南京大学

研 究 生 毕 业 论 文

（申请工程硕士学位）

论 文 题 目	基于资源不变量的Move智能合约模糊测试系统
作 者 姓 名	袁博
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

2022 年 05 月 20 日

学　　号：**MF20320210**

论文答辩日期：**2022 年 05 月 20 日**

指 导 教 师：　　**（签字）**

Move Smart Contract Fuzzing System Based on Resource Invariant

By

Bo Yuan

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 2022

学 位 论 文 原 创 性 声 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，本论文不句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。

论文作者签名：_____

日期：　　年　　月　　日

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： 基于资源不变量的 Move 智能合约模糊测试系统

工程硕士（软件工程领域） 专业 2020 级硕士生姓名： 袁博

指导教师（姓名、职称）： 陈振宇 教授

摘 要

Libra 区块链是由 Facebook 主导开发的一种面向金融领域的新兴区块链平台，该平台使用 Move 智能合约进行数字资产的管理和交易。Move 智能合约的安全性直接关系到用户数字资产的安全，因此开展 Move 智能合约测试以保障其安全性很有必要。然而，由于 Libra 区块链和 Move 智能合约的新颖性，目前尚缺乏 Move 智能合约测试技术的相关研究。模糊测试能够通过随机生成大量测试输入，监控区块链系统运行异常结果来检测智能合约漏洞。因此，针对 Move 智能合约，可以利用模糊测试的方法检测其存在的漏洞，提高 Move 智能合约的安全性和可靠性，从而保护 Libra 区块链上用户财产安全。

本文通过分析 Move 智能合约中的资源在交易前后的结果，提出了一种 Move 智能合约测试预言——资源不变量，该测试预言可以有效检测 Move 智能合约中有关资源权限和资源使用方面的漏洞。基于此，本文设计并实现了基于资源不变量的 Move 智能合约模糊测试系统，该系统具有 Move 智能合约解析、Libra 区块链账户生成、测试脚本生成、测试脚本执行、测试结果查看等功能。该 Move 智能合约模糊测试系统通过解析待测试 Move 智能合约中的函数以构建函数种子池，系统随机引擎随机从函数种子池中挑选种子生成测试脚本。系统根据测试脚本执行结果，依据资源不变量判定 Move 智能合约中是否存在漏洞。为了丰富函数种子池，尽可能多地探索 Move 智能合约的执行路径，同时保证测试脚本顺利执行，系统还提出两种函数种子变异策略，用来在测试脚本执行后对测试脚本所选取的种子进行变异，以生成新的测试脚本。

本文的基于资源不变量的 Move 智能合约模糊测试系统采用 Vue 框架和 Spring Boot 框架分别搭建系统的前后端逻辑，并使用 Mysql 数据库完成数据的持久化。通过系统实际运行，本文提出的基于资源不变量的 Move 智能合约模糊测试系统可以有效检测出 Move 智能合约中的错误使用 signer、资源的意外销毁、资源的错误修改等漏洞。经过功能测试，该模糊测试系统满足预期设计的功能性需求，系统简便可用。

关键词： Libra 区块链，Move 智能合约，模糊测试

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Move Smart Contract Fuzzing System Based on Resource Invariant

SPECIALIZATION: Software Engineering

POSTGRADUATE: Bo Yuan

MENTOR: Professor **Zhenyu Chen**

Abstract

The Libra blockchain is an emerging blockchain platform developed by Facebook for the financial sector. The platform uses Move smart contracts to manage and trade digital assets. The security of Move smart contracts is directly related to the security of users' digital assets, so it is necessary to carry out the Move smart contract test to ensure its security. However, due to the novelty of the Libra blockchain and Move smart contracts, there is currently a lack of relevant research on the Move smart contract testing technology. Fuzzing can detect smart contract vulnerabilities by randomly generating a large number of test inputs and monitoring the abnormal results of the operation of the blockchain system. Therefore, for the Move smart contract, fuzzing can be used to detect its existing vulnerabilities, improve the security and reliability of the Move smart contracts, and protect the safety of users' property on the Libra blockchain.

By analyzing the results of the resources in Move smart contracts before and after the transaction, this thesis proposes a kind of Move smart contract test oracle—resource invariant, which can effectively detect vulnerabilities of resource permission and resource usage in Move smart contracts. This thesis designs and implements a Move smart contract fuzzing system based on resource invariant, which has functions of Move smart contract parsing, Libra blockchain account generation, test script generation, test script execution, test result viewing and so on. The Move smart contract fuzzing system builds a function seed pool by parsing the functions in the Move smart contract to be tested, and the system random engine randomly selects seeds from the function seed pool to generate test scripts. According to the execution result of test scripts, The system determines whether there is a vulnerability in the Move smart contract by using resource invariant. In order to enrich the function seed pool, explore the

execution path of the Move smart contract as much as possible, and ensure the smooth execution of test scripts, the system also proposes two function seed mutation strategies, which are used to mutate the seeds selected by the executed test scripts, to generate new test scripts.

The Move smart contract fuzzing system based on resource invariant in this thesis adopts Vue framework and Spring Boot framework to build the front-end and back-end logic of the system respectively, and uses Mysql database to complete data persistence. Through the actual operation of the system, the system proposed in this thesis can effectively detect vulnerabilities in Move smart contracts, such as the wrong use of signers, the accidental destruction of resources, ,the wrong modification of resources and so on. After a series of functional tests, the system meets the functional requirements of the expected design, and the system is usable.

Keywords: Libra Blockchain,Move Smart Contract, Fuzzing

目录

表 目 录 ····· **vii**

图 目 录 ····· **ix**

第一章 绪论 ····· **1**

1.1 课题背景和意义 ····· 1

1.2 国内外研究现状 ····· 2

1.2.1 智能合约安全分析 ····· 2

1.2.2 模糊测试 ····· 4

1.3 本文主要工作 ····· 5

1.4 本文组织架构 ····· 5

第二章 相关概念和技术 ····· **7**

2.1 Libra 区块链 ····· 7

2.1.1 Libra 概述 ····· 7

2.1.2 Libra 逻辑数据模型 ····· 8

2.1.3 Libra 协议 ····· 9

2.2 Move 智能合约 ····· 11

2.2.1 Move 概述 ····· 11

2.2.2 Move 语言特性 ····· 12

2.3 智能合约模糊测试技术 ····· 13

2.3.1 模糊测试概述 ····· 13

2.3.2 模糊测试方法 ····· 14

2.4 本章小结 ····· 15

第三章 基于资源不变量的 Move 智能合约模糊测试方法 ····· **17**

3.1 引言 ····· 17

3.2 Move 智能合约漏洞类型分析 ····· 18

3.2.1 错误使用 signer ····· 18
3.2.2 函数可见性声明错误 ····· 19
3.2.3 资源的意外销毁 ····· 21
3.2.4 错误的资源修改 ····· 22
3.3 Move 智能合约模糊测试方法 ····· 23
3.3.1 Move 智能合约测试预言 ····· 23
3.3.2 Move 智能合约模糊测试框架 ····· 24
3.4 本章小结 ····· 27

第四章 需求分析与概要设计 ····· 29
4.1 系统整体概述 ····· 29
4.2 需求分析 ····· 29
4.2.1 用例分析 ····· 29
4.2.2 功能性需求 ····· 33
4.2.3 非功能性需求 ····· 34
4.3 总体设计 ····· 34
4.3.1 系统架构 ····· 35
4.3.2 4+1 视图 ····· 36
4.3.3 持久化设计 ····· 40
4.4 本章小结 ····· 42

第五章 系统实现与测试 ····· 43
5.1 系统详细设计与实现 ····· 43
5.1.1 测试种子生成模块设计与实现 ····· 43
5.1.2 测试脚本生成模块详细设计与实现 ····· 46
5.1.3 模糊测试执行模块详细设计与实现 ····· 50
5.1.4 报告生成模块详细设计与实现 ····· 53
5.1.5 系统运行展示 ····· 55
5.2 系统测试 ····· 57
5.2.1 测试环境 ····· 57
5.2.2 系统测试设计 ····· 57
5.2.3 测试结果与分析 ····· 60
5.3 本章小结 ····· 60

第六章 总结与展望 ····· **61**
6.1 总结 ····· 61
6.2 展望 ····· 61

参考文献 ····· **63**

简历与科研成果 ····· **68**

致谢 ····· **69**

表 目 录

4.1 Move 智能合约解析用例描述 ····· 30
4.2 区块链账户创建用例描述 ····· 31
4.3 测试脚本生成用例描述 ····· 31
4.4 测试脚本执行用例描述 ····· 32
4.5 测试结果查看用例描述 ····· 32
4.6 测试结果下载用例描述 ····· 33
4.7 Move 智能合约信息表 ····· 41
4.8 Move 智能合约函数信息表 ····· 41
4.9 区块链账户信息表 ····· 41
4.10 模糊测试执行结果信息表 ····· 42

5.1 测试环境配置表 ····· 58
5.2 Move 智能合约解析测试用例 ····· 58
5.3 区块链账户生成测试用例 ····· 59
5.4 Move 智能合约模糊测试测试用例 ····· 59
5.5 模糊测试结果展示测试用例 ····· 60
5.6 功能测试用例执行结果 ····· 60

图 目 录

2.1 四账户账本状态图 …… 9
2.2 资源示例图 …… 9
2.3 Libra 协议图 …… 10
2.4 模块结构图 …… 11
2.5 模块代码示例图 …… 11
2.6 脚本结构图 …… 12
2.7 脚本代码示例图 …… 12
2.8 智能合约模糊测试流程图 …… 14

3.1 错误使用 signer 漏洞代码示例图 …… 19
3.2 错误使用 signer 漏洞触发示例图 …… 19
3.3 函数可见性声明漏洞代码示例图 …… 20
3.4 函数可见性声明错误漏洞触发示例图 …… 20
3.5 资源的意外销毁漏洞代码示例图 …… 21
3.6 资源的意外销毁漏洞触发示例图 …… 21
3.7 错误的资源修改漏洞代码示例图 …… 22
3.8 资源的错误修改漏洞触发示例图 …… 23
3.9 Move 智能合约模糊测试框架图 …… 25

4.1 系统用例图 …… 30
4.2 系统整体架构图 …… 35
4.3 逻辑视图 …… 36
4.4 开发视图 …… 37
4.5 进程视图 …… 39
4.6 物理视图 …… 40

5.1 测试种子生成模块类图 …… 44
5.2 测试种子生成模块活动图 …… 44

5.3 测试种子生成模块关键代码 45
5.4 测试脚本生成模块类图 46
5.5 区块链账户生成活动图 47
5.6 测试脚本生成功能活动图 48
5.7 区块链账户生成功能关键代码 49
5.8 测试脚本生成功能关键代码 50
5.9 模糊测试执行模块类图 51
5.10 模糊测试执行模块活动图 52
5.11 模糊测试执行模块关键代码 52
5.12 报告生成模块类图 53
5.13 报告生成模块活动图 54
5.14 报告生成模块关键代码 55
5.15 Move 智能合约预处理页面展示图 56
5.16 区块链账户生成页面展示图 56
5.17 测试脚本展示页面展示图 57
5.18 模糊测试执行结果展示页面展示图 57

第一章　绪论

1.1　课题背景和意义

Libra 区块链是一个分散的、可编程的数据库, 最初由 Facebook 在 2019 年 6 月发布的白皮书中提出并主导开发 [1]。从结构上来看，Libra 更接近联盟链，其数据库由数十至一百个左右 Libra 协会创始会员担任的“验证节点”进行维护。在技术上，Libra 提出了一种全新的智能合约编程语言 Move [2]，在数字资产管理方面具有更强的安全性。Libra 区块链通过允许以单一货币作为抵押，承担价值尺度、流通手段、贮藏手段、支付手段和世界货币等货币基本职能，从而建立一个简单的全球支付系统和金融基础设施。因此，确保 Libra 区块链上数字资产的安全性是重中之重。智能合约是一段存储在区块链上的计算机程序，程序代码中包含了交易双方之间的协议条款以及协议条款的触发条件。智能合约被部署在区块链上后，在满足预定条件后将会自动执行。目前，已有数以万计的智能合约被部署在以太坊 [3] 等主流区块链上。Libra 区块链采用由 Move 语言编写的智能合约来对区块链上的数字资产进行管理，因此需要研究如何提高 Move 智能合约的安全性来保护数字资产。

智能合约管理着区块链平台的数字货币，引来许多攻击者采取各种手段企图非法获取他人的数字资产。智能合约本质上是一段计算机程序，开发人员在编写程序时不可避免地会出现代码错误和缺陷，导致智能合约出现可被不法分子利用的安全漏洞。同时，智能合约一旦在区块链上完成部署后就再也不能被修改，导致智能合约漏洞无法被修复。据 Slowmist[4] 统计，智能合约安全漏洞已经造成数以亿计美元的财产损失。Move 智能合约充分考虑了数字资产的安全性，其利用一种名为资源的数据结构来管理 Libra 区块链上的数字资产。资源如同现实中的物理资产一样只能被移动，不能被复制。Move 智能合约开发人员如果对 Move 智能合约中资源的特性理解不深入，他们在利用资源进行数字资产管理时就容易编写出可被攻击者利用的安全漏洞。通过调查和分析，Move 智能合约中可能存在资源权限和资源使用两方面的问题。资源权限问题可能导致用户无法使用和管理特定的资源，资源使用问题可能导致用户间交易出错。因此，Move 智能合约中有关资源的安全漏洞可能导致 Libra 区块链上用户数字资产的损失，凸显了 Move 智能合约漏洞检测的重要性。

模糊测试是故意向系统发送无效数据以希望触发错误情况或故障的过程 [5]。对智能合约进行模糊测试时，首先利用随机引擎生成大量的测试输入，并构成

可执行交易。区块链执行交易后，随机引擎会根据交易执行反馈信息动态调整生成新的测试输入，从而探索尽可能多的智能合约状态空间。智能合约模糊测试通过使用测试预言来比较预期测试输出和实际测试输出，进而判断智能合约中是否存在安全漏洞。本文为了检测 Move 智能合约中是否存在安全漏洞，提出了一种名为资源不变量的测试预言。基于此，本文提出一种基于资源不变量的 Move 智能合约模糊测试方法 Libra Move Fuzz（LMF）。LMF 根据 Move 语言特性在资源权限和资源使用方面对 Move 智能合约可能存在的漏洞展开分析，并结合资源不变量为潜在漏洞设计了测试预言用于模糊测试判定。在对 Move 智能合约进行模糊测试时，LMT 首先随机生成测试脚本并在 Libra 本地链上执行，然后根据脚本执行反馈动态调整测试脚本的生成，最后结合测试预言判定 Move 智能合约是否存在安全漏洞以及存在哪些安全漏洞。在此基础上，本文基于 LMF 应用 Vue 和 Spring Boot 框架设计并实现了一个 Move 智能合约模糊测试系统。该系统由测试种子生成模块、测试脚本生成模块、模糊测试执行模块和报告生成模块组成，实现了 Move 智能合约上传，测试脚本生成、测试脚本执行、漏洞信息查看和漏洞报告生成等功能。

1.2 国内外研究现状

本文研究基于资源不变量的 Move 智能合约模糊测试方法，主要涉及智能合约安全分析和模糊测试两个方向，下面分别介绍国内外研究现状。

1.2.1 智能合约安全分析

智能合约安全漏洞一方面造成了用户的财产损失，另一方面也降低了用户对区块链的信任。近年来，为了在智能合约上线前尽可能地检测出安全漏洞，研究者们通过分析智能合约提出了一系列智能合约安全分析方法，包括形式化验证 [6][7][8]、符号执行 [9][10]、深度学习 [11][12] 以及模糊测试 [13][14][15] 等技术。本节将从以上四个方面介绍智能合约安全分析的研究现状。

形式化验证通过形式化语言把智能合约中的代码进行形式化建模，然后结合严谨的数学证明和推理，验证智能合约中函数功能的正确性和安全性 [16]。Grishchenko[17] 等人提出了一种 *F** framwork，通过对以太坊虚拟机 EVM 字节码的语义进行形式化建模，并以 Ocaml 形式对智能合约字节码进行再编译，最后实现以函数编程语言 *F** 的形式描述智能合约源码以及字节码。 *F** framwork 验证智能合约函数功能的正确性和安全性以达到检测以太坊智能合约漏洞的目的。Amani[18] 等人提出了一种形式化验证方法 Isabelle/HOL，该方法通过将智能合约拆分成多个基本块或者将智能合约字节码序列构造成直线代码块，并对这些合

约基本块或者直线代码块进行逻辑推理验证。Kalra[19] 等人提出了一种静态分析工具 ZEUS, 其利用形式化验证技术实现了对可重入漏洞、异常无序漏洞等多种智能合约漏洞的检测。成都链安科技研发的 VaaS[20] 是一种支持以太坊智能合约的“一键式”智能合约形式化验证平台，能够检测整数溢出、可重入漏洞等多种类型漏洞，并对漏洞位置进行精确定位。

符号执行法的主要思想是通过采用抽象的符号代替精确值作为程序输入变量，得出每个路径抽象的输出结果 [21]。针对智能合约漏洞检测，符号执行法首先符号化智能合约中的变量值，接着逐条解释执行程序中的指令，在解释执行过程中更新执行状态、搜集路径约束，以探索智能合约程序中全部可执行路径，从而发现智能合约安全问题。Luu[22] 等人提出了一种智能合约静态检测工具 Oyente，其利用符号执行的方法能够实现对可重入漏洞、交易顺序依赖漏洞等多种智能合约安全漏洞的检测。Nikolić[23] 等人提出了一种基于符号分析的区块链分析工具 Maian，其主要针对“挥霍”、“贪婪”和“自杀”3 种类型的智能合约漏洞进行分析和检测。Tsankov[24] 等人提出了静态安全分析工具 Securify，其主要针对以太坊智能合约，通过符号化分析智能合约的依赖图关系，提取代码的语义信息，从而检测智能合约安全漏洞。Krupp[25] 等人提出的智能合约分析工具 TeEther 利用以太坊虚拟机字节码查找关键执行路径，并以符号方式执行路径，从而实现对智能合约漏洞的自动识别和自动利用。

近年来，深度学习的大热催生了其在程序安全领域的大规模应用，深度学习也表现出了在智能合约安全分析领域的适应性以及可扩展性。Tann[26] 等人提出的 SaferSC 是第一个基于深度学习的智能合约漏洞检测模型, 其利用 LSTM 网络构建操作码模型，针对 Maian 提出的 3 类智能合约漏洞进行分析，提高了检测这三类漏洞的准确率。Zhuang[27] 等人提出了一种利用合约图 (contract graph) 来检测智能合约漏洞的方法 DR-GCN, 其利用图卷积神经网络为转换成合约图结构的智能合约源码构建了漏洞检测模型。Wang[28] 等人提出的 ContractWard 对智能合约操作码进行规范化, 保留了原有的语义特征，并利用多种机器学习算法对智能合约安全漏洞进行检测。

模糊测试是一种动态软件分析技术。模糊测试针对目标程序生成大量随机的测试用例, 并监视目标程序在执行测试用例过程中的异常状态来判断是否出现安全问题。Jiang[29] 等人提出的 ContractFuzzer 是一种以太坊智能合约安全漏洞模糊测试框架, 其基于智能合约二进制代码和函数 ABI 接口构造测试用例，从而实现对 7 类智能合约安全漏洞的检测。Wüstholz[30] 等人提出的 Harvey 是一种应用于智能合约的灰盒模糊测试工具，其利用需求驱动对智能合约交易序列进行模糊测试，提高了智能合约安全漏洞的检测效率。

形式化验证方法依赖于严谨的数学推导和证明，且无法执行动态分析，无法保证漏洞一定存在可达的可执行路径，误报率较高。符号执行法虽然能够遍历合约代码中所有可执行路径，提高检测效果，但一定程度上也造成了漏洞检测过程中的资源消耗和时间开销过高的问题。深度学习法通过预处理智能合约来构建数据集供深度学习模型进行学习，但神经网络是一个“黑箱”，可解释性较差，无法给出智能合约漏洞在合约代码中的准确位置。模糊测试法由于测试数据的随机性导致路径覆盖率无法达到理想状态，但与其他技术相比模糊测试具有良好的可扩展性和适用性。因此经过综合考虑，加上 Move 智能合约安全分析方面暂未有针对 Move 智能合约进行模糊测试的已出版的文献研究，本文选择采用模糊测试的方法对 Move 智能合约进行漏洞检测。

1.2.2 模糊测试

模糊测试是一种比较常规的软件测试技术，其在不同领域都有研究和应用，例如在物联网模糊测试和内核模糊测试两个领域有着较为集中的研究。本节将从以上两个领域介绍模糊测试在其他领域的研究现状，并分析智能合约模糊测试的不同之处。

近年来，“万物互联”成为当下非常流行的概念，但物联网领域的快速发展导致针对物联网设备的攻击开始增多，不少研究者开始将模糊测试技术运用到复杂的物联网环境中。Zheng[31] 等人提出的 FIRM-AFL 是一种针对物联网硬件的灰盒模糊测试工具，其对可在系统仿真器中被模拟的 POSIX 兼容固件进行模糊测试从而解决兼容性问题，并使用增强过程仿真技术解决了模糊测试的性能瓶颈，提高了漏洞检测速度。Chen[32] 等人提出的自动模糊测试框架 IOT-Fuzzer，其通过识别和重用程序特定的逻辑来改变测试用例，可以检测出物联网设备中 15 个内存损坏漏洞。

