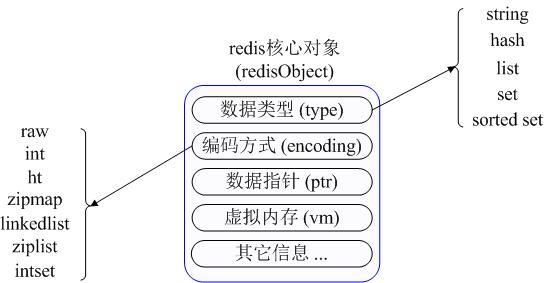
# Redis内存使用优化与存储

## Redis常用数据类型

Redis最为常用的数据类型主要有以下五种：

* String
* Hash
* List
* Set
* Sorted set

在具体描述这几种数据类型之前，我们先通过一张图了解下Redis内部内存管理中是如何描述这些不同数据类型的：



首先Redis内部使用一个redisObject对象来表示所有的key和value,redisObject最主要的信息如上图所示：type代表一个value对象具体是何种数据类型，encoding是不同数据类型在redis内部的存储方式，比如：type=string代表value存储的是一个普通字符串，那么对应的encoding可以是raw或者是int,如果是int则代表实际redis内部是按数值型类存储和表示这个字符串的，当然前提是这个字符串本身可以用数值表示，比如:"123" "456"这样的字符串。

这里需要特殊说明一下vm字段，只有打开了Redis的虚拟内存功能，此字段才会真正的分配内存，该功能默认是关闭状态的，该功能会在后面具体描述。通过上图我们可以发现Redis使用redisObject来表示所有的key/value数据是比较浪费内存的，当然这些内存管理成本的付出主要也是为了给Redis不同数据类型提供一个统一的管理接口，实际作者也提供了多种方法帮助我们尽量节省内存使用，我们随后会具体讨论。

下面我们先来逐一的分析下这五种数据类型的使用和内部实现方式：

* String

**常用命令：**

set,get,decr,incr,mget 等。

**应用场景：**

String是最常用的一种数据类型，普通的key/value存储都可以归为此类，这里就不所做解释了。

**实现方式：**

String在redis内部存储默认就是一个字符串，被redisObject所引用，当遇到incr,decr等操作时会转成数值型进行计算，此时redisObject的encoding字段为int。

* Hash

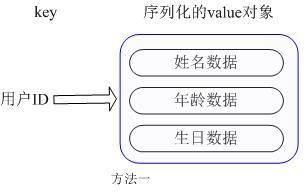
**常用命令：**

hget,hset,hgetall 等。

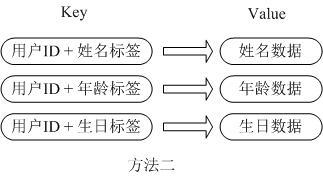
**应用场景：**

我们简单举个实例来描述下Hash的应用场景，比如我们要存储一个用户信息对象数据，包含以下信息：

用户ID为查找的key，存储的value用户对象包含姓名，年龄，生日等信息，如果用普通的key/value结构来存储，主要有以下2种存储方式：

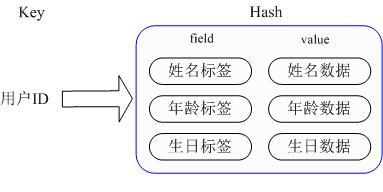


第一种方式将用户ID作为查找key,把其他信息封装成一个对象以序列化的方式存储，这种方式的缺点是，增加了序列化/反序列化的开销，并且在需要修改其中一项信息时，需要把整个对象取回，并且修改操作需要对并发进行保护，引入CAS等复杂问题。



第二种方法是这个用户信息对象有多少成员就存成多少个key-value对儿，用用户ID+对应属性的名称作为唯一标识来取得对应属性的值，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，如果存在大量这样的数据，内存浪费还是非常可观的。

那么Redis提供的Hash很好的解决了这个问题，Redis的Hash实际是内部存储的Value为一个HashMap，并提供了直接存取这个Map成员的接口，如下图：



也就是说，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，这样对数据的修改和存取都可以直接通过其内部Map的Key(Redis里称内部Map的key为field), 也就是通过 key(用户ID) + field(属性标签) 就可以操作对应属性数据了，既不需要重复存储数据，也不会带来序列化和并发修改控制的问题。很好的解决了问题。

这里同时需要注意，Redis提供了接口(hgetall)可以直接取到全部的属性数据,但是如果内部Map的成员很多，那么涉及到遍历整个内部Map的操作，由于Redis单线程模型的缘故，这个遍历操作可能会比较耗时，而另其它客户端的请求完全不响应，这点需要格外注意。

**实现方式：**

上面已经说到Redis Hash对应Value内部实际就是一个HashMap，实际这里会有2种不同实现，这个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。

* List

**常用命令：**

lpush,rpush,lpop,rpop,lrange等。

**应用场景：**

Redis list的应用场景非常多，也是Redis最重要的数据结构之一，比如twitter的关注列表，粉丝列表等都可以用Redis的list结构来实现，比较好理解，这里不再重复。

