15 | ZAB协议：如何实现操作的顺序性?

2020-03-16 韩健

分布式协议与算法实战 进入课程 >

讲述：于航
时长 12:31 大小 11.47M

你好，我是韩健。

很多同学应该使用过 ZooKeeper，它是一个开源的分布式协调服务，比如你可以使用它进行配置管理、名字服务等等。在 ZooKeeper 中，数据是以节点的形式存储的。如果你要用 ZooKeeper 做配置管理，那么就需要在里面创建指定配置，假设创建节点"/geekbang"和"/geekbang/time"，步骤如下：

复制

```
[zk: localhost:2181(CONNECTED) 7] create /geekbang 123
Created /geekbang
[zk: localhost:2181(CONNECTED) 8] create /geekbang/time 456
Created /geekbang/time
```

我们分别创建了配置"/geekbang" 和"/geekbang/time"，对应的值分别为 123 和 456。那么在这里我提个问题：你觉得在 ZooKeeper 中，能用兰伯特的 Multi-Paxos 实现各节点数据的共识和一致吗？

当然不行。因为兰伯特的 Multi-Paxos，虽然能保证达成共识后的值不再改变，但它不管关心达成共识的值是什么，也无法保证各值（也就是操作）的顺序性。这是为什么呢？这个问题是 ZAB 协议着力解决的，也是理解 ZAB 协议的关键。

不过，虽然大家都在提 ZAB 协议，但是在我看来，ZAB 协议和 ZooKeeper 代码耦合在一起，也就是说，你是无法单独使用 ZAB 协议的，所以一般而言，只需要理解 ZAB 协议的架构和基础原理就可以了，不需要对代码和细节做太多的深究。所以，我会从 ZAB 协议的最核心设计目标（如何实现操作的顺序性）出发，带你了解它的基础原理。

为什么 Multi-Paxos 无法实现操作顺序性？

兰伯特的 Multi-Paxos 解决的是一系列值如何达成共识的问题，它关心的是，对于指定序号的位置，最多只有一个指令（Command）会被选定，但它不关心选定的是哪个指令，也就是说，它不关心指令的顺序性（也就是操作的顺序性）。

这么说可能比较抽象，为了方便你理解，我举个具体的例子演示一下（一个 3 节点的 Multi-Paxos 集群），为了演示方便，我们假设当前所有节点被选定的指令的最大序号为 100，也就是说，新提议的指令对应的序号将为 101。