内核是现代计算机操作系统中最基本的组成部分，其可以决定应用程序对计算机硬件的操作权限和操作时间。内核安全漏洞威胁着整个计算机操作系统的安全，因此，不少研究者专门研究如何利用模糊测试等技术对内核安全进行检测和分析。syzkaller[33] 是由 Google 开发的一种内核模糊测试工具，其利用系统调用对内核进行模糊测试。Jeong[34] 提出的 Razzer 是一种在内核中查找竞争错误的工具，其针对内核中潜在的数据竞争点进行模糊测试，从而发现内核竞争漏洞。Corina[35] 等人提出的一种接口感知的模糊测试工具 DIFUZE，其可自动生成有效输入并触发内核驱动程序的执行，该工具检测出了 32 种新的内核驱动漏洞。

模糊测试作为一种比较成熟的漏洞检测技术，其在各个领域都有应用前景。

与其他领域尤其是传统程序相比，智能合约模糊测试存在一些困难和挑战。一方面，智能合约的函数不仅可以内部调用其他函数，也可以调用外部合约的函数，导致智能合约状态变量受多方面影响，无法有效生成测试用例；另一方面，智能合约不会像一般的计算机程序那样崩溃，在出现异常情况时它们的执行可能会被默默地恢复，从而难以判断漏洞的存在。为了对 Move 智能合约漏洞进行有效的检测，本文充分考虑 Move 智能合约特性，研究模糊测试在 Move 智能合约漏洞检测方向的应用。

1.3 本文主要工作

本文提出了基于资源不变量的 Move 智能合约模糊测试方法并实现了一个 Move 智能合约模糊测试系统，主要工作如下：

在方法上，本文通过分析 Move 智能合约编程语言的特性，研究 Move 智能合约中可能存在的资源权限与资源使用方面的漏洞，并结合资源不变量为每一种漏洞设计了测试预言。在模糊测试过程中，本文首先对 Move 智能合约进行解析，从而获取合约中的函数及对应的函数签名。接着本文利用随机引擎生成函数输入构建出函数种子，并利用种子生成测试脚本。然后本文在 Libra 本地链上执行测试脚本，并依照测试预言判定 Move 智能合约中是否存在安全漏洞。最后，随机引擎还会根据脚本执行信息对未通过测试判定的测试脚本中用到的函数种子进行变异，以探索更多的合约路径。

在系统上，本文在理论研究的基础上，设计并实现了一个 Move 智能合约模糊测试系统。该系统由测试种子生成模块、测试脚本生成模块、模糊测试执行模块和报告生成模块组成，实现了 Move 智能合约上传，测试脚本生成、测试脚本执行、漏洞信息查看和漏洞报告生成等功能。本系统基于前后端分离的架构进行实现，使用 Vue 作为前端框架，Spring boot 作为后端框架，两端统一采用 JSON 数据格式，利用符合 RESTful API 规范的接口进行通信；采用 MySQL 和文件系统对数据进行持久化处理。

1.4 本文组织架构

本文详细介绍了基于资源不变量的 Move 智能合约模糊测试方法的设计和 Move 智能合约模糊测试系统的实现。全文共分为六个章节，各章节主要内容如下：

第一章，绪论。本章首先介绍了基于资源不变量的 Move 智能合约模糊测试系统的课题背景和研究意义，然后从智能合约安全分析和模糊测试两方面介绍

了国内外研究现状，阐明本文使用模糊测试作为 Move 智能合约漏洞检测的原因。最后介绍了本文的主要工作。

第二章，相关概念和技术。本章首先介绍了 Libra 区块链的相关知识和特点。接着介绍了 Move 智能合约的组成部分以及所用 Move 语言的语言特性。最后介绍了模糊测试的基本流程和主要优势。

第三章，基于资源不变量的 Move 智能合约模糊测试方法。本章首先分析了 Move 智能合约在资源权限和资源使用方面的几类漏洞。接着，给出资源不变量的定义，介绍了根据资源不变量设计的漏洞测试预言。最后，详细描述了本文提出的 Move 智能合约模糊测试方法的设计思路。

第四章，需求分析与概要设计。本章首先对 Move 智能合约模糊测试系统进行需求分析，给出系统用例图，确定系统的功能性需求和非功能性需求。接着结合整体架构图、4+1 视图以及数据库存储表对系统进行了概要设计。

第五章，系统实现与测试。本章首先介绍系统四个模块的详细设计与实现，包括测试种子生成模块、测试脚本生成模块、模糊测试执行模块以及报告生成模块，接着通过系统截图展示系统的操作流程并加以描述说明。然后对系统功能性进行测试设计，并对测试结果进行分析。

第六章，总结与展望。本章总结了本文主要工作，指出 Move 智能合约模糊测试系统当前存在的不足以及下一步工作的主要方向。

第二章　相关概念和技术

本章主要介绍相关概念和技术，包括 Libra 区块链、Move 智能合约以及智能合约模糊测试技术。首先，简要介绍本课题使用的区块链平台——Libra 区块链的基础知识，然后介绍 Libra 的核心机制，包括 Libra 协议以及逻辑数据模型。本课题针对 Move 智能合约进行模糊测试方面的研究，因此接着介绍 Move 智能合约的基础知识。最后，对智能合约模糊测试的基本工作流程进行详细描述。

2.1　Libra 区块链

2.1.1　Libra 概述

区块链是一种在计算机网络的各节点之间共享交易的分布式数据库，链上信息很难被篡改和破解。区块链将信息组织在名为区块的结构中，每个区块拥有一组信息。区块具有一定的存储容量，当某个区块被填满时，该区块不再被加入信息并与之前填满的区块相连，形成一个被称为区块链的数据链。区块链因其在加密货币系统（如比特币）中维护安全和分散的交易记录的关键作用而闻名。区块链具有去中心化、防篡改等特点，使其能够在不需要可信第三方的情况下产生信任，保证了数据记录的安全性和保真性。目前，区块链已在供应链 [36][37]、医疗 [38][39]、物联网 [40][41] 等领域被广泛应用。

Libra 区块链是一种使用 Libra 协议实现的加密认证数据库，它的目标是成为满足数十亿人日常金融需求的金融服务基础设施。Libra 区块链数据库存储了一个可编程资源的账本——Libra Coin，Libra Coin 作为一种单一货币用以抵押，满足 Libra 区块链金融货币需求。与传统区块链如以太坊相比，Libra 区块链做出了 3 点改进。（1）Libra 区块链提出了一种全新的智能合约编程语言——Move 编程语言。Move 语言从 Solidity 智能合约常见的安全漏洞中吸取经验，通过使用静态类型系统、设计资源结构和摒弃动态委派等措施，尽可能降低了使用 Move 语言编写的智能合约出现漏洞的风险。（2）Libra 区块链采用了基于拜占庭容错的 LibraBFT 共识机制。相比基于 PoS 及 PoW 等共识机制的区块链，Libra 区块链具有高交易处理量、低延迟和更高能效等优势。（3）Libra 区块链采用 Merkle 树的存储结构存储数据。基于 Merkle 树，Libra 区块链可以长期记录交易历史和状态，并且可以随时读取数据以及获知数据发生的变化，保证了 Libra 区块链上数据的完整性。

2.1.2 Libra 逻辑数据模型

Libra 区块链中的所有数据都存储在一个单一版本的数据库中 [42][43]。区块链数据库版本号是一个无符号的 64 位整数，版本号在数值上等于区块链系统已执行的交易数。某个版本 i 的 Libra 区块链数据库包含一个元组 (T_i,O_i,S_i)，其中 T_i 表示交易，O_i 表示交易执行的输出，S_i 表示账本状态。给定一个执行函数 Apply，上述元组的含义是: 根据账本状态 S_{i-1} 执行交易 T_i 产生交易输出 O_i 和一个新的账本状态 S_i，即

$$Apply(S_{i-1}, T_i) \rightarrow < O_i, S_i >$$

交易。交易代表 Libra 区块链上任意两个账户之间数据和资产的交换。当一个交易被提交到 Libra 区块链上并被执行后，Libra 区块链的账本状态将会更新。一个交易包括一个交易脚本（用 Move 语言编写）和交易脚本的参数（如发送方账户地址、交易金额、接收方账户地址等）。Libra 区块链通过运行脚本来执行交易，并将交易脚本参数和当前的账本状态作为输入，以产生一个完全确定的交易输出。

交易输出。交易输出指执行完一次交易后更新的账本状态、执行状态码的集合。执行状态码记录了交易执行的结果（如 success、error 等）。

账本状态。账本状态存储了 Libra 区块链上所有账户的基本信息，包括每个账户在给定版本中持有的 Libra Coin 数量。每个验证者都必须知晓当前最新版本的账本状态才能执行新交易。Libra 区块链使用基于账户的数据模型 [44] 对账本状态进行编码。账本状态被结构化为键值对存储，它将帐户地址映射到帐户值。账本状态中的账户值是已发布的 Move 资源和模块的集合。Move 资源存储数据值，模块存储代码。

以图2.1为例，进一步说明 Libra 帐户的状态结构以及各帐户间的关系。在图中，椭圆代表 Move 资源，矩形代表 Move 模块。Move 资源和 Move 模块之间的有向边表示该 Move 资源的类型是由箭头所指的 Move 模块声明的。地址 0x10 上的帐户包含 Currency.T 资源，该资源由 Currency 模块声明。Currency 模块的代码存储在地址 0x30 上。地址 0x20 的帐户包含 MyModule.T 资源，MyModule.T 资源由存储在地址 0x40 上的 MyModule 模块声明。地址 0x30 上的账户同样包含 Currency.T 资源，该资源由同地址上的 Currency 模块声明。

账户。在 Libra 区块链中，账户分为账户地址和账户内容两部分。账户地址是一个 256 位的值，唯一标识一个账户。账户内容包含有任意数量的 Move 模块和 Move 资源。为了创建一个新帐户，用户首先为签名方案生成一个新的验证/签

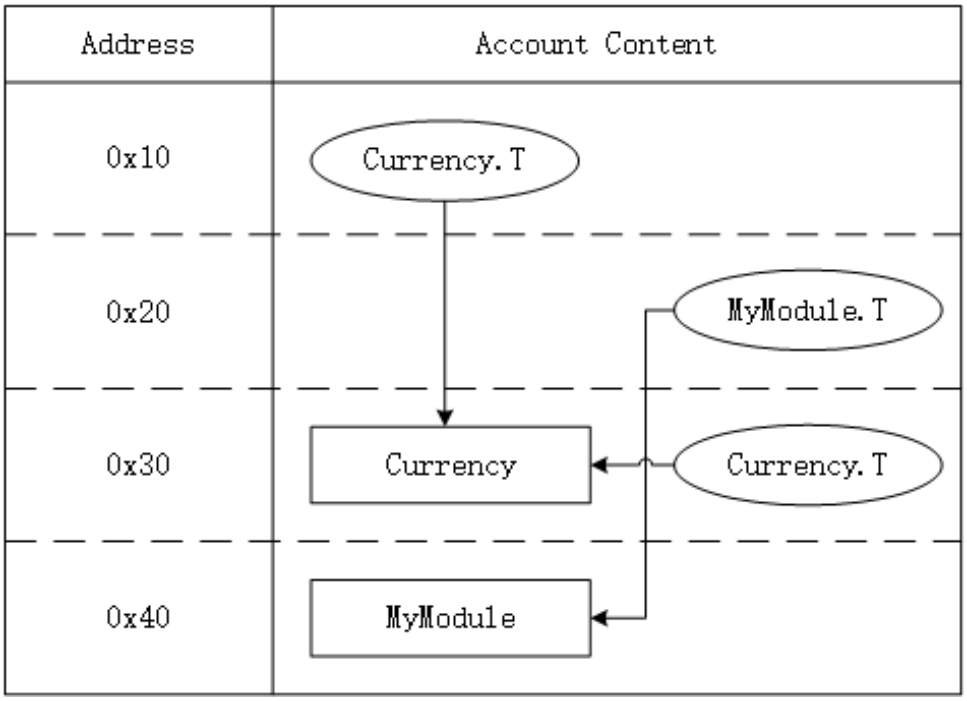

图 2.1: 四账户账本状态图

名密钥对 (v_k, s_k)，并使用公共验证密钥 v_k 的加密哈希值作为帐户地址 $a = H(v_k)$。当从现有账户发送的交易调用 create_account(a) 指令时，区块链在地址 a 上创建一个新账户并更新在账本状态中。当一个新账户在地址 a 上被成功创建后，用户可以使用私有签名密钥 s_k 签署要从该帐户发送的交易。

资源。资源是将命名字段绑定到简单值（如整数）或复杂值（如其他资源）的记录。如图2.2所示，资源由 resource 关键字定义，其后是资源名称 Coin，资源的值是整数类型 u64。每个资源都由一个模块来声明它的资源类型。资源类型是标称类型 [45]，由资源名称和资源声明模块的名称和地址组成。如图2.1中的 MyModule.T 资源，它的类型是 0x40.MyModule.T。其中 0x40 是存储 MyModule 模块的地址，MyModule 是模块的名称，MyModule.T 是资源的名称。

```
1 resource Coin {
2     value: u64
3 }
```

图 2.2: 资源示例图

2.1.3 Libra 协议

Facebook 团队提出了一种用以实现 Libra 区块链的 Libra 协议 [1]。Libra 协议允许来自不同机构的一组验证节点共同处理交易并维护 Libra 区块链数据库。

多个验证节点处理交易并进行交互，以就数据库的状态达成共识。

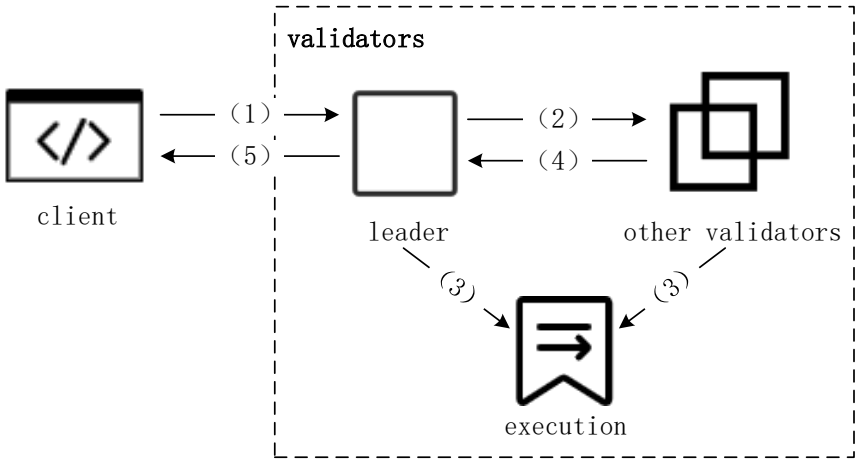

图 2.3: Libra 协议图

如图 2.3所示，Libra 协议中有两种实体类型进行交互以完成交易的提交及处理：负责维护数据库的验证节点（Validators）和可以对数据库执行查询操作并提交交易来修改数据库状态的客户端（Client）。交互的具体流程如下：在第（1）步中，客户端提交交易，验证节点接收并开始处理该交易。在第（2）步中，接收到客户端提交的交易的验证节点将作为领导节点（leader），领导节点会向其他验证节点广播一个交易区块，该交易区块包括由客户端直接提交给领导节点的交易和通过其他验证节点间接提交的交易。在第（3）步中，区块链上的所有验证节点均执行由领导节点发出的交易。在第（4）步中，所有验证节点执行完交易后将依据共识协议对交易列表以及交易执行结果进行投票，若投票达成一致，交易结果将存储在数据库中。在第（5）步中，领导节点向客户端返回交易结果和代表当前版本数据库完整状态（包括整个版本历史）的签名。

客户端可以向验证节点发出查询以从 Libra 区块链数据库中读取数据。由于 Libra 区块链数据库是经过共识协议验证的，因此验证节点可以向客户端保证其查询结果的准确性。作为数据查询响应的一部分，验证节点还会返回其内部存储的最新数据库版本 i 的签名验证器。此外，客户端可以选择通过同步验证节点的交易历史来创建整个区块链数据库的副本。在创建副本的同时，客户端可以对验证节点是否正确执行交易进行验证，这增加了 Libra 区块链的透明度。其他客户端可以从持有副本的客户端那里读取数据，如同直接向验证节点发起查询，获取查询数据一样。

2.2 Move 智能合约

2.2.1 Move 概述

Libra 区块链使用 Move 语言编写智能合约。Move 语言 [2] 是一种安全灵活的 Libra 区块链编程语言，其设计灵感来源于 Rust。Move 语言是一种可执行的字节码语言，用户可以通过 Move IR（Move 中间表示层）编写脚本和模块。Move 的关键特性是能够自定义资源类型，其语义受线性逻辑启发。资源类型用于编码行为类似于普通程序值的可编程资产：资源可以存储在数据结构中，作为参数传递给过程等。Move 类型系统为资源提供了特殊的安全保证，即一个资源永远无法被复制，只能移动。此外，资源类型只能由声明该类型的模块创建或销毁，这些保证由 Move VM 静态强制执行。Move 智能合约分为模块和脚本两种类型，介绍如下：

```
module <address>::<identifier> {
    (<use> | <friend> | <type> | <function> | <constant>)*
}
```

图 2.4: 模块结构图

```
module 0x1::MyModule {
    struct Coin has copy, drop { i: u64 }
    use Std::Debug;

    const ONE: u64 = 1;

    public fun add(x: u64) {
        let sum = x + ONE;
        Debug::print(&sum);
    }
}
```

图 2.5: 模块代码示例图

(1) **模块**: 模块是定义结构类型以及对这些类型进行操作的函数的库。结构类型定义了 Move 的全局存储模式，模块函数定义了更新存储的规则。模块本身也存储在全局存储中。模块的基本结构如图2.4所示。其中 address 是一个合法的区块链地址，表示模块在该地址上发布，由该地址所有。identifier

是模块的名称。在模块内部可以包含 use 声明、friend 声明、资源结构类型、常量以及一些功能函数。模块中的每个组件的具体示例如图2.5所示。其中，模块的地址是 0x1，模块的名称是 MyModule。模块包含一个结构类型 Coin，结构中值的类型是整数 u64。模块内部还包含一个名叫 add 的函数，接收函数输入并与常量 ONE 相加。

(2) **脚本**: 脚本是可执行的入口点，类似于传统语言中的 main 函数。脚本通常调用已发布模块的函数，这些函数执行对全局存储的更新。脚本的基本结构如图2.6所示。脚本代码块必须包含一个用于声明所调用的 Move 模块的 use 声明和一个 main 函数。main 函数是脚本中唯一的函数，可以有任意数量的参数，不能有返回值。脚本示例如图2.7所示。脚本中调用了发布在 0x1 地址上的 MyModule 模块中的 add 函数。

```
script {
    <use>*
    <constants>*
    fun <identifier><[type parameters: constraint]*>([identifier: type]*)
        <function_body>
}
```

图 2.6: 脚本结构图

```
script {
    use 0x1::MyModule;
    fun main(x: u64) {
        MyModule::add(x);
    }
}
```

图 2.7: 脚本代码示例图

2.2.2 Move 语言特性

Move 语言在设计之初就充分考虑了区块链上用户数字资产的安全性，总体而言，Move 语言具备如下 3 种特性：

(1) Move 语言采用静态类型系统。类型系统本质上是一种逻辑约束。静态类型系统的优点是在编译 Move 智能合约的时候就可以发现很多低级编程错

误，而不是在 Libra 区块链运行 Move 智能合约时才发现 Move 智能合约存在漏洞。Move 语言从虚拟机开始就是静态类型化的，中间语言层 IR 也是类型化的，Move 上层高级语言也是静态类型，这三者同时保证了 Move 智能合约不再会出现低级错误。

(2) Move 语言采用“资源是一等公民”理念。“资源是一等公民”（First-class Resources）的含义是在编程的时候首要考虑的被编程对象是资源（Resources）。资源是和值（Value）相对应的概念。值是可以随意拷贝的，而资源只能被移动，不能被拷贝。Move 智能合约采用了一种吸收了传统理论“线性逻辑”的类型，叫做“资源类型”。数字资产可以用“资源类型”来定义，这样一来，数字资产就像资源一样，满足线性逻辑中的一些特性：数字资产不能被复制；数字资产不能凭空消失。

(3) 不支持动态委派：动态委派是一种非常灵活的语言机制。如果程序在运行之前，就能知道它到底都调用了哪个函数，那么这些函数调用是“静态”的，如果直到程序运行的时候，通过观察才能得知，则这个函数调用被称为是“动态”的。动态特性不利于程序的推理，也更容易出安全问题。在以太坊智能合约设计中就存在许多“动态特性”，比如支持函数指针做参数，合约做参数，delegatecall 等等。而在 Move 语言中，完全不支持任何形式的“动态委派”或者“动态特性”，所有的合约执行路径都能在编译的时候确定，然后可以进行非常充分的分析、验证。

Move 智能合约吸取了如以太坊智能合约中安全漏洞的经验和教训，在设计 Move 语言机制时加入了上述 3 种特性，一定程度上保护了 Move 智能合约管理的数字资产。但，Move 语言引入了一种管理数字资产的全新的数据结构——资源，资源是否会引起 Move 智能合约出现新的安全漏洞，这将是本文研究工作的重点。

2.3 智能合约模糊测试技术

2.3.1 模糊测试概述

智能合约模糊测试是一种通过向智能合约程序提供随机的测试用例并监控智能合约运行过程中的异常来检测智能合约中是否存在漏洞的方法。在软件测试中，测试预言是一种用于确定测试是否通过或失败的机制 [46]。针对智能合约模糊测试，测试预言将被测智能合约的实际输出（对于给定的测试用例输入）与被测智能合约的预期输出进行比较以完成测试判定。

