

区块链系统的管理方法和区块链节点

5 本申请要求于 2023 年 12 月 29 日提交中国国家知识产权局、申请号为 202311862343.4、
申请名称为“区块链系统的管理方法和区块链节点”的中国专利申请的优先权，其全部内
容通过引用结合在本申请中。

技术领域

10 本说明书实施例属于区块链领域，尤其涉及一种区块链系统的管理方法和区块链
节点。

背景技术

15 区块链（Blockchain）系统是分布式数据存储、点对点传输、共识机制、加密算
法等计算机技术的新型应用模式。区块链系统中按照时间顺序将数据区块以顺序相连
的方式组合成链式数据结构，并以密码学方式保证的不可篡改和不可伪造的分布式账
本。由于区块链系统具有去中心化、信息不可篡改、自治性等特性，区块链系统受到
人们越来越多的重视和应用。

发明内容

20 本发明的目的在于提供一种区块链系统的管理方法和区块链节点。

第一方面，提供了一种区块链系统的管理方法，所述方法包括：第一区块链系统
接收第一交易，所述第一区块链系统中部署有包括第一方法函数的第一卷叠合约，所
述第一卷叠合约的合约状态中存储有 N 个第二区块链系统的注册信息，所述第一交易
用于调用所述第一方法函数，请求将第三区块链系统作为第 i 个区块链系统对应的上
25 一层区块链系统，所述第三区块链系统是所述第一区块链系统和其余 N-1 个第二区
块链系统中的一个；所述第一区块链系统根据所述第一交易执行所述第一方法函数，实
现：生成主权迁移事件，所述主权迁移事件中包括所述第 i 个第二区块链系统和所述
第三区块链系统的标识信息；所述第三区块链系统根据所述主权迁移事件，利用所述
第 i 个第二区块链系统的注册信息，在所述第三区块链系统中设置与所述第 i 个第二
30 区块链系统对应的第二卷叠合约。

第二方面，提供了一种区块链系统的管理方法，所述方法由部署有第一卷叠合约
的第一区块链系统执行，所述第一卷叠合约中包括第一方法函数，所述第一卷叠合约
的合约状态中存储有 N 个第二区块链系统的注册信息，所述方法包括：接收用于
调用所述第一方法函数的第一交易，所述第一交易请求将第三区块链系统作为第 i 个
35 区块链系统对应的上一层区块链系统，所述第三区块链系统是所述第一区块链系统和
其余 N-1 个第二区块链系统中的一个；根据所述第一交易执行所述第一方法函数，实
现：生成主权迁移事件，所述主权迁移事件中包括所述第 i 个第二区块链系统和所述
第三区块链系统的标识信息，使所述第三区块链系统根据所述主权迁移事件，利用所
述第 i 个第二区块链系统的注册信息，在所述第三区块链系统中设置与所述第 i 个第

二区块链系统对应的第二卷叠合约。

第三方面，提供了一种第一区块链系统中的区块链节点，所述第一区块链系统中部署有第一卷叠合约，所述第一卷叠合约中包括第一方法函数，所述第一卷叠合约的合约状态中存储有 N 个第二区块链系统的注册信息，所述区块链节点包括：交易接收单元，配置为接收用于调用所述第一方法函数的第一交易，所述第一交易请求将第三区块链系统作为第 i 个区块链系统对应的上一层区块链系统，所述第三区块链系统是所述第一区块链系统和其余 N-1 个第二区块链系统中的一个；交易执行单元，配置为根据所述第一交易执行所述第一方法函数，实现：生成主权迁移事件，所述主权迁移事件中包括所述第 i 个第二区块链系统和所述第三区块链系统的标识信息，使所述第三区块链系统根据所述主权迁移事件，利用所述第 i 个第二区块链系统的注册信息，在所述第三区块链系统中设置与所述第 i 个第二区块链系统对应的第二卷叠合约。

第四方面，提供了一种计算设备，包括存储器和处理器，所述存储器中存储有可执行代码，所述处理器执行所述可执行代码时，实现第二方面中所述的方法。

第五方面，提供了一种计算机可读存储介质，其上存储有计算机程序，当所述计算机程序在计算设备中执行时，计算设备执行第二方面中所述的方法。

本说明书实施例提供的技术方案中，在位于 Layer1 的第一区块链系统中第一卷叠合约的支持下，位于 Layer2、Layer3 甚至更低层的区块链系统能够快速完成主权迁移，即快速更换其对应的上一层区块链系统，无需各区块链系统进行线下沟通，能够同时兼顾效率 and 安全性。

附图说明

为了更清楚地说明本说明书实施例的技术方案，下面将对实施例描述中所需要使用的附图作简单地介绍，显而易见地，下面描述中的附图仅仅是本说明书中记载的一些实施例，对于本领域普通技术人员来讲，在不付出创造性劳动性的前提下，还可以根据这些附图获得其他的附图。

- 图1是一实施例中Layer2的原理示意图之一；
- 图2是一实施例中Layer2的原理示意图之二；
- 图3是一实施例中Layer2的原理示意图之三；
- 图4是一实施例中Layer2的原理示意图之四；
- 图5是一实施例中采用Rollup技术的多层卷叠网络的示意图之一；
- 图6是一实施例中区块链系统主权迁移方式的示意图之一；
- 图7是一实施例中区块链系统主权迁移方式的示意图之二；
- 图8是一实施例中区块链系统主权迁移方式的示意图之三；
- 图9是一实施例中提供的一种实现多层网络卷叠的方法流程图；
- 图10是一实施例中提供的一种智能合约的合约状态存储示意图；
- 图11是一实施例中提供的一种区块链系统的管理方法的流程图；
- 图12是一实施例中提供的一种区块链节点的结构示意图。

具体实施方式

为了使本技术领域的人员更好地理解本说明书中的技术方案，下面将结合附图，对本说明书实施例中的技术方案进行清楚、完整地描述，显然，所描述的实施例仅仅是本说明书一部分实施例，而不是全部的实施例。基于本说明书中的实施例，本领域普通技术人员在没有作出创造性劳动前提下所获得的所有其他实施例，都应当属于本说明书保护的范围。

分布式系统中存在经典的CAP定理-Consistency（一致性）、Availability（可用性）、Partition tolerance（分区容错性），三者不可兼得，简称“不可能三角”问题。区块链系统中也存在一个不可能三角：效率、去中心化、安全。尽管这里的区块链系统“不可能三角”还没有明确的理论论证，但是是对现有区块链系统的总结。这里的效率、去中心化和安全，分别定义如下：效率：每秒处理交易的数量，即TPS（Transaction per Second）；去中心化：参与节点门槛足够低，保证系统中有大量分布式的节点；安全：对区块链系统发动攻击的难度足够大。

针对上述区块链系统“不可能三角”的问题，公链项目，也是Layer1/主网，选择了安全性和去中心化，牺牲了效率，目前只有大约12~15TPS。当有大量交易需要处理时，尤其是很多操作复杂的交易时，Layer1就会拥堵。除了普通的转账交易外，Layer1还作为DeFi（去中心化金融）和NFT（同质化代币）等火爆应用的主要平台，在交易盛行时，Layer1拥堵的问题就变得十分严重，带来较大的交易延迟。

高昂的Gas fee（汽油费，或交易费）很多时候成为交易的障碍。尽管已从PoW（工作量证明）转向了PoS（权益证明），但这只是改变了区块链上记账权的获取方式，而需要支付给获得记账权的节点的交易费没有明显减少。这是因为Gas费的设计针对的是节点执行交易（包括执行智能合约代码）的消耗，这部分消耗不改变的情况下Gas费并不会降低。例如，一笔去中心化交易所的货币兑换交易的交易费在拥堵时可能会超过100美元，这让很多用户望而却步。

Layer 2技术是构建在Layer1之上的扩展性解决方案，旨在解决Layer1的效率低和交易费用高的问题。如图1所示，交易可以在Layer2上快速执行，Layer2按照一定的时机将最终状态同步回Layer1。Layer2用于专门提供高速的交易处理，而安全性和去中心化交由Layer1去保障。这样，减轻Layer1压力的同时，Layer2上执行交易的Gas费可以大幅下降，将Layer2上交易执行的最终状态（而不是Layer2上所有的状态）同步回Layer1所需的Gas费也可以大幅下降。在Layer1之上架构Layer 2的技术，实际上是将交易或合约的执行过程与最终状态的保存分离开，执行过程放到了Layer2，Layer1只需要保存最终的状态。这个设计中，需要保证Layer2上交易或合约的执行过程的正确性，并且写入Layer1的状态与Layer2上正确执行的结果是一致的。

Layer1的Layer2包括一种称为Rollup的机制。

Rollup的核心思路如图1所示，是在Layer1上保存能够验证交易过程的凭证，而将交易过程（计算过程）还有状态存储运行在Layer2中。所谓交易过程的凭证，包括一组交易执行前的状态（Pre-State）和该组交易执行后的状态（Post-State）以及该组交易，可以用于验证这组交易对应的状态转移是否正确，还可以用于还原出Layer2上所有交易的执行过程和所有账户的状态，从而消除Layer2上由于数据可用性带来的安全风险

Rollup的原理具体可以如图2和图3（图3中主要是下文中的OP-Rollup）所示。

创建智能合约的交易发送到Layer1上，经过Layer1的共识之后，Layer1上各节点可以执行这个交易，从而完成合约的部署。具体的，可以是由区块链节点的EVM/WASM来执行这个交易。EVM是一个图灵完备的虚拟机，这意味着可以通过它实现各种复杂的逻辑，这也是区块链2.0相对于区块链1.0的最大改进之一。用户在Layer1中发布和调用智能合约可以在EVM上运行。执行该部署智能合约的交易之后，Layer1上产生一个与该智能合约对应的合约账户，该合约账户具有链上的地址，并且包括余额balance、计数器nonce、合约代码的hash值Codehash，合约存储的根Storage_Root。通过Codehash和Storage_Root，该合约的代码和账户存储可以保存在该合约账户中。智能合约的行为由合约代码控制，而智能合约的账户存储则保存了合约的状态。换句话说，智能合约使得区块链上产生包含合约代码和账户存储（Storage）的虚拟账户。后续，Layer1上的节点可以接收调用部署的智能合约的交易请求，该交易请求可以包括调用的合约的地址、调用的合约中的函数和输入的参数。一般的，该交易请求经过共识后，区块链各个节点可以各自独立执行指定调用的智能合约。

