火龙果·整理
uml.org.cn

硕士学位论文

基于 Docker 的服务器运维平台的设计与实现

学位申请人:
学 科 专 业 :
指 导 教 师 :
答 辩 日 期 :

万方数据

火龙果·整理
uml.org.cn

摘 要

在高校科研管理信息化建设进程中，数据中心服务器基础架构规模不断扩大，项目研发环境的多变导致服务器运维过程愈加复杂。如何高效的进行服务器运维成为亟待解决的问题。得益于云计算产业的不断发展，且虚拟化技术越来越成熟，服务器运维方式也在发生改变，传统的运维方式已经远远不能满足业务多样性、快速响应的需求。为了提高服务器运维效率，本文提出了一套基于虚拟化技术的运维方案，构建了基于 Docker 的服务器运维平台。

本文首先对国内外服务器运维平台的发展现状进行调研，结合高校科研管理信息化建设过程中的实际情况，整理出平台的业务和具体功能需求。接着提出了平台运维方案并进行了架构设计，将 vSphere 虚拟化管理功能集成到本平台，选用 Docker 容器技术作为应用的运行环境，并通过 Django Web 框架实现了平台业务模块，包括运维管理、应用管理和用户管理三个部分。最后对运维平台做了整体的性能、功能和安全性测试，确保了系统的稳定性、健壮性和安全性。

本文的亮点是引入了两种不同级别的虚拟化技术：vSphere 虚拟化技术和 Docker 虚拟化技术。具体实现使用各类开源的 API 接口并进行了封装，可以做到运维资源一次配置，多次部署，管理方便灵活。经过测试使用，基于 Docker 的服务器运维平台能有效改善数据中心项目研发环境的管理方式和服务器的运维方式，提升了项目研发和服务器运维的效率。

关键词：云计算；虚拟化；Docker；Django；运维平台

万方数据

火龙果 · 整理
uml.org.cn

Abstract

With the development of university scientific research management,The scale of data center infrastructure continues to expand, Project development environment and server operations process is complicated. How to unified management development environment and efficient server operations has become a problem to be solved. Benefited from the continuous development of the cloud computing industry, Virtualization technology is becoming more and more mature, the traditional way of server operations by script already can not satisfy the needs of the business, that develop a server oprations platform based on Docker.

Firstly, the paper conducts research for the status of server operations platform at home and abroad. Combined with the current problem of development of university scientific research management, the paper sort out the needs of server operations platform. Then design the operational platform of technical architecture, vSphere virtualization management functions were integrated into this platform in data center, selects the Docker container technology as the architecture of the application, and through the Django Web development framework implements the platform business modules, including user management, operations management and application management modules. Finally, to ensure the robustness and stability of the platform, we make the overall performance and functional testing.

The highlight of this article is introduced the two kinds of different levels of virtualization technology: vSphere virtualization technology and Docker virtualization technology. The concrete implementation using various open source API interface and encapsulation, can achieve the operational resources configuration, deployment for many times, convenient and flexible management. After testing, server operations platform based on Docker can improve server management and data center operation of way project development environment, to enhance the research and development projects and server operation and maintenance efficiency.

Keywords：Cloud computing, Virtualization, Docker, Django, Operations platform

万方数据

火龙果·整理
uml.org.cn

目 录

摘 要 ..... **I**
Abstract ..... **II**
缩略语 ..... **V**
1 绪论
1.1 课题背景及来源 ..... (1)
1.2 课题现状及分析 ..... (2)
1.3 研究内容及意义 ..... (2)
1.4 本文的组织结构 ..... (3)
2 相关技术简介
2.1 虚拟化技术 ..... (4)
2.2 Web 开发技术 ..... (9)
2.3 持续集成服务 ..... (11)
2.4 本章小结 ..... (12)
3 运维平台需求分析与设计
3.1 总体需求分析 ..... (13)
3.2 总体架构设计 ..... (14)
3.3 数据库结构设计 ..... (23)
3.4 用户管理模块设计 ..... (23)
3.5 运维管理模块设计 ..... (20)
3.6 应用管理模块设计 ..... (29)
3.7 本章小结 ..... (31)
4 运维平台的实现
4.1 开发环境与框架配置 ..... (32)
4.2 环境搭建 ..... (32)
4.3 用户管理模块实现 ..... (34)
4.4 运维管理模块实现 ..... (34)

万方数据

火龙果·整理
uml.org.cn

4.5 应用管理模块实现.....(42)
4.6 本章小结.....(44)
5 平台的测试
5.1 测试方案.....(45)
5.2 测试结果分析.....(47)
5.3 本章小结.....(55)
6 总结与展望
6.1 论文总结.....(57)
6.2 论文展望.....(57)
致 谢 .....(59)
参考文献 .....(60)

万方数据

火龙果 · 整理
uml.org.cn

缩略语

AJAX - Asynchronous JavaScript and XML	异步 JavaScript 和 XML
API - Application Programming Interface	应用程序编程接口
B/S - Browser/Server	浏览器/服务器
C/S - Client/Server	客户端/服务器
CI - Continuous integration	持续集成
CSS - Cascading Style Sheets	层叠样式表
DOM - Document Object Model	文档对象模型
HTML - HyperText Markup Language	超文本标记语言
HTTP - HyperText Transfer Protocol	超文本传输协议
IaaS - Infrastructure as a Service	基础设施即服务
JSON - JavaScript Object Notation	JavaScript 对象表示法
MTV – Model-Template-View	模型-模板-视图
MVC - Model-View-Controller	模型-视图-控制器
PaaS- platform as a service	平台即服务
SaaS - Software as a Service	软件即服务
SDK - Software Development Kit	软件开发工具包
UI - User Interface	用户界面
W3C - World Wide Web Consortium	万维网联盟

万方数据

火龙果 · 整理
uml.org.cn

1 绪论

本课题来源于数据中心的实际项目。随着高校科研管理信息化建设的发展，数据中心服务器集群变得越来越庞大，项目研发环境和服务器运维愈加复杂，数据中心仍然采用终端运行命令行脚本的方式进行配置、部署和管理主机和应用服务。如何通过先进的互联网技术改变相对传统的服务器运维和项目研发测试环境部署过程，提高服务器资源利用率和研发部署效率，是数据中心亟待解决的问题，也是近年来各大企业和研发团队比较关注的一个方向。

本章主要介绍课题相关背景现状，课题的意义以及本文的组织结构。

1.1 课题背景及来源

随着互联网行业的快速发展，信息化技术不断融入到人们生产生活当中。无论是互联网新兴的企业还是传统企业，都在致力于信息化建设，越来越多基于 B/S 架构的 Web 应用不断涌现，由此引发了互联网领域云计算产业的蓬勃发展。

从 2006 年亚马逊公司正式推出弹性云计算[1]服务开始，云计算产业不断发展，形成了三大服务模式，基础设施即服务(IaaS)、平台即服务(PaaS)和软件即服务(SaaS)：

软件即服务（SaaS）：用户不需要处理底层硬件、网络架构和操作系统。服务供应商直接提供可使用的应用程序，用户通过使用已有的程序来实现自己的业务。

平台即服务（PaaS）：用户不需要处理底层硬件、网络架构，只需要使用服务商提供的应用程序环境即可

基础设施即服务（IaaS）：用户不需要处理底层硬件，对网络架构和操作系统及应用有处理权限。

这些服务形式的划分，改变了传统研发部署模式，对互联网行业的研发测试运维以及项目管理带来了一定程度上的技术革新。在服务器运维领域[2]，从业者提出了自动化运维概念，探索了 Web 开发相关技术与传统运维技术的结合，形成了一种新的方向即研发型运维。Docker 是目前比较热门的容器技术，为复杂多样的研发环境提供了便捷的管理方式，本文的研究目的就是结合新兴的互联网技术，通过开发一个 Web 系统解决数据中心服务器运维不便的问题。

本文的研究内容基于教育部相关课题（No.10JZDW004、No.11JF005）和华中科技大学自主创新基金项目（No.2011WA001、No.2013WZ020）。以上课题旨在建设面向全国高校的“科研管理数据库”，整合国内各大高校的科研管理数据，对日常的科研活动进行管理，提高当前高校科研管理水平。本文的研究内容为以上课题的信息化

万方数据

火龙果 · 整理
uml.org.cn

建设和数据中心研发环境和运维提供技术支持。

1.2 课题现状及分析

近年来，国内外云计算领域的平台非常多，下面简要介绍比较知名的一些平台[3]。

1) Google App Engine

Google App Engine 是一个应用托管平台。开发者不需要考虑服务器的物理资源和系统配置，通过预先配置好的应用程序运行环境，上传自己的 Web 应用，就可以为用户提供服务。Google App Engine 通过云计算的技术提高了程序的对负载的适应性。

2) Amazon Web Service

Amazon Web Service 是亚马逊推出的一款平台即服务应用。它支持一键部署应用，Auto scalling，负载平衡、数据库迁移等一系列功能，用户可以灵活选择软件开发平台进行开发，网站流量会随着需求的变化而扩展和缩减。一键部署到世界各地数据中心，为用户提供最优服务。

3) 阿里云

阿里云是国内知名的云计算服务提供商。阿里云拥有自主独立研发的飞天开放平台，负责管理 Linux 集群的物理资源，通过分布式的方式为互联网用户提供服务。

国内外的云计算服务已经开展的如火如荼，这些平台为大量的中小企业和团队提供差异化的服务，提升了运维效率。针对高校数据中心的云计算相关平台仍然在探索阶段，按照当前的发展趋势，传统的服务器运维方式将不能满足将来高校科研管理工作。而数据中心由于数据信息的敏感性，服务器环境独立自主搭建，在运维管理上还处于命令行脚本方式管理，需要改进服务器运维方式，提升研发效率和服务器可维护性。

1.3 研究内容及意义

本课题选择 Docker 技术，将传统服务器应用转移到 Docker 虚拟化环境中，做到应用一次配置，多次部署，管理方便灵活。其次将数据中心 vSphere 虚拟化管理集成到本平台，降低原有虚拟化管理的复杂性[4]。同时将传统命令行脚本管理方式 Web 化，设计了基于 Docker 的服务器运维平台，为开发人员、测试人员、运维人员，项目管理人员提供一个 Web 操作平台[5]。

本课题的设计与实现来源于数据中心的实际需求。随着数据中心相关项目增多，需要管理的服务器环境越来越复杂，尤其是一些项目新技术的使用，使得搭建开发和测试环境越来复杂，运维工作量不断增加。该运维平台将提升数据中心的研发能力和

万方数据

火龙果 · 整理
uml.org.cn

数据中心网站的可靠性以及稳定性。

1.4 本文的组织结构

本论文分为六个章节，每个章节的所要阐述的内容如下：

第一章：绪论。本章节介绍了本课题背景及来源，接着阐述了现状并进行了分析，最后给出了本课题的研究内容及意义。

第二章：相关技术概述。本章节主要介绍了基于 Docker 的服务器运维平台设计与实现中所涉及的相关技术。

第三章：需求分析和概要设计。本章节首先介绍了基于 Docker 的服务器运维平台的总体需求分析，然后进行了总体架构设计，接着介绍了平台各个业务模块的设计。

第四章：平台实现。本章节首先介绍了平台开发环境以及生产环境的搭建过程，并完成了基于 Docker 的服务器运维平台的实现。

第五章：系统测试。本章节首先介绍了基于 Docker 的服务器运维平台的测试方案和业务测试方案，接着根据测试方案进行了测试和测试结果的分析。

第六章：总结和展望。该章对论文进行了总结，然后针对后续工作进行了展望。

万方数据

火龙果 · 整理
uml.org.cn

2 相关技术简介

基于 Docker 的服务器运维平台是一个 Web 系统，旨在对服务器的资源和应用进行管理。整个平台的设计与实现主要涉及到了虚拟化技术和 Web 开发技术[6]。

本章首先对虚拟化技术进行了介绍，这是运维平台的基础，其中介绍了本课题使用的 VMware vSphere 虚拟化技术、Docker 容器技术；然后介绍了 Web 开发框架，这是运维平台的业务功能实现的骨架，按照从整体到局部，从服务器端到客户端的顺序依次介绍了 Web 设计模式、后端技术、Web 服务网关接口、Web 服务代理和前端框架。最后根据应用场景介绍了项目持续集成的相关概念。

2.1 虚拟化技术

现如今的 x86 服务器的设计存在局限性，每次只能运行一个操作系统和若干应用，而且一旦物理机器或者操作系统出现故障，平均修复时间可能是小时级别。如何快速修复服务，这为 IT 部门带来了挑战。另一个方面，考虑到应用灾难冗余，一个数据中心会部署多台服务器，而每台服务器的性能利用率只有 5%到 15%，无论以哪种标准来衡量，都十分的低效[7]。虚拟化（Virturalization）技术应运而生。

虚拟化是将计算机的各个实体资源，如物理机、网络、内存及存储等，予以抽象成各部分统一的资源，使用户可以以更好的方式来使用这些资源[8]。虚拟化技术有硬件的虚拟化和软件的虚拟化两种形式，本课题主要使用软件虚拟化[9]。

2.1.1 VMware vSphere 虚拟化技术

2.1.1.1 VMware vSphere 虚拟化架构

传统的数据中心逐渐向虚拟化架构转变，本课题的基础设施架构采用的是 VMware 公司的 vSphere 产品，原称为 VMware Infrastructure，是一整套虚拟化应用产品[10][11]。vSphere 中的核心组件为 VMware ESXi，在 ESXi 系统层之上可以通过 vSphere client 创建多个 VM（虚拟机），并为这些虚拟机安装 Linux/Windows 系统操作[12]。ESXi 支持硬件虚拟化技术，其虚拟机服务器在性能与稳定性上并不逊色于物理机直接安装的服务器，而且便于统一管理控制，大大提高了服务器管理效率。单台 ESXi 主机的架构如图 2.1 所示：

万方数据

火龙果 · 整理
uml.org.cn

图 2.1 ESXi 主机架构

通过多台 ESXi 主机和底层的存储网络，可以搭建出企业级的虚拟化环境，如下图 2.2 所示：

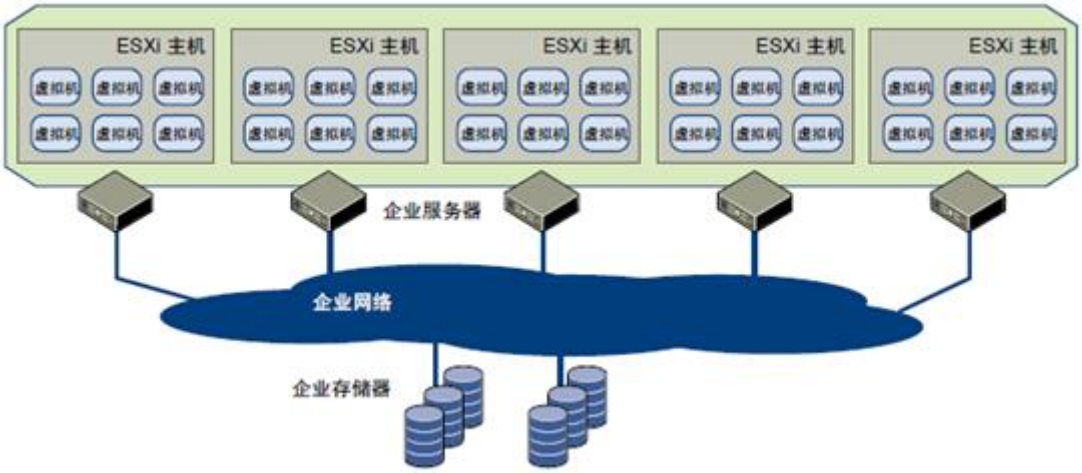


图 2.2 企业级虚拟化环境

vSphere 提供配套的虚拟化管理工具和软件，例如 vSphere client、VMware SDK 以及 vCenter Server 等，让服务器管理和客户端开发更加便捷，整个基础设施层如图 2.3 所示：

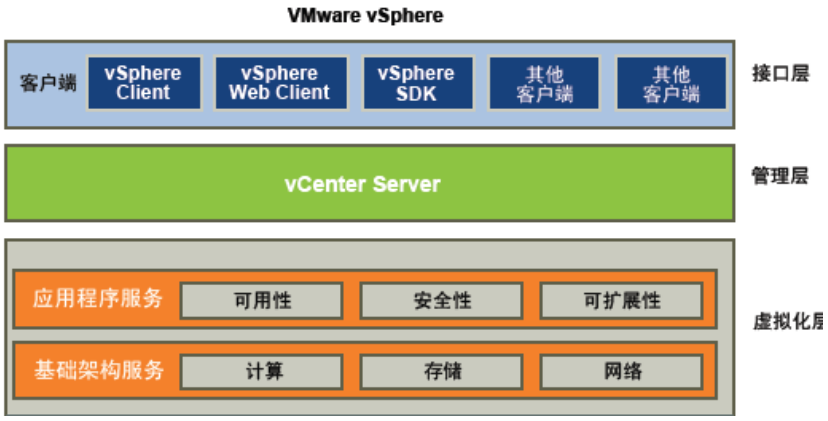


图 2.3 vSphere 虚拟架构

万方数据

火龙果·整理
uml.org.cn

由于 VMware vSphere 仅提供了从物理机到虚拟机的操作系统级虚拟化服务，但对于系统之上的应用服务没有很好的管理功能，不能满足日常研发运维需要，需要抽取出 vSphere 主机管理功能集成到服务器运维平台上统一管理[13]。

2.1.1.2 vSphere 客户端开发

vSphere SDK 是 VMware 公司提供一套开发工具，包括基于不同语言实现的工具库、完善的 vSphere API 和 ESXi 以及 vCenter Server 提供的 Web 服务，用来帮助开发者开发客户端程序。

pyVmomi 是一个开源的，基于 python 实现的 API 工具库[14]，它使用 vSphere API，实现了管理、监控虚拟主机的功能，用以管理基础设施，包括数据中心，数据存储，网络等。本课题的主机管理部分功能使用了该开源工具库。

2.1.2 Docker 容器技术

Docker 是一个开源的引擎，它属于虚拟化技术的一种，可以轻松的为任何应用创建一个轻量级的、可移植的、自给自足的容器。Docker 基于 Go 语言实现，诞生于 2013 年初。Docker 的目标是“Build，Ship and Run Any App，Anywhere”，即通过对应用组件的封装（Packaging）、分发（Distribution）、部署（Deployment）、运行（Runtime）等生命周期的管理，达到应用组件级别的“一次封装，到处运行”。Docker 基于 Linux 的多项开源技术提供了高效、敏捷和轻量级的容器方案，并且支持在多种主流云平台(PaaS)和本地系统上部署，可以说 Docker 为应用开发和部署提供了“一站式”的解决方案，Docker 在开发和运维过程中具有以下几个方面的优势[15]；

1） 更快速的交付和部署。运维人员使用 Docker 镜像为项目构建了一套特定的环境，该镜像打包后可以直接供研发、测试和运维人员使用，这保证了 Web 应用运行环境的一致性。通过快速创建和删除镜像，Web 开发可以进行快速迭代，提高了开发效率；
2） 更高效的资源利用。Docker 容器相当于操作系统的一个进程，容器启动和销毁都在秒级；
3） 更轻松的迁移和扩展。Docker 容器可以多个平台上运行，兼容各种物理机、虚拟机和网络环境，通过镜像打包可以方便迁移到其他平台上运行，多个容器应用配合更加方便；
4） 更简单的更新管理。Docker 允许使用 Dockerfile 文件来配置镜像，只需要修改配置文件就可以创建新的镜像投入使用。

