19-波动的响应延迟：如何应对变慢的Redis？（下）

你好，我是蒋德钧。

上节课，我介绍了判断Redis变慢的两种方法，分别是响应延迟和基线性能。除此之外，我还给你分享了从Redis的自身命令操作层面排查和解决问题的两种方案。

但是，如果在排查时，你发现Redis没有执行大量的慢查询命令，也没有同时删除大量过期keys，那么，我们是不是就束手无策了呢？

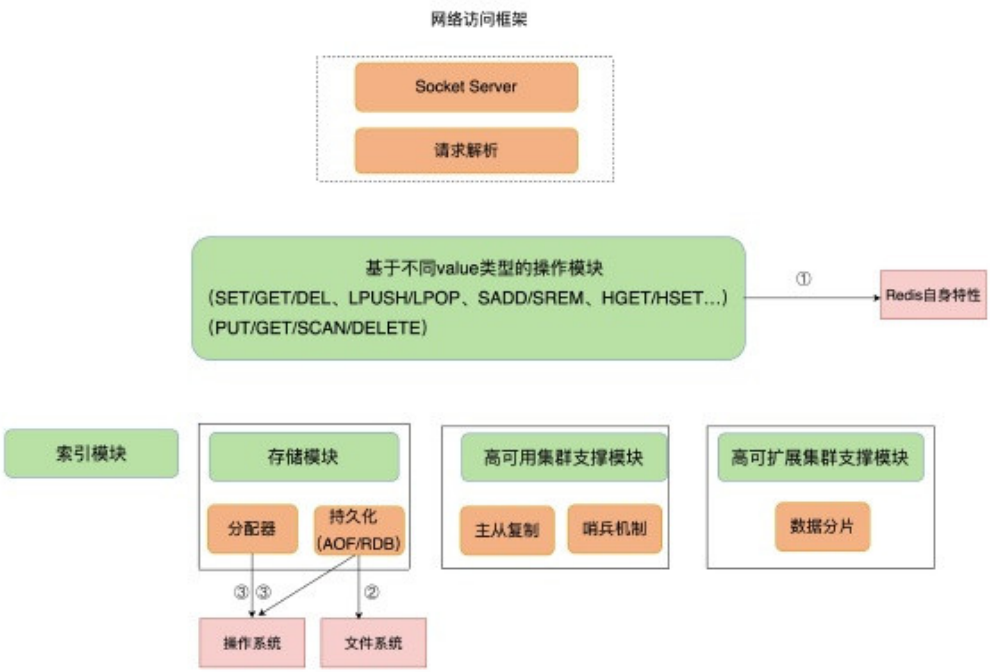
当然不是！我还有很多“锦囊妙计”，准备在这节课分享给你呢！

如果上节课的方法不管用，那就说明，你要关注影响性能的其他机制了，也就是文件系统和操作系统。

Redis会持久化保存数据到磁盘，这个过程要依赖文件系统来完成，所以，文件系统将数据写回磁盘的机制，会直接影响到Redis持久化的效率。而且，在持久化的过程中，Redis也还在接收其他请求，持久化的效率高低又会影响到Redis处理请求的性能。

另一方面，Redis是内存数据库，内存操作非常频繁，所以，操作系统的内存机制会直接影响到Redis的处理效率。比如说，如果Redis的内存不够用了，操作系统会启动swap机制，这就会直接拖慢Redis。

那么，接下来，我再从这两个层面，继续给你介绍，如何进一步解决Redis变慢的问题。



文件系统：AOF模式

你可能会问，Redis是个内存数据库，为什么它的性能还和文件系统有关呢？

我在前面讲过，为了保证数据可靠性，Redis会采用AOF日志或RDB快照。其中，AOF日志提供了三种日志写回策略：no、everysec、always。这三种写回策略依赖文件系统的两个系统调用完成，也就是write和fsync。

write只要把日志记录写到内核缓冲区，就可以返回了，并不需要等待日志实际写回到磁盘；而fsync需要把日志记录写回到磁盘后才能返回，时间较长。下面这张表展示了三种写回策略所执行的系统调用。

AOF写回策略	执行的系统调用
no	调用write写日志文件，由操作系统周期性地将日志写回磁盘
everysec	每秒调用一次fsync，将日志写回磁盘
always	每执行一个操作，就调用一次fsync将日志写回磁盘

