南京大学
NANJING UNIVERSITY

研 究 生 毕 业 论 文

（申请硕士专业学位）

论 文 题 目 人机协同 Java 字节码漏洞扫描系统的设计与实现

作 者 姓 名 蒋燕

专 业 名 称 软件工程

研 究 方 向 软件工程

指 导 教 师 陈振宇 教授　房春荣 助理研究员

2020 年 5 月 22 日

学　　号：MF1832071

论文答辩日期：2020 年 5 月 23 日

指 导 教 师：　　（签字）

The Design and Implementation of Human-machine Collaboration-Based System for Java Vulnerability Scanner

By

Yan Jiang

Supervised by

Professor **Zhenyu Chen**

Research Assistant **Chunrong Fang**

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： 人机协同 Java 字节码漏洞扫描系统的设计与实现

软件工程 专业 2018 级硕士生姓名： 蒋燕

指导教师（姓名、职称）： 陈振宇 教授 房春荣 助理研究员

摘 要

软件系统规模以及复杂性的不断增大，软件安全问题层出不穷，其产生的原因往往是程序本身在代码设计或实现过程中的错误或缺陷（称为漏洞），普通软件工程师的缺陷密度一般为 50—250 个缺陷/KLOC。在源码级别对程序进行漏洞扫描和安全审计，可以在源头上减少 10%—50% 安全漏洞的产生。但现存基于词法分析的静态漏洞扫描工具未充分考虑上下文，无法准确判别误报漏洞代码的固有特征，存在大量误报信息，开发人员需要手动筛选正报漏洞，不仅增大维护成本，甚至导致部分开发人员弃用扫描工具。

为了降低目前漏洞扫描工具误报率，本文依托于公司的众包审核平台，设计并实现了一个人机协同 Java 字节码漏洞扫描系统。本文对静态漏洞扫描工具以及常见误报漏洞进行了分析，详细研究了字节码上下文提取、代码特征提取以及机器学习分类模型，同时融合众包专家审核，并结合实际场景中的漏洞扫描需求，实现该系统。首先，系统基于静态漏洞扫描工具对项目进行漏洞扫描，并确保其完备性。其次，基于 Joana（Java Object-sensitive ANAlysis）程序切片工具对漏洞相关代码进行上下文提取，并基于 N-gram 语言模型对代码上下文进行特征提取。然后，利用基于完全匹配、随机森林算法的双层分类模型对扫描漏洞结果进行误报过滤。最后，将漏洞结果送予众包审核进行专家误报漏洞过滤，并将审核结果存储留作后续过滤模型的迭代训练。并根据最终漏洞结果为用户提供机器过滤以及专家审核融合的完备、低误报漏洞扫描报告。本系统主要分为交互展示模块、漏扫核心模块以及迭代学习模块，并使用 SpringBoot 框架、Pug 模板引擎、微服务等技术与架构完成系统的实现。

本文实现的人机协同 Java 字节码漏洞扫描系统提供低误报漏洞扫描服务。在 OWASP 数据集上实验表明，本系统在 95.39% 召回率的情况下，其精准率可以达到 89.71%，与原版扫描工具相比，本系统将误报率减少近 22%。本系统在确保低漏报率的基础上有效地降低传统静态漏洞扫描工具的误报率，从而节约维护成本，帮助开发者提高代码整体质量。目前，本系统已在公司平台上线，用于支撑公司静态漏洞扫描服务。

关键词：漏洞扫描，误报过滤，众包审核，人机协同

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 Human-machine Collaboration-Based System for Java Vulnerability Scanner

SPECIALIZATION: Software Engineering

POSTGRADUATE: Yan Jiang

MENTOR: Professor Zhenyu Chen, Research Assistant Chunrong Fang

Abstract

With the continuous increase in the scale and complexity of software systems, software security issues are emerging. The main cause of these issues is often errors or defects(called vulnerability) in the code itself, and ordinary software engineers generate generally 50-250 defects per KLOC. Vulnerability scanning and security auditing at the source level can reduce security vulnerabilities by 10% to 50% and save 5% to 20% of maintenance costs. However, most existing static vulnerability scanners, which are based on lexical analysis, ignore the context of vulnerability, which cannot accurately identify the inherent characteristics of false-positive vulnerability code. There is usually a large number of false-positive cases, such that developers need to manually check the results, and even abandon scanners, which increases maintenance costs.

In order to reduce the false positive ratio of current vulnerability scanners and save maintenance costs for developers, this thesis design and implement a human-computer collaborative Java bytecode vulnerability scanning system. It analyzes static vulnerability scanners and common false positive vulnerabilities. This thesis also studies bytecode context extraction, code feature extraction, and related machine learning classification models. It also integrates crowdsourcing auditing and combines the vulnerability scanning requirements in actual scenarios to implement the system. First, the system scans submitted projects based on static vulnerability scanners and ensures the completeness of vulnerabilities. Secondly, the vulnerability-related code is extracted by Joana (Java Object-sensitive ANAlysis) slicing tool, and the relevant features are extracted from the context. Then, multi-layer classification models based on algorithms such as similarity, random forest are used to filter the scan vulnerability results. Finally, the vulnerability results are sent to the crowdsourcing auditors for manually false

positive filtering, and the results are stored for iterative training of subsequent filtering models. So far, it is used to provide two complete vulnerability report with low false positive ratio, which are after-filtering report and after-auditing report. This system is mainly divided into interactive display module, vulnerability scanning core module and iterative learning module, and uses SpringBoot framework, Pug template engine, microservice and other technologies and architecture to implement the system.

The human-machine collaborative Java bytecode vulnerability scanning system implemented in this thesis can provide better vulnerability scanning service. Experiments on the OWASP dataset show that the precision of this system can reach 89.71% when the recall rate is 95.39%. Compared with the original scanner, this system reduces the false positive ratio by nearly 22%. The system can effectively reduce the false positive ratio of traditional static vulnerability scanners while ensuring a low false negative ratio. Thereby the system can save maintenance costs and help developers improve code quality. At present, this system has been launched on the company's platform to support the development of the company's static vulnerability scanning service.

Keywords: Vulnerability Scanning, False Positive Filter, Crowdsourcing Audit, Human-Machine Colloboration

目录

表 目 录

插图

第一章　引言

1.1　研究背景与意义

随着计算机技术的广泛使用，软件系统的复杂性和规模也在不断增大，在给生活带来极大便利的同时，也暴露出更多的安全问题 [1]。近年来，软件安全问题层出不穷，其产生的主要原因往往是程序本身在代码设计或实现过程中存在错误或缺陷（称为漏洞），恶意攻击者通常会利用这些安全漏洞攻击软件系统，从而导致各类信息被窃、重要数据丢失、服务器性能下降，严重威胁被攻击软件系统和用户的利益安全。360 互联网安全中心 [2] 指出，软件开发过程中通常会引入大量缺陷，普通软件工程师的缺陷密度一般为 50-250 个缺陷/KLOC[1]。同时《2019 中大型政企机构网络安全建设发展趋势研究报告》 [3] 中指出源码安全审计可以在系统开发阶段发现潜在安全漏洞，可节约 5%-20% 后期安全维护费用，并且可减少 10%-50% 系统安全漏洞。因此在源码级别对程序进行漏洞扫描和安全审计，可以在源头上减少安全漏洞的产生，从而节约维护费用，有效减少恶意攻击事件的发生，保证软件安全性。

程序分析技术的迅速发展也推进了代码漏洞扫描技术的进展，尤其是代码漏洞静态扫描技术。但是代码漏洞静态检测方法在不执行程序的情况下进行漏洞扫描，难以在有限时间内判定抽象路径的可达性 [4]。为了不漏掉真实漏洞，即保证低漏报，大多静态扫描工具使用基于模式匹配、数据流分析等技术，并采用过度近似策略，这些工具通常不能充分了解程序上下文，会受到无关信息的干扰 [5]。以上原因导致现有漏洞扫描工具在尽可能检测出漏洞的同时，会产生大量的误报漏洞，即现有工具一般低漏报、高误报。开发人员或安全工程师需要手动检查以及修正这些漏洞，而大量误报的存在增加了工程师筛选真正漏洞的成本，这也是导致部分开发者弃用漏洞扫描工具的原因之一 [6]。因此，降低漏洞扫描工具的误报率至关重要。

由于误报漏洞的某些固有特征，如不可达漏洞代码等，目前直接通过静态漏洞扫描工具无法舍弃这种误报，但在安全工程师或开发人员眼中却一目了然。如图 1.1中的代码片段，可能产生漏洞的位置为第 10 行，由于用户可控的参数 param 可能会赋值给 bar，但是第 6 行的条件判断永远为 true，因此第 9 行代码永远不会被执行，故该代码不存在漏洞，但很多漏洞扫描工具会将其作为漏洞返回，此为误报。

[1]KOL(千行代码)，是一个软件系统有多大或需要多少人来完成其编码工作的一个度量标准

```java
@RequestMapping
@ResponseBody
public String pathtraver_fp(String param) {
  String bar;
  int i = 106;
  if ((7 * 18) + i > 200)
    bar = "This_should_always_happen";
  else
    bar = param;
   new File(new File(Utils.testfileDir), bar);
}
```

图 1.1: 误报漏洞示例代码

综上所述，由于代码的固有特征以及静态扫描的本质缺陷，静态漏洞扫描工具存在大量误报结果，而纯人工漏洞检测需要手动检测相关代码，存在成本高、耗时长等缺点。因此设计一种人机协同的、有效降低漏洞扫描误报率的服务是有意义且十分必要的。通过本系统，用户提交漏洞扫描任务后，多种扫描工具将对项目进行漏洞扫描，以确保漏洞结果的完备性；通过基于特征提取与机器学习的误报过滤模型失败误报漏洞，最后通过众包专家审核人工核实是否正报，降低整体漏洞扫描的误报率，进而减少开发人员或安全工程师审计代码安全性的时间，节约项目开发与维护成本，帮助开发人员提高系统整体安全性。如何结合机器学习以及众包审核等技术来实现上述系统即是本文研究与阐述的重点。

1.2　研究现状

代码漏洞扫描相关领域一直都是工业界与学术界的研究热点，本节主要介绍漏洞扫描相关领域的国内外研究现状。

1.2.1　漏洞扫描现状

代码漏洞扫描一般依赖于两种程序分析技术：动态程序分析和静态程序分析。其中，动态程序分析技术通过运行程序并观察程序在运行过程中的相关行为来进行漏洞扫描 [7]。基于动态程序分析技术的漏洞扫描可以准确发现程序在运行时可能会产生的漏洞，但其扫描成本较高，漏洞覆盖率较低，无法有效扫描出漏洞。与之相反，静态程序分析技术不需要执行程序，直接扫描代码中可能存在的错误 [8]。基于静态程序分析技术的漏洞扫描通过分析程序代码中不同的控制流程以及变量可能存在的不同值在程序运行前进行漏洞扫描，也正因如此，静态扫描通常会产生大量的误报信息。

常见静态分析技术包括：词法分析、语法分析、抽象语法树分析、语义分析、控制流分析、数据流分析、污点分析 [9] 和无效代码分析等。其中，词法分析 [10] 基于模式匹配，对源码字符流进行扫描，将之与预定义漏洞正则表达式匹配，从而判断是否漏洞，这类分析技术实现较为简单，但未考虑上下文，只能识别特定类型漏洞，会产生大量误报。数据流与污点分析等主要借助数据流图对代码中的数据流向进行相关分析，这一类分析方法适合于数组越界、注入类等漏洞，但无法很好分析容器变量、控制流等信息，也会存在大量误报。

市场上效果较好的主流 Java 静态扫描工具主要有如下几种：基于缺陷模式匹配的 PMD [11]，它能检测源代码中如不必要对象创建、未使用字段等漏洞；基于缺陷模式匹配的 CheckStyle，它主要用来分析代码编写是否符合规定的编码规则；基于缺陷模式匹配、数据流分析以及污点分析的 SpotBugs[2]和 Find-sec-bugs[3]插件，它们主要对 Java 字节码进行相关漏洞扫描；商业工具 FortifySCA，它自定义统一的中间表示形式可以分析绝大多数主流编程语言编写的代码 [12]。其中，PMD 和 CheckStyle 主要关注代码规范问题，SpotBugs 和 Find-sec-bugs 更注重分析安全问题，而 FortifySCA 是商业工具，价格较为昂贵。

以上工具主要存在误报率高的问题 [13]，增加了开发者或安全工程师手动筛选正报漏洞的成本，甚至导致开发人员弃用扫描工具。MITRE[4]组织发起 CWE（Common Weakness Enumeration）[5]项目对软件系统的常见漏洞进行了分类整理，并为每种漏洞提供相关示例以及解决方案。这一标准可以为工程师提供统一标准参考，针对高误报问题，学术界则提出了很多误报过滤技术。

1.2.2　误报过滤现状

过高的误报率会增加开发人员的开发维护成本，甚至导致开发人员弃用漏洞扫描工具。降低漏洞扫描工具误报率的方法一般分为两种：一是在扫描工具层面，改进工具本身的扫描机制，提高扫描工具的准确性；二是在扫描结果处理层面，对扫描工具产生的结果进行二次过滤。前者需要对工具本身使用的核心技术进行改进或替换，不利于后期迭代扩展，该方法的实现难度较大。后者在漏洞扫描工具的基础上，通过构建误报检测模型对漏洞结果进行过滤，以区分误报漏洞和正报漏洞，该方法基于代码特征分析与机器学习中的分类模型，无需关注扫描工具底层原理，适用性较广，故研究者较多。

Bharti 等人 [14] 通过结合程序切片、迭代上下文扩展（ICE）以及有界模型

[2]https://spotbugs.github.io/

[3]http://find-sec-bugs.github.io/

[4]https://www.mitre.org/

[5]https://cwe.mitre.org/

检查的循环抽象（LABMC）来对程序中的大循环进行抽象，从而检测出由于大循环造成的相关漏洞误报，该方法局限于大循环里的误报漏洞，对于其他位置的误报并不能很好检测出。Ibéria 等人 [15] 使用常见的分类模型如 ID3、KNN、朴素贝叶斯等对基于污点传播扫描的漏洞进行误报检测，此分类模型效果较好，但只针对基于污点传播扫描的漏洞，类型有所局限。JooWoo 等人 [16] 使用抽象语法树表示代码的结构性特征，并使用支持向量机模型（SVM）来进行分类模型的训练，该方法忽略了代码的语义信息。Koc 等人 [17] 提取误报和正报代码的字节码指令集，并结合贝叶斯和长短时记忆模型在六种不同的漏洞类型上进行分类的训练，精确度接近 80%，该方法实验数据较少，无法保证真实环境下的效果，且实验涉及漏洞较少，无法保证其他漏洞类型上的效果。Arzt 等人 [18] 提出一种基于符号执行方法来检测安卓程序中的数据流误报，通过检测扫描出的漏洞代码执行路径，找出进入该执行路径的相关条件，判断这些条件是否能被满足，从而消除逻辑上无法执行的路径上产生的相关漏洞，该方法需要大量计算资源，在处理大规模程序时无法保证产生结果。

使用机器学习等技术的最大问题是能否在真实工程项目中获得如实验中足够高的准确率。以上相关研究的实验均是在相关机构公布的数据集或少数开源项目上进行，无法保证其在真实项目中的效果。因此，我们需要结合众包审核技术来扩增大量真实的数据集，以此确保误报过滤在真实项目中的效果。

1.2.3　众包技术现状

众包技术可以在极大程度上解决误报过滤模型前期训练数据不足的问题。众包技术 [19] 是软件测试领域的一个新兴趋势，众包技术通过将任务直接发布至互联网，并招募大量众包工人参与完成任务，从而获取大量真实的数据。其中定向众包 [20] 是一种众包工人分配方式，定向众包通过众包平台将相关任务分配给指定的众包工人，这些定向的众包工人一般为某一领域的专家或某一组织的内部成员。因此，一般可以认为定向众包工人能够高效高质量地完成任务，即众包审核后的结果是可靠的。

现有众包技术一般用于众包测试领域，即平台发布测试任务，众包工人完成任务并填写相关测试报告，经被测产品方确认后给予众包工人相应奖励，如 HackerOne、补天、漏洞盒子等现有企业级产品。本文结合漏洞扫描与众包审核技术，将扫描后的漏洞分发给相关领域专家进行误报漏洞审核，进一步降低系统整体的漏洞率；同时，众包审核后产生的真实数据为误报过滤模型的迭代训练提供了数据支撑。

1.3　本文主要研究工作

针对当前代码漏洞静态扫描工具误报率高，开发人员难以快速筛选正报结果从而导致扫描工具不可用或不易用的问题，本文深入研究现有代码漏洞扫描工具的原理与优缺点，并结合众包审核技术设计并构建了人机协同的低误报 Java 字节码漏洞扫描系统。

本系统主要通过以下三个方面的设计来实现该目标：

首先系统关注于扫描工具的改造与集成。对比选取市场上主流漏洞扫描工具，并根据 CWE 标准为工具能扫描出的漏洞修改或补充相关描述信息。扫描工具在本身扫描机制实现中可能存在不合理的地方，从而直接导致部分误报的产生，因此，本文深入研究工具的扫描机制以及相应漏洞列表，并结合部分误报漏洞特征，在扫描工具层面手动屏蔽部分漏洞的扫描。此部分需对扫描工具本身进行相应改造与封装，并按统一标准集成到系统中。

其次系统集中于如何训练分类模型，实现对误报漏洞的过滤。本文将采用代码缩减技术与机器学习相结合的方法 [17] 实现误报过滤模型的构建。漏洞相关代码一般是少量的，本文基于扫描工具漏洞相关位置信息提取漏洞相关上下文，以消除无关代码的干扰。然后，根据具体实验效果，选取 N-gram 与随机森林进行分类模型的训练。

最后系统围绕如何结合众审进一步实现误报过滤，同时促进误报过滤模型的迭代训练，实现人机协同的迭代漏洞扫描系统。根据机器扫描结果与众审结果生成人机融合扫描报告，误报更低可靠性更高。众审结果不断扩增真实数据集，促进误报过滤模型的迭代训练，使得系统整体误报率更低。

综合上述需求与设计，本系统使用 Spring Boot 框架开发，并使用 Docker 进行独立部署，与其他服务松耦合。系统使用 HTTP 协议与其他服务进行数据传输，并向外提供 RESTful 接口，简单标准易扩展。其中，封装的扫描工具和过滤模型需按指定标准集成接入，方便工具和模型的快速迭代和统一管理。静态报告采用 Pug 模板引擎进行编译，可自动化渲染出图文并茂的漏洞扫描报告。

1.4　本文组织结构

本文的组织结构如下：

第一章引言部分。本章首先介绍了项目背景以及本项目的研究意义，然后分析了国内外静态漏洞扫描的研究现状以及存在的不足，最后介绍了本文的主要研究工作和论文的组织结构。

第二章相关技术综述。介绍了本项目涉及的几种常见漏洞类型，根据本项目的具体需求，选取适合本项目的相关技术方案及框架。主要包括扫描工具、程序切片技术、分类算法、微服务以及页面渲染工具等。

第三章漏洞扫描系统的需求分析与设计。首先对系统进行了整体概述，然后结合用例图、逻辑视图等方式对系统进行了功能以及非功能性需求分析，最后介绍了交互展示模块、漏扫核心模块、迭代学习模块三个模块的概要设计。

第四章漏洞扫系统的具体实现。在第三章的基础上，阐述了交互展示模块、漏扫核心模块以及迭代学习模块的具体实现，并给出了相应的核心类图、关键代码以及界面截图。

第五章漏洞扫描系统的测试与实验评估。首先介绍了系统的测试环境，然后对整个系统进行了相关功能与性能测试，最后对本系统降低漏洞扫描误报率效果进行了实验验证。

第六章总结与展望。对本文所做工作进行了总结，分析了研究工作中存在的不足，并对未来的进一步工作进行了展望。

第二章　相关技术综述

2.1　常见漏洞类型

一般来说，Web 安全漏洞对 Java 程序危害最大，研究常见漏洞类型可以有效帮助漏洞扫描与分析。本节主要介绍 3 种 Web 安全中常见的漏洞：SQL 注入、路径遍历和弱加密。

2.1.1　SQL 注入

SQL 注入是一种针对携带数据系统应用的漏洞，尤其是 Web 应用。其本质是，攻击者利用该漏洞在 SQL 语句中注入恶意代码，而 SQL 服务器会对这些恶意代码进行解析并执行，从而导致数据库数据被修改、敏感信息泄露等后果 [21]。SQL 注入攻击的关键条件主要有两点，一是用户可以控制自身输入，二是用户输入会被拼接到 SQL 查询语句中并被解析执行。图 2.1代码片段展示了最典型 SQL 注入方式，第 2 行动态拼接用户输入的“name”和“passwd”形成 SQL 语句（如第 5 行所示），若攻击者将“name”输入为“tom”，“passwd”输入为“passwd’ or ‘a’ = ‘a”，则最终执行的 SQL 语句如第 7 行，此时 SQL 执行条件恒为 true，导致数据库数据泄露。

```
1  ……
2  String query = "select * from user where name = '"
       + name + "' and passwd = '" + passwd + "'";
4  ……
5   select * from user where name = <name> and passwd
        = <passwd>;
7   select * from user where name = 'tom' and
      passwd = 'pass' or 'a' = 'a';
```

图 2.1: SQL 注入示例代码

SQL 注入常见的防护措施是破坏两个关键条件，一是通过严格转义和过滤控制用户输入，二是利用预处理和参数化使用户输入无法直接拼接成 SQL 语句，从而恶意代码不会被服务器解析执行。本系统主要通过验证代码中是否存在破坏两个关键条件来进行 SQL 注入相关漏洞的扫描。

2.1.2　路径遍历

路径遍历漏洞[1][2]又称目录遍历漏洞，指攻击者通过操纵使用“../”序列及其变体引用文件的变量或使用绝对文件路径，绕过服务器安全限制，访问存储在文件系统上的任意文件和目录，包括应用程序源代码或配置和关键的系统文件，甚至执行系统命令。其产生原因一般是程序在实现上没有对用户输入的特殊字符进行严格控制与过滤。如图 2.2示例代码片段，第 5 行直接根据用户输入文件路径读取文件内容返回，攻击者可利用该漏洞访问任意文件。

```
1   @RequestMapping
2   @ResponseBody
3   public String readFile(String param) {
4     try {
5       FileInputStream fis = new FileInputStream(new File(param));
6       InputStreamReader isReader = new InputStreamReader(fis);
7       BufferedReader br = new BufferedReader(isReader);
8       String line = null;
9       StringBuilder sb = new StringBuilder();
10      while ((line = br.readLine()) != null) {
11        sb.append(line);
12      }
13      return sb.toString();
14  ......
15  }
```

图 2.2: 路径遍历示例代码

路径遍历漏洞最常见的防护措施是严格验证用户输入，尤其是需要过滤掉“../”等目录跳转符，此外，还可以通过判断最终的文件路径确保请求文件完整目录在合法范围。本系统通过验证代码是否对用户输入文件路径进行足够安全的校验来进行相关漏洞扫描。

2.1.3　弱加密

弱加密[3]是指程序使用的加密算法容易被破解，如 DES 加密算法可以通过穷举法在有限时间内破解。攻击者可以使用暴力破解或其他方法破解弱加密算法，在未被授权的情况下获取已加密的信息，从而造成个人隐私信息泄露甚至财产损失等严重后果。如图 2.3所示代码片段，第 3~5 行实例化 DES 加密算法的密

[1]https://cwe.mitre.org/data/definitions/23.html

[2]https://cwe.mitre.org/data/definitions/36.html

[3]http://cwe.mitre.org/data/definitions/261.html

钥生成器并指定操作模式为加密，第 7~8 行对输入进行加密并返回加密后结果。其中 DES 加密算法完全依赖于密钥，易受穷举法攻击，代码潜在弱加密漏洞。

```
1   public String encrypt(String input) {
2     try {
3       Cipher cipher = Cipher.getInstance("DES");
4       SecretKey key = KeyGenerator.getInstance("DES").generateKey();
5       cipher.init(Cipher.ENCRYPT_MODE, key);
6       byte[] inputByte = input.getBytes();
7       byte[] result = cipher.doFinal(inputByte);
8       return Arrays.toString(result);
9     } catch (Exception e) {……}
10    return "none";
11  }
```

图 2.3: 弱加密示例代码

弱加密漏洞一般防护措施是避免使用不安全或强度弱的加密算法，如 DES、3DES 加密算法等，在安全性要求较高的系统中，应使用如 AES、RSA 等强加密算法对敏感数据进行加密。

2.2　漏洞扫描相关工具

Java 漏洞扫描一般有两种方式，源代码和字节码扫描。针对字节码进行漏洞扫描更能保证开发者代码的隐私性，且字节码更便于分析。本节将对 Java 字节码扫描工具以及字节码分析框架进行阐述。

2.2.1　SpotBugs 工具

SpotBugs 是由美国马里兰大学推出的一款 Java 静态扫描工具，主要通过将 Java 字节码与一组预定义的缺陷模式进行匹配，以寻找并定位 Java 代码中可能存在的漏洞。目前预定义 400 多种缺陷模式，并且开发者可以自定义缺陷模式以实现定制化检测。SpotBugs 主要使用缺陷模式匹配和数据流分析技术进行静态扫描，其中缺陷模式匹配技术是将代码与从代码分析经验中收集的缺陷模式进行正则匹配，以发现并定位疑似漏洞；而数据流分析技术通过代码中变量引用信息以分析变量在程序中的传递流向，同时还能检测代码数据流异常。

