# 概述

本文档学习记录redis3.0源码的笔记

# 数据结构

参考<http://zhangtielei.com/posts/blog-redis-dict.html>

## 简单动态字符串

Simple Dynamic String（SDS），是一个具有长度（len）和剩余空间（free）的字符数组。结构体定义文件sds.h/sdshdr

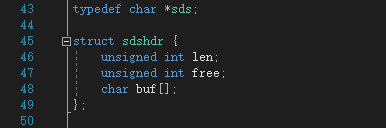


图2- 1 简单动态字符串结构体定义

## 链表

redis中定义的链表结构是一个双向链表，链表节点具有前驱pre和后继next，同时链表具有头节点head和尾节点tail。其定义文件在adlist.h。

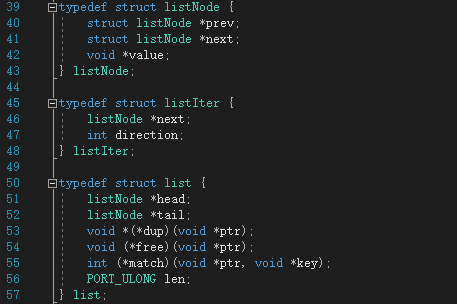


图2- 2 链表结构体定义

## 字典结构

redis中的字典结构（dict）类似于java语言中的hashmap结构，其底层实现是通过数组来存放每一个dictEntry。

dictEntry是一个键值对结构，包含一个void指针的key和一个void指针的value，同时包含一个next指针，该指针是解决hash冲突而采用的拉链法，将所有hash值一样的元素通过next指针串在一起。

dictht是包含dictEntry数组的结构，除了该数组之后，还包含数组大小size，sizemask=size-1（方便获取index，因为size是2的幂，size-1就是二进制全为1的数）。

dict结构体则是包含2个dictht元素的数组（如果没有进行rehash（即rehashidx=-1），只有ht[0]有效，如果rehashidx！=-1，处于rehash过程，所有的操作都要在两个dictht上进行）。

两个hash表最主要的作用是实现dict扩容时（rehash）的增量rehash（incremental rehash），也就是当dict负载超过load factor时，则将进行增量rehash，当该dict有增删改查操作时，需要重新计算index并将key，value存放在另一个hash表。

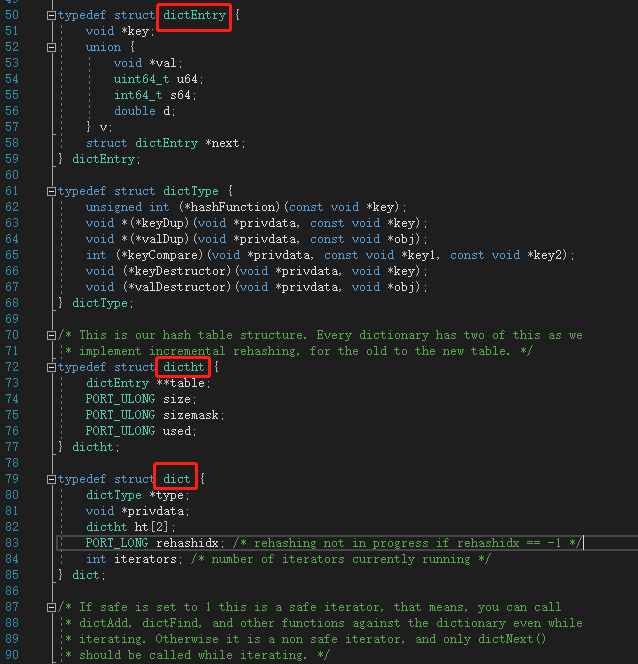


图2- 3 字典结构体定义

Rehash步骤大致如下：

1. 将rehashidx=0，表示在ht[0]中index=0的bucket中所有key需要进行rehash并转移到ht[1]中对应index中；
2. 如果有请求访问redis，先判断rehashidx是否为-1，不为-1则将ht[0]中对应rehashidx进行扩容，同时rehashidx+1，便于下一次请求；

