分类号　　密级

华中农业大学学士学位论文

基于 HyperLedger 的食品安全区块链构建

HyperLedger-based Food Safety Blockchain Construction

姓　　名：刘开

学　　号：**2013310200305**

专 业 班 级：计算机科学与技术 **1403**

学 位 类 型：工学学士学位

指 导 教 师：倪福川　讲师

华中农业大学信息学院

中国·武汉

目录

摘要

食品安全溯源系统是对食品安全监管的有效手段，传统溯源系统的数据往往由供应链中的核心机构单独维护，数据很容易被人为篡改。本文通过对区块链技术的工作原理和技术特征进行总结分析,论述了将区块链技术应用于食品溯源系统的必要性和可行性。区块链技术在防伪溯源系统中有着强大的优势，如去中心化、数据不可篡改和特有的时间戳等特性。本文针对当前区块链技术的发展情况及相关主流的应用，使用超级账本 Fabric 构建了一个简易的基于区块链的食品安全溯源系统。系统主要分为四个层次，由下到上依次为区块链底层平台、智能合约层、业务层和应用层。使用 Go 语言编写与区块链账本交互的链码，Web 应用采用 express 框架进行搭建，通过 Node.js SDK 与区块链网络进行通信。通过对溯源系统的测试，可以防止节点对数据的篡改。设计的溯源系统阶段主要为种子到产品的种植生产过程，网络中节点的信息录入、查询、用户注册等功能均通过测试。

关键词：食品安全溯源；区块链；去中心化；超级账本；Fabric；智能合约；

Abstract

The food safety traceability system is an effective means for food safety supervision. The data of traditional traceability systems are often maintained by the core agencies in the supply chain, and the data is easily tampered with. This article summarizes the working principle and technical characteristics of blockchain technology and discusses the necessity and feasibility of applying blockchain technology to food traceability systems. Blockchain technology has strong advantages in anti-counterfeit traceability systems, such as decentralized, data irrevocable, and unique time stamps. Aiming at the development of current blockchain technology and related mainstream applications, a simple blockchain-based food safety traceability system was constructed using the Hyperledger Fabric. The system is mainly divided into four levels, from bottom to top are the blockchain underlying platform, smart contract layer, business layer and application layer. The Go code is used to write the chaincode that interacts with the blockchain ledger. The Web application is built using the express framework and communicates with the blockchain network through the Node.js SDK. By testing the traceability system, it is possible to prevent the node from tampering with the data. The design traceability system stage is mainly the seed to product planting production process. The node's information input, query, user registration and other functions have passed the test.

Key words: Food safety traceability; Blockchain; Decentralization; Hyperledger; Fabric; Smart contract;

1. 前言

1.1 研究背景与意义

1.1.1 背景

当前，随着人们生活水平的持续提升，食品安全越来越受到大众的关注。科技进步极大的带动了食品行业的发展，食品种类更加多样化，加工过程也愈加复杂。食品安全是立国之本，是建立以人为本的和谐社会的重要指导因素，它与人民群众的身体健康、社会稳定，以及政府和国家的形象密切相关。然而，近几十年来，食品安全事故在世界各地频频出现，食品造假问题是全球性难题之一，即便是知名食品生产加工企业也有很多被发现存在极大的安全问题。比如 2008 年的三鹿“三聚氰胺奶粉”事件，2011 年德国的二恶英“毒饲料”事件，2013 年新西兰的恒天然企业浓缩乳清蛋白粉检出肉毒杆菌事件等。另外还有诸如“地沟油”、“镉大米”、“僵尸肉”等食品安全问题，以及造成的“疯牛病”、“禽流感”、“口蹄疫”等较大范围爆发的疾病。时至今日，我国食品安全问题已经变得非常严重，大众在选择食品时充满各种担忧。食品原料主要来自于农副产品，农副产品从种植、库存、加工、包装、运输、销售，要经过许多步骤，其中任意环节都有可能会出现问题。中国政府高度重视食品安全问题，构建食品溯源系统是提高食品安全的重要举措之一，将食品认证、供应链跟踪作为保障食品安全的措施，以此可以快速搜索和发现食品的污染源。

近些年来，由于比特币的横空出世及高速发展，区块链开始受到人们极大的重视。比特币被认为是区块链诞生的摇篮，它是一个分布式的 P2P 网络，矿工使用挖矿过程来进行维护，生成新区块并对交易信息进行记账（杨宝华 2017）。比特币与传统交易系统的差异之处在于它是以去中心化（Decentralization）为基础的信任。根据 CoinDesk 的 2017 年第二季度区块链状态汇报内容，2017 年比特币市值提升超过 2 倍，加密数字货币超过 800 种，总市值超过 1000 亿美元，前三大货币依次为比特币、以太币和瑞波币（Ravikant 2017）。区块链技术的诞生打破了人们对传统交易的认知约束，比特币的出现改变了传统的数字货币交易方式，对分布式记账技术的发展有着深远的影响。而后，以太坊项目引入了智能合约这个重要特性，开发出了能够使用图灵完备脚本语言的虚拟机，使数字货币交易的功能变得更加灵活和复杂（Ethereum White-Paper 2015）。同时，智能合约的诞生也使得区块链可以作用于除货币交易以外的其他范畴，比如金融服务、征信权属管理、资源共享、贸易管理、物联网等。此外，区块链技术也受到了政府机构和资本市场的极大关注。

当前国内食品的安全质量受到较为严峻的信任难题，唯有重新构建食品安全治理模式，进而实现人们可信任的大范围监视，才是解决目前信任危机的唯一办法（李想和石磊 2014）。在 2016 年区块链技术高速发展，对于经济战略、企业界及学术界而言，区块链都受到极大重视（董宁和朱轩彤 2017）。由于区块链存在数据实时共享和不可篡改等特性，把区块链应用于食品领域，将区块链当作底层技术开发食品安全溯源系统，利用分布式账本结合智能合约构建新型的溯源逻辑，对食品溯源的可信任性、安全性和效力将会有极大的提升。

1.1.2 研究意义

早在 2003 年，溯源技术就已经被提出来了，政府机构竭尽全力的推动食品追溯系统的构建，但是最终成效却并不理想，以至于一直以来食品安全问题屡屡发生。主要原因是传统食品溯源数据是以中心化方式存储的，极容易被人为因素所影响。溯源过程的各道工序负责方把数据和信息存储在一个数据库中，除了自身和系统提供方，其他人都无法了解到详细信息，个人存放数据的不可靠性和被篡改的风险就会极大的增加。因此，消费者和大众用户对其并不太容易接受，进而就不会认可和使用该溯源系统。然而，区块链技术的去中心化（也可以称为多中心化）的分布式记账方式正好可以有效解决了这一问题。食品安全溯源系统如果利用区块链技术，那么可以让互相怀疑的人们构建起信任，可以高效率低成本地解决信任危机（孙志国等 2016）。

从 2017 年开始，互联网行业和传统食品企业便已经瞄准食品溯源系统在区块链上的开发潜力，例如溯源链（TACChain）、唯链（Vechain），以及沃尔玛食品溯源系统等。溯源链是共有链，目标是解决产品在生产、加工、销售等工序中的可靠溯源难题，进而解决消费者的信任问题。此外，在 2017 年 12 月 14 日，沃尔玛、IBM、京东、清华合作建立了中国首个“安全食品区块链溯源联盟”（王崇民和王翠竹 2018）。虽然以上溯源系统均使用区块链作为基本技术，但从公布的报告中可知现在还处于研究开发中，另外几者在应用场景、底层实现技术、研发和运营模式等方面也均有所不同。首先溯源链 TAC 完美结合了以太坊和比特币的优点，唯链则仅仅使用以太坊技术，而沃尔玛食品溯源系统采用超级账本技术。

区块链技术被公认为是在蒸汽机、电力系统和互联网之后可以推动人类发展的核心技术。未来，人类将会进入人工智能、物联网和区块链占主导技术的时代，若能将三者联合起来将会缔造极大的价值（张增骏等 2018）。区块链可当作树立信任的对象，进而改变人们信息交互的体例。存放在区块中的数据是透明的，而用户身份是被加密过的，唯有在授权后才可以使用，因而保障了信息的安全和用户的隐私。目前，随着国家打假力度的加大，以及消费者对食品的不信任性，防伪溯源的市场需求变得更加紧迫。区块链技术在防伪溯源系统中有着强大的优

势，比如去中心化、数据不可篡改和特有的时间戳等，相对其他技术而言会更加有效。综上所述，如果区块链与溯源相结合，将会给食品领域带来极大推动作用。

1.2 国内外研究现状

食品安全溯源诞生的主要原因是欧盟为了应对上世纪 90 年代爆发的“疯牛病”，开始构建和完善溯源系统，当时的食品主要指的是牛肉等畜牧类。欧盟于 2002 年成立了欧盟食品安全局，并颁布了相关的法律法规，限制在欧盟内销售的牛肉类、蔬菜等食品都需要提供必要的溯源信息，禁止无法溯源的食品进行销售（Koutsoumanis and Gougouli 2015）。自此之后，多数发达国家例如美国、澳大利亚、日本等也着手开始进行食品溯源系统的构建，在全国推广食品可追溯体系。比如，在 2008 年，美国农业部推出了 NAIS 项目，倡导使用标准动物识别号码，目的是在保障牛肉制品的可溯源性（邢文英 2006）。2001 年，日本将食品从最初的牛肉扩展到其他肉食、生鲜蔬菜等产业，可追溯系统发挥着监管食品安全的用处（陈欢和方越锋 2016）。

相对而言，我国对食品溯源方面的研究则开始的较晚，但是相关研究数量提升的很快。溯源相关研究起始于 2002 年，在 2004 年完成了对食品安全溯源系统的构建，直至目前，我国存在多种多样的食品溯源系统，具有代表性的有国家食品安全追溯平台和产品质量电子监管网等。之后政府一系列相关工作，都体现了政府当局越来越重视食品安全监管的作用。通过构建透明的监控体系，增强配套规章制度建设，最终完成食品安全、高效和可溯源的目标。另外，对食品溯源体系与新技术的结合研究中也在如火如荼的进行中。目前，应用在食品溯源系统中的技术有 RFID 射频技术、二维码、物联网技术、Web Service 技术等，还可使用智能手持终端设备进行控制和管理。彭远斌等对已有的物品编码方式和溯源体系进行总结分析后，提出了一种新型的溯源物品编码模式（彭远斌等 2014）。

从一开始以比特币、以太坊等为代表的共有链项目，发展为现在区块链相关的多种形式的创业公司、金融企业及互联网行业，区块链正在慢慢发展及发挥技术作用。另外，共享经济和供应链中也能够结合区块链技术，用户可以查找存放的交易记录，进行数据的追踪和分享。在供应链溯源中，可以把产品生产和加工等数据存放在区块链账本中，系统通过提供深度回溯等核心功能，可以达到相关信息的公开透明。价值较高的食品、药品及生产质检信息等也能够通过区块链来保障安全（张增骏等 2018）。此外，2017 年 3 月 28 日在新西兰，阿里巴巴与恒天然等签订了有关跨境食品溯源的互信合作协议，旨在利用区块链等最新技术，在中新间持续推行可溯源的跨境食品供应链（李子晨 2017）。

1.3 研究目的

构建食品安全溯源体系具有重大的战略意义，一旦发生了食品安全意外事件，可以通过食品溯源系统高效和准确地确定有问题的环节，将责任准确定位，并可以及时将该问题食品从市场中撤回，将危害最小化。但直至目前为止，已有的食品溯源系统还存在很多天然的不足，比如数据存储在统一的中央数据库中，大多数工序需要人为操作，任何环节的数据都可能会被篡改等缺陷，从而造成政府、企业和大众之间的信息不对称。目前，我国食品安全现存的首要问题有种植环境和生产过程被污染，食品安全监管制度缺陷造成的违法生产经营，利益驱动造成的厂家恶性竞争等（黄若君 2013）。

将区块链当作底层技术构建新型的食品可追溯体系，分布式账本使参与方均存储一份账本信息，利用区块链特有的优势实现信息不可篡改和可追踪溯源，这是目前解决食品安全困境的有效方法之一。区块链能够加强食品供应链的安全性和透明性，高效完成对食品的召回和检验工作。目前基于区块链技术的食品安全溯源系统案例还比较少，大多数都处于研究阶段。本文通过对区块链的发展历史和特有特性进行研究，研究学习区块链领域中典型的比特币、以太坊和 Hyperledger（超级账本）项目，重点学习超级账本社区的早期项目 Fabric，论文最终的研究目的是利用 Fabric 完成一个食品安全溯源系统的构建。

2. 相关理论综述

2.1 食品安全溯源

目前，全球对食品的可追溯性还没有统一的定义，大多只是对食品溯源系统的需求有共同的约束。比如食品溯源系统应该高效地完成卫生管理部门的质检工作，可以提高食品生产的自控能力，可以高效分析会对产品质量产生影响的多种因素，进而可以对食品起到很好的监控作用，系统需要包含从生产到销售整个供应链的公共信息。此外，通过食品溯源系统能够让企业极大的减少召回食品的费用，有效的挽救损失和消费者的信任。欧盟委员会将食品溯源定义为在种植、加工和销售等工序中，对有可能作为食品成分中的物质的追踪过程（何静等 2014）。国际标准化组织对其定义为对产品进行有效回溯和追踪的能力，回溯指的是从产品向前依次识别一个或者一批食品详细信息的过程，追踪指的是从种子向后对产品信息进行记录和采集的过程（Inerney et al 2011）。食品溯源系统可以标识食品的来源，提供其从种植到销售的全部信息，在供应链的各个阶段中识别产品身份，是由产品信息采集、存储和质检等一系列溯源工序构成的整体（Aiello et al

2011）。食品溯源体系大致流程如图 1 所示，主要涉及到农牧业、制造商、分销商与零售商，每个阶段都有其独有的作用，可以对各方分别进行内部溯源和外部溯源，检查是否每道工序都是安全的。

