# 状态膨胀和无状态性

*此文谨献给 Alexey Akhunov 和 Igor Mandrigin，感谢你们在无状态性概念上的贡献。*

## 一. 引言

本文的目的在于向大家介绍一种解决 “状态数据膨胀” 问题的技术路径 —— “无状态性（statelessness）”。“状态数据膨胀” 是所有允许用户自主写入状态数据的公链都会面临的问题，指的是因为用户及合约的不断增加，在全节点处保存的状态数据会越来越多。不断增加的状态数据会带来一系列风险，包括：抬高全节点的运行门槛；使链上操作的定价失衡。

在本文中我主要拿以太坊区块链来举例，这是因为我对以太坊最熟悉，但状态数据膨胀问题是普遍的，不是以太坊独有的。我同样也会举出我所知的其他项目对这个问题的思考。

为使大家充分理解 “无状态性” 的意义，我少不了要在开篇交代一下问题的背景。然后我会介绍以太坊社区为此提出的几个概念，以及相关的 开放问题/权衡取舍。最后，我会再回到最初的问题，以期通过对问题的多样描述（以及引出来的解决方案思路），拓宽对该问题的思考。

## 二. 状态数据膨胀问题

可以把 “区块链协议” 理解为一种让不定数量的计算机的行为统一步调成一台计算机的协议。那么，其运行会在参与相关协议的计算机（即 “节点”）处产生两种数据：一种是**区块数据**，即我们常说的区块链，这部分数据记录了这个网络（这台计算机）在过去发生的所有事情；另一种是**状态数据**，即代表当前整个网络的状态的数据，对以太坊来说，“状态” 的内涵包括：哪个账户有多少余额、已经发出过多少笔事务（不包括曾经发出的事务的内容，那是区块数据）；哪个合约的代码是什么，其内部的哪个存储项的值是什么；还有一些与共识机制运行相关的数据。

状态数据的特殊性在于：一方面，状态数据是历史区块（所包含的事务）执行之后的结果；另一方面，它又是执行新区块的前提。因此，在当前的以太坊区块链上，全节点必须保存状态数据，这样才能（通过执行区块来）验证新收到区块的合法性。

可以想象，因为用户和合约的数量会不断增加，如果不施加一些控制，状态数据的规模是会不断增大的。这就是状态数据膨胀问题。

状态膨胀问题带来的影响是确定的：它会让区块验证变得越来越困难。因为同样是读取和写入一个状态对象，在状态对象有 10 万个的时候，比之状态对象只有 1000 个的时候，资源开销是上升的。也正因此，状态膨胀会使运行全节点的门槛提高（单位时间内，资源的开销 —— 主要是硬盘随机读写 —— 不断上升），还会使 EVM 各操作的 Gas 开销比例逐渐失衡，产生对节点的 DoS 攻击向量。（注：资源开销的上升幅度与客户端软件的实现有关。）

但状态膨胀问题的原因为何（因此应当如何纠正），则有许多种不同的角度。比如，Vitalik 对该问题的归因[1]是：

> “这些操作只需事务的发送者一次性缴交按 gas 用量来计量的手续费，但会给整个网络造成永久的持续性成本”

换言之，问题的根源是支付和成本的不匹配 —— 当前的以太坊协议既做不到能够合理地删除状态来给区块链 “瘦身”[2]，又做不到让用户持续为保存自己的状态付费。假设能做到，这个问题就能得到控制，不会这么严重了。因此，他倾向于 “状态保质期” 方向的改进：用户创建或更新某个状态对象之后，只能保证全节点会在一段时间内存储这个对象；逾期则必须采用额外的手段（因此是额外的支付）来 “复活” 这个对象。

对问题的其他归因和 “状态保质期” 概念，我们后文还会讲到。现在我们先转向 “无状态性（statelessness）”。

*注 1：Vitalik Buterin，https://ethfans.org/posts/a-theory-of-ethereum-state-size-management*

*注 2：在当前的以太坊协议中，的确有操作（`SELFDESTRUCT`）可以做到删除合约的存储项，也为该操作码的使用提供了一种粗糙的激励机制（返还 Gas 机制）。但事实证明，返还 Gas 机制不仅没有改善状态膨胀问题，还促成了 Gas Token 的出现，加剧了状态膨胀问题；而 SELFDESTRUCT 则导致了别的一些问题。见《务实地取消 SELFDESTRUCT》*

