



西南科技大学

Southwest university of science and technology

# 本科毕业设计（论文）

## 基于 Docker 的分布式应用控制系统 设计与实现

学 院 名 称	计 算 机 科 学 与 技 术 学 院
专 业 名 称	软 件 工 程
学 生 姓 名	罗 雪
学 号	20121179
指 导 教 师	杨 雷 讲 师

二〇一六年六月

# 基于 Docker 的分布式应用控制系统 设计与实现

**摘要：**Docker 是近年来兴起的环境部署工具，因其可以实现可移植的应用部署而受到广大运维和开发的欢迎。Docker 相关操作较多，对应的 Linux 命令也比较复杂，不便记忆。但 Remote API 的产生解决了这一问题，根据 Remote API 可以开发出 Docker 相关操作的可视化控制系统。系统主要包括主机管理、容器管理、镜像管理和权限管理等模块，采用前后端完全分离的开发模式，前后台开发框架分别为 Angular 和 ThinkPHP，后台使用 PHP curl 向 Docker Server 发送 GET/POST/DELETE 等请求。该系统几乎可以代替 Docker 命令行操作，这大大减少了系统部署和维护的时间，主机中容器和镜像运行情况也可以更加直观的呈现，有利于更加快速的发现运行问题，减轻运维工作。系统通过测试，满足了运维高效维护的要求，达到了设计目标。

**关键词：**Docker； 容器； 镜像； Remote API； 可视化控制

---

# Docker-based Distributed Application Control System

## Design and Implementation

**Abstract:** Docker is an environmental deployment tool in recent years. Docker is very popular in operation engineers and programmers, because it can achieve portable application deployment. Docker operations are quite a lot, the corresponding linux commands are also complicated, which is inconvenience memory. And Remote API produced solved the problem, according to which it can develop a control system of Docker operations. The system mainly includes the host management, container management, image management and rights management module, using the foreground and background side of the complete separation development model, the development framework for Angular and ThinkPHP, and background make GET/POST/DELETE calls through PHP curl. The system can replace almost docker commands, which greatly reduced deployment and maintenance time, also, container and image running conditions can be more intuitive, and it is conducive to more quickly discover the running problem, reduce the maintenance's work. The system is tested, it meets the requirements of the efficient maintenance and achieves the design goal.

**Key words:** Docker, container, image, Remote API, visual control

# 目 录

第 1 章 绪论.....	1
1.1 选题研究的目的和意义.....	1
1.2 国内外选题相关研究现状.....	1
1.3 选题目标.....	1
1.4 选题实施方案.....	2
1.5 本章小结.....	2
第 2 章 应用控制系统相关技术.....	3
2.1 系统后台开发技术.....	3
2.1.1 开源容器引擎 Docker.....	3
2.1.2 Docker Remote API.....	3
2.1.3 CURL 编程.....	4
2.1.4 后台开发框架 ThinkPHP.....	4
2.2 系统前端开发技术.....	4
2.2.1 前端开发框架 Angular.....	4
2.2.2 前端样式框架 Bootstrap.....	5
2.2.3 前端代码模块化 Require.....	5
2.2.4 前端工程构建工具 Gulp.....	5
2.3 本章小结.....	5
第 3 章 应用控制系统需求分析.....	7
3.1 术语约定.....	7
3.2 需求概述.....	7
3.3 系统业务需求分析.....	7
3.4 系统用户需求分析.....	7
3.5 系统功能需求分析.....	8
3.5.1 主机管理.....	8
3.5.2 容器管理.....	12
3.5.3 镜像管理.....	16
3.5.4 系统日志.....	18

3.5.5 用户管理.....	18
3.6 系统非功能性需求.....	19
3.7 本章小结.....	20
<b>第 4 章 应用控制系统概要设计.....</b>	<b>21</b>
4.1 系统架构设计.....	21
4.2 数据库设计.....	22
4.2.1 数据库环境设计.....	22
4.2.2 命名规则.....	22
4.2.3 实体.....	23
4.2.4 实体之间的关系.....	24
4.2.5 表设计.....	25
4.3 本章小结.....	27
<b>第 5 章 应用控制系统详细设计与实现.....</b>	<b>28</b>
5.1 总体设计.....	28
5.1.1 开发环境.....	28
5.1.2 系统架构.....	28
5.2 主机管理详细设计与实现.....	28
5.2.1 主机管理模块设计.....	28
5.2.2 创建主机.....	29
5.2.3 删除主机.....	31
5.2.4 查询主机.....	32
5.2.5 创建主机分组.....	33
5.2.6 删除主机分组.....	34
5.2.7 查询主机分组.....	35
5.2.8 更改主机所在分组.....	36
5.3 容器管理详细设计与实现.....	37
5.3.1 容器管理模块设计.....	37
5.3.2 创建容器.....	38
5.3.3 操作容器.....	39
5.3.4 查询容器日志.....	41

5.3.5 查询容器进程.....	41
5.3.6 查询容器列表.....	43
5.3.7 查询容器详细信息.....	44
5.3.8 重命名容器.....	45
5.4 镜像管理详细设计与实现.....	46
5.4.1 镜像管理模块设计.....	46
5.4.2 创建镜像.....	46
5.4.3 查询镜像.....	48
5.4.4 删除镜像.....	49
5.5 系统日志管理详细设计与实现.....	50
5.5.1 系统日志管理模块设计.....	50
5.5.2 查询系统日志.....	50
5.6 用户管理详细设计与实现.....	51
5.6.1 用户管理模块设计.....	51
5.6.2 用户登录.....	52
5.6.3 退出系统.....	53
5.6.4 获取用户列表.....	53
5.4 本章小结.....	54
第 6 章 应用控制系统测试.....	55
6.1 测试环境.....	55
6.2 测试原则.....	55
6.3 功能性测试.....	55
6.3.1 主机管理模块.....	55
6.3.2 容器管理模块.....	59
6.3.3 镜像管理模块.....	64
6.3.4 用户管理模块.....	65
6.3.5 系统日志管理模块.....	65
6.4 兼容性测试.....	66
6.5 测试过程中发现的问题总结与分析.....	67
6.6 本章小结.....	67

---

结论.....	68
致谢.....	69
参考文献.....	70

# 第 1 章 绪论

## 1.1 选题研究的目的和意义

环境搭建与部署是产品实际开发过程中的第一步,其操作过程极易产生错误,如:在若干不同版本操作系统、不同配置的机器上搭建无差异化开发,测试环境难度高;产品研发环境与发布环境往往不同,正式上线通常会出现难以预料的问题,产品发布风险较高。项目扩大的过程中,参与人员流动性较大,每个人都要部署自己的开发环境,着实浪费时间。而且人为操作存在不可避免的失误,改正这些失误需要消耗更多的时间,代价较大,而 Docker 的出现,则解决了这些让人头疼的问题的。

Docker 的所有操作都只能在 Linux 系统下进行,环境部署发布需要记忆复杂的 Linux 命令,对于不清楚 Docker 运行原理的开发和运维需要大量的时间理清其原理,同时,使用起来也非常吃力,如果存在一个 Docker 可视化管理工具,那就事半功倍了,无需 Linux 基础,无需记忆复杂的 Linux 命令,只需简单的鼠标和键盘即可完成一项操作。

运维和开发要想查看多个服务器中镜像容器的运行情况,必须要依次访问进行查看,在选题系统中,无需这样麻烦,如:打开容器管理项即可查看各个主机对应容器列表信息,镜像管理也是如此,大大节约了运维时间。

## 1.2 国内外选题相关研究现状

目前国内外已有 Docker 可视化管理工具有 Kubernetes、Dcokersh、DcokerUI、Shipyard 等,这些工具存在的同时也让 Docker 变得越来越强大。既然已经有了这些扩展工具,为什么还要开发一款?工具多种多样,总有些功能不能满足公司项目的需求,同时由于 Docker 本身的安全系数不高,为了保证公司内部机密不外泄,为自己公司量身定做一款 Docker 开发工具也是有必要的。本项目开发主要是针对实习公司内部需要进行 Docker 管理工具开发,以便提高公司后期开发、测试、运维的工作效率。

## 1.3 选题目标

选题要实现通过 Remote API 对 Docker 容器和镜像的可视化操作,实现功能包括容器管理和镜像管理,为了方便同时查看多个 Docker Server 运行情况,还需要实现



主机管理模块。环境部署属于公司机密，只能特定的人员才能进行操作，所以权限管理也是必要的。实现这些基本功能后，系统便可以对多个 Docker Server 进行基本的操作。

## 1.4 选题实施方案

1、了解 Docker 相关的基本知识及其基本工作原理和工作流程，在此基础上，熟悉 Docker Remote API，根据 PHP curl 编程实现向 Docker Server 发送 POST/GET/DELETE 等请求，从而远程对容器和镜像进行操作。

2、学习类似系统（如 shipyard）展现方式和风格，并作出对比，最后制定出适合公司内部最优方案。

3、进行系统总体设计，如整个系统框架结构，开发需要使用的技术。

4、进行系统详细设计，如系统功能模块设计，数据库设计等。

5、根据总体设计和详细设计，实现对应系统功能。

6、系统测试，包括安全测试和功能测试。

## 1.5 本章小结

本章介绍了选题的研究目的意义及相关研究现状，根据这些明确了选题目标和实施方案，为后期系统开发指明了方向。同时，确定了选题目标，根据选题目标给出了选题的实施方案。经过本章介绍，选题是什么，做什么，如何做等疑问有了清晰的回答。

## 第 2 章 应用控制系统相关技术

### 2.1 系统后台开发技术

#### 2.1.1 开源容器引擎 Docker

Docker<sup>[1]</sup>是一种容器技术，同时也被称为轻量级虚拟技术或者容器型虚拟技术。它最大的特点就是“Build Once, Run Anywhere”。容器与容器之间相互隔离，没有任何接口，完全使用沙箱机制，性能开销极低，运行速度相当快。开发或者运维可以打包自己的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上。形象一点比喻，容器就是集装箱，部署人员把代码打包到容器集装箱中，Docker 就充当搬运工，把应用运到各个地方。

服务器 Docker Server 启动后，无需进行 Linux 命令操作，在控制系统中便可以对 Server 中的容器和镜像进行操作。

#### 2.1.2 Docker Remote API

The Remote API has replaced rcli<sup>[2]</sup>. Docker Remote API 的作用便在于代替 rcli 命令行操作，在控制系统上发送相应请求。API 列表如下：

表 2-1 Remote API

操作	Remote API	请求方式
List containers	/containers/json	GET
Create a container	/containers/create	POST
Inspect a container running inside a container	/containers/(id)/json	GET
List processes	/containers/(id)/top	GET
Get container logs	/containers/(id)/logs	GET
Start a container	/containers/(id)/start	POST
Stop a container	/containers/(id)/stop	POST
Restart a container	/containers/(id)/restart	POST
Kill a container	/containers/(id)/kill	POST
Remove a container	/containers/(id)/remove	DELETE
Rename a container	/containers/(id)/rename	POST
Pause a container	/containers/(id)/pause	POST
Unpause a container	/containers/(id)/unpause	POST
Create an image	/images/create	POST
List Images	/images/json	GET
Remove an image	/images/(name)	DELETE

### 2.1.3 CURL 编程

Curl 语言是由美国的麻省理工学院开发的一种被设计来编写网络程序的编程语言，编程效率高，是一种支持多重继承、范型等数据类型的面向对象编程语言。用 Curl 写的程序既可以运行于浏览器中，又可以像普通客户端程序那样独立于浏览器运行。

PHP 中包含 Curl 库，有很多相关 API，如：curl\_init()、curl\_version 等。要在 PHP 中进行 curl 编程，需要单独开启 curl 库，通过 php\_info()可以验证，要开启 curl 库也比较简单，只需取消行注释 extension=php\_curl.dll。

系统主要使用 curl 编写 Docker SDK，向 Docker Server 发送 GET/POST/DELETE 等请求，进而操作 Docker Server 中的镜像和容器。

### 2.1.4 后台开发框架 ThinkPHP

ThinkPHP 是一个免费开源的，快速、简单的面向对象的轻量级 PHP 开发框架，遵循 Apache2 开源协议，是为了敏捷 WEB 应用开发和简化企业应用开发而诞生的。ThinkPHP 从诞生以来一直秉承简洁实用的设计原则，在保持出色的性能和至简的代码的同时，也注重易用性。并且拥有众多的原创功能和特性，在社区团队的积极参与下，在易用性、扩展性和性能方面不断优化和改进，众多的典型案例确保可以稳定用于商业以及门户级的开发。

ThinkPHP 可以支持 windows/Unix/Linux 等服务器环境，正式版需要 PHP5.0 以上版本支持，同时支持 MySQL、PgSQL、Sqlite 以及 PDO 等多种数据库，ThinkPHP 框架本身没有什么特别模块要求，具体的应用系统运行环境要求视开发所涉及的模块。系统使用 Windows + ThinkPHP + Nginx + MySQL 进行开发。

## 2.2 系统前端开发技术

### 2.2.1 前端开发框架 Angular

AngularJS 是一款由 Google 维护的开源 JavaScript 库，用来协助单一页面应用程序运行。其最大的特性为双向数据绑定，允许模型和视图之间的自动同步，且应用逻辑与 DOM 操作解耦，这使得对 DOM 的操作不再重要并且提升了可测试性。将应用程序的客户端与服务器端分离，这允许客户端和服务器的开发可以同时进行。指导开发者完成构建应用程序的整个历程：从用户界面的设计，到编写业务逻辑，再到测

试。

由于 Angular1.3+已经不支持 IE8 及以下的浏览器,故而选题选择了 Angular 1.2.27 作为前端开发框架。

### 2.2.2 前端样式框架 Bootstrap

Bootstrap 基于 HTML、CSS、JAVASCRIPT,是目前很受欢迎的前端框架。它简洁灵活,使 Web 开发更加快捷。它由 Twitter 的设计师 Mark Otto 和 Jacob Thornton 合作开发,是一个 CSS/HTML 框架。Bootstrap 提供了优雅的 HTML 和 CSS 规范,它即是由动态 CSS 语言 Less 写成。

使用 Bootstrap 能够大大减少前端 UI 开发时间,故系统使用 Bootstrap 进行 UI 开发。

### 2.2.3 前端代码模块化 RequireJS

前端工程由 JS 编写,如果不使用模块化编程方式,全局污染将永久存在,因此,模块化编程前端工程中显得尤为重要。系统使用目前国内外比较权威的模块化工具 RequireJS。

### 2.2.4 前端工程构建工具 Gulp

前端工程构建工具的翘首无非就是 Gulp 和 Grunt 这两种。Gulp 是一种基于流的自动化构建工具。可以无需像使用 Grunt 那样书写配置文件,直接在 gulpfile.js 中书写简单易懂的 JS 代码即可完成配置。Gulp 还可以同时运行多个不相干的任务,这是 Grunt 不能做到的。相比 Grunt, Gulp 更容易让前端开发人员上手构建项目。

雅虎网站性能优化的建议中有减少 HTTP 请求数量、优化图片等, Gulp 可以通过 gulp-concat 和 gulp-uglify 对 JS、CSS 进行合并压缩,减少 HTTP 请求数量,也可以通过 gulp-imagemin 对图片进行压缩,减少图片大小,加快图片加载速度。

由于 less 拥有其独特的语法,可以进行变量定义和类的共用等,故而在前端开发中比较盛行,但 Less 在网页中编译速度较慢,会影响页面响应速度,通过 gulp-less 可以在开发阶段随时将 Less 编译成 CSS,由此在技术和速度之间达成一致。

## 2.3 本章小结

本章主要介绍了基于 Docker 的分布式应用控制系统的相关技术,包括 Docker、Curl 编程、ThinkPHP、Angular、Gulp 等。Docker 作为服务器监听某一端口,ThinkPHP

（含 curl 编程）开发的 Dcoker 客户端向该端口发送请求（Remote API），Dcoker 服务端执行相应的 Curl 命令，执行结果返回给前端，以 Angular 作为开发框架的前端工程将结果显示在浏览器中。

## 第 3 章 应用控制系统需求分析

### 3.1 术语约定

系统涉及到一些不常用的知识点，故写下术语约定，降低读者阅读难度。

表 3-1 术语约定

名词	解释
Docker	容器引擎
Container	容器
Image	镜像
Registry	仓库

### 3.2 需求概述

基于 Docker 的分布式应用控制系统提供 Docker Remote API 相关的可视化操作，简化操作的同时，能够更直观地展示容器和镜像信息列表，方便了用户对容器和镜像的管理。

系统主要展示最新的主机列表信息、容器列表信息、镜像列表信息，登录用户在查看这些信息的同时可以对其进行其它操作，如：查看系统日志、连接主机、主机分组、拉取镜像、删除镜像、创建容器、删除容器、暂停容器、移除容器、重启容器、终止容器、查看容器日志和进程，且可对任意数量的容器进行批量操作。

