链滴

Redis-05- 持久化技术：RDB 和 AOF

作者：Anileh

原文链接：https://ld246.com/article/1652174927074

来源网站：链滴

Redis持久化技术

1. RDB

1.1 RDB基本介绍

RDB是什么：在指定的时间间隔内将内存中的数据集快照写入磁盘

（1） 备份是如何被执行的

Redis会单独创建（fork）一个子进程来进行持久化，会先将数据写入到 一个临时文件中，待持久化程都结束了，再用这个临时文件替换上次持久化好的文件。

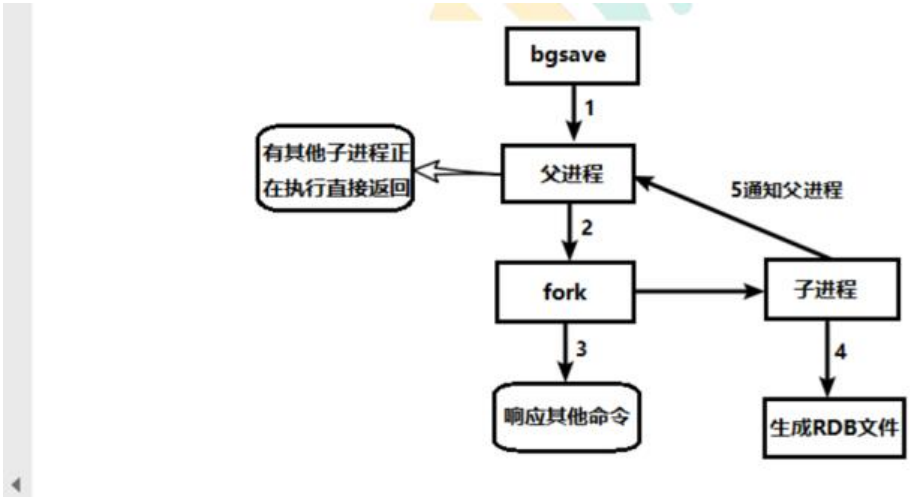
整个过程中，主进程是不进行任何IO操作的，这就确保了极高的性能 如果需要进行大规模数据的恢，且对于数据恢复的完整性不是非常敏感，那RDB方式要比AOF方式更加的高效。

RDB的缺点：最后一次持久化后的数据可能丢失。

（2）fork

- fork的作用是复制一个与当前进程一样的进程。新进程的所有数据（变量、环境变量、程序计数器） 数值都和原进程一致，但是是一个全新的进程，并作为原进程的子进程
- 在Linux程序中，fork()会产生一个和父进程完全相同的子进程，但子进程在此后多会exec系统调用出于效率考虑，Linux中引入了** “写时复制技术” **
- 一般情况父进程和子进程会共用同一段物理内存，只有进程空间的各段的内容要发生变化时，才会父进程的内容复制一份给子进程。