# 三. “无状态性” 诸概念

## （一）无状态性

“无状态性” 背后的观念是：状态膨胀问题之所以是一个问题，是因为验证区块的节点必须先有状态数据（先得知道某个账户有多少钱），然后才能验证区块（然后才能知道该账户花 X 元到底合不合法）；但如果我们能做到无需在本地存储状态数据就能验证区块，那就无需担心这个问题了。

具体做法是，在广播区块时，附带传播一份对该区块内所有事务所访问状态的证据，一个完全不保存任何状态的节点，也可以凭借这些证据恢复出恰好足以验证该区块内所有事务的状态树，并且验证之后可以把这些状态数据都丢弃。这里的 “所访问状态的证据”，通常称为 “见证数据（witness）”。

无状态性代表着对状态膨胀问题的釜底抽薪式解决方案，彻底免去区块验证对状态数据的需要，自然就不需要担心状态膨胀的问题了。另一方面，各节点还可以自行选择存储哪些状态，允许开发者为自己的使用场景定制所需的客户端。

但是，从节点的角度看，这意味着一种交换：以增大带宽开销为代价，来节约硬盘的读写和存储开销（尽管读写开销确实是当前全节点的成本中最大的一部分）。那么见证数据到底有多大？

如果不对以太坊协议作任何改变，见证数据的大小会在 MB 级别 [3]）。相比之下，当前（区块 Gas 上限为 1250 万）的以太坊区块数据量约为 45 KB [4]。由此看来，见证数据的带宽开销是非常大的。过去两年中，无状态以太坊研究的主要方向，就是思考如何才能降低见证数据的大小，方向有：（1）代码默克尔化，减少需要包含在见证数据中的合约代码数据 [5]；（2）改变以太坊状态树的格式，比如将十六进制树改为二进制树 [6]，或者将默克尔树改为 KZG 承诺树（也就是所谓的 Verkle Tree [7]）；（3）以区块为单位提供见证数据，而非以事务为单位，这样可以免去一些重复的证据。改为二进制树可以节约一半的带宽开销甚至更多；而 Verkle Tree 可以产生固定大小的见证数据。

除了带宽开销增大以外，无状态性还带来了一种革命性的变化：网络中可能大部分节点都不会保存状态数据，至少不会保存全量的状态数据。那么以往我们习以为常的属性，在无状态客户端为主的网路中就不一定能实现了。这些问题都要重新思考：

1. 当前条件下，用户无需担心谁能为自己提供钱包服务；因为自己账户的状态在全网所有全节点处都有备份；但是 “无状态性” 实现后，谁来为用户提供见证数据呢？用户如何知道哪里的节点存储了自己的状态？一种乐观的假设是：出块者处一定存储了全部的状态，否则就会影响其出块；因此至少出块者一定能提供这样的服务。但这会影响钱包服务的可得性吗？（实际上，在撰文的此刻，我已经不再担心这个问题了。假设无状态性的带宽开销能降低，全状态节点的门槛也会降低，定制化状态节点的门槛也会降低，所以钱包服务的提供门槛实际上是降低了的。可以认为无状态性确实改变了钱包服务提供的假设，但不应该认为使之恶化了）
2. 如果一个节点是无状态的，仅保存了区块数据，虽然它能参与历史区块广播和历史事务广播，但它没办法参与新事务的广播。准确点来说，TA 是没法验证新收到的事务的有效性的。那么这样的节点要如何在 P2P 网络中存在（其他对等节点会把它当成一个无用的节点而中断链接）？事务要如何在网络中传播？这是否意味着我们不论如何都需要某种以事务为单位提供见证数据的机制？还是说无状态节点只能在一个专门的次级网络中生存？（基于后者的解决思路是形成一个可以按需请求状态数据的次级网络 [8]，就像我们常用的 BitTorrent 种子下载网络一样；当然，为广播中的事务增加见证数据也是一个办法）
3. 如何为见证数据指定 Gas 开销？又或者说，如何限制见证数据的大小？

上面这个清单仅仅是当前的研究成果，它仍有可能变得更长。这些运行假设的变化，极有可能比它的带宽开销上的影响更为重要；如果不分析清楚这些假设，就没有办法判断无状态性是否真的值得实现，或者何种实现方式更好。

