CloudHub概要文档

[1 引用 2](#_Toc13665481)

[1.1 目的 2](#_Toc13665482)

[1.2 范围 2](#_Toc13665483)

[1.2.1 系统目标 2](#_Toc13665484)

[1.2.2 主要软件需求 2](#_Toc13665485)

[1.2.3 软件设计约束 3](#_Toc13665486)

[2 数据设计 3](#_Toc13665487)

[2.1 数据对象和进行的数据结构 3](#_Toc13665488)

[3 体系结构 4](#_Toc13665489)

[3.1 程序结构 5](#_Toc13665490)

[3.2 类图 6](#_Toc13665491)

[4 界面设计 7](#_Toc13665492)

[4.1人机界面规约 7](#_Toc13665493)

[4.2人机界面设计规约 8](#_Toc13665494)

[5 接口设计 8](#_Toc13665495)

[5.1 外部接口设计 8](#_Toc13665496)

[5.1.1 外部数据接口 8](#_Toc13665497)

[5.1.2 外部系统或设备接口 8](#_Toc13665498)

[5.2 内部接口设计 8](#_Toc13665499)

[5.2.1 内部接口调用关系 8](#_Toc13665500)

[5.2.2 内部接口约定 9](#_Toc13665501)

[6 模块过程设计 14](#_Toc13665502)

[6.1 登陆功能 14](#_Toc13665503)

[6.2 注册功能 15](#_Toc13665504)

[6.3 密码找回 15](#_Toc13665505)

[6.4 个人中心 16](#_Toc13665506)

[6.5 容器创建 16](#_Toc13665507)

[6.6 容器操作 17](#_Toc13665508)

[6.7 后台管理 17](#_Toc13665509)

[7 测试部分 17](#_Toc13665510)

[7.1 测试方针 17](#_Toc13665511)

[7.2 集成策略 17](#_Toc13665512)

# 1 引用

## 1.1 目的

本文档旨在描述CloudHub平台的概要设计，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。

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。

## 1.2 范围

### 1.2.1 系统目标

开发一个可靠实用的paas平台

### 1.2.2 主要软件需求

该paas平台主要功能需求包括：

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

### 1.2.3 软件设计约束

不可暴露容器远程控制接口，文件上下传接口

# 2 数据设计

## 2.1 数据对象和进行的数据结构

用户：描述用户的基本信息，包括账户id，账户密码，账户email，账户权限，个人资料

type Userinfo\_secure struct {

Id string //用户Id 用于登陆

Email string //用户电子邮箱

Password string //用户密码

Level int //用户权限(0:Root,1:Admin,2:User)

}

type Userinfo\_account struct {

Id string //用户Id 用于登陆

Name string //用户昵称

Sex int //性别(0:保密.1:男性,2:女性)

