|  |  |  |  |
| --- | --- | --- | --- |
| **修订历史** | | | |
| 日期 | 版本 | 说明 | 作者 |
| 7/13/2018 | 1.0 | 第一次迭代测试 | 金瑞洋、王见思、宋逸凡、李翌珺 |
| 7/26/2018 | 1.1 | 第二次迭代测试 |
|  |  |  |
|  |  |  |

**“慧眼识踪“——基于深度学习的人员即时搜寻系统**

**架构设计文档**

版本1.1

目录

[1.引言 1](#_Toc520383066)

[1.1 文档目的 1](#_Toc520383067)

[1.2 名词术语 1](#_Toc520383068)

[2. 需求概述 2](#_Toc520383069)

[2.1. 功能性 2](#_Toc520383070)

[2.1.1 Use case图及说明 2](#_Toc520383071)

[2.1.2“查看摄像头参数”用例规约 4](#_Toc520383072)

[2.1.3 查看实时监控”用例规约 5](#_Toc520383073)

[2.1.4“查看历史监控视频”用例规约 5](#_Toc520383074)

[2.1.5“根据图片搜索目标”用例规约 6](#_Toc520383075)

[2.2. 易用性 6](#_Toc520383076)

[2.3. 可靠性 7](#_Toc520383077)

[2.3.6 错误或缺陷率： 7](#_Toc520383078)

[2.4. 性能 7](#_Toc520383079)

[2.5 可支持性 7](#_Toc520383080)

[3.架构设计思想 8](#_Toc520383081)

[4. 架构体系描述 8](#_Toc520383082)

[5. 架构图 9](#_Toc520383083)

[5.1物理视图 9](#_Toc520383084)

[其中对Nginx静态服务与Tomcat动态服务做负载均衡: 10](#_Toc520383085)

[5.2逻辑视图 10](#_Toc520383086)

[6.技术选型 10](#_Toc520383087)

[7.过程视图 12](#_Toc520383088)

[7.1 通信协议 12](#_Toc520383089)

[7.2 时序图 12](#_Toc520383090)

[8.数据视图 13](#_Toc520383091)

[9 接口设计 13](#_Toc520383092)

[9.1 动态资源服务接口 13](#_Toc520383093)

[9.1.1 摄像头 14](#_Toc520383094)

[9.1.2 地图 14](#_Toc520383095)

[9.1.3 历史视频文件 15](#_Toc520383096)

[9.2 视频直播服务 15](#_Toc520383097)

[9.2.1 视频推流 15](#_Toc520383098)

[9.2.2 视频播流 16](#_Toc520383099)

[9.3 静态资源服务 16](#_Toc520383100)

[9.3.1 静态网页 16](#_Toc520383101)

[9.3.2 视频文件 16](#_Toc520383102)

[9.4 识别服务 16](#_Toc520383103)

[9.4.1 从直播中进行搜索 16](#_Toc520383104)

[9.4.2 从历史视频中进行搜索 17](#_Toc520383105)

[10 部署运维 17](#_Toc520383106)

[11 遗留问题 18](#_Toc520383107)

# 1.引言

本项目针对寻找走失人员、跟踪犯罪嫌疑人和定位传染病人等众多现实场景下都必须要解决的搜寻指定人员的问题，设计并开发了一个“基于深度学习的人员即时搜寻系统”。

## 1.1 文档目的

本文档是慧眼识踪（GETS）系统进行详细设计和编码的重要依据。对于整体系统结构关系进行详细描述，阐述系统总体框架，说明了系统中所采取的相关技术，并对于相关内容作出一定规定。为开发人员今后设计、编码、测试提供了可参考模板。

