链滴

redis 持久化（RDB 与 AOF）的方式比较

作者：sirwsl

原文链接：https://ld246.com/article/1592320563697

来源网站：链滴

持久化：

将内存中的数据在一定时间内保存一次到硬盘，当出现数据丢失（死机等），再次将硬盘中的数据进读取恢复回来

一、RDB方式

1.文件保存的方式:

- save保存
- bgsave(后台执行保存,save优化)
- 自动执行

2.save保存方式

(1)配置文件:

```
dbfilename dumo.rdb
#设置本地数据库文件名，默认为dump.rdb (eg:dump-6379.rdb)
dir
#设置rdb文件的存储路径/一般为data目录
rdbcompression yes
#是否采用压缩数据。若采用为LZF压缩（开启可以使得存储文件得到压缩，但是会增加cpu运行时间如果关闭文件变得巨大，但可以节约cpu运行时间）
rdbchecksum yes
#是否在读写过程中进行RDB文件格式校验，默认开启（如果关闭，可能会有数据损坏的风险，但是以提高读写性能<约10%的时间消耗>）
```

(2)持久化后的文件:

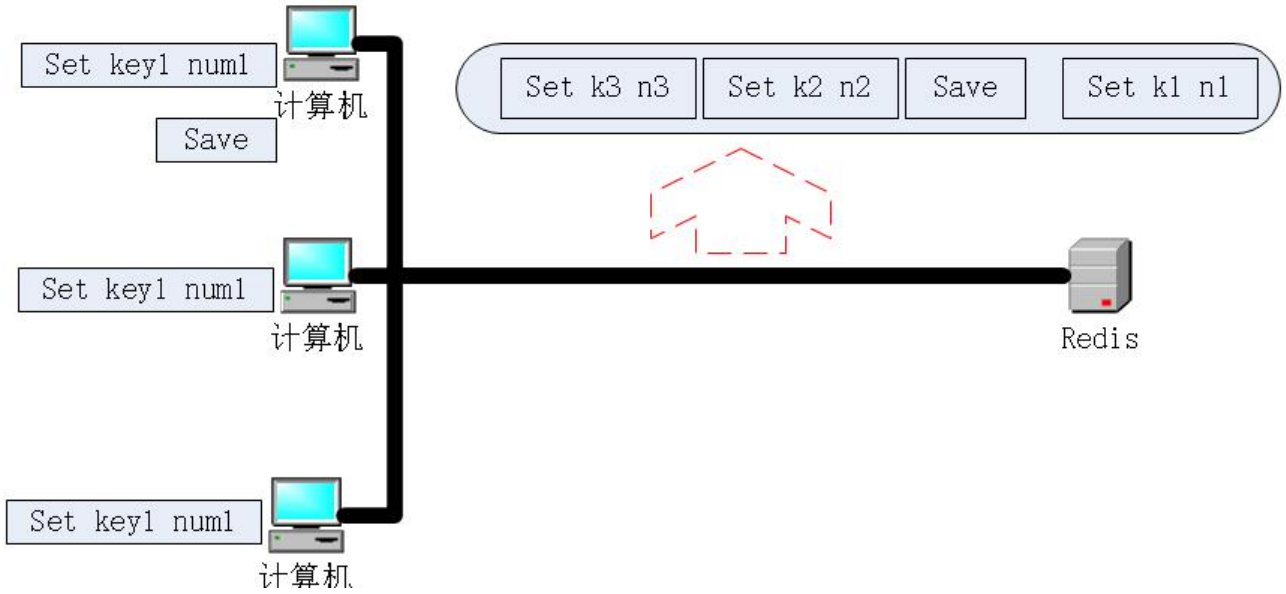
xxx.rdb

(3)save工作原理:

redis单线程任务执行队列,当多个计算机同一时间发送请求时,所有的指令全部进入队列,然后逐一被执行

风险性: 线上环境使用sava可能造成长时间的阻塞

优化:采用bgsave（后台执行保存）



2.bgsave保存方式:

bgsave对save的阻塞问题进行相应的优化，redis内部所涉及到的RDB操作都是采用bgsave的方式进，save被弃用

(1)配置文件:

```
stop-writes-on-bgsave-error yes
#在存储的过程中如果发生错误就停止保存，默认状态都是为开启
```

(2)工作原理:

当客户端发送一段指令之后,redis接收后返回massage,同时调用fork函数生成子进程,然后进行rdb文创建后返回.

