# 为什么要写这个文档

在自己阅读和理解“Paxos made simple"一文的过程中，以及在给别人介绍的过程中，遇到了特别多的障碍。于是在想，到底在哪些方面造成了Paxos难以理解？感觉主要在以下几个方面：

1. 没有一个贯穿始终的例子，读者完全凭想象在理解这个本来就难懂的协议流程，没法落地和形成深刻印象。
2. 文章开始没有非常明确、有重点地介绍目标和问题，然后Majority, Chosen这些词介绍过程，首先让大家想到是个选举（就做一个决定），而没明白作者提出这个协议的出发点。到文章后半部分隐含地说明是用来一致性地决定状态序列时，很多人已经迷糊了。因此，明确要解决的问题的边界，非常重要。
3. 作者为了简化，在讲解单实例及推导过程时(从P1->P2->P2a->P2b->P2c->两阶段协议)，完全没有提到状态序列的概念，这几个点条件或者原则之间的关系也让人费解。这造成了后来突然抛出个多实例状态机时（a sequence of separate instances），跳跃性太大，无法体会作者的意图。

于是，我想换个方式来介绍怎么一步步引入的，以及一些问题背后的为什么。这不是要改变协议或者创新，仅仅是试图换个方式描述。

# 问题的引入

## 鸡蛋不能都放在一个篮子里

假设你开了个蛋糕店，每天生产和销售大量的蛋糕、布丁、冰激凌等。你有个处理订单的系统，能够处理每一笔订单。用户可以通过手机App或者网页查询。可以想象，如果要让客户信赖，这个订单系统应该做到：

1. 客户下的订单，无论是要几块蛋糕或者冰激凌，一旦成功，系统不能反悔(客户主动取消订单是另外一回事)。
2. 如果用户下订单成功，那么订单不能丢失。
3. 如果不能满足用户的需求，比如蛋糕不足，系统在客户下订单时就告诉用户订单失败。

假设你有台服务器，运行着订单处理系统，软件也没有bug，运行的很好，一切看起来那么正常。

但是有一天某个店小二跟你报告电脑硬盘坏了，即使头天晚上做了备份，今天的订单也全丢了。或者，由于天热发生风扇故障或其他小问题导致系统不能正常服务，用户订了的蛋糕也没法派送，虽然之前的订单没丢，业务也会有中断。

这里其实提到了两个可能出现的问题：丢订单(数据)，这个属于可靠性问题；如果订单数据没丢，只是一段时间用户不能下订单，这个属于可用性问题。

## 一山怎样才能容二虎？

你当然不希望一个风扇故障引起用户抱怨或者其他惨案；你更不希望丢失了订单数据，因为那些通过网络付了款的人用户可能会堵了你的门。

你有个朋友，建议你上个高可用系统，由两台服务器组成，例如M1和M2，只要其中一个能干活就可以，这样从可靠性和可用性两个方面，都有所大幅提高，毕竟两个机器硬盘同时故障的概率小很多。

看起来挺不错的方案，问题在哪呢？ 我们考虑M1和M2怎么干活。你的客户通过App或者浏览器下订单时，可能发给M1，也可能发给M2，这种情况下，如果现在只剩下一个草莓蛋糕，张三和李四都通过App下单，但是分别是M1和M2处理的，那么可能两个人都认为自己订蛋糕成功了，但是去取的时候，有一个人发现没有了。

## 杳无音讯的假死

你决定只有一个服务器能接收请求，另一个只是备用。比如M1是主服务器，M2从备用。客户默认找M1下订单，M1宕机才找M2。M1收到订单后，M1和M2做同样的处理，它们都处理完成才算成功。这下可靠性貌似提高了，假设M2宕机了，M1就独立处理订单了。可是如果M2好好的，M1宕机了呢？这个看起来也很简单: 直接让M2接订单并独立干活，不理M1不就完了吗？

问题在于：谁决定让M2独立干活，抛弃M1？M2自己？还是店老板您呢？

显然不能是店老板，你没法一直在那看着，也没那工夫。

M2如何判断M1宕机了呢？常见的方法是定期给M1发送心跳，并等待M1的回复。但是在网络不稳定的情况下，**可能出现客户张三能跟M1通讯，李四能跟M2通讯，但是M1和M2之间的通讯中断。在每个订单被处理时，都认为另外一个服务器挂了，自己应该独立处理**。这个就是脑裂问题，关于这个问题，有个用的比较多的解释如下：

看到这里，我们知道不是简单地把几个机器放在一起就变成了一个如你所想象的正常工作的集群。我们需要保证正确性、可靠性、可用性的机制。而目前已有的分布式一致性保障机制基本都是基于Paxos。

