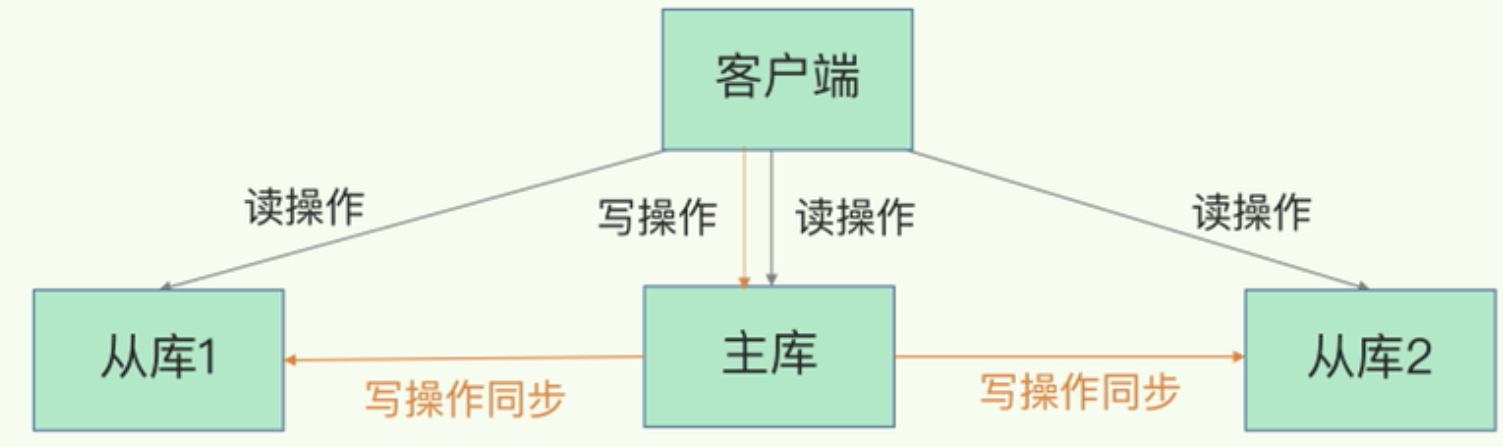
数据同步

Redis提供了主从库模式，以保证数据副本的一致，主从库之间采用的是读写分离的方式。

• 读操作：主库、从库都可以接收；

• 写操作：首先到主库执行，然后，主库将写操作同步给从库。



一、主从库间第一次同步

启动多个Redis实例时，它们相互之间就可以通过replicaof（Redis 5.0之前使用slaveof）命令形成主库和从库的关系，之后会按照三个阶段完成数据的第一次同步。



1.1 主从库间建立连接、协商同步

主要是为全量复制做准备。在这一步，从库和主库建立起连接，并告诉主库即将进行同步，主库确认回复后，主从库间就可以开始同步了。

- [ ] 从库给主库发送psync命令，表示要进行数据同步，主库根据这个命令的参数来启动复制。psync命令包含了主库的runID和复制进度offset两个参数。

• runID，是每个Redis实例启动时都会自动生成的一个随机ID，用来唯一标记这个实例。当从库和主库第一次复制时，因为不知道主库的runID，所以将runID设为“？”。

• offset，此时设为-1，表示第一次复制。

- [ ] 主库收到psync命令后，会用FULLRESYNC响应命令带上两个参数：主库runID和主库目前的复制进度offset，返回给从库。从库收到响应后，会记录下这两个参数。FULLRESYNC响应表示第一次复制采用的全量复制。

1.2主库将所有数据同步给从库

从库收到数据后，在本地完成数据加载。这个过程依赖于内存快照生成的RDB文件。

- [ ] 主库执行bgsave命令，生成RDB文件，接着将文件发给从库。

- [ ] 从库接收到RDB文件后，会先清空当前数据库，然后加载RDB文件。这是因为从库在通过replicaof命令开始和主库同步前，可能保存了其他数据。为了避免之前数据的影响，从库需要先把当前数据库清空。

