目录

[1 引言 2](#_Toc13685103)

[1.1 编写目的 2](#_Toc13685104)

[1.2 背景 2](#_Toc13685105)

[1.3 参考资料 2](#_Toc13685106)

[1.4 术语定义及说明 2](#_Toc13685107)

[2 设计概述 3](#_Toc13685108)

[2.1 任务和目标 3](#_Toc13685109)

[2.1.1 需求概述 3](#_Toc13685110)

[2.1.2 运行环境概述 3](#_Toc13685111)

[2.1.3 条件与限制 3](#_Toc13685112)

[2.1.4 详细设计方法和工具 3](#_Toc13685113)

[3 系统详细需求分析 4](#_Toc13685114)

[3.1 详细需求分析 4](#_Toc13685115)

[3.1.1 详细功能需求分析 4](#_Toc13685116)

[3.1.2 详细性能需求分析 5](#_Toc13685117)

[3.1.3 详细资源需求分析 5](#_Toc13685118)

[3.1.4 详细系统运行环境及限制条件分析 5](#_Toc13685119)

[3.2 详细系统运行环境及限制条件分析接口需求分析 5](#_Toc13685120)

[3.2.1 系统接口需求分析 5](#_Toc13685121)

[3.2.2 现有硬件、软件资源接口需求分析 6](#_Toc13685122)

[4 总体方案确认 9](#_Toc13685123)

[4.1 系统总体结构确认 9](#_Toc13685124)

[4.2 系统详细界面划分 10](#_Toc13685125)

[4.2.1 应用系统与支撑系统的详细界面划分 10](#_Toc13685126)

[5 系统详细设计 11](#_Toc13685127)

[5.1 系统结构设计及子系统划分 11](#_Toc13685128)

[5.2 系统功能模块详细设计 11](#_Toc13685129)

[5.3 系统界面详细设计 12](#_Toc13685130)

[5.3.1 内部界面设计 12](#_Toc13685131)

[5.3.2 用户界面设计 12](#_Toc13685132)

[6 数据库系统设计 13](#_Toc13685133)

[6.1 设计要求 13](#_Toc13685134)

[6.2 信息模型设计 13](#_Toc13685135)

[6.3 数据库设计 13](#_Toc13685136)

[6.3.1 设计依据 13](#_Toc13685137)

[6.3.2 数据库逻辑结构 13](#_Toc13685138)

[6.3.3 物理结构设计 14](#_Toc13685139)

[6.3.4 数据库安全 14](#_Toc13685140)

[6.3.5 数据字典 14](#_Toc13685141)

# 引言

## 编写目的

本文档旨在描述CloudHub平台的详细设计，主要内容包括系统功能简介、系统结构设计、系统结构设计、系统接口设计、模块设计和界面设计等，各个模块独立，耦合性小，便于开发，。

本文档预期的读者包括设计人员、开发人员、项目管理人员、测试人员。

## 背景

基于docker轻量级的特性，在当今IT行业越来越盛行，如阿里、腾讯等平台都为此设计提供服务的Paas平台，需要收取一定的费用，为了让广大用能够免费体验docker容器的新世界，我们为此开展了这次基于docker容器的Pass平台的开发，通过go与docker的结合为用户提供了docker的使用流程

开发项目：基于docker的Paas平台

团队成员：陈璐、潘华著、邹羽、刘虎、李心雨

指导老师：丁波

## 参考资料

Godoc官网原生接口文档

Dockrhub官网

K8s官网

K8s 常见问题总结

## 术语定义及说明

K8s (kubernetes)：是一个开源的，用于管理云平台中多个主机上的容器化的应用，Kubernetes的目标是让部署容器化的应用简单并且高效（powerful）,Kubernetes提供了应用部署，规划，更新，维护的一种机制。

Docker: 让开发者可以打包他们的应用以及依赖包到一个可移植的镜像中，然后发布到任何流行的 Linux或Windows 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口

