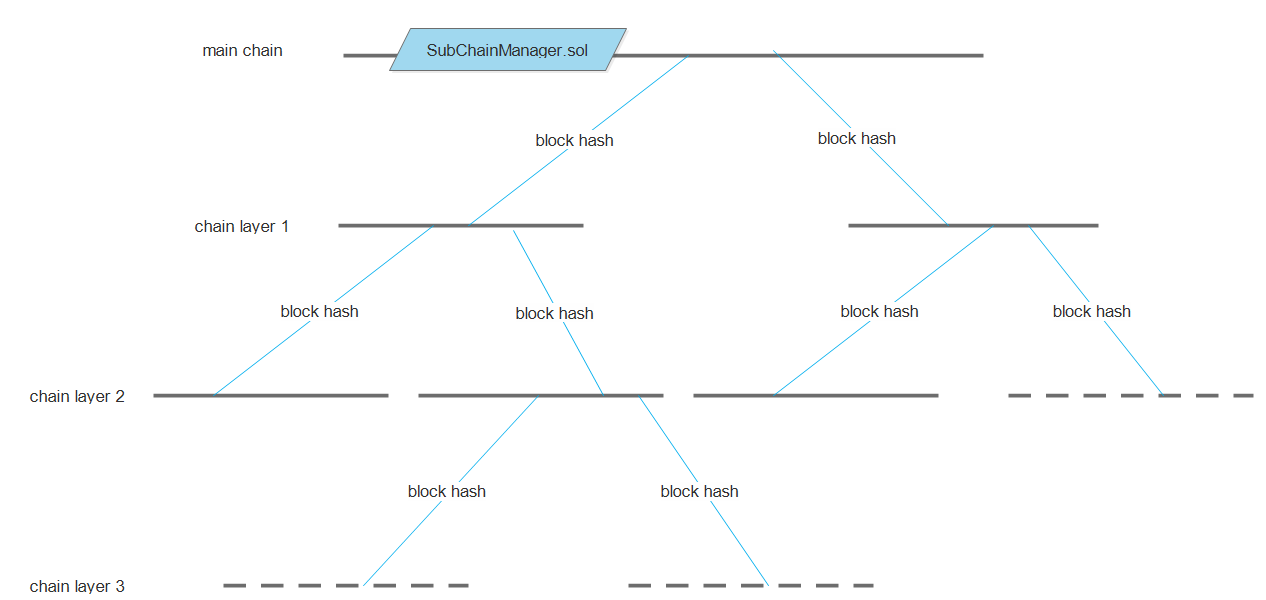
# lanchain扩容方案分析

## 总体介绍

综合对各种扩容方式的分析，从原理及实现复杂度的角度考虑，采用多个子链的扩容方式对lanchain平台进行扩容，以提高lanchain平台各场景下的业务处理能力。

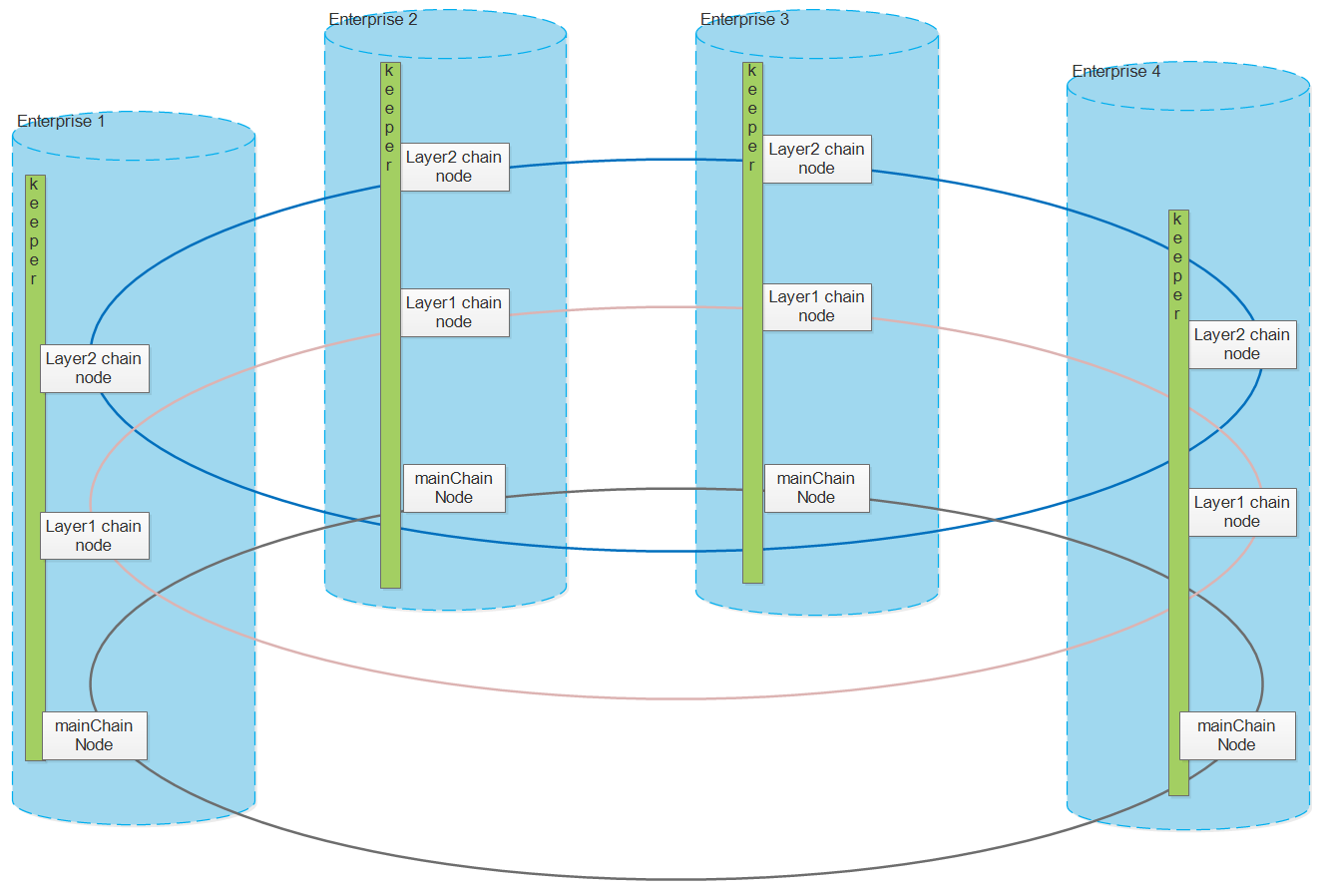
在此扩容方案中，多个子链按照树形结构进行组织。联盟中的各validator按照共识协议达成共识之后能够完成子链的创建及删除。

图一展示了子链树形结构。



图一. 子链树形结构示意图

图二为实际的拓扑场景示意图。



图二. 实际拓扑场景示意图

### 树形结构子链的优势

树形结构在转账交易中的作用参考Plasma原理实现.

此部分着重说明存证上链场景中采用属性结构的优势.

* 多层树形子链结构可以通过增加层数进行快速扩容,也可以通过在同一层增加子链数量进行线性扩容,结构结构灵活.
* 单层子链结构部署相对容易,但是只能进行线性扩容,如果子链数量较多也会对主链造成压力,影响系统整理性能.
* 虽然多层树形子链结构从实现上来说较单层子链结构复杂，但是在实际部署时可以根据具体需求退化成单层子链结构。
* 树形子链结构分支根据需求划分成不同的功能域,方便维护管理.
* ~~采用Merkle树存储子链块hash,能够快速验证子链完整性~~,加快新节点数据同步速度,同时节点可以按需同步部分子链,而不需要同步全部子链上的数据.
* 子链不同分支运行不同DAPP,根据需求灵活配置分支参数.

