金融服务平台

——数据区块链存储

软件设计说明书

1. 引言

1.1 编写目的

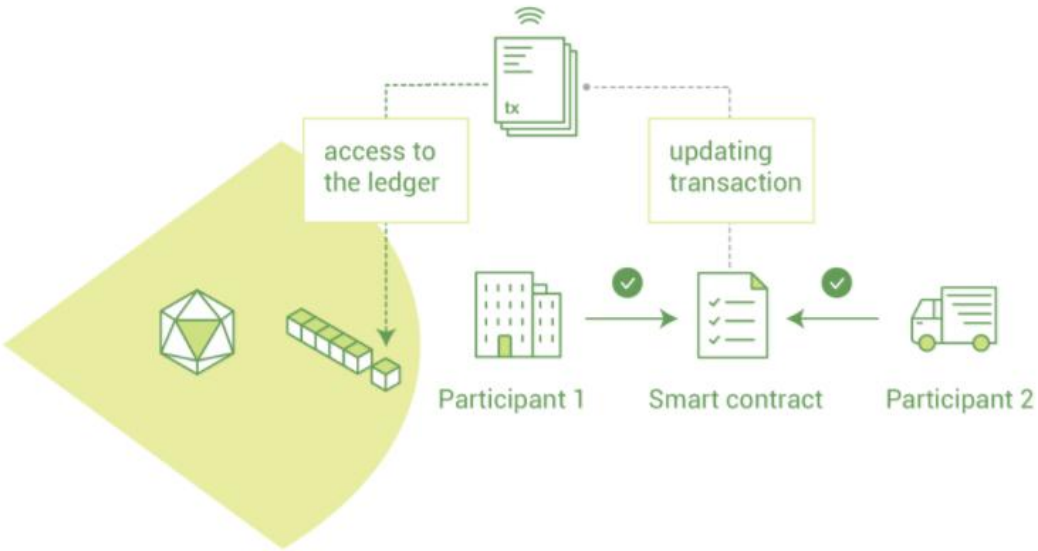
本说明书为金融服务平台的数据区块链存储模块编写。本说明书明确了软件开发涉及到的框架设计、工程架构和软件功能要求。

本说明书可以作为项目前期规划任务时的指导，也可以作为项目后期分析错误、评估需求完成度以及检查系统正确性的参考文档。

1.2 背景

利用区块链技术构建的金融服务平台的的核心是一个分布式的账本。该账本记录区块链网络中发生的所有交易的详细信息。

利用区块链技术所建立的账本经常被描述为去中心化的。区块链网络中的每个节点都保存着一个副本，所有节点通过更新副本和相互通信共同维护着账本。



除了去中心化的特性，区块链的另一个显著特性是信息只能以附加的方式记录在区块链中。同时，区块链还使用特定加密技术，使得交易一旦产生其内容便无法被修改。区块链的不可修改性使确认信息源变得容易，因为信息一旦产生其源头便是确定且不可被修改的。

以上的优点使区块链技术日益火热，由于区块链技术的这些特性的特殊性，该技术逐渐被用于支持金融领域的发展。

本模块即是利用区块链技术对交易信息进行入链操作。

1.3 定义

Anchor Peer - 锚定对等节点

锚定对等节点(锚节点)本质上是一个已经加入了某个管道组织的主要用于发现其他对等节点的节点——在一个管道中，锚节点可以被这个管道的其他任何节点发现和通信。管道中的所有组织都有一个或多个锚节点，一个组织可以通过锚节点来发现管道内其他组织的所有节点。

Block - 区块

区块即由密码学串接并保护内容的交易记录，每一个区块都包含了前一个区块的加密散列、相应时间戳记以及交易数据。这种块链中的第一个区块称为起源区块，区块由命令系统创建，并且由对等节点验证。

Chain - 链

分布式账本的链是一个交易日志，其结构为由哈希连接的交易块。对等节点从命令服务接受交易区块，根据认可策略与并发冲突将区块的交易标注为有效或无效，并将区块追加到对等节点的文件系统上的哈希链中。

Chaincode - 链码

HyperLedger Fabric 中的智能合约就是区块链的代码，也就是所谓链码链码有区块链网络外部的客户端应用程序调用，主要用于管理对世界状态中一组键值对的访问和修改。

Channel - 通道

通道是一个对私有区块链的覆盖，它使得数据能够保持隔离和保密。通道特定的账本在通道中是与所有对等节点共享的，并且交易方必须对通道进行适当的身份认证才能与账本进行交互。通道是由一个配置块来定义的。

Commit - 提交

一个通道中的每个对等节点都会验证有序的交易区块，然后将区块提交（写入

或追加）至该通道上的账本副本上。对等节点还会将每个区块中的每笔交易的状态标记为有效或者无效。

Concurrency Control Version Check（CCVC） - 并发控制版本检查

CCVC 是一种在通道上保持同步状态的方法。对等节点并行的执行交易，并且，在交易提交至账本之前，对等节点会检查在执行时读取的数据是否未更改。如果读取的数据在执行时间和提交时间之间被更改，那么会发生一个 CCVC 冲突，该交易就会在账本中被标记为无效，并且状态数据库中的值不会更新。

Configuration Block - 配置区块

配置块也是一个区块，这个区块中包含了定义系统的链或通道的成员与策略的配置数据。对某个通道或整个网络的配置修改（比如，成员离开或加入）都将导致生成一个新的配置区块并追加到适当的链上。这个配置区块会包含创始区块的内容加上增量。

Consensus - 共识

共识是整个交易流程的一个术语，用于生成订单协议并确认构成区块块的交易集合的正确性。

Current State - 当前状态

见世界状态（World State）。

Dynamic Membership - 动态成员

Hyperledger Fabric 支持添加/删除成员，对等节点和命令服务节点，而不会影响整个网络的可操作性。当业务关系调整并且出于各种原因需要添加/删除实体时，动态成员至关重要。

Endorsement - 认可

Endorsement 指特定对等节点执行链码交易并将请求响应(proposal response)返回给客户端应用程序的过程。请求响应包括链码的执行的响应消息、结果（读取集和写入集）和事件，以及用作对等节点链码执行的证明的签名。 链码应用程序具有相应的认可策略，其中指定了认可的对等节点。

