南京大学

研 究 生 毕 业 论 文

（申请工程硕士学位）

论 文 题 目 基于区块链技术的软件资产交易子系统设计与实现

作 者 姓 名 唐珊珊

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 刘嘉 副教授

2019 年 4 月 13 日

学　　号：

论文答辩日期：**2019 年 4 月 13 日**

指 导 教 师：　　（签字）

The Design and Implementation of Software Asset Transaction Subsystem Based on Blockchain Technology

By

Shanshan Tang

Supervised by

Associate Professor **Jia Liu**

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

April 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：基于区块链技术的软件资产交易子系统设计与实现

工程硕士（软件工程领域） 专业 2017 级硕士生姓名： 唐珊珊

指导教师（姓名、职称）： 刘嘉 副教授

摘 要

在目前软件侵权盗版现象普遍，测试数据真实性存疑的背景下，软件作为一种数字资产，难以在互联网上进行可靠的确权和交易。本文提出基于区块链技术的软件资产管理系统，利用区块链技术去中心化、数据可追溯、不可篡改的特性，实现软件资产及时确权和可靠交易，建立新一代软件价值流通体系，提高软件价值流通效率。

本文首先分析软件资产管理系统的业务需求，论证使用区块链技术的必要性。从数据存储、共识机制和智能合约三个方面对比现有的区块链技术，制定适合面向软件资产管理系统的区块链平台的技术路线，依据技术路线选取开源框架Hyperledger Fabric 完成基于区块链的软件资产交易子系统的设计开发，利用区块链技术保证软件资产数据不可篡改。基于智能合约实现了软件自动验收功能，提高软件的交付和验收效率，保证验收过程公开透明，有助于减少软件交易纠纷。

为解决软件资产管理系统需要存储大量文件和区块链存储能力低下的矛盾，本文设计了链上存储和链外存储相结合的基于IPFS 的区块链文件存储方案，扩展区块链存储能力的同时也保证了文件的不可篡改性。本系统提供高度抽象化的，功能全面简单易用的编程接口，在数据设计、架构设计、代码层次上都维持了良好的扩展性，为灵活应对业务变化打下良好的基础。提供区块链平台的自动化部署方案，简化后续的开发、测试和部署流程。

最终测试和分析结果表明软件资产交易子系统能够实现基于区块链的软件资产数据存储和文件存储，利用智能合约完成自动验收。系统吞吐量达到248tps，能够支持真实场景中的业务需求。在保证系统可用性和安全性的前提下提供1/3的容错率。目前已有测评机构接入系统。本文是区块链技术应用于软件领域的一次探索，验证了使用区块链技术保护软件资产的可行性，为软件资产的交易和流通系统提供新思路。

关键词： 软件资产，区块链，智能合约，软件测试，软件验收

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 Asset Transaction Subsystem Based on Blockchain Technology

SPECIALIZATION: Software Engineering

POSTGRADUATE: Shanshan Tang

MENTOR: Associate Professor Jia Liu

Abstract

With the development of Internet technology, digital assets are widely used. In the context of the current phenomenon of software infringement and piracy, software as a digital asset is difficult to confirm and trade on the Internet. This thesis proposes a software asset management system based on blockchain technology, which utilizes the characteristics of decentralization, data traceability and non-tamperability of blockchain technology to ensure timely verification of software assets and reliable transactions, to improve the efficiency of software value circulation.

This thesis analyze the business requirements of the blockchain-based software asset management system , and demonstrated the necessity of using blockchain technology. Compare the existing blockchain technology from data storage, consensus mechanism and smart contract aspects, and selected the open source framework Hyperledger Fabric to implement the software asset transaction subsystem. Implement software auto-acceptance based on smart contracts, which improves software delivery and acceptance efficiency, ensures open and transparent acceptance process, and helps to reduce software transaction disputes.

In order to solve the contradiction between the large files storage and the low storage capacity of the blockchain, an IPFS-based blockchain file storage scheme combining on-chain storage and off-chain storage is designed, which expands the blockchain storage capacity. The function of the software asset transaction subsystem is encapsulated into a highly abstract, fully functional and easy-to-use programming interface, which maintains good scalability in data design, architecture design, and code level, and lays a good foundation for flexible response to business changes. System provides

an automated deployment solution that simplifies development, testing, and deployment processes.

The final test and analysis show that the software asset transaction subsystem can well support blockchain-based software asset data storage and file storage, using smart contracts to complete automatic acceptance. System throughput reaches 248tps. System provides 1/3 of the fault tolerance rate while ensuring system availability and security. This work is an exploration of the application of blockchain technology in the field of software. It verifies the feasibility of using blockchain technology to protect software assets, and provides new ideas for the trading and circulation system of software assets.

Keywords: Software Asset, Blockchain, Smart Contract, Software Testing, Software acceptance

目　录

表　目　录

图　目　录

第一章　引言

1.1　项目背景及意义

近年来互联网技术不断发展，越来越多的资产以“数字资产”的形式存在。常见的数字资产包括数字积分和虚拟货币等。软件也是一种数字资产。传统资产的交换依赖于面对面的物物交换。像软件这样纯数字化的资产可以容易地被下载、复制和传播，方便人们进行资产分配和交换，但由于缺少物理凭证，难以明确所有权的归属 [1]。与其它数字资产相比，软件资产变现的方式不是传统的所有权转让，而是通过提供服务产生价值，软件服务的质量则需要由软件测试数据来量化评估。在软件资产交易过程中，为保证软件资产的质量，需要对软件进行充分测试，建立严格的软件验收制度，落实验收流程。软件的测试需要丰富的专业设备和较强的专业能力，一般交由专业的第三方软件测评机构完成 [2]。因此，软件资产管理需要关注所有权的确认以及测试数据的真实可靠。

软件资产管理建立在使用正版软件的基础上。使用盗版软件不仅存在潜在的法律风险，而且由于盗版软件不太可能收到关键的安全补丁和更新支持，更容易受到攻击，软件盗版是对软件资产管理的真正威胁 [3]。但随着软件作为数字资产的价值增长，全球软件盗版的速度也在增长。造成软件侵权、盗版现象普遍的原因是软件资产所有权确权难、追溯难，投诉手续复杂。软件作为一种数字资产，易于仿制，更容易出现伪造现象 [4]。非法拷贝的传播成本低，导致软件资产价值流失 [5]。此外，在软件资产流通过程中也存在诸多风险。软件资产往往存储于云存储等中心化存储环境中，存在资产数据和交易数据被篡改的风险，容易导致资产流失等不安全事件 [6]。测试数据作为评估软件服务价值的重要依据，在传递过程中容易被篡改或造假，导致软件资产的价值不能得到正确的评估。软件验收过程不透明，软件采购方和供应方之间可能产生纠纷。如何保证软件资产在互联网上进行可靠的确权与流通成为接下来要解决的问题。

区块链技术由于其去中心化、可追溯、防篡改的特性，受到政府和各行业的广泛关注，利用共识算法、智能合约等技术构建数据层面的信任，实现由整个网络来记账，而非某一个中心机构。重点是实现了多个参与方之间在数据层面的不可篡改和不可抵赖 [7]。基于区块链技术可以构建一个多方参与的去信任化的软件资产管理系统，无需可信的第三方中介就可以完成价值交换过程 [8]。基于区块链建立新一代软件资产管理系统，利用区块链的链式数据存储、共识机制和智能合约能技术，具有以下优势：1）实现软件资产所有权的全网证明，

及时确权。2）不同机构间无需信任即可进行软件资产交易。3）保证软件测试数据、交易记录和所有权数据不可篡改，侵权行为和非法交易发生时能够做到有据可查，快速维权。4）实现基于智能合约的自动验收功能，节省人力成本，提高软件资产变现效率。

1.2　国内外研究现状

1.2.1　软件资产管理系统研究现状

企业为保证使用的软件已经获得授权、避规避法律风险、更合理地安排软件采购支出，往往会建立企业内部的软件资产管理系统 [9]。其目标是管理、控制和保护一个组织的软件资产，包括管理使用其他软件资产所带来的风险。国际上有许多与此相关的标准颁布，如ISO/IEC19770-1、ITTL、IBPL等，为企业如何实施软件资产管理提供了成熟的标准 [10]。目前也有许多基于以上标准开发的企业软件资产管理系统。但是此类系统都是在企业内部使用，在企业采购软件之后对所有的软件资产进行统筹和规划，无法跨组织使用，更无法对软件的确权和交易过程进行管理。

扩展至数字资产管理领域，目前国内外大部分研究多从某一细分领域讨论数字资产管理面临的挑战，对于数字资产管理可应用的技术研究主要集中在区块链领域。梅兰妮·斯万在2015年首次提出区块链金融的概念，她认为任何形式的资产都可以在区块链上进行注册、存储和交易，不仅仅是加密货币，还包括知识产权、软件、医疗数据等 [11]。在此基础上，Martin Zeilinger等提出了一种基于区块链保护数字艺术作品版权的方法 [12]。作者指出，可以利用区块链的公共区块哈希标识数字艺术品的原始出处和作者身份，避免版权盗用，利用只能合约完成数字艺术品的登记和出售。Garrick Hileman 等分析了区块链技术用于数字资产管理的优势 [13]。他认为，与传统的中心化系统相比，使用区块链的分布式账本存储资产数据能够提高系统效率，防止数据被恶意篡改，增强系统的可审计性，减少运营成本，区块链的P2P 架构可以抵御某些类型的网络攻击，提供更高的可用性和可靠性。

虽然国内外许多学者提出了数字资产管理的初步方案，但是这些方案都还处于理论阶段，尚无实践表明可行。此外，它们针对的是广泛的数字资产管理，没有针对软件资产在交易过程中需要测试和验收的特点进行设计和优化。

1.2.2　区块链技术研究现状

区块链这一概念源于比特币的底层实现技术。2008年“中本聪”发表了论

文《比特币：一种点对点电子现金系统》[14]，并在次年发布了首个区块链货币加密系统——比特币（Bitcoin），网络上互不信任的个体可以在无中介的情况下使用比特币进行支付。2014 年出现的以太坊（Ethereum）区块链平台 [15] 在比特币的基础协议之上提供了图灵完备的编程语言以供用户来构建任何可以精确定义的智能合约或交易类型，首次将智能合约应用到区块链。2015年，Linux 基金会发起了开源的商业区块链项目超级账本（Hyperledger）[16]。超级账本旗下有多个区块链平台项目，包括IBM 贡献的Fabric 项目，Intel 贡献的Sawtooth 项目，以及Iroha、Burrow、Indy 等。其中最受关注的是Fabric项目，不同于比特币和以太坊，Hyperledger Fabric是专门针对于企业级的区块链应用而设计的联盟链平台，成员节点必须经过授权才可加入 [17]。2017 年腾讯发布了TrustSQL，提供了自适应的共识机制、秒级的交易确认，支持使用SQL 语句进行数据插入和查询[1]。

近年来，区块链技术发展迅猛，截止到2018年3月底，我国主营区块链业务的公司数量达456 家，应用领域从最初的金融领域扩展到医疗、能源、公益、物联网等多个领域 [18]。麦肯锡研究报告中提到：区块链是目前最有潜力触发颠覆性革命浪潮的技术 [19]。

1.3　本文的主要工作

本文根据软件资产的特性，分析软件资产在确权和流通过程中的痛点，明确区块链在软件资产管理上的应用优势，设计了基于区块链的软件资产管理系统。利用区块链技术，在软件资产的生产、确权、交易、验收过程中涉及的多个机构之间构建一个去中心化、去信任化的网络，实现软件所有权信息的即时全网公正，保证软件资产、测试数据、交易数据不可篡改，利用智能合约进行自动验收，从而使得软件资产的确权和交易流程更加可靠和高效。

本文基于区块链技术实现了软件资产管理系统的软件资产交易子系统。首先，根据软件资产管理的需求定义所需的数据实体，根据数据特点和区块链的技术特点设计数据存储方式，定义一套语义明确，扩展性强，功能完备的数据访问接口。其次，利用智能合约实现自动验收功能。在此基础上，为软件资产管理系统的业务后台提供与区块链平台交互的客户端，将系统功能包装为简单易用的编程接口，方便业务后台的开发。最后，为区块链网络的众多组件提供一套自动化的部署方案，简化网络搭建、智能合约部署的流程，方便系统的部署和实施。

[1] https://trustsql.qq.com/

1.4　本文的组织结构

本文总共分为五个章节，组织结构如下。

第一章，引言部分。介绍项目的背景，分析了国内外在软件资产管理和区块链方向的研究现状以及待解决的问题。

第二章，相关概念与技术。对比相关技术的优缺点，制定了项目的技术方案，介绍项目中使用的核心概念和技术，Hyperledger Fabric框架、星际文件系统IPFS、LDAP、Docker等。

第三章，软件资产交易子系统的需求分析与概要设计。通过涉众分析和用例分析明确了项目需求，绘制了用例图和时序图展示分析需求的结果，并以用例描述表的形式给出了主要用例的具体内容。对项目进行了概要设计，从逻辑视角、开发视角、进程视角和部署视角展示了项目架构，给出数据持久化模型，并对主要模块进行了进一步分析与设计。

第四章，软件资产交易子系统的详细设计与实现。在概要设计的基础上，完成了链上文档存储模块、自动验收模块、链外文件存储模块、用户认证模块的详细设计，给出了类图和时序图，并展示了关键代码。

第五章，总结与展望。总结论文期间所做的工作，并就项目的未来方向作了进一步展望。

第二章　相关概念与技术

2.1　区块链

传统的数据库管理系统（关系数据库、NoSQL非关系数据库）都是由中心机构进行管理和维护，中心机构对所有数据拥有绝对的控制权。其它机构无法了解数据的存储更新过程，因而无法完全信任数据库中的数据。在多个机构协作的模式下，使用传统的中心化数据库无法解决信任问题，因此每个参与方都需要维护一套独立的数据库管理系统，机构间的协作需要耗费大量人力和时间成本进行数据对账和清算。如果存在一种多方维护、多方信任的数据库，则可以显著减少清算成本。

区块链（Blockchain）就是这样一种多方维护的分布式数据库，任何一方都无法完全控制这些数据，只能按照严格的规则和共识进行更新，从而实现了可信的，多机构之间的信息共享和监督。区块链技术是近年来的一种新兴技术，由计算机领域的众多经典技术组合而成，包括共识算法、哈希算法、P2P网络技术、非对称加密技术等 [20]。狭义来讲，区块链是一种按照时间顺序将数据组织为一个个区块并以哈希值连接起来的一种链式数据结构。广义的区块链技术则是利用这种链式数据结构来存储和验证数据、利用分布式共识算法来更新数据、利用自动化脚本代码(智能合约) 来操作数据的一种去中心化架构模型与分布式计算范式 [21]。区块链基础架构模型如图 2.1 所示 [22]。

区块链具有去中心化、不可篡改性、匿名性和可验证性四个特点 [23]。1）去中心化。区块链在数据层面构建分布式节点之间的去信任化环境，基于区块链可以真正实现点对点的交易而无需中心机构的验证，由此降低交易的额外消耗，交易的时间也不会受到其它机构性能的影响。2）不可篡改性。发生在区块链系统中每一笔交易都会被广播至所有节点，已打包的区块会被各个节点所验证和记录。同时，使用PoW等分布式共识算法形成强大的算力抵御外部攻击，对区块中交易进行篡改几乎是不可行的。3）匿名性。在区块链系统中，用户通过按照一定规则自动生成的单个或多个地址与其它用户进行交易，其本身信息并未体现其中。这在一定程度上保证了用户的隐私性。4）可验证性。区块链系统为数据增加了时间维度，发生在区块链系统中的每一笔交易都被赋予一个时间戳，具有极强的可验证性和可追溯性。

区块链按准入机制分为三类：公有链，联盟链和私有链 [24]。1）公有链。面向全网公开，任何个体或者团体都可以加入或退出网络、发送交易、且交易

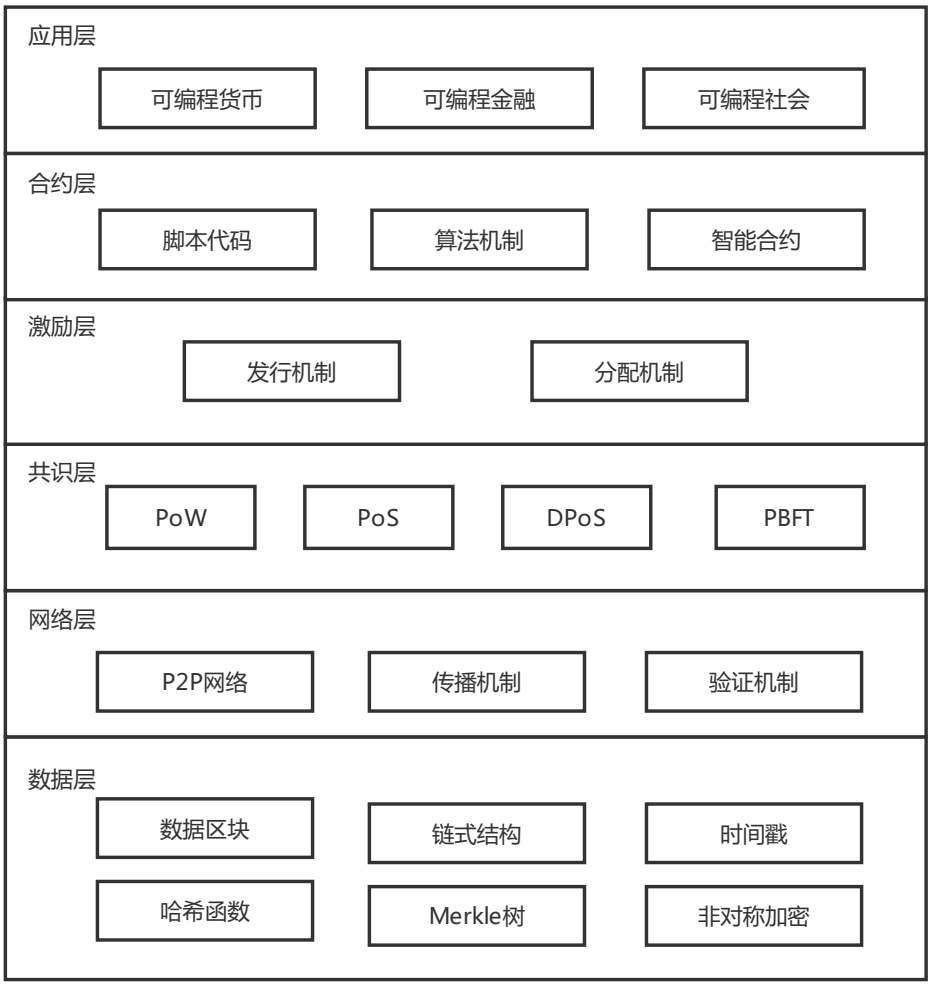

图 2.1: 区块链基础架构模型

能够获得该区块链的有效确认。一般依赖代币机制维护系统安全运行。2）联盟链。由某个群体内部指定多个预选的节点作为记账人，每个区块的生成由所有记帐节点共同决定。该类区块链有权限控制机制，只有被允许的联盟成员才能够参与账本维护。3）私有链。仅仅使用区块链技术进行记账，可能是一个组织或个人独享该区块链的写入权限。

接下来本文将针对区块链的特点和项目需求，对区块链实现的关键技术：数据存储、共识算法、智能合约进行研究。

2.1.1　数据存储

从项目需求出发，我们需要将软件、测试报告等文件存储在区块链上。但由于区块链要求系统内每个节点保存一份数据备份, 这对于日益增长的海量

数据存储来说是极为困难的。截至2017 年5月，比特币区块链的容量已经达到107.89GB，并且还在逐日增加。目前有近1000万个节点贡献了100GB 以上的存储空间来维护区块链网络。也就是说比特币系统使用了超过1000PB的空间仅仅存储了100GB左右的数据。以太坊存放大量数据也是极其昂贵的，存储1MB的数据需要花费3.76ETH。可见，区块链存储数据成本高，容量小，将文件数据全部存储在链上是不明智的。通常的做法是将文件存储在链外，链上存储文件的哈希值。链外文件存储可采用中心化存储和分布式存储两种方案。当前多采用中心化的云存储，因为它实现简单，可以利用很多成熟的云存储服务。但中心化的云存储存在以下问题：1）成本：典型的云存储数据中心都是大型服务器，需要严格维护，成本高昂；2）效率：通常数据中心距离用户较远，数据读写会有延迟；3）安全性：中心服务器被攻击的风险较高，如果测试报告等原始文件丢失，区块链上存储的哈希数据也将失去意义；4）可用性：存在单点故障，中心服务器不可用会导致整个系统不可用。而采用分布式存储网络除了开发较为困难之外，优点众多：成本低、可无限扩展存储容量，由于原始文件分布式的存在各个节点上，每个节点只保存部分数据，数据被盗取的风险大大降低。迅雷链正是采用了这种方式解决区块链的存储问题[1]。

表 2.1: 文件存储方案对比

实现方式	开发难度	存储成本	安全性	扩展性
中心化存储	低	低	低	低
分布式存储	高	高	高	高

经过表 2.1 的对比，本文采用分布式存储的方式将文件存储在链外，并将文件哈希值写入区块链，以保证文件数据的不可篡改性。分布式文件存储网络将采用IPFS(见 2.3 节) 进行搭建。

2.1.2　共识机制

如何达成高效共识是分布式网络中要解决的重要问题。正如社会系统中“民主”和“集中”的对立关系相似, 决策权越分散的系统达成共识的效率越低、但系统稳定性和满意度越高; 而决策权越集中的系统更易达成共识, 但同时更易出现专制和独裁。区块链中共识机制的设计的目标是平衡区块链的一致性和可用性，在不过分影响实际使用体验的情况下保证相对可靠的一致性 [25]。区块