根据 Docker 的技术特点，结合业界的探索，Docker 具有以下几个方面的应用：

1) Web 应用自动化测试和持续集成[16]；

万方数据

火龙果·整理
uml.org.cn

2) 使用在微服务的架构中[17]；

3) 搭建自定义的 PaaS 环境[18]。

Docker 容器技术由镜像、容器、仓库、网络等主要技术组成，下面主要介绍镜像、容器和 Docker API。

2.1.2.1 镜像

Docker 镜像（Image）类似于虚拟机镜像，可以理解为一个面向 Docker 引擎的只读模板。每个 Docker 镜像是由一系列只读的镜像按照一定的层次构成，每个镜像层都有不同的文件内容，用唯一的 ID 号表示。Docker 镜像构成了容器启动时其进程内部可见的文件系统（Rootfs），该文件系统包含容器所需要的系统文件、工具和容器文件。多个镜像的构成的 Rootfs 如图 2.4 所示（不代表实际情况 Rootfs 的内容划分）：

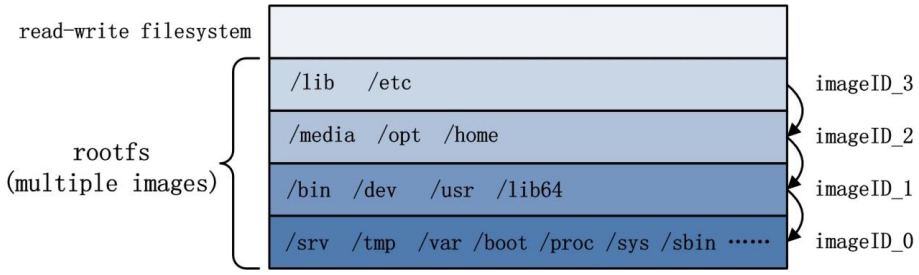


图 2.4 Rootfs 构成

通过镜像的形式，原先较为臃肿的 rootfs 被逐渐打散成轻便的多层。镜像除了轻便的特性，同时还有上文提到的只读特性，如此一来，在不同的容器、不同的 rootfs 中镜像完全可以用来复用。

多镜像组织关系与复用关系如图 2.5 所示（图中镜像名称的举例只为将镜像之间的关系阐述清楚，并不代表实际情况中相应名称镜像之间的关系）。

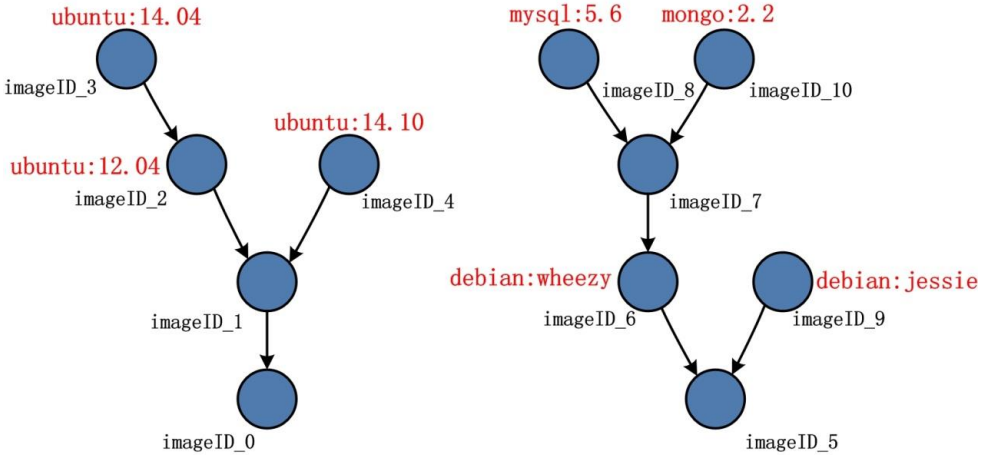


图 2.5 镜像间组织关系

万方数据

火龙果 · 整理
uml.org.cn

2.1.2.2 容器

容器（Container）是通过镜像运行起来的一个实例，类似于一个轻量级沙箱[19]，对容器自身来说，它拥有独立的、完整的最小系统环境，各种命名空间（亦即资源），包括：PID 进程，MNT 文件系统，NET 网络，IPC，UTS，主机名等；对于宿主系统来说，一个容器就是一个系统进程。如图 2.6 描述了容器与宿主操作系统的关系：

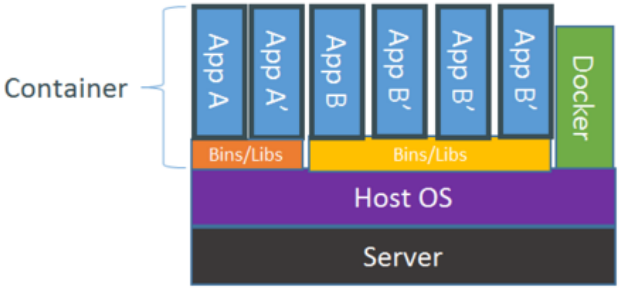


图 2.6 容器与宿主操作系统关系

镜像自身是只读的，容器从镜像启动的时候，Docker 会在镜像的最上层创建一个可写层，方便容器的灵活启动和销毁。图 2.7 描述了容器与镜像的关系：

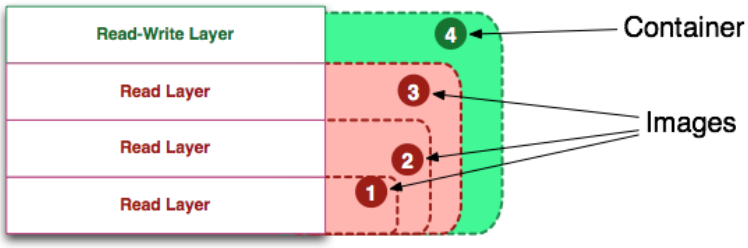


图 2.7 容器与镜像的关系

2.1.2.3 Docker 客户端开发

Docker 是基于 C/S（客户端/服务器）模型开发的，要实现本课题的镜像容器管理等功能，需要了解如何开发 Docker 客户端应用。

Docker-py[20]是基于 python 开发的 API 工具库，在使用 python 作为开发语言的系统中与 Docker 服务器通信。Docker-py 基本包含原生命令支持的功能，本文涉及到的主要有镜像、容器、仓库管理功能。

2.1.3 VMware 虚拟化与 Docker 容器技术的对比

本课题使用了两种虚拟化技术，VMware 虚拟化技术和 Docker 容器技术，他们两者的对比如题 2.8 所示：

万方数据

火龙果·整理
uml.org.cn

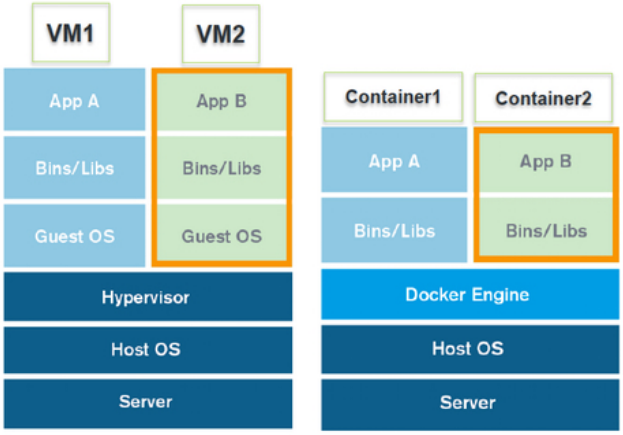


图 2.8 VMware 虚拟化与 Docker 容器技术的对比

本课题的基础设施层是基于 VMware vSphere 虚拟化技术，而应用层是基于 Docker 容器技术，前者是面向操作系统层，借助 Hypervisor 层将底层硬件资源抽象化，供虚拟操作系统使用，应用则部署运行在客户操作系统之上，不仅系统性能开销大，资源消耗也较多；而后者是面向应用层，直接在宿机主平台加载运行应用程序，使用 LXC（Linux Containers）技术使得进程间互相隔离[21]，在性能上接近宿主机的进程。如表表示虚拟机与容器以及传统物理服务器方式的对比：

表 2.1 虚拟机与容器以及传统物理服务器方式对比

特性	容器	虚拟机	物理机
部署耗时	一次下载或构建秒级部署	克隆分钟级、安装几十分钟	小时
启动速度	秒级	分钟级	分钟
硬盘使用	一般为 MB	一般为 GB	一般为 GB
性能	接近原生	弱于	原生
系统支持数量	单机支持上千个容器	一般几十个	一个
隔离性	安全隔离	安全隔离	安全隔离

2.2 Web 开发技术

本课题旨在开发一个 Web 平台，该平台的实现需要 Web 服务器后端技术包括 Web 框架，Web 服务器网关接口，前端框架和 Web 代理服务器共同完成。

2.2.1 Web 设计模式

MVC 模式[22]（Model-View-Controller）是软件工程的一种软件架构模式。它由三个重要部分组成：

万方数据

火龙果·整理
uml.org.cn

Model：封装数据及逻辑。将数据库实例对象封装成对象供 View 层调用；

View：查询数据并展现给用户。用于管理信息的显示；

Conctroller：截获用户请求并改变数据。

Django 框架使用一种叫做 MTV 的模式，实质上仍然是 MVC 模式[23][24]的一种。其中 MVC 模式下的控制器通过 URLs 控制，Django 框架关注模型（Model）、模板（Template）和视图（View），下面简要叙述下 MTV 模式下用户与服务器交互的过程。

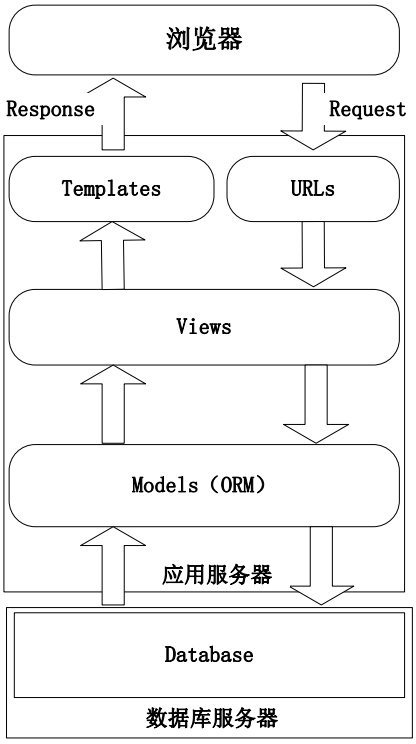


图 2.9 MTV 模式

如图 2.9 所示，用户通过浏览器发送请求到应用服务器，Django 框架根据 URLs 判断用户的请求路径，发送给对应的 View，这里 View 就是业务层，根据业务逻辑获取所需要的数据，数据通过 Model 层来获取，Model 层封装了数据库中各类实例对象。数据通过 Model 层返回给调用者 View，View 层将数据封装到 Template 中返回给浏览器进行解析。至此，用户完成一次完整的交互过程。若其中某一环节出现问题，则需要有良好的异常处理机制保证整个流程的完整性[25]。

2.2.2 Web 后端框架

基于 Python 语言有大量优秀的开源 Web 开发框架如 Django、Web.py、flask 等等[26]，开发者根据技术需求使用这些框架能够起到事半功倍的效果，这里我们使用 Django 框架来进行运维平台的开发[27]。

Django 是一个鼓励快速开发与简洁、务实设计的高层 Python Web 框架。它使用

万方数据

火龙果·整理
uml.org.cn

MTV 的 Web 架构，页面采用模板的方式，内置了大量方法使得模板更加易用；业务逻辑部分采用 url 路由请求分发，通过 view 来处理业务逻辑，可以方便调用基于 Python 的各种应用程序类库；数据库部分采用良好的 ORM 机制方便开发者快速开发应用[28]。

2.2.3 Web 服务网关接口

Web 服务器网关接口（Web Server Gateway Interface，缩写为 WSGI），它基于现存的 CGI（Common Gateway Interface）标准而设计的[29]。

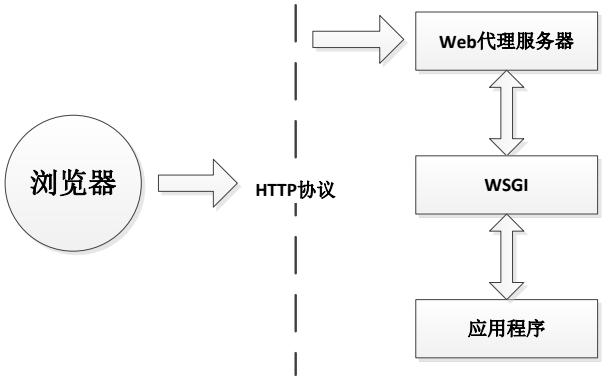


图 2.10 WSGI 通信过程

在处理一个 WSGI 请求时，服务器会为应用程序提供环境变量和回调函数（callback function），当应用程序处理完成请求后，通过回调函数，将结果返回给服务器。本文使用的是 uwsgi，它是一个快速的、纯 C 语言开发的、自维护、对开发者友好的 WSGI 服务器，旨在提供专业的 Python Web 应用发布和开发功能。

2.2.4 Web 服务代理

Web 服务器通过监听服务器端口，响应用户请求，提供多种插件与后端应用程序进行通信。Nginx 是一款面向性能设计的 http 服务器，同时也是一款反向代理服务器、负载均衡服务器和缓存服务器，相较于 Apache 具有占有内存少，稳定性高等优势[30]。不同于 Apache 为每一个请求创建一个线程，Nginx 使用异步逻辑，所以负载均衡能力强大。整体采用模块化设计，有丰富的模块库和第三方模块库，配置灵活[31]。

2.2.5 Web 前端框架

服务器运维平台的前端页面采用 bootstrap 和 jQuery 来完成。bootstrap 是一款出色的前端界面框架，它具有平滑的界面元素，直观的显示控件和强大的自适应能力。在多个浏览终端、不同操作系统平台和不同品牌的浏览器上具有较好的界面展示效果；在页面的交互上使用 jQuery，jQuery 是一个快速，小巧，功能丰富的 JavaScript 库[32]。它使 HTML 文档的遍历和操作、事件处理、动画和 ajax 请求更加易用[33]。

万方数据

火龙果 · 整理
uml.org.cn

2.3 持续集成服务

持续集成（Continuous integration，简称 CI）是指频繁（一天多次）的将项目代码集成到主干[34]。目的是让项目可以快速迭代，同时还能保证高质量[35][36]。Martin Fowler 说过，"持续集成并不能消除 Bug，而是让它们非常容易发现和改正"。

持续集成的好处：

（1）快速修正运行中的错误，提高系统可用性；

（2）防止分支大幅偏离主干，长期没有集成是会导致后期集成困难。

目前提供持续进程的开源项目非常多，有 Jenkins、Buildbot、Travis CI 等，本文主要使用 jenkins 作为持续集成服务器，结合 Docker 容器技术的优势，将部分持续集成功能引入到本平台。

2.4 本章小结

本章主要介绍了基于 Docker 的服务器管理平台的设计和实现过程中需要用到的相关技术，其中虚拟化技术主要使用了 Docker 容器和 VMware 虚拟化；Web 开发框架中介绍了 Django 框架，选用了 uwsgi 作为服务器网关接口、前端展示框架、Nginx 作为 http 代理服务器。最后介绍本文需要用到的持续集成的概念。

万方数据

火龙果 · 整理
uml.org.cn

3 运维平台需求分析与设计

需求分析和设计是系统实现过程的重要环节，本章将前期进行的调研结果和数据中心实际情况结合，给出了服务器运维的定义，并从几个典型的业务场景中进行详细的需求分析，然后根据需求分析的结果进行基于 Docker 的服务器管理平台总体架构设计、数据库设计、用户管理模块设计、运维模块设计和应用管理模块设计。

3.1 总体需求分析

本文所要解决的主要问题是运维，这里特指互联网服务器运维。运维的职责覆盖了产品从设计到发布、运行维护、变更升级及至下线的生命周期，不仅仅是传统的 IT 运维。

目前数据中心服务器运维工作均是通过登录服务器执行命令行脚本的方式完成。由此积累了大量的自动化脚本，这些脚本分散，缺少统一管理和业务功能上的连贯性，不够直观，操作起来工作量大还容易出错；同时又由于中心项目逐渐增多，传统的项目环境搭建和后期维护效率低下。

本课题结合以上情况，调研了目前云计算产品，充分挖掘产品功能，结合数据中心的日常服务器管理需求进行具体的需求分析。从市场上各类 PaaS 产品来看，它们都基于一定的业务场景进行设计，例如为解决用户无法直观管理服务器，设计了控制面板类 Web 应用，用来查看和管理服务器[37]；例如为了解决 Web 应用发布的问题，提供了自动化发布的 Web 应用，方便管理应用发布的工作[38]。在数据中心也有很多场景值得分析，下面分析三个典型的应用场景。

场景一：项目研发环境的运维

随着 Web 项目增多，项目组在沟通好项目需求之后，开发人员就需要针对项目需求，进行技术选型，完成项目的技术架构设计。接下来需要着手项目基础环境搭建、创建代码库、配置研发测试环境和服务器环境，最后将研发环境中的应用发布到服务器上。应用发布上线时需要检出代码，形成 Web 系统安装包，通过手动或者运行半自动化脚本发布到服务器上的 Web 容器中。此外针对应用系统的生产环境和测试环境的不同，还需要设置不同的数据源和搭建测试环境。

场景二：运维人员管理服务器

目前运维人员日常工作中主要集中在研发人员环境的安装和配置、服务器环境的安装和配置、服务器管理这三个方面。研发人员加入研发团队时需要配置虚拟研发环境，为保证研发环境一致性，这里使用虚拟机模板克隆虚拟机的方式进行虚拟机创建。

万方数据

火龙果·整理
uml.org.cn

需要一个 Windows 或 Linux 操作系统的模板，模板上已经安装了操作系统、集成开发环境、相关的编译软件和其他应用软件。克隆完成虚拟机后还需要创建账号、配置权限和分配 ip；通过模板克隆服务器操作系统后，需要配置操作系统 ip 地址，安装应用，做好应用环境的初始化工作；在服务器管理上，会有需要批量执行脚本命令需求，在其他情况下需要登录服务器查看服务器运行异常情况，更新软件或者调整应用等。

场景三：项目人员关系复杂，项目成员包括项目主管，研发人员，测试人员，数据库人员和运维人员等。各类人员各司其职，其中管理人员可以对项目进行立项，创建代码仓库，对项目组成员进行管理，为项目配置项目组，为不同的组配置权限。项目组组长可以更改项目组成员[39]。而现有的环境下，这些操作都需要运维人员手动登录到相应服务器上执行命令完成，没有一个管理系统对人员角色与权限作出规范。

根据这些业务场景的描述，可以得出以下几个方面的需求：

1) 研发环境和部署环境的自动化管理

研发环境的创建会在研发、运维人员研发和运维过程中反复进行，影响研发运维效率，而且通常情况下创建的环境使用率并不高，故需要有一个高效创建项目运行环境的平台，提供模板环境的创建、使用和删除。需要对项目部署过程进行自动化处理，减少项目发布的时间，提高发布频率使得 Web 应用能更快的改善缺陷。