当前，有一种对无状态性的分类是：

- 弱无状态性：以区块为单位提供见证数据；出块者仍需保存全部状态数据
- 强无状态性：以事务为单位提供见证数据，理论上不需要保持全部状态数据也可以出块

在我看来，两种方向存在短长，但不应认为网络的最终形态可以被规划出来。从节约带宽的角度，弱无状态性是更优选择；但相应地也要面对上文所述新事务传播问题。后文我们还会回到这个分类。

下面我们进入第二个概念：“准-无状态性”。

*注 3：Alexey Akhunov，https://ethfans.org/posts/data-from-the-ethereum-stateless-prototype，本篇为理解“无状态性”最佳的入门读物。在 Alexey 所测试的主要时间段里，区块的平均大小约为 20 KB（800 万 Gas），而测算出来的见证数据平均大小在 1MB 左右。这意味着，在当前的区块大小下，必须预期见证数据有更大的大小。*

*注 4：https://etherscan.io/chart/blocksize*

*注 5：Sina Mahmoodi，https://ethfans.org/posts/evm-bytecode-merklization*

*注 6：Igor Mandrigin，https://ethfans.org/posts/stateless-ethereum-binary-tries-experiment*

*注 7：Dankrad Feist，https://notes.ethereum.org/_N1mutVERDKtqGIEYc-Flw*

*注 8：Piper Merriam，https://ethfans.org/posts/state-availability-getnodedata-dht-approach-dev-update*

## （二）准-无状态性

准-无状态性的概念来自这篇文章 [9]。其背后的直觉是：无状态性对见证数据的要求是能帮助完全无状态的节点从 0 开始建构出能验证相应区块的状态树，但这很有可能浪费了带宽资源；因为一旦收到了见证数据之后，节点就已经有一棵稀疏的状态树了，这棵树是可以重用的，犯不着把它全删了，收到一个新区块又从 0 开始建构。

因此，假定节点在收到一个区块前，其状态数据的存量情况是可知的，我们就可以借助见证数据为之提供一个状态增量，这个增量加上其初始存量，恰好足以建构出能验证新区块的状态树。如此一来，我们就既能获得无状态性的大部分好处（通过附带见证数据，让区块验证的硬盘读写开销最小化），又不需要像无状态性那样，付出那么高的带宽开销。

而且，准-无状态性还有一个额外的好处：当状态数据变得越来越多，状态树变得越来越深越来越密，无状态下的见证数据理论上来说也会不断增大，即带宽开销趋于上升；但准-无状态性的见证数据仅提供增量数据，所以其见证数据也是最小化的。

麻烦在于，这个“节点的状态数据存量”如何可知？理论上来说，除非我们从某个一致的存量开始实行准-无状态性，否则是无法对这个存量达成同步的。但只需“一致”，这个数目具体是多少，无关紧要。由此我们发现了准-无状态性的另一个优点：假使状态数据已经变得过于庞大，我们可以将它们“清零”，从无状态重新开始，启动一次新的、由富状态节点（比如出块节点）向验证节点传送见证数据（和状态）的进程。

这就是 ReGenesis，定期重启以太坊这台计算机。

*注 9：Igor Mandrigin，https://ethfans.org/posts/semi-stateless-initial-sync-experiment*

## （三）Regenesis

ReGenesis 的构想最早由 Alexey 提出 [10]，更细致的解读可见此文 [11]。大意是：定期（比如每 100 万个区块）将协议设定为空状态，开始实施准-无状态的流程 —— 每一个区块在传播时，都必须附带一些见证数据，使得一个从上一次 regenesis 事件开始清零了状态数据、并获得了自那以来所有区块附带见证数据的节点，都能验证该区块。

也就是说，每次触发 regenesis 事件，一个只求验证区块、不求产生区块的节点就可以把本地的状态数据清零，凭借新收到的区块附带的见证消息，建构起刚好足以验证该区块的状态树；每个见证数据，都会带来增量的状态数据，只要节点缓存了自己的状态树，则见证数据都是刚好够用的。当节点本地的状态数据足够多，甚至有可能出现无需附带见证数据的区块。但是，又不必担心状态数据的膨胀问题，因

为节点会定期清零状态。

ReGenesis 是一个特别均衡的方案（实际上也是当今的 “无状态以太坊” 研究的主要方向）。而且，因为它追求的是准-无状态性，虽然 ReGenesis 要求实现基于事务的见证数据，它绕过了基于事务见证数据的无状态性的一个缺点：需要频繁更新一笔事务的见证数据，每一次状态根更新（也即每一次出块），都要求更新未打包事务的见证数据；但是，在准-无状态性中，准-无状态节点每收到一个区块，本地的状态就会多一些，而事务被打包的时间必定晚于其广播的时间，所以事务在广播时附带的见证数据必定足以应付验证的需要。

