南京大学

研 究 生 毕 业 论 文

（申请工程硕士学位）

论 文 题 目 基于区块链的软件知识产权认证系统的设计与实现

作 者 姓 名 杨登辉

学科、专业名称 工程硕士（软件工程领域）

研 究 方 向 软件工程

指 导 教 师 陈振宇 教授

2019 年 5 月 27 日

学　　号：**MF1732161**

论文答辩日期：**2019 年 5 月 8 日**

指 导 教 师：　　（签字）

The Design and Implementation of Software Intellectual Property Certification System Based on Blockchain

By

Denghui Yang

Supervised by

Professor **Zhenyu Chen**

A Thesis
Submitted to the Software Institute
and the Graduate School
of Nanjing University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Engineering

Software Institute

May 2019

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：基于区块链的软件知识产权认证系统的设计与实现

工程硕士（软件工程领域） 专业 2017 级硕士生姓名： 杨登辉

指导教师（姓名、职称）： 陈振宇 教授

摘 要

软件复制和传播成本低，使得软件知识产权侵权盗版现象日益严重，影响了软件生态健康发展。国内外大多数提供软件知识产权证明和信息追溯服务的应用系统都依赖于中心机构，受传统信任模型的影响，恶意用户或利益方能够篡改数据。在维权的过程中，第三方机构提供的服务给软件开发方带来了较高时间成本。

本文设计并实现了一个基于区块链技术的软件知识产权认证系统，以期能够保证软件知识产权在互联网上的可信传播。本系统通过区块链技术的去中心化、不可篡改、可追溯、多方维护的特性，保证软件知识产权认证信息能够安全存证和多方共享。本系统基于Vue框架和Spring Boot框架实现前端Web平台和后端业务服务，维护了系统良好扩展性。利用LDAP和Fabric CA实现联盟链节点的用户信息管理和身份认证。采用基于Hyperledger Fabric和IPFS协议的区块链应用程序解决方案，搭建联盟链网络，实现了软件知识产权全网公证和信息不可篡改，同时解决了区块链数据快速膨胀问题。本系统实现了证书工具客户端以生成软件证书，保证软件制品不会泄露。通过TiDB数据库缓存IPFS文件数据，降低数据查询耗时并加快服务响应速度。利用Docker部署系统各个服务，实现Kubernetes 集群管理容器的调度、扩展和负载均衡。

本系统已经部署试运行，结合阿里云和腾讯云部署的两个Fabric共识节点对系统进行了测试。测试结果表明系统能够基于联盟链提供可靠的软件知识产权认证与存证服务。同时在100tps的交易吞吐量下，系统联盟链仍能维持良好的可用性。系统联盟链网络中，每个节点都拥有完整一致的账本，账本数据由所有参与方共同维护，不会被利益群体恶意篡改。最终，用户能够不依赖于第三方机构，在联盟链网络中对软件知识产权进行全网认证并在任意节点准确追溯知识产权信息。

关键词： 知识产权保护，软件产权认证，数字签名，联盟链，星际文件系统，链码

南京大学研究生毕业论文英文摘要首页用纸

THESIS: The Design and Implementation of Software Intellectual Property Certification System Based on Blockchain

SPECIALIZATION: Software Engineering

POSTGRADUATE: Denghui Yang

MENTOR: Professor Zhenyu Chen

Abstract

The low cost of software copying and dissemination makes the software intellectual property infringement and piracy more and more serious, which affects the healthy development of software ecology. Most domestic and foreign application systems that provide software intellectual property certification and information traceability services rely on central institutions. Due to the influence of traditional trust models, malicious users or stakeholders can tamper with data. In the process of safeguarding rights, the services provided by third-party organizations bring high time costs to software developers.

This thesis designs and implements a software intellectual property certification system based on blockchain, in order to ensure the reliable dissemination of software intellectual property on the Internet. With the decentralized, incorruptible, traceable and Multi-party maintenance of the blockchain, the system ensures the software intellectual property certification information can be securely stored and shared by multiple parties. The system implements the front-end Web platform based on the Vue framework and back-end service based on the Spring Boot framework, which maintains the system's good scalability. User information management and identity authentication of the consortium blockchain nodes are implemented by using LDAP and Fabric CA. The blockchain application solution based on Hyperledger Fabric and IPFS protocol is used to build a coalition chain network, which realizes software intellectual property network-wide notarization and information cannot be falsified, and solve the problem of rapid expansion of blockchain data. The system implements the certificate tool client to generate software certificates to ensures software products are not leaked. IPFS file data is cached through TiDB database to reduce data query time and speed up service

response. Various services of the system are deployed through Docker and Kubernetes cluster manages the scheduling, expansion and load balancing of containers.

This system has been deployed for trial operation, and the system has been tested with two Fabric consensus nodes deployed by Alibaba Cloud and Tencent Cloud. The test results show that the system can provide reliable software intellectual property certification and deposit services based on the consortium blockchain. At the same time, under the transaction throughput of 100tps, the system consortium blockchain can still maintain good availability. In the system consortium blockchain network, each node has a complete and consistent account book, and the book data is jointly maintained by all the participants, and will not be maliciously tampered by the interest groups. ultimately, users can authenticate software intellectual property rights in the consortium blockchain network and can accurately trace intellectual property information at any node, without relying on third-party organizations.

Keywords: Intellectual Property Protection, Software Property Certification, Digital Signature, Consortium Blockchain, IPFS, Chaincode

目　录

表　目　录

图　目　录

第一章　绪论

1.1　项目背景与意义

十八届五中全会提出要全面深化知识产权领域改革，加强知识产权保护，加快知识产权交易平台建设。国务院印发《关于新形势下加快知识产权强国建设的若干意见》，对知识产权强国建设做出明确安排。知识产权的“十三五”规划已经明确被纳入国家“十三五”的重点专项规划之中，这也是知识产权规划第一次进入国家的重点专项规划，体现了政府对知识产权保护工作的高度重视和大力支持。

互联网的不断发展，加大了传统知识产权的客体范围，知识产权保护出现了新的领域，即对软件的知识产权保护。软件知识产权是一种数字化资产，以无形的智慧劳动成果为主。随着软件在各行各业的普及，软件的市场需求快速增长，软件知识产权通过交易变现给软件开发方带来了巨大的经济效益。但软件易被复制、传播便捷的特性不可避免的带来了软件知识产权侵权问题 [1]。恶意开发者通过软件源代码“再使用”和“逆向工程”等方式进行盗版侵权，甚至通过造假的方式窃取软件知识产权所有权，造成了软件开发方财产损失，让软件开发者的软件创新积极性备受打击。目前，提供软件知识产权认证和管理服务的应用系统大多依赖于中心机构管理数据。这些系统受限于中心化的信任模式，数据无法多方共同维护，容易被恶意篡改 [2]。由于软件版本变化频繁，这些系统认证效率低，追溯软件产权所有历史信息困难。此外，恶意开发方侵权成本低，相反软件开发方需要第三方机构参与维权，导致成本高、效率低。如何保证软件知识产权在互联网上进行可靠的认证与快速追溯产权信息成为亟需解决的问题。

去中心化、不可篡改、可追溯的区块链技术为上述问题提供了新的解决方案。本文基于区块链技术，协同相关软件管理机构作为验证者节点搭建联盟链网络，建立去中心化软件知识产权认证与管理服务平台，打造全网认证、数据共享的软件知识产权管理新生态。系统通过区块链账本存储数据，保证了软件知识产权认证信息的不可篡改性以及可追溯性。软件开发方可以不依赖第三方机构的服务，从区块链账本中获取软件产权信息进行维权。此外，对于恶意开发方通过伪造产权信息获取利益的行为，侵权记录被全网公开并且不可篡改，这样的方式提高了侵权成本，能在一定程度上遏制侵权现象。

1.2　区块链技术在国内外的发展现状

在国内，国家超前布局前沿阵地，积极探索基于区块链的行业应用。国务院印发《“十三五”国家信息化规划》，将区块链技术作为国家布局重点。各地政府纷纷响应号召成立区块链实验地、研究院 [3]。

国际上，根据Juniper Research去年7月发布的一项研究，全球有超过一半的大公司正在研究区块链技术，目标是将它们整合到产品中，以打破供应链管理，电子商务，知识产权等领域的传统模式。

从中本聪发布的比特币白皮书 [4]中诞生的区块链技术，集成分布式数据库(分类账) [5]、共识机制、点对点传输、加密算法等技术，有效解决了传统交易模式受制于“基于信任的模式”(trust based model) [4]的问题。区块链发展至今分为三个阶段：区块链1.0时代，最典型的应用是加密货币 [6]，其中最突出的就是比特币，其全新的发行方式、交易模式对货币的结构、特征、作用、职能以及社会效应产生了理念的革新 [7]；区块链2.0时代，核心概念是智能合约 [8]，即“一套以数字形式指定的承诺，包括合约参与方可以在上面执行这些承诺的协议” [9]，智能合约的出现为区块链变革社会当前商业模式，构建可编程系统提供了基础 [10]；区块链3.0 时代，分布式应用(DApp) [11] 的出现使得区块链应用从金融领域扩展到供应链管理、电子投票各个领域。通过其构建的信任机制，将改变当前社会商业模式，从而引起新一轮的技术创新和产业变革。

在供应链管理领域，区块链被认为具有巨大的前景，可以解决供应链中物源追踪、成本等问题 [12]。在供应链管理中，制造产品所需的材料和所有流程都需要被管理，直到交付到最终消费者。传统的供应链管理，特别是对于跨国企业，追踪所有的记录几乎是不可能的，缺少透明度就会造成潜在的成本和用户信任问题，甚至会对公司的品牌名气造成负面影响。基于区块链的供应链管理体系，记录的存储和追溯会变得容易。企业可以通过产品标签和内置感应器来获取产品信息，通过区块链来追踪产品从产出地到最终交付地的过程，降低供应链管理成本，加速管理流程，同时保证交易的安全性。

在电子投票领域，区块链的应用可以保障电子投票系统所需的一些重要的特性 [13]。(1) 隐私性。投票系统需要保证投票方的隐私，投票应该是匿名的，以防止被强制投票。(2)公共可验证性。电子投票应提供某种公共可验证性，所有人看到的投票结果是真实一致的，否则，电子投票解决方案的发起者或为利益相互妥协的双方可能能够随意改变投票结果。传统的电子投票都是以发起人或群体为中心的，涉及到多方参与者，并且彼此互相不信任，同时又需要保证公众可验证性。将区块链技术引入电子投票系统，所有参与者权利平等，最终数据结果一致且无法被恶意篡改。

1.3　知识产权领域的国内外发展现状

目前，国内外知识产权保护的系统解决方案普遍采用人工结合中心机构管理的模式。在这种传统模式下，系统容易受到恶意用户的DDoS攻击 [14]和女巫攻击 [15]，造成数据丢失和系统故障。同时因利益相互妥协的多方可能能够篡改数据。此外，知识产权的确权和维权依赖于第三方机构，成本高且耗时长。特别是对于数字化产物的知识产权，相对其传播速度与变更速度，现有模式显得周期长、效率低下。

对于传统知识产权保护存在的痛点，区块链技术带来了新的解决方案。因其去中心化、分布共享、不可篡改、开放等特点，能很好的解决传统保护模式中存在的产权证明造假、所有权追溯难、维权成本高的问题，主要体现在以下几个方面。

存证：当出现产权纠纷时，在法律途径上法官主要是以可信时间戳为主，原件、发布时间等等证据显得不关键。区块链方案的优势就体现在其能制造出一个不可更改的时间戳。知识产权和认证时间生成的独一无二的hash被记录到区块链中。区块链不可篡改的特性保证了上链时间戳的真实性，使其在维权时具有一定的法律效力。

成本：传统知识产权保护需要进行申请登记，登记的成本不菲，还需要复杂的流程和人力成本，可信时间戳为申请产权保护之时。而区块链可以通过制智能合约自动按照既定规则进行验证和存证，进行存证之后，可信时间戳为信息入链的时间，可以作为维权时的法律证据，大大降低了成本。原创者即使还没及时进行产权登记，也可以一定程度上保证其自身权益。

第三方效率：传统知识产权保护依托于可信第三方机构 [16]，知识产权保障力度取决于第三方机构的执行力。维权难的关键原因是第三方机构执行效率低下。而区块链技术通过智能合约按照既定规则自动记录数据,所有节点都可以按规则查询账本数据，移除第三方因素，信息共享在链上所有节点，无法被篡改，极大地提高了效率。

随着区块链技术日渐成熟，国内已经出现了很多结合区块链进行数字产物知识产权保护的项目，比如方维平台提供基于区块链的产权保护解决方案；Rideip乘法网打造基于区块链技术的知识产权存证、确权、保护、托管、投资服务于一体的云平台。在国外，同样有很多企业致力于利用区块链解决数字产物知识产权保护问题。例如BlockAI 启动区块链版权服务项目；Ascribe利用区块链给知识产权进行时间标记创建可持续所有权结构；创业公司Mine提出一个创新型的独特项目Mediachain，推出一个新的元数据协议，允许数字创意者在创作作品上附加信息，然后数据再传输到区块链。

在数字化产物中，软件不同于音乐、视频这类媒体产物，其独有特点表现在具有一定的使用周期，需要迭代维护，而且随着技术的发展，周期会越来越短。对于软件知识产权，传统基于中心机构的系统更加无法满足需求，无法有效地追溯软件历史版本的产权信息。这是目前利用软件旧版本进行“再工程”的盗版侵权现象屡见不鲜的原因之一。基于区块链的解决方案不仅能够对软件知识产权认证信息进行多方共享、安全存证，同时还能以绑定不可变时间戳的方式记录软件知识产权历史版本认证信息。维权时，用户能从任意节点账本获取软件产权信息与产权历史版本变更证据链。

1.4　本文主要研究的工作

本文针对软件知识产权这一特定实体，基于区块链技术实现可信认证，安全存证，多方维护的软件产权认证系统。联合各地区相关软件管理机构共同组成联盟链网络，允许有能力提供软件开发服务的软件开发方获得授权加入联盟链。实现软件知识产权在联盟链网络中全网认证并将认证信息存储到区块链上存证，形成去中心化、分布式、数据安全的软件知识产权账本。同时，考虑到软件具有一定使用周期的特性，本文实现对软件知识产权版本的管理，保证软件产权的历史合法版本链的完整，防止利用软件旧版本进行升级再盗版的侵权行为。系统的主要功能在于软件证书认证上链以及认证信息区块链存储。

软件证书认证功能为软件开发方提供软件知识产权的安全认证。考虑到开发方软件的隐私安全以及软件的多样性，本文设计用软件证书来代替软件进行网络传输。将软件知识产权和软件实体制品在系统内解耦。证书中保存软件制品或软件核心文件库加密后的hash值，同时附带更多的信息包括用户数字签名、时间戳，以关联用户和软件证书，保存从软件信息到具体用户的数据链，便于后续追责。认证过程中，通过用户公钥解密签名对比软件hash，验证用户数字签名，确保认证信息上链前未被篡改。

认证信息区块链存储功能为软件开发方提供知识产权认证信息安全存储。文本将验证过用户数字签名的软件知识产权认证信息存储到联盟链账本。去中心化的联盟链网络节点和作为验证者的相关软件管理机构节点共同保证账本数据的真实性和公信力。智能合约发起附带时间戳的交易提案，由联盟链节点检验并共识，以保证认证数据的不可篡改和可追溯。本文考虑到目前区块链所面临的数据膨胀问题，采用“IPFS+区块链”的存储模式，将认证信息非核心数据存储到IPFS网络，将IPFS网络生成的内容相关hash以及核心数据存储到区块链上。IPFS 文件系统采用内容相关的存储模式，生成文件内容唯一hash标识。区块链保证了文件hash不可篡改，通过该hash访问IPFS获取的文件也不会改变。

本文实现前后端分离的系统，提供软件知识产权认证的平台服务。前台系统采用Vue生态搭建。组件化设计让视图层次分明，提高代码的可读性同时有利于应对用户需求的快速变更。引入路由导航实现局部渲染，加快页面渲染速度，提高系统性能。实现中心化状态管理，将视图与数据状态解耦，保证页面上所有组件的状态一致性，同时记录了所有用户操作引起的状态变更，有利于开发人员对系统进行扩展和维护。后台系统采用微服务架构，分为认证业务服务，区块链服务，用户管理服务。服务使用Docker独立部署。服务内部使用Spring Boot框架实现，采用分层设计，下层向上层提供接口，屏蔽底层实现，有利于代码可维护性。认证业务服务负责对前台系统提供软件认证的相关业务服务。区块链服务对外提供区块链的存取服务，内部实现对Fabric的接口调用，执行chaincode智能合约完成区块链操作。用户管理服务依托于LDAP实现。由节点所属机构自行管理LDAP，Fabric CA只对节点LDAP用户授权允许访问区块链。本文的证书工具实现独立客户端，可供用户离线使用，生成软件证书。证书中包含软件加密的hash值，LDAP用户私钥加密的用户签名，时间戳等信息。用户通过证书与软件知识产权认证系统交互，进行产权认证。

1.5　本文的组织结构

本文详细介绍了基于联盟链的软件知识产权认证系统的设计与实现，其组织结构如下：

第一章绪论。对整个系统的工程背景和现实意义进行描述、介绍了国内外区块链技术现状、区块链在知识产权保护领域的发展概况以及本文主要工作。

第二章相关技术背景。介绍了系统主要使用的技术和开源框架。包括Hyperledger Fabric、Vue生态、Spring Boot框架、IPFS星际文件系统等。

第三章基于区块链的软件产权认证系统的分析与设计。详细分析了系统的需求，包括功能需求和非功能需求。接着以四个系统视图描述了系统架构，依据系统的总体设计按照业务需求划分系统模块，主要业务模块包括证书工具模块、产权认证模块、版本管理模块。最后详细描述系统数据库实体设计。

第四章基于区块链的软件产权认证系统的实现。详细介绍了证书工具模块、产权认证模块、版本管理模块，以及中心化管理模块和区块链服务模块。最后对系统进行单元测试，集成测试和性能测试，并展示系统运行实例。

第五章总结与展望。对本文所做工作、系统解决方案进行了概括。并对基于区块链的软件产权认证系统的进一步工作和方向进行了描述。

第二章　技术综述

2.1　区块链

2009年，神秘学者或团体中本聪发表了文章《比特币：一种点对点的电子现金系统》[4]，开创区块链技术的先河。区块链是一种由多方共同维护，使用密码学保证传输和访问安全，能够实现数据一致存储、难以篡改、防止抵赖的记账技术，也称为分布式账本技术 [3]。区块链是由多种技术融合创新的成果，包括块- 链式结构、P2P点对点网络、共识机制、密码学、智能合约等。

区块链的典型架构，通常包括数据层、网络层、共识层、激励层、合约层[17]。数据层封装了底层区块数据以及包含相关的数据加密和时间戳等技术。网络层主要包括P2P 对等网络机制、数据传播机制和数据验证机制等。共识层协调全网各节点数据记录一致性，包括共识机制即多个个体达成一致的机制[18]和对网络节点的封装。激励层引入经济因素到区块链技术体系中来，主要包括经济激励的发行机制和分配机制 [19]。合约层封装各类脚本和智能合约，是区块链可编程特征的基础。

目前，按照访问区块链数据是否需要作为验证者的节点授权，区块链被分为非许可链和许可链。非许可链以比特币和以太坊 [20]为代表，无需提前获得加入网络的授权许可，任何人都可以加入网络成为节点获取到全部信息，有利于防止对等网络存在的女巫攻击 [15]。许可链依据共识范围分为私有链和联盟链，具有一组作为验证者的中央机构节点对区块链参与者设置权限验证，只有获得许可的参与者才能成为节点，获取区块链网络中的数据。许可链的优势在于扩展性，参与者数量少，能够根据交易数量的增长及时扩展生成区块所需的计算能力。典型的项目就是Linux开源的Hyperledger Fabric [21]。

公有链通过经济形式激励更多节点参与共识，而联盟链或私有链的节点参与进来通常是为了存取链上可信数据 [22]。相对于为了获取经济利益而成为节点而言，联盟链或私有链的节点会更主动和有义务去维护系统的稳定运行。本系统的使用场景是对软件知识产权的认证与认证信息管理。只有获得相关软件管理机构授权的节点才可以加入到区块链中，同时共识范围是在管理机构和其他参与节点之间而不仅仅是管理机构内，因此本系统使用联盟链。

目前比较成熟的联盟链方案包括Hyperledger Fabric和EEA(企业以太坊联盟)。EEA是在Hyperledger Fabric 之后推出的，力求引领基于以太坊的标准

区块链设计，其技术基础是摩根大通开源的Quorum[1]平台，在效率方面要高于Fabric。联盟链项目想要发展成为领域生态圈，扩展性设计是必须要考虑的问题。EEA基于以太坊，理论上节点扩展性是没有限制的，但是缺乏实际项目支撑，而Fabric的节点扩展性可以满足目前大多数落地项目的要求，同时可以支持较多的客户端。

对比两个方案的共识机制，EEA支持Raft [23]和IBFT [24]，Hyperledger Fabric支持PBFT [25]，如表 2.1所示。PBFT 是基于消息传递的共识机制，和IBFT一样都是基于拜占庭容错的共识算法，BFT的变体。Raft强调可用性和最终一致性，效率非常高，但是在安全性方面较差，防欺诈通常只能事后检查。

表 2.1: EEA和Fabric共识机制对比

方案	EEA		Hyperledger Fabric
共识算法	Raft	IBFT	PBFT
效率	非常高	高	高
TPS	400-800		300-500
容错率	50%	33%	33%
分叉	无链分叉	无链分叉	无链分叉
通信复杂度	O(n)	$O(n^2)$	$O(n^2)$

在智能合约方面，Quorum的智能合约方案基本沿袭了以太坊公链思路。Hyperledger Fabric的chaincode设计是一个开创性的智能合约设计框架，同时chaincode基于Docker容器执行，满足区块链和容器之间低耦合设计。

EEA是由三十几家创始成员组成，大多是银行机构。Quorum目前也是被应用在金融领域，在其他领域缺乏有效的案例，无法评估其稳定性。Hyperledger Fabric是由Linux基金会管理的开源项目，成熟且完整，有大量的国际化落地应用支撑，尽管在理论设计上，效率和扩展性都比不上EEA，但也能够满足本系统的应用场景和需求，其稳定性才是最为契合工程应用需求的特性。

结合上述的技术分析以及目前两个方案的发展状况，本系统基于Hyperledger Fabric开源项目搭建联盟链。

2.2　Vue生态

从狭义的角度来看，Vue是一套前端框架，但从广义的角度，它是一套Web应用前端构建通用解决方案，包含自己的技术生态体系。本系统平台是以Vue作为前端的技术体系。

[1]http://www.jpmorgan.com/quorum

2.2.1　Vue框架

Vue是一套构建用户界面的渐进式框架[2]。由尤雨溪在2014年第一次提出并分享在git上，目前，在大量的社区开发者的共同推动下，已经迭代到2.6 版本。

Vue的核心定位并不是一个框架。其设计上也没有完全遵循MVVM模式，只有State和View两部分，Vue的核心功能强调的是从状态到界面的映射，并不重视代码的组织结构。渐进式是Vue 框架设计理念的体现，同时也是其使用方式。渐进式代表的是主张少，即框架对使用者的强制性要求低，框架的扩展性好。对比React、Angular，Vue没有规定必须要用哪些部件、怎么用。可以使用它去开发一个完整的项目，也可以只使用其中一部分，搭配自己设计的下层部件，或者只使用它做整个系统的某一部分。Vue由五个部件组成，分别是声明式渲染、组件系统、客户端路由、大规模状态管理、构建工具。如图 2.1所示。

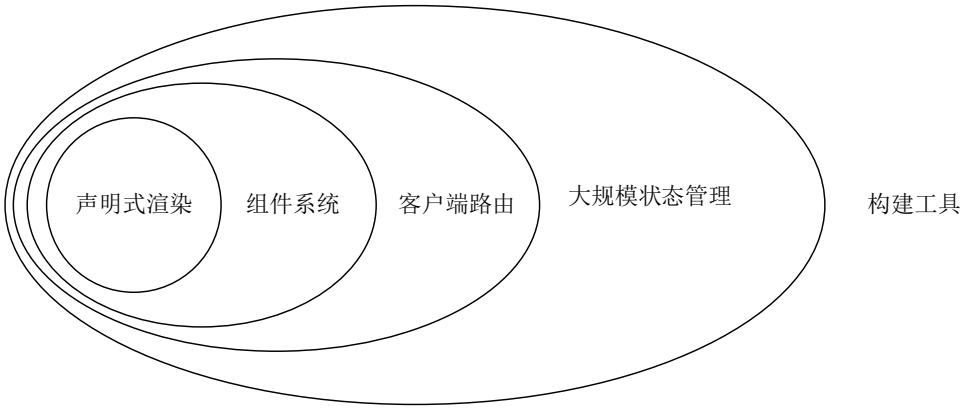


图 2.1: Vue组成部件图

2.2.2　渐进式部件

声明式渲染，Vue最为基础的组件,也是Vue唯一必须要使用的部件。用声明式的方式编程，代码能更加直观，易于维护。其内部对页面渲染的实现采用了“虚拟DOM”算法。将DOM更新操作变成自动化，通过Virtual DOM [26]将状态变更自动地反应到视图变更。在这个过程中，Vue 的响应式系统来侦测在渲染过程中所依赖的数据来源。侦测到数据来源之后，就可以精确感知数据源状态的变动。渲染生成一个新的VirtualDOM树，与旧的VirtualDOM树进行对比，就能以最少的操作描述出界面状态的变动，然后将变动施加到真实DOM树上，以最少的代价更新原生DOM，降低性能损耗，提高界面渲染速度。

[2]https://cn.vuejs.org/

组件系统是为了实现页面组件化，提高系统的可复用性和代码的可维护性。其核心思想是把UI结构映射到恰当的组件树。通过组件间的通讯来相互变更状态达到交互的要求。不同与其他的框架，Vue的组件系统引入了一个单文件组件概念，组件的所有实现部分包含在一个文件里，基于构建工具实现。