2) 主机资源集中管理

需要将主机的一些常见功能集中起来，例如批量执行命令、为 Linux 服务器创建用户、给研发人员创建开发机和 Linux 应用服务器，以此提研发和运维效率。

3) 项目成员角色权限管理

需要将项目成员集中起来管理，通过角色权限系统，方便修改项目组已经项目组成员的权限和角色信息，减少运维人员的手动操作，提高项目开发效率。

4) 使用 Web 交互方式实现功能

为了便于各类人员使用和兼容不同的浏览器平台和操作系统，需要设计一个基于 B/S 架构的运维平台。

3.2 总体架构设计

根据 3.1 节的总体需求分析，本节进行了总体架构的设计，主要从平台运维管理方案设计、运维管理架构设计、Web 架构设计和平台功能架构四个部分进行介绍。

3.2.1 运维平台方案设计

需求分析中提出的研发环境和部署环境的自动化管理以及主机资源的管理，这些都涉及到运维的范畴。为了提高研发环境的创建的简便性和可重复利用性，本文选择

万方数据

火龙果·整理
uml.org.cn

了 Docker 技术（见 2.1.2 节的 Docker 容器技术介绍），运维人员首先创建基本功能的 Docker 镜像，然后从这些镜像创建容器，通过容器启动时参数的调整和配置来自定义容器，使之适合不同场景下的需求。例如运维人员创建了一个 svn 代码管理应用的镜像，项目管理员创建项目时会使用该镜像，通过传入项目名称，项目组权限信息从该镜像启动一个项目专属的代码容器，项目成员就可以根据仓库地址提交和更新代码了。由于 Docker 的技术优势，容器的创建过程非常简单和迅速，而且可以根据不同的项目需求多次创建不同的项目容器，项目容器之间完全隔离，互不影响[40][41]。Docker 镜像与容器的运维方案如图 3.1 所示：

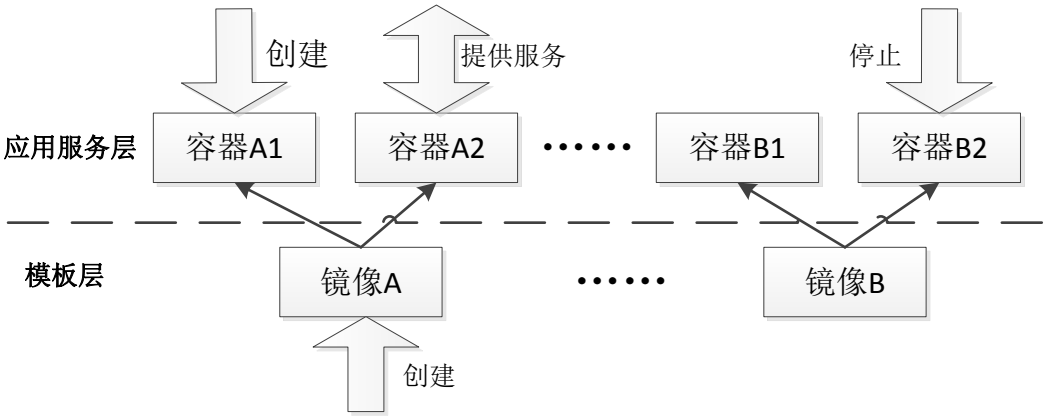


图 3.1 镜像与容器的运维方案

首先用户创建镜像，该镜像安装有具体的应用程序。多个不同应用程序镜像构成模板层，提供给用户研发运维过程使用。具体使用时，用户通过镜像创建容器，运行着的容器作为应用服务层提供各类应用服务。运维管理过程就是对容器和镜像资源进行管理。

针对主机资源的管理，数据中心依托 VMware vSphere 虚拟化技术，这里需要提供一个与虚拟机管理服务器通信的接口和与 Linux 服务器通信的方法，本文选用开源工具 pyvmomi 与虚拟机管理服务器交互，选用基于 SSH 协议的开源工具库 paramiko 与主机进行交互，并将它们包装成接口供运维平台服务层调用。主机运维管理方案设计如图 3.2 所示：

万方数据

火龙果·整理
uml.org.cn

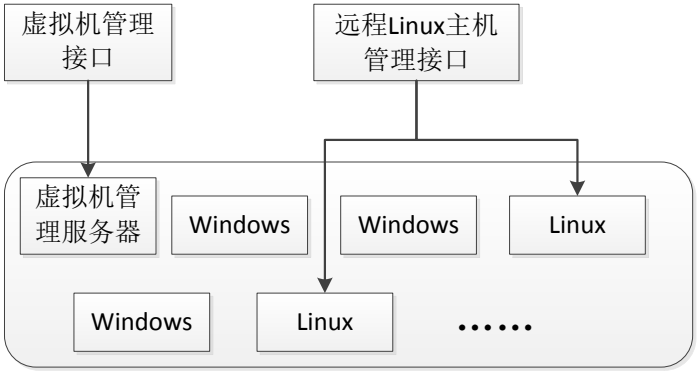


图 3.2 主机运维管理方案

由图 3.2 可知，虚拟机管理服务器管理所有虚拟机，包括 Windows 和 Linux 操作系统。虚拟机管理接口负责与虚拟机管理服务器进行通信，远程 Linux 主机管理接口负责与 Linux 虚拟主机通信。两者整合共同完成主机运维管理功能。

3.2.2 运维平台架构设计

根据需求分析和上一小节介绍的运维平台方案设计，可以知道主机管理主要提供服务器主机级别的支持；而研发、运维环境的管理主要是针对 Docker 服务器上的镜像以及容器资源进行统一管理。整合两个管理功能，设计如图 3.3 所示的运维平台架构：

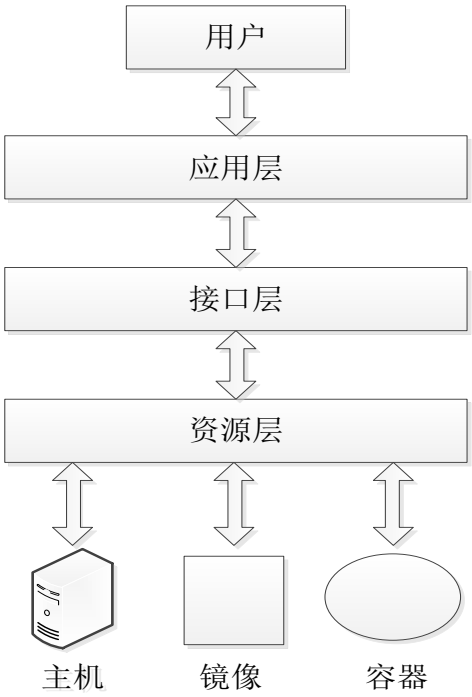


图 3.3 运维平台架构

主机管理主要依赖现有数据中心 VMware vSphere 虚拟化架构（见 2.1.1 节描述的

万方数据

火龙果·整理
uml.org.cn

VMware vSphere 虚拟化技术），而容器技术则是在虚拟主机上如图 3.3 所示，整个平台从上至下分为三层：应用层、接口层和资源层。下面分别介绍：

1) 应用层是集成所有运维平台的功能入口；
2) 接口层是根据不同的管理对象设计的接口进行的抽象；
3) 资源层是所有运维平台所管理的资源的抽象，包括主机、镜像和容器。

3.2.3 Web 架构设计

为了让用户使用更加方便的使用运维平台，运维平台采用普遍的 B/S 结构，浏览器端通过 Http 接口与服务端通信，服务端采用 Django 框架，前端采用 jQuery +bootstrap 的前端开发框架，数据库采用 MySQL，平台的整体技术架构示意图如图 3.4 所示。

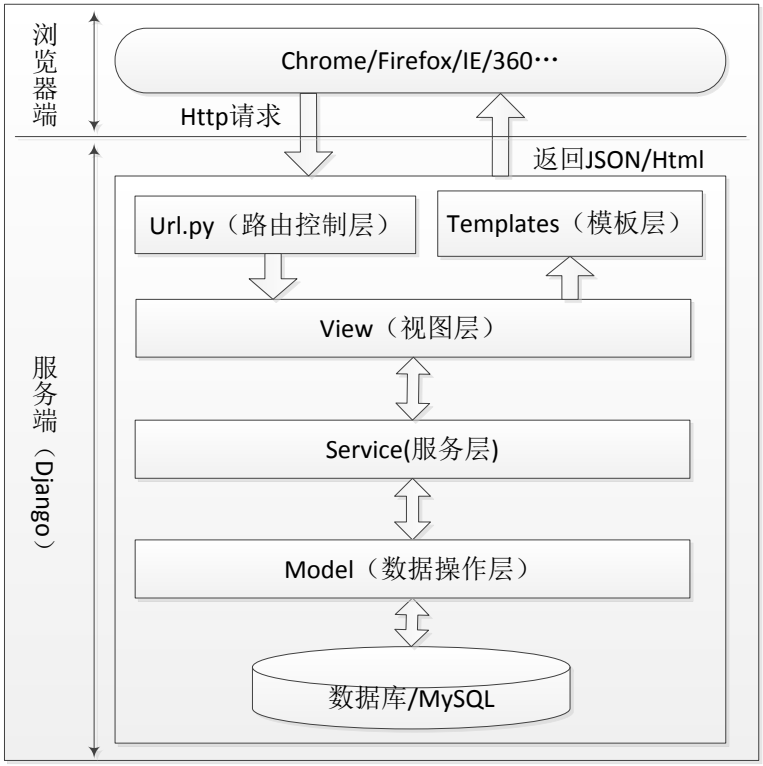


图 3.4 平台 Web 技术架构

服务端采用“路由控制-视图-服务层-数据操作层-模板层”的设计模式，具体说明如下：

1) 路由控制层：负责响应用户的请求，根据用户请求的 url 地址将用户请求内容分发到视图层的不同业务逻辑处理；
2) 视图层：平台所有功能的响应接口，根据不同的请求调用相应的服务层完成用户请求，将服务层返回的数据填充至模板层发送给客户端；
3) 服务层：为所有业务功能提供支持，在涉及数据库操作时会调用数据操作层

万方数据

火龙果·整理
uml.org.cn

来进行数据库读写操作，而在进行资源处理的业务时会调用相应的应用程序接口；

4) 数据操作层：封装了运维平台管理对象，如主机、镜像和容器，还封装了平台的用户、角色、权限以及项目组。

3.2.4 功能架构设计

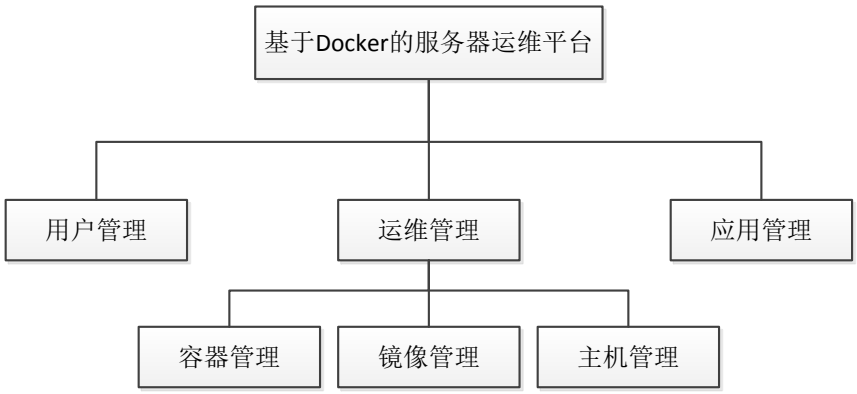


图 3.5 功能架构

整个服务器运维平台在功能架构设计如图 3.5 所示，分为用户管理、运维管理和应用管理和三大部分。用户管理是运维平台的基础功能，涉及用户登录、注册和找回密码等基本功能，还包括用户角色、权限和项目组的管理等高级功能。运维管理的管理对象是运维平台的基础软件资源，包括容器、镜像、主机这三种资源。应用管理则是整合底层资源，针对研发和运维场景下的常见应用和服务进行管理，主要有代码库管理和持续集成服务的管理。各个部分简要介绍如下：

1) 用户管理

用户管理部分具体功能设计如图 3.6 所示：

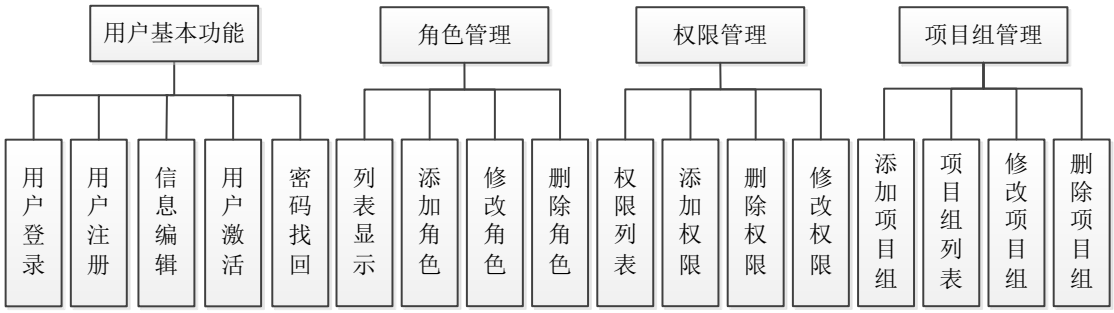


图 3.6 用户管理模块

整个用户管理具体分为四个部分，分别是用户基本功能、角色、权限和项目组的管理。各部分功能概要介绍如下：

用户基本功能管理，主要包括有登录、注册、信息编辑用户激活和密码找回；

角色管理功能包括列表显示，添加角色、修改角色和删除角色；

万方数据

火龙果·整理
uml.org.cn

权限管理功能包括列表显示，添加权限、修改角色和删除权限；

项目组管理功能是针对项目分配人员，有添加项目组，成员添加，项目组列表和组成员列表。

2) 运维管理

运维管理具体功能设计如图 3.7 所示：

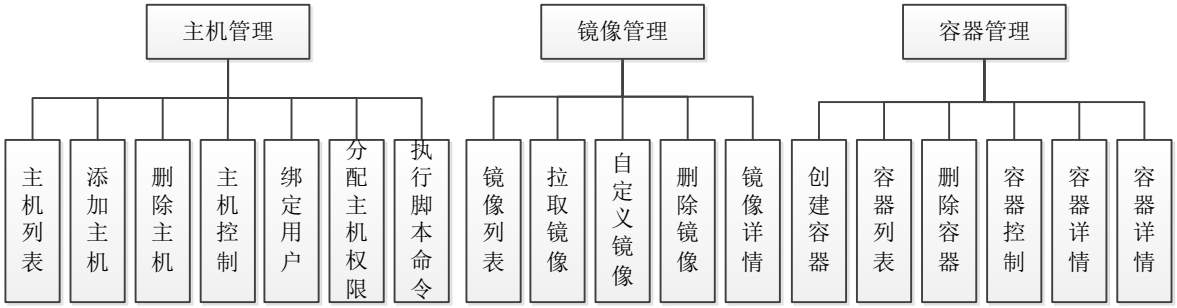


图 3.7 运维管理模块

整个运维管理模块具体分为三个部分，分别是主机管理、镜像管理和模块管理。各部分功能概要介绍如下：

主机管理功能的管理对象是现有的虚拟主机和物理主机，包括的功能有主机的列表、添加主机、删除主机、主机控制、绑定用户、分配主机权限和执行脚本命令。

镜像是容器运行的基础，镜像管理对用户创建的镜像进行管理，包括镜像列表的查看、通过 pull 方式拉取镜像、通过 dockerfile 自定义镜像、删除镜像和查看镜像详情功能。

容器管理是管理用户创建的应用运行实例，包括容器列表、创建容器、删除容器、容器的状态控制、查看容器详情和日志功能。

3) 应用管理

应用管理功能具体设计如图 3.8 所示：

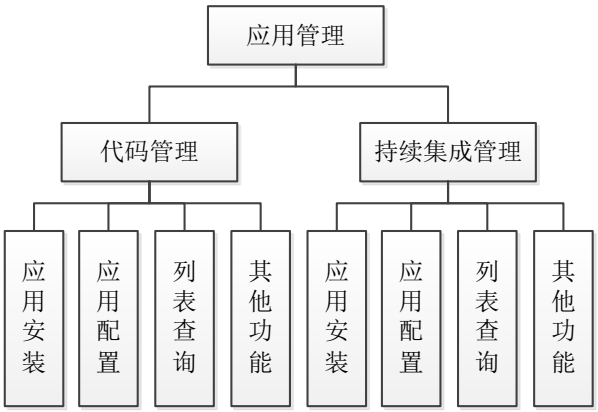


图 3.8 应用管理功能

万方数据

火龙果·整理
uml.org.cn

目前本文在应用管理部分仅设计两个部分，一是代码管理功能，二是持续集成管理功能。他们是项目研发、运维过程中比较重要的两个方面。项目创建之初，需要创建项目代码仓库供研发人员更新和检出代码；在持续集成服务上，需要运维人员配置管理项目部署方案。

