# Redis持久化简介

Redis数据库的读写速度快，在很大程度上是因为Redis将所有的数据都存储在了内存中；然而，当Redis进程退出或重启之后，所有存储在内存中的数据就会丢失，为了解决这样的问题，我们希望Redis采用某种方式将数据从内存保存到硬盘中，来保证数据的不丢失，这个过程就是Redis的持久化。

Redis支持两种持久化方式：AOF持久化、RDB持久化。AOF持久化方式会将每次执行的命令及时保存到硬盘中；RDB持久化方式会根据指定的规则“定时”将数据保存到硬盘中。

# AOF持久化

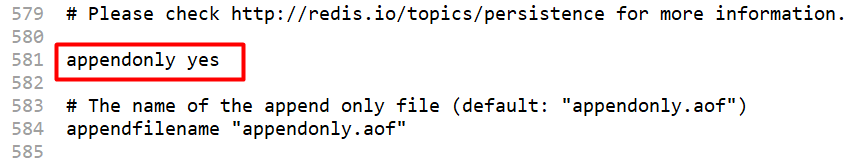
## AOF持久化介绍

1. AOF持久化方式会将服务器执行的所有写操作命令保存到AOF文件中，在服务器启动时，通过加载并执行AOF文件中保存的命令，来恢复数据。
2. Redis会以Redis协议格式来保存AOF文件中的所有命令，而不是以明文的形式来保存，所以即使打开AOF文件，也看不懂是些什么命令。
3. AOF文件还可以被重写（Rewrite），使得AOF文件的体积得到优化，且AOF文件的体积不会大于其保存数据的实际大小。

当使用Redis来存储一些需要长久保存的数据时，一般需要打开AOF持久化。

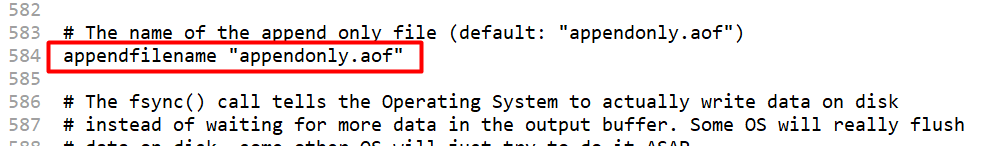
## AOF持久化的配置

1. 在默认情况下，AOF持久化是关闭着的；如果需要采用AOF持久化方式来保存数据，就要开启AOF持久化。可以通过修改配置文件redis.conf中的appendonly参数来开启。