Container:容器，通过镜像创建

Image:镜像

# 设计概述

## 任务和目标

前端设计任务：界面布局合理，简洁明了，又不失风格，让用户能够简单上手，具有自然的引导提示。

中间件设计任务：能够正确的实现逻辑的处理，支持与后端的重连机制，模块化的封装，参数化的定义，能够对前端的操作进行安全的判断与处理，对于用户隐藏后端的接口。

后端设计任务：对数据库进行合理的设计、sql语句的合理，支持对数据库的重连机制，对中间键传输的参数给予一定的安全判断和错误的返回。

### 需求概述

项目是基于docker的Paas平台，业务包括用户注册，用户登录，不同镜像的创建容器、启动容器、重启容器、停止容器、容器实时信息等，后台的用户管理、镜像管理、容器管理。

输入为用户更具网页点击对应功能

输出是中间件通过用户请求给予对应的反馈信息

主要的功能是docker容器的部署，创建、启动、停止、删除等操作

性能需求：用户能够流畅的在网页上操作，用户信息反馈延时不能超过2s，用户人数能够300人同时在线，异常断开连接时能够进行重连，系统的安全性和数据的安全性

### 运行环境概述

Window10的操作系统、Mysql数据库系统、阿里云服务器、胖中间件模式（前端-中间件-后端）

### 条件与限制

不可暴露容器远程控制接口，文件上下传接口

### 详细设计方法和工具

详细设计采用的工具用UML类图、E-R图，数据流程图、业务流程图等

# 系统详细需求分析

## 详细需求分析

### 详细功能需求分析

#### 用户功能需求:

1. 用户注册：通过账号，密码，邮箱以及邮箱验证码进行注册

账号要唯一，密码需要8-16位的字母与数字的组合，邮箱要有效，用户后期密码找回的凭证，验证码有效期为1分钟

1. 用户登录：通过账号、密码进行登录

账号、密码与后端数据库数据进行匹配

1. 用户密码找回：通过邮箱邮箱进行密码的找回，进行密码的重置
2. 用户信息管理：用户可以设置属于自己的信息，修改自己的信息

能够修改自己的邮箱、密码等

#### docker操作功能需求:

1. 创建容器：用户通过不同的服务镜像创建容器

在创建容器时，用户可以给自己的容器进行命名，能够选择所需要的cpu、内存的资源，对于mysql等服务，需要输入密码的能够让用户自己设定密码

1. 启动容器：通过点击启动，可以启动自己所创建的容器
2. 远程连接：在用户启动容器后能够远程连接给用户虚拟出一个操作的终端，用户你能够在终端界面进行操作，不能将远程的服务器的ip和端口暴露给用户
3. 重启容器：通过点击重启，能够对运行中的容器进行重启
4. 关闭容器：通过点击关闭，能够将运行中的容器进行关闭
5. 删除容器：通过店家删除，将创建的容器进行删除，注意删除容器前提要将容器关闭
6. 文件的传输：支持用户从本地向容器传输文件，也能够从容器中拷贝出文件
7. 容器使用的实时显示：将容器使用的cpu、内存等实时数据进行实时的反馈给用户
8. 容器端口的映射：向用户提供端口的映射信息

#### 后台管理功能需求：

1. 用户管理：对用户进行管理，管理员能够修改用户的权限，删除用户，分页的显示用户信息、支持模糊查询
2. 容器管理：对容器信息进行管理，管理员能够删除对应用户的容器、修改用户的容器信息，查询所有用户、部分用户的容器信息，支持模糊查询、分页显示
3. 镜像管理：对镜像信息进行管理，能够输入镜像名自动从远程下载需要的镜像，添加、删除、修改、查找镜像的信息，支持模糊查询、分页显示

#### 前端功能需求分析：

1. 能够合理的排版用户操作界面，便于用户的操作与理解，能够符合我们业务的风格，采用Echars等来渲染用户界面

### 详细性能需求分析

1. 用户响应时间不超过2s，提高用户体验，算法、处理的优化
2. 在线人数同时300人，能够支撑住多人同时在线操作，且不会引起处理的混乱
3. 支持中间件与后端的重连机制，当后端宕机，中间件会尝试去重连，当后端重启后，中间件会自动连接上，不需要中间件重新重启
4. 支持后端与数据库的重连机制，当与数据库断开连接后，后端会尝试去重连数据库，

