06-数据同步：主从库如何实现数据一致？

你好，我是蒋德钧。

前两节课，我们学习了AOF和RDB，如果Redis发生了宕机，它们可以分别通过回放日志和重新读入RDB文件的方式恢复数据，从而保证尽量少丢失数据，提升可靠性。

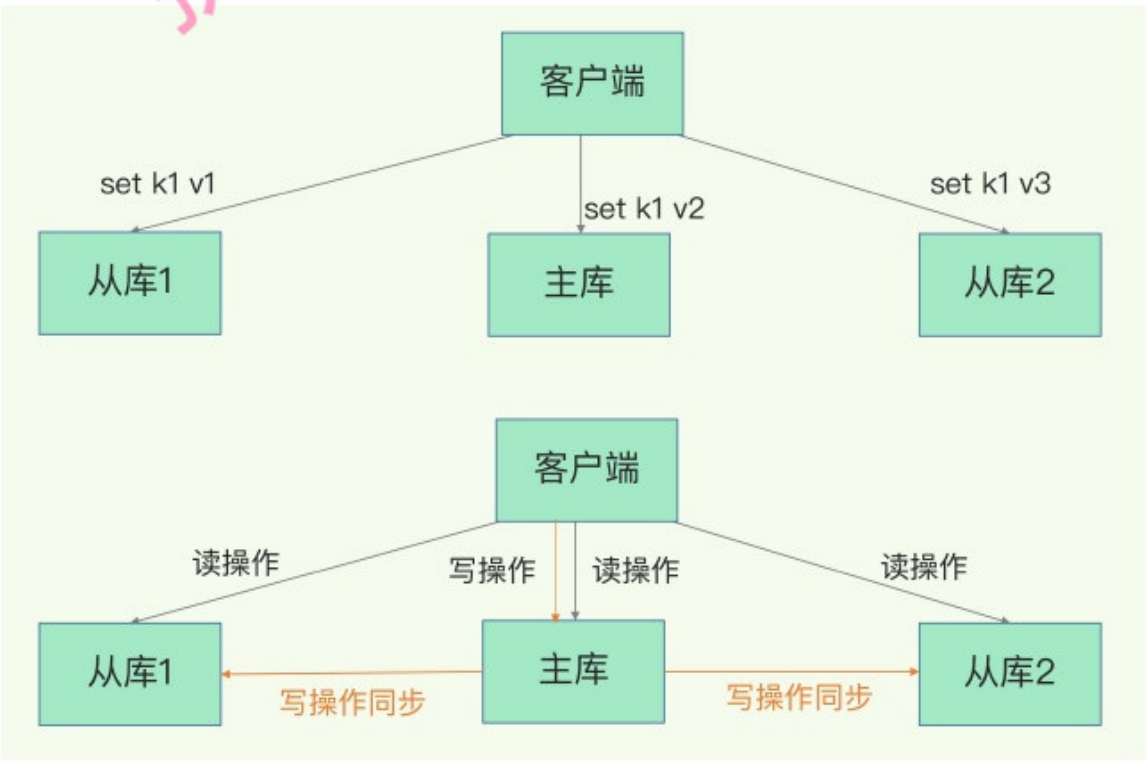
不过，即使用了这两种方法，也依然存在服务不可用的问题。比如说，我们在实际使用时只运行了一个Redis实例，那么，如果这个实例宕机了，它在恢复期间，是无法服务新来的数据存取请求的。

那我们总说的Redis具有高可靠性，又是什么意思呢？其实，这里有两层含义：一是**数据尽量少丢失**，二是**服务尽量少中断**。AOF和RDB保证了前者，而对于后者，Redis的做法就是**增加副本冗余量**，将一份数据同时保存在多个实例上。即使有一个实例出现了故障，需要过一段时间才能恢复，其他实例也可以对外提供服务，不会影响业务使用。

多实例保存同一份数据，听起来好像很不错，但是，我们必须要考虑一个问题：这么多副本，它们之间的数据如何保持一致呢？数据读写操作可以发给所有的实例吗？

实际上，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。



那么，为什么要采用读写分离的方式呢？

你可以设想一下，如果在上图中，不管是主库还是从库，都能接收客户端的写操作，那么，一个直接的问题就是：如果客户端对同一个数据（例如k1）前后修改了三次，每一次的修改请求都发送到不同的实例上，在