客户端路由，官方提供vue-router支持。针对单页面应用而言，建立一个路由表，实现局部渲染对应状态变化，维护一个URL对应到一个应用的状态，降低视图渲染延迟，快速响应用户操作。vue-router的职责就是将应用的URL和组件树的状态之间的映射关系声明式的管理起来。通过局部重新渲染组件来直接渲染出这个URL对应的界面。

大规模的状态管理，又称作中心化状态管理，用于管理平台级应用。当业务变得复杂，应用的变化多而杂，涉及到多人协作开发的时候，系统就会需要在多个组件之间共享数据，多个组件维护同一份数据状态。状态管理使得这样大体量应用依然能够高效运行，保证其可维护性。VueX就是Vue框架提供的中心化状态管理方案。将影响多个组件的全局状态从组件树中提取出来，建立一个中心化仓库Store来管理。当应用状态发生变化时，只允许所有组件通过异步操作产生action显式触发仓库的mutation来改变数据状态，保证所有的数据变化都被记录下来以及同步到所有组件。对于开发者，当出现bug时这些mutation的记录就能有助于更好的理解应用中的状态变化，从而发现问题。

Vue提供基于webpack的构建工具vue-cli，以模块化的方式实现打包和工程构建。同时提供包括开箱即用的热重载、多语言预处理和编译插件、css模块打包scoped特性、异步组件懒加载等优化性能，提高开发效率的功能。

对比目前工业界比较有名的三个前端开源框架，Vue、React、Angular，本系统采用Vue框架实现前台系统。考虑到系统扩展性，为了更好应对区块链技术的变革和应用业务场景的快速膨胀，Vue 框架能够最少地影响整个前台架构，快速应对变更。

2.3　微服务

互联网时代的到来，各大互联网平台都呈现出用户群庞大，网站流量巨大，并发量要求高的现象。并且用户需求变化频繁，平台需要针对一系列性能问题进行优化。传统的架构已经不能很好满足当前繁杂的需求。近几年，微服务架构 [27]被提出。它描述了一种将软件应用设计成可独立部署提供服务的部件的特定方式。微服务架构将系统应用拆分成一个个小型服务进行开发。每个服务内部可以使用不同的技术栈进行应用程序开发 [28]，服务之间通过不同的通信协议提供对外接口。最大程度地做到服务间高内聚、松耦合。

实现微服务架构通常需要一个集中式服务注册中心，以及某种服务发现机制。同时还需要内嵌负载均衡机制。微服务框架中的每个服务通常会被部署为一个独立的进程。常用的微服务架构方案有ZeroC IceGrid [29]、Spring Cloud [30]与Docker Swarm [31]。

ZeroC IceGrid服务注册中心就是Ice Registry，服务发现机制称为命名查询服务，可以根据服务名查询对应的服务地址。IceGrid通过客户端API内嵌的算法实现负载均衡。

Spring Boot就是常用的一种微服务实现框架，而Spring Cloud是基于Spring Boot实现微服务架构的框架，一个更好的整体方案。它与其他实现方式最明显的区别在于Spring Boot是采用Java语言，所以Spring Cloud只能采用Java语言。Eureka模块是Spring Cloud的服务注册中心，它与Zuul服务网关、Ribbon组件共同协作实现负载均衡。

Docker Swarm是Docker公司发布类似于Kubernetes [32]的基于容器技术的微服务平台。Swarm Manager负责服务集群的管理、维持集群状态以及任务调度。Swarm manager给每个服务都分配了唯一的DNS标识名，可以使用最简单的轮询机制来实现服务发现和负载均衡。

微服务架构实现了部署和更新服务的简易性，简化运维人员复杂的手工配置和处理工作。动态地发现和绑定服务提供系统模块之间的低耦合设计，提高系统可维护性。本系统采用的是基于Spring Boot结合Kubernetes实现的微服务架构方案。参考Spring Cloud组成架构，基于Spring Boot实现服务内部的对外接口，通过Docker容器部署运行。采用Kubernetes集群代替Spring Cloud集成的服务注册中心，实现对应用实例的注册、管理、发现、访问。

2.4 Spring Boot

Spring Boot是由Pivotal团队提供的基于java的服务端开发框架，其设计目的就是用来简化目前使用的基于Spring应用的初始化搭建以及开发过程 [33]。相对Spring 框架，Spring Boot 省去了繁琐的XML 配置，使用特定的注解方式来进行自动配置 [34]。通过starter POM来简化Maven配置项，让Maven配置变得简单。自动化依赖包管理，解决依赖包版本冲突问题。Spring Boot 还集成了嵌入式的Web服务器，无需部署WAR文件，只需要通过命令行指令就可以启动应用程序。Spring Boot化繁为简让配置自动化、开发变得简单、代码易维护，同时提供对主流的构建工具、日志框架、缓存框架良好的支持，在工业界的使用趋势日渐超越Spring。

本系统使用Spring Boot作为后端框架。Spring Boot作为目前最常用的Java后端框架。其提供良好的REST风格接口设计，简化了配置，自带servlet容器可以独立运行，自动管理依赖包，支持各种主流工具，对开发人员屏蔽底层实现。这些优化设计使得Sprint Boot框架能够开箱即用，减少开发人员在项目配置和管理上的时间成本，提高开发效率。

2.5　数据加密

数据加密技术根据密钥类型的不同，分为对称加密算法和非对称加密算法。本小节主要介绍AES、RSA两种加密算法。

AES是典型的对称加密算法，比DES安全性更高，速度更快。AES由美国国家标准技术研究所在1997年提出，用以取代DES。AES的加密过程由四种不同的变换组成，包括三个替代和一个置换 [35]。字节代替是指用一个S盒完成从字节到字节的映射。行移位是一个简单的置换。类混淆是利用域GF上算术特性的一个代替。轮密钥加是将当前分组和扩展密钥的一部分进行按位异或。解密过程是和加密过程相反的运算过程。

RSA加密算法是公开密钥系统的代表，典型的非对称加密算法，1977年由麻省理工学校工作的三人一起提出。被广泛使用在电子商业。RSA的安全性是建立在,具有大素数因子的合数其因子分解困难这一规则上。其处理速度较慢，不适合大量数据文件加密。对于密钥管理，RSA要优于AES算法，加密过程不必网络传输需要保密的密钥。RSA加密只需要交换公钥。公钥负责加密，私钥负责解密。私钥负责签名，公钥负责验证。

2.6　IPFS

IPFS [36]全称Inter-Planetary File System，中文名称星际文件系统，是一个面向全球分布的、点对点分布式文件系统。2014年5月，该项目由Juan Benet发起，几个月后，IPFS协议实验室成立，并发布了IPFS。2016年IPFS团队创建了libp2p、IPLD、multiformats、Orbit等模块。2017年，IPFS开始规模性的使用，已存储了50亿份文件，也开始有更多的代码贡献者参与到社区。

IPFS出现之前，有一些其他分布式存储网络的项目，比如BitTorrent、Kazaa、Napster等。其根本的目的都是为了能搭建出一个比现在普遍使用的HTTP协议更好的资源网络，以弥补HTTP 存在的缺陷。IPFS就是基于这样一个背景下诞生，与HTTP相比，IPFS表现出了下载速度快、全球存储、安全、数据永存等优势。

IPFS本质上是一种内容可寻址、版本化、点对点超媒体的分布式存储、传输协议。具有以下几个特征：

内容可寻址：IPFS只关心文件内容，通过文件内容生成唯一的哈希标识，内容寻址会通过唯一的标识去访问 [37]，并且提前检验这个标识是否已经存储过。如果被存储过，直接从其它节点读取它，不需要重复存储，一定意义上节约了空间。

大文件切片：放到IPFS节点中的文件，不关心文件的存储路径或者名字。IPFS提供将大文件进行切片存储的功能，使用时并行下载多个切片。

去中心化，分布式网络结构：这样的网络结构适用于解决区块链存储能力的瓶颈，将大量的超媒体数据存在IPFS上。

加密存储：IPFS向加密数据中加入数字信息特有的加密散列，存储的文件对应的hash就无法变更，hash与文件一一对应。

本系统通过IPFS去中心化的P2P分布式文件存储网络，将大量超媒体文件存储在区块链外 [38]，根据文件内容生成唯一加密哈希值。区块链不再需要存储相关的业务数据库中的数据，转而存储业务数据在IPFS中对应的内容相关hash值，以此来解决区块链的存储能力问题。纽约创业公司Mine Labs的Mediachain 项目就是基于Hyperledger Fabric和IPFS协议的区块链解决方案 [39]来保护知识产权。

2.7　TiDB

TiDB是PingCAP公司设计的开源分布式HTAP数据库[3]。设计上结合了RDBMS和NoSQL 的特性，既能像非关系型数据库一样存储大量的数据，又能像传统关系型数据库一样提供便捷的ACID事务。PingCAP 设计TiDB 的目标是为联机事务处理（OLTP）和联机分析处理（OLAP）场景提供一站式解决方案。

TiDB具有高度兼容性，MySQL可以轻松无缝迁移至TiDB。同时通过增加新节点即可实现TiDB的水平弹性扩展，解决高并发、海量数据场景。相比于传统的主从服务器数据复制模式，TiDB 采用基于Raft的多数派选举协议，提供数据强一致性保证，且在大部分副本没有丢失的情况下，可以自动实现故障的恢复。此外，TiDB是云原生数据库，支持公有云、私有云、混合云模式。

本系统是通过XML文件的形式存储链外数据到IPFS网络中。考虑到从IPFS读取文件并解析成数据对象的效率问题，系统采用TiDB作为缓存数据库，对用户所需的业务数据进行存储，方便用户再次查询。

[3]https://pingcap.com/docs/

2.8 LDAP

LDAP是基于X.500标准的轻量级目录访问协议 [40]，是目前使用最广泛的目录访问协议。LDAP通常泛指由目录数据库和访问协议组成的系统，本质上是一个非关系型数据库。

LDAP目录数据库是一个优化查询和搜索的数据库，提供了良好的读性能，以树状层次结构组织数据。目录信息集合就是一个目录信息树(DIT) [41]。树中的基本数据单元被称为条目，每个条目就是一条记录，由属性组成。每个条目有一个唯一标识DN，用户通过DN 定位条目在树中的位置，访问其属性数据。

LDAP协议是开放的标准协议，支持跨平台的Internet协议。在市场上得到各大厂商的广泛认可，已经成为大多数软件产品支持的用于统一身份认证的标准协议。它易于在任何应用系统中通过简单的LDAP客户端程序访问到LDAP目录，常被公司或机构内部用于实现用户身份管理。

本系统是采用联盟链方案而不是公有链，节点参与方是管理机构或者软件开发企业。这些机构或公司内部已经使用LDAP实现了统一身份管理，为方便部署区块链节点，同时保护各节点的用户隐私，本文使用LDAP存储用户信息，并结合Fabric CA完成用户在系统中的区块链访问权限认证。

2.9 本章小结

本章主要描述了本系统平台使用到的主要技术和工具，通过使用这些工业界成熟的框架和开源技术，包括Fabric、IPFS、Vue、Spring Boot、LDAP等，构建具有良好扩展性并能支持需求快速扩展的系统。包括前端交互平台、证书工具、基于微服务架构设计的后端服务和依赖Fabric开源项目实现的底层区块链服务。最终实现区块链技术在软件知识产权领域的实践应用。

第三章　基于区块链的软件知识产权认证系统的需求分析与设计

3.1　项目总体规划

软件这一数字产物具有两种特点。第一，软件不是物理实体而是逻辑实体，这也使得软件知识产权不同于传统实体知识产权。第二，软件具有一定使用周期，随着软件行业发展，这个周期会被不断缩短，软件需要不断的更新维护。这使得软件知识产权不同于音乐这类数字媒体知识产权，需要不断地迭代更新并且通常情况产权交易双方只会交易软件知识产权使用权，仍需要软件开发方来提供后续维护。本系统是基于区块链技术在软件知识产权领域的一次应用，针对软件这一特殊实体，实现快速确定产权所属，维护不可篡改产权归属数据，高效提供维权证据链。

本系统主要实现软件开发方在系统上进行软件知识产权认证与版本管理。软件开发方成为联盟链网络授权节点，以LDAP中的用户身份登录系统进行软件知识产权认证和账本数据访问。上传由系统证书工具生成的软件证书，系统负责验证证书信息和用户数字签名，将软件证书以及相关业务数据存储到星际文件系统IPFS中，并将生成的存储路径hash和核心认证信息记录到联盟链上。

系统主要由证书工具、前端交互平台、业务处理服务、区块链服务四个部分组成。出于对软件开发方的软件隐私保护以及软件类型多样性考虑，系统将软件知识产权概念与软件制品分离，以能标识软件并包含更多产权信息的证书代替软件制品，作为软件知识产权载体在系统内交互。证书工具将软件加密hash、用户数字签名、以及时间戳生成统一XML格式证书文件。软件开发方使用该证书在系统内进行知识产权认证。前端交互平台根据用户所属节点在Fabric中的权限展示软件知识产权认证相关操作入口，用户进入交互页面进行相关操作，认证软件知识产权和管理产权版本。业务处理服务负责对软件开发者上传的证书进行内容校验，验证数字签名一致性，完成软件知识产权认证以及对软件产权版本管理等业务逻辑的处理。区块链服务负责存储软件产权认证信息到区块链账本中进行全网公证。证书文件以及软件知识产权认证信息被存储到IPFS中，IPFS通过存储的内容生成相应的地址hash，将hash和用户信息通过智能合约存储到区块链。

3.2　系统需求分析

在对项目进行总体的规划和确定主要业务后，对系统进行需求分析，得到系统相应的功能性需求和非功能性需求。

3.2.1　系统涉众

本系统的用户涉众为软件开发方，软件需求方如表 3.1所示。软件开发方是有能力进行软件开发的公司或个人，通过授权节点获取服务或者加入联盟链获取账本数据。软件开发方使用软件证书生成工具子系统，将开发的软件制品生成唯一对应的证书，证书中附带开发方用户数字签名。开发方可以使用证书在系统内进行软件知识产权认证，认证信息校验后会记录到联盟链节点账本中。对于软件具有一定使用周期的特性，系统提供软件知识产权版本管理服务。软件开发方可以更新软件知识产权版本，查询软件产权随时间变更过程。软件需求方是有相关业务领域软件开发需求但是没有研发能力的机构或公司。需求方使用系统证书生成工具对开发方交付的软件进行验证，验证交付软件制品是否与软件证书对应。

表 3.1: 用户角色说明

用户角色	用户特征
软件开发方	有能力进行软件开发的公司或个人。需要通过系统进行软件知识产权的认证。
软件需求方	有软件开发需求但是没有研发能力的机构或公司。需要通过系统对比交付的软件制品和证书进行一致性验证。

3.2.2　功能性需求分析

本系统分为软件知识产权认证服务平台和证书工具。证书的生成与验收服务抽离成一个客户端子系统，软件开发方可以通过证书工具生成软件证书，通过证书代替软件进行后续的一致性验证和用户数字签名验证。同时软件开发方和软件需求方都可以通过该工具验证软件是否被篡改。软件开发方通过LDAP登录系统平台，生成Fabric CA授权证书。开发方上传软件证书实现对软件知识产权的认证。考虑到软件具有一定使用周期的特性，会不断更新迭代，系统支持对软件产权进行更新，按照时间顺序对软件产权的各个版本进行管理，方便软件开发方随时能查看、下载所属软件各版本证书。认证信息上链后，用户能通过上链时间、hash、channel名等条件查询到所属区块信息，上链交易提案信息，以及节点信息和账本信息。

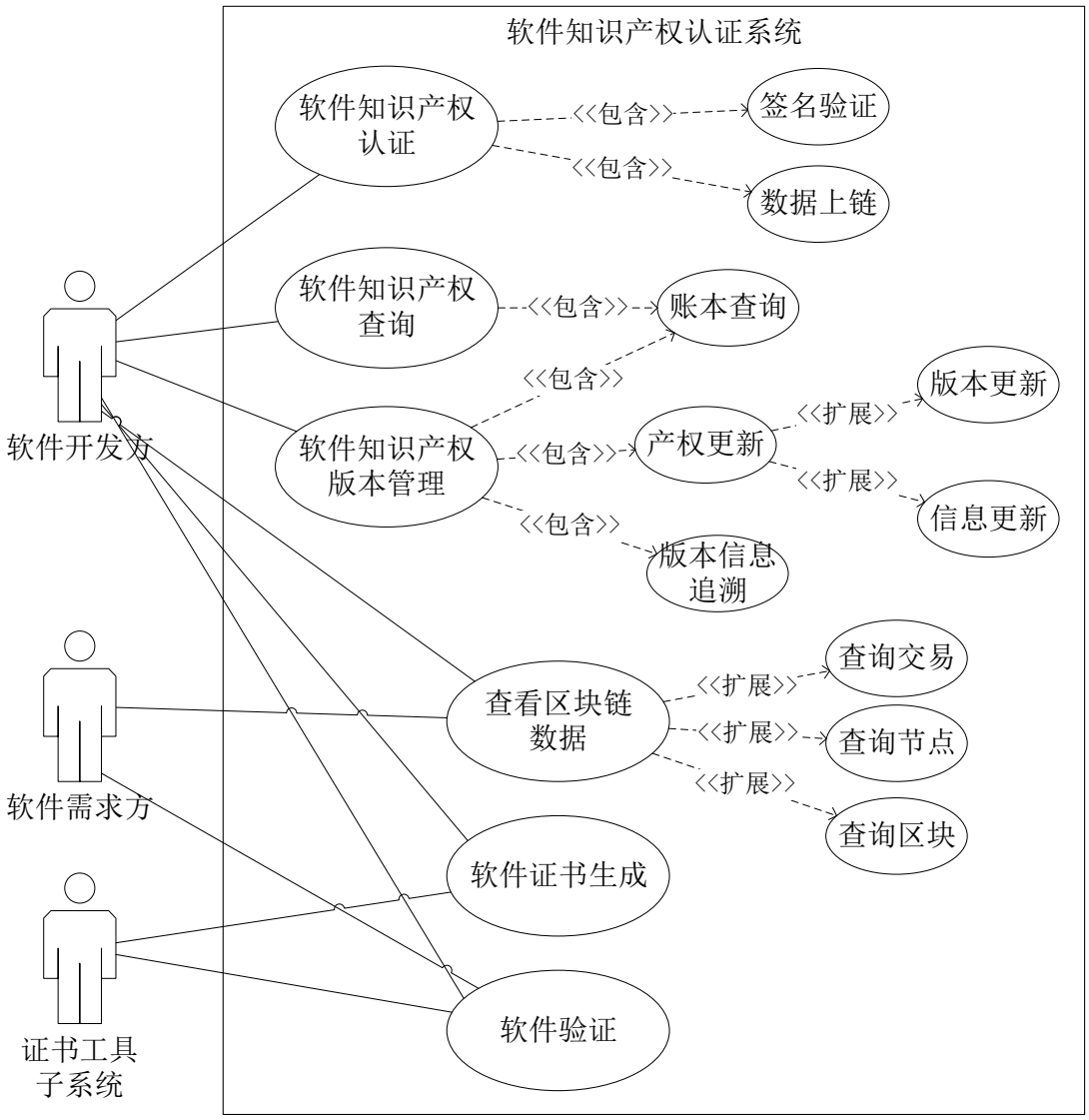


图 3.1: 系统用例图

根据对系统涉众的特性分析，得出系统相关用例，如图 3.1所示。系统用例主要包括软件知识产权认证，软件知识产权查询，软件知识产权版本管理，区块链信息查询，软件证书生成，软件证书与软件制品一致性验证。下文将对系统主要功能需求进行详细用例描述。

表 3.2描述了软件开发方使用证书工具生成软件证书的用例。生成证书是软件知识产权认证的前提。在使用证书工具进行软件证书生成之前，软件开发方必须使用LDAP注册用户获取LDAP 的用户私钥。证书生成工具需要提供可交互的界面,获取用户输入相关参数包括用户名，可选加密算法，私钥文件，证书文件路径，证书输出路径。系统依据用户选择的加密算法，实现软件文件加密

并读取用户私钥，使用用户私钥对软件加密结果进行RSA加密生成数字签名。对于证书所需的用户私钥，用户可以选择从LDAP上获取并输入私钥文件路径或通过证书工具登录LDAP自动获取。系统默认是需要用户提供私钥文件路径。软件开发方可以配置LDAP输入用户名和密码，LDAP验证登录授权后，系统可以通过LDAP获取对应用户私钥。

表 3.2: 软件证书生成用例描述

ID	UC1
用例名称	软件证书生成。
用例描述	软件开发方利用证书生成工具对软件文件进行哈希散列，同时使用者提供LDAP个人用户私钥加密软件哈希值，对证书进行数字签名。证书生成工具结合软件hash，用户签名和时间戳生成XML 格式证书文件。
参与者	软件开发方
优先级	高
前置条件	软件开发方下载了证书生成工具。软件开发方使用LDAP并注册了用户。
后置条件	无
基本操作流程	1.软件开发方输入使用者用户名。 2.选择要使用的加密算法。 3.输入LDAP私钥文件路径。 4.输入需要认证的软件的路径。 5.选择生产证书的输出路径。 6.点击生成。
可选流程	如果软件开发方没有下载LDAP私钥文件。 3.a.1开发方可以在证书生成工具进行LDAP验证。跳转到LDAP授权交互界面。 3.a.2输入用户名和密码。 3.a.3点击验证。工具获取LDAP用户私钥。

表 3.3描述了软件开发方进行软件产权认证的用例。软件开发人员进行软件知识产权认证是系统的主要业务功能,核心在于把软件证书hash与认证信息记录到区块链账本中。用户使用软件知识产权认证服务的前提是用户在联盟链节点登录，并且被Fabric CA授予访问区块链的权限。软件开发方发起认证请求需要输入软件名，描述和上传软件证书，软件制品可以根据用户实际情况选择是否上传。区块链记录认证信息和用户标识。根据软件具有一定使用周期后需要迭代更新的特性，系统提供软件知识产权版本更新服务。用户选择所需更新的软件知识产权，并上传新的证书与软件制品。考虑到区块链不提供信息更新操作，链上记录不可篡改，系统通过存储软件新的版本产权信息及版本更新记录来完成软件知识产权版本更新服务。软件知识产权认证信息是否能成功记录到区块链账本中，需要等待“背书”节点验证交易提案规则，并向共识节点提交交易，构造区块。最终Committer节点对区块校验后，将认证信息写入区块链。

表 3.3: 软件知识产权认证用例描述

ID	UC2
用例名称	软件知识产权认证。
用例描述	软件开发方对自己的软件知识产权进行所有权认证，软件证书信息以及认证信息存入区块链，保证所有权认证信息的不可篡改。
参与者	软件开发方
优先级	高
前置条件	用户从联盟链节点登录，经过Fabric CA验证并生成证书，允许用户访问节点账本
后置条件	共识节点成功构造区块，Committer节点获取交易区块做最终检查并写入账本。
基本操作流程	1.选择认证新软件。 2.输入软件名及描述。 3.上传软件证书。 4.上传软件制品。 5.点击提交。
可选流程	如果是对已有软件知识产权版本的更新。 1.a.1选择更新产权版本。 1.a.2选择需要更新的已有软件产权 如果软件开发方不上传软件制品。 4.a.1转入正常流程5。

表 3.4描述了软件知识产权查询用例。软件开发方认证软件知识产权后，可以访问区块链账本查询已有的软件知识产权的所有信息。用户登录系统后，Fabric CA生成用户证书,授权用户访问区块链数据的权限。用户通过系统前端平台相关入口查看已有的软件知识产权认证信息，下载软件知识产权证书和软件制品。软件开发方可以填写查询表单，按照表单条件查询符合条件的特定软件知识产权认证信息。数据查询只需要有节点账本访问权限，“背书”节点不需要将请求提案提交到共识节点。查询请求通过背书节点验证，执行智能合约。

表 3.4: 软件知识产权查询用例描述

ID	UC3
用例名称	软件知识产权查询。
用例描述	根据特定条件查询用户已有软件知识产权。
参与者	软件开发方
优先级	高
前置条件	用户从联盟链节点登录，经过Fabric CA验证并生成证书，允许用户访问节点账本
后置条件	背书节点校验通过。
基本操作流程	1.查询用户已有的所有软件知识产权。 2.填写查询表单。 3.查询特定的软件知识产权。

表 3.5: 软件知识产权版本追溯用例描述

ID	UC4
用例名称	软件知识产权版本查询。
用例描述	软件经过一定周期后需要迭代更新，软件产权版本也随之更新
参与者	软件开发方
优先级	高
前置条件	软件产权有历史版本
后置条件	背书节点校验通过。
基本操作流程	1.选择已有的软件知识产权。 2.查询该软件知识产权的所有版本。 3.查看某一个版本的信息。
可选流程	无

表 3.5描述了查看已有软件的历史版本的用例。由于软件这一逻辑实体自身具有一定使用周期的特性，软件知识产权不同于其他数字化产权。系统需要更新软件产权版本并保留软件知识产权随时间变化的追溯链。用户选择需要查看历史版本的软件知识产权，系统按照时间顺序展示出软件知识产权的迭代变更和版本之间的层次关系。用户可以查看其任意版本的详细信息。

表 3.6: 区块链账本信息查询用例描述

ID	UC5
用例名称	区块链账本信息查询
用例描述	用户通过搜索条件查询节点账本信息，包括区块信息、交易信息等
参与者	软件开发方，软件需求方
优先级	高
前置条件	用户登陆并有访问区块链账本的权限。节点部署了Hyperledger Explorer
后置条件	无
基本操作流程	1.选择查询时间段 2.输入交易id 3.点击查询 4.查看详细信息
可选流程	如果要查询满足条件的一系列信息 2.a.1输入相关查询条件