# Paxos到底能干什么

前面已经说明了单个服务器所存在的单点故障问题，也说明了多个服务器下，网络分割可能带来的脑裂问题。

Paxos的基本的想法是，多个服务器，组成一个Paxos小组或者委员会。做决定不是某一个委员说了算，而是过半数同意通过的才能形成决议。一旦形成决议，则不能修改，所有的小组成员都必须遵从这个决定。比如，如果决定第1个被处理的订单时张三的，第2个是李四的，第3个被处理订单是王五的。中间的某个或者某些订单可能因为蛋糕卖完了而失败，但是这不是Paxos要解决的问题，那是供需关系问题。

**为什么是一半以上(majority)？**

1) 一半以上是必须的，因为两个majority必定有交集。否则可能只剩下一个蛋糕，而两个客户都预订成功了。

2）为什么不要求全部或者90%以上同意？ 要提高可用性，总得允许小组的成员们请个事假、婚假什么的。

|  |  |  |
| --- | --- | --- |
| **关于Paxos的论断** | **能否做到** | **备注** |
| 能保证所有订单都成功吗？ | No | 用户能否买到蛋糕，依赖于你今天能做多少蛋糕。 |
| 能保证系统100%的可用性吗？ | No | 比如大规模停电就不能。但是只要过半数机器能工作就可以正常服务。 |
| 如果有大量客户端并发下订单，Paxos能保证和预知按照什么顺序处理订单吗？ | No | 消息可能延迟或者丢失，各个订单间的相对顺序不能预先确定。 |
| Paxos能让系统每秒处理更多的订单吗？ | No | Paxos小组内的各个成员执行相同序列的操作，并没有直接提高并发度。 |
| 不会产生有歧义的决议，一个序号最多只对应一个订单 | Yes |  |
| 保证各个服务器上处理订单的顺序是相同的，不论这个顺序具体是什么，不论各个订单到底是成功还是失败。 | Yes |  |
| 只要故障的服务器个数<k/2，即有majority个正常工作，系统就能正常提供服务，且保持状态一致 | Yes |  |

Paxos的一些假设：

1) 消息传输是异步的：消息可以有大的延迟或者丢失，但是不能被篡改

2）状态的持久化：成员可以短暂或者永久离开，只要有过半成员正常运行，协议即可运转；成员必须记住自己就某件事自己给谁投过票。

# Paxos的推导过程理解

个人认为paper中先讨论的单实例，没有跟状态变化关联，后来突然冒出个多实例，是造成**难以理解的因素**之一。因此，下面的例子中，我们还是讨论谁的订单被第1个处理，谁的第2个被处理...，这样比较容易切换。

上述问题可以归结为：各个服务器务必按照相同顺序处理订单，每个订单是否都成功不是一致性协议要解决的问题。在其中一个服务器故障后，如果还有其他服务器可以继续进行处理，那么切换过程不能造成混乱。即使发生切换，对客户来说，必须跟单个服务器在做处理是一样的结果，就是可用性更高了一点而已。具体可以理解为：

各个服务器都执行相同的订单序列，例如:

1号订单，是张三的2个草莓蛋糕

2号订单，是李四的3个慕斯蛋糕

3号订单，是王五的2个巧克力奶昔

...

注意，张三是不是真的下订单时间比李四早，在这里不重要，上面的三个人完全可能是同时下的订单，也可以是张三下订单比李四晚，但是张三的手机用的是更快的5G网络。我们不在乎谁先来的，只要各个服务器都认为张三的订单是1号订单，李四的是2号订单......就可以了。当然，排在后面的订单有可能买不着哦。

按照上面的理解，我们假设Paxos小组有5个委员(长老)组成，他们分别为：东邪(黄药师)、西毒(欧阳锋)、南帝(一灯法师)、北丐(洪七公)、中通(老顽童周伯通)。把这五个人按上北下南、左西右东来排列，也容易记住。

为什么是这五大长老? 主要从模型上比较匹配：

1. 五大长老说话算数，不会赖账，而且记忆力很好；
2. 五大长老并不住在一起，通过写信完成讨论订单，信件可以被延迟或者丢失，但是江湖人士都认识五大长老的字迹和签名，不会被伪造；
3. 他们有自己的活动安排，可能去云游了一段时间再回来，云游期间不收信件，更不会答复。
4. 江湖人士都知道这五大长老，在下订单时，可以找这五大长老中的任何一个说明自己的需求，然后由五大长老代表江湖人士下订单，长老们同时有具有投票权。(其实下订单的完全可以不是这五大长老，但是那样画图起来费劲，我们知道提交订单(提议者)与投票者这两个角色可以独立就可以了！)