在下述章节会对本部分内容进行详细说明.

## 子链管理

子链的创建，销毁及子链管理的所有工作都在主链上进行,由“智能合约 + DAPP”来完成。

SubChainManager.sol智能合约部署在主链上，负责子链的创建,销毁及各种管理工作.是子链管理的核心合约.

智能合约中的主要角色如下：

Owner: 创建智能合约的人，通过一定条件owner角色可以转移给其他用户。

Validators：创建子链过程中的验证者，只有validators列表中的所有用户通过共识算法达成了共识，子链才能被创建或者销毁。现有validators可以投票添加新的validator及删除现有validator.

Consensus：共识算法类型，owner可以修改使用的共识算法，validators按照共识算法达成共识来决定子链的操作行为。

SubChainTree: 维护子链的树形结构关系.

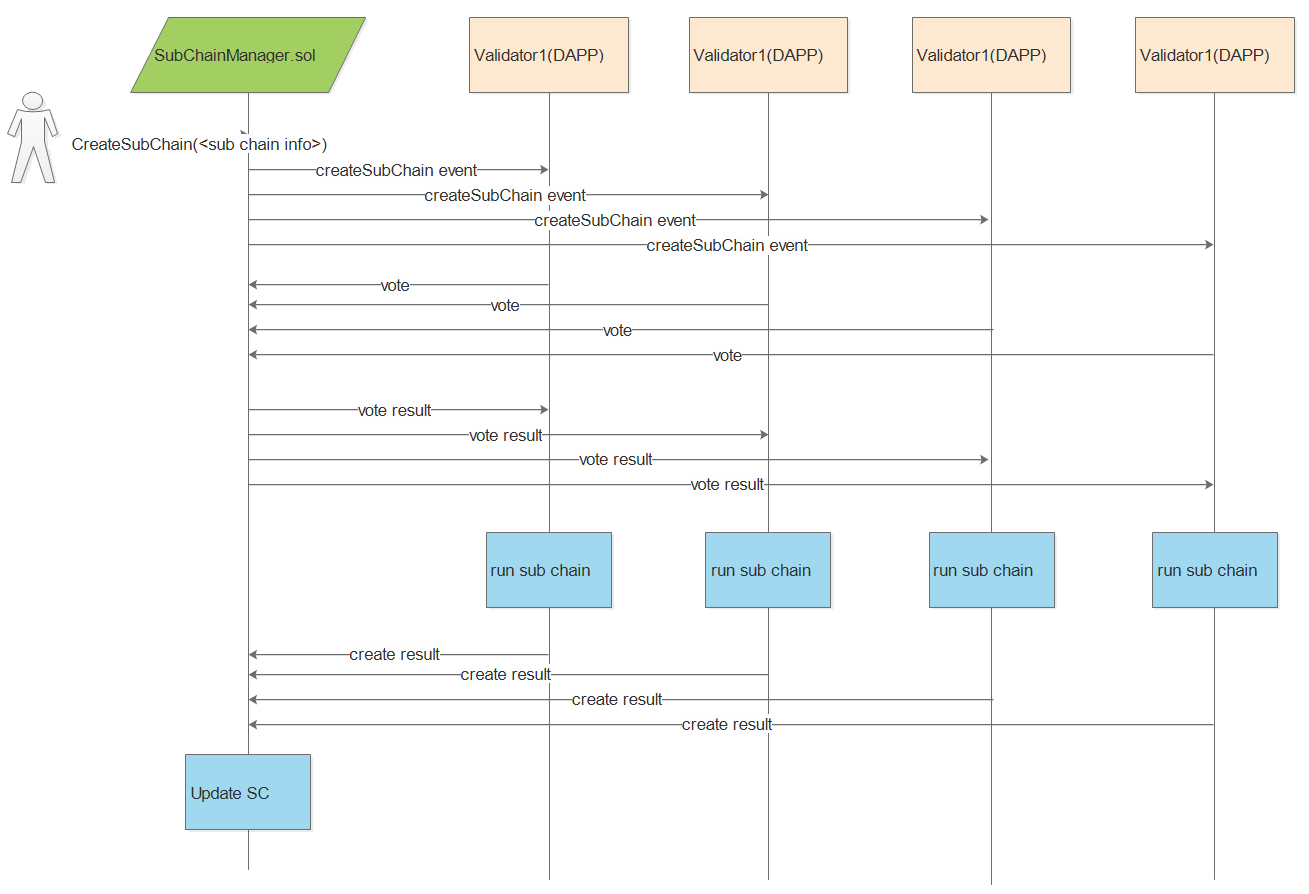
其他内容待细化。

### 子链创建

子链创建过程示意图如下:

(子链销毁过程与子链创建过程相同.)

(Plasma运行在公链中时,在公链环境下没有相对可靠的节点,因此需要激励机制鼓励用户参与到私链的验证中来.而在联盟链中,有相对可信的节点,因此可以通过目前已经应用于联盟链中的算法来达到各节点之间的共识,不需要激励机制.)



图三. 子链创建示意图