数据库重连上，不需要后端的重新重启。

1. 动态刷新页面，当添加了新的镜像时，会动态的在前端显示新增的镜像

### 详细资源需求分析

Window10的操作系统，4台阿里云服务器用户k8s集群搭建、Mysql服务器作为数据服务器、docker服务器

### 详细系统运行环境及限制条件分析

不允许暴露后端数据库的接口：前端与后端服务器通过中间件进行隔离，保证用户无法直接与数据库进行操作

Docker上传下载文件、远程连接不暴露docker服务器的ip与端口，通过底层接口直接实现终端的映射与数据的传输

## 详细系统运行环境及限制条件分析接口需求分析

### 系统接口需求分析

由前端js调用中间件提供的http接口，再由中间件调用后台rpc服务器接口获取数据返回给前端js。

### 现有硬件、软件资源接口需求分析

中间件：

http处理器接口：

//登陆

func (this \*Handler) Login\_handler(w http.ResponseWriter, r \*http.Request)

//注册

func (this \*Handler) Register\_handler(w http.ResponseWriter, r \*http.Request)

//密码找回

func (this \*Handler) Retrieve\_handler(w http.ResponseWriter, r \*http.Request)

//主页

func (this \*Handler) Index\_handler(w http.ResponseWriter, r \*http.Request)

//个人信息

func (this \*Handler) Account\_handler(w http.ResponseWriter, r \*http.Request)

//创建容器

func (this \*Handler) Createdocker\_handler(w http.ResponseWriter, r \*http.Request)

//我的docker

func (this \*Handler) Mydocker\_handler(w http.ResponseWriter, r \*http.Request)

//docker容器资源（websocket）

func (this \*Handler) Resource\_handler(w http.ResponseWriter, r \*http.Request)

//docker容器终端（websocket）

func (this \*Handler) Shell\_handler(w http.ResponseWriter, r \*http.Request)

//docker容器文件上传（websocket）

func (this \*Handler) Upload\_handler(w http.ResponseWriter, r \*http.Request)

//docker容器文件下载

func (this \*Handler) Download\_handler(w http.ResponseWriter, r \*http.Request)

//刷新已加载的模板

func (this \*Handler) Refresh\_handler(w http.ResponseWriter, r \*http.Request)

rpc管理器接口：

//连接rpc服务器

func (this \*Rpc\_manger) conn\_rpc()

//统一rpc调用

func (this \*Rpc\_manger) Rpc\_call(method string, args interface{}, reply interface{}) error

//初始化rpc管理器

func (this \*Rpc\_manger) Init(host string)

//关闭rpc管理器

func (this \*Rpc\_manger) Close()

验证码管理器接口：

//发送验证码 参数:电子邮箱

func (this \*Captcha\_manger) Send\_captcha(email string) error

//获取以及产生的验证码 参数：电子邮箱

func (this \*Captcha\_manger) Get\_captcha(email string) string

//删除验证码 参数：电子邮箱

func (this \*Captcha\_manger) Delete\_captcha(email string)

//计时器 删除超时的验证码

func (this \*Captcha\_manger) timer\_captcha()

//验证码管理器初始化

func (this \*Captcha\_manger) Init(user string, pwd string, domain string, timeout int)

//验证码管理器关闭

func (this \*Captcha\_manger) Close()

docker管理器接口：

//连接docker服务器

func (this \*Docker\_manger) conn\_docker()

// 列出镜像

func (this \*Docker\_manger) Listimage(image \*[]string) error

//下载镜像

func (this \*Docker\_manger) Pullimage(image string) error

//创建docker容器

func (this \*Docker\_manger) Createcontainer(image string, ports string, cmd string) (string, error)

//限制 docker资源

func (this \*Docker\_manger) Containerupdate(containerID string, frequency int64, memory int64) error

//docker容器启动

func (this \*Docker\_manger) Startcontainer(containerID string) error

//docker容器重启

func (this \*Docker\_manger) Restartcontainer(containerID string) error

//docker容器关闭

func (this \*Docker\_manger) Stopcontainer(containerID string) error

// docker容器删除

func (this \*Docker\_manger) Removecontainer(containerID string) error

//返回docker容器host

func (this \*Docker\_manger) Returncontainerinfo(containerID string) ([]string, error)

//docker容器资源使用率

func (this \*Docker\_manger) Containerres(containerID string) ([]byte, error)