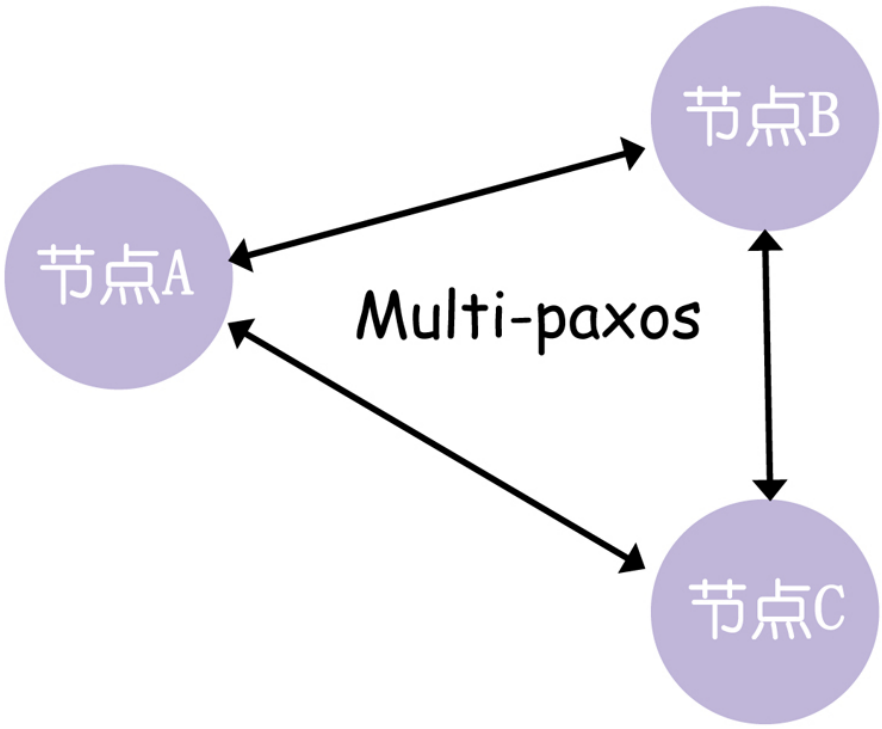

图1

首先节点 A 是领导者，提议了指令 X、Y，但是因为网络故障，指令只成功复制到了节点 A。

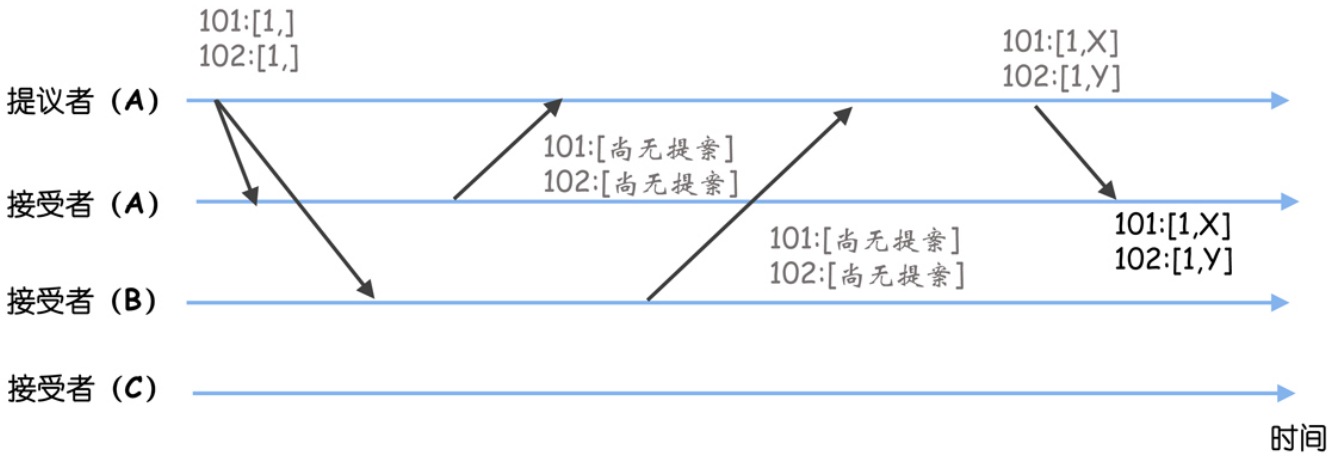

图2

假设这时节点 A 故障了，新当选的领导者为节点 B。节点 B 当选领导者后，需要先作为学习者了解目前已被选定的指令。节点 B 学习之后，发现当前被选定指令的最大序号为 100（因为节点 A 故障了，它被选定指令的最大序号 102，无法被节点 B 发现），那么它可以从序号 101 开始提议新的指令。这时它接收到客户端请求，并提议了指令 Z，指令 Z 被成功复制到节点 B、C。