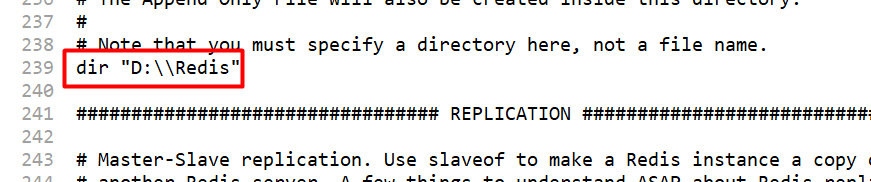


1. 开启AOF持久化后，服务器每执行一条写命令，Redis就会将该命令写入AOF文件中。

AOF文件的默认名称是appendonly.aof，可以通过appendfilename参数来修改AOF文件的名称，如下：



AOF文件默认存放在Redis的安装目录中，可以通过dir参数设置存放路径，如下：



与AOF持久化相关的配置参数如下：

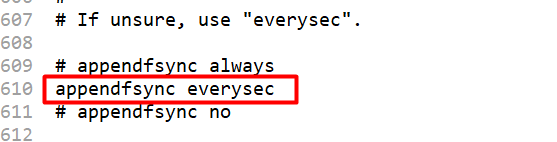
1. appendonly：是否开启AOF持久化，默认为no，表示不开启，设置为yes表示开启；
2. appendfilename：AOF文件名，默认为appendonly.aof，可以修改它；
3. dir：AOF文件和RDB文件存放的位置，默认为Redis的安装目录，可以修改它；
4. appendfsync：fsync持久化策略，默认为everysec；
5. ao-appendfsync-on-rewrite：在重写AOF文件过程中，是否禁止fsync，默认为no，如果修改为yes，则可以减轻重写AOF文件时CPU和硬盘的负载，但同时可能会丢失重写AOF文件过程中的数据；所以需要在负载与安全性之间进行平衡。
6. auto-aof-rewrite-percentage：指定Redis重写AOF文件的条件，默认为100，它会对比上次生成的AOF文件大小。如果当前AOF文件的增长量大于上次AOF文件的100%，就会触发A重写操作；如果值为0，则不会触发重写操作。
7. auto-aof-rewrite-min-size：指定触发重写操作的AOF文件的大小，默认为64MB。如果当前AOF文件的大小低于该值，此时就算AOF文件的增长比例达到了auto-aof-rewrite-percentage选项设置的值，也不会触发重写操作。只有同时满足这个两个参数设置的条件，才会触发AOF重写操作。
8. auto-load-truncated：当AOF文件结尾遭到损坏时Redis在启动时是否仍然加载AOF文件。默认为yes。

## AOF持久化的实现过程

开启AOF持久化之后，Redis服务器每执行一条写命令，AOF文件都会记录这条写命令，所以AOF文件持久化不需要手动触发，只要在配置文件中配置了相应的参数即可自动持久化。AOF持久化的实现过程如下：

1. 命令追加：Redis服务器每执行一条写命令，这条命令都会被追加到缓冲区aof\_buf中。但并没有直接将命令写入AOF文件中。
2. 写入AOF文件和同步AOF文件：根据appendfsync参数设置的同步策略，将缓冲区aof\_buf中的数据内容同步到硬盘中。
3. 当appendfsync参数值为always时，服务器每写入一条命令，都会将缓冲区aof\_buf中的所有内容写入AOF文件中，并同步AOF文件。但这个过程加大了硬盘I/O的负载，从而降低了Redis的性能。所以使用always策略的效率比较低，但它是最安全的。
4. 当appendfsync参数值为no时，会将缓冲区aof\_buf中的所有内容写入AOF文件中，但不会同步AOF文件，至于什么时候同步则交给操作系统来决定，通常同步周期为30秒。在使用no策略时，AOF文件的同步不可控，且缓冲区的内容会越来越多，一旦发生故障，将会丢失大量数据。因为不用执行AOF同步操作，所以AOF写入数据的速度是最快的，效率也是最高的。
5. 当appendfsync参数值为everysex时，会将缓冲区aof\_buf中的所有内容写入AOF文件中。而AOF文件的同步操作则由一个专门的文件同步线程负责，每隔一秒执行一次同步操作。everysex策略时前面两种策略的折中，即满足了效率又考虑了安全性，推荐使用。

在Redis配置文件中appendfsync参数如下：



## AOF文件重写

**认识文件重写**

1. 为了解决AOF文件体积过大的问题，Redis提供了AOF文件重写功能。

如A客户端执行了两条命令：

set name zhangsan

set name lisi

AOF文件重写就会将其压缩为一条语句set name lisi，用最少的命令来实现压缩AOF文件的目的。

1. AOF文件重写之所以可以压缩AOF文件，减少AOF文件的体积，其原因如下：
2. AOF文件重写会丢弃过期的数据，过期的数据是不会被写入AOF文件中的。
3. AOF文件重写会丢弃无效的命令，无效的命令是不会被写入AOF文件中的。无效的命令包括重复设置某个键值对值，删除某些数据时的命令。
4. AOF文件重写可以将多条命令合并为一条命令，然后写入AOF文件中。

**AOF文件重写的触发方式**

AOF文件的重写的触发有两种方式：手动触发和自动触发。

1. 手动触发：客户端通过执行bgrewriteaof命令触发AOF文件重写。
2. 自动触发：自动触发是通过设置Redis配置文件中auto-aof-rewrite-percentage和auto-aof-rewrite-min-size参数的值，以及aof\_current\_size和aof\_base\_size状态来确定何时触发的。