//连接docker容器实现远程终端功能

func (this \*Docker\_manger) Containershell(conn\_web \*websocket.Conn, settings []string)

//docker容器文件上传

func (this \*Docker\_manger) Containerfile\_upload(conn\_web \*websocket.Conn, settings []string)

//docker容器文件下载

func (this \*Docker\_manger) Containerfile\_download(w http.ResponseWriter, r \*http.Request, settings []string)

//初始化docker管理器

func (this \*Docker\_manger) Init(host string, cpu\_f int64)

//关闭docker管理器

func (this \*Docker\_manger) Close()

后端：

用户注册管理：

//添加用户

func (s \*Server) Insert\_userinfo\_secure(userinfo Userinfo\_secure, none \*int) error

//删除用户

func (s \*Server) Delete\_userinfo\_secure(id string, none \*int) error

//修改用户信息

func (s \*Server) Update\_userinfo\_secure(userinfo Userinfo\_secure, none \*int) error

//通过id单用户查找

func (s \*Server) Select\_userinfo\_secure\_byid(id string, userinfo\_out \*Userinfo\_secure) error

//通过email单用户查找

func (s \*Server) Select\_userinfo\_secure\_byemail(email string, userinfo\_out \*Userinfo\_secure) error

//模糊查找  
func (s \*Server) Select\_userinfo\_secures(id string, userinfo \*[]Userinfo\_secure) error

//分页查找

func (s \*Server) List\_userinfo\_secures(page int, userinfo \*[]Userinfo\_secure) error

//管理员用户查找

func(s\*Server)Listadmin\_userinfo\_secures(none int,userinfo \*[]Userinfo\_secure) error

//用户个数查找

func (s \*Server) List\_total(none int, num \*int) error

用户信息管理：

//添加用户信息

func (s \*Server) Insert\_userinfo\_account(userinfo Userinfo\_account, none \*int) error

//删除用户信息

func (s \*Server) Delete\_userinfo\_account(id string, none \*int) error

//修改用户信息

func (s \*Server) Update\_userinfo\_account(userinfo Userinfo\_account, none \*int) error

//查找单个用户信息

func (s \*Server) Select\_userinfo\_account(id string, userinfo\_out \*Userinfo\_account) error

镜像管理：

//添加镜像设置

func (s \*Server) Insert\_imagesetting(imageinfo Imagesetting, none \*int) error

//删除镜像设置

func (s \*Server) Delete\_imagesetting(imagename string, none \*int) error

//修改镜像设置

func (s \*Server) Update\_imagesetting(imageinfo Imagesetting, none \*int) error

//查询镜像设置

func (s \*Server) Select\_imagesetting(imagename string, imageinfo \*Imagesetting) error

//模糊查询

func (s \*Server) Select\_imagesettings(imagename string, imageinfo \*[]Imagesetting) error

//分页查找

func (s \*Server) List\_imagesettings(page int, imagesetting \*[]Imagesetting) error

容器管理：

//添加docker信息

func (s \*Server) Insert\_dockerinfo(dockerinfo Dockerinfo, none \*int) error

//通过containerid删除docker信息

func (s \*Server) Delete\_dockerinfo(containerid string, none \*int) error

//更新docker信息

func (s \*Server) Update\_dockerinfo(dockerinfoin Dockerinfo, none \*int) error

//通过id查询docker信息

func (s \*Server) Select\_dockerinfos(id string, dockerinfo \*[]Dockerinfo) error

