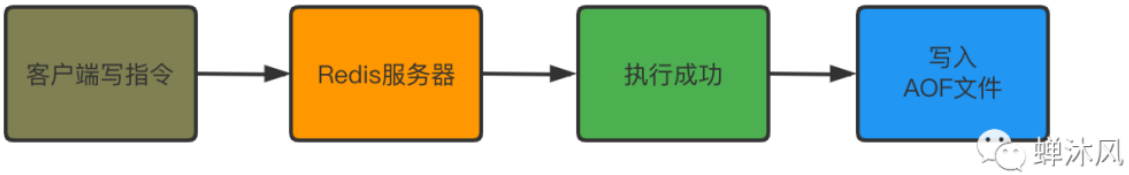
Redis持久化策略——AOF

什么是AOF

AOF(Append Only File)以文本的形式（文本格式由Redis自定义，后文会讲到），通过将所有对数据库的写入命令记录到AOF文件中，达到记录数据库状态的目的。

注意：AOF文件只会记录Redis的写操作命令，因为读命令对数据的恢复没有任何意义



Redis默认并未开启AOF功能，redis.conf配置文件中，关于AOF的相关配置如下

```
# 是否开启AOF功能（开启：yes 关闭：no）
appendonly yes
# 生成的AOF文件名称
appendfilename 6379.aof
# AOF写回策略
appendfsync everysec
# 当前AOF文件大小和最后一次重写后的大小之间的比率>=指定的增长百分比则进行重写
# 如100代表当前AOF文件大小是上次重写的两倍时候才重写
auto-aof-rewrite-percentage 100
# AOF文件最小重写大小，只有当AOF文件大小大于该值时候才可能重写,4.0默认配置64mb。
auto-aof-rewrite-min-size 64mb
```

AOF日志的生成过程

从我们发送写指令开始到指令保存在AOF文件中，需要经历4步，分别为命令传播、命令追加、文件写入和文件同步。

Everysec

如果用户未指定appendOnlyFile的值，则默认值为Everysec。每秒同步，每个写命令执行完，只是先把日志写到 AOF文件的内核缓冲区，理论上每隔1秒把缓冲区中的内容同步到磁盘，且同步操作有单独的子线程进行，因此不会阻塞主进程。

需要注意的是，我们用的是「理论上」这样的措辞，实际运行中该模式对fsync或fdatasync的调用并不是每秒一次，而是和调用flushAppendOnlyFile函数时Redis所处的状态有关。

每当 flushAppendOnlyFile 函数被调用时， 可能会出现以下四种情况：

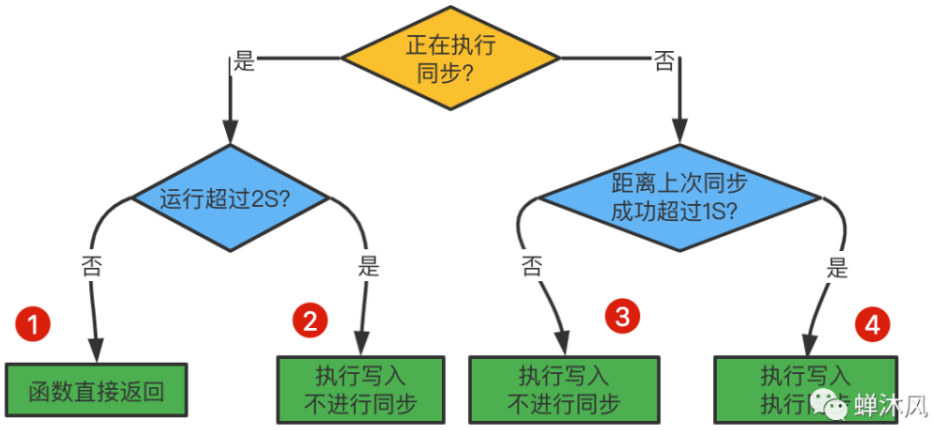
子线程正在执行同步，并且

- 这个同步的执行时间未超过 2 秒，那么程序直接返回；
- 这个同步已经执行超过 2 秒，那么程序执行写入操作，但不执行新的同步操作。但是，这时的写入操作必须等待子线程先完成原本的同步操作，因此这里的写入操作会比平时阻塞更长时间。

