南京大学

研 究 生 毕 业 论 文

（申请工程硕士学位）

论 文 题 目 数据库 SQL 标准适配性自动化测试系统设计与实现

作 者 姓 名 李珍鸿

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 05 月 20 日

学　　号：**MF1932105**

论文答辩日期：**2021 年 05 月 20 日**

指 导 教 师：　　**（签字）**

Design and Implementation of Adaptive Automatic Test System for Database SQL Standard

By

Zhenhong Li

Supervised by

Associate Professor **Zhenyu Chen**

A Thesis
Submitted to the Software Institute
and the Graduate School
of Nanjing University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Engineering

Software Institute

May 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：数据库 SQL 标准适配性自动化测试系统设计与实现

工程硕士（软件工程领域）专业 2019 级硕士生姓名：李珍鸿

指导教师（姓名、职称）：陈振宇 教授

摘 要

近年来，随着国产计算机技术的发展，国产软件与国产操作系统井喷式的涌现，慢慢渗透入人们的日常生活中。由于国产软件与国产操作系统种类，版本众多，国产操作系统与国产软件的适配性的问题亟待解决。适配性测试作为保障软件与操作系统适配性的重要手段，发挥着不可替代的作用。数据库软件作为众多软件的底层支撑，为支撑软件的正常运行发挥了重要作用，数据库软件的适配性决定了许多软件的适配性，但传统的数据库与操作系统的适配性测试存在测试成本高、测试效率低、可重复性低的问题，并且标准测试集缺乏。

本文提出了数据库 SQL 标准适配性自动化测试技术，用于自动化生成数据库的适配性测试报告，设计并实现了数据库适配性自动化测试系统，旨在建立数据库软件自动化测试流程，为数据库开发人员提供参考价值。为了解决标准测试集缺乏的问题，测试数据库的功能正确性，本文根据 ANSI 的 SQL:1999 标准定义了包含 206 条的 SQL 测试集，涵盖了 18 种数据类型等，同时又根据 ANSI 的 SQL-92 标准定义了包含 187 条的 SQL 测试集，涵盖了 14 种数据类型等，两者都覆盖了多种查询条件。为了解决传统数据库测试人工在物理机器上部署操作系统与数据库，导致测试成本高，测试效率低的问题，本文采用了 Docker 容器技术自动化对操作系统与数据库进行部署。测试执行模块通过自动化执行 SQL 并收集结果的方式对部署好的数据库进行功能性的验证，生成测试报告。

本系统为 java web 项目，用户可以通过访问网站使用测试上传数据库或者数据库连接 URL 功能，通过服务运行自动化测评结果，并生成对应的适配性测试报告。对数据库开发者和测试者提供有效建议。

关键词：数据库测试，适配性测试，容器，系统

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Design and Implementation of Adaptive Automatic Test System for Database SQL Standard

SPECIALIZATION: Software Engineering

POSTGRADUATE: Zhenhong Li

MENTOR: Associate Professor Zhenyu Chen

Abstract

In recent years, with the development of domestic computer technology, domestic software and domestic operating systems have sprung up and gradually penetrated into people's daily lives. Due to the wide variety and versions of domestic software and domestic operating systems, the compatibility of domestic operating systems and domestic software needs to be resolved urgently. As an important means to ensure the adaptability of software and operating system, adaptability testing plays an irreplaceable role. As the underlying support of many software, database software plays an important role in supporting the normal running of the software. The compatibility of database software determines the compatibility of many software, but the traditional database and operating system compatibility test has High test costs, low test efficiency, low repeatability, and lack of standard test sets.

This thesis proposes an automated testing technology for database SQL standard compatibility, which is used to automatically generate database compatibility test reports, design and implement a database compatibility test system, aiming to establish an automated testing process for database software and provide database developers with Reference value. In order to solve the problem of the lack of standard test sets and test the functional correctness of the database, this article defines a SQL test set containing 206 items according to the ANSI SQL:1999 standard, covering 18 data types, and at the same time according to ANSI The SQL-92 standard defines a SQL test set containing 187 items, covering 14 data types, both of which cover a variety of query conditions.In order to solve the problem of high test cost and low test efficiency caused by manual deployment of operating systems and databases on physical machines in traditional database testing, this article uses Docker container technology to automate the

deployment of operating systems and databases.The test execution module verifies the functionality of the deployed database by automatically executing SQL and collecting results, and generates test reports.

The system is a java web project. Users can use it by visiting the website, test uploading the database or database connection URL, run the automated evaluation results through the service, and generate the corresponding adaptability test report.Provide effective advice to database developers and testers.

Keywords: Database testing, compatibility testing, container, system

目录

表 目 录

图 目 录

第一章　引言

1.1　选题的背景和意义

可移植性、易用性、维护性、效率、可靠性、功能性，作为 GB/T 16260《软件工程产品质量》纲要中定义的软件六大质量特性 [1]。作为软件的六大特性之一，可移植性无非是评价软件的一个重要指标，一个运行条件苛刻，可移植性低的软件，会使得用户群体收到限制，难以得到发展。因此，如何保障软件的可移植性成为了一个重要的问题，关系到了软件用户的广度和软件的发展。

随着国家信息创新的发展，涌现出了许多的国产操作系统和国产软件，特别是在如今以美国为首的西方国家的技术封锁的大环境下，国产操作系统和国产软件成为我们唯一的解决方案。国产化软件也在慢慢的进入我们的生活中，国产统信等系统正在慢慢替代着我们原本使用的 Windows，macos 等国外操作系统，国产 parlor-db 等数据库也在替代 oracle 等国外数据库，国产 WPS 等办公软件替代 word 等国外办公软件，丰富与便利了人们的生活。但是随着国产化软件的发展，软件和操作系统的适配性问题也随之而来。目前对于软件可移植性测试的技术并不成熟，关于这方面的研究比较少，主要的方式还是传统的手工测试的方式，这带来了以下几个问题：

成本大：现有的数据库适配性测试，直接在物理机器上部署操作系统，一个机器上只能运行一个系统，硬件成本高；并且无法自动化地部署，需要人工部署，人力成本高。

自动化程度低：现有的数据库与系统适配性测试，都是人工进行操作系统的安装部署，并在其上安装部署数据库，无法自动化的进行部署与测试。

可重复性低：由于是人工的方式进行测试，会受到测试人员主观因素的影响，难以复现。

如今，人工智能和大数据得到广泛发展，在医疗自动诊断，汽车自动驾驶等领域得到广泛应用，其中离不开数据的支持，数据越来越成为当今各大行业、公司发展的核心资源。随着互联网技术的发展，数据在计算机领域地位的提升，数据库软件的支撑作用也越来越重要。运行稳定且高效的数据库软件，已经成为许多计算机软件运行中不可或缺的部分。功能完备且运行良好的软件，需要其依赖的数据库运行流畅、响应快速、计算高效。要保证数据库软件运行的正确性，数据库测试是不可或缺的方法。[2]。数据库与操作系统的适配性作为软

件与操作系统适配性的基石，如果数据库与操作系统不兼容，则依赖该数据库的软件都不可在该操作系统上运行。为了实现自动化的测试数据库在某操作系统平台下的 SQL 标准适配性，减少人力成本和提高测试的效率，我们提出了数据库 SQL 标准适配性自动化测试技术，目的在于自动化测试国产数据库的适配性，生成有效的测试报告。

1.2　国内外研究现状及分析

随着计算机技术的不断发展，软件已经深入我们生活的每一个角落，与我们生活息息相关，国产软件作为国家核心竞争力的重要成分，发展国产软件与硬件已经成为国家级别的技术战略，特别是在如今以美国为首的西方国家的技术封锁之下，国产软件成为国家科学技术发展不可或缺的部分。随着国产软件与系统的发展，国产软件在国产系统的可移植性成为一个攻克技术难题，如何自动化评估软件的适配性成为一个解决难题的一个重要组成部分。数据库软件作为基础软件之一，在支撑软件的运行方面有着重要作用。

当前的数据库可移植性度量主要都是通过人工的方式进行，首先人工在物理机器上部署待适配操作系统，其次在操作系统上部署数据库，再针对数据库进行功能性验证，整个过程有以下几个缺点，成本高：一个物理机器只能部署一个操作系统，同时只能进行一个数据库的验证，硬件成本高，通过人工的方式部署，人力成本高，且对测试人员的能力有要求。自动化程度低，主要通过人工的方式进行，缺少自动化的流程。缺乏通用测试集，数据库软件种类众多，每个数据库都有自己的方言，缺少标准测试集。

1.2.1　信创软件

进入 21 世纪以后，计算机技术得到了快速发展，为我国计算机领域的发展提供了重要机遇。但是由于西方国家的技术封锁，我们国家需要在科学与技术关键的领域实现自主创新，掌握自主知识产权，使整个产业自主可控。目前，国产化信息产业发展迅速，在硬件芯片，数据库软件，操作系统软件，中间件，办公软件等多个领域都遍布国产化。操作系统作为计算机运行的基础，是计算机硬件资源和应用程序软件的纽带，具有重要作用。银河麒麟，Deepin，红旗 linux 等操作系统作为国产化操作系统的典型代表。2020 年 5 月，统信软件发布了统信 UOS 统一操作系统，全面支持了各架构 CPU，是国产化操作系统的重大突破。在操作系统国产化领域，涌现了以武汉达梦 DB、南大通用 GBase DB、神州通用 DB 等为代表的传统数据库，和阿里 Ocean Base、腾讯 DB 和华为 Gauss DB 等云数据库。但国产化软件存在版本繁多，互不兼容，难以集成的问题。软件开发过程中，

应用系统集成部署中的兼容性，稳定性，易用性问题亟待解决。推动信息产业国产化与自主生态建设，必须要解决上述问题，降低应用系统开发，集成，维护成本。

1.2.2　可移植性

研究者们对可移植性早就有了研究与定义，在国家技术监督局提出的软件工程产品质量书中对可移植性的定义为“一个系统，产品或组件可以从一个硬件、软件、运行或使用环境移植到另一种环境的有效性和效率的度量”[1]。软件可移植性的主要目标是“在新的环境可重用完整的现有程序功能之前，提高应用程序从当前运行环境移植到新的环境的能力”[3].“适应性”，“可安装性”和“可重新放置性”是可移植性的三个子特性。适应性是指软件单元可以有效，高效地适应目标环境的程度 [1]。可安装性是在目标环境中可以成功安装/卸载软件单元的程度。可重新放置性是指在相同环境中可以出于同一目的用另一个软件单元替换一个软件单元的程度。

由于计算环境的多样性不断增长，所有类型的软件在其一生中都应迁移到各种环境中 [3]。现阶段，尝试移植现有产品时，多数是通过临时技术获得的软件可移植性 [4]。在许多情况下，软件但也都基于软件结构和目标环境等因素来实现软件的可移植性，目标环境包括了例如，硬件，编程语言和操作系统等因素 [5]。为了在不同硬件环境-操作系统之间实现软件的可移植性，主要采用的是几种方法，例如重用二进制文件或源代码并使用可解释的代码，API 仿真器虚拟化或 Web 技术 [6]。然而，现阶段对软件可移植性的度量研究非常少，主要的软件适配性测试仍然通过人工的方式进行，缺少自动化流程，人力成本高并且测试水平参差不齐，受到测试人员的主观与能力的影响。软件度量标准定义了对软件及其过程（例如开发和维护）的关键属性进行定量测量的基础。存在两种软件指标：过程指标和产品指标 [7]。

过程指标度量了软件流程的成本，例如所使用的资源（例如金钱，设备和人日）或生产度量（例如调试的代码行）。产品指标标准度量软件质量的特征，它们可以确定大小，性能，复杂性或偶而具有其他属性，例如可维护性 [7]。根据研究结果表明，软件可移植性的研究是分散的，目前更关注开发与其他方面保持高度软件可移植性的方法/工具，并且关于如何衡量软件可移植性尚无共识 [8]。目前已经有不少的研究者研究检查在特定计算平台下的软件质量属性，Mojica 等 [9] 通过对 20 万个免费 Android 应用程序进行了实证研究，从而回顾了智能手机环境中的“可重用性”属性。Nayebi 等 [10] 通过研究 Apple App Store 上的 11 种应用程序，总结了智能手机环境中的“可用性”属性。

1.2.3　容器化技术

早在上个世纪五十年代，研究者们就开始对虚拟化技术的探索，1959 年，作为虚拟化技术最早的提出者，计算机科学家 ChristopherStrachey 首次提出了虚拟化的基本概念，发表了一篇学术报告，名为《在大型机器中共享时间》[11]。但是到了 70-80 年代, 随着集成电路的发展，计算机硬件资源成本越来越低，个人计算机越来越普及。为了降低硬件成本而设计的虚拟化技术变得不重要，很少人去研究与发展, 只是在一些行规的硬件资源共享中继续存在。但是随着近些年来计算机技术的发展出现了一些问题，首先，计算机软硬件变得越来越复杂，管理成本增大，其次，为了适应用户复杂与多样化的需求，服务化成为主流。为了解决这些问题，可以屏蔽复杂的计算机物理资源，为软件的快速部署带来极大便利的虚拟化技术又变得热门 [12]，计算机体系结构领域的研究者们又重新将目光放到计算虚拟化技术方向。为了解决软件在不同系统的可移植性问题，容器技术被研究者们提出，产生了 Linux Container。LXC 技术可以在操作系统层上划分出许多虚拟的运行环境，即为容器，一个容器对应着一个可运行进程的虚拟环境 [13]。为了更有效地解决资源使用需求可能存在的冲突，容器技术将原本由系统管理的计算机硬件资源高效的分配到独立的组。这样的技术既不需要编译，也不需要模拟指令集，比其他虚拟技术更高效 [14]。

虚拟化技术可解耦下层计算机设备与上层操作系统、软件程序, 使得计算机资源得到最大化利用，也提高了资源利用的灵活性，因此可以用于解决适配性技术成本高，软件与系统组合复杂的问题。随着国产软件地发展，国产数据库与国产系统发挥的作用也越来越大，数据库与系统作为软件的基础支撑部分，在软件运行过程中扮演了重要的角色，然而随着国产数据库与国产系统种类，版本的不断壮大，国产数据库与国产软件是否适配的问题也日益严重。快速变化的环境与日新月异的技术手段使得软件对数据库与操作系统的质量、功能要求更为严格。软件对操作系统与数据库的依赖越来越高，从而需要及时有效评估数据库的 SQL 适配性。在软件开发领域，作为保障软件的正确性与安全性的重要手段，软件测试技术具有重要的作用，[15]。基于云计算测试是一种新型的测试模式，其具低成本、灵活性高、可靠性高等特点 [16]。基于云计算的软件测试屏蔽了复杂的物理测试环境，可以降低部署的人力物力成本，提高了测试效率 [17], 可以为我们测试数据库与软件适配性提供一个新的思路。

1.3　本文主要工作

本文的研究在国家“信息创新”的背景下进行，旨在为国产数据库的 SQL 标准适配性测试提供帮助，推动国产数据库的推广与使用。目前数据库 SQL 标

准的适配性测试主要通过人工直接在物理硬件上部署操作系统，然后在操作系统上部署数据库，然后执行 SQL 测试的方式进行，流程复杂，硬件成本大，缺少自动化的流程，无法自动化生成测试报告。容器技术具有成本低、标准化、运行环境可移植，为数据库的 SQL 标准适配性测试提供了新的方向。通过采用容器化技术进行部署操作系统与安装数据库，可以使得整个流程自动化地进行，同时减少了物理硬件成本，为了解决流程中可能出错的问题，本系统在自动化流程中增加监控与日志分析。同时现有的 SQL 标准缺乏系统的测试集，难以对众多种类的数据库进行统一的测试。

基于上述分析，本文的工作重点在于分析整理 SQL 测试集，基于 SQL 测试集进行数据库的 SQL 标准适配性测试，并生成适配性测试报告。首先，为了解决测试集缺乏的问题，本文根据数据库中比较通用的 ANSI 的 SQL-92 和 SQL:1999 标准，分析整理提出了对应的 SQL 测试集。本文将从上传数据库测试与数据库 URL 测试两个角度来拆分测试任务，对于上传数据库软件测试，为了解决传统数据库测试成本高，自动化程度低的问题，通过虚拟化容器技术，在容器中自动化部署操作系统并在其上安装运行数据库，进行数据库的 SQL 标准适配性测试，而对于数据库 URL 测试，则通过 Java 动态加载类的方式，动态注册连接驱动，通过 JDBC 的方式连接数据库，进行数据库的 SQL 标准适配性测试。通过实时监控与日志分析技术对测试过程中的运行状态与执行 SQL 的结果进行分析，从而生成适配性测试报告。为了防止出现意外情况，在出现偶发错误，如网络错误等原因，提供了人工对测试任务进行重试的功能。根据上述需求拆分用例，实现了一个 java web 应用，并对系统进行了测试，保证了系统的正确运行。

1.4　本文的组织结构

第一章引言，阐述了数据库 SQL 标准适配性自动化测试技术的项目背景及意义、国内外相关的研究现状、本文的主要工作。

第二章技术综述，介绍了实现本系统所需要的技术与工具，包括微服务框架、容器技术，数据库语言等。

第三章需求分析与概要设计，将系统需求拆分成功能用例，总共分为 9 个功能用例，通过用例表描述了功能用例。

第四章详细设计及实现，详细描述了系统的四大主要功能模块的设计，展示了类图，进程图等，并通过关键代码描述了模块的详细实现。

第五章测试与分析，为了保证系统的正确运行，对系统的功能设计了测试用例，并进行了单元测试与性能测试。

第六章总结与展望，对系统的整体实现进行了总结，并提出了系统现存的不足与可能的解决方案。

第二章　技术综述

2.1　数据库标准

2.1.1　SQL 语言

作为在编程领域中的一种领域特定语言，SQL(Structured Query Language) 是为管理关系数据流管理系统（RDSMS）中的流处理，或者管理关系数据库管理系统（RDBMS）中的数据而设计的 [18]。SQL 语言在处理合并实体与变量关系的数据（即结构化数据）中特别高效。SQL 解决了 ISAM 或 VSAM 等旧的 API 中存在的缺点，首先，它提出了通过单个命令就可以访问多个数据记录的理念；其次，消除了需要定义获取数据方式的问题，例如定义用不用索引。

SQL 由基于元组关系演算和关系代数模型设计，可以大概归类为以下四种子语言：DQL，数据查询语言；DDL 数据定义语言；DCL 数据控制语言和 DML 数据操作语言 [19]。SQL 包括数据结构定义（schema 创建和修改），数据记录操作（插入，更新和删除），数据记录查询和控制访问权限等操作。

1970 年，Edgar F. Codd 提出了自己的关系模型，发表了具有影响力的一篇论文《大型共享数据库的数据关系模型》[20]。SQL 使用了 Edgar F. Codd 的关系模型，虽然并未完全符合，但 SQL 成为成为数据库语言的标准并被广泛使用。

2.1.2　ANSI SQL 标准

1986 年，ANSI 美国国家标准协会出版了 SQL-86 标准，这是首个标准化的 SQL 语言的版本，将 SQL 指定为关系数据库的标准语言。SQL-86 标准作为第一代标准，起到了重要的作用。1989 年，ASNI 出版了 SQL-89 标准，将完整性约束加入标准。

1992 年，ANSI 发布 ANSI SQL-92 标准。作为 SQL 标准非常重要的一个版本，SQL-92 完善了 SQL 标准，加入了 DATE、TIME 等数据类型，加入了新的标量运算，新的集合运算，事务隔离级别，用户权限，动态 SQL 语句，可移动的游标等等。

1999 年，ANSI 发布了 SQL:1999 标准，增加了新的数据类型布尔类型，增加了大量的新功能，例如正则表达式匹配，OLAP 功能，过程或控制流语句，触发器等等。SQL-86 标准的正文只有几十页，到了 SQL-92 标准时大约有 500 页，而 SQL:1999 标准的征文则超过了 1000 页，从 SQL:1999 标准开始，SQL 标准变得