SpotBugs 具有缺陷模式较为完善、漏洞类型比较完备等优点。并且一些实验表明，SpotBugs 具有非常可观的检测能力 [22–24]。本系统将对 SpotBugs 进行相应改造与集成并将其作为核心 Java 静态漏洞扫描工具。

2.2.2　Find-sec-bugs 工具

Find Security Bugs 是 SpotBugs 工具的一个扩展插件，该工具支持 816 种 API 签名共计可检测 135 种漏洞类型，同时覆盖主流框架和类库，如 SpringMVC、Struts、Tapestry 等。Find-sec-bugs 工具主要参照 OWASP TOP 10 和 CWE 标准，部分漏洞给出相关参考链接，具有较高可靠性。与 SpotBugs 相比，Find-sec-bugs 工具主要关注如 CSRF、SQL 注入、命令执行、路径遍历等 Web 安全漏洞，其主要采用污点传播技术进行分析。其作为扩展插件，需依赖于 SpotBugs 运行，因此本系统同时集成 SpotBugs 和 Find-sec-bugs 作为核心扫描工具，主要关注其中 Web 安全相关的漏洞。

2.2.3　程序切片技术

代码之间的依赖关系较为复杂，大量漏洞无关代码的存在严重干扰对漏洞信息的判断，因此，需要排除无关代码。而程序切片技术可以提取指定的部分代码，本系统使用该技术提取漏洞上下文。

程序切片技术自诞生以来，一直被广泛研究 [25]。Weiser 在 1979 年首次提出程序切片概念，并将其作为基于控制流图（CFG）的数据流问题解决方案 [26]。随后，Ottenstein 等人首次提出利用程序依赖图（PDG）来计算程序切片 [27]，其切片算法主要基于程序依赖图中的图可达性。然而该方法只适用于单个程序过程的切片，为了处理多个函数间的切片，Horwitz 等人在 PDG 基础上，引入了系统依赖图（SDG），同时提出 SDG 的遍历算法来进行函数间切片 [28]。之后，众多学者陆续提出了诸如信息流关系、并行算法、规范条件等方法对程序切片进行不断优化 [29–32]。

针对 Java 语言进行程序切片的主流工具主要有 WALA（T.J. Watson Libraries for Analysis）[4]和 Joana（Java Object-sensitive ANAlysis）[5]。WALA 最初是 IBM T.J. Watson 研究中心 DOMO 研发项目的一部分，主要通过 Java 字节码分析 Java 类型系统和类层次结构，利用上下文信息、指针分析、调用图构造以及基于 SSA 的寄存器转换中间语言等来进行切片。Joana 是 Karlsruhe 理工学院基于 WALA 实现的另一款 Java 程序切片工具，其主要使用技术包括指针分析、异常分析以及程序依赖图。Joana 根据程序信息流构建 SDG，每条语句都通过节点来表示，节点之间的边则表示语句间信息依赖关系，从而更准确提取相关程序切片。本系统选择 Joana 作为程序切片核心工具。

[4]http://wala.sourceforge.net/wiki/index.php/Main_Page

[5]https://pp.ipd.kit.edu/projects/joana/

2.3　语言模型与分类算法

语言模型源于自然语言处理领域，是用来计算一个句子概率的模型，一般应用在问答系统、分词、语音识别、自动补全等相关应用上。常见语言模型主要有 RNN [33]、LSTM [34] 等神经网络语言模型以及 N-gram [35] 等统计语言模型。机器学习领域常见分类算法有朴素贝叶斯、决策树 [36]、支持向量机 [37]、随机森林等。其中 N-gram 模型中之前出现的词对之后出现的词具有很强的约束力，比较适合用于代码语言模型构建；而随机森林相较于其他分类模型具有较大优势，它能够随机选择特征子集以处理高维度数据，并且模型训练速度快、泛化能力强。因此，本文使用 N-gram 语言模型对漏洞相关信息进行预处理，并使用随机森林作为误报过滤的分类模型。

2.3.1　N-gram 模型

N-gram 算法基于统计语言模型，其利用长度为 N 的滑动窗口操作将文本内容划分成长度为 N 的文本片段序列。其中每一个文本片段称为 gram，再过滤掉出现频数少的 gram，最后形成关键 gram 列表，即文本的向量特征空间，而每一个 gram 就是一个特征向量维度。该模型的假设前提为，第 N 个词只与前面 N-1 个词相关，因此，整个文本的概率为各个词出现概率的乘积 [38]。

对于语言序列 $S(\omega_1, \omega_2, \omega_3, \ldots, \omega_n)$，基于统计概率计算句子概率大小：

$$P(S) = P(\omega_1, \omega_2, \omega_3, \ldots, \omega_n) \tag{2.1}$$

首先，由条件概率和链式法则，可得：

$$P(\omega_1, \omega_2, \omega_3, \ldots, \omega_n) = P(\omega_1)P(\omega_2 \mid \omega_1)P(\omega_3 \mid \omega_1, \omega_2) \ldots P(\omega_n \mid \omega_1, \ldots, \omega_{n-1}) \tag{2.2}$$

为了简化计算公式2.2，引入马尔科夫假设（Markov assumption）[39]，即当前词出现的概率只依赖于前面有限个（$m-1$）词，即

$$P(\omega_i \mid \omega_1, \omega_2, \ldots, \omega_{i-1}) = P(\omega_i \mid \omega_{i-m+1}, \ldots, \omega_{i-1}) \tag{2.3}$$

基于公式2.3，当设置 $m = 1, 2, 3, \ldots$ 时可以得到相应的一元模型（Unigram）、二元模型（Bigram）、三元模型（Trigram）等。其中，当 $m = 3$ 时，得到三元模型如下：

$$P(\omega_1, \omega_2, \ldots, \omega_n) = \prod_{i-1}^{n} P(\omega_i \mid \omega_{i-2}, \omega_{i-1}) \tag{2.4}$$

此外，需要给原序列首部加上起始符，以确保句首词的条件概率有意义，同时表征句首词出现的条件概率。为了使得 N-gram 模型能对任意长度序列进行概率分布建模，还需在原序列尾部加入结束符。

N-gram 模型具有参数易训练、可解释性强、完全包含前 $n-1$ 个词的全部信息等优点。本系统将使用 3-gram 模型对待分类漏洞进行预处理操作。

2.3.2　Random Forest 算法

随机森林（Random Forest）是指利用多棵决策树对样本进行训练并预测的一种分类器，其最终预测类别一般由某些树输出类别的众数决定。随机森林属于集成学习（Ensemble Learning）中的 Bagging 算法 [40]。

决策树是一种基本分类器，一般用于二分类问题，构建好的决策树呈树形结构，可以看做由一系列学习训练得到的 if-then-else 规则组成。决策树是最简单的机器学习分类算法，易于实现，可解释性强。而随机森林由多个决策树组成，不同决策树相互独立。当进行分类任务时，随机森林中的每个决策树独立对该任务进行分类，最终随机森林预测的分类结果由类别输出中的众数决定。

构建使用随机森林一般分为以下四个步骤：

1. 构造子数据集：假设原数据集数量为 N，有放回随机选择 N 个数据构造子数据集。子数据集数据量与原数据集相同，但其元素不完全相同。不同子数据集元素可以重复，同一子数据集中元素也可以重复。将构造的子数据集进行决策树训练，并作为决策树根节点处数据。

2. 构建决策树：决策树在节点分裂时，随机从所有属性 M 中选取少数属性 m，再从 m 个属性中基于某种策略选择最优属性作为该节点的分裂属性。重复此步骤，直到节点不能再分裂，即决策树构建完成。

3. 构建随机森林：重复步骤 1 和 2，构建大量决策树，从而组成随机森林。

4. 使用随机森林：当对新数据进行分类时，森林中每棵决策树都对该数据生成一个分类结果，根据所有决策树的投票结果，得到随机森林的最终分类结果。

随机森林在当前很多数据集上，相对其他算法有很大的优势，它能够处理高维度数据，且泛化能力较强，不容易陷入过拟合，具有较好的抗噪能力。由于随机森林中每棵决策树相互独立，因此易做成并行化方法，训练速度快。其准确率可以和 Adaboost 媲美，对错误和离群点更鲁棒。因此，本系统使用随机森林作为误报过滤的基本分类模型。

2.4　系统核心技术栈

本系统采用前后端分离的设计模式，前端页面使用 AngularJS 模板引擎，服务端使用 Spring Boot 框架，前后端之间采用 RESTful 风格的 HTTP 请求进行通信，同时本系统将使用 Docker 容器化与微服务架构。本节将对本系统核心技术栈进行相应阐述。

2.4.1　Docker 容器技术

随着物理服务器性能的提升，一台物理服务器上可能同时运行多个服务，此时多个服务会相互影响，比如某个服务内存泄漏可能导致整个服务器崩溃，其他服务也会瘫痪。因此，需要一种技术将每个服务独立出来，拥有独立的内存和资源，即虚拟化技术。最早提出的容器化技术是虚拟机，虚拟机将物理硬件资源虚拟化，按需分配和使用，虚拟机使用过程域真实操作系统一样，当废弃不用时直接删除虚拟机文件即可回收资源，方便集中管理。

但虚拟机非常庞大，并且对硬件资源消耗大，因此 Linux 提出另一种虚拟化技术，即 Linux 容器。Linux 容器并未虚拟出一个完整的操作系统，而是提供应用运行所必须的资源，以最小消耗达到虚拟机的效果。与虚拟机相比，虚拟机时操作系统级别的隔离，而容器是进程级别的隔离。

而 Docker 是对 Linux 容器的封装，提供简单实用的用户接口，让开发者可以将应用以及依赖包打包到一个可移植容器中，然后发布到 Linux 机器上。其中，容器使用沙箱机制以确保相互之间不会影响。Docker 可以在操作系统上，提供一个额外的软件抽象层以及操作系统虚拟化（即容器化）的自动管理机制，从而让应用程序可以部署在容器中自动化隔离运行 [41]。根据行业公司 451 研究[6]：“Docker 是有能力打包应用程序机器虚拟容器，可以在任何 Linux 服务器上运行的依赖性工具，这有助于实现灵活性和便携性，应用程序在任何地方都可以运行，无论是公有云、私有云，还是单机等。”[7]

Docker 可以解决虚拟机资源消耗问题，如图 2.4，虚拟机架构中，虚拟机运行在服务器操作系统上，客户操作系统运行在虚拟机上，应用程序则运行在客户机操作系统上，此时一台服务器大部分资源都消耗在硬件虚拟化和客户机操作系统本身。而 Docker 容器架构中，容器引擎之上运行虚拟服务器，应用程序运行虚拟服务器中，其中虚拟服务器和服务器操作系统使用同一内核、同一文件系统，通过隔离技术保证各个虚拟服务器的工作相互独立。此时虚拟服务器并不对物理服务器硬件进行虚拟化，因此虚拟服务器性能接近于物理服务器性能。

[6]https://451research.com/

[7]https://docs.readthedocs.io/en/stable/

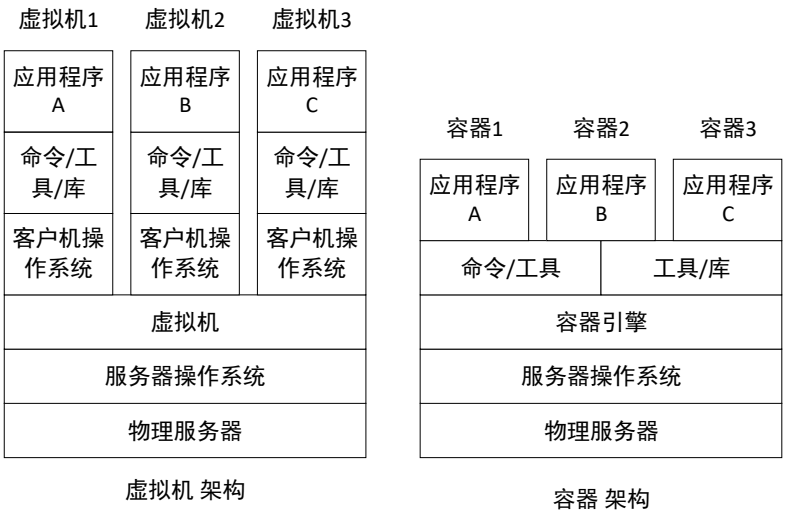

图 2.4: 虚拟机与 Docker 架构对比图

除此之外，Docker 还具有可快速部署、提供一次性运行环境、提供弹性云服务等优点，因此，本系统将以 Docker 形式发布，便于集成到平台。

2.4.2　微服务

目前应用主流设计一般会采用微服务架构，其思路是将整个应用分解为小型、互相连接的微服务，一个微服务完成某个特定功能，每个微服务都有自身业务逻辑，一些微服务还会提供 API 接口给其他微服务和应用客户端使用。通常情况下，客户端不能直接访问后台微服务，而是通过 API Gateway 来转发请求，API Gateway 一般负责服务路由、负载均衡、缓存、访问控制和鉴权等任务。

微服务架构有很多优点。首先，将单体应用分解为一组服务，整体应用可以模块化管理各服务，有利于服务的重用。其次，每个微服务业务与技术独立，可由小规模团队独立开发，并且开发团队可以自由选择合适的语言与技术栈。第三，每个微服务可独立部署。最后，每个服务都可独立扩展 [42]。因此平台整体采用微服务架构，本系统实现为平台的一个微服务。

Spring Cloud 为开发者提供快速构建分布式系统的通用模型工具，包括配置管理、服务发现、熔断器、分布式会话、集群状态等。而 Dubbo 是阿里巴巴提出的一个开源分布式服务框架，其致力于提供高性能和透明化的 RPC 远程访问调用方案以及 SOA 服务治理方案，其核心部分主要包含：提供对多种基于长连接的 NIO 框架抽象封装以实现远程通讯，提供基于接口方法的透明远程过程调用以实现集群容错，基于注册中心目录服务实现服务自动发现。本系统主要采用 Dubbo 分布式服务框架。

2.4.3　Pug 模板引擎

模板引擎是将静态部分糅合的一种实现机制或技术，其主要目标是分离用户界面与业务数据，根据特定格式的文档数据，模板引擎可以渲染生成一个标准 HTML 文件。目前使用较广的模板引擎主要有 Pug[8]、EJS 和 Handlebars。

express 是基于 Node.js 平台，快速、开发、极简的 Web 开发框架，而 Pug 是 express 的默认模板引擎。与 EJS 和 Handlebars 模板引擎对比，前二者仍主要使用 HTML 原始标签进行开发，而 Pug 使用自定义类 HTML 语言进行页面开发，经过配置和编译才能在浏览器中运行。相比之下，Pug 代码风格更简洁，开发效率高，且具有超强可读性、灵活易用缩进、拥有编译和运行时上下文错误报告等优点。因此本文使用 Pug 模板引擎作为报告页面渲染工具。

2.5　本章小结

本章主要针对常见漏洞类型、漏洞扫描相关技术、语言模型与分类算法以及系统核心技术栈四个方面进行了相关技术阐述。首先，介绍了 SQL 注入、任意文件上传、跨站请求伪造以及命令执行四种常见 Web 安全漏洞的示例以及相应修复方案。其次，对漏洞系统中涉及的漏洞扫描工具 SpotBugs、Find-sec-bugs 扫描工具以及程序切片技术进行了详细阐述。然后，介绍了 N-gram 语言模型以及随机森林算法的原理和优缺点。最后，介绍了本系统开发过程中涉及的核心技术栈，主要包括 Docker 容器技术、微服务架构以及 Pug 模板引擎。

[8]Pug，原名 Jade：https://pugjs.org/api/getting-started.html

第三章　系统需求分析与概要设计

本章将概述人机协同 Java 字节码扫描系统的需求分析与设计。首先，概述本系统的开发目标和总体功能流程。其次，分析本系统的功能和非功能需求，并设计相关用例图进一步说明功能需求。然后，对系统进行整体设计，并结合 4+1 视图进一步说明系统模块，同时阐述本系统涉及的持久化对象设计。最后，将对系统各个模块进行详细设计。

3.1　系统整体概述

现有静态漏洞扫描工具主要基于词法分析以及过度近似策略进行漏洞的扫描，导致扫描结果中存在大量误报漏洞，无法有效实现代码审计节约维护成本的预期效果。针对这一问题，本文设计并实现了人机协同的 Java 字节码漏洞扫描系统，结合机器过滤以及专家审核降低漏洞扫描工具的误报率，在尽可能返回所有真实漏洞的情况下，减少开发人员手动筛选正报漏洞的成本，从而节约维护成本。

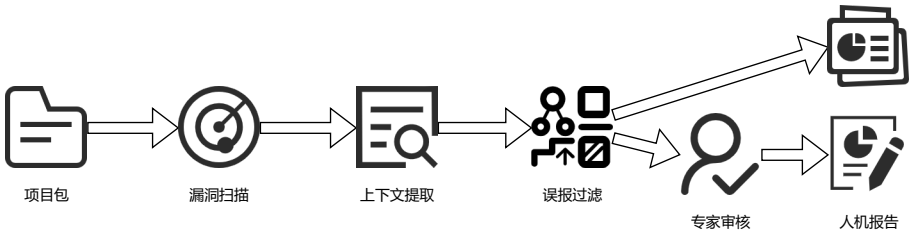

图 3.1: 人机协同 Java 字节码漏洞扫描流程

图 3.1展示了本系统基于人机协同进行 Java 字节码漏洞扫描的整体流程。用户提交项目包并发布扫描任务，本系统对该项目包进行基于静态分析技术的漏洞扫描，其次对扫描结果进行漏洞相关代码上下文以及相关特征提取，然后经过相似性和机器学习分类模型两层误报过滤，再根据用户是否选择众包审核服务选择性发布众包审核任务，最后根据机器扫描结果以及众包审核人工结果生成扫描报告供用户查阅。通过改造并集成合适的漏洞扫描工具，以确保扫描出漏洞的完备性，从而保证结果的可靠性以及代码的安全性。通过漏洞相关代码上下文提取，一方面为误报过滤模型提供漏洞相关特征，以便进行误报过滤；另一方面，为众包审核工人提供更直观的漏洞审核内容，并经过相关推荐算法精

准下发给适合的众包审核工人，同时可为众包审核工人提供相似漏洞的参考审核结果，从而提高人工审核的效率。通过两层误报过滤，可以更有效地降低误报率。众包审核的历史数据，仍可作为误报过滤模型迭代训练更新的训练集。最终，通过上述方案，本系统显著降低了漏洞扫描误报率，从而节约了开发人员手动审核成本，提供了更有效率的漏洞扫描服务。

3.2　系统需求分析

需求分析是软件工程中的一个关键过程，也是系统开发的根本依据。本节将就功能性和非功能性两个方面明确本系统的整体需求，同时使用相关用例来描述功能性需求。

3.2.1　漏洞类型分析

根据 OWASP TOP 10[1]，2020 年十大 Web 安全漏洞类型分别为注入、错误的授权、敏感数据泄露、XXE、损坏的访问控制、错误的安全配置、XSS、不安全的反序列化、使用有漏洞的第三方库以及不足够的日志和监控。结合 SpotBugs 和 Find-sec-bugs 工具定义的漏洞列表，本系统主要关注 Web 安全方面的漏洞，并将工具提供漏洞映射到常见漏洞类型上，此处选取最常见类型如下：

1. SQL 注入：使用用户不可信输入拼接 SQL 语句，可能导致执行非预期操作，从而造成信息泄露、绕过验证等后果。
2. 弱加密：使用如 DES 弱加密算法对密码等敏感信息进行加密，攻击者可以暴力破解此类加密算法，从而造成信息泄露等后果。
3. 路径遍历：直接使用用户不可信输入拼接文件路径，可能导致读取非合法路径下的文件，从而造成任意文件访问等后果。
4. XSS 跨站脚本：攻击者通过在链接中插入恶意代码并将其植入到提供给其它用户使用的页面中，从而造成信息窃取等后果。
5. 命令注入：使用用户不可信输入拼接命令执行语句，攻击者可通过构造特殊命令字符的方式将恶意代码植入，从而导致恶意命令执行。
6. 其他：其他映射类型还包括序列化、CSRF 跨站请求伪造、XPath 注入、伪随机数、重定向 LDAP 注入、XXE 外部实体注入等。

[1]https://owasp.org/www-project-top-ten/

3.2.2　功能性需求

本系统是完整的软件系统，因此，首先本系统满足基本用户管理功能，在此基础上，功能需求主要围绕交互展示模块、漏扫核心模块以及迭代学习模块三部分进行分析。本系统主要功能需求如表 3.1所示。其中 R1、R2、R3、R4 为用户需求，R5、R6、R7 为满足用户需求所产生的系统需求。

表 3.1: 系统功能性需求列表

需求 ID	需求名称	需求描述
R1	任务管理	任务发布者可以对任务进行管理，包括任务创建、任务配置与任务发布；管理员可以审核任务，包括同意任务发布、拒绝任务发布和查看任务信息。
R2	查看任务列表	任务发布者可以查看自己发布的所有任务，每个任务能看到扫描进度。
R3	发布众审	任务发布者可以将完成扫描的任务发布众审，在众审服务上可以自定义选择需要发布众审的漏洞，并且可以自定义审核内容。
R4	查看报告详情	任务发布者可以查看并下载扫描后生成的漏洞报告，若发布众审，则还可以查看和下载众审后人机融合报告。
R5	漏洞扫描	系统需对用户提交的项目进行 Java 字节码漏洞扫描，并提取漏洞相关代码上下文，生成静态报告。
R6	误报过滤	系统需对扫描结果进行相似性和机器学习分类两层过滤，舍弃误报漏洞并标记疑似误报漏洞。
R7	迭代训练	系统需收集众审数据集，不断扩增误报过滤模型训练集，周期性迭代训练并更新误报过滤模型。

在任务管理模块，用户可以创建扫描任务，填写任务基本信息并进行扫描流程相关配置，之后可以发布该任务。接着，系统管理员可以查看该任务详情，并同意或拒绝该任务发布。最后，当系统管理员同意任务发布时，该任务将自动发布，并按照用户配置的流程进行扫描。

在任务列表页面，用户可以查看自己发布的所有任务，并且可以按任务进度进行筛选。对于已完成扫描的任务，用户可以查看并下载生成的静态报告；同时，用户还可以前往众审服务，选择漏洞列表并自定义审核内容，将该任务发布众审。当众审完成后，用户可以在该界面查看并下载生成的众审后报告。

任务成功发布后，将根据任务配置自动分配给相应的扫描流程，调用相应的静态扫描工具对用户提交项目进行漏洞扫描。扫描结束后，使用字节码分析工具对漏洞相关代码提取上下文信息。之后，使用预训练的分类模型对漏洞结果进行误报过滤。其中，对于明显误报漏洞直接舍弃，对于不确定误报漏洞，将

在报告中标记为疑似误报，并建议用户进行众包专家审核。最后，生成完整的静态报告，以供用户查阅。

迭代训练部分需要收集众包审核数据以扩展训练集，对模型进行再训练优化后，更新替换现有误报过滤模型。

3.2.3　非功能性需求

非功能需求一般包括质量保障、性能需求、系统约束以及可扩展性等 [43]，本系统的非功能需求如表 3.2所示。

表 3.2: 系统非功能性需求列表

名称	内容	需求类别
扫描工具可扩展	系统可以集成新扫描工具，也可以添加检测项。	可扩展性
过滤模型迭代更新	系统应在扫描流程抽象好相关接口，以便于过滤模型的迭代更新。	可扩展性
高响应	使用分析类接口应有状态信息返回，在网络正常情况下，查询与读写操作应小于 500ms。	性能需求
低漏报低误报	系统需保证漏洞扫描结果低漏报低误报。	性能需求
数据完整	应对数据进行备份，系统中工具发生崩溃时，保证数据库中数据不丢失；服务器宕机时，应通过负载均衡等方法及时切换备用服务器。	质量保障
操作简单	系统界面需简介美观，操作应给予相应提示语，人机交互方式应符合大部分人的需求。	易用性
报告易理解	扫描报告需风格一致，漏洞描述详尽、图文并茂，提供友好型交互方式。	易用性

为应对误报过滤模型的迭代更新，并保证扫描工具的插件式扩展集成，本系统需要抽象接口，方便模型的更新和工具的集成，增强系统的可扩展性。为应对业务增长带来的服务器访问压力，本系统需做好水平扩展准备，以确保增加服务器数量可以有效减少负载。系统需及时进行数据备份，并同时进行日志记录，以保证服务宕机时可以快速查询相关数据以排错。本系统设计简洁美观的交互界面，以确保用户拥有较好的交互体验。同时，本系统通过机器学习过滤与众审专家审核实现人机协同漏洞扫描，从而保证系统整体低漏报低误报。

3.2.4　用例分析

根据上述系统功能性需求的相关分析，得到本系统的用例图如图 3.2所示。本系统核心用户分为平台管理员、普通用户和众审专家。总共涉及 6 个相关用例，分别为审核管理、漏洞审核任务、任务管理、执行漏洞扫描任务、发布众审以及查阅漏洞扫描报告。

平台管理员的用例为审核管理，包括扫描任务审核、众审任务审核以及众审专家身份验证审核。众审专家用例为漏洞审核任务相关，主要包括查看待审核任务列表、接受或拒绝分配到的审核任务、查看审核任务详情以及进行漏洞审核。普通用户用例主要包括任务管理、执行扫描任务、发布众审以及扫描报告相关。其中，任务管理主要包括扫描任务创建、扫描任务基本信息填写以及流程配置、任务发布，任务发布类型包括扫描任务和众审任务，普通用户发布的任务需要平台管理员进行相应审核才能执行。执行扫描任务包括漏洞扫描和误报过滤。发布众审包括需审核漏洞筛选、众审专家选择以及众审内容项自定义配置。扫描报告相关包括扫描报告查阅以及报告下载。