Birthday string //生日

Address string //地址

}

容器：描述容器的基本信息，包括容器拥有者id，容器id，容器镜像类型

type Dockerinfo struct {

Id string //docker拥有者Id

Containerid string //docker容器的Id

Name string //docker的名字

Os string //镜像的系统类型

Version string //镜像的版本

}

镜像：描述镜像的基本信息，包括镜像名称，镜像设置

type Imagesetting struct {

Image string //镜像的名字

Port string //设置映射的端口

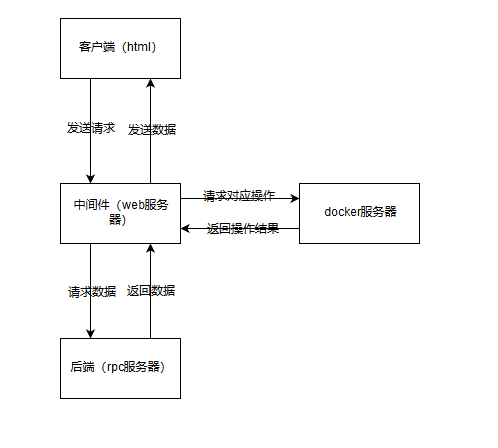
Cmd string //启动时执行的命令

}

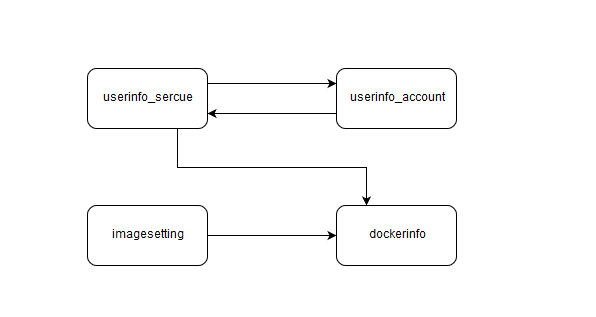
# 3 体系结构

本项目采用胖中间件的三层c/s架构，旨在转移性能瓶颈，提高用户使用体验

项目总体结构图：

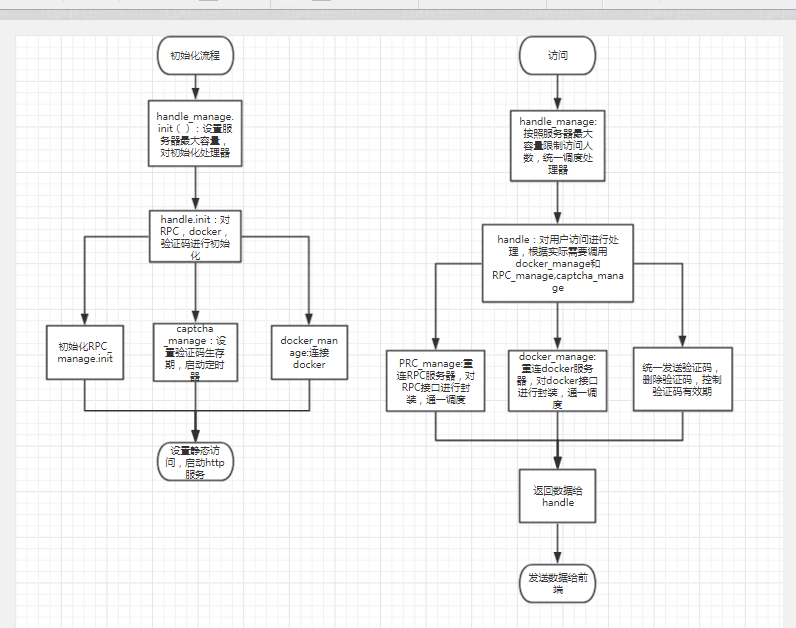


数据库联系图：

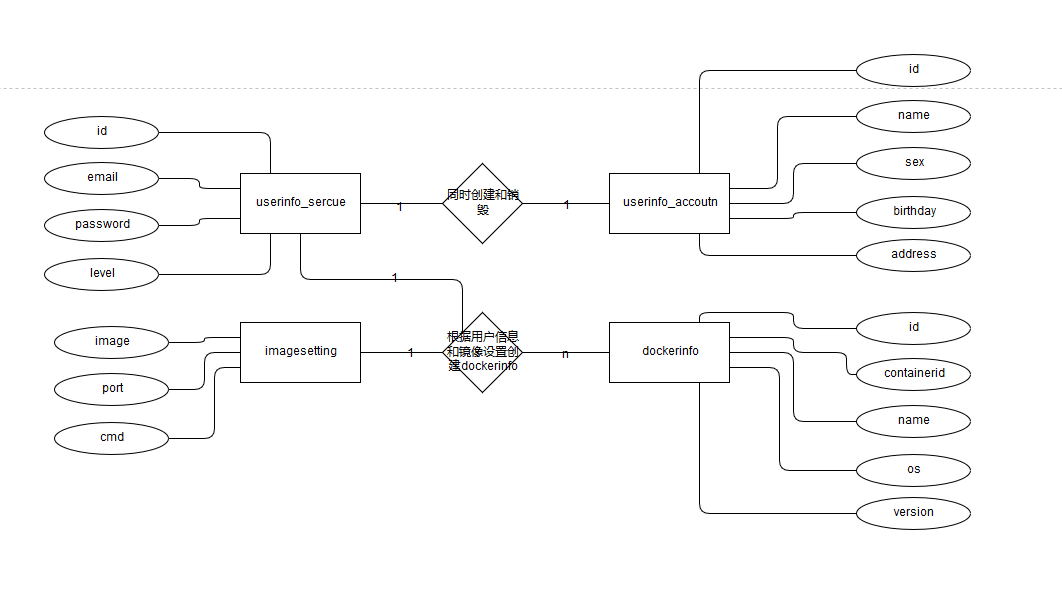


## 3.1 程序结构

中间件：

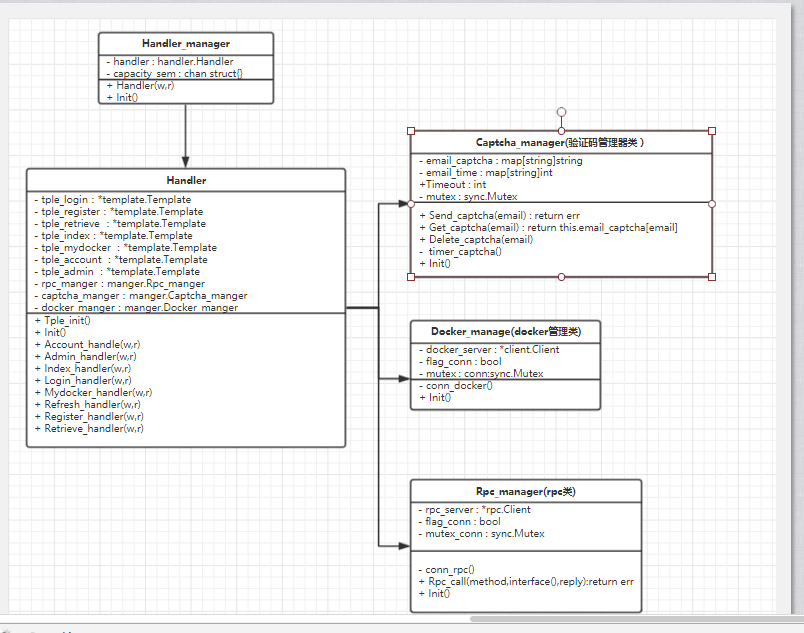


后端：

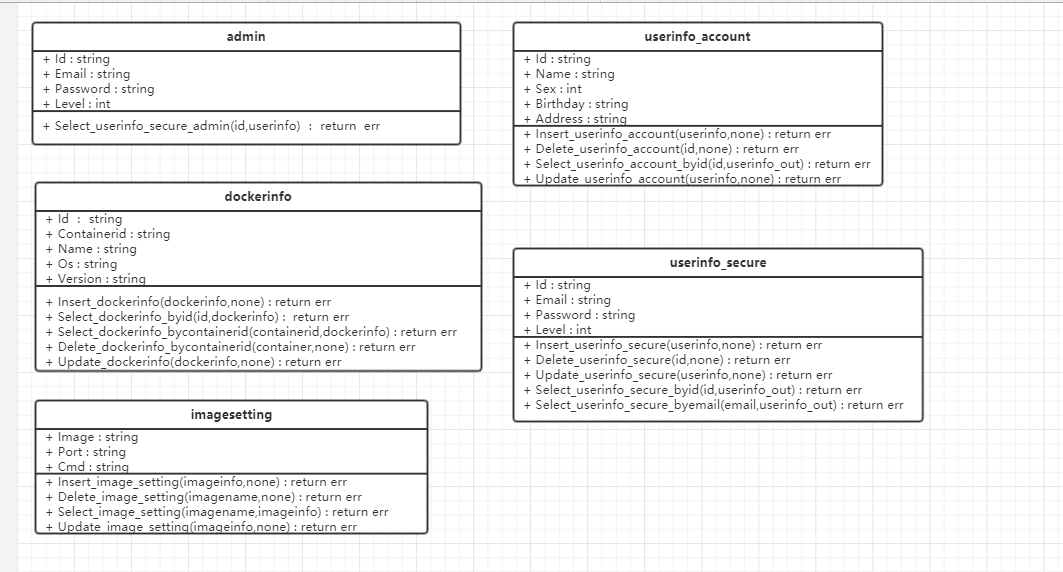


## 3.2 类图

中间件：



