1. **Redis--优化详解**

[2016-05-02](https://blog.xiaoxiaomo.com/2016/05/02/Redis-%E4%BC%98%E5%8C%96%E8%AF%A6%E8%A7%A3/)  2527

[Redis](https://blog.xiaoxiaomo.com/tags/Redis/)

1. [优化的一些建议](https://blog.xiaoxiaomo.com/2016/05/02/Redis-%E4%BC%98%E5%8C%96%E8%AF%A6%E8%A7%A3/#%E4%BC%98%E5%8C%96%E7%9A%84%E4%B8%80%E4%BA%9B%E5%BB%BA%E8%AE%AE)
2. [优化实例分析](https://blog.xiaoxiaomo.com/2016/05/02/Redis-%E4%BC%98%E5%8C%96%E8%AF%A6%E8%A7%A3/#%E4%BC%98%E5%8C%96%E5%AE%9E%E4%BE%8B%E5%88%86%E6%9E%90)
   1. [管道性能测试](https://blog.xiaoxiaomo.com/2016/05/02/Redis-%E4%BC%98%E5%8C%96%E8%AF%A6%E8%A7%A3/#%E7%AE%A1%E9%81%93%E6%80%A7%E8%83%BD%E6%B5%8B%E8%AF%95)
   2. [hash的应用](https://blog.xiaoxiaomo.com/2016/05/02/Redis-%E4%BC%98%E5%8C%96%E8%AF%A6%E8%A7%A3/#hash%E7%9A%84%E5%BA%94%E7%94%A8)
   3. [Instagram内存优化](https://blog.xiaoxiaomo.com/2016/05/02/Redis-%E4%BC%98%E5%8C%96%E8%AF%A6%E8%A7%A3/#Instagram%E5%86%85%E5%AD%98%E4%BC%98%E5%8C%96)
3. [启动时WARNING优化](https://blog.xiaoxiaomo.com/2016/05/02/Redis-%E4%BC%98%E5%8C%96%E8%AF%A6%E8%A7%A3/#%E5%90%AF%E5%8A%A8%E6%97%B6WARNING%E4%BC%98%E5%8C%96)

　　本片博客，刚开始会讲解一下redis的**基本优化**，然后会举一些优化**示例代码或实例**。最后讲解一下，默认启动redis时，所报的一些**警示错误**。

1. 优化的一些建议
2. **尽量使用短的key**  
   当然在精简的同时，不要完了key的“见名知意”。对于value有些也可精简，比如性别使用0、1。
3. **避免使用keys \***  
   keys \*, 这个命令是阻塞的，即操作执行期间，其它任何命令在你的实例中都无法执行。当redis中key数据量小时到无所谓，数据量大就很糟糕了。所以我们应该避免去使用这个命令。可以去使用[SCAN](http://www.redis.cn/commands/scan.html),来代替。
4. **在存到Redis之前先把你的数据压缩下**  
   redis为每种数据类型都提供了两种内部编码方式，在不同的情况下redis会自动调整合适的编码方式。
5. **设置**key**有效期**  
   我们应该尽可能的利用key有效期。比如一些临时数据（短信校验码），过了有效期Redis就会自动为你清除！
6. **选择回收策略(maxmemory-policy)**  
   **当 Redis 的实例空间被填满了之后，将会尝试回收一部分key**。根据你的使用方式，强烈建议使用 **volatile-lru（默认）**策略——前提是你对key已经设置了超时。但如果你运行的是一些类似于 cache 的东西，并且没有对 key 设置超时机制，可以考虑使用 **allkeys-lru** 回收机制，[具体讲解查看](http://redis.io/topics/lru-cache#eviction-policies) 。maxmemory-samples 3 是说每次进行淘汰的时候 会随机抽取3个key 从里面淘汰最不经常使用的（默认选项）

maxmemory-policy 六种方式 :  
volatile-lru：只对设置了过期时间的key进行LRU（默认值）  
allkeys-lru ： 是从所有key里 删除 不经常使用的key  
volatile-random：随机删除即将过期key  
allkeys-random：随机删除  
volatile-ttl ： 删除即将过期的  
noeviction ： 永不过期，返回错误

1. 使用bit位级别操作和byte字节级别操作来减少不必要的内存使用。

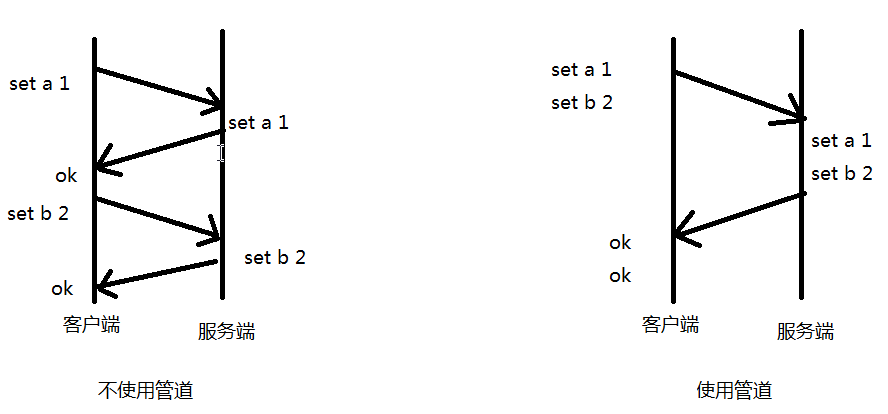
**bit位级别操作**：GETRANGE, SETRANGE, GETBIT and SETBIT  
**byte字节级别操作**：GETRANGE and SETRANGE

