Redis的主从复制功能，可以实现Redis实例的高可用，避免单个 Redis 服务器的单点故障，并且可以实现负载均衡。

**一：主从复制过程**

Redis的复制功能分为同步(sync)和命令传播(command propagate)两个操作：

同步操作用于将从节点的数据库状态更新至主节点当前所处的数据库状态；

命令传播操作则用于在主节点的数据库状态被修改，导致主从节点的数据库状态不一致时，让主从节点的数据库重新回到一致状态；

1：同步

当客户端向从节点发送SLAYEOF命令，或者从节点的配置文件中配置了slaveof选项时，从节点首先需要执行同步操作，也就是将从节点的数据库状态更新至主节点当前所处的数据库状态。

在Redis2.8版本之前，从节点对主节点的同步操作，是通过从节点向主节点发送SYNC命令来完成。过程如下：

a：从节点向主节点发送SYNC命令；

b：主节点收到SYNC命令后，执行BGSAVE命令，在后台生成一个RDB文件，并使用一个缓冲区记录从现在开始执行的所有写命令。

c：当主节点的BGSAVE命令执行完毕时，主节点会将生成的RDB文件发送给从节点，从节点接收并载人这个 RDB文件，将自己的数据库状态更新至主服务器执行BGSAYE命令时的状态。

d：主节点将记录在缓冲区里面的所有写命令发送给从节点，从节点执行这些写命令，将自己的数据库状态更新至主节点数据库当前所处的状态。

2：命令传播

在同步操作执行完毕之后，主从服务器两者的数据库将达到一致状态。但当主节点执行客户端发送的写命令时，主从服务器状态将不再一致。

为了让主从服务器再次回到一致状态，主服务器将自己执行的写命令，发送给从服务器执行，当从服务器执行了相同的写命令之后，主从服务器将再次回到一致状态。

3：完全重同步和部分重同步

以上就是旧版Redis执行主从复制时的过程。它有个缺点，就是当主从节点间的连接断开后，从节点会发送SYNC命令来重新进行一次完整复制操作。这样即使断开期间主节点的变化很小（甚至没有），也需要将主节点中的所有数据重新快照并传送一次。这种实现方式显然不太理想。

自2.8版开始，Redis支持部分重同步功能。该功能通过”PSYNC”命令实现。部分重同步是基于如下3点实现的：

a：从节点会保存主节点的运行ID。每个Redis 运行实例均会拥有一个唯一的运行ID，每当实例重启后，就会自动生成一个新的运行ID。

b：在命令传播阶段，主节点每将一个命令传送给从节点时，都会同时把该命令存放到一个积压队列(backlog)中，并记录下当前积压队列中，存放的命令的偏移量范围。

c：同时，从节点接收到主节点传来的命令时，会记录下该命令的偏移量。主节点和所有从节点都记录了命令的偏移量。

当主从连接准备就绪后，从节点会发送一条”PSYNC”命令，格式为”PSYNC <runid> <offset>”。

从节点第一次连接主节点是，置runid为”?”，offset为”-1”。如果是断链重连，则从节点发送之前保存的主节点运行ID和复制偏移。

主节点收到”PSYNC”命令后，会执行以下判断来决定此次重连是否可以执行部分重同步：

a：首先判断从节点传送来的<runid>是否和自己的运行ID相同；

b：然后判断从节点传送来的复制偏移量<offset>是否在积压队列中；

如果以上两个条件都满足，则可以执行部分重同步，并将积压队列中相应的命令发送给从节点。如果不满足，主节点会进行一次完全重同步，也就是进行之前版本中收到”SYNC”命令后的操作。

主从复制功能是从节点主动发起，主节点配合完成的，因此，本文先介绍从节点在主从复制时的流程。

注意，下面的流程都基于Redis3.0.5版本。

**二：从节点属性**

在Redis源码中，表示Redis服务器的全局结构体struct redisServer server中，与主从复制相关的，从节点属性如下：

server.masterhost：记录主节点的ip地址；

server.masterport：记录主节点的端口号；

server.repl\_transfer\_s：socket描述符，用于主从复制过程中，从节点与主节点之间的TCP通信，包括主从节点间的握手通信、接收RDB数据，以及后续的命令传播过程；

server.repl\_transfer\_fd：文件描述符，用于从节点将收到的RDB数据写到本地临时文件；

server.repl\_transfer\_tmpfile：从节点上，用于记录RDB数据的临时文件名；

server.repl\_state：记录主从复制过程中，从节点的状态。

server.master：当从节点接受完主节点发来的RDB数据之后，进入命令传播过程。从节点就将主节点当成一个客户端看待。server.master就是redisClient结构的主节点客户端，从节点接收该server.master发来的命令，像处理普通客户端的命令请求一样进行处理，从而实现了从节点和主节点之间的同步；

server.master->reploff：从节点记录的复制偏移量，每次收到主节点发来的命令时，就会将命令长度增加到该复制偏移量上，以保持和主节点复制偏移量的一致。

server.master->replrunid：从节点记录的主节点运行ID。

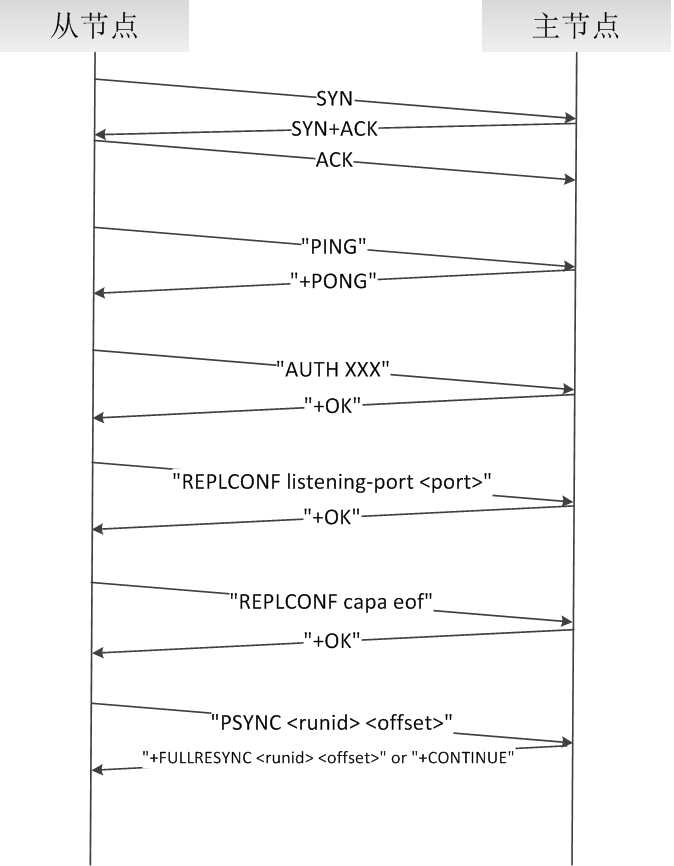
server.cached\_master：主从节点复制过程中（具体应该是命令传播过程中），如果从节点与主节点之间连接断掉了，会调用freeClient(server.master)，关闭与主节点客户端的连接。为了后续重连时能够进行部分重同步，在freeClient中，会调用replicationCacheMaster函数，将server.master保存到server.cached\_master。该redisClient结构中记录了主节点的运行ID，以及复制偏移。当后续与主节点的连接又重新建立起来的时候，使用这些信息进行部分重同步，也就是发送"PSYNC <runid> <offset>"命令。

server.repl\_master\_runid和server.repl\_master\_initial\_offset：从节点发送"PSYNC <runid> <offset>"命令后，如果主节点不支持部分重同步，则会回复信息为"+FULLRESYNC <runid> <offset>"，表示要进行完全重同步，其中<runid>表示主节点的运行ID，记录到server.repl\_master\_runid中，<offset>表示主节点的初始复制偏移，记录到server.repl\_master\_initial\_offset中。