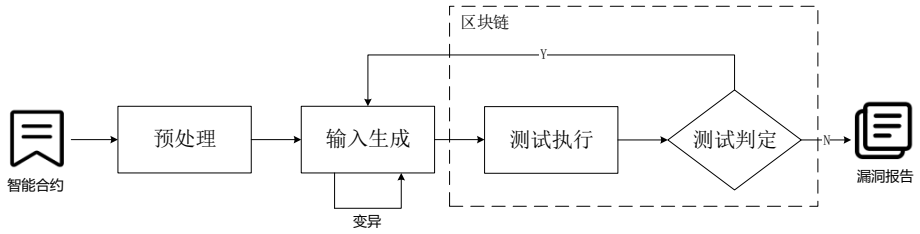

图 2.8: 智能合约模糊测试流程图

一般来说，智能合约模糊测试的流程如图2.8所示，大致可以分为预处理、输入生成、测试执行和测试判定 4 个阶段。

(1) 预处理：该阶段主要工作是获取待测智能合约的相关信息 (包括函数 ABI 签名、智能合约字节码等)，同时准备插桩等技术手段监测智能合约在模糊测试过程中的运行状态。

(2) 输入生成：该阶段主要工作是根据预处理阶段获取的智能合约信息确定测试输入对象，接着制定模糊测试输入生成算法，用于生成大量的符合待测智能合约函数输入规范的合法测试输入。模糊测试输入算法中包含变异策略，用于对随机产生的测试输入进行变异，以生成更多样的测试输入。

(3) 测试执行：该阶段主要工作是在区块链上发起交易，利用输入生成阶段生成的合法输入随机调用待测智能合约中的函数以启动模糊测试过程。

(4) 测试判定：该阶段主要工作是使用测试预言判定待测智能合约在测试过程中是否出现异常，如有异常出现，则记录对应的测试输入及漏洞相关信息。

已有的智能合约模糊测试工作基本都包含这 4 个阶段，只是在各个阶段的研究内容有所不同。在智能合约模糊测试工作中，测试输入的生成和测试预言的制定是比较重要的研究内容。测试输入的质量关系到模糊测试对智能合约代码的覆盖情况。测试预言的质量关系到模糊测试的漏洞检测能力，即能否尽可能地测试出智能合约中包含的所有类型的漏洞。

2.3.2 模糊测试方法

根据对程序源代码的分析程度和依赖程度，模糊器可分为黑盒模糊测试、白盒模糊测试和灰盒模糊测试。

（1）黑盒模糊测试

“黑盒”[47] 是形容看不见待测程序内部状态，只能看见待测程序输入和输出状态的一种术语。黑盒模糊测试也被称为输入输出驱动 (IO-driven) 的测试，或

者是数据驱动 (data-driven) 的测试 [48]。黑盒模糊测试将待测程序看作一个黑匣子，只能获得程序输入数据格式等信息。在测试过程中, 黑盒模糊测试不关心待测程序的执行状态, 只通过待测程序的输出数据来判断程序执行是否出现异常。

黑盒模糊测试有一定的优势，首先，待测智能合约的代码不一定被项目方开源，此时可以运用黑盒模糊测试对其进行测试；其次，黑盒模糊测试工具往往比较简单，因而检测速度也较快。但黑盒模糊测试也存在一些问题，其中最主要的问题是黑盒模糊测试工具无法获取待测程序的内部结构信息, 会生成大量无效输入数据, 导致无法有效检测出待测程序中的深层次漏洞。

（2）白盒模糊测试

相较于黑盒模糊测试，白盒模糊测试 [49] 通过符号执行等技术分析待测程序的内部结构，并利用在执行待测程序时收集的信息来生成测试用例。借助内部信息的丰富性，白盒模糊测试能够生成高质量的输入数据，从而解决了黑盒模糊测试存在的测试覆盖率低以及深层漏洞检测能力低等问题。但白盒模糊测试无法对不开源的智能合约进行测试，而且面对较为复杂的智能合约，符号执行一方面可能存在路径爆炸问题，另一方面进行全面的程序分析会导致资源消耗过高，进而影响模糊测试的整体效率。

（3）灰盒模糊测试

灰盒模糊测试 [50] 是一种介于黑盒模糊测试和白盒模糊测试之间的中间方法, 它只能获得智能合约中的部分内部信息。灰盒模糊测试不会推理待测程序的完整语义，而是对待测程序执行轻量级静态分析或收集待测程序的动态执行信息。灰盒模糊测试利用近似的、不完美的信息来尽可能多的执行测试输入的同时提高模糊测试的速度。

相较于黑盒模糊测试和白盒模糊测试，灰盒模糊测试的设计较为灵活，在对待测智能合约进行分析获得的内部信息较多时，灰盒模糊测试则更偏向于白盒模糊测试，反之则更偏向于黑盒模糊测试。因此，灰盒模糊测试能够在检测能力和资源消耗之间寻找一个平衡点，获得较好的检测效果。

综上考虑，本文研究的基于资源不变量的 Move 智能合约模糊测试方法采取灰盒模糊测试的方法，在对待测 Move 智能合约进行分析时，除了获取函数信息外，还会收集测试脚本执行日志信息来指导函数输入变异。

2.4 本章小结

本章主要介绍了系统中涉及到的 Libra 区块链、Move 智能合约以及模糊测试的基本知识。首先，介绍了该 Move 智能合约模糊测试系统的 Move 智能合约

运行平台——Libra 区块链，着重阐述了与 Move 智能合约密切相关的 Libra 区块链逻辑数据模型。然后，介绍了 Move 智能合约的两种类型：模块和脚本，阐述了两者的具体内容和区别，并给出了相应的示例代码。最后，介绍了智能合约模糊测试的基本工作流程。同时，介绍了模糊测试的三种方法，结合三者的优缺点引出本文模糊测试方法的类别。

第三章　基于资源不变量的 Move 智能合约模糊测试方法

本章主要介绍基于资源不变量的 Move 智能合约模糊测试方法。首先，引言部分通过分析传统区块链智能合约漏洞，阐述 Move 智能合约可能存在的问题以及拟采用的测试方法。接着，对 Move 智能合约漏洞类型进行分析，具体阐述 4 类 Move 智能合约漏洞。最后，给出资源不变量的定义并介绍 4 类漏洞的测试预言，并结合 Move 智能合约模糊测试框架图对本文方法的各阶段进行了具体介绍。

3.1　引言

区块链中智能合约的出现使得用户可以方便地创建各类去中心化的应用程序，极大地提升了区块链应用的灵活性，但与此同时，也增加了风险性。区块链智能合约层的风险主要来源于智能合约编写中产生的安全漏洞。一方面，智能合约的产生比较灵活，只要具备一定的计算机基础，任何人都可以创建或者调用智能合约。但是，由于部分编码人员缺乏对底层运行机制和原理的深入理解，智能合约灵活的产生方式容易引起安全问题。另一方面，由于区块链的不可篡改性，当部署后的智能合约出现安全漏洞时，用户无法通过打补丁或者更新的方式来修补漏洞。此外，智能合约的使用往往和数字资产等经济利益息息相关，因此智能合约中的漏洞造成的损失也更加惨重。

当前针对智能合约漏洞类型的研究主要集中于以太坊智能合约 [51][52]。以太坊智能合约安全漏洞可以从 Solidity 代码层、EVM 执行层和区块链系统层进行分类 [53]。从 Solidity 代码层的角度，漏洞主要存在代码重入漏洞、整数溢出漏洞、异常处理漏洞等类型；从 EVM 执行层的角度，漏洞主要存在短地址漏洞、以太丢失漏洞、Tx.Origin 漏洞等类型；从区块链系统层的角度，漏洞主要存在时间戳依赖漏洞、交易顺序依赖漏洞等类型。

与以太坊智能合约不同，Move 智能合约编程语言具有静态类型系统和不支持动态委派的特性，即 Move 语言在底层实现方面就完全杜绝了溢出问题，同时其不支持任何形式的“动态委派”，因此在 Move 智能合约中不会因为整数溢出导致数值方面的错误，也不存在代码重入、委托调用之类的漏洞。然而，Move 语言中引入了一类用于管理数字资产的新的数据类型——资源（Resources），错误的编码可能导致 Move 智能合约在资源权限和使用过程中存在新的问题，进而对数字资产的管理和使用产生影响。

通过第一章对智能合约安全分析国内外研究现状的分析可知，目前智能合约漏洞挖掘存在形式化验证、符号执行、模糊测试等技术。通过形式化验证的方法保障智能合约的安全性固然好，但是也存在一定的局限性。在智能合约的形式化验证过程中，需要编码人员分析智能合约并描述其规范，门槛较高且需要一定的学习成本。符号执行存在路径爆炸的问题，可能导致模糊测试的资源消耗和时间消耗过高。模糊测试是常用的智能合约动态测试技术，具有良好的适用性和可扩展性。使用模糊测试对智能合约进行安全分析时，通常首先利用随机引擎生成大量的随机数据，构成可执行交易；然后参考测试结果的反馈，随机引擎动态调整生成的数据，从而探索尽可能多的智能合约状态空间；最后基于测试预言分析每一笔交易的执行结果，检测是否存在安全漏洞。模糊测试技术已被应用于以太坊智能合约测试中，能够模拟智能合约实际的执行过程，具有较低的误报率等优点。因此本文采用模糊测试的技术来对 Move 智能合约进行漏洞检测。

3.2 Move 智能合约漏洞类型分析

Move 智能合约采用“资源是一等公民”编程规则，含义是在编程的时候首要考虑的被编程对象是资源。Move 智能合约中引入了资源这一数据类型来管理用户的数字资产，资源类型的潜在问题可以分为在资源权限和资源使用两方面出现问题。资源权限指哪些用户可以使用智能合约拥有者定义的资源，可能存在错误使用 signer，函数可见性声明错误等漏洞。资源使用指智能合约拥有者编码的资源处理逻辑，可能出现资源的意外销毁、错误的资源修改等漏洞。

3.2.1 错误使用 signer

在以太坊中交易的发起方使用 msg.sender 进行表示，而在 Libra 中则通过 signer 类型获取，如果没有正确的使用 signer 类型则可能造成漏洞的产生。

如果不使用 signer 作为参数，则存在函数调用者越权的风险。Libra 中的数据类型只有资源可以被持久化的存储在区块链上，而资源只能被其拥有者操作，这些操作是通过不同的函数定义实现的。如果函数的参数类型定义错误可能产生意外的漏洞，其中一种情形就是用地址类型 address 代替 signer 类型作为参数。地址类型的参数对应输入为一个链上地址，而 signer 则会自动绑定为交易的发起方，无法通过传参进行指定。使用 signer 的好处是，定义的 public 函数能够被任意地址调用的同时，只能接受调用者作为参数传入，因此只能操作调用者所有的资源。如果使用地址类型 address 替代 signer，则输入不同的地址参数将可能使得该地址下的资源被任意操作。在图3.1所示代码中，第 2 行中参数 account

使用了地址类型 address 进行传参，第 6 行对该地址下的资源 Token 进行了扣除，并在第 7 行作为返回值返回该部分资源，因此任意账户都可以通过调用函数 withdraw_from_address 对参数 account 下的 Token 进行提款操作。

```
1 public fun withdraw_from_address( // withdraw some Token from the account
2     account: address,
3     amount: u64
4     ) :Token acquires Token
5     {
6         borrow_global_mut<Token>(account).value -= amount;
7         Token{value: amount}
8     }
```

图 3.1: 错误使用 signer 漏洞代码示例图

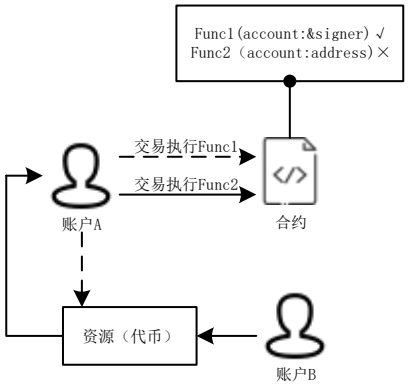

图 3.2: 错误使用 signer 漏洞触发示例图

如图3.2所示是错误使用 signer 漏洞触发示例图。合约中的函数 Func1 正确使用了 signer，不包含漏洞，而函数 Func2 未正确使用 signer（包括未使用 signer 和未对 signer 进行检查）导致存在漏洞。如虚线箭头所示，当账户 A 通过脚本调用函数 Func1 时，其只能使用自己账户下的资源；但如实线箭头所示，当账户 A 通过脚本调用函数 Func2 时，其可以从账户 B 下取出一定的资源并存入自己账户下，从而导致账户 A 非法获得账户 B 的资源。

3.2.2 函数可见性声明错误

Libra 中采用的 Move 智能合约中可见性分为 public 和 private 两种，如果不设置关键字，那么可见性默认为 private。私有函数 private 只能被合约内部调用，

共有函数 public 可以被外部调用，错误的设置函数可见性则可能导致漏洞。如果不定义可见性，则会将函数设置为 private，从而无法被外部调用，使得合约部分功能无法实现；反之，将私有函数设置为 public 则可能导致函数被外部调用，发生意外的问题。在图3.3所示的代码中，函数 do_mint 的功能是铸造数值为 amount 的 GameCoin，并将其作为返回值返回，一般只有合约账户拥有铸币权，如果该函数被设置为 public 的可见性，则会导致其被外部调用，进而使得任意账户均可以进行铸币，导致 GameCoin 的超发。withdraw_from_signer 函数的功能是为交易发起方 account 从其地址下取出数量为 amount 的代币 Token。如果函数定义时函数未加 public 关键字，将会导致该函数不可以被外部调用，用户将无法将自己账户下的代币取出，造成贪婪合约（只进不出）。

```
1  public fun do_mint(amount: u64): GameCoin {
2      GameCoin {value: amount }
3  }
4
5  fun withdraw_from_signer(
6      account: &signer,
7      amount: u64
8      ): Token acquires Token
9      {
10         borrow_global_mut<Token>(Signer::address_of(account)).value -= amount;
11         Token{value: amount}
12     }
```

图 3.3: 函数可见性声明漏洞代码示例图

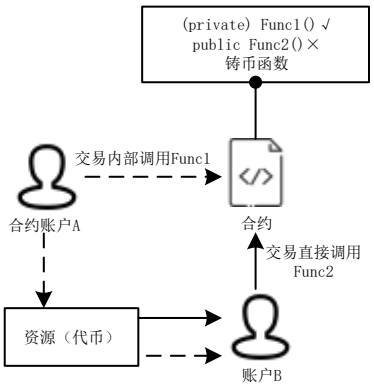

图 3.4: 函数可见性声明错误漏洞触发示例图

如图3.4所示是函数可见性声明错误漏洞触发示例图。图中函数的功能是铸币，即生成一定量的资源或代币。首先合约中的函数 Func1 使用了 private 关键字，不含有漏洞，而函数 Func2 将私有函数设置成了 public 导致存在漏洞。如虚线箭头所示，账户 B 可以请求合约账户 A 通过交易脚本调用函数 Func1 进行铸币，生成指定数量的资源并将资源存入账户 B 下；但如实线箭头所示，当账户 B 通过交易脚本调用函数 Func2 进行铸币时，其可以未经合约账户 A 允许就生成一定的资源并存入自己账户下，从而导致账户 B 非法获得资源。

3.2.3 资源的意外销毁

在 Move 智能合约中，资源类型满足“线性逻辑”，资源只能被移动，不能被拷贝，也不能凭空消失。资源的特性使得其被使用时与传统的数值具有差别，不谨慎的使用会产生问题。例如，在查询资源值时，需要以资源的引用作为输入，如果直接以资源为输入，在返回资源数值时会导致资源被 move 掉。在图3.5所示代码中，token_value 函数接受资源作为输入，并将资源的数值作为返回值。代码中未使用输入资源的引用作为参数，则在返回其数值时该资源因使用而被 move 掉，从而造成了该资源被销毁。

```
1 public fun token_value(check: Token): u64 {
2     let Token {value: amount} = check;
3     amount
4 }
```

图 3.5: 资源的意外销毁漏洞代码示例图

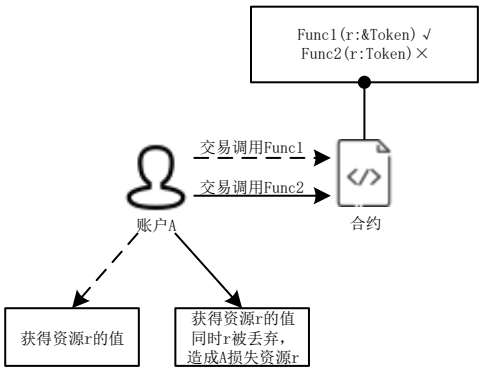

图 3.6: 资源的意外销毁漏洞触发示例图

如图3.6所示是资源的意外销毁漏洞触发示例图。首先合约中的函数 Func1

使用了资源的引用作为函数输入，不含有漏洞，而函数 Func2 直接将资源作为函数输入导致存在漏洞。如虚线箭头所示，当账户 A 通过交易脚本调用函数 Func1 查询资源 r 的值时，其能够获得所要查询的资源 r 的数值，且资源 r 在账户 A 下依旧存在；但如实线箭头所示，当账户 A 通过交易脚本调用函数 Func2 查询资源 r 的值时，其在获得资源 r 的值的同时，资源 r 被意外销毁，造成账户 A 损失了这一部分资源。

3.2.4 错误的资源修改

资源的特殊性在于其不能被复制且不能被无声名的丢弃，但是资源的计算过程仍然要依靠数值计算。鉴于这样的特性，在 Move 智能合约中增加、减少、转移资源时也存在一定的危险性。一方面，资源虽然不会凭空产生和消失，但是错误的数值计算会导致资源值在修改时产生错误，错误的数值计算过程导致资源凭空的增加或减少，或者错误的转移。在如图3.7所示代码中，deposit_to_address 函数能够被任意账户 account 调用，用来将其地址下 amount 数量的资源存到 to 地址下。然而资源的数量在修改时，第 6 行和第 7 行代码中的数量增减代码存在错误，错误的将减法编辑为了加法，将减法编辑为了加法，导致代币的转移存在错误，转出者的资源值增加而接收者的资源值减少。

```
1 fun deposit_to_address( // deposite some token from signer to address
2     account: &signer,
3     to: address,
4     amount: u64
5     ) acquires Token {
6         borrow_global_mut<Token>(Signer::address_of(account)).value += amount;
7         borrow_global_mut<Token>(to).value -= amount;
8     }
```

图 3.7: 错误的资源修改漏洞代码示例图

如图3.8所示是错误的资源修改漏洞触发示例图。首先合约中的函数 Func1 对资源修改的计算是无误的，不含有漏洞，而函数 Func2 错误的进行了资源修改导致存在漏洞。如虚线箭头所示，当账户 A 通过交易脚本调用函数 Func1 向账户 B 转账时，账户 A 减少的资源值等于账户 B 增加的资源值，转账成功；但如实线箭头所示，当账户 A 通过交易脚本调用函数 Func2 向账户 B 进行转账时，账户 A 的资源值增加，而账户 B 的资源值减少，或者账户 A 减少的资源值与账户 B 增加的资源值不相等，转账过程出现问题。

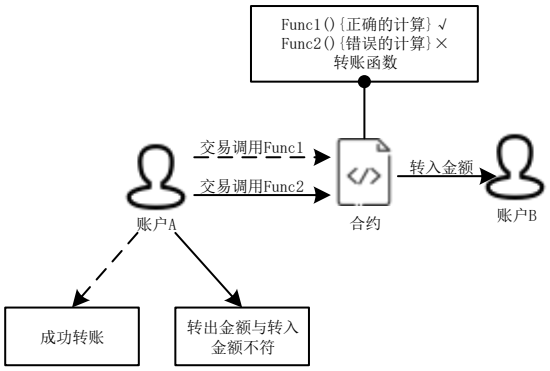

图 3.8: 资源的错误修改漏洞触发示例图

3.3 Move 智能合约模糊测试方法

3.3.1 Move 智能合约测试预言

针对资源类型相关的潜在漏洞，本节提出了一种测试预言——资源不变量，来检测 Move 智能合约中可能存在的与资源相关的安全漏洞。Move 语言使用资源编码行为类似于普通程序值的可编程资产：资源可以存储在数据结构中，作为参数传递给过程等等。Move 类型系统为资源提供了特殊的安全保证，即资源永远无法复制，只能移动。从这些特性中，可以引申资源类型的特殊属性——资源总量在交易前后保持不变，称其为资源不变量，其定义如下：

资源不变量：对于每个交易，如果交易过程中没有进行铸币操作和销毁操作，则交易前后，参与交易的账户资源总量保持不变。

即对于每个交易 $< s, r, v >$，有

$$(R(r)) + (R(s)) = 0$$

其中 $(x) = post(x) - pre(x)$，$pre(x)$ 和 $post(x)$ 分别表示交易前后变量 x 的值，s 表示交易发送方，r 表示交易接收方，v 表示交易发送的资源数量，$R(r)$ 和 $R(s)$ 表示交易接收方和发送方的资源数量。资源不变量的含义是，交易前后发送账户的资源金额变化量与接收账户资源金额变化量数值相等、符号相反，即交易前后，参与交易的账户资源总量不变。

根据 3.2 节对 Move 智能合约资源相关漏洞的分析，可以根据资源不变量为每一种漏洞设计如下测试预言。

(1) **错误使用 signer 漏洞**: 错误使用 signer 漏洞能够使用户调用非自己账户下的资源。此类漏洞的测试预言设计为当一个账户发起取款交易后，该账户

拥有的资源总量保持不变，但区块链上某一账户拥有的资源总量减少，且减少的数额等于取款交易的数额。

(2) **函数可见性声明错误**: 函数可见性声明错误能够使用户调用其自身无权限访问的合约函数，从而非法获取资源。此类漏洞的测试预言设计为当一个非合约所有者账户的用户账户发起交易后，区块链上所有账户拥有的资源总量增加，且增加的数额等于交易的数额。