后端：



# 4 界面设计

这一部分主要是给出界面设计的总体要求和界面序列，以及界面设计规约。

## 4.1人机界面规约

给出界面风格，约定和操作要求，设计出用户所有界面。

登陆界面：

判断用户id是否大于4位，密码是否大于8位，由字母数字组成。

注册界面：

判断用户id是否大于4位，密码是否大于8位，由字母数字组成，邮箱与验证码是否对应。

账户找回界面：

判断用户邮箱是否与验证码对应，检查密码是否大于8位，由字母数字组成

主页：

无

容器创建界面：

判断用户输入参数是否正确，提交用户输入信息给后台

容器操作界面：

判断用户输入参数是否为空

后台管理界面：

判断用户输入参数是否正确，避免无效的向后台提交数据，对数据进行分页

## 4.2人机界面设计规约

给出界面序列关键，以及每个界面的操作规则和处理规则。

1、注册->登陆（普通用户）->创建容器->操作容器

2、密码找回->登陆（普通用户）->创建容器->操作容器

3、登陆（管理员）->后台管理

4、密码找回->登陆（管理员）->后台管理

5、个人中心->修改密码->登陆

# 5 接口设计

## 5.1 外部接口设计

### 5.1.1 外部数据接口

暂无

### 5.1.2 外部系统或设备接口

显示器使用VGA或HDMI或DVI接口

键盘/鼠标使用ps2或usb接口

## 5.2 内部接口设计

### 5.2.1 内部接口调用关系

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。

### 5.2.2 内部接口约定

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