在图三中,子链的创建由主链上的各validator所在节点来负责,此种方案可以保证对子链的有效管控. 在实际部署过程中,也可以考虑任意用户都可以创建子链.

“任意用户均可创建子链”的方案需要考虑如下问题:

1. 运行在公链环境下,由任意用户创建的子链可以无任何限制接入到根链拓扑中.用户在子链上的操作不会发给wan coin,只有在块hash同步到wanchain时才会发费wan coin.
2. 运行在联盟链环境下,由任意用户创建的子链需要满足一定条件,并由联盟的各validators验证通过之后才能加入到根链的拓扑中.

此种子链部署方式理论上能够使得整个子链树无限扩展，而且因为有validators验证机制的存在，在联盟链的应用场景中，子链的扩展更安全可信。

### 子链销毁

与子链创建流程相同,参见图三.

### 共识方式

* **链上共识**

链上共识即所有validator把结果写入到智能合约中，共识结果在智能合约中达成并通过事件广播共识结果给各validators。

链上共识实现较链下共识简单，但是个validator均需通过交易的方式提交结果给智能合约,达成共识时间较长。

子链的创建及销毁不会频繁发生,不需要考虑效率.

* **链下共识**

链下共识即共识过程完全在链下通过DAPP达成，达成结果后把经过签名的共识结果上链。

链下共识达成共识的速度较链上共识方法快，但是需要DAPP支持各种共识算法，实现上较链上共识复杂。

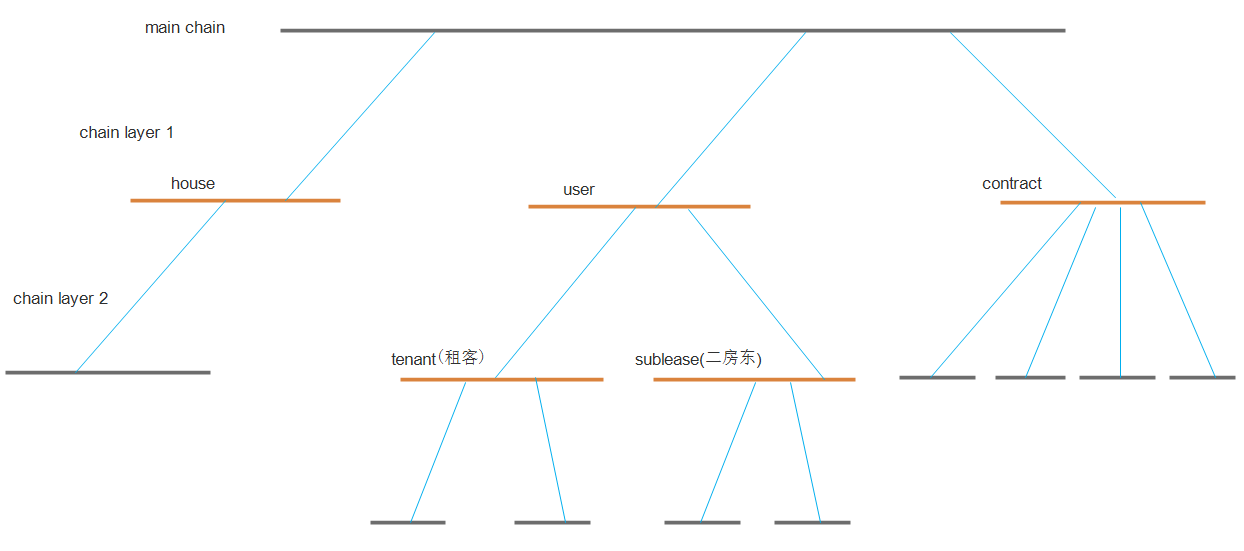
推荐采用”链上共识”.