(3) **资源的意外销毁漏洞**: 资源的意外销毁漏洞导致用户资源的意外损失。此类漏洞的测试预言设计为当一个账户发起查询资源数额交易后，该账户拥有的该类型资源的总量减少，且减少的数额等于被查资源的数额。

(4) **错误的资源修改漏洞**: 错误的资源修改漏洞导致用户间进行资源转账时，账户资源余额修改不一致。此类漏洞的测试预言设计为当资源转出账户发起转账交易后，区块链上所有账户的资源总量改变，且资源转出账户交易前的资源总量不等于交易后的资源总量加上转账金额，资源转入账户交易前的资源总量不等于交易后的资源总量减去转账金额。

3.3.2 Move 智能合约模糊测试框架

如图3.9所示是 Move 智能合约模糊测试框架图。模糊测试的流程如下：首先模糊测试器获取待测 Move 智能合约的函数信息，生成一组初始种子并加入到种子池中。在模糊测试循环的每次迭代过程中从种子池中随机挑选一组种子生成一组测试脚本。模糊测试器在 Libra 本地链上执行每个测试脚本，当测试脚本执行完成后，模糊测试器根据本文提出的测试预言对测试脚本执行结果进行验证。如果检测到违反测试预言，模糊测试器将停止本次测试，并报告漏洞以及提供对应的测试脚本，用于漏洞复现。否则，模糊测试器会利用变异策略对未发现漏洞的测试脚本所选取的种子进行变异，以生成用于下一次迭代的变异种子。

本文提出的 Move 智能合约模糊测试方法在 Libra 本地链下，通过执行脚本的形式对 Move 智能合约进行模糊测试，测试流程主要分为 4 个阶段，分别是种子生成阶段、测试执行阶段、测试判定阶段和种子变异阶段。

种子生成阶段。种子生成阶段用于构建函数种子池并生成测试脚本。该阶段主要包含两部分，分别是在 Libra 本地链上部署智能合约生成账本状态（Ledger state）S_{i-1} 以及解析智能合约生成交易 T_i，可以定义为：

$$(S_{i-1}, T_i)$$

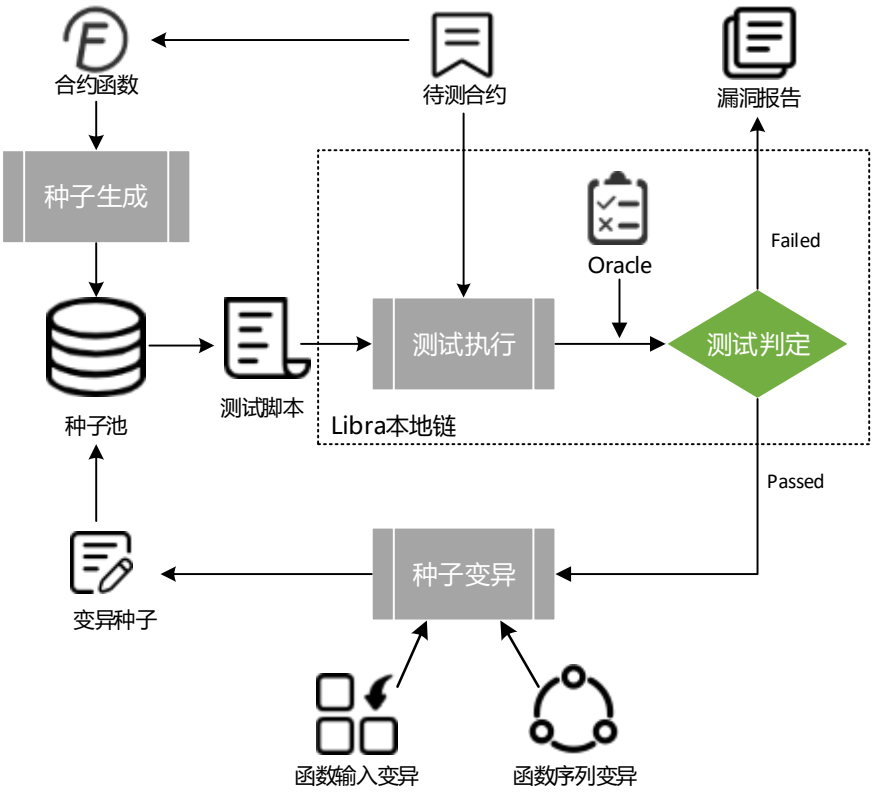

图 3.9: Move 智能合约模糊测试框架图

(1) 账本状态

账本状态 S_{i-1} 包含账户地址 S_K 及其对应的值 S_V，以键值对的形式存在。其中 i-1 表示版本号，值主要是已发布 Move 资源（resource）和模块（module）的集合，可以形式化为如下表示：

$$(< SK_0, SV_0 >, < SK_1, SV_1 >, ...)$$

可以看出，账本状态的实质是已经发布的 Move 模块和资源。在这个状态中，待测 Move 智能合约以及测试脚本中调用的模块和资源均已发布在 Libra 本地链中，这是包含测试脚本的交易能够顺利执行的基本条件。

(2) 交易

宏观上看，一个交易由一个交易脚本（用 Move 代码编写）和交易脚本的参数（例如，接收帐户地址或要发送的 LibraCoin 数量）组成。从结构上看，交易由发送方地址 A、发送方公钥 K、程序 P、Gas 价格 GP、最大 Gas

数量 GN、序列号 SN，可以形式化描述为一个六元组:

$$(A, K, P, GP, GN, SN)$$

其中程序 P 即交易脚本，主要包含了函数调用序列，可以定义为:

$$< f_0(p_0), f_1(p_1), , f_n(p_n) >$$

其中 f 为函数名，p 为函数输入具体参数，$f(p)$ 为一个种子。在生成测试脚本时，首先通过解析 Move 模块中的函数 f 以及对应函数的输入参数 p，接着采用随机策略生成函数参数输入数据，进而产生函数种子 $f(p)$，然后随机选取多个函数种子生成函数调用序列，最后根据函数调用序列依据 Move 语法生成测试脚本程序 P。

测试执行阶段。测试执行阶段用于配置 Libra 本地链测试环境并执行模糊测试，获得测试输出。

(1) 测试配置设定

测试脚本执行之前，需要先配置 Libra 本地链测试环境。这里的测试环境主要指前置的账本状态（已发布模块和资源），以及测试需要的 Libra 区块链账户等相关信息。

(2) 测试脚本执行

执行交易 T_i 将产生一个新的账本状态 S_i，以及交易输出 O_i（包括执行状态码 C 和事件列表 E）。执行状态代码记录执行交易的结果 (例如 success、error 等), 事件列表是执行交易输出的日志信息。测试输出由执行测试用例之后新的账本状态和交易输出组成，因此可以形式化为一个三元组：

$$(S_i, C, E)$$

测试判定阶段。测试判定阶段用于判定测试结果是否符合预期，本阶段的输出为判定结果。判定结果的形式为测试脚本以及该测试脚本是否通过测试预言，可以形式化为一个三元组：

$$(Testscript, Testoutput, Result)$$

若测试脚本未通过测试预言时，则输出漏洞报告以及测试脚本。若测试脚本通过测试预言，则进入种子变异阶段，生成新的变异种子，进而生成新的测试脚本用于模糊测试下一轮测试迭代。

种子变异阶段。种子变异阶段用于变异通过测试判定的测试脚本中选用的函数种子。本文提出的种子变异策略主要有两种：

(1) 函数输入变异

针对地址类型参数，方法直接从现有的 Libra 区块链账户地址列表中枚举；针对整数类型参数，方法首先根据当前种子中该参数的值确定 2 个范围（大于该值、小于该值），接着在这两个范围内选择随机数，以产生新的参数输入。

(2) 函数调用序列变异

对于给定的函数调用序列，通过随机插入或删除一个函数或交换两个函数调用顺序的方式进行变异。比如 Move 智能合约中取款交易的一个成功的函数调用序列是 *mint* → *withdraw* → *burn*, 初始生成的测试脚本中的调用序列可能不是如此，那么可以通过变异函数调用序列生成能够成功执行取款交易的测试脚本。

3.4 本章小结

本章介绍了基于资源不变量的 Move 智能合约模糊测试方法的研究工作。首先，从传统区块链智能合约存在的漏洞出发，分析 Move 智能合约的特性以及可能存在的安全漏洞。接着，对 Move 智能合约在资源权限和资源使用两方面可能存在的漏洞进行了分析。然后，给出了资源不变量的定义并介绍如何使用资源不变量检测这些漏洞。最后，详细描述了 Move 智能合约模糊测试框架。

第四章　需求分析与概要设计

本章介绍 Move 智能合约模糊测试系统的需求分析与概要设计，首先对系统进行需求分析，即通过对系统进行用例分析，确定系统的功能性需求和非功能性需求；然后根据需求分析的结果，对系统进行概要设计，给出系统的整体架构、视图和持久化存储表。

4.1　系统整体概述

本系统针对 Libra Move 智能合约进行深入的安全分析，并为开发者提供漏洞测试报告，帮助开发者推进 Move 智能合约优化完善。系统首先对开发者提供的 Move 智能合约进行分析，并提取合约函数相关信息（名称、参数类型、返回值等）构建种子库。然后系统从种子库中随机挑选种子构建初始测试脚本，并在 Libra 本地链上运行测试脚本进行测试。接着基于测试预言对测试脚本执行结果进行判定，对于通过判定的测试脚本选择合适的变异策略对脚本中的函数进行变异生成变异种子并加入种子库中；对于未通过判定的测试脚本记录测试执行信息及测试脚本信息用于复现漏洞。最后系统分析记录的脚本执行信息及对应的测试语言得出 Move 智能合约中存在的漏洞信息，并为开发者提供测试脚本执行报告。

为了提供上述功能，本系统采用 Vue 完成前端页面设计与实现，页面提供 Move 智能合约文件上传，测试脚本生成，模糊测试执行，测试结果查看等可交互操作支持，并基于 Spring Boot 框架实现后端数据传输、测试脚本执行和数据库访问等服务。系统采用 MySQL 存储合约函数信息和 Libra 区块链账户信息等持久化数据。

4.2　需求分析

4.2.1　用例分析

图4.1给出了系统整体用例图。用户在系统上可以上传 Move 智能合约文件，配置 Libra 本地链网络，创建并执行模糊测试任务，查看和下载测试结果。其中，用户每次上传 Move 智能合约文件，都会触发合约解析操作，对 Move 智能合约函数签名进行存储和展示。用户通过系统可以创建 Libra 本地链账户，并查看所创建账户的账户索引、账户地址、账户类型和账户余额等信息。用户生成初始测

试脚本后将执行生成的测试脚本，用户可以查看并下载测试脚本执行结果以及对应的执行日志。

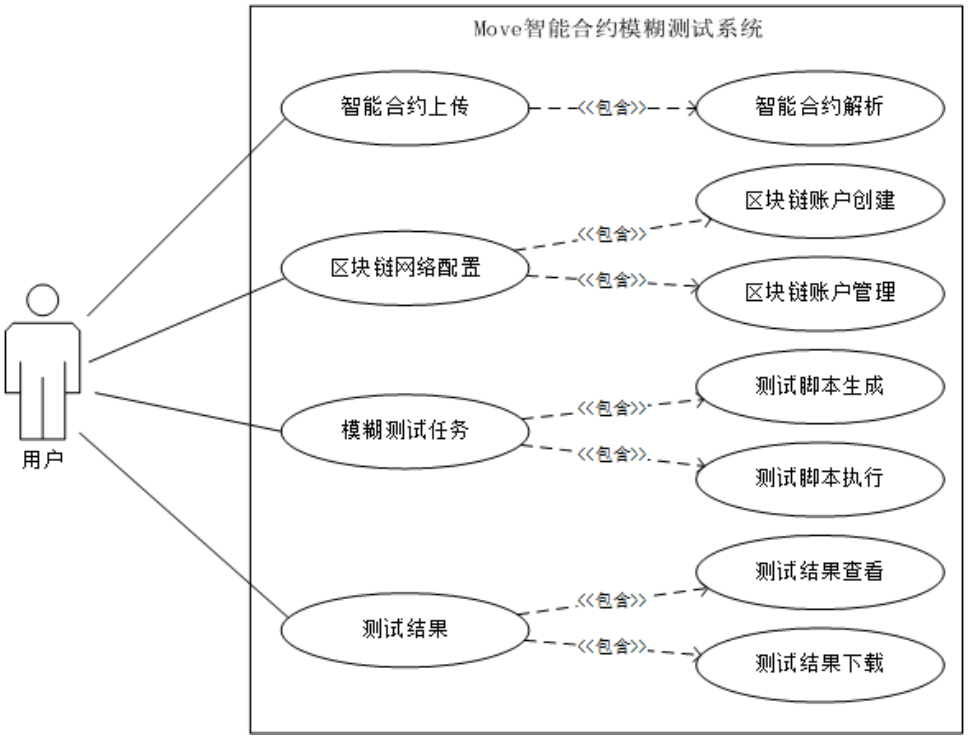

图 4.1: 系统用例图

用例图中的 Move 智能合约解析用例、区块链账户创建用例、测试脚本生成用例、测试脚本执行用例、测试结果查看用例、测试结果下载用例为主要用例，下面针对这些主要用例给出详细用例描述。

表 4.1: Move 智能合约解析用例描述

ID	UC1
名称	Move 智能合约解析
参与者	系统使用者
触发条件	用户点击“解析”
前置条件	用户上传 Move 智能合约文件
后置条件	智能合约函数信息保存到数据库中
优先级	高
正常流程	1. 用户点击” 解析” 按钮 2. 页面展示解析结果
异常流程	1a. 用户上传 move 智能合约文件非.move 文件，则解析失败

表 4.1展示了 Move 智能合约解析的用例描述。当用户完成登录后，可以上传并解析待测试的 Move 智能合约文件。用户上传的 Move 智能合约文件的后缀

名必须为.move，当用户上传成功后点击“解析”按钮即可对 Move 智能合约进行解析。解析完成后，页面将会展示用户所上传 Move 智能合约的函数相关信息，包括函数名称、函数参数类型、函数返回值类型等。

表 4.2: 区块链账户创建用例描述

ID	UC2
名称	区块链账户创建
参与者	系统使用者
触发条件	用户点击“创建”
前置条件	Move 智能合约解析完成
后置条件	区块链账户信息保存到数据库中
优先级	高
正常流程	1. 用户点击” 创建” 按钮 2. 页面展示账户信息
异常流程	1a. 后端创建账户时间过长或失败，提示账户创建失败

表 4.2展示了区块链账户创建的用例描述。当用户解析完所上传的 Move 智能合约后，即可创建用于模糊测试的区块链账户。用户点击“创建”按钮后，后端便尝试在 Libra 本地链上创建账户，创建时间受 Libra 本地链性能影响。当用于测试的区块链账户创建成功后，页面将会展示所创建区块链账户的相关信息，包括账户索引、账户地址、账户类型以及账户余额。

表 4.3: 测试脚本生成用例描述

ID	UC3
名称	测试脚本生成
参与者	系统使用者
触发条件	用户点击测试脚本生成
前置条件	测试用区块链账户创建完成
后置条件	无
优先级	高
正常流程	1. 用户填写测试脚本规模并选择测试账户地址并确认 2. 用户点击测试脚本生成按钮 3. 系统展示测试脚本列表 4. 用户查看测试脚本代码
异常流程	1a. 用户填写的测试脚本规模不在指定区间内，提示错误并重新填写

表 4.3展示了测试脚本生成的用例描述。测试脚本生成主要用于配置模糊测试所需的参数，包括测试脚本生成规模和测试区块链账户地址等，测试脚本生成规模代表使用种子库中的初始种子生成的初始测试脚本个数，测试区块链账户从 UC2 中生成的账户列表中选取一个区块链账户地址使用。当用户完成测试

脚本生成配置后，后端会根据测试脚本规模配置生成对应数目的初始测试脚本，页面将会展示所生成的初始测试脚本的名称，用户点击测试脚本名称可查看对应的测试脚本代码。

表 4.4: 测试脚本执行用例描述

ID	UC4
名称	测试脚本执行
参与者	系统使用者
触发条件	用户点击执行测试任务
前置条件	用户测试脚本生成完成
后置条件	无
优先级	高
正常流程	1. 用户点击执行测试任务按钮 2. 系统展示过渡动画 3. 系统展示测试过程中生成的所有测试脚本的名称、状态和日志信息
异常流程	3a. 测试任务执行失败，提示错误

表 4.4展示了测试脚本执行的用例描述。用户点击执行测试脚本按钮启动测试任务，系统后端在 Libra 本地链上开始执行初始测试脚本，并进行测试判定、种子变异等一系列操作，系统前端展示过渡动画。测试脚本执行完成后，系统前端展示模糊测试过程中生成的所有测试脚本名称及对应的脚本执行状态，状态包括未执行、编译失败和执行成功。

表 4.5: 测试结果查看用例描述

ID	UC5
名称	测试结果查看
参与者	系统使用者
触发条件	用户点击某一测试脚本
前置条件	系统存在执行成功的测试脚本
后置条件	测试脚本执行结果保存到数据库中
优先级	高
正常流程	1. 用户点击测试脚本执行列表 2. 系统展示测试脚本执行列表 3. 用户点击某个测试脚本 4. 系统展示本测试脚本的代码与执行结果 5. 用户点击某测试脚本日志 6. 系统展示测试脚本执行日志信息
异常流程	无

表 4.5展示了测试结果查看的用例描述。在测试脚本执行列表中会展示所有执行的测试脚本。用户点击测试脚本执行列表中的某一测试脚本可以查看其源

代码和执行状态。用户点击某一测试脚本的日志选项，系统以弹窗形式展示所选测试脚本的执行日志信息，可以观察到测试脚本是否违反测试预言。

表 4.6: 测试结果下载用例描述

ID	UC6
名称	测试结果下载
参与者	系统使用者
触发条件	用户点击测试结果下载
前置条件	系统存在执行成功的测试脚本
后置条件	无
优先级	高
正常流程	1. 用户点击测试脚本执行列表 2. 系统展示测试脚本执行列表 3. 用户点击执行日志下载按钮 4. 系统开始下载本次测试脚本执行日志到本地 5. 用户点击某测试脚本下载按钮 6. 系统开始下载该测试脚本代码到本地
异常流程	无

表 4.6展示了测试结果下载的用例描述。测试脚本执行成功后即可下载测试结果。用户点击某一检测出漏洞的测试脚本的执行日志下载按钮，该测试脚本的执行日志将以 pdf 格式下载。测试脚本执行日志包含违反的测试预言，即 Move 智能合约存在的漏洞信息。用户点击测试脚本列表中某一测试脚本的下载按钮，系统将会把该测试脚本的源代码以 pdf 格式下载。

4.2.2 功能性需求

根据系统用例分析，将 Move 智能合约模糊测试系统的功能性需求可以分为智能合约解析、区块链账户生成、模糊测试执行、测试结果展示四部分。

智能合约解析。Move 智能合约测试人员将待测试 Move 智能合约上传到系统，并进行 Move 智能合约解析操作。智能合约解析是模糊测试任务的基础，通过解析获取 Move 智能合约函数相关信息并构建初始种子库，模糊测试需要选取初始种子生成初始测试脚本以开启测试。

区块链账户生成。Move 智能合约测试人员根据自身需求生成测试区块链账户，包括合约账户和用户账户。用户创建测试区块链账户后，系统后端在 Libra 本地链上的合约账户上编译并发布待测试 Move 智能合约。

模糊测试执行。模糊测试执行是本系统的核心功能，主要包括初始测试脚本生成、测试脚本执行、测试脚本种子变异、测试脚本结果判定四部分。用户开始执行模糊测试后系统开始在后端 Libra 本地链上执行测试脚本，测试完成后将

输出漏洞测试结果信息。

测试结果展示。Move 智能合约测试人员查看已成功执行的测试脚本结果。本系统将以列表和弹窗形式展示漏洞测试结果以及所有测试脚本执行结果和执行日志，同时支持所有测试结果的下载功能。

4.2.3 非功能性需求

本系统目的是帮助 Move 智能合约开发人员和 Move 智能合约测试人员测试分析 Libra Move 智能合约的质量。为了方便使用，本系统需要根据具体使用场景的需要，满足以下非功能性需求。

性能。为了保证前端用户操作的流畅性，需对系统前端各个页面规定最长响应时间。文件上传成功反馈响应时间不应高于 1 秒，系统执行模糊测试的时间对于简单 Move 智能合约不应超过 5 分钟，对于复杂 Move 智能合约不应超过 30 分钟，测试结果下载响应时间不应超过 1 秒。

易用性。目前完整的操作流程需要用户上传待测试 Move 智能合约、配置区块链网络、创建测试任务，流程相对复杂。因此需要保证页面的可交互性，优化整个操作流程提升用户体验。

可用性。由于 Libra 本地链编译执行测试脚本时间较长，测试失败需要重新开始测试，测试效率难以保证。因此需要降低测试任务执行失败的概率，提高系统可用性。

可维护性。Move 智能合约编程语言以及 Libra 区块链仍然处于不断更新的阶段，可能出现 Move 脚本编译或执行命令的修改或者区块链内部机制的变更。因此，需要保证系统的可维护性，保证系统持续可用。

可扩展性。随着对 Libra 区块链和 Move 智能合约的研究不断深入，Move 智能合约使用范围可能增加，内置的种子变异策略无法持续满足需求。因此，系统需要提供变异策略更新接口，保证良好的可扩展性。

4.3 总体设计

本节以上一节需求分析中总结出的功能性需求和非功能性需求为基础，对 Move 智能合约模糊测试系统进行了总体设计。首先，从宏观角度给出系统的架构设计，整体把握系统的层次。其次，给出系统的逻辑视图、进程视图、开发视图、物理视图，从多个角度对系统的设计进行描述。

