主从复制过程中，主节点根据从节点发来的命令执行相应的操作。结合上一章中讲解的从节点在主从复制中的流程，本章以及下一篇文章讲解一下主节点在主从复制过程中的流程。

本章主要介绍完全重同步流程。

**一：从节点建链和握手**

从节点在向主节点发起TCP建链，以及复制握手过程中，主节点一直把从节点当成一个普通的客户端处理。也就是说，不为从节点保存状态，只是收到从节点发来的命令进而处理并回复罢了。

从节点在握手过程中第一个发来的命令是”PING”，主节点调用redis.c中的pingCommand函数处理，只是回复字符串”+PONG”即可。

接下来从节点向主节点发送”AUTH xxx”命令进行认证，主节点调用redis.c中的authCommand函数进行处理，该函数的代码如下：

void authCommand**(**redisClient **\***c**)** **{**

**if** **(!**server**.**requirepass**)** **{**

addReplyError**(**c**,**"Client sent AUTH, but no password is set"**);**

**}** **else** **if** **(!**time\_independent\_strcmp**(**c**->**argv**[**1**]->**ptr**,** server**.**requirepass**))** **{**

c**->**authenticated **=** 1**;**

addReply**(**c**,**shared**.**ok**);**

**}** **else** **{**

c**->**authenticated **=** 0**;**

addReplyError**(**c**,**"invalid password"**);**

**}**

**}**

server.requirepass根据配置文件中"requirepass"的选项进行设置，保存了Redis实例的密码。如果该值为NULL，说明本Redis实例不需要密码。这种情况下，如果从节点发来”AUTH xxx”命令，则回复给从节点错误信息："Client sent AUTH, but no password is set"。

接下来，对从节点发来的密码和server.requirepass进行比对，如果匹配成功，则回复给客户端”+OK”，否则，回复给客户端错误信息："invalid password"。

从节点接下来发送"REPLCONF listening-port <port>"和"REPLCONF capa eof"命令，告知主节点自己的监听端口和“能力”。主节点通过replication.c中的replconfCommand函数处理这些命令，代码如下：

void replconfCommand**(**redisClient **\***c**)** **{**

int j**;**

**if** **((**c**->**argc **%** 2**)** **==** 0**)** **{**

*/\* Number of arguments must be odd to make sure that every*

*\* option has a corresponding value. \*/*

addReply**(**c**,**shared**.**syntaxerr**);**

**return;**

**}**

*/\* Process every option-value pair. \*/*

**for** **(**j **=** 1**;** j **<** c**->**argc**;** j**+=**2**)** **{**

**if** **(!**strcasecmp**(**c**->**argv**[**j**]->**ptr**,**"listening-port"**))** **{**

long port**;**

**if** **((**getLongFromObjectOrReply**(**c**,**c**->**argv**[**j**+**1**],**

**&**port**,NULL)** **!=** REDIS\_OK**))**

**return;**

c**->**slave\_listening\_port **=** port**;**

**}** **else** **if** **(!**strcasecmp**(**c**->**argv**[**j**]->**ptr**,**"capa"**))** **{**

*/\* Ignore capabilities not understood by this master. \*/*

**if** **(!**strcasecmp**(**c**->**argv**[**j**+**1**]->**ptr**,**"eof"**))**

c**->**slave\_capa **|=** SLAVE\_CAPA\_EOF**;**

**}**

...

**}**

addReply**(**c**,**shared**.**ok**);**

**}**

“REPLCONF”命令的格式为"REPLCONF <option> <value> <option> <value> ..."。因此，如果命令参数是偶数，说明命令格式错误，回复给从节点客户端错误信息："-ERR syntax error"；

如果从节点发来的是"REPLCONF listening-port <port>"命令，则从中取出<port>信息，保存在客户端的slave\_listening\_port属性中，记录从节点客户端的监听端口，主节点使用从节点的IP地址和监听端口，作为从节点的身份标识；

如果从节点发来的是"REPLCONF capa eof"命令，则将从节点客户端的能力属性slave\_capa增加SLAVE\_CAPA\_EOF标记，表示该从节点支持无硬盘复制。目前为止，仅有这一种能力标记。

**二：完全重同步时，从节点状态转换**

接下来，从节点会向主节点发送”SYNC”或”PSYNC”命令，请求进行完全重同步或者部分重同步。

主节点收到这些命令之后，如果是需要进行完全重同步，则开始在后台进行RDB数据转储（将数据保存在本地文件或者直接发给从节点）。同时，在前台接着接收客户端发来的命令请求。为了使从节点能与主节点的状态保持一致，主节点需要将这些命令请求缓存起来，以便在从节点收到主节点RDB数据并加载完成之后，将这些累积的命令流发送给从节点。

从收到从节点的”SYNC”或”PSYNC”命令开始，主节点开始为该从节点保存状态。从此时起，站在主节点的角度，从节点的状态会发生转换。

主节点为从节点保存的状态记录在客户端结构redisClient中的replstate属性中。从主节点的角度看，从节点需要经历的状态分别是：REDIS\_REPL\_WAIT\_BGSAVE\_START、REDIS\_REPL\_WAIT\_BGSAVE\_END、REDIS\_REPL\_SEND\_BULK和REDIS\_REPL\_ONLINE。

当主节点收到从节点发来的”SYNC”或”PSYNC”命令，并且需要完全重同步时，将从节点的状态置为REDIS\_REPL\_WAIT\_BGSAVE\_START，表示该从节点等待主节点后台RDB数据转储的开始；

接下来，当主节点开始在后台进行RDB数据转储时，将从节点的状态置为REDIS\_REPL\_WAIT\_BGSAVE\_END，表示该从节点等待主节点后台RDB数据转储的完成；

主节点在后台进行RDB数据的转储的时候，依然可以接收客户端发来的命令请求，为了能使从节点与主节点保持一致，主节点需要将客户端发来的命令请求，保存到从节点客户端的输出缓存中，这就是所谓的为从节点累积命令流。当从节点的复制状态变为REDIS\_REPL\_ONLINE时，就可以将这些累积的命令流发送个从节点了。

如果主节点在进行后台RDB数据转储时，使用的是有硬盘复制的方式（将RDB数据保存在本地文件），则RDB数据转储完成时，将从节点的状态置为REDIS\_REPL\_SEND\_BULK，表示接下来要将本地的RDB文件发送给客户端了；当所有的RDB数据发送完成后，将从节点的状态置为REDIS\_REPL\_ONLINE，表示可以向从节点发送累积的命令流了。

如果主节点在进行后台RDB数据转储时，使用的是无硬盘复制的方式（将RDB数据直接通过网络发送给从节点），则RDB数据发送完成之后，收到从节点发来的第一个"REPLCONF ACK <offset>"后，就将从节点的状态置为REDIS\_REPL\_ONLINE，表示可以向从节点发送累积的命令流了。