不同的实例上执行，那么，这个数据在这三个实例上的副本就不一致了（分别是v1、v2和v3）。在读取这个数据的时候，就可能读取到旧的值。

如果我们非要保持这个数据在三个实例上一致，就要涉及到加锁、实例间协商是否完成修改等一系列操作，但这会带来巨额的开销，当然是不太能接受的。

而主从库模式一旦采用了读写分离，所有数据的修改只会在主库上进行，不用协调三个实例。主库有了最新的数据后，会同步给从库，这样，主从库的数据就是一致的。

那么，主从库同步是如何完成的呢？主库数据是一次性传给从库，还是分批同步？要是主从库间的网络断连了，数据还能保持一致吗？这节课，我就和你聊聊主从库同步的原理，以及应对网络断连风险的方案。

好了，我们先来看看主从库间的第一次同步是如何进行的，这也是Redis实例建立主从库模式后的规定动作。

主从库间如何进行第一次同步？

当我们启动多个Redis实例的时候，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。

例如，现在有实例1（ip：172.16.19.3）和实例2（ip：172.16.19.5），我们在实例2上执行以下这个命令后，实例2就变成了实例1的从库，并从实例1上复制数据：

```
replicaof  172.16.19.3  6379
```

接下来，我们就要学习主从库间数据第一次同步的三个阶段了。你可以先看一下下面这张图，有个整体感知，接下来我再具体介绍。

第一阶段是主从库间建立连接、协商同步的过程，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**。

具体来说，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。

这里有个地方需要注意，**FULLRESYNC响应表示第一次复制采用的全量复制，也就是说，主库会把当前所有的数据都复制给从库**。

在第二阶段，**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。

具体来说，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。否则，Redis的服务就被中断了。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。

最后，也就是第三个阶段，主库会把第二阶段执行过程中新收到的写命令，再发送给从库。具体的操作是，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。这样一来，主从库就实现同步了。

主从级联模式分担全量复制时的主库压力

通过分析主从库间第一次数据同步的过程，你可以看到，一次全量复制中，对于主库来说，需要完成两个耗时的操作：生成RDB文件和传输RDB文件。

如果从库数量很多，而且都要和主库进行全量复制的话，就会导致主库忙于fork子进程生成RDB文件，进行数据全量同步。fork这个操作会阻塞主线程处理正常请求，从而导致主库响应应用程序的请求速度变慢。此外，传输RDB文件也会占用主库的网络带宽，同样会给主库的资源使用带来压力。那么，有没有好的解决方法可以分担主库压力呢？

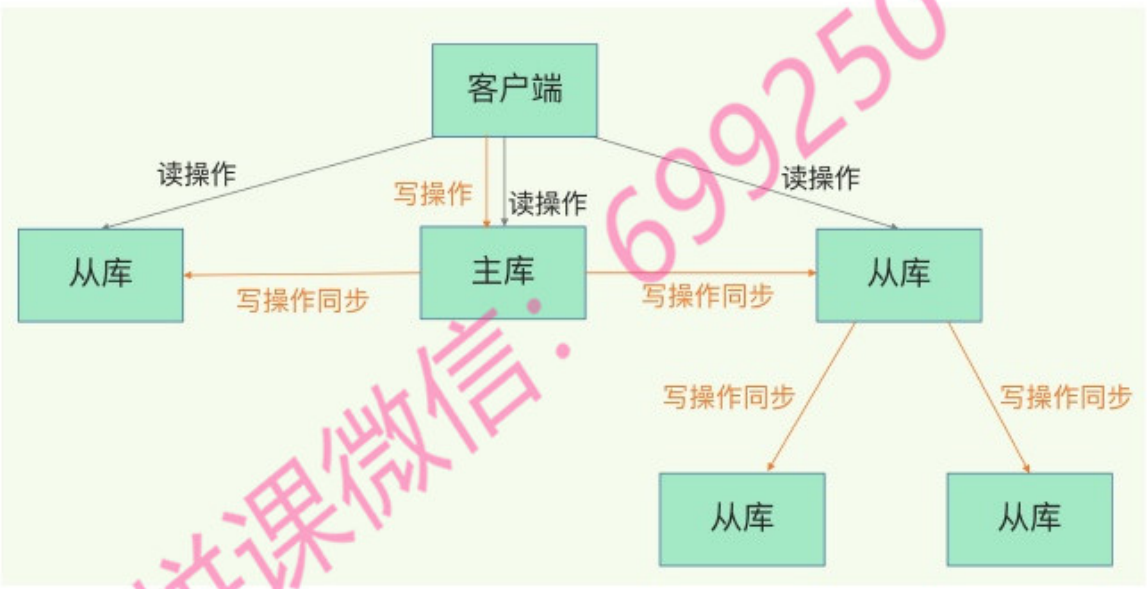
其实是有的，这就是“主-从-从”模式。

在刚才介绍的主从库模式中，所有的从库都是和主库连接，所有的全量复制也都是和主库进行的。现在，我们可以**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**。