Genesis Block - 初始区块

用于初始化订购服务的配置块，或用作链上的第一个块。

Gossip Protocol - Gossip 协议

Gossip 数据传播协议执行三个功能：1）管理对等节点的发现和通道成员; 2）在渠道的所有对等节点中传播分类帐数据; 3）在通道上的所有对等节点上同步账本状态。

Hyperledger　Fabric-CA

Hyperledger Fabric-CA 是默认的证书管理组件，它向网络成员及其用户颁发基于 PKI 的证书。CA 为每个成员颁发一个根证书（rootCert），为每个授权用户颁发一个注册证书（eCert），为每个注册证书颁发大量交易证书（tCerts）。

Init - 初始化

一种初始化链码应用程序的方法。所有链码都需要一个 Init 函数。默认情况下，永远不会执行此功能，但是可以使用链码定义来请求执行 Init 函数以初始化链码。

Instantiate - 实例化

在特定通道上启动和初始化链代码应用程序的过程。实例化后，安装了链代码的对等方可以接受链代码调用。此方法用于以前版本的 chaincode 生命周期。对于当前用于在通道上启动链码的过程，新的 Fabric 链码生命周期作为 Fabric v2.0 Alpha 的一部分引入

Invoke - 调用

用于调用链码函数。客户端应用程序通过向对等节点发送交易请求来调用链码。对等节点将执行链码并向客户端应用程序返回已批准的请求响应。客户端应用程序将收集足够的请求响应以满足认可策略，然后将提交交易结果以进行排序，验证和提交。客户端应用程序可以选择不提交交易结果。譬如，如果该调用仅查询账本，除非需要在账本上记录该读取记录以进行审计，否则客户端应用程序通常不会提交只读性质的交易。调用包括：通道标识符，要调用的链码函数和参数数组。

Leading Peer - 主导节点

每个组织可以在他们关联的每个通道上拥有多个对等节点。这些对等节点中的

一个或多个应该充当该通道的主要对等节点，以便代表该组织与网络命令服务进行通信。命令服务向通道上的主要对等节点提供区块，然后将其分发给同一组织内的其他对等区块。

Ledger - 账本

账本由两个不同但相关的部分组成：区块链和状态数据库（见 World State），与其他分类账本不同，区块链是不可变的，一旦将区块添加到链中，它就无法更改。

Membership Service Provider（MSP） - 成员服务提供程序

成员服务提供程序（MSP）是指系统的一个抽象组件，它为客户端提供凭据，并为其提供参与 Hyperledger Fabric 网络的对等节点。客户端使用这些凭据来验证其交易，并且对等节点使用这些凭据来验证交易处理结果（认可）。该接口与系统的交易处理组件进行强连接。以这样的方式定义成员服务组件，可以在无需修改系统的交易处理组件的核心的情况下，无冲突地插入成员服务组件的替代组件。

Membership Services - 成员服务

成员服务在许可的区块链网络上对身份进行身份验证，授权和管理。在节点和 orders 中运行的成员服务代码都对区块链操作进行身份验证和授权。它是基于 PKI 的成员服务提供程序（MSP）抽象实现

Ordering Service - 排序服务或共识服务

又称为 orderer，是一个被定义的交易集合，用于将交易排序到块中。排序服务独立于对等流程而存在，并以先来先服务的方式为网络上的所有通道排序交易。排序服务旨在支持开箱即用的 SOLO 和 Kafka 品种之外的可插拔实施。排序服务是整个网络的通用绑定;它包含与每个成员关联的加密身份资料。

Proposal - 请求

针对通道中特定对等节点的认可请求。每个请求都是 Init 或 Invoke（读/写）请求。

Query - 查询

查询是一个链码调用，它读取账本当前状态但不写入账本。

State Database - stateDB

当前状态数据存储在状态数据库中，用于从链码进行高效效的读取和查询。支持的数据库包括 levelDB 和　　。

System Chain - 系统链

包含在系统级别定义网络的配置块。系统链位于订购服务中，与通道类似，其初始配置包含以下信息：MSP 信息，策略和配置详细信息。对整个网络的任何改变（例如，新的组织加入或正在添加的新订购节点）将导致将新的配置块添加到系统链。

Transaction - 交易

每当使用 chaincode 或 FabToken 客户端从账本读取或写入数据时，创建一个交易。

World State - 世界状态

表示区块链日志中包含的所有键的最新值。

1.4 参考资料

《软件工程实用教材》（吕云翔）。

核心技术说明：维基百科，Hyperledger Fabric 官方文档。

2. 总体设计

2.1 需求规定

与综合运营平台系统进行数据和信息的交互，接受转账、提现等信息，实现上述信息的入链功能和查询功能。在转账、提现操作完成之后，对应的信息进入区块链中储存，且不能被修改，以供需要查询时使用。

2.1.2 系统性能

2.1.2.1 精度

由于金融行业的特殊性，要求尽可能实现信息 100%的准确性。

2.1.2.2 时间特性要求

能够在尽可能短的时间内完成入链或查询操作，尽可能缩短交易或查询所花费的时间。

2.1.2.3 可靠性

要求系统具有高可靠性，有效地处理各种异常情况，并正确地记录数据。

2.1.3 输入输出要求

为提高数据传输的效率，输入输出的数据交换格式均为 String 格式。输入和输出的内容必须包含识别 ID 以及被加密过的信息。详见 5.1 用户接口。

2.1.4 数据管理能力要求

应保证数据的完整性、保密性、准确性和有据性，防止数据的损坏、泄露、篡改、破坏等。同时，由于金融领域的行业特性，可能还需要数据管理系统的冗余能力和快速响应能力。

2.1.5 故障处理要求

系统应具备故障处理、系统恢复、数据恢复等能力。

2.2 运行环境

该项目运行环境对硬件设施要求适中，但需要一定的软件支持基础。

2.2.1 设备

本项目采用 PC 端开发：

- 处理器型号：Intel 2.0GHz 及以上。
- 内存剩余空间：4G 及以上。
- 外存剩余空间：20G 及以上
- 网络配置：1000M 网卡，串口。

2.2.2 支持软件

- 操作系统：MacOS X, *nix, Windows 10
- Docker: 17.06.2-ce 及以上版本
- Docker Compose: 1.14.0 及以上版本
- Go: 1.9.x 及以上
- Node.js: 8.9.x
- npm: 5.6.0 及以上版本
- 数据库: CouchDB

