南京大学

研 究 生 毕 业 论 文

（申请工程硕士学位）

论 文 题 目 基于区块链的软件交付过程管理系统的设计与实现

作 者 姓 名 陈圣超

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：**MF1732014**

论文答辩日期：**2019 年 5 月 14 日**

指 导 教 师：　　（签字）

The Design and Implementation of Software Delivery Process Management System Based on Blockchain

By

Shengchao Chen

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：基于区块链的软件交付过程管理系统的设计与实现

工程硕士（软件工程领域）专业 2017 级硕士生姓名：陈圣超

指导教师（姓名、职称）：陈振宇 教授

摘 要

软件交付是是软件外包中的重要一环。传统外包软件交付过程常常存在着需求变更不透明、第三方测评结果不可信或被篡改、交付软件与被测评软件不一致等问题。区块链技术具有去中心化、不可篡改、可溯源和可执行智能合约等特点。将区块链技术应用于软件交付过程，能够保障软件及其关键数据的可信与可溯源，以期能够解决传统软件交付过程存在的问题。

本文设计和实现了一个基于区块链的软件交付过程管理系统。本系统为软件需求方、软件开发方与软件测评方提供订单管理、软件证书管理、测试报告管理和软件自动验收等功能。本系统采用当前流行的联盟链框架Hyperledger Fabric作为本系统的区块链底层实现。本系统采取微服务架构，系统被划分为细粒度且业务聚焦的若干服务，包括用户服务、文件服务、订单服务、软件证书服务和自动验收服务等。每个服务中均采用分层架构，分为展示层、控制层、逻辑层和数据持久层。为减轻区块链数据存储压力，本系统将软件证书等文件存入星际文件传输系统，在区块链中仅存储关键对象与代表文件地址的哈希值。本系统的展示层使用Vue框架生态，业务逻辑层使用SpringBoot框架，缓存数据库使用Redis与TiDB，服务发现与注册选用Consul，用户认证使用LDAP。本系统使用Kubernetes进行服务编排与部署，有较强的水平拓展性与较高的服务可用性。

本系统通过区块链的不可篡改性保证系统数据的真实可信。为方便用户将页面展示信息与链上存储信息进行比对，订单详情页面已内置诸多可跳转至Hyperledger Explorer的超链接。通过将链上数据与页面数据进行对比，用户可确定软件测试结果是否被篡改以及交付软件与测评软件是否一致，这使得软件交付过程的可靠性得到保障。本系统引入TiDB作为星际文件系统数据缓存层，经过对读接口进行测试，缓存的加入使得读接口的效率提高近一倍。目前该系统已进入试运行阶段，已有数家软件测评结构顺利接入本系统，以本系统为支撑的软件交付生态正在建立。

关键词： 区块链，星际文件传输系统，软件交付，微服务

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 Delivery Process Management System Based on Blockchain

SPECIALIZATION: Software Engineering

POSTGRADUATE: Shengchao Chen

MENTOR: Professor Zhenyu Chen

Abstract

Software delivery is an important part of software outsourcing. Traditional outsourcing software delivery processes often have problems such as opaque demand changes, untrustworthy or falsified third-party evaluation results, and inconsistencies between the delivered software and the tested software. Blockchain technology is characterized by decentralization, traceability, and executable smart contracts. Applying blockchain technology to the software delivery process ensures the trustworthiness and traceability of the software and its critical data, in order to solve the problems of the traditional software delivery process.

This thesis designs and implements a blockchain-based software delivery process management system. The system provides order management, software certificate management, test report management and software automatic acceptance for software demanders, software developers and software assessors. The system adopts the current popular alliance chain framework Hyperledger Fabric as the bottom layer of the blockchain of the system. The system adopts a micro-service architecture, and the system is divided into several services with fine-grained and business-focused services, including user services, file services, order services, software certificate services, and automatic acceptance services. Each service adopts a layered architecture, which is divided into display layer, control layer, logic layer and data persistence layer. In order to alleviate the pressure of blockchain data storage, the system stores software certificates and other files into the interstellar file transmission system, and stores only the hash values of key objects and representative file addresses in the blockchain. The technical route of the system is: the presentation layer uses the Vue framework ecology, the business logic layer uses the SpringBoot framework, the cache database uses Redis and

TiDB, the service discovery and registration uses Consul, and the user authentication uses LDAP. This system uses Kubernetes for service orchestration and deployment, with strong horizontal scalability and high service availability.

The system guarantees the authenticity of the system data through the irreversible modification of the blockchain. In order to facilitate the user to compare the page display information with the stored information on the chain, the order details page has built-in hyperlinks that can be jumped to the Hyperledger Explorer. By comparing the data on the chain with the page data, the user can determine whether the results of the evaluation report have been tampered with and whether the delivery software is consistent with the evaluation software, which ensures the reliability of the software delivery process. This system introduces TiDB as the data cache layer of the IPFS. After testing the read interface, the addition of cache makes the efficiency of the read interface nearly double. At present, the system has entered the trial operation stage, and several software evaluation structures have been successfully connected to the system. The software delivery ecosystem supported by this system is being established.

Keywords: Blockchain, IPFS, Software Delivery, Microservices

目　录

表　目　录

图　目　录

第一章　引言

1.1　背景与研究意义

随着信息技术的不断发展，软件正在不断优化大众的生活方式，越来越多的传统企业与公共事业单位也在努力借助电子信息技术提升生产效率和服务效率。因资金或技术实力等原因，大部分传统企业和公共事业单位在短期内无法建设自有的信息技术团队。在这种背景下，信息服务产业中的软件服务外包成为一种较为可行的方案 [1]。如图 1.1 展示了传统软件外包服务的主要流程。主要参与者包括软件产品需求方、软件开发方以及软件测试机构。软件产品的需求方通过某种渠道发布其任务需求，软件开发方与软件需求方确认需求后进入开发阶段，软件开发方完成开发后将软件制品交给有资质的软件测试方进行测评，软件测评方对软件进行测评并形成电子或书面形式的测评报告，测评报告与软件制品一同交付给软件需求方进行验收 [2]。

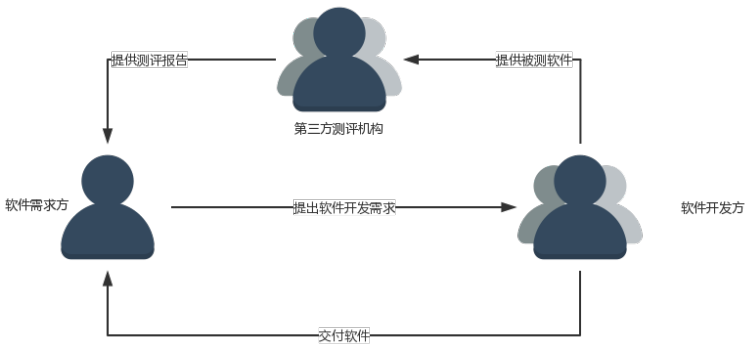

图 1.1: 传统软件外包服务主要流程

和传统实体贸易中的产品不同，软件产品是一种无形资产，其验收和测试需要较强的专业能力，且不够透明公开。软件测试数据容易被篡改和造假，软件需求方很难验证软件性能指标是不是达到了预期。软件产品交付与验收过程由于缺乏统一的标准与平台，显得过程复杂且效率低下 [3]。

如何通过互联网进行可靠的软件交付并对软件产品进行验证成为优化软件外包流程的重要问题。去中心、防篡改且可溯源的区块链技术为这一问题提供了更好的应用解决方案 [4]。基于区块链，建立新一代软件交付过程管理平台，

打造互信协同的软件服务生态圈。依托区块链网络，软件需求更改记录可查[5]，软件开发的利益得到保障，测试数据的不可篡改，真实性得到强力公证，以软件测试数据作为软件服务价值的重要权衡，打造互信软件市场，降低信任成本，也有利于维护整个软件外包开发的生态管理。

1.2　国内外研究现状

1.2.1　国内现有软件交付平台

目前国内较大的软件交付平台有：智城网、猪八戒网以及国家软件与信息服务外包公共支撑平台。智城网提供在线项目交易与管理服务，是一家运营数十年的老牌外包网站，主要提供移动应用开发、平面设计以及网站建设等方面的服务。猪八戒网是综合类型众包服务网站，站内有较多服务商为企业或个人提供专属的解决方案，例如商标定制、广告设计与软件开发服务等，最终制品通过猪八家平台进行交付。国家软件与信息服务外包公共支持平台在国家工信部的指导下建立与运营，其目标是建成覆盖我国软件与信息服务外包企业聚集地区的公共服务支持体系，提升软件与信息服务外包产业的核心竞争力，为各级政府机构、行业协会、产业园区和企业提供完善和持续性的公共服务。这些平台均有一定管理项目功能，但对保证软件交付质量并无太多有效机制，传统软件交付中可能存在的例如虚假软件、虚假测试报告、中心化导致数据篡改等问题依然存在 [6]。

1.2.2　区块链技术应用现状

区块链是一种去中心化、数据不可篡改、更新历史可追溯的由多方维护的分布式数据库。任何一方都无法完全控制或者独立修改这些数据，只能严格遵循规则以及特有的共识对数据进行更新。基于这些特征，链上数据可以被多方共享与共同监管 [7]。当前区块链从节点权限角度主要可以分为公有链和联盟链：公有链的典型代表有比特币和以太坊，任何人接入互联网的人都可以在任意时间加入其中，公有链上的所有数据均对所有人开放；联盟链是有特定参与成员节点组成的联盟，成员间业务往来产生的数据信息均被写入联盟链，每个节点均持有一份数据，联盟链限定了使用规模和接入权限。联盟链的最典型实现是开源区块链项目Hyperledger，它由Linux基金会负责维护 [8]。不同类型的区块链框架实现均因其各自特点，有着各自适用的场景。区块链技术因其基本特点目前在投票统计、医疗档案、能源管理、资产登记、存货清单和专利发明等信息共享领域已有一些落地项目尝试 [9–11]。区块链技术在国内已有一些业

务场景的实践，网易、京东、百度和腾讯等互联网厂商都在积极得开发基于区块链的业务与数据平台。

区块链有着天然的去中心化特质，很多强中心化场景可能被区块链技术提供新的思路 [12]。金融行业是国民经济领域里中心化程度较高的产业之一，区块链的去中心化等特征可以为金融领域中的不同个体搭建有效的信任机制，通过区块链减少大量信任中介可以大幅度优化行业运转速度，提高生产效率。金融产业中的大量数据，例如交易信息、股权证明、电子票据以及债券信息等均可存入区块链中，成为可被共享的链上数据。区块链创业公司Chain和国际银行卡组织Visa已经共同开发出B2B跨境支付项目，在数十个国家的银行中已进入生产。除此以外，Visa和Chain联合开发的区块链系统可以实现支付交易的实时处理，从而提高效率，降低成本。

区块链技术还可以应用于居民医疗领域。目前，众多医疗机构正面临着无法安全得进行跨平台与跨地区进行共享数据的问题 [13]。在医疗服务商之间搭建可信的数据通道，可以帮助医疗人员获取多方面信息，这有利于进一步提高医疗诊断准确率，从而改善患者的治疗效果，在一定程度上降低医疗成本。

区块链技术在司法事务方面也有极大的应用前景，利用区块链技术可以在不同的地区、政府部门之间可以安全地共享可信的政务信息 [14]。在网络著作权、互联网金融等领域存在数量庞大的潜在诉讼案件。该类潜在纠纷案件与机构之间信息不对称、网络信息隐私保护不足以及治理风险识别机制不健全密切相关 [4]。将司法数据存入区块链，保障司法数据的不可篡改性，可达成“诉源治理”机制，坚固“司法是维护社会公平正义的最后一道防线”这一理念。例如广州互联网法院开发的“网通法链”智慧信用生态系统，通过司法区块链、可信电子证据平台和司法信用共治平台来推进法律与区块链的结合。司法区块链为整个生态系统提供底层的数据加密、存储、交换基础；可信电子证据平台依靠司法区块链信息不可篡改的特性，实现电子数据的存储、调取等功能，从而降低司法过程中的举证、质证成本，提高司法效率；司法信用共治平台，依靠司法区块链的数据传输、访问安全的特性，平台共建方之间共享信用数据，并通过大数据分析和数据挖掘等技术，为社会大众提供信用评估服务。

1.3　本文主要研究工作

本项目实现一个去中心化的软件交付过程管理系统，基于区块链技术实现去中心化、不可篡改、多方写入、共同维护的特性，在软件交付验收领域的机构之间搭建新的信任体系，其主要目标是优化软件交付的流程，提高软件测评机构结果的可信度。同时在一般的区块链分布式应用架构的基础上，把区块链

层做薄，引入了新的业务逻辑层与缓存机制，提高了项目的性能。

本论文重点讨论基于区块链的软件交付过程管理系统的设计与业务实现部分，同时对系统自动部署流程进行一定的分析。本系统对外提供多种形式的访问与调用形式：包括但不局限于OPENAPI、内置SDK等。项目的业务逻辑层使用框架Spring框架，考虑的角度包括：易用性、可拓展性、安全性等。本项目使用了微服务架构，考虑的角度主要是：可维护性、可拓展性以及性能等。在设计接口时本项目考虑了业务的可拆分与实现的可替换，注意兼容性与可拓展性，为以后的继续开发打下了基础。

本项目除了实现数据管理平台的相关业务需求，计划在数据建模上进行更高度的抽象，为建立更多数据类型的平台提供抽象度较高的接口与详细的二次开发文档。为了提高测试与部署效率，本项目使用Ansible、Jenkins、Docker和kubernetes等开源工具进行一整套的持续集成与持续部署的实践。

1.4　本文的组织结构

本文总共分为六个章节，组织结构如下。

第一章为绪论，分析了互联网时代软件交付过程中现存问题与痛点，介绍了在溯源追踪方面有天然优势的区块链技术，介绍了市场上现有的区块链应用方向，在这此基础上，提出了一套基于区块链的软件交付过程管理平台方案，综合利用Spring生态、Vue生态、DevOps等多维度的知识，进一步提升软件知识产权管理平台的易用性、可靠性、可维护性和安全性等质量属性。

第二章为技术综述，本章介绍了项目所使用的核心技术和框架，包括区块链的类型与开源实现、星际文件系统（IPFS）、Vue前端框架、Docker、kubernetes和Ansible等。

第三章为需求分析与概要设计，本章对整个项目从功能需求和非功能需求两个角度进行了分析，并根据分析结果进行了项目的概要设计，为第四章的详细设计提供基础。

第四章为详细设计和实现，本章对项目中的功能模块进行了详细设计，并提供了关键逻辑的代码实现或伪代码。第五章为自动部署流程与系统测试，本章对项目的部署方式进行了详细的介绍，同时从单元测试与接口测试对系统进行测试。

第六章为总结与展望，本章总结了项目开发与论文撰写中的工作，列出了具有通用性的设计和工具，同时对项目中的不足进行了分析，就整个软件交付过程管理系统的未来发展进行了一定的展望。

第二章　技术综述

2.1　区块连技术概述

区块链是一种去中心化、数据不可篡改、更新历史可追溯的由多方维护的分布式数据库，任何一方都无法完全控制或者独立修改这些数据，只能严格遵循规则以及特有的共识对数据进行更新，基于这些特征，链上数据可以被多方共享与共同监管。区块链是数字货币比特币的底层实现技术。比特币于2008年由匿名用户“中本聪”在互联网提出 [15]，它构建了一种点对点的去中心化的货币系统。2014年，Buterin提出了以太坊平台，以太坊在比特币的基础协议之上还提供了图灵完备的编程语言以编写与运行智能合约，这是智能合约首次被应用到区块链中 [16]。2015年12月，Linux基金会发起了Hyperledger开源区块链项目，该项目旨在发展跨行业的商业区块链平台。超级账本中最受关注的是Fabric项目，它由IBM 最初发起。与面向公共的比特币和以太坊有所不同，Hyperledger Fabric被设计来服务于企业级的区块链应用，并引入成熟的成员管理服务 [17, 18]。

区块链从读取权限角度可分为公有链和联盟链这两种。公有链指的是全世界任何人都可读取、可发送交易且可参与共识过程的区块链。公有链构建在开放网络上，节点可以自由的加入和退出，访问门槛低且所有数据默认公开，通常被认为是“完全去中心化”的。共识算法采用PoW、PoS等，交易效率极低 [19]。而联盟链中，只有获得特定许可的节点才能接入网络，节点之间传递的消息必须是可验证来源的。身份管理以及权限控制机制是联盟链中的重要组件，准入机制的存在杜绝了女巫攻击，使得传统共识算法有了用武之地，相比公有链能够高效处理交易。相比于公有链，联盟链的吞吐量获得质的飞跃，这一特性可以很好的满足企业级应用的性能需求 [20]。

由于本项目具有以下特点：（1）在许可网络而非公共网络中运行，接入本系统的组织均需要获得授权。（2）对测试结果的认证可能有多种规则且较为复杂，需要支持智能合约。（3）对系统吞吐量有一定要求。因此考虑使用支持智能合约的联盟链来实现。在目前的开源实现中，上文提到的Hyperledger Fabric是最为成功且最为成熟的 [21]，国内众多的厂商如华为、苏宁等也都基于此搭建自己的企业级区块链服务。故本项目选用Hyperledger Fabric作为底层区块链实现。

2.2　星际文件系统

在计算机的世界里，两个用户如果想要互相发送信息，他们需要通用的规则集来定义信息的传输方式和时间，这些规则被广泛地称为通信协议。从1980年代开始，通信协议使得世界上的计算机可以相互连接并通信。当今的互联网主要是客户端和服务器间的交互，这种交互依赖以超文本媒体传输协议（也就是我们常说的HTTP协议）为主的IP协议。很长一段时间里，数据存储在集中式服务器中，并通过基于位置的寻址进行访问。这样可以更轻松地分发、管理和保护数据，并扩展服务器和客户端的容量。然而，在安全性、隐私性和效率领域这种方式存在许多弱点：服务器的控制转换为对数据的控制。这意味着任何控制服务器的一方都可以访问，更改和删除服务器上的数据。在基于位置的寻址中，数据通过它所在的位置而不是其内容来识别。这种限制意味着用户必须一直到特定位置访问一段数据，即使相同的数据可以在更近的地方获得。也无法判断数据是否已被更改，因为客户端只需知道数据在哪里，而不知道数据是什么。

星际文件系统是点对点协议InterPlanetary File System 的中文名，一般简称为IPFS。该协议是Juan Benet在2014年5月份发起。IPFS试图通过一种新颖的p2p文件共享系统解决CS模型和HTTP协议的不足 [22]。

IPFS由如下几个关键组件构成：分布式哈希表、块交换、Merkle DAG、版本控制系统、自认证文件系统。哈希表是将信息存储为键/值对的数据结构，在分布式哈希表中，数据分布在计算机网络上，并且有效地协调以实现节点之间的有效访问和查找。块交换组件是IPFS实现的一个数据交换工具，用于节点间的数据传输。Merkle DAG是默克尔树（Merkle Tree）和有向无环图（Directed Acyclic Graph）的结合，默克尔树确保在p2p网络上交换的数据块是正确的、未损坏的且不变的。通过使用加密散列函数组织数据块来完成此验证 [23]。这只是一个函数，它接受输入并计算与该输入相对应的唯一字母数字字符串——也就是我们说的哈希值。Merkle DAG结构的另一个强大功能是它构建了分布式版本控制系统，它允许用户独立复制和编辑文件的多个版本，存储这些版本以及稍后将编辑与原始文件合并。IPFS的最后一个重要组成部分是自我认证文件系统，它是一个分布式文件系统，不需要特殊的数据交换权限。它是“自我认证”的，因为提供给客户端的数据是通过文件名进行身份验证的。有了这个认证，用户可以安全地访问远程内容 [24]。

简单得讲，使用分布式哈希表，节点可以存储和共享数据，而无需中央协调，Merkle DAG可实现唯一标识，防篡改和永久存储的数据，这样用户可以通过版本控制系统访问编辑数据的过去版本。

IPFS从根本上改变了查找的方式，这是它最重要的特征。使用HTTP我们查找的是位置，而使用IPFS我们查找的是内容。IPFS的做法则是不再关心中心服务器的位置，也不考虑文件的名字和路径，只关注文件中可能出现的内容。把资源文件放到IPFS节点，它会得到一个新名字——HashKey，这是一个由文件内容计算出的加密哈希值。哈希值直接反映文件的内容，哪怕只修改1比特，哈希值也会完全不同 [25]。

和FastDFS等其他的文件系统相比，IPFS扩展了加密货币和其他区块链技术,这都归功于IPFS提供的数据持久性与不可变更性，虽然这些特征的代价是额外的存储空间。本项目中，通过将一部分文件存在IPFS中从而做薄了区块链中存储的数据，区块链中仅存IPFS文件的Hash值，整个系统的性能得到了一定的提升。

2.3　前端框架Vue与ElementUI

Vue是一套构建用户界面的渐进式框架。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并非一个传统意义上的前端框架,其设计上也并没有完全遵循MVVM模式，只包含State和View 这两部分，Vue的核心设计思想强调的是状态到界面的映射，而并非代码的组织结构，所以说Vue并不像一个传统意义上的前端框架，更像是一个视图模板引擎。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 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。

ElementUI是由饿了么公司开源出的桌面端组件库，其目标用户为前端开发者、网页设计师以及产品经理，ElementUI与Vue 等前端框架有良好的兼容性。ElementUI遵从一致性、及时反馈、高效与可控的设计原则，为众多开发者提供优雅的数据展示方案。ElementUI的一致性体现在与现实生活习惯一致以及所有元素在界面中一致。为了让用户快速感知当前状态与自己是否进行了操作，Element在每次用户操作后均有界面样式变化与交互动态效果。ElementUI旨在帮助用户提高操作效率，同时尽量语言表达清晰且表意明确，让用户快速理解进而作出决策。在使用ElementUI的过程中，ElementUI 根据场景给出用户操作

建议或安全提示，用户拥有绝对自主权，可对ElementUI提供的流程根据自己的需求进行裁剪。

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 [26]。本项目以及整个项目接入的平台都是以Vue作为前端的技术体系。

2.4　微服务

2.4.1　微服务简介

最初软件开发一般选用单体架构即所有业务均写在同一项目之中，随着项目变大会导致代码膨胀并难以维护 [27]。后来为了具备一定的可扩展性和可靠性，垂直架构被提出，通过负载均衡改善前后端通信瓶颈问题，再后来SOA架构尝试解决了复杂应用系统之间是如何集成与互通 [28]，现如今的微服务架构则是进一步探讨一个复杂应用系统该被如何设计才能够更好得进行团队开发与更便于管理 [29]。微服务架构的本质就是围绕业务领域组件来创建各细业务粒度应用，让应用可以独立得被开发、管理与部署。