Layer1中的Storage_root是一个MPT树根节点的哈希，这个MPT树对合约账户的状态的存储进行组织。MPT全称为Merkle Patricia Tree，是结合了Merkle Tree（默克尔树）和Patricia Tree（压缩前缀树，一种更节省空间的Trie树，字典树）的一种树形结构。Merkle Tree算法对每个交易都计算一个Hash值，然后两两连接再次计算Hash，一直到最顶层的Merkle根。Layer1中采用改进的MPT树，例如是16叉树的结构，通常也简称为MPT树。实际上，Layer1区块头中包括三颗MPT树的树根，分别是Transaction_Root（简写为Tx_Root）、State_Root、Receipt_Root。State_Root当前区块中所有账户的状态组成的MPT树的根的哈希值，即指向State_Root的为一颗MPT形式的状态树state trie。从这个MPT的根节点到叶子节点中每个节点的一部分值按照顺序串联起来可以构成账户地址并作为key，叶子节点中存储的账户信息为这个账户地址对应的value，构成了key-value键值对。前述的key可以是sha3(Address)，即账户地址的hash值（hash算法例如采用sha3算法），其存储的值value可以为rlp(Account)，即账户信息的rlp（Recursive Length Prefix，递归长度前缀）编码。账户信息是[nonce, balance, storageRoot, codeHash]构成的四元组。对于外部账户（Externally Owned Account，简称EOA）来说，一般只有nonce和balance两项，而storageRoot、codeHash字段默认存储空字符串/全0字符串。即外部账户不存储合约，也不存储合约执行后产生的状态变量。合约账户包括Nonce, Balance, Storage root, CodeHash。不论是外部账户还是合约账户，其账户信息一般都位于一个单独的叶子节点（Leaf Node）中。

对于state trie中的一个合约账户（Contract Account，简写为CA）来说，其storage_Root指向另一颗同为MPT形式的树，其中存储了合约执行涉及的状态变量（state variable）的数据。这个storage_Root指向的MPT形式的树为Storage Trie，即storage_Root中存储Storage Trie的根节点的hash值。一般的，这个Storage Trie树存储的也是key-value键值对。key表明状态变量的地址，其取值可以是合约中的状态变量声明的位置（从0开始计数的值）经过一定规则处理后得到的结果，例如是sha3

（状态变量声明的位置），或者是sha3（合约名称+状态变量声明的位置）。value用于存储状态变量的取值（例如是经RLP编码的值）。从根节点到叶子节点的路径上存储的一部分数据连起来构成key，叶子节点中存储value。State trie和其中的Storage trie中，除了叶子节点外的根节点、中间节点采用图3中云朵状来表示其中灵活和复杂多变的结构。

如图2所示，Rollup中可以在Layer1上部署Rollup Contract。Layer2中，可以接收发起的交易，如图中的Tx_21、Tx_22等。一个批次的交易称为一个交易组（即Batch），作为一个整体按序执行并更新状态。如图1和2中所示，Batch 1假设可以包括Tx_21、Tx_22、Tx_23这3笔交易，Batch 2假设可以包括Tx_24、Tx_25、Tx_26这3笔交易。Layer2中可以按照一种规则对这些交易进行排序，进而可以按照顺序执行排序后的这些交易，例如按照交易中的时间戳排序。一个Batch中的交易在按序执行后，产生的状态可以包括若干，例如Batch 1中的交易分别为：

Tx_21: Alice→Bob 20；表示Alice转给Bob 20个单位的资产；

Tx_22: Alice→Charlie 10；表示Alice转给Charlie 10个单位的资产；

15 Tx_23: Bob→Charlie 10；表示Bob转给Charlie 10个单位的资产；

Batch 1中的交易执行前，假设共有4个状态，这里称为Batch 1的初始状态，即图3中由Pre State Root锁定的状态值，分别是：

Alice: 50, Bob: 100, Charlie: 150, David: 200

Batch 1中的交易执行后，这4个状态更新为Batch 1的终局状态，即图3中由Post State Root锁定的状态值，分别是：

Alice: 20, Bob: 110, Charlie: 170, David: 200

Layer2中可以采用Merkle树来组织Batch 1中的交易按序执行前、后的状态。如图3所示，在Layer2中可以将一个Batch的交易及其相关的状态变化组织在一起，例如将Batch 1中的交易的hash值按照Merkle树组织后得到的Merkle根锁定在Batch的头

25 （Header）中，这个头类似Layer 1中的区块头，这个Merkle根即Batch的头中的Tx_Root。同样的，如前所述，可以将Batch 1中的交易在执行前的所有状态值按照Merkle树组织后得到的Merkle根锁定在Batch的头中，即Pre State Root；可以将Batch 1中的交易在按序执行后的所有状态值的hash按照Merkle树组织后得到的Merkle根锁定在Batch的头中，即Post State Root。对于Batch M来说，在其Header中还可以设置一个时间戳（Timestamp）用于记录该Batch生成的时间，当然还可以设置其它字段的信息。对于第M+1个Batch来说，其Header中的Pre State Root与第M个Batch的Header中的Post State Root相等，如图3中这两个根之间的连线所表示。如此Batch之间则形成了一个链式结构。

Layer2的Rollup机制，可以包括Optimistic-Rollup（乐观Rollup，简称OP-Rollup）、35 ZK-Rollup（零知识证明Rollup，ZK指的是zero knowledge，即零知识）、TEE-Rollup机制等。

OP-Rollup机制中，在Layer2里可以设置TxPool（交易池）、Relayer（中继器）以及Sequencer（定序器），在ZK-Rollup以及TEE-Rollup机制中，Layer2中还可以设置Prover（证明器）；其中对于TEE-Rollup机制而言，Prover（证明器）可以设置于

TEE中。

下面示例性描述OP-Rollup机制。如图3中所示，假设TxPool中收集到Tx21、Tx23、Tx25、Tx22、Tx26、Tx24这些交易。Sequencer按照这些交易中的时间戳对这些交易排序并打包，例如将Tx21、Tx22、Tx23打包到Batch M中，并将这些交易的hash值组织为Merkle后生成Merkle Root（即交易根），并将该Merkle Root的hash值填入生成的Batch M中Header内的Tx_Root中。需要说明的是，在计算Merkle树的树根时，一般对于小于 2^n 的情况，可以将最后一个交易赋值多份以用于补齐 2^n ，从而计算2叉的Merkle树的树根。如图3中，按照时间顺序排序为最末的Tx23重复一次后补齐了4个交易，即使交易的数量达到 2^2 ，进而可以计算得到2叉的Merkle树的树根。Sequencer按照交易中的时间戳对交易排序后将Tx21、Tx22、Tx23打包到Batch M+1。

在Batch M中的交易序列Tx21、Tx22、Tx23被执行前，所有的状态包括如前所述，Alice: 50, Bob: 100, Charlie: 150, David: 200。Sequencer可以将这些状态组织成Merkle树，并将该Merkle树根节点的hash值作为Batch M所对应的前状态根，存储于Batch M Header的Pre State Root字段中。进而Sequencer可以按照顺序执行Tx21、Tx22、Tx23这3笔交易；执行这3笔交易的结果生成了新的状态，如前所述，Alice: 20, Bob: 110, Charlie: 170, David: 200。Sequencer可以将这些更新后的状态组织成Merkle树，并将该Merkle树根节点的hash值作为Batch M所对应的后状态根，存储于Batch M Header的Post State Root字段中。

进而，Sequencer可以通过Relayer发送一笔调用Rollup Contract的交易1至Layer1，该交易1中包含Batch M Header中的部分或全部字段以及经过压缩的交易序列Tx21、Tx22、Tx23。Tx21、Tx22、Tx23经过压缩后，所占空间可以大大降低。调用的Rollup Contract合约中的接口，可以包括将压缩后的交易数据（Tx21、Tx22、Tx23）存储于较便宜（Gas费用较低）的位置calldata上。在Layer1智能合约中，calldata 是一种特殊的数据位置（data location），用于存储通过函数调用的输入数据。具体的，calldata数据一般直接存储在Layer1的交易数据中，其通常比其它数据位置（如memory和storage）更加节省gas。虽然本文中略去了压缩交易Batch所包括的交易序列的过程，然而可以理解的是下文中所述的任意交易Batch所包括的交易序列，既可以是未经压缩的交易序列，也可能是经压缩后的交易序列。

值得注意的是，在一笔Layer1交易中，签名部分占了整体的约一半的体积，其From字段可以是0字节，因为该From字段可以从签名中恢复出来。而在Rollup中，通常可以采用BLS或者其它方案进行聚合，平均每笔交易的签名被压缩到约0.5字节，相应的，Rollup交易中由于无法从这种聚合签名中恢复出交易的From字段，因此这里需要单独采用4字节来表示。

BLS 签名方案最初是由斯坦福大学教授 Dan Boneh 等人于 2001 年提出的一种签名方案，基于双线性映射构造。一般一个用户对自身账户的控制，体现在用自身私钥对一笔交易进行签名。单个签名如上表格中所示，可能达到68字节，占用较大体积。多个账户发起的多笔交易，则每个账户需要对自身的每笔交易分别签名。而签名聚合，可以把多个签名聚合成一个签名，因此大大节省了签名的字节空间。这种聚合可以是同一个消息由不同签名者签名的聚合，也可以是不同消息的签名聚合。公钥聚合，可

以将多个不同公钥聚合成一个总的公钥，由该总公钥即可对聚合签名完成验证，而无需在验证总签名时由每个公钥参与计算。通常可以采用Schnorr签名算法、BLS签名算法实现签名聚合。相对于Schnorr签名算法来说，BLS签名算法实现的聚合签名体积更小，约是 Schnorr 的二分之一。

- 5 上述压缩方案和采用BLS进行签名聚合，是理想中的设计方案。尽管如此，实际中的几种Rollup方案中，目前一笔压缩后的交易仍然占用40-50字节的空间，且其中尚不包括签名，也就是说还未采用BLS签名聚合。

