南京大學

研 究 生 毕 业 论 文

（申请工程硕士学位）

论 文 题 目 基于联盟链的众测数据溯源系统设计与实现

作 者 姓 名 刘子寒

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

2020 年 5 月 23 日

学　　号 ： **MF1832109**

论文答辩日期 ： **2020 年 5 月 23 日**

指 导 教 师 ：　　**（签字）**

The Design and Implementation of Crowdtest Data Provenance System Based on Consortium Blockchain

By

Zihan Liu

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 2020

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：基于联盟链的众测数据溯源系统设计与实现

工程硕士（软件工程领域）专业 2018 级硕士生姓名：刘子寒

指导教师（姓名、职称）：陈振宇 教授

摘 要

众包测试是众包活动在软件测试领域的应用。众测数据如测试报告在众测平台中经过一系列处理交付给需求方，由于中间过程缺少透明度，用户很难判断其来源与可靠性。数据溯源通过对数据来源与转换过程的追溯，能够确保数据真实性。当前的数据溯源系统多采用中心化架构，恶意用户为了利益能够进行数据篡改，溯源数据本身的真实性无法保障。联盟链是有准入机制的区块链，账本数据由经过验证的节点共同维护，具有去中心化、不可篡改的特点。将联盟链技术应用于众测数据溯源，能够保障溯源数据安全。

为实现可信众测数据溯源，本文搭建众测需求方、众测工人、众测平台方组成的联盟链，设计并实现了基于联盟链的众测数据溯源系统。系统主要提供众测数据采集、众测数据联盟链存储及众测数据追溯功能。溯源系统实时采集众测数据并附加来源信息作为溯源数据，关键数据以哈希值方式保存保护用户数据隐私。为实现溯源数据的可信存储，众测数据联盟链存储过程作为联盟链网络中的交易经过各参与方共识验证，由智能合约自动完成。溯源数据作为交易账本多方维护、难以篡改。众测用户通过追溯数据来源信息验证数据真实性，在对审核结果有异议时追溯数据审核信息确定责任人。技术实现层面，本文采用业界主流框架进行开发。前端采用 Vue 框架，服务层采用 Springboot 框架，联盟链存储层采用 Hyperledger Fabric 框架。为保证系统性能和可扩展性，系统采用 Nginx 实现前端反向代理与负载均衡，采用 Docker 部署系统各个服务，Kubernetes 集群管理容器的调度、扩展和负载均衡。

本文对系统进行了功能测试与性能测试。系统业务层接口在 2 秒内 40 次请求情况下平均响应时间为 500 毫秒，系统联盟链在 50tps 的交易吞吐量下平均延迟为 110 毫秒。测试结果表明系统能够基于联盟链提供可靠的众测数据溯源服务。本系统为众包测试提供众测数据溯源服务，提高了众测参与方之间的信任，为众包测试的发展做出了贡献。

关键词：众包测试，数据溯源，联盟链，智能合约

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 Crowdtest Data Provenance System Based on Consortium Blockchain

SPECIALIZATION: Software Engineering

POSTGRADUATE: Zihan Liu

MENTOR: Professor Zhenyu Chen

Abstract

Crowdsourcing testing is the application of crowdsourcing activities in the field of software testing. Crowdtesting data such as test reports are delivered to the demander through a series of processes in the crowdtesting platform. Due to the lack of transparency in the intermediate process, users are difficult to judge their source and reliability. Data provenance can ensure the authenticity of data by tracing the data source and transformation process. Most current data provenance systems use a centralized architecture. Malicious users can tamper with the data for the benefit, and the authenticity of the traceability data itself cannot be guaranteed. Consortium blockchain is a blockchain with an admission mechanism. The ledger data is jointly maintained by the verified nodes, and has the characteristics of decentralization and tamper-resistance. The application of the Consortium blockchain technology to the traceability of crowdtesting data can ensure the security of data provenance.

In order to achieve reliable traceability of public test data, a Consortium blockchain composed of crowdtesting demanders, crowd workers, and crowdtest platform parties is established in this thesis, and a crowdtest data provenance system based on consortium blockchain is designed and implemented. This system collects crowdtest data in real time and attaches source information as provenance data for consortium blockchain storage. The stored process is verified by each participant as a transaction in the consortium blockchain network and is automatically completed by the smart contract. Traceability data is maintained as a transaction ledger by multiple parties and is difficult to tamper with. The crowdtesting user verifies the authenticity of the data by viewing the data source information. When there is any objection to the audit result, the user

checks the data audit information to determine the responsible person. In terms of technology implementation, this thesis uses the industry's mainstream framework for development. The front-end uses the Vue framework, the service layer uses the Springboot framework, and the consortium blockchain storage layer uses the Hyperledger Fabric framework. To ensure system performance and scalability, each service of the system is deployed using Docker. Kubernetes clusters manage container scheduling, scaling, and load balancing.

This thesis carried out a functional test and a performance test on the system. The test results show that the system can provide reliable source data for crowdsourcing testing based on consortium blockchain. The consortium blockchain in this system can still maintain good availability under the transaction throughput of 50tps. This system provides crowdtesting data traceability services for crowdsourcing testing, improves trust among participants, and contributes to the development of crowdsouring testing.

Keywords: Crowdsourcing Testing, Data Provenance, Consortium Blockchain, Smart Contract

目录

表 目 录

图 目 录

第一章　引言

1.1　项目背景与意义

众包活动通过召集互联网群体共同完成任务，借助群体力量解决计算机单独难处理的问题 [1]。众包测试是一种支持软件测试的众包活动 [2]。相较于传统测试，众包测试拥有丰富的用户群体、真实的测试场景及快速的测试反馈，能够提升测试能力及测试效率。众测数据是众包测试的核心资产。众测数据来源于不同众测参与方，包括需求方提交的测试需求文档、待测软件，工人提交的测试报告及众测平台交付的最终报告。

随着众包测试的快速发展，众测参与者对于众测数据的真实性提出了更高的要求。需求方需要确保平台发布真实的测试任务，众测工人需要确保平台报告审核公平公正，众测审核人员需要确定审核的数据没有被篡改。数据溯源是验证数据真假的有效途径，根据追踪路径重现数据的历史状态和演变过程，实现数据历史档案的追溯 [3]。通过对众测数据溯源，众测参与方可以查看数据历史状态来确定数据真实性。然而，当前众测数据溯源存在一定问题。一方面，溯源系统大多依赖于传统数据库，单方维护情况下容易被恶意篡改。另一方面，数据溯源工作一般由中心化机构进行，溯源流程不透明，公信力不足。数据溯源的问题导致众测用户无法信任追溯的数据，从而无法验证数据真实性。

区块链是存储按照时间顺序排序的交易记录分布式账本，具有去中心化、可追溯、防篡改的特性 [4]。区块账本数据多个节点维护，只能追加无法篡改，数据安全性得到保障。联盟链是区块链的联盟形式，区块由联盟中各个节点维护。联盟链账本数据只有经过授权的节点可以访问，保护账本数据隐私。在数据溯源中应用区块链技术，采用区块链作为数据溯源存储层，将数据溯源流程通过公开透明的智能合约实现，能够防止数据溯源被恶意篡改。目前，基于区块链的数据溯源在物联网、供应链等领域有着广泛应用 [5][6]。

本文基于区块链技术，协同需求方、众测工人方、众测平台方作为区块链节点搭建联盟链，通过智能合约将众包测试中的数据附加数据来源信息存于联盟链上，追溯数据时从账本数据获取数据来源信息，实现众测数据溯源系统。系统通过联盟链账本存储数据溯源信息，数据多方公正，无法篡改。需求方可以通过账本数据确保平台方发布正确软件版本。众测工人可以查询已提交报告在众测活动中的流转过程，实现对众测平台的信任。平台方可以追溯众测流程各个阶段数据，实现对众测数据的来源管理。通过研究，本文期望给出基于联盟链的众

测数据溯源方法。在理论上，进一步深化区块链技术在数据资产保护、可信数据溯源方面研究；在应用上，将所提出的方法应用到众测平台上，为众包测试的发展做出贡献。

1.2　国内外研究现状

本文重点研究利用区块链技术实现众测数据的可信溯源，下面将分别从区块链技术、众包测试技术以及数据溯源技术三个方面讨论国内外的研究现状。

1.2.1　区块链技术

2008 年中本聪发布比特币白皮书，标识着区块链的诞生 [7]。2009 年中本聪创建比特币区块链的创始区块，并实现了比特币的交易。区块链去中心化的共识机制、可编程的智能合约以及多方维护的公共账本，为传统交易模式受限于“基于信任的模式”[8] 提供了新的解决方案。区块链根据节点权限开放程度分为公有链、联盟链及私有链：公共区块链中记录对所有节点可见，所有节点都可以参与共识过程，效率较低，目前应用主要为以太坊 (Ethereum)[9]；联盟链共识过程只有预选节点参与，效率较高，Linux 基金会维护的开源区块链项目 Hyperledger Fabric[10] 是联盟链的典型实现；私有链仅允许来自一个特定组织的节点加入共识过程，通常为组织内部使用。

区块链技术因其去中心化、不可篡改、可追溯的特点得到学者的广泛关注研究，并将其应用于数据溯源、居民医疗、数字资产等领域。在云系统数据溯源领域，Liang 等人提出了 ProvChain，对云数据用户的操作进行监控，使用区块链技术将不可更改的时间戳制作记录，并为每个数据记录生成区块链收据以进行验证，实现云系统中的数据溯源 [11]。在供应链数据溯源领域，Montecchi 等人将产品生产流程中的数据存入区块链中，帮助顾客查看真实完整的产品来源信息 [12]。在区块链应用程序溯源领域，Ruan 等人设计了 LineageChain，通过在智能合约执行期间采集数据来源信息并存储于 Merkle 树中，实现区块链应用程序的数据追溯 [13]。在居民医疗领域，Han 等人提出了一种健康信息存储系统区块链架构，该架构采用联盟链搭配私有链的混合区块链方式，改善数据验证的延迟问题，并具有安全、可信赖、防篡改的特性 [14]。Yue 等人提出了一种基于区块链的医疗保健数据网关架构，并设计统一数据模式使患者能够轻松安全地拥有、控制和共享自己的数据 [15]。在数字资产领域，Burstall 等人提出数字知识资产领域的区块链可以用于多种目的，能够很容易地渗透到许多其他数字知识资产行业 [16]。

1.2.2　众包测试技术

众包测试是软件测试的一个新兴趋势，具有众包和云平台的优势、有效性和效率。相较于传统软件测试活动，众包测试基于大量众测工人的多平台测试，能够带来更真实、快速的测试结果。关于众包测试的研究集中于完善众包测试流程、提高测试质量, 包括众测任务设计、众测报告审查及众测激励。

在众多测试工人参与测试的情况下，将复杂的测试任务分解和设计是提高测试效果的重要手段。Tung 等人将众包测试工人协作测试问题定义为测试任务分配的 NP 完全问题，并将其表达为整数线性规划（ILP）问题 [17]。他们提出了基于众包的协作测试方法。该方法包括两个阶段：训练阶段和测试阶段。训练阶段将原始问题转换为 ILP 问题，测试阶段使用启发式策略解决 ILP 问题。在此基础上，Guo 等人设计了一种由任务划分算法和贪婪任务分配算法组成的实时协作测试方法，该方法从测试用例中动态选择任务组，并将测试用例或任务的难度与测试人员的能力相匹配，找到合适的测试人员 [18]。

在测试报告的审查和处理领域，学者也进行了大量的研究。面对大量的测试报告，审核人员可以通过多样性策略和风险策略提高检查效率 [19]。针对移动应用测试报告包含的屏幕截图和较短的描述性文本，Feng 等人提出一种基于多目标优化的优先化技术，以协助检查众包测试报告 [20]。他们采用空间金字塔（SPM）技术来测量屏幕截图的相似性，并应用自然语言处理技术来测量测试报告文本之间的距离。分类是处理测试报告的有效方法，Wang 等人提出一种基于聚类的分类方法，该方法将相似的报告聚类在一起，然后使用集成方法基于大多数相似的聚类建立分类器 [21]。此外，他们还提出了基于局部的主动学习方法，能够从众包测试报告中对真实故障进行分类 [22]。

众测奖励机制反映了需求方或众测审核者将如何评价众测工人的任务完成情况 [23]。不同众测平台在众测流程中的参与度不同，奖励机制也不同。MoocTest、Tencent Test、Aliyun 等平台在众测流程中参与度较弱，主要根据众测工人的任务量进行考核，完成某一份任务的奖励固定；Applause、uTest、Testin 等平台参与众测任务的设计，对测试任务的内容和形式有要求，众测流程参与度较强，有明确的众测奖励机制，按照缺陷和任务的等级进行奖励。

本文研究的众测数据溯源，主要基于 MoocTest 平台的众测流程，对于需求方的待测软件和需求文档、众测工人提交的测试报告、审核人员的审核数据及报酬分配数据进行溯源。MoocTest 平台为众测工人定义好测试报告上传格式，通过审核人员审查测试报告评分并整合最终报告，奖励机制主要通过审核人员的审查情况来判定报酬分配。

1.2.3　数据溯源技术

数据溯源由英文 “data provenance” 翻译而来，不同领域定义各异。Buneman 等人在数据库系统中将数据溯源定义为对数据来源及其到达数据库过程的描述 [24]。Lanter 将地理信息系统中的数据溯源定义为描述用于衍生数据的材料和演变过程的信息 [25]。数据溯源不仅可以与数据产品相关联，而且还可以与创建数据的过程相关联。Greenwood 等人扩展了 Lanter 提出的定义，将数据溯源看作记录实验工作流程、注释信息和实验演变过程的元数据 [26]。本文的溯源是指对众测流程中数据转换流程和来源的真实性验证。

数据溯源模型定义了溯源信息的格式及存储方式 [27]。开放的数据溯源模型 [28] 假定数据的来源可以由带注释的因果关系图表示，该图是有向无环图，并带有捕获与执行相关的更多信息的注释。Bowers 提出了 Time-Value Centric(TVC) 模型，用来处理医疗事件流的溯源信息 [29]。TVC 模型中输出数据流中的每个条目都链接到输入数据样本的某些特定时间窗口，这些时间窗口有助于生成特定的输出条目，其时间依赖性可能随着数据值的变化而变化。

数据溯源追踪方法主要有标注法 [30] 和反向查询法 [31]。标注法将原始数据的注释信息标注到数据上，注释信息通常为数据来源、数据产生时间、关联数据信息等，数据溯源通过查看目标数据相关联的注释信息实现。标注法实现简单，但在大型复杂系统中，标注信息很有可能比原始数据多，存储压力较大，效率比较低。Woodruff 等人针对这个问题提出了一种支持细粒度数据溯源的方法：逆置函数反向查询法 [32]。他们的方法不是依靠元数据，而是构造逆置函数和验证函数，通过数据转换过程的反向查询实现数据溯源。反向查询法不需要存储注释信息，存储开销小，但是逆置函数和验证函数构造复杂，实现难度较大。

数据溯源应用广泛，Goble 将数据溯源应用分为数据质量、审计追踪、数据派生、数据产权和数据解释五个方面 [33]。数据质量：数据溯源可用于基于源数据和转换来估计数据质量和数据可靠性 [34]；审计追踪：数据溯源可以审核数据及其生成过程 [35]，发现数据生成中的错误 [36]; 数据派生：数据溯源信息包括用于派生特定数据集的步骤，可以将其视为创建数据的方法 [37]; 数据产权：数据溯源可以帮助确定用于生成特定数据的源数据的所有权, 用户可以查看源数据的创建者，并验证其产权 [38]。数据解释：数据溯源随数据出处一起提供的注释可以帮助在上下文中解释数据，特别是对于生成后很久还在使用的存档数据。

数据溯源监控数据在工作流程中的转换情况，并将数据附带注释信息单独存储用于溯源。如何在保证数据溯源信息详细的情况下不侵犯数据隐私，同时不给系统带来数据安全风险，是数据溯源必须考虑的问题。

1.3　本文主要研究工作

结合以上研究现状，本文针对众测数据进行数据溯源，并采用区块链技术保证数据溯源的真实可靠，实现基于联盟链的众测数据溯源系统。本系统搭建需求方、众测工人方、众测平台方共同组成的联盟链，溯源数据作为分布式账本数据保存在联盟链上，数据上链过程由智能合约完成，并通过多方共识验证，防止篡改。众测用户可以查看受信任的数据来源验证数据真实性。

系统提供众测数据采集功能。众测活动如提交需求、上传报告、审核报告等进行时会产生需求文档、测试报告、审核结果等众测数据，溯源系统将这些数据实时采集，考虑到众测数据格式不一致，如待测软件 APK、测试报告中的截图等，众测数据统一转换为哈希值，系统将数据来源信息、时间戳、数据处理信息结合原始数据哈希构成溯源数据模型，方便后续数据追溯。

系统提供众测数据联盟链存储功能。为保证溯源数据的安全性，溯源系统将溯源数据存储在联盟链上，数据上链过程作为联盟链网络中的交易由智能合约完成，各个节点参与交易的共识验证，交易透明可追溯。溯源数据作为联盟链账本数据由多方共同维护，同时溯源模型中只保存原始数据哈希值，可以保护数据隐私，防止信息泄漏。

系统为众测用户提供众测数据溯源验证功能。需求方可以查看需求数据和最终报告的来源信息并进行验证；众测工人可以查看测试报告及报告审核来源信息并进行验证；平台方可以查看系统追溯的各阶段众测数据来源及审核信息。数据来源包含数据提交者及数据提交时间，审核信息包括数据审核者及审核结果。系统比对链上数据与众测平台数据哈希确定数据是否被篡改。各参与方通过数据溯源验证数据真实性，并能追溯数据在众测流程中的流转情况，出现纠纷时及时确责。

为保证系统可用性，本文对系统进行了功能测试，并对联盟链进行性能测试，确保在高并发情况下溯源系统能提供稳定的数据溯源服务。

1.4　本文的组织结构

本文详细介绍了基于联盟链的众测数据溯源系统的设计与实现，其组织结构如下：

第一章，引言部分。对项目的背景与意义进行阐述，分析国内外在区块链、众包测试、数据溯源方面的研究现状，并介绍主要研究工作。

第二章，技术综述。介绍与本系统相关的理论与技术基础。研究区块链的特性、架构，介绍众测流程及数据，对比数据溯源技术，选择项目技术方案。

第三章，溯源系统的需求与设计。对系统进行整体概述，并详细分析系统的需求，通过四个系统视图展示系统架构，最后介绍持久化模型设计。

第四章，溯源系统的详细设计与实现。详细介绍了需求数据服务、测试报告服务、最终报告服务、众测任务服务、联盟链服务的实现，并给出联盟链配置与部署方案。

第五章，溯源系统的测试与案例分析。对系统进行功能测试与性能测试，并就众测参与方需求溯源案例进行分析。

第六章，总结与展望。总结论文完成的工作，对于众测数据溯源系统的不足进行分析，并对项目未来的发展做展望。

第二章　技术综述

本系统针对众包测试流程中的数据，通过数据溯源技术追溯众测数据的来源和转换流程，并通过区块链技术解决数据溯源过程中的信任及数据真实性问题。下面分别介绍相关的区块链技术、众包测试技术以及数据溯源技术。

2.1　区块链技术

区块链技术起源于 2008 年由化名为“中本聪”(Satoshi nakamoto) 的学者在密码学邮件组发表的奠基性论文《比特币: 一种点对点电子现金系统》[7]。区块链是一种以分布式方式实现的、去中心化、防篡改的数字账本 [39]。区块链网络中节点可以在共享分类账中记录交易，任何交易在发布后无法更改。区块链具有去中心化、持久性、匿名性、可审核性等特性 [40]，各特性介绍如下：

去中心化：区块链网络中的交易无需中央机构的认证，可以在任何两个对等点之间进行。通过这种方式，区块链可以显着降低服务器成本, 包括开发成本和运营成本，并减轻中央服务器的性能瓶颈。

持久性：区块链网络中的每个节点都要确认分布在整个网络中的每个交易，交易信息由节点验证并记录在区块中，因此交易数据难以伪造。

匿名性：区块链用户使用特定规则生成的地址与区块链网络进行交互。此外，用户可以生成多个地址来混淆个人信息以避免身份暴露。区块链的去中心化性使得不再有任何中央方保留用户的私人信息。

可审核性：区块链上的每个交易都经过验证并带有时间戳记录，用户可以通过访问分布式网络中的任何节点验证和跟踪以前的记录，这提高了存储在区块链中数据的可追溯性和透明度。

2.1.1　区块链基础架构

区块链基础架构如图 2.1所示，包括数据层、网络层、共识层、激励层、合约层。下面简要介绍各层作用及基本机制。

1) 数据层 (Data Layer)

数据层提供了处理区块链数据的关键技术。区块链获得的数据使用非对称加密、哈希函数和带时间戳的 Merkle 树数据结构捆绑成链块，存储在区块链网络所有节点上。在这一层中，Merkle 树和时间戳可以被认为是区块链分类账的

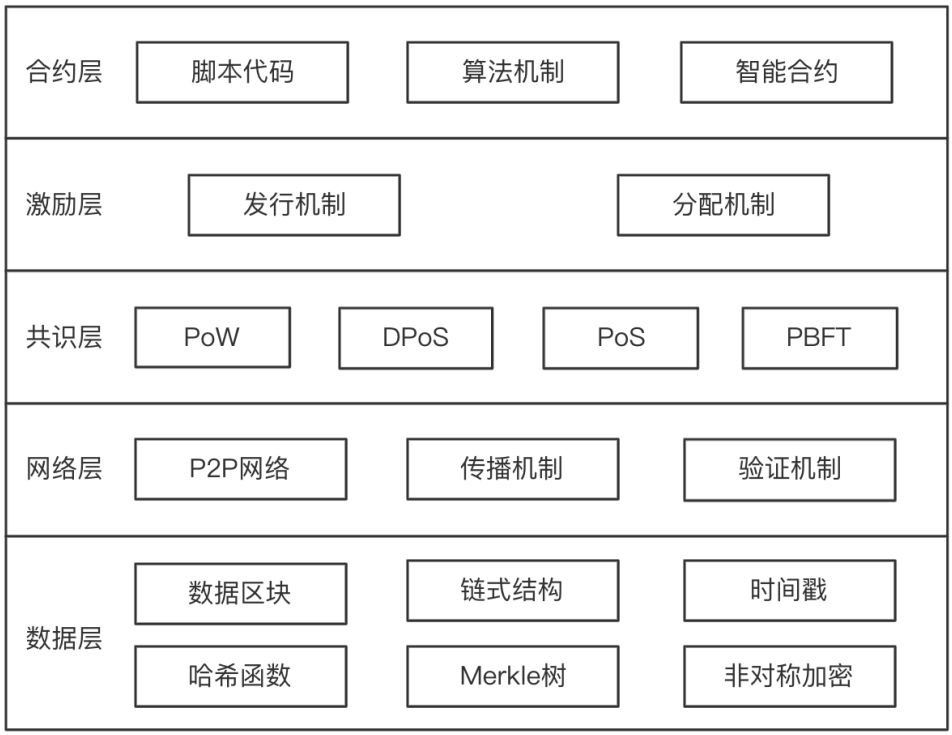


图 2.1: 区块链基础架构模型

两个重要组成部分。前者有助于实现对区块链数据存在和完整性快速、高效的安全验证，而后者则有助于实现区块链数据的可追溯性和精确定位。

2) 网络层 (Network Layer)