### SDK相关

平台SDK需要修改,SDK需要支持以下功能:

* + - 子链管理(创建,删除)
    - 能够按域对子链操作,对于同一个域的子链在交易上链的时候能够自动进行负载均衡.
    - 用户在实际操作的时候对子链是无感知的.

## 存证上链场景



图四. 存储存储结构

在存证上链场景中有三种链类型(参考Plasma模型):

1. Root chain: 根链, 负责block hash的最终上链,可以是联盟链的根链也可以是wanchain.
2. Parent chain: 父链, 子链的块hash存储在父链上. 每个父链有其具体名称,在主链的”子链管理合约”中维护的子链树形关系中存储了父链名称及其访问方法(ip,端口等).
3. Entity chain: 实体链,真正存储需要上链的存证数据. 一个parent chain可以关联多个entity chain, 一个entity chain只有一个parent chain.

存证的ChainURI:

通过ChainURI能够唯一定位一个存证信息, ChainURI的格式如下:

ChainURI://house/0x1234567890

ChainURI://user/tenant/id:1234567890 & tel: 13910768493

ChainURI://contract/path1/path2/path3/path4/0x99999999

等.

存证的ChainURI类似于传统的URL格式,此部分的实现可以参考EEA spec中的ENS部分, 目前ENS已经有实现方案(https://ens.domains, 此实现尚未具体分析,后续需要分析此实现方案及查找是否还有其他实现方案), 如果ENS适用于此存证方案,那么可以考虑把ENS部分作为一个独立生态组件实现,并逐步扩展到更多应用场景.

ENS模块中可以考虑基于中心化方案实现,可以通过对各存证实体进行缓存,优化查询路径等方式加快存证的查找及确认过程.

### 单一存证上链

* **单子链存储**
  + 参考图四中house域
  + 域下只有一条子链用于存储存证.
* **并发多子链存储**
  + 参考图四中contract域
  + 在contract域下有4个子链,当对contract做存证的时候由SDK保证存证在4个子链中轮询以平衡负载,增加并行效率.
  + 用户并不能感知到数据的存储方式(在单链存储还是在多链存储),由SDK提供封装性及并发性的保证.
  + 需要提供一种方法在存证查询的时候能够定位到存证所在的链(比如说通过ID hash映射子链等方式,此种映射方法很多,下一步考虑具体实现方法.)
  + 对于大用户量的并发行为”并发多子链存储”的方式能够在多个子链上合理均衡负载,达到支持大用户量并发的目的.

### 关联性存证上链

关联性存证即一个存证上链的前提是其引用的存证已经在链上存在,如果被引用存证不存在的话则存在引用缺失的风险.

