32-Redis主从同步与故障切换，有哪些坑？

你好，我是蒋德钧。

Redis的主从同步机制不仅可以让从库服务更多的读请求，分担主库的压力，而且还能在主库发生故障时，进行主从库切换，提供高可靠服务。

不过，在实际使用主从机制的时候，我们很容易踩到一些坑。这节课，我就向你介绍3个坑，分别是主从数据不一致、读到过期数据，以及配置项设置得不合理从而导致服务挂掉。

一旦踩到这些坑，业务应用不仅会读到错误数据，而且很可能会导致Redis无法正常使用，我们必须要全面地掌握这些坑的成因，提前准备一套规避方案。不过，即使不小心掉进了陷阱里，也不要担心，我还会给你介绍相应的解决方案。

好了，话不多说，下面我们先来看看第一个坑：主从数据不一致。

主从数据不一致

主从数据不一致，就是指客户端从从库中读取到的值和主库中的最新值并不一致。

举个例子，假设主从库之前保存的用户年龄值是19，但是主库接收到了修改命令，已经把这个数据更新为20了，但是，从库中的值仍然是19。那么，如果客户端从从库中读取用户年龄值，就会读到旧值。

那为啥会出现这个坑呢？其实这是因为**主从库间的命令复制是异步进行的**。

具体来说，在主从库命令传播阶段，主库收到新的写命令后，会发送给从库。但是，主库并不会等到从库实际执行完命令后，再把结果返回给客户端，而是主库自己在本地执行完命令后，就会向客户端返回结果了。如果从库还没有执行主库同步过来的命令，主从库间的数据就不一致了。

那在什么情况下，从库会滞后执行同步命令呢？其实，这里主要有两个原因。

一方面，主从库间的网络可能会有传输延迟，所以从库不能及时地收到主库发送的命令，从库上执行同步命令的时间就会被延后。

另一方面，即使从库及时收到了主库的命令，但是，也可能会因为正在处理其它复杂度高的命令（例如集合操作命令）而阻塞。此时，从库需要处理完当前的命令，才能执行主库发送的命令操作，这就会造成主从数据不一致。而在主库命令被滞后处理的这段时间内，主库本身可能又执行了新的写操作。这样一来，主从库间的数据不一致程度就会进一步加剧。

那么，我们该怎么应对呢？我给你提供两种方法。

首先，**在硬件环境配置方面，我们要尽量保证主从库间的网络连接状况良好**。例如，我们要避免把主从库部署在不同的机房，或者是避免把网络通信密集的应用（例如数据分析应用）和Redis主从库部署在一起。

