Redis对外提供数据访问服务时候，使用的是常住内存的数据，为了在Redis Server重启之后数据可以得到回复，Redis具备将数据持久化到硬盘中的能力

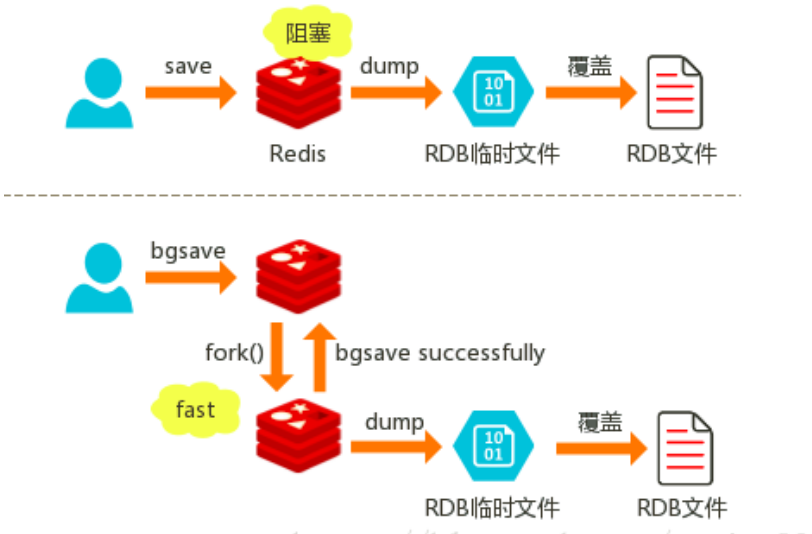
1.全量模式持久化(RDB)

基于全量的持久化就是在某个时刻，将Redis的所有数据持久化到硬盘中，形成一个快照以二进制的方式保存到磁盘中。当Redis 重启时，通过加载最近一个快照数据，可以将Redis 恢复至最近一次持久化状态上。

1.1 写入

Redis的全量写入包含两种方式：save和bgsave，两种处理逻辑如下

- save是有客户端显示触发的，也可以在Redis shutdown时触发。Save本身是单线程串行的方式执行，因此当数据量大时，有可能会发生Redis Server长时间卡顿的问题。但是其备份期间不会有其他命令执行，因此备份在这一刻是一致性的。
- bgsave可以有客户端显示触发，也可以用过配置定时任务触发，也可以在master-slave分布式结构下有slave节点触发。bgsave命令在执行的时候会fork一个子进行，子进程提交完成之后，会立即给客户端返回相应，备份操作在后台异步执行，此期间不会影响Redis的正常相应



bgsave相对于Save来说，其优势是异步执行，不影响后续的命令执行。但是Fork子进程时，涉及父进程的内存复制，此时会增加服务器的内存开销。当内存开销高到使用虚拟内存时，bgsave的Fork子进程会阻塞运行，可能会造成秒级的不可用。因此使用bgsave需要保证服务器空闲内存足够。

命令	save	bgsave
IO类型	同步	异步
是否阻塞	阻塞	非阻塞（在fork是阻塞）
复杂度	O(n)	O(n)
优点	不会消耗额外内存	不阻塞客户端命令
缺点	阻塞客户端命令	需要Fork子进程，内存开销大

1.2 载入

当 Redis 服务器启动时， rdbLoad 函数就会被执行， 它读取 RDB 文件， 并将文件中的数据库数据载入到内存中。

调用 rdbLoad 函数载入 RDB 文件时，不能进行任何和数据库相关的操作，在载入期间， 服务器每载入 1000 个键就处理一次所有已到达的请求， 不过只有 PUBLISH 、 SUBSCRIBE 、 PSUBSCRIBE 、 UNSUBSCRIBE 、 PUNSUBSCRIBE 五个命令的请求会被正确地处理， 其他命令一律返回错误。 等到载入完成之后， 服务器才会开始正常处理所有命令， 服务器才会开始正常处理所有命令不过订阅与发布方面的命令可以正常执行，因为它们和数据库不相关联。