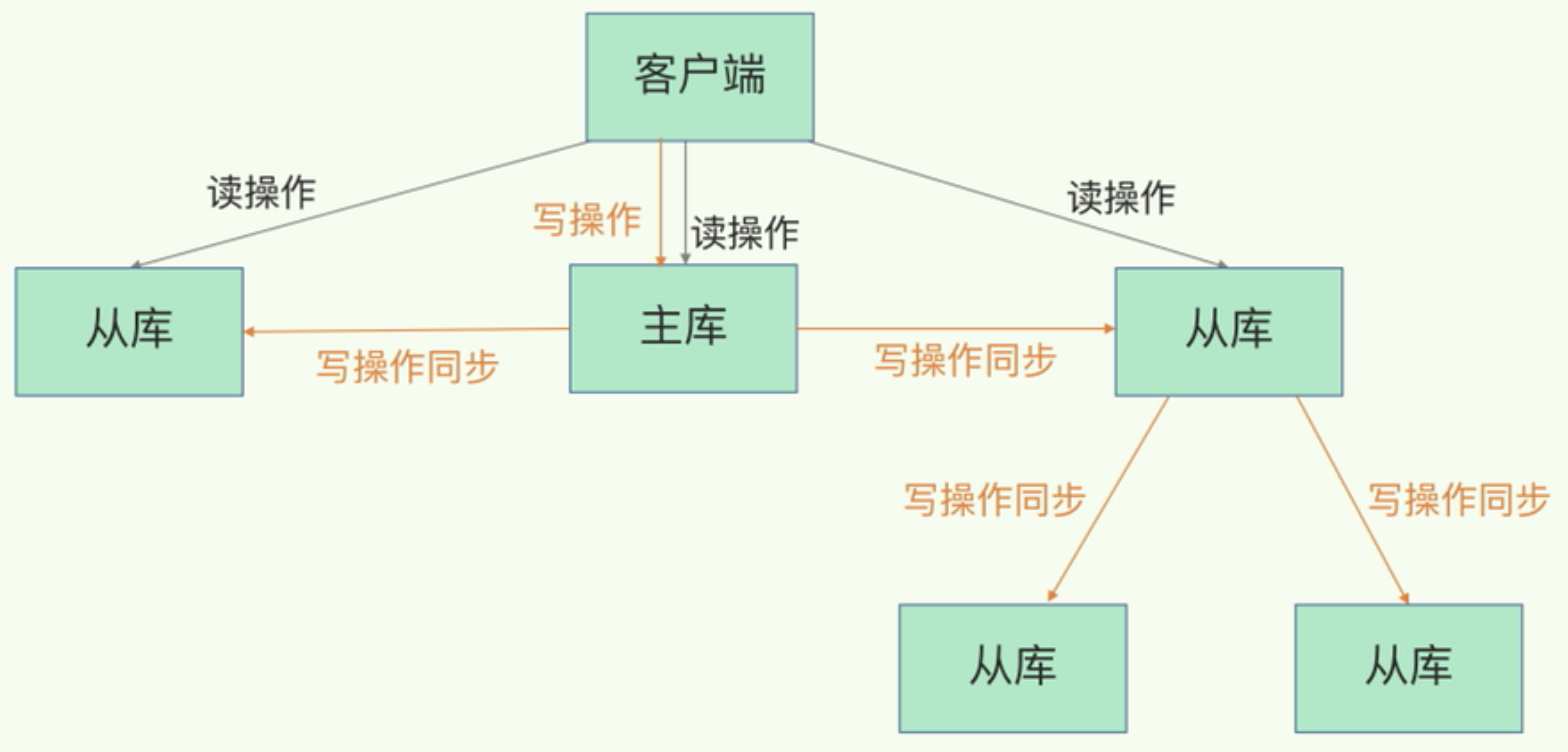
- [ ] 在主库将数据同步给从库的过程中，主库不会被阻塞，仍然可以正常接收请求。但是，这些请求中的写操作并没有记录到刚刚生成的RDB文件中。为了保证主从库的数据一致性，主库会在内存中用专门的replication buffer，记录RDB文件生成后收到的所有写操作。

