# 文档说明

此文档介绍了Lanchain平台针对各场景的扩容方案。

# 应用场景与扩容方案

## 2.1 竞拍场景扩容方案

### 1. 场景介绍

区块链上的竞拍流程与传统拍卖方式类似.首先,拍卖者向区块链平台提交竞拍物品信息及发布竞拍信息.在指定的拍卖时间范围内,认证用户通过提交自己竞拍价格的方式参与竞拍.用户能够通过平台接口周期性获得当前最高竞拍价格.最终,在拍卖时间结束时最高价格的用户获得拍卖品.

在区块链平台上进行竞拍具有防欺诈,过程透明可查等特点.

此竞拍场景扩容方案只关注竞拍过程的性能及吞吐率的提高,不关注存证(例如:竞拍品信息验证及存证)及身份认证过程(例如:只有授权用户才能发布竞拍信息及参与竞拍).

### 方案介绍

此扩容方案采用单一根链与多个子链的方式进行扩容,目的是提高竞拍场景中大量竞拍行为同时发生时平台的交易吞吐率,提高平台整体的交易tps.

方案中竞拍行为在各子链上发生, keeper服务周期性的扫描所有子链,从各子链获得一定时间范围内的最高竞价信息并更新到根链上.最终通过调用根链上的合约接口获得最终竞价信息.

方案特点如下:

* 联盟内通过共识的方式管理子链（创建，删除）,保证子链安全性及可信性.
* 通过随时增加子链的数量来达到动态扩充平台容量(吞吐率,整体tps等)的目的.
* keeper服务程序采用中心化或分布式方式以适用于不同安全环境.
* 主链的最高竞拍记录包含子链id,块hash及交易hash及keeper标识,方便数据快速查找及验证.

竞拍场景扩容方案示意图如下:

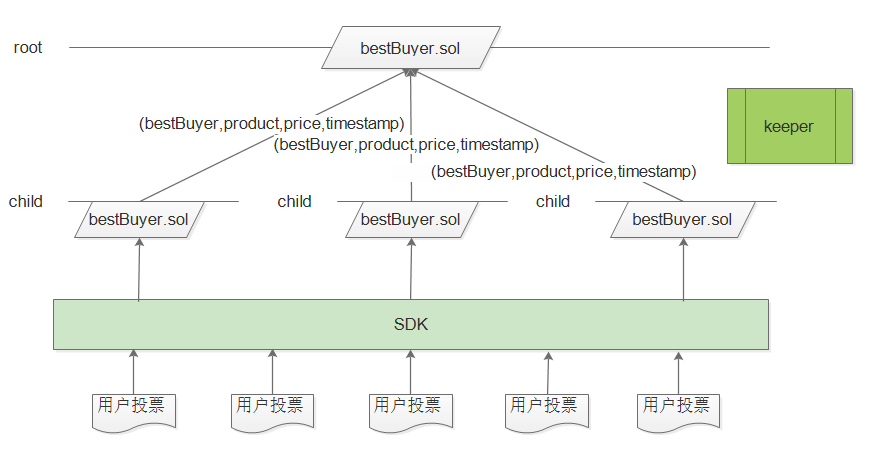


图1. 竞拍场景扩容方案示意图

方案中各部分说明如下:

* root: 根链,可以为wanchain等公链,也可以为指定联盟链.
* child: 子链.
* Keeper: 服务程序,负责子链与根链之间的数据更新,具体参见下文.
* bestBuyer.sol: 竞拍示例合约,同时部署在根链及所有子链上.
* SDK: 提供给用户的平台API. SDK需要提供在各子链间进行负载均衡的能力,具体参见下文.
* 用户投票: 用户通过调用平台SDK进行竞拍信息提交及参与竞拍,具体参见下文.

#### 子链管理

此方案中子链的创建及销毁需要在联盟内部通过共识算法达成共识.只有达成共识之后联盟的各节点才能创建或者删除子链.从而保护子链在联盟中的可信性及稳定性.

子链共识结果存在根链的锚定合约anchor.sol中,此合约不止存储子链创建时的共识证明,还能够存储各子链的其他锚定信息(此方案中不需要此功能).

子链管理示意图如下:

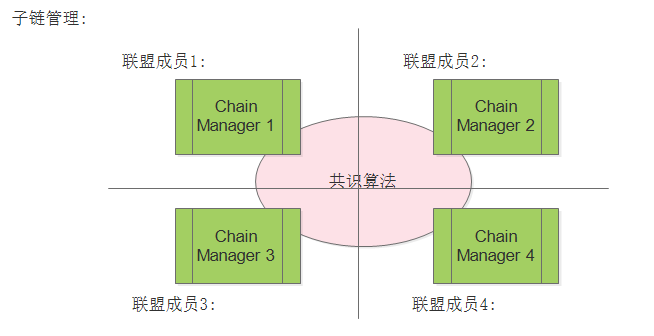


图2. 子链管理示意图

Anchor.sol合约主要成员说明如下:

*childChain:*

*String id: 子链id.*

*Int status: 子链当前状态: 运行,销毁.*

*Mapping History: history为一个映射.*

*String Action: 子链操作行为.*

*Mapping Signers: 各方签名,只有联盟成员的签名达到一定要求,action才会被执行.*

*Int timestamp: 时间戳.*

#### 竞拍交易部署

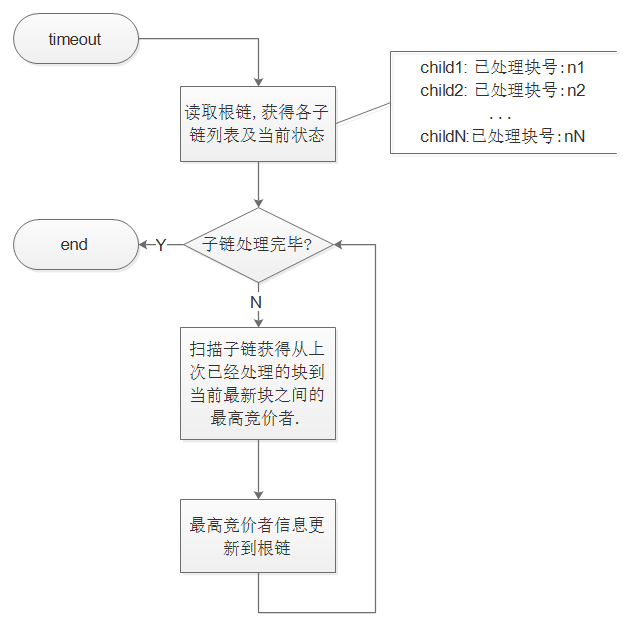
* 用户通过调用平台SDK完成竞拍交易的部署.
* SDK内部负责在所有当前正在运行的子链上部署竞拍交易并返回部署结果给用户.
* 用户对竞拍交易部署的部署过程及子链架构无感知.

#### **竞拍过程**

* 用户通过调用平台SDK参与竞拍.
* SDK根据各子链的负载进行负载均衡,把用户的竞拍请求均衡分配到不同子链上.
* 用户调用SDK接口监听竞拍价格更新通知.
* 用户对子链架构无感知.

#### **keeper服务相关**

* keeper服务运行流程如下:



图三. keeper处理流程.

* keeper周期性向主链提交的信息主要结构如下:

*bestBuyer {*

*buyerHash: 最高竞价者的唯一标识.*

*productHash: 竞品唯一标识.*

*Price: 竞拍价格.*

*timeStamp: 时间戳.*

*txHash: 最高竞价所在的交易hash.*

*blockHash: 最高竞价所在的块hash.*

*chainId: 最高竞价所在的子链ID.*

*signedKeepers: keeper签名(分布式keeper需要多方签名).*

*}*

* keeper中心化与分布式说明

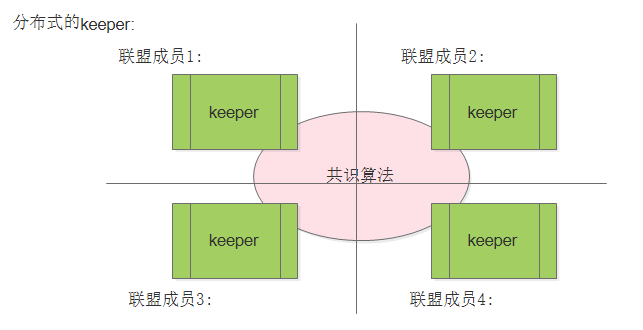
keeper作为一个链外服务程序可以采用中心化方式或者分布式方式部署,下面分别说明两种方式的优势及使用场景.

keeper中心化部署方式:

* 效率高.
* 部署及实现简单.
* 不能防止单keeper作弊(单点作弊).
* 虽然不能防止单点作弊,但是能够通过链上数据验证keeper是否作弊.

keeper分布式部署方式:

keeper分布式部署结构图如下:



图四. 分布式keeper结构图

分布式keeper特点如下:

* 多个keeper部署在联盟中不同成员中,保证分布性.
* 各keeper独立从各child链获取bestBuyer.
* keeper间通过共识协议对准备提交到主链的信息进行确认.
* 由提交节点负责提交bestkeeper到root chain上.
* 分布式方式能够防止单keeper作弊.

### 开放问题

#### keeper提交信息的可信性问题

**分布式keeper方式:**

**数据提交前验证:**

此方式每次提交都需要首先在各联盟成员keeper间对提交数据达成共识.这种方式在keeper提交信息到根链之前就已经完成了信息的可信性验证.

在keeper提交信息中包含各keeper的签名,通过链下签名，链上验签的方式保证信息可信性.

**数据提交后验证:**

此方式各keeper监听特定合约事件.在指定keeper提交数据后,触发合约事件.其他keeper收到合约事件后对已上链数据进行验证,并把验证结果(签名)上链.

**中心化keeper方式:**

中心化keeper需要运行在受信环境中,但是仍然存在单点作弊的可能.在部署时需要根据实际情况综合考量.

对于中心化及分布式keeper部署方式,均可以通过对比根链及子链上数据一致性的方式来检查竞拍交易的可信性.

**是否可以对keeper提交的信息进行实时验证?**

对keeper提交的信息实时验证目前有如下两种设想方案:

方案一: gwan能够跨链访问链外数据.

访问链外数据可以通过如下两种方法实现:

方法一:修改EVM代码实现,在EVM执行交易时能够访问链外数据源(子链)进行信息验证.（需要深入分析EVM机制）

方法二:通过go语言实现预编译合约,在合约中实现链外数据的访问(目前没有对此方案深入分析)

在交易执行过程中访问链外数据进行可信性验证需要考虑到网络情况,延时等对交易处理过程的影响.

方案二:

准实时方法。

通过交易提交后发出事件，各keeper对事件响应并处理的方式保证数据可信性。参考”数据提交后验证”部分.