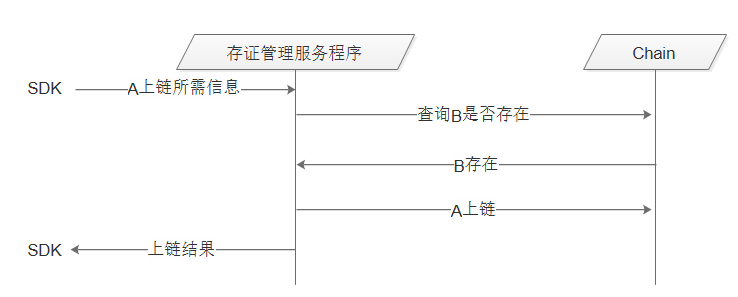
关联性存证上链分两种情况进行分析:

1. 同步关联性存证上链,下述case1,case2,case3为同步方式的存证上链.
2. 异步关联性存证上链,下述case4为异步方式的存证上链.

例如: A->B (存证A上链首先需要B在链上已经存在)

**Case1:**

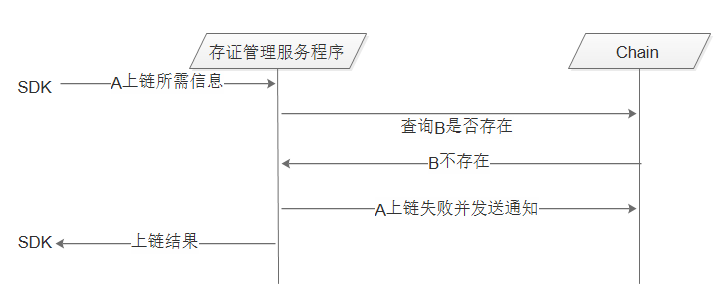
A->B: A上链,前提B上链



图五. B在链上已经存在

**Case2:**

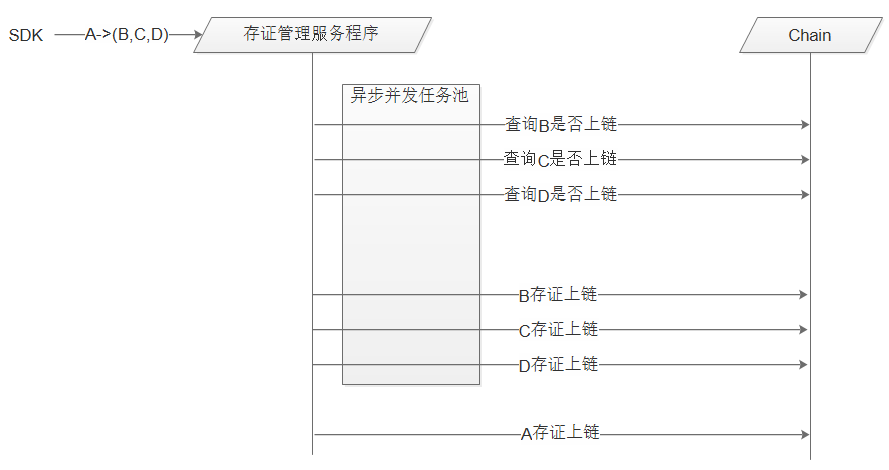
A->B: A上链,B在链上不存在



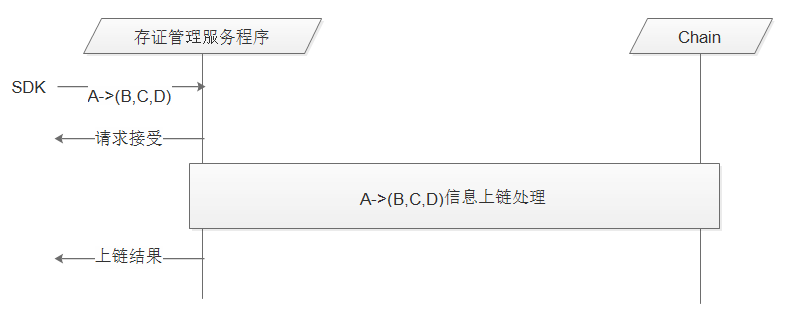
图六. B在链上不存在

**Case3:**

1. >(B,C,D): A上链,前提要求B,C,D均已经上链(如B,C,D没有上链,那么把B,C,D进行上链)



**Case4:**



此异步方式对单一关联性存证上链信息的效率没有显著提高,但是对整体tps要求较高的场景有整体性能的提升.