## 1.2 名词术语

搜索目标：需要搜寻系统根据监控画面锁定位置的人物

GETS：God Eye Tracking System， “慧眼识踪”人员搜寻系统

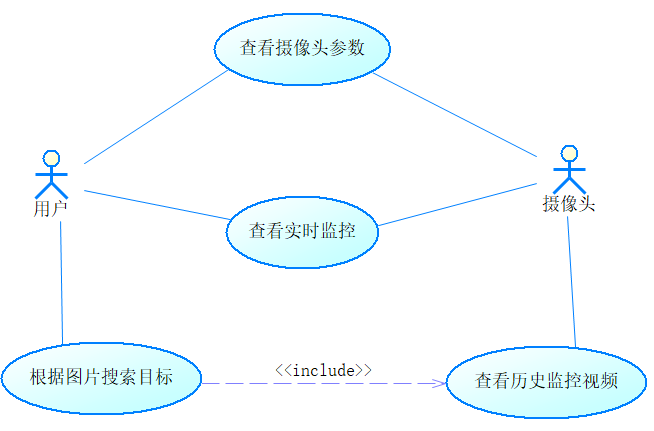
MTBF：Mean Time Between Failure，平均失效间隔时间

MTTR：Mean Time To Restoration，平均恢复时间

# 2. 需求概述

## 2.1. 功能性

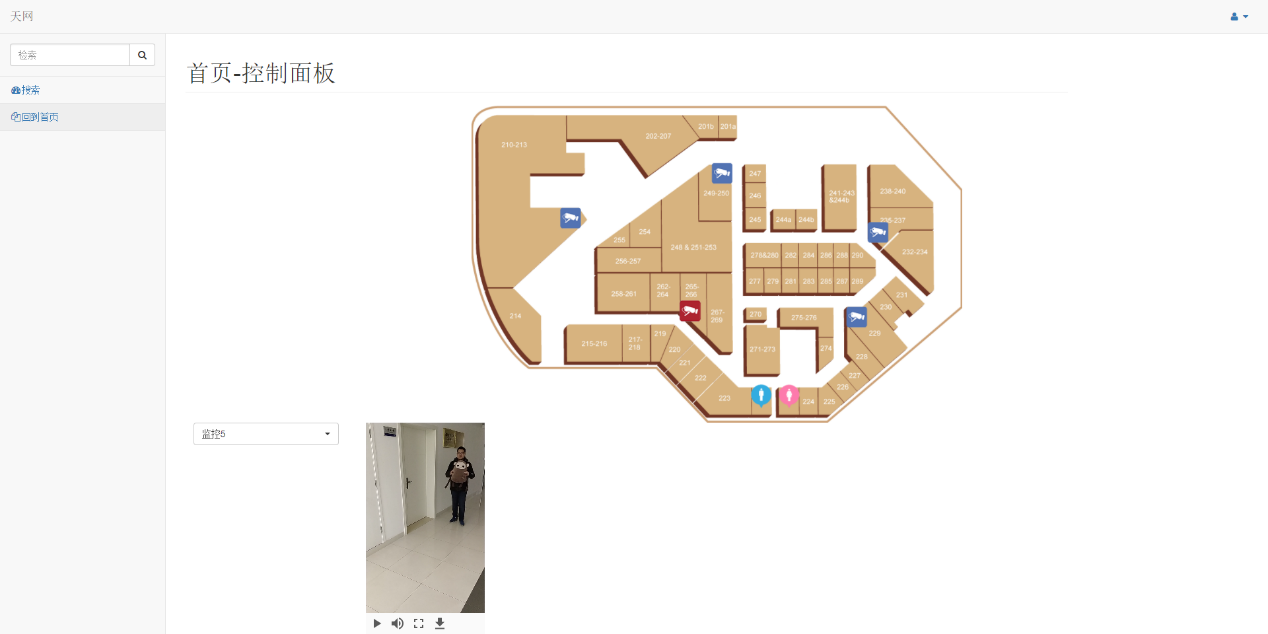