[1]https://open.onethingcloud.com/

链技术上支持的典型共识机制有工作量证明（PoW）、权益证明（PoS）和拜占庭一致性协议（PBFT） [26]。三类共识机制的对比见表 2.2：

表 2.2: 共识机制对比

共识机制	一致性	容错率	扩展性	性能	资源消耗
工作量证明	有分叉	<50%	差	高延迟	高
权益证明	有分叉	<50%	良好	低延迟	低
拜占庭一致性协议	无分叉	<33%	差	低延迟	低

工作量证明机制中，只有完成工作量证明的节点才能提出这一阶段的待定区块，此后网络中的节点在这一区块后继续尝试完成工作量证明，产生新的区块。当某一节点收到两个不同的待定区块时，选择链更长的那个区块进行验证。链越长意味着该链包含的工作量越多。工作量证明机制存在严重的效率问题，且需要耗费大量算力和电力资源，造成资源的浪费。后来有学者提出可以尝试通过解决寻找大素数、求解正交向量、最短路径等问题代替现有的无意义的计算来避免资源浪费，但仍然无法解决效率问题 [27]。

由于工作量证明机制在资源消耗和效率方面表现不佳，权益证明机制受到广泛关注。由于PoW 中收益取决于节点算力，使得一些用户只专注于提高算力而忽视了区块链系统生态的维护。权益证明本质上是采用系统中的相关权益证明代替PoW中的算力证明。合理假设，权益的拥有者更乐于维护系统生态。在PoS的基础上发展出了股份授权证明机制（DPoS），首先以权益作为选票选出记账节点，然后在记账节点之间轮流记账。DPoS大幅降低了参与记账节点的数量，一定程度上提升了效率。但目前尚无完善的，基于权益证明机制的区块链的实际应用。

拜占庭一致性协议主要研究在分布式系统中，如何在有错误节点的情况下实现系统中所有正确节点对某个输入值达成一致。在去中心情况下，拜占庭一致性协议可实现区块链一致性，且无需多余的计算量，在性能上有良好的表现，但在安全性和可扩展性方面存在问题，其安全性依赖于失效节点不超过全网节点的1/3，适合于来源较为可靠的私有链或联盟链 [28]。

拜占庭一致性协议虽然在容错率和扩展性方面表现不佳，但是在一致性、性能和资源消耗方面表现优秀。本项目中，加入区块链网络的组织和机构（如软件测评机构，软件提供商等）需要经过授权和验证，不存在不受控制的恶意节点，并且节点总数较少，因此对共识机制的容错率和扩展性要求较低，因此决定采用拜占庭一致性协议作为共识机制搭建联盟链。

2.1.3　智能合约

智能合约是1994年由密码学家尼克萨博(Nick Szabo)提出的理念。根据Nick Szabo的定义：智能合约是指能够自动执行合约条款的计算机程序 [29]。由于缺乏相应的平台来执行合约，智能合约一直停留在理论阶段，区块链技术的出现恰好为智能合约提供了安全可靠的记录载体和执行环境。智能合约逐渐成为区块链技术应用最重要的特征 [30] [31]。

区块链上的智能合约必须同时具备两种性质：确定性和可终止性。智能合约会在区块链网络的多个节点中运行。如果一个智能合约是非确定性的，那么不同节点运行的结果就可能不一致，从而导致共识无法达成，网络陷入停滞；如果智能合约是永不停止的，那么节点将耗费无穷多的时间和资源去运行合约，同样导致网络进入停滞状态。

表 2.3: 智能合约方案对比

实现方案	如何避免非确定性	如何保证终止性	执行环境
比特币方案	合约代码使用非图灵完备的基于栈的脚本编写。不提供可能导致不确定性的指令	只提供有限的指令集，不提供可能导致死循环的跳转、循环指令	直接执行
以太坊方案	提供图灵完备的智能合约语言。不提供系统函数，只能访问链内数据从而保证确定性	使用计价器。按照指令收费，统计gas费用，gas消耗完则直接终止	在虚拟机（EVM）中执行。区块链的业务逻辑与EVM高耦合
Fabric方案	提供图灵完备的智能合约语言，可访问系统函数。底层机制上无法避免非确定性，只能依靠开发人员自我约束	使用计时器。用时间标准衡量一个合约是否陷入死循环	在容器（Docker）中执行。低耦合的设计，区块链与容器之间几乎没有依赖关系

表 2.3 对比了技术上较为成熟的三种智能合约方案，可以发现，Fabric 方案采用高度模块化的思想，提供了高内聚低耦合的通用技术框架和灵活的智能合约编写语言，便于扩展和移植。因此选用Fabric 方案来实现智能合约。

2.1.4 Hyperledger Fabric

基于上文中结合项目需求对区块链技术的调研，我们选用了支持拜占庭一致性协议，提供了高度模块化的智能合约方案的Hyperledger Fabric 作为开发框架。Hyperledger Fabric（以下简称Fabric）是在Linux基金会下建立的一个开源的企业级联盟链平台 [32]。与其他流行的区块链平台相比，Fabric提供了一些关键的差异化功能：

- 模块化。Fabric具有高度模块化的架构，提供了可插拔的共识协议和身份管理协议。其智能合约可以使用通用编程语言进行编写，在容器环境内运行以进行隔离，可以方便地升级和扩展。
- 隐私和机密性。Hyperledger Fabric是一个经过许可的平台，通过其通道架构实现机密性。基本上，Fabric 网络上的参与者可以在参与者子集之间建立一个“通道”，该通道应该被授予对特定交易集合的可见性。因此，只有那些参与通道的节点才能访问智能合约和交易数据，从而保护两者的隐私和机密性。
- 性能。主流区块链平台中，性能一直是一个难以解决的问题。目前比特币每秒只能进行大约7笔交易，即7 TPS，以太坊也只有10-20 TPS。而经过1.1版本优化之后，Fabric的系统吞吐量达到2700 TPS，节点网络利用率达到1680 Mbps (发送)和240 Mbps (接收) [33]。

在Fabric中，智能合约叫做chaincode，实质上是控制区块链网络中的不同实体之间交互的业务逻辑。chaincode是独立可运行的应用程序，部署在区块链的网络节点上，在Docker容器（见 2.4节）中运行。它是与区块链交互的唯一渠道和生成交易（Transaction）的唯一来源。chaincode可以由Go，Java或Node.js编写，Fabric提供了chaincode API以访问和改变账本数据，并提供了安装、初始化、升级命令用来管理chaincode的生命周期[2]。

Hyperledger Fabric 提供了一个可选的组件Fabric CA，对网络内各个实体的身份证书进行管理。它负责Fabric 网络内所有实体身份的注册和证书管理，包括签发、更新和撤销数字证书（身份证书ECerts和交易证书TCerts）。Fabric CA在Hyperledger Fabric 网络中的作用如图 2.2 所示。

[2]https://hyperledger-fabric.readthedocs.io/en/release-1.3/

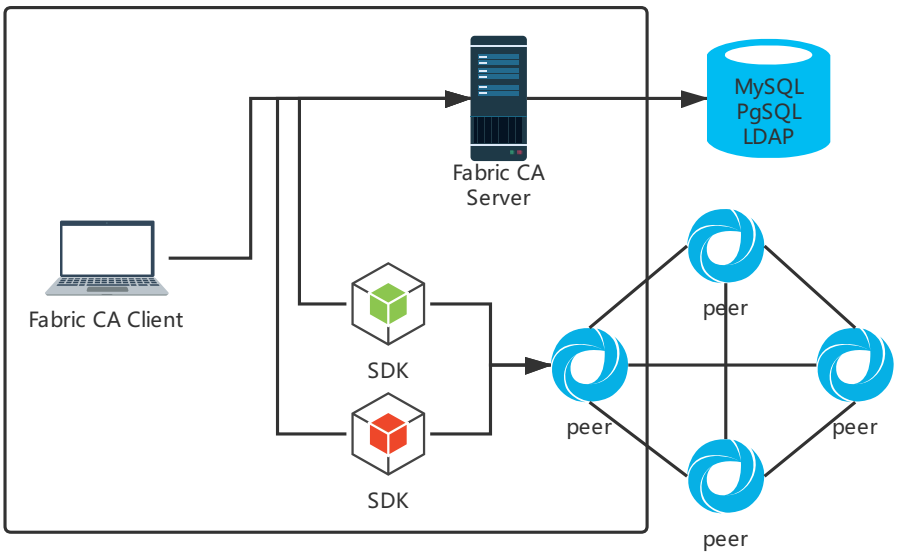

图 2.2: Fabric CA在区块链网络中的作用

Fabric CA提供了客户端和SDK通过HTTP接口访问Fabric CA Server，Fabric节点可以通过SDK 与Fabric CA Server 进行交互。Fabric CA 的身份信息保存在数据库或LDAP中。目前Fabric CA 支持的数据库有MySQL、PostgreSQL、和SQLite，默认使用SQLite 数据库。如果配置了LDAP，则身份信息将保留在LDAP 而不是数据库中。

2.2　LDAP

LDAP是目录访问协议的一种，它基于X.500标准，但是简化了实现方法，所以称为轻量级的目录服务 [34]。与关系数据库不同，LDAP的信息模型建立在“条目”的基础上，条目是属性的集合，每个条目具有一个全局可辨识的DN（Distinguished Name），用户可以通过DN唯一地引用条目。条目倾向于包含描述性的属性信息，支持复杂的过滤器查询操作 [35]。

LDAP将条目组织为树形结构，在目录服务中称之为目录信息树（Directory Information Tree，DIT）。LDAP实现了单个条目或批量条目的增删改查，并对搜索进行了优化，允许通过设置查询过滤器匹配查询规则，限定在DIT的任一分枝上进行查找，这一特性使目录服务相比于关系型数据库的数据处理速度快一个数量级。

LDAP支持广泛的应用编程语言，如C、Java、PHP、Perl等都有自己的LDAP API，用户可以轻松选择适合自己的编程语言进行LDAP相关应用的开发。最新

的LDAP协议版本是V3，其后续的开发已经纳入全球互联网最具权威的技术标准化组织IETF（Internet Engineering Task Force）。

目前，LDAP协议已经成为一个被广范支持的用于统一身份认证跨平台和标准的协议，它易于集成不同应用系统的特点成为支持网络系统的重要底层基础技术之一，是进行统一身份认证的首选方案。

考虑到许多企业和机构已经在使用LDAP做统一身份认证，为方便软件资产交易系统在机构中部署和实施，提供通用的身份认证接口，本文使用LDAP存储用户身份数据，结合Facric CA实现用户身份认证。

2.3 IPFS

星际文件系统IPFS（Inter-Planetary File System）[3]是一个面向全球的、点对点的分布式版本文件系统，目标是为将所有具有相同文件系统的计算设备连接在一起 [36]。IPFS使用基于内容的寻址，不关心文件的名字和路径，只关注文字中可能出现的内容。一个文件被放入IPFS时，会生成一个由文件内容计算出的加密哈希值，哪怕文件只修改1比特，哈希值也会完全不同。使用一个哈希值向IPFS请求文件时，它会通过分布式的哈希索引表找到文件所在的节点，验证并返回文件数据 [37]。IPFS通过网络删除重复具有相同哈希值的文件，并跟踪每个文件的版本历史记录。每个网络节点只存储它感兴趣的内容以及一些索引信息，使得文件系统速度更快、更安全、更健壮、更持久。

IPFS基于内容的寻址方式使得它可以很好地与区块链结合。区块链只存储文件IPFS生成的加密哈希，文件本身存储于区块链外的IPFS系统中。用户无需信任IPFS中的文件数据，可以通过区块链中存储的哈希值来验证文件是否经过篡改 [38]。这种方案在大大减轻区块链存储负担的同时，也保证了文件数据的不可篡改性 [39]。因此本文使用IPFS结合区块链实现可靠的文件存储。

2.4 Docker

Docker 是一个开源的应用容器引擎，能够为任何应用快速的创建一个可移植的、轻量级、自给自足的容器 [40]，容器使用沙箱机制，可以将一个物理主机分成若干个相互独立的部分，有效利用物理主机资源 [41]。Docker通常用于以下场景：自动化打包和部署应用程序，创建轻量、私密的运行环境，自动化测试和持续集成等。

[3]https://ipfs.io/

Docker具有以下三个特点：1）轻量。在一台机器上运行的多个Docker容器可以共享这台机器的操作系统内核；它们能够迅速启动，只需占用很少的计算和内存资源。镜像是通过文件系统层进行构造的，并共享一些公共文件。这样就能尽量降低磁盘用量，并能更快地下载镜像。2）跨平台。Docker容器基于开放式标准，能够在所有主流Linux版本、Microsoft Windows以及包括VM、裸机服务器和云在内的任何基础设施上运行。3）安全。Docker赋予应用的隔离性不仅限于彼此隔离，还独立于底层的基础设施。Docker 默认提供最强的隔离，因此应用出现问题，也只是单个容器的问题，而不会波及到整台机器。

此外，Docker提供了一个工具软件Compose用于定义和运行多容器Docker应用。借助Compose，开发者可以使用YAML文件来配置应用程序的服务，使用单个命令即可从配置中创建并启动所有服务。由于本项目涉及的组件较多，为简化配置，提高测试和部署效率，本文利用Docker容器对项目进行编配和部署。

2.5　本章小结

本章首先对区块链技术进行了介绍，对区块链的关键实现技术数据存储、共识机制和智能合约的主流实现技术进行了对比研究，结合项目需求确定了项目的技术方案。接着选定了符合技术方案的开发框架Hyperledger Fabric，最后简要介绍了项目开发过程中要用到的其它技术和工具。

第三章　软件资产交易子系统的需求分析与概要设计

软件资产交易子系统是软件资产管理系统的一部分，为其提供基于区块链的可靠数据存储，管理软件资产交易中的确权、流通和验收过程。为完成软件资产交易子系统的分析和设计，首先需要对软件资产管理系统进行需求分析，明确系统的功能需求和非功能需求。依据系业务需求，设计出功能完备的子系统。本章首先对软件资产管理系统进行需求分析，从中获取软件资产交易子系统的需求，进而完成子系统的设计。本章用到的符号含义如表 3.1 所示。

表 3.1: 符号表

符号	含义	符号	含义
$R = \{R_1, \ldots, R_i, \ldots, R_n\}$	软件需求方集合	S_i	软件制品
$D = \{D_1, \ldots, D_i, \ldots, D_n\}$	软件开发方集合	$Cert_i$	软件证书
$T = \{T_1, \ldots, T_i, \ldots, T_n\}$	软件测评机构集合	Req_i	软件需求
$O = \{O_1, \ldots, O_i, \ldots, O_n\}$	订单集合	AC_i	软件验收标准
$K^p_{user_i}, K^s_{user_i}$	用户公钥、用户私钥	TR_i	软件测试结果
$Hash(M)$	M 的哈希值	AR_i	软件验收结果
$Sign(M, K^s_{user_i})$	私钥$K^s_{user_i}$ 对M 的数字签名		

3.1　软件资产管理系统需求分析

3.1.1　涉众分析

基于软件行业中存在的软件资产确权、维权难，验收和交付效率低下，缺少可信凭证的现状，我们提出基于区块链的软件软件资产管理系统，管理软件资产从生产到交易验收的整个生命周期，在软件需求方、开发方和软件测评机构之间建立可靠的软件资产登记和流通体系。软件资产管理系统的涉众及其特征和期望如表 3.2 所示。软件需求方表示为$R = \{R_1, \ldots, R_i, \ldots, R_n\}$，软件开发方表示为$D = \{D_1, \ldots, D_i, \ldots, D_n\}$，软件测评机构表示为$T = \{T_1, \ldots, T_i, \ldots, T_n\}$。

表 3.2: 系统涉众分析

涉众	涉众特征与期望
软件需求方	因自身的业务发展而产生了软件需求，但不具备开发和测试的专业能力。需要委托专业的软件开发机构完成软件开发。对软件制品的质量有一定的要求，委托有资质的软件测评机构出具软件测试报告以辅助验收过程。希望资产管理系统能够帮助他们与有资质的开发方及测评机构达成合作，减少软件采购过程中的不确定性，提高验收效率。
软件开发方	有专业的软件开发能力，接受软件需求方的委托，以达到需求方的验收标准为目标进行软件开发。他们希望软件的所有权归属能够明确，在软件被盗版和非法使用时能够快速维权，希望软件的需求和验收标准是透明可靠的。
软件测评机构	有专业的测试能力，接受委托对软件进行功能测试和性能测试，能够给出具有说服力和可信度的测试报告。为保证自身的公信力，希望出具的测试报告在传递过程中不被篡改。

3.1.2　用例分析

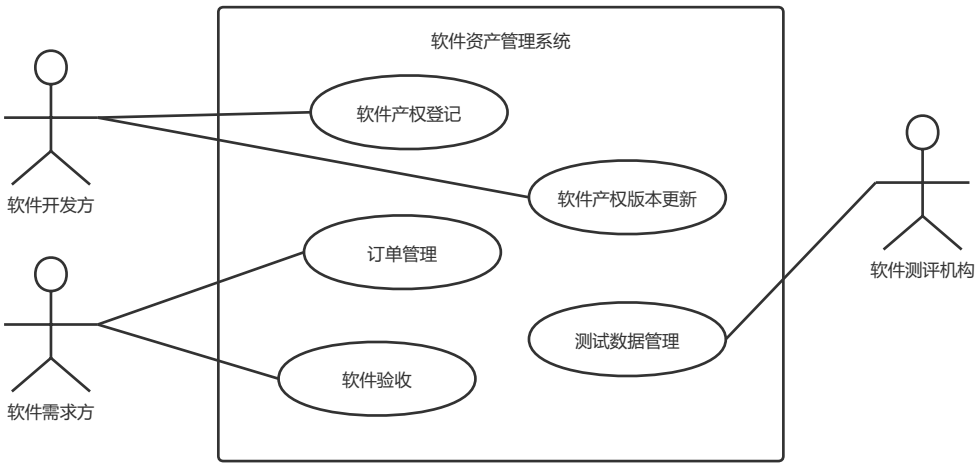

图 3.1: 系统用例图

根据系统涉众的特征和期望，得出系统用例如图 3.1 所示。在软件资产的整个生命周期中，我们设计了一个订单实体来关联软件交付过程中的各方参与者，并记录与之关联的软件和测试报告等数字资产。订单集合表示为$O = \{O_1, \ldots, O_i, \ldots, O_n\}$. 首先由软件需求方$R_i$ 创建软件订单$O_i = (D_i, T_i, Req_i, AC_i)$,

订单内容包括要合作的软件开发方D_i, 测评机构T_i, 软件需求Req_i以及软件验收标准AC_i. D_i 依据需求完成软件开发并在系统上完成产权登记。软件资产通常是以制品而非源代码形式进行交易，软件开发方在交付之前也可能进行代码混淆来防止源代码泄露，因此本系统对软件制品而非源代码进行产权认证。开发方提供制品后，T_i 接受委托对软件进行测试，产出测试报告TR_i 为软件资产的价值背书并为软件验收提供依据。最后，R_i 依据TR_i 和AC_i 对软件进行验收。验收通过后公布验收结果AR_i，完成交付过程。

表 3.3: 软件产权登记用例描述

描述项	说明
参与者	软件开发方，目标是在系统中完成软件资产所有权归属认证
触发条件	软件开发方完成了开发过程，需要为软件登记所有权归属
前置条件	软件开发方提供完整的软件制品，并且下载了系统提供的文件采样工具
后置条件	无
正常流程	1.用户登录软件采样工具。 2.使用文件采样工具生成软件资产所有权证书。 3.登陆资产管理系统，系统验证用户身份 4.选择软件产权认证，输入软件名称、描述、和自定义属性 5.上传软件资产所有权证书并确认 6.系统在区块链上登记所有权归属信息
扩展流程	1a.用户名或密码错误 1.登录失败，拒绝生成证书 3a.用户身份没有通过验证 1.系统提示错误并拒绝产权登记

结合上述用例图，给出关键用例的详细用例描述。表 3.3 给出了软件产权登记的用例描述。登记软件产权需要记录用户身份信息，用户使用系统时需要提供密钥对$Key_{user_i} = (K^p_{user_i}, K^s_{user_i})$用于身份认证，其中公钥$K^p_{user_i}$ 保存在系统中，私钥$K^s_{user_i}$由用户自行保存。软件存在的形态多种多样，系统提供了一个文件采样工具为软件制品生成唯一可验证的证书$Cert_i = (Hash(S_i), Sign(S_i, K^s_{user_i}))$证书内容包含软件内容哈希$Hash(S_i)$ 和软件所有者对软件内容的数字签名$Sign(S_i, K^s_{user_i})$，$Cert_i$ 作为软件产权归属的凭证保存在区块链上，利用区块链技术保证其内容不可篡改，实现软件资产的及时确权。

为使用户能够验证软件制品与软件证书的一致性，以及验证软件产权归属，文件采样工具提供证书验证功能：

$$VerifyContent(S_i, Cert_i); \qquad VerifyOwner(S_i, Cert_i, K^p_{user_i})$$

$VerifyContent$计算S_i的哈希值是否与$Cert_i$中记录的哈希值相同，若通过验证则证明S_i的内容未经篡改，$VerifyOwner$使用用户公钥$K^p_{user_i}$和S_i验证$Cert_i$中的数字签名，若验证通过则证明该软件的所有者是$K^p_{user_i}$指代的用户。因此任何用户在获取S_i和$Cert_i$后，可以利用采样工具验证软件内容的真实性和软件产权的归属。考虑到软件会不断迭代的特性，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。

用户使用采样工具生存证书$Cert_i$后，需要登陆软件资产管理系统上传证书，由系统将其保存在O_i中写入区块链。采样工具和资产管理系统使用同一套用户身份信息，系统使用LDAP实现统一身份认证。为保证所有权证书在被写入区块链之前的安全性，系统要求上传证书的用户与证书中记录的用户相同。