网络层指定了分散的通信模型以及分布式网络、数据转发和验证的相关机制。区块链在拓扑上建模为 P2P 网络，网络中的所有参与节点一直在监听网络，并根据预定义的检查列表验证广播的数据或数据块，无效的块将被丢弃，其他块将被转发到相邻节点，只有大多数节点接受的一个区块将被追加到区块链中。

3) 共识层 (Consensus Layer)

区块链使用各种共识算法来保证分布式节点之间共享账本的数据一致性和容错能力 [41]。区块链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)[42]。具体对比在 2.1.2 节描述。

4) 激励层 (Incentive Layer)

激励层将经济奖励纳入区块链系统。众包任务激励机制很简单：一旦创建了新的区块，就会发行一定数量的加密货币作为奖励，并将其分配给获胜的节点，以激励整个网络继续进行数据验证和区块创建。激励层是区块链的关键组成部分和主要驱动力。

5) 合约层 (Contract Layer)

合约层打包了各种智能合约、机制和算法，它们可以用作高级业务逻辑来激活存储在区块链上的静态数据和资产。智能合约可以狭义地定义为一组由区块链存储和保护的自验证、自执行和自执行状态响应规则。智能合约包括一组可执行函数和状态变量。在执行函数时，合约中的状态变量会根据功能中实现的逻辑而变化。区块链网络上的任何用户都可以通过向合约发送交易来触发合约中的功能。合约代码在参与网络的每个节点上执行，作为新块验证的一部分。

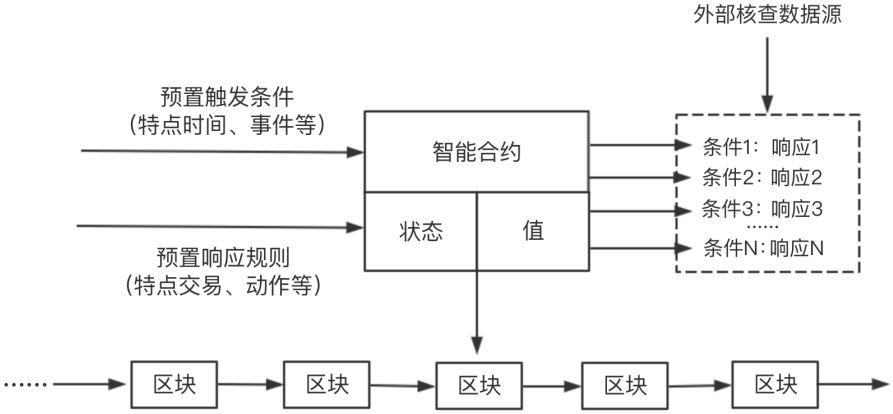


图 2.2: 智能合约运作机理

图 2.2展示了智能合约的运作机理 [43]：一旦一组团体同意一组预定义的条款或规则，他们就可以将它们编成智能合约，对其进行密码签名，并将其广播到 P2P 网络进行验证。经验证的合同将被打包到分类账中的一个大块中。一旦触发了一个或多个前提条件，这些规定和相关动作将被激活并自动执行，而无需人工干预。

本文所述系统用智能合约实现溯源数据的上链和查询功能，众测流程中提交的数据及对数据的操作实时触发智能合约交易，经过节点共识完成区块链账本数据的更新，确保溯源流程真实可靠。

2.1.2　共识机制对比

PoW 是比特币网络中使用的一种共识策略 [7]。在 PoW 中，网络的每个节点都在计算块头的哈希值。区块头包含一个随机数，矿工会频繁更改该随机数以获取不同的哈希值。共识要求计算值必须等于或小于某个给定值。当一个节点达到目标值时，它将该块广播到其他节点，并且所有其他节点必须相互确认哈希值的正确性。如果该区块经过验证，其他矿工则将此新区块附加到自己的区块链中。

PoS 是 PoW 的一种节能替代方案 [44]。PoS 中的矿工必须证明货币数量的所有权，系统中具有最高权益的节点获得共识权利。PoS 通过随机选择要添加到区块链的权益者来代替 PoW 的竞争，PoS 的最简单实现涉及每个区块链分支从区块链货币中统一随机选择，所选货币的所有者将收到附加到选择其货币的分支并同时收集奖励的选项。

PBFT 是一种可容忍拜占庭式错误的复制算法 [45]，可以处理多达 1/3 的恶意拜占庭副本。共识过程一轮确定一个新块，在每个回合中，将根据一定规则选择一个主要节点并且负责处理交易。整个过程可以分为三个阶段：预先准备，准备和提交。在每个阶段中，如果节点从 2/3 以上的节点收到投票，共识将进入下一阶段。因此，PBFT 要求网络知道每个节点。

表 2.1: 共识机制对比

	PoW	**PoS**	**PBFT**
能源节约	否	部分的	完全的
容错性	<25% 算力	<51% 权益	<33% 错误节点
权限身份管理	开放	开放	授权

不同的共识算法具有不同的优缺点，三类共识机制对比如表 2.1所示。在节点身份管理方面，PBFT 需要知道每个矿工的身份以便在每个回合中选择主要矿工，在 PoW 和 PoS 机制下，节点可以自由加入网络；在能源节约方面，PoW 要求矿工连续对块头进行哈希处理以达到目标值，能源消耗较大。PoS 的矿工仍然必须对区块头进行哈希处理以搜索目标值，但是由于搜索空间被设计为有限的，工作量大大减少。PBFT 共识过程中没有挖掘，极大节省了能源；在容错性方面，通常将哈希功率的 51％作为获得网络控制权的阈值，但是 PoW 系统中的自私采矿策略 [46] 可以帮助矿工仅通过 25％的散列能力来获得更多收入，PBFT 需要处理 1/3 个故障节点。

通过上面的对比分析，实用拜占庭容错共识机制具有良好的节约能源能力，在实现区块链账本一致性同时无需多余的计算量，性能较好。本项目采用联盟链结构，经过授权的节点不存在不受控制的作恶现象，且节点数量较少，对共识机制容错性要求较低，因此本项目采用 PBFT 作为共识机制搭建联盟链。

2.1.3　区块链分类

当前的区块链系统根据节点的权限不同分为三类：公有链、联盟链及私有链。表 2.2给出了三类区块链在参与共识群体、数据可读权限等方面的差异:

表 2.2: 区块链分类与比较

	公有链	联盟链	私有链
参与共识群体	所有节点	部分节点	某个组织
数据可读权限	公开	公开或受限	公开或受限
数据不可变性	不可变	可变	可变
交易效率	低	高	高
去中心化	是	部分	否
共识过程	无需许可	需要许可	需要许可

参与共识群体：公有链中每个节点都可以参与共识过程，联盟链中只有一组选定的节点负责验证区块，私有链由一个决定最终共识的组织完全控制；

数据可读性权限：公有链中的所有交易公开可见，联盟链和私有链中存储的信息是公开的还是受限制的由组织或联盟决定；

数据不可变性：公有链中的交易存储在分布式网络的不同节点中，数据几乎不可能篡改，联盟链或私有链的数据由组织和联盟保存，数据有可能被篡改；

交易效率：公有链网络上存在大量节点，因此传播交易和区块时间消耗较多，同时考虑到网络安全性，公有链的限制更加严格，事务吞吐量受到限制并且等待时间长，因此公有链交易效率低于联盟链和私有链；

去中心化：三种类型的区块链之间的主要区别在于，公有链是完全去中心化的，联盟链是部分中心化的，而私有链由单个组织控制是完全中心化的；

共识过程：公有链允许任意节点加入共识，联盟链和私有链中的节点需要联盟或组织的许可才可以参与共识过程。

本文研究众测数据溯源，众测流程参与方的数据属于私有财产无法公开，且参与方之间属于合作关系，是典型的联盟链结构。因此，本文采用联盟链技术，众测活动参与方作为联盟链节点共同验证溯源数据，确保溯源数据真实可靠。

2.1.4 Hyperledger Fabric

Hyperledger Fabric[1]（以下简称 Fabric）是在 Linux 基金会下由 IBM 和 Digital Assets 开发的开源区块链基础架构，具有模块化体系结构、智能合约、可配置的共识机制以及节点成员权限服务。Fabric 架构根据提供的服务进行逻辑组织，包括区块链服务、会员服务和链码服务。

区块链服务是 Fabric 架构的核心部分，包括共识管理、分布式账本、帐本

[1]hyperledger.org/projects/fabric

存储和点对点协议。共识管理负责提供共识算法的接口，Fabric 支持本项目选择的 PBFT 共识算法。分布式账本是智能合约用来在交易执行期间存储相关状态信息的数据库，交易由链码执行并更新世界状态，世界状态存储在 InnoDB 中。Fabric 中消息的结构由协议缓冲区定义，通过使用不同的消息，网络可以发现节点并执行身份验证及交易。

会员服务包括用户注册、身份验证等功能。Fabric 支持根据用户角色分配权限，权限分配及身份管理由公钥基础架构支持。Fabric CA 是身份证书管理组件，负责用户身份证书及交易证书的颁发与管理。用户注册必须经过注册证书颁发机构颁发有效证书来验证身份，交易证书颁发机构颁发交易证书后，用户方可在网络上发送交易。

链码（Chaincode）是 Fabric 中的智能合约，用户可以编写自定义的链码来实现业务逻辑（交易）。链码部署在区块链节点上，在 Docker 中运行。Fabric 节点分为验证节点和非验证节点。验证节点负责在 Fabric 网络中验证交易并维护分布式账本，非验证节点发起交易，不负责执行和验证。验证节点使用拜占庭容错算法作为执行复制状态机的共识协议。复制状态机接受的交易类型包括唤醒交易、执行交易和查询交易。唤醒交易是从节点上获取链码，并准备执行；执行交易即为链码执行交易更新账本世界状态，并提供交易是否成功信息；查询交易是指查询世界状态数据并返回给用户。

2.2　众包测试流程

众测主要参与者包括需求方、众测平台、众测工人以及平台审核人员。需求方是有软件测试需求的公司，可以在众测平台提交测试需求。众测平台负责整个众测流程的管理，包括任务发布、报告处理、报告交付等。众测工人为在众测平台完成测试任务的个体，工人线下测试需求软件后提交测试报告，并能获得相应报酬。平台审核人员负责审核需求方提交的需求数据是否符合规范，评审并整合测试报告。

众包测试主要流程如图 2.3所示。首先，需求方将待测软件及测试需求文档上传到众测平台；接着，众测平台发布通过审核的测试任务并召集众测工人进行测试；然后，众测工人选择测试任务进行测试，完成测试后提交测试报告；最后，审核人员审核测试报告，并整理最终测试报告反馈给需求方。

不同众测平台测试报告提交与审核流程不同。慕测众测平台测试报告提交阶段采用在线填写测试报告的方式。众测工人可以在测试报告中创建不同的缺陷报告，缺陷报告的详细信息包括应用截图及相应的文字描述，测试报告提交后无法修改。测试报告审核以评分形式进行，审核人员对于每个缺陷报告打分，

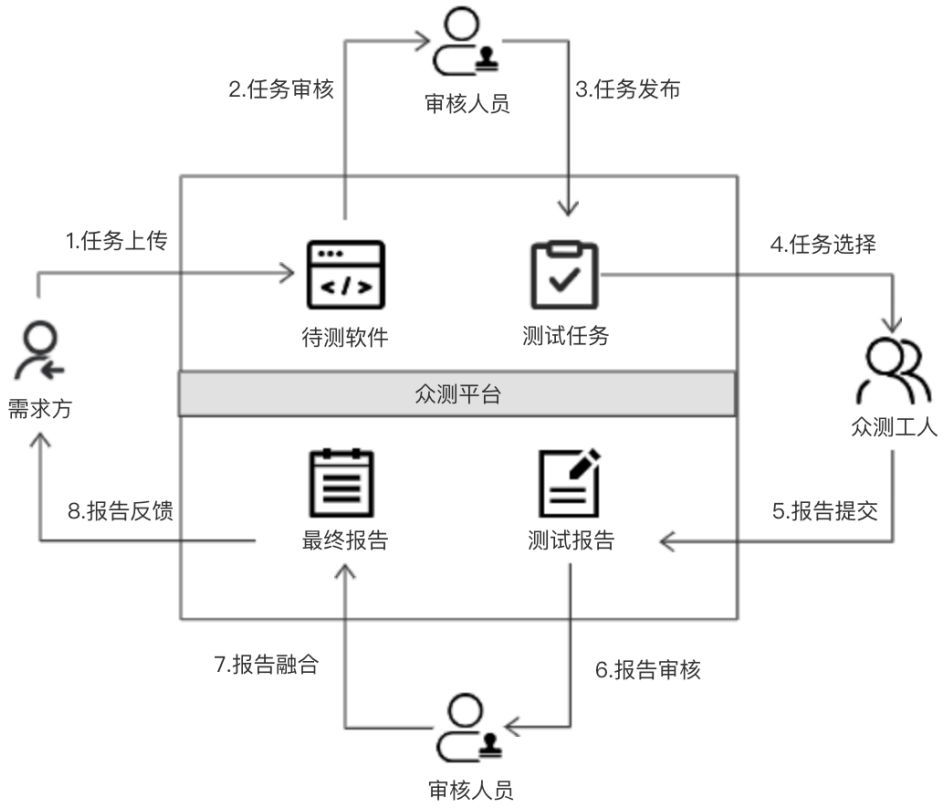


图 2.3: 众包测试流程

工人测试报告中包含的所有缺陷报告得分结果累加作为此次任务工人的总得分。在测试报告融合阶段，审核人员筛选高得分且错误类型不同的缺陷报告组成最终报告，最终报告是众测任务的结果反馈给需求方。

2.3　数据溯源技术

随着可用存储空间的增加和信息流加速发展，人们对于数据的创建过程和来源信息越来越感兴趣。数据溯源是指通过一定技术实现对数据的来源、创建过程的追溯，不同系统中数据溯源功能不同。在科学领域，将实验过程和分析与实验产生和使用的数据关联起来，对于实验数据的溯源可以帮助人们更好的验证实验方法。在商业领域，有合作关系的公司会交换商业数据，当数据质量较差时，数据溯源被用来识别和纠正不良数据的来源，以保证数据质量。

数据溯源首先要分析数据在系统中转换的方式，确定数据溯源体系架构，接着利用数据溯源方法，实现对数据的追溯。下面介绍数据溯源体系架构及数据溯源方法。

2.3.1　数据溯源体系架构

数据产品的溯源信息集中在两个概念上：数据产品的来源和为产生数据而进行的转换。数据处理体系结构是指这些过程执行，使用数据产品并带来数据转换的方式，主要有面向服务的体系结构和数据库结构。

面向服务的体系结构通常允许以诸如 WSFL 或 BPEL 之类的语言编写的工作流文档的形式指定转换图。转换过程被建模为 Web 或 Grid 服务，数据产品是这些服务的输入和输出。可以通过跟踪工作流程的执行情况并通过逻辑或物理 ID 识别每个服务的输入和输出数据产品，来确定工作流程中涉及的数据集的来源。这些跟踪可以由工作流引擎自动生成，然后由用户注释，以在数据产品上提供其他元数据。可以将有关工作流的静态信息与运行时详细信息结合起来，以构成数据溯源信息。如果工作流引擎未收集工作流跟踪，则服务的分布式性质将使每个服务提供者和客户端都有责任来生成它们的调用日志，这些日志被汇总以形成工作流的溯源。

在数据库体系结构中，更新查询和函数形成了转换数据的数据处理组件。关系数据库中的数据产品可以是数据库中更细粒度的视图、表、元组、属性或数据项。数据仓库是使用数据库的数据处理系统的原型，仓库通过提取、清理和转换步骤导入数据，这些步骤建模为对多个数据源的查询。除了选择和联接之类的典型关系运算符外，这些查询还可以调用用户定义的函数，这些函数实现为存储过程调用。利用这种丰富的功能，可以构造复杂的数据流图，数据流图将作为查询的一部分由数据库执行。这提供了类似于面向服务的体系结构中工作流的功能。数据库中查询结果的溯源使用数据库中属性的注释来表示。

2.3.2　数据溯源方法

数据溯源方法主要为“Eager”法和“Lazy”法 [47]。Eager 法通过随数据转换时附带的数据来源信息进行溯源，依赖于对源数据项和数据转换流程的注释；Lazy 法仅在需要时对数据溯源，依赖于数据转换的反转或输入流的跟踪。

Eager 法又称为元数据支持法、标记法、归因法或注释法。Liang[11] 等人使用标记法来进行云系统环境下的数据溯源。他们使用文件钩子 (File Hooks) 实时监听云系统中用户对文件的操作，在文件发生创建、更改、分享等操作时，将用户信息、时间戳、文件信息、操作影响者标注到文件操作信息中作为溯源信息记录到数据库中。在云系统中数据出现恶意篡改时，可以通过数据溯源信息及时确定责任人，并确定恶意数据的影响范围，完成对云系统的数据管控。

Eager 法数据溯源可以通过查看与一条输出数据相关联的注释来完全确定出处，因此无需探究源数据库。然而，在大型系统中，工作流程复杂，数据流转

频繁，对源数据的标注信息可能超过数据本身的大小，给数据存储带来巨大压力。针对这种情况，Lazy 法构造可逆函数描述数据流转过程，可逆函数通常是“弱可逆”的，在返回所有正确答案时也可能产生错误的答案，因此 Lazy 法要求提供验证函数来验证所有可逆函数反转得到的结果，剔除错误答案。

本系统针对众测数据进行溯源，主要追溯需求方需求文档、选手提交的测试报告在众测平台中的流转过程，数据量不大，因此选择 Eager 法，通过在数据流转过程中添加标注信息，实现数据溯源。

2.4　本章小结

本章主要介绍系统实现相关的技术。首先，介绍了区块链技术，包括区块链基础架构、共识机制、区块链分类及本系统采用的联盟链架构 Fabric。接着介绍了众包测试技术，主要对本系统溯源的慕测众测平台的众测流程进行介绍。最后介绍了数据溯源技术，包括数据溯源体系架构及数据溯源方法。本章的主要目的是为系统的实现提供理论支持与技术支撑。

第三章　溯源系统的需求分析与设计

本章概述系统的需求分析与设计。首先，本章给出系统的整体概述，简要介绍系统目标与总体功能。然后，本章通过系统涉众分析进行系统用例设计，并详细描述用例。接着，本章对系统进行总体设计和模块划分，描述系统逻辑视图、开发视图、进程视图和物理视图。最后，本章对系统持久化模型进行详细描述。

3.1　系统整体概述

众包测试在不断发展的同时，众测活动参与者对于众测数据的真实性提出了更高的要求。众测数据溯源通过追溯数据的来源及数据产生过程保证数据的可靠性，避免数据欺诈。然而数据溯源信息本身存在安全隐患，溯源信息无法得到众测参与方的信任。本系统根据慕测平台的众测流程实时采集众测数据，利用区块链技术联合需求方、众测工人方、平台方搭建联盟链，通过智能合约将溯源信息存储到区块链上。存储过程作为联盟链网络上的交易多方共识，解决数据溯源的安全及信任问题。

系统主要由前端交互平台、数据溯源服务、联盟链客户端三个部分组成。前端交互平台根据用户不同身份展示众测数据溯源的相关操作入口，用户进入相关页面进行数据溯源，验证数据真实性。数据溯源服务是系统业务逻辑实现，主要根据众测用户数据溯源需求分为需求数据服务、测试报告服务、最终报告服务及众测任务服务。其中，需求数据服务负责需求数据、需求审核数据及最终报告数据的采集及溯源验证；测试报告服务负责测试报告及报告审核数据的采集及溯源验证；最终报告服务负责最终测试报告融合数据的采集及溯源验证；众测任务服务主要是针对平台方，提供众测活动各个阶段的数据溯源服务。联盟链客户端封装访问联盟链操作，提供通道创建、智能合约调用等服务，为业务层提供易于操作的接口，方便业务层业务开发。

3.2　系统需求分析

3.2.1　系统涉众

如表 3.1所示，本系统的用户涉众为需求方、众测工人方和众测平台。参与方作为区块链节点搭建联盟链，系统实时接入众测数据，并通过智能合约将数据以溯源模型的格式存储到联盟链上，溯源信息作为区块链账本信息保存，多

方公证。需求方可以追溯需求数据和最终报告的提交及审核信息，众测工人可以追溯测试报告提交及审核信息，众测平台可以对众测流程进行溯源。

表 3.1: 用户角色说明

用户角色	用户特征
需求方	在众测平台上提交测试需求及待测软件的公司或个人，需要通过系统进行众测需求的溯源，包括需求提交信息及审核信息。
众测工人方	在众测平台上完成众测任务的个人，需要通过系统进行测试报告的溯源，包括报告提交信息及审核信息。
众测平台	负责整个测试流程管理的平台，需要将众测数据发送给溯源系统，并通过系统进行测试流程的溯源。

3.2.2　功能性需求分析

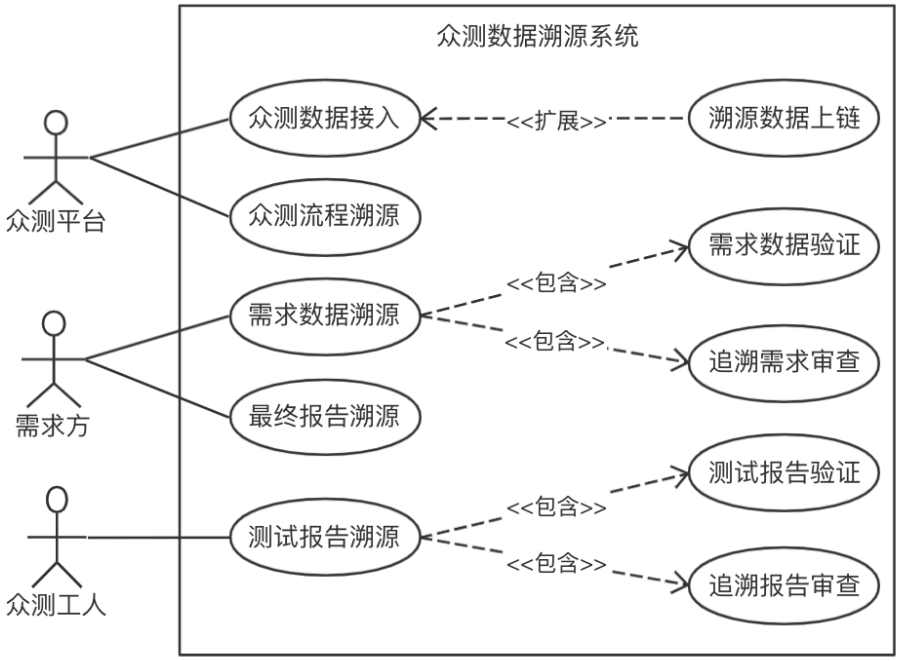


图 3.1: 系统用例图

本系统针对慕测众测平台的众测数据进行数据溯源，根据对系统涉众众测平台、需求方、众测工人的用户特征分析，得到如图 3.1所示系统相关用例。系统用例主要包括众测数据采集、众测流程溯源、需求数据溯源、最终报告溯源和测试报告溯源。数据采集扩展为数据联盟链存储，数据溯源包含数据来源验证及查看审查结果。下面将对系统需求进行详细用例描述。