这种做法可以将一次rehash分摊到多次请求上去，这样可以避免某一次请求因为进行rehash而导致响应延时。

有关字典的操作在dict.c文件。

### 创建

创建一个字典结构的源码如下，其主要工作是创建一个dict指针，dict中的两个hash表都未分配空间。

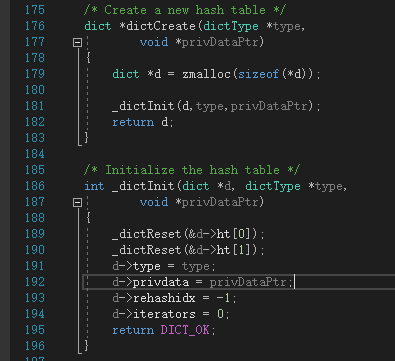


图2- 4 创建字典结构

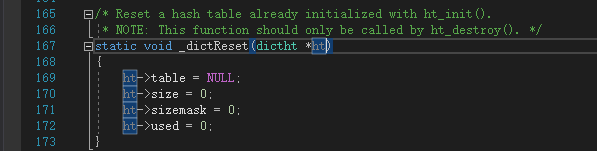


图2- 5初始化dictht

### 增

在dict中增加一个(key，value)分为了两步，第一步是先将key存放在对应bucket，然后再给Entry赋值value，代码如下：

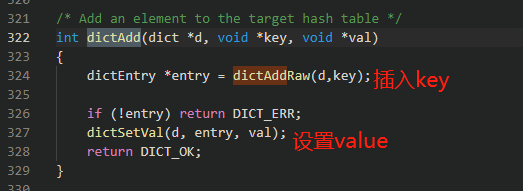


图2- 6 dict插入接口

插入key时，首先判断是否在rehashing，如果在rehash，则推进一步，并且在ht[1]中计算index，否则在ht[0]中计算index；然后再判断是否存在相同的key，存在只返回

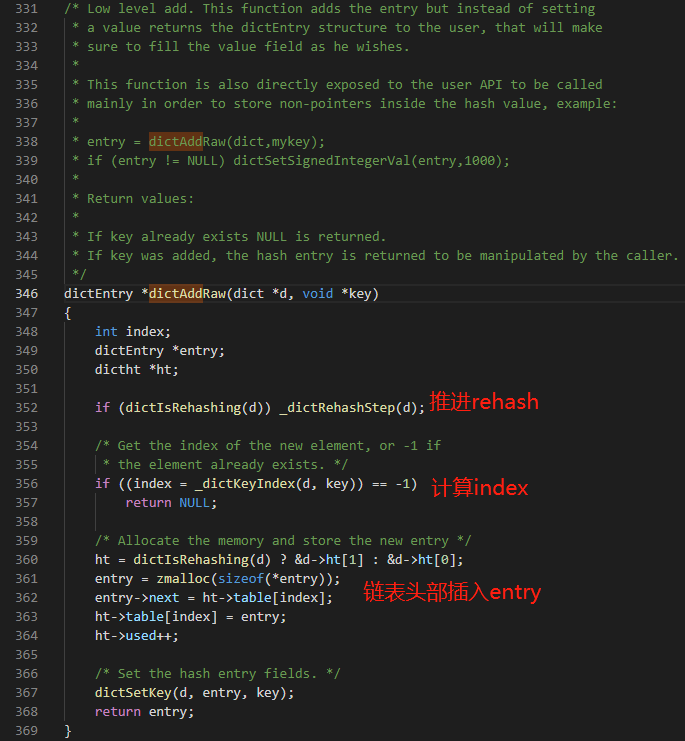


图2- 7 插入前判断

获取key对应的index时，需要先判断对应的key是否已存在，代码如下：

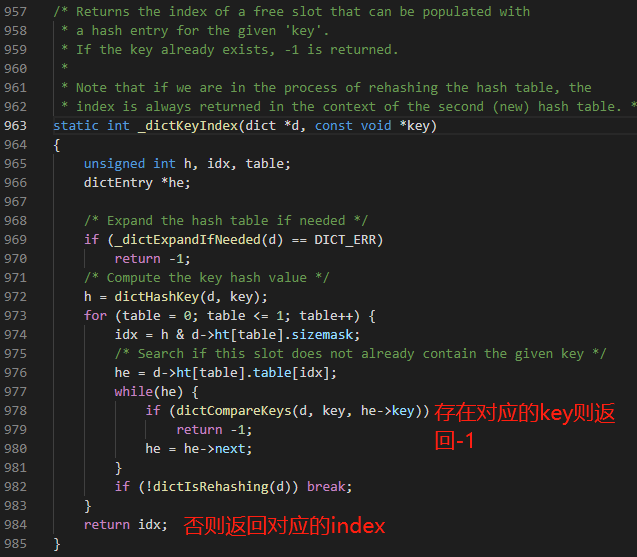


图2- 8 计算index

总结一下：

1. 是否推进rehash；
2. 判断key是否存在，存在返回-1，否则返回对应index；
3. 如果处于rehashing中，在ht[1]中插入
4. 插入时在bucket链表的头部插入

