## 官网介绍：

Redis提供了不同的持久性选项：

* RDB（Redis DataBase）持久性以指定的时间间隔执行数据集的时间点快照。
* AOF（Append Only File）持久性记录服务器接收的每个写入操作，将在服务器启动时再次播放，重建原始数据集。使用与Redis协议本身相同的格式以仅追加方式记录命令。当Redis太大时，Redis能够重写日志背景。
* 如果您愿意，只要服务器正在运行，您就可以根据需要禁用持久性。
* 可以在同一实例中组合AOF和RDB。请注意，在这种情况下，当Redis重新启动时，AOF文件将用于重建原始数据集，因为它保证是最完整的。

最重要的是要理解RDB和AOF持久性之间的不同权衡。

## RDB（Redis DataBase）

### RDB的优势

* RDB是Redis数据的一个非常紧凑的单文件时间点表示。RDB文件非常适合备份。例如，您可能希望在最近24小时内每小时归档您的RDB文件，并且每天保存RDB快照30天。这使您可以在发生灾难时轻松恢复不同版本的数据集。
* RDB非常适合灾难恢复，可以将单个压缩文件传输到远端数据中心，也可以传输到Amazon S3（可能是加密的）。
* RDB最大限度地提高了Redis的性能，因为Redis父进程为了坚持不懈而需要做的唯一工作就是分配一个将完成所有其余工作的孩子。父实例永远不会执行磁盘I / O或类似操作。
* 与AOF相比，RDB允许使用大数据集更快地重启。

### RDB的缺点

* 如果您需要在Redis停止工作时（例如断电后）将数据丢失的可能性降至最低，则RDB并不好。您可以配置生成RDB的不同保存点（例如，在对数据集进行至少五分钟和100次写入之后，但您可以有多个保存点）。但是，您通常每五分钟或更长时间创建一个RDB快照，因此如果Redis因任何原因停止工作而没有正确关闭，您应该准备丢失最新的数据分钟。
* RDB经常需要fork（）才能使用子进程持久存储在磁盘上。如果数据集很大，Fork（）可能会非常耗时，如果数据集非常大且CPU性能不佳，可能会导致Redis停止服务客户端几毫秒甚至一秒钟。AOF也需要fork（），但你可以调整你想要重写日志的频率而不需要对耐久性进行任何权衡。

## AOF（Append Only File）

### AOF优势

* 使用AOF Redis更持久：您可以使用不同的fsync策略：根本不使用fsync，每秒执行fsync，每次查询都使用fsync。使用fsync的默认策略，每秒写入性能仍然很好（使用后台线程执行fsync，并且当没有fsync正在进行时，主线程将努力执行写入。）但是您只能丢失一秒的写入。
* AOF日志是仅附加日志，因此如果停电，则没有搜索，也没有损坏问题。即使由于某种原因（磁盘已满或其他原因）日志以半写命令结束，redis-check-aof工具也能够轻松修复它。
* 当Redis太大时，Redis能够在后台自动重写AOF。重写是完全安全的，因为当Redis继续附加到旧文件时，使用创建当前数据集所需的最小操作集生成一个全新的文件，并且一旦第二个文件准备就绪，Redis会切换两个并开始附加到新的那一个。
* AOF以易于理解和解析的格式一个接一个地包含所有操作的日志。您甚至可以轻松导出AOF文件。例如，即使您使用FLUSHALL命令刷新了所有错误，如果在此期间未执行重写日志，您仍然可以保存数据集，只需停止服务器，删除最新命令，然后重新启动Redis。

