专题20：Paxos 协议（史上最全、定期更新）

本文版本说明：V2

> 此文的格式，由markdown 通过程序转成而来，由于很多表格，没有来的及调整，出现一个格式问题，尼恩在此给大家道歉啦。
>
> 由于社群很多小伙伴，在面试，不断的交流最新的面试难题，所以，《Java面试红宝书》， 后面会不断升级，迭代。
>
> 本专题，作为 《Java面试红宝书》专题之一， 《Java面试红宝书》一共**30个面试专题，后续还会增加**

《Java面试红宝书》升级的规划为：

后续基本上，**每一个月，都会发布一次**，最新版本，可以扫描扫架构师尼恩微信， 发送 “领取电子书” 获取。

尼恩的微信二维码在哪里呢？

具体可以百度搜索 **疯狂创客圈 总目录**

面试问题交流说明：

如果遇到面试难题，或者职业发展问题， 或者中年危机问题，都可以来 疯狂创客圈社群交流，

加入交流群，加尼恩微信即可，

入交流群，加尼恩微信即可， 发送**“入群”**

Paxos有多重要呢?

Paxos协议/算法是分布式系统中比较重要的协议，它有多重要呢?

大牛说：

> Google Chubby的作者Mike Burrows说过这个世界上只有一种一致性算法，那就是Paxos，其它的算法都是残次品。

实际上：

> 理解了这两个分布式协议之后(Paxos/2PC)，学习其他分布式协议会变得相当容易。

Paxos算法及变种算法在分布式系统中应用广泛。
基于Paxos算法的变种有：ZAB、Raft。

Zookeeper 中的ZAB协议也是Paxos算法的变种。Zookeeper通过ZAB协议实现数据一致性，以提供数据一致性。

在分布式系统中，节点之间主要使用消息投递方式来完成。但通过消息投递的方式会遇到很多意外的情况，例如网络问题、进程挂掉、机器挂掉、进程很慢没有响应、进程重启等情况，这就会造成消息重复、一段时间内部不可达等现象。而 Paxos 算法就是基于消息传递且具有高度容错特性的一致性算法。换句话说，Paxos算法的作用就是在可能发生这些异常情况的分布式系统中，快速且正确地在集群内部对某个数据的值达成一致。

拜占庭将军问题

在各类介绍 Paxos 算法的文章中，都会提到著名的“拜占庭将军问题”，以及偶尔也会提到的“两军问题”。关于这两个问题的详细介绍可以阅读这篇下面这篇文章，基本讲清楚了。

简单的来说，拜占庭将军问题描述了这样一个场景：

> 拜占庭帝国有许多支军队，不同军队的将军之间必须制订一个统一的行动计划，从而做出进攻或者撤退的决定，同时，各个将军在地理上都是被分隔开来的，只能依靠军队的通讯员来进行通讯。然而，在所有的通讯员中可能会存在叛徒，这些叛徒可以任意篡改消息，从而达到欺骗将军的目的。

这就是著名的“拜占廷将军问题”。从理论上来说，在分布式计算领域，试图在异步系统和不可靠的通道上来达到一致性状态是不可能的。因此在对一致性的研究过程中，往往假设信道是可靠的。事实上，大多数系统都是部署在同一个局域网中的，因此消息被篡改的情况非常罕见；另一方面，由于硬件和网络原因而造成的消息不完整问题，只需一套简单的校验算法即可避免——因此，在实际工程实践中，可以假设不存在拜占庭问题，即假设所有消息都是完整的，没有被篡改的。

拜占庭将军问题与Paxos 的关系

拜占庭将军问题是由 Paxos 算法作者莱斯利·兰伯特提出的点对点通信中的基本问题。该问题要说明的含义是，在不可靠信道上试图通过消息传递的方式达到一致性是不可能的。