### 2.1.1 Use case图及说明



查看摄像头参数：允许用户查看某建筑物平面图，并查看摄像头位置以及相关参数（方向、俯角等），同时平面图自动显示摄像头覆盖位置。

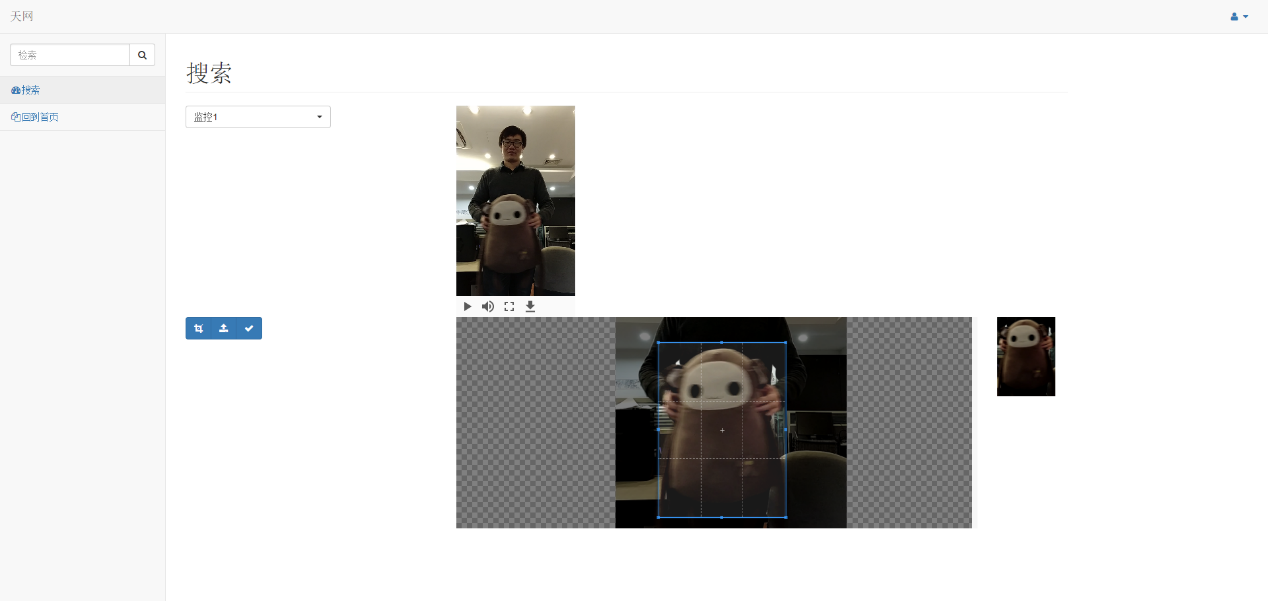


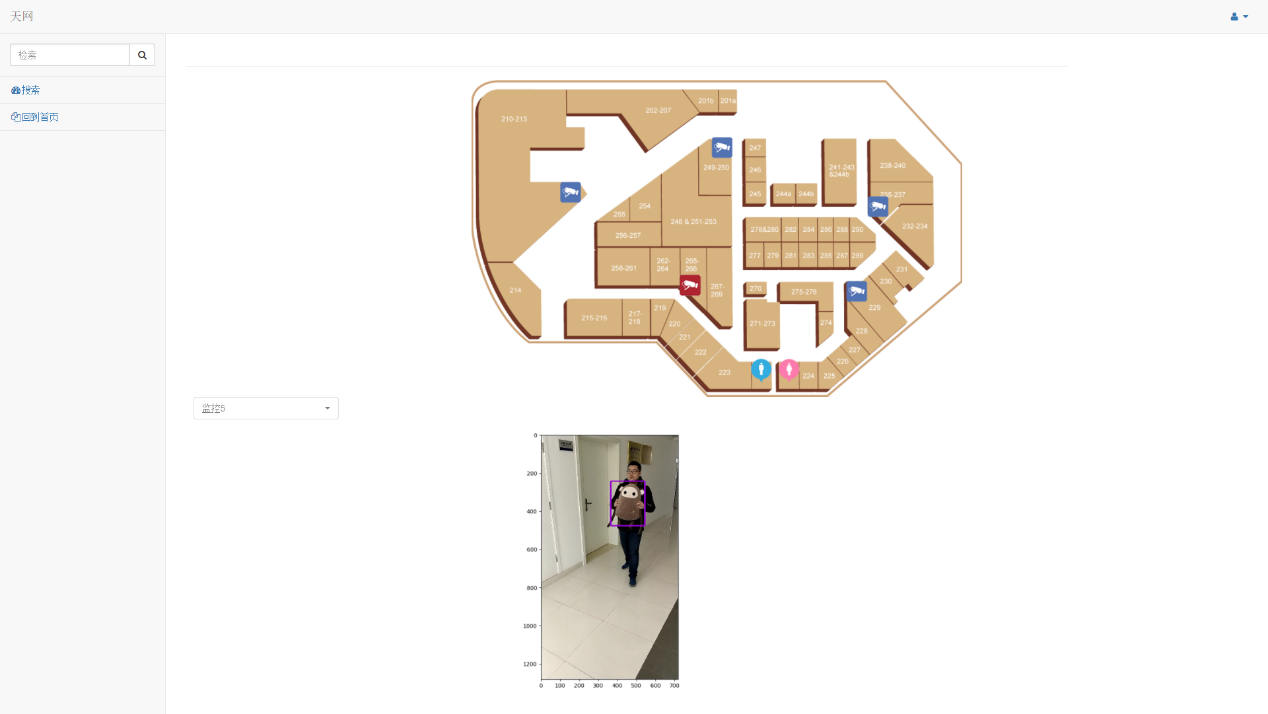
查看实时监控：在打开平面图以后，用户可以通过在地图中点击摄像头，使被选中的摄像头高亮显示，并查看该摄像头的实时视频。