表 3.4: 软件验收用例描述

描述项	说明
参与者	软件需求方
触发条件	软件需求方需要对采购的软件进行验收
前置条件	需求方给出验收标准，开发方给出软件制品，测评机构给出测试结果
后置条件	无
正常流程	1.用户从系统中下载软件测试报告和软件证书 2.对比测试报告和验收标准，给出验收结果上传系统 3.从软件开发方获取软件 4.验证软件、证书、测试报告一致性，确认数据未经篡改 5.完成软件交付过程，关闭订单
扩展流程	1a.用户选择自动验收服务 1.上传规范化的验收标准 2.系统自动验收，生成验收结果并存入区块链 3.用户查看验收结果 4a.软件没有通过验证 1.验收不通过，结束软件验收流程

表 3.4 给出了软件验收的用例描述。测评机构T_i完成对软件的测试后出具测试报告保存在O_i 中:

$$TR_i = (Hash(Cert_i), TestResult, Sign(TestResult, K^s_{user_j}))$$

TR_i包括被测软件的证书哈希值$Hash(Cert_i)$，测试结果数据$TestResult$以及测试人员对测试结果的数字且签名$Sign(TestResult, K^s_{user_j})$，此时$O_i = (D_i, T_i, Req_i, AC_i, Cert_i, TR_i)$, 且O_i中的内容对参与订单参与方R_i, D_i, T_i 可见。系统提供测试报告的验证功能$VerifyTR(TR_i, Cert_i)$，计算$Hash(Cert_i)$并与TR_i中存储的证书哈希对比，若结果一致则表示TR_i确实是针对$Cert_i$ 中的软件的测试结果。R_i 获取TR_i 后，可以根据TR_i 和AC_i 执行验收，在人工验收流程中，用户从系统中下载这些数据，完成人工验收，再把验收结果上传至系统。此外，系统提供了可选的自动验收服务，基于智能合约对软件进行自动验收，既保证验收过程公开透明，又起到提高软件验收效率、加快软件交付过程的作用。如果使用自动验收服务，需要保证TR_i 和AC_i 准符合系统规定的格式，系统通过智能合约获取这些数据并自动地对比生成测试结果AR_i 并保存在O_i 中。最终的订单数据为:

$$O_i = (D_i, T_i, Req_i, AC_i, Cert_i, TR_i, AR_i)$$

订单管理用例包含软件开发方对订单的创建、更新、查询和删除操作，与之类似，测试数据管理用例包含了对测试数据的创建、更新、查询和删除。它们流程较为简单，此处略去详细的用例描述。

3.1.3　功能需求

经过系统涉众分析和用例分析，将软件资产管理系统的功能需求分为订单管理、软件证书管理、测试数据管理和自动验收四部分。

订单管理。软件需求方创建订单，上传软件需求和验收标准，并指定合作的软件开发方和软件测评机构。用户可以查看于自己相关的订单列表和订单详情，能够获取需求文档、验收标准等文件。

软件证书管理。软件开发方能够在订单中上传软件证书，软件证书由配套的文件采样工具生成。软件测试机构和软件需求方能够获取软件证书，查看证书的变更历史。

测试数据管理。软件测评机构能够在订单中获取软件，对其进行测试后上传该软件的测试报告。测试报告允许进行更新。软件需求方可以查询测试报告及其变更历史。

自动验收。根据软件验收标准和测试报告完成自动验收。软件测评机构向系统上传格式化的测评结果，系统在收到测评结果后进入自动验收流程，并生成验收结果写入区块链。订单的参与者均可查看该订单相关的验收结果，验收结果包含验收标准中各项指标的达标情况。

3.1.4　非功能需求

软件资产管理系统为了保证软件资产价值在互联网上进行可靠的传递与交换，需要对一些关键数据如软件、测试数据、软件验收标准进行可靠的存储。系统最为关注的非功能需求包括安全性、数据一致性和数据可靠性。

系统安全性。系统存储了软件生命周期中产生的核心资产，需要避免因遭受攻击或权限管理不当造成的数据泄露或丢失。

数据一致性。系统承担了软件产权认证的需求，软件产权证书作为软件所有权归属的凭证，需要在参与系统的所有组织间形成共识。软件交易过程的各个参与方对软件生命周期中的重要产物（软件、软件所有权证明、测试报告、验收标准、验收结果）需要保持相同的视图，对交易过程中的关键活动如验收过程需要达成共识。

数据可靠性。软件产权证书必须保证真实可靠、不可篡改或伪造，才能够保证证书的公信力和有效性。软件需求方希望获取真实的测试结果和被测软件，软件开发方希望看到真实的验收标准和验收结果，仅仅依靠各方之间的信任保证数据真实可靠是不现实的，需要从技术层面保证数据的可靠性，使得各个参与方在互不信任的情况下也能够完成软件交易流程。

区块链技术可以很好地满足上述需求。在区块链中，数据分布式地存储在所有节点上，避免了传统的中心化存储模式由于数据集中而容易受到攻击的缺点，大大提高了系统数据的安全性。基于共识机制完成数据写入和更新，保证所有节点上的数据一致性，利用智能合约实现自动验收，保证所有参与方就验收过程达成共识。利用哈希、Merkle 树和时间戳保证数据可验证、不可篡改或伪造，在无信任的环境下保证数据的安全可靠。因此本文基于区块链技术实现软件资产交易子系统以支撑软件资产管理系统的功能需求和非功能需求。

3.2　软件资产交易子系统需求分析

3.2.1　软件资产交易流程

通过软件资产管理系统的需求分析得出软件资产交易流程如图 3.2 所示。为实现软件资产的可靠交易，支持软件资产管理系统的功能需求和非功能需求，

本文基于区块链技术实现件资产交易子系统，提供可靠、可信、不可篡改的软件产权数据和交易数据存储方案，以及基于智能合约的自动化验收功能，为软件资产交易过程提供可靠保障。通过集成该子系统，期望能够保证软件资产的所有权和软件测试数据可追溯、不可篡改，利用智能合约进行自动验收，加速供需匹配，提高软件资产变现效率，为软件资产的确权和流通提供可靠保障。

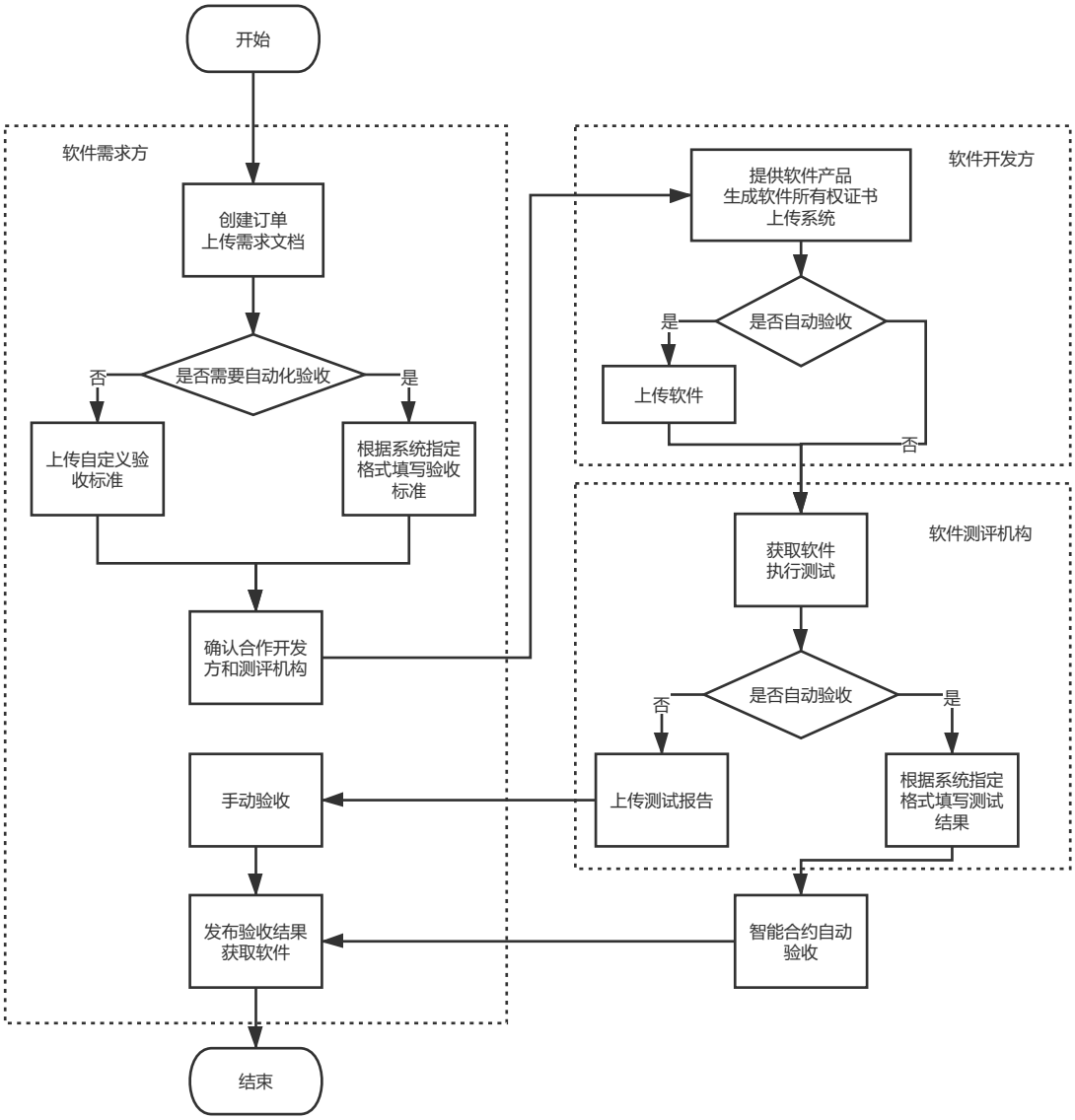

图 3.2: 软件资产交易工作流图

3.2.2　系统特性与约束

软件资产交易子系统的需求包括数据存储和基于智能合约的自动验收。数据存储包括软件资产证明、订单、测试数据、验收标准、验收结果等可以直接

写入区块链的数据，和软件证书、软件、测试报告、需求文档等体积较大的不便直接写入区块链的文件。本文将需要写入区块链的各类数据抽象为“文档”，把数据存储的需求分为链上文档存储和链外文件存储两部分，总结软件资产交易子系统特性如下：1）对区块链上的文档进行增加、删除、修改、查看操作，并保存变更记录。2）查询区块链上文档的变更记录，包括一份文档历史上所有的添加、修改、删除操作，操作时间、操作内容和操作者。3）存储大体积的文件，并保证文件内容不可变。4）根据软件验收标准和测试报告完成自动验收。

软件资产交易子系统需要与其它子系统合作完成软件资产管理系统的功能，根据外部组件的技术特点和组件之间约定的通信方式，得出系统约束有：1）区块链上的文档以key-value结构存储。2）系统将运行与Linux 服务器上。3）系统以多节点的形式部署在多台物理服务器上，节点间能够互相通信，保持数据同步。4）使用区块链客户端的系统基于Spring 框架开发，区块链客户端需要以Spring依赖的形式提供。

3.2.3　功能需求

为了支持软件资产管理系统的功能需求，软件资产交易子系统需要实现链上存储和链外存储相结合的数据存储功能和基于智能合约的自动验收功能。

链上文档存储。1）用户发起新建文档请求时，系统将Java 对象形式的文档对象转换为key-value形式并将其存储在区块链上。2）用户发起更新文档请求时，系统为该文档创建一个新版本，记录更新后的文档数据。3）用户发起删除文档请求时，系统为该文档创建一个内容为空的新版本，并标记为已删除。4）用户查询单个文档内容时，系统返回文档最新版本的内容，如果文档的最新版本标记为已删除则返回空。5）用户检索文档列表时，系统返回满足检索条件的所有文档，其中不包含标记为已删除的文档。6）用户查询文档变更记录时，系统返回该文档的所有历史版本以及变更时间。7）系统需要存储的实体包括订单、软件、测试报告、验收标准及验收结果。

链外文件存储。用户上传文件时，系统存储文件并为文件生成一个不可变的摘要。可以使用该文件摘要获取原始文件。

自动验收。用户调用自动验收接口时，系统通过对比软件的验收标准和测试报告来验证软件的各项指标是否通过验收，自动生成验收结果文档并写入区块链。用户查询验收结果时，系统返回验收结果文档，包含各项验收指标是否达标的信息。

3.2.4　非功能需求

安全性。系统有一个默认的网络管理员账号，只有该账号可以设置合法的网络节点。系统不允许未经授权的节点加入网络。每个节点有默认的节点管理员，只有该账号可以对用户授权。系统不允许未经授权的用户访问数据。只有在参与区块链网络的所有节点间达成共识的交易才能够完成数据写入。

可用性。某一节点写入的数据应在所有节点上可以访问，包括链上数据和链外数据。网络中少于1/3的节点出现故障时，网络仍可正常运行。

可维护性。系统需要存储新类型文档时能够在一人日内完成修改和部署。

3.3　软件资产交易子系统总体设计

3.3.1　系统架构

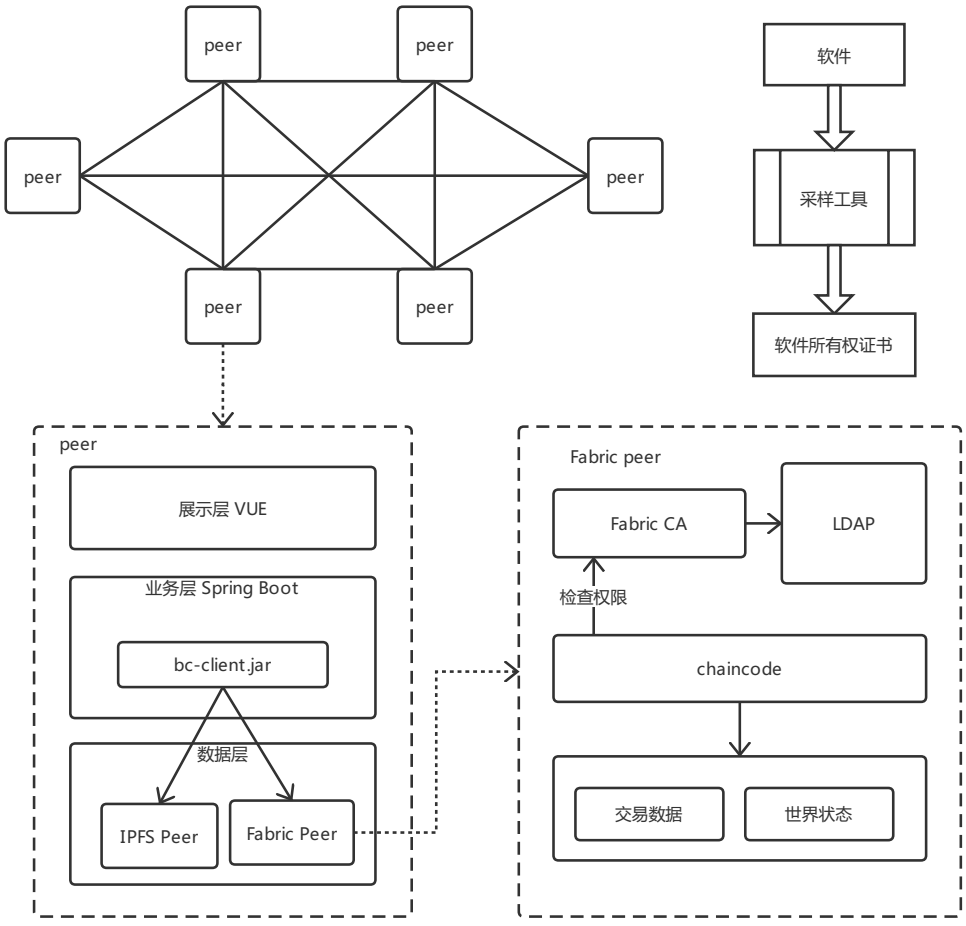

图 3.3: 系统架构图

软件资产交易子系统整体采用P2P的网络拓扑结构。区块链网络由参与维护网络的对等节点组成。每一个对等节点都包含了完整的系统组件，各节点之间通过区块链底层设施进行数据交换和同步，依照共识机制完成数据更新。如图 3.3 所示，每一个对等节点由展示层，业务层和数据层组成。展示层负责图形界面的展示和交互；业务层负责处理复杂的业务逻辑，并调用区块链客户端bc-client对区块链上的数据进行读写；数据层包含IPFS节点和Fabric 节点，分别用来管理链外数据和链上数据。其中Fabric节点由Fabric CA，chaincode和数据账本组成，Fabric CA负责身份认证和权限管理；chaincode是运行在区块链上的代码，用来处理数据和更新账本和编写智能合约；数据账本则保存了所有的区块数据和交易数据，Fabric 还提供了一个称为“世界状态”的数据库，用来保存所有数据的最新状态，以提高数据查询效率。

此外，系统提供文件采样工具为软件生成所有权证书。采样工具与系统都依靠LDAP 完成用户身份认证，共享一套用户信息，但与系统没有直接交互。

3.3.2　4+1视图

本文以Philippe Kruchten提出的“4+1”视图 [42]方法给出系统的架构设计。系统的用例和场景已在 3.1.2 节中给出。本节从逻辑视角、开发视角、进程视角、和物理部署视角四个方面给出系统架构模型，描述系统的静态和动态模型以及软件到硬件的映射。

(1) 逻辑视图

图 3.4展示了软件资产交易子系统的逻辑设计方案。依据系统的功能需求，从逻辑上将系统提供的服务分为四个部分。链外文件存储模块负责存储文件和生成摘要，链上文档存储模块完成区块链上存储的文档的增删改查功能，自动验收模块调用智能合约进行自动验收，用户认证模块处理用户的身份认证。

参考分层的架构风格，将系统分为应用层、合约层和数据层。应用层负责上层业务的实现，并为软件资产管理后台系统提供功能全面、简单易用的接口。合约层包括与区块链交互所需的智能合约。数据层用于存储各类持久化数据，根据不同的需要，分为基于Fabric的区块链存储、基于IPFS 的链外文件存储和基于LDAP 的用户数据存储三个部分。

逻辑视角描述了系统的抽象结构，要将其转化为一个物理上的系统仍需要将逻辑设计从开发、进程和部署三个角度进行实现。

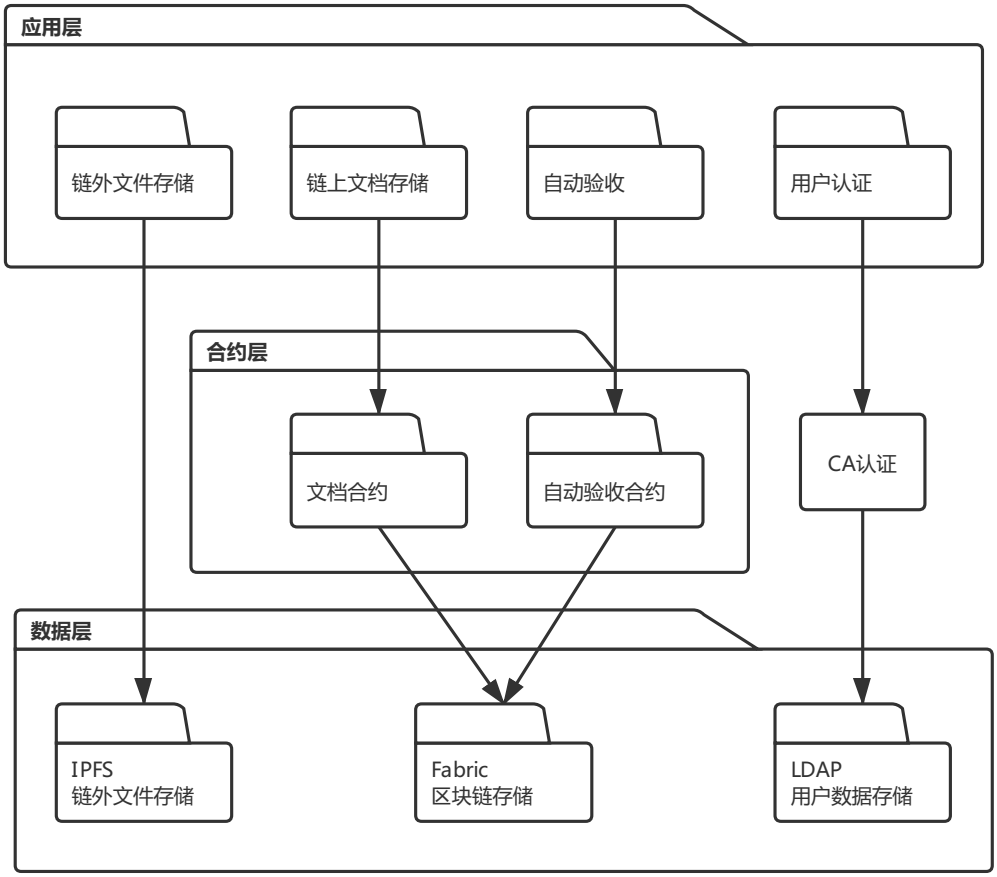

图 3.4: 软件资产交易子系统的逻辑设计方案

(2) 开发视图

在软件资产交易子系统的众多组件中，应用层的区块链客户端和合约层的chaincode 有较为复杂的开发工作。在软件资产管理系统中，业务层通过区块链客户端访问区块链，可以说，区块链客户端承担了业务层和Fabric之间的桥梁作用，也是区块链模块的重要组成部分。chaincode 是运行在区块链上的代码，Fabric 为chaincode提供了一套丰富的接口，可以编写脚本对账本数据进行读写操作和编写智能合约。本节从开发视角介绍这两部分的架构设计。

本项目中，业务层选用SpringBoot框架实现，因此提供给业务层使用的区块链客户端也是基于Spring框架，以jar 包形式提供。Hyperledger Fabric中的chaincode允许使用Java，Node.js，和Go 语言编写，考虑到Fabric本身使用Go语言编写，Go语言编写chaincode的文档比较全面，并且效率更高，与Fabric 框架集成也较为顺畅，本项目中使用Go 语言编写chaincode。

