**RDB**

RDB持久化是把当前进程数据生成快照保存到硬盘的过程，触发RDB持久化过程分为**手动触发**和**自动触发**。

**手动触发**

save命令和bgsave命令

save命令：阻塞当前Redis服务器，直到RDB过程完成为止，对于内存比较大的实例会造成长时间阻塞，线上环境不建议使用。

bgsave命令：Redis进程执行fork操作创建子进程，RDB持久化过程由子进程负责，完成后自动结束。阻塞只发生在fork阶段，一般时间很短。

**自动触发**

1.使用save相关配置，如“save m n”。表示m秒内数据集存在n次修改时，自动触发bgsave。

2.如果从节点执行全量复制操作，主节点自动执行bgsave生成RDB文件并发送给从节点。

3.执行debug reload命令重新加载Redis时，也会自动触发save操作。

4.默认情况下执行shutdown命令时，如果没有开启AOF持久化功能则自动执行bgsave

**bgsave的运作流程：**

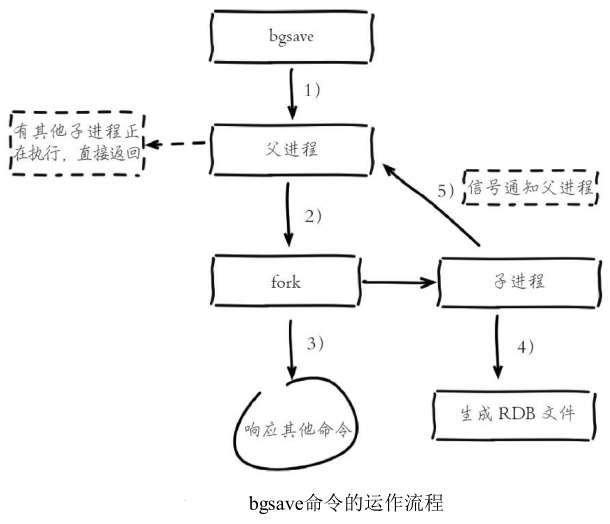
1、执行bgsave命令，Redis父进程判断当前是否存在正在执行的子进程，如RDB/AOF子进程，如果存在bgsave命令直接返回。

2、父进程执行fork操作创建子进程，fork操作过程中父进程会阻塞，通过info stats命令查看latest\_fork\_usec选项，可以获取最近一个fork操作的耗时，单位为微秒。

3、父进程fork完成后，bgsave命令返回“Background saving started”信息并不再阻塞父进程，可以继续响应其他命令。

4、子进程创建RDB文件，根据父进程内存生成临时快照文件，完成后对原有文件进行原子替换。执行lastsave命令可以获取最后一次生成RDB的时间，对应info统计的rdb\_last\_save\_time选项。

5、进程发送信号给父进程表示完成，父进程更新统计信息，具体见info Persistence下的 rdb\_\* 相关选项。



**RDB文件的处理**

保存：RDB文件保存在dir配置指定的目录下，文件名通过dbfilename配置指定。可以通过执行config set dir{newDir}和config set dbfilename{newFileName}运行期动态执行，当下次运行时RDB文件会保存到新目录。

压缩：Redis默认采用LZF算法对生成的RDB文件做压缩处理，压缩后的文件远远小于内存大小，默认开启，可以通过参数config set rdbcompression{yes|no}动态修改。