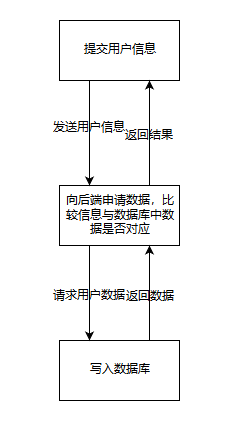
func (s \*Server) Select\_dockerinfo(containerid string, dockerinfo \*Dockerinfo) error

//分页查找

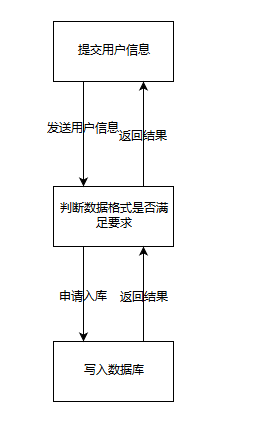
func (s \*Server) List\_dockerinfos(page int, dockerinfo \*[]Dockerinfo) error

# 6 模块过程设计

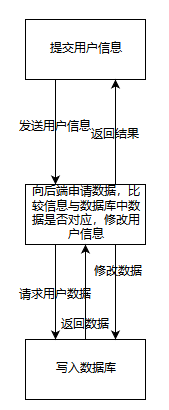
## 6.1 登陆功能



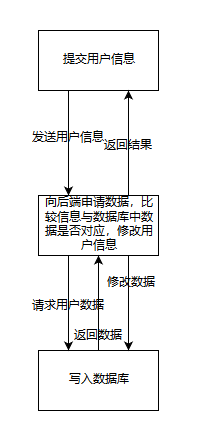
## 6.2 注册功能



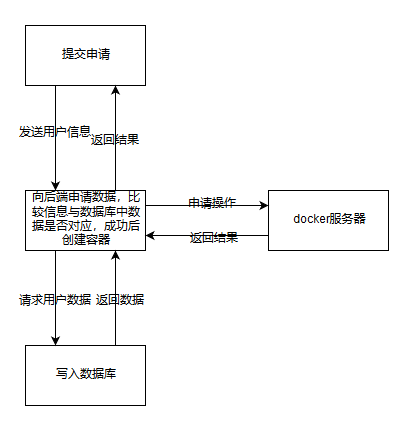
## 6.3 密码找回



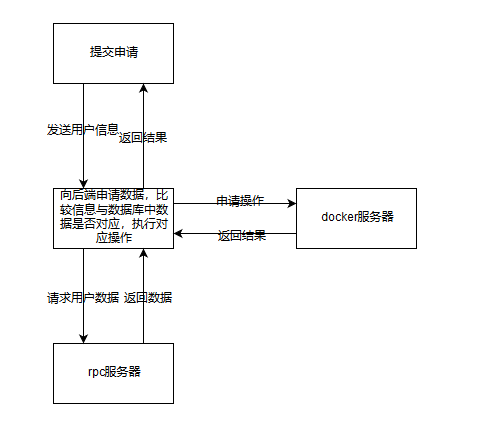
## 6.4 个人中心



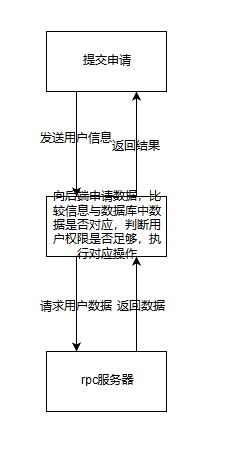
## 6.5 容器创建



## 6.6 容器操作



## 6.7 后台管理



# 7 测试部分

## 7.1 测试方针

首先对规定的正常的功能进行测试，以黑盒测试为主，白盒测试为辅助来设计测试用例，测试用例要尽量覆盖所有的程序功能，记录测试中的问题。

## 7.2 集成策略

采用自底向上的增量式测试策略