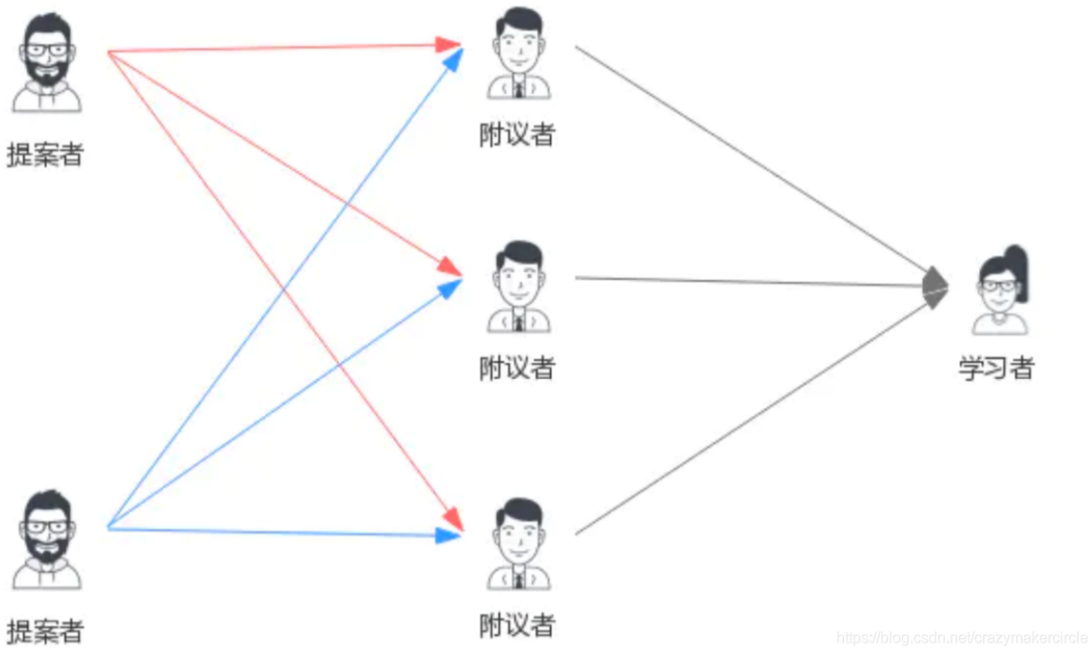
所以，Paxos 算法的前提是不存在拜占庭将军问题，即信道是安全的、可靠的，集群节点间传递的消息是不会被篡改的。

Paxos 算法

Paxos 算法是分布式技术大师 Lamport 提出的。Lamport 为了讲述这个算法，假想了一个叫做 Paxos 的希腊城邦进行选举的情景。这个算法也是因此而得名。在他的假想中，这个城邦要采用民主提议和投票的方式选出一个最终的决议，但由于城的居民没有人原意把全部时间和精力放在这种事情上，所以他们只能不定时的来参加提议，不定时来了解提议、投票进展，不定时的表达自己的投票意见。 Paxos 算法的目标就是让他们按照少数服从多数的方式，最终达成一致意见。

