

## 04-AOF日志：宕机了，Redis如何避免数据丢失？

你好，我是蒋德钧。

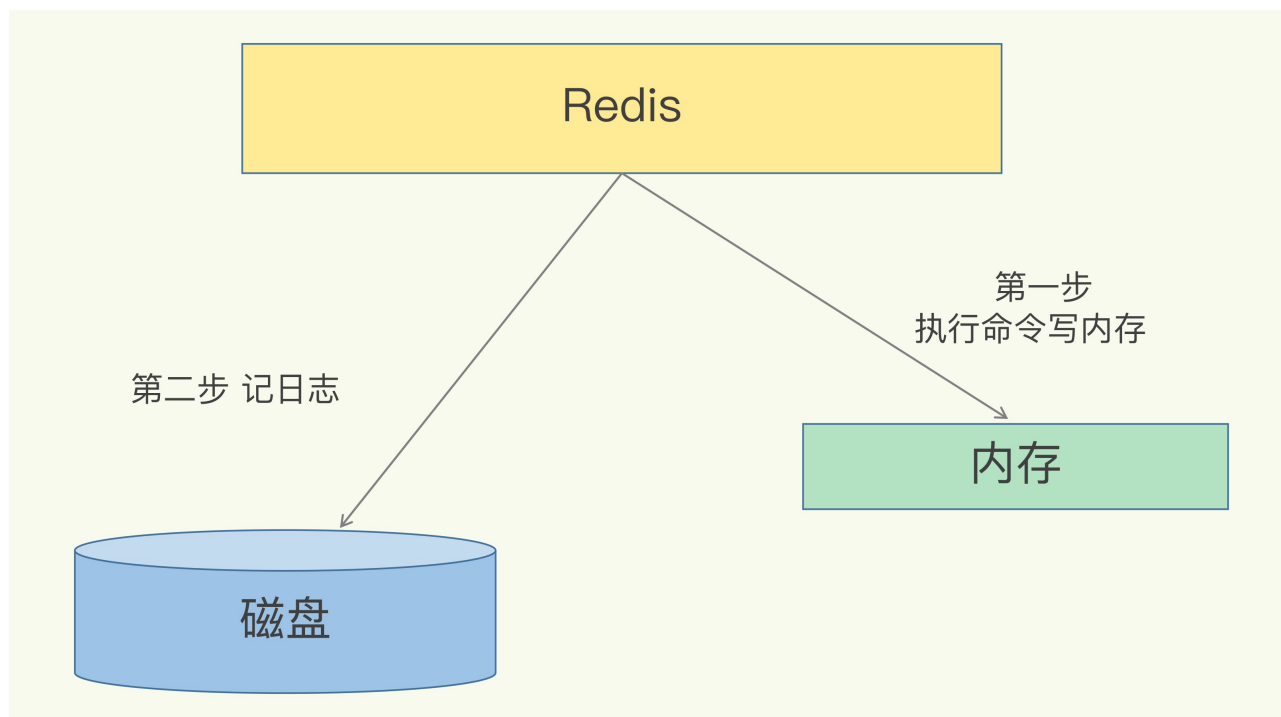
如果有人问你：“你会把Redis用在什么业务场景下？”我想你大概率会说：“我会把它当作缓存使用，因为它把后端数据库中的数据存储在内存在，然后直接从内存中读取数据，响应速度会非常快。”没错，这确实是Redis的一个普遍使用场景，但是，这里也有一个绝对不能忽略的问题：**一旦服务器宕机，内存中的数据将全部丢失。**

我们很容易想到的一个解决方案是，从后端数据库恢复这些数据，但这种方式存在两个问题：一是，需要频繁访问数据库，会给数据库带来巨大的压力；二是，这些数据是从慢速数据库中读取出来的，性能肯定比不上从Redis中读取，导致使用这些数据的应用程序响应变慢。所以，对Redis来说，实现数据的持久化，避免从后端数据库中进行恢复，是至关重要的。

目前，Redis的持久化主要有两大机制，即AOF日志和RDB快照。在接下来的两节课里，我们就分别学习一下吧。这节课，我们先重点学习下AOF日志。

### AOF日志是如何实现的？

说到日志，我们比较熟悉的是数据库的写前日志（Write Ahead Log, WAL），也就是说，在实际写数据前，先把修改的数据记到日志文件中，以便故障时进行恢复。不过，AOF日志正好相反，它是写后日志，“写后”的意思是Redis是先执行命令，把数据写入内存，然后才记录日志，如下图所示：