此方案需要多笔交易对一次提交做确认.已经上链的数据并不能确保可靠,需要后续多笔交易中各keeper对此数据签名.

## 2.2 身份验证场景方案

### 1. 场景介绍

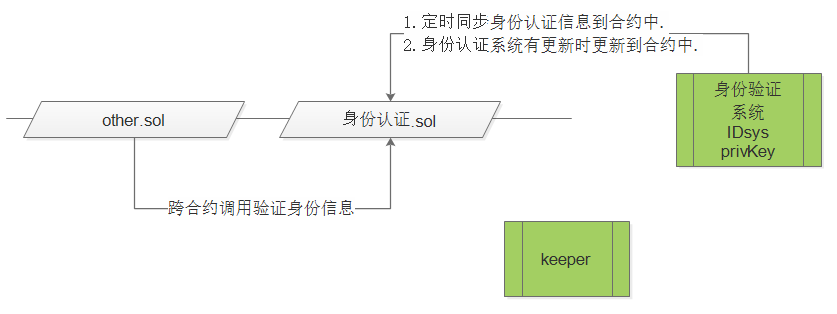
身份验证场景本身不涉及到对区块链性能及交易吞吐率的扩容.

在区块链的各种场景中往往需要对执行交易的账户做权限验证,例如:验证账户是否能够操作合约中的指定函数等.

### 2. 方案一: 链上身份验证

#### 2.1 方案说明

方案示意图如下:



图五. 链上身份验证

* 身份认证.sol保持与身份认证系统同步.
* keeper服务负责定时及在有更新时把认证信息更新到链上合约中.
* keeper根据可信性可以采用中心化或者分布式方式部署。
* Other.sol为业务合约,内部调用身份认证合约接口进行链上身份认证.

#### 2.2 方案优势及特点分析

链上认证方案优势:

实现简单,不需要修改现有gwan平台组件实现即可实现此方案.

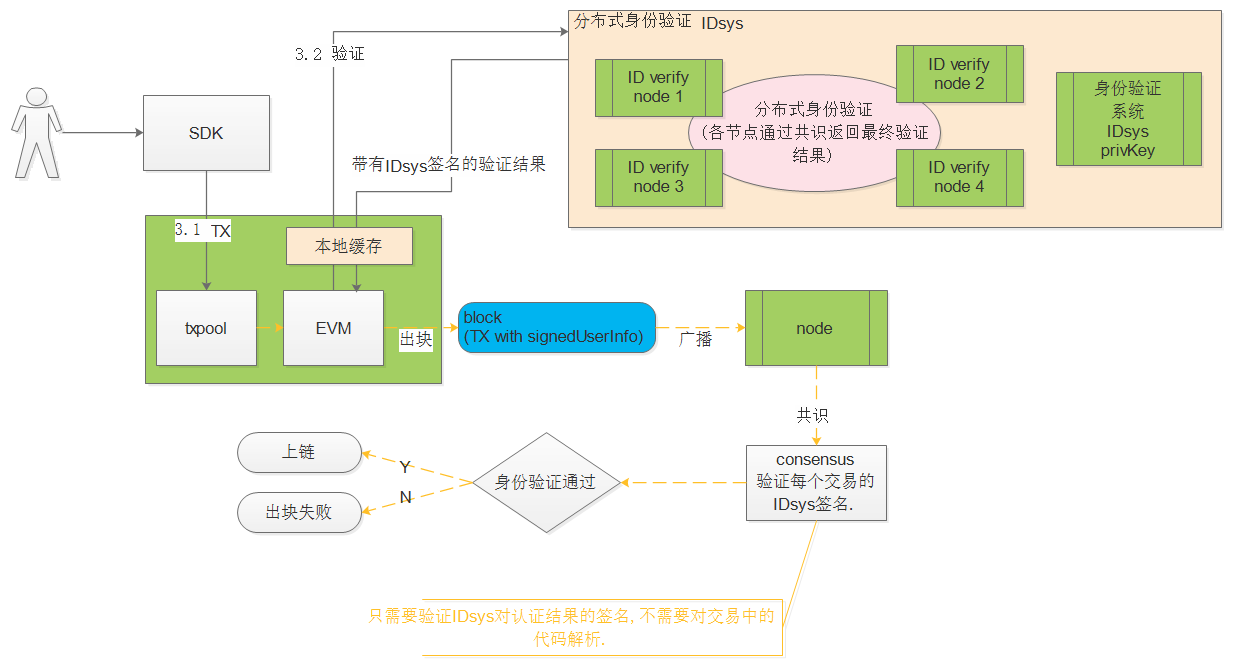
链上认证方案的不足如下:

1. 在实际部署中,身份认证系统可能为第三方系统,需要考虑认证信息是否能够公开及上链.
2. keeper可信性: 通过分布式共识解决。参考竞拍场景扩容方案中的开放问题.
3. 多个链如果需要身份认证,那么每个链都要存储身份认证信息,链上数据量较大.
4. 链上及链下认证信息有可能不同步.

### 3. 方案二: 链下身份认证

#### 3.1 方案说明

方案示意图如下:



图六. 链下方案示意图

* 用户通过SDK发送交易.
* 如果合约函数调用了身份验证接口,那么在EVM执行这个函数的时候会进行身份验证.在链下验证方案中,EVM会调用身份验证系统接口(socket接口,rpc调用等)发送验证请求并接收验证结果.
* 身份验证通过,交易能够被打包到区块中,否则为无效交易,用户收到无效交易提示.
* 交易数据结构中带有身份验证信息作为交易的一部分,这些信息会跟随交易一并打入到区块中.
* 区块在各节点进行共识的过程中,各节点会扫描交易列表,对需要身份验证的交易进行验证检查.根据共识算法,节点对身份验证结果达成一致通过,区块才能真正上链,否在,共识算法需要返回区块共识失败,区块为无效区块,不能上链.