主要角色

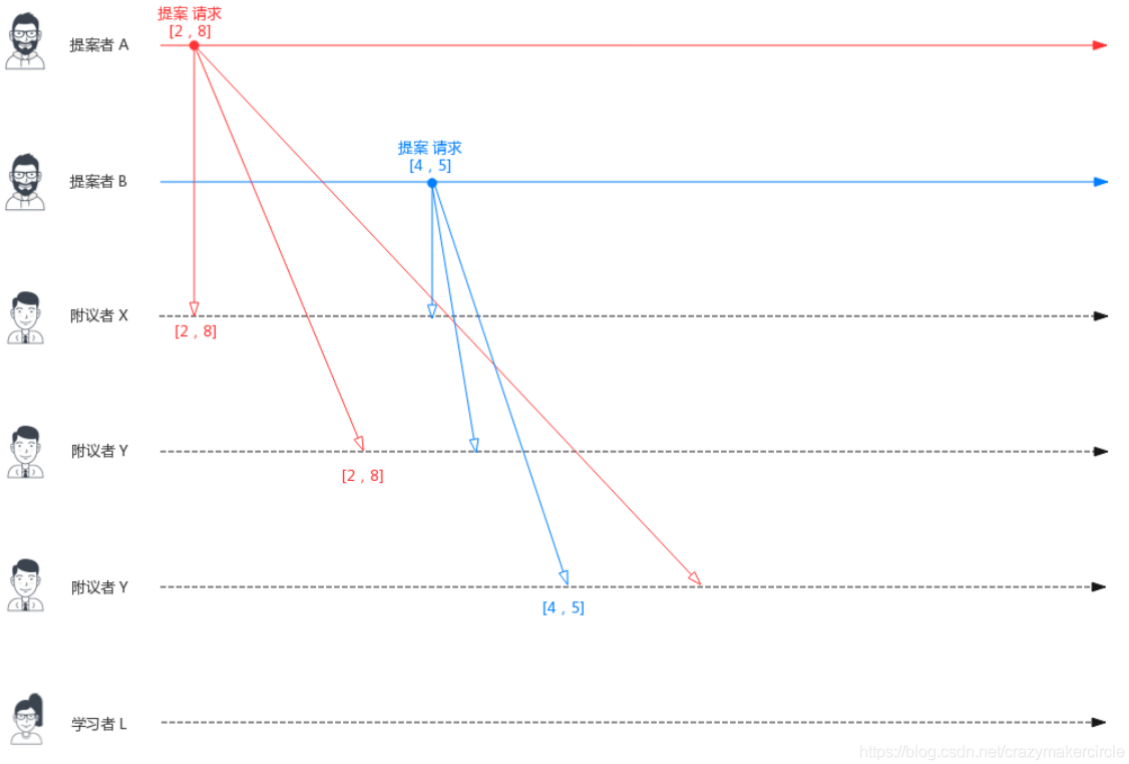
1. Proposer（提案者/提议者）：提议一个值，用于被投票决议。
2. Acceptor（附议者/接受者）：对每个提议进行投票。
3. Learner（学习者/告知者）：被告知投票的结果，不参与投票过程。



执行过程

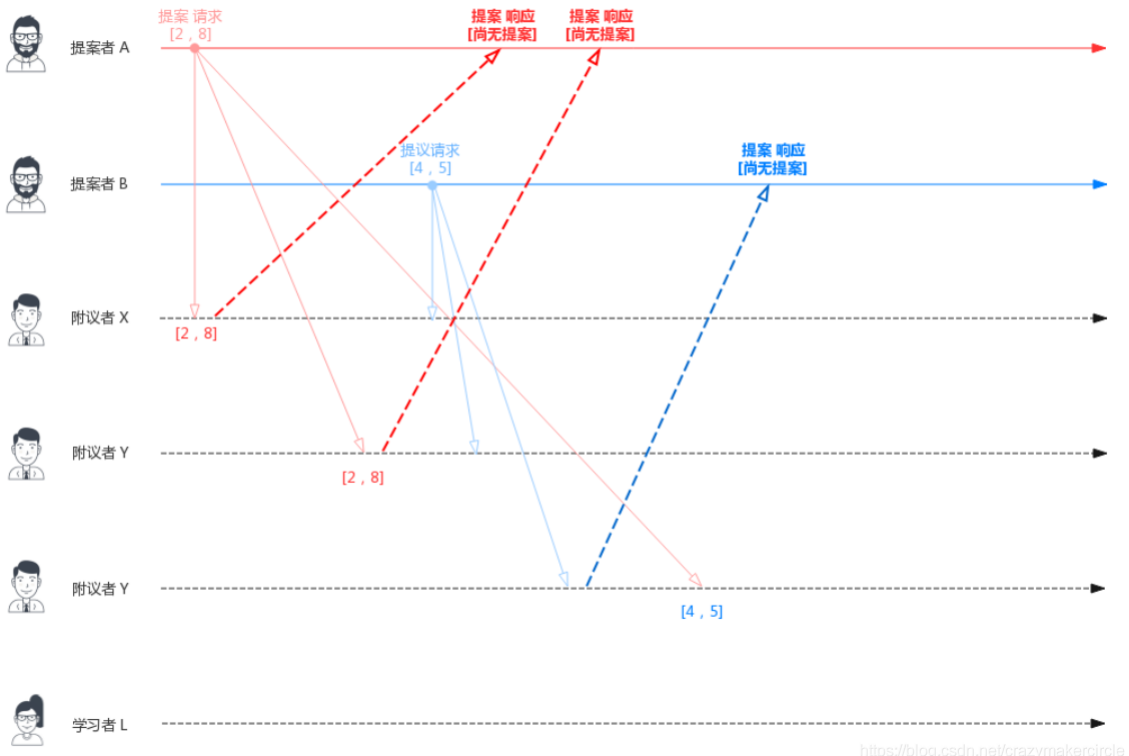
规定一个提议包含两个字段：[n, v]，其中 n 为序号（具有唯一性），v 为提议值。