表 3.2: 众测数据接入用例描述

ID	UC1
用例名称	众测数据接入
用例描述	众测活动中众测平台将数据实时发送给溯源系统
参与者	众测平台方
优先级	高
前置条件	无
后置条件	排序节点对交易排序，对等节点验证交易更新账本数据
正常流程	1. 在需求方提交需求数据时将需求数据发送给溯源系统； 2. 需求数据审核后将审核结果及审核者信息发送给溯源系统； 3. 众测工人提交测试报告后将测试报告发送给溯源系统； 4. 测试报告审核后将报告得分及审核者信息发送给溯源系统； 5. 最终测试报告融合后将最终报告融合信息发送给溯源系统。

众测数据接入用例描述如表 3.2所示。众测平台实时将众测数据发送给溯源系统。具体地，众测平台将需求方提交的需求文档及待测软件、众测工人提交的测试报告，平台审核人员对于需求任务的审核结果、对测试报告的评审结果，以及融合最终测试报告的数据发送给本系统。

表 3.3: 众测流程溯源用例描述

ID	UC2
用例名称	众测流程溯源
用例描述	平台方在溯源系统查看众测任务流程
参与者	众测平台方
优先级	高
前置条件	众测数据已接入溯源系统
后置条件	无
正常流程	1. 点击需求提交阶段查看需求提交情况； 2. 点击需求审核阶段查看需求审核情况； 2. 点击报告提交阶段查看部分测试报告； 3. 点击报告审核阶段查看报告审核情况； 4. 点击报告融合阶段查看最终报告情况
可选流程	2.a 点击“查看全部”按钮，系统显示所有报告信息

众测流程溯源用例描述如表 3.3所示。众测平台管理员可以查看众测任务溯源信息，系统展示需求提交及审核结果、已提交测试报告数据、已审核缺陷报告数据、最终报告融合数据。用户点击“查看全部”按钮可以查看所有数据。

表 3.4: 需求数据溯源用例描述

ID	UC4
用例名称	需求数据溯源
用例描述	需求方在系统查看需求数据及交付的最终报告数据
参与者	需求方
优先级	高
前置条件	用户已被识别与授权
后置条件	无
正常流程	1. 系统展示用户参与的众测任务及需求数据、任务状态； 2. 用户点击数据验证； 3. 系统展示链上数据哈希与众测平台数据哈希比对结果； 4. 用户点击区块链信息； 5. 系统展示数据上链详细信息，包括交易时间、区块链哈希等。

需求数据溯源用例描述如表 3.4所示。需求数据溯源主要展示链上需求数据，包括任务名称、需求文档、待测软件、状态、状态更新时间，并提供数据验证及区块链信息查看功能。

表 3.5: 最终报告溯源用例描述

ID	UC5
用例名称	最终报告溯源
用例描述	需求方和众测平台方可以在系统查看最终报告来源信息
参与者	需求方、众测平台方
优先级	高
前置条件	用户已被识别与授权
后置条件	无
正常流程	1. 用户点击最终报告来源查看； 2. 系统展示最终报告详细数据； 3. 用户点击具体缺陷报告的数据验证； 4. 系统展示数据验证结果； 5. 用户点击具体缺陷报告的区块链信息； 6. 系统展示缺陷报告上链详细信息，包括交易时间、区块链哈希等。

最终报告溯源用例描述如表 3.5所示。最终报告溯源主要展示链上最终报告溯源信息，为需求方与众测平台方服务。最终报告数据主要包括最终报告包含的缺陷报告来源信息，即报告提交者、报告得分、报告审核者。为验证数据，系统提供缺陷报告数据验证和区块链信息查看功能。

表 3.6: 测试报告溯源用例描述

ID	UC6
用例名称	测试报告溯源
用例描述	众测工人在系统查看已提交的测试报告数据及审核数据
参与者	众测工人
优先级	高
前置条件	用户已被识别与授权
后置条件	无
正常流程	1. 用户点击参与的众测任务； 2. 系统展示用户已提交的测试报告、当前状态、得分等信息； 3. 用户点击状态为已审核的测试报告； 4. 系统展示测试报告包含的缺陷报告具体审核数据； 5. 用户点击数据验证； 6. 系统展示数据验证结果； 7. 用户点击区块链信息； 8. 系统展示数据上链详细信息。

测试报告溯源用例描述如表 3.6所示。众测工人可以在溯源系统上查看在众测平台上提交的测试报告及当前状态，对于已审核的测试报告可以查看审核结果，审核信息具体到测试报告包含的每个缺陷报告的得分及对应评审人。工人可以通过比对链上数据哈希与众测平台数据哈希验证数据是否被篡改，系统同时提供数据上链时交易信息的查看。

3.2.3　非功能性需求分析

在非功能性需求方面，本系统是面向众测参与方提供众测数据溯源服务，数据接入随着众测任务实时进行。为了系统用户能够查询到真实可靠的溯源数据，系统需要对性能、易用性、可扩展性有一定要求。

性能：系统前端各个界面的响应时间不高于 1 秒，降低用户的等待时间，提高系统的交互体验。前端异步调用服务端的接口时，响应时间不高于 1 秒。服务端接口可承受 200 个用户并发使用。

易用性：系统界面简洁美观，布局合理，容易理解，人机交互体验应该满足大部分用户的要求。

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

3.3　系统总体设计

系统需求分析给出了系统需要实现的功能，本节结合系统需求进行总体架构设计。通过系统架构设计能够得到系统的层次结构并将需求转化为具体功能模块，为后续模块设计提供设计基础。为更好实现系统，本小结借助系统逻辑视图、系统开发视图、系统进程视图和系统物理视图进行系统概要设计。

3.3.1　系统架构

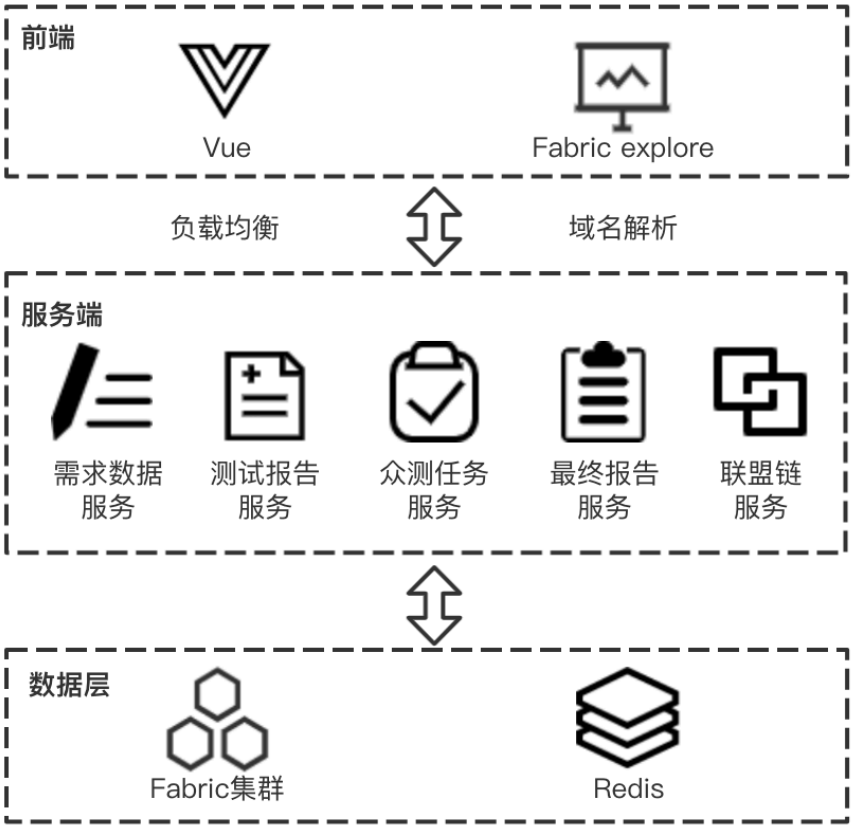


图 3.2: 系统架构图

本系统架构如图 3.2所示。系统采用三层架构分为前端、服务端和数据层。前端负责系统与用户的交互，采用开源 Vue 框架实现页面展示逻辑，Fabric explore 是 Fabric 区块链网络浏览器，负责溯源系统的区块链信息查看。前端与服务端之间通过 Nginx 实现负载均衡，前端通过解析服务端接口地址获取相关数据。服务端采用 Springboot 框架进行开发，包括需求数据服务、测试报告服务、众测任务服务、最终报告服务、联盟链服务五个模块。其中，需求数据服务负责需求提交、需求审核阶段数据的采集，以及针对需求方的数据溯源；测试报告服务负责测试报告提交、报告审核阶段数据的采集，以及针对众测工人的数据溯源；

最终报告服务提供报告融合阶段数据的采集，以及最终报告溯源服务；众测任务服务主要是针对众测平台方提供众测任务整个流程的溯源；联盟链服务主要提供智能合约部署、节点账户验证、账本数据存取等服务。数据层包括区块链节点账本数据存储及 Redis 缓存。为实现溯源数据真实可靠，溯源数据存储于联盟链账本上，参与方作为联盟链节点共同维护账本数据。Redis 负责用户登陆数据缓存，提高系统运行效率。

3.3.2　4+1 视图

本文以 Philippe Kruchten 提出的“4+1”视图 [48] 方法给出系统的架构设计。本节从逻辑视图、开发视图、进程视图和物理部署视图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

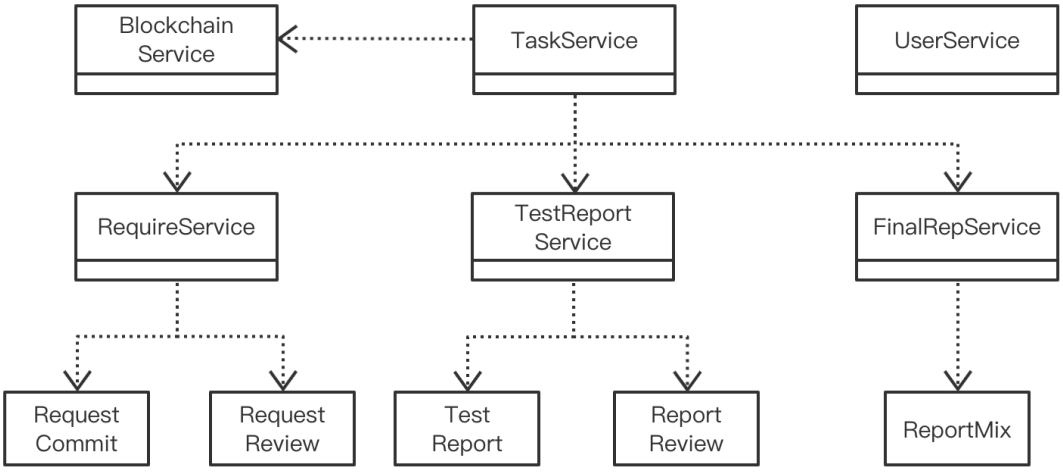


图 3.3: 系统逻辑视图

逻辑视图描述了系统包含的对象模型。如图 3.3所示，系统数据实体抽象为需求提交（RequestCommit）、需求审查（RequestReview）、测试报告（TestReport）、报告审查（ReportReview）、报告融合（ReportMix）等，系统中的基础服务抽象为用户服务（UserService）、任务服务（TaskService）、区块链服务（BlockchainService）、需求服务（RequireService）、测试报告服务（TestReportService）、最终报告服务（FinalRepService）等。UserService 提供用户登陆验证功能。TaskService 提供众测平台方溯源测试任务功能。BlockchainService 提供区块链功能，包括智能合约部署、调用等。RequrieService 提供需求数据的上链及溯源功能。TestReportService 提供工人测试报告的上链及溯源功能。FinalRepService 提供最终报告上链及溯源功能。

(2) 开发视图

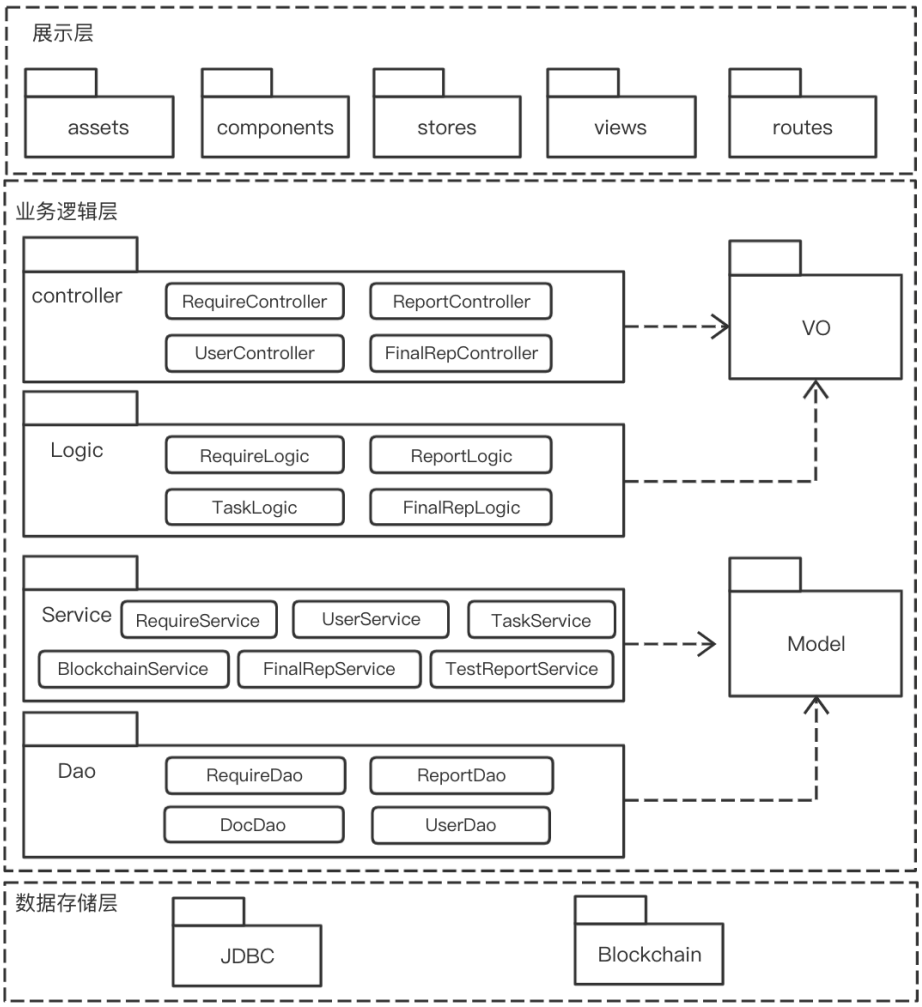


图 3.4: 系统开发视图

通过用户的角度对系统逻辑结构进行设计，明确了系统的逻辑结构。接下来从开发者的角度对系统的工程开发结构进行设计。如图 3.4所示，系统总体结构遵循典型的分层架构设计，实现前后端分离。有利于系统的代码维护。系统总体分为展示层、业务逻辑层和数据存储层三层。展示层中，assets 包存放资源目录，components 包存放可服用组件，stores 包存放应用级数据，views 包存放业务界面，routes 包存放路由数据。业务逻辑层采用分层架构，Controller 包负责与前端界面交互，管理路由请求，Logic 包负责处理复杂的业务请求，Service 将 Dao 层中对数据的存储及查询封装，VO 是逻辑层与前端交互的数据模型，Model 是逻辑层与数据层交互的数据模型。数据存储层封装了数据库交互接口，包括与传统数据库交互的 JDBC 以及与区块链存储交互的 Blockchain。

(3) 进程视图

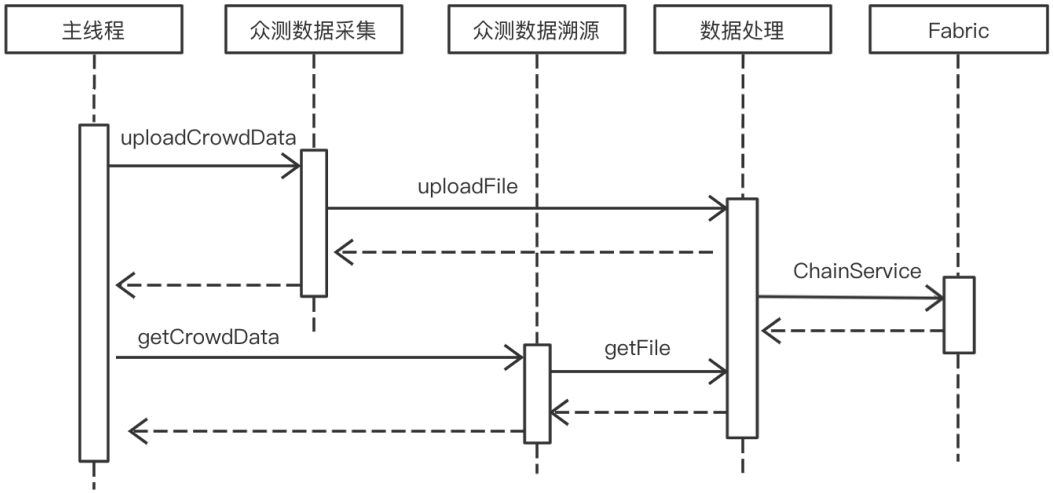


图 3.5: 系统进程视图

通过从用户和开发者的角度对系统结构进行设计，明确了系统划分的独立服务。接下来从系统各服务进程间交互的角度介绍系统服务间的调用关系。图 3.5为众测数据溯源系统的进程视图，主线程分为众测数据采集和众测数据溯源两部分调用。众测数据采集指在众测活动需求提交、需求审查、测试报告提交、测试报告审核、最终报告融合阶段实时采集众测数据。数据处理后发送给联盟链，联盟链调用智能合约服务 ChainService 将数据存储于交易账本上，存储过程作为联盟链网络中的交易由各个节点共识验证。众测数据溯源首先判断前端用户类型，针对需求方提供需求数据、需求审核数据溯源，并在任务已完成时提供最终报告溯源。针对工人方提供测试报告及报告审核数据溯源。针对平台方提供五个阶段的数据溯源。溯源服务调用 ChainService 查询账本数据，查询作为交易由智能合约完成，不需要节点共识。

(4) 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。系统前端与服务层单独部署在 Docker 容器中，用户通过浏览器访问系统前端，前端通过 HTTP 协议调用业务层服务，业务层暴露服务地址给上层。业务层通过 GRPC 协议与底层 Fabric 节点交互。Fabric 节点包含了 Fabric 相关组件，包括智能合约构件 Chaincode、物理载体构件 Fabric peer 以及节点用户身份验证构件 Fabric CA。

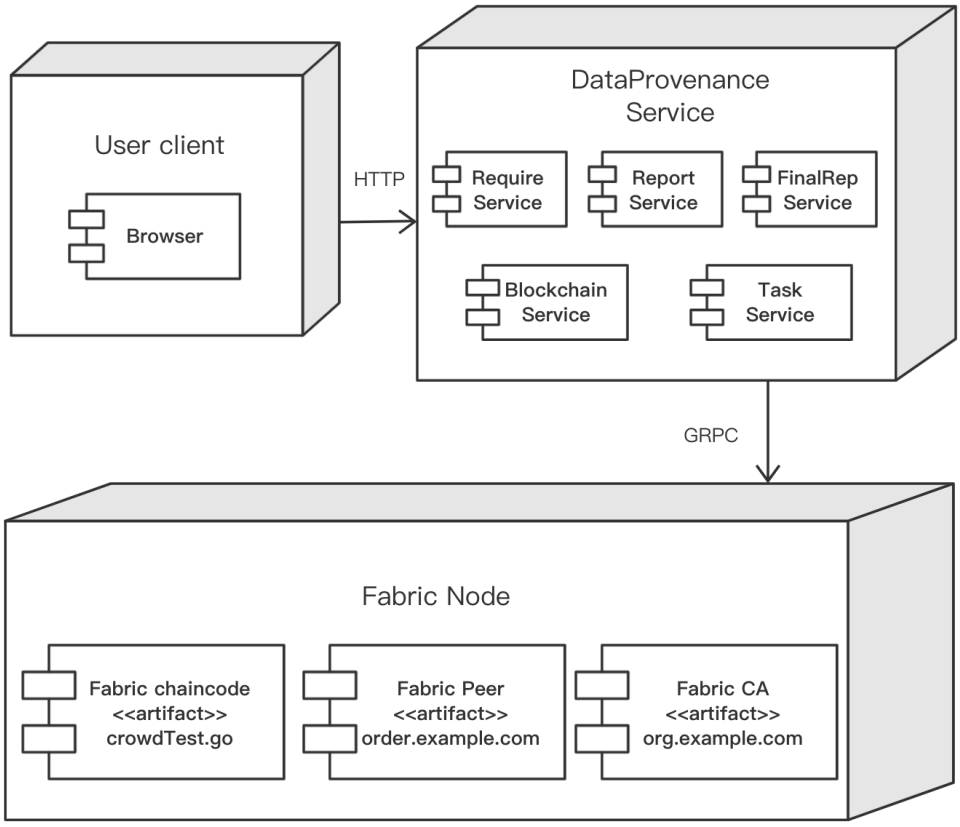


图 3.6: 系统物理视图

3.4　持久化模型设计

本项目采用开源的 Hyperledger Fabric（以下简称 Fabric）作为联盟链基础设施。Fabric 将每一笔交易数据写入区块，排序节点将网络中的交易排序发送给对等节点，经过对等节点验证的区块数据将更新到世界账本。账本数据由各个节点同时保存并保持一致，只能追加，不能更改与删除。Fabric 支持 CouchDB[1]映射世界账本中的数据，CouchDB 是开源的面向文档的键值数据库，提供以 JSON 作为数据格式的 REST 接口来对其进行操作。

本系统通过 Java 的 UUID 类生成通用唯一识别码作为业务数据键值对中的键（Key），业务对象即众测数据作为键值对中的值（Value）。根据众测活动流程中涉及的数据，本系统设计了众测数据的联盟链存储模型，主要是需求提交数据 RequestCommit、需求审核数据 RequestReview、测试报告提交数据 TestReport、测试报告审核数据 ReportReview、测试报告融合数据 ReportMix 以及众测任务状态数据 TaskState，下面分别介绍各个数据模型。

[1]https://couchdb.apache.org/

表 3.7: 联盟链对象 RequestCommit 属性表

字段	含义	类型
taskId	测试任务 Id	string
taskName	测试任务名称	string
requestId	需求 Id	string
requesterId	需求者 Id	string
requesterName	需求者名称	string
requestDocHash	需求文档哈希值	long
requestDocName	需求文档名称	string
testSoftwareName	测试软件名称	string
updateTime	更新时间	long