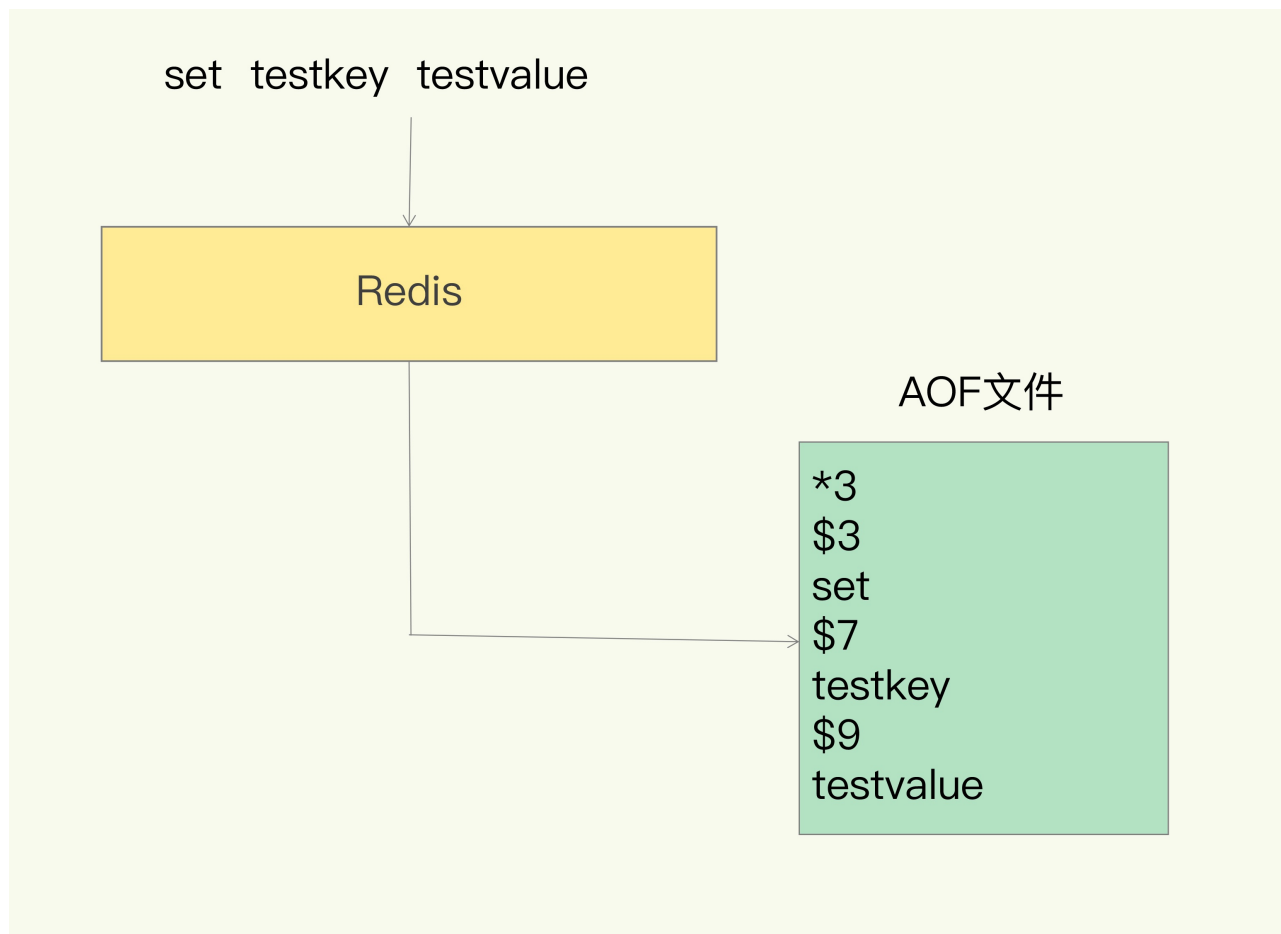


那AOF为什么要先执行命令再记日志呢？要回答这个问题，我们要先知道AOF里记录了什么内容。

传统数据库的日志，例如redo log（重做日志），记录的是修改后的数据，而AOF里记录的是Redis收到的每一条命令，这些命令是以文本形式保存的。

我们以Redis收到“set testkey testvalue”命令后记录的日志为例，看看AOF日志的内容。其中，“\*3”表示当前命令有三个部分，每部分都是由“\$+数字”开头，后面紧跟着具体的命令、键或值。这里，“数字”表示这部分中的命令、键或值一共有多少字节。例如，“\$3 set”表示这部分有3个字节，也就