当写回策略配置为everysec和always时，Redis需要调用fsync把日志写回磁盘。但是，这两种写回策略的具体执行情况还不太一样。

在使用everysec时，Redis允许丢失一秒的操作记录，所以，Redis主线程并不需要确保每个操作记录日志都写回磁盘。而且，fsync的执行时间很长，如果是在Redis主线程中执行fsync，就容易阻塞主线程。所以，当写回策略配置为everysec时，Redis会使用后台的子线程异步完成fsync的操作。

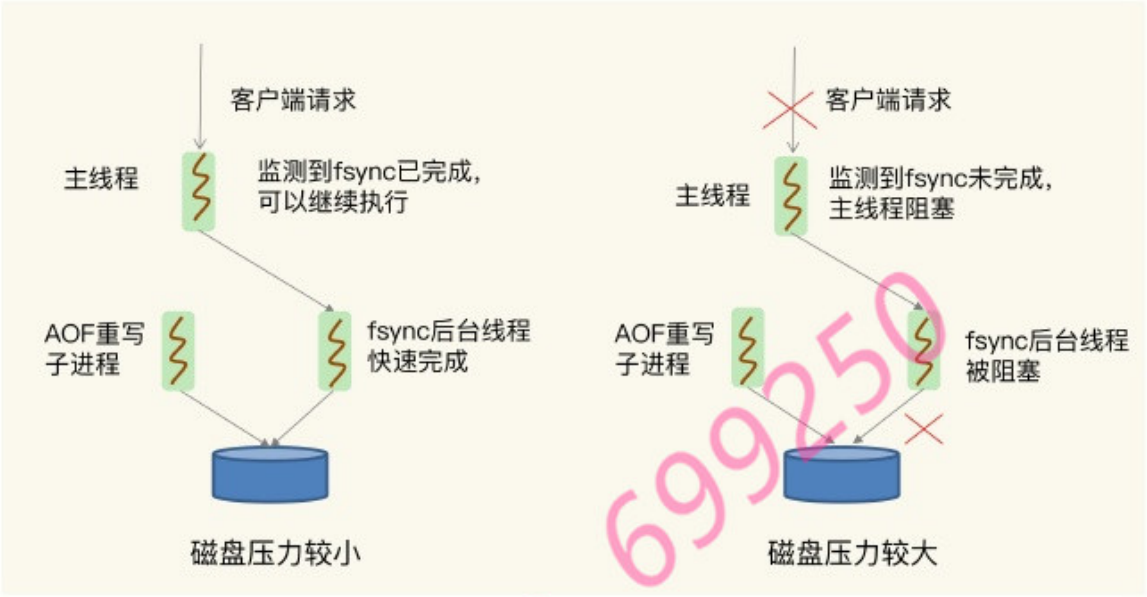
而对于always策略来说，Redis需要确保每个操作记录日志都写回磁盘，如果用后台子线程异步完成，主线程就无法及时地知道每个操作是否已经完成了，这就不符合always策略的要求了。所以，always策略并不使用后台子线程来执行。

另外，在使用AOF日志时，为了避免日志文件不断增大，Redis会执行AOF重写，生成体量缩小的新的AOF日志文件。AOF重写本身需要的时间很长，也容易阻塞Redis主线程，所以，Redis使用子进程来进行AOF重写。

但是，这里有一个潜在的风险点：AOF重写会对磁盘进行大量IO操作，同时，fsync又需要等到数据写到磁盘后才能返回，所以，当AOF重写的压力比较大时，就会导致fsync被阻塞。虽然fsync是由后台子线程负责执行的，但是，主线程会监控fsync的执行进度。

当主线程使用后台子线程执行了一次fsync，需要再次把新接收的操作记录写回磁盘时，如果主线程发现上一次的fsync还没有执行完，那么它就会阻塞。所以，如果后台子线程执行的fsync频繁阻塞的话（比如AOF重写占用了大量的磁盘IO带宽），主线程也会阻塞，导致Redis性能变慢。

为了帮助你理解，我再画一张图来展示下在磁盘压力小和压力大的时候，fsync后台子线程和主线程受到的影响。



好了，说到这里，你已经了解了，由于fsync后台子线程和AOF重写子进程的存在，主IO线程一般不会被阻塞。但是，如果在重写日志时，AOF重写子进程的写入量比较大，fsync线程也会被阻塞，进而阻塞主线程，导致延迟增加。现在，我来给出排查和解决建议。

