南京大學

研 究 生 毕 业 论 文
（申请工程硕士学位）

论 文 题 目 基于业务监控数据的 **Web** 服务诊断系统设计与实现

作 者 姓 名 孙加辉

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

2021 年 **5** 月 **20** 日

学　　号：**MF1932155**

论文答辩日期：**2021** 年 **5** 月 **20** 日

指 导 教 师：　　（签字）

The Design and Implementation of Web Service Diagnosis System Based on Business Monitoring Data

by

Jiahui Sun

Supervised by

Professor Zhenyu Chen

A dissertation submitted to
the graduate school of Nanjing University
in partial fulfilment of the requirements for the degree of
MASTER OF ENGINEERING
in
Software Engineering

Software Institute
Nanjing University

May 20, 2021

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：基于业务监控数据的 Web 服务诊断系统设计与实现
工程硕士（软件工程领域）专业 2019 级工程硕士生姓名：孙加辉
指导教师（姓名、职称）：陈振宇 教授

摘　　要

Web 服务具有跨平台性与开放性，可为所有安装浏览器的终端提供服务。为了更方便地向用户提供服务，大批互联网企业选择开发自己的 Web 服务。部分互联网企业缺乏专业的测试人员，为了保证服务质量，他们通常会将 Web 服务的测试工作外包给测试机构完成，从而导致 Web 服务开发成本提高与开发速度降低。如何在保证 Web 服务开发成本与开发速度的前提下减少服务中的缺陷，是互联网企业关注的重要问题之一。

在此背景下，本文设计并实现了基于业务监控数据的 Web 服务诊断系统，通过业务监控数据获取用户请求在 Web 服务中的流转路径以及各业务模块的运行信息，帮助开发人员与运维人员诊断出 Web 服务中的低性能方法与未进行权限控制的接口。业务监控数据是指 Web 服务中各业务模块在响应用户请求时所产生的日志等数据。本系统借助 SkyWalking 收集 Web 服务中接口调用等数据，通过 Spring AOP 收集 Web 服务中各方法运行信息等数据。本系统将 Web 服务中各方法根据运行时间等参数进行加权处理并聚类，根据聚类结果与历史诊断结果分析出 Web 服务中低性能系统方法。本系统通过检测用户访问序列中异常来诊断 Web 服务中未进行权限控制的接口，首先从历史正常访问序列中挖掘频繁序列模式，之后将当前用户访问序列与频繁序列模式进行模式匹配，根据匹配结果分析当前用户的访问行为是否存在异常，并获取 Web 服务中未进行权限控制的接口等信息。

本系统已在慕测平台测试环境部署并稳定运行两周，诊断出该 Web 服务存在 7 个低性能方法与 2 个未进行权限控制的接口。通过功能测试与效果测试证明本系统可以帮助开发人员或运维人员诊断出 Web 服务中低性能方法与未进行权限控制的接口。

关键词：Web 服务诊断，自动化测试，业务监控，异常检测，缺陷定位

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 Web Service Diagnosis System Based on Business Monitoring Data

SPECIALIZATION: Software Engineering

POSTGRADUATE: Jiahui Sun

MENTOR: Professor Zhenyu Chen

Abstract

Web services are cross-platform and open, and can provide services for all terminals that install browsers. In order to provide services to users more conveniently, some enterprises choose to develop their own web services. Some enterprises lack professional testers. In order to ensure the quality of service, they often outsource the testing work of Web services to testing institutions, which leads to the increase of development cost and the decrease of development speed. How to reduce the defects of Web services under the premise of ensuring the cost and speed of Web services development is one of the important issues concerned by Internet enterprises.

In this context, this thesis designs and implements a web service diagnosis system based on business monitoring data, which collect the flow path of user requests in the Web service and running status of each business module through business monitoring. This system can helps developers and operators to diagnose the low-performance methods and interfaces without permission control in Web services. Business monitoring data refer to the log information generated by each business module in the Web service when responding to user requests. The system uses SkyWalking and Spring AOP to collect business monitoring data. SkyWalking collects interface calls in Web services and other data. Spring AOP collects method operation information and other data in Web services. In this system, all system methods are weighted and clustered according to the running time and other parameters, and low performance system methods are obtained according to the clustering results and historical diagnosis results. This system locates the interface without permission control in Web service by diagnosing the anomalies in user access sequences. Firstly, it mines frequent sequence patterns from

user normal access sequences, then matches the current user access sequence with frequent sequence patterns, and weights them according to the frequency of each frequent sequence pattern. Finally, according to the matching results, we analyze whether there is an anomaly in the current user's access behavior, and get the information such as the interfaces without access control.

The system has been deployed in the test environment of mooctest platform for two weeks. In the test environment, it has been diagnosed that there are seven low performance methods and two interfaces without permission control in the web service. it is proved that this system can help developers and operators to quickly locate the low performance methods and the interfaces without permission control in Web services.

keywords: Web Service Diagnose, Automated Testing, Business Monitoring, Anomaly Detection, Defect Location

目　录

图目录

表目录

第一章　引言

1.1　项目背景和意义

随着互联网技术的发展与计算机的普及，人们的工作和生活与 Web 服务越来越密不可分 [1]。由于 Web 服务更新速度快并且支持跨平台使用 [2]，无论是 PC 端还是移动端，用户都可以简单快捷地通过 Web 服务获取实时信息，因此 Web 服务的使用场景非常广泛。根据中国国信网 2021 年发布的《第 47 次中国互联网络发展状况统计报告》[①]统计，截至 2020 年 12 月，在我国注册域名的网站数量为 443 万个，网页数量为 3155 亿个，较 2019 年 12 月同比增长 5.9%，我国网民规模为 9.89 亿，规模庞大的用户群体对 Web 服务质量 [3] 与性能提出了较高要求。

实际业务场景复杂多变，单体架构的 Web 服务内部逻辑变得越来越冗杂，代码的可读性 [4] 与可维护性大大降低。部分互联网企业开始使用微服务架构，将单体架构的 Web 服务拆分为一组高内聚、松耦合的微服务 [5]，其中不同的微服务可以由不同的开发团队独立开发部署，也可以独立进行功能扩展或动态扩容，以应对复杂多变的业务需求。

随着微服务架构的广泛使用，越来越多的互联网企业开始采用此架构开发 Web 服务。大型互联网企业对开发人员编码能力要求较高，且有专门的测试团队和运维团队，大型互联网企业的技术团队通常能够在 Web 服务出现异常时迅速定位并修复系统中的缺陷 [6]，可以很大程度上保证 Web 服务质量。而部分小型互联网企业缺少专业的测试团队与运维团队，通常由开发人员兼顾软件的测试工作与运维工作。然而普通的开发人员往往缺乏专业的软件测试与系统运维知识，且采用微服务架构的 Web 服务系统结构较为复杂，普通开发人员难以快速定位并修复基于微服务架构开发的 Web 服务中的缺陷，无法保证 Web 服务质量与用户体验，从而造成企业的客户流失与经济损失。

目前很多企业选择将软件外包给第三方测试服务提供商进行测试，以减少软件中的缺陷。由于测试服务提供商对待测系统了解不足，需要花费时间进行

[①] http://www.cac.gov.cn/2021-02/03/c_1613923423079314.htm

学习，因此无法在短时间内完成测试任务，从而导致项目上线速度降低。对于部分需要频繁更新项目的小型企业而言，需要频繁对 Web 服务进行测试，测试的费用提高了整个项目的成本。如何在保证 Web 服务的开发成本与开发速度的前提下减少项目中的缺陷，以及如何保证系统高性能与高质量地运行，成了当今互联网企业亟需解决的问题。

在此背景下，本论文设计并实现了一种基于业务监控数据的 Web 服务诊断系统，利用业务监控数据跟踪用户请求在 Web 服务中的流转路径以及收集各业务模块的响应状态，帮助开发人员与运维人员快速诊断出 Web 服务中的低性能方法与未进行权限控制的接口。本系统将慕测平台[①]作为源系统进行监控，使用 SkyWalking 与 Spring AOP 收集业务监控数据，其中 SkyWalking 收集 Web 服务中接口调用等数据，Spring AOP 收集 Web 服务中各方法运行信息等数据。本系统通过检测用户访问序列中的异常来定位 Web 服务中未进行权限控制的接口，首先从历史正常访问序列中挖掘频繁序列模式，之后将频繁序列模式及其子串与当前用户访问序列进行模式匹配，并根据各频繁序列模式的频数进行加权处理，最后基于匹配结果分析当前用户的访问行为是否存在异常，得到未进行权限控制的接口等信息。本系统对所有的系统方法根据运行时间等参数进行加权处理并聚类，根据聚类结果与历史诊断结果得到低性能系统方法。

1.2 研究现状

1.2.1 Web 服务监控技术现状

Web 服务监控 [7] 是查看系统是否稳定运行的重要方式之一。为了保证服务质量与用户体验，互联网企业通常会根据 Web 服务实际使用场景搭建监控系统来实时查看 Web 服务的运行状态。

监控系统可以直接监控 Web 服务本身的运行状态，也可以通过监控服务器的 CPU 使用率或内存使用率等硬件信息间接监控 Web 服务。监控系统可以获取服务器的实时状态，当 CPU 使用率或内存使用率等指标达到预先设定的阈值时便会向管理员发送预警信息。目前大部分监控系统只关注系统运行时的服务器硬件信息，而对业务监控数据的关注度不够。常见的 Web 服务监控技术有 SkyWalking 以及 ELK [8] 等。

[①] http://www.mooctest.net/

SkyWalking 是一款国产应用性能管理 [9] 系统，提供系统链路追踪与指标数据聚合及可视化等功能，能够对 Web 服务以及容器化服务 [10] 等进行监控。SkyWalking 已加入 Apache 基金会并遵循 Apache 的规则建立了开源社区，迄今为止 SkyWalking 社区已经非常活跃，开发人员提出的问题可以迅速得到解决。SkyWalking 使用场景广泛，能够支持 Java、Node.js 以及.NET 等编程语言。SkyWalking 探针对所监控的系统侵入性 [11] 较低，无需修改原系统代码即可收集运行数据。然而 SkyWalking 灵活性较低，除了原生提供的监控项以外，开发人员难以自定义想要获取服务运行数据。

ELK 是 Elasticsearch、Logstash、Kibana 三个开源工具的缩写，提供了一套系统日志监控 [12] 方案。ELK 部署简单，适合中小型企业使用，其中 Elasticsearch 是开源搜索引擎工具 [13]，在本系统中的主要功能是存储数据与查询数据；Logstash 是开源的数据处理工具，主要功能是收集数据与过滤数据；Kibana 是开源的可视化工具，主要功能是将收集的指标数据展示给用户。ELK 适用于所有可以产生日志的应用场景，是当前主流的系统监控解决方案之一，慕测平台的日志监控便是通过 ELK 实现。然而 ELK 运行时需要对日志进行整合处理，因此在处理日志规模较大的场景时效率较低。

1.2.2 Web 服务诊断技术现状

Web 服务诊断是指查找出 Web 服务中存在的缺陷，以保证系统能够稳定运行。常见的 Web 服务诊断技术主要有基于硬件监控信息进行分析诊断与基于服务组合状态矩阵 [14] 进行诊断等。

基于硬件监控信息分析诊断是指收集系统运行时服务器信息，分析服务运行是否存在异常或预测系统是否可能会发生异常。iSE 实验室的张双江等人提出了一种基于 XGBoost 和 LSTM 的智能监控系统，该系统根据 Web 服务运行时服务器的 CPU 使用率、内存使用率以及网络带宽等硬件信息进行趋势预测，从而实现对 Web 服务的诊断。该方式诊断系统状态时需要依据系统运行信息，因此需要借助上述的 Web 服务监控技术提供监控数据。

基于服务组合进行诊断主要适用于系统结构稳定的 Web 服务，即将多个服务的实时运行状态、输入参数以及输出参数作为变量进行建模分析，诊断系统运行是否存在故障。赵雪琴等提出了一种基于路径匹配的故障诊断方法 [15]，该诊断方法根据 Web 服务运行时的输入参数与输出结果等数据构建状态矩阵 [16]，通过对比状态矩阵与实际矩阵间的差异诊断系统是否存在异常。但是

由于业务调整频繁且实现技术更新换代速度快，Web 服务的的系统结构无法长期保持不变，因此基于服务组合的诊断方法难以应用在大部分实际场景中。

1.2.3 Web 服务技术应用现状

服务是指服务商为用户提供的一组功能，Web 服务则是指基于超文本传输协议（http）等标准的互联网协议 [17] 的服务，能够为用户提供便捷的信息浏览与信息交流功能。Web 服务具有开放性与跨平台性，其中开放性是 Web 服务的基础，用户可以通过 Web 服务访问全球共享的海量资源，Web 服务的跨平台性是指 Web 服务可以为不同操作系统或不同浏览器的用户提供服务。Web 服务还可以为开发者用户提供服务，即提供 API 供开发人员调用，无论开发者在开发应用时使用什么编程语言，只需要该编程语言支持 http 或 https 等协议，便可以通过该 Web 服务提供的接口调用服务。Web 服务适用场景众多，学校与政府等机构可以使用 Web 服务实现信息公开，阿里巴巴与慕测科技等互联网企业也可以使用 Web 服务快捷地为用户提供复杂多样的服务。作为一种已经被用户广泛接受的服务方式，Web 服务也存在局限性。首先是 Web 服务业务调整频繁，服务架构可能会在短时间内多次更新换代，对于开发人员而言，需要经常学习最新技术以保证系统的运行效率，对于测试人员与运维人员而言，需要频繁地测试以保证系统的质量。其次是 Web 服务没有客户端，发布的消息无法主动向用户推送，导致用户黏度不足，以致于用户流失。

目前，无论是南京大学 [①]等非盈利性组织，还是阿里巴巴 [②]等盈利性互联网企业，大部分组织或企业都会开发 Web 应用为用户提供服务。其中慕测平台通过 Web 服务为企业用户提供软件自动化测试 [18] 等功能，为学校用户或学生用户提供开发者测试考试或练习等功能，借助 Web 服务的开放性等优势，能够快捷方便地为更多的学生或教师提供服务。

据《第 47 次中国互联网络发展状况统计报告》统计，截至 2020 年底，我国网民使用手机上网的比例高达 99.7%。随着移动网络的发展与智能移动终端的普及，Web 服务在移动端大放异彩。微信等移动端应用支持内嵌 Web 网页 [19]，用户可以通过固定入口访问网站，可以便捷地与其他用户进行分享，提高消息传输的效率。

[①]https://www.nju.edu.cn/
[②]https://www.1688.com/

1.3 本文主要工作

Web 服务具有支持跨平台使用、方便扩展以及内容更新速度快的特点，在互联网企业大部分实际场景中得到了广泛应用。目前的监控技术更多关注于服务器内存使用率等硬件信息，对业务监控数据关注度不足，导致大量有用信息被忽视。上文所述的诊断技术要求所诊断的 Web 服务系统架构稳定且内部逻辑不会频繁变更，而实际业务复杂多变，Web 服务内部逻辑通常会变更频繁，因此这些诊断技术难以应用于实际场景中。为了帮助开发人员或运维人员快速诊断出 Web 服务中的低性能方法与未进行权限控制的接口，本文设计并实现了基于业务监控数据的 Web 服务诊断系统。本系统通过 SkyWalking 与 Spring AOP 两种方式收集业务监控数据，诊断系统中是否存在异常，其中 SkyWalking 收集 Web 服务中接口调用等数据，Spring AOP 收集 Web 服务中各方法运行信息等数据。本文的主要工作如下：

（1）收集并处理业务监控数据。本系统首先通过 SkyWalking 监控平台和 Spring AOP 技术获取源系统的业务监控数据；其次通过 Filebeat 转发 Spring AOP 获取的日志信息；之后系统将收集的业务数据进行持久化，保存到 Elasticsearch 搜索引擎中，以便数据处理时使用；最后本系统从 Elasticsearch 搜索引擎中请求数据，根据时间信息与用户 id 进行处理，并将筛选处理后的业务监控数据保存到 MySQL 数据库中。

（2）用户异常行为诊断。本系统首先从数据库中获取用户指定时间段内正常的历史访问序列，过滤访问序列中 heartbeat 等无效信息；其次根据正常用户访问序列挖掘频繁序列模式；之后借助 KMP 算法将用户访问序列与频繁序列模式及其子串进行匹配，根据匹配结果诊断当前访问序列是否异常；最后通过当前访问序列中的异常信息将可能未进行权限控制的接口展示给用户。

（3）系统方法异常诊断。本系统首先从 MySQL 数据库中获取系统方法信息，过滤其中的访问错误信息，对系统方法运行时的运行时间、CPU 使用以及 JVM 使用等信息进行归一化以及加权处理；其次根据处理后的属性对系统方法进行聚类；最后根据聚类结果将低性能系统方法展示给用户。

（4）实现具备上述功能的系统。本系统中用户只需打开相应页面并选择需要监控的时间段，即可查看到最终的诊断结果。系统实现了可视化，用户可以在浏览器中进行所有操作，查看用户异常行为以及未进行权限控制的接口等信息，也可以查看性能低下的系统方法信息。

（5）系统测试。本系统通过 Docker 容器进行部署运行，本文对系统进行相关的功能性测试、性能测试以及效果测试，测试本系统功能的可用性以及本系统在指定运行环境中对高并发场景的支持度。本文将慕测平台作为源系统进行诊断，验证本系统的有效性。

1.4 本文组织结构

本文共分为六个章节，组织结构如下：

第一章，引言部分。本章首先介绍了基于业务数据监控的 Web 服务诊断系统设计与实现的背景和意义；之后从 Web 服务监控技术现状、Web 服务诊断技术现状以及 Web 服务技术应用现状多个方向分析研究现状；最后介绍了设计与实现基于业务监控数据的 Web 服务诊断系统的主要工作与论文的组织结构。

第二章，相关技术综述。本章主要对实现本系统时所涉及到的相关技术框架和算法做进一步的介绍和分析。主要包括监控数据收集与处理技术、前后端技术框架与容器技术以及进行异常诊断的算法等三大技术板块。

第三章，系统需求分析与概要设计。本章首先对基于业务监控数据的 Web 服务诊断系统进行涉众分析与需求分析，根据分析结果确定系统的用户群体、功能需求以及非功能需求；之后借助系统用例图与系统用例描述对系统进行详细的分析；之后介绍系统的总体设计，借助“4+1”视图对系统总体架构进行多角度的描述。最后对本系统进行了模块划分，介绍了系统中各功能模块的流程设计以及数据库模型的设计。

第四章，系统详细设计与实现。本章主要对业务监控数据收集模块、业务监控数据处理模块、用户访问序列异常诊断模块以及系统方法异常诊断模块的设计与实现进行详细说明，分别介绍了各模块的总体架构设计、核心类图、顺序图以及关键代码。

第五章，系统测试与案例展示。本章主要对系统进行进行功能测试、性能测试以及效果测试。在慕测平台测试环境部署本系统，并对其进行一周的诊断以及模拟不同的场景对本系统进行测试。通过功能测试结果、性能测试结果以及实际服务的诊断结果来验证本系统的可用性。

第六章，总结和展望。本章主要对设计与实现本系统所做的工作进行总结，分析本系统目前已经达到的效果，并提出本系统的不足与未来工作方向。

第二章　相关技术综述

2.1　基于 SkyWalking 的 Web 服务监控技术

随着企业的发展，业务复杂性不断增加，对 Web 服务提出了更高要求，单体的 Web 服务难以满足日益复杂的业务需求。为了解决上述问题，开发人员引进了微服务架构，将单体 Web 服务拆分为多个低耦合且可独立开发部署的微服务。微服务架构可以缓解业务复杂引发的问题，但是却提高了整个系统的复杂性，一个微服务发生故障可能导致整个系统的崩溃，因此快速定位发生故障的位置至关重要。SkyWalking 是一个开源的应用性能监控系统，能够对微服务以及容器化服务等进行监控，能够快速地定位 Web 服务中的问题。SkyWalking 通过 java 探针收集需要的指标，可以收集、分析、聚合并可视化来自不同服务间的数据，实现分布式追踪 [20]，根据收集的指标确定服务间的关系。SkyWalking 监控时无需修改原系统代码，对所监控的系统侵入性较低。

SkyWalking 的总体架构分为三个部分：SkyWalking-collector、SkyWalking-web、SkyWalking-agent。其中 SkyWalking-agent 为 java 探针，可以低侵入性地收集与发送所监控应用的指标数据，与需要监控的应用部署在同一服务器上，如果应用以分布式的方式部署，则需要在相应的多个服务器上部署 SkyWalking-agent；SkyWalking-collector 是指标数据收集器，接收并持久化 SkyWalking-agent 发送的数据，SkyWalking-collector 可以使用 Elasticsearch 与 H2 等方式持久化数据，可以与监控的应用部署在不同服务器上；SkyWalking-web 是 SkyWalking 的可视化平台，用来展示整合的数据。

本系统中通过 SkyWalking 收集 Web 服务中接口调用等数据，通过 Elasticsearch 实现业务监控数据的持久化。本系统中在单独的服务器上部署 SkyWalking-collector，并在所有部署 Web 服务的服务器上运行 SkyWalking-agent，由于在本系统中不需要对 SkyWalking 收集监控数据实现可视化，所以无需部署 SkyWalking-web。

除 SkyWalking 以外，常用的应用性能监控系统还有 Zipkin、CAT 等。Zipkin 是开源的分布式跟踪系统，根据时间序列收集服务的运行指标，Zipkin

可以灵活配置需要监控的指标，但是需要在所监测的系统代码中修改配置信息，对所监测系统的侵入性较高，且配置较为复杂，上手难度较高，因此本系统未选用 Zipkin 作为监控工具；CAT 是开源的实时应用监控系统，主要功能是实时监控应用的运行情况，CAT 的实现方案是通过在代码中进行埋点来实现监控，比如拦截器与注解等，对所监测系统代码的侵入性较高，集成成本较高，风险较大，因此本系统未选用 CAT 作为监控工具。经过分析比较，本系统最终选用 SkyWalking 作为监控工具。

2.2 基于 Spring AOP 的 Web 服务日志收集技术

本系统所监控的项目基于 SpringBoot 框架 [21] 开发，SpringBoot 框架旨在帮助开发人员通过最少的配置尽快地搭建 Web 项目。SpringBoot 框架内嵌 Tomcat 等多种 Web 服务器 [22]，可简化部署流程，降低部署成本。SpringBoot 框架提供 IOC 和 AOP 等特性，能够提高 Java 开发人员的开发效率。

AOP 即面向切面编程 [23]，是面向对象编程（OOP） [24] 的进阶与补充。AOP 的目标是保证开发人员在不修改业务源码的前提下，可以为程序中的业务源码添加某种或某些通用功能。AOP 框架有很多种，主流的有 AspectJ [25] 与 Spring AOP [26] 两种方式，其中 AspectJ 是静态实现，在编译时生成代理对象，而 Spring AOP 是动态 AOP 实现，在程序运行阶段动态地生成代理对象，与前者相比，Spring AOP 的使用更加便捷。通过 Spring AOP，开发人员可以将通用的非核心业务逻辑从核心业务逻辑中解耦，使开发人员在开发程序时能够专注于核心的业务逻辑而不用考虑事务处理、日志管理以及异常处理等非核心业务逻辑，从而提高开发效率。获取系统运行时指标数据的方式有多种，除了上述通过 SkyWalking 以外，Zipkin 与 CAT 也是监控系统的常用方式。与 Zipkin 等方式相比，Spring AOP 对所监控系统的侵入性更低，且易于维护，为了尽可能减少监控对原系统的影响，本系统选择复用原系统获取运行信息的方法，通过 Spring AOP 获取系统日志。

本系统通过 Spring AOP 收集 Web 服务中各方法运行信息等。由于 SkyWalking 无法收集到本系统需要的全部数据，因此需要借助 Spring AOP 获取部分数据，并通过 Filebeat 转发获取的日志信息。Filebeat 是使用 Golang 实现的轻量级日志收集器，用于转发和集中日志信息，可以根据配置将指定日志信息转发到 Elasticsearch 与 Logstash 等工具中。Filebeat 具有高可靠性，能够读取转

发指定日志行，当输出日志的系统发生故障时，Filebeat 会等待系统恢复，并在系统恢复后从日志中断的位置继续转发。Filebeat 使用便捷，通过简单的命令即可收集、解析以及可视化常见格式的日志。当日志转发到 Elasticsearch 或 Logstash 时，Filebeat 会采用灵活的处理方式应对大量的数据，当 Elasticsearch 或 Logstash 处理数据发生拥堵时，Filebeat 也会相应的降低日志读取速度，当 Elasticsearch 或 Logstash 处理数据的速度恢复时，Filebeat 也会恢复原有的读取速度。Filebeat 支持容器化，可以在 Docker 中部署 Filebeat 便可获取完整的日志信息，简化系统部署流程，降低系统部署成本。

本系统使用 Elasticsearch 作为监控信息的持久化工具。Elasticsearch 是一个开源的数据存储与数据搜索工具，遵循 RESTful 风格，能够快捷地整合与搜索大量数据。Elasticsearch 适用范围广，可以运行在单台服务器上处理少量数据，也可以运行在大型分布式集群上处理海量数据。Elasticsearch 是一个轻量级应用，部署方便，对服务器的硬件要求低。Elasticsearch 可以作为 MySQL 等数据库的一种补充，提供全文检索、相关度排名以及复杂数据分析等传统数据库所不能提供的功能。

2.3 基于 PrefixSpan 的用户访问序列异常诊断技术

本系统基于 PrefixSpan 算法诊断 Web 服务中用户访问序列是否存在异常。PrefixSpan（Prefix-Projected Pattern Growth）算法 [27]，即前缀投影 [28] 的模式挖掘，是一个使用数据集投影技术的序列模式 [29] 挖掘算法。本算法的核心思想是分治与深度优先搜索，基于序列数据集生成投影数据集，并对生成的投影数据集依次进行处理，挖掘频繁序列模式。PrefixSpan 算法从长度为 1 的元素开始，将其作为序列模式搜索初始序列数据集，分别得到各序列模式所对应的后缀，根据预先定义的支持度阈值淘汰出现频率未达到该阈值的序列模式，将得到的后缀作为下次递归挖掘的序列数据集。之后再以相同的方式递归挖掘频繁序列模式，直到无法再挖掘出更长的频繁序列模式为止。

表2-1是一个示例序列数据集，其中 A、B 等字母表示不同事件，事件左右顺序代表事件发生的时间顺序，如序列 ABC 的含义是 A 事件发生后 B 事件发生，最后 C 事件发生，而同一括号内的事件则没有时间顺序，如（AB）则代表当前可能是 A 事件先发生，也可能是 B 事件先发生。PrefixSpan 算法的目的是挖掘一个序列数据集中所有频繁序列模式，频繁序列模式是指原序列数据