1. 尽可能地使用hashes哈希存储。
2. 当业务场景不需要数据持久化时，关闭所有的持久化方式可以获得最佳的性能。
3. 想要一次添加多条数据的时候可以使用管道。
4. **限制redis的内存大小**（64位系统不限制内存，32位系统默认最多使用3GB内存）  
   数据量不可预估，并且内存也有限的话，尽量限制下redis使用的内存大小，这样可以避免redis使用swap分区或者出现OOM错误。（使用swap分区，性能较低，如果限制了内存，当到达指定内存之后就不能添加数据了，否则会报OOM错误。可以设置maxmemory-policy，内存不足时删除数据。）
5. SLOWLOG [get/reset/len]

slowlog-log-slower-than 它决定要对执行时间大于多少微秒(microsecond，1秒 = 1,000,000 微秒)的命令进行记录。  
slowlog-max-len 它决定 slowlog 最多能保存多少条日志，当发现redis性能下降的时候可以查看下是哪些命令导致的。

1. 优化实例分析
2. 管道性能测试

**redis的管道功能在命令行中没有，但是redis是支持管道的，在java的客户端(jedis)中是可以使用的**：



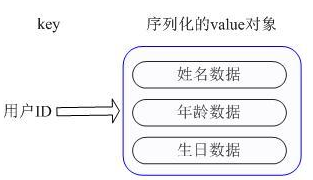
* 示例代码

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 | *//注：具体耗时，和自身电脑有关(博主是在虚拟机中运行的数据)* */\*\**  *\* 不使用管道初始化1W条数据*  *\* 耗时：3079毫秒*  *\* @throws Exception*  *\*/* @Test public void NOTUsePipeline() throws Exception {  Jedis jedis = JedisUtil.getJedis();  long start\_time = System.currentTimeMillis();  for (int i = 0; i < 10000; i++) {  jedis.set("aa\_"+i, i+"");  }  System.out.println(System.currentTimeMillis()-start\_time); }  */\*\**  *\* 使用管道初始化1W条数据*  *\* 耗时：255毫秒*  *\* @throws Exception*  *\*/* @Test public void usePipeline() throws Exception {  Jedis jedis = JedisUtil.getJedis();   long start\_time = System.currentTimeMillis();  Pipeline pipelined = jedis.pipelined();  for (int i = 0; i < 10000; i++) {  pipelined.set("cc\_"+i, i+"");  }  pipelined.sync();*//执行管道中的命令*  System.out.println(System.currentTimeMillis()-start\_time); } |

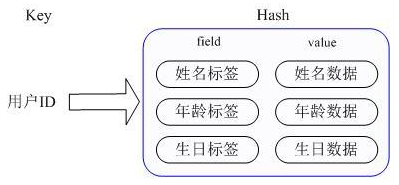
1. hash的应用

**示例**：我们要存储一个用户信息对象数据，包含以下信息：  
key为用户ID，value为用户对象（姓名，年龄，生日等）如果用普通的key/value结构来存储，主要有以下2种存储方式：

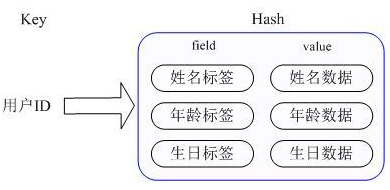
1. 将用户ID作为查找key,把其他信息封装成一个对象以**序列化的方式存储**  
   缺点：增加了序列化/反序列化的开销，引入复杂适应系统（Complex adaptive system，简称[CAS](https://www.douban.com/group/topic/19342736/)）修改其中一项信息时，需要把整个对象取回，并且修改操作需要对并发进行保护。



1. 用户信息**对象有多少成员就存成多少个key-value对**  
   虽然省去了序列化开销和并发问题，但是用户ID为重复存储。



* Redis提供的Hash很好的解决了这个问题，**提供了直接存取这个Map成员的接口**。**Key仍然是用户ID, value是一个Map**，这个Map的key是成员的属性名，value是属性值。(***内部实现****：Redis Hashd的Value内部有2种不同实现，Hash的成员比较少时Redis为了节省内存会采用类似一维数组的方式来紧凑存储，而不会采用真正的HashMap结构，对应的value redisObject的encoding为zipmap,当成员数量增大时会自动转成真正的HashMap,此时encoding为ht* )。



1. Instagram内存优化

Instagram可能大家都已熟悉，当前火热的拍照App，月活跃用户3亿。四年前Instagram所存图片3亿多时需要解决一个问题：想知道每一张照片的作者是谁（通过图片ID反查用户UID），并且要求查询速度要相当的块，如果把它放到内存中使用String结构做key-value:

|  |  |
| --- | --- |
| 1 2 3 | HSET "mediabucket:1155" "1155315" "939" HGET "mediabucket:1155" "1155315" "939" |