##### 主要数据结构说明

1. 验证请求数据结构

验证请求为EVM发送给身份验证系统的验证请求信息.

主要结构如下:

Request {

Transaction hash; //交易hash

From; //账户地址

Function name; //函数名称

}

1. 验证结果数据结构

Result {

Transaction hash; //交易hash

From; //账户地址

Function name; //函数名称

Result; //验证结果,允许/不允许

startTime; //认证结果有效期的起始时间.

Duration; //认证结果有效性的持续时间.

Signers; //身份验证系统签名,可能有多个签名.

}

##### 关键处理流程说明

1. 节点本地缓存认证信息

区块链节点在本地通过图六中的”本地缓存”模块来缓存认证信息.在认证过程中首先访问本地缓存,如果本地缓存中存在认证结果,那么对结果中的有效期起始时间及持续时间进行检查.结果在有效期之内,则不需要再次访问认证系统.如果认证结果过期,需要重新访问认证系统,并在缓存中更新认证结果.

1. 身份认证系统

身份认证系统可以为第三方系统,也可以是另外一个存储认证信息的区块链.在EVM执行交易的过程中通过调用身份认证系统的API获得身份认证结果.

身份认证系统可以为中心化,也可以为分布式.其自身持有自己的私钥,并通过私钥对身份认证结果进行签名,需要对身份认证信息验证的节点持有身份认证系统的公钥,通过验签过程来保证认证结果有效及可靠.

1. 块有效性验证

以目前的IBFT共识算法为例,在节点间通过共识验证块有效性的过程中,需要对块中的所有交易的身份认证信息进行检查,验证所有交易的权限.只有所有交易的权限检查通过,那么节点才能认为新块是有效的.

在块共识过程中对交易进行身份验证是为了方式单点作弊的发生.

节点在对块中交易进行身份验证的过程中只需要使用自身保存的认证系统公钥对交易中认证系统的签名验签及对认证内容进行检查即可,不需要访问身份认证系统.

1. 认证函数列表合约

认证函数列表合约存储了需要进行认证的函数列表.主要目的是为了防止单节点“对身份认证失败的交易故意不带有认证信息”的作弊情况发生。

此合约可以在编译的时候包含进业务合约,使得在对交易的执行过程进行身份验证时不需要频繁跨合约调用.

##### 安全及防作弊说明

此方案从以下几个方面对节点作弊可能性做了分析.

1. 单节点伪造认证结果信息

例如: 交易认证系统返回的结果为不通过,节点修改认证结果为通过.

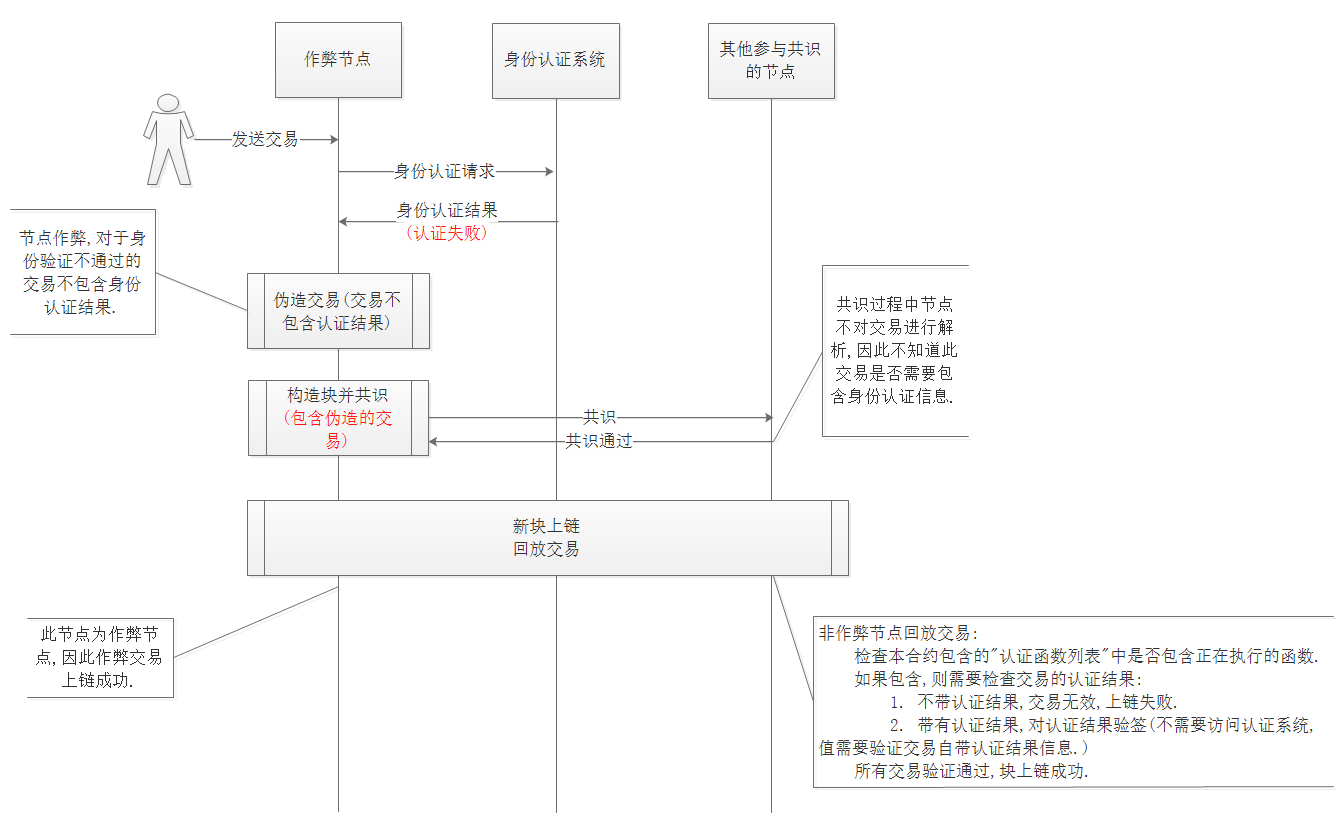
出块后节点间会通过共识算法对块进行有效性验证,在验证过程中,伪造交易验签失败,导致新块验证失败,此块不会上链.