3.3 数据库结构设计

3.3.1 数据库实体设计

根据需求分析和功能设计，设计实体以及实体之间关系如图 3.8 所示：

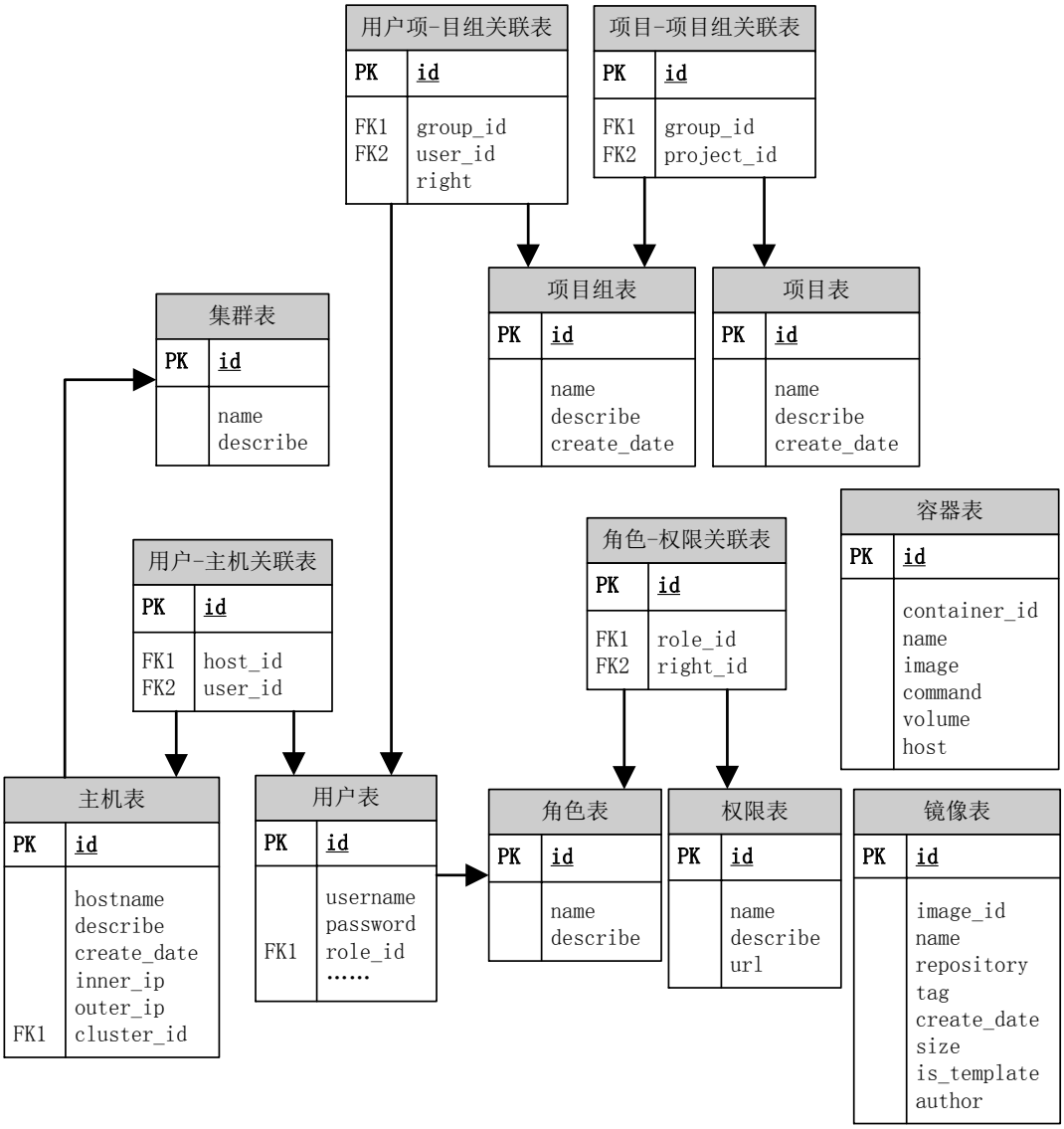


图 3.9 实体以及实体间关系

1) 用户实体：用户表是根据用户的各种身份和信息属性抽象出来的，记录有用

万方数据

火龙果·整理
uml.org.cn

户的账号、密码、邮箱、审核状态、是否超级用户、注册时间、过期时间等；

2) 角色实体：角色表是用户角色信息的抽象，包括角色名称、描述和 id；
3) 权限实体：权限表是根据用户角色所对应的权限的抽象，包括权限 id、名称、描述和内容，平台采用访问 url 作为权限的粒度划分；
4) 项目组实体：项目组表是若干人员为某一项目成员集合的抽象，包括有项目组 id、项目组名称、项目组成员；
5) 项目实体：项目表是对项目信息的抽象，包括项目 id、名称、描述等；
6) 主机实体：主机表是对主机属性的抽象，包括主机 id、主机名称、内网 id、外网 id、主机创建时间等；
7) 集群实体：集群表是对主机集合的抽象，若干主机组成一个功能集群，该实体包括集群 id、集群名称、集群描述；
8) 镜像实体：镜像表是 Docker 镜像资源的抽象，包括镜像 id、镜像名称、镜像创建时间、镜像创建作者、镜像描述、镜像所在仓库地址、镜像标签信息、镜像大小等；
9) 容器实体：容器表是对 Docker 容器资源的抽象，包括容器 id、容器名称、容器创建时间等。

根据需求，不同实体之间拥有不同的对应关系，用户和角色间存在一对一关系，角色和权限之间存在多对多关系，主机和用户之间存在多对多关系，主机和主机集群间存在多对一的关系，用户与项目组之间存在多对多关系，项目组与项目之间存在多对多关系。下面仅介绍用户、角色、权限、项目与项目组之间的实体关系，如图 3.10 所示。其他实体间关系设计类似，不再赘述。

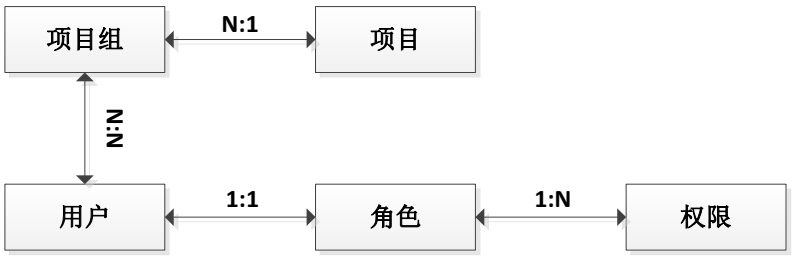


图 3.10 用户、角色、权限、项目与项目组之间的实体关系

一个用户只拥有一种角色，用户与角色之间是一对一的关系；每种角色都拥有一定的权限，角色与权限之间是一对多的关系；一个项目组由多名项目成员组成，用户会同时参与到不同项目组，项目组与用户之间是多对多的关系；一个项目可能会被多个项目组管理，例如项目管理员组拥有多个项目的参与权限，项目与项目组之间是一对多的关系。以上实体关系在数据库中均有中间实体对应，该实体设计如表 3.1 所示

万方数据

火龙果·整理
uml.org.cn

表 3.1 中间实体设计：

表 3.1 中间实体设计

键	数据类型	描述
id	int	关系唯一标识
A_id	int	实体 A 的 id
B_id	int	实体 B 的 id

其中 A_id 和 B_id 分别是实体 A 和 B 的主键。

3.3.2 数据库表设计

根据功能需求和上一小节的数据库实体设计出数据库表，本小节介绍主要的几个表的设计：

1) 用户表设计如表 3.2 所示：

表 3.2 用户表字段设计

键	数据类型	描述
id	int	用户唯一标识
email	varchar	电子邮件地址
username	varchar	账号名
password	varchar	登录密码
role_id	int	用户角色
last_login	datetime	上一次登录时间
join_date	datetime	账号创建时间
expire_date	datatime	账号过期时间
reset_password_token	varchar	密码重置令牌
reset_password_expires	datetime	密码重置过期时间
is_active	boolean	是否激活
is_superuser	boolean	是否是超级管理员

2) 角色表设计如表 3.3 所示：

表 3.3 角色表字段设计

键	数据类型	描述
id	int	角色唯一标识
name	varchar	角色名
describe	varchar	描述

其他表诸如项目表、项目组表字段设计与角色表字段设计类似，都是三个字段的形式：id 作为唯一标示，name 作为命名标志，describe 作为 name 的描述字段。权限表设计与他们的不同之处在与增加了一个权限路径的字段，该字段用来与用户请求路径进行匹配。具体表字段设计不再赘述。

万方数据

火龙果·整理
uml.org.cn

3) 主机表设计如表 3.4 所示：

表 3.4 主机表设计

键	数据类型	描述
id	int	主机唯一标识
hostname	varchar	主机名称
innerIP	varchar	内网地址
outerIP	varchar	外网地址
describe	varchar	描述
createdate	datetime	创建时间
cluster_id	int	所属集群 id

4) 镜像表设计如表 3.5 所示：

表 3.5 镜像表设计

键	数据类型	描述
id	int	镜像数据库中唯一标识
image_id	varchar	镜像在 Docker 服务器中 id
describe	varchar	描述信息
registery	varchar	镜像所在注册服务器
repository	varchar	镜像仓库
tag	varchar	镜像标签
create_date	datetime	创建时间
size	int	大小
is_template	boolean	是否是模板镜像
auther	varchar	作者

3）容器表设计如表 3.6 所示：

表 3.6 容器的表字段设计

键	数据类型	描述
id	int	容器在数据库中唯一标识
container_id	varchar	容器在 Docker 服务器中 id
describe	varchar	描述信息
image_id	int	容器所依赖的镜像 id
tag	varchar	标签
create_date	datetime	创建时间

3.4 用户管理模块设计

运维平台面向的用户为项目管理员、研发人员、测试人员、运维人员、系统管理

万方数据

火龙果·整理
uml.org.cn

员，其中研发、测试、运维人员和项目管理员的角色又会根据项目进行分组。超级管理员具有运维平台所有功能权限，是平台最初始的用户，工程启动时会通过 django 的创建超级用户的脚本生成。其他用户均由运维平台注册页面进行注册和具有创建用户权限的管理员用户创建。为了确保运维平台的安全性，所有注册用用户（包括管理员创建的用户）默认是否激活字段设置为 false，表明用户还未激活，该用户暂时无法访问运维平台和进行任何操作。

整个用户管理模块利用了 Django 自带的后端用户认证框架，自定义修改了用户角色、权限和用户组部分，整体接口设计图 3.11 所示：

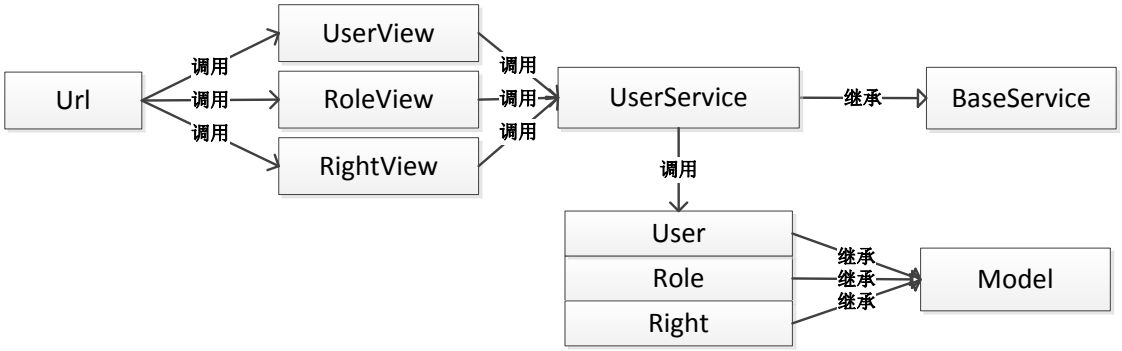


图 3.11 用户管理接口设计

Url 根据用户请求分别调用业务层 UserView、RoleView 和 RightView。其中 UserView 中包含用户的基本功能如登录、注册、注销等；RoleView 中包含用户角色管理功能如角色列表、角色添加、角色的修改和删除功能；RightView 则包含用户的权限管理如权限列表、权限添加、权限修改和删除功能。最终它们都会调用 UserService 层方法。UserService 是用户管理的底层 Service 接口，提供对 User、Role 和 Right 对象的管理，这些对象都继承自 Django 的 Moldel 类。Model 实现了对底层数据库的操作，基本是增删改查的操作。

3.5 运维管理模块设计

运维模块是对主机以及主机上的基础应用资源进行管理的功能模块，包括主机、镜像和容器的管理功能。

3.5.1 主机管理功能设计

原有主机管理方式是通过终端软件或者直接在服务器上连接显示器和键盘进行操作，运维管理平台通过 Web 界面提供常用的主机管理功能操作。用户在界面上执行某一操作时，后端业务层至主机资源层的架构设计如图 3.12 所示：

万方数据

火龙果·整理
uml.org.cn

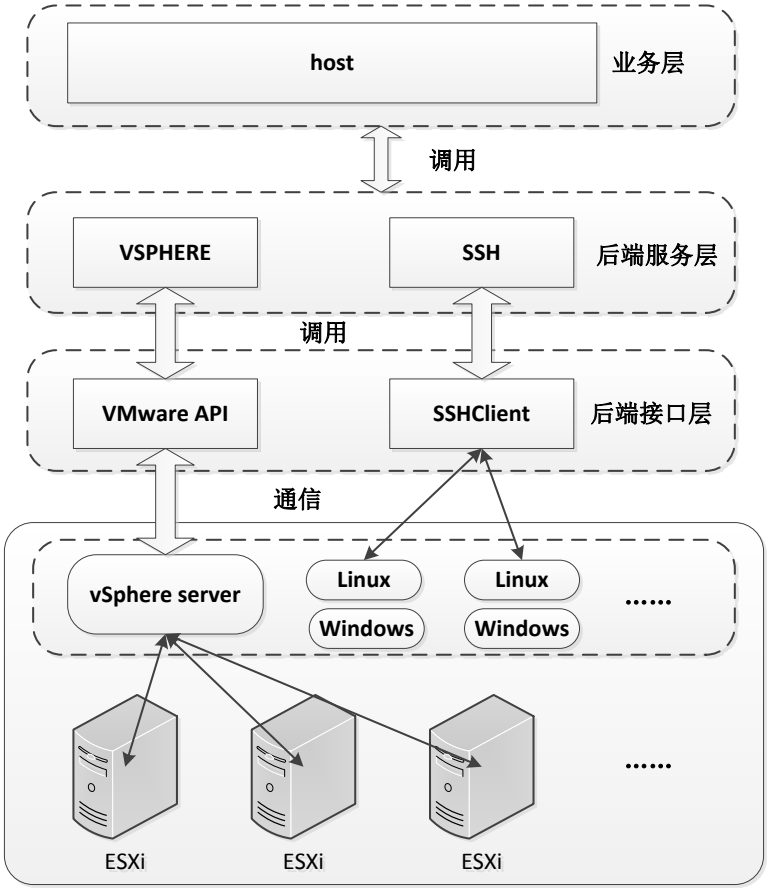


图 3.12 主机管理架构设计

业务层 host 提供主机管理的各种方法，包括主机列表、主机控制、绑定用户、设置权限和远程批量执行命令方法，根据业务内容的不同本文封装了两个重要后端服务类：VSPHERE 类和 SSH 类。

1）VSPHERE 是封装了 VMware API 的类，该类主要提供主机列表、主机控制和设置权限的功能。该类初始化时，会根据配置信息连接到虚拟机管理服务器（vSphere Server）。由于 vSphere Server 提供管理所有 ESXi 主机以及其上所有虚拟主机的功能，连接成功时创建一个管理实例，通过这个管理实例可以获取主机信息、配置主机权限和控制主机状态。设计虚拟主机管理接口如图 3.13 所示：

万方数据

火龙果·整理
uml.org.cn

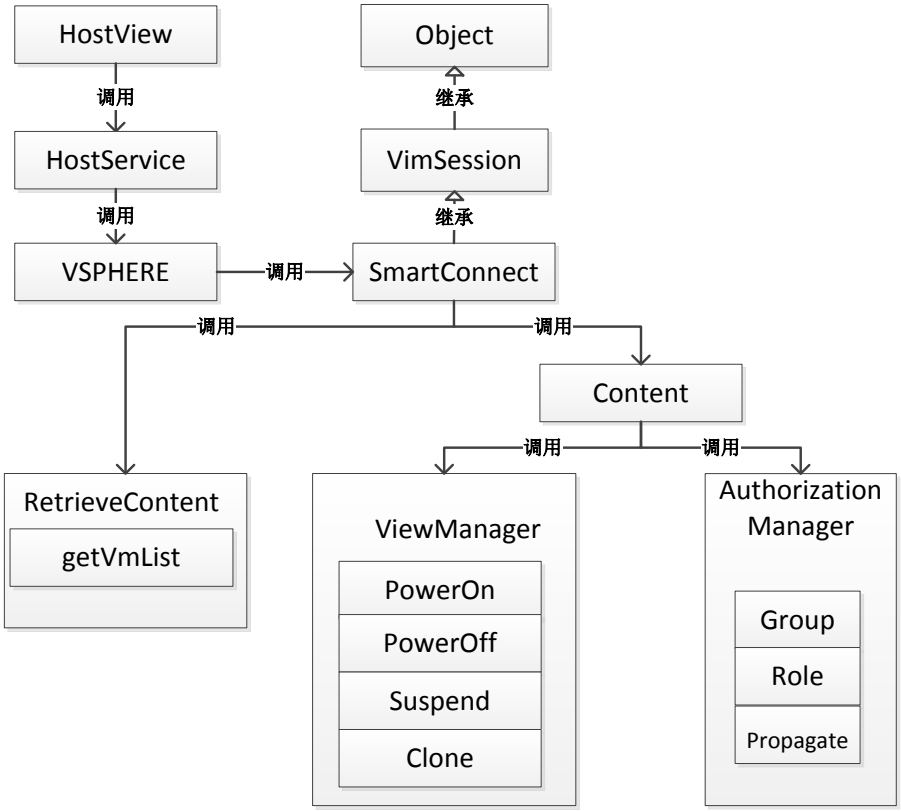


图 3.13 虚拟主机管理接口

HostView 是处理主机管理功能的业务类，它调用 HostService 完成请求并返回视图给用户。HostService 类主要实现了对 VSPHERE 类的操作，VSPHERE 是笔者封装的一个 vSphere server 的功能类，它通过调用 SmartConnect 获得 vSphere Server 对象实例。通过该实例的方法 RetrieveContent 可以获得实例的所有管理对象信息，再经过数据筛选展示在前端页面上。该实例调用 Content 后，可以得到两个管理对象即 ViewManager 和 AuthorizationManager。其中 ViewManager 可以对主机进行克隆和状态控制，AuthorizationManager 用来设置虚拟机对象的用户、角色、权限以及权限的作用域。

2）SSH 是封装了 paramiko 的类，paramiko 是一个基于 ssh 协议的工具类，该类提供 Python 环境下通过 ssh 协议连接主机的功能。SSH 类初始化时，会根据主机及密码信息建立与远程主机的 ssh 连接。在批量执行命令时，需要利用多线程技术与多个远程主机建立连接并执行命令。Linux 主机管理接口设计如图 3.14 所示：

万方数据

火龙果·整理
uml.org.cn

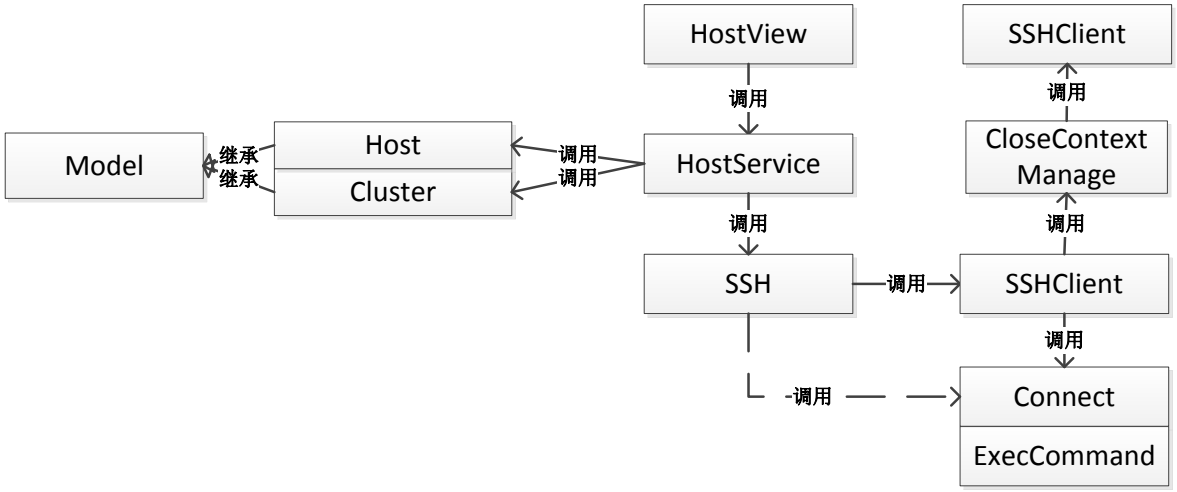


图 3.14 Linux 主机管理接口

SSH 通过调用 SSHClient 的 Connect 方法建立远程主机的连接，连接建立后通过调用 ExecCommand 方法在远程主机上执行命令并获取返回信息。