子线程没有在执行同步 ，并且

- 上次成功执行同步距今不超过1秒，那么程序执行写入，但不执行同步；
- 上次成功执行同步距今已经超过1秒，那么程序执行写入和同步 。

可以用流程图表示这四种情况：



在Everysec模式下

- 如果在情况1下宕机，那么我们最多损失小于2秒内的所有数据。
- 如果在情况2下宕机，那么我们损失的数据可能会超过2秒。

因此AOF在Everysec模式下只会丢失 1 秒钟数据的说法实际上并不准确。

Always

每个写命令执行完，立刻同步地将日志写回磁盘。此模式下同步操作是由 Redis 主进程执行的，所以在同步执行期间，主进程会被阻塞，不能接受命令请求。

AOF同步策略小结

对于三种 AOF 同步模式， 它们对Redis主进程的阻塞情况如下：

- 不同步（No）：写入和同步都由主进程执行，两个操作都会阻塞主进程；
- 每一秒钟同步一次（Everysec）：写入操作由主进程执行，阻塞主进程。同步操作由子线程执行，不直接阻塞主进程，但同步操作完成的快慢会影响写入操作的阻塞时长；
- 每执行一个命令同步一次（Always）：同模式 1 。

因为阻塞操作会让 Redis 主进程无法持续处理请求， 所以一般说来， 阻塞操作执行得越少、完成得越快， Redis 的性能就越好。

No的同步操作只会在AOF关闭或Redis关闭时执行， 或由操作系统内核触发。在一般情况下， 这种模式只需要为写入阻塞，因此它的写入性能要比后面两种模式要高， 但是这种性能的提高是以降低安全性为代价的：在这种模式下，如果发生宕机，那么丢失的数据量由操作系统内核的缓存冲洗策略决定。

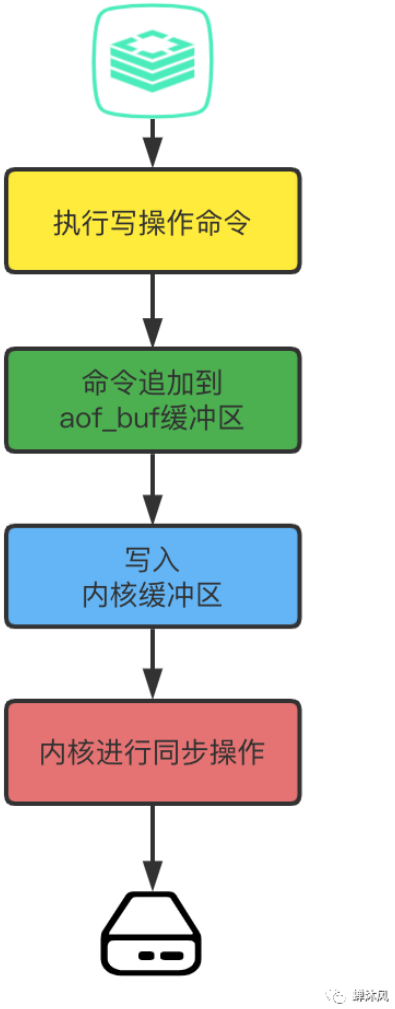
Everysec在性能方面要优于Always ， 并且在通常情况下，这种模式最多丢失不多于2秒的数据， 所以它的安全性要高于No ，这是一种兼顾性能和安全性的保存方案。

Always的安全性是最高的，但性能也是最差的，因为Redis必须阻塞直到命令信息被写入并同步到磁盘之后才能继续处理请求。

三种AOF模式的特性可以总结为如下表格

同步策略	同步时机	写入是否阻塞	同步是否阻塞	宕机丢失的数据量	优点	缺点
No	操作系统控制	阻塞	阻塞	操作系统最后一次同步之后的数据	性能最高	可靠性最低
Everysec	近似每秒同步	阻塞	不阻塞	一般情况下不超过2S的数据	性能适中	宕机丢失一部分数据
Always	每次都同步	阻塞	阻塞	最多丢失一个命令的数据	可靠性最高，数据基本不丢失	性能差，因为每个写操作都要同步磁盘

AOF生成过程小结



最后总结一下AOF文件的生成过程。以下步骤都是在AOF开启的前提下进行的

- Redis成功执行写操作指令，然后将写的指令按照自定义格式追加到aof_buf缓冲区，这是第一个缓冲区；
- Redis主进程将aof_buf缓冲区的数据写入到内核缓冲区，这是第二个缓冲区；
- 根据AOF同步策略适时地将内核缓冲区的数据同步到磁盘，过程结束。

AOF文件的载入和数据还原

AOF文件中包含了能够重建数据库的所有写命令，因此将所有命令读入并依次执行即可还原Redis之前的数据状态。

Redis 读取AOF文件并还原数据库的详细步骤如下：

- 创建一个不带网络连接的伪客户端（fake client），伪客户端执行命令的效果，和带网络连接的客户端执行命令的效果完全相同；
- 读取AOF所保存的文本，并根据内容还原出命令、命令的参数以及命令的个数；
- 根据指令、指令的参数等信息，使用伪客户端执行命令。