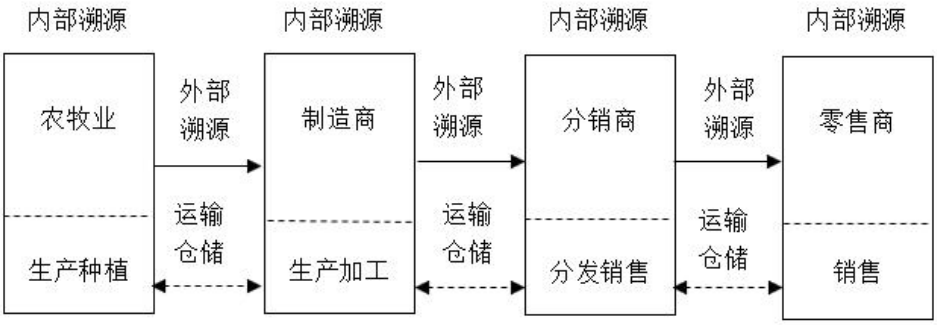

图 1 食品溯源体系基本流程

Fig.1 Basic flow of food traceability system

要使溯源系统能够运作，首先要解决的就是如何对食品进行编码和标识。另外，产品的质量和安全监控以及原产地检验是食品生产和销售中的重要环节，也是粮食和农业组织应该首先关注的问题。食品从生产到分销的每道工序，都需要承接上道工序的编码信息，并产生本工序的详细信息，然后加上相关的交易信息存储到数据库中。食品安全溯源系统的构建，有利于高效找到问题所在，并及时的召回存在问题的食品，进而可以有效控制食品质量，为广大消费者提供良好的食品安全环境。同时，也为食品监管机构和食品制造企业提供一个共享信息的平台，加强了食品信息的透明度，保证了交易的可靠性，保障了消费者的知情权，对建立市场信用具有重要的战略指导意义。

2.2 区块链技术

2.2.1 区块链简介

区块链技术利用共识机制，无需中心化机构便能解决双重支付难题，使得数字货币范畴完成了重大突破。提起区块链，往往首先想到的就是比特币，区块链技术的核心出现在比特币中，但是区块链却不等于比特币。区块链是按时间顺序将区块串联而成的一种结构，区块中存放了一段时间内的交易（即对账本的操作），并对其使用密码学技术进行加密，从而保障用户隐私和数据安全，并且分布式账本只能够添加，而无法进行删除操作。但区块链也并不仅仅是数据库，即便区块链的核心是点对点的分布式账本，但是它解决的核心问题使参与方可以互信，主要是利用共识机制保障各节点的数据一致性。

区块链中的首个区块被叫做创世区块，此后的区块都需要含有上一区块头的 Hash 值，该值是利用 SHA256 算法对其二次 Hash 获得的一个数字指纹。这个 Hash 值保障了交易信息的安全性，若区块信息被修改了，则它的值也会转变，需重新计算后续的区块 Hash 值。除非有及其强大的计算力才可以完成这个工作，因此这个工作量保证了区块的安全。此外，识别区块的方法除了 Hash 值，还能利用区块高度来识别，第一区块的区块高度为 0，之后产生区块的高度都比前一区块高出一个位置（Andreas 2017）。

图 2 区块链基础架构模型

Fig.2 Blockchain infrastructure model

如图 2 所示，区块链体系主要可以分为数据层、网络层、共识层、激励层、合约层和应用层六个层次（何渝君和龚国成 2017）。其中，共识层包含的是使节点数据保持一致性的共识算法，是区块链中的核心技术；合约层包含各种算法、脚本和智能合约，对区块链技术的发展具有尤为重要的推动作用，使得区块链可以应用于更加复杂和灵活的场景。在该体系中，带时间戳的区块结构、共识机制及智能合约是核心特性。

区块链技术的核心特点是去中心化和数据无法篡改，让所有的信息均使用预定环节自动执行，极大的减少了成本。由于参与方均保存一份相同的账本，使交易记录保持安全和透明，因此也可以称作为多中心化。区块链中包含的技术有很多，它是将已有的密码学、记账技术、分布式等范畴的成功进行结合，比特币在数字货币范畴的重大成功使得区块链受到了众多领域的关注。

2.2.2 Merkle Tree 结构

Merkle Tree 是区块链中的一个重要构成元素，广泛应用于文件系统和 P2P 系统中，比特币和 Fabric 使用的是典型二叉 Merkle Tree，但以太坊使用的是改进后的 Merkle Patricia Tree（MPT，Merkle 前缀树）。Merkle Tree 是 Hash 二叉树的一种，树的所有节点存放的均为 Hash 值，其中叶子节点的 Hash 值使用 SHA256 计算获得，非叶子节点的值则由子节点串联后再次使用进行 SHA256 而得。因此，创建它需要从下向上依次进行散列值的计算。

如图 3 所示，在比特币系统中，所有区块均包含 Merkle Tree 结构， Root 的 Hash 值放在区块头部。其中任何交易信息的改变都会影响整个 Merkle Tree，通过 Merkle Tree 结构可以利用部分 Hash 快速的检测数据是否完整，以及发现改变的数据块。此外，在区块链中，Merkle Hash 可以避免添加新交易时，需要重新计算本区块内的所有交易的 Hash。

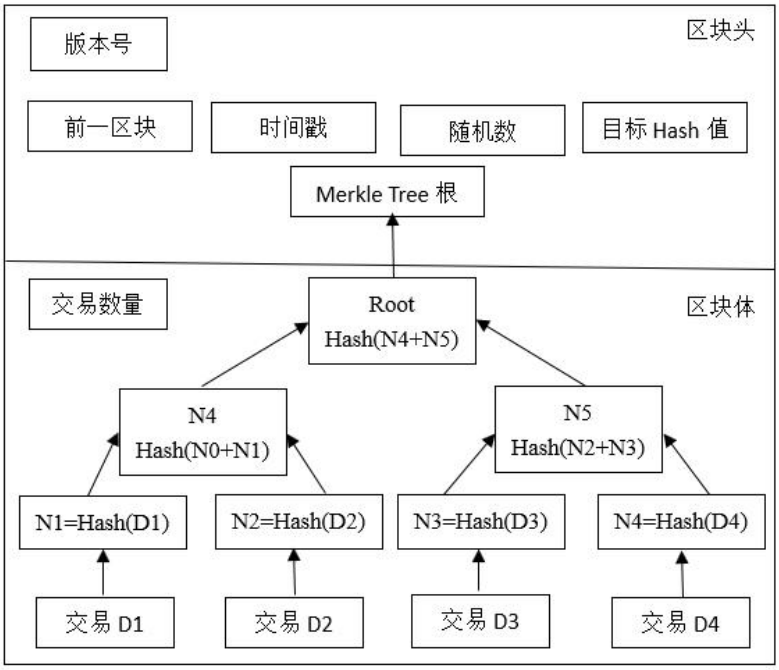

图 3 区块基本结构

Fig.3 Block basic structure

2.2.3 共识机制

共识机制可以说是区块链技术的灵魂，分布式系统需要解决的核心问题便是需要保证各节点信息的一致性。因为 P2P 网络中会有延迟存在，所以每个节点观测到交易的顺序也许并不一样，共识机制可以说是对多个交易的顺序获得共识的

一类算法。现如今存在很多种共识算法，但还未存在完美的，所有算法都存在优缺点，算法也是在一步一步的改进中，并且某些共识算法就只是为了解决某些特定的问题而诞生的。

共识算法总体可分为两大类，即 Crash Fault Tolerance（CFT）和 Non-Crash（Byzantine） Fault Tolerance（BFT）。前者指节点出现故障的原因也许是网络延迟或机器停止运行（宕机）导致的，并不会伪造信息，包括 Paxos、RAFT 等，这种情况相对来说比较容易解决，而后者则用来解决存在恶意节点等情况。目前常用的几种共识算法大致有以下几种:

（1）PoW，即工作量证明机制；PoW 典型的应用是比特币，如同字面含义一样，工作量越大，那么收益则越大。

（2）PoS，即权益证明机制；PoS 的典型应用是点点币，类似于股权凭证和投票系统，通过计算持有货币的百分比和持有时间来决定获得本次记账权利的概率，也可以称为股权证明算法，此外 DPoS 是改进算法。

（3）PBFT，即实用拜占庭容错算法；拜占庭将军问题是对现实问题的模型化，处理有少数节点作恶的情形。算法思想主要是对状态机副本进行复制，以计算为基础，并没有代币奖励，在保证安全前提下提供了（n-1）/3 的容错性。

区块链利用共识机制和加密技术，使一个不可信的网络变的可信，参与方可以在某些方面达成一致，而无需信任单个节点。

2.3 超级账本 Fabric

2.3.1 Hyperledger 简介

区块链的发展过程可分为 1.0、2.0 和 3.0 时代。其中，1.0 时代主要应用于数字货币范畴，解决中心化和可信问题；2.0 时代引入了智能合约，在区块链背景下，智能合约可当作一种运行在区块链之上的通用计算模式，使用高级编程语言将现实中的业务逻辑在区块链中进行实现，具有代表性的便是以太坊；3.0 时代便是为形形色色的行业带来解决难题的方法，进入万物互联时代，具有代表性的是 Hyperledger 项目，该项目使得区块链技术获得了更大的进步。

如图 4 所示为 HyperLedger 的基本架构，Hyperledger 在之前区块链模型的基础上，对用户参与的权限管理进行了改变。在相同类型的公司竞争和合作中，Hyperledger 可当作 B2B 和 B2C 的交易规约，不仅契合法律法规，也可对多种要求完成技术支持工作（Hyperledger White Paper 2016）。可以说， Hyperledger 提供了对身份进行辨别和审计，以及保障隐私的模型，大大减少了计算时间，使得满足各个行业应用的需求成为可能（张增骏等 2018）。

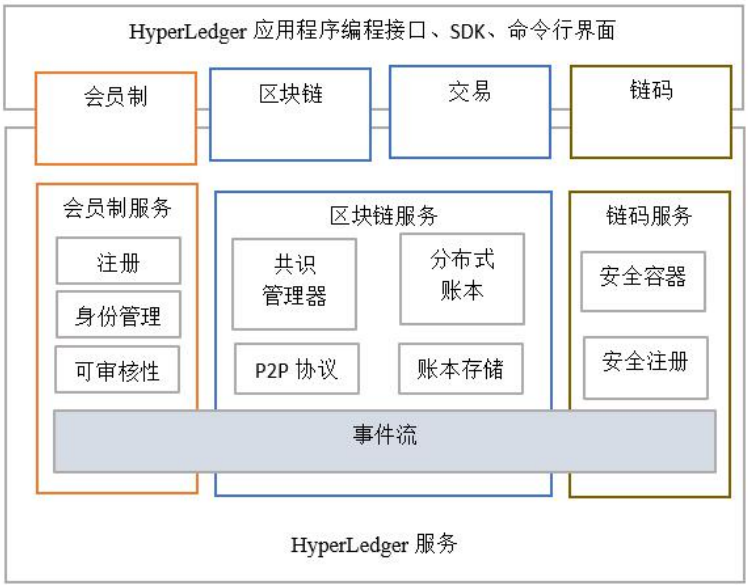

图 4 超级账本基本架构

Fig.4 HyperLedger basic architecture

2.3.2 Fabric 架构与组件

Fabric 作为 Hyperledger 社区一个有代表性的早期项目，首次提供了面向联盟范畴的分布式账本平台实现，实现了完备的权限管理功能。因此，Fabric 相对其他项目最明显的差异在于私有和许可，与共有链中无需许可便允许未知身份的参与者加入网络不同，Fabric 使用 MSP（Membership Service Provider）来登记所有的成员，避免了需要通过工作量证明协议来保证交易有效，从而缩短了运算周期实现有效扩展应对业内各种运用要求（Hyperledger Fabric Docs 2017）。MSP 提供了身份注册功能，用户获得证书后可以向交易认证中心申请交易证书，只有这样才可以进行交易，交易证书信息永久保存在区块链上，保障了网络的安全性。

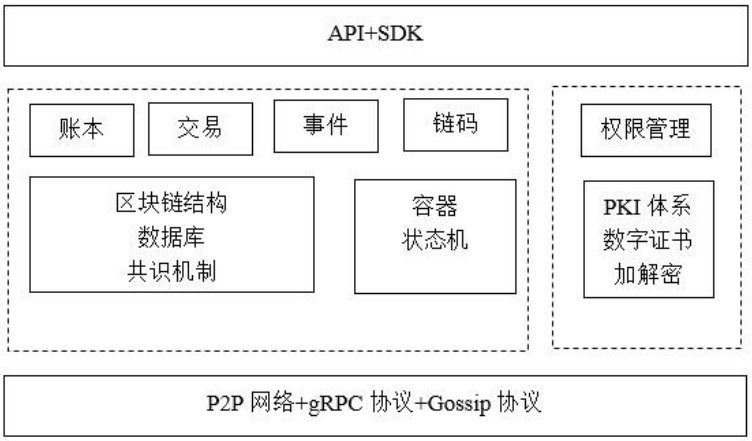

图 5 Fabric 整体架构

Fig.5 Fabric overall architecture

Fabric 的整体架构如图 5 所示，从中可以很明显看出各组件间的联系，架构采用层次化结构，加强了网络的安全隔离和可插拔性，使得开发工作变得更加灵活。Fabric 为应用了 SDK 的 API，目前比较常用的是 fabric-sdk-java 和 fabric-sdk-node，通过 SDK 可以访问 Fabric 中的 Ledger（账本）、Transaction（交易）、Event（事件）、Chaincode（链码）等资源。事件存在于各组件中，使它们之间的异步通讯获得技术支持；链码依赖于容器和状态机，部署在网络节点上是与 Fabric 区块链交互的唯一方式；底层是节点构成的点对点网络，利用 gRPC 完成交互工作，各节点使用 Gossip 协议来以一种可扩展和强适应的方式广播账本和通道数据，完成账本同步（杨宝华 2017）。

Fabric 中的核心组件有 Ledger、Block（区块）、Node（节点）、Channel（通道）、Chaincode、共识机制、权限管理、CA（Certificate Authority，认证中心和数字证书发证系统）模块等，通过 Fabric 网络可以对各种资源进行配置与管理机制。其中，Fabric CA 项目以 PKI 体系为基础，为网络提供证书服务，采用了典型的 CS 架构。CA 服务端提供了完整的证书管理功能，可以服务于其他 CA，也可以用来签发用户证书，包括单个 RootCert（根证书）和 ECert（Enrollment Certificate，注册证书，长期拥有，用于身份认证），以及大量 TCert（Transaction Certificate，交易证书，在每次交易时生成，用于交易的签名），另外还有 TLSCert 用于 TLS 传输。CA 客户端可使用 fabric-ca-client 命令与服务端交互，可以进行登陆获取 ECert、注册用户、证书签发、吊销证书等。