是“set”命令。



但是，为了避免额外的检查开销，Redis在向AOF里面记录日志的时候，并不会先去对这些命令进行语法检查。所以，如果先记日志再执行命令的话，日志中就有可能记录了错误的命令，Redis在使用日志恢复数据时，就可能会出错。

而写后日志这种方式，就是先让系统执行命令，只有命令能执行成功，才会被记录到日志中，否则，系统就会直接向客户端报错。所以，Redis使用写后日志这一方式的一大好处是，可以避免出现记录错误命令的情况。

除此之外，AOF还有一个好处：它是在命令执行后才记录日志，所以**不会阻塞当前的写操作**。

不过，AOF也有两个潜在的风险。

首先，如果刚执行完一个命令，还没有来得及记日志就宕机了，那么这个命令和相应的数据就有丢失的风险。如果此时Redis是用作缓存，还可以从后端数据库重新读入数据进行恢复，但是，如果Redis是直接用作数据库的话，此时，因为命令没有记入日志，所以就无法用日志进行恢复了。

其次，AOF虽然避免了对当前命令的阻塞，但可能会给下一个操作带来阻塞风险。这是因为，AOF日志也是在主线程中执行的，如果在把日志文件写入磁盘时，磁盘写压力大，就会导致写盘很慢，进而导致后续的操作也无法执行了。

仔细分析的话，你就会发现，这两个风险都是和AOF写回磁盘的时机相关的。这也就意味着，如果我们能够控制一个写命令执行完后AOF日志写回磁盘的时机，这两个风险就解除了。

### 三种写回策略

其实，对于这个问题，AOF机制给我们提供了三个选择，也就是AOF配置项appendfsync的三个可选值。

- **Always**，同步写回：每个写命令执行完，立马同步地将日志写回磁盘；
- **Everysec**，每秒写回：每个写命令执行完，只是先把日志写到AOF文件的内存缓冲区，每隔一秒把缓冲区中的内容写入磁盘；
- **No**，操作系统控制的写回：每个写命令执行完，只是先把日志写到AOF文件的内存缓冲区，由操作系统决定何时将缓冲区内容写回磁盘。

针对避免主线程阻塞和减少数据丢失问题，这三种写回策略都无法做到两全其美。我们来分析下其中的原因。

- “同步写回”可以做到基本不丢数据，但是它在每一个写命令后都有一个慢速的落盘操作，不可避免地会影响主线程性能；
- 虽然“操作系统控制的写回”在写完缓冲区后，就可以继续执行后续的命令，但是落盘的时机已经不在Redis手中了，只要AOF记录没有写回磁盘，一旦宕机对应的数据就丢失了；
- “每秒写回”采用一秒写回一次的频率，避免了“同步写回”的性能开销，虽然减少了对系统性能的影响，但是如果发生宕机，上一秒内未落盘的命令操作仍然会丢失。所以，这只能算是，在避免影响主线程性能和避免数据丢失两者间取了个折中。

我把这三种策略的写回时机，以及优缺点汇总在了一张表格里，以方便你随时查看。

配置项	写回时机	优点	缺点
Always	同步写回	可靠性高，数据基本不丢失	每个写命令都要落盘，性能影响较大
Everysec	每秒写回	性能适中	宕机时丢失1秒内的数据
No	操作系统控制的写回	性能好	宕机时丢失数据较多

到这里，我们就可以根据系统对高性能和高可靠性的要求，来选择使用哪种写回策略了。总结一下就是：想要获得高性能，就选择No策略；如果想要得到高可靠性保证，就选择Always策略；如果允许数据有一点丢失，又希望性能别受太大影响的话，那么就选择Everysec策略。

但是，按照系统的性能需求选定了写回策略，并不是“高枕无忧”了。毕竟，AOF是以文件的形式在记录接收到的所有写命令。随着接收的写命令越来越多，AOF文件会越来越大。这也就意味着，我们一定要小心AOF文件过大带来的性能问题。

这里的“性能问题”，主要在于以下三个方面：一是，文件系统本身对文件大小有限制，无法保存过大的文件；二是，如果文件太大，之后再往里面追加命令记录的话，效率也会变低；三是，如果发生宕机，AOF中记录的命令要一个个被重新执行，用于故障恢复，如果日志文件太大，整个恢复过程就会非常缓慢，这就会影响到Redis的正常使用。