首先，你可以检查下Redis配置文件中的appendfsync配置项，该配置项的取值表明了Redis实例使用的是哪种AOF日志写回策略，如下所示：

写回策略	配置项
no	appendfsync no
everysec	appendfsync everysec
always	appendfsync always

如果AOF写回策略使用了everysec或always配置，请先确认下业务方对数据可靠性的要求，明确是否需要每一秒或每一个操作都记日志。有的业务方不了解Redis AOF机制，很可能就直接使用数据可靠性最高等级的always配置了。其实，在有些场景中（例如Redis用于缓存），数据丢了还可以从后端数据库中获取，并不需要很高的数据可靠性。

如果业务应用对延迟非常敏感，但同时允许一定量的数据丢失，那么，可以把配置项no-appendfsync-on-rewrite设置为yes，如下所示：

```
no-appendfsync-on-rewrite yes
```

这个配置项设置为yes时，表示在AOF重写时，不进行fsync操作。也就是说，Redis实例把写命令写到内存后，不调用后台线程进行fsync操作，就可以直接返回了。当然，如果此时实例发生宕机，就会导致数据丢失。反之，如果这个配置项设置为no（也是默认配置），在AOF重写时，Redis实例仍然会调用后台线程进行fsync操作，这就会给实例带来阻塞。

如果的确需要高性能，同时也需要高可靠数据保证，我建议你考虑**采用高速的固态硬盘作为AOF日志的写入设备。**

高速固态盘的带宽和并发度比传统的机械硬盘的要高出10倍及以上。在AOF重写和fsync后台线程同时执行时，固态硬盘可以提供较为充足的磁盘IO资源，让AOF重写和fsync后台线程的磁盘IO资源竞争减少，从而降低对Redis的性能影响。

操作系统：swap

如果Redis的AOF日志配置只是no，或者就没有采用AOF模式，那么，还会有什么问题导致性能变慢吗？

接下来，我就再说一个潜在的瓶颈：**操作系统的内存swap**。

内存swap是操作系统里将内存数据在内存和磁盘间来回换入和换出的机制，涉及到磁盘的读写，所以，一旦触发swap，无论是被换入数据的进程，还是被换出数据的进程，其性能都会受到慢速磁盘读写的影响。

Redis是内存数据库，内存使用量大，如果没有控制好内存的使用量，或者和其他内存需求大的应用一起运行了，就可能受到swap的影响，而导致性能变慢。

这一点对于Redis内存数据库而言，显得更为重要：正常情况下，Redis的操作是直接通过访问内存就能完成，一旦swap被触发了，Redis的请求操作需要等到磁盘数据读写完成才行。而且，和我刚才说的AOF日志文件读写使用fsync线程不同，swap触发后影响的是Redis主IO线程，这会极大地增加Redis的响应时间。

说到这儿，我想给你分享一个我曾经遇到过的因为swap而导致性能降低的例子。

在正常情况下，我们运行的一个实例完成5000万个GET请求时需要300s，但是，有一次，这个实例完成5000万GET请求，花了将近4个小时的时间。经过问题复现，我们发现，当时Redis处理请求用了近4小时的情况下，该实例所在的机器已经发生了swap。从300s到4个小时，延迟增加了将近48倍，可以看到swap对性能造成的严重影响。

那么，什么时候会触发swap呢？

通常，触发swap的原因主要是**物理机器内存不足**，对于Redis而言，有两种常见的情况：

- Redis实例自身使用了大量的内存，导致物理机器的可用内存不足；
- 和Redis实例在同一台机器上运行的其他进程，在进行大量的文件读写操作。文件读写本身会占用系统内存，这会导致分配给Redis实例的内存量变少，进而触发Redis发生swap。

针对这个问题，我也给你提供一个解决思路：**增加机器的内存或者使用Redis集群**。

操作系统本身会在后台记录每个进程的swap使用情况，即有多少数据量发生了swap。你可以先通过下面的命令查看Redis的进程号，这里是5332。

```
$ redis-cli info | grep process_id
process_id: 5332
```