## AOF文件处理

开启了AOF持久化，那么Redis服务器启动的时候，会首先加载AOF文件中的命令来恢复数据。这个过程如下：

1. 创建一个伪客户端，用于执行AOF文件中的写命令。
2. 读取AOF文件中的数据，分析并提取AOF文件中所保存的一条写命令。
3. 使用伪客户端执行被读取出的命令。
4. 重复执行（2）和（3），直到将AOF文件中的所有命令读取完毕，并成功执行为止。

这样，数据就得到了恢复。如果Redis服务器启动加载AOF文件时，发现AOF文件被损坏了，那么服务器会拒绝加载这个AOF文件，以此来确保数据的一致性。而AOF文件被损坏的原因可能是程序正在对AOF文件进行写入与同步时，服务器出现停机故障。我们可以通过以下方法来修改AOF文件。

1. 及时备份现有的AOF文件；
2. 利用Redis自带的redis-check-aof程序，对原来的AOF文件进行修复，命令如下：

**redis-check-aof -fix**

使用diff -u来对比元素AOF文件和修复后的AOF文件，找出这两个文件的不同之处。

修复AOF文件之后，重启Redis服务器重新加载，进行数据恢复。

## AOF持久化的优劣

**AOF持久化优点如下：**

1. Redis持久化更长：通过设置不同的fsync策略来达到更长的持久化。
2. 兼容性较好：AOF文件是一个日志文件，它的作用是记录服务器执行的所有写命令。当文件因为某条写命令写入失败时，可以使用redis-check-aof进行修复，然后继续使用。
3. 支持后台重写：当AOF文件的体积过大时，在后台可以自动地对AOF文件进行重写，以减少AOF的体积。
4. AOF文件易于读取和加载：AOF文件保存了数据库的所有写命令，这些命令采用Redis协议格式追加到AOF文件中，因此非常容器读取和加载。

**AOF持久化缺点如下：**

1. AOF文件的体积会随着时间的推移逐渐变大，导致在加载时速度会比较慢，进行影响数据库的恢复速度，性能快速下降。
2. 根据所使用的fsync策略，使用AOF文件恢复数据的速度可能会慢于使用RDB文件恢复数据的速度。
3. 因为AOF文件的个别命令，可能会导致在加载时失败，从而无法进行数据恢复。

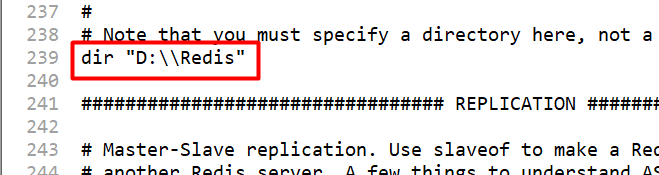
# RDB持久化

## RDB持久化介绍

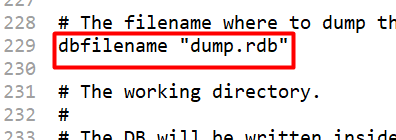
1. RDB持久化与AOF持久化的目的相同，都是Redis为了避免数据丢失的一种持久化机制。
2. RDB持久化是通过RDB文件来完成的，Redis服务器启动时，会加载RDB文件来恢复数据。（如果配置了AOF持久化，则会优先使用AOF文件恢复数据）

## RDB持久化的配置

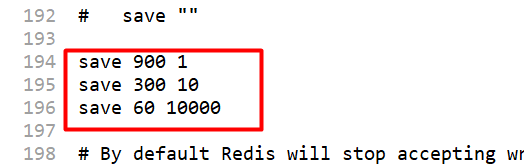
1. 在默认情况下，RDB文件存放在Redis安装目录中，可以通过Redis配置文件中的dir参数修改。如下：