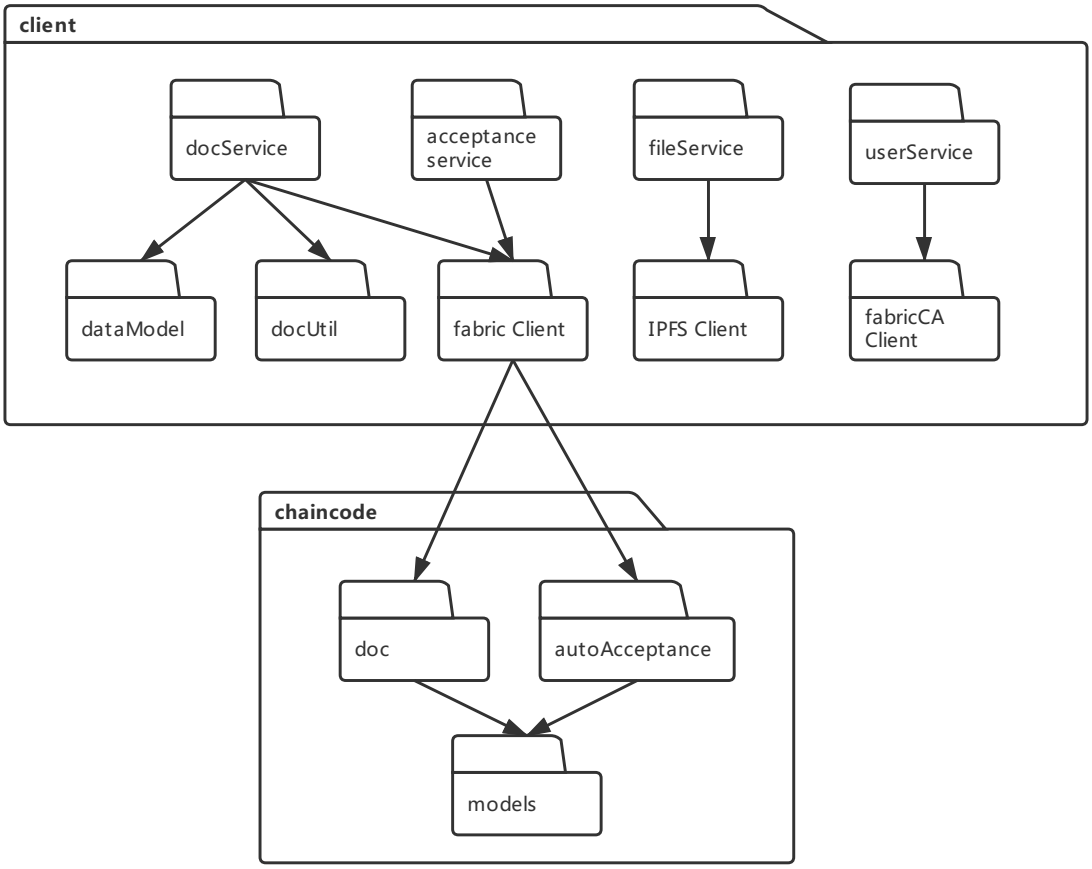

图 3.5: 软件资产交易子系统的开发包图

图 3.5 给出了软件资产交易子系统的开发视图。系统分为区块链客户端和chaincode两部分完成开发。

客户端中，将逻辑设计中的每个模块转化为一个开发包。docService链上文档存储服务需要处理各种不同类型文档的增删改查，需要依赖包含不同类型文档类的dataModel 包和能够处理各类文档的格式转换的docUtil包。由于需要与区块链交互，docService和负责自动验收的acceptanceService 都需要依赖fabricClient包来访问区块链数据和调用智能合约。链外文件存储服务fileService 依赖IPFS完成链外文件存储。用户认证服务userService依赖fabric-CAClient包完成用户数字证书认证。

区块链客户端对合约层chaincode的调用通过fabricClient 进行，chaincode包含了处理文档的智能合约doc和处理自动验收的智能合约autoAcceptance。二者都依赖于定义文档和数据格式的models包。

(3) 进程视图

图 3.6 展示了软件资产交易子系统的运行时进程。客户端进程与chaicode进程、Fabric CA进程和IPFS进程协作完成业务功能。用户的认证流程由client进程发起，通过HTTP协议通知Fabric CA进程，Fabric CA 进程接到请求后，进一步通知LDAP 进程验证用户身份，为用户生产证书并返回client进程。后台系统需要访问区块链账本数据时，需要通过client进程发起请求，通过HTTP协议通知Fabric chaincode进程，chaincode 生成交易后通知Fabric Peer完成区块链共识过程，并完成数据写入。类似的，文件相关的服务由client进程与IPFS进程协作完成。此外，不同节点间的Fabric进程会互相通信进行区块同步，IPFS 进程互相通信进行文件同步。图中所有进程都在Docker容器中运行，这些容器可以在一台机器上，也可以分散在多台机器上。

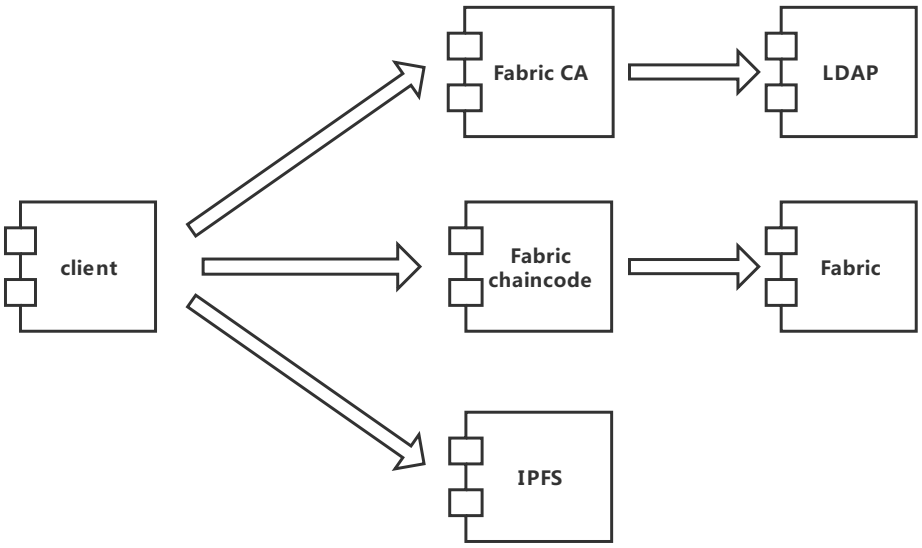

图 3.6: 软件资产交易子系统的进程图

(4) 物理部署视图

图 3.7 展示了软件资产交易子系统的物理部署方式。其中，区块链客户端构建放在客户端节点上，客户端节点需提供Linux操作系统和JDK1.8环境。Fabric相关组件部署在Fabric节点上，包括chaincode构件、Fabric peer 构件和Fabric CA构件。用户数据节点部署了LDAP 构件。各构件运行在Docker容器中，所以在服务器资源足够的情况下，也可以将所有构件部署在同一节点上。

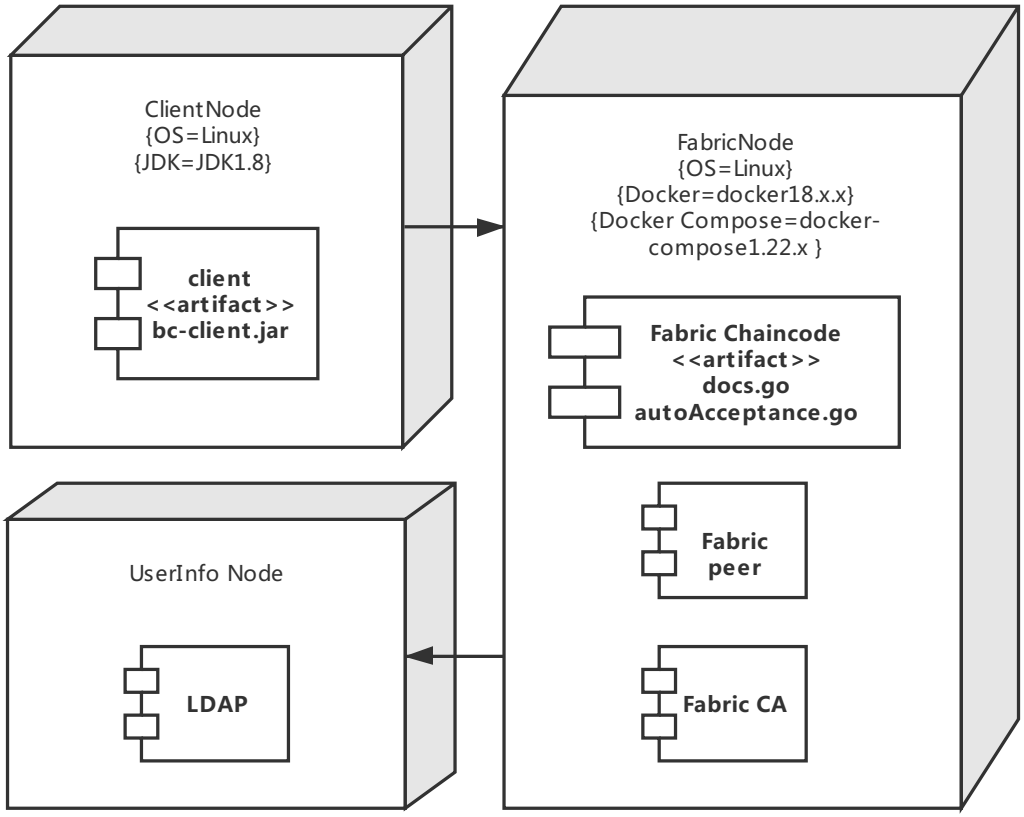

图 3.7: 软件资产交易子系统的部署图

3.3.3　持久化对象设计

软件资产交易子系统依据系统需求定义了若干持久化对象，包括软件、订单、测试结果、验收标准和验收结果。

软件Software 保存了软件的名称、创建者、软件证书的哈希等。记录软件信息是为了声明软件的知识产权归属，能够保证软件和其创作者之间的所属关系不可篡改。表 3.5 列举了Software对象的关键字段。

表 3.5: Software的主要字段

字段	字段类型	描述
name	String	软件名称
user	SystemUser	产权归属
softwareCertHash	String	软件证书哈希
contributorMap	Map<String, Object>	软件贡献者
attriMap	Map<String, Object>	软件自定义属性

订单Order记录了软件资产的需求、生产、测试、交付过程。表 3.6 列举了订单对象的主要字段。包含订单的基本信息、订单关联的机构（软件开发方、测试机构）、订单流转过程中产生的关键文件（软件、需求文档、验收标准、测试数据）的摘要。订单记录在区块链上，保证了数据的可追溯性和不可篡改性，使得软件生产和交付过程公开透明，有据可查，避免不必要的纠纷。

表 3.6: Order的主要字段

字段	字段类型	描述
name	String	订单名称
user	SystemUser	订单创建者，即软件需求方
deliveryTime	Long	软件交付时间
suppliers	List<SystemUser>	甲方指定的软件供应商列表
testOrganizations	List<SystemUser>	甲方指定的软件测试机构列表
acceptanceCriteriaHash	String	软件验收标准的哈希值
requirementHash	String	软件需求的哈希值
status	Integer	订单状态
softwareHashes	List<String>	软件的哈希值列表
reportHashes	List<String>	测试报告的哈希值列表
testTaskHash	List<String>	测试任务的哈希值列表

此外，为保证自动验收过程是公开、透明、可验证的，系统基于智能合约实现自动验收功能。智能合约分布式地运行在所有节点上，其输出的交易同样需要经过共识过程才能写入区块链，因此必须保证智能合约在所有节点上的运行结果是一致的。为了消除智能合约执行过程中的不确定性，智能合约中需要用到的数据必须来自区块链上已确认的交易。

系统实现一个智能合约通过对比软件验收标准和测试结果的各项指标，得出最终的验收结果并持久化，该智能合约所需的输入数据：验收标准、测试结果和输出数据：验收结果都需要保存在区块链上。表 3.7 列举了以上三个自动验收相关对象的关键字段。AcceptanceCriteria验收标准中存储一个Map，保存了软件的各项指标（如性能指标：qps，兼容性指标：设备兼容百分比等）应处于的范围，TestResult 测试结果中存储一个对应的Map，保存该软件在各项指标上的实际表现。对比产生的AutoAcceptResult验收结果中同样保存一个Map，标记每项指标是否通过验收。

表 3.7: 自动验收相关对象的主要字段

对象类型	字段	字段类型	描述
AcceptanceCriteria	orderId	String	订单ID
	acceptanceMap	Map<String, Object>	验收标准Map
TestResult	orderId	String	订单ID
	testResultMap	Map<String, String>	验收标准Map
	softwareCertHash	String	软件证书哈希
	testTaskHash	String	测试任务哈希
AutoAcceptResult	orderId	String	订单ID
	acceptanceMap	Map<String, String>	验收结果Map
	softwareCertHash	String	软件证书哈希
	testTaskHash	String	测试任务哈希

3.4　软件资产交易子系统核心模块设计

上文已经从不同视角分析了软件资产交易子系统的体系结构，并划分了功能模块。本节结合业务需求对系统的核心业务模块：链上文档存储模块和自动验收模块进行细化，进一步完善系统的概要设计。

3.4.1　链上文档存储模块

如上节所述，软件资产交易子系统需要持久化存储5种类型的文档。为满足业务层的调用需求，在区块链客户端中，这5中类型的文档是以Java类的形式定义、分别存取的。而在区块链账本中，使用的是Fabric提供的key-value数据库，以字符串形式存储文档。此外，访问Fabric的区块链数据需要通过调用chaincode完成，chaincode在被调用时只能接受字符串数组格式的参数。

图 3.8 以Software类型的文档为例，展示了文档读写过程中在各个阶段所需要的数据格式。因此，在处理文档数据的读写过程中，需要提供一种灵活的方式将复杂的嵌套Java 对象转换为字符串数组，并能将Fabric中存储的字符串数据还原为Java对象。

为满足这一条件，我们将链上文档存储模块拆分成交易处理和交易数据映射两个子模块。交易处理子模块的职责是与chaincode交互，调用chaincode的相关函数生成数据写入和更新交易，读取交易数据。根据系统需求提供条件查询、分页查询和历史数据查询的功能。交易数据映射子模块的职责是在客户端Java对象与其它数据格式之间建立映射关系，自动地将Java 对象转化为

字符串数组类型的参数格式，并能在获取区块链账本数据后，将JSON字符串还原为Java 对象并组织成列表对象List<T>、分页对象Page<T>、历史数据对象History<T>。

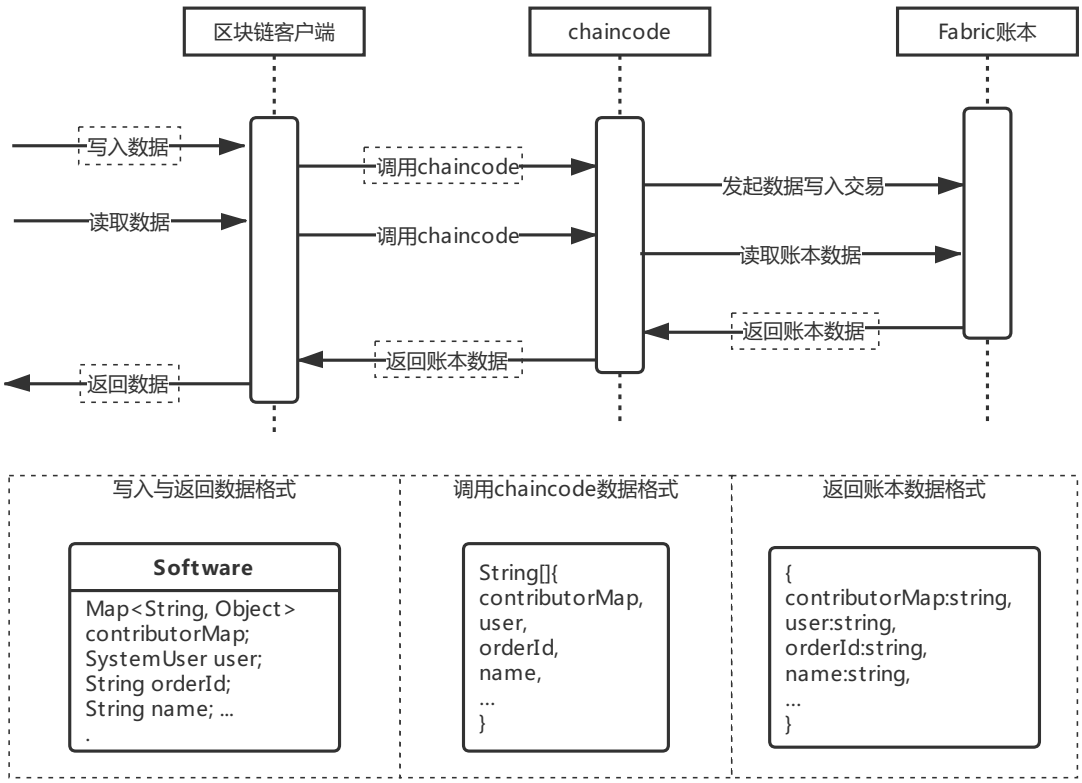

图 3.8: 文档存储流程中的数据格式变化

3.4.2　自动验收模块

要实现自动验收，必须定义规范化的验收标准和测试结果的数据格式，以便智能合约解析并对比数据。本文定义验收标准AC_i 和测试结果TR_i的数据格式如下：

$$AC_i = \left\{ \begin{array}{c} AC_i^1 = (name, operator, value) \\ AC_i^2 = (name, operator, value) \\ \vdots \\ AC_i^n = (name, operator, value) \end{array} \right\} \qquad TR_i = \left\{ \begin{array}{c} TR_i^1 = (name, value) \\ TR_i^2 = (name, value) \\ \vdots \\ TR_i^n = (name, value) \end{array} \right\}$$

一个验收标准AC_i包含多个条目，每个条目AC_i^j定义为一个三元组。其中name表示该条验收标准的名称，operator 为操作符，value表示数值。例

如，“项目的单元测试覆盖率应达到90%”这条验收标准可用三元组表示为$AC_i^j = (unitTestCoverage, \geq, 90\%)$. 与$AC_i$对应的测试结果$TR_i$与$AC_i$ 包含相同数量的条目，每个条目定义为一个二元组，例如，“项目的单元测试覆盖率为95%”可表示为$TR_i^j = (unitTestCoverage, 95\%)$.