1.2 RDB持久化流程



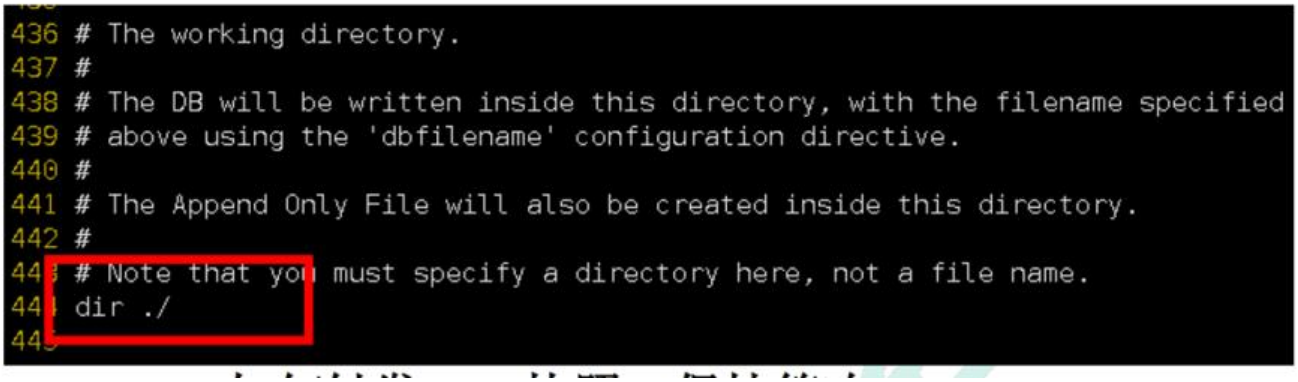
1.2.dump.rdb文件

在redis.conf中配置文件名称，默认为dump.rdb

1.2.1 配置位置

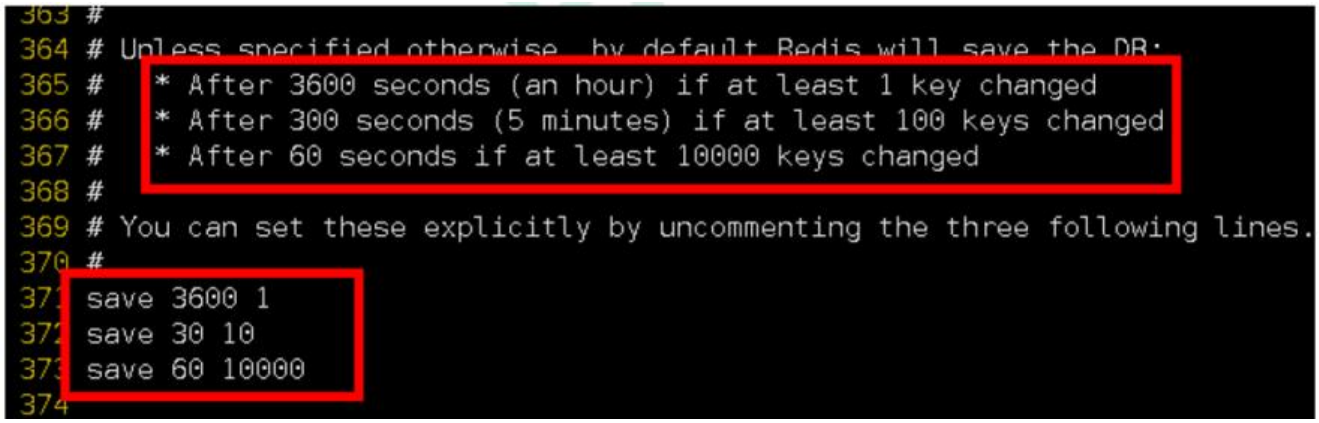
rdb文件的保存路径，默认在Redis启动命令行所在的目录下

dir "/myredis/"



1.2.2 如何触发RDB快照

配置文件中默认的快照配置

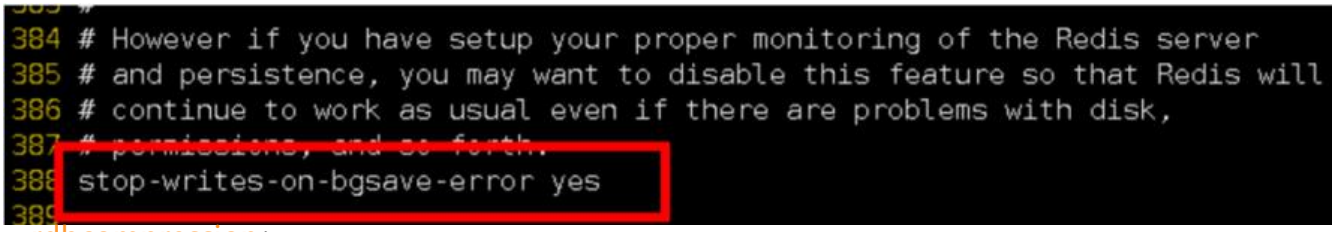


- save命令VSbgsave命令:

save：只管保存，全部阻塞，手动保存

bgsave：redis会在后台异步执行快照操作，快照同时可以响应客户端请求，
可以通过lastsave命令获取最后一次成功执行快照的时间

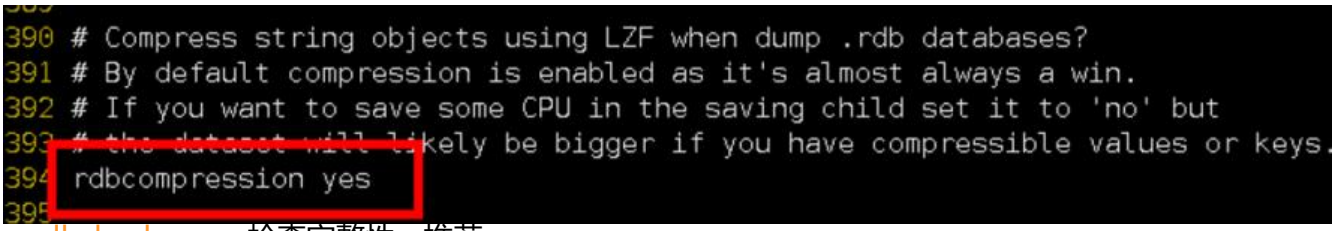
- flushall命令：产生dump.rdb空文件，无意义
- stop-writes-on-bgsave-error：当Redis无法写入磁盘的话，直接关掉Redis的写操作。推荐yes.



- rdbcompression：

对于存储到磁盘中的快照，可以设置是否进行压缩存储。如果是的话，redis会采用LZF算法进行压缩。

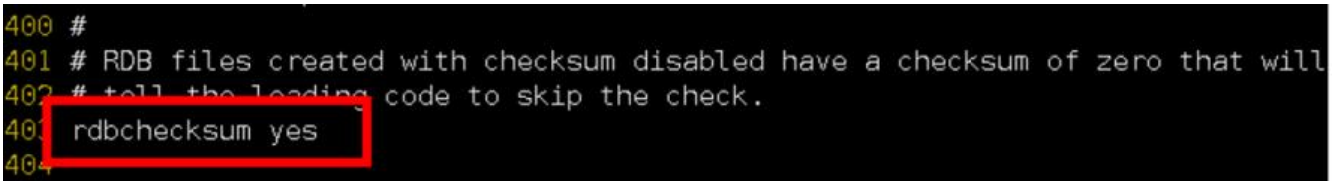
如果你不想消耗CPU来进行压缩的话，可以设置为关闭此功能。推荐yes



- rdbchecksum：检查完整性，推荐yes.