4.3.1 系统架构

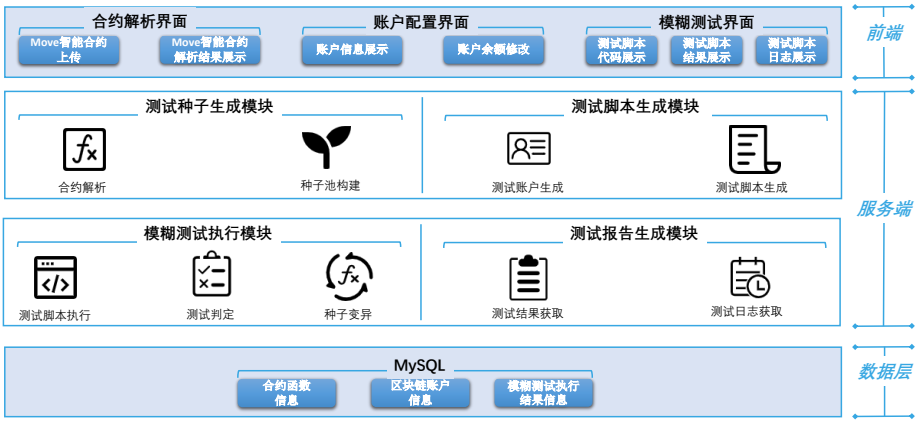

图 4.2: 系统整体架构图

Move 智能合约模糊测试系统的整体架构设计如图 4.2所示。下面将从前端交互、服务端处理和数据持久化三个部分出发对系统整体架构进行描述说明。

该 Move 智能合约模糊测试系统的前端交互使用目前较为主流的 Vue 前端框架，并设计了合约解析界面、账户配置界面和模糊测试结果展示界面以和用户进行交互，用户在各界面进行操作后，前端界面将向后端发送用户操作请求并且接收后端处理响应。合约解析界面的合约上传功能负责与服务端的测试种子生成模块进行交互，传输待测 Move 智能合约信息，合约解析结果展示功能将展示待测 Move 智能合约的解析结果；账户配置界面的账户信息功能负责与服务端的测试脚本生成模块进行交互，传输测试用区块链账户信息；模糊测试结果展示界面的脚本代码展示功能负责与服务端的测试脚本生成模块进行交互，传输初始测试脚本信息用以展示；测试脚本结果展示功能和测试脚本日志展示功能与服务端的模糊测试执行模块和测试报告生成模块进行交互，告知服务端开始执行模糊测试，并将模糊测试过程中的测试脚本信息传输给前端用以展示。

该 Move 智能合约模糊测试系统的服务端使用 Spring Boot 框架进行搭建和开发，服务端向前端开放 RESTfulAPI，降低前端和服务端的耦合性。服务端由测试种子生成模块、测试脚本生成模块、模糊测试执行模块和测试报告生成模块构成，其中测试种子生成模块主要负责实现 Move 合约解析功能和种子池构建功能；测试脚本生成模块主要负责实现测试合约账户生成功能和初始测试脚本生成功能；模糊测试执行模块主要负责实现测试脚本执行功能和种子变异功能；测试报告展示模块主要负责实现测试结果展示功能和测试日志展示功能。

该 Move 智能合约模糊测试系统的数据持久化层使用 MySQL 数据库存储相关信息，主要包括测试过程中生成的区块链账户信息、解析出的待测 Move 智能

合约函数信息以及测试过程中生成的测试脚本数据等。

4.3.2 4+1 视图

本节采用 Philippe Kruchten 的“4+1”视图对本系统的总体设计进行进一步的描述说明。下面分别从逻辑视图、开发视图、进程视图以及物理视图四个角度给出本文的 Move 智能合约模糊测试系统的架构设计。

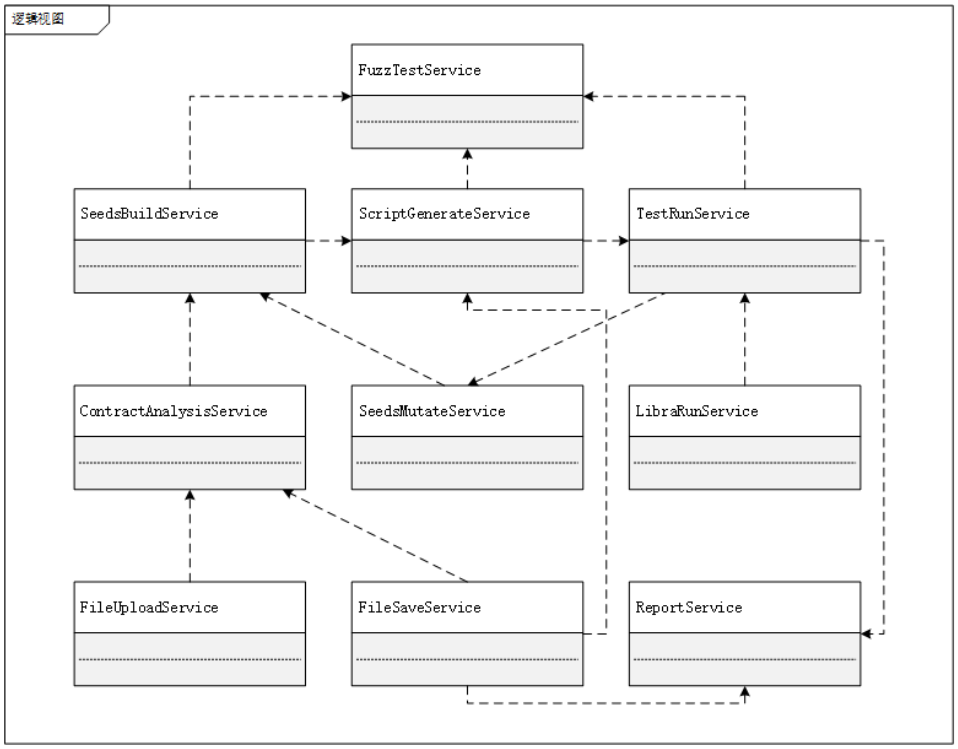

图 4.3: 逻辑视图

逻辑视图。逻辑视图通常是从用户的角度出发进行构建，因此可以按照面向对象的思路，将系统抽象为类，并采用 UML 类图作为逻辑视图的具体表示。该 Move 智能合约模糊测试系统的逻辑视图如图 4.3所示。系统中的主要服务可以抽象成模糊测试服务（FuzzTestService）、种子池构建服务（SeedsBuildService）、测试脚本生成服务（ScriptGenerateService）、测试执行服务（TestRunService）、Move 智能合约解析服务（ContractAnalysisService）、种子变异服务（SeedsMutateService）、Libra 区块链运行服务（LibraRunService）、文件上传服务（FileUploadService）、文件下载服务（FileSaveService）、报告生成服务（ReportService）。其中，模糊测试服务是本系统的核心，由种子池构建服务、测试脚本生成服务以及测试执行服务构成。种子池构建服务主要负责实现构建待测 Move 智能合约的函数种子池，从而为生成测试脚本提供服务。种子池构建服务还依赖于 Move 智能合

约解析服务和种子变异服务。Move 智能合约解析服务负责实现解析用户上传的待测 Move 智能合约，得到相应的合约函数信息。Move 智能合约解析服务依赖于文件上传服务和文件下载服务。种子变异服务负责实现根据模糊测试执行过程中测试脚本执行信息按照预定的变异策略对合约函数种子进行变异，并将其加入到种子池中，其依赖测试执行服务。测试脚本生成服务主要负责实现随机挑选种子池中的函数种子生成模糊测试脚本，提供给测试执行服务进行脚本执行。测试执行服务主要负责实现测试脚本执行以及对应的执行结果展示，其依赖 Libra 区块链运行服务和报告生成服务。Libra 区块链运行服务主要负责实现部署用户上传的 Move 智能合约，区块链账户生成、测试脚本编译等功能。文件上传服务负责实现待测 Move 智能合约的用户上传功能。文件下载服务主要负责实现智能合约解析结果文件下载、测试脚本下载、测试脚本执行日志下载等功能。报告生成服务主要负责实现模糊测试执行过程中每个测试脚本的执行结果和执行日志信息收集。

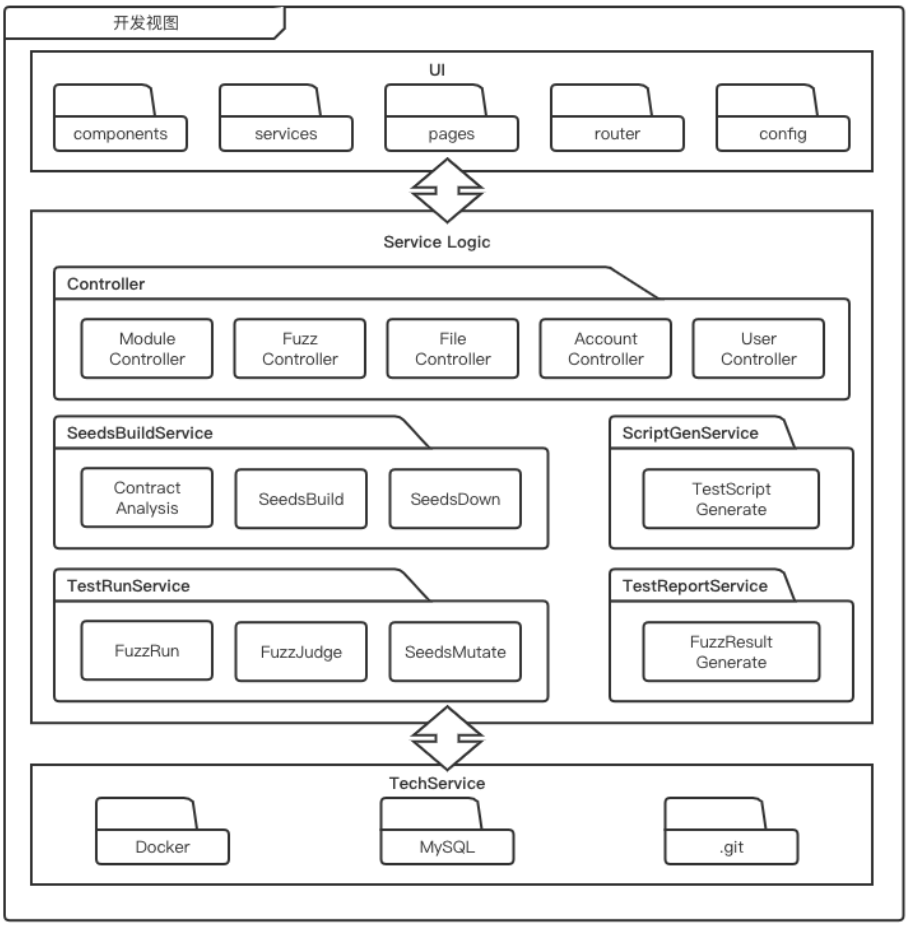

图 4.4: 开发视图

开发视图。开发视图描述了在开发环境中软件的静态组织结构，侧重于软件模块的组织和管理。该 Move 智能合约模糊测试系统的开发视图如图 4.4所示。

前端 UI 层主要包含 components、services、pages、router、config5 个文件夹。components 文件夹中包含前端页面组件的.vue 格式文件，前端页面组件主要包括侧边栏组件、页眉组件等。services 文件夹中包含 user.js 和 simulation_env.js 文件，用于存放用户信息管理和模糊测试的相关请求库和 API。pages 文件夹中包含整个前端各页面展示的.vue 格式文件，每个页面都有一个对应的.vue 文件。router 文件夹中包含了 config.js 文件，为系统前端配置了相应的路由规则。config 文件夹中包含了保存系统全局配置的.js 格式文件。

业务逻辑层 ServiceLogic 根据系统模块和功能划分，主要包含 Controller、SeedsBuildService、ScriptGenService、TestRunService 和 TestReportService5 个包。Controller 包负责接收系统前端用户请求并调用相应测服务进行处理。SeedsBuildService 包负责待测 Move 智能合约函数种子池构建，主要包含智能合约解析（ContractAnalysis）、种子池构建（SeedsBuild）和种子下载（SeedsDown）三个部分的代码。TestRunService 包负责模糊测试执行，主要包含测试脚本执行（FuzzRun）、测试判定（FuzzJudge）和种子变异（SeedsMutate）三个部分的代码。ScriptGenService 包负责测试脚本生成。TestReportService 包负责模糊测试测试脚本执行结果报告生成。

技术服务层 TechService 为系统使用的基础服务，包括使用 MySQL 数据库进行数据存储，使用 Docker 构建系统镜像，便于部署和运行，使用 Gitee 对系统代码进行管理。

进程视图。进程视图描述了系统在使用过程中各部分的通信方式。该 Move 智能合约模糊测试系统的进程视图如图4.5所示。前端在用户操作过程中通过与 Move 智能合约解析服务、测试脚本生成服务以及 Move 智能合约模糊测试服务进行交互，实现 Move 智能合约解析、测试脚本生成、模糊测试执行等功能。其中，Move 智能合约解析服务（ContractAnalysisServer）处理智能合约解析请求时与文件存储服务交互，从本地接受用户上传的 Move 智能合约，并支持将智能合约解析结果以文件形式保存到本地。测试脚本生成服务（ScriptGenServer）处理测试脚本生成请求时首先获取智能合约解析后生成的函数种子池，然后根据前端配置利用种子生成对应的测试脚本。Move 智能合约模糊测试服务（TestServer）处理模糊测试执行请求时首先通过 ScriptRunClient 编译执行已经生成的测试脚本，接着通过 OracleJudgeClient 运用测试预言对测试脚本执行结果进行判定，对没通过判定的测试脚本通过 ReportClient 整合测试结果和执行日志信息生成测试报告，对通过判定的测试脚本通过 MutateClient 进行函数种子变异并加入到种

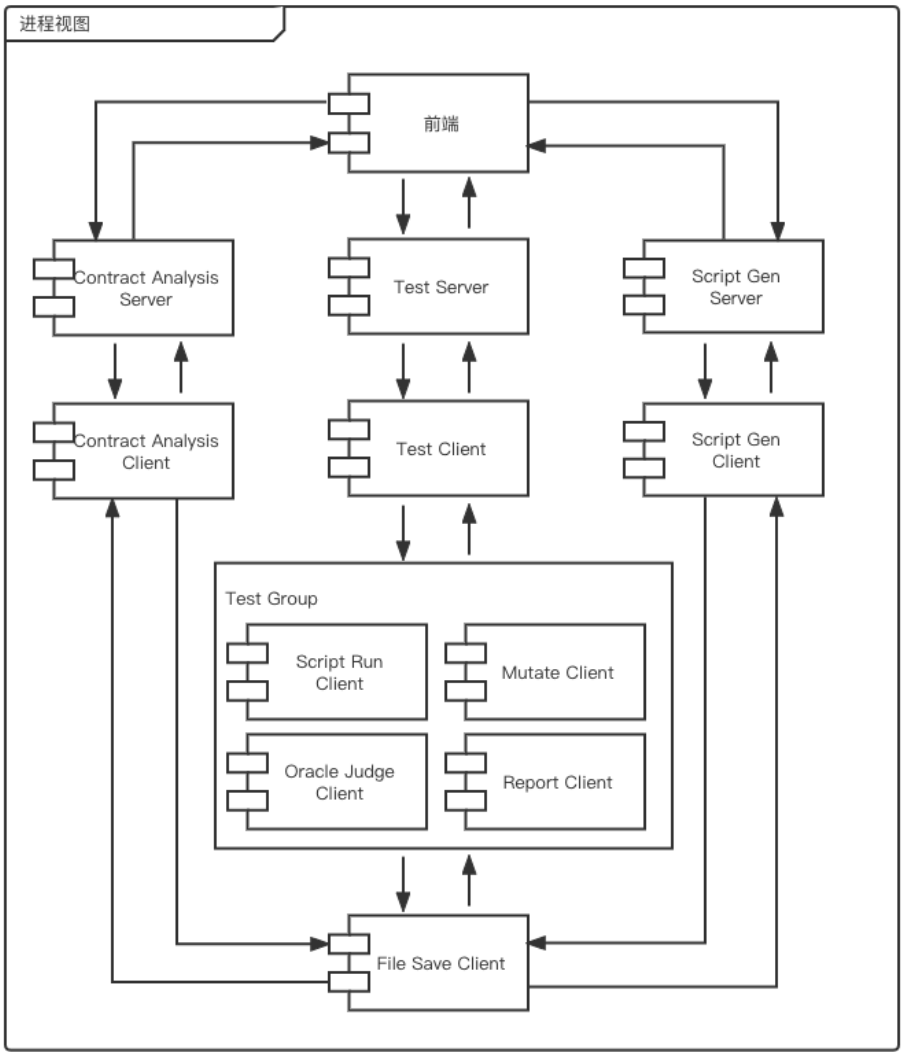

图 4.5: 进程视图

子池中，最后将模糊测试执行过程中所有测试脚本信息返回给前端。

物理视图。物理视图向运维人员展示了系统的部署设计以及部署后系统的运行方式，该 Move 智能合约模糊测试系统的物理视图如图4.6所示。图中每一个方块代表一个物理机，分别是用户客户端、系统 Web 服务器、系统后端服务器以及数据库服务器。用户在自己电脑上通过 Web 浏览器访问该系统并向 Web 服务器发送 HTTP 请求，该 Move 智能合约模糊测试系统的 Web 服务器接收请求后向系统后端服务器发起访问请求。后端分为业务请求和数据存储，业务请求负责响应业务逻辑，包括 Move 智能合约解析、测试脚本生成以及 Move 智能合约模糊测试等业务。数据库服务器负责用户信息、Move 智能合约函数信息、测试脚本执行信息等数据的存储。

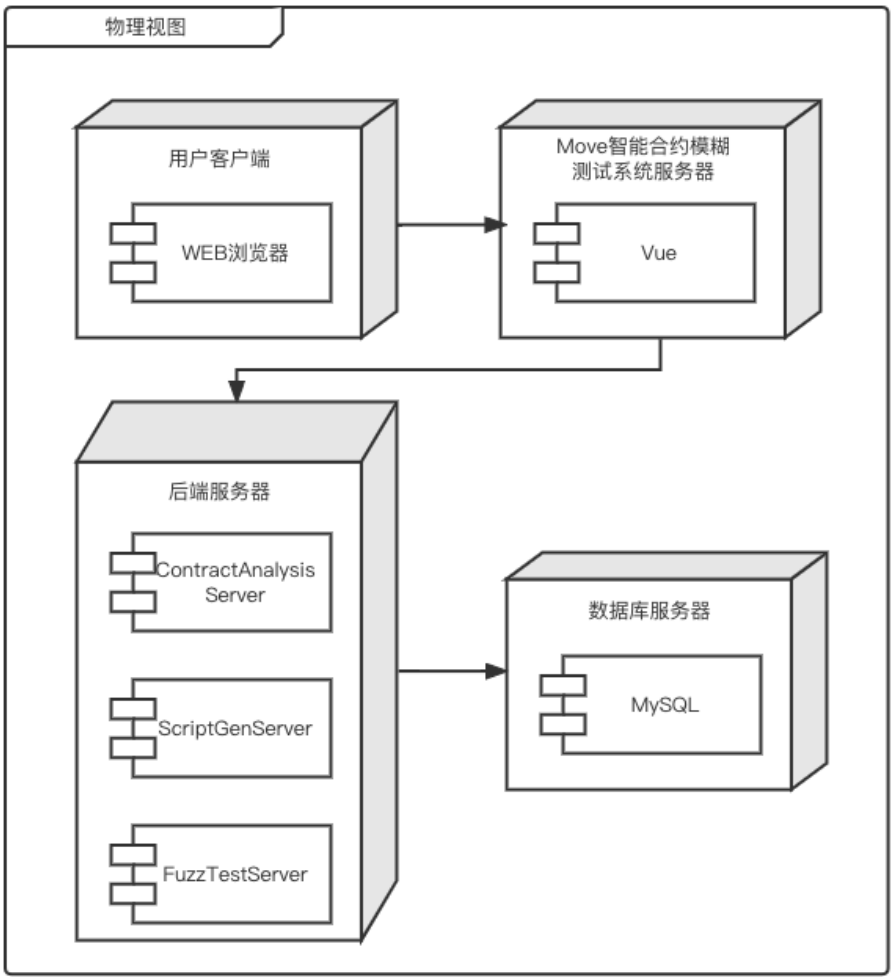

图 4.6: 物理视图

4.3.3 持久化设计

该 Move 智能合约模糊测试系统的持久化设计主要包括待测 Move 智能合约的函数信息、测试用区块链账户信息和模糊测试执行结果信息。下面给出上述信息的数据库表结构并一一说明。

Move 智能合约信息表如表4.7所示。Move 智能合约信息表主要存储用户上传的 Move 智能合约的基本信息，包括 Move 智能合约唯一标识 id、Move 智能合约名称 contract_name 以及该 Move 智能合约中函数个数 function_count。

表 4.7: Move 智能合约信息表

字段名	类型	描述
id	int	Move 智能合约唯一标识 ID
contract_name	varchar	Move 智能合约名称
function_count	varchar	Move 智能合约函数个数

表 4.8: Move 智能合约函数信息表

字段名	类型	描述
id	int	主键
contract_id	int	所属 Move 智能合约
func_name	varchar	函数名称
return_type	varchar	函数返回类型
param_name	varchar	函数参数名称
param_type	varchar	函数参数类型

Move 智能合约函数信息表如表4.8所示。Move 智能合约函数信息表主要存储 Move 智能合约的函数信息，使用 Move 智能合约信息表中的 Move 智能合约唯一标识 ID 作为 conract_id 标识所属智能合约。此外，Move 智能合约函数信息表还包括函数名称 func_name、函数返回类型 return_type、函数参数名称 param_name 以及函数参数类型 param_type。