类似的，验收结果定义为AR定义为：

$$AR_i = \left\{ \begin{array}{c} AR_i^1 = (name, result) \\ AR_i^2 = (name, result) \\ \vdots \\ AR_i^n = (name, result) \end{array} \right\}$$

进行自动验收时定义函数$Verify(C_i^j, TR_i^j)$ 验证该条目是否通过验收。例如TR_i^j中 unitTestCoverage 的实际值95% 满足了验收标准中定义的≥ 90%的条件，则这项验收指标的验收结果为通过，若不满足条件即为不通过。将验收标准中每个条目的验证结果记录在AR_i中存入区块链。

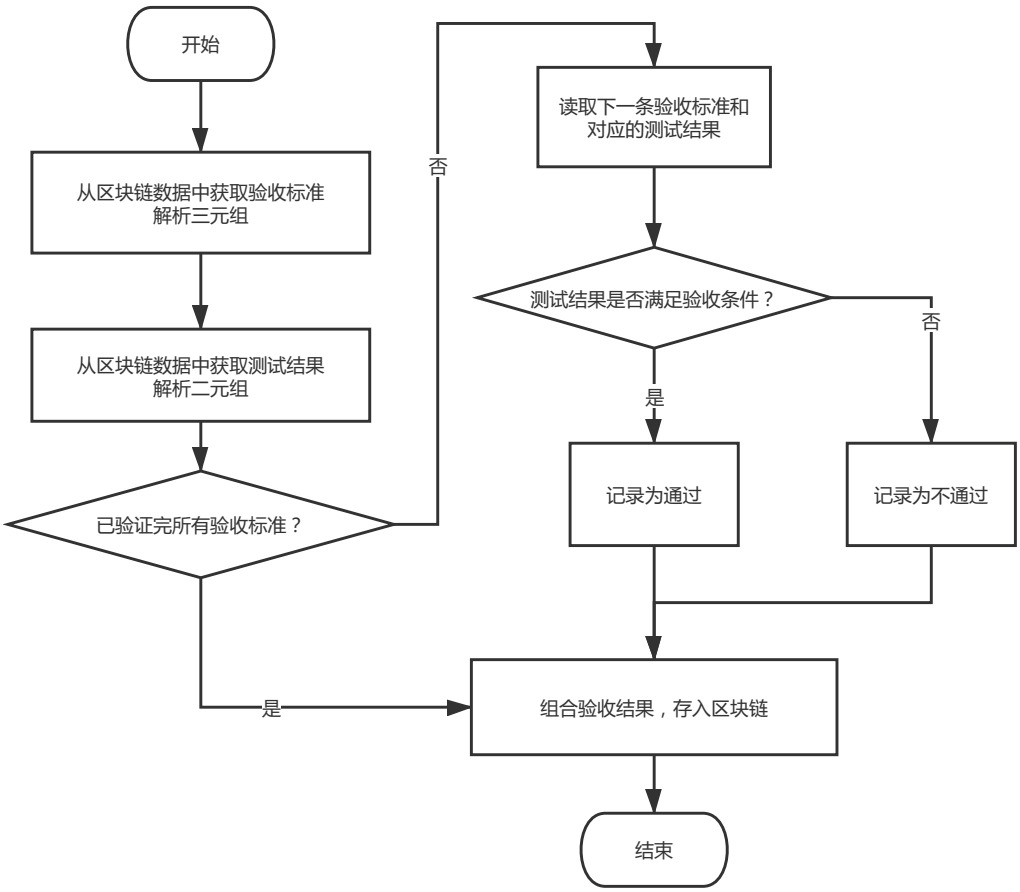

图 3.9: 自动验收流程

图 3.9 展示了智能合约执行自动验收过程的流程。首先从区块链数据中读取验收标准和测试结果数据，然后遍历所有验收标准，依次验证相应的测试结果是否满足验收条件、记录验收结果，最后组合验收结果并存入区块链中，完成自动验收流程。

3.5　本章小结

本章首先给出了软件资产管理系统的总体介绍，以涉众分析表和用例图形式对系统需求进行了分析，并给出了主要用例的用例述表。基于系统需求得出了软件资产交易子系统的功能需求和非功能需求，对其进行了概要设计，从逻辑视角、开发视角、进程视角、部署视角描述了系统架构，并给出了系统的持久化对象设计，将系统拆分为链上文档存储模块、自动验收模块、链外文件存储模块、用户认证模块，并对其中较为复杂的模块进行了进一步分析。

第四章　软件资产交易子系统的详细设计与实现

4.1　交易处理模块

4.1.1　交易处理模块的详细设计

交易处理模块是链上文档存储模块的一个子模块，需要区块链客户端中的DocService和Fabric中的chaincode协同完成，依据文档的读写请求生成数据写入和数据更新交易，提供对区块链账本数据的增删改查功能。其中查询服务包括根据键值查询单个文档、条件查询、分页查询和历史数据查询。

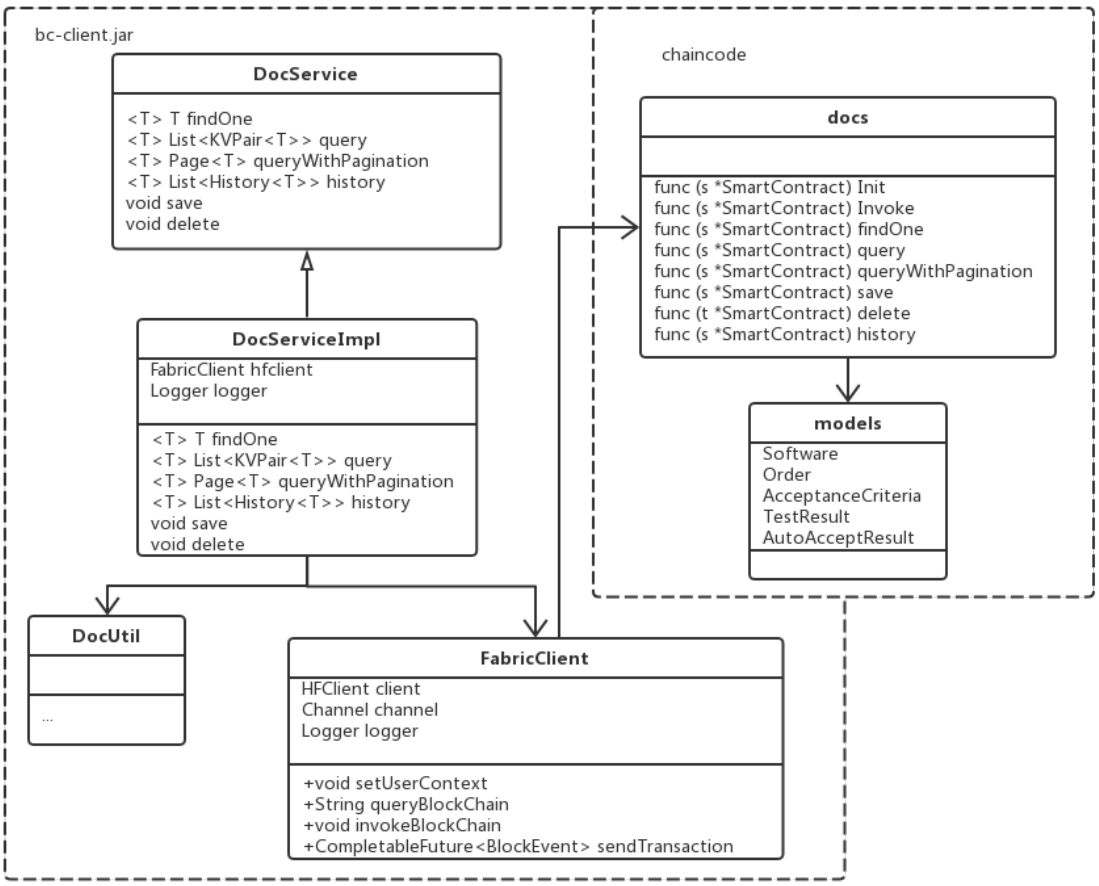

图 4.1: 交易处理模块类图

交易处理相关的类图如图 4.1 所示。DocServiceImpl实现DocService接口对外提供文档的读写服务。DocServiceImpl依赖FabricClient与chaincode进行交

互，依赖DocUtil对文档进行转码映射（DocUtil相关内容将在下一小节介绍）。FabricClient通过Fabric提供的Java SDK调用chaincode，完成对区块链账本的读写。Fabric支持使用CouchDB作为账本数据库，因此可以利用CouchDB 的条件查询语句对区块链数据进行查询。query方法接受一个字符串类型的查询条件语句作为参数，调用chaincode进行复杂条件查询。

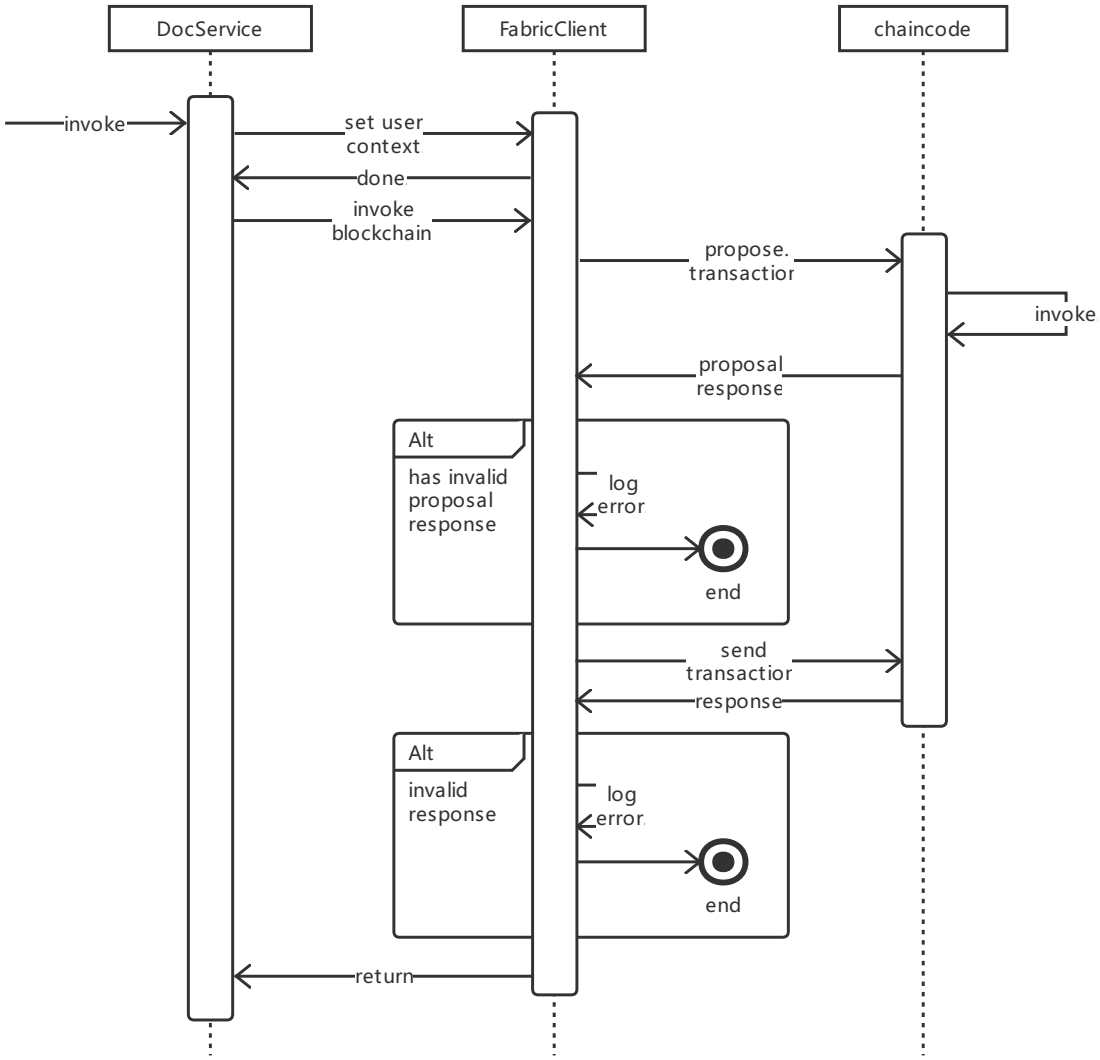

图 4.2: 交易处理模块invoke时序图

对区块链账本中数据的操作分为invoke和query两种，invoke操作是指需要向区块链账本中写入数据的操作。包括文档存储、更新和删除。图 4.2 展示了invoke操作的时序图。用户发起invoke 操作时，由DocService调用FabricClient的invokeBlockchain 方法，接着FabricClient根据具体的参数生成交易提案请求并发

送给chaincode。交易提案被广播给区块链网络中的所有节点，征求各个节点的意见，并返回交易提案的响应结果。若响应结果无效则过程终止，若响应有效则说明节点对交易提案达成了共识，可以进一步发送交易请求，将新的交易数据写入区块链。

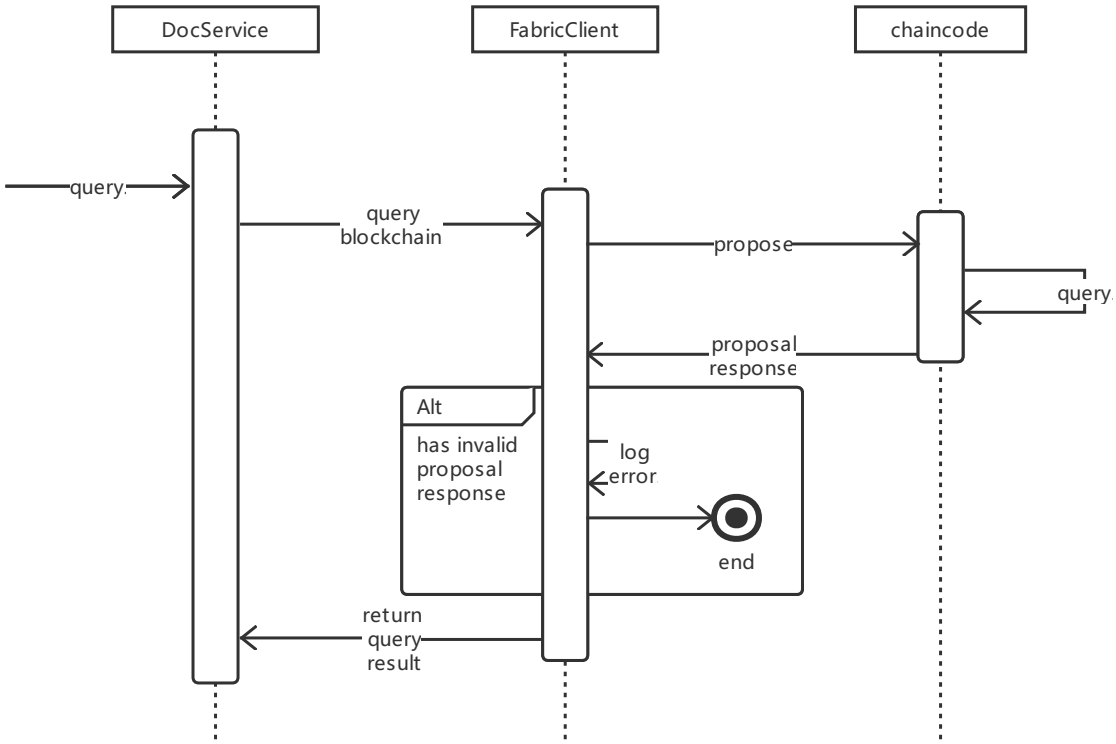

图 4.3: 交易处理模块query时序图

如图 4.3 所示，query操作是指单纯的数据查询操作。由于无需写入数据，所以也不需要节点间的共识过程。当用户发起query请求时，DocService调用FabricClient的queryBlockChain 方法，FabricClient根据传入的参数生成查询请求，调用chaincode获取查询结果并返回。

4.1.2　交易处理模块的实现

图 4.4 给出了DocServiceImpl类的部分实现。DocServiceImpl类实现 DocService 接口，通过FabricClient调用chaincode实现键值查询、分页查询、历史版本查询和插入、更新、删除功能。定义了Page，History等数据对象，并将chaincode返回的字符串形式的结果转换为Java对象返回，为业务层的开发提供了便利。

```
public class DocServiceImpl implements DocService {
    public <T> Page<T> queryWithPagination(Type type, String queryString, int
pagesize, String bookmark) {
        String result = hfClient.queryBlockChain
(ChainCode.docs.getName(),Function.queryWithPagination.getName(),new
String[]{queryString, String.valueOf(pagesize),bookmark});
        return DocUtil.string2Page(type,result);
    }
    public <T> List<History<T>> history(Type type, String key) {
        String result = hfClient.queryBlockChain
(ChainCode.docs.getName(),Function.history.getName(),new String[]{key});
        return DocUtil.string2HistoryList(type, result);
    }
    public void save(String key, Doc doc) {
        String[] param = DocUtil.doc2Array(doc);
        param[0] = key;
        hfClient.invokeBlockChain(ChainCode.docs.getName(),
Function.save.getName(),param);
    }
}
```

图 4.4: DocServiceImpl部分实现

DocServiceImpl所依赖的FabricClient以query和invoke两种形式与区块链数据交互。invoke操作由invokeBlockchain方法实现。执行invoke操作时需要向Fabric发送交易提案并等待共识结果，由于此过程耗时较长，采用异步调用的方式，将发送交易请求过程包装为一个异步调用方法sendTransaction，该方法返回CompletableFuture对象。invokeBlockchain方法调用sendTransaction发送交易请求，从CompletableFuture中获取结果时设置超时时间为60秒，避免应用长时间等待。图 4.5 展示了sendTransaction方法的部分代码，首先构造交易提案请求（Transaction Proposal Request，tpr），tpr中包含需要调用的chaincode的名称、chaincode中要调用的方法名称以及调用参数。组装完成后向通道发起交易提案请求，等待提案返回结果ProposalResponse。区块链网络中的每一个节点都会返回一个ProposalResponse，获取所有ProposalResponse的集合后检查它们的有效性，如果全部有效，则进一步发送交易请求，完成交易写入。如果发现无效的ProposalResponse，则记录失效信息，中止交易过程并抛出运行时异常。

FabricClient中的invoke和query操作都做了必要的错误处理并详细记录错误日志，方便对业务执行情况进行分析。

```
public CompletableFuture<BlockEvent.TransactionEvent> sendTransaction(…){
   TransactionProposalRequest tpr = client.newTransactionProposalRequest();
   //组装 tpr，略
   Collection<ProposalResponse> responses = channel.sendTransactionProposal(tpr);
   List<ProposalResponse> invalid =
      responses.stream().filter(ProposalResponse::isInvalid).collect(Collectors.toList());
   if (!invalid.isEmpty()) {
      invalid.forEach( /*log error*/ );
      throw new RuntimeException("invalid response(s) found");
   }
   return channel.sendTransaction(responses);
}
```

图 4.5: FabricClient的invoke实现

query操作由queryBlockchain方法实现，如图 4.6 所示。query过程是单纯的数据读取操作，不需要向区块链中写入交易，因此无需共识过程，直接向Fabric发送查询请求并等待查询结果。

```
public String queryBlockChain(String chaincode, String function, String[] args) {
      QueryByChaincodeRequest qpr = client.newQueryProposalRequest();
      //组装 qpr，略
      Collection<ProposalResponse> res = channel.queryByChaincode(qpr);
      for (ProposalResponse pres : res) {
            if(pres.isInvalid()){
                  logger.error("tx "+pres.getTransactionID()+"invalid"+pres.getMessage());
            }else{
                  String stringResponse =
                              new String(pres.getChaincodeActionResponsePayload());
                  logger.info("query result:"+stringResponse);
                  return stringResponse;
            }
      }
      return null;
}
```

图 4.6: FabricClient的query实现

对区块链数据的访问最终要由FabricClient调用chaincode 完成。本文编写了名为docs的chaincode 处理各类文档的增删改查功能。根据Fabric的规则，编写chaincode需要实现Init和Invoke两个方法。Init方法在初始化以及更新一个chaincode时被调用，完成必要的初始化逻辑。Invoke方法是chaincode的调用入口，FabricClient的invoke和query操作最终都是通过调用chaincode中的Invoke方法来完成。如图 4.7 所示，在docs的Invoke方法中，根据第一个参数判断要调用的是什么方法，接着去调用该方法的具体实现。

```
func (s *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {
    function, args := APIstub.GetFunctionAndParameters()
    if function == "findOne" {
        return s.findOne(APIstub, args)
    } else if function == "save" {
        return s.save(APIstub, args)
    } else if function == "queryWithPagination" {
        return s.queryWithPagination(APIstub, args)
    }else if function == "history" {
        return s.history(APIstub, args)
    }……//其它方法
    return shim.Error("Invalid Smart Contract function name.")
}
```

图 4.7: docs的invoke方法实现

以查询文档的方法query为例，在DocService中调用query 方法时， DocServiceImpl 调用FabricClient 的queryBlockchain 方法，传入参数为{”docs”,”query”,-”{”selector”:{”type”:”1”,”user”:”{”name”:”user1”}”}}”}，然后FabricClient 调用名为docs 的 chaincode ，传入这组参数。接到调用请求后，docs中的Invoke方法开始执行，首先判断第一个参数内容，当参数为“query”时，继续调用docs中的query方法，传入后续参数即查询条件{”selector”:{”type”:”1”,”user”:”{”name”:-”user1”}”}}。在docs的query方法内部，利用Fabric 提供的API，根据查询条件从couchDB中查询满足条件的文档列表，将查询结果包装为JSON格式字符串并返回。图 4.8 展示了query方法的部分代码。

```
func (s *SmartContract) query(stub shim.ChaincodeStubInterface, args []string)
sc.Response {
    if len(args) != 1 {
        return shim.Error("Incorrect number of arguments. Expecting 1")
    }
    queryString := args[0]
    queryResults, err := getQueryResultForQueryString(stub, queryString)
    if err != nil {
        return shim.Error(err.Error())
    }
    return shim.Success(queryResults)
}
```

图 4.8: docs的query方法实现

4.2　交易数据映射模块

交易数据映射模块与交易处理模块协同完成链上文档存储功能。在交易处理模块完成交易生成、数据写入和交易读取功能的基础上，交易数据映射模块在区块链客户端和chaincode的之间建立数据映射关系，利用Java的反射技术，封装繁琐的数据格式转换逻辑，使区块链客户端与区块链账本交互时无需关心数据格式问题。

4.2.1　交易数据映射模块的详细设计

如图 4.9 所示，交易数据映射模块的核心类是DocUtil，DocServiceImpl依赖DocUtil 将Java中的文档对象转换为chaincode能够识别的字符串数组，并将chaincode返回的字符串类型的结果转换为Java对象。DocUtil 的具体职责将Java对象中不同类型的字段都以合适的方式转换为字符串，并能从字符串中还原数据对象。考虑到不同的字段可能会需要不同的转换方式，有必要提供一种灵活的机制让用户定义对象的转码和解码方式。因此，本文定义了一个字段注解@Parse，和枚举类型ParseType，声明了3种转码方式供用户选择式，DocUtil 可根据注解指定的方式对Java对象进行相应的处理。

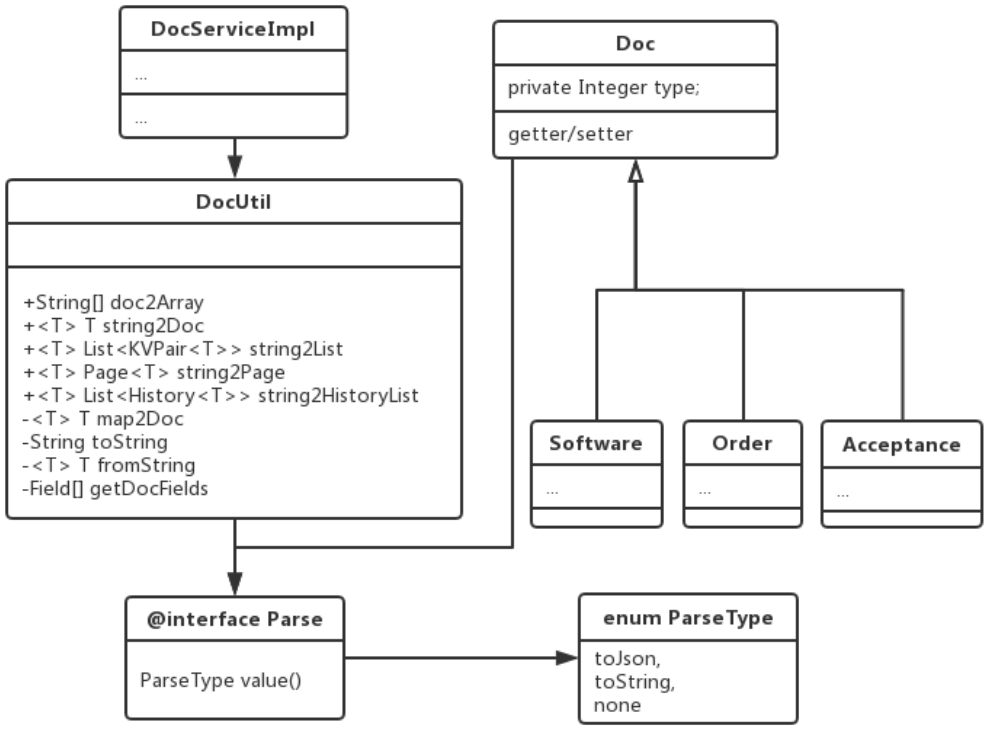

图 4.9: 交易数据映射模块类图

4.2.2　交易数据映射模块的实现

本文规定以Java对象形式表示文档实体时，所有字段必须为对象类型，不允许使用基本数据类型，这样可以方便统一的数据转换，免去繁琐的类型判断和对基本数据类型的特殊处理，提高代码的可读性和可维护性。