表 3.7是需求方在众测平台上提交需求任务数据在联盟链中的字段描述，主要记录需求数据与众测任务之间的映射。平台方溯源时通过 taskId 查询该任务所有需求提交数据，taskName 为众测任务名称。需求方溯源时通过 requestId 查询对应需求人员提交的所有需求数据，requester 为需求者名称。众测平台发布需求任务时，需求文档代表了需求方最明确的测试需求，为了确定众测工人得到的需求文档满足需求方要求，未被篡改，联盟链中存储了需求文档的哈希值 requestDocHash，哈希值的唯一性与联盟链账本数据的不可变性保证了需求数据的真实可靠。requestDocName 为需求文档名称，testSoftwareName 为待测软件名称。updateTime 为数据提交时间。

表 3.8: 联盟链对象 RequestReview 属性表

字段	含义	类型
taskId	测试任务 Id	string
taskName	测试任务名称	string
requestId	需求 Id	string
requestReviewer	需求审核者	string
reviewResult	审核结果	string
updateTime	更新时间	long

表 3.8是平台对需求数据审核信息在联盟链中的字段描述，主要记录需求审核数据与需求数据之间的映射。taskId 在平台方查看需求审核阶段数据时使用，requestId 对应 RequestCommit 里的需求 Id，requestReviewer 记录了审核需求数据的审核者信息，当需求方对审核结果出现异议时能及时追溯到责任人。reviewResult 记录需求审核结果。

表 3.9: 联盟链对象 TestReport 属性表

字段	含义	类型
taskId	测试任务 Id	string
taskName	测试任务名称	string
testReportId	测试报告 Id	string
testReportName	测试报告名称	string
reportHash	测试报告 hash	long
bugReportList	缺陷报告列表	list
workerId	测试工人 Id	long
workerName	测试工人名称	long
updateTime	更新时间	long

表 3.9是众测工人提交的测试报告数据在联盟链中的字段描述，主要记录测试报告数据与测试任务之间的映射。testReportId 为测试报告唯一标识，testReportName 为报告名称。为了防止测试报告被恶意篡改，联盟链存储测试报告唯一哈希值 reportHash。bugReportList 为测试报告包含的缺陷报告列表，每个缺陷报告包含了缺陷报告 Id 及缺陷报告名称。根据众测平台提交测试报告的要求，众测工人只能根据模版提交一份测试报告，测试报告中可以包含多个缺陷报告。workerId 为众测工人唯一标识，workerName 为工人名称。

表 3.10: 联盟链对象 ReportReview 属性表

字段	含义	类型
taskId	测试任务 Id	string
taskName	测试任务名称	string
testReportId	测试报告 Id	string
bugReportId	缺陷报告 Id	string
bugReportScore	缺陷报告得分 hash	long
reportReviewer	报告审核者	string
updateTime	更新时间	long

表 3.10是平台对于测试报告审核数据在联盟链中的字段描述，主要记录缺陷报告审核数据与缺陷报告所属测试报告之间的映射。平台方溯源报告审核数据时，后台通过 taskId 获取该众测任务下的所有报告审核数据。众测平台审核测试报告时以测试报告中的缺陷报告为单位，给每个缺陷报告打分，测试报告的总分数由缺陷报告得分累加获得。bugReportId 为缺陷报告唯一标识，bugReportScore 为缺陷报告得分，reportReviewer 为缺陷报告审核者。

表 3.11: 联盟链对象 ReportMix 属性表

字段	含义	类型
taskId	测试任务 Id	string
taskName	测试任务名称	string
BugReportList	缺陷报告列表	list
ReportHash	缺陷报告 Id	long
ReportMixer	最终报告融合者	string
updateTime	更新时间	long

表 3.11是平台融合测试报告生成最终报告数据在联盟链中的字段描述。主要记录最终报告与众测任务之间的映射，最终报告由报告融合者选取的缺陷报告组成，一次众测任务只有一份最终报告交付给需求方。

表 3.12: 联盟链对象 TaskState 属性表

字段	含义	类型
taskId	测试任务 Id	string
taskName	测试任务名称	string
TaskState	测试任务状态	Integer
updateTime	更新时间	long

表 3.12是记录众测任务状态数据在联盟链中的字段描述。任务状态分为 5 种状态，分别为需求提交、需求审核、测试报告提交、测试报告审核以及最终报告融合，众测活动中各个阶段提交的数据存储于联盟链时会更新任务状态。

3.5　本章小结

本章对系统需求进行分析并进行设计。首先对系统进行整体概述，描述系统总体规划与相关模块。然后根据系统不同参与方描述涉众的需求分析，给出系统用例图，并详细描述系统用例分析，阐述系统需要满足的功能性需求与非功能性需求。接着对系统进行架构设计，从逻辑视图给出系统各个服务之间的关系，从开发视图给出系统模块层次结构，从进程视图给出系统交互顺序，从物理视图给出系统部署方案。最后对系统持久化模型设计进行详细描述。

第四章　溯源系统的详细设计与实现

本章在第三章分析与设计的基础上，通过类图、时序图及关键实现代码，详细介绍系统需求数据服务、测试报告服务、最终报告服务、众测任务服务、联盟链服务的设计与实现，并描述联盟链配置与部署方案。

4.1　需求数据服务

4.1.1　需求数据服务的详细设计

需求数据服务主要针对需求方的数据溯源提供服务，包括需求数据采集与溯源。采集的需求数据包括需求方提交的待测软件、测试文档以及平台对于需求任务审核结果。需求数据溯源根据众测任务当前状态展示不同信息，在任务完成后会展示最终报告数据。需求数据服务核心类图如图 4.1所示。ReqController 类是对外提供的控制器。RequireLogic 类是具体业务逻辑的实现。RequireService 类根据具体需求数据调用 DocDao 将数据存入联盟链中。

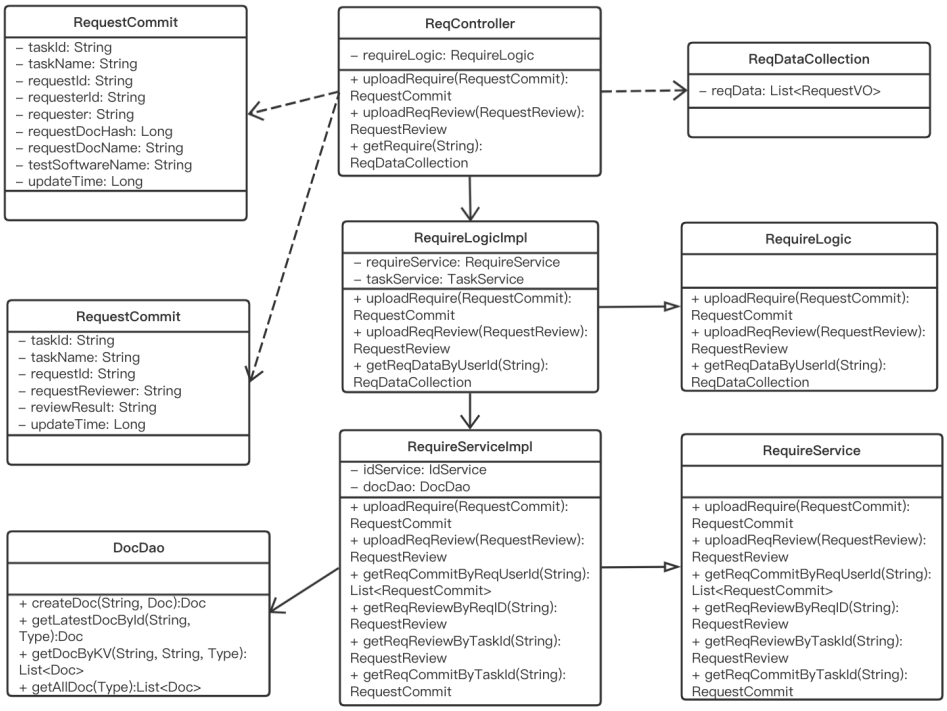


图 4.1: 需求数据服务核心类图

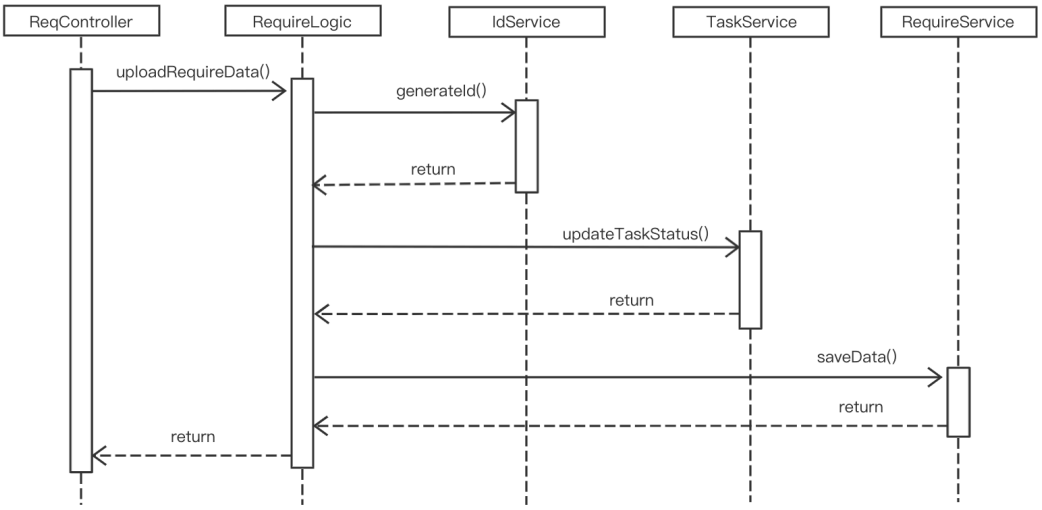


图 4.2: 需求数据采集顺序图

需求数据采集顺序图如图 4.2所示。ReqController 在接收需求数据后，调用 RequireLogic 中的需求数据上传接口。RequireLogic 调用 IdService 生成唯一 Id 作为需求数据在联盟链中存储的键，接着调用 TaskService 更新任务状态，最后调用 RequireService 数据存储接口，将需求数据 Id 及内容封装发送给联盟链。

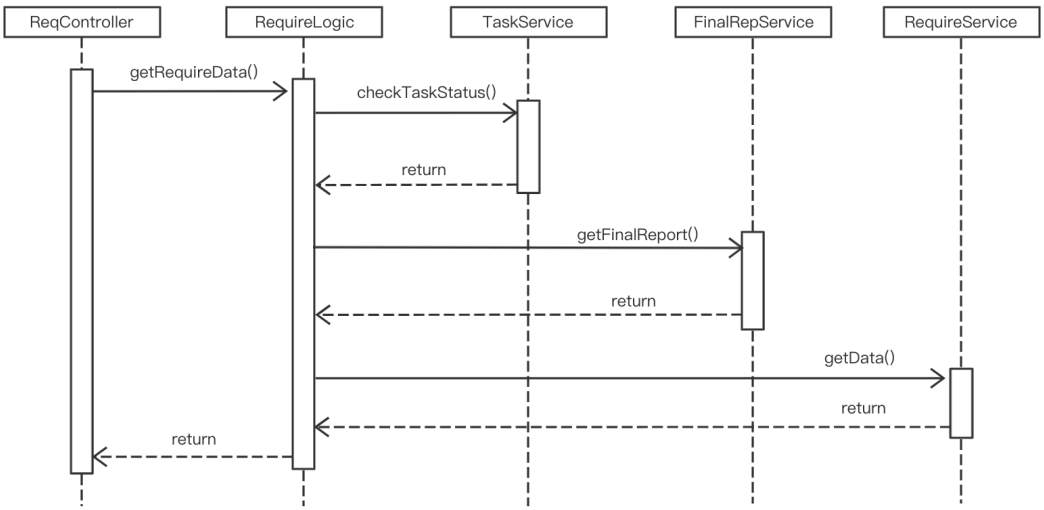


图 4.3: 需求数据溯源顺序图

需求方需求数据溯源顺序图如图 4.3所示。ReqController 在接收前端发送的需求数据溯源要求后，首先检查众测任务状态，根据任务不同状态提取不同数据。在任务提交及审核阶段调用 RequireService 获取需求数据及需求审核数据，在任务完成阶段调用 FinalRepService 获取最终报告数据。

4.1.2　需求数据服务的实现

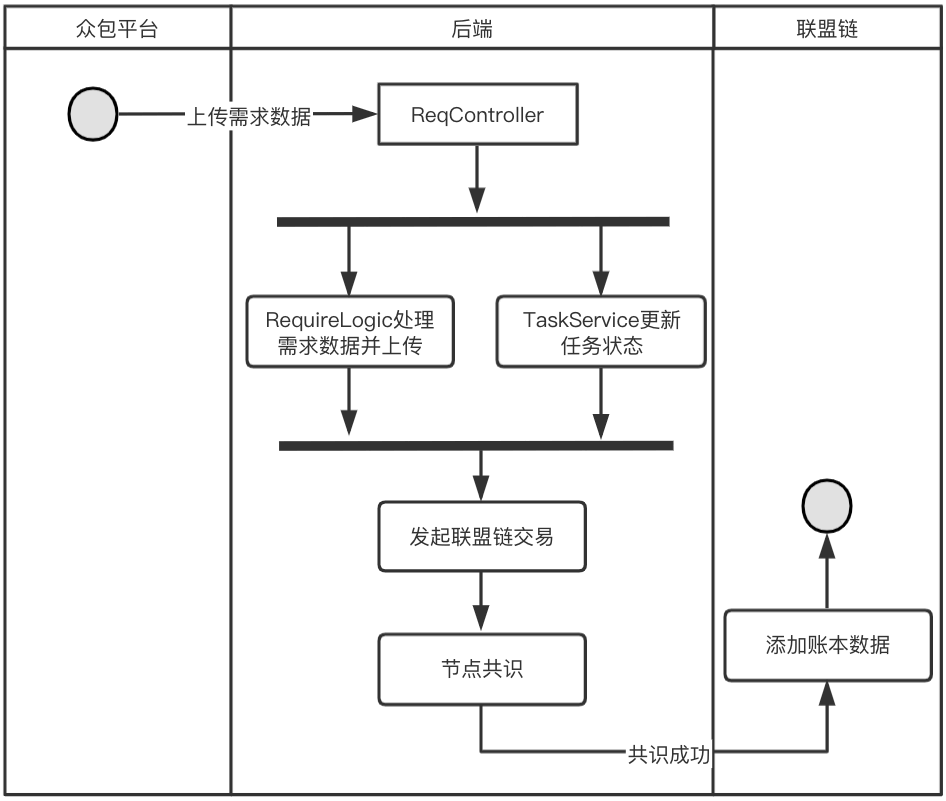


图 4.4: 需求数据上传活动图

如图 4.4所示，众包平台通过 HTTP 协议调用本系统需求数据上传接口 ReqController，系统获取数据后调用 RequireLogic 方法处理需求数据，计算需求文档及待测软件哈希值。服务层将处理后的数据组装成 Fabric 智能合约中定义的数据模型，调用 DocService 发起联盟链交易。交易需要经过节点共识，共识成功则将数据添加进联盟链。

图 4.5给出了需求数据采集关键代码。需求数据上传时，调用任务服务类中的 updateTaskStatus 方法更新任务状态，传入参数为任务 Id、任务名称及任务状态，0 代表需求提交阶段。然后将封装好的数据模型 requestCommit 发送给 requireService 需求数据上传接口。需求审核数据上传逻辑与需求数据上传逻辑一致，任务状态更新为 1 代表需求数据审核阶段。uploadRequire 方法具体通过 idService 的 genId 方法生成唯一 id 作为业务数据的键，genId 调用 Java 的 UUID 工具生成唯一通识码。服务层将业务数据键及值发送给 Dao 层处理。createDoc 为业务数据上传联盟链的统一接口，调用 docService 中的 save 方法，docService 为联盟链服务中封装对智能合约的调用，具体实现在 4.5 节中阐述。

```
//需求数据上传
public RequestCommit uploadRequire(RequestCommit requestCommit) {
taskService.updateTaskStatus(requestCommit.getTaskId(),requestCommit.getTaskNam
e(),0);
    return this.requireService.uploadRequire(requestCommit);
}
//requireServiceImpl
public RequestCommit uploadRequire (RequestCommit requestCommit) {
    String docId = idService.genId();
    this.docDao.createDoc(docId,requestCommit);
    return requestCommit;
}
public Doc createDoc(String docId, Doc doc) {
    if(docId == null) {
        return null;
    }
    try {
        docService.save(docId, doc);
    } catch (InterruptedException e) {
        e.printStackTrace();
    } //省略异常捕捉代码
    return doc;
}
```

图 4.5: 需求数据采集关键代码

需求方数据溯源活动如图 4.6所示，前端发起需求数据溯源请求，并发送具体需求方 Id 给后端 ReqController 接口。后端接收到请求后通过 RequireLogic 查询需求方的所有需求数据，对于需求方参与的每个众测任务，通过 TaskService 查询任务当前状态，如果任务已完成，通过 FinalRepService 查询平台交付的最终报告数据。数据查询操作由 DocService 中的 findOne 及 query 函数实现，findOne 通过 Fabric 账本数据的键查询对应的所有业务数据，query 函数通过业务数据中某个具体字段查询字段所属数据模型的全部数据。数据查询不更改账本数据，因此不需要节点共识，直接调用智能合约相应方法查询账本即可。

在众测活动进行时，需求方可以通过溯源需求提交数据，确保平台发布的任务满足测试需求。在平台交付最终报告后，需求方可以追溯最终报告来源信息。最终报告数据包含了缺陷报告列表，为保证最终报告来源真实可靠，对于每一个缺陷报告，溯源系统查询缺陷报告的审核数据，获取缺陷报告得分及评审人信息。如果查询不到审核数据，则代表缺陷报告为平台伪造，最终报告真实性存疑；如果缺陷报告得分过低，代表最终报告融合者操作有误，最终报告质量存疑。需求方可以通过缺陷报告审核者确定责任人。

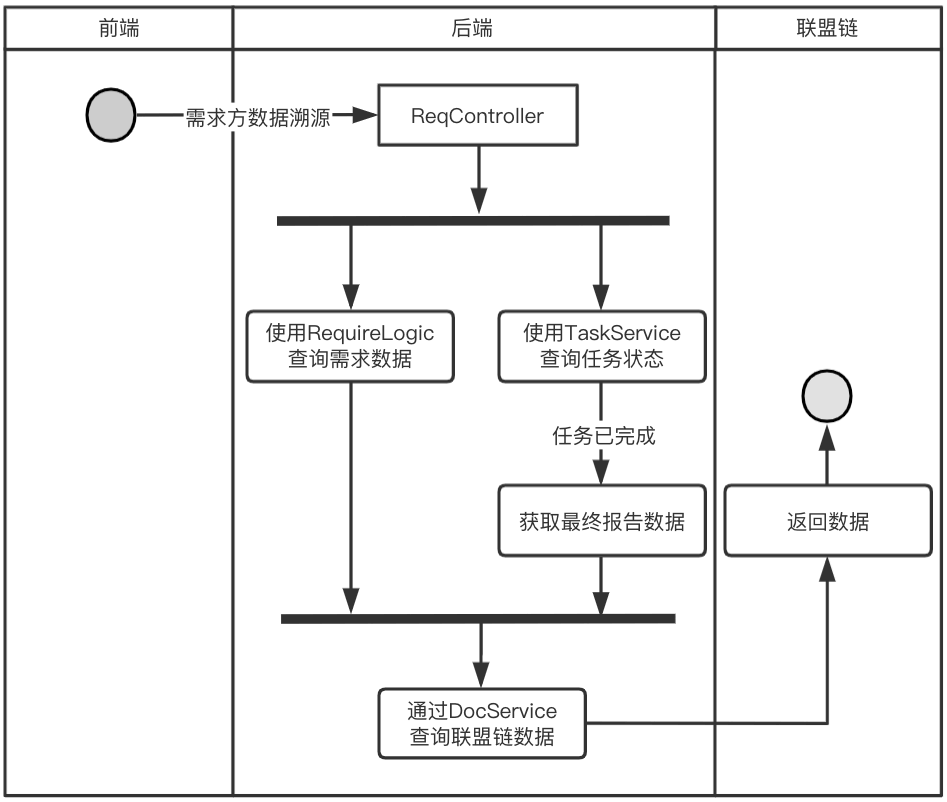


图 4.6: 需求方数据溯源活动图

图 4.7给出了需求数据溯源关键代码。根据具体需求方 Id，首先查找需求提交数据，需求方可能参与多个众测任务，因此得到需求提交列表。接着对于每个需求数据，查询任务当前状态。最后通过各个属性赋值操作，将 RequestCommit 转换为 requestVO 格式返回给前端。

```
public ReqDataCollection getReqDataByUserId(String usrId) {
    ReqDataCollection reqDataCollection = new ReqDataCollection();
    List<RequestVO> requestVOList= new ArrayList<>();
    List<RequestCommit> requestCommitList = this.requireService.getReqCommitByUserId(usrId)
    requestCommitList.forEach(v->{
        String taskStatus = this.taskService.getTaskStatusBytaskID(v.getTaskId());
        //省略赋值代码
        requestVOList.add(requestVO);});
    reqDataCollection.setReqData(requestVOList);
    return reqDataCollection;}
```

图 4.7: 需求数据溯源关键代码

4.2　测试报告服务

4.2.1　测试报告服务的详细设计

测试报告服务主要针对众测工人的数据溯源提供服务，包括测试报告采集与溯源。采集的数据包括众测任务中每个测试工人提交的测试报告，以及平台方对测试报告的审核数据。溯源服务主要提供测试报告提交及审核数据的溯源。测试报告服务核心类图如图 4.8所示。ReportController 类是负责与前端交互的控制器，提供了测试报告上传、报告审核数据上传及报告溯源的接口。ReportLogic 类是具体业务逻辑实现，在数据采集阶段将数据处理为持久化模型，由于平台审核测试报告以报告中的缺陷报告为单位，ReportLogic 类在数据溯源阶段将测试报告与报告包含的缺陷报告审核结果关联发送给控制器。RequireService 类封装 Dao 层的数据读取操作。TestReport 类与 ReportReview 类是测试报告及报告审核结果的数据模型，TestRepCollection 类是报告溯源数据模型。