非常丰富了，内容覆盖了许多方面。由于 ISO 标准习惯上采用冒号，从 SQL:1999 开始，ANSI 用冒号（:）替换了短横线（-）作为标准简称的分割符。与此同时标准的命名遇到了千年虫的问题，所以标准制定的年份标识改用四位数字。

2003 年，2006 年，2008 年，2011 年，ANSI 分别发布了 SQL:2003，SQL:2006，SQL:2008，SQL:2011，除了删除了 BIT 和 BIT VARYING 数据类型，新的数据类型 DECFLOAT，几乎没有对数据类型进行修改，但是新增了许多 SQL 操作与支持，SQL 语言变得复杂和完备。

2.2　Spring 框架

为了解决 EJB 的复杂繁重，Spring 被开发与提出，是一个 Java 语言的企业应用开发框架，它轻量级并且开源，提供了功能齐全的 MVC 模块，可用于开发 Java Web 应用程序 [21]。Spring 两个核心功能分别是依赖注入（DI, Dependency Injection）：在运行需要时注入依赖对象，而非在编译时就建立对象的依赖，动态管理 Java Bean 对象之间的复杂依赖关系，将对象之间的依赖解耦，提高了代码可维护性；面向切面编程（AOP, Aspect Oriented Programming）：将公共服务代码（如日志，安全，事务等）独立成切面代码，而不在业务代码中实现，从而实现功能代码高复用，功能高内聚，具有良好的隔离性。Spring 框架对功能进行模块化，使用者可以按需配置模块，减少多余的模块引用。为了使得使程序代码对框架依赖最小，Spring 框架采用了非侵入式设计。并且 Spring 提供了了企业应用开发的许多公共的问题的解决方案，是极其优秀的全栈集成框架。

2.2.1　Spring Boot 框架

为了简化 Spring 项目的搭建以及开发，Pivotal 团队开发了 Spring Boot，并将其开源，提高开发者效率 [22]。Spring Boot 框架根据开发者添加的 jar 依赖自动配置 Spring 框架，并提供了如指标，健康检查等生产就绪型功能。此外，Spring Boot 框架内嵌了 Tomcat 容器，不再需要部署 war 文件，可通过 Jar 包直接启动。对于 RESTful 风格的微服务架构的开发者，Spring Boot 框架可以提高他们的效率。Spring Boot 精简 Spring 配置，解决了 Spring 框架与其他框架整合时配置繁重的问题。

2.3　容器技术

Linux Container（简称 LXC）是一个开源容器平台，提供了一组工具、模板、库和语言绑定。LXC 采用高效的命令行界面模式，使得容器启动所需时间降低，改善容器启动时的用户体验。LXC 在操作系统层级实现了虚拟化环境，可在许

多基于 Linux 的系统上安装，是轻量级的虚拟化技术 [23]。与 KVM、XEN 等平台虚拟化技术不同，LXC 实现隔离的机制简单而高效，实现了可移植与扩展的虚拟化机制，支持在单个物理机器上运行数千个容器 [12]。具体来说，LXC 依靠 Cgroups 和 Namespace 两个机制实现了隔离。为了实现管理和控制计算机资源，研究者们设计与实现了 Cgroups 系统 [24]。LXC 依靠 Cgroups 按层结构划分、聚类进程，由 Linux 内核实现并提供 [23]。“Cgroups 通过一个结构体 css_set 来存储指向进程所关联分组的资源信息，一个 css_set 结构关联着一个被加入分组的进程，在结构体中存储着指向资源控制子系统地址的指针，资源控制子系统包括 CPU、Memory 等” [25]。在 Namespace 机制下，每个分组有单独的命名空间，是在操作系统层级实现虚拟化的基础 [26]。Namespace 机制将计算机资源分成不同的部分，一个命名空间看成一个容器，一个分组内的进程共享对应命名空间内的资源，不同容器里面的进程互不干扰 [27]。

容器技术的优点：

敏捷部署：与虚拟机技术相比，容器技术创建的效率更高，容器轻量级的脚本可以提高了性能，并降低了使用资源。

提高效率：容器消除了跨服务依赖和冲突。作为独立的进程，容器之间无需考虑同步问题，可以独立升级，提高了开发者的效率。

版本控制：每个镜像都定义了版本信息并实现版本控制，可以比较版本之间的差异，查看不同版本的容器等等。

可复用：镜像封装了操作系统以及应用依赖，可以满足运行软件所必需，因此镜像可以在不同环境中可移植。

标准化：基于开放标准开发，使得容器是平台通用的，在所有主流操作系统平台，例如 linux，mac os，windows 等都可以运行。

安全：通过资源分配技术使得容器之间的资源是相互隔离的，单个容器的变化对其他容器而言是不可见的，不会互相影响。

2.3.1　Docker 容器技术

Docker [28] 是一种高级容器引擎技术，由 PaaS 提供商 dotCloud 开发，并遵从了 Apache2.0 开源协议 [29]。Docker 可以管理计算机物理硬件资源，将硬件资源分配到容器中，并将应用软件和物力硬件资源隔离。Docker 减少从编写代码到打包部署软件的所需时间，更高效率地打包代码、集成测试以及部署软件。Docker 集成了一整套标准的工作流程与工具，来帮助软件开发者管理和部署应用。Docker 容器内可运行大多数主流软件，并且容器之间相互隔离，因此在一个物理计算机上同时运行多个容器是安全可靠的，并且无需像 hypervisor 一样需

要资源消耗，节约了开发者的物理硬件资源。

2.3.2 Docker-java 库

Docker-java 是开源的通过 java 连接调用 docker api 的库。提供了大量的 docker api 的封装，交互的一系列接口，方便调用。

2.4 React 技术

为满足前端 MVC 架构的需求，Facebook 开发了 React，一个用于开发 web 前端界面的 Javascript 开源库 [30]。由于采用了声明式设计，React 可以采用纯函数的方式，使得前端代码具有高可靠性。为了提高开发效率，React 提供了封装构造组件的方式，提高了代码可复用性。为了解决传统前端框架操作 DOM 低效的问题，React 提出了 Virutal DOM，这是一个管理 DOM 的轻量级框架，提高了 web 页面渲染的效率。结合支持函数式编程的简洁的 Flux 实现 Redux，React 前端页面与数据操作逻辑分离，使得项目整体结构更为简洁且直观。

2.5 Google cAdvisor

为了监控 Docker 容器运行时所消耗资源和性能指标，Google 开发了监控容器的工具 cAdvisor（Container Advisor）。cAdvisor 是一个容器，他像守护程序一样运行，可以对正在运行的容器的运行信息进行采集，聚类，处理和导出等操作。cAdvisor 对部署机器以及容器资源实时采集性能数据，包括内存使用量、网络吞吐量、CPU 占用时间及文件系统输入输出情况。对于每个容器，它保留资源隔离参数，历史资源消耗和网络使用统计信息。此数据可以按照单个容器或者整个部署机器的粒度导出。

2.6 Promethues

Prometheus 是一个持续获取监控系统数据的开源工具包，从 2012 年开始在 SoundCloud 上开发与发布，并被众多机构和公司所接受。许多编程人员人员都使用与关注该项目，开源社区表现活跃。作为一个多维数据模型，Prometheus 包含许多时间序列数据与度量名称，每一条时间序列数据包括了键/值对信息。Prometheus 获取监控目标的方式有服务发现和静态配置两种，然后通过实时的 http 请求不断地获取时间序列数据。Promethues 支持 PromQL 查询语言对数据进行查询与分析，极大了方便了开发者们分析与使用数据。

2.7　本章小结

本章主要介绍了数据库 SQL 标准适配性自动化测试系统的相关技术框架。首先介绍了系统相关的数据库标准，然后介绍了然后介绍了本系统为了部署操作系统测试床而采用的 docker 技术以及用于调用 docker api 管理控制 docker 容器的 Docker-java 库，然后介绍了后端实现的 Spring 相关技术，和前端实现所用的 React 框架。本章主要目的介绍了系统实现相关的理论研究和技术基础。

第三章　　需求分析与概要设计

3.1　总体规划

本系统主要用户是数据库的测试人员和开发人员。如图3.1所示的数据库SQL 标准适配性自动化测试系统流程图, 对于输入数据库，选择系统列表进行测试的场景，系统首先启动所选择操作系统的 docker 容器，将用户输入的数据库文件分发到各个容器中。系统对数据库文件进行解压，安装，然后运行数据库测试脚本，并收集运行的输出日志，并对日志进行分析。同时也支持用户通过指定数据库 URL 的方式测试数据库的适配性，此项测试需要用户上传数据库连接驱动，现阶段只支持 java 语言的驱动，输入数据库登录用户名，密码。在新建 URL 测试任务之后，系统通过 JDBC 的方式连接数据库，并对数据库进行测试集的验证。

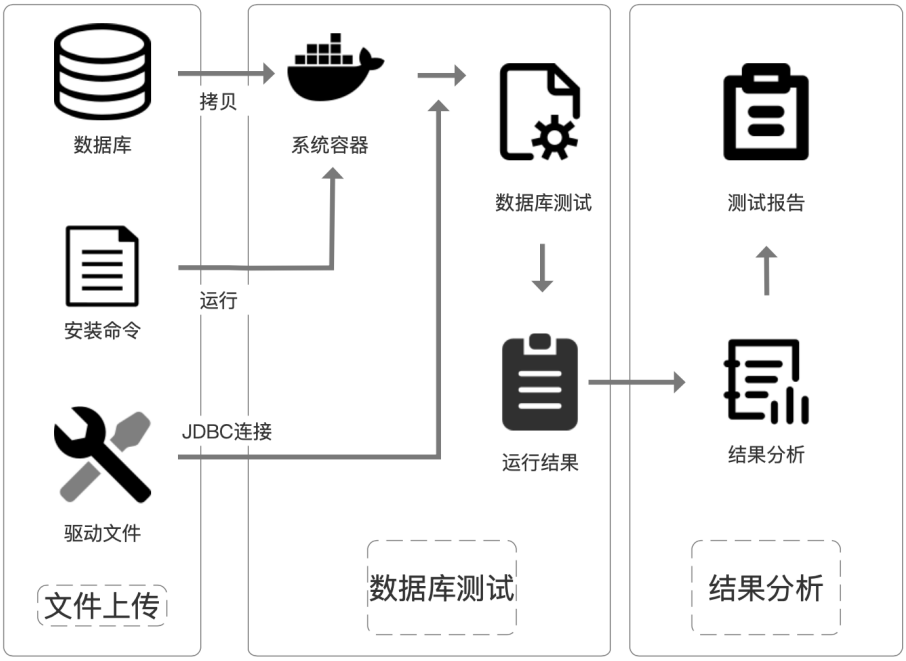

图 3.1: 系统结构图

本系统总体由权限管理模块，文件管理模块，测试任务管理模块，测试执

行模块四部分组成：权限管理模块负责对用户的请求进行合法性检验，对请求进行转发，阻止非法请求，保证安全性；文件上传模块负责实现用户上传的数据库文件、数据库安装命令文件和连接驱动文件资源管理功能，包括文件查看、删除、下载等功能；测试管理模块实现上传数据库测试任务和数据库 URL 测试任务管理功能，包括新建、查询、删除测试任务等功能；测试执行模块实现启动对应的操作系统 docker 镜像，并在容器中安装、运行数据库，并运行数据库测试。同时负责运行 SQL 的结果进行分析的工作，并生成测试报告。本章将对系统需求进行拆分，描述功能用例。

3.2　系统需求分析

3.2.1　系统功能需求

数据库 SQL 标准适配性自动化测试技术测试系统，对于上传数据库测试，根据用户所选择的操作系统与数据库的匹配对，运行指定的操作系统容器镜像，并在容器中部署，运行数据库，然后对容器中运行的数据库进行 SQL 标准测试；对于数据库 URL 测试，通过动态加载驱动类的方式，连接数据库 URL，然后对数据库进行 SQL 标准测试。两类任务的测试集是根据 SQL:92 和 SQL-1999 标准收集的 SQL 测试集，从而得出数据库 SQL 标准适配性测试报告。

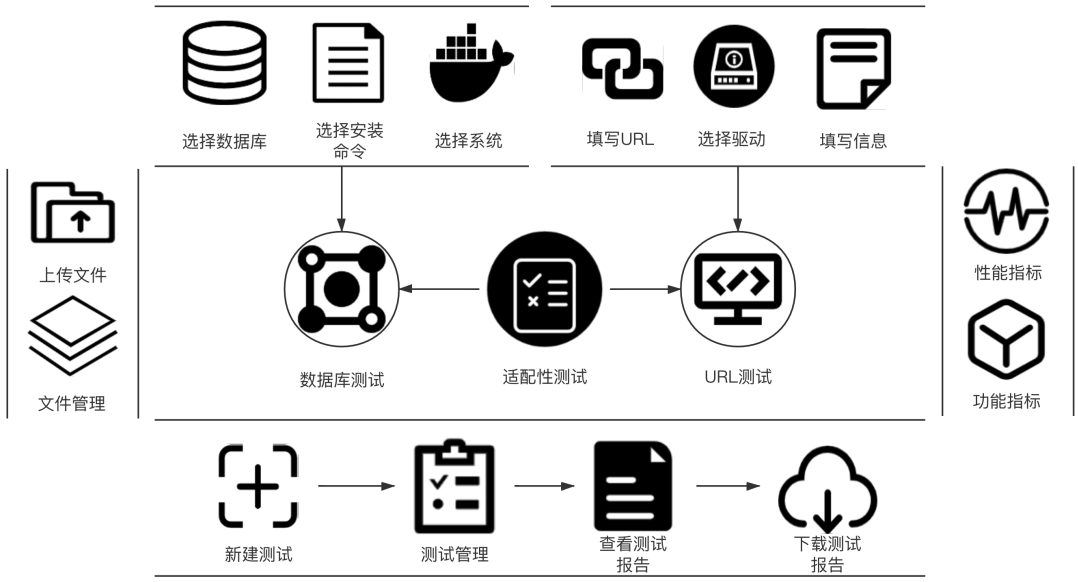

图 3.2: 系统流程图

本系统通过数据库功能测试来测试数据库与系统的适配性。本系统支持对用户上传的系统或者数据库进行测试，适配性的评估主要是根据整理的 SQL-92 与 SQL:1999 标准测试集进行功能性测试。表3.1显示了系统的功能性需求列表。

表 3.1: 功能性需求列表

ID	名称	描述
1	上传数据库	用户访问网站，然后上传数据库文件，在用户上传之后，后台会对数据库文件进行存储。本系统支持 gzip 压缩或者 xz 算法压缩的 tar 文件。
2	上传安装命令	用户访问网站，然后上传安装命令文件，在用户上传之后，后台会对安装文件进行存储。本系统支持文本文件。
3	上传连接驱动文件	用户访问网站，然后上传连接驱动文件，在用户上传之后，后台会对数据库文件进行存储。本系统支持实现 java JDBC 的连接驱动
4	文件管理	用户访问网站，对上传文件进行管理，在文件上传之后，用户可以在文件列表中对文件进行查看信息，删除等操作
5	新建上传数据库测试	系统支持对上传数据库的测评，启动用户选择系统列表的容器作为测试床，将用户上传的数据库运行到对应的测试系统 docker 容器上，并进行测试集的验证。
6	新建数据库 URL 测试	系统支持通过 URL 连接的方式对已安装数据库测评，，用户选择连接驱动文件，通过 JAVA 动态加载类，注册驱动文件，通过 JDBC 连接数据库，并进行 SQL 测试集的验证。
7	管理测试任务	系统可以通过测试任务管理界面对测试任务进行管理，包括查看详情，删除等操作。
8	管理测试报告	在测试任务运行结束之后，系统会生成数据库 SQL 标准适配性测试报告，用户可以查看与下载测试报告。

首先，本系统支持对上传数据库 SQL 标准的适配性进行评估，本系统对操作系统容器中运行的数据库进行 SQl 测试集的验证，得到运行的日志，通过分析日志，得出数据库在该操作系统上的功能运行情况，从而得出适配性报告。其次，系统支持通过指定 URL 的方式测试已经部署好的数据库，通过 java 动态加载类的方式注册连接驱动类，并通过 JDBC 的连接数据库，运行测试数据集。综

上所述，系统核心业务功能需求可以总结为以下两点：

上传功能：本系统支持用户上传数据库文件，或者上传连接驱动文件。对于上传数据库文件，系统将数据库文件传入远程 docker 服务器中对应的操作系统容器中，并将其解压，安装，初始化，为测试提供测试床环境。

测试功能：包含数据库适配测试和数据库 URL 测试两种测试类型，用户发起数据库适配测试，可选择已上传数据库文件，选择数据库安装命令文件，并指定需要适配的操作系统列表，运行数据库与所选系统的适配测试。这个功能会生成对应的数据库功能测试日志。在得到测试日志之后，系统对测试生成的日志进行分析，通过关键词匹配算法分析，从而得出数据库功能运行是否正常，生成数据库与系统的适配报告。用户发起数据库 URL 测试，需要填写待测试的数据库 URL、用户名、密码，选择已上传连接驱动文件，运行数据库 SQL 标准适配性测试，这个功能通过 JDBC 的方式连接数据库，并运行测试集，主要通过运行 SQL 返回的消息判断测试执行的结果，从而生成数据库 SQL 标准适配性测试报告。

3.2.2　非功能需求

表3.2描述了本系统的非功能性需求列表，为了保证本系统的可并发访问与访问速度，对本系统提出了非功能性需求，包括响应时间，并发量，可扩展性等。

表 3.2: 非功能需求列表

类型	非功能需求描述
响应时间	页面之间跳转的响应时间低于 1s
	在点击查看测试报告后，测试报告生成时间低于 2.5s
并发量	对接口的并发访问量，至少支持 100 个
可靠性	对系统运行过程中的日志信息记录，且保证系统功能的相对独立
安全性	本系统需要用户登录才可使用，对用户请求进行合法性检验，防止非法访问
可扩展性	对系统功能模块进行拆分，前后端分离，提高可拓展性

3.2.3　用例设计

本系统的主要为数据库开发与测试人员提供建议，进行数据库的适配性测试任务，但是分为两种测试需求，一种是对为安装数据库软件进行测试，需要本系统安装与部署数据库，然后进行测试；一种是对已安装好的数据库，通过 URL

连接数据库的方式进行测试。将系统的需求拆分成功能用例，用例图如图3.2.3所示。本系统主要有 9 个功能用例，分别是上传数据库、上传安装命令、上传连接驱动文件、上传文件管理、新建数据库测试任务、新建数据库 URL 测试任务、测试任务管理、测试报告查看、测试报告导出。