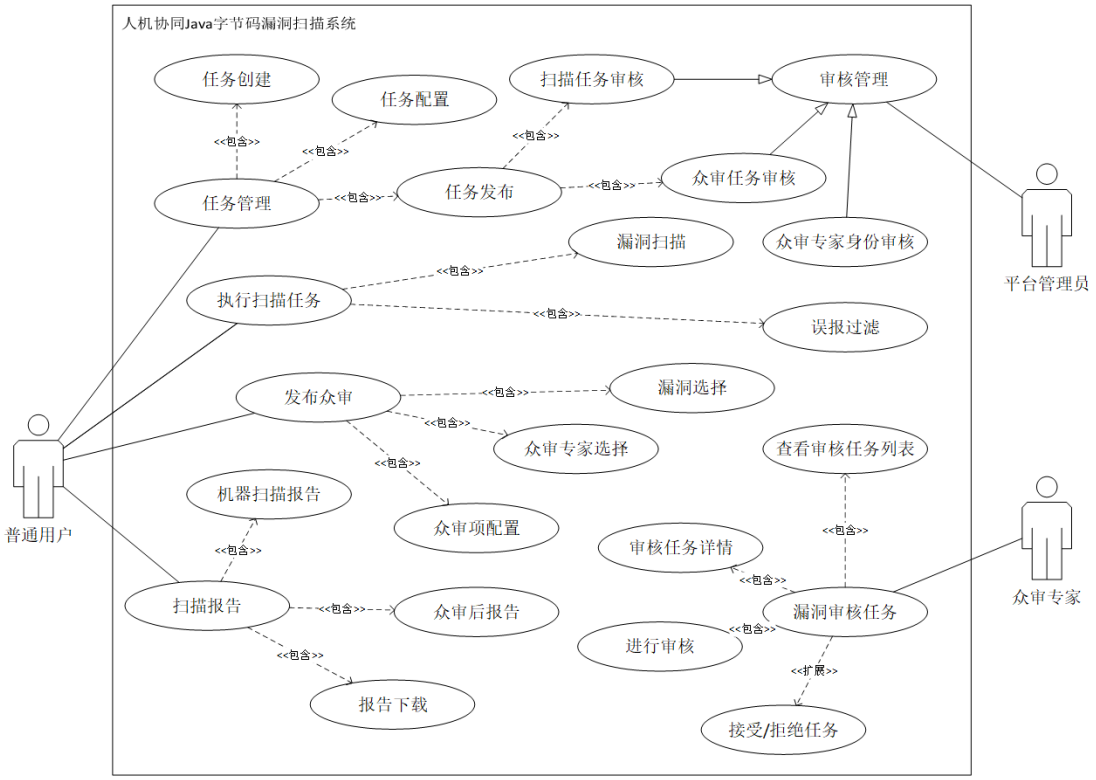

图 3.2: 系统用例图

接下来，本小节将就上述 6 个用例进行具体阐述并给出相关用例描述。

平台管理员的用例主要为审核管理，包括扫描任务审核、众审任务审核以及众审专家身份验证审核。其用例描述如表 3.3所示，当用户身份转换成众审专家或用户提交扫描任务以及用户发布众审任务时，平台管理员需要进行相关内容审核以决定是否允许任务执行。

众审专家的用例主要为漏洞审核任务相关，其用例描述如表 3.4所示。众审专家可以查看分配到的漏洞审核任务列表，之后根据自身情况选择接受或拒绝

表 3.3: 审核管理用例描述

用例 ID	UC1
用例名称	审核管理
参与者	平台管理员
用例说明	管理员进行相关内容审核
前置条件	管理员登录系统并被授权
基本事件流	1. 管理员进入任务审核页面，系统显示所有扫描任务，并且把待审核任务按时间顺序置顶显示； 2. 管理员点击“详情”查看任务详情； 3. 系统显示任务详情描述，并提供“同意”、“拒绝”、“返回”选项按钮； 4. 管理员点击“同意”，审核通过任务。
其它事件流	1a. 管理员进入用户审核页面，系统按时间升序显示所有众审专家； 3a. 系统显示众审专家申请详情，并提供审核选项； 4a. 管理员点击“拒绝”，审核不通过该任务。
后置条件	相关任务审核通过，进入后续执行阶段

该漏洞审核任务；当进行漏洞审核时，众审专家可以查看漏洞描述详情以及待审核内容项，从而进行审核内容的填写以及提交审核报告。

表 3.4: 漏洞审核任务用例描述

用例 ID	UC2
用例名称	漏洞审核任务
参与者	众审专家
用例说明	众审专家查看漏洞审核任务详情并进行漏洞审核
前置条件	众审专家登录系统并被授权，众审任务审核通过
基本事件流	1. 众审专家进入所有众审任务页面，系统显示所有众包审核任务，待执行任务按时间顺序置顶显示； 2. 众审专家点击“详情”查看众审任务详情； 3. 系统显示众审任务详情，并提供“接受”、“拒绝”、“返回”选项按钮； 4. 众审专家点击“接受”按钮，接受该漏洞审核任务； 5. 众审专家进行漏洞审核并提交审核报告。
其它事件流	4a. 众审专家点击“拒绝”，不接受该任务漏洞审核。
后置条件	众审平台汇总融合该任务的所有众审报告

普通用户用例主要包括任务管理、执行扫描任务、发布众审以及扫描报告相关。任务管理用例如表 3.5所示，用户创建扫描任务，上传带扫描项目字节码文件包，填写项目基本信息，并进行任务扫描相关选项配置（如是否需要机器误报过滤、扫描器数量及类型选择等），最后发布扫描任务，等待平台管理员审核。

表 3.5: 任务管理用例描述

用例 ID	UC3
用例名称	任务管理
参与者	普通用户
用例说明	普通用户进行任务创建、扫描选项配置以及任务发布
前置条件	普通用户登录系统
基本事件流	1. 普通用户进入发布扫描任务页面，系统显示任务发布引导； 2. 普通用户点击“上传文件”，并选择需扫描项目包进行文件上传； 3. 用户填写项目名称等基本信息，并选择服务项目为“Java 漏洞扫描”； 4. 用户进行扫描任务具体配置，如是否需误报过滤、扫描器数量等； 5. 用户提交扫描任务，系统显示“提交成功，等待审核”。
其它事件流	2a. 用户可直接选择“最近上传项目包”，无需重新上传文件； 4a. 用户选择默认配置选项，扫描器数量为 1，需要误报过滤
异常事件流	2b. 上传文件失败，系统提示错误原因并建议“重新上传”；
后置条件	该漏洞扫描任务等待平台管理员审核

执行扫描任务用例如表 3.6所示，根据任务创建时用户的扫描配置，选择一个或多个扫描器对提交项目进行漏洞扫描，若用户选用误报过滤服务，则扫描后将结果进行误报过滤。

表 3.6: 执行扫描任务用例描述

用例 ID	UC4
用例名称	执行扫描任务
参与者	普通用户
用例说明	根据用户任务配置，系统进行漏洞扫描以及误报过滤等流程执行
前置条件	普通用户创建扫描任务，并且管理员审核通过
基本事件流	1. 普通用户进入扫描任务列表页面，系统显示该用户所有扫描任务，并可根据状态进行筛选； 2. 用户点击“执行”，开始审核通过扫描任务的执行； 3. 系统根据用户配置进行漏洞扫描、误报过滤等流程； 4. 扫描任务完成后，用户可以在任务列表页面看到进度。
其它事件流	3a. 用户主动停止该扫描任务。
异常事件流	4a. 项目扫描失败，告知用户失败原因。
后置条件	生成漏洞扫描机器报告

发布众审用例如表 3.7所示，主要包括需审核漏洞筛选、众审专家选择以及众审内容项自定义配置。用户将扫描结果发布众审，选择需要众审的漏洞列表，同时选择分配众审任务的众审专家，并进行需要众审内容项的自定义配置，最后一键发布，等待众审结果。

表 3.7: 发布众审用例描述

用例 ID	UC5
用例名称	发布众审
参与者	普通用户，平台管理员，众审专家
用例说明	用户将机器扫描结果发往众审专家进行人工审核
前置条件	扫描任务已完成并生成扫描报告
基本事件流	1. 用户进入任务列表页面，系统按时间顺序显示所有任务以及任务进度； 2. 用户对完成扫描的任务“发布众审”，系统跳转至创建众审任务页面； 3. 用户勾选需审核漏洞、指定众审专家，并自定义配置审核内容项，最后一键发布众审； 4. 平台管理员查看众审任务详情，点击“同意”，审核通过该任务； 5. 众审专家“接受”众审任务，进行漏洞审核并提交报告。
其它事件流	3a. 用户使用默认配置发布众审，即审核所有漏洞、系统指派众审专家、审核项为是否误报； 4a. 平台管理员不同意该任务，则众审任务不通过； 5a. 众审专家“拒绝”分配的众审任务。
后置条件	该漏洞扫描任务等待平台管理员审核

扫描报告任务用例如表 3.8所示，主要包括漏洞扫描报告与众审后融合报告的查阅与下载。只执行漏洞扫描，则会生成机器扫描（并且误报过滤）后漏洞报告，众审完成后，将根据众审结果自动生成人机融合扫描报告。

表 3.8: 扫描报告用例描述

用例 ID	UC6
用例名称	扫描报告
参与者	普通用户
用例说明	普通用户在线查阅扫描报告，并下载报告进行离线分析
前置条件	扫描任务完成，众审任务完成
基本事件流	1. 普通用户进入扫描任务列表页面，系统按时间顺序显示所有扫描任务； 2. 用户点击“查看报告”，系统跳转至相应报告详情页面； 3. 用户进入报告详情页面，查看具体扫描结果； 4. 用户点击“下载报告”，系统创建下载任务并开始下载报告。
异常事件流	3b. 报告下载失败，系统提示错误原因。
后置条件	无

通过上述相关图表说明，本小节阐述了本系统涉及的 3 大用户角色：普通用户、平台管理员和众审专家，以及 6 个核心用例，并通过用例描述对每个用例进行了详细说明。

3.3　系统总体设计

根据上节需求分析得到的功能性需求、非功能性需求以及相关用例描述，本小节将详细阐述本系统的总体设计。首先，通过对系统的架构设计，展示并描述本系统的整体框架以及层次结构，从而划分出本系统各个模块以及各模块之间的职责与联系。其次，通过逻辑视图、开发视图、进程视图以及物理视图四个视角进一步详述本系统的总体设计。最后，介绍本系统涉及的持久化对象并阐述存储方案。

3.3.1　系统架构

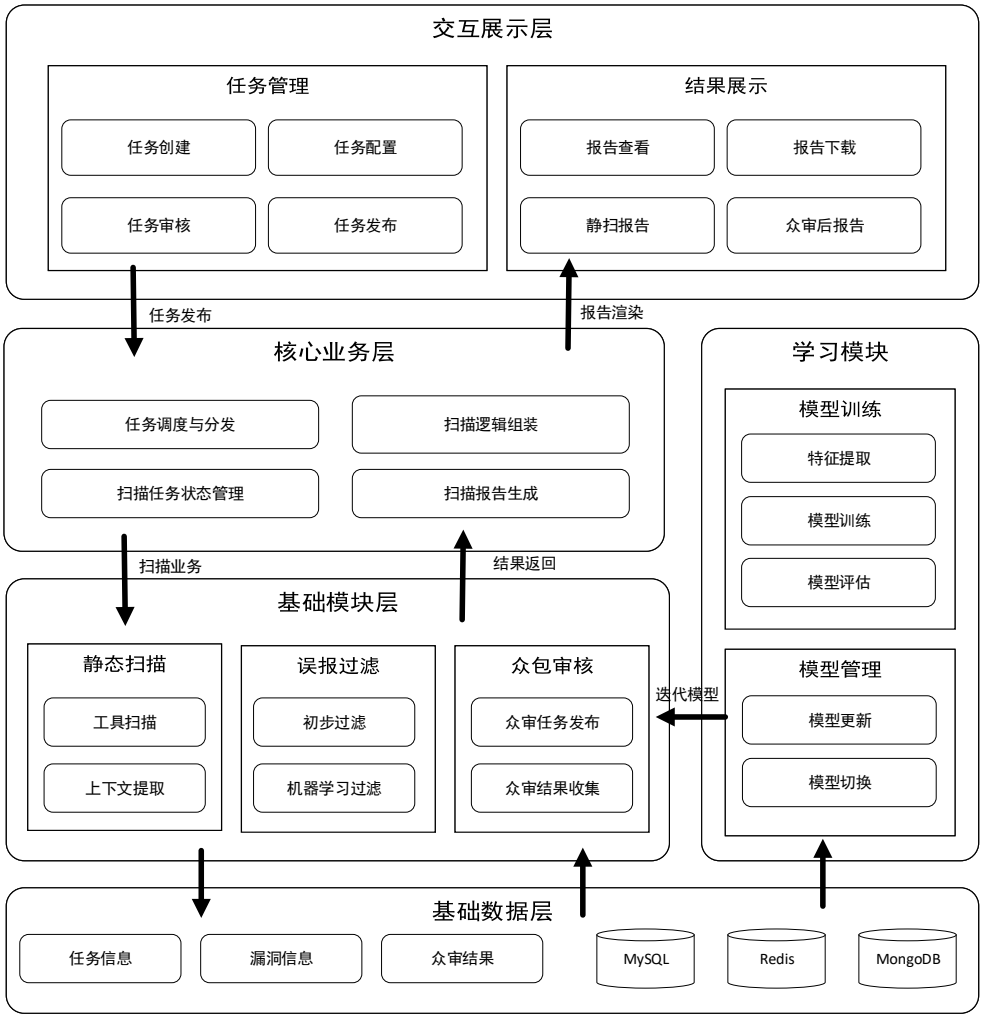

图 3.3: 系统架构图

图 3.3描述了人机协同的 Java 字节码漏洞扫描系统的架构图，包括与用户直接交互的交互展示层、服务端主要工作流程实现的核心业务层、职责单一可快

速更新组装的基础模块层、为误报过滤提供分类模型的学习模块以及为上述所有功能提供数据支撑的基础数据层。

本系统主要负责为开发者提供低误报高效率的漏洞扫描服务，包括创建配置以及调度分发漏洞扫描任务、根据用户相关任务配置组装漏扫业务逻辑、进行项目的漏洞扫描、误报过滤、众包审核、查看和下载生成的扫描报告、利用不断扩增的众审数据迭代训练过滤模型等相关功能。

交互展示层中，普通用户进入漏洞扫描任务的创建流程：上传待扫描的项目代码包，并进行是否需要误报过滤服务的选择；然后管理员审核该扫描任务，审核通过内自动开始扫描任务的执行。任务执行完成后，生成该任务的扫描结果报告供用户查阅与下载，报告内容包括：漏洞类型与风险等级统计、每个漏洞的相关描述、漏洞的相关代码的上下文、漏洞的具体位置信息、漏洞的示例以及示例推荐解决方案、漏洞的参考链接以及 CWE 相应标准等。扫描任务完成后，用户可以发布众审，众审完成后则会生成融合众审结果的扫描报告。

核心业务层中，通过 HTTP 请求为交互展示层提供相应业务功能。主要通过 Redis 缓存实现扫描任务状态的管理、通过基础模块层的模块细化实现逻辑组装、通过 Pug 编译引擎实现扫描报告的生成。

基础模块层是本系统的核心，静态扫描包括不同工具的扫描功能的具体集成，误报过滤包括多种误报过滤模型，众包审核包括众审任务信息的转换与发布以及众审结果的收集。该层通过每个具体功能的精细化与统一化，为核心业务层中具体扫描逻辑组装提供基础。实现可根据用户的相关配置自由选择扫描工具、过滤模型、是否众审等服务，同时为后续可能集成的工具、模型等提供了便捷性。

学习模块中，主要为误报过滤持续更新过滤模型。该模块将不断扩增的漏洞扫描结果以及专家审核结果作为训练数据集，通过字节码分析技术、特征提取并利用随机森林等分类算法训练过滤模型，选取效果提升明显的模型更换系统中使用的模型。通过学习模块，实现了过滤模型的迭代学习。

基础数据层，主要为系统提供基本数据支撑，包括任务基本信息、漏洞的数据信息以及众审服务反馈的信息。其中 Redis 主要用来实现任务状态的管理，MongoDB 存储漏洞结果以及众审反馈信息，MySQL 主要用来描述漏洞本身的原始信息。

按模块划分，系统主要分为三个模块：交互展示模块、漏扫核心模块以及迭代学习模块。其中交互展示模块为图 3.3中的交互展示层，漏扫核心模块包括图 3.3中的核心业务层和基础模块层，迭代学习模块为图 3.3中的学习模块，图 3.3中的基础数据层为整体系统提供基础数据支持。

从工程角度而言，本系统前后端分离，前端页面使用强大的 AngularJS 模板和较为成熟的 Bootstrap 组件库，使用方便的同时保持快速高效的渲染能力。其中，报告页面使用 Pug 模板引擎，其简介的语法可以保证报告快速迭代更新，提高开发效率。部分页面更新请求采用 AJAX 异步请求，以保证局部更新低延迟。系统服务端采用 Spring Boot 框架开发，其强大的可扩展性可以保证其他服务组件的快速集成，同时服务端向外提供 RESTful API，提高客户端使用便捷性以及服务端的可伸缩性。服务内部部分模块工具使用 Docker 进行容器化，以提升模块工具的可复用性和可扩展性，从而简化工具的升级迭代。

3.3.2　4+1 视图

Philippe Kruchten 早于 1995 年提出 4+1 视图的概念 [44]，从 5 个不同视角来描述软件体系结构，这 5 个视角结合起来才能反映系统的全部内容。本小节将从这 5 个视角对本文设计的系统进行相关描述，包括场景视图、逻辑视图、开发视图、进程视图以及物理视图。其中场景视图以用户角度分析系统的每个用例，即 UML 中的用例图，已在上节图 3.2中详细说明，这里不再赘述。

逻辑视图提供用户的最终视角，将系统分解成一系列功能抽象并展示各功能抽象之间的依赖关系，用来描述系统的功能需求。在面向对象程序设计中，类图用来描述系统的静态结果，包括类、类内部结构以及类之间依赖关系。因此，这里使用类图来描述系统的逻辑视图。本系统逻辑视图如图 3.4所示，本系统被分解成一系列功能关键抽象。

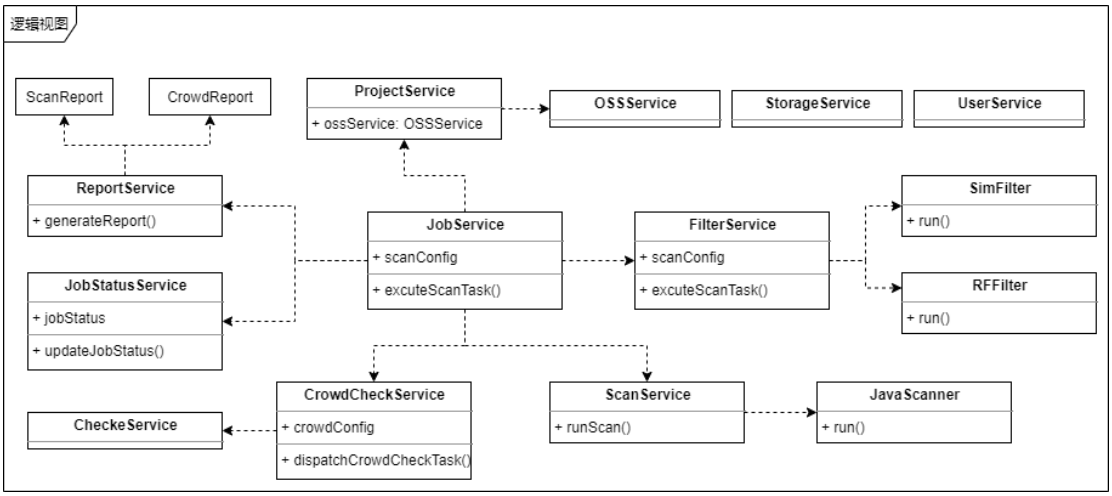

图 3.4: 系统逻辑视图

JobService 为核心，提供系统进行任务扫描流程的统筹调度安排工作。JobService 需要调度多个功能模块来完成扫描任务，其中，ProjectService 负责扫描

任务初始化，借助 OSSService 下载扫描项目包以及上传结果报告；ReportService 提供扫描报告和众审后报告渲染生成功能；JobStatusService 用于监控整个扫描流程，实时更新相关任务的扫描状态；CrowdCheckService 提供分发众包审核功能，调用 CheckService 对漏洞进行相关项审核；ScanService 提供多种类型 Java 扫描器对项目进行漏洞扫描；FilterService 提供多个过滤器来对扫描结果进行误报过滤，主要包括相似度过滤和随机森林过滤；而 UserService 提供用户登录注册、权限身份验证等基本功能。StorageService 提供整个系统存储相关功能，包括任务状态、任务基本信息、扫描结果以及报告信息等。

进程视图侧重系统的运行特性，描述系统的并发和同步设计。进程视图主要服务于系统集成人员，旨在解决进程、并发、同步以及通信等方面的问题。本系统的进程视图使用 UML 中活动图来表示，如图 3.5所示。

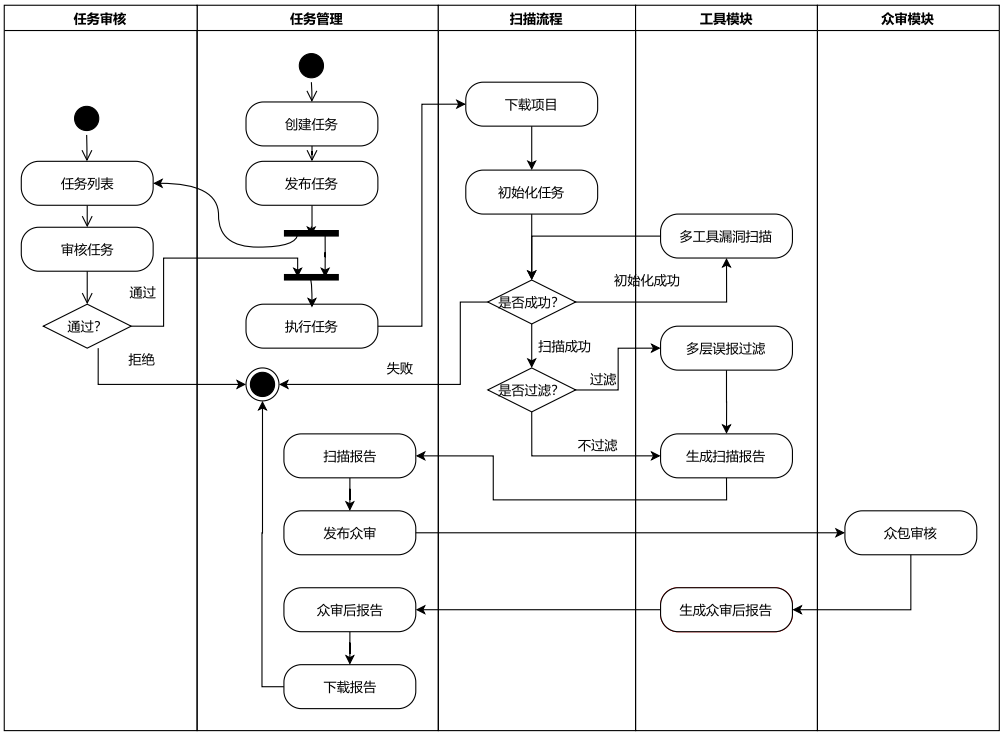

图 3.5: 系统进程视图

普通用户进入任务管理进程，创建、配置并发布扫描任务；当接收到待审核任务时，平台管理员对该任务进行审核。审核通过后，用户开始执行任务，任务流程交于扫描流程主线程，该线程负责扫描流程的各种调度。首先，下载带扫描项目并初始化工作环境；其次，调度工具模块中多种工具进行漏洞扫描；然后根据用户配置是否过滤，进行多层误报过滤；最后生成扫描报告。此后，用户可查