集中满足支持度阈值的序列。以表2-1中展所示的序列数据集为例，预先定义支持度阈值为 3。首先，PrefixSpan 算法会统计序列数据中所有的元素，并计算每个元素的支持度，分别为 {A:4,B:4,C:5,D:2}。由于元素 D 的支持度小于预先定义的支持度阈值，所以从频繁序列模式集合中淘汰元素 D，并从原始序列数据集中删除元素 D 得到新的序列数据集 {A(AB)AC,BC,(CAB)BC,C(AB)C,ACC} 以及只有一个元素的频繁序列模式集合 {A,B,C}。之后使用分治的方法分别考虑三个频繁序列模式。以频繁序列模式 A 为例，对于序列数据组中的每一序列：如果该序列不包含频繁序列模式 A，则将该序列变为空；如果该序列包含频繁序列模式 A 且该序列中的第一个 A 不在 () 内，则取当前位置的后缀作为新的序列，如序列 A(AB)AC 更新为序列 (AB)AC；如果该序列包含序列模式 A 且序列中的第一个 A 在 () 内，则使用符号'_' 来代替 A，并删除之前的数据，如序列 (CAB)BC 更新为 (_B)BC。更新后的数据重复上述步骤，直至所有的序列为空，无法挖掘新的频繁序列模式。

表 2-1: PrefixSpan 示例初始序列数据表

序列 ID	Sequence
1	A(AB)DAC
2	BC
3	(CAB)BC
4	C(AB)C
5	ADCC

除了 PrefixSpan 算法以外，被广泛使用频繁项集挖掘算法还有 Apriori [30] 算法与 FP-growth [31] 算法等。Apriori 算法实现简单，因此基于 Apriori 算法的程序开发成本低，而且 Apriori 算法适用于稀疏数据集，使用场景广泛，但是由于该算法执行时需要多次遍历数据集，所以该算法的执行效率较低。FP-growth 算法使用树形数据结构，该算法的挖掘过程中不需要产生候选频繁项集，而是直接生成频繁项集，减少了挖掘过程中遍历数据集的次数，因此与其他算法相比，FP-growth 算法在挖掘频繁项集时具有较高的执行效率。Apriori 算法和 FP-growth 算法只能挖掘频繁项集，均不适用于频繁序列模式的挖掘，而本系统中事件发生的先后顺序至关重要，因此选用 PrefixSpan 算法来挖掘 Web 服务中用户正常访问序列的频繁序列模式。

本系统通过 PrefixSpan 算法从历史用户正常访问序列中挖掘频繁序列模式，作为用户的历史正常行为模式集，再通过匹配历史正常行为模式集与用户

当前访问序列，分析该用户访问序列中是否存在异常，进而诊断该访问序列中是否存在未进行权限控制的接口。本系统使用 KMP 算法进行模式匹配，提高系统的执行效率。

2.4 基于 DBSCAN 的低性能方法诊断技术

本系统基于 DBSCAN 算法诊断 Web 服务中低性能方法。DBSCAN（Density—Based Spatial Clustering of Application with Noise）算法 [32] 是一种基于密度的空间聚类算法，目的是找出所有样本点中的样本点高密度区域，将这些样本点高密度区域作为聚类簇，每个聚类簇都是一个单独的类别。本算法首先寻找一个核心点，之后再从该核心点向其密度可达的区域扩展，从而得到包含该核心点的样本点高密度区域，之后再对剩余的样本点重复上述操作，直至所有的样本点都被处理完。本算法有两个重要参数和一个重要衡量方法 [33]，两个重要参数分别是邻域半径 eps 以及邻域内最少样本点阈值 minPts，一个重要衡量方法是计算两个样本点之间距离的方法，常用的衡量方法有欧式距离、曼哈段距离以及余弦相似度等，其中欧式距离使用最为广泛。

表 2-2: DBSCAN 示例样本数据表

样本点	X 轴坐标	Y 轴坐标
P1	0	0
P2	1	2
P3	3	1
P4	8	8
P5	9	10
P6	10	7
P7	2	0
P8	0	12

表2-2是一个示例样本数据集，X 轴坐标与 Y 轴坐标代表样本的两个属性，在实际使用场景中样本可能具有更多的属性。以表2-2中示例的数据集为例，本例中设定 eps 为 3，minPts 为 2，并选用欧式距离作为两个样本点之间距离的计算方法。DBSCAN 算法会遍历所有样本点找到核心点，首先考虑样本点 P1，计算其他样本点与 P1 之间的距离，将各样本点与 P1 的距离与 eps 相比，可得 P1 的邻域为 {P1,P2,P7}，P1 邻域包含 3 个样本点，大于 minPts，因此可知该样本点为核心点。将 P1 作为核心点建立簇 C1={P1,P2,P7}，遍历样本

点寻找样本点 P1 密度可达的样本点。由于 P1 的邻域为 {P1,P2,P7}，分别计算 P2 与 P7 的邻域，通过计算可得 P2 的邻域为 {P2,P3}，P7 的邻域为 {P2,P3,P7} ，因为样本点 P1 密度可达样本点 P2，样本点 P2 密度可达样本点 P3，所以样本点 P1 密度可达样本点 P3，因此样本点 P3 也属于簇 C1，继续计算样本点 P3 的邻域，未发现新的样本点，因此停止本次扩展。重新遍历未聚类的样本点，考虑点 P4，重复上述步骤可得簇 C2={P4,P5,P6}。最终聚类结果为：簇 C1={P1,P2,P3,P7}，簇 C2={P4,P5,P6}，而样本点 P8 为边界点。

除 DBSCAN 算法以外，常用聚类算法还有 K-means [34] 算法、基于网格的聚类算法 [35] 以及凝聚层次聚类算法 [36] 等。其中 K-means 算法可以简单高效的处理大型数据集，算法的时间复杂度和空间复杂度较低，实现较为简单，然而使用 K-means 算法时需要预先设定需要聚类的类别数，不适用于无法预知聚类类别数的使用场景。基于网格的聚类算法聚类速度与样本数量无关，因此在处理大型数据集时效率很高，然而基于网格的聚类算法不适合处理高维数据。凝聚层次聚类算法从将单个样本点做为单独簇开始，合并两个相近的簇并循环迭代，直至聚类的数量达到要求为止，凝聚层次聚类算法效率较低，不适用于处理大量数据的场景。

本系统需要根据指定时间段内的系统方法执行时间与 CPU 占用率等信息进行聚类，数据维度较高，并且无法预知聚类的类别数量。而基于密度的聚类算法在使用时无需预知聚类数量，且能高效率的处理高维数据，因此本系统选用基于密度的聚类算法。

2.5 基于 Django 的服务端技术

Django [37] 是一个可以快速搭建并部署 Web 服务的开发框架，基于 Python 语言开发，是开发 Web 服务的主流框架之一。Django 框架的目的是帮助开发人员便捷地开发高质量且易维护的 Web 应用程序。Django 框架具有如下特点：

（1）Django 框架具有强大的数据库功能。Django 框架内嵌 ORM 框架 [38]，数据模型的设计不依赖于特定的数据库类型，在代码中定义模型类与数据库中的表相关联，通过操作对象对数据库中的表进行添加记录、删除记录、修改记录以及查询记录等操作。

（2）Django 遵循 MVC 设计 [39]。MVC 即模型层（Model）、视图层（View）以及控制层（Controller）。MVC 架构通过将代码拆分为不同层次

来将系统的数据处理、业务处理、视图展示解耦，有利于管理业务逻辑复杂的项目代码，提高项目开发效率，便于开发人员对于项目的维护。

（3）Django 框架提供模板系统。Django 框架提供使用简单且易扩展的模板系统，包括 HTML 代码和逻辑控制代码。服务端通过将变量嵌入到 HTML 代码中进行渲染，实现页面的动态展示。本系统为了解耦，将前端与服务端分离开发，因此未使用到该特性。

（4）Django 框架提供缓存系统。当收到客户端首次请求时，Django 的缓存系统会将变量或网页缓存磁盘或内存中。客户端首次请求之后的 http 请求信息中会附带缓存过期时间参数，当缓存过期后 Django 会重新请求数据并缓存，否则会直接从缓存中获取数据返回给客户端。

本系统作为基于业务监控数据的 Web 服务诊断系统，需要对用户访问服务时产生的大量数据进行分析和处理。相较于 Java 等编程语言，Python 对于数据分析与处理的支持更为完善，借助 Python 提供的类库可以便捷地处理大量数据，提高代码的可读性。作为开源框架，Django 框架具有强大的社区和文档支持，便于开发人员快速上手，也便于开发人员在开发过程中遇到问题能够快速解决。与 Flask 等轻量级开发框架不同，Django 框架原生提供更多的功能，包括 Session 管理、缓存管理以及权限认证等。Django 框架基于 Python 开发，Python 语言本身是一门可读性高、用途广泛且类库丰富的编程语言，所以 Django 框架的用途也变得非常广泛，尤其是项目中包含处理或分析大量数据的场景且对编程语言没有特殊需求时，Django 框架通常是开发人员的第一选择。除了 Django 框架以外，SpringBoot 框架也受到广大开发人员的推崇，但是由于编程语言不同，SpringBoot 框架更新代码后重新启动项目速度比 Django 框架慢，所以 Django 框架通常能给开发人员更好的开发体验。为了进一步解耦，本系统实现了前后端分离，后端基于 Django 框架开发，前端则基于 Vue.js 框架开发，前后端之间通过 http 请求进行交互。

2.6 Docker 容器技术

软件的生命周期 [40] 包括问题定义、需求分析、软件设计、编码、测试、运行维护等多个阶段，在一个互联网企业中，软件的各个阶段可能由不同的团队负责，而软件编码阶段开发人员使用的开发环境与测试阶段测试人员使用的测试环境可能存在差异，不同团队的技术人员可能因为运行环境的差异而产生

分歧，通常需要额外花费时间进行交流，从而影响软件开发效率。随着容器技术的推广，上述问题逐渐得到解决。

Docker 是一个轻量级容器引擎 [41]，核心功能是将开发人员所开发的项目代码、数据及其运行环境打包成可运行、可传输的镜像，通过简易的启动命令或启动脚本即可将所打包的镜像部署在指定服务器上，实现了项目一次打包，到处运行，实现了项目运行环境和底层环境间的解耦。容器的使用屏蔽了项目开发与部署之间的环境差异，开发人员可以在 Mac 或 Windows 环境中开发项目，而在 Linux 系统上部署开发完成的项目。启动或停止 Docker 容器时无需启动或停止底层操作系统，提升了项目部署的效率，降低了项目持续集成和持续部署的难度。Docker 容器还可以隔离不同的应用环境，使得对环境要求不同的服务可以部署在相同的服务器上，服务不会受到外界干扰，保证整个系统的安全性，因此本系统选用 Docker 容器进行部署。

2.7 本章小结

本章主要介绍了开发本系统所使用到的算法与技术框架，通过分析不同算法和技术框架的使用场景与区别，阐述了选择该算法和技术框架的原因。首先介绍了 SkyWalking 与 Spring AOP 两个业务监控数据收集技术，并分析了其他监控技术的优缺点。之后介绍了频繁序列模式挖掘算法 PrefixSpan 算法，并分析比较该算法与其他相似算法的优缺点。之后介绍了聚类算法 DBSCAN 算法，并分析了其他聚类算法的优缺点与使用场景。最后介绍了系统开发框架 Django 以及系统编排部署的容器工具 Docker。

第三章　系统需求分析与概要设计

3.1　系统整体概述

随着计算机与智能移动设备的普及，人们对于信息交流的效率与质量要求越来越高，Web 服务便是在这样的背景下提出的一种利用网络技术提高信息交流效率的解决方案。Web 服务可以处理简单的请求，也可以处理业务逻辑复杂的请求。当前主流的 Web 服务开发框架有 SpringBoot 与 Django 等，Web 服务开发框架能够帮助开发人员快速便捷地开发 Web 服务。SpringBoot 等主流 Web 服务框架支持通过注解等方式对接口进行权限控制，然而在开发人员开发过程中容易忽视对某些接口进行权限控制，导致部分用户可以跳过前置步骤直接调用某些接口，如在某些考试系统中用户可以跳过阅卷而直接调用相应接口向系统中写分，系统的安全性难以保证。

随着业务逻辑复杂度日益增加，互联网企业对 Web 服务的性能与质量要求不断提高，为了能够高效率地开发高性能的 Web 服务，开发人员开始采用微服务架构。微服务架构具有良好的隔离性，各服务相互独立。微服务架构中不同的服务可由不同的团队开发，当某个服务出现问题或者需要升级更新时，对其他服务影响较小，提高项目整体开发效率。微服务架构对各服务的性能与健壮性要求很高，当某个服务出现问题而导致系统发生故障时，该问题可以被运维人员快速发现，而某些服务只是性能低，无法引发系统崩溃，只是导致用户请求响应过慢或消耗硬件资源较多，上述问题难以被运维人员发现，从而降低了整个系统的服务质量。如何保证 Web 服务的安全性，以及如何快速定位 Web 服务中性能较低的系统方法，成了开发人员与运维人员亟待解决的问题。

本系统旨在提供 Web 服务用户访问序列异常诊断功能与 Web 服务中是否存在低性能方法的诊断功能。相对于传统只关注 CPU 指标与内存消耗等硬件信息的监控系统，本系统更关注 Web 服务中的业务数据。本系统通过 SkyWalking 与 Spring AOP 技术收集业务监控数据，分析 Web 服务中是否存在未进行权限控制的接口以及是否存在性能低下的系统方法。本系统前后端分别选用 Django 框架与 Vue.js 框架进行开发。本系统设计了用户访问序列异常诊断模块与系统

方法异常诊断模块等模块，用户访问序列异常诊断模块通过 PrefixSpan 算法从历史用户访问序列中挖掘频繁序列模式，利用 KMP 算法进行模式匹配，诊断当前用户是否存在异常访问行为。系统方法异常诊断模块使用 DBSCAN 算法对系统方法进行聚类，根据聚类结果分析存在异常的系统方法。

3.2 系统需求分析

本系统的涉众主要包括 Web 服务的运维人员与开发人员等。对于运维人员而言，可以通过本系统查看是否有用户存在异常访问行为，以及查看是否有系统方法存在性能问题，这样便可以及时对系统进行维护或及时调整部署环境的硬件资源，以保证系统能够平稳运行。开发人员可以通过本系统查看诊断信息，及时修复代码中存在的问题，提高服务质量，避免不必要的损失。

3.2.1 功能性需求分析

根据上述的涉众分析，可对本系统进行功能性需求分析，本系统具体的功能性需求如表3-1所示。

表 3-1: 系统功能需求列表

需求 ID	需求名称	需求描述
R1	启动监控	系统部署后，用户可以启动监控获取业务监控数据。
R2	停止监控	监控启动后，用户可以停止获取监控数据。
R3	修改监控配置信息	用户可修改监控配置信息，如业务监控数据存储地址等。
R4	选定诊断时间段	用户可以设置诊断的时间段，系统会处理诊断该时间段内的监控信息。
R5	设置用户正常访问序列	用户设置系统中正常用户的访问序列，供诊断阶段使用。
R6	查看单用户访问序列	用户可查看所监控系统中单个用户的访问序列。
R7	查看访问序列诊断结果	用户可以查看系统对指定时间段内是否有用户访问序列存在异常诊断结果。
R8	查看系统方法诊断结果	用户可以查看系统对指定时间段内是否有系统方法存在异常的诊断结果。

本系统主要包括用户访问序列诊断与展示以及系统方法信息诊断与展示等功能，用户需要自主选择需要诊断的时间段。运维人员与开发人员可以查看指定用户在指定时间段内对于所监控系统的访问序列，也可以查看当前所选择时间段内的用户访问序列异常诊断结果与系统方法异常诊断结果。用户访问序列异常诊断结果包括异常访问序列发生时间、异常访问序列所访问的接口以及产生此访问序列的用户 id 等信息，系统方法异常诊断结果包括方法所在项目程序文件、方法平均执行时间以及方法执行时使用的 CPU 等信息。

3.2.2 非功能性需求分析

本系统旨在提供 Web 服务用户访问序列异常诊断功能和 Web 服务中是否存在低性能系统方法的诊断功能，帮助运维人员与开发人员快速诊断出系统中低性能方法与未进行权限控制的接口，结合慕测平台的实际使用场景，所以本系统需要满足可用性、性能、可扩展性以及易用性等方面的要求，本系统需要满足的非功能性需求如下:

（1）系统的可用性。可用性是系统一切功能的前提。应当保证本系统能够在除断电外等意外之外的场景中提供稳定的服务，并且能在运行过程中输出日志等信息，以便发生故障时能够及时排查并恢复。

（2）系统的性能。系统应该是高性能的。在收集用户对所监控系统的访问信息以及所监控系统的运行信息时，不应该影响所监控系统的运行性能，同时在分析处理业务数据信息时应该保证较快的速度以及较低的硬件资源使用率，用户在请求系统时应保证响应时间在用户接受范围内。

（3）系统的可扩展性。系统应该是可扩展的。系统的各功能模块在更新时不应该影响系统其它模块的运行。系统设计应符合高内聚低耦合的设计思想，以便后续进行功能扩展。

（4）系统的易用性。系统应该是易于操作的。考虑到一个软件的运维与开发是由不同的团队负责的，而不同团队对于系统的理解可能存在差异，运维团队的成员可能对编码开发了解不足，因此系统应对降低使用门槛，直观简洁地将用户需要的信息展示出来，使得用户能够快速上手。

3.2.3 系统用例图

根据上文所述本系统的涉众分析、功能性需求分析以及非功能性需求分析，本系统的用例图如图3-1所示。本系统的用户群体为软件运维人员与软件开发人员，一共有停止监控、启动监控、修改监控配置信息、选定诊断时间段、查看单用户访问序列、设置用户正常访问序列、查看用户访问序列诊断结果以及查看系统方法诊断结果等八个系统用例。

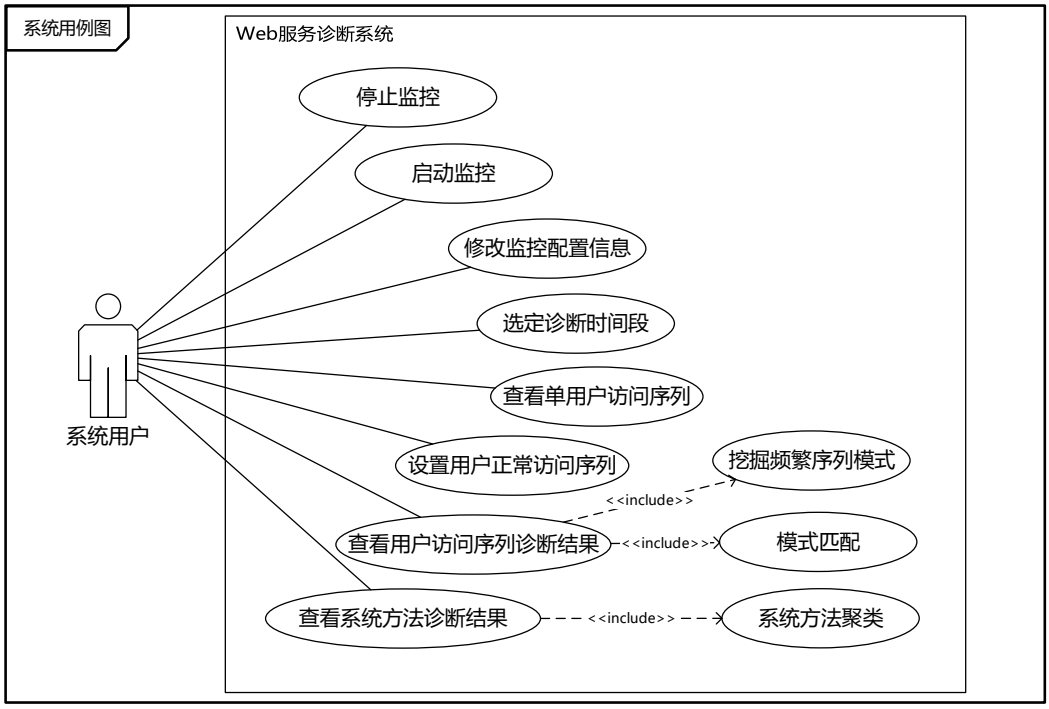

图 3-1: 系统用例图

3.2.4 系统用例描述

依据上文分析得到本系统的功能性需求，包括启动监控、停止监控、修改监控配置信息、选定诊断时间段、设置用户正常访问序列、查看单用户访问序列、查看用户访问序列诊断结果以及查看系统方法诊断结果等系统用例，下文将对上述系统用例进行详细阐述。

启动监控是指用户可以启动本系统获取监控数据的功能，与停止监控功能相配合可以实时启停本系统，避免浪费服务器资源，实现资源的合理利用。启动监控的详细用例描述如表3-2所示。

表 3-2: 启动监控用例描述表

ID	*UC*_1
用例名称	启动监控
用例参与者	系统用户 目的是启动监控，本系统开始收集监控信息
触发方式	用户打开管理页面并点击“启动监控”按钮
前置条件	本系统已经部署完成
后置条件	无
基本操作流	1. 用户成功进入本系统； 2. 用户打开系统管理页面； 3. 用户点击“启动监控”按钮。

停止监控是指用户可以关闭本系统获取监控数据的功能，与启动监控功能相配合可以实现实时启停本系统。停止监控后，直至下次启动监控期间的监控数据无法被本系统获取。停止监控的详细用例描述如表 3-3所示。

表 3-3: 停止监控用例描述表

ID	*UC*_2
用例名称	停止监控
用例参与者	运维人员或开发人员 目的是停止监控，本系统停止收集监控信息
触发方式	用户打开管理页面并点击“停止监控”按钮并确认
后置条件	无
前置条件	1. 本系统已经部署完成 2. 已启动监控，系统处于获取监控信息状态
基本操作流	1. 用户成功进入本系统； 2. 用户打开系统管理页面； 3. 用户点击“停止监控”按钮。

修改监控配置信息是指用户可以修改本系统中监控的配置信息。为了避免监控数据丢失，保证监控数据的完整性，因此本功能需要在停止监控后使用。修改监控配置信息的详细用例描述如表 3-4所示。

设定诊断时间段功能是指用户可以在本系统中自定义诊断监控数据的时间段，系统会根据用户所选时间段进行后续的用户访问序列诊断或系统方法诊断。如果用户选择的时间范围超过最大限制 MAX_TIME_PERIOD（默认为 24 小时）或当前时间段内没有监控数据则会提示用户输入的参数异常。设置监控时间段的详细用例描述如表 3-5所示。

表 3-4: 修改监控配置信息用例描述表

ID	*UC*_3
用例名称	修改监控配置信息
用例参与者	运维人员或开发人员 目的是修改监控配置信息，更改数据源
触发方式	用户填写配置信息后点击“提交”按钮
后置条件	无
前置条件	1. 本系统已经部署完成 2. 已停止监控
基本操作流	1. 用户成功进入本系统； 2. 用户打开系统管理页面； 3. 用户点击“修改监控配置”按钮； 4. 用户填写需要修改的配置项； 5. 用户点击“提交”按钮。

表 3-5: 设定诊断时间段用例描述表

ID	*UC*_4
用例名称	设定诊断时间段
用例参与者	运维人员或开发人员 目的是设置对所监控系统需要监控的时间段
触发条件	用户打开了相应页面并点击设置时间段组件
后置条件	无
基本操作流	1. 用户成功进入本系统； 2. 用户打开页面； 3. 用户点击时间组件选择设定时间段。

查看单用户访问序列是指用户在本系统中查看所监控系统中所有用户的访问序列的能力。本系统会获取所监控系统中用户的 userId 与 sessionId，系统用户可以根据所监控系统中的 userId 与 sessionId 查看指定用户的访问序列，访问序列中包括访问时间、访问接口以及本次访问消耗时间等信息。查看单用户访问序列的详细用例描述如表 3-6所示。

设置用户正常访问序列是指用户设置可以用来挖掘频繁序列模式的历史正常访问序列，用户可以对历史正常访问序列进行添加、删除、修改以及查询等操作。当系统对用户访问序列进行诊断时，系统会先查看数据库中是否已存在频繁序列模式，如果数据库中没有则会从正常访问序列中挖掘频繁序列模式。设置用户正常访问序列的详细用例描述如表 3-7所示。

表 3-6: 查看单用户访问序列用例描述表

ID	*UC_5*
用例名称	查看单用户访问序列
用例参与者	运维人员或开发人员 目的是查看指定用户在所监控系统中的访问序列
触发条件	用户打开了相应页面并选择 userId 与 sessionId
前置条件	系统已获取业务监控数据
后置条件	无
基本操作流	1. 用户成功进入本系统； 2. 用户打开查看单个用户访问序列的页面； 3. 用户根据 userId 选择用户； 4. 用户根据 sessionId 进一步选择需要查看的访问序列。

表 3-7: 设置用户正常访问序列用例描述表

ID	*UC_6*
用例名称	设置用户正常访问序列
用例参与者	运维人员或开发人员 目的是设置用户正常访问序列用例
触发条件	用户打开设置用户正常访问序列页面
前置条件	系统已获取业务监控数据
后置条件	无
基本操作流	1. 用户成功进入本系统； 2. 用户打开设置正常访问序列页面； 3. 用户点击设置正常访问序列按钮； 4. 用户输入筛选条件并点击“查找”按钮； 5. 用户勾选需要选择的用户访问序列； 6. 用户点击“确认修改”按钮。

查看用户访问序列诊断结果功能是系统向运维人员或开发人员展示用户访问序列诊断结果的能力，该功能是设置用户正常访问序列与设定诊断时间段的后续操作，如果系统中未设置用户正常访问序列则该功能会被禁用。如果系统中存在频繁序列模式，执行该功能时系统会根据之前挖掘的频繁序列模式自动进行模式匹配，展示匹配结果。如果本系统中没有已挖掘的频繁序列模式，则本系统会根据业务监控数据进行挖掘并持久化。设置完用户正常访问序列后，用户使用本功能时只需要设置诊断时间段，无需进行额外操作，提高系统的易用性。查看用户访问序列诊断结果的详细用例描述如表3-8所示。

表 3-8: 查看用户访问序列诊断结果用例描述表

ID	*UC_7*
用例名称	查看用户访问序列异常诊断结果
用例参与者	运维人员或开发人员 目的是查看指定时间段内是否有用户存在异常行为
触发条件	用户打开相应页面并设定诊断时间段
前置条件	1. 系统已获取业务监控数据 2. 用户已设置用户正常访问序列 3. 用户已选择诊断时间段
后置条件	无
基本操作流	1. 用户成功进入本系统； 2. 用户打开查看用户访问序列诊断结果页面； 3. 用户设定诊断时间段； 4. 用户点击“确认诊断”按钮。

查看系统方法诊断结果功能是系统向运维人员或开发人员展示系统方法分析结果的能力，执行该功能时会根据本系统中保存的聚类结果展示诊断信息，如果本系统中没有聚类结果，则会自动对所监控系统的方法根据执行时间等参数进行聚类，并将聚类结果保存到本系统中。查看系统方法诊断结果的详细用例描述如表3-9所示。