### 3.3 系统业务需求分析

使用 Docker 进行开发、测试、产品上线部署等过程的环境配置时，需要在 Linux 环境中操作镜像和容器，每次都要输入长长的并不容易记忆命令行，对于 Linux 环境基础比较薄弱的同学来说，并不是很友好，熟练掌握 Linux 的开发和测试也并不多。基于 Docker 的分布式应用控制系统技工的可视化操作，不仅操作简便，信息显示工整，便于查询，对于 Docker 初学者减少了 Linux 这一道门槛。

### 3.4 系统用户需求分析

从系统用户角色来看，开发主要针对项目成员和项目管理员两大类型用户，其中项目成员可以是开发人员、测试人员和运维人员。

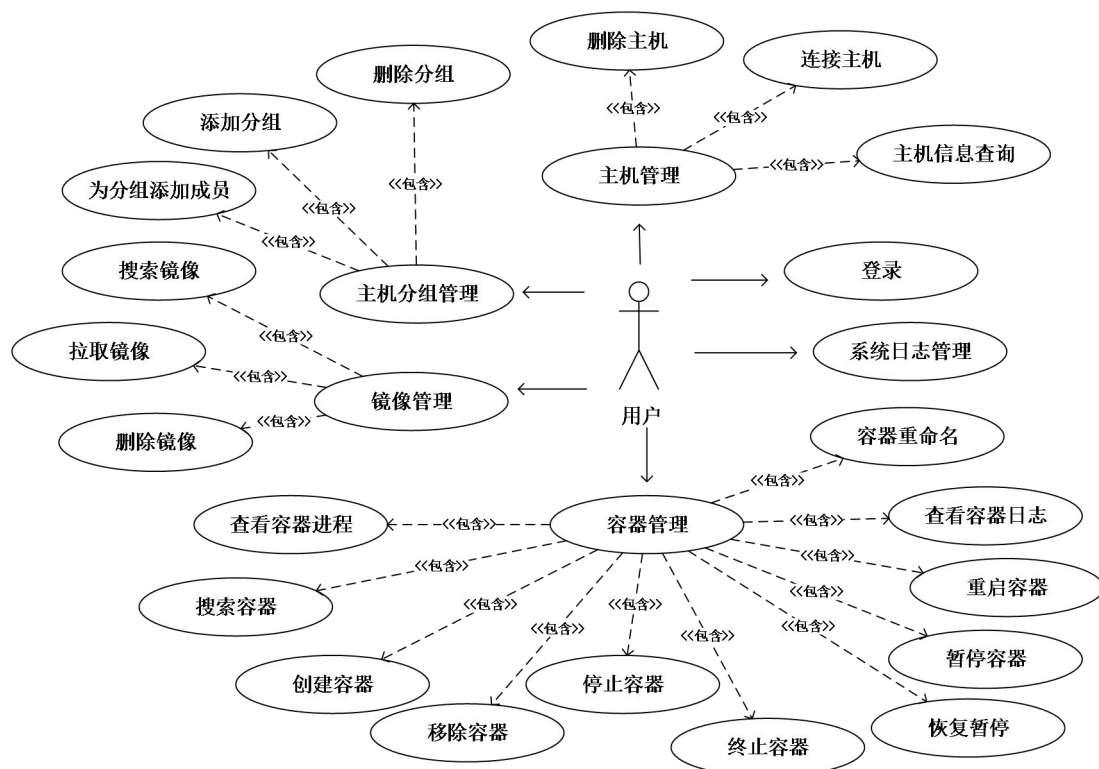


图 3-1 系统用例图

### 3.5 系统功能需求分析

基于 Docker 的分布式应用控制系统主要包括主机管理、Container 管理、Image 管理以及系统日志等模块。

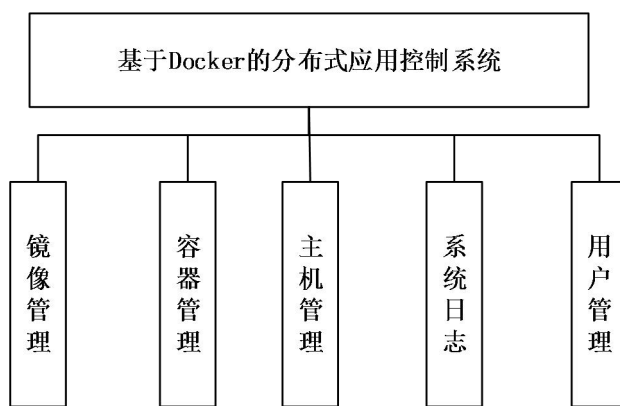


图 3-2 系统功能模块图

#### 3.5.1 主机管理

系统会显示主机列表以及主机分组信息，用户可以查看到某一主机具体在哪个分组，主机分组是可以进行动态创建和删除的。具体功能描述见表 3-2 到表 3-8。

表 3-2 创建主机

功能名称	创建主机	参与者	管理员
功能描述	用户填写好主机相关信息后，提交“创建”请求即可创建一个新主机。需要填写的主机信息包括：主机名、主机 IP、主机端口号、主机分组（默认为未分组）。其中，主机名只能由数字、字母和下划线组成，主机 IP 和端口号符合对应的数字要求，主机分组不能输入只能选择。		
前置条件	用户已登录系统		
基本事件流	1. 用户输入主机信息，并提交“创建”请求 2. 系统验证输入数据是否有效 3a. 系统验证通过，创建新主机 4. 系统返回“创建成功”提示信息		
异常事件流	3b. 系统验证不通过，给出具体字段的错误提示信息		
后置条件	系统记录操作日志，并显示最新的主机列表		

表 3-3 删除主机

功能名称	删除主机	参与者	管理员
功能描述	用户可以删除不再适用的主机，一旦删除便不可恢复，主机列表中不再显示，所以需要提醒用户慎重操作。一次只能删除一个主机，不能进行批量操作。		
前置条件	用户已登录系统		
基本事件流	1. 用户选择待删除的主机，提交“删除”请求 2. 系统删除用户选择的主机 3a. 系统返回“删除成功”提示信息		
异常事件流	3b. 若系统删除过程中出现未知错误，返回“删除失败”提示信息		
后置条件	系统记录操作日志，并显示最新主机列表信息		

表 3-4 查看主机列表

功能名称	查看主机列表	参与者	管理员
功能描述	系统会显示主机列表信息，信息内容包括：主机名、主机 IP、主机端口、主机所在分组、运行中的容器数量、停止运行的容器数量、镜像数量。		



(续表 3-4)

前置条件	用户已登录系统
基本事件流	1. 用户请求显示主机列表 2a. 系统显示主机列表信息
异常事件流	2b. 由于某些未知原因，主机列表信息拉取失败，返回“主机列表信息获取失败，请刷新重试！”提示信息。
后置条件	系统记录操作日志

表 3-5 添加主机分组

功能名称	添加主机分组	参与者	管理员
功能描述	主机分组主要用于对主机进行分组管理，方便主机查询和使用。添加主机分组只需填写分组名称，分组名称只能由数字、字母和下划线组成，且不可重复。默认有一个分组，组名为“未分组”。分组删除后，该分组的所有主机归为未分组。		
前置条件	用户已登录系统		
基本事件流	1. 用户输入组名，提交“添加”请求 2. 系统验证组名是否符合要求 3a. 系统验证通过 4. 系统返回“分组添加成功”提示信息		
异常事件流	3b. 系统验证失败，返回失败提示信息，如“组名只能由数字、字母和下划线组成”、“该分组已存在”。		
后置条件	系统记录操作日志，并显示主机分组列表		

表 3-6 删除主机分组

功能名称	删除主机分组	参与者	管理员
功能描述	如果主机分组命名不合适或者不再需要该分组，用户可删除该分组，分组一经删除不可恢复，所以要提醒用户慎重选择。主机分组删除后，该分组的所有主机归为未分组。		
前置条件	用户已登录系统		

(续表 3-6)

基本事件流	1. 用户选择待删除的主机分组，发送“删除”请求 2a. 系统删除用户选择的主机分组，并将该分组的所有主机分组置为“未分组” 3. 系统返回“删除成功”提示信息
异常事件流	2b. 若系统删除分组过程中出现未知错误，则返回提示信息“删除失败，请再次尝试！”
后置条件	系统记录操作日志，并显示主机分组列表

表 3-7 查看主机分组列表

功能名称	查看主机分组列表	参与者	管理员
功能描述	系统会显示所有分组，信息只包含分组名称		
前置条件	用户已登录系统		
基本事件流	1. 用户请求显示所有分组 2a. 系统显示所有分组名称		
异常事件流	2b. 系统显示出现未知错误，返回提示信息“显示分组列表错误，请刷新重试”。		
后置条件	系统记录操作日志		

表 3-8 更改主机所在分组

功能名称	更改主机所在分组	参与者	管理员
功能描述	由于主机用途可以随时更换，主机所在分组也需要跟随需求做出相应更改，故给出该功能。更换分组时只能选择不能输入。		
前置条件	用户已登录系统		
基本事件流	1. 用户找到待更改分组的主机，并选择目标分组 2a. 系统根据用户选择将主机所在分组设置为目标分组		
异常事件流	2b. 系统设置分组时出现未知错误，返回提示信息“更改分组出现错误，请重试！”		
后置条件	系统记录操作日志，并显示最新主机列表（包含最新主机分组）		

### 3.5.2 容器管理

容器（container）是 Docker 技术的核心，所以容器操作比较多。用户除了可以搜索查看容器列表信息外，还可以对容器进行启动、重启、暂停、恢复暂停、重命名、终止、移除等操作。不同的操作，容器也会对应不同状态：停止、终止、暂停运行中。容器操作一般会生成记录，所以可以通过查看容器日志排查操作异常。容器运行过程中，查看容器进程也是必不可少的。容器状态转换图如图 3-3 所示。

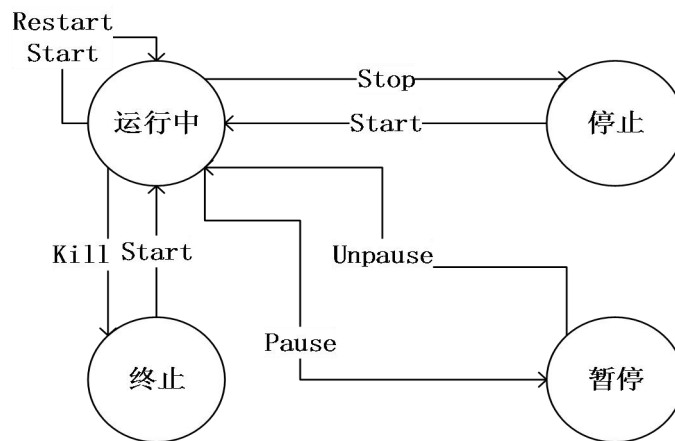


图 3-3 容器状态图

具体功能描述见表 3-9 到表 3-20。

表 3-9 创建容器

功能名称	创建容器	参与者	管理员
功能描述	创建容器，也叫部署容器。在创建容器之前，对应的镜像文件必须已经存在。代码打包只能在已创建好的容器中运行。容器信息较多，每次创建无需输入每个信息，只需输入几个必要的信息即可。		
前置条件	用户已登录系统，对应镜像文件已创建完成		
基本事件流	1. 用户输入容器基本信息，并发送“创建”容器请求 2. 系统验证输入信息 3a. 系统验证通过 4. 返回“容器创建成功”提示信息		
异常事件流	3b. 系统验证不通过，返回创建失败的原因		
后置条件	系统记录操作日志		

表 3-10 启动容器

功能名称	启动容器（start）	参与者	管理员
功能描述	“停止”、“终止”和“运行中”状态的容器都能被启动。启动容器后，容器状态变为“运行中”。“暂停”状态的容器不能被启动，只能使用“恢复暂停”操作才能继续运行容器。可选择多个容器批量执行该操作。		
前置条件	用户已登录系统，容器状态不是“暂停”		
基本事件流	1. 用户选择要启动的容器，并发送“启动”容器请求 2a. 系统启动相应容器		
异常事件流	2b. 启动过程中出现未知错误，系统返回“容器启动错误，请重试！”。		
后置条件	系统记录操作日志		

表 3-11 暂停容器

功能名称	暂停容器（pause）	参与者	管理员
功能描述	只有“运行中”状态的容器可以被暂停，被暂停后，也只能通过恢复暂停操作，让容器继续运行。可选择多个容器批量执行该操作。		
前置条件	用户已登录系统，容器状态为“运行中”		
基本事件流	1. 用户选择要暂停的容器，并发送“暂停”容器请求 2a. 系统暂停相应容器		
异常事件流	2b. 暂停过程中出现未知错误，系统返回“容器暂停错误，请重试！”。		
后置条件	系统记录操作日志		

表 3-12 恢复暂停容器

功能名称	恢复暂停容器（unpause）	参与者	管理员
功能描述	只有“暂停”状态的容器能够被恢复暂停，可选择多个容器批量执行该操作。		
前置条件	用户已登录系统，容器状态为“暂停”		
基本事件流	1. 用户选择要恢复暂停的容器，并发送“恢复暂停”容器请求 2a. 系统恢复暂停相应容器		
异常事件流	2b. 恢复暂停过程中出现未知错误，系统返回“容器恢复暂停错误，请重试！”。		
后置条件	系统记录操作日志		

表 3-13 停止容器

功能名称	停止容器 (Stop)	参与者	管理员
功能描述	“暂停”状态的容器不能被停止；“运行中”状态的容器可以被停止，停止容器后，容器状态变为“停止”；可选择多个容器批量执行该操作。		
前置条件	用户已登录系统，且主机已连接。容器不能为“暂停状态”。		
基本事件流	1. 用户选择要停止的容器，并发送“停止”容器请求 2a. 系统停止相应容器		
异常事件流	2b. 停止过程中出现未知错误，系统返回“容器停止错误，请重试！”。		
后置条件	系统记录操作日志		

表 3-14 终止容器

功能名称	终止容器 (Kill)	参与者	管理员
功能描述	只能终止运行状态的容器；“暂停”、“停止”、“终止状态”均不能进行终止容器操作。可选择多个容器批量执行该操作。		
前置条件	用户已登录系统，且主机已连接。容器状态为“运行”。		
基本事件流	1. 用户选择要终止的容器，并发送“终止”容器请求 2a. 系统终止相应容器		
异常事件流	2b. 终止过程中出现未知错误，系统返回“容器终止错误，请重试！”。		
后置条件	系统记录操作日志		

表 3-15 重启容器

功能名称	重启容器(Restart)	参与者	管理员
功能描述	只有“暂停”状态的容器不能被重启；“运行中”、“停止”、“终止”状态的容器都可以被重启。可选择多个容器批量执行该操作。		
前置条件	用户已登录系统，且主机已连接。容器状态不为“暂停”。		
基本事件流	1. 用户选择要重启的容器，并发送“重启”容器请求 2a. 系统重启相应容器		
异常事件流	2b. 重启过程中出现未知错误，系统返回“容器重启错误，请重试！”。		
后置条件	系统记录操作日志		

表 3-16 移除容器

功能名称	移除容器 (Remove)	参与者	管理员
功能描述	任何状态的容器都可以被移除。可选择多个容器批量执行该操作。		
前置条件	用户已登录系统，且主机已连接		
基本事件流	1. 用户选择要移除的容器，并发送“移除”容器请求 2a. 系统移除相应容器		
异常事件流	2b. 移除过程中出现未知错误，系统返回“移除容器错误，请重试！”。		
后置条件	系统记录操作日志		

表 3-17 查看容器日志

功能名称	查看容器日志	参与者	管理员
功能描述	用户可以查看某一容器的运行日志，进行错误排查		
前置条件	用户已登录系统，且主机已连接		
基本事件流	1. 用户选择某一容器，发送请求“显示”容器日志 2a. 系统显示对应容器的日志列表		
异常事件流	2b. 容器日志显示过程中出现未知错误，返回提示信息“容器日志获取失败，请重试”。		
后置条件	系统记录操作日志		

表 3-18 查看容器进程

功能名称	查看容器进程	参与者	管理员
功能描述	用户可以查看某一容器的进程，进程信息包括：UID/ PID/ PPID/ C/ STIME/ TTY/ TIME/ CMD。		
前置条件	用户已登录系统，且主机已连接		
基本事件流	1. 用户选择目标容器，发送“显示”容器进程请求 2a. 系统显示容器进程列表		
异常事件流	2b. 系统显示容器进程过程中出现未知错误，返回提示“获取容器进程失败，请重试！”		
后置条件	系统记录操作日志		

表 3-19 查看容器列表

功能名称	查看容器列表	参与者	管理员
功能描述	显示的容器列表信息包括：容器编号（Id）、名字(Name)、状态(Status)、镜像（Image）、命令（Command）、端口号（Ports）、” IP”、“创建时间”。 可根据容器名称、容器状态和容器所在主机对容器列表进行筛选，默认显示所有容器。容器名称只能输入，容器状态主机只能选择不能输入。		
前置条件	用户已登录系统，且主机已连接		
基本事件流	1. 用户输入筛选条件，发送“显示”容器列表请求 2. 系统验证输入信息 3b. 系统验证通过 4. 系统显示容器列表信息		
异常事件流	3b. 系统验证不通过，系统返回错误提示信息。		
后置条件	系统记录操作日志		

