高可用（1）——深入理解Redis的主从复制原理！

场景切入：

在今天话题开始之前，我们先来看这么一个场景：

如果我把数据都存在一台服务器上，那可能会发生什么不可挽救的场面？比如：

- 宕机了……那在此期间如果还来不及将数据持久化，妥妥的数据丢失
- 持久化文件是放在磁盘里的这个毋庸置疑，但是如果磁盘他老人家蚌了，那数据都丢失，你也差不多得蚌了。

路人甲：那简单，多用几台服务器就完了呗，多大点事

路人乙：你倒是站着说话不腰疼，这么多台服务器，我在这查是个a，在那边查是个b，那到底是a还是b？？？？

重点来喽！

主角：主从复制

1.介绍

主从复制是Redis分布式的基石，也是Redis高可用的保障。在Redis中，被复制的服务器称为主服务器（Master），对主服务器进行复制的服务器称为从服务器（Slave）。

2. 第一次同步

我将第一次同步主要分为六个步骤：每一个小点分别都和图上对应上，建议就图食用

①从服务器发起请求

```
replicaof <Master 的 IP 地址> <Master 的 Redis 端口号>
```

此时从服务器向主服务器发起PSYNC请求

- ?：代表的是主服务器的**runID**，但因为此时的从服务器并不知道主服务器的runID（每一台Redis服务器在启动的时候会产生一个随机ID，可以用来标识自己）,所以用了占位符?
- -1 :这个指的是主服务器复制进度**offset**，因此此时还没开始复制，所以使用的是-1,此时的-1也是标识着这是主从服务器的第一次同步，

```
psync ? -1
```

②主服务器的响应

主服务器会用自己的runID替代掉原先的?占位符，再把主服务器当前的复制进度替代掉原来的-1，

当然，fullresync语句一出来，就意味着采用的是**全量复制**，当然，与此对应的当然还有一个**增量复制**咯，至于这个是什么，后面会讲到。

```
fullresync <runID> <offset>
```

③主服务器执行bgsave命令产生RDB文件

其实就是主进程fork出了一个子进程，由子进程来完成RDB的复制。所以这个时候是不会阻塞主进程的操作的，真正会产生阻塞的——是主进程在fork子进程的过程。

灵魂拷问：这时候如果有写操作怎么办？我是写到原来的RDB文件呢，还是不写呢?