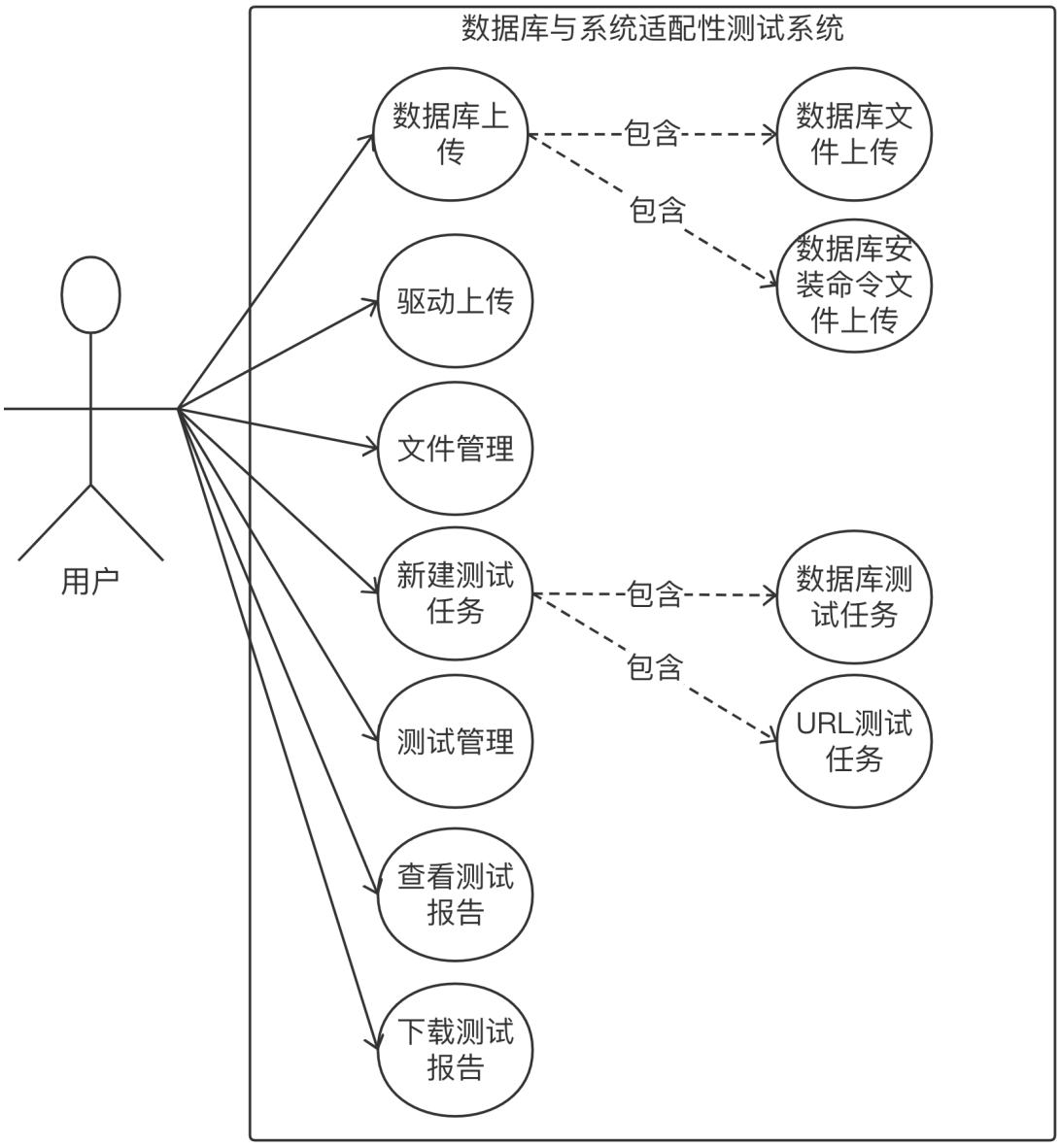

表 3.3: 数据库上传用例描述

ID	01	用例名	数据库上传
描述	系统用户上传数据库文件		
参与者	数据库测试人员		
触发条件	在测试上传数据库之前，系统用户需要上传数据库文件		
前置条件	系统用户点击上传数据库按钮，进入对应界面		
后置条件	前端界面反馈文件上传的结果		
优先级	高		
正常流程	1. 用户点击上传数据库按钮，进入对应界面 2. 选择数据库文件 3. 填写其他信息，如运行命令头等 4. 确认上传，前端向后端发送上传数据库请求		
扩展流程	2a. 后端文件保存失败，前端提示保存失败信息 3a. 未填写运行命令头，前端提示必须填写 4a. 取消上传文件，页面跳转为首页		
特殊流程	1. 若已存在重名文件，则提示用户，并由用户选择是否覆盖，避免文件冲突 2. 运行数据头对应执行 SQL 时附带的命令头，必须正确填写，否则在数据库测试运行语句时会出错		

表3.3对上传数据库用例进行了描述，在用户需要测试上传数据库时，首先需要上传数据库文件，需要选择上传的数据库文件，输入备注，输入数据库运行命令头，然后点击确认上传。数据库文件的格式为压缩文件格式，需要搭配后面的安装命令实现在容器中安装与启动数据库，数据库命令头则为功能测试中执行 sql 不可或缺的部分，必须在上传数据库时指定。确认上传后，后台会进行数据库文件的持久化，并保存数据库与安装命令和运行命令头的匹配数据库文件的上传状态会在点击确认之后即时反馈。

表3.4对上传安装命令用例进行了描述，在用户需要测试上传数据库时，首先需要上传安装命令，选择上传的数据库安装命令文件，输入备注，然后点击确认上传。安装命令作为之后在操作系统容器中安装与启动数据库的手段，应为一个文本文件，包含了解压，安装，初始化，启动数据库等操作，每一行为一个单独的命令，与数据库一般是一一对应的关系。确认上传后，后台会进行安装命令文件的保存，上传状态会在确认之后即时反馈。

表 3.4: 安装命令上传用例描述

ID	02	用例名	安装命令上传
描述	系统用户上传安装命令文件		
参与者	数据库测试人员		
触发条件	在测试上传数据库前，系统用户需要上传安装命令文件		
前置条件	用户点击上传安装命令按钮，进入对应界面		
后置条件	前端界面反馈文件上传的结果		
优先级	高		
正常流程	1. 用户点击上传安装命令按钮，进入对应界面 2. 用户选择安装命令文件 3. 用户填写其他信息，如备注等 4. 用户确认上传，前端界面向后端发送上传安装命令文件请求		
扩展流程	2a. 后端文件保存失败，前端提示保存失败信息 4a. 取消上传，页面跳转为首页		
特殊需求	1. 若已存在重名文件，则提示用户，并由用户选择是否覆盖，避免文件冲突		

连接驱动文件上传测试用例如表3.5所示，在用户需要测试数据库 URL 时，首先需要上传连接驱动文件。连接驱动文件作为 URL 测试任务的重要组成成分，是 JDBC 连接数据库 URL 并执行 SQL 测试集的工具，是不可或缺的，用户在测试数据库 URL 之前，必须上传对应的连接驱动文件，才能正确的新建对应的测试任务，并且上传的连接驱动文件必须为实现了 java JDBC 接口的，否则会在测试时出现错误。在上传驱动文件时，还需指定动态加载的类名，方便本系统在注册驱动动态加载类的时候指定类名，避免错误。确认连接驱动文件上传后，后台会进行连接驱动文件的保存，上传状态会在上传之后即使反馈。

文件管理的用例描述如表3.6所示，数据库文件，安装命令文件和连接驱动文件作为测试任务的重要组成部分，对于用户新建测试任务是不可或缺的，本系统提供了文件管理功能，为用户提供文件信息查看，或者过期文件清理，以及对已上传文件下载等功能。用户可通过文件名、上传时间等条件对列表进行过滤，从而方便的查看到用户所需要的文件记录，并支持对上传文件信息的删除操作，并允许用户下载数据库、安装命令或连接驱动等已上传文件。

表 3.5: 上传连接驱动文件用例描述

ID	03	用例名	连接驱动文件上传
描述	系统用户进行连接驱动文件上传		
参与者	系统用户		
触发条件	在测试数据库 URL 之前，系统用户需要上传连接驱动		
前置条件	用户点击上传连接驱动按钮，进入对应界面		
后置条件	前端界面反馈文件上传的结果		
优先级	高		
正常流程	1. 用户点击上传连接驱动按钮，进入对应界面 2. 选择连接驱动文件 3. 填写其他信息，如类名等 4. 确认上传，前端界面向后端发送上传连接驱动文件请求		
扩展流程	2a. 后端文件保存失败，前端提示保存失败信息 3a. 未填写类名，前端提示类名必须填写 4a. 取消上传，页面跳转到首页		
特殊流程	1. 若已存在重名文件，则提示用户，并由用户选择是否覆盖，避免文件冲突 2. 类名对应动态加载注册驱动时的 java 类名，必须正确填写，否则在连接数据库 URL 时会出错		

新建上传数据库测试任务用例描述如表3.7所示。用户从已上传数据库文件库列表中选择数据库文件，从已上传安装命令文件库列表中选择安装命令文件，从操作系统列表中选择部署数据库软件的操作系统测试床，确认新建上传数据库的测试任务，后台会首先保存任务数据，并不直接执行。在测试任务执行过程中，首先启动所选择操作系统的容器，然后获取任务对应数据库文件和安装命令文件，在 docker 服务器对应的操作系统容器中对数据库进行初始化、安装、运行数据库，然后分别进行 SQL-92 与 SQL-99 测试集的验证等流程，生成数据库功能测试日志，根据 SQL 执行过程中的返回结果，分析 SQL 执行情况，并将最终计算的适配报告到数据库中。上传数据库测试任务提交后，后台会通过定时任务异步地获取与执行测试任务。测试任务是后台异步的执行与更新状态，用户可在测试历史管理界面中查看对应数据库测试任务的测试状态，如果测试已完成，则可以查看与下载对应的测试报告，也可下载执行过程中生成的日志。

表 3.6: 已上传文件管理用例描述

ID	04	用例名	已上传文件管理
描述	系统用户对已上传文件进行管理		
参与者	系统用户		
触发条件	系统用户需要管理数据库文件、安装命令文件或 连接驱动文件		
前置条件	系统用户点击文件管理，进入对应界面		
后置条件	前端界面显示文件列表，并对用户之后的操作进行处理		
优先级	高		
正常流程	1. 用户点击文件管理按钮，进入对应界面，分为数据库文件管理、安装命令文件管理、驱动文件管理 2. 填写过滤条件，查询满足条件的文件列表 3. 点击文件记录详情按钮，前端界面显示文件记录详情信息 4. 用户点击文件记录删除按钮，前端界面显示删除结果 5. 点击文件记录的下载按钮 6. 前端节目向后端发送请求，建立文件流，下载文件		
扩展流程	3a. 后端获取不到文件详情，前端界面显示文件已不存在 4a. 删除操作失败，前端界面显示删除失败 6a. 如果文件不存在，前端界面显示文件已丢失		

新建 URL 测试任务用例描述如表3.8所示。用户从已上传驱动文件库列表中选择驱动文件文件用于 java JDBC 连接数据库 URL，输入测试数据库连接 URL、登录所需用户名、密码，确认新建 URL 的测试任务，后台会首先保存任务数据，并不直接执行。在测试任务执行过程中，首先会通过 java 动态加载类的方式，将连接驱动加载注册类，然后通过 JDBC 的方式连接数据库 URL，并分别进行 SQL-92 与 SQL-99 测试集的验证，运行 SQL 测试语句并收集执行结果，通过分析 SQL 执行结果，与上传数据库测试不同，通过 JDBC 的方式运行 SQL，在遇到错误时会抛出异常，要对异常进行处理，得到适配性测试指标保存数据库中。上传数据库测试任务提交后，后台会通过定时任务异步地获取与执行测试任务。测试任务是后台异步的执行与更新状态，用户可在测试历史管理界面中查看对应数据库测试任务的测试状态，如果测试已完成，则可以查看与下载对应的测试报告，也可下载执行过程中生成的日志。

表 3.7: 新建数据库测试任务用例描述

ID	05	用例名	新建上传数据库测试任务
描述	系统用户发起对上传数据库的测试		
参与者	数据库测试用户		
触发条件	系统用户需要对已上传的数据库进行 SQL 标准测试		
前置条件	用户点击新建上传数据库测试按钮，进入对应界面		
后置条件	前端界面反馈任务新建的状态信息，界面跳转到测试任务管理		
优先级	高		
正常流程	1. 用户点击新建上传数据库测试按钮，进入对应界面 2. 用户选择待测试数据库 3. 用户选择安装命令，完善适配系统等信息 4. 用户确认新建，前端向后端发送新建上传数据库测试任务请求		
扩展流程	4a. 用户取消新建测试，界面跳转至首页		

表 3.8: 新建 URL 测试任务用例描述

ID	06	用例名	新建数据库 URL 测试任务
描述	系统用户发起对数据库 URL 的测试		
参与者	URL 测试用户		
触发条件	用户需要对已安装的数据库进行 SQL 适配性测试		
前置条件	用户点击新建数据库 URL 测试，进入对应界面		
后置条件	前端界面反馈任务新建的状态信息，界面跳转至测试任务管理		
优先级	高		
正常流程	1. 用户点击新建数据库 URL 测试，进入对应界面 2. 用户选择已上传库中的连接驱动文件 3. 用户填写需要测试的 URL，用户名，密码 4. 用户确认新建，前端向后端发送新建数据库 URL 测试任务请求		
扩展流程	4a. 用户取消新建测试，界面跳转至首页		

测试任务用例图如3.9所示，用户新建测试任务成功后，可以在进入对应类型的测试任务管理界面实时查看测试任务的执行状态，测试任务列表支持条件查询，包括文件名，上传时间，执行状态等条件，用户可以高效的获取到自己想查看的测试任务信息。测试任务列表展示了每一个测试任务的基本信息，用户

可以点击详情按钮查看详情，或者点击删除按钮删除任务。同时，对于已经运行完成的测试任务，支持查看测试报告，下载测试报告，重新运行等操作。

表 3.9: 测试管理用例描述

ID	07	用例名	测试任务管理
描述	系统用户需要管理已新建的测试任务，包括上传数据库和数据库 URL 测试任务		
参与者	系统用户		
触发条件	在新建测试任务之后，用户需要对测试任务进行管理		
前置条件	用户点击测试任务管理，进入对应界面		
后置条件	前端界面显示对应类型的测试任务列表信息		
优先级	高		
正常流程	1. 用户点击测试任务管理，进入对应界面 2. 用户可输入条件对任务列表进行过滤 3. 前端界面显示过滤后的任务列表 4. 用户点击任务记录的查看详情按钮 5. 前端界面显示详情模态框，显示详情信息 6. 用户点击任务记录的删除按钮 7. 前端界面对删除结果进行显示		
扩展流程	4a. 后端获取不到任务详情，前端界面显示获取失败 7a. 后端删除失败，前端界面显示删除失败		

在测试任务运行结束后，用户可以在测试任务管理界面查看任务的测试报告。上传数据库测试任务的测试报告中展示了数据库 SQL 标准测试结果和运行时 docker 资源状态测试报告，而 URL 测试任务的测试报告，则只有数据库 SQL 标准测试结果。测试报告查看用例描述如表3.10所示。

表 3.10: 查看测试报告用例描述

ID	08	用例名	查看测试报告
描述	在测试任务运行结束之后，用户查看测试报告		
参与者	系统用户		
触发条件	在测试任务运行结束之后，用户需要查看对应的测试报告		
前置条件	用户点击测试任务管理，进入对应界面		
后置条件	前端界面显示测试报告信息		
优先级	高		
正常流程	1. 用户点击测试任务管理，进入对应界面 2. 用户可输入条件对任务列表进行过滤 3. 前端界面显示过滤后的任务列表 4. 用户点击任务记录的测试报告按钮 5. 前端界面显示测试报告		
扩展流程	4a. 后端获取测试报告失败，前端界面显示获取失败 4b. 如果测试任务运行未完成，前端界面提示测试任务未完成		

表3.11描述了测试报告下载用例，测试报告作为本系统最重要的产物，是用户使用本系统的最终目的，为了用户可以离线查看，对报告进行分享等。用户可以点击下载保存测试报告到本地，报告为通用的 PDF 格式。

表 3.11: 下载测试报告用例描述

ID	09	用例名	下载测试报告
描述	用户将测试报告下载保存到本地		
参与者	系统用户		
触发条件	在测试任务运行结束之后，系统用户需要将测试任务的 的测试报告下载保存到本地		
前置条件	系统用户点击测试报告查看，进入对应界面		
后置条件	系统将测试报告保存到本地		
优先级	中		
正常流程	1. 系统用户点击测试报告查看，进入对应界面 2. 点击下载按钮，下载测试报告 3. 前端界面将 HTML 转成 pdf 文件，并保存到用户本地文件夹		
扩展流程	3a. pdf 转换失败，前端界面显示下载失败		

3.3　系统设计

3.3.1　系统架构设计

本系统将前端界面与后端服务分离。后端采用 MVC 架构，将功能模块分离，向前端提供 RESTful 的接口。前端采用了主流的 React 框架，实现用户提供与系统交互的 web 界面，通过 UMI 实现与后端的交互。为了提高请求的访问速度，后端采用了异步阻塞模型，处理请求的负载均衡通过 Nginx 来实现。如图3.3所示，系统从用户到底层划分三个部分，分别为前端页面、后端服务处理、数据持久化。

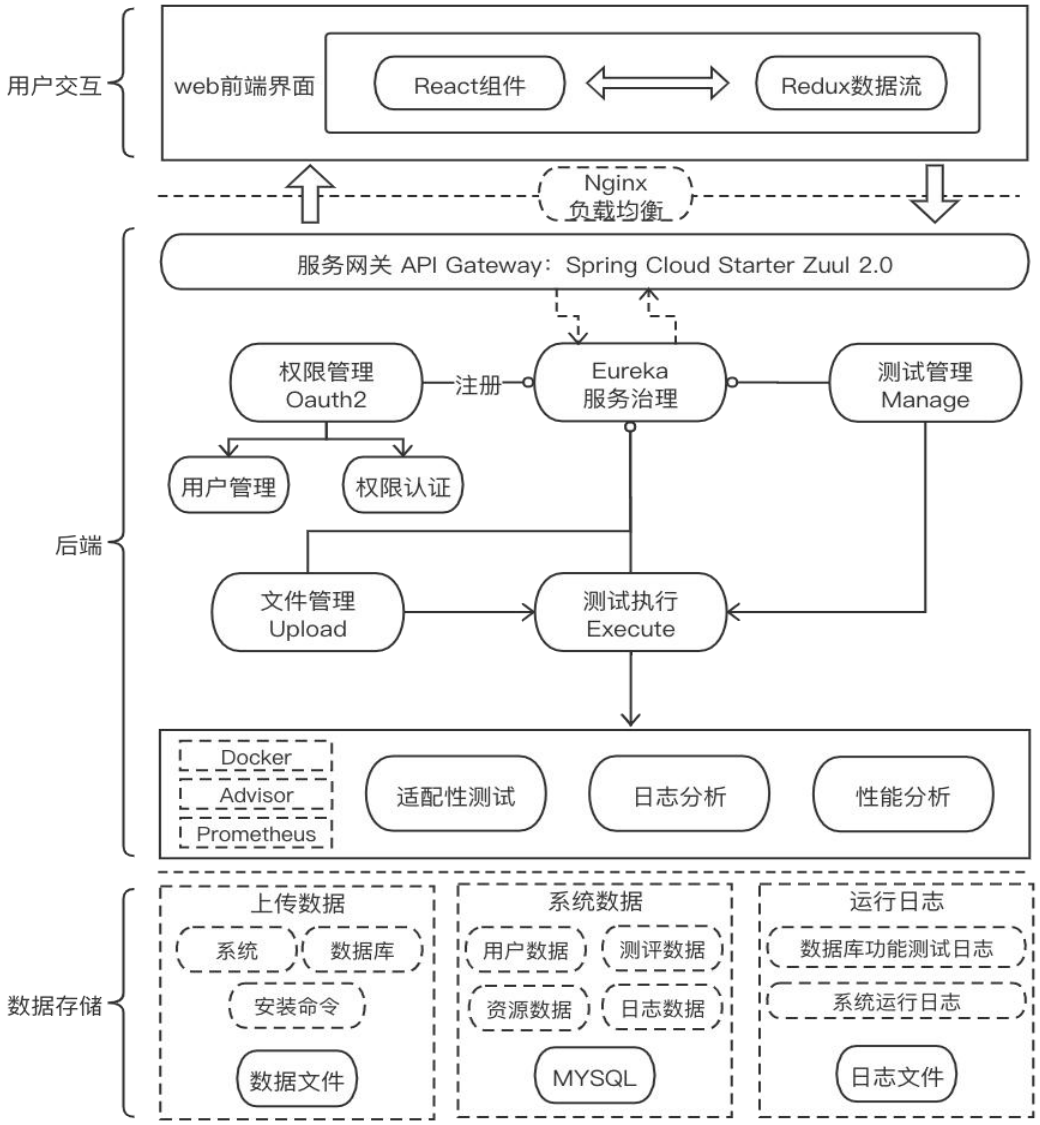

图 3.3: 数据库与系统适配性测试系统框架图

系统后端架构设计分为权限管理、文件管理、测试任务管理和测试执行四

个模块。权限管理模块，用于实现用户登录与用户请求合法性验证，隔离非法请求，保证安全性；文件管理模块对数据库文件，安装命令文件和连接驱动文件进行管理，提供上传，查看，删除等操作；测试管理模块负责对上传数据库测试和数据库 URL 测试任务的管理，负责新建与查询等测试任务的操作；测试执行模块，负责从任务管理模块获取待测试任务，从文件管理模块获取文件资源，并执行测试任务、对 SQL 执行的结果进行计算，如果是上传数据库测试任务则还需对运行时 Docker 容器的性能指标进行收集，生成适配性测试报告，并实现查看与下载测试报告的功能。