**实现方式：**

Redis list的实现为一个双向链表，即可以支持反向查找和遍历，更方便操作，不过带来了部分额外的内存开销，Redis内部的很多实现，包括发送缓冲队列等也都是用的这个数据结构。

* Set

**常用命令：**

sadd,spop,smembers,sunion 等。

**应用场景：**

Redis set对外提供的功能与list类似是一个列表的功能，特殊之处在于set是可以自动排重的，当你需要存储一个列表数据，又不希望出现重复数据时，set是一个很好的选择，并且set提供了判断某个成员是否在一个set集合内的重要接口，这个也是list所不能提供的。

**实现方式：**

set 的内部实现是一个 value永远为null的HashMap，实际就是通过计算hash的方式来快速排重的，这也是set能提供判断一个成员是否在集合内的原因。

* Sorted set

**常用命令：**

zadd,zrange,zrem,zcard等

**使用场景：**

Redis sorted set的使用场景与set类似，区别是set不是自动有序的，而sorted set可以通过用户额外提供一个优先级(score)的参数来为成员排序，并且是插入有序的，即自动排序。当你需要一个有序的并且不重复的集合列表，那么可以选择sorted set数据结构，比如twitter 的public timeline可以以发表时间作为score来存储，这样获取时就是自动按时间排好序的。

**实现方式：**

Redis sorted set的内部使用HashMap和跳跃表(SkipList)来保证数据的存储和有序，HashMap里放的是成员到score的映射，而跳跃表里存放的是所有的成员，排序依据是HashMap里存的score,使用跳跃表的结构可以获得比较高的查找效率，并且在实现上比较简单。

## 常用内存优化手段与参数

通过我们上面的一些实现上的分析可以看出redis实际上的内存管理成本非常高，即占用了过多的内存，作者对这点也非常清楚，所以提供了一系列的参数和手段来控制和节省内存，我们分别来讨论下。

首先最重要的一点是不要开启Redis的VM选项，即虚拟内存功能，这个本来是作为Redis存储超出物理内存数据的一种数据在内存与磁盘换入换出的一个持久化策略，但是其内存管理成本也非常的高，并且我们后续会分析此种持久化策略并不成熟，所以要关闭VM功能，请检查你的redis.conf文件中 vm-enabled 为 no。

其次最好设置下redis.conf中的maxmemory选项，该选项是告诉Redis当使用了多少物理内存后就开始拒绝后续的写入请求，该参数能很好的保护好你的Redis不会因为使用了过多的物理内存而导致swap,最终严重影响性能甚至崩溃。

另外Redis为不同数据类型分别提供了一组参数来控制内存使用，我们在前面详细分析过Redis Hash是value内部为一个HashMap，如果该Map的成员数比较少，则会采用类似一维线性的紧凑格式来存储该Map, 即省去了大量指针的内存开销，这个参数控制对应在redis.conf配置文件中下面2项：

hash-max-zipmap-entries 64

hash-max-zipmap-value 512

hash-max-zipmap-entries

含义是当value这个Map内部不超过多少个成员时会采用线性紧凑格式存储，默认是64,即value内部有64个以下的成员就是使用线性紧凑存储，超过该值自动转成真正的HashMap。

hash-max-zipmap-value 含义是当 value这个Map内部的每个成员值长度不超过多少字节就会采用线性紧凑存储来节省空间。

以上2个条件任意一个条件超过设置值都会转换成真正的HashMap，也就不会再节省内存了，那么这个值是不是设置的越大越好呢，答案当然是否定的，HashMap的优势就是查找和操作的时间复杂度都是O(1)的，而放弃Hash采用一维存储则是O(n)的时间复杂度，如果

成员数量很少，则影响不大，否则会严重影响性能，所以要权衡好这个值的设置，总体上还是最根本的时间成本和空间成本上的权衡。

同样类似的参数还有：

list-max-ziplist-entries 512

说明：list数据类型多少节点以下会采用去指针的紧凑存储格式。

list-max-ziplist-value 64

说明：list数据类型节点值大小小于多少字节会采用紧凑存储格式。

set-max-intset-entries 512

说明：set数据类型内部数据如果全部是数值型，且包含多少节点以下会采用紧凑格式存储。

最后想说的是Redis内部实现没有对内存分配方面做过多的优化，在一定程度上会存在内存碎片，不过大多数情况下这个不会成为Redis的性能瓶颈，不过如果在Redis内部存储的大部分数据是数值型的话，Redis内部采用了一个shared integer的方式来省去分配内存的开销，即在系统启动时先分配一个从1~n 那么多个数值对象放在一个池子中，如果存储的数据恰好是这个数值范围内的数据，则直接从池子里取出该对象，并且通过引用计数的方式来共享，这样在系统存储了大量数值下，也能一定程度上节省内存并且提高性能，这个参数值n的设置需要修改源代码中的一行宏定义REDIS\_SHARED\_INTEGERS，该值默认是10000，可以根据自己的需要进行修改，修改后重新编译就可以了。

## Redis的持久化机制

Redis由于支持非常丰富的内存数据结构类型，如何把这些复杂的内存组织方式持久化到磁盘上是一个难题，所以Redis的持久化方式与传统数据库的方式有比较多的差别，Redis一共支持四种持久化方式，分别是：

