**缓存穿透**

缓存和数据库中都没有数据，用户不断发起没有没有数据的请求，会导致数据库压力过大。

可能是攻击者。

解决办法：

1、增加参数合法性校验

2、缓存没有的数据key-null

**缓存击穿**

缓存中没数据，数据库中有数据。大量请求会导致数据库压力过大。

可能数据过期。

解决办法：

1、设置热点数据永不过期

2、可以借用单利模式的思想。双重检查，来实现只有一个请求会到数据库。

**缓存雪崩** （这里不说系统雪崩）

缓存中大量数据集中过期，查询数据量巨大，引起数据库机器压力过大。

解决办法：

1、缓存数据过期时间设置随机数，防止大量数据集中过期。

2、设置热点数据不过期

参考文档：

<https://www.cnblogs.com/liushoudong/p/12679174.html>

redis删除策略

过期数据的删除方案

定时删除+定期删除+惰性删除

redis内存淘汰机制

新写入数据时，没有内存空间，这时需要将redis内存中不怎么使用的数据换到磁盘。

在redis.conf中配置

配置：maxmemory-policy volatile-lru

检测易失性数据（可能会过期的数据集server.db[i].expires）

volatile-lru  -->  从已设置过期时间的数据集中挑选最近最少使用的数据淘汰

volatile-lfu-->从已设置过期时间的数据集中挑选最不经常使用的数据淘汰

volatile-ttl-->从已设置过期时间的数据集中挑选将要过期的数据淘汰

volatile-random -->从已设置过期时间的数据集中任意选择数据淘汰

检测全库数据（所有数据集server.db[i].dict）

allkeys-lru  -->  当内存不足以容纳新写入数据时，在键空间中，移除最近最少使用的key（最常用）

allkeys-random-->从数据集中任意选择数据淘汰

allkeys-lfu-->当内存不足以容纳新写入数据时，在键空间中，移除最不经常使用的key

放弃数据驱逐

no-eviction-->禁止驱逐数据（redis4.0默认策略），也就是说当内存不足以容纳新写入数据时，新写入操作或报错，回引发OOM（Out of memory）

可以通过info命令查看缓存命中次数

延时队列

1、RabbitMQ可以实现

Map<String, Object> *argsMap* = new HashMap();

*argsMap*.put("x-message-ttl", 600000);  
*argsMap*.put("x-dead-letter-exchange", userpartiExchange);  
*argsMap*.put("x-dead-letter-routing-key", userpartiRoutingKey);

channel.queueDeclare(userpartiDelayQueue, true, false, false, argsMap);

2、redis也可以，用SortedSet，score存消息时间戳

zrangebyscore 获取n秒之前的数据。

redis内存优化

redis中的所有key和value都是redisObject

object查看key的RedisObject信息

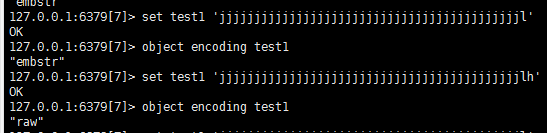
127.0.0.1:6379[7]> object encoding LIVE:LIKE:978

"int"

OBJECT 命令有多个子命令：

* OBJECT REFCOUNT <key> 返回给定 key 引用所储存的值的次数。此命令主要用于除错。
* OBJECT ENCODING <key> 返回给定 key 锁储存的值所使用的内部表示(representation)。
* OBJECT IDLETIME <key> 返回给定 key 自储存以来的空转时间(idle， 没有被读取也没有被写入)，以秒为单位。