1. 在默认情况下，RDB文件名为dump.rdb，可以通过Redis配置文件中的dbfilename参数修改。如下：

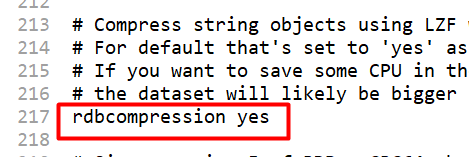


1. save m n表示在m秒内被修改的键的个数大于n，会触发dbsave命令，如果没有设置该配置，则表示RDB持久化是关闭着的。如下：

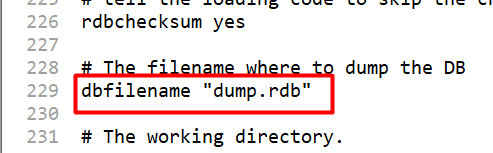


可以配置多个。

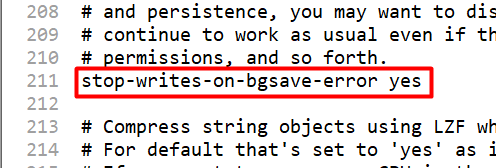
1. rdbcompression参数表示是否开启压缩RDB文件，默认为yes，表示开启；不开启则设置为no。如下：



1. rdbchecksum参数表示是否检查RDB文件被损坏。在服务器进行RDB文件的写入与读取时会用到它。默认为yes。如果设置为no，则表示不开启，在服务器对RDB文件写入与读取时，可以提升性能，但无法确定RDB文件是否被损坏。如下：



1. stop-writes-on-bgsave-error参数表示当执行bgsave命令出现错误时，Redis是否终止执行写命令。参数值默认为yes，表示当硬盘出现问题时，服务器可以及时发现，及时避免大量数据丢失；当设置为no时，表示服务器执行bgsave命令发送错误时，服务器也会继续执行命令；如果对Redis服务器的系统设置了监控，建议将该参数设置为no。如下：



## RDB文件的创建

RDB持久化方式是通过加载RDB文件来实现的。那么RDB文件是如何生成的呢？

**生成RDB文件可以通过save命令和bgsave命令完成。**

1. save

save命令用于创建RDB文件，以同步的方式。命令如下：

**save**

执行save命令后，会阻塞Redis服务器进程，此时服务器将不能继续执行其他命令，必须等待RDB文件创建完成后才可以执行。

1. bgsave

bgsave用于创建RDB文件，以异步的方式。命令如下：

**bgsave**

1. 执行bgsave命令后，Redis会产生一个子进程，由子进程将数据保存到硬盘中，完成RDB文件的创建，而服务器可以继续执行其他命令。
2. 执行bgsave命令返回一个”background saving started”，但我们并不知道RDB文件有没有创建完成。此时我们可以通过lastsave命令来查看最近一次保存数据的时间，来判断数据RDB有没有创建完成。
3. 在RDB持久化的配置中，存在一个save参数，通过配置该参数的值，可以自动执行bgsave命令。

## RDB持久化的优劣

RDB持久化具有以下优点：

1. RDB文件是一个经过压缩的二进制文件，文件紧凑，体积较小，适用于做数据备份；
2. RDB持久化适用于灾难性恢复，且恢复数据时的速度要快于AOF持久哈；
3. Redis持久化可以很大程度上提升性能。父进程在执行bgsave命令时会开启一个子进程，将所有与保存相关的功能交给子进程处理，而父进程可以继续处理其他相关操作。

RDB持久化的具有以下缺点：

1. 在服务器出现故障时，如果没有触发RDB文件的创建，那么它很可能会丢失大量数据；
2. 当数据量非常大时，使用gbsave命令开启一个子进程来完成保存操作时，会比较耗时，将占用太多的CPU时间，从而影响服务器的性能；
3. RDB文件存在兼容性问题，老版本的Redis不支持新版本的RDB文件。

# AOF持久化与RDB持久化的抉择

为了避免数据丢失所引发的强大问题，建议同时使用AOF持久化与RDB持久化，以保证数据的持久性与安全性。

如果非要二选一，推荐使用RDB。

1. 使用RDB持久化，便于进行数据的备份，同时也能提升服务器的性能，并且RDB恢复数据的速度要快于AOF恢复数据的速度

缺点：必须承受服务器出现故障，则会丢失部分数据的风险。

1. 使用AOF持久化，生成的AOF文件体积过大，恢复数据时速度会比较慢，会严重影响服务器的性能。

（个人认为：AOF用于保存数据，RDB用于备份数据）