表 3.6描述了区块链账本信息查询用例，用户获得授权允许访问区块链后，可以根据产权认证信息中展示的交易id查询节点账本的交易详细信息，包括交易提案包含的数据，提交的channel，执行的智能合约，所属区块信息，背书节点等。保证了整个认证信息上链过程的可验证性与数据的一致性。

表 3.7: 软件证书验证用例描述

ID	UC6
用例名称	软件证书验证用例
用例描述	系统是以证书作为知识产权认证媒介，通过证书可以校验软件是否被篡改或替换。
参与者	软件开发方，软件需求方
优先级	中
前置条件	有参照的证书与要验证的软件制品。下载了证书工具
后置条件	无
基本操作流程	1.从系统获取要验证的软件的证书。 2.填写证书的文件路径。 3.输入软件的文件路径。 4.使用证书生成工具验证。
可选流程	无

表 3.7描述了软件验证用例。软件产权认证系统需要提供软件制品与证书之间的一致性验证功能。通过对目标软件进行加密，对比得到的hash值与证书里提供的软件加密hash值是否一致，从而验证软件制品是否被篡改。用户使用软件验证服务需要从系统下载需要的软件证书。用户在软件证书工具验证界面填写参考证书文件路径与需要对比的软件文件路径。证书工具读取证书信息，对比软件加密hash与证书中的hash，判断软件是否被篡改显示验证结果。

3.2.3　非功能性需求分析

对于系统非功能性需求方面，考虑到系统是面向用户的软件知识产权提供服务，为了让用户能更安全的使用系统同时维持系统的可用性，系统需要对安全性、扩展性、区块链存储性能有一定要求。

安全性：软件产权认证系统涉及到软件开发方的核心资产。整个认证过程包括后续的存储都要保证数据安全性，特别是信息存储到区块链账本之前，需要保证在产权认证过程中，产权信息的完整性和真实性。

区块链数据膨胀：区块链数据快速膨胀是当前区块链技术面临的严峻问题。在应用区块链技术到工程领域时，必须要考虑到区块链数据存储成本，业务数据过多会导致每个节点的账本存储成本过大。系统应尽可能少的只将核心数据存储到区块链，减少区块链的数据膨胀速度。

业务可扩展性：系统业务场景会不断扩展，用户的需求会快速变化。系统在设计与实现上需要有一定程度的解耦与分层设计，以保证系统良好的扩展性和代码的可维护性。

3.3　系统总体设计

从需求分析中我们得到系统功能需求和性能需求，本节将结合需求进行系统的总体设计。通过系统总体设计，能够把握系统整体架构和结构层次，划分出各个子系统和主要的功能模块以及其相应的职责。将上一节描述的需求模型转化为逻辑模型。为后续的模块详细设计和数据模型设计提供设计基础。本节将先描系统述架构设计，然后从逻辑视图，进程视图，开发视图，物理视图四个角度来描述系统的总体概要设计 [42]，划分并总结出主要的业务模块。

3.3.1　系统架构设计

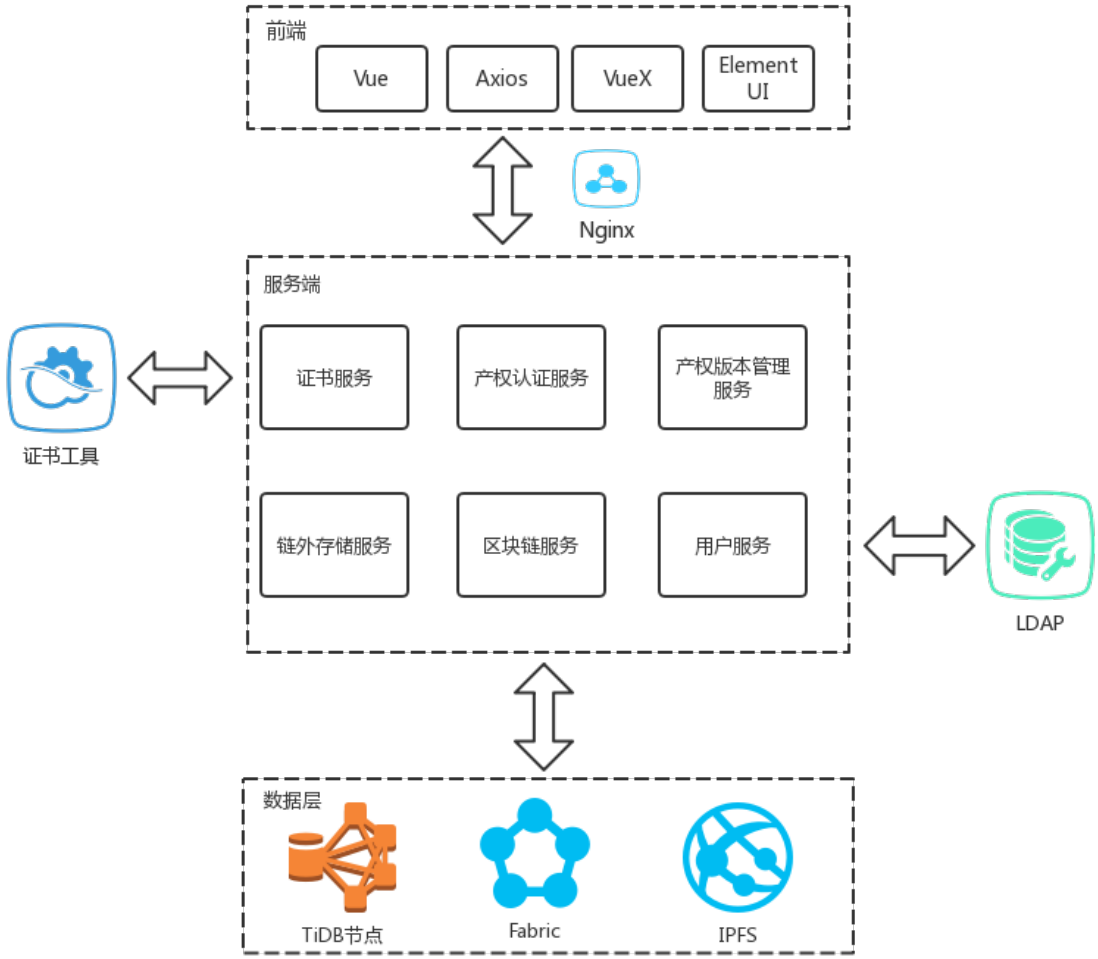


图 3.2: 系统架构图

本系统是基于区块链的分布式应用，图 3.2描述了系统的整体架构。系统整体采用了P2P的网络拓扑结构，每个系统节点由前端、服务端、数据层三部分构成。前端采用Vue框架、Axios异步编程框架、Vuex中心化状态管理插

件、ElementUI开源组件，实现系统的视图展示和用户交互。服务端采用Spring Boot实现业务服务，包括证书服务、产权认证服务、产权版本管理服务、链外文件存储服务、区块链服务、用户管理服务。其中，证书服务由证书工具客户端提供，支持离线服务。用户管理服务基于LDAP协议实现，由节点所属机构内部管理节点用户。数据层包括IPFS节点，Fabric节点，TiDB节点。IPFS节点用来管理链外数据，数据以文件的形式存储到IPFS网络。Fabric节点用来管理链上数据，由区块链账本、Fabric CA和chaincode链码组成。区块链账本用来保存所有的链上交易数据，所有节点账本数据相同。Fabric CA结合LDAP完成用户身份认证和区块链访问权限管理。chaincode是Fabric独特的智能合约实现方式，自动执行数据上链更新账本和获取账本授权数据。TiDB节点作为系统的缓存数据库，缓存IFPS中被访问的文件数据，提高系统数据查询响应速度。

3.3.2　系统逻辑视图

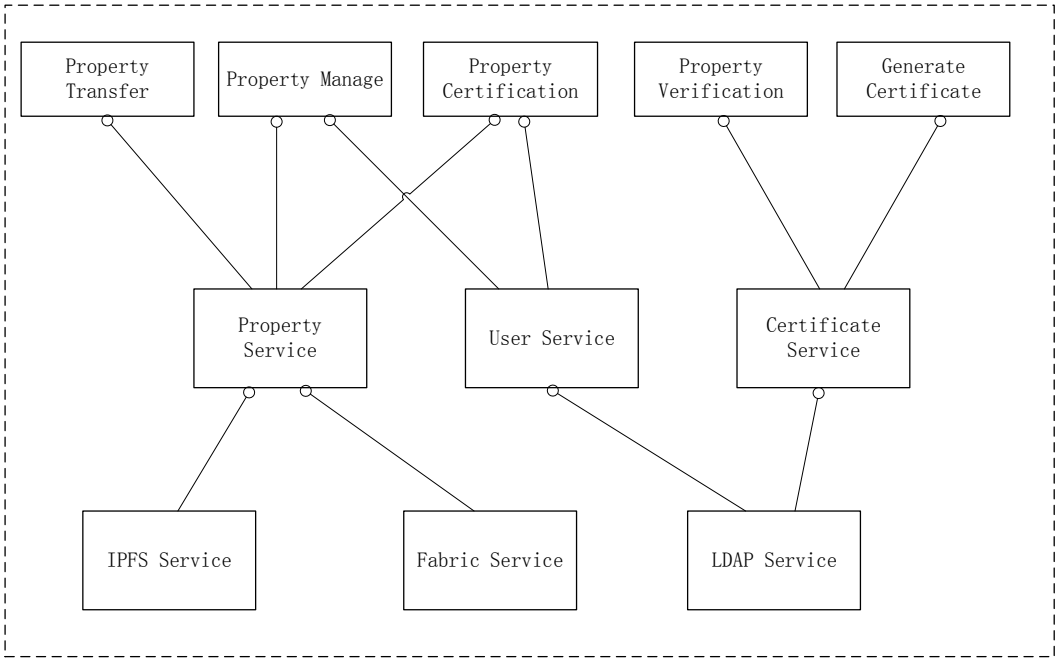


图 3.3: 系统逻辑视图

如图 3.3所示，系统逻辑视图从用户的角度描述了系统逻辑结构设计。以系统功能需求用例为出发点，用户会使用本系统进行软件证书生成，软件知识产权认证，软件产权版本管理，软件一致性验证和软件产权转移等业务。

图中PropertyService是对软件产权相关服务的统称，是系统业务的核心服务模块，提供包括软件知识产权认证、软件产权更新与版本管理、软件产

权查询等服务。依赖于底层的IPFS 星际文件存储网络提供数据文件化存储和Fabric超级账本服务提供区块链操作接口。UserSevice负责对系统部署的该区块链节点的用户进行信息维护和产权数字签名验证，提供用户相关的信息支持PropertyService。依赖于LDAP提供的用户信息获取接口和权限管理服务。CertificateService负责将用户的软件制品进过加密转换为hash值，与用户的私钥数字签名和时间戳，共同生成XML格式软件证书。同时也实现对证书和软件制品的一致性对比验证功能。依赖于LDAP提供用户私钥。IPFSService实现软件认证业务数据存储到IPFS分布式存储网络，返回内容hash，并能根据hash随时读取到原始数据。FabricService实现通过FabricClient调用chaincode智能合约发起交易提案实现区块链数据上链与查询等一系列区块链操作，提供软件认证关键信息数据上传区块链和从区块链查询的服务。LDAPService提供节点用户信息的查询、管理以及公钥私钥查询。

3.3.3　系统开发视图

通过从用户的角度对系统逻辑结构进行设计，明确了系统的逻辑结构。接下来从开发者的角度对系统的工程开发结构进行设计。如图 3.4系统总体的结构遵循典型的分层架构设计，实现前后端分离。有利于系统的代码维护。

系统总体分为三层，展示层、业务层、数据层。展示层以用户业务逻辑为起点设计交互界面和组件，包括软件产权认证页面、软件产权管理页面、软件证书生成页面等。其中软件证书的生成和一致性验证是基于子系统证书生成工具来进行交互。平台所有的页面实现视图与状态分离，数据变动都通过Dispatches操作分发模块来统一通知Store状态管理模块。Store模块实现对数据状态的中心化管理以及与后端交互的统一接口。记录了所有的用户异步操作以及数据变动的过程，降低了前端系统的耦合度。证书生成工具的交互页面是独立的客户端界面，用以给用户填写个人私钥以及软件路径等信息。

业务层拆分为两个子系统，软件证书在线认证子系统负责软件产权的在线认证与管理，证书生成工具实现软件真实制品和软件证书之间的对应关系。软件证书在线认证子系统处理web 平台的软件产权相关业务请求，controller模块监听前端请求，发起线程调用软件产权服务和用户服务，从数据层请求相关数据，完成业务逻辑，返回相应数据给前端。证书生成工具通过客户端界面获取用户输入的信息，在用户提供私钥文件路径的情况下，不需要发起http请求，可以离线对软件进行加密，生成证书文件，包括MD5加密部分、XML文件生成部分。同时也可以离线验证软件和证书的一致性。

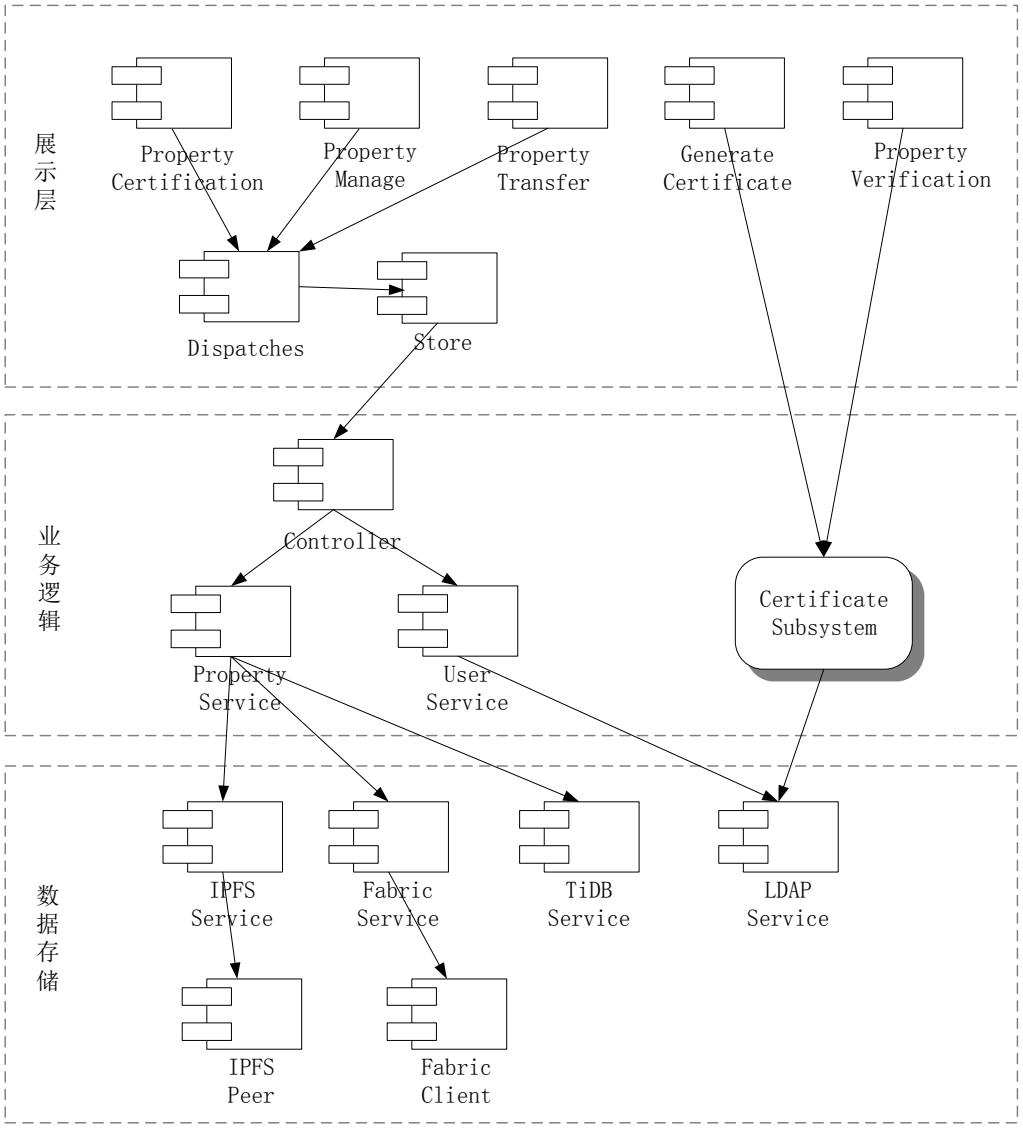


图 3.4: 系统开发视图

数据层以区块链存储为核心，通过FabricService调用FabricClient提交交易提案，对上层提供区块链服务接口。IPFSService封装IPFS网络对外原生接口提供文件存储服务，通过IPFS 来存储大量的业务数据，解决区块链数据膨胀问题同时保证数据的不可篡改。TiDB分布式数据库作为缓存数据库，存储用户查询过的IPFS数据，避免重复解析IPFS文件，优化用户体验。LDAP用来管理授权用户信息，独立部署由节点机构管理。LDAP授权用户才能在Fabric CA上获得证书，有访问区块链的权限。

3.3.4　系统进程视图

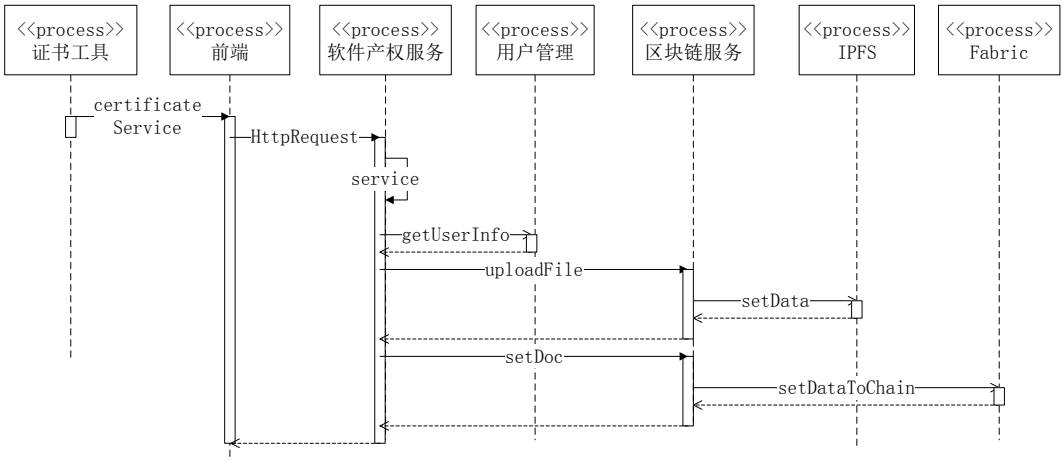


图 3.5: 系统进程视图

通过从用户和开发者的角度对系统结构进行设计，明确了系统划分的独立服务。接下来从系统各服务进程间交互的角度介绍系统服务间的调用关系。图3.5为软件知识产权认证系统的进程视图，图中展示的每个部分都作为一个独立进程运行。证书工具进程与其他进程相互独立，它是运行在用户设备上的客户端工具，可以离线提供证书生成与验证服务。系统典型的认证流程从前端进程开始，前端是运行在浏览器端的web进程，响应用户交互，发起HTTP请求给后端产权服务进程。软件产权服务进程首先会运行相关业务逻辑服务，判断是否需要调用用户管理进程获取用户信息，再将需要存储的相关数据传输到数据存储进程。数据存储服务进程根据产权服务的逻辑，选择通过IPFS进程进行数据存储，还是将核心上链数据提供给Fabric进程进行区块链存储。最终，产权服务进程将结果返回给交互进程展示。

3.3.5　系统物理视图

图 3.6描述了在区块链网络每个节点，系统在物理机器上的部署情况。按分层结构和微服务设计进行物理部署。证书工具单独部署可供下载直接使用。前端单独部署，业务层软件知识产权服务和区块链服务通过Docker部署到kubernetes集群，向外提供服务。数据层考虑到区块链的特性，分为三个部分部署到kubernetes集群。Hyperledger Fabric 单独部署，节点既作为联盟链背书节点提供区块链交易验证与发起服务，同时也作为共识节点对其他节点的交易提案进行共识。节点中部署的chaincode智能合约提供区块链存取数据服务。部

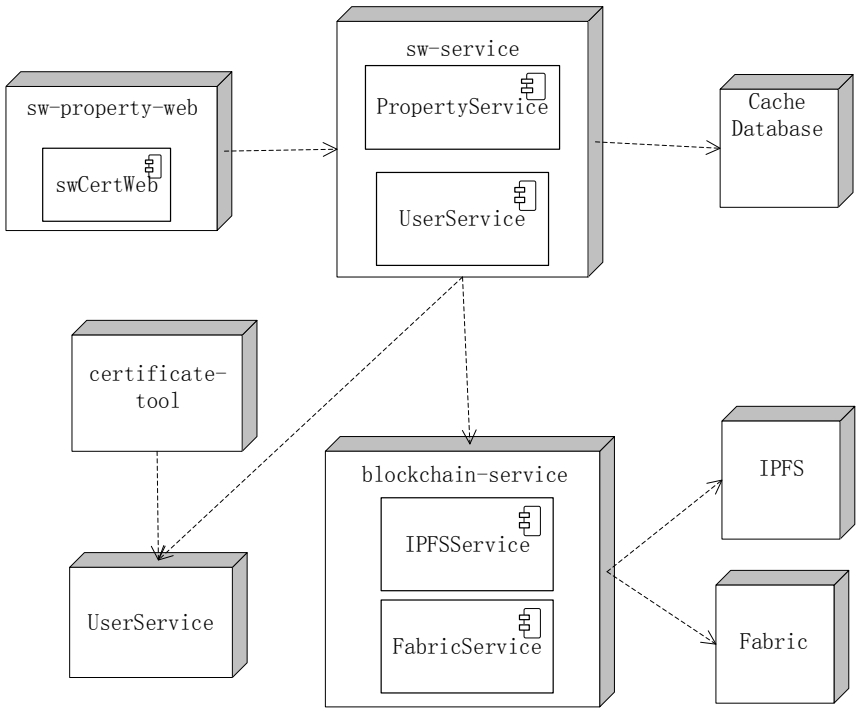


图 3.6: 系统物理视图

署IPFS节点加入IPFS网络，IPFS 提供大量超媒体文件存储服务，缓解区块链的数据存储压力，提高系统性能。部署TiDB集群作为缓存数据库，加快数据访问效率，系统不用每次处理查询请求都要从IPFS查找文件解析数据。这样的物理结构是遵循微服务的理念，保证业务的变化不影响系统结构，只会对每个单独的服务内部进行修改，所有的服务内部都被屏蔽只向外提供接口。服务之间通过HTTP 或REST API等方式调用。

3.4　系统模块设计

从逻辑视图、开发视图、进程视图、物理视图四个角度分析了系统的整体结构，将系统划分成了不同服务模块。本节，从业务需求的角度，结合系统设计对系统的几个重要业务模块进行详细的设计，包括证书工具模块、软件产权认证模块、版本管理模块。每个模块包含前端交互界面，后端业务逻辑处理，区块链底层服务三个方面。

3.4.1　证书工具模块设计

证书工具模块主要是实现软件与证书之间的转换，以证书来代替真实软件在系统中传输。同时能通过证书来验证软件的真实性和未被篡改。

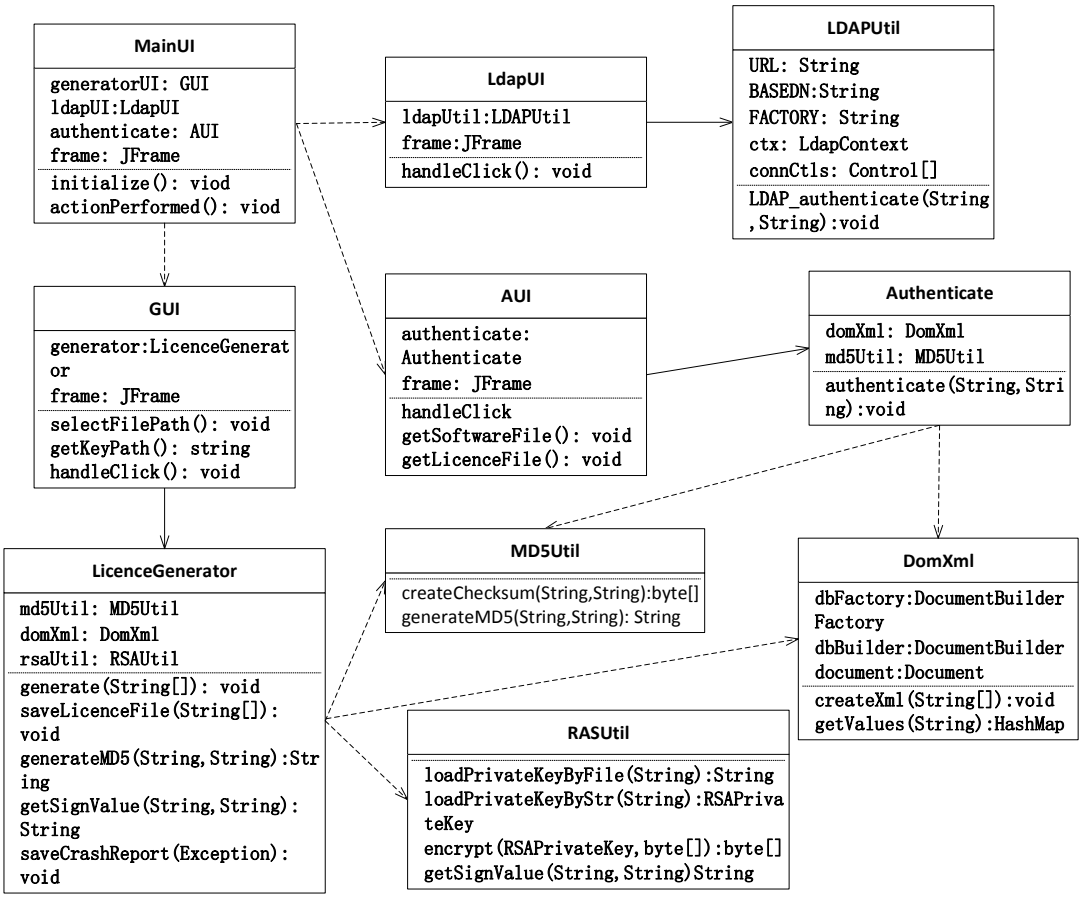


图 3.7: 证书工具模块类图

考虑到系统模块解耦，软件证书的生成和验证可以单独作为一个客户端工具独立运行。如果以静态服务的形式集成到前台系统会造成项目文件过大，首页加载速度低下。该服务会带来客户端浏览器压力。如果以后端服务的形式集成，违背了对用户软件隐私保护的目的，同时对软件制品的类型就有了条件限制，不利于维护平台的业务扩展性。作为一个独立的客户端工具使用，不仅对系统屏蔽了软件制品的真实实体，保护了软件开发方的资产隐私，同时还可以离线使用。