**三：建链和握手过程**

从节点在收到客户端发来的”slaveof”命令时，或者在配置文件中配置了”slaveof”选项时，就会向主节点建链，开始主从复制过程。

在主节点将实际的RDB数据发送给从节点之前，还需要经历握手过程，这非常类似于TCP建链的三次握手。该过程由从节点主动发起，主节点作出相应的回应。握手过程如下：



该握手过程中，从节点的状态会发生转换，从REDIS\_REPL\_CONNECT状态起，一直到REDIS\_REPL\_RECEIVE\_PSYNC状态期间，都算是握手过程。

1：TCP建链

在Redis源码中，使用server.repl\_state记录从节点的状态。在Redis初始化时，该状态为REDIS\_REPL\_NONE。

当从节点收到客户端用户发来的”SLAVEOF” 命令时，或者在读取配置文件，发现了”slaveof”配置选项，就会将server.repl\_state置为REDIS\_REPL\_CONNECT状态。该状态表示下一步需要向主节点发起TCP建链。

在定时执行的函数serverCron中，会调用replicationCron函数检查主从复制的状态。该函数中，一旦发现当前的server.repl\_state为REDIS\_REPL\_CONNECT，则会调用函数connectWithMaster，向主节点发起TCP建链请求，其代码如下：

int connectWithMaster**(**void**)** **{**

int fd**;**

fd **=** anetTcpNonBlockBestEffortBindConnect**(NULL,**

server**.**masterhost**,**server**.**masterport**,**REDIS\_BIND\_ADDR**);**

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

redisLog**(**REDIS\_WARNING**,**"Unable to connect to MASTER: %s"**,**

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

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

**}**

**if** **(**aeCreateFileEvent**(**server**.**el**,**fd**,**AE\_READABLE**|**AE\_WRITABLE**,**syncWithMaster**,NULL)** **==**

AE\_ERR**)**

**{**

close**(**fd**);**

redisLog**(**REDIS\_WARNING**,**"Can't create readable event for SYNC"**);**

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

**}**

server**.**repl\_transfer\_lastio **=** server**.**unixtime**;**

server**.**repl\_transfer\_s **=** fd**;**

server**.**repl\_state **=** REDIS\_REPL\_CONNECTING**;**

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

**}**

server.masterhost和server.masterport分别记录了主节点的IP地址和端口号。它们要么是在slaveof选项中配置，要么是”SLAVEOF”命令中的参数。

首先调用anetTcpNonBlockBestEffortBindConnect，向主节点发起connect建链请求；该函数创建socket描述符，将该描述符设置为非阻塞，必要情况下会绑定本地地址，然后connect向主节点发起TCP建链请求。该函数返回建链的socket描述符fd；

然后注册socket描述符fd上的可读和可写事件，事件回调函数都为syncWithMaster，该函数用于处理主从节点间的握手过程；

然后将socket描述符记录到server.repl\_transfer\_s中。置主从复制状态server.repl\_state为REDIS\_REPL\_CONNECTING，表示从节点正在向主节点建链；

2：复制握手

当主从节点间的TCP建链成功之后，就会触发socket描述符server.repl\_transfer\_s上的可写事件，从而调用函数syncWithMaster。该函数处理从节点与主节点间的握手过程。也就是从节点在向主节点发送TCP建链请求，到接收RDB数据之前的过程。其代码如下：

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

char tmpfile**[**256**],** **\***err **=** **NULL;**

int dfd**,** maxtries **=** 5**;**

int sockerr **=** 0**,** psync\_result**;**

socklen\_t errlen **=** **sizeof(**sockerr**);**

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

REDIS\_NOTUSED**(**privdata**);**

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

*/\* If this event fired after the user turned the instance into a master*

*\* with SLAVEOF NO ONE we must just return ASAP. \*/*

**if** **(**server**.**repl\_state **==** REDIS\_REPL\_NONE**)** **{**

close**(**fd**);**

**return;**

**}**

*/\* Check for errors in the socket. \*/*

**if** **(**getsockopt**(**fd**,** SOL\_SOCKET**,** SO\_ERROR**,** **&**sockerr**,** **&**errlen**)** **==** **-**1**)**

sockerr **=** errno**;**

**if** **(**sockerr**)** **{**

redisLog**(**REDIS\_WARNING**,**"Error condition on socket for SYNC: %s"**,**

strerror**(**sockerr**));**

**goto** error**;**

**}**

*/\* Send a PING to check the master is able to reply without errors. \*/*

**if** **(**server**.**repl\_state **==** REDIS\_REPL\_CONNECTING**)** **{**

redisLog**(**REDIS\_NOTICE**,**"Non blocking connect for SYNC fired the event."**);**

*/\* Delete the writable event so that the readable event remains*

*\* registered and we can wait for the PONG reply. \*/*

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

server**.**repl\_state **=** REDIS\_REPL\_RECEIVE\_PONG**;**

*/\* Send the PING, don't check for errors at all, we have the timeout*

*\* that will take care about this. \*/*

err **=** sendSynchronousCommand**(**SYNC\_CMD\_WRITE**,**fd**,**"PING"**,NULL);**

