# 7 持久化

在一些情况下，我们会希望Redis 在重启后能够保证数据不丢失，例如：

1·将Redis 作为数据库使用时。

2·将Redis 作为缓存服务器，有可能出现的缓存雪崩会使服务无法响应。

这时我们希望Redis 能将内存中的数据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“定时”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**。Redis 允许同时开启 AOF 和 RDR ，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。

## 7.1 RDB方式

RDB 方式的持久化是通过**快照**完成的，当符合一定条件时Redis 会自动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：

1·根据配置规则进行自动快照；

2·用户执行save或bgsave 命令；

3·执行flushall命令；

4·执行复制时；

### 7.1.1根据配置规则进行自动快照

Redis 允许用户自定义快照条件，当符合快照条件时， Redis 会自动执行快照操作。

快照条件可以由用户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 。每当时间 M 内被更改的键的个数大于 N 时，即符合自动快照条件。

|  |
| --- |
| 例如 Redis安装目录中包含的样例配置文件中预置的 3 个条件：  save 900 1 // 在900秒内至少有1个键被更改则进行快照  save 300 10 // 在300秒内至少有10个键被修改则进行快照  save 60 10000 // 在60秒内至少有10000个键被修改则进行快照 |
| 每条快照条件占一行，并且以 save 参数开头。  同时可以存在多个条件，条件之间是“或”的关系。 |

### 7.1.2用户执行save或bgsave 命令

当进行服务重启、手动迁移以及备份时，我们需要**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** |
| 当执行 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。当数据库中的数据比较多时，这一过程会导致 Redis 较长时间不响应，所以要尽量避免在生产环境中使用这一命令。 |
| **BGSAVE 命令(推荐)** |
| 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 时间戮，如：( integer ) 1423537869。  **执行自动快照时，redis采用的策略即异步快照**。 |

### 7.1.3执行flushall命令

当执行flushall命令时，Redis 会**清除数据库中的所有数据**。

**需要注意的是，不论清空数据库的过程是否触发了自动快照条件，只要自动快照条件不为空，Redis 就会执行一次快照操作**。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。当没有定义自动快照条件时，执行flushall则不会进行快照。

### 7.1.4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。

### 7.1.5 快照原理

Redis 默认会将快照文件存储在 Redis 当前进程的工作目录中的 dump.rdb 文件中，可以通过配置 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** |
| 1·Redis 使用 **fork 函数**复制一份当前进程（父进程）的副本（子进程）;  2·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；  3·当子进程写入完所有数据后会用该临时文件替换掉旧的 RDB 文件，至此一次快照操作完成。 |
| **fork函数** |
| 在执行 fork 的时候操作系统（类 Unix 操作系统）会使用**写时复制(copy-on-write)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，所以新的 RDB 文件存储的是执行 fork 那一刻的内存数据。  copy-on-write也保证了在 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．这就意味着当系统内存只有 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）。为此需要**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文件中加入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。  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的，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。进行内存用量估算时很容易忽略这一问题，造成内存用量超限。 |

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**（可以通过配置rdbcompression 参数以禁用压缩节省 CPU 占用）的二进制文件，所以占用的空间会小于内存中的数据大小，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**。根据数据量大小与结构和服务器性能不同，这个时间也不同。通常将一个记录 1000 万个字符串类型键、大小为1 GB 的快照文件载入到内存中需要花费 20 到30 秒。

**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**。

## 7.2 AOF方式

当使用 Redis 存储非临时数据时，一般需要打开 AOF 持久化来降低进程中止导致的数

据丢失风险。 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，另外使用较快的硬盘可以提高AOF 的性能。

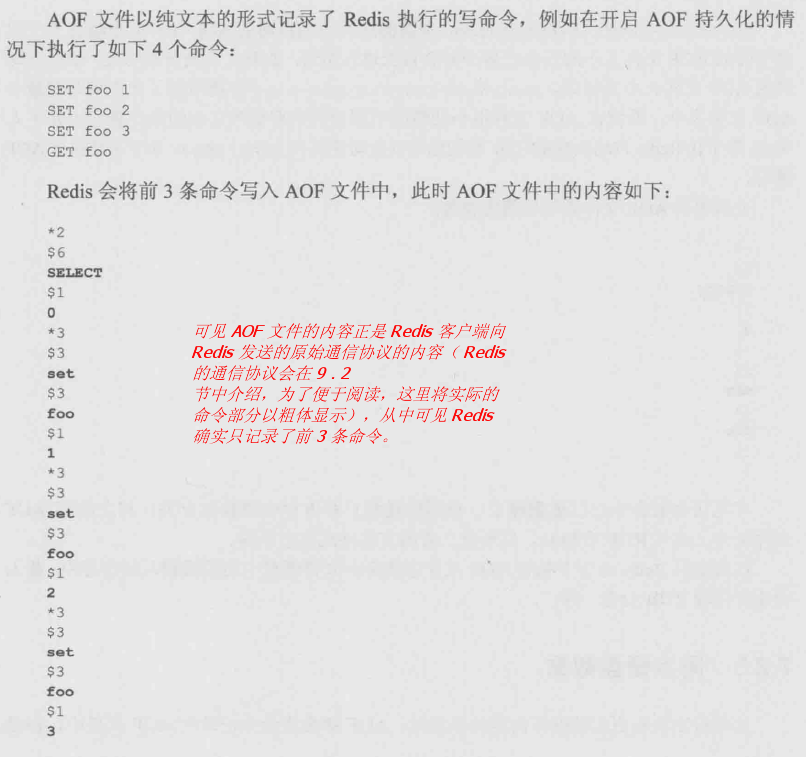
### 7.2.1 开启AOF

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 中的数据的命令， 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。

### 7.2.2 AOF的实现



然而这时有一个问题是前 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，即使内存中实际的数据可能并没有多少。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 | 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 64mb | 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 自动执行重写外我们还可以主动使用 **BGrewriteAOF 命令手动执行 AOF重写**。

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。

### 7.2.3同步硬盘数据

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，但是事实上，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 中我们可以通过 appendfsync 参数设置同步的时机：

|  |  |
| --- | --- |
| **appendfsync everysec** | 默认情况下 Redis 采用everysec规则，即每秒执行一次同步操作 |
| appendfsync 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 |
| appendfsync 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的everysec就足够了，既兼顾了性能又保证了安全。 | |