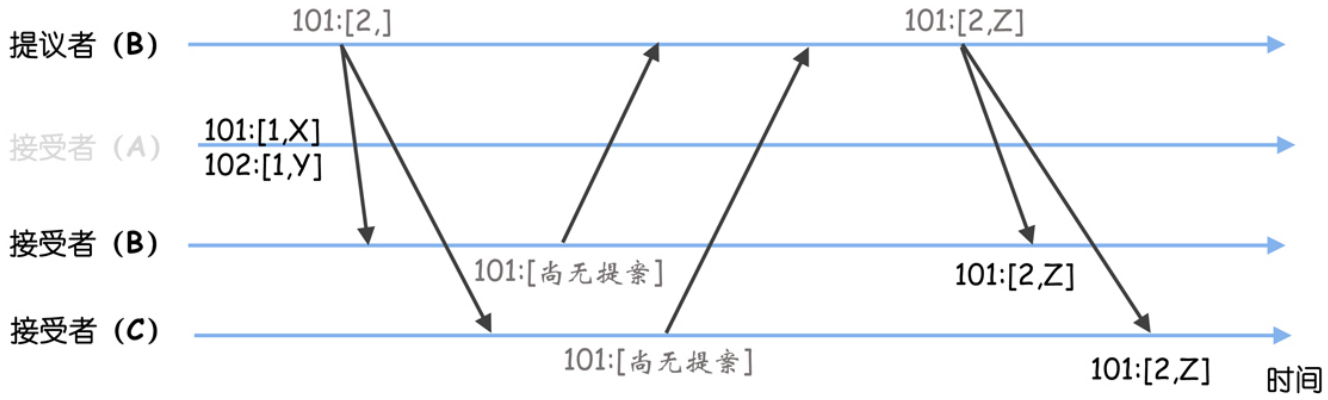

图3

这时节点 B 故障了，节点 A 恢复了，选举出领导者 C 后，节点 B 故障也恢复了。节点 C 当选领导者后，需要先作为学习者，了解目前已被选定的指令，这时它执行 Basic Paxos 的准备阶段，就会发现之前选定的值（比如 Z、Y），然后发送接受请求，最终在序号 101、102 处达成共识的指令是 Z、Y。就像下图的样子。

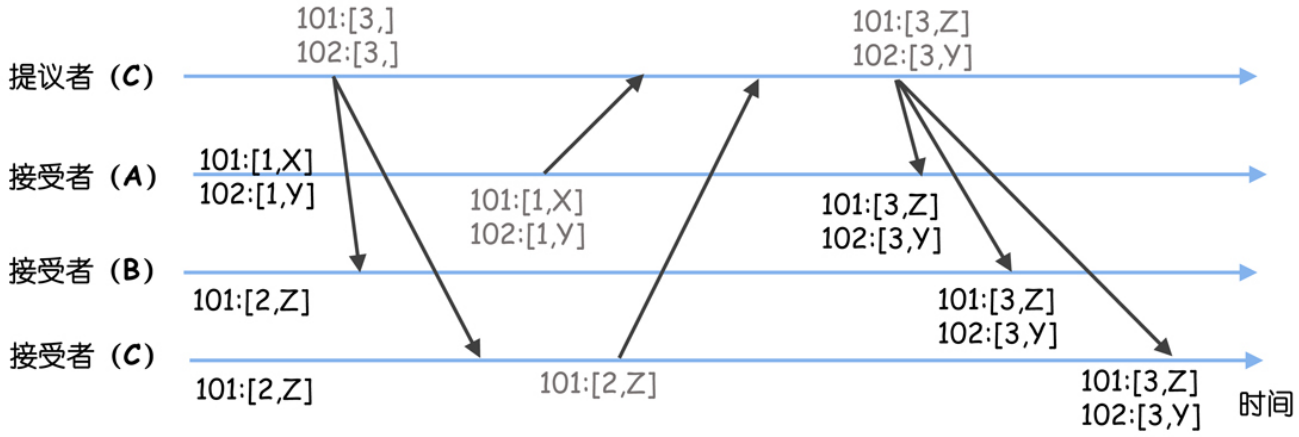

图4

在这里，你可以看到，原本预期的指令是 X、Y，最后变成了 Z、Y，也就是说，虽然 Multi-Paxos 能就一系列值达成共识，但它不关心达成共识后的值是什么，这显然不是我们想要的结果。

比如，假设在 ZooKeeper 中直接使用了兰伯特的 Multi-Paxos，这时咱们创建节点"/geekbang"和"/geekbang/time"，那么就可能出现，系统先创建了节点"/geekbang/time"，这样肯定就出错了：