在数据存储层，根据数据类型的不同，进行区别的处理，后端运行中生成的业务数据存储在 Mysql 数据库中；程序运行中生成的日志保存为日志文件，当程序出现故障时，方便查看错误；对于用户上传的文件包括数据库文件，安装命令文件和连接驱动文件等，则保存在文件系统中，便于维护与获取。

3.3.2　逻辑设计

本小节对数据库 SQL 标准适配性自动化测试系统逻辑设计进行描述，根据上述的功能拆分和架构设计，对本系统的逻辑结构进行了设计，逻辑视图如图3.4所示，描述了系统内部各个模块的关系。

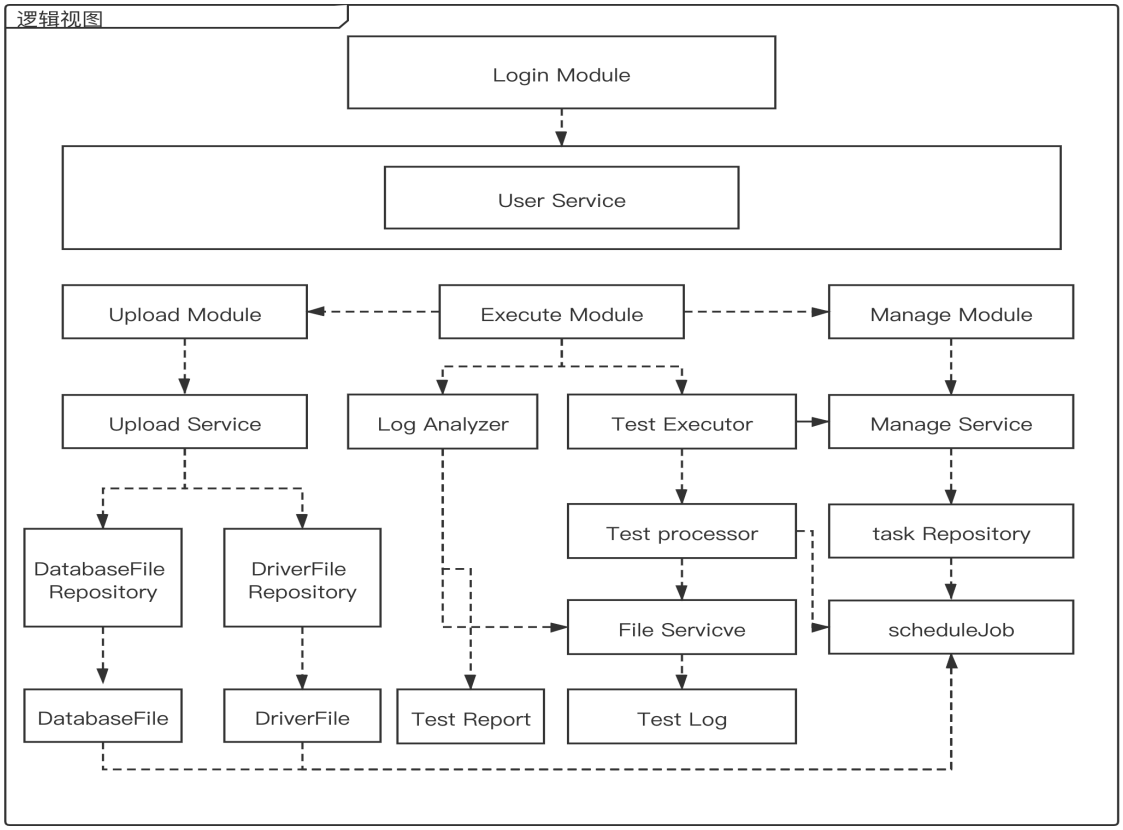

图 3.4: 数据库与系统适配性测试系统逻辑视图

在逻辑层上，系统主要分为四个模块，分别为 Login Module、Upload Module、Manage Module 和 Execute Module。Login Module 依赖了 UserService，负责实现系统用户的登录，对用户发起的请求进行合法性认证与服务转发处理。用户登录之后才可以正常使用本系统；Upload 模块依赖了 Upload Service 实现文件记录的处理，和 File Service 实现文件资源的管理，实现上传文件管理。本系统中涉及用户上传的文件有三种类型 Database File 数据库文件、intall File 命令文件和 Driver File 连接驱动文件，每类文件对应一个 Repository 类实现与数据库的交互操作；Manage Module 依赖 Manage Service，Manage Service 负责实现与管理测试任务相关的逻辑操作，测试任务包含上文中提到的数据库测试测试任务 Database schedule 和 URL schedule 测试任务两种类型，测试任务执行过程中，会生成对应的 Metrics 测试结果，并存储在系统数据库中，供测试报告生成时需要；Execute Module 主要负责测试任务的执行与测试过程中的执行结果分析的功能，为 Manage Module 提供 docker 容器运行管理、测试任务运行，测试结果分析和指标计算的服务。

3.3.3　开发设计

React 是本系统的前端所采用框架，作为一个分层的结构，主要分为 config、pages、locales、services、models、node modules 等模块，如图3.5所示。

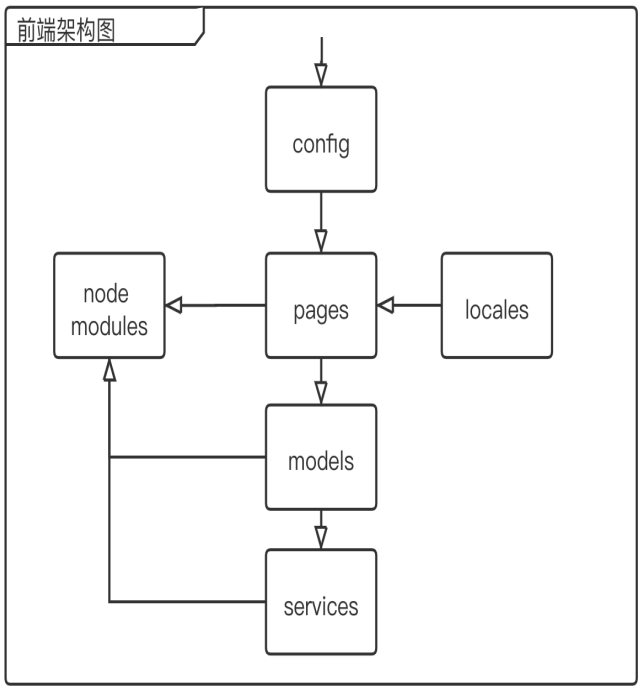

图 3.5: 前端结构图

- config 包保存了前端的许多配置文件，包括页面路由，请求代理等，设置

了页面的跳转规则以及前端发送给后端请求的转发规则。

- page 包内保存了前端的所有页面，每一个界面为一个文件夹，每个文件夹包括了页面的 jsx 静态文件、css 文件等。
- locales 包保存了前端的语言信息，为前端页面的多语言化提供了支持。
- services 负责与后端进行数据交互，是数据请求模块。
- model 保存了与数据处理相关的类，在向后端请求数据之后，往往需要对数据进行处理才能在前端界面显示，同时 model 包也管理了数据的状态变化。
- node modules 保存前端页面中所依赖的组件与工具，包括了组件包，打包工具，请求工具等，是前端界面运行所必须的。

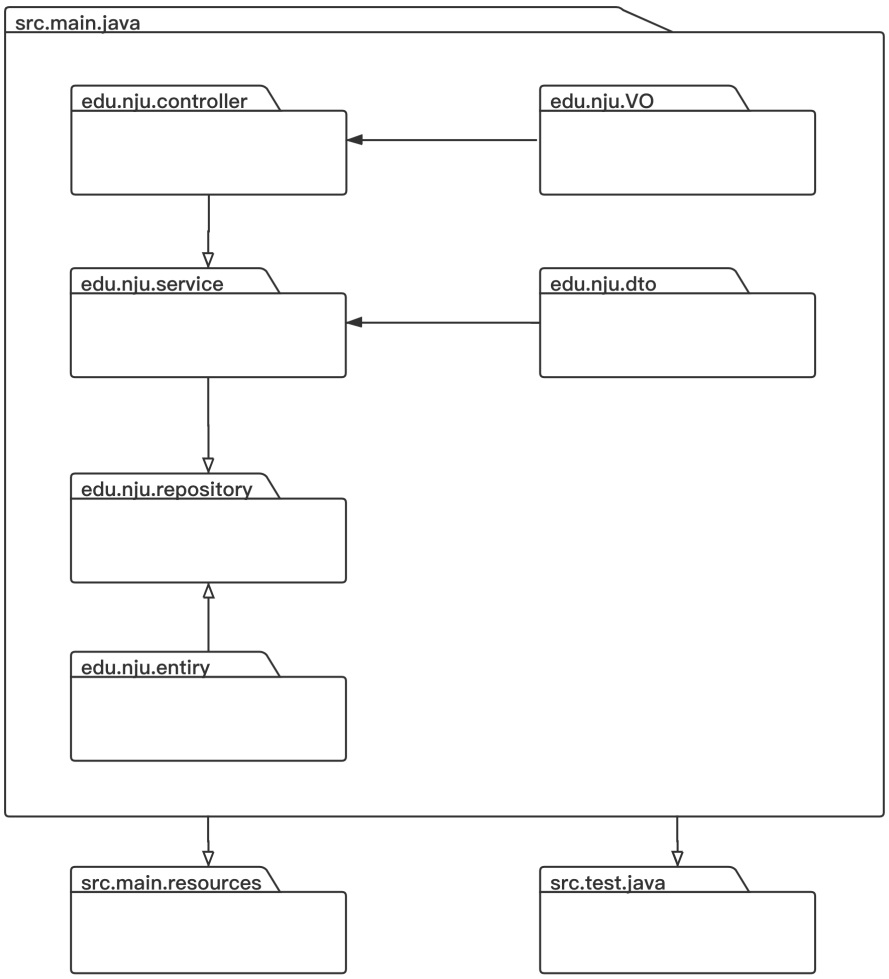

图 3.6: 后端结构图

参考 Spring Boot 官方建议的项目结构, 本系统遵循 MVC 架构风格，后端包结构图如图3.6所示。

- controller 包主要负责对 HTTP 请求的拦截、转发和解析，实现具体业务模块的流程控制; service 包定义了接口, 并实现了具体的逻辑处理; repository 包负责与数据库进行交互，采用了 Hibernate 框架。
- entities 包中对应系统涉及的数据库实体类信息，数据库中的每一张表分别对应包中的一个类。
- VO 包保存了前端向后端发送数据请求的参数，为了方便前端的数据请求，将参数封装为对象，方便后端解析与逻辑处理。
- dto 包保存了向后端发送数据请求的结果对象, 将后端的数据进行封装, 防止不必要的数据泄漏，减少了前后端请求的数据量，也提高了数据的安全性。
- repository 包对应的是 spring data jpa 所实现的类，负责与数据库交互，进行数据持久化与查询等相关操作。
- resources 中保存了配置文件信息，如 mysql 连接信息，运行日志保存信息等。
- test 包保存了单元测试用例，对整个系统的功能点进行了测试。

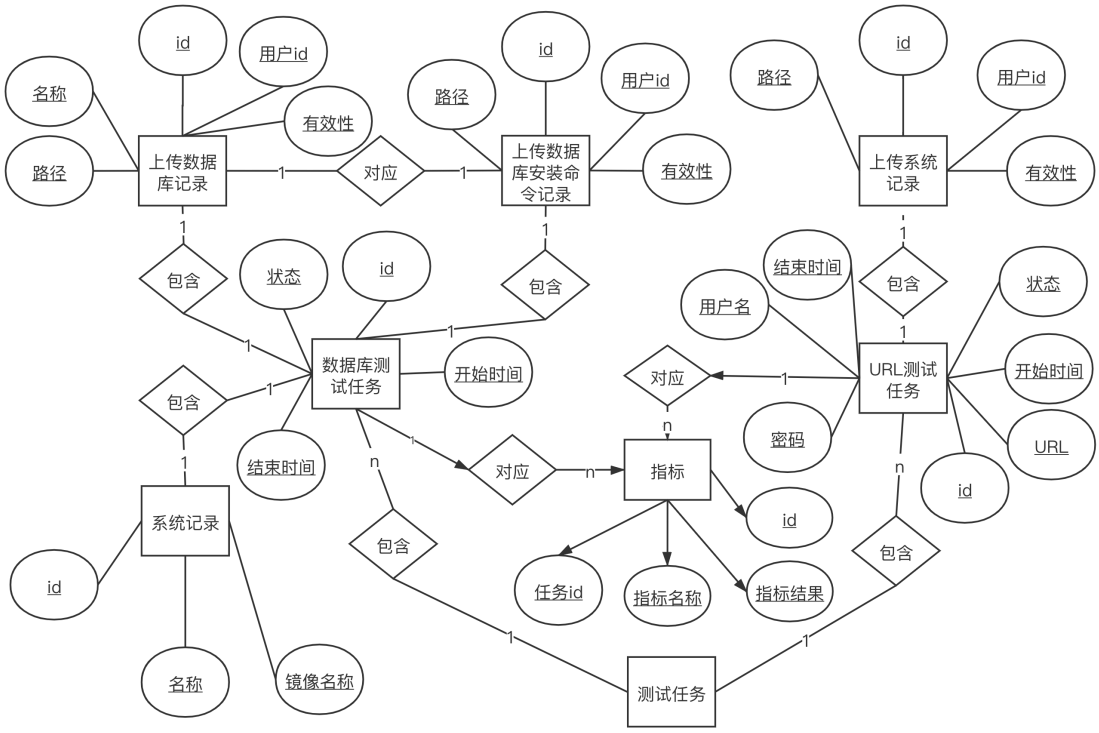

图 3.7: 实体关系图

本系统的数据库实体关系图如图3.7所示，三种上传文件都有数据库实体记录对应，在上传文件成功后，在数据库中保存对应的数据库文件、安装命令文件或连接驱动文件记录。数据库测试任务实体包含数据库文件记录、安装文件命令记录和所选的操作系统列表，用于测试上传数据库文件的测试评估；URL 测试任务实体中包含连接驱动文件记录、数据库连接 URL、用户名和密码。测试结果实体用于保存测试 SQL 的结果信息，每一条测试结果对应着一条 SQL 的执行。一个测试任务实体对应着许多的测试结果实体，对测试结果实体的分析，形成了测试报告。

根据所分的四个功能模块设计了数据库表。权限管理模块涉及用户实体 User，保存了用户的基本信息，由于用户信息较为通用，在此不与展示。文件管理模块用到了上传数据库安装命令记录实体 UploadInstallFile、上传数据库记录实体 UploadDatabaseFile、上传连接驱动记录实体 UploadDriverFile 和系统镜像实体 ImageFile，分别对应数据库中的表3.12upload_install_file、3.13upload_database_file、表3.14upload_driver_file 和3.15image_file 四张表.。

表 3.12: upload_install_file 表字段详细设计

数据库字段	意义	数据类型	详情描述
install_id	数据库安装命令 id	bigint	数据库安装命令记录主键
user_id	用户记录 id	bigint	系统用户主键
install_name	数据库安装命令文件名称	varchar(45)	指定的数据库安装命令文件名称
install_path	数据库安装命令文件路径	varchar(45)	指定的数据库安装命令文件路径
upload_time	上传时间	datetime	文件上传时间
is_delete	是否已经被删除	boolean	用于判断文件是否过期
remark	备注	varchar(45)	安装命令文件备注

表 3.13: upload_database_file 表字段详细设计

数据库字段	意义	数据类型	详情描述
database_id	上传数据库文件 id	bigint	上传数据库文件记录主键
user_id	用户记录 id	bigint	系统用户主键
database_name	数据库文件名称	varchar(45)	指定的数据库文件名称
database_path	数据库文件路径	varchar(45)	指定的数据库文件路径
run_command	数据库运行命令头	varchar(45)	数据库运行命令头
upload_ime	上传时间	datetime	文件上传时间
is_delete	是否已经被删除	boolean	用于判断文件是否过期
remark	备注	varchar(45)	数据库文件备注

表3.12, 表3.13和表3.14分别对数据库安装命令记录, 数据库文件记录和连接驱动文件记录数据库表进行了描述。对于这三个记录, 由于本系统删除是逻辑删

除，isDelete 字段表示了该上传文件是否是已经被三处的过期文件。UploadTime 保存了文件上传的时间，remark 则为文件的备注，方便使用人员识别文件的意义。

表 3.14: upload_driver_file 表字段详细设计

数据库字段	意义	数据类型	详情描述
driver_id	上传连接驱动文件记录 id	bigint	上传驱动文件记录主键
user_id	用户记录 id	bigint	系统用户主键
driver_name	连接驱动文件名称	varchar(45)	指定的连接驱动文件名称
class_name	注册类名称	varchar(45)	指定的连接驱动类名称
driver_path	系统文件路径	varchar(45)	指定的连接驱动文件路径
upload_time	上传时间	datetime	文件上传时间
is_delete	是否已经被删除	boolean	用于判断文件是否已被删除
remark	备注	varchar(45)	系统文件备注

表3.15描述了系统文件记录数据库表详细涉及，这个表保存了当前系统中可供用户测试数据库适配性选择的操作系统列表，作为数据库安装、运行测试的测试床，表中保存了系统文件的别名和 docker 名。表中主要包含了一些常用的 linux 操作系统 Docker 镜像，例如 ubuntu，centos 的常用版本。

表 3.15: image_file 表字段详细设计

数据库字段	意义	数据类型	详情描述
image_id	系统镜像记录 id	bigint	系统镜像记录主键
imageName	系统名称	varchar(45)	指定的系统名称
imageDockerName	系统 docker 名称	varchar(45)	系统对应的 docker 镜像名称

管理模块 Manage Module 和执行模块 Execute Module 涉及表 database_schedule_job、url_schedule_job、metics、DockerResource 分别用于存储上传数据库测

试任务记录实体 DataBaseScheduleJob、数据库 URL 测试任务记录实体 UrlScheduleJob，结果指标记录实体 Metics 和容器性能记录实体 DockerResource, 分别对应表3.16、表3.17, 表3.18和表3.19所示。

上传数据库测试任务如表3.16所示, database_schedule_job 记录了任务 id, 数据库文件记录 id, 系统记录 id, 安装命令文件记录 id, 测试任务执行状态, 测试任务开始时间，测试任务结束时间。URL 测试任务如表3.17所示，url_schedule_job 记录了任务 id，连接驱动文件 id，连接 URL、用户名、密码，测试任务执行状态，测试任务开始时间，测试任务结束时间。任务执行状态包含等待调度、正在执行、执行结束三种状态。

表 3.16: database_schedule_job 表字段详细设计

数据库字段	意义	数据类型	详情描述
schedule_job_id	上传数据库测试任务记录 id	bigint	上传数据库测试任务记录主键
user_id	用户记录 id	bigint	系统用户主键
database_id	数据库 id	bigint	上传数据库记录的 id
install_id	数据库安装命令 id	bigint	上传数据库安装命令记录的 id
image_id	系统镜像 id	bigint	系统的 id
start_time	开始时间	datetime	测试任务的开始时间
end_time	结束时间	datetime	测试任务的结束时间
state	状态	tinyint(1)	任务的状态
is_delete	是否已经被删除	boolean	用于判断任务是否已被删除