证书生成的内部实现类设计如图 3.7所示，其中MainUI、GUI、LdapUI、AUI是用户与客户端工具交互的界面，包括主界面、LDAP登录界面、证书生成界面、证书验证界面。LicenceGenerator类是响应界面证书生成请求的类，提供证书生成并保存到指定路径的服务。Authenticate类是响应界面证书验证请求的类，提供软件与证书一致性对比验证服务。DomXml类负责将需要包含的信息以Element的形式逐行写入XML文件以及从XMl文件中读取属性返回HashMap。MD5Util工具类提供对软件文件进行加密生成唯一散列值的服务。RSAUtil工具

类提供数字签名服务，使用LDAP 提供的用户私钥对软件散列值进行RSA加密生成用户数字签名。

该模块通过加密技术从软件制品文件中生成一份唯一证书文件。证书中要包含用户私钥加密的用户数字签名和时间戳等信息。该证书可以代表软件在系统中进行认证和存储并且不能反向解析得到软件。证书能和软件进行对比验证，证书中的hash能唯一标识软件制品，验证软件是否被替换或者篡改。

用户使用证书工具进入证书生成交互界面。输入用户名后，需要提供用户的私钥文件，如果用户已经使用LDAP验证过，证书工具会调用LDAP服务获取用户私钥。证书中不需要包含公钥，软件认证服务可以使用文件中的用户信息从LDAP获取。指定好软件制品文件路径和证书生成路径后，证书工具会获取软件文件并对文件进行MD5加密，生成与内容对应的唯一hash 值。之后使用用户私钥对MD5散列值进行RSA加密生成用户数字签名。其作用是在软件产权认证信息上链时，通过用户公钥解密用户签名获得加密的hash，对比软件hash，验证软件证书是否被篡改，保证网络传输中信息没有被篡改。以XML 的格式保存用户签名、散列hash、时间戳、用户信息、加密算法，生成证书文件并写入到指定路径。

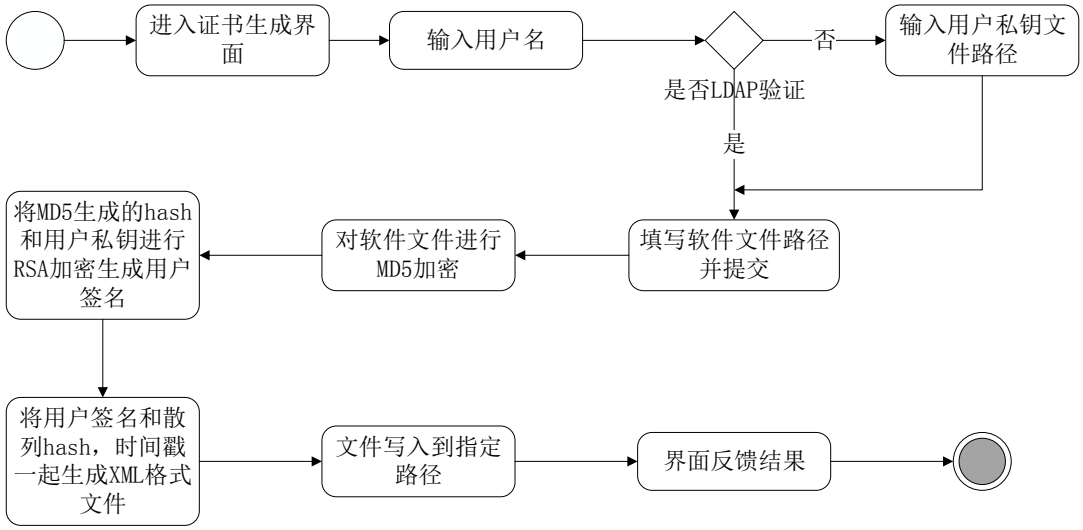


图 3.8: 证书生成流程图

图 3.8描述了证书的生成的具体流程。用户可以得到软件的证书文件进行后续的产权认证服务。软件证书包含软件的加密hash，hash值与文件内容一一对应，保证证书能验证软件的真实性和未被篡改。用户可以对比软件证书和相关软件，确定软件是否是软件证书标识的软件。

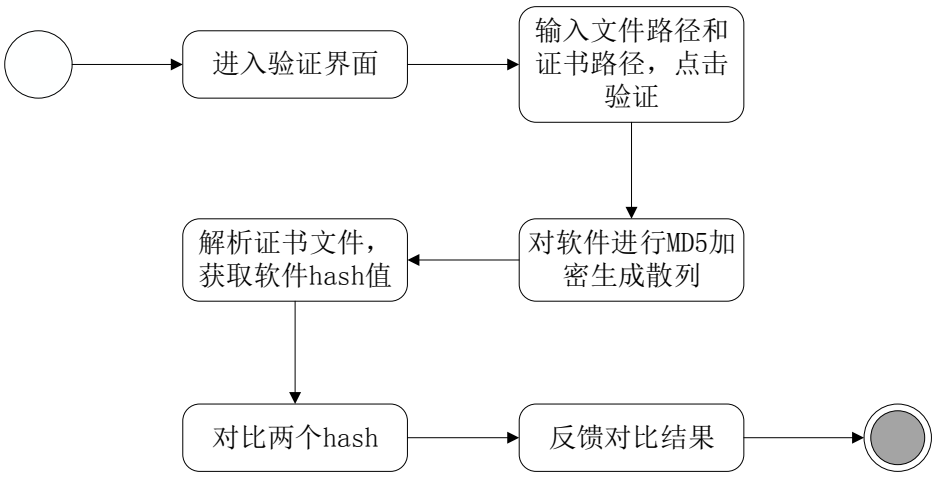


图 3.9: 证书验证流程图

如图 3.9所示的活动图展示了证书与软件对比验证的过程。用户需要选择目标软件文件路径和证书路径。工具对目标软件进行MD5加密生成散列值。散列值与文件内容相关，内容不变散列值不变。解析证书文件获取证书中保存的hash值，对比两个散列值就可以知道软件是否是证书所标识的原版软件。

3.4.2　产权认证模块设计

软件产权认证即软件知识产权的确权，通过区块链保证软件产权信息的安全性、不可篡改，所有授权节点的产权认证信息都会被区块链记录下来。区块链每个节点都有完整的产权认证信息账本，可以随时追溯软件知识产权的所属方，并且真实可信。软件产权认证模块的主要功能包括两个部分：(1)软件开发方对自己开发的软件进行知识产权认证，上传软件证书，将软件产权认证信息上链存证。(2)软件更新迭代时，软件开发方更新该软件的产权信息，上传新的证书，将软件产权更新信息记录到链上。

由于系统基于区块链技术进行数据存储，区块链只允许“增和查”的服务，不提供对账本已有数据进行更新。因此系统对于软件产权信息的更新设计是把软件产权的新版本作为新的产权认证记录提交到链上，通过Leaf生成的标识来追溯软件版本的变更过程。

如图 3.10描述了软件认证详细类图，系统分层设计将类分为了四层。SoftwareController类是对外提供的接口，负责处理前端请求返回处理完的业务数据，具体的逻辑处理依赖于SoftwareLogic。Logic层负责调用各个服务完成软件认证业务，这样的设计保证了服务的单一性和低耦合，每个服务只向下层依赖，服务之间的逻辑由逻辑层来实现。SoftwareLogic负责处理软件认证具

体业务逻辑，包括解析软件证书，用户数字签名和软件加密散列值是否一致等验证逻辑，以及验证通过后发起软件产权认证上链操作。SoftwareService类负责上传非关键数据到IPFS并将核心数据转化为区块链服务所需的Doc对象，再由DocDao实现认证信息上链。IPFSService类提供Object对象转换为XML格式进行文件存储以及文件读取并转换为所需的实体对象，依赖于FileService类提供的从IPFS中上传文件和下载文件功能。DocService类是区块链服务对外提供信息上链服务类，由产权业务服务的DocDao类调用，该服务会执行节点上chaincode智能合约实现软件认证信息上链。

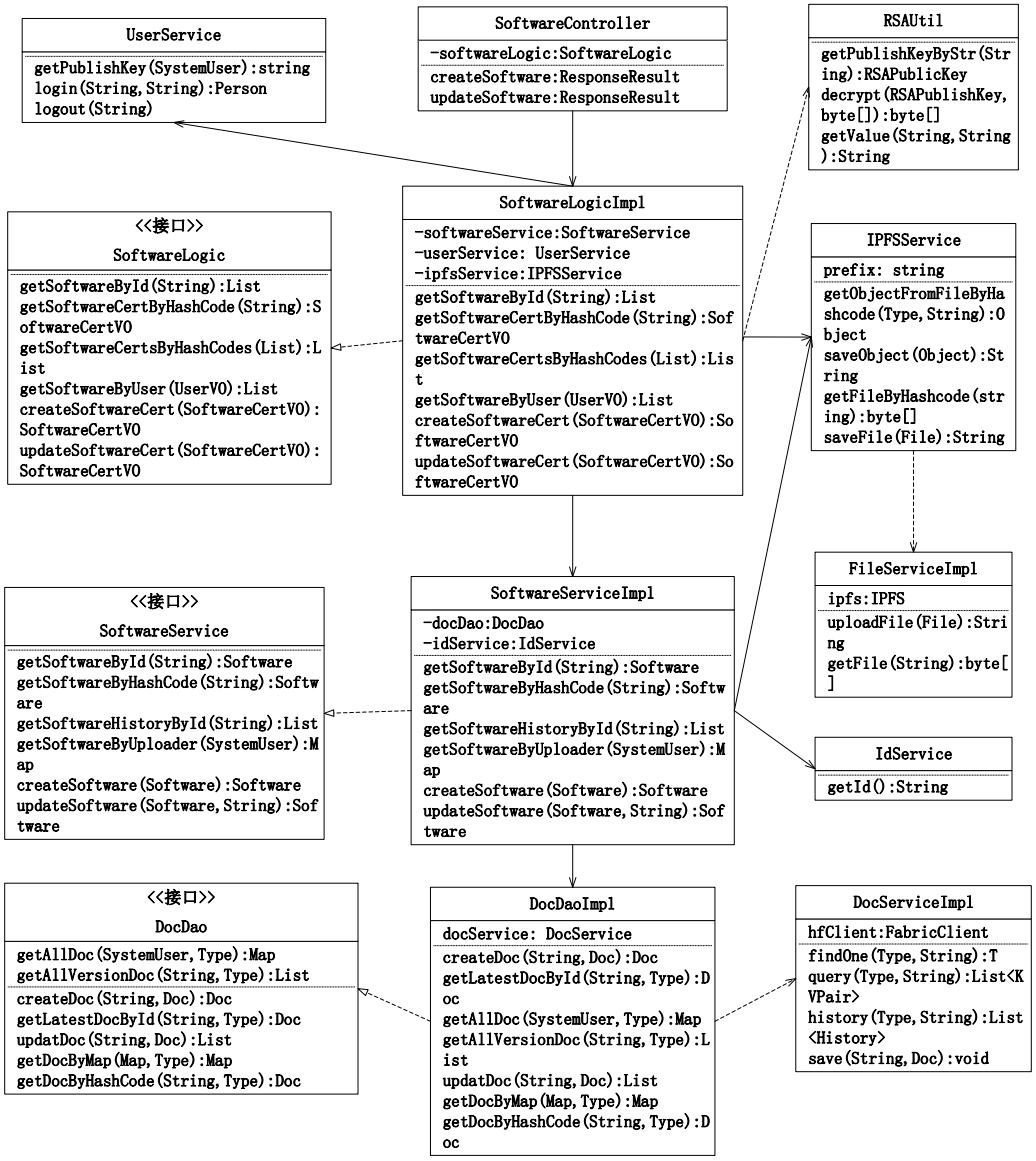


图 3.10: 软件认证模块类图

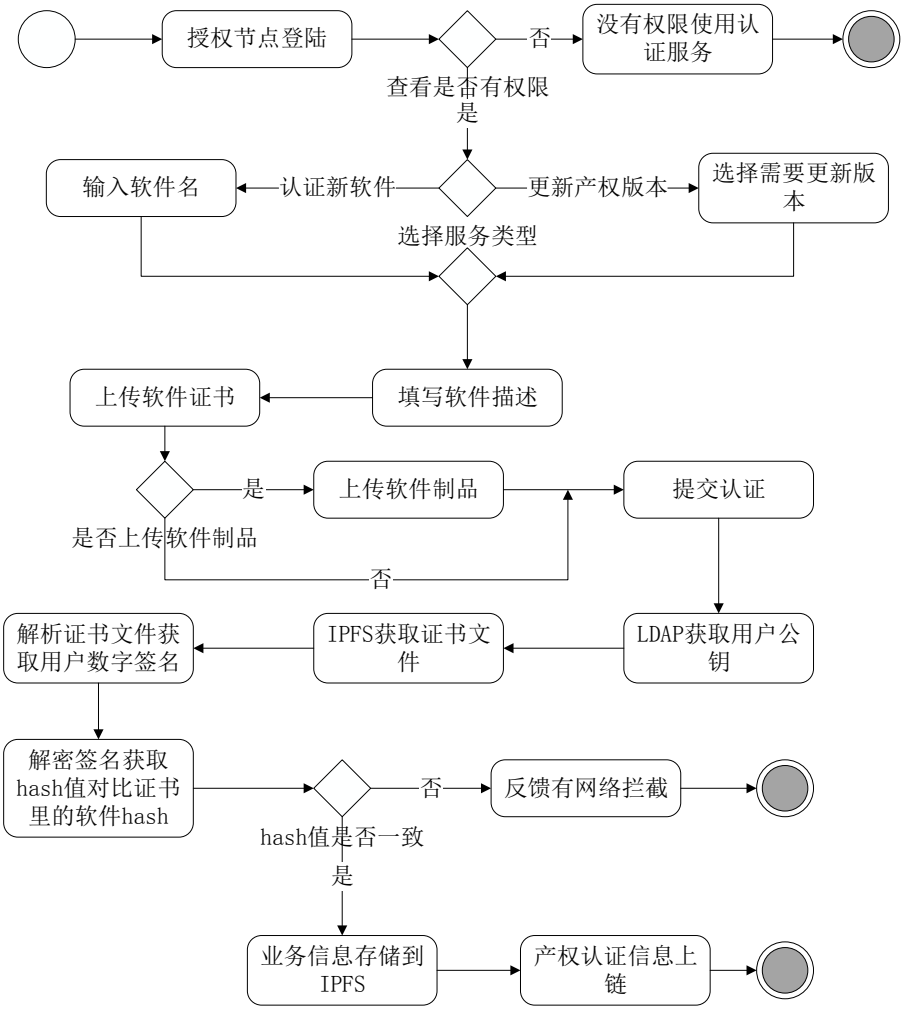


图 3.11: 软件产权认证活动图

软件产权认证的具体流程如图 3.11所示，分为五个步骤。首先，查看登陆用户在授权节点Fabric CA里的身份权限，是否生成证书允许用户访问区块链。接着，选择软件认证的服务类型，可以是认证新软件，可以是更新已有软件知识产权的版本。软件开发方填写产权认证所需的软件相关信息。如果软件开发方选择的是更新产权版本就会自动保留上一个版本的部分信息，包括软件名，软件描述等，用户可以更改这些信息。然后，软件开发方必须上传软件证书到IPFS文件存储网络，获取IPFS根据文件内容生成的hash 地址。开发方可以根据情况选择是否上传软件制品。完成上传后提交认证表单。其次，后端认证服务从LDAP获取用户公钥，通过上传证书返回的hash值从IPFS里获取证书文件，解析证书文件得到用户私钥加密的数字签名。使用用户公钥对数字签名进行RSA解密，对比解密后的hash 值和证书里存的软件hash值是否一致。如果不

一致反馈给用户出现了网络拦截，有人试图修改产权信息。最后，将软件相关的业务信息以XML文件的形式存到IPFS网络，利用Leaf生成软件产权id，作为该软件的标识。将id、证书hash、用户信息、以及业务信息hash 通过智能合约存储到区块链上。

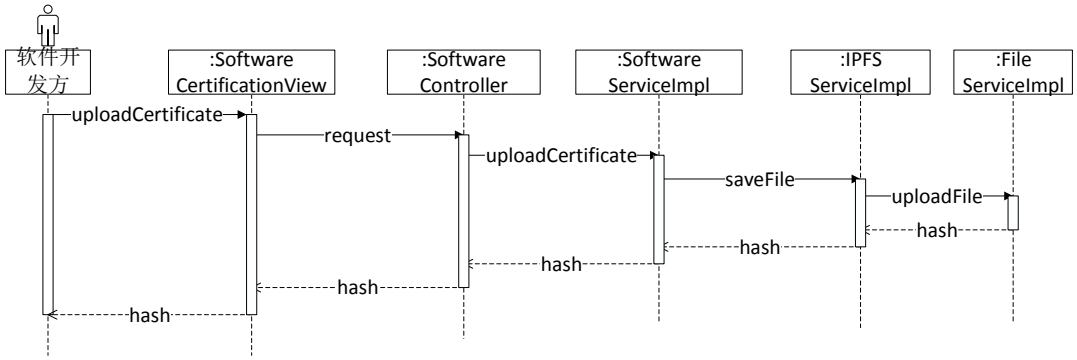


图 3.12: 证书上传时序图

软件产权认证过程中，软件证书以及软件制品上传都是存储到IPFS网络，会根据上传的内容生成一个唯一hash值。图 3.12描述了系统上传软件证书文件的过程。区别于使用oss上传到阿里云服务器的存储方式，IPFS是去中心化的存储网络，同时它是基于内容相关的文件存储模式，以hash值作为文件标识。系统采用这样的存储方式一方面是使区块链的数据存储尽可能薄，另一方面是考虑到系统的容错机制，某些存储节点崩溃不会影响到整个系统的运行。

3.4.3　版本管理模块设计

考虑到软件具有一定使用周期，需要迭代更新的特性，随着软件迭代，软件知识产权也需要更新认证信息。对软件知识产权的管理包括：管理软件知识产权所有版本变更，查看用户所有的软件产权，搜索特定的软件产权等服务。

根据系统的开发视图，系统的分层设计是以实体对象来划分，软件产权的管理涉及到的类如类图 3.13所示。版本管理模块涉及到对软件产权的查询和产权版本的管理。softwareController类提供软件知识产权版本管理相关业务接口，接收前端平台发起的HTTP请求。SoftwareLogic类负责提供指定条件查询软件产权、产权历史各版本查询、软件产权授权等业务逻辑处理服务。SoftwareService实现业务数据的查询，包括从缓存数据库查询数据、从IPFS解析实体属性、发起区块链数据查询、交互对象与持久化对象之间的转化。SoftwareAttributeDao实现使用SQL语句执行TiDB数据库操作，查询缓存的数据，减少IPFS查询并解析文件的次数，快速响应客户端请求。DocDao调用区

块链数据服务，将获取的抽象对象Doc转化为Software实体。DocService负责调用Fabric提供的智能合约中的query方法查询区块链上存储的数据，并将数据从字符串处理成继承Doc类的实体。

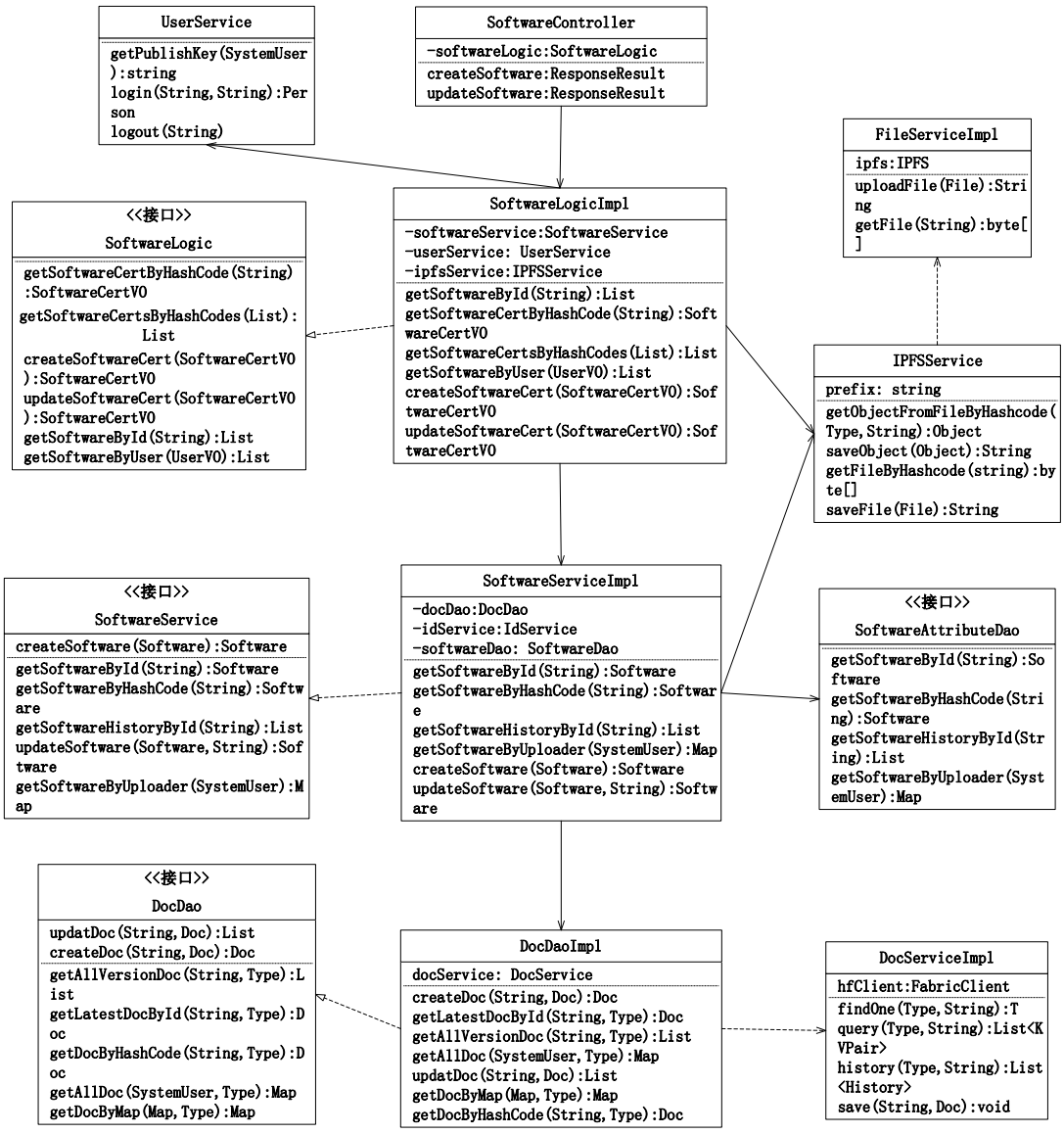


图 3.13: 版本管理模块类图

软件开发方进入软件产权管理界面，可以选择指定的软件查询或者获取用户的所有软件产权列表。前端会缓存用户的最近查询记录，用户可以直接按照前端缓存的上一次的查询表单查询指定的软件产权。前端发起HTTP请求查询用户所有软件产权，用登录用户作为从Fabric CA获取了授权证书的SystemUser，查询区块链上用户已有的软件产权。区块链服务会先以查询

表单为查询的条件调用区块链服务获取链上认证数据，将查询到的结果数据转换成KVPair<Doc>键值对对象数组。然后后端业务服务会通过每个键值对对象的softwareAttrHash 属性作为key查询TiDB是否有缓存，如果有数据缓存就不需要再耗时去IPFS查询并解析文件。如果没有缓存，通过属性hash去IPFS找到对应的文件，解析存储的软件实体的属性。最终将软件产权数据转换为VO实体格式，并以自定义的Response格式返回给前端状态管理中心Store，触发页面局部渲染。图 3.14描述了软件产权查询的两种不同方式和其流程。