表 4.9: 区块链账户信息表

字段名	类型	描述
id	int	账户索引，唯一标识
account_address	varchar	区块链账户地址
account_type	varchar	区块链账户类型
trans_id	int	区块链账户交易序列号
account_balance	int	区块链账户余额

区块链账户信息表如表4.9所示。区块链账户信息表主要存储部署 Move 智能合约以及执行测试脚本的 Libra 本地链账户信息，包括唯一标识区块链账户的账户索引 id、区块链账户地址 account_address、区块链账户类型 account_type、区块链账户交易序列号 trans_id 以及区块链账户余额 account_balance。

模糊测试执行结果信息表如表4.10所示。模糊测试执行结果信息表主要存储模糊测试过程中各测试脚本的执行结果信息，包括测试编号 id、测试脚本名称 script_name、测试脚本执行输出 output、测试脚本违反的测试预言 oracle、漏洞类型 vul_type 以及测试脚本执行日志信息 log_info。

表 4.10: 模糊测试执行结果信息表

字段名	类型	描述
id	int	测试编号，唯一标识
script_name	varchar	测试脚本名称
output	varchar	测试脚本执行输出
oracle	varchar	测试脚本违反的测试预言
vul_type	varchar	漏洞类型
log_info	varchar	测试脚本执行日志信息

4.4 本章小结

本章首先对基于资源不变量的 Move 智能合约模糊测试系统进行用例分析，并根据用例分析的结果给出系统的功能性需求和非功能性需求，并将系统划分为测试种子生成模块，测试脚本生成模块，模糊测试执行模块以及测试报告生成模块。接着对系统进行总计设计，描述了系统的整体架构，并从逻辑视图，开发视图，进程视图和物理视图对系统的开发及部署进行了详细描述，并给出了数据持久化设计。

第五章　系统实现与测试

5.1　系统详细设计与实现

本节对该 Move 智能合约模糊测试系统的主要模块进行了详细设计与实现，包括测试种子生成模块、测试脚本生成模块、模糊测试执行模块以及测试报告生成模块，其中测试脚本生成模块和模糊测试执行模块为本系统中核心的两个功能模块。

5.1.1　测试种子生成模块设计与实现

测试种子生成模块主要负责 Move 模糊测试的准备工作，包括 Move 智能合约解析和 Move 智能合约函数种子池构建两个功能。Move 智能合约解析主要负责解析用户上传的 Move 智能合约中函数相关信息，包括函数名称、函数返回值类型、函数参数个数以及每个参数的类型。用户首先在前端文件上传框中上传需要进行测试的 Move 智能合约，接着服务端获取前端返回的 Move 智能合约并进行解析，然后将解析结果保存在数据库中并将解析结果返回给前端向用户进行展示。Move 智能合约函数种子池构建主要负责根据 Move 智能合约解析结果和漏洞类型初始化函数种子池，即一个漏洞类型对应一个容易触发漏洞的函数集合，此阶段在服务端成功解析 Move 智能合约之后进行。

测试种子生成模块的核心类图如图 5.1所示。其中 SeedsGenController 主要负责调用 ContractAnalysisService 进行 Move 智能合约解析、调用 SeedsBuildService 进行 Move 智能合约函数种子池构建。ContractAnalysisService 主要接收前端上传的 Move 智能合约，并负责解析 Move 智能合约获取智能合约中函数信息并将解析结果返回给前端。SeedsBuildService 负责初始化 Move 智能合约函数种子池以及根据模糊测试执行过程中测试脚本编译和执行日志进行函数种子变异，进而丰富函数种子池以成功测试出漏洞。

测试种子生成模块的活动图如图 5.2所示。用户通过浏览本地文件选择 Move 智能合约进行上传，后端成功接收前端请求后检查 Move 智能合约是否存在，如果 Move 智能合约存在则构建文件流对象读取 Move 智能合约内容，并返回给前端展示。接着用户在前端点击合约解析按钮后，前端向后端发起合约解析请求。后端接收请求后对 Move 智能合约中的函数进行解析，并根据解析结果构建函数种子池，同时向前端返回解析结果进行展示。用户可以点击解析结果按钮选择将 Move 智能合约解析结果以文件形式下载到本地。

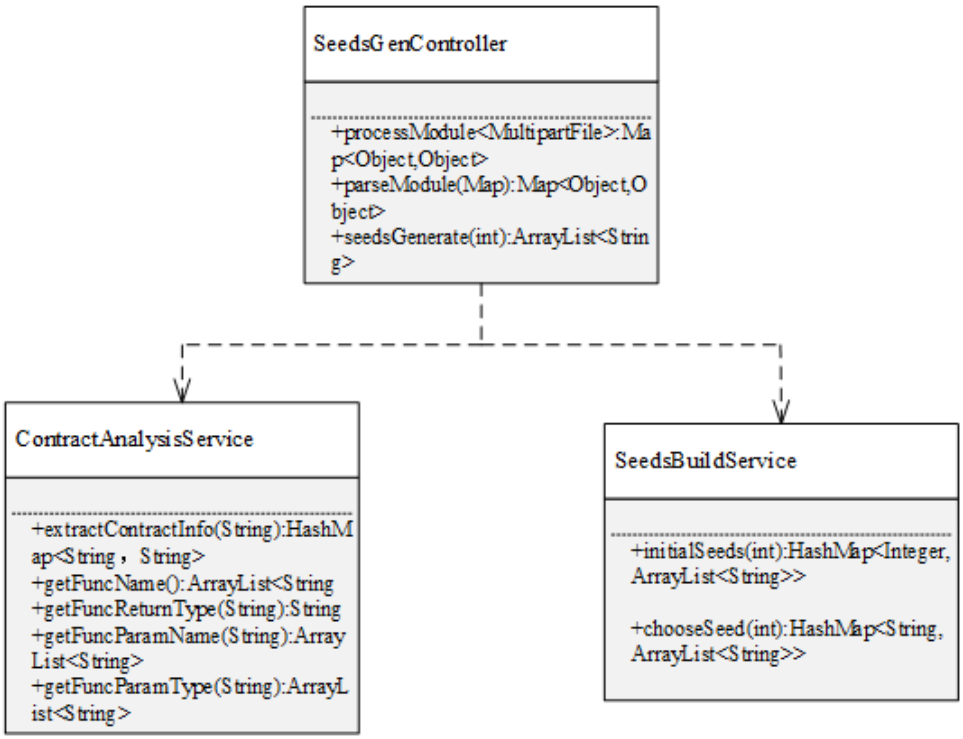

图 5.1: 测试种子生成模块类图

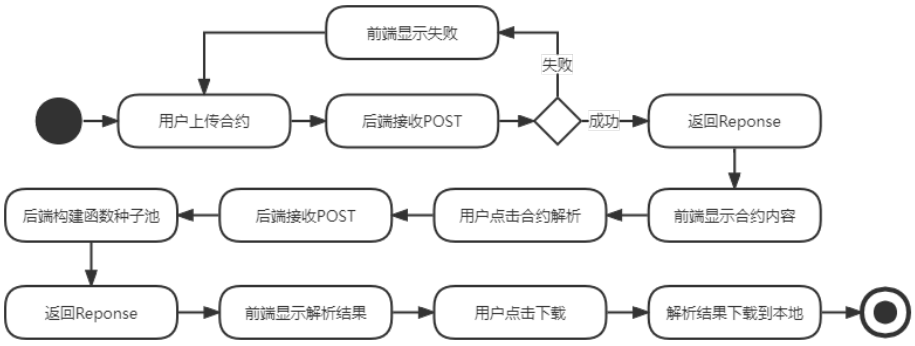

图 5.2: 测试种子生成模块活动图

图5.3所示代码展示了测试种子生成模块的关键代码。extractContractInfo()方法主要负责解析用户上传的 Move 智能合约。该方法接收合约文件路径作为参数。代码的 5-6 行负责解析 Move 智能合约部署地址，代码的 7-8 行负责解析 Move 智能合约中模块的名字，这两者用于测试脚本绑定相应的 Move 智能合约。代码的 9-28 行负责解析 Move 智能合约中每个函数的名称、返回值类型以及函数中各参数的类型，用于测试脚本生成正确的函数调用。

```
public static void extractContractInfo(String modulePath) {
    BufferedReader br = fileRead(modulePath);
    while ((line = br.readLine()) != null) {
        if (line.contains("address 0x") && !line.contains(")") && line.contains("{")) {
            FuzzInfo.moduleAddr = line.substring(line.indexOf("address 0x") + 8,
                line.indexOf("{")).trim();
        } else if (line.contains("module")) {
            FuzzInfo.moduleName = line.substring(line.indexOf("module") + 7,
                line.indexOf("{")).trim();
        } else if (line.contains("fun ")) {
            List<String> paraType = new ArrayList<>();
            List<String> paraName = new ArrayList<>();
            //解析函数名称
            String func_name = line.substring(line.indexOf("fun ") + 4,
                line.indexOf("("));
            //解析函数返回类型
            String func_return_type = "";
            if (line.contains("acquires")) {
                func_return_type = line.substring(line.indexOf(")") + 1,
                    line.indexOf("acquires")).replace(":", "").trim();
            } else {
                func_return_type = line.substring(line.indexOf(")") + 1,
                    line.indexOf("{")).replace(":", "").trim();
            }
            //解析函数各参数类型
            String[] params = line.substring(line.indexOf("(") + 1,
                line.indexOf(")")).split(",");
            for (int i = 0; i < params.length; i++) {
                if (params[i].equals("")) {
                    continue;
                }
                paraName.add(params[i].substring(0, params[i].indexOf(":")).trim());
                paraType.add(params[i].substring(params[i].indexOf(":") + 1).trim());
            }
        }
    }
}
```

图 5.3: 测试种子生成模块关键代码

5.1.2 测试脚本生成模块详细设计与实现

测试脚本生成模块主要负责 Move 智能合约模糊测试过程中测试脚本的生成，包括生成区块链账户和测试脚本生成两个功能。生成区块链账户主要负责调用 move-cli 命令在 Libra 本地链上创建一个区块链账户并且完成铸币操作，并在前端展示生成的区块链账户信息，包括账户索引、账户地址、账户类型以及账户余额等。测试脚本生成主要负责首先从 Move 智能合约函数种子池中随机挑选函数种子构建一个完整的函数调用，并结合 Move 智能合约解析出的模块地址和模块名称生成一个符合 Move 语言基本编程要求的测试脚本。

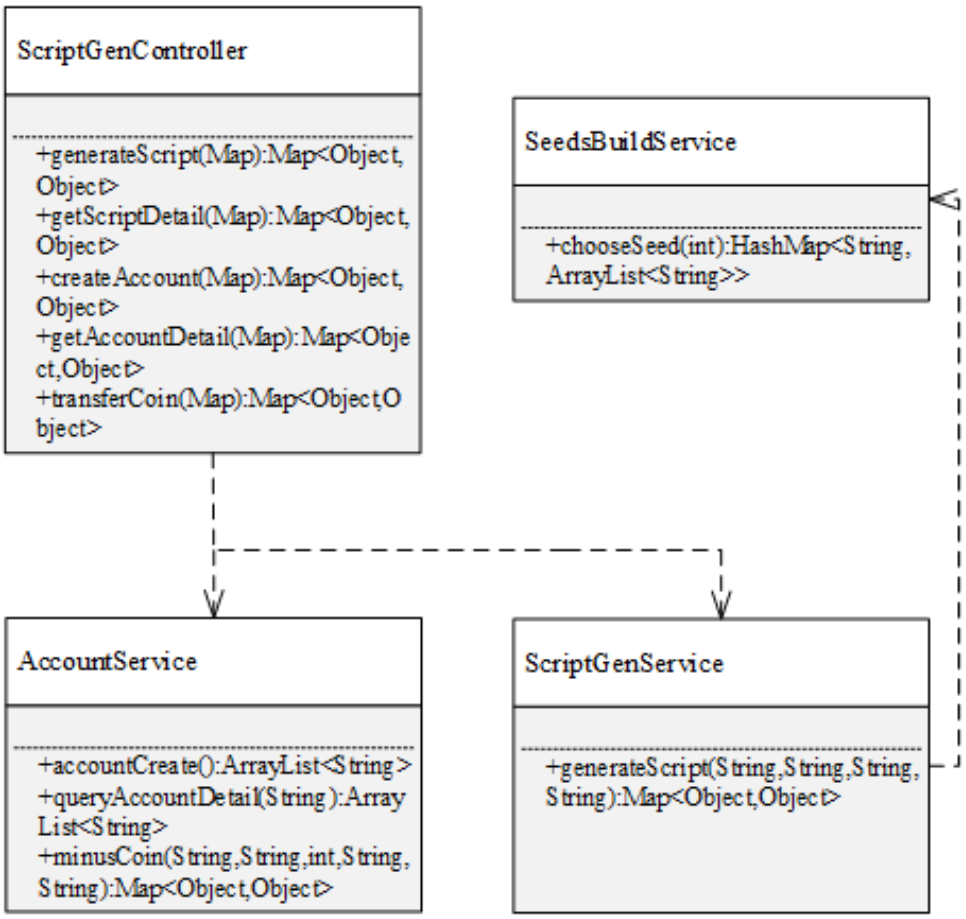

图 5.4: 测试脚本生成模块类图

测试脚本生成模块的核心类图如图 5.4所示。其中 ScriptGenController 主要负责调用 AccountService 进行区块链账户的创建、调用 ScriptGenService 进行 Move 智能合约测试脚本的生成。AccountService 负责处理前端发起的区块链账户创建请求、查询请求和余额修改请求，使用 accountCreate() 方法在 Libra 本地链上创建并初始化一个账户，使用 queryAccountDetail() 查询账户的具体各项信

息，使用 minusCoin() 方法完成区块链账户余额的修改。ScriptGenService 负责根据用户前端设置的初始测试脚本规模生成相应个数的初始测试脚本，并且在 Move 模糊测试过程中依据测试具体情况不断生成新的测试脚本。测试脚本的生成依赖于 SeedsBuildService 中的 chooseSeed() 方法挑选函数种子。

区块链账户生成功能的活动图如图 5.5所示。用户在前端点击创建账户按钮，后端成功接收请求后在 Libra 本地链上创建并初始化一个账户，后端成功创建区块链账户后，向前端返回账户信息供用户查看。用户可以点击账户详情按钮向后端发起查询请求，后端接收到查询请求后调用 Libra 本地链接口查询所创建区块链账户的具体信息，并返回给前端以弹出框形式向用户展示。用户还可以点击修改余额按钮对区块链账户的余额进行修改，后端接收到余额修改请求后，在 Libra 本地链上以交易的方式修改账户余额，并将修改结果返回前端。

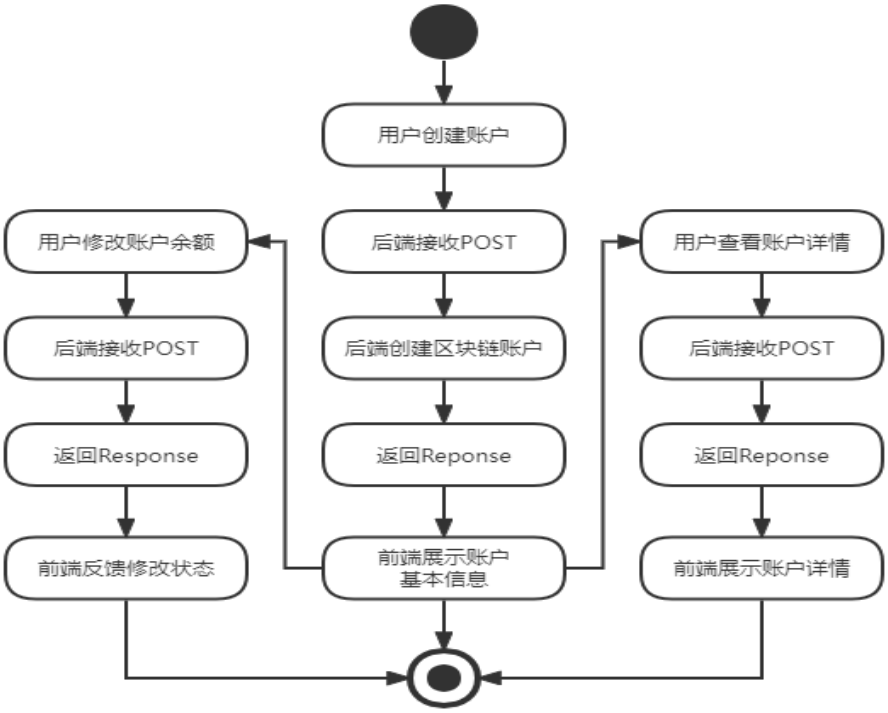

图 5.5: 区块链账户生成活动图

测试脚本生成功能的活动图如图 5.6所示。用户在前端通过文本框设置初始测试脚本规模，然后点击脚本生成按钮，后端成功接收请求后从函数种子池中随机挑选种子并调用相应的方法生成测试脚本。在模糊测试执行过程中后端会根据测试情况选择是否从种子池中选择种子生成新的测试脚本。

如图5.7所示代码展示了测试脚本生成模块中区块链账户生成功能的关键代

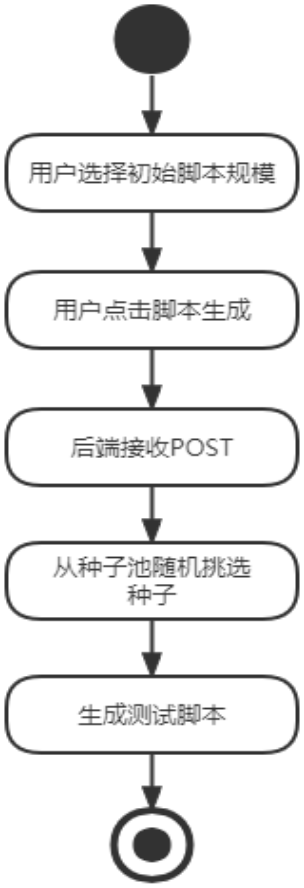

图 5.6: 测试脚本生成功能活动图

码。accountCreat() 方法负责在 Libra 本地链上创建并初始化一个区块链账户。代码的 1-16 行负责创建区块链账户，对区块链账户进行铸币操作以及提取所创建区块链账户的账户索引和账户地址两个信息。代码的 3-4 行负责调用 shell 脚本在 Libra 本地链上创建区块链账户，shell 脚本的内容如代码 19-21 行所示。代码的 24-33 行为 python 代码，负责从区块链账户创建日志中提取账户索引和账户地址信息并写入本地 txt 文件供代码第 7 行读取使用。

如图5.8所示代码展示了测试脚本生成模块中测试脚本生成功能的关键代码。choosSeed() 方法负责从函数种子池中随机挑选种子，用于生成测试脚本。代码的 2-7 行负责随机挑选指定数量的函数种子。代码的 10-11 行负责打乱函数种子的先后顺序，使得挑选出来的函数种子因调用顺序不同而产生不同的测试脚本。代码的 12-22 行负责根据函数参数类型的不同，生成不同的随机测试输入值。最终，经过处理后的函数种子遵守 Move 编程规范以 IO 输出方式形成测试脚本文件。

```
public static List<String> accountCreate() {
    //调用shell脚本
    Process ps = Runtime.getRuntime().exec(pathUserCreate);
    ps.waitFor();

    //读取生成的区块链账户信息
    BufferedReader bufferedReader = new BufferedReader(new
        FileReader("/home/usr/shell/user_create_tim/account_info.txt"));
    String index = bufferedReader.readLine();
    String address = bufferedReader.readLine();
    boolean b = LibraUtils.mintCoin(index, 100);
    if (!b) {
        return null;
    } else {
        return new ArrayList<>(Arrays.asList(index, address));
    }
}

//shell脚本，用于在本地链上生成账户
#!/bin/bash
> /home/usr/shell/start/user_create.log
screen -x -S generate -p 0 -X stuff "a c\r"

#从日志中提取账户相关信息
f = open('/home/usr/shell/start/user_create.log', 'r')
f1 = open('/home/usr/shell/user_create_tim/account_info.txt', 'w')

for line in f.readlines():
    if line.startswith("Created/retrieved"):
        s = line.split(" ")
        index = s[3][1:]
        address = s[5]
        f1.write(index + '\n')
        f1.write(address)
```

图 5.7: 区块链账户生成功能关键代码

```
public static void chooseSeed(int secLength) {
    HashMap<String, List<String>> seedFuncPara = new HashMap<>();
    for (int i = 0; i < secLength; i++) {
        int n = MoveFuzz.randInt(0,FuzzInfo.funcName.size());
        String func = FuzzInfo.funcName.get(n);
        seedFuncPara.put(func,FuzzInfo.funcParamType.get(func));
    }
    FuzzInfo.seed = new HashMap<>();
    FuzzInfo.seedFuncOrder = new ArrayList<>();
    List<Map.Entry<String,List<String>>> entries = new ArrayList<>
        (seedFuncPara.entrySet());
    Collections.shuffle(entries);
    for (Map.Entry<String, List<String>> entry : entries) {
        FuzzInfo.seedFuncOrder.add(entry.getKey());
        List<String> seedParam = new ArrayList<>();
        for(String s: entry.getValue()) {
            if(s.equals("u64")) {
                int n = new Random().nextInt(50);
                seedParam.add(n + "");
            }
            else if(s.contains("Coin")) {
                seedParam.add("token");
            }
        }
    }
}
```