- 接前文，Layer1中接收到该调用Rollup Contract的交易1后，经过共识，在一些节点上执行这个交易1，并将该交易1及其执行结果连同其它一些交易及执行结果一并生成一个区块，如图3中的Block N。所述其它一些交易，可以包括普通转账交易和涉及其它智能合约的交易。其中，普通转账交易和涉及其它智能合约的交易都可以涉及交易的执行。特别是其它智能合约的交易，可能涉及较为复杂的执行逻辑。而这里的交易1，调用Rollup Contract中存入calldata位置的函数，主要涉及存入指定位置，即不涉及复杂合约逻辑的执行。也就是说，Layer1里的Rollup Contract，主要作用是
- 10 将Layer2上发来的交易中指定的内容存入指定位置，此外不涉及复杂的执行逻辑。例如，作为一个例子，Tx21、Tx22、Tx23存入由Block N Header中Tx_Root锁定到交易树中的Tx_12中。此外，作为一种实现，交易1中的Batch M Header中的Pre State Root、Post State Root可以通过Rollup Contract分别存入Layer1上合约状态中的状态1和状态2中。如前所述，状态1和状态2包含在由Block N Header中State_Root锁定的state trie中，具体例如是storage trie中。为了加以区分，Layer1上合约状态中的状态1和状态2例如分别称为Last State root和Current State root。此外，为了简化，也可以在Layer1上合约状态中仅保存Current State root，而不保存Last State root，如下面的代码示例所示。在另一些实现中，Last State root和Current State root也可以与Layer2中的交易一样存入Layer1中的calldata内，这里不再赘述。对于前者，
- 15 例如是调用如下的Layer1中的commitBatch() 合约接口：function
CommitBatch(bytes[] calldata _transactions, bytes32 _preStateRoot, bytes32
_postStateRoot)，CommitBatch() 合约接口内部可以设置基本的校验逻辑，例如可以包括验证当前交易1中的Pre State Root是否与合约存储中的Current State Root相等，如果相等则将交易1中传入的Layer2上的交易存入calldata中，并将合约存储中的
- 20 Current State Root更新为Post State Root。

- 通过上述方式，OP-Rollup可以在Layer2中执行各个交易，并且可以在一个Batch中捆绑几十、数百甚至更多的Layer2交易，且只发布最低限度的信息至Layer1上。所谓最低限度的信息，包括如前所述的压缩后的Layer2上的交易和这批交易执行前后的状态根的hash值。如前所示，这里例如可以通过一个Layer1上的交易发送，如图3中的交易1。可见，这相当于一次性在Layer2上执行了一批交易，但是在Layer1上只执行了一个交易。同时OP-Rollup并不附带任何证明。这是出于假设提交的Layer2上的信息不存在欺诈或恶意行为，因此其是“Optimistic”（乐观）这样的名字。虽然存在这样的假设，为了预防和震慑，仍然会要求提交Layer2信息的交易者在OP-Rollup的Rollup Contract中质押一部分Layer1上的资产（典型的是链上原生的token）。对于提交到
- 30
- 35

Layer1上的Layer2信息，尽管其中的交易是经过压缩的，如前所述的是，仍然可以用于还原出Layer2上所有交易的执行过程和所有账户的状态，从而消除Layer2上由于数据可用性带来的安全风险。

OP-Rollup尽管是乐观的，但Layer1上仍然可以设置一个挑战期，任何人都可以在挑战期内使用欺诈证明提出挑战。典型的原因包括挑战者证明该Batch内的某个交易没有被正确的执行，即该交易的执行没有产生正确的状态。如果一个Batch内的某个交易被证明是无效的，则该Batch也是无效的，Layer1上的这个Rollup合约可以将其中的Layer2的Batch链进行回滚，该无效Batch及其后面的所有Batch都会变成孤块。一旦欺诈证明成功，保证金中的一部分会支付给挑战者，剩余部分则销毁。反之，如果直到挑战期结束都没有人提交欺诈证明，Rollup合约可以确定这个Batch。欺诈证明是Optimistic Rollup方案采用的一种数据有效性的验证方式。目前，欺诈证明可分为单轮交互型和多轮交互型两种。

一种单轮交互型欺诈证明的实现中，在Layer1上重放一遍有争议的交易，检测是否存在无效状态后进行提交。例如，Alice作为验证人将OP-Rollup压缩后的数据同步到Layer1，同时质押保证金。如果挑战者Bob对该数据存在争议，则需在窗口期（挑战期）内发起挑战且同样质押保证金。OP-Rollup的Rollup Contract将会在Layer1上重新计算该区块中交易来判定对错。错误方的保证金将被罚没，正确方将获得奖励。

这里需要说明的是，Layer1上的calldata中尽管有Layer2上所有的压缩后的交易，但在验证欺诈证明过程中，并不直接将calldata中Batch M中压缩后的交易直接取出执行，而是基于挑战者传入的Batch M中的所有交易原文模拟执行。这是因为，calldata中的压缩后Batch M的交易一般仅用于还原出Layer2上所有交易的执行过程和所有账户的状态，从而消除Layer2上由于数据可用性带来的安全风险；且calldata中的压缩后Batch M的交易，不带签名的情况下本身的真实性是有待确定的，而挑战者传入的交易是完整交易，可以验证交易的有效性；再者，还因为无法验证初始状态，合约存储中只是状态根而不是状态，除非从Batch 0开始重新顺序执行所有的交易直至Batch M-1执行后得到的全部状态，而这显然是不经济不高效的。

上述单轮交互型欺诈证明，增加了必须在链上发布的数据，且需要在Layer1上重新执行Batch M中的全部交易，并且重放这些交易还会产生巨大的燃料成本，因此，OP-Rollup正在转向多轮交互式证明，以更高的效率实现相同的目标。

多轮交互型欺诈证明中，当Bob对Sequencer同步的数据提出挑战后，Sequencer需将争议的Batch范围二等分为前、后两段范围并通过例如dispute()合约接口返回，然后由Bob进一步选择继续挑战的二分后的一个范围，再由Sequencer将争议范围二等分……如此循环，直至将争议范围缩减到一个具体的交易。这里Bob会发送挑战的交易的原文（包括每个交易的签名）及该交易执行前的全局状态，Sequencer会发送挑战的交易执行前的Pre State Root（这里也称为状态承诺）和执行后的Post State Root，交由Layer1的Rollup Contract合约进行计算和判定。具体的，例如Rollup Contract合约基于所挑战交易执行前的Pre State Root验证挑战交易执行前的全局状态，并在该挑战交易执行前的全局状态基础上执行挑战的交易，生成执行后的状态，在挑战交易执行后的全局状态的基础上生成Post State Root，与Sequencer发送的Post State

Root相比，如果不同，则说明挑战成功，反之挑战失败。在实际应用中，Arbitrum不是使用二分的方式，而是使用了K分，即每次将N个指令分为N/K组去寻找欺诈指令，效率更高。

因此，相比单轮交互型，多轮交互型能以更低的成本来解决争议，例如二等分协议可以减少发布在链上的数据量（无需为每笔交易发布状态提交），也更容易支持复杂的智能合约和处理更高要求的争议。不过，随着交互次数的增加，它的争议窗口期也会更长一些。

整体来说，由于OP-Rollup的设计中需要把与状态更新和验证相关的数据都放到Layer1上，因此OP-Rollup的拓展性和对吞吐量的提升程度最为有限，且需要7天甚至更长的等待期。这意味着，从OP-Rollup中提取资产会产生延迟，需要等待窗口期过后。

ZK-Rollup不同于Optimistic Rollup，因为ZK-Rollup使用零知识(zero knowledge)证明技术。ZK指的是在不披露必要信息的情况下向另一方证明某事（一项交易或状态）的能力。ZK-Rollup中，与OP-Rollup不同的是，Layer2中还会为交易Batch生成一个证明（包括proof）（例如采用ZK-SNARK 等的密码学证明算法），这个证明中的proof可以证明Post State Root是在Pre State Root基础上按序正确执行对应的交易Batch中的交易序列后得到的，只需要验证proof即可以确认这一点，而不需要重新执行Batch中的交易来验证。

生成证明的过程，可以由Sequencer将Layer2中一个Batch中所有交易（包括发送方、接收方、金额）以及这个Batch中的交易执行前后的Pre State Root、Post State Root和Proving Key等信息打包发送至Layer2上的一个证明器，由证明器将这些信息转换为一种特殊的形式，称为witness（见证者；选择其中的一部分作为公共输入，另一部分作为隐私输入）。进而，证明器采用其内布置的零知识证明的计算电路（一种特定的算法）对公共输入+隐私输入的有效性和正确性进行验证。如果验证正确，电路执行成功，否则电路执行失败。电路执行成功的情况下，证明器可以输出一个proof，该proof用以证明打包的数据是有效且正确的。而且，这个proof的体积较小，一般只有几十字节。上述生成证明的过程，类似于重新执行的过程，并且包含大量的电路计算，更为复杂，所以一般比较耗时。需要说明，由proof本身无法倒推出隐私输入，只有知道zk-SNARK的验证密钥的一方才能验证用对应创建密钥加密的proof。公共输入可以用来验证打包的数据的有效性和正确性，但无法用来获取打包的数据的具体内容。

采用零知识证明，证明器可以通过Relayer将较小的proof发布到Layer1上，并可以由Rollup Contract快速验证。proof体积较小，这进一步降低了Layer1上Gas的消耗。而且，在ZK-Rollup中这个验证过程通常由智能合约自动完成，且不用像OP-Roll那样再设置窗口期，提取资产也会变得迅速，这是OP-Rollup和ZK-Rollup之间的主要特性差异之一。

具体的验证过程，首先，将公共输入（可以从Block N中提交的交易1计算得到，例如是在交易1提交至Layer1的Rollup Contract时由Rollup Contract中的自动执行计算逻辑计算得到）、proof和验证密钥（验证密钥可以通过Layer1上的治理合约保存并提供至Rollup Contract）输入到智能合约中的验证逻辑（例如包括ZK-SNARK的验证算法），由验证逻辑使用验证算法检查proof是否有效。如果proof有效，那么公共输入

对应的打包数据就被认为是有效且正确的。如果验证成功，验证算法会输出一个确认信息，证明公共输入对应的打包数据是有效且正确的；否则，验证算法会输出一个错误信息，标示数据的有效性或错误性。

然而，ZK-SNARK这类的零知识证明在生成proof过程中的计算量非常大，也意味着生成proof将比较耗时。假设如图4中所示，通过交易1将经过压缩的Tx21、Tx22、Tx23存入Layer1中calldata中，并将Post State Root存入Layer1上合约状态中的Current State Root。这个交易1如图4位于Layer1的Block N中。则Layer2中生成对应proof所需要的复杂计算量带来的延迟，可能在一段时间通过交易2将Batch M的proof传入Layer1上的Block N+412中。从Block N到Block N+412，显然也带来了一定的延迟，这个延迟可能接近小时级。