然后，进入Redis所在机器的/proc目录下的该进程目录中：

```
$ cd /proc/5332
```

最后，运行下面的命令，查看该Redis进程的使用情况。在这儿，我只截取了部分结果：

```
$cat smaps | egrep '^(Swap|Size)'
Size: 584 kB
Swap: 0 kB
Size: 4 kB
Swap: 4 kB
Size: 4 kB
Swap: 0 kB
Size: 462044 kB
Swap: 462008 kB
Size: 21392 kB
Swap: 0 kB
```


每一行Size表示的是Redis实例所用的一块内存大小，而Size下方的Swap和它相对应，表示这块Size大小的内存区域有多少已经被换出到磁盘上了。如果这两个值相等，就表示这块内存区域已经完全被换出到磁盘了。

作为内存数据库，Redis本身会使用很多大小不一的内存块，所以，你可以看到有很多Size行，有的很小，就是4KB，而有的很大，例如462044KB。**不同内存块被换出到磁盘上的大小也不一样**，例如刚刚的结果中的第一个4KB内存块，它下方的Swap也是4KB，这表示这个内存块已经被换出了；另外，462044KB这个内存块也被换出了462008KB，差不多有462MB。

这里有个重要的地方，我得提醒你一下，当出现百MB，甚至GB级别的swap大小时，就表明，此时，Redis实例的内存压力很大，很有可能会变慢。所以，swap的大小是排查Redis性能变慢是否由swap引起的重要指标。

一旦发生内存swap，最直接的解决方法就是**增加机器内存**。如果该实例在一个Redis切片集群中，可以增加Redis集群的实例个数，来分摊每个实例服务的数据量，进而减少每个实例所需的内存量。

当然，如果Redis实例和其他操作大量文件的程序（例如数据分析程序）共享机器，你可以将Redis实例迁移到单独的机器上运行，以满足它的内存需求量。如果该实例正好是Redis主从集群中的主库，而从库的内存很大，也可以考虑进行主从切换，把大内存的从库变成主库，由它来处理客户端请求。

操作系统：内存大页

除了内存swap，还有一个和内存相关的因素，即内存大页机制（Transparent Huge Page, THP），也会影响Redis性能。

Linux内核从2.6.38开始支持内存大页机制，该机制支持2MB大小的内存页分配，而常规的内存页分配是按4KB的粒度来执行的。

很多人都觉得：“Redis是内存数据库，内存大页不正好可以满足Redis的需求吗？而且在分配相同的内存量时，内存大页还能减少分配次数，不也是对Redis友好吗？”

其实，系统的设计通常是一个取舍过程，我们称之为trade-off。很多机制通常都是优势和劣势并存的。Redis使用内存大页就是一个典型的例子。

虽然内存大页可以给Redis带来内存分配方面的收益，但是，不要忘了，Redis为了提供数据可靠性保证，需要将数据做持久化保存。这个写入过程由额外的线程执行，所以，此时，Redis主线程仍然可以接收客户端写请求。客户端的写请求可能会修改正在进行持久化的数据。在这一过程中，Redis就会采用写时复制机制，也就是说，一旦有数据要被修改，Redis并不会直接修改内存中的数据，而是将这些数据拷贝一份，然后再进行修改。

如果采用了内存大页，那么，即使客户端请求只修改100B的数据，Redis也需要拷贝2MB的大页。相反，如果是常规内存页机制，只用拷贝4KB。两者相比，你可以看到，当客户端请求修改或新写入数据较多时，内存大页机制将导致大量的拷贝，这就会影响Redis正常的访存操作，最终导致性能变慢。

那该怎么办呢？很简单，关闭内存大页，就行了。

首先，我们要先排查下内存大页。方法是：在Redis实例运行的机器上执行如下命令：

```
cat /sys/kernel/mm/transparent_hugepage/enabled
```

如果执行结果是always，就表明内存大页机制被启动了；如果是never，就表示，内存大页机制被禁止。

在实际生产环境中部署时，我建议你不要使用内存大页机制，操作也很简单，只需要执行下面的命令就可以了：

```
echo never /sys/kernel/mm/transparent_hugepage/enabled
```

小结

这节课，我从文件系统和操作系统两个维度，给你介绍了应对Redis变慢的方法。

为了方便你应用，我给你梳理了一个包含9个检查点的Checklist，希望你在遇到Redis性能变慢时，按照这些步骤逐一检查，高效地解决问题。

1. 获取Redis实例在当前环境下的基线性能。
2. 是否用了慢查询命令？如果是的话，就使用其他命令替代慢查询命令，或者把聚合计算命令放在客户端
   ...