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Parse {
    ParseType value() default ParseType.none;
}
public enum ParseType {toJson, toString, none }
```

图 4.10: @Parse注解相关实现代码

如图 4.10 所示，我们定义了一个字段注解@Parse，接受一个ParseType类型的参数以指定转码方式。ParseType中定义了toJson，toString和none三种类型，其含义如下：

- **ParseType.toJson** toJson 类型表示在数据格式转换过程中该字段的Java对象将转换为JSON 字符串，还原Java对象时以JSON 格式解析字符串内容。这种转换方式依赖Google提供的Gson工具实现，Java对象转为字符串调用Gson的toJson方法，从字符串还原Java对象时调用Gson 的fromJson方法实现，适用于可被Gson解析的嵌套类型的字段。这种方式保证了对象转换为字符串后的可读性，将Java对象转换为通用的JSON 格式存储于区块链中存储，为之后可能的数据分析功能留出广泛的可能性。

- **ParseType.toString** toString 类型调用对象的toString方法将对象转换为字符串，还原Java 对象时以字符串作为参数调用该类型的构造方法。因此使用toString类型标记的字段类型必须提供形如Constructor(String s)的构造方法。Java中基本数据类型的包装类型如Integer，Boolean等都提供了这样的构造方法，因此这些类型的字段可以使用toString标记。此外，提供toString这种转换方式的重要原因是为了支持用户自定义数据转换方式。用户可以重写该类型的toString方法，并提供相应的构造方法来从字符串还原Java对象，实现特殊的数据映射方式。

- **ParseType.none** none 类型表示不做数据转换工作，直接赋值。该类型适用于本身即为String 类型并且不需要做特殊处理的字段。

```
private static String toString(ParseType parseType, Object value) {
   if (parseType==ParseType.none) {return (String)value; }
   else if (parseType==ParseType.toJson) {return gson.toJson(value) }
   else {return value.toString();}
}
private static <T> T fromString(Class T, ParseType parseType, String rawValue) {
   ……
}
```

图 4.11: 数据格式转换方法实现

如图4.11所示，DocUtil中定义了toString和fromString方法，依据ParseType完成数据转换功能。toString方法将Java对象转化为字符串，接受一个Java对象和转码方式ParseType作为参数，判断ParseType的类型，并按照上文的描述将对象按照该类型的规则转换为字符串。相应的，fromString方法从字符串中还原Java对象。它接受要还原的对象类，转码方式ParseType和字符串数据作为参数，根据ParseType的定义完成数据转换。

```
public static String[] doc2Array(Doc doc) {
    Field[] fields = getDocFields(doc.getClass());
    String[] param = new. String[fields.length+1];
    for(int i=0;i<fields.length;i++) {
       fields[i].setAccessible(true);
       Object fieldValue = fields[i].get(doc);
       ParseType parseType = fields[i].getAnnotation(Parse.class).value();
       param[i+1] = toString(parseType,fieldValue);
    }
    return param;
}
```

图 4.12: doc2Array方法实现

DocUtil中的doc2Array方法负责将文档对象由Java对象转换为chaincode可接受的字符串数组的形式。如图4.12所示，doc2Array方法中利用Java反射技术，获取该文档对象的所有字段，遍历这些字段，获取字段上的@Parse注解及以及注解中的参数ParseType，而后调用toString方法，根据不同的ParseType对字段进行不同的处理，将各字段转为字符串后的值拼接为字符串数组。

将字符串数据还原为文档对象的方法同样利用了Java的反射和范型技术。如图4.13所示，string2Doc方法的参数为需要转换为的Java对象的类型docType，和原始的字符串数据。chaincode以JSON字符串的形式返回文档，string2Doc方法首先对传入的JSON字符串进行初步解析，构造为JSON对象，以方便地获取某个字段的值。然后同样地，依据传入的docType获取该类型上的所有字段及其字段上的@Parse注解指定的转码方式，最后调用fromString方法将字符串还原为字段的类型，拼装完整的文档对象。

```
public static <T> T string2Doc(Type docType, String str){
    Class clazz = ((Class)docType);
    JsonObject rawDoc = new JsonParser().parse(str).getAsJsonObject();
    T doc = (T)clazz.newInstance();
    Field[] fields = getDocFields(clazz);
    for(Field field:fields) {
        field.setAccessible(true);
        Class fieldsType = field.getType();
        String rawFieldValue = rawDoc.get(field.getName()).getAsString();
        ParseType parseType = field.getAnnotation(Parse.class).value();
        field.set(doc, fromString(fieldsType,parseType, rawFieldValue));
    }
    return doc;
}
```

图 4.13: string2Doc方法实现

DocUtil的实现对用户来说是完全透明的，而且由于使用了Java 范型和反射技术，其转码功能与特定的文档对象类型无关。同样以Software类型的文档为例，只需将Software 中的每个字段标注@Parse注解指定转码类型，如图 4.14 所示，用户在使用DocService与区块链账本交互时将自动完成转码和解码过程，为业务层的开发带来了极大的便利。

```
public class Software extends Doc{
    @Parse(ParseType.toJson)
    private Map<String, Object> contributorMap;
    @Parse(ParseType.none)
    private String name;
    ......
}
```

图 4.14: @Parse使用实例

此外，为了支持文档服务模块中所需的列表查询、分页查询和历史数据查询的功能，DocUtil还提供了将字符串转换为列表List<Doc>，分页数据对象Page<Doc>和历史数据列表List<History<Doc>>的方法，都是基于以上两个核心方法实现，此处不再赘述。

4.3　自动验收模块

4.3.1　自动验收模块的详细设计

自动验收由区块链客户端和chiancode协作完成。区块链客户端负责对外提供自动验收接口，定义验收数据格式，包装验收数据结构，chaincode负责实现自动验收的具体过程。

自动验收模块的类图如图 4.15 所示。AcceptanceServiceImpl 实现了 AcceptanceService 接口对外提供自动验收服务。StandardObject 定义了 3.4.2 节中描述的验收标准三元组$AC_i^j = (name, operator, value)$ 的数据类型，其中操作符operator由枚举类型Operator 定义，目前包括相等、大于、大于等于、小于、小于等于五种操作，后续可以根据业务需求的变化进行扩展。 AcceptanceServiceImpl 调用chaincode 中的智能合约acceptance进行自动验收。

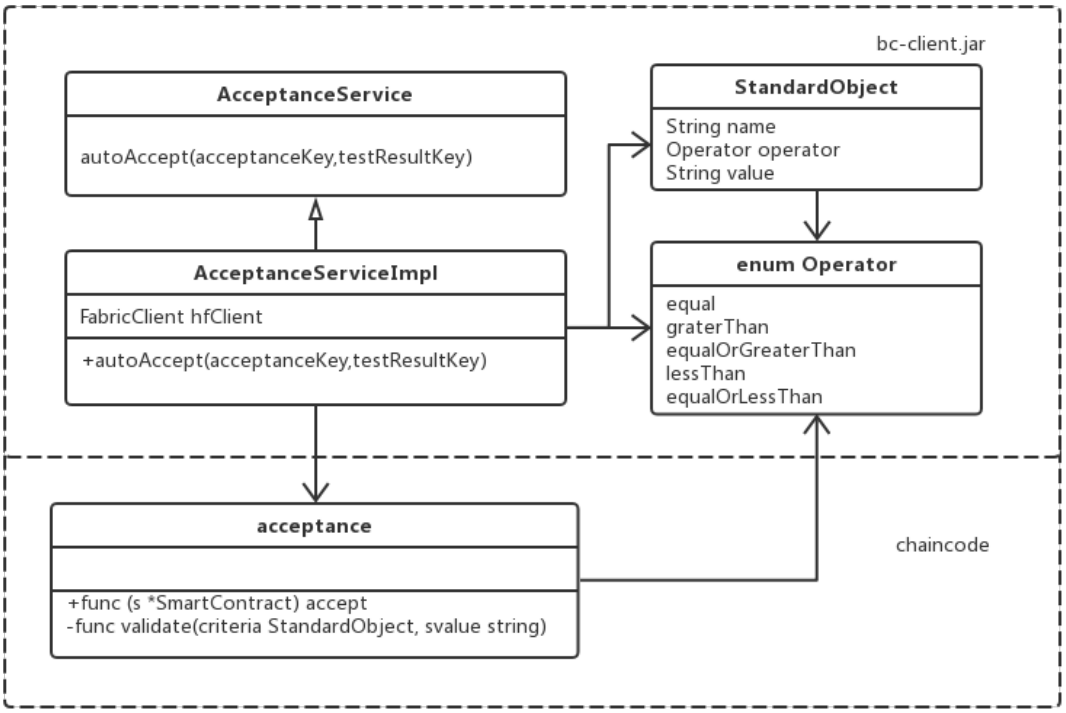

图 4.15: 自动验收模块类图

自动验收模块时序图如图 4.16所示。AcceptanceService向智能合约发起自动验收请求，智能合约首先访问区块链数据，获取需要验证的验收标准和测试结果。然后遍历所有验收标准，依次验证软件的测试结果是否达到了验收标准。最后将验收结果写入区块链并返回。

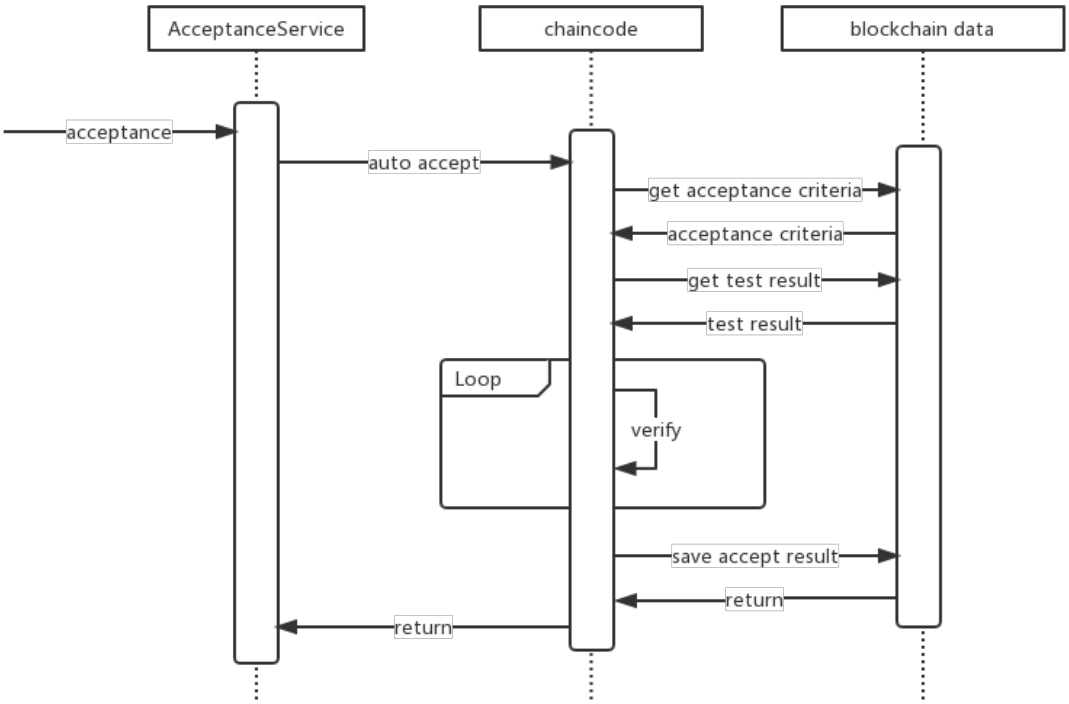

图 4.16: 自动验收模块时序图

4.3.2　自动验收模块的实现

图 4.17 展示了自动验收智能合约的部分实现。在智能合约中，使用Map结构保存AC_i, TR_i 和AR_i. 使用name作为键，其余字段作为值分别构造Map如下：

$$Map_{AC_i}[name_j] = AC_i^j(name_j, operator, value)$$

$$Map_{TR_i}[name_j] = TR_i^j(name_j, value) \qquad Map_{AR_i}[name_j] = AR_i^j(name_j, result)$$

autoAccept方法根据传入的验收标准文档的key值从区块链中获取验收标准文档，根据传入的测试结果文档key值获取测试结果文档，将其解析为Map形式。遍历验收标准Map中的所有键值对，根据键从测试结果Map中取出实际值，与验收标准进行对比，得出该条验收结果，写入验收结果的Map。遍历结束后组装验收结果数据并写入区块链。

```
func (s *SmartContract) autoAccept(APIstub shim.ChaincodeStubInterface, args []string)
sc.Response {
   //获取并解析验收标准
    acceptanceAsByte, _ := APIstub.GetState(args[1])
    var acceptance = &Acceptance{}
    json.Unmarshal(acceptanceAsByte, &acceptance)
    acceptanceStr := strings.Replace(acceptance.AcceptanceMap,"\\","",-1);
   //获取并解析测试结果
    …
    resultMap := make(map[string]string)
    for k,v := range acceptanceMap{
       resultMap[k] = strconv.FormatBool(validate(v,testResultMap[k]))
    }
    //组装验收结果并存入区块链
    autoAcceptResult.Result = string(resultMap)
    APIstub.PutState(args[0], json.Marshal(autoAcceptResult))
    return shim.Success(docAsBytes)
}
```

图 4.17: 自动验收智能合约的部分实现

4.4　链外文件存储模块

链外文件存储功能由区块链客户端调用IPFS实现。如图 4.18 所示。 File-ServiceImpl 持有一个IPFS 对象的引用，通过调用IPFS提供的Java SDK，完成链外文件存储功能。

该服务提供两个方法，1）uploadFile方法接受File对象，将其包装为IPFS指定的文件流，调用IPFS的add方法将文件流写入IPFS文件网络并获取它的加密哈希，最后将加密哈希转换成可读的base58字符串并返回。2）getFile方法接受的参数为base58编码的文件哈希字符串，首先将其转换为IPFS可识别的文件指针，调用IPFS 的cat方法获取文件内容的二进制流并返回。

```
public class FileServiceImpl implements FileService {
    private IPFS ipfs;
    public String uploadFile(File file) throws IOException {
        NamedStreamable.FileWrapper fileStream =
                                        new NamedStreamable.FileWrapper(file);
        MerkleNode addResult = ipfs.add(fileStream).get(0);
        logger.info("file "+file.getName()+" uploaded to IPFS");
        return addResult.hash.toBase58();
    }
    public byte[] getFile(String hash) throws IOException {
        Multihash filePointer = Multihash.fromBase58(hash);
        byte[] fileContents = ipfs.cat(filePointer);
        logger.info("downloaded file by hash:"+hash+" from IPFS");
        return fileContents;
    }
}
```

图 4.18: FileServiceImpl的实现

区块链网络中的每一个节点都持有一个IPFS节点负责存储文件，为了保证区块链网络中每一个节点都能访问到网络中所有的文件数据，需要在它们持有的IPFS节点之间搭建私有网络，实现文件共享。本文在测试环境中搭建了私有IPFS文网络如图 4.19 所示。

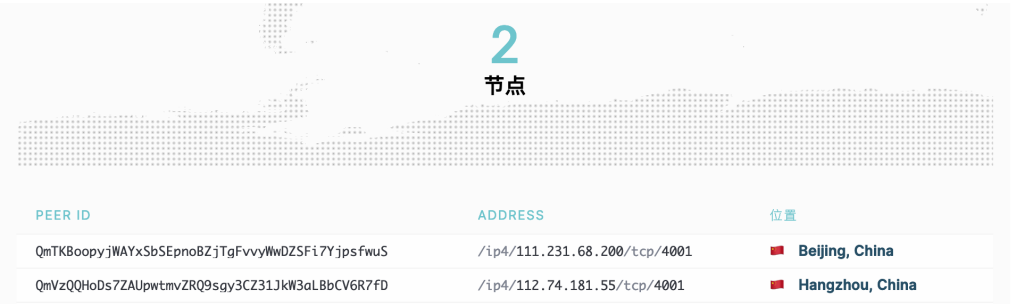

图 4.19: IPFS私有网络

4.5　用户认证模块

4.5.1　用户认证模块的详细设计

用户认证模块负责用户的登录认证和权限控制。由区块链客户端、Fabric CA 、 LDAP 三个组件协作完成。用户认证模块类图如图 4.20 所示。UserServiceImpl实现UserService接口对外提供用户服务。UserServiceImpl调用FabricCAClient与Fabric CA进行交互，完成用户的登录认证功能。

根据Fabric CA的机制，用户认证由 UserService.enroll 和 UserService.change-UserContext 两个方法协同完成。enroll方法利用用户名和密码认证用户身份，如果通过验证则返回包含用户证书的用户对象。调用changeUserContext 方法将用户上下文更改为当前的用户对象后，区块链客户端会使用该用户的身份与区块链账本进行交互。

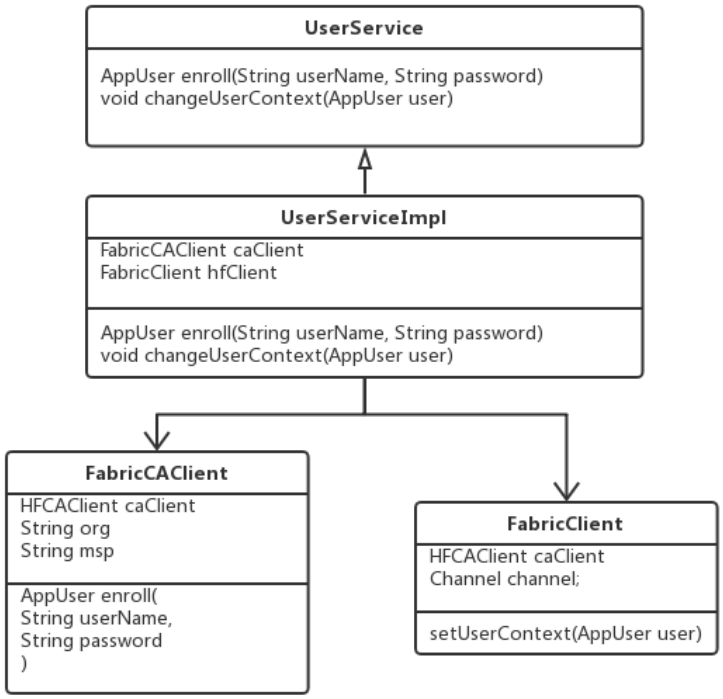

图 4.20: 用户服务模块类图

图 4.21 为用户登录认证和权限验证的时序图。系统使用LDAP存储用户数据，用户可以在LDAP上完成注册并配置相关属性。用户向系统发起登录请求时，UserService 调用FabricCAClient 进行登录验证，Fabric CA 调用LDAP 服务

验证用户名和密码，验证通过后获取用户及其属性，并将LDAP上配置的用户属性转换为Fabric CA 用户属性，为用户生成数字证书，返回包含证书的用户对象。若用户由于用户名密码错误等原因验证失败将无法生成证书。用户验证成功后可以调用FabricClient设置用户上下文，使用该用户的身份访问区块链数据。未设置用户上下文或使用伪造、无效证书设置上下文的访问请求将被Fabric拒绝。chaincode中可以指定只有具备某些属性的用户才能够访问特定数据，区块链客户端以某一用户身份调用chaincode时，chaincode首先检查该用户的属性值是否符合要求，不符合则拒绝访问，利用LDAP、Fabric CA和chaincode配合实现基于属性的访问控制。

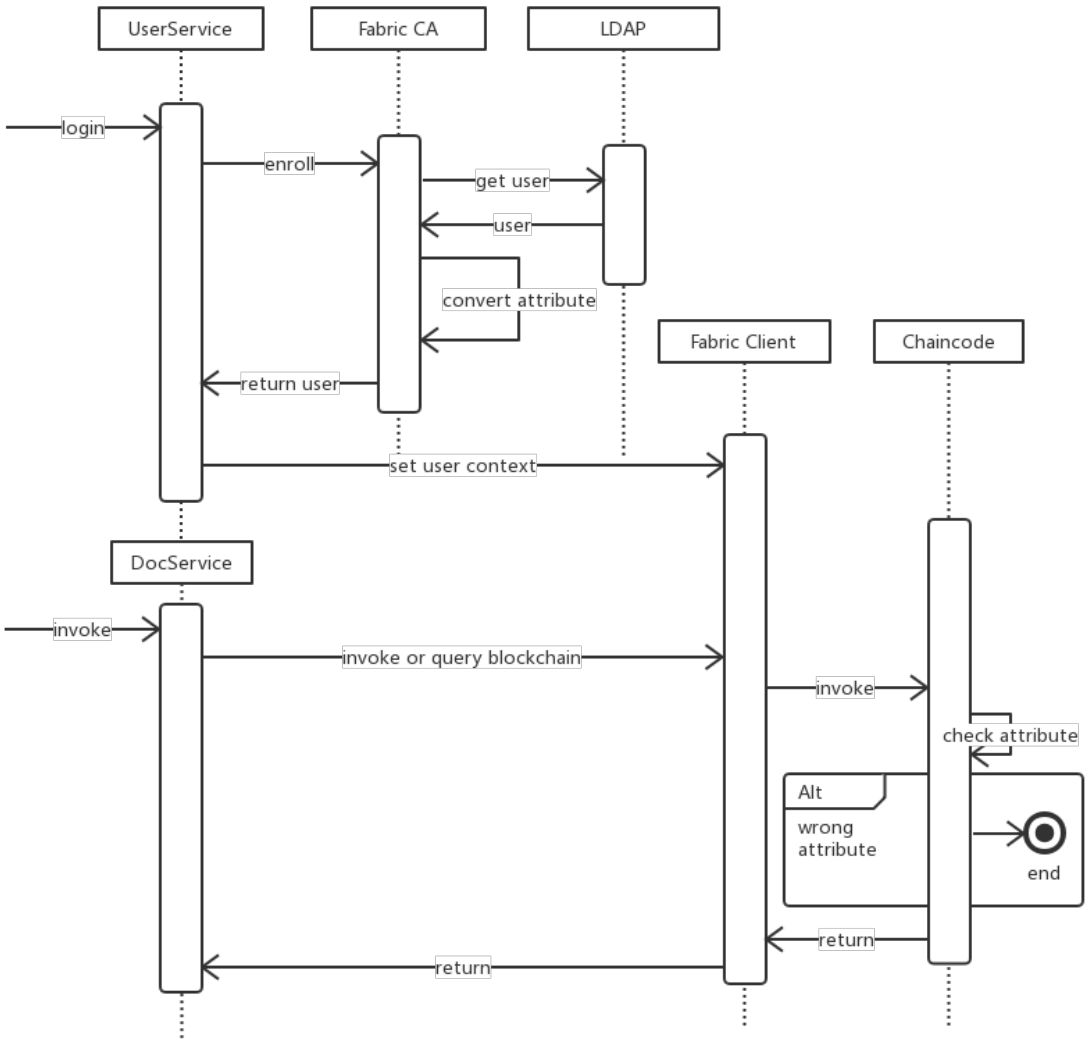

图 4.21: 用户认证模块时序图

4.5.2　用户认证模块的实现

如图 4.22所示，UserServiceImpl持有FabricCAClient和FabricClient的引用，用户发起登录请求时，首先调用FabricCAClient的login方法，验证用户身份并获取用户属性，然后调用FabricClient将用户上下文切换为当前登录的用户。此后客户端将以该用户的身份向区块链发送或查看交易，chaincode 就可以依据这些属性的值检查用户的权限。