看扫描报告并发布众审，众审专家审核完成后，系统生成众审后报告，用户仍可查看并下载。

开发视图从开发者角度，描述软件在开发环境下的静态结构以及软件模块的组织和管理，并规范和约束开发环境的结构。图 3.6展示了本系统的开发视图。

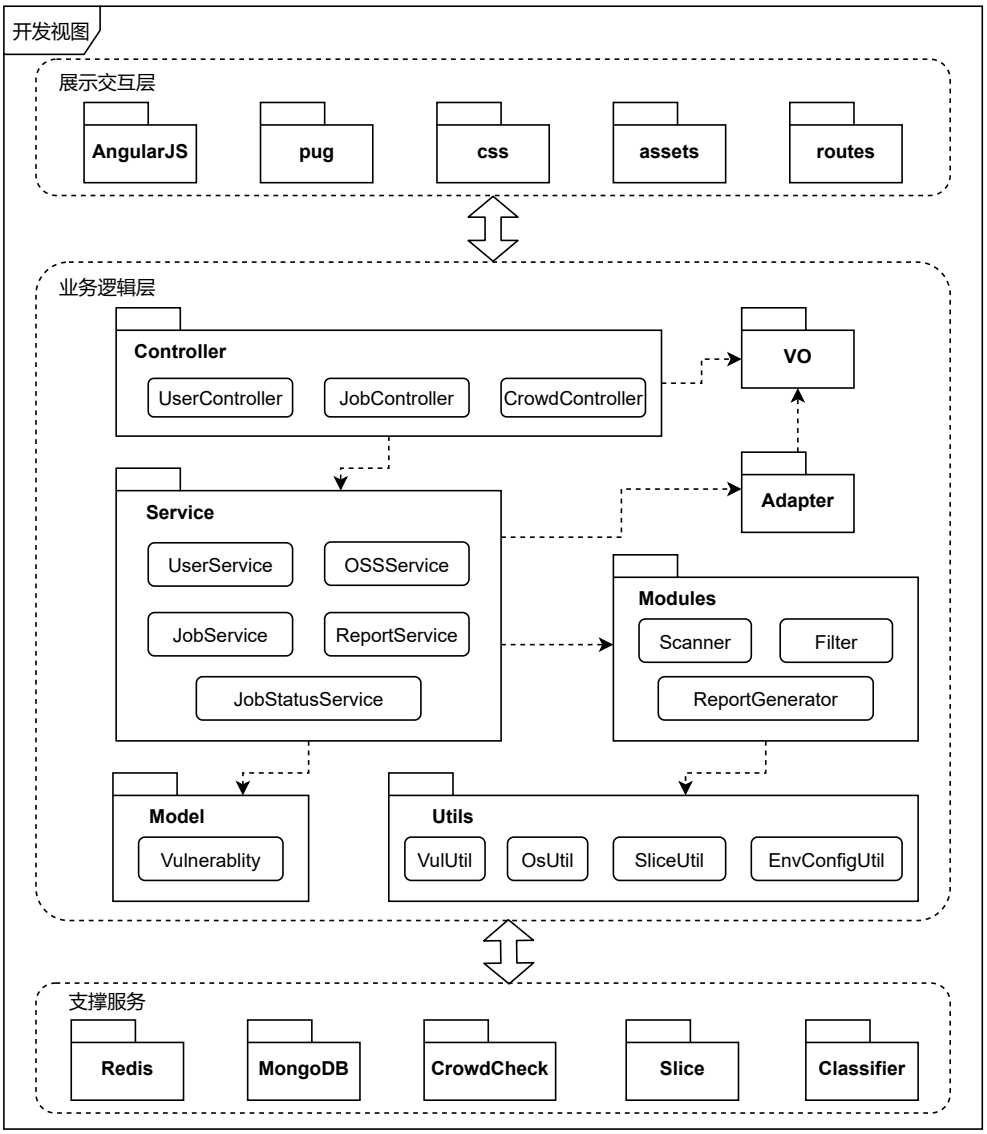

图 3.6: 系统开发视图

系统遵循典前后端分离 [45]，总体分为三层，展示交互层、业务逻辑层和支撑服务层。展示交互层主要包括用户交互页面的相关模板引擎和静态资源，其中 AngularJS 为主页面入口代码包，Pug 为扫描报告渲染相关代码包，css 为通用页面样式，assets 包主要存放页面所需相关静态资源，routes 包为路由数据目

录。业务逻辑层采用典型分层架构设计，Controller 包负责控制器管理，监听前端 HTTP 请求，启动线程进行相关业务处理并返回数据给前端。Service 包负责 Controller 层下发业务逻辑的具体执行。Modules 包主要提供系统核心基础模块，包括扫描器、误报过滤器以及报告生成器。Utils 包提供系统通用模块，如漏洞信息读取、系统命令执行、程序切片调用等。Model 和 VO 包是数据的不同表现形式，Adapter 包为数据转换提供相关适配器。支撑服务层为系统提供独立服务与数据支持，包括 Redis 缓存、CrowdCheck 众审服务、Slice 切片技术以及 Classifier 误报分类模型训练与评估。

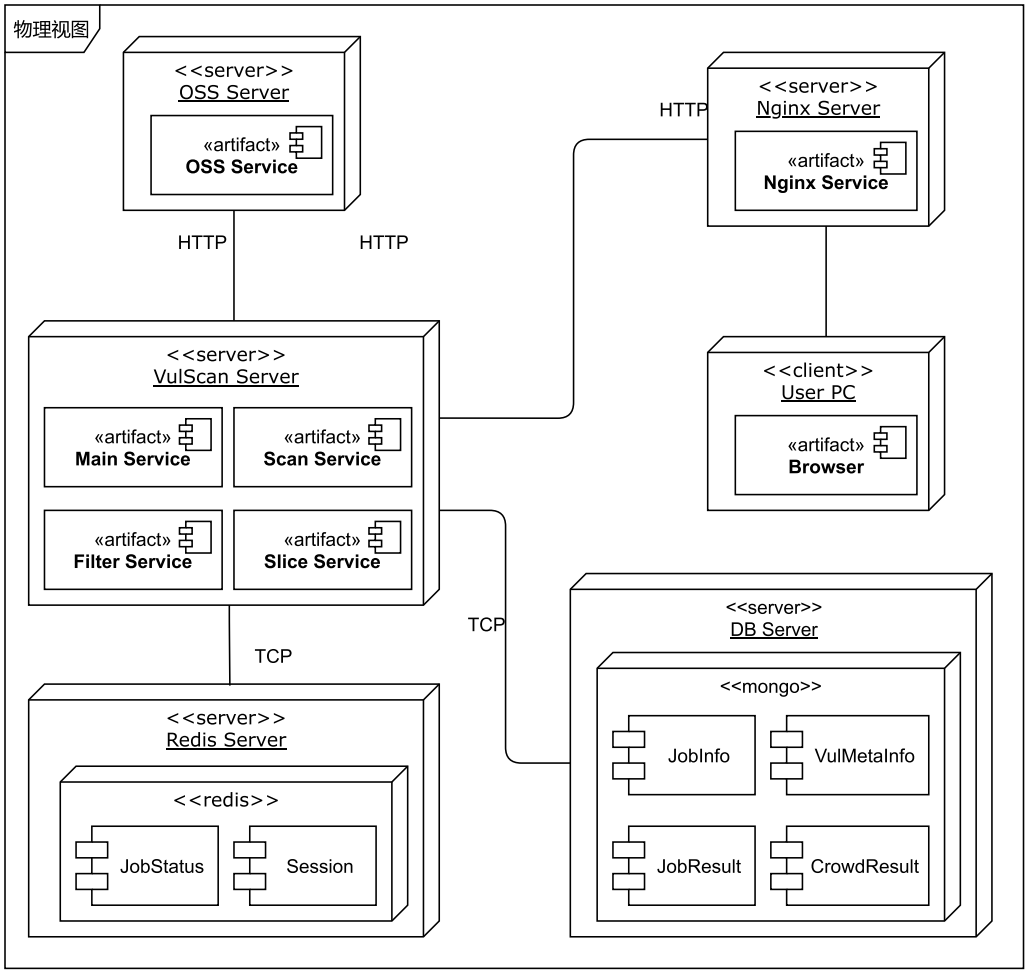

图 3.7: 系统物理视图

图 3.7展示了本系统的物理视图，该视图从部署角度，描述系统硬件配置，以解决系统的拓扑结构、系统部署、网络通信等问题。用户通过浏览器访问本系统，创建执行扫描任务以发起 HTTP 请求，该请求首先通过 Nginx 服务器进行负载均衡，按一定策略将请求转发到合适的应用服务器。Redis 服务器主要用于

存储 Session 信息与任务状态信息，保证扫描任务唯一性。扫描结果相关数据以 BSON 格式存储在 MongoDB 服务器上，便于大量存储与管理。OSS 服务器是阿里云对象存储服务器，主要用于存储待扫描项目包以及扫描后报告结果。

3.3.3　持久化对象设计

MongoDB 是一个高性能、开源、无模式的文档型数据库，与关系型数据库 MySQL 相比，其支持更多更复杂的数据类型，无需事先定义结构就可以创建记录，在数据存储与扩展方面有更大优势。本系统主要提供漏洞扫描服务，对外提供查询事务较少，需要存储复杂结构数据较多，因此，本系统采用 MongoDB 作为持久化数据库。

本系统涉及的持久化对象主要包括漏洞元数据信息、任务元信息、扫描结果信息以及众审结果数据，这四种对象均需存储在 MongoDB 中。接下来，将给出这四种持久化对象的详细设计。

表 3.9: TaskMetaInfo 主要字段

字段名	类型	描述
taskId	String	任务 id，唯一标识
taskType	String	任务类型，Java
taskUser	SystemUser	任务发布用户
downloadUrl	String	待扫描项目包地址
scanReportUrl	String	扫描完成后生成报告地址
crowdReportUrl	String	众审后生成报告地址

如表 3.9所示，任务元信息对象 TaskMetaInfo 存储任务 id、任务类型、提测用户、待扫描项目包地址、扫描后生成扫描报告地址以及众审报告地址。记录任务元信息保证任务及其扫描结果的唯一性以及可查询性。其中，SystemUser 是平台用户服务中统一用户对象；downloadUrl、scanReportUrl 以及 crowdReportUrl 均为 OSS 服务器上地址。

漏洞元数据信息对象记录了漏洞本质相关描述信息，表 3.10展示了漏洞元数据对象的主要字段。包括漏洞的唯一 id、漏洞名称、漏洞所属类型、漏洞风险等级、漏洞概述、漏洞详细描述、漏洞示例、漏洞参考以及漏洞解决方案。其中，漏洞风险等级 riskLevel 分为 5 个等级，分别为信息、警告、低危、中危和高危，中高危漏洞是开发者需要特别关注的漏洞。

表 3.10中 references、examples 和 solutions 分别涉及 Reference、Example 以及 Solution 对象。参考引用 Reference 对象包括参考 id、参考漏洞标准名称以及

参考链接。漏洞示例 Example 对象包括示例 id 和示例内容。解决方案 Solution 对象包括解决方案 id 和具体解决方案描述。如表 3.11、表 3.12和表 3.13所示。

表 3.10: VulnerabilityMetaInfo 主要字段

字段名	类型	描述
vulId	Integer	漏洞 id，唯一标识
name	String	漏洞名称
category	String	漏洞类型
riskLevel	Integer	漏洞风险等级
overview	String	漏洞概述
description	String	漏洞详细描述
references	List<Reference>	漏洞参考列表
examples	List<Example>	漏洞示例列表
solutions	List<Solution>	漏洞推荐解决方案列表

表 3.11: ReferenceInfo 主要字段

字段名	类型	描述
refName	String	参考漏洞名称
refUrl	String	参考链接

表 3.12: ExampleInfo 主要字段

字段名	类型	描述
example	String	示例内容

表 3.13: SolutionInfo 主要字段

字段名	类型	描述
solution	String	解决方案描述

扫描结果 ScanResult 对象存储漏洞扫描结果，表 3.14展示了扫描结果对象的主要字段。扫描结果对象主要包括任务 id、任务执行相关描述信息以及扫描结果漏洞列表。其中，任务信息 TaskInfo 如表 3.15所示，主要包括任务 id、提测人信息、扫描时间以及任务消耗时长。漏洞结果信息 Vulnerability 如表 3.16所示，与 VulnerabilityMetaInfo 相比，其增加了漏洞的上下文信息。

表 3.14: ScanResult 主要字段

字段名	类型	描述
taskId	Integer	所属任务 id
taskInfo	TaskInfo	任务信息
result	List<Vulnerability>	扫描结果漏洞列表

表 3.15: TaskInfo 主要字段

字段名	类型	描述
taskId	Integer	任务 id
user	SystemUser	提测人信息
scanTime	DateTime	执行扫描时间
costTime	Long	扫描任务消耗时长

表 3.16: 扫描结果 Vulnerability 增加字段

字段名	类型	描述
hashCode	String	漏洞实例 hash 值
location	String	漏洞产生位置
content	String	漏洞代码上下文

众审后结果 CrowdResult 对象存储漏洞众审信息，表 3.17展示了众审结果对象的主要字段。与 ScanResult 相比，增加了 crowdResult 众审结果概述，该数据结构由众审服务制定。其中漏洞结果列表中的 Vulnerability 对象增加了 checkContent 字段，如表 3.18所示，该字段提供审核内容项以及审核结果信息。

表 3.17: CrowdResult 主要字段

字段名	类型	描述
taskId	Integer	所属任务 id
crowdResult	CrowdResult	众审结果概况
result	List<Vulnerability>	漏洞列表

表 3.18: 众审结果 Vulnerability 增加字段

字段名	类型	描述
checkContent	Map<Object, Object>	审核项及审核结果

3.4　系统各模块设计

上文已从需求分析、逻辑视图、进程视图、开发视图和物理视图五个角度分析了本系统的整体结构体系，并将系统划分成了不同的服务模块。本节将结合具体业务需求对划分的服务模块进行进一步设计，包括交互展示模块、漏扫核心模块以及迭代学习模块。

3.4.1　交互展示模块设计

交互展示模块主要用于接收用户请求转发至服务端以及从服务端获取请求结果渲染页面，展示给用户，即本系统直接与用户接触的模块。该模块主要包括用户任务管理流程以及报告生成支撑服务。

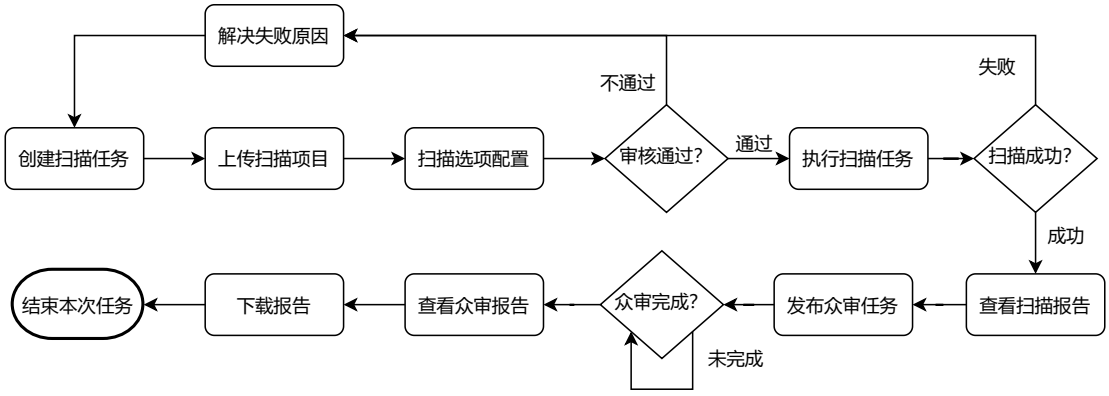

图 3.8: 交互展示模块流程图

交互展示模块总体流程图如图 3.8所示，用户创建扫描任务，上传待扫描项目包至平台 OSS 服务器上，此时可以快速选择上次上传项目包，当选择最近项目包时，无需重新上传。之后，用户填写任务基本信息以及扫描选项简单配置，包括项目名称、相关描述、是否误报过滤、扫描任务类型（C++/Java）等。任务创建成功后，需等待平台管理员审核，审核通过后，用户可以执行该任务。任务扫描执行完成后，用户可以登录系统查看生成的扫描报告，同时可以发布众审任务，发布众审页面提供查看漏洞报告信息、编辑漏洞报告信息、定制审查选项等功能。定制审核选项功能支持对于指定漏洞项设置审查项，并设置审核标签和数据说明，目前支持文本框、单复选框、多选框等基本内容项。众审结束后系统生成众审后报告，用户可以查看并下载结果报告。至此，一次扫描任务流程结束。对于每个步骤中出现导致流程中断的错误，系统会及时给予用户相关反馈并协助用户重新创建发布任务。

为了更好为用户提供静态报告在线查阅与下载功能，本系统需设计完成报告页面渲染工具，独立于系统前端页面。该工具适配于不同种类、不同语言的扫描器产生的结果，其输入为生成的 json 格式数据，输出是静态页面包（包括 html、js、resource 以及 css），该静态页面包将部署在 OSS 中以方便用户在线查看以及下载。

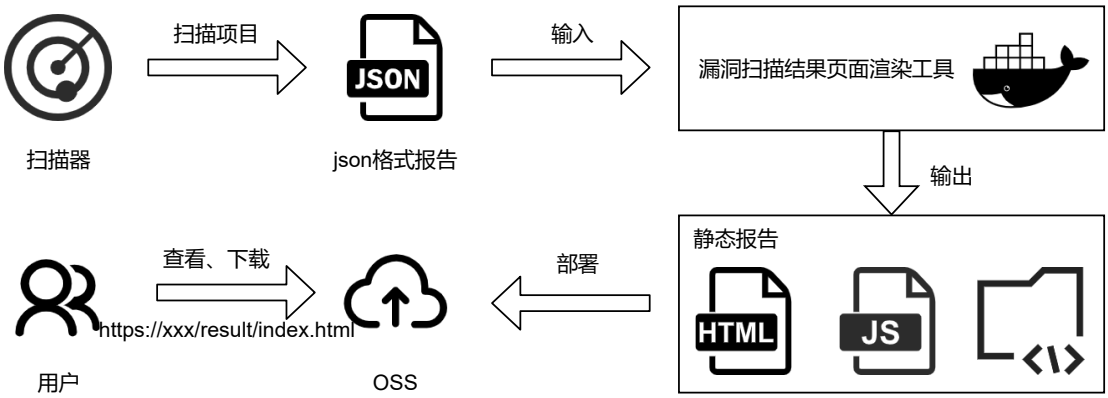

图 3.9: 报告渲染器工作流程图

图 3.9展示了报告渲染器工作流程，扫描器得到扫描结果转换成指定格式 json 数据，然后调用 pug 编译命令将页面模板与 json 数据融合渲染成报告页面。生成的静态报告页面会直接部署到 OSS 中，用户通过 OSS 中部署好的链接访问静态页面查看报告结果。静态报告结果时具有交互性质的网页应用，用户可以进行点击、筛选等交互。下载静态报告即下载整个报告文件夹，用户可以本地点击查看。其中，该报告渲染器进行 Docker 化，方便其部署安装使用，直接调用命令即可将数据渲染成页面并部署生成的静态报告页面。

本系统需调用多个扫描器、过滤分类器以及页面渲染工具等基础模块服务，有的服务耗时比较长并且可能同时调用多个服务，并且系统需同时处理多个扫描任务，为保证系统响应时间短、为用户提供更好的用户体验，本系统采用任务队列来处理所有服务调用。任务队列一般采用生产消费模型，包括生产者、任务处理中间方以及任务消费者，其中生产者负责生产任务，中间方负责接收任务处理请求，对任务进行调度并将任务分发给消费者处理。Celery 是一个 python 编写的分布式任务调度模块，其架构简洁且具有丰富的扩展性，适用于构建分布式 Web 服务。因此本系统使用 Celery 进行任务分发调度，并使用 RabbitMQ 作为 Celery 的消息代理任务队列 [46]。

3.4.2　漏扫核心模块设计

漏扫核心模块是本系统的核心模块，主要提供漏洞扫描、漏洞相关代码上下文提取、误报过滤以及众包审核服务。

漏扫核心模块架构设计如图 3.10所示，其主要功能是完成扫描任务的整个流程，为用户提供误报率低的漏洞报告。扫描任务执行主要流程是：首先，从 OSS 对象存储服务器上下载项目包，并调用对应类型扫描器对项目包进行静态扫描，得到初始漏洞结果；其次，通过漏洞定位，使用基于 Joana 程序后向切片技术从字节码文件中提取漏洞相关代码上下文信息；然后，根据用户任务配置，对初始漏洞列表进行多层误报过滤，并采用不同策略对不同等级误报进行相应处理；之后，发布众审得到众审专家对结果漏洞的审核意见；最后，使用报告渲染器生成相应扫描报告，并将报告上传至 OSS 中便于用户查阅下载。

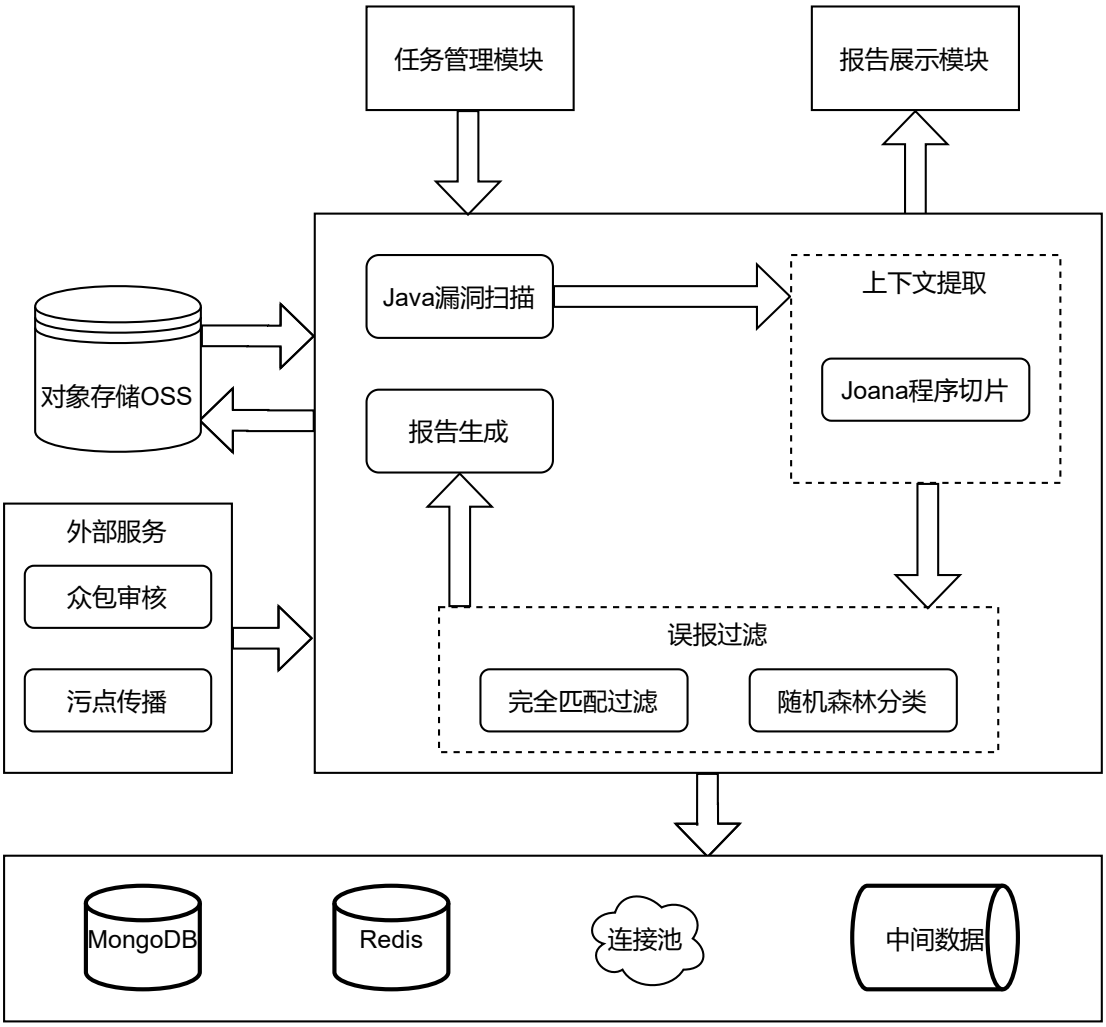

图 3.10: 漏扫核心模块架构图

其中，不同编程语言项目使用不同的漏洞扫描器进行漏洞扫描，本文主要描述 Java 字节码漏洞扫描，因此选取覆盖面广且持续维护更新的 SpotBugs 及其安全漏洞扫描插件 Find-sec-bugs 作为漏洞扫描器。原始扫描工具可扫描漏洞列表共计 10 大类型 593 项，本项目根据实际需求结合实践项目经验，只关注于安全性类漏洞，手动去除一些实践中以及不成熟的漏洞模式，并将其重新归类成常见漏洞类型共计 146 项。为保证漏洞的标准性，对这 146 项漏洞模式进一步人工审查，使用业内公认标准 CWE 将其标准化；并且根据 CWE 标准描述，对漏洞示例代码以及解决方案进行扩充，构建适合本系统的漏洞模式库。现有 Java 扫描工具覆盖漏洞列表统计如图 3.11所示。为方便后续添加多工具融合扫描，本系统将扫描工具设计成可接入 Docker 化插件工具，制定统一接入标准输入输出，并实现 start、status 以及 stop 三个操作原语。扫描工具与主系统平台解耦，提高扫描器可扩展性。