图 5.8: 测试脚本生成功能关键代码

5.1.3 模糊测试执行模块详细设计与实现

模糊测试执行模块主要负责 Move 智能合约模糊测试的执行，包括编译发布 Move 智能合约、编译执行测试脚本、种子变异等功能。编译发布 Move 智能合约功能负责将用户上传的 Move 智能合约中的模块地址修改成系统在 Libra 本地链中为其创建的区块链账户地址，然后调用脚本编译该 Move 智能合约并在创建的区块链账户上发布该 Move 智能合约。编译执行测试脚本功能负责在 Libra 本地链上调用脚本编译并且执行所有未执行过的测试脚本。测试脚本中已经预置了测试预言，在测试脚本执行后如果违反了测试预言则会在日志中输出相关信息，即测试脚本执行过程中完成了测试判定。本文提出的 Move 智能合约模糊

测试方法中的变异策略在系统中体现为根据测试脚本编译日志和执行日志增减挑选的函数种子数量、更改函数种子顺序以及重新生成新的函数参数输入。

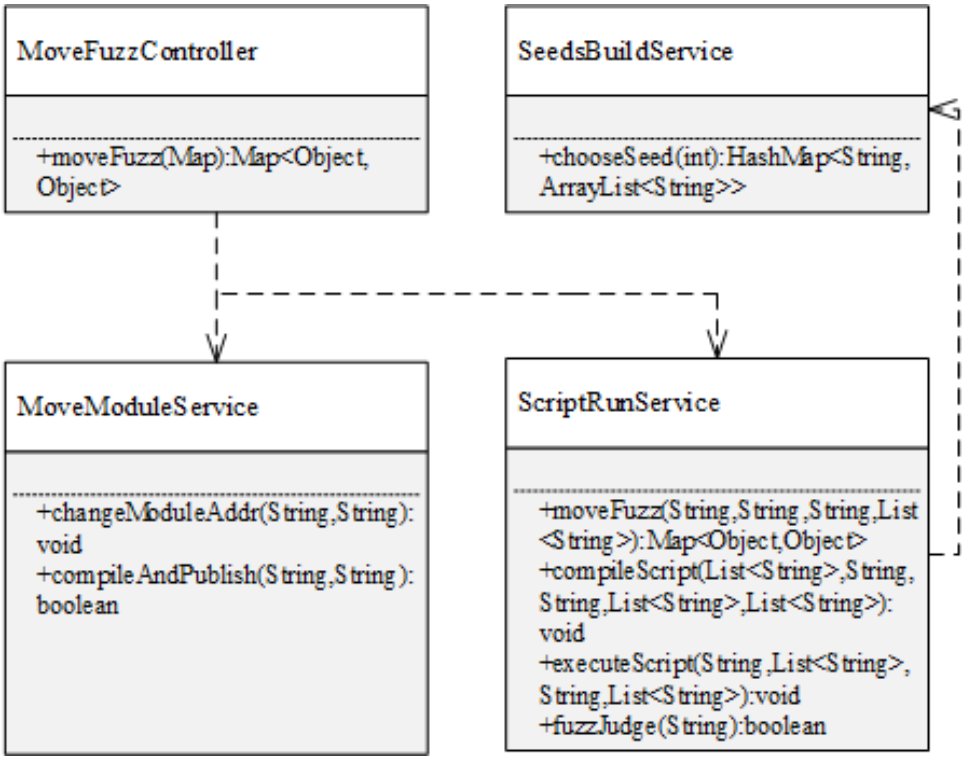

图 5.9: 模糊测试执行模块类图

模糊测试执行模块的核心类图如图 5.9所示。其中 MoveFuzzController 主要负责调用 MoveModuleService 进行 Move 智能合约的编译和发布、调用 ScriptRunService 进行 Move 智能合约测试脚本的编译和执行。MoveModuleService 负责使用 changeModuleAddr() 方法修改 Move 智能合约模块地址，使用 compileAndPublish() 方法调用 shell 脚本编译和执行 Move 智能合约。ScriptRunService 负责处理前端发起的初始测试脚本执行请求，使用 compileScript() 方法编译测试脚本，使用 executeScript() 方法执行测试脚本。在每次测试脚本执行完成后，SeedsBuildService 负责变异函数种子以生成新的测试脚本，变异方法为根据变异策略使用不同的函数输入调用 chooseSeed() 方法。

模糊测试执行模块的活动图如图 5.10所示。用户在前端发起模糊测试请求，后端成功接收到请求后，首先在创建好的区块链账户上变异并发布 Move 智能合约，然后系统继续在创建的账户上编译和执行测试脚本，接着系统根据测试脚本执行日志中的相关输出信息进行测试判定，如果有脚本通过测试判定，则结束模糊测试流程进入到报告生成模块；如果所有已执行的测试脚本都没有通过测试判定，则根据本文提出的模糊测试方法中的变异策略变异当前函数种子，

并生成新的测试脚本进行测试直至有测试脚本通过判定。

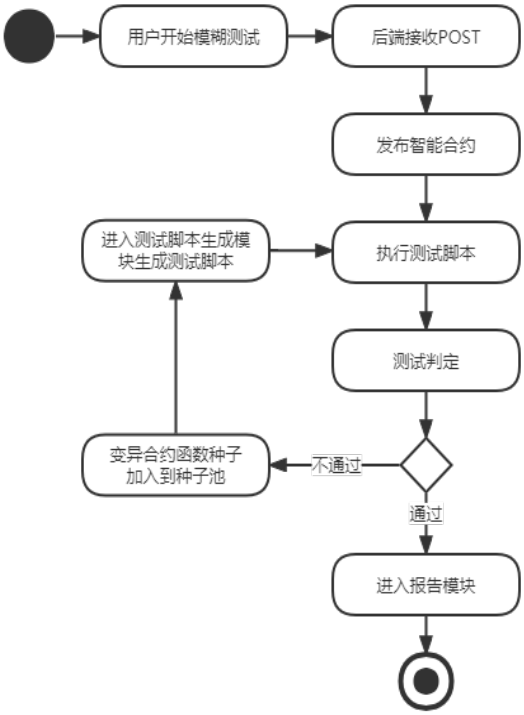

图 5.10: 模糊测试执行模块活动图

```
1  #!/bin/bash
2  screen -x -S generate -p 0 -X stuff "dev c $1 $2 /libra/language/stdlib/modules\r"
3
4  #!/bin/bash
5  screen -x -S generate -p 0 -X stuff "dev p  $1 $2\r"
6
7  #!/bin/bash
8  screen -x -S generate -p 0 -X stuff "dev c $1 $2 $3 /libra/language/stdlib/modules\r"
9
10 #!/bin/bash
11 screen -x -S generate -p 0 -X stuff "dev e $1 $2\r"
```

图 5.11: 模糊测试执行模块关键代码

如图5.11所示代码展示了模糊测试执行模块的关键代码。代码的 1-2 行展示了对 Move 智能合约进行编译的 shell 脚本。代码的 4-5 行展示了对 Move 智能

合约进行发布的 shell 脚本。代码的 7-8 行展示了对测试脚本进行编译的 shell 脚本。代码的 10-11 行展示了对测试脚本进行执行的 shell 脚本。

5.1.4 报告生成模块详细设计与实现

报告生成模块主要负责 Move 智能合约模糊测试执行完成后的整理过程，包括文件下载和执行结果查看两个功能。文件下载负责为用户提供 Move 智能合约模糊测试过程中生成的所有测试脚本源码以及测试脚本执行日志的下载功能。执行结果查看负责向用户展示 Move 智能合约模糊测试过程中所有测试脚本的状态，包括编译是否成功，执行是否成功以及是否违反测试预言从而触发漏洞等，还向用户提供了测试脚本执行日志的前端在线展示功能。

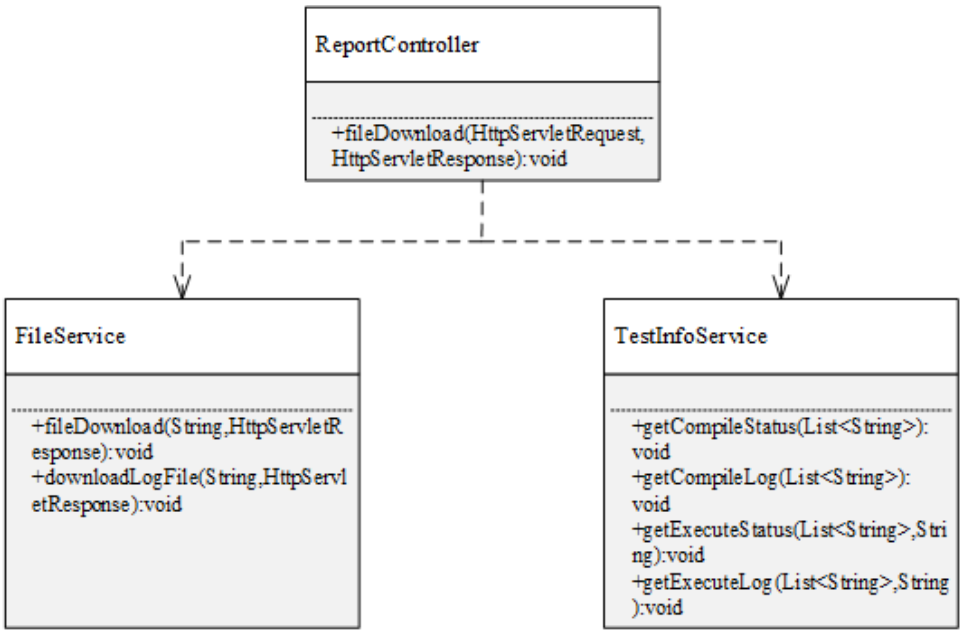

图 5.12: 报告生成模块类图

报告生成模块的核心类图如图 5.12所示。其中 ReportController 主要负责调用 FileService 进行测试脚本及其执行日志的下载，调用 TestInfoService 向用户展示测试脚本在编译和执行过程中的状态、日志等信息。FileService 使用 fileDownload() 方法提供测试脚本的下载功能，使用 downloadLogFile() 方法提供测试脚本执行日志的下载功能。TestInfoService 使用 getCompileStatus() 方法获取测试脚本的编译状态，使用 getCompileLog() 方法获取测试脚本的编译日志，使用 getExecuteStatus() 方法获取测试脚本的执行状态，使用 getExecuteLog() 方法获取测试脚本的执行日志。

报告生成模块的活动图如图 5.13所示。用户在前端选择一个测试脚本点击，

后端接收到请求后将测试脚本转换为 Buffer 流进行处理并将信息返回给前端，前端以弹出框的形式展示用户所选测试脚本的源码。用户选择查看某一测试脚本的执行日志，后端将收到前端发起的查询请求，并且将测试脚本执行日志转换为 Buffer 流进行处理并将信息返回给前端，前端以以弹出框的形式展示用户所选测试脚本的执行日志信息。用户点击某一测试脚本的下载按钮发起下载请求，后端成功接收到文件下载请求后构建文件流对象并生成 FileReponse 对象，后端将其返回给前端，前端即可完成执行日志的下载。

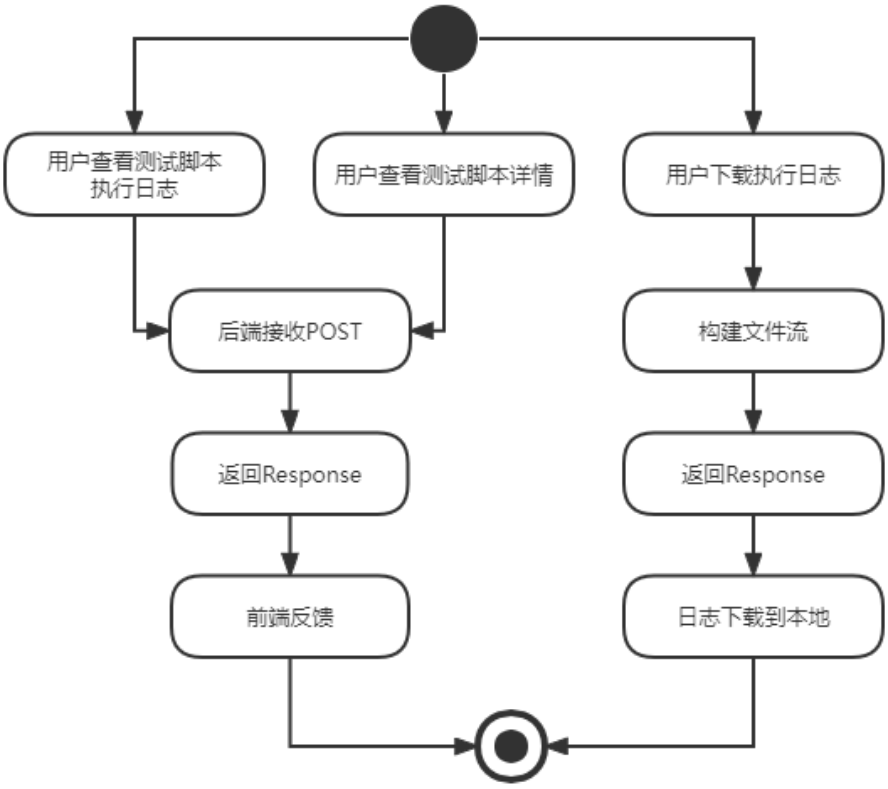

图 5.13: 报告生成模块活动图

根据上述设计，本文对报告生成模块进行了实现。如图5.14所示代码展示了报告生成模块的关键代码。downloadLogFile() 方法主要负责下载用户指定测试脚本的执行日志，该方法以测试脚本文件名作为参数输入。代码的第 6-7 行设置 HttpServletResponse 对象的响应头。代码的第 10-11 行获取测试脚本执行日志的 BufferedInputStream 流。代码的第 12 行获取 OutputStream 输出流。代码的第 14-18 行首先将测试脚本执行日志的字符转换为字节数组，然后使用 OutputStream 流向前端输出字符数组，并由前端完成文件的下载。

```
public void downloadLogFile(String fileName, HttpServletResponse response) {
    String path = logDownloadPath + fileName;

    File file = new File(path);

    response.setContentType("application/force-download");
    response.addHeader("Content-Disposition", "attachment;fileName=" +
        URLEncoder.encode(fileName, "UTF-8"));

    byte[] buffer = new byte[1024];
    FileInputStream fis = new FileInputStream(file);
    BufferedInputStream bis = new BufferedInputStream(fis))
    OutputStream os = response.getOutputStream();

    int i = bis.read(buffer);
    while (i != -1) {
        os.write(buffer, 0, i);
        i = bis.read(buffer);
    }
}
```

图 5.14: 报告生成模块关键代码

5.1.5 系统运行展示

本文实现的 Move 智能合约模糊测试系统包含 4 个页面，分别是 Move 智能合约预处理页面、区块链账户生成页面、测试脚本展示页面以及模糊测试执行结果展示页面。本节将以截图的形式展示系统的 4 个页面，并作一定的描述。

如图5.15所示是 Move 智能合约预处理页面。该页面的功能主要是 Move 智能合约的上传和解析。用户在使用时首先点击页面上的 [验证对象选择] 按钮，在本地选择待测试的 Move 智能合约进行上传。然后点击 [验证对象解析] 按钮即可解析上传的 Move 智能合约，解析结果会在文本框中向用户展示。用户可以点击 [解析结果下载] 按钮下载解析结果 pdf 文件。

如图5.16所示是区块链账户生成页面。该页面的功能是生成并展示 Libra 本地链账户。用户点击 [创建账户] 按钮即可在 Libra 本地链上创建一个新的 Libra 用户账户。用户可在页面上直接看到账户的基本信息，也可点击 [账户详情] 按钮在弹窗中查看所生成账户的详细信息。

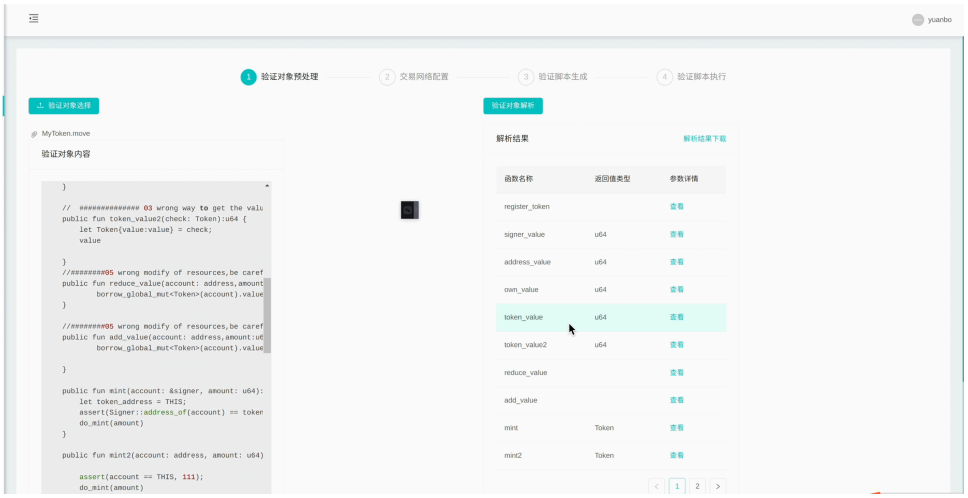

图 5.15: Move 智能合约预处理页面展示图

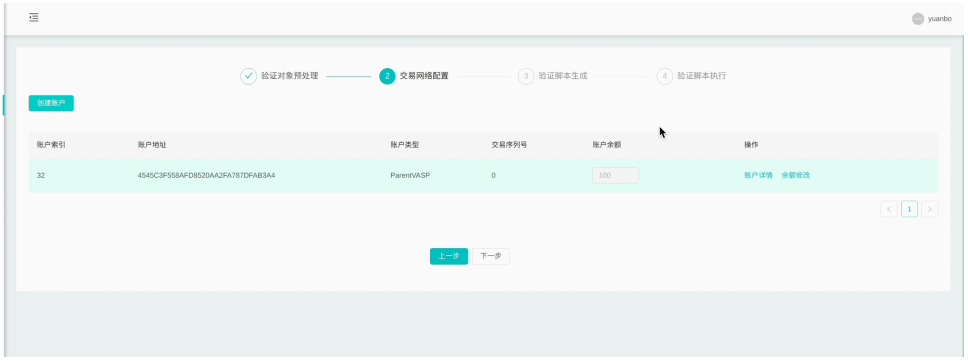

图 5.16: 区块链账户生成页面展示图

如图5.17所示是测试脚本展示页面。该页面的功能是生成并展示模糊测试启动前的初始测试脚本信息。用户可以预先在输入框中输入一个数字指定生成的初始测试脚本的规模，点击 [测试脚本生成] 按钮生成指定规模的测试脚本。列表中展示测试脚本，用户可点击测试脚本 [查看] 按钮查看该测试脚本源代码，也可下载某一测试脚本。

如图5.18所示是模糊测试执行结果展示页面。该页面的功能是启动模糊测试并展示模糊测试执行结果信息。用户可点击 [测试脚本执行] 按钮在 Libra 本地链上执行初始测试脚本并启动模糊测试。在模糊测试执行完成后，列表中将展示模糊测试执行过程中生成的所有测试脚本的执行情况。用户可点击日志 [查看] 按钮查看每个测试脚本的执行日志详情，包括违反的测试预言及触发的漏洞。

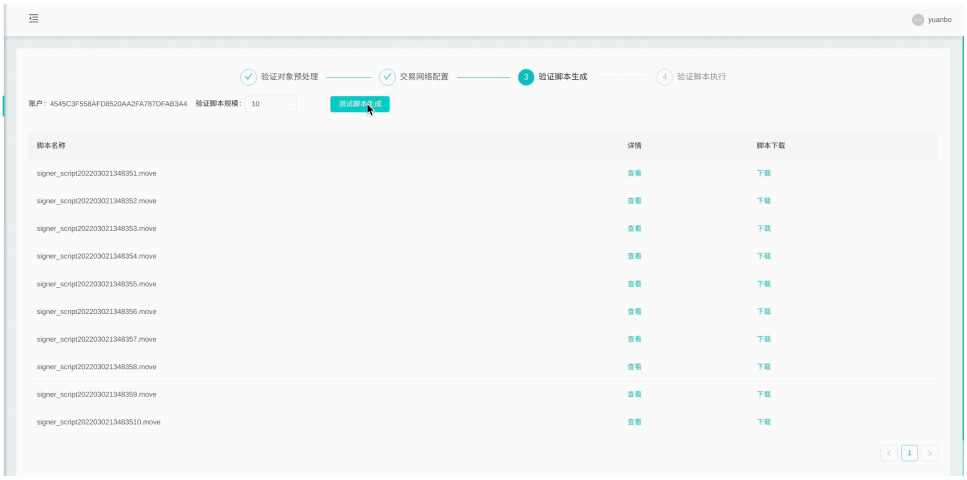

图 5.17: 测试脚本展示页面展示图

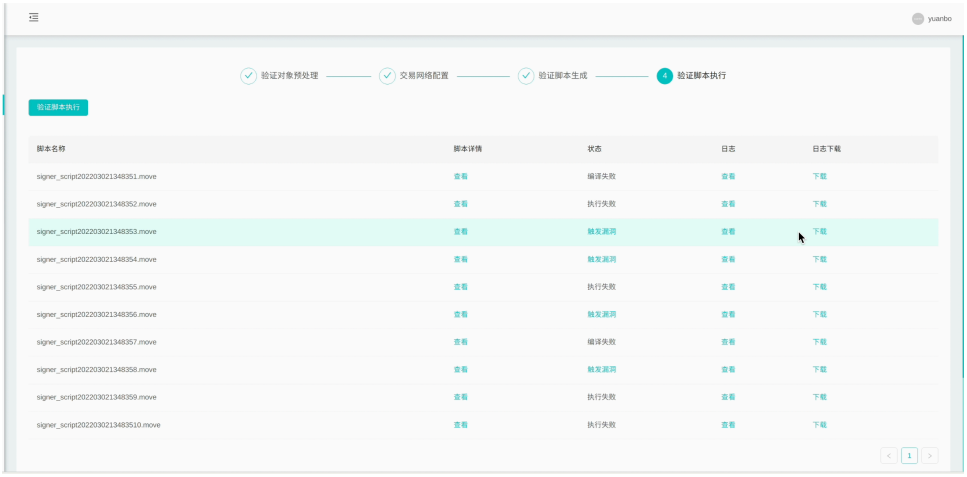

图 5.18: 模糊测试执行结果展示页面展示图

5.2 系统测试

5.2.1 测试环境

该 Move 智能合约模糊测试系统的测试环境如表 5.1所示。表中列举了系统部署和运行所需要的软件和硬件设备。系统进行 Move 智能合约模糊测试需要搭建 Libra 本地链环境。系统的前端使用 Vue 框架进行构建，并通过 Npm 和 Node 进行前端依赖的管理。系统的 API 接口使用 SpringBoot 进行构建，因此需要 Java 运行环境。系统执行 Python 脚本收集模糊测试信息，为此需要 Python 运行环境。