表 3-9: 查看系统方法诊断结果用例描述表

ID	*UC_8*
用例名称	查看系统方法异常诊断结果
用例参与者	运维人员或开发人员
触发条件	用户打开相应页面并设定诊断时间段
前置条件	1. 系统已获取业务监控数据 2. 用户已选择诊断时间段
后置条件	无
基本操作流	1. 用户成功进入本系统； 2. 用户打开查看系统方法诊断页面； 3. 用户设定诊断时间段； 4. 用户点击“确认诊断”按钮。

3.3 系统总体设计

3.3.1 系统整体架构设计

本系统是一个利用业务监控数据对 Web 服务进行异常诊断的系统，旨在为运维人员或开发人员提供 Web 服务用户访问序列异常诊断功能与 Web 服务中低性能系统方法诊断功能，本系统的架构图如图 3-2所示。本系统前端基于 Vue.js 框架开发，用于展示对所监控系统的诊断结果。本系统的服务端基于 Django 框架开发，主要包括业务监控数据收集模块、业务监控数据处理模块、用户访问序列异常诊断模块以及系统服务异常诊断模块共四个模块。本系统使用 Elasticsearch 与 MySQL 进行持久化业务监控数据与诊断分析结果等信息。从工程设计角度描述如下：

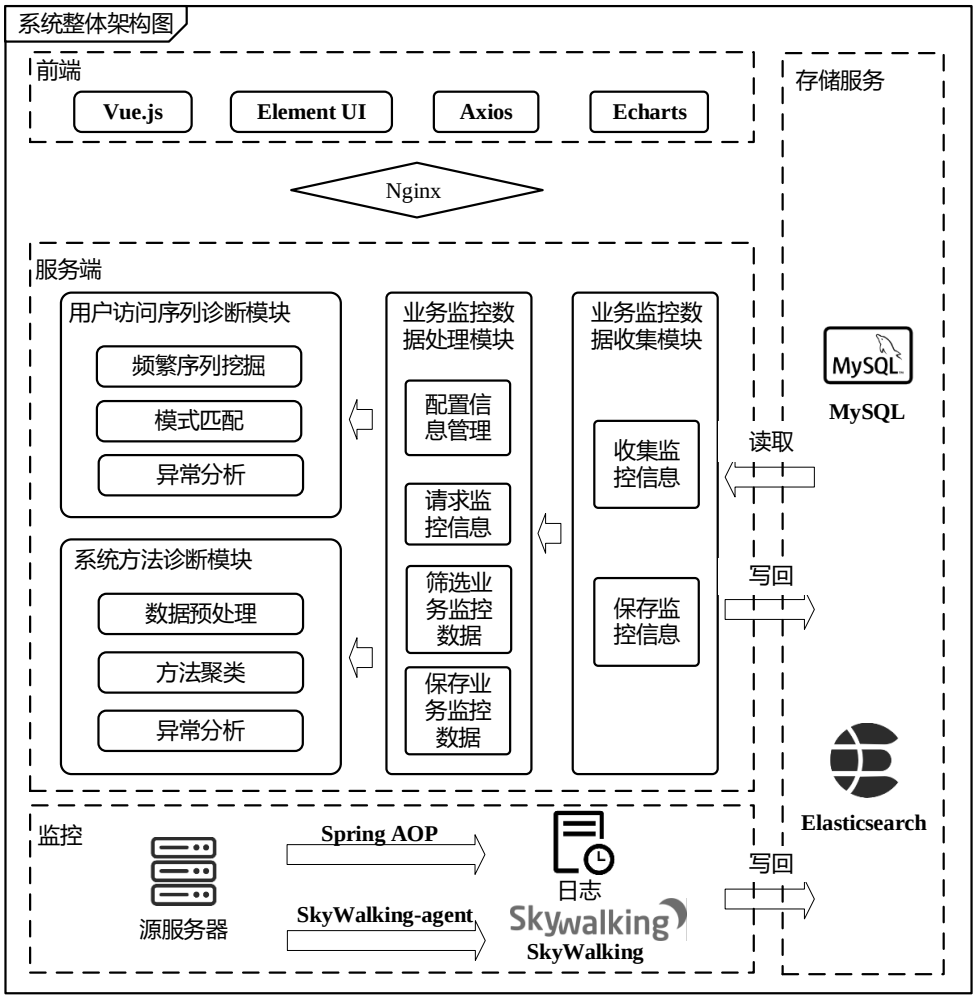

图 3-2: 系统整体架构图

系统前端基于 Vue.js 框架开发，Vue.js 是一个轻量级的用于构建 Web 界面的高性能开发框架，Vue.js 开发框架方便与第三方库整合使用，如 Element UI 与 Axios 等。Vue.js 采用 MVVM 架构，实现了双向数据绑定，即响应式数据绑定，当视图数据或变量其中一个发生改变时，另一个也会同步变化，因此开发人员能够将界面的设计与业务逻辑分离开，提高开发效率。系统前端与服务端使用 http 协议进行交互，基于 Axios 库开发。前端组件库采用 Element UI 与 Echarts。本系统属于监控系统，用户与系统交互较少，用户与系统的交互多为获取数据，因此本系统对于前端要求不高。Element UI 库能够提供丰富且方便使用的组件库；Echarts 能够提供丰富的图表库，借助 Echarts 可以直观地对数据进行展示；Axios 能够实现前端与服务端便捷地 http 交互。

系统后端主要使用 Django 框架开发。Django 开发框架与常用的 SpringBoot 开发框架类似，具有强大的社区和文档支持，学习成本低，开发人员能够快速地学习使用。Django 开发框架还原生地提供很多组件功能，包括 Session 管理、缓存管理以及权限认证等，因此基于 Django 框架能够快速搭建功能丰富的高性能 Web 服务。由于系统服务端需要处理分析大量业务监控数据，而 Python 类库对于数据分析与处理的支持比 Java 更为完善，因此选用基于 Python 开发的 Django 框架作为本系统的开发框架。

为了降低系统的部署成本与提高系统的部署效率，本系统使用 Docker 容器作为部署工具。Docker 容器是一个被广泛使用轻量级容器引擎，能够将项目代码、数据以及运行时所需的运行环境打包成可运行、方便传输的镜像，通过简易的启动命令即可将所打包的镜像部署在指定服务器上，是目前被广泛使用的主流容器之一。

3.3.2 系统模块划分

如图 3-2所示，本系统的服务端主要分为四个模块，分别是业务监控数据收集模块、业务监控数据处理模块、用户访问序列诊断模块以及系统方法诊断模块，下文将分别对各模块进行介绍。

业务监控数据收集模块的主要功能是将业务监控数据收集到服务端并进行持久化，通过 SkyWalking 与 Spring AOP 技术两种方式获取监控数据，SkyWalking 监控平台对所监控系统的侵入性低，但通过 SkyWalking 监控平台无法获取本系统需要的全部监控数据；通过 Spring AOP 技术可以自定义需要监控的数据项，但是对所监控系统的侵入性较高，为了降低对所监控系统的影响

且获取足够的系统监控数据，本系统采用 SkyWalking 与 Spring AOP 两种途径相结合的方式获取业务监控数据。本模块使用 Filebeat 转发业务监控数据，并使用 Elasticsearch 搜索引擎将收集的业务监控数据持久化到本地，供业务监控数据处理模块使用。

业务监控数据处理模块对业务监控数据收集模块收集的数据进行处理。业务监控数据收集模块收集的数据冗杂繁多且难以理解，无法直接使用，因此需要通过本模块对数据进行处理，从系统难以直接使用的数据中挖掘出方便使用的数据。处理后的数据易于理解，为了方便读取与查询，本模块使用 MySQL 对处理后的数据进行持久化。

用户访问序列诊断模块根据业务监控数据处理模块处理后的数据进行分析。用户访问序列诊断模块借助 PrefixSpan 算法从历史用户正常访问序列中挖掘频繁序列模式，并使用 KMP 算法将当前用户在所监控系统中的访问序列与频繁序列模式及其子串进行模式匹配。最后系统根据匹配结果分析当前用户访问序列是否存在异常。

系统方法诊断模块也是根据业务监控数据处理模块处理后的数据进行分析。系统方法诊断子模块借助 DBSCAN 聚类算法对所监控系统的方法进行聚类，并根据聚类结果与历史诊断结果分析得出性能低下的系统方法。

系统采用 http 协议实现前端与后端之间的交互，系统后端接收并解析前端的请求。系统可以根据解析后的请求调用异常诊断模块与业务数据收集模块等其他模块，也可以根据业务逻辑对数据库进行增删改查等操作，并将处理后的结果反馈给前端进行展示。在之后的更新中可以接入邮件系统或短信系统，可以将系统发现的异常及时通过邮件或短信反馈给用户。

3.3.3 4+1 视图

本文参照 Philippe Kruchten 教授论文《The 4+1 View Model of Architecture》中首次提出的“4+1”视图方法给出本系统的架构设计。如图 3-3所示，“4+1”视图是从五个不同视角描述软件架构视图，分别是逻辑视图、开发视图、进程视图、物理视图以及场景视图。由于描述软件架构时需要分析的内容复杂繁多，难以直接对整个软件架构进行描述，因此需要采用“4+1”视图进行描述。“4+1”视图中每种视图都分别只从各自的角度描述软件架构，描述只涵盖软件的特定方面，忽略软件的其他方面。在上一节的用例分析中已从场景视图对系统进行了详细的描述，故本节中只从其他四个视图的角度对系统软件架构进行描述。

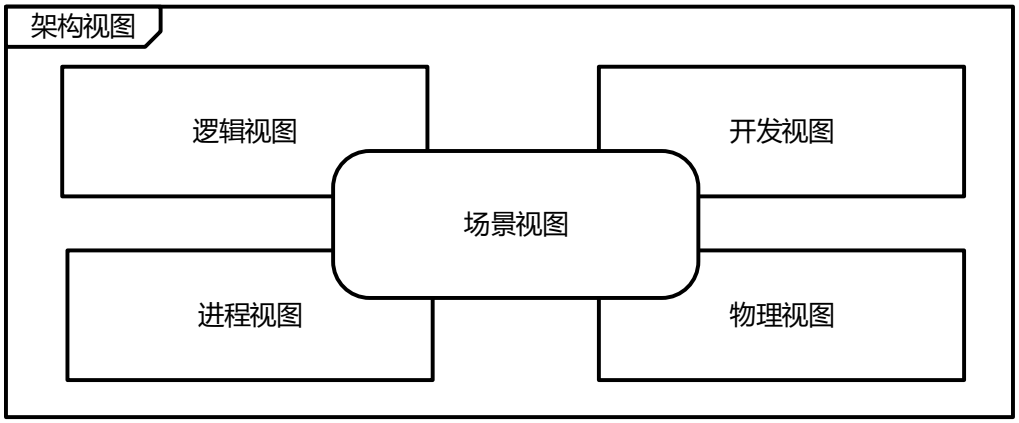

图 3-3: 系统架构视图

（1）逻辑视图

逻辑视图从用户的角度描述系统架构，主要描述系统的功能需求，即系统为最终用户提供了哪些功能与服务。本系统采用面向对象的设计思想，因此可通过 UML 类图来展示逻辑视图。业务逻辑划分如图 3-4所示。

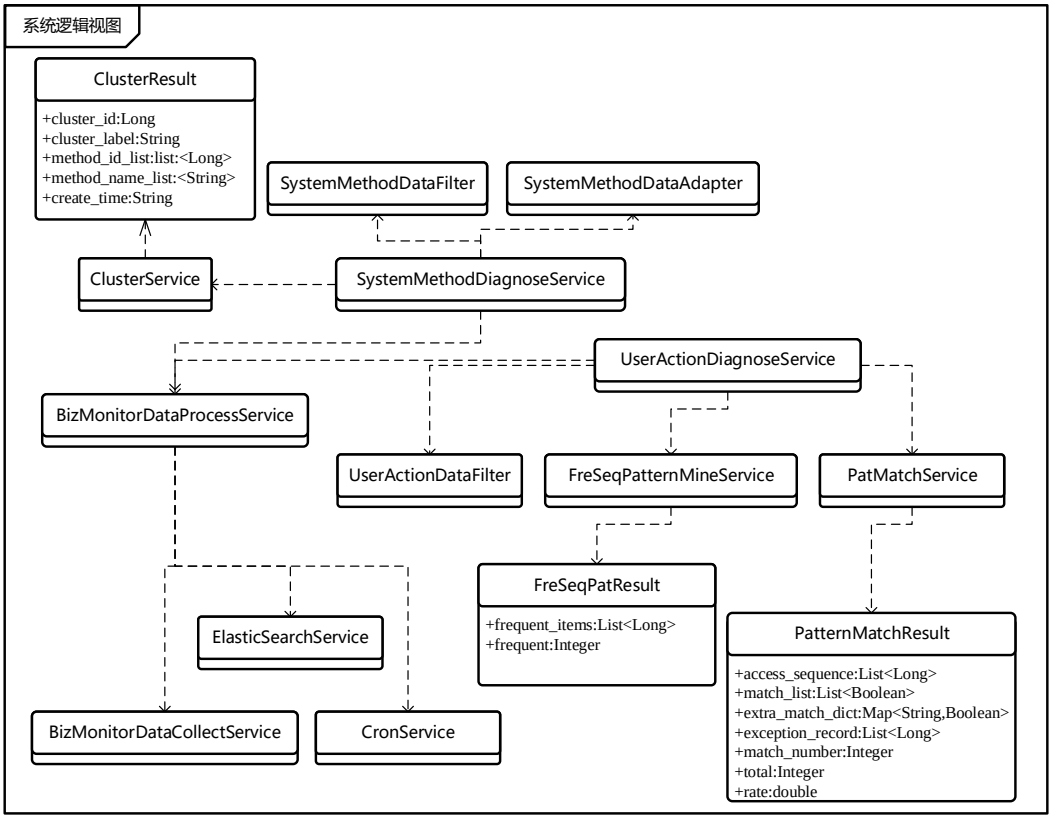

图 3-4: 系统逻辑视图

SystemMethodDiagnoseService 与 UserActionDiagnoseService 是本系统的核心服务，主要功能为诊断系统方法是否存在异常以及诊断用户访问序列是否存在异常。SystemMethodDataFilter 负责过滤掉业务监控数据中运行信息不完整的记录。SystemMethodDataAdapter 负责对系统方法信息进行转换，对运行时信息进行加权处理。ClusterService 负责对处理后的系统方法进行聚类。ClusterResult 为聚类结果的抽象类，包括聚类唯一标识、聚类标签以及该类的方法 id 等成员变量。UserActionDataFilter 负责过滤掉用户访问序列中 heartbeat 等无效访问记录。FreSeqPatternMineService 负责从用户正常访问序列中挖掘频繁序列模式。PatMatchService 负责对字符串或序列进行模式匹配。FreSeqPatResult 为频繁序列模式挖掘结果的抽象类，包括频繁序列模式以及频繁序列模式出现频数等成员变量。PatternMatchResult 为模式匹配结果的抽象类，包括各访问序列的匹配情况、原访问序列、异常访问记录、匹配数量以及匹配比例等成员变量。BizMonitorDataProcessService 负责处理业务监控数据，包括获取数据、出来数据以及保存数据等。BizMonitorDataCollectService 负责收集业务监控数据。ElasticsearchService 负责从 Elasticsearch 搜索引擎中查询系统需要的业务监控数据。CronService 为系统定时服务，负责定时周期性调用 BizMonitorDataCollectService 中的方法，获取业务监控数据。

（2）开发视图

开发视图是指从开发者的角度描述系统软件架构，主要描述系统中软件模块的组织与管理，即系统具体实现的静态组织结构。当通过开发视图描述系统软件架构时，需要考虑软件内部设计，如软件代码设计的通用性与简洁性、软件是否存在局限性以及是否充分解耦等。本系统的开发视图如图 3-5所示。

本系统的开发视图采用层次结构对系统架构进行描述，共分为三层，分别是 UI、Service Logic 以及 Tech Service。

UI 是展示层，本系统的展示层由 Vue 文件、HTML 文件、CSS 文件、JavaScript 文件以及用于配置或传输的 JSON 文件组成。

Service Logic 是服务逻辑层，本系统的服务逻辑层采用 MVC 架构，其中 Controller 包负责对 http 请求进行接收并解析，根据解析之后的请求调用 Service 包；Request 包是前端发送请求的对象实体；DTO 包是响应前端请求后返回给前端的对象实体；Service 包负责收集业务监控数据、处理业务监控数据以及分析业务监控数据等具体的业务逻辑，为 Controller 包服务；Common 包提供通用方法与系统变量，为 Service 包服务；Adapter 包提供数据格式化与数据过滤

等功能，为 Controller 包与 Service 包服务；Model 包为系统对象实体，与数据库中的表相互对应；Dao 包是利用 Model 对底层数据库进行增、删、改、查等操作的封装，与 Model 一起为 Service 包服务。

Tech Service 是指本系统中使用到的第三方库，主要有容器技术 Docker、前端组件库 ElementUI、搜索引擎 Elasticsearch、数据库 MySQL 以及日志数据转发工具 Filebeat。

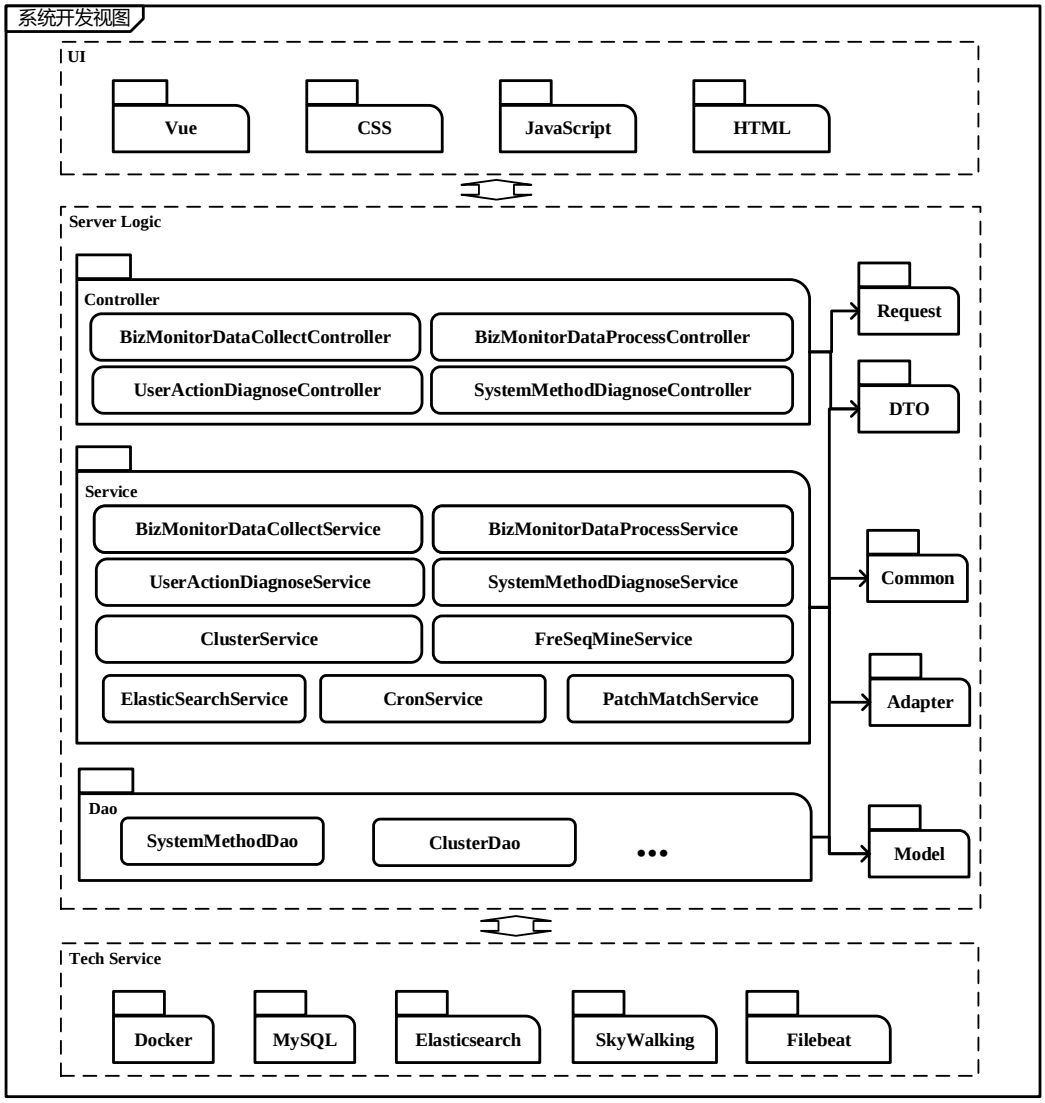

图 3-5: 系统开发视图

（3）进程视图

进程视图从系统集成人员的角度描述系统架构，主要描述系统的运行特性，即程序运行时性能与可用性等非功能性需求。当通过进程视图描述软件架构时，需要描述系统在集成时相关进程或线程之间的关系以及系统的动态运行

信息，动态运行信息是指上述逻辑视图中提出的各类何时在系统中运行。本系统的开发视图如图3-6所示。

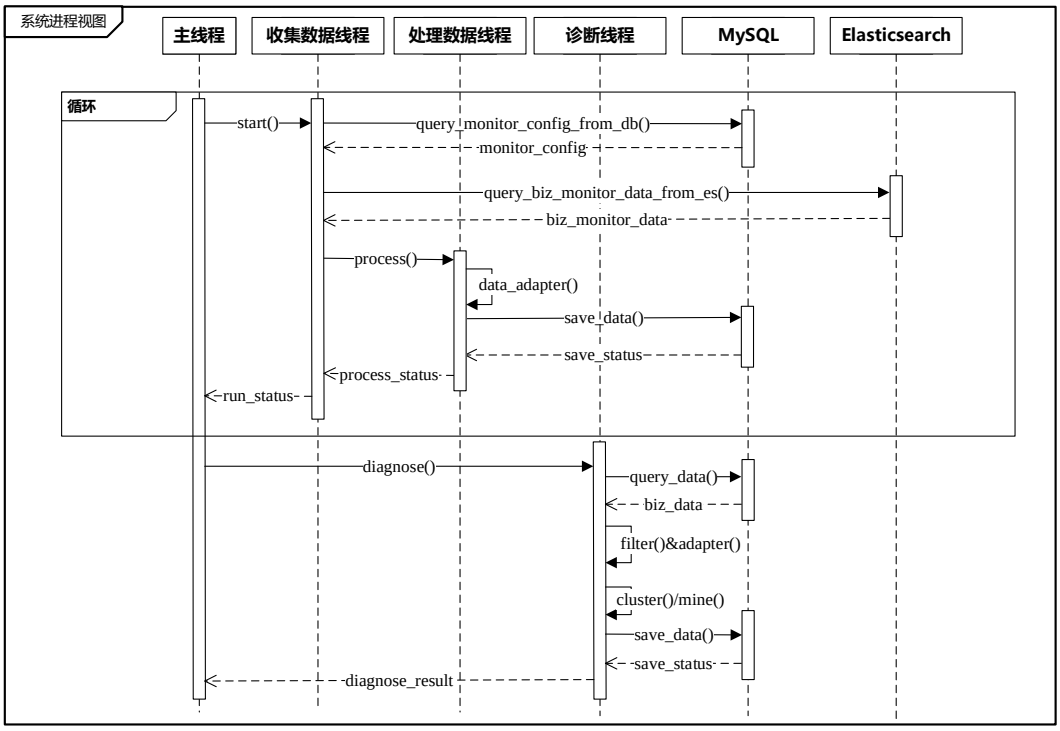

图 3-6: 系统进程视图

系统未收到请求时运行在主线程。每隔一段时间系统的收集数据线程会被定时激活运行，从 Elasticsearch 服务中请求业务监控数据，并将获取的数据保存到 MySQL 数据库中。当主线程收到用户查看系统诊断信息时，系统首先查看数据库中是否已存在所选时间段的诊断记录，如果诊断信息已存在则直接返回；否则会调用诊断服务重新获取诊断信息并将诊断结果保存到数据库中，再将诊断结果返回给用户。当系统收到用户查看单用户访问序列的请求时，系统会根据查询条件返回指定信息。

（4）物理视图

物理视图从运维人员的角度描述系统架构，主要描述软件部署软件到硬件的对应关系。由于程序落地时需要运行在具体的服务器中，所以通过物理视图描述软件架构时，需要考虑系统在实际运行场景中的运行需求与状态。本系统的物理视图如图3-7所示。

用户通过客户端的浏览器访问指定 URL 即可本系统。本系统通过反向代理服务 Nginx 对用户请求进行负载均衡，将请求转发给 Django 服务端进行处

理，Django 服务端使用 http 请求从部署 Elasticsearch 的服务器上请求业务监控数据，以及通过 tcp 的方式访问 MySQL 数据库。此外还需要将 SkyWalking 的组件 SkyWalking-agent 与 Filebeat 部署在与所监控的系统相同的服务器上，SkyWalking-agent 收集业务监控数据转发到 SkyWalking-collector 中，Filebeat 收集日志信息转发到 Elasticsearch 中。本系统中所有服务均使用 Docker 容器部署运行，能够提高部署效率以及降低部署成本。

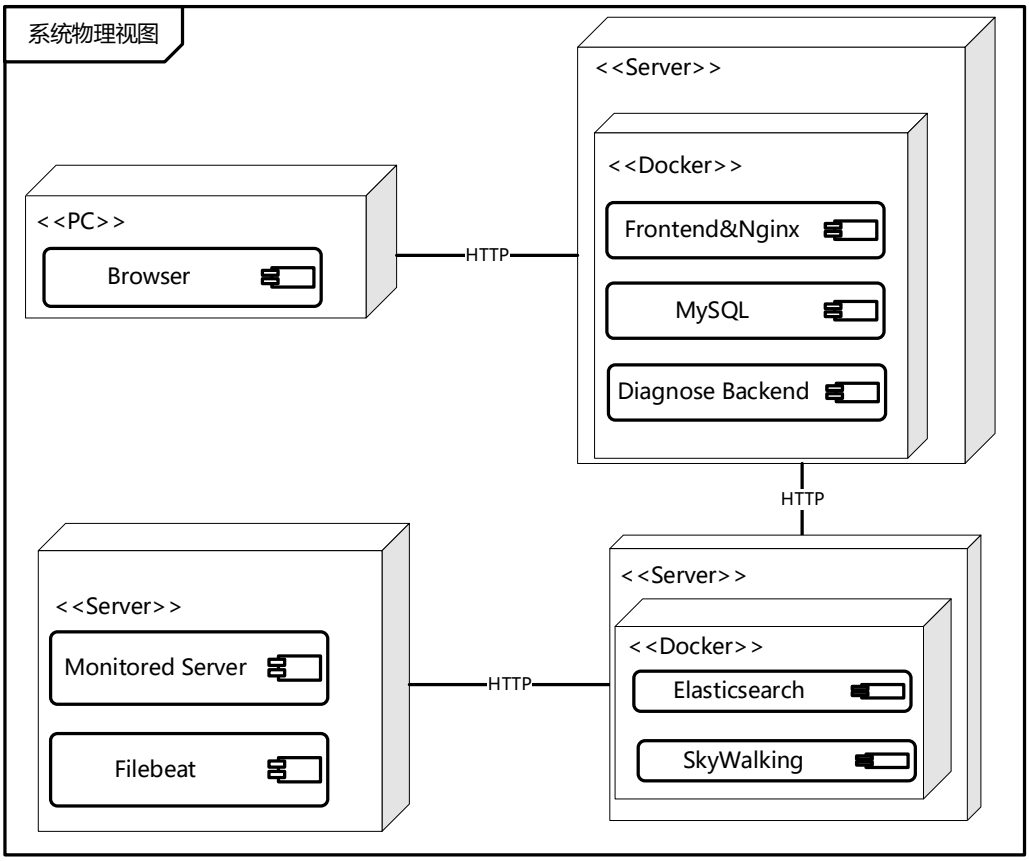

图 3-7: 系统物理视图

3.3.4 业务监控数据收集模块流程设计

业务监控数据收集模块的主要功能是获取业务监控数据与转发并持久化业务监控数据，流程如图3-8所示。本模块的主要步骤如下：

（1）部署并配置 SkyWalking-agent 与 SkyWalking-collector，启动需要监控的 Web 服务，使 Web 服务处于运行状态，使用 SkyWalking-agent 监控并转发业务监控信息，SkyWalking-collector 接收业务监控数据。