图 1 五大长老作为提议者和接收者的paxos小组

我们把不同的长老用不同的颜色代替，提议的颜色跟提议人相同，而举手的颜色，跟提议人的颜色一致。这样我们可以通过数红色的手的个数，确定几个人同意了南帝的提议。

Paxos初始规则 P1： 谁最先来我先投谁！

## 早起就是好办事

假设今天早上蛋糕店系统刚上线，南帝就收到了华筝的请求，说要买几个蛋糕。优于华筝起得太早，没人冲突，且各大长老都在，提议很快得到了一致同意，华筝的就成了第一个被处理的订单！



图 1 全体一致通过

## 稍晚点出门就塞车

假设折天华筝、黄蓉和貂蝉（找个美女名字哈）起床差不多早，分别就近联系北丐、南帝和中通要下订单。三大长老分别提出了自己的主张，并且都得到了部分人的回应，如下图所示：



图 1 多提议并发，不能形成决议

在上述情况下，我们必须要有规则来处理，最起码的一条：如果冲突了，得允许再投票，否则没法往下玩啦。

Paxos规则演变： 得允许一个长老投多次

## 别把长老们的名声搞坏

假设南帝最早提议华筝，并且有三个长老(包括自己)同意了，这就形成了决议，虽然七公和黄老邪根本不在场。



图 1 过半长老投票形成决议

问题是: 七公在不知道决议形成的情况下，提议了黄蓉。对于已经在上轮投票的西毒和伯通来说，到底是支持还是不支持七公的提议呢？ 因为他们并不知道决议以及形成，只知道自赞成过南帝的提议而已！

1) 投过的不再投了。由于上次投票有没有形成决议不确定，可能造成一直没有形成决议。

2) 直接再投，可能形成了两次不同的决议，黄蓉和华筝都认为自己的订单是第一个被处理的!



图 1 不算太冒失的再提议(老顽童两次都投票)

Paxos规则演变(P2)：允许形成多次决议，但是决议内容必须相同！



图 1 先礼貌地问询下



图 1 重新形成内容相同的决议

Paxos规则演变(P2a)： 一旦形成决议，即v被选中，那么后续只能accept v。

例子很简单，如果已经超过半数投票，黄蓉的订单被选中为第一个处理，那么后续就只能同意黄蓉的订单排第一。

由于长老们接受的都是提议，必须要从提议的角度来解决。

为了让协议变得可操作，进一步演变：

Paxos规则演变(P2b)： 一旦对某个序号形成决议，假设对应的值是v，那么后续的提议只能是v。

例子很简单，如果已有经过半数的长老投票，黄蓉(v)的订单被选中为第1个处理(序号），那么对于第1个处理的对象，后续只能允许提议黄蓉，不允许提议其他的。

==> **隐含的要点**： 提议者跟提议内容(Value)的独立！

## 实现的问题

前面的规则演变，P2b是要求在已经形成的决议后，新的提议者只能提议该决议的值v。问题一：如何确定已形成决议的v？或者没有形成决议？

问题二：假设已经形成了决议，其他提议者，怎么知道形成了什么决议？

### 时间先后如何确定？

什么是之前？ 什么之后? 在异步网络环境下，各个机器的时钟可能不严格同步。 按照发送时间还是接收时间去比较？同样的两个消息，不同接收者的接收顺序可能不同。没有统一的时间(使用逻辑时间(PN)，保证其递增，参见4.5.1)

### 如何应对可能正在形成的决议？

可能某个决议正在形成，也可能没有；在一个决议的形成过程中(3个长老即可)，未参与该决议的长老可能又就发起新的提议。新的提议者并不知道有决议正在形成，也无法刻意等待决议形成。预知未来的决议太困难(让没发生的决议不发生，承诺机制，参见4.5.2)

### 无法及时获知的决议

假设形成了决议，但是唯一知道决议(根据大家的投票结果)的提议者长期云游去了。以上图为例，南帝提议华筝形成了决议，唯一知道决议形成的南帝，他在答复华筝后云游去了，长期不回复其他消息，其他人长期无法知道决议形成了。因此，无法保证及时、准确地获知已形成的决议(决议前必有value，充分考虑可能已产生的决议，使用其value，参见4.5.3)

## 先占坑再提议，可能在为别人做嫁衣

对于某个处理序号，从提议者角度需要保证两点：

1)如果我提议前，形成了决议接受了v，那么我提议的，一定要是v;

