**RDB**

flushall save bgsave 命令 和 配置的自动存盘配置 会触发执行写rdb文件。

自动触发配置如： save 300 10， 表示满足 TIME\_NOW – lastsave > 300 并且 dirty >= 10 时 会触发自动存盘。可以配置多个save条件，只要满足一条即可触发存盘。

最终调用 rdbSave 保存到db文件中，默认保存到 dump.rdb 文件中。

bgsave会fork子进程去保存数据。

rdbSave(dump.rdb) 执行时，写将数据写入到临时文件 temp-PID.rdb 中，最后在rename成dump.rdb。使用

fflush(fp)

fsync(fileno(fp))

确保文件缓冲区数据写回到磁盘上。

rdbSave -> rdbSaveRio

bgsave情况下，在时钟函数serverCron中，如果rdb\_child\_pid存在的话，会wait3等待进程结束，最终调用 backgroundSaveDoneHandler 函数处理rdb文件写入完成后续处理。

普通写rdb是写入文件的，因此会调用 backgroundSaveDoneHandlerDisk。主要是在成功后更新dirty和记录最近一次写rdb的时间。

pid = wait3(&statloc,WNOHANG,NULL) **WNOHANG:**return immediately if no child has exited. If **WNOHANG** was specified in *options* and there were no children in a waitable state, then **waitid**() returns 0 immediately.

多个bgsave命令时如何处理？当有正在进行中的子进程在save时，如果有bgsave命令时，会设置下 rdb\_bgsave\_scheduled 标记，在timer函数中只有检查到没有子进程存在时会调度进行bgsave执行。

删除文件使用 unlink 系统api。

**AOF**

通过配置文件 appendonly yes 打开aof功能。

[1] feedAppendOnlyFile将client命令写入aof缓冲区 aof\_buf 。当命令执行的db与当前aof写入的db（aof\_selected\_db）不一致时要先写一个select dbid命令。

[2] flushAppendOnlyFile 将aof\_buf在时钟函数serverCron中会被定时写回文件中。注意这里写回文件只是调用write写入到系统的磁盘缓冲区。最终缓冲区中的内容同步到磁盘依赖 aof\_fsync 的策略。

AOF\_FSYNC\_ALWAYS write完直接同步到磁盘中 aof\_fsync -> fdatasync

AOF\_FSYNC\_EVERYSEC 定时同步到磁盘中 aof\_background\_fsync -> bioCreateBackgroundJob。使用 aof\_last\_fsync 变量来控制刷新频率。（Redis有BIO\_NUM\_OPS=2个后台线程，执行后台任务，包括这里的aof内容写回磁盘。使用pthread\_cond\_wait pthread\_cond\_signal 控制线程执行。具体见bio模块。）

AOF\_FSYNC\_NO 依赖系统的同步磁盘机制

该函数被调用时机：

aof\_flush\_postponed\_start != 0 当前有bio在同步磁盘缓存到磁盘文件时，aof的write会被延期，这里会检查时间间隔尝试下一次的write。

aof\_last\_write\_status == C\_ERR 上一次write失败了。在尝试调用一次。

beforeSleep中被调用，定时调用。

[3] aof rewrite 机制（rewriteAppendOnlyFileBackground）

Client命令bgrewriteaof主动触发。

Cilent通过配置命令修改配置，开启appendonly yes后也会触发

定时触发，需要的参数：

aof\_rewrite\_perc 百分比，默认100

aof\_rewrite\_min\_size aof文件大小阈值，默认64M

aof\_current\_size 当前aof文件大小，是一个始终增长的值

aof\_rewrite\_base\_size 是上一次rewrite后的aof文件大小。

growth = (server.aof\_current\_size\*100/ aof\_rewrite\_base\_size) – 100

=(server.aof\_current\_size-aof\_rewrite\_base\_size) \* 100 / aof\_rewrite\_base\_size

growth >= aof\_rewrite\_perc 时即增长率超过100%（翻一倍时）会触发重写。

如果当前有子进程正在rewrite aof，此时命令会额外在写到rewrite\_buf 中（aofRewriteBufferAppend）。Rewrite是将内存各个db遍历重新写成一个aof文件，然后替换掉原来的aof文件。注意开启子进程后，实际db中的内容已经包含了aof\_buf中的命令了，但是不会包含rewrite\_buf中的内容。

父进程会调用 aofChildWriteDiffData 将aof重写期间rewritebuf中的数据通过管道发送给子进程。子进程在重写期间会调用 aofReadDiffFromParent获取新增的命令，保存到aof\_child\_diff 变量中。子进程在db内容重写结束后，将aof\_child\_diff 内容也一起写入到aof文件中。最后会重命名成另一个tmp文件。

时钟主逻辑会探测到aof子进程执行完毕，调用 backgroundRewriteDoneHandler 处理后续的逻辑。主要是rename成配置的aof文件名。注意：

系统rename 对文件大小和文件是否被打开的影响？目前测试两个大文件A,B，先打开文件，在rename A->B, 然后在从打开的文件描述符读，依然能读到原来A，B对应的内容。原因？？？

rdb和aof rewrite时格式差别？

Rdb: type[int] key[string = len + str] value[根据不同类型write]