那么它的缺点在哪里呢？

（1）如果一笔事务在事先并不能确定自己会访问哪些交易（这被称为 “动态访问模式” [12]），则有可能并不能提供足够的见证数据并因此失败。这并不是一个不可解决的问题，因为根据准-无状态性，只需多次发起并补充见证数据，总能成功。但这总归意味着用户可能要多次发送同一笔事务。

（2）不确定 ReGenesis 下节点会不会也抛弃区块数据，所以 rollup 这样 layer-2 方案可能会受到影响。

从我们上面推论下来，ReGenesis 很像一种无状态或者准-无状态方案。但是，从另一个角度看，ReGenesis 也很像一种 “状态保质期” 方案或者 “状态租金” 方案：每过一段时间，所有的状态对象都会过期、失活，而你为了让自己的状态复活，必须支付额外一笔代价；而用户的一次付费，也只能保证相关状态在节点处能存储一段时间，而无法保证能永久存储。

从这个角度看，ReGenesis 会面临另一个所有状态保质期方案都会面对的问题：“复活冲突”。如果有人在已经失活的状态存储位置直接创建了一个新状态（而不是通过提供见证数据来复活这个状态），那就会产生冲突 —— 从前的富状态节点（存储着已经失活的旧状态）无法与无状态节点（存储着新建的状态）达成共识。对应的 ReGenesis 上，就是 regenesis 事件触发后，有人尝试在原本已经有状态的存储位置上新建一个状态（比如直接给自己的账户存进 10 ETH），一直存储着状态数据的富状态节点，就将无法与刚清零了状态的无状态节点，对状态根达成共识（因为余额对不上）。

Vitalik 为此提出的解决方案 [12] 是：给账户安排一个与其创建时期相对应的标识符（例如，同一个地址 0x1234，在第 0 个 regenesis 时段创建的账户就标记为 (0, 0x1234)，而第 4 个 regenesis 时段创建的账户就标记为 (4, 0x1234)），从而明确用户到底是想复活自己在从前时段创建的状态，还是想新建一个状态，从而解决复活冲突问题。

当前，ReGenesis 已经完全被当成一个状态保质期方案，与弱无状态性所代表的无状态性方案相竞争 [12]。但在我看来，由于其运行需要使用见证数据，而围绕见证数据使用的研究是从无状态性的研究中生发出来的，所以无状态性是一个不应被放弃的视角，尤其是准-无状态性的概念，非常富有启发性。

*注 10：Alexey Akhunov，https://ethfans.org/posts/regenesis-resetting-ethereum-to-reduce-the-burden-of-large-blockchain-and-state*

*注 11：Igor Mandrigin，https://ethfans.org/posts/37566*

*注 12：Vitalik Buterin，https://ethresear.ch/t/resurrection-conflict-minimized-state-bounding-take-2/8739*

### （四）总结

从无状态性和准-无状态性的概念出发，我们最终走向了 ReGenesis 这种较为均衡的方案。从状态保质期的角度出发，也可以得出 ReGenesis 相当有竞争力的结论。但是，这绝不意味着无状态性的研究已经完成或者已被淘汰。事实上，我们还未能确定无状态性带来的所有影响是否都已被分析清楚了，因此，也无法断言已经出现了绝对值得实现的方案。而且，就算方案确定了，工程上获得安全高效的成果，仍有待时日（比如，假设真的把状态树从默克尔树改为 Verkle Tree，工程上需要花多少时间？）。因此，对无状态性的实现，我并不那么乐观。

