策略。
done
```

结果：没有出现报错情况，脚本执行完之后在redis里执行get操作，发现redis里只有7014-9999这段数据，说明redis执行了最近最少使用的淘汰策略。



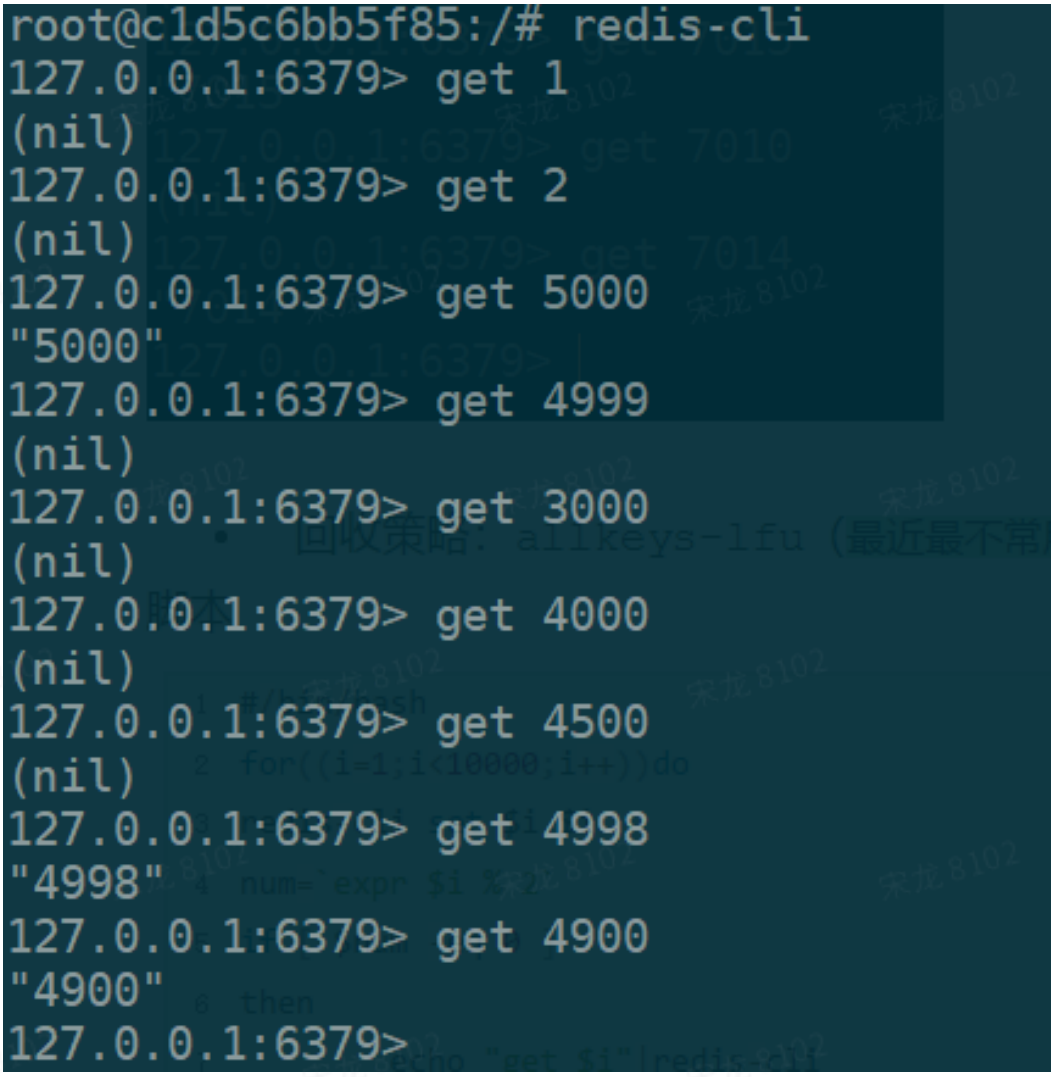
3、回收策略：allkeys-lfu（最近最不常用的淘汰）

shell脚本：