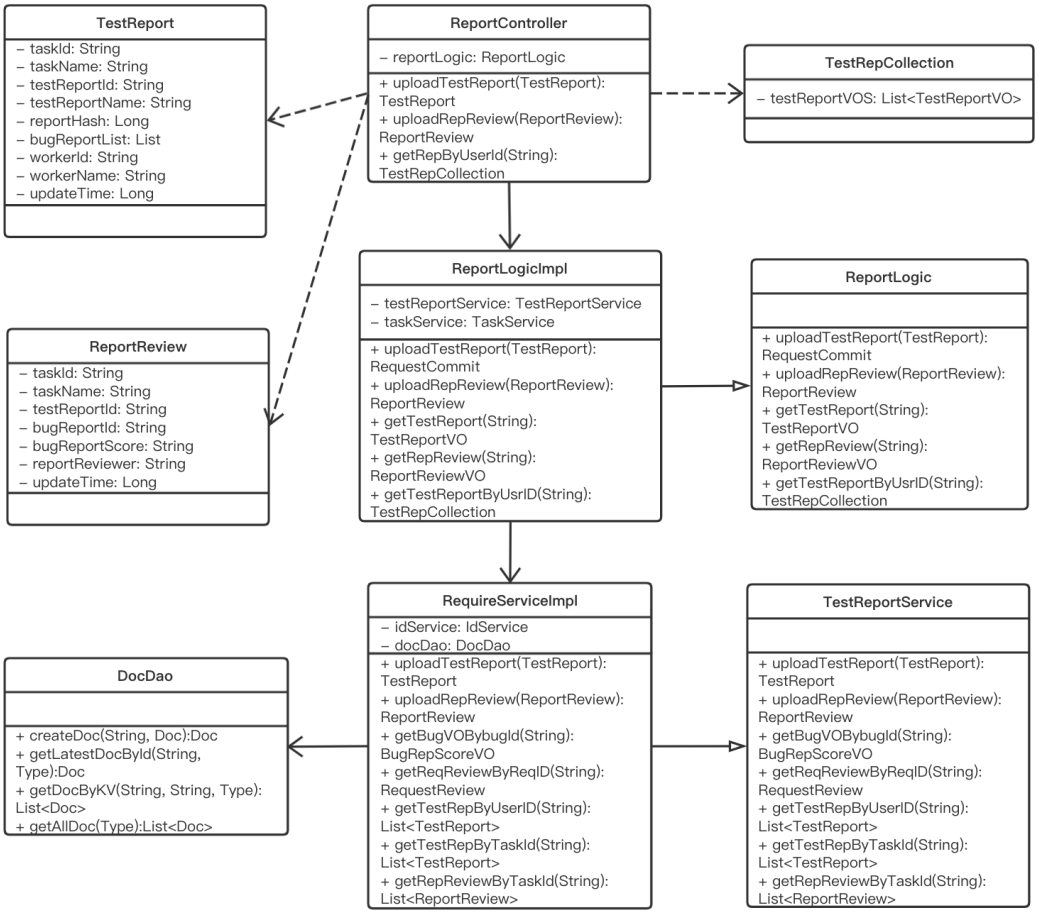


图 4.8: 测试报告服务核心类图

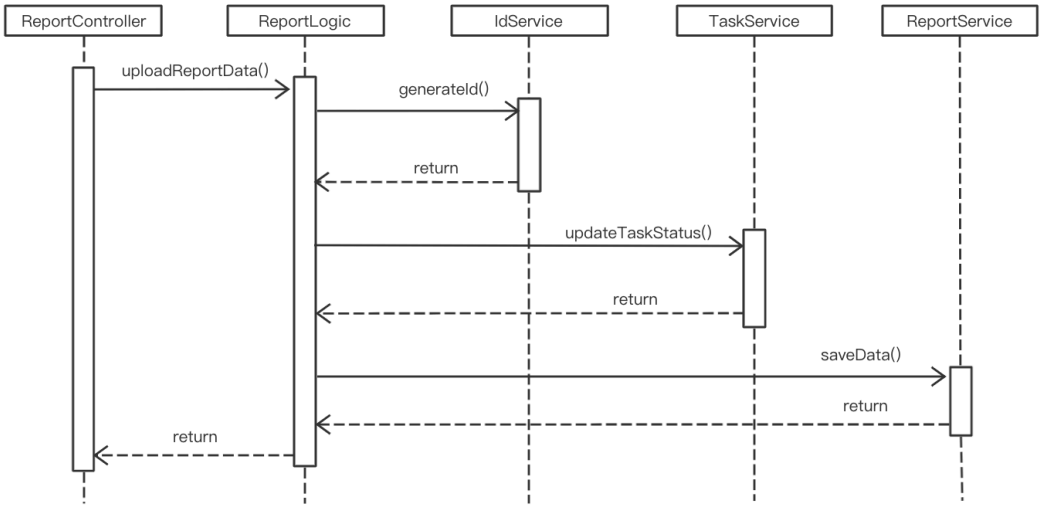


图 4.9: 测试报告采集顺序图

测试报告采集调用顺序如图4.9所示。在接收到众测平台测试报告数据上传要求时，ReportController 调用 ReportLogic 中的测试报告上传接口，ReportLogic 首先会调用 IdService 生成测试报告数据在联盟链中存储的唯一键，接着更新众测任务状态，最后调用 ReportService 存储数据。

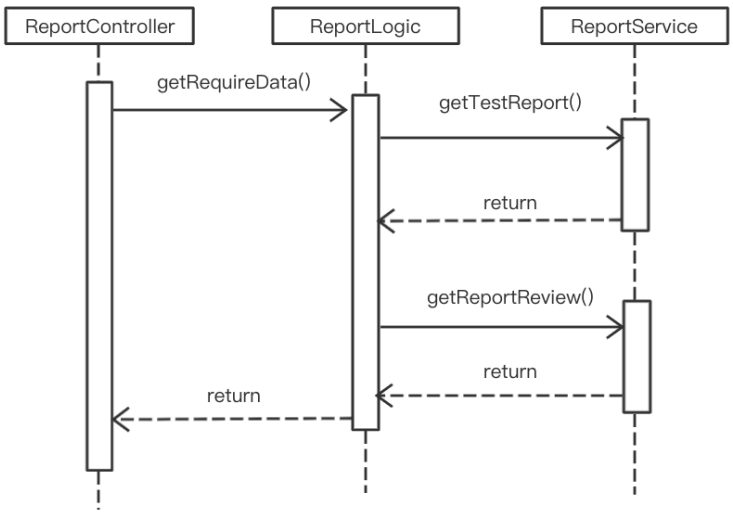


图 4.10: 测试报告溯源顺序图

测试报告溯源调用顺序如图 4.10所示。在接收到众测工人发起的数据溯源请求时，ReportController 调用 ReportLogic 中的测试报告溯源方法，查询测试报告数据及报告审核数据，ReportService 封装底层数据读取过程，并将数据模型发送给 Logic 层。

4.2.2　测试报告服务的实现

```
@ApiOperation(value = "/uploadTestReport", notes = "上传测试报告")
@RequestMapping(value = "/testReport", method = RequestMethod.POST)
public ResponseResult<TestReport> uploadTestReport(@RequestBody TestReport testReport){
    testReport.setType(2);
    TestReport result = reportLogic.uploadTestReport(testReport);
    return new ResponseResult<TestReport>().setData(result);
}
@ApiOperation(value = "/uploadRepReview", notes = "上传缺陷报告审核数据")
@RequestMapping(value = "/reportReview", method = RequestMethod.POST)
public ResponseResult<ReportReview> uploadRepReview(@RequestBody ReportReview
reportReview){
    reportReview.setType(3);
    ReportReview result = reportLogic.uploadRepReview(reportReview);
    return new ResponseResult<ReportReview>().setData(result);
}
@ApiOperation(value = "/getTestReportByUsrId", notes = "根据众测工人 ID 获得测试报告数据")
@RequestMapping(value = "/testReport/{usrId}", method = RequestMethod.GET)
public ResponseResult<TestRepCollection> getRepByUserId(@PathVariable String usrId){
    TestRepCollection result = reportLogic.getTestReportByUsrID(usrId);
    return new ResponseResult<TestRepCollection>().setData(result);
}
```

图 4.11: 测试报告服务 Controller 层关键代码

图 4.11给出了测试报告服务控制层关键代码。众测工人填写完测试报告后，众测平台调用上传测试报告接口/uploadTestReport/testReport，并将测试报告数据封装为 TestReport 数据模型，TestReport 包含了任务 Id、任务名称、测试报告名称、报告 OSS 地址、报告包含的缺陷报告列表、众测工人名称及更新时间。控制层得到数据后，将 Type 属性设置为 2，代表测试报告数据，并调用逻辑层 uploadTestReport 接口，将测试报告存入联盟链。众测平台审核人员对缺陷报告审核时，审核数据通过系统提供的/uploadRepReview/reportReview 接口实时上传，将报告审核数据封装为 ReportReview 模型，模型由缺陷报告 Id，缺陷报告所属的测试报告 Id，缺陷报告得份、报告审核者及更新时间组成。控制层得到数据模型后将 Type 属性设置为 3，代表报告审核数据，并通过 reportLogic 报告逻辑层的 uploadRepReview 接口存入联盟链。在数据溯源流程中，众测工人可以查看所有提交的测试报告溯源数据，包括测试报告来源、报告评审人及得分情况。众测工人登陆溯源系统后，前端调用 getRepByUserId 接口，通过逻辑层的 getTestReportByUsrID 方法获得，以 TestRepCollection 报告集合模型返回给前端展示，TestRepCollection 数据模型为 TestReportVO 组成的 List。

```
//测试报告上传
public TestReport uploadTestReport(TestReport testReport) {
    String taskId = testReport.getTaskId();
    String status = this.taskService.getTaskStatusBytaskID(taskId);
    if(status.equals("-1")||!status.equals("2")){
        this.taskService.updateTaskStatus(taskId,testReport.getTaskName(),2);
    }
    return this.testReportService.uploadTestReport(testReport);
}
//报告审核数据上传
public ReportReview uploadRepReview(ReportReview reportReview) {
    String taskId = reportReview.getTaskId();
    String status = this.taskService.getTaskStatusBytaskID(taskId);
    if(status.equals("-1")||!status.equals("3")){
        this.taskService.updateTaskStatus(taskId,reportReview.getTaskName(),3);
    }
    return this.testReportService.uploadRepReview(reportReview);
}
//获取任务状态
public String getTaskStatusBytaskID(String taskId) {
    TaskState taskState = (TaskState) this.docDao.getLatestDocById(taskId,TaskState.class);
    if(taskState.getTaskState()==null){
        return "-1";
    }
    return taskState.getTaskState().toString();
}
```

图 4.12: 报告数据采集关键代码

图 4.12给出了报告数据采集关键代码。uploadTestReport 是测试报告上传方法，接收 TestReport 数据模型数据。方法首先获取 testReport 中的 taskId，传入 taskService 中的 getTaskStatusBytaskID 方法，获取测试任务状态。考虑到一次众测任务有多个众测工人提交报告，如果众测任务状态不为 2（报告提交阶段），则调用 taskService 中的 updateTaskStatus 方法，将众测任务状态更新为报告提交阶段。最后调用 testReportService 中的 uploadTestReport 方法，将测试报告数据交由服务层处理。uploadRepReview 方法是报告审核数据上传方法，业务逻辑与测试报告上传一致，任务状态更新为 3 代表报告审核阶段。getTaskStatusBytaskId 方法为根据任务 Id 获取任务状态方法。获取任务状态通过调用 docDao 中的 getLatestDocById 方法实现，该方法接收具体键及数据模型类型作为参数，并根据数据类型查询具体数据模型，并返回具体键下的所有业务数据。如果没有任务状态，则返回-1 代表需要更新，否则返回具体状态数值。

```
public TestRepCollection getTestReportByUsrID(String usrId) {
    TestRepCollection result = new TestRepCollection();
    List<TestReportVO> testReportVOS = new ArrayList<>();
    List<TestReport> testReports = this.testReportService.getTestRepByUserID(usrId);
    testReports.forEach(v->{
        TestReportVO testReportVO = new TestReportVO();
        testReportVO.setTaskId(v.getTaskId());
        //省略参数设置代码
        List<String> bugList = v.getBugReportList();
        List<BugRepScoreVO> bugRepScoreVOS = new ArrayList<>();
        bugList.forEach(w->{
        BugRepScoreVO bugRepScoreVO =testReportService.getBugVOBybugId(w);
            if(bugRepScoreVO.getBugID()==null){
                bugRepScoreVO.setBugID(w);
            }
            bugRepScoreVOS.add(bugRepScoreVO);
        });
        testReportVO.setBugReport(bugRepScoreVOS);
        testReportVOS.add(testReportVO);
    });
    result.setTestReportVOS(testReportVOS);
    return result;
}
```

图 4.13: 测试报告溯源关键代码

图 4.22给出了测试报告溯源关键代码。getTestReportByUsrId 方法接收用户 Id 参数，返回用户所有已提交测试报告集合 TestRepCollection。具体地，通过 testReportService 方法中的 getTestRepByUserId 方法，获取测试报告列表 testReports，对于报告列表中的每份测试报告，获取报告包含的缺陷报告列表属性 bugList，对于每个缺陷报告，通过 testReportService 中的 getBugVOBybugId 方法，获取缺陷报告审核数据 bugRepScoreVO，如果当前缺陷报告已审核，则将审核数据添加进 bugRepScoreVOS 中。最后，返回 TestRepCollection 给前端，交由前端渲染页面各个属性。

众测工人可以通过溯源系统追溯测试报告详细信息，包括报告提交时间、提交选手及报告得分情况。测试报告得分是平台奖励众测工人的重要依据，测试报告得分由报告包含的缺陷报告得分累加获得，对于每个缺陷报告，溯源系统追溯缺陷报告得分及评审人数据。当选手对报告得分有异议时，可以通过缺陷报告审核数据查看具体得分来源情况。

4.3　最终报告服务

4.3.1　最终报告服务的详细设计

最终报告服务是针对众测流程中的报告融合环节设计的服务，包括报告融合数据采集及最终报告溯源服务。报告融合数据包括最终报告哈希值及包含的缺陷报告列表。溯源服务主要由需求方及平台方发起，在任务完成时，需求方可以查看平台交付的最终报告来源信息，平台方在追溯众测流程中的报告融合环节时，可以查看审核人员融合的最终报告数据。最终报告服务核心类图如图 4.14所示。FinalRepController 类是对外提供的控制器，定义了上传最终报告数据接口 uoploadFinalReport 以及最终报告数据溯源接口 getFinalReport。FinalReportLogic 类处理具体业务逻辑，在采集最终报告数据时，将平台传递的数据转换为预定义的联盟链账本数据格式，并发送给 Service 层，在数据溯源时，将获得的数据转换为数据溯源模型发送给 Controller 层。FinalRepService 类为上层封装数据存储接口，DocDao 类是调用联盟链客户端的接口，根据参数访问账本数据。FinalReportVO 类是与平台交互的数据模型，FinalRepVO 类是与前端交互的数据模型，ReportMix 是与联盟链交互的数据模型。

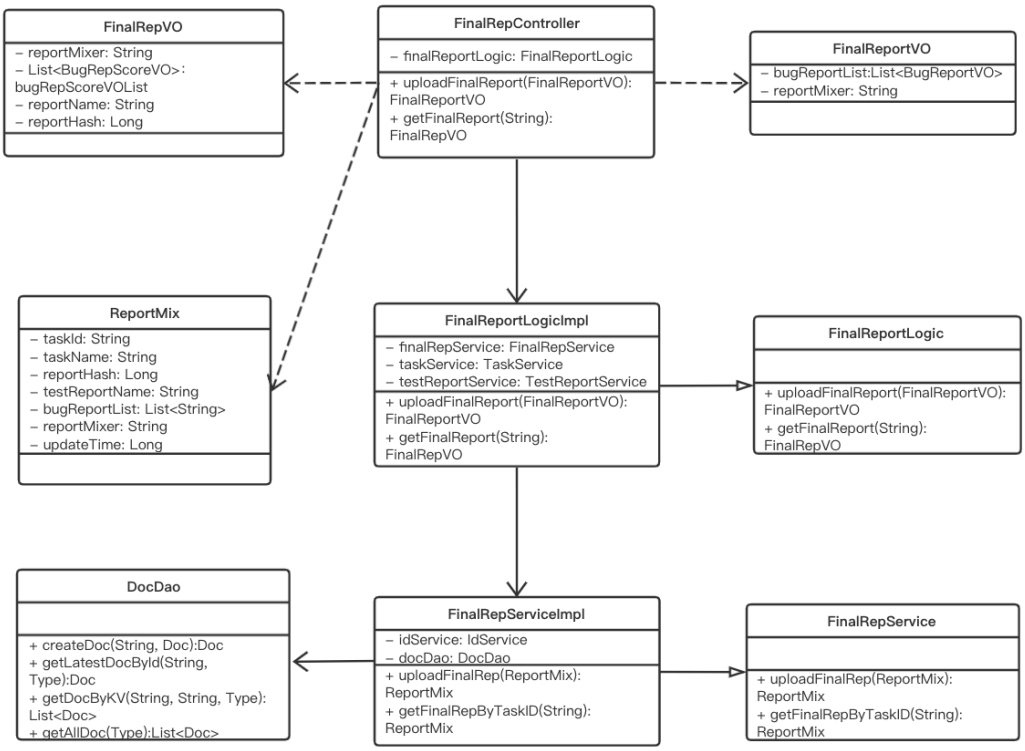


图 4.14: 最终报告服务核心类图

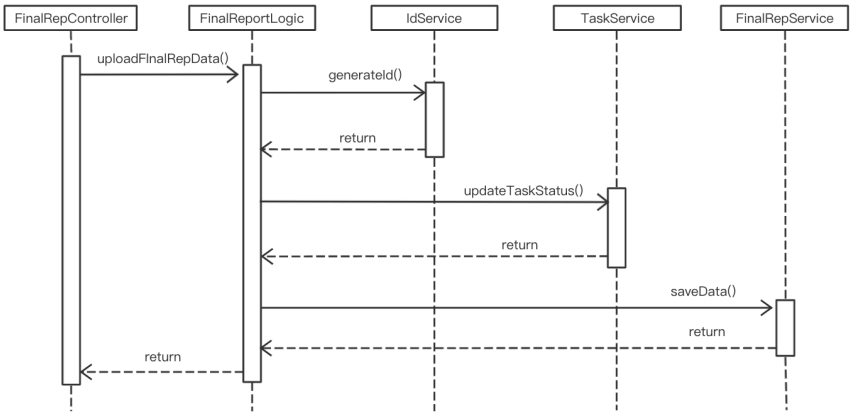


图 4.15: 最终报告数据采集顺序图

最终报告数据采集调用顺序如图 4.15所示。FinalRepController 接收到平台发送的最终报告数据后，调用 FinalReportLogic 处理数据，转换为联盟链账本数据模型，并将 IdService 生成的唯一 Id 作为数据模型的键发送给服务层，同时，调用 TaskService 更新任务状态。FinalRepService 将最终报告数据发送给联盟链。

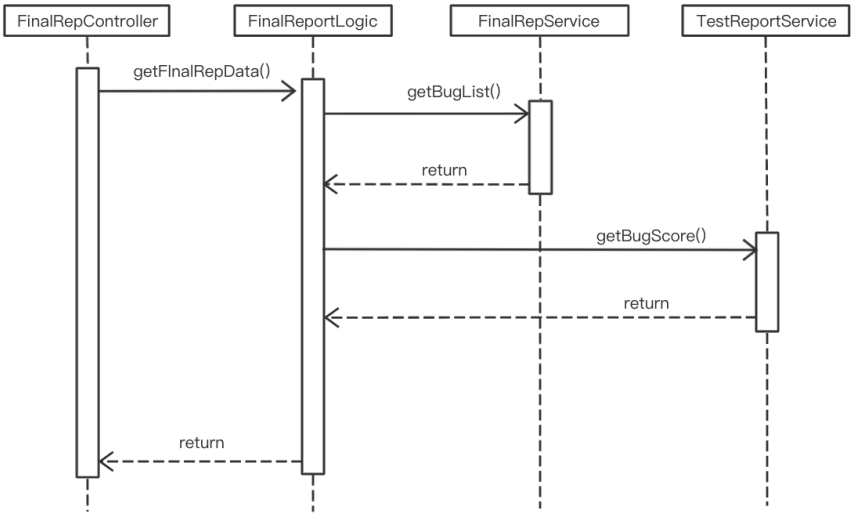


图 4.16: 最终报告数据溯源顺序图

最终报告数据溯源调用顺序如图 4.16所示。FinalRepController 接收到最终报告溯源命令后，调用 FinalReportLogic 查询对应任务的最终报告数据，数据包含了缺陷报告列表，根据缺陷报告 Id，调用 TestReportService 查询每个缺陷报告的得分，Logic 层将数据封装为最终报告数据溯源模型后发送给 Controller 层。

4.3.2　最终报告服务的实现

图 4.17给出了最终报告数据采集关键代码。最终报告数据上传时首先调用 taskService 的 updateTaskStatus 方法，将众测任务状态更新为 4，代表报告融合阶段，考虑到众测任务中报告融合阶段只有一次，因此不做当前任务状态检查。然后将控制层传入 FinalReportVO 模型转换为联盟链智能合约中定义的 ReportMix 模型，对于最终报告包含的缺陷报告列表，将缺陷报告 Id 存入 bugReportVOList 实体类，并添加进 ReportMix。最后调用最终报告服务中的 uploadFinalRep 方将数据发送给 Service 层。

```
//最终报告数据上传
public FinalReportVO uploadFinalReport(FinalReportVO finalReportVO) {
    this.taskService.updateTaskStatus(finalReportVO.getTaskId(),finalReportVO.getTaskName(),4);
    ReportMix reportMix = new ReportMix();
    reportMix.setTaskId(finalReportVO.getTaskId());
    reportMix.setTaskName(finalReportVO.getTaskName());
    reportMix.setReportMixer(finalReportVO.getReportMixer());
    reportMix.setUpdateTime(finalReportVO.getUpdateTime());
    List<BugReportVO> bugReportVOList = finalReportVO.getBugReportList();
    List<String> bugRep = new ArrayList<>();
    bugReportVOList.forEach(v->{
        bugRep.add(v.getBugReportID());
    });
    reportMix.setBugReportList(bugRep);
    this.finalRepService.uploadFinalRep(reportMix);
    return finalReportVO;
}
```

图 4.17: 最终报告数据采集关键代码

图 4.18给出了最终报告数据溯源关键代码。getFinalReport 为 FinalRepLogic 中处理最终报告数据溯源接口。需求方进入溯源系统后，点击具体参与的众测任务，当任务状态为已完成时，系统展示最终报告来源信息。溯源接口获得具体任务 Id 后，调用 finalRepService 中的 getFinalRepByTaskID 方法，获取报告融合数据实体 reportMix。bugList 为最终报告包含的缺陷报告列表，对于列表中每一个缺陷报告，系统调用 testReportService 中的 getBugVOBybugId 方法，获取报告审核数据，bugRepScoreVO 封装了缺陷报告 Id，缺陷报告名称，报告审核者及报告得分。最后，将组装的前端交互实体 FinalRepVO 返回给前端。前端获得最终报告数据后进行页面渲染，展示具体数据。

```
//最终报告数据溯源
public FinalRepVO getFinalReport(String taskId) {
    FinalRepVO result = new FinalRepVO();
    List<BugRepScoreVO> bugRepScoreVOS = new ArrayList<>();
    ReportMix reportMix = this.finalRepService.getFinalRepByTaskID(taskId);
    List<String> bugList = reportMix.getBugReportList();
    bugList.forEach(v->{
        BugRepScoreVO bugRepScoreVO = testReportService.getBugVOBybugId(v);
        bugRepScoreVOS.add(bugRepScoreVO);
    });
    result.setReportMixer(reportMix.getReportMixer());
    result.setBugRepScoreVOList(bugRepScoreVOS);
    result.setReportName("最终交付报告");
    result.setReportHash(reportMix.getReportHash());
    return result;
}
```