由于 vCenter Server 中所有虚拟主机的权限数据信息和用户信息都记录在服务器端，给虚拟主机添加用户的使用权限时，权限信息不需要保存在本地数据库中，每次需要修改和添加时直接对 vCenter Server 进行操作。在为 Linux 主机创建用户账号和密码时，由于没有专门的服务器管理和记录所有 Linux 主机以及主机对应的账号信息，所有这些账号信息通过 Host 和 Cluster 对象保存在数据库中。

3.5.2 镜像管理功能设计

镜像管理功能主要包括镜像的列表显示、镜像的创建、删除镜像和查看镜像详情。其中镜像创建功能提供两种方式，一是通过拉取（pull）方式从镜像仓库获取已经配置完成的镜像，二是用户自定义 Dockerfile 文件，用户通过在客户端运行 docker 命令进行编译生成镜像。镜像管理模块将 django 框架与 Docker-py API 结合，该部分的功能接口如图 3.15 所示：

万方数据

火龙果·整理
uml.org.cn

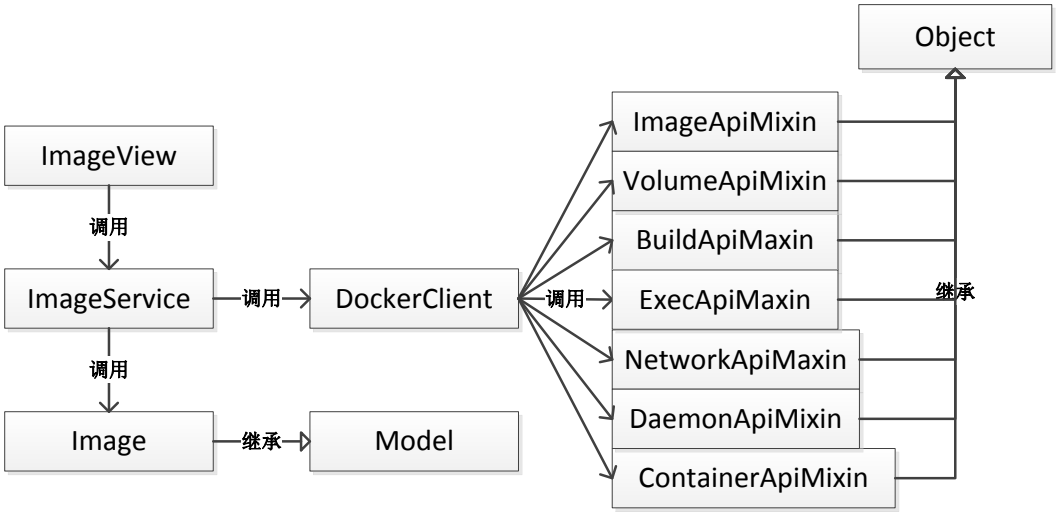


图 3.15 镜像管理功接口

ImageView 是处理镜像管理功能的业务类，它调用 ImageService 完成请求并返回视图给用户。ImageService 封装了 DockerClient 的各项功能，包括对 Docker 服务的各种组件的操作，其中 ImageApiMixin 是处理镜像功能的 API 类，ContainerApiMixin 是处理容器功能的 API 类。下表列出了 DockerAPI 的一些常见功能说明：

表 3.7 Docker API 功能说明

API	功能
build	镜像构建
containers	获取 docker 服务器容器列表，默认是返回正在运行的容器列表
create_container	创建容器
create_network	创建 docker 的虚拟网卡
images	获得 docker 服务器镜像列表
info	获取 docker 服务器系统配置
inspect_container	获取容器的详细信息
inspect_image	获取镜像的详细信息
pull	从远端镜像仓库获取镜像到本地 Docker 服务器
remove_container	删除容器
remove_image	删除镜像
restart	重启容器
start	启动容器
stop	停止容器

万方数据

火龙果·整理
uml.org.cn

3.5.3 容器管理功能设计

容器管理功能包括容器列表、创建、删除、查看容器详情等功能。容器管理过程的通信流程同镜像管理功能通信流程，其接口设计与图 3.14 相似，不同的是，管理对象换成了 container，在 DcokerClient 处主要调用 ContainerApiMixin，这里不再赘述。Docker 容器管理的 API 功能说明见表 3.7。

3.6 应用管理模块设计

应用管理模块设计了具体使用镜像和容器资源的使用场景，在 3.2.1 小节平台运维管理方案中，笔者阐述了如何通过 Docker 容器技术解决应用环境管理低效的问题。这里根据项目实际需要，管理的应用有 Svn、持续集成等，针对不同的应用需要设计不同的业务操作，通用的业务操作有：添加应用、应用状态查看、列表信息查看、修改应用配置文件等。下面介绍代码管理功能设计和 Web 应用管理功能设计。

3.6.1 代码管理功能设计

代码管理工具目前最受欢迎的有 Svn、Git 等。本平台采用 Svn 作为运维平台的代码管理工具，它采用集中式的方式管理代码，代码集中在远程服务器上，用户使用时首先检出代码到本地进行编辑，然后提交。代码管理功能包括创建工程代码仓库，配置工程项目组以及项目组权限信息[42]。本功能将 Svn 封装成镜像，运行在容器中，项目代码通过 Docker volume 指令挂载在网络存储服务器的目录上，项目库认证过程用 Apache 的 http 服务器完成。其中认证方式采取数据库认证，选用数据库中用户表的用户名和密码字段以及项目表的项目组字段作为用户校验。由于镜像的可复用性和容器的隔离性，设计镜像、容器和项目 svn 代码库的关系如图 3.16 所示：

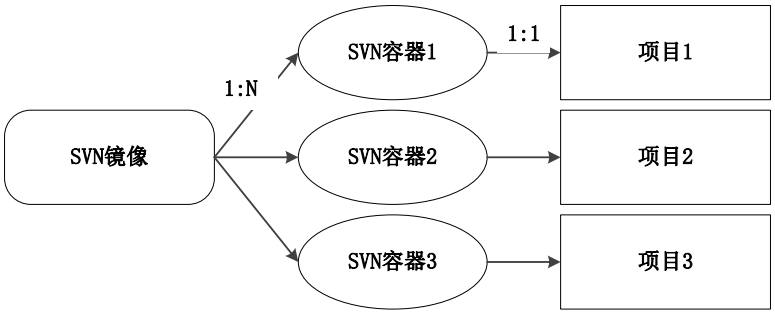


图 3.16 镜像、容器和项目 svn 代码库关系图

运维人员首先编写 svn 应用的对应的 dockerfile，将其中的关键参数通过 ENV 环境变量的方式进行设置，做到通过一次构建 svn 的 Docker 镜像多次使用。当需要创建新的项目时，运行容器时通过传入参数创建并运行特定项目的容器。Svn 容器运行时示意图如图 3.17 所示：

万方数据

火龙果·整理
uml.org.cn

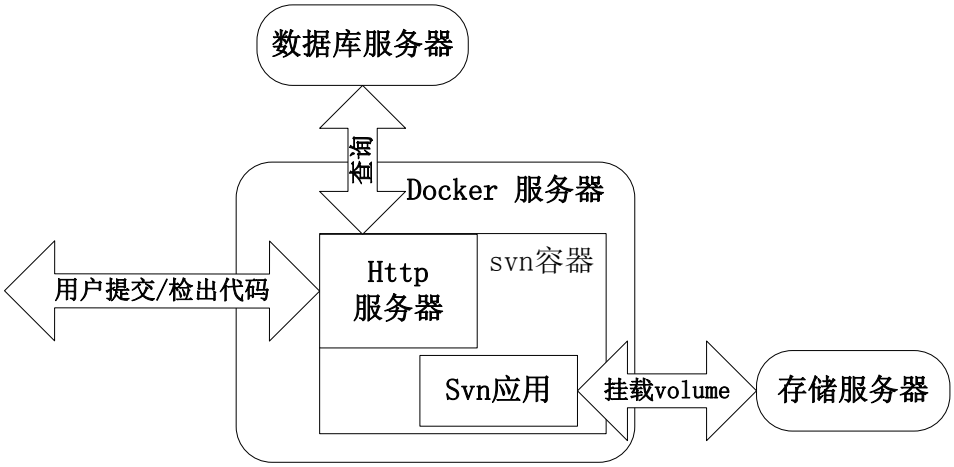


图 3.17 svn 容器运行时示意图

用户首先发出提交或者检出代码的请求，运行在 Docker 服务器上的 svn 容器的网卡接收到请求后，传到内部 http 服务器上，http 服务器根据请求的用户名、密码和用户组去数据库服务器中的数据库核查该用户的合法性，通过验证后 svn 应用处理请求：读取或写入通过 volume 挂载的存储服务器中的代码。最后数据或响应通过 http 服务器返回。

3.6.2 持续集成管理功能的设计

Jenkins 是一个开源持续集平台。运维平台将 jenkins 的部分功能引入，提供 Web 项目的列表显示、持续集成的控制和日志显示，同时将 jenkins 运行在 Docker 容器中便于管理。设计 Web 项目部署运行流程如图 3.18 所示：

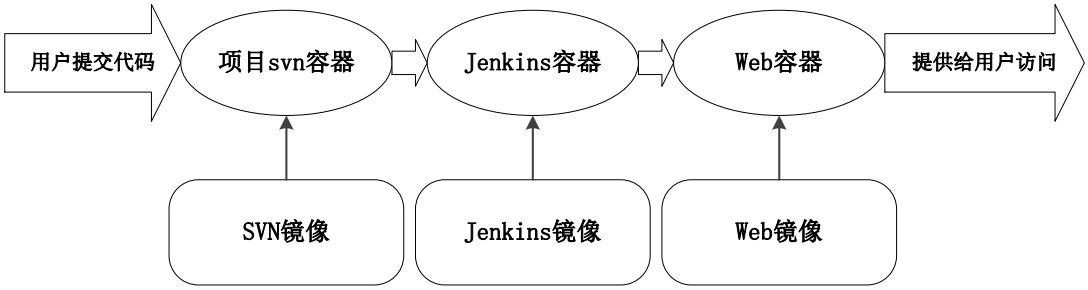


图 3.18 Web 项目部署运行流程

用户提交代码到项目的 svn 容器，在通过自动或手动方式触发 jenkins 容器进行项目构建，构建完成发布到对应 Web 容器，供用户访问。

Web 项目部署管理功能通过 jenkins API 进行访问，获取项目列表和项目状态信息，通过发送请求执行构建和删除功能。该部分主要实现了 JenkinsServer 类，持续集成服务接口设计如图 3.19 所示：

万方数据

火龙果·整理
uml.org.cn

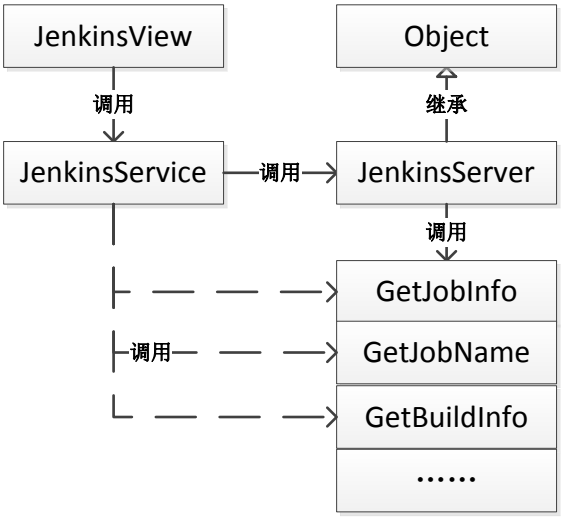


图 3.19 持续集成服务接口

JenkinsService 通过调用 JenkinsServer 类实现持续集成的管理功能。JenkinsServer 类在初始化根据配置信息连接 jenkins 服务器，连接成功后返回服务器实例。利用该实例可以获取任务的信息、任务的名称和任务的构建信息等。JenkinsService 将这些信息进行处理实现 Web 项目列表功能。于此同时服务器实例可以针对任务作出构建、取消构建和删除构建的操作，JenkinsService 将这些功能封装实现 Web 项目构建、删除 Web 项目构建等操作。

3.7 本章小结

本章主要对服务器运维平台进行整体和细节方面的分析和设计。首先分析了基于 Docker 的服务器运维平台的总体需求，包括业务需求分析和功能需求分析。然后从用户管理模块、运维模块、应用管理模块三个方面说明了系统的具体设计。

万方数据

火龙果·整理
uml.org.cn

4 运维平台的实现

在第三章中，对本课题的需求进行了详细的分析，并在此基础上完成了系统了总体架构设计和各功能模块的设计，为管理平台的实现奠定了基础。本章节首先说明了开发环境和框架配置以及他们的搭建过程，然后详细说明了管理平台各个功能模块的具体实现。

4.1 开发环境与框架配置

运维平台的开发环境如表 4.1 所示：

表 4.1 开发环境

工具/环境	软件名称
集成开发环境	PyCharm 5.0
虚拟平台	ESXI 5.5
虚拟平台服务端软件	VSphere 5.5
虚拟平台客户端管理软件	VMware vSphere Client 5.5
Docker 容器	Docker 1.9
Docker 容器 python API	Docker-py 1.7.2
服务端操作系统	CentOS 6.5
代码编译器	Python 2.7.11
数据库	Mysql 5.5
客户端软件	Chrome、Firefox、IE 等浏览器
Http 服务器	Nginx 1.7
Web 框架	Django 1.8.7
vSphere API	Pyvmomi 6.0.0

4.2 环境搭建

由于基于 Docker 服务器管理平台涉及平台研发框架、线上环境以及相关配套环境，这里简要讲解以这三个方面环境的搭建。

4.2.1 开发框架搭建

开发环境使用 Ubuntu14.04 x64 操作系统，集成开发工具选择 pyCharm5.0，编译器使用 python2.7.11，这里略过系统、集成开发工具、编译运行环境的安装和配置。

万方数据

火龙果·整理
uml.org.cn

服务器管理平台采用 django 框架进行开发，前端使用 bootstrap，交互使用 jQuery。开发框架安装步骤（以下步骤均在 Linux 命令行下执行）：

1) Django 框架安装

执行 pip install Django==1.8.7 即可安装 1.8.7 版本的 django 框架。

2) 安装 MySQL-python

MySQL-python 是 python 访问 MySQL 数据库的第三方模块库。执行 pip install MySQL-python 即可安装该模块库。

3) 创建工程和应用

在目录中创建 django 工程，执行 django-admin.py startproject dops，之后执行 python manage.py startapp web 创建主体应用 web。

4) 前端框架安装

下载 bootstrap 和 jQuery 文件移动到工程下的 statics 目录。静态文件需要在 html 文件中引用，由于 django 采用模板继承方式，工程所有页面继承 index.html。

至此平台的开发框架搭建完成。

4.2.2 服务器环境搭建

服务器管理平台基于 Docker 引擎和虚拟化管理平台。

1) 虚拟化环境搭建

平台环境主要搭建在若干高性能服务器上。安装虚拟化软件 ESXi。虚拟机服务器建立在 ESXi 平台上，由 ESXi 管理 VMware vSphere Client 来创建。通过 VMware vSphere Client 创建了如下几个虚拟服务器，服务器安装的操作系统均为 CentOS，虚拟服务器表如表 4.2 所示：

表 4.2 主要服务器列表

服务器名称	IP	提供服务
vCenter-serv	192.168.**.150	虚拟机管理
App-serv	192.168.**.28	Django 应用、Nginx、mysql
Docker-serv	192.168.**.168	Docker，私有仓库

这里平台需要使用的主要服务器就是以上三台服务器，其余虚拟服务器和开发人员使用的虚拟机均由以上三台机器纳入到平台的管理系统中。

2) Docker 环境安装

在 Docker-serv 上安装执行 yum install Docker.io，执行 docker 则 Docker 服务启动。

4.2.3 生产环境搭建

平台开发和调试环境均在笔者本地进行，后期上线到正式生产环境时需要搭建生产环境，由 Nginx+uwsgi 方式实现 django 应用的访问。具体搭建过程如下：

万方数据

火龙果·整理
uml.org.cn

1) Nginx 服务器的安装

在 App-serv 下载 Nginx 安装包，解压缩进入文件目录，执行 make && make install 完成 Nginx 安装。

2) 安装 WSGI 服务器

在 App-serv 上下载 uwsgi 安装包，解压缩进入文件目录执行 make，然后将编译完成的 uwsgi 文件拷入到系统/usr/bin 目录下完成安装。

3) 配置 Nginx

修改/usr/local/Nginx/conf/Nginx/conf，添加一下 server 配置[43]：

```
    server {
        listen      80;
        server_name  appserv;
        location / {
              uwsgi_pass appserv:8080;
              include uwsgi_params;
        }}
```

4) 配置 uwsgi

```
[uwsgi]
socket = 127.0.0.1:8080
chdir = /home/root/project/dops
wsgi-file = dops/wsgi.py
processes = 4
logdate = true
memory-report = true
enable-threads = true
daemonize = /usr/local/Nginx/logs/django.log
```

5) 启动

在 App-serv 上分别执行/usr/bin/uwsgi –ini /etc /uwsgi.ini 和 nginx，启动完成。访问本地 App-serv 的 80 端口就可以看到欢迎页面。

4.3 用户管理模块实现

4.3.1 用户基本功能实现

用户基本功能根据用户角色的不同，包括用户注册、登录、个人信息编辑、密码找回和用户审核等功能。

所有有关用户请求的内容都要进入 user.py 处理，根据不同请求调用 userService.py 中对应处理代码，最后通过 ORM 模型跟数据库进行数据通信。概要步骤如下：

用户访问运维平台进行注册、登录、修改个人信息、密码找回等操作，url 路由请求分发会调用 user.py 中对应的 register、login、edit、forgetpasswd 等方法，这些方法

万方数据

火龙果·整理
uml.org.cn

首先获取 request 中的用户数据，然后调用 UsersServer.py 的 register、login、edit、forgetpasswd 等方法进行业务逻辑处理，最后通过 Model 封装的 User 的 save 方法保存到 MySQL 数据库当中。

4.3.2 角色权限管理功能实现

管理功能的实现由权限添加页面、角色添加页面，权限和角色列表、权限认证模块几个部分组成，其中权限验证模块是全局使用，作用到 view 层方法之上。权限验证流程如图 4.1 所示：

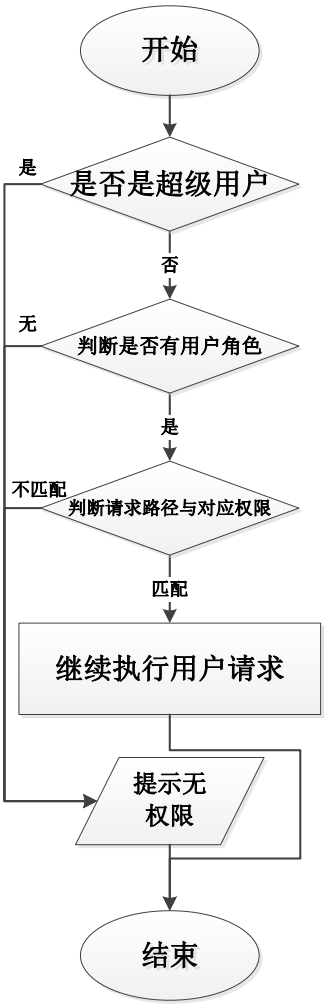


图 4.1 权限验证流程

用户通过浏览器访问某一地址时，url 路由匹配到具体方法，若方法上有权限验证，则权限验证模块根据当前访问用户在数据库中查找出该用户的所有权限，判断当前访问路径是否在用户拥有的权限列表中，若存在则权限验证通过，并执行相关业务操作，返回结果页面；若权限验证没有通过则返回权限拒绝页面。