2.2.3 控制

来自运营平台的消息队列（Message Queue）对本模块的运行进行控制。消息队列是多条消息组成的队列，每条消息可能是入链请求、信息修改请求或是查询请求中的一类。消息的类别决定了模块的行为。

2.3 基本设计概念和处理流程

2.3.1 基本设计理念

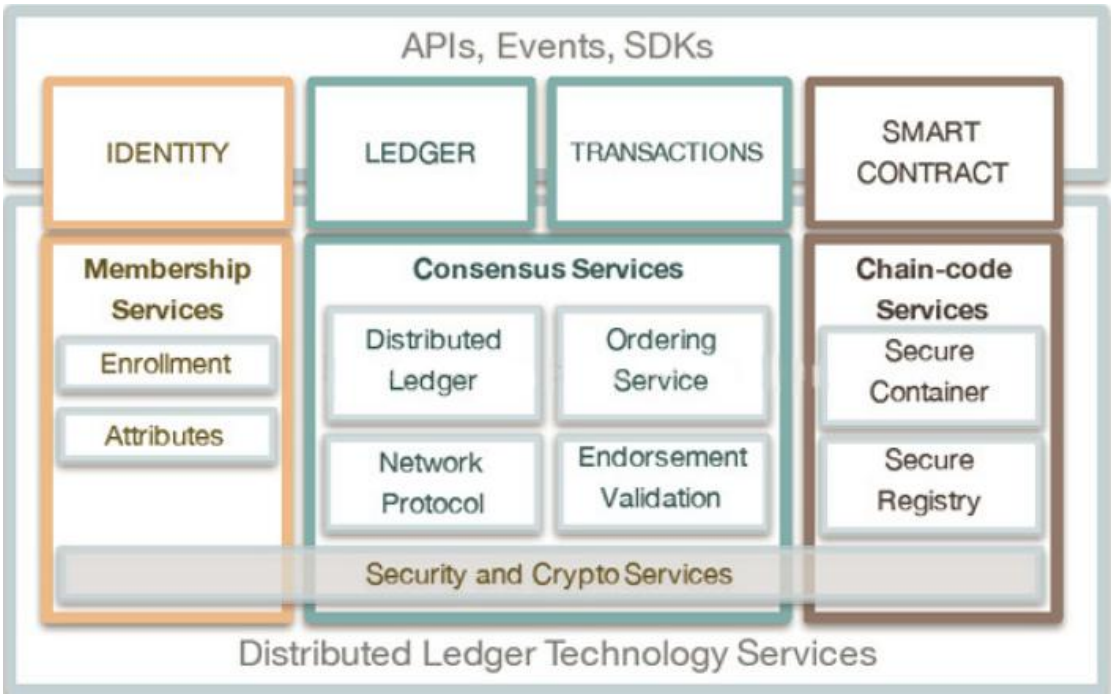
使用 Hyperledger Fabric 框架进行设计开发。在需求确定后，明确基本的核心功能，再对各个部分功能进行分批次地分析、设计、编码，实现迭代式的敏捷开发过程。

2.3.2 处理流程

搭建底层节点(Peers)
实现业务逻辑链码(Chain code)
链码层基础上搭建应用
实现接口
包装接口为服务并注册到系统后端
实现上层查询、修改
与其他组信息传输

