以下所有逻辑仅考虑单机版设备的情况，暂时不考虑联网版设备的情况

# 人员导入逻辑

## 要求

人员导入包括人员和特征

人员和特征成功导入后，数据库中的人员和特征均更新到新数据

设备使用的数据切换到导入的数据（内存管理）

如果导入中间出现问题，则仍使用原有的数据

能提供导入进度

导入人员的文件格式为逗号分隔的csv文件，第一行为行头，内容为：

用户uuid, 姓名, 用户id, 管理员, 性别, 卡号, 身份证号码, 保留, 职业,常用住址, 用户密码

如果某一行格式不正确，通知主线程发生错误，导入终止

导入特征的文件格式为sqlite，数据表中至少包含以下字段：

irisdataid, personid, groupid, iricode, devsn, eyeflag, regtime

如果某一行数据的personid在导入的人员表中没有，通知主线程发生错误，导入终止

## 导入进度估算方式

一条人员数据，如果全部都是最大允许长度，约占450字节，估算时按照一条人员数据512字节估算；

一条虹膜数据，如果全部都是最大允许长度，约占1136字节，估算时按照一条虹膜数据1152字节估算；

这样一条虹膜数据的字节数为人员数据的2.25倍。

所以，在计算导入进度时，一条人员数据加权为1，一条虹膜数据加权为2.25。

例如，人员数据总条数为PN，虹膜数据总条数为IN，此时已导入px条人员数据，ix条虹膜数据，则导入百分比为：

## 问题

导入完成后，需要更新内存中的数据，怎么控制？在后台控制批量更新，不算为导入时间

虹膜特征保存在数据库中做不做BASE64的变换？

# 特征管理逻辑

## 数据结构

### 人员信息结构

|  |  |  |
| --- | --- | --- |
| typedef struct \_\_PersonInfo  {  uuid\_t Id;  uuid\_t FeatureID;  std::string CardID; //卡号  std::string WorkSn; //工号  std::string Name;  std::string Depart;  std::string Sex;  }PeopleInfo; | | |
| 数据 | 类型 | 说明 |
| Id | uuid\_t | 人员ID，UUID，唯一 |
| FeatureID | uuid\_t | 特征ID，UUID |
| CardID | std::string | 卡号 |
| WorkSn | std::string | 工号 |
| Name | std::string | 姓名 |
| Depart | std::string | 部门名称 |
| Sex | std::string | 性别 |

### 增加特征的数据结构

|  |  |  |
| --- | --- | --- |
| typedef struct \_\_InfoData  {  uuid\_t ud;  IdentifyType EyeFlag;  unsigned char FeatureData[g\_FeatureSize];  PeopleInfo PeopleData;  }InfoData,\*PInfoData; | | |
| 数据 | 类型 | 说明 |
| ud | uuid\_t | 特征ID，UUID，唯一 |
| EyeFlag | IdentifyType | 标志特征是左眼还是右眼  1：左眼；2：右眼 |
| FeatureData | unsigned char | 特征，长度根据使用的算法决定 |
| PeopleData | PeopleInfo | 人员信息 |
|  |  |  |
|  |  |  |

### 内部人员节点结构

|  |  |  |
| --- | --- | --- |
| typedef struct \_\_PeopleNode  {  uuid\_t PeopleID; //人员ID  std::string Name;  std::string Sex;  std::string WorkSn;  std::string CardID;  std::string Depart;  std::list<GUID> LeftFeature; //右眼特征ID数组  std::list<GUID> RightFeature; //左眼特征ID数组  }PeopleNode,\*PPeopleNode; | | |
| 数据 | 类型 | 说明 |
| PeopleID | uuid\_t | 人员ID，UUID，唯一 |
| Name | std::string | 姓名 |
| Sex | std::string | 性别 |
| WorkSn | std::string | 工号 |
| CardID | std::string | 卡号 |
| Depart | std::string | 部门名称 |
| LeftFeature | std::list<GUID> | 左眼特征ID链表，此人所有的左眼特征ID放在这个列表中 |
| RightFeature | std::list<GUID> | 右眼特征ID链表，此人所有的右眼特征ID放在这个列表中 |

### 人员ID和人员信息映射

|  |  |  |
| --- | --- | --- |
| typedef std::map<std::string, std::shared\_ptr<PeopleNode> > PeopleMap;  通过这个映射，通过人员ID可以迅速查找到人员信息，并且找到此人所有的特征ID链表  适用于通过人员ID删除特征时，找到需要删除的特征  由于使用了映射的方式，每个人员在这个映射中只能出现一次 | | |
| 名称 | 类型 | 说明 |
| 键 | std::string | 人员ID |
| 键值 | std::shared\_ptr<PeopleNode> | 保存人员信息，智能指针形式，数据类型为[PeopleNode](#_内部人员节点结构)。  只要这个人存在，人的信息总是保存在这个映射中 |

### 内部控制节点（特征和人员对应关系）结构

|  |  |  |
| --- | --- | --- |
| typedef struct \_\_ControlNode  {  std::shared\_ptr<PeopleNode> p;  uuid\_t FeatureID;  }ControlNode,\*PControlNode;  中间控制数据结构体：目的是解决多个特征对应一个人员时不重复的加载人员空间 | | |
| 数据 | 类型 | 说明 |
| p | std::shared\_ptr<PeopleNode> | [人员信息](#人员节点)，智能指针形式，数据类型为[PeopleNode](#_内部人员节点结构)。有特征时，每条特征都对应着一个这样的节点。由于保存的是指针，所以不会耗费大量的内存。 |
| FeatureID | uuid\_t |  |

# 舵机控制逻辑

## 原理

通过设置PWM占空比控制舵机角度。

/sys/class/pwm/pwmchip1/pwm0目录下

设置周期：echo 20000000 > period

设置角度：echo 1500000 > duty\_cycle

1500000对应的角度为90度

手动控制方式可参见“06Other\舵机相关说明”目录下的文档。

## 过程

1、眼睛预置的合适位置，Y方向的值为325

相机高度方向视场，预置值为67 (g\_cameraFOVHeight)

人脸图像采集高度，目前为720 (ov5640\_g\_height)

2、调用detectTracking(img.bits(), img.height(), img.width(), channel, rect, m\_Info)

检测眼睛位置

（dectectTracking在libKCFTracking.so中实现）

调用detectTracking前，获取一次舵机的当前角度，在检测到结果后，设置舵机运动角度时使用这个当前角度。因为考虑到检测速度和舵机运动，有可能检测到结果后的舵机位置已经和检测前的位置不一样了，所以先记录一次舵机位置。

3、输出参数m\_Info.flag==true，表示检测到人脸

计算当前眼睛位置Y方向的值和预置的合适位置

像素差=info.browCY-325

计算转动相对角度

offsetY = 0.0f – 像素差\*相机高度方向视场角/图像高度/1.9f

（其中除以1.9f，是为了减小舵机运动的震荡）

4、转动相对角度+调用detectTracking前获取的舵机角度=本次要设置的舵机角度

5、如果连续30幅人脸图像中都没有检测到人脸，则使舵机归位到初始位置（90度）