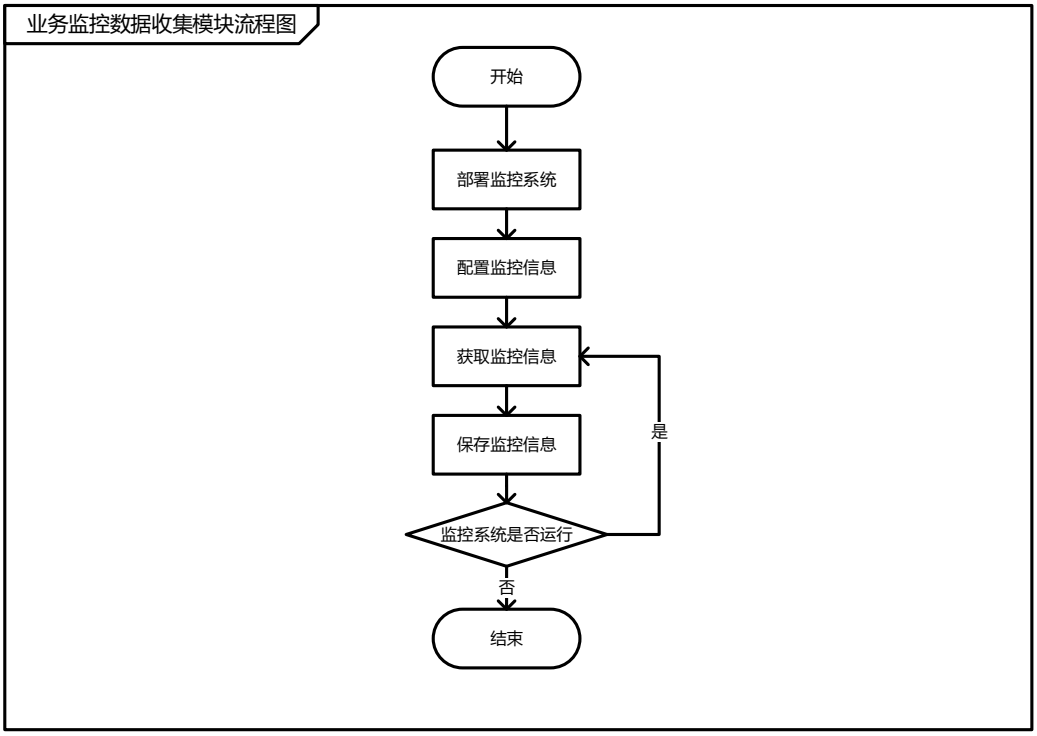

图 3-8: 业务监控数据收集模块流程图

（2）借助 Spring AOP 生成系统运行的日志信息，通过 Filebeat 工具转发指定日志信息。

（3）持久化业务监控数据，将系统监控数据和日志信息保存到 Elasticsearch 搜索引擎中，以便业务数据处理模块的使用。

3.3.5 业务监控数据处理模块流程设计

业务监控数据处理模块的主要功能是对业务监控数据收集模块获取的数据进行预处理，并将处理后的数据保存到 MySQL 数据库中。由于上一步收集的监控数据方式不同，收集的数据格式存在差异，每条数据包含的信息量超过本系统所需，因此需要数据预处理对复杂冗长的业务监控数据进行筛选。业务监控数据处理模块对业务监控数据进行筛选以及格式转换等处理，转化为能被关系型数据库保存的数据格式。业务监控数据处理模块的流程如图 3-9所示。本模块的主要步骤如下:

（1）通过 http 请求根据指定查询条件在 Elasticsearch 搜索引擎中分别获取指定时间段内的业务监控数据与日志信息。

（2）从日志信息中提取业务监控数据，按照时间序列与 Web 服务中的用户 id 对业务监控数据进行划分。

（3）对业务监控数据进行筛选处理，如删除 heartbeat 等无效请求、归一化系统运行指标数据等操作、过滤监控信息中无需使用的属性等。

（4）持久化处理结果，系统将处理后的业务监控数据按照时间序列保存到 MySQL 数据库中，以便用户访问序列异常诊断模块与系统方法异常诊断模块的使用。

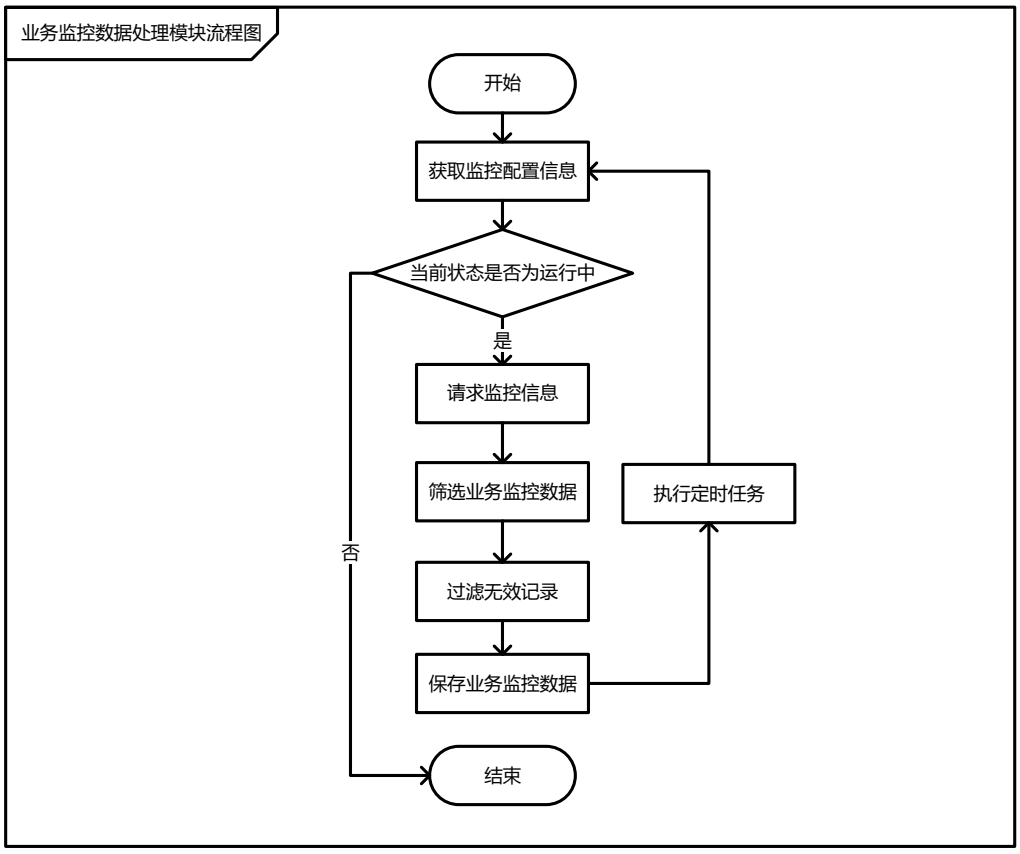

图 3-9: 业务监控数据处理模块流程图

3.3.6 用户访问序列异常诊断模块设计

用户访问序列异常诊断模块的主要功能是分析用户在所监控系统中的业务请求序列是否存在异常。用户访问序列异常诊断模块流程如图 3-10所示。本模块的主要步骤如下：

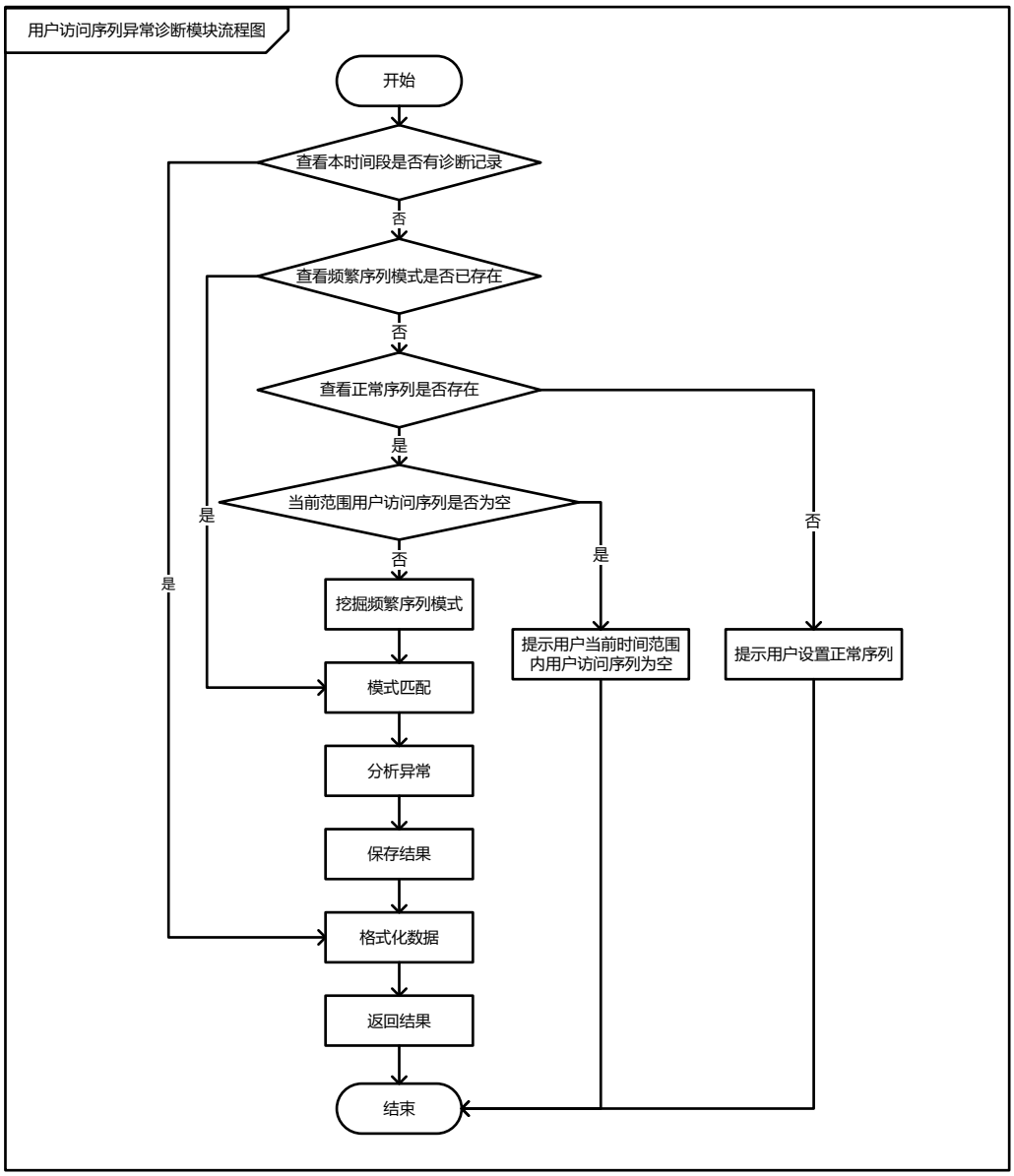

图 3-10: 用户访问序列异常诊断模块流程图

（1）判断当前时间段是否已存在诊断结果，如果不存在诊断记录则转步骤（2），否则直接将诊断结果格式化后返回。

（2）判断本次诊断是否需要重新挖掘频繁序列模式，如需要重新挖掘频繁序列模式则转步骤（3），否则从数据库获取频繁序列模式并转步骤（6）。

（3）判断是否系统是否需要重新设置用户正常访问序列数据，如果需要重新设置，则保存用户输入的正常访问序列数据，否则转步骤（4）。

（4）从 MySQL 数据库中查取用户的正常访问序列信息。

（5）使用关联规则挖掘算法从用户的正常访问序列信息中挖掘频繁序列模式，并将挖掘得到的频繁序列模式保存到 MySQL 数据库中。

（6）将待分析的访问需要与频繁序列模式及其子串进行匹配，根据频繁序列模式的频数进行加权处理，根据匹配结果诊断行为序列是否存在异常。

（7）将步骤（6）中的诊断结果格式化后返回给用户，并将诊断结果保存到 MySQL 数据库中。

3.3.7 系统方法异常诊断模块流程设计

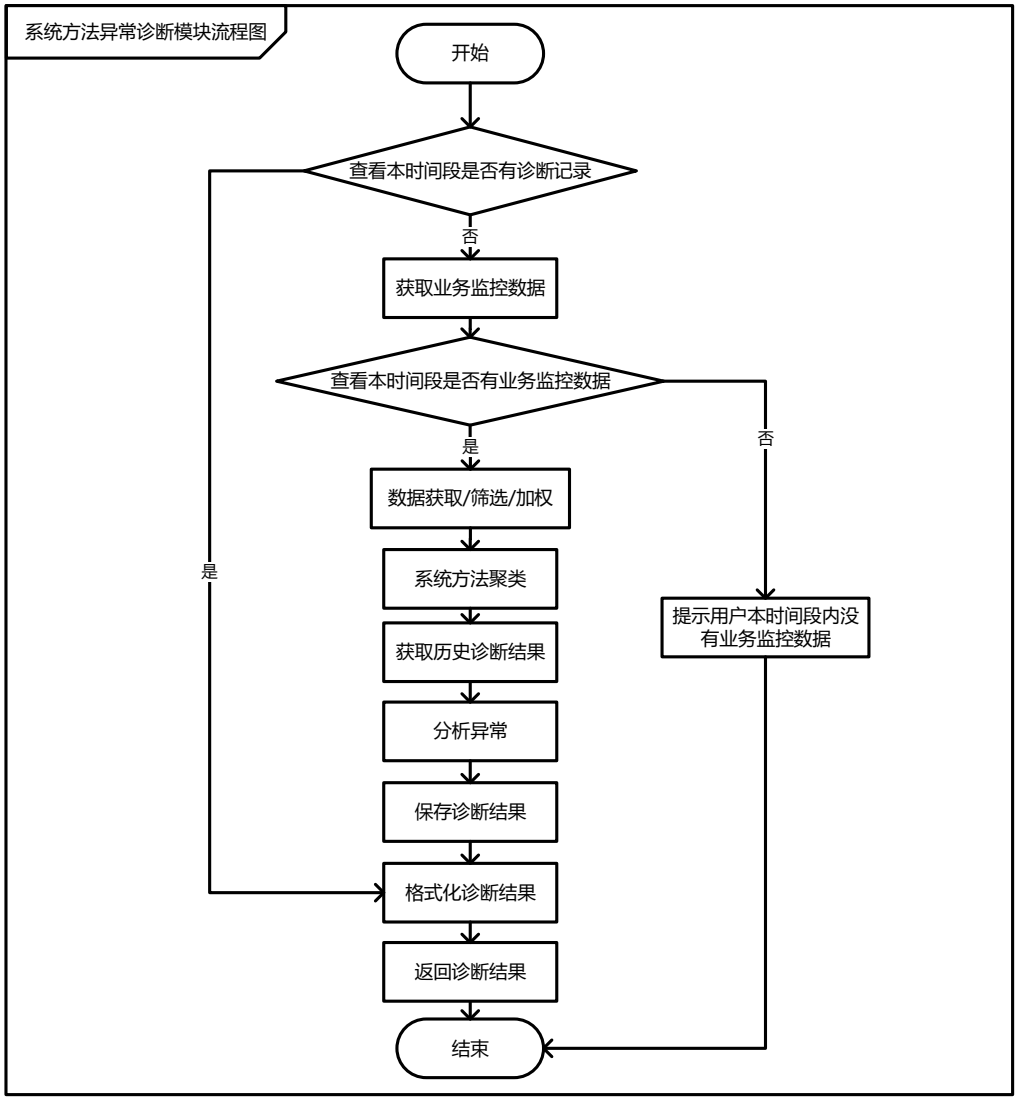

图 3-11: 系统方法异常诊断模块流程图

系统方法异常诊断模块的主要功能是监控系统在运行时用户业务请求中各方法是否存在异常。系统方法异常诊断模块的流程如图3-11所示。本模块的主要步骤如下：

（1）判断指定时间段是否已经有诊断结果存在，如果已有则将诊断结果格式化并返回，否则转步骤（2）。

（2）从 MySQL 中获取指定时间范围内的业务监控数据，如果指定时间范围内没有业务监控数据则提示用户当前时间范围内没有业务监控数据，如果指定时间范围内有业务监控数据则转步骤（3）。

（3）对业务监控数据进行预处理，过滤运行数据不完整的方法记录。从 MySQL 中获取用户对于 CPU 以及内存等硬件信息的权重，并将权重添加到系统方法的运行数据中。

（4）使用聚类算法对系统方法进行聚类，分别计算各类别中 CPU 使用率与内存使用率等信息，并将聚类结果保存到 MySQL 数据库中。

（5）从 MySQL 中查询历史系统方法异常诊断结果作为参考，根据聚类结果对系统方法进行诊断。

（6）将步骤（5）中分析得出的诊断结果返回给用户，并将诊断结果保存到 MySQL 数据库中。

3.4 数据模型设计

本系统的服务端采用关系型数据库 MySQL 实现数据的持久化，存储处理后的业务监控数据、分析监控数据获得的中间结果以及诊断结果等信息。由于处理后的业务监控数据与诊断结果均为结构化数据，因此在本系统中选用关系型数据库 MySQL 作为持久化工具。本系统中有若干抽象实体，包括 Spring AOP 与 SkyWalking 获取的业务监控信息、监控配置信息、系统方法信息以及聚类结果信息、诊断结果等。

实体关系图描述了系统中各抽象实体的属性以及抽象实体之间的联系。本系统的实体关系图如图3-12所示。图中列出了系统中部分关键抽象实体的属性及关系，其中表 system_method 保存从业务监控数据中筛选出的系统方法运行信息，主要为所监控系统中 service 层方法的运行信息；表 access_sequence 保存从业务监控数据中筛选出的用户访问序列信息；cluster_result 为聚类结果表，保存系统方法聚类后的结果数据；normal_access_sequence 为用户正

常访问序列，从用户访问序列表中选取，用来挖掘频繁序列模式；prefixspan_frequent_items 为频繁序列模式表，用来保存挖掘得到的频繁序列模式；diagnose_record 为诊断记录表，保存诊断时间、诊断类型以及诊断状态等信息；access_sequence_diagnose_result 与 system_method_diagnose_result 分别为用户访问序列诊断结果表与系统方法诊断结果表。除图3-12展示的实体关系以外，系统中还有配置信息表、Spring AOP 日志信息表以及 SkyWalking 信息表等非关键抽象实体。

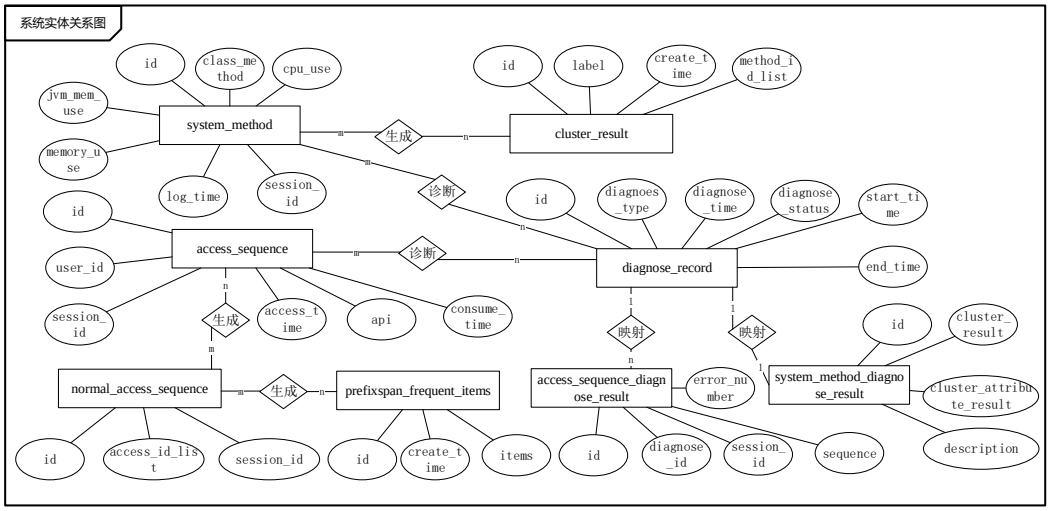

图 3-12: 系统实体关系图

配置信息表对应系统从 Elasticsearch 搜索引擎中获取数据的配置信息对象，该对象包括索引唯一标识 beat_index、索引名称 beat_name、最近一次获取数据是否成功标识 last_time_success 等。其详细属性设计如表3-10所示。

表 3-10: 配置信息表

字段	类型	含义
id	int	记录唯一标识
beat_name	varchar	Elasticsearch 索引名称
beat_index	varchar	Elasticsearch 索引唯一标识
from_time	varchar	查询 Elasticsearch 获取数据时间段起点
to_time	varchar	查询 Elasticsearch 获取数据时间段终点
last_time_success	tinyint	最近一次查询 Elasticsearch 是否成功
url	varchar	获取监控信息的 url
is_deleted	tinyint	该记录是否被删除
is_started	tinyint	该配置是否已生效

SkyWalking 日志信息表对应 SkyWalking 监控平台收集的业务监控数据对象。本系统对 SkyWalking 监控平台获取的业务监控数据进行处理并持久化，包括请求端点名称 end_point_name、请求是否出错 is_error、请求时间 start 以及接口持续时间 duration 等。其属性设计如表 3-11所示。

表 3-11: SkyWalking 日志信息表

字段	类型	含义
id	int	记录的唯一标识
end_point_name	varchar	请求端点名称，url 等
is_error	tinyint	本次请求是否出错
start	varchar	本次请求歧视时间
duration	varchar	本次请求持续时间，单位为 ms
trace_id	varchar	请求唯一标识

Spring AOP 日志信息表对应通过 Spring AOP 收集的业务监控数据对象。系统通过 Spring AOP 技术从所监测系统收集日志，系统对日志进行收集处理，提取其中业务监控数据对象并持久化，该对象包括用户唯一标识 user_id、用户唯一会话标识 session_id 以及访问接口 url 等字段。其属性设计如表 3-12所示。

表 3-12: Spring AOP 日志信息表

字段	类型	含义
id	int	记录唯一标识
user_id	varchar	用户唯一标识
session_id	varchar	用户的会话唯一标识
log_time	varcahr	本条业务监控数据记录时间
type	varchar	本条业务监控数据类型，request/response
url	varchar	本条业务监控对应的请求 url
http_method	varchar	http 请求类型，GET、POST 等
class_method	varchar	本条业务监控数对应的运行方法
request_ip	varchar	本条业务监控数据对应的请求源 ip
request_args	varchar	本条业务监控数据对应的请求参数
consume_time	varchar	本条业务监控数据对应的处理时间
cpu_user_use	varchar	本条业务监控数据所在服务器 CPU 使用率
jvm_mem_total	varchar	本条业务监控数据运行时 JVM 内存总量
jvm_mem_use	varchar	本条业务监控数据运行时 JVM 内存使用数量
memory_total	varchar	本条业务监控数据运行时服务器内存总量
memory_use	varchar	本条业务监控数据运行时服务器内存使用量
response_args	varchar	本条业务监控数据返回参数

用户请求信息表对应用户对所监控系统的访问信息，包括用户访问系统时请求的接口 url、访问消耗时间 consume_time、访问次数 total_time 以及该记录是否有效 is_deleted 等字段。其属性设计如表 3-13所示。

表 3-13: 用户请求信息表

字段	类型	含义
id	int	记录的唯一标识
url	varchar	请求的 url
consume_time	varchar	本次请求花费时间
total_time	int	请求总数
is_deleted	tinyint	本次记录是否被删除

系统方法信息表对应所监控系统中各方法运行时信息，包括系统方法名称 class_method、运行消耗时间 consume_time 以及该记录是否被逻辑删除 is_deleted 等。其属性设计如表 3-14所示。

表 3-14: 系统方法信息表

字段	类型	含义
id	int	记录的唯一标识
class_method	varchar	系统方法名称
consume_time	varchar	系统方法运行花费时间
jvm_mem_total	varchar	jvm 内存大小
jvm_mem_use	varchar	系统方法运行使用 jvm 内存大小
memory_use	varchar	系统方法运行使用内存大小
memory_total	varchar	内存总大小
cpu_use	varchar	系统方法运行使用 CPU
user_id	varchar	用户唯一标识
session_id	varchar	会话唯一标识
is_deleted	tinyint	本条记录是否被删除

系统方法信息权重表对应使用聚类算法对系统方法进行聚类时，系统方法各属性占的权重，包括唯一标识 id、记录创建时间 create_time、记录修改时间 modify_time、系统方法属性名称 system_method_attribute_name、系统方法属性权重 system_method_attribute_weight 以及该记录是否被删除的标记 is_deleted 等。其属性设计如表 3-15所示。

表 3-15: 系统方法信息权重表

字段	类型	含义
id	int	记录的唯一标识
create_time	varchar	本记录创建时间
modify_time	varchar	本记录最近修改时间
system_method_attribute_name	varchar	系统方法属性名称
system_method_attribute_weight	double	系统方法属性权重
is_deleted	tinyint	本条记录是否被删除

频繁序列模式信息表对应系统中用到的频繁序列模式等信息，本系统通过频繁序列挖掘算法从处理后的业务监控数据中挖掘获取的频繁序列模式，包括记录唯一标识 id、记录创建时间 create_time 以及频繁序列模式 frequent_items。其属性设计如表 3-16所示。