所以，我们就要采取一定的控制手段，这个时候，**AOF重写机制**就登场了。

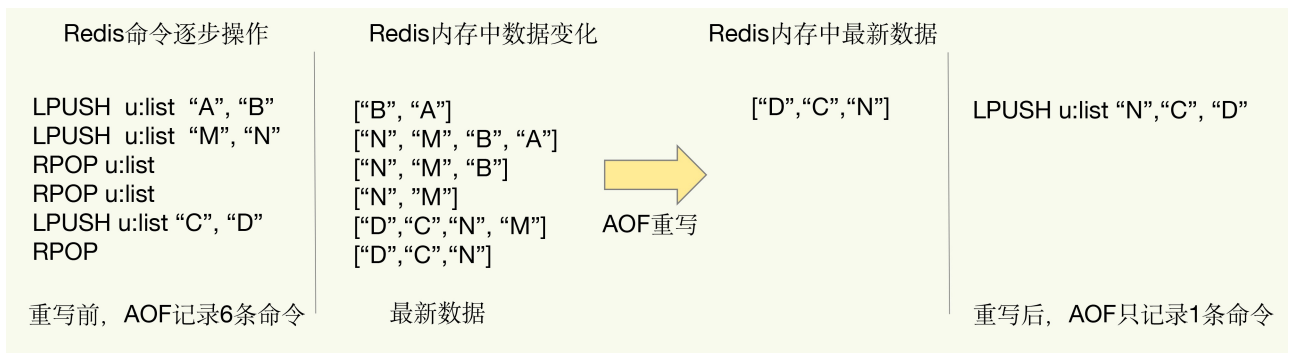
## 日志文件太大了怎么办？

简单来说，AOF重写机制就是在重写时，Redis根据数据库的现状创建一个新的AOF文件，也就是说，读取数据库中的所有键值对，然后对每一个键值对用一条命令记录它的写入。比如说，当读取了键值对“testkey”：“testvalue”之后，重写机制会记录set testkey testvalue这条命令。这样，当需要恢复时，可以重新执行该命令，实现“testkey”：“testvalue”的写入。

为什么重写机制可以把日志文件变小呢？实际上，重写机制具有“多变一”功能。所谓的“多变一”，也就是说，旧日志文件中的多条命令，在重写后的新日志中变成了一条命令。

我们知道，AOF文件是以追加的方式，逐一记录接收到的写命令的。当一个键值对被多条写命令反复修改时，AOF文件会记录相应的多条命令。但是，在重写的时候，是根据这个键值对当前的最新状态，为它生成对应的写入命令。这样一来，一个键值对在重写日志中只用一条命令就行了，而且，在日志恢复时，只用执行这条命令，就可以直接完成这个键值对的写入了。

下面这张图就是一个例子：



当我们对一个列表先后做了6次修改操作后，列表的最后状态是[“D”，“C”，“N”]，此时，只用LPUSH u:list “N”，“C”，“D”这一条命令就能实现该数据的恢复，这就节省了五条命令的空间。对于被修改过成百上千次的键值对来说，重写能节省的空间当然就更大了。

不过，虽然AOF重写后，日志文件会缩小，但是，要把整个数据库的最新数据的操作日志都写回磁盘，仍然是一个非常耗时的过程。这时，我们就要继续关注另一个问题了：重写会不会阻塞主线程？