查看历史监控视频：选取某个摄像头，可以播放其历史监控视频。

根据图片搜索目标：用户可以在历史视频中定格并框选得到目标对象（如下图所示，通过在某张截图中框选玩偶作为搜索目标），也可以自行上传图片并框选出目标对象。系统会根据搜索目标出现的位置和行进速度，计算出可能的范围，对范围内的摄像头的视频进行搜索，找出搜索目标的位置，如下图所示：





### 2.1.2“查看摄像头参数”用例规约

描述：用户进入后即可在首页查看全区域平面图与摄像头信息。

执行者：用户

基本流：

1. 系统显示建筑物平面图、摄像头位置及摄像头覆盖位置

2. 用户将鼠标移至某一摄像头上方

3. 系统显示该摄像头相关参数、位置信息并高亮其覆盖位置

备选流：

2a. 用户查看平面图后直接登出

用例结束

3a. 该摄像头数据无法获取

系统提示“该摄像头数据异常”，返回1

非功能需求：系统响应用户需求时间不超过3秒

业务规则：3a. 系统同时最多只能显示一个摄像头的详细信息（相关参数与位置信息）

### 2.1.3 查看实时监控”用例规约

描述：用户通过在前端页面显示的地图中点击摄像头，被选中的摄像头高亮显示，并给出摄像头的实时视频

执行者：用户

基本流：

1. 用户浏览界面并选择摄像头，单击选中的摄像头

2. 显示对应摄像头的实时视频，被选中的摄像头高亮，前端页面进行调整

备选流：

1a. 用户没有选中摄像头

返回1

2a. 对应摄像头数据调取失败

系统提示“该摄像头数据异常”，返回1

2b. 用户选中其他摄像头

返回1

前置条件：系统位于首页

后置条件：若正常反馈则显示摄像头的实时视频

非功能需求：系统响应的时间不超过3秒

业务规则：2a. 同时最多只能显示一个摄像头的画面

### 2.1.4“查看历史监控视频”用例规约

描述：用户可以在选取某个摄像头后播放其历史数据

执行者：用户

基本流：

1. 用户单击选中摄像头

2. 系统高亮选中摄像头

3. 用户点击“查看历史视频”

4. 系统播放该摄像头的历史视频

备选流

2a. 用户点击其它摄像头

系统切换到选中摄像头，回到2

4a. 对应摄像头数据调取失败

系统提示“该摄像头数据异常”，返回1

前置条件：系统在查看平面图界面

后置条件：用户可在观看视频时任意暂停、继续、快进、快退

非功能需求：系统响应的时间不超过3秒

业务规则：3a. 用户仅能查看距当前时间一周以内的历史视频

### 2.1.5“根据图片搜索目标”用例规约

描述：用户可以在摄像头的历史数据或自行上传的图片中选定搜索目标，根据用户输入图片，系统在历史视频以及直播视频流中搜索，并返回可能位置。

执行者：用户

基本流：

1. 用户调用“查看历史数据”用例，观看该摄像头历史视频并选择一帧暂停

2. 用户点击“截取”按钮，在视频上截图

3. 系统展示被截取图像，询问是否确认上传

4. 用户选择确认上传

5. 系统显示选择成功

6. 用户选择单层搜索

7. 根据用户上传图片，返回对象可能位置显示对应摄像头的实时视频，被选中的摄像头高亮，前端页面进行调整

备选流：

1a. 用户点击“本地上传图片”