权限认证模块的具体实现代码如下：

万方数据

火龙果·整理
uml.org.cn


```
def permission_verify():
    def decorator(view_func):
        def _wrapped_view(request, *args, **kwargs):
            current_user = User.objects.get(username=request.user)
            if not current_user.is_superuser:  # 判断用户如果是超级管理员则具有
所有权限
                if not current_user.role:  # 如果用户无角色，直接返回无权限
                    return HttpResponseRedirect('/permissiondeny')
                # 获得该角色的所有权限
                match_url = []
                for x in role_permission_list:
                    if request.path == x.url or request.path.rstrip('/') ==
x.url:  # 精确匹配，判断 request.path 是否与 permission 表中的某一条相符
                        match_url.append(x.url)
                    elif request.path.startswith(x.url):  # 判断 request.path
是否以 permission 表中的某一条 url 开头
                        match_url.append(x.url)
                if len(match_url) == 0:
                    return HttpResponseRedirect('/permissiondeny')
            return view_func(request, *args, **kwargs)
        return _wrapped_view
    return decorator
```

在应用中某个需要判断权限的方法之上通过 Python 装饰器的方式使用，例子如下：

```
@login_required
@permission_verify() # 添加权限验证模块
def list(request):
        ......
```

4.4 运维管理模块实现

4.4.1 主机管理功能实现

主机管理功能主要包括主机列表页面、添加主机页面、添加集群页面、绑定主机页面、批量执行页面、创建主机页面。下面主要说明绑定主机功能和主机控制功能页面。

在主机管理中，经常会给 Linux 主机分配项目人员的账号，项目人员可以登录主机执行一些不常见的操作(平台未涉及到的功能)。此时需要远程给主机添加制定用户，并配置登陆密码。主机绑定用户的实现如图 4.2 所示：

万方数据

火龙果·整理
uml.org.cn

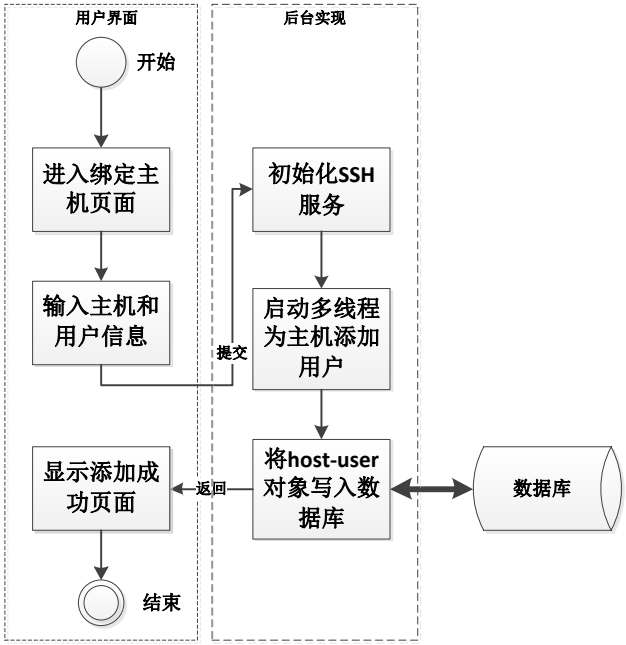


图 4.2 主机绑定用户实现示意图

批量命令执行，是在界面上输入多个主机的地址，输入要执行的命令，点击提交，后台接收到任务后会通过多线程方式执行命令，这里依然是通过 ssh 接口完成主机登录和命令执行的。批量命令执行的实现如图 4.3 所示：

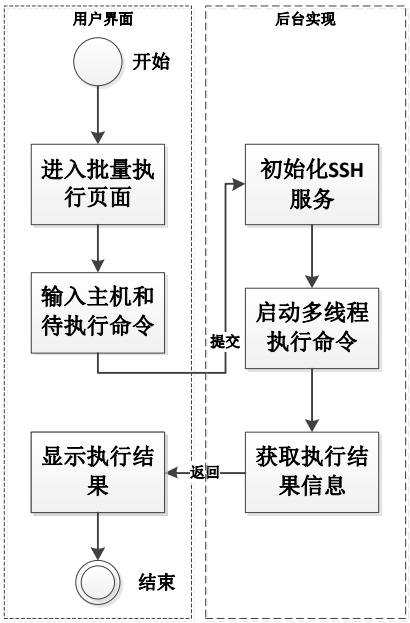


图 4.3 批量命令执行的实现示意图

以上两个部分都用到了 parakimo 库，该库是 SSH 通信的 python 实现，在平台里封装成了 ssh 类，提供基础功能如 remote_execute、add_user 功能，该 SSH 类主要实现如下：

万方数据

火龙果·整理
uml.org.cn


```
class Ssh():
    def __init__(self):
        self.stdin = []
        self.stdout = []
        self.stderr = []
        self.username = dops.settings.SSH_USERNAME
        self.password = dops.settings.SSH_PASSWORD

    def remote_execute(self, host, username='', password='', cmd=''):
        ......

    def add_user(self, host, add_username=''):
        ......

# 批量执行，多线程方式调用 remote_execute 方法
def batch_execute(hosts, operator_name='', cmd=''):
......
```

其中批量执行命令会使用多线程方式，每一个线程连接一台远程主机并调用 remote_execute。

主机管理功能还包括虚拟主机的管理，是将 VMware API 进行封装，封装成 VSPHERE 类，提供获取虚拟主机列表状态信息，虚拟机的开机、关机、挂起等操作。

其中显示主机列表和控制主机状态实现如图 4.4 所示：

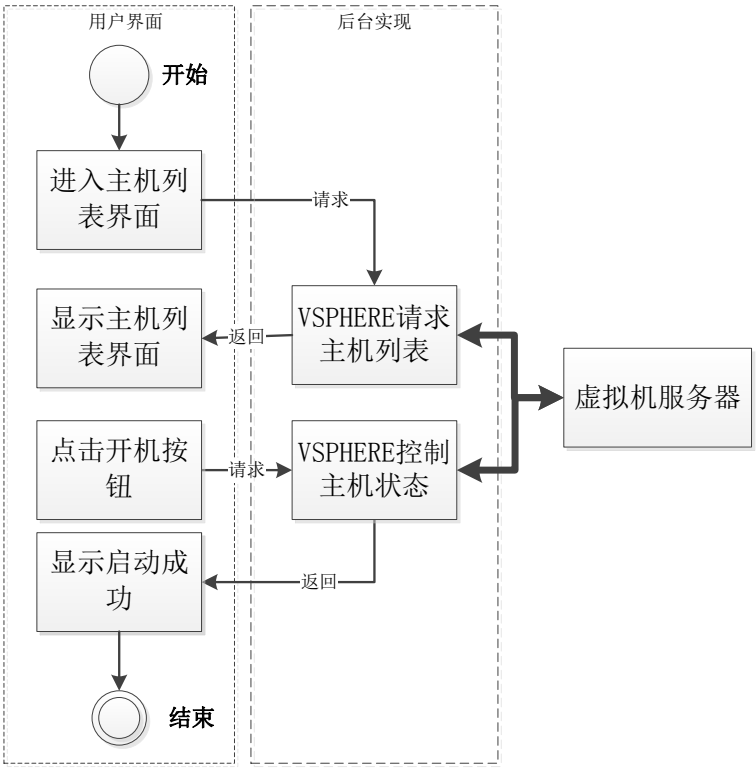


图 4.4 主机管理实现示意图

万方数据

火龙果·整理
uml.org.cn

主要实现代码如下：

```
vm_infos = []

class VSPHERE(): # vSphere 功能类
    def __init__(self):
        context = ssl.SSLContext(ssl.PROTOCOL_TLSv1)
        context.verify_mode = ssl.CERT_NONE
        si = SmartConnect(host=settings.VSPHERE_HOST,
                          user=settings.VSPHERE_USERNAME,
                          pwd=settings.VSPHERE_PASSWORD,
                          port=int(443),
                          sslContext=context)
        atexit.register(Disconnect, si)
        self.client = si
    def vmlist(self):
        content = self.client.RetrieveContent()

    def poweron(self, vmnames):
        content = self.client.content

        objView=content.viewManager.CreateContainerView(content.rootFold
        er, [vim.VirtualMachine], True)

     def poweroff(self, vmnames):
     def suspend(self, vmnames):
```

该类初始化时，会配置一个基于 SSL 安全会话环境，然后读取配置文件中虚拟及服务器的主机信息如地址、用户名和密码与主机建立连接，最后调用 vmlist()获取主机列表信息、调用 poweron 方法开机，其他功能类似。

4.4.2 镜像管理功能实现

镜像管理功能中最重要的部分是创建镜像。在 Docker 容器技术中，创建镜像有两种方式：一种是通过 pull 方式拉取镜像仓库的已经创建好的镜像到本地；另一种方式是编写 Dockerfile 文件，在客户端上通过 build 命令进行创建，通过这种方式创建镜像的基础镜像还是通过 pull 方式从仓库中获取，但镜像创建的后续过程则根据编写的 Dockerfile 来进行。通过这种方式可以将镜像最小化成一个文件形式，大大提高了镜像创建的灵活性和存储成本。一个通用 Dockerfile 文件说明如下：

```
# image name
# 版本
# VERSION 0.1
# 基础镜像
FROM      centos:6
# 作者
MAINTAINER author_name
# 更新系统软件源
```


万方数据

火龙果·整理
uml.org.cn


```
RUN yum -y update
# 安装应用
RUN yum -y install appname
# 删除安装完成的安装包
RUN yum clean all
# 设置工作目录
RUN mkdir -p /data/work
# 设置数据卷目录
VOLUME /data/work
# 设置环境变量
ENV PATH var
# 运行命令
RUN command
# 添加文件到指定路径
ADD configuration.conf /path/
# 暴露访问端口
EXPOSE 80
CMD service servicename start
```

镜像管理功能中创建镜像实现如图 4.5 所示：

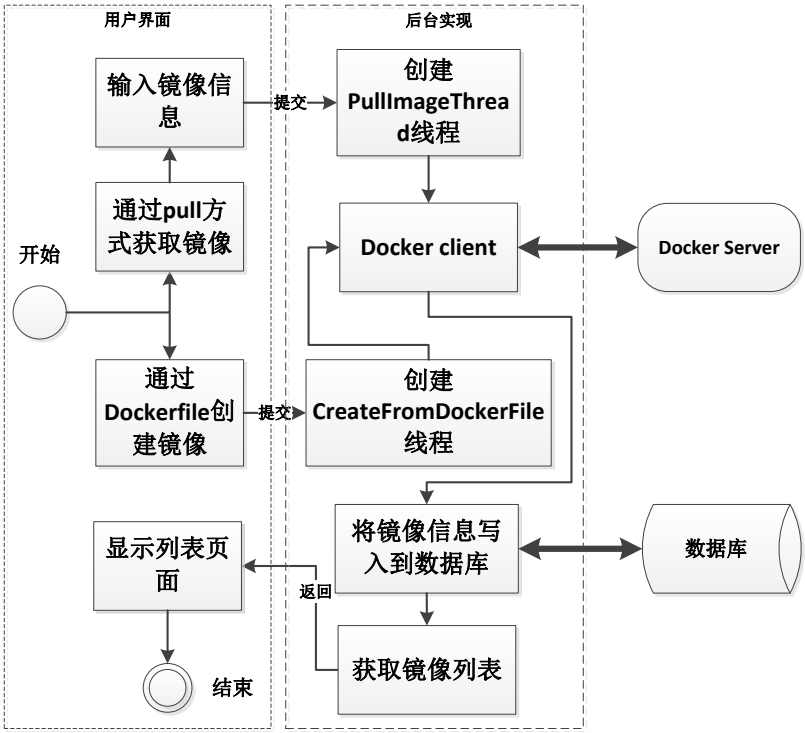


图 4.5 创建镜像的实现示意图

镜像管理基于 Docker-py 接口进行封装，imageService.py 中实现了镜像管理有关的业务。具体类和方法如下：

1) get_images()：从数据库中获取所有镜像；
2) get_image(image_id)：从 Docker daemon 中获取指定 id 镜像的详细信息；

万方数据

火龙果·整理
uml.org.cn

3) CreateFromDockerfile(Thread.thread)：通过 Dockerfile 创建镜像；
4) PullImageThread()：获取远程 Docker 镜像仓库镜像；
5) delete(image_id)：从 Docker 本地镜像库和数据库删除该镜像。

4.4.3 容器管理功能实现

容器管理主要包括创建容器和获取容器信息，其中创建容器的实现如图 4.6 所示：

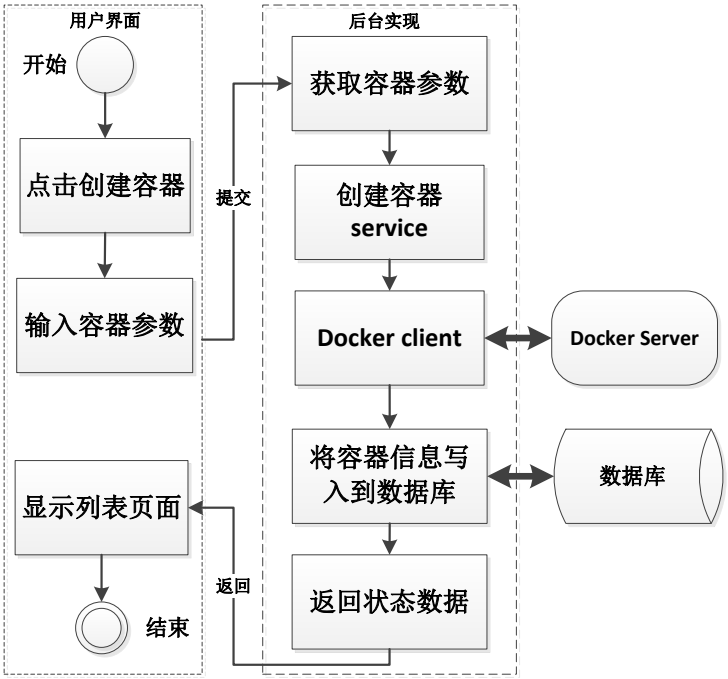


图 4.6 创建容器的实现示意图

用户进入容器管理界面之后，点击创建容器，出现容器参数表单，需要填写的内容包括容器所使用镜像，容器启动时命令和环境变量等等。填写完成后提交该表单，后台接收到请求后，获取表单信息，提交给 containerService.py 中的 create 方法，提交给 Docker-serv 处理，完成后返回信息到前台页面。

容器管理基于 Docker-py 接口进行封装，containerService.py 中实现了容器管理的相关业务。具体方法如下：

1) get_containers()：从 Docker daemon 中获取容器列表；
2) get_container(container_id)：获取指定 id 容器的详细信息；
3) create()：根据 post 请求参数创建容器；
4) start(container_id)：启动容器；
5) stop(container_id)：停止容器；
6) log(container_id)：获取容器运行日志；
7) delete(container_id)：删除容器。

万方数据

火龙果·整理
uml.org.cn

4.5 应用管理模块实现

4.5.1 代码管理功能实现

创建基于 Docker 的 svn 镜像的实现如图 4.7 所示：

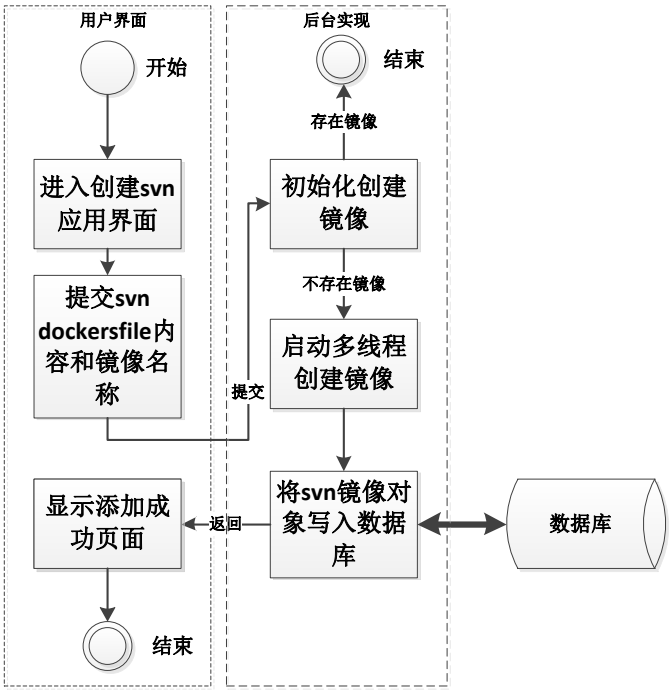


图 4.7 创建基于 Docker 的 svn 镜像的实现示意图

在创建代码仓库的页面，添加项目名称、描述、添加项目组以及相关组的权限信息，提交后，平台首先检查数据库中是否存在该项目，存在则提示用户相同项目代码库已经创建。若不存在则进入下一步，初始化。该过程先创建新的镜像，该镜像基于已经创建过的镜像生成，已经包含了 svn 应用，创建过程中项目名称作为变量赋值给该镜像名和 svn 创建的代码仓库名已经路径信息；若创建过程失败则向用户提示相关错误信息，流程结束。最后将信息写入到数据库中，流程结束。项目组信息可以通过 http 的 mod_mysql 进行认证，需要在/etc/httpd/conf/httpd.conf 添加以下内容。

```
<Location /svn>
      DAV svn *开启 DAV 模块支持!
      SVNParentPath /data/svn/ *项目的父目录
      AuthzSVNAccessFile /data/svn/authz *SVN 访问认证文件
      AuthName "SUBVERSION" *认证名称
      AuthType Basic *基本认证
      AuthMYSQLEnable on *开启 Mysql 认证
      AuthMySQLHost 192.168.**.28 mysql 主机所在 ip（默认不写指本地）
      AuthMYSQLUser root *数据库访问用户名
```


万方数据

火龙果·整理
uml.org.cn


```
    AuthMySQLPassword ***** *数据库访问的密码
    AuthMYSQLDB dops *存放认证信息的数据库名称
    AuthMYSQLUserTable users *存放认证信息的表名称
    AuthMYSQLNameField user_name *存放认证用户名的字段名称
    AuthMYSQLPasswordField user_passwd *存放认证密码的字段名称
    AuthMySQLGroupTable group
    AuthMySQLGroupField name
    AuthMySQLPwEncryption md5 数据库密码字段采用 MD5 加密（这里需要将
django 的认证模块采用 MD5 加密）
    Require valid-user *表示只有认证的用户才能登陆
    Require group GROUP_NAME
</Location>
```

Svn 镜像的 dockerfile 如下所示：

```
......
# 安装所有软件，包括 svn、http 服务器、认证模块
RUN yum -y install subversion httpd mod_auth_mysql mod_dav_svn
&& yum clean all
# 创建 svn 工作目录
RUN mkdir -p /data/svn

# 容器内需要挂载的目录，指定 svn 代码目录
VOLUME /data/svn

# 根据容器启动出传入的参数创建
ENV SVN_REPONAME repos
RUN svnadmin create /data/svn/$SVN_REPONAME

# 改变 svn 目录的所有者
RUN chown apache:apache /data/svn -R

# 删除原有的 http 服务器的配置文件
RUN rm -v /etc/httpd/conf/httpd.conf

# 添加自定义的 http 服务器配置文件
ADD httpd.conf /etc/httpd/conf/

# 开启 Apache http 服务器的默认端口
EXPOSE 80
CMD service httpd start
```