2.4 结构

2.4.1 Hyperledger Fabric 架构



Hyperledger Fabric 架构的核心包括三部分：

Identity, Ledger & Transactions, Smart Contract，

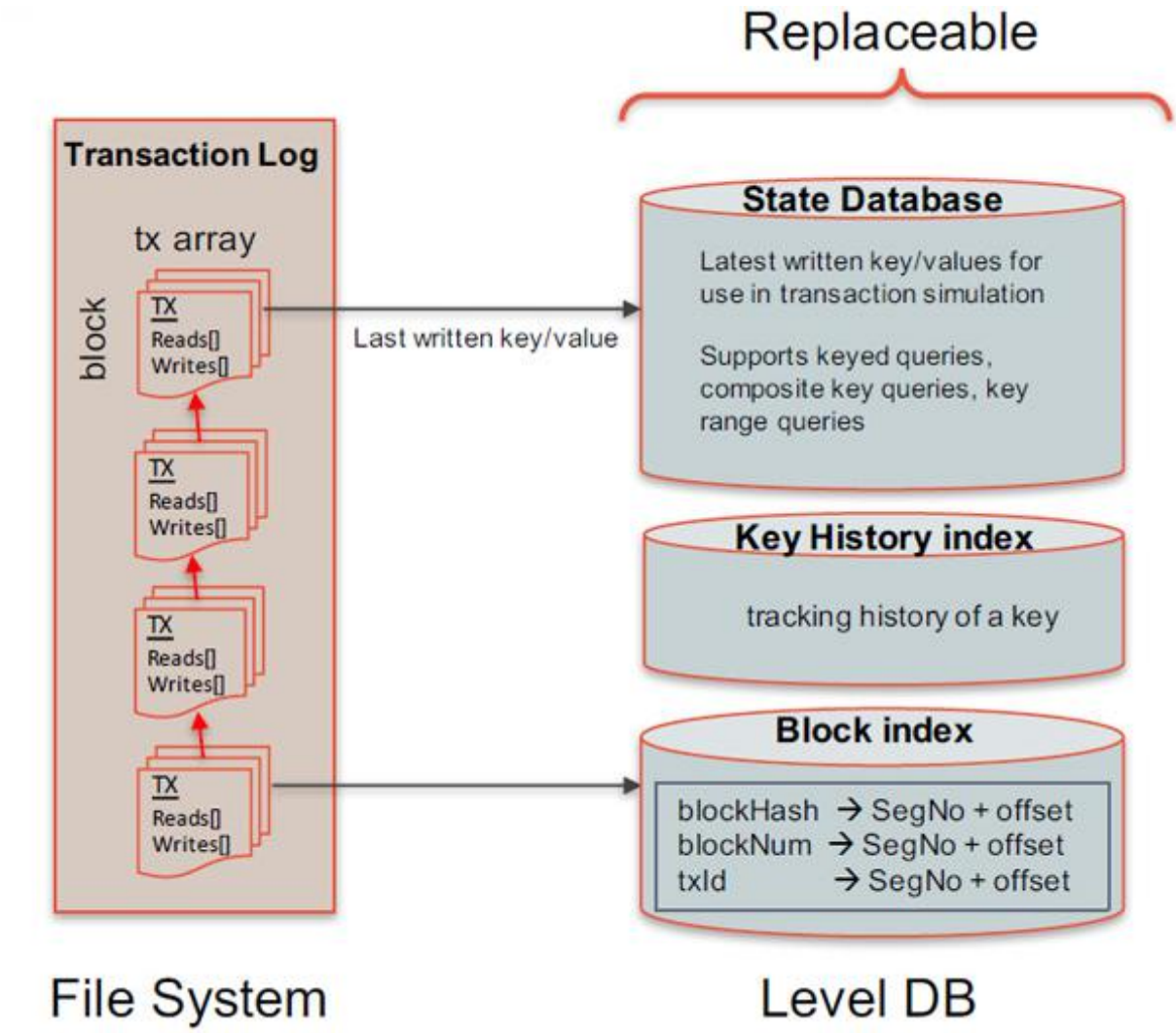
分别负责身份管理、区块链存储、业务逻辑的实现。

• Identity

Identity 提供成员服务,在有许可的区块链网络上对身份信息进行认证、授权和管理。在 peer 和 orderer 中运行的成员服务的代码会为区块链操作进行认证和授权。它是基于 PKI 的 MSP 实现。

• Ledger & transactions

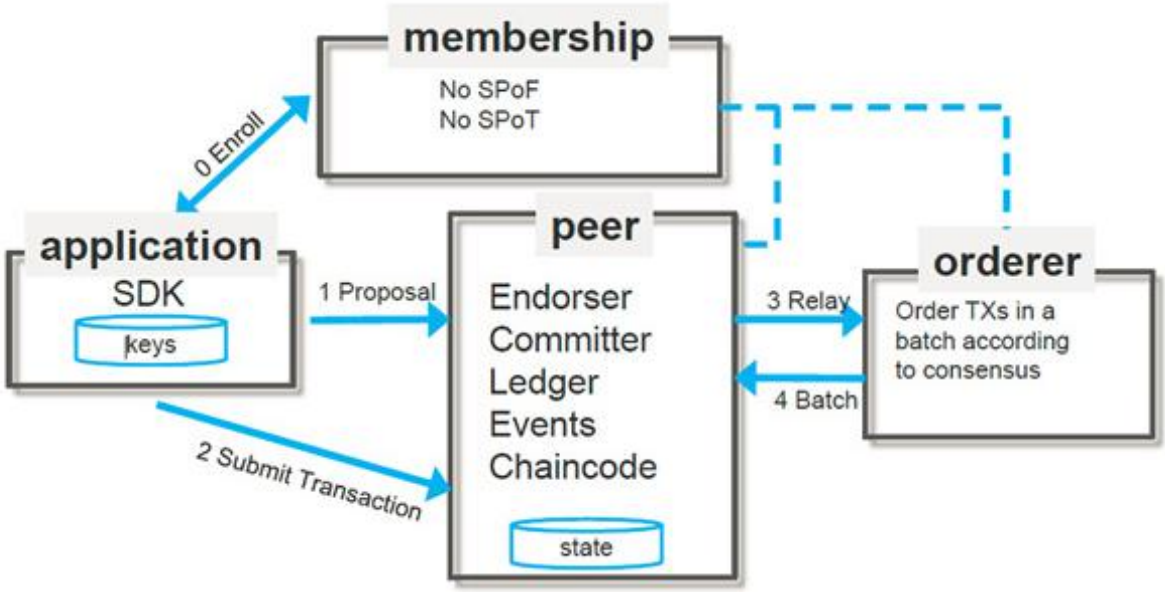
Ledger 是由 channel 的 chain 和 channel 中每个 peer 维护的 world state。如下图所示。chain 记录历史交易。state 对应账本的当前最新状态，它是一个 key-value 类型的数据库。



- Smart Contract

Smart Contract 是一个运行在区块链账本上的软件，它对链上的资产进行编码。

2.4.2 Hyperledger Fabric 共识网络



共识是使用于交易过程的一个广义术语。它代表了对于排序的同意书和确认构成区块的交易集的正确性。其中，peer 是一个网络实体节点，它维护 ledger 并运行 Chaincode 容器来对 ledger 执行读写操作。orderer 节点则提供排序服务。peer 和 orderer 节点共同由 Member 拥有和维护。

在实际应用时，程序首先触发一个提案，endorser 收到提案并验证合法后，SDK 验证背书节点的签名，并判断各节点返回的提案结果是否一致且按照指定的背书策略执行。客户端在收到应答后，打包组成一个交易并签名发送给 orderer。而 orderer 对其进行共识排序，然后按照区块链生成策略，将一批交易生成新的区块并发送给提交节点。提交节点收到新生成的区块并进行校验，确认区块中的每笔交易都符合当前节点的状态，完成后将区块追加到本地的区块链。