表 3-20 容器重命名

功能名称	容器重命名（Rename）	参与者	管理员
功能描述	可以对任何状态的容器进行重命名，容器名只能由数字、字母、-、_等组成。		
前置条件	用户已登录系统，且主机已连接		
基本事件流	1. 用户选择想要重命名的容器，并输入新的容器名 2. 系统验证输入数据 3a. 系统验证通过 4. 系统显示新的容器名		
异常事件流	3b. 系统验证失败，返回错误提示信息。		
后置条件	系统记录操作日志		

### 3.5.3 镜像管理

用户可以从阿里云（或 DCloud）拉取镜像，查看所有镜像列表，也可以对想要查看的镜像进行筛选，如果镜像不再被需要，也可删除该镜像。具体功能描述见表

3-21 到表 3-23。

表 3-21 创建镜像

功能名称	创建镜像	参与者	管理员
功能描述	镜像不像容器那样在 Docker 服务中进行创建，而是从远程镜像库中拉取需要的镜像放在 Docker 服务中。拉取镜像时只需输入镜像名称即可。镜像拉取需要较长时间，需提醒用户耐心等待。		
前置条件	用户已登录系统，且主机已连接		
基本事件流	1. 用户输入镜像名称 2. 系统过滤镜像名称 3a. 系统拉取镜像成功 4. 系统显示镜像列表		
异常事件流	3b 系统拉取镜像失败，返回拉取失败原因，如：“该镜像不存在”。		
后置条件	系统记录操作日志，并显示最新镜像列表		

表 3-22 删除镜像

功能名称	删除镜像	参与者	管理员
功能描述	对于不再需要的镜像可以执行删除操作，删除操作不能批量执行，只能单个执行。		
前置条件	用户已登录系统，且主机已连接		
基本事件流	1. 用户选择待删除的镜像，发送“删除”请求 2a. 系统删除已选择的镜像		
异常事件流	2b. 系统删除系统过程中出现未知错误，系统返回提示“删除镜像失败，请重试！”		
后置条件	系统记录操作日志，并显示最新镜像列表		

表 3-23 查看镜像列表

功能名称	查看镜像列表	参与者	管理员
功能描述	可根据镜像名称筛选镜像列表，列表信息包括：镜像名（Name）、镜像 ID（ID）、镜像创建时间（Create Time）、镜像大小（Size）、镜像虚拟大小（Virtual Size）、镜像版本。		



(续表 3-23)

前置条件	用户已登录系统，且主机已连接
基本事件流	1. 用户输入镜像名称（也可不输入），发送“显示”镜像列表请求 2. 系统过滤镜像名称 3a. 系统显示镜像列表
异常事件流	3b. 系统显示过程中出现未知错误，系统返回提示“镜像列表拉取失败，请重试！”
后置条件	系统记录操作日志

### 3.5.4 系统日志

系统会记录登录用户的一切操作，生成系统日志，运行环境出现问题时，方便排查以及责任追究。对于一定日期后的日志，管理员也可进行删除。具体功能描述见表 3-24。

表 3-24 查看系统日志

功能名称	查看系统日志	参与者	管理员
功能描述	如果容器服务运行过程中出现问题，通过查看系统日志进行排查和责任追究。日志列表较多，分页显示，每页 35 条。系统日志可根据用户和时间筛选目标日志。用户名只能输入，日期范围只能选择		
前置条件	用户已登录系统，且主机已连接		
基本事件流	1. 用户输入查询条件（用户名和时间），并发送“显示”系统日志列表请求 2. 系统根据验证查询条件 3a. 系统验证通过 4. 系统分页显示系统日志		
异常事件流	3b. 查询条件验证不通过，返回查询失败错误提示信息。		
后置条件	无		

### 3.5.5 用户管理

该模块只包含用户登录功能，所需账号和密码由系统自行分配，不能通过注册实现。具体功能描述见表 3-25 和表 3-26。

表 3-25 用户登录

功能名称	用户登录	参与者	游客
功能描述	用户输入用户名和密码通过验证后，即可进入系统。一旦进入系统，便拥有所有请求操作权限。用户名和密码只能是字母、数字和下划线组成。长度限制 50。		
前置条件	用户已登录系统，且主机已连接		
基本事件流	1. 用户输入登录信息，发送“登录”请求 2. 系统验证输入信息 3a. 系统验证通过 4. 系统开启		
异常事件流	3b. 系统验证失败，返回失败提示信息，如：“用户名不存在”，“密码错误”。		
后置条件	系统记录操作日志		

表 3-26 退出系统

功能名称	退出系统	参与者	管理员
功能描述	已登录系统的用户，离开时不希望其他人使用该账户操作，可选择退出系统，以保证安全性。		
前置条件	用户已登录系统		
基本事件流	1. 用户选择退出系统 2. 系统执行退出		
后置条件	系统跳转至登录页面		

### 3.6 系统非功能性需求

#### 1、系统可靠性

系统能否正常运行取决于系统的可靠性，数据库系统使用 MySQL。MySQL 数据库是一款支持 ACID 即：事务的原子性、一致性、独立性、持久性的 InnoDB（数据库引擎）的事务性数据库，采用该数据库保证了对数据进行操作的正确安全等要求。

#### 2、系统可扩展性

系统开发中，结束后，都可能遇到需求变更或添加，如果系统拥有较强的扩展性，这些都不会出现很大的问题，或许，还能够很友好的解决。系统采用前后端完全分离的开发方式，任何一方的改变都将尽可能少的影响另外一方。系统整体采用 MVC 模式进行开发，对 Controller 和 Model 层进行了严格分类，可以很轻松地增删相应的功能点。

### 3、系统可维护性

系统开发完成后，维护成了日常主要的工作内容，所以系统的可维护性在开发时成了必不可少的要求。系统前后端的 MVC 模式可以很快定位错误出现的位置，并给出相应的解决方案。开发时必要的注释也会让代码更加容易阅读。

### 4、系统安全性

系统使用 MD5 算法对用户密码进行加密，密码只可生成和验证，不可解密，增强用户密码安全性。

系统会记录每一个用户的每一项操作日志，可以很快追溯事故原由以及相应的负责人。

### 5、系统交互性

系统前端开发要求至少兼容 IE8+，采用渐进增强的思想进行开发，即保证低浏览器能够正常显示运行的情况下，提高高端浏览器的优美视觉感。

## 3.7 本章小结

本章介绍了应用控制系统的需求分析阶段，进行了基本术语约定之后，对系统的业务需求、用户需求、功能需求以及非功能性需求进行了分析。针对用户，在功能方面满足其对主机、镜像和容器的快捷操作，在非功能方面满足可靠性、易用性和安全性等需求。

## 第4章 应用控制系统概要设计

根据系统的业务需求、用户需求、功能需求和非功能性需求进行分析，对系统做出概要设计，本章将从系统架构设计功能模块设计和数据库设计等当面进行解说。

### 4.1 系统架构设计

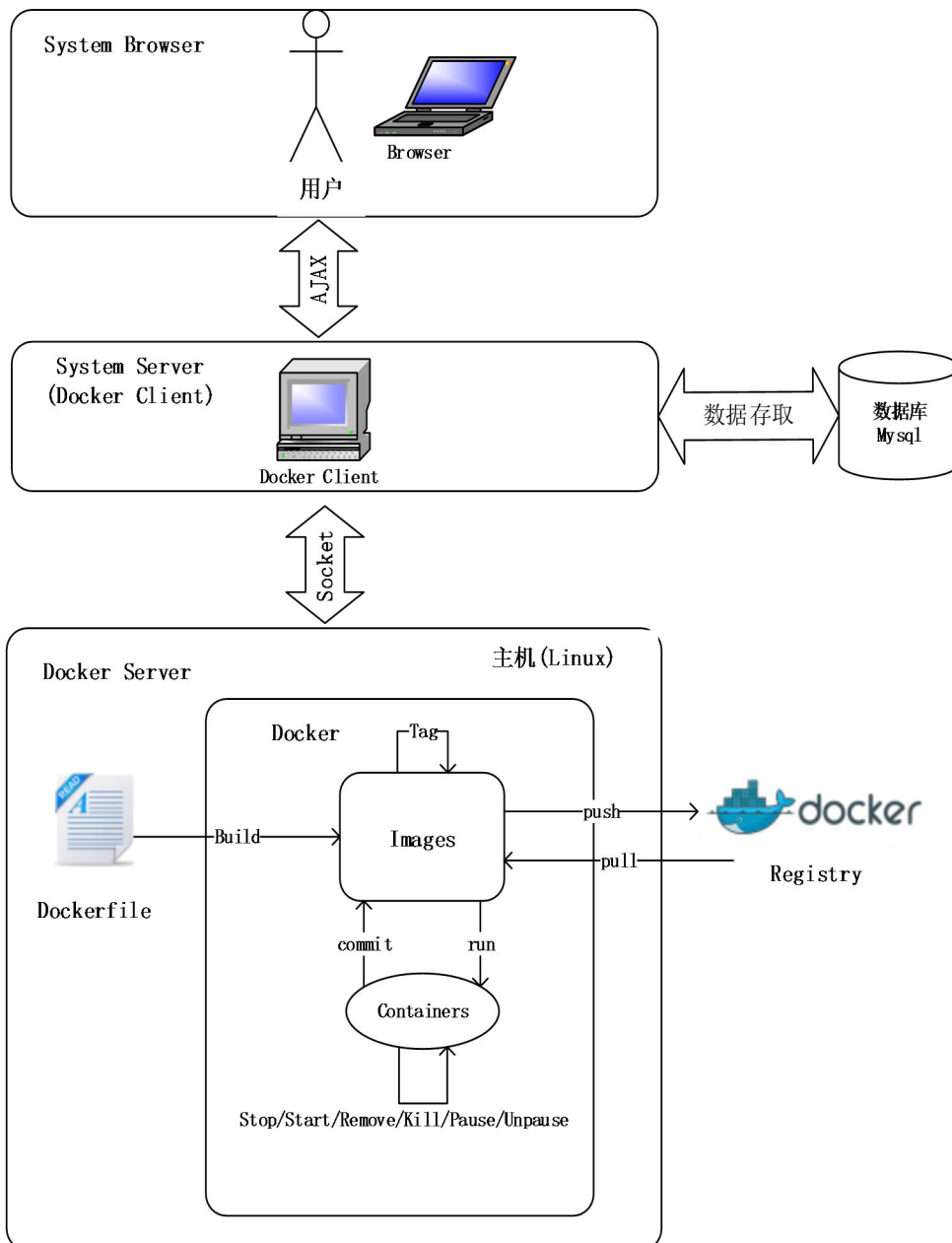


图 4-1 系统架构图

本系统主要为三种类型的人员进行设计：开发人员、测试人员、运维人员。主要

分 4 层，System Browser、System Server、Docker Server 以及 MySQL。

底层 MySQL 主要提供数据服务，使用版本号 5.6.24。

Docker Server 层是服务层，提供容器和镜像相关服务，运行在 Linux 系统中。和 System Server 之间通过 socket 进行通信，Docker Server 监听某一端口，解析 System Server 发来的请求，并按照 Docker Remote API 约定执行对应的命令。Docker 服务版本号为 1.10.3。

System Server 是主要的服务层，与上层的用户层(System Browser)通过 Ajax 直接通信，收到用户层发来的请求后，对其进行解析和处理，最终以 Docker Remote API 的形式发送请求只服务器监听的端口，交友 Docker Server 进行下一步处理。该服务层使用 PHP 作为运行脚本，Curl 编程辅助开发。

System Browser 为用户层，JS 作为运行脚本，结合 HTML、CSS 把 Docker Remote API 相关操作可视化，清晰明了地展示容器和镜像相关信息，用户的操作先经过 System Server 过滤处理，发往 Docker Server 直接执行，同时 System Server 将在数据库中记录相关操作。

## 4.2 数据库设计

### 4.2.1 数据库环境设计

数据库管理工具：Mysql5.6.24

数据库可视化工具：Navicat 10

数据库设计工具：Power Designer 16.5

### 4.2.2 命名规则

#### 1 表命名规则

数据库表的命名以都为小写且为有意义的名词单词，如 container,user。如果表名由几个单词组成，则单词间用下划线(“\_”)分割，container\_info,host\_info 等。表名尽量用全名，限制在 30 个字符内。当表的全名超过 30 字符时，可用缩写来减少表名的长度，如 description --> desc；information --> info；address --> addr 等。

#### 2 表字段命名规则

数据库表的命名以都为小写且为有意义的名词单词。如果字段由几个单词组成，则单词间用下划线(“\_”)分割，如 `container_satus`，`image_number` 等。字段名限制在 30 个字符内。当字段名超过 30 字符时，可用缩写来减少字段名的长度。

#### 4.2.3 实体

主机，是根据实际需求抽象出来的一个实体，该实体包含主机 IP、名称、端口号、状态、镜像数量、运行容器数量、停止容器数量等基本属性。主机实体如图 4-2 所示。

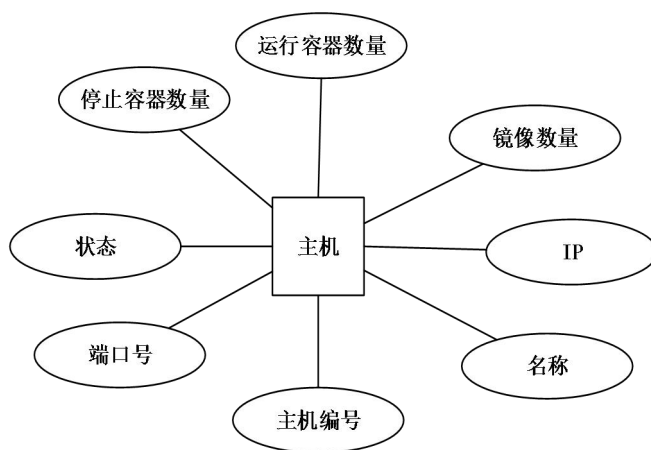


图 4-2 主机实体图

主机分组，主机需要分类展示，故产生该实体。该实体包含 2 个基本属性：分组编号和分组名称。分组实体如图 4-3 所示。



图 4-3 主机分组实体图

容器，是系统的核心部分，故抽象出一个实体。该实体主要包含如下属性：创建时间、镜像名称、容器编号、容器名称、容器 ID、容器状态、端口列表、Command。容器实体如图 4-4 所示。

镜像，容器只有在镜像的基础上才能生成，作为一个实体单独保存数据信息。镜像实体如图 4-5 所示。

用户，系统唯一的权限限制便是用户登录与否。因此作为独立的实体保存用户数

据。用户实体如图 4-6 所示。

系统日志，是用户在系统中进行一切操作的记录，作为独立实体记录数据。系统日志实体如图 4-7 所示。

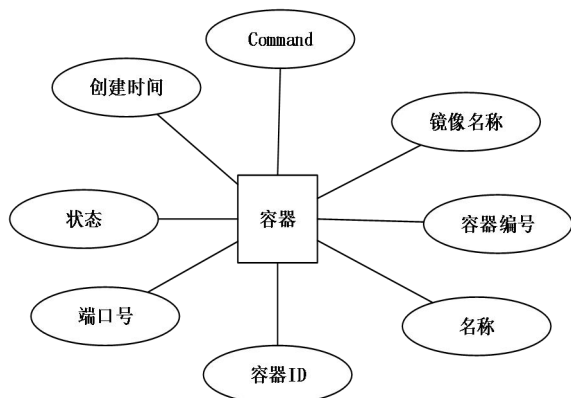


图 4-4 容器实体图

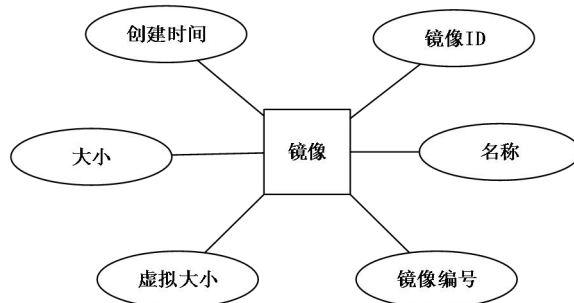


图 4-5 镜像实体图

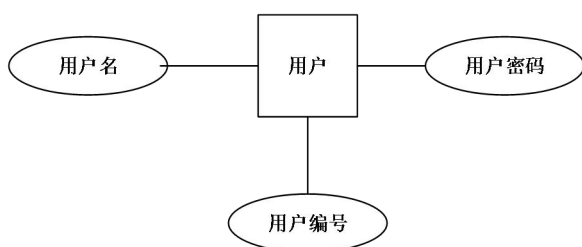


图 4-6 用户实体图

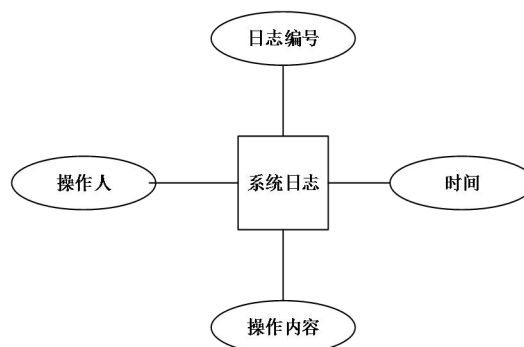


图 4-7 系统日志实体图

#### 4.2.4 实体之间的关系

主机分组与主机：主机分组与主机是一对多的关系，主机分组可以包含多个主机，一个主机只属于一个主机分组。

主机与容器：主机与容器是一对多的关系，一个主机可以包含多个容器，一个容器只属于一个主机。

主机与镜像：主机与镜像是一对多的关系，一个主机可以包含多个容器，一个镜像只属于一个主机。虽然不同的主机镜像名称一样，但是镜像的内容，如何改镜像相关的容器不一样，所以，镜像实质上是不一样的。