复制代码

```
1 [zk: localhost:2181(CONNECTED) 6] create /geekbang/time 456
2 Node does not exist: /geekbang/time
```

因为创建节点"/geekbang/time"时，找不到节点"/geekbang"，所以就会创建失败。

在这里我多说几句，兰伯特有很多关于分布式的理论，这些理论都很经典（比如拜占庭将军问题、Paxos），但也因为太早了，与实际场景结合的不多，所以后续的众多算法是在这个基础之上做了大量的改进（比如，PBFT、Raft 等）。关于这一点，我在13 讲也强调过，你需要注意。

另外我还想补充一下，在我看来，在ZAB 论文中，关于 Paxos 问题（Figure 1 ）的分析是有争议的。因为 ZooKeeper 当时应该考虑的是 Multi-Paxos，而不是有多个提议者的 Basic Paxos。对于 Multi-Paxos 而言，领导者作为唯一提议者，不存在同时多个提议者的情况。也就是说，Multi-Paxos 无法保证操作的顺序性的问题是存在的，但原因不是文中演示的原因，**本质上是因为 Multi-Paxos 实现的是一系列值的共识，不关心最终达成共识的值是什么，不关心各值的顺序。**

既然 Multi-Paxos 不行，ZooKeeper 怎么实现操作的顺序性的呢？答案是它实现了 ZAB 协议。

你可能会说了：Multi-Paxos 无法实现操作的顺序性，但 Raft 可以啊，为什么 ZooKeeper 不用 Raft 呢？这个问题其实比较简单，因为 Raft 出来的比较晚，直到 2013 年才正式提出，在 2007 年开发 ZooKeeper 的时候，还没有 Raft 呢。

ZAB 是如何保证操作的顺序性的？

与兰伯特的 Multi-Paxos 不同，ZAB 不是共识算法，不基于状态机，而是基于主备模式的原子广播协议，最终实现了操作的顺序性。

这里我说的主备，就是 Master-Slave 模型，一个主节点和多个备份节点，所有副本的数据都以主节点为准，主节点采用二阶段提交，向备份节点同步数据，如果主节点发生故障，数据最完备的节点将当选主节点。而原子广播协议，你可以理解成广播一组消息，消息的顺序是固定的。

需要你注意的是，ZAB 在这里做了个优化，为了实现分区容错能力，将数据复制到大多数节点后（也就是如果大多数节点准备好了），领导者就会进入提交执行阶段，通知备份节点

执行提交操作。**在这一点上，Raft 和 ZAB 是类似的，我建议你可以对比着 Raft 算法来理解 ZAB。**

讲到这儿我再多说一句，前面几讲的留言中有同学问状态机的事情：在 Multi-Paxos、Raft 中为什么需要状态机？这是一个很棒的问题，为你的深入思考点个赞！所以咱们先来看一下这个问题。

什么是状态机?

本质上来说，状态机指的是有限状态机，它是一个数学模型。你可以这么理解：状态机是一个功能模块，用来处理一系列请求，最大的特点就是确定性，也就是说，对于相同的输入，不管重复运行多少次，最终的内部状态和输出都是相同的。

就像你敲击键盘，在 Word 文档上打字一样，你敲击键盘的顺序决定了 Word 文档上的文字，你按照相同的顺序敲击键盘，一定能敲出相同的文字，这就是一个现实版的状态机。

那么为什么在 Multi-Paxos、Raft 中需要状态机呢?

你想一下，Multi-Paxos、Raft 都是共识算法，而共识算法是就一系列值达成共识的，达成共识后，这个值就不能改了。但有时候我们是需要更改数据的值的，比如 KV 存储，我们肯定需要更改指定 key（比如 X）对应的值，这时我们就可以通过状态机来解决这个问题。

比如，如果你想把 X 的值改为 7，那你可以提议一个新的指令“SET X = 7”，当这个指令被达成共识并提交到状态机后，你查询到的值就是 7 了，也就成功修改了 X 的值。

讲到这儿，你应该理解什么是状态机，为什共识算法需要状态机了吧？在解决这个问题之后，咱们说回刚刚的话题：ZAB 协议如何保证操作的顺序性?