```
public class UserServiceImpl implements UserService {
    private FabricCAClient caClient;
    private FabricClient hfClient;
    public void loginAndChangeContext(String userName, String password) {
        AppUser user = caClient.login(userName,password);
        hfClient.setUserContext(user);
    }
}
```

图 4.22: UserServiceImpl部分实现

图 4.23 展示了FabricCAClient的部分实现。用户认证前需构造一个 EnrollmentRequest 来说明需要获取的用户属性。login方法使用用户名、密码和 EnrollmentRequest 作为参数调用Fabric CA的enroll 方法，获取用户证书。

```
public class FabricCAClient {
    private EnrollmentRequest er= new EnrollmentRequest();
    {enrollmentRequest.addAttrReq("software"); …} //添加属性
    public AppUser login(String userName, String password) {
        Enrollment enrollment = caClient.enroll(userName, password,er);
        return new AppUser(userName, org, msp, enrollment);
    }
}
```

图 4.23: FabricCAClient部分实现

4.6　系统的配置与部署

软件资产交易子系统依赖的组件众多，一个节点包含的组件如图 4.24 所示。其中包括实现区块链共识机制、存储交易数据的Fabric Peer，存储账本数据当前状态的couchDB，运行在区块链上的代码chaincode，与区块链节点交互的命令行接口Fabric cli，一个可选的排序节点Fabric Orderer，和负责为用户颁发证书的Fabric CA。此外还有负责文件存储的IPFS节点，以及负责用户数据存储的LDAP 及其操作界面LDAP UI。

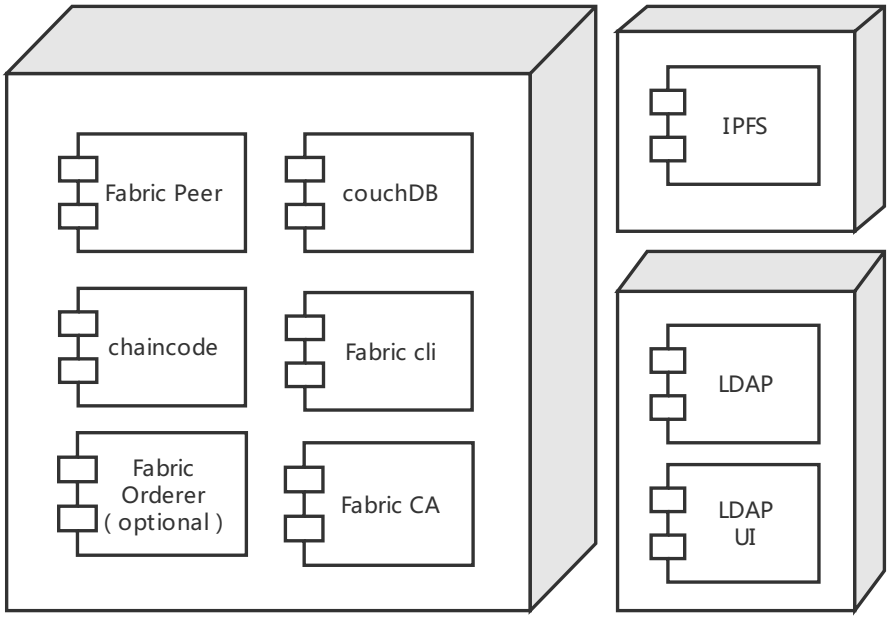

图 4.24: 软件资产交易子系统组件

部署这些组件并使其相互协作是一个极为繁琐的过程。本文为简化开发和部署流程，使用Docker技术为每一个组件提供单独的容器，并编写了docker-compose脚本，定义了组件的编排和协作方式，使用docker-compose实现一键部署。本项目提供了用于开发和测试的单机部署方案和用于实际生产环境的多机部署方案。下文将逐一介绍。

4.6.1　单机部署方案

为了方便系统的开发和测试，项目提供了单机部署脚本，包含一个最小化的可运行的区块链平台组件集合。图 4.25 给出了单节点部署时使用的docker-compose文件的部分内容。文件中定义了所需的服务及其环境变量、挂载目录和开放端口信息。

```
services:
  ca.example.com:
    image: hyperledger/fabric-ca
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_CERTFILE=/fabric-ca-server-config/cert.pem
      - FABRIC_CA_SERVER_CA_KEYFILE= /fabric-ca-server-config/...
    ports: "7054:7054"
    command: sh -c 'fabric-ca-server start'
    volumes:
      - ./crypto-config/org1.example.com/ca/:/etc/hyperledger/fabric-ca-server-config
      - ./fabric-ca-server-config.yaml: /fabric-ca-server-config.yaml
    container_name: ca.example.com
  //其它组件配置细节省略
  orderer.example.com: image: hyperledger/fabric-orderer ...
  peer0.org1.example.com: image: hyperledger/fabric-peer...
  couchdb: image: hyperledger/fabric-couchdb ...
  cli: image: hyperledger/fabric-tools ... …
```

图 4.25: 单机部署的docker-compose部分代码

以Fabric CA的配置为例，Fabric CA组件使用官方提供的hyperledger/fabric-ca镜像，设置环境变量指定Fabric CA 的工作目录和所需的证书文件以及密钥文件的位置。暴露7054端口用于与其它组件通信。指定容器的启动命令，在启动容器时启动Fabric CA服务，最后将它的配置文件和证书密钥文件挂载到容器外部，方便组件的配置和部署。

项目启动过程中，首先需要启动各组件，然后对区块链网络进行配置，安装和初始化chaincode。项目提供shell脚本完成自动化部署。脚本首先使用Docker Compose 工具启动组件，等待各组件启动完毕后创建区块链网络通道，并将Fabric Peer节点加入该通道。最后使用Fabric cli工具在节点上安装和初始化chaincode。

```
docker-compose -f docker-compose.yml up -d
    ca.example.com
    orderer.example.com
    peer0.org1.example.com couchdb
export FABRIC_START_TIMEOUT=10;
sleep ${FABRIC_START_TIMEOUT}
docker exec
    -e "CORE_PEER_LOCALMSPID=Org1MSP"
    -e "CORE_PEER_MSPCONFIGPATH=/etc/…/Admin@org1.example.com/msp"
    peer0.org1.example.com peer channel create
    -o orderer.example.com:7050 -c common -f /etc/hyperledger/configtx/channel.tx
docker exec
    -e "CORE_PEER_LOCALMSPID=Org1MSP"
    -e "CORE_PEER_MSPCONFIGPATH=/etc/…/Admin@org1.example.com/msp"
    peer0.org1.example.com peer channel join -b common.block
```

图 4.26: 单节点组件启动和网络配置部分代码

图 4.26 展示了脚本中启动组件和区块链网络配置的部分代码。首先使用docker-compose命令启动简单网络中的ca、orderer、peer、couchDB组件，组件启动一般在10秒内可以完成（1核2G以上配置），设置10秒等待时间，等待组件全部启动后在peer组件上运行peer channel create命令创建名为common的通道，接着运行peer channel join命令将节点加入common 通道。至此完成了单节点区块链网络的配置。

图 4.27 展示了脚本中安装和初始化chaincode的部分代码。首先启动Fabric的命令行工具组件cli，在cli中设置环境变量CORE_PEER_ADDRESS指向网络中的peer组件，使cli可以与peer节点交互。设置变量CC_SRC_PATH制定要安装的chaincode的源代码路径。在cli容器中运行peer chaincode install和peer chaincode instantiate命令完成chaincode的安装和初始化。至此单节点网络部署完成，可以使用区块链客户端访问。

```
docker-compose -f ./docker-compose.yml up -d cli
CC_SRC_PATH=github.com/docs
docker exec
    -e "CORE_PEER_LOCALMSPID=Org1MSP"
    -e "CORE_PEER_MSPCONFIGPATH=/opt/…/Admin@org1.example.com/msp"
    cli peer chaincode install -n docs -v 1.0 -p "$CC_SRC_PATH" -l golang
docker exec
    -e "CORE_PEER_LOCALMSPID=Org1MSP"
    -e "CORE_PEER_MSPCONFIGPATH=/opt/…/Admin@org1.example.com/msp"
    cli peer chaincode instantiate
    -o orderer.example.com:7050 -C common -n docs -l golang -v 1.0
    -c '{"Args":[""]}' -P "OR ('Org1MSP.member')"
```

图 4.27: 单节点chaincode安装和初始化部分代码

4.6.2　多机部署方案

实际生产环境中，会有多个机构参与组成区块链网络，需要在每个机构中部署一个节点。节点分为两类，一类是包含排序节点Orderer的完全节点，一类是不包含Orderer的业务节点。区块链网络由至少一个完全节点和多个业务节点构成。以一个完全节点和一个业务节点的区块链网络部署过程为例，首先需要定义区块链网络的参与者，并为其生成必要的密钥和证书材料。Fabric提供了密钥生成工具cryptogen，可以根据configtx.yaml文件中定义的参与机构生成所需的密钥和证书。

图 4.28 展示了configtx.yaml 中对机构的定义。以OrdererOrg为例，首先制定组织名称OrdererOrg和生成的文件存放路径crypto-config/ordererOrganizations-/example.com/msp。然后定义组织中节点的读写权限策略。读策略中，以数字签名验证用户身份，验证规则为”OR(’OrdererMSP.member’)”，表示该组织中的任何一个成员节点都可以读取节点数据。写策略的规则与读策略相同。管理策略同样以数字签名验证身份，验证规则定义为”OR(’OrdererMSP.admin’)”，表示只有节点管理员允许使用节点的管理功能，如安装、升级和初始化chaincode、加入通道、退出通道等。

```
- &OrdererOrg
    Name: OrdererOrg
    MSPDir: crypto-config/ordererOrganizations/example.com/msp
    Policies:
        Readers:
            Type: Signature
            Rule: "OR('OrdererMSP.member')"
        Writers:
            Type: Signature
            Rule: "OR('OrdererMSP.member')"
        Admins:
            Type: Signature
            Rule: "OR('OrdererMSP.admin')"
```

图 4.28: configtx.yaml中的机构定义示例

生成必要的密钥和证书材料后可以开始构建区块链网络。根据实际业务场景，两个机构的节点部署在两台不同的物理服务器上，如表 4.1 所示。部署步骤为：1）在主机A上利用docker-compose启动Org1节点组件，添加主机B的IP地址。2）在主机B上利用docker-compose启动Org2节部组件，添加主机A的IP地址。3）使用主机A的cli组件执行网络的初始化脚本，建立区块链网络通道，将Org1和Org2节点加入该通道并在两个节点上分别安装和初始化chaincode。4）在Org1和Org2节点的IPFS组件之间构建私有的文件传输网络。为它们添加相同的密钥，并在种子节点列表中加入对方的地址。

表 4.1: 部署方案说明

机构	节点类型	包含组件	部署主机
Org1	完全节点	包含业务节点的全部组件以及Fabric Orderer和Fabric cli	主机A
Org2	业务节点	Fabric CA、Fabric Peer、couchDB、IPFS、LDAP	主机B

项目提供了shell脚本自动化地执行上述步骤，简化项目的部署流程。与单节点网络部署过程类似，关键步骤为使用cli组件连接到各节点，调用peer channel和peer chaincode命令完成区块链网络的配置和chaincode的安装和初始化。图4.29 展示了网络配置过程的部分代码。createChannel方法创建区块链网络通道并验证通道是否创建成功。joinChannelWithRetry方法将节点加入通道，需要在通道创建完成后执行。由于通道的创建需要的时间受网络情况、节点数量影响，有很大的不确定性，为保证部署脚本的健壮性，joinChannelWithRetry方法提供了超时重试机制。本文中依据经验设定等待时间为10秒，最大重试次数为3次。首先调用peer channel join 命令尝试加入通道，如果加入失败则等待10秒后重试。尝试3次后仍然失败则打印错误日志，退出脚本。

```
createChannel() {
    PEER=$1;ORG=$2;setGlobals $PEER $ORG;
    peer channel create -o orderer.example.com:7050 -c $CHANNEL_NAME
        -f ./channel-artifacts/channel.tx >&log.txt;
    res=$? verifyResult $res "Channel creation failed"
}
joinChannelWithRetry() {
    PEER=$1;ORG=$2;setGlobals $PEER $ORG;
    peer channel join -b $CHANNEL_NAME.block >&log.txt; res=$?
    if [ $res -ne 0 -a $COUNTER -lt $MAX_RETRY ]; then
        COUNTER=$(expr $COUNTER + 1)
        echo "peer${PEER}.org${ORG} failed to join the channel,
        Retry after $DELAY seconds" sleep $DELAY
        joinChannelWithRetry $PEER $ORG
    else COUNTER=1 fi
    verifyResult $res "After $MAX_RETRY attempts,
    peer${PEER}.org${ORG} has failed to join channel '$CHANNEL_NAME' "
}
```

图 4.29: 多节点网络配置部分代码

chaincode的安装和初始化实现如图 4.30 所示，installChaincode方法调用peer chaincode install 和peer chaincode instantiate命令在节点上安装和初始化chaincode 。根据Hyperledger Fabric的机制，网络中的每一个节点都需要执行安装命令，而初始化命令在一个通道上只需执行一次。

```
installChaincode() {
    PEER=$1; ORG=$2; setGlobals $PEER $ORG
    peer chaincode install
        -n docs -l ${LANGUAGE} -p ${CC_SRC_PATH} >&log.txt
    peer chaincode instantiate
        -o orderer.example.com:7050
        -C $CHANNEL_NAME -n docs -l ${LANGUAGE}
        -c '{"Args":[""]}' -P "OR ('Org1MSP.peer','Org2MSP.peer')" >&log.txt; res=$?
    verifyResult $res "Chaincode installation on peer${PEER}.org${ORG} has failed"
}
```

图 4.30: 多节点chaincode安装和初始化部分代码

4.7　本章小结

本章主要描述了交易处理模块、交易数据映射模块、自动验收模块、链外文件存储模块和用户认证模块的详细设计与实现，通过类图和时序图建立模块的静态模型和动态模型，展示了关键代码，并给出了软件资产交易子系统的自动化部署方案。

第五章　软件资产交易子系统的测试与分析

5.1　系统测试

5.1.1　测试环境

表 5.1 展示了软件资产交易子系统的测试环境，其中用到两台装配了Docker和Docker Compose 的Linux 服务器，主机A为腾讯云服务器，单核2G配置，部署Org1节点。主机B为阿里云服务器，单核2G配置，部署Org2节点。Org1节点为包含orderer 组件的完全节点，需要额外开放7050端口用于orderer的通信。本节利用 4.6.2 节中介绍的自动化部署脚本搭建双节点网络。

表 5.1: 测试环境说明

机构	节点类型	主机环境	开放端口
Org1	完全节点	主机A 腾讯云服务器单核2G Ubuntu16.04.10 Docker18.06.1 Docker Compose1.22.0 Fabric 1.3.0	7054,7051,5001,7050
Org2	业务节点	主机B 阿里云服务器单核2G Ubuntu16.04.4 Docker18.06.1 Docker Compose1.23.1 Fabric 1.3.0	7054,7051,5001

5.1.2　功能测试

本节根据第三章中的需求分析为软件资产交易子系统设计测试用例，分模块进行功能测试，验证子系统能否满足业务需求，验证区块链网络的联通性。本节内容包括自动化测试用例的设计、编写、执行和对测试结果的总结。通过自动化测试和持续集成方法时刻保持系统功能的正确性和稳定性，减少开发过

程中引入的错误。软件资产交易子系统通过区块链客户端访问，新建Spring项目模拟后台系统引入客户端jar 包，使用JUnit和Spring Test编写自动化测试用例，调用客户端中提供的接口测试系统功能。本节就每个重要的功能模块给出典型的测试用例，旨在说明测试范围。根据系统的设计与实现，系统从功能角度划分为链上文档存储模块、自动验收模块、链外文件存储模块和用户认证模块。接下来分模块设计测试用例并给出测试结果。

(1) 链上文档存储模块

链上文档存储模块承担了对文档数据增删改查的职责，根据数据写入请求生成交易，将文档的变更记录在区块链中。本节对链上文档存储模块的每一个功能接口设计一个测试用例，如表 5.2 所示。其中测试用例doc.save测试文档保存接口，文档保存接口承担了文档的写入和更新功能，保存时如果文档key值不存在则新建文档，若key值存在则更新。保存成功后的预期结果是文档写入成功并返回数据写入交易的ID，可以使用查询接口查询到该文档的最新状态。测试用例doc.net用于测试区块链网络的联通性，验证区块链节点间的数据同步情况，检查区块链网络能否正常运行。

表 5.2: 链上文档存储模块测试用例

ID	说明	输入	预期输出	结果
doc.save	写入文档	文档对象、文档ID	写入成功，返回交易ID	通过
doc.findOne	查找文档	文档ID	文档对象	通过
doc.query	搜索文档	查询条件	文档对象列表	通过
doc.history	查询文档变更记录	文档ID	文档变更记录	通过
doc.delete	删除文档	文档ID	删除成功	通过
doc.net	文档在节点间的同步	在Org1节点上传文档	在Org2节点能够查询该文档	通过

执行测试用例后观察区块链节点日志验证测试是否通过。以用例doc.query为例，查询type 为1，用户名为user1的所有文档。节点输出日志如图 5.1 所示。可见系统依据查询条件返回了正确的查询结果集，测试通过。其它用例同样以查询日志的方法验证，此处不再赘述。

- getQueryResultForQueryString queryString:
{”selector”:{”type”:”1”,”user”:”{”name”:”user1”}”}}
- getQueryResultForQueryString queryResult:
[{”key”:”7ea1d688-8cd8-4197-8b16-438c71d80135”,
”doc”:{”type”:”1”,”user”:”{”name”:”user1”}”,”name”:”order1”,...

图 5.1: 文档查询日志

(2) 自动验收模块

执行自动验收的前置条件是，已上传标准化的验收标准，已上传标准化的测试结果。自动验收模块测试用例如表 5.3 所示。用例autoaccept.exec测试自动验收的执行，给出验收标准和测试结果的ID，调用chaincode 实现的智能合约完成自动验收，预期结果为生成正确的验收结果并写入区块链，返回数据写入交易的ID。用例autoaccept.get 调用文档查询接口查询生成的验收结果内容，判断内容是否符合预期。

表 5.3: 自动验收模块测试用例

ID	说明	输入	预期输出	结果
autoaccept.exec-.pass	执行自动验收	验收标准ID，满足验收标准的测试结果ID	生成验收结果并写入区块链	通过
autoaccept.exec-.notpass	执行自动验收	验收标准ID，不满足验收标准的测试结果ID	生成验收结果并写入区块链	通过
autoaccept.get	查看验收结果	验收结果ID	返回验收结果	通过

执行用例前需要构造软件验收标准和软件测试结果作为测试数据。将软件验收标准设置为兼容性指标在80%以上，性能指标在1000 qps以上，软件测试结果设置为兼容性指标90%，性能指标900 qps。根据以上测试数据，预期输出的验收结果应为兼容性指标达标，性能指标不达标。测试数据的标准表示以及执行用例后生成的验收结果的内容如图 5.2 所示，发现交易已完成，并且验收结果为{compatible=true, performance=false}, 即兼容性指标验收通过，性能指标验收不通过，符合预期输出。

Software acceptance criteria:

[{”name”:”compatible”,” operator”:”>=”,”80”},

{”name”: ”performance”,” operator”:”>=”,”1000”}]

Software test result:

[{”name”:”compatible”,”90”},{”name”:”perfo rmance”,”900”}]

Auto acceptance result

Transacion tx:

6353afff9ae30031318b12fd1c98e817476d33d459e438bbc0b1b0b3b0a completed.

query result={compatible=true, performance=false}, orderId=o1,

softwareHash=12fd1c98e817476d33d459ebc0b, testResultId=a1…}

图 5.2: 自动验收用例执行情况

(3) 链外文件存储模块

链外文件存储模块的功能包括文件上传与下载，测试用例如表如表 5.4 所示。文件服务模块与IPFS进行交互，将文件存储于IPFS 节点上。用例file.net 是为了验证IPFS私有网络中各节点之间的联通性，保证区块链网络中的任何一个节点使用相同的文件哈希都应该能访问到相同的文件。

表 5.4: 链外文件存储模块测试用例

ID	说明	输入	预期输出	结果
file.upload	上传文件	文件	上传成功，返回文件哈希	通过
file.download	下载文件	文件哈希	返回文件二进制流	通过
file.net	文件在节点间的同步	在Org1节点上传文件	在Org2节点获取到该文件	通过

上传内容为“this is a test file”的文本文件，查看日志发现上传文件生成的哈希值为QmRLucMa2nv8rYifPhX9hnmueE5p7uXA6vSkCzkDibUvHs，在IPFS节点上使用ipfs cat命令使用哈希值查找该文件输出如下图 5.3 所示。可见IPFS节点已正确存储了该文件，测试通过。

