[**redis 的两种持久化方式及原理**](http://zhengdl126.iteye.com/blog/2191834)

**博客分类：**

* [memcache redis](http://zhengdl126.iteye.com/category/70624)

Redis是一种高级key-value数据库。它跟memcached类似，不过数据可以持久化，而且支持的数据类型很丰富。有字符串，链表，集 合和有序集合。支持在服务器端计算集合的并，交和补集(difference)等，还支持多种排序功能。所以Redis也可以被看成是一个数据结构服务 器。  
Redis的所有数据都是保存在内存中，然后不定期的通过异步方式保存到磁盘上(这称为“半持久化模式”)；也可以把每一次数据变化都写入到一个append only file(aof)里面(这称为“全持久化模式”)。

第一种方法**filesnapshotting**：默认redis是会以快照的形式将数据持久化到磁盘的（一个二进 制文件，dump.rdb，这个文件名字可以指定），在配置文件中的格式是：save N M表示在N秒之内，redis至少发生M次修改则redis抓快照到磁盘。当然我们也可以手动执行save或者bgsave（异步）做快照。

工作原理简单介绍一下：当redis需要做持久化时，redis会fork一个子进程；子进程将数据写到磁盘上一个临时RDB文件中；当子进程完成写临时文件后，将原来的RDB替换掉，这样的好处就是可以**copy-on-write**

还有一种持久化方法是**Append-only**：filesnapshotting方法在redis异常死掉时， 最近的数据会丢失（丢失数据的多少视你save策略的配置），所以这是它最大的缺点，当业务量很大时，丢失的数据是很多的。Append-only方法可 以做到全部数据不丢失，但redis的性能就要差些。AOF就可以做到全程持久化，只需要在配置文件中开启（默认是no），appendonly yes开启AOF之后，redis每执行一个修改数据的命令，都会把它添加到aof文件中，当redis重启时，将会读取AOF文件进行“重放”以恢复到 redis关闭前的最后时刻。

LOG Rewriting随着修改数据的执行AOF文件会越来越大，其中很多内容记录某一个key的变化情况。因此redis有了一种比较有意思的特性：在后台重建AOF文件，而不会影响client端操作。在任何时候执行**BGREWRITEAOF**命令，都会把当前内存中最短序列的命令写到磁盘，这些命令可以完全构建当前的数据情况，而不会存在多余的变化情况（比如状态变化，计数器变化等），缩小的AOF文件的大小。**所以当使用AOF时，redis推荐同时使用BGREWRITEAOF**。

AOF文件刷新的方式，有三种，参考配置参数**appendfsync** ：**appendfsync always**每提交一个修改命令都调用fsync刷新到AOF文件，非常非常慢，但也非常安全；**appendfsync everysec**每秒钟都调用fsync刷新到AOF文件，很快，但可能会丢失一秒以内的数据；**appendfsync no**依靠OS进行刷新，redis不主动刷新AOF，这样最快，但安全性就差。默认并推荐每秒刷新，这样在速度和安全上都做到了兼顾。

可能由于系统原因导致了AOF损坏，redis无法再加载这个AOF，可以按照下面步骤来修复：首先做一个AOF文件的备份，复制到其他地方；修复原始AOF文件，执行：**$ redis-check-aof –fix ;可以通过diff –u命令来查看修复前后文件不一致的地方；重启redis服务。**

LOG Rewrite的工作原理:同样用到了copy-on-write：首先redis会fork一个子进程；子进程将最新的AOF写入一个临时文件；父进程 增量的把内存中的最新执行的修改写入（这时仍写入旧的AOF，rewrite如果失败也是安全的）；当子进程完成rewrite临时文件后，父进程会收到 一个信号，并把之前内存中增量的修改写入临时文件末尾；这时redis将旧AOF文件重命名，临时文件重命名，开始向新的AOF中写入。

最后，为以防万一（机器坏掉或磁盘坏掉），记得定期把使用 filesnapshotting 或 Append-only 生成的\*rdb \*.aof文件备份到远程机器上。我是用crontab每半小时SCP一次。我没有使用redis的主从功能 ，因为半小时备份一次应该是可以了，而且我觉得有如果做主从有点浪费机器。这个最终还是看应用来定了。

========================

数据持久化通俗讲就是把数据保存到磁盘上，保证不会因为断电等因素丢失数据。

redis 需要经常将内存中的数据同步到磁盘来保证持久化。redis支持两种持久化方式，一种是 Snapshotting（快照）也是默认方式，另一种是Append-only file（缩写aof）的方式。先介绍下这两种dump方式再讲讲自己遇到的一些现象和想法，前面的内容是从网上整理出来的。  
  
**Snapshotting**  
快照是默认的持久化方式。这种方式是就是将内存中数据以快照的方式写入到二进制文件中,默认的文件名为dump.rdb。可以通过配置设置自动做快照持久 化的方式。我们可以配置redis在n秒内如果超过m个key被修改就自动做快照，下面是默认的快照保存配置

save 900 1  #900秒内如果超过1个key被修改，则发起快照保存  
save 300 10 #300秒内容如超过10个key被修改，则发起快照保存  
save 60 10000

下面介绍详细的快照保存过程

1.redis调用fork,现在有了子进程和父进程。

2. 父进程继续处理client请求，子进程负责将内存内容写入到临时文件。由于os的写时复制机制（copy on write)父子进程会共享相同的物理页面，当父进程处理写请求时os会为父进程要修改的页面创建副本，而不是写共享的页面。所以子进程的地址空间内的数 据是fork时刻整个数据库的一个快照。