简单来说，我们在部署主从集群的时候，可以手动选择一个从库（比如选择内存资源配置较高的从库），用于级联其他的从库。然后，我们可以再选择一些从库（例如三分之一的从库），在这些从库上执行如下命令，让它们和刚才所选的从库，建立起主从关系。

```
replicaof  所选从库的IP 6379
```

这样一来，这些从库就会知道，在进行同步时，不用再和主库进行交互了，只要和级联的从库进行写操作同步就行了，这就可以减轻主库上的压力，如下图所示：



好了，到这里，我们了解了主从库间通过全量复制实现数据同步的过程，以及通过“主-从-从”模式分担主库压力的方式。那么，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。

听上去好像很简单，但不可忽视的是，这个过程中存在着风险点，最常见的就是**网络断连或阻塞**。如果网络断连，主从库之间就无法进行命令传播了，从库的数据自然也就没办法和主库保持一致了，客户端就可能从从库读到旧数据。

接下来，我们就来聊聊网络断连后的解决办法。

主从库间网络断了怎么办？

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继续同步。听名字大概就可以猜到它和全量复制的不同：全量复制是同步所有数据，而增量复制只会把主从库网络断连期间主库收到的命令，同步给从库。

那么，增量复制时，主从库之间具体是怎么保持同步的呢？这里的奥妙就在于repl_backlog_buffer这个缓冲区。我们先来看下它是如何用于增量命令的同步的。

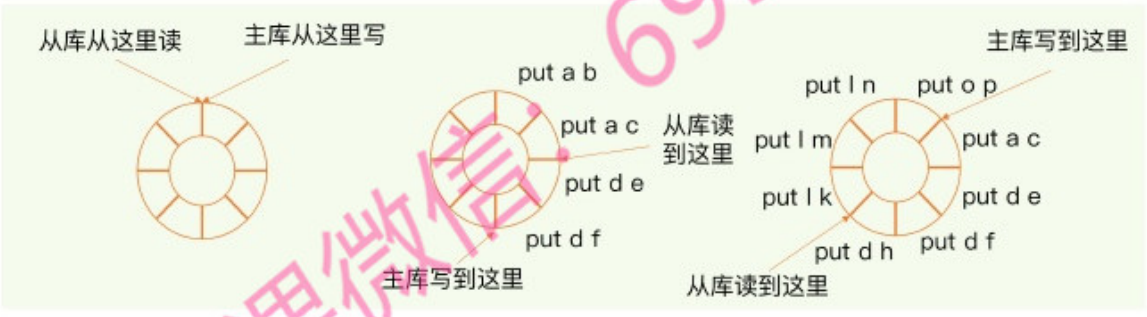
当主从库断连后，主库会把断连期间收到的写操作命令，写入replication buffer，同时也会把这些操作命令