如何实现操作的顺序性?

首先，ZAB 实现了主备模式，也就是所有的数据都以主节点为准：

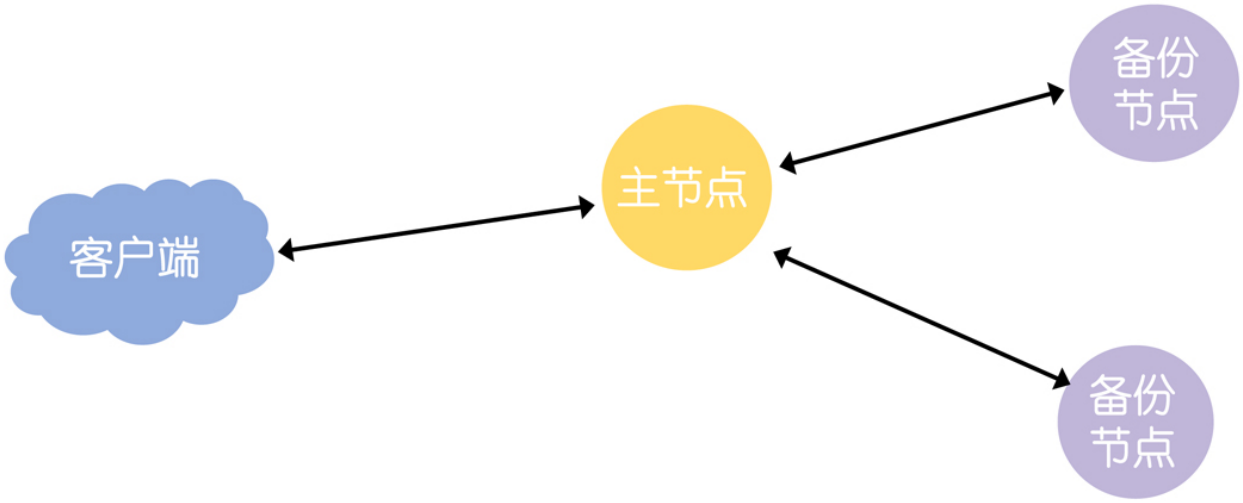

图5

其次，ZAB 实现了 FIFO 队列，保证消息处理的顺序性。

另外，ZAB 还实现了当主节点崩溃后，只有日志最完备的节点才能当选主节点，因为日志最完备的节点包含了所有已经提交的日志，所以这样就能保证提交的日志不会再改变。

你看，ZAB 协议通过这几个特性就能保证后来的操作不会比当前的操作先执行，也就能保证节点"/geekbang"会在节点"/geekbang/time"之前创建。

学到这里，想必你已经发现了，这些特性好像和 Raft 很像。是的，因为在前面几讲，我们已经学习了 Raft 算法，所以你可以类比 Raft 来理解，在 Raft 中：

所有日志以领导者的为准；

领导者接收到客户端请求后，会基于请求中的指令，创建日志项，并将日志项缓存在本地，然后按照顺序，复制到其他节点和提交；

在 Raft 中，也是日志最完备的节点才能当选领导者。

内容小结

本节课我主要带你了解了状态机、为什么 Multi-Paxos 无法实现操作的顺序性，以及 ZAB 协议如何保证操作的顺序性。我希望你明确这样几个重点。

1. 状态机最大的特点是确定性，对于相同的输入不管运行多少次，最终的内部状态和输出都是相同的。需要你注意的是，在共识算法中，我们可以通过提议新的指令，达成共识后，提交给状态机执行，来达到修改指定内容的效果，比如修改 KV 存储中指定 key 对应的值。
2. ZAB 是通过“一切以领导者为准”的强领导者模型和严格按照顺序提交日志，来实现操作的顺序性的，这一点和 Raft 是一样的。