用户和系统日志：用户和系统日志是一对多的关系，一个用户可以有多个系统日

志，一个系统日志只属于一个用户。实体关系图如图 4-8 所示。

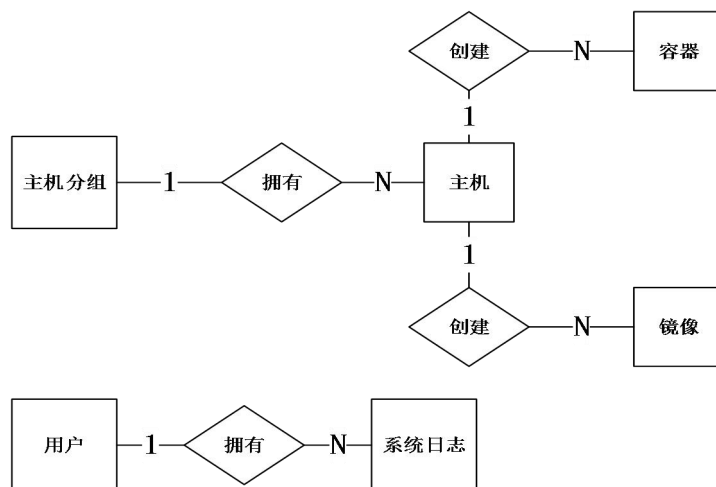


图 4-8 实体关系图

#### 4.2.5 表设计

根据上一节抽象出来的实体与实体之间的联系，设计出数据库表，本小结将介绍这些数据表的设计，如表 4-1 至表 4-6 所示。

表 4-1 container\_info 表

Field	Type	Null	Key	Comment
container_key	bigint	No	PRI	容器编号
container_id	varchar	No	-	容器 ID
container_name	varchar	No	-	容器名称
container_status	varchar	No	-	容器状态
image	varchar	No	-	镜像名称
command	varchar	No	-	命令
ports	varchar	Yes	-	端口号
created	timestamp	No	-	创建时间
host_id	int	No	FK	主机编号

表 4-2 image\_info 表

Field	Type	Null	Key	Comment
image_key	bigint	No	PRI	镜像编号



(续表 4-2)

image_id	varchar	No	-	镜像 ID
image_name	varchar	No	-	镜像名称
created_time	datetime	No	-	创建时间
image_size	float	No	-	镜像大小
image_virtual_size	float	No	-	镜像虚拟大小
host_id	int	No	FK	主机编号

表 4-3 host\_info 表

Field	Type	Null	Key	Comment
host_id	int	No	PRI	主机编号
host_ip	varchar	No	-	主机 IP
host_name	varchar	No	-	主机名称
host_port	int	No	-	主机端口号
running_container_number	varchar	No	-	运行容器
all_container_number	int	No	-	使用容器
image_number	int	No	-	镜像数量
host_group_id	int	No	FK	主机分组编号

表 4-4 host\_group 表

Field	Type	Null	Key	Comment
host_group_id	int	No	PRI	主机分组编号
host_group_name	varchar	No	-	主机分组名称

表 4-5 user\_info 表

Field	Type	Null	Key	Comment
user_id	int	No	PRI	用户编号
password	varchar	No	-	用户密码
true_name	varchar	Yes	-	真实姓名

表 4-6 system\_log 表

Field	Type	Null	Key	Comment
log_id	bigint	No	PRI	日志编号
created_time	datetime	No	-	生成时间
url	varchar	No	-	操作
user_id	int	No	FK	用户编号

### 4.3 本章小结

本章主要介绍了基于 Docker 的分布式系统的系统架构设计和数据库设计。系统架构设计主要展现系统的整个生态环境，是如何运行起来的。数据库设计介绍了实体与实体之间的关系，以及数据表的详细设计。

## 第 5 章 应用控制系统详细设计与实现

### 5.1 总体设计

#### 5.1.1 开发环境

表 5-1 开发环境

名称	环境
开发工具	PHPStorm10.0.3 + Nginx + XShell + VMWare
数据库	version 5.6.24
底层服务器 Docker	version 1.10.3, 运行 Linux 服务器系统
开发系统	windows10
开发语言	后台使用 PHP5.5, 前端使用 JS+HTML+CSS
代码版本管理	git、oschina

#### 5.1.2 系统架构

项目前后端均采用 MVC 模式的框架进行开发。

后台框架为 ThinkPHP，Model 层主要负责和数据库进行交互，从 Model 层中分离出来的 Service 层主要负责和 Docker Server 进行数据交互，由于采用前后端完全分离式开发，所以在后台项目中不使用 View 层，Control 层主要负责前端、Model 层以及 Service 层的通信。

前端框架为 Angular，Model 层，即模型层，是负责管理应用程序的数据。它响应来自视图的请求，同时也响应指令从控制器进行自我更新。View 层在一个特定的格式的演示数据，由控制器决定触发显示数据。它们是基于脚本的模板系统，如 JSP，ASP，PHP，非常容易使用 AJAX 技术的集成。控制器负责响应于用户输入并执行交互数据模型对象。控制器接收到输入，它验证输入，然后执行修改数据模型的状态的业务操作。

### 5.2 主机管理详细设计与实现

#### 5.2.1 主机管理模块设计

主机管理模块是系统的主要模块之一，用于对主机群进行管理，包括主机的增删

改查操作和主机分组的增删改查操作。本小结介绍该模块的详细设计与实现。主机基本信息包括主机编号、主机 IP、主机名称、主机端口号、主机所有容器数量、主机运行中容器数量、主机镜像数量和主机所在分组。模块界面设计如下图 5-1 所示。

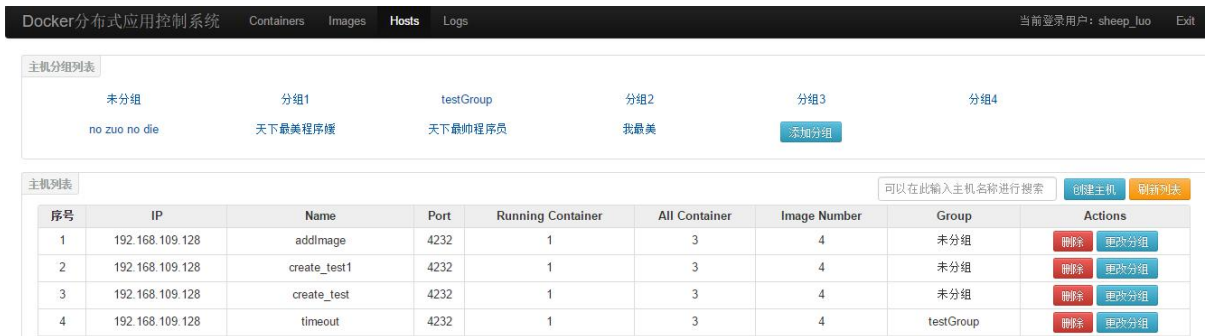


图 5-1 主机管理界面

### 5.2.2 创建主机

创建主机后台编写除了前端传来的基本数据，还需要从 Docker Server 获取更多的详细数据，容器和镜像的数量，同时检测该主机 IP 是否正确，可以正常连接主机并获取到容器和镜像数量后，和前端传过来的主机 IP、名称、端口号一同存进数据库。详细设计如表 5-2 所示。

表 5-2 创建主机详细设计

方法名称	创建主机	方法属性	公有
Url	Home/Host/createHost		
参数	<p>hostIp String 合法 IP</p> <p>hostName String 主机名称只能包含数字字母和下划线，长度为 3 到 50!</p> <p>hostPort Int 端口号范围为 0 ~ 65535</p> <p>groupId Int 主机所在分组编号</p>		
返回值	<p>success Bool 请求成功或失败</p> <p>error String 请求失败的原因 请求失败时，返回该参数</p> <p>hostId int 主机编号 主机创建成功后，返回 hostId</p>		
特殊说明	all_container_num 和 running_container_num 根据主机地址直接从 Docker Server 中获取到之后再和其它参数一起存进数据库		

(续表 5-2)

流程图	
关键代码	<pre>\$data['all_container_number'] = count(\$linkBool); \$data['running_container_number'] = count(\$DockerSDK-&gt;getContainers(0)); \$data['image_number'] = count(json_decode(\$DockerSDK-&gt;listImages())); \$HostInfoModel = D('HostInfo'); \$createBool = \$HostInfoModel-&gt;create(); \$hostId = \$HostInfoModel-&gt;add();</pre>

创建主机界面设计如图 5-2 所示。

添加主机

Host Ip:

请输入主机IP

Host Name:

请输入主机名称

Port:

请输入端口号

Group:

---请选择---

关闭

添加

图 5-2 创建主机界面

### 5.2.3 删除主机

对于不必要的主机可以进行删除，删除的时候将同时删除该主机包含的镜像和容器。详细设计如表 5-3 所示。

表 5-3 删除主机详细设计

方法名称	删除主机	方法属性	公有
Url	Home/Host/deleteHost		
参数	hostId Int 主机编号		
返回值	success Bool 删除成功或失败 error String 删除失败的原因，删除失败时将返回该字段		
特殊说明	删除顺序必须是：删除容器->删除镜像->删除主机，先删除镜像可能会因为该镜像运行着某个容器导致删除失败，如果主机包含镜像和容器则无法进行删除操作，因为他们之间存在着外键关系。		
流程图	<pre> graph TD     Start([开始]) --&gt; Filter[过滤参数]     Filter --&gt; ParamCheck{参数是否符合要求?}     ParamCheck -- 否 --&gt; Fail1([返回删除失败, 及失败原因])     ParamCheck -- 是 --&gt; DelContainer[删除主机包含的容器]     DelContainer --&gt; DelContainerCheck{删除成功?}     DelContainerCheck -- 否 --&gt; Fail2([返回删除失败, 及失败原因])     DelContainerCheck -- 是 --&gt; DelImage1[删除主机包含的镜像]     DelImage1 --&gt; DelImage1Check{删除成功?}     DelImage1Check -- 否 --&gt; Fail3([返回删除失败, 及失败原因])     DelImage1Check -- 是 --&gt; DelImage2[删除主机包含的镜像]     DelImage2 --&gt; DelImage2Check{删除成功?}     DelImage2Check -- 否 --&gt; Fail4([返回删除失败, 及失败原因])     DelImage2Check -- 是 --&gt; Success([返回删除成功])           </pre>		
关键代码	<pre> //删除主机包含的容器 \$deleteBool = \$containerInfoModel-&gt;where(['host_id' =&gt; \$hostId])-&gt;delete(); //删除主机包含的镜像 \$deleteBool = \$HostInfo-&gt;delete(\$hostId);           </pre>		

## 5.2.4 查询主机

主机列表查询分两种，一种是直接从数据库读取缓存数据，一种是根据现有主机从 Docker Server 中读取数据，并同时同步数据至数据库。详细设计如表 5-4 和表 5-5 所示。

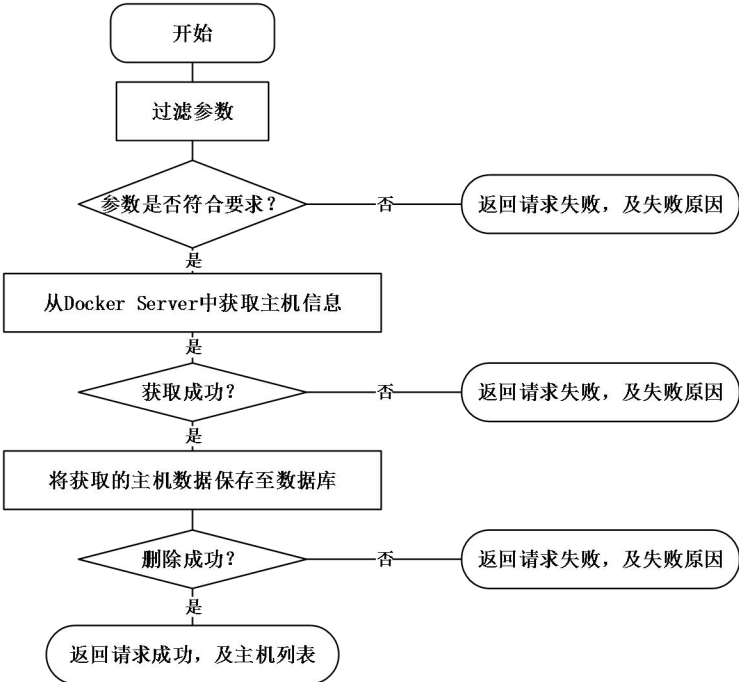
表 5-4 查询主机详细设计

方法名称	查询主机	方法属性	公有
Url	Home/Host/searchHostList		
参数	无		
返回值	success Bool 请求成功或失败 error String 请求失败的原因，请求失败时返回该数据 hostList Array 主机列表，请求成功时会返回该数据		
特殊说明	数据库查询		
流程图	<pre> graph TD     Start([开始]) --&gt; Query[查询主机列表]     Query --&gt; Decision{查询成功?}     Decision -- 是 --&gt; Success([返回请求成功，及主机列表])     Decision -- 否 --&gt; Failure([返回请求失败，及失败原因])           </pre>		
关键代码	<pre> \$HostListInfo = new HostListViewModel(); //D('HostList'); \$hostListData = \$HostListInfo-&gt;order('host_id desc')-&gt;select();           </pre>		

表 5-5 刷新主机详细设计

方法名称	刷新主机列表	方法属性	公有
Url	Home/Host/refreshHostList		
参数	hostList Array 待更新数据的主机列表		
返回值	success Bool 请求成功或失败 error String 请求失败的原因，请求失败时返回该数据 hostList Array 主机列表，请求成功时会返回该数据		
特殊说明	从 Docker Server 中查询并更新数据库中的主机相关信息		

(续表 5-5)

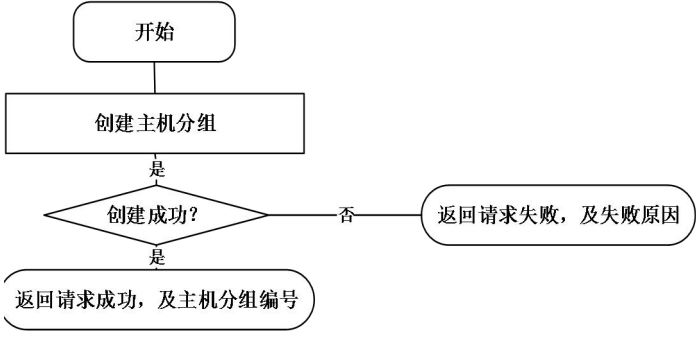
流程图	 <pre> graph TD     Start([开始]) --&gt; Filter[过滤参数]     Filter --&gt; ParamCheck{参数是否符合要求?}     ParamCheck -- 否 --&gt; Fail1([返回请求失败, 及失败原因])     ParamCheck -- 是 --&gt; GetInfo[从Docker Server中获取主机信息]     GetInfo --&gt; GetSuccess{获取成功?}     GetSuccess -- 否 --&gt; Fail2([返回请求失败, 及失败原因])     GetSuccess -- 是 --&gt; SaveDB[将获取的主机数据保存至数据库]     SaveDB --&gt; DeleteSuccess{删除成功?}     DeleteSuccess -- 否 --&gt; Fail3([返回请求失败, 及失败原因])     DeleteSuccess -- 是 --&gt; ReturnSuccess([返回请求成功, 及主机列表]) </pre>
关键代码	<pre> foreach(\$hostList as \$host) {     \$hostAddress = \$host-&gt;hostIp.'.'.\$host-&gt;hostPort;     \$DockerSDK = new DockerSDKService(\$hostAddress);     \$linkBool = \$DockerSDK-&gt;getContainers(1);     if(\$linkBool == null) {         continue;     }     \$data['all_container_number'] = count(\$linkBool);     \$data['running_container_number'] = count(\$DockerSDK-&gt;getContainers(0));     \$data['image_number'] = count(json_decode(\$DockerSDK-&gt;listImages()));     \$HostInfoModel = D('HostInfo');     \$HostInfoModel-&gt;where(["host_id" =&gt; \$host-&gt;hostId])-&gt;save(\$data); }; \$hostList = (new HostListViewModel())-&gt;order('host_id desc')-&gt;select(); </pre>