在存储快照后，还可以让redis使用CRC64算法来进行数据校验，

但是这样做会增加大约10%的性能消耗，如果希望获取到最大的性能提升，可以关闭此功能



1.2.3 RDB备份

先通过config get dir 查询rdb文件的目录

将*.rdb的文件拷贝到别的地方

rdb的恢复步骤：

- 关闭Redis
- 先把备份的文件拷贝到工作目录下 cp dump2.rdb dump.rdb
- 启动Redis, 备份数据会直接加载

1.3 RDB的优点

- 适合大规模的数据恢复
- 对数据完整性和一致性要求不高更适合使用
- 节省磁盘空间
- 恢复速度快

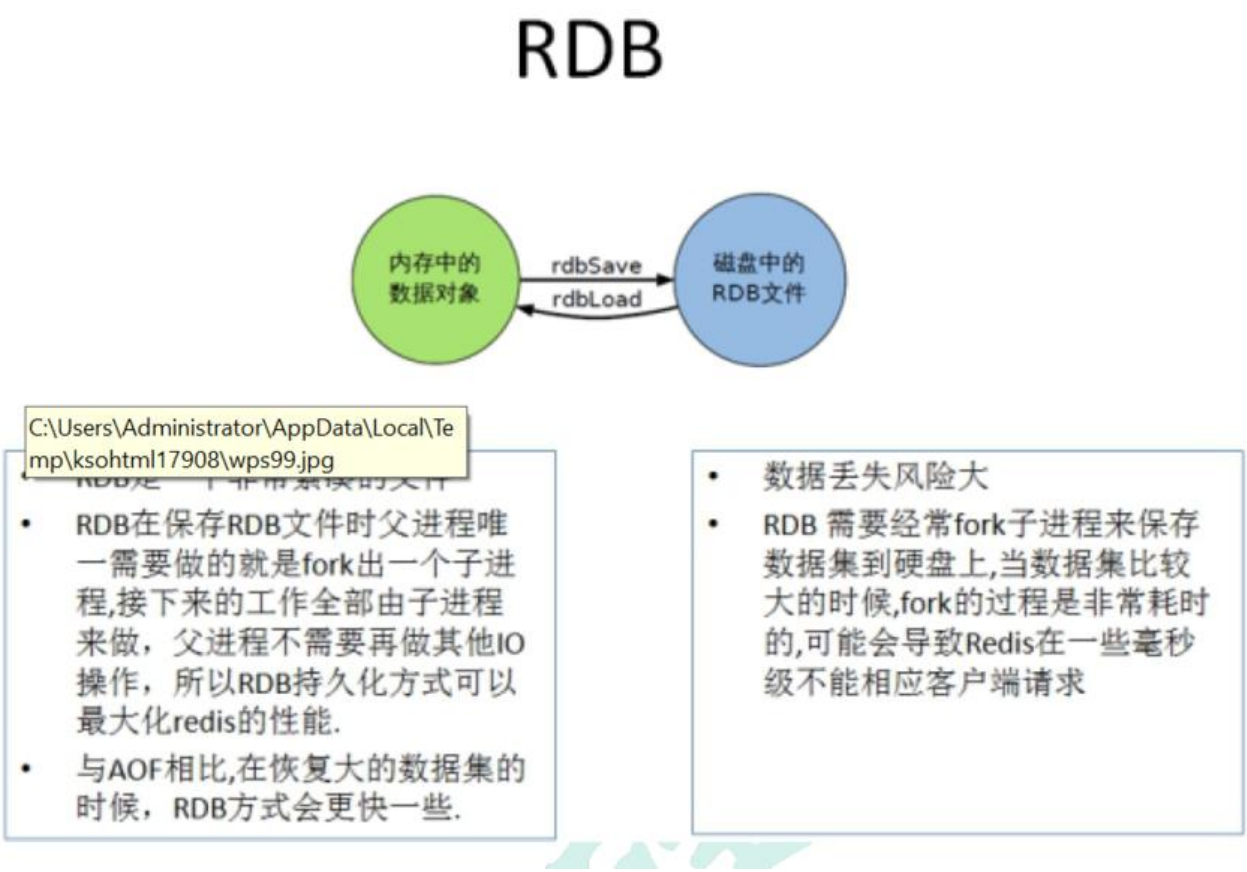
1.4 RDB的劣势

- Fork的时候，内存中的数据被克隆了一份，大致2倍的膨胀性需要考虑

● 虽然Redis在fork时使用了写时拷贝技术,但是如果数据庞大时还是比较消耗性能。

● 在备份周期在一定间隔时间做一次备份，所以如果Redis意外down掉的话，就会丢失最后一次快照的所有修改。

1.5 总结



2. AOF（Append Only File）

2.1 基本介绍

以日志的形式来记录每个写操作（增量操作），将Redis执行过得所有写指令记录下（不记录读操作，

只许追加文件但不可以改写文件。redis启动时会读取该文件来重新构建数据，换言之，redis重启根日志文件

的内容将执行写指令来完成数据的恢复工作。