1.3 数据结构

前面介绍了保存和读取 RDB 文件的两个函数，现在，是时候介绍 RDB 文件本身了。

一个 RDB 文件可以分为以下几个部分：

```
+-------+-------------+-----------+----------------+-----+-----------+
| REDIS | RDB-VERSION | SELECT-DB | KEY-VALUE-PAIRS | EOF | CHECK-SUM |
+-------+-------------+-----------+----------------+-----+-----------+

                    |<-------- DB-DATA ---------->|
```

- REDIS

  文件的最开头保存着 `REDIS` 五个字符，标识着一个 RDB 文件的开始。

  在读入文件的时候，程序可以通过检查一个文件的前五个字节，来快速地判断该文件是否有可能是 RDB 文件。

- RDB-VERSION

  一个四字节长的以字符表示的整数，记录了该文件所使用的 RDB 版本号。

  目前的 RDB 文件版本为 `0006` 。

  因为不同版本的 RDB 文件互不兼容，所以在读入程序时，需要根据版本来选择不同的读入方式。

- DB-DATA

这个部分在一个 RDB 文件中会出现任意多次，每个 `DB-DATA` 部分保存着服务器上一个非空数据库的所有数据。

- SELECT-DB

  这域保存着跟在后面的键值对所属的数据库号码。

  在读入 RDB 文件时，程序会根据这个域的值来切换数据库，确保数据被还原到正确的数据库上。

- KEY-VALUE-PAIRS

  因为空的数据库不会被保存到 RDB 文件，所以这个部分至少会包含一个键值对的数据。

- EOF

  标志着数据库内容的结尾（不是文件的结尾），值为 `rdb.h/EDIS_RDB_OPCODE_EOF` （`255`）。

- CHECK-SUM

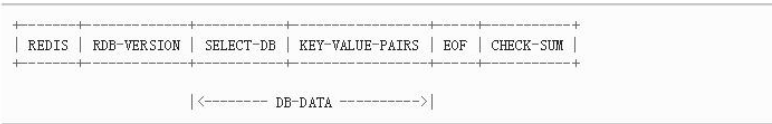
  RDB 文件所有内容的校验和， 一个 `uint_64t` 类型值。

  REDIS 在写入 RDB 文件时将校验和保存在 RDB 文件的末尾， 当读取时， 根据它的值对内容进行校验。

  如果这个域的值为 `0` ， 那么表示 Redis 关闭了校验和功能。

1.4 总结

- `rdbSave` 会将数据库数据保存到 RDB 文件，并在保存完成之前阻塞调用者。
- SAVE 命令直接调用 `rdbSave` ，阻塞 Redis 主进程； BGSAVE用子进程调用 `rdbSave` ，主进程仍可继续处理命令请求。
- SAVE执行期间， AOF 写入可以在后台线程进行， BGREWRITEAOF 可以在子进程进行，所以这三种操作可以同时进行。
- 为了避免产生竞争条件， BGSAVE执行时， SAVE 命令不能执行。
- 为了避免性能问题， BGSAVE 和BGREWRITEAOF 不能同时执行。
- 调用 `rdbLoad` 函数载入 RDB 文件时，不能进行任何和数据库相关的操作，不过订阅与发布方面的命令可以正常执行，因为它们和数据库不相关联。
- RDB 文件的组织方式如下：



2.增量模式持久化(AOF)