**if** **(**err**)** **goto** write\_error**;**

**return;**

**}**

*/\* Receive the PONG command. \*/*

**if** **(**server**.**repl\_state **==** REDIS\_REPL\_RECEIVE\_PONG**)** **{**

err **=** sendSynchronousCommand**(**SYNC\_CMD\_READ**,**fd**,NULL);**

*/\* We accept only two replies as valid, a positive +PONG reply*

*\* (we just check for "+") or an authentication error.*

*\* Note that older versions of Redis replied with "operation not*

*\* permitted" instead of using a proper error code, so we test*

*\* both. \*/*

**if** **(**err**[**0**]** **!=** '+' **&&**

strncmp**(**err**,**"-NOAUTH"**,**7**)** **!=** 0 **&&**

strncmp**(**err**,**"-ERR operation not permitted"**,**28**)** **!=** 0**)**

**{**

redisLog**(**REDIS\_WARNING**,**"Error reply to PING from master: '%s'"**,**err**);**

sdsfree**(**err**);**

**goto** error**;**

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

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

"Master replied to PING, replication can continue..."**);**

**}**

sdsfree**(**err**);**

server**.**repl\_state **=** REDIS\_REPL\_SEND\_AUTH**;**

**}**

*/\* AUTH with the master if required. \*/*

**if** **(**server**.**repl\_state **==** REDIS\_REPL\_SEND\_AUTH**)** **{**

**if** **(**server**.**masterauth**)** **{**

err **=** sendSynchronousCommand**(**SYNC\_CMD\_WRITE**,**fd**,**"AUTH"**,**server**.**masterauth**,NULL);**

**if** **(**err**)** **goto** write\_error**;**

server**.**repl\_state **=** REDIS\_REPL\_RECEIVE\_AUTH**;**

**return;**

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

server**.**repl\_state **=** REDIS\_REPL\_SEND\_PORT**;**

**}**

**}**

*/\* Receive AUTH reply. \*/*

**if** **(**server**.**repl\_state **==** REDIS\_REPL\_RECEIVE\_AUTH**)** **{**

err **=** sendSynchronousCommand**(**SYNC\_CMD\_READ**,**fd**,NULL);**

**if** **(**err**[**0**]** **==** '-'**)** **{**

redisLog**(**REDIS\_WARNING**,**"Unable to AUTH to MASTER: %s"**,**err**);**

sdsfree**(**err**);**

**goto** error**;**

**}**

sdsfree**(**err**);**

server**.**repl\_state **=** REDIS\_REPL\_SEND\_PORT**;**

**}**

*/\* Set the slave port, so that Master's INFO command can list the*

*\* slave listening port correctly. \*/*

**if** **(**server**.**repl\_state **==** REDIS\_REPL\_SEND\_PORT**)** **{**

sds port **=** sdsfromlonglong**(**server**.**port**);**

err **=** sendSynchronousCommand**(**SYNC\_CMD\_WRITE**,**fd**,**"REPLCONF"**,**

"listening-port"**,**port**,** **NULL);**

sdsfree**(**port**);**

**if** **(**err**)** **goto** write\_error**;**

sdsfree**(**err**);**

server**.**repl\_state **=** REDIS\_REPL\_RECEIVE\_PORT**;**

**return;**

**}**

*/\* Receive REPLCONF listening-port reply. \*/*

**if** **(**server**.**repl\_state **==** REDIS\_REPL\_RECEIVE\_PORT**)** **{**

err **=** sendSynchronousCommand**(**SYNC\_CMD\_READ**,**fd**,NULL);**

*/\* Ignore the error if any, not all the Redis versions support*

*\* REPLCONF listening-port. \*/*

**if** **(**err**[**0**]** **==** '-'**)** **{**

redisLog**(**REDIS\_NOTICE**,**"(Non critical) Master does not understand "

"REPLCONF listening-port: %s"**,** err**);**

**}**

sdsfree**(**err**);**

server**.**repl\_state **=** REDIS\_REPL\_SEND\_CAPA**;**

**}**

*/\* Inform the master of our capabilities. While we currently send*

*\* just one capability, it is possible to chain new capabilities here*

*\* in the form of REPLCONF capa X capa Y capa Z ...*

*\* The master will ignore capabilities it does not understand. \*/*

**if** **(**server**.**repl\_state **==** REDIS\_REPL\_SEND\_CAPA**)** **{**

err **=** sendSynchronousCommand**(**SYNC\_CMD\_WRITE**,**fd**,**"REPLCONF"**,**

"capa"**,**"eof"**,NULL);**

**if** **(**err**)** **goto** write\_error**;**

sdsfree**(**err**);**

server**.**repl\_state **=** REDIS\_REPL\_RECEIVE\_CAPA**;**

**return;**

**}**

*/\* Receive CAPA reply. \*/*

**if** **(**server**.**repl\_state **==** REDIS\_REPL\_RECEIVE\_CAPA**)** **{**

err **=** sendSynchronousCommand**(**SYNC\_CMD\_READ**,**fd**,NULL);**

*/\* Ignore the error if any, not all the Redis versions support*

*\* REPLCONF capa. \*/*

**if** **(**err**[**0**]** **==** '-'**)** **{**

redisLog**(**REDIS\_NOTICE**,**"(Non critical) Master does not understand "

"REPLCONF capa: %s"**,** err**);**

**}**

sdsfree**(**err**);**

server**.**repl\_state **=** REDIS\_REPL\_SEND\_PSYNC**;**

**}**

*/\* Try a partial resynchonization. If we don't have a cached master*

*\* slaveTryPartialResynchronization() will at least try to use PSYNC*

*\* to start a full resynchronization so that we get the master run id*

*\* and the global offset, to try a partial resync at the next*

*\* reconnection attempt. \*/*

**if** **(**server**.**repl\_state **==** REDIS\_REPL\_SEND\_PSYNC**)** **{**

**if** **(**slaveTryPartialResynchronization**(**fd**,**0**)** **==** PSYNC\_WRITE\_ERROR**)** **{**

err **=** sdsnew**(**"Write error sending the PSYNC command."**);**

**goto** write\_error**;**

**}**

server**.**repl\_state **=** REDIS\_REPL\_RECEIVE\_PSYNC**;**

**return;**

**}**

*/\* If reached this point, we should be in REDIS\_REPL\_RECEIVE\_PSYNC. \*/*

**if** **(**server**.**repl\_state **!=** REDIS\_REPL\_RECEIVE\_PSYNC**)** **{**

redisLog**(**REDIS\_WARNING**,**"syncWithMaster(): state machine error, "

"state should be RECEIVE\_PSYNC but is %d"**,**

server**.**repl\_state**);**

**goto** error**;**

**}**

psync\_result **=** slaveTryPartialResynchronization**(**fd**,**1**);**

**if** **(**psync\_result **==** PSYNC\_WAIT\_REPLY**)** **return;** */\* Try again later... \*/*

*/\* Note: if PSYNC does not return WAIT\_REPLY, it will take care of*

*\* uninstalling the read handler from the file descriptor. \*/*

**if** **(**psync\_result **==** PSYNC\_CONTINUE**)** **{**

redisLog**(**REDIS\_NOTICE**,** "MASTER <-> SLAVE sync: Master accepted a Partial Resynchronization."**);**