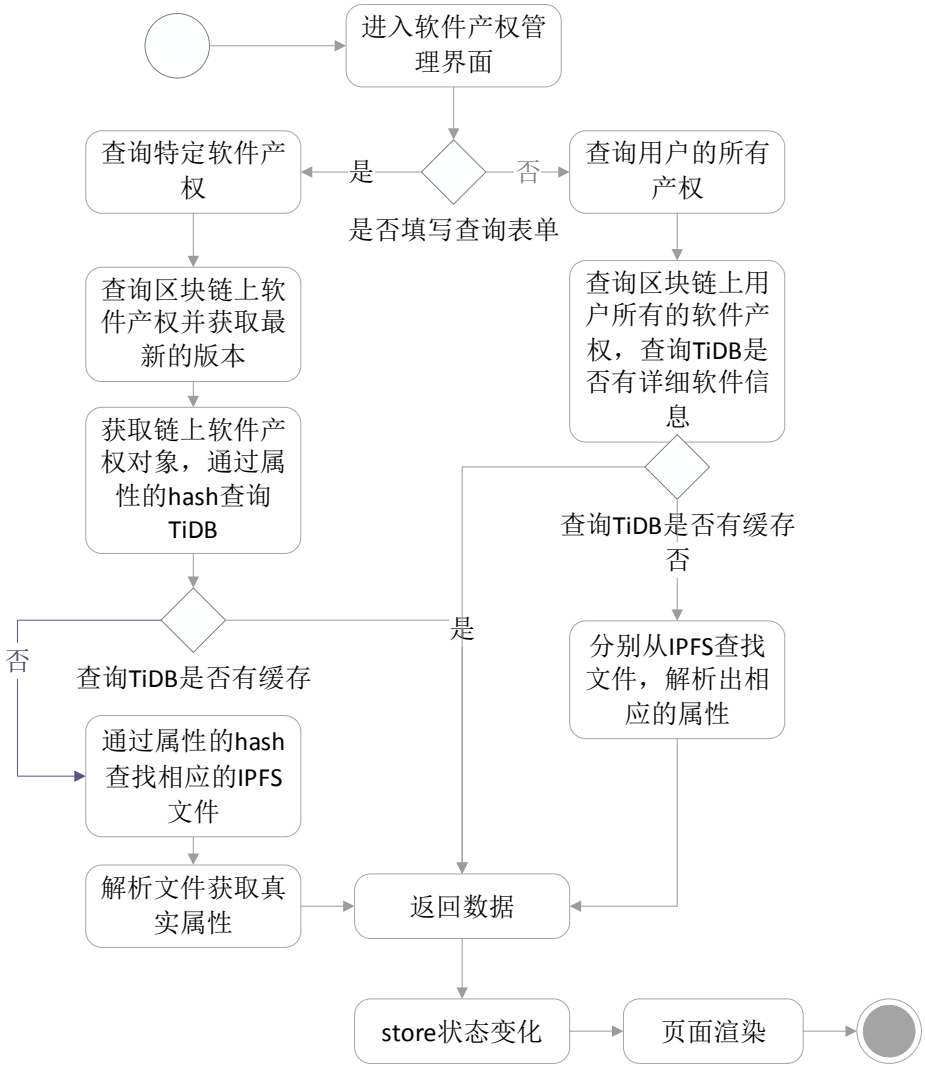


图 3.14: 软件产权查询活动图

当软件开发方获取已有产权的所有历史版本，区块链服务会通过id作为key查询属于该产权的所有历史版本。如流程图 3.15 所示。软件产权实体的id属性，不同于传统数据库实体的唯一标识key。该属性是用来追溯软件的

所有版本。这样设计的目的是利用区块链可追溯的特性，按时间戳顺序通过id获取账本中所有的软件产权历史版本认证信息。账本数据经过预处理转换为History类型，再返回给业务服务解析数据，转换为前端VO实体格式，封装Response返回给web，页面上以时间顺序展示软件知识产权版本变更。

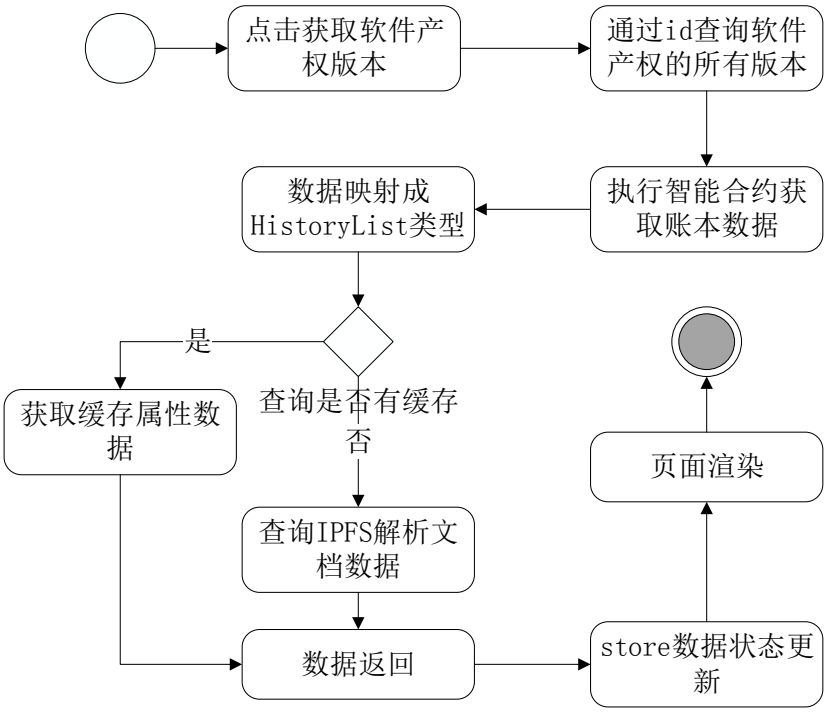


图 3.15: 版本查询活动图

3.5　数据库实体设计

遵循领域驱动设计，系统的各个服务内部使用逻辑实体，与外部交互使用交互实体，这样使得每个服务模块都有各自的系统边界，彼此解耦。系统基于区块链结合IPFS星际文件系统实现数据存储，涉及到的关键实体包括证书实体certificate，软件产权实体software_property，用户实体user，软件产权属性实体software_attribute，区块链存储Doc实体。图 3.16是系统ER图，描述了实体的设计与实体间的关系。

系统实体中，user实体是系统从LDAP用户管理服务获取的用户信息实体，后端服务通过实体转换成Fabric CA验证所需的system_user实体。其中domain为主属性，用来标识节点配置的LDPA用户管理服务。name属性是LDAP上用户的标识名,password是用户登陆LDAP的密码。通过domain、name和password属性登陆LDAP获取用户信息的访问权限，获取用户公钥解密软件证书的数字签名。

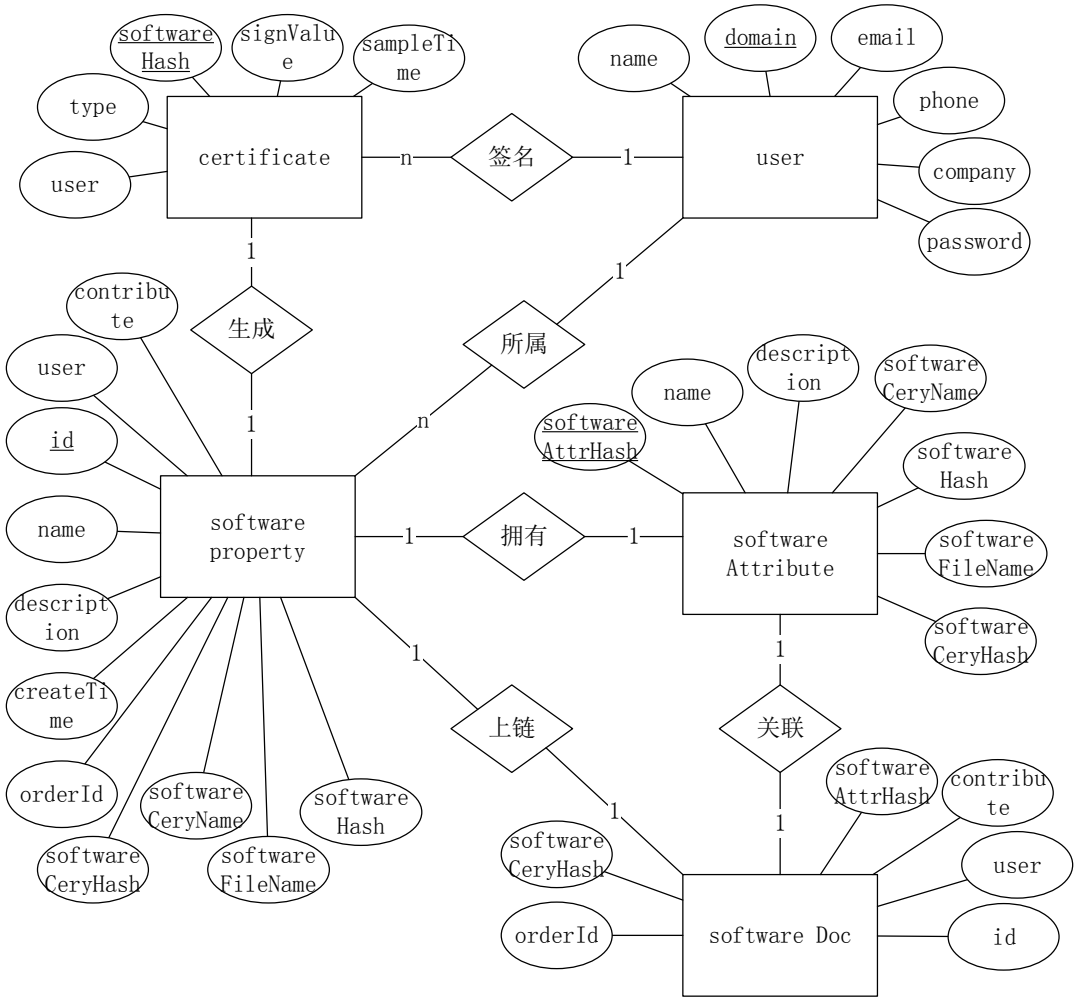


图 3.16: 实体关系图

证书实体certificate是证书工具生成的证书信息实体，以XML格式文件存储到IPFS网络，便于系统服务端解析成证书对象。证书实体属性如表 3.8所示。user属性是生成证书的用户信息对象，sigmValue是由用户在LDAP中的用户私钥经过RSA加密生成的用户数字签名。type属性是证书生成过程中对软件文件的加密散列算法类型，softwareHash是由所选算法生成的软件加密hash。sampleTime是证书生成的时间戳，只用于表示证书生成时间，软件知识产权认证时间由区块链交易上链时间戳记录。

软件知识产权实体software_property的属性如表 3.9所示。id是知识产权的唯一标识，由Leaf分布式ID生成系统生成，它标识了区块链上同一软件的不同版本，便于版本管理。createTime是认证信息上传到区块链的时间戳。orderId是软件产权授权时提供的订单实体的id。softwareCertHash是软件知识产权证书存储

表 3.8: software_certification实体表

字段	含义	类型	可空	备注说明
user	生成证书的用户	User	否	User是用户对象实体
type	加密算法类型	string	否	可选的散列加密算法
softwareHash	软件加密hash	string	否	
signValue	用户数字签名	string	否	证书的生成用户标识
sampleTime	时间戳	string	否	

在IPFS网络生成的hash值。IPFS是内容相关存储网络，可以通过hash值从IPFS查询文件，每个证书都是唯一存在具有唯一hash值。系统通过softwareCertHash而不是id，唯一标识区块链上某一版本产权实体。softwareCertName是证书的文件名，softwareFileName是软件文件名。通过hash从IPFS中获取的文件是以二进制流的形式，需要进行文件类型转换，通过文件名将二进制流封装成对应文件名后缀格式。softwareHash是用户上传软件制品到IPFS的存储hash。认证过程用户可以选择上传软件制品，软件会被存储到IPFS中返回hash，而且IPFS保证了不会有相同文件被重复存储。user是系统中进行软件认证的用户，是系统中认证的软件知识产权所属方。txId是认证信息上链时区块链执行的交易提案标识，通过txId可以在Fabric-explorer查询到认证信息所属区块。contributorMap 字段是可选字段，为了业务扩展性而设计，考虑到真实存在的协同开发场景，存储各个开发方知识产权分配占比。

表 3.9: software_property实体表

字段	含义	类型	可空	备注说明
id	软件id	string	否	标识软件一系列版本
name	软件名	string	否	
description	软件描述	string	是	
createTime	认证时间戳	string	否	区块链交易上链时间戳
orderId	所需订单id	string	是	平台产权授权的订单id
softwareCertHash	软件证书hash	string	否	key唯一标识
softwareCertName	证书文件名	string	否	存储到IPFS的证书文件名
softwareHash	软件制品hash	string	是	用户上传的软件在IPFS的hash
softwareFileName	软件文件名	string	是	
user	产权所属用户	User	否	User是用户实体
txId	区块链交易Id	string	否	数据上链时发起的交易提案Id
contributorMap	产权分配	map	是	扩展属性，产权可能由多个外包人员开发

表 3.10: software_attribute实体表

字段	含义	类型	可空	备注说明
softwareAttrHash	软件属性对象hash	string	否	key唯一标识
name	软件名	string	否	
description	软件描述	string	是	
softwareCertHash	软件证书hash	string	否	key唯一标识
softwareCertName	证书文件名	string	否	存储到IPFS的证书文件名
softwareHash	软件制品hash	string	是	用户上传的软件在IPFS的hash
softwareFileName	软件文件名	string	是	

系统的数据存储方式不同于传统分布式数据存储模式，使用基于区块链结合IPFS文件存储系统的分布式数据存储解决方案。为了解决区块链账本随着吞吐量的增长，节点所需数据存储空间快速膨胀问题，区块链上不存储大部分业务属性数据，包括名称、描述、链上记录时间戳、证书文件名、软件文件名、软件hash。这些软件产权属性数据作为software_attribute实体对象以XML 文件格式存储到IPFS网络，同时该实体作为缓存数据库TiDB的software_attribute表持久化对象，如表 3.10所示。software_attribute实体以IPFS网络根据文件内容生成的地址hash作为唯一标识。系统不需要每次请求区块链软件知识产权数据都根据属性数据hash重复查询IPFS并解析文件，直接从TiDB读取数据，快速响应用户的软件产权数据查询请求。

表 3.11: 区块链账本软件实体表

字段	含义	类型	可空	备注说明
id	软件id	string	否	标识软件一系列版本
orderId	所需订单id	string	是	平台产权授权的订单id
softwareCertHash	软件证书hash	string	否	软件知识产权唯一标识
softwareAttrHash	产权属性hash	string	是	用户上传的软件在IPFS的hash
user	产权所属用户	User	否	User是用户实体
contributorMap	产权分配	map	是	系统扩展属性，产权可能由多个外包人员开发

区块链账本中存储的数据结构应尽可能的薄，以最少的数据量存储软件产权核心信息。这也是系统采用IPFS文件存储网络的重要原因，将大量的业务数据存储在IPFS上，保证链上只存储核心数据，又能够通过链上数据从IPFS得到完整的软件产权实体信息。表 3.11描述了区块链上存储的软件知识产权实体属性。将软件产权的名称、描述、制品路径等业务相关的非核心数据存储到IPFS，链上只需要存储能够标识软件知识产权与用户之间的所属关系的核心数据，包

括id、订单id、证书存储hash、属性存储hash、所属用户方、产权分配。在区块链服务子系统，为了方便交互，软件产权认证数据会转换成继承了Doc抽象类的software_doc实体存储到链上。

3.6　本章小结

本章首先对系统进行整体描述，描述系统总体规划以及相关的功能。然后对系统进行需求分析，确定系统涉众，描述系统所需满足的功能性需求和非功能性需求。接着分别从逻辑视图、开发视图、进程视图、物理视图描述了系统的总体设计。结合系统的总体设计和业务需求，将系统拆分成不同的核心业务模块进行了详细设计，包括证书工具模块、产权认证模块、版本管理模块。最后对系统内部的交互实体、领域对象、以及区块链存储数据进行了详细描述。

第四章　基于区块链的软件知识产权认证系统的实现

4.1　证书工具模块

使用软件证书生成工具将软件制品进行加密，将加密后的哈希散列以及用户数字签名，时间戳共同生成一份XML格式软件知识产权证书，证书唯一对应某一软件制品。通过证书在系统内进行产权认证，既能保证软件开发方的软件隐私安全防止软件源代码泄露，又能满足软件制品的类型多样性需求。下文将具体实现软件证书生成工具需要提供的两个服务，证书生成与证书验证。

4.1.1　证书生成子模块实现

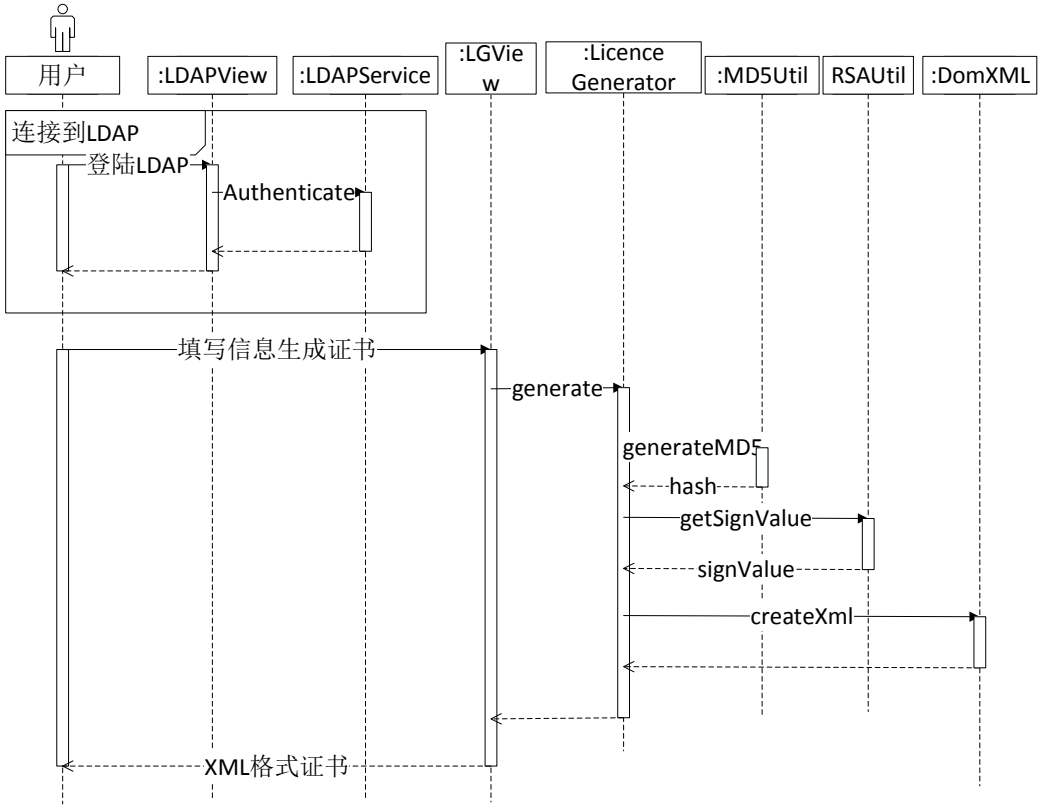


图 4.1: 证书生成子模块时序图

依据软件证书生成流程的设计，用户需要填写用户信息，选择软件文件路径，用户私钥文件，以及加密方式和输出路径，然后发起证书生成操作。方法

实现主要涉及到对用户选择的软件进行MD5 加密，获取用户私钥生成证书数字签名，生成XML格式文件。其中用户私钥可以通过工具登陆验证LDAP获取。

具体生成证书过程中，类之间调用关系如时序图 4.1所示。软件证书生成过程的核心实现类是LicenceGenerator类，同时依赖MD5工具类、RSA工具类以及DomXML类。由LGview视图向LicenceGenerator类发起证书生成操作，LicenceGenerator 类调用MD5工具类获得加密hash，调用RSA工具类生成数字签名，最后调用DomXml类完成XML格式证书生成，并输出到指定文件路径。

```
private void generate(String username, String uKeyId, String alg, String keyFile, String
sourcePath, String targetPath) {
        String MD5value = generateMD5(sourcePath, alg);
        //RSA 加密
        String signValue = getSignValue(MD5value, keyFile);
        saveLicenceFile(username, uKeyId, alg, MD5value, signValue, targetPath);
    }
```

图 4.2: 证书生成代码

LicenceGenerator类的私有方法generate实现了生成软件证书的功能，如图4.2所示。参数是用户在UI界面上的输入值。首先是对软件文件使用MD5Util工具进行哈希散列加密，然后调用RSAUtil的getSignValue方法将加密散列值和用户的私钥文件加密生成用户数字签名。最后通过saveLicenceFile方法调用DomXml类的createXml 方法，将用户信息、MD5散列值、加密方式、用户数字签名一起生成XML格式文件输出到指定路径。

```
public String getSignValue (String value, String path){
        String signValue = "";
        try {
            //获取用户私钥
            String rsa = loadPrivateKeyByFile(path);
            RSAPrivateKey privateKey = loadPrivateKeyByStr(rsa);
            //加密
            byte[] a = encrypt(privateKey, value.getBytes());
            signValue = a.toString();
        } //省略后续信息处理
        return   signValue;
    }
```

图 4.3: RSA加密代码

证书文件中留存有用户的数字签名，一方面是为了保证证书在认证数据上链前没有被篡改，另一方面是为了将软件证书与LDAP用户关联起来，保证追溯系统用户的真实身份可以关联到节点自身维护的LDAP用户，在维权时能提供完整的证据链一直追溯到节点的授权用户。图 4.3所示为RSA加密生成数字签名的代码。loadPrivateKeyByFile方法从私钥文件中读取出私钥字符串，loadPrivateKeyByStr方法将私钥字符串实例化为PKCS8EncodedKeySpec对象，还原成RSA私钥。encrypt方法使用RSA 私钥实例化cipher 类，初始化为加密模式，对MD5值进行RSA加密。在用户私钥的获取上，工具提供用户登录LDAP的功能，由LDAPService连接LDAP验证用户登录，自动获取LDAP用户私钥。

4.1.2　证书验证子模块实现

软件证书信息被存储到区块链上，保证证书信息不可篡改。同时证书的软件加密值与软件制品文件内容相对应，因此从区块链服务获取的软件证书能追溯到生成证书的软件制品。证书工具提供验证软件证书与软件制品之间是否一致的服务。防止恶意开发者篡改软件源代码，伪造认证信息。根据验证证书与软件的一致性的流程设计，证书验证包括对目标软件进行加密，解析证书文件获取散列值，对比两个hash值验证一致性。

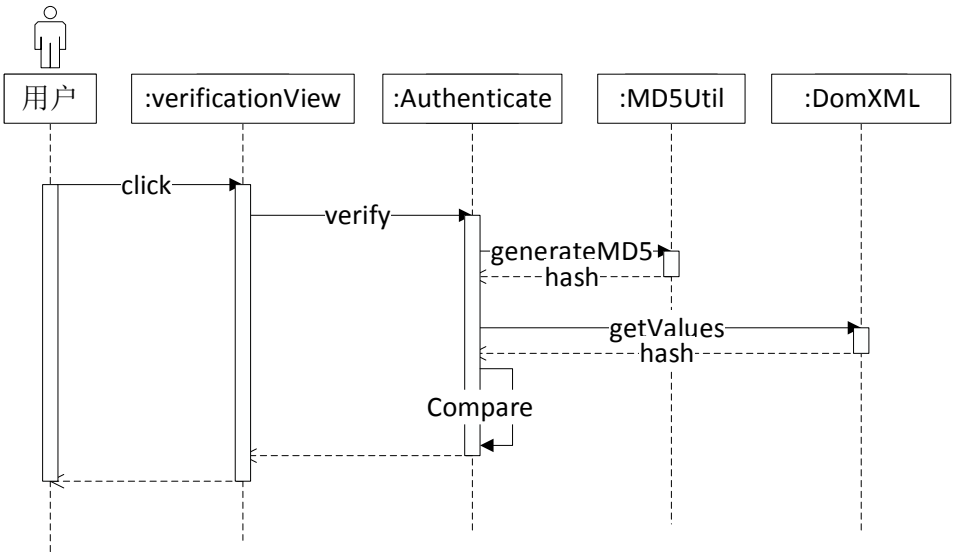


图 4.4: 证书验证子模块时序图

如图 4.4所示，证书验证的时序图。用户从主界面点击验证入口，进入证书验证界面。点击证书验证界面的验证按钮触发验证业务，系统调

```
public void authenticate(String file, String licencePath) {
        try {
            HashMap<String, String> licence = new HashMap<>();
            licence = domXml.getValues(licencePath);
            String alg = licence.get("type");
            String value = licence.get("value");
            String newValue = md5Util.generateMD5(file, alg);
            if (value.equals(newValue)) {
                //省略展示逻辑
            } else {
            //省略验证失败后续异常提示代码
    }
```

图 4.5: 证书验证代码

用Authenticate类执行验证操作。Anthenticate类调用MD5加密工具类获得hash值，再从DomXML类获得hash，对比完成验证。

验证证书与软件的一致性主要通过Authenticate类实现业务逻辑。具体实现的类方法代码如图 4.5所示。Authenticate类实例调用authenticate方法，实现证书与软件的MD5散列值对比。其核心在于MD5加密的唯一性，内容不同的文件加密出来的hash值会不同。authenticate 方法先调用DomXML类对证书文件进行解析，得到证书内容的HashMap。再通过MD5Util实现对目标软件的加密，得到目标软件散列值。最后对比证书记录的hash值与目标软件散列值，验证证书标识的软件与该软件制品是否一致。

MD5Util类的generateMD5方法提供文件加密功能。方法内部调用工具类的私有方法createChecksum方法。createChecksum方法根据文件路径以字节流的形式读取软件文件数据，然后实例化MessageDigest类，使用MessageDigest类的update方法来处理文件字节流生成128bit的加密散列值。

4.2　产权认证模块

用户可以在任意部署了系统平台服务的区块链节点快速认证软件知识产权。认证信息经过背书节点校验后执行智能合约自动发起交易提案写入区块链，不需要任何第三方参与。经过一段时间共识后，所有节点账本都有该软件产权认证信息，保证了认证信息全网维护、无法被篡改。

根据软件产权认证模块的设计，软件产权认证服务支持认证新的软件知识产权以及更新已有软件产权的版本。然后调用不同的业务服务接口进行信息验

证，对于底层区块链服务实现来说，对外提供服务接口不区分产权版本更新过程和产权创建过程，都是向联盟链发起一个TransactionProposaRequest交易提案来执行chaincode智能合约的invoke方法向区块链写入新的认证信息。软件知识产权认证服务实现过程如时序图 4.6所示。