Aof文件 根据不同类型write，基本已可读的文本形式写入db内容。

**Repl**

主从配置 slaveof xxx 完成配置。最终调用 slaveofCommand。开启cluster后，主从模式无效。设置masterhost 和 masterport属性，设置repl\_state=REPL\_STATE\_CONNECT。

在repl的定时timer函数replicationCron中，如果处于state\_connect状态下，会调用connectWithMaster连接master。注意是同步连接。此时repl\_state=REPL\_STATE\_CONNECTING。注意连接成功后会注册 可读可写事件，对应回调函数为syncWithMaster。因为注册了可写事件，因此下一次事件循环时会调用syncWithMaster。此函数和 sendSynchronousCommand 结合，调用发送和接受数据完成一系列初始的数据通讯。

连接成功后会进行一次全量同步。收到master的应答后，重新注册read事件，使用readSyncBulkPayload 读取数据。

Master从 收到PSYNC开始执行，syncCommand。设置此client的replstate=SLAVE\_STATE\_WAIT\_BGSAVE\_START。加入到slaves列表中。此时master存在3中情况：

1. 没有后台rdb进程存在，master启动rdb进程。并且将所有START状态的slave设置为SLAVE\_STATE\_WAIT\_BGSAVE\_END
2. 存在rdb进程，并且是rdb\_child\_type=RDB\_CHILD\_TYPE\_DISK类型类型。

查找一个当前正在等待bgsave结束的slave，如果找不到，则等待下一次的BGSAVE。

如果当前master正在执行正常的bgsave，但是有存在不需要同步数据的slave，此时因为找不到一个合适的slave，因此只能等待下一个bgsave。否则保持和找到的一个slave状态一致，都等待bgsave结束后进行后续的处理。

1. 存在rdb进程，rdb\_child\_type == RDB\_CHILD\_TYPE\_SOCKET类型，此种类型下只能等待下一次的BGSAVE。

上面等待下一个BGSAVE是因为slave发送PSYNC后，如果master处于一些特殊状态下，例如已经bgsave时，master不会回复slave，导致slave超时后，slave会重新和master认证并在次发送PSYNC命令。

下面针对slave请求psync触发bgsave处理。先考虑全量同步。

Master为slave启动bgsave后会创建一个repl\_backlog。用于保存在bgsave期间的命令。在master启动子进程bgsave期间，用于执行某个cmd后，会保存到repl\_backlog缓冲区中，该缓冲区的数据是为了发送给所有 SLAVE\_STATE\_WAIT\_BGSAVE\_START 状态的slave的。而对于其他slave，则直接发送命令同步给所有其他slave。

此时master上的slave连接处于 WAIT\_BGSAVE\_END 状态，而slave实例则在等待接受数据。

Master检测到bgsave完成，backgroundSaveDoneHandlerDisk调用到updateSlavesWaitingBgsave。会遍历处理所有处于 SLAVE\_STATE\_WAIT\_BGSAVE\_START 或是 SLAVE\_STATE\_WAIT\_BGSAVE\_END 状态的slave连接。对于前者，会再次启动一个bgsave。

对于后者的处理：

设置replstate = SLAVE\_STATE\_SEND\_BULK

打开rdb文件，添加可写事件，使用sendBulkToSlave发送数据。

发送完成后设置slave的replstate = SLAVE\_STATE\_ONLINE，移除write事件。

再次添加write事件，sendReplyToClient。这里主要是发送master在bgsave期间的client命令，这些命令被缓存到了reply中。(slave连接没有添加write事件时，这些reply都会被缓存起来，到这里时才会真正发送)

Repl\_backlog是一个环形缓存区，使用的位置标记始终增长，实际使用时取模。

slave全量同步过程中和master断开连接了：cancelReplicationHandshake -> replicationAbortSyncTransfer。移除read事件，关闭socket，关闭并删除文件，repl\_state状态重新切换成 REPL\_STATE\_CONNECT。后续需要重新进行一次完整同步。

slave全量同步完成后和master连接断开了：全量同步完成后 slave端将repl\_state设置为REPL\_STATE\_CONNECTED，master将slave的repl\_state设置为SLAVE\_STATE\_ONLINE。Timeout后，关闭连接。

slave端：server. repl\_transfer\_size是要传输的rdb文件总长度；server.master->reploff = server.repl\_master\_initial\_offset，此值是master的master\_repl\_offset，即请求全量同步时获取到的这个offset为那个时间点master的全部数据记录的一个标记点，例如当时是OFF\_A1。Master开始生成rdb文件过程中，如果有client命令，会增加master端的master\_repl\_offset，例如已经变成了OFF\_A2，同时也会把一份数据加入到此slave连接的缓冲区中。在rdb文件同步完成后同步此缓冲区内数据，即把A2-A1的内容发送给slave，此时slave使用 readQueryFromClient 处理接受到的数据，内部会更新slave自己的同步偏移量值reploff(见replicationCreateMasterClient函数。)。因此当slave和master连接断开后，可以使用slave端保存的reploff发给master，由master判断此记录标记点是否能够continue传输。

master端：server.repl\_master\_initial\_offset 在有slave后会使用。这个值是一直递增的。