测试结果如表3.18所示，记录了执行任务 id，即数据库测试任务 id 或者系统测试任务 id，指标名称，执行的 sql 语句，期望的输出日志，实际的输出日志，指标结果。指标结果包括运行正常和运行失败两种结果。Metric 表代表了测试报告中的功能指标，用于记录日志分析中得到的结果，代表了数据库与系统的功能适配性。

表 3.17: url_schedule_job 表字段详细设计

数据库字段	意义	数据类型	详情描述
schedule_job_id	任务记录 id	bigint	任务记录记录主键
user_id	用户 id	bigint	系统用户主键
driver_id	连接驱动记录 id	bigint	连接驱动记录 id
start_time	开始时间	datetime	测试任务的开始时间
end_time	结束时间	datetime	测试任务的结束时间
sql_url	数据库连接 url	varchar(255)	待测试的数据库连接 url
username	数据库用户名	varchar(255)	登录数据库所需用户名
password	数据库密码	varchar(255)	登录数据库所需密码
state	状态	tinyint(1)	任务的状态
is_delete	是否已经被删除	boolean	用于判断任务是否已被删除

表 3.18: Metric 表字段详细设计

数据库字段	意义	数据类型	详情描述
metric_id	测试结果记录 id	bigint	测试结果记录主键
schedule_job_id	执行任务 id	int(11)	数据库或 url 测试任务 id
name	指标名称	varchar(45)	指标名称
sql	执行的 sql 语句	varchar(45)	测试的结果日志
excute_time	执行时间	datatime	执行测试语句所花费的时间
expect_log	期望日志	varchar(255)	期望的结果日志
real_log	实际日志	varchar(255)	实际的日志
result	指标结果	varchar(45)	指标结果
type	指标类型	varchar(45)	用于判断是什么类型测试的结果

DockerResourse 表如3.19所示，记录了执行任务 id，即上传数据库测试任务 id，cpu 占用，内存占用，网络输入，网络输出，块输入，块输出。DockerResourse

表记录了数据库与系统适配性测试任务执行时容器的性能指标，作为测试报告的重要组成部分。

表 3.19: DockerResourse 表字段详细设计

数据库字段	意义	数据类型	详情描述
id	任务记录 id	int(11)	任务记录记录主键
schedule_job_id	执行任务 id	int(11)	数据库测试任务 id
max_cpu	最大 cpu 占用率	int(11)	容器最大占服务器 cpu 百分比
avg_cpu	平均 cpu 占用率	int(11)	容器平均占服务器 cpu 百分比
max_memory	最大内存占用率	int(11)	容器最大占用内存
max_net_input	最大网络输入	int(11)	容器最大的网络输入
avg_net_input	平均网络输入	int(11)	容器平均的网络输入
net_output	网络输出	int(11)	容器的网络输出
max_block_input	最大块输入	int(11)	容器最大的块输入
avg_block_input	平均块输入	int(11)	容器平均的块输入
max_block_output	最大块输出	int(11)	容器最大的块输出
avg_block_output	平均块输出	int(11)	容器平均的块输出

3.3.4　进程设计

本节将介绍数据库 SQL 标准适配性自动化测试系统的进程设计，分为文件管理, 测试管理, 测试执行三个模块。

文件管理进程视图如图3.8所示，用户上传数据库文件、安装命令文件或者连接驱动文件，后端将文件保存到文件系统中，并在数据库新增对应的文件记录。由于文件保存处理速度较快，因此文件管理服务反馈后，微服务便将处理状态反馈前端，前端需要同步等待，可以及时的得到反馈。在上传文件成功之后，用户可以在文件管理界面查看与管理文件信息。对于三类文件，上传文件与文件管理的流程相似。

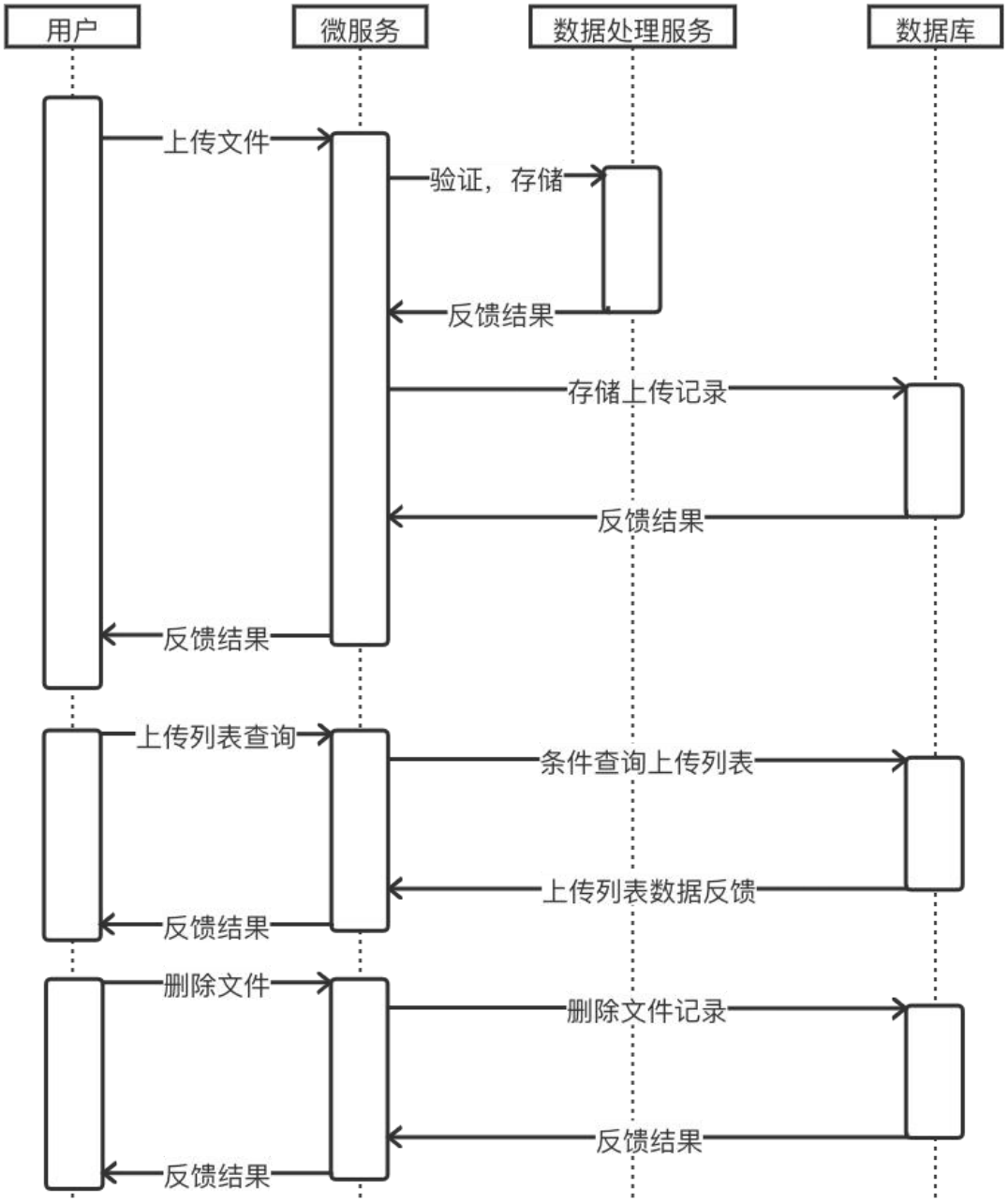

图 3.8: 上传管理进程视图

图3.9展示了测试模块的进程设计，数据库 SQL 标准适配性自动化测试最重要的核心功能即对数据库进行测试。当用户有测试需求时，首先需要上传测试必须文件，然后新建测试任务，后台微服务根据任务类型创建一个测试任务，对于上传数据库任务，需要保存所选择的待测试数据库，数据库安装命令文件，操作系统；对于数据库 URL 测试，需要保存连接驱动文件，数据库连接 URL，用

户名，密码，然后同步在数据库中新增一条测试记录。在数据库保存测试记录后，后端立即返回结果，前端显示新建结果。

如图3.10所示, 测试执行模块为定时任务，定时地获取测试任务，并根据测试任务的信息，异步执行数据库在对应系统下运行的功能测试，并生成日志。根据测试运行的反馈信息，更新测试状态到数据库中，测试任务包含等待调度，正在执行、运行结束三种状态，在执行测试过程中，会将 SQL 运行时产生的结果进行保存。后端执行任务异步进行，用户可在任意时刻查看测试任务的状态。

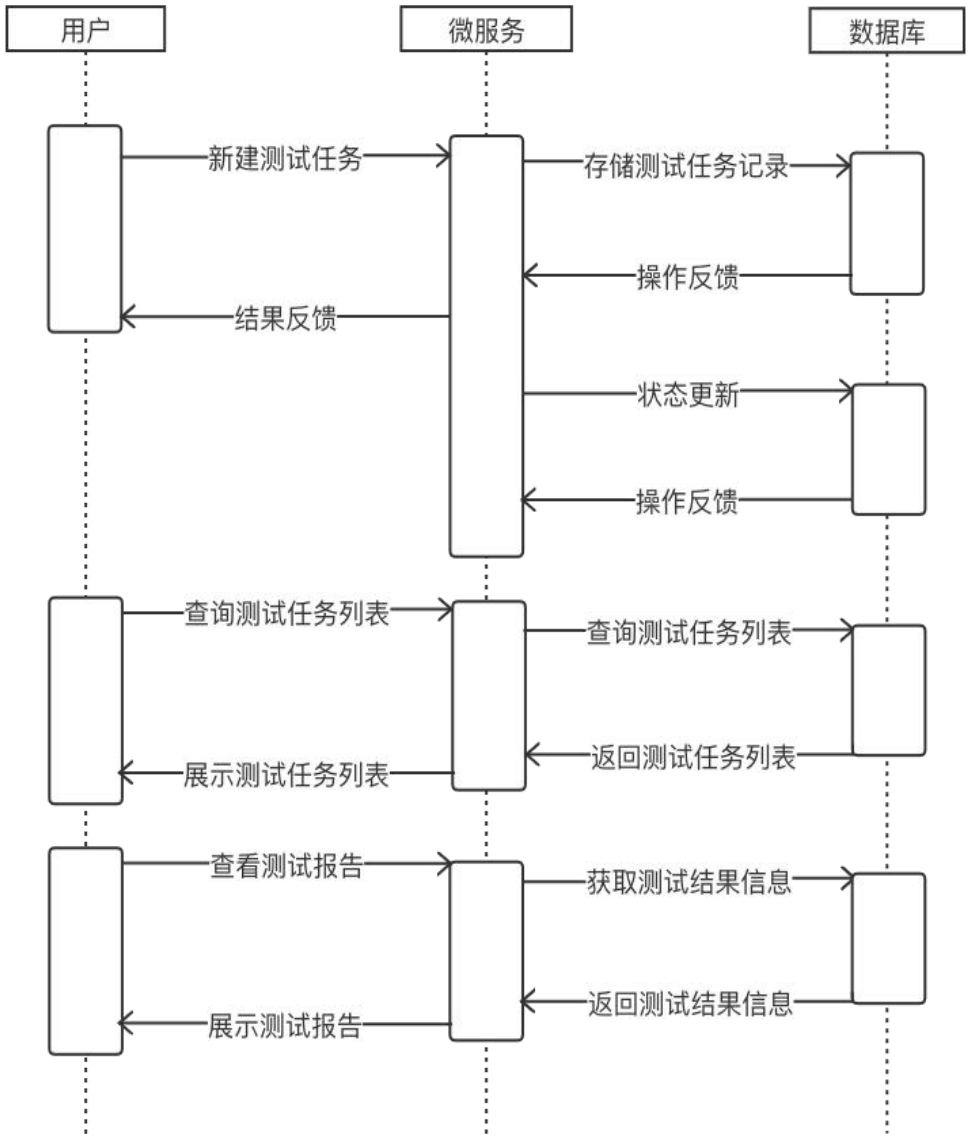

图 3.9: 测试管理进程视图

数据库 SQL 标准适配性自动化测试技术最终给用户的结果是测试报告，因此要对数据库 SQL 标准测试返回的结果进行分析。对于上传数据库测试，是通过运行 docker 命令的方式执行 SQL 语句，并对执行后容器输出信息进行分析，得到每一条测试 SQL 的测试结果。对于数据库 Url 测试，通过 JDBC 连接的方式执行 SQL 语句，并对返回的结果集进行分析，得到每一条测试 SQL 的测试结果。

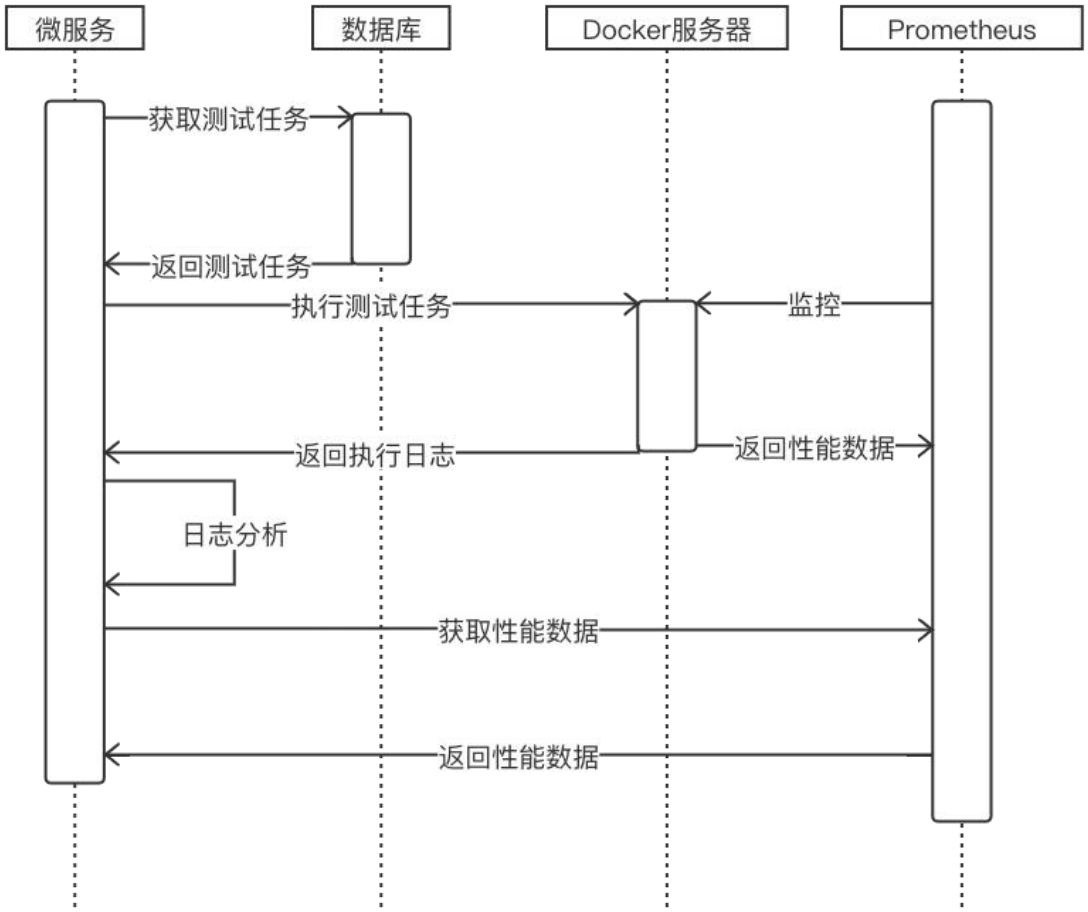

图 3.10: 测试管理进程视图

3.3.5　系统部署设计

系统整体的部署设计如图3.11，将前端与后端部署在不同的服务器上。后端部署为微服务，包括 upload 上传管理微服务、login 权限管理微服务、manage 测试管理微服务、execute 测试执行微服务。每个微服务部署在不同的服务器上，可按需动态扩展与缩减部署的服务器数量，提高系统处理用户请求的能力。前端 React 代码打包后通过 Nginx 部署服务，实现负载均衡。

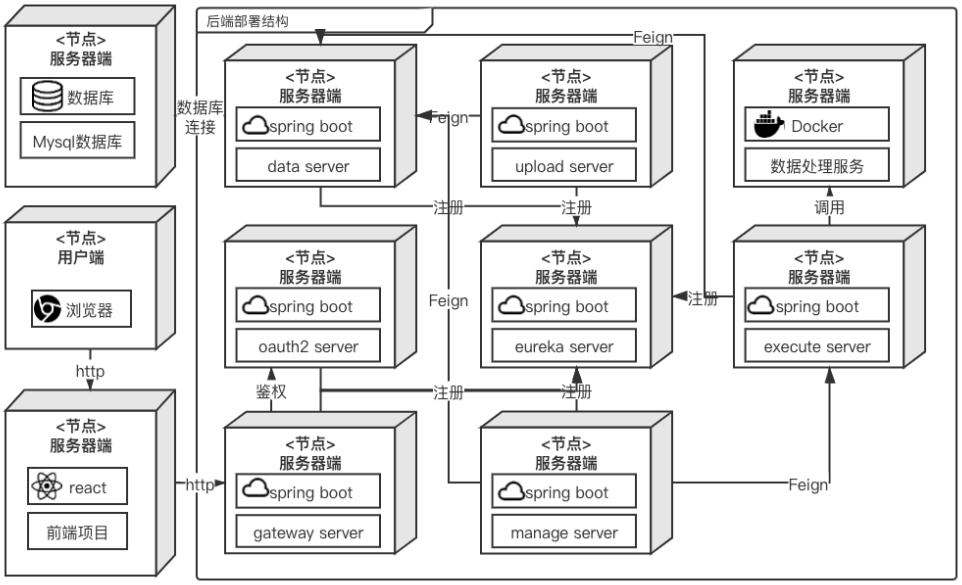

图 3.11: 系统部署图

3.4　Docker 性能评估指标设计

本小节针对数据库 SQL 标准适配性自动化测试技术进行了性能指标的设计。性能评估指标主要评估数据库运行时所占系统资源，如 CPU、内存等，主要通过 google advisor 工具来收集，然后通过 Promethues 计算；功能指标主要评估数据库的功能是否正常，如创建库表、插入数据等。功能评估指标主要是通过本系统分析 SQL SQL-92 与 SQL:1999 标准，然后设计与整理的 SQL 标准测试集进行评估。

最大 CPU 占用主要评估数据库在运行的时候，所需要的最大的处理器资源，主要根据数据库运行时所占 CPU 时间计算，计算公式如下：

$$max\_cpu = max(sum(irate(container\_cpu\_usage\_seconds\_totalid = "dockerId"[5m]))) \tag{3.1}$$

平均 CPU 占用主要评估数据库在运行的时候，所需要的平均的处理器资源，主要根据数据库运行时所占 CPU 时间计算，计算公式如下：

$$avg\_cpu = avg(sum(irate(container\_cpu\_usage\_seconds\_totalid = "dockerId"[5m]))) \tag{3.2}$$

最大内存占用主要评估数据库在运行的时候，所需要的最大的内存资源，主要根据数据库运行时所占内存计算，计算公式如下：

$$max\_mem = max\_over\_time(container_memory_usage_bytesid = dockerId[1w]) \quad (3.3)$$

平均内存占用主要评估数据库在运行的时候，所需要的平均的内存资源，主要根据数据库运行时所占内存计算，计算公式如下：

$$max\_mem = avg\_over\_time(container\_memory\_usage\_bytesid = dockerId[1w]) \quad (3.4)$$