字符串长度小于45，encoding 是embstr



是embstr类型时，字符串SDS和redisObject一起分配。值操作数据库一次。

高并发场景下，控制字符串不要太长。

参考文档：

<https://www.jianshu.com/p/666452a22855>

缩减key-value对象

共享整数对象池[0,9999]。在开发中能用整数就用整数，来节省内存

redis内存优化

一.redisObject对象

二.缩减键值对象

三.共享对象池

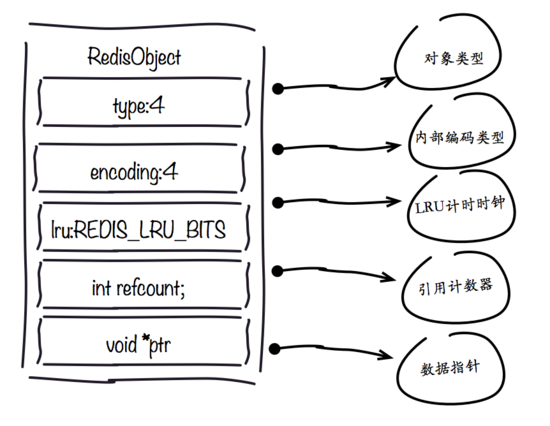
四.字符串优化

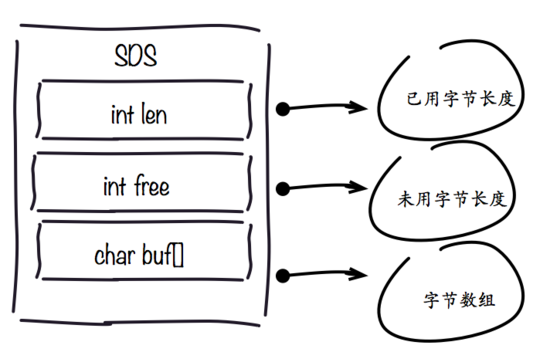
五.编码优化

六.控制key的数量

参考文档：

<https://www.cnblogs.com/williamjie/p/11288062.html>





SDS的优点：

* O(1)时间复杂度获取：字符串长度，已用长度，未用长度。
* 可用于保存字节数组，支持安全的二进制数据存储。
* 内部实现空间预分配机制，降低内存再分配次数。
* 惰性删除机制，字符串缩减后的空间不释放，作为预分配空间保留。

预分配机制：

1) 第一次创建len属性等于数据实际大小，free等于0，不做预分配。

2) 修改后如果已有free空间不够且数据小于1M，每次预分配一倍容量。

3) 修改后如果已有free空间不够且数据大于1MB，每次预分配1MB数据。

采用预分配的道理

因为字符串(SDS)存在预分配机制，日常开发中要小心预分配带来的内存浪费，例如下表的测试用例。

开发提示:尽量减少字符串频繁修改操作如append，setrange, 改为直接使用set修改字符串，降低预分配带来的内存浪费和内存碎片化

用hash来存储java对象类型的数据。还支持部分属性操作。

hash-max-ziplist-value 默认64

encoding是redis底层使用的数据结构。编码不同影响内存的占用和读写效率

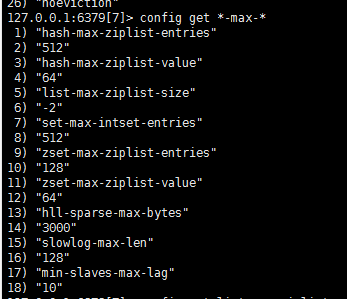
object encoding key

**表：type和encoding对应关系表**

| **类型** | **编码方式** | **数据结构** |
| --- | --- | --- |
| string | raw  embstr  int | 动态字符串编码  优化内存分配的字符串编码  整数编码 |
| hash | hashtable  ziplist | 散列表编码  压缩列表编码 |
| list | linkedlist  ziplist  quicklist | 双向链表编码  压缩列表编码  3.2版本新的列表编码 |
| set | hashtable  intset | 散列表编码  整数集合编码 |
| zset | skiplist  ziplist | 跳跃表编码  压缩列表编码 |