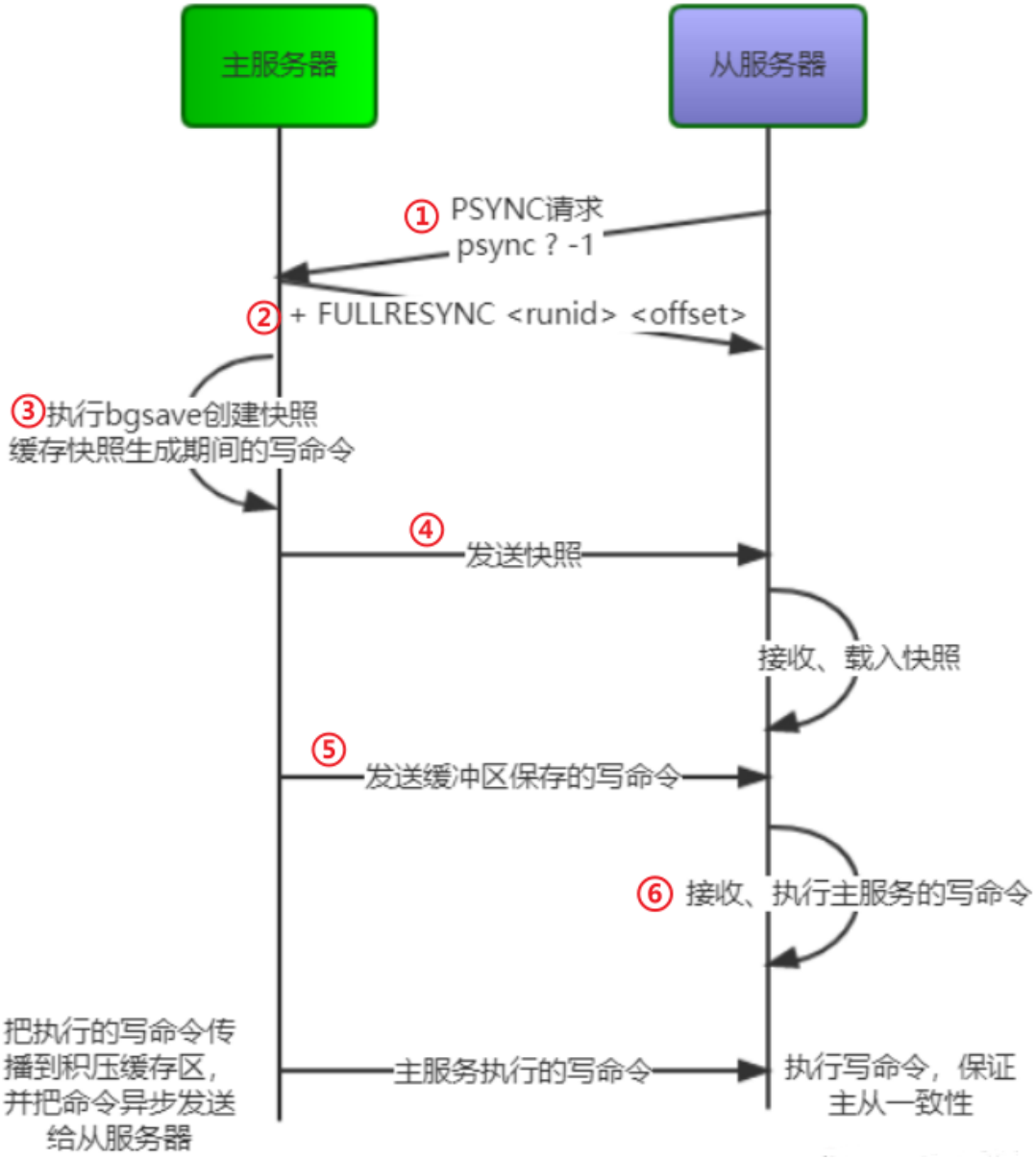
答案是：不写，诶~~我写另一个地方

在redis中还有一个缓冲区，叫做replication buffer ，那在这个过程中我就会先把数据存在这个缓冲区里，直到RBD文件在从服务器写完后，就有它的一席之地咯。

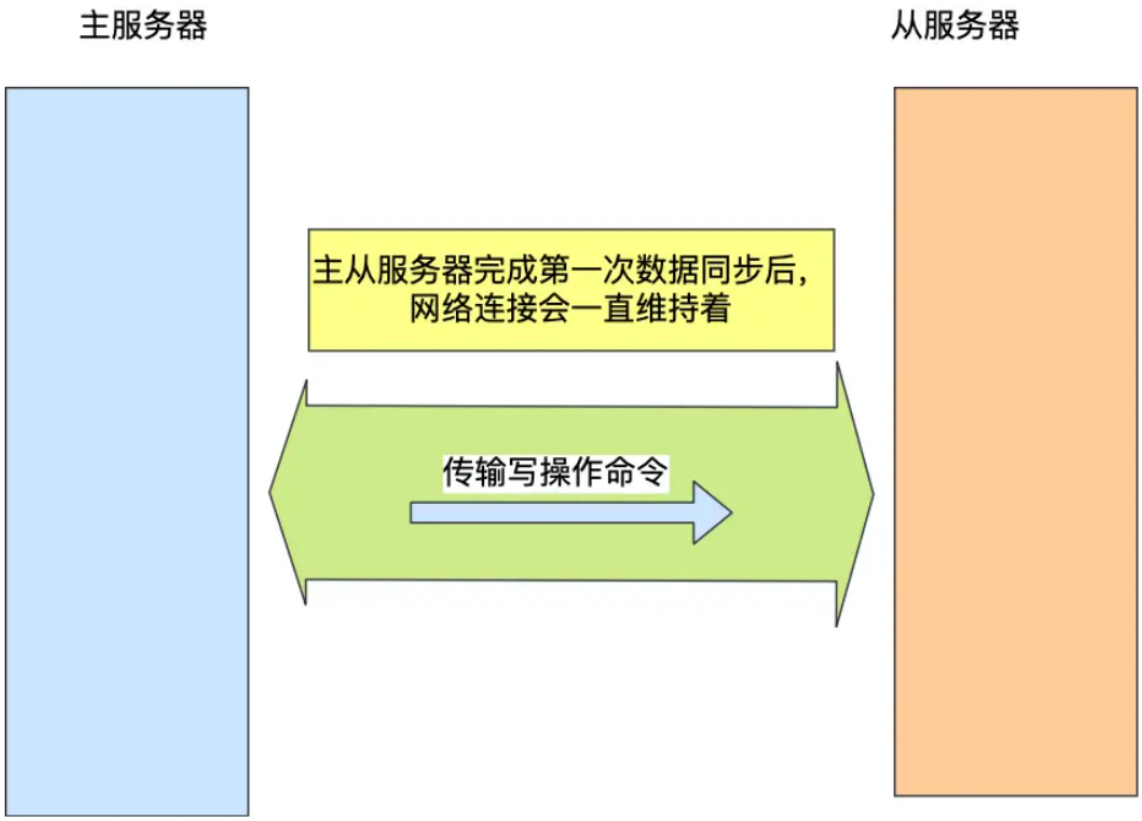
④发送RDB文件给从服务器，从服务器写入RDB文件

⑤发送缓冲区的命令

这时候就是我们的replication buffer缓冲区发挥作用了，此时会把在这个期间执行的写入操作同步给从服务器，至此，第一次同步的过程就结束啦！



3. 每一次同步都这么复杂？——命令传播



- 如果说每一次的同步都得确认下对方身份，那整个数据的同步估计够呛，因此主从服务器是处于一种长连接的状态，能够避免频繁的TCP连接和断开带来的性能开销，这也称为"基于长连接的命令传播",很明显，后面的主从同步是基于此实现的

连接断了这么搞?