### AOF的缺点

* AOF文件通常比同一数据集的等效RDB文件大。
* 根据确切的fsync策略，AOF可能比RDB慢。一般来说，fsync设置为每秒性能仍然非常高，并且在fsync禁用的情况下，即使在高负载下也应该与RDB一样快。即使在写入负载很大的情况下，RDB仍能够提供有关最大延迟的更多保证。
* 在过去，我们遇到了特定命令中的罕见错误（例如，有一个涉及阻塞命令，如BRPOPLPUSH）导致生成的AOF在重新加载时不会重现完全相同的数据集。这个错误很少见，我们在测试套件中进行测试，自动创建随机复杂数据集并重新加载它们以检查一切是否正常，但RDB持久性几乎不可能出现这种错误。为了更清楚地说明这一点：Redis AOF逐步更新现有状态，如MySQL或MongoDB，而RDB快照一次又一次地创建所有内容，这在概念上更加健壮。但是 - 1）应该注意的是，每次通过Redis重写AOF时，都会从数据集中包含的实际数据开始重新创建，与总是附加的AOF文件（或者重写旧的AOF而不是读取内存中的数据）相比，对bug的抵抗力更强。2）我们从未向用户提供过关于在现实世界中检测到的AOF损坏的单一报告。

## 1.1RDB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，也就是行话讲的Snapshot快照，它恢复时是将快照文件直接读到内存里。

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，待持久化过程都结束了，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复，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的缺点是最后一次持久化后的数据可能丢失。

## 1.2Fork

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、程序计数器等）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。

## 1.3RDB保存的是dump.rdb文件

## 1.4配置位置

RDB默认的持久化策略是：

save 60 10000

save 300 10

save 900 1

###############################SNAPSHOTTING ###############################

#

# Save the DB on disk:

#

# save <seconds> <changes> 命令

#

# Will save the DB if both the given number of seconds and the given

# number of write operations against the DB occurred.

#

# In the example below the behaviour will be to save:

# after 900 sec (15 min) if at least 1 key changed

# after 300 sec (5 min) if at least 10 keys changed

# after 60 sec if at least 10000 keys changed

#

# Note: you can disable saving completely by commenting out all "save" lines.

#

# It is also possible to remove all the previously configured save

# points by adding a save directive with a single empty string argument

# like in the following example:

#

# save "" 禁用

save 900 1 900秒内改过1次

#save 300 10 300秒内改过10次

save 120 10

save 60 10000 60秒内改过10000次

# By default Redis will stop accepting writes if RDB snapshots are enabled

# (at least one save point) and the latest background save failed.

# This will make the user aware (in a hard way) that data is not persisting

# on disk properly, otherwise chances are that no one will notice and some

# disaster will happen.

#

# If the background saving process will start working again Redis will

# automatically allow writes again.

#

# However if you have setup your proper monitoring of the Redis server

# and persistence, you may want to disable this feature so that Redis will

# continue to work as usual even if there are problems with disk,

# permissions, and so forth.

stop-writes-on-bgsave-error yes 如果配置成no，表示你不在乎数据不一致或者有其它手段发现和控制。

# Compress string objects using LZF when dump .rdb databases?

# For default that's set to 'yes' as it's almost always a win.

# If you want to save some CPU in the saving child set it to 'no' but

# the dataset will likely be bigger if you have compressible values or keys.

rdbcompression yes 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，如果不想消耗CPU来进行压缩的话，可以设置为关闭此功能

# Since version 5 of RDB a CRC64 checksum is placed at the end of the file.

# This makes the format more resistant to corruption but there is a performance

# hit to pay (around 10%) when saving and loading RDB files, so you can disable it

# for maximum performances.

#

# RDB files created with checksum disabled have a checksum of zero that will

# tell the loading code to skip the check.

rdbchecksum 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。

# The filename where to dump the DB

dbfilename dump.rdb

# The working directory.

#

# The DB will be written inside this directory, with the filename specified

# above using the 'dbfilename' configuration directive.

#

# The Append Only File will also be created inside this directory.

#

# Note that you must specify a directory here, not a file name.

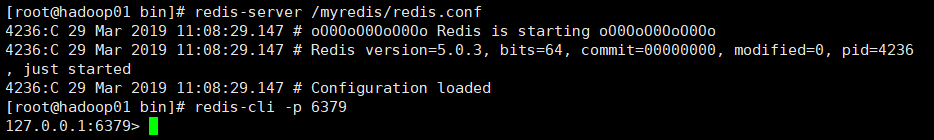
dir ./

|  |  |
| --- | --- |
| 查看redis是否启动的方法 | |
| ps -ef|grep redis |  |
| netstat -lntp | grep 6379 |  |
| lsof -i :6379 |  |