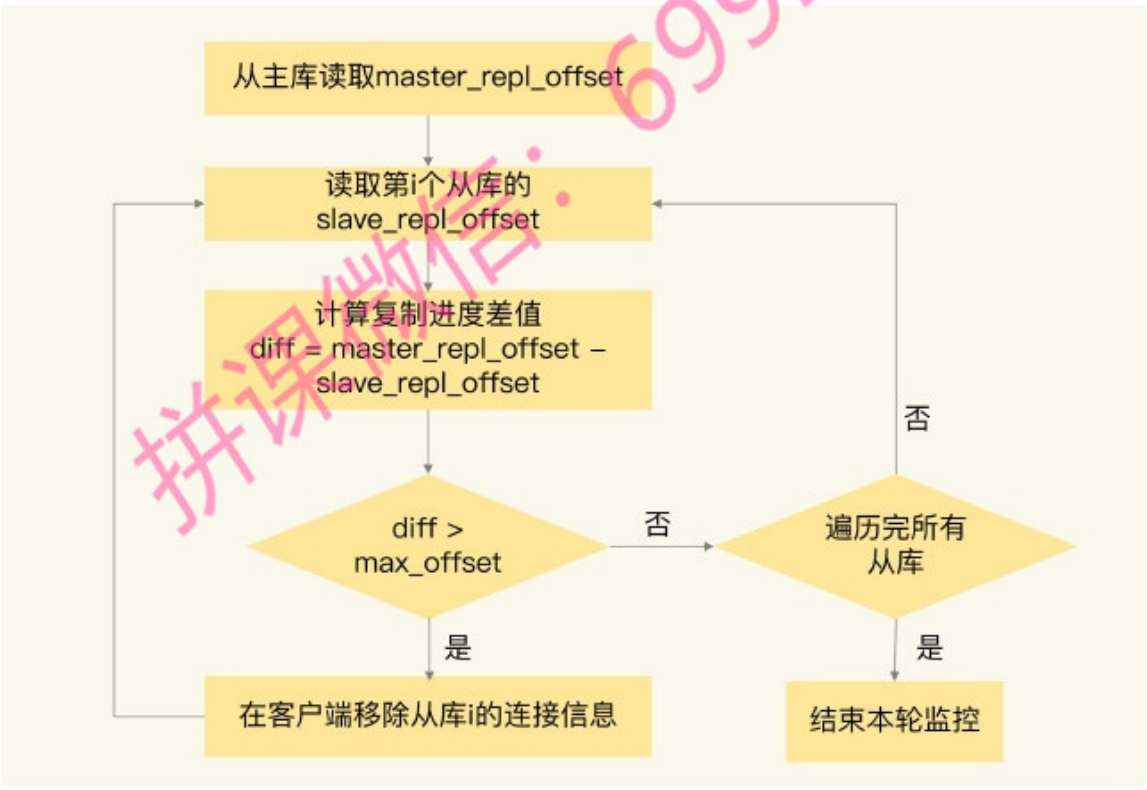
另外，**我们还可以开发一个外部程序来监控主从库间的复制进度**。

因为Redis的INFO replication命令可以查看主库接收写命令的进度信息（master_repl_offset）和从库复制

写命令的进度信息（slave_repl_offset），所以，我们就可以开发一个监控程序，先用INFO replication命令查到主、从库的进度，然后，我们用master_repl_offset减去slave_repl_offset，这样就能得到从库和主库间的复制进度差值了。

如果某个从库的进度差值大于我们预设的阈值，我们可以让客户端不再和这个从库连接进行数据读取，这样就可以减少读到不一致数据的情况。不过，为了避免出现客户端和所有从库都不能连接的情况，我们需要把复制进度差值的阈值设置得大一些。

我们在应用Redis时，可以周期性地运行这个流程来监测主从库间的不一致情况。为了帮助你更好地理解这个方法，我画了一张流程图，你可以看下。



当然，监控程序可以一直监控着从库的复制进度，当从库的复制进度又赶上主库时，我们就允许客户端再次跟这些从库连接。

除了主从数据不一致以外，我们有时还会在从库中读到过期的数据，这是怎么回事呢？接下来，我们就来详细分析一下。

读取过期数据

我们在使用Redis主从集群时，有时会读到过期数据。例如，数据X的过期时间是202010240900，但是客户端在202010240910时，仍然可以从从库中读到数据X。一个数据过期后，应该是被删除的，客户端不能再读取到该数据，但是，Redis为什么还能在从库中读到过期的数据呢？

其实，这是由Redis的过期数据删除策略引起的。我来给你具体解释下。

Redis同时使用了两种策略来删除过期的数据，分别是惰性删除策略和定期删除策略。

先说惰性删除策略。当一个数据的过期时间到了以后，并不会立即删除数据，而是等到再有请求来读写这个数据时，对数据进行检查，如果发现数据已经过期了，再删除这个数据。

这个策略的好处是尽量减少删除操作对CPU资源的使用，对于用不到的数据，就不再浪费时间进行检查和删除了。但是，这个策略会导致大量已经过期的数据留存在内存中，占用较多的内存资源。所以，Redis在使用这个策略的同时，还使用了第二种策略：定期删除策略。

定期删除策略是指，Redis每隔一段时间（默认100ms），就会随机选出一定数量的数据，检查它们是否过期，并把其中过期的数据删除，这样就可以及时释放一些内存。

清楚了这两个删除策略，我们再来看看它们为什么会导致读取到过期数据。

首先，虽然定期删除策略可以释放一些内存，但是，Redis为了避免过多删除操作对性能产生影响，每次随机检查数据的数量并不多。如果过期数据很多，并且一直没有再被访问的话，这些数据就会留存在Redis实例中。业务应用之所以会读到过期数据，这些留存数据就是一个重要因素。