### 大文件存证上链

大文件存证上链方式:

* IPFS/ ethereum SWARM分块分布式存储,merkle树上链.
* 中心化存储,hash上链.
* 数据分块上链,merkle树上链.

### 性能分析

#### 单一存证上链性能分析

单一存证上链场景中如果采用一个实体链,则性能与扩容前相同.

多个同质实体链可以提高单一存证上链的并发程度,提高倍数与实体链数量成正比.

*扩容后平台整体TPS = 扩容前tps \* 实体链数量*

#### 关联性存证上链性能分析

关联性存证上链场景中,所有被关联存证均已在链上存在,此时整条上链信息上链时间为:

1. *>(B,C,D))上链时间 = A上链时间 + B,C,D并发查询时间*

在最坏情况下,所有被关联存证均没有上链,则此时整条信息上链时间为:

1. *>(B,C,D))上链时间 = A上链时间 + B,C,D并发上链时间*

上述时间的范围为:

*单一存证单链上链时间 <= (A->(B,C,D))上链时间 <= 2 \* 单一存证单链上链时间*

不采用此扩容方案,则(A->(B,C,D))上链时间为:

*扩容前(A->(B,C,D))上链时间 >= 2 \* 单一存证单链上链时间*

综上,从原理上证明了扩容方案对单一关联性存证上链能够较扩容前提高上链速度.

**异步方式:**

没有采用异步方式,关联性存证的最优上链时间为单一存证单链上链时间.即上链10笔交易的最优时间为 10\*单一存证单链上链时间.

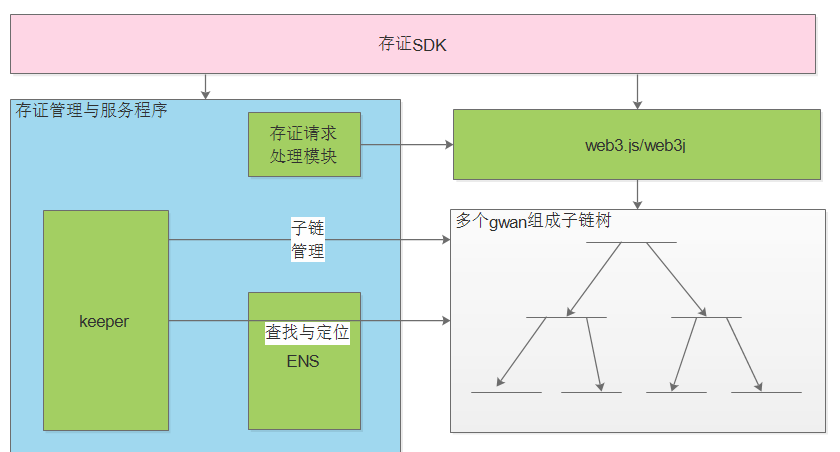
采用并发操作,10笔信息上链的时间最优时间可以与单一存证单链上链时间相当.

综上,异步方式能在同步方式的基础上进一步提高性能.

**中心化:**

在此方案中,对部分操作及模块以中心化的方式运行也能够对性能有所提高.

## 整体模块组织形式



---- END ----

问题:

1. 存证查找方式及性能. 每个存证的URL能够定位一个查找范围.
2. 关联存证查询的事务性问题.
3. 链之间的数据依赖性.
4. 转账交易:
5. 数据存证是否需要关联性检查?
6. 数据存证只需要关注存证hash,不需要存证真正上链.
7. 权限数据,业务规则与数据分开.
8. 拍卖
9. 上链