redis 的持久化方式

对于persistence持久化存储，Redis提供了两种持久化方法：

- Redis DataBase(简称RDB)
  - 执行机制：快照，直接将databases中的key-value的二进制形式存储在了rdb文件中
  - 优点：性能较高（因为是快照，且执行频率比aof低，而且rdb文件中直接存储的是key-values的二进制形式，对于恢复数据也快）
  - **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：在save配置条件之间若发生宕机，此间的数据会丢失
  - **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**
- Append-only file (简称AOF)
  - 执行机制：将对数据的每一条修改命令追加到aof文件
  - 优点：数据不容易丢失
  - **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：性能较低（每一条修改操作都要追加到aof文件，执行频率较RDB要高，而且aof文件中存储的是命令，对于恢复数据来讲需要逐行执行命令，所以恢复慢）
  - **AOF文件比RDB文件大，且恢复速度慢。**

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。

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

#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

##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  (即禁止使用rdb)

#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

#当后台RDB进程导出快照（一部分的key-value）到rdb文件这个过程出错时（即最后一次的后台保存失败时），

#该种方式会让用户知道在数据持久化到硬盘时出错了（相当于一种监控）；

#如果安装了很好的redis持久化监控，可设置为"no"

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 (如果希望RDB进程节省一点CPU时间，设置为no，但是可能最后的rdb文件会很大)

#在redis重启后，从rdb文件向内存写数据之前，是否先检测该rdb文件是否损坏(根据rdb文件中的校验和check_sum)
```

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。

- 以上三条save命令只要发生任一条，bgsave命令都会发生，这就有两个问题，假设60s内有10000个key发生了改变（写入、删除、更新），那么是否会立即进行持久化呢？在这次持久化之后，假设又过了240s，而在此期间没有任何的key的改变操作，此时是否要发生一次持久化（因为满足300s发生了10个key的改变，这里是改变了10000个key）？
  - 不会立即进行持久化：redis默认每隔100ms使用serverCron函数检查一次save配置的条件是否满足，满足则进行bgsave，这样的话，如果在100ms内，我已经满足了bgsave的条件，那么我真正执行bgsave的时候也要等到serverCron执行过来的时候
  - 不会再发生持久化：redis有两个参数dirty（记录上一次bgsave之后的key的修改数，上边的在240s内例子就是0）和lastsave（上一次成功执行bgsave命令的时间），配置中的每一个save配置的修改数指的就是dirty，而每一个时间段就是以lastsave为起点计算的。
- 注释掉所有的save命令，RDB将不起作用
- rdbcompression yes：配置成这样是不是每一个字符串在存储到rdb文件中时，都要进行一次压缩操作？
  - 不是：设置为yes之后，只有当字符串的长度大于等于21个字节时，才会进行压缩
- rdbchecksum yes：这个校验和存储在哪里？为什么通过比对校验和可以判断文件是否损坏？
  - 校验和（check_sum）存储在RDB文件的最后八个字节中（详细的RDB文件结构，查看《Redis这基于实现》"第10章 RDB持久化"），简单的RDB文件结构如下：

    - RDB文件开头的前五个字节"REDIS"是判断一个文件是不是RDB文件的标准(类似于class文件中的"魔数")
    - 接下来的4个字节：RDB文件版本号（db_version）
    - databases(注意是复数)：这里存放各个库redisDb中存储的key-value信息（是整个数据持久化和恢复的核心）

    - EOF(1个字节)：RDB文件正文的结束
    - check_sum(8个字节)：检验和，该值是根据前边四部分值算出来的，在持久化的时候将该值算出来并写入rdb文件的末尾；在根据rdb文件恢复数据的时候，再根据rdb文件中的前边四部分值计算出一个校验和，然后与当前rdb文件中的check_sum（即后八个字节）的内容进行比对，如果一样，说明没损坏，如果不一样，说明前四部分有数据损坏（即该文件损坏）
- 在Redis服务器启动时，redis会自动检测是否有rdb文件（前提是没有aof的时候），如果有，则根据rdb文件恢复数据，此时在恢复数据完成之前，会阻塞客户端对redis的读写操作

使用RDB恢复数据：
自动的持久化数据存储到dump.rdb后。实际只要重启redis服务即可完成（启动redis的server时会从dump.rdb中先同步数据）

客户端使用命令进行持久化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。

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。

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

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

##always 每一个命令，都立即同步到aof文件中去（很安全，但是速度慢，因为每一个命令都会进行一次磁盘操作）IO开支较大。

##everysec每秒将数据写一次到aof文件，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 将写入工作交给操作系统，由操作系统来判断缓冲区大小，统一写到aof文件（速度快，但是同步频率低，容易丢数据）

##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”

# 在RDB持久化数据的时候，此时的aof操作是否停止，若为yes则停止

# 在停止的这段时间内，执行的命令会写入内存队列，等RDB持久化完成后，统一将这些命令写入aof文件

# 该参数的配置是考虑到RDB持久化执行的频率低，但是执行的时间长，而AOF执行的频率高，执行的时间短，
```

若同时执行两个子进程（RDB子进程、AOF子进程）效率会低（两个子进程都是磁盘读写）

但是若改为yes可能造成的后果是，由于RDB持久化执行时间长，在这段时间内有很多命令写入了内存队列，

最后导致队列放不下，这样AOF写入到AOF文件中的命令可能就少了很多

在恢复数据的时候，根据aof文件恢复就会丢很多数据

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”

AOF重写：把内存中的数据逆化成命令，然后将这些命令重新写入aof文件

重写的目的：假设在我们在内存中对同一个key进行了100次操作，最后该key的value是100，

那么在aof中就会存在100条命令日志，这样的话，有两个缺点：

1）AOF文件过大，占据硬盘空间 2）根据AOF文件恢复数据极慢（需要执行100条命令）

如果我们将内存中的该key逆化成"set key 100"，然后写入aof文件，

那么aof文件的大小会大幅度减少，而且根据aof文件恢复数据很快（只需要执行1条命令）

注意：下边两个约束都要满足的条件下，才会发生aof重写；

假设没有第二个，那么在aof的前期，只要稍微添加一些数据，就发生aof重写

当aof的增长的百分比是原来的100%（即是原来大小的2倍，例如原来是100m，下一次重写是当aof文件是200m的时候），AOF重写

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：

1) 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

- 如果既配置了RDB，又配置了AOF，则在进行数据持久化的时候，都会进行，但是在根据文件恢复数据的时候，以AOF文件为准，RDB文件作废
    - 需要注意：数据的恢复是阻塞操作（此间所到来的任何客户端读写请求都失效）
- bgsave和bgrewriteaof（后台aof重写）这两个命令不可以同时发生
    - 如果bgsave在执行，此间到来的bgrewriteaof在bgsave执行之后，再执行
    - 如果bgrewriteaof在执行，此间到来的bgsave丢弃
- RDB和AOF可以同时配置，但是最后还原数据库的时候是以aof文件来还原的