**Redis中数据存储模式有2种：cache-only,persistence;**

cache-only即只做为“缓存”服务，不持久数据，数据在服务终止后将消失，此模式下也将不存在“数据恢复”的手段，是一种安全性低/效率高/容易扩展的方式；

persistence即为内存中的数据持久备份到磁盘文件，在服务重启后可以恢复，此模式下数据相对安全。

**对于persistence持久化存储，Redis提供了两种持久化方法：**

Redis DataBase(简称RDB)

Append-only file (简称AOF)

除了这两种方法，Redis在早起的版本还存在虚拟内存的方法，现在已经被废弃。

**一、RDB概述**

RDB是在某个时间点将数据写入一个临时文件，持久化结束后，用这个临时文件替换上次持久化的文件，达到数据恢复。

优点：使用单独子进程来进行持久化，主进程不会进行任何IO操作，保证了redis的高性能

缺点：RDB是间隔一段时间进行持久化，如果持久化之间redis发生故障，会发生数据丢失。所以这种方式更适合数据要求不严谨的时候

这里说的这个执行数据写入到临时文件的时间点是可以通过配置来自己确定的，通过配置redis在n秒内如果超过m个key被修改这执行一次RDB操作。这个操作就类似于在这个时间点来保存一次Redis的所有数据，一次快照数据。所有这个持久化方法也通常叫做snapshots。

RDB默认开启，redis.conf中的具体配置参数如下；

#dbfilename：持久化数据存储在本地的文件

dbfilename dump.rdb

#dir：持久化数据存储在本地的路径，如果是在/redis/redis-3.0.6/src下启动的redis-cli，则数据会存储在当前src目录下

dir ./

##snapshot触发的时机，save <seconds> <changes>

##如下为900秒后，至少有一个变更操作，才会snapshot

##对于此值的设置，需要谨慎，评估系统的变更操作密集程度

##可以通过“save “””来关闭snapshot功能

#save时间，以下分别表示更改了1个key时间隔900s进行持久化存储；更改了10个key300s进行存储；更改10000个key60s进行存储。

save 900 1

save 300 10

save 60 10000

##当snapshot时出现错误无法继续时，是否阻塞客户端“变更操作”，“错误”可能因为磁盘已满/磁盘故障/OS级别异常等

stop-writes-on-bgsave-error yes

##是否启用rdb文件压缩，默认为“yes”，压缩往往意味着“额外的cpu消耗”，同时也意味这较小的文件尺寸以及较短的网络传输时间

rdbcompression yes

snapshot触发的时机，是有“间隔时间”和“变更次数”共同决定，同时符合2个条件才会触发snapshot,否则“变更次数”会被继续累加到下一个“间隔时间”上。snapshot过程中并不阻塞客户端请求。snapshot首先将数据写入临时文件，当成功结束后，将临时文件重名为dump.rdb。

使用RDB恢复数据：

RDB文件的载入工作是在服务器启动时自动执行的，并没有专门的命令。但是由于AOF的优先级更高，因此当AOF开启时，Redis会优先载入AOF文件来恢复数据；只有当AOF关闭时，才会在Redis服务器启动时检测RDB文件，并自动载入。服务器载入RDB文件期间处于阻塞状态，直到载入完成为止。

Redis载入RDB文件时，会对RDB文件进行校验，如果文件损坏，则日志中会打印错误，Redis启动失败。

客户端使用命令进行手动持久化save存储：

./redis-cli -h ip -p port save

./redis-cli -h ip -p port bgsave

一个是在前台进行存储，一个是在后台进行存储。我的client就在server这台服务器上，所以不需要连其他机器，直接./redis-cli bgsave。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。

其他自动触发机制:

除了save m n 以外和手动调用命令外，还有一些其他情况会触发bgsave：

在主从复制场景下，如果从节点执行全量复制操作，则主节点会执行bgsave命令，并将rdb文件发送给从节点执行shutdown命令时，自动执行rdb持久化

**save m n的实现原理**

Redis的save m n，是通过serverCron函数、dirty计数器、和lastsave时间戳来实现的。

serverCron是Redis服务器的周期性操作函数，默认每隔100ms执行一次；该函数对服务器的状态进行维护，其中一项工作就是检查 save m n 配置的条件是否满足，如果满足就执行bgsave。

dirty计数器是Redis服务器维持的一个状态，记录了上一次执行bgsave/save命令后，服务器状态进行了多少次修改(包括增删改)；而当save/bgsave执行完成后，会将dirty重新置为0。

例如，如果Redis执行了set mykey helloworld，则dirty值会+1；如果执行了sadd myset v1 v2 v3，则dirty值会+3；注意dirty记录的是服务器进行了多少次修改，而不是客户端执行了多少修改数据的命令。