AOF默认不开启：在redis.conf中配置文件名称，默认为append.aof，AOF 文件的保存路径与RDB路径一致

AOF和RDB同时开启：系统默认取AOF的数据（数据不会存在丢失）

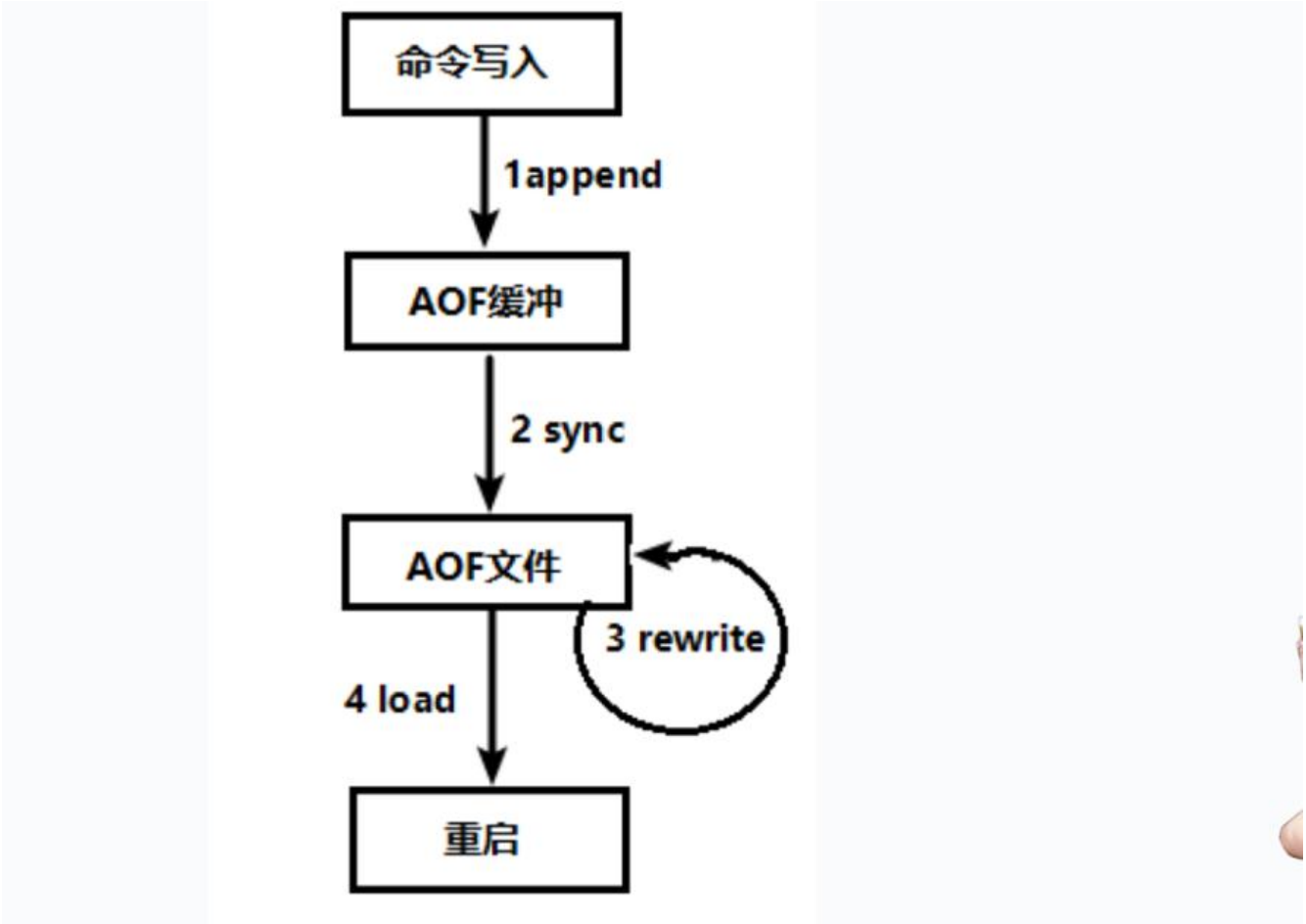
2.2 AOF持久化流程

（1）客户端的请求写命令会被append到AOF缓存区内

（2） AOF缓存区根据AOF持久化策略[always, everysec, no]将操作sync到磁盘的AOF文件中

（3）AOF文件大小超过重写策略或手动重写时，会对AOF文件rewrite，压缩AOF文件容量

（4）Redis服务重启时，会重新load AOF文件中的写操作来恢复数据



2.3 AOF基本操作

AOF 的备份机制和性能虽然和RDB不同，但是备份和恢复的操作与RDB一致：

拷贝备份文件，需要恢复时在拷贝到Redis的工作目录下，启动系统即加载

正常恢复：

- 修改默认的appendonly将no改为yes
- 将有数据的aof文件复制保存到对应目录（查看目录： config get dir）
- 恢复： 重启Redis然后重新加载

异常恢复：

- 修改默认的appendonly将no改为yes
- 若AOF文件损坏，通过/usr/local/bin/redis-check-aof--fix appendonly.aof 进行恢复
- 备份AOF 文件