### 删

删除操作比较简单，先判断是否rehashing，然后定位，最后删除。代码如下：

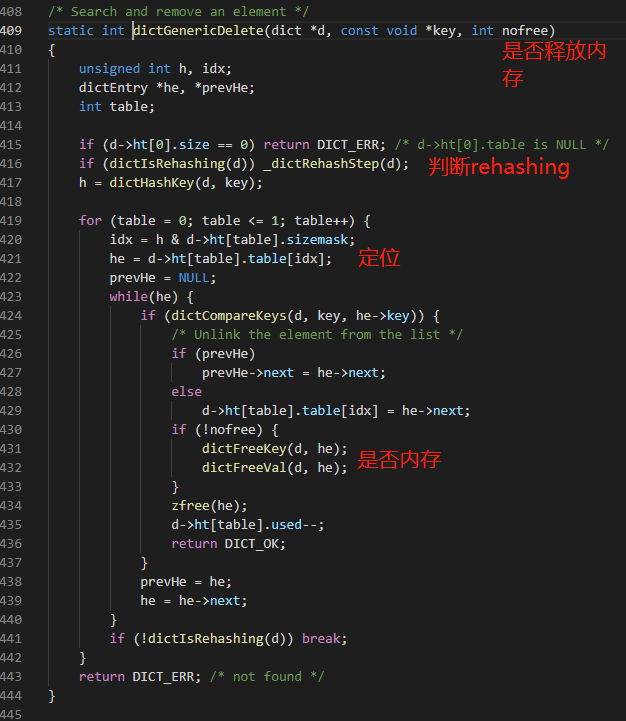


图2- 9删除操作

### 改

修改指定key值的value，先判断rehashing，然后定位，最后修改value。代码如下：

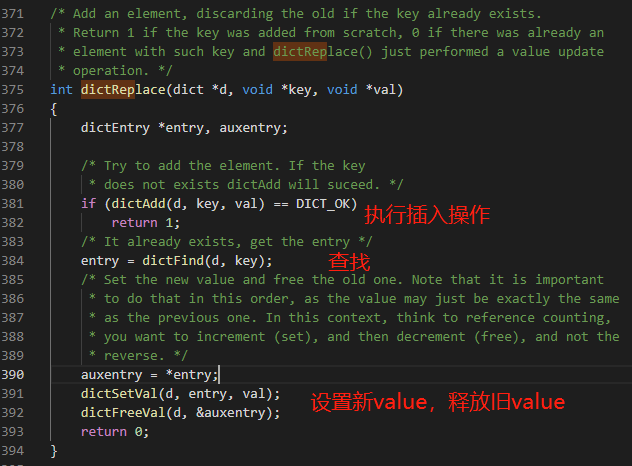


图2- 10 修改操作

### 查

查询指定key值的value，先判断rehashing，然后定位，最后返回value。代码如下：

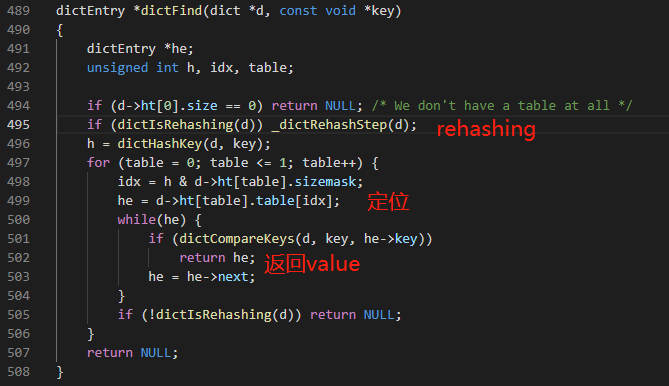


图2- 11 查询操作

### Rehash

（均摊法）Rehash操作可能存在两种情况，第一种就是需要rehash的bucket不为空，则直接推进一步，如果需要rehash的bucket为空，redis规定一次最多处理连续10个空bucket。如果本次rehash之后ht[0]中used=0，表示所有的节点都已rehash，需要先释放ht[0]，然后将ht[1]赋值给ht[0]，ht[1]设置为null。具体代码如下所示：