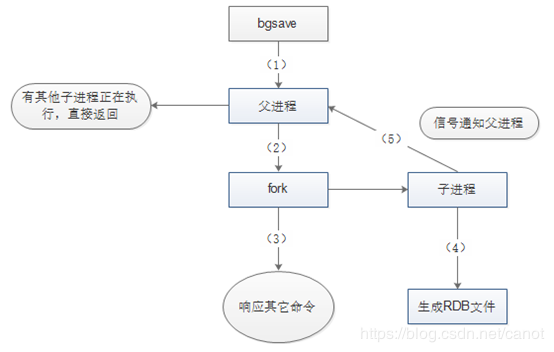
lastsave时间戳也是Redis服务器维持的一个状态，记录的是上一次成功执行save/bgsave的时间。

save m n的原理如下：每隔100ms，执行serverCron函数；在serverCron函数中，遍历save m n配置的保存条件，只要有一个条件满足，就进行bgsave。对于每一个save m n条件，只有下面两条同时满足时才算满足：

当前时间-lastsave > m

dirty >= n

**snapshot执行流程**



Redis父进程首先判断：当前是否在执行save，或bgsave/bgrewriteaof（后面会详细介绍该命令）的子进程，如果在执行则bgsave命令直接返回。bgsave/bgrewriteaof 的子进程不能同时执行，主要是基于性能方面的考虑：两个并发的子进程同时执行大量的磁盘写操作，可能引起严重的性能问题。

父进程执行fork操作创建子进程，这个过程中父进程是阻塞的，Redis不能执行来自客户端的任何命令

父进程fork后，bgsave命令返回”Background saving started”信息并不再阻塞父进程，并可以响应其他命令

子进程创建RDB文件，根据父进程内存快照生成临时快照文件，完成后对原有文件进行原子替换

子进程发送信号给父进程表示完成，父进程更新统计信息

**RDB常用配置总结**

下面是RDB常用的配置项，以及默认值:

save m n：bgsave自动触发的条件；如果没有save m n配置，相当于自动的RDB持久化关闭，不过此时仍可以通过其他方式触发

stop-writes-on-bgsave-error yes：当bgsave出现错误时，Redis是否停止执行写命令；设置为yes，则当硬盘出现问题时，可以及时发现，避免数据的大量丢失；设置为no，则Redis无视bgsave的错误继续执行写命令，当对Redis服务器的系统(尤其是硬盘)使用了监控时，该选项考虑设置为no

rdbcompression yes：是否开启RDB文件压缩。Redis默认采用LZF算法对RDB文件进行压缩。虽然压缩耗时，但是可以大大减小RDB文件的体积，因此压缩默认开启；可以通过命令关闭

rdbchecksum yes：是否开启RDB文件的校验，在写入文件和读取文件时都起作用；关闭checksum在写入文件和启动文件时大约能带来10%的性能提升，但是数据损坏时无法发现

dbfilename dump.rdb：RDB文件名

dir ./：RDB文件和AOF文件所在目录

**二、AOF概述**

Append-only file，将“操作 + 数据”以格式化指令的方式追加到操作日志文件的尾部，在append操作返回后(已经写入到文件或者即将写入)，才进行实际的数据变更，“日志文件”保存了历史所有的操作过程；当server需要数据恢复时，可以直接replay此日志文件，即可还原所有的操作过程。AOF相对可靠，它和mysql中bin.log、apache.log、zookeeper中txn-log简直异曲同工。AOF文件内容是字符串，非常容易阅读和解析。

优点：可以保持更高的数据完整性，如果设置追加file的时间是1s，如果redis发生故障，最多会丢失1s的数据；且如果日志写入不完整支持redis-check-aof来进行日志修复；AOF文件没被rewrite之前（文件过大时会对命令进行合并重写），可以删除其中的某些命令（比如误操作的flushall）。

缺点：AOF文件比RDB文件大，且恢复速度慢。

我们可以简单的认为AOF就是日志文件，此文件只会记录“变更操作”(例如：set/del等)，如果server中持续的大量变更操作，将会导致AOF文件非常的庞大，意味着server失效后，数据恢复的过程将会很长；事实上，一条数据经过多次变更，将会产生多条AOF记录，其实只要保存当前的状态，历史的操作记录是可以抛弃的；因为AOF持久化模式还伴生了“AOF rewrite”。

AOF的特性决定了它相对比较安全，如果你期望数据更少的丢失，那么可以采用AOF模式。如果AOF文件正在被写入时突然server失效，有可能导致文件的最后一次记录是不完整，你可以通过手工或者程序的方式去检测并修正不完整的记录，以便通过aof文件恢复能够正常；同时需要提醒，如果你的redis持久化手段中有aof，那么在server故障失效后再次启动前，需要检测aof文件的完整性。

