# Redis持久化

Redis提供两种持久化方式RDB和AOF。

## RDB持久化

持久化方式是将数据库中的所有数据写入RDB文件中。当redis中数据量较大的时候，一次性写入所有数据一来耗时很长，二来对性能损耗也比较大。

### 相关配置项

save 300 10，表示300秒内有至少10个key发生变化，则生成RDB文件。

dbfilename <filename>，指定生成的RDB文件名称。

dir <filepath>，RDB文件生成的路径。

### RDB文件的创建

redis提供两个命令用来创建RDB文件，SAVE和BGSAVE。区别是SAVE命令会阻塞进程，即在生成RDB文件期间，redis不能对外提供服务。BGSAVE是通过子进程的方式创建RDB文件。redis仍然可以对外提供服务。

BGSAVE命令执行期间，对于SAVE、BGSAVE、BGREWRITEAOF三个命令处理比较特殊。在BGSAVE执行期间，SAVE是不能执行的，否则父子进程就同时被阻塞了。BGSAVE也是不能执行，因为两个子进程同时操作一个RDB文件会出现问题。BGREWRITEAOF命令则是会延后到BGSAVE命令执行完成之后再执行（而反过来，如果在执行BGREWRITEAOF命令期间，BGSAVE命令会被直接拒绝）。

除了通过命令的方式，redis还会根据上面说的配置定时创建。在serverCron函数里面有一段逻辑会判断当前redis是否已经满足了save配置中的各个条件，如果已经满足则调用rdbSaveBackground函数。因此save配置对应的是BGSAVE的处理方式。实际使用中save配置是可以配置很多个不同条件的，只要有一个条件满足，就会触发BGSAVE的处理。

#### 关键函数

##### rdbsave

入参filename：需要创建的RDB文件名称

出参：成功返回REDIS\_OK，失败返回REDIS\_ERR。

SAVE命令直接调用该函数。

##### rdbSaveBackground

入参filename：需要创建的RDB文件名称

出参：成功返回REDIS\_OK，失败返回REDIS\_ERR。

BGSAVE命令调用该函数，在函数中创建子进程，通过子进程执行rdbsave函数。函数流程如下：

1. 判断BGSAVE是否已经在执行，已经在执行就返回REDIS\_ERR。
2. 记录执行BGSAVE之前redis被修改的次数。
3. 更新尝试执行BGSAVE的时间。
4. 记录当前时间t1，并创建子进程。
5. 子进程关闭所有的sockets连接，调用rdbSave创建RDB文件。
6. 文件创建成功调用exit返回1，失败调用exit返回0。
7. 父进程记录子进程id，BGSAVE开始事件，关闭rehash等操作之后就可以正常退出，去执行其他任务。

需要注意的是，这个函数里没有调用wait函数在子进程退出的时候回收子进程的资源。而是在serverCron函数里面调用了wait3来回收子进程的资源。

##### backgroundSaveDoneHandler

入参exitcode：子进程退出时候设置的返回值。

入参bysignal：

出参：无。

该函数用来处理BGSAVE完成之后的一些事宜，主要是对于server中dirty、lastsave等字段的更新。然后调用udateSlavesWaitingBgsave，处理等待BGSAVE结束的各个slave。

### RDB文件的读取

RDB文件在redis启动的时候读取，不依赖于任何命令。要弄清楚RDB文件的读取逻辑，首先要弄清楚RDB文件的格式。



1. 文件开头通过”REDIS”这5个字符来标识是否是RDB文件。因为RDB保存的是二进制数据，所以不是”REDIS”字符串。
2. db\_version表示RDB文件的版本号，固定4字节。
3. databases表示redis中各个db以及db中所存储的各类数据。如果redis中没有数据，那么这个字段是空的，长度为0。或者说就没有这个字段。
4. EOF表示databases数据的结束。
5. check\_sum是根据前面所有的数据计算出来的校验和。

databases字段展开之后格式如下：

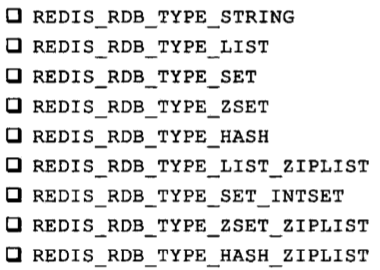


1. SELECTDB是一个常量，固定1字节，标识着每一个DB数据的开始。
2. db\_number则是对应的db的index。
3. key\_value\_pairs就是该数据库中所有的键值对。

key\_value\_pairs里面的键值对分为不带过期时间的键值对和带有过期时间的键值对。不带过期时间的键值对格式如下：



1. TYPE固定为1字节。TYPE实际上是数据数据在数据库中以什么类型的对象进行存储。有了这个信息，redis就知道要怎么读取key和value。



1. key\value就是键值对。key一定是字符串，但是value的数据类型就不一定了。

带有过期时间的键值对格式如下:



1. EXPIRETIME\_MS为常量，固定1字节。
2. ms大小为8字节，单位为毫秒，是一个时间戳，就是键值对的过期时间。
3. 其他字段就和上面一样了。

具体对于RDB文件的解析过程，参见关键函数中的rdbLoad。

#### 关键函数

##### loadDataFromDisk

入参：无。

出参：无。

这个函数是读取AOF文件和RDB文件的入口，在main函数中被调用。函数判定如果开启了AOF持久化，就不会读取RDB文件。

##### rdbLoad

入参filename：RDB文件名称。

出参：正常返回REDIS\_OK，异常返回REDIS\_ERR。

先从文件中读入9个字节，实际上就是读入REDIS和data\_version进行校验，通过之后才开始载入RDB文件数据。

while循环当中首先读type，代码上写的是读取type，但实际上就是读取一个字节的数据。因为根据上面的格式，如果有数据的话，最先读到的应该是SELECTDB。然后格式不断的读取数据，直到遇到EOF，循环结束。

最后计算检验和，不匹配的话直接调用exit(1)退出子进程。把异常交给父进程处理。

## AOF持久化

与RDB持久化不同之处在于，AOF是保存数据库执行的写命令来记录数据库状态。AOF机制会将所有写命令写入AOF文件。不同于RDB文件是二进制文件，AOF是一个文本文件，可以直接看到保存的所有写命令。

### 相关配置项

appendonly yes //默认为no。

appendfilename [AOF文件名称]

appendfsync always //同步持久化，每次执行写命令，就会将写命令记录到磁盘，完整性最好但是性能最差。

appendfsync everysec //（默认值）异步持久化，每一秒将执行过的写命令记录到磁盘，出现异常的时候会有一定的数据丢失。

appendfsync no //将记录到磁盘的时机交给操作系统，linux默认是30秒将缓冲区的数据写入磁盘。

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

这两个配置项的作用是控制AOF rewrite机制的触发条件，如果当前的AOF文件超过上次rewrite之后AOF文件一倍大小，且当前AOF文件大于64MB的时候触发AOF rewrite。

### AOF命令追加

上面的配置可以看出，一般情况下redis不会每次执行写命令的时候就将命令写入AOF文件中。而是将写命令追加到redisServer结构中的aof\_buf中。

#### 关键函数

##### feedAppendOnlyFile

入参cmd：需要追加的命令。

入参dictid：执行命令使用的db id。

入参argv：

入参argc：

出参：无。

1. 函数调用点在propagate，每次命令处理完成之后由该函数决定是否要将命令传播到AOF文件和slave机器。
2. 校验待追加命令所对应的db id和当前AOF文件所对应的db id是否相同，如果不同需要先加入一条select命令。
3. 对于EXIPRE或者SETEX等涉及到过期时间相关的命令，需要进行特殊处理。
4. 其他命令不需要翻译，就将命令和参数组合成一个buf。
5. 将buf的内容写入aof\_buf中。

### AOF文件写入

redis通过serverCron循环来执行AOF文件写入的操作。redis并没有把AOF文件写入过程放在执行命令的过程中，而是等待redis命令执行完成之后且在执行下一个命令之前判断是否要将aof缓存中的内容写入AOF文件。

#### 关键函数

##### flushAppendOnlyFile

入参fore：是否为强制写入。从调用点可以看出，只有在stopAppendOnly的时候force才会为1。其他时候force都为0。

出参：无。

1. 如果AOF的写入策略为everysec，且当前已有sync操作在后台执行（这里在后台执行的sync操作可以是aof写入也可以是aof的rewrite操作），且之前没有推迟AOF写入操作，那么推迟当前的写入操作。
2. 如果之前已经推迟过写入操作，且推迟时间超出了2s，那么就一定会执行AOF写入操作。
3. 写入完成之后，如果aof缓存不超过4k的话，就重用这个缓存。否则会释放这个缓存，这应该是处于节省内存空间的考虑。
4. 如果策略是everysec，当前时间已经超过了1s且当前没有sync操作在后台执行，则调用aof\_background\_fsync创建一个后台写入事件。