主从服务器在完成第一次同步后，就会基于长连接进行命令传播。但是，我们也知道网络延迟问题又或者是断开，都是随时可能会发生的事情，那如果断开了，主从服务器之间的数据就会出现不一致的问题，那如果说情况不算那么糟糕，网络恢复正常了，我要如何解决主从数据不一致问题呢？全量复制？在Redis2.8之前确实是这样子，很明显这很浪费资源，于是又提出了一个新的观点：**增量复制**

一步一步来，我们先来看看网络恢复后会怎么做。

①从服务器会给主服务器发送

```
psync <runID> <offset>
```

②关键来了：主从服务器的再次同步

我们先来认识上面的offset所处的结构：repl_backlog_buffer ，它是一个环形的结构（整个主从复制的过程中只会出现一个，也就是多个Slave公用的）,如图，会存放主服务器的进度和从服务器的进度，所以在第一步收到offset的时候，会判断是否在主服务器的进度后面，如果是则要返回

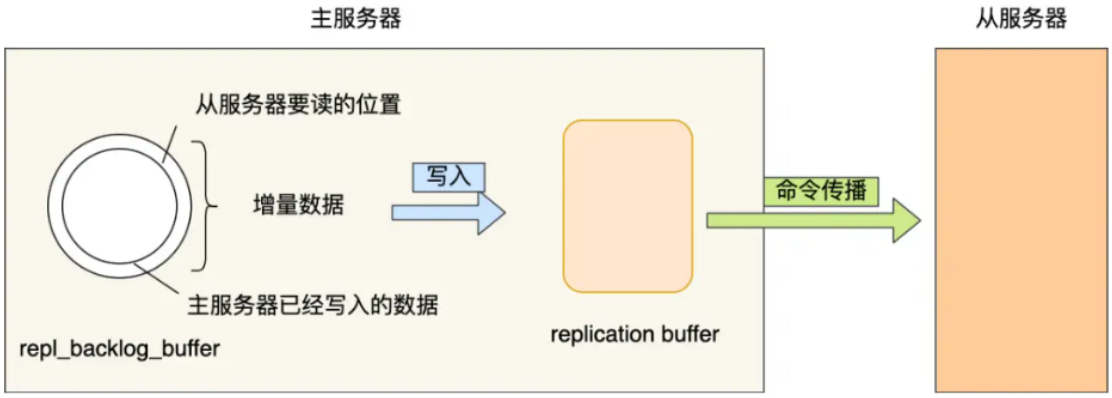
```
continue
```

说明采用的是增量复制，增量数据就是两个offset的差值了，此时就会把数据写入到replication buffer ，通过命令传播的形式来完成主从服务器的同步。

- 那顺便来讲讲这个replication buffer，他不像repl_backlog_buffer是公有的，相反，有多少台从服务器就有多少个replication buffer，说到底是因为从服务器和主服务器的同步内容不可能都一模一样，而这个缓冲区又是实现命令传播的前提，所以replication buffer是为每台从服务器“量身定做的”。这个缓冲区也是有限制的，可以通过**client-output-buffer-limit**参数来调整这个值的大小，如果replication buffer满了，是会导致从服务器断开连接的。

值得说明的是，repl_backlog_buffer其实很小，默认只有1M,而且它是覆盖式的写入，所以很有可能会出现一个问题：从服务器要读取的数据已经不存在 repl_backlog_buffer 缓冲区里，那么主服务器会返回以下语句，表示采用**全量复制**的方式。

```
FULLRESYNC < runid > < offset >
```



当然，我们要避免全量复制的情况，这要这么搞嘞？

当然是在配置文件修改咯

```
repl-backlog-size 1mb
```

那这个值要怎么确定呢？来看一个公式

```
second * write_size_per_second
second:表示你断开连接的时间
write_size_per_second:主服务器平均每秒产生的写命令数据量大小
```

所以很明显啦，如果想要降低全量复制发生的可能性，就要使得repl_backlog_buffer的大小至少要满足上面的公式。

以上就是Redis主从复制的内容啦！！！希望对你们有所帮助

【文章所选用的图片均来自网络】