**return;**

**}**

*/\* PSYNC failed or is not supported: we want our slaves to resync with us*

*\* as well, if we have any (chained replication case). The mater may*

*\* transfer us an entirely different data set and we have no way to*

*\* incrementally feed our slaves after that. \*/*

disconnectSlaves**();** */\* Force our slaves to resync with us as well. \*/*

freeReplicationBacklog**();** */\* Don't allow our chained slaves to PSYNC. \*/*

*/\* Fall back to SYNC if needed. Otherwise psync\_result == PSYNC\_FULLRESYNC*

*\* and the server.repl\_master\_runid and repl\_master\_initial\_offset are*

*\* already populated. \*/*

**if** **(**psync\_result **==** PSYNC\_NOT\_SUPPORTED**)** **{**

redisLog**(**REDIS\_NOTICE**,**"Retrying with SYNC..."**);**

**if** **(**syncWrite**(**fd**,**"SYNC\r\n"**,**6**,**server**.**repl\_syncio\_timeout**\***1000**)** **==** **-**1**)** **{**

redisLog**(**REDIS\_WARNING**,**"I/O error writing to MASTER: %s"**,**

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

**goto** error**;**

**}**

**}**

*/\* Prepare a suitable temp file for bulk transfer \*/*

**while(**maxtries**--)** **{**

snprintf**(**tmpfile**,**256**,**

"temp-%d.%ld.rdb"**,(**int**)**server**.**unixtime**,(**long int**)**getpid**());**

dfd **=** open**(**tmpfile**,**O\_CREAT**|**O\_WRONLY**|**O\_EXCL**,**0644**);**

**if** **(**dfd **!=** **-**1**)** **break;**

sleep**(**1**);**

**}**

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

redisLog**(**REDIS\_WARNING**,**"Opening the temp file needed for MASTER <-> SLAVE synchronization: %s"**,**strerror**(**errno**));**

**goto** error**;**

**}**

*/\* Setup the non blocking download of the bulk file. \*/*

**if** **(**aeCreateFileEvent**(**server**.**el**,**fd**,** AE\_READABLE**,**readSyncBulkPayload**,NULL)**

**==** AE\_ERR**)**

**{**

redisLog**(**REDIS\_WARNING**,**

"Can't create readable event for SYNC: %s (fd=%d)"**,**

strerror**(**errno**),**fd**);**

**goto** error**;**

**}**

server**.**repl\_state **=** REDIS\_REPL\_TRANSFER**;**

server**.**repl\_transfer\_size **=** **-**1**;**

server**.**repl\_transfer\_read **=** 0**;**

server**.**repl\_transfer\_last\_fsync\_off **=** 0**;**

server**.**repl\_transfer\_fd **=** dfd**;**

server**.**repl\_transfer\_lastio **=** server**.**unixtime**;**

server**.**repl\_transfer\_tmpfile **=** zstrdup**(**tmpfile**);**

**return;**

error**:**

aeDeleteFileEvent**(**server**.**el**,**fd**,**AE\_READABLE**|**AE\_WRITABLE**);**

close**(**fd**);**

server**.**repl\_transfer\_s **=** **-**1**;**

server**.**repl\_state **=** REDIS\_REPL\_CONNECT**;**

**return;**

write\_error**:** */\* Handle sendSynchronousCommand(SYNC\_CMD\_WRITE) errors. \*/*

redisLog**(**REDIS\_WARNING**,**"Sending command to master in replication handshake: %s"**,** err**);**

sdsfree**(**err**);**

**goto** error**;**

**}**

函数中如果发生了错误，则错误处理的方式是：删除socket描述符上注册的可读和可写事件，然后关闭描述符，置状态server.repl\_state为REDIS\_REPL\_CONNECT，等待下次调用replicationCron时重连主节点；

首先检查当前主从复制状态server.repl\_state是否为REDIS\_REPL\_NONE，若是，则说明握手过程期间，从节点收到了客户端执行的"slave no one"命令，因此直接关闭socket描述符，然后返回；

然后调用getsockopt，检查当前socket描述符的错误，若出错，则执行错误处理流程；

如果当前的复制状态为REDIS\_REPL\_CONNECTING，则说明是从节点connect主节点成功后，触发了描述符的可写事件，从而调用的该回调函数。这种情况下，先删除描述符上的可写事件，然后将状态设置为REDIS\_REPL\_RECEIVE\_PONG，向主节点发送"PING"命令，然后返回；

如果当前的复制状态为REDIS\_REPL\_RECEIVE\_PONG，则说明从节点收到了主节点对于"PING"命令的回复，触发了描述符的可读事件，从而调用的该回调函数。这种情况下，首先读取主节点的回复信息，正常情况下，主节点的回复只能有三种情况："+PONG"，"-NOAUTH"和"-ERR operation not permitted"（老版本的redis主节点），如果收到的回复不是以上的三种，则直接进入错误处理代码流程。否则，将复制状态置为REDIS\_REPL\_SEND\_AUTH（不返回）；

当前的复制状态为REDIS\_REPL\_SEND\_AUTH，如果配置了"masterauth"选项，则向主节点发送"AUTH"命令，后跟"masterauth"选项的值，然后将状态置为REDIS\_REPL\_RECEIVE\_AUTH，然后返回；

如果从节点没有配置"masterauth"选项，则将复制状态置为REDIS\_REPL\_SEND\_PORT（不返回）；

如果当前的复制状态为REDIS\_REPL\_RECEIVE\_AUTH，说明从节点收到了主节点对于"AUTH"命令的回复，触发了描述符的可读事件，从而调用的该回调函数。这种情况下，首先读取主节点的回复，如果回复信息的首字节为"-"，说明认证失败，直接进入错误处理流程；否则，将状态置为REDIS\_REPL\_SEND\_PORT（不返回）；

如果当前复制状态为REDIS\_REPL\_SEND\_PORT，则向主节点发送"REPLCONF listening-port <port>"命令，告知主节点本身的端口号，然后将复制状态置为REDIS\_REPL\_RECEIVE\_PORT后返回；

如果当前的复制状态为REDIS\_REPL\_RECEIVE\_PORT，说明从节点收到了主节点对于"REPLCONF listening-port"命令的回复，触发了描述符的可读事件，从而调用的该回调函数。这种情况下，首先读取主节点的回复，如果回复信息的首字节为"-"，说明主节点不认识该命令，这不是致命错误，只是记录日志而已；然后将复制状态设置为REDIS\_REPL\_SEND\_CAPA（不返回）；

如果当前的复制状态为REDIS\_REPL\_SEND\_CAPA，则向主节点发送"REPLCONF capa eof"命令，告知主节点本身的"能力"，然后将复制状态置为REDIS\_REPL\_RECEIVE\_CAPA后返回；

如果当前的复制状态为REDIS\_REPL\_RECEIVE\_CAPA，说明从节点收到了主节点对于"REPLCONF capa eof"命令的回复，触发了描述符的可读事件，从而调用的该回调函数。这种情况下，首先读取主节点的回复，如果回复信息的首字节为"-"，说明主节点不认识该命令，这不是致命错误，只是记录日志，然后将复制状态设置为REDIS\_REPL\_SEND\_PSYNC（不返回）；

如果复制状态为REDIS\_REPL\_SEND\_PSYNC，则调用slaveTryPartialResynchronization函数，向主节点发送"PSYNC <psync\_runid> <psync\_offset>"命令。