针对计算复杂、生成证明所需时间久这个问题，对应提出了TEE-Roll机制。TEE-Rollup机制相对于ZK-Rollup的不同之处主要在于：Prover采用TEE（Trusted Execution Environment，可信执行环境）实现，而且Prover可以在排序Sequencer提供的某个交易Batch的交易根、前状态根和后状态根等信息通过验证后，采用TEE中的私钥对该交易batch的相关信息签名，进而将签名作为证明并通过Relayer将其发布到Layer1上，并可以由Rollup Contract快速验证。

Layer2提交的任意交易Batch所对应的挑战期结束后，或者说任意交易Batch所对应的证明在Layer2的Rollup Contract中通过验证后，该任意Batch及其对应的合约调用交易（例如前述用于调用Rollup Contract中Commit Batch（）合约接口的交易1）即进入不可更改（Finality）状态。

随着Rollup技术的兴起，以Rollup技术为核心的模块化应用链（AppChain）设计越来越丰富。AppChain是指为了实现独立业务目的或应用而设计的区块链系统，例如游戏或DeFi应用。AppChain可以基于公链/主链运行，但其又有独立的共识机制、区块生成和交易验证方式。在分布式应用（Decentralized Application, DApp）的开发中，AppChain可以解决公链的扩容问题，提高交易处理的速度和效率。同时，AppChain仍然继承公链的优点，包括其安全性、去中心化等特性。例如，Layer1中的Plasma和Cosmos中的Zone都是AppChain的例子。它们具有独立运行的能力，并且可以与主链进行交互，通过主链来实现安全性和去中心化。在AppChain中，可以根据具体的应用需求来定制区块生成的速度、交易费用、智能合约的功能等。这使得AppChain比公链更灵活，更适应特定的业务需求。

任意APPChain可以注册在位于Layer1的区块链系统中，自身位于Layer2中，Layer1中的区块链系统具有该APPChain的主权，即Layer1中的区块链系统为该任意APPChain对应的上一层区块链系统；该任意APPChain可以注册在位于Layer2的区块链系统中，自身则位于Layer3中，位于Layer2中的相应区块链系统则具有该APPChain的主权，即位于Layer2中的相应区块链系统为该任意APPChain对应的上一层区块链系统；与之相应的是，如果任意APPChain位于Layer2中，则其可能具有某个位于Layer3中的其它APPChain主权，即其自身可能作为某个位于Layer3中区块链系统所对应的上一层区块链系统。可以理解的是，包括多个APPChain的多层卷叠网络甚至可能继续扩展到Layer4、layer5甚至更多层。

示例性的，参见图 5，采用 Rollup 技术的多层卷叠网络中，可以包括位于 Layer1 中的区块链系统 B11，位于 Layer2 的区块链系统 B21 和区块链系统 B22，以及位于 Layer3 的区块链系统 B31。其中 B11 具有 B21 和 B22 的主权，即 B11 为 B21 和 B22 对应的上一层区块链系统；B21 具有 B31 的主权，即 B21 为 B31 对应的上一层区块链系统。与之相应的是，B21 与 B11 之间、B22 与 B11 之间、B31 与 B21 之间，可以采用前述各种 Rollup 机制实现卷叠。

如果位于 Layer2、Layer3 甚至更低层的某个区块链系统期望进行主权迁移，即某个区块链系统可能期望更换其对应的上一层区块链系统，例如任意区块链系统 B31 期望将其对应的上一层区块链系统由 B21 更换为 B22，则通常需要区块链系统 B31、B21 和 B22 的管理成员进行线下沟通，进而进行相应的数据传递来实现主权迁移，效率相对较差且安全性较低。

本说明书实施例中至少提供了一种区块链系统的管理方法和区块链系统。涉及第一区块链系统（即位于 Layer1 的区块链系统）和 N 个第二区块链系统（即位于 Layer2、Layer3 甚至更低层的区块链系统）；该第一区块链系统中部署中有第一卷叠合约，该第一卷叠合约中包括第一方法函数，该第一卷叠合约的合约状态中存储有 N 个第二区块链系统各自的注册信息。当任意第 i 个第二区块链系统的管理成员期望将某个第三区块链系统作为该第 i 个第二区块链系统对应的上一层区块链系统时，可以向第一区块链系统发送用于调用该第一方法函数的第一交易；该第一区块链系统即可根据第一交易执行第一方法函数，实现：生成对应的主权迁移事件，其中包括第 i 个第二区块链系统和第三区块链系统的标识信息；与之相应的是，该第三区块链系统即可根据该主权迁移事件，利用第 i 个第二区块链系统的注册信息，在该第三区块链系统中设置与第 i 个第二区块链系统对应的第二卷叠合约，成为第 i 个第二区块链系统对应的上一层区块链系统。如此，在位于 Layer1 的第一区块链系统所部署第一卷叠合约的支持下，位于 Layer2、Layer3 甚至更低层的区块链系统能够快速完成主权迁移，即快速更换其对应的上一层区块链系统，无需各区块链系统的管理成员线下沟通，可同时兼顾效率和安全性。

本说明书实施例中为了方便描述，将利用区块链系统 B11 表征位于 Layer1 中的第一区块链系统，利用智能合约 Rollup Contract C1 表征部署在第一区块链系统中的第一卷叠合约，以及将主要以 N 个第二区块链系统具体包括区块链系统 B21、B22 和 B31 为例进行示例性描述。

首先，Rollup Contract C1 中可以包括用于支持 N 个区块链系统均向区块链系统 B11 注册的方法函数，例如合约接口 Identity register()。区块链系统 B31 的管理成员例如可以向区块链系统 B11 发送用于调用 Identity register() 的注册交易，区块链系统 B11 可以根据该注册交易执行 Identity register()，实现在 Rollup Contract C1 的合约状态中存储并生效区块链系统 B11 的注册信息，区块链系统 B11 的注册信息可以来自该注册交易，其可以包括但不限于区块链系统 B11 的如下各项信息中的部分或全部：标识信息、创世块、管理成员地址集合、排序器地址集合、共识算法、交易 batch 的存储方式、证明类型/Rollup 机制以及其对应的验证算法。

类似的，其它第二区块链系统例如 B21、B22 均可注册到区块链系统 B11 中，使得 Rollup Contract C1 的合约状态中存储并生效 B21、B22、B31 等各个第二区块链系统

的注册信息。

示例性的，Rollup Contract C1 的合约状态中，可以存储 B21、B22、B31 等区块链系统的注册信息各自对应的状态指示，该状态信息用于指示对应的注册信息是否生效；例如，状态指示的取值为预定数值 1，则表征其对应的注册信息处于生效状态，该注册信息所属区块链系统的主权仍然在区块链系统 B11 中，即表征该注册信息所属区块链系统对应的上一层区块链系统为区块链系统 B11；与之相应的，状态指示的取值为预定数值 0，则表征其对应的注册信息处于无效状态，该注册信息所属区块链系统的主权已经不在区块链系统 B11 中，即表征该注册信息所属区块链系统对应的上一层区块链系统已经不再是区块链系统 B11。

区块链系统 B21、B22 和 B31 完成注册到位于 Layer1 中的区块链系统 B11 时，B21、B22 和 B31 均位于 Layer2 中。即 Rollup Contract C1 的合约状态中所存储 B21、B22、B31 的注册信息均处于已生效状态，例如 B21、B22、B31 的注册信息各自对应的状态指示的取值均为 1。

任意区块链系统例如区块链系统 B31 完成注册到位于 Layer1 中的区块链系统 B11 之后，可能按照其管理成员的要求，进行如下迁移形式 1~迁移形式 3 等多种方式的主权迁移。

迁移形式 1，区块链系统 B31 的主权由较高 Layer 迁移至较低 Layer。例如请参照图 6 所示，B31 的主权可能由位于 Layer1 的 B11 迁移至位于 Layer2 的 B21，自身进入 Layer3 中。

迁移形式 2，区块链系统 B31 的主权可能在相同 Layer 中进行迁移。例如请参照图 7 所述，B31 的主权可能由位于 Layer2 的 B21 迁移至位于 Layer2 的 B22，自身仍然在 Layer3 中。

迁移形式 3，区块链系统 B31 的主权由较低的 Layer 迁移至较高 Layer。例如请参照图 8 所示，B31 的主权可能由位于 Layer2 的 B22 迁移至位于 Layer1 的 B11，自身进入 Layer2 中。

与区块链系统 B11 具有或终止其对其它第二区块链系统主权的解决方案类似，如果一个区块链系统例如区块链系统 B21 具有另外一个区块链系统例如区块链系统 B31 的主权，即区块链系统 B21 为区块链系统 B31 对应的上一层区块链系统，则区块链系统 B21 中可能基于区块链系统 B31 的注册信息，部署区块链系统 B31 对应的卷叠合约 Rollup Contract C2；可以通过在 Rollup Contract C2 的合约状态中存储并生效区块链系统 B31 的注册信息，例如通过设置对应的状态指示的取值，使得区块链系统 B21 具有或者终止其对区块链系统 B31 的主权。

基于前述 Layer1、Layer2 和 Layer3，首先介绍一种实现多层网络卷叠的方法。

图 9 为本说明书实施例中提供的一种实现多层网络卷叠的方法。该方法描述了位于 Layer3 的任意第 i 个第二区块链系统（例如区块链系统 B31）、位于 Layer2 的另一个第二区块链系统（例如区块链系统 B21）和位于 Layer1 的区块链系统 B11 实现多层网络卷叠的过程；即，以 B11 为 B21 对应的上一层区块链系统、B21 为 B31 对应的上一层区块链系统进行描述。

部署于区块链系统 B11 的卷叠合约 Rollup Contract C1 的合约状态中，除包括 N

个第二区块链系统的注册信息外，还可以包括 N 个第二区块链系统的全局管理信息，例如还包括以区块链系统 B21、B22 和 B31 各自的标识信息为 key 存储的全局管理信息。其中，Rollup Contract C1 中可以包括允许区块链系统 B21 的 Relayer/Sequencer 调用的合约接口 Commit Batch()。

- 5 位于 Layer2 的区块链系统 B21 中可以包括基于区块链系统 B31 的注册信息设置的卷叠合约 Rollup Contract C2，该 Rollup Contract C2 中例如包括允许区块链系统 B31 的 Relayer/Sequencer 调用的合约接口 Commit Batch()，该 Rollup Contract C2 的合约状态中还可以包括以区块链系统 B31 的标识信息为 key 存储的管理信息 G31。