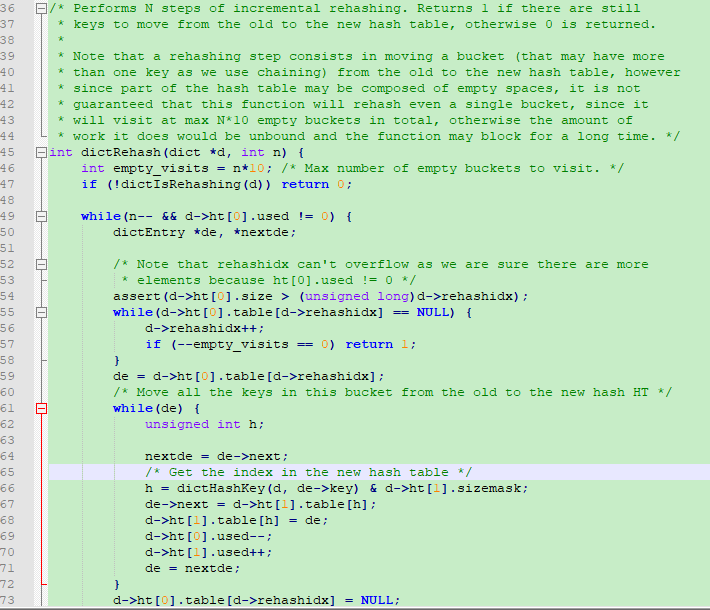


图2- 12 Rehash过程

## 跳表

# 服务端Server

## 初始化默认配置

此阶段是给redis所有需要的配置参数进行默认赋值，也就是加载redis内部设置的默认值。此阶段通过调用redis.c文件中实现的initServerConfig方法进行的。

服务器在启动的开始阶段就需要执行这个方法来设置一些配置参数，因为在后续的执行过程中需要用到一些配置参数，因此需要提前设置默认值；

## 初始化Sentinel（可选）

这一步是可选操作，需要根据启动命令或者启动参数来判断是否执行这一步。如果启动服务器采用“redis-sentinel”命令或者命令中带有“-sentinel”参数，则需要进行sentinel的初始化操作。

判断是否初始化sentinel的方法实现如下，位于redis.c/checkForSentinelMode

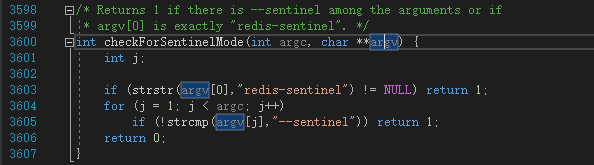


图3- 1 sentinel模式判断逻辑

## 入参解析

经过上面的初始化操作之后，接下来需要对启动命令中带有的参数进行解析。比如启动服务时指定配置文件（conf）所在路径。理解这一步需要了解redis启动的所有命令。

## 加载配置（可选）

如果启动命令指定了配置文件，服务启动时需要加载指定的配置文件并为相应的参数进行赋值。加载配置方法如下，位于config.c/loadServerCofig。

## 初始化服务

初始化服务主要为服务器server分配空间，设置server对象中各个属性的值，主要包括如下过程：

1. 初始化db

初始化db主要为db数组分配连续的内存空间，数组大小默认为16，可以通过配置文件设置dbnum。

## 加载持久化数据

加载持久化数据方法位置redis.c/loadDataFromDisk。Redis提供两种数据持久化方案，一种是AOF，一种是RDB方式，通过代码可以知道，Redis优先加载AOF方式，只要aof方式状态为on，则采用AOF加载，否则加载RDB文件。两种加载方式逻辑需要重点关注。