1.3 主库把第二阶段执行过程中新收到的写命令发送给从库

- [ ] 当主库完成RDB文件发送后，就会把此时replication buffer中的修改操作发给从库，从库再重新执行这些操作。

二、主从级联模式分担全量复制时的主库压力

通过“主-从-从”模式将主库生成RDB和传输RDB的压力，以级联的方式分散到从库上。



一旦主从库完成了全量复制，它们之间就会一直维护一个网络连接，主库会通过这个连接将后续陆续收到的命令操作再同步给从库，这个过程也称为基于长连接的命令传播，可以避免频繁建立连接的开销。

三、主从库间网络断连

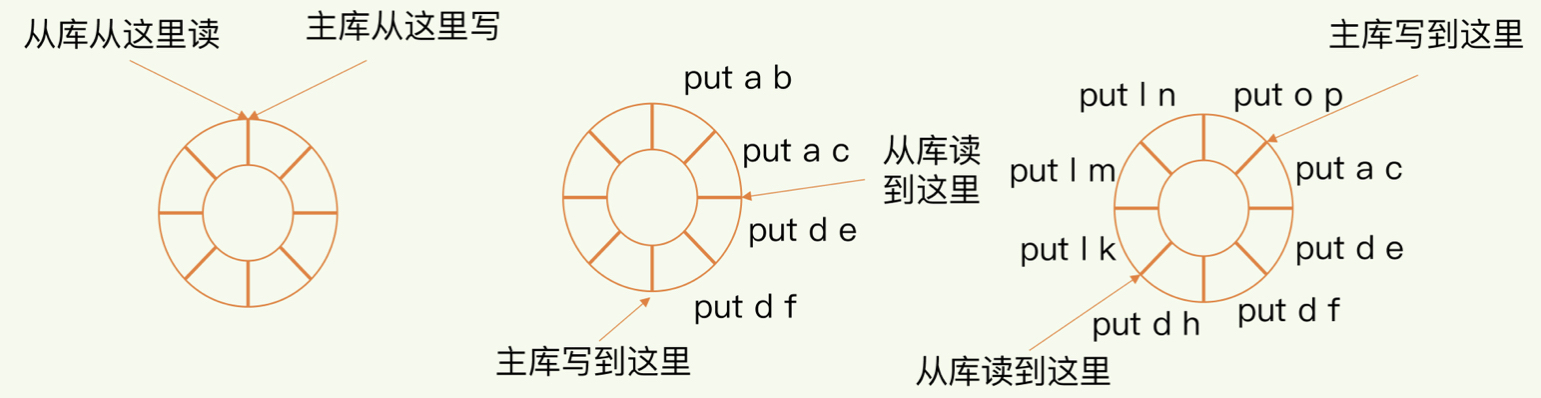
在Redis 2.8之前，如果主从库在命令传播时出现了网络闪断，那么，从库就会和主库重新进行一次全量复制，开销非常大。

从Redis 2.8开始，网络断了之后，主从库会采用增量复制的方式，只会把主从库网络断连期间主库收到的命令，同步给从库。

- [ ] 当主从库断连后，主库会把断连期间收到的写操作命令，写入replication buffer，同时也会把这些操作命令也写入repl\_backlog\_buffer这个缓冲区。repl\_backlog\_buffer是一个环形缓冲区，主库会记录自己写到的位置，从库则会记录自己已经读到的位置。