表 3-16: 频繁序列模式信息表

字段	类型	含义
id	int	记录的唯一标识
create_time	varchar	本次频繁序列模式挖掘时间
frequent_items	text	频繁序列模式集

用户访问序列记录表保存用户对于所监控系统的访问行为，包括用户唯一标识 user_id、会话唯一标识 session_id、访问时间 access_time、访问接口 api 等属性。其属性设计如表 3-17所示。

表 3-17: 用户访问序列记录表

字段	类型	含义
id	int	记录的唯一标识
user_id	varchar	用户唯一标识
session_id	varchar	会话唯一标识
access_time	varchar	访问时间
api	varchar	访问接口
consume_time	varchar	接口响应时间

用户正常访问序列表可以用来挖掘频繁序列模式，需要手动配置，包括会话 id、访问序列信息 method_id_list 等信息。其属性设计如表 3-18所示。

表 3-18: 用户正常访问序列表

字段	类型	含义
id	int	记录的唯一标识
session_id	varchar	本次频繁序列模式挖掘时间
method_id_list	text	频繁序列模式集

系统方法聚类结果信息表对应系统调用聚类服务产生的聚类结果，包括聚类结果生成时间 create_time、聚类唯一标识 cluster_id、聚类结果标签 cluster_label 以及方法 id 列表 method_id_list 等信息。其具体属性设计如表3-19所示。

表 3-19: 聚类结果信息表

字段	类型	含义
id	int	记录的唯一标识
create_time	varchar	聚类结果创建时间
db_scan_id	varchar	聚类唯一标识
cluster_label	varchar	聚类结果标签
method_id_list	text	系统方法聚类结果

系统方法诊断结果表保存系统方法的诊断结果，包括诊断记录唯一标识 diagnose_id、系统方法 id 列表 origin_method_id_list、聚类结果类别列表 cluster_label_list、处理后的聚类结果信息 cluster_result、聚类结果描述 description 等信息。其具体属性设计如表3-20所示。

表 3-20: 系统方法诊断结果表

字段	类型	含义
id	int	记录的唯一标识
diagnose_id	int	诊断记录唯一标识
origin_method_id_list	text	系统方法 id 列表
cluster_label_list	text	聚类结果类别列表
cluster_result	text	处理后的聚类结果信息
cluster_attribute_detail	text	聚类后属性详情
description	text	诊断结果描述
start_time	varchar	诊断数据起始时间
end_time	varchar	诊断数据终止时间

用户访问序列诊断结果表记录用户访问序列的诊断结果，包括诊断记录唯一标识 diagnose_id、用户唯一标识 user_id、会话唯一标识 session_id、访问序列详情 sequence、异常系数 error_number、诊断数据起始时间 start_time、诊断数据终止时间 end_time 等信息。其具体属性设计如表 3-21所示。

表 3-21: 用户访问序列诊断结果表

字段	类型	含义
id	int	记录的唯一标识
diagnose_id	int	诊断记录唯一标识
user_id	varchar	用户唯一标识
session_id	varchar	会话唯一标识
sequence	text	序列详情
error_number	double	异常系数
start_time	varchar	诊断数据起始时间
end_time	varchar	诊断数据终止时间

诊断记录表保存系统的诊断记录，使用诊断记录表可以在用户重复诊断时，直接返回数据库中已有的诊断信息。诊断记录表包括诊断类型 diagnose_type、诊断状态 diagnose_status、诊断记录时间 diagnose_time、诊断数据起始时间 start_time、诊断数据终止时间 end_time 等信息。其具体属性设计如表 3-22所示。

表 3-22: 诊断记录表

字段	类型	含义
id	int	记录的唯一标识
diagnose_type	varchar	诊断类型
diagnose_status	varchar	诊断状态
diagnose_time	varchar	诊断时间
start_time	varchar	诊断数据起始时间
end_time	varchar	诊断数据终止时间

3.5 本章小结

本章主要对基于业务监控数据的 Web 服务诊断系统进行需求分析与总体设计。首先对本系统进行了整体概述，通过简单的涉众分析得出本系统的主要用户群体为 Web 服务运维人员与开发人员，并对本系统进行功能性需求分析与非功能性需求分析。其次根据用户实际场景给出系统用例并通过表格的方式对各系统用例进行详细描述。接下来对本系统进行概要设计与功能模块设计，并利用“4+1”视图从不同的视角对本系统架构进行了多方面的介绍。之后给出系统中各模块的流程设计。最后介绍了本系统使用的持久化工具与本系统中数据模型的字段描述。

第四章　系统详细设计与实现

4.1　用户访问序列异常诊断服务详细设计与实现

用户访问序列异常诊断模块负责诊断一组用户访问序列是否存在异常。本模块的挖掘阶段是指通过挖掘算法从正常访问序列中提取频繁序列模式。分析阶段是指使用待分析的访问序列与频繁序列模式进行匹配，并根据匹配结果进行分析。保存阶段将诊断结果保存到 MySQL 中。

4.1.1　用户访问序列异常诊断服务架构图

图 4-1为用户访问序列异常诊断服务的架构图。前端通过 http 请求访问服务端，服务端进行异常诊断后返回诊断结果，并将结果保存到 MySQL 中。

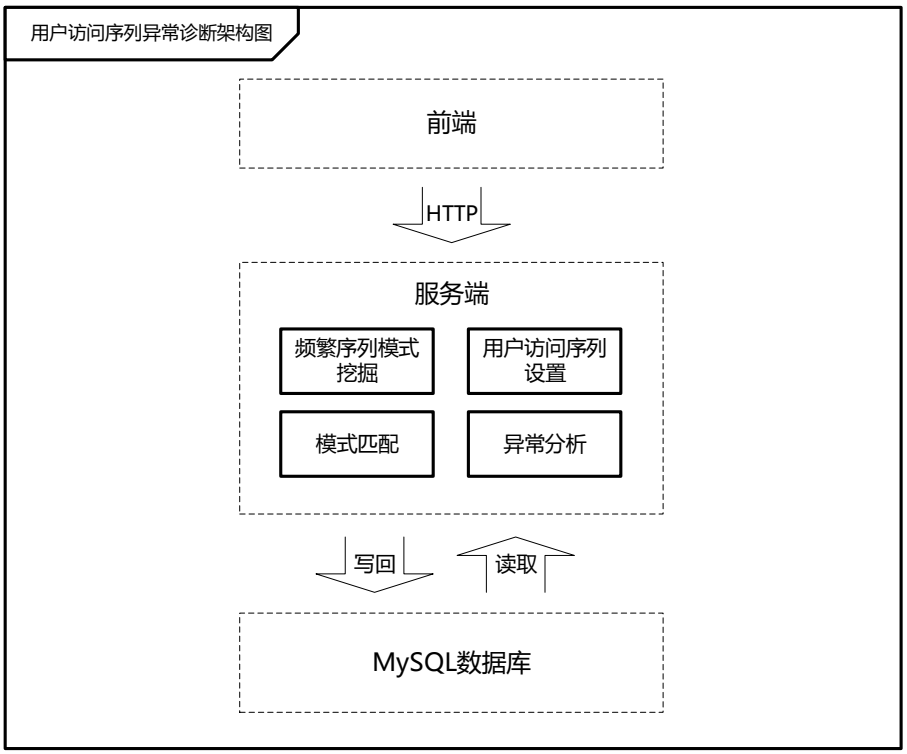

图 4-1: 用户访问序列异常诊断模块架构图

4.1.2 用户访问序列异常诊断服务核心类图

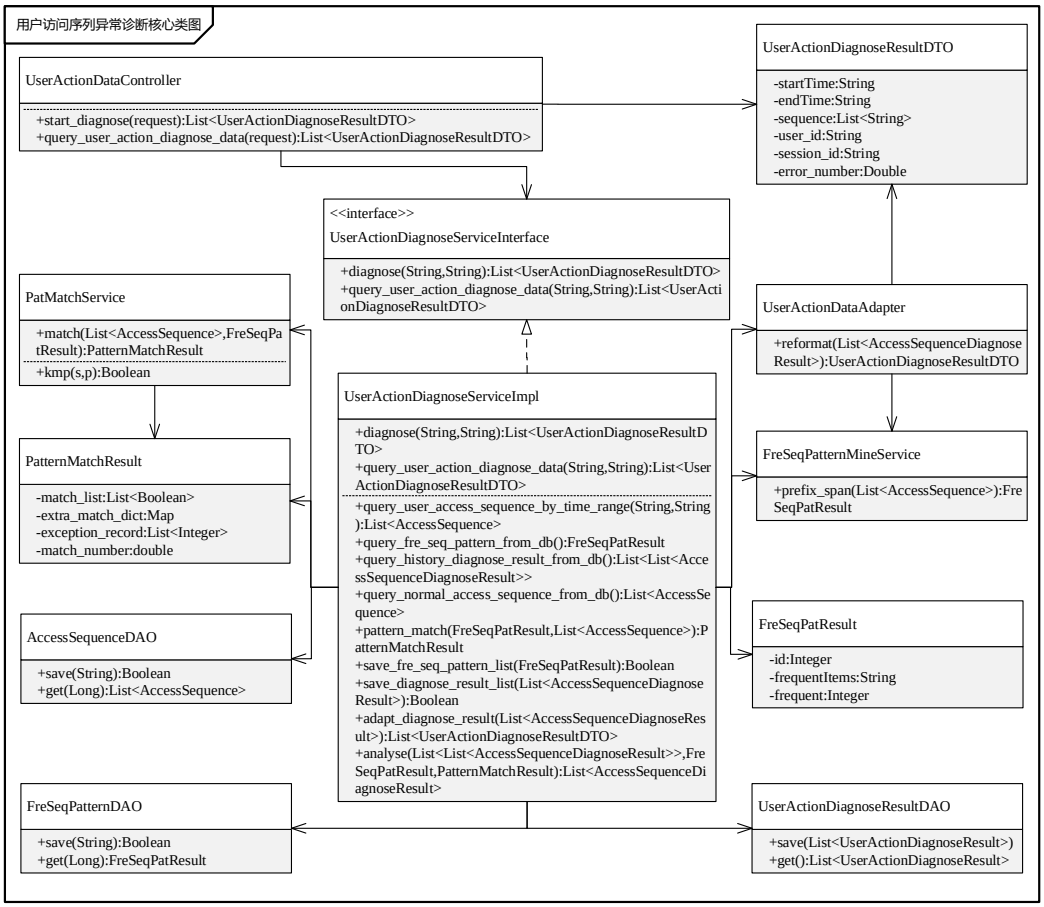

图 4-2: 用户访问序列异常诊断模块类图

图4-2是用户访问序列异常诊断服务的核心类图。UserActionDataController为用户访问序列异常诊断服务对外服务的控制层，可以通过 http 方式被调用，负责 http 请求的解析与转发，具体服务通过则调用下层的 UserActionDiagnoseService 实现。UserActionDiagnoseServiceInterface 是用户访问序列异常诊断接口，具体的实现类为 UserActionDiagnoseServiceImpl，通过接口与实现类分离的方式实现上层调用与下层实现的解耦，对外提供保存诊断结果（save_diagnose_result）、获取频繁序列模式（query_fre_seq_pattern_from_db）、获取正常访问序列（query_normal_access_sequence_from_db）、模式匹配（pattern_match）、查询诊断结果（query_user_action_diagnose_result）以及结果分析（analyse）等方法。FreSeqPatternMineService 是频繁序列模式挖掘方法的具体实现类，对外提供挖掘频繁序列模式（prefix_span）等方法。PatMatch-

Service 为模式匹配的实现类，对外提供匹配（match）等方法。FreSeqPattern-DAO、AccessSequenceDAO 以及 UserActionDiagnoseResultDAO 等为 dao 层实现类，负责从数据库中查询历史诊断信息以及向 MySQL 回写频繁序列模式等服务端与数据库之间的交互。UserActionDiagnoseResultDTO 为后端返回前端的诊断结果抽象类。PatternMatchResult 为模式匹配结果信息的抽象类，FreSeqPatResult 为使用挖掘算法获取的频繁序列模式结果的抽象类。

4.1.3 用户访问序列异常诊断服务顺序图

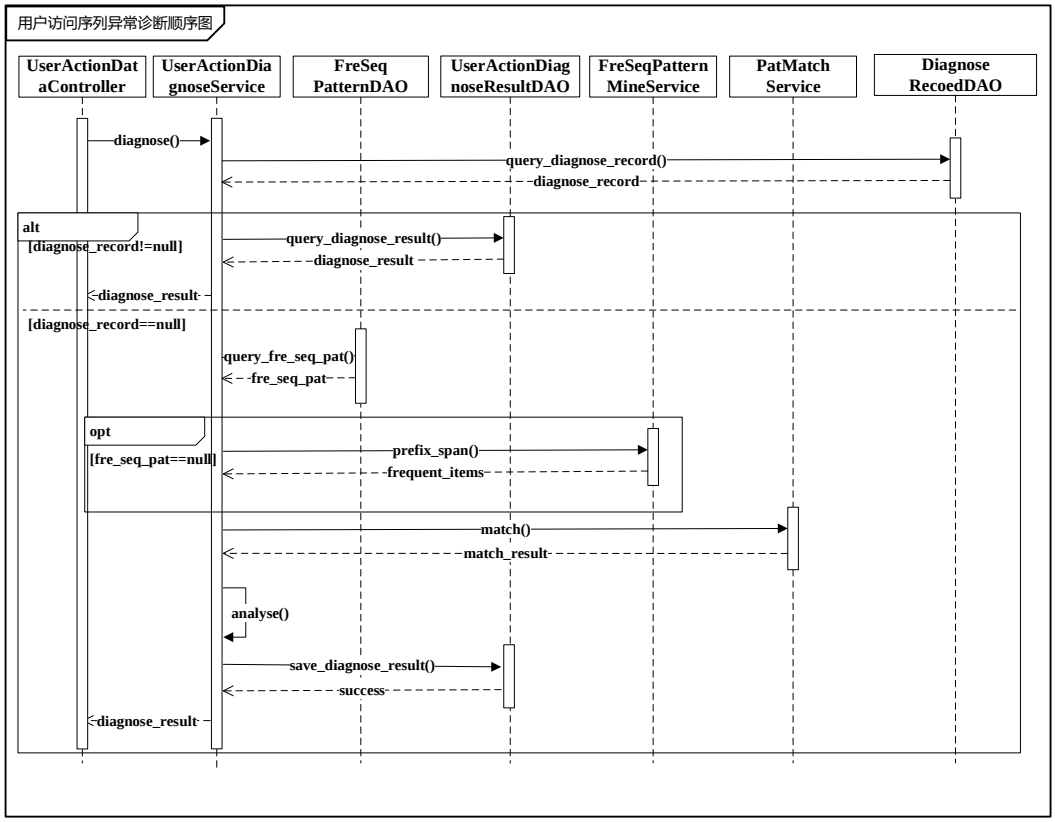

图 4-3: 用户访问序列异常诊断模块顺序图

图4-2是用户访问序列异常诊断服务的顺序图。当系统收到用户的查询请求时，UserActionDataController 类会首先对请求进行解析，获取其中时间段信息，调用 query_access_sequence_diagnose_result_list_by_time_range() 方法查询该时间段内是否已有诊断结果，如果已有诊断结果则直接返回该结果。如果数据库中没有该时间段的诊断结果，系统会调用 query_fre_seq_pattern_from_db()

从数据库中获取频繁序列模式，如果频繁序列模式结果不存在则会调用查询正常访问序列方法从数据库中查询正常访问序列，如果数据库中没有正常访问序列则提示用户需设置正常访问序列，用户可以通过 set_normal_access_seq_from_db() 方法设置正常访问序列。获取到正常访问序列后，系统会调用 FreSeqPatternMineService 类中的 prefix_span() 方法挖掘频繁序列模式。挖掘正常访问序列得到频繁序列模式后，系统会调用 pattern_match() 方法进行模式匹配，接下来系统会调用 query_history_diagnose_result_list() 方法获取历史诊断结果。之后系统将历史诊断结果与匹配结果作为参数调用 analyse() 获取当前诊断结果。最后系统会调用 save_match_result() 与 save_analyse() 方法将匹配结果与诊断结果保存到 MySQL 数据库中。

4.1.4 用户访问序列异常诊断服务关键代码

```
def match_pat(access_sequence, pattern_list):
    """
    :param access_sequence: 用户访问序列
    :param pattern_list: 频繁序列模式
    :return:
    """
    match_dict = collections.OrderedDict()
    extra_match_dict = {}
    exception_record = {}
    match_number = 0
    # 匹配频繁序列模式是否在用户访问序列中
    for pattern in pattern_list:
        item_list = pattern[1]
        is_match = kmp(access_sequence, item_list)
        if is_match == -1:
            match_dict[str(item_list)] = False
        else:
            match_number += 1
            match_dict[str(item_list)] = True
    #  匹配频繁序列模式的子串是否在用户访问序列中
    extra_match_dict, exception_record = match_sub_pat(access_sequence=access_sequence, pattern_list=pattern_list,
                                        match_dict=match_dict)
    return list(match_dict.values()), extra_match_dict, exception_record, match_number
```

图 4-4: 用户访问序列异常诊断服务代码

图4-4为用户访问序列诊断服务的模式匹配部分代码。首先对频繁序列模式与用户访问序列进行模式匹配，记录用户访问序列与各频繁序列模式的匹配情况。之后将频繁序列模式的子串与当前用户访问序列进行模式匹配，并根据匹配结果记录是否存在异常访问行为。

图4-5为用户访问序列诊断服务的主流程代码。在接收到用户诊断请求时，系统首先会根据诊断请求的起始时间与终止时间查询数据库中诊断记录，如果诊断记录已存在，则直接格式化数据库中诊断结果并返回。如果数据库中不存在诊断记录，系统则会从数据库中获取频繁序列模式，如果频繁序列模式不存在，系统则会根据正常序列模式进行挖掘。之后系统会根据频繁序列模式对当前用户访问序列进行匹配，并根据匹配结果进行诊断。之后系统将保存本次诊断记录，并将格式化的诊断结果返回给前端进行展示。

```
def diagnose(start_time, end_time):
    # 如数据库中已有诊断记录则直接返回诊断结果
    diagnose_result = query_access_sequence_diagnose_result_list_by_time_range(start_time=start_time, end_time=end_time)
    if diagnose_result is not None:
        return reformat_diagnose_result(diagnose_result)
    # 从数据库中获取频繁序列模式
    fre_seq_pattern_list = query_fre_seq_pattern_from_db()
    # 如果数据库中没有频繁序列模式，则重新挖掘
    if fre_seq_pattern_list is None:
        # 从数据库中获取用户正常访问序列
        normal_access_sequence_list = query_normal_access_sequence_from_db()
        # 如果用户正常访问序列则无法诊断，抛出异常提示用户预先设置
        if normal_access_sequence_list is None or len(normal_access_sequence_list) == 0:
            raise Exception("诊断出错。用户正常访问序列不存在，请及时设置！")
        # 如果用户正常访问序列数量低于MIN_ACCESS_SEQUENCE_LENGTH，抛出异常提示用户补充设置
        if len(normal_access_sequence_list) < MIN_ACCESS_SEQUENCE_LENGTH:
            raise Exception("诊断出错。用户正常访问序列数量不足，请及时补充设置！")
        # 挖掘频繁序列模式
        fre_seq_pattern_list = fre_seq_pattern_mine_service.prefix_span(normal_access_sequence_list)
        # 保存频繁序列模式
        save_fre_seq_pattern_list(fre_seq_pattern_list=fre_seq_pattern_list)
    # 获取指定时间段内用户的访问序列
    access_sequence_list = query_user_access_sequence_by_time_range(start_time=start_time, end_time=end_time)
    # 模式匹配
    match_result_list = pattern_match(fre_seq_pattern_list=fre_seq_pattern_list,
                                      access_sequence_list=access_sequence_list)
    # 频繁项元素
    fre_items_list = get_fre_items_list(fre_seq_pattern_list=fre_seq_pattern_list)
    access_single_check_result = access_single_check(fre_seq_pattern_list=fre_items_list,
                                                     access_sequence_list=access_sequence_list)
    # 根据模式匹配结果分析是否存在异常
    diagnose_result_list = analyse(access_single_check_result=access_single_check_result,
                                   match_result_list=match_result_list)
    # 保存诊断结果
    diagnose_result_list = save_diagnose_result_list(start_time=start_time, end_time=end_time,
                                   diagnose_result_list=diagnose_result_list)
    # 格式化诊断结果
    diagnose_result_dto = reformat_diagnose_result(diagnose_result_list=diagnose_result_list)
    # 返回诊断结果
    return diagnose_result_dto
```

图 4-5: 用户访问序列异常诊断服务主流程代码

4.2 系统方法异常诊断服务详细设计与实现

系统方法异常诊断模块负责诊断 Web 服务中是否有存在异常的系统方法，诊断系统运行时是否有发生错误、运行时间过长或者硬件资源占用率过高的系统方法并将其展示给运维人员或开发人员，以便及时修复该方法，保证 Web 服务平稳运行。本模块的预处理阶段是指根据系统配置项对业务监控数据的方法属性进行加权操作以及过滤掉监控信息不完整的记录等操作。系统方法聚类是指将系统方法按照运行时的属性信息进行聚类。保存数据是指将诊断结果与聚类结果写回到 MySQL 中。

4.2.1 系统方法异常诊断服务架构图

系统方法异常诊断服务的架构图如图 4-6所示。前端通过 http 请求访问服务端，服务端进行异常诊断后返回诊断结果，并将诊断结果与聚类结果等数据保存到 MySQL 中。

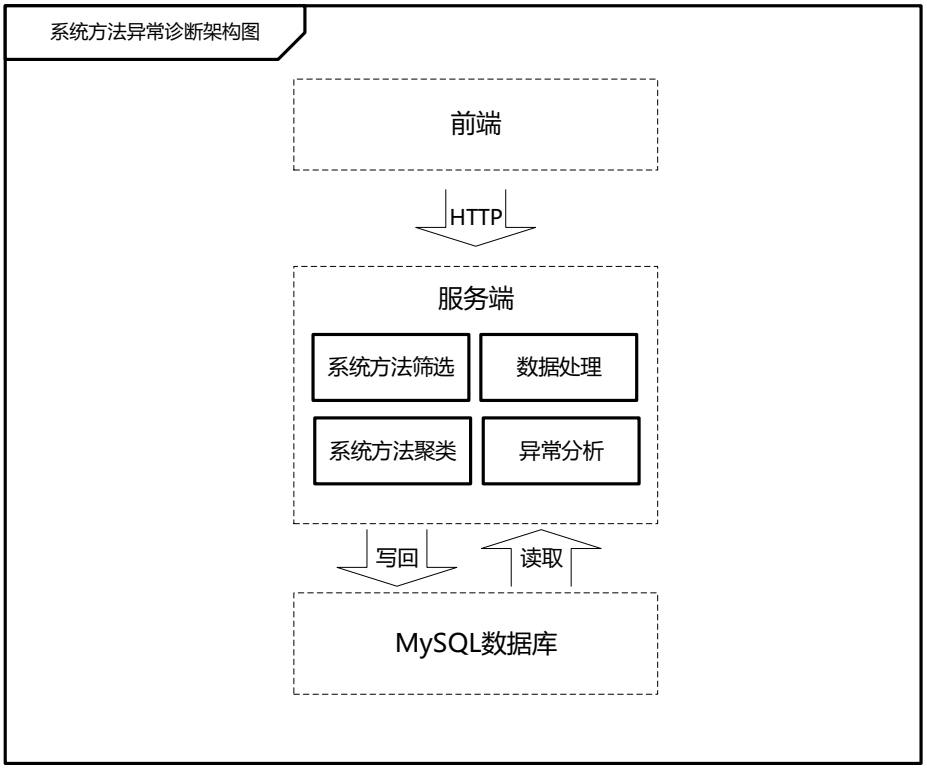

图 4-6: 系统方法异常诊断模块架构图

4.2.2 系统方法异常诊断服务核心类图

图4-7是系统方法异常诊断服务的核心类图。SystemMethodDataController为用户系统方法异常诊断服务对外服务的控制层，可以通过 http 的方式被调用，负责对 http 请求进行解析与转发，具体服务通过调用下层的 service 实现。SystemMethodDiagnoseServiceInterface 是用户访问序列异常诊断接口，SystemMethodDiagnoseService 是具体的实现类，对外提供获取诊断结果（query_system_method_diagnose_result_by_time_range）、获取聚类结果（query_cluster_result）、数据筛选处理（system_method_filter）、获取指定时间段内的业务监控数据（query_system_method_list_by_time_range）、数据过滤与调整（system_method_adapt）、聚类（cluster）、获取历史诊断结果（get_history_diagnose_result_list）以及结果分析（analyse）等方法。ClusterService 是系统方法聚类的实现类，ClusterService 对外提供聚类（cluster）等方法。ClusterResultDAO 以及 SystemMethodDAO 等为 dao 层实现类，负责在数据库中读写聚类信息与在数据库中读写系统方法信息等。SystemMethodDiagnoseResultDTO 为后端返回前端的诊断结果抽象类。ClusterResult 是聚类结果的抽象类。SystemMethod 是系统方法具体信息的抽象类。