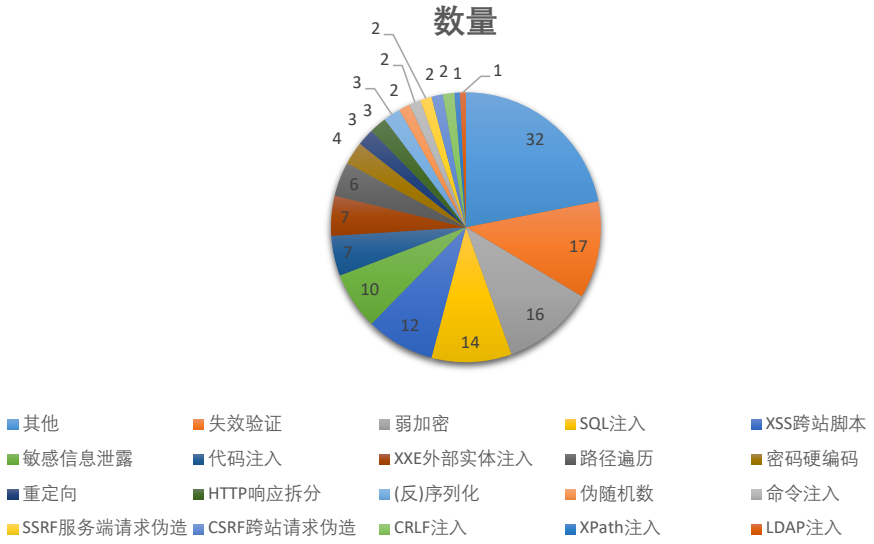

图 3.11: 漏洞模式列表统计图

Java 字节码文件是 Java 编译器编译 Java 源码产生的目标文件，其由 8 位字节的二进制流构成，各个数据项按顺序紧密从前向后排列 [47]。Java 字节码文件包含类的所有信息，扫描工具通过分析字节码二进制流对项目进行漏洞扫描。为更好将漏洞信息展示给众审专家以及用户，本模块使用 Joana 切片工具，根据扫描器对漏洞的定位提取漏洞代码相关上下文信息。SSA（静态单赋值形式）是中间代码形式的一种，它可以提供更精细的控制流图，同时编码控制流信息和数据流信息。本模块使用基于 Joana 切片工具从定位行到函数入口对漏洞相关代码进行后向切片，并将切片结果格式化为类 SSA 形式，以此提供漏洞上下文信息。此外，本文还将使用基于污点传播的外部程序切片工具分析获取漏洞污点

传播树，进一步辅助众审专家以及用户理解漏洞详情。

误报过滤部分采取多种策略对扫描漏洞结果进行误报过滤，目前主要使用两种策略：首先，若该漏洞之前已被人工标记为误报，则直接将该漏洞标记为误报，此为完全匹配过滤，以保证专家审核结果可以在下一次扫描相同项目时得到利用；其次，使用 N-gram 语言模型对漏洞上下文进行特征提取，并使用随机森林分类器对漏洞进行正误报预测，过滤误报漏洞。通过上述误报过滤机制，总体上直接过滤掉高误报漏洞，并将低误报漏洞归类为疑似误报漏洞，最终提供漏洞列表给用户，并标记出疑似漏洞以供用户参考。此部分需设计出抽象过滤器，每种策略提供统一接口，方便过滤器增加与更新，保证系统可扩展性。

众包审核服务是众审平台提供的外部服务，用户可将本系统扫描漏洞结果一键发布众审，并在众审平台进行审核漏洞列表配置以及审核项自定制。平台为众审专家提供漏洞详细信息描述，包括漏洞名称、类型、风险等级、示例、解决方案、参考链接、上下文以及部分辅助切片信息，并智能推荐相似漏洞审核建议，以辅助众审专家进行漏洞审核。众审结果需及时反馈给本系统，一方面生成融合报告提供给用户，另一方面扩增误报过滤模型数据集。

3.4.3　迭代学习模块设计

迭代学习模块是本系统的重要辅助模块，包含数据集预处理、特征提取、模型训练以及模型评估等子模块，主要为漏扫核心模块持续提供准确率高的误报过滤分类器，也是人机协同的重要体现之一。

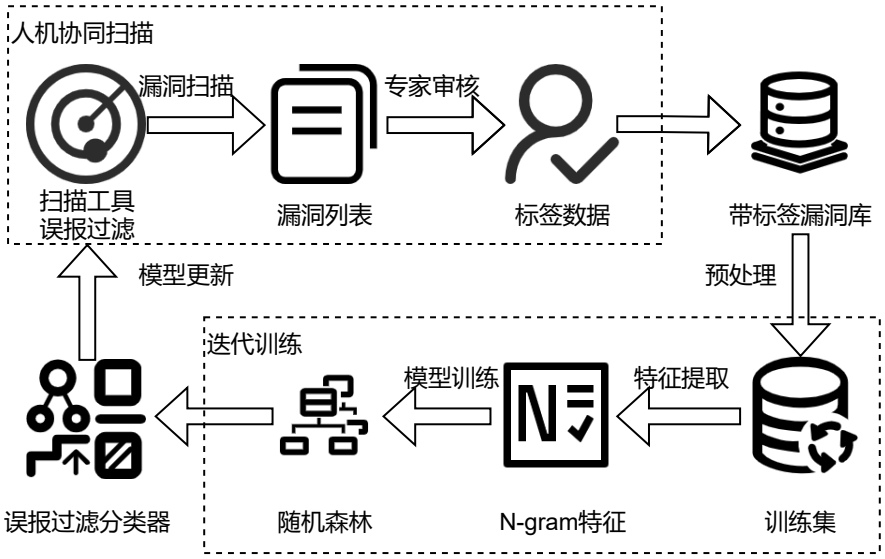

图 3.12: 模型训练流程图

迭代学习模块整体流程图 3.12如图所示，主要可分为两个重要步骤，数据

集扩充和模型训练。在数据集扩充步骤，首先通过本系统漏洞扫描工具得到漏洞列表，即训练集。再通过众审平台专家审核得到是否误报标签，即得到训练集对应的标签值。本系统在为用户提供扫描服务同时，可以不断扩增漏洞库，提供大量真实数据集。在模型训练阶段，首先将带标签漏洞库中数据进行预处理，主要对漏洞上下文进行分词、去除停用词，并使用统一规范形式表示上下文代码中变量名等无用信息，从而得到符合要求的训练集。然后，使用 N-gram 语言模型对漏洞上下文进行特征提取，首先对上下文内容进行分词，得到上下文序列；再对上下文序列进行 gram 切分，得到 gram 频度列表，并选择频度大于设定阈值的 gram 片段作为特征向量；最后，每个 gram 片段就是一个维度，从而得到特征向量表。最后，基于随机森林分类模型训练误报过滤分类器，并将分类器更新到扫描系统中，从而实现迭代学习过程。

综上，本问实现人机协同 Java 字节码漏洞扫描系统，其核心是利用专家审核和机器过滤不断扩增真实项目数据集，再使用扩增数据集持续迭代训练误报过滤分类器，从而提供低误报低漏报的漏洞扫描服务。

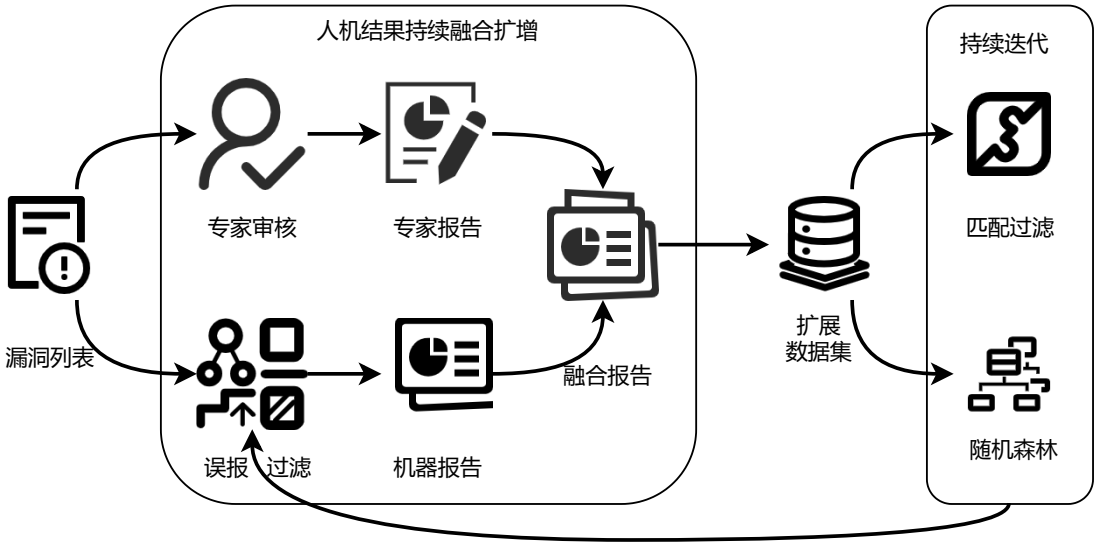

图 3.13: 迭代学习示意图

图 3.13展示了本系统通过人机协同实现迭代学习训练的过程。漏洞扫描工具产生的漏洞列表分别分发到众审专家以及误报过滤器中，专家与机器均对漏洞进行正误报判断，分别生成相应的专家报告和机器报告。系统基于一定策略对两份报告进行归纳，在生成融合报告的同时，持续扩增带标签漏洞数据集。该数据集一方面提供匹配过滤的对照集，实现实时误报过滤；另一方面，为随机森林分类模型提供扩充训练集，不断提升分类效果，从而实现周期性误报过滤效

果提升。此二者通过实时和周期性不断完善误报过滤器，从而实现人机协同迭代训练的漏洞扫描系统。

3.5　本章小结

本章首先概述人机协同 Java 字节码漏洞扫描系统的整体流程，由此分析阐述本系统的功能性以及非功能性需求，并通过相关用例描述加以详细说明。然后，介绍系统总体设计和系统架构图，并从用例图、逻辑视图、进程视图、开发视图以及物理视图多角度拆分与分析系统架构。之后，简要介绍本系统持久化对象设计。最后，进行系统功能模块拆分，并分别对交互展示模块、漏扫核心模块以及迭代学习模块三个核心模块进行详细设计描述。

第四章　详细设计与实现

本章基于第三章对系统的需求分析与架构设计，重点阐述交互展示模块、漏扫核心模块以及迭代学习模块三大核心模块的详细实现。本章将以时序图描述各模块关键组件的调用过程，并通过核心类图和关键代码阐述各模块具体实现，最后将展示本系统的实现界面。

4.1　交互展示模块

交互展示模块直接与用户交互，是本系统的前端页面模块，主要包括任务管理、任务分发以及报告页面渲染器子模块。本节将对这三个子模块的实现进行详细说明。

4.1.1　任务管理与分发的实现

任务管理子模块是作为本系统的门户，为用户提供扫描任务创建、任务配置、任务发布、任务执行、发布众审以及任务报告查阅功能。此模块涵盖所有用户操作页面，是系统主要功能对外的展现。

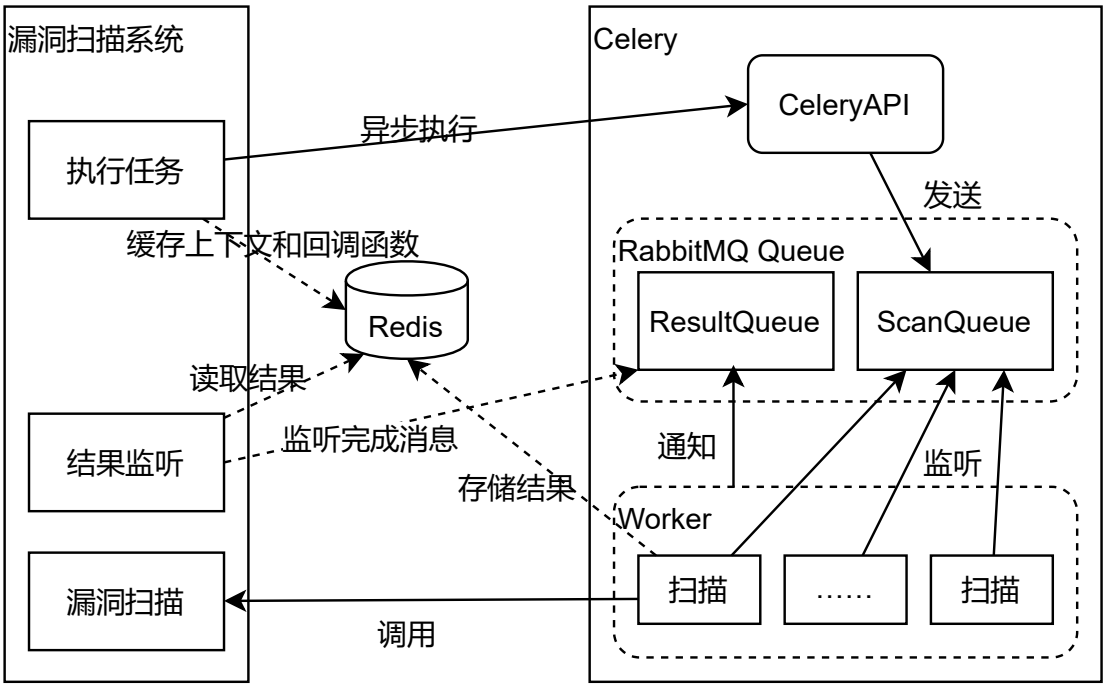

图 4.1: 任务分发框架图

漏洞扫描根据项目大小以及漏洞数量需要不同的扫描耗时，有些扫描任务耗时较长，并且需保证多任务并发执行的可用性。本系统使用异步任务调度来分发执行扫描任务，用户执行任务后无需等待，任务完成后会直接通知用户，提高系统整体吞吐量与响应效率，带给用户良好的交互体验。

本系统使用 Celery 作为异步任务调度工具，使用 RabbitMQ 作为中间消息代理，使用 Redis 用于缓存上下文和回调函数以及中间结果，从而实现对扫描任务的异步调度执行。其处理框架如图 4.1所示，执行扫描任务时，系统首先对任务工作环境进行初始化；然后根据扫描语言类型以及相关任务配置，将当前上下文以及任务完成后的回调函数缓存到 Redis 中。之后，系统将任务请求及相关配置发送给 CeleryAPI，此时 CeleryAPI 将任务转存至 RabbitMQ 的任务队列。Celery 中消费者持续监控任务队列，并根据任务配置调用相应扫描工具执行队列中的新任务；当任务执行完成，Celery 将发送完成消息到 RabbitMQ 中的结果队列，并将执行结果存储到 Redis 中。最后，系统监听到结果队列中的新消息，取出 Redis 中对应结果，并执行对应的回调函数，执行相应回调逻辑。

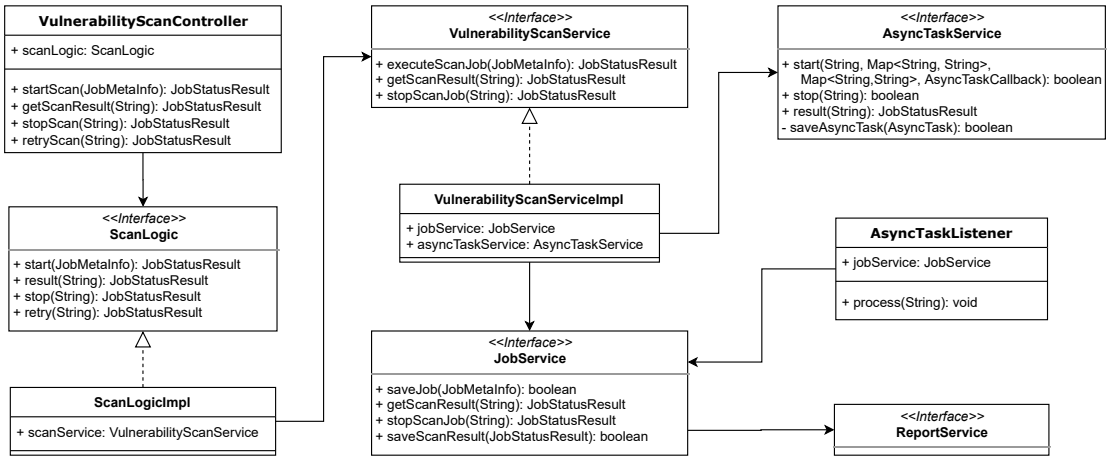

图 4.2: 任务分发子模块核心类图

图 4.2展示了本子模块的核心类图设计，VulnerabilityScanController 是对外提供接口的控制器，主要提供执行任务、停止任务以及获取任务结果等功能。VulnerabilityScanService 利用 JobService 和 AsyncTaskService 完成任务的异步执行。其中 JobService 主要提供任务信息的存储、任务结果的存储以及任务结果的读取服务。AsyncTaskService 将扫描任务请求转发至 Celery 异步任务队列中，并配以相关参数辅助执行，同时还将缓存上下文与回调函数。AsyncTaskListener 用于监听 Celery 异步队列中的结果队列，当监听到新消息时，从缓存中读取相关回调函数并继续任务下一流程。ReportService 主要提供报告渲染服务。

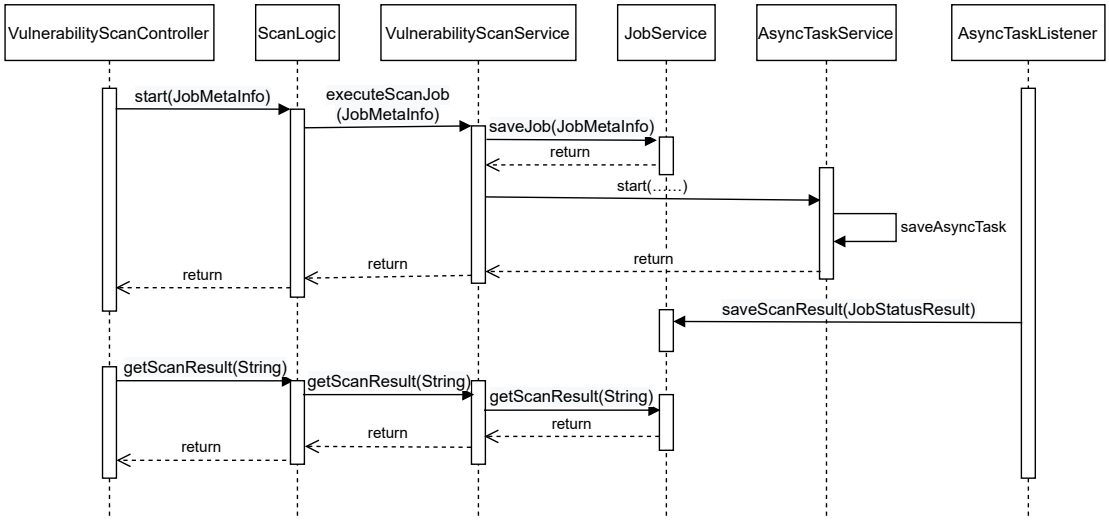

图 4.3: 任务分发子模块时序图

任务分发子模块时序图如图 4.3所示，当接收到执行任务请求时，VulnerabilityScanController 调用 ScanLogic 的 start 方法启动整个漏洞扫流程。ScanLogic 首先会通过私有方法初始化工作环境，再调用 OSSService 相关方法下载待扫描项目，最后调用 VulnerabilityScanService 中的 executeScanJob 方法发起漏洞扫描。VulnerabilityScanService 首先调用 JobService 中的 saveJob 方法将任务信息存储至数据库，再通过调用 AsyncTaskService 中的 start 方法将扫描任务下发给 Celery 异步任务队列。当扫描任务完成，Celery 将会发送消息到结果队列，AsyncTaskListener 用于监听结果队列，并将新消息对应的扫描结果通过 JobService 存储至数据库。当 VulnerabilityScanController 收到获取扫描结果请求时，会通过 ScanLogic 和 VulnerabilityScanService 最终调用 JobService 中的 getScanResult 方法，返回该任务的结果对象。若扫描任务已完成，则返回生成报告在 OSS 中的 url 地址；否则，则返回未完成状态。

本模块核心为任务执行与异步调度，故只展示任务异步执行与异步任务监听的部分关键代码，分别如图 4.4和图 4.5所示。AsyncTaskService 中的 start 方法首先根据相关参数以及扫描任务类型名称初始化 AsyncTaskDTO 对象以存储异步任务相关信息，并设置异步任务结果队列名称；然后通过 HTTP 请求将任务发送至 Celery 中；最后缓存上下文以及回调函数。AsyncTaskListener 中通过 Springboot 集成的 RabbitMQ 监听注解设置结果队列监听函数，当结果队列中产生新消息时，将执行监听函数。监听函数后续处理主要先将结果读取出来并存储至数据库，然后从 Redis 缓存中加载上下文与回调函数，并执行对应回调函数后续操作。后续操作主要为误报过滤以及静态报告渲染生成。

```
public boolean start(String name, Map<String, String> params,
      Map<String, String> context, AsyncTaskCallBack callback) {
  AsyncTaskDTO asyncTaskDTO = new AsyncTaskDTO();
  ……
  asyncTaskDTO.setQueueName(rabbitMQConfigure.getResultQueue());
  HttpEntity<AsyncTaskDTO> httpEntity = new HttpEntity<>(
         asyncTaskDTO, getHeader());
  String url = generateUrl("start");
  rt.postForObject(url, httpEntity, AsyncTaskStatus.class);
  ……
  putContextAndCallback2Cache(asyncTaskDTO, callBackClass);
  return true;
}
```

图 4.4: 任务执行关键代码

```
@RabbitListener(queues = {"#{rabbitMQConfigure.getResultQueue()}"})
public void process(@Payload String result) {
  ……
  JSONObject jResult = JSONObject.fromObject(result);
  JobStatusResult statusResult = new JobStatusResult();
  statusResult.setCode(jResult.getInt("code"));
  statusResult.setMessage(jResult.getString("message"));
  statusResult.setData(jResult.getString("data"));
  jobService.saveScanResult(statusResult);
  ……
  return;
}
```

图 4.5: 结果队列监听关键代码

4.1.2　报告前端渲染实现

静态报告页面需适配不同种类、不同语言的扫描器产生的结果，当输入指定格式的 json 数据时，可以通过相关编译命令将事先设计好的页面模板与 json 数据融合并渲染成静态报告页面包，此页面包中包括 html 页面、css 样式、js 页面相关逻辑以及 resource 静态资源，最后生成的页面包将部署在 OSS 服务中。为保证报告页面与系统主页面松耦合，静态报告页面需与主页面分离，报告渲染器将每个任务的结果均渲染生成对应的报告页面包存储在 OSS 上，方便用户在线查看与下载。其核心类图如图 4.6所示，其中 ReportLogic 承接控制器入口逻辑，主要有获取在线报告 url 和下载报告 url 两个方法。ReportService 是报告服务核心，主要提供获取报告、生成报告、保存报告等功能。ReportDao 是报告数

据连接层，提供数据库报告数据的增改查服务。ReportAdapter 用于将扫描结果 JobStatusResult 转换成 ReportData 以生成前端页面包。MongoAPIUtil 为封装好的 MongoDB 接口，为系统提供统一访问 Mongo 数据库功能。

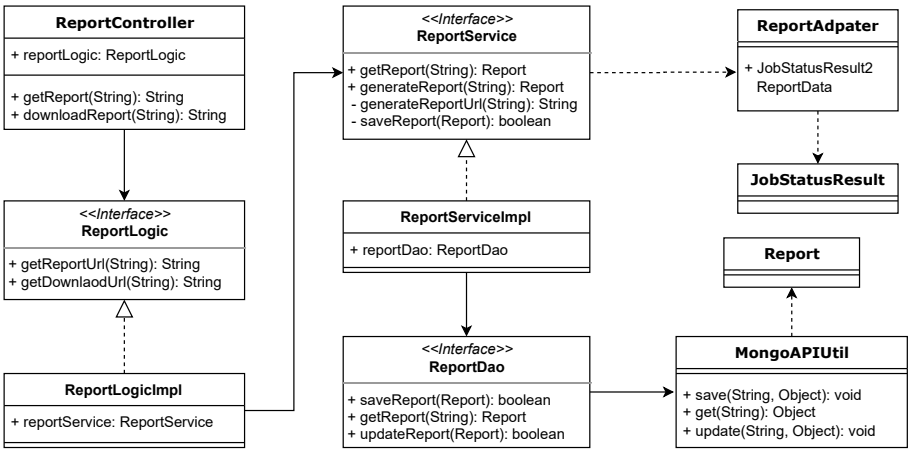

图 4.6: 报告渲染核心类图

报告渲染器生成报告时序图如图 4.7所示。ReportController 向外提供在线查看以及下载报告接口，查看报告时，ReportLogic 调用 ReportService 中的 getReportUrl 以获取报告在 OSS 上地址。ReportService 首先查询 MongoDB，若数据库中存在报告记录，则直接返回报告地址；否则，调用自身方法 generateReport 通过报告渲染器生成报告，并将新生成报告存储至数据库中，最后返回报告地址。前端跳转至报告 url，用户即可在线查看报告。

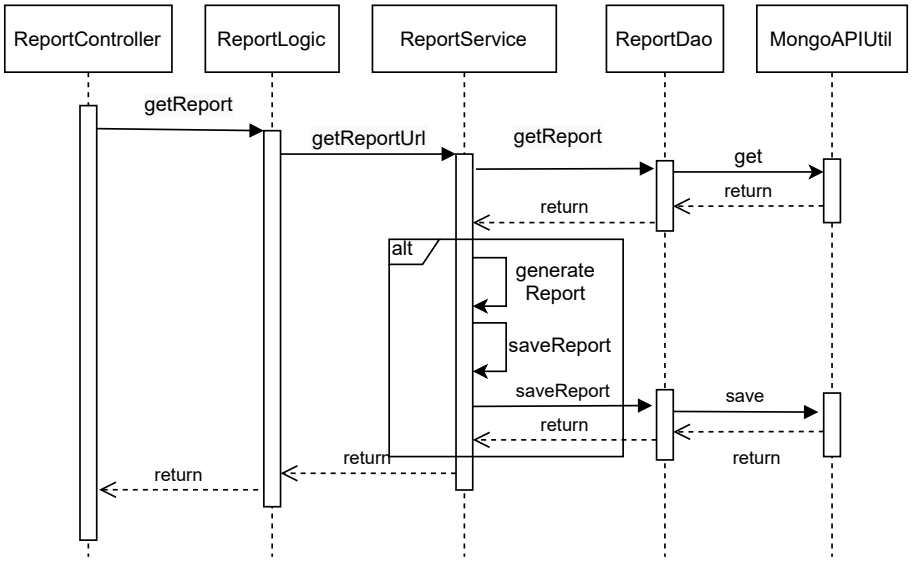

图 4.7: 查看报告时序图

图 4.8展示了调用报告渲染器生成报告的核心代码，ReportService 的 generateReport 方法首先将轻量级报告渲染器复制到当前任务工作环境，然后组装编译命令，之后通过 OsUtil 命令行封装方法执行生成报告指令，最后保存生成的静态报告信息并将生成报告上传至 OSS。

```
1  public void generateReport(String jobId) {
2    String jobResult = jobService.getJobResult(jobId);
3    String builderPath = copyBuilderToPath(jobId, Consts.REPORT_NAME);
4    FileUtil.writeDataToJsonFile(jobResult, builderPath +
5          REPORT_JSON_UPLOAD_PATH, REPORT_JSON_FILENAME);
6    String commands = COMMANDS + builderPath + Consts.REPORT_NAME;
7    OsUtil.runCommand(commands);
8    String url = ossService.uploadFileToTraceDir(new File(builderPath
9          + Consts.REPORT_NAME), jobId, Consts.REPORT_FILE_NAME);
10   saveReport(jobId, url);
11   ……;
12 }
```