### 5.2.5 创建主机分组

主机分组有一个默认值为“未分组”，该分组不能创建不能删除。详细设计如表 5-6 所示。



表 5-6 创建主机详细设计

方法名称	创建主机分组	方法属性	公有
Url	Home/Host/createHostGroup		
参数	groupName String 分组名称		
返回值	groupId int 分组编号		
特殊说明	无		
流程图	 <pre>graph TD; Start([开始]) --&gt; Create[创建主机分组]; Create --&gt; Success{创建成功?}; Success -- 是 --&gt; ReturnSuccess([返回请求成功, 及主机分组编号]); Success -- 否 --&gt; ReturnFailure([返回请求失败, 及失败原因]);</pre>		
关键代码	<pre>\$HostGroup = D('HostGroup'); \$groupId = \$HostGroup-&gt;add(['host_group_name' =&gt; \$groupData]);</pre>		

创建主机分组如图 5-3 所示。

添加主机分组

分组名称:

请输入分组名称

关闭

添加

图 5-3 创建主机分组界面

### 5.2.6 删除主机分组

由于主机和主机分组存在主外键关系, 所以删除主机分组前, 需要把该分组包含的主机所在分组置为“未分组”, 再进行删除。“未分组”一栏不允许删除。详细设计如表 5-7 所示。

表 5-7 删除主机分组详细设计

方法名称	删除主机分组	方法属性	公有
Url	Home/Host/deleteHostGroup		
参数	groupId 分组编号		
返回值	success Bool 删除成功或失败 error String 删除失败的原因，删除失败时返回该值		
特殊说明	无		
流程图	<pre> graph TD     Start([开始]) --&gt; Filter[过滤参数]     Filter --&gt; ParamCheck{参数是否符合要求?}     ParamCheck -- 否 --&gt; Fail1([返回请求失败，及失败原因])     ParamCheck -- 是 --&gt; SetGroup[将主机所在分组设置为“未分组”]     SetGroup --&gt; SetSuccess{设置成功?}     SetSuccess -- 否 --&gt; Fail2([返回请求失败，及失败原因])     SetSuccess -- 是 --&gt; DeleteHost[删除主机]     DeleteHost --&gt; DeleteSuccess{删除成功?}     DeleteSuccess -- 否 --&gt; Fail3([返回请求失败，及失败原因])     DeleteSuccess -- 是 --&gt; Success([返回请求成功])           </pre>		
关键代码	<pre> //删除主机分组时，将该组所在的主机分组置为“未分组”，“未分组”对应 //分组编号为1 \$HostModel = D('HostInfo'); \$saveData['host_group_id'] = 1; \$searchData['host_group_id'] = \$groupId; \$HostModel-&gt;where(\$searchData)-&gt;save(\$saveData); //删除分组 \$HostGroupModel = D('HostGroup'); \$deleteBool = \$HostGroupModel-&gt;delete(\$groupId);           </pre>		

### 5.2.7 查询主机分组

查询结果信息包括分组名称和分组编号。详细设计如表 5-8 所示。

表 5-8 查询主机分组详细设计

方法名称	查询主机分组	方法属性	公有
Url	Home/Host/searchHostGroupLis		
参数	无		
返回值	success Bool 请求失败或成功 error String 请求失败的原因 groupList Array 主机分组列表，请求成功时返回		
特殊说明	无		
流程图	 <pre> graph TD     Start([开始]) --&gt; Query[查询主机分组]     Query --&gt; Decision{查询成功?}     Decision -- 是 --&gt; Success([返回请求成功，及主机分组列表])     Decision -- 否 --&gt; Failure([返回请求失败，及失败原因])           </pre>		
关键代码	<pre> \$HostGroup = D('HostGroup');; \$hostGroupData = \$HostGroup-&gt;order('host_group_id')-&gt;select();           </pre>		

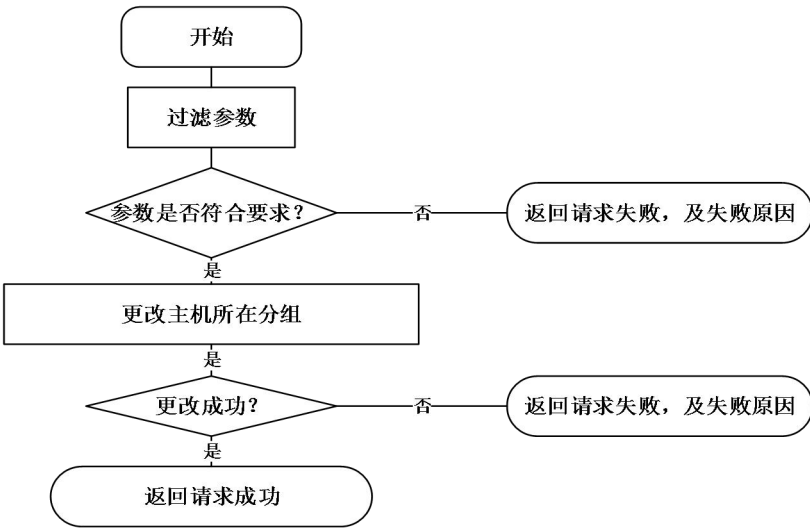
### 5.2.8 更改主机所在分组

主机所在分组并非一成不变，可以根据需要随时设置主机所在分组。详细设计如表 5-9 所示。

表 5-9 更改主机分组详细设计

方法名称	更改主机所在分组	方法属性	公有
Url	Home/Host/changeHostGroup		
参数	hostId int 主机编号 groupId int 主机分组编号		
返回值	success Bool 请求失败或成功 error Strig 请求失败的原因		
特殊说明	无		

(续表 5-9)

流程图	
关键代码	<pre>\$HostModel = D('HostInfo'); \$saveData['host_group_id'] = \$groupId; \$searchData['host_id'] = \$hostId; \$saveBool = \$HostModel-&gt;where(\$searchData)-&gt;save(\$saveData);</pre>

更改主机所在分组界面设计如图 5-4 所示。

### 更改主机分组

请选择分组：

分组2

关闭

更改

图 5-4 更改主机所在分组界面

## 5.3 容器管理详细设计与实现

### 5.3.1 容器管理模块设计

容器管理模块是系统的主要模块之一，用于对容器的增删改查、启动、停止、终止、暂停等操作。本小结介绍该模块的详细设计与实现。容器基本信息包括：容器编号、容器 ID、容器名称、容器状态、命令、端口号以及所在的主机。容器编号为数据库中的主键，容器 ID 为 Docker Server 中的容器唯一标识。模块界面设计如下图 5-5 所示。

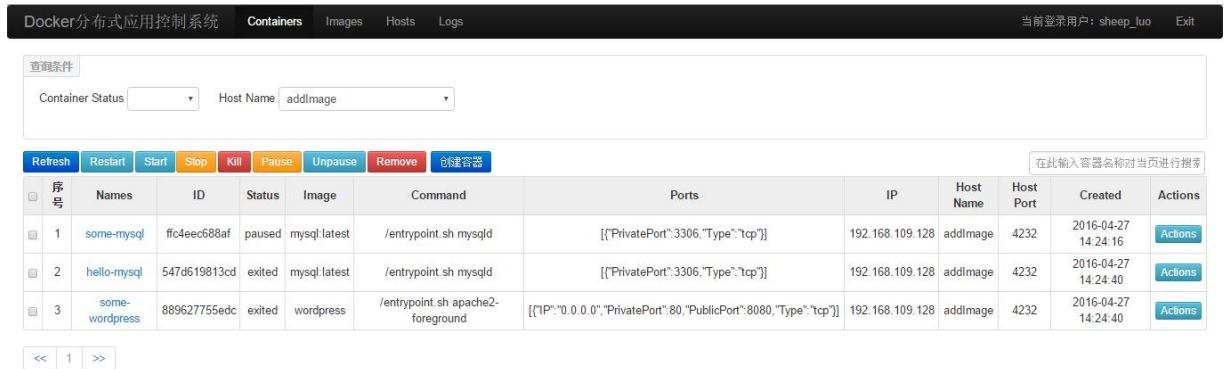


图 5-5 容器管理

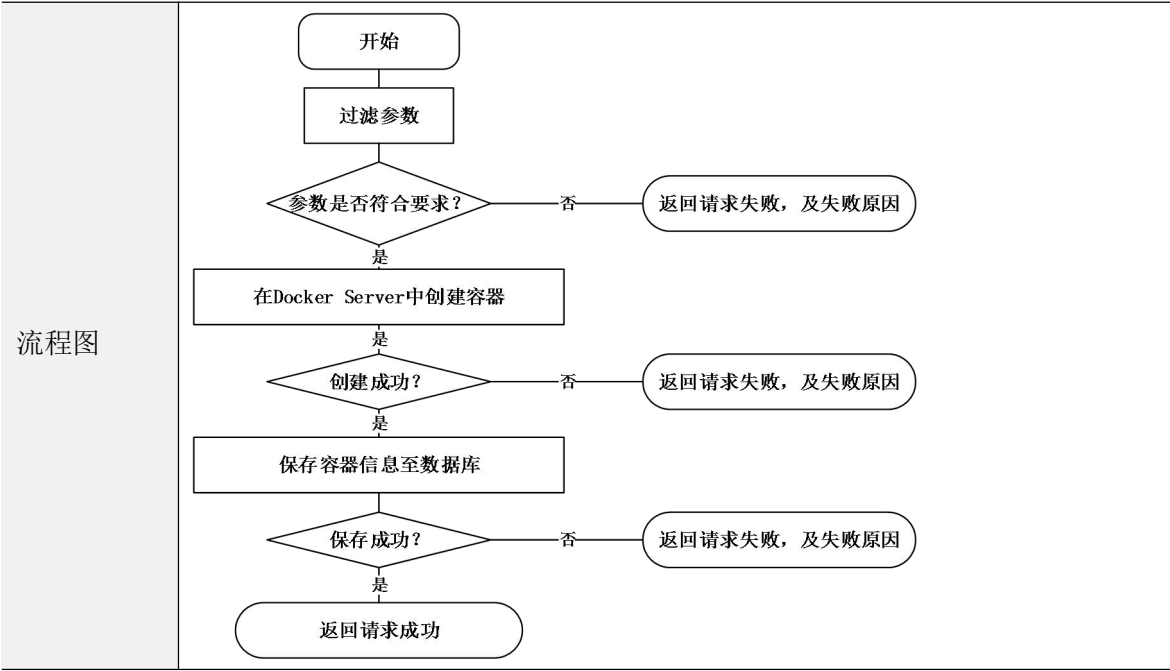
### 5.3.2 创建容器

容器创建需要的参数非常多，此处创建只输入重要参数即可。容器创建成功后，主机列表中的所有容器数量加 1。点击图 5-2 中“创建容器”按钮，即可进入创建容器界面。详细设计如表 5-10 所示。

表 5-10 创建容器详细设计

方法名称	创建容器	方法属性	公有
Url	Home/Container/createContainer		
参数	imageName Must 镜像名称 hostInfo Must 主机信息，包括主机 IP、编号、ID 用冒号隔开 containerConf Must 容器参数配置		
返回值	success Bool 请求成功或失败 error String 请求失败的原因		
特殊说明	containerConf 中包含了所有容器配置		
关键代码	<pre>\$DockerSDK = new DockerSDKService(\$hostAddress); \$containerInfo = \$DockerSDK-&gt;createContainer(\$imageName, \$containerConf); //向 Docker 服务发送 get 请求 container 信息，并将信息存入数据库 \$data = \$DockerSDK-&gt;inspectContainer(\$containerInfo['Id']); ..... \$ContainerInfoModel-&gt;data(\$saveData)-&gt;add(); ..... \$hostInfoModel = D('HostInfo'); \$hostInfoModel-&gt;where(['host_id' =&gt; \$hostId])-&gt;setInc('all_container_number'); \$hostInfoModel-&gt;where(['host_id' =&gt; \$hostId])-&gt;setInc('running_container_number');</pre>		

(表 5-10)



创建容器界面设计如图 5-6 所示。

创建容器

Base Config

Host Address 192.168.109.128:4232

Image Name

Hostname

Domain Name

Command

Memory

Container Name

Create Container

More Config

Network Mode

Container DNS

Environment Variables

Environment Variables Name	Environment Variables Value	添加
----------------------------	-----------------------------	----

Volumes

Host Path	Container Path	添加
-----------	----------------	----

Container Links

Container Name	Container Alias	添加
----------------	-----------------	----

Exposed Ports

Container Port	请选择 Protocol	Host Listen Address	Host Port	添加
----------------	--------------	---------------------	-----------	----

图 5-6 创建容器界面

5.3.3 操作容器

容器操作包括：启动容器、重启容器、移除容器、暂停容器、恢复暂停容器、停止容器、终止容器。从 Docker Server 中对容器实施操作后，更改数据库相应的容器状态，以及主机中所有容器数量和运行容器数量。这些操作均支持容器批量操作。详细设计如表 5-11 所示。

表 5-11 操作容器详细设计

方法名称	操作容器	方法属性	公有
Url	Home/Container/operateContainer		
参数	<p>containerId String DockerServer 中的容器 ID，单个操作必须</p> <p>containerKey int 数据库中的容器主键，单个操作必须</p> <p>containerStatus String 容器当前状态，单个操作必须</p> <p>hostInfo String 主机信息，单个操作必须，包括主机 IP、端口号、ID</p> <p>containers String 批量操作容器，批量操作必须</p> <p>operation String 容器操作，都必须，包括 start/ restart/ kill/ pause/ unpause/ remove/stop，通过 operation 动态确定具体操作</p>		
返回值	<p>success Bool 请求成功或失败</p> <p>error String 请求失败的原因</p> <p>status String 容器当前状态</p>		
特殊说明	<p>删除容器：所有容器数量减 1，若容器本身处于运行状态，运行容器数量也减 1；</p> <p>其它操作：容器状态从” running” 变为其它，运行容器数量减 1，若容器状态从其它变为” running”，运行容器数量加 1；</p>		
流程图	<pre> graph TD     Start([开始]) --&gt; Filter[过滤参数]     Filter --&gt; ParamCheck{参数是否符合要求?}     ParamCheck -- 否 --&gt; Fail1([返回请求失败，及失败原因])     ParamCheck -- 是 --&gt; OperateCheck{operate是移除容器?}     OperateCheck -- 是 --&gt; Remove[在Docker Server中移除容器]     OperateCheck -- 否 --&gt; Operate[操作容器]     Remove --&gt; RemoveSuccess{移除成功?}     RemoveSuccess -- 是 --&gt; RunningCheck1{容器原来状态为running?}     RunningCheck1 -- 是 --&gt; ReduceTotalRunning1[主机中容器总数量和运行容器数量减1]     RunningCheck1 -- 否 --&gt; ReduceTotalRunning2[主机中运行容器总数量减1]     Operate --&gt; OperateSuccess{操作成功?}     OperateSuccess -- 是 --&gt; RunningCheck2{容器状态运行变为非运行状态?}     RunningCheck2 -- 是 --&gt; ReduceTotalRunning2     RunningCheck2 -- 否 --&gt; NonRunningCheck{容器状态非运行变为运行状态?}     NonRunningCheck -- 是 --&gt; IncreaseTotalRunning[主机中运行容器总数量加1]     ReduceTotalRunning1 --&gt; FinalSuccessCheck{操作成功?}     ReduceTotalRunning2 --&gt; FinalSuccessCheck     IncreaseTotalRunning --&gt; FinalSuccessCheck     FinalSuccessCheck -- 是 --&gt; Success([返回请求成功])     FinalSuccessCheck -- 否 --&gt; Fail2([返回请求失败，及失败原因])     Fail1 --&gt; Fail2   </pre>		
关键代码	<pre> \$containerDetail = \$DockerSDK-&gt;inspectContainer(\$Options['containerId']); \$ContainerModel-&gt;where(['container_key' =&gt; \$Options['containerKey']]) -&gt; save(\$saveData); </pre>		

### 5.3.4 查询容器日志

容器进程随时有变，且变化较大，故不保存至数据库，直接从 Docker Server 中进行查询。详细设计如表 5-12 所示。

表 5-12 查询容器日志详细设计

方法名称	查询容器日志	方法属性	公有
Url	Home/Container/getContainerLogs		
参数	containerId String 容器编号 hostAddress String 主机地址包括 ip 和端口号，如：192.168.109.128:4232		
返回值	success Bool 请求成功或失败 error String 请求失败的原因 containerLogs String 容器日志		
特殊说明	无		
流程图			
关键代码	<pre>\$DockerSDK = new DockerSDKService(\$hostAddress); \$containerLogs = \$DockerSDK-&gt;logsContainer(\$containerId); //转码后，前端才能获取到数据 \$containerLogs = mb_convert_encoding(\$containerLogs,'utf-8','auto');</pre>		