**RDB总结**

**优点：**

1.RDB是一个紧凑压缩的二进制文件，代表Redis在某个时间点上的数据

快照。非常适用于备份，全量复制等场景。

2.Redis加载RDB恢复数据远远快于AOF的方式。

**缺点：**

1.没办法做到实时持久化/秒级持久化，bgsave每次运行都要执行fork操作创建子进程，属于重量级操作，频繁执行成本过高。

2.RDB文件使用特定二进制格式保存，Redis版本演进过程中有多个格式的RDB版本，存在老版本Redis服务无法兼容新版RDB格式的问题。

**AOF**

AOF（append only file）持久化：

以独立日志的方式记录每次写命令，重启时再重新执行AOF文件中的命令达到恢复数据的目的。

作用： 解决了数据持久化的实时性。

**使用AOF**

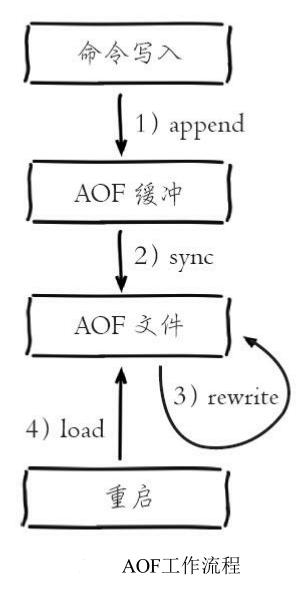
开启AOF功能需要设置配置：appendonly yes，默认不开启。

AOF文件名通过appendfilename配置设置，默认文件名是appendonly.aof。

保存路径同RDB持久化方式一致，通过dir配置指定。

AOF的工作流程操作：

命令写入(append) **->** 文件同步(sync) **->** 文件重写(rewrite) **->** 重启加载(load)



1、所有的写入命令会追加到aof\_buf（缓冲区）中。

2、AOF缓冲区根据对应的策略向硬盘做同步操作。

3、随着AOF文件越来越大，需要定期对AOF文件进行重写，达到压缩的目的。

4、当Redis服务器重启时，可以加载AOF文件进行数据恢复。

关于AOF的两个疑惑：

**a、**AOF直接采用文本协议格式的理由如下：

文本协议具有很好的兼容性。

开启AOF后，所有写入命令都包含追加操作，直接采用协议格式，避免了二次处理开销。

文本协议具有可读性，方便直接修改和处理。

**b、**AOF为什么把命令追加到aof\_buf中？

Redis使用单线程响应命令，如果每次写AOF文件命令都直接追加到硬盘，那么性能完全取决于当前硬盘负载。

先写入缓冲区aof\_buf中，还有另一个好处，Redis可以提供多种缓冲区同步硬盘的策略，在性能和安全性方面做出平衡。

**文件同步**

Redis提供了多种AOF缓冲区同步文件策略，由参数**appendfsync**控制。

**appendfsync可配置的值**

**always**

**作用**：命令写人aof\_ buf后调用系统fsync操作同步到AOE文件，fsync完成后线程返回。

**说明**：命令写人aof\_ buf后调用系统write操作，write完成后线程返回。fsync 同步文件操作由专

门线程每秒调用一次。

**no**

**作用**：命令写人aof\_ buf后调用系统write操作，不对AOE文件做fsync同步，同步硬盘操作由操作

系统负责，通常同步周期最长30秒。

**说明**：操作系统每次同步AOF文件的周期不可控，而且会加大每次同步硬盘的数据量，虽然提升

了性能，但数据安全性无法保证。

**everysec**

**作用**：命令写人aof\_ buf后调用系统write操作，write完成后线程返回。fsync 同步文件操作由专门线程每秒调用一次。

**说明**：是建议的同步策略，也是默认配置，做到兼顾性能和数据安全性。理论上只有在系统突然宕机的情况下丢失1秒的数据。

**重写机制**

AOF文件重写是把Redis进程内的数据转化为写命令同步到新AOF文件的过程。是为了解决AOF文件会越来越大的问题。

重写后的AOF文件为什么可以变小？有如下原因：

1、进程内已经超时的数据不再写入文件。

2、旧的AOF文件含有无效命令，重写使用进程内数据直接生成，这样新的AOF文件只保

留最终数据的写入命令。

3、多条写命令可以合并为一个，为了防止单条命令过大造成客户端缓冲区溢出，对于list、set、hash、zset等类型操作，以64个元素为界拆分为多条。

AOF重写降低了件占用空间，更小的AOF文件可以更快地被Redis加载。

AOF重写过程可以手动触发和自动触发：

**手动触发**：直接调用bgrewriteaof命令。

**自动触发**：根据auto-aof-rewrite-min-size和auto-aof-rewrite-percentage参数确定自动触发时机。

auto-aof-rewrite-min-size：表示运行AOF重写时文件最小体积，默认为64MB。

auto-aof-rewrite-percentage：代表当前AOF文件空间（aof\_current\_size）和上一次重写后AOF文件空间（aof\_base\_size）的比值。

自动触发时机=aof\_current\_size>auto-aof-rewrite-min-size&&（aof\_current\_size-aof\_base\_size）/aof\_base\_size>=auto-aof-rewrite-percentage