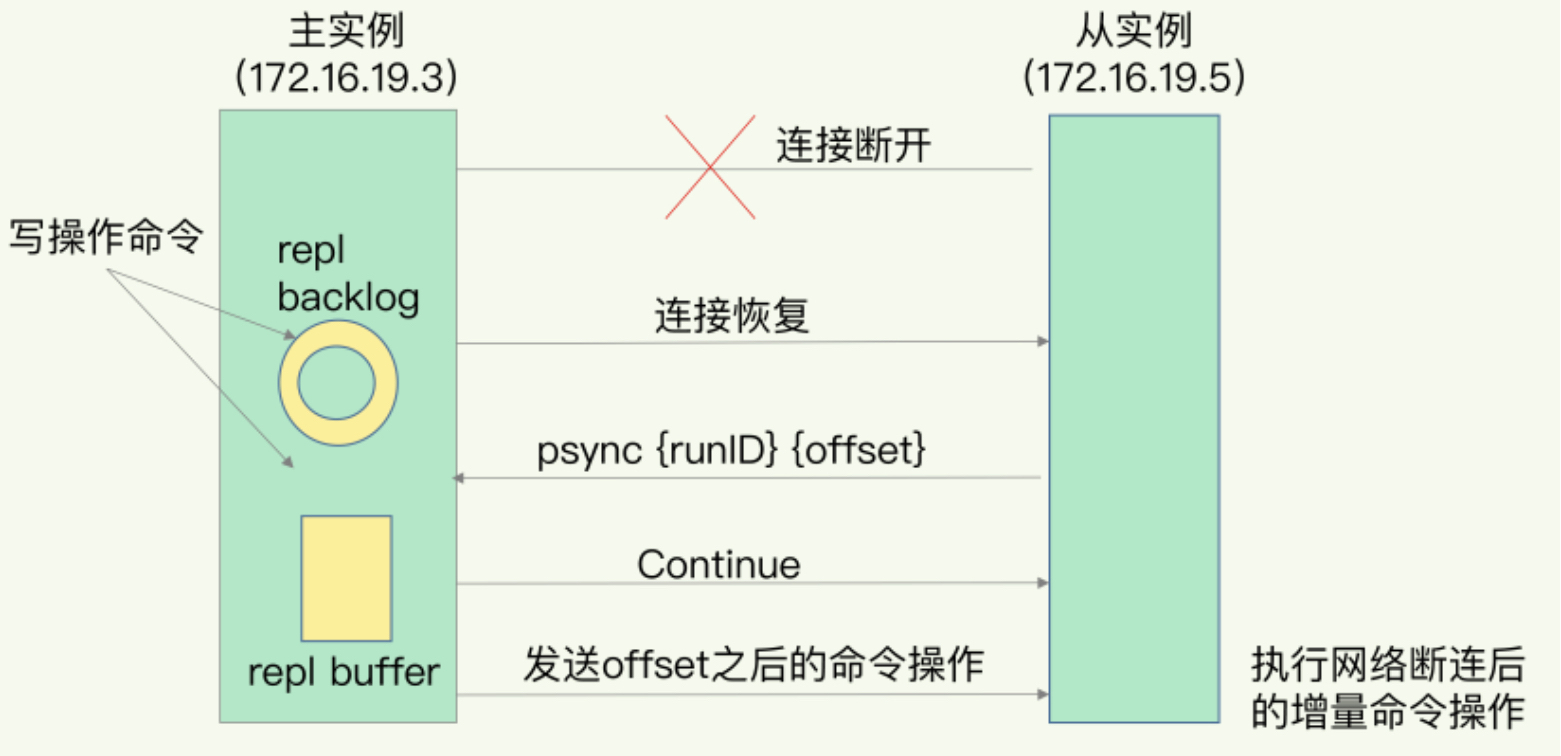
- [ ] 刚开始的时候，主库和从库的写读位置在一起，这算是它们的起始位置。随着主库不断接收新的写操作，它在缓冲区中的写位置会逐步偏离起始位置，我们通常用偏移量来衡量这个偏移距离的大小，对主库来说，对应的偏移量就是master\_repl\_offset。主库接收的新写操作越多，这个值就会越大。

- [ ] 从库在复制完写操作命令后，它在缓冲区中的读位置也开始逐步偏移刚才的起始位置，此时，从库已复制的偏移量slave\_repl\_offset也在不断增加。正常情况下，这两个偏移量基本相等



- [ ] 主从库的连接恢复之后，从库首先会给主库发送psync命令，并把自己当前的slave\_repl\_offset发给主库，主库会判断自己的master\_repl\_offset和slave\_repl\_offset之间的差距。

- [ ] 在网络断连阶段，主库可能会收到新的写操作命令，所以，一般来说，master\_repl\_offset会大于slave\_repl\_offset。此时，主库只用把master\_repl\_offset和slave\_repl\_offset之间的命令操作同步给从库就行。