查询容器日志界面设计如图 5-7 所示。

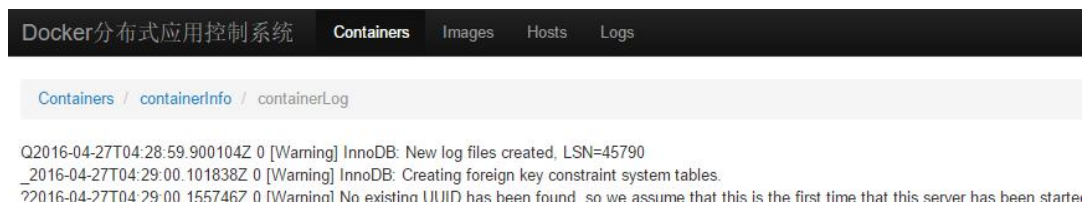


图 5-7 查询容器日志界面



### 5.3.5 查询容器进程

容器进程随时有变，且变化较大，故不保存至数据库，直接从 Docker Server 中进行查询。详细设计如表 5-13 所示。

表 5-13 查询容器进程详细设计

方法名称	查询容器进程	方法属性	公有
Url	Home/Container/getContainerProcess		
参数	containerId String 容器编号 hostAddress String 主机地址包括 ip 和端口号，如：192.168.109.128:4232		
返回值	success Bool 请求成功或失败 error String 请求失败的原因 processList Array 进程信息		
特殊说明	无		
流程图	<pre> graph TD     Start([开始]) --&gt; Filter[过滤参数]     Filter --&gt; ParamCheck{参数是否符合要求?}     ParamCheck -- 否 --&gt; Fail1([返回请求失败, 及失败原因])     ParamCheck -- 是 --&gt; Query[从Docker Server查询进程信息]     Query --&gt; SuccessCheck{查询成功?}     SuccessCheck -- 否 --&gt; Fail2([返回请求失败, 及失败原因])     SuccessCheck -- 是 --&gt; Success([返回请求成功, 及进程信息])           </pre>		
关键代码	<pre> \$DockerSDK = new DockerSDKService(\$hostAddress); \$processList = \$DockerSDK-&gt;getContainerProcess(\$containerId);           </pre>		

查询容器进程界面设计如图 5-8 所示。



图 5-8 查询容器进程界面

## 5.3.6 查询容器列表

查询容器列表分 2 种，一种是从 Docker Server 中查询后，更新数据数据，再从数据库中进行查询，另一种是直接从数据库中进行查询。详细设计如表 5-14 所示。

表 5-14 查询容器列表详细设计

方法名称	查询容器列表	方法属性	公有
Url	Home/Container/searchContainerList		
参数	<p>defaultData int 值为 1，需要从 Docker Server 查询数据，值为 0 直接从数据库查询，默认为 0</p> <p>containerStatus String 容器状态</p> <p>hostInfo String 主机信息，包括主机 IP，端口号，和主机 ID，用冒号隔开</p> <p>pageSize int 分页大小</p> <p>nowPage int 当前页</p>		
返回值	<p>success Bool 请求成功或失败</p> <p>error String 请求失败原因</p> <p>containerList Array 容器列表</p>		
特殊说明	查询方式根据 defaultData 决定		
流程图	<pre> graph TD     Start([开始]) --&gt; Filter[过滤参数]     Filter --&gt; ParamCheck{参数是否符合要求?}     ParamCheck -- 否 --&gt; Fail1([返回请求失败，及失败原因])     ParamCheck -- 是 --&gt; DefaultCheck{defaultData=0?}     DefaultCheck -- 否 --&gt; Docker[从Docker Server获取容器列表]     Docker --&gt; DockerSuccess{获取成功?}     DockerSuccess -- 否 --&gt; Fail2([返回请求失败，及失败原因])     DockerSuccess -- 是 --&gt; SaveDB[保存容器列表至数据库]     SaveDB --&gt; SaveSuccess{保存成功?}     SaveSuccess -- 否 --&gt; Fail3([返回请求失败，及失败原因])     SaveSuccess -- 是 --&gt; QueryDB[根据条件从数据库查询容器列表]     QueryDB --&gt; QuerySuccess{查询成功?}     QuerySuccess -- 否 --&gt; Fail4([返回请求失败，及失败原因])     QuerySuccess -- 是 --&gt; End([返回请求成功，及容器列表]) </pre>		

(续表 5-14)

关键代码	<pre> if(\$defaultData == 1) {     //从 Docker Server 获取容器列表     \$DockerSDK = new DockerSDKService(\$hostAddress);     \$containerList = \$DockerSDK-&gt;getContainers(1); } ..... \$containerList = \$ContainerInfoModel-&gt;where(\$searchData)-&gt;order('container_key desc')-&gt;page(\$nowPage, \$pageSize)-&gt;select(); \$count = \$ContainerInfoModel-&gt;where(\$searchData)-&gt;count();// 查询满足要求的 总记录数 </pre>
------	---

### 5.3.7 查询容器详细信息

容器属性非常多，不便于存在数据库中，故此处直接从 Docker Server 中获取。详细设计如表 5-15 所示。

表 5-15 查询容器详细信息详细设计

方法名称	查询容器详细信息	方法属性	公有
Url	Home/Container/getContainerById		
参数	containerId String 容器在 Docker Server 中的唯一编号 hostAddress String 主机地址包括 ip 和端口号，如：192.168.109.128:4232		
返回值	success Bool 请求成功或失败 error String 请求失败原因 containerDetail String 容器详细信息		
特殊说明	无		
流程图	<pre> graph TD     Start([开始]) --&gt; Filter[过滤参数]     Filter --&gt; ParamCheck{参数是否符合要求?}     ParamCheck -- 否 --&gt; Fail1([返回请求失败, 及失败原因])     ParamCheck -- 是 --&gt; Query[从Docker Server中查询容器详细信息]     Query --&gt; SuccessCheck{查询成功?}     SuccessCheck -- 否 --&gt; Fail2([返回请求失败, 及失败原因])     SuccessCheck -- 是 --&gt; Success([返回请求成功, 及容器详细信息]) </pre>		
关键代码	<pre> \$DockerSDK = new DockerSDKService(\$hostAddress); \$containerDetail = \$DockerSDK-&gt;inspectContainer(\$containerId); </pre>		

查询容器详细信息界面设计如图 5-9 所示。

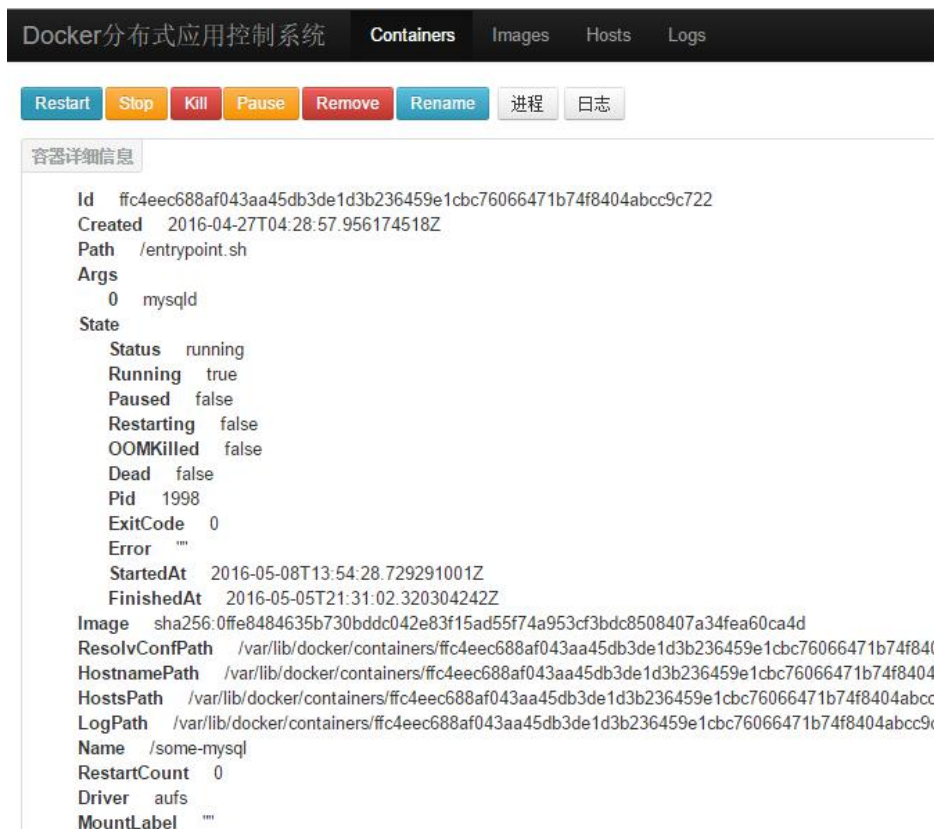


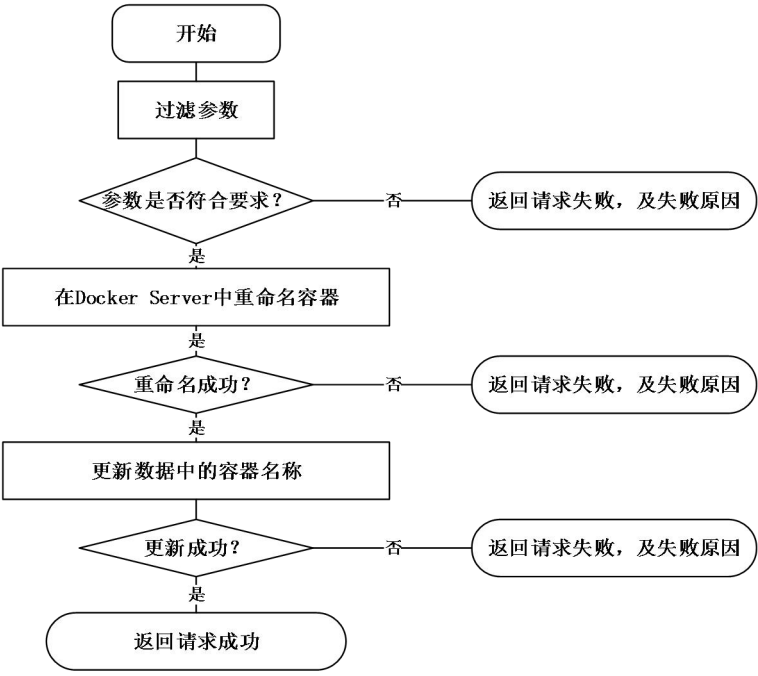
图 5-9 查询容器信息界面

### 5.3.8 重命名容器

任何状态的容器都可以进行容器重命名,但是新名称不得与现有的任何容器名称相同,否则将重命名失败。详细设计如表 5-16 所示。

表 5-16 重命名容器详细设计

方法名称	重命名容器	方法属性	公有
Url	Home/Container/renameContainer		
参数	containerId String DockerServer 中的容器 ID, 单个操作必须 containerKey int 数据库中的容器主键, 单个操作必须 hostAddress String 主机地址包括 ip 和端口号, 如: 192.168.109.128:4232 newName String 容器新名称		
返回值	success Bool 请求成功或失败 error String 请求失败原因		
特殊说明	无		

流程图	 <pre> graph TD     Start([开始]) --&gt; Filter[过滤参数]     Filter --&gt; Check1{参数是否符合要求?}     Check1 -- 否 --&gt; Fail1([返回请求失败, 及失败原因])     Check1 -- 是 --&gt; Rename[在Docker Server中重命名容器]     Rename --&gt; Check2{重命名成功?}     Check2 -- 否 --&gt; Fail2([返回请求失败, 及失败原因])     Check2 -- 是 --&gt; Update[更新数据中的容器名称]     Update --&gt; Check3{更新成功?}     Check3 -- 否 --&gt; Fail3([返回请求失败, 及失败原因])     Check3 -- 是 --&gt; Success([返回请求成功]) </pre>
关键代码	<pre>//更改 Docker Server 数据库中的容器名称 \$renameBool = \$DockerSDK-&gt;renameContainer(\$containerId, \$newName); \$ContainerModel-&gt;where(array('container_id' =&gt; \$containerKey)) -&gt; save(\$saveData);</pre>

## 5.4 镜像管理详细设计与实现

### 5.4.1 镜像管理模块设计

镜像管理模块是系统的主要模块之一，用于对镜像拉取、删除和查询。本小结介绍该模块的详细设计与实现。镜像的基本信息包括：镜像名称、Size、Virtual Size、创建时间、镜像 ID，主机信息等。模块界面设计如下图 5-10。

Docker分布式应用控制系统 Containers Images Hosts Logs 当前登录用户: sheep_luo Edit								
Pull Image Refresh		addImage		在此输入镜像名称可进行搜索				
序号	Image Name	Size	Virtual Size	Created	ID	hostName	hostIp	Action
1	alpine:latest	4798060	4798060	2016-04-02 04:53:00	d7a513a663c1	addImage	192.168.109.128:4232	删除
2	wordpress:latest	516607000	516607000	2016-03-09 12:09:37	e146b48c1da2	addImage	192.168.109.128:4232	删除
3	ubuntu:latest	187960000	187960000	2016-03-04 05:38:53	07c86167cdc4	addImage	192.168.109.128:4232	删除
4	mysql:latest	361272000	361272000	2016-03-02 18:49:50	0fe8484635b	addImage	192.168.109.128:4232	删除

图 5-10 镜像管理

### 5.4.2 创建镜像

创建镜像，也叫拉取镜像。镜像拉取成功后，主机中镜像数量加 1。详细设计如

表 5-17 所示。

表 5-17 创建镜像详细设计

方法名称	创建镜像	方法属性	公有
Url	Home/Image/createImage		
参数	imageName String 镜像名称 hostInfo String 主机信息，包括主机、端口号、IP		
返回值	success Bool 请求成功或失败 error String 请求失败的原因 imageInfo Object 镜像信息		
特殊说明	无		
流程图	<pre> graph TD     Start([开始]) --&gt; Filter[过滤参数]     Filter --&gt; ParamCheck{参数是否符合要求?}     ParamCheck -- 否 --&gt; Fail1([返回请求失败, 及失败原因])     ParamCheck -- 是 --&gt; Pull[从远程仓库拉取镜像]     Pull --&gt; PullSuccess{拉取成功?}     PullSuccess -- 否 --&gt; Fail2([返回请求失败, 及失败原因])     PullSuccess -- 是 --&gt; GetInfo[从Docker Server获取镜像信息]     GetInfo --&gt; GetSuccess{获取成功?}     GetSuccess -- 否 --&gt; Fail3([返回请求失败, 及失败原因])     GetSuccess -- 是 --&gt; SaveDB[将镜像信息保存至数据库]     SaveDB --&gt; SaveSuccess{删保存功?}     SaveSuccess -- 否 --&gt; Fail4([返回请求失败, 及失败原因])     SaveSuccess -- 是 --&gt; ReturnSuccess([返回请求成功, 镜像信息])           </pre>		
关键代码	<pre> //在 Docker Server 中创建镜像 \$DockerSDK = new DockerSDKService(\$hostAddress); \$createInfo = \$DockerSDK-&gt;createImage(\$imageName); //获取镜像信息 \$imageInfo = \$DockerSDK-&gt;inspectImage(\$imageName); //将信息保存至数据库 \$imageModel-&gt;field('image_id,image_name,created_time,image_tag,image_size, image_virtual_size,host_id')-&gt;add(\$addData);           </pre>		

创建镜像界面设计如图 5-11 所示。

图 5-11 镜像管理

### 5.4.3 查询镜像

镜像查询包括数据库查询和 Docker Server 查询两种，defaultData 不同，查询方式不同，详细设计如表 5-18 所示。

表 5-18 查询镜像详细设计

方法名称	查询镜像	方法属性	公有
Url	Home/Image/searchImagesList		
参数	defaultData int 值为 0 时，从数据库直接查询，值为 1 时，从 Docker Server 查询 hostInfo String 包括主机 ip、端口号、ID，用冒号隔开		
返回值	success Bool 请求成功或失败 error String 请求失败的原因 imageList Array 镜像列表信息		
特殊说明	无		
流程图	<pre> graph TD     Start([开始]) --&gt; Filter[过滤参数]     Filter --&gt; ParamCheck{参数是否符合要求?}     ParamCheck -- 否 --&gt; Fail1([返回请求失败，及失败原因])     ParamCheck -- 是 --&gt; DefaultCheck{defaultData=0?}     DefaultCheck -- 否 --&gt; Docker[从Docker Server获取镜像列表]     DefaultCheck -- 是 --&gt; DBQuery[根据条件从数据库查询镜像列表]     Docker --&gt; DockerSuccess{获取成功?}     DockerSuccess -- 否 --&gt; Fail2([返回请求失败，及失败原因])     DockerSuccess -- 是 --&gt; SaveDB[保存镜像列表至数据库]     SaveDB --&gt; SaveSuccess{保存成功?}     SaveSuccess -- 否 --&gt; Fail3([返回请求失败，及失败原因])     SaveSuccess -- 是 --&gt; DBQuery     DBQuery --&gt; DBSuccess{查询成功?}     DBSuccess -- 否 --&gt; Fail4([返回请求失败，及失败原因])     DBSuccess -- 是 --&gt; ReturnSuccess([返回请求成功，及镜像列表])         </pre>		