1. 单节点作弊,作弊交易中不带有认证信息

例如: 对于需要带有认证信息的交易(调用了身份验证函数), 作弊节点不把验证结果存入交易中. 因为在块验证过程中并不会解析交易数据及回放交易,因此验证节点并不能知道交易是否需要带有验证信息.

对于此种情况,在块共识过程中不会检测到作弊行为.块经过共识过程视为有效块并广播到所有节点进行上链.在各节点对块上交易进行回放过程中根据”认证函数列表合约”判断是否需要对交易进行身份验证,如需要身份验证则对比交易附带的身份认证信息.

示意流程图如下:



图七. 防作弊流程示意图

#### 3.2 开放问题

1. 链外数据访问

此问题关注于如何在交易执行过程中访问链外的认证系统.

可参考目前摩根大通区块链的实现.在摩根大通的私有交易处理过程中,会访问链外的私有交易处理系统.

1. 交易执行过程中对身份进行验证

目前的方案考虑在EVM执行交易的过程中,对要执行的函数进行检测,对于需要身份验证的函数进行身份验证.

需要进一步确认此种方式的可行性及实现复杂度.

需要从实现角度考虑是否有其他实现复杂度更低的方法.

1. 身份验证过程中的异常处理

此方案中的身份验证过程可能会遇到网络超时,访问不可达等情况.目前对这些情况的处理方式为视为认证失败.

1. 提高身份认证过程效率

因为身份认证过程是在交易执行过程中发生.需要考虑采用并发机制,流水线,预认证等方式提交身份认证过程的效率,以减少交易处理时间,加快系统整体对交易的处理速度.

1. 各节点认证信息获取不一致问题

需要确保在身份认证系统修改了认证信息后各节点获取的信息一致性问题。

### 4. 方案综合分析

对于身份认证场景主要考虑了链上和链下两种身份认证方式.

其中,链上方式部署实现较链下方式更为简单.

链下方式实现较为复杂,对于方案中的一些具体步骤需要进一步从实现角度更深入分析复杂度及考虑是否有更优方法.

## 2.3 存证场景扩容方案

#### 1. 场景介绍

此场景主要针对证据存证信息进行上链.

实际应用中,存证信息上链可划分为几种类型:

1. 单一存证信息上链:

A: 存证信息A上链.

1. 关联存证信息上链:
   1. 单一关联信息:
2. >B: 存证信息(A,B)上链,其中A的有效性依赖于B在链上存在.
   1. 多关联信息:
3. >(B,C,D):存证信息(A,B,C,D)上链,其中A的有效性依赖于B,C,D在链上存在.B,C,D之间为并行关系,不存在互相依赖.

此种关系可退化为单一关系信息A->B,A->C,A->D.

综合上述存证信息的类型提出了两种扩容方案:

方案一不检查存证信息的关联性,多个子链存储的内容相同;

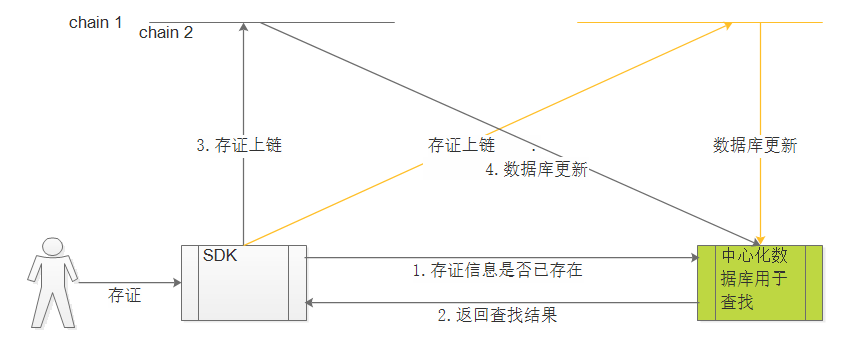
方案二考虑了存证各属性之间的关联性,存证信息的各部分存储在不同的子链上,在存证信息上链的时候会检查信息完备性,只有检查通过的信息才能够上链.

下面分别对两个方案进行说明.

#### 2. 方案一: 不对存证分区存储

此方案中多个子链的功能相同,每个子链存储的内容也相同,不对子链功能做区分.