//通过containerid查询docker信息

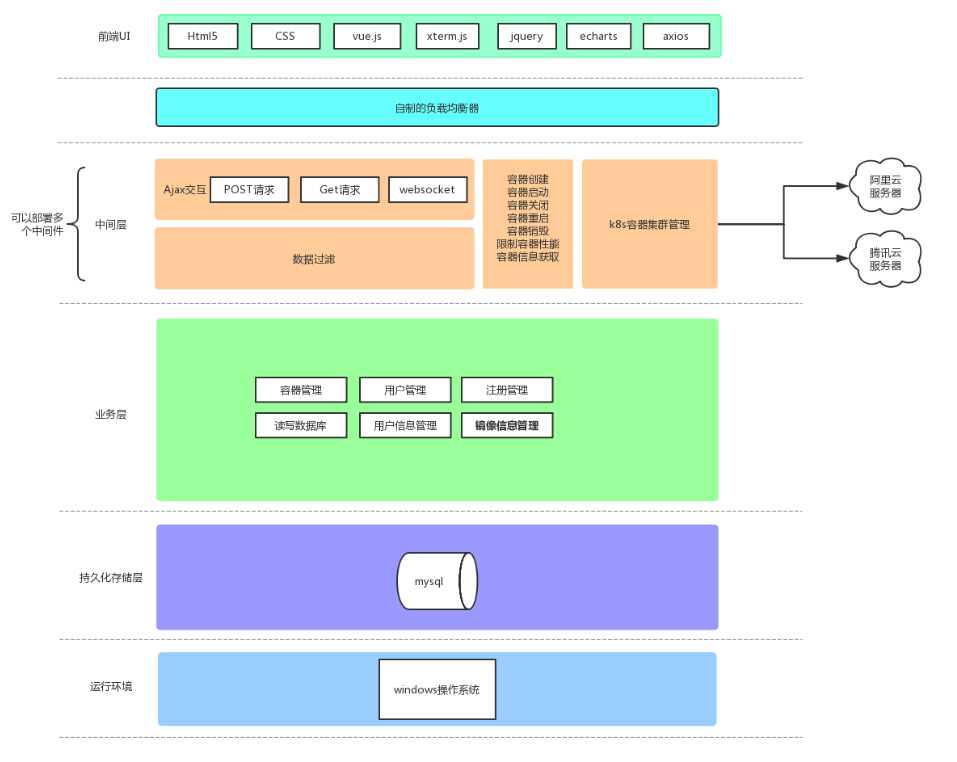
func (s \*Server) Select\_dockerinfo(containerid string, dockerinfo \*Dockerinfo) error

//分页查找

func (s \*Server) List\_dockerinfos(page int, dockerinfo \*[]Dockerinfo) error

# 总体方案确认

## 系统总体结构确认



## 系统详细界面划分

### 应用系统与支撑系统的详细界面划分

中间件服务器通过rpc的方式与后端服务器进行交互，通过tcp协议的方式与docker服务器进行交互，后端通过tcp协议与mysql服务器进行交互，前端通过http协议以及websocket协议与中间件服务器进行交互，用户通过html与前端交互。