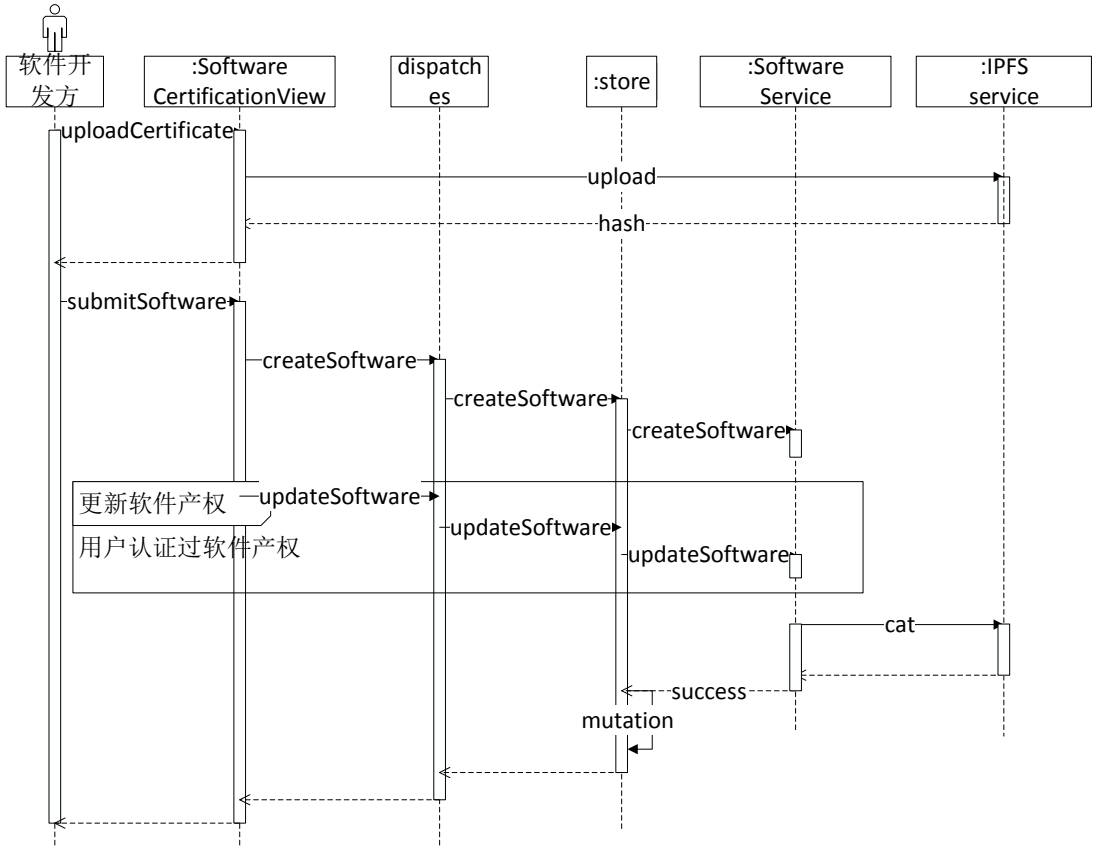


图 4.6: 软件产权认证时序图

认证过程涉及到的前后端主要类如表 4.1所示。组件类是前端系统单页面应用实现的基础，通过路由导航和局部渲染实现组件拼接生成页面。认证页面由SoftwareCertification、UploadSoftwareForm组件实现。用户异步操作与数据状态变更由Dispatches和Store类实现。SoftwareController类和FileController类接收前端请求并执行产权认证和文件上传处理逻辑。SoftwareLogic类和FileLogic类实现具体的认证逻辑和文件上传逻辑。SoftwareService类实现认证信息存储服务，IPFSService类实现上传证书和获取证书解析证书信息服务，UserService类提供用户公钥，IdService类通过Leaf生成id，RSAUtil类使用用户公钥解密证书中用户数字签名。DocDao类管理数据查询，负责调用区块链子服务模块对外提供的接口，获取所需的数据并将Doc对象转换为service层所需的实体对象。

表 4.1: 软件知识产权认证相关类表

类名	分类	功能说明
SoftwareCertification	组件类	根据路由导航实现局部渲染展示软件证书页面
UploadSoftwareForm	组件类	产权认证表单组件
Dispatches	异步操作类	分发界面异步请求，分离组件视图与业务
Store	状态管理类	中心化管理前端平台数据状态
SoftwareController	控制类	提供软件认证服务对外接口，接收软件认证请求
FileController	控制类	提供文件上传接口
SoftwareLogic	逻辑类	处理软件知识产权认证过程中的业务逻辑
FileLogic	逻辑类	处理文件上传服务的业务逻辑
SoftwareService	服务类	创建软件认证信息实体并调用下层服务存储信息
IPFSService	服务类	实现证书文件上传服务
UserService	服务类	从LDAP获取用户公钥
IdService	服务类	使用Leaf生成软件产权id
RSAUtil	工具类	解密用户数字签名
DocDao	数据访问类	调用区块链服务发起认证信息上链请求

用户通过LDAP账户登录平台，获取节点区块链授权证书后，选择软件产权认证服务，服务默认是用户认证新的软件产权。用户可以通过界面上switch组件切换到知识产权版本更新服务，用户选择已有软件产权，数据从中心化状态管理Store中获取，不需要触发实时请求，这样就不会出现数据加载延迟造成画面卡顿，同时减少网络请求。用户上传证书文件，界面显示上传进度条等待服务端存储文件到IPFS并返回由文件内容生成的hash。用户填写认证信息表单并确认提交。前端系统首先会进行表单校验，确认软件名和证书hash已经存在，再通知状态管理中心Store发起认证请求。

软件知识产权认证页面复用了认证表单组件，通过父子组件间的参数传递完成特性化设置形成Vue单组件文件，页面加载通过Vue模板引擎和vue-router路由导航实现局部渲染生成HTML。表单提交前要经过表单验证过程，通过ref标识从界面模板中获取submitSoftware表单中el-form组件，el-form组件实现了类方法validate。validate方法会把表单组件的prop 属性中rules对象作为验证规则参数。Validate方法验证完成后会调用Promise.resolve()返回一个Promise对象，validate的参数就是Promise.resolve()执行的回调函数，会向回调函数中传入两个参数，valid是表单验证的结果，obj是未通过的验收属性，对应于rules里需要检查的属性。表单验证通过后，系统会执行判断逻辑，判断表单组件服务类型的开关参数是否为true。页面没有向表单组件传入开关参数允许更新已有软件版本，则系统提交表单，请求创建新软件。如果页面允许组件提供

产权更新服务，系统会根据用户选择的Switch 值选择发起上传新软件请求或者更新已有软件版本请求。creteSoftware方法会通知Store发起软件认证请求，updateSoftwareVersion则会保留选择的软件产权id属性作为需要更新的软件产权发起版本更新请求。

```
@Override
public ResponseResult createSoftwareCert(SoftwareCertVO softwareCertVO) {
        ResponseResult responseResult = new ResponseResult<SoftwareCertVO>();
        Software software = this.softwareVO2software(softwareCertVO);
        //获取用户公钥
        String key = this.userService.getPublishKey(software.getUser());
        //从 ipfs 获取证书
        SoftwareCert cert = this.ipfsService.getObjectFromFileByHashcode
(SoftwareCert.class,software.getSoftwareCertHash());
        //对比解密的签名和软件 MD5 值
        if (RSAUtil.getValue(cert.signValue,key) == cert.value) {
            SoftwareCertVO result = this.software2SoftwareVO
(softwareService.createSoftware(software), null);
            responseResult.setData(result);
            responseResult.setCode(200);
        } else {
            responseResult.setCode(403);
            responseResult.setMsg('证书验证失败，拒绝写操作');
        }
        return responseResult;
    }
```

图 4.7: 软件产权认证逻辑代码

在服务端接收到用户软件产权认证请求后，控制层调用SoftwareLogic类的createSoftwareCert方法，如图 4.7所示软件知识产权认证的实现代码。将参数对象预处理成软件产权实体后，createSoftwareCert方法从LDAP 获取用户公钥。再通过文件服务从IPFS 中获取证书hash 对应的证书文件并解析XML文件转换成软件证书对象，调用RSAUtil 工具类对证书中的数字签名进行解密。解密过程和证书生成过程中的加密过程相反，使用初始化为RSA解密模式的cipher类实例，通过用户公钥解密数字签名得到MD5 值，与证书中的MD5值对比。对比一致则通过createSoftware方法进行软件认证数据的上链存储。对软件对象实体中的业务属性封装成XML 格式文件上传到IPFS 网络并返回软件属性散列值，再将属性hash值和软件实体转换成SoftwareDoc实体，以Leaf生成的id作为key存储

到区块链上。最后以ResponseResult格式作为后端返回数据的统一格式，如果散列值对比不一致，则将错误信息添加到responseResult中，并加入系统定义的错误码返回给前端。状态管理中心Store 会解析后端的responseResult 中的信息并将认证结果反馈给用户。

4.3　版本管理模块

用户能通过系统联盟链在任意节点准确追溯到软件知识产权信息以及历史版本变更信息，不用依赖于第三方服务。根据版本管理模块的设计，软件产权管理提供查看所属的软件产权信息、管理软件产权版本等服务。

4.3.1　账本查询实现

软件知识产权信息查询服务实现过程如时序图 4.8所示。用户经过LDAP登陆验证并获得Fabric CA 的授权证书后，进入软件知识产权管理页面。系统会从状态管理中心Store 里获取用户的查询表单记录，如果记录不为空则把记录自动作为表单数据显示。用户可以填写查询表单，把表单数据作为查询请求参数，通过异步分发中心通知状态管理中心store，执行相应的异步行为，发起查询特定软件产权信息的网络请求。如果表单数据为空则发起获取用户所有软件产权信息的网络请求。

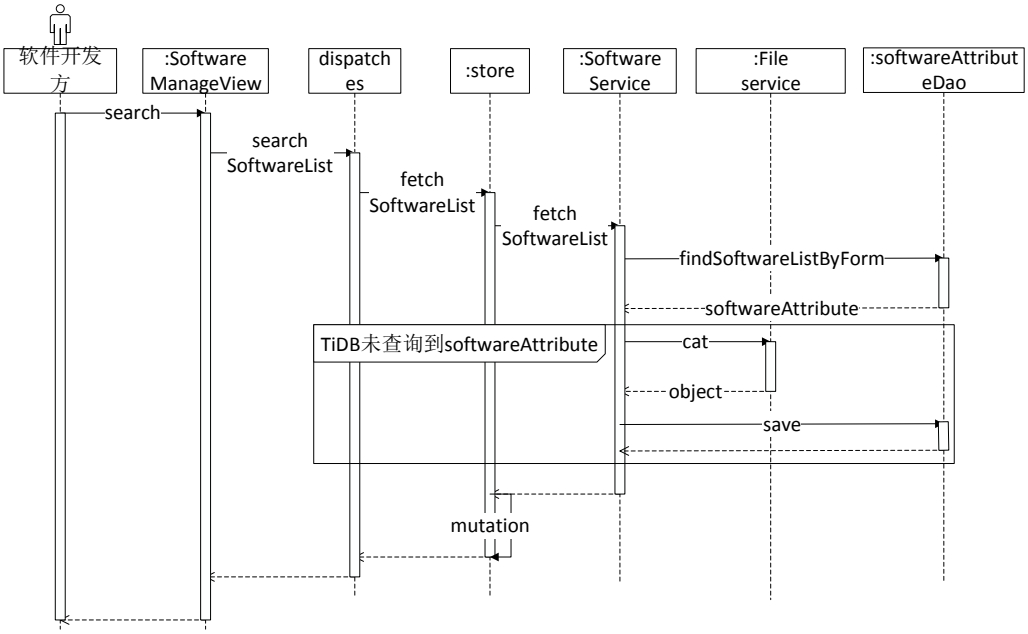


图 4.8: 产权信息查询时序图

在进入页面时，页面开始加载，页面初始化前会先对类的计算属性进行初始化，从store获取到searchForm 查询表单，并初始化获取已有软件产权列表方法和查询产权方法。之后页面开始初始化，触发其create生命周期钩子，进行查询表单的判断，是否表单为空，如果表单不为空就执行searchSoftwareList方法查询软件产权，如果为空执行getSoftwareList方法获取已有产权。页面通过dispatch向Store发起异步操作，调用相应的action方法，通过Axios实例向后端发起HTTP请求。

后端收到前端查询请求后，Logic层会调用相应的服务。SoftwareService类调用DocDao的方法，发起区块链数据查询请求。区块链服务中，DocService类执行query方法实现查询区块链账本。该方法的实现会调用FabricClient 类的queryBlockChain方法发起交易提案，执行节点上chaincode智能合约查询节点账本数据。区块链上数据会以字符串数组的形式返回，通过数据类型预处理，将链上数据转换成系统内的领域实体对象，以id为key的KVPair键值对对象，值为Software_Doc对象。

在获取到区块链上软件知识产权数据后，需要补全前端交互所需的业务属性。DocDao类在获取区块链服务返回的键值对数组后进行数据类型转换，转换成以id为key的map对象。SoftwareService类对该map对象进行遍历，对每个软件产权实例调用类私用方法getSoftwareAllData得到包含所有数据的产权实体。

```
private Software getSoftwareAllData(Doc doc) {
    Software software = new Software();
    //TiDB 获取缓存的属性数据
    SoftwareAttribute softwareAttribute = softwareAttributeDao.
    findBySoftwareHash(doc.getSoftwareAttrHash());
    if (softwareAttribute == null) {
        //没有缓存从 ipfs 获取并解析文件
        softwareAttribute = this.ipfsService.getObjectFromFileByHashcode
(SoftwareAttribute.class, doc.getSoftwareAttrHash());
        softwareAttribute.setSoftwareAttrHash(doc.getSoftwareAttrHash());
        //存入 TiDB 作为缓存
        softwareAttributeDao.save(softwareAttribute);
    }
    software = this.setAttributeToSoftware(softwareAttribute,doc);
    return software;
}
```

图 4.9: 获取产权详细信息实现代码

查询用户已有的所有软件产权信息的方法实现代码如图 4.9所示。通过软件产权属性hash查询TiDB是否有缓存的SoftwareAttribute 对象数据，鉴于区块链数据不可篡改的性质，TiDB上存储的这些软件产权数据不会出现变更。如果从TiDB 查询到了缓存数据，直接将业务属性数据和区块链上数据转换为软件产权对象。如果TiDB 没有缓存数据，后端服务通过产权属性hash去IPFS查询文件并解析成SoftwareAttribute对象，同时将数据存储到缓存数据库TiDB，方便下次查询。最终将所有数据提交到Logic 层进行格式转换并封装，返回给前端。前端收到请求返回的数据，Store更新相应的状态，并通过回调函数将数据返回给组件，组件响应数据变化以及Store全局状态的变更，局部渲染生成页面。

4.3.2　版本管理实现

软件这一数字化逻辑实体，具有使用周期较短的特性，需要持续变更，不断迭代。这使得软件知识产权也需要随时间变化进行认证信息更新。

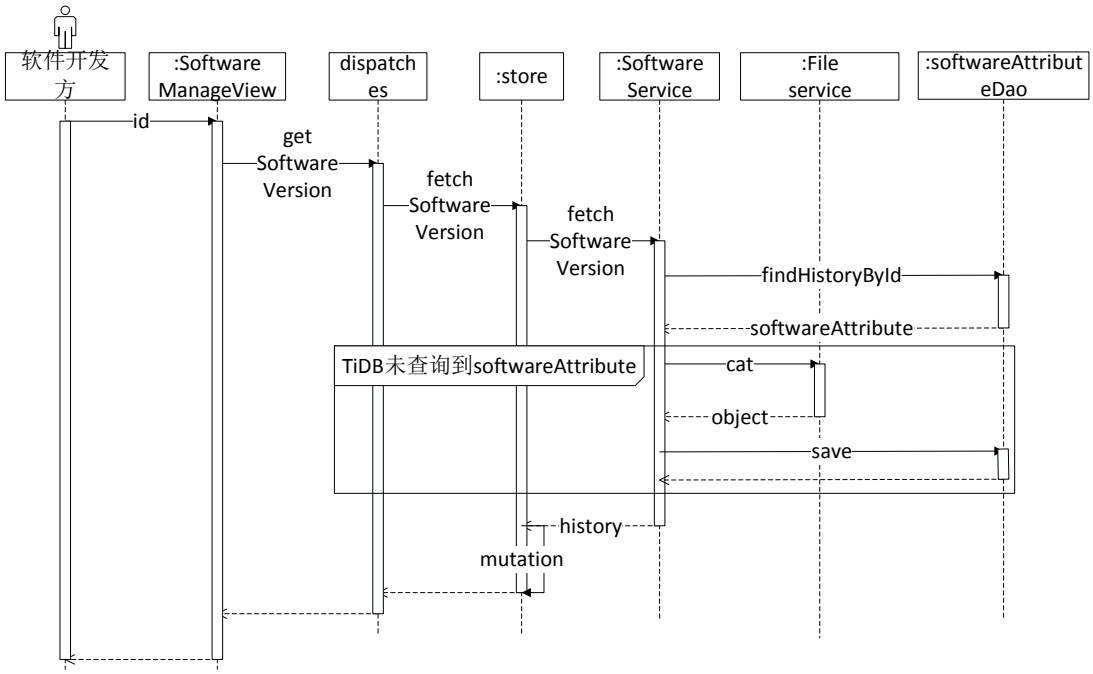


图 4.10: 获取产权历史版本时序图

软件知识产权历史版本查询服务执行过程如时序图 4.10所示。用户经过LDAP 登陆验证并获得Fabric CA 的授权证书后，进入软件知识产权管理页面。系统根据用户填写的查询表单查询已有的软件知识产权，展示获取的软件知识产权最新版本。用户查看某个软件产权的所有版本，把产权id 属性作为请求参数，通过dispatch分发请求通知状态管理中心Store。Store会触发相应

的action，向服务端请求获取区块链上该软件产权的所有历史版本。前端获取数据后，通过Accordion组件按时间顺序展示产权的历史变更过程，用户可以查看各版本详细信息。该过程中涉及到的主要类如表 4.2所示。

SoftwareManager是产权管理页面类，SimpleSearchForm是通用查询表单组件，Mysoftware是软件产权列表展示组件，SoftwareVersionAccordion是产权版本展示组件，Pagination是通用的分页组件。Store负责软件产权管理页面组件间的状态一致性管理。Dispatches处理用户操作请求，触发状态管理中心相应action。SoftwareController接收查询产权版本的HTTP 请求并返回系统设定的标准化对象ResponseResult。SoftwareService处理查询产权版本业务。IPFSService获取IPFS文件并解析为softwareAttribute对象。SoftwareAttributeDao负责查询TiDB缓存数据库。DocDao负责调用区块链服务模块提供的方法获取链上账本数据服务。

表 4.2: 软件知识产权版本管理相关类表

类名	分类	功能说明
SoftwareManager	组件类	软件产权管理页面
MySoftware	组件类	用户的已有软件知识产权列表组件
SoftwareVersionAccordion	组件类	产权版本手风琴组件
SimpleSearchForm	组件类	查询表单组件
Dispatches	异步操作类	分发界面异步请求，分离组件视图与业务
Store	状态管理类	中心化管理前端平台数据状态
SoftwareController	控制类	提供软件产权管理服务对外接口
SoftwareLogic	逻辑类	调用下层服务，完成软件知识产权数据管理相关业务逻辑处理
SoftwareService	服务类	实现软件产权信息模糊查询，历史版本管理等服务
IPFSService	服务类	提供IPFS文件查询和解析服务
SoftwareAttributeDao	数据访问类	查询TiDB缓存的数据
DocDao	数据访问类	调用区块链服务模块执行智能合约获取链上账本数据

软件产权历史版本展示通过SoftwareVersionAccordion组件实现。单文件组件软件产权版本管理“手风琴”SoftwareVersionAccordion采用ElementUI组件库提供的collapse折叠面板组件实现。利用prop属性softwareHistoricVersion循环渲染折叠面板列表，注入自定义的title模块作为面板的标题。引入SoftwareDetail组件，作为Accordion展开时的展示内容。组件向外提供可输入的softwareHistoricVersion属性，实现功能单一的可复用组件，组件内部与Store通信，通过上层的父页面向外emit事件，由管理页面调用dispatch 向Store通信发起软件产权版本数据请求。后端收到请求后service 会调用Dao层的getAllVersionDoc方法，向区块链服务发起数据查询，以软件id为参数查询所有的软件历史版本。

```
//从区块链上查询软件 history
@Override
public <T> List<History<T>> history(Type type, String key) throws
InvalidArgumentException, ProposalException {
    String result = hfClient.queryBlockChain(ChainCode.docs.getName(),
    Function.history.getName(),new String[]{key});
    return DocUtil.string2HistoryList(type, result);
}
```

图 4.11: 获取产权历史版本数据实现代码

区块链查询产权历史数据方法具体代码实现如图 4.11所示。区块链服务收到查询请求，调用区块链查询方法，选择查询历史版本的智能合约获取区块链账本数据，数据是以字符串数组的形式返回，调用DocUtil 工具类进行格式转换，转换成History对象数组，包含区块链交易ID、doc 对象数据、上链时间戳。上链时间戳就是软件产权的认证时间戳。区块链服务将History对象数组返回给service 层，service 层获取其中的doc对象，并通过getSoftwareAllData查询TiDB或IPFS得到完整的软件产权数据。经过格式处理和封装后返回给前端。前端平台接收到标准化格式响应数据后，获取其中版本列表数据，更新store中数据状态，自动同步到SoftwareVersionAccordion组件，完成异步渲染，将软件产权各版本信息以时间顺序展示给用户。

4.4　区块链服务模块

4.4.1　数据上链实现

不同于传统平台的数据存储方式，软件产权认证服务中数据存储有两个过程。首先是将如软件产权名称、描述、证书文件名、软件制品地址hash等非核心业务数据进行链外存储，存储到IPFS分布式存储网络生成内容相关的属性hash。然后将属性hash以及必要数据包括软件知识产权id、所属用户、证书hash等，通过智能合约存储到区块链上。

系统将区块链相关服务从业务服务层分离，实现区块链子服务模块向上层提供区块链操作服务，服务内部实现底层的数据存储实现。软件知识产权认证过程包括用户输入的业务数据存储到IPFS网络和认证信息通过智能合约存储到区块链账本。涉及到的主要类包括IPFSService，IPFSAPI，DocService，FabricClientService，FabricClientAPI。具体实现过程如时序图 4.12所示。

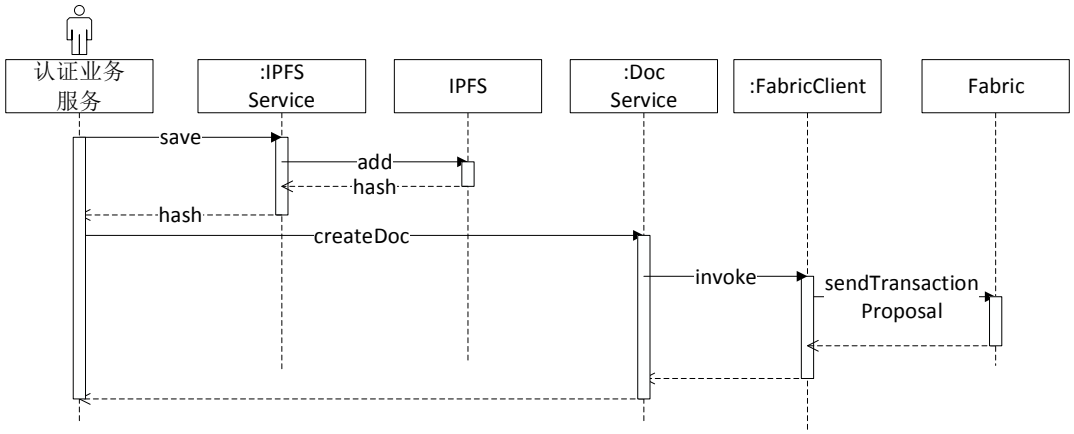


图 4.12: 产权认证信息上链时序图

IPFS对外提供了文件读写接口，软件知识产权的业务信息需要经过封装成文件才能存储。图 4.13 描述了IPFSService类的saveObject方法实现，将对象写入XML文件中并存储到IPFS。saveFile方法调用IPFS项目提供的API 实现上传文件到IPFS网络。IPFS 生成的hash会作为实体属性记录在区块链上。链上hash值无法篡改，IPFS的地址hash值与原文件内容唯一对应不会出现相同文件重复存储，因此文件被恶意篡改后，hash不会获取到被篡改的文件。

```
public String saveObject(Object object) {
    String hashcode = null;
    XmlMapper xmlMapper = new XmlMapper();
    String fileName = prefix + idService.genId() + ".xml";
    try {
        xmlMapper.writeValue(new File(fileName), object);
        File file = new File(fileName);
        hashcode = this.saveFile(file);
        file.delete();
    } //省略异常处理
    return hashcode;
}
```

图 4.13: IPFS对象存储实现代码

DocService类的saveSoftwareCert方法实现软件认证信息区块链上链逻辑，将继承Doc抽象类的Software实体转换成字符串数组，id设置成上链信息的key。然后调用invokeBlockChain 方法实现交易提案提交上链。图 4.14描述了执行智能合约完成交易上链的代码。FabricClient类的invokeBlockChain方法创建一个交

易提案TransactionProposalRequest，设置调用的chaincode智能合约的标识，选择调用的具体执行方法，将上链数据设置为args。从Fabric CA 获取经过身份验证的合法证书加入到channel通道，channel向Endorser 节点发起交易提案请求，Endorser节点完成对提案的背书，返回Endorser节点的处理结果，判断交易是否遵循智能合约条件。在多个Endorser 检查提案通过，不存在检查结果不通过的节点后，通过channel 正式提交交易，发送到提供共识服务的节点。

```
//FabricClient 类发起区块链 invoke 方法
public void invokeBlockChain(String chaincode, String function, String[] args) throws
ProposalException, InvalidArgumentException, InterruptedException,
ExecutionException, TimeoutException {
    //创建一个交易提案
    TransactionProposalRequest transactionProposalRequest = client.
    newTransactionProposalRequest();
ChaincodeID cid = ChaincodeID.newBuilder().setName(chaincode).build();
//配置 chaincode 的 id 以及调用的智能合约方法和参数
    transactionProposalRequest.setChaincodeID(cid);
    transactionProposalRequest.setFcn(function);
    transactionProposalRequest.setArgs(args);
//向背书节点发起交易提案
    Collection<ProposalResponse> invokeRes = channel.sendTransactionProposal
    (transactionProposalRequest);
    for (ProposalResponse res : invokeRes) {
        if(res.isInvalid()) {
            logger.error("transaction"+res.getTransactionID()+"proposal invalid from
peer"+ res.getPeer().getName());
            throw new RuntimeException("invalid response(s) found");
        }
}
//正式提交 Transaction
    channel.sendTransaction(responses);
}
```