## AOF重写会阻塞吗？

和AOF日志由主线程写回不同，重写过程是由后台线程bgrewriteaof来完成的，这也是为了避免阻塞主线程，导致数据库性能下降。

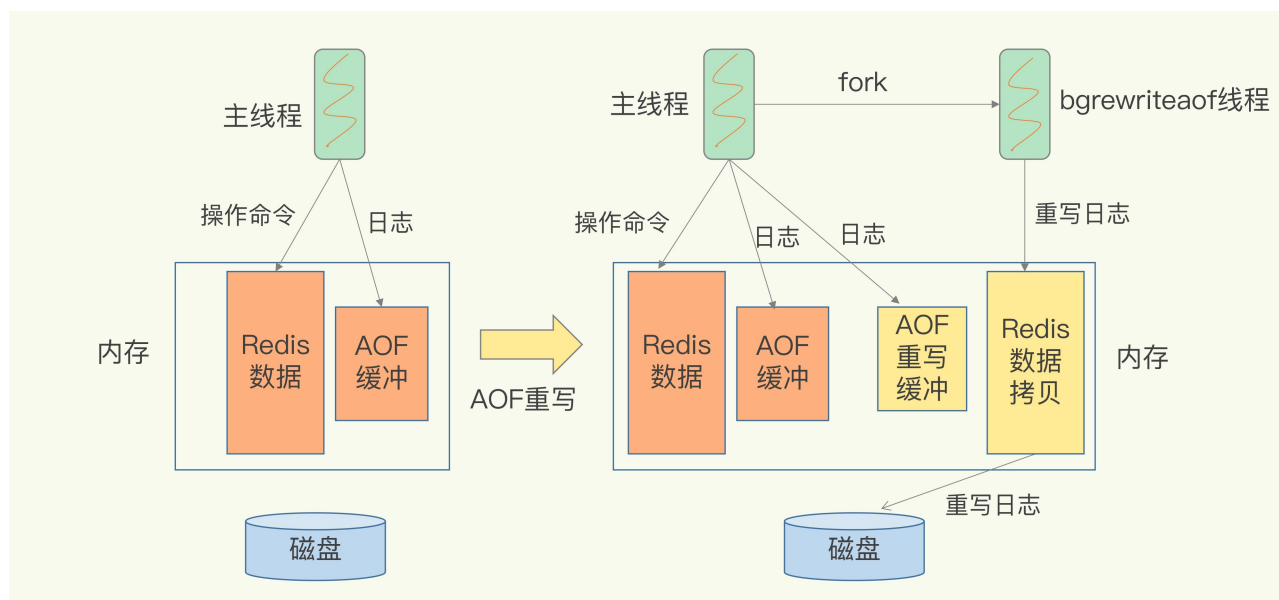
我把重写的过程总结为“**一个拷贝，两处日志**”。

“一个拷贝”就是指，每次执行重写时，主线程fork出后台的bgrewriteaof子进程。此时，fork会把主线程的内存拷贝一份给bgrewriteaof子进程，这里面就包含了数据库的最新数据。然后，bgrewriteaof子进程就可以在不影响主线程的情况下，逐一把拷贝的数据写成操作，记入重写日志。

“两处日志”又是什么呢？

因为主线程未阻塞，仍然可以处理新来的操作。此时，如果有写操作，第一处日志就是指正在使用的AOF日志，Redis会把这个操作写到它的缓冲区。这样一来，即使宕机了，这个AOF日志的操作仍然是齐全的，可以用于恢复。

而第二处日志，就是指新的AOF重写日志。这个操作也会被写到重写日志的缓冲区。这样，重写日志也不会丢失最新的操作。等到拷贝数据的所有操作记录重写完成后，重写日志记录的这些最新操作也会写入新的AOF文件，以保证数据库最新状态的记录。此时，我们就可以用新的AOF文件替代旧文件了。



总结来说，每次AOF重写时，Redis会先执行一个内存拷贝，用于重写；然后，使用两个日志保证在重写过程中，新写入的数据不会丢失。而且，因为Redis采用额外的线程进行数据重写，所以，这个过程并不会阻塞主线程。

## 小结

这节课，我向你介绍了Redis用于避免数据丢失的AOF方法。这个方法通过逐一记录操作命令，在恢复时再逐一执行命令的方式，保证了数据的可靠性。

这个方法看似“简单”，但也是充分考虑了对Redis性能的影响。总结来说，它提供了AOF日志的三种写回策略，分别是Always、Everysec和No，这三种策略在可靠性上是从高到低，而在性能上则是从低到高。

此外，为了避免日志文件过大，Redis还提供了AOF重写机制，直接根据数据库里数据的最新状态，生成这些数据的插入命令，作为新日志。这个过程通过后台线程完成，避免了对主线程的阻塞。

其中，三种写回策略体现了系统设计中的一个重要原则，即trade-off，或者称为“取舍”，指的就是在性能和可靠性保证之间做取舍。我认为，这是做系统设计和开发的一个关键哲学，我也非常希望，你能充分地理解这个原则，并在日常开发中加以应用。

不过，你可能也注意到了，落盘时机和重写机制都是在“记日志”这一过程中发挥作用的。例如，落盘时机的选择可以避免记日志时阻塞主线程，重写可以避免日志文件过大。但是，在“用日志”的过程中，也就是使用AOF进行故障恢复时，我们仍然需要把所有的操作记录都运行一遍。再加上Redis的单线程设计，这些命令操作只能一条一条按顺序执行，这个“重放”的过程就会很慢。

那么，有没有既能避免数据丢失，又能更快地恢复的方法呢？当然有，那就是RDB快照了。下节课，我们就

一起学习一下，敬请期待。

## 每课一问

这节课，我给你提两个小问题：

1. AOF日志重写的时候，是由bgrewriteaof子进程来完成的，不用主线程参与，我们今天说的非阻塞也是指子进程的执行不阻塞主线程。但是，你觉得，这个重写过程有没有其他潜在的阻塞风险呢？如果有的话，会在哪里阻塞？
2. AOF重写也有一个重写日志，为什么它不共享使用AOF本身的日志呢？

