# RDB特性

fork一个进程，遍历hash table，利用copy on write，把整个db dump保存下来。save, shutdown, slave 命令会触发这个操作。

粒度比较大，如果save, shutdown, slave 之前crash了，则中间的操作没办法恢复。

## RDB优点

* RDB是一个非常紧凑的文件,它保存了某个时间点得数据集,非常适用于数据集的备份
* RDB是一个紧凑的单一文件,很方便传送到另一个远端数据中心或者亚马逊的S3（可能加密），非常适用于灾难恢复.
* 与AOF相比,在恢复大的数据集的时候，RDB方式会更快一些.
* RDB持久化方式可以最大化redis的性能.

## RDB缺点

* RDB丢失的数据会比较多。
* 当数据集比较大的时候,fork的过程是非常耗时的,会影响redis的性能，岁人AOF也需要fork，但可以调节日志文件的重写频率来提高数据集耐久度。

# 1.02 AOF特性

Redis 可以在 AOF 文件体积变得过大时，自动地在后台对 AOF 进行重写： 重写后的新 AOF 文件包含了恢复当前数据集所需的最小命令集合。 整个重写操作是绝对安全的，因为 Redis 在创建新 AOF 文件的过程中，会继续将命令追加到现有的 AOF 文件里面，即使重写过程中发生停机，现有的 AOF 文件也不会丢失。

## AOF优点

* 使用AOF 会让你的Redis更加耐久
* AOF文件是一个只进行追加的日志文件,所以不需要写入seek,即使由于某些原因(磁盘空间已满，写的过程中宕机等等)未执行完整的写入命令,你也也可使用redis-check-aof工具修复这些问题.
* AOF 文件的内容非常容易被人读懂
* AOF重写新文件未完成，会在旧文件继续写追加。

# 1.03 redis 一致性hash算法

1. **平衡性**

平衡性是指哈希的结果能够尽可能分布在所有的缓冲(Cache)中去，这样可以使得所有的缓冲空间得到利用。很多哈希算法都能够满足这一条件。

1. **单调性**

单调性是指如果已经有一些内容通过哈希分派到了相应的缓冲中，又有新的缓冲加入到系统中。哈希的结果应该能够保证原有已分配的内容可以被映射到原有的或者新的缓冲中去，而不会映射到旧的缓冲集合中的其他缓冲区。

1. **分散性**

在分布式环境中，终端有可能看不到所有的缓冲，而只能看到其中的一部分。当终端希望通过哈希过程将内容映射到缓冲上去，由于不同终端所见的缓冲范围有可能不同，从而导致哈希的结果不一致，最终的结果是相同的内容被不同的终端映射到不同的缓冲区中。这种情况显然是应该避免的，因为它导致相同内容被存储到不同缓冲中去，降低了系统存储的效率。分散性的定义就是上述情况发生的严重程度。好的哈希算法应该能够尽量避免不一致的情况发生，也就是尽量降低分散性。

1. **负载**

负载问题实际上是从另一个角度看待分散性问题。既然不同的终端可能将相同的内容映射到不同的缓冲区中，那么对于一个特定的缓冲区而言，也可能被不同的用户映射到不同的内容。与分散性一样，这种情况也是应当避免的，因此好的哈希算法应能够尽量降低缓冲的负荷。