无硬盘复制的RDB数据转储，之所以要等到收到第一个"REPLCONF ACK <offset>"后，才能将从节点的状态置为REDIS\_REPL\_ONLINE。结合注释：https://github.com/antirez/redis/commit/bb7fea0d5ca7b3a53532338e8654e409014c1194，个人理解是因为无硬盘复制的RDB数据，不同于有硬盘复制的RDB数据，它没有长度标记，从节点每次从socket读取的数据量都是固定的(4k)。下面是从节点读取RDB数据时调用的readSyncBulkPayload函数中，每次read之前，计算要读取多少字节的代码，usemark为1表示无硬盘复制：

/\* Read bulk data \*/

**if** **(**usemark**)** **{**

readlen **=** **sizeof(**buf**);**

**}** **else** **{**

left **=** server**.**repl\_transfer\_size **-** server**.**repl\_transfer\_read**;**

readlen **=** **(**left **<** **(**signed**)sizeof(**buf**))** **?** left **:** **(**signed**)sizeof(**buf**);**

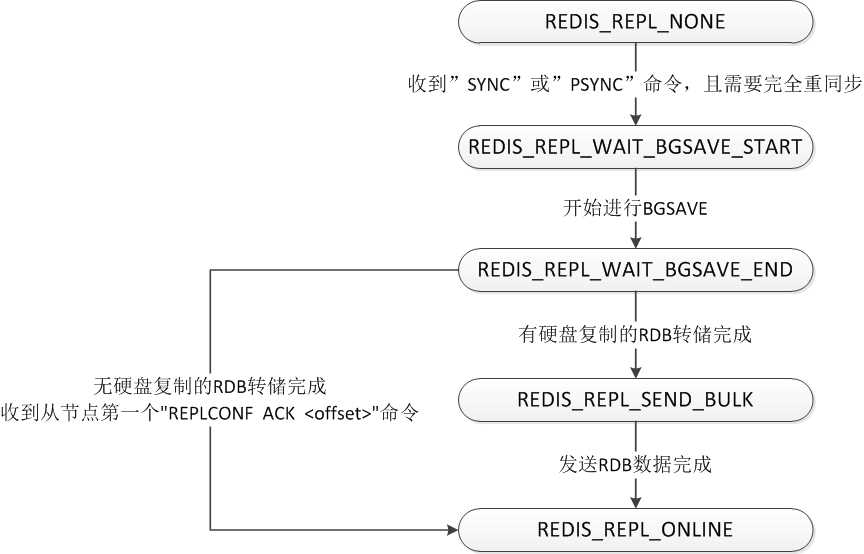
**}**

因此，主节点在通过socket发送完RDB数据之后，如果接着就使用该socket发送累积的命令流，则从节点读取数据时，最后读到的数据中，有可能一部分是RDB数据，剩下的部分是累积的命令流。而此时从节点接下来就要加载RDB数据，无法处理这部分累积的命令流，只能丢掉，这就造成了主从数据库状态不一致了。

因此，等到从节点发来第一个"REPLCONF ACK <offset>"消息之后，此时能保证从节点已经加载完RDB数据，可以接收累积的命令流了。因此，这时才可以将从节点的复制状态置为REDIS\_REPL\_ONLINE。

有硬盘复制的RDB数据，因为数据头中包含了数据长度，因此从节点知道总共需要读取多少RDB数据。因此，有硬盘复制的RDB数据转储，在发送完RDB数据之后，就可以立即将从节点复制状态置为REDIS\_REPL\_ONLINE。

根据以上的描述，总结从节点的状态转换图如下：



**三：SYNC或PSYNC命令的处理**

主节点收到从节点发来的”SYNC”或”PSYNC”命令后，如果需要为该从节点进行完全重同步，将从节点的复制状态置为REDIS\_REPL\_WAIT\_BGSAVE\_START。开始在后台进行RDB数据转储时，则将复制状态置为REDIS\_REPL\_WAIT\_BGSAVE\_END。

这里有一个问题，考虑这样一种情形：当主节点收到从节点A的”SYNC”或”PSYNC”命令后，要为该从节点进行完全重同步时，在将A的复制状态变为REDIS\_REPL\_WAIT\_BGSAVE\_END时刻起，主节点在前台接收客户端的命令请求，将该命令情求保存到A的输出缓存中，并在后台进行有硬盘复制的RDB数据转储。

在后台进行有硬盘复制的RDB数据转储尚未完成时，如果又有新的从节点B发来了”SYNC”或”PSYNC”命令，同样需要完全重同步。此时主节点后台正在进行RDB数据转储，而且已经为A缓存了命令流。那么从节点B完全可以重用这份RDB数据，而无需再执行一次RDB转储了。而且将A中的输出缓存复制到B的输出缓存中，就能保证B的数据库状态也能与主节点一致了。因此，直接将B的复制状态直接置为REDIS\_REPL\_WAIT\_BGSAVE\_END，等到后台RDB数据转储完成时，直接将该转储文件同时发送给从节点A和B即可。

但是如果此刻主节点进行的是无硬盘复制的RDB数据转储，这意味着主节点是直接将RDB数据通过socket发送给从节点A的，从节点B也就无法重用RDB数据了，因此需要再次执行一次BGSAVE操作。

下面就是主节点收到”SYNC”或”PSYNC”命令的处理函数syncCommand的代码：

void syncCommand**(**redisClient **\***c**)** **{**

*/\* ignore SYNC if already slave or in monitor mode \*/*

**if** **(**c**->**flags **&** REDIS\_SLAVE**)** **return;**

*/\* Refuse SYNC requests if we are a slave but the link with our master*

*\* is not ok... \*/*

**if** **(**server**.**masterhost **&&** server**.**repl\_state **!=** REDIS\_REPL\_CONNECTED**)** **{**

addReplyError**(**c**,**"Can't SYNC while not connected with my master"**);**

**return;**

**}**

*/\* SYNC can't be issued when the server has pending data to send to*

*\* the client about already issued commands. We need a fresh reply*

*\* buffer registering the differences between the BGSAVE and the current*

*\* dataset, so that we can copy to other slaves if needed. \*/*

**if** **(**listLength**(**c**->**reply**)** **!=** 0 **||** c**->**bufpos **!=** 0**)** **{**

addReplyError**(**c**,**"SYNC and PSYNC are invalid with pending output"**);**

**return;**

**}**

redisLog**(**REDIS\_NOTICE**,**"Slave %s asks for synchronization"**,**

replicationGetSlaveName**(**c**));**

*/\* Try a partial resynchronization if this is a PSYNC command.*

*\* If it fails, we continue with usual full resynchronization, however*

*\* when this happens masterTryPartialResynchronization() already*

*\* replied with:*

*\**

*\* +FULLRESYNC <runid> <offset>*

*\**

*\* So the slave knows the new runid and offset to try a PSYNC later*

*\* if the connection with the master is lost. \*/*

**if** **(!**strcasecmp**(**c**->**argv**[**0**]->**ptr**,**"psync"**))** **{**

**if** **(**masterTryPartialResynchronization**(**c**)** **==** REDIS\_OK**)** **{**

server**.**stat\_sync\_partial\_ok**++;**

**return;** */\* No full resync needed, return. \*/*

**}** **else** **{**

char **\***master\_runid **=** c**->**argv**[**1**]->**ptr**;**

*/\* Increment stats for failed PSYNCs, but only if the*

*\* runid is not "?", as this is used by slaves to force a full*

*\* resync on purpose when they are not albe to partially*

*\* resync. \*/*

**if** **(**master\_runid**[**0**]** **!=** '?'**)** server**.**stat\_sync\_partial\_err**++;**

**}**

**}** **else** **{**

*/\* If a slave uses SYNC, we are dealing with an old implementation*

*\* of the replication protocol (like redis-cli --slave). Flag the client*

*\* so that we don't expect to receive REPLCONF ACK feedbacks. \*/*

c**->**flags **|=** REDIS\_PRE\_PSYNC**;**

**}**

*/\* Full resynchronization. \*/*

server**.**stat\_sync\_full**++;**

*/\* Setup the slave as one waiting for BGSAVE to start. The following code*

*\* paths will change the state if we handle the slave differently. \*/*

c**->**replstate **=** REDIS\_REPL\_WAIT\_BGSAVE\_START**;**

**if** **(**server**.**repl\_disable\_tcp\_nodelay**)**

anetDisableTcpNoDelay**(NULL,** c**->**fd**);** */\* Non critical if it fails. \*/*