系统在截取图像界面显示用户选择的图片，进入2

1b. 用户没有选择暂停，视频播放完毕

用例结束

2a. 用户选择放弃

用例结束

6a. 用户选择多层搜索

系统根据图片找到对象在所有楼层的可能位置，返回对应摄像头的实时视频，并高亮被选中摄像头

7a. 系统无法找到任何可能位置

系统提示“找不到目标”，用例结束

前置条件：系统位于首页

后置条件：能返回对象可能位置

业务规则：7a. 可显示任意数量的摄像头视频

## 2.2. 易用性

系统应方便用户使用，有基础计算机水平的用户学习时间应不超过5分钟；同时系统提供在线支持帮助，以方便用户使用该系统。一个学习过的用户应该可以在平均3分钟或最多5分钟时间以内完成人员搜索操作（系统搜索时间不计）。

## 2.3. 可靠性

**2.3.1 可用性：**系统必须保证每天24小时不间断运行，年平均正常运行时间达到99.5%，同时应当正确处理发生的异常或错误，并返回错误信息

**2.3.2 MTBF：**约一个月

**2.3.3 MTTR：**约一小时

**2.3.4 精确度：**系统视频输出视频分辨率720p，识别准确率在70%以上

**2.3.5 最高错误或缺陷率：**不大于60bugs/KLOC

### 2.3.6 错误或缺陷率：

小错误：指页面显示异常，网页的某部分无法显示。错误率<10%。

大错误：指网页对客户操作无响应。错误率<3%.

严重错误：指系统崩溃、数据库崩溃等令整个软件无法工作的错误。错误率应为0。

## 2.4. 性能

**2.4.1 响应时间：**响应时间不超过10s

**2.4.2 吞吐量：**考虑到服务器性能，系统应支持10个并发用户同时搜索，其余搜索请求将加入等待队列

**2.4.3 容量：**系统最多可支持100名用户同时在线

**2.4.4 降级模式：**系统降级运行时，可查看实时监控和历史记录，无法进行搜索

**2.4.5 资源利用情况：**系统应有至少16GB内存与1TB硬盘空间可供使用，同时应与监控系统、数据库等外界系统保持通信，通信速度应不低于1MB/s

## 2.5 可支持性

**2.5.1 编码标准：**系统编码使用utf-8字符集，系统语言使用中文

**2.5.2 命名约定：**变量命名使用驼峰式命名法

# 3.架构设计思想

B/S模式，统一客户端，简化系统开发、维护和使用。

系统整体采用微服务模式。将一体化应用分解、模块化，变成多个单体应用，易于理解、开发与维护；项目组内部不同服务的负责人可以自由选择技术，最终只需提供对外的api接口；每个服务独立部署，利用docker、kubernetes，在云端可以轻松实现集群部署。