控制编码类型

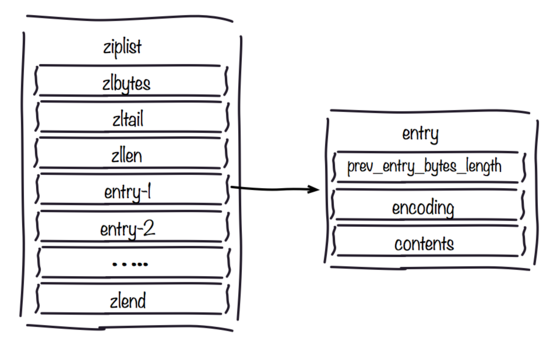
查看redis当前版本支持的配置参数使用命令：config get \*



可以设置参数使用命令 config set param paramvalue

**hash,list,set,zset内部编码配置**

| **类型** | **编码** | **决定条件** |
| --- | --- | --- |
| hash | ziplist | 满足所有条件:  value最大空间(字节)<=hash-max-ziplist-value  field个数<=hash-max-ziplist-entries |
| 同上 | hashtable | 满足任意条件:  value最大空间(字节)>hash-max-ziplist-value  field个数>hash-max-ziplist-entries |
| list | ziplist | ziplist 满足所有条件:  value最大空间(字节)<=list-max-ziplist-value  链表长度<=list-max-ziplist-entries |
| 同上 | linkedlist | 满足任意条件  value最大空间(字节)>list-max-ziplist-value  链表长度>list-max-ziplist-entries |
| 同上 | quicklist | 3.2版本新编码:  废弃list-max-ziplist-entries和list-max-ziplist-entries配置  使用新配置:  list-max-ziplist-size:表示最大压缩空间或长度,最大空间使用[-5-1]范围配置，默认-2表示8KB,正整数表示最大压缩长度  list-compress-depth:表示最大压缩深度，默认=0不压缩 |
| set | intset | 满足所有条件:  元素必须为整数  集合长度<=set-max-intset-entries |
| 同上 | hashtable | 满足任意条件  元素非整数类型  集合长度>hash-max-ziplist-entries |
| zset | ziplist | 满足所有条件:  value最大空间(字节)<=zset-max-ziplist-value  有序集合长度<=zset-max-ziplist-entries |
| 同上 | skiplist | 满足任意条件:  value最大空间(字节)>zset-max-ziplist-value  有序集合长度>zset-max-ziplist-entries |

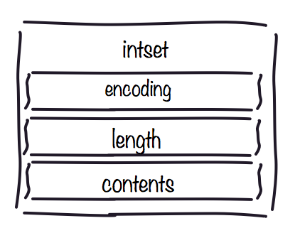


ziplist结构字段含义：

* 1) zlbytes:记录整个压缩列表所占字节长度，方便重新调整ziplist空间。类型是int-32，长度为4字节
* 2) zltail:记录距离尾节点的偏移量，方便尾节点弹出操作。类型是int-32，长度为4字节
* 3) zllen:记录压缩链表节点数量，当长度超过216-2时需要遍历整个列表获取长度，一般很少见。类型是int-16，长度为2字节
* 4) entry:记录具体的节点，长度根据实际存储的数据而定。
  + a) prev\_entry\_bytes\_length:记录前一个节点所占空间，用于快速定位上一个节点，可实现列表反向迭代。
  + b) encoding:标示当前节点编码和长度，前两位表示编码类型：字符串/整数，其余位表示数据长度。
  + c) contents:保存节点的值，针对实际数据长度做内存占用优化。
* 5) zlend:记录列表结尾，占用一个字节。

根据以上对ziplist字段说明，可以分析出该数据结构特点如下:

* 1) 内部表现为数据紧凑排列的一块连续内存数组。
* 2) 可以模拟双向链表结构，以O(1)时间复杂度入队和出队。
* 3) 新增删除操作涉及内存重新分配或释放，加大了操作的复杂性。
* 4) 读写操作涉及复杂的指针移动，最坏时间复杂度为O(n2)。
* 5) 适合存储小对象和长度有限的数据。



intset的字段结构含义：

1) encoding:整数表示类型，根据集合内最长整数值确定类型，整数类型划分三种:int-16，int-32，int-64。

2) length:表示集合元素个数。

3) contents:整数数组，按从小到大顺序保存。

intset保存的整数类型根据长度划分，当保存的整数超出当前类型时，将会触发自动升级操作且升级后不再做回退。升级操作将会导致重新申请内存空间，把原有数据按转换类型后拷贝到新数组。

开发提示：使用intset编码的集合时，尽量保持整数范围一致，如都在int-16范围内。防止个别大整数触发集合升级操作，产生内存浪费。

过多的键同样会消耗大量内存。使用Redis时不要进入一个误区，大量使用get/set这样的API，把Redis当成Memcached使用。对于存储相同的数据内容利用Redis的数据结构降低外层键的数量，也可以节省大量内存。

用hash来降低最外层key的数量