2.5 功能需求与系统模块的关系

	数据存储	获取需存信息	返回查询信息
搭建底层节点(Peers)	√		√
实现业务逻辑链码(Chain code)	√		√
链码层基础上搭建应用	√		√

实现接口	√	√	√
包装接口为服务并注册到系统后端	√	√	√
上层查询、修改		√	√
与其他组信息传输		√	√

2.6 人工处理过程

人工处理过程包括人工发起交易以及人工查询交易信息。

2.7 尚未解决的问题

需要明确入链的交易信息具体有哪些内容，是否适用于建立的数据库。

需要明确查询交易信息的方式，可能的方式包括交易 ID、关键字等方式。

3.系统的结构

Hyperledger Fabric 的架构以及共识网络见本设计说明书 2.4 结构。

系统结构设计：

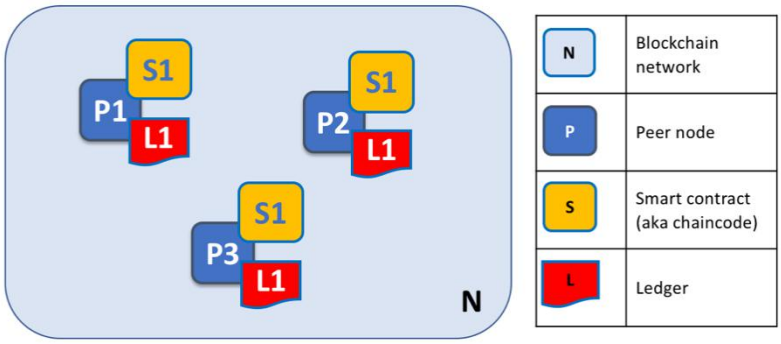
- 1 个 Hyperledger Fabric CA 节点
- 2 个 organization，每个 organization 包含 1 个锚节点
- 1 个 client application
- 1 个 ordered 节点
- 1 个 couchDB 数据库
- 1 个 channel，以 ordered 为核心，包含两个 organization 下面的所有节点

4.模块设计说明

4.1 底层节点(Peers)

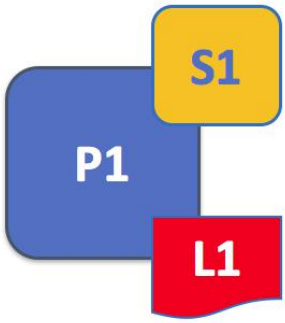
4.1.1 模块描述

一个区块链网络主要由一组节点(peer nodes 或简称 peers)组成。节点(Peers)是网络的基本组成部分因为其拥有账本(ledgers)和智能条约(smart contracts)。智能条约和账本用来各自概括贡献进程和共享信息。

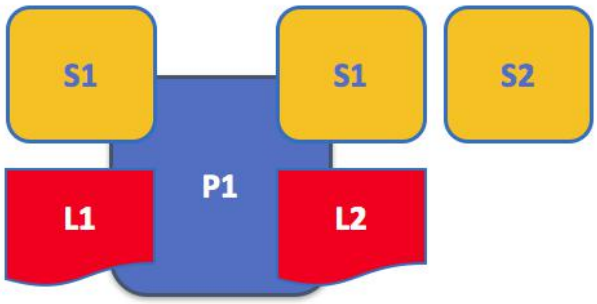


节点可以被创建、启动、停止、重构和删除。他们提供了一组接口(API)，能够使管理员和应用与它们提供的服务交互。

更加具体的来看，一个节点拥有账本的实例和链码的实例。一个单独的节点可以包含多个账本和链码。



如上图，节点 P1 拥有一个账本实例 L1 和一个链码实例 S1。



如上图，P1 拥有两个账本实例 L1 和 L2。L1 有访问链码 S1 的权限。L2 拥有访问链码 S1 和 S2 的权限。

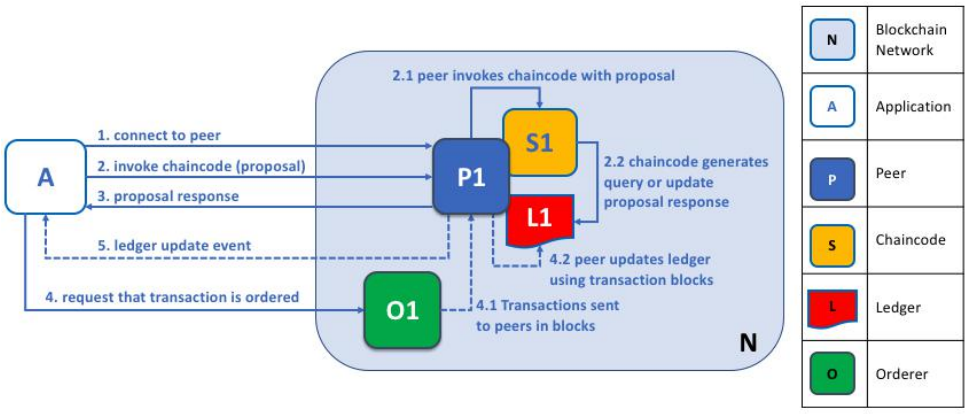
4.1.2 功能

• 应用程序和节点

账本查询和应用交互包括节点与应用间的简单的三个对话步骤。账本更新需要两个额外的步骤。如下图所示。

查询操作简化后的三个步骤如下：

1. 应用在连接账本和链码之前需要先连接到节点。区块链组织提供了一组接口供开发者很方便的连接到节点。
2. 执行链码进行查询或更新一个账本 ，查询结果会立即返回。
3. 节点返回结果至节点在本地的账本拷贝。