后端系统采用分层模式。面向接口的编码，以之为根据进行详细设计，实现各层之间的解耦。

# 4. 架构体系描述

GETS系统主要分为以下几个服务：视频直播流转发服务、动态资源服务、识别服务、静态资源服务。每个服务暴露api进行通信。

其中后端分层分为：数据访问层、业务层、控制层。通过分层，定义层与层之间的接口，进行层级之间的解耦，增强可扩展性、可维护性，降低集成难度，提高效率。

# 5. 架构图

## 5.1物理视图structure-2.0.png

## 

## 其中对Nginx静态服务与Tomcat动态服务做负载均衡:pasted-image.png

## 5.2逻辑视图分层.png

# 6.技术选型

- 技术选择

- 前端： React.js

- 后端： Java, Springboot

- 识别服务： Python, TensorFlow, Flask

- 直播服务： Nginx-rtmp

- 数据库： Mysql

* 集群管理：Rancher
* CI/CD: Rancher Pipeline

- 版本管理： github

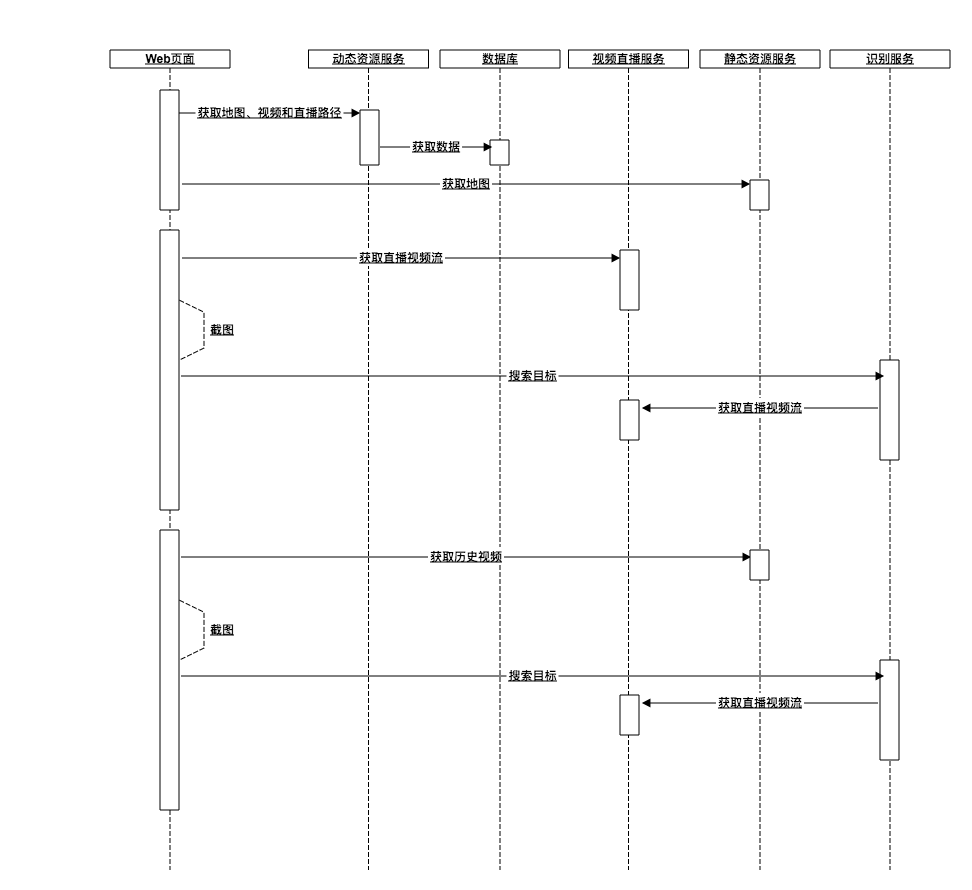
- 运行环境： Chrome

# 7.过程视图

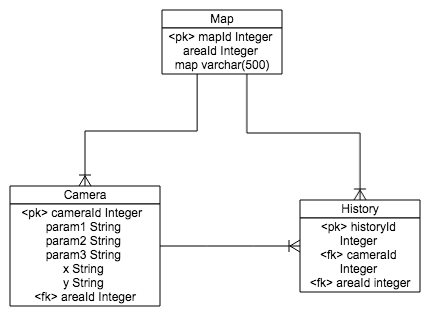
## 7.1 通信协议

大部分服务以HTTP协议暴露 api，视频直播传输使用标准rtmp协议

## 7.2 时序图



# 8.数据视图

类图

# 9 接口设计

## 9.1 动态资源服务接口

• 协议: HTTP

• port: 8081 (default)

• 返回JSON对象，JSON对象格式如数据视图-类图中所示

### 9.1.1 摄像头

获取摄像头信息

获取全部摄像头信息

url: /api/camera/all

method: GET

获取指定地区摄像头信息

url: /api/camera

method: GET

query parameters

areaid - 地区id

保存/修改摄像头信息

url: /api/camera/save

method: POST

request body parameters

Camera JSON 对象

cameraid - 摄像头id（新建摄像头则将该参数设置为0）

param1 - 参数一

param2 - 参数二

param3 - 参数三

x - 在指定地区x方向的百分比坐标

y - 在指定地区y方向的百分比坐标

areaid - 地区id

删除摄像头信息

url: /api/camera/delete

method: DELETE

request body parameters

cameraid - 需要删除的摄像头id

### 9.1.2 地图

获取指定地图信息

url: /api/map

method: GET

query parameters

areaid - 地区id

保存/修改地图信息

url: /api/map/save

method: POST

request body parameters

Map JSON 对象

areaid - 地区id

map - 地图的base64编码

删除地图信息

url: /api/map/delete

method: DELETE

request body parameters

areaid - 地区id

### 9.1.3 历史视频文件

获取历史信息

获取所有历史信息

url: /api/history/all

method: GET

获取指定地区历史信息

url: /api/history

method: GET

query parameters

areaid - 地区id

保存/修改历史文件信息

url: /api/history/save

method: POST

request body parameter

History JSON 对象

historyid - 历史文件id （新建历史文件信息则该参数置0）

cameraid - 摄像头id

areaid - 地区id

filename - 历史文件文件名

删除历史文件信息

url: /api/history/delete

method: DELETE

request body parameters

history

## 9.2 视频直播服务

### 9.2.1 视频推流

• 协议: RTMP

• url: /stream/{$STREAM\_NAME}

• port: 1935 (default)