图 6 为 Fabric1.0 区块链网络基本示意图，在 1.0 中将 0.6 中的单一节点分为 Peer（Endorser 背书节点和 Committer 记账节点）和 Orderer（排序节点）。其中 Endorser 节点用于对 Client 端的 TX Proposal（交易提案）完成检验和背书；Committer 节点用于在检测后将链码的执行结果写入账本，所有 Peer 均为 Committer 节点，Endorser 节点是动态角色，只有当 Client 向它发送交易请求时才是 Endorser 节点；Orderer 节点完成对链码执行结果的共识。

Fabric 1.0 中的账本也可以分为三种，即 Block Ledger、State Ledger 和 History Ledger，其中 Block Ledger 存放在所有的记账节点和 Ordering Service 中，采用文件系统实现，仅允许添加新的区块，存储了 Transaction 的读写集；State Ledger 存放的是 Fabric 区块链系统中所有变量值的集合，也叫做 World State，默认使用典型的 key-value 数据库 LevelDB，因此 Fabric 的账本也被称为 kvledger，也可选择 CouchDB 数据库，该账本存放在所有的记账节点中；History Ledger 是 1.0 新加的账本，可以通过配置选项选择是否打开，可以查看某个账号的历史交易信息，存放的是对交易的索引。

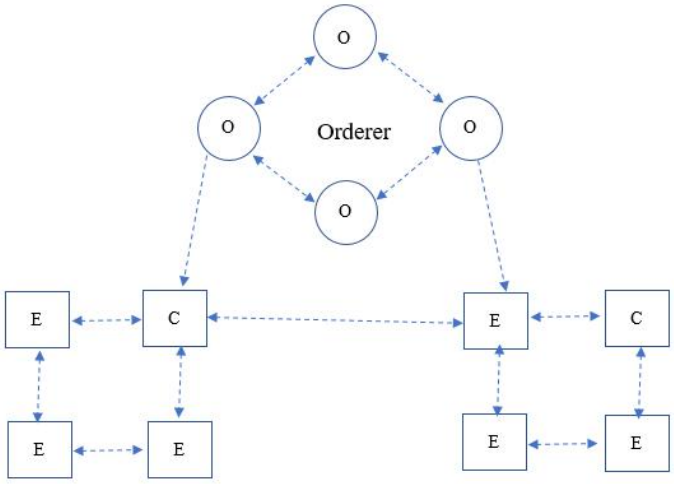

图 6 HyperLedger Fabric1.0 区块链网络

Fig.6 HyperLedger Fabric1.0 blockchain network

在 Fabric 中，共识由 Ordering Service 来完成，主要经过了背书、排序和验证三个阶段。Fabric 中的每个区块均包含一系列的交易信息，并基于 Merkle Tree 结构存储，信息包括发起方的数字签名（T-Cert），作用是在共识时对比交易信息与其它节点的数据是否相同。区块中还含有 World State 的 Hash 值，在共识时检验是不是与其它节点的 State 相同。Fabric 支持可插拔的共识协议，使平台能够更加有效地根据使用案例进行定制，即允许网络构建者依据业务需求来选择采用的共识机制。Fabric 共识机制目前包含 Solo 和 Kafka/Zookeeper，使用 kafka 集群方式完成交易的全局排序，只实现了 CFT 共识，因此并不能防止恶意节点的攻击，后续 Fabric 版本会再添加 SBFT（Simplified）。

2.3.3 Fabric 交易流程

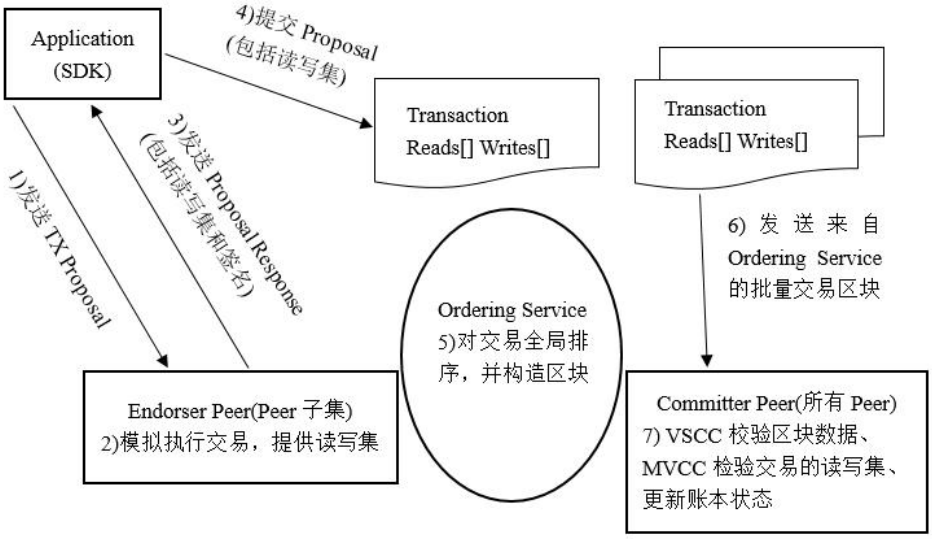

图 7 Fabric 交易生命周期

Fig.7 Fabric transaction lifecycle

Fabric 处理交易的大致过程如图 7 所示，首先 Client 通过 gRPC 接口向一个或多个 Endorser Peer 发送对交易的 Proposal（由消息头和消息结构两部分组成）；Endorser Peer 对提案进行合法性检验（包括格式、签名，以及提交者在当前通道上是否有权限等），模拟执行链码，却不会把结果存入账本中，而是以读写集（ReadWriteSet）的形式返回给 Client 一个提案响应（ProposalResponse）；Client 收集较多背书信息后，将交易发给 Orderer；Orderer 获得共识后生成 Block，然后将更新的状态和账本等信息通过原子广播发布给 Committer 节点；各 Committer Pee 使用 Gossip 验证交易，达成一致后存入本地账本中。

表 1. 系统链码综述

Table 1. System chaincode review

链码名称	是否可被外部调用	是否可被其他链码调用	是否可被替代	主要用处
CSCC	是	否	否	负责 Join Channel 和 Config Update 等
ESCC	否	否	是	负责对传入数据进行签名背书
QSCC	是	是	否	负责查询 Ledger
LSCC	是	是	否	负责 Deploy Invoke
VSCC	否	否	是	负责对签名和策略进行验证

Fabric1.0 中的交易被 Endorser Server 获得后，还需使用 System Chaincode 完成一些系统功能。如今共有五个系统级链码，对它们的含义和用处综述如表 1 所示。应用链码在部署时，其背书和验证过程都需要与系统链码进行关联；ESCC 完成对提案的背书工作；VSCC 判定有效性（比如背书），通过后进行记账。此外，kvledger 还将使用 MVCC（Muti-Version Concurrency Control）对读写集检验。

对于创建通道、Peer 节点加入通道、安装链码、实例化链码和调用链码的过程与交易流程存在部分相同之处，在此不再叙述。

3. 系统总体设计

3.1 溯源环节设计

已有的食品溯源系统涉及到的参与方有生产方、运输方、分销方和零售方等。本文主要将生产环节作为样例进行设计，对从种子到产品的过程进行细分，设计的食品关键环节的信息主要包括育秧（种子浸药与秧苗喷药）、种植（种植地点与种植用药）、仓储（入库信息、仓库信息与种植信息）、粗加工（产品进料信息）

以及深加工（产品原料信息），食品溯源包括真实性溯源（正向溯源）和安全性溯源（反向溯源）。溯源测试食品主要为稻米类，食品相关信息汇总如表 2 所示。

表 2. 食品相关信息汇总

Table 2. Summary of Food Related Information

信息	属性	含义	信息	属性	含义
	Seedid	种子 ID		Seedlingid	秧苗 ID
种子	Variety	品种	Seedling	Nurseryplace	育秧地点
Seed	Type	类型	秧苗	Startdate	育秧起始日期
	Personid	质检员 ID		Personid	质检员 ID
Seedsoak	Seedid	种子 ID	Plantuse	Plantid	种植 ID
-drug	Drugid	药品 ID	-drug	Drugid	药品 ID
种子浸药	Concentration	浓度(ml/L)	种植用药	Dosage	剂型
	Startdate	起始日期		Effect	药品作用
	Enddate	结束日期		Date	用药日期
	Personid	质检员 ID		Personid	质检员 ID
	Seedlingid	秧苗 ID		Inputid	入库 ID
Seedling	Drugid	药品 ID	Input	Harvestdate	收割日期
-spraydrug	Dosage	用量(ml)	入库管理	Quantity	入库量(kg)
秧苗喷药	Date	用药日期		Inputdate	入库日期
	Personid	质检员 ID		Personid	质检员 ID
	Drugid	药品 ID		Materialid	原料 ID
	Name	药品名称		Kind	种类
Drug	Dosage	剂型	Material	Weight	重量(t)
药品	Standard	生产标准	原料管理	Source	来源
	Effect	药品作用		Date	日期
	Personid	质检员 ID		Personid	质检员 ID
Plant2input 种	Plantid	种植 ID	Warehouse2fe	Warehouseid	仓库 ID
植入库	Inputid	入库 ID	ed 出库管理	Feedid	进料 ID
	Personid	质检员 ID		Personid	质检员 ID
Seed2	Seedid	种子 ID	Seedling2	Seedlingid	秧苗 ID
-seedling	Seedlingid	秧苗 ID	-plant	Plantid	种植 ID
种子秧苗	Personid	质检员 ID	秧苗种植	Personid	质检员 ID
	Product	产品 ID		Personid	质检员 ID
	Name	名称		Name	姓名
Product	Specification	口味	Person	Sex	性别
产品	Flavor	规格	质检员	Workplace	工作单位
	Date	日期		Job	岗位
	Personid	质检员 ID		Password	登录密码

续表

	Feedid	进料 ID		Plantid	种植 ID
	Weight	稻谷重量	Plant 种植	Place	种植地点
Feed	Watercontent	水分含量		Startdate	种植起始日期
进料管理	Brokenrice	碎米率		Personid	质检员 ID
	Qingmilv	青米率	User	Userid	用户 ID
	Date	日期	用户	Name	姓名
	Personid	质检员 ID		Password	登录密码
Feed2	Feedid	进料 ID	Inout2warehouse	Inputid	入库 ID
product 进料			入库仓库转换	Warehouseid	仓库 ID
产品转换	Productid	产品 ID		Personid	质检员 ID
	Personid	质检员 ID			
Warehouse	Warehouseid	仓库 ID	Material2	Materialid	原料 ID
仓库管理	Place	仓库地点	product		
	Capacity	容量(t)	原料产品转换	Productid	产品 ID
	Standard	生产标准			
	Personid	质检员 ID		Personid	质检员 ID

3.2 系统架构设计

3.2.1 食品溯源系统分层架构

图 8 为食品溯源系统分层架构图，主要分为四个层次，由下到上分别为区块链底层平台、智能合约层、业务层和应用层，具体每个层次的功能如下所示：

(1)区块链底层平台：提供对分布式账本和 World State 的维护，以及对智能合约的全生命周期管理等区块链功能，实现数据无法篡改和链码的业务逻辑。另外，通过 fabric-ca 提供成员注册和注销等功能。

(2)智能合约层：智能合约通过链码来实现，功能包括对各信息的查询和添加等，编写不同功能的链码，只有安装该链码的节点才能调用该链码操作账本。

(3)业务层：负责应用程序的后端服务，给 Web 应用提供 Restful 的接口，用于处理应用层的请求。后端服务的基本功能包括对所有信息的管理工作，通过 Hyperledger Fabric1.0 提供的 Node.js SDK 和区块链网络进行通信。

(4)应用层：Web 应用采用 express 框架进行搭建，提供用户交互的界面操作，包括用户操作的功能和业务操作的功能。普通用户可以进行注册和登录操作，但只能查询各食品的信息，按照种子 ID 和产品 ID 可以分别进行真实性溯源（正向溯源）和安全性溯源（反向溯源）。

各个层之间采用不同的接口，业务层的 Node.js SDK、智能合约和区块链底层平台之间采用 gRPC 接口，业务层和 Web 应用间采用 RESTful 接口进行通信。

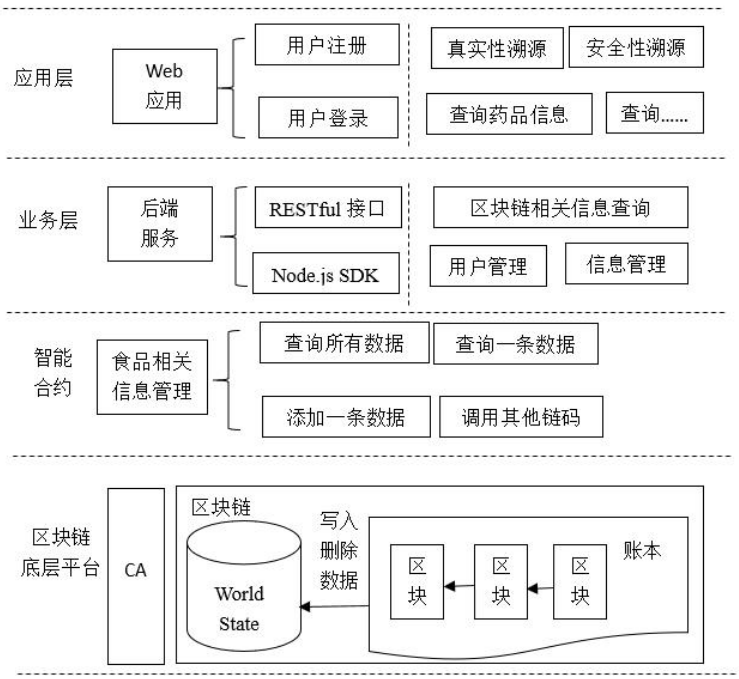

图 8 食品溯源系统分层架构

Fig.8 Food traceability system layered architecture

3.2.2 ChainCode 设计

表 3. myccfoodall 链码接口定义

Table 3. The myccfoodall chaincode interface definition

方法	分支方法	功能	参数示例
Init	无	无	[]
	initLedger	初始化账本，存入一些数据	[]
	initPerson	新增一个质检员	["FOOD888","Tom", "男","米业生产","技术员","123"]
	initUser	新增一个用户	["123456","张三", "123456"]
	readOneById	根据 ID 读取一个记录	["FOOD888"]
	readOneByIdOther	根据 ID 调用另一个链码读取	["myccfoodone", "readOneById","MATERIAL000"]
	readAll	根据范围查询一个类型的所有数据	["PERSON000", "PERSON999"]
Invoke	readAllOther	根据范围调用另一个链码查询一个类型的所有数据	["myccfoodtwo","readAll", "MATERIAL000","MATERIAL999"]
	deleteOne	根据 ID 删除一个记录	["FOOD888"]