参照图 9 所示，该方法可以包括但不限于如下步骤 S901~步骤 S907。

- 10 步骤 S901，区块链系统 B21 从区块链系统 B31 接收交易 Tx1，交易 Tx1 用于调用 Rollup Contract C2 中的 Commit Batch()，交易 Tx1 中包括区块链系统 B31 提交的交易 batch b3 所包括的交易序列和交易 batch b3 对应的状态数据 S1。

- 区块链系统 B31 的 Sequencer 可以从区块链系统 B31 的 TxPool 中，获得区块链系统 B31 中的各个外部账户所发起的一个或多个用户交易，组成属于某个交易 batch（例如交易 batch b3）的交易序列；接着，基于该交易序列被执行前的区块链系统 B31 的世界状态，按顺序执行该交易序列中的多个用户交易；进而，可以对应的获取该交易 batch b31 的 batch 序号、交易根、前状态根以及后状态根等信息，并由 Sequencer/Relayer 向区块链系统 B21 发起交易 Tx1。
- 15

- 交易 batch b3 所对应的状态数据 S1 可以包括：交易 batch b3 所包括交易序列被执行前，区块链系统 B31 的世界状态所对应的状态根，即交易 batch b3 所对应的前状态根；以及，交易 batch b3 所包括交易序列被执行后，区块链系统 B31 的世界状态所对应的状态根，即交易 batch b3 所对应的后状态根。需要说明的是，状态数据 S1 也可以仅包括前述的后状态根。
- 20

在一些实施例中，交易 Tx1 中还可以包括交易 batch b3 的 batch 序号。

- 25 步骤 S903，区块链系统 B21 根据交易 Tx1 执行 Rollup Contract C2 中的 Commit Batch()，实现：存储 batch b3 所包括的交易序列，并在区块链系统 B31 的管理信息 G31 中，将区块链系统 B31 的第二状态数据更新为状态数据 S1。

- 交易 Tx1 中还包括交易 batch b3 的 batch 序号的情况下，区块链系统 B21 根据交易 Tx1 执行 Rollup Contract C2 中的 Commit Batch() 时，还可以实现：在区块链系统 B31 的管理信息 G31 中，根据 batch b3 的 batch 序号新增位置索引，该位置索引用于指示 batch b3 所包括的交易序列在区块链系统 B21 中存储位置。
- 30

- 参照图 10 所示，Rollup Contract C2 对应的合约账户 (CA_2) 可以包括 Nonce, Balance, Storage root, CodeHash；其中 storage root 指向另一颗同为 MPT 形式的树，这个 Storage Trie 树存储的是 key-value 对，可以区块链系统 B31 的标识信息为 key 存储区块链系统 B31 的管理信息 G31，管理信息 G31 中包括区块链系统 B31 的第二状态数据以及一条或多条位置索引。其中，第二状态数据中例如可以通过前述示例的 Last State Root 和 Current State Root 两个字段分别存储 batch b3 所对应的前状态根和后状态根。其中为了使不同区块链系统的管理信息相互独立的存储于 Rollup Contract C2 的合约状态中，可以将管理信息 G31 所包括的第二状态数据和全部位置
- 35

索引，整体作为以区块链系统 B31 的标识信息为 key 的 value 进行存储。

在一种可能的实施方式中，batch b3 所包括的交易序列可以直接存储在交易 Tx1 中，batch b3 对应的位置索引可以包括，batch b3 的 batch 序号以及交易 Tx1 在区块链系统 B11 中所属区块的区块号、交易 Tx1 在所属区块中的交易序号。在另一种可能的实施方式中，batch b3 所包括的交易序列可以存储在区块链系统 B31 的管理信息 G31，位置索引可以包括 batch b3 的 batch 序号以及 Batch b3 所包括的交易序列在管理信息 G31 中的位置信息。

步骤 S905，区块链系统 B21 向区块链系统 B11 发送交易 Tx2，交易 Tx2 用于调用 Rollup Contract C1 中的 Commit Batch()，交易 Tx2 中包括区块链系统 B21 提交的交易 batch b2 所包括的交易序列、交易 batch b2 所对应的状态数据 S2，以及 Rollup Contract C2 合约状态中通过管理信息 G31 存储的区块链系统 B31 的标识信息和第二状态数据。

Rollup Contract C1 中的合约接口 Commit Batch() 即 Rollup Contract C1 所包括的第三方法函数，交易 Tx2 即用于调用第三方法函数的第三交易，交易 batch b2 即第二交易 batch，交易 batch b2 所对应的状态数据 S2 可以包括第二交易 batch 所对应的前状态根和后状态根。

区块链系统 B21 的 Sequencer 可以首先从区块链系统 B21 的 TxPool 中，获得区块链系统 B21 中的各个外部账户所发起的多个用户交易，组成属于某个交易 batch（例如交易 batch b2）的交易序列；接着，基于该交易序列被执行前的区块链系统 B21 的世界状态，按顺序执行该交易序列中的多个用户交易；进而，可以对应的获取该交易 batch b2 的 batch 序号、交易根、前状态根以及后状态根等信息，并由 Sequencer/Relayer 向区块链系统 B21 发起交易 Tx2。

需要特别说明的是，当区块链系统 B21 是区块链系统 B31 对应的上一层区块链系统 B31 的情况下，交易 Tx2 中才会包括 Rollup Contract C2 的合约状态中通过管理信息 G31 存储的区块链系统 B31 的标识信息和第二状态数据。当区块链系统 B21 并非区块链系统 B31 对应的上一层区块链系统 B31 的情况下，即使区块链系统 B21 中包括区块链系统 B31 的管理信息 G31，交易 Tx2 中也并不会包括通过管理信息 G31 存储的区块链系统 B31 的标识信息和第二状态数据。

典型示例中，区块链系统 B21 的 Sequencer 可以查询 Rollup Contract C2，获知哪些区块链系统的注册信息已在区块链系统 B21 中生效，即获知当前将区块链系统作为上一层区块链系统的其它区块链系统例如区块链系统 B31；进而，区块链系统 B21 的 Sequencer 发起的用于调用 Rollup Contract C1 中的合约接口 Commit Batch() 的交易 Tx2 中，即可从 Rollup Contract C2 中查询到前述其它区块链系统例如区块链系统 B31 的管理信息 G31，并将管理信息 G31 中存储的区块链系统 B31 的标识信息和第二状态数据，加入到交易 Tx2 中。

在一些实施例中，交易 Tx2 中还可以包括交易 batch b2 的 batch 序号。

步骤 S907，区块链系统 B11 根据交易 Tx2 执行 Rollup Contract C1 中的合约接口 Commit Batch()，实现：存储交易 batch b2 所包括的交易序列，并在区块链系统 B21 的全局管理信息 G22 中，将区块链系统 B21 的第三状态数据更新为状态数据 S2，

以及在区块链系统 B31 的全局管理信息 GZ31 中,根据区块链系统 B31 的标识信息和第二状态数据,将区块链系统 B31 的第四状态数据更新为第二状态数据。

交易 Tx2 中还包括 batch b2 的 batch 序号的情况下,区块链系统 B11 根据交易 Tx2 执行 Rollup Contract C1 中的合约接口 Commit Batch(),还可以实现:在区块链系统 B21 的全局管理信息 GZ2 中新增位置索引,该位置索引用于指示 batch b2 在区块链系统 B11 中的存储位置。

参照图 10 所示, Rollup Contract C1 对应的合约账户 (CA_1) 可以包括 Nonce, Balance, Storage root, CodeHash; 其中 storage root 指向另一颗同为 MPT 形式的树, 这个 Storage Trie 树存储的是 key-value 对, 可以区块链系统 B21、区块链系统 B31 的标识信息为 key, 存储区块链系统 B21 的全局管理信息 GZ21 和区块链系统 B31 的全局管理信息 GZ31。全局管理信息 G21 中包括区块链系统 B21 的第三状态数据以及一条或多条位置索引; 其中第三状态数据中例如可以通过前述示例的 Last State Root 和 Current State Root 两个字段分别存储 batch b2 所对应的前状态根和后状态根。类似的, 第四状态数据中例如可以通过前述示例的 Last State Root 和 Current State Root 两个字段对应存储第二状态数据。其中为了使不同区块链系统的全局管理信息相互独立的存储于 Rollup Contract C1 的合约状态中, 可以将全局管理信息 GZ31 所包括的全部主权迁移记录、全部位置索引进而第四状态数据, 整体作为以区块链系统 B31 的标识信息为 key 的 value 进行存储; 类似的, 将全局管理信息 GZ21 所包括的全部位置索引进而第二状态数据, 整体作为以区块链系统 B31 的标识信息为 key 的 value 进行存储。

如果区块链系统 B31 在某个时间段内所对应的上一层区块链系统包括区块链系统 B11, 则其全局管理信息 GZ31 中还可能包括区块链系统 B31 提交部分交易 batch 对应的位置索引。

当任意区块链系统在某个时间段内以某个指定区块链系统作为其对应的上一层区块链系统时, 指定区块链系统中可以通过该任意区块链系统的管理信息或全局管理信息, 维护该任意区块链系统在该时间段内提交的各个交易 batch 所分别对应的位置索引, 从而可以支持在该任意区块链系统发生故障时, 可以通过相应的位置索引查询到该任意区块链系统已提交的交易 batch 所包括的交易序列在指定区块链系统中的存储位置, 进而可以利用已提交的交易 batch 所包括的交易序列, 恢复该任意区块链系统或者该任意区块链系统中的部分数据。

需要理解, 前述图 9 所示方法实施例中的方法步骤, 仅仅描述了包括 Layer1、Layer2 和 Layer3 的多层卷叠网络中, 通过证明器/中继器提交交易 batch 的过程, 实际上, 位于 Layer2 中的区块链系统 B21、位于 Layer3 中的区块链系统 B31, 还可能通过其各自对应的证明器, 向其各自对应的上一层区块链系统提交相关交易 batch 的证明, 相关交易 batch 的证明通过验证后, 相关交易 batch 以及其对应的合约调用交易即可进入不可更改 (Finality) 状态。

基于前述 Layer1、Layer2 和 Layer3, 接着介绍一种区块链系统的管理方法。

图 11 为本说明书实施例中提供的一种管理方法的流程图。该方法涉及第一区块链系统 (记为区块链系统 B11) 和 N 个第二区块链系统, 该方法示例性描述了 N 个第二