(续表 5-18)

关键代码	<pre>//从 Docker Server 中提取数据 if(\$defaultData == 1) {     \$hostAddress = \$hostInfo[0].".\$hostInfo[1];     \$DockerSDK = new DockerSDKService(\$hostAddress);     \$containerChanges = \$DockerSDK-&gt;listImages(); } \$ImageModel = new ImageListViewModel(); \$ImageList = \$ImageModel-&gt;where(['host_id' =&gt; \$hostInfo[2]])-&gt;select();</pre>
------	---

#### 5.4.4 删除镜像

根据镜像名称从 Docker Server 删除镜像，根据镜像主键从数据库中删除镜像，详细设计如表 5-19 所示。

表 5-19 删除镜像详细设计

方法名称	删除镜像	方法属性	公有
Url	Home/Image/deleteImage		
参数	imageName String 镜像名称 imageKey int 镜像主键 hostInfo String 主机信息，包括主机、端口号、IP		
返回值	success Bool 请求成功或失败 error String 请求失败的原因		
特殊说明	无		
流程图	<pre> graph TD     Start([开始]) --&gt; Filter[过滤参数]     Filter --&gt; ParamCheck{参数是否符合要求?}     ParamCheck -- 否 --&gt; Fail1([返回请求失败, 及失败原因])     ParamCheck -- 是 --&gt; DockerDel[从Docker Server中删除镜像]     DockerDel --&gt; DockerSuccess{删除成功?}     DockerSuccess -- 否 --&gt; Fail2([返回请求失败, 及失败原因])     DockerSuccess -- 是 --&gt; DBDel[从数据库中删除镜像]     DBDel --&gt; DBSuccess{删除成功?}     DBSuccess -- 否 --&gt; Fail3([返回请求失败, 及失败原因])     DBSuccess -- 是 --&gt; Success([返回请求成功])           </pre>		



(续表 5-19)

关键代码	<pre>//从 Docker Server 中移除镜像 \$DockerSDK = new DockerSDKService(\$hostAddress); \$removeBool = \$DockerSDK-&gt;removeImage(\$imageName); //从数据库中移除镜像 \$imageModel = D('ImageInfo'); \$deleteBool = \$imageModel-&gt;where(['image_key' =&gt; \$imageKey])-&gt;delete();</pre>
------	---

## 5.5 系统日志管理详细设计与实现

### 5.5.1 系统日志管理模块设计

系统日志管理模块是系统的主要模块之一，用于进行系统各项操作进行记录。本小结介绍该模块的详细设计与实现。日志基本信息包括：操作时间、操作人和具体操作。模块界面设计如下图 5-12。

Docker 分布式应用控制系统 Containers Images Hosts Logs 当前登录用户: sheep_luo Exit				
查询条件				
操作人 <input type="text"/> 日期范围 <input type="text"/> 到 <input type="text"/> Search				
序号	操作人	具体操作	操作时间	
1	sheep_luo	禁用用户名列表	2016-05-01 19:26:34	
2	sheep_luo	查询主机列表	2016-05-01 19:26:58	
3	sheep_luo	查询镜像列表	2016-05-01 19:26:59	
4	sheep_luo	查询镜像列表	2016-05-01 19:27:00	
5	sheep_luo	查询镜像列表	2016-05-01 19:27:01	
6	sheep_luo	查询镜像列表	2016-05-01 19:27:02	

图 5-12 系统日志管理

### 5.5.2 查询系统日志

用户进行每一项操作之后，会自动记录日志。查询系统日志时，条件包括用户编号，开始日期和结束日期，开始日期和结束日期必须同时存在或同时不存在，若同时存在，查询日志的时间将在该范围内。详细设计如表 5-20 所示。

表 5-20 查询系统日志详细设计

方法名称	查询系统日志	方法属性	公有
Url	Home/User/searchSystemLog		
参数	userId int 用户编号 startDate date 开始日期 endDate date 结束日志		

(续表 5-20)

返回值	success Bool 请求成功或失败 error String 请求失败的原因 logList Array 系统日志列表
特殊说明	无
流程图	<pre> graph TD     Start([开始]) --&gt; Filter[过滤参数]     Filter --&gt; ParamCheck{参数是否符合要求?}     ParamCheck -- 否 --&gt; Fail1([返回请求失败, 及失败原因])     ParamCheck -- 是 --&gt; Query[查询系统日志]     Query --&gt; SuccessCheck{查询成功?}     SuccessCheck -- 否 --&gt; Fail2([返回请求失败, 及失败原因])     SuccessCheck -- 是 --&gt; Success([返回请求成功, 及系统日志])         </pre>
关键代码	<pre> \$pageSize = I('get.pageSize') == "" ? 20 : I('get.pageSize'); \$nowPage = I('get.nowPage') == "" ? 1 : I('get.nowPage'); \$systemLogsModel = new SystemLogsViewModel(); \$logList = \$systemLogsModel-&gt;where(\$searchData)-&gt;page(\$nowPage, \$pageSize)-&gt;select(); \$count = \$systemLogsModel-&gt;where(\$searchData)-&gt;count();// 查询满足要求的 总记录数         </pre>

## 5.6 用户管理详细设计与实现

### 5.6.1 用户管理模块设计

用户管理模块是系统的模块之一，主要用于对系统进行权限管理，增加系统安全度。本小结介绍该模块的详细设计与实现。用户基本信息包括：用户名、用户编号和用户密码，登录只需要使用用户名和用户密码。用户登录前后端均采用 MD5 方式进行加密和验证。该模块只有用户登录，没有用户注册。登录所需的账号和密码通过后台自动生成。登录界面设计如图 5-13。



图 5-13 用户登录

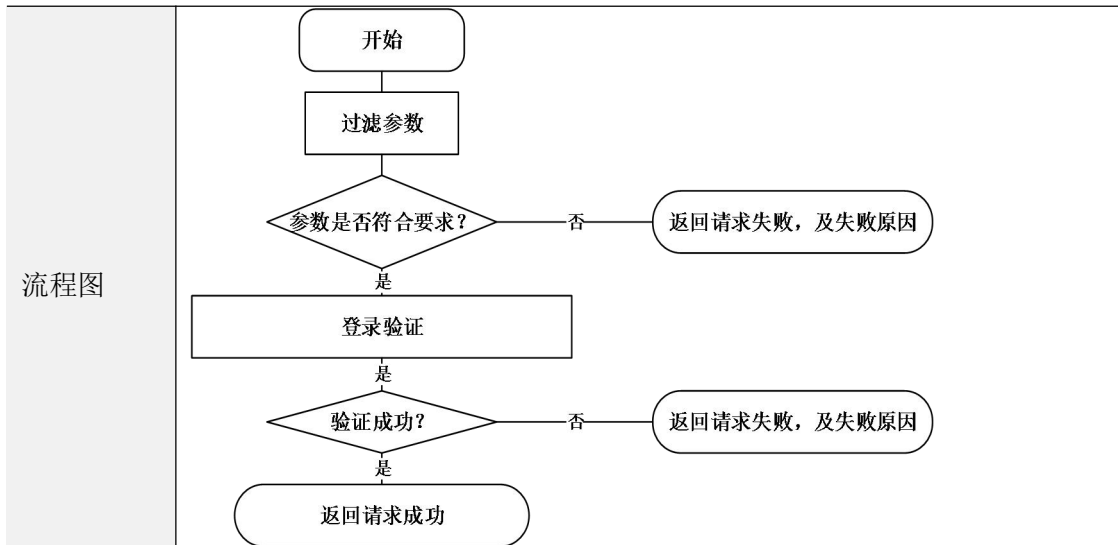
### 5.6.2 用户登录

用户登录详细设计如表 5-21 所示。

表 5-21 用户登录详细设计

方法名称	用户登录	方法属性	公有
Url	Home/User/login		
参数	username String 用户名 password String 用户密码		
返回值	success Bool 请求成功或失败 error String 请求失败的原因		
特殊说明	无		
关键代码	<pre>\$loginData['username']))-&gt;field('user_id, password')-&gt;select(); if(!\$UserInfo) {     \$this-&gt;ajaxReturn(['success'=&gt;false, 'error'=&gt;'用户名不存在!']); } if(!password_verify(\$loginData['password'], \$UserInfo[0]['password'])) {     \$this-&gt;ajaxReturn(['success'=&gt;false, 'error'=&gt;'密码错误!']); }</pre>		

(续表 5-21)



### 5.6.3 退出系统

退出系统详细设计如表 5-22 所示。

表 5-22 退出系统详细设计

方法名称	退出系统	方法属性	公有
Url	Home/User/exitSystem		
参数	无		
返回值	无		
特殊说明	无		
流程图	<pre> graph TD     Start([开始]) -- 是 --&gt; ExitSystem[退出系统]     ExitSystem -- 是 --&gt; End([结束])           </pre>		
关键代码	session('userLogin', null);		

### 5.6.4 获取用户列表

该功能主要用于系统日志查询中进行用户分别查询时使用。详细设计如表 5-23 所示。

表 5-23 获取用户列表详细设计

方法名称	查询系统日志	方法属性	公有
Url	Home/User/getUsers		

(续表 5-23)

参数	无
返回值	success Bool 请求成功或失败 error String 请求失败的原因 userList Array 用户列表
特殊说明	无
流程图	<pre>graph TD; Start([开始]) --&gt; Query[查询用户列表]; Query --&gt; Success{查询成功?}; Success -- 是 --&gt; ReturnSuccess([返回请求成功, 用户列表]); Success -- 否 --&gt; ReturnFailure([返回请求失败, 及失败原因]);</pre>
关键代码	<code>\$userModel = D('UserInfo');</code> <code>\$users = \$userModel-&gt;field('user_id,user_name')-&gt;select();</code>

## 5.4 本章小结

本章主要讲述了系统各个模块的详细设计内容，各个功能都有对应的实现方法，针对每个方法详细描述了参数、返回值、流程图和关键代码。本章内容为后期开发做好了充足的准备。

## 第 6 章 应用控制系统测试

### 6.1 测试环境

硬件环境：华硕 K450V

操作系统：Window10

运行环境： VMWare(Docker Server) + MySQL 5.6+ PHPStorm10 +Nginx

测试浏览器：Chrome49、FireFox45 、IE8、IE9、IE10、IE11

### 6.2 测试原则

软件测试是指在规定的条件下对程序进行操作，以发现程序错误，衡量软件质量，并对其是否能满足设计要求进行评估的过程，也是使用人工操作或者软件自动运行的方式来检验它是否满足规定的需求或弄清预期结果与实际结果之间的差别的过程。

测试应用程序只能显示在应用程序中存在一个或多个缺陷，但是，仅仅通过测试并不能证明应用程序没有错误。因此，设计测试用例使其尽可能多的找到缺陷是很重要的。本章根据需求说明中的功能尽可能的设计了完整的测试用例。但是，除了受测试应用具有非常简单的逻辑结构和有限的输入，否则进行所有测试数据和场景的组合是不可能的事。故在本章的测试中，针对某些过多输入的功能点只设计了典型的测试用例。

本章所测内容均已达到软件需求规格说明书要求，所发现的问题也得到全部修复。

### 6.3 功能性测试

#### 6.3.1 主机管理模块

该模块主要针对主机和主机的增删改查操作进行测试。详细测试结果见表 6-1 至表 6-7。

表 6-1 创建主机功能测试

用例编号	输入	期望输出	实际输出	测试结果
createHost-01	输入主机地址：192.168.0 输入主机名称：test01 输入端口号：1342 选择主机分组：未分组 点击“添加”	系统提示“主机地址错误”	系统提示“主机地址错误”	通过
createHost-02	输入主机地址：192.168.0.1 输入主机名称：cascd, 输入端口号：1342 选择主机分组：未分组 点击“添加”	系统提示“主机名称只能包含字母数字和下划线且长度为3到50！”	系统提示“主机名称只能包含字母数字和下划线且长度为3到50！”	通过
createHost-03	输入主机地址：192.168.0.1 输入主机名称：ctest01 输入端口号：1342 选择主机分组：未分组 点击“添加”	由于主机192.168.0.1不存在，故系统应当提示“请检查主机IP和端口号是否正确！”	系统提示“请检查主机IP和端口号是否正确！”	通过
createHost-04	输入主机地址：192.168.109.128 输入主机名称：addImage 输入端口号：4232 选择主机分组：未分组 点击“添加”	系统提示“该主机名称已存在！”	系统提示“该主机名称已存在！”	通过
createHost-05	输入主机地址：192.168.109.128 输入主机名称：hostTest 输入端口号：4232 选择主机分组：未分组 点击“添加”	创建成功，显示最新主机列表	创建成功，显示最新主机列表	通过

表 6-2 删除主机功能测试

用例编号	输入	期望输出	实际输出	测试结果
deleteHost-01	选择待删除的主机	删除成功，显示最细主机列表	删除成功，显示最细主机列表	通过

表 6-3 查看主机列表功能测试

用例编号	输入	期望输出	实际输出	测试结果
hostList-01	进入主机管理页面	系统自动显示主机列表，列表内容包括：序号、IP、Name、Port、Running Container、All Container、Image Number、Group、Actions（主机操作）	系统自动显示主机列表，列表内容包括：序号、IP、Name、Port、Running Container、All Container、Image Number、Group、Actions（主机操作）	通过
hostList-02	在主机名称搜索框输入相关词汇	系统根据词汇显示出相关主机列表	系统根据词汇显示出相关主机列表	通过

表 6-4 添加主机分组功能测试

用例编号	输入	期望输出	实际输出	测试结果
addHostGroup-01	输入分组名称: '!xsan' 点击“添加”	系统提示“分组名称不能包含特殊符号！”	系统提示“分组名称不能包含特殊符号！”	通过
addHostGroup-02	输入分组名称: 分组2	系统提示“该分组名称已存在”	系统提示“该分组名称已存在”	通过
addHostGroup-03	输入分组名称: 哈哈	添加成功	添加成功	通过
addHostGroup-04	输入分组名称: 分组,。;	系统提示“分组名称不能包含特殊符号！”	系统提示“分组名称不能包含特殊符号！”	通过



表 6-5 删除主机分组功能测试

用例编号	输入	期望输出	实际输出	测试结果
deleteHostGroup-01	删除“未分组”	系统提示“删除失败！可能原因：主机分组编号不合法，或者待删除分组为“未分组”。”	系统提示“删除失败！可能原因：主机分组编号不合法，或者待删除分组为“未分组”。”	通过
deleteHostGroup-02	删除“分组1”	删除成功，分组列表不再显示“分组1”，“分组1”包含的主机所在分组变更为“未分组”	删除成功，分组列表不再显示“分组1”，“分组1”包含的主机所在分组变更为“未分组”	通过

表 6-6 查看主机分组列表功能测试

用例编号	输入	期望输出	实际输出	测试结果
groupList-01	访问主机管理页面	显示主机分组列表，列表内容只包含分组名称	显示主机分组列表，列表内容只包含分组名称	通过

表 6-7 更改主机所在分组功能测试

用例编号	输入	期望输出	实际输出	测试结果
changeGroup-01	选择待更改分组的主机 选择新分组 点击“更改”	主机所在分组变为新分组	主机所在分组变为新分组	通过
changeGroup-02	选择待更改分组的主机 不选择分组 点击“更改”	系统不做任何修改	系统不做任何修改	通过

## 6.3.2 容器管理模块

该模块主要针对容器的增删改查操作进行测试。详细测试结果见表 6-8 至表 6-20。

表 6-8 创建容器功能测试

用例编号	输入	期望输出	实际输出	测试结果
createContainer-01	无任何输入 点击“Create Container”	系统提示“请选择镜像名称”	系统提示“请选择镜像名称”	通过
createContainer-02	选择镜像名称 点击“Create Container”	创建成功，显示最新容器列表。	创建成功，显示最新容器列表。	通过
createContainer-03	选择镜像名称 输入 Command 点击“Create Container”	创建成功，显示最新容器列表。	创建成功，显示最新容器列表。	