希望你能好好思考一下这两个问题，欢迎在留言区分享你的答案。另外，也欢迎你把这节课的内容转发出去，和更多的人一起交流讨论。

## 精选留言：

● Kaito 2020-08-12 11:38:16

问题1，Redis采用fork子进程重写AOF文件时，潜在的阻塞风险包括：fork子进程 和 AOF重写过程中父进程产生写入的场景，下面依次介绍。

a、fork子进程，fork这个瞬间一定是会阻塞主线程的（注意，fork时并不会一次性拷贝所有内存数据给子进程，老师文章写的是拷贝所有内存数据给子进程，我个人认为是有歧义的），fork采用操作系统提供的写实复制(Copy On Write)机制，就是为了避免一次性拷贝大量内存数据给子进程造成的长时间阻塞问题，但fork子进程需要拷贝进程必要的数据结构，其中有一项就是拷贝内存页表（虚拟内存和物理内存的映射索引表），这个拷贝过程会消耗大量CPU资源，拷贝完成之前整个进程是会阻塞的，阻塞时间取决于整个实例的内存大小，实例越大，内存页表越大，fork阻塞时间越久。拷贝内存页表完成后，子进程与父进程指向相同的内存地址空间，也就是说此时虽然产生了子进程，但是并没有申请与父进程相同的内存大小。那什么时候父子进程才会真正内存分离呢？“写实复制”顾名思义，就是在写发生时，才真正拷贝内存真正的数据，这个过程中，父进程也可能会产生阻塞的风险，就是下面介绍的场景。

b、fork出的子进程指向与父进程相同的内存地址空间，此时子进程就可以执行AOF重写，把内存中的所有数据写入到AOF文件中。但是此时父进程依旧是有流量写入的，如果父进程操作的是一个已经存在的key，那么这个时候父进程就会真正拷贝这个key对应的内存数据，申请新的内存空间，这样逐渐地，父子进程内存数据开始分离，父子进程逐渐拥有各自独立的内存空间。因为内存分配是以页为单位进行分配的，默认4k，如果父进程此时操作的是一个bigkey，重新申请大块内存耗时会变长，可能会产生阻塞风险。另外，如果操作系统开启了内存大页机制(Huge Page，页面大小2M)，那么父进程申请内存时阻塞的概率将会大大提高，所以在Redis机器上需要关闭Huge Page机制。Redis每次fork生成RDB或AOF重写完成后，都可以在Redis log中看到父进程重新申请了多大的内存空间。

问题2，AOF重写不复用AOF本身的日志，一个原因是父子进程写同一个文件必然会产生竞争问题，控制竞争就意味着会影响父进程的性能。二是如果AOF重写过程中失败了，那么原本的AOF文件相当于被污染了，无法做恢复使用。所以Redis AOF重写一个新文件，重写失败的话，直接删除这个文件就好了，不会对原先的AOF文件产生影响。等重写完成之后，直接替换旧文件即可。[53赞]

● 徐鹏 2020-08-12 09:01:21

有几个问题想请教一下：

- 1、文中多处提到bgrewriteaof子进程，这个有点迷糊，主线程fork出来的bgrewriteaof是子线程还是子进程？
- 2、AOF重写会拷贝一份完整的内存数据，这个会导致内存占用直接翻倍吗？
- 3、如果一个key设置了过期时间，在利用AOF文件恢复数据时，key已经过期了这个是如何处理的呢？[1



0赞]

- Darren 2020-08-12 10:41:14

AOF工作原理：

- 1、Redis 执行 fork() ，现在同时拥有父进程和子进程。
- 2、子进程开始将新 AOF 文件的内容写入到临时文件。
- 3、对于所有新执行的写入命令，父进程一边将它们累积到一个内存缓存中，一边将这些改动追加到现有 AOF 文件的末尾,这样即使在中途发生停机，现有的 AOF 文件也还是安全的。
- 4、当子进程完成重写工作时，它给父进程发送一个信号，父进程在接收到信号之后，将内存缓存中的所有数据追加到新 AOF 文件的末尾。
- 5、搞定！现在 Redis 原子地用新文件替换旧文件，之后所有命令都会直接追加到新 AOF 文件的末尾。

试着讨论下留言童鞋的几个问题

一、其中老师在文中提到：“因为主线程未阻塞，仍然可以处理新来的操作。此时，如果有写操作，第一处日志就是指正在使用的 AOF 日志，Redis 会把这个操作写到它的缓冲区。这样一来，即使宕机了，这个 AOF 日志的操作仍然是齐全的，可以用于恢复。”

这里面说到“Redis 会把这个操作写到它的缓冲区，这样一来，即使宕机了，这个 AOF 日志的操作仍然是齐全的”，其实对于有些人有理解起来可能不是那么好理解，因为写入缓冲区为什么还不都是数据；