方案示意图如下:



图八. 不对存证分区存储的扩容方案

##### 2.1 说明

此方案中所有子链的功能及存储数据,合约完全相同.由SDK负责在多个链之间进行负载均衡.

如需扩充平台性能,只需要创建一个新的子链,SDK把新子链添加到负载均衡列表中,后续的交易在所有子链中进行负载均衡.

##### 2.2 方案优势及特点分析

方案优点:

实现及架构简单.

方案缺点:

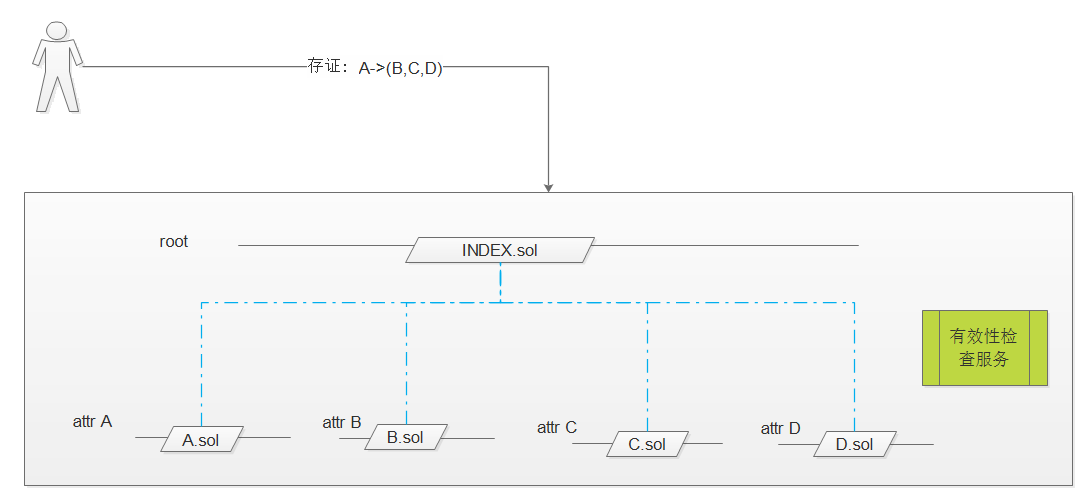
* 各子链完全相同,存在数据冗余.
* 无法检查属性依赖性及存证完备性.

#### 方案二: 属性分区及哈希锁授权

此方案中，通过哈希锁方式保证了用户对平台的上链行为认可之后存证完整信息才能进入到索引合约INDEX.sol中。只有存证完整进入到了索引合约中才能视为存证有效。

方案中,把存证信息按照属性进行拆分,平台对存证依赖的各属性进行检查,检查通过后存证属性能够被添加到子链，然后用户对已经上链的属性进行检查，如果用户对已上链结果认可，则通过哈希锁方式授权平台把存证添加到根链索引合约中，存证被添加到索引合约之后此存证信息才真正有效。

方案结构图如下:



图九. 存证完备性检查方案示意图

关键组件说明:

* Root: 此方案中根链合约INDEX.sol具有两个功能:

1. 存储存证的标识信息,根据此信息能够组合完整存证信息.
2. 存证的有效性验证信息,确保存证的真实可靠.
3. 只有在INDEX.sol的存证信息才是真实有效的。

* 子链: 按照属性划分,各子链存储不同的存证属性.

属性在子链上存在并不能保证合约整体有效。

* 有效性检查服务:

此服务提供的功能如下:

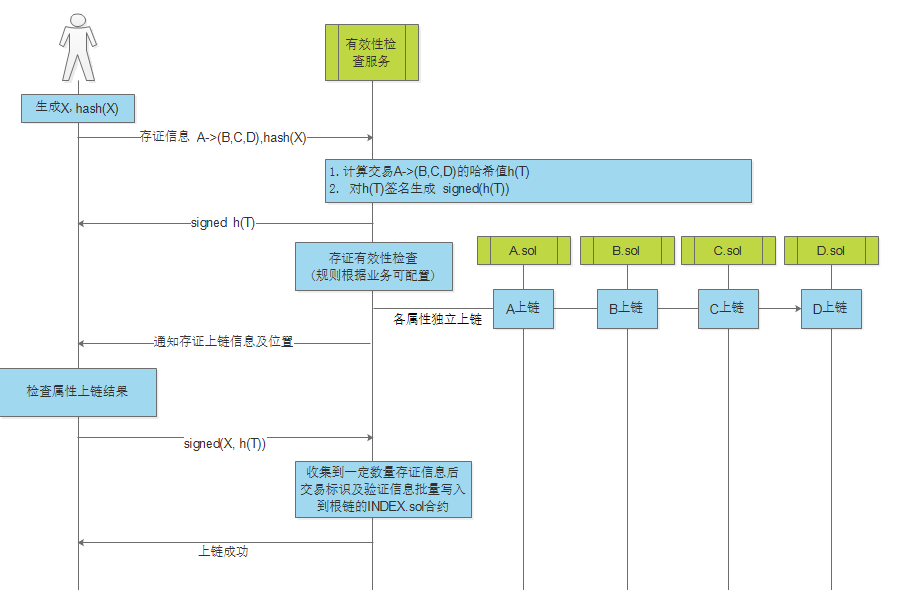
1. 对存证交易进行有效性,返回检查结果.
2. 存证信息批量更新到INDEX.sol合约.

* 用户:

通过SDK发起存证上链交易的实体,可以是第三方也可以是平台本身.