### AOF文件读取

因为AOF文件中保存的是具体的redis指令，对于redis服务器来说，必须要有redisCli才能够处理具体的redis指令。因此创建了一个不带网络连接的redisCli来专门处理AOF文件中的redis指令。

#### 关键函数

##### loadAppendOnlyFile

入参filename：AOF文件名。

出参：正常REDIS\_OK，异常REDIS\_ERR。

1. 创建一个fakeClient，专门用来处理AOF文件中的命令。
2. AOF文件中\*后面的数字表示命令一共有几个参数。每个参数前面有$跟着数字，这个数字表示该参数一共有几个字符。\*数字、$数字、参数之间使用/r/n进行分隔。
3. 每次根据上面格式对AOF文件命令解析之后，通过fakeClient来执行命令。重复这个过程直到AOF文件中所有的命令全部读取完成。

### AOF重写

AOF文件记录的是redis的写命令，随着时间推移，文件大小会不断的增大。而且文件中会存在很多无效的命令，比如时间1对A进行了set操作，时间2又对A进行了set操作。此时时间1对A进行的set操作就没有意义了。因此需要通过AOF重写机制来控制AOF文件的大小。

redis通过BGREWRITEAOF命令和上面章节中说明的配置项两种方式来触发AOF重写机制。AOF重写机制并不会对当前AOF文件进行分析或者修改，而是通过数据库当前状态来决定需要记录哪些命令。

比如对于key A，AOF文件中存在5条对于其的操作。那么AOF重写的时候，只需要读取当前redis中key A所对应的值即可，不需要具体分析5条操作是什么，这样就可以用一条命令代替5条命令。

redis为了避免AOF重写阻塞服务，因此通过创建子进程的方式来执行AOF重写服务。但是子进程重写的过程中，主进程还在处理请求，这样就会导致出现最终生成的AOF文件和数据库中的数据不一致。因此redis使用AOF重写缓冲区来保存AOF重写过程中的写命令。在上面feedAppendOnlyFile函数里，就会根据情况，调用aofRewriteBufferAppend将命令添加到AOF重写缓存中。然后当AOF重写任务执行结束之后，由父进程将AOF重写缓冲中的数据写入AOF重写文件中。

#### 关键函数

##### rewriteAppendOnlyFileBackground

入参：无。

出参：正常REDIS\_OK，异常REDIS\_ERR。

1. 有AOF重写任务还未结束，退出。
2. 创建子进程调用rewriteAppendOnlyFile来执行重写任务。成功exit(0)，失败exit(1)。
3. 父进程记录重写开始的时间，并关闭哈希表的rehash功能。

##### rewriteAppendOnlyFile

入参filename：上面函数中输入的临时文件名。

出参：正常REDIS\_OK，异常REDIS\_ERR。

1. 创建一个用于保存重写内容的临时文件，临时文件名和函数入参的文件名不一样。
2. REDIS\_AOF\_AUTOSYNC\_BYTES来限制每次写入的字节数，防止缓存中积累的命令太多。当写入的命令长度超过这个限制时，系统会调用一次fsync将数据写入磁盘。
3. 遍历数据库，每个数据库的命令之前加入一个select命令。
4. 根据数据库中所有value的类型来决定生成什么样的写命令。
5. 所有命令都写入AOF文件之后就给文件重命名为入参文件名。但入参文件名其实还是一个临时文件名，之所以这样是因为后面还有一个将AOF缓冲数据写入AOF重写文件中的过程。

##### aofRewriteBufferAppend

入参s：指向需要加入重写缓存的字符串首部的指针。

入参len：需要加入重写缓存的字符串长度。

出参：无。

1. 重写缓存是一个链表，首先判断链表中是否已有数据。如果没有，新建新一个链表节点用来存储数据。
2. 如果已经有数据，判断链表最后一个节点的剩余空间是否能够保存待加入数据。如果够的话直接将数据写入。不够的话就新建一个链表节点。
3. 新建的链表节点是加入链表尾部的。

##### aofRewriteBufferWrite

入参fd：AOF重写文件的fd。

出参：写入AOF重写文件的数据总长度。

整个函数很简单，就是遍历AOF重写缓存链表，将链表中每个节点的数据写入重写文件中。