# 系统详细设计

## 系统结构设计及子系统划分

1、用户登陆

2、用户注册

3、用户账户信息管理

4、用户密码找回

5、docker容器创建及配置

6、docker容器使用及文件上下传

7、后台用户管理

8、后台docker镜像管理

9、后台容器管理

11、在线html页面刷新

12、日志记录

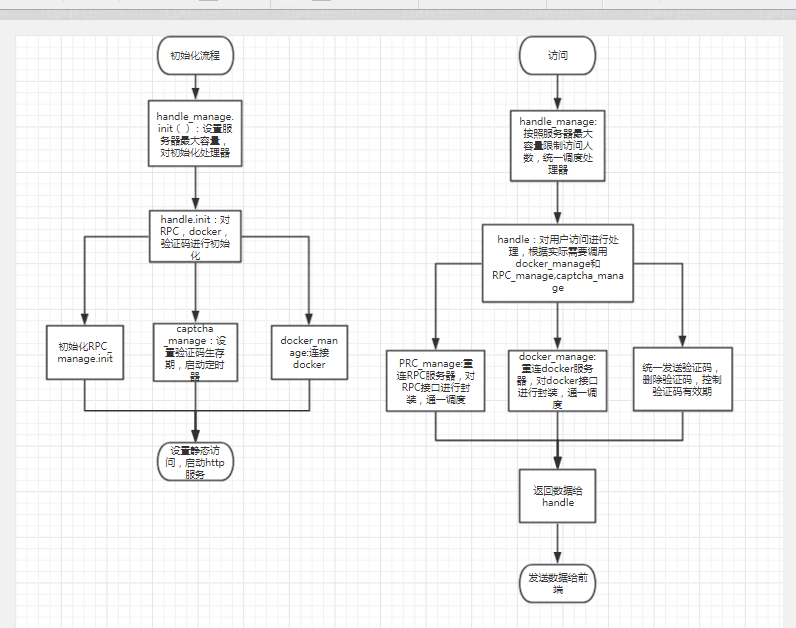
## 系统功能模块详细设计

|  |  |  |  |
| --- | --- | --- | --- |
| **模块编号** | **模块名称** | **需求描述** | **操作流程描述(包括输入和输出)** |
| CloudHub-001 | 登陆 | 用户根据页面提示输入所需信息进行登录 | 用户输入用户Id、用户密码进行登录，反馈用户数据是否符合要求 |
| CloudHub-002 | 注册 | 用户根据页面提示输入所需信息进行注册 | 用户输入用户id、用户密码、重复密码、输入邮箱获取验证码，输入验证码，根据用户输入信息反馈是否正确 |
| CloudHub-003 | 密码找回 | 用户根据页面提示输入所需信息进行密码找回 | 用户输入邮箱获取验证码，输入验证码，若验证码匹配正确，进入修改密码界面，输入密码，二次输入密码确认修改。 |
| CloudHub-004 | 创建容器 | 用户根据提示选择，进行创建 | 用户选择提供的镜像，选择容器的cpu频率和内存容器创建容器 |
| CloudHub-005 | 我的docker | 在该页面提供用户对容器的操作 | 包括：启动、关闭、重启、销毁、重命名、上传/下载文件、获取镜像映射信息等 |