图 4.8: 报告生成关键代码

4.2　漏扫核心模块

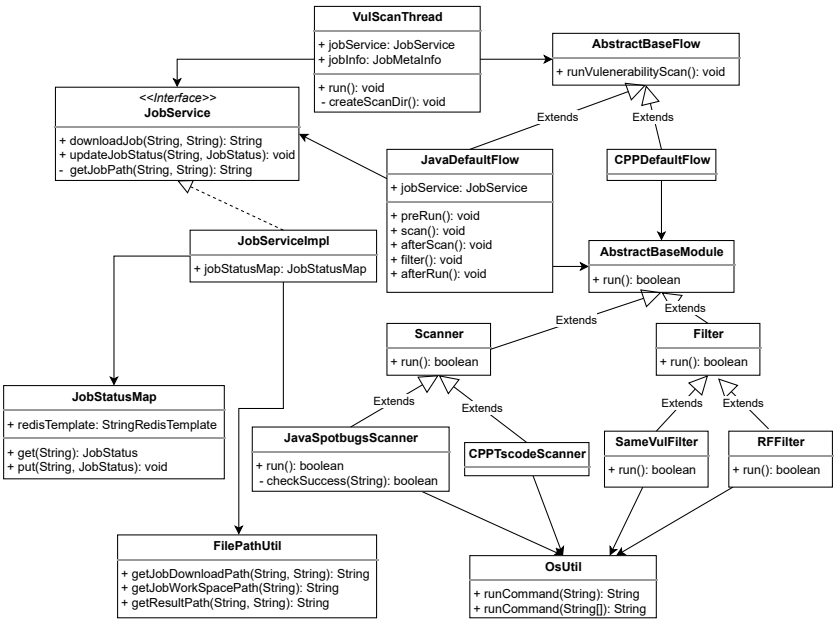

图 4.9: 漏洞扫描模块核心类图

漏扫核心模块不与用户直接交互，当异步任务消费者监听到扫描任务时，便会开始执行该模块。该模块按照工作流程模式实现，其核心类图如图 4.9所示。其中，VulScanThread 为任务扫描异步线程，扫描任务执行入口为其 run 方法。

AbstractBaseFlow 为扫描流程抽象类，其 runVulenerabilityScan 方法规范了任务执行抽象流程，即 preRun、scan、afterScan、filter 以及 afterRun。所有扫描流程均继承自抽象流程类，为扫描任务提供统一接口，方便管理与扩展。JavaDefaultFlow 是 Java 字节码漏洞扫描的默认扫描流程，其每一步均调用工具模块中的对应工具完成。AbstractBaseModule 为工具抽象类，其扩展为 Scanner 扫描器与 Filter 过滤器，JavaSpotbugsScanner 是扫描器的一种，而 SameVulFilter 和 RFFilter 是目前集成的误报过滤器，工具均通过 OsUtil 执行相关命令。通过工具抽象，系统可以迭代集成新扫描器或过滤器，统一接口提高了系统可扩展性。扫描流程仍需利用 JobService 实现下载待扫描项目包和更新任务状态功能，其通过 FilePathUtil 管理工作环境相关路径，通过 JobStatusMap 实现任务状态缓存。此外，CPPDefaultFlow 与 CPPTscodeScanner 均与 C++ 漏洞扫描任务流程相关，VulScanThread 将根据扫描任务类型执行相关任务流程。

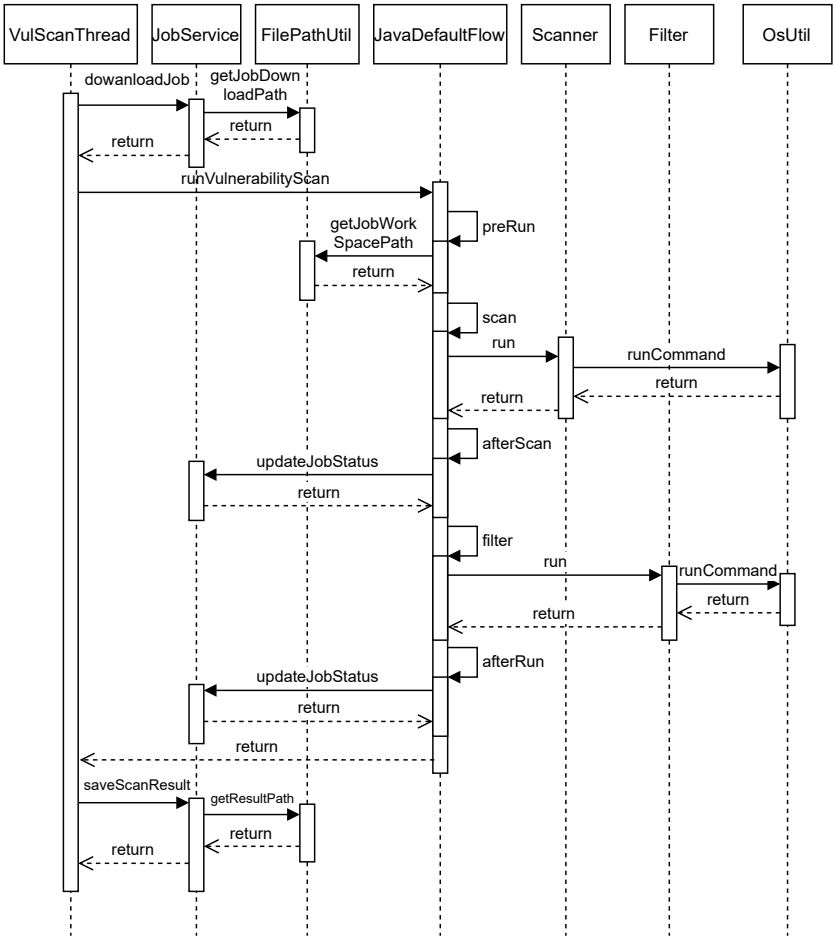

图 4.10: 漏洞扫描模块时序图

漏洞扫描模块时序图如图 4.10所示，当异步任务消费者监听到扫描任务时，

会启动 VulScanThread 线程执行扫描流程。VulScanThread 首先调用 JobService 并借助工作环境辅助类下载待扫描项目包；然后根据扫描类型调用不同扫描流程，若为 Java 字节码，则调用 JavaDefaultFlow 流程中的 runVulnerabilityScan 方法，使用默认流程配置执行 Java 字节码漏洞扫描流程。JavaDefaultFlow 首先执行 preRun 方法通过 FilePathUtil 获取工具环境；其次，调用配置扫描器的 run 方法开始漏洞扫描；漏洞扫描完成后需通过 JobService 更新缓存中任务状态；然后，执行 filter 方法调用配置过滤器的 run 方法开始进行误报漏洞过滤；最后执行 afterRun 方法借助 JobService 再次更新缓存中任务状态。其中扫描器与过滤器均借助 OSUtil 中封装好的命令行接口执行。扫描流程结束后，VulScanThread 通过 JobService 将任务结果存储至数据库。

```
1  public void run() {
2    …… // 省略获取 url、jobId、timeout 等代码
3    filePath = jobService.downloadJob(downloadUrl, jobId);
4    ……
5    String taskType = jobInfo.getTaskType();
6    switch (taskType) {
7      case Consts.SCAN_JAVA:
8        flow = new JavaDefaultFlow(filePath, jobId, timeout);
9        ……}
10   TimeLimiter timeLimiter = new SimpleTimeLimiter();
11   Callable<String> taskFlow = () -> {
12     flow.runVulnerabilityScan();
13     ……};
14   ……
15   timeLimiter.callWithTimeout(taskFlow, timeout,MINUTES, true);
16   ……
17 }
```

图 4.11: 漏洞扫描流程启动关键代码

图 4.11展示了 VulScanThread 中 run 方法的部分关键代码，首先通过 JobService 下载待扫描项目包，然后根据扫描类型选择不同扫描流程，最后执行扫描流程并设置超时时间。

4.2.1　漏洞扫描工具集成

为促进工具模块化与标准化，提高系统扩展性，本系统为扫描工具制定统一接入标准。所有集成到本系统的漏洞扫描工具均需改造成命令行调用方式，提供三个基本操作原语 start、status 和 stop。除通过命令行直接调用执行外，工具还需设计 Dockerfile 完成 Docker 部署，为后续工具统一 Dokcer 化预留扩展空间。

Java 字节码漏洞扫描工具采用开源项目 SpotBugs 及其插件 Find-sec-bugs，本节将对该扫描工具的改造与集成实现进行相关说明。

改造工具第一步为审核并筛选缺陷模式、完善扩充漏洞描述。原始扫描工具共设计 10 大类缺陷模式，共计 593 项漏洞。首先，依据 CWE 和 CVE 标准描述以及工具官方文档说明，只保留安全类型的漏洞，并手动删除其中官方用于测试但未删除的样例、历史遗留的一些噪声模式、处于实验阶段还未成熟的模式以及根据 CWE 标准说明和人工审核判定为不是漏洞的模式。然后，将剩余 146 项安全类漏洞归类为如图 3.11中的 20 类。最后，对漏洞描述进行整理扩充。原始工具所有漏洞描述均存储在内部 XML 格式化文件中，官方漏洞描述包括漏洞名称、类型、风险等级、位置信息、概述以及详细描述，其中位置信息与概述为漏洞实例相关信息，而其它则为漏洞模式相关信息。详细描述中通过带标签 html 语言融合漏洞描述、代码示例以及解决方法于一体，为了更好的存储、管理与扩充，本系统将详细描述部分抽出，并结合 CWE 标准描述和实践经验，扩充漏洞描述的漏洞代码示例、解决方案推荐以及参考链接。

```
1  public static JobStatus doScan() {
2    filePath = EnvConfig.SCAN_FILE_PATH;
3    ……
4    collection = scanWithSpotbugs();
5    ……
6    List<Vulnerability> vulnerabilities =
7           BugCollectionAdapter.BugCollection2VulList(collection);
8    ……
9    return new JobStatus(Response.SUCCESS.getCode(),
10          Response.SUCCESS.getMsg(), vulnerabilities);
11   }
```

图 4.12: 扫描工具主流程关键代码

第二步将原始工具封装成符合系统标准的命令行工具。工具使用 Apache 的 commons-cli 包解析命令行参数格式，以实现三个基本原语操作。其中，扫描主流程关键代码如图 4.12所示。首先获取解析参数时设置的待扫描项目包路径，然后调用私有方法 scanWithSpotbugs 方法使用 SpotBugs 工具进行漏洞扫描，最后使用数据适配器将原始工具漏洞列表转换成系统规范漏洞列表。原始工具官方提供命令行调用方式，支持生成 xml、html 以及控制台输出格式报告，这些报告内容各有侧重点。为保证获取漏洞所有相关信息并自定义选择有用信息，改造工具通过添加依赖直接调用 SpotBugs 内部扫描方法，并获取包含所有信息的 BugCollection 结果对象。BugCollectionAdapter 用于将原始信息转换成本系统规

范信息，并扩充漏洞实例相关描述，最终生成改造版结果返回给系统以渲染生成静态报告。为方便跟进官方更新，工具改造采用非侵入式方式，即不修改原工具，而是使用适配器将原漏洞列表根据系统漏洞描述文件转换成系统漏洞列表。

为后续 Docker 化预留扩展空间，最后，为封装工具设计 Dockerfile，其关键代码如图 4.13所示。封装工具需运行在 Java 环境中，选用预先集成了 Java 的 anapsix/alpine-java[1]作为基础镜像，该镜像建立在轻量级 Alpine 系统上，可以减少服务器资源消耗。

```
1  #Base image
2  FROM anapsix/alpine-java:latest
3  ……
4  #Add scanner master
5  COPY . /scanner
6  WORKDIR /workspace
7  ……
8  ENV PATH /scanner/scanner/bin:$PATH
9  RUN chmod 77 /scanner/scanner/bin/jscan
10 ……
11 ENTRYPOINT ["sh", "/scanner/scanner/bin/jscan"]
```

图 4.13: Dockerfile 关键代码

4.2.2　上下文提取实现

扫描工具漏洞结果中位置信息主要包括类名、方法名、方法签名、字段名以及产生该漏洞对应源代码行数，本部分将基于 Joana 程序切片工具，根据漏洞产生行数以及位于方法对漏洞进行上下文切片。

SpotBugs 中每个漏洞都包含一系列针对漏洞位置的注解信息，包括类注解、方法注解、行注解、字段注解等，而 Joana 切片工具需要根据类信息、方法信息以及源码行数信息来对相关代码进行切片，因此切片之前，需要解析这些注解并组装成 Joana 需要的信息。系统首先根据每个漏洞包含的注解信息，提取出类注解、方法注解和行注解，在分析这些注解得到类全限定名、方法名、方法签名以及漏洞产生代码起止行数，最后将这些信息传入 Joana 切片器进行上下文提取。

图 4.14展示了使用 Joana 进行程序后向切片的关键代码片段，首先根据待切片方法信息构建 SDG（3~8 行），此步骤较为耗时，因此将构建好的 SDG 进行缓存，一定程度上提高切片速度。然后根据待切片方法与代码行数信息提取 SDG 中与漏洞行数相关的节点集合（11~14 行），此处主要根据起止行数信息寻找位

[1]https://hub.docker.com/r/anapsix/alpine-java/

于起止行之间的所有节点。之后使用 Joana 对这些节点进行后向切片（16 行），主要是从终止行相关节点逆向切片直至方法入口。最后将切片内容进行格式化，统一成系统需要格式（17 行），每行切片主要包括语句索引、语句类型、语句操作、相关值类型以及具体语句内容信息，并将格式化后的最终切片作为漏洞上下文信息返回。

```
1  public String computeSlice(Func func, Location line)
2            throws SlicerException {
3    SDG sdg = null;
4    if (sdgCache.containsKey(func)) {
5      sdg = sdgCache.get(func);
6    } else {
7      ......
8        sdg = this.buildSDG(func);
9      ......
10   }
11   JoanaSDGSlicer jSlicer = new JoanaSDGSlicer(sdg);
12   HashSet<SDGNode> sliceNodes;
13   ......
14     sliceNodes = jSlicer.getNodesAtLocation(line, func);
15   ......
16   Collection<SDGNode> slice = jSlicer.slice(sliceNodes);
17   String result = Formatter.prepareSliceForEncoding(slice);
18   return result;
19 }
```

图 4.14: Joana 切片关键代码

4.2.3　误报过滤实现

误报过滤器需封装成命令行工具，并提供启动方法。本系统目前设计实现两种误报过滤器，第一种计算漏洞描述及上下文与历史数据相似度，从而得到误报评分；第二种是基于 N-gram 与随机森林分类器的误报过滤器。

如模块时序图 4.10所示，扫描工具完成扫描后，将进入主流程的 filter 方法环节，此环节依次使用过滤器列表中的误报过滤器对扫描结果进行漏洞误报过滤。此部分关键代码如图 4.15所示，若扫描不成功则不进行误报过滤，否则依次进行多重误报过滤。此处使用 List 列表容纳多个过滤器，方便过滤器的灵活配置与统一调用。

```
private List<AbstractBaseModule> filters;
……
void filter() {
  if (!scanSuccess) {
    this.filterSuccess = false;
    return;
  }
  this.filters.forEach(filter -> {
    filterSuccess |=  filter.run(this.filePath, this.jobId);
  });
  ……
}
```

图 4.15: 误报过滤关键代码

4.2.4　转众包审核实现

众审服务是平台提供的外部众包审核服务，该服务将对用户创建的任务进行众包审核并生成审核报告。漏洞扫描任务完成后，用户可将本系统漏洞结果发布众审。转众审任务部分关键代码如图 4.16所示。本系统接收到发布众审请求，首先查询得到对应任务的漏洞列表，然后进行数据转换和封装，将漏洞信息转换成众审服务规定格式数据，最后将众审漏洞信息发送至众审服务，前端页面也立即跳转至创建众审任务页面。

```
public CrowdReviewReportVo createCrowdReview(String jobId,
       String fileUrl) {
  ……
  List<Vulnerability> vulList = jobService.getResultList(jobId);
  List<CrowdReviewItem> items = JobResultListAdapter
       .getCrowdReviewData(vulList, fileUrl);
  CrowdReviewDto crowdReviewDto = new CrowdReviewDto();
  ……
  crowdReviewDto.setCrowdReviewItems(items);
  RestTemplate rt = new RestTemplate();
  HttpEntity<CrowdReviewDto> httpEntity = new HttpEntity<>(
         crowdReviewDto, getHeader());
  String url = getCreateCrowdReviewUrl();
  CrowdReviewReportVo crowdReviewReportVo = rt.postForObject(
         url, httpEntity, CrowdReviewReportVo.class);
  ……
}
```

图 4.16: 转众包审核关键代码

众审服务中创建众审任务页面需要用户填写审核任务基本信息，其中漏洞列表信息展示情况如图 4.17所示，使用视图模式展示每个漏洞相关描述；定制审核选项部分如图 4.18所示，用户可以按照自身需求自定义配置审查选项。

图 4.17: 众审漏洞列表展示页面

图 4.18: 众审定制审查项页面

4.3　迭代学习模块

迭代学习模块通过众审服务不断扩增真实漏洞数据及其是否误报标签，此部分带标签漏洞库可以再作为误报过滤模型的训练数据集。在大多数应用中，使用机器学习等技术的最大挑战是如何在真实工程项目中获得足够高的准确率，而改进训练数据集通常是最快的能够提升准确率的途径。因此，本系统使用来自真实项目的漏洞库以及众审专家审核的标签作为迭代学习模型的训练集，可以有效改进模型效果。迭代学习模块整体流程图以及迭代学习示意图分别如图 3.12和图 3.13所示，第三章中已经说明，此处不再赘述。本节主要详细介绍误报过滤模型训练过程的实现。

误报过滤模型训练模块架构图如图 4.19所示，整体分成数据预处理、N-gram提取特征以及随机森林分类模型训练三个子模块，同时有相关数据与验证方式用于支撑工作。其中，数据预处理子模块对漏洞扫描和众审结合生成的带标签漏洞库进行相关解析。首先，提取漏洞实例信息中的上下文片段（若有切片则加上切片信息）作为训练数据，提取漏洞实例审核信息作为是否误报标签；再对上

下文片段进行分词并去除停用词；最后对部分字符进行抽象化，以剔除具体值对结果的干扰。N-gram 特征提取子模块使用 N-gram 语言模型提取上下文片段特征，即依次通过词频计算、略去低频词、构建字典，最后得到每个上下文片段的特征向量，同时生成语料字典库以用于预测。随机森林分类训练子模块首先进行训练集与测试集划分，再构建训练模型，最后进行分类训练，最终生成误报过滤器。本系统使用交叉验证进行结果观察，并且通过多次实验学习出最佳参数配置。

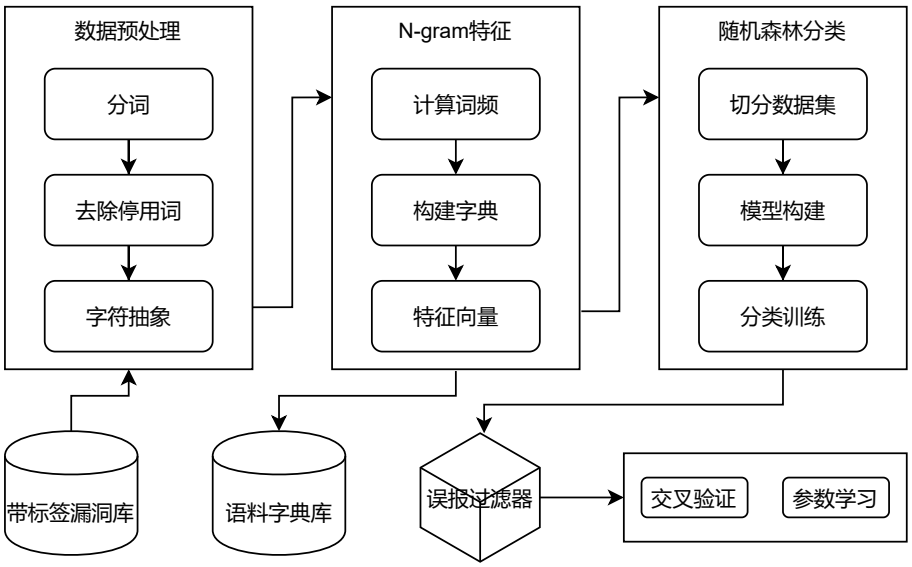

图 4.19: 模型训练架构图

4.3.1　数据集与数据预处理

随着众审任务不断推进，本系统将持续更新扩充漏洞数据集，但仍需要部分数据集来解决模型冷启动问题。OWASP Benchmark[2]是致力于评估静态检测工具效果的项目，它提供了一些 Java 语言编写的测试代码，其中每个文件对应一个漏洞项，并且给出是否误报标签，本文选用该数据集作为初始数据集进行启动模型创建。

模型训练子模块的输入数据为带标签的漏洞库，主要包括漏洞名称、漏洞上下文信息以及是否误报标签，分类器主要依赖特征为漏洞上下文信息，因此需先对漏洞上下文信息进行预处理操作。首先，如图 4.20所示，漏洞上下文片段为一串字符文本，需对其进行单词划分，得到一组有意义的单词。主要分词策略包括但不限于以下几个方面：

[2]https://owasp.org/www-project-benchmark/

```
1 :: ENTR :: entry :: null ::eumes.pathtraver_fp(java.lang.String)
7 :: EXPR :: assign :: Ljava/lang/Object :: v7 = #(126) + #(106)
8 :: PRED :: IF :: Z :: if (v7 <= #(200)) goto 11
9 :: EXPR :: assign :: null :: PHI v10 = #(This_always_happen), v4
13 :: NORM :: declaration :: Ljava/io/File :: v11 = new java.io.File
14 :: NORM :: declaration :: Ljava/io/File :: v12 = new java.io.File
15 :: EXPR :: assign :: Ljava/lang/String :: v13 = testfileDir
```

图 4.20: 上下文示例代码

1. 将切片按行分开，并按“::”分割，以提取每行切片中包含的各部分内容，并去除行索引和语句类型（即前两列）。

2. 将 L 引导的类签名具体值与 L 分离开。如“Ljava/lang/String”改成“L 空格 java/lang/String”。

3. 将驼峰式名称进行分离。如“PDFConfigReader”改成“PDF 空 Config 空 Reader”，“getErrorMAP”改成“get 空 Error 空格 MAP”等。

然后去除停用词阶段，首先针对类 SSA 代码具体特征，去除“/”、“;”、“<”、“_”等无意义特殊字符；然后，去除除空格外的其他类型空白符。

最后，字符抽象阶段主要将一些特定值字符换成统一规范字符，以排除变量具体值对特征提取的干扰。其具体表现为：变量名的抽象，将变量名替换成 VAR VN 形式；字符值的抽象，将字符值替换成 STRING SN 形式；数字的抽象，如将数字统一替换成 N3P 等。经过预处理后，如图 4.20中的上下文片段将转换成如图 4.21所示。此处为展示效果，保留了换行符。

```
entry null eumes pathtraver fp ( java lang String )
assign java lang Object VAR V0 = N3P + N3P
IF Z if ( VAR V0 <= N3P ) goto 11
assign null PHI VAR V1 = STRING S0  VAR V2
declaration java io File VAR V3 = new java io File
declaration java io File VAR V4 = new java io File
assign java lang String VAR V5 = testfile Dir
```

图 4.21: 预处理后上下文示例图

预处理子模块关键代码如图所示，主要通过正则表达式匹配方法进行上述相关处理操作。其中 text 为漏洞库中上下文片段，value 为初步分词后 text 中每个词，sample 为预处理操作后新生成的上下文片段信息。

```
def preprocessing(text: str) -> str:
  ……
  all_matches = re.findall(STRING_PATTERN, text)
  counter = 0
  for s in set(all_matches):
    text = text.replace(s, 'STRING S' + str(counter) + ' ')
    counter += 1
  ……
  value = re.sub(r'L([a-zA-Z0-9]+(/[a-zA-Z0-9])+)', '\g<1>', value)
  value = re.sub(r'([A-Z]+[a-z0-9]*)', ' \g<1> ', value)
  value = re.sub(r'([A-Z]+)([A-Z][a-z0-9]+)', '\g<1> \g<2>', value)
  value = re.sub(r'( |^)\d\d\d+($| )', ' N3P ', value)
  ……
  return sample
```