接下来，我们重新回到状态膨胀问题，看看对这个问题的不同表述，会不会给我们带来一些启发。

## 四. 问题到底出在哪儿?

状态膨胀问题到底意味着什么？它在何种意义上是一个问题？如何为之归因并提出可行的解决方案？如何衡量一种解决方案是否有价值？

你至少可以从以下几个方面，来思考状态膨胀给我们带来的影响：

（一）它会使区块验证的成本不断升高，因此抬高全节点运行的门槛；

（二）它意味着一种只增不减的永久性负担，对网络的长期存续和去中心化是一种威胁；

（三）它意味着链上操作的 Gas 定价有持续存在的失衡风险（也是 DoS 风险）；

我们一个一个来。

## （一）验证成本

为什么状态数据的增加会提高区块验证的成本？因为对以太坊来说，（1）区块的执行和区块的验证是一回事，都是完全执行所有的计算过程；（2）区块验证要求本地具备最新的状态数据；（3）状态数据的增加意味着对单个状态对象的读取和写入变得更困难。这三个条件中，任一因果关系被打破或被削弱，状态膨胀给区块验证带来的影响都会被削弱甚至消除。

假如区块的验证与执行不是一回事、区块验证的开销是个常量，则状态膨胀问题不会影响区块验证的成本；这就是 Mina 等新兴区块链采取的思路：每次出块，出块者都会为区块的执行过程生成一个计算完整性的证明，验证方只需验证这个证明就可以验证相关区块的合法性，而验证的开销是个常量，这就打破了状态膨胀对区块验证成本的影响。理论上来说，以太坊也可以引入类似的技术；而我们要付出的代价是：找出一种安全可靠的、能够为通用计算实现计算完整性证明的工程实现、可能完全改变应用开发者的体验、需要重新设计链上操作的 Gas 消耗量比例。

同理，无状态性就是在打破或说影响第（2）条因果关系。实现无状态性之后，区块验证就不再要求本地存储最新的状态数据了，但它还留下了一种风险：随着状态数据的增加，见证数据增大（因此验证区块的带宽开销增大）的风险。

而（3）则向我们指出，优化客户端同样有助于减少这个问题对我们的影响。举例而言：Geth 客户端 1.10 版 [13] 实现了状态快照，该功能可以保证状态读取的开销是一个常量，也即消除了状态膨胀对状态读取的开销的影响；但是，即使优化到极致，状态写入的开销仍然会随着状态数量的增加而增加。

*注 13：Péter Szilágyi，https://ethfans.org/posts/geth-v1-10-0-introduction*

## （二）永久性负担

从这个角度看，问题出在：（1）用户可强制要求节点存储自己的状态；（2）支付与成本在时段上的不匹配，一次付费永久存储。

无状态性的激进和革命之处就在于，它针对的是（1），而不是（2），也即它不把这个问题解释为一个经济机制问题，也不考虑改变经济模型来解决这个问题。无状态性完全消除了节点与本地的一部分特殊数据（状态数据）交互以验证区块的需要，从而完全免去了存储状态数据需要。实现了无状态性之后，存储部分还是全部状态完全由节点运营者自己决定，用户实际上无法强制所有节点来存储自己的状态，只能预期矿池或者钱包服务提供商这样的专业运营商会存储自己的状态，而这是完全公平的。

而针对（2）的解决方案可分为两类：一类的倾向是：一次付费、有限时段存储；无限时段存储、持续付费。前者的代表是各种状态保质期方案；后者的代表是 Nervos 区块链。

在 Nervos 区块链上，用户要存储状态时就必须绑定一定数量的区块链原生资产 CKB，绑定的代币数量与可写入的状态数据大小是成恒定比例的；由此，用户在存储状态时，虽然没有货币支出，但因资产绑定而承受的不便利（或者说损失的机会收益），就是一种持续的、与存储状态的时长一一对应的成本。因此，这一模式约束了用户存储状态的行为；而且 CKB 的数量就构成了状态数据的上限，控制了状态膨胀。

而状态保质期方案，相比无状态性就显得有点半心半意 —— 在许多状态保质期方案中，它虽然能约束状态数据的大小（因为给定时间段内能刷新的状态对象数量是有限的），但不一定能降低与特殊数据交互的需要，它需要额外的数据结构来记录失活状态的信息，也需要与这一数据交互。解决复活冲突实际上也就是在解决这个问题。

另外，包括 ReGenesis 在内的状态保质期方案也意味着，整个网络的事务处理容量，会被一类并不创造经济价值、仅仅是为了缴纳租金（或者说购买保险）的状态复活事务给占去一块。这使人疑心状态保质期到底是不是一个长期可持续的、能获得民意支持的方向，毕竟，用户是会越来越多的。