在该函数中，如果从节点缓存了主节点，说明该从节点之前与主节点的连接断掉了，现在是重新连接，因此尝试进行部分重同步。置psync\_runid为保存的主节点ID，置psync\_offset为保存的主节点复制偏移加1；如果从节点没有缓存主节点，说明需要进行完全重同步，则置psync\_runid为"?"，置psync\_offset为"-1"；

发送命令成功后函数返回，将复制状态置为REDIS\_REPL\_RECEIVE\_PSYNC后返回；

接下来的代码处理握手过程的最后一个状态REDIS\_REPL\_RECEIVE\_PSYNC，走到这里，复制状态只能是REDIS\_REPL\_RECEIVE\_PSYNC，如果不是则进入错误处理流程；

调用slaveTryPartialResynchronization读取主节点对于"PSYNC"命令的回复：

如果回复信息以"+CONTINUE"开头，说明主节点可以进行部分重同步，这种情况下，设置复制状态为REDIS\_REPL\_CONNECTED，后续将主节点当成一个客户端，接收该主节点客户端发来的命令请求，像处理普通客户端一样处理即可。因此函数slaveTryPartialResynchronization返回PSYNC\_CONTINUE后，该函数直接返回即可；

如果回复信息以"+FULLRESYNC"开头，说明主节点虽然认识"PSYNC"命令，但是从节点发送的复制偏移psync\_offset已经不在主节点的积压队列中了，因此需要进行完全重同步。解析出回复信息中的主节点ID，保存在server.repl\_master\_runid中；解析出主节点复制偏移初始值，保存在server.repl\_master\_initial\_offset中；然后函数slaveTryPartialResynchronization返回PSYNC\_FULLRESYNC；

如果回复信息不属于以上的情况，说明主节点不认识"PSYNC"命令，这种情况下，函数slaveTryPartialResynchronization返回PSYNC\_NOT\_SUPPORTED；

不管函数slaveTryPartialResynchronization返回PSYNC\_FULLRESYNC，还是返回PSYNC\_NOT\_SUPPORTED，都表示接下来要进行完全重同步过程：

首先断开当前实例与所有从节点的连接，因为接下来要进行完全重同步，本实例会接收主节点发来的完全不同的数据，因此此举可以让该实例的从节点重新进行复制同步过程（从而也接收这些数据）；

然后调用freeReplicationBacklog，释放本实例的积压队列server.repl\_backlog；

如果slaveTryPartialResynchronization函数返回的是PSYNC\_NOT\_SUPPORTED，说明这是老版本的主节点，不支持"PSYNC"命令，因此向主节点发送"SYNC"命令（主节点收到该命令后，直接发送RDB数据）；

接下来，就是为接收主节点发送来的RDB数据做准备：

首先创建保存RDB数据的临时文件"temp-<unixtime>.<pid>.rdb"，该文件的描述符记录到server.repl\_transfer\_fd中；

然后，注册socket描述符server.repl\_transfer\_s上的可读事件，事件回调函数为readSyncBulkPayload；

最后，置复制状态为REDIS\_REPL\_TRANSFER，表示开始接收主节点的RDB数据。然后执行下列操作后返回：

server**.**repl\_state **=** REDIS\_REPL\_TRANSFER**;**

server**.**repl\_transfer\_size **=** **-**1**;**

server**.**repl\_transfer\_read **=** 0**;**

server**.**repl\_transfer\_last\_fsync\_off **=** 0**;**

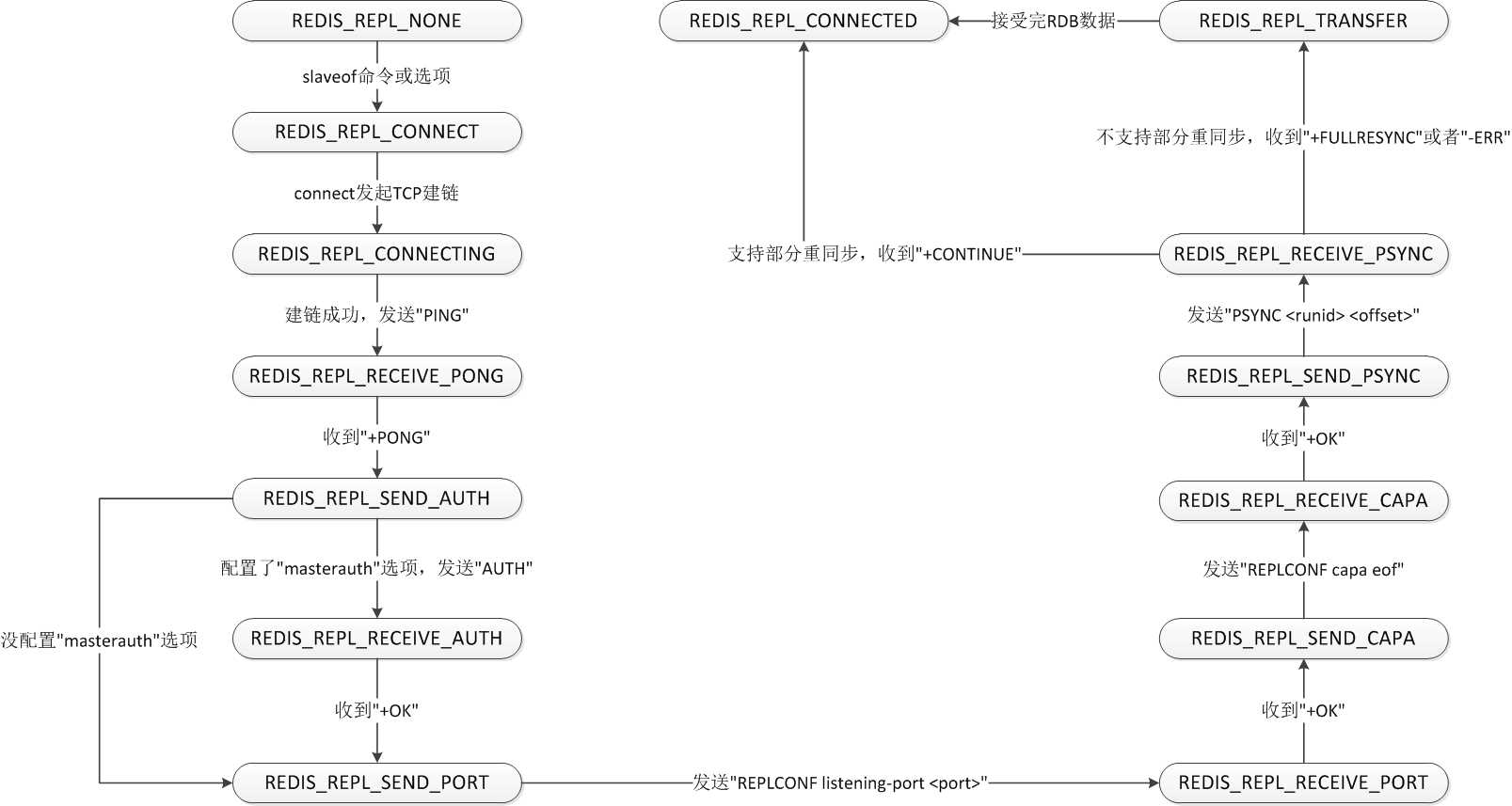
server**.**repl\_transfer\_fd **=** dfd**;**