微服务的好处可以总结如下：每个微服务组件都简单灵活，负责独立业务，可由小规模团队独立开发，可独立部署 [30]；不需要一个庞大的应用服务器来支撑，可以由一个小团队更专注专业负责其开发与运维，相应得也就更高效可靠；微服务架构遵循“高内聚低耦合”的设计思想，每个微服务业务相对独立，且每个微服务有良好的可水平拓展性与可修改性；微服务架构与开发语言无关，每个微服务开发团队可自由选择合适的语言和工具。

2.4.2　微服务应用平台的常见解决方案

在引入了微服务架构后，也会暴露一些新的难点，例如依赖服务变更困难、团队之间的服务接口文档变更不透明、依赖服务导致难以上线、验证独立服务较为困难等。同时微服务还引入了分布式架构中的一系列问题，例如分布式事务的处理、依赖服务不稳定和服务上线依赖等。从部署运维角度分析，部署制品数量的增多与监控进程的增多带来更复杂的运维场景。

为尽可能解决上述问题，尽可能享受微服务架构带来的便利，通常会在开发、测试、部署运维等环节综合运用多种技术与方案。主流的微服务框架一般需要包括服务发现与服务治理，服务容错，服务熔断，服务网关，服务配置，负载均衡，消息总线，服务跟踪等功能。Spring Cloud是由Netflix开源出的一套成熟组件，由独立组件负责上述的每个功能。除了Spring Cloud整体方案，利

用Nginx、Consul、Etcd以及Dubbo等由不同团队维护的开源软件也可以实现微服务架构。

2.5 DevOps工具链

DevOps代表着开发（Development）和运维（Operations）的组合，强调软件开发人员和运维人员的沟通与合作，通过容器与编排等技术来使得软件构建、软件自动测试、软件部署发布更加得快捷、频繁与可靠。换句话讲，DevOps是一个完整的涉及软件开发、软件测试和软件部署的工作流，这个工作流以持续集成和持续部署为基础，来优化软件开发、测试、系统运维中的所有环节 [31]。广义的DevOps工具链包括代码管理、构建工具、自动部署、持续集成、配置管理、容器、容器编排、服务注册与发现、日志管理、系统监控压力测试、消息总线以及项目管理等诸多类型的工具。本小节仅选取系统开发中最有代表性的容器技术、编排技术、服务器管理技术和服务发现技术进行介绍与分析。

2.5.1 容器与Docker

容器提供了一种逻辑打包机制，以这种机制打包的应用可以脱离其实际运行的环境。利用这种脱离，不管目标环境是私有数据中心、公有云，还是开发者的个人笔记本电脑，开发者都可以轻松、一致地部署基于容器的应用。容器化使开发者和运维部署团队的关注点泾渭分明-开发者专注于应用逻辑和依赖关系，而运维部署团队可以专注于部署和管理，不必为应用细节分心，例如具体的软件版本和应用特有的配置。虚拟机是另一种较为成熟的虚拟化技术——在主机操作系统上运行且以虚拟化途径访问底层硬件的客机操作系统。与虚拟机相似，容器也让用户可以将应用与库和其他依赖项打包，提供独立环境来运行软件服务。但是，两者的相似性仅此而已，因为容器为开发者和运维部署团队提供了更加轻型、具有众多优势的运营单元。

虚拟机要求虚拟化硬件堆栈，与此不同的是，容器在操作系统级别进行虚拟化，且可以直接在操作系统内核上运行多个容器。也就是说，容器更轻巧：它们共享操作系统内核，启动速度更快，且与启动整个操作系统相比其占用的内存微乎其微。容器让开发者可以创建与其他应用相隔离的可预测环境。容器几乎能在任何地方运行，极大减轻了开发和部署工作量。容器会在操作系统一级虚拟化CPU、内存、存储和网络资源，为开发者提供在逻辑上与其他应用相隔离的沙盒化操作系统接口。可用的容器格式有许多，Docker是当前最受欢迎、开发者生态最为良好的容器格式，故本系统也采用Docker作为容器实现。

Docker 是一个开源的应用容器引擎，由Go语言进行开发，通过Apache2.0协议开源。Docker可以让开发者把应用打包成一个标准镜像，并且以容器的方式运行。Docker容器将一系列软件包装在一个完整文件系统中，这个文件系统包含应用程序所需的一切，包括代码、运行时工具、系统工具以及系统依赖。几乎任何可以装在服务器上的东西都可以被装入Docker中。这些策略保证了Docker容器内应用程序运行环境的稳定性 [32]。

Docker依赖Linux Kernel中的Namspace和Cgroups功能 [33]，所以即使Docker也可以在Windows上运行，本质上是先在Windows 上装了Linux系统虚拟机后再运行Docker。故本系统选用Centos等Linux系列服务器进行部署安装，以避免不必要的性能损失。

2.5.2　容器编排工具－Kubernetes

Kubernetes 在2015年发布，最初由谷歌创建。其Kubernetes本质上是开源的集群管理软件，用于部署、运行和管理Docker容器。通过Kubernetes，开发人员可以专注于应用程序本身，而不用担心提供它们运行时的底层基础设施。

Kubernetes使与部署和管理应用相关的所有工作都得以简化。Kubernetes会自动执行发布和回滚操作，并监控服务的运行状况以在出现不良影响之前阻止那些存在问题的发布 [34]。它还会对服务不间断地执行运行状况检查，重新启动有故障或停滞的容器，且只会在确认已成功启动服务时向客户提供服务。此外，Kubernetes还会自动根据利用率上下调节服务容量，确保在需要的时刻只运行需要的服务容器。与容器一样，Kubernetes支持对集群进行声明式管理，以便对设置进行版本控制。最重要的是，Kubernetes可在任何地方使用，开发者可以在本地部署、公有云部署以及混合部署之间进行协调。这让整个团队的基础架构可以覆盖位于任何位置的用户，让应用可以实现更高的可用性，让整个团队可以在安全与费用上取得平衡，一切都可根据项目具体需求定制。

2.5.3　Ansible

Ansible是一个开源的IT自动化引擎，可以消除运维人员工作中的重复性事务，还可以显着提高IT环境的可扩展性，一致性和可靠性 [35]。Ansible中的自动执行任务可被分为三种。配置在基础架构中设置所需的各种服务器是Ansible最常见的功能。Ansible也可用于配置管理，也就是更改应用程序、操作系统或设备的配置；启动和停止服务; 安装或更新应用程序; 实施安全政策; 或执行各种其他配置任务。Ansible还可以用于应用程序部署，通过自动将内部开发的应用程序部署到生产系统，使DevOps更容易更流畅。

与IT自动化工具例如Puppet和Chef想比，Ansible更易上手。Ansible使用简单的配置语言YAML来进行任务描述，而Puppet和Chef是用Ruby来描述其工作流，Ansible极大降低了学习成本，尤其是对那些不常写代码的系统管理员更为友好。Ansible部署还有着无代理这一特性，换句话说Ansible只需要系统具有Python和SSH，而无必在每个要管理的系统上安装代理客户端。Playbooks可用于在一个文件中执行多个任务。当开发者执行复杂任务时，例如分别配置可供MySQL运行与供Redis运行的服务器，使用Role来对服务器进行角色划分来进行差别部署。这种机制显著提高Ansible脚本的可复用性与可移植性。在Ansible执行任务流之前，Ansible会对要管理的主机进行信息收集并称之为Facts。通过对Facts进行收集，Ansible可以保证任务执行的幂等性，这样既可以提高整体流程的健壮性也可以提高复用性。

2.5.4 Consul

很多项目在向基于微服务的架构的进行演进时都会遇到如何进行服务发现这个问题，也就是如何跟踪所有已部署的服务。例如，部署在云中的业务服务Service A该怎么知道如何查找数据库服务Service B？一个直观的解决方案就是开发者可以将服务B的地址硬编码到服务A的配置中，如果服务B移动到不同的服务器或数据中心，则需要重新部署服务A。但是，在当今快速迭代的业务环境中，仅仅因为依赖服务重启而导致的停机时间是不可接受的。Consul为代表的服务发现工具可以有效解决这类问题 [36]。

Consul是一个分布式，高度可用服务网格系统，除了允许服务相互发现之外，Consul还允许开发者通过内置运行状况检查来监控群集运行状况，还可以充当分布式键值存储。Consul的由agent和server组成。agent允许服务公布它们的存在，而Consul的server收集agent共享的信息，并作为服务信息的中央存储库。agent在提供服务的节点上运行，并负责健康检查服务以及节点本身。Consul还可以与当前流行的容器管理平台进行配合使用，如Kubernetes和Azure Service Fabric。虽然Kubernetes和Service Fabric等容器编排工具都提供了自己的服务发现和健康检查机制，但Consul允许这些平台与位于其管理边界之外的服务集成。例如，在Kubernetes集群外部运行的Web服务或数据库，可以通过配置部署在Kubernetes上的Consul进行服务发现。

2.6 本章小结

本章节综述了基于区块链的知识产权管理平台在开发部署中用到的关键技术与解决方案，并结合系统中的具体业务场景，充分阐述选用每项技术的理由。

本章首先介绍了区块链技术的核心优势与常见实现，然后介绍了用来存储文件的星际文件系统，接着介绍项目中使用的前端框架与组件库，最后介绍了项目的微服务架构和DevOps工具链对开发部署的帮助。

第三章　系统需求分析与概要设计

3.1　系统整体概述

本系统主要实现了软件产品交付过程管理系统，主要是为了提供更好的软件交付体验、增强软件制品和软件测评报告的可信度。有软件产品需求的软件需求方在平台发布项目需求后，可以自行选择有开发资质的软件服务提供商来进行项目开发，也可以自行选择有软件测评资质的独立测评机构对软件产品进行测评并出具报告。具有软件开发资质的软件服务提供商可以通过上传特定格式的软件证书，把软件特征信息存储于区块链之中，既能保护自己的知识产权也使整个软件交付过程更流畅。

本系统可横向被分为两个部分，一部分是包含区块链数据存储在内的业务逻辑，另一部分是负责信息展示的、提供交互操作的Web前端部分。其中业务逻辑部分可以被进一步划分为用户认证与鉴权服务、文件系统服务、订单服务、验收服务。用户认证与鉴权服务的主要功能是控制访问权限，在业务逻辑层进行资源管理。文件系统服务主要提供了文件的上传下载，以及业务对象与XML文件之间的互相转换功能。订单服务负责订单实体的状态维护，验收服务可以根据用户需求对测评机构给出的验收报告与验收标准进行比较。

表 3.1: 系统涉众分析结果

涉众名称	涉众特征与期望
软件需求方	因自身的业务发展而产生了新的软件需求，在本平台发布订单后选择有资质的提供商与测评机构，依赖本平台完成交易。有一定的软件操作能力，有把软件需求文档化的能力，对软件制品有质量要求。
软件提供商	有一定的软件开发能力，通过本平台与软件需求方打成合作，可以通过本平台进行软件制品证书的上传，并通过上传证书保障自己的知识产权，有一定的二次开发能力，可以通过API与本平台进行对接。
软件测评机构	有一定的软件测试能力，可以根据对软件的测评结果出具有行业说服力和可行度的测试报告，有一定的二次开发能力，可以通过API与本平台进行对接。

本系统使用区块链存储订单数据、软件数据以及自动验收功能的测试数据，使用星际文件系统存储软件证书以及其他数据模型的可读格式化文档，通过TiDB对星际文件系统中的对象作持久化的缓存，同时提供设计良好的接

口SDK供用户接入使用，中间层采用了微服务的架构来减少功能模块之间的耦合。本系统使用了SpringBoot来构建微服务程序，每个服务独立部署。本系统在前端展示开发时选取了Vue框架生态，其丰富的组件库和高效的性能极大优化了开发流程提高了用户体验。

表 3.2: 功能需求列表

需求ID	需求名称	需求描述
R1	创建与更新订单	软件需求方可以根据自身需求创建订单，并填写交付时间等信息，执行上传需求文档、制定验收标准、更改软件提供方和软件测评方、更改订单状态等操作。
R2	查看订单列表	每个用户均可以看到与其相关的所有订单，订单被按照用户担任的角色进行了分类，每个分类下订单按照创建时间进行倒序排列。
R3	查看订单详情	每个用户均可查看到其有对应权限的订单的详细信息，可以下载需求文档等文件，软件提供方和软件测评方在订单未被确认前可自行退出订单。
R4	查看可加入的公开订单列表	用户可以在软件自由市场查看被公开的订单，并查看发布方的联系方式，主动加入订单。
R5	测试报告管理	软件测评方可以在订单详情界面，上传对应的测试报告，可多次上传对报告进行更新。软件需求方可以查看并下载测试报告，可以查看测试报告的修改历史。
R6	软件证书管理	软件开发方可以在订单详情上传软件证书，软件证书通过系统配套离线工具生成。软件需求方可以下载软件证书，可以查看软件证书变更历史。
R7	上传自软件验收结果	软件测评方可以通过SDK或API形式向指定站点传输其产生的测评结果，后台在接受到测评结果后会执行自动验收程序，并将测评结果与验收结果写入区块链。
R8	查看自动验收结果	软件需求方、软件测评方、软件开发方均可以在订单详情页查看自动验收的结果，页面中会详细展示各项标准是否达标。

3.2　系统涉众分析

本系统的涉众主要包括软件需求方、软件提供方和软件测评机构三种类型，其各自的特征与期望如表 3.1 所示。软件需求方因自身的业务发展而产生了新的软件需求，在本平台发布订单后选择有资质的提供商与测评机构，依赖本平台完成交易。一般软件需求方拥有把软件需求文档化的能力，希望通过本系统保障软件交付过程顺利开展。软件提供商是有较强软件开发能力的团队或组织，通过本平台与软件需求方达成合作，可以通过本平台进行软件制品证书的上传。软件测评机构是有较强软件测试能力的组织或机构，可以根据对软件的测评结果出具有行业说服力和可行度的测试报告。

3.3　系统需求分析

3.3.1　功能性需求分析

根据涉众特征和期望分析，可进一步进行针对涉众进行功能性分析。本系统主要需求如表 3.2 所示。

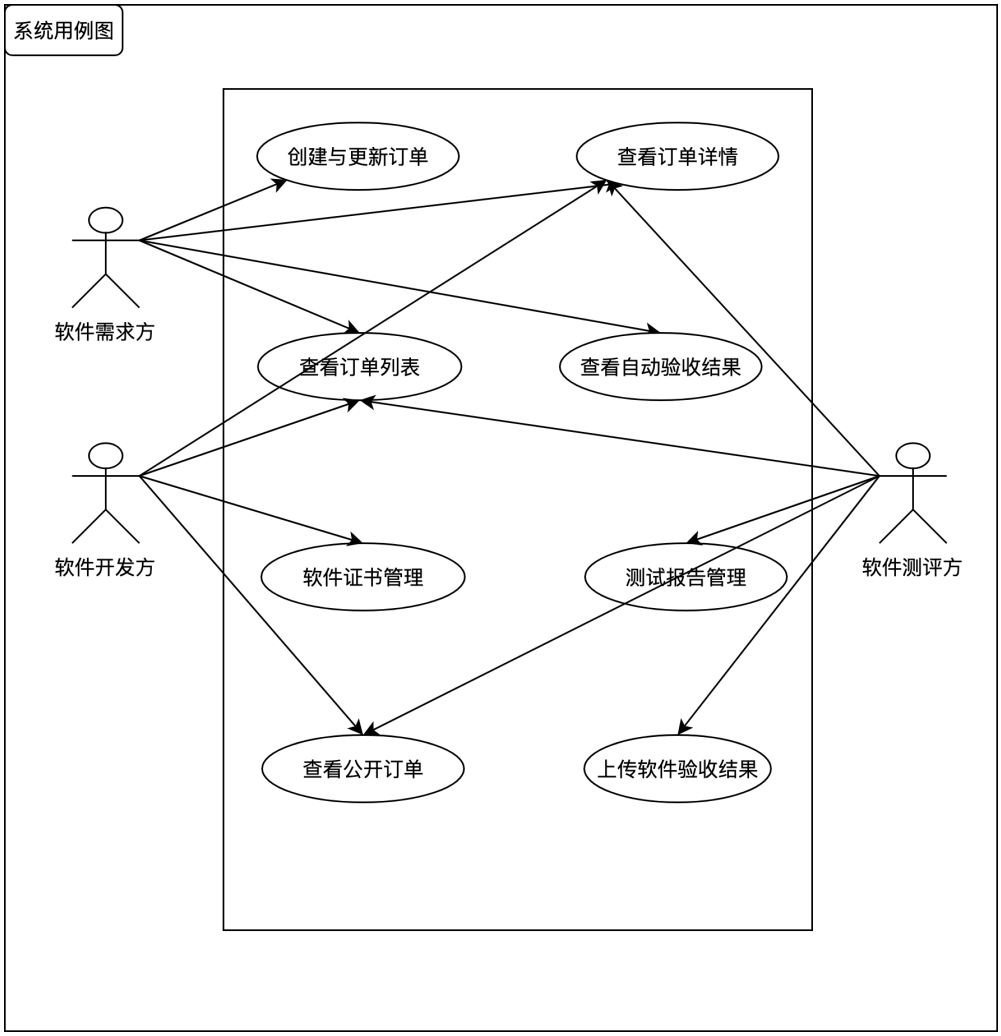

图 3.1: 系统用例图

在软件需求方的业务流程中，首先软件需求方要可以在系统中创建订单，在订单中填写必要的信息，上传必要的文档，对验收方式及其验收标准进行定性与定量。订单可以被设置为不同的可见性，软件需求方可为订单添加开发商和软件测评方，订单中的基本信息在确认后不再允许进行变更。在验证过程中，选择自动化验收的订单由系统后台自动验收等待交付，手动验收需要软件需求方自行下载软件测评方提供的测试数据与测试报告自行决定验收结果,验收成功

后订单进入交付阶段。软件交付阶段软件需求方确认软件提供商交付的软件与测评软件一致并在系统中进行订单状态更新后，交付阶段完成。

在软件测评方的业务流程中，软件测评方可以查看被公开的软件订单，软件测评方也可查看其承担测评任务的订单信息，可以查看其有权限的订单的详细信息并下载相关文件如需求文档等。在得到待测软件并对其进行测试后，软件测评方可以在前端页面上传文档形式的软件测评报告。对于有自动验收需求的订单，软件测评方也可以通过API或者SDK的接入形式，把测试结果上传至后台，由后台的业务逻辑配合区块链中的智能合约，对软件进行自动验收。

在软件提供方的业务流程中，软件提供方可以查看到被公开的软件订单，软件提供方可以查看自己参与开发的订单，可以查看其有权限的订单的详细信息并下载相关文件如需求文档等。软件提供方由软件需求商加入订单，在订单被确认之前可自由退出。软件提供商开发软件产品完成后，需要上传自己的软件证书以供软件需求方和测试报告中的软件信息进行对比认证。

表 3.3: 系统非功能性需求列表

需求名称	需求描述
可用性	系统不应存在单点故障;系统要保障全年99%的可用性，若因意外导致系统崩溃或服务不可用，应在15分钟内恢复或启动备份集群以提供服务。
性能	在网络连接正常的情况下，后台接口返回数据的时间应小于200ms，特性复杂的查询和写操作应小于500ms。
安全性	系统应采用严密的身份认证，对所有来源的请求进行辨别。
易用性	系统接口设计应符合一般的API设计风格，命名方法应选用较为通用流行的方法；前端展示页面需要有合适友好的提示语，符合一般的人机交互认知。
兼容性	系统应能在一般意义上的服务器上部署；系统提供的API和SDK应尽可能通用。
伸缩性	当用户量、并发量超过系统负载时，应可以快速部署多节点、多集群，且对上游的客户透明。
可拓展性	系统内部对关键业务方法抽象出接口，如文件系统、缓存系统，以便实现更新、升级与替换。

3.3.2　非功能性需求分析

由于本系统被设计用来服务真实的软件交付流程，所以对系统的可用性、稳定性、可拓展性以及性能等方面均有一定的要求。经过仔细分析，本系统需要满足的非功能性需求如表 3.3所示，为了应对业务增长带来的访问请求压力，系统需要具备快速水平拓展的能力，每个节点自身需要有一定自我恢复能力以避免系统不可用；为了保护数据的可信与可溯源，系统需要在多层次对读取数

据权限与修改数据权限均进行控制；面向开发者的SDK和API需要有较强的可读性，面向一般用户的Web 展示页面要有较强的易用性；本系统采用了分布式的架构，部署环境复杂多变，系统需要具备在一般的服务器中快速部署的能力；为应对文件系统、数据库选型等方面的变化，本系统需要做好业务层面的接口抽象，方便多种业务实现间的切换，以提供整个系统的易修改性。

3.3.3　系统用例图

根据上述系统功能性需求描述，得到系统用例图如图 3.1所示的8个用例，分别是创建订单与更新订单、查看订单列表、查看订单详情、查看可加入的公开订单列表、测试报告管理、软件证书管理、上传自动验收订单、查看自动验收结果。

表 3.4: 创建与更新订单用例描述

ID	UC1
名称	创建与更新订单
参与者	软件需求方
触发条件	软件需求方点击创建订单或更新订单
前置条件	软件需求方必须已被识别和授权
后置条件	无
优先级	高
正常流程	1. 软件需求方登录完成； 2. 系统界面左侧导航栏展示创建订单； 3. 软件需求方填写订单名称，确定交付日期，上传需求文档，指定验收形式与验收标准； 4. 软件需求方可提交订单，系统将订单内容写入区块链； 5. 软件需求方在未确认订单之前，可以增删软件提供方和软件测评方，可以更新需求和其他信息。
特殊需求	只有处在草稿状态的订单允许修改信息

3.3.4　系统用例描述

创建与更新订单用例描述如表 3.4所示。软件需求方登录本系统，在订单创建页面进行订单基础信息填写，其中包括订单名称，订单交付时间、验收方式、是否自动化验收等，并且上传详细的需求文档。软件需求方可以通过系统对草稿状态的订单进行修改，可以自行添加或者移除软件提供商和软件测评机构。在软件进入开发状态后，订单数据写入区块链中，基础信息无法再修改。

表 3.5: 查看订单详情用例描述

ID	UC2
名称	查看订单详情
参与者	软件需求方、软件测评方、软件提供方
触发条件	软件需求方、软件测评方、软件提供方在订单列表中选中一个订单
前置条件	选中订单的用户需是订单相关的软件需求方、软件测评方或软件提供方中的一个
后置条件	无
优先级	高
正常流程	1. 用户在其可查看的订单列表中选择一个订单； 2. 系统返回订单信息，并展示订单信息； 3. 用户点击该订单中的需求文档； 4. 系统通过浏览器将文档下载至用户的默认下载路径； 5. 用户点击查看订单中的验收标准； 6. 系统返回验收标准的详细信息并在界面展示； 7. 用户点击查看该订单修改历史； 8. 系统返回该订单所有历史数据，并按照时间顺序在页面展示。
异常流程	1. 用户通过地址直接访问无权限的订单资源； 2. 系统拒绝用户访问该订单，并提示导航至订单列表页面。