### （三）操作码 Gas 定价失衡

链上操作的 Gas 消耗量的绝对值是没有意义的，一个操作消耗 100 Gas 并不意味着什么；有意义的是不同操作的 Gas 消耗量比例：一个消耗 100 Gas 的操作的计算开销被认为是一个消耗 1 Gas 的操作的 100 倍。它的意义有两重：（1）它引导应用开发者优化代码的方向，假设两个操作路径都可以达到同样的效果，而一条路径的 Gas 开销更低，开发者就会实现这个路径；（2）如果这个比例失衡了，那就意味着某种操作的名义开销（Gas 消耗量）偏离了其实际开销，而这样的操作码就更有可能被利用来发起对节点的 DoS 攻击。著名的 “上海攻击” 就是这样：攻击者在事务中包含了大量 Gas 消耗量很低、但节点实际计算开销很大的操作码，而处理这些事务会使节点负载过大而掉线。

状态膨胀就会引发这样的失衡问题：状态操作的 Gas 开销是一个常量，但状态读写的开销实际上并不是一个常量，它会随着状态数量的增加而增加。那为什么状态操作的 Gas 开销不能实时、动态变化呢？因为状态读写逻辑上是由 EVM 来实现的，但实际上并不是，它是由客户端软件的数据库来实现的，不同客户端的实际开销既不统一，也无法被 EVM 感知。

值得提醒的是：上述两种效果是会相互叠加的。如果相关操作的名义开销比例，恰好引导着应用开发者使用这种实际开销被低估的资源，则问题就会变得更严重 —— 其实状态膨胀问题本身就是这样的一个例子，状态读写的名义开销定得太低了。

当前以太坊社区的做法是，通过硬分叉来调整不同操作的 Gas 消耗量。这就等于是说，以太坊当前的治理程序，充当了 EVM 感知状态操作开销的信息输入机制。这样做当然不能完美地解决这个问题，它同时也提醒了我们，以太坊这台电脑有多么依赖人类工程师的定期修理。

此外，合理的 Gas 定价还要面临的另一个挑战是：它要用同一个度量单位，来度量本质上需要不同资源的操作的开销比例。

谈到这一点，是因为无状态性也受该问题的影响。在当前，节点执行区块的主要开销并不是计算，而是状态的读写；而使用见证数据取代了状态数据之后，主要的开销就变成了带宽。当然，出块者生成见证数据需要计算，但即使我们不考虑这一点，或认定当前的状态操作名义消耗量已足够准确，我们仍必须考虑怎么约束见证数据大小的问题。好的办法无疑仍是价格，但其 Gas 消耗量应如何确定呢？（注：CALLDATA 类型数据的定价就有点像是根据带宽消耗量来定价。这值得我们深思。）

### （四）回到无状态性

了解了这些问题，我们也许更能明白无状态性究竟解决了什么问题，又留下了哪些问题等待解决。实际上，我们再回过头来看无状态性，其原理是很单纯的：以带宽开销换取硬盘读写的节约。但是，能截然认为这一定是值当的吗？假设我们生活在一个硬盘读写就是很便宜，而带宽就是非常稀缺的世界里呢？或者说，假设日后两种资源的价格会倒转过来呢？

换言之，无论选择无状态性还是保持原状，从事后来看可能都是不够灵活的选择 —— 最理想的情况，应该是节点能自己选择牺牲哪个换取哪个，并随时切换，而不是由一个治理程序强制大家选择某一种，这种选择再好，终究只是有限头脑的智慧的产物，而且都不能排除要依赖治理程序（在日后把选择颠倒过来）的风险。

这是否意味着，最优的方案应该是一个跟当前的以太坊网络部分重叠、平行运行的无状态客户端网络？用户能自由加入，自由切换？我们能拥有这样的方案吗？

## 五. 结语

自 19 年 3 月了解了 “无状态性” 概念以来，我便认定，无状态性是以太坊最值得研究的底层协议演化方向，唯有它能既解决状态膨胀问题，又维持以太坊的经济属性。今天我仍这么相信。只是，对状态膨胀问题的思考，会不断牵涉到以太坊的底层范式 —— 全局状态和链上计算 —— 最终具象化为一个问题：如何为链上操作制定名义开销比例（gas cost）？无论要对以太坊作什么改进，都绕不开这个问题。最终你也会得到跟我一样的结论：以太坊的设计失衡之处不是一处，而是两处。无状态性之后，我们仍有迢迢长路。

的确，本来，对任何弥赛亚的期待，都是一种幻想。此际，让我们先解决以太坊长期生存道路上最大的障碍：状态膨胀。