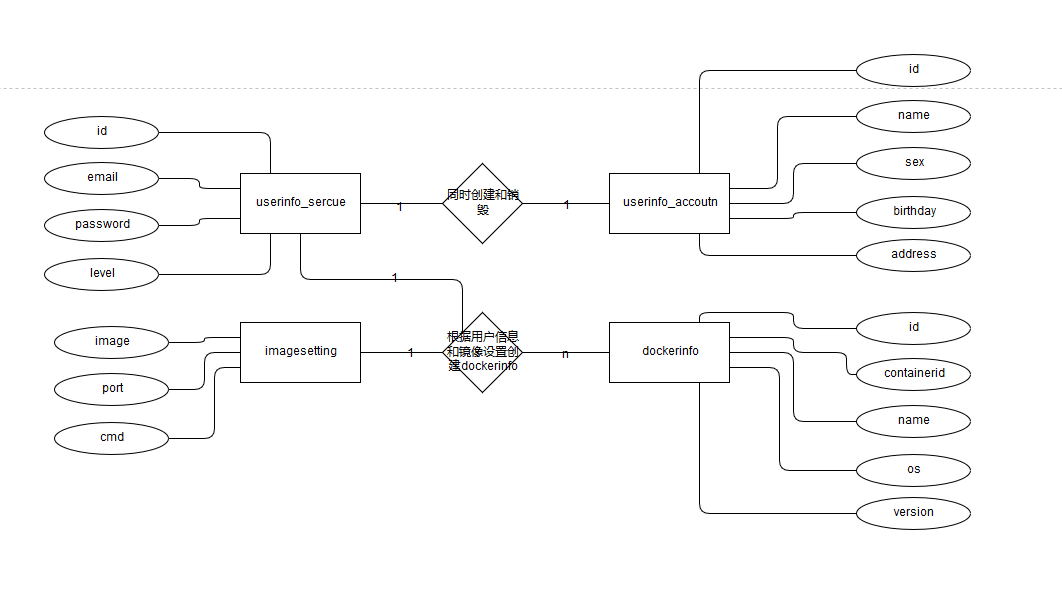
## 系统界面详细设计

### 内部界面设计

中间件：



后端：



### 用户界面设计

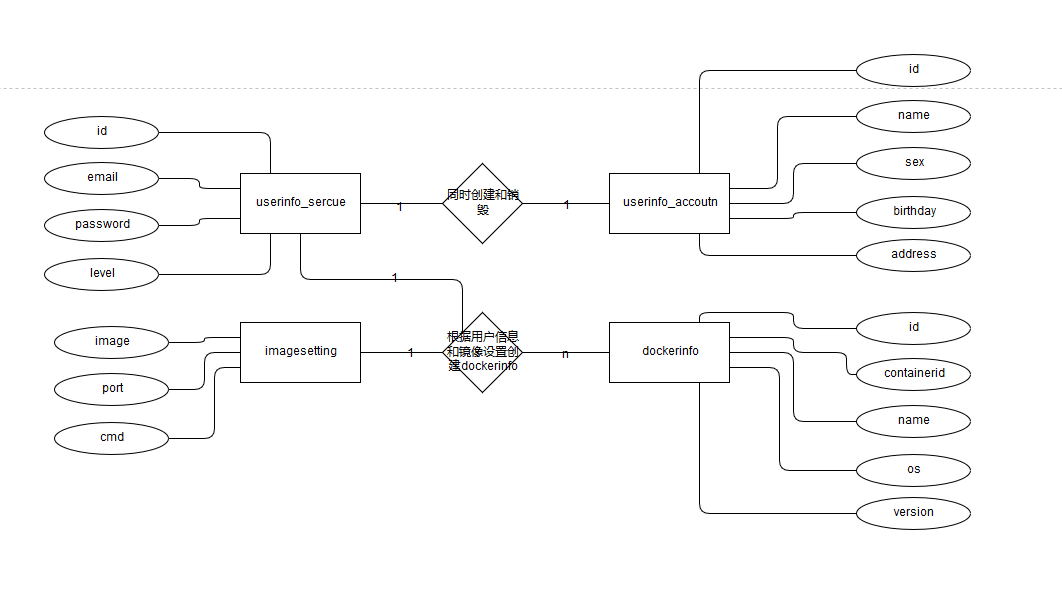
以主流前端设计风格为基准，融入前端个人审美，打造带有独特自身风格的前端界面

# 数据库系统设计

## 设计要求

提供信息的增删改查，防止SQL注入，每个字段的格式要求合理

## 信息模型设计



## 数据库设计

### 设计依据

根据项目需求设计合理表结构

### 数据库逻辑结构

用户注册/登录表：（账号，邮箱，密码，权限）

用户信息表：（账号，昵称，性别，生日，地址）

Docker信息表：（账号，容器，容器名，镜像名称，版本号）

Image\_setting信息表：（镜像名，端口号映射，cmd命令，env环境变量）

### 物理结构设计

用户注册表的id作为用户信息表的外键，删除用户时要先删除其信息，再删除其注册表

### 数据库安全

防SQL注入，隐藏数据库ip

### 数据字典

用户登陆表 userinfo\_secure：

|  |  |  |  |
| --- | --- | --- | --- |
| userinfo\_secure | | | |
| 字段 | 类型 | 约束 | 备注 |
| id | varchar(20) | PRIMARY KEY  NOT NULL | 用户id用于登录 |
| email | varchar(20） | PRIMARY KEY | 邮箱用于密码找回和注册 |
| password | varchar(20) | NOT NULL | 密码 |
| level | int | NOT NULL | 用户权限(0:root 1:admin 2:user) |
| 初始给予一个默认的用户相当于root | | | |

用户信息表 userinfo\_account：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 约束 | 备注 |
| id | varchar(20) | PRIMARY KEY  FOREIGN KEY  NOT NULL | 用户id |
| name | varhchar(20) | NOT NULL | 用户昵称 默认昵称等于id |
| sex | varchar(10) | NOT NULL DEFAULT 0 | 用户性别 |
| birthday | varchar(15) |  | 生日 |
| address | varchar(50) |  | 地址 |

Docker 信息表（用户以及对应的docker）

|  |  |  |  |
| --- | --- | --- | --- |
| user\_info | | | |
| 字段 | 类型 | 约束 | 备注 |
| id | varchar(20) | NOT NULL | Docker所有者的id |
| containerid | Varchar(100) | NOT NULL  PRIMARY KEY | Docker容器的id |
| name | varchar(20) | NOT NULL | 用户申请容器的名字 |
| os | varchar(50) | NOT NULL | 镜像名 |
| version | varchar(20) | NOT NULL | 镜像版本 |

Image\_setting 信息表

|  |  |  |  |
| --- | --- | --- | --- |
| user\_info | | | |
| 字段 | 类型 | 约束 | 备注 |
| image | varchar(20) | NOT NULL  PRIMARY KEY | 镜像的名称 |
| port | Varchar(20) | NOT NULL | 要映射的id |
| cmd | varchar(100) |  | Cmd命令 |
| env | varchar(100) |  | 环境变量 |