Redis 将所有对数据库进行过写入的命令（及其参数）记录到 AOF 文件， 以此达到记录数据库状态的目的， 为了方便起见， 我们称呼这种记录过程为同步。

2.1 文件写入和保存

每当服务器常规任务函数被执行、 或者事件处理器被执行时， `aof.c/flushAppendOnlyFile` 函数都会被调用， 这个函数执行以下两个工作：

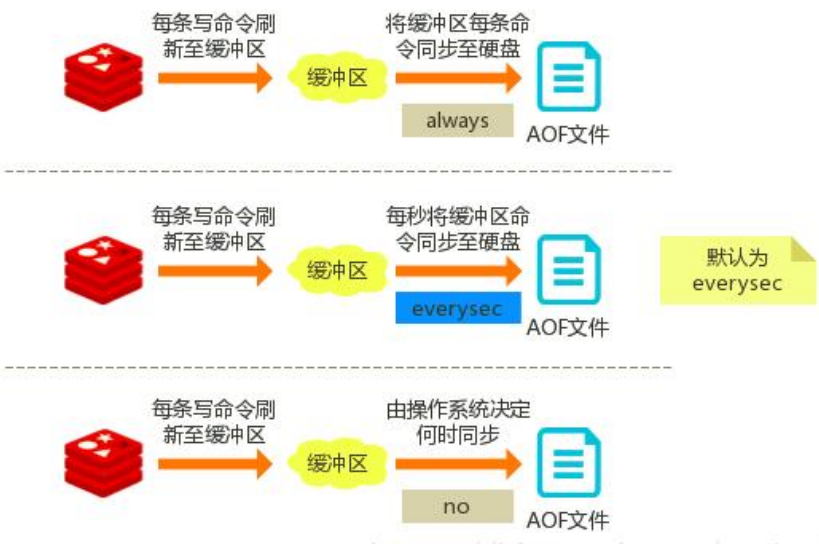
- WRITE：根据条件，将 `aof_buf` 中的缓存写入到 AOF 文件。
- SAVE：根据条件，调用 `fsync` 或 `fdatasync` 函数，将 AOF 文件保存到磁盘中。

两个步骤都需要根据一定的条件来执行， 而这些条件由 AOF 所使用的保存模式来决定

2.2 AOF保存模式

Redis的AOF 包含3 种同步策略

- always：每一次的刷新缓冲区，都会同步触发同步操作。因为每次的写操作都会触发同步，所以该策略会降低Redis的吞吐量，但是这种模式会拥有最高的容错能力。
- every second：每秒异步的触发同步操作，这种是Redis的默认配置。
- no：由操作系统决定何时同步，这种方式Redis无法决定何时落地，因此不可控



2.3 AOF 保存模式对性能和安全性的影响

因为阻塞操作会让 Redis 主进程无法持续处理请求， 所以一般说来， 阻塞操作执行得越少、完成得越快， Redis 的性能就越好。

- 1.不保存（AOF_FSYNC_NO）：写入和保存都由主进程执行，两个操作都会阻塞主进程。

- 2.每一秒钟保存一次（AOF_FSYNC_EVERYSEC）：写入操作由主进程执行，阻塞主进程。保存操作由子线程执行，不直接阻塞主进程，但保存操作完成的快慢会影响写入操作的阻塞时长。
- 3.每执行一个命令保存一次（AOF_FSYNC_ALWAYS）：和模式 1 一样。

综合起来，三种 AOF 模式的操作特性可以总结如下：

模式	WRITE 是否阻塞？	SAVE 是否阻塞？	停机时丢失的数据量
AOF_FSYNC_NO	阻塞	阻塞	操作系统最后一次对 AOF 文件触发 SAVE 操作之后的数据。
AOF_FSYNC_EVERYSEC	阻塞	不阻塞	一般情况下不超过 2 秒钟的数据。
AOF_FSYNC_ALWAYS	阻塞	阻塞	最多只丢失一个命令的数据。