总文件读入大小主要评估数据库在运行时，文件读入操作资源的数量，主要根据数据库运行时所读取文件流大小计算，计算公式如下：

$$fs\_read = container\_fs\_reads\_bytes\_totalid = dockerId \quad (3.5)$$

总文件写入大小主要评估数据库在运行时，文件写入操作资源的数量，主要根据数据库运行时所读取文件流大小计算，计算公式如下：

$$fs\_write = container\_fs\_writes\_bytes\_totalid = dockerId \quad (3.6)$$

3.5　本章小结

本章首先总体规划数据库 SQL 标准适配性自动化测试系统，分析了本系统的功能性需求和非功能性需求，将系统功能拆分为四个主要模块：权限管理、文件管理、测试管理和测试执行，并总结了九个用例。展示了本系统架构、逻辑、开发、进程和部署 4+1 视图。最后详细阐述了数据库 SQL 标准适配性自动化测试系统的 docker 性能指标设计。

第四章　详细设计与实现

本章将介绍本系统的详细设计与实现。上一章将系统拆分权限管理、文件管理、测试任务管理和测试执行四个模块，本章将分别详细介绍这四个模块的设计与实现。同时通过一些系统的界面展示系统的实现。

4.1　权限管理模块详细设计与实现

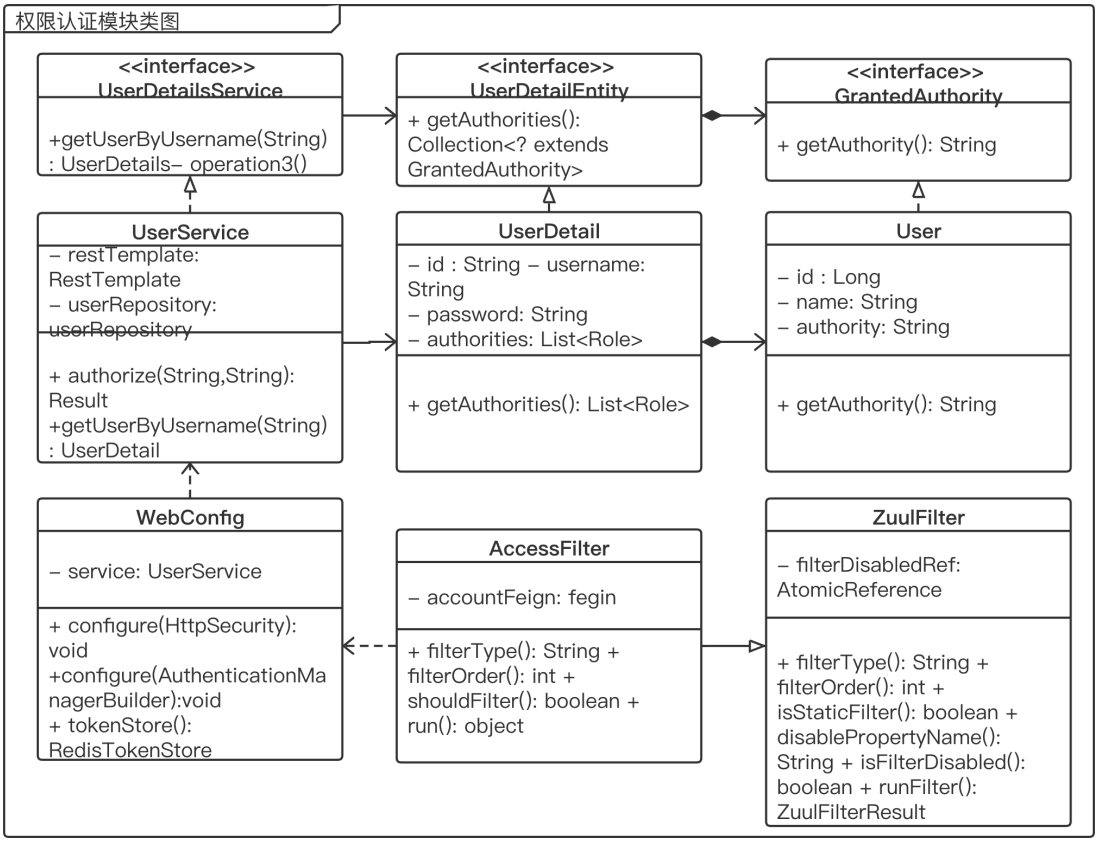

图 4.1: 权限管理模块类图

4.1.1　权限管理模块的详细设计

权限认证模块负责实现用户登录与用户请求合法性验证，图4.1展示了其详细设计类图。本系统结合慕测主站提供的开放认证接口实现用户登录，以及对用户的请求进行合法性认证。在第一次登录时，通过慕测主站的用户登录接口

对用户登录进行验证，在获取完用户信息之后，会将用户信息保存在本系统中，减少请求的重复。同时利用慕测主站的网关拦截用户请求，进行合法性认证，阻止非法的请求，保证系统的安全性。

4.1.2　用户登录的实现

数据库 SQL 标准适配性自动化测试系统用户权限由慕测主站管理。用户登录通过调用主站的开发接口实现，系统会获取用户信息并保存。在保存了用户登录信息之后，本系统则不需要再向慕测主站获取用户信息，减少了请求的路径，提高处理请求效率。图4.2展示了关键代码，在获取请求之后，系统首先判断用户是否已存在系统中，如果不存在，则需要请求用户信息，然后将用户信息保存到系统中，若系统中已存在用户信息，则直接获取。

```
//用户登录相关代码
@Override
public Result<UserDetail> getUser(String username){
    List<User> users = userRepository.getByUsername(username);
    UserDetail userDetail = new UserDetail();
    if (users.size() == 0) {
        // 如果当前用户未存在系统中
        ResponseEntity<UserDetail> userDetail = restTemplate.
        getForEntity(mooctestAuthURL, UserDetail.class);
        // 解析获取的用户信息并保存在系统中
        update(entity, userDetail);
    } else {
        // 获取本地保存用户信息
        userDetail = new UserDetail(users.get(0));
    }
    return Result.successOf(userDetail);
}
```

图 4.2: 用户登录关键代码

4.1.3　请求验证的实现

通过慕测网关微服务认证用户请求，网关微服务采用 Zuul2.0 编程模型，为了降低请求延迟，支持请求高并发，对后端服务的 API 请求基于 Netty Client 实现处理。为了保证请求的合法性，通过慕测网关微服务对后端所接受的所有请求进行统一认证处理，在经过认证之后，将用户密钥保存在请求头中，作为请求的唯一标识，接着转发到对应微服务进行请求处理。图4.3展示了网关请求合法

性认证的关键代码。

```
// 用户请求认证
@Override
public void authWeb() {
    // 获取请求
    RequestContext context = RequestContext.getCurrentContext();
    HttpServletRequest httpRequest = context.getRequest();

    // 获取用户密钥
    String token = httpRequest.getHeader(StringUtil.USER_TOKEN);

    //如果头部不带用户密钥 ， 则检查参数中是否有用户密钥
    if (StringUtils.isEmpty(token)) {
        token = request.getParameter(StringUtil.USER_TOKEN);
    } else {
        //获取用户信息
        User user = userService.getUser(token);
        JSONObject jsonObject = JsonUtil.parseObject(user.getData().toString());
        if (jsonObject != null && jsonObject.containsKey(StringUtil.ID)) {
            // 把用户 id 加到请求中
            context.addZuulRequestHeader(StringUtil.USER_ID,
            jsonObject.getString(StringUtil.ID));
        }
    }
}
```

图 4.3: 网关认证关键代码

4.2　文件管理模块详细设计与实现

4.2.1　文件管理模块的详细设计

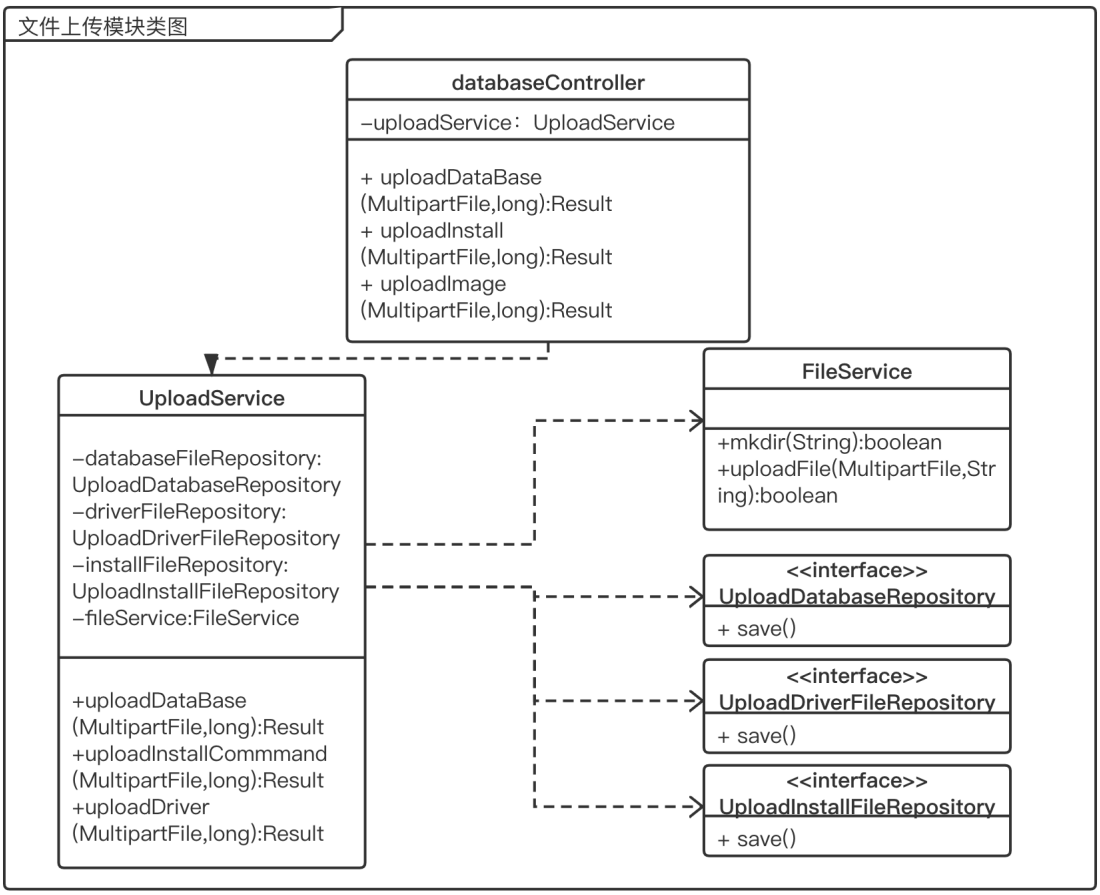

图 4.4: 文件上传模块类图

图4.4展示文件管理模块核心类图，文件管理模块负责实现用户上传，查看与删除数据库文件，安装命令文件和连接驱动文件的功能。用户上传文件的请求由 UploadController 拦截并处理，UploadController 依赖于业务逻辑层的 UploadService。同时，UploadService 依赖 FileService 实现将文件保存到后端文件系统中。FileManageController 负责处理用户查看文件列表，按条件查询文件列表，删除与下载文件的请求，依赖业务逻辑层的 FileManageService。UploadDatabaseFileRepository，UploadInstallFileRepository 和 UploadDriverFileRepository 分别表示对上传的数据库实体，安装命令实体和连接驱动实体进行相关数据库操作的类。

4.2.2　文件上传的实现

```
//上传数据库文件
public Result<String> uploadDatabaseFile(MultipartFile databaseFile,long userId, String commandHead, String remark) {
    //创建目录
    String filePath = fileService.mkdirDatabase(userID，databaseFile);
    //保存文件
    String message = fileService.uploadFile(databaseFile, filePath, commandHead, remark);
    //保存记录
    uploadDatabaseFileRepository.save(new UploadDatabaseFile(userId, databaseFile.getOriginalFilename(), filePath));
    return Result.ofSuccess(message);
}
//上传安装命令文件
public Result<String> uploadInstallFile(MultipartFile installFile, long userId, String remark) {
    //创建目录
    String filePath = fileService.mkdirInstall(userID，databaseFile);
    //保存文件
    String message = fileService.uploadFile(installFile,filePath);
    //保存记录
    uploadInstallFileRepository.save(new UploadInstallFile(userId, filePath, remark));
    return Result.ofSuccess(message);
}
//上传连接驱动文件
public Result<String> uploadDriverFile(MultipartFile imageFile,long userID, String remark, String className, String driverName) {
    //创建目录
    String filePath = fileService.mkdirInstall(userID，databaseFile);
    //保存文件
    String message = fileService.uploadFile(installFile,filePath);
    //保存记录
    uploadDriverFileRepository.save(new UploadDriverFile(userId, filePath, className, remark, driverName));
    return Result.ofSuccess(message);
}
```

图 4.5: 上传文件关键代码

本系统中的数据库文件，安装命令文件和连接驱动文件上传操作都需要对文件进行上传。作为测试任务的重要组成成分，数据库文件，安装命令文件和连接驱动文件需要在用户新建相关测试前上传，然后才能在新建测试任务时选择。用户上传文件时，前端将文件流传入后端，后端对文件进行持久化，如果存在同名文件，则提示用户已存在该文件，用户可以选择覆盖或者取消上传，由于对不同用户进行了文件夹分离，因此不存在不同用户的同名文件覆盖的情况。在文件持久化之后，数据库会新增一条对应的上传文件记录，供测试任务提交时使用。

图 4.6: 文件上传界面

图4.6显示了三种文件上传的界面，对于三种文件的上传，都需要提交本地文件，与填写备注，备注是为了便于使用者在上传之后标识文件。上传数据库的时候，需要指定数据库的运行命令头，以便执行 sql 测试集。对于连接驱动文

件，需要填写类名，作为 java 动态加载类，新建类时

4.2.3　文件管理的实现

在上传文件成功之后，后台会对文件进行持久化，解析处理操作，用户可以分别在对应文件的管理界面进行已上传文件列表的查看与管理。文件管理主要实现对用户已经上传的数据库文件，安装命令文件，连接驱动文件的条件查询，查看，下载，删除等操作，为避免误删除，本系统在表中设置了 is_delete 字段，用于表明记录的逻辑删除，而并不是物理删除。如图4.7所示，以已上传数据库文件管理为例，用户需要查看或者管理已上传数据库文件时，点击进入数据库文件管理界面，前端页面显示文件上传列表，如果用户需要对文件列表进行过滤，通过指定文件名，上传时间等条件进行过滤，对文件记录可以进行下载，删除操作。

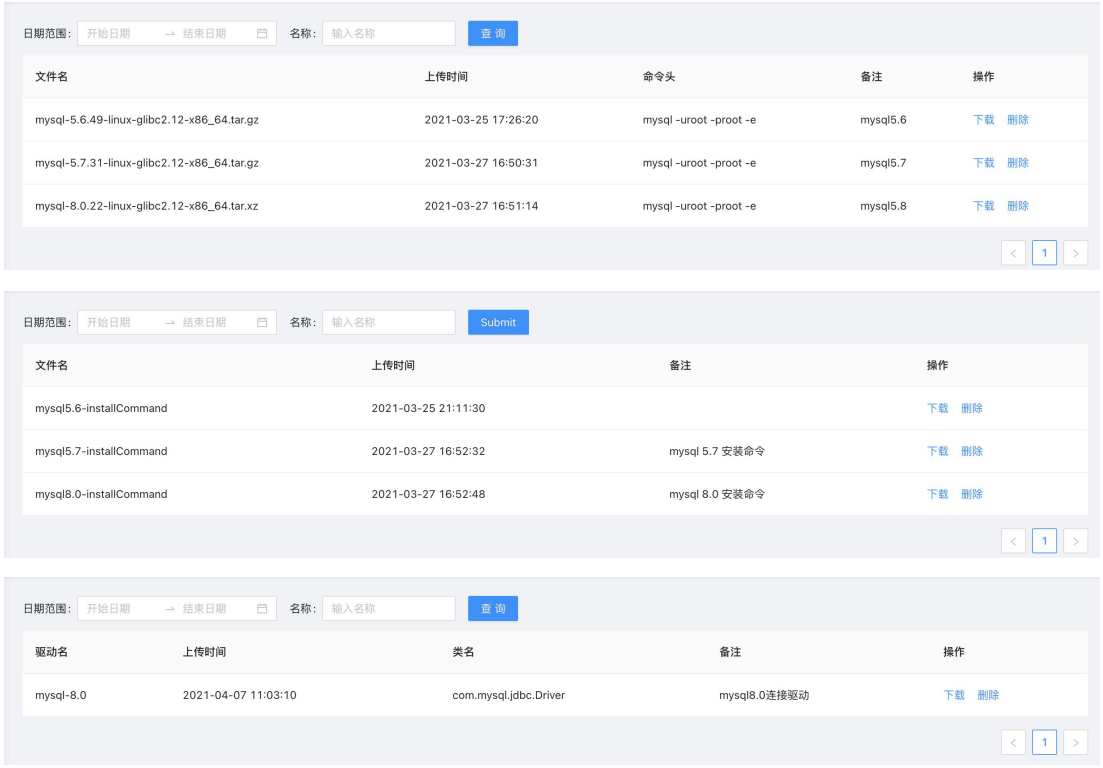

图 4.7: 文件管理界面

4.3　测试任务管理模块详细设计与实现

4.3.1　测试任务管理模块的详细设计

测试任务管理模块负责实现管理测试任务相关的功能，包括管理测试任务，查看与下载测试报告，下载日志等。测试任务管理为此模块的核心功能，包括

测试任务的新建，任务列表查询，查看任务详情等操作。测试任务包括上传数据库测试和数据库 URL 测试，对于上传数据库测试任务，用户需从已上传数据库文件列表中选择待测试数据库，并从安装命令文件列表中选择对应的安装命令，然后选择操作系统，提交测试任务；对于数据库 URL 测试任务，用户上传系统文件，选择适配数据库列表，新建系统测试任务。新建了测试任务以后，系统会有一个模块拉取状态为等待调度的任务（state 为 0），然后批量的执行任务测试，在测试运行过程中对测试 SQL 执行结果进行分析，在测试任务直接结束之后，分析总结所有的执行结果生成测试报告。对于每个测试任务，可以查看测试任务类型，开始执行的时间，结束执行的时间，任务状态，运行完成后生成的测试报告，以及运行过程中的日志。