5.2.2 系统测试设计

本小节根据第四章中描述的该 Move 智能合约模糊测试系统的需求分析，从 Move 智能合约解析、区块链账户生成、Move 智能合约模糊测试和模糊测试结

表 5.1: 测试环境配置表

设备或软件	配置或版本信息
Libra	e7ecab683bf37465d3a99cb297c22f326316d908
JDK	1.8.0
Python	3.6.9
SpringBoot	2.4.2
Maven	3.8.3
MySQL	5.7.36
Ipfs	0.7.0
Vue	3.0
Node	v16.4.1
Npm	7.18.1
浏览器	Chrome、FireFox、Edge
操作系统	Ubuntu18.04LTS
系统内存	128G

果展示 4 个方面编写相应的测试用例进行功能测试。

表 5.2: Move 智能合约解析测试用例

测试编号	TC1
测试名称	Move 智能合约解析
测试功能	用户上传 Move 智能合约并进行合约解析
前置条件	用户进入 Move 智能合约解析页面
正常流程	1. 用户点击合约上传按钮，在本地选择 Move 智能合约上传； 2. 系统接收到上传的 Move 智能合约，开始解析合约； 3. 当 Move 智能合约解析结果生成后，用户可点击下载生成的解析结果文件。
预期结果	用户上传 Move 智能合约后，成功进行文件格式检查并返回结果，智能合约成功传入系统。 Move 智能合约解析结果生成后可在前端页面在线查看，也可正常下载到本地查看。 上传情况如下： 1. 如 Move 智能合约文件格式不符，系统提示格式错误并重新上传； 2. 如 Move 智能合约文件不存在，系统提示文件不存在并重新上传； 3. 如 Move 智能合约满足要求，合约上传成功。

Move 智能合约解析的测试用例如表 5.2所示。测试用例对用户进行 Move 智能合约解析的流程进行了模拟。Move 智能合约解析功能包含智能合约上传和智能合约解析两个部分。在预期情况下，智能合约上传接口能够正常访问，并且能够成功进行 Move 智能合约文件格式检查，并返回相应的标识；如果 Move 智能合约成功上传，智能合约解析接口能够正常访问并且正确解析出结果返回给前端。前端接收到 Move 合约解析结果，在文本框中向用户展示具体内容。

区块链账户生成测试用例如表 5.3所示。测试用例对用户进行 Libra 区块链

表 5.3: 区块链账户生成测试用例

测试编号	TC2
测试名称	区块链账户生成
测试功能	用户生成 Libra 区块链账户并查看账户具体信息
前置条件	用户进入区块链账户生成页面
正常流程	1. 用户点击账户生成按钮； 2. 系统在 Libra 本地链上开始创建新账户； 3. 当 Libra 区块链新账户成功生成后，用户可查看账户基本信息； 4. 用户点击账户详情按钮，系统展示账户详细信息。
预期结果	Libra 区块链账户生成接口能够正常访问，并返回创建成功标识和账户基本信息。 账户详情查看接口能够正常访问，并返回账户详细信息，前端以弹窗信息展示。

账户生成和账户信息查看的流程进行了模拟。在预期情况下，Libra 区块链账户生成接口能够正常访问，并且返回账户基本信息，前端以列表形式展示 Libra 区块链账户基本信息；如果 Libra 区块链账户成功创建，LIbra 区块链账户详情查看接口能够正常访问，并且返回 Libra 区块链账户详细信息，前端以弹窗形式展示 Libra 区块链账户详情。

表 5.4: Move 智能合约模糊测试测试用例

测试编号	TC3
测试名称	Move 智能合约模糊测试
测试功能	用户执行测试脚本生成和测试脚本执行操作
前置条件	用户完成区块链账户生成
正常流程	1. 用户进入测试脚本生成页面，点击测试脚本生成按钮； 2. 系统生成测试脚本并返回测试脚本信息； 3. 用户进入模糊测试执行页面，点击测试脚本执行按钮； 4. 系统开始执行模糊测试相关流程，并返回对应的标识。
预期结果	测试脚本生成接口能够正常访问，并返回生成成功标识和测试脚本信息。 测试脚本执行接口能够正常访问，并返回执行成功标识。

Move 智能合约模糊测试的测试用例如表 5.4所示。测试用例对用户进行 Move 智能合约模糊测试的流程进行了模拟。Move 智能合约模糊测试包括测试脚本生成和测试脚本执行两个部分。在预期情况下，测试脚本接口能够正常访问，并且返回测试脚本成功生成标识，前端以列表形式展示生成的测试脚本信息；如果测试脚本成功生成，测试脚本执行接口能够正常访问，并返回模糊测试成功执行标识，并跳转到模糊测试结果展示页面。

模糊测试结果展示的测试用例如表 5.5所示。测试用例对用户查看模糊测试结果和测试脚本信息的操作进行了模拟。在预期情况下，测试脚本展示接口能够正常访问，并返回相应的脚本信息；测试脚本日志查看接口能够正常访问，并

表 5.5: 模糊测试结果展示测试用例

测试编号	TC4
测试名称	模糊测试结果展示
测试功能	用户查看 Move 智能合约模糊测试结果并且下载测试脚本执行信息
前置条件	用户成功进行 Move 智能合约模糊测试
正常流程	1. 用户点击某一测试脚本，弹窗显示测试脚本源代码； 2. 用户点击某一测试脚本日志查看按钮，弹窗显示测试脚本执行日志信息； 3. 用户点击下载日志按钮，查看模糊测试测试脚本执行日志。
预期结果	测试脚本展示接口能够正常访问，并返回测试脚本代码信息。 测试脚本日志查看接口能够正常访问，并返回测试脚本执行日志信息。 测试脚本日志下载接口能够正常访问，并返回测试脚本执行日志文件。

返回相应的日志信息；测试脚本日志下载接口能够正常访问，并返回测试脚本日志文件。

5.2.3 测试结果与分析

表 5.6: 功能测试用例执行结果

测试用例编号	测试用例描述	测试结果
TC1	Move 智能合约解析	通过
TC2	区块链账户生成	通过
TC3	Move 智能合约模糊测试	通过
TC4	模糊测试结果展示	通过

功能测试的执行结果如表 5.6所示。本文依次执行功能测试用例设计阶段设计的测试用例进行系统功能测试，并严格遵照测试用例中设计的测试流程进行系统操作。根据测试结果表中的数据看出，所有 4 个测试用例的执行结果全部通过。测试结果表明系统满足需求分析中提出的功能性需求，系统各项功能能够正常运行。

5.3 本章小结

本章首先分别介绍了基于资源不变量的 Move 智能合约模糊测试系统的四个主要模块的详细设计和实现，结合类图、流程图和关键代码对各模块的实现细节进行了描述。接着展示了系统运行的主要界面，并进行介绍。最后对系统进行了功能性测试设计并对功能性测试结果进行了分析。

第六章　总结与展望

6.1　总结

随着区块链的不断发展，各类智能合约的大规模使用导致的财产损失屡见不鲜，智能合约安全漏洞在以太坊等主流区块链平台上也经常被攻击者利用。Libra 作为一类旨在创立全球支付系统的区块链平台，其对数字资产的安全性看得尤为重要。Move 智能合约作为运行在 Libra 区块链上的一类智能合约，它的安全性和可靠性与 Libra 数字资产息息相关。为了提高 Move 智能合约的安全性和可靠性，本文结合 Move 智能合约中资源的特性，采用模糊测试的技术，提出了一种基于资源不变量的 Move 智能合约模糊测试方法。该方法首先解析 Move 智能合约源代码，从而构建种子池。其次随机挑选种子生成测试脚本，然后根据测试预言对测试脚本执行结果进行判定，根据测试脚本执行结果是否异常判定 Move 智能合约中是否包含漏洞。最后对测试脚本使用的种子进行变异，生成新的函数输入和函数调用序列，形成变异后的种子加入到种子池中。

为了有效检测 Move 智能合约中存在的漏洞，本文提出了一种测试预言——资源不变量，并给出了详细定义及描述。本文设计并实现了基于资源不变量的 Move 智能合约模糊测试系统。该系统前端使用 Vue 框架进行搭建，后端则使用 Spring Boot 框架搭建代码逻辑并为前端提供 Restful API。该系统由测试种子生成模块，测试脚本生成模块，模糊测试执行模块以及测试报告生成模块构成，实现了 Move 智能合约上传与解析，Libra 区块链账户生成，测试脚本生成，测试脚本执行，测试结果查看与下载等功能。系统采用 Docker 技术进行打包，确保了系统的可移植性，便于在不同环境下部署该系统，降低了运维的开销。

经过系统测试运行，本文提出的基于资源不变量的 Move 智能合约模糊测试方法能够检测出 Move 智能合约在资源权限方面的漏洞，比如错误使用 signer，函数可见性声明错误等；在资源使用方面的漏洞，比如错误的资源修改，错误的资源销毁等，为 Move 智能合约开发者提供一定的帮助。

6.2　展望

本文初步实现了基于资源不变量的 Move 智能合约模糊测试系统，系统在使用便利性和功能多样性方面还存在一些需要改进之处：

（1）目前系统使用的资源不变量测试预言仅能检测出在 Move 智能合约交易

前后与资源相关的漏洞，后续将会尝试扩展测试预言以检测出更多类型的 Move 智能合约漏洞。

（2）目前系统初始生成的测试脚本由于随机调用 Move 智能合约函数，未考虑某些函数调用有前后序关系，导致有一定比例的测试脚本无法在 Libra 本地链上成功编译，延长了模糊测试执行时间。后续将会优化测试脚本生成逻辑，以提高模糊测试的效率。

（3）为了提高系统与用户之间的交互性，后续将会增加相关配置，允许用户指定 Move 智能合约函数及其输入以启动模糊测试。后续也将会优化 Move 智能合约解析逻辑，以适应不同开发人员开发的不同风格的 Move 智能合约代码。

参考文献

[1] AMSDEN Z. The Libra BlockChain[EB/OL]. 2020 (2020/5/26) [2022/5/20]. https://diem-developers-components.netlify.app/papers/the-diem-blockchain/2020-05-26.pdf.

[2] BLACKSHEAR S. Move: A Language With Programmable Resources[EB/OL]. 2020 (2020/5/26) [2022/5/20]. https://diem-developers-components.netlify.app/papers/diem-move-a-language-with-programmable-resources/2020-05-26.pdf.

[3] ETHERSCAN. Etherscan[EB/OL]. 2022 (2022/1/20) [2022/5/20]. https://etherscan.io/.

[4] HACKED S. Slowmist[EB/OL]. 2022 (2022/4/8) [2022/5/20]. https://hacked.slowmist.io/.

[5] SUTTON M, GREENE A, AMINI P. Fuzzing: brute force vulnerability discovery[M]. USA : Pearson Education, 2007.

[6] BAI X, CHENG Z, DUAN Z, et al. Formal modeling and verification of smart contracts[C] // Proceedings of the 2018 7th international conference on software and computer applications. 2018 : 322 – 326.

[7] ABDELLATIF T, BROUSMICHE K-L. Formal verification of smart contracts based on users and blockchain behaviors models[C] // 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS). 2018 : 1 – 5.

[8] GARFATTA I, KLAI K, GAALOUL W, et al. A survey on formal verification for solidity smart contracts[C] // 2021 Australasian Computer Science Week Multiconference. 2021 : 1 – 10.

[9] SHISHKIN E. Debugging smart contract’s business logic using symbolic model checking[J]. Programming and Computer Software, 2019, 45(8) : 590 – 599.

[10] WEI Z, WENYIN Z, JIURU W, et al. Smart contract vulnerability detection scheme based on symbol execution[J]. Journal of Computer Applications, 2020, 40(4) : 947.

[11] SHI Y, SAGDUYU Y E, DAVASLIOGLU K, et al. Vulnerability detection and analysis in adversarial deep learning[M]. USA : Springer, 2018.

[12] WU F, WANG J, LIU J, et al. Vulnerability detection with deep learning[C] // 2017 3rd IEEE International Conference on Computer and Communications (ICCC). 2017 : 1298 – 1302.

[13] LIANG H, PEI X, JIA X, et al. Fuzzing: State of the art[J]. IEEE Transactions on Reliability, 2018, 67(3) : 1199 – 1218.

[14] TAKANEN A, DEMOTT J D, MILLER C, et al. Fuzzing for software security testing and quality assurance[M]. USA : Artech House, 2018.

[15] LI J, ZHAO B, ZHANG C. Fuzzing: a survey[J]. Cybersecurity, 2018, 1(1) : 1 – 13.

[16] FEY G. Assessing system vulnerability using formal verification techniques[C] // International Doctoral Workshop on Mathematical and Engineering Methods in Computer Science. 2011 : 47 – 56.

[17] GRISHCHENKO I, MAFFEI M, SCHNEIDEWIND C. A semantic framework for the security analysis of ethereum smart contracts[C] // International Conference on Principles of Security and Trust. 2018 : 243 – 269.

[18] AMANI S, BÉGEL M, BORTIN M, et al. Towards verifying ethereum smart contract bytecode in Isabelle/HOL[C] // Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs. 2018 : 66 – 77.

[19] KALRA S, GOEL S, DHAWAN M, et al. Zeus: analyzing safety of smart contracts.[C] // Ndss. 2018 : 1 – 12.

[20] VAAS. Automated Formal Verification Platform for Smart Contract.[EB/OL]. 2019 (2022/4/20) [2022/5/20].
https://www.lianantech.com/.

[21] BALDONI R, COPPA E, D' ELIA D C, et al. A survey of symbolic execution techniques[J]. ACM Computing Surveys (CSUR), 2018, 51(3) : 1 – 39.

[22] LUU L, CHU D-H, OLICKEL H, et al. Making smart contracts smarter[C] // Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. 2016 : 254 – 269.

[23] NIKOLIĆ I, KOLLURI A, SERGEY I, et al. Finding the greedy, prodigal, and suicidal contracts at scale[C] // Proceedings of the 34th annual computer security applications conference. 2018 : 653 – 663.

[24] TSANKOV P, DAN A, DRACHSLER-COHEN D, et al. Securify: Practical security analysis of smart contracts[C] // Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018 : 67 – 82.

[25] KRUPP J, ROSSOW C. {teEther}: Gnawing at Ethereum to Automatically Exploit Smart Contracts[C] // 27th USENIX Security Symposium (USENIX Security 18). 2018 : 1317 – 1333.

[26] TANN W J-W, HAN X J, GUPTA S S, et al. Towards safer smart contracts: A sequence learning approach to detecting security threats[J]. arXiv preprint arXiv:1811.06632, 2018.

[27] ZHUANG Y, LIU Z, QIAN P, et al. Smart Contract Vulnerability Detection using Graph Neural Network.[C] // IJCAI. 2020 : 3283 – 3290.

[28] WANG W, SONG J, XU G, et al. Contractward: Automated vulnerability detection models for ethereum smart contracts[J]. IEEE Transactions on Network Science and Engineering, 2020, 8(2) : 1133 – 1144.

[29] JIANG B, LIU Y, CHAN W. Contractfuzzer: Fuzzing smart contracts for vulnerability detection[C] // 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE). 2018 : 259 – 269.

[30] WÜSTHOLZ V, CHRISTAKIS M. Harvey: A greybox fuzzer for smart contracts[C] // Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2020 : 1398 – 1409.

[31] ZHENG Y, DAVANIAN A, YIN H, et al. FIRM-AFL:High-Throughput Greybox Fuzzing of IoT Firmware via Augmented Process Emulation[C] // 28th USENIX Security Symposium (USENIX Security 19). 2019 : 1099 – 1114.

[32] CHEN J, DIAO W, ZHAO Q, et al. IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing[C] // NDSS. 2018.

[33] GOOGLE. syzkaller[EB/OL]. 2015 (2022/4/20) [2022/5/20]. https://github.com/google/syzkaller.

[34] JEONG D R, KIM K, SHIVAKUMAR B, et al. Razzer: Finding kernel race bugs through fuzzing[C] // 2019 IEEE Symposium on Security and Privacy (SP). 2019 : 754 – 768.

[35] CORINA J, MACHIRY A, SALLS C, et al. Difuze: Interface aware fuzzing for kernel drivers[C] // Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017 : 2123 – 2138.

[36] 朱建明, 付永贵. 基于区块链的供应链动态多中心协同认证模型 [J]. 网络与信息安全学报, 2016, 2(1) : 27 – 33.

[37] 周立群, 李智华. 区块链在供应链金融的应用 [J]. 信息系统工程, 2016(7) : 49 – 51.

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

[39] CHEN Y, DING S, XU Z, et al. Blockchain-based medical records secure storage and medical service framework[J]. Journal of medical systems, 2019, 43(1) : 1 – 9.

[40] 赵阔, 邢永恒. 区块链技术驱动下的物联网安全研究综述 [J]. 信息网络安全, 2017(5) : 1 – 6.

[41] CHRISTIDIS K, DEVETSIKIOTIS M. Blockchains and smart contracts for the internet of things[J]. Ieee Access, 2016, 4 : 2292 – 2303.

[42] BERNSTEIN P A, HADZILACOS V, GOODMAN N. Concurrency control and recovery in database systems[M]. USA : Addison-wesley Reading, 1987.

[43] REED D P. Naming and synchronization in a decentralized computer system[D]. USA : Massachusetts Institute of Technology, 1978.

[44] WOOD G, OTHERS. Ethereum: A secure decentralised generalised transaction ledger[J]. Ethereum project yellow paper, 2014, 151(2014) : 1 – 32.

[45] PIERCE B C. Types and programming languages[M]. USA : MIT press, 2002.

[46] BARR E T, HARMAN M, MCMINN P, et al. The oracle problem in software testing: A survey[J]. IEEE transactions on software engineering, 2014, 41(5) : 507 – 525.

[47] BEIZER B. Black-box testing: techniques for functional testing of software and systems[M]. USA : John Wiley & Sons, 1995.

[48] MYERS G J, SANDLER C, BADGETT T. The art of software testing[M]. USA : John Wiley & Sons, 2011.

[49] GODEFROID P, LEVIN M Y, MOLNAR D A, et al. Automated whitebox fuzz testing.[C] // NDSS. 2008 : 151 – 166.

[50] PHAM V-T, BÖHME M, SANTOSA A E, et al. Smart greybox fuzzing[J]. IEEE Transactions on Software Engineering, 2019, 47(9) : 1980 – 1997.

[51] 张潇藜, 马佳利, 刘子昂. 以太坊 Solidity 智能合约漏洞检测方法综述 [J]. 计算机科学, 2022, 49(3) : 52 – 61.

[52] 付梦琳, 吴礼发, 洪征. 智能合约安全漏洞挖掘技术研究 [J]. 计算机应用, 2019, 39(7) : 1959 – 1966.

[53] ATZEI N, BARTOLETTI M, CIMOLI T. A survey of attacks on ethereum smart contracts (sok)[C] // International conference on principles of security and trust. 2017 : 164 – 186.

简历与科研成果

基本情况

袁博，男，汉族，1998 年 2 月出生，江苏省启东市人。

教育背景

2020 年 9 月 ~ 2022 年 6 月 南京大学软件学院 硕士

2016 年 9 月 ~ 2020 年 6 月 北京交通大学计算机与信息技术学院 本科

致　　谢

两年的研究生生涯即将迎来尾声，在此衷心向在学习和生活上帮助过我的老师和同学表示感谢。

首先，我要感谢南京大学 iSE 实验室和我的研究生导师陈振宇老师，陈老师对待学术严谨认真，对待每一位学生耐心负责，从研究生入学确定研究方向到毕业论文的开题以及最终完成都给予了我许多指导和帮助，在此我由衷感谢陈老师对我的关心和支持。感谢实验室的王兴亚老师，王老师在项目、毕设以及论文方面会经常和我交流并给出帮助和建议，监督和指导我完成了各项工作，在我研究生道路上起到了指引的作用。

其次，我要感谢在实验室的这两年里所有帮助过我的学长学姐和同学们，感谢你们帮助我度过研究生生涯早期的迷茫阶段，感谢你们对我学业以及求职的帮助和建议。感谢陆军工程大学的孙金磊博士对我毕设研究内容以及毕业论文的指导和帮助。我还要感谢我的舍友和朋友，感谢大家两年来的陪伴，希望大家以后都能前途似锦。

感谢南京大学提供的学习环境和生活环境。感谢软件学院所有任课老师的教导，你们传授的知识和做人做事的道理将让我终生受益。两年来，疫情反反复复，感谢学校所有工作人员的辛苦付出。

最后，我要感谢我的家人。感谢你们一直以来对我的关爱和支持，有你们作为坚实后盾，我无论遇到什么困难都不会轻言放弃。