server**.**repl\_transfer\_lastio **=** server**.**unixtime**;**

server**.**repl\_transfer\_tmpfile **=** zstrdup**(**tmpfile**);**

**四：从节点的复制状态转换**

根据以上的握手过程，总结出从节点的复制状态转换图，如下图所示：



在这些状态中，REDIS\_REPL\_CONNECT状态是从节点的初始状态，在状态转移过程中，出现了任何错误，都会关闭socket描述符，然后将状态置为REDIS\_REPL\_CONNECT，等待下次调用定时函数replicationCron时，重新连接主节点。

从REDIS\_REPL\_RECEIVE\_PONG状态到REDIS\_REPL\_RECEIVE\_PSYNC状态之间，是主从节点间的握手过程。

REDIS\_REPL\_RECEIVE\_PSYNC状态之后，如果主节点支持部分重同步，则从节点进入状态REDIS\_REPL\_CONNECTED，后续从节点将主节点当成客户端server.master，从节点接收客户端server.master发来的命令，像处理普通客户端的命令请求一样进行处理，从而实现了从节点和主节点之间的同步；

如果主节点不支持部分重同步，则需要进行完全重同步，从节点进入REDIS\_REPL\_TRANSFER状态，开始接收主节点发来的RDB数据。一旦从节点接收到完整的RDB数据，则加载该RDB数据，加载完成之后，从节点进入REDIS\_REPL\_CONNECTED状态，将主节点当成客户端server.master，接收客户端server.master发来的命令，实现了从节点和主节点之间的同步；

**五：接收RDB数据**

正常情况下，完全重同步需要主节点将其中的数据转储到RDB文件中，然后将该文件发送给从节点。如果硬盘IO效率较差，则这种操作对于主节点的性能会造成会影响。

从2.8.18版本开始，Redis引入了“无硬盘复制”选项，开启该选项时，Redis 在与从节点进行复制初始化时将不会将快照内容存储到硬盘上，而是直接通过网络发送给从节点，避免了硬盘的性能瓶颈。不过该功能还在试验阶段，可以在配置文件中使用"repl-diskless-sync"选项来配置开启该功能。

有硬盘复制的RDB数据和无硬盘复制的RDB数据，它们的格式是不一样的。有硬盘复制的RDB数据，主节点将数据保存到RDB文件后，将文件内容加上"$<len>/r/n"的头部后，发送给从节点。无硬盘复制的RDB数据，主节点直接将数据发送给从节点，而不再先保存到本地文件中，这种格式的RDB数据以"$EOF:<XXX>\r\n"开头，以"<XXX>"结尾。开头和结尾中的<XXX>内容相同，都是40字节长的，由"0123456789abcdef"中的字符组成的随机字符串。

在syncWithMaster函数中，握手过程结束后，需要进行完全重同步时，从节点注册了socket描述符server.repl\_transfer\_s上的可读事件，事件回调函数为readSyncBulkPayload。从节点调用该函数接收主节点发来的RDB数据，该函数的代码如下：

#define REPL\_MAX\_WRITTEN\_BEFORE\_FSYNC (1024\*1024\*8) /\* 8 MB \*/

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

char buf**[**4096**];**

ssize\_t nread**,** readlen**;**

off\_t left**;**

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

REDIS\_NOTUSED**(**privdata**);**

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

*/\* Static vars used to hold the EOF mark, and the last bytes received*

*\* form the server: when they match, we reached the end of the transfer. \*/*

static char eofmark**[**REDIS\_RUN\_ID\_SIZE**];**

static char lastbytes**[**REDIS\_RUN\_ID\_SIZE**];**

static int usemark **=** 0**;**

*/\* If repl\_transfer\_size == -1 we still have to read the bulk length*

*\* from the master reply. \*/*

**if** **(**server**.**repl\_transfer\_size **==** **-**1**)** **{**

**if** **(**syncReadLine**(**fd**,**buf**,**1024**,**server**.**repl\_syncio\_timeout**\***1000**)** **==** **-**1**)** **{**

redisLog**(**REDIS\_WARNING**,**

"I/O error reading bulk count from MASTER: %s"**,**

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

**goto** error**;**

**}**

**if** **(**buf**[**0**]** **==** '-'**)** **{**

redisLog**(**REDIS\_WARNING**,**

"MASTER aborted replication with an error: %s"**,**

buf**+**1**);**

**goto** error**;**

**}** **else** **if** **(**buf**[**0**]** **==** '\0'**)** **{**

*/\* At this stage just a newline works as a PING in order to take*

*\* the connection live. So we refresh our last interaction*

*\* timestamp. \*/*

server**.**repl\_transfer\_lastio **=** server**.**unixtime**;**

**return;**

**}** **else** **if** **(**buf**[**0**]** **!=** '$'**)** **{**

redisLog**(**REDIS\_WARNING**,**"Bad protocol from MASTER, the first byte is not '$' (we received '%s'), are you sure the host and port are right?"**,** buf**);**

**goto** error**;**

**}**

*/\* There are two possible forms for the bulk payload. One is the*

*\* usual $<count> bulk format. The other is used for diskless transfers*

*\* when the master does not know beforehand the size of the file to*

*\* transfer. In the latter case, the following format is used:*

*\**

*\* $EOF:<40 bytes delimiter>*

*\**

*\* At the end of the file the announced delimiter is transmitted. The*

*\* delimiter is long and random enough that the probability of a*

*\* collision with the actual file content can be ignored. \*/*

**if** **(**strncmp**(**buf**+**1**,**"EOF:"**,**4**)** **==** 0 **&&** strlen**(**buf**+**5**)** **>=** REDIS\_RUN\_ID\_SIZE**)** **{**