3. 是否对过期key设置了相同的过期时间？对于批量删除的key，可以在每个key的过期时间上加一个随机数，避免同时删除。
4. 是否存在bigkey？ 对于bigkey的删除操作，如果你的Redis是4.0及以上的版本，可以直接利用异步线程机制减少主线程阻塞；如果是Redis 4.0以前的版本，可以使用SCAN命令迭代删除；对于bigkey的集合查询和聚合操作，可以使用SCAN命令在客户端完成。
5. Redis AOF配置级别是什么？业务层面是否的确需要这一可靠性级别？如果我们需要高性能，同时也允许数据丢失，可以将配置项no-appendfsync-on-rewrite设置为yes，避免AOF重写和fsync竞争磁盘IO资源，导致Redis延迟增加。当然， 如果既需要高性能又需要高可靠性，最好使用高速固态盘作为AOF日志的写入盘。
6. Redis实例的内存使用是否过大？发生swap了吗？如果是的话，就增加机器内存，或者是使用Redis集群，分摊单机Redis的键值对数量和内存压力。同时，要避免出现Redis和其他内存需求大的应用共享机器的情况。
7. 在Redis实例的运行环境中，是否启用了透明大页机制？如果是的话，直接关闭内存大页机制就行了。
8. 是否运行了Redis主从集群？如果是的话，把主库实例的数据量大小控制在2~4GB，以免主从复制时，从库因加载大的RDB文件而阻塞。
9. 是否使用了多核CPU或NUMA架构的机器运行Redis实例？使用多核CPU时，可以给Redis实例绑定物理核；使用NUMA架构时，注意把Redis实例和网络中断处理程序运行在同一个CPU Socket上。

实际上，影响系统性能的因素还有很多，这两节课给你讲的都是应对最常见问题的解决方案。

如果你遇到了一些特殊情况，也不要慌，我再给你分享一个小技巧：仔细检查下有没有恼人的“邻居”，具体点说，就是Redis所在的机器上有没有一些其他占内存、磁盘IO和网络IO的程序，比如说数据库程序或者数据采集程序。如果有的话，我建议你将这些程序迁移到其他机器上运行。

为了保证Redis高性能，我们需要给Redis充足的计算、内存和IO资源，给它提供一个“安静”的环境。

每课一问

这两节课，我向你介绍了系统性定位、排查和解决Redis变慢的方法。所以，我想请你聊一聊，你遇到过Redis变慢的情况吗？如果有的话，你是怎么解决的呢？

欢迎你在留言区分享一下自己的经验，如果觉得今天的内容对你有所帮助，也欢迎分享给你的朋友或同事，我们下节课见。

精选留言：

- Kaito 2020-09-21 00:06:01
  关于如何分析、排查、解决Redis变慢问题，我总结的checklist如下：

  1、使用复杂度过高的命令（例如SORT/SUION/ZUNIONSTORE/KEYS），或一次查询全量数据（例如LRANGE key 0 N，但N很大）

  分析：a) 查看slowlog是否存在这些命令 b) Redis进程CPU使用率是否飙升（聚合运算命令导致）

  解决：a) 不使用复杂度过高的命令，或用其他方式代替实现（放在客户端做） b) 数据尽量分批查询（LRANGE key 0 N，建议N<=100，查询全量数据建议使用HSCAN/SSCAN/ZSCAN）

  2、操作bigkey

分析：a) slowlog出现很多SET/DELETE变慢命令（bigkey分配内存和释放内存变慢） b) 使用redis-cli -h $host -p $port --bigkeys扫描出很多bigkey

解决：a) 优化业务，避免存储bigkey b) Redis 4.0+可开启lazy-free机制

3、大量key集中过期

分析：a) 业务使用EXPIREAT/PEXPIREAT命令 b) Redis info中的expired_keys指标短期突增

解决：a) 优化业务，过期增加随机时间，把时间打散，减轻删除过期key的压力 b) 运维层面，监控expired_keys指标，有短期突增及时报警排查

4、Redis内存达到maxmemory

分析：a) 实例内存达到maxmemory，且写入量大，淘汰key压力变大 b) Redis info中的evicted_keys指标短期突增

解决：a) 业务层面，根据情况调整淘汰策略（随机比LRU快） b) 运维层面，监控evicted_keys指标，有短期突增及时报警 c) 集群扩容，多个实例减轻淘汰key的压力