查看订单详情是本系统所有用户均需要使用到的重要功能，软件测评方和软件提供方通过该功能了解软件需求方的要求，各方还可以通过查看订单修改历史功能查看每一版本的软件信息，做到信息的可溯源。查看订单详情用例描述如表 3.5所示。

表 3.6: 查看订单详情用例描述

ID	UC3
名称	查看订单列表
参与者	软件需求方、软件测评方、软件提供方
触发条件	用户登录或点击导航栏首页
前置条件	用户必须已被识别和授权
后置条件	无
优先级	高
正常流程	1. 用户正常登录本系统； 2. 系统显示用户以不同身份参与的三种订单列表； 3. 用户点击查看其中一种类型的列表； 4. 系统展示用户作为软件需求方、软件测评方或软件提供方参与的所有订单，并按照订单状态和时间进行排序； 5. 用户可以切换查看类型； 6. 系统返回对应类型的订单列表； 7. 用户可以根据订单名称对其参与的订单进行模糊搜索； 8. 系统返回根据订单名称模糊查找的相关订单。

查看订单列表是本系统用户在登录后首先用到的功能，考虑到本项目中每个用户可在不同订单中担任不同的角色，故在界面中应根据用户在订单中承担的角色对订单进行分类，即可以每位用户在登录后均可查看其发起的订单，其参与测评的订单，其负责提供软件的订单。在选择某一类订单进行查看后，该类订单根据时间和状态进行排序展示。用例描述如表 3.6所示。

表 3.7: 查看公开订单用例描述

ID	UC4
名称	查看订单详情
参与者	软件需求方、软件测评方、软件提供方
触发条件	用户点击查看公开订单
前置条件	用户必须已被识别和授权
后置条件	无
优先级	高
正常流程	1. 用户点击导航栏中的公开订单； 2. 系统返回处在可查看状态的公开订单，订单按照发布时间倒序排列； 3. 用户点击列表中的订单； 4. 系统返回被选中订单的部分公开信息。
异常流程	1. 用户通过地址直接访问无权限的订单资源； 2. 系统拒绝用户访问该订单，并提示导航至订单列表页面。

查看公开订单的功能主要适用于订单创建初期，被公开的订单可以被所有的用户在软件公开市场查看，用户可以申请作为软件测评方或者软件提供方被关联到订单中，处于开发及后续阶段的订单无法被在此处被查看。用例描述如表 3.7所示。

软件证书管理功能主要面向软件提供方，在完成软件产品的开发后，软件提供方可以上传其软件的证书至系统，再讲证书与其匹配的订单关联。软件提供方可以在本系统中上传、更新和查看软件证书。软件证书的内容主要由软件签等部分组成。软件需求方也可以在订单详情页中下载订单所对应的软件证书。用例描述如 3.8所示。

测试报告管理功能主要面向的是软件测评方和软件需求方。软件测评方在完成对软件的测评后可在订单详情界面上传软件测评报告，测评报告会被存入星际文件系统。软件需求方可以在订单详情下载对应软件测评报告。用例描述如表 3.9所示。

表 3.8: 软件证书管理功能用例描述

ID	UC5
名称	查看订单详情
参与者	软件提供方、软件需求方
触发条件	用户点击导航栏中的证书管理
前置条件	用户必须已被识别和授权
后置条件	无
优先级	高
正常流程	1. 软件提供方点击证书管理 2. 系统返回软件提供方上传的所有软件证书 3. 软件提供方上传证书 4. 系统返回证书结果 5. 软件需求方在订单详情界面，点击下载软件证书 6. 系统通过浏览器下载软件证书至客户默认下载路径
异常流程	1. 用户通过浏览器访问其无权限的软件证书路径 2. 系统拒绝其请求，并导航至用户首页

上传软件测评数据功能主要面向的是软件测评方。针对有自动验收需求的订单，软件测评方在完成对软件的测评后需要通过API或者SDK将软件测评数据传至系统，系统将软件测评数据存入区块链并执行自动验收的自动合约。用例描述如表 3.10所示。

表 3.9: 测试报告管理用例描述

ID	UC6
名称	测试报告管理
参与者	软件测评方、软件需求方
触发条件	软件测评方完成软件测评
前置条件	用户必须已被识别和授权，软件测评方已被添加至相关订单
后置条件	无
优先级	高
正常流程	1. 软件测评方完成软件测评，并形成软件测评报告； 2. 软件测评方在订单详情界面点击上传软件测评报告； 3. 系统通过浏览器调出文件选择栏； 4. 软件测评方选择本地测评报告进行上传； 5. 系统后台将软件测评报告存储至星际文件传输系统，并更新订单系统； 6. 软件需求方在订单详情界面点击下载测评报告； 7. 系统通过浏览器将被选测评报告下载至软件需求方本地默认下载路径。

表 3.10: 上传软件测评数据用例描述

ID	UC7
名称	上传软件测评数据
参与者	软件测评方
触发条件	软件测评方完成软件测评
前置条件	用户必须已被识别和授权，软件测评方已被添加至相关订单
后置条件	无
优先级	高
正常流程	1. 软件测评方完成软件测评，通过API或者SDK的方式将软件测评数据传至系统； 2. 系统在进行必要的校验后将软件测评数据写入区块链； 3. 系统返回软件测评数据已写入成功或写入失败与失败日志。 4. 系统根据订单中的软件自动验收标准和软件测评方上传的软件测评数据执行自动验收的智能合约。

查看自动验收结果主要是面向软件需求方。在软件测评方上传软件测评数据以及区块链系统完成自动验收流程后，软件需求方可在软件订单详情中查看每项验收指标的测评情况与自动验收结果。用例描述如表 3.11所示。

表 3.11: 查看自动验收结果用例描述

ID	UC8
名称	查看自动验收结果
参与者	软件需求方
触发条件	自动验收流程完成
前置条件	用户必须已被识别和授权
后置条件	无
优先级	高
正常流程	1. 软件需求方点击查看指定订单详情； 2. 系统返回指定订单详细信息； 3. 软件需求方点击查看自动验收详细结果； 4. 系统返回指定订单相关的自动验收标准与测评结果，并显著标记未达标测评结果。

3.4　系统概要设计

根据需求分析中总结出的功能需求和非能性需求，本小节针对系统进行总体架构设计。通过对系统进行服务划分与总体架构设计，我们能够展示出系统的整体框架与层次结构，划分出系统中各子服务的相应职责与子服务之间的联系，将需求模型进一步转化为物理系统模型。在描述架构设计的过程中，本小节借助逻辑视图、进程视图、开发视图和物理视图进行描述。

3.4.1　系统服务划分与架构设计

本系统的系统架构图如图 3.2所示，本小节从设计与技术的角度进行介绍。为了保证业务层面的高内聚低耦合，开发阶段的快速迭代与个性化部署，本系统采用了微服务架构，选用consul作为服务发现与服务注册的中间件。本系统提供了丰富的业务API与集成SDK，最大限度得丰富了用户的使用方式。

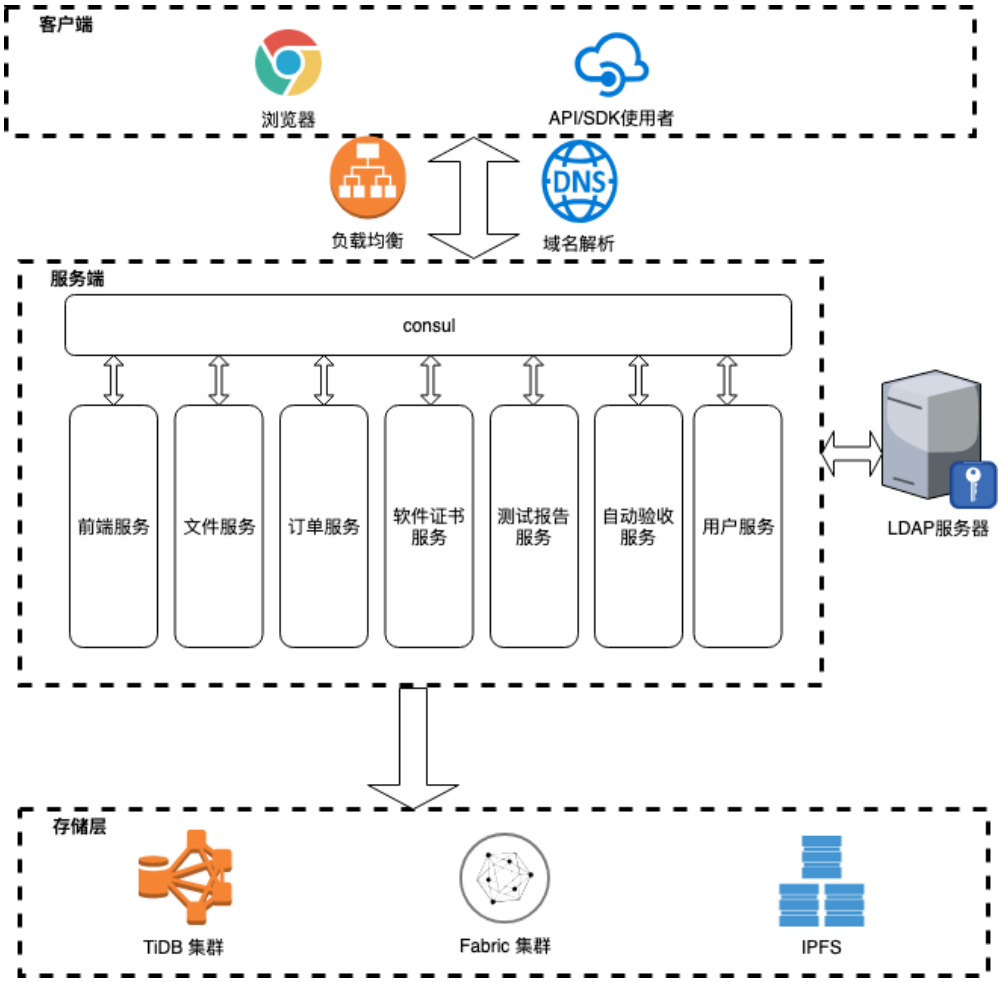

图 3.2: 系统架构图

前端服务采用了Vue框架及其生态中的路由管理器等组件，其优雅的框架设计和出色的性能既提高开发效率也带来高效的渲染效率。本系统选用了饿了么开源出的elementUI组件库配合Vue框架进行信息展示，elementUI是饿了么从其自身使用Vue框架过程中抽离出的一套标准组件库，和Vue框架及其生态有着天然的默契。前端服务在项目启动时向consul发送其地址信息进行服务注册，并获得其他业务服务的地址与信息。前端服务通过HTTP请求对其他业务提供的资源与接口进行访问。

除了前端服务外，服务端的其他服务均采用了SpringBoot框架进行业务逻辑开发，SpringBoot框架可拓展性强且支持众多服务组件。根据常见的微服务划分标准，结合对系统用例进行分析，本系统中业务服务可被划分为用户服务、订单服务、软件证书服务、测试报告服务、自动验收服务、文件服务，每个服务均独立使用一个容器资源，详细的Kubernetes部署方案见本文第五章相关内容。所有业务服务在启动时均向consul集群进行服务注册，并得到相关的服务地址以完成后续的业务逻辑。所有的业务服务均采用较为传统的分层设计，较大程度得减少业务耦合且有良好的可拓展性。在用户服务的设计过程中，出于兼容存量系统与减少重复开发的目的，本系统选择接入了LDAP来存储于认证用户信息 [37]。

系统采用了Kubernetes的ingress-nginx组件为负载均衡器，同时ingress-nginx还提供了集群外部访问Kubernetes集群内部服务的能力，强力支持了本系统对外提供API这一功能。

为了减少区块链的存储压力，本系统中仅订单信息，软件信息，自动验收相关信息存储于区块链之中，其他例如需求文档等均存储于IPFS之中，同时为了优化IPFS中数据的读取效率，采用了TiDB和redis作为其缓存，可通过配置文档进行选择切换。

3.4.2　架构视图

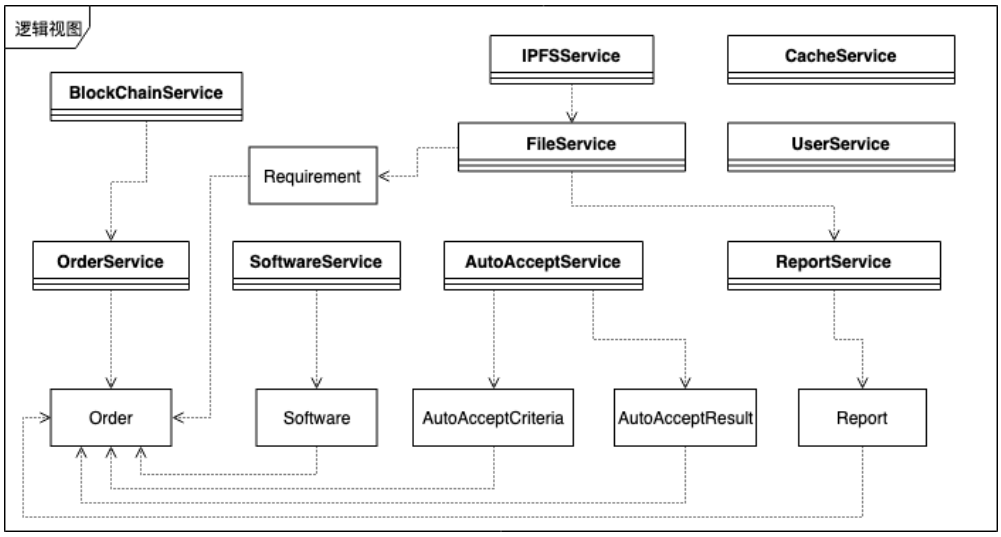

图 3.3: 逻辑视图

Philippe Kruchten 教授于1995年提出的“4+1”视图法是目前在软件架构领域工程运用中最富盛名且运用最广的一种方法 [38]。该视图法中“1”代表场景

视图，一般用于描述用户场景，本文的图3.1系统用例图就是该视图在UML中的实例。余下的“4”代表着逻辑视图、开发视图、进程视图以及物理视图，它们是从不同角度对系统架构进行的描述。本小节将分别给出本系统架构对应的逻辑视图、开发视图、进程视图以及物理视图，对系统的开发、运行状态、以及部署等进行描述。

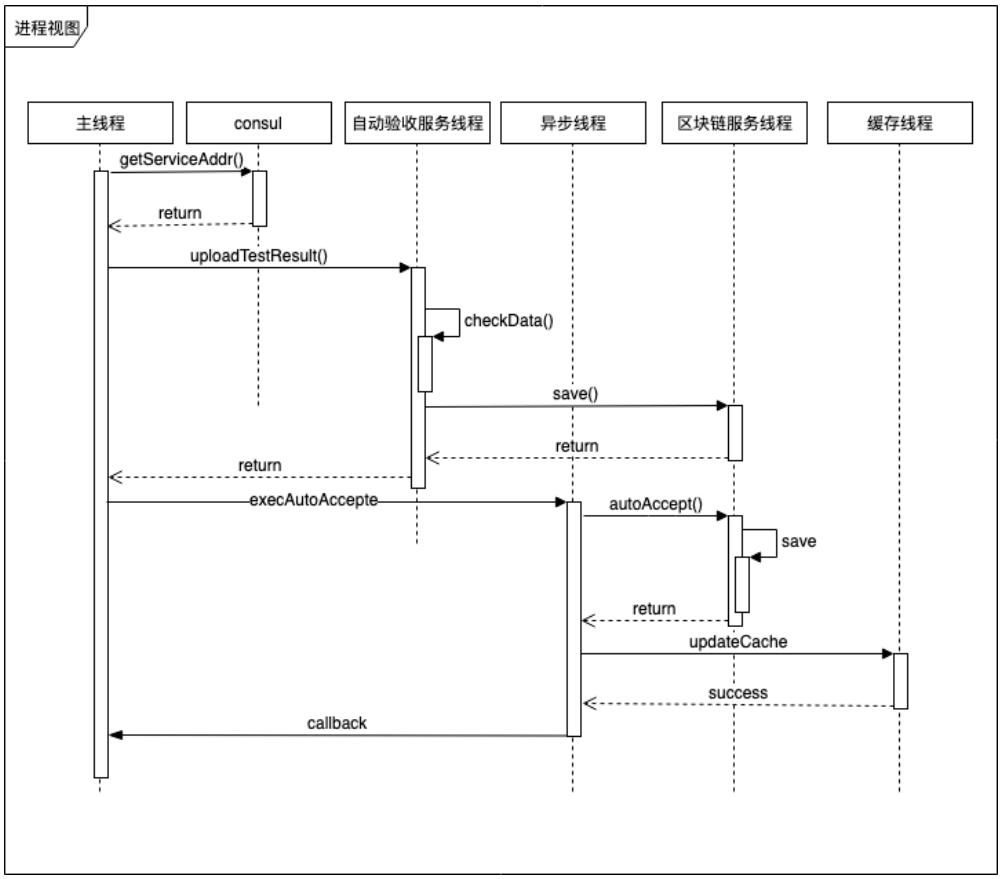

图 3.4: 进程视图

逻辑视图主要是从功能性需求上描述系统，用户根据该视图即可得知系统服务是否满足其的功能需求。在面向对象设计方法中，类图通常被用来描述类的结合以及他们之间的逻辑关系，因此类图可作为逻辑视图的载体。如图3.3所示，本系统进行需求分析与细化，系统中的实体被抽象为：订单（Order）、软件（Software）、测评报告（Report）、需求（Requirement）、自动化验收标准(AutoAcceptCriteria)、自动化验收结果（AutoAcceptResult）等，系统中的基础服务被抽象为用户服务（UserService）、文件服务（FileService）、订单服务（OrderService）、软件服务（SoftwareService）、测评报告服务（ReportService）、自动验收服务（AutoAcceptService）等。UserService提供用户登录与注销等功

能。FileService提供了文件上传与下载等功能。SoftwareService提供了软件信息上传、更新和查找等功能。OrderService提供了订单创建和更新状态等功能功能。ReportService提供了测评报告的上传和下载等功能。AutoAcceptService提供了测试结果上传、自动验收和验收对比结果查询等功能。

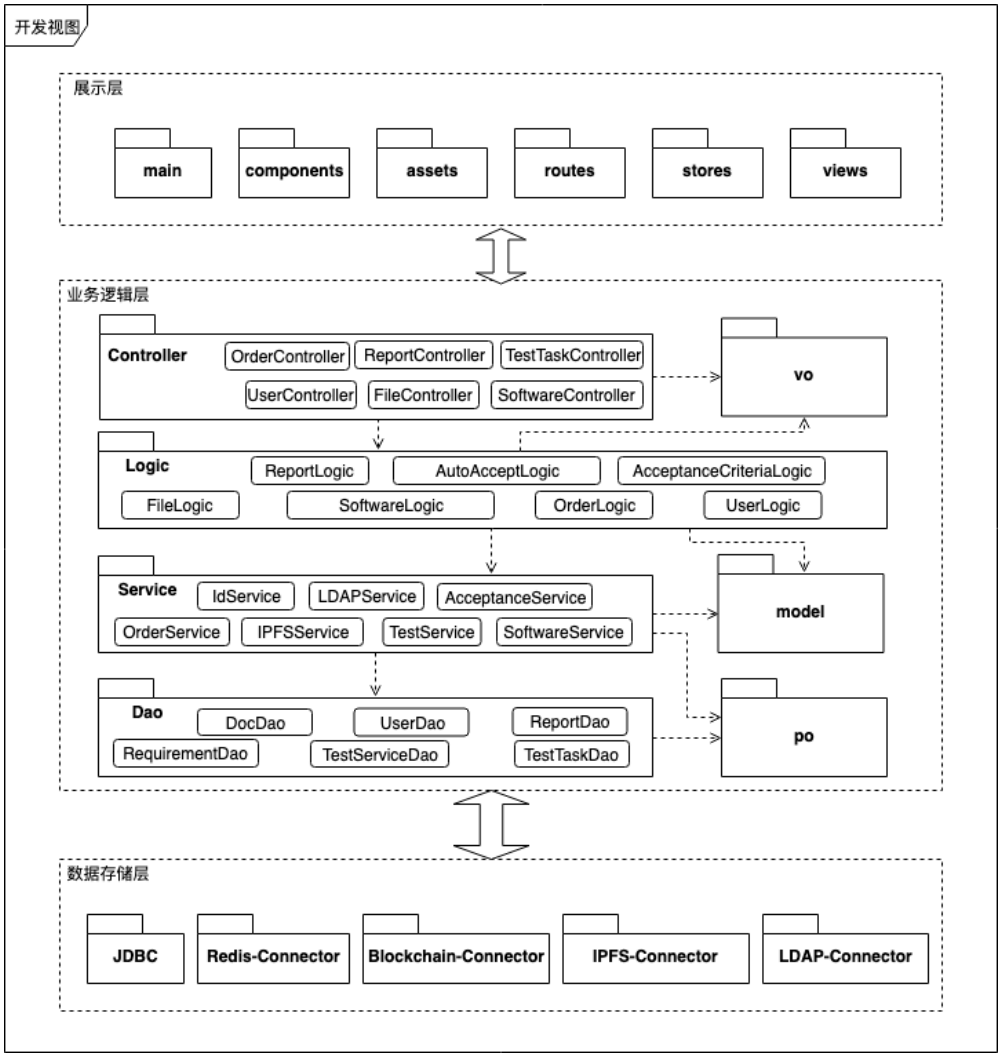

图 3.5: 开发视图

进程视图用于描述系统在并发性、分布性、系统完整性上的问题，关注逻辑视图中的主要抽象是如何与进程对应，也就是关注对象是被哪个进程控制与执行以及进程间的互相通信。如图 3.4所示，本系统中，当某服务意图访问其他服务时，会先向consul查询其通信地址，再根据通信地址与API进行服务调用。在执行自动验收的过程中，自动验收服务线程先进行数据校验，再同步调用区块链服务将测评结果数据存至区块链，当区块链服务返回存储结果后，自动验

收服务线程将结果返回至主线程。主线程启动异步线程调用自动验收服务，异步线程会将结果存储至区块链，异步调用完成后回调主线程的事件处理函数，结果会被更新至缓存数据库。

开发视图的受众主要是熟悉软件开发的工程师，该视图关注软件开发中面临的模块管理与包组织结构。图 3.5展示了本系统的包组织结构。展示层中，main包中存放入口代码，component包中是可复用组件，assets包是资源目录，routes存放路由数据，store包中存放应用级数据，views包是项目页面目录。业务逻辑层主要采用了经典的分层架构，Controller包负责管理路由请求，Logic包处理较为复杂的业务逻辑，Service包对Dao层提取出的数据进行简单封装，vo、model和po是数据在不同业务层次中的不同形态。数据存储层是一些业务独立的连接中间件，负责建立系统与不同存储系统之间连接。因TiDB支持Mysql协议，故直接使用较为成熟的JDBC框架。