* 定时快照方式(snapshot)
* 基于语句追加文件的方式(aof)
* 虚拟内存(vm)
* Diskstore方式

在设计思路上，前两种是基于全部数据都在内存中，即小数据量下提供磁盘落地功能，而后两种方式则是作者在尝试存储数据超过物理内存时，即大数据量的数据存储，截止到本文，后两种持久化方式仍然是在实验阶段，并且vm方式基本已经被作者放弃，所以实际能在生产环境用的只有前两种，换句话说Redis目前还只能作为小数据量存储（全部数据能够加载在内存中），海量数据存储方面并不是Redis所擅长的领域。下面分别介绍下这几种持久化方式：

**定时快照方式(snapshot)：**

该持久化方式实际是在Redis内部一个定时器事件，每隔固定时间去检查当前数据发生的改变次数与时间是否满足配置的持久化触发的条件，如果满足则通过操作系统fork调用来创建出一个子进程，这个子进程默认会与父进程共享相同的地址空间，这时就可以通过子进程来遍历整个内存来进行存储操作，而主进程则仍然可以提供服务，当有写入时由操作系统按照内存页(page)为单位来进行copy-on-write保证父子进程之间不会互相影响。

该持久化的主要缺点是定时快照只是代表一段时间内的内存映像，所以系统重启会丢失上次快照与重启之间所有的数据。

**基于语句追加方式(aof)：**

aof方式实际类似mysql的基于语句的binlog方式，即每条会使Redis内存数据发生改变的命令都会追加到一个log文件中，也就是说这个log文件就是Redis的持久化数据。

aof的方式的主要缺点是追加log文件可能导致体积过大，当系统重启恢复数据时如果是aof的方式则加载数据会非常慢，几十G的数据可能需要几小时才能加载完，当然这个耗时并不是因为磁盘文件读取速度慢，而是由于读取的所有命令都要在内存中执行一遍。另外由于每条命令都要写log,所以使用aof的方式，Redis的读写性能也会有所下降。

**虚拟内存方式：**

虚拟内存方式是Redis来进行用户空间的数据换入换出的一个策略，此种方式在实现的效果上比较差，主要问题是代码复杂，重启慢，复制慢等等，目前已经被作者放弃。

**diskstore方式：**

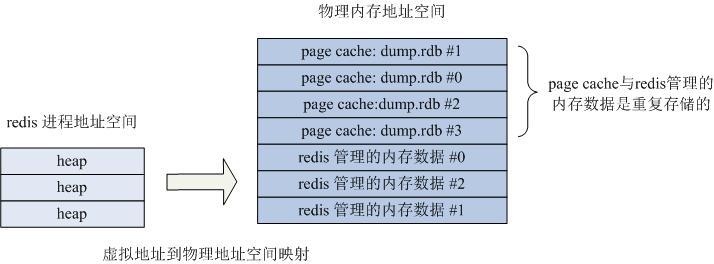
diskstore方式是作者放弃了虚拟内存方式后选择的一种新的实现方式，也就是传统的B-tree的方式，目前仍在实验阶段，后续是否可用我们可以拭目以待。

## Redis持久化磁盘IO方式及其带来的问题

有Redis线上运维经验的人会发现Redis在物理内存使用比较多，但还没有超过实际物理内存总容量时就会发生不稳定甚至崩溃的问题，有人认为是基于快照方式持久化的fork系统调用造成内存占用加倍而导致的，这种观点是不准确的，因为fork 调用的copy-on-write机制是基于操作系统页这个单位的，也就是只有有写入的脏页会被复制，但是一般你的系统不会在短时间内所有的页都发生了写入而导致复制，那么是什么原因导致Redis崩溃的呢？

答案是Redis的持久化使用了Buffer IO造成的，所谓Buffer IO是指Redis对持久化文件的写入和读取操作都会使用物理内存的Page Cache,而大多数数据库系统会使用Direct IO来绕过这层Page Cache并自行维护一个数据的Cache，而当Redis的持久化文件过大(尤其是快照文件)，并对其进行读写时，磁盘文件中的数据都会被加载到物理内存中作为操作系统对该文件的一层Cache,而这层Cache的数据与Redis内存中管理的数据实际是重复存储的，虽然内核在物理内存紧张时会做Page Cache的剔除工作，但内核很可能认为某块Page Cache更重要，而让你的进程开始Swap ,这时你的系统就会开始出现不稳定或者崩溃了。我们的经验是当你的Redis物理内存使用超过内存总容量的3/5时就会开始比较危险了。

下图是Redis在读取或者写入快照文件dump.rdb后的内存数据图：



## 总结：

1. 根据业务需要选择合适的数据类型，并为不同的应用场景设置相应的紧凑存储参数。
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以及最大的内存使用量。
3. 如果需要使用持久化，根据是否可以容忍重启丢失部分数据在快照方式与语句追加方式之间选择其一，不要使用虚拟内存以及diskstore方式。
4. 不要让你的Redis所在机器物理内存使用超过实际内存总量的3/5。