- 恢复：重启redis，然后重新加载

2.4 AOF同步评率

appendfsync always：始终同步，每次Redis的写入都会立刻计入日志，性能较差但数据完整性较好

appendfsync everysec：每秒同步，每秒记入日志一次，如果宕机，本秒的数据可能丢失

appendfsync no：redis不主动进行同步，把同步时机交给操作系统

2.5 Rewrite压缩

2.5.1 什么是Rewrite：

AOF采用文件追加方式，文件会越来越大为避免出现此种情况，新增了重写机制，

当AOF文件的大小超过所设定的阈值时，Redis就会启动AOF文件的内容压缩，

只保留可以恢复数据的最小指令集，可以使用命令bgrewriteaof

2.5.2 重写原理，如何实现重写

AOF文件持续增长而过大时，会fork出一条新进程来将文件重写(也是先写临时文件最后再rename)，

redis4.0版本后的重写，是指上就是把rdb 的快照，以二级制的形式附在新的aof头部，作为已有的史数据，

替换掉原来的流水账操作。

no-appendfsync-on-rewrite：

- no-appendfsync-on-rewrite=yes ：不写入aof文件只写入缓存，用户请求不会阻塞，但是在这段间如果宕机会丢失这段时间的缓存数据。降低数据安全性，提高性能
- no-appendfsync-on-rewrite=no： 还是会把数据往磁盘里刷，但是遇到重写操作，可能会发生塞。数据安全，但是性能降低

触发机制，何时重写：Redis会记录上次重写时的AOF大小，默认配置是当AOF文件大小是上次rewrite后大小的一倍且文件大于64M时触发

重写虽然可以节约大量磁盘空间，减少恢复时间。但是每次重写还是有一定的负担的，因此设定Redi要满足一

定条件才会进行重写。

auto-aof-rewrite-percentage：设置重写的基准值，文件达到100%时开始重写

（文件是原来重写后文件的2倍时触发）

auto-aof-rewrite-min-size：设置重写的基准值，最小文件64MB。达到这个值开始重写。

例如：文件达到70MB开始重写，降到50MB，下次什么时候开始重写？100MB

系统载入时或者上次重写完毕时，Redis会记录此时AOF大小，设为base_size,

如果Redis的AOF当前大小>= base_size +base_size*100% (默认)且当前大小>=64mb(默认)的情况，Redis

会对AOF进行重写。

2.5.3 重写流程

（1）bgrewriteaof触发重写，判断是否当前有bgsave或bgrewriteaof在运行，如果有，则等待该命结束后

再继续执行。

（2）主进程fork出子进程执行重写操作，保证主进程不会阻塞。

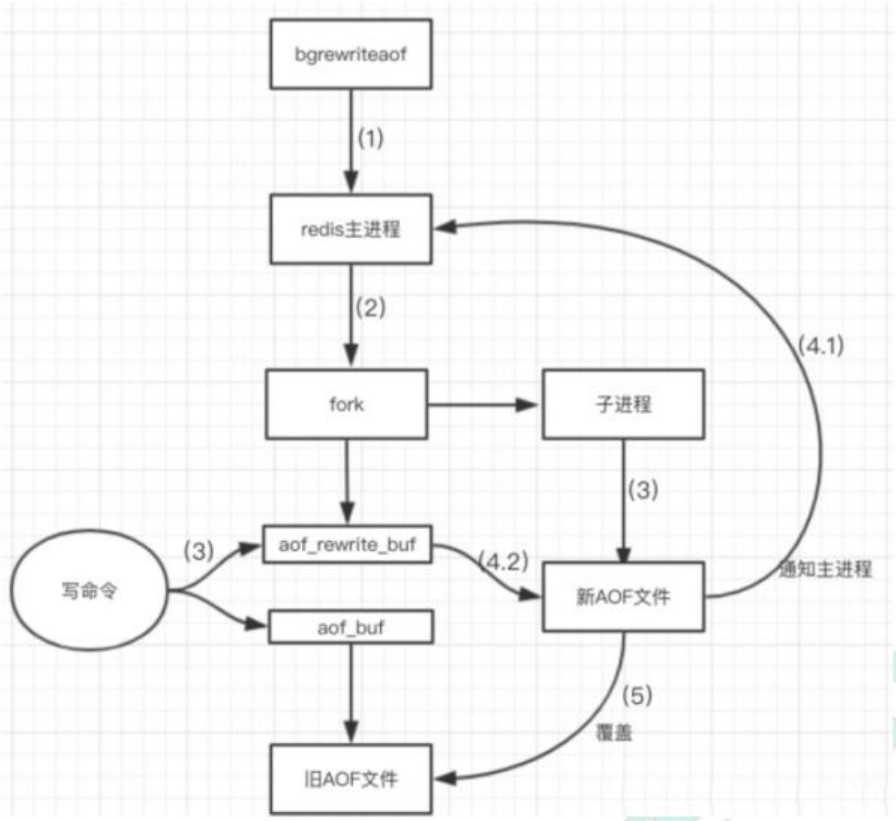
（3）子进程遍历redis内存中数据到临时文件，客户端的写请求同时写入aof_buf缓冲区和aof_rewrite buf,