c**->**repldbfd **=** **-**1**;**

c**->**flags **|=** REDIS\_SLAVE**;**

listAddNodeTail**(**server**.**slaves**,**c**);**

*/\* CASE 1: BGSAVE is in progress, with disk target. \*/*

**if** **(**server**.**rdb\_child\_pid **!=** **-**1 **&&**

server**.**rdb\_child\_type **==** REDIS\_RDB\_CHILD\_TYPE\_DISK**)**

**{**

*/\* Ok a background save is in progress. Let's check if it is a good*

*\* one for replication, i.e. if there is another slave that is*

*\* registering differences since the server forked to save. \*/*

redisClient **\***slave**;**

listNode **\***ln**;**

listIter li**;**

listRewind**(**server**.**slaves**,&**li**);**

**while((**ln **=** listNext**(&**li**)))** **{**

slave **=** ln**->**value**;**

**if** **(**slave**->**replstate **==** REDIS\_REPL\_WAIT\_BGSAVE\_END**)** **break;**

**}**

*/\* To attach this slave, we check that it has at least all the*

*\* capabilities of the slave that triggered the current BGSAVE. \*/*

**if** **(**ln **&&** **((**c**->**slave\_capa **&** slave**->**slave\_capa**)** **==** slave**->**slave\_capa**))** **{**

*/\* Perfect, the server is already registering differences for*

*\* another slave. Set the right state, and copy the buffer. \*/*

copyClientOutputBuffer**(**c**,**slave**);**

replicationSetupSlaveForFullResync**(**c**,**slave**->**psync\_initial\_offset**);**

redisLog**(**REDIS\_NOTICE**,**"Waiting for end of BGSAVE for SYNC"**);**

**}** **else** **{**

*/\* No way, we need to wait for the next BGSAVE in order to*

*\* register differences. \*/*

redisLog**(**REDIS\_NOTICE**,**"Waiting for next BGSAVE for SYNC"**);**

**}**

*/\* CASE 2: BGSAVE is in progress, with socket target. \*/*

**}** **else** **if** **(**server**.**rdb\_child\_pid **!=** **-**1 **&&**

server**.**rdb\_child\_type **==** REDIS\_RDB\_CHILD\_TYPE\_SOCKET**)**

**{**

*/\* There is an RDB child process but it is writing directly to*

*\* children sockets. We need to wait for the next BGSAVE*

*\* in order to synchronize. \*/*

redisLog**(**REDIS\_NOTICE**,**"Waiting for next BGSAVE for SYNC"**);**

*/\* CASE 3: There is no BGSAVE is progress. \*/*

**}** **else** **{**

**if** **(**server**.**repl\_diskless\_sync **&&** **(**c**->**slave\_capa **&** SLAVE\_CAPA\_EOF**))** **{**

*/\* Diskless replication RDB child is created inside*

*\* replicationCron() since we want to delay its start a*

*\* few seconds to wait for more slaves to arrive. \*/*

**if** **(**server**.**repl\_diskless\_sync\_delay**)**

redisLog**(**REDIS\_NOTICE**,**"Delay next BGSAVE for SYNC"**);**

**}** **else** **{**

*/\* Target is disk (or the slave is not capable of supporting*

*\* diskless replication) and we don't have a BGSAVE in progress,*

*\* let's start one. \*/*

**if** **(**startBgsaveForReplication**(**c**->**slave\_capa**)** **!=** REDIS\_OK**)** **return;**

**}**

**}**

**if** **(**listLength**(**server**.**slaves**)** **==** 1 **&&** server**.**repl\_backlog **==** **NULL)**

createReplicationBacklog**();**

**return;**

**}**

在函数中，如果当前的客户端标志位中已经有REDIS\_SLAVE标记了，则直接返回；

如果当前Redis实例是其他主节点的从节点，并且该从节点的复制状态不是REDIS\_REPL\_CONNECTED，说明当前的从节点实例，还没有到接收并加载完其主节点发来的RDB数据的步骤，这种情况下，该从节点实例是不能为其下游从节点进行同步的，因此向其客户端回复错误信息，然后返回；

如果当前的客户端输出缓存中已经有数据了，说明在SYNC(PSYNC)命令之前的命令交互中，该Redis实例尚有回复信息还没有完全发送给该从节点客户端，这种情况下，向该从节点客户端回复错误信息，然后返回；

这是因为主节点接下来需要为该从节点进行后台RDB数据转储了，同时需要将前台接收到的其他客户端命令请求缓存到该从节点客户端的输出缓存中，这就需要一个完全清空的输出缓存，才能为该从节点保存从执行BGSAVE开始的命令流。因此，如果从节点客户端的输出缓存中尚有数据，直接回复错误信息。

在主节点收到从节点发来的SYNC(PSYNC)命令之前，主从节点之间的交互信息都是比较短的，因此，在网络正常的情况下，从节点客户端中的输出缓存应该是很容易就发送给该从节点，并清空的。

接下来开始处理PSYNC或者SYNC命令：

如果用户发来的是"PSYNC"命令，则首先调用masterTryPartialResynchronization尝试进行部分重同步，如果成功，则直接返回即可。部分重同步的细节会在下一篇文章中讲解；

如果不能为该从节点执行部分重同步，则接下来需要进行完全重同步了。首先如果用户发来的"SYNC"命令，则将REDIS\_PRE\_PSYNC标记增加到客户端标记中，表示该从节点客户端是老版本的Redis实例；接下来就准备进行完全重同步了，先增加server.stat\_sync\_full的值；

首先将该从节点客户端的复制状态置为REDIS\_REPL\_WAIT\_BGSAVE\_START，表示该从节点需要主节点进行BGSAVE；

如果server.repl\_disable\_tcp\_nodelay选项为真，则取消与从节点通信的socket描述符的TCP\_NODELAY选项；

将REDIS\_SLAVE标记记录到从节点客户端的标志位中，以标识该客户端为从节点客户端；

将该从节点客户端添加到列表server.slaves中；

接下来开始分情况处理：

情况1：如果当前已有子进程正在后台将RDB转储到本地文件，则轮训列表server.slaves，找到一个复制状态为REDIS\_REPL\_WAIT\_BGSAVE\_END的从节点客户端。