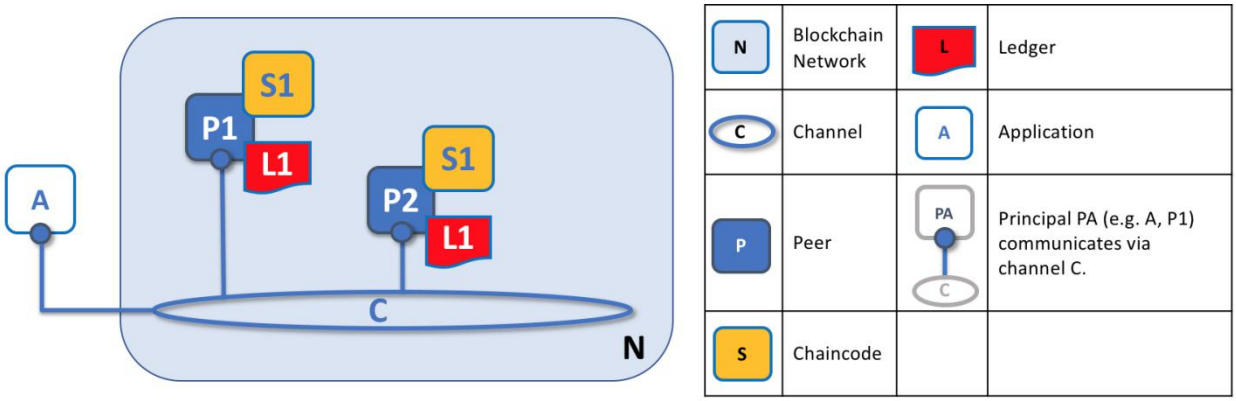


更新操作开始时与查询工作一样，但与账本查询不同，一个单独的节点不能进行更新，因为其他的节点要先同意这次操作，这一步骤称为达成一致(consensus)。第一个额

外的步骤是要求应用向整个网络发送一组合适的更新请求，并发给它们各自的账本。第二个额外的步骤是向应用发送异步通知，因为整个排序过程是需要一定的时间。

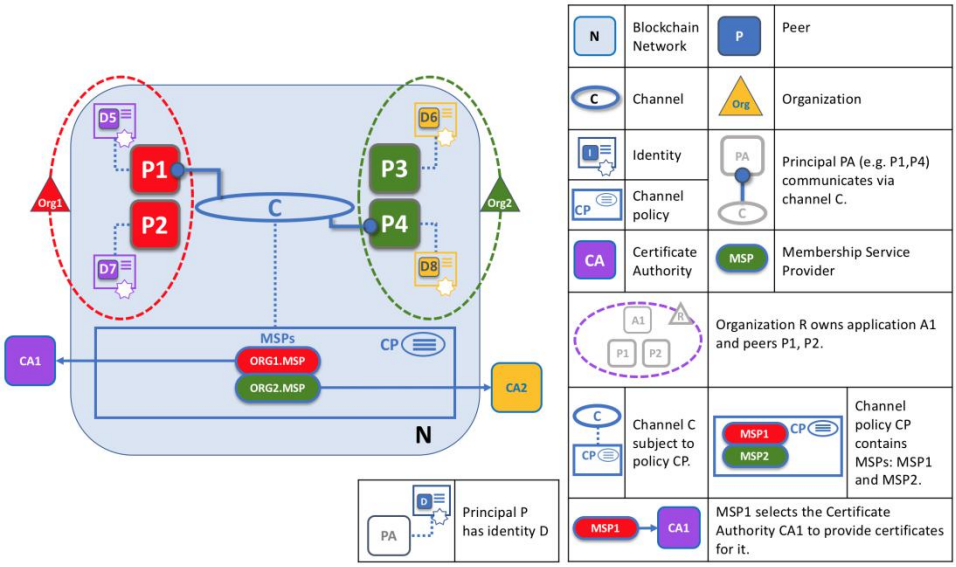
• 节点和频道

节点和节点、节点和应用之间是通过频道(Channels)进行交互的。通过频道，一个区块链网络内的组成可以私密地互相交流和商议。频道的存在形式与节点并不相同，可以认为频道是由一组物理节点组成的一种逻辑结构。



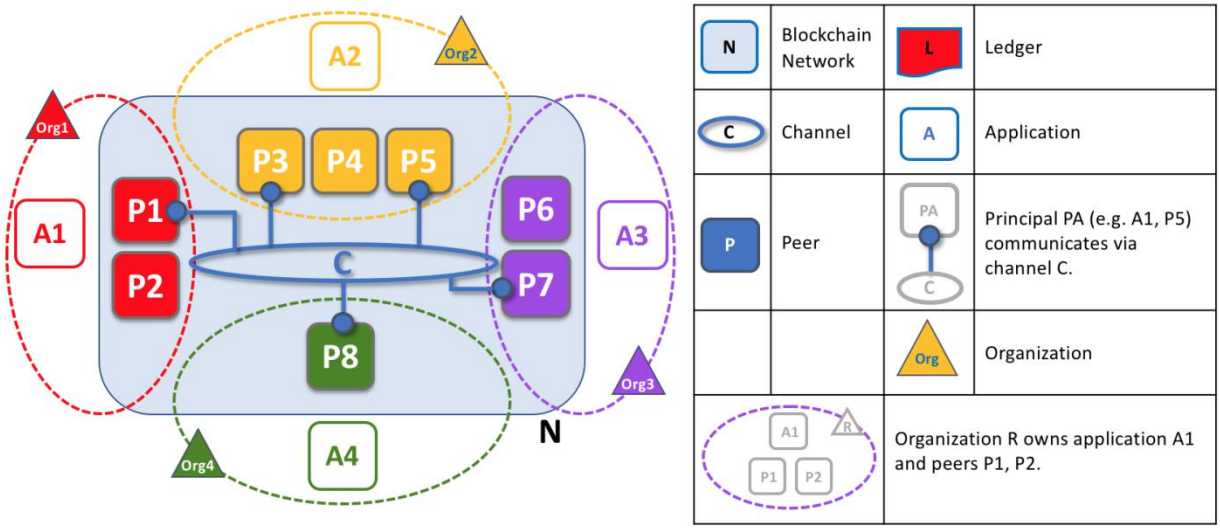
• 节点和身份

节点拥有通过特殊证书颁发机构颁发的数字证书，称为身份(Identity)。每当一个节点连接频道时，节点的电子证书通过 MSP 识别拥有的节点拥有组织。



• 节点与组织(Organizations)

区块链网络是由组织的集合而非单独的组织所构成的。节点是构成这种分布式网络的核心，因为节点被各个组织拥有



4.1.3 性能

可以被创建、启动、停止、重构和删除

4.1.4 设计方法

- 环境配置
- 生成证书文件
- Orderers 节点创建
- Peer 节点处理
- Channel 创建
- chaincode 安装部署、实例化与测试

4.1.5 测试计划

安装官方的链码演示(chaincode demo)，安装之后实例化一盒 chaincode 并试执行相关的命令

4.2 逻辑链码(chaincode)