4.5.2 持续集成管理功能的实现

项目持续集成功能主要是依赖jenkins提供的自动化发布功能，通过python-jenkins API与jenkins服务器进行通信。主要实现了JenkinsServer类，提供list()、build()、install()和delete()方法供上层调用。

用户首先访问/app/jenkins/jobs，url.py路由分发将请求转发到app_jenkins.py的对

万方数据

火龙果 · 整理
uml.org.cn

应方法上，最后调用 JenkinsServer 类中的服务完成请求，最后 app_jenkins.py 将 jenkins 服务中的信息返回到前端页面。该类主要实现了 job_List()方法获取所有项目的构建配置，实现了 install_service()安装 jenkins 的服务，实现了 delete_job()方法删除项目的构建配置。

4.6 本章小结

本章主要详细介绍了管理平台各个部分的实现，首先对开发环境及框架配置作了简要介绍，然后介绍了用户管理模块的实现，接着描述了基础模块实现，包括主机管理、镜像管理、容器管理功能的实现，然后介绍了应用模块实现。

万方数据

火龙果·整理
uml.org.cn

5 平台的测试

上一章详细介绍了平台主要功能的实现过程，本章将对基于 Docker 的服务器运维平台进行测试，并对测试结果进行分析。

5.1 测试方案

本平台的测试环境如表 5.1 所示：

表 5.1 测试环境

环境	名称
运行平台	Django 1.8.7
部署服务器	Ubuntu 14.04
浏览器	Chrome 39+、Firefox 35+、IE 10+
数据库	MySQL 5.6
客户端调试工具	Firebug、Chrome 开发者工具等
服务端调试工具	PyCharm 5.0

本平台在开发过程中主要采用白盒测试，根据代码逻辑进行功能调试，调试工具是集成开发环境 Pycharm。

5.1.1 功能测试方案

发布到应用服务器后采用黑盒测试，本系统的测试工具主要有 Chrome 浏览器、Firefox 以及它们的调试插件。功能测试主要依靠操作人员手工操作各个功能页面，检查返回页面成功与否。在一些与后端服务器交互的业务操作中，还需要通过相关传统的命令行方式去检验功能是否完成。

1) 用户管理功能测试

按照用户管理功能的需求，测试涉及用户（不同角色）可以进行的所有操作，包括用户登录、注册、修改个人信息、找回密码等基本功能，还包括用户角色、权限、用户组管理功能，主要是手工操作并在数据库中确认；

2) 运维管理功能测试

运维管理功能的测试涉及到多个管理对象，需要分别设计不同的测试方案。

主机管理功能的测试除了在页面上点击和显示外，还需要到登录到对应服务器上去查看各业务功能是否实现。例如批量执行命令时，需要登录到远程 Linux 主机上查看命令的历史记录是否包含执目标命令；执行远程主机控制操作时，除检查主机列表

万方数据

火龙果·整理
uml.org.cn

的状态信息外，还需要到 vCenter Server 或直接通过 VMware Workstation 上查看主机状态。

镜像管理需要在页面上测试创建、列表查看、详情查看、删除功能，并在 Docker-server 上使用 docker 相关命令检查镜像存在与否和数据库中查看数据是否正确。

容器管理需要在页面上测试创建、列表查看、详情查看、删除功能，并在 Docker-server 上使用 docker 相关命令检查容器是否存在，针对容器的业务功能还需要进一步检查。例如运行 nginx 的容器，在浏览器中访问，能够得到正确的页面显示。

3) 应用管理功能测试

应用管理功能主要包括代码管理功能测试和 Web 应用部署测试。

主要是对 svn 代码管理功能进行测试，测试功能出手动在页面点击外，还需要查看 Docker-server 上 Svn 管理功能是否正常使用。

5.1.2 业务测试方案

业务测试主要是在实际应用场景下，不同角色用户通力配合完成的一整套流程的测试。这里主要测试项目创建到发布业务流程的测试。该业务的测试流程如图 5.1：

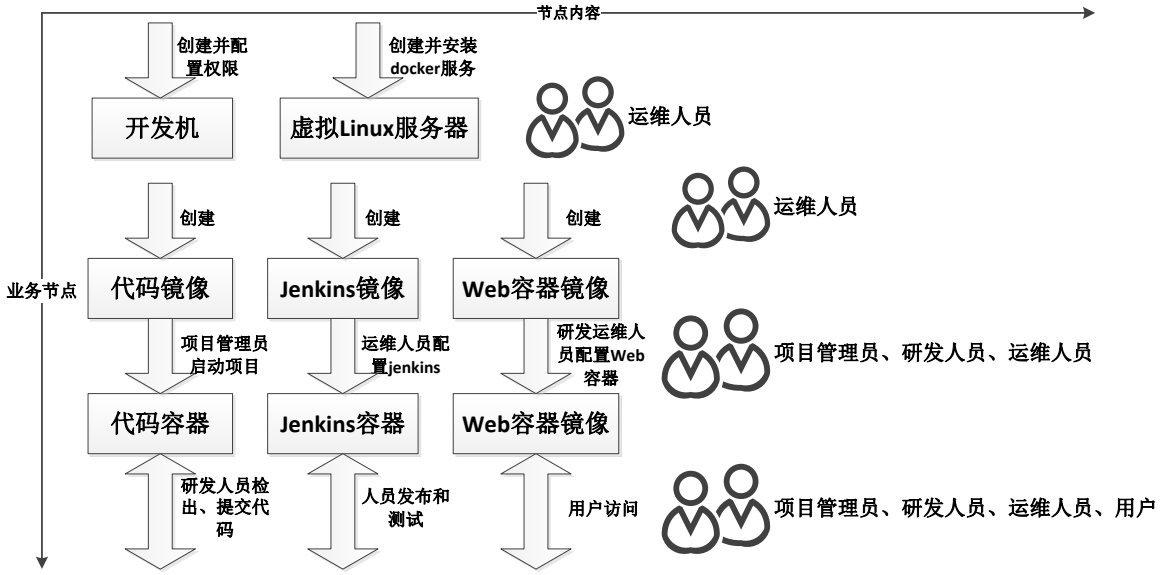


图 5.1 业务测试流程

如图所示，该业务流程从两个维度展开介绍：纵向维度和横向维度。纵向维度是不同时间的业务节点，横向是此时间内业务节点的内容（省略其他服务内容如数据库、代理服务器），最后一列是当前节点可以操作的用户角色。首先运维人员给研发人员创建开发机，创建虚拟 Linux 服务器。然后创建代码，持续集成服务以及 Web 容器的镜像供项目的开展使用；项目启动时，项目管理员创建项目以及项目代码库，运维人

万方数据

火龙果·整理
uml.org.cn

员配置持续集成服务，研发和测试人员可以自定义使用 Web 镜像，创建自己的测试环境；最后进行日常研发活动，检入、检出代码、测试发布以及正式发布。

5.2 功能测试结果分析

5.2.1 用户管理功能测试分析

根据 5.1 小节中描述的测试方案对用户管理功能进行测试，其各项功能测试结果如表 5.2 所示：

表 5.2 用户管理功能测试

序号	功能	输入与动作	预期响应结果	实际响应结果	平均响应时间（ms）	是否通过
01	用户登录	输入用户名密码，点击登录	登录成功则显示用户主界面；登录失败则提示错误信息	登录成功则显示用户主界面；登录失败则提示错误信息	69/30	是
02	用户列表	点击用户基本功能	显示用户列表界面	显示用户列表界面	44	是
03	用户注册	输入用户名、密码、确认密码、邮箱，点击提交	显示登录页面，显示注册成功，等待管理员审核	显示登录页面，显示注册成功，等待管理员审核	100	是
04	用户进入修改个人信息页面	点击修改个人信息	显示修改个人信息页面，邮箱字段已读取，密码为空	显示修改个人信息页面，邮箱字段已读取，密码为空	30	是
05	用户修改个人信息	输入需要修改的字段，点击提交按钮	显示用户主页面，提示操作成功	显示用户主页面，提示操作成功或失败原因	66/40	是
06	进入审核页面	管理员登录，点击审核页面	显示审核页面，列出未审核用户	显示审核页面，列出未审核用户	34	是
07	审核用户	点击审核栏单选框	选择正常，提示用户审核完成	选择正常，提示用户审核完成	10	是
08	密码找回	点击忘记密码，输入邮箱	提交正常，邮箱收到重置密码邮件	提交正常，邮箱收到重置密码邮件	5900	是

万方数据

火龙果·整理
uml.org.cn

09	删除用户	点击用户列表页面的删除按钮	提示用户删除成功	提示用户删除成功	85	是
10	管理员创建用户	管理员进入添加用户页面，输入用户信息	提示创建用户成功	提示创建用户成功	113	是
11	添加角色	进入添加角色页面，输入角色信息，点击提交	提示创建角色成功，显示角色列表页面	提示创建角色成功，显示角色列表页面	379	是
12	添加权限	进入添加权限页面，输入权信息，点击提交	提示创建权限成功，显示权限列表页面	提示创建权限成功，显示权限列表页面	158	是
13	角色列表	点击角色列表页面	显示角色列表	显示角色列表	29	是
14	权限列表	点击权限列表页面	显示权限列表	显示权限列表	35	是
15	删除角色	点击角色列表页面删除按钮	提示删除角色成功	提示删除角色成功	81	是
16	删除权限	点击权限列表页面删除按钮	提示删除权限成功	提示删除权限成功	100	是
17	项目组列表	点击项目组管理	显示项目组列表	显示项目组列表	32	是
18	创建项目组	进入创建项目组页面，输入项目组信息，点击添加	提示创建项目组成功，显示项目组列表页面	提示创建项目组成功，显示项目组列表页面	70	是
19	删除项目组	点击项目组列表页面的删除按钮	提示删除成功按钮，显示项目组列表	提示删除成功按钮，显示项目组列表	96	是
20	为项目组	进入项目组成	提示分配项目组成	提示分配项目组成	101	是

万方数据

火龙果 · 整理
uml.org.cn

	分配人员	员分配页面选择成员，点击提交	员成功	员成功		

通过分析表 5.2 可以得出，进入列表类页面时响应时间在 100ms 以内，这是因为运维平台是单页面应用，在首次访问页面时已经将需要的静态资源文件缓存起来，再次访问该页面的时候，浏览器直接从缓存读取，另外列表中数据是分段请求的，当列表数据量非常大的时候一次性读取所有数据不仅查询数据库比较慢，且网络传输过程也会有所影响，所以响应速度较快。表中所有测试项目的结果说明用户管理部分的基本功能已经完成，成功和失败均有对应的响应。

1) 用户管理界面上半部分是用户操作子菜单，提供可以进行的操作。下半部分是显示栏，显示当前操作的内容，如用户列表界面显示如图 5.2 所示：

图 5.2 用户列表界面

点击进入用户修改信息界面，可以看到如图 5.2 所示的修改用户信息界面用户名和邮件已经默认填写完成，若用户需要修改可自行编辑，下面的原始密码和新密码均为空，这里测试修改密码功能，填写完成密码和新密码后，点击提交，系统提示修改成功，返回用户列表界面。

角色管理界面，选择添加角色，界面如图 5.3 所示：

万方数据

火龙果·整理
uml.org.cn

图 5.3 添加角色

添加角色时下面会提供可选择的用户权限，由于角色和权限是一对多的关系，设计如图所示的控件方便选择权限。

2) 项目组管理功能，添加时下面会提供可选择的用户权限，由于项目组和项目成员是一对多的关系，设计界面与添加角色页面类似，提供双向选择控件。

5.2.2 运维管理功能测试分析

根据 5.1 小节的运维测试方案，测试时首先在页面上操作，查看页面响应是否符合预期，再登录到相关服务器查询业务的具体执行情况，详细测试功能如表 5.3 所示：

表 5.3 运维功能测试

序号	功能	输入与动作	预期响应结果	实际响应结果	平均响应时间（ms）	是否通过
01	镜像列表	点击镜像列表	显示镜像列表	显示镜像列表	35	是
02	通过 pull 方式创建镜像	提交创建镜像的仓库位置，名称和标签	提示创建成功，跳转到镜像列表页面，通过 Docker images 命令查看存在该镜像	提示创建成功，跳转到镜像列表页面，通过 Docker images 命令查看存在该镜像	30	是
03	通过 Dockerfile 创建镜像	撰写 Dockerfile，提交	提示创建成功，跳转到镜像列表页面，通过 Docker images 命令查看	提示创建成功，跳转到镜像列表页面，通过 Docker images 命令查看	36	是

万方数据

火龙果·整理
uml.org.cn

			存在该镜像	存在该镜像		
04	镜像详情	点击镜像名称	弹出镜像详情对话框	弹出镜像详情对话框	86	是
05	删除镜像	点击删除镜像	显示镜像列表页面，通过 Docker images 命令查看该镜像已删除	显示镜像列表页面，通过 Docker images 命令查看该镜像已删除	393	是
06	容器列表	点击容器列表	显示容器列表	显示容器列表	34	是
07	创建容器	提交创建容器的各项参数	提示创建成功显示容器列表	提示创建成功显示容器列表	100	是
08	容器详情	点击容器名称	弹出容器详情对话框	弹出容器详情对话框	20	是
09	容器日志	点击容器日志	弹出容器日志对话框	弹出容器日志对话框	66	是
10	删除容器	点击删除容器	提示容器删除	提示容器删除	90	是
11	容器控制	依次点击运行、停止	依次提示容器运行、容器停止，显示容器列表页面	依次提示容器运行、容器停止，显示容器列表页面	62/66	是
12	主机列表	点击主机列表	显示主机列表	显示主机列表	740	是
13	绑定用户	填写一个或多个主机和一个或多个用户名，提交绑定	提示绑定成功，登录远程主机用该用户的默认密码登录成功	提示绑定成功，登录远程主机用该用户的默认密码登录成功	2230	是
14	批量执行命令	填写主机列表和要执行的命令，提交	提示运行成功，弹出运行结果，登录主机查看命令历史存在该命令	提示运行成功，弹出运行结果，登录主机查看命令历史存在该命令	2940	是

其中创建镜像时间不定，当本地存在镜像缓存时，创建镜像时间短，当本地没有缓存，则需要远程下载，根据镜像大小下载时间不一。显示主机列表是通过读取 vCenter Server 根据虚拟机的数量，显示速度会偏慢。

1) 主机管理

万方数据

火龙果 · 整理
uml.org.cn

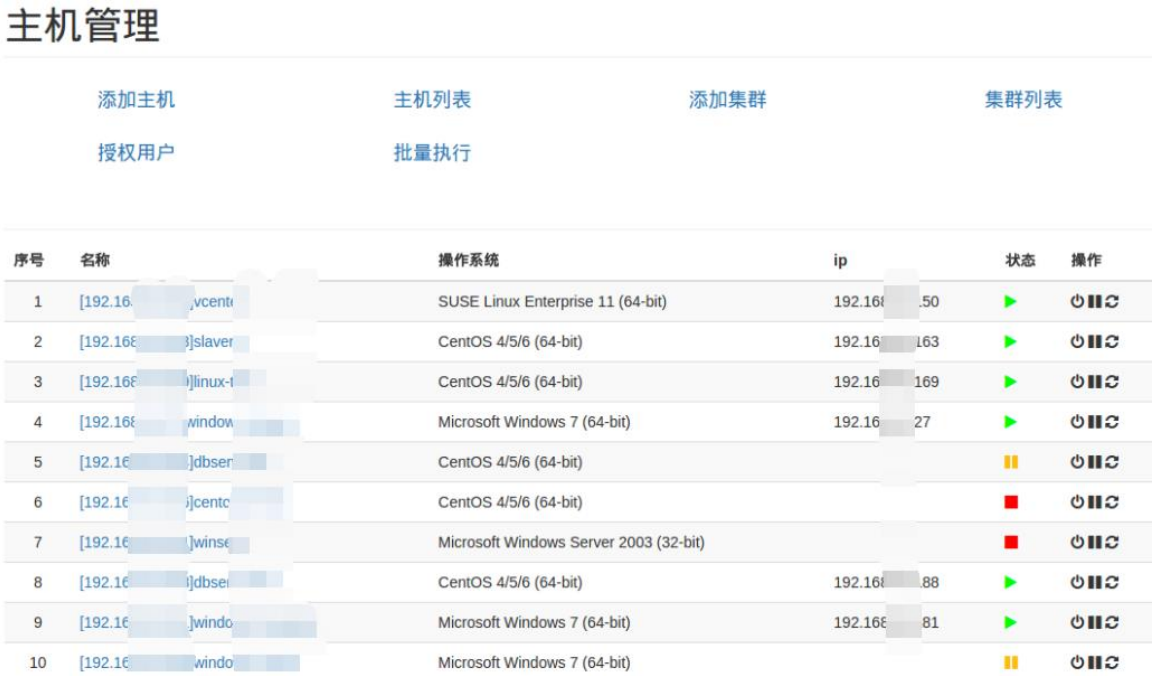


图 5.4 主机列表

主机管理列表界面如图 5.4 所示，依次显示虚拟主机名称、所使用的操作系统、运行时 ip、运行状态以及可以进行的控制操作。

2) 镜像管理

图 5.5 镜像列表

镜像列表如图 5.5 所示，依次显示镜像所在的仓库、标签、镜像 id、镜像文件大小和创建时间，点击操作列的删除可删除镜像，点击详情可以查看镜像详情界面，镜像详情界面显示如图 5.6 所示：

万方数据

火龙果 · 整理
uml.org.cn

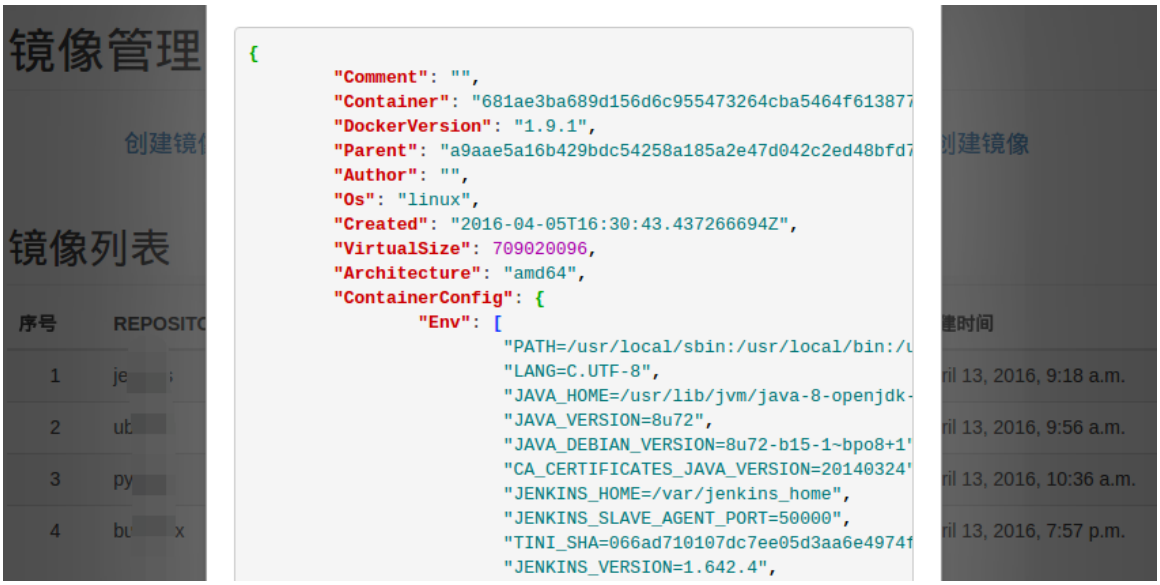


图 5.6 镜像详情

列表内容的上方是管理菜单栏，提供了创建镜像和从 dockerfile 创建镜像的功能，从 dockerfile 创建镜像界面提供一个文本编辑框，修改内容后点击提交即可创建镜像。