如果找到了一个这样的从节点客户端A，并且A的能力是大于当前从节点的。那么主节点为从节点A，在后台开始进行RDB数据转储时，同时会将前台收到的命令流缓存到从节点A的输出缓存中。因此当前发来SYNC(PSYNC)命令的从节点完全可以重用这份RDB数据，以及从节点A中缓存的命令流，而无需再执行一次RDB转储。等到本次BGSAVE完成之后，只需要将RDB文件发送给A以及当前从节点即可。

因此，找到这样的从节点A后，只要复制A的输出缓存中的内容到当前从节点的输出缓存中，然后调用replicationSetupSlaveForFullResync，将该从节点客户端的复制状态置为REDIS\_REPL\_WAIT\_BGSAVE\_END，然后向其发送"+FULLRESYNC"回复即可；

如果找不到这样的从节点客户端，则主节点需要在当前的BGSAVE操作完成之后，重新执行一次BGSAVE操作；

情况2：如果当前有子进程在后台进行RDB转储，但是是直接将RDB数据通过socket直接发送给了从节点。这种情况下，当前的从节点无法重用RDB数据，必须在当前的BGSAVE操作完成之后，重新执行一次BGSAVE操作；

情况3：如果当前没有子进程在进行RDB转储，并且当前的从节点客户端可以接受无硬盘复制的RDB数据。这种情况下，先暂时不进行BGSAVE，而是在定时函数replicationCron中在执行，这样可以等到更多的从节点，以减少执行BGSAVE的次数；

情况4：如果当前没有子进程在进行RDB转储，并且当前的从节点客户端只能接受有硬盘复制的RDB数据，则调用startBgsaveForReplication开始进行BGSAVE操作；

最后，如果当前的列表server.slaves长度为1，并且server.repl\_backlog为NULL，说明当前从节点客户端是该主节点实例的第一个从节点，因此调用createReplicationBacklog创建积压队列；

**四：开始BGSAVE操作**

由函数startBgsaveForReplication执行BGSAVE操作。在开始执行BGSAVE操作时，需要向从节点发送"+FULLRESYNC <runid> <offset>"信息，从节点收到该信息后，会保存主节点的运行ID，以及复制偏移的初始值，以便后续断链时可以进行部分重同步。

startBgsaveForReplication的代码如下：

int startBgsaveForReplication**(**int mincapa**)** **{**

int retval**;**

int socket\_target **=** server**.**repl\_diskless\_sync **&&** **(**mincapa **&** SLAVE\_CAPA\_EOF**);**

listIter li**;**

listNode **\***ln**;**

redisLog**(**REDIS\_NOTICE**,**"Starting BGSAVE for SYNC with target: %s"**,**

socket\_target **?** "slaves sockets" **:** "disk"**);**

**if** **(**socket\_target**)**

retval **=** rdbSaveToSlavesSockets**();**

**else**

retval **=** rdbSaveBackground**(**server**.**rdb\_filename**);**

*/\* If we failed to BGSAVE, remove the slaves waiting for a full*

*\* resynchorinization from the list of salves, inform them with*

*\* an error about what happened, close the connection ASAP. \*/*

**if** **(**retval **==** REDIS\_ERR**)** **{**

redisLog**(**REDIS\_WARNING**,**"BGSAVE for replication failed"**);**

listRewind**(**server**.**slaves**,&**li**);**

**while((**ln **=** listNext**(&**li**)))** **{**

redisClient **\***slave **=** ln**->**value**;**

**if** **(**slave**->**replstate **==** REDIS\_REPL\_WAIT\_BGSAVE\_START**)** **{**

slave**->**flags **&=** **~**REDIS\_SLAVE**;**

listDelNode**(**server**.**slaves**,**ln**);**

addReplyError**(**slave**,**

"BGSAVE failed, replication can't continue"**);**

slave**->**flags **|=** REDIS\_CLOSE\_AFTER\_REPLY**;**

**}**

**}**

**return** retval**;**

**}**

*/\* If the target is socket, rdbSaveToSlavesSockets() already setup*

*\* the salves for a full resync. Otherwise for disk target do it now.\*/*

**if** **(!**socket\_target**)** **{**

listRewind**(**server**.**slaves**,&**li**);**

**while((**ln **=** listNext**(&**li**)))** **{**

redisClient **\***slave **=** ln**->**value**;**

**if** **(**slave**->**replstate **==** REDIS\_REPL\_WAIT\_BGSAVE\_START**)** **{**

replicationSetupSlaveForFullResync**(**slave**,**

getPsyncInitialOffset**());**

**}**

**}**

**}**

*/\* Flush the script cache, since we need that slave differences are*

*\* accumulated without requiring slaves to match our cached scripts. \*/*

**if** **(**retval **==** REDIS\_OK**)** replicationScriptCacheFlush**();**

**return** retval**;**

**}**

参数mincapa，表示从节点的"能力"，也就是是否能接受无硬盘复制的RDB数据。如果选项server.repl\_diskless\_sync为真，并且参数mincapa中包含SLAVE\_CAPA\_EOF标记，说明可以为该从节点直接发送无硬盘复制的RDB数据，因此调用rdbSaveToSlavesSockets，直接在后台将RDB数据通过socket发送给所有状态为REDIS\_REPL\_WAIT\_BGSAVE\_START的从节点；

否则，调用rdbSaveBackground，在后台将RDB数据转储到本地文件；

如果rdbSaveToSlavesSockets或者rdbSaveBackground返回失败，说明创建后台子进程失败。需要断开所有处于REDIS\_REPL\_WAIT\_BGSAVE\_START状态的从节点的连接；

轮训列表server.slaves，找到所有处于状态REDIS\_REPL\_WAIT\_BGSAVE\_START的从节点。首先删除该从节点客户端标志位中的REDIS\_SLAVE标记，然后将其从server.slaves中删除；回复从节点错误信息，然后增加REDIS\_CLOSE\_AFTER\_REPLY标记到客户端标志位中，也就是回复完错误消息后，立即关闭与该从节点的连接；最后返回；

如果当前进行的是有硬盘复制的RDB转储，则轮训列表server.slaves，找到其中处于状态REDIS\_REPL\_WAIT\_BGSAVE\_START的从节点，调用replicationSetupSlaveForFullResync函数将其状态置为REDIS\_REPL\_WAIT\_BGSAVE\_END，并且发送"+FULLRESYNC <runid> <offset>"回复；