2)如果我提议前，并没有形成决议，那么我可以按照我的意愿去提议。

==》问题是： I从我询问完，到别人收到我的提议这段期间，可能在我不知情的情况下形成了某个决议！ II.或者别人跟我同时问，同时提议,各获得2票。

### PN是个什么鬼? 简单有效的逻辑时间

由于系统需要确定的时间比较先后的机制，而各个机器的物理时间不能直接用于比较，Paxos提出了使用Proposal Number（PN)的概念。这个概念非常容易让人混淆，我们先按照逻辑时间了来解释。

对逻辑时间的要求是什么？

两个不同的逻辑时间值，其比较结果是确定的；

不需要集中的逻辑时间产生器（集中的产生器及同步本身就是一个难题）；

不会产生重复的逻辑时间值。

在Paxos协议中，逻辑时间是整数，每个长老被分配单独的逻辑时间集合，比如七公使用的是个位数等于5的集合，可以是15, 25, ..105等，不断增长。其他长老类似，这样互相不冲突，可以直接比较大小。

**PN在本质上**是不断增长的逻辑时间！

### 承诺：不会形成逻辑时间更早的决议

难题：即使有了逻辑时间PN，由于异步化，我怎么知道PN比我小的提议，会不会在将来形成决议？如果是等待？ 等待多久呢？

Paxos对于这个问题的解决思路是：我们无法预知未来，但可以阻止未来可能发生的一些事 ==> 如果过半数的长老没有赞成过PN<a的提议，且承诺不会让PN<a的决议形成，那么它肯定无法形成 ==>占坑协议。

假设某长老提出了PN=a的占坑请求，收到请求的长老可能做的回应是：

1) 如果已有PN>a的提议或者占坑请求，那么直接拒绝或者无视即可；

2) 如果没有PN>a的任何请求，也没有PN<a的提议出现，承诺不接受PN<a的提议；注意，PN<a的占坑请求可能已经被同意，但是被废除了。

3) 没有PN>a的任何请求，但是已经接受了PN<a(PN=x, x<a，提议的值是v)的提议，则将x和v返回。

提议号与提议内容的松耦合。大家已经约定了占坑是有限保护。决议则是持久性的。



图 1 PN=101，占坑成功



图 1 PN=105，占坑成功



图 1 PN=101的提议被拒绝



图 1 PN=105的提议产生了决议(西毒自己也接受了)

### 小心行事：遵从可能已形成的决议

PN<a的提议，如果可能已形成(有长老收到过它的 value)，Paxos会保守地对待。

**占坑过程的可能结果如下：**

如果未收到majority个回复，占坑失败。增加PN重试。

如果收到了超过majority个回复(可能有少量拒绝或者没收到的)，并且没有返回任何PN<a的提议，那么PN<a的一定没有形成决议（因为两个majority必有交集）。

如果收到了超过majority个回复，部分带有PN<a的提议(含PN和value)，那么取其中PN最大的提议的value，作为新提议的value。注意，这个value不一定形成了决议，但是为了保守起见，仍然使用了它。

### 执行一轮协议的产出

对于一个提议者来说，执行了一轮两阶段协议(占坑和决议)的四种可能产出：

1. 被PN更大的的占坑请求给被废掉了，得重来；（自认倒霉）
2. 帮别人做了嫁衣(完成别人未完成的决议)；(帮别人就是帮自己)
3. 重复形成了别人的决议，Learn了状态修改；（求知的代价）
4. 让自己提议的内容，形成了决议。(目标达成)

## 一张图说明推理过程



## Leader的引入：效率，还是效率！

**上面的协议，不考虑后面提到的leader**，**到底能否让系统正确工作？** 先不说效率问题。

==>能，可能北丐在很多轮都是占坑被废，然后帮别人把序号i的决议形成或者重新做了一遍。然后再尝试i+1。这样可能很低效，有的请求可能非常倒霉地被推迟了很多轮。但是从系统的角度，再往前推进不断形成新的决议，且没有不一致问题。

Leader引起的两个主要改变：1) 两阶段协议，变成了大部分情况下的单次通讯，开销做到最低；2)当选期间，只有一个提议者，冲突很少。

效率体现在两个方面：

用最少的报文完成一个决议；

避免相互纠缠

1. ==> prepare request(proposal number n)

2. <==, response(保证不accept比n小的，保护我已经accept的最大的proposal number accepted.val, accpeted.num)

3.==> accept request(proposal number n, val v)

4.<== accept (n, v)

acceptor需要持久化的部分： max\_responded\_pn (proposal number), max\_accepted\_pn, max\_accepted\_val.

requestor需要持久化的部分：last\_proposed\_pn, last\_proposed\_val，需要二者一致

P1, P2, P2a, P2b, P2c之间的关系，已经在visio图中有，但是不美观。

既然一个value被majority个accept后，就称为被chosen，如果一个acceptor能够accept多个value，那么会形成混乱，多个都被chosen。因此，每个acceptor只能接受一个value。

为什么要有requirement P1?