4.2.1 模块描述

链码是一个继承了提前写好的接口的一个软件。链码运行在一个单独的 Docker 容器中。链码与账本相关，软件层面，链码运行在账本上，可以对资产编码，其中的交易指令可以修改资产。链码是执行，获取、更改键值或其他数据库状态信息的规则。链码的运行会让一组键值写入提交到网络并应用于所有节点的账本。

4.2.2 功能

对资产进行编码，其中的业务逻辑可以用来修改资产。

4.2.3 流程逻辑

- 当 peer 节点收到客户点请求 (proposal)后，会发送一个消息对象给对应的链码
- 链码调用链码中的 invoke 方法，通过发送获取数据(getState)和写入数据消息(putState)，向 peer 节点获取账本状态信息和发送
- 链码发送最终结果到 peer 节点，节点对输入(proposal)和输出(propsalreponse)进行背书签名，完成第一个阶段的前面提交
- 客户端收集所有 peer 节点的提交信息，组装事物并签发，发送事务到 orderer 节点排队，最终 order 产生区块，并发送到各个 peer 节点，把输入和输出“写在”账本上，完成流程。

4.2.4 接口

为 Fabric SDK(Java)实现插入、查询接口。

4.2.5 设计方法

- 创建 Package(包)
- 包签名(Package signing)

- 安装 chaincode
- chaincode 实例化

4.2.6 测试计划

完成链码在一个生命周期内的操作：打包、安装、实例化和升级。如果无报错顺利执行，则测试通过。

4.3 Fabric SDK

4.3.1 模块描述

区块链的应用层面，为实现底层区块链的实际操纵，需要编写 Java 方法调用底层区块链的接口。SDK 帮助加速 Java 应用程序管理区块链的频道和链码的生命周期，同时也提供了方法用以执行用户链码、查询区块链交易和监控频道活动。

4.3.2 实现功能

SDK 提供了很多方法。本项目需实现底层区块链的插入和查询操作。

4.3.3 性能分析

保证代码正确性的前提下，降低插入和查询操作的算法复杂度以优化算法。

4.3.4 接口

Fabric SDK 实际调用了底层节点 Peers 和业务逻辑链码 Chaincode 向外提供的接口，同时 Fabric SDK 中实现的 java 方法又是 Chaincode Service 的基础

4.3.5 存储分配

使用 Java 语言编写代码对内存和硬盘空间有一定的要求。

4.3.6 测试计划

利用黑盒和白盒测试Fabric SDK对于底层节点和业务逻辑链码的控制情况和区块链插入以及查询的结果是否符合预期。

4.4 Chaincode Service

4.4.1 模块描述

链码服务模块允许其他组查询或更新区块链中记录的交易信息，以达到区块链系统的公证目的。该模块充当区块链系统和外界交互的重要模块。

4.4.2 功能

- 向区块链插入实名信息和合同签署信息
- 插入放款和还款信息
- 查询实名信息和合同签署信息
- 查询放款和还款信息

4.4.3 性能分析

保证代码正确性的前提下，降低插入和查询操作的算法复杂度以优化算法。

4.4.4 设计方法

- 插入操作：检验插入数据的合法性之后插入区块链。
- 查询操作：直接调用区块链内置的查询方法。

4.4.5 流程逻辑

插入实名信息和合同签署信息：方法要求的各种参数→调用 InsertTransaction 方法→插入成功返回 true，插入失败返回 false

插入放款与还款信息：方法要求的各种参数→调用 InsertBalanceChange 方法→插入成功返回 true，插入失败返回 false

查询实名信息与合同签署信息：交易记录 ID→调用 QueryTransaction 方法→返回 String 形式的实名信息和合同签署信息

查询放款与还款信息：交易记录 ID→调用 QueryBalanceChange 方法→返回 String 形式的放款与还款信息。

4.4.6 测试计划

- 单元测试：测试分为白盒测试和黑盒测试，主要测试正确性和响应时间。
- 集成测试：与其他模块以及平台的其余系统对接，测试兼容性、稳定性。

5. 接口设计

5.1 用户接口

区块链存储系统将向虚拟账户系统暴露以下操作接口：

(1)invokeUserInformation 方法

方法描述：向区块链中插入实名信息

方法参数：

参数类型	参数名	参数描述	备注
String	userName	用户名	无
String	encrypted_message	加密信息	由上游进行加密

返回参数：

参数类型	参数描述	备注
void	无	无

抛出异常：

异常类型	异常描述	备注
WriteFailureException	自定义异常，写入失败时抛出	无

(2)queryUserInformation 方法

方法描述：从区块链中查询用户信息

方法参数：

参数类型	参数名	参数描述	备注
String	userName	用户名	无

返回参数：

参数类型	参数描述	备注
String	用户信息的加密信息	由上游进行解密

抛出异常：

异常类型	异常描述	备注
ReadFailureException	自定义异常，要查找的数据	无

	不存在于区块链中时抛出	

(3)invokeFinancingApply 方法

方法描述：向区块链中插入融资申请信息

方法参数：

参数类型	参数名	参数描述	备注
long	recordId	融资申请记录的 ID	无
String	encrypted_message	加密信息	由上游进行加密