5、大量短连接请求

分析：Redis处理大量短连接请求，TCP三次握手和四次挥手也会增加耗时

解决：使用长连接操作Redis

6、生成RDB和AOF重写fork耗时严重

分析：a) Redis变慢只发生在生成RDB和AOF重写期间 b) 实例占用内存越大，fork拷贝内存页表越久 c) Redis info中latest_fork_usec耗时变长

解决：a) 实例尽量小 b) Redis尽量部署在物理机上 c) 优化备份策略（例如低峰期备份） d) 合理配置repl-backlog和slave client-output-buffer-limit，避免主从全量同步 e) 视情况考虑关闭AOF f) 监控latest_fork_usec耗时是否变长

7、AOF使用awalys机制

分析：磁盘IO负载变高

解决：a) 使用everysec机制 b) 丢失数据不敏感的业务不开启AOF

8、使用Swap

分析：a) 所有请求全部开始变慢 b) slowlog大量慢日志 c) 查看Redis进程是否使用到了Swap

解决：a) 增加机器内存 b) 集群扩容 c) Swap使用时监控报警

9、进程绑定CPU不合理

分析：a) Redis进程只绑定一个CPU逻辑核 b) NUMA架构下，网络中断处理程序和Redis进程没有绑定在同一个Socket下

解决：a) Redis进程绑定多个CPU逻辑核 b) 网络中断处理程序和Redis进程绑定在同一个Socket下

10、开启透明大页机制

分析：生成RDB和AOF重写期间，主线程处理写请求耗时变长（拷贝内存副本耗时变长）

解决：关闭透明大页机制

11、网卡负载过高

分析：a) TCP/IP层延迟变大，丢包重传变多 b) 是否存在流量过大的实例占满带宽

解决：a) 机器网络资源监控，负载过高及时报警 b) 提前规划部署策略，访问量大的实例隔离部署

总之，Redis的性能与CPU、内存、网络、磁盘都息息相关，任何一处发生问题，都会影响到Redis的性能。

主要涉及到的包括业务使用层面和运维层面：业务人员需要了解Redis基本的运行原理，使用合理的命令、规避bigke问题和集中过期问题。运维层面需要DBA提前规划好部署策略，预留足够的资源，同时做好监控，这样当发生问题时，能够及时发现并尽快处理。 [27赞]

- 岑子华宝宝萌萌哒 2020-09-21 09:58:01

  echo never /sys/kernel/mm/transparent_hugepage/enabled
  这个是不是得改成
  echo never > /sys/kernel/mm/transparent_hugepage/enabled
  这样？ [3赞]

- yeek 2020-09-21 08:59:03
  如果redis是独立部署，那么当内存不足时，淘汰策略和操作系统的swap机制 哪个会优先执行？

  曾遇到过线上触发内存淘汰的场景，并未观察当时的swap情况，感谢老师的建议 [1赞]

- 东 2020-09-21 08:33:40
  “8. 是否运行了 Redis 主从集群？如果是的话，把主库实例的数据量大小控制在 2~4GB，以免主从复制时，从库因加载大的 RDB 文件而阻塞。” 这个2~4G的经验值和主库的内存大小无关吗？比如主从库内存都是64G， 这个主库数据量依然是2~4G？ [1赞]

- 云学 2020-09-21 23:35:33
  除了绑定cpu也可以提升redis进程静态优先级，得到更多cpu调度

- 土豆白菜 2020-09-21 21:13:04
  老师会讲布隆过滤器吗？

- tt 2020-09-21 09:56:17
  可以再仔细分析一下，在虚拟机上部署REDIS实例时，由于虚拟化软件本身的内存管理算法导致的SWAP

- yeek 2020-09-21 08:51:01

  当主线程使用后台子线程执行了一次 fsync，需要再次把新接收的操作记录写回磁盘时，如果主线程发现上一次的 fsync 还没有执行完，那么它就会阻塞。所以，如果后台子线程执行的 fsync 频繁阻塞的话（比如 AOF 重写占用了大量的磁盘 IO 带宽），主线程也会阻塞，导致 Redis 性能变慢。

  这段没懂，redis主线程和后台子线程之间有状态通信吗？主线程调用fsync对子线程而言是任务队列的方式还是同步调用的方式？ 我去看看源码吧……