下图演示了两个 Proposer（提案者） 和三个 Acceptor（附议者） 的系统中运行该算法的初始过程，每个 Proposer 都会向所有 Acceptor 发送提议请求。



当 Acceptor 接收到一个提议请求，包含的提议为 [n1, v1]，并且之前还未接收过提议请求，那么发送一个提议响应，设置当前接收到的提议为 [n1, v1]，并且保证以后不会再接受序号小于 n1 的提议。

如下图，Acceptor X 在收到 [n=2, v=8] 的提议请求时，由于之前没有接收过提议，因此就发送一个 [no previous] （尚无提案）的提议响应，并且设置当前接收到的提议为 [n=2, v=8]，并且保证以后不会再接受序号小于 2 的提议。其它的 Acceptor 类似。

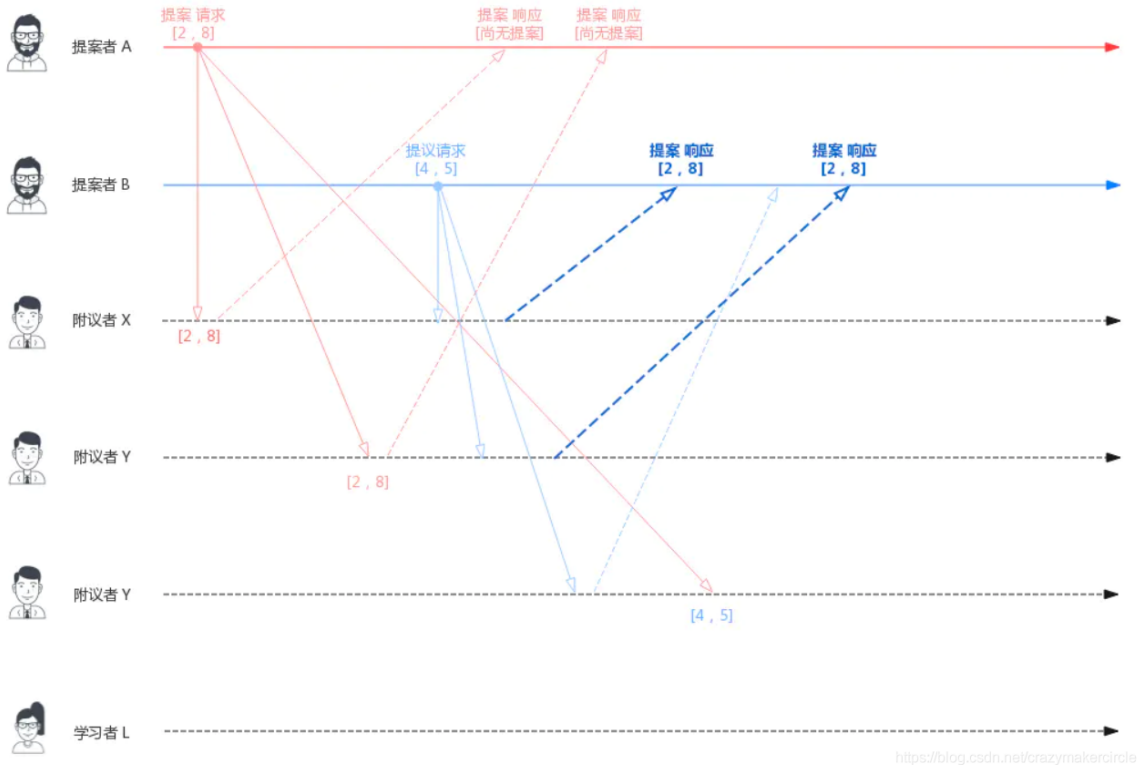


如果 Acceptor 之前已经接收过提议 [n1, v1]，现在接收到一个提议请求，提议为 [n2, v2]。

- 如果 n1 > n2，那么就丢弃该提议请求；
- 否则，发送提议响应，该提议响应包含之前已经接收过的提议 [n1, v1]，设置当前接收到的提议为 [n2, v2]，并且保证以后不会再接受序号小于 n2 的提议。