启动redis：

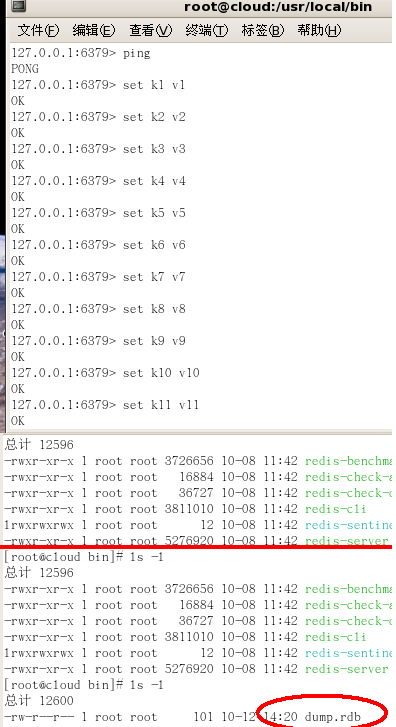
开启redis服务：redis-server /myredis/redis.conf；

开启redis客户端：redis-cli -p 6379；



## 1.5如何触发RDB快照

* 配置文件中的快照配置



冷拷贝后重新使用，可以cp dump.rdb dump\_new.rdb

* 命令save或者是bgsave

save：save时只保管，其它不管，全部阻塞

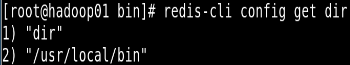
bgsave：Redis会在后台异步进行快照操作，快照同时还可以响应客户端请求。可以通过lastsave命令获取最后一次成功执行快照的时间。

* 执行flushall命令，也会产生dump.rdb文件，但里面是空的，无意义。

## 1.6如何恢复

将备份文件（dump.rdb）移动到redis安装目录并启动服务即可。

redis-cli config get dir 获取目录



## 1.7优势与劣势

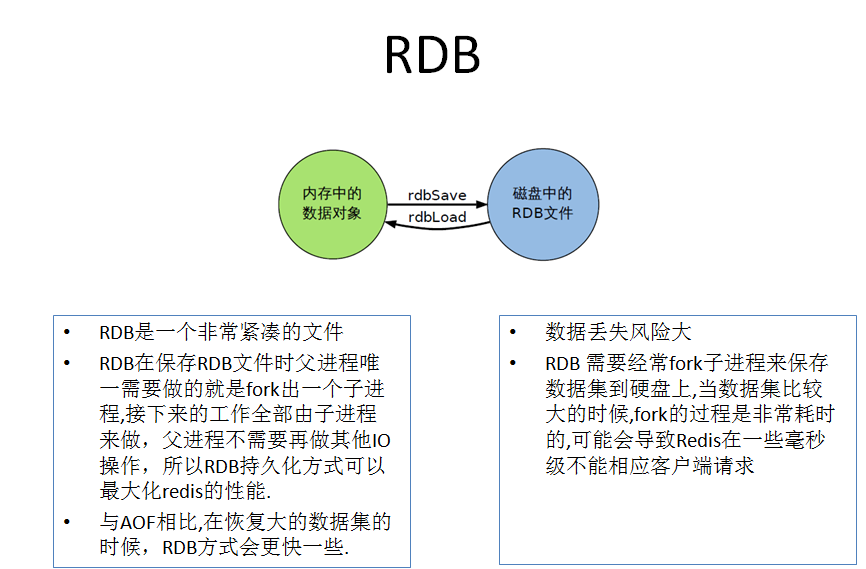
优势：适合大规模的数据恢复，对数据完整性和一致性要求不高。

劣势：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后的所有修改。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。