区块链系统中的任意第 i 个第二区块链系统（记为区块链系统 B31）进行主权迁移的过程。

参照图 11 所示，该方法可以包括但不限于如下步骤 S1101~步骤 S1109 中的部分或全部。

- 5 可选地，区块链系统 B11 所部署 Rollup Contract C1 中包括第二方法函数，例如利用合约接口 Termination sovereignty() 表征该第二方法函数。如果区块链系统 B31 当前对应的上一层区块链系统是区块链系统 B11 或者其余 $N-1$ 个第二区块链系统中的第四区块链系统，并且区块链系统 B31 的管理成员，期望该第四区块链系统在区块链系统 B31 提交的第一交易 batch 进入不可更改状态后，第四区块链系统不再作为区块链系统 B31 对应的上一层区块链系统，则区块链系统 B31 的管理成员可以向区块链系统 B11 对应的发起第二交易（记为交易 Tx3）。

- 10 与之相应的，区块链系统 B11 可以对应执行步骤 S1101，接收交易 Tx3，交易 Tx3 用于调用 Rollup Contract C1 中的 Termination sovereignty()，交易 Tx3 请求第四区块链系统在区块链系统 B31 提交的第一交易 batch 进入不可更改状态后，第四区块链系统不再作为区块链系统 B31 对应的上一层区块链系统。

下文中将主要以 N 个第二区块链系统包括区块链系统 B21、B22 和 B31 为例进行示例性描述。在此基础上，前述的第四区块链系统则可以是区块链系统 B11、B21 或 B22。

交易 Tx3 中例如可以包括区块链系统 B31 所提交第一交易 batch 的 Batch 序号，以及区块链系统 B31 和第四区块链系统各自的标识信息。

- 20 步骤 S1103，区块链系统 B11 根据交易 Tx3 执行 Rollup Contract C1 中的 Termination sovereignty()，实现：生成主权终止事件，主权终止事件中包括第一交易 batch 的 batch 序号、区块链系统 B31 和第四区块链系统的标识信息。

- 25 区块链系统 B11 根据交易 Tx3 执行 Termination sovereignty() 的结果或相关信息可以记录在交易 Tx3 所对应的收据 (receipt) 中。具体的，合约执行结果/相关信息可以表现为收据中的事件 (event)，事件中通常可以包括 topic 字段和 data 字段，其中 topic 字段用于记录事件的主题、data 字段用于记录相应的信息。事件的格式可以在合约中指定，通过内置的 SDK，客户端或区块链节点可以监听特定 topic 的事件，并在监听到特定 topic 事件的情况下，拉取相应 data 中的内容，以及可以监听到特定 topic 或对应 data 中的某些内容后执行预设的处理。

- 30 通过这种事件机制，区块链系统 B11 可以将执行结果存放到对应某个 topic 的事件中，从而第四区块链系统可以通过其 Sequencer/Relayer 自身内置的区块链客户端监听该 topic，获得相应的主权终止事件，并通过主权终止事件获知其需要在区块链系统 B31 提交的第一交易 batch 进入不可更改状态后，不再作为区块链系统 B31 所对应的上一层区块链系统。

- 35 与之相应的，在步骤 S1105，第四区块链系统根据主权终止事件，在确认第一交易 batch 已进入不可更改状态后，在其部署的第三卷叠合约中无效区块链系统 B31 的注册信息。

第四区块链系统例如是区块链系统 B11，此种情况下该第三卷叠合约可以是 Rollup Contract C1。因区块链系统 B11 当前仍具有区块链系统 B31 的主权，因此其

自身或者 B11 的管理成员能够基于第一交易 batch 的 batch 序号感知第一交易 batch 是否已进入不可更改状态。当第一交易 batch 已进入不可更改状态后，区块链系统 11 中的相关基础设施可以自动发起、或者由区块链系统 B11 的管理成员发起，发起用于调用 Rollup Contract C1 中特定方法函数例如合约接口 Indicate change() 的目标交易，区块链系统 B11 根据目标交易执行合约接口 Indicate change()，实现终止区块链系统 B11 对区块链系统 B31 的主权，例如在 Rollup Contract C1 的合约状态中，将区块链系统 B31 的注册信息所对应的状态指示由 1 置为 0。

前文以第四区块链系统是区块链系统 B11 为例进行示例性描述，然而可以理解的是其它区块链系统例如区块链系统 B21、B22 作为第四区块链系统时，可以通过相似的策略发起用于调用其自身所部署第三卷叠合约的目标交易，进而通过相似的方法实现终止对 B31 的主权。

第四区块链系统执行目标交易时，不仅实现终止其对区块链系统 B31 的主权，还可以对应的生成主权响应事件，该主权响应事件中例如包括相应的 topic 以及区块链系统 B31 的标识信息，从而使得区块链系统 B31 以及其管理成员获知第一交易 batch 已进入不可更改状态。

区块链系统 B31 还可以通过其它方式获知第一交易 batch 已进入不可更改状态，例如当第四区块链系统是区块链系统 B11 的情况下，区块链系统 B31 的证明器通过相关交易向区块链系统 B11 提交第一交易 batch 所对应的证明并通过验证后，区块链系统 B11 可能对该相关交易生成相应的证明通知事件，从而使区块链系统 B31 以及其管理成员能够通过该相关交易所对应的证明通知事件，获知第一交易 batch 已进入不可更改状态。

当区块链系统 B31 的管理成员感知到第一交易 batch 已进入不可更改状态，并且期望将区块链系统 B31 的主权迁移至第三区块链系统时，可以对应的发起第一交易（记为交易 Tx4）。

与之相应的，区块链系统 B11 可以接着执行步骤 S1107，接收用于调用 Rollup Contract C1 中 Sovereign transfer () 的交易 Tx4，交易 Tx4 请求将第三区块链系统作为区块链系统 31 对应的上一层区块链系统，第三区块链系统是区块链系统 B11 和其余 N-1 个第二区块链系统中的一个。可以理解，Sovereign transfer () 表征 Rollup Contract C1 所包括的第一方法函数。

交易 Tx4 中例如可以包括第三区块链系统的标识信息，此外还可以包括第一交易 batch 的 batch 序号、区块链系统 B31 的标识信息和第四区块链系统的标识信息等。

步骤 S1109，区块链系统 B11 根据交易 Tx4 执行 Rollup Contract C1 中的 Sovereign transfer ()，实现：生成主权迁移事件，其中包括区块链系统 B31 和第三区块链系统的标识信息。

结合前文的示例，N 个第二区块链系统包括区块链系统 B21、B22 和 B31，前述的第四区块链系统则可以是区块链系统 B11、B21 或 B22。则，前述的第三区块链系统可以是区块链系统 B11、B21 或 B22，其中第三区块链系统不同于第四区块链系统。

主权迁移事件中还可以包括区块链系统 B31 的注册信息。

区块链系统 B11 根据交易 Tx4 执行 Rollup Contract C1 中的 Sovereign transfer

()时, 参照图 10, 还可以实现在区块链系统 B31 的全局管理信息 GZ31 中, 新增一条主权迁移记录, 其中包括第三区块链系统的标识信息和第一交易 batch 的下一个交易 batch 所对应的 batch 序号。

- 步骤 S1111, 第三区块链系统根据主权迁移事件, 利用区块链系统 31 的注册信息, 在第三区块链系统中设置与区块链系统 31 对应的第二卷叠合约。

第三区块链系统可以通过前述事件监听机制获得主权迁移事件, 即第三区块链系统的 Sequencer/Relayer 以及管理成员可能感知主权迁移事件。当第三区块链系统中未部署卷叠合约的情况下, 第三区块链系统的 Sequencer/Relayer 可以自动发起合约部署交易, 或者由第三区块链系统的管理成员发起合约部署交易, 第三区块链系统可以通过执行该合约部署交易完成部署第二卷叠合约, 并通过相应的接口调用交易调用合约接口例如 Identity register(), 通过对接口调用交易的执行完成在第二卷叠合约的合约状态中存储并生效区块链系统 31 的注册信息, 从而使第三区块链系统成为区块链系统 B31 所对应的上一层区块链系统。可以理解的是, 当第三区块链系统中已部署第二卷叠合约的情况下, 第三区块链系统的 Sequencer/Relayer 自动发起前述接口调用交易, 或者由第三区块链系统的管理成员发起接口调用交易, 第三区块链系统可以通过执行该接口调用交易完成在第二卷叠合约的合约状态中存储并生效区块链系统 31 的注册信息, 成为区块链系统 B31 所对应的上一层区块链系统。

第三区块链系统可以从主权迁移事件中获得区块链系统 B31 的注册信息, 也可以通过其它方式从 Rollup Contract C1 中获取区块链系统 B31 的注册信息。以便后续过程中, Rollup Contract C1 能够按照区块链系统 B31 的注册信息, 处理由区块链系统 B31 的 Sequencer/Relayer 和/或 Prover/Relayer 所发起的用于调用 Rollup Contract C1 的合约调用交易。

步骤 S1113, 第三区块链系统在第二卷叠合约的合约状态中, 存储区块链系统 B31 最新提交并已进入不可更改状态的第一交易 batch 所对应的第一状态数据。

参照前文如图 9 所示实施例记载的相关内容, 区块链系统 B31 提交交易 batch b3 后, 交易 batch b3 所对应的状态数据 S1 会作为第二状态数据, 存储于上一层区块链系统 B21 所部署智能合约 Rollup Contract C2 中, 该第二状态数据还会通过区块链系统 B21 向区块链系统 B11 发起的交易 Tx2 传递到区块链系统 B21, 并作为第四状态数据存储于区块链系统 B21。

与之相应的, 当第一交易 batch 已进入不可更改状态后, 区块链系统 B21 中所存储区块链系统 B31 的第四状态数据, 也即是第一交易 batch 所对应的第一状态数据。由此, 主权迁移事件中还可以包括该区块链系统 B31 的第四状态数据, 也即包括该第一状态数据。

第三区块链系统还可以通过其它方式获得第一交易 batch 所对应的第一状态数据。示例性的, 第三区块链系统可以从区块链系统 B31 的注册信息中获取创世块, 以及从曾经作为区块链系统所对应的上一层区块链系统的各个区块链系统中, 获取区块链系统 B31 已提交并进入不可更改状态的全部交易 batch 各自包括的交易序列和证明, 通过重放创世块和全部交易 batch 各自包括的交易序列, 获得第一交易 batch 所对应的第一状态数据。再如, 通过去中心化的基础设施例如中继器, 从第四区块链系统中获

得第一交易 batch 所对应的第一状态数据。

区块链系统 B31 的管理成员，还可能期望复用区块链系统所对应上一层区块链系统的基础设施，例如排序器、证明器以及中继器等各项基础设施中的一项或多项。与之相应的，在一些实施例