如下图:

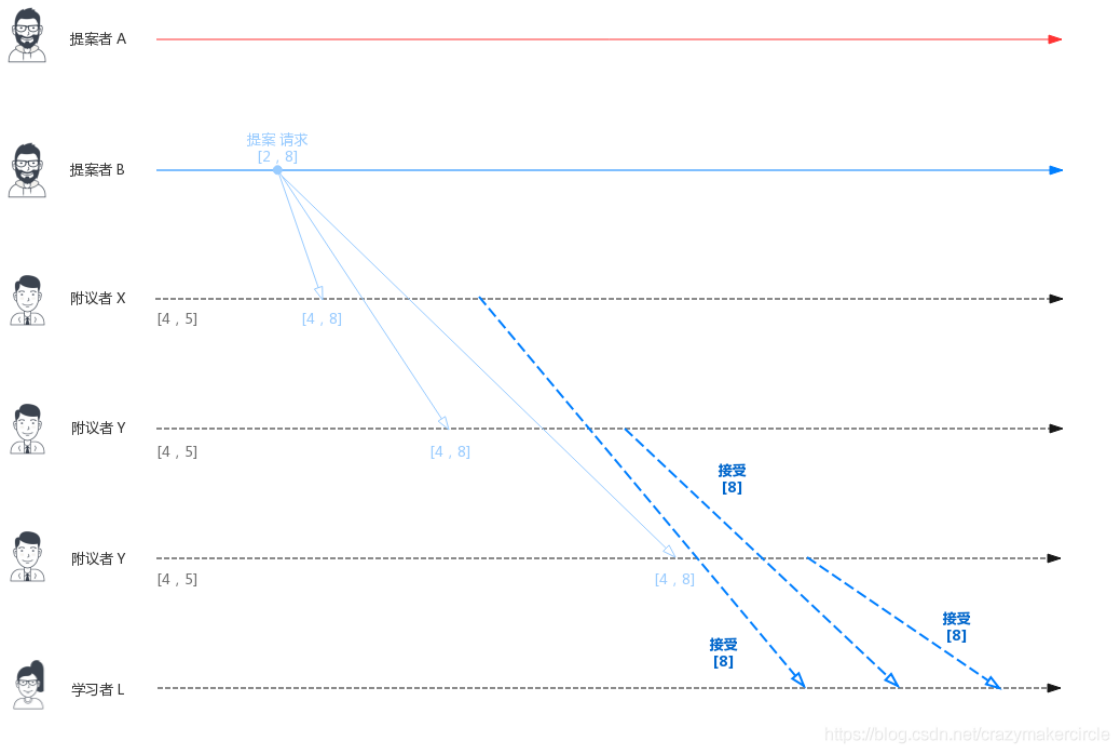
- Acceptor Z 收到 Proposer A 发来的 [n=2, v=8] 的提议请求，由于之前已经接收过 [n=4, v=5] 的提议，并且 2 < 4，因此就抛弃该提议请求；
- Acceptor X 收到 Proposer B 发来的 [n=4, v=5] 的提议请求，因为之前接收到的提议为 [n=2, v=8]，并且 2 <= 4，因此就发送 [n=2, v=8] 的提议响应，设置当前接收到的提议为 [n=4, v=5]，并且保证以后不会再接受序号小于 4 的提议。
- Acceptor Y 类似。