返回参数：

参数类型	参数描述	备注
void	无	无

抛出异常：

异常类型	异常描述	备注
WriteFailureException	自定义异常，写入失败时抛出	无

(4)queryFinancingApply 方法

方法描述：从区块链中查询融资申请信息

方法参数：

参数类型	参数名	参数描述	备注
long	recordId	融资申请记录的 ID	无

返回参数：

参数类型	参数描述	备注
String	融资申请信息的加密信息	由上游进行解密

抛出异常：

异常类型	异常描述	备注
ReadFailureException	自定义异常，要查找的数据不存在于区块链中时抛出	无

(5)invokeContract 方法

方法描述：向区块链中插入签署的合同的信息

方法参数：

参数类型	参数名	参数描述	备注
long	recordId	签署的合同的 ID	无
String	encrypted_message	加密信息	由上游进行加密

返回参数：

参数类型	参数描述	备注
void	无	无

抛出异常：

异常类型	异常描述	备注
WriteFailureException	自定义异常，写入失败时抛出	无

(6)queryContract 方法

方法描述：从区块链中查询签署的合同的信息

方法参数：

参数类型	参数名	参数描述	备注
long	recordId	签署的合同的 ID	无

返回参数：

参数类型	参数描述	备注
String	合同信息的加密信息	由上游进行解密

抛出异常：

异常类型	异常描述	备注
ReadFailureException	自定义异常，要查找的数据不存在于区块链中时抛出	无

(7)invokeLoan 方法

方法描述：向区块链中插入放款信息

方法参数：

参数类型	参数名	参数描述	备注
long	recordId	放款记录的 ID	无
String	encrypted_message	加密信息	由上游进行加密

返回参数：

参数类型	参数描述	备注
void	无	无

抛出异常：

异常类型	异常描述	备注
WriteFailureException	自定义异常，写入失败时抛出	无

(8)queryLoan 方法

方法描述：从区块链中查询融资申请信息

方法参数：

参数类型	参数名	参数描述	备注
long	recordId	放款记录的 ID	无

返回参数：

参数类型	参数描述	备注
String	放款记录的加密信息	由上游进行解密

抛出异常：

异常类型	异常描述	备注
ReadFailureException	自定义异常，要查找的数据不存在于区块链中时抛出	无

(9)invokeLoan 方法

方法描述：向区块链中插入还款信息

方法参数：

参数类型	参数名	参数描述	备注
long	recordId	还款记录的 ID	无
String	encrypted_message	加密信息	由上游进行加密

返回参数：

参数类型	参数描述	备注
void	无	无

抛出异常：

异常类型	异常描述	备注
WriteFailureException	自定义异常，写入失败时抛出	无

(10)queryLoan 方法

方法描述：从区块链中查询还款信息

方法参数：

参数类型	参数名	参数描述	备注
long	recordId	还款记录的 ID	无

返回参数：

参数类型	参数描述	备注
String	还款记录的加密信息	由上游进行解密

抛出异常：

异常类型	异常描述	备注
ReadFailureException	自定义异常，要查找的数据不存在于区块链中时抛出	无

5.2 外部接口

通信接口：

消息队列（Message Queue, MQ）：是一种进程间通信或同一进程的不同线程间的通信方式，软件的队列用来处理一系列的输入，通常是来自用户。消息队列技术主要用于解决应用耦合、异步消息、流量削锋等问题。系统采用 Rabbit MQ 最初起源于金融系统，用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不俗。Rabbit MQ 主要是为了实现系统之间的双向解耦而实现的。

软件接口：

• **Hyper Ledger Fabric**：是一种开源企业级许可分布式分类账本技术（distributed ledger technology，DLT）平台，专为在企业环境中使用而设计，与其他流行的分布式分类帐或区块链平台相比，可提供一些关键的差异化功能。其优点在于：1、是第一个

支持在通用编程语言（如 Java，Go 和 Node.js）中创建的智能合约的分布式账本平台，而不是受限制的特定于域的语言（DSL）；2、获得了法律许可；3、支持可插拔的共识协议；4、不需要本机加密货币。

• **Docker**：Docker 是一种执行操作系统级虚拟化的计算机程序，用于运行容器，让应用程序部署在容器下的工作可以自动化进行，借此在 Linux 操作系统上，提供一个额外的软件抽象层，以及操作系统层虚拟化的自动管理机制。

6. 系统数据结构设计

6.1 逻辑结构设计要点

ChainCode：Hyperledger Fabric 创建的区块链代码信息，涵盖所属 channel 等信息

Orders：Hyperledger Fabric 创建的 order 信息，涵盖单机和集群两种方案

Peers：Hyperledger Fabric 创建的节点信息，包含有 cli，org，ca，couchDB 等节点服务器关联启动服务信息等

ChaincodeManager：智能合约操作的总控制器

FabricManager：区块链网络的控制类，通过调用上述类实现对整个网络的控制

6.2 数据结构与程序的关系

	模块 4.1	模块 4.2	模块 4.3	模块 4.4
ChainCode		√	√	
Orders		√		
Peers	√		√	
ChaincodeManager			√	
FabricManager				√

7. 系统出错处理设计

7.1 异常类别

程序运行时可能会抛出以下异常，需要程序员对这些异常对进行捕获处理。

ReadFailureException：本系统自建的异常类，位于 service.utils.exception 包下。当上游调用我们的 api 查询区块链中的信息时，如果要查询的信息不存在则抛出此异常。

WriteFailureException：本系统自建的异常类，位于 service.utils.exception 包下。当上游调用我们的 api 要将信息存储在区块链中时，如果信息存储失败则抛出此异常。

IOException：java.io 包中的输入输出异常

NoSuchAlgorithmException：java.security 包下的异常，当某一特定的加密算法在当前环境中被请求调用却失败时被抛出

NoSuchProviderException：java.security 包下的异常，当某一特定的安全提供者在当前环境被请求却失败时被抛出

InvalidKeySpecException：java.security.spec 包下的异常，当出现无效秘钥时，该异常被抛出

CryptoException: hyperledger.fabric.sdk.exception 中自建的加密异常

InvalidArgumentException: hyperledger.fabric.sdk.exception 中自建的非法参数异常

TransactionException: hyperledger.fabric.sdk.exception 中自建的交易异常

Exception:上述异常的父类，为 hyperledger.fabric.sdk 中异常的总类

7.2 健壮性设计

除了 7.1 中所述之异常外，程序运行过程中可能会遭遇不可预见的错误，比如存储数据时因断电或遭到黑客攻击而中断等，此时本程序应当有一定的补救措施将程序恢复至错误发生前的状态。

1、**备份**

定期将数据存储至企业级云盘或者公司自己的存储设备，并且使用日志记录技术，将一段时间内的操作记录下来。

2、**恢复**

当系统的数据丢失时，可以通过启用备份同时通过执行日志中记录的操作恢复数据。

7.3 可靠性以及可维护性设计

区块链自带分布式账本模式本身是具备极高可靠性的设计，区块链自身的分布式账本模式在极大程度上保证了区块链上存储数据的正确性和一致性，在系统中加入日志记录机制，可进一步加强区块链的可靠性，有利于系统的检查和维护操作。