因为无硬盘复制的RDB数据转储，已经在rdbSaveToSlavesSockets中进行过该过程了，所以这里只处理有硬盘复制的情况。

最后，调用replicationScriptCacheFlush。

下面是函数replicationSetupSlaveForFullResync的代码：

int replicationSetupSlaveForFullResync**(**redisClient **\***slave**,** long long offset**)** **{**

char buf**[**128**];**

int buflen**;**

slave**->**psync\_initial\_offset **=** offset**;**

slave**->**replstate **=** REDIS\_REPL\_WAIT\_BGSAVE\_END**;**

*/\* We are going to accumulate the incremental changes for this*

*\* slave as well. Set slaveseldb to -1 in order to force to re-emit*

*\* a SLEECT statement in the replication stream. \*/*

server**.**slaveseldb **=** **-**1**;**

*/\* Don't send this reply to slaves that approached us with*

*\* the old SYNC command. \*/*

**if** **(!(**slave**->**flags **&** REDIS\_PRE\_PSYNC**))** **{**

buflen **=** snprintf**(**buf**,sizeof(**buf**),**"+FULLRESYNC %s %lld\r\n"**,**

server**.**runid**,**offset**);**

**if** **(**write**(**slave**->**fd**,**buf**,**buflen**)** **!=** buflen**)** **{**

freeClientAsync**(**slave**);**

**return** REDIS\_ERR**;**

**}**

**}**

**return** REDIS\_OK**;**

**}**

在从节点发来"PSYNC"或"SYNC"命令后，为从节点进行完全重同步时，立即调用该函数，更改从节点客户端的复制状态为REDIS\_REPL\_WAIT\_BGSAVE\_END；

首先设置从节点客户端的psync\_initial\_offset属性为参数offset。一般情况下，参数offset是由getPsyncInitialOffset函数得到，该函数返回主节点上当前的的复制偏移量。但是如果从节点客户端B是在主节点进行RDB转储时新连接到主节点的，并且它找到了一个可以复用RDB数据和输出缓存的从节点A，则需要使用A->psync\_initial\_offset为参数调用本函数。也就是说，B还需要复用A的初始复制偏移量；

然后设置从节点客户端复制状态为REDIS\_REPL\_WAIT\_BGSAVE\_END，表示等待RDB转储完成；从复制状态变为REDIS\_REPL\_WAIT\_BGSAVE\_END的时刻起，主节点就开始在该从节点客户端的输出缓存中，为从节点累积命令流了。因此，设置server.slaveseldb为-1，这样可以在开始累积命令流时，强制增加一条"SELECT"命令到客户端输出缓存中，以免第一条命令没有选择数据库；

如果该从节点发送的是PSYNC命令，则直接回复给从节点信息："+FULLRESYNC <runid> <offset>"，注意这里是直接调用的write发送的信息，而没有用到输出缓存。这是因为输出缓存此时只能用于缓存命令流。从节点收到该信息后，会保存主节点的运行ID，以及复制偏移的初始值，以便后续断链时可以进行部分重同步。

**五：为从节点累积命令流**

从主节点在为从节点执行BGSAVE操作的时刻起，准确的说是从节点的复制状态变为REDIS\_REPL\_WAIT\_BGSAVE\_END的时刻起，主节点就需要将收到的客户端命令请求，缓存一份到从节点的输出缓存中，也就是为从节点累积命令流。等到从节点状态变为REDIS\_REPL\_ONLINE时，就可以将累积的命令流发送给从节点了，从而保证了从节点的数据库状态能够与主节点保持一致。

前面提到过，主节点收到”SYNC”或”PSYNC”命令后，调用syncCommand时处理时，就需要保证从节点的输出缓存是空的，而且即使是需要回复从节点"+FULLRESYNC"时，也是调用write，将信息直接发送给从节点客户端，而没有使用客户端的输出缓存。这就是因为要使用客户端的输出缓存来为从节点累积命令流。

当主节点收到客户端发来的命令请求后，会调用call函数执行相应的命令处理函数。在call函数的最后，有下面的语句：

/\* Propagate the command into the AOF and replication link \*/

**if** **(**flags **&** REDIS\_CALL\_PROPAGATE**)** **{**

int flags **=** REDIS\_PROPAGATE\_NONE**;**

**if** **(**c**->**flags **&** REDIS\_FORCE\_REPL**)** flags **|=** REDIS\_PROPAGATE\_REPL**;**

**if** **(**c**->**flags **&** REDIS\_FORCE\_AOF**)** flags **|=** REDIS\_PROPAGATE\_AOF**;**

**if** **(**dirty**)**

flags **|=** **(**REDIS\_PROPAGATE\_REPL **|** REDIS\_PROPAGATE\_AOF**);**

**if** **(**flags **!=** REDIS\_PROPAGATE\_NONE**)**

propagate**(**c**->**cmd**,**c**->**db**->**id**,**c**->**argv**,**c**->**argc**,**flags**);**

**}**

上面的语句中，dirty表示在执行命令处理函数时，数据库状态是否发生了变化。只要dirty不为0，就会为flags增加REDIS\_PROPAGATE\_REPL和REDIS\_PROPAGATE\_AOF标记。从而调用propagate，该函数会调用replicationFeedSlaves将该命令传播给从节点。

replicationFeedSlaves函数的代码如下：

void replicationFeedSlaves**(**list **\***slaves**,** int dictid**,** robj **\*\***argv**,** int argc**)** **{**

listNode **\***ln**;**

listIter li**;**

int j**,** len**;**

char llstr**[**REDIS\_LONGSTR\_SIZE**];**

*/\* If there aren't slaves, and there is no backlog buffer to populate,*

*\* we can return ASAP. \*/*

**if** **(**server**.**repl\_backlog **==** **NULL** **&&** listLength**(**slaves**)** **==** 0**)** **return;**

*/\* We can't have slaves attached and no backlog. \*/*

redisAssert**(!(**listLength**(**slaves**)** **!=** 0 **&&** server**.**repl\_backlog **==** **NULL));**