图 4.18: 最终报告数据溯源关键代码

4.4　众测任务服务

4.4.1　众测任务服务的详细设计

众测任务服务主要针对众测平台提供整个众测流程的数据溯源服务，包括需求提交阶段、需求审核阶段、报告提交阶段、报告审核阶段、报告融合阶段。考虑到需求提交与审核阶段数据较少，将两阶段数据合并展示，具体展示需求方提交的需求文档及待测软件，审核人员对需求数据的审核结果。报告提交阶段系统追溯已提交的测试报告数据，包括报告来源，报告包含的缺陷报告详细信息。报告审核阶段系统追溯已审核的缺陷报告数据，包括报告得分及审核者信息。报告融合阶段追溯最终报告包含的缺陷报告审核信息。

图 4.19是众测任务服务核心类图。TaskController 类是对外提供的控制器，根据前端要求返回不同阶段众测数据。TaskLogic 类是平台方数据溯源业务逻辑实现，依赖 RequireService、TaskService、TestReportService、FinalRepService 服务类实现众测数据溯源。其中 getAllTask 方法获取所有已上链的众测任务信息，getTaskDetail 方法获取具体众测任务状态信息，getReqByTaskId 根据任务 Id 获取需求数据，getRepMixByTaskId 获取报告融合数据，getRepReviewByTaskId 获取报告审核数据。TaskService 类提供众测任务状态更新、状态查询、任务详情获取等功能。TaskCollection 类、ReportCollection 类、RepReCollection 类、TestRepColleciton 类是 Controller 层与前端交互的数据模型，TaskState 是联盟链中存储的众测任务状态数据模型。

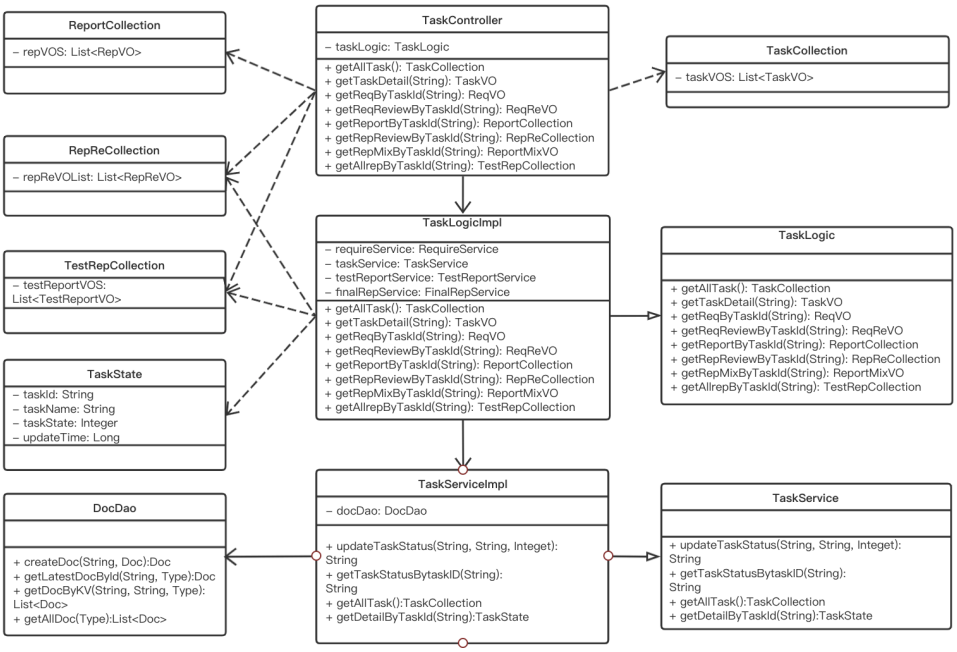


图 4.19: 众测任务服务核心类图

众测任务溯源调用顺序如图 4.20所示。TaskController 调用 TaskLogic 的众测数据溯源接口获取相关众测数据，TaskLogic 调用 RequireService 获取需求及需求审核数据，调用 TestReportService 获取测试报告及报告审核数据，调用 FinalRepService 获取最终报告融合数据。获取的数据后封装为前端交互数据模型发送给 Controller 层。

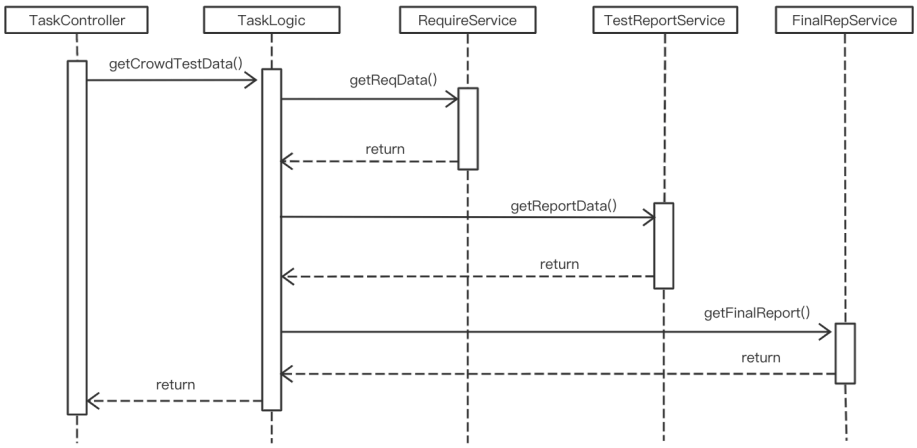


图 4.20: 众测任务溯源顺序图

```
public TaskVO getDetailByTaskId(String taskId) {
    TaskVO result = new TaskVO();
    TaskState taskState = this.taskService.getDetailByTaskId(taskId);
    if(taskState.getTaskState() == 0){
        result.setTaskStatus("需求提交阶段");
    }else if ...//省略状态转换代码
    Long releaseTime = this.requireService.getReqReviewByTaskId(taskId).getUpdateTime();
    result.setUpdateTime(releaseTime);
    return result;
}
```

图 4.21: 获取任务详细信息关键代码

4.4.2　众测任务服务的实现

图 4.21给出了获取具体任务详细信息相关实现代码。getDetailByTaskId 方法得到 controller 层传入的众测任务 Id 后，调用 taskService 服务类中的 getDetailByTaskId 方法，获得具体任务的状态，并将数值型状态转换为易于审查的字符串类型，其中，0 代表需求提交阶段，1 代表需求审核阶段，2 代表报告提交阶段，3 代表报告审核阶段，4 代表报告融合阶段。releaseTime 选取需求审查更新时间作为任务更新时间，代表任务正在进行中。

```
public TestRepCollection getAllrepByTaskId(String taskId) {
    TestRepCollection result = new TestRepCollection();
    List<TestReportVO> testReportVOS = new ArrayList<>();
    List<TestReport> testReports = this.testReportService.getTestRepByTaskId(taskId);
    testReports.forEach(v->{
        TestReportVO testReportVO = new TestReportVO();
        testReportVO.setTaskId(v.getTaskId());
        //省略参数赋值代码
        List<String> bugList = v.getBugReportList();
        List<BugRepScoreVO> bugRepScoreVOS = new ArrayList<>();
        bugList.forEach(w->{
            BugRepScoreVO bugRepScoreVO = testReportService.getBugVOBybugId(w);
            if(bugRepScoreVO.getBugID()==null){
                bugRepScoreVO.setBugID(w);
            }
            bugRepScoreVOS.add(bugRepScoreVO);
        });
        testReportVO.setBugReport(bugRepScoreVOS);
        testReportVOS.add(testReportVO);
    });
    result.setTestReportVOS(testReportVOS);
    return result;
}
```

图 4.22: 平台方测试报告溯源关键代码

图 4.22给出了平台方测试报告溯源关键代码。逻辑层根据平台方选择的具体众测任务，通过 taskId 查询此任务下所有已提交的测试报告得到列表，并将报告列表中的每一份测试报告转换为 TestReportVO 格式，测试报告中包含缺陷报告列表，通过 testReportService 中的 getBugVOBybugId 方法获取每个缺陷报告得分信息，如果查询的缺陷报告没有审核信息，则只添加缺陷报告 Id，对于有得分信息的缺陷报告将实体类 bugRepScoreVO 添加进缺陷报告审核结果列表，最后封装为 testReportVOS，返回给 Controller 层。

4.5　联盟链服务

4.5.1　联盟链服务的详细设计

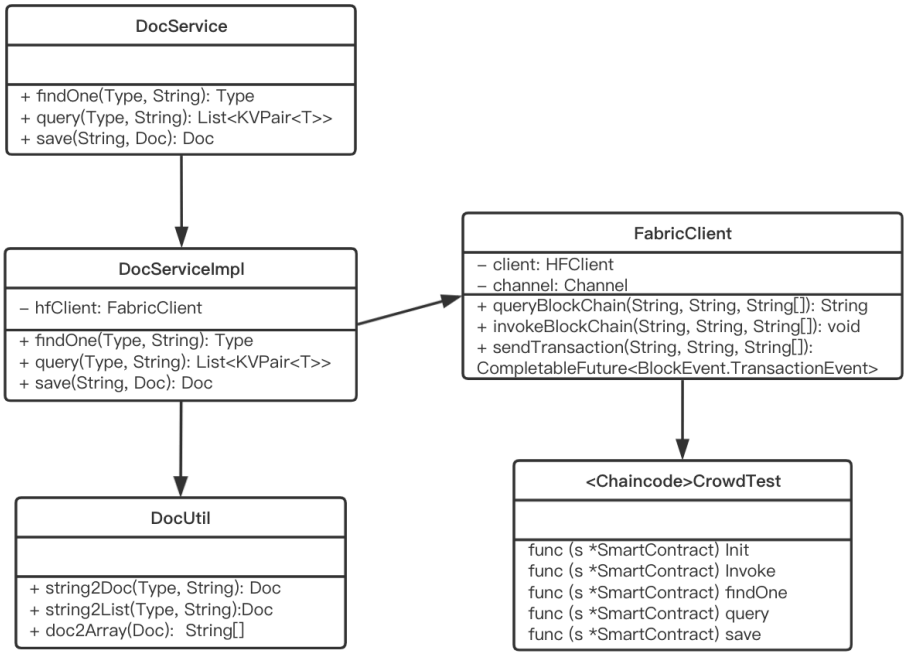


图 4.23: 账本读取模块核心类图

本项目在 Fabric-Java-SDK 基础上，实现了联盟链服务，提供通道创建、智能合约 ChainCode 部署等功能，并封装智能合约读取账本数据操作，为其他服务提供屏蔽底层复杂操作的接口。图 4.23是智能合约账本数据读取模块核心类图，DocServiceImpl 实现 DocService 接口对外提供账本数据读写功能。DocServiceImpl 依赖 FabricClient 与智能合约进行交互，queryBlockChain 是对账本数据的读取接口，可以传入智能合约具体方法，invokeBlockchain 方法通过 sendTransaction 方法向联盟链发送交易请求存入数据。DocUtil 提供账本数据转码映射功能，CrowdTest 是 go 语言编写的智能合约。

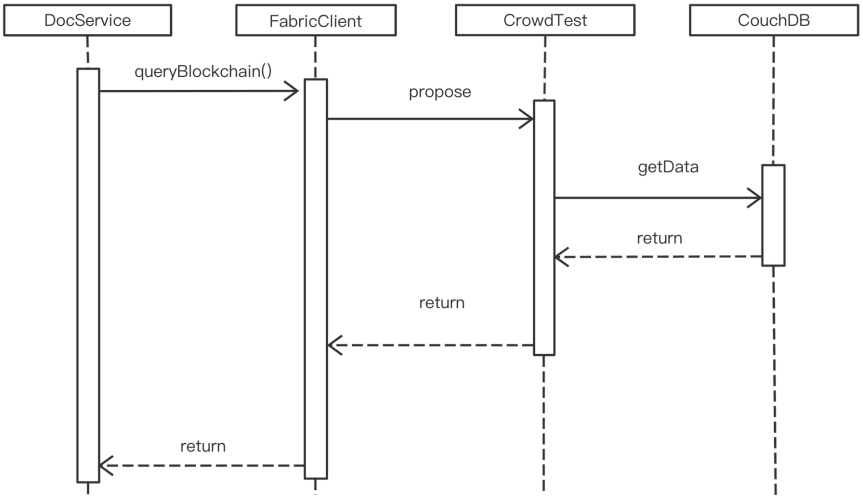


图 4.24: 账本数据读取顺序图

Fabric 提供 query 与 invoke 两种对账本数据的操作。query 操作指读取账本数据，调用顺序如图 4.24所示。DocService 获取业务层读取账本数据请求后，调用 FabricClient 中的查询方法，FabricClient 发起查询提议，查询操作不更新账本数据，不需要节点的共识。CrowdTest 根据查询请求传入的参数查询映射账本数据的数据库 CouchDB，并返回查询结果。

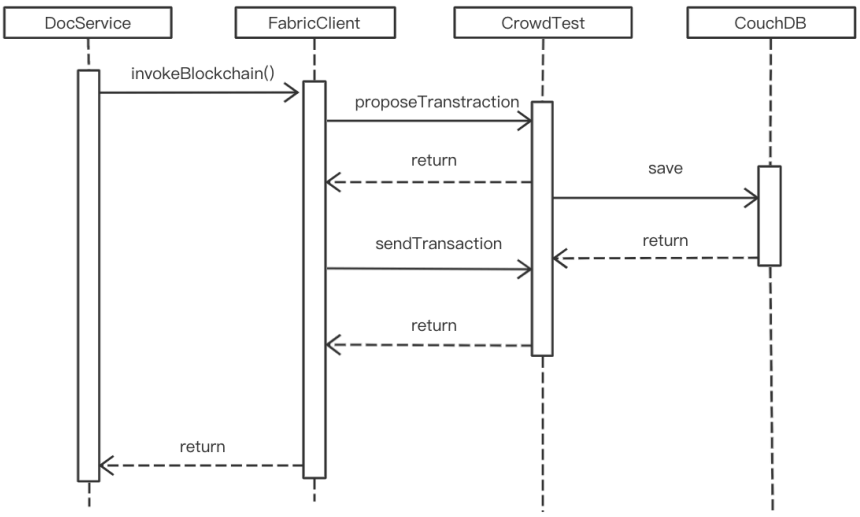


图 4.25: 账本数据写入顺序图

账本数据写入调用如图 4.25所示。DocService 发起查询区块链请求后调用 FabricClient 中的 invokeBlockchain 方法，FabricClient 在查询账本数据前先发起交易请求并广播给联盟链网络中所有节点，若交易请求中的参数非法或节点共

识失败，则返回相应失败停止交易，若共识成功则将交易请求发送给智能合约，智能合约根据传入的参数执行 save 方法，将数据写入联盟链。

4.5.2　联盟链服务的实现

```
public HFClient getHfClient(@Value("${fabric.ca.url}") String caUrl,
                            @Value("${fabric.secret}")String secret) throws Exception{
    CAClient caClient = new CAClient(caUrl, null);
    UserContext adminUserContext = new UserContext();
    adminUserContext = caClient.enrollAdminUser("admin", secret);
    CryptoSuite cryptoSuite = CryptoSuite.Factory.getCryptoSuite();
    HFClient hfClient = HFClient.createNewInstance();
    hfClient.setCryptoSuite(cryptoSuite);
    hfClient.setUserContext(adminUserContext);
    return hfClient;
}
```

图 4.26: HFClient 创建关键代码

图 4.26给出了 HFClient 创建关键代码。HFClient 是 Fabric-Java-SDK 提供的 SDK 使用客户端，在调用 SDK 其他接口时需要注册客户端。HFClient 创建需要添加经过授权的用户。CAClient 是 Fabric CA 客户端实例，对联盟链网络的访问首先要经过 CA 的认证。CAClient 根据配置文件传入的 CA 地址生成 CA 客户端实例，adminUserContext 是客户端用户实例，通过 CAClient 注册为可以访问节点的用户。CryptoSuite 是密钥解析工具，HFClient 通过 createNewInstance() 创建实例，并设置 CryptoSuite 及 adminUserContext 完成创建。

```
public Channel getChannel(HFClient hfClient, //省略配置文件参数), String channelName) {
    //省略异常检查
    Peer peer = hfClient.newPeer( peerName, peerUrl);
    Orderer orderer = hfClient.newOrderer(ordererName, ordererUrl);
    Channel channel = hfClient.newChannel(channelName);
    channel.addPeer(peer);
    channel.addOrderer(orderer);
    channel.initialize();
    return channel;
}
```

图 4.27: Channel 创建核心代码

创建完 HFClient 后，需要创建联盟链网络通道，并将参与交易的节点添加进通道，通道是 Fabric 为了保证交易数据隐私的子网。如图 4.27所示，Peer 是通

过 HFClient 创建维护账本的网络节点，Orderer 是通过 HFClient 创建的提供共识服务的网络节点。Channel 是通道实例，在添加 Peer 和 Orderer 后执行 initialize() 操作完成初始化。

```
public String queryBlockChain(String chaincode, String function, String[] args) throws
ProposalException, InvalidArgumentException {
    QueryByChaincodeRequest qpr = client.newQueryProposalRequest();
    ChaincodeID cid = ChaincodeID.newBuilder().setName(chaincode).build();
    //省略 qpr 组装代码
    Collection<ProposalResponse> res = channel.queryByChaincode(qpr);
    for (ProposalResponse pres : res) {
        //省略异常处理代码
        String stringResponse = new String(pres.getChaincodeActionResponsePayload());
        logger.info("query result:"+stringResponse);
        return stringResponse;
    }
    return null;
}
```

图 4.28: 账本数据读取关键代码

创建完 HFClient 和 Channel 后可以实现对联盟链账本数据的读取。图 4.28给出了账本数据读取关键代码。client 是 HFClient 实例，调用 newQueryProposalRequest 方法生成 query 查询实体，并添加参数传入的智能合约、具体方法及查询参数。查询实体交由通道的 queryByChaincode 方法发送给联盟链网络，返回查询结果。如果查询结果不合法则打印出具体交易 Id 及结果信息，方便问题排查。

```
public CompletableFuture<BlockEvent.TransactionEvent> sendTransaction(…)
        throws InvalidArgumentException, ProposalException {
    TransactionProposalRequest tpr = client.newTransactionProposalRequest();
    ChaincodeID cid = ChaincodeID.newBuilder().setName(chaincode).build();
  //省略 tpr 组装代码
    Collection<ProposalResponse> responses = channel.sendTransactionProposal(tpr);
    List<ProposalResponse> invalid =
    responses.stream().filter(ProposalResponse::isInvalid).collect(Collectors.toList());
    if (!invalid.isEmpty()) {
      invalid.forEach(response -> logger.error("tx "+response.getTransactionID()+"
      proposal invalid "+response.getMessage()));
    throw new RuntimeException("invalid response(s) found");
}
    return channel.sendTransaction(responses);
}
```

图 4.29: 账本数据写入关键代码

账本数据写入由于新增数据，写入过程作为联盟链网络中的交易由各个节点共识验证。如图 4.29所示，sendTrasaction 参数与 queryBlockChain 参数与一致，HFClient 生成 TransactionProsalRequest 交易请求，并添加方法传入的各个参数。组装好的交易请求由 channel 的 sendTransaction 方法发送给通道，通道返回各个节点的响应 ProposalResponse。对于每一个响应检查有效性，如果全部有效则将所有响应作为参数发送给 sendTransaction，写入账本数据，如果有无效响应则抛出发现无效响应的异常。

```
func (s *SmartContract) Init(APIstub shim.ChaincodeStubInterface) sc.Response {
    return shim.Success(nil)
}

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 == "query" {
        return s.query(APIstub, args)
    }
    return shim.Error("Invalid Smart Contract function name.")
}
```

图 4.30: 智能合约 Invoke 关键代码

对联盟链数据的访问由智能合约完成，本文基于 Fabric 智能合约 Chaincode 实现规则编写名为 CrowdTest 的智能合约。如图 4.30所示，智能合约需要实现 Init 与 Invoke 两个方法。Init 方法在 Chaincode 初始化及更新时调用，完成必要的初始化逻辑。Invoke 方法接收智能合约方法名作为参数，并根据方法名调用方法具体实现。

智能合约 save 函数实现联盟链数据存储功能。根据 Fabric 底层 CouchDB 数据存储规则，业务数据作为键值对中的值存储，本文定义多个数据模型对应众测活动不同阶段数据，数据模型中 type 字段作为类型标识符。如图 4.31所示，业务数据第一个参数为对应业务数据类型（0 对应 RequestCommit 模型，1 对应 RequestReview 模型，2 对应 TestReport 模型，3 对应 ReportReview 模型，4 对应 ReportMix 模型，5 对应 TaskState 模型）。根据不同类型数据，将参数中的字符组装进数据模型，并调用 Marshall 方法转换为 Byte 实体 docAsBytes。PutState 方法将 docAsBytes 存储于交易账本，完成业务数据联盟链存储。

```
func (s *SmartContract) save(APIstub shim.ChaincodeStubInterface, args []string)
sc.Response {
    doctype := args[1]
    var docAsBytes []byte
    if doctype == "0" {
        doc := RequestCommit{Type:doctype, TaskId: args[2] ...}
        docAsBytes, _ = json.Marshal(doc)
    } //省略相同存储逻辑
    APIstub.PutState(args[0], docAsBytes)
    return shim.Success(nil)
}
```

图 4.31: 智能合约 save 关键代码

业务层通过 DocService 访问联盟链，图 4.32给出了 DocService 实现关键代码。DocService 参数 Type 对应扩展 Doc 类的具体业务数据类型。findOne 函数为键值插叙，通过账本数据的唯一键 (Key) 查找对应业务数据，返回数据通过 DocUtil 的 string2Doc 方法转换为 Java 对象。query 函数为值数据查询，通过业务数据中的某个字段查询字段所属数据模型所有数据，多条业务数据组成的字符串转换为 List 返回。save 函数提供业务数据联盟链存储功能，调用 HFClient 的 InvokeBlockChain 方法将数据存入交易账本，交易失败时抛出异常。

```
public class DocServiceImpl implements DocService {
    public <T> T findOne(Type type, String key) throws ... {
        String result = hfClient.queryBlockChain(ChainCode.crowdTest.getName(),
                                Function.findOne.getName(),new String[]{key});
        return DocUtil.string2Doc(type,result);
    }
    public <T> List<KVPair<T>> query(Type type, String queryString) throws ...{
        String result = hfClient.queryBlockChain(ChainCode.crowdTest.getName(),
                    Function.query.getName(), new String[]{queryString});
        return DocUtil.string2List(type,result);
    }
    public void save(String key, Doc doc) throws ... {
        String[] param = DocUtil.doc2Array(doc);
        param[0] = key;
        hfClient.invokeBlockChain(ChainCode.crowdTest.getName(),
                    Function.save.getName(),param);
    }
```