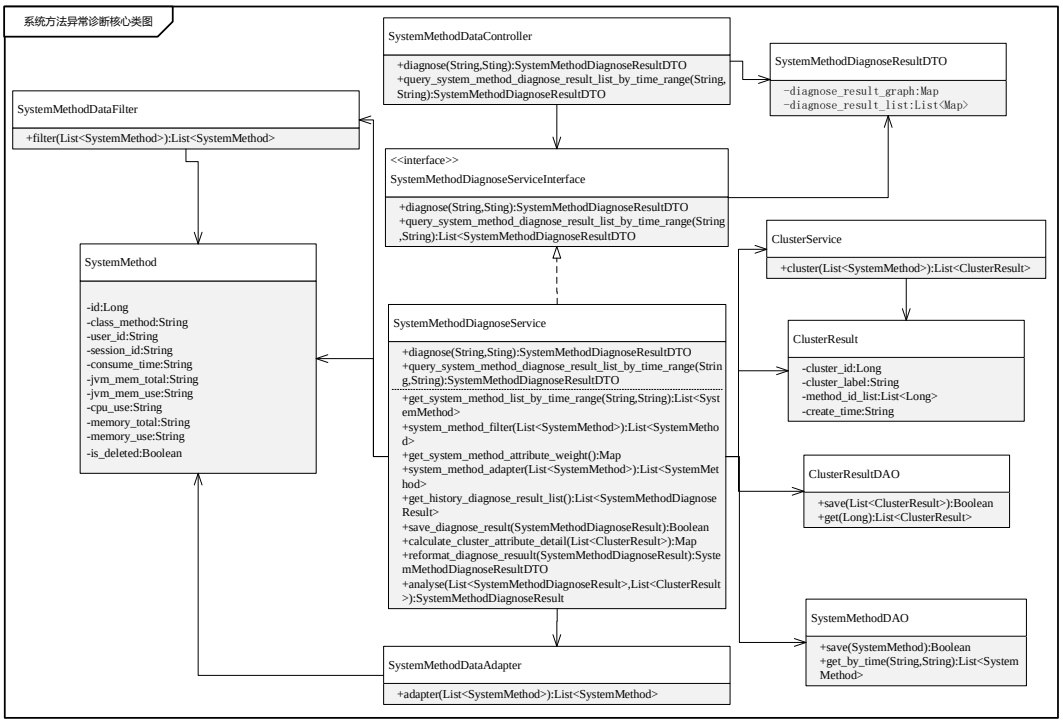

图 4-7: 系统方法异常诊断模块类图

4.2.3 系统方法异常诊断服务顺序图

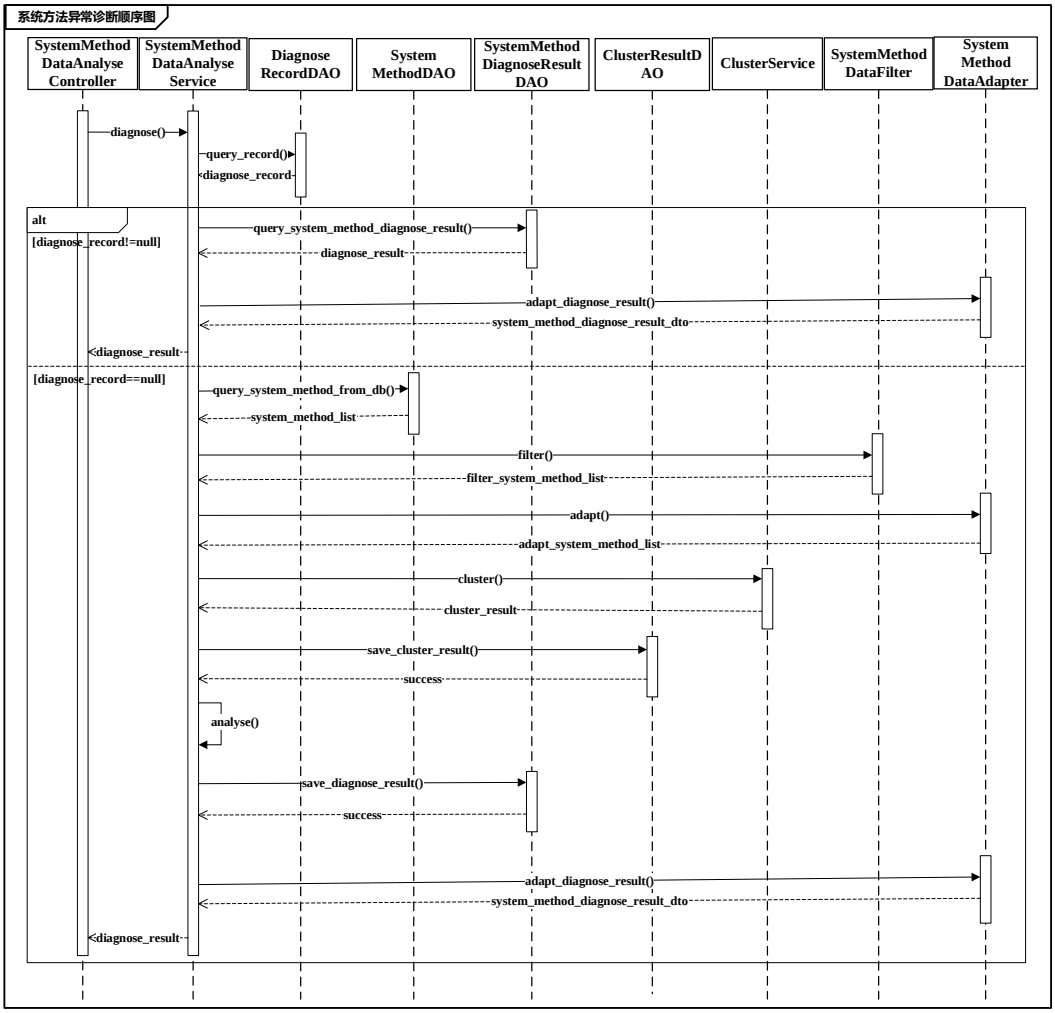

图 4-8: 系统方法异常诊断模块顺序图

图4-8是系统方法异常诊断服务的顺序图。当系统收到用户的查询请求时，SystemMethodDataController 类会首先对请求进行解析，获取其中时间段信息。SystemMethodDataController 会根据请求参数调用 SystemMethodDiagnoseService 中的 query_system_method_diagnose_result_list_by_time_range() 方法查询该时间段内是否已有诊断结果，如果已有诊断结果则直接返回该结果。如果数据库没有该时间段的诊断结果，系统会调用 query_cluster_result_from_db() 从数据库中获取聚类结果，如果数据库中不存在当前时间段内系统方法的聚类结果则会调用 query_system_method_list_from_db() 查询指定时间段内的系统方法信息，如果当前时间段内没有系统方法信息则会提示用户重新选

择查询时间段。获取到系统方法信息后，系统会调用 system_method_filter() 与 system_method_adapter() 方法对系统方法进行预处理，根据系统配置项对业务监控数据的方法属性进行加权操作。预处理之后系统会调用 ClusterService 类中的 cluster() 方法对系统方法进行聚类。聚类之后系统会调用 calculate_cluster_attribute_detail() 方法分别计算各类别中平均属性信息。处理之后系统将查询历史诊断结果并将其与当前聚类结果作为参数调用 analyse() 进行诊断。最后系统会调用 save_cluster_result() 与 save_diagnose_result() 方法将当前聚类结果与当前诊断结果保存到 MySQL 数据库中。

4.2.4 系统方法异常诊断服务关键代码

```
def diagnose(start_time, end_time):
    """
    诊断时间段内的系统方法是否存在异常
    :param start_time: 起始时间
    :param end_time: 终止时间
    :return: 诊断结果
    """
    # 查询诊断记录
    # 如果存在诊断记录，则返回诊断结果
    diagnose_result = query_system_method_diagnose_result_list_by_time_range(start_time=start_time,
end_time=end_time)
    if diagnose_result is not None:
        return reformat_diagnose_result(origin_diagnose_result=diagnose_result)
    # 获取系统方法信息
    system_method_list = get_system_method_list_by_time_range(start_time=start_time, end_time=end_time)
    if system_method_list is None:
        raise Exception('诊断出错。当前时间段内无监控数据！')
    # 过滤信息
    filter_system_method_list, id_to_detail_dict = system_method_filter(system_method_list=system_method_list)
    # 调整系统方法信息（加权）
    adapter_system_method_list, id_list = system_method_adapter(system_method_list=filter_system_method_list)
    # 聚类
    cluster_id, cluster_result_list = cluster_service.cluster(adapter_system_method_list=adapter_system_method_list,
                                                              id_list=id_list)
    # 获取历史诊断结果
    history_diagnose_result_list = get_history_diagnose_result_list()
    # 根据历史诊断结果与聚类结果进行分析
    diagnose_result = analyse(history_diagnose_result_list=history_diagnose_result_list,
                              cluster_id=cluster_id,
                              id_to_detail_dict=id_to_detail_dict,
                              cluster_result_list=cluster_result_list)
    # 保存诊断结果
    diagnose_result = save_diagnose_result(diagnose_result=diagnose_result, start_time=start_time,
                                           end_time=end_time)
    # 格式化结果
    diagnose_result_dto = reformat_diagnose_result(origin_diagnose_result=diagnose_result)
    return diagnose_result_dto
```

图 4-9: 系统方法异常诊断服务关键代码

图4-9为系统方法诊断服务的关键代码。在接收到用户诊断请求时，系统首先会解析出请求中的诊断起始时间与诊断终止时间。之后会根据起始时间与终止时间查询数据库中的诊断记录，如果诊断记录不为空，则从数据库中获取诊断结果并进行格式化，将其返回给用户。如果诊断记录为空则获取系统方法信息进行聚类，并根据聚类结果与历史诊断结果进行诊断。最后将诊断结果进行格式化并返回给用户。

4.3 业务监控数据收集服务详细设计与实现

业务监控数据收集模块负责收集系统运行时的监控数据，所监控的系统运行时，本系统会根据时间序列收集系统运行信息。系统运行信息通过 Filebeat 进行转发，使用 Elasticsearch 搜索引擎进行持久化，提高本系统处理数据时获取业务监控数据的效率。

4.3.1 业务监控数据收集服务架构图

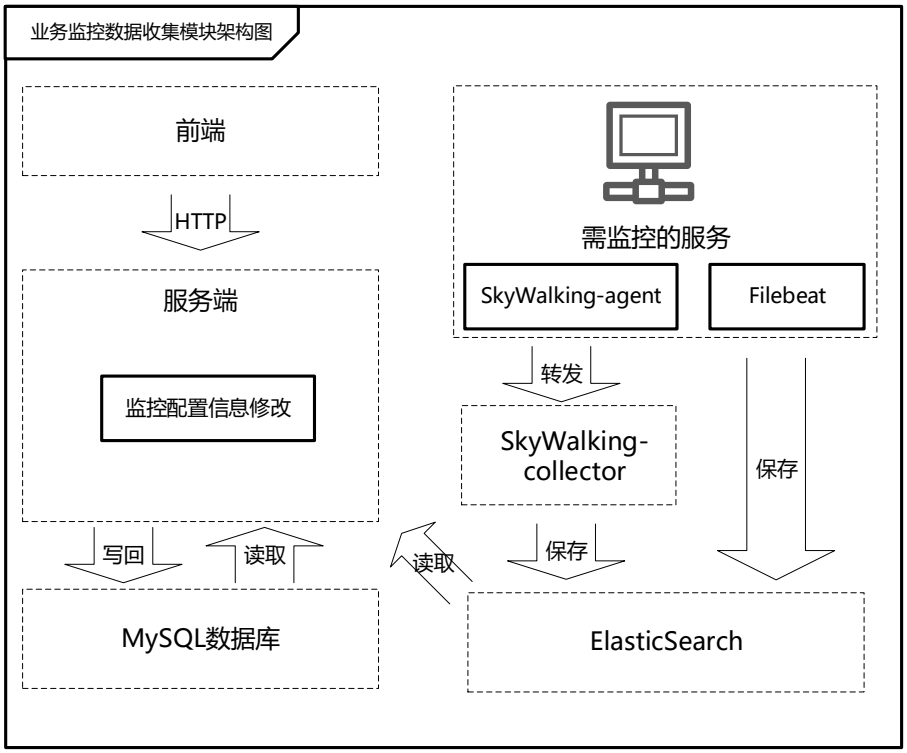

图 4-10: 业务监控数据收集服务架构图

业务监控数据收集服务的架构图如图4-10所示，前端通过 http 请求访问服务端，对监控配置信息进行修改，SkyWalking 与 Spring AOP 收集原始业务监控数据并保存到 Elasticsearch 中。

4.3.2 业务监控数据收集服务核心类图

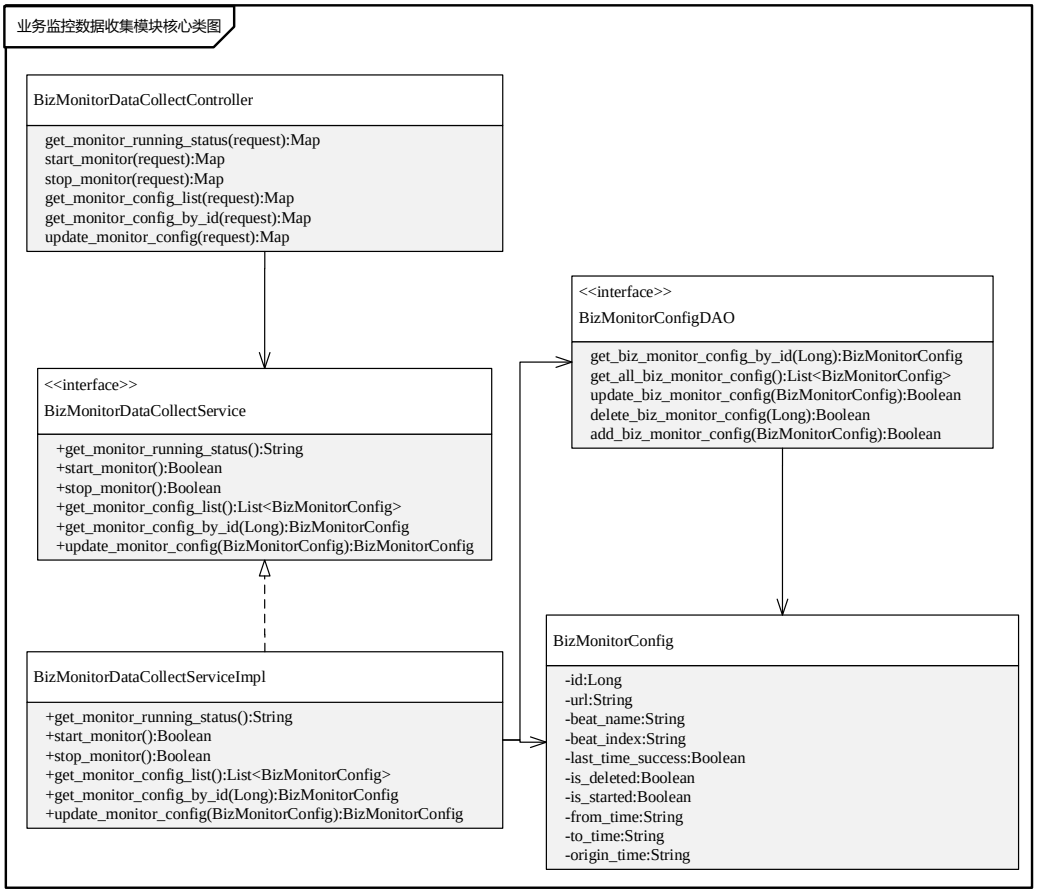

图 4-11: 业务监控数据收集服务核心类图

图4-11是业务监控数据收集服务的核心类图。由于业务监控数据收集服务的主要工作为搭建部署监控服务，所以在服务端中的代码部分较少，仅有修改配置信息等少数代码。其中 BizMonitorDataCollectController 为业务监控数据收集服务对外服务的控制层，可以通过 http 方式被调用，负责 http 请求的解析与转发，具体服务通过调用下层的 service 实现。BizMonitorDataCollectServiceInterface 是业务监控数据收集服务接口，BizMonitorDataCollectService 是具体的实现类，对外提供开始修改配置信息（update_monitor_config）、启动监控

（start_monitor）、获取监控状态（get_monitor_running_status）以及停止监控（stop_monitor）等服务。BizMonitorConfigDAO 负责系统读写业务监控数据收集服务配置信息。BizMonitorConfig 是业务监控数据收集服务配置信息的抽象类，包括 Elasticsearch 服务地址等信息。

4.3.3 业务监控数据收集服务顺序图

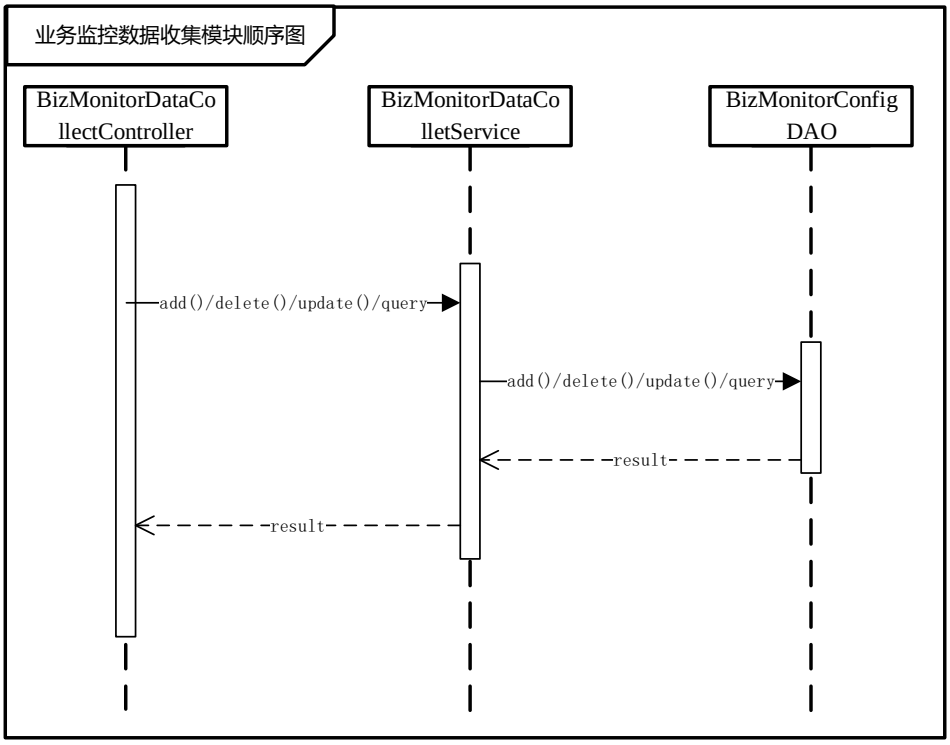

图 4-12: 业务监控数据收集服务顺序图

图 4-12是业务监控数据收集服务的顺序图。当需要监控的服务处于运行状态时，SkyWalking-agent 会自动监控并转发业务监控信息，SkyWalking-collector 接收业务监控数据并保存到 Elasticsearch 服务中。当系统运行生成日志信息时，Filebeat 工具会自动转发指定日志信息并会将日志信息保存到 Elasticsearch 搜索引擎中，以便业务数据处理模块的使用。当系统接收到用户请求时，会通过 BizMonitorDataCollectController 对请求进行解析，并根据解析后的请求调用 BizMonitorDataCollectService 接口中的方法获取监控配置信息或修改监控配置信息等数据。

4.4 业务监控数据处理服务详细设计与实现

业务监控数据处理服务负责查询、处理并保存业务监控数据。本系统的定时任务会周期性调用业务监控数据处理服务。处理业务监控信息时，系统首先会通过 http 请求根据指定查询条件在 Elasticsearch 搜索引擎中获取指定时间段内的业务监控数据，处理数据是指过滤信息不完整的业务监控数据以及从日志信息中提取业务监控数据关键属性等操作。保存数据是指按照时间序列将处理后的数据保存到 MySQL 数据库中。

4.4.1 业务监控数据处理服务架构图

业务监控数据处理服务架构图如图4-10所示，从 Elasticsearch 中获取原始业务监控数据，对其进行筛选等操作，并将处理后的业务监控数据保存到 MySQL 中。

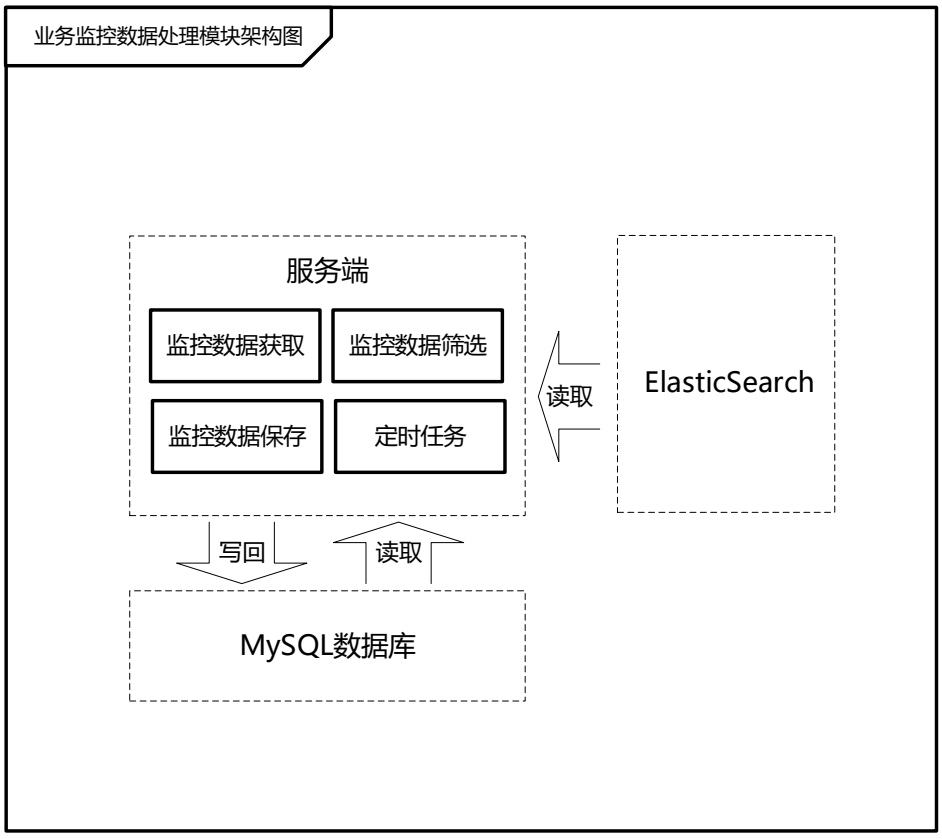

图 4-13: 业务监控数据处理模块架构图

4.4.2 业务监控数据处理服务核心类图

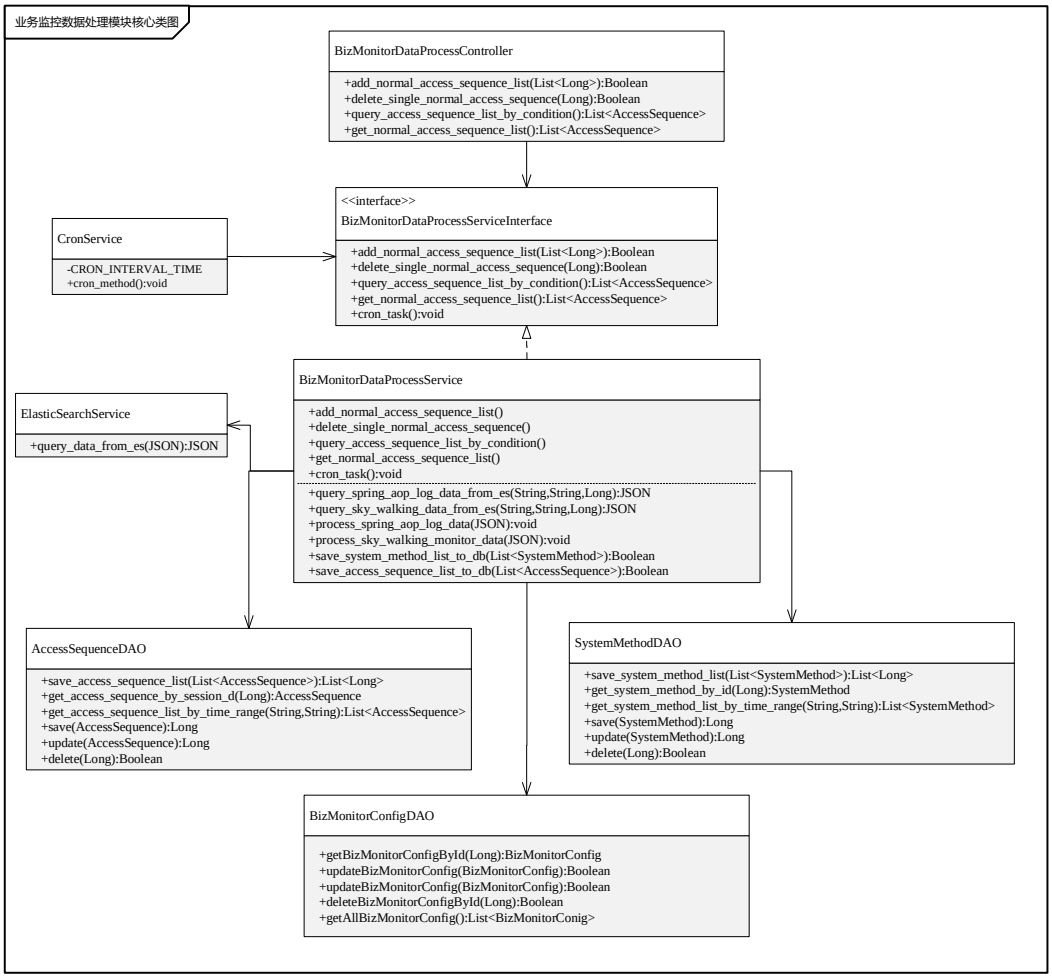

图 4-14: 业务监控数据处理模块类图

业务监控数据处理服务的核心类图如图4-14所示。其中 BizMonitorDataProcessController 为业务监控数据处理服务对外服务的控制层，可以通过 http 方式被调用，负责 http 请求的解析与转发，具体服务通过调用下层的 BizMonitorDataProcessService 实现。BizMonitorDataProcessServiceInterface 是业务监控数据处理接口，BizMonitorDataProcessService 是具体的实现类，对外提供获取 Spring AOP 收集的日志信息（query_spring_aop_log_data_from_es）、获取 SkyWalking 收集的监控信息（query_sky_walking_monitor_data_from_es）、处理 Spring AOP 收集的日志信息（process_spring_aop_log_data）、处理 SkyWalking 收集监控信息（process_sky_walking_monitor_data）以及保存处理后的业务监控数据（save_system_method_list_to_db 与 save_access_sequence_list_to_db）

等方法。CronService 为系统定时任务服务，负责以 CRON_INTERVAL_TIME（默认为 60 秒）作为时间间隔周期性的调用系统中的服务，提供定时方法（cron_method）。