2.4 AOF 文件的读取和数据还原

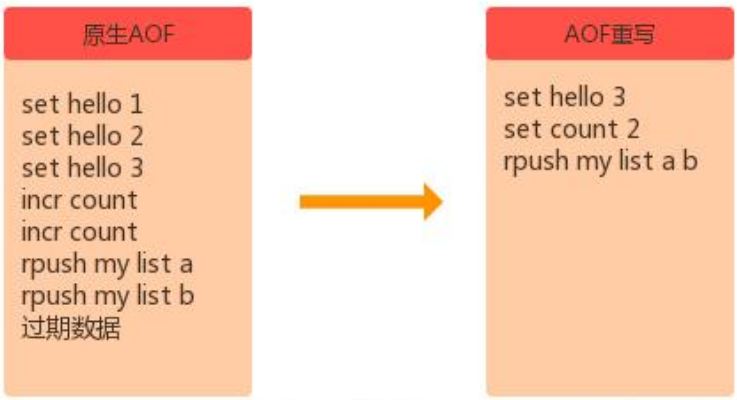
AOF 文件保存了 Redis 的数据库状态， 而文件里面包含的都是符合 Redis 通讯协议格式的命令文本。 这也就是说， 只要根据 AOF 文件里的协议， 重新执行一遍里面指示的所有命令， 就可以还原 Redis 的数据库状态了。 Redis 读取 AOF 文件并还原数据库的详细步骤如下：

- 1.创建一个不带网络连接的伪客户端（fake client）。
- 2.读取 AOF 所保存的文本，并根据内容还原出命令、命令的参数以及命令的个数。
- 3.根据命令、命令的参数和命令的个数，使用伪客户端执行该命令。
- 4.执行 2 和 3 ，直到 AOF 文件中的所有命令执行完毕。

完成第 4 步之后， AOF 文件所保存的数据库就会被完整地还原出来。

2.5 AOF 重写的实现

随着Redis 持续的运行，会有大量的增量数据append 到AOF 文件中。为了减小硬盘存储和加快恢复速度，Redis 通过`rewrite` 机制合并历史AOF 记录。如下所示：



整个流程描述如下：

- 历史AOF：以快照的方式保存。
- 快照写入期间的增量：待快照写入完成之后append 到快照文件中。
- 后续的增量：写入新的AOF。

2.5.1 AOF 后台重写的触发条件

AOF 重写可以由用户通过调用 BGREWRITEAOF 手动触发。

另外， 服务器在 AOF 功能开启的情况下， 会维持以下三个变量：

- 记录当前 AOF 文件大小的变量 `aof_current_size` 。
- 记录最后一次 AOF 重写之后， AOF 文件大小的变量 `aof_rewrite_base_size` 。
- 增长百分比变量 `aof_rewrite_perc` 。

每次当 `serverCron` 函数执行时， 它都会检查以下条件是否全部满足， 如果是的话， 就会触发自动的 AOF 重写：

- 1.没有 BGSAVE 命令在进行。
- 2.没有 BGREWRITEAOF 在进行。
- 3.当前 AOF 文件大小大于 server.aof_rewrite_min_size （默认值为 1 MB ）。
- 4.当前 AOF 文件大小和最后一次 AOF 重写后的大小之间的比率大于等于指定的增长百分比。默认情况下， 增长百分比为 100% ， 也即是说， 如果前面三个条件都已经满足， 并且当前 AOF 文件大小比最后一次 AOF 重写时的大小要大一倍的话， 那么触发自动 AOF 重写。

2.6 总结

- AOF 文件通过保存所有修改数据库的命令来记录数据库的状态。
- AOF 文件中的所有命令都以 Redis 通讯协议的格式保存。
- 不同的 AOF 保存模式对数据的安全性、以及 Redis 的性能有很大的影响。
- AOF 重写的目的是用更小的体积来保存数据库状态，整个重写过程基本上不影响 Redis 主进程处理命令请求。
- AOF 重写是一个有歧义的名字，实际的重写工作是针对数据库的当前值来进行的，程序既不读写、也不使用原有的 AOF 文件。
- AOF 可以由用户手动触发，也可以由服务器自动触发。