Redis的强劲性能很大程度上是由于将所有数据都存储在了内存中，然而当Redis重启后，所有存储在内存中的数据就会丢失。在一些情况下，希望Redis能将数据从内存中以某种形式同步到硬盘中，使得重启后可以根据硬盘中的记录恢复数据。这一过程就是持久化。

Redis支持两种方式的持久化，一种是RDB方式，另一种是AOF方式。前者会根据指定的规则将内存中的数据存储在硬盘上，而后者在每次执行命令后将命令本身记录下来。两种持久化方式可以单独使用其中一种，但更多情况下是将二者结合使用。

**一：RDB方式**

RDB方式的持久化是通过快照(snapshotting)完成的，将内存中的所有数据生成一份副本并存储在硬盘上，这个过程即为“快照”。Redis会在以下几种情况下对数据进行快照：

根据配置规则进行自动快照；

用户执行 save或 bgsave命令；

执行flushall命令；

执行复制（replication）时；

1：根据配置规则进行自动快照

Redis允许用户自定义快照条件，符合条件时，Redis会自动执行快照操作。进行快照的条件可在配置文件中自定义，由两个参数构成：时间窗口M和改动的键的个数N。当时间M内被更改的键的个数大于N时，即符合自动快照条件。

配置文件中，每条快照条件占一行，并且以save参数开头。可以存在多个条件，条件之间是“或”的关系。例如下面的3个条件：

save 900 1

save 300 10

save 60 10000

save 900 1 的意思是在15分钟（900 秒）内有一个或一个以上的键被更改，则进行快照。同理，save 300 10表示在300 秒内至少有10个键被修改则进行快照。

2：执行save或bgsave命令

当进行服务重启、手动迁移或者备份时，需要手动执行快照操作。Redis提供了两个命令来完成这一任务：

save命令，执行save命令时，Redis同步地进行快照操作，在快照执行的过程中会阻塞所有来自客户端的请求。数据比较多时，这一过程会导致Redis较长时间不响应，所以尽量避免在生产环境中使用这一命令。

bgsave命令，手动执行快照推荐使用bgsave命令。bgsave命令可以在后台异步地进行快照操作，快照的同时服务器还可以继续响应来自客户端的请求。执行bgsave后Redis会立即返回OK表示开始执行快照操作，如果想知道快照是否完成，可以通过lastsave命令获取最近一次成功执行快照的时间，返回结果是一个Unix时间戳，如：

127.0.0.1**:**6379**>** lastsave

**(**integer**)** 1449281306

异步快照的具体过程在”快照原理”时讲述，执行自动快照时Redis采用的策略即异步快照。

3：执行 flushall命令

flushall命令会清除所有数据库中的所有数据。执行此命令时，只要自动快照条件不为空，Redis就会执行一次快照操作。当没有定义自动快照条件时，执行flushall不会进行快照。

例如，当定义的快照条件为当1秒内修改10000个键时进行自动快照，而当数据库里只有一个键时，执行flushall命令也会触发快照，即使这一过程实际上只有一个键被修改了。

4：执行复制时

当设置了主从模式时，Redis会在复制初始化时进行自动快照。关于主从模式和复制的过程会后续详细介绍，这里只需要了解当使用复制操作时，即使没有定义自动快照条件，并且没有手动执行过快照操作，也会生成RDB快照文件。

**二：RDB快照原理**

Redis默认将快照文件存储在Redis当前进程的工作目录中的dump.rdb文件，可以通过配置文件中的dir和dbfilename两个参数分别指定快照文件的存储路径和文件名。

快照的过程如下：

a：Redis使用fork函数创建子进程；

b：父进程继续接收并处理客户端发来的命令，而子进程开始将内存中的数据写入硬盘中的临时文件；

c：当子进程写入完所有数据后会用该临时文件替换旧的RDB文件，至此一次快照操作完成。所以新的RDB文件存储的是执行fork一刻的内存数据。

执行fork时，操作系统会使用写时复制策略，即fork函数发生的一刻父子进程共享同一内存数据，当父进程要更改其中某片数据时（如执行一个写命令），操作系统会将该片数据复制一份以保证子进程的数据不受影响。

写时复制策略保证了在fork的时刻虽然看上去生成了两份内存副本，但实际上内存的占用量并不会增加一倍。 比如系统内存只有2 GB，而Redis数据库的内存有1.5 GB时，执行fork后内存使用量并不会增加到3 GB（超出物理内存）。

另外需要注意的是，当进行快照的过程中，如果写入操作较多，造成fork前后数据差异较大，是会使得内存使用量显著超过实际数据大小，因为内存中不仅保存了当前的数据库数据，而且还保存着 fork 时刻的内存数据。

要确保Linux系统允许应用程序申请超过可用内存（物理内存和交换分区）的空间，方法是在/etc/sysctl.conf 文件加入：”vm.overcommit\_memory = 1”，然后重启系统或者执行”sysctl vm.overcommit\_memory=1”确保设置生效。

通过上述过程可以发现Redis在进行快照的过程中不会修改RDB文件，只有快照结束后才会将旧的文件替换成新的，也就是说任何时候RDB文件都是完整的。这使得我们可以通过定时备份 RDB 文件来实现Redis数据库备份。