表 6-9 启动容器功能测试

用例编号	输入	期望输出	实际输出	测试结果
startContainer-01	选择待启动的容器 some-mysql， 该容器状态为 paused; 点击“Start”	系统提示“Cannot start a paused container, try unpause instead.”	系统提示“Cannot start a paused container, try unpause instead.”	通过
startContainer-02	选择待启动的多个容器 点击“Start”	选中容器本身不是暂停状态，则容器状态变为 running，否则系统提示“Cannot start a paused container, try unpause instead.”	选中容器本身不是暂停状态，则容器状态变为 running，否则系统提示“Cannot start a paused container, try unpause instead.”	通过

表 6-10 暂停容器功能测试

用例编号	输入	期望输出	实际输出	测试结果
pauseContainer-01	选择待暂停的容器 hello-mysql, 此时该容器状态为 running; 点击 “Pause”	容器状态变为 paused	容器状态变为 paused	通过
pauseContainer-02	选择待暂停的容器 hello-mysql, 此时该容器状态为 exited; 点击 “Pause”	系统提示 “ container is not running”	系统提示 “container is not running”	通过

表 6-11 重启容器功能测试

用例编号	输入	期望输出	实际输出	测试结果
restartContainer-01	选择待重启的容器 hello-mysql, 该容器状态为 exited; 点击 “Restart”	该容器状态变为 running	容器状态变为 running	通过
restartContainer-02	选择待重启的容器 some-mysql, 该容器状态为 paused; 点击 “Restart”	系统提示 “ Cannot start a paused container, try unpause instead.”	系统提示 “Cannot start a paused container, try unpause instead.”	通过
restartContainer-03	选择待重启的多个容器 点击 “Restart”	选中容器本身不是暂停状态, 则容器状态变为 running, 否则系统提示 “Cannot start a paused container, try unpause instead.”	选中容器本身不是暂停状态, 则容器状态变为 running, 否则系统提示 “ Cannot start a paused container, try unpause instead.”	通过

表 6-12 恢复暂停容器功能测试

用例编号	输入	期望输出	实际输出	测试结果
unpauseContainer-0 1	选择待恢复暂停的容器 some-mysql，此时容器 状态为 paused； 点击 “Unpause”	容器状态变 为 running	容 器 状 态 变 为 running	通过
unpauseContainer-0 2	选择待恢复暂停的容器 some-mysql，此时容器 状态为 running； 点击 “Unpause”	系 统 提 示 “ Container is not paused”	系 统 提 示 “ Container is not paused”	通过

表 6-13 停止容器功能测试

用例编号	输入	期望输出	实际输出	测试结果
stopContainer-0 1	选 择 待 停 止 的 容 器 some-mysql，该 容 器 状 态 为 paused； 点击 “Stop”	系统提示 “Cannot stop unpauseContainer”	系统提示 “Cannot stop unpauseContainer ”	通过
stopContainer-0 2	选 择 待 停 止 的 容 器 some-mysql，该 容 器 状 态 为 running； 点击 “Stop”	容 器 状 态 变 为 exited	容 器 状 态 变 为 exited	通过

表 6-14 查看容器详细信息功能测试

用例编号	输入	期望输出	实际输出	测试结果
containerInfo- 01	访 问 容 器 管 理 页 面，在容器列表中 点击想要查看的 容器名称	页面跳转至容器详 细信息页面，并显 示对应容器详细信 息	页面跳转至容器详 细信息页面，并显示对应 容器详细信息	通过

表 6-15 终止容器功能测试

用例编号	输入	期望输出	实际输出	测试结果
killContainer-01	选择待停止的容器 some-mysql, 该容器状态为 paused; 点击 “Stop”	系统提示 “Cannot kill unpauseContainer”	系统提示 “Cannot kill unpauseContainer”	通过
killContainer-02	选择待停止的容器 some-mysql, 该容器状态为 running; 点击 “Stop”	容器状态变为 exited	容器状态变为 exited	通过

表 6-16 重命名容器功能测试

用例编号	输入	期望输出	实际输出	测试结果
renameContainer-01	选择容器 some-mysql; 输入容器新名称 (旧名称一样)	系统不做任何反应	系统不做任何反应	通过
renameContainer-02	选择容器 some-mysql; 输入容器新名称	系统显示最新容器列表	系统显示最新容器列表	通过

表 6-17 移除容器功能测试

用例编号	输入	期望输出	实际输出	测试结果
removeContainer-01	选择待移除的容器 sheep-mysql	容器移除成功, 系统显示最新容器列表	容器移除成功, 系统显示最新容器列表	通过
removeContainer-02	选择待移除的多个容器	容器移除成功, 系统显示最新容器列表	容器移除成功, 系统显示最新容器列表	通过

表 6-18 查看容器列表功能测试

用例编号	输入	期望输出	实际输出	测试结果
containerList-0 1	访问容器管理页面，不输入任何查询条件	系统显示默认主机中容器列表	系统显示默认主机中容器列表	通过
containerList-0 2	输入查询条件：容器状态选为 running；	系统显示默认主机中状态为 running 的容器列表	系统显示默认主机中状态为 running 的容器列表	通过
containerList-0 3	在容器名称搜索框中输入关键词汇	系统显示容器名称与关键词汇相关的容器列表	系统显示容器名称与关键词汇相关的容器列表	通过
containerList-0 4	在查询条件中选择主机分组为 addImage	系统显示主机 addImage 中所有容器列表	系统显示主机 addImage 中所有容器列表	通过

表 6-19 查看容器进程功能测试

用例编号	输入	期望输出	实际输出	测试结果
containerProcess-0 1	查看容器 some-mysql 进程，此时该状态为 exited	系统提示“Container is not running”	系统提示“Container is not running”	通过
containerProcess-0 2	查看容器 some-mysql 进程，此时该状态为 paused	系统显示容器进程信息列表	系统显示容器进程信息列表	通过
containerProcess-0 3	查看容器 some-mysql 进程，此时该状态为 running	系统显示容器进程信息列表	系统显示容器进程信息列表	通过
containerProcess-0 4	查看容器 some-mysql 进程，此时该状态为 created	系统提示“Container is not running”	系统提示“Container is not running”	通过

表 6-20 查看容器日志功能测试

用例编号	输入	期望输出	实际输出	测试结果
containerLog-01	查看容器 some-mysql 日志	系统显示容器 所有日志	系统显示容器所有 日志	通过

### 6.3.3 镜像管理模块

该模块主要针对镜像的增删改查操作进行测试。详细测试结果见表 6-21 至表 6-24。

表 6-21 拉取镜像功能测试

用例编号	输入	期望输出	实际输出	测试结果
pullImage-01	输入 镜 像 名 称 : nxasdjnjaskj 点击“添加”	系统提示“该镜像 不存在”	系统提示“该镜像 不存在”	通过
pullImage-02	输入 镜 像 名 称 : alpine:3.2 点击“添加”	镜像拉取成功, 显 示最新镜像列表	镜像拉取成功, 显 示最新镜像列表	通过

表 6-22 查看镜像列表功能测试

用例编号	输入	期望输出	实际输出	测试结果
imageList-01	访问镜像管理页面	系统自动显示默认 主机中所有镜像列 表	系统自动显示默 认主机中所有镜 像列表	通过
imageList-02	访问镜像管理页面 选择想要查看的主机 addImage	系 统 显 示 主 机 addImage 中所有镜 像列表	系统显示主机 addImage 中所有 镜像列表	通过

表 6-23 刷新镜像列表功能测试

用例编号	输入	期望输出	实际输出	测试结果
refreshList-01	访问镜像管理页面, 点击“Refresh”	系统自动刷新该主机 所有镜像信息	系统自动刷新该主 机所有镜像信息	通过

表 6-24 删除镜像功能测试

用例编号	输入	期望输出	实际输出	测试结果
deleteImage-01	选择待删除的镜像	显示最新的镜像列表	显示最新的镜像列表	通过

### 6.3.4 用户管理模块

该模块主要针对用户登录和退出进行测试。详细测试结果见表 6-25 至表 6-26。

表 6-25 用户登录功能测试

用例编号	输入	期望输出	实际输出	测试结果
login-01	直接点击“登录”	提示“用户名和密码不能为空”	提示“用户名和密码不能为空”	通过
login-02	输入用户名: asds 输入密码: xasdj 点击“登录”	提示“用户名不存在”	提示“用户名不存在”	通过
login-03	输入用户名: sheep_luo 输入密码: sheep	提示“密码错误!”	提示“密码错误!”	通过
login-04	输入用户名: sheep_luo 输入密码: sheep_luo	登录成功, 页面跳转至容器管理页面	登录成功, 页面跳转至容器管理页面	通过
login-05	不输入用户名 输入密码: xxx	提示“用户名和密码不能为空”	提示“用户名和密码不能为空”	通过
login-06	输入用户名: sheep_luo 不输入密码	提示“用户名和密码不能为空”	提示“用户名和密码不能为空”	通过

表 6-26 退出系统

用例编号	输入	期望输出	实际输出	测试结果
exitSystem-01	点击“exit”	用户退出系统, 页面跳转至登录页面	用户退出系统, 页面跳转至登录页面	通过

### 6.3.5 系统日志管理模块

该模块主要针对系统日志查询进行测试。详细测试结果见 6-27。



表 6-27 查看系统日志列表功能测试

用例编号	输入	期望输出	实际输出	测试结果
systemLog-01	访问系统日志页面	系统显示默认的 日志列表 20 条	系统显示默认的 日志列表 20 条	通过
systemLog-02	输入查询条件如下， 选择操作人：sheep_luo 选择开始日期：无 选择结束日志：无 点击“Search”	系 统 显 示 sheep_luo 的前 20 条系统日志列表	系 统 显 示 sheep_luo 的 前 20 条操作记录	通过
systemLog-03	输入查询条件如下， 选择操作人：无 选 择 开 始 日 期 2016-05-01 选 择 结 束 日 志 2016-05-05 点击“Search”	系统显示日期在 2016-05-01 和 2016-05-05 之 间 的前 20 系统日志 列表	系统显示日期在 2016-05-01 和 2016-05-05 之 间 的前 20 系统日 志	通过
systemLog-04	输入查询条件如下， 选择操作人：sheep_luo 选 择 开 始 日 期 2016-05-01 选 择 结 束 日 志 2016-05-05 点击“Search”	系 统 显 示 sheep_luo 在 2016-05-01 到 2016-05-05 之 间 的前 20 条系统日 志列表	系 统 显 示 sheep_luo 在 2016-05-01 到 2016-05-05 之 间 的前 20 条系统 日志	通过
systemLog-05	无任何查询条件 点击查看第 5 页系统日 志	系统显示第 5 页 的系统日志列表	系统显示第 5 页 的系统日志列表	通过

## 6.4 兼容性测试

表 6-28 展示了兼容性测试过程出现的问题以及该问题是否已经解决。其中，“Y”表示在该浏览器中存在问题，“N”表示在该浏览器中不存在问题。

表 6-28 兼容性测试结果

	Chrome49	firefox45	IE11	IE10	IE9	IE8	是否更正
Angular 指令运行出错	N	N	N	N	N	Y	是
添加分组时报错：trim 方法错误	N	N	N	N	N	Y	是
日期选择器中当天日期 Hover 时，IE8、IE9 和其它浏览器表现差别太大	N	N	N	N	Y	Y	是

## 6.5 测试过程中发现的问题总结与分析

有些系统 bug 在开发阶段就能发现，但完美开发总是少见，测试总能发现多多少少的问题。对系统进行了功能测试之后，发系统中出现了较多的边界问题，导致测试结果和期望不一样。发现这些问题后，进行了及时更正，保证了实际运行结果和期望一致。

信息系统运行于各大浏览器中，其特殊性决定了兼容性测试的重要性。选题要求信息系统在 chrome、firefox、IE8+等浏览器中正常显示和运行，故而对这些浏览器进行了逐个测试，也发现了一些问题，问题主要集中在 IE8 和 IE9。对于出现的问题，系统也进行了相应的修正。

## 6.6 本章小结

本章主要讲述了系统开发结束后的测试阶段，包括测试环境概述、功能性测试详细描述、兼容性测试等。在进行了一系列的测试工作之后，对测试工作进行了简要的总结与分析。

## 结论

结合国内外选题相关的研究现状，本文主要首先介绍了选题的研究目的和意义。就目前来说，**Docker** 是环境部署大势所趋，一个项目部署在多个服务器是再正常不过的事儿，提升 **Docker** 容器镜像操作速度和集群化管理也显得越来越重要。因此选题着重研究容器管理、镜像管理和主机管理，选题有了大概内容后，便对选题设计了实施方案，如何进行，如何实施，如何结束。

接着介绍了选题相关的技术，**Docker**、**PHP**、**curl** 编程都是作者未曾涉猎的知识范围，为了顺利完成毕业设计，在这几方面下了不少功夫。

清楚了选题的背景和相关技术后，拟写了相关需求规格说明。分别从业务需求、功能性需求和非功能性需求几方面进行详细阐述。在明确了具体要实现什么之后，便是如何实现。接下来介绍的系统概要设计和详细设计便针对如何实现系统需求进行了详细描述。信息系统一般的设计思想是从整体到局部，从概要到详细，有了这一过程，才能对如何实现系统功能有足够清晰的认识和把握。

系统开发完成后还需要进行良好的测试工作才能保证系统在线上正常运行，因此，本文结尾处书写了系统主要功能的测试用例。

系统的完成大大减少了运维和开发 **Docker** 管理的操作时间，提高了各自的工作效率。

系统完成后，也存在着一些不足的地方，如系统日志，由于管理员的每次操作都进行了记录，随着后期操作越来越多可能会对系统造成一定的性能影响，可以考虑使用一些日志中间件代替。

## 致谢

在前端开发和论文编写等方面，杨雷老师一直悉心给出建议，节约了不少开发和论文编写时间，最后顺利完成，感谢杨老师一路辛苦付出。

系统设计前期，对自己要做什么几乎没有概念。Docker、SDK、CURL 编程、Remote API、ThinkPHP，这一系列的陌生单词让人头晕目眩，找不着地方下口。由于系统基于 Docker，便对 Docker 进行了一系列的基础学习。感谢刘星杰同学的指导，让我对 Docker 着一单词有了初步的认识。我所需要掌握的无非是在 Linux 系统中进行镜像和容器的创建、查询、删除等命令。

Docker 镜像和容器相关命令有对应的 Remote API，在 PHP 中操作这些 API 相当于书写 Linux 命令，只是需要 CURL 编程牵线搭桥，感谢鄢鹏权学长帮忙梳理这一层关系。

由于要进行后台开发，PHP 学习成为必要。在 ThinkPHP 学习过程中，感谢熊艺峰同学的指点，这为我后期独立后台开发奠定了良好基础。

## 参考文献

- [1]王飞.基于 Docker 的研发部署管理平台的设计与实现[D].北京: 北京交通大学.2015.
- [2]Ben Golub.Docker Remote API v1.22[CP].[https://docs.docker.com/engine/reference/api/docker\\_remote\\_api\\_v1.22/](https://docs.docker.com/engine/reference/api/docker_remote_api_v1.22/).2013/2016.
- [3]鞠春利 刘印锋.基于 Docker 的私有 PaaS 系统构建[J].计算机信息与技术.2014, 10: 80, 83.
- [4]曾金龙, 肖新华, 刘清.Docker 开发实践[M].北京: 人民邮电出版社, 2015.7: 2-47.
- [5]Brad Green, Shyam Seshadri.用 Angular 开发下一代 Web 应用[M].北京: 电子工业出版社, 2013: 1-47.
- [5]Shyam Seshadri, Brad Green.AngularJs 即学即用[M].北京: 中国电力出版社, 2016: 22-40, 55-73, 144-171.
- [6]陶国荣.Angular Js 实战[M].北京: 机械工业出版社, 2010: 16-69.
- [7]传智博客高教产品研发部.PHP 网站开始实例教程[M].北京: 人民邮电出版社, 2015: 182-201.
- [8]林玉斌. 基于 THINKPHP 的毕业论文选题系统的设计[J]. 计算机光盘软件与应用, 2013, No.21104: 197-198.
- [9]Aravind Shenoy, Ulrich Sossou.Bootstrap 开发精解[M].北京: 机械工业出版社, 2016: 31-96.
- [10]向华. Web 功能测试技术研究[J]. 软件导刊, 2006, 19: 7-8.
- [11]马朝霞. 基于 Web 应用软件测试内容及方法的研究[J]. 中外企业家, 2015, No.49715: 193.
- [12]Andrej Harsani, Viktor Zeman.How to make REST calls in PHP[CP].<https://support.ladesk.com/061754-How-to-make-REST-calls-in-PHP>, 2015/2016.
- [13]刘思尧, 李强, 李斌.基于 Docker 技术的容器隔离性研究[J].国际 IT 传媒品牌.2015, 36 (4): 110-113.
- [14]Pete Bacon Darwin.Angular Js v1.2.27[CP].<https://code.angularjs.org/1.2.27/docs/api/ng/function>.2014/2016.
- [15]Lukasz Kryger.Updating to RequireJs 2.0 [CP].<http://requirejs.org/docs/api.html.2015-5-22/2016-05-10>.