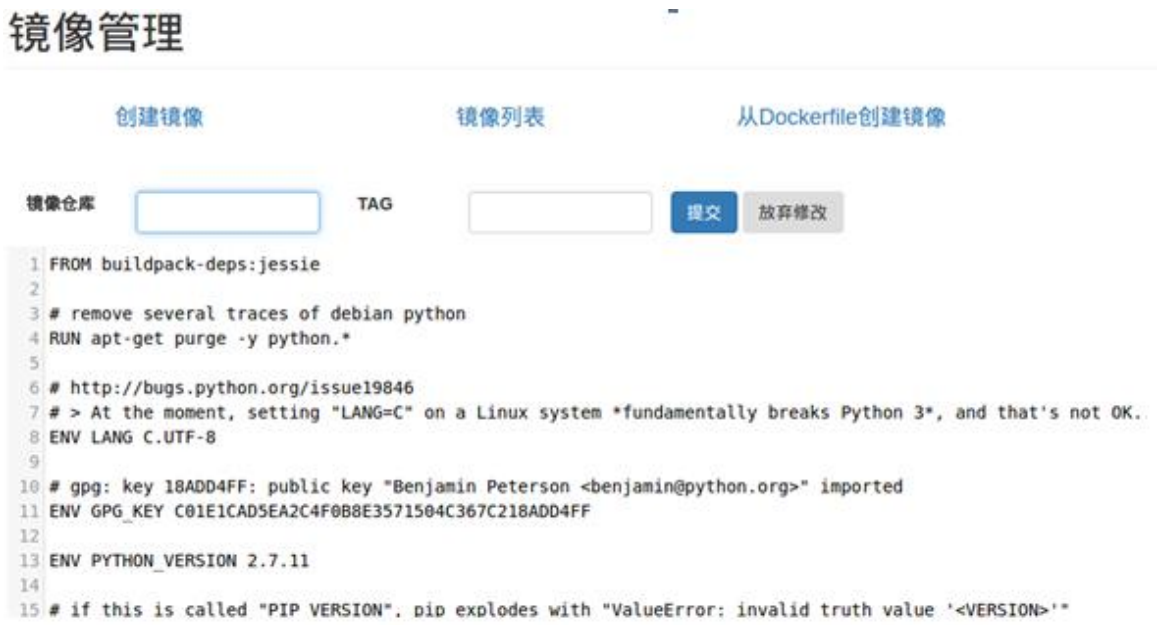


图 5.7 从 dockerfile 创建镜像

3) 容器管理

万方数据

火龙果·整理
uml.org.cn

图 5.8 容器列表

列表界面如图 5.8 所示，依次显示容器的 id、名称、所使用镜像、创建时间、运行状态。最后一列提供启动容器、停止容器、查看容器日志详情和删除容器的操作。

5.2.3 应用管理模块测试分析

根据 5.1 小节中描述的测试方案对用户管理功能进行测试，其各项功能测试结果如表所示：

表 5.4 应用管理模块测试

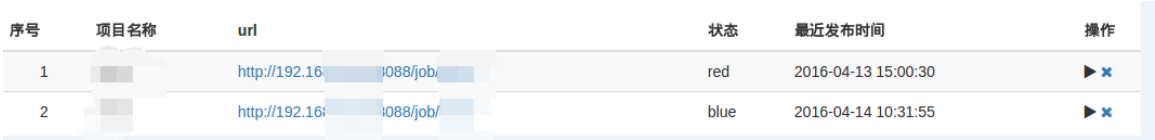
序号	功能	输入与动作	预期响应结果	实际响应结果	平均响应时间（ms）	是否通过
01	svn 项目列表界面	进入 svn 项目列表页面	显示 svn 项目列表页面	显示 svn 项目列表页面	30	是
02	进入创建项目代码仓库页面	进入创建项目代码仓库页面	显示创建项目代码仓库页面	显示创建项目代码仓库页面	42	是
03	创建项目代码仓库	输入项目名称、描述和项目组信息，提交	显示 svn 项目列表页面	显示 svn 项目列表页面	31	是
04	安装 svn 服务	点击安装 svn 服务	显示 svn 项目列表页面，提示创建成功	显示 svn 项目列表页面，提示创建成功	30	是
05	进入 jenkins 管理界面	点击 jenkins 管理功能	显示 jenkins jobs 列表	显示 jenkins jobs 列表	50	是
06	安装 jenkins 服务	点击安装 jenkins 服务	显示 jenkins jobs 列表，提示安装成功	显示 jenkins jobs 列表，提示安装成功	30	是

万方数据

火龙果 · 整理
uml.org.cn

进入应用管理界面，子菜单栏显示可以进行管理的应用，点击进入 svn 管理，可以看见创建仓库修改用户组等功能，点击创建仓库，可以看到信息编辑界面，填写完成仓库名称，添加项目组并设置用户组权限后点击提交，提示创建完成，同时在容器管理界面出现该项目的 svn 容器。

持续集成管理界面如图 5.9 所示，列表显示项目名称，项目管理界面链接、最近一次发布时间和发布状态（red 显示最近一次未构建成功，blue 表示最近一次构建成功），最后一列提供 build job 操作和删除操作，其中点击 url 可以跳转到该项目的 jenkins 管理界面进行工程配置。



序号	项目名称	url	状态	最近发布时间	操作
1		http://192.16 :8088/job/	red	2016-04-13 15:00:30	▶ ✖
2		http://192.16 :8088/job/	blue	2016-04-14 10:31:55	▶ ✖

图 5.9 持续集成管理

5.3 业务测试结果分析

通过对业务功能的测试我们可以得出数据中心在使用本运维平台之前与之后的运维情况，如表 5.5 介绍了在业务运维过程中在使用运维平台前后的对比分析。

表 5.5 使用基于 Docker 服务器运维平台前后对比

对比条目	使用运维平台之前	使用运维平台之后	效果
虚拟机主机	登录 vSphere client，操作复杂过程漫长	登录运维平台，直接界面上点击，设置主机名称，提交即可	配置服务器完成后即可使用直达服务器的应用资源管理
应用	下载、编译、安装。有时为了应用间的隔离，会增加虚拟机	一次完成 docker 镜像，需要环境时直接指定参数启动镜像	大大缩短应用的安装时间，节省磁盘空间
日志查看	登录远程服务器，查看日志记录	运维平台上应用容器一目了然，点击相应的应用即可查看日志	减少登录次数和寻找应用日志的时间

从表中可以看出，在使用了虚拟化技术构建的运维平台之后，解决了第三章提出的需求。业务测试过程整体上功能通过，仍有一下细节问题考虑不够周到。首先用户在使用这些操作的时候，不同镜像之间的连接方式（目前采用 ip 地址）缺乏直观展示，每次都需要去查看 docker 容器详情，连接方式处理不够自动化和直观。其次是镜像与容器数量变多时，难以找到自己的镜像或者容器，检索过程费时。

万方数据

火龙果 · 整理
uml.org.cn

5.4 测试中问题总结

1) 功能类

用户操作完成后的提示和持续监控资源状态不够直观，没有用户专属的控制面板，每次使用镜像或者容器时非常麻烦。

2) 性能类

服务器运维管理平台管理的资源类目较多，一方面要能支持多用户进行操作时 Web 平台的访问；二是所有管理的资源都集中在功能固定的服务器上，虽然 Docker 可以在应用故障时快速启动应用，但是对于容器和镜像资源最好采用分布式的方案架构。后期 Web 平台可以采用分布式服务架构，Docker 服务器也可以采取多主机联动。

3) 安全类

整个平台对服务器的操作权限较大，笔者已经设计了基于角色权限的认证系统，Django 框架自身也有防止跨站攻击的中间件，除此之外在没有其他安全模块的情况下，需要有用户审计支持，所有角色的用户的所有操作都需要形成数据库和文本日志，以便在出现问题时能找到原因和责任人。

5.5 本章小结

本章主要进行了基于 Docker 的服务器运维平台的测试，并分析了测试结果。平台基本功能已经完成，但存在若干问题，本章还对测试过程中出现的问题进行了总结。

万方数据

火龙果 · 整理
uml.org.cn

6 总结与展望

6.1 论文总结

本文针对数据中心研发和运维现状，提出了服务器运维平台的设计需求，设计并实现了基于 Docker 的服务器运维平台。结合虚拟化技术，探索服务器运维方式，解决了数据中心研发运维效率低的问题，提高了研发效率和促进了运维自动化的发展。

本文首先分析了研发和运维过程中存在的问题，结合当前流行了虚拟化技术和 Web 开发技术，提出了基于 Docker 的服务器运维平台的设计需求；接着进行了服务器运维管理平台的的详细设计和实现，完成的主要功能为：

1) 用户管理：用户基础功能、角色、权限、用户组管理；
2) 运维管理：包括镜像管理、容器管理、主机管理；
3) 应用管理：包括代码管理功能和持续集成功能。

本文的亮点是将虚拟化技术进行整合，运用在研发和运维的日常工作中。用 Django 框架和 bootstrap/jQuery 框架实现平台的业务开发，完成整个运维平台的实现。最后对服务器运维平台进行了功能和安全测试，测试结果表明服务器运维平台基本功能已经完成，各项功能测试正常，在一定程度上满足了服务器运维的需求。

6.2 论文展望

到目前为止，本文的研究工作已告一段落，服务器运维平台达到了初期既定的目标，由于项目开发时间和本人研发知识有限，还存在一些需要扩展和改进的地方，简要概述如下：

前端界面展示优化。目前仅仅在界面上显示了功能文字链接，不够美观，在交互提示上不否详细，对于正确提示和错误提示的处理不够友好。后期优化可以丰富界面元素，使用更加人性化的控件提示信息。

主机管理功能完善。目前主机功能中登录远程主机还不够方便，不能直接在网页端显示命令行终端。后期需要实现一个通过网页直接登录远程虚拟机终端的方案；另外在绑定用户上通过密码方式登录不够安全，可以考虑使用 ssh key 的方式执行命令和登录主机。

容器管理功能完善。目前容器都是运行在一台 Docker Server 上，如果 Server 出现单点故障，导致应用无法工作，考虑利用 Open VSwitch 构建多主机 Docker 网络[44]，实现跨主机容器运行，保证应用的稳定。

万方数据

火龙果 · 整理
uml.org.cn

应用管理功能扩展。目前应用管理功能仅涉及了代码管理功能和 Web 应用部署功能。考虑将网络管理、容器编排功能加入进来，实现所有系统无缝对接，提高效率。

关于 Docker 镜像容器的使用，Docker 是基于应用的运维管理模式，它的唯一标示是容器 id，而当前大多数运维系统是基于资源和 ip 的运维模式，所以两种模式基本是不相容的。而且围绕 ip 的运维模式，又衍生出了发布、部署、监控、稳定性、容量评估等等一系列的系统，所以如果要用 Docker，就意味着这些系统都要做对应的改造，这个改造工作量很大，本文只是初步进行了探索，还需要进一步研究。

万方数据

火龙果 · 整理
uml.org.cn

致 谢

两年的研究生生活即将画上句号，这期间有太多难忘的经历，在这里感谢学校，老师，同学们还有我的父母对我的照顾。

由衷感谢我的导师王玉明副教授，本文的完成离不开他的耐心指导。王老师治学严谨，工作认真负责。这两年里，王老师给予我极大的帮助和鼓励，他的教导，使我受益匪浅。在学习和科研工作中，他以身作则，严格要求我们，并打造了一只优秀干练的实验室团队；在为人处事上，他教会了我们学会思考，踏实做事；在生活上，他热心帮助我们解决日常生活中碰到的困难。

感谢实验室的同学们，是他们一直陪伴着我学习奋斗，一起成长，一起收获，在这个大家庭里，我感到非常温馨。本文的完成也离不开他们的支持，感谢他们参与论文设计的讨论和平台测试。

感谢我的朋友们，是他们的支持和关怀，让我的求学生涯不会孤单，在前进的路上我感到充满信心。

最后感谢我的父母，是他们几十年含辛茹苦把我养大成人，在学业上默默支持我，在困难关口鼓励我，让我勇攀高峰，生活上无微不至的照顾让我没有了后顾之忧。他们是我的坚实后盾，我将继续勇往直前。

万方数据

火龙果·整理
uml.org.cn

参考文献

[1] Peter Mell, Timothy Grance. The NIST Definition of Cloud Computing, NIST Special Publication 800-145, 2011. 2-3

[2] 杨小晔. IT 运维管理平台的设计与实现: [硕士学位论文]. 北京: 北京邮电大学, 2011.

[3] 刘瑶. PaaS 云平台技术研究与应用: [硕士学位论文]. 西安: 长安大学, 2015.

[4] 储久良, 李玲. 虚拟化技术在高校数据中心的应用. 实验室研究与探索, 2012(12):67-71

[5] 刘琳羽, 南凯. 一种基于 Docker 的开发者服务平台设计. 科研信息化技术与应用,2015,05:65-72

[6] 谢阳. 基于虚拟化技术的Web服务平台及应用系统研究: [硕士学位论文]. 上海: 东华大学, 2014.

[7] 程伍端. 计算机虚拟化技术的分析与应用. 计算机与数字工程, 2008, 36(11): 175-178.

[8] 李双权, 王燕伟. 云计算中服务器虚拟化技术探讨. 邮电设计技术, 2011,10:27-33.

[9] 杨保华, 戴王剑, 曹亚伦. Docker 技术入门与实站. 北京: 机械工业出版社, 2015. 7

[10] Forbes Guthrie，Scott Lowe，Kendrick Coleman. VMware vSphere Design. Hoboken, New Jersey: John Wiley & Sons, 2013. 106-120

[11] 汤小康. 服务器虚拟化技术在校园网中的应用. 计算机时代, 2009, 2: 14-15.

[12] 王春海. VMware vSphere 企业运维实战. 北京: 人民邮电出版社, 2014. 81

[13] 杜明. 基于 Web 的虚拟化管理方法设计. 电脑知识与技术, 2013, 8: 88

[14] Github. https://github.com/vmware/pyvmomi

[15] 谢超群. Docker 容器技术在高校数据中心的应用. 贵阳学院学报(自然科学版),2015,04:27-29

[16] 王飞. 基于Docker 的研发部署管理平台的设计与实现: [硕士学位论文]. 北京: 北京交通大学, 2015.

[17] 陈春霞. 基于容器的微服务架构的浅析. 信息系统工程,2016,03:95-96.

[18] 张建, 谢天钧. 基于 Docker 的平台即服务架构研究. 信息技术与信息化, 2014,10:131-134.

万方数据

火龙果·整理
uml.org.cn

[19] 刘思尧,李强,李斌. 基于 Docker 技术的容器隔离性研究. 软件,2015,04:110-113.
[20] Github. https://github.com/docker/docker-py
[21] Dua R, Raja A R, Kakadia D. Virtualization vs containerization to support paas//Cloud Engineering (IC2E), 2014 IEEE International Conference on. IEEE, 2014: 610-614
[22] 刘克. MVC 架构及其在 Web 应用开发中的应用. 计算机应用与软件. 2006(07): 57-59
[23] 高昂, 魏惠茹, 李晓东等. MVC 设计模式研究. 电脑知识与技术, 2016, 01: 88-89
[24] Leff A, Rayfield J T. Web-application development using the model/view/controller design pattern. In:Enterprise Distributed Object Computing Conference, 2001: 118-127
[25] 姚娜. 基于 Django 的自动化运维管理系统的设计与实现: [硕士学位论文]. 西安: 西安电子科技大学, 2015.
[26] Magnus Lie Hetland. Python 基础教程（第 2 版）. 司维, 曾军崴, 谭颖华. 北京: 人民邮电出版社, 2014. 261-273
[27] 贾志娟. 基于 Django 框架的软件自动化测试分布式部署系统的研究与实现: [硕士学位论文]. 北京: 北京邮电大学,2012.
[28] Hu H, Jiang X, Zhu Q, et al. A Cloud Computing Based Management System for Wireless Mesh Networks[C]//High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on. IEEE, 2013: 1209-1212
[29] 李绘英. 基于 WSGI 规范的 Web 数据库应用系统研发: [硕士学位论文]. 武汉: 武汉理工大学, 2012.
[30] Ni S, Lin Z, Cai Q, et al. Design of Distributed and High Concurrency Log System Based on MongoDb. 2014, 4(11): 262-268
[31] 聂鹏. 基于 Nginx 的云计算访问控制网关的设计与实现: [硕士学位论文]. 北京: 北京交通大学, 2011.
[32] 黄競. 基于 jQuery 框架的 Web 前端系统构建方法的研究与应用: [硕士学位论文]. 北京: 北京邮电大学, 2013.
[33] Anthony T Holdener III. Ajax the Definitive Guide. O'Reilly Media, Inc, 2008. 1-70

万方数据

火龙果 · 整理
uml.org.cn

[34] Fowler M, Foemmel M. Continuous integration, 2006[J]. URL http://www. martinfowler. com/articles/continuousIntegration. html, 2012.

[35] 卞孟春. 基于 Jenkins 的持续集成方案设计与实现: [硕士学位论文]. 北京: 中国科学院大学（工程管理与信息技术学院）, 2014.

[36] 陈刚, 羌铃铃. 软件项目开发中的持续集成研究. 项目管理技术, 2011, 12: 103-106

[37] 余浩维. PaaS 云中 Web 容器及调度的设计与实现: [硕士学位论文]. 北京: 北京邮电大学, 2015.

[38] Duvall P M, Matyas S, Glover A. Continuous integration: improving software quality and reducing risk. State of New Jersey: Pearson Education, 2007.15-20

[39] 胡宸. IT 项目管理系统的设计与实现: [硕士学位论文]. 成都: 电子科技大学, 2013.

[40] Vohra D. Pro Docker. New York: Apress, 2016: 50-93.

[41] Mouat A. Using Docker. California: O'Reilly 2015: 100-201

[42] 张存超, 任洪敏. 基于 Web 的 SVN 权限管理系统的设计与实现. 微型机与应用, 2013, 11:22-25.

[43] 高俊峰. 高性能 Linux 服务器构建实战. 北京: 机械工业出版社, 2014. 267-296

[44] PFAFF, B., PETTIT, J., KOPONEN, T., JACKSON, E., ZHOU, A., RAJAHALME, J., GROSS, J., WANG, A., STRINGER, J., SHELAR, P., AMIDON, K., AND CASADO, M. The Design and Implementation of Open vSwitch. In USENIX NSDI (2015)

万方数据