表 4. myccfoodone 链码接口定义

Table 4. The myccfoodone chaincode interface definition

方法	分支方法	功能	参数示例
Init	无	无	[]
	initLedger	初始化账本，存入一些数据	[]
	initDrug	新增一个药品	["CCJHB","消毒剂","液态","国标","消毒","FOOD001"]
	initSeed	新增一个种子	["TLY83","江早 361","早稻","FOOD001"]
	initSeedling	新增一个秧苗	["TLY83HN0117020101","1 号大棚","2018-01-01","","FOOD001"]
	initSeedsoakdrug	新增一个种子浸药	["JZ361JX01","CCJHB"," 29.5","2018-02-11","2018-02-17","FOOD001"]
Invoke	initSeedling-spraydrug	新增一个种子喷药	["TLY83HN0117020101","CCJHB","20.5","2018-03-17","FOOD001"]
	initPlant	新增一个种植	["NT1ZLY171ZJ","1 号农田","2018-03-25","FOOD001"]
	initPlantusedrug	新增一个种植用药	["NT1ZLY171ZJ","CCJHB","9.2","防治稻瘟","2018-04-10","FOOD001"]
	initSeed2seedling	新增一个种子秧苗	["TLY83","TLY83HN0117020101","FOOD001"]
	initSeedling2plant	新增一个秧苗种植	["TLY83HN0117020101","NT1ZLY171ZJ ","FOOD001"]
	readOneById	根据 ID 读取一个记录	["CCJHB"]
	readOneByIdOther	根据 ID 调用另一个链码读取	["myccfoodall","readOneById","FOOD001"]
	readAll	根据范围查询一个类型的所有数据	["DRUG000","DRUG999"]
	readAllOther	根据范围调用另一个链码查询一个类型的所有数据	["myccfoodtall","readAll","PERSON000","PERSON999"]
	deleteOne	根据 ID 删除一个记录	["CCJHB"]

由于通道是排序服务创建的隔离不同链上交易的实例，加入到不同通道的节点将存储不同的数据。对于本系统而言，所有的数据都不需要隐私，因此只需要一个通道即可。但是由 3.1 节可知食品溯源过程涉及到很多的环节，Fabric 联盟链环境使用两个组织（Org1 和 Org2）进行测试，每个组织负责的环节不同，因此设计三个链码，分别为 myccfoodall、myccfoodone 和 myccfoodtwo，其中 myccfoodall 安装在所有参与节点上，用于操作 User 和 Person；myccfoodone 仅安装在 Org1 中的节点上，用于操作 Drug、Seed、Seedling、Seedsoakdrug、

Seedlingspraydrug 、 Plant 、 Plantusedrug 、 Seed2seedling 和 Seedling2plant ；myccfoodtwo 仅安装在 Org2 中的节点上，用于操作 Input 、 Warehouse 、 Warehouse2feed、Plant2input、Input2warehouse、Feed2product、Feed、Material、Product 和 Material2product。

表 5. myccfoodtwo 链码接口定义

Table 5. The myccfoodtwo chaincode interface definition

方法	分支方法	功能	参数示例
Init	无	无	[]
	initLedger	初始化账本，存入一些数据	[]
	initInput	新增一个入库	["NT1201707224","2018-04-22", "439906"," 2018-04-26", "FOOD001"]
	initWarehouse	新增一个仓库	["CK11","银欣米业厂"," 9676", "标准","FOOD001"]
	initWarehouse2feed	新增一个出库	["CK11","CK11DMJG,"FOOD001"]
	initPlant2input	新增一个种植入库批次转换	["NT1ZLY171ZJ ", " NT1201707224", "FOOD001"]
Invoke	initInput2warehouse	新增一个入库仓库批次转换	["NT1201707224"," CK11","FOOD001"]
	initFeed2product	新增一个进料产品批次转换	["CK11DMJG", "JC5CMGEN01 ","FOOD001"]
	initFeed	新增进料管理	["CK11DMJG"," 169440"," 14.5 ","3.5","2.5", "2018-04-27","FOOD001"]
	initMaterial	新增一个原料	["BSTMS0517","白砂糖", "10","四川王五糖业", "2018-04-30","FOOD001"]
	initProduct	新增一个产品	["JC5CMGEN","糙米羹","红枣味","360g*12","2018-05-02", "FOOD001"]
	initMaterial2product	新增一个原料产品批次转换	["BSTMS0517","JC5CMGEN"," FOOD001"]
	readOneById	根据 ID 读取一个记录	["JC5CMGEN"]
	readOneByIdOther	根据 ID 调用另一个链码读取	["myccfoodall","readOneById", "FOOD001"]
	readAll	根据范围查询一个类型的所有数据	["PRODUCT000","PRODUCT999"]
	readAllOther	根据范围调用另一个链码查询一个类型的所有数据	["myccfoodtall","readAll", "PERSON000","PERSON999"]
	deleteOne	根据 ID 删除一个记录	["JC5CMGEN"]

表 3、表 4 和表 5 分别为链码 myccfoodall、myccfoodone 和 myccfoodtwo 部分接口定义，其他的类似，接口由调用函数名称和调用参数两部分组成，链码在实例化时调用 Init 函数。

3.2 搭建 Hyperledger Fabric1.0 网络

开发操作系统为 Ubuntu16.04，其余环境要求如表 6 所示。搭建的 Fabric1.0 环境包括 1 个 Orderer 节点、4 个 Peer 节点（分属于两个组织）、2 个 CA 节点（每个组织各一个）、1 个客户端节点。所需 Fabric Docker 镜像有 fabric-peer、fabric-orderer、fabric-ca、fabric-tools、fabric-couchdb 等，可以使用源码中的 download-dockerimages.sh 下载全部镜像，需安装的库有 libtool、libltdl-dev。

表 6. 开发环境汇总

Table 6. Development environment summary

环境名称	版本
Go	V1.9.2
Docker	V17.11.0-ce
Docker-Compose	V1.12.0
Python-pip	V8.1.1
Node	V6.9.5
Npm	V3.10.10
Fabric	https://github.com/hyperledger/fabric/tree/release-1.0
Fabric CA	https://github.com/hyperledger/fabric-ca/tree/release-1.0

数据库使用可选的 CouchDB 替换默认的 LevelDB，CouchDB 是一种文档型数据库， 它的文档是无横式的（Schemaless），并不强制文档具有某种特定的结构。此外，CouchDB 可以使用字节数组和 JSON 数据进行操作，相对于 LevelDB（单个键查询、组合键查询和键范围查询）而言，CouchDB 还支持按字段的复杂查询。

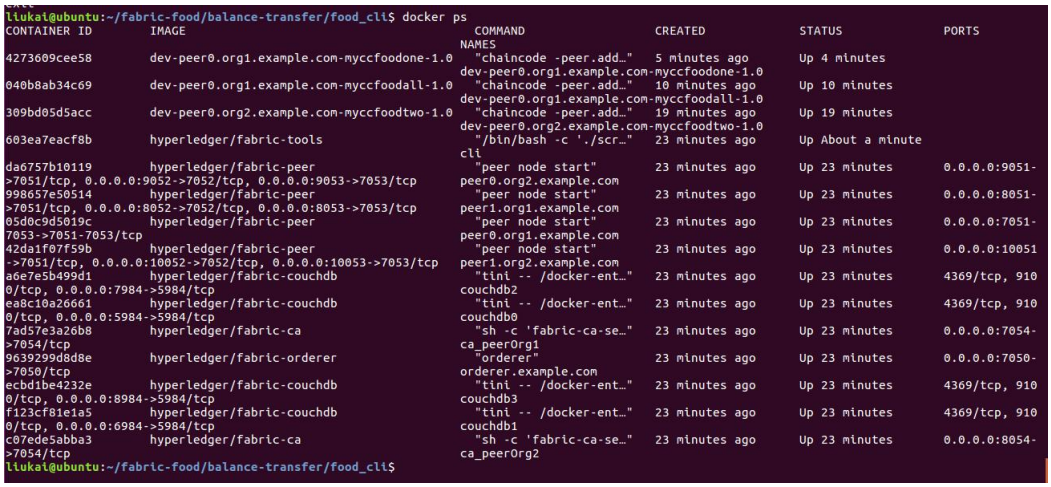

图 9 运行中容器

Fig.9 Running container

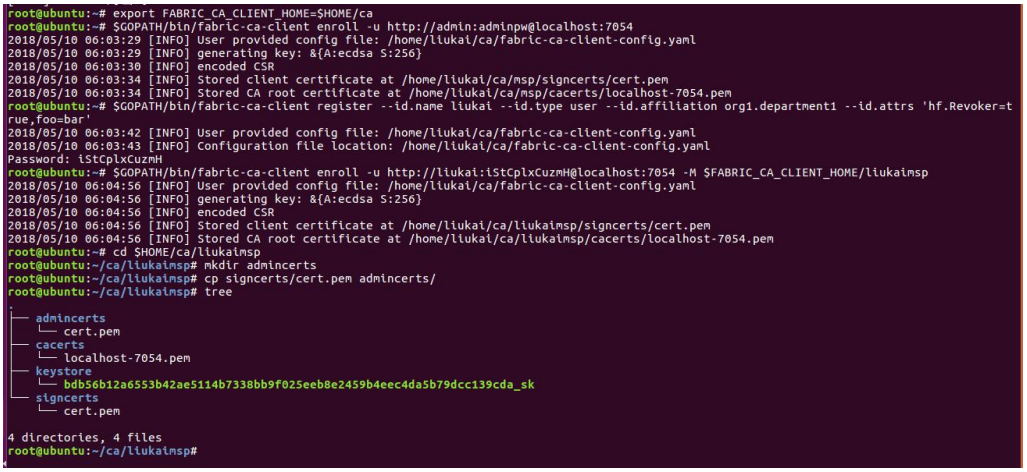

图 10 使用 CA Client 生成新用户

Fig.10 Use the CA Client to Generate New Users

环境搭建的主要过程参考附录 A，完成后运行的 Docker 容器如图 9 所示，另外使用步骤四的用户注册过程如图 10 所示。

3.3 搭建基于 Kafka 的排序服务

Fabric1.0 默认的共识是 Solo，即单节点共识，实际生产环境中需要搭建 Kafka 集群进行多节点共识。kafka 能够有序的管理信息，并且使参与副本能保存数据的一致性。kafka 集群需要使用 zookeeper 进行管理，为了能够满足 crash 容错，需最少搭建 4 台构成 kafka 集群，此时可容许一台出错。zookeeper 集群个数需为超过 1 的奇数，目的是防止 split-brain 和单节点故障出现。

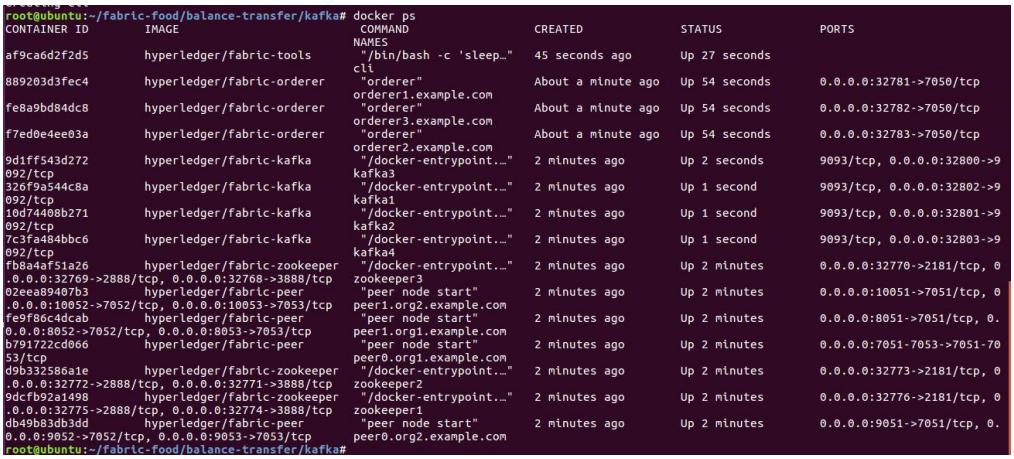

图 11 kafka 运行中容器

Fig.11 Kafka running container

集群按照依赖关系进行启动，即首先 zookeeper，其次 kafka，最后 Orderer 集群。在该环境中，Client 可将交易发给任意 Orderer 节点完成全局排序。搭建单机测试环境中有 3 个 zookeeper 节点、4 个 kafka 节点、3 个 orderer 节点、4 个 peer 节点，以及 1 个 cli 节点，完成后运行的 Docker 容器如图 11 所示。

4. 基于区块链技术的溯源实现

4.1 食品信息初始化

编写链码的语言主要有 Go 和 Java，由于 Fabric 本身由 Go 开发，因此大多数使用 Go 进行链码的开发。Go 语言编写的链码需包含一个名为 SimpleChaincode 的结构体，还需含有三个基本函数 main、Init 和 Invoke。其中 main 是链码入口函数，其他函数中的 shim.ChaincodeStubInterface 参数包含大量的接口方法，主要是有关链码调用参数解析、交易信息解析、操作状态数据、调用其他链码、事件处理以及与辅助操作的方法， 常用接口和具体说明如表 7 所示。

表 7. 链码常用接口用法

Table 7. Common interface usage for chaining

接口名称	说明
GetArgs() byte	得到调用函数名称和参数列表
GetCreator()	获得当前用户
PutState(key string, value []byte) error	把键值对放入状态数据库中
GetState(key string) ([]byte, error)	根据键查询对应的值
DelState(key string) error	根据键删除状态数据库中的数据
GetStateByRange(startKey,endKey string) (StateQueryIteratorInterface, error)	按区间左闭右开查询键对应的值
CreateCompositeKey() (string, error)	生成复合键
SplitCompositeKey() (string, []string, error)	拆分复合键