## 1.8如何停止

动态停止所有RDB规则的方法：redis-cli config set save “”

## 1.9RDB小总结



## 2.1AOF是什么

以日志形式来记录每个写操作，将Redis执行过的所有写指令记录下来（读操作不记录），只许追加文件但不可以改写文件，redis启动之初会读取该文件重新构建数据，换言之，redis重启的话就根据日志文件的内容将写指令从前到后执行一次以完成数据的恢复工作。

## 2.2AOF保存的是appendonly.aof文件

## 2.3配置位置



## 2.4AOF启动/修复/恢复

dump.rdb文件和appendonly.aof文件可以共存，启动的时候先找appendonly.aof，如果文件出错则启动失败。

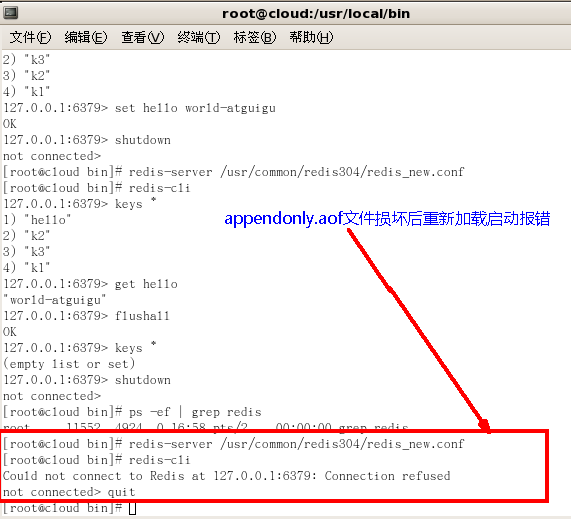
* 正常恢复
* 启动：修改默认的appendonly no，改为yes

将有数据的aof文件复制一份保存到对应目录（redis-cli config get dir）

* 恢复：重启redis然后重新加载
* 异常恢复
* 启动：修改默认的appendonly no，改为yes

备份被写坏的AOF文件

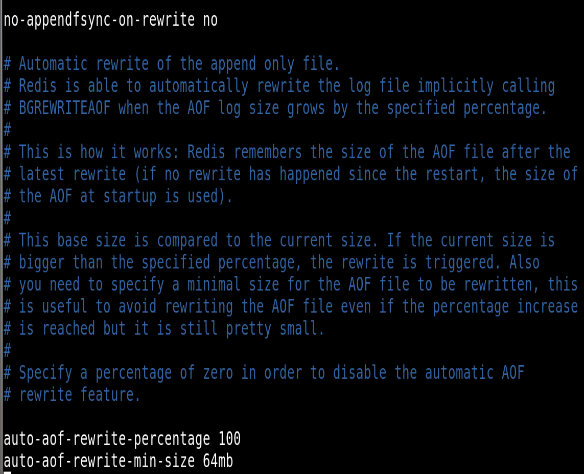
* 修复：redis-check-aof --fix appendonly.aof进行修复