图 4.22: 预处理关键代码

4.3.2　分类模型训练的实现

N-gram 是一种基于统计语言模型的算法，其基本思想是通过滑动窗口得到文本的字节片段集合，通过对每个字节片段的词频统计最终形成关键片段列表，即该文本的特征向量。随机森林分类器是一种 Bagging 算法，其核心思想是并行构建 n 个决策树，通过一定策略组合成一个强分类器，因此随机森林分类效果一般比决策树好。本项目使用 Tri-gram（三元模型）进行漏洞上下文片段特征的提取，同时使用随机森林进行分类模型训练。

```
def train(slice_dir: str, label_dir: str, train_percent: float = 1,
        saveto: str = None, data_length: int = 0)
  ……
  dataset = TextDataset(slice_dir, label_dir,
        preprocessing.preprocessing, data_length)
  train_data, test_data = dataset.divide(train_percent)
  ……
  ngram_tokenizer = NGramTokenizer(freq_gt=WORD_FREQ)
  ngram_tokenizer.build_dict(dataset)
  ……
  train_x, train_y = ngram_tokenizer.tokenize_labeled_batch(train_data)
  ……
  rf_model = RandomForestClassifier(n_estimators=440, n_jobs=-1)
  rf_model.fit(train_x, train_y)
  ……
```

图 4.23: 模型训练关键代码

分类模型训练子模块关键代码如图 4.23所示。首先，通过 TextDataset 构建

数据集，指定数据预处理方法对数据进行上节所述预处理操作，并按比例划分训练集和测试集。然后，构建 N-gram 字典并提取上下文特征，NGramTokenizer 使用 sklearn 中的 CountVectorizer 计算数据的 N-gram 特征，并生成 N-gram 数据字典。最后，创建随机森林分类器，并将数据集输入启动分类器训练过程。训练结束后，对分类器模型以及 N-gram 字典进行持久化。

4.3.3　模型相关参数评估

本节主要将上述实现模型与决策树分类算法就 OWASP Benchmark1.1 数据集进行效果对比，并通过实验选取随机森林相关参数的较优值，以训练得到本系统误报过滤启动模型。

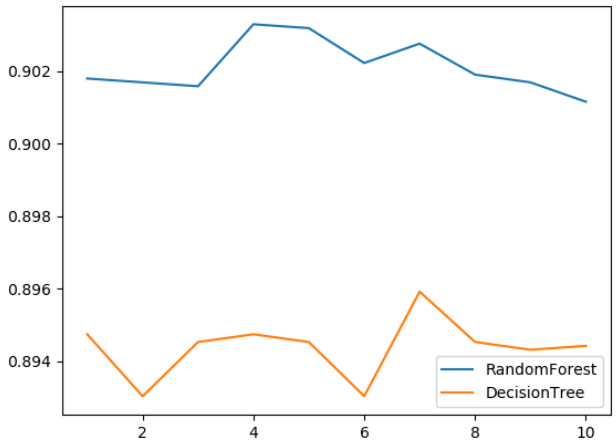

图 4.24: 随机森林与决策树效果对比图

图 4.24展示了随机森林与决策树分类的效果对比图，其中横轴表示十折交叉验证轮次，纵轴表示模型综合评分，可以看出随机森林始终优于决策树。

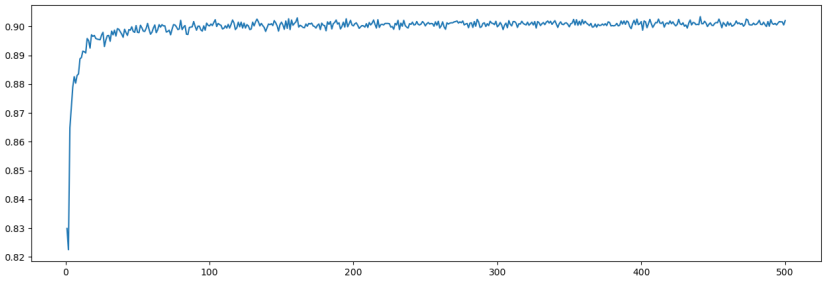

图 4.25: n_estimator 学习曲线

影响随机森林分类效果的主要参数是 n_estimator，其含义为随机森林中决策树的数量，多个决策树分类的融合效果优于单个决策树，但并非树越多效果越好，

因此通过验证不同决策树数量的训练效果来确定最优 n_estimator 值。图 4.25展示了 n_estimator 取值范围从 0 到 500 时模型的评分变化曲线。可以看出，整体趋势为取值越大，模型效果越好，其峰值出现与 n_estimator 为 440 时，故初始启动模型按此参数进行训练。

4.4　页面展示

本小节将通过部分系统运行页面截图以更直观展示本系统内容。

图 4.26: 创建任务页面

图 4.26是用户登录并进入发布静态扫描任务后的页面。用户在此页面创建扫描任务，可以选择重新上传带扫描项目包，也可快速使用上次使用过的项目包，并进行扫描服务类型的选择，最后填写项目相关描述即可提交测试，等待管理员审核。

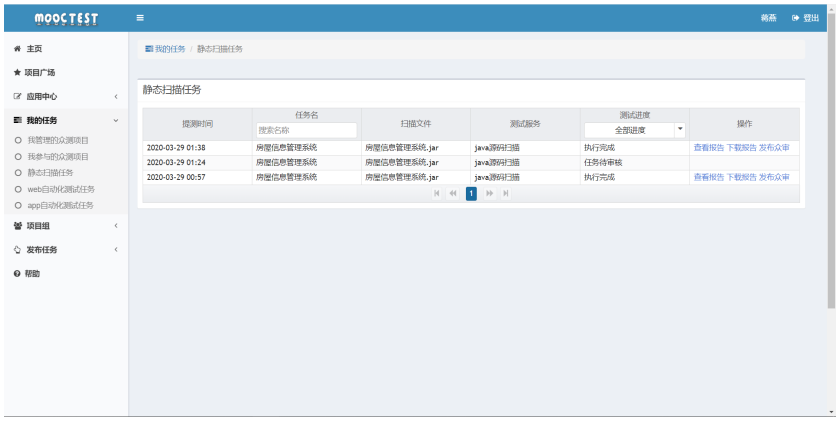

图 4.27: 任务列表页面

图 4.27展示了用户发布过的所有任务列表。在此页面，用户可以查看之前发布的任务是否审核通过，以及已经执行任务的当前进度。对于审核通过的任务，

用户可以进行执行任务操作；对于已经完成扫描的任务，用户可以查看或下载报告以及发布众审；对于审核未通过的任务，用户可以查看详情以及未通过原因。

图 4.28: 报告漏洞统计页面

图 4.28展示了静态报告中漏洞统计页面。该页面主要展示扫描项目基本信息如提测人、提测时间、扫描耗时等；并且以饼图方式展示漏洞在风险等级以及类型上的占比情况。通过点击饼图对应区域可直接跳转至该区域对应漏洞列表。

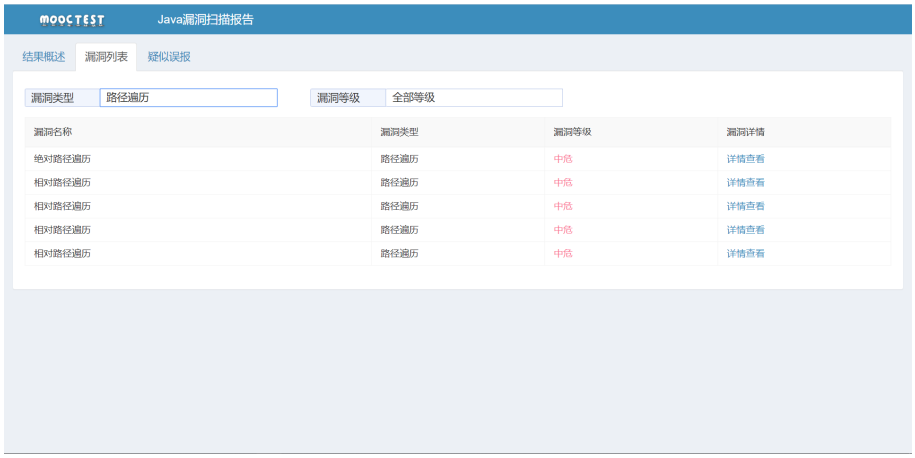

图 4.29: 报告漏洞列表页面

图 4.29展示了静态报告中漏洞列表页面。该页面展示所有扫描出的漏洞列表，支持通过漏洞类型以及漏洞等级进行筛选显示；同时使用不同颜色标记漏洞风险等级。误报列表页面与漏洞列表页面相似，误报列表是本系统标记为误报的漏洞列表，同时返回给用户以供用户参考。

图 4.30展示了漏洞详情页面。该页面展示漏洞实例的详细描述信息，包括

名称、类型、风险等级、概述、描述、位置、上下文信息、示例代码、解决方案以及相关链接等信息。各模块可以选择性折叠扩展，方便用户查阅感兴趣的部分。

图 4.30: 报告漏洞详情页面

4.5　本章小结

本章主要对人机协同 Java 字节码漏洞扫描系统关键模块进行详细设计并详述了相关实现细节。首先，利用相关框架图、核心类图、时序图以及关键代码详述了交互展示模块的实现。其次，对漏扫核心模块进行了类图的详细设计以及相应实现代码进行了描述。然后，对迭代学习模块中模型训练相关子模块的主要实现进行了详述，并对模型进行了相应评估。最后，展示并简述了本系统主要实现页面。

第五章　系统测试与实验分析

本章将依据第三章的需求分析与第四章实现描述，对系统各模块功能、系统响应性能以及涉及模型效果进行测试与评估。本章首先描述测试环境，然后对系统进行功能与性能测试并展示测试效果，最后对误报过滤模型进行评估。

5.1　测试环境

表 5.1展示了人机协同 Java 字节码漏洞扫描系统的测试环境，其主要服务器均使用阿里云 ECS 服务器，并根据服务器具体用途采用不同配置，后端服务器考虑 CPU 与内存大小，采用 2 核 4G 内存，前端服务器使用基础版本 Ubuntu 18.04，数据库服务器选用大容量硬盘。

其中，客户端使用用户计算机的 Chrome 浏览器作为与系统交互平台。本系统涉及服务无需高强度计算性能或瞬时交易响应，故选择 ECS 服务 4G 内存配置。主服务器上主要涉及运行程序包括主程序运行 JDK、主程序运行容器 Tomcat、工具运行容器 Docker 以及缓存服务 Redis。前端程序使用 ECS 服务器，主要运行主前端服务 Angular 2 以及静态模板引擎 Pug v2。前端服务器与后端服务器之间使用 RESTful 风格的 API 进行通信。数据库服务器主要包括阿里云 OSS 对象存储服务以及本系统主数据库 MongoDB。

表 5.1: 系统测试环境

设备名称	运行程序	相关描述
用户计算机 (Win10 专业版)	Chrome77.0(64 位)	客户端，用户与平台交互
用户计算机 (Ubuntu 18.04)	JDK 1.8.0_151	扫描器健壮性测试环境
ECS 服务器 (2 核 4G 内存)	Spring Boot 2.2.5	主服务使用框架
	JDK 1.8.0_151	主程序 JDK 版本
	Tomcat 8.5.53	主程序运行容器
	Redis 4.0.7	缓存服务
	Docker 18.03	工具运行容器
ECS 服务器 (1 核 2G 内存)	Angular 2.4	主前端使用框架
	Pug v2	静态报告模板引擎
ECS 服务器	OSS 对象存储	阿里云对象存储服务
	MongoDB 3.6.8	主程序使用数据库

5.2　功能与性能测试

5.2.1　功能测试

功能测试以用户使用角度对系统各功能进行验证，本小节将根据表 3.1系统功能性需求列表，对系统功能测试用例逐项测试，以检测系统是否达到预期。

表 5.2展示了审核管理的相关测试过程与预期结果，其主要针对如表 3.3描述的用例 UC1，进行平台管理员对用户提交扫描任务的审核工作，包括审核通过与审核拒绝。此项测试用例覆盖了审核管理各项功能。

表 5.2: 审核管理测试用例

测试 ID	TC1
测试名称	审核管理
测试功能	平台管理员可以查看所有扫描任务，待审核任务置顶，显示任务进度；管理员可以查看任务详细信息，并选择“通过”或“拒绝”该扫描任务
测试步骤	1. 管理员进入任务审核页面，查看所有扫描任务是否按预期显示； 2. 管理员点击“详情”查看待审核任务详情； 3. 管理员点击“同意”，审核通过任务； 4. 管理员点击“返回”，查看另一待审核任务详情，并点击“拒绝”； 5. 管理员点击“返回”，查看所有任务状态；
预期结果	1. 系统显示所有扫描任务，并且把待审核任务按时间升序置顶显示； 2. 页面显示该任务详细信息，并提供“通过”、“拒绝”和返回三个按钮； 3. 页面显示任务审核通过，并提供“返回”按钮； 4. 页面显示任务审核不通过，并提供“返回”按钮； 5. 页面显示所有任务列表，之前审核通过和不通过的任务状态分别为审核通过和审核不通过。

表 5.3: 任务管理测试用例

测试 ID	TC3
测试名称	任务管理
测试功能	普通用户进行任务创建以及任务发布
测试步骤	1. 普通用户进入发布扫描任务页面； 2. 普通用户点击“选择文件”，并选择需扫描项目包进行文件上传； 3. 用户填写项目名称和项目描述，并选择服务项目为“Java 漏洞扫描”； 4. 用户点击“提交测试”按钮。
预期结果	1. 系统显示上传目标指引； 2. 页面显示上传进度，上传成功后，页面提示“上传成功”，并且目标文件显示为刚上传文件名称； 4. 系统提示“提交成功，等待审核”。

表 5.3展示了任务管理相关测试过程与预期结果，其主要针对表 3.5中用例描述进行用户创建以及提交扫描任务功能的测试，主要关注文件上传是否成功、提交任务是否成功以及页面是否正确反馈信息。

表 5.4: 执行扫描任务测试用例

测试 ID	TC4
测试名称	执行扫描任务
测试功能	系统按正常流程进行扫描任务执行，并实时更新任务进展状态
测试步骤	1. 普通用户进入扫描任务列表页面； 2. 用户对测试进度筛选“任务审核通过”状态任务； 3. 用户选择第一个任务点击“执行”，开始审核通过扫描任务的执行； 4. 在该页面停留等待，并观察该任务状态变化。
预期结果	1. 系统显示该用户所有扫描任务，并可根据名称或状态进行筛选； 2. 页面显示所有“任务审核通过”状态任务列表； 3. 该任务状态变成“正在排队”； 4. 在一定时间内，该任务状态依次变成“正在执行”、“执行完成”。

表 5.4展示了任务执行相关测试过程与预期结果，其针对表 3.6中用例描述进行用户执行任务的功能测试，主要关注任务执行过程中状态变化是否正确。

表 5.5: 扫描报告测试用例

测试 ID	TC6
测试名称	扫描报告
测试功能	普通用户可以在线查阅扫描报告，并下载报告进行离线分析
测试步骤	1. 普通用户进入扫描任务列表页面并筛选“执行完成”状态任务； 2. 用户点击“查看报告”； 3. 鼠标可悬停风险等级统计饼图高危区块，并可以点击饼图相应区块； 4. 进行类型筛选为“安全”，等级筛选为“中危”； 5. 点击第一行漏洞后“详情”。
预期结果	1. 系统按时间显示所有执行完成的扫描任务； 2. 系统跳转至该任务静态报告主页，该页面显示任务基本信息以及漏洞统计信息饼图； 3. 鼠标悬停显示高危漏洞占比以及数量，点击后可跳转至高危漏洞列表页； 4. 页面显示所有安全类型中危等级的漏洞列表； 5. 页面跳转至该漏洞详情页，展示漏洞名称、漏洞等级、漏洞类型、漏洞描述、漏洞示例代码、漏洞解决方案以及参考链接等信息。

表 5.5展示了扫描报告相关测试过程和预期结果，其针对表 3.8中用例描述进行用户查阅、下载静态报告功能测试，主要关注静态报告是否展示正确信息。

表 5.6: 测试用例执行结果

测试 ID	用例 ID	执行结果
TC1	UC1	通过
TC3	UC3	通过
TC4	UC4	通过
TC6	UC6	通过

最终，测试人员严格按照上述测试用例描述相关步骤执行各测试用例，其测试结果如表 5.6所示，所有测试用例均符合预期结果。证明本系统满足用户基本功能需求，符合系统功能设计。

5.2.2　性能测试

本小节对本系统的相关性能进行测试，主要利用 JMeter 工具对本系统的两个常用接口进行压力测试 [48]，即启动任务接口和查询状态接口。

表 5.7展示了用户并发请求数设为 200 时的接口压力测试汇总表。其中，对于任务启动接口，2 秒内共进行 200 次并发访问请求，平均响应时间为 201ms，响应时间最大值为 205 秒，其中 99% 的用户请求都能在 203ms 内得到系统响应。

表 5.7: 接口压力测试汇总表

接口	并发量	平均值	中位数	99% 百分位	最小值	最大值	异常率	吞吐量
任务启动	200	201	201	203	200	205	0.00%	167.5/sec
结果查询	200	204	203	235	201	247	0.00%	166.7/sec

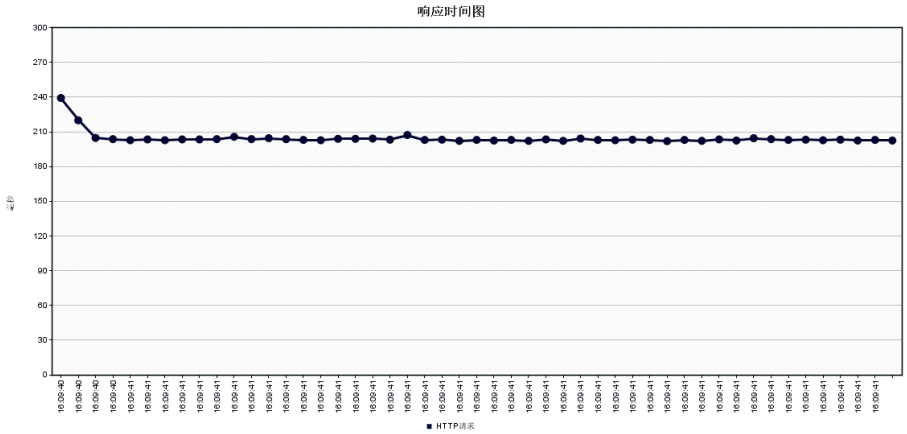

图 5.1: 结果查询接口响应时间图

对于结果查询接口，2 秒内共进行 200 次并发访问请求，平均响应时间为 204ms，响应时间最大值为 247ms，其中 99% 的用户请求能在 235ms 内得到响应。两个系统常用接口在测试过程中均没有错误请求的发生，其吞吐量平均每秒处理 167 条请求，系统正常通过压力测试。

其中，结果查询接口的响应时间如图 5.1所示，图中以 20ms 为间隔记录了 200 次并发请求查询接口的响应时间，可以看出测试结果响应时间较为平稳，均在 210ms 左右。综上，本系统可以在实际场景中满足用户需求，在较高并发情况下也能平稳运行。

5.2.3　健壮性测试

健壮性是指一个计算机系统在执行过程中处理错误，以及程序遭遇不正确输入或处理异常时继续正常运行的能力。本小节主要对改造的扫描器进行健壮性测试，以保证其可用性。扫描器通过处理多种可能出现异常，记录相关错误信息日志，并返回准确描述错误的结果，以此保证扫描器遭遇异常时不会崩溃退出，从而保证工具的可用性。

表 5.8: maven 仓库测试集结构表

总数	有效数	文件损坏数	无字节码数	正常项目数
100	96	11	11	74

本次测试仅对改造未集成的扫描器进行，并以此作为能否集成系统的标准之一。本次测试环境使用 Ubuntu 18.04 系统，8GB 内存，开启 4 线程同时扫描测试集。测试数据集预选取 maven 仓库最流行的前 100 个 jar 包[1]并选取使用量最大的版本，实际数据结构如表5.8所示。总共获取 96 个 jar 包，略去 4 个安卓项目 aar 包，其中 11 项为文件损坏[2]，无法正常解压，另 11 项中无字节码文件，剩余正常 jar 包共 74 项。

表 5.9: maven 仓库测试结果表

结果	数量	描述
扫描成功	73	扫描完成，且有结果输出
扫描未进行	11	文件损坏导致文件无法解压
扫描未完成	11	jar 包中无字节码文件
扫描报错	1	工具抛出内存溢出错误

[1]https://mvnrepository.com/popular，获取时间为 2019 年 10 月 24 日，具体数据包随时间可能发生变化

[2]实验保留此 11 项，以验证扫描器能正常处理文件损坏异常

本次测试结果如表5.9所示。其中，73 项成功扫描并输出漏洞结果。11 项为因网络原因未下载完整 jar 包，导致文件损坏无法正常解压，此 11 项在进行文件格式检查时被拦截，不会进行扫描过程；对于该 11 项数据，工具返回正确错误码以及错误信息。另 11 项扫描流程未完成，原因是 jar 包中没有字节码文件，工具返回正确错误信息。1 项由于项目内字节码数量过多，工具错误日志中记录内存溢出错误，但工具成功输出部分漏洞结果。此错误可以通过调整 JVM 内存大小限制有效减少出现次数。

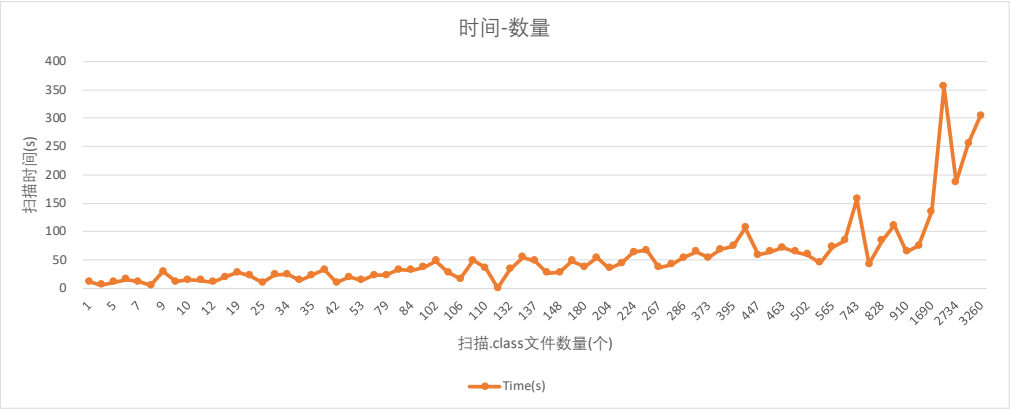

图 5.2: Maven 数据集字节码数量与扫描时间关系图

图5.2展示了本次测试各项目中字节码文件数量与扫描时间对应图，可以看出，本扫描器能在一个可接受等待时间内进行一定规模项目的扫描。

综上，本扫描器可以成功扫描正常项目包，对于扫描过程中出现的异常，能够实时记录错误信息，并返回准确错误信息。同时，对于内存溢出错误，本扫描器记录错误信息，并在崩溃退出之前返回部分漏洞结果。本扫描器可用性较高。

5.3　实验分析

本文针对现有传统静态扫描工具误报率高的问题，提出基于随机森林分类模型的误报过滤器以及众审专家辅助机制，降低漏洞扫描结果误报率，提供低误报漏洞扫描服务。本次实验将本系统误报率与原版 SpotBugs 工具误报率进行对比，以验证本系统可以有效降低静态扫描工具误报率。本次实验选取 OWASP Benchmark1.1 项目作为评估数据集，本小节将详述本次实验数据集、实验流程、评估指标并通过结果分析具体描述系统误报过滤效果。

5.3.1　数据集

OWASP Benchmark 项目[3]致力于评估静态漏洞扫描工具检测漏洞的能力，其提供覆盖 11 种安全性漏洞的测试数据集，每个数据都给出正误报标签。目前，OWASP 共发布两个主版本数据集，其中 v1.1 共包含 21041 项，v1.2 共包含 2740 项。v1.2 本质上是 v1.1 的简化与修复版本，其主要目的是为一些动态扫描工具提供较少量评估数据集。本系统扫描耗时较短，因此选取 v1.1 数据集进行模型训练与测试，以更好保证模型准确率。OWASP v1.1 数据集规模如表 5.10所示，其包含 SQL 注入、路径遍历、弱加密、弱哈希、伪随机数等共 11 类常见 Web 安全漏洞，且每类漏洞均有正误报覆盖。

表 5.10: OWASP 项目数据规模表

数据集	测试用例总数	真实用例数	误报用例数
OWASP v1.1	21041	11835	9206

整个实验建立在此数据集上，将数据集按 9:1 比例随机划分成训练集和测试集，其中训练集数据按本文提出分类算法进行过滤模型训练，测试集用以评估本文实现系统相较于原版工具的误报过滤效果。

5.3.2　实验流程

本次实验需要对比传统工具扫描结果与本文实现系统扫描结果，来说明本系统的漏洞扫描能力以及误报过滤效果，因此本次实验将按以下步骤进行：

1. 将 OWASP v1.1 数据集按 9:1 比例随机划分为训练集和测试集，并将训练集作为模型训练输入数据得到对应过滤模型与预料字典。

2. 分别使用原版扫描器与本文实现系统对测试集进行漏洞扫描，针对得到的结果统计分析正报、误报漏洞数量信息。

3. 计算并比较二者的准确率、精准率、召回率等评估指标。

4. 以上实验进行三次，最终评估指标取三次实验平均值。

[3]https://owasp.org/www-project-benchmark/

5.3.3　评估指标

二分类器的性能可以通过一些定量指标来评价，本文将采用准确率、精准率、召回率、F1-score 以及 ROC 曲线指标来对系统误报过滤效果进行评估 [49]。

正确率（Accuracy），即预测正确的数据占测试数据总数的比例，其计算公式为[4]：$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$。精准率（Precision），即预测为正且实际也为正的数据占所有预测为正测试数据总数的比例，其计算公式为：$Precision = \frac{TP}{TP+FP}$。召回率（Recall），即预测为正且实际也为正的数据占所有实际为正的测试数据的比例，其计算公式为：$Recall = \frac{TP}{TP+FN}$。F1-score 是精准率与召回率的调和平均，综合考虑了二者而得出的指标，具有一定的可靠性，其计算公式为：$F1-score = \frac{2\times Precision\times Recall}{Precision+Recall}$。