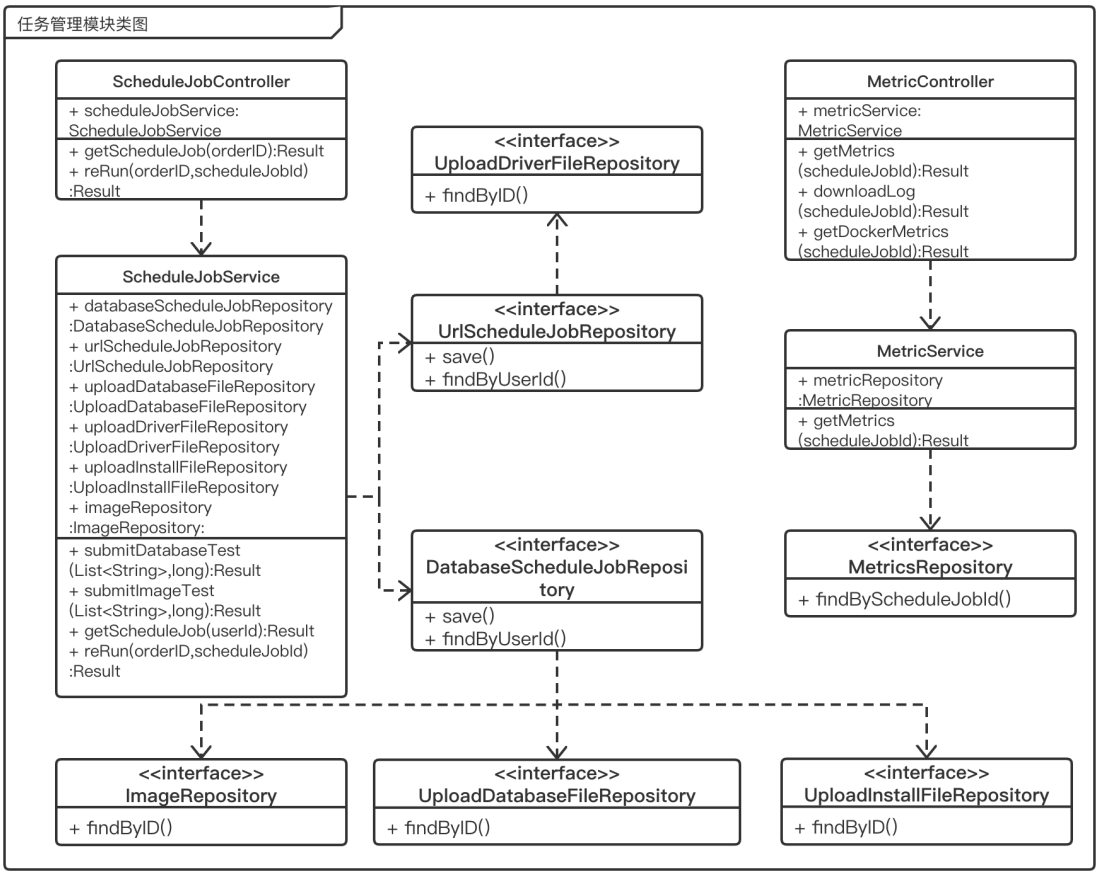

图 4.8: 任务管理模块类图

图4.8展示了任务管理模块核心类图，管理测试任务的请求由 ScheduleController 拦截处理，业务逻辑依赖 ScheduleService 实现，测试任务分为上传数据库测试任务和数据库 URL 测试任务两种类型，分别通过 DatabaseScheduleJobRepository 和 UrlScheduleJobRepository 两个类与数据库管理两种任务记录。同

时，上传数据库测试任务又依赖 ImageRepository 管理数据库运行时的操作系统，UploadDatabaseRepository 来管理所需要测试的数据库文件，UploadInstallFileRepository 来管理数据库安装所需要的命令；数据库 URL 测试任务依赖 UploadDirverRepository 类管理所需要的连接驱动文件。MetricController 供用户获取测试任务结果信息，依赖 MetricService 进行业务逻辑处理，获取 SQL 测试集所生成的结果，和 docker 运行时资源的消耗。

4.3.2　测试任务新建的实现

```
//新建上传数据库测试任务
public Result processDatabaseTest(long imageID, long userID, long databaseId, long installId) {
    boolean success = databaseScheduleJobRepository.save(new DatabaseScheduleJob(userID, databaseId, installId, imageID));
    return Result.ofSuccess(success);
}
//新建数据库 URL 测试任务
public Result processUrlTest(long userID, long driverId, String sqlUrl, String username, String password) {
    boolean success = sqlScheduleRepository.save(new UrlScheduleJob(userID, driverId, sqlUrl, username, password));
    return Result.ofSuccess("运行成功");
}
```

图 4.9: 测试任务新建关键代码

测试任务主要包括上传数据库测试任务 DatabaseScheduleJob 和数据库 URL 测试任务 UrlScheduleJob 两种类型，上传数据库测试任务需要用户从已上传数据库选择待测试数据库，选择已上传安装命令文件，选择需要测试适配的操作系统列表，填入数据库运行命令头，新建上传数据库测试；数据库 URL 测试任务需要用户从已上传连接驱动文件库中选择连接驱动文件，输入测试数据库 URL、用户名密码，新建数据库 URL 测试。测试任务的执行是异步的，与测试任务的提交不绑定，在用户提交了任务之后，系统会通过定时任务自动化的获取待执行任务，并批量执行。本模块主要介绍测试任务的新建，查看详情，查看报告等功能。

测试任务新建关键代码如图4.9所示，对于上传数据库测试任务，新建数据库与操作系统对应的测试执行任务 DatabaseScheduleJob 记录，记录包括，待测

试数据库 Id，安装命令文件 Id，操作系统 Id。对于数据库 URL 测试任务，新建数据库 URL 测试任务 UrlScheduleJob 记录，记录包括待测试数据库 URL、用户名、密码，连接驱动文件 Id。

```
//获取上传数据库测试任务列表
public List<ScheduleJobPO> getScheduleJob(long userID) {
    List<ScheduleJobPO> scheduleJobList = new ArrayList<>();
    //获取用户所有的上传测试任务
    List<DatabaseScheduleJob> databaseScheduleJobs = databaseScheduleJobRepository.findByUserId(UserId);
    //根据执行任务构造展示任务详情
    for (DatabaseScheduleJob databaseScheduleJob : databaseScheduleJobs) {
        //根据测试任务构造 PO
        ScheduleJobPO scheduleJob = ScheduleJobPO.getFromDataBaseJob(databaseScheduleJob);
        scheduleJobList.add(scheduleJob);
    }
    return scheduleJobList;
}
//获取数据库 URL 测试任务列表
public List<UrlScheduleJobPO> getScheduleJob(long userID) {
    List<UrlScheduleJobPO> scheduleJobList = new ArrayList<>();
    //获取用户所有的上传测试任务
    List<UrlScheduleJob> urlScheduleJobs = urlScheduleRepository.findByUserId(UserId);
    //根据执行任务构造展示任务详情
    for (UrlScheduleJob urlScheduleJob : urlScheduleJobs) {
        //根据测试任务构造 PO
        UrlScheduleJob scheduleJob = UrlScheduleJobPO.getFromUrlJob(urlScheduleJob);
        scheduleJobList.add(scheduleJob);
    }
    return scheduleJobList;
}
//获取执行任务的结果列表
public Result<List<MetricPO>> getMetrics(long scheduleJobId, String testType) {
    //直接根据 scheduleJobId 与测试类型获取列表
    List<Metrics> metricsList = meticsRepository.findByScheduleJobIdAndType(scheduleJobId，testType);
    List<MetricPO> metircPOList = metricsList.stream().map(metrics -> new MetricPO(metrics)).collect(Collectors.toList());
    return Result.ofSuccess(metircPOList);
}
```

图 4.10: 测试任务管理关键代码

查看任务列表分为上传数据库任务列表查询和 Url 测试列表查询。可以通过时间，状态等来筛选任务，方便用户查看任务。在 scheduleJob 详情界面，可以选择重试任务，查看结果报告，下载结果报告，下载执行日志等等操作。查看报告与重试任务为前端向后端发送对于的请求，后端返回下载日志是通过前端请求后端获取文件流实现，关键代码如图4.10和4.11所示。而下载报告则是点击了查看报告之后，前端向后端获取数据，并进行页面渲染，最终通过 js 实现 html 输出 pdf 文件，代码较为通用，这里就不展示了。

```
//上传数据库测试任务重试
public void reRunDatabaseTest(long scheduleJobId) {
    //移除原有日志
    fileService.rmDatabaseLogFile(filePath, scheduleJobId);
    //重置任务状态
    DatabaseScheduleJob databaseScheduleJob =
    databaseScheduleJobRepository.findById(scheduleJobId);
    databaseScheduleJob.setState(0);
    databaseScheduleJob.setStartTime(null);
    databaseScheduleJob.setEndTime(null);
    databaseScheduleJobRepository.save(databaseScheduleJob);
    //删除原有的结果列表
    meticsRepository.deleteAllByScheduleJobIdAndType(scheduleJobId,"数据库测试");
    dockerMetricRepository.deleteAllByScheduleJobId(scheduleJobId);
}
//数据库 URL 测试任务重试
public void reRunUrlTest(long scheduleJobId) {
    //移除原有日志
    fileService.rmUrlLogFile(scheduleJobId);
    //重置任务状态
    UrlScheduleJob sqlScheduleJob = sqlScheduleRepository.findById(scheduleJobId);
    sqlScheduleJob.setState(0);
    sqlScheduleJob.setStartTime(null);
    sqlScheduleJob.setEndTime(null);
    sqlScheduleRepository.save(sqlScheduleJob);
    //删除原有的结果列表
    meticsRepository.deleteAllByScheduleJobIdAndType(scheduleJobId,"URL 测试");
}
//下载测试日志
public void fileDownLoad(HttpServletResponse response, long scheduleJobId, String
type) throws IOException {
    //设置消息头
    setHead(response);
    //获取日志路径
    String filePath = fileService.getLogPath(scheduleJobId,  type);
    File file = new File(filePath);
    //写入文件流
    OutputStream os   = response.getOutputStream();
    fileService.writeStream(os, file);
}
```

图 4.11: 测试任务管理关键代码

图 4.12: 新建测试任务界面

测试任务管理列表界面如图4.13所示，上面为上传数据库测试任务管理，下面为 URL 测试任务管理，用户在新建测试任务之后，可以在测试任务管理界面查看测试任务执行状态，如果测试任务已结束，则可以查看对应的测试报告，用户可以对测试任务进行查看详情，查看报告，下载日志，删除任务等操作。

图 4.13: 测试任务记录管理界面

图4.10展示了上传数据库的测试报告图，对于上传数据库测试，是在 docker

操作系统容器中部署数据库，因此我们也实时监控 docker 的状态，测试过程中 docker 的运行状态，cpu 占用，运行时间等总体数据。针对数据库 SQL-92 和 SQL-99 标准，分别有一个测试报告，测试报告记录了对于数据类型，查询关键字，隔离级别等的支持程度，详情里可以查看具体的测试条目，每一行包括数据库功能名，执行 sql 语句，期望输出，实际输出，是否成功，执行所需时间。

图 4.14: 测试报告示意图

测试项	测试语句	期望日志	实际日志	结果
CHAR	CREATE TABLE `test`.`test_char1` (value CHAR(255) NOT NULL PRIMARY KEY) ;			成功
VARCHAR	CREATE TABLE `test`.`test_varchar1` (value VARCHAR(255) NOT NULL PRIMARY KEY) ;			成功
BIT	CREATE TABLE `test`.`test_bit1` (value BIT(255) NOT NULL PRIMARY KEY) ;		ERROR 1439 (42000) at line 1: Display width out of range for column 'value' (max = 64)	失败

图 4.15: 测试项示意图

4.4　测试执行模块详细设计与实现

4.4.1　测试执行模块的详细设计

通过定时任务的方式，测试执行模块获取待执行测试任务并异步执行测试任务，分析测试任务过程中所执行的 SQL 的结果，生成测试报告，该模块对应架构中的 Execute 微服务。该模块核心类图如图4.16所示，ProcessTaskService 负责获取测试任务和执行测试任务，AnalysisService 负责对测试任务执行生成的日志进行分析，从而得到测试报告。此模块依赖 FileService 对文件进行写入与读取，通过数据库相关 Repository 获取与更新对应数据库记录。

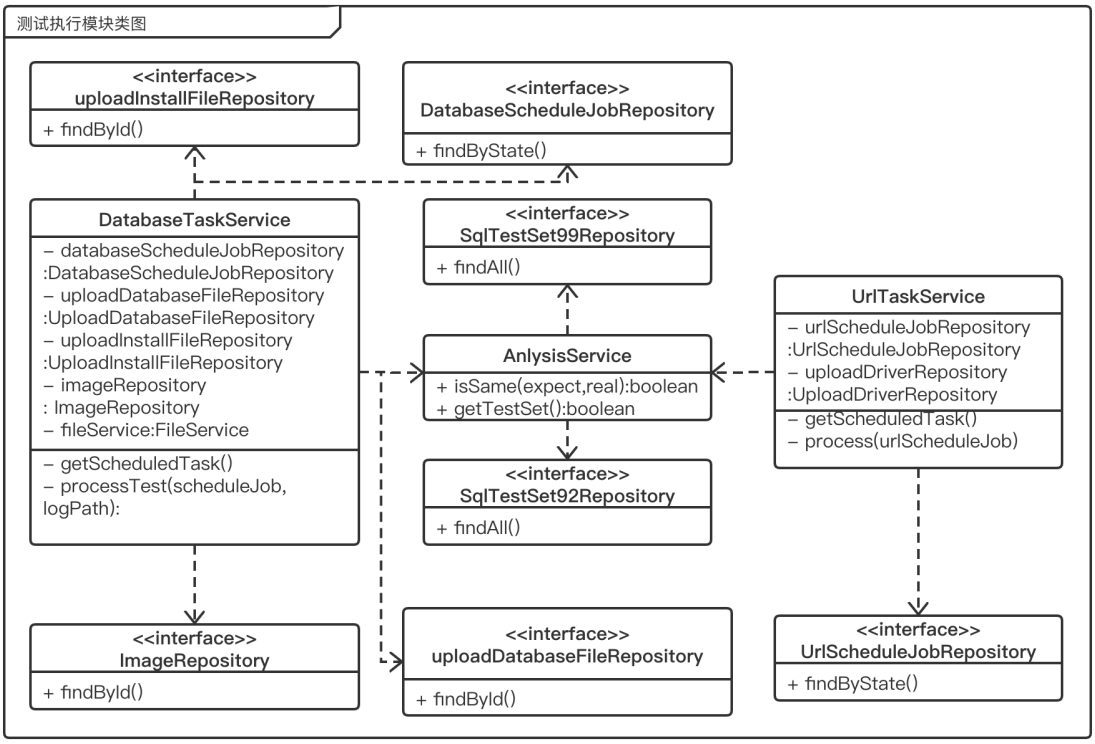

图 4.16: 测试执行模块类图

4.4.2　测试任务获取与执行

测试任务获取与执行关键代码如图4.17所示，对于获取测试任务，系统设置了一个每分钟运行一次的定时任务，定时任务会从 DatabaseScheduleJobRepository 和 UrlScheduleJobRepository 中获取状态为等待调度的数据库测试任务 databaseScheduleJob 和 URL 测试任务 urlScheduleJob，即为 state = 0 的任务，然后对这些任务进行批量的执行，每一个任务新建了一个线程执行，线程通过线程池调度执行。获取调度任务之后，首先将任务状态设置为 1，即正在执行状态，并设置开始时间，防止重复调度。

执行测试任务，如果是上传数据库测试任务，则根据 databaseId 从 uploadDatabaseFileRepository 中获取上传数据库的详细信息和运行数据库命令头信息，根据 installId 从 uploadInstallFileRepository 中获取上传数据库安装命令文件，根据 imageId 从 ImageRepository 中获取适配系统的详细信息。根据获取到的系统信息，首先在 docker 服务器中运行对应的系统，然后根据数据库信息，将数据库复制到 docker 容器中，根据安装命令文件，安装、初始化并运行数据库，得到一个运行在对应操作系统容器的数据库。通过运行数据库命令头，对此数据库运行 SQL 测试集，进行适配性测试，从而得到测试日志，供分析模块获取分析。

```
@Scheduled(fixedRate = 60000)
public void databaseScheduledTask() {
    //获取所有未调度的数据库测试任务
    List<DatabaseScheduleJob> databaseScheduleJobList =
    databaseScheduleJobRepository.findByState(0);
    //处理每一个数据库测试任务
    for (DatabaseScheduleJob scheduleJob : databaseScheduleJobList) {
        scheduleJob.setState(1);
        scheduleJob.setStartTime(new Timestamp(System.currentTimeMillis()));
        databaseScheduleJobRepository.save(scheduleJob);
        //执行测试
        processDatabaseTest(scheduleJob);
        //执行日志分析
        AnalysisLog(fileName, scheduleJob.getId());
        //更新状态
        scheduleJob.setEndTime(new Timestamp(System.currentTimeMillis()));
        scheduleJob.setState(2);
        databaseScheduleJobRepository.save(scheduleJob);
    }
}
@Scheduled(fixedRate = 60000)
public void urlScheduledTask() {
    List<UrlScheduleJob> sqlScheduleList = sqlScheduleRepository.findByState(0);
    for (UrlScheduleJob scheduleJob : sqlScheduleList) {
        scheduleJob.setState(1);
        scheduleJob.setStartTime(new Timestamp(System.currentTimeMillis()));
        urlScheduleRepository.save(scheduleJob);
        //执行测试
        processUrlTest(scheduleJob);
        //更新状态
        scheduleJob.setEndTime(new Timestamp(System.currentTimeMillis()));
        scheduleJob.setState(2);
        sqlScheduleRepository.save(scheduleJob);
    }
}
```

图 4.17: 测试任务获取关键代码

如果是 URL 测试任务，则根据 id 从 uploadDriverFileRepository 中获取上传连接驱动程序的详细信息，并从中 UrlScheduleJob 获取待测试数据库的连接 URL、登录用户名、密码等。通过 java 动态加载类机制，导入连接驱动类，然后通过 Java JDBC 的方式连接登录数据库，运行 SQL 测试集，分析 SQL 运行结果。在测试任务运行结束之后，更新测试任务状态，并持久化到数据库中。

```
/构造上传数据库测试床
public void buildDocker(DatabaseScheduleJob scheduleJob) throws IOException {
    //运行对应系统的 docker 容器，并获取 id
    List<String> result = commandService.runDocker(os);
    //获取 id，如果为空则失败
    if (result.size() == 0) {
        throw new IOException();
    }
    String dockerId = result.get(0);
    //拷贝数据库入系统中
    dockerService.copyDatabase(databaseFilePath);
    //监控 docker 容器
    dockerService.monitor（dockerId);
    //获取数据库安装命令，并在 docker 中执行数据库命令
    List<String> installCommands = fileService.getFileCommand(installFilePath);
    for (String command : installCommands) {
        dockerService.exec(dockerService.getDockerCommand(command, dockerId));
    }
    scheduleJob.setDockerId(dockerId);
    databaseScheduleJobRepository.save(scheduleJob);
}
//构造数据库 URL 测试连接
public Connection getConnect(UrlScheduleJob sqlScheduleJob) throws IOException {
UploadDriver uploadDriver =
uploadDriverRepository.findById(sqlScheduleJob.getDriverID());
    //获取文件
    File file = new File(uploadDriver.getDriverPath());
    if(!file.exists()){
        LOG.info("对应的驱动 jar 不存在.");
    }
    //动态加载类
    URLClassLoader loader = new URLClassLoader(new URL[] { file.toURI().toURL() });
    String driverClassName = uploadDriver.getClassName();
    Driver driver = (Driver) loader.loadClass(driverClassName).newInstance();
    //动态注册驱动
    DriverManager.registerDriver(driver);
    //建立 JDBC 连接
    String url = urlScheduleJob.getUrl)
    String username = urlScheduleJob.getUsername();
    String password = urlScheduleJob.getPassword();
    Connection con = DriverManager.getConnection(url,username ,password);
    return connection;
}
```

图 4.18: 测试任务执行关键代码

4.4.3　测试结果分析

```
ProcessPO isSame(String[] expect, List<String> realResult) throws SQLException {
    List<String> results = getRealLog(realResult);
    boolean success = true;
    if (expect.length != results.size()) {
        success = false;
    } else {
        for (int i = 0; i < expect.length; i++) {
            if (!results.get(i).contains(expect[i])) {
                success = false;
                break;
            }
        }
    }
    ProcessPO processPO = new ProcessPO();
    processPO.setResult(success);
    processPO.setStringBuilder(getString(results));
    return processPO;
}
public void getDockerMetric(long scheduleJobId, String dockerId, Timestamp startTime,
Timestamp endTime) {
    double avgMemory = promService.getAvgMemoryUsage(dockerId);
    double maxMemory = promService.getMaxMemoryUsage(dockerId);
    double avgCPU = promService.getAvgCPUUsage(dockerId, startTime, endTime);
    double maxCPU = promService.getMaxCPUUsage(dockerId, startTime, endTime);
    double fsRead = promService.getFSReadSQL(dockerId);
    double fsWrite = promService.getFSWriteSQL(dockerId);
    DockerMetric dockerMetric = new DockerMetric();
    dockerMetric.setScheduleJobId(scheduleJobId);
    dockerMetric.setAvgCPU(avgCPU);
    dockerMetric.setAvgMemory(avgMemory);
    dockerMetric.setFsRead(fsRead);
    dockerMetric.setFsWrite(fsWrite);
    dockerMetric.setMaxCPU(maxCPU);
    dockerMetric.setMaxMemory(maxMemory);
    dockerMetricRepository.save(dockerMetric);
}
```