因为repl\_backlog\_buffer是一个环形缓冲区，所以在缓冲区写满后，主库会继续写入，此时，就会覆盖掉之前写入的操作。如果从库的读取速度比较慢，就有可能导致从库还未读取的操作被主库新写的操作覆盖了，这会导致主从库间的数据不一致。

可以调整repl\_backlog\_size这个参数。这个参数和所需的缓冲空间大小有关。缓冲空间的计算公式是：缓冲空间大小 = 主库写入命令速度 \* 操作大小 - 主从库间网络传输命令速度 \* 操作大小。在实际应用中，考虑到可能存在一些突发的请求压力，我们通常需要把这个缓冲空间扩大一倍，即repl\_backlog\_size = 缓冲空间大小 \* 2。

四、脑裂

指在主从集群中，同时有两个主节点，它们都能接收写请求。而脑裂最直接的影响，就是客户端不知道应该往哪个主节点写入数据，结果就是不同的客户端会往不同的主节点上写入数据。而且，严重的话，脑裂会进一步导致数据丢失。

4.1 发生脑裂的原因

4.1.1 确认是不是数据同步出现了问题

在主从集群中发生数据丢失，最常见的原因就是主库的数据还没有同步到从库，结果主库发生了故障，等从库升级为主库后，未同步的数据就丢失了。

如果是这种情况的数据丢失，可以通过比对主从库上的复制进度差值来进行判断，也就是计算master\_repl\_offset和slave\_repl\_offset的差值。如果从库上的slave\_repl\_offset小于原主库的master\_repl\_offset，那么，就可以认定数据丢失是由数据同步未完成导致的。

4.1.2 排查客户端的操作日志

在主从切换后的一段时间内，有一个客户端仍然在和原主库通信，并没有和升级的新主库进行交互。这就相当于主从集群中同时有了两个主库。

4.1.3 发现是原主库假故障

采用哨兵机制进行主从切换，当主从切换发生时，一定是有超过预设数量（quorum配置项）的哨兵实例和主库的心跳都超时了，才会把主库判断为客观下线，然后，哨兵开始执行切换操作。哨兵切换完成后，客户端会和新主库进行通信，发送请求操作。

但是，在切换过程中，既然客户端仍然和原主库通信，这就表明，原主库并没有真的发生故障（例如主库进程挂掉）。主库是由于某些原因无法处理请求，也没有响应哨兵的心跳，才被哨兵错误地判断为客观下线的。结果，在被判断下线之后，原主库又重新开始处理请求了，而此时，哨兵还没有完成主从切换，客户端仍然可以和原主库通信，客户端发送的写操作就会在原主库上写入数据了。

4.2 脑裂导致数据丢失的原因

主从切换后，从库一旦升级为新主库，哨兵就会让原主库执行slave of命令，和新主库重新进行全量同步。而在全量同步执行的最后阶段，原主库需要清空本地的数据，加载新主库发送的RDB文件，这样一来，原主库在主从切换期间保存的新写数据就丢失了。

4.3 应对脑裂问题

Redis提供了两个配置项来限制主库的请求处理

- [ ] min-slaves-to-write：主库能进行数据同步的最少从库数量；

- [ ] min-slaves-max-lag：主从库间进行数据复制时，从库给主库发送ACK消息的最大延迟（以秒为单位）。

主库连接的从库中至少有N个从库，和主库进行数据复制时的ACK消息延迟不能超过T秒，否则，主库就不会再接收客户端的请求了。

即使原主库是假故障，它在假故障期间也无法响应哨兵心跳，也不能和从库进行同步，自然也就无法和从库进行ACK确认了。这样一来，min-slaves-to-write和min-slaves-max-lag的组合要求就无法得到满足，原主库就会被限制接收客户端请求，客户端也就不能在原主库中写入新数据了。

等到新主库上线时，就只有新主库能接收和处理客户端请求，此时，新写的数据会被直接写到新主库中。而原主库会被哨兵降为从库，即使它的数据被清空了，也不会有新数据丢失。