*/\* Send SELECT command to every slave if needed. \*/*

**if** **(**server**.**slaveseldb **!=** dictid**)** **{**

robj **\***selectcmd**;**

*/\* For a few DBs we have pre-computed SELECT command. \*/*

**if** **(**dictid **>=** 0 **&&** dictid **<** REDIS\_SHARED\_SELECT\_CMDS**)** **{**

selectcmd **=** shared**.**select**[**dictid**];**

**}** **else** **{**

int dictid\_len**;**

dictid\_len **=** ll2string**(**llstr**,sizeof(**llstr**),**dictid**);**

selectcmd **=** createObject**(**REDIS\_STRING**,**

sdscatprintf**(**sdsempty**(),**

"\*2\r\n$6\r\nSELECT\r\n$%d\r\n%s\r\n"**,**

dictid\_len**,** llstr**));**

**}**

*/\* Add the SELECT command into the backlog. \*/*

**if** **(**server**.**repl\_backlog**)** feedReplicationBacklogWithObject**(**selectcmd**);**

*/\* Send it to slaves. \*/*

listRewind**(**slaves**,&**li**);**

**while((**ln **=** listNext**(&**li**)))** **{**

redisClient **\***slave **=** ln**->**value**;**

**if** **(**slave**->**replstate **==** REDIS\_REPL\_WAIT\_BGSAVE\_START**)** **continue;**

addReply**(**slave**,**selectcmd**);**

**}**

**if** **(**dictid **<** 0 **||** dictid **>=** REDIS\_SHARED\_SELECT\_CMDS**)**

decrRefCount**(**selectcmd**);**

**}**

server**.**slaveseldb **=** dictid**;**

*/\* Write the command to the replication backlog if any. \*/*

**if** **(**server**.**repl\_backlog**)** **{**

char aux**[**REDIS\_LONGSTR\_SIZE**+**3**];**

*/\* Add the multi bulk reply length. \*/*

aux**[**0**]** **=** '\*'**;**

len **=** ll2string**(**aux**+**1**,sizeof(**aux**)-**1**,**argc**);**

aux**[**len**+**1**]** **=** '\r'**;**

aux**[**len**+**2**]** **=** '\n'**;**

feedReplicationBacklog**(**aux**,**len**+**3**);**

**for** **(**j **=** 0**;** j **<** argc**;** j**++)** **{**

long objlen **=** stringObjectLen**(**argv**[**j**]);**

*/\* We need to feed the buffer with the object as a bulk reply*

*\* not just as a plain string, so create the $..CRLF payload len*

*\* and add the final CRLF \*/*

aux**[**0**]** **=** '$'**;**

len **=** ll2string**(**aux**+**1**,sizeof(**aux**)-**1**,**objlen**);**

aux**[**len**+**1**]** **=** '\r'**;**

aux**[**len**+**2**]** **=** '\n'**;**

feedReplicationBacklog**(**aux**,**len**+**3**);**

feedReplicationBacklogWithObject**(**argv**[**j**]);**

feedReplicationBacklog**(**aux**+**len**+**1**,**2**);**

**}**

**}**

*/\* Write the command to every slave. \*/*

listRewind**(**server**.**slaves**,&**li**);**

**while((**ln **=** listNext**(&**li**)))** **{**

redisClient **\***slave **=** ln**->**value**;**

*/\* Don't feed slaves that are still waiting for BGSAVE to start \*/*

**if** **(**slave**->**replstate **==** REDIS\_REPL\_WAIT\_BGSAVE\_START**)** **continue;**

*/\* Feed slaves that are waiting for the initial SYNC (so these commands*

*\* are queued in the output buffer until the initial SYNC completes),*

*\* or are already in sync with the master. \*/*

*/\* Add the multi bulk length. \*/*

addReplyMultiBulkLen**(**slave**,**argc**);**

*/\* Finally any additional argument that was not stored inside the*

*\* static buffer if any (from j to argc). \*/*

**for** **(**j **=** 0**;** j **<** argc**;** j**++)**

addReplyBulk**(**slave**,**argv**[**j**]);**

**}**

**}**

该函数用于主节点将收到的客户端命令请求，缓存到积压队列以及所有状态不是REDIS\_REPL\_WAIT\_BGSAVE\_START的从节点的输出缓存中。也就是说，当从节点的状态变为REDIS\_REPL\_WAIT\_BGSAVE\_END的那一刻起，主节点就一直会为从节点缓存命令流。

这里要注意的是：如果当前命令的数据库id不等于server.slaveseldb的话，就需要向积压队列和所有状态不是REDIS\_REPL\_WAIT\_BGSAVE\_START的从节点输出缓存中添加一条"SELECT"命令。这也就是为什么在函数replicationSetupSlaveForFullResync中，将server.slaveseldb置为-1原因了。这样保证第一次调用本函数时，强制增加一条"SELECT"命令到积压队列和从节点输出缓存中。

这里在向从节点的输出缓存中追加命令流时，调用的是addReply类的函数。在之前的《Redis服务器与客户端间的交互》中介绍过，这些函数用于将信息添加到客户端的输出缓存中，这些函数首先都会调用prepareClientToWrite函数，注册socket描述符上的可写事件，然后将回复信息写入到客户端输出缓存中。

但是在从节点的复制状态变为REDIS\_REPL\_ONLINE之前，是不能将命令流发送给从节点的。因此，需要在prepareClientToWrite函数中进行特殊处理。在该函数中，有下面的代码：

/\* Only install the handler if not already installed and, in case of

\* slaves, if the client can actually receive writes. \*/

**if** **(**c**->**bufpos **==** 0 **&&** listLength**(**c**->**reply**)** **==** 0 **&&**

**(**c**->**replstate **==** REDIS\_REPL\_NONE **||**

**(**c**->**replstate **==** REDIS\_REPL\_ONLINE **&&** **!**c**->**repl\_put\_online\_on\_ack**)))**

**{**

*/\* Try to install the write handler. \*/*

**if** **(**aeCreateFileEvent**(**server**.**el**,** c**->**fd**,** AE\_WRITABLE**,**

sendReplyToClient**,** c**)** **==** AE\_ERR**)**

**{**

freeClientAsync**(**c**);**

**return** REDIS\_ERR**;**

**}**

**}**

上面的代码保证了，当从节点客户端的复制状态尚未真正的变为REDIS\_REPL\_ONLINE时，是不会注册socket描述符上的可写事件的。（c->repl\_put\_online\_on\_ack标志的作用，会在下面的” BGSAVE操作完成”小节中讲解）

还需要注意的是，在写事件的回调函数sendReplyToClient中，有下面的代码：

**if** **(**c**->**bufpos **==** 0 **&&** listLength**(**c**->**reply**)** **==** 0**)** **{**

c**->**sentlen **=** 0**;**

aeDeleteFileEvent**(**server**.**el**,**c**->**fd**,**AE\_WRITABLE**);**

**...**

**}**

因此，当输出缓存中的内容全部发给客户端之后，就会删除socket描述符上的可写事件。这就保证了在主节点收到SYNC或PSYNC命令后，从节点的输出缓存为空时，该从节点的socket描述符上是没有注册可写事件的。

**六：BGSAVE操作完成**

当主节点在后台执行BGSAVE的子进程结束之后，主节点父进程wait到该子进程的退出状态后，会调用updateSlavesWaitingBgsave进行BGSAVE的收尾工作。

前面在”SYNC或PSYNC命令的处理”一节中提到过，如果主节点为从节点在后台进行RDB数据转储时，如果有新的从节点的SYNC或PSYNC命令到来。则在该新从节点无法复用当前正在转储的RDB数据的情况下，主节点需要在当前BGSAVE操作之后，重新进行一次BGSAVE操作。这就是在updateSlavesWaitingBgsave函数中进行的。

updateSlavesWaitingBgsave函数的代码如下：

void updateSlavesWaitingBgsave**(**int bgsaveerr**,** int type**)** **{**

listNode **\***ln**;**

int startbgsave **=** 0**;**

int mincapa **=** **-**1**;**

listIter li**;**

listRewind**(**server**.**slaves**,&**li**);**

**while((**ln **=** listNext**(&**li**)))** **{**

redisClient **\***slave **=** ln**->**value**;**

**if** **(**slave**->**replstate **==** REDIS\_REPL\_WAIT\_BGSAVE\_START**)** **{**

startbgsave **=** 1**;**

mincapa **=** **(**mincapa **==** **-**1**)** **?** slave**->**slave\_capa **:**

**(**mincapa **&** slave**->**slave\_capa**);**

**}** **else** **if** **(**slave**->**replstate **==** REDIS\_REPL\_WAIT\_BGSAVE\_END**)** **{**

struct redis\_stat buf**;**

*/\* If this was an RDB on disk save, we have to prepare to send*

*\* the RDB from disk to the slave socket. Otherwise if this was*

*\* already an RDB -> Slaves socket transfer, used in the case of*

*\* diskless replication, our work is trivial, we can just put*

*\* the slave online. \*/*

**if** **(**type **==** REDIS\_RDB\_CHILD\_TYPE\_SOCKET**)** **{**

redisLog**(**REDIS\_NOTICE**,**

"Streamed RDB transfer with slave %s succeeded (socket). Waiting for REPLCONF ACK from slave to enable streaming"**,**

replicationGetSlaveName**(**slave**));**

*/\* Note: we wait for a REPLCONF ACK message from slave in*

*\* order to really put it online (install the write handler*

*\* so that the accumulated data can be transfered). However*

*\* we change the replication state ASAP, since our slave*

*\* is technically online now. \*/*

slave**->**replstate **=** REDIS\_REPL\_ONLINE**;**

slave**->**repl\_put\_online\_on\_ack **=** 1**;**

slave**->**repl\_ack\_time **=** server**.**unixtime**;** */\* Timeout otherwise. \*/*