usemark **=** 1**;**

memcpy**(**eofmark**,**buf**+**5**,**REDIS\_RUN\_ID\_SIZE**);**

memset**(**lastbytes**,**0**,**REDIS\_RUN\_ID\_SIZE**);**

*/\* Set any repl\_transfer\_size to avoid entering this code path*

*\* at the next call. \*/*

server**.**repl\_transfer\_size **=** 0**;**

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

"MASTER <-> SLAVE sync: receiving streamed RDB from master"**);**

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

usemark **=** 0**;**

server**.**repl\_transfer\_size **=** strtol**(**buf**+**1**,NULL,**10**);**

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

"MASTER <-> SLAVE sync: receiving %lld bytes from master"**,**

**(**long long**)** server**.**repl\_transfer\_size**);**

**}**

**return;**

**}**

*/\* 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**);**

**}**

nread **=** read**(**fd**,**buf**,**readlen**);**

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

redisLog**(**REDIS\_WARNING**,**"I/O error trying to sync with MASTER: %s"**,**

**(**nread **==** **-**1**)** **?** strerror**(**errno**)** **:** "connection lost"**);**

replicationAbortSyncTransfer**();**

**return;**

**}**

server**.**stat\_net\_input\_bytes **+=** nread**;**

*/\* When a mark is used, we want to detect EOF asap in order to avoid*

*\* writing the EOF mark into the file... \*/*

int eof\_reached **=** 0**;**

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

*/\* Update the last bytes array, and check if it matches our delimiter.\*/*

**if** **(**nread **>=** REDIS\_RUN\_ID\_SIZE**)** **{**

memcpy**(**lastbytes**,**buf**+**nread**-**REDIS\_RUN\_ID\_SIZE**,**REDIS\_RUN\_ID\_SIZE**);**

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

int rem **=** REDIS\_RUN\_ID\_SIZE**-**nread**;**

memmove**(**lastbytes**,**lastbytes**+**nread**,**rem**);**

memcpy**(**lastbytes**+**rem**,**buf**,**nread**);**

**}**

**if** **(**memcmp**(**lastbytes**,**eofmark**,**REDIS\_RUN\_ID\_SIZE**)** **==** 0**)** eof\_reached **=** 1**;**

**}**

server**.**repl\_transfer\_lastio **=** server**.**unixtime**;**

**if** **(**write**(**server**.**repl\_transfer\_fd**,**buf**,**nread**)** **!=** nread**)** **{**

redisLog**(**REDIS\_WARNING**,**"Write error or short write writing to the DB dump file needed for MASTER <-> SLAVE synchronization: %s"**,** strerror**(**errno**));**

**goto** error**;**

**}**

server**.**repl\_transfer\_read **+=** nread**;**

*/\* Delete the last 40 bytes from the file if we reached EOF. \*/*

**if** **(**usemark **&&** eof\_reached**)** **{**

**if** **(**ftruncate**(**server**.**repl\_transfer\_fd**,**

server**.**repl\_transfer\_read **-** REDIS\_RUN\_ID\_SIZE**)** **==** **-**1**)**

**{**

redisLog**(**REDIS\_WARNING**,**"Error truncating the RDB file received from the master for SYNC: %s"**,** strerror**(**errno**));**

**goto** error**;**

**}**

**}**

*/\* Sync data on disk from time to time, otherwise at the end of the transfer*

*\* we may suffer a big delay as the memory buffers are copied into the*

*\* actual disk. \*/*

**if** **(**server**.**repl\_transfer\_read **>=**

server**.**repl\_transfer\_last\_fsync\_off **+** REPL\_MAX\_WRITTEN\_BEFORE\_FSYNC**)**

**{**

off\_t sync\_size **=** server**.**repl\_transfer\_read **-**

server**.**repl\_transfer\_last\_fsync\_off**;**

rdb\_fsync\_range**(**server**.**repl\_transfer\_fd**,**

server**.**repl\_transfer\_last\_fsync\_off**,** sync\_size**);**

server**.**repl\_transfer\_last\_fsync\_off **+=** sync\_size**;**

**}**

*/\* Check if the transfer is now complete \*/*

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

**if** **(**server**.**repl\_transfer\_read **==** server**.**repl\_transfer\_size**)**

eof\_reached **=** 1**;**

**}**

**if** **(**eof\_reached**)** **{**

**if** **(**rename**(**server**.**repl\_transfer\_tmpfile**,**server**.**rdb\_filename**)** **==** **-**1**)** **{**

redisLog**(**REDIS\_WARNING**,**"Failed trying to rename the temp DB into dump.rdb in MASTER <-> SLAVE synchronization: %s"**,** strerror**(**errno**));**

replicationAbortSyncTransfer**();**

**return;**

**}**

redisLog**(**REDIS\_NOTICE**,** "MASTER <-> SLAVE sync: Flushing old data"**);**

signalFlushedDb**(-**1**);**

emptyDb**(**replicationEmptyDbCallback**);**

*/\* Before loading the DB into memory we need to delete the readable*

*\* handler, otherwise it will get called recursively since*

*\* rdbLoad() will call the event loop to process events from time to*

*\* time for non blocking loading. \*/*

aeDeleteFileEvent**(**server**.**el**,**server**.**repl\_transfer\_s**,**AE\_READABLE**);**

redisLog**(**REDIS\_NOTICE**,** "MASTER <-> SLAVE sync: Loading DB in memory"**);**

**if** **(**rdbLoad**(**server**.**rdb\_filename**)** **!=** REDIS\_OK**)** **{**

redisLog**(**REDIS\_WARNING**,**"Failed trying to load the MASTER synchronization DB from disk"**);**

replicationAbortSyncTransfer**();**

**return;**

**}**

*/\* Final setup of the connected slave <- master link \*/*

zfree**(**server**.**repl\_transfer\_tmpfile**);**

close**(**server**.**repl\_transfer\_fd**);**

server**.**master **=** createClient**(**server**.**repl\_transfer\_s**);**

server**.**master**->**flags **|=** REDIS\_MASTER**;**

server**.**master**->**authenticated **=** 1**;**

server**.**repl\_state **=** REDIS\_REPL\_CONNECTED**;**

server**.**master**->**reploff **=** server**.**repl\_master\_initial\_offset**;**

memcpy**(**server**.**master**->**replrunid**,** server**.**repl\_master\_runid**,**

**sizeof(**server**.**repl\_master\_runid**));**

*/\* If master offset is set to -1, this master is old and is not*

*\* PSYNC capable, so we flag it accordingly. \*/*

**if** **(**server**.**master**->**reploff **==** **-**1**)**

server**.**master**->**flags **|=** REDIS\_PRE\_PSYNC**;**

redisLog**(**REDIS\_NOTICE**,** "MASTER <-> SLAVE sync: Finished with success"**);**

*/\* Restart the AOF subsystem now that we finished the sync. This*

*\* will trigger an AOF rewrite, and when done will start appending*

*\* to the new file. \*/*

**if** **(**server**.**aof\_state **!=** REDIS\_AOF\_OFF**)** **{**

int retry **=** 10**;**

stopAppendOnly**();**

**while** **(**retry**--** **&&** startAppendOnly**()** **==** REDIS\_ERR**)** **{**

redisLog**(**REDIS\_WARNING**,**"Failed enabling the AOF after successful master synchronization! Trying it again in one second."**);**

sleep**(**1**);**

**}**

**if** **(!**retry**)** **{**

redisLog**(**REDIS\_WARNING**,**"FATAL: this slave instance finished the synchronization with its master, but the AOF can't be turned on. Exiting now."**);**

exit**(**1**);**

**}**

**}**

**}**

**return;**

error**:**

replicationAbortSyncTransfer**();**

**return;**

**}**

server.repl\_transfer\_size的值表示要读取的RDB数据的总长度（仅对有硬盘复制的RDB数据而言）。如果当前其值为-1，说明本次是第一次接收RDB数据。因此，首先调用syncReadLine，读取主节点发来的第一行数据("\r\n"之前的内容)到buf中，读取的超时时间为5s，如果在5s之内还读不到"\n"，则syncReadLine返回-1，因此调用函数replicationAbortSyncTransfer，终止本次复制过程，然后返回；