如果使用 CouchDB，还可以使用更加丰富的 GetQueryResult(query string)。另外，本系统中的链码均可以通过 InvokeChaincode(chaincodeName string, args [][]byte, channel string)调用其他链码，这样可以进行模块化编程，需要将[]string 转成[][]byte 调用。如果被调用的链码和调用的链码不在相同的链上，那么便会生成一个新的交易模拟器，返回调用执行的 Response 给链码，但是并不会增加读写集到交易中。因此，对于调用不同链的链码只能查询状态数据，而不能写入。

本系统中的三个链码均有一个 initLedger 分支方法进行账本的初始化工作，主要是存入一些数据，由于账本使用非关系型数据库（NoSQL），因此查询数据比较麻烦，需要编写复合键。由于只能进行按照键的范围查询，为了实现查询某个类型变量的所有数据，需要按照某种格式作为键存放，比如对于药品信息的键均为“DRUG+编号”的形式，为了可以按照药品 ID 查询数据，还需要将药品 ID 和“DRUG+编号”分别作为键和值放入账本中。

由于在定义 yaml 文件时为 CouchDB 定义了映射端口，因此可以通过“http://ip:5984/_utils/”访问 CouchDB，但一般在生产环境中，为了账本的安全，并不需要开放端口，本文开放端口的主要目的是为了对数据进行演示和观察。

CouchDB 的 Web 管理界面如图 12 所示，点击通道名 mychannel 能够查看具体的数据库信息，信息如图 13 所示。由于数据格式为“链码名称\u0000 数据”，需用%00 替换\u0000。

因此使用“cur http://ip:port/mychannel/myccfoodall%00PERSON1”命令可以查询具体的一条数据。然后使用“curl -X PUT http://ip:port/mychannel/myccfoodall%00PERSON1 -d '{"_id":"myccfoodall\u0000PERSON1","_rev":"105bef6d99dda7d364b516db8d22647ae","chaincodeid":"myccfoodall","data":{"personid":"FOOD00264","name":"夏良军","sex":"男","workplace":"新地点","job":"监仓","password":"123"},"version":"4:0"}'”命令篡改 World State 数据，此时该节点的状态数据就与其他节点的不同了，Block Ledger 数据并不会发生变化，通过客户端对各节点的查询便可以发现错误。

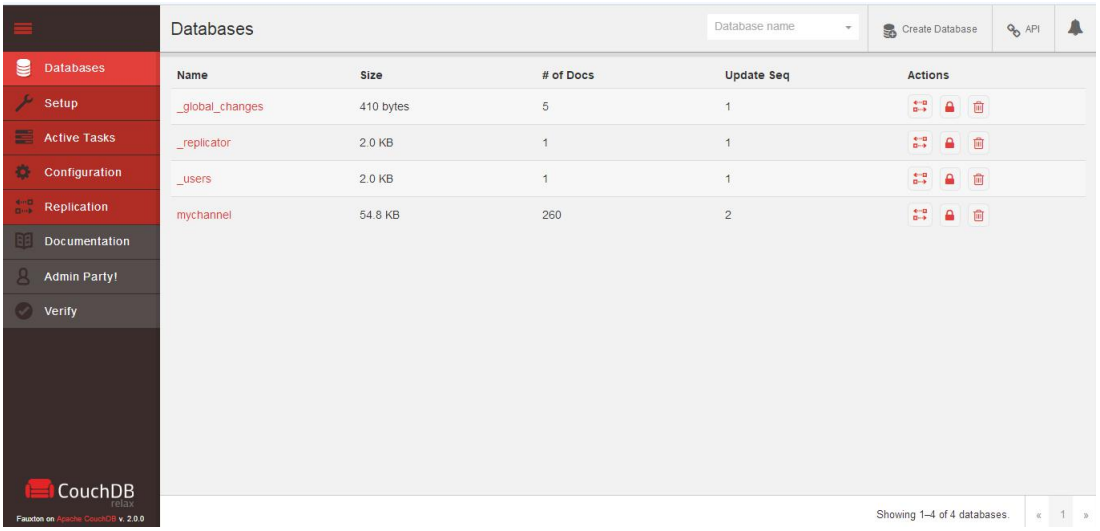

图 12 CouchDB 的 Web 管理界面

Fig.12 CouchDB Web Management Interface

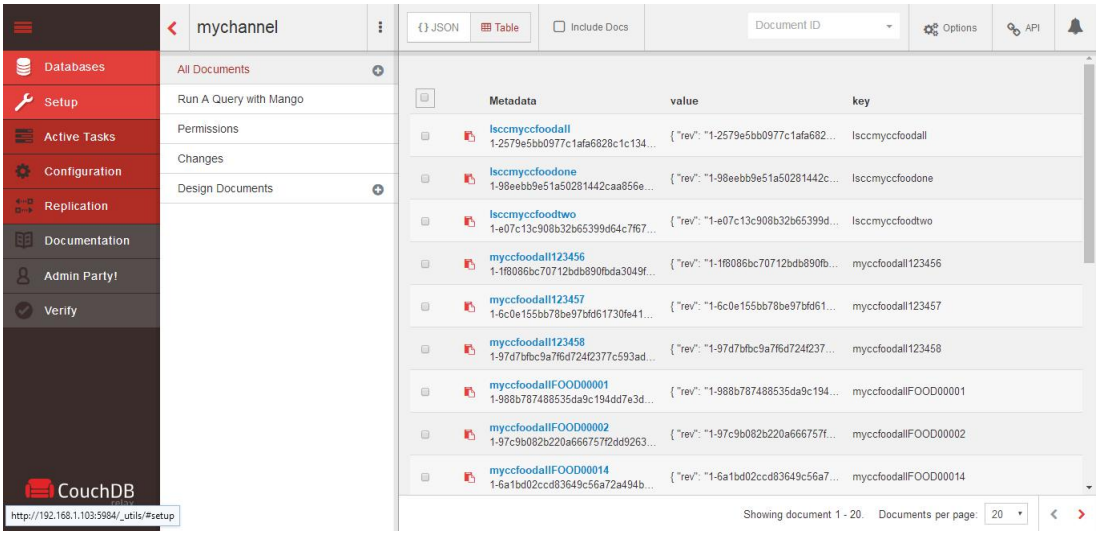

图 13 CouchDB 的 mychannel 数据库

Fig.13 CouchDB's mychannel database

4.2 食品溯源测试

4.2.1 用户基础操作

图 14 用户注册

Fig.14 User registration

普通用户可以通过注册来进行食品信息的查询，由于一般用户只能进行查询工作，为简便用户使用，并不需要使用 CA 注册，使用默认已有的证书进行签名和认证，用户在系统中的注册界面如图 14 所示，注册成功后使用 id 和 password 进行登录，登录成功后进入的系统首页如图 15 所示，用户可以很方便的查询各个节点的质检员信息、育秧相关信息、种植相关信息、仓储相关信息、粗加工相关信息，以及深加工相关信息，各信息使用表格进行展示，比如质检员信息如图 16 所示，在“search”输入框中可以通过关键字进行信息的查询，在上方的下拉框中可以选择网络中的节点，之后点击“开始查询”按钮便可更新为所需节点的 World State 数据。

图 15 食品溯源系统首页

Fig.15 Food Traceability System Home

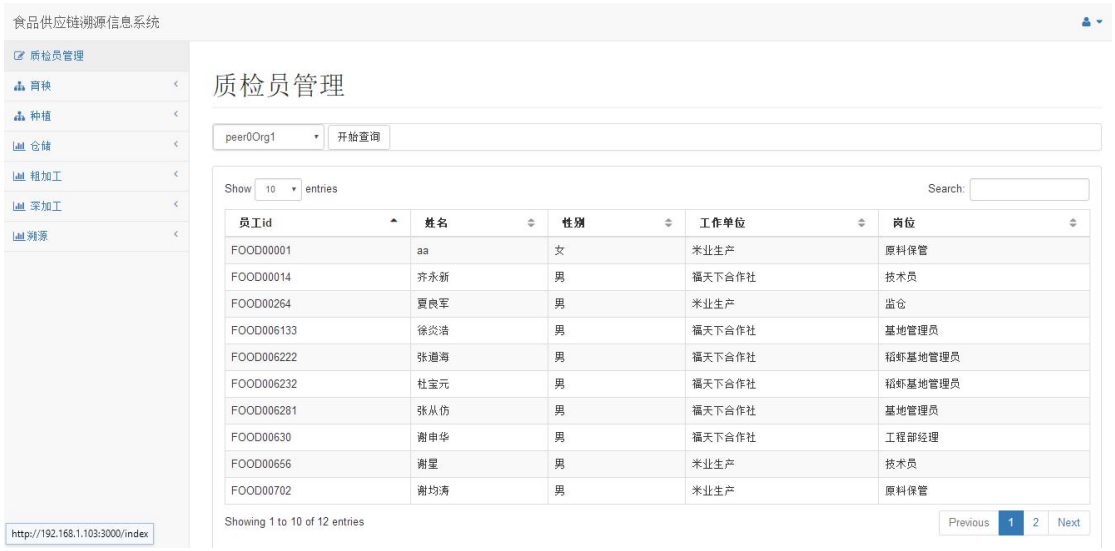

图 16 查询食品相关信息

Fig.16 Query food related information

4.2.2 真实性溯源

用户登录系统之后可以进行正向溯源和反向溯源，按照种子 ID 进行溯源即为正向溯源，也可以称为真实性溯源。种子到产品共经历了七个阶段，分别为种子、秧苗、种植、收割入库、仓储、进料和产品。种子和产品的关系是多对多，即一个种子最终可能会演变为多个产品，因为不同的产品可能用到的某个原材料是一样，同时一个产品也可能对应多个种子，这是因为产品往往是由多个原材料制造的。

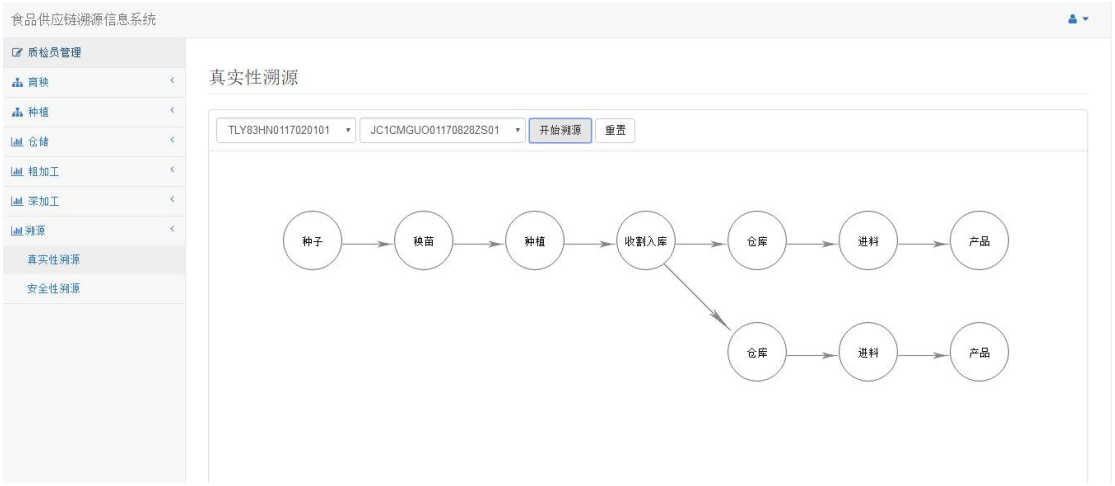

图 17 正向溯源

Fig.17 Forward tracing

如图 17 所示，从下拉框中选择一个种子 ID，点击“开始溯源”按钮，便会显示整个种子到产品的流程，点击“重置”按钮可以重新开始溯源。另外，点击具体的某个阶段所在的圆形区域，则可以查看具体的信息，信息的格式为“与类型相关的所有属性+对应的值”，比如与种子相关的类型有种子、种子浸药和药品信息，因此显示的种子信息为“seedid:TLY83HN0117020101，variety:潭两优 83，type:早稻，name:消毒剂，dosage:液态，standard:国标，effect:消毒，concentration:32，startdate:2017-03-17，enddate: 2017-03-18”，其他类型的信息与其类似，具体内容如图 18 所示。

图 18 正向溯源具体食品相关信息

Fig.18 Forward traceability of specific food-related information

4.2.3 安全性溯源

按照产品 ID 进行溯源即为反向溯源，也可以称为安全性溯源，相比于真实性溯源，安全性溯源的作用更大。在安全性溯源界面中，将食品安全的风险按照颜色分为绿色、黄色和红色三个等级，绿色表示很安全，黄色表示相对安全，红色表示表示有问题存在。溯源过程为从下拉框中选择一个产品 ID，点击“开始溯源”按钮，便会显示产品到种子的整个流程，由于按照产品 ID 溯源到一开始的种子 ID 时，可能该种子按照正向溯源可能会溯源到其他的产品，因此如果发现存在其他产品便使用弹框提示用户，具体信息如图 19 所示。输入框中可以输入“1”和“2”两个数字，其中“1”表示该产品检测结果正常，“2”表示该产品检验发现质量安全问题，如果输入“1”，那么结果如图 20 所示，默认食品溯源每个阶段均使用黄色显示，如果其他产品检验结果正常，那么两个产品从同一个种子 ID 出发的相同阶段标记为绿色，表示这些阶段很安全。相对而言，如果输入“2”，那么结果如图 21 所示，相同阶段标记为红色，表示这些阶段存在很