5 中，Rollup Contract C1 中还可以包括第四方法函数，例如利用合约接口 Facility management () 表征第四方法函数。在此基础上，区块链系统 B31 的管理成员可以向区块链系统 B11 发起用于调用 Facility management () 的第四交易（记为交易 Tx5），交易 Tx5 请求区块链系统 B31 复用第三区块链系统中用于执行预定事务的基础设施，例如交易 Tx5 中可以包括区块链系统 B31 的标识信息、第三区块链系统的标识信息和指示信息，该指示信息用于指示需要复用的基础设施。接着，
10 区块链系统 B11 可以根据交易 Tx5 执行 Rollup Contract C1 中的 Facility management ()，实现：生成设施复用事件，其中包括区块链系统 B31 和第三区块链系统的标识信息，此外还可以包括前述的指示信息。第三区块链系统可以通过事件监听机制获得前述设施复用事件，进而，可以根据设施复用事件，通过其自身所包括的并由相应指示信息所指示的基础设施，执行区块链系统 B31 中所需执行的预定事务。示例性的，例如
15 可以由第三区块链系统中的排序器、证明器执行需要在区块链系统 B31 中执行的部分事务。

与前述方法实施例基于相同的构思，本说明书实施例中还提供了一种第一区块链系统中的区块链节点 120，所述第一区块链系统中部署有第一卷叠合约，所述第一卷叠合约中包括第一方法函数，所述第一卷叠合约的合约状态中存储有 N 个第二区块链系统的注册信息。参见图 12 所示，所述区块链节点 120 包括：交易接收单元 1201，
20 配置为接收用于调用所述第一方法函数的第一交易，所述第一交易请求将第三区块链系统作为第 i 个区块链系统对应的上一层区块链系统，所述第三区块链系统是所述第一区块链系统和其余 N-1 个第二区块链系统中的一个；交易执行单元 1203，配置为根据所述第一交易执行所述第一方法函数，实现：生成主权迁移事件，所述主权迁移事件
25 中包括所述第 i 个第二区块链系统和所述第三区块链系统的标识信息，使所述第三区块链系统利用所述第 i 个第二区块链系统的注册信息，在所述第三区块链系统中设置与所述第 i 个第二区块链系统对应的第二卷叠合约。

本说明书实施例中还提供了一种计算机可读存储介质，其上存储有计算机程序/指令，当所述计算机程序/指令在计算机中执行时，令计算机执行前述各个实施例中由
30 区块链系统 B11、区块链系统 B21、区块链系统 B31、第三或第四区块链系统所执行的方法步骤。

本说明书实施例中还提供了一种计算设备，包括存储器和处理器，所述存储器中存储有计算机程序/指令，所述处理器执行所述计算机程序/指令时，实现前述各个实施例中由区块链系统 B11、区块链系统 B21、区块链系统 B31、第三或第四区块链系统
35 所执行的方法步骤。

在 20 世纪 90 年代，对于一个技术的改进可以很明显地区分是硬件上的改进（例如，对二极管、晶体管、开关等电路结构的改进）还是软件上的改进（对于方法流程的改进）。然而，随着技术的发展，当今的很多方法流程的改进已经可以视为硬件电路结构的直接改进。设计人员几乎都通过将改进的方法流程编程到硬件电路中来得到

相应的硬件电路结构。因此，不能说一个方法流程的改进就不能用硬件实体模块来实现。例如，可编程逻辑器件（Programmable Logic Device, PLD）（例如现场可编程门阵列（Field Programmable Gate Array, FPGA））就是这样一种集成电路，其逻辑功能由用户对器件编程来确定。由设计人员自行编程来把一个数字系统“集成”在一

5 片 PLD 上，而不需要请芯片制造厂商来设计和制作专用的集成电路芯片。而且，如今，取代手工地制作集成电路芯片，这种编程也多半改用“逻辑编译器（logic compiler）”软件来实现，它与程序开发撰写时所用的软件编译器相类似，而要编译之前的原始代码也得用特定的编程语言来撰写，此称之为硬件描述语言（Hardware Description Language, HDL），而 HDL 也并非仅有一种，而是有许多种，如 ABEL（Advanced Boolean

10 Expression Language）、AHDL（Altera Hardware Description Language）、Confluence、CUPL（Cornell University Programming Language）、HDL、JHDL（Java Hardware Description Language）、Lava、Lola、MyHDL、PALASM、RHD（Ruby Hardware Description Language）等，目前最普遍使用的是 VHDL（Very-High-Speed Integrated Circuit Hardware Description Language）与 Verilog。本领域技术人员也应该清楚，只需要

15 将方法流程用上述几种硬件描述语言稍作逻辑编程并编程到集成电路中，就可以很容易得到实现该逻辑方法流程的硬件电路。

控制器可以按任何适当的方式实现，例如，控制器可以采取例如微处理器或处理器以及存储可由该（微）处理器执行的计算机可读程序代码（例如软件或固件）的计算机可读介质、逻辑门、开关、专用集成电路（Application Specific Integrated

20 Circuit, ASIC）、可编程逻辑控制器和嵌入微控制器的形式，控制器的例子包括但不限于以下微控制器：ARC 625D、Atmel AT91SAM、Microchip PIC18F26K20 以及 Silicone Labs C8051F320，存储器控制器还可以被实现为存储器的控制逻辑的一部分。本领域技术人员也知道，除了以纯计算机可读程序代码方式实现控制器以外，完全可以通过将方法步骤进行逻辑编程来使得控制器以逻辑门、开关、专用集成电路、可编程逻辑

25 控制器和嵌入微控制器等的形式来实现相同功能。因此这种控制器可以被认为是一种硬件部件，而对其内包括的用于实现各种功能的装置也可以视为硬件部件内的结构。或者甚至，可以将用于实现各种功能的装置视为既可以是实现方法的软件模块又可以是硬件部件内的结构。

上述实施例阐明的系统、装置、模块或单元，具体可以由计算机芯片或实体实现，

30 或者由具有某种功能的产品来实现。一种典型的实现设备为服务器系统。当然，本申请不排除随着未来计算机技术的发展，实现上述实施例功能的计算机例如可以为个人计算机、膝上型计算机、车载人机交互设备、蜂窝电话、相机电话、智能电话、个人数字助理、媒体播放器、导航设备、电子邮件设备、游戏控制台、平板计算机、可穿戴设备或者这些设备中的任何设备的组合。

35 虽然本说明书一个或多个实施例提供了如实施例或流程图所述的方法操作步骤，但基于常规或者无创造性的手段可以包括更多或者更少的操作步骤。实施例中列举的步骤顺序仅仅为众多步骤执行顺序中的一种方式，不代表唯一的执行顺序。在实际中的装置或终端产品执行时，可以按照实施例或者附图所示的方法顺序执行或者并行执行（例如并行处理器或者多线程处理的环境，甚至为分布式数据处理环境）。术语“包

括”、“包含”或者其任何其他变体意在涵盖非排他性的包含，从而使得包括一系列要素的过程、方法、产品或者设备不仅包括那些要素，而且还包括没有明确列出的其他要素，或者是还包括为这种过程、方法、产品或者设备所固有的要素。在没有更多限制的情况下，并不排除在包括所述要素的过程、方法、产品或者设备中还存在另外的相同或等同要素。例如若使用到第一，第二等词语用来表示名称，而并不表示任何特定的顺序。

5 为了描述的方便，描述以上装置时以功能分为各种模块分别描述。当然，在实施本说明书一个或多个时可以把各模块的功能在同一个或多个软件和/或硬件中实现，也可以将实现同一功能的模块由多个子模块或子单元的组合实现等。以上所描述的装置
10 实施例仅仅是示意性的，例如，所述单元的划分，仅仅为一种逻辑功能划分，实际实现时可以有另外的划分方式，例如多个单元或组件可以结合或者可以集成到另一个系统，或一些特征可以忽略，或不执行。另一点，所显示或讨论的相互之间的耦合或直接耦合或通信连接可以是通过一些接口，装置或单元的间接耦合或通信连接，可以是电性，机械或其它的形式。

15 本发明是参照根据本发明实施例的方法、装置（系统）、和计算机程序产品的流程图和/或方框图来描述的。应理解可由计算机程序指令实现流程图和/或方框图中的每一流程和/或方框、以及流程图和/或方框图中的流程和/或方框的结合。可提供这些计算机程序指令到通用计算机、专用计算机、嵌入式处理机或其他可编程数据处理设备的处理器以产生一个机器，使得通过计算机或其他可编程数据处理设备的处
20 理器执行的指令产生用于实现在流程图一个流程或多个流程和/或方框图一个方框或多个方框中指定的功能的装置。

这些计算机程序指令也可存储在能引导计算机或其他可编程数据处理设备以特定方式工作的计算机可读存储器中，使得存储在该计算机可读存储器中的指令产生包括指令装置的制造品，该指令装置实现在流程图一个流程或多个流程和/或方框图一个
25 方框或多个方框中指定的功能。

这些计算机程序指令也可装载到计算机或其他可编程数据处理设备上，使得在计算机或其他可编程设备上执行一系列操作步骤以产生计算机实现的处理，从而在计算机或其他可编程设备上执行的指令提供用于实现在流程图一个流程或多个流程和/或方框图一个方框或多个方框中指定的功能的步骤。

30 在一个典型的配置中，计算设备包括一个或多个处理器(CPU)、输入/输出接口、网络接口和内存。

内存可能包括计算机可读介质中的非永久性存储器，随机存取存储器(RAM)和/或非易失性内存等形式，如只读存储器(ROM)或闪存(flash RAM)。内存是计算机可读介质的示例。

35 计算机可读介质包括永久性和非永久性、可移动和非可移动媒体可以由任何方法或技术来实现信息存储。信息可以是计算机可读指令、数据结构、程序的模块或其他数据。计算机的存储介质的例子包括，但不限于相变内存(PRAM)、静态随机存取存储器(SRAM)、动态随机存取存储器(DRAM)、其他类型的随机存取存储器(RAM)、只读存储器(ROM)、电可擦除可编程只读存储器(EEPROM)、快闪记忆体或其他内存技术、只读光

盘只读存储器 (CD-ROM)、数字多功能光盘 (DVD) 或其他光学存储、磁盒式磁带，磁带磁盘存储、石墨烯存储或其他磁性存储设备或任何其他非传输介质，可用于存储可以被计算设备访问的信息。按照本文中的界定，计算机可读介质不包括暂存电脑可读媒体 (transitory media)，如调制的数据信号和载波。