图 4.19: 测试结果分析关键代码

日志分析，对于在 docker 容器中部署数据库，需要对安装部署的过程进行日志的监控，从而保证部署的正确性，及时获取安装部署过程中出错信息。日

志分析的算法主要也是基于关键词判断，判断日志中有没有出发错误的关键词，如果触发了错误关键词，则说明过程中很有可能出错了。测试结果分析针对执行 SQL 的结果进行分析和校对。运行测试的过程，即为批量的执行 SQL 的过程，每一条 SQL 包括了 SQL 语句，以及期望日志。对于测试结果分析关键代码如图4.19所示，首先判断 SQL 执行的结果是否报错，如果报错则为执行失败，如果没有报错，则首先获取 SQL 测试集对应的校对 SQL，首先判断是否有校对的 SQL，如果有校对的 SQL 则执行校对 SQL，然后判断执行校对 SQL 的结果是否与预期结果相符，如果没有校对 SQL 且有预期结果，则比较此 SQL 执行的结果是否与预期结果符合，如果没有预期结果，则直接跳过。对于上传数据库测试任务，还需要从 Prometheus 中获取与收集容器在数据库测试期间运行的性能数据，根据对应的资源分析 Prom 语句，获取与分析 CPU，内存，文件等资源的消耗情况，方便用户得知数据库在运行时所消耗的资源详情。

4.5　本章小结

本章详细阐述本系统的详细设计与实现，分为权限管理，文件管理，测试管理和测试执行进行阐述。展示了每个模块的核心类图，通过数据库表设计来阐述系统所涉及的持久化数据的数据结构，通过关键代码来阐述功能的具体实现思路，页面截图展现了系统实现的真实情况。

第五章　测试与分析

为了验证第四章详细设计和实现的效果，证明系统的可用性与性能，本章将描述对本系统的单元测试与性能测试。从测试准备阶段、功能性测试和非功能性测试三个部分来阐述对数据库 SQL 标准适配性自动化测试系统的测试。

5.1　测试准备阶段

5.1.1　测试目标

根据系统功能需求设计功能测试用例，在测试环境中部署并测试系统，包括了以下的测试上传数据库文件、上传连接驱动文件、上传安装命令文件、上传文件管理、新建数据库测试任务、新建数据库 Url 测试任务、查看测试任务、查看及下载测试报告等，通过测试来验证数据库 SQL 标准适配性自动化测试系统是否可以满足功能需求。

5.1.2　测试环境

本系统部署在实验室环境下的物理服务器，系统功能测试环境配置信息如表5.1所示。后端微服务均通过 maven 构建打包部署，通过 Nginx 配置打包部署前端 React 项目。阿里云服务器对前端进行端口映射，使得物理服务器支持公网 IP 访问。

表 5.1: 测试环境

名称	属性
服务器	物理服务器 128GB 内存
操作系统	Ubuntu 16.04
数据库	Mysql 5.7.29
浏览器	chrome 浏览器 90.0.4
支撑库	Nginx 1.19.10、Maven 3.8.1、NodeJS 14.15.4、Docker19.03.13

5.2　功能性测试

基于系统功能需求设计功能性测试用例，每一个测试用例都包含了相应的操作与预期的结果，如果实际结果与预期相符，则证明功能的正确性，如果不相

符，则改进与修复缺陷。功能测试主要包括上传文件，文件管理，新建与执行测试任务，管理测试任务，查看与下载测试报告五个大的测试用例。上传的文件有三种类型，分别是上传数据库文件、上传安装命令文件和上传连接驱动文件。测试任务有两种类型，分别是上传数据库测试和数据库 Url 测试。查看与下载测试报告，只针对测试状态为运行结束的测试任务。

表 5.2: 上传文件测试用例

名称	操作	预期结果	实际结果
上传文件	1. 点击导航栏数据库，安装命令，或者连接驱动文件上传按钮 2. 点击选择文件按钮，选择本地文件上传 3. 点击选择文件按钮，选择新的文件上传 4. 填写备注，文件名，若是数据库文件，还需填写数据库运行命令头；若是连接驱动文件，还需填写动态加载类名 5. 点击提交按钮，确认上传	1. 进入对应的文件上传界面 2. 上传成功后，界面显示已上传文件名 3. 新上传的文件会覆盖旧的，并显示新的文件名 4. 页面显示用户填写内容 5. 系统提示文件上传成功	正确

表5.2描述了文件上传功能的测试用例，分别进行数据库、安装命令和连接驱动上传功能测试，其中覆盖一次只能上传一个文件的功能测试。由于三个测试用例重复性较高，因此这里只描述了一个总体的测试用例，实际的测试是上传数据库，上传安装命令，上传连接驱动三个上传文件测试用例分别进行了测试。三个测试用例都可正确执行，系统对应功能正常。

表 5.3: 文件管理测试用例

名称	操作	预期结果	测试结果
上传文件记录查询	1. 点击导航栏文件管理按钮，分别进入数据库管理，安装命令管理，连接文件管理界面 2. 输入过滤信息后，确认查询	1. 界面跳转至对应的文件管理 2. 文件列表更新为满足过滤条件的文件列表	正确
上传记录详情查看	3. 点击一条记录中的详情按钮	3. 界面显示对应记录详情	正确
上传记录删除	4. 点击一条文件记录的删除按钮	4. 界面刷新，对应记录不显示，提示删除成功	正确
下载文件	5. 点击一条文件记录的下载按钮	6. 建立文件请求流，将文件保存到本地	正确

表5.3描述了文件管理的测试用例，对管理已上传文件记录的功能进行了测试，包括文件查询，删除，下载等功能。与上传文件测试用例描述相似，由于三种文件管理流程相似，因此这里合并进行了描述，但是实际测试是分别对数据库管理，安装命令管理，连接驱动管理进行了测试。三个测试用例都可正确执行，系统对应功能正常。

表5.4描述了新建测试任务的测试用例，对新建两种类型的测试任务进行了测试，包括上传数据库测试任务和数据库 url 测试任务。新建上传数据库测试任务功能，用户首先从已上传数据库中选择待测数据库，从已上传安装命令文件中选择对应安装命令，从操作系统列表中选择部署数据库的操作系统，确认提交；新建数据库 Url 测试任务，需填写待测试数据库连接 Url，登录所需用户名，密码，从已上传连接驱动文件列表中选择对应连接驱动文件，确认提交。测试用例都可正确执行，系统功能正常。

表 5.4: 新建测试任务测试用例

名称	操作	预期结果	测试结果
新建数据库测试任务	1. 点击导航栏中上传数据库测试按钮 2. 点击数据库选择框 3. 选择数据库文件 4. 点击安装命令选择框 5. 选择安装命令文件 6. 点击操作系统选择框 7. 从操作系统列表中选择待部署系统 8. 填写运行命令头等信息，确认新建	1. 页面跳转至新建数据库测试任务 2. 显示已上传数据库命令列表 3. 选择框中显示选择的数据库名 4. 显示已上传安装命令列表 5. 选择框中显示选择的安装命令名 6. 显示数据库中操作系统列表 7. 选择框中显示选择的操作系统名 8. 提示新建成功，进入测试历史界面	正确
执行上传数据库测试	9. 等待测试任务执行结束	9. 数据库测试任务状态变成运行结束	正确
新建 Url 测试任务	1. 点击导航栏中 Url 测试按钮 2. 点击连接驱动选择框 3. 选择连接驱动文件 4. 输入连接 Url，用户名，密码 5. 点击确认按钮	1. 界面跳转至新建 Url 测试任务 2. 显示已上传的连接驱动列表 3. 选择框中显示选择的连接驱动名 4. 界面显示已输入信息 5. 提示新建成功，进入测试历史界面	正确
执行数据库 Url 测试	6. 等待测试任务执行结束	6. 数据库 Url 测试任务状态变成运行结束	正确

表5.5是测试任务管理功能的测试用例，主要验证测试任务的条件查询、查

看子执行任务，以及执行任务的详情查看正确性。由于两类任务的测试管理功能类似，因此这里合并进行了描述，但实际测试是分别对两类任务的测试管理功能进行了测试。测试用例都可正确执行，系统功能正常。

表 5.5: 测试任务管理测试用例

名称	操作	预期结果	测试结果
测试任务查询	1. 点击导航栏测试历史按钮 2. 填写过滤条件之后，确认查询	1. 页面跳转至测试历史，显示测试任务列表 2. 页面刷新，显示过滤之后的文件列表	正确
测试任务详情查看	3. 点击测试任务的详情	3. 弹出测试任务记录详情模态框，展示测试基本信息	正确

表 5.6: 查看测试报告功能测试用例

名称	操作	预期结果	测试结果
下载测试日志	1. 进入测试任务管理界面，点击任务的下载日志按钮	1. 建立文件流连接，保存日志文件到本地	正确
测试报告查看	2. 点击测试任务的测试报告按钮	2. 界面显示任务的测试报告	正确
测试报告下载	3. 点击测试报告界面的下载按钮	3. 前端将 HTML 转换成 PDF，并保存文件到本地	正确

表5.6是查看测试报告功能的测试用例，主要关注在对上传数据库测试任务和数据库 url 测试任务的测试报告查看和下载功能的验证。上传数据库测试与数据库 Url 测试报告查看和下载的流程是一样的，因此这里只进行了一个总体的描述，但是实际测试是分别对两类测试任务的功能进行了测试。测试用例都可正确执行，系统功能正常。

5.3　非功能性测试

表 5.7: 响应时间测试用例

功能名称	平均时间	最大时间	最小时间	测试结果
用户登录	578.3ms	975.3ms	437.2ms	正确
新增数据库上传记录	3149.1ms	3513.4ms	2873.1ms	正确
新增安装命令上传记录	1022.3ms	1712.6ms	765.1ms	正确
新增连接驱动上传记录	1013.7ms	1615.9ms	866.1ms	正确
管理数据库上传记录	1250.7ms	1821.5ms	982.6ms	正确
管理安装命令上传记录	1251.7ms	1823.5ms	979.6ms	正确
管理连接驱动上传记录	1239.7ms	1798.5ms	985.6ms	正确
新建数据库测试任务	1524.2ms	2012.4ms	973.1ms	正确
新建数据库 url 测试任务	1312.1ms	1632.6ms	633.9ms	正确
测试历史查看	912.4ms	1356.1ms	715.3ms	正确
测试任务详情查看	853.1ms	1357.9ms	692.3ms	正确
测试日志下载	2189.5ms	2325.3ms	1021.5ms	正确
测试报告查看	2133.1ms	2516.7ms	1576.9ms	正确
测试报告下载	2365.8ms	2923.7ms	1569.5ms	正确

根据第三章描述的非功能性需求，本章对系统的非功能性测试进行了设计，主要包括请求效率和并发的支持。在系统设计与实现过程中已经考虑到了非功能需求中的可靠性、安全性、可扩展等指标：通过记录系统在运行过程中生成的日志保障系统在出现故障时可追溯；系统的安全性通过权限认证模块对用户的访问验证来保证；通过前后端分离，已经功能模块拆分的方式提高系统可扩展

性；对于比较公用的界面，前端进行封装，使其成为一个组件，并且将用户界面与数据处理分离，保证了前端的可扩展性。

系统的用户界面响应时间通过使用工具 Fiddler 录制了请求，并对系统进行测试，分别对用户登录、上传文件、文件管理，新建测试任务、查看测试历史结果、查看与下载测试报告都进行了测试，每个功能进行 100 次请求测试，并统计其平均响应时间，如表5.7统计结果展示了测试的统计结果。从表中可以看出，上传数据库文件记录是平均耗时最大的，由于数据库文件一般都较大，但是耗时也控制在 4s 以内。除此之外，测试日志下载，测试报告查看，测试报告下载好耗时较大，因为需要请求文件流或者是计算与渲染测试报告，总体时间控制在 3s 以内。其他的核心功能模块，响应时间都在一秒左右。为了保证请求并发性，对系统全部接口使用性能测试工具 Jemter 进行了压测，并发量设置为 100。

5.4　本章小结

为了验证本系统功能的正确性和可靠性，根据第三章中所设计的用例设计了本系统的功能性测试用例。并根据常用的非功能性需求进行了非功能性的测试设计。本章描述数据库 SQL 标准适配性自动化测试系统进行单元测试和性能测试，描述了测试的环境，并针对每一个测试用例的测试目标及执行情况进行了描述。

第六章　总结与展望

6.1　总结

数据库 SQL 标准适配性自动化测试系统，旨在提供数据库运行时满足 SQL 标准的适配性指标，用于指导数据库开发与测试人员。本文研究了数据库 SQL 标准适配性自动化测试系统的设计与实现，进行数据库适配性测试系统的开发。首先介绍了选题的背景和意义，信创软件，可移植性性技术与评估、容器化技术等。然后描述了实现本系统所需要的技术与工具，例如 spring boot 框架，react 框架，docker 容器等。

随后，本文详细分析了数据库 SQL 标准适配性自动化测试技术的功能需求和非功能需求，根据需求分解了用例，并展示了用例图，详细描述了每个用例。然后展示了系统的 “4+1 视图”，介绍了每个模块的实现细节。并介绍了数据库的 SQL 适配性的指标评估体系。本系统的四个模块分别为权限管理模块负责请求的合法性认证、文件管理模块负责上传文件资源管理、测试任务管理负责测试结果的展示和测试执行模块负责测试任务执行，测试报告的生成与导出工作。

最后，本文详细描述每个功能模块的设计与实现，并展示了核心的功能界面，核心模块类图、数据库表设计和实现关键代码，展示了功能实现的细节。在上述工作完成后，基于功能设计并描述了测试用例，确认了系统功能的正确性与可靠性。

本系统提供上传数据库和数据库 Url 两种方式对数据库进行测评，为数据库的开发和测试人员提供测试平台，促进国产数据库的发展。本人主要负责对数据库 SQL92 标准和数据库 99 标准进行分析整理，总结了对应标准的 SQL 测试集，然后进行了本系统的后端和前端的开发的工作。

6.2　展望

本文初步实现了数据库 SQL 标准适配性自动化测试技术，实现了数据库的 SQL 标准适配性测试评估功能，但是系统仍然存在一些不足之处，以下是一些改进的展望：

一。对于 SQL 标准的整理，针对了常用的一些数据库功能与操作设计了 SQL 测试集，但是由于是人工整理，可能存在 SQL 标准覆盖不全的现象，这可能需要更加系统性的整理与总结。

二。本系统通过 docker 工具直接对 docker 容器管理，并没有用 k8s 对 docker 容器进行管理，通过引入 k8s 可以对 docker 容器进行更好的管理，包括存储编排，自动化伸缩，自动化部署等等。

参考文献

[1] 国家技术监督局, 软件工程产品质量（gb-t 16260-2006）.

[2] 周薇, 数据库系统安全性测试技术研究, 计算机技术与发展 34 (2).

[3] J. D. Mooney, Developing portable software, in: R. Reis (Ed.), Information Technology, Selected Tutorials, IFIP 18th World Computer Congress, Tutorials, 22-27 August 2004, Toulouse, France, Vol. 157 of IFIP, Kluwer/Springer, 2004, pp. 55–84.

[4] A. Johansson, J. Svensson, Techniques for software portability in mobile development, Master's thesis (2009).

[5] J. D. Mooney, Strategies for supporting application portability, Computer 23 (11) (1990) 59–70.

[6] D. V. Silakov, A. V. Khoroshilov, Ensuring portability of software, Program. Comput. Softw. 37 (1) (2011) 41–47.

[7] J. D. Mooney, Issues in the specification and measurement of software portability, in: 15th International Conference on Software Engineering, Baltimore, Citeseer, 1993.

[8] H. Ghandorh, A. Noorwali, A. B. Nassif, L. F. Capretz, R. Eagleson, A systematic literature review for software portability measurement: Preliminary results, in: Proceedings of the 9th International Conference on Software and Computer Applications, ICSCA 2020, Langkawi, Malaysia, February 18-21, 2020, ACM, 2020, pp. 152–157.

[9] I. J. M. Ruiz, B. Adams, M. Nagappan, S. Dienst, T. Berger, A. E. Hassan, A large-scale empirical study on software reuse in mobile apps, IEEE Softw. 31 (2) (2014) 78–86.

[10] F. Nayebi, J. Desharnais, A. Abran, An expert-based framework for evaluating ios application usability, in: 2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software

Process and Product Measurement, Ankara, Turkey, October 23-26, 2013, IEEE Computer Society, 2013, pp. 147–155.

[11] C. Strachey, Time sharing in large, fast computers, in: Information Processing, Proceedings of the 1st International Conference on Information Processing, UNESCO, Paris 15-20 June 1959, UNESCO (Paris), 1959, pp. 336–341.

[12] 金海, 廖小飞, 面向计算系统的虚拟化技术, Ph.D. thesis (2008).

[13] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, C. A. F. D. Rose, Performance evaluation of container-based virtualization for high performance computing environments, in: 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2013, Belfast, United Kingdom, February 27 - March 1, 2013, IEEE Computer Society, 2013, pp. 233–240.

[14] 丘诗雅, 基于应用虚拟化技术的安全移动办公解决方案, Ph.D. thesis (2011).

[15] 朱少民, 软件测试方法和技术, 清华大学出版社有限公司, 2005.

[16] Y. Zheng, L. Cai, S. Huang, Z. Wang, VM scheduling strategies based on artificial intelligence in cloud testing, in: 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2014, Las Vegas, NV, USA, June 30 - July 2, 2014, IEEE Computer Society, 2014, pp. 1–7.

[17] P. Mell, T. Grance, et al., The nist definition of cloud computing.

[18] D. D. Chamberlin, Oral history interview with donald d. chamberlin.

[19] M. Chatham, Structured Query Language By Example-Volume I: Data Query Language, Lulu. com, 2012.

[20] E. F. Codd, A relational model of data for large shared data banks, in: Software pioneers, Springer, 2002, pp. 263–294.

[21] R. Johnson, J. Höller, A. Arendsen, T. Risberg, C. Sampaleanu, Professional Java Development with the Spring Framework, John Wiley & Sons, 2007.

[22] H. Suryotrisongko, D. P. Jayanto, A. Tjahyanto, Design and development of back-end application for public complaint systems using microservice spring boot, Procedia Computer Science 124 (2017) 736–743.

[23] 汪恺, 张功萱, 周秀敏, 基于容器虚拟化技术研究, 计算机技术与发展 8.

[24] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. C. Bavier, L. L. Peterson, Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors, in: P. Ferreira, T. R. Gross, L. Veiga (Eds.), Proceedings of the 2007 EuroSys Conference, Lisbon, Portugal, March 21-23, 2007, ACM, 2007, pp. 275–287.

[25] S. J. Vaughan-Nichols, New approach to virtualization is a lightweight, Computer 39 (11) (2006) 12–14.

[26] Y. Zheng, D. M. Nicol, A virtual time system for openvz-based network emulations, in: 25th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation, PADS 2011, Nice, France, June 14-17, 2011, IEEE Computer Society, 2011, pp. 1–10.

[27] 刘思尧, 李强, 李斌, 基于 docker 技术的容器隔离性研究, 软件学报 (4) (2015) 110–113.

[28] R. Dua, A. R. Raja, D. Kakadia, Virtualization vs containerization to support paas, in: 2014 IEEE International Conference on Cloud Engineering, Boston, MA, USA, March 11-14, 2014, IEEE Computer Society, 2014, pp. 610–614.

[29] 杨莎莎, 邹华, 托管 paas 平台安全容器的设计与实现, 软件学报 33 (12) (2012) 1–5.

[30] A. Banks, E. Porcello, Learning React: functional web development with React and Redux, " O'Reilly Media, Inc.", 2017.

致　　谢

版权与原创性说明

作者签名:

日期: 2021 年 05 月 XX 日