```
ipfs cat QmRLucMa2nv8rYifPhX9hnmueE5p7uXA6vSkCzkDibUvHs
this is a test file
```

图 5.3: IPFS查找文件内容

(4) 用户认证模块

用户认证模块测试用例设计如表 5.5 所示。用户认证模块与 Fabric CA 和 LDAP 协作，用户使用LDAP 上配置的用户名和密码登录系统，从Fabric CA换取证书，利用该证书作为身份证明访问区块链数据。密码错误或未登录的用户会被系统拦截，用例user.auth验证在用户未登录情况下尝试访问区块链，预期的系统输出为拦截请求，输出错误日志。执行该用例发现请求被拦截，打印错误日志如图 5.4 所示，可见用户认证失败，测试通过。

表 5.5: 用户认证模块测试用例

ID	说明	输入	预期输出	结果
user.enroll.success	用户登录	正确用户名、密码	登录成功，返回用户证书	通过
user.enroll.fail	用户登录	错误用户名、密码	登录失败	通过
user.auth	用户授权	未授权的用户访问区块链接口	拒绝访问	通过

```
hyperledger.fabric_ca.sdk.HFCAClient     :
POST request to http://*.*.*.*:7054/api/v1/enroll    with request body:
{"certificate_request":"-----BEGIN CERTIFICATE REQUEST-----
\nMIHLwNCAARCfyTWEEzpj6E…3hA0=\n-----END CERTIFICATE REQUEST-----\n",
"attr_reqs":[{"name":"software"}]}, failed with status code: 401.
Response: {"result":"","errors":[{"code":20,"message":"Authentication failure"}],
"messages":[],"success":false}
```

图 5.4: 未授权用户访问系统错误日志

5.1.3　性能测试

本文使用区块链压测工具Hyperledger Caliper[1] 完成性能测试。Caliper是一个区块链性能基准测试框架，允许用户使用预定义的测试用例测试区块链应用的性能，包括交易的成功率、延迟、吞吐量和资源占用。本节为chaincode中的每个方法编写一组测试用例，验证本系统是否满足性能需求。

本文对chaincode中的主要方法进行压测，按照并发请求数取50tps, 100tps, 150tps, 200tps, 300tps 进行压测。并发数为50tps时的测试结果如图 5.5 所示。

Test	Name	Succ	Fail	Send Rate	Max Latency	Min Latency	Avg Latency	Throughput
1	save	1000	0	50 tps	0.40 s	0.04 s	0.17 s	50 tps
2	findOne	1000	0	50 tps	0.24 s	0.01 s	0.09 s	50 tps
3	query	1000	0	50 tps	0.28 s	0.02 s	0.11 s	50 tps
4	delete	1000	0	50 tps	0.32 s	0.04 s	0.13 s	50 tps
5	queryWithPagenation	1000	0	50 tps	0.29 s	0.03 s	0.09 s	50 tps
6	history	1000	0	50 tps	0.27 s	0.03 s	0.09 s	50 tps
7	autoAccept	1000	0	50 tps	0.52 s	0.09 s	0.22 s	50 tps

TYPE	NAME	Memory(max)	Memory(avg)	CPU(max)	CPU(avg)	Traffic In	Traffic Out	Disc Read	Disc Write
Docker	peer0.org2.example.com	150.7MB	148.6MB	47.77%	29.58%	3.8MB	6.4MB	44.0KB	3.7MB
Docker	peer0.org1.example.com	120.3MB	118.4MB	44.10%	29.95%	3.7MB	8.5MB	92.0KB	3.7MB
Docker	orderer.example.com	29.6MB	27.4MB	26.89%	17.40%	2.6MB	4.8MB	48.0KB	2.6MB
Docker	ca.org2.example.com	19.2MB	19.2MB	0.00%	0.00%	0B	0B	0B	0B
Docker	ca.org1.example.com	6.3MB	6.3MB	0.00%	0.00%	0B	0B	0B	0B

图 5.5: Caliper性能测试汇总报告

可见在50tps并发条件下，各方法性能表现良好，请求失败率0%，吞吐量均达到50tps，延迟较高的autoAccept方法最高延迟为0.52秒，所有方法平均延迟为0.12 秒。在资源占用方面，两个业务节点相对消耗资源较多，Org1和Org2的业务节点分别占用了148.6MB和118.4MB 的内存，最高占用47.77%和44.10%的CPU 资源。排序节点Orderer和证书颁发机构Fabric CA对内存、CPU 和磁盘压力较小。总体来说本系统在资源消耗方面表现良好。

通过不断增加并发数量，经过一组连续的测试，得出性能测试结果如图5.6所示当并发数由50tps 逐渐增加到300tps时，各方法的请求失败率仍保持在0%，平均延迟时间由0.12秒增加至4.01秒，最高延迟时间由0.25秒增加至6.01秒。系

[1]https://hyperledger.github.io/caliper/

统吞吐量由50tps增加至248tps。可见随着并发数量的增加，系统延迟逐渐上升，但仍保持在可用范围内。综上，本系统能够满足真实场景下的高并发请求，系统在高峰值请求场景下，依然能够正常运行，保证了较高的可用性。

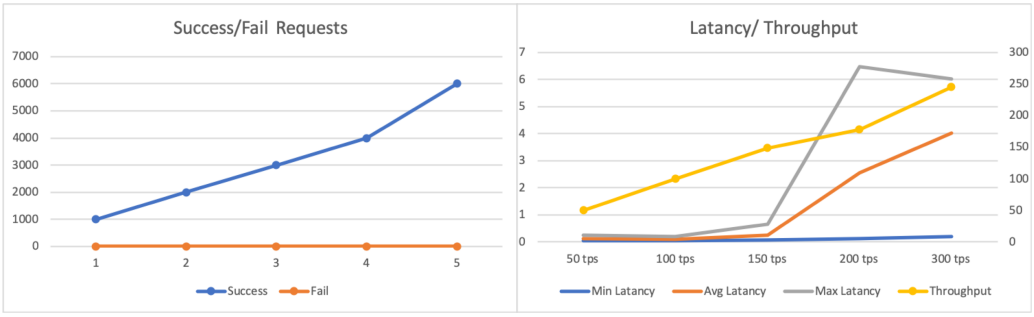

图 5.6: 性能测试结果

5.2　系统安全性分析

软件资产交易子系统使用PBFT作为共识算法。共识过程分为预准备、准备和确认三个阶段。假设系统总节点数为N, 恶意节点数为f。图 5.7 展示了PBFT算法的执行流程。

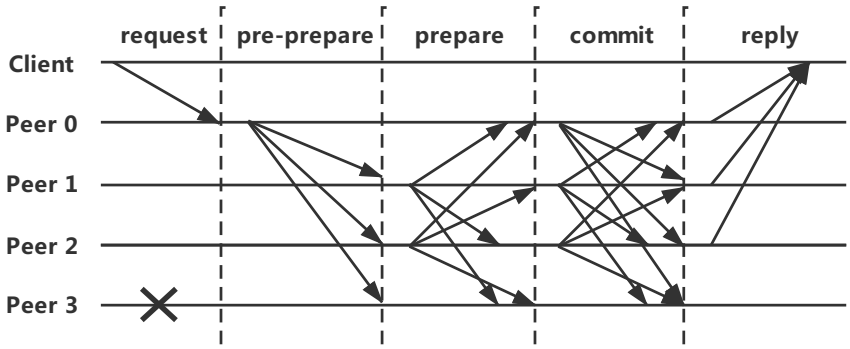

图 5.7: PBFT执行流程

客户端Client向Peer0发送请求后，进入预准备阶段，Peer0为受到的请求分配一个序号n, 向其它节点广播预准备消息$<< pre-prepare, n, d, i > m >$, 其中$i$表示节点身份，$m$为客户端发送的请求消息，$d$ 为消息摘要。Peer1和Peer2收到预准备消息时，检查d 与m 的摘要是否一致，在此之前是否接受过序号为n, 摘要为d的消息，若检查通过则进入准备阶段。Peer3 为失效节点，不执行检查过

程。在准备阶段Peer1和Peer2 向其它节点广播准备消息$< prepare, n, d, i >$, 并将预准备消息和准备消息写入本地消息日志。节点收到$2f$个与预准备消息一致的准备消息时，进入确认阶段，向其它节点广播确认消息$< commit, n, D(m), i >$, 当节点收到$2f$个与预准备消息一致的确认消息时，确认阶段完成。

网络中存在f个恶意节点的情况下，系统必须在$N-f$个节点的沟通中达成正确的共识。由于无法预测恶意节点的行为（发送错误消息或不发送消息），为保证正常节点数大于作恶节点数，有$(N-f)-f>f$, 从而得出$N>3f$ [43]。

基于上述分析，在软件资产交易子系统中失效/恶意节点数量少于总节点数的1/3的情况下，恶意节点无法篡改区块连数据，系统能够保证安全性和可用性。系统使用P2P的网络架构，避免单点故障，基于智能合约完成验收过程，保证了验收过程可信，不存在验收数据被某一中心机构持有和篡改的风险。

5.3　本章小结

本章对软件资产交易子系统进行了测试和分析。首先分模块对系统进行功能测试，给出了测试用例设计和测试结果。使用Caliper对系统进行性能测试，给出了系统在不同并发请求数下的压测结果。随后分析了系统的安全性属性。最终结果表明软件资产交易子系统能够正确满足需求分析中要求的功能，能够在高并发条件下保证系统可用性和安全性。

第六章　总结与展望

6.1　总结

软件资产管理系统是在互联网行业迅速发展，软件作为一种数字资产缺乏可靠、高效的确权和交易手段的背景下，利用区块链技术为软件资产的价值流通提供可靠保障的系统。基于区块链技术开发的软件资产交易子系统为软件资产管理系统提供了定制化的区块链数据存储和智能合约服务，管理软件资产交易周期，明确软件资产所有权归属，避免软件资产在交易过程中的价值流失。

本文首先分析了软件资产管理系统的业务需求，进而得出软件资产交易子系统的功能需求和非功能需求。研究了区块链的关键技术：数据存储、共识机制和智能合约，对比了普遍应用的几种技术方案，分析其优缺点和使用场景，制定了切合项目需求的技术路线，依据技术路线选取了开源框架Hyperledger Fabric 完成系统开发，保证了软件产权数据、测试数据一经确认就达成全网共识，不可篡改，实现可靠的资产确权和资产流通。

为解决软件资产管理系统需要存储大量文件和区块链存储能力低下的矛盾，系统采用了链外文件存储的解决方案。将文件存储在IPFS私有网络中，为文件生成基于内容的唯一哈希值，区块链中只存储文件的哈希。扩展了区块链存储能力的同时也保证了文件的不可篡改性。

软件资产交易子系统以客户端的形式对外暴露服务，将所有功能服务封装为Java编程接口，提供了基于Spring的Java依赖包。后台业务系统只需引入平台客户端依赖就可以方便地访问区块链数据和调用智能合约。

为方便后续维护，系统提供了高度抽象的文档存储接口，为灵活应对系统的业务扩展提供了基础。并且编写了面向开发和测试环境的单节点部署脚本和面向生产环境的多节点部署脚本，实现了系统的自动化部署，简化了开发和部署流程。最终测试结果表明基于区块链技术的软件资产交易子系统能够很好地支持软件资产管理系统的业务，有良好的可用性和可扩展性。

6.2　进一步展望

本项目是区块链技术用于软件领域的一次探索，旨在可靠地记录软件资产所有权归属和软件资产在交易过程中产生的测试数据、验收数据等。未来可以扩展的工作有以下几点：

提供易用的的区块链网络管理功能。现阶段如果要增加或移出网络中的节点，需要在服务器上运行shell 脚本完成，过程繁琐且容易出现错误。接下来可以提供易用性更高的图形界面，方便运维人员进行网络管理。

系统记录的软件资产所有权证明只是参与该区块链网络的节点认可的所有权凭证。如何使该凭证具备法律效力，能够在更广泛的范围中发挥作用是接下来要解决的重要问题。可以考虑在系统中引入监管部门或相关法律机构。

考虑利用智能合约完成软件资产交易过程。一个软件系统可能是由多个软件提供商合作完成的，软件资产的产权归属于多个机构。基于智能合约实现软件资产交易，按照预定的分配规则完成资产分配，可以减少交易中的纠纷，削减人力成本，大大提高软件资产交易效率。

系统记录了大量可靠的的软件资产数据，可以从这些数据中挖掘更多价值。例如，利用软件版本变更记录和对应的测试报告，分析软件迭代过程中的质量变化和功能演变，基于软件的测试数据和验收数据，评估软件开发方和软件测评机构的专业能力等。

参考文献

[1] Q. Liu, R. Safavi-Naini, N. P. Sheppard, Digital rights management for content distribution, in: ACSW Frontiers 2003, 2003 ACSW Workshops - the Australasian Information Security Workshop (AISW) and the Workshop on Wearable, Invisible, Context-Aware, Ambient, Pervasive and Ubiquitous Computing (WICAPUC), Adelaide, South Australia, February 2003, 2003, pp. 49–58.

[2] W. T. Councill, Third-party testing and the quality of software components, IEEE Software 16 (4) (1999) 55–57.

[3] D. Kim, J. Moon, S. Cho, J. Choi, M. Park, S. Han, L. Chung, A birthmark-based method for intellectual software asset management, in: S. Lee, S. Kim, L. Hanzo, R. Ismail (Eds.), The 8th International Conference on Ubiquitous Information Management and Communication, ICUIMC '14, Siem Reap, Cambodia - January 09 - 11, 2014, ACM, 2014, pp. 39:1–39:6.

[4] F. Mintzer, G. W. Braudaway, M. M. Yeung, Effective and ineffective digital watermarks, in: Proceedings 1997 International Conference on Image Processing, ICIP '97, Santa Barbara, California, USA, October 26-29, 1997, IEEE Computer Society, 1997, pp. 9–12.

[5] J. Van Wijk, Dealing with piracy: Intellectual asset management in music and software, European Management Journal 20 (6) (2002) 689–698.

[6] C. S. Collberg, J. W. Davidson, R. Giacobazzi, Y. X. Gu, A. Herzberg, F. Wang, Toward digital asset protection, IEEE Intelligent Systems 26 (6) (2011) 8–13.

[7] S. Huh, S. Cho, S. Kim, Managing iot devices using blockchain platform, in: 2017 19th international conference on advanced communication technology (ICACT), IEEE, 2017, pp. 464–467.

[8] Y. Yuan, F. Wang, Towards blockchain-based intelligent transportation systems, in: 19th IEEE International Conference on Intelligent Transportation Systems, ITSC 2016, Rio de Janeiro, Brazil, November 1-4, 2016, IEEE, 2016, pp. 2663–2668.

[9] B. E. Albert, R. P. dos Santos, C. M. L. Werner, Software ecosystems governance to enable IT architecture based on software asset management, in: 7th IEEE International Conference on Digital Ecosystems and Technologies, DEST 2013, Menlo Park, CA, USA, July 24-26, 2013, IEEE, 2013, pp. 55–60.

[10] K. Waedt, A. Ciriello, M. Parekh, E. Bajramovic, Automatic assets identification for smart cities: Prerequisites for cybersecurity risk assessments, in: IEEE International Smart Cities Conference, ISC2 2016, Trento, Italy, September 12-15, 2016, IEEE, 2016, pp. 1–6.

[11] E. Portmann, Rezension "blockchain: Blueprint for a new economy", HMD - Praxis Wirtschaftsinform. 55 (6) (2018) 1362–1364.

[12] M. Zeilinger, Digital art as ‘monetised graphics’: Enforcing intellectual property on the blockchain, Philosophy & Technology 31 (1) (2018) 15–41.

[13] G. Hileman, M. Rauchs, Global blockchain benchmarking study, Cambridge Centre for Alternative Finance, University of Cambridge 122.

[14] S. Nakamoto, et al., Bitcoin: A peer-to-peer electronic cash system.

[15] G. Wood, Ethereum: A secure decentralised generalised transaction ledger, Ethereum project yellow paper 151 (2014) 1–32.

[16] E. Androulaki, C. Cachin, A. D. Caro, A. Sorniotti, M. Vukolic, Permissioned blockchains and hyperledger fabric, ERCIM News 2017 (110).

[17] 邵奇峰, 金澈清, 张召, 钱卫宁, 周傲英, 区块链技术:架构及进展, 计算机学报v.41；No.425 (5) (2018) 969–988.

[18] 工信部, 中国区块链技术和应用发展白皮书, Tech. rep., 2016 (2016).

[19] J.-T. Lorenz, B. Münstermann, M. Higginson, P. B. Olesen, N. Bohlken, V. Ricciardi, Blockchain in insurance-opportunity or threat, McKinsey & Company Insurance (6).

[20] 何蒲, 于戈, 张岩峰, 鲍玉斌, 区块链技术与应用前瞻综述, 计算机科学44 (4) (2017) 1–7.

[21] 蔡亮, 李启雷, 梁秀波, 区块链技术进阶与实战, 人民邮电出版社, 2018.

[22] 袁勇, 王飞跃, 区块链技术发展现状与展望, 自动化学报42 (4) (2016) 481–494.

[23] Z. Zheng, S. Xie, H. Dai, X. Chen, H. Wang, Blockchain challenges and opportunities: a survey, IJWGS 14 (4) (2018) 352–375.

[24] 章峰, 史博轩, 蒋文保, 区块链关键技术及应用研究综述, 网络与信息安全学报4 (4) (2018) 22–29.

[25] A. Baliga, Understanding blockchain consensus models, in: Persistent, 2017.

[26] C. Cachin, M. Vukolic, Blockchain consensus protocols in the wild (keynote talk) 91 (2017) 1:1–1:16.

[27] M. Ball, A. Rosen, M. Sabin, P. N. Vasudevan, Proofs of useful work, IACR Cryptology ePrint Archive 2017 (2017) 203.

[28] 刘肖飞, 基于动态授权的拜占庭容错共识算法的区块链性能改进研究, Ph.D. thesis, 浙江大学(2017).

[29] N. Szabo, Smart contracts: building blocks for digital markets, EXTROPY: The Journal of Transhumanist Thought,(16) 18.

[30] V. Buterin, et al., A next-generation smart contract and decentralized application platform, white paper.

[31] M. Alharby, A. van Moorsel, Blockchain-based smart contracts: A systematic mapping study, CoRR abs/1710.06372.

[32] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, J. Yellick, Hyperledger fabric: a distributed operating system for permissioned blockchains (2018) 30:1–30:15.

[33] P. Thakkar, S. Nathan, B. Viswanathan, Performance benchmarking and optimizing hyperledger fabric blockchain platform, in: 26th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2018, Milwaukee, WI, USA, September 25-28, 2018, IEEE Computer Society, 2018, pp. 264–276.

[34] T. A. Howes, M. C. Smith, G. S. Good, Understanding and deploying ldap directory services, Political Science Quarterly 118 (3) (1999) 365 – 388.

[35] R. E. Voglmaier, The ABCs of LDAP: how to install, run, and administer LDAP services, CRC Press, 2003.

[36] J. Benet, IPFS - content addressed, versioned, P2P file system, CoRR abs/1407.3561.

[37] M. S. Ali, K. Dolui, F. Antonelli, Iot data privacy via blockchains and IPFS, in: S. Mayer, S. Schneegass, B. Anzengruber, A. Ferscha, G. Anderst-Kotsis, J. Paradiso (Eds.), Proceedings of the Seventh International Conference on the Internet of Things, IOT 2017, Linz, Austria, October 22-25, 2017, ACM, 2017, pp. 14:1–14:7.

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

[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, IEEE, 2017, pp. 1–5.

[40] C. Boettiger, An introduction to docker for reproducible research, with examples from the R environment, CoRR abs/1410.0846.

[41] C. Boettiger, An introduction to docker for reproducible research, Operating Systems Review 49 (1) (2015) 71–79.

[42] P. Kruchten, Architecture blueprints - the ”4+1” view model of software architecture (1995) 540–555.

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

致　　谢

首先感谢刘嘉老师和陈振宇老师在我的论文写作和项目开发过程中给我的悉心指导，为我指明了方向，在项目的设计开发过程中给了我很多督促和帮助，在论文写作上提出了很多建设性的意见。感谢两位老师在我研究生期间对我的指导，他们严谨的态度和极强的行动力是我受益终生。

感谢与我共同完成项目的杨登辉和陈圣超同学。他们在项目开发过程中始终认真负责，有责任心，为我提供了许多帮助，一起度过这段充实的时间。

感谢黄勇先生，在项目开发过程中帮助我解决了很多棘手的问题，传授了许多经验。

感谢我的朋友朱静怡、李灏宇、张杨洋，感谢他们一致以来的温暖和陪伴，是因为他们我才有继续前行的动力，感谢你们。