然后解析读取到的内容，如果符合无硬盘复制的RDB数据格式，则将40字节的随机串记录到静态变量eofmark中，并且置usemark为1，置server.repl\_transfer\_size为0，然后返回；

如果不符合无硬盘复制的RDB数据格式，则认为是有硬盘复制的RDB数据，从buf中解析得到RDB数据的长度，记录到server.repl\_transfer\_size中，并且置usemark为0后返回；

后续可读事件触发，再次调用该函数时，server.repl\_transfer\_size已不再是-1，开始接收真正的RDB数据了。usemark为0，表示是有硬盘复制的RDB数据，为1，表示是无硬盘复制的的RDB数据；

接下来调用read，读取RDB数据内容到buf中。read返回值为nread，如果nread小于等于0，要么说明发生了错误，要么说明主节点终止了链接，无论哪种情况，都是调用函数replicationAbortSyncTransfer，终止本次复制过程，然后返回；

如果nread大于0，则先将其增加到server.stat\_net\_input\_bytes中；

如果是无硬盘复制的RDB数据，则每次read之后，都判断是否接收到了末尾40字节的随机串：如果nread大于等于40，则将buf中后40个字节复制到lastbytes中；否则，将buf复制到lastbytes中的尾部。然后比对lastbytes和eofmark，如果相同，说明已经接收到了末尾，置eof\_reached为1；

然后，将buf写入到描述符server.repl\_transfer\_fd中，也就是从节点保存RDB数据的临时文件中；

然后将nread增加到server.repl\_transfer\_read中，该属性记录了当前已读到的RDB数据的长度；

如果是无硬盘复制的RDB数据，并且已经读到了末尾，则将临时文件中末尾的40字节的随机串删除；

每当读取了8M的数据后，都执行一次sync操作，保证临时文件内容确实写到了硬盘； 如果是有硬盘复制的RDB数据，且server.repl\_transfer\_read等于server.repl\_transfer\_size，则说明已经接收到所有数据，置eof\_reached为1；

如果所有的RDB数据已经接收完了，则首先将保存RDB数据的临时文件改名为配置的RDB文件名server.rdb\_filename；然后调用signalFlushedDb，使得本实例的所有客户端感知到接下来要清空数据库了。然后就是调用emptyDb，清空所有数据，回调函数是replicationEmptyDbCallback，每当处理了字典哈希表中65535个bucket之后，就调用一次该函数，向主节点发送一个"\n"，以向主节点证明本实例还活着；

然后删除server.repl\_transfer\_s上的可读事件，这是因为在调用rdbLoad加载RDB数据时，每次调用rioRead都会调用processEventsWhileBlocked处理当前已触发的事件，如果不删除该可读事件的话，就会递归进入的本函数中（因此，从节点在加载RDB数据时，是不能处理主节点发来的其他数据的）；

接下来就是调用rdbLoad加载RDB数据；

加载完RDB数据之后，就已经完成了完全重同步过程。接下来，从节点会将主节点当成客户端，像处理普通客户端那样，接收主节点发来的命令，执行命令以保证主从一致性。

因此，首先关闭RDB临时文件描述符server.repl\_transfer\_fd，然后就使用socket描述符server.repl\_transfer\_s创建redisClient结构server.master，因此后续还是使用该描述符接收主节点客户端发来的命令；

将标记REDIS\_MASTER记录到客户端标志中，以表明该客户端是主节点；

将复制状态置为REDIS\_REPL\_CONNECTED，表示主从节点已完成握手和接收RDB数据的过程；

主节点之前的发送"PSYNC"命令回复为"+FULLRESYNC"时，附带的初始复制偏移记录到了server.repl\_master\_initial\_offset中，将其保存到server.master->reploff；附带的主节点ID记录到了server.repl\_master\_runid中，将其保存到server.master->replrunid中；如果server.repl\_master\_initial\_offset为-1，说明主节点不认识"PSYNC"命令，因此将REDIS\_PRE\_PSYNC记录到客户端标志位中；

完成以上的操作之后，如果本实例开启了AOF功能，则首先调用stopAppendOnly，然后循环10次，调用startAppendOnly开始进行AOF转储，直到startAppendOnly返回REDIS\_OK。如果startAppendOnly失败次数超过10次，则直接exit退出！！！

**六：命令传播**

当复制状态变为REDIS\_REPL\_CONNECTED后，表示进入了命令传播阶段。后续从节点将主节点当成一个客户端，接收该主节点客户端发来的命令请求，像处理普通客户端一样处理即可。

在读取客户端命令的函数readQueryFromClient中，一旦从节点读到了追节点发来的同步命令，会将命令长度增加到从节点的复制偏移量server.master. reploff中：

**if** **(**nread**)** **{**

sdsIncrLen**(**c**->**querybuf**,**nread**);**

c**->**lastinteraction **=** server**.**unixtime**;**

**if** **(**c**->**flags **&** REDIS\_MASTER**)** c**->**reploff **+=** nread**;**

server**.**stat\_net\_input\_bytes **+=** nread**;**

**}**

这样，从节点的复制偏移量server.master. reploff就能与主节点保持一致了。

与普通客户端不同的是，主节点客户端发来的命令请求无需回复，因此，在函数prepareClientToWrite中，有下面的语句：

int prepareClientToWrite**(**redisClient **\***c**)** **{**

**...**

*/\* Masters don't receive replies, unless REDIS\_MASTER\_FORCE\_REPLY flag*

*\* is set. \*/*

**if** **((**c**->**flags **&** REDIS\_MASTER**)** **&&**

**!(**c**->**flags **&** REDIS\_MASTER\_FORCE\_REPLY**))** **return** REDIS\_ERR**;**

**...**

**}**

每次向客户端输出缓存追加新数据之前，都要调用函数prepareClientToWrite函数。如果该函数返回REDIS\_ERR，表示无需向输出缓存追加新数据。

客户端标志中如果设置了REDIS\_MASTER标记，就表示该客户端是主节点客户端server.master，并且在没有设置REDIS\_MASTER\_FORCE\_REPLY标记的情况下，该函数返回REDIS\_ERR，表示无需向输出缓存追加新数据。

其他有关主从复制的代码，可以参考：

https://github.com/gqtc/redis-3.0.5/blob/master/redis-3.0.5/src/replication.c