ROC 曲线是一个综合衡量分类器性能的二维曲线，其纵轴表示 TP 率，横轴表示 FP 率。一般来说，若曲线整体为凸形，则表示分类性能较好，且凸点越靠近左上顶点，说明分类器性能越好。定量上，可以计算曲线与横轴之间的面积（AUC），且 AUC 越大，分类器性能越好。

5.3.4　结果分析

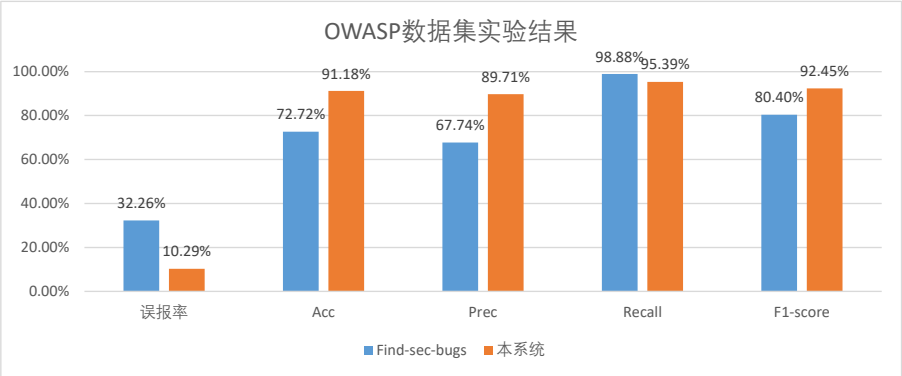

图 5.3: OWASP 实验 ROC 曲线图

分别使用原版扫描器与本文实现系统对测试集进行漏洞扫描，三次实验平均结果如图 5.3所示。可以看出，原版扫描器具有较高的召回率，即 98.88% 的真实漏洞都能成功扫描出，这在一定程度上保证了扫描器的低漏报。这一过度近似策略同时带来了大量误报（误报率为 32.26%）。与之相反，本文实现系统在保证 95.39% 召回率的情况下，将漏洞误报率减少近 22%，在准确率和 F1-score

[4]公式中涉及 TP 表示实际为正报且预测为正报，FP 表示实际为误报预测为正报，TN 表示实际为误报预测为误报，FN 表示实际为正报预测为误报

指标上也高于原版扫描器，牺牲的召回率具有较高的收益比。本次实验生成的ROC 曲线如图5.4所示，AUC 值为 0.94，表明误报分类器具有较好的性能。

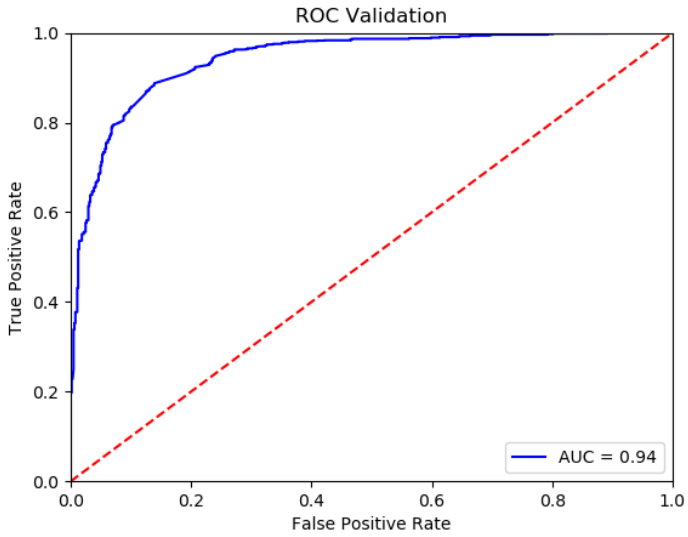

图 5.4: OWASP 实验 ROC 曲线图

OWASP 项目 v1.1 数据集共包含 11 类漏洞，分别为 cmdi（命令注入）、crypto（弱加密）、hash（弱哈希算法）、ldapi（ldap 注入）、pathraver（路径遍历）、securecookie（cookie 相关）、sqli（SQL 注入）、weakrand（伪随机）、xpathi（xpath 注入）、xss 和 trustbound（可信边界），其覆盖系统采用扫描器共 40 类具体漏洞。图5.5展示了误报过滤模型在各类漏洞上的性能，具体指标为 F1-score。可以看出，本系统实现误报过滤器在各类型漏洞上相较于原版 Find-sec-bugs 工具均有较明显的过滤效果，对于提升效果较小的注入类和 xss 漏洞，本系统将接入外部基于污点传播的误报过滤器来保证其过滤效果。

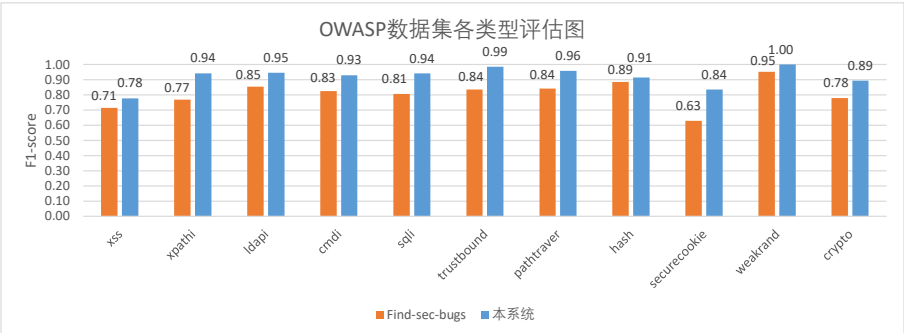

图 5.5: OWASP 数据集各类型效果图

综上，与原版静态 SpotBugs 和 Find-sec-bugs 扫描工具相比，本文实现系统

能在保证在少量牺牲召回率的情况下，有效减少扫描工具误报漏洞数，即本系统误报过滤可以有效降低扫描工具误报率。

5.4　案例分析

本节将通过两份相似代码，来展示分析本文实现系统处理正报和误报漏洞时的流程。图 5.6展示了两份可能存在路径遍历漏洞的代码，其中（a）代码片段为两份代码公共部分，两份代码不同之处在于 *doSomething*() 方法的具体实现存在差异。（b）为一种处理方式，即将用户输入直接与系统目录拼接，返回完整文件路径；（c）为第二种处理，通过条件判断以决定返回哪种路径，由于判断条件永远为 true，即返回路径为安全路径；因此（b）为路径遍历漏洞正报示例，（c）为误报示例。

```
1  ……
2  public String readFile(String param) throws IOException {
3    String fileName = doSomething(param);
4    File file = new File(fileName);
5    try {
6      String result = FileUtils.readFileToString(file);
7      return result;
8    } catch (IOException e) {
9      throw e;
10   }
11 }
```

（a）公共代码片段

```
1  ……
2  public String doSomething(String param)
3  {
4    String fileName;
5    String workDir = "dir"
6           + File.separator;
7    fileName = workDir + param;
8
9    return fileName;
10 }
```

（b）第一种处理代码

```
1  public String doSomething(String param) {
2    String fileName;
3    int i = 86;
4    if ((7 * 42) - i > 200) {
5      fileName = "savePath";
6    } else {
7      fileName = param;
8    }
9    return fileName;
10 }
```

（c）第二种处理代码

图 5.6: 路径遍历漏洞代码片段图

使用原版 SpotBugs 和 Find-sec-bugs 工具对这两份代码进行漏洞扫描，将同时报出路径遍历漏洞。但在本文实现系统中，漏洞扫描结束后，将使用 Joana 切片工具对报出的两个漏洞分别进行切片得到漏洞上下文，如图 5.7所示。可以看出，二者上下文的差异也体现 *doSomething*() 方法的切片中。

接下来，本文实现系统将对两份上下文进行预处理，并使用 N-gram 语言模型提取 3-gram 特征，最后使用本文实现随机森林分类模型进行误报过滤。结果

```
1::ENTR::entry::null::demo.PathB.readFile(java.lang.String)
6::CALL::call::Ljava/lang/String::v5 = p0 $this.doSomething(p1 $param)
11::NORM::declaration::Ljava/io/File::v6 = new java.io.File
78::ENTR::entry::null::demo.PathB.doSomething(java.lang.String)
79::EXIT::exit::Ljava/lang/String::demo.PathB.doSomething(java.lang.String)
83::NORM::compound::Ljava/lang/String::return p1 $param
```

（b）代码上下文

```
1::ENTR::entry::null::demo.PathC.readFile(java.lang.String)
6::CALL::call::Ljava/lang/String::v5 = p0 $this.doSomething(p1 $param)
11::NORM::declaration::Ljava/io/File::v6 = new java.io.File
78::ENTR::entry::null::demo.PathC.doSomething(java.lang.String)
79::EXIT::exit::Ljava/lang/String::demo.PathC.doSomething(java.lang.String)
83::EXPR::assign::Ljava/lang/Object::v6 = #(294) - #(86)
84::PRED::IF::Z::if (v6 <= #(200)) goto 17
86::EXPR::assign::null::PHI v9 = #(savePath), p1 $param
87::NORM::compound::Ljava/lang/String::return v9
```

（c）代码上下文

图 5.7: 路径遍历漏洞代码片段图

显示，本系统将（b）代码标记为真实漏洞，将（c）代码标记为误报漏洞，即系统成功识别过滤了该误报，这说明本系统可以有效降低静态扫描工具误报率。

5.5　本章小结

本章主要对人机协同 Java 漏洞扫描系统进行功能和性能测试，并对系统提出的误报过滤体系进行了实验评估。首先，对系统测试环境进行了相关软硬件描述，主要介绍使用到的服务器配置以及运行程序版本和角色。其次，根据第三章用例描述阐述了各测试用例设计，并展示了测试执行结果；同时，利用 JMeter 对系统两个最常用接口进行了压力测试，并通过测试结果汇总表和响应时间图进行说明。然后，使用 OWASP 测试数据对误报过滤模型进行实验分析，验证误报过滤体系能有效降低静态扫描工具误报率。最后，对比展示了两份潜在路径遍历漏洞的相似代码，并对比了本文实现系统对二者的扫描过程和结果，使用具体例子展示了本文实现系统的实际效果。

第六章　总结与展望

6.1　总结

软件系统规模不断增大，软件安全问题也层出不穷，在源码级别对程序进行漏洞扫描可以一定程度减少漏洞产生。多数静态扫描工具基于词法分析扫描漏洞，采用过度近似策略以保证结果的完备性，这导致其误报率过高。针对这一问题，本文设计并实现人机协同 Java 字节码漏洞扫描系统，通过机器误报过滤以及人工众包审核，有效降低漏洞扫描工具误报率，为开发人员节约成本。

本文为实现上述目标，主要阐述了以下三个核心方面的设计与实现：

首先对比选取市场上主流漏洞扫描工具，并根据 CWE 标准为工具漏洞列表补充相关描述信息。最终选取开源工具 SpotBugs 和 Find-sec-bugs 插件作为 Java 字节码漏洞扫描核心工具，并将其改造封装成符合系统接入标准的统一工具，主要实现 start、status 和 stop 三个原语操作。同时提供相应 Dockerfile 部署文件为后续 docker 化工作提供扩展。借助 Joana 程序后向切片工具提取漏洞相关上下文内容，作为漏洞描述扩充项之一。

其次利用 OWASP benchmark1.1 数据集进行初始误报过滤分类模型的训练与评估，其中 90% 数据为训练集，10% 数据为测试数据集。分类主要针对对象为漏洞上下文信息，对漏洞上下文进行分词、去除停用词等预处理操作，再利用 N-gram 提取三元特征，最后使用随机森林作为分类器进行模型训练。

最后，为进一步迭代改进误报过滤模型，系统接入众包审核服务，众审专家对漏洞进行人工审核，并融合机器扫描过滤和人工审核报告。众审结果可以提取为带正误报标签的漏洞库，为误报过滤迭代训练提供数据支撑。

本系统使用 Angular2 作为前端架构，Pug 模板引擎作为静态报告生成框架，为用户提供友好的交互方式。使用 Spring Boot 作为后端框架，并使用 Docker 进行独立部署，提供轻量级微服务。同时使用 Redis 作为缓存以减少数据库访问压力，MongoDB 作为数据库存储大量漏洞信息。系统与其他服务通过 HTTP 协议进行数据传输，并对外提供 RESTful 接口。

本文首先阐述了研究背景与意义，并介绍相关技术及工具。其次，通过框架图、流程图、用例图等描述了系统概要设计和各模块功能设计。然后，通过具体时序图和关键代码描述了系统详细设计与实现方法。最后，对系统功能和性能进行测试，并通过实验验证系统可以有效降低漏洞扫描工具误报率。

6.2 展望

本文设计并实现了一个人机协同 Java 字节码漏洞扫描系统并已部署到线上服务器，此系统致力于为用户提供低误报低漏洞的漏洞扫描服务。为进一步提高系统效果，未来将针对以下几个方式改进系统。

1. 扩展扫描器数量。目前本系统仅支持单扫描器，后续为系统改造并集成多个扫描器，扫描器接入均需符合统一接入标准。通过多个扫描器扫描结果融合，以确保结果完备性，从而降低漏报率。

2. 扩展漏洞特征。目前本系统使用 Joana 程序后向切片工具提取漏洞上下文，其在大型项目上存在时间隐患，后续需对其进一步优化。同时，系统目前使用 N-gram 语言模型提取上下文三元特征，后续将进一步扩展使用上下文的语义语法等特征。

3. 扩展误报过滤模型。目前本系统使用随机森林作为误报过滤分类器，该分类模型在小量数据集上效果较好。后续随着真实项目漏洞数据库扩展，将尝试使用深度学习等方法在大量数据集上训练模型，选取效果优胜者作为系统过滤模型，从而提供系统准确率。

4. 适配漏洞类型。目前对于所有类型误报过滤，本系统均使用随机森林分类模型，该模型在各类型漏洞上过滤效果有明显差异。后续将针对特定类型漏洞，使用特定分类算法进行模型训练，将过滤模型与漏洞类型适配，进一步降低误报率。

此外，随着系统用户量增长以及访问需求增长，将采用分布式部署方式以满足更高的并发性需求。

参考文献

[1] C. Banerjee, S. Pandey, Research on software security awareness: problems and prospects, ACM SIGSOFT Software Engineering Notes 35 (5) (2010) 1–5.

[2] 360 团队, 360 代码安全保障系统 v5.0 技术白皮书 (2015) 1–2.

[3] 中国测评 CSTC, 2019 中大型政企机构网络安全建设发展趋势研究报告 (2019) 2–4.

[4] T. Ball, B. Cook, V. Levin, S. K. Rajamani, Slam and static driver verifier: Technology transfer of formal methods inside microsoft, in: International Conference on Integrated Formal Methods, Springer, 2004, pp. 1–20.

[5] O. Tripp, S. Guarnieri, M. Pistoia, A. Aravkin, Aletheia: Improving the usability of static security analysis, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 762–774.

[6] B. Johnson, Y. Song, E. Murphy-Hill, R. Bowdidge, Why don't software developers use static analysis tools to find bugs?, in: 2013 35th International Conference on Software Engineering (ICSE), IEEE, 2013, pp. 672–681.

[7] M. D. Ernst, Invited talk static and dynamic analysis: synergy and duality, in: C. Flanagan, A. Zeller (Eds.), Proceedings of the 2004 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis For Software Tools and Engineering, PASTE'04, Washington, DC, USA, June 7-8, 2004, ACM, 2004, p. 35.

[8] P. Louridas, Static code analysis, IEEE Software 23 (4) (2006) 58–61.

[9] 肖锋, 张玉清, 源码审核技术中的词法分析, 中国科学院研究生院学报 26 (03) (2009) 408–414.

[10] 周严, 基于污点分析的静态漏洞检测可扩展框架, Master's thesis, 南京大学 (2017).

[11] T. Copeland, PMD applied, Vol. 10, Centennial Books Arexandria, Va, USA, 2005.

[12] 王斌, 基于 Fortify SCA 的隐通道分析技术的研究与实现, Master's thesis, 北京理工大学 (2016).

[13] T. Kremenek, D. Engler, Z-ranking: Using statistical analysis to counter the impact of static analysis approximations, in: International Static Analysis Symposium, Springer, 2003, pp. 295–315.

[14] B. Chimdyalwar, P. Darke, A. Chavda, S. Vaghani, A. Chauhan, Eliminating static analysis false positives using loop abstraction and bounded model checking, in: International Symposium on Formal Methods, Springer, 2015, pp. 573–576.

[15] I. Medeiros, N. Neves, M. Correia, Detecting and removing web application vulnerabilities with static analysis and data mining, IEEE Transactions on Reliability 65 (1) (2015) 54–69.

[16] J. Yoon, M. Jin, Y. Jung, Reducing false alarms from an industrial-strength static analyzer by SVM, in: Proceedings of 21st Asia-Pacific Software Engineering Conference, Vol. 2, IEEE, 2014, pp. 3–6.

[17] U. Koc, P. Saadatpanah, J. S. Foster, A. A. Porter, Learning a classifier for false positive error reports emitted by static code analysis tools, in: Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, 2017, pp. 35–42.

[18] S. Arzt, S. Rasthofer, R. Hahn, E. Bodden, Using targeted symbolic execution for reducing false-positives in dataflow analysis, in: Proceedings of the 4th ACM SIGPLAN International Workshop on State of the Art in Program Analysis, 2015, pp. 1–6.

[19] 章晓芳, 冯洋, 众包软件测试技术研究进展, 软件学报 29 (1) (2018) 69–88.

[20] T. D. LaToza, A. Van Der Hoek, Crowdsourcing in software engineering: Models, motivations, and challenges, IEEE software 33 (1) (2015) 74–80.

[21] Z. S. Alwan, M. F. Younis, Detection and prevention of sql injection attack: A survey, International Journal of Computer Science and Mobile Computing 6 (8) (2017) 5–17.

[22] S. Edwards, J. Spacco, D. Hovemeyer, Can industrial-strength static analysis be used to help students who are struggling to complete programming activities?, in: Proceedings of the 52nd Hawaii International Conference on System Sciences, 2019, pp. 1–10.

[23] K. Liu, D. Kim, T. F. Bissyandé, S. Yoo, Y. Le Traon, Mining fix patterns for findbugs violations, IEEE Transactions on Software Engineering (2018) 1–1.

[24] E. H. Oskouei, O. Kalıpsız, Comparing bug finding tools for java open source software (2018).

[25] B. Xu, J. Qian, X. Zhang, Z. Wu, L. Chen, A brief survey of program slicing, ACM SIGSOFT Software Engineering Notes 30 (2) (2005) 1–36.

[26] M. Weiser, Program slicing, IEEE Transactions on software engineering (4) (1984) 352–357.

[27] K. J. Ottenstein, L. M. Ottenstein, The program dependence graph in a software development environment, ACM Sigplan Notices 19 (5) (1984) 177–184.

[28] S. Horwitz, T. Reps, D. Binkley, Interprocedural slicing using dependence graphs, ACM Transactions on Programming Languages and Systems (TOPLAS) 12 (1) (1990) 26–60.

[29] J.-F. Bergeretti, B. A. Carré, Information-flow and data-flow analysis of while-programs, ACM Transactions on Programming Languages and Systems (TOPLAS) 7 (1) (1985) 37–61.

[30] V. Berzins, Software Merging and Slicing, IEEE Computer Society Press, 1995.

[31] I. S. Chung, W. K. Lee, G. S. Yoon, Y. R. Kwon, Program slicing based on specification, in: Proceedings of the 2001 ACM symposium on Applied computing, 2001, pp. 605–609.

[32] W. K. Lee, I. S. Chung, G. S. Yoon, Y. R. Kwon, Specification-based program slicing and its applications, Journal of Systems Architecture 47 (5) (2001) 427–443.

[33] K. Li, H. Xu, Y. Wang, D. Povey, S. Khudanpur, Recurrent neural network language model adaptation for conversational speech recognition., in: Interspeech, ISCA, 2018, pp. 3373–3377.

[34] L. Verwimp, J. Pelemans, P. Wambacq, et al., Character-word lstm language models, arXiv preprint arXiv:1704.02813.

[35] Y. Kim, Y. Jernite, D. Sontag, A. M. Rush, Character-aware neural language models, in: Thirtieth AAAI Conference on Artificial Intelligence, AAAI Press, 2016, pp. 2741–2749.

[36] J. Tanha, M. van Someren, H. Afsarmanesh, Semi-supervised self-training for decision tree classifiers, International Journal of Machine Learning and Cybernetics 8 (1) (2017) 355–370.

[37] A. Mathur, G. M. Foody, Multiclass and binary svm classification: Implications for training and classification users, IEEE Geoscience and remote sensing letters 5 (2) (2008) 241–245.

[38] P. F. Brown, V. J. D. Pietra, P. V. de Souza, J. C. Lai, R. L. Mercer, Class-based n-gram models of natural language, Computational Linguistics 18 (4) (1992) 467–479.

[39] T. Huang, Y. Nevmyvaka, A practical markov chain monte carlo approach to decision problems., in: FLAIRS Conference, AAAI Press, 2001, pp. 520–524.

[40] M. Pal, Random forest classifier for remote sensing classification, International Journal of Remote Sensing 26 (1) (2005) 217–222.

[41] K. J. Lidl, Understanding docker, login Usenix Mag. 42 (4).

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

[43] 张仁良, 软件架构中的非功能需求, 微型电脑应用 1 (2009) 61–64.

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

[45] 孟祥双, 前后端分离式 web 应用开发研究, 电子元器件与信息技术 3 (6) (2019) 40–43.

[46] E. Ayanoglu, Y. Aytas, D. Nahum, Mastering RabbitMQ, Packt Publishing Ltd, 2016.

[47] P. Bertelsen, Semantics of java byte code (1997).

[48] 佘青, 利用 apache jmeter 进行 web 性能测试的研究, Ph.D. thesis (2012).

[49] M. Hossin, M. Sulaiman, A review on evaluation metrics for data classification evaluations, International Journal of Data Mining & Knowledge Management Process 5 (2) (2015) 1.

简历与科研成果

基本情况 蒋燕，女，汉族，1996 年 5 月出生，安徽省安庆市人。

教育背景

2018.9～2020.6 南京大学软件学院 硕士

2014.9～2018.6 南开大学软件学院 本科

读研期间成果

1. 江苏省博士后科研资助计划：基于持续融合模型的移动应用测试自动生成（2018K028C），2018-2019。
2. 南京南瑞集团项目：调度自动化软件安全漏洞与风险实验能力提升关键技术研究，2018-2019。
3. 房春荣，**蒋燕**，史洋洋，陈振宇，李玉莹，“一种结合源代码语义与语法特征的基于 CNN 的 bug 定位方法”，申请号：2019109519990，已受理。
4. 房春荣、史洋洋、**蒋燕**、陈振宇、李玉莹，“一种基于微服务的安卓众包在线验证方法”，申请号：2019109519986，已受理。
5. 房春荣、史洋洋、**蒋燕**、王逸璠、陈振宇、李玉莹，“一种功能粒度上基于语义信息的源代码相似度评估方法”，申请号：2019109519971，已受理。
6. 房春荣、龚爱、徐文远、**蒋燕**、李玉莹、陈振宇，“一种基于 Android 源代码的风险等级预测方法”，申请号：201810092876.1，已受理。

致　　谢

在本篇毕业设计完成之际，我向所有在我毕设完成过程中指导和帮助过我的人致以最真挚的感谢。

首先，感谢我的指导老师陈振宇老师和房春荣老师，这一次论文写作过程中离不开两位老师的谆谆教诲。在选题之初，两位老师帮我整理思路，并指明可行性方向。本次论文写作期间恰好不幸遭遇新冠肺炎疫情，但在次期间，两位老师多次通过会议线上跟踪我们的项目以及论文进度，并不断提出宝贵意见，不断帮助我改进项目与论文，督促我及时完成项目与论文。在我遇到瓶颈与打击时，房老师多次抽出时间和我进行沟通，鼓励我振奋精神，并给予技术性支持。再次感谢两位老师！

其次，感谢我的实验室“iSE 实验室”以及实验室里众多兄弟姐妹，感谢实验室为我提供良好的学习氛围以及资源平台，感谢实验室成员协助我攻克不熟悉领域的难题，热心为我提供技术性支持与帮助。

我要感谢母校南京大学，两年的研究生生涯虽然短暂但却充实，感谢母校为我提供良好的学习平台，我将谨记“励学敦行，诚朴雄伟”的校训砥砺前行。

此外，我要感谢父母和家人对我的鼓励与帮助，在疫情期间想尽办法为我提供能专心完成项目以及论文的环境。

最后，我要感谢各位论文审稿老师和专家，向你们的专业与敬业致敬。我也将努力成为向你们那样的人，无论未来在什么岗位上，都将认真对待。

版权与原创性说明

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

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

作者签名：

日期: 2020 年 5 月 26 日