图 4.14: 执行智能合约信息上链代码实现

4.4.2　节点账本查询实现

由于系统使用区块链结合IPFS的存储模式，用户查询软件知识产权认证历史数据需要经过两个过程，链上数据查询和IPFS文件查询。具体底层数据查询过程如时序图 4.15所示。DocService 类负责预处理参数，调用FabricClient类

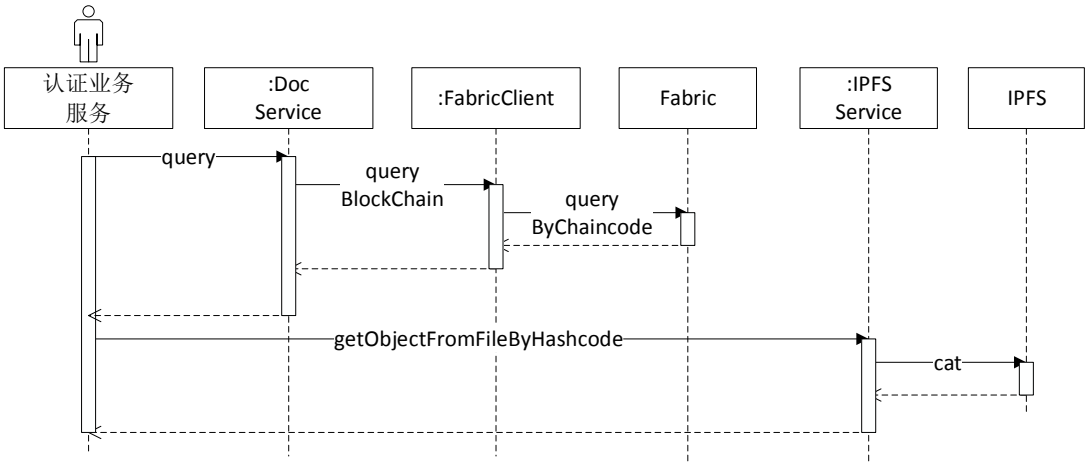


图 4.15: 区块链账本数据查询时序图

的queryBlockChain方法获取数据，然后处理成Doc对象传递给业务服务模块。FabricClient类创建channel通道发起提案查询节点账本。IPFSService负责以二进制流的方式从IPFS节点读取文件并解析成对象。

```
//调用 chaincode 的 query 方法
ChaincodeID ccId = ChaincodeID.newBuilder().setName(chaincode).build();
QueryByChaincodeRequest queryByChaincodeRequest
= client.newQueryProposalRequest();
queryByChaincodeRequest.setArgs(args);
queryByChaincodeRequest.setFcn(func);
queryByChaincodeRequest.setChaincodeID(ccId);
//通过 channel 通道发起 query 请求
Collection<ProposalResponse> queryRes = channel.queryByChaincode
(queryByChaincodeRequest);
for(ProposalResponse pr : queryRes) {
    if(!pr.isVerified() || pr.isInvalid()) {
        //省略输出报错 log
    }else {
        String payload = pr.getProposalResponse().
getResponse().getPayload().toStringUtf8();
        return payload;
    }
}
```

图 4.16: 执行智能合约查询账本数据代码

queryBlockChain方法通过chaincode智能合约查询区块链数据，具体实现如图 4.16所示。在用户登录时就已经在Fabric CA生成证书，后续的用户请求不需要再验证权限。查询过程首先封装查询请求，包括chaincode标识、调用的智能合约function、以及查询的参数，没有参数就会查询注册用户所有的软件产权。然后通过channel 通道发起query 请求，执行chaincode智能合约中的queryByChaincode方法，方法内部调用sendTransactionProposal，请求中可以设置多个目标背书节点，向所有目标背书节点发送交易提案，并提取出所有节点响应的结果组成一个集合返回。所有的目标节点都维护相同的数据账本，返回某个背书节点的有效响应结果作为数据查询结果。

查询到链上数据并对数据进行实体映射后，IPFS文件服务根据链上数据存储的hash查询IPFS网络获取软件产权业务信息。如图 4.17所示，以hash值从IPFS获取到文件字节流，通过xmlMapper 读取字节流并生成对象，对象的类型与参数type所表示的类型一致，这样实现了更加通用的解析方法，可以解析不同文件生成相应的对象。

```
@Override
public Object getObjectFromFileByHashcode(Type type, String hashcode) {
    byte[] fileBytes = this.getFileByHashcode(hashcode);
    XmlMapper xmlMapper = new XmlMapper();
    Object object = null;
    try {
        object = xmlMapper.readValue
(new ByteArrayInputStream(fileBytes), type.getClass());
        return object;
    } //省略异常处理
    return object;
}
```

图 4.17: 从IPFS获取数据对象实现代码

4.5　中心化状态管理模块

在前端平台中，多个组件由同一数据维护组件状态、导致视图变更时，实现中心化状态管理是个很好的解决方案。本系统利用VueX实现了平台的中心化状态管理，一方面能够维护组件间的状态一致性，记录所有的状态数据变化，增强代码可读性，使得开发人员能够快速定位异常数据变化。另一方面数据状态进行统一管理，减少了相同数据重复请求的次数，降低服务器压力。

图 4.18展示了系统软件产权实体状态管理Store的实现。Store必须要包括的三个部件，state，mutations，actions。state包含需要维护的软件相关状态数据，以用户的已有软件产权列表为例，作为系统的全局状态，用户只能查询状态数据不能直接对数据进行操作。mutations是对外提供的操作state 的接口，用户的已有软件产权发生改变时只能通过mutations 提供的方法setSoftwareList显式的触发改变列表数据。Actions负责异步操作，组件状态的变化一定是由异步事件引起，actions执行这些软件产权相关操作，如果有相关state 需要更新则向mutations提交commit执行变更。

```
//software store
const state: State = {
   softwareList: [],
   searchForm: Object.create(null),
   //省略属性
};
const mutations = {
   //省略 mutation 中的方法
};
const actions = {
   //省略 action 中的方法
};
const software = {
   state,
   mutations,
   actions,
   //省略配置项
};
```

图 4.18: 中心化状态管理Store实现

在实现了software的Store状态管理中心后，需要通过VueX插件将Store注入到Vue框架中，保证客户端能开始监听管理中心中的状态。Vuex以模块形式引入实现的softwareStore 并注入Vue实例。Vue框架在浏览器上初始化后，会对状态管理中心开启状态监听和数据管理。本系统还在Store与交互视图之间增加了一层dispatch，用于分发视图中发起的异步事件调用相关的actions，这样的实现分离了view视图和store，组件不需要知道有状态管理中心存在。

4.6　系统测试

依据上文中系统需求分析和实现描述，本小节主要描述软件产权认证系统的测试，包括单元测试、集成测试、联盟链性能测试，以及运行实例展示。

4.6.1　系统测试目标

通过系统测试，在Fabric节点数量少的情况下，检测系统三个核心模块的实现能否满足系统需求，验证区块链解决方案的可行性。验证软件开发方的软件知识产权通过系统是否能够安全的存储到区块链上。验证用户能否按照特定条件从区块链节点账本获取软件知识产权的认证信息和产权历史版本信息。

4.6.2　系统测试环境

系统测试前需要部署一个测试环境，根据系统的物理视图，服务端需要部署包括前端Web平台、后端产权认证服务子模块、区块链服务子模块、LDAP、TiDB集群节点、Fabric集群、IPFS 节点。客户端使用chrome浏览器运行Web平台，同时测试设备需要下载证书工具使用证书服务。服务端测试环境部署在腾讯云上，后端业务服务和区块链服务都是以Docker容器化部署运行。前端web平台和后端业务服务之间通过Ngnix实现跨域访问与反向代理，Ngnix还可以提供负载均衡功能。服务器部署LDAP管理节点测试用户，提供测试用户信息。服务器部署TiDB集群作为缓存数据库。同时服务器还需要部署IPFS，加入IPFS文件存储网络成为节点，默认存储空间10个G。Hyperledger Fabric集群分别部署在腾讯云和阿里云，阿里云上部署了一个共识节点，腾讯云上使用Docker部署一个共识节点和一个包含单个背书节点的org，节点角色的区分只是相对测试环境来说，节点可以扮演Fabric联盟链网络中的多种角色。系统具体测试环境配置信息如表 4.3所示。

表 4.3: 系统测试环境

环境项	测试环境
客户端	Chrome浏览器
云服务器	腾讯云，阿里云
容器	Docker
操作系统	Ubuntu16.04
节点数量	3
Hyperledger Fabric版本	1.3.0
IPFS版本	1.2.2
其他软件	TiDB，Nginx，LDAP，Leaf

4.6.3　单元测试

系统服务端用java语言开发。单元测试是对本系统中的最小测试单元，即java类，进行检查和验证。利用Junit对其中重要的业务逻辑类进行测试。

表 4.4: 单元测试用例表

测试用例	说明
testCreateSoftware	测试创建软件知识产权认证信息方法
testCreateSoftwareWithEmptyParams	传入异常参数创建软件知识产权认证信息
testCheckWithTamperedCertificateHash	更换证书，测试认证信息校验
testCheckWithTamperedSoftwareHash	篡改软件制品hash值，测试认证信息校验
testCheckWithTamperedUserSignature	篡改证书中用户数字签名，测试认证信息校验
testQueryAllSoftware	测试查询用户已有的所有软件知识产权方法
testQuerySoftwareList	测试根据表单查询特定软件知识产权
testQuerySoftwareListWithEmptyParams	传入空表单测试查询软件知识产权
testQuerySoftwareHistory	测试获取软件产权历史变更方法
testQuerySoftwareHistoryWithEmptyId	传入空id测试查询软件产权历史变更
testSaveToBlockChain	测试区块链数据上链服务
testGetHistoryFormBlockChain	测试从链上账本获取软件产权历史服务
testQueryFormBlockChain	测试区块链账本查询服务

单元测试包含的用例如表 4.4所示。主要测试软件产权认证方法、用户数字签名验证过程、产权认证历史版本信息查询方法和区块链服务提供的服务。对于软件知识产权认证过程，分别传入正常参数和异常参数测试认证服务能否正确校验参数并返回预期结果。对于认证过程中用户数字签名校验过程，通过更换证书、篡改软件加密hash、篡改用户数字签名，测试认证服务的校验方法是否能校验出签名异常。对于软件知识产权查询功能，测试能否获取用户所有软件知识产权并能根据查询表单进行模糊查询。对于软件知识产权历史版本数据

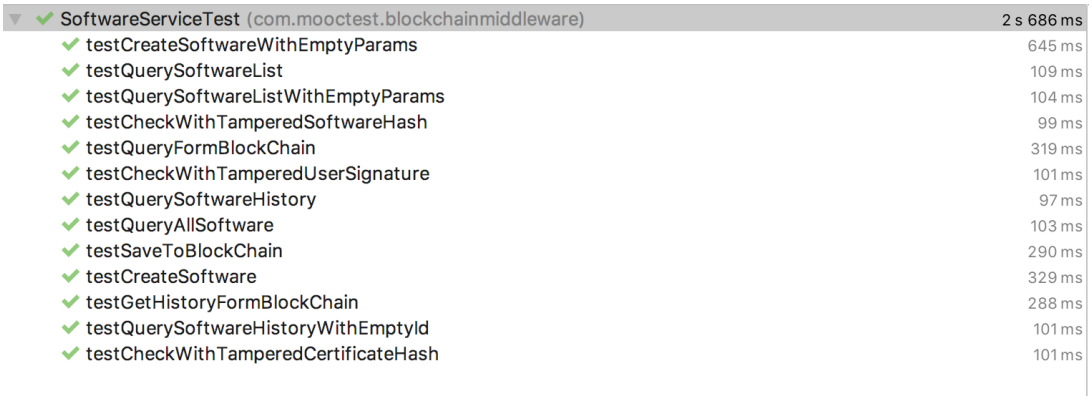


图 4.19: 单元测试结果展示

管理，测试获取软件知识产权历史数据的方法，当传入异常参数时，方法能否抛出异常。对于区块链服务，测试区块链数据上链的方法和从链上账本查询数据的方法，检测是否执行相应智能合约并发起共识返回执行结果。按照设定的参数进行测试，单元测试输出结果与预期一致，如图 4.19所示。单元测试结果表明系统核心业务类以及区块链服务类都能提供良好的服务。

4.6.4　集成测试

集成测试是基于单元测试的基础上，将系统的各个组成模块按照系统设计的要求组成可运行系统进行测试。根据系统的设计与实现，系统分为三个模块，证书工具模块、产权认证模块、版本管理模块。集成测试针对这些业务模块设计用例，确保集成的系统能满足系统需求分析的预期目标。模拟用户行为，按照特定条件下的预期步骤操作，判断系统输出结果是否与预期输出结果一致。

表 4.5: 证书工具模块测试用例

测试功能	步骤	预期结果	测试结果
LDAP登录	1.输入用户名和密码 2.点击验证	登陆成功，证书工具获取用户私钥	通过
软件证书生成	1.输入用户名 2.选择加密算法 3.选择文件路径、软件文件路径、文件生成目标路径	XML格式证书生成到指定路径，界面状态显示生成成功	通过
查看软件证书内容是否正确	1.打开已生成的证书文件 2.查看文件是否包含所需内容	证书包含时间戳，软件制品hash，加密算法，用户数字签名，用户信息。	通过
证书验证软件真实性	1.选择待验证软件 2.选择对比的软件证书	验证通过	通过
证书验证软件被篡改	1.选择内容被篡改的软件 2.选择对比的软件证书	软件和证书不一致	通过

证书工具模块主要是负责软件和唯一证书之间的转换，能将软件制品加密附带用户签名和时间戳生成XML格式的证书文件，同时能使用证书验证相应软件是否被篡改。该模块的测试主要包括测试能否加密软件生成证书，验证生成的证书是否包含软件hash、用户数字签名、时间戳，验证证书能否确定软件的真实性。如表 4.5所示，测试结果与预期一致。结果表明证书工具可以实现加密软件并附带用户数字签名，时间戳等信息生成预期的软件证书，并且能够验证软件证书和软件制品是否一致。

表 4.6: 软件认证模块测试用例

测试功能	步骤	预期结果	测试结果
软件知识产权认证	1.登陆系统平台获取区块链访问权限 2.认证界面输入软件名和描述 3. 上传软件证书 4.上传软件制品提交认证	认证成功，节点发起交易提案认证信息上链。	通过
验证用户数字签名	1.篡改软件证书的软件加密hash 2.上传被篡改的证书进行产权认证 3.提交认证表单	用户数字签名验证不通过，界面展示证书验证失败，拒绝写操作	通过
更新软件产权版本	1.选择认证类型为更新已有软件产权 2.选择待更新的软件产权 3.填写新信息表单并提交	软件产权新版本认证信息上链，节点发起交易提案。	通过

产权认证模块主要是负责将软件知识产权和所属方的信息记录到区块链上，保证认证信息的安全存储、不可篡改、可追溯、联盟链所有节点共享。该模块的测试主要包含两个部分，软件产权认证信息能否上链，产权版本变更新信息能否上链。其中包括对证书中用户数字签名的验证。经过模拟用户在平台上操作完成测试，测试结果与预期一致，如表 4.6所示。认证信息被记录到区块链，不可篡改可追溯。测试结果表明软件知识产权认证系统能够很好地提供软件知识产权认证服务，通过对证书中用户数字签名的验证保证认证信息的真实性，调用区块链服务执行智能合约完成认证信息上链存储。

表 4.7: 软件管理模块测试用例

测试功能	步骤	预期结果	测试结果
查看用户已有软件产权功能	1.登陆并授权允许访问节点账本 2.进入软件产权管理界面	分页展示用户认证过的所有软件产权。	通过
查询特定的软件产权功能	1.填写查询条件表单 2.提交查询请求	展示满足查询表单条件的软件产权。	通过
产权信息缓存	再次点击表单查询	没再次查询IPFS解析文件。	通过
获取某一软件产权随时间迭代的所有版本	1.选择某一软件产权 2.展开表格下来项 3.查看版本详细信息	按时间顺序展示该软件产权所有的迭代版本和版本的详细信息	通过
产权链上交易记录查询	1.选择需要查询账本交易提案详情的软件产权 2.点击访问区块	界面显示授权成功，更新授权订单的信息。	通过

版本管理模块主要是负责从区块链上获取用户软件产权认证信息以及软件知识产权随时间变更的版本管理。该模块的测试主要是验证授权用户能否从区块链上获取已有的软件产权信息，验证软件产权版本变更信息是否存储到区块链上，验证是否可按时间追溯到软件产权历史版本。如表 4.7所示。测试结果与预期一致，从区块链上查询的数据与认证过的产权数据集相同。

在测试过程中，使用了Robot Framework实现Web平台的自动化测试，测试结果如图 4.20所示。各业务模块的集成测试结果表明本系统能够给用户提供良好的软件知识产权认证服务，并将所有信息通过执行智能合约提交到联盟链网络进行全网共识，上链存储。

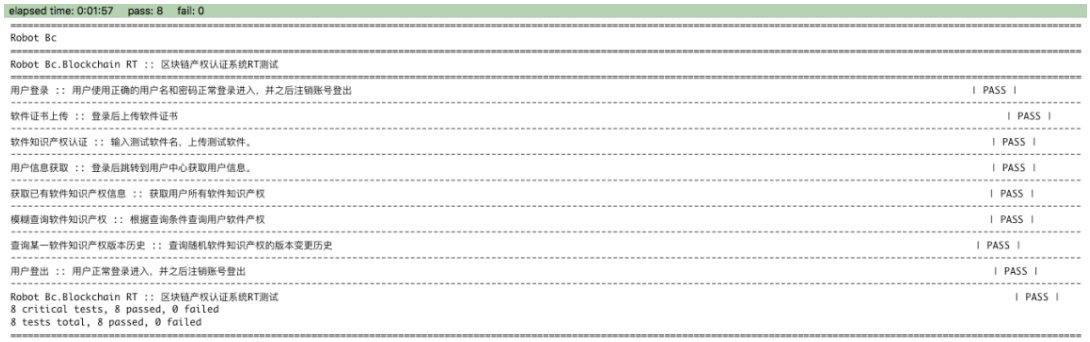


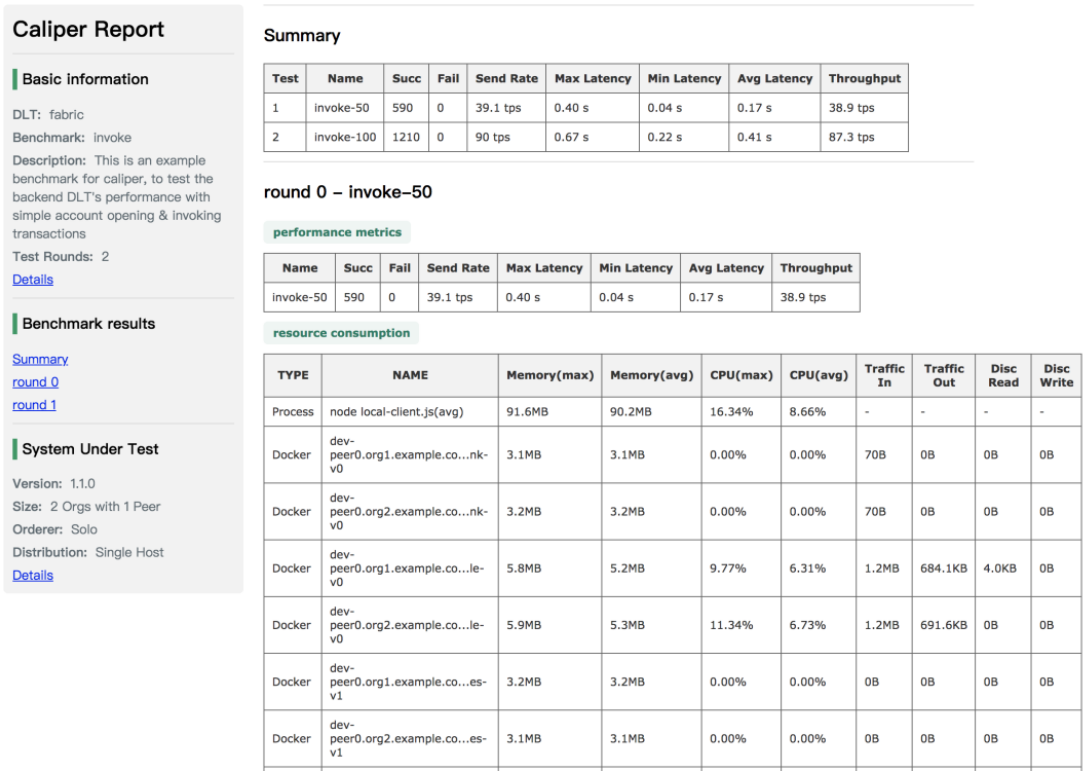
图 4.20: 集成测试结果展示

4.6.5　区块链性能测试

区块链性能测试是在测试环境下，通过自动化测试工具模拟正常上链请求，针对本系统联盟链的各项性能指标进行测试，主要包括吞吐量和共识延迟。考虑到系统联盟链是基于Hyperledger Fabric 搭建，区块链性能测试采用Caliper 开源测试工具进行模拟测试。Caliper能够按照设定的吞吐速率执行chaincode智能合约，向测试环境的背书节点发起交易提案，检测交易提案执行过程和共识过程中的一系列性能数据，以及每个节点的资源消耗。

本小节通过Caliper 执行认证信息上链智能合约进行测试。为了测试不同交易频率下区块链吞吐量和延迟，通过设定fixed-rate配置分别以50tps和100tps的速率执行智能合约。Caliper 执行每轮实验的测试结果包括交易提案成功与失败数量，交易发起速率以及区块链吞吐量，平均延迟以及最大最小延迟，每个节点的性能损耗等。测试结果如图 4.21 所示。第一次测试，Caliper 以39.1tps的速度提交590次上链交易，联盟链的吞吐速率几乎与发起速度一致，平均延迟在0.2秒以内。第二次测试，在90tps 的交易频率下，系统联盟链仍可以维持良

好的响应速度，保证最大共识延迟在1秒以内。测试结果表明，本系统联盟链在交易吞吐量过大的情况下仍能提供良好的可用性，满足系统当前的业务场景。



Caliper Report

Basic information

DLT: fabric
Benchmark: invoke
Description: This is an example benchmark for caliper, to test the backend DLT's performance with simple account opening & invoking transactions
Test Rounds: 2
Details

Benchmark results

Summary
round 0
round 1

System Under Test

Version: 1.1.0
Size: 2 Orgs with 1 Peer
Orderer: Solo
Distribution: Single Host
Details

Summary

Test	Name	Succ	Fail	Send Rate	Max Latency	Min Latency	Avg Latency	Throughput
1	invoke-50	590	0	39.1 tps	0.40 s	0.04 s	0.17 s	38.9 tps
2	invoke-100	1210	0	90 tps	0.67 s	0.22 s	0.41 s	87.3 tps

round 0 – invoke–50

performance metrics

Name	Succ	Fail	Send Rate	Max Latency	Min Latency	Avg Latency	Throughput
invoke-50	590	0	39.1 tps	0.40 s	0.04 s	0.17 s	38.9 tps

resource consumption

TYPE	NAME	Memory(max)	Memory(avg)	CPU(max)	CPU(avg)	Traffic In	Traffic Out	Disc Read	Disc Write
Process	node local-client.js(avg)	91.6MB	90.2MB	16.34%	8.66%	-	-	-	-
Docker	dev-peer0.org1.example.co...nk-v0	3.1MB	3.1MB	0.00%	0.00%	70B	0B	0B	0B
Docker	dev-peer0.org2.example.co...nk-v0	3.2MB	3.2MB	0.00%	0.00%	70B	0B	0B	0B
Docker	dev-peer0.org1.example.co...le-v0	5.8MB	5.2MB	9.77%	6.31%	1.2MB	684.1KB	4.0KB	0B
Docker	dev-peer0.org2.example.co...le-v0	5.9MB	5.3MB	11.34%	6.73%	1.2MB	691.6KB	0B	0B
Docker	dev-peer0.org1.example.co...es-v1	3.2MB	3.2MB	0.00%	0.00%	0B	0B	0B	0B
Docker	dev-peer0.org2.example.co...es-v1	3.1MB	3.1MB	0.00%	0.00%	0B	0B	0B	0B

图 4.21: Caliper测试报告

4.6.6　系统运行展示

本系统基于Hyperledger Fabric搭建联盟链网络，实现了软件知识产权的可靠认证和数据管理，保证了认证信息由多方共同维护、可快速追溯、不可篡改，解决了个人或利益相关群体篡改伪造产权信息的潜在隐患，以及由第三方机构提供产权保护服务带来的效率与成本问题。

系统主要提供三部分交互界面。第一部分界面是软件知识产权认证服务界面,提供授权用户产权认证服务。在授权加入联盟链成为参与节点，或通过系统联盟链节点客户端注册成为授权用户后，用户可以通过浏览器访问系统Web服务平台。在经过LDAP 和Fabric CA 验证身份信息，生成授权证书允许访问区块链后，用户使用平台提供的软件知识产权认证服务，如图 4.22 所示。用户输入软件名，填写软件相关描述，上传由系统证书工具生成的软件证书文件到IPFS星际文件系统进行链外存储，根据软件真实情况选择性上传软件真实制品，然后点击“确认提交”按钮发起认证信息上链请求。后端服务从LDAP 获