重写缓冲区保证原AOF文件完整以及新AOF文件生成期间的新的数据修改动作不会丢失。

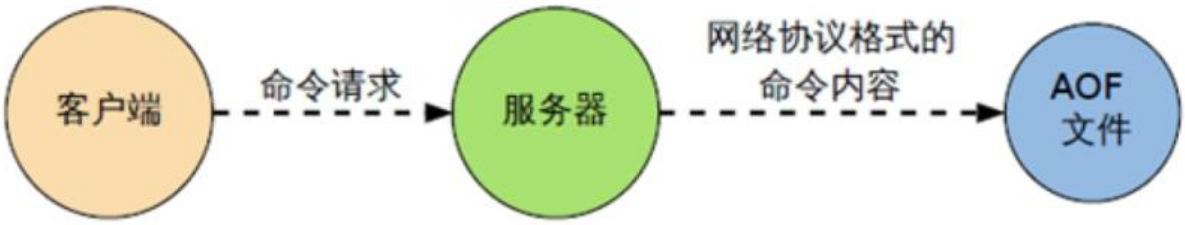
（4）子进程写完新的AOF文件后，向主进程发信号，父进程更新统计信息。

主进程把aof_rewrite_buf中的数据写入到新的AOF文件。

（5）使用新的AOF文件覆盖旧的AOF文件，完成AOF重写。



2.6 AOF的优势



- 备份机制更稳健，丢失数据概率更低。
- 可读的日志文本，通过操作 AOF 稳健，可以处理误操作。

- 备份机制稳健，丢失数据概率更低
- 可读的日志文件，通过操作AOF稳健，可以处理误操作

2.7AOF的劣势

- 比起RDB占用更多的磁盘空间。
- 恢复备份速度要慢。
- 每次读写都同步的话，有一定的性能压力。
- 存在个别Bug，造成恢复异常

2.8 总结

3. RDB与AOF的对比

3.1 选择建议

官方推荐两个都启用。

如果对数据不敏感，可以选单独用RDB。

不建议单独用 AOF，因为可能会出现Bug。

如果只是做纯内存缓存，可以都不用。

3.2 两者对比

（1）基本信息

- RDB持久化方式能够在指定的时间间隔能对你的数据进行快照存储
- AOF持久化方式记录每次对服务器写的操作，当服务器重启的时候会重新执行这些命令来恢复原始数

据，AOF命令以redis协议追加保存每次写的操作到文件末尾.

- Redis还能对AOF文件进行后台重写，以缩小使得AOF文件的体积

（2）只做缓存

如果只希望数据在服务器运行的时候存在，也可以不使用任何持久化方式.

（3）同时开启RDB和AOF

- 在这种情况下,当redis重启的时候会优先载入AOF文件来恢复原始的数据, 因为在通常情况下AOF文保存的数据集要比RDB文件保存的数据集要完整.
- RDB的数据不实时，同时使用两者时服务器重启也只会找AOF文件。

（4）只使用AOF?

- 不建议，因为RDB更适合用于备份数据库(AOF在不断变化不好备份)， RDB可以快速重启，而且不有AOF可能潜在的bug。

3.3 性能建议

因为RDB文件只用作后备用途，建议只在**Slave上持久化RDB文件**，而且只要15分钟备份一次就够了只保留**save 9001**这条规则。

如果使用AOF，

- 好处是在最恶劣情况下也只会丢失不超过两秒数据，启动脚本较简单只load自己的AOF文件就可以。
- 代价,一是带来了持续的IO，二是AOF 将rewrite过程中产生的新数据写到新文件会造成阻塞

只要硬盘许可，尽量减少AOF rewrite的频率，AOF重写的基础大小默认值64M太小了，可以设到5G上。

默认超过原大小100%大小时重写可以改到适当的数值。