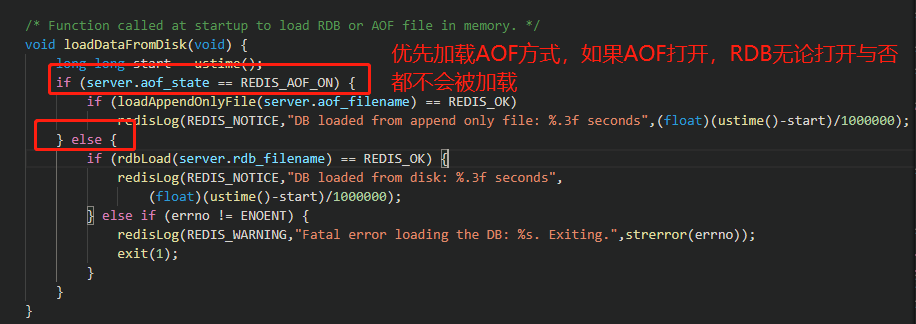


图2- 13 aof和rdb优先级

### 加载AOF

在理解加载AOF文件时，首先需要理解AOF文件存储格式，如下是一个aof文件样本：

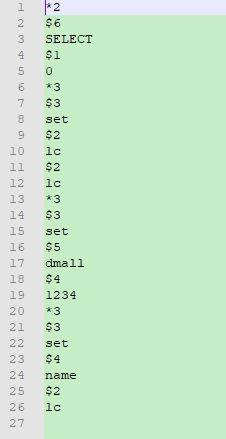


图3- 2 aof文件样本

图3- 2 对应redis命令如下：

>select 0

>set lc lc

>set dmall 1234

>set name lc

每一个redis命令行存放到aof文件时都按照如下格式进行存储

* 假设改命令行具有n个参数（包括指令，比如set，hset等）
* aof文件需要存放2\*n + 1行
* 第一行表示n，并以星号（\*）开始；
* 然后每一个参数占用2行，前一行表示该参数长度，以($)打头，后一个参数就是具体值

Aof文件加载流程图可以大致如图3- 3所示：



图3- 3 aof文件加载流程

Redis中fake client与一般的client最大的区别是其fake client的套接字编号fd为-1，因为fake client不能用于处理网络事件，只能用于载入aof文件和执行lua脚本。

### 加载RDB

Redis产生的RDB文件都是一个二进制文件，文件内容不存在空格，换行符等。文件结构格式如所示：

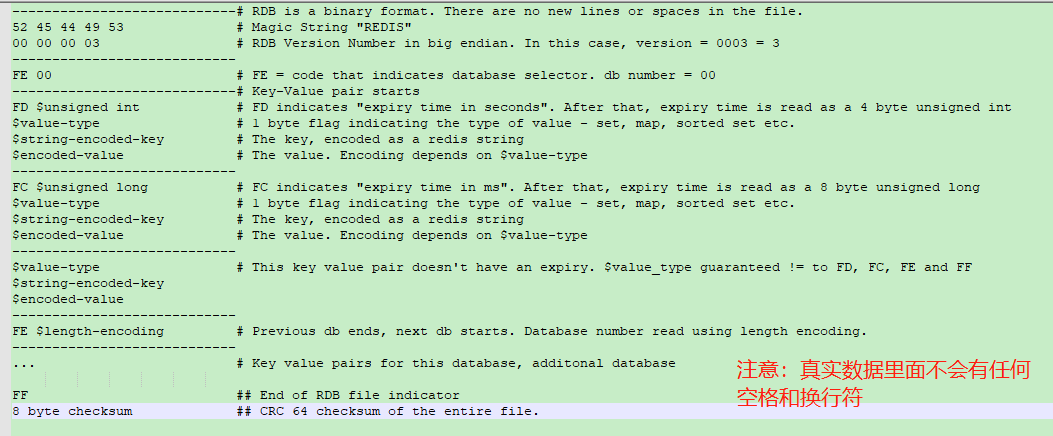


图3- 4 RDB文件格式

注：

参考地址<https://github.com/sripathikrishnan/redis-rdb-tools/wiki/Redis-RDB-Dump-File-Format>

## 监听事件

# 客户端

# 持久化

Redis持久化包括两种方式，一种是aof（appendonlyfile），另外一种是rdb。针对这两种方式，现详细学习记录其实现原理。

参考

[https://www.cnblogs.com/kismetv/p/9137897.html#t4](https://www.cnblogs.com/kismetv/p/9137897.html" \l "t4)

## AOF

打开或者关闭AOF持久化可以采用三种方式，分别是：

* Redis配置文件redis.conf配置appendonly yes
* Redis运行过程（run time）中通过config 命令打开，比如config set appendonly yes
* Redis运行过程（run time）中通过config 命令关闭，比如config set appendonly no

### 配置文件打开aof

通过这种方式启动aof在redis启动时检测配置文件，前提appendonly为yes，详见3.6.1。

### config命令打开aof

通过这种方式需要client发送‘config set appendonly yes’命令给服务器。服务器接收到该命令后，先解析config的指令，需要调用config.c/configSetCommd方法，该方法会根据config指令的参数调用aof.c/startAppendOnly函数。代码如下：