最后我想说的是，兰伯特的 Multi-Paxos 只考虑了如何实现共识，也就是，如何就一系列值达成共识，未考虑如何实现各值（也就是操作）的顺序性。最终 ZooKeeper 实现了基于主备模式的原子广播协议，保证了操作的顺序性，而且，ZAB 协议的实现，影响到了后来的共识算法，也就是 Raft 算法，Raft 除了能就一些值达成共识，还能保证各值的顺序性。

学习完本讲内容后，你可以看到，Raft 算法和 ZAB 协议很类似，比如主备模式（也就是领导者、跟随者模型）、日志必须是连续的、以领导者的日志为准是日志一致等等。你可以想一下，那为什么它们会比较类似呢？

我的看法是，“英雄所见略同”。比如 ZAB 协议要实现操作的顺序性，而 Raft 的设计目标，不仅仅是操作的顺序性，而是线性一致性，这两个目标，都决定了它们不能允许日志不连续，要按照顺序提交日志，那么，它们就要通过上面的方法实现日志的顺序性，并保证达成共识（也就是提交）后的日志不会再改变。

课堂思考

我提到在 ZAB 中，写操作必须在主节点上执行，主节点是通过简化版的二阶段提交向备份节点同步数据。那么如果读操作访问的是备份节点，能保证每次都能读到最新的数据吗？为什么呢？欢迎在留言区分享你的看法，与我一同讨论。

最后，感谢你的阅读，如果这篇文章让你有所收获，也欢迎你将它分享给更多的朋友。

上一篇　14 | PoW算法：有办法黑比特币吗?

精选留言 (8)

写留言

Jialin
2020-03-16

关于 ZAB 协议的讲解有点差强人意，起初对这节课讲解充满了期待，现在，看来还得自己琢磨

> 作者回复: 期待哪方面的内容呢？基于实用性、篇幅等考虑，这次ZAB没作为重点。方便的话，可以聊聊你的想法，我后面补充。

1　5

猿人谷
2020-03-16

如果读操作访问的是备份节点，是不能保证每次都能实时的读到最新数据的。但Zookeeper 保证在一定的时间段内，客户端最终一定能够从服务端上读取到最新的数据状态。

2

夜空中最亮的星（华仔...

2020-03-17

不能保证

展开

qinsi

2020-03-16

状态机是一种抽象表述，只要满足状态机特性的实体都可以叫状态机：处于相同状态的多个状态机，在执行了相同的输入序列后，到达的最终状态也是相同的。这个特性使得状态机之间可以很方便地同步状态。

刘楠

2020-03-16

应该可以吧，虽然 ZAB协议中Leader等待follower的ACK反馈是指”只要半数以上的follower成功反馈即可，不需要收到全部follower反馈”，但zookeeper采用ZAB协议的核心就是只要有一台服务器提交了提案，就要确保所有的服务器最终都能正确提交proposal。这也是CAP/BASE最终实现一致性的一个体现，不知道理解的对不对。

展开

小晏子

2020-03-16

如果读操作访问的是备份节点，不能保证每次都能读到最新的数据。因为zk中，只要有大多数节点写入成功，就认为本次写入成功，这样就可能访问到过期数据。比如一个zk集群有10000台节点，当进行写入的时候，如果已经有6K个节点写入成功，zk就认为本次写请求成功。但是这时候如果一个客户端读取的刚好是另外4K个节点的数据，那么读取到的就是旧的过期数据。

展开

Geek_3894f9

2020-03-16

课后思考题，答案是不能。因为只要多数备份机成功，则写操作就成功了，而此时，如果去访问还没有写成功的备份机，则读不到主机上刚写成功的数。

展开

每天晒白牙

2020-03-16

思考题

我认为不能保证每次读取到的都是最新的数据，因为 ZAB 协议才用了主备的模式，备用节点只要大多数复制成功就算成功。如果此时客户端刚好读取到了没有完成复制的备用节点，读取到的数据就不是最新的

展开