大的安全隐患。另外，与真实性溯源类似，点击具体的某个阶段所在的圆形区域，可以查看具体的信息，具体内容如图 22 所示。

图 19 反向溯源

Fig.19 Reverse tracing

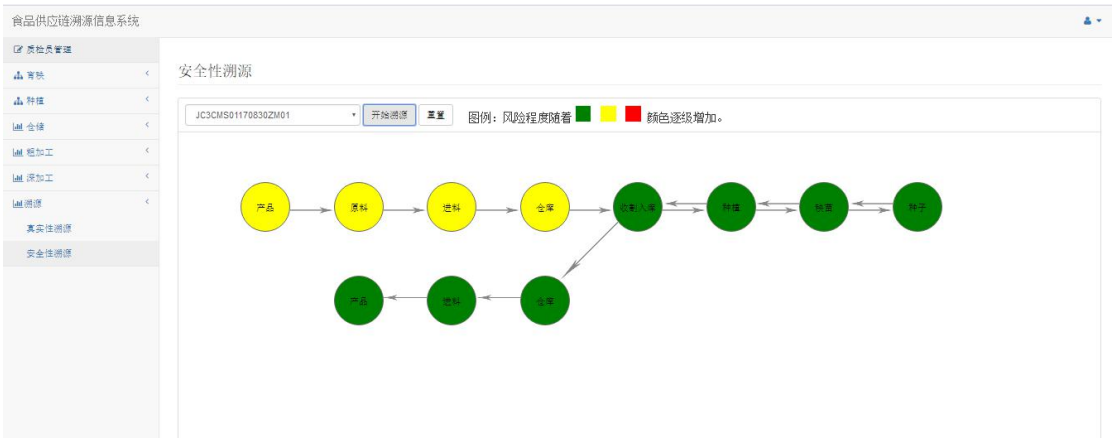

图 20 反向溯源情况一

Fig.20 Reverse tracing case one

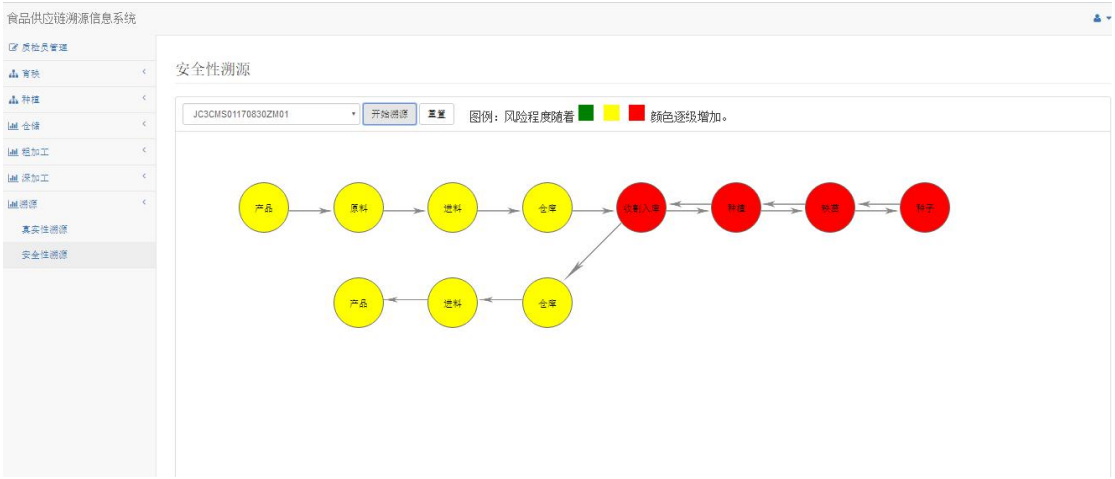

图 21 反向溯源情况二

Fig.21 Reverse tracing case two

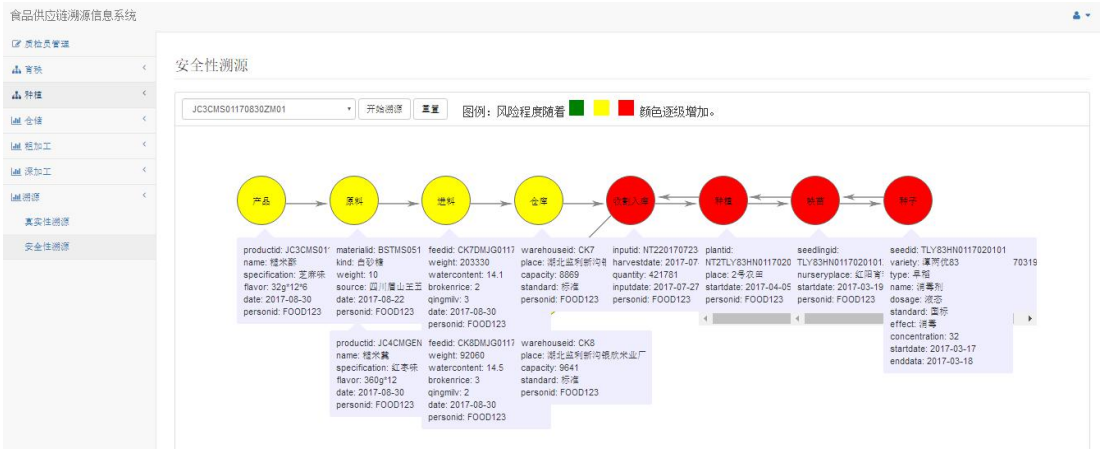

图 22 反向溯源具体食品相关信息

Fig.22 Reverse tracing specific food related information

4. 3 Node.js SDK 测试

参考官网中的示例，使用 Node.js SDK 与 Fabric 网络进行交互，首先需要启动 Node 网络服务，使用 API 进行操作主要功能有注册用户、创建通道、节点加入通道、安装链码、初始化链码、执行链码以及对账本的各种查询（按照交易 ID 查询交易详细信息、按照区块编号查询具体区块信息及查询链信息），返回的信息为 JSON 格式。

首先，使用 API 时需要使用用户证书，进行用户注册时会返回一个 Token，基于 JSON Web Token 实现，用于在 HTTP 通信过程中进行身份认证。生成的公私钥存放在/tmp/fabric-client-kvs_peerOrg1 和/tmp/fabric-client-kvs_peerOrg2 文件夹中，API 访问方式为“ORG1_TOKEN=$(curl -s -X POST http://ip:4000/users \
-H "content-type: application/x-www-form-urlencoded" \
-d 'username=Jim&orgName=org1') ”
与“ORG1_TOKEN=$(echo $ORG1_TOKEN | jq ".token" | sed "s/\"//g")”，具体内容如图 23 所示。然后可以使用 API 传入 channel. tx 的路径进行创建通道、将各组织的 Peer 节点加入通道中、为 Peer 节点安装特定的链码、初始化链码等，这些内容也可以使用脚本文件 scripts. sh 操作，下面是一些使用 API 进行的查询操作，通过 API 可以很容易查询与区块链有关的各类信息。

```
liukai@ubuntu:~$ ORG1_TOKEN=$(curl -s -X POST \
>   http://localhost:4000/users \
>   -H "content-type: application/x-www-form-urlencoded" \
>   -d 'username=Jim&orgName=org1')
liukai@ubuntu:~$ echo $ORG1_TOKEN
{"success":true,"secret":"iSRdHBdKMogu","message":"Jim enrolled Successfully","token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1MjU5MDk
5OTEsInVzZXJuYW1lIjoiSmltIiwib3JnTmFtZSI6Im9yZzEiLCJpYXQiOjE1MjU4NzM5OTF9.FiH8CwVtdBxddNOrlGg39-qM0tQGP3ZdWqco0NKiDyY"}
liukai@ubuntu:~$ ORG1_TOKEN=$(echo $ORG1_TOKEN | jq ".token" | sed "s/\"//g")
liukai@ubuntu:~$ ORG2_TOKEN=$(curl -s -X POST \
>   http://localhost:4000/users \
>   -H "content-type: application/x-www-form-urlencoded" \
>   -d 'username=Barry&orgName=org2')
liukai@ubuntu:~$ echo $ORG2_TOKEN
{"success":true,"secret":"fHIKrYusSbhe","message":"Barry enrolled Successfully","token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1MjU5M
TAwMzEsInVzZXJuYW1lIjoiQmFycnkiLCJvcmdOYW1lIjoib3JnMiIsImlhdCI6MTUyNTg3NDAzMX0.S0sbSrgDoOhBYLBjEcQbMnHGKNoSGjS4SW5GIxTZ5Ao"}
liukai@ubuntu:~$ ORG2_TOKEN=$(echo $ORG2_TOKEN | jq ".token" | sed "s/\"//g")
liukai@ubuntu:~$
```

图 23 注册新用户

Fig.23 Register new user

首先，API 中最重要的便是操作链码，即 Invoke 和 Query。首先通过链码可以查询信息，API 访问需要的参数为 Channel 名称、Peer 编号、链码名称、链码方法和方法参数，具体的访问方式为“curl -s -X POST\
http://ip:4000/channels/mychannel/chaincodes/myccfoodtwo/query \
-H "authorization: Bearer $ORG2_TOKEN" -H "content-type: application/json" \
-d '{"peers":["peer1"],"fcn":"readAll","args":["PRODUCT000","PRODUCT999"] }'”，返回的信息为查询的结果值。同样的，也可以进行 invoke，具体的访问方式为“curl -s -X POST http://ip:4000/channels/mychannel/chaincodes/myccfoodall/invoke \
-H "authorization: Bearer $ORG1_TOKEN" -H "content-type: application/json" \
-d '{ "peers":["peer1"],"fcn":"initPerson","args":["FOOD888","张三","男","米业生产","监仓","123"]}'”，操作成功将返回的交易 ID，后台输出信息分别如图 24 和图 25 所示。

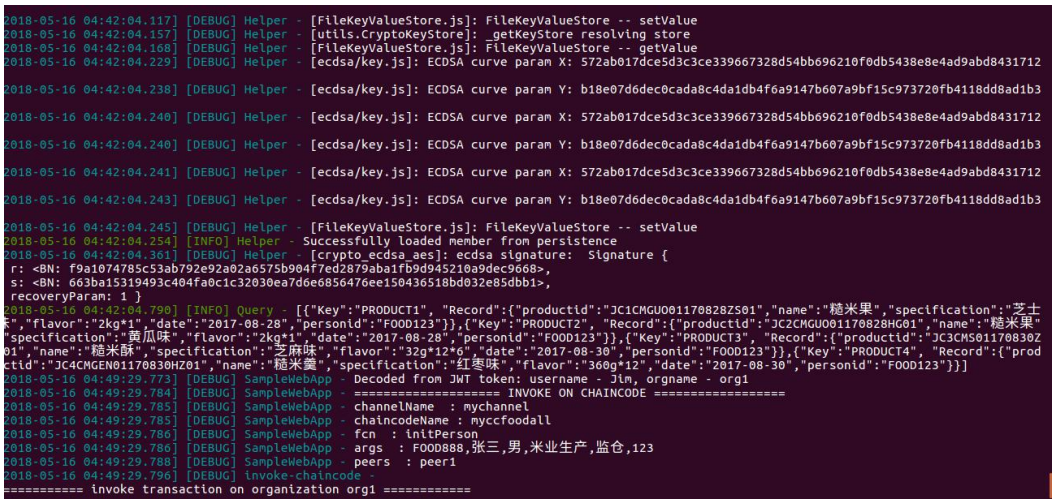

图 24 查询链码后台信息

Fig.24 Query chaincode background information

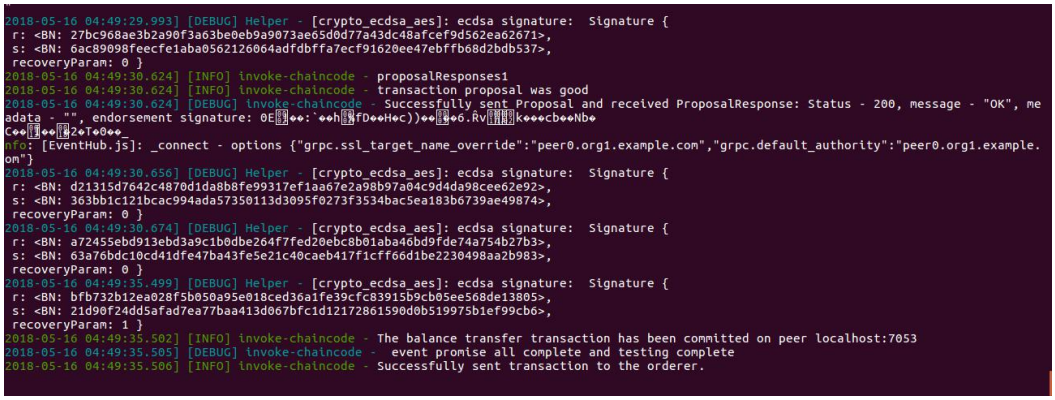

图 25 执行链码后台信息

Fig.25 Invoke chaincode background information

使用 API 可以查询某个节点已经安装和实例化的链码，返回信息为链码的名称、版本和路径，查询已经安装链码的 API 访问方式为“curl -s -X GET \
"http:// ip:4000/chaincodes?peer=peer1&type=installed" \
-H "authorization: Bearer $ORG1_TOKEN" -H "content-type: application/json"”，结果如图 26 所示；查询已经实例化链码的 API 访问方式为“curl -s -X GET \
"http:// ip:4000/chaincodes?peer=peer1&type=instantiated" \
-H "authorization: Bearer $ORG1_TOKEN" -H "content-type: application/json"”，结果如图 27 所示。

图 26 查询某个节点安装的链码

Fig.26 Query the chaincode installed by a node

图 27 查询某个节点实例化的链码

Fig.27 Query the node's instantiated chaincode

链结构（Chain）与 Channel 是一一对应的， Channel 是 Fabric 的一个重要特性，它与消息队列中的主题概念很相似，只有加入其中的 Peer 才能收到其中的信息，一个 Peer 加入了几个 Channel 中，那么该 Peer 节点就维护几个链的信息。API 访问需要的参数为 Peer 编号和 Channel 名称，具体的访问方式为
“curl -s -X GET "http://ip:4000/channels/mychannel?peer=peer1" \
-H "authorization: Bearer $ORG1_TOKEN" -H "content-type: application/json"”，结果如图 28 所示，从中可以看出当前链的高度为 9，还有 currentBlockHash 等信息。如果执行一些调用链码的后，再次进行查询，则结果如图 29 所示，其中链的高度已经变成了 10。