**测试**：1百万数据会用掉70MB内存，3亿张照片就会用掉21GB的内存。当时(四年前)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）,想把它放到16G机型中还是不行的。

* 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-zipmap-max-entries参数来控制的。**经过实验，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**。

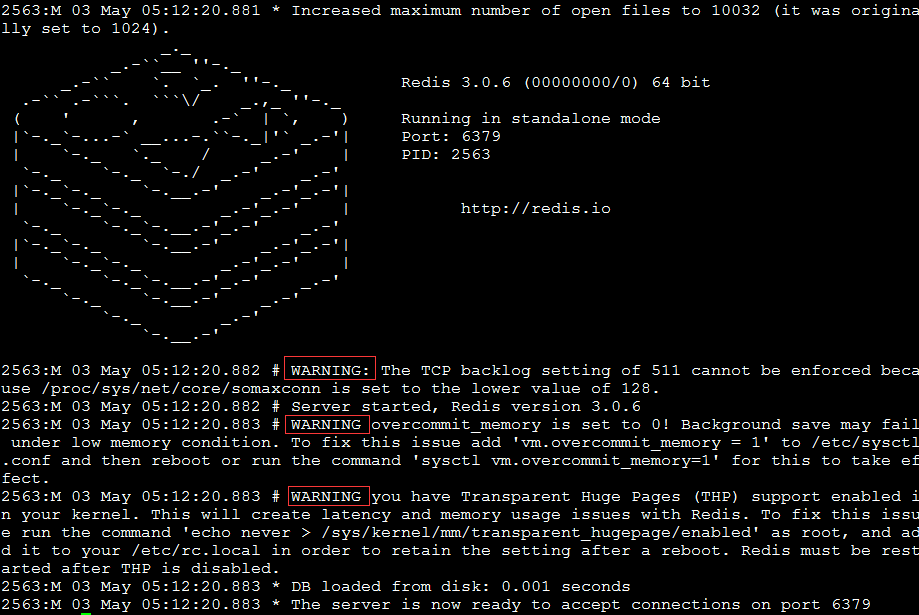
|  |  |
| --- | --- |
| 1 2 3 | HSET "mediabucket:1155" "1155315" "939" HGET "mediabucket:1155" "1155315" "939" |

**测试**：1百万消耗16MB的内存。总内存使用也降到了5GB。当然我们还可以优化，去掉mediabucket:key长度减少了12个字节。

|  |  |
| --- | --- |
| 1 2 3 | HSET "1155" "315" "939" HGET "1155" "315" "939" |

1. 启动时WARNING优化

在我们启动redis时，默认会出现如下三个警告：



* 一、修改**linux**中**TCP**监听的最大容纳数量

|  |  |
| --- | --- |
| 1 2 | WARNING: The TCP backlog setting of 511 cannot be enforced because  /proc/sys/net/core/somaxconn is set to the lower value of 128. |

**在高并发环境下你需要一个高backlog值来避免慢客户端连接问题**。注意Linux内核默默地将这个值减小到/proc/sys/net/core/somaxconn的值，所以需要确认增大somaxconn和tcp\_max\_syn\_backlog两个值来达到想要的效果。  
echo 511 > /proc/sys/net/core/somaxconn  
**注意**：这个参数并不是限制redis的最大链接数。如果想限制redis的最大连接数需要修改maxclients，默认最大连接数为10000

* 二、修改linux内核内存分配策略

|  |  |
| --- | --- |
| 1 2 3 | 错误日志：WARNING overcommit\_memory is set to 0! Background save may fail under low memory condition.  To fix this issue add 'vm.overcommit\_memory = 1' to /etc/sysctl.conf and then reboot or  run the command 'sysctl vm.overcommit\_memory=1' |

原因：  
**redis在备份数据的时候，会fork出一个子进程，理论上child进程所占用的内存和parent是一样的，比如parent占用的内存为8G，这个时候也要同样分配8G的内存给child,如果内存无法负担，往往会造成redis服务器的down机或者IO负载过高，效率下降**。所以内存分配策略应该设置为 1（表示内核允许分配所有的物理内存，而不管当前的内存状态如何）。  
内存分配策略有三种  
可选值：0、1、2。  
0， 表示内核将检查是否有足够的可用内存供应用进程使用；如果有足够的可用内存，内存申请允许；否则，内存申请失败，并把错误返回给应用进程。  
1， 不管需要多少内存，都允许申请。  
2， 只允许分配物理内存和交换内存的大小(交换内存一般是物理内存的一半)。

* 三、关闭Transparent Huge Pages(THP)  
  THP会造成内存锁影响redis性能，建议关闭

Transparent HugePages ：用来提高内存管理的性能  
Transparent Huge Pages在32位的RHEL 6中是不支持的

执行命令

echo never > /sys/kernel/mm/transparent\_hugepage/enabled  
把这条命令添加到这个文件中/etc/rc.local