#### 方案说明

下面以一条存证上链的过程为例对流程进行说明.



存证上链流程如下:

1. 用户准备对存证A->(B,C,D)上链。
   1. 生成随机数X及其哈希hash(X)
   2. 发送存证信息及hash(X)给存证服务程序.
2. 服务程序获得存证信息后生成此条存证的哈希值h(T),此哈希值h(T)作为上链成功后此存证信息的唯一索引.

存证服务程序根据业务规则对存证属性进行拆分，检查属性有效性，然后对各属性分别上链。

属性处理完毕发送处理结果给用户，有以下两种情况：

所有属性上链成功，发送给用户处理成功标识及各属性位置；

属性上链失败，发送处理失败标识及原因。

两种情况用户均可以通过链上数据核实。

1. 用户收到属性处理结果后通过结果中的信息查询子链，对属性上链结果再次确认。确保上链信息无误之后发送由自己签名的X及存证信息标识hash(T)给存证服务程序。
2. 服务程序收到用户发送过来的确认信息后，需要：
   1. 检查用户签名，X，h(X)，h(T)的完整性。
   2. 上述检查通过后缓存此条信息用来后续批量更新到根链索引合约中。

索引合约更新方法如下:

存证服务程序收集一定数量的合约索引信息定期批量上链.

合约内需要对存证信息的“X,hash(X),hash(T),用户签名,平台签名”做一致性及完备性检查，检查通过的信息允许添加到索引合约中。

#### 方案特点

1. 保证存证依赖关系有效性。
2. 哈希锁方式使得根链合约中的存证数据必须得到双方认可，能够防止平台作弊，弱化了对平台可信性的依赖，对存证请求发起方来说实现了“上链即可信”。
3. 根链批量上链及多子链方式对平台性能做了扩容。
4. 针对tps要求高的属性，可对特定子链进行局部平行扩容。

### 方案对比说明及综合分析

方案一实现简单,适合不需要对存证做依赖性检查的场景，对平台可信性要求较高。

方案二能够对存证做依赖性检查,且用户能够对平台行为检查及授权，防止平台作弊。能够保证根链索引合约可信性，从而保证了存证可信性。

## 2.4 货币支付场景扩容方案

#### 1. 场景介绍

#### 2.支付通道

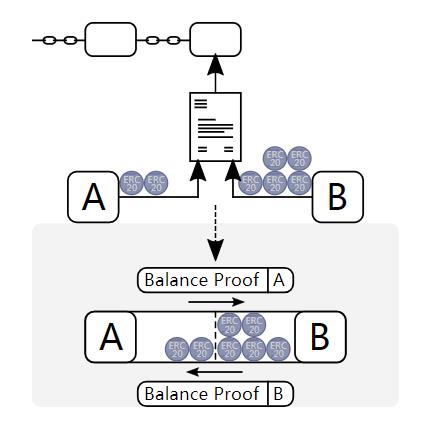
2.1

交易双方构建一个通道，将交易放在链下进行，双方各自保留带有双方签名的交易凭据，只有在需要关闭通道时，利用交易凭据将最终的参与者资产状态上链。

整个方案中主要包括两部分：用于通道管理和通道结算的智能合约和链下节点交互的节点网络(交易在节点网络上传输)。

方案的流程：

1. 参与者向智能合约注册打开一个指定对端参与者地址的通道，如果对端已经打开一个指向自己的通道，则直接加入。
2. 参与者双方在该通道上抵押一定的资产，资产由智能合约冻结
3. 参与者双方在链下建立一个直接相连的通讯渠道
4. 在链下，参与者任何一方在该通道上发起交易，可以选择direct交易或者HTLK(hash时间锁)交易，交易完成后参与者双方保留交易凭据，凭据包含nonce（交易序号）,发起地址，对端地址，转移资产数额，hash锁(direct交易不需要)及对应秘钥以及交易双发的对上面信息的签名。对于不能直接相连的参与者双方同样可以通过节点网络进行间接交易，每个间接交易由一系列有序发生的HTLK交易组成.
5. 关闭通道，结算并赎回各自最终的资产（即最终资产上链）。这里有两种方式：
6. 交易双方就各自最终的资产状态在线下进行协商确定，并生成一个带有双方签名的凭据，凭据确定双方最终的资产在该通道上的分配方案。各自向合约提交该凭据来关闭通道并赎回相应资产。
7. 交易的任意一方向智能合约发起关闭通道的操作，同时将自己保存的交易凭据传递给合约，此时另一方必须在结算周期来提交自己的交易凭据，智能合约校验双方凭据的有效性，并最终依据合法的凭据来确定通道资产的最终分配方案，对交易双发退还相应的资产。



2.2 优缺点

优点：

1. 支付速度快，能达到近即时

2. 低成本，支付花费少

3. 扩展性，参与者越多TPS越高

4. 隐私性，由于是链下交易，多次交易只对最终状态进行上链，外界无法知道交易详情

缺点：

1. 每建立一个通道都需要抵押一定的资产，每个用户不能建立太多的直连通道，因此大部分时候只能通过间接方式转移资产，对中间节点形成依赖，中间节点的稳定性影响整个交易过程。
2. 要求节点必须在线，而且中间节点在支付过程中掉线可能会造成自身经济损失。