**P1**. An acceptor must accept the first proposal that it receives //首个必须接受

==>不考虑消息丢失等问题，如果只有一个proposer，只propose了一个value，我们也希望能成功。所以必须acceptor必须接受其收到的第一个proposal。

按照P1，可能存在多个proposal被并发，每个都被少数派accept，从而不能形成结论，即不能chose任何value。为了保证这种情况下能够继续，或者多轮proposal能最终成功，就要求每个acceptor能够允许接受多个proposal。

为了不混淆，需要对各个proposal进行编号，编号方法另外描述。

既然允许每个acceptor接受多个proposal，并且一个proposal一旦被majority个接受，那么其value就被选择。那么就可能有多个value被chosen，即多次形成决议。注意，majority个acceptor接受即认为形成决议，但是proposer可能并未及时收到决议，或者决议本身丢失了，导致新一轮proposal。为了保证多次可能的决议的一致性，这就要求：

**P2**. If a proposal with value *v* is chosen, then every higher-numbered proposal that is chosen has value *v*. **//****只有一个value能被选中**

满足P1的一个自然想法是：

**P2*a***. If a proposal with value *v* is chosen, then every higher-numbered proposal accepted by any acceptor has value *v*. **//只允许接受已被选中的v**(在已有value v被选中的情况下)

由于通信是异步的，一个刚刚醒来的proposer，根本不能知道那些value被accept了，但是其proposal的number可能更大。按照p1，acceptor需要接受它，但是可能违反了P2a。为此，对P2a进行加强。即为了同时满足p1和P2a,P2a变成：

**P2*b*.** If a proposal with value *v* is chosen, then every higher-numbered proposal issued by any proposer has value *v*.  **//只允许提议已被选中的v** (在已有value v被选中的情况下)

如何满足P2b的一段推导，很有意思：

原文下面一段，需要好好理解：

Combining this with the induction assumption, the hypothesis that *m* is chosen implies:

Every acceptor in *C* has accepted a proposal with number in *m . .* (*n −* 1), and every proposal with number in *m . .* (*n −* 1) accepted by any acceptor has value *v*.

我认为这里面的C，可能只是个泛指。对于不同的proposal number，即*m . .* (*n −* 1)，对应的C可能是不同的。但是都有个majority集合。

Since any set *S* consisting of a majority of acceptors contains at least one member of *C* (这句话是很重要的过渡。前面没有提到S，为什么？), we can conclude that a proposal numbered *n* has value *v* by ensuring that the following invariant is maintained:

**P2*c***. For any *v* and *n*, if a proposal with value *v* and number *n* is issued, then there is a set *S* consisting of a majority of acceptors such that either (a) no acceptor in *S* has accepted any proposal numbered less than *n*, or (b) *v* is the value of the highest-numbered proposal among all proposals numbered less than *n* accepted by the acceptors in *S*. **//****总是提议能看到的PN最高的value**

实际上，这个不变式就是为了保证proposal number变化时，*v*的不变性。即要么S中任意一个都没有accept过小于n的proposal，这时候肯定没有形成一致，即没有一个value被accept了；要么v就是S中acceptor接受过的proposal number最高prosoal对应的value。

如何从P2c推出能达到P2b?

如果在PN 为m时，v被选中了，那么表示有个majority C，全部accept了v。

我们要论证按照P2C这个不变式，在PN为m+1.. n时，一定能会propose v。

在确定PN=m+1的proposal 的value x时，我们找到了某个majority，S中的最大value，那么S一定包含C中某个acceptor，因为二者都是majority。而m是最大值，所以一定选择对应m的value，即v。

在PN=m+2..n区间，考虑任意的PN i，由于在PN属于[m, i-1]的proposal 的value都是v，而S必然至少有一个与C重合，因此查到的PN最大的value，仍然是v。

highest-numbered 为什么重要？

首先，既然说highest，那么就隐含着，大家accept的proposal number可能是不同的。

其次，不存在number x, x > highest-number，且被majority个接受了。否则S中一定有至少一个接受了x，矛盾。

我理解，这样好实现，在后面，直接让acceptor承诺，不accept小于n的proposal。

注意，在讨论前面这些条件时，并没有牵涉到prepare 和 accept两个阶段，我认为是按照就一个阶段讨论的。