图 4.32: DocService 实现关键代码

4.6　联盟链配置与部署

```
networks:
  custom:
services:
  couchdb:
    container_name: couchdb
    image: hyperledger/fabric-couchdb
    environment:
      - COUCHDB_USER=
      - COUCHDB_PASSWORD=
    ports:
      - "5984:5984"
    networks:
      - custom
  ca.org1.example.com:
    image: hyperledger/fabric-ca:1.4.1
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca-org1
      - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-ca-server/…
      - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server/…
    ports:
      - "7054:7054"
    command: sh -c 'fabric-ca-server start -b admin:adminpw -d'
    volumes:
      -../network_resources/crypto-config/peerOrganizations/org1.example.com/ca/:…
        container_name: ca_peerOrg1
    networks:
      - custom
```

图 4.33: CouchDB 及 Fabric CA 模块部署关键代码

本项目提供联盟链网络部署脚本，包含最小化可运行 Fabric 组件，各个组件依赖 docker 容器运行。图 4.33给出了 CouchDB 及 Fabric-CA 组件部署脚本 docker-compose 关键代码。脚本定义 Fabric 世界状态数据库为 CouchDB，配置 docker 镜像 hyperledger/fabric-couchdb，并配置数据库用户名密码，将容器中端口映射到服务器端口 5984 暴露给其他组件。ca.org1.example.com 为身份认证组件 Fabric-ca 相关部署，镜像文件采用最新的 1.4.1 版本，并配置 CA 证书及密钥文件地址，7054 端口为 Fabric-CA 服务器默认端口。命令 fabric-ca-server start 启动 Fabric-CA 服务器，为各个节点提供身份验证功能，admin 账户为 CA 缺省管理员账户，联盟链服务通过管理员账户注册新用户，并创建用户可以访问的通道。服务器初始化后，会生成包含 keyStore 的 CA 服务器私钥和证书文件等。volumes 将 CA docker 镜像中的配置及证书密钥文件挂载到挂载到本地。

```
orderer.example.com:
  container_name: orderer.example.com
  image: hyperledger/fabric-orderer:1.4.1
  environment:
    - ORDERER_GENERAL_LOGLEVEL=debug
    - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
    - ORDERER_GENERAL_GENESISMETHOD=file
    - ORDERER_GENERAL_GENESISFILE=/etc/hyperledger/configtx/genesis.block
    - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
    - ORDERER_GENERAL_LOCALMSPDIR=/etc/hyperledger/msp/orderer/msp
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/orderer
  command: orderer
  networks:
    - custom
  ports:
    - 7050:7050
  volumes:
      - ../network_resources/config/:/etc/hyperledger/configtx
      - ../network_resources/cryptoconfig..
      - ../network_resources/crypto-config/peerOrganizations..
```

图 4.34: Orderer 节点部署关键代码

Fabric 联盟链中的交易经历三个阶段：交易提案、交易排序及交易验证。Orderer 节点在交易排序阶段负责收集所有网络中发生的交易并进行排序，已排序的交易被打包成区块并分发给连接到 Orderer 节点的对等节点，经过所有节点验证成功的交易区块将更新到账本数据，否则标记为失效。图 4.34给出了 Orderer 节点部署关键代码。docker 镜像文件采用 fabric-orderer1.4.1 版本。环境设置时指定成员服务提供者 MSP 组件，并确定工作目录。orderer 命令启动节点。volumes 将 Orderer 节点配置文件挂载到本地。

对等节点 Peer 是组成 Fabric 联盟链网络的基本要素，智能合约与账本数据由 Peer 托管。本系统设置 3 个对等节点分别对应需求方节点、众测工人方节点以及众测平台方节点，众测数据经过节点验证后存于账本上，账本数据无法删除只能追加，保证了众测溯源数据的安全性。如图 4.35所示，Peer 节点配置通过 docker-compose 文件执行，docker 镜像采用 fabric-peer1.4.1 版本。Peer 环境配置智能合约日志级别为 DEBUG，方便调用智能合约出错时问题排查，账本数据库设置为 CouchDB。peer node start 命令启动对等节点。端口 7051 为 GRPC 通信端口，7053 为 Peer 事件监听端口。depends 命令定义了对等节点依赖于上文描述的 Orderer 排序节点。

```
peer0.org1.example.com:
  container_name: peer0.org1.example.com
  image: hyperledger/fabric-peer:1.4.1
  environment:
    - CORE_PEER_ID=peer0.org1.example.com
    - CORE_CHAINCODE_LOGGING_LEVEL=DEBUG
    - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb:5984
    - …
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric
  command: peer node start
  ports:
    - 7051:7051
    - 7053:7053
  networks:
    - custom
  volumes:
      - /var/run/:/host/var/run/ …
  depends_on:
    - orderer.example.com
    - couchdb
```