我的理解其实这个就是写入缓冲区，只不过是由appendfsync策略决定的，所以说的不丢失数据指的是不会因为子进程额外丢失数据。

二、AOF重新只是回拷贝引用(指针)，不会拷贝数据本身，因此量其实不大，那写入的时候怎么办呢，写时复制，即新开辟空间保存修改的值，因此需要额外的内存，但绝对不是redis现在占有的2倍。

三、AOF对于过期key不会特殊处理，因为Redis keys过期有两种方式：被动和主动方式。

当一些客户端尝试访问它时，key会被发现并主动的过期。

当然，这样是不够的，因为有些过期的keys，永远不会访问他们。无论如何，这些keys应该过期，所以定时随机测试设置keys的过期时间。所有这些过期的keys将会从删除。

具体就是Redis每秒10次做的事情：

测试随机的20个keys进行相关过期检测。

删除所有已经过期的keys。

如果有多于25%的keys过期，重复步骤1.

至于课后问题，看了 @与路同飞 童鞋的答案，没有更好的答案，就不回答了 [7赞]

- 曾弢麟 2020-08-12 12:41:39

问题一:在AOF重写期间，Redis运行的命令会被积累在缓冲区，待AOF重写结束后会进行回放，在高并发情况下缓冲区积累可能会很大，这样就会导致阻塞，Redis后来通过Linux管道技术让aof期间就能同时进行回放，这样aof重写结束后只需要回放少量剩余的数据即可

问题二：对于任何文件系统都是不推荐并发修改文件的，例如hadoop的租约机制，Redis也是这样，避免重写发生故障，导致文件格式错乱最后aof文件损坏无法使用，所以Redis的做法是同时写两份文件，最后通过修改文件名的方式，保证文件切换的原子性

这里需要纠正一下老师前面的口误，就是Redis是通过使用操作系统的fork()方式创建进程，不是线程，也由于这个原因，主进程和fork出来的子进程的资源是不共享的，所以也出现Redis使用pipe管道技术来同步主进程的aof增量数据 [3赞]

• d 2020-08-12 19:26:23

AOF 是什么的缩写， 还是说就是这个名字？ [2赞]

• Anony 2020-08-12 09:04:07

上节说redis单线程处理网络IO和键值对的读写，持久化是由额外线程处理的。那AOF由额外线程处理的话，为什么会影响主线程呢？和主线程之间是什么关系？ [2赞]

• 与路同飞 2020-08-12 08:06:24

1.子线程重新AOF日志完成时会向主线程发送信号处理函数，会完成（1）将AOF重写缓冲区的内容写入到新的AOF文件中。（2）将新的AOF文件改名，原子地替换现有的AOF文件。完成以后才会重新处理客户端请求。

2.不共享AOF本身的日志是防止锁竞争，类似于redis rehash。 [2赞]

• 注定非凡 2020-08-14 09:06:19

1，作者讲了什么？

本章讲了Redis两种持久化机制之一：AOF机制原理

aof日志记录了redis所有增删改的操作，保存在磁盘上，当redis宕机，需要恢复内存中的数据时，可以通过读取aof日志恢复数据，从而避免因redis异常导致的数据丢失

2，作者是怎么把这事给讲明白的？

（1）作者先讲述redis宕机会导致内存数据丢失，需要有一种机制在redis重启后恢复数据。

（2）介绍了AOF通过记录每一个对redis数据进行增删改的操作日志，可以实现这种功能

（2）介绍了AOF的运行机制，数据保存机制，以及由此带来的优点和缺点

3，为了讲明白，作者讲了哪些要点，有哪些亮点？

（1）亮点：记录操作的时机分为：“写前日志和写后日志”，这个是我之前所不知道的

（2）要点1：AOF是写后日志，这样带来的好处是，记录的所有操作命令都是正确的，不需要额外的语法检查，确保redis重启时能够正确的读取回复数据

（3）要点2：AOF日志写入磁盘是比较影响性能的，为了平衡性能与数据安全，开发了三种机制：①：立即写入②：按秒写入③：系统写入

（4）要点3：AOF日志会变得巨大，所以Redis提供了日志重整的机制，通过读取内存中的数据重新产生一份数据写入日志

4，对于作者所讲，我有哪些发散性的思考？

作者说系统设计“取舍”二字非常重要，这是我之前未曾意识到的。作者讲了fork子进程机制，是Linux系统的一个能力，在刘超的课中讲过，这鼓舞了我继续学习的信心

5，将来有哪些场景，我可以应用上它？

目前还没有机会直接操作生产的redis配置，但现在要学习，争取将来可以直接操作

[1赞]