- 5 本领域技术人员应明白，本说明书一个或多个实施例可提供为方法、系统或计算机程序产品。因此，本说明书一个或多个实施例可采用完全硬件实施例、完全软件实施例或结合软件和硬件方面的实施例的形式。而且，本说明书一个或多个实施例可采用在一个或多个其中包含有计算机可用程序代码的计算机可用存储介质（包括但不限于磁盘存储器、CD-ROM、光学存储器等）上实施的计算机程序产品的形式。
- 10 本说明书一个或多个实施例可以在由计算机执行的计算机可执行指令的一般上下文中描述，例如程序模块。一般地，程序模块包括执行特定任务或实现特定抽象数据类型类型的例程、程序、对象、组件、数据结构等等。也可以在分布式计算环境中实践本说明书一个或多个实施例，在这些分布式计算环境中，由通过通信网络而被连接的远程处理设备来执行任务。在分布式计算环境中，程序模块可以位于包括存储设备在内的本地和远程计算机存储介质中。
- 15 本说明书中的各个实施例均采用递进的方式描述，各个实施例之间相同相似的部分互相参见即可，每个实施例重点说明的都是与其他实施例的不同之处。尤其，对于系统实施例而言，由于其基本相似于方法实施例，所以描述的比较简单，相关之处参见方法实施例的部分说明即可。在本说明书的描述中，参考术语“一个实施例”、“一些实施例”、“示例”、“具体示例”、或“一些示例”等的描述意指结合该实施例或示例描述的具体特征、结构、材料或者特点包含于本说明书的至少一个实施例或示例中。在本说明书中，对上述术语的示意性表述不必须针对的是相同的实施例或示例。而且，描述的具体特征、结构、材料或者特点可以在任一个或多个实施例或示例中以合适的方式结合。此外在不相互矛盾的情况下，本领域的技术人员可以将本说明书中描述的不同实施例或示例以及不同实施例或示例的特征进行结合和组合。
- 20 以上所述仅为本说明书一个或多个实施例的实施例而已，并不用于限制本说明书一个或多个实施例。对于本领域技术人员来说，本说明书一个或多个实施例可以有各种更改和变化。凡在本说明书的精神和原理之内所作的任何修改、等同替换、改进等，均应包含在权利要求范围之内。
- 25

权 利 要 求 书

1. 一种区块链系统的管理方法，所述方法包括：

第一区块链系统接收第一交易，所述第一区块链系统中部署有包括第一方法函数的第一卷叠合约，所述第一卷叠合约的合约状态中存储有 N 个第二区块链系统的注册信息，所述第一交易用于调用所述第一方法函数，所述第一交易请求将第三区块链系统作为第 i 个区块链系统对应的上一层区块链系统，所述第三区块链系统是所述第一区块链系统和其余 $N-1$ 个第二区块链系统中的一个；

所述第一区块链系统根据所述第一交易执行所述第一方法函数，实现：生成主权迁移事件，所述主权迁移事件中包括所述第 i 个第二区块链系统和所述第三区块链系统的标识信息；

所述第三区块链系统根据所述主权迁移事件，利用所述第 i 个第二区块链系统的注册信息，在所述第三区块链系统中设置与所述第 i 个第二区块链系统对应的第二卷叠合约。

2. 根据权利要求 1 所述的方法，所述方法还包括：所述第三区块链系统在所述第二卷叠合约的合约状态中，存储所述第 i 个区块链系统最新提交并已进入不可更改状态的第一交易组 batch 所对应的第一状态数据。

3. 根据权利要求 1 所述的方法，所述第三区块链系统根据所述主权迁移事件，利用所述第 i 个第二区块链系统的注册信息，在所述第三区块链系统中设置与所述第 i 个第二区块链系统对应的第二卷叠合约，包括：所述第三区块链系统根据所述主权迁移事件，在其已部署的第二卷叠合约中生效所述第 i 个第二区块链系统的注册信息。

4. 根据权利要求 1 所述的方法，所述第一卷叠合约中还包括第二方法函数；

其中，所述方法还包括：

所述第一区块链系统接收用于调用所述第二方法函数的第二交易，所述第二交易请求第四区块链系统在所述第 i 个区块链系统提交的第一交易 batch 进入不可更改状态后，不再作为所述第 i 个第二区块链系统对应的上一层区块链系统，所述第四区块链系统是所述第一区块链系统和其余 $N-1$ 个第二区块链系统中不同于所述第三区块链系统的区块链系统；

所述第一区块链系统根据所述第二交易执行所述第二方法函数，实现：生成主权终止事件，所述主权终止事件中包括所述第一交易 batch 的 batch 序号、所述第 i 个第二区块链系统和所述第四区块链系统的标识信息；

所述第四区块链系统根据所述主权终止事件，在确认所述第一交易 batch 已进入不可更改状态后，在其部署的第三卷叠合约中无效所述第 i 个第二区块链系统的注册信息。

5. 根据权利要求 4 所述的方法，所述第一卷叠合约的合约状态中包括所述 N 个第二区块链系统的全局管理信息，所述第四区块链系统属于其余 $N-1$ 个区块链系统，所述第一卷叠合约中还包括第三方法函数；其中，所述方法还包括：

所述第一区块链系统从所述第四区块链系统接收用于调用所述第三方法函数的第三交易，所述第三交易中包括所述第四区块链系统提交的第二交易 batch 所包括的交易序列、所述第二交易 batch 所对应的前状态根和后状态根，以及所述第三卷叠合约

的合约状态中存储的所述第 i 个第二区块链系统的标识信息和第二状态数据；

所述第一区块链系统根据所述第三交易执行所述第三方法函数，实现：存储所述第二交易 batch 所包括的交易序列，在所述第四区块链系统的全局管理信息中，根据所述第二交易 batch 所对应的前状态根和后状态根更新所述第四区块链系统的第三状态数据，以及在所述第 i 个第二区块链系统的全局管理信息中，根据所述第 i 个第二区块链系统的标识信息和第二状态数据，更新所述第 i 个第二区块链系统的第四状态数据。

6. 根据权利要求 5 所述的方法，所述第三交易中还包括所述第二交易 batch 的 batch 序号；其中，所述根据所述第三交易执行所述第三方法函数，还实现：在所述第四区块链系统的全局管理信息中，根据所述第二交易 batch 的 batch 序号新增一条 batch 位置索引，用于指示所述第二交易 batch 在所述第一区块链系统中的存储位置。

7. 根据权利要求 6 所述的方法，所述 batch 位置索引包括：所述第二交易 batch 的 batch 序号、所述第三交易在所述第一区块链系统中所属区块的区块号和交易序号。

8. 根据权利要求 5 所述的方法，所述主权迁移事件中还包括所述第一状态数据，所述第一状态数据是所述第 i 个第二区块链系统的全局管理信息中所包括的所述第四状态数据。

9. 根据权利要求 1 所述的方法，所述第一卷叠合约的合约状态中还包括所述 N 个第二区块链系统的全局管理信息；其中，所述根据所述第一交易执行所述第一方法函数，还实现：在所述第 i 个第二区块链系统的全局管理信息中，新增一条主权迁移记录，包括所述第三区块链系统的标识信息和所述第一交易 batch 的下一个交易 batch 所对应的 batch 序号。

10. 根据权利要求 9 所述的方法，所述主权迁移记录中还包括所述第一状态数据。

11. 根据权利要求 1 所述的方法，所述主权迁移事件中还包括所述第 i 个第二区块链系统的注册信息。

12. 根据权利要求 1 所述的方法，所述第三区块链系统属于其余 N-1 个第二区块链系统，所述第一卷叠合约还包括第四方法函数；其中，所述方法还包括：

所述第一区块链系统从所述第 i 个第二区块链系统接收用于调用所述第四方法函数的第四交易，所述第四交易请求由复用所述第三区块链系统中用于执行预定事务的基础设施；

所述第一区块链系统根据所述第四交易执行所述第四方法函数，实现：生成设施复用事件，其中包括所述第 i 个第二区块链系统和所述第三区块链系统的标识信息；

所述第三区块链系统根据所述设施复用事件，通过所述第三区块链系统中的基础设施执行所述第 i 个第二区块链系统中所需执行的所述预定事务。

13. 根据权利要求 12 所述的方法，所述第三区块链系统中的基础设施包括如下各项中的至少一项：排序器、证明器以及中继器。

14. 根据权利要求 1-13 中任一项所述的方法，所述第 i 个第二区块链系统的注册信息包括以下各项信息中的至少一项：标识信息、创世块、管理成员地址集合、排序器地址集合、交易 batch 的存储方式、Rollup 机制以及其对应的验证算法。

15. 一种区块链系统的管理方法，所述方法由部署有第一卷叠合约的第一区块链

系统执行，所述第一卷叠合约中包括第一方法函数，所述第一卷叠合约的合约状态中存储有 N 个第二区块链系统的注册信息，所述方法包括：

接收用于调用所述第一方法函数的第一交易，所述第一交易请求将第三区块链系统作为第 i 个区块链系统对应的上一层区块链系统，所述第三区块链系统是所述第一区块链系统和其余 N-1 个第二区块链系统中的一个；

根据所述第一交易执行所述第一方法函数，实现：生成主权迁移事件，所述主权迁移事件中包括所述第 i 个第二区块链系统和所述第三区块链系统的标识信息，使所述第三区块链系统根据所述主权迁移事件，利用所述第 i 个第二区块链系统的注册信息，在所述第三区块链系统中设置与所述第 i 个第二区块链系统对应的第二卷叠合约。

16. 一种第一区块链系统中的区块链节点，所述第一区块链系统中部署有第一卷叠合约，所述第一卷叠合约中包括第一方法函数，所述第一卷叠合约的合约状态中存储有 N 个第二区块链系统的注册信息，所述区块链节点包括：

交易接收单元，配置为接收用于调用所述第一方法函数的第一交易，所述第一交易请求将第三区块链系统作为第 i 个区块链系统对应的上一层区块链系统，所述第三区块链系统是所述第一区块链系统和其余 N-1 个第二区块链系统中的一个；

交易执行单元，配置为根据所述第一交易执行所述第一方法函数，实现：生成主权迁移事件，所述主权迁移事件中包括所述第 i 个第二区块链系统和所述第三区块链系统的标识信息，使所述第三区块链系统根据所述主权迁移事件，利用所述第 i 个第二区块链系统的注册信息，在所述第三区块链系统中设置与所述第 i 个第二区块链系统对应的第二卷叠合约。