其次，惰性删除策略实现后，数据只有被再次访问时，才会被实际删除。如果客户端从主库上读取留存的过期数据，主库会触发删除操作，此时，客户端并不会读到过期数据。但是，从库本身不会执行删除操作，如果客户端在从库中访问留存的过期数据，从库并不会触发数据删除。那么，从库会给客户端返回过期数据吗？

这就和你使用的Redis版本有关了。如果你使用的是Redis 3.2之前的版本，那么，从库在服务读请求时，并不会判断数据是否过期，而是会返回过期数据。在3.2版本后，Redis做了改进，如果读取的数据已经过期了，从库虽然不会删除，但是会返回空值，这就避免了客户端读到过期数据。所以，**在应用主从集群时，尽量使用Redis 3.2及以上版本**。

你可能会问，只要使用了Redis 3.2后的版本，就不会读到过期数据了吗？其实还是会的。

为啥会这样呢？这跟Redis用于设置过期时间的命令有关系，有些命令给数据设置的过期时间在从库上可能会被延后，导致应该过期的数据又在从库上被读取到了，我来给你具体解释下。

我先给你介绍下这些命令。设置数据过期时间的命令一共有4个，我们可以把它们分成两类：

- EXPIRE和PEXPIRE：它们给数据设置的是**从命令执行时开始计算的存活时间**；
- EXPIREAT和PEXPIREAT：**它们会直接把数据的过期时间设置为具体的一个时间点**。

这4个命令的参数和含义如下表所示：

	过期时间设置命令	参数	含义
第一类	EXPIRE	<key> <ttl>	将key 的存活时间设置为 ttl 秒
	PEXPIRE	<key> <ttl>	将key 的存活时间设置为 ttl 毫秒
第二类	EXPIREAT	<key> <timestamp>	将key 的过期时间设置为timestamp指定的秒数时间点
	PEXPIREAT	<key> <timestamp>	将key 的过期时间设置为timestamp指定的毫秒数时间点

为了方便你理解，我给你举两个例子。

第一个例子是使用EXPIRE命令，当执行下面的命令时，我们就把testkey的过期时间设置为60s后。

```
EXPIRE testkey 60
```

第二个例子是使用EXPIREAT命令，例如，我们执行下面的命令，就可以让testkey在2020年10月24日上午9点过期，命令中的1603501200就是以秒数时间戳表示的10月24日上午9点。

```
EXPIREAT testkey 1603501200
```

好了，知道了这些命令，下面我们来看看这些命令如何导致读到过期数据。

当主从库全量同步时，如果主库接收到了一条EXPIRE命令，那么，主库会直接执行这条命令。这条命令会在全量同步完成后，发给从库执行。而从库在执行时，就会在当前时间的基础上加上数据的存活时间，这样一来，从库上数据的过期时间就会比主库上延后了。

这么说可能不太好理解，我再给你举个例子。

假设当前时间是2020年10月24日上午9点，主从库正在同步，主库收到了一条命令：EXPIRE testkey 60，这就表示，testkey的过期时间就是24日上午9点1分，主库直接执行了这条命令。

但是，主从库全量同步花费了2分钟才完成。等从库开始执行这条命令时，时间已经是9点2分了。而EXPIRE命令是把testkey的过期时间设置为当前时间的60s后，也就是9点3分。如果客户端在9点2分30秒时在从库上读取testkey，仍然可以读到testkey的值。但是，testkey实际上已经过期了。

为了避免这种情况，我给你的建议是，**在业务应用中使用EXPIREAT/PEXPIREAT命令，把数据的过期时间设置为具体的时间点，避免读到过期数据。**

好了，我们先简单地总结下刚刚学过的这两个典型的坑。

致），数据不一致是难以避免的。我给你提供了应对方法：保证良好网络环境，以及使用程序监控从库复制进度，一旦从库复制进度超过阈值，不让客户端连接从库。

- 对于读到过期数据，这是可以提前规避的，一个方法是，使用Redis 3.2及以上版本；另外，你也可以使用EXPIREAT/PEXPIREAT命令设置过期时间，避免从库上的数据过期时间滞后。不过，这里有个地方需要注意下，**因为EXPIREAT/PEXPIREAT设置的是时间点，所以，主从节点上的时钟要保持一致，具体的做法是，让主从节点和相同的NTP服务器（时间服务器）进行时钟同步**。