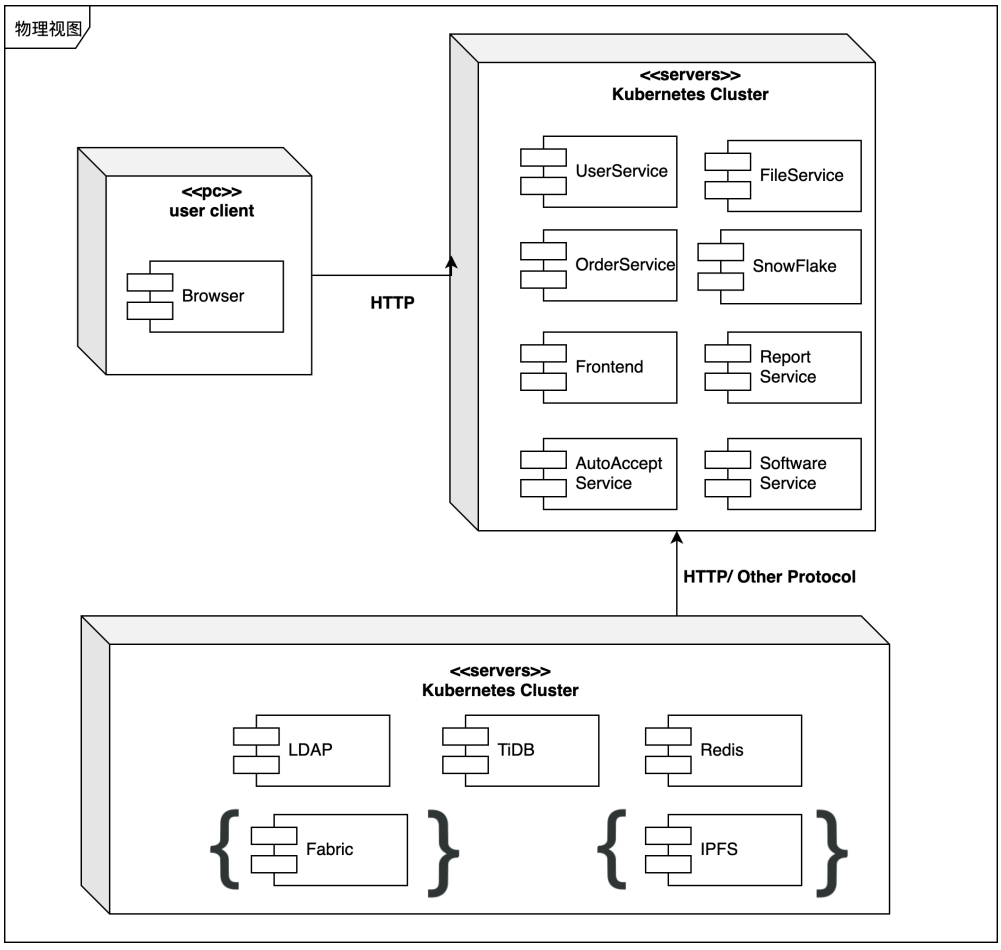

图 3.6: 物理视图

物理视图是运维人员理解系统的重要途径，是表达软件与硬件之间映射的重要方法。图 3.6是本系统的物理视图。用户通过浏览器对本系统资源进行访问。微服务中的每个服务均由一个Docker容器实例运行，其中SnowFlake为Facebook开源的唯一ID 生成器。同一Kubernetes集群中，服务间调用均使用内部IP。LDAP、TiDB、Redis、IPFS集群和Fabric集群同样通过Kubernetes集群进行部署。业务服务与数据存储之间通过HTTP协议或其他自定协议进行通信。

3.5　持久化模型设计

本系统中采用Fabric联盟链来保证数据可信性与可溯源性，为了减轻链上存储负担，本系统引入星际文件传输系统来存储文件，这些文件既包括需求文档等用户上传文件如需求文档，也包括由对象转成的XML类型文件。星际文件系统的引入，使得区块链上的可以仅存对象或者文件的哈希值也能保证其唯一性与不可篡改性。从XML文件解析到Java对象占用一定内存与时间资源，引入Redis或TiDB作为数据缓存恰好可以缓解这一问题。Redis是非持久化内存数据库，启动时需要重新加载缓存但运行时速度较快；TiDB是持久化NoSQL，启动时无需重新加载但运行时速度相比Redis较慢。本系统中可以通过配置文件来指定缓存使用Redis或者TiDB。本小结将介绍区块链数据模型与IPFS对象模型。

3.5.1　区块链数据模型设计

本系统选用HyperLedger Fabric作为区块链基础设施，CoachDB是HyperLedger Fabric中默认的State Database，区块链中操作到的业务数据都被存在CoachDB中。CoachDB是完全局域RESTful API的键值数据库 [39]。本系统中，由SnowFlake生成出全局唯一ID作为业务数据键值对中的键（Key），业务对象如Order本身作为键值对中的值（Value）。本小节枚举存储在区块链中的业务对象，主要是订单数据、软件证书数据、自动验收标准数据、测评数据以及自动验收结果数据。订单数据与软件证书数据中存储订单属性或属性对象在星际文件系统中的位置信息。自动验收相关数据应需要使用智能合约进行验收故也要存入区块链中。

表 3.12是订单数据存储在区块链中的字段描述。订单对象是本系统中最为核心、关联信息最多的对象。为了减轻区块链服务的存储负担，减少冗余存储，验收标准、软件需求、测试任务和测试报告等均只存储其hash值，对象本身以文件形式存储在IPFS 中。

表 3.12: 区块链对象Order属性表

字段	含义	类型	备注说明
id	订单Id	string	键属性
name	名称	string	
user	软件需求方	user	
deliveryTime	预定交付时间	long	
suppliers	软件提供方	list<user>	
testOrganizations	软件测评方	list<user>	
acceptanceCriteriaHash	验收标准对象hash	string	此hash值即验收标准对象在IPFS中的地址
requirementHash	软件需求hash	string	IPFS地址
softwareHashes	交付软件hash	list<string>	
status	订单状态	int	
testTaskHash	测试任务hash	string	IPFS地址
reportHash	测试报告hash	List<string>	IPFS地址
isPublishOrder	订单公开性	boolean	

表 3.13是软件数据存储在区块链中的字段描述。主要记录上传者与软件之间的映射，软件证书实体被存储在IPFS中，软件数据对象中仅持有其hash。

表 3.13: 区块链对象Software属性表

字段	含义	类型	备注说明
id	订单Id	string	键属性
name	名称	string	
user	软件提供方	user	
uploadTime	上传时间	long	
softwareCertHash	软件证书hash	list<string>	IPFS地址
description	描述	string	
contributionMap	软件贡献字段	map<user, object>	拓展字段，用于描述软件者对软件的贡献程度，非必须

表 3.14是自动验收标准数据存储在区块链中的字段描述。只有当用户为某订单选择了自动验收时，该订单才会有对应的自动验收标准对象存于区块链之中。该对象主要存储了自动验收标准与订单之间的映射关系，acceptanceMap存储了每项验收标准的细则。standardObject是标准名、比较符和比较值组成的三元组，此三元组设计被用来支持自动验收。

表 3.14: 区块链对象AcceptStandard属性表

字段	含义	类型	备注说明
id	自动验收标准id	string	键属性
orderId	名称	string	
acceptanceMap	验收标准细则	map<string, standardObject>	

表 3.15是测评结果数据存储在区块链中的字段描述。软件测评方在对软件进行测评生成测评结果数据后，可通过本系统提供的API接口将数据写入区块链。测评结果对象存储了订单信息与测评结果之间的映射，其中resultMap的值对象出于兼容性的考虑采用了string类型。

表 3.15: 区块链对象TestResult属性表

字段	含义	类型	备注说明
id	测评结果id	string	键属性
orderId	所属订单id	string	
testTaskHash	测试任务hash	string	IPFS地址
softwareHash	测评软件hash	string	IPFS地址
resultMap	测试结果	map<string, string>	

表 3.16是自动验收结果存储在区块链中的字段描述。在由智能合约完成自动验收过程后，自动验收结果被生成并写入区块链。其中resultMap的键与TestResult对象中的resultMap键一致。

表 3.16: 区块链对象AutoAcceptResult属性表

字段	含义	类型	备注说明
id	测评结果id	string	键属性
orderId	所属订单id	string	
testResultId	测评结果id	string	
softwareHash	测评软件hash	string	IPFS地址
resultMap	自动验收结果明细	map<string, boolean>	

3.5.2 IPFS对象模型设计

本系统中，有两种文件存储于IPFS中，一种为需求文档、软件证书和测试报告等由用户直接上传的文件，一种为测试任务和验收标准等由Java对象格式

化得到的XML文件。本小节重点描述后一种对象模型。TiDB或者Redis中缓存内容就是在IPFS对象模型的基础上添加IPFS地址作为键的值。

表 3.17是测评报告对象在IPFS的XML标签属性表。该对象描述描述了测评报告的各项属性，其中reportHash为测评报告文件存储在IPFS中的摘要。

表 3.17: IPFS对象模型Report属性表

字段	含义	类型	备注说明
uploader	软件测评方	string	对象序列化
orderId	所属订单id	string	
reportHash	测评报告文件hash	string	IPFS地址
createTime	对象创建时间	long	

表 3.18是验收标准存储在IPFS的XML标签属性表。该对象描述描述了订单的验收细节，其中criteriaJSONString只在订单需要自动验收时不为空。为提高字段的可读性，此处将自动验收标准转为Json字符串存储。

表 3.18: IPFS对象模型AcceptanceCriteria属性表

字段	含义	类型	备注说明
description	验收标准描述	string	
isAutoAcceptable	是否自动验收	boolean	
criteriaJsonString	验收标准	string	对象存储为Json字符串
orderId	所属订单id	string	IPFS地址
creatTime	对象创建时间	map<string, boolean>	

表 3.19是需求对象存储在IPFS的XML标签属性表。该对象描述软件测评方上传需求文档行为的相关信息，反应订单与需求文档之间的映射关系。

表 3.19: IPFS对象模型Requirement属性表

字段	含义	类型	备注说明
uploader	软件需求方	string	对象序列化
orderId	所属订单id	string	
requirementHash	需求文档hash	string	IPFS地址
createTime	对象创建时间	long	

表 3.20是测试服务存储在IPFS的XML标签属性表。该对象记录软件测评方提供的测试服务对象。

表 3.20: IPFS对象模型TestService属性表

字段	含义	类型	备注说明
name	服务名称	string	
description	服务描述	string	
linkedUrl	回调地址	string	可选
ServiceProvider	测评软件hash	string	对象序列化
isActive	当前是否活跃	boolean	
createTime	创建时间	long	

表 3.21是测试任务存储在IPFS的XML标签属性表。软件需求方可为订单选择不同测试服务，该对象记录了测试服务与订单之间的关系。

表 3.21: IPFS对象模型TestTask属性表

字段	含义	类型	备注说明
testServiceHash	测试服务hash	string	
orderId	所属订单id	string	
Status	测评结果id	string	
createTime	创建时间	long	

3.6　本章小结

本章节主要对系统从涉众、功能性需求和非功能性需求进行分析并形成图表。在系统用例图和系统整体架构图的基础上，对整个系统架构从逻辑视图、进程视图、开发视图和物理视图进行多角度拆分与描述；接下来介绍了本系统中使用到的持久化技术，重点描述区块链数据模型和IPFS对象模型的设计思路与字段含义。

第四章　详细设计与实现

4.1　用户服务详细设计与实现

4.1.1　用户服务架构设计

用户服务负责系统用户登录与权限控制。用户服务架构设计如图 4.1所示。用户服务在服务启动时向consul写入地址信息与端口信息，处理由展示端发出的登录登出请求，也为系统中其他服务提供权限校验功能。

为了兼容存量系统，本系统接入了LDAP作为用户服务底层实现。服务端接受到登录请求后根据consul提供的地址信息对LDAP 服务器进行访问，获取用户凭证，并将用户登录信息存储在Redis缓存中。为了提高系统数据安全性，用户对Fabric集群进行数据读写前均由用户服务进行权限验证，并将Fabric返回的用户凭证缓存至Redis中。

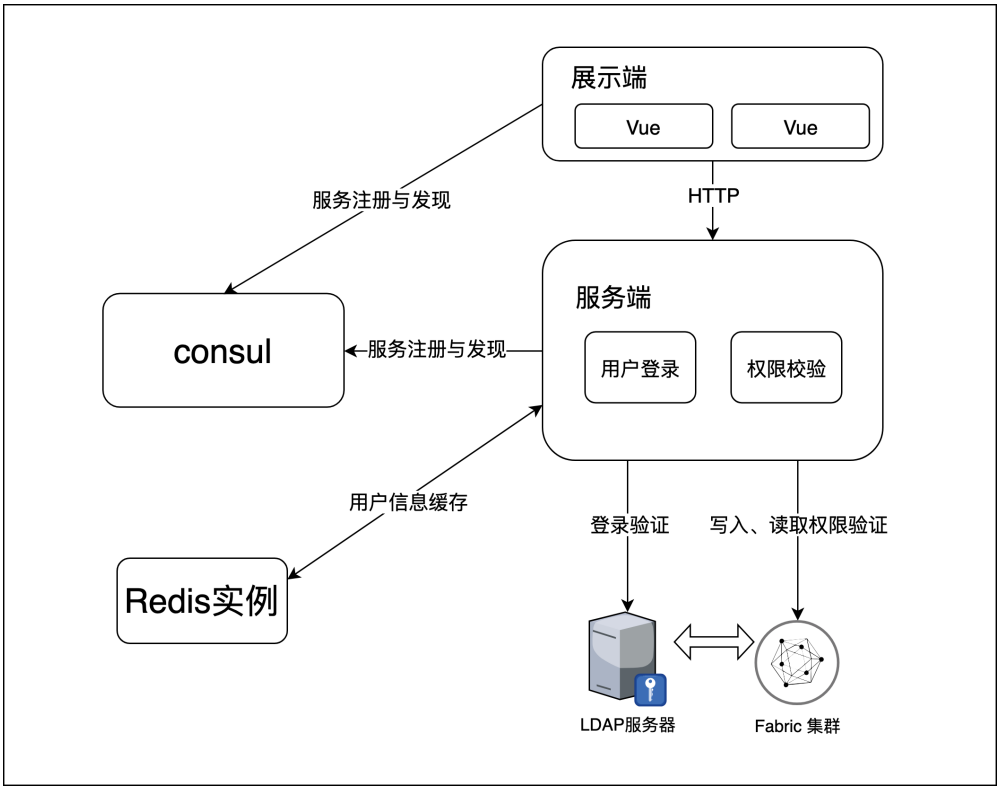

图 4.1: 用户服务架构图

4.1.2　用户服务核心类图

如图 4.2所示是用户服务核心类图设计，UserController是对外提供服务的控制器，主要提供用户登录登出与获取用户凭证功能。UserLogic对参数进行有效性检验，依赖LdapUserService 和FabricUserService登录功能与获取凭证功能，此外UserLogic还负责处理User与UserVO的映射转换处理。User与Person对象的转换处理由UserService中的私有方法进行实现。LdapUserService通过调用LdapUserDao提供的查询接口实现用户登录，并将用户登录状态存入Redis，FabricUserService通过调用FabricUserDao提供的查询接口实现获取用户凭证。UserVO、User和Person三个对象是同一数据在不同服务粒度层次的不同抽象。

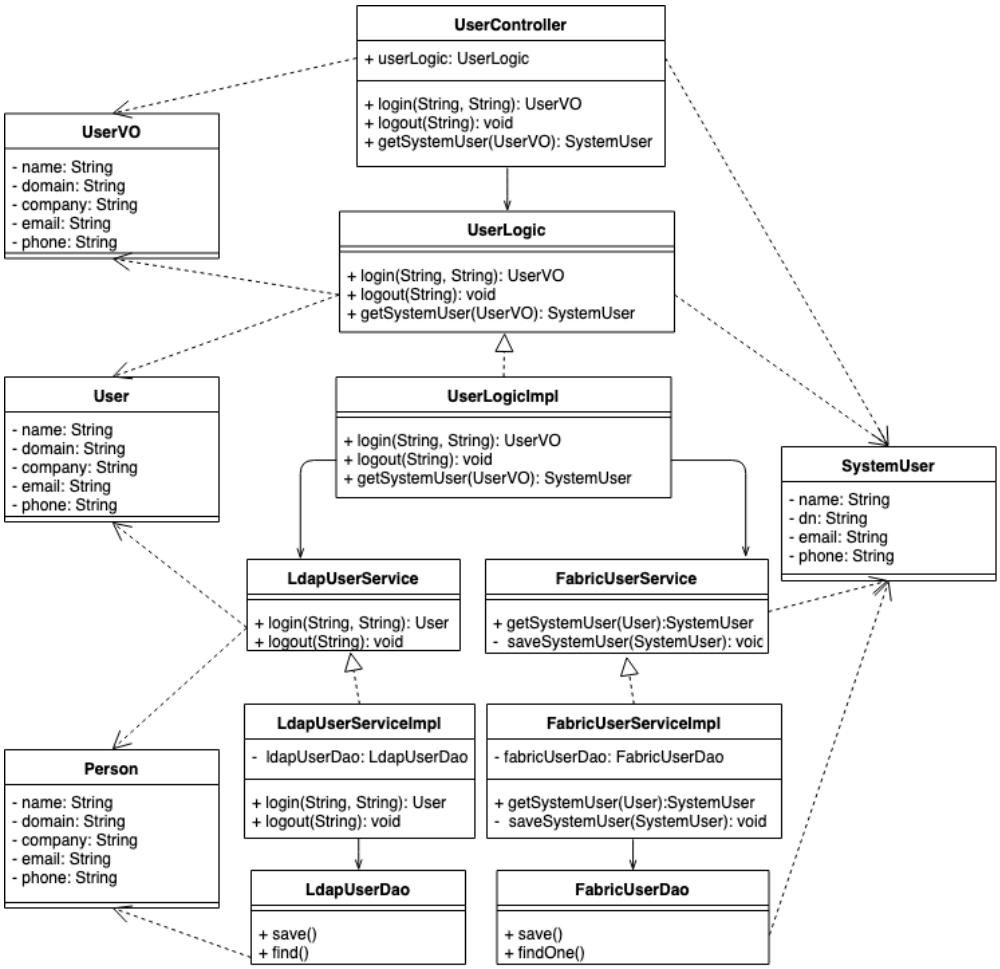

图 4.2: 用户服务核心类图

4.1.3　用户登录顺序图

如图 4.3所示是用户服务提供登录服务和获取凭证服务的服务调用顺序，是系统对用户权限进行管理的核心。在收到用户请求登录请求时，UserController会调用UserLogic中的登录接口，UserLogic进行必要的参数有效性检验后调用LdapUserService 定义的接口，登录校验完成后UserLogic将User对象映射转换为UserVO对象返回至控制器层。

Service层接口LdapUserService 和FabricUserService 在每次查询时均会先请求缓存，如缓存内无对应数据则调用对应Dao层接口且将结果更新至Redis缓存。UserController调用用户登出时亦会删除对应User缓存。

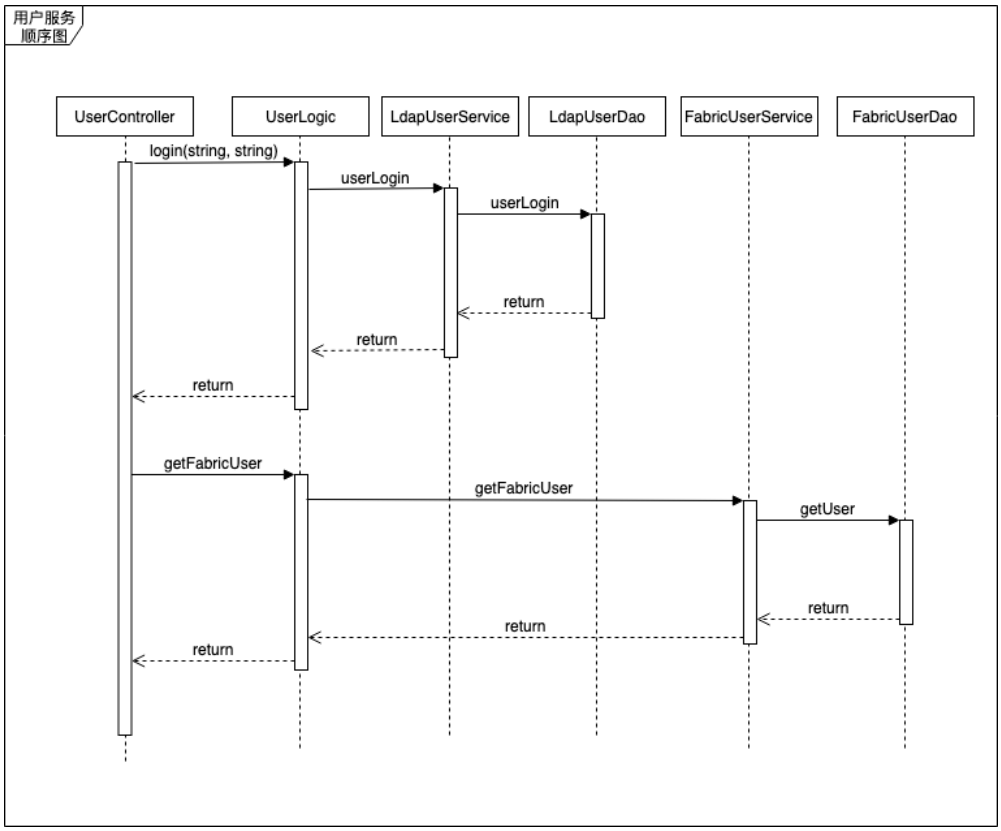

图 4.3: 用户服务顺序图

4.1.4　用户登录关键代码

如图 4.4所示为用户服务登录功能的主要逻辑，涉及参数校验、缓存读取与更新。为不影响阅读，此处省略了Dao层调用与异常处理等非核心代码，仅展示正常路径下UserLogicImpl与UserLdapServiceImpl的部分代码。其中UserLogic层负责参数校验，以及通过其私有方法完成UserVO与User的相互转