```
1 #/bin/bash
```

```
for((i=1;i<10000;i++))do
redis-cli set $i $i
num=`expr $i % 2`
if [ $num -eq 0 ]
then
 echo "get $i"|redis-cli
fi
done;
```

结果：未出现报错，当超出内存时，优先淘汰掉使用频率低的数据，写入数据超过最大内存时，先淘汰掉没被使用过的单数，然后继续写入数据，再次超过最大内存时，会将最先写入的双数淘汰。

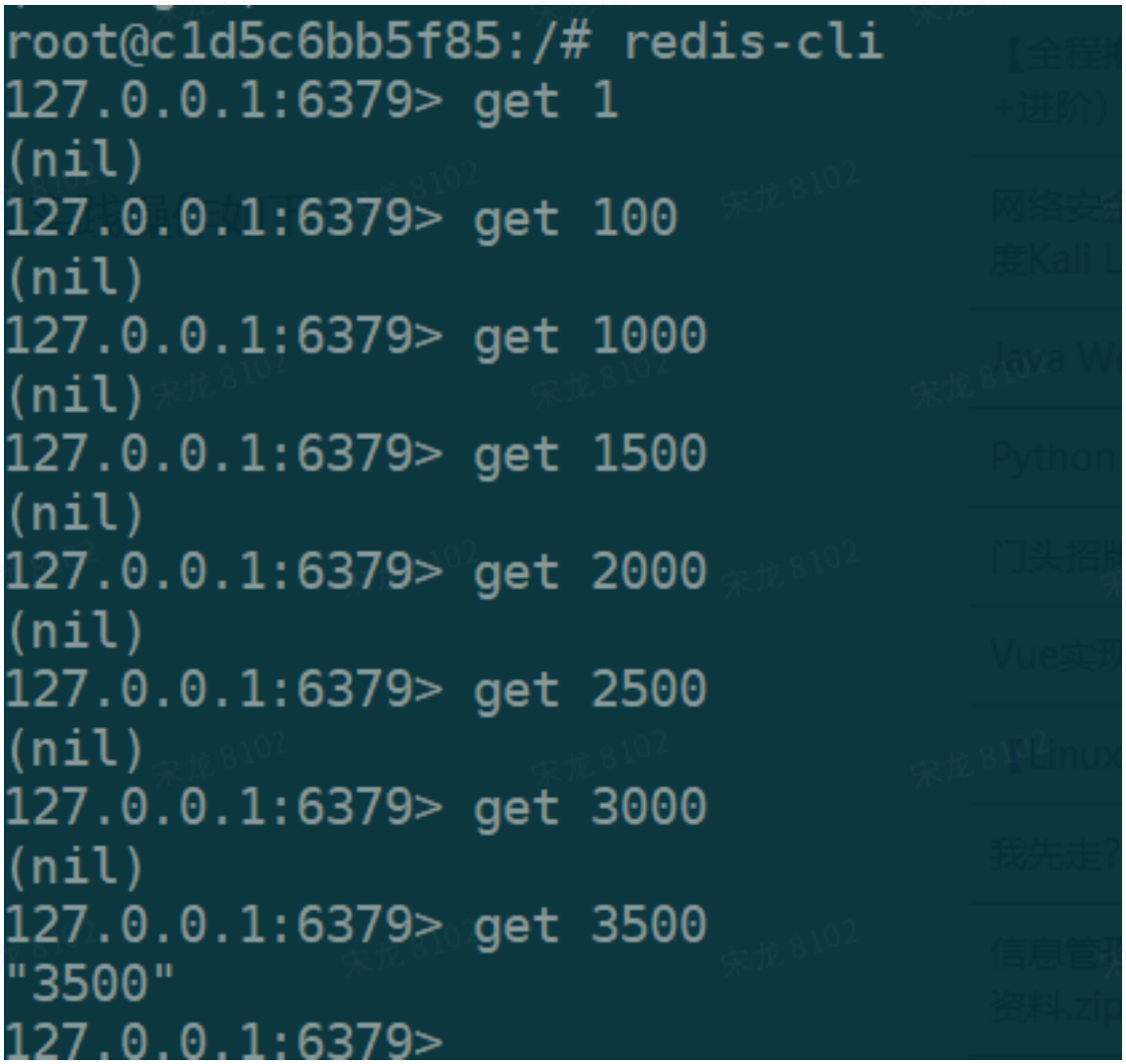


4、回收策略：volatile-ttl 当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，有更早过期时间的key优先移除

shell脚本：

