AOF恢复机制

AOF是Redis数据持久化的一种方式， 是一种预写日志。 AOF同样有一个文件， 合适的时机， 将Redis所做的数据修改添加到aof日志中， 然后重启Redis时加载日志。

这就导致了一个问题， 就是Redis重启时要参考两个文件， 一个是RDB文件， 一个是AOF预写日志。 这里的规则是， 如果开启了AOF预写日志， 那么Redis就参考AOF预写日志， 否则就参考RDB日志。 开启AOF日志需要在redis.conf中开启：

```
#
# Please check http://redis.io/topics/persistence for more information.

appendonly no

# The name of the append only file (default: "appendonly.aof")

appendfilename "appendonly.aof"
```

将appendonly 设置为yes， 即表示开启AOF， 设置为no， 表示为关闭AOF。 然后appendfilename 表示AOF日志的文件名， 后面的"appendonly.aof"就说明这个预写日志的文件名是appendonly.aof。

AOF工作流程

AOF命令追加

AOF主要经过将命令追加到内存， 然后将内存中的数据拷贝到磁盘中的策略实现AOF预写日志。 首先命令追加就是主进程将写命令写入到buff内存中。

AOF同步文件策略

AOF第二步就要进行文件同步

常见的有三种策略：

- 一种是always：每次redis将命令追加到buff内存， 都会调用写入磁盘操作。 频率最高的， 但是性能最低的。
- 一种是everysecond：每次将redis的写命令追加到buff内存， 但是没过一秒才进行一次buff内存和磁盘的同步， 即将buff内存中的数据拷贝到内存， 所以理论上最多丢失1秒的数据。 频率减少， 性能增加。

- 一种是no：只是将命令追加到buff内存中， 但是不将buff内存中的数据同步到磁盘， 同步时机由操作系统决定。

AOF重写机制

AOF文件一旦数据量大了， Redis恢复需要遍历整个文件， 那么就要消耗很大的时间， 为了减少遍历的资源占用， 就将AOF进行压缩， 这个叫做AOF的重写机制。具体机制如下

AOF重写机制

AOF重写机制是对冗余的命令进行压缩， 得到一个更精简的AOF文件。

重写可以通过配置文件配置自动触发：

```
#
# Specify a percentage of zero in order to disable the automatic AOF
# rewrite feature.

auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
```

也可以通过命令手动触发：bgrewriteaof

重写流程是：

- 先fork一个子进程。
- 然后子进程创建一个新的临时AOF文件， 并且子进程读取当前数据库的状态保存到临时AOF文件中。
- 此时父进程可能正在接受上层的写请求， 那么他就要一遍向旧的AOF缓冲区中追加命令， 一边向一个新的临时重写缓冲区中追加命令。
- 当子进程完成临时AOF文件的重写， 就通知主进程， 主进程再将临时重写缓冲区的数据追加到临时AOF文件中， 最后替换旧文件为新文件。

AOF的优缺点

优点

- AOF记录的是Redis的流水，记录的是做了那些修改， 而RDB记录的是一个快照， 是记录的全量数据。 如果AOF和RDB的频率相同， 那么RDB的时间消耗要大于AOF的消耗。因为AOF只需要记录命令做的修改， 而RDB要重新生成一份全量数据。所以AOF能够精确到更细的时间粒度， 数据的安全性更高。
- AOF是文本， 可以防止误删， 损坏AOF文件。

缺点

- RDB内部保存的是二进制， 恢复的速度更快。 AOF是文本， 恢复的速度要慢一些。
- RDB文件是二进制文件， 更小。
- RDB使用后台进程同步数据库状态， AOF使用主进程追加命令到缓冲区， RDB对性能的影响更小。