换。UserService层负责调用UserDao进行认证登录，并对认证结果进行缓存。其中RELESETIME为系统默认的缓存过期时间，可在服务配置文件中进行配置。

```
//   UserLogicImpl
public UserVO login(String userName, String password) {
            //校验参数
           if ( StringUtils.isBlank(userName) || password.isBlank(userName)) {
                 //省略异常处理流程
                 throw new InvaildExecption();
           }
           User user = userLdapService.login( username, password);
           UserVO userVO = this.model2VO(user);
           return userVO;
}

// UserLdapServiceImpl
public User login(String userName, String password) {
          User userFromReids = redisService.get(userName);
          // 查询并更新缓存
          if (userFromReids != null ) {
               //更新缓存过期时间
               redisService.set(user, userFromReids, RELEASETIME);
               return userFromReis;
          } else {
               User resultUser = ldapUserDao.findBy( userName, password );
               //查询结果存至缓存
               redisService.set(user, resultUser, RELEASETIME);
               return resultUser;
          }
}
```

图 4.4: 用户服务核心代码

4.2　文件服务设计与实现

4.2.1　文件服务架构设计

文件服务是本系统中的基础服务，为其他业务提供文件传输相关的业务支持。文件服务架构设计如图 4.5所示。文件服务在服务启动时向consul写入地址信息与端口信息，处理由展示端或其他服务发出的文件上传下载请求。文件服务还在Logic 层提供从文件读取对象和将对象持久化至文件系统的功能。其中持久化至IPFS中的对象，在完成持久化操作后，由系统主动缓存至TiDB中，对象属性包括其在IPFS中的地址。这样的设计可以保证缓存对象真实可信。

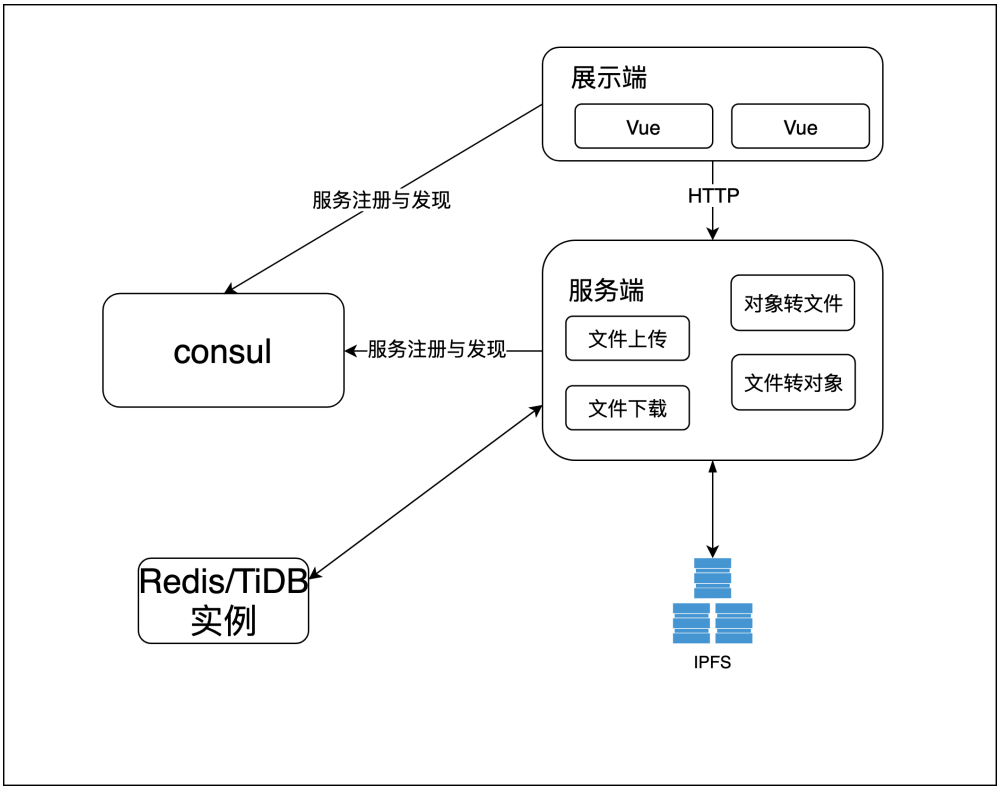

图 4.5: 文件服务架构图

4.2.2　文件服务核心类图

如图 4.6所示是文件服务核心类图设计，FileController是本服务对外提供服务的控制器，主要提供文件上传与文件下载，FileLogic依赖FileService 实现文件上传下载，此外FileLogic 还负责处理字符流和文件类的映射转换处理。FileService 通过调用IPFSService提供的接口实现文件上传与下载。

FileService还实现了格式化文件与Java对象之间的转换，Java 对象在被存储至IPFS后也会被更新至缓存以以便后续访问。目前支持XML类型文件、JSON类型和YAML类型文件与Java对象互换，可通过文件后缀判断其类型。

在FileController层，用户上传的文件以MultipartFile存于内存中，经过转换存为Byte数组类型的字节流传入FileLogic层。从IPFS中读取到的文件也是以字节流的形式读取，经过FileService和FileLogic层传递后，在FileController中转为StreamingResponseBody传至客户端，客户端可通过HTTP响应头判断出文件类型。

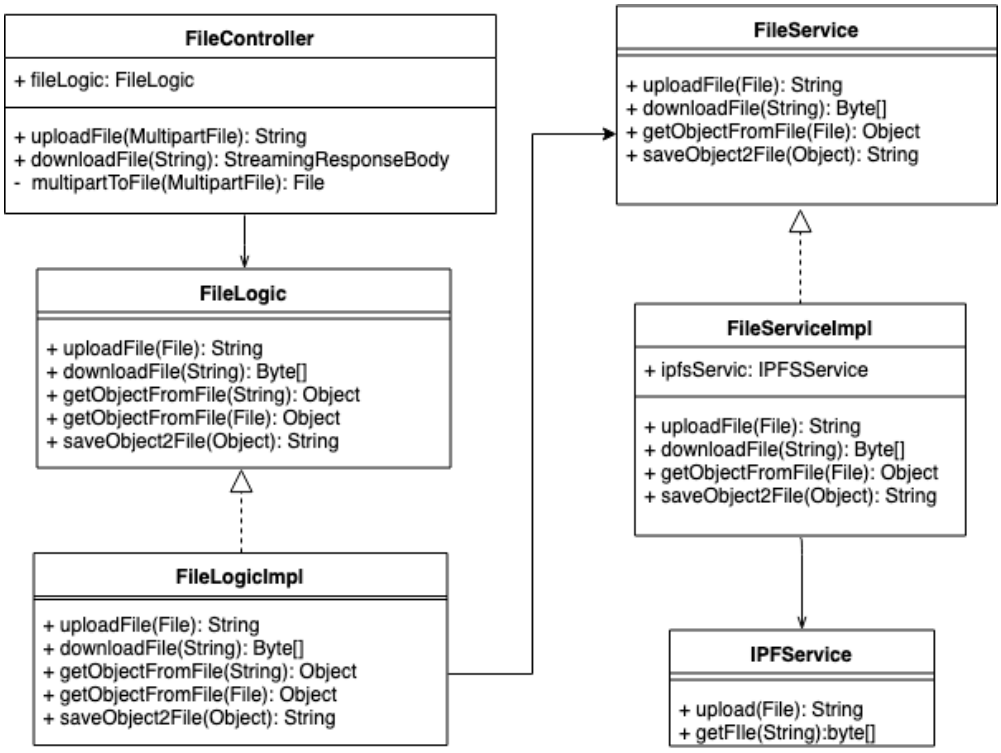

图 4.6: 文件服务核心类图

4.2.3　文件上传与下载顺序图

如图 4.7所示是文件服务提供文件上传与文件下载功能的服务调用顺序。

在接受到文件上传请求时，FileController先通过其私有方法将MultipartFile对象转换成业务逻辑更易处理的File对象，再调用FileLogic中的save方法将文件传至IPFS中，上传文件方法的返回值是该文件的hash值，也是其在IPFS中的相对路径，可以通过该返回值对查询该文件及其上传时间等信息。为保证数据安全性，上传过程中产生的临时文件由FileLogic调用删除方法进行删除，避免因本地缓存产生数据泄露。FileController在接受到下载请求后，会调用FileLogic的getFile方法，再由FileLogic调用FileService获得IPFS中的文件字节流，最终在FileController中将字节流以StreamingResponseBody对象传至用户客户端，由浏览器将文件保存至客户端默认地址，文件名即为该文件在IPFS中的存储地址。

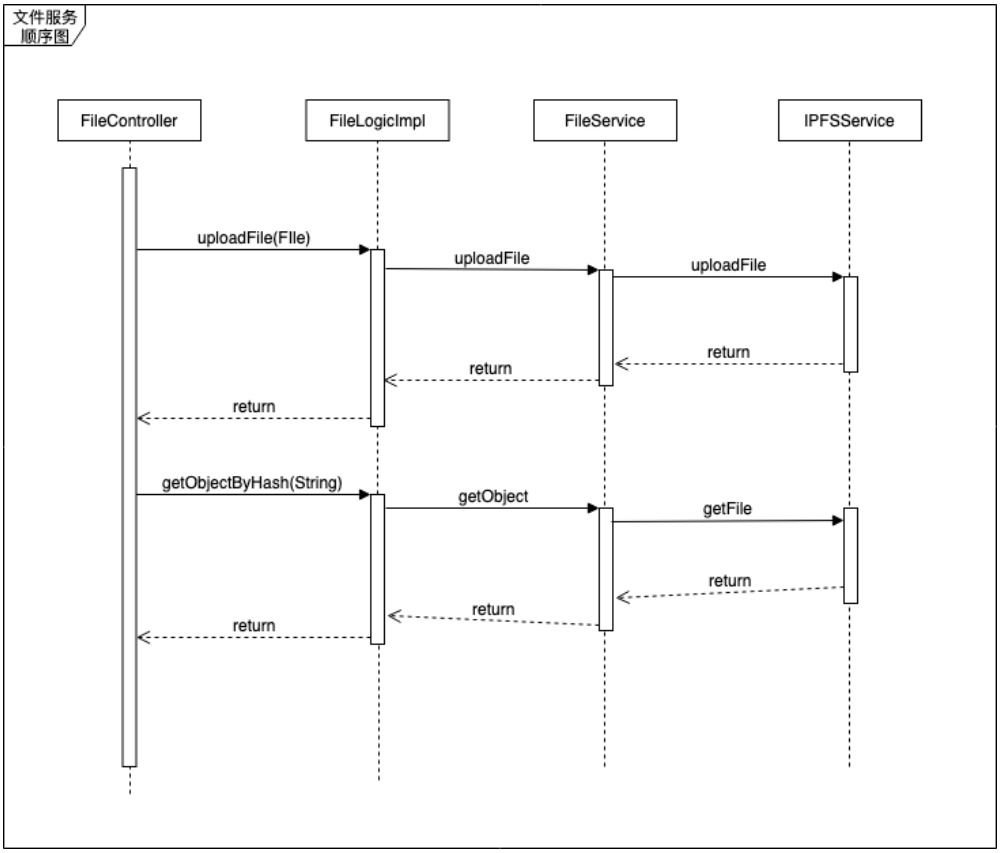

图 4.7: 文件服务顺序图

4.2.4　文件下载与对象转换关键代码

如图 4.8所示为文件服务中关键功能的代码片段，包括文件下载功能，以及讲Java对象转为XML格式文件并存入IPFS的功能。为不影响阅读，此处简单的层级调用，仅展示FileController与FileService中的部分核心代码。

每个上传至系统的文件均可得到一个文件Hash值，也就是该文件存储于IPFS中的地址。用户通过该Hash值可从系统中下载诸如需求文档等文件。FileController通过调用FileLogic获得Hash值对应文件的字节数组，将字节数组转化为输出流后写入Response中，用户客户端可通过Response中设置的ContentType来判断文件类型并保存。

把Java对象存入IPFS首先需要将Java对象转成XML对象，通过IdService生成全局唯一Id作为临时文件名，通过XMLMapper将对象写入XMl文件，再调用传文件至IPFS的接口，最后将刚刚生成的临时文件删除。

```
// FileController
public StreamingResponseBody downloadFile(HttpServletResponse response,
@RequestParam("hashcode")String hashcode) throws FileNotFoundException {
      byte[] fileBytes = fileLogic.getFileByHashcode(hashcode);
      response.setContentType("application/force-download");
      response.addHeader("Content-Disposition", "attachment;fileName=" + hashcode +
".xml");
      InputStream inputStream = new ByteArrayInputStream(fileBytes);
      return outputStream -> {
            int nRead;
            byte[] data = new byte[1024];
            while ((nRead = inputStream.read(data, 0, data.length)) != -1) {
            outputStream.write(data, 0, nRead);
      }
   };
}

// FileServiceImpl
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();
      } catch (IOException e) {
            e.printStackTrace();
      }
      return hashcode;
}
```

图 4.8: 文件服务核心代码

4.3　订单服务设计与实现

4.3.1　订单服务架构设计

订单服务是本系统最核心的服务，提供创建订单、查询订单、更新订单、查询列表等重要功能。订单服务架构设计如图 4.9 所示。订单服务在服务启动时向Consul注册其地址信息与端口信息。

订单数据是本系统中最核心的数据，订单数据会被存储至区块链中，区块链在一定程度上可看成是只有增操作和查操作的分布式数据库，订单的每次变

更都被记录在区块链中，根据订单编号即可获得此订单在任一时刻的状态。这种全量变更的特性，要求每个订单对象尽可能小，需求文档和测试报告等文档内容仅在订单中存储对应hash值。

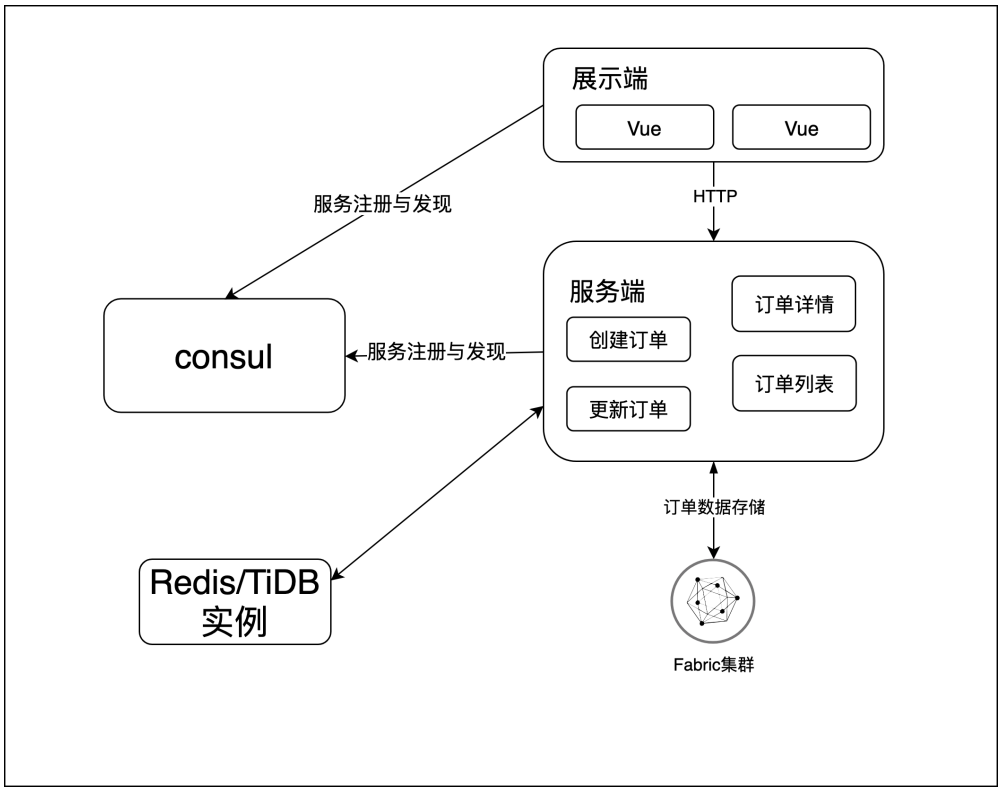

图 4.9: 订单服务架构图

4.3.2　订单服务核心类图

如图4.10所示是订单服务核心类图设计。订单服务是本系统中功能最多、业务最复杂的服务，故涉及到的类较多，为不影响阅读体验，下图仅列出最核心的类与接口。OrdeController负责接受客户端发出的HTTP请求，主要提供订单创建、订单查询、订单属性更新以及订单列表查询等功能。OrderLogic依赖诸多接口，例如OrderService、IdServic、RequirementLogic、ReportLogic、AcceptanceCriteriaLogic 以及UserLogic等。OrderLogic 对OrderService提供的接口进行封装，将订单更新细化成对订单各属性分别更新的多个接口，此细粒度设计可避免错误更新。为减少区块链的数据存储压力，订单对象存储关键属性。如图 4.10右侧Order对象所示，需求文档、验收标准和测试报告等属性均为Hash值，此Hash值为该对象在IPFS上的存储地址。以Order中Requirement属性为例，OrderLogic在对订单数据进行写操作时需要

把RequirementVO等对象转换成代表该对象的Hash 值，在对订单数据进行读操作时，需要利用RequirementLogic 提供的接口由Hash值获得VO对象。

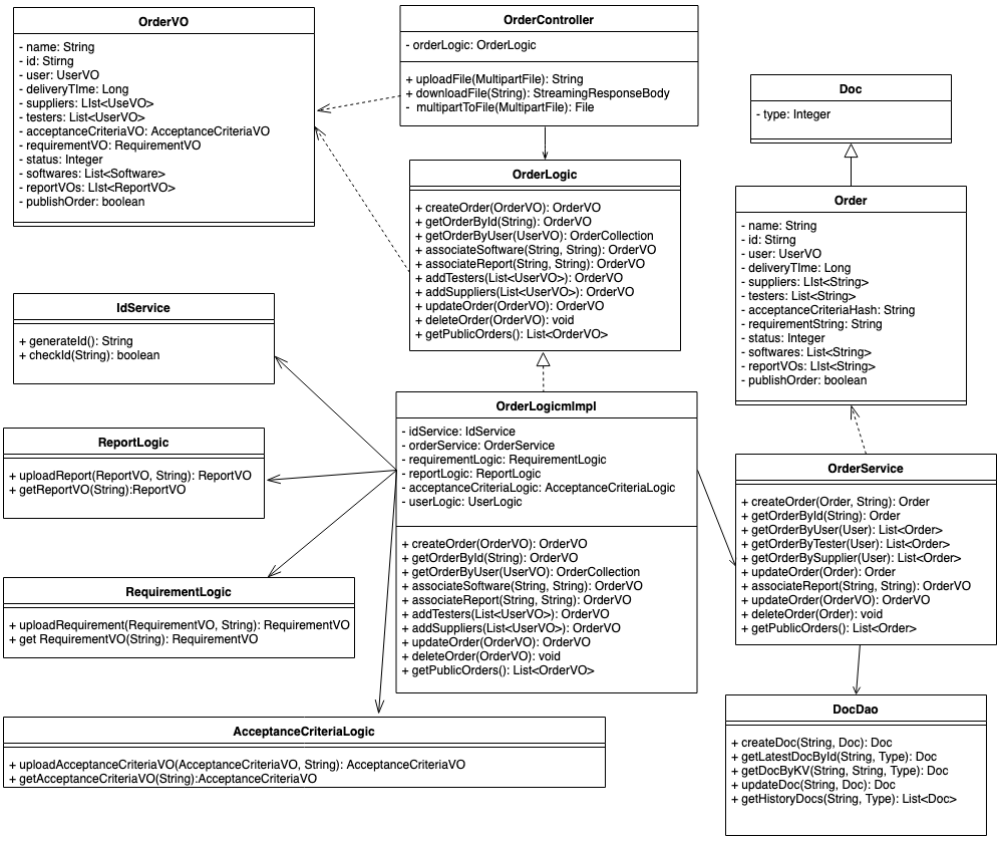

图 4.10: 订单服务核心类图

4.3.3　订单创建顺序图

如图 4.11所示为订单服务中订单创建功能的业务调用顺序图，查询订单以及更新订单等功能均与之类似。OrderController 在接受到订单创建请求后，调用OrderLogic中创建订单接口。OrderLogic对订单参数进行必要检验后，调用IdService生成唯一订单Id，再由RequirementLogic和AcceptanceCriteriaLogic将RequirementVO和AcceptanceCriteriaVO存入IPFS 中并在TiDB作缓存。在OrderVO对象向Order对象转换时，RequiremntVO 和AcceptanceVO 由其Hash值设置入Order中对应属性。OrderService在存入订单对象后会返回其对象，该对象会再由OrderLogic进行转换后经OrderController返回至客户端，OrderVO中的需求等对象均由其Logic实现负责查询。

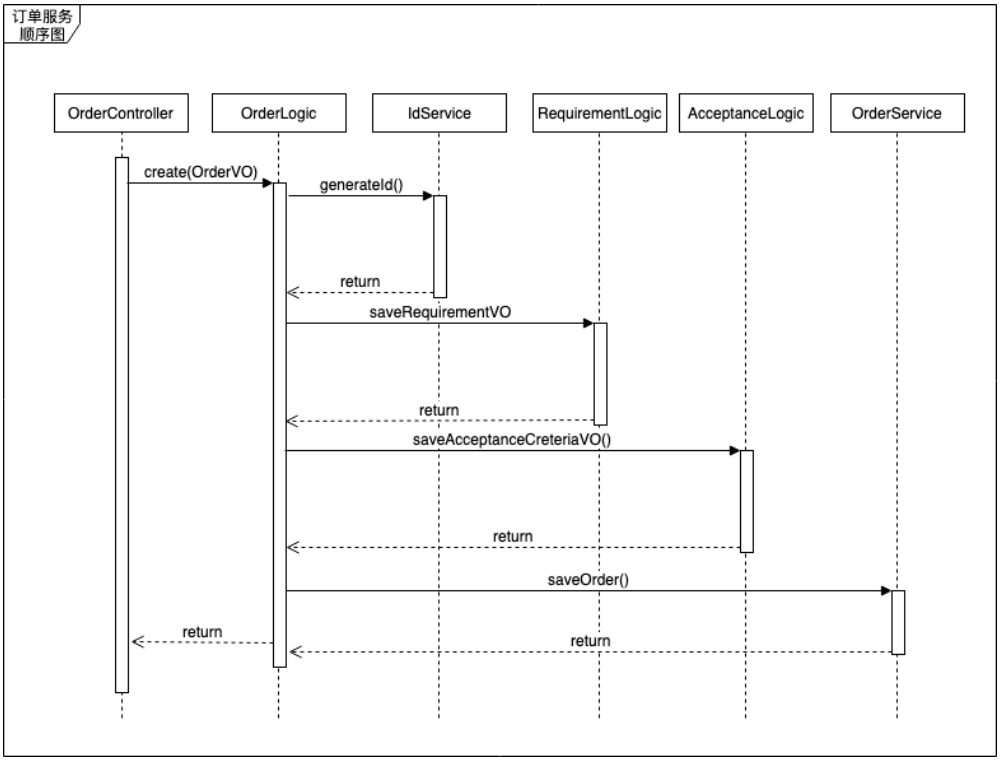

图 4.11: 订单服务顺序图

4.3.4　订单创建关键代码

如图 4.12所示为创建订单功能核心代码，为不影响代码，此处省略参数检查与对象转换代码，仅展示核心代码。该方法展示订单对象创建方法在Logic层中的业务流程。

创建订单是整个软件交付流程的前提，也是系统中最为关键的功能之一。OrderLogic层先会对OrderVO中参数进行有效性检验。通过IdService生成全局唯一Id将其定义为订单Id，此Id即为订单存在区块链中的Key值，可根据此Id查询订单状态以及历史变更记录。在生成订单Id后，OrderLogic会调用需求服务和验收服务对订单中的对应进行进行存储，并得到响应的hash值。这些hash值即为需求对象和验收标准对象存于IPFS中的地址。通过一系列赋值操作，OrderLogic可将OrderVO转换为Order，再调用OrderService对Order对象进行存储。

```
public OrderVO createOrder(OrderVO orderVO) {
    // 省略参数检查
    String orderId = idService.genId();
    orderVO.setCreateTime(System.currentTimeMillis());
    orderVO.setStatus(orderVO.getStatus());
    // 验收标准 vo 要存，需求标准 vo 要存
    RequirementsVO requirementsVO = orderVO.getRequirement();
    String requirementRecordHash =
requirementLogic.uploadRequirement(requirementsVO, orderId);
    AcceptanceCriteriaVO acceptanceCriteriaVO = orderVO.getAcceptanceCriteria();
    String acceptanceCriteriaRecordHash =
acceptanceCriteriaLogic.createAcceptanceCriteriaVO(acceptanceCriteriaVO);
    // 省略 OrderVO 转 Order 代码
    Order order = this.vo2model(orderVO);
    order.setRequirementHash(requirementRecordHash);
    order.setAcceptanceCriteriaHash(acceptanceCriteriaRecordHash);
    Map<String, Order> result = orderService.createOrder(order, orderId);
    // 省略 Order 转 OrderVO 代码
    return this.model2vo(result.get(orderId), orderId);
}
```