**}** **else** **{**

**if** **(**bgsaveerr **!=** REDIS\_OK**)** **{**

freeClient**(**slave**);**

redisLog**(**REDIS\_WARNING**,**"SYNC failed. BGSAVE child returned an error"**);**

**continue;**

**}**

**if** **((**slave**->**repldbfd **=** open**(**server**.**rdb\_filename**,**O\_RDONLY**))** **==** **-**1 **||**

redis\_fstat**(**slave**->**repldbfd**,&**buf**)** **==** **-**1**)** **{**

freeClient**(**slave**);**

redisLog**(**REDIS\_WARNING**,**"SYNC failed. Can't open/stat DB after BGSAVE: %s"**,** strerror**(**errno**));**

**continue;**

**}**

slave**->**repldboff **=** 0**;**

slave**->**repldbsize **=** buf**.**st\_size**;**

slave**->**replstate **=** REDIS\_REPL\_SEND\_BULK**;**

slave**->**replpreamble **=** sdscatprintf**(**sdsempty**(),**"$%lld\r\n"**,**

**(**unsigned long long**)** slave**->**repldbsize**);**

aeDeleteFileEvent**(**server**.**el**,**slave**->**fd**,**AE\_WRITABLE**);**

**if** **(**aeCreateFileEvent**(**server**.**el**,** slave**->**fd**,** AE\_WRITABLE**,** sendBulkToSlave**,** slave**)** **==** AE\_ERR**)** **{**

freeClient**(**slave**);**

**continue;**

**}**

**}**

**}**

**}**

**if** **(**startbgsave**)** startBgsaveForReplication**(**mincapa**);**

**}**

参数bgsaveerr表示后台子进程的退出状态；type如果为REDIS\_RDB\_CHILD\_TYPE\_DISK，表示是有硬盘复制的RDB数据；如果为REDIS\_RDB\_CHILD\_TYPE\_SOCKET，表示是无硬盘复制的RDB数据；

在函数中，轮训列表server.slaves，针对其中的每一个从节点客户端。如果有从节点客户端当前的复制状态为REDIS\_REPL\_WAIT\_BGSAVE\_START，说明该从节点是在后台子进程进行RDB数据转储期间，连接到主节点上的。并且没有合适的其他从节点可以进行复用。这种情况下，需要重新进行RDB数据转储或发送，因此置startbgsave为1，并且置mincapa为，状态为REDIS\_REPL\_WAIT\_BGSAVE\_START的所有从节点的"能力"的最小值；

如果从节点客户端当前的状态为REDIS\_REPL\_WAIT\_BGSAVE\_END，说明该从节点正在等待RDB数据处理完成（等待RDB转储到文件完成或者等待RDB数据发送完成）。

如果type为REDIS\_RDB\_CHILD\_TYPE\_SOCKET，说明无硬盘复制的RDB数据已发送给该从节点客户端，因此，置该从节点客户端的复制状态为REDIS\_REPL\_ONLINE，然后置从节点客户端中的repl\_put\_online\_on\_ack属性为1，表示在收到该从节点第一个"replconf ack <offset>"命令之后，才真正的调用putSlaveOnline将该从节点置为REDIS\_REPL\_ONLINE状态，并且开始发送缓存的命令流；这样处理的目的，已经在之前的”完全重同步时，从节点状态转换”一节中解释过了，不再赘述。

如果type为REDIS\_RDB\_CHILD\_TYPE\_DISK，说明RDB数据已转储到文件，接下来需要把该文件发送给所有从节点客户端。

如果bgsaveerr为REDIS\_ERR，则直接调用freeClient释放该从节点客户端（无硬盘复制的RDB数据发送，已经在函数backgroundSaveDoneHandlerSocket中处理过这种情况了，因此无需在本函数中处理）；

如果bgsaveerr为REDIS\_OK，打开RDB文件，描述符记录到slave->repldbfd中；置slave->repldboff为0；置slave->repldbsize为RDB文件大小；置从节点客户端的复制状态为REDIS\_REPL\_SEND\_BULK；

置slave->replpreamble为需要发送给从节点客户端的RDB文件的长度信息。从节点通过该信息判断要读取多少字节的RDB数据，这也是为什么有硬盘复制的RDB数据，不需要等待从节点第一个"replconf ack <offset>"命令，而可以直接在发送完RDB数据之后，直接调用putSlaveOnline将该从节点置为REDIS\_REPL\_ONLINE状态；

然后重新注册从节点客户端的socket描述符上的可写事件，事件回调函数为sendBulkToSlave；

轮训完所有从节点客户端之后，如果startbgsave为1，则使用mincapa调用函数startBgsaveForReplication，重新开始一次RDB数据处理过程。

1：有硬盘复制的RDB数据

有硬盘复制的RDB数据，接下来需要把RDB文件发送给所有从节点。这是通过从节点socket描述符上的可写事件的回调函数sendBulkToSlave实现的。在该函数中，需要用到从节点客户端的下列属性：

slave->repldbfd，表示打开的RDB文件描述符；

slave->repldbsize，表示RDB文件的大小；

slave->repldboff，表示已经向从节点发送的RDB数据的字节数；

slave->replpreamble，表示需要发送给从节点客户端的RDB文件的长度信息；

sendBulkToSlave函数的代码如下：

void sendBulkToSlave**(**aeEventLoop **\***el**,** int fd**,** void **\***privdata**,** int mask**)** **{**

redisClient **\***slave **=** privdata**;**

REDIS\_NOTUSED**(**el**);**

REDIS\_NOTUSED**(**mask**);**

char buf**[**REDIS\_IOBUF\_LEN**];**

ssize\_t nwritten**,** buflen**;**

*/\* Before sending the RDB file, we send the preamble as configured by the*

*\* replication process. Currently the preamble is just the bulk count of*

*\* the file in the form "$<length>\r\n". \*/*

**if** **(**slave**->**replpreamble**)** **{**

nwritten **=** write**(**fd**,**slave**->**replpreamble**,**sdslen**(**slave**->**replpreamble**));**