RDB文件是经过压缩（可以配置 rdbcompression 参数以禁用压缩节省CPU 占用）的二进制格式，更加利于传输。Redis启动后会读取RDB快照文件，将数据从硬盘载入到内存。

通过RDB方式实现持久化，一旦Redis异常退出，就会丢失最后一次快照以后更改的所有数据。这就需要开发者根据具体的应用场合，通过组合设置自动快照条件的方式来将可能发生的数据损失控制在能够接受的范围。例如，使用Redis存储缓存数据时，丢失最近几秒 的数据或者丢失最近更新的几十个键并不会有很大的影响。如果数据相对重要，希望将损失降到最小，则可以使用 AOF方式进行持久化。

**三：AOF方式**

使用Redis存储非临时数据时，一般需要打开AOF持久化来降低进程终止导致的数据丢失。AOF可以将Redis执行的每一条写命令追加到硬盘文件中，这一过程显然会降低Redis的性能，但是大部分情况下这个影响是可以接受的。

默认情况下Redis没有开启AOF(append only file)方式的持久化，可以通过 appendonly参数启用：

appendonly yes

开启AOF持久化后每执行一条会更改Redis中数据的命令，Redis就会将该命令写入硬盘中的AOF文件。AOF文件的保存位置和RDB文件的位置相同，都是通过dir参数设置的，默认的文件名是appendonly.aof，可以通过appendfilename 参数修改：

appendfilename appendonly.aof

AOF文件以纯文本的形式记录了Redis执行的写命令，比如在开启AOF持久化的情况下执行了如下4个命令：

127.0.0.1**:**6379**>** set foo 1

OK

127.0.0.1**:**6379**>** set foo 2

OK

127.0.0.1**:**6379**>** set foo 3

OK

127.0.0.1**:**6379**>** get foo

"3"

Redis会将前3条命令写入AOF文件中，此时AOF文件中的内容如下：

**\***2

$6

**SELECT**

$1

**0**

**\***3

$3

**set**

$3

**foo**

$1

**1**

**\***3

$3

**set**

$3

**foo**

$1

**2**

**\***3

$3

**set**

$3

**foo**

$1

**3**

AOF文件的内容正是Redis客户端向Redis发送的原始通信协议的内容，从中可见Redis确实只记录了前3条命令。然而这有一个问题是前2条命令其实都是冗余的，因为这两条的执行结果会被第三条命令覆盖。随着执行的命令越来越多，AOF文件的大小也会越来越大，很自然地，我们希望Redis可以自动优化AOF文件，就上例而言，就是将前两条无用的记录删除，只保留第三条。

实际上Redis也正是这样做的，每当达到一定条件时Redis就会自动重写AOF文件，这个条件可以在配置文件中设置：

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

auto-aof-rewrite-percentage参数，表示当目前的AOF文件大小超过上一次重写时的AOF文件大小的百分之多少时会再次进行重写，如果之前没有重写过，则以启动时的AOF文件大小为依据。

auto-aof-rewrite-min-size参数限制了允许重写的最小AOF文件大小，通常在AOF文件很小的情况下即使其中有很多冗余的命令我们也并不太关心。

除了让Redis自动执行重写外，还可以主动使用bgrewriteaof命令手动执行AOF重写：

127.0.0.1**:**6379**>** bgrewriteaof

Background append only file rewriting started

上例中的AOF文件重写后的内容为：

**\***2

$6

**SELECT**

$1

**0**

**\***3

$3

**SET**

$3

**foo**

$1

**3**

可见冗余的命令已经被删除了。重写的过程只和内存中的数据有关，和之前的AOF文件无关，这与RDB很相似， 只不过二者的文件格式完全不同。

在启动时Redis会逐个执行AOF文件中的命令来将硬盘中的数据载入到内存中，载入的速度相较RDB会慢一些。

虽然每次执行更改数据库内容的操作时，AOF都会将命令记录在AOF文件中，但是由于操作系统的缓存机制，数据并没有真正地写入硬盘，而是进入了系统的硬盘缓存。在默认情况下系统每30秒会执行一次同步操作，以便将硬盘缓存中的内容真正地写入硬盘，在这30秒的过程中如果系统异常退出则会导致硬盘缓存中的数据丢失。

一般来讲启用AOF持久化的应用都无法容忍这样的损失，这就需要Redis在写入AOF文件后主动要求系统将缓存内容同步到硬盘中。在 Redis 中可以通过 appendfsync 参数设置同步的时机：

# appendfsync always

appendfsync everysec

# appendfsync no

默认情况下Redis采用everysec规则， 即每秒执行一次同步操作。always表示每次执行写入都会执行同步，这是最安全也是最慢的方式。no表示不主动进行同步操作，而是完全交由操作系统来做 （即每30秒一次），这是最快但最不安全的方式。一般情况下使用默认值 everysec就足够了，既兼顾了性能又保证了安全。

Redis允许同时开启AOF和RDB， 既保证了数据安全又使得进行备份等操作十分容易。此时重新启动Redis后，Redis 会使用AOF文件来恢复数据，因为AOF方式的持久化可能丢失的数据更少。