除了同步过程中有坑以外，主从故障切换时，也会因为配置不合理而踩坑。接下来，我向你介绍两个服务挂掉的情况，都是由不合理配置项引起的。

不合理配置项导致的服务挂掉

这里涉及到的配置项有两个，分别是**protected-mode和cluster-node-timeout。**

1.protected-mode 配置项

这个配置项的作用是限定哨兵实例能否被其他服务器访问。当这个配置项设置为yes时，哨兵实例只能在部署的服务器本地进行访问。当设置为no时，其他服务器也可以访问这个哨兵实例。

正因为这样，如果protected-mode被设置为yes，而其余哨兵实例部署在其它服务器，那么，这些哨兵实例间就无法通信。当主库故障时，哨兵无法判断主库下线，也无法进行主从切换，最终Redis服务不可用。

所以，我们在应用主从集群时，要注意将protected-mode 配置项设置为no，并且将bind配置项设置为其它哨兵实例的IP地址。这样一来，只有在bind中设置了IP地址的哨兵，才可以访问当前实例，既保证了实例间能够通信进行主从切换，也保证了哨兵的安全性。

我们来看一个简单的小例子。如果设置了下面的配置项，那么，部署在192.168.10.3/4/5这三台服务器上的哨兵实例就可以相互通信，执行主从切换。

```
protected-mode no
bind 192.168.10.3 192.168.10.4 192.168.10.5
```

2.cluster-node-timeout配置项

这个配置项设置了Redis Cluster中实例响应心跳消息的超时时间。

当我们在Redis Cluster集群中为每个实例配置了“一主一从”模式时，如果主实例发生故障，从实例会切换为主实例，受网络延迟和切换操作执行的影响，切换时间可能较长，就会导致实例的心跳超时（超出cluster-node-timeout）。实例超时后，就会被Redis Cluster判断为异常。而Redis Cluster正常运行的条件就是，有半数以上的实例都能正常运行。

所以，如果执行主从切换的实例超过半数，而主从切换时间又过长的话，就可能有半数以上的实例心跳超时，从而可能导致整个集群挂掉。所以，**我建议你将cluster-node-timeout调大些（例如10到20秒）**。

小结

这节课，我们学习了Redis主从库同步时可能出现的3个坑，分别是主从数据不一致、读取到过期数据和不合理配置项导致服务挂掉。

为了方便你掌握，我把这些坑的成因和解决方法汇总在下面的这张表中，你可以再回顾下。

坑	原因	解决方法
主从数据不一致	主从数据异步复制	使用外部监控程序对比主从库复制进度，不让客户端从落后的从库中读取数据
读到过期数据	过期数据删除策略	1. 使用Redis3.2及以上版本 2. 使用EXPIREAT/PEXPIREAT命令给数据设置过期时间点
不合理配置项导致服务挂掉	protected-mode、cluster-node-timeout配置不合理	1. 设置protected-mode为no 2. 调大cluster-node-timeout

最后，关于主从库数据不一致的问题，我还想再给你提一个小建议：Redis中的slave-serve-stale-data配置项设置了从库能否处理数据读写命令，你可以把它设置为no。这样一来，从库只能服务INFO、SLAVEOF命令，这就可以避免在从库中读到不一致的数据了。

不过，你要注意下这个配置项和slave-read-only的区别，slave-read-only是设置从库能否处理写命令，slave-read-only设置为yes时，从库只能处理读请求，无法处理写请求，你可不要搞混了。

每课一问

按照惯例，我给你提个小问题，我们把slave-read-only设置为no，让从库也能直接删除数据，以此来避免读到过期数据，你觉得，这是一个好方法吗？

欢迎在留言区写下你的思考和答案，我们一起交流讨论。如果你觉得今天的内容对你有所帮助，也欢迎你分享给你的朋友或同事。我们下节课见。

精选留言：

- Kaito 2020-11-02 00:13:14
  把 slave-read-only 设置为 no，让从库也能直接删除数据，以此来避免读到过期数据，这种方案是否可行？