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。

同样，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。

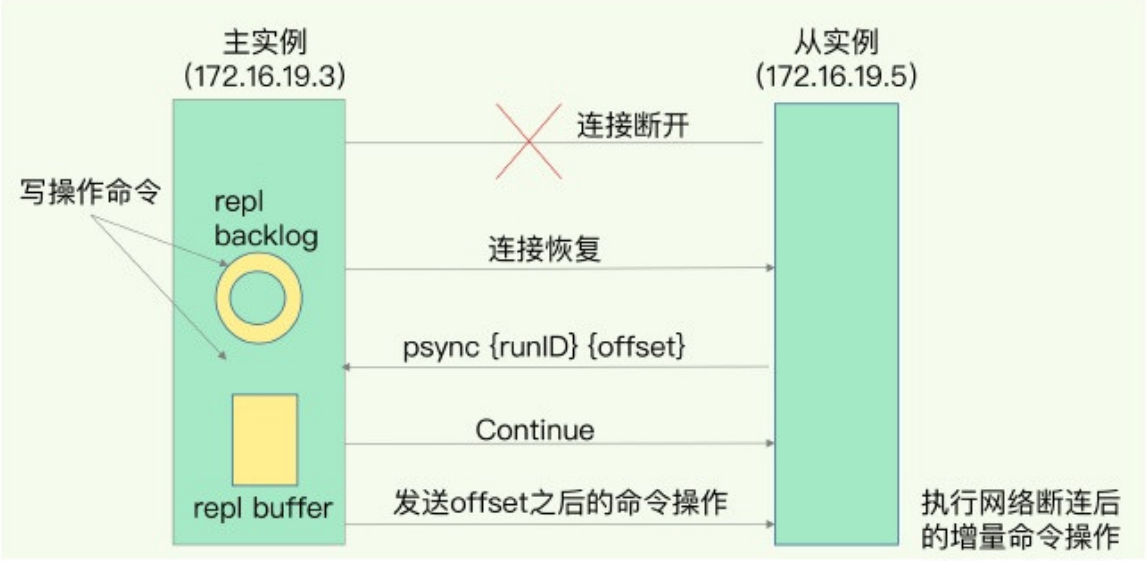


主从库的连接恢复之后，从库首先会给主库发送psync命令，并把自己当前的slave_repl_offset发给主库，主库会判断自己的master_repl_offset和slave_repl_offset之间的差距。

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。

就像刚刚示意图的中间部分，主库和从库之间相差了put d e和put d f两个操作，在增量复制时，主库只需要把它们同步给从库，就行了。

说到这里，我们再借助一张图，回顾下增量复制的流程。



不过，有一个地方我要强调一下，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**。

因此，我们要想办法避免这一情况，一般而言，我们可以调整**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，这也就是repl_backlog_size的最终值。

举个例子，如果主库每秒写入2000个操作，每个操作的大小为2KB，网络每秒能传输1000个操作，那么，有1000个操作需要缓冲起来，这就至少需要2MB的缓冲空间。否则，新写的命令就会覆盖掉旧操作了。为了应对可能的突发压力，我们最终把repl_backlog_size设为4MB。

这样一来，增量复制时主从库的数据不一致风险就降低了。不过，如果并发请求量非常大，连两倍的缓冲空间都存不下新操作请求的话，此时，主从库数据仍然可能不一致。

针对这种情况，一方面，你可以根据Redis所在服务器的内存资源再适当增加repl_backlog_size值，比如说设置成缓冲空间大小的4倍，另一方面，你可以考虑使用切片集群来分担单个主库的请求压力。关于切片集群，我会在第9讲具体介绍。

小结

这节课，我们一起学习了Redis的主从库同步的基本原理，总结来说，有三种模式：全量复制、基于长连接的命令传播，以及增量复制。

全量复制虽然耗时，但是对于从库来说，如果是第一次同步，全量复制是无法避免的，所以，我给你一个小建议：**一个Redis实例的数据库不要太大**，一个实例大小在几GB级别比较合适，这样可以减少RDB文件生成、传输和重新加载的开销。另外，为了避免多个从库同时和主库进行全量复制，给主库过大的同步压力，我们也可以采用“主-从-从”这一级联模式，来缓解主库的压力。

长连接复制是主从库正常运行后的常规同步阶段。在这个阶段中，主从库之间通过命令传播实现同步。不过，这期间如果遇到了网络断连，增量复制就派上用场了。我特别建议你留意一下repl_backlog_size这个配置参数。如果它配置得过小，在增量复制阶段，可能会导致从库的复制进度赶不上主库，进而导致从库重新进行全量复制。所以，通过调大这个参数，可以减少从库在网络断连时全量复制的风险。

不过，主从库模式使用读写分离虽然避免了同时写多个实例带来的数据不一致问题，但是还面临主库故障的潜在风险。主库故障了从库该怎么办，数据还能保持一致吗，Redis还能正常提供服务吗？在接下来的两节课里，我会和你具体聊聊主库故障后，保证服务可靠性的解决方案。

每课一问

按照惯例，我给你提一个小问题。这节课，我提到，主从库间的数据复制同步使用的是RDB文件，前面我们学习过，AOF记录的操作命令更全，相比于RDB丢失的数据更少。那么，为什么主从库间的复制不使用AOF呢？

好了，这节课就到这里，如果你觉得有收获，欢迎你帮我把今天的内容分享给你的朋友。

精选留言：

- Geek_121747 2020-08-17 06:53:26
  每课一问：1.因为AOF文件比RDB文件大，网络传输比较耗时，
  2.从库在初始化数据时，RDB文件比AOF文件执行更快