AOF默认关闭，开启方法，修改配置文件reds.conf： **appendonly yes**

##此选项为aof功能的开关，默认为“no”，可以通过“yes”来开启aof功能

##只有在“yes”下，aof重写/文件同步等特性才会生效

appendonly yes

##指定aof文件名称

appendfilename appendonly.aof

##指定aof操作中文件同步策略，有三个合法值：always everysec no,默认为everysec

appendfsync everysec

##在aof-rewrite期间，appendfsync是否暂缓文件同步，"no"表示“不暂缓”，“yes”表示“暂缓”，默认为“no”

no-appendfsync-on-rewrite no

##aof文件rewrite触发的最小文件尺寸(mb,gb),只有大于此aof文件大于此尺寸是才会触发rewrite，默认“64mb”，建议“512mb”

auto-aof-rewrite-min-size 64mb

##相对于“上一次”rewrite，本次rewrite触发时aof文件应该增长的百分比。

##每一次rewrite之后，redis都会记录下此时“新aof”文件的大小(例如A)，那么当aof文件增长到A\*(1 + p)之后

##触发下一次rewrite，每一次aof记录的添加，都会检测当前aof文件的尺寸。

auto-aof-rewrite-percentage 100

AOF是文件操作，对于变更操作比较密集的server，那么必将造成磁盘IO的负荷加重；此外linux对文件操作采取了“延迟写入”手段，即并非每次write操作都会触发实际磁盘操作，而是进入了buffer中，当buffer数据达到阀值时触发实际写入(也有其他时机)，这是linux对文件系统的优化，但是这却有可能带来隐患，如果buffer没有刷新到磁盘，此时物理机器失效(比如断电)，那么有可能导致最后一条或者多条aof记录的丢失。通过上述配置文件，可以得知redis提供了3中aof记录同步选项：

always：每一条aof记录都立即同步到文件，这是最安全的方式，也以为更多的磁盘操作和阻塞延迟，是IO开支较大。

everysec：每秒同步一次，性能和安全都比较中庸的方式，也是redis推荐的方式。如果遇到物理服务器故障，有可能导致最近一秒内aof记录丢失(可能为部分丢失)。

no：redis并不直接调用文件同步，而是交给操作系统来处理，操作系统可以根据buffer填充情况/通道空闲时间等择机触发同步；这是一种普通的文件操作方式。性能较好，在物理服务器故障时，数据丢失量会因OS配置有关。

其实，我们可以选择的太少，everysec是最佳的选择。如果你非常在意每个数据都极其可靠，建议你选择一款“关系性数据库”吧。

AOF文件会不断增大，它的大小直接影响“故障恢复”的时间,而且AOF文件中历史操作是可以丢弃的。\*\*AOF rewrite操作就是“压缩”AOF文件的过程，当然redis并没有采用“基于原aof文件”来重写的方式，而是采取了类似snapshot的方式：基于copy-on-write，全量遍历内存中数据，然后逐个序列到aof文件中。因此AOF rewrite能够正确反应当前内存数据的状态，这正是我们所需要的；\*\*rewrite过程中，对于新的变更操作将仍然被写入到原AOF文件中，同时这些新的变更操作也会被redis收集起来(buffer，copy-on-write方式下，最极端的可能是所有的key都在此期间被修改，将会耗费2倍内存)，当内存数据被全部写入到新的aof文件之后，收集的新的变更操作也将会一并追加到新的aof文件中，此后将会重命名新的aof文件为appendonly.aof,此后所有的操作都将被写入新的aof文件。

**如果在rewrite过程中，出现故障，将不会影响原AOF文件的正常工作，只有当rewrite完成之后才会切换文件，因为rewrite过程是比较可靠的。**

触发rewrite的时机可以通过配置文件来声明，同时redis中可以通过bgrewriteaof指令人工干预。

redis-cli -h ip -p port bgrewriteaof

因为rewrite操作/aof记录同步/snapshot都消耗磁盘IO，redis采取了“schedule”策略：无论是“人工干预”还是系统触发，snapshot和rewrite需要逐个被执行。

AOF rewrite过程并不阻塞客户端请求。系统会开启一个子进程来完成。

**三.总结：**

AOF和RDB各有优缺点，这是有它们各自的特点所决定：

AOF更加安全，可以将数据更加及时的同步到文件中，但是AOF需要较多的磁盘IO开支，AOF文件尺寸较大，文件内容恢复数度相对较慢。

\*2) snapshot，安全性较差，它是“正常时期”数据备份以及master-slave数据同步的最佳手段，文件尺寸较小，恢复数度较快。

可以通过配置文件来指定它们中的一种，或者同时使用它们(不建议同时使用)，或者全部禁用，在架构良好的环境中，master通常使用AOF，slave使用snapshot，主要原因是master需要首先确保数据完整性，它作为数据备份的第一选择；slave提供只读服务(目前slave只能提供读取服务)，它的主要目的就是快速响应客户端read请求；但是如果你的redis运行在网络稳定性差/物理环境糟糕情况下，建议你master和slave均采取AOF，这个在master和slave角色切换时，可以减少“人工数据备份”/“人工引导数据恢复”的时间成本；如果你的环境一切非常良好，且服务需要接收密集性的write操作，那么建议master采取snapshot，而slave采用AOF。

链接地址：<https://blog.csdn.net/canot/article/details/52886923>