example: rtmp://127.0.0.1:1935/stream/hello

### 9.2.2 视频播流

• 协议: HLS

• url: live/{$STREAM\_NAME}.m3u8

• port: 8080 (default)

example: <http://127.0.0.1:8080/live/hello.m3u8>

虽然HLS本身实现会导致20s左右延迟，但是其他播放协议均需要flash支持，而HLS只需要html5即可实现视频转发。苹果移动设备不支持flash。

## 9.3 静态资源服务

• 协议: HTTP

• port: 80 (default)

### 9.3.1 静态网页

• url: /

### 9.3.2 视频文件

• url: /vod/{$VIDEO\_NAME}

## 9.4 识别服务

• 协议: HTTP

* port：5000(default)

### 9.4.1 从直播中进行搜索

• url: /stream

* method: POST
* request body parameters

img – 截图的base64编码

• return: JSON

picture - 搜索结果的JPG图片base64编码（无编码头）

### 9.4.2 从历史视频中进行搜索

• url: /history

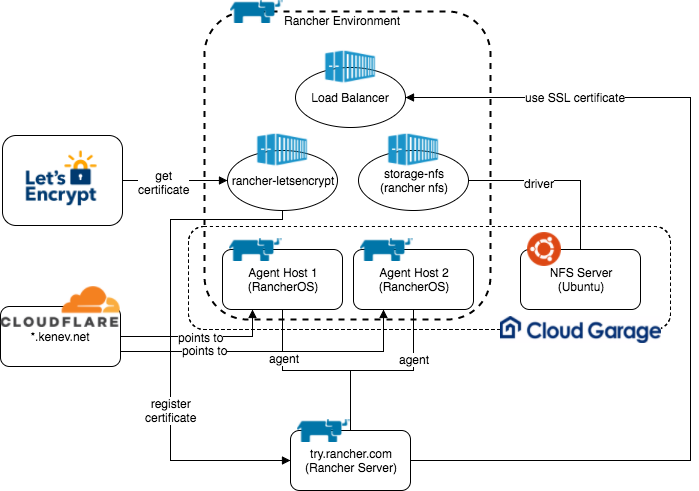
* method: POST
* request body parameters

img – 截图的base64编码

• return: JSON

picture - 搜索结果的JPG图片base64编码（无编码头）

# 10 部署运维



利用rancher做集群管理。

一台服务器作为rancher 的master(rancher server)节点监控、管理其他节点。另外一台服务器作为worker节点部署应用。

每一个worker节点(rancher agent)自带healhcheck, ipsec, network-services, scheduler等功能。

在rancher主界面，直接导入docker-compose.yml, rancher-compose.yml 即可生成某一服务的复数镜像以及负载均衡。

利用rancher pipeline 实现CI/CD 全自动流水线，并且不间断更新运行中的镜像。

静态资源服务器采用Nginx，动态资源服务器采用Tomcat，视频转发服务器采用Nginx-rtmp

# 11 遗留问题

视频转发负载均衡 - 若视频发往容器A，必须从容器A出。而rancher自带的HAproxy负载均衡，视频发往容器A，并不能保证用户获取视频时从容器A获取。

性能需求 - 实际并没有高性能服务器提供。个人服务器只有2GB内存，同时2个摄像头进行直播，并且数人同时访问网页观看直播时，其他服务也都在该服务器上跑，CPU使用率即会达到99%，服务器卡死。因此性能需求极大程度上受到服务器性能限制。。

kubernates集群 - 由于没有那么多节点，并且学生机服务器内存不足无法作为k8s master节点，因此无法采用kubernates集群，只能使用rancher默认的cattle

CI/CD – 受服务器网络带宽所限，在git fetch以及docker push这两步会花费大量时间，有时甚至在git fetch时失败。