图 4.35: Peer 节点部署关键代码

本项目提供 shell 脚本配置智能合约，Fabric 智能合约需要部署到所有对等节点上，负责对账本数据的操作。溯源系统智能合约使用 go 语言编写，命名为 CrowdTest。如图 4.36所示，peer chaincode install 命令根据智能合约名称及位置部署合约 crowdTest。peer chaincode instantiate 执行智能合约初始化操作，配置 Orderer 节点端口、通道 channel 以及对等节点。

```
installChaincode() {
PEER=$1; ORG=$2; setGlobals $PEER $ORG
peer chaincode install
-n crowdTest -l ${LANGUAGE} -p ${CC_SRC_PATH} >&log.txt
peer chaincode instantiate
-o orderer.example.com:7050
-C $CHANNEL_NAME -n crowdTest -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 4.36: 智能合约配置脚本关键代码

4.7　系统实例展示

本文实现的众测数据溯源系统采用联盟链技术，将溯源数据存储于联盟链账本上，由联盟链网络中各个节点共识验证，账本数据读写通过智能合约实现，确保数据真实可靠。在用户追溯数据来源时，系统将众测数据在众测流程中的转换过程详细展示，明确数据来源。系统服务于众测各参与方，包括需求方、众测工人以及众测平台管理人员，下面展示系统主要运行界面。

图 4.37: 需求方用户主页

图 4.37展示了需求方用户主页。主页上方展示了区块链当前信息，包括节点数、区块高度、交易笔数以及链码数，用户可以点击详情跳转到 Fabric Explorer 展示更详细的区块链数据。主页中间部分分为两块，左边展示了用户的个人信息，右边展示了已提交需求的审核状态以及众测任务状态。主页下方展示了需求方参与的众测任务，包括任务名称、提交的需求文档、待测软件，当前状态。状态为”已完成“时，需求方可以点击“最终报告”查看最终报告来源信息。“溯源”按钮链接到众测任务数据溯源界面。用户点击“数据验证”按钮可以查看众测平台保存的需求数据与联盟链上数据哈希值对比结果，哈希值一致说明需求数据未被篡改。用户点击“区块链信息”可以查看需求数据在联盟上的存储情况，包括数据哈希、数据区块信息等。

图 4.38展示了需求方最终报告溯源界面。需求方可以查看最终报告融合时间及融合者。系统通过展示最终报告包含的缺陷报告来源信息实现最终报告溯源。对于每一份缺陷报告，系统展示了提交缺陷报告的众测工人、报告得分、报

图 4.38: 最终报告溯源界面

告评审人。当需求方发现缺陷报告质量不高时，可以查看报告评审人信息，进行后续追责。

图 4.39: 众测工人用户主页

图 4.39展示了众测工人用户主页。众测工人可以通过测试报告审核分析查看已审核及未审核报告数量，通过缺陷报告采纳分析查看被融合到最终报告中的缺陷报告数量。主页下方展示了用户提交的测试报告，包括众测任务名称、测试报告名称、报告当前状态、提交时间等。用户可以点击“数据验证”查看测试报告是否被篡改。“区块链信息”展示了测试报告区块链存储信息。

图 4.40: 众测工人数据溯源界面

图 4.40展示了众测工人数据溯源界面。众测工人可以查看需求提交阶段、报告提交阶段、报告审核阶段的数据来源。需求提交阶段展示了当前众测任务的需求数据来源，报告提交阶段展示了测试报告提交者及提交时间，报告审核阶段展示了每个缺陷报告的评审信息。众测工人可以通过查看报告来源验证报告是否为本人提交，当对审核结果有异议时通过审核结果来源进行后续验证。

图 4.41: 众测平台方主页

图 4.41展示了众测平台方用户主页。任务状态分析展示了任务完成数量，需求状态分析展示了已审核需求数量，报告状态分析展示了已审核报告数量。页面下方展示了所有众测任务信息，用户点击“溯源”按钮可以查看对应众测任务各个阶段的数据来源信息。

4.8　本章小结

本章主要描述了众测数据溯源系统主要服务的详细设计与实现，包括需求数据服务、测试报告服务、最终报告服务、众测任务服务、联盟链服务。详细设计通过类图和顺序图描述服务的静态模型和动态模型，实现部分通过项目关键代码展示。接着，本章介绍了联盟链账本数据映射数据库 CouchDB、验证组件 Fabric CA 等组件的配置与部署方案。最后，本章展示了系统主要运行界面，展示系统为不同众测参与方提供的数据溯源功能。

第五章　溯源系统的测试与案例分析

依据系统需求分析与实现描述，本章进行系统测试及案例分析。首先描述测试案例，并介绍测试环境及评价指标。然后进行测试设计，包括功能测试、性能测试及安全测试。最后对测试结果进行分析并进行案例分析。

5.1　案例描述

众测溯源系统实时接入众测数据，并提供数据追溯功能。为验证系统能够在真实场景下完成业务逻辑，并提供可靠服务，本文选取慕测平台 2020 年 4 月众测月度赛数据作为测试数据进行系统测试。

表 5.1: 月度赛众测数据集

数据名称	数据来源	数据类型	数据数量
咕咚翻译 APP	需求方	软件 APK	1
咕咚翻译需求文档	需求方	文字、图片	1
测试报告	众测工人	文字、图片	64
缺陷报告	众测工人	文字、图片	408
审核结果	平台方	数值	408
最终测试报告	平台方	文字、图片	1

表 5.1给出了月度赛众测数据。测试软件为咕咚翻译 APP，属于软件 APK；需求文档为咕咚翻译需求文档，通过文字及图片描述；测试报告总共有 64 份，通过文字及图片描述；测试报告包含的缺陷报告为 408 例，包括文字描述与图片描述；平台对所有缺陷报告进行审核，审核结果为数值类型；最终报告由平台方融合缺陷报告获得，仅有 1 份。

图 5.1给出了众测数据示例，包括需求方在众测平台提交需求文档、众测工人在众测平台填写的缺陷报告、平台对于缺陷报告的审核结果以及平台交付给需求方的最终测试报告。子图（a）为需求文档示例，需求文档包含了待测软件的描述信息及功能测试需求。子图（b）为缺陷报告示例，众测工人填写的数据包括缺陷复现程度、严重性、分类及描述，用户可以上传具体应用截图描述缺陷。子图（c）为众测评审人员对缺陷报告的审核结果示例，根据其他众测工人对缺陷报告的点赞、点踩数，以及缺陷报告描述情况给出综合得分。子图（d）为最终测试报告示例，最终测试报告由缺陷报告组成，展示了缺陷报告类别、严重程度、可复现程度以及相关描述。

(a) 需求文档　(b) 缺陷报告

(c) 缺陷报告审核结果　(d) 最终测试报告

图 5.1: 众测数据示例

5.2　测试环境

表 5.2给出了系统部署各个模块的硬件环境，系统需要部署前端 Web 平台、众测数据服务以及联盟链节点。其中，前端 Web 平台与众测数据服务模块部署在同一台云服务器上，方便业务层接口调试，服务器型号为 ecs.c1.large，操作系统为 Ubuntu16.04；联盟链节点部署在两台云服务器上，分别为排序节点与对等节点，节点之间通过 GRPC 通信。

表 5.2: 硬件环境

模块	硬件环境
前端 Web 平台	云服务器，型号 ecs.c1.large，Ubuntu16.04，1 台
众测数据服务	云服务器，型号 ecs.c1.large，Ubuntu16.04，1 台
联盟链模块	云服务器，型号 ecs.c1.large，Ubuntu16.04，2 台

表 5.3给出了系统各个模块的软件环境。前端 Web 平台基于 Vue 3.0 构建，通过 Chrome 浏览器运行，前后端之间通过 Nginx 1.10.2 实现跨域访问和反向代

理，Nginx 同时支持负载均衡功能。众测数据服务基于 Springboot 2.0.02 框架，通过 JDK 1.8.0 实现，系统采用 Maven 3.3.9 打包部署业务层服务。联盟链节点基于 Hyperledger Fabric 1.4.1 构建，联盟链各个组件通过 Docker 18.06.01 容器运行。

表 5.3: 软件环境

模块	软件环境
前端 Web 平台	Chrome 浏览器、Vue 3.0、Nginx 1.10.2
众测数据服务	JDK 1.8.0、Maven 3.3.9、Springboot 2.0.2
联盟链节点	Hyperledger Fabric 1.4.1，Docker 18.06.01

5.3　评价指标

表 5.4: 系统测试评价指标

指标项	指标描述
响应时间	用户感受系统为其服务所耗费的时间
吞吐量	系统在单位时间能处理的数据数量
资源使用率	系统提供服务时 CPU 占有率、内存使用率等
并发用户数	某一物理时刻同时向系统提交请求的用户数量
事物成功率	系统在单位时间内完成一定数量事物中成功的事物比率

表 5.4给出了系统测试评价指标，包括响应时间、吞吐量、资源使用率、并发用户数及事物成功率。响应时间能够体现系统在不同负载下的应答时间，系统用户数量越多响应时间越大；吞吐量体现了单位时间内系统能处理的用户请求数量，通过吞吐量可以找到系统的瓶颈问题 [49]；资源使用率是指系统占用服务器 CPU 大小，系统内存使用率等；并发用户数指同时向系统发送请求的用户数量；事物成功率指测试过程中单位时间成功测试事物的数量，事物成功率高代表系统稳定性较好。

5.4　测试设计

5.4.1　功能测试设计

功能测试又被称为黑盒测试，通过设计功能测试用例验证系统各项功能。本节设计测试用例模拟众测平台发送众测数据，验证联盟链账本数据写入功能。众测数据追溯测试从用户角度出发，模拟用户在系统中的正常使用流程，对需求方、工人方、平台方数据溯源进行功能测试。

表 5.5: 众测数据联盟链写入测试用例

测试 ID	TC1
测试名称	众测数据联盟链写入
测试功能	众测平台能将数据实时发送给溯源系统并进行联盟链存储
测试步骤	1. 通过 SawggerUI 模拟众测平台调用溯源系统数据采集接口； 2. 溯源系统处理接收到的众测数据 3. 溯源系统发起联盟链数据写入交易提案； 4. 联盟链对交易进行共识并将数据写入账本。
预期结果	1. 数据采集接口能正常访问； 2. 溯源系统接收到众测数据； 3. 联盟链返回交易验证结果； 4. 联盟链账本数据更新。

表 5.5展示了众测数据联盟链写入测试用例。用例模拟众测平台调用溯源系统数据采集接口，溯源系统收到数据后通过联盟链服务发起交易提案，联盟链将经过共识的交易数据更新到账本。在预期情况下，系统采集接口能正常访问接收到众测数据，联盟链及时响应交易请求，账本数据得到更新。

表 5.6: 需求方需求数据溯源测试用例

测试 ID	TC2
测试名称	需求方需求数据溯源
测试功能	需求方可查看已提交的需求数据及需求审核数据
测试步骤	1. 需求方登录系统； 2. 查看需求数据是否按状态展示； 3. 点击”已完成“按钮，查看是否筛选出已完成任务； 4. 点击”数据验证”按钮，查看是否弹出数据验证结果； 5. 点击“区块链信息”按钮，查看是否弹出区块链数据。
预期结果	1. 需求方登录成功； 2. 显示需求数据及状态； 3. 显示已完成任务； 4. 弹出数据验证结果； 5. 弹出区块链数据。

表 5.6展示了需求方需求数据溯源测试用例。用例模拟需求方登录系统，查看需求数据并点击“数据验证”及“区块链信息”按钮。在预期情况下，系统展示需求方已提交的需求数据及对应众测任务状态，并能按要求展示数据验证结果及数据区块链存储信息。

表 5.7: 需求方最终报告溯源测试用例

测试 ID	TC3
测试名称	需求方最终报告溯源
测试功能	需求方可追溯平台交付的最终报告来源
测试步骤	1. 点击“最终报告”按钮，查看是否跳转到最终报告溯源页面； 2. 点击“详情”按钮，查看是否列出缺陷报告得分信息； 3. 点击“数据验证”按钮，查看是否弹出最终报告验证结果； 4. 点击“区块链信息”按钮，查看是否弹出区块链数据。
预期结果	1. 平台跳转到最终报告溯源页面； 2. 显示缺陷报告列表，包含审核结果； 3. 弹出数据验证结果； 4. 弹出区块链数据。

表 5.7展示了需求方最终报告溯源测试用例。用例模拟需求方选择最终报告，并查看最终报告包含的缺陷报告。在预期情况下，系统展示最终报告溯源界面，并能显示报告包含的缺陷报告来源信息，用户可以验证数据并查看数据区块链存储信息。

表 5.8: 众测工人测试报告溯源测试用例

测试 ID	TC4
测试名称	众测工人测试报告溯源
测试功能	众测工人可追溯测试报告来源及审核结果
测试步骤	1. 众测工人登录系统； 2. 查看测试报告数据是否按状态展示； 3. 点击已审核测试报告，查看是否展示缺陷报告审核结果； 4. 点击“数据验证”按钮，查看是否弹出测试报告验证结果； 5. 点击“区块链信息”按钮，查看是否弹出区块链数据。
预期结果	1. 众测工人登录成功； 2. 显示工人已提交测试报告列表； 3. 显示缺陷报告列表，包含审核结果； 4. 弹出数据验证结果； 5. 弹出区块链数据。

表 5.8展示了众测工人测试报告溯源测试用例。用例模拟众测工人登录系统，查看测试报告数据，查询数据验证结果及区块链信息。在预期情况下，系统展示已提交测试报告列表，并给出测试报告审核状态，对于已审核报告，能够展示缺陷报告得分来源数据。

表 5.9: 平台方众测任务溯源测试用例

测试 ID	TC5
测试名称	平台方众测任务溯源
测试功能	平台方可追溯众测任务在各个阶段的数据来源
测试步骤	1. 平台方登录系统； 2. 查看众测任务状态； 3. 点击某个众测任务，查看是否跳转到众测任务溯源界面； 4. 点击报告“全部”按钮，查看是否跳转到测试报告溯源页面； 5. 点击审核“全部”按钮，查看是否跳转到缺陷报告审核溯源页面； 6. 点击“最终报告”按钮，查看是否跳转到最终报告溯源页面。
预期结果	1. 平台方登录成功； 2. 显示平台方已发布的众测任务列表； 3. 显示众测任务溯源界面； 4. 显示众测任务所有已提交测试报告页面； 5. 显示已审核缺陷报告页面； 6. 显示最终报告溯源页面。

表 5.9展示了平台方众测任务溯源测试用例。用例模拟众测平台管理人员登录系统，查看众测任务各个阶段数据来源。在预期情况下，系统展示需求提交阶段、需求审核阶段、测试报告提交阶段、缺陷报告审核阶段、最终报告融合阶段五个阶段数据。在测试报告提交阶段，点击“全部”按钮，系统跳转到测试报告溯源页面，展示所有已提交的测试报告，包括包含的缺陷报告。在缺陷报告审核阶段，点击“全部”按钮，系统跳转测试报告审核溯源页面，展示所有已审核缺陷报告得分数据。在最终报告融合阶段，点击“来源查询”按钮，系统跳转到最终报告溯源页面。

5.4.2　性能测试设计

本文性能测试包括业务层接口性能测试及联盟链性能测试。业务层接口通过 JMeter[1]性能测试工具测试。联盟链采用开源的 Hyperledger Caliper[2]区块链系性能测试基准框架完成测试。

1）业务层接口性能测试设计

JMeter 支持用户创建线程访问接口地址，并查看系统接口响应时间。本文首先在 JMeter 中创建线程组，线程组中的线程可以看作虚拟用户，测试执行期间不会发生改变。接着在线程组中添加 HTTP GET 请求，HTTP 地址为系统业

[1]https://jmeter.apache.org/

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

务层服务器接口地址。然后在线程组中设置 40 个用户，每个用户在两秒内同时对服务器发送请求，并循环请求两次。最后添加用户监听测试结果的聚合报告。表 5.10给出了测试的业务层接口。

表 5.10: 业务层接口

接口名称	接口作用
查看提交信息	根据需求方 ID 获取需求方提交的需求数据
查看审查信息	根据需求 ID 获取平台对需求数据的审查结果
获取测试报告	根据任务 ID 获取众测工人提交的测试报告
查询最终报告	根据任务 ID 查询众测任务的最终报告
获取测试任务	获取所有众测任务
查询报告融合	根据最终报告 ID 获取报告融合数据
查询提交序列	根据任务 ID 获取此任务下的所有测试报告
查询审核序列	根据任务 ID 获取此任务下的所有报告审核数据

2）联盟链性能测试设计

表 5.11: 智能合约方法

方法名称	方法作用
save	将数据存储于联盟链上
findOne	根据账本数据 ID 查询对应数据
query	根据关键字查询包含关键字的所有账本数据

Caliper 是区块链性能基准测试框架，支持多个版本的联盟链，包括 Hyperledger Fabric。用户可以通过 Caliper 设置吞吐速率执行联盟链中的智能合约，检测交易成功率、延迟和吞吐量，并查看系统资源占用情况。如表 5.11所示，本文对智能合约三个方法 save、findOne、query 进行压测。为了测试不同交易频率下联盟链的延迟及吞吐量，并发请求数设置为 50tps、100tps、150tps、200tps、250tps，通过 5 轮测试查看联盟链性能。

5.4.3　安全测试设计

溯源系统数据采集与查询通过智能合约实现，如果智能合约存在漏洞则无法保证溯源过程的真实性。智能合约为根据业务需求实现的 Fabric Chaincode，本文通过 Chaincode Scanner[3]工具对智能合约进行安全性分析，检查智能合约漏

[3]https://chaincode.chainsecurity.com/

洞。Chaincode Scanner 根据用户指定的 Chaincode URL 查找具体代码并分析，给出安全分析结果。

表 5.12: 智能合约漏洞

漏洞名称	安全隐患
未检查的输入参数	输入参数可能存在空值情况，影响方法正常运行
写操作后读取	对相同变量进行写操作之后读取，可能读取旧值
账本数据幻读	将幻读结果操作账本，可能导致账本数据异常
全局变量使用	对账本数据操作依赖于全局变量可能导致账本数据异常
并发操作	智能合约中使用并发可能导致异常

表 5.12给出了 Chaincode Scanner 可以检测的智能合约漏洞。其中，未检查的输入参数、并发操作漏洞可能导致智能合约方法无法正常运行。写操作后读取、账本数据幻读及全局变量使用可能导致账本数据异常。

5.5　测试结果与分析

5.5.1　功能测试结果与分析

表 5.13: 功能测试用例执行结果

测试用例 ID	用例描述 ID	测试结果
TC1	UC1	通过
TC2	UC2	通过
TC3	UC3	通过
TC4	UC4	通过
TC5	UC5	通过

表 5.13给出了系统功能测试执行结果。测试人员严格按照测试用例描述步骤，依次执行所有测试用例，执行结果全部通过。系统完成了需求分析里提炼的系统功能需求，被测系统行为符合产品功能设计。

5.5.2　性能测试结果与分析

1）业务层接口性能测试结果

JMeter 测试结果如表 5.14所示。由表可知，系统接口正确处理了 200 次请求，200 次请求平均响应时间为 484 毫秒，最小响应时间为 397 毫秒，最大响应时间为 583 毫秒，系统没有出现异常，平均吞吐量接近 123/sec。因此，系统接口功能正常，在高并发情况下响应时间较短。

表 5.14: JMeter 测试结果

Label	样本	平均值	中位数	最小值	最大值	异常%	吞吐量
查看提交信息	200	501	417	491	542	0.00%	16.5/sec
查看审查信息	200	494	503	463	524	0.00%	17.2/sec
获取测试报告	200	544	517	501	583	0.00%	13.5/sec
查询最终报告	200	412	421	397	443	0.00%	16.1/sec
获取测试任务	200	442	431	401	487	0.00%	18.2/sec
报告融合信息	200	461	454	413	502	0.00%	15.4/sec
报告提交序列	200	505	507	467	531	0.00%	11.2/sec
报告审核序列	200	513	523	483	541	0.00%	15.2/sec
TOTAL	1600	484	471	397	583	0.00%	123.3/sec

2）联盟链性能测试结果

表 5.15: 智能合约性能测试结果

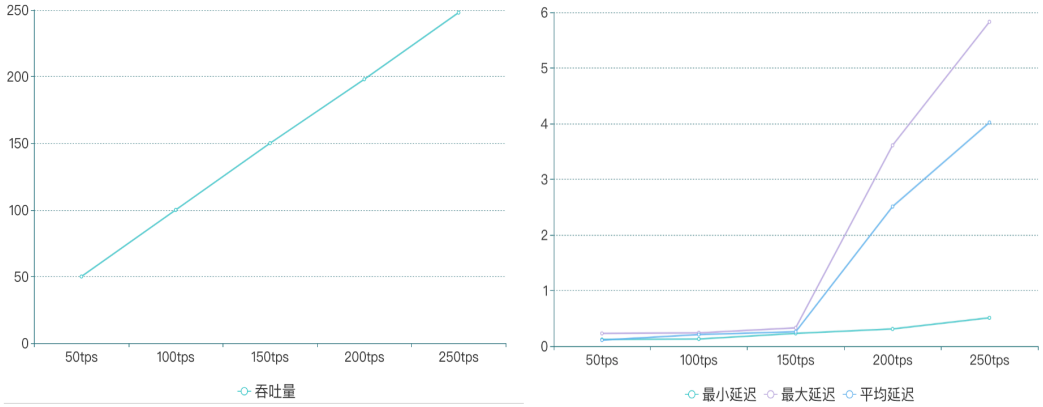
名称	成功数	失败数	发送速率	最大延迟	最小延迟	平均延迟	吞吐量
save	1000	0	50tps	0.40s	0.04s	0.17s	50tps
findOne	1000	0	50tps	0.24s	0.04s	0.09s	50tps
query	1000	0	50tps	0.28s	0.02s	0.11s	50tps

智能合约性能测试结果如表 5.15所示，在每秒 50 笔交易情况下，智能合约各个方法全部执行成功。save 方法写入账本数据需要节点共识验证，平均延迟为 0.17 秒，findOne 和 query 方法查询账本不用共识，延迟约为 0.1 秒。智能合约最大延迟 0.4 秒，平均延迟 0.11 秒，吞吐量达到 50tps。

表 5.16: 联盟链资源占用率

名称	内存（最大）	内存（平均）	CPU（最大）	CPU（平均）
peer0.org1.example.com	140.7MB	138.2MB	48.33%	29.45%
peer0.org2.example.com	120.1MB	117.4MB	44.32%	29.34%
orderer.example.com	28.6MB	27.4MB	21.79%	14.23%

表 5.16给出了联盟链系统资源占用情况。对等节点需要验证并存储账本数据，因此资源占用率相对较高。对等节点 Org1 和 Org2 分别占用 140.7MB 和 120.1MB 内存，CPU 平均占用率分别为 29.45% 和 29.34%。排序节点仅负责联盟链网络中的交易排序，资源占用较少。排序节点 Orderer 内存平均使用 27.4MB，CPU 平均占用 14.23%。



(a) 吞吐量　　(b) 延迟

图 5.2: 高并发智能合约性能测试结果

图 5.2给出了不断增加并发量后智能合约性能测试结果，包括吞吐量及延迟表现。当并发数由 50tps 逐渐增加至 250tps 时，各方法平均延迟由 0.11 秒增加至 4.02 秒，最大延迟由 0.23 秒增加至 5.83 秒，系统吞吐量由 50tps 增加至 248tps。可见系统吞吐量达到高并发要求，系统延迟在 50tps、100tps、150tps 条件下较低，由于服务器性能原因，在 200tps 及 250tps 延迟较高。

5.5.3　安全测试结果与分析

表 5.17: 智能合约安全测试结果

漏洞类型	漏洞数量
未检查的输入参数	0
写操作后读取	0
账本数据幻读	0
全局变量使用	0
并发操作	0

智能合约安全测试结果如表 5.17所示。Chaincode Scanner 扫描智能合约每一行代码，结果显示未检查的输入参数、写操作后读取、账本数据幻读、全局变量使用、并发操作漏洞数量均为 0，智能合约未发现安全漏洞，安全性较高。

5.6　案例分析

慕测平台 2020 年 4 月份月度赛众测时间为 4 月 25 日下午 1 点至下午 5 点，历经三个小时。如图 5.6所示，系统成功接入 64 个众测工人提交的 64 份测试报告，包括 408 份缺陷报告，并进行联盟链存储。

图 5.3: 溯源系统数据统计图

图 5.4给出了溯源系统溯源详情界面。系统溯源功能运行正常，众测任务处于报告提交阶段，平台方可以查看需求提交阶段、需求审核阶段及报告提交阶段的众测数据来源。在报告提交阶段，平台方可以查看已提交测试报告来源信息，包括报告名称、提交者以及提交时间。

图 5.4: 溯源系统溯源详情图

图 5.5给出了众测比赛中系统页面响应时间，X 轴为时间戳，Y 轴为响应时间。系统页面平均响应时间约为 500 毫秒，在 15:40 页面响应时间达到 1482 毫秒，在 16:10 达到 1372 毫秒，考虑到该时间为众测工人大量提交测试报告，系统负载增加所致。系统整体响应时间在 2 秒以内，用户体验良好。

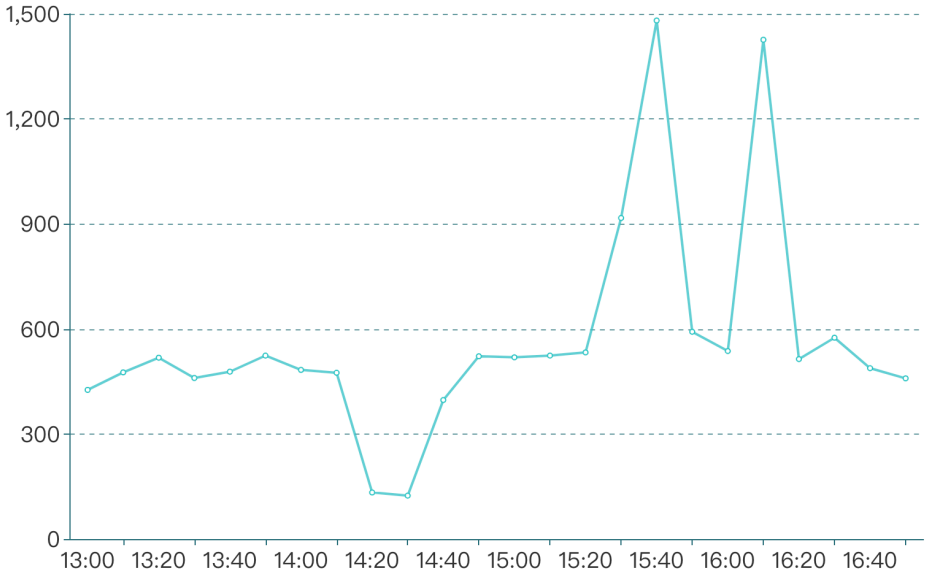


图 5.5: 页面响应时间性能统计

图 5.6给出了众测比赛中系统 CPU 占用情况，X 轴为时间戳，Y 轴为占用百分比。系统 CPU 占用平均约为 3%，在 15:40 分占用为 6.53%，在 16:10 分达到 22.77%，之后恢复到平均水平。和响应时间增加一致，该时间段工人提交大量报告，系统负载增加导致 CPU 占用增加，但仍为可用范围。

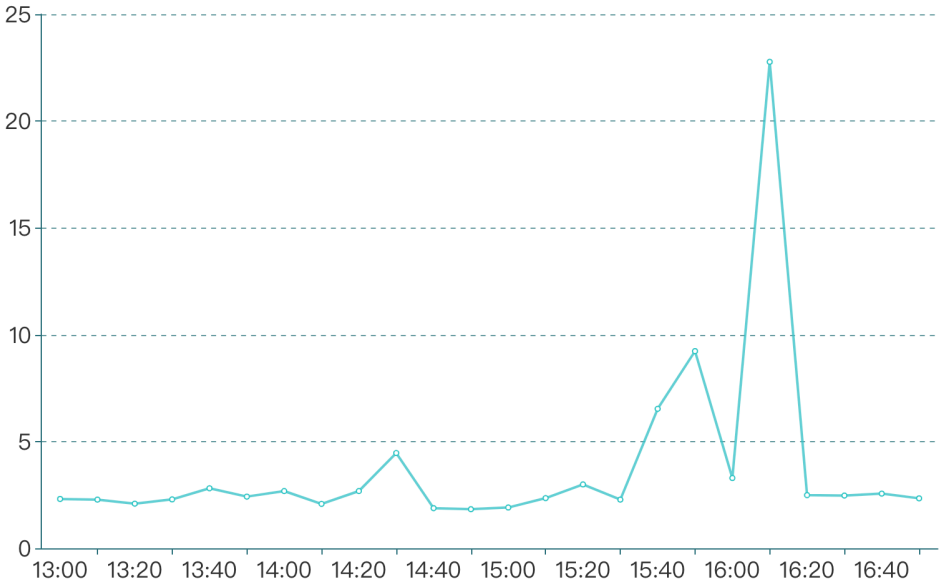


图 5.6: CPU 占用性能统计

5.7　本章小结

本章对众测数据溯源系统进行测试与案例分析。首先进行测试案例分析，说明系统测试数据来源并给出测试数据描述；其次介绍测试环境，描述测试节点、容器等配置；接着描述系统测试性能指标；然后根据系统用例描述设计功能测试用例，给出 JMeter、Caliper、Scanner 性能测试工具使用方案；最后通过图表方式给出系统测试结果，测试结果表明系统满足业务需求，在高并发情况下仍能提供稳定的服务，同时智能合约安全性较高，系统溯源服务安全可信。

第六章　总结与展望

6.1　总结

随着众包测试的发展，众测用户对众包测试数据的真实性提出了更高要求。众测数据溯源可以追溯数据来源信息，验证数据真实性。然而当前数据溯源多采用中心化架构，溯源数据存储于传统数据库，存在数据丢失及被恶意篡改现象，如何保证溯源数据安全是一个关键问题。联盟链去中心化、不可篡改的特性为解决众测数据溯源信任及安全问题提供了新的思路。本文设计并实现了基于联盟链的众测数据溯源系统，将溯源数据存储于联盟链上，溯源数据由联盟链网络参与者共同维护，不可篡改，数据读写过程由经过验证的智能合约实现，确保众测数据溯源真实可靠。

基于联盟链的众测数据溯源系统主要包括众测数据采集、众测数据联盟链存储以及众测数据的溯源。根据慕测平台众包测试流程，本系统在需求提交、需求审核、报告提交、报告审核及报告融合阶段实时采集众测数据，溯源系统获取数据后向联盟链发起交易请求，各个节点验证数据后同意交易提案，调用智能合约将数据作为交易账本存储于联盟链上，账本由各个节点共同维护，只能追加，无法篡改。数据溯源阶段，本系统针对需求方、众测工人、众测平台提供数据来源追溯服务。需求方可以追溯需求数据审核数据来源，在任务完成时，追溯最终报告来源信息。众测工人可以追溯测试报告评审得分来源。众测平台管理员可以通过溯源系统查看众测各个阶段数据的来源信息。

技术实现层面，本文采用业界主流框架进行开发。前端采用 React 框架，服务层采用 Springboot 框架，联盟链存储层采用 Hyperledger Fabric 框架。为保证系统性能和可扩展性，采用 Docker 部署系统各个服务，Kubernetes 集群管理容器的调度、扩展和负载均。本文最后对溯源系统进行功能测试、性能测试与安全测试。测试结果表明，系统能够在有效时间内完成用户请求响应，联盟链网络在 50tps 的交易负载情况下仍能保持良好的可用性，智能合约安全性较高，可以提供可信数据溯源服务。

6.2　展望

当前系统还存在很多不足及可扩展的空间，未来可以从三个方面对系统进行完善和改进：

第一，追溯更多众测数据。当前数据溯源追溯了众测各参与方的核心数据，未来可以追溯众测流程中采用的技术数据，如审核测试报告得分采用的加权方式，让众测流程更加透明公正。

第二，优化溯源数据展示方式。当前溯源系统按众测流程各个阶段展示数据来源，数据之间的关系表示不够明显，未来考虑优化溯源数据展示方式，加强各阶段数据关系的展示。

第三，添加更多数据溯源节点。当前溯源系统包括需求方、众测工人、众测平台三类节点，未来考虑加入更多测评机构到众测流程中，测评机构作为联盟链节点共同维护账本数据。

参考文献

[1] E. Segev, Crowdsourcing contests, European Journal of Operational Research 281 (2) (2020) 241–255.

[2] 冯剑红, 李国良, 冯建华, 众包技术研究综述, 计算机学报 38 (9) (2015) 1713–1726.

[3] Y. Simmhan, B. Plale, D. Gannon, A survey of data provenance in e-science, SIGMOD Record 34 (3) (2005) 31–36.

[4] 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) (2016) 11–20.

[5] M. Sigwart, M. Borkowski, M. Peise, S. Schulte, S. Tai, Blockchain-based data provenance for the internet of things, in: Proceedings of the 9th International Conference on the Internet of Things, 2019, pp. 15:1–15:8.

[6] P. Cui, J. Dixon, U. Guin, D. DiMase, A blockchain-based framework for supply chain provenance, IEEE Access 7 (2019) 157113–157125.

[7] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Tech. rep., Manubot (2019).

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

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

[10] E. Androulaki, A. Barger, et al., Hyperledger fabric: a distributed operating system for permissioned blockchains, in: Proceedings of the 13th EuroSys Conference, 2018, pp. 30:1–30:15.

[11] X. Liang, S. Shetty, et al., Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability, in: Proceed-

ings of the 17th International Symposium on Cluster, Cloud and Grid Computing, 2017, pp. 468–477.

[12] M. Montecchi, K. Plangger, M. Etter, It’s real, trust me! establishing supply chain provenance using blockchain, Business Horizons 62 (3) (2019) 283–293.

[13] P. Ruan, G. Chen, T. T. A. Dinh, Q. Lin, B. C. Ooi, M. Zhang, Fine-grained, secure and efficient data provenance on blockchain systems, VLDB Endowment 12 (9) (2019) 975–988.

[14] H. Han, M. Huang, Y. Zhang, U. A. Bhatti, An architecture of secure health information storage system based on blockchain technology, in: Proceedins of the 4th International Conference on Cloud Computing and Security, 2018, pp. 578–588.

[15] X. Yue, H. Wang, D. Jin, M. Li, W. Jiang, Healthcare data gateways: Found healthcare intelligence on blockchain with novel privacy risk control, Journal of Medical Systems 40 (10) (2016) 218:1–218:8.

[16] R. Burstall, B. Clark, Blockchain, ip and the fashion industry, Managing Intell 266 (5) (2017) 9–10.

[17] Y. Tung, S. Tseng, A novel approach to collaborative testing in a crowdsourcing environment, Journal of Systems and Software 86 (8) (2013) 2143–2153.

[18] S. Guo, R. Chen, H. Li, A real-time collaborative testing approach for web application: Via multi-tasks matching, in: Proceedings of the 16th International Conference on Software Quality, Reliability and Security, 2016, pp. 61–68.

[19] Y. Feng, Z. Chen, J. A. Jones, C. Fang, B. Xu, Test report prioritization to assist crowdsourced testing, in: Proceedings of the 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 225–236.

[20] Y. Feng, J. A. Jones, Z. Chen, C. Fang, Multi-objective test report prioritization using image understanding, in: Proceedings of the 31st International Conference on Automated Software Engineering, 2016, pp. 202–213.

[21] J. Wang, Q. Cui, Q. Wang, S. Wang, Towards effectively test report classification to assist crowdsourced testing, in: Proceedings of the 10th International Symposium on Empirical Software Engineering and Measurement, 2016, pp. 6:1–6:10.

[22] J. Wang, S. Wang, Q. Cui, Q. Wang, Local-based active classification of test report to assist crowdsourced testing, in: Proceedings of the 31st International Conference on Automated Software Engineering, 2016, pp. 190–201.

[23] W. Wu, W. Tsai, W. Li, An evaluation framework for software crowdsourcing, Frontiers of Computer Science 7 (5) (2013) 694–709.

[24] P. Buneman, S. Khanna, W. C. Tan, Why and where: A characterization of data provenance, in: Proceedings of the 8th International Conference on Database Theory, 2001, pp. 316–330.

[25] D. P. Lanter, Design of a lineage-based meta-data base for gis, Cartography and Geographic Information Systems 18 (4) (1991) 255–261.

[26] J. Zhao, C. Wroe, et al., Using semantic web technologies for representing e-science provenance, in: Proceedings of the 3rd International Semantic Web Conference, 2004, pp. 92–106.

[27] 明华, 张勇, 符小辉, 数据溯源技术综述, 小型微型计算机系统 33 (9) (2012) 1917–1923.

[28] L. Moreau, B. Clifford, J. Freire, et al., The open provenance model core specification, Future Generation Computer Systems 27 (6) (2011) 743–756.

[29] M. Wang, M. Blount, et al., A time-and-value centric provenance model and architecture for medical event streams, in: Proceedings of the 1st International Workshop on Systems and Networking Support for Healthcare and Assisted Living Environments, 2007, pp. 95–100.

[30] L. Chiticariu, W. C. Tan, G. Vijayvargiya, Dbnotes: a post-it system for relational databases based on provenance, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2005, pp. 942–944.

[31] B. Omelayenko, M. Klein, Tracing data lineage using schema transformation pathways, Knowledge transformation for the Semantic Web 95 (2003) 64.

[32] A. Woodruff, M. Stonebraker, Supporting fine-grained data lineage in a database visualization environment, in: Proceedings of the 13th International Conference on Data Engineering, 1997, pp. 91–102.

[33] C. Goble, Position statement: Musings on provenance, workflow and (semantic web) annotations for bioinformatics, in: Workshop on Data Derivation and Provenance, 2002.

[34] H. V. Jagadish, F. Olken, Database management for life sciences research, SIGMOD Record 33 (2) (2004) 15–20.

[35] S. Miles, P. T. Groth, M. Branco, L. Moreau, The requirements of using provenance in e-science experiments, Journal of Grid Computing 5 (1) (2007) 1–25.

[36] H. Galhardas, D. Florescu, D. E. Shasha, E. Simon, C. Saita, Improving data cleaning quality using a data lineage facility, in: Proceedings of the 3rd International Workshop on Design and Management of Data Warehouses, 2001, p. 3.

[37] I. T. Foster, J. Vöckler, M. Wilde, Y. Zhao, The virtual data grid: A new model and architecture for data-intensive collaboration, in: Proceedings of the 1st Biennial Conference on Innovative Data Systems Research, 2003, pp. 11–17.

[38] R. Bose, A conceptual framework for composing and managing scientific data lineage, in: Proceedings of the 14th International Conference on Scientific and Statistical Database Management, 2002, pp. 15–19.

[39] Z. Zheng, S. Xie, et al., An overview of blockchain technology: Architecture, consensus, and future trends, in: Proceedings of the 6th International Congress on Big Data, 2017, pp. 557–564.

[40] Z. Zheng, S. Xie, H. Dai, X. Chen, H. Wang, Blockchain challenges and opportunities: a survey, International Journal of Web and Grid Services 14 (4) (2018) 352–375.

[41] C. L. P. Chen, G. Wen, Y. Liu, F. Wang, Adaptive consensus control for a class of nonlinear multiagent time-delay systems using neural networks, IEEE Transactions on Neural Networks and Learning Systems 25 (6) (2014) 1217–1226.

[42] C. Cachin, M. Vukolic, Blockchain consensus protocols in the wild (keynote talk), in: Proceedings of the 31st International Symposium on Distributed Computing, 2017, pp. 1:1–1:16.

[43] 欧阳丽炜, 王帅, 袁勇, 倪晓春, 王飞跃, 智能合约: 架构及进展, 自动化学报 45 (3) (2019) 445–457.

[44] C. T. Nguyen, D. T. Hoang, et al., Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities, IEEE Access 7 (2019) 85727–85745.

[45] N. Stifter, A. Judmayer, E. R. Weippl, Revisiting practical byzantine fault tolerance through blockchain technologies, in: Security and Quality in Cyber-Physical Systems Engineering, 2019, pp. 471–495.

[46] I. Eyal, E. G. Sirer, Majority is not enough: Bitcoin mining is vulnerable, in: Proceedings of the 18th International Conference on Financial Cryptography and Data Security, 2014, pp. 436–454.

[47] W. C. Tan, Research problems in data provenance, IEEE Data Engineering Bulletin 27 (4) (2004) 45–52.

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

[49] I. Molyneaux, The art of application performance testing: from strategy to tools, 2014.

简历与科研成果

基本情况 刘子寒，男，汉族，1993 年 1 月出生，江苏省淮安市人。

教育背景

2018.9 ~ 2020.7	南京大学软件学院	硕士
2011.9 ~ 2015.6	南京大学计算机科学与技术学院	本科

致　　谢

首先，我想感谢我的导师陈振宇老师，以及王兴亚老师、房春荣老师、何铁科老师和刘嘉老师，感谢各位老师在我研究生阶段给予的指导和帮助。在毕业设计阶段，感谢各位老师在毕设选题到项目开发，直至论文最终完成各个阶段持续性的指导与建议。

其次，我想感谢实验室的乔力、张皓明、常家鑫等同学，感谢他们在毕设项目设计与实现阶段无私的建议和帮助。感谢我的室友张弛、叶继凡、袁豪三位同学，感谢他们在我硕士生涯生活学习上的帮助。

然后，我想感谢南京大学软件学院对我的培养，让我对软件工程领域有了浓厚的兴趣与必备的知识，感谢软院所有老师的辛勤付出与谆谆教诲。

最后，我想感谢我的父母和女友，感谢他们在我读书阶段对我的鼓励和支持，感谢他们在我遇到挫折时的关心与理解。