4.4.3 业务监控数据处理服务顺序图

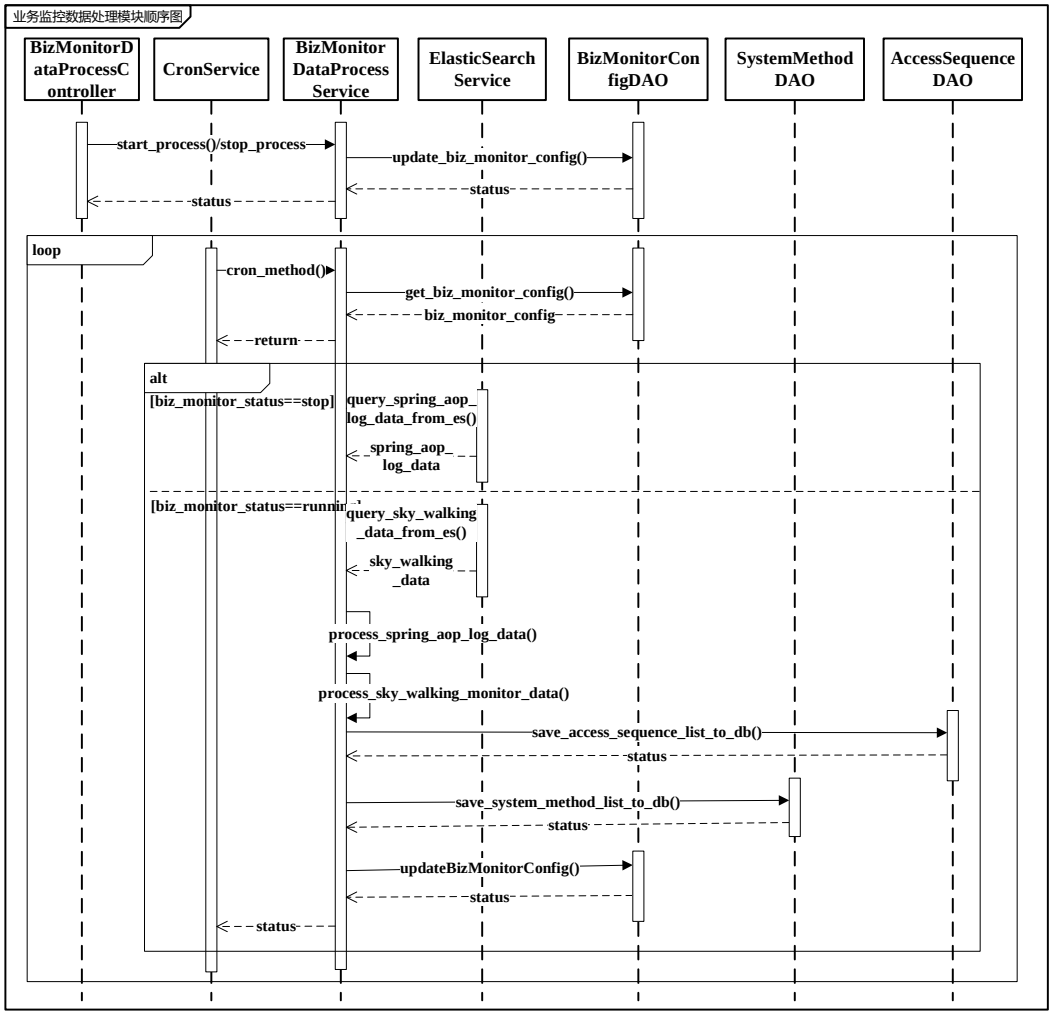

图 4-15: 业务监控数据处理服务顺序图

图4-15为业务监控数据处理服务的顺序图。CronService 服务会周期性的调用 BizMonitorDataProcessConfigService 中的 query_process_status_from_db() 方法查询当前业务监控数据收集配置的状态，当配置中的处理状态为运行中（process_status=1），系统则会开始处理监控信息，否则便不会处理监控信息。当系统收到用户的请求时，BizMonitorDataProcessController 类会首先对请求进行解析，并根据请求结果调用 service 层的服务。当用户请求为修

改或查看业务监控数据的处理配置信息时，系统便会调用 BizMonitorDataProcessConfigService 中的开始处理业务监控信息 start_process() 或停止处理业务监控信息 stop_process() 等方法。stop_process() 方法首先会检查业务监控数据处理配置的当前状态，如果当前状态为已停止（process_status=0），则会向用户返回错误信息，提示用户业务监控数据处理配置的当前状态为已停止，否则系统会备份当前状态，之后将当前状态修改为已停止。start_process() 方法会首先查询业务监控数据处理配置的当前状态，如果当前状态为运行中（process_status=1），则会向用户返回错误信息，提示用户业务监控数据处理配置的当前状态为已启动，否则会备份配置的当前状态，之后将当前状态修改为运行中，系统会获取当前时间，设置获取监控信息的时间段信息。BizMonitorDataProcessService 接口中的方法可被定时任务调用。当定时任务查看当前的业务监控数据处理状态为运行中（process_status=1）时，便会调用 BizMonitorDataProcessService 服务处理监控数据。系统中的定时任务首先会通过 query_spring_aop_log_data_from_es() 与 query_sky_walking_monitor_data_from_es() 从 Elasticsearch 搜索引擎中获取指定时间段内的原始业务监控数据。系统获取到原始业务监控数据后会调用 process_spring_aop_log_data() 与 process_sky_walking_log_data() 从原始业务监控数据中筛选需要的数据，过滤冗杂数据。通过 save_system_method_list_to_db() 与 save_access_sequence_list_to_db() 将处理后的业务监控数据保存到 MySQL 数据库中。保存完毕后会更新业务监控数据处理配置中的状态与时间。当处理过程中出现异常时，系统会最多重复处理该时间段的原始业务监控数据 MAX_PROCESS_TIME（默认为 3）次，超过 MAX_PROCESS_TIME 次后系统会自动跳过该时间段，继续处理之后的数据。

4.5 本章小结

本章主要阐述了基于业务监控数据的 Web 服务诊断系统的详细设计与实现，包括用户访问序列诊断模块、系统方法诊断模块、业务监控数据收集模块以及业务监控数据处理模块。本章通过架构图、核心类图、顺序图以及关键代码对各模块进行描述，其中架构图可以描述各模块的总体架构设计，核心类图与关键代码可以从静态代码的角度描述系统设计思路，顺序图可以从动态运行的角度描述系统设计思路。

第五章　系统测试与系统实例展示

本章节的主要内容为依据第三章中分析得出的系统功能性需求与非功能性需求，详细设计与实现对本系统的测试，并展示系统运行实例截图。为了保障本系统在实际部署后能够平稳运行，需要对本系统的功能与性能等进行全面测试。本章节将从测试环境准备、功能测试、性能测试以及效果测试等不同方面对本系统进行测试。

5.1　测试准备

5.1.1　测试目标

根据第三章中对于本系统的需求分析，本章中对本系统的测试主要包括功能测试、性能测试以及效果测试三个方面。

（1）功能测试。功能测试是指根据系统用例编写测试用例，并根据测试用例对系统各功能进行测试，以验证系统是否正确实现了需求分析阶段所列出的功能。功能测试的主要内容包括系统正常功能测试、系统异常功能测试、接口测试以及界面测试等。在本系统中，功能测试主要包括各系统各页面间能否正常跳转、能否启动监控、能否停止监控、能否查看用户访问序列诊断结果以及是否查看系统方法诊断结果等。

（2）性能测试。性能测试是指借助 JMeter 等自动化测试工具模拟正常负载以及异常负载等多种场景对系统性能进行测试，以验证系统能否应对模拟的应用场景。在本系统中，性能测试主要包括能否在源系统并发量较大的情况下收集业务监控数据，以及当源系统产生的业务监控数据量较大时本系统能否对源系统进行诊断。

（3）效果测试。效果测试是指系统提供的服务能否满足用户需求。在本系统中，效果测试是指能否准确诊断 Web 服务中的低性能方法以及未进行权限控制的接口。

5.1.2 测试环境

为了便于获取日志信息与监控信息，本系统将 Filebeat 与 SkyWalking-agent 与所监控的服务部署在同一服务器（Server0）中；由于需要向 Elasticsearch 中保存的数据复杂繁多，为了提高监控数据存储的效率，本系统将 SkyWalking-collector 与 Elasticsearch 部署在相同的服务器（Server1）中；系统将前端、服务端以及 MySQL 数据库部署在相同的服务器（Server2）中。系统测试环境详情如表5-1所示。

表 5-1: 系统测试环境表

设备与软件	描述信息
Server0	配置：4C16G 系统：Ubuntu 1604 带宽：外网 4M
Server1	配置：4C16G 系统：Ubuntu 1604 带宽：外网 4M
Server2	配置：4C16G 系统：Ubuntu 1604 带宽：外网 4M
Django	版本：3.1.3
Python	版本：3.8
Vue.js	版本：2.5.2
Echarts	版本：5.0.2
Docker	版本：19.03.6
Elasticsearch	版本：7.9.3
SkyWalking	版本：8.2.0
MySQL	版本：5.7

5.2 功能测试

本节的主要内容为测试基于业务监控数据的 Web 服务诊断系统能否满足第三章分析得出的用户需求。

5.2.1 测试设计

本小节主要根据第三章中分析得出的功能性需求设计系统测试用例，对系统的主要功能进行测试，通过对比系统的实际表现与预期结果，验证本系统各功能是否准确。

表5-2中所展示的是启动监控测试用例，提供了启动监控功能的具体测试过程与预期结果等信息。该测试用例针对用例描述 *UC*_1 设计，主要目的为覆盖启动监控时用户的各项操作步骤以及验证启动监控后系统的状态是否更新为“运行中”并开始获取监控信息。

表 5-2: 启动监控测试用例表

测试用例 ID	*TC*_1
测试名称	启动监控测试用例
测试功能	用户可实时启/停监控系统，用户启动监控后， 本系统从当前时间开始收集监控信息。
测试步骤	1. 用户进入本系统，是否可以看到管理页面； 2. 点击管理页面，是否跳转到管理页面； 3. 点击“启动监控”按钮，是否弹出确认弹窗； 4. 点击“取消”，系统监控状态是否仍为“停止”； 5. 再次点击“启动监控”按钮，是否弹出确认弹窗； 6. 点击弹窗中的“确认”，系统监控状态是否变为“运行中”。
预期结果	1. 用户进入本系统，可以看到管理页面； 2. 点击管理页面，可以跳转到管理页面； 3. 点击“启动监控”按钮，弹出确认弹窗； 4. 点击“取消”，系统监控状态仍为“停止”； 5. 再次点击“启动监控”按钮，弹出确认弹窗； 6. 点击弹窗中的“确认”，系统监控状态变为“运行中”。

表5-3中所展示的是停止监控测试用例，提供了停止监控功能的具体测试过程与预期结果等信息。该测试用例针对用例描述 *UC*_2 设计，主要目的为覆盖停止监控时用户的各项操作步骤以及验证停止监控后系统状态是否更新为“停止”并停止获取监控信息。

表5-4中所展示的是修改监控配置信息测试用例，提供了修改监控配置信息功能的具体测试过程与预期结果等信息。该测试用例针对用例描述 *UC*_3 设计，主要目的为覆盖修改监控配置信息时用户的各项操作步骤以及验证用户操作后系统的监控配置信息是否得到更新。

表 5-3: 停止监控测试用例表

测试用例 ID	*TC*_2
测试名称	停止监控测试用例
测试功能	用户可实时启/停监控系统，用户启动监控后， 本系统从当前时间开始收集监控信息。
测试步骤	1. 用户进入本系统，点击管理页面，是否跳转到管理页面； 2. 点击“停止监控”按钮，是否弹出确认弹窗； 3. 点击“取消”，系统监控状态是否仍为“停止”； 4. 再次点击“停止监控”按钮，是否弹出确认弹窗； 5. 点击弹窗中的“确认”，系统监控状态是否变为“停止”。
预期结果	1. 用户进入本系统，点击管理页面，跳转到管理页面； 2. 点击“停止监控”按钮，弹出确认弹窗； 3. 点击“取消”，系统监控状态仍为“停止”； 4. 再次点击“停止监控”按钮，弹出确认弹窗； 5. 点击弹窗中的“确认”，系统监控状态变为“停止”。

表 5-4: 修改监控配置信息测试用例表

测试用例 ID	*TC*_3
测试名称	修改监控配置信息测试用例
测试功能	用户通过本功能修改系统中的监控配置信息。
测试步骤	1. 用户进入本系统，是否可以看到管理页面； 2. 点击管理页面，是否跳转到管理页面； 3. 是否可以查看到当前监控配置信息； 4. 点击“修改配置信息”，系统是否弹出编辑页面； 5. 编辑监控信息后，点击“提交按钮”，监控信息是否已更新。
预期结果	1. 用户进入本系统，可以看到管理页面； 2. 点击管理页面，跳转到管理页面； 3. 用户可以查看到当前监控配置信息； 4. 点击“修改配置信息”，系统弹出编辑页面； 5. 编辑监控信息后，点击“提交按钮”，监控信息更新。

表5-5中所展示的是选定诊断时间段测试用例，提供了选定诊断时间段功能的具体测试过程与预期结果等信息。该测试用例针对用例描述 *UC*_4 设计，主要目的为覆盖选定诊断时间段时用户的各项操作步骤。

表5-6中所展示的是设置用户正常访问序列测试用例，提供了设置用户正常访问序列功能的具体测试过程与预期结果等信息。该测试用例针对用例描述 *UC*_5 设计，主要目的为覆盖设置用户正常访问序列时用户的各项操作步骤。

表 5-5: 选定诊断时间段测试用例表

测试用例 ID	*TC*_4
测试名称	选定诊断时间段测试用例
测试功能	用户可以选择用户 id 与 sessionId 获取指定的访问序列。
测试步骤	1. 用户进入本系统，是否可以看到用户访问序列诊断页面或系统方法诊断页面； 2. 点击用户访问序列诊断页面或系统方法诊断页面，是否跳转到相应页面； 3. 是否可以查看到时间控件以及“开始诊断”按钮； 4. 点击时间控件，是否可以修改起始时间与终止时间； 5. 选择完起始时间与终止时间后，查看控件时间是否更新。
预期结果	1. 用户进入本系统，可以看到用户访问序列诊断页面或系统方法诊断页面； 2. 点击用户访问序列诊断页面或系统方法诊断页面，跳转到相应页面； 3. 可以查看到时间控件以及“开始诊断”按钮； 4. 点击时间控件，可以修改起始时间与终止时间； 5. 选择完起始时间与终止时间后，查看控件时间已更新。

表 5-6: 设置用户正常访问序列测试用例表

测试用例 ID	*TC*_5
测试名称	设置用户正常访问序列测试用例
测试功能	用户可以设置正常的用户访问序列。
测试步骤	1. 用户进入本系统，是否可以查看到管理页面； 2. 点击管理页面链接，系统是否跳转到管理页面； 3. 是否可以查看到已有的用户访问序列； 4. 点击“删除”按钮，是否可以弹出确认弹窗； 5. 点击“添加”按钮，是否可以弹出选择用户访问序列页面； 6. 是否可以选择用户 id 与 sessionId； 7. 选择用户 id 与 sessionId 后是否可以预览访问序列； 8. 点击“添加”按钮，系统是否更新正常用户访问序列。
预期结果	1. 用户进入本系统，可以查看到管理页面； 2. 点击管理页面链接，系统可以跳转到管理页面； 3. 可以查看到已有的用户访问序列； 4. 点击“删除”按钮，可以弹出确认弹窗； 5. 点击“添加”按钮，可以弹出选择用户访问序列页面； 6. 可以选择用户 id 与 sessionId； 7. 选择用户 id 与 sessionId 后可以预览访问序列； 8. 点击“添加”按钮，系统更新正常用户访问序列。

表5-7中所展示的是查看单用户访问序列测试用例，提供了查看单用户访问序列功能的具体测试过程与预期结果等信息。该测试用例针对用例描述*UC_6*设计，主要目的为覆盖查看单用户访问序列时用户的各项操作步骤。

表 5-7: 查看单用户访问序列测试用例表

测试用例 ID	*TC_6*
测试名称	查看单用户访问序列测试用例
测试功能	用户可以查看单用户访问序列。
测试步骤	1. 进入本系统，是否可以查看到查看单用户访问序列页面； 2. 点击查看单用户访问序列页面链接，是否跳转到该页面； 3. 点击用户 id 下拉框，系统能否显示用户 id 信息； 4. 点击 sessionId 下拉框，系统能否显示 sessionId 信息； 5. 点击 sessionId，能否显示该用户的访问序列。
预期结果	1. 进入本系统，可以查看到查看单用户访问序列页面； 2. 点击查看单用户访问序列页面链接，系统跳转到该页面； 3. 点击用户 id 下拉框，系统显示用户 id 信息； 4. 点击 sessionId 下拉框，系统显示 sessionId 信息； 5. 点击 sessionId，显示该用户的访问序列。

表 5-8: 查看用户访问序列诊断结果测试用例表

测试用例 ID	*TC_7*
测试名称	查看用户访问序列诊断结果测试用例
测试功能	用户可以查看用户访问序列诊断结果。
测试步骤	1. 进入本系统，是否看到查看用户访问序列诊断结果页面； 2. 点击查看用户访问序列诊断结果页面链接，是否正常跳转； 3. 未选择时间段时，“开始诊断”按钮是否置灰； 4. 选择时间段后，“开始诊断”按钮是否可用； 5. 点击“开始诊断”按钮，系统是否提示用户诊断已开始； 6. 诊断结束后，用户能否查看到诊断结果。
预期结果	1. 进入本系统，可以看到查看用户访问序列诊断结果页面； 2. 点击查看用户访问序列诊断结果页面链接，系统正常跳转； 3. 未选择时间段时，“开始诊断”按钮置灰； 4. 选择时间段后，“开始诊断”按钮变为可用； 5. 点击“开始诊断”按钮，系统提示用户诊断已开始； 6. 诊断结束后，用户可以查看到诊断结果。

表5-8中所展示的是查看用户访问序列诊断结果测试用例，提供了查看用户访问序列诊断结果功能的具体测试过程与预期结果等信息。该测试用例针对

用例描述 *UC*_7 设计，主要目的为覆盖查看用户访问序列诊断结果时用户的各项操作步骤。

表5-9中所展示的是查看系统方法诊断结果测试用例，提供了查看系统方法诊断结果功能的具体测试过程与预期结果等信息。该测试用例针对用例描述 *UC*_8 设计，主要目的为覆盖查看系统方法诊断结果时用户的各项操作步骤。

表 5-9: 查看系统方法诊断结果测试用例表

测试用例 ID	*TC*_8
测试名称	查看系统方法诊断结果测试用例
测试功能	用户可以查看系统方法诊断结果。
测试步骤	1. 用户进入本系统，是否看到查看系统方法诊断结果页面； 2. 点击查看系统方法诊断结果页面链接，是否正常跳转； 3. 未选择时间段时，“开始诊断”按钮是否置灰； 4. 选择时间段后，“开始诊断”按钮是否可用； 5. 点击“开始诊断”按钮，系统是否提示用户诊断已开始； 6. 诊断结束后，用户能否查看到诊断结果。
预期结果	1. 用户进入本系统，可以看到查看系统方法诊断结果页面； 2. 点击查看系统方法诊断结果页面链接，跳转到该页面； 3. 未选择时间段时，“开始诊断”按钮置灰； 4. 选择时间段后，“开始诊断”按钮变为可用； 5. 点击“开始诊断”按钮，系统提示用户诊断已开始； 6. 诊断结束后，用户可以查看到诊断结果。

5.2.2 测试执行

表 5-10: 功能测试用例执行结果表

测试用例 ID	**用例描述 ID**	**测试结果**
*TC*_1	*UC*_1	通过，监控启动成功。
*TC*_2	*UC*_2	通过，监控停止成功。
*TC*_3	*UC*_3	通过，监控配置修改成功。
*TC*_4	*UC*_4	通过，诊断时间设定成功。
*TC*_5	*UC*_5	通过，正常访问序列设定成功。
*TC*_6	*UC*_6	通过，系统显示用户访问序列。
*TC*_7	*UC*_7	通过，系统显示诊断结果。
*TC*_8	*UC*_8	通过，系统显示诊断结果。

按照测试用例中的测试步骤严格执行，将系统实际的运行结果与测试用例中的预期结果相对比，并对实际结果与预期结果之间的差异进行记录。

表5-10为功能测试的结果。从最终测试结果可知，本系统实现的功能满足第三章中分析得出的功能性需求，功能上可以应用于实际场景。

5.3 性能测试

本节的主要内容为测试基于业务监控数据的 Web 服务诊断系统的性能在实际使用中是否能够满足用户需求。通过 JMeter 模拟正常并发场景以及高并发场景，测试本系统能否在不同的场景中稳定运行，验证本系统能否在性能上满足实际需要与是否具备实用价值。通过分析本系统中各功能，发现性能瓶颈为业务监控数据处理模块与用户访问序列异常诊断模块，针对这两处性能瓶颈，设计了以下两种模拟场景对本系统进行测试。

表 5-11: 性能测试不同并发场景测试结果表

每秒请求数量	系统平均处理时间	内存占用
1	162 ms	45.3%
10	246 ms	46.7%
30	525 ms	49.2%
50	677 ms	50.5%
80	859 ms	51.9%

表 5-12: 性能测试不同时间范围场景测试结果表

诊断时间段	系统响应时间	内存占用
10:00:00-10:30:00	1.53 秒	51.2%
10:00:00-11:00:00	1.88 秒	57.3%
10:00:00-11:30:00	2.23 秒	58.0%
10:00:00-12:00:00	2.50 秒	58.6%
10:00:00-13:00:00	2.82 秒	59.3%

本节中设计两种模拟场景对系统的性能进行测试。首先是通过 JMeter 模拟用户对源系统（以慕测平台为例）发出请求，逐步提高请求的数量并观察诊断系统处理业务监控数据的耗时以及内存使用率等指标。由于本系统对于用户数量不敏感，所以本测试场景只使用单用户。测试结果如表5-11所示。通过对源系统（以慕测平台为例）的应用场景分析，可知系统并发量最大的时间段为举办大赛期间，根据统计可知，其最大同时在线人数为 1073，每秒最大请求数为 50，全场比赛平均每秒请求数为 4，而本诊断系统在每秒请求为 80 的场景

下表现良好，符合预期效果。另一种模拟场景是选择不同的时间段范围进行诊断，在每秒请求数为 8 的条件下逐步加大诊断的时间范围。据统计可知，源系统（以慕测平台为例）中比赛时长主要为 2 小时，本系统在不同诊断时间范围表现如表 5-12所示。通过以上两种场景的测试可得，本系统在性能上可以应用于实际场景。

5.3.1 效果测试

本节的主要内容为测试本系统在实际使用场景中能否准确的对 Web 服务进行诊断，测试本系统的诊断结果能否给开发人员或运维人员提供帮助，提高开发效率并提高系统的稳定性。本小节中，以慕测平台作为源系统进行诊断，分别使用本系统对慕测平台私有云与慕测平台测试服进行诊断，对两个 Web 服务各诊断一周时间。慕测平台私有云部署的代码与慕测平台测试服部署的代码不同，慕测平台测试服部署最新的代码，慕测平台私有云部署的则是在原代码中添加了 3 处循环查数据库等低性能代码段的代码。诊断结果如表 5-13所示。

表 5-13: 慕测平台效果测试结果表

服务	测试时间范围	系统方法异常	未控制权限的接口
mooctest 私有云	2021 年 3 月 15 日-2021 年 3 月 22 日	10 个	2 个
mooctest 测试环境	2021 年 3 月 22 日-2021 年 3 月 29 日	7 个	2 个

5.4 系统实例展示

图 5-1为基于业务监控数据的 Web 服务诊断系统的首页。通过本页面，用户可以查看到管理页面跳转链接、用户访问序列诊断页面跳转链接以及系统方法诊断页面跳转链接。除此以外，用户在首页还可以通过选择用户 id 以及 sessionId 查看单用户的访问序列。

图 5-2为管理页面，在本页面中，用户可以启动或停止监控、修改监控配置信息以及设置正常用户访问序列。

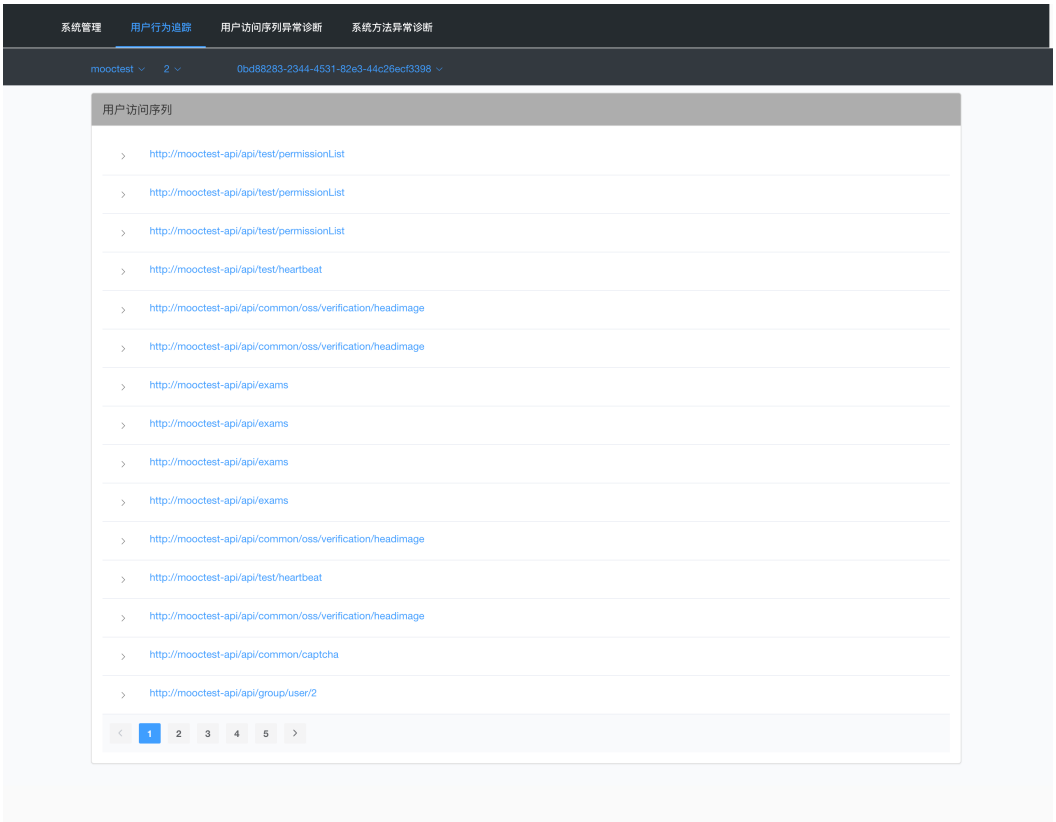

图 5-1: 用户访问序列页面截图

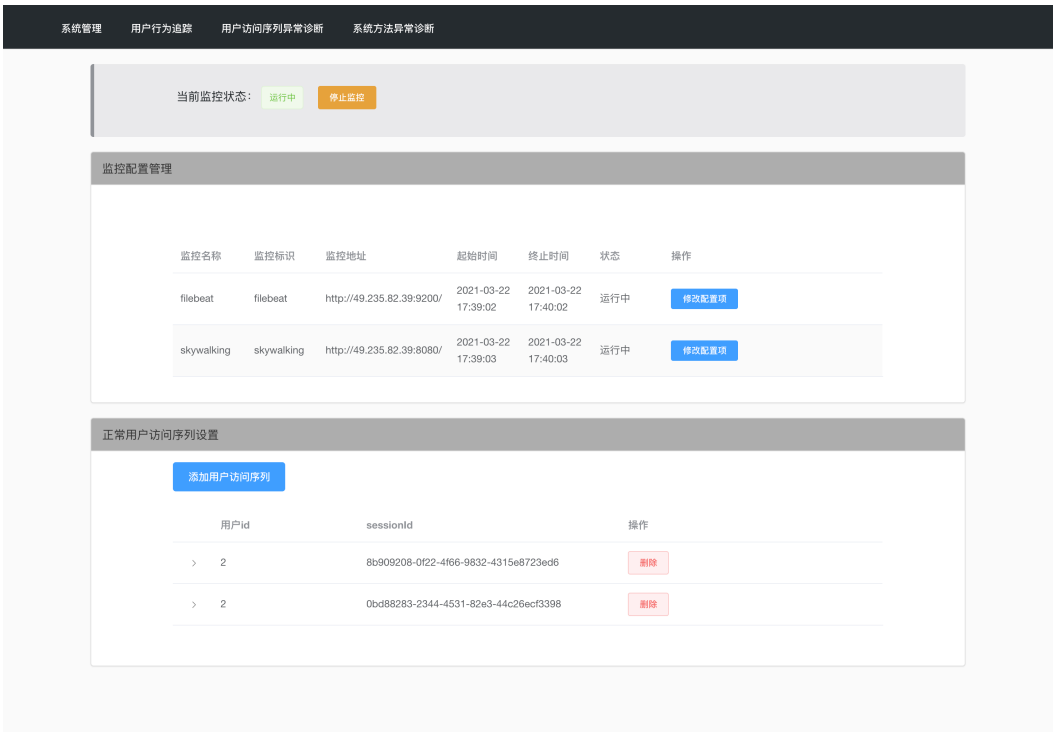

图 5-2: 管理页面截图

图 5-3为系统方法诊断页面。在本页面中用户可以设定诊断时间段，设定好诊断时间段后用户可以点击页面中的“开始诊断”按钮。点击“开始诊断”按钮后系统开始诊断系统方法是否存在异常，诊断结束后将诊断结果返回给用户。