该方法的核心是调用rewriteAppendOnlyFileBackground函数，其主要流程如下

1. Redis父进程首先判断当前是否存在正在执行 bgsave/bgrewriteaof的子进程，如果存在则bgrewriteaof命令直接返回，如果存在bgsave命令则等bgsave执行完成后再执行。前面曾介绍过，这个主要是基于性能方面的考虑。
2. 父进程执行fork操作创建子进程，这个过程中父进程是阻塞的。
3. 父进程fork后，bgrewriteaof命令返回”Background append only file rewrite started”信息并不再阻塞父进程，并可以响应其他命令。Redis的所有写命令依然写入AOF缓冲区，并根据appendfsync策略同步到硬盘，保证原有AOF机制的正确。
4. 由于fork操作使用写时复制技术，子进程只能共享fork操作时的内存数据。由于父进程依然在响应命令，因此Redis使用AOF重写缓冲区(图中的aof\_rewrite\_buf)保存这部分数据，防止新AOF文件生成期间丢失这部分数据。也就是说，bgrewriteaof执行期间，Redis的写命令同时追加到aof\_buf和aof\_rewirte\_buf两个缓冲区。
5. 子进程根据内存快照，按照命令合并规则写入到新的AOF文件。
6. 子进程写完新的AOF文件后，向父进程发信号，父进程更新统计信息，具体可以通过info persistence查看。
7. 父进程把AOF重写缓冲区的数据写入到新的AOF文件，这样就保证了新AOF文件所保存的数据库状态和服务器当前状态一致。
8. 使用新的AOF文件替换老文件，完成AOF重写。

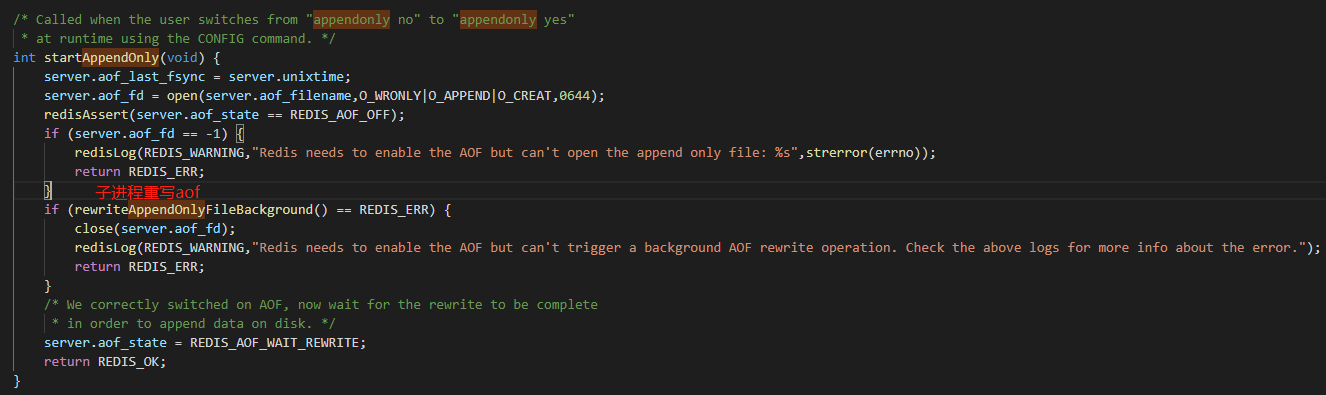


图5- 1 config命令打开aof

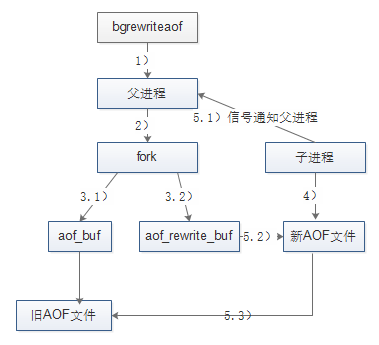


图5- 2 bgrewriteaof函数流程图

### config命令关闭aof

通过config命令关闭aof的具体实现在aof.c/stopAppendOnly，其基本思路是将aof\_buf中缓存的数据flush到磁盘，然后设置相应的server参数。

## RDB

# IO