3.当子进程将快照写入临时文件完毕后，用临时文件替换原来的快照文件，然后子进程退出。

client 也可以使用save或者bgsave命令通知redis做一次快照持久化。save操作是在主线程中保存快照的，由于redis是用一个主线程来处理所有 client的请求，这种方式会阻塞所有client请求。所以不推荐使用。另一点需要注意的是，每次快照持久化都是将内存数据完整写入到磁盘一次，并不 是增量的只同步脏数据。如果数据量大的话，而且写操作比较多，必然会引起大量的磁盘io操作，可能会严重影响性能。

另外由于快照方式是在一定间隔时间做一次的，所以如果redis意外down掉的话，就会丢失最后一次快照后的所有修改。如果应用要求不能丢失任何修改的话，可以采用aof持久化方式。下面介绍

**Append-only file**

aof 比快照方式有更好的持久化性，是由于在使用aof持久化方式时,redis会将每一个收到的写命令都通过write函数追加到文件中(默认是 appendonly.aof)。当redis重启时会通过重新执行文件中保存的写命令来在内存中重建整个数据库的内容。当然由于os会在内核中缓存 write做的修改，所以可能不是立即写到磁盘上。这样aof方式的持久化也还是有可能会丢失部分修改。不过我们可以通过配置文件告诉redis我们想要 通过fsync函数强制os写入到磁盘的时机。有三种方式如下（默认是：每秒fsync一次）

appendonly yes              //启用aof持久化方式  
# appendfsync always      //每次收到写命令就立即强制写入磁盘，最慢的，但是保证完全的持久化，不推荐使用  
appendfsync everysec     //每秒钟强制写入磁盘一次，在性能和持久化方面做了很好的折中，推荐  
# appendfsync no    //完全依赖os，性能最好,持久化没保证

aof 的方式也同时带来了另一个问题。持久化文件会变的越来越大。例如我们调用incr test命令100次，文件中必须保存全部的100条命令，其实有99条都是多余的。因为要恢复数据库的状态其实文件中保存一条set test 100就够了。为了压缩aof的持久化文件。redis提供了bgrewriteaof命令。收到此命令redis将使用与快照类似的方式将内存中的数据 以命令的方式保存到临时文件中，最后替换原来的文件。具体过程如下

1. redis调用fork ，现在有父子两个进程  
2. 子进程根据内存中的数据库快照，往临时文件中写入重建数据库状态的命令  
3.父进程继续处理client请求，除了把写命令写入到原来的aof文件中。同时把收到的写命令缓存起来。这样就能保证如果子进程重写失败的话并不会出问题。  
4.当子进程把快照内容写入已命令方式写到临时文件中后，子进程发信号通知父进程。然后父进程把缓存的写命令也写入到临时文件。  
5.现在父进程可以使用临时文件替换老的aof文件，并重命名，后面收到的写命令也开始往新的aof文件中追加。

需要注意到是重写aof文件的操作，并没有读取旧的aof文件，而是将整个内存中的数据库内容用命令的方式重写了一个新的aof文件,这点和快照有点类似。

**运维上的想法**

其实快照和aof一样，都使用了Copy-on-write技术。多次试验发现每次做数据dump的时候，内存都会扩大一倍(关于这个问题可以参考我去年写的[redis的内存陷阱](http://www.iteye.com/topic/808293)，很多人用redis做为缓存，数据量小，dump耗时非常短暂，所以不太容易发现)，这个时候会有三种情况：

一：物理内存足以满足，这个时候dump非常快，性能最好

二：物理内存+虚拟内存可以满足，这个时候dump速度会比较慢，磁盘swap繁忙，服务性能也会下降。所幸的是经过一段比较长的时候数据dump完成了，然后内存恢复正常。这个情况系统稳定性差。

三： 物理内存+虚拟内存不能满足，这个时候dump一直死着，时间久了机器挂掉。这个情况就是灾难！

如果数据要做持久化又想保证稳定性，建议留空一半的物理内存。如果觉得无法接受还是有办法，下面讲：

快照和aof虽然都使用Copy-on-write，但有个不同点，快照你无法预测redis什么时候做dump，aof可以通过bgrewriteaof命令控制dump的时机。

根据这点我可以在一个服务器上开启多个redis节点(利用多CPU)，使用aof的持久化方式。

**例 如在24G内存的服务器上开启3个节点，每天用bgrewriteaof定期重新整理数据，每个节点dump的时间都不一样，这 样理论上每个节点可以消耗6G内存，一共使用18G内存，另外6G内存在单个节点dump时用到，内存一下多利用了6G！ 当然节点开的越多内存的利用率也越高。如果带宽不是问题，节点数建议 = CPU数。**  
我的应用里为了保证高性能，数据没有做dump，也没有用aof。因为不做dump发生的故障远远低于做dump的时候，即使数据丢失了，自动修复脚本可以马上数据恢复。毕竟对海量数据redis只能做数据分片，那么落到每个节点上的数据量也不会很多。

redis的虚拟内存建议也不要用，用redis本来就是为了达到变态的性能，虚拟内存、aof看起来都有些鸡肋。

现在还离不开redis，因为它的mget是现在所有db里性能最好的，以前也考虑过用tokyocabinet hash方式做mget，性能不给力。直接用redis，基本上单个redis节点mget可以达到10W/s

纠错

之前说过redis做数据dump的时候内容会扩大一倍，后来我又做了些测试，发现有些地方说的不对。

top 命令并不是反映真实的内存占用情况，在top里尽管fork出来的子进程占了和父进程一样的内存，但是当做dump的时候没有写操作，实际使 用的是同一份内存的数据。当有写操作的时候内存才会真实的扩大（具体是不是真实的扩大一倍不确定，可能数据是按照页分片的），这才是真正的Copy- on-write。

基于这点在做数据持久化会更加灵活。