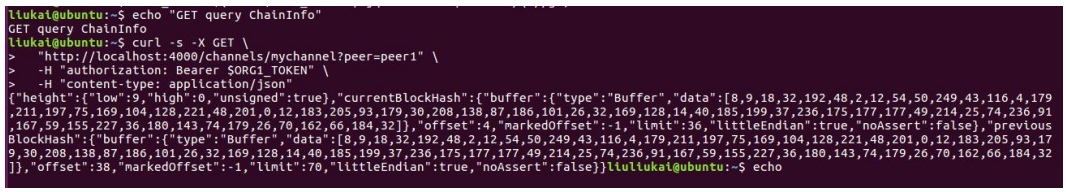

图 28 查询链信息

Fig.28 Query chain information

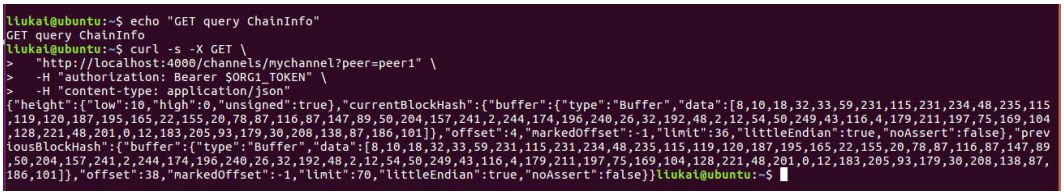

图 29 再次查询链信息

Fig.29 Query chain information again

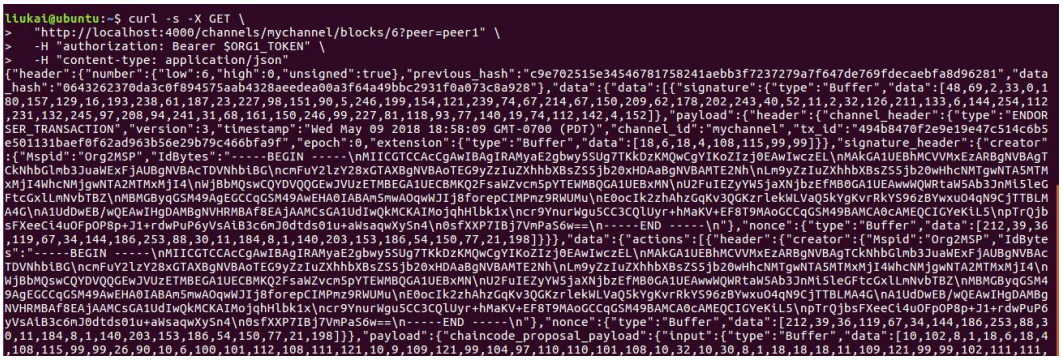

图 30 查询指定的区块信息

Fig.30 Query the specified block information

通过查询链的信息可以知道链的高度，然后可以根据区块序号查询具体的区块信息，API 访问需要的参数为 Channel 名称、区块序号和 Peer 编号，具体的访问方式为“curl -s -X GET "http://ip:4000/channels/mychannel/blocks/1?peer=peer1" \ -H "authorization: Bearer $ORG1_TOKEN" -H "content-type: application/json"”，返回结果如图 30 所示，区块结构主要包括区块头（Header）、数据域（Data）和元数据域（Metadata）三个部分。其中区块头部分用于创建数据结构，主要包括区块序号（Number）、前一个区块头部的 Hash 值（PreviousHash），以及本区块的 Hash 值（DataHash）。数据域使用信封（Envelope）结构记录交易数据，各交易按照 Merkle Tree 结构存放。元数据域存放的是一些辅助信息，包括签名信息（Signatures）、通道最新配置区块的索引（LastConfig）、通道排序服务信息（Orderer）等。

另外，每次调用链码进行交易都会获得一个交易 ID（TransactionID），通过交易 ID 可以查询到具体的交易信息，API 访问需要的参数为 Channel 名称、交易 ID 和 Peer 编号，具体的访问方式为“curl -s -X GET \
http://ip:4000/channels/mychannel/transactions/77cc9d7ea0c881944e73a4823419ffbb83fb383c5c5910cfc9659bc8252624af?peer=peer1 \
-H "authorization: Bearer $ORG1_TOKEN" -H "content-type: application/json"”，返回结果如图 31 所示，交易信息处理签名头部（SignatureHeader）和背书结构（执行交易得到结果的读写集和签名）外，还有与该交易相关的类型（Type）、时间戳（Timestamp）、链码信息、通道信息和调用参数等。

图 31 按照交易 ID 查询交易信息

Fig.31 Query transaction information by transaction ID

5. 总结与展望

本文首先通过对相关文献综述，总结了当前溯源系统的主要特点和应用情况，分析了传统追溯系统存在的一些问题。然后，通过对区块链技术的工作原理和技术特征进行总结分析，论述了将区块链技术应用于食品溯源系统的必要性和可行性。针对当前区块链技术的发展情况以及相关主流的应用，使用超级账本 Fabric 构建了一个简易的基于区块链的食品安全溯源系统。系统主要分为四个层次，由下到上分别为区块链底层平台、智能合约层、业务层和应用层。使用 Go 语言编写操作账本的链码，Web 应用采用 express 框架进行搭建，调用 Node. js SDK 与区块链网络完成通信。通过对溯源系统的测试，可以防止节点对数据的篡改，设计的溯源系统阶段主要为种子到产品的种植生产过程，系统中的相应组织的信息录入、查询、用户注册等功能均通过测试。

传统食品溯源系统的数据往往由其中的核心机构进行单独维护，缺乏必要的验证机制，存在数据被篡改的情况。在当今社会形势下，广大群众对食品安全的要求越来越强烈，这对食品可追溯系统的构建提出了更严峻的要求。目前利用区块链构建食品安全溯源的相关案例还并不多，超级账本 Fabric 提供了面向联盟链范畴的分布式账本平台实现，实现了比较完备的权限管理功能。本文构建的溯源系统每个组织负责的不同的食品环节，各自维护不同的区块链节点，这些节点共同组成了区块链网络。使用 Fabric 作为底层技术对录入的数据完整性进行验证，利用网络的共识机制保障了账本的一致性，如果有节点篡改数据可以通过查询和验证的方式快速发现。进而系统也可以搭配其他监督管理手段，迫使参与到溯源系统的组织无法篡改数据。即将溯源系统部署方式由传统的中心化管理模式改为

多中心共同管理，提高了食品信息的真实性和安全性。相对于公有链而言，采用联盟链能够合理的调整区块链中的去中心化程度与共识机制，从而可以针对性解决溯源系统中信息录入方式不统一、数据存储不安全以及信息交换时隐私无法得到保障等问题。也使得区块链技术在食品溯源系统发挥优势的同时，并不会给服务器带来较大负担。

综上所述，由于自身知识和研究水平有限，本文所构建的食品溯源系统还比较简易，主要利用分布式账本技术解决了信息不安全问题，但应用区块链的好处还不仅仅如此。另外系统也只能够保证在数据录入系统后，如果数据违反设定的规则和要求，以及篡改数据信息时，就能够通过系统逻辑和区块链特性查找出来。但是对于溯源系统，最重要的应该保证数据源的真实性，未来研究的着重点应该在此，即设计更加严格的系统机制防止提供虚假数据。比如需要上传身份信息验证，如果发现数据造假将会得到极大的惩罚等。因此，当前系统还存在一些不足之处，根据以往新兴技术的发展历程，可以预知在不久后将会出现较为成熟的基于区块链的食品安全溯源系统。

参考文献

1.陈欢，方越锋．探讨我国食品溯源的应用现状及优化建议．中国集体经济，2016，01:110-111

2.董宁, 朱轩彤．区块链技术演进及产业应用展望．信息安全研究，2017, 3:200-210

3.何渝君，龚国成．区块链技术在物联网安全相关领域的研究．电信工程技术与标准化，2017，30:12-16

4.何静，刘启强，赵恒煜．食品安全溯源与监管的国内外研究综述．广东科技，2014，23:208-210

5.黄若君．我国食品安全现状及存在问题分析．沿海企业与科技，2013，05:13-16

6.李子晨．阿里依靠区块链助力全球食品溯源．2017-04-07．http://finance.china.com.cn/news/gjjj/20170328/4154024.shtml

7.李想，石磊．行业信任危机的一个经济学解释：以食品安全为例．经济研究，2014，49:169-181

8.彭远斌, 丛新元, 杨民. 食品溯源系统的物品编码设计．现代农业科技，2014，01:295-297

9.孙志国，李秀峰，王文生，冀智强．区块链技术在食品安全领域的应用展望．农业网络信息，2016，12:30-31

10.王崇民, 王翠竹. 区块链来了，供应链透明化还远吗. 食品安全导刊，2018，21:56-57

11.邢文英. 美国的农产品质量安全可追溯制度. 世界农业, 2006，04:39-41

12.杨宝华，陈昌. 区块链原理、设计与应用. 北京：机械工业出版社，2017. 75-82，249-289

13.张增骏，董宁，朱轩彤，陈剑雄. 深度探索区块链：Hyperledger 技术与应用. 北京：机械工业出版社. 2018. 16-19

14. Aiello D, De L D, Gionfriddo E, et al. Review: multistage mass spectrometry in quality, safety and origin of foods. European Journal of Mass Spectrometry, 2011, 17:1

15. Andreas M. Antonopoulos.Mastering Bitcoin. 2nd Edition. O'Reilly Media, 2017.287-294

16. Engeljohn D. Continuous Food Safety Innovation as a Management Strategy: Public Perspective (PowerPoint). General Information, 2007

17.Ethereum White-Paper. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.2015. https://github.com/Ethereum/wiki/wiki/White-Paper

18. Hyperledger Fabric Docs.A Blockchain Platform for the Enterprise: Hyperledger Fabric.2017, https://hyperledger-fabric.readthedocs.io

19. Hyperledger White Paper. 2016-07-18. http://www.8btc.com/hyperledger-whitepaper

20. Inerney B M, Corkery G, Ayalew G, et al. A preliminary in vivo study on the potential application of a novel method of e-tracking in the poultry food chain and its potential impact on animal welfare. Computers & Electronics in Agriculture, 2011, 79:51-62.

21. Kim Y G, Woo E. Consumer acceptance of a quick response (QR) code for the food traceability system: Application of an extended technology acceptance model (TAM). Food Research International, 2016, 85:266-272

22. Koutsoumanis K P, Gougouli M. Use of Time Temperature Integrators in food safety management. Trends in Food Science & Technology, 2015, 43:236-244.

23.Ravikant Agrawal. CoinDesk Research's 2017 State of Blockchain report.2017.https://media.coindesk.com/uploads/2017/09/state_of_blockchain_q2_2017.pdf

附录 A 环境搭建

步骤零：安装 Go、Docker 和 Docker-Compose 等必要开发工具，下载 Fabric 和 Fabric CA 源码，以及 Fabric Docker 镜像。

步骤一：配置 crypto-config.yaml（配置组织和节点关系）、configtx.yaml（配置 Peer 的 MSP、锚节点，以及 Orderer 的共识算法、区块大小、超时时间等）、docker-compose-cli-couchdb.yaml（对 Orderer、Peer、CA、CLI、CouchDB 节点进行配置）、scripts.sh 脚本等文件，使用 make 编译生成工具 cryptogen 与 configtxgen。

步骤二：使用“cryptogen generate --config=./crypto-config.yaml”命令生成公私钥和证书；使用“configtxgen -profile TwoOrgsOrdererGenesis -outputBlock ./channel-artifacts/genesis.block”命令生成 Orderer Service 创世区块；使用“configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID mychannel”命令生成通道配置交易；使用“configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID mychannel -asOrg Org1MSP”和“configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID mychannel -asOrg Org2MSP”命令生成锚节点（AnchorPeer）配置。

步骤三：使用“docker-compose -f docker-compose-cli-couchdb.yaml up -d”命令启动网络所有节点，然后使用脚本文件 scripts.sh 或者 Node.js SDK 初始化 Fabric 环境，主要涉及到的环节有创建 Channel、将各个 Peer 加入 Channel、更新组织锚节点信息、Install ChainCode、Instantiate ChainCode 等，可使用“docker logs -f cli”查看日志信息。

步骤四：在步骤二中会默认为每个组织生成一个管理员和一个用户证书，实际生产环境中需要使用 CA Client 生成新用户，可以通过客户端命令或者 RESTful API（服务前缀为/api/v1）进行操作。在客户端使用“fabric-ca-client register --id.name xx --id.type user --id.affiliation org1.department1 --id.attrs 'hf.Revoker=true,foo=bar'”命令注册用户，系统会返回一个密码，也可使用“—id.secret”指定密码。收到密码之后使用 enroll 命令，给新用户生成 MSP 的私钥和证书。