* 恢复：重启redis然后重新加载

## 2.5Rewrite（重写）

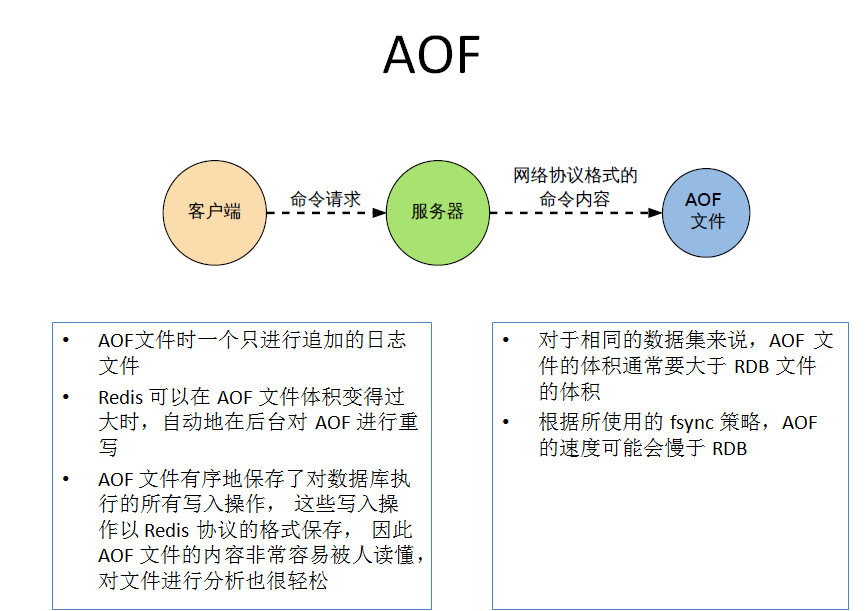
* 是什么：AOF采用文件追加方式，文件会越来越大，为避免出现此种情况，新增了重写机制，当AOF文件的大小超过所设定的阈值时，Redis就会启动AOF文件的内容压缩，只保留可以恢复数据的最小指令集，可以使用命令bgrewriteaof
* 重写原理：AOF文件持续增长而过大时，会fork出一条新进程来将文件重写（也是先写临时文件最后再rename），遍历新进程的内存中数据，每条记录有一条的Set语句。重写aof文件的操作，并没有读取旧的aof文件，而是将整个内存中的数据库内容用命令的方式重写了一个新的aof文件，这点和快照有点类似。
* 触发机制：Redis会记录上次重写时的AOF大小，默认配置是当AOF文件大小是上次rewrite后大小的一倍且文件大于64M时触发。（大公司日志文件大小3G起）



## 2.6优势与劣势

* 优势：
* 每修改同步：appendfsync always 同步持久化 每次发生数据变更会被立即记录到磁盘 性能较差但数据完整性比较好。
* 每秒同步：appendfsync everysec 默认推荐 异步操作，每秒记录 如果一秒内宕机，有数据丢失。
* 不同步：appendfsync no 从不同步
* 劣势：
* 相同数据采集的数据而言aof文件要远大于rdb文件，恢复速度慢于rdb；
* aof运行效率要慢于rdb，每秒同步策略效率较好，不同步效率和rdb相同

## 2.7AOF小总结



## 3.1RDB和AOF总结（Which one）

* 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的数据，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，是的AOF文件的体积不至于过大。
* 只做缓存：如果你只希望你的数据在服务器运行的时候存在，你也可以不使用任何持久化方式。
* 同时开启两种持久化方式：
  + 在这种情况下，当redis重启的时候就会优先载入AOF文件来恢复原始的数据，因为在通常情况下AOF文件保存的数据集要比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。那要不要只使用AOF呢？建议不要，因为RDB更适合用于备份数据库（AOF在不断变化不好备份），快速重启，而且不会有AOF可能潜在的bug，留着作为一个万一的手段。
* 性能建议：

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 900 1这条规则。

如果开了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文件就可以了。代价一是带来了持续的IO，二是AOF重写的最后将rewrite过程中产生的新数据写到新文件造成的阻塞几乎是不可避免的。只要硬盘许可，应该尽量减少AOF重写的频率，AOF重写的基础大小默认值64M太小了，可以设到5G以上。默认超过原大小100%时重写可以改到适当的数值。

如果不开AOF，仅靠Mater-Slave Replication（主从复制）实现高可用性也可以。能省掉一大笔IO也减少了重写时带来的系统波动。代价是如果Master/Slave同时倒掉，会丢失十几分钟的数据，启动脚本也要比较两个Master/Slave中的RDB文件，载入较新的那个，新浪微博就选用了这种架构。