[DLOGDB详细设计]

Software Design Specification

Revision: 0.1

**Revision Date**: [date of revision]

2023-8-6

Author/Editor(s):

[刘彦涛]

To Be Approved By:

[段翰聪]

目录

[1.dlogdb 3](#_Toc143441815)

[1.1 文档版本 3](#_Toc143441816)

[1.2 概述 3](#_Toc143441817)

[1.3 dlogdb设计 4](#_Toc143441818)

[1.3.1 整体框架 4](#_Toc143441819)

[1.3.2 模块划分 5](#_Toc143441820)

[2.模块详细设计 6](#_Toc143441821)

[2.1 通信模块 6](#_Toc143441822)

[2.1.1 模块功能描述 6](#_Toc143441823)

[2.1.2 模块详细设计 6](#_Toc143441824)

[2.2 线程池模块 9](#_Toc143441825)

[2.2.1 模块功能描述 9](#_Toc143441826)

[2.2.2 模块详细设计 9](#_Toc143441827)

[2.2.3 类详细设计 9](#_Toc143441828)

[2.3 存储模块 10](#_Toc143441829)

[2.3.1 模块功能描述 10](#_Toc143441830)

[2.3.2 模块详细设计 11](#_Toc143441831)

[2.3.3 类详细设计 13](#_Toc143441832)

[3.数据挖掘 15](#_Toc143441833)

[3.1 目标轨迹生成 15](#_Toc143441834)

[3.1.1 功能描述 15](#_Toc143441835)

[3.1.2 输入输出 15](#_Toc143441836)

[3.1.3 实现细节 15](#_Toc143441837)

[3.1.4 接口 15](#_Toc143441838)

[3.2 频次分析 17](#_Toc143441839)

[3.2.1 功能描述 17](#_Toc143441840)

[3.2.2 输入输出 17](#_Toc143441841)

[3.2.3 实现细节 17](#_Toc143441842)

[3.3 落脚点分析 17](#_Toc143441843)

[3.3.1 功能描述 17](#_Toc143441844)

[3.3.2 输入输出 18](#_Toc143441845)

[3.3.3 实现细节 18](#_Toc143441846)

[3.3.4 接口 18](#_Toc143441847)

[3.4 同行分析 19](#_Toc143441848)

[3.4.1 功能描述 19](#_Toc143441849)

[3.4.2 输入输出 20](#_Toc143441850)

[3.4.3 实现细节 20](#_Toc143441851)

[3.4.4 接口 20](#_Toc143441852)

[3.5 伴随分析 22](#_Toc143441853)

[3.5.1 确定性伴随分析 22](#_Toc143441854)

[3.5.2 非确定性伴随分析 22](#_Toc143441855)

# dlogdb

## 文档版本

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Revision Date** | **Editor(s)** | **Changes** |
| Initial Version | 2023.8.20 | 刘彦涛 |  |

## 概述

dlogdb是其中的子系统，主要负责目标的轨迹挖掘，处理具体的业务逻辑，它提供的主要功能如下：

负责与相关外部模块通信，包括 videogateway、app。

具体任务的执行，包括目标的轨迹挖掘、频次分析、落脚点分析、伴随分析;

## dlogdb设计

### 整体框架



图1-1 整体框架



图1-2 Dlogdb系统时序图

4

### 模块划分

dlogdb主要包括通信模块、线程池、存储模块和数据挖掘。如图所示

图1-2

通信模块负责与外部模块通信，线程池模块负责具体执行具体的Task，存储模块负责实际存储和管理数据。

通信模块由muduo负责接收来自app、videogateway的网络通信，并根据cmd生成对应的Task，并将其放入工作线程池；工作线程池负责进行业务处理。存储模块负责存储发送来的数据包括任务id、摄像头id、抓拍记录等，数据挖掘模块的是具体的业务实现。

# 模块详细设计

## 通信模块

### 模块功能描述

通信模块的主要功能是负责与多种通信实体进行网络通信。通信模块接收外部模块发来的请求，根据任务类型，生成对应的Task，并由Task推动任务执行，在任务完成后，通信模块负责将数据返回给外部模块。

### 模块详细设计

#### 消息

消息表示用于通信的数据，采用报头（Header）+负载（body）格式进行数据通信。

消息类型分为以下两类：

req：请求消息，由客户端发送到服务器。

ack：回应消息，由服务器发送到客户端，用于回应客户端的请求。

通信数据格式为protobuffer。protobuffer的message定义见下文及附录。

#### 传输

客户端与服务器通过socket进行数据传输。端口号定为9001。

消息包括两部分，消息头和负载,负载为经过protobuffer序列化后的string信息。具体定义如下表2-1。

表2-1

|  |  |  |
| --- | --- | --- |
| 名称 | 标识 | 描述 |
| 消息头部 | cmd | 4字节 |
| 负载长度 | length | 4字节，不包括消息标识和消息本字段的长度 |
| 错误代码 | error | 4字节 |
| 附加参数1 | para1 | 4字节，在写入人脸信息时，保存client\_id的size（即地点id的大小） |
| 附加参数2 | para2 | 4字节 |