附录 B 关键代码

foodsafetyall_chaincode .go

```
package main

import (
	"bytes"
	"encoding/json"
	"fmt"
	"strconv"
	"github.com/hyperledger/fabric/core/chaincode/shim"
	pb "github.com/hyperledger/fabric/protos/peer"
)
type SimpleChaincode struct {
}
type Value struct {
	Valueid string `json:"valueid"` //保存 id，根据 ID 查询数据
}
//用户
type User struct {
	Userid	string `json:"userid"`	//用户 id,主键
	Name	string `json:"name"`	//姓名
	Password string `json:"password"` //密码
}
//质检员
type Person struct {
	……
}
func main() {
	err := shim.Start(new(SimpleChaincode))
	if err != nil {
		fmt.Printf("Error starting Simple chaincode: %s", err)
	}
}

// 初始化链码
func (t *SimpleChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {
	return shim.Success(nil)
}

func (t *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
	function, args := stub.GetFunctionAndParameters()
```

```
    fmt.Println("invoke is running " + function)
    if function == "deleteOne" { //删除一个记录
        return t.deleteOne(stub, args)
    } else if function == "readOneById" { //根据 id 读取一个记录
        return t.readOneById(stub, args)
    } else if function == "readOneByIdOther" { //根据 id 调用另一个链码读取
        return t.readOneByIdOther(stub, args)
    } else if function == "readAll" { //根据范围查询一个类型的所有
        return t.readAll(stub, args)
    } else if function == "readAllOther" { //根据范围调用另一个链码查询一个类型的所有
        return t.readAllOther(stub, args)
    } else if function == "initLedger" { //初始化所有类型的数据
        return t.initLedger(stub)
    } else if function == "initPerson" { //新增一个质检员
        return t.initPerson(stub, args)
    } else if function == "initUser" { //新增一个用户
        return t.initUser(stub, args)
    }
    fmt.Println("invoke did not find func: " + function)
    return shim.Error("Received unknown function invocation")
}
func (t *SimpleChaincode) initLedger(stub shim.ChaincodeStubInterface) pb.Response {
    persons := []Person{
        Person{Personid: "FOOD00264", Name: "夏良军", Sex: "男", Workplace: "米业生产",
Job: "监仓", Password: "123"},
    }
    i := 0
    for i < len(persons) {
        fmt.Println("i is ", i)
        personAsBytes, _ := json.Marshal(persons[i])
        //把 Personid 作为键
        newvalueAsBytes, _ := json.Marshal(&Value{"PERSON" + strconv.Itoa(i+1)})
        stub.PutState("PERSON"+strconv.Itoa(i+1), personAsBytes)
        //放入主键 ID，可以根据 ID 查询
        stub.PutState(persons[i].Personid, newvalueAsBytes)
        fmt.Println("Added", persons[i])
        i = i + 1
    }
    //存放最大标号的下一个，便于插入新数据
    valueAsBytes, _ := json.Marshal(&Value{strconv.Itoa(i)})
    stub.PutState("YPERSON", valueAsBytes)
    //初始化用户
    ……
    return shim.Success(nil)
}
```

```
func (t *SimpleChaincode) readOneById(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    var id, jsonResp string
    var err error
    if len(args) != 1 {
        return shim.Error("Incorrect number of arguments. Expecting name of the marble to query")
    }
    id = args[0]
    //由 ID 得到新键
    valAsbytes, err := stub.GetState(id)
    if err != nil {
        jsonResp = "{\"Error\":\"Failed to get state for " + id + "\"}"
        return shim.Error(jsonResp)
    } else if valAsbytes == nil {
        jsonResp = "{\"Error\":\"ValueID does not exist: " + id + "\"}"
        return shim.Error(jsonResp)
    }
    valueToTransfer := Value{}
    err = json.Unmarshal(valAsbytes, &valueToTransfer) //unmarshal it aka JSON.parse()
    if err != nil {
        return shim.Error(err.Error())
    }
    fmt.Println("id" + valueToTransfer.Valueid)
    valAsbytes2, err := stub.GetState(valueToTransfer.Valueid)
    if err != nil {
        jsonResp = "{\"Error\":\"Failed to get state for " + valueToTransfer.Valueid + "\"}"
        return shim.Error(jsonResp)
    } else if valAsbytes2 == nil {
        jsonResp = "{\"Error\":\"ValueID does not exist: " + valueToTransfer.Valueid + "\"}"
        return shim.Error(jsonResp)
    }
    return shim.Success(valAsbytes2)
}

func (t *SimpleChaincode) readAll(stub shim.ChaincodeStubInterface, args []string) pb.Response {

    if len(args) != 2 {
        return shim.Error("Incorrect number of arguments. Expecting 2")
    }
    startKey := args[0]
    endKey := args[1]
    resultsIterator, err := stub.GetStateByRange(startKey, endKey)
```

```
    if err != nil {
        return shim.Error(err.Error())
    }
    defer resultsIterator.Close()

    var buffer bytes.Buffer
    buffer.WriteString("[")
    bArrayMemberAlreadyWritten := false
    for resultsIterator.HasNext() {
        queryResponse, err := resultsIterator.Next()
        if err != nil {
            return shim.Error(err.Error())
        }
        if bArrayMemberAlreadyWritten == true {
            buffer.WriteString(",")
        }
        buffer.WriteString("{\"Key\":")
        buffer.WriteString("\"")
        buffer.WriteString(queryResponse.Key)
        buffer.WriteString("\"")
        buffer.WriteString(", \"Record\":")
        // Record is a JSON object, so we write as-is
        buffer.WriteString(string(queryResponse.Value))
        buffer.WriteString("}")
        bArrayMemberAlreadyWritten = true
    }
    buffer.WriteString("]")
    fmt.Printf("- readAllPerson:\n%s\n", buffer.String())
    return shim.Success(buffer.Bytes())
}

func (t *SimpleChaincode) initUser(stub shim.ChaincodeStubInterface, args []string) pb.Response
{
    var err error
    if len(args) != 3 {
        return shim.Error("Incorrect number of arguments. Expecting 3")
    }
    //检查用户是否存在
    userAsBytes, err := stub.GetState(args[0])
    if err != nil {
        return shim.Error("Failed to get user: " + err.Error())
    } else if userAsBytes != nil {
        fmt.Println("The user already exists: " + args[0])
        return shim.Error("The user already exists: " + args[0])
    }
```

```
    user := &User{args[0], args[1], args[2]} //创建实体转成 JSON
    userJSONasBytes, err := json.Marshal(user)
    if err != nil {
        return shim.Error(err.Error())
    }
    valueAsBytes, err := stub.GetState("YUSER")
    if err != nil {
        return shim.Error("Failed to get YUSER:" + err.Error())
    } else if valueAsBytes == nil {
        return shim.Error("YUSER does not exist")
    }

    valueToTransfer := Value{}
    err = json.Unmarshal(valueAsBytes, &valueToTransfer)
    if err != nil {
        return shim.Error(err.Error())
    }
    intValue, _ := strconv.Atoi(valueToTransfer.Valueid)
    stub.PutState("USER"+strconv.Itoa(intValue), userJSONasBytes)
    newvalueAsBytes, _ := json.Marshal(&Value{"USER" + strconv.Itoa(intValue)})
    valueToTransfer.Valueid = strconv.Itoa(intValue + 1)
    valueJSONasBytes, _ := json.Marshal(valueToTransfer)
    stub.PutState("YUSER", valueJSONasBytes)
    stub.PutState(args[0], newvalueAsBytes) //放入主键 ID，可以根据 ID 查询
    return shim.Success(nil)
}
func (t *SimpleChaincode) readAllOther(stub shim.ChaincodeStubInterface, args []string)
pb.Response {
    if len(args) != 4 {
        return shim.Error("Incorrect number of arguments. Expecting 4")
    }
    stringss := args[1:]
    bargs := make([][]byte, len(stringss))
    for i, arg := range stringss {
        bargs[i] = []byte(arg)
    }
    return stub.InvokeChaincode(args[0], bargs, "")
}
```

query.js

```
var path = require('path');
var fs = require('fs');
var util = require('util');
var hfc = require('fabric-client');
```

```
var Peer = require('fabric-client/lib/Peer.js');
var EventHub = require('fabric-client/lib/EventHub.js');
var config = require('../configsdk.json');
var helper = require('./helper.js');
var logger = helper.getLogger('Query');

var queryChaincode = function(peer, channelName, chaincodeName, args, fcn, username, org, callback) {
    var resData = {
        code: 0,
        data: {}
    }
    var channel = helper.getChannelForOrg(org);
    var client = helper.getClientForOrg(org);
    var target = buildTarget(peer, org);
    return helper.getRegisteredUsers(username, org).then((user) => {
        tx_id = client.newTransactionID();
        var request = {
            chaincodeId: chaincodeName,
            txId: tx_id,
            fcn: fcn,
            args: args
        };
        return channel.queryByChaincode(request, target);
    }, (err) => {
        logger.info('Failed to get submitter \"+username+'\");
        return 'Failed to get submitter \"+username+'\'. Error: ' + err.stack ? err.stack :
            err;
    }).then((query_responses) => {
        console.log("returned from query");
        if (!query_responses.length) {
            console.log("No payloads were returned from query");
        } else {
            console.log("Query result count = ", query_responses.length)
        }
        if (query_responses[0] instanceof Error) {
            console.error("error from query = ", query_responses[0]);
        }
        console.log("Node SDK Response is ", query_responses[0].toString('utf8'));
        var oo = JSON.parse(query_responses[0].toString('utf8')); //字符串转成 JSON
        var srcArr = [];
        if (oo instanceof Array) { //判断对象是否是 Array 的实例
            oo.forEach(function(v, i) {
                srcArr[i] = v.Record;
            });
```

```
                resData.code = 1;
                resData.data = srcArr;
                console.log('Node SDK Data: ', srcArr);
            } else {
                resData.code = 1;
                resData.data = oo;
                console.log('Node SDK Data: ', oo);
            }
            callback(resData);
        }).catch((err) => {
            console.error("Caught Error", err);
            callback(resData);
        });
};
function buildTarget(peer, org) {
    var target = null;
    if (typeof peer !== 'undefined') {
        let targets = helper.newPeers([peer], org);
        if (targets && targets.length > 0) target = targets[0];
    }
    return target;
}
exports.queryChaincode = queryChaincode;
```

trace.js

```
var seed2seedlingModel = require("../models/seed2seedlingModel");
var seedling2plantModel = require("../models/seedling2plantModel");
var plant2inputModel = require("../models/plant2inputModel");
var input2warehouseModel = require("../models/input2warehouseModel");
var warehouse2feedModel = require("../models/warehouse2feedModel");
var feed2productModel = require("../models/feed2productModel");
var config = require('../config');
var fabricQueryModel = require("../models/fabricQueryModel");

let api = {},private = {};
let Trace = {
    api: api,
    _private: private
}
Trace.trace = function(req, res, next) {
    let kind = req.body.kind,
        code = req.body.code;
    console.log(kind + " : " + code);
    private.trace(kind, code, function(data) {
```

```
            res.send(data);
        });
}
private.trace = function(kind, code, cb) {
        if (kind == 'seed') {
            this.forwardTrace(code, cb);
        } else if (kind == 'product') {
            this.backwardTrace(code, cb);
        } else {
            next();
        }
}
// 反向溯源，从产品开始
private.backwardTrace = function(productid, cb) {
        var options = {
            user_id: config.USER_ID_ADMIN_ORG1,
            msp_id: config.MSP_ID_ORG1,
            channel_id: config.CHANNEL_ID,
            chaincode_id: config.CHAINCODE_ONE_ID,
            network_url: config.NETWOEK_URL_ORG1_PEER0, //因为启用了 TLS，所以是
grpcs,如果没有启用 TLS，那么就是 grpc
            privateKeyFolder: config.PRIVATEKEYFOLDER_ADMIN_ORG1,
            signedCert: config.SIGNEDCERT_ADMIN_ORG1,
            tls_cacerts: config.TLS_CACERTS_ORG1_PEER0,
            server_hostname: config.SERVER_HOSTNAME_ORG1_PEER0
        };
        var options2 = {
            user_id: config.USER_ID_ADMIN_ORG2,
            msp_id: config.MSP_ID_ORG2,
            channel_id: config.CHANNEL_ID,
            chaincode_id: config.CHAINCODE_TWO_ID,
            network_url: config.NETWOEK_URL_ORG2_PEER0,
            privateKeyFolder: config.PRIVATEKEYFOLDER_ADMIN_ORG2,
            signedCert: config.SIGNEDCERT_ADMIN_ORG2,
            tls_cacerts: config.TLS_CACERTS_ORG2_PEER0,
            server_hostname: config.SERVER_HOSTNAME_ORG2_PEER0
        };
        var fcn = "readOneById"; //invoke 操作方法
        var fcn2 = "readI2WByWarehouseid"; //invoke 查询入库方法
        var args = ["XFEEDPRODUCT" + productid]; //参数
        var resData = {
            code: 0,
            data: null
        }
        var srcArr = [];
```

```
    fabricQueryModel.findData(options2, fcn, args, function(redata1) {
        args = ["XMATERIALPRODUCT" + redata1.data.productid]; //参数
        fabricQueryModel.findData(options2, fcn, args, function(redata2) {
            args = ["XWAREHOUSEFEED" + redata1.data.feedid]; //参数
            fabricQueryModel.findData(options2, fcn, args, function(redata3) {
                args = [redata3.data.warehouseid]; //参数
                fabricQueryModel.findData(options2, fcn2, args, function(redata4) {
                    redata4.data.forEach(function(v4, i) {
                        args = ["XPLANTINPUT" + v4.inputid];
                        fabricQueryModel.findData(options2, fcn, args, function(redata5) {
                            args = ["XSEEDLINGPLANT" + redata5.data.plantid]; //参数
                            fabricQueryModel.findData(options, fcn, args, function(redata6)
{
                                args = ["XSEEDSEEDLING" + redata6.data.seedlingid];
                                fabricQueryModel.findData(options, fcn, args,
function(redata7) {
                                    var value = new Object();
                                    value.productid = redata1.data.productid;
                                    value.materialid = redata2.data.materialid;
                                    value.feedid = redata3.data.feedid;
                                    value.warehouseid = v4.warehouseid;
                                    value.inputid = redata5.data.inputid;
                                    value.plantid = redata6.data.plantid;
                                    value.seedlingid = redata7.data.seedlingid;
                                    value.seedid = redata7.data.seedid;
                                    srcArr.push(value);
                                    if (srcArr.length == redata4.data.length) {
                                        resData.code = 1;
                                        resData.data = srcArr;
                                        console.log(srcArr);
                                        cb(resData);
                                    }
                                });
                            });
                        })
                    });
                });
            });
        });
    }

module.exports = Trace;
```

致谢

此刻，本科阶段的学习和生活即将结束。在本论文即将完成之际，回顾过去的四年大学生活，感受颇多。

经过了几个月的努力，本论文的得以顺利完成，首先要感谢我的毕业论文指导老师倪福川老师。倪老师在论文开题、研究方法、研究资料、系统搭建和正文撰写等重要阶段都给予了我悉心的指导，提供了许多有益的改善性意见，帮助我在陷入困惑时及时走出困境，给了我极大的帮助和鼓舞。

此外，也要感谢同学及朋友们在论文撰写过程中给予我的帮忙和支持，使我获得了极大的动力和启发。还要感谢参考文献中所有的作者们，阅读他们的研究文章，使我对研究课题有了更好的出发点。以及感谢在四年的本科学习和生活期间传授我知识的所有老师，你们无私的奉献精神和爱岗敬业的治学态度，使我对专业理论知识有了更进一步的理解，将知识和自己的工作互相印证，将会受益匪浅。最后要感谢我的家人们在大学期间给予我的关系和鼓励，在以后的生活中，我也将更加努力的学习和工作，不辜负家人对我的殷殷期望。

转瞬间本科阶段已近结束，我也将步入新的征程。最后，再一次感谢所有在大学生活学习期间，以及本论文撰写过程中给予帮助我的老师和同学。