取用户公钥对证书中用户数字签名进行解密验证。证书校验通过后，执行智能合约发起交易提案完成认证信息上链存储。通过区块链账本数据不可篡改的特性，认证信息上链后保证了用户的产权所有权不能被恶意篡改。

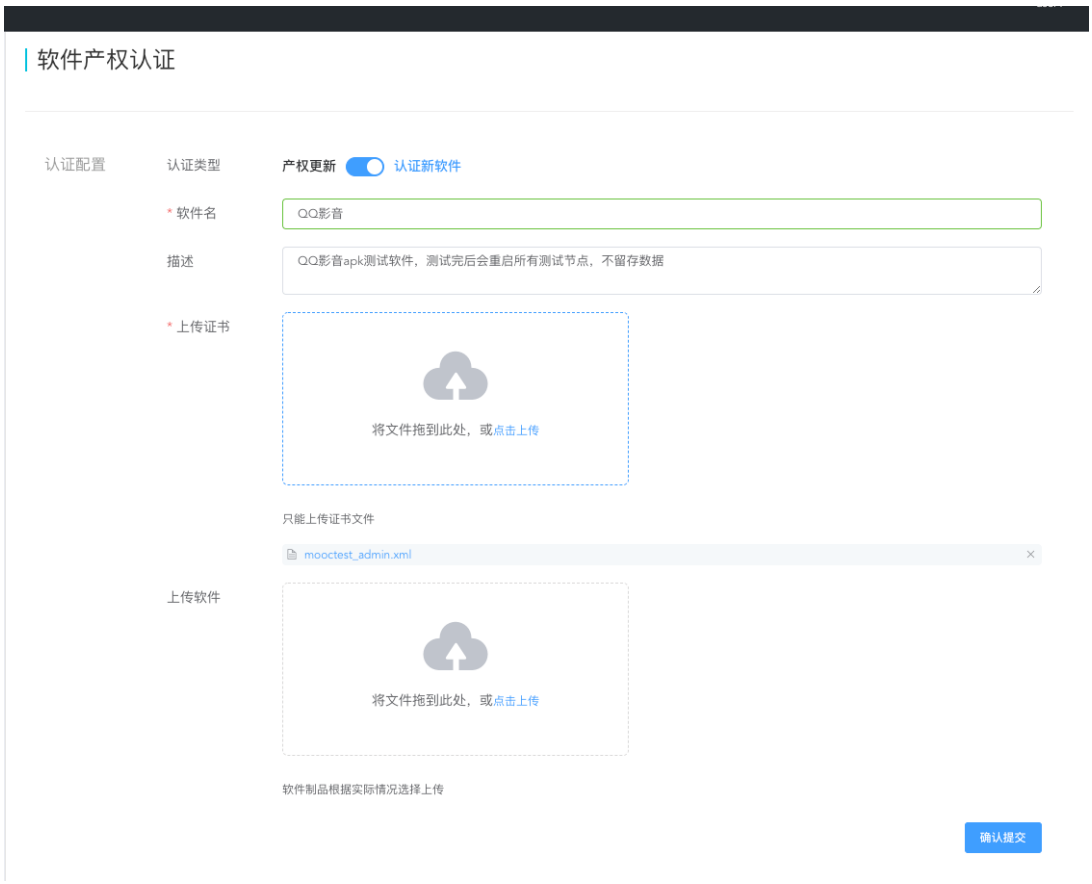


图 4.22: 软件产权认证服务界面

由于软件使用周期短，需要不断迭代更新的特性，软件知识产权认证信息需要随着软件版本迭代进行更新。用户在产权认证服务界面切换服务类型开关，选择更新产权版本服务并选择需要更新的软件产权。新的版本会自动保留原有产权的大部分信息。用户可以修改软件名，填写新版本的软件信息相关描述，上传新的软件证书和软件制品。考虑到区块链链上数据不可篡改，服务端通过生成唯一标识，将唯一标识作为认证信息上链交易提案的key通过智能合约记录到账本中，以此来追溯同一软件不同版本。

第二部分界面是用户产权管理界面。对于已认证的软件知识产权，用户可以通过产权管理界面获取区块链账本记录的详细信息。系统支持根据表单模糊查询特定的用户软件知识产权信息，查询表单包含软件id，时间和关键key。

如图 4.23所示。从区块链账本以及IPFS中获取的软件知识产权认证信息包括软件id，软件产权最新更新版本的上链时间，证书的IPFS地址hash，产权授权订单的标识，区块链交易id 等。用户可以点击“下载证书”按钮从IPFS 下载软件证书，点击“查看订单”按钮查看产权授权的详细情况。此外，用户通过触发列表行下拉展开事件，查看该软件产权在区块链账本中存储的所有历史信息和版本变更。界面通过“Accordion”方式按时间顺序展示产权历史变更，用户可以查看每个历史版本的详细信息，同时能够通过区块链交易id 查询该版本产权认证信息对应的区块链账本记录详情。

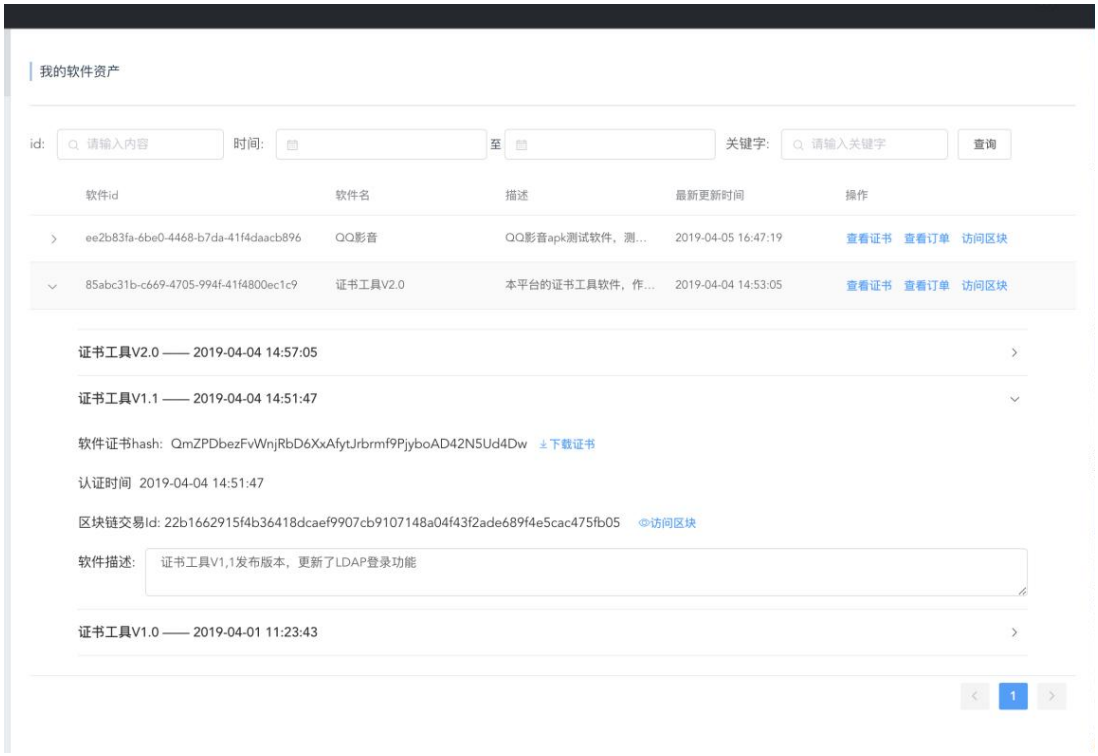


图 4.23: 产权版本管理界面

系统最后提供了区块链管理界面，用来验证产权认证信息上链结果。区块链管理界面提供联盟链信息查询服务,包括查看联盟链基本信息、所属节点信息、区块信息、交易详情、节点智能合约信息等。用户通过交易id查询区块链账本中该交易的详细信息，验证产权认证信息是否正确上链存储。界面查询结果列表显示交易的发起节点，channel标识名，交易的id，交易请求头的类型，执行的chaincode 智能合约，交易上链时间戳。用户点击列表中交易记录，其详情内容以Dialog 框的形式展示在界面上，如图 4.24 所示。Dialog 内容框包括整个交易提交过程的详细信息，包括参与交易提案“背书”过程的节点以及“背

书”验证有效性code。其中，writes属性以键值对的形式展示了交易提案的详细信息。以软件产权实体id为key，value为详细的认证信息。用户通过对比value存储的认证信息和产权管理页面的相应产权信息是否一致，验证所属的软件产权认证过程是否正确完成，认证信息是否准确写入账本。

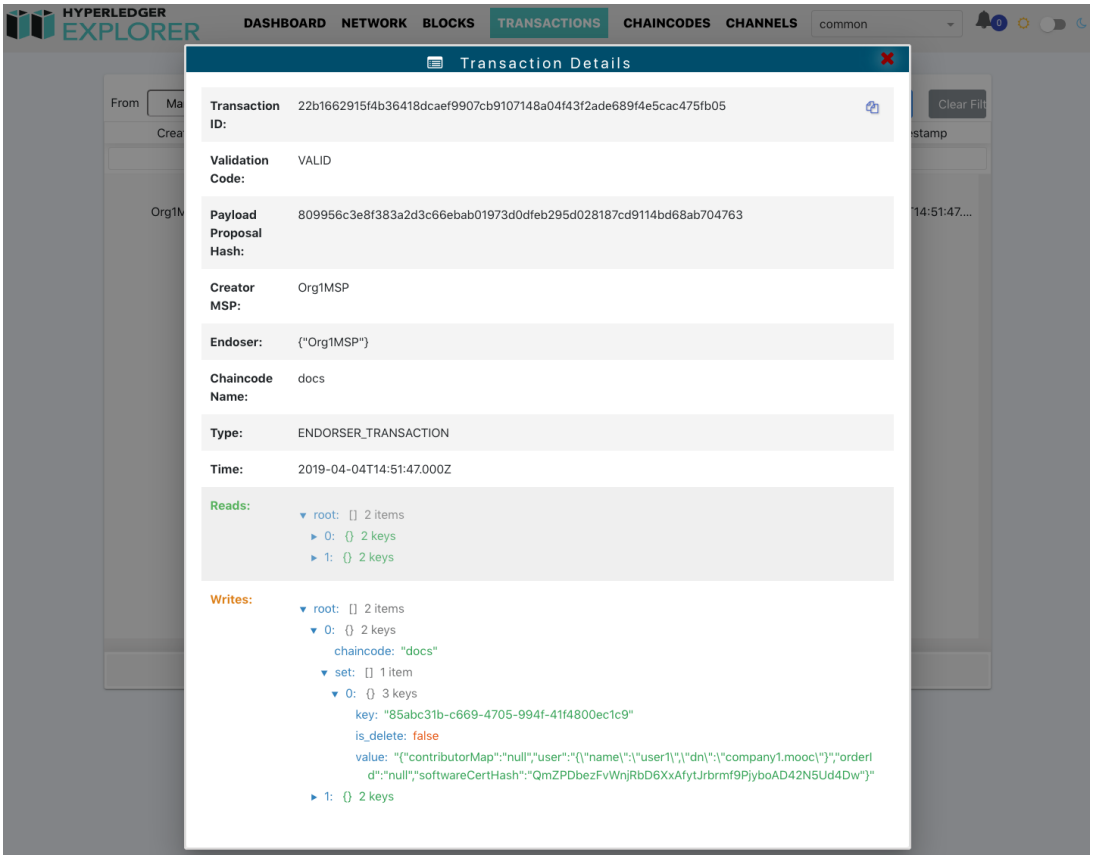


图 4.24: 账本交易详情界面

4.7　本章小结

本章主要描述了软件知识产权认证系统主要模块的实现，包括证书工具模块、产权认证模块、版本管理模块、中心化管理模块、区块链服务模块。证书工具模块实现了证书生成与验证，通过软件加密hash、用户数字签名、时间戳等信息生成软件证书，利用证书在系统内进行产权认证，有利于保护开发方的软件隐私。产权认证模块实现了软件知识产权认证服务，对用户提交的信息进行校验，对上传的证书进行解析并验证用户数字签名，验证通过后再发起上链存储请求，保证认证信息的安全性。中心化管理模块实现了前端平台的视图与数据状态分离，统一管理数据状态，保证前端平台良好的扩展性和状态一致性。

版本管理模块实现了产权版本管理，用户能从区块链账本中获取到软件产权的认证信息和历史版本信息。区块链服务实现了IPFS结合区块链节点账本的信息存储服务，通过IPFS链外存储业务信息，执行智能合约发起区块链交易提案实现认证信息上链和从节点账本获取数据。最后描述了系统的测试过程和运行结果，测试包括核心类的单元测试，系统的集成测试和联盟链的性能测试。

第五章　总结和展望

5.1　总结

互联网时代让软件行业蓬勃发展，各个领域的软件应用纷纷出现在市场上。在各个行业实现软件化的趋势下，软件传播迅速以及成果易复制的特点使得软件知识产权侵权盗版现象频繁出现。如何保证软件知识产权在互联网上进行安全可靠的所有权证明与信息追溯成为亟待解决的问题。本文认为去中心、不可篡改、可追溯的区块链为这一问题提供了新的解决思路。

本文基于区块链技术实现了软件知识产权认证系统。应用区块链技术去中心化、不可篡改、可追溯、分布共享的特性，解决依赖中心机构的应用系统中存在的存证、信息追溯、效率和信任问题，实现软件知识产权的可靠认证和安全存证。产权信息由联盟链网络参与者共同维护，不可篡改。系统主要涉及的开源技术有Hyperledger Fabric、IPFS 星际文件系统、Vue 框架、Spring Boot 框架等。系统主要实现以下几个部分。

系统实现了软件证书生成工具，将软件制品与软件产权解耦，把软件加密哈希附带用户数字签名和时间戳生成软件证书。目的是为了保护软件开发方的软件隐私以及维护软件类型的扩展性。系统实现了前端交互平台，设计了基于Vuex的平台前端中心化状态管理体系，将平台前端的数据状态和视图解耦，有利于在业务扩张下保持平台的可维护性。系统实现了软件知识产权的认证和版本管理服务。软件认证过程中，系统先从IPFS获取证书文件，对证书内的数字签名进行解密验证，确保认证信息准确无误后，调用区块链服务执行智能合约进行产权信息安全存证。对于软件知识产权版本变更，系统能利用区块链技术的可追溯性准确追踪软件知识产权所有的历史版本信息。系统实现了基于区块链结合IPFS的数据存储模式，将非核心信息存储到IPFS，区块链只存储少量核心数据和hash 值，这是目前解决区块链数据膨胀问题的有效解决方案。同时系统为了提高数据的查询效率，快速响应用户操作，通过TiDB 对IPFS 文件数据进行缓存，节省了IPFS 文件查询解析过程。

本系统目前已经在公司部署。结合阿里云和腾讯云上部署的共识节点，对系统进行单元测试，集成测试，区块链性能测试。测试结果表明基于联盟链的软件知识产权认证系统能够提供可靠的软件知识产权认证与存证服务。同时在100tps的交易负载情况下，系统联盟链仍能维持良好的可用性。通过本系统，用户能够不依赖于第三方服务进行软件知识产权的认证与存证。认证信息不可

被恶意篡改，由联盟链参与节点共享。最终，用户能在维权时在任意节点快速获取完整的产权信息，提高维权效率，降低维权成本。

5.2 展望

本系统已经可以稳定上线运行，但联盟链节点数量较少。系统将进一步邀请更多相关软件管理机构作为验证者节点参与到联盟链当中，并授权可信任的软件开发方作为参与者加入到联盟链，提高联盟链的稳定性、安全性和区块链账本记录在维权时的法律效力。

在业务方面，系统平台将进一步应用到软件众包领域，实现对软件产权分配和众包协同开发。目前软件产权的所属方是参与节点以公司为单位授权的用户，实际开发中，会出现越来越多众包协同开发的情况。结合相关众包技术，平台将进一步实现软件知识产权依据开发者贡献进行产权分配，依靠区块链技术记录各个开发方的产权占比，通过智能合约实现产权变现后按自动分配，防止众包开发者的权益受到侵害。同时平台业务广度将向软件产权交易变现方向延伸，促进认证的软件产权在平台实现快速变现，打造去中心化共同维护的一站式软件知识产权服务平台。

在性能方面，系统将使用更加高效的区块链共识算法，来降低节点数量增长后共识过程的延迟。同时优化系统存储策略和缓存策略，维持系统良好的可用性和可扩展性。

参考文献

[1] D. B.Marron, D. G. Steel, Which countries protect intellectual property? the case of software piracy, Economic Inquiry 38 (2) (2010) 159–174.

[2] W. Tsai, L. Feng, H. Zhang, Y. You, L. Wang, Y. Zhong, Intellectual-property blockchain-based protection model for microfilms, in: 2017 IEEE Symposium on Service-Oriented System Engineering, SOSE 2017, San Francisco, CA, USA, April 6-9, 2017, 2017, pp. 174–178.

[3] 中国信息通信研究院, 区块链白皮书（2018）, 中国信息通信研究院和可信区块链推进计划, 2018.

[4] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system.

[5] Y. Kano, T. Nakajima, A novel approach to solve a mining work centralization problem in blockchain technologies, Int. J. Pervasive Computing and Communications 14 (1) (2018) 15–32.

[6] A. Eross, F. Mcgroarty, A. Urquhart, S. Wolfe, The intraday dynamics of bitcoin, Social Science Electronic Publishing.

[7] A. Narayanan, J. Bonneau, E. W. Felten, A. Miller, S. Goldfeder, Bitcoin and Cryptocurrency Technologies - A Comprehensive Introduction, Princeton University Press, 2016.

[8] N. Szabo, Formalizing and securing relationships on public networks, First Monday 2 (9).

[9] G. W. Peters, E. Panayi, Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money, CoRR abs/1511.05740.

[10] 欧阳丽炜, 王帅, 袁勇, 倪晓春, 王飞跃, 区块链智能合约的发展现状:架构、应用与发展趋势, 自动化学报(2019) 1–13.

[11] P. Zhang, M. A. Walker, J. White, D. C. Schmidt, G. Lenz, Metrics for assessing blockchain-based healthcare decentralized apps, in: 19th IEEE International Conference on e-Health Networking, Applications and Services, Healthcom 2017, Dalian, China, October 12-15, 2017, 2017, pp. 1–4.

[12] K. Wüst, A. Gervais, Do you need a blockchain?, in: Crypto Valley Conference on Blockchain Technology, CVCBT 2018, Zug, Switzerland, June 20-22, 2018, 2018, pp. 45–54.

[13] PANG, Mao-hua, Shou-shan, WANG, Yang, Full privacy preserving electronic voting scheme, Journal of China Universities of Posts and Telecommunications 19 (4) (2012) 86–93.

[14] X. Sun, R. Torres, S. G. Rao, On the feasibility of exploiting P2P systems to launch ddos attacks, Peer-to-Peer Networking and Applications 3 (1) (2010) 36–51.

[15] J. R. Douceur, The sybil attack, in: Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers, 2002, pp. 251–260.

[16] S. Goldfeder, J. Bonneau, R. Gennaro, A. Narayanan, Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin, in: Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers, 2017, pp. 321–339.

[17] K. Bojana, K. Elena, M. Anastas, Blockchain implementation quality challenges: A literature review, in: Sixth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, 2017.

[18] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, G. Danezis, Consensus in the age of blockchains, CoRR abs/1711.03936.

[19] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, E. W. Felten, Sok: Research perspectives and challenges for bitcoin and cryptocurrencies, in: 2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015, 2015, pp. 104–121.

[20] G.Wood, Ethereum: a secure decentralised generalised transaction ledger, Ethereum Project Yellow Paper.

[21] E. Androulaki, A. Barger, V. Bortnikov, et al, Hyperledger fabric: a distributed operating system for permissioned blockchains, in: Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018, 2018, pp. 30:1–30:15.

[22] M. Li, J. Weng, A. Yang, W. Lu, Crowdbc: A blockchain-based decentralized framework for crowdsourcing, IACR Cryptology ePrint Archive 2017 (2017) 444.

[23] D. Ongaro, J. K. Ousterhout, In search of an understandable consensus algorithm, in: 2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014., 2014, pp. 305–319.

[24] A. Baliga, I. Subhod, P. Kamat, S. Chatterjee, Performance evaluation of the quorum blockchain platform, CoRR abs/1809.03421.

[25] M. Castro, B. Liskov, Practical byzantine fault tolerance, in: Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999, 1999, pp. 173–186.

[26] G. Psaila, Virtual DOM: an efficient virtual memory representation for large XML documents, in: 19th International Workshop on Database and Expert Systems Applications (DEXA 2008), 1-5 September 2008, Turin, Italy, 2008, pp. 233–237.

[27] C. Esposito, A. Castiglione, K. R. Choo, Challenges in delivering software in the cloud as microservices, IEEE Cloud Computing 3 (5) (2016) 10–14.

[28] C. Xia, Y. Zhang, L. Wang, S. Coleman, Y. Liu, Microservice-based cloud robotics system for intelligent space, Robotics and Autonomous Systems 110 (2018) 139–150.

[29] L. Yonggang, Z. Jinbiao, G. Libing, W. Yi, Y. Ruihai, Research on distributed network communication based on ice middleware, in: Proceedings of 2016 6th International Conference on Machinery,Materials,Environment,Biotechnology and Computer(MMEBC 2016), 2016, p. 5.

[30] J. Carnell, Spring Microservices in Action, Greenwich: Manning Publications, 2017.

[31] 黄凯, 孟庆永, 谢雨来, 冯丹, 秦磊华, 基于Docker swarm集群的动态加权调度策略, 计算机应用38 (05) (2018) 1399–1403.

[32] V. M. Gracia, R. Tolosana-Calasanz, J. Á. Bañares, U. Arronategui, O. F. Rana, Characterising resource management performance in kubernetes, Computers & Electrical Engineering 68 (2018) 286–297.

[33] 翟永超, Spring clond微服务实战, 电子工业出版社, 2017.

[34] S. Hatma, J. D. Puji, T. Aris, Design and development of backend application for public complaint systems using microservice spring boot, Procedia Computer Science 124 (2017) 736–743.

[35] E. A. Samir, R. Naoufal, Compactrio based real time implementation of aes algorithm for embedded applications, International Journal of Embedded and Real-Time Communication Systems 10 (2) (2019) 19–36.

[36] N. Nizamuddin, H. R. Hasan, K. Salah, Ipfs-blockchain-based authenticity of online publications, in: Blockchain - ICBC 2018 - First International Conference, Held as Part of the Services Conference Federation, SCF 2018, Seattle, WA, USA, June 25-30, 2018, Proceedings, 2018, pp. 199–212.

[37] 范贤丽, 范春晓, 吴岳辛, 基于区块链和IPFS技术实现粮食供应链隐私信息保护, 应用科学学报 (02) (2019) 179–190.

[38] T. Xiao, Y. Huang, Design and implementation of web system based on blockchain, in: Cloud Computing and Security - 4th International Conference, ICCCS 2018, Haikou, China, June 8-10, 2018, Revised Selected Papers, Part II, 2018, pp. 706–717.

[39] N. Rifi, E. Rachkidi, N. Agoulmine, N. C. Taher, Towards using blockchain technology for iot data access protection, in: 17th IEEE International Conference on Ubiquitous Wireless Broadband, ICUWB 2017, Salamanca, Spain, September 12-15, 2017, 2017, pp. 1–5.

[40] D. xu, X. Hu, The design of npki system based ldap, 2011 International Conference on Intelligence Science and Information Engineering (2011) 202–205.

[41] J. T. M. Viteri, M. I. G. Valero, A. R. E. León, User management with ldap(light weight directory access protocol)for access to technology and information services in companies, Journal of Science and Research 1 (2016) 10–15.

[42] P. K. Rational, Architectural blueprintsthe 4+1 view model of software architecture, 1995.

简历与科研成果

基本情况 杨登辉，男，汉族，1996 年 2 月出生，湖北省咸宁市人。

教育背景

2017.9～2019.6 南京大学软件学院 硕士

2013.9～2017.6 华中科技大学软件学院 本科

参与项目

1. 南京南瑞集团项目：移动众测平台软件（20170413），2017-2018
2. 国家新技术实验室创新项目-面上项目：基于群体智能的移动应用自动化测试研究（ZZKT2017B09），2017.10-2019.9

致　　谢

两年的硕士研究生学习生涯即将结束，在这两年时光里，许多老师和同学给予我帮助，关心，教导。在论文完成之际，我想向所有帮助、关心指导过我的人致以最真挚的感谢。

首先我要由衷地感谢我的导师陈振宇教授。感谢陈老师在研究生期间对我严厉要求，教导我认真对待每一个项目，每一份工作。感谢陈老师在毕业设计过程中认真负责，有耐心的指导论文的细节，指出毕业项目中不足之处，为学生的毕业设计提供全力支持。

其次我要感谢在论文指导上提供帮助的房春荣老师以及技术上指导我项目设计与开发的黄勇老师。感谢房老师能够花费时间和精力指导我写作方式以及论文修改。感谢黄勇老师两年来一直在技术工程领域引领我们前进，交流技术心得，费心讲解各种先进的技术和项目架构。

接着我要感谢实验室两年来一起开发各个项目，一起解决各种难题的同学。在每个项目中，我们组成一个团队一起分析需求、规划迭代计划、分享技术心得、相互协作彼此扶持。

然后我要感谢南京大学软件学院和ise实验室两年来对我的培养。提供优秀的教学资源和良好的基础设施，包括学院机房设备、实验室工作环境、云服务器资源。

最后我要再一次感谢我的导师陈老师。感谢导师在研究生期间对我的指导和栽培。并向每一位参与本论文审稿的老师表达最真诚的谢意。

版权及论文原创性说明

任何收存和保管本论文的单位和个人，未经作者本人授权，不得将本论文转借他人并复印、抄录、拍照或以任何方式传播，否则，引起有碍作者著作权益的问题，将可能承担法律责任

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

作者签名：

日期：　年　月　日