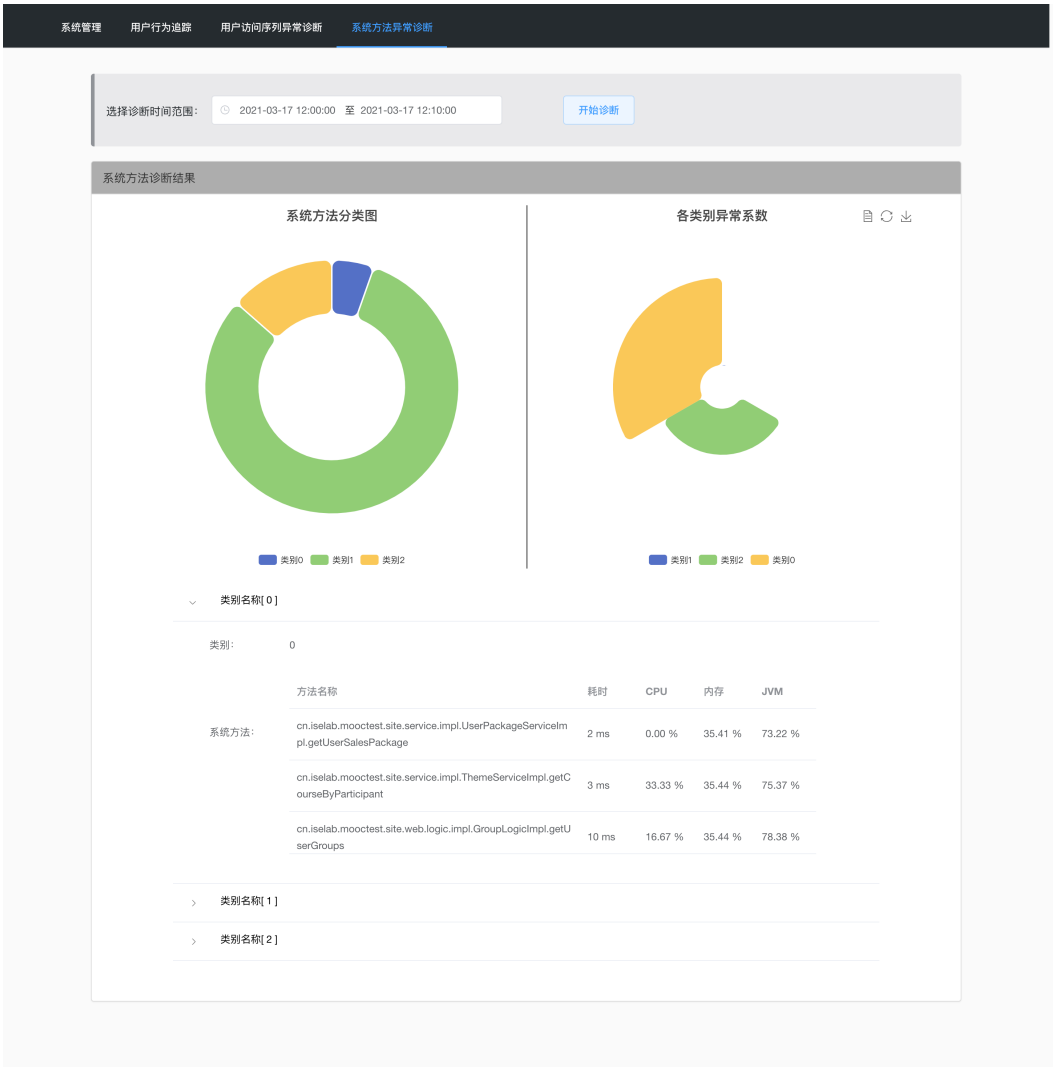

图 5-3: 系统方法诊断结果截图

图 5-4为用户访问序列诊断页面。在本页面中用户可以设定诊断时间段，设定好诊断时间段后用户可以点击页面中的“开始诊断”按钮。点击“开始诊断”按钮后系统开始诊断用户访问序列是否存在异常，诊断结束后展示诊断结果。

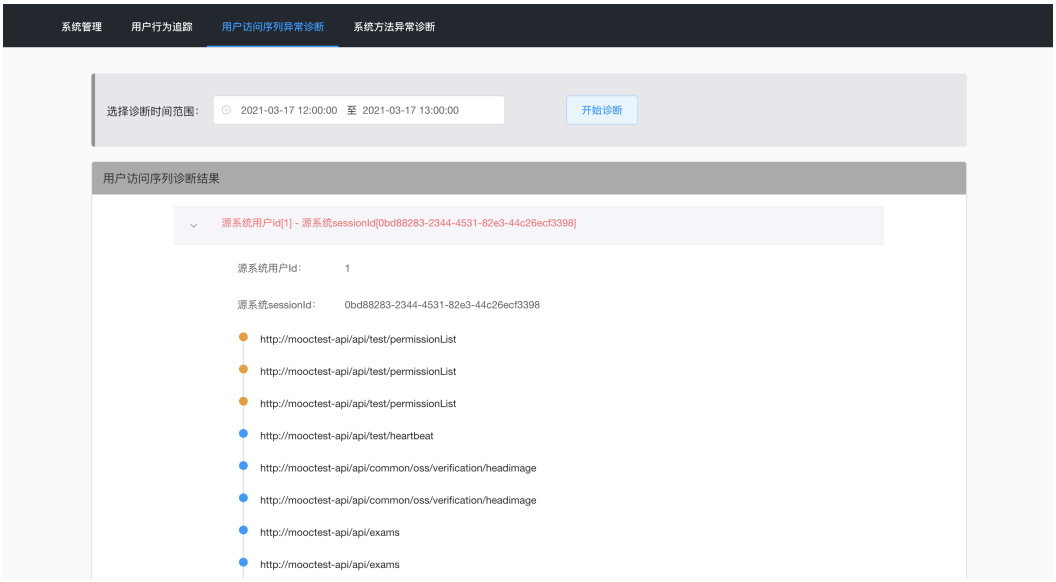

图 5-4: 用户访问序列诊断结果截图

5.5 本章小结

本章主要对基于业务监控数据的 Web 服务诊断系统进行功能测试、性能测试以及效果测试。其中对于本系统的功能测试主要根据第三章的需求分析进行测试，对分析得出的功能性需求进行逐一测试，测试结果表明本系统在功能上能够满足用户需求。本章的性能测试结合系统实际使用场景，从高并发量与大数据量两种场景对系统进行测试，通过性能测试得出本系统在性能上能够满足用户需求。本系统的效果测试是对慕测平台进行诊断，从诊断结果上看，本系统可以帮助开发人员发现系统中所存在的缺陷，提高开发效率。

第六章　总结与展望

6.1　总结

Web 服务由于其跨平台性与开放性，能够为世界各地所有安装浏览器的终端提供服务。日益增长的用户数量对 Web 服务的软件质量提出了较高要求。部分互联网企业缺少专业的测试团队，因此他们开发的 Web 服务质量难以保证，而质量低下的 Web 服务可能在上线后发生故障，也可能有不法分子通过 Web 服务中未进行权限控制的接口盗取企业机密，不及时定位并解决 Web 服务中的缺陷可能给企业造成无法弥补的经济损失。为了提高 Web 服务质量，目前众多互联网企业选择将自己开发的 Web 服务外包给测试服务提供商进行测试，然而由于第三方测试人员对该系统不够了解，需要花费时间进行学习后才能对该系统进行测试，因此无法在短时间内完成测试任务，从而导致 Web 服务上线时间推迟。另外，测试服务提供商的测试费用对于代码更新频繁的中小型互联网企业而言可能难以接受。因此，如何在保证开发成本与开发速度的前提下减少 Web 服务中的缺陷，成了各互联网企业重点关注的问题之一。

在此背景下，本论文设计并实现了基于业务监控数据的 Web 服务诊断系统，本系统将慕测平台作为源系统进行监控，通过 SkyWalking 与 Spring AOP 技术收集业务监控数据。本系统从历史正常访问序列中挖掘频繁序列模式，通过模式匹配分析当前用户的访问序列是否存在异常，并获取用户异常行为发生的时间、用户 id 以及未进行权限控制的接口等信息。本系统对所有的系统方法根据运行时间等参数进行聚类，并基于聚类结果定位性能低下的系统方法。本系统可以帮助开发人员或运维人员快速定位 Web 服务中的低性能方法与未进行权限控制的接口。

本文首先介绍了本系统研究的背景与意义，并介绍了 Web 服务监控与诊断等相关技术的研究现状。其次介绍了系统依赖的技术框架、算法理论知识以及相应的软件工具等，并通过与其他相似技术进行对比来阐述本系统中使用的相关技术、算法以及软件工具的优缺点与选用理由。之后借助系统用例图与用例描述对本系统的功能性需求与非功能性需求进行详细分析。为了更全面的描

述本系统的架构，本文采用“4+1”视图模型，从不同的视角对本系统架构进行阐述。之后对本系统进行功能模块的划分，包括业务监控数据收集模块、业务监控数据处理模块、用户访问序列诊断模块以及系统方法诊断模块等。之后通过架构图、核心类图、顺序图以及关键代码等方式介绍了系统的详细设计与实现，分别从静态与动态的不同角度分别对系统中划分的各个模块进行了详细描述。为了保证系统能够平稳运行，为用户提供高质量的服务，本文对系统的功能、性能以及效果等方面进行了测试，在测试期间，系统能够在高并发与大数据量的场景下稳定运行。本文最后展示了系统的实例截图，展示了本系统能够帮助开发人员或运维人员快速定位 Web 服务中低性能方法与未进行权限控制的接口，从而提高开发效率以及 Web 服务的服务质量。

6.2 展望

基于业务监控数据的 Web 服务监控系统的目标是帮助开发人员或运维人员快速定位 Web 服务中的低性能系统方法与未进行权限控制的接口，提高 Web 服务的开发效率与提升 Web 服务的稳定性，该系统还有诸多方面可以优化：

（1）Spring AOP 技术对系统存在侵入性，无法与需要监控的系统完全解耦。由于 SkyWalking 对所监控的系统侵入性较低，后续可以基于 SkyWalking 进行二次开发，通过 SkyWalking 进行埋点获取全部业务监控数据。

（2）本系统现阶段只能诊断基于 Spring 开发的 Web 服务。由于本系统使用 Spring AOP 技术收集 Web 服务的部分业务监控数据，因此本系统能诊断的 Web 服务受到开发框架限制。由于 SkyWalking 支持的开发框架较多，后续会基于 SkyWalking 进行二次开发，不再使用 Spring AOP 收集业务监控数据。

（3）聚类算法的参数需要人工设定。后续可以通过机器学习的方式自动调整聚类算法的参数，减少人工参与，提高诊断准确率。

（4）对历史诊断数据未充分使用。可以通过数据挖掘等方法对历史诊断数据进行充分使用，从历史数据中挖掘信息，反馈到当前的诊断结果中。

致　　谢

此刻，论文即将撰写完毕，我想对在项目开发和论文撰写过程中，对我提供过指导与帮助的人，致以真挚的感谢。

首先我要感谢我的指导老师陈振宇老师，我的毕业设计从初步想法到开题准备，再到设计思路，陈振宇老师都提出了很多指导意见，让我能够及时调整研究方向，帮助我避免方向上的错误。其次我要感谢黄勇老师，在我的技术选型以及项目开发过程中提出了很多具有建设性的意见，并且在我项目开发遇到问题时能帮助我快速找到解决方案。从大四开始我便进入 iSE 实验室平台组，进行慕测平台与群智平台等平台的开发与运维工作，在 iSE 实验室这两年多的学习是我人生中不可多得的宝贵经历。在 iSE 实验室平台组的开发过程同时也是我的学习过程，在陈振宇老师与黄勇老师的帮助下，我对软件开发有了新的认识与体会，从而对自己今后的职业规划也有了新的想法。

十分感谢南京大学软件学院，是南京大学软件学院的全体教职工的尽心尽责，才创造了一个学风优良、明德崇智、积极向上的学习环境，让我从一个对软件开发知之甚少的少年成长为一个能为社会所用的人，并找到心仪的工作。

由衷感谢 iSE 实验室的各位同学，感谢他们在项目开发过程中与学习生活中对我的帮助，他们给我提出很多重要意见，帮助我提高开发能力与培养良好学习习惯。特别是薛晓波同学与郭超同学，在我的毕业设计想法陷入窘境时，薛晓波同学帮助我理清思路，找到问题的解决方案；郭超同学在我的研究生阶段提供了大量的帮助和鼓励，在我松懈时能够及时督促我学习。感谢他们，他们的帮助使我的研究生生活更加充实。

特别感谢我的父母家人，父母的支持与理解是我的最坚强的后盾。感谢我的女朋友，她和我一起分担忧愁与分享喜悦。父母家人的关怀与爱是我在人生路上前进的源源动力。

最后，感谢在百忙之中拨冗审阅论文以及参与答辩的各位评审专家，向你们的专业与敬业致敬，向你们的负责与尽责致敬！

参考文献

[1] X. Fang, Y. Xie, A study on chinese people's everyday life information seeking: taking a comparison between web and weibo, Int. J. Serv. Technol. Manag. 25 (2) (2019) 179–196.
URL https://doi.org/10.1504/IJSTM.2019.098209

[2] M. Wäljas, K. Segerståhl, K. Väänänen-Vainio-Mattila, H. Oinas-Kukkonen, Cross-platform service user experience: a field study and an initial framework, in: M. de Sá, L. Carriço, N. Correia (Eds.), Proceedings of the 12th Conference on Human-Computer Interaction with Mobile Devices and Services, Mobile HCI 2010, Lisbon, Portugal, September 7-10, 2010, ACM International Conference Proceeding Series, ACM, 2010, pp. 219–228.
URL https://doi.org/10.1145/1851600.1851637

[3] T. Orehovacki, Evaluating the quality of social web applications using the LSP method, iJET 15 (10) (2020) 55–68.
URL https://www.online-journals.org/index.php/i-jet/article/view/13671

[4] R. P. L. Buse, W. Weimer, Learning a metric for code readability, IEEE Trans. Software Eng. 36 (4) (2010) 546–558.
URL https://doi.org/10.1109/TSE.2009.70

[5] O. Al-Debagy, P. Martinek, A metrics framework for evaluating microservices architecture designs, J. Web Eng. 19 (3-4) (2020) 341–370.
URL https://doi.org/10.13052/jwe1540-9589.19341

[6] F. Belli, A. T. Endo, M. Linschulte, A. da Silva Simão, A holistic approach to model-based testing of web service compositions, Softw. Pract. Exp. 44 (2) (2014) 201–234.
URL https://doi.org/10.1002/spe.2161

[7] J. Simmonds, S. Ben-David, M. Chechik, Monitoring and recovery for web service applications, Computing 95 (3) (2013) 223–267.
URL https://doi.org/10.1007/s00607-012-0215-y

[8] M. Bajer, Building an iot data hub with elasticsearch, logstash and kibana, in: I. Awan, F. Portela, M. Younas (Eds.), 5th International Conference on Future Internet of Things and Cloud Workshops, FiCloud Workshops 2017, Prague, Czech Republic, August 21-23, 2017, IEEE Computer Society, 2017, pp. 63–68.
URL https://doi.org/10.1109/FiCloudW.2017.101

[9] A. Muñoz, J. Gonzalez, A. Maña, A performance-oriented monitoring system for security properties in cloud computing applications, Comput. J. 55 (8) (2012) 979–994.
URL https://doi.org/10.1093/comjnl/bxs042

[10] C. Pahl, Containerization and the paas cloud, IEEE Cloud Comput. 2 (3) (2015) 24–31.
URL https://doi.org/10.1109/MCC.2015.51

[11] E. Allayiotis, Characterization of mobile web quality of experience using a non-intrusive, context-aware, mobile-to-cloud system approach, Ph.D. thesis, University of Central Lancashire, Preston, UK (2017).
URL http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.727097

[12] O. I. Sheluhin, A. Osin, Anomaly states monitoring of large-scale systems with intellectual analysis of system logs, in: 24th Conference of Open Innovations Association, FRUCT 2019, Moscow, Russia, April 8-12, 2019, IEEE, 2019, pp. 395–401.
URL https://doi.org/10.23919/FRUCT.2019.8711915

[13] V. Zamfir, M. Carabas, C. Carabas, N. Tapus, Systems monitoring and big data analysis using the elasticsearch system, in: 22nd International Conference on Control Systems and Computer Science, CSCS 2019, Bucharest, Romania, May 28-30, 2019, IEEE, 2019, pp. 188–193.
URL https://doi.org/10.1109/CSCS.2019.00039

[14] K. S. M. Chan, J. Bishop, J. Steyn, L. Baresi, S. Guinea, A fault taxonomy for web service composition, in: E. D. Nitto, M. Ripeanu (Eds.), Service-Oriented Computing - ICSOC 2007 Workshops, International Workshops, Vienna, Austria, September 17, 2007, Revised Selected Papers, Vol. 4907 of Lecture Notes in Computer Science, Springer, 2007, pp. 363–375.
URL https://doi.org/10.1007/978-3-540-93851-4_36

[15] 赵雪琴, 付媛媛, 云计算环境下大规模 Web 服务故障诊断技术研究, 计算机测量与控制 22 (9) (2014) 2760–2762.
URL https://kns.cnki.net/kcms/detail/detail.aspx?FileName=JZCK201409017&DbName=CJFQ2014

[16] L. Ardissono, L. Console, A. Goy, G. Petrone, C. Picardi, M. Segnan, D. T. Dupré, Enhancing web services with diagnostic capabilities, in: 2005 IEEE International Conference on Web Services (ICWS 2005), 11-15 July 2005, Orlando, FL, USA, IEEE Computer Society, 2005, pp. 182–191.
URL https://doi.org/10.1109/ECOWS.2005.12

[17] M. Belshe, R. Peon, M. Thomson, Hypertext transfer protocol version 2 (HTTP/2), RFC 7540 (2015) 1–96.
URL https://doi.org/10.17487/RFC7540

[18] M. Catelani, L. Ciani, V. L. Scarano, A. Bacioccola, Software automated testing: A solution to maximize the test plan coverage and to increase software reliability and quality in use, Comput. Stand. Interfaces 33 (2) (2011) 152–158.
URL https://doi.org/10.1016/j.csi.2010.06.006

[19] C. Gan, H. Li, Understanding the effects of gratifications on the continuance intention to use wechat in china: A perspective on uses and gratifications, Comput. Hum. Behav. 78 (2018) 306–315.
URL https://doi.org/10.1016/j.chb.2017.10.003

[20] F. Wuhib, R. Stadler, Distributed monitoring and resource management for large cloud environments, in: N. Agoulmine, C. Bartolini, T. Pfeifer, D. O'Sullivan (Eds.), Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management, IM 2011, Dublin, Ireland, 23-27 May 2011, IEEE, 2011,

pp. 970–975.
URL https://doi.org/10.1109/INM.2011.5990531

[21] K. Guntupally, R. Devarakonda, K. Kehoe, Spring boot based REST API to improve data quality report generation for big scientific data: ARM data center example, in: N. Abe, H. Liu, C. Pu, X. Hu, N. K. Ahmed, M. Qiao, Y. Song, D. Kossmann, B. Liu, K. Lee, J. Tang, J. He, J. S. Saltz (Eds.), IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018, IEEE, 2018, pp. 5328–5329.
URL https://doi.org/10.1109/BigData.2018.8621924

[22] V. Piantadosi, S. Scalabrino, R. Oliveto, Fixing of security vulnerabilities in open source projects: A case study of apache HTTP server and apache tomcat, in: 12th IEEE Conference on Software Testing, Validation and Verification, ICST 2019, Xi'an, China, April 22-27, 2019, IEEE, 2019, pp. 68–78.
URL https://doi.org/10.1109/ICST.2019.00017

[23] J. M. Félix, F. Ortin, Aspect-oriented programming to improve modularity of object-oriented applications, J. Softw. 9 (9) (2014) 2454–2460.
URL https://doi.org/10.4304/jsw.9.9.2454-2460

[24] J. I. Brachthäuser, Design and implementation of effect handlers for object-oriented programming languages, Ph.D. thesis, University of Tübingen, Germany (2020).
URL https://publikationen.uni-tuebingen.de/xmlui/handle/10900/102021/

[25] A. Przybylek, An empirical study on the impact of aspectj on software evolvability, Empir. Softw. Eng. 23 (4) (2018) 2018–2050.
URL https://doi.org/10.1007/s10664-017-9580-7

[26] S. A. Vidal, C. A. Marcos, A catalog of aspect refactorings for spring/aop, J. Univers. Comput. Sci. 19 (1) (2013) 157–182.
URL https://doi.org/10.3217/jucs-019-01-0157

[27] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, M. Hsu, Prefixspan: Mining sequential patterns by prefix-projected growth, in: D. Georgakopou-

los, A. Buchmann (Eds.), Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany, IEEE Computer Society, 2001, pp. 215–224.
URL https://doi.org/10.1109/ICDE.2001.914830

[28] A. Kemmar, Y. Lebbah, S. Loudni, P. Boizumault, T. Charnois, Prefix-projection global constraint and top-k approach for sequential pattern mining, Constraints An Int. J. 22 (2) (2017) 265–306.
URL https://doi.org/10.1007/s10601-016-9252-z

[29] A. R. Honarvar, T. Zaree, Frequent sequence pattern based activity recognition in smart environment, Intell. Decis. Technol. 12 (3) (2018) 349–357.
URL https://doi.org/10.3233/IDT-180340

[30] X. Shengsha, An apriori algorithm to improve teaching effectiveness, Int. J. Perform. Eng. 16 (5) (2020) 792–799.
URL https://doi.org/10.23940/ijpe.20.05.p13.792799

[31] X. Yang, X. Lin, X. Lin, Application of apriori and fp-growth algorithms in soft examination data analysis, J. Intell. Fuzzy Syst. 37 (1) (2019) 425–432.
URL https://doi.org/10.3233/JIFS-179097

[32] S. F. Galán, Comparative evaluation of region query strategies for DBSCAN clustering, Inf. Sci. 502 (2019) 76–90.
URL https://doi.org/10.1016/j.ins.2019.06.036

[33] A. Starczewski, P. Goetzen, M. J. Er, A new method for automatic determining of the DBSCAN parameters, J. Artif. Intell. Soft Comput. Res. 10 (3) (2020) 209–221.
URL https://doi.org/10.2478/jaiscr-2020-0014

[34] A. Al-Dhamari, R. Sudirman, N. H. Mahmood, Abnormal behavior detection using sparse representations through sequential generalization of k-means, Turkish J. Electr. Eng. Comput. Sci. 29 (1) (2021) 152–168.
URL https://doi.org/10.3906/elk-1904-187

[35] J. Chen, X. Lin, Q. Xuan, Y. Xiang, FGCH: a fast and grid based clustering algorithm for hybrid data stream, Appl. Intell. 49 (4) (2019) 1228–1244.
URL https://doi.org/10.1007/s10489-018-1324-x

[36] J. F. Cui, H. S. Chae, Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems, Inf. Softw. Technol. 53 (6) (2011) 601–614.
URL https://doi.org/10.1016/j.infsof.2011.01.006

[37] M. Christie, S. Marru, E. Abeysinghe, D. Upeksha, S. Pamidighantam, S. P. Adithela, E. Mathulla, A. Bisht, S. Rastogi, M. E. Pierce, An extensible django-based web portal for apache airavata, in: G. A. Jacobs, C. A. Stewart (Eds.), PEARC '20: Practice and Experience in Advanced Research Computing, Portland, OR, USA, July 27-31, 2020, ACM, 2020, pp. 160–167.
URL https://doi.org/10.1145/3311790.3396650

[38] A. J. Rafsanjani, S. Mirian-Hosseinabadi, Lightweight formalization and validation of ORM models, J. Log. Algebraic Methods Program. 84 (4) (2015) 534–549.
URL https://doi.org/10.1016/j.jlamp.2015.03.001

[39] H. Wang, F. Fang, Research on e-commerce supply chain design based on MVC model and virtual image technology, IEEE Access 8 (2020) 98295–98304.
URL https://doi.org/10.1109/ACCESS.2020.2996675

[40] J. E. T. Akinsola, A. S. Ogunbanwo, J. O. Okesola, I. J. Odun-Ayo, F. D. Ayegbusi, A. A. Adebiyi, Comparative analysis of software development life cycle models (SDLC), in: R. Silhavy (Ed.), Intelligent Algorithms in Software Engineering - Proceedings of the 9th Computer Science On-line Conference 2020, Volume 1, Vol. 1224 of Advances in Intelligent Systems and Computing, Springer, 2020, pp. 310–322.
URL https://doi.org/10.1007/978-3-030-51965-0_27

[41] M. de Benedictis, A. Lioy, Integrity verification of docker containers for a lightweight cloud environment, Future Gener. Comput. Syst. 97 (2019) 236–246.
URL https://doi.org/10.1016/j.future.2019.02.026

简历与科研成果

基本信息

孙加辉，男，汉族，1997 年 2 月出生，江苏省盐城市人。

教育背景

2019 年 9 月 — 2021 年 6 月 南京大学软件学院 硕士

2015 年 9 月 — 2019 年 6 月 武汉大学计算机学院 本科

攻读工程硕士学位期间完成的学术成果

1. Wen W, **Sun J**, Li Y, et al. Design and Implementation of Software Test Laboratory Based on Cloud Platform[C]//2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2019: 138-144.

攻读工程硕士学位期间参与的科研课题

1. 国家重点研发计划：信息产品及科技服务集成化众测服务平台研发与应用（2018YFB1403400），2019-2021