• 杨逸林 2020-08-12 19:55:49

老师没有说什么时候会触发重写 ROF 文件，我 Google 一下， 重写的触发方式有下面两个。

1. 手动执行 bgrewriteaof 触发 AOF 重写

2. 在 redis.conf 文件中配置重写的条件，如：





auto-aof-rewrite-min-size 64MB // 当文件小于64M时不进行重写

auto-aof-rewrite-min-percentage 100 // 当文件比上次重写后的文件大100%时进行重写

[原文](<https://www.jianshu.com/p/f72008c4d49f>)



还有传统关系型数据库并不是很"慢", 其实很多数据库都会做优化, 把热点数据放入内存。例如 Mysql 的 buffer\_pool, oracle 也会在你分页查询的时候, 缓存下一页, 只要你数据没改过, 也是很快的返回的。

老师说的 AOF 文件, 和 mysql binlog\_format=row 格式很像, 都是保存执行命令/语句, 只不过 mysql 是照搬 sql, 不会做这种最终数据一致的操作。还有 RDB 应该和 binlog\_format= statement 很像吧, 我看都是教一起开着的。redolog undolog binlog 难道天下数据库是一家, 思想理念照着抄吗?    



[1赞]

- 脱缰的野马\_\_ 2020-08-12 14:12:45

文章前面说到redo log日志记录的是修改后的数据, 但是在丁奇老师的MySQL实战中讲解的是redo log记录是对数据的操作记录, 修改后的数据是保存在内存的change buffer中的 [1赞]

- 秋梵 2020-08-12 13:10:31

老师我有个问题想问。文章里说, 在子进程进行重写期间, 如果有新的写操作, Redis 会把这个操作写到它的缓冲区。但是缓冲区也是存在内存中的, 如果宕机, 内存的数据会丢失, 新的写操作同样会丢失, 这样的话要怎么保证AOF的日志操作是齐全的? [1赞]

- test 2020-08-12 12:54:02

1.fork时候的开销

2.如果使用同一个文件会需要有文件锁的竞争 [1赞]

- 一步 2020-08-14 08:49:33

appendfsync: no 是系统控制进行写磁盘, 这里是利用文件系统的机制吗? 如果是利用的文件系统的哪个功能, 一般多久进行持久化写磁盘呢?

- 吴小智 2020-08-13 10:20:26

两个日志的合并, 是分离后, 主进程写的日志, 往子进程的重写日志里合并吧? 感觉文中这点没写清楚?

- Leslie\_Lamport 2020-08-13 03:00:50

“一个拷贝”就是指, 每次执行重写时, 主线程 fork 出后台的 bgrewriteaof 子进程。此时, fork 会把主线程的内存拷贝一份给 bgrewriteaof 子进程, 这里面就包含了数据库的最新数据。然后, bgrewriteaof 子进程就可以在不影响主线程的情况下, 逐一把拷贝的数据写成操作, 记入重写日志。

— 拷贝数据的cost多大?

- AstonPutting 2020-08-13 00:28:51

既然aof是以文本的形式记录redis的操作, 是不是也可以用于不同Redis版本之间的备份与恢复, 比如说redis4的aof日志恢复到redis3上

- ruier 2020-08-12 20:59:15

有个Django写的Redis管理小系统, 有兴趣的朋友可以看一看, 名字叫repoll  
<https://github.com/NaNShaner/repoll>

- 青生先森 2020-08-12 19:18:12

redis重写机制只针对还没有写入磁盘的信息吗?

- 寰宇寰宇 2020-08-12 17:04:46

aof重写的触发条件：

1. 手动执行bgrewriteaof触发aof重写
2. 配置redis.conf文件的aof重写条件，如auto-aof-rewrite-min-size和auto-aof-rewrite-min-percentage

- Archer30 2020-08-12 16:49:55

问题1回答：如果子进程写入事件过长，并且这段事件，会导致AOF重写日志，积累过多，当新的AOF文件完成后，还是需要写入大量AOF重写日志里的内容，可能会导致阻塞。

问题2回答：我觉得评论区里的大部分回答 防止锁竞争，应该是把问题理解错了，父子两个进程本来就没有需要竞争的数据，老师所指的两个日志应该是“AOF缓冲区”和“AOF重写缓冲区”，而不是磁盘上的AOF文件，之所有另外有一个“AOF重写缓冲区”，是因为重写期间，主进程AOF还在继续工作，还是会同步到旧的AOF文件中，同步成功后，“AOF缓冲区”会被清除，会被清除，会被清除！