  我个人觉得这个问题有些歧义，因为尽管把 slave-read-only 设置为 no，其实 slave 也不会主动过期删除从 master 同步过来的数据的。

  我猜老师想问的应该是：假设让 slave 也可以自动删除过期数据，是否可以保证主从库的一致性？

  其实这样也无法保证，例如以下场景：

  1、主从同步存在网络延迟。例如 master 先执行 SET key 1 10，这个 key 同步到了 slave，此时 key 在主

从库都是 10s 后过期，之后这个 key 还剩 1s 过期时，master 又执行了 expire key 60，重设这个 key 的过期时间。但 expire 命令向 slave 同步时，发生了网络延迟并且超过了 1s，如果 slave 可以自动删除过期 key，那么这个 key 正好达到过期时间，就会被 slave 删除了，之后 slave 再收到 expire 命令时，执行会失败。最后的结果是这个 key 在 slave 上丢失了，主从库发生了不一致。

2、主从机器时钟不一致。同样 master 执行 SET key 1 10，然后把这个 key 同步到 slave，但是此时 slave 机器时钟如果发生跳跃，优先把这个 key 过期删除了，也会发生上面说的不一致问题。

所以 Redis 为了保证主从同步的一致性，不会让 slave 自动删除过期 key，而只在 master 删除过期 key，之后 master 会向 slave 发送一个 DEL，slave 再把这个 key 删除掉，这种方式可以解决主从网络延迟和机器时钟不一致带来的影响。

再解释一下 slave-read-only 的作用，它主要用来控制 slave 是否可写，但是否主动删除过期 key，根据 Redis 版本不同，执行逻辑也不同。

1、如果版本低于 Redis 4.0，slave-read-only 设置为 no，此时 slave 允许写入数据，但如果 key 设置了过期时间，那么这个 key 过期后，虽然在 slave 上查询不到了，但并不会在内存中删除，这些过期 key 会一直占着 Redis 内存无法释放。

2、Redis 4.0 版本解决了上述问题，在 slave 写入带过期时间的 key，slave 会记下这些 key，并且在后台定时检测这些 key 是否已过期，过期后从内存中删除。

但是请注意，这 2 种情况，slave 都不会主动删除由 *master 同步过来带有过期时间的 key*。也就是 master 带有过期时间的 key，什么时候删除由 master 自己维护，slave 不会介入。如果 slave 设置了 slave-read-only = no，而且是 4.0+ 版本，slave 也只维护直接向自己写入 的带有过期的 key，过期时只删除这些 key。

另外，我还能想到的主从同步的 2 个问题:

1、主从库设置的 maxmemory 不同，如果 slave 比 master 小，那么 slave 内存就会优先达到 maxmemroy，然后开始淘汰数据，此时主从库也会产生不一致。

2、如果主从同步的 client-output-buffer-limit 设置过小，并且 master 数据量很大，主从全量同步时可能会导致 buffer 溢出，溢出后主从全量同步就会失败。如果主从集群配置了哨兵，那么哨兵会让 slave 继续向 master 发起全量同步请求，然后 buffer 又溢出同步失败，如此反复，会形成复制风暴，这会浪费 master 大量的 CPU、内存、带宽资源，也会让 master 产生阻塞的风险。 [14赞]

- 杨逸林 2020-11-02 07:45:05
  不是个好方法，如果不同客户端，去非当前从库读取数据时，就会出现缓存不一致的情况。 [4赞]

- Geek_9a0c9f 2020-11-02 22:11:24
  我现在有个问题，redis的读写是分离的么，之前讲的不是主库读写都行，从库只能读么？？？？？

- 零点 2020-11-02 14:17:33
  只有主库提供服务，从库只保证高可用

- yeek 2020-11-02 13:40:44
  主从双写会带来不少问题，我们一般在进行redis垂直拆分的时候，线上不停服更新，会短暂打开从库写功能

- test 2020-11-02 13:02:37
  slave-serve-stale-data是在主从复制中从服务器是否可以响应客户端的请求，slave-read-only 是设置从库能否处理写命令。

  把 slave-read-only 设置为 no，让从库也能直接删除数据，会造成主从不一致，不推荐使用。