**if** **(**nwritten **==** **-**1**)** **{**

redisLog**(**REDIS\_VERBOSE**,**"Write error sending RDB preamble to slave: %s"**,**

strerror**(**errno**));**

freeClient**(**slave**);**

**return;**

**}**

server**.**stat\_net\_output\_bytes **+=** nwritten**;**

sdsrange**(**slave**->**replpreamble**,**nwritten**,-**1**);**

**if** **(**sdslen**(**slave**->**replpreamble**)** **==** 0**)** **{**

sdsfree**(**slave**->**replpreamble**);**

slave**->**replpreamble **=** **NULL;**

*/\* fall through sending data. \*/*

**}** **else** **{**

**return;**

**}**

**}**

*/\* If the preamble was already transfered, send the RDB bulk data. \*/*

lseek**(**slave**->**repldbfd**,**slave**->**repldboff**,**SEEK\_SET**);**

buflen **=** read**(**slave**->**repldbfd**,**buf**,**REDIS\_IOBUF\_LEN**);**

**if** **(**buflen **<=** 0**)** **{**

redisLog**(**REDIS\_WARNING**,**"Read error sending DB to slave: %s"**,**

**(**buflen **==** 0**)** **?** "premature EOF" **:** strerror**(**errno**));**

freeClient**(**slave**);**

**return;**

**}**

**if** **((**nwritten **=** write**(**fd**,**buf**,**buflen**))** **==** **-**1**)** **{**

**if** **(**errno **!=** EAGAIN**)** **{**

redisLog**(**REDIS\_WARNING**,**"Write error sending DB to slave: %s"**,**

strerror**(**errno**));**

freeClient**(**slave**);**

**}**

**return;**

**}**

slave**->**repldboff **+=** nwritten**;**

server**.**stat\_net\_output\_bytes **+=** nwritten**;**

**if** **(**slave**->**repldboff **==** slave**->**repldbsize**)** **{**

close**(**slave**->**repldbfd**);**

slave**->**repldbfd **=** **-**1**;**

aeDeleteFileEvent**(**server**.**el**,**slave**->**fd**,**AE\_WRITABLE**);**

putSlaveOnline**(**slave**);**

**}**

**}**

如果slave->replpreamble不为NULL，说明需要发送给从节点客户端RDB数据的长度信息，因此，直接调用write向从节点客户端发送slave->replpreamble中的信息。如果写入了部分数据，则将slave->replpreamble更新为未发送的数据，如果slave->replpreamble中的数据已全部发送完成，则释放slave->replpreamble，置其为NULL；否则，直接返回，下次可写事件触发时，接着向从节点发送slave->replpreamble信息；

如果slave->replpreamble为NULL，说明已经发送完长度信息了，接下来就是要发送实际的RDB数据了。

首先调用lseek将文件指针定位到该文件中未发送的位置，也就是slave->repldboff的位置；然后调用read，读取RDB文件中REDIS\_IOBUF\_LEN个字节到buf中；

然后调用write，将已读取的数据发送给从节点客户端，write返回值为nwritten，将其加到slave->repldboff中。

如果slave->repldboff的值等于slave->repldbsize，则表示RDB文件中的所有数据都发送完成了，因此关闭打开的RDB文件描述符slave->repldbfd；删除socket描述符上的可写事件，然后调用putSlaveOnline函数，更改该从节点客户端的复制状态为REDIS\_REPL\_ONLINE，接下来就可以开始向该从节点客户端发送累积的命令流了（尽管此时从节点可能还在进行RDB数据的加载，而无暇处理这些累积的命令流。不过好在有TCP输入缓冲区，可以先暂存下来，如果TCP的输入缓存被填满了，则不会向主节点发送ACK，则主节点的TCP输出缓存的剩余空间就会越来越少，当减少到水位线以下时，就不会在触发可写事件了）；

2：无硬盘复制的RDB数据

对于无硬盘复制的RDB数据，主节点收到从节点发来的第一个"replconf ack <offset>"命令之后，才真正的调用putSlaveOnline将该从节点置为REDIS\_REPL\_ONLINE状态。

以上的过程是在replconf命令处理函数replconfCommand中处理的。之前在”从节点建链和握手”小节中，已经看过该函数的部分代码了，接下来是该函数处理"replconf ack <offset>"命令的代码：

**else** **if** **(!**strcasecmp**(**c**->**argv**[**j**]->**ptr**,**"ack"**))** **{**

*/\* REPLCONF ACK is used by slave to inform the master the amount*

*\* of replication stream that it processed so far. It is an*

*\* internal only command that normal clients should never use. \*/*

long long offset**;**

**if** **(!(**c**->**flags **&** REDIS\_SLAVE**))** **return;**

**if** **((**getLongLongFromObject**(**c**->**argv**[**j**+**1**],** **&**offset**)** **!=** REDIS\_OK**))**

**return;**

**if** **(**offset **>** c**->**repl\_ack\_off**)**

c**->**repl\_ack\_off **=** offset**;**

c**->**repl\_ack\_time **=** server**.**unixtime**;**

*/\* If this was a diskless replication, we need to really put*

*\* the slave online when the first ACK is received (which*

*\* confirms slave is online and ready to get more data). \*/*

**if** **(**c**->**repl\_put\_online\_on\_ack **&&** c**->**replstate **==** REDIS\_REPL\_ONLINE**)**

putSlaveOnline**(**c**);**

*/\* Note: this command does not reply anything! \*/*

**return;**

**}**

可见，这里在客户端的repl\_put\_online\_on\_ack属性为1，并且复制状态为REDIS\_REPL\_ONLINE的情况下，调用putSlaveOnline函数，将该从节点的状态真正置为REDIS\_REPL\_ONLINE，并开始向该从节点发送累积的命令流。

3：putSlaveOnline函数

完全重同步的最后一步，就是调用putSlaveOnline函数，将从节点客户端的复制状态置为REDIS\_REPL\_ONLINE，并开始向该从节点发送累积的命令流。

该函数的代码如下：

void putSlaveOnline**(**redisClient **\***slave**)** **{**

slave**->**replstate **=** REDIS\_REPL\_ONLINE**;**

slave**->**repl\_put\_online\_on\_ack **=** 0**;**

slave**->**repl\_ack\_time **=** server**.**unixtime**;** */\* Prevent false timeout. \*/*

**if** **(**aeCreateFileEvent**(**server**.**el**,** slave**->**fd**,** AE\_WRITABLE**,**

sendReplyToClient**,** slave**)** **==** AE\_ERR**)** **{**

redisLog**(**REDIS\_WARNING**,**"Unable to register writable event for slave bulk transfer: %s"**,** strerror**(**errno**));**

freeClient**(**slave**);**

**return;**

**}**

refreshGoodSlavesCount**();**

redisLog**(**REDIS\_NOTICE**,**"Synchronization with slave %s succeeded"**,**

replicationGetSlaveName**(**slave**));**

**}**

首先将从节点客户端的复制状态置为REDIS\_REPL\_ONLINE，置客户端属性slave->repl\_put\_online\_on\_ack为0。表示该从节点已完成初始同步，接下来进入命令传播阶段；

然后，重新注册该从节点客户端的socket描述符上的可写事件，事件回调函数为sendReplyToClient，用于向从节点发送缓存的命令流。该函数也是向普通客户端回复命令时的回调函数；

最后，调用refreshGoodSlavesCount，更新当前状态正常的从节点数量；