3.save与bgsave比较:

| 方式 | save | bgsave |

col1	col2	col3
读写	同步	异步
阻塞	会	不会
额外内存	不需要	需要

原文链接：redis 持久化（RDB 与 AOF）的方式比较

启动进程	不启动	启动

4.自动执行:

redis会根据你所设置的save指令进行每个多少时间达到所监控的数量就进行一次文件的保存

(1)配置文件:

```
save second change
#在second时间内对key进行变化量监控，达到指定数量就进行持久化,根据自己的需求设置
save 900 1
save 300 10
save 60 100
```

5.RDB注意事项注意:

- save保存的过程要根据业务进行设置，频度过高过低都可能造成灾难性
- save对于second与changes的设置通常是互补的对应关系，尽量不出现包含

自动启动中save配置启动后执行的都是bgsave操作

6.RBF优点:

- RDB是一个紧凑的二进制压缩文件，存储效率高
- 内部存储是redis在某个时间的数据快照，适用于数据备份
- RDB数据恢复的速度比AOF快很多

应用：（容灾设计：服务器每过一段时间就进行bgsave备份，将RDB文件拷贝至远程服务器，用于难恢复）

7:RBF缺点:

- 无法做到实时持久化，可能存在数据丢失
- bgsave每次的执行都需要fork操作进行子进程创建，要牺牲一部分性能
- Rdis有许多RDB文件格式，版本未进行统一，各个版本之间可能存在无法兼容现象。
- 基于快照思想、数据量巨大时效率低下。

二.AOF方式:

AOF:以独立日志的方式记录每次写的命令，在重启的时候执行AOF中的命令达到恢复数据的目的

AOF为Redis的持久化主流方式，解决了数据的时效性

1.生成文件:

xxx.aof

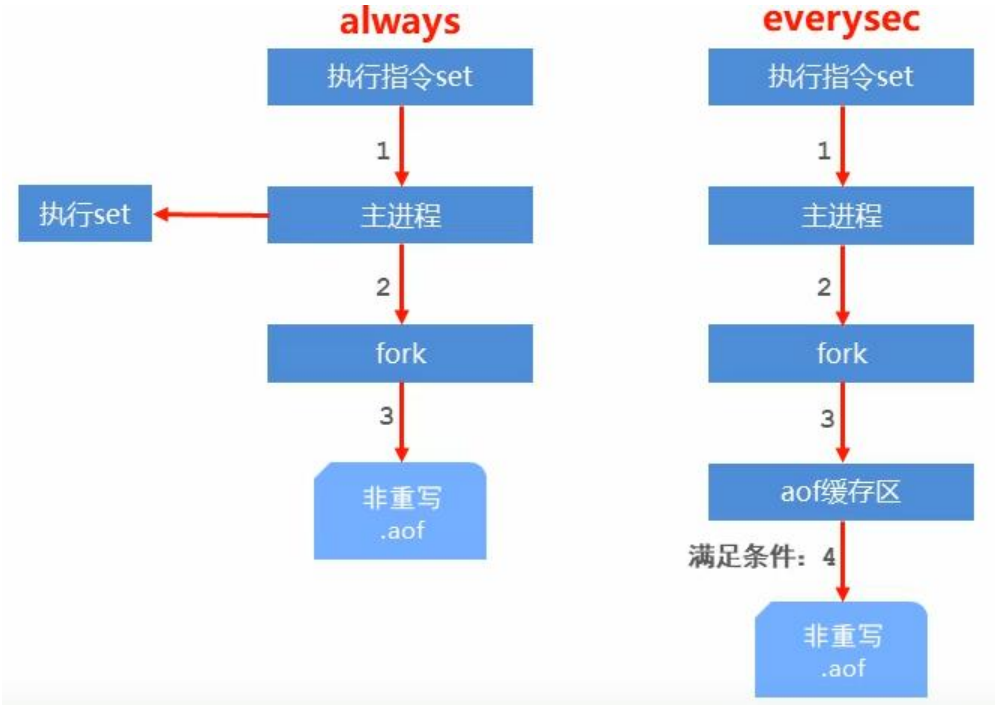
2.AOF写数据的三种方式:always（每次）、everysec（每秒）、n

（系统控制）

always：每次写入操作的同时将指令同步到AOF中，数据O误差，但会是性能降低

everysec：每秒将缓冲区指令同步到AOF中，数据准确性较高（非100%）性能较高

no:由操作系统控制每次同步到AOF中的周期，整体过程不可控



3.配置文件:

```
appendonly yes
#是否开启AOF持久化功能，默认不开启
appendfsync always|everysec|no
#配置AOF的保存数据频率
appendfilename filename
#AOF的持久化文件名。默认为：appendonly.aof(eg:appendonly-6379.aof)
dir
#AOF持久化保存文件的位置
```

4.AOF重写机制:

● 不断写入指令到AOF，文件会变得越来越大，为了解决这个问题，Redis引入了AOF重写机制进行件压缩。

● **AOF重写**：将Redis进程内的数据转化为命令同步更新到AOF文件的过程。将单说就是对同一个数的若干指令执行结果转化为最终结果用对应指令进行记录

5.作用:

● 降低磁盘的占用量，提高磁盘的利用率

● 提高持久化效率，降低持久化的时间，提高IO性能

● 降低数据恢复的用时，提高效率。

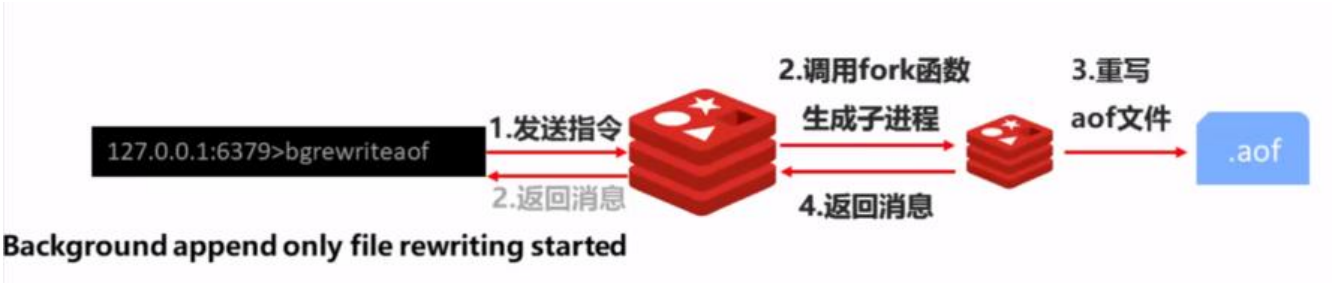
原文链接：redis 持久化（RDB 与 AOF）的方式比较

6.AOF重写规则:

- 进程内已经超时的数据不写入文件
- 对数据没有影响的以及无效的指令不写入文件
- 对同一数据的多条写入指令合并为一条指令

7.手动重写:

手动执行指令:bgrewiteaof,之后执行如下图(与RDB的bgsave过程一样,生成文件不同)



8.自动重写(略):

由redis自己判断完成重写.需要增加配置字段:

auto-aof-rewrite-min-size size

auto-aof-rewrite-percentage percentage

触发条件:

aof_current_size(当前大小)>auto-aof-rewrite-min-size

aof_current_size（当前尺寸）- aof_base_size/aof_base_size>=auto-aof-rewrite-percentage4

三、AOF与RDB对比:

持久化方式	RDB	AOF
占用存储空间	小（数据级：压缩）	大（指令级：重写）
存储速度	慢	快
恢复速度	快	慢
数据安全性	会丢失数据	依据策略决定
资源消耗	高/重量级	低/轻量级
启动优先级	低	高

原文链接：redis 持久化（RDB 与 AOF）的方式比较