每个通道每个转移的资产数额不能超过两个参与者抵押资产的总和，而且赎回资产过程复杂，花费时间较长，因此仅适合小额频繁支付。

3. 代理商机制

3.1

代理商机制的原理是将交易双方将一定的资产存入一个链上的代理商管理合约J中的资金池V，然后通过一个代理商进行支付，即A将资产支付给代理商O,代理商再将相应的资产支付给B,一个合法的交易凭证需要A,O,B三方签名或者A,B,J的三方签名，为了防止代理商作恶，代理商需要提交不少于资金池V的抵押金。且代理商停止服务赎回抵押金前有一个较长的等待时间，允许所有用户有足够的时间进行退出赎回操作。

方案的流程如下：

1. A将制定资产用hash锁加锁方式存入代理商管理合约AC中
2. 代理商O签发一个A的资产余额证明，并用AC的公钥加密后提交到AC上，AC解密证明后验证资产数额和签发对是否正确。
3. A将HTLK的secret提交给AC解锁之前的资产，AC验证secret正确后，将O签发的资产余额证明通过事件发给A,同时将资产放入资金池V中.
4. A在链下向O发起一笔向B的转账交易，额度为x。O签名交易后返给A,并同时等待B上线后将其发送给B，B最终签名交易后形成一个有效的三方签名的proof,并将proof返给O,O再返给A，B在赎回资产时需要该proof。
5. A退出代理商赎回资产时，需要提交O签名的资产余额证明，A的所有支出交易列表和A的所有进账proof.





Balance proof：证明用上次结算后的余额状态。每次余额计算方式为

Balance prooflast – Sum (TXs )+ Sum(Proofs)

用户退出代理商赎回资产需要提供balance proof+ TXs(未计算支出)+Proofs(所有收入凭据)，为了防止任意一方重复提交已经结算过的TX或Proof，每次结算都记录本次结算的所有TXs和Proofs的merkle hash.一旦发现一方提交重复的TX或Proof,只需要提交对应TX和proof对应的Balance Proof 以及其merkle路径即可反驳。

TX：每个转账交易发起者A签名一个Tx,代表一笔资产的支出，经过代理商O和接收者B签名后，Tx转变为B的一个交易证明，B最终签名后建议将证明返回给O,O在返回给A。为了防止代理商O隐瞒Tx，A在退出代理商时应该将所有没有B签名的TXs提交给合约AC,由AC签名替代O的签名，并以事件进行广播。如果此时B已经退出代理商，则直接TXs对应资产转入B的账号。

Proof: 交易有效的证明，必须集齐三方签名，SignA+SignO+SignB或者SignA+SignAC+SignB

优缺点：

优点：

1. 线下交易，拥有支付通道类似的TPS性能。
2. 因为一次抵押可以和所有其他用户交易，可以支持相对大额交易。
3. 交易对端可以不在线。

缺点：

1. 代理商的抵押金数额限制代理商的服务能力
2. 对于代理商恶意隐瞒交易的情况，交易对端只有在发起者退出网路后才能知晓。

4. 方案对比说明及综合分析

支付通道和代理商机制都是讲链上交易转到链下，最终结算才将状态进行核对上链，均支持较高的TPS.区别是代理商机制支持相对大额交易且允许交易对端不在线。但相对的为了保证资金的绝对安全，代理商需要交纳大量的抵押金，且抵押金的数额决定其服务能力。

5. 开放问题

1. 支付通道在交易者掉线情况下，可能会造成经济损失
2. 代理商机制为了绝对的资金安全，需要代理商交纳相对数额较大的抵押金，抵押金限制了其服务能力。另外为了提高考虑代理商的服务意愿，应当有相应的奖励机制。
3. 考虑到代理商抵押资金有限的情况，可以扩展多代理商机制，支持跨代理商交易，以及类似银行间结算机制的代理商结算机制。更进一步，将代理商机制结合跨链转账技术，实现跨链的高TPS支付。
4. 代理商机制对隐瞒交易的纠正必须发生在发起者退出后，这点需要考虑合理性，是否需要有相关机制进行补充解决。

## 2.5 供应链金融场景扩容方案

#### 1. 场景介绍

供应链下游企业因为自身资质条件的原因，没有足够的信用度，也缺乏相应的背书机构背书，经常会遇到融资困难。相反上游企业没有这些问题，因此可以通过上游企业发行指定数额的票据资产，由上游企业或其他诸如银行或其他可信机构背书，上游企业将票据作为订单的费用分拆支付给不同的下游企业。因为有足够信力的机构背书，下游企业利用这些票据资产当做一种数字金融资产进行买卖，通过卖出来进行融资。这种票据的发行和买卖行为与证券交易一样，单位时间发生的交易数量大，现有区块链的TPS满足不了该需求。

2. 独立多链

2.1 <方案具体说明>

将票据设计为一种满足ERC20 标准的数字资产，同时将不同的票据发行和交易放在不同的链上，依据单个链的TPS能力，一个链承载一个或多个不同票据。

方案包括一个访问层和状态缓存。

状态缓存主要保存所有账户在各链上拥有的票据资产状态。缓存更新由票据事件触发。

访问层对接用户和区块链，为了保证每个票据在链上的TPS,由访问层限制单个票据买卖的TPS,保证某些高频的票据不会影响其他票据的TPS.



2.2 <方案优势及特点分析>

1. 由于票据之间完全独立不相关，而且单个票据的买卖对TPS要求不高，对比证券交易，一般单个TPS在100以内，对ibft共识每秒接近1000 TX的效率完全可以满足

2. 具有几乎无限的可扩展性，每个新增的票据带来的TPS诉求可以通过部署一条新的链来满足。

---- END ----