- 执行 2 和 3，直到AOF文件中的所有命令执行完毕。

注意：为了避免对数据的完整性产生影响， 在服务器载入数据的过程中， 只有和数据库无关的发布订阅功能可以正常使用， 其他命令一律返回错误。

AOF重写

AOF的作用是帮我们还原Redis的数据状态，其中包含了所有的写操作，但是正常情况下客户端会对同一个KEY进行多次不同的写操作，如下

```
127.0.0.1:6379[3]> SET name chanmufeng1
OK
127.0.0.1:6379[3]> SET name chanmufeng2
OK
127.0.0.1:6379[3]> SET name chanmufeng3
OK
127.0.0.1:6379[3]> SET name chanmufeng4
OK
127.0.0.1:6379[3]> SET name chanmufeng
OK
```

例子中对name的数据进行写操作就进行了5次，其实对我们而言仅需要最后一条指令而已，但是AOF会将这5条指令都记录下来。更极端的情况是有些被频繁操作的键， 对它们所调用的命令可能有成百上千、甚至上万条， 如果这样被频繁操作的键有很多的话，AOF文件的体积就会急速膨胀。

首先，AOF文件的体积受操作系统大小的限制，本身就不能无限增长；

其次，体积过于庞大的AOF文件会影响指令的写入速度，阻塞时间延长；

最后AOF文件的体积越大，Redis数据恢复所需的时间也就越长。

为了解决AOF文件体积庞大的问题，Redis提供了rewrite的AOF重写功能来精简AOF文件体积。

AOF重写的实现原理

虽然叫AOF「重写」，但是新AOF文件的生成并非是在原AOF文件的基础上进行操作得到的，而是读取Redis当前的数据状态来重新生成的。不难理解，后者的处理方式远比前者高效。

为了避免阻塞主线程，导致数据库性能下降，和 AOF 日志由主进程写回不同，重写过程是由子进程执行bgrewriteaof来完成的。这样处理的最大好处是：

- 子进程进行 AOF重写期间，主进程可以继续处理命令请求；
- 子进程带有主进程的数据副本，操作效率更高。

这里有两个问题值得我们来思考一下

1.为什么使用子进程，而不是多线程来进行AOF重写呢？

如果是使用线程，线程之间会共享内存，在修改共享内存数据的时候，需要通过加锁来保证数据的安全，这样就会降低性能。

如果使用子进程，操作系统会使用**「写时复制」**的技术：fork子进程时，子进程会拷贝父进程的页表，即虚实映射关系，而不会拷贝物理内存。子进程复制了父进程页表，也能共享访问父进程的内存数据，达到共享内存的效果。

RDB和AOF对比

RDB的优点：

- RDB文件非常紧凑，节省内存空间；
- RDB 在恢复大数据集时的速度比 AOF 的恢复速度要快；
- 适合全量备份、全量复制的场景，经常用于灾难恢复（对数据的完整性和一致性要求相对较低的场合）

RDB的缺点：

- 服务器宕机时，可能会丢失部分数据；
- 每次保存RDB的时候，Redis都要fork出一个子进程，这个过程是阻塞的，如果数据集巨大，那阻塞的时间就会很长。

AOF的优点：

- 数据更加完整，丢失数据的可能性较低；
- AOF日志文件可读，并且可以对AOF文件修复。

AOF的缺点：

- AOF日志记录在长期运行中逐渐庞大，恢复起来非常耗时，需要定期对AOF日志进行瘦身处理；
- 恢复备份速度比较慢。

- 如何选择使用哪种持久化方式?

一般来说， 如果想达到足以媲美 PostgreSQL 的数据安全性， 你应该同时使用两种持久化功能。

如果你非常关心你的数据， 但仍然可以承受数分钟以内的数据丢失， 那么你可以只使用 RDB 持久化。

有很多用户都只使用 AOF 持久化， 但我们并不推荐这种方式： 因为定时生成 RDB 快照（snapshot）非常便于进行数据库备份， 并且 RDB 恢复数据集的速度也要比 AOF 恢复的速度要快， 除此之外， 使用 RDB 还可以避免之前提到的 AOF 程序的 bug 。

Note: 因为以上提到的种种原因， 未来我们可能会将 AOF 和 RDB 整合成单个持久化模型， （这是一个长期计划。）接下来的几个小节将介绍 RDB 和 AOF 的更多细节。