图 4.12: 订单服务核心代码

4.4　软件证书服务设计与实现

4.4.1　软件证书服务架构设计

软件证书服务为软件开发方提供软件证书上传、关联软件证书与订单等功能。软件证书服务架构设计如图 4.13所示。软件证书服务在服务启动时向Consul写入其地址信息与端口信息，处理由展示端发出的软件证书管理相关请求，例如上传软件证书与下载软件证书等。

软件证书服务将软件证书文件与软件本身存至IPFS，将软件证书对象存存入区块链中。此设计既可以减轻链上数据存储压力，也可记录软件特征，保证数据可信与可追溯。

软件证书可由本系统配套工具生成，证书中包含上传者信息，上传者的LDAP公钥摘要，软件Hash值等关键信息。这些信息可用于确认上传者身份以及确认软件身份，以便在测试以及验收过程中确定软件是否统一。

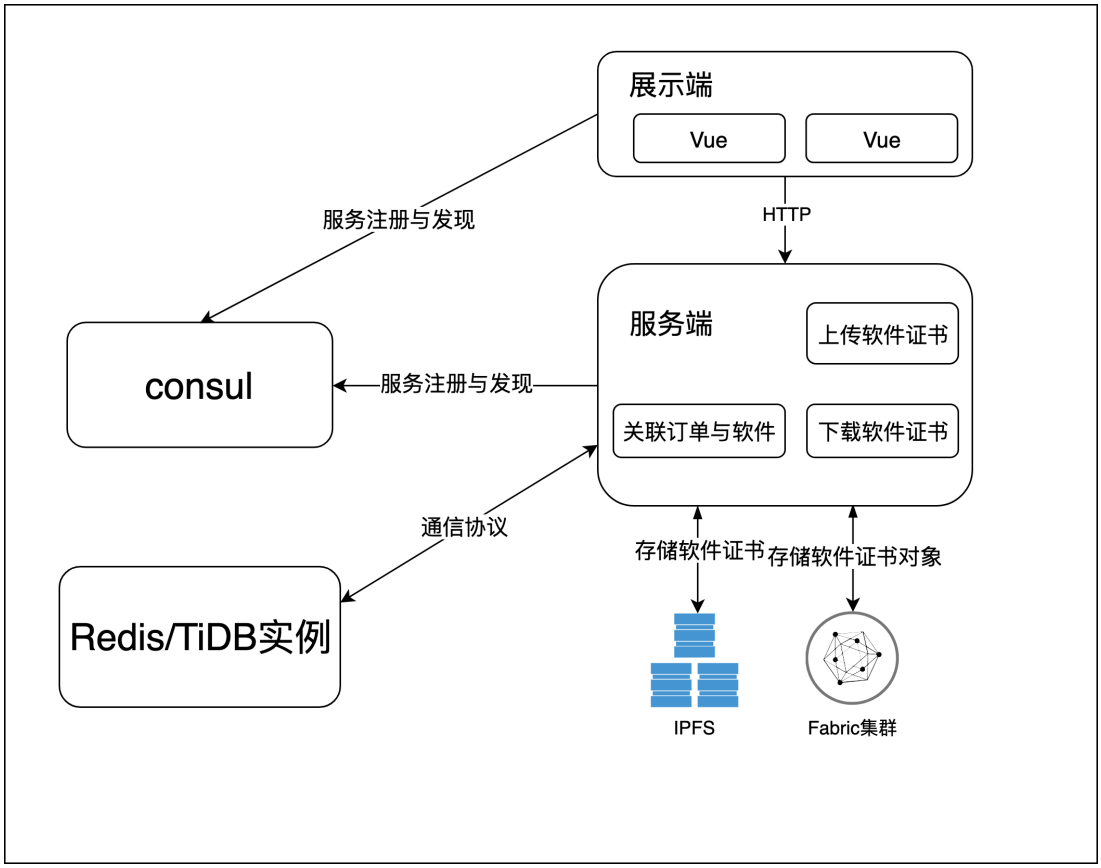

图 4.13: 软件证书服务架构图

4.4.2　软件证书服务核心类图

如图 4.14所示为软件证书服务核心类图。

SoftwareController为提供服务的控制器，主要提供软件证书对象管理功能。SoftwareLogic负责处理核心逻辑，提软件证书对象的增查操作功能。SoftwareLogic 依赖FileLogic中的根据Hash获取Java 对象接口，实现对软件证书的校验功能。与上文提到的OrderService类似，SoftwareService依赖DocService来实现区块链对象的增与查功能。

Software与SoftwareVO对象为同一数据在不用服务中的不同表现形式。Software类与Order类一样均继承自Doc类。SoftwareVO对象中包含软件证书Id，根据此Id可以查询该软件证书的历史版本，从而可以查询到软件的变更版本。

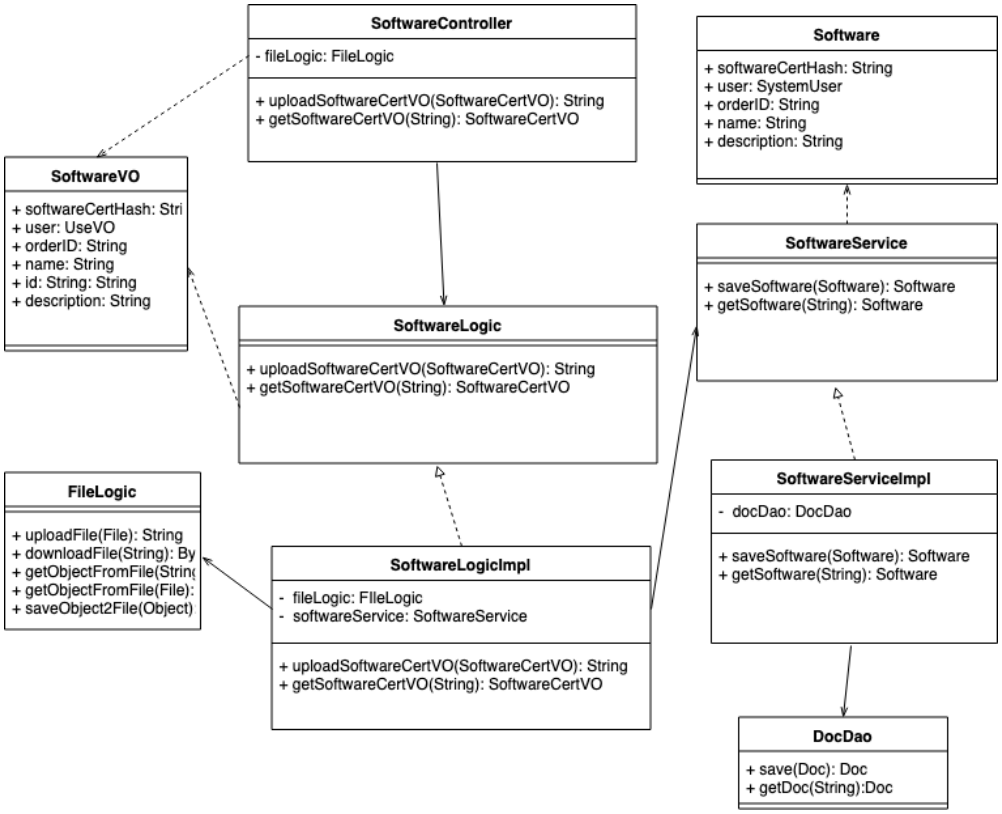

图 4.14: 软件证书服务核心类图

4.4.3　软件证书存储顺序图

如图 4.15所示是软件证书服务提供软件证书对象存储功能的服务调用顺序图。SoftwareController在收到上传软件证书的请求后，调用SoftwareLogic中创建软件证书对象接口。用户上传的SoftwareCertVO包含软件证书Hash值，这代表着软件证书存储在IPFS上的地址，用户先前已经传至IPFS中。通过软件证书Hash值，SoftwareLogic调用FileLogic中的getObject方法可以得到Software对象，再由SoftwareLogic将Software与SoftwareCertVO中的信息进行对比，若符合则通过SoftwareService与DocDao将SoftwareCertVO存入区块链，若两者不相符合则停止流程。

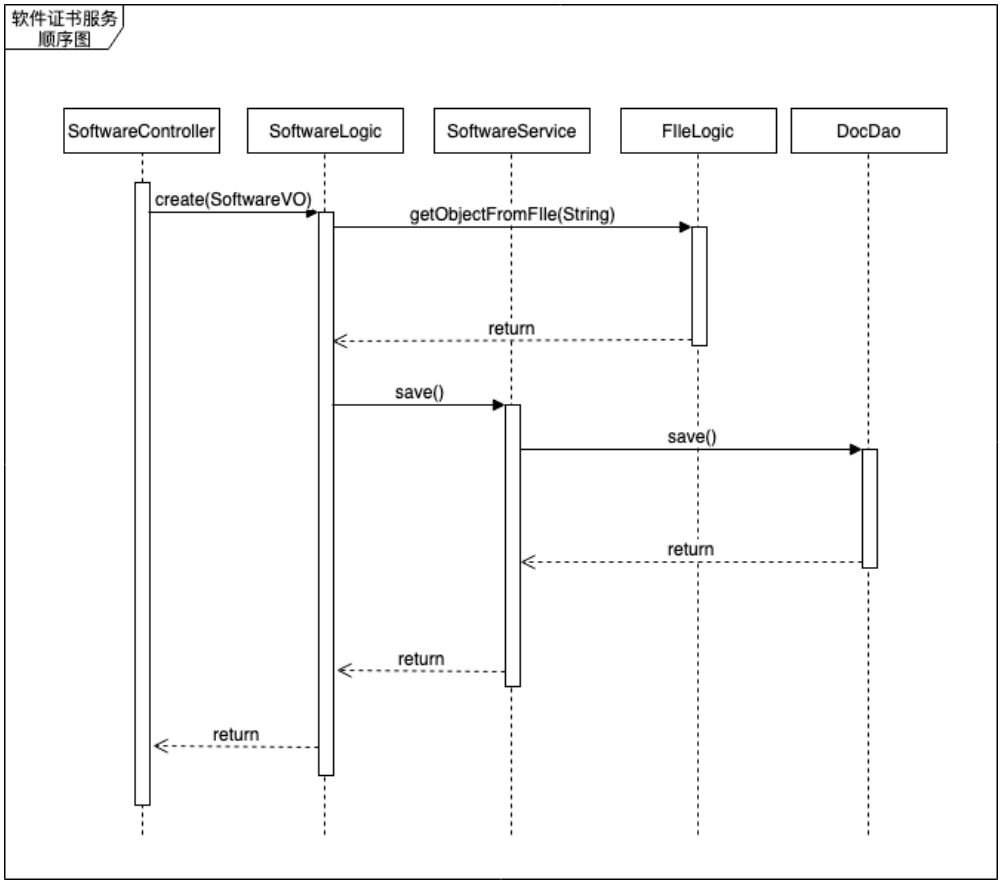

图 4.15: 软件证书服务顺序图

4.4.4　存储软件证书关键代码

如图 4.16所示为软件证书服务存储软件证书功能的核心代码，此段代码摘自SoftwareLogic的实现类SoftwareLogicImpl。SoftwareCertVO会先被校验其参数有效性，若缺少关键字段如软件证书Hash值（即软件证书文件在IPFS中的地址）则直接异常返回。通过FileLogic中的getObject方法，可以得到软件证书文件中的内容，包含上传者信息与其他软件相关信息。然后通过IdService生成全局唯一的Id，这是SoftwareCert在区块链中的Key，以后可以通过该Key查询软件证书对象在区块链中的变更历史。接下来对比软件证书对象与用户上传的SoftwareCertVO，若两者相符合则通过SoftwareService调用DocDao将SoftwareCert存入区块链中，若不相符则异常退出。

```
Public SoftwareCertVO createSoftwareCert(SoftwareCertVO softwareCertVO) {
    ……
    // 参数校验代码已省略
    SoftwareCert softwareCert = this.softwareVO2software(softwareCertVO);
    String softwareCertId = idService.generateId();
    Object softwareCertSavedInIPFS =
fileLogic.getObjectFromFIle(softwarecertVO.getSoftwareHash());
    …
    // 对象比较代码已省略
    Map<String, Software> source = softwareService.createSoftware(softwareCert,
softwareId );
    softwareCertVO.setCreateTime(System.currentTimeMillis());
    softwareCertVO.setUpdateTime(System.currentTimeMillis());
    String softwareId = (String) source.keySet().toArray()[0];
    SoftwareCertVO result = this.software2SoftwareVO(source.get(softwareId),
softwareId);
    return result;}
```

图 4.16: 软件证书服务核心代码

4.5　自动验收服务设计与实现

4.5.1　自动验收服务架构设计

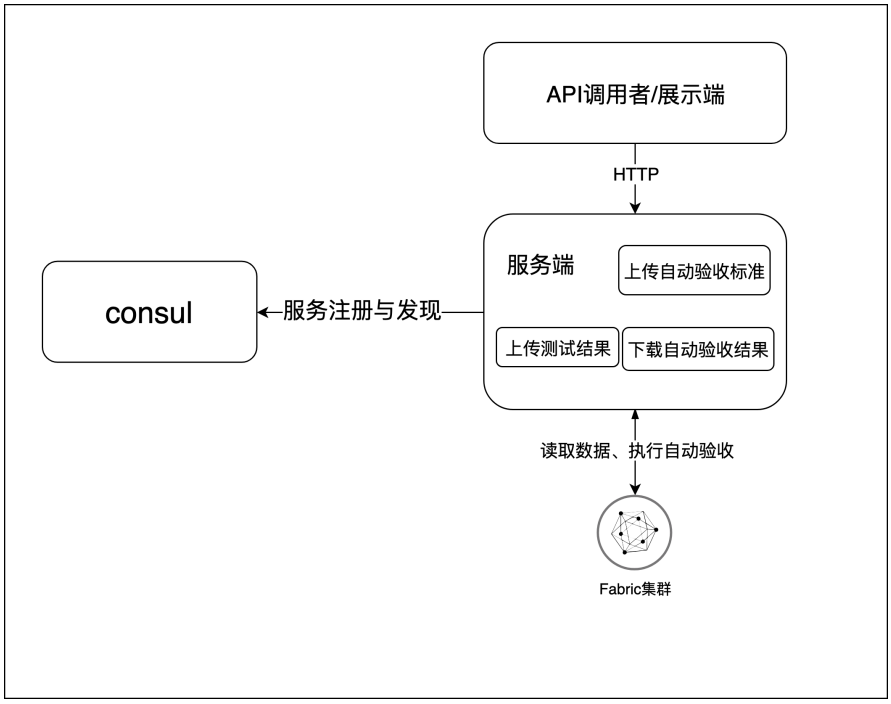

图 4.17: 自动验收服务架构图

自动验收服务负责软件测评结果上传、软件订单自动验收与其结果的查询等功能。自动验收服务架构设计如图4.17所示。自动验收服务在服务启动时向Consul写入其地址信息与端口信息，处理由客户端或展示端的请求。自动验收涉及到测评结果、自动验收标准等数据均存于区块链中，区块链底层会通过运行智能合约来得到自动验收结果。自动运行智能合约保障了结果可信，为本系统增强可信度。

4.5.2　自动验收服务核心类图

如图 4.18是自动验收服务核心类设计，AutoAcceptController是本服务的控制器，主要提供对上传测评结果、获得自动验收结果的功能。AutoAcceptLogic负责校验软件提供商上传的软件测评结果的有效性，并在保存完毕后通知底层区块链服务异步执行自动验收智能合约。

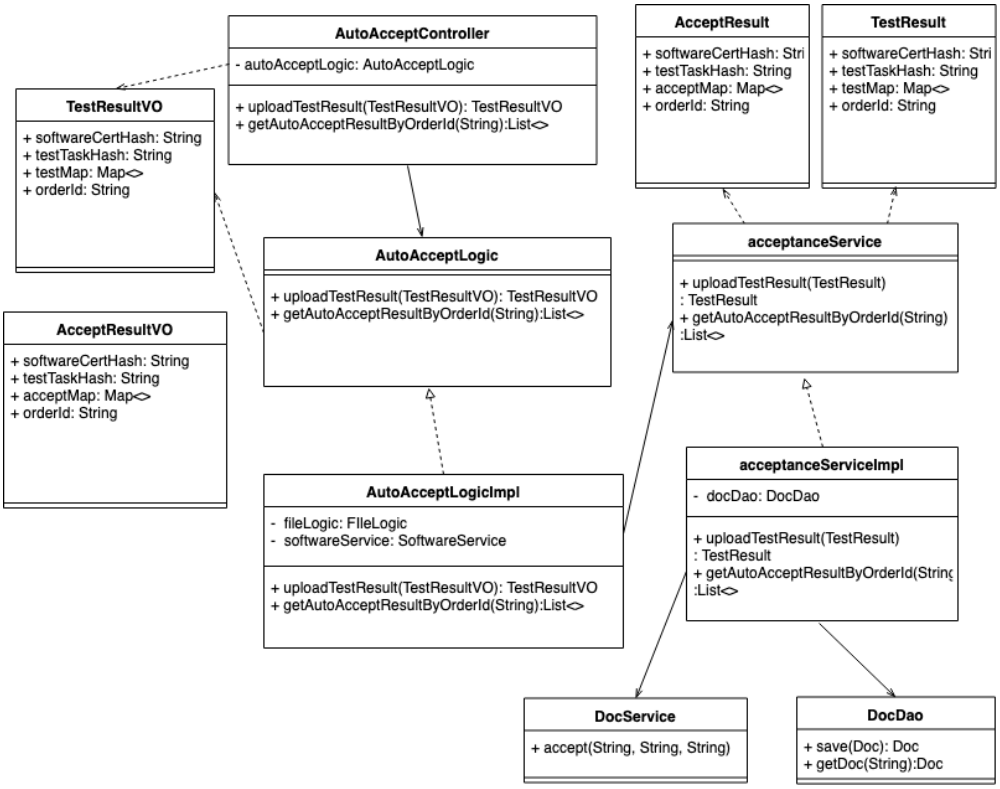

图 4.18: 自动验收服务核心类图

软件测评方上传的TestResultVO 中主要包含测评结果和测试任务的标识，测评结果以键值对的形式返回，其中的键与自动验收标准中的键一致。自动验收标准中以标准名、操作符以及数值三元组形式存储至区块链中。

4.5.3　自动验收顺序图

如图 4.19所示是自动验收服务接收软件测试结果的服务调用顺序图，是本服务中最为重要的流程。在收到软件测评方上传的测试结果后，AutoAcceptController会调用AutoAcceptLogic进行业务处理。AutoAcceptLogic在接受软件测评结果后，首先对数据有效性进行验证，验证通过的数据会通过AutoAcceptService存入区块链中并返回结果Hash。在AutoAcceptService 完成数据存储后，AutoAcceptService会通过异步线程执行自动验收功能，自动验收功能完成后，自动验收结果会被写入区块链。