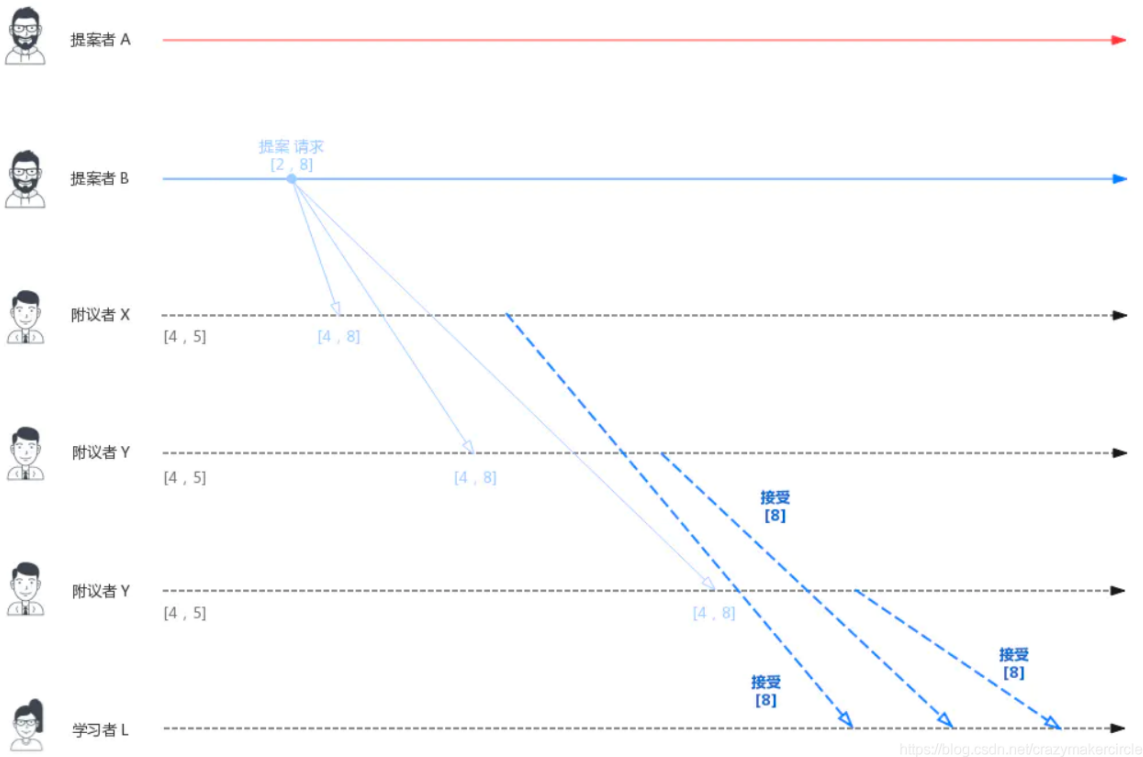
当一个 Proposer 接收到超过一半 Acceptor 的提议响应时，就可以发送接受请求。

Proposer A 接收到两个提议响应之后，就发送 [n=2, v=8] 接受请求。该接受请求会被所有 Acceptor 丢弃，因为此时所有 Acceptor 都保证不接受序号小于 4 的提议。

Proposer B 过后也收到了两个提议响应，因此也开始发送接受请求。需要注意的是，接受请求的 v 需要取它收到的最大 v 值，也就是 8。因此它发送 [n=4, v=8] 的接受请求。



Acceptor 接收到接受请求时，如果序号大于等于该 Acceptor 承诺的最小序号，那么就发送通知给所有的 Learner（学习者）。当 Learner 发现有大多数的 Acceptor 接收了某个提议，那么该提议的提议值就被 Paxos 选择出来。



Paxos 算法的一致性

Paxos 算法的一致性主要体现在以下几点：

- 每个提案者在提出提案时都会首先获取到一个具有全局唯一性的、递增的提案编号 N， 即在整个集群中是唯一的编号 N，然后将该编号赋予其要提出的提案。
- 每个表决者在 accept 某提案后，会将该提案的编号 N 记录在本地，这样每个表决者中保存的已经被 accept 的提案中会存在一个编号最大的提案，其编号假设为 maxN。每个表决者仅会 accept 编号大于自己本地 maxN 的提案。
- 在众多提案中最终只能有一个提案被选定。
- 一旦一个提案被选定，则其它服务器会主动同步(Learn)该提案到本地。
- 没有提案被提出则不会有提案被选定。

参考文献：

《从Paxos到ZooKeeper》

https://angus.nyc/2012/paxos-by-example/