```
#/bin/bash
for((i=1;i<5000;i++))do
redis-cli set $i $i
if [ $i -gt 1000 ]
then
 redis-cli EXPIRE $i 3600
elif [ $i -gt 2000 ]
then
 redis-cli EXPIRE $i 3000
elif [ $i -gt 3000 ]
then
 redis-cli EXPIRE $i 2400
elif [ $i -gt 4000 ]
then
 redis-cli EXPIRE $i 1800
else
 redis-cli EXPIRE $i 1200
fi
done;
```

结果：1-1000，1000-2000，2000-3000段内的数据没有，猜测volatile-ttl策略是优先淘汰过期时间较长的key



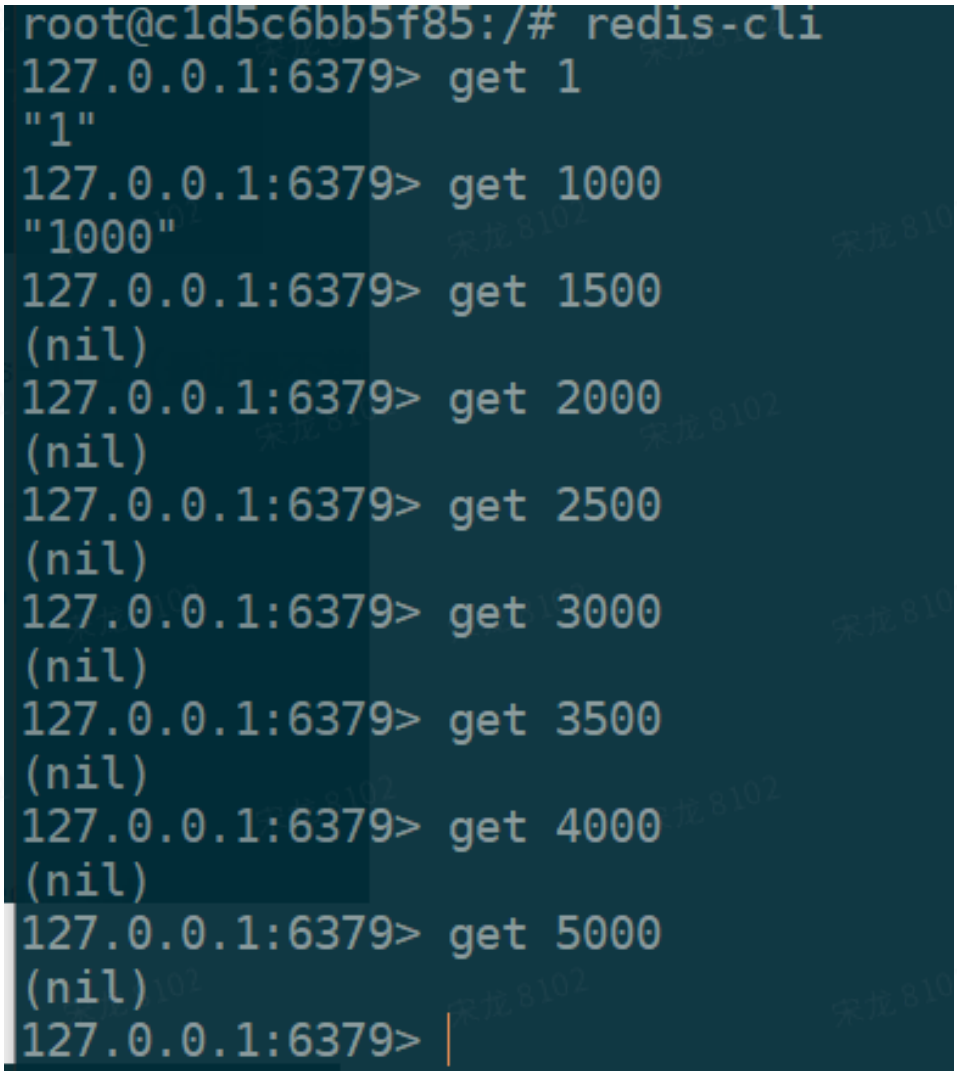
验证以上猜想：

shell脚本：

```
#/bin/bash
for((i=1;i<5000;i++))do
redis-cli set $i $i
if [ $i -gt 1000 ]
then
 redis-cli EXPIRE $i 1200
elif [ $i -gt 2000 ]
then
 redis-cli EXPIRE $i 1800
elif [ $i -gt 3000 ]
then
 redis-cli EXPIRE $i 2400
elif [ $i -gt 4000 ]
then
```

```
 redis-cli EXPIRE $i 3000
else
 redis-cli EXPIRE $i 3600
fi
done;
```

结果：3000-5000段内的数据不存在，1-3000数据存在，验证volatile-ttl策略是优先淘汰过期时间较长的数据；



最终采用volatile-lru策略