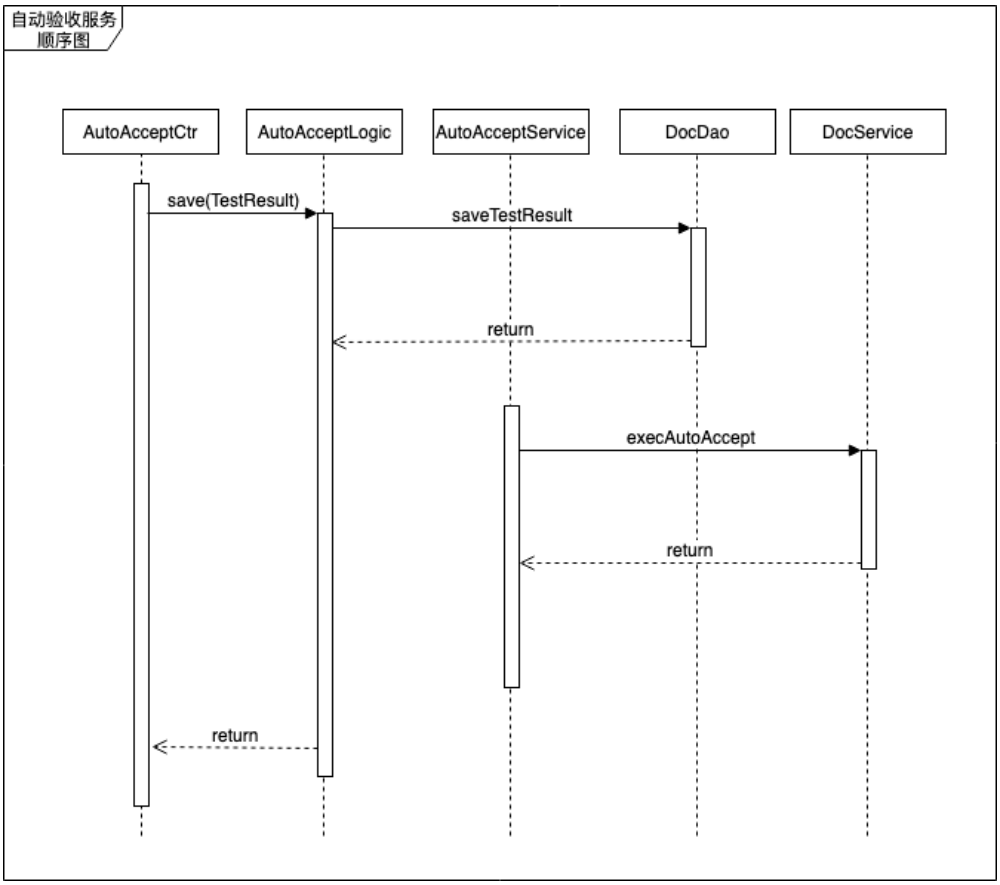

图 4.19: 自动验收服务顺序图

4.5.4　自动验收关键代码

如图 4.20所示为自动验收服务中接受软件测评结果与根据自动验收标准执行自动验收的主要逻辑。由测评机构上传的测试结果先被写入区块链中。之后通过订单Id获得此订单对应的软件验收标准Id。最后通过异步线程执行调用自动验收功能，自动验收功能完成后，自动验收结果会被写入区块链。

```
public String createTestResult(TestResult testResult) {
    String testResultId = idService.genId();
    TestResult result = (TestResult) docDao.createDoc(testResultId, testResult);
    String autoAcceptanceStandardId =
getAutoAcceptanceStandardIdByOrderId(testResult.getOrderId());
    if(autoAcceptanceStandardId == null) {
        return null;
    }
    String autoAcceptResultId = idService.genId();
    doAutoAcceptResult(autoAcceptResultId, autoAcceptanceStandardId, testResultId);
    return testResultId;
}
```

图 4.20: 自动验收服务核心代码

4.6　系统实例展示

本小节提供部分系统运行截图以更直观展示系统运行状态。

图 4.21是用户登录后进入的用户主页部分。用户可通过左侧导航栏快速进入响应功能页面。本页面主要提供用户历史数据统计以及订单列表功能。此页面快速查看登录用户已发起订单次数、提供软件次数以及提供测评服务次数。页面内还有登录用户可见的软件列表与订单列表，通过这些列表，用户可快速查看相关软件详情或订单详情。

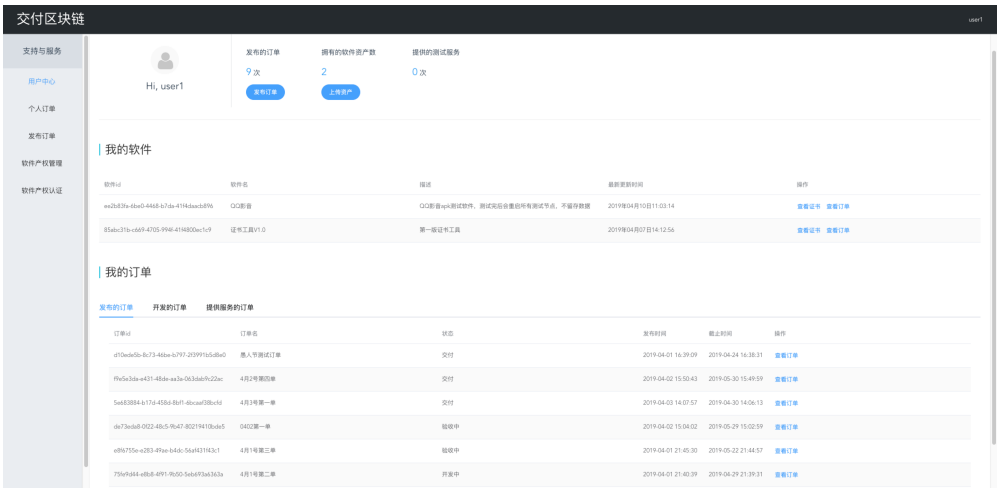

图 4.21: 用户主页运行截图

图 4.22: 订单详情运行截图

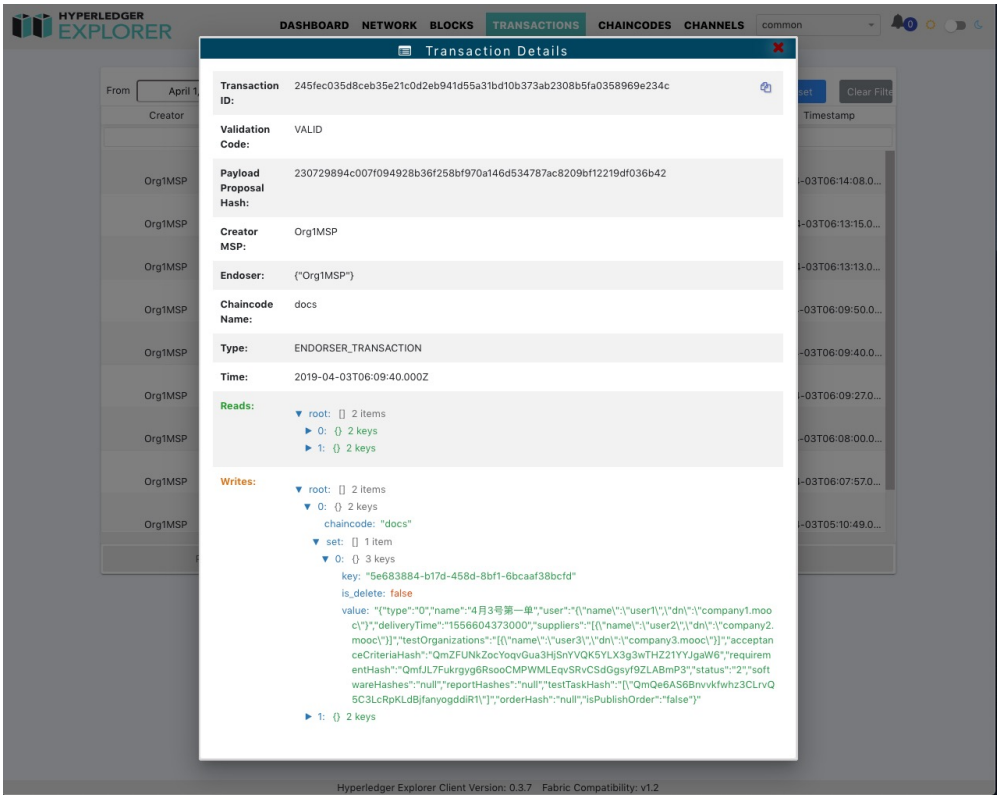

图 4.23: 链上订单数据截图

图 4.22是订单详情运行截图。该订单为已交付订单，通过此订单详情页面，用户可以详细查看订单相关信息，也可对软件需求文件、软件证书和测评报告等内容进行下载。软件需求方为软件验收订单确定相关标准，目前主要支持性能标准与兼容性标准，其中性能测试标准包括最大启动时间、最大启动时间、平均CPU占比、平均内存占比和平均流量耗用。图 4.22还展示当前订单相关的最新交易号，通过点击此链上交易号，可跳转至区块链浏览器中查看该区块的具体信息。

图 4.23即为本订单最新交易的区块信息，区块信息中Key值与订单详情中订单Id相等可证明此交易与订单为对应关系。通过区块链信息与订单展示信息的对比，可以确定该订单信息的真实性，从而保障软件交付过程的可信度。

图 4.24为软件验收结果运行截图。在软件测评方完成测评并提交测评数据至本系统后，本系统会根据订单中的验收标准对该订单进行自动验收。本演示订单中所有标准均已通过，每项数据的最终结果也可通过此页面获得。

图 4.25即为测试结果存储至区块链时的交易信息，区块信息中表示的订单Id与订单详情中订单Id相等可证明此测试结果与订单为对应关系。通过比较图 4.24与图 4.25，可清晰看出链上数据与页面展示信息是一致的，进而可以确定软件测试数据未被篡改，整个自动验收可信有效。

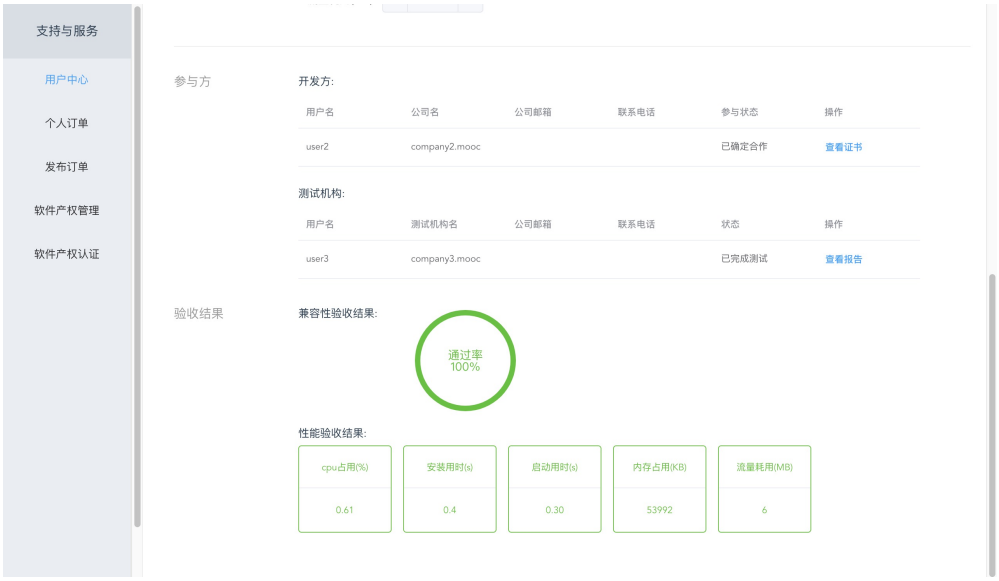

图 4.24: 验收结果运行截图

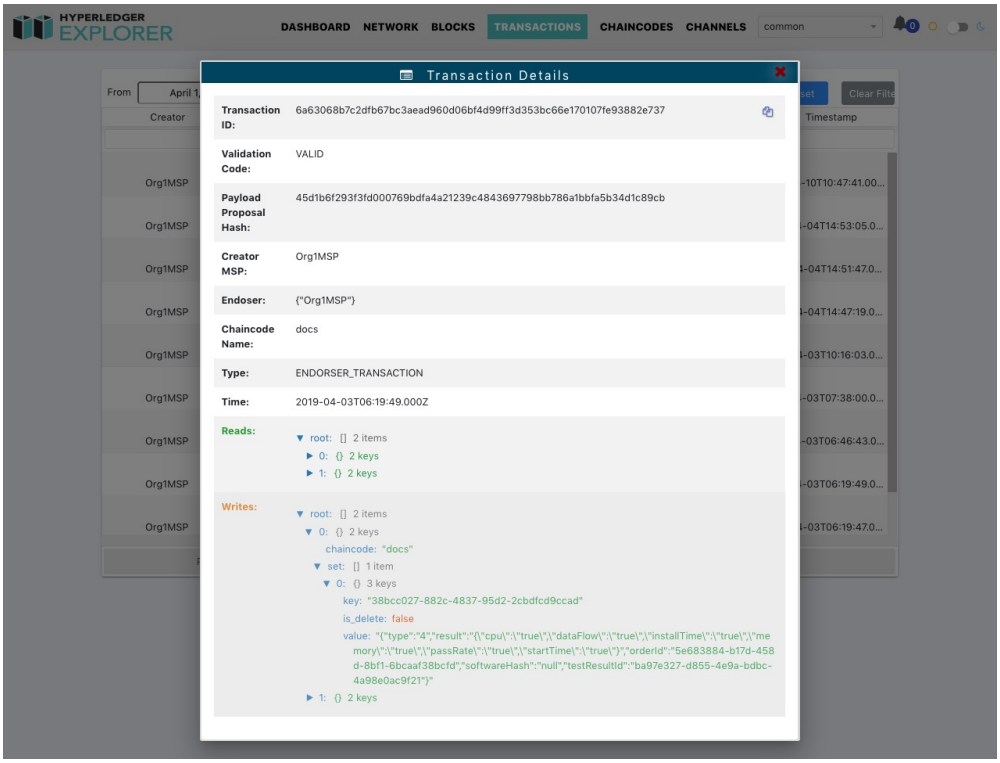

图 4.25: 链上测试结果数据截图

4.7　本章小结

本章主要对本系统中关键服务进行详细设计并介绍实现细节。本章介绍的关键服务有用户服务、文件服务、订单服务、软件证书服务以及自动验收服务，每个服务从服务架构和详细类图两个方面做详细设计，还通过流程图和关键代码来展示关键服务中关键业务逻辑的实现细节。最后通过实际截图对使用流程进行演示，并将链上信息与展示信息进行对比，体现出本系统确能保障软件交付过程中关键数据可信有效。

第五章　自动部署实现与系统测试

5.1　系统自动部署设计与实现

5.1.1　自动部署流程设计

软件部署是软件开发中重要一环，本系统借助Ansible、Docker、Kubernetes和Jenkins等工具，设计并实现了一套适用于快速迭代部署的实践方案。借助Ansible脚本初始化服务器，由Jenkins负责代码更新同步与镜像打包，由DockerHub负责镜像管理，通过Kubernetes和Docker部署系统子服务。

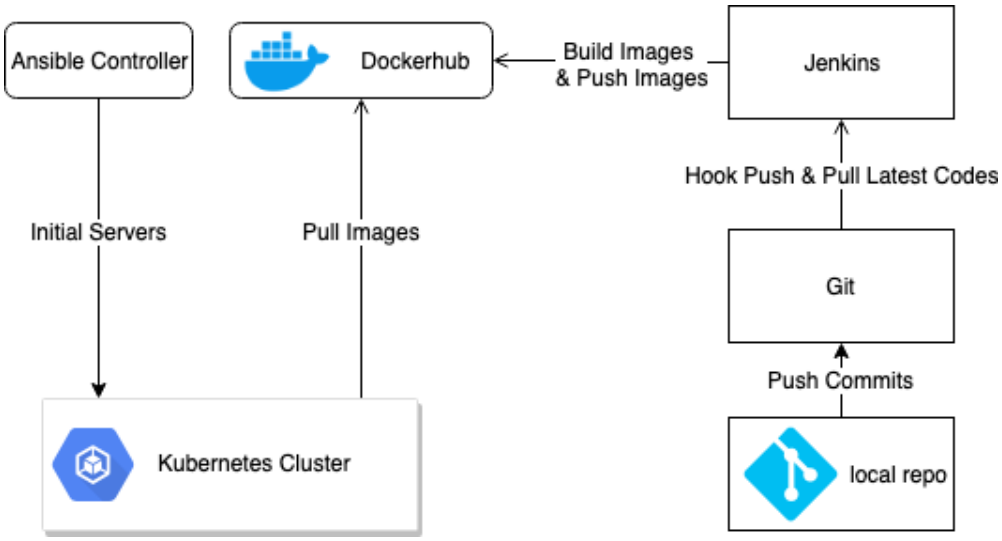

图 5.1: 自动部署流程图

图 5.1是在本系统中从代码到可访问服务的部署流程。其中Kubernetes集群由多台服务器组成，一台服务器为Master节点，其余为一般Node节点。每台服务器负载若干个Pod，业务微服务与系统服务均运行在Pod之中。在系统被启动前，Ansible控制机负责通过Ansible脚本对Kubernetes集群中的机器进行分角色初始化。本系统采用Git作为版本控制工具，在完成某次代码编写后，由本地仓库向远程仓库提交代码。Jenkins服务器在Git指定仓库指定分支上Hook代码更新操作。Jenkins通过预先设置的任务依次进行可执行文件打包、容器镜像打包、上传容器镜像至DockerHub服务器等任务。当Jenkins中的部署任务被触发后，Jenkins使用kublet工具对指定业务服务进行升级，Kubernetes集群会从DockerHub拖取最新容器镜像版本并更新集群中的业务服务。

5.1.2　Ansible初始化解析

Ansible负责本系统中Kubernetes集群中所有节点的初始化，主要包括基础软件包安装和基础网络环境配置等。Kubernetes集群节点可分为Master节点和一般Node节点。

Ansible可按照角色Role对服务器进行初始化，如图 5.2所示，本套Ansible脚本共定义3个角色，分别为server，k8s_master和k8s_node。server角色对所有服务器进行初始化，负责下载Docker、配置私有DockerHub地址、配置防火墙、配置公共容器镜像仓库、基础镜像下载等工作，使服务器成为可供Kubernetes运行的服务。k8s_master角色对Kubernetes中的Master 主节点进行初始化，负责启动Kubernetes集群、配置Flannel网络插件、启动Ingress等Kubernetes系统命名空间服务。k8s_node角色对Kubernetes中的一般Node节点进行初始化，负责加入Kubernetes集群。deploy.yml文件中定义角色也主机的映射关系，每种角色的主要执行脚本存于角色目录下tasks中。

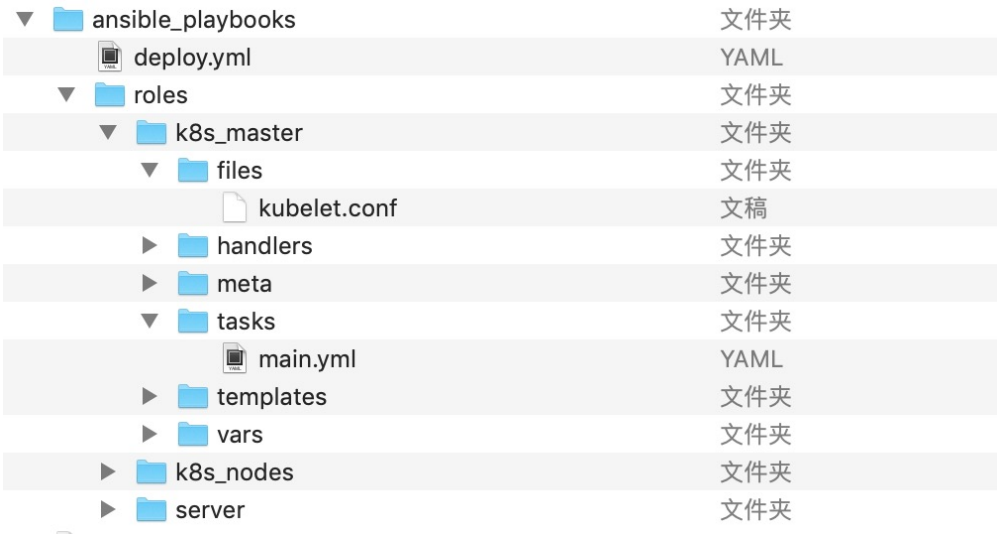

图 5.2: 初始化Ansible脚本结构目录图

5.1.3　Dockerfile模板解析

本系统中各微服务均使用Maven进行依赖管理与项目打包，由生成的可执行包可运行在Docker容器中。本系统中，Dockerfile存于SpringBoot项目根目录，与源代码一同纳入代码版本控制管理，这样设计既便于开发人员参与运维交付，也便于Jenkins在执行镜像打包任务时获取项目代码。

```
# Builder container
FROM local.dockerhub.com/mvn-builder AS builder
COPY . /root/workspace/services
WORKDIR /root/workspace/services
RUN set -ex && mvn clean package

# Runner container
FROM local.dockerhub.com/debian-jdk8
COPY  --from=builder  /root/workspace/services/target/order-service-1.0-SNAPSHOT.jar
/root/dists/order-service.jar
COPY --from=builder /usr/local/bin/docker-entrypoint.sh /usr/local/bin
COPY start-service.sh /usr/local/bin
RUN set -ex \
 && chmod a+x /usr/local/bin/start-service.sh \
 && mkdir -p /root/logs

EXPOSE 8082
ENTRYPOINT ["docker-entrypoint.sh"]
```

图 5.3: 订单服务Dockerfile脚本截图

如图5.3所示为本系统订单服务中用于生成订单服务容器镜像的Dockerfile脚本。出于数据安全性考虑，订单服务的服务打包与最终服务运行由两个容器分别运行，这样在运行容器中只有二进制可执行文件而没有源代码。本系统选用配置有Maven 环境的mvn-builder容器作为专用打包构建容器。为保证运行环境统一，本系统选用debian-jdk8来作为运行基础容器。在构建订单服务镜像过程中，源代码最先在builder镜像中被打包成可执行jar包，接着可执行jar包被拷贝至运行镜像，同时被拷贝至运行镜像的还有服务启动脚本start-service.sh和容器启动后默认执行脚本docker-entrypoint.sh。订单服务使用8082 端口，故Dockerfile中也暴露出容器8082端口以供其他服务访问。

5.1.4　Kubernetes通用Deployment解析

本系统中业务服务均使用Kubernetes中的Deployment形式在集群中进行部署。Deployment是Kubernetes中用来运行无状态应用的一种对象，Deployment通过YAML配置文件描述使用者所期望的集群状态，这种状态包括可用副本个数、端口占用情况和内存占用等。引入Deployment后，业务服务可实现平滑升级与快速水平拓展。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: order-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: order
    spec:
      containers:
      - name: order
        image: order-service:20190401
        ports:
        - containerPort: 8082
        resource:
          request:
            memory: "128Mi"
            cpu: "1"
          limits:
            memory: "256Mi"
            cpu: "2"
        livenessProbe:
          exec:
            command:
            - cat
            - /tmp/health
          initianDelaySeconds:15
          timeoutSeconds:1
```

图 5.4: 订单服务Deployment配置文件截图

图 5.4所示为本系统中订单服务Deployment配置文件摘要。YAML格式为配置文件提供较强可读性。为保证服务高可用性，本配置文件中订单服务被要求生成3个副本，其他非关键服务一般设为2副本。本配置文件通过containers中各属性来指定服务名称、服务镜像版本和运行端口等信息。为提高服务器资源利用率，资源相关的参数也被写入配置。resource中request 属性表明该容器所需资源最低限，换言之运行容器只会被分配到剩余资源满足request属性的节点上。Resource中limits属性该容器所占用资源的上限，容器在运行时因若占用资源超过limits属性中配置会被系统关闭，同时Kubernetes会新拉起一个该服务实例以保证使用者对集群状态的要求。为减少因服务不健康而导致服务中断，探活配置也被写入本YAML文件，当探活指针达到超时设置仍无反应，Kubernetes会关闭该容器并拉一个新实例。

5.2　系统测试

5.2.1　测试目标与测试环境

本系统测试的设计主要从两方面进行考虑，一方面是面对开发者，保证新提交的代码对既有功能没有负面影响；另一方面是面对使用者，保证系统提供的接口可访问且数据正确。

根据上述目标，本小结的系统测试将重点关注单元测试与接口测试。单元测试通过短小的代码片段来测试细粒度函数的可用性与正确性，随着系统代码持续增加的单元测试有助于代码质量的持续改进。接口测试通过特定的交互点来验证子系统间数据的有效性和正确性，根据本系统实际情况，此处接口测试特指为前端与后端交互的接口测试 [40]。

表 5.1: 硬件和系统环境

服务器名称	硬件环境
Jenkins服务器	腾讯云服务器，4处理器8G内存，40G硬盘，Centos7系统
Dockerhub服务器	腾讯云服务器，2处理器4G内存，1T硬盘，Centos7系统
Ansible控制服务器	腾讯云服务器，1处理器2G内存，20G硬盘，Centos7系统
Kubernetes集群主节点	腾讯云服务器，16处理器64G内存，200G硬盘，Centos7系统
Kubernetes集群从节点	腾讯云服务器，16处理器64G内存，200G硬盘，Centos7系统，2台
TiDB服务器	腾讯云服务器，2处理器4G内存，1T硬盘，Centos7系统
Redis服务器	腾讯云服务器，4处理器8G内存，1T硬盘，Centos7系统
IPFS服务器	腾讯云服务器，4处理器8G内存，200G硬盘，Centos7系统，2台
Fabric服务器	腾讯云服务器，4处理器8G内存，200G硬盘，Centos7系统，2台

在5.1小节自动化部署基础上，测试环境的搭建较为方便。如表 5.1所示是部署本系统各个服务与基础设施所需要的硬件资源和操作系统版本。本系统部署于腾讯云服务器，主要使用Centos7操作系统。Ansible控制服务器只需存储脚本与配置文件，故可选用较为经济的服务器。Dockerhub服务器、IPFS服务器和Fabric服务器因其存储需求所以其硬盘存储配置较高，Kubernetes集群中三台服务器因需运行业务服务，故对处理器与内存需求较高。

5.2.2　单元测试

单元测试是软件测试中重要一环，既可以验证代码正确性，也可以保障代码重构过程可靠性，还是回归测试和持续集成的基础。本系统主要使用Junit框架完成单元测试，同时还分别使用Mockito框架与Moco工具来模拟Java服务调用与Http服务调用 [41]。

本系统采用微服务架，每个独立服务均使用分层架构，一般分为Dao层、Service层、Logic层和Controller层。根据单元测试关注最小可测单元的特点，在测试某个具体类时，利用Mockito框架和Moco工具手动模拟其他服务或数据源的调用与返回，这样有利于关注该本身业务逻辑的正确性。

本系统单元测试主要从Dao层、Service层和Logic层分别展开。因各层负责业务特性不同，测试重点又各有不同。Dao层主要与数据库等数据源进行交互，故测试重点主要为连通性与语法正确性。Service层负责细粒度底层业务，故测试重点为代码覆盖与判定覆盖。Logic层负责参数校验与服务调用，故测试重点为异常参数校验。

表 5.2: 订单创建功能单元测试用例表

单元测试编号	单元测试签名
UT-Order-Dao-1	testOrderDaoCreateConnection()
UT-Order-Dao-2	testOrderDaoCreateNormal()
UT-Order-Dao-3	testOrderDaoCreateNull()
UT-Order-Service-1	testOrderServiceCreateNormal()
UT-Order-Service-2	testOrderServiceCreateUnConnected()
UT-Order-Logic-1	testOrderLogicCreateNormal()
UT-Order-Logic-2	testOrderLogicCreateEmptyName()
UT-Order-Logic-3	testOrderLogicCreateEmptyUser()
UT-Order-Logic-4	testOrderLogicCreateErrorTime()
UT-Order-Logic-5	testOrderLogicCreateDuplicateId()
UT-Order-Logic-6	testOrderLogicCreateEmptyRequirementVO()
UT-Order-Logic-7	testOrderLogicCreateErrorAcceptanceCriteria()

如表 5.2为订单创建功能各层单元测试用例设计表，为提高代码可读性，单元测试签名尽可能贴进其描述的正常或异常流程特征。OrderDao相关单元测试被设计来检测数据源连通性与验证保存数据逻辑。OrderService相关单元测试被设计来检测正常流程与验证Dao层不可用情况的处理逻辑。OrderLogic相关单元测试被设计来验证参数检验逻辑，异常情况包括订单无订单名、订单无用户、订单交付日期不合法、订单id重复、订单无需求文档、订单无验收标准和订单验收标准数据无效等。

如图 5.5为订单创建功能单元测试结果图，Dao层接口由于涉及与数据源进行交互耗时较长，其余单元测试服务耗时均在预期之内。本系统在后续开发中，每次构建前均需运行单元测试，若单元测试未能全部通过，则需要分析未通过情况，根据实际情况修改业务代码或单元测试代码。

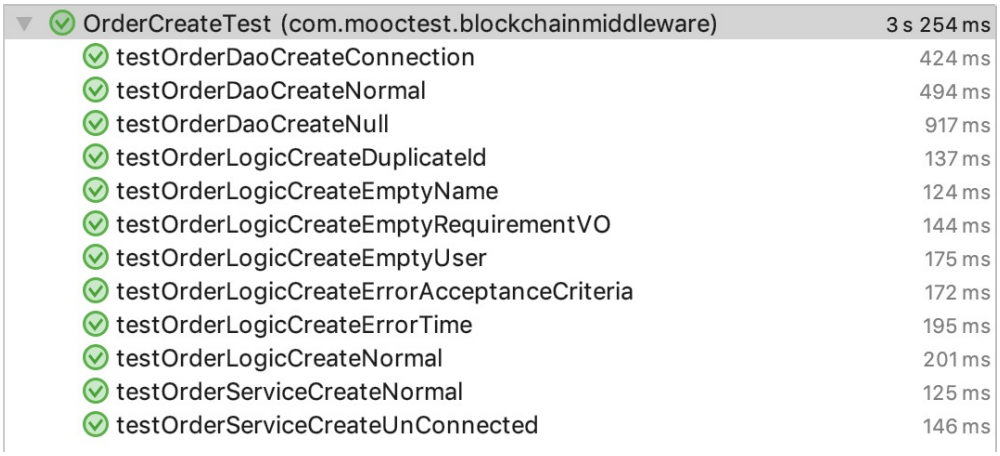

图 5.5: 订单创建功能单元测试结果图

5.2.3　接口测试

接口测试是测试系统组件间接口的一种测试。一般接口测试主要用于检测外部系统与系统之间以及内部各个子系统之间的交互点。本系统采用前后端分离，前后端通过RESTful API进行数据传输，数据对象一致在系统对接中至关重要 [42]。接口测试在本系统中主要负责验证交互数据格式。

本系统主要使用Postman工具发送HTTP请求对后端服务接口的可访问性与返回对象的数据结构正确性进行验证。限于篇幅，本小节仅展示订单创建功能接口测试用例图，以及实现数据验证的Postman测试代码。如表 5.3所示为订单创建接口测试用例图。接口测试是建立在相关单元测试通过基础上，强依赖Logic层、Service层以及Dao层的正确性。

表 5.3: 订单创建接口测试用例表

用例编号	IT-Order-1
测试目标	测试订单创建接口可访问性与返回对象正确性
接口签名	public ResponseResult¡OrderVO¿ createOrder(@RequestBody OrderVO orderVO)
前置条件	表5.2中所列单元测试均通过
输入	发送POST请求，在RequestBody填写正常或异常数据
预期输出	正常输入内容返回200状态码和订单数据，异常输入内容返回500状态码和错误描述

图5.6为订单接口创建接口测试结果。本接口测试共执行10轮共计30个POST请求，从参数正常、参数异常和无权限三个角度对接口进行测试，并利用Postman工具中的脚本对返回结果进行格式校验。本系统对后端返回参数进行包装，通过code 字段表示HTTP状态码。

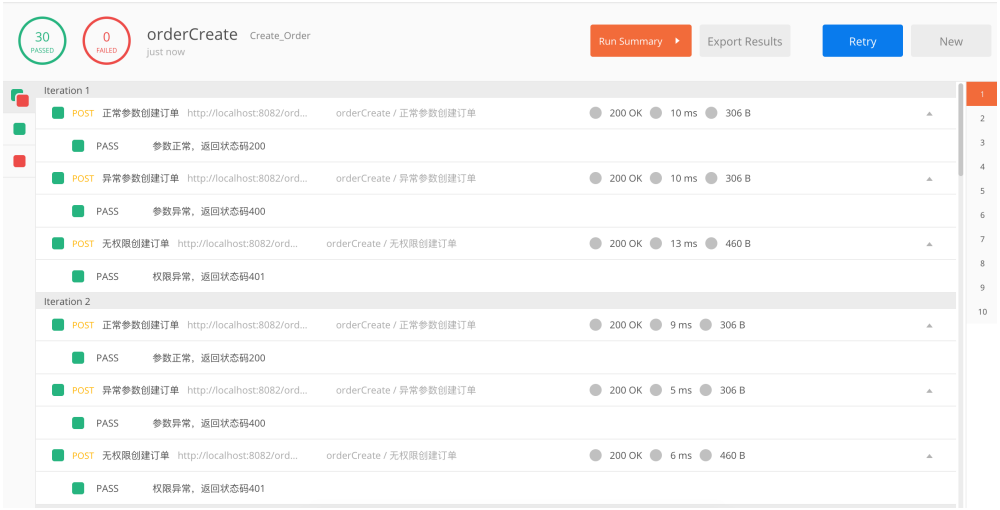

图 5.6: 订单接口创建接口测试结果图

5.2.4　缓存性能分析

为减轻区块链存储压力，本系统引入IPFS存储用户上传文件和对象文件。这些对象文件由Java对象转为XML，通过文件存储的形式实现了持久化。但对于任何编程语言，频繁读写小文件对性能的影响均是无法忽略的。本系统引入TiDB来处理缓存IPFS中的对象文件数据，配合一定的缓存算法，系统的性能有了较大提高，这种技术选型对一般的区块链应用也有一定的参考意义。本小节将通过对比实验，直观得展示该方法对系统性能的提升。

本小节主要使用Wrk这一工具对是否使用缓存这两种情况分别进行压力测试，测出一定时间内的每秒查询率，也就是在压力测试中常说的QPS指标。限于篇幅，本小节仅测试在“读取特定软件验收标准”这一功能点上使用缓存是否带来提升。

Wrk是一款较为流行的HTTP基准测试命令行工具，可通过不同的配置对接口进行评估。本小节使用-t、-c和-d这三个参数来构造不同的测试用例，其中-t代表线程数，-c代表连接数，-t代表压测时间。如图 5.4为性能测试用例表。

表 5.4: 性能测试用例表

用例编号	线程数(个)	连接数(个)	时间(秒)
pt-1	8	400	30
pt-2	16	800	30
pt-3	32	1600	30

图 5.5为上述测试用例执行对比，可以显著看出经过TiDB缓存后，系统QPS有较大幅度提升。在只读的理想状态下，缓存给系统带来近一倍的性能提升。但此处仅针对读操作且不涉及缓存更新与脏读数据，所以在实际情况中这种提升会有一定程度下降。除此以外，还可以观察到在连接数上升后，系统QPS上升趋势减缓，经过分析Wrk执行结果，可发现该现象是由连接数过多从而导致超时查询变多，进而降低了QPS 增速。

表 5.5: 性能对比测试结果

用例编号	无缓存	有缓存
pt-1	452	789
pt-2	802	1465
pt-3	1389	2187

5.3　本章小结

本章主要对系统自动化部署与系统测试进行分析与描述。

由于系统采取了微服务架构，在部署方式上常规单体应用有所差别，借助Ansible脚本初始化服务器，由Jenkins负责代码更新同步与镜像打包，由DockerHub负责镜像管理，通过Kubernetes和Docker部署系统子服务。

为保证系统在开发中与运行中的质量与稳定性，本章还对系统进行多角测试。单元测试主要依赖Junit框架，同时还分别使用Mockito框架与Moco工具来模拟Java服务调用与Http服务调用。接口测试由Postman工具完成，Postman对后端服务接口的可访问性与返回对象的数据结构正确性进行验证。通过单元测试与接口测试，本系统的软件质量与功能质量均得到一定保证。

第六章　总结与展望

6.1　总结与展望

6.1.1　总结

为改善与解决传统软件交付过程中存在的需求变更不透明、三方测评不可信和交付软件与测评实体不一致等缺点，本系统利用区块链技术去中心化、可溯源以及不可篡改等优势，将订单数据、软件证书数据和自动验收相关数据存于区块链中，同时引入IPFS用于存储非关键数据减轻区块链存储压力，使得整个系统更加合理。关键信息上链后，需求变更根据订单历史被追溯，三方测评结果不可篡改，交付过程中涉及到的软件实体信息均在区块链与IPFS中可追踪，这些变化改善了传统软件交付流程，整个系统也因此变得可信与稳固。

本论文还详细介绍了在系统开发与部署过程中涉及到的科学理论、技术选型以及开源框架，这部分对快速了解系统技术栈有较大帮助。接下来。论文第三章综合利用软件需求、软件架构、人机交互等软件工程领域的知识，通过4+1视图、架构图、类图和流程图等形式对系统的功能需求和非功能需求进行系统分析与阐述。在论文第四章，系统各服务的主要逻辑以代码节选或者伪代码形式呈现，同时还给出真实的使用案例来表明系统是如何保障软件交付过程真实可信。最后，由于系统采取了微服务架构，在部署方式与监控方式均与常规单体应用有所差别，故论文第五章对系统的部署方式进行了详细的介绍，并给出具有代表性的脚本与工程文件。除此以外，第五章还给了单元测试和接口测试的方案与测试结果。

6.1.2　展望

本系统在多方面都还有很多不足之处和可拓展的空间。

第一，本系统在功能上还有一些欠缺和不足。目前系统中的订单确认功能还稍显简陋，订单确定在现实场景中需要涉及到招标、投标以及签写合同等过程，本系统目前仅可作为线下过程的一种补充，后续的开发中可考虑将招标投标结果和合同文档等进行电子化。此外本系统目前已对接集成南京慕测公司的移动端兼容性测试服务与移动端性能测试服务，以后需要通过进一步的线下运营接入更多测试服务提供商以丰富系统多元性。

第二，本系统内部接口设计时已尽可能考虑底层技术与底层服务的可替换

性，例如文件系统与缓存系统，不同底层技术之间的组合是否会带来更好地性能或者更高的稳定性，还需要进一步的探索与测评。与此同时，底层技术的选用应可在配置文件中进行配置。

第三，虽然系统在设计之初与编写过程中已尽可能注意性能优化，但随着业务量进一步上升，本系统还是会面临一定的性能压力，此时需要对区块链底层技术的选用进行考量。

最后，本系统还有和众包相结合的可能性。链上的软件证书中可进一步包含具体开发者对本项目的贡献程度，这样链上的所有软件证书信息通过某种方法综合起来可得到具体某位开发者的某种开发能力，这对众包流程中的众包工人选择是一种帮助。

参考文献

[1] M. L. Gordon, T. P. Walsh, Outsourcing technology in government: Owned, controlled, or regulated institutions, Journal of Government Information 24 (4) (1997) 267–283.

[2] 王德双, 戴金龙, 崔启亮, 软件外包全流程解析, 程序员 (8) (2006) 58–61.

[3] 王梅源, 软件外包项目全过程风险管理研究, Ph.D. thesis, 华中科技大学(2006).

[4] A. Kosba, A. Miller, E. Shi, Z. Wen, C. Papamanthou, Hawk: The blockchain model of cryptography and privacy-preserving smart contracts, in: Security and Privacy, 2016.

[5] 朱雪峰, 金芝, 关于软件需求中的不一致性管理, 软件学报16 (7) (2005) 1221–1231.

[6] 徐瑾, 中国软件外包业研究综述与展望, 经济学动态 (11) (2009) 75–78.

[7] 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.

[8] J. Sousa, A. Bessani, M. Vukoli, A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform.

[9] F. Buccafurri, G. Lax, S. Nicolazzo, A. Nocera, Tweetchain: An alternative to blockchain for crowd-based applications, in: International Conference on Web Engineering, 2017.

[10] F. M. Ametrano, Bitcoin, blockchain, and distributed ledger technology, Social Science Electronic Publishing.

[11] N. Zhang, Y. Wang, C. Kang, J. Cheng, H. E. Dawei, H. District, Blockchain technique in the energy internet: Preliminary research framework and typical applications, Proceedings of the Csee.

[12] Block chain based instrument data management system, China Instrumentation.

[13] 薛腾飞, 傅群超, 王枞, 王新宴, 基于区块链的医疗数据共享模型研究, 自动化学报43 (9) (2017) 1555–1562.

[14] 杨东, 潘曌东, 区块链带来金融与法律优化, 中国金融 (8) (2016) 25–26.

[15] C. Decker, R. Wattenhofer, Information propagation in the bitcoin network, in: IEEE Thirteenth International Conference on Peer-to-peer Computing, 2013.

[16] 沈鑫, 裴庆祺, 刘雪峰, 区块链技术综述, 网络与信息安全学报2 (11).

[17] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, Hyperledger fabric: A distributed operating system for permissioned blockchains.

[18] H. Sukhwani, N. Wang, K. S. Trivedi, A. Rindos, Performance modeling of hyperledger fabric (permissioned blockchain network), in: 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), 2018.

[19] 范捷, 易乐天, 舒继武, 拜占庭系统技术研究综述, 软件学报 (6) (2013) 1346–1360.

[20] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, K. L. Tan, Blockbench: A framework for analyzing private blockchains.

[21] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, B. T. An, S. Chen, The blockchain as a software connector, in: Software Architecture, 2016.

[22] M. Kelly, S. Alam, M. L. Nelson, M. C. Weigle, Interplanetary wayback: Peer-to-peer permanence of web archives, in: International Conference on Theory and Practice of Digital Libraries, Springer, 2016, pp. 411–416.

[23] M. Li, J. Weng, A. Yang, W. Lu, R. Deng, Crowdbc: A blockchain-based decentralized framework for crowdsourcing, IEEE Transactions on Parallel and Distributed Systems PP (99) (2018) 1–1.

[24] D. Dias, J. Benet, Distributed web applications with ipfs, tutorial.

[25] J. S. Park, M.-S. Chen, P. S. Yu, An effective hash-based algorithm for mining association rules, Vol. 24, ACM, 1995.

[26] 张波, 冯玉琳, 黄涛, 基于对象视图模型WebView的Web应用框架, 软件学报13 (10) (2002) 1985–1990.

[27] 孙昌爱, 金茂忠, 刘超, 软件体系结构研究综述, 软件学报13 (7) (2002) 1228–1237.

[28] 张广胜, 蒋昌俊, 汤宪飞, 徐岩, 面向服务的企业应用集成系统描述与验证, 软件学报18 (12) (2007) 3015–3030.

[29] 王子勇, 王焘, 张文博, 陈宁江, 左春, 一种基于执行轨迹监测的微服务故障诊断方法, 软件学报28 (6) (2017) 1435–1454.

[30] H. Kang, M. Le, S. Tao, Container and microservice driven design for cloud infrastructure devops, in: 2016 IEEE International Conference on Cloud Engineering (IC2E), 2016, pp. 202–211.

[31] M. Httermann, DevOps for Developers, 2012.

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

[33] 陈显鹭, 王炳燊, 秦妤嘉, 自己动手写Docker, 电子工业出版社, 2017.

[34] 闫健勇, Kubernetes权威指南, 电子工业出版社, 2017.

[35] 陈金窗, 沈灿, 刘政委, Ansible 自动化运维: 技术与最佳实践, 机械工业出版社, 2016.

[36] 龚鹏, 微服务分布式构架开发实战, 人民邮电出版社, 2018.

[37] V. Koutsonikola, A. Vakali, Ldap: framework, practices, and trends, IEEE Internet Computing 8 (5) (2004) 66–72.

[38] P. Kruchten, The 4+1 view model of architectu, IEEE Software 12 (6) (1995) 42–50.

[39] J. C. Anderson, J. Lehnardt, N. Slater, CouchDB: The Definitive Guide Time to Relax, 2010.

[40] 郭勇, 邓波, 衣双辉, 面向服务的网格软件测试环境, 软件学报17 (11) (2006) 2335–2340.

[41] V. Massol, T. Husted, JUnit in Action, 2010.

[42] 侯可佳, 白晓颖, 陆皓, 李树芳, 周立柱, 基于接口语义契约的Web服务测试数据生成, 软件学报24 (9) (2013) 2020–2041.

简历与科研成果

基本情况 陈圣超，男，汉族，1995 年 4 月出生，江苏省南通市人。

教育背景

2017.9～2019.6 南京大学软件学院 硕士

20013.9～2017.6 南京大学软件学院 本科

参与项目

1. 国家重点研发计划课题：基于协同编程现场的智能实时质量提升方法与技术（2018YFB1003901），2018-2021
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），2017-2019

致　　谢

在论文完成之际，我要向所有指导和帮助过我的人致以最诚挚的感谢。

首先感谢我的导师陈振宇教授，陈老师在学习和生活上均给予我了很多帮助和指导。大四和研究生加起来这三年，我在iSE实验室接触了很多新的技术，也完成了从“写代码”到“写可维护的代码”这一转变。在毕业设计阶段，感谢陈老师从选题到开题以及后期的代码实现阶段提供的全程负责指导。正是陈老师的督促与帮助，我才能按时完成毕业设计的全部内容。

其次，我还要感谢iSE实验室的诸多师长。感谢房春荣老师一直以来的关心和照顾，毕业论文编写阶段房老师也给了我诸多的建议和思路。感谢黄勇老师一直以来的技术支持，他在Linux运维、DevOps和设计模式等软件工程的诸多方面都教会了我很多。此外，还要感谢iSE 实验室2017 级的研究生同门们，在找工作和写毕设的过程中，我感受了他们普遍拥有的特质——自信、踏实、靠谱，与大家一起读书工作的两年非常愉快。然后，我要感谢母校南京大学对我的培养。算上本科生涯，我已经在南大软院学习生活了6年，今年我24岁，我人生的四分之一都在这座大学之中接受各位师长的照顾。从仙林的远东大道到鼓楼的费彝民楼，母校的每一寸土地都在我心中烙印；从“hello world”到入职国内大厂，母校也见证了我的成长。我会牢记诚朴雄伟的校训，将之体现在我的工作生活中，不负母校的培养。

最后，感谢我的父母，他们为我提供了优质的教育资源与良好的物质条件，他们在本职工作中的出色表现是我一直以来学习工作中的榜样，他们对家庭的付出是我未来努力想要做到的。在我求职和写毕设过程中遇到困难的时候，他们虽然不能提供直接的帮助，却也用和蔼的言语帮助我稳定心态。感谢他们这么多年来的培养与付出，我会在毕业后努力工作关心家人，成为像他们一样的，对家庭对社会有用的人。

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