#### muduo网络库

通信模块采用muduo网络库与多种通信实体进行网络通信，muduo基于Reactor模式实现。

多Reactor多线程，也就是所谓的one loop per thread，有一个mainReactor负责accept连接，已建立的连接由subReactor全权控制：当一个新的连接到来，main Reactor只负责把它分配给某个subReactor，然后不再关心该连接。一个Reactor对应一个EventLoop

1. 网络通信模块类图



图2-1

当服务端收到一个任务请求后，根据任务类型，生成对应的function回调，将任务放入线程池中执行任务。MuduoServer是负责网络通信，负责监听外部的请求，并建立连接。slave\_server继承自MuduoServer，负责连接完成后任务的处理。

1. 类详细设计
2. 类名：MuduoServer
   1. 简介

MuduoServer负责监听和接受来自Client的连接和请求。

* 1. 类的定义如下图所示



图 2-2

1. 类名 ：slave\_server
   1. 简介

建立连接时处理连接事件，以便后续对该连接的数据进行处理。

* 1. 类的定义如下图



图2-3

1. 类名 ：PackageCodec
   1. 简介
      1. 对接收到数据进行解析处理。
      2. 发送执行任务完成后的结果给客户端。
   2. 类的定义如下图



图2-4

## 线程池模块

### 模块功能描述

线程池模块中的线程负责执行具体的任务，包括目标轨迹生成、落脚点分析、同行分析等等。

### 模块详细设计

### 类详细设计

1. 类名：ThreadPool
   1. 类定义如下图所示



图2-5

## 存储模块

### 模块功能描述

本系统中，会有多路摄像头数据同时写入，每路摄像头的数据都按照时间顺序写入并进行存储。

### 模块详细设计

根据 Leveldb 的设计思想，在内存中存在一个 MemTable 和一个 ImmutableMemTable，在磁盘上存在多个 SST。在数据写入时，首先写入 MemTable，该MemTable 是一个跳表，可以对数据进行高效的插入。当内存中的 MemTable 达到阈值后，转化为 Immutable MemTable，并生成新的 MemTable 接收新数据，后台线程会定期的将 Immutable MemTable 写入磁盘，形成 SST。Leveldb 这样做的好处是，为保证数据有序存储，内存中的 MemTable 能够将乱序数据通过跳表结构进行快速的排序和插入，将原本磁盘的随机写转化为 MemTable 持久化时的顺序写，充分利用了机械磁盘的顺序写优势，提高吞吐量。然而针对我们的应用场景，如果直接采用 Leveldb 的思想，则会导致多个摄像头发来的数据在内存中同一个 MemTable 上缓存，这样就会导致数据重叠，难以区分各个摄像头产生的数据。此外，由于监控数据按照时间先后顺序产生，天然有序，因此 MemTable 不必采用复杂的跳表结构。存储引擎采用如图 2-7 所示的多缓存存储策略。

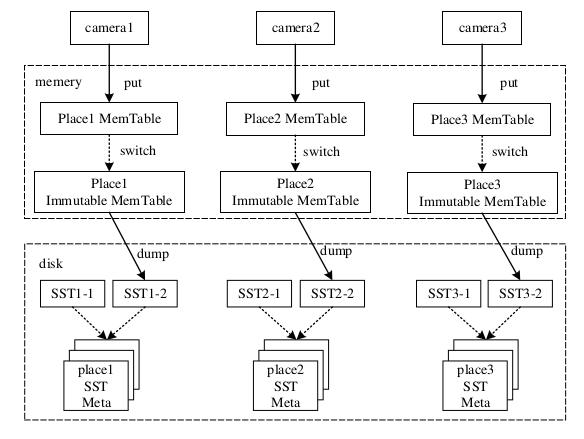


图2-6

buffer为缓存，其中 key 值为监控数据的时间信息 timestamp，其大小固定、不重复，在每个 time 字段后面，附加固定长度的index信息，offset为偏移量，用于标识该 key，如图所示。值所对应的 value在数据区域的位置；value 值为图像的二进制信息 data。



图2-7

将上述的 MemTable 持久化到磁盘即可构成磁盘上的一个数据块（即 SST）。为了更好地组织各个数据块，数据信息中，以数据块中第一个数据的时间作为开始时间 first\_time，以摄像头的编号作为位置信息 placeID(也就是client\_id )，此外还标识出本数据块在磁盘中的存储路径 disk\_path(也就是filepath)。每个数据块的元数据信息如图所示。每个Block\_id为块内第一个记录的时间戳+一秒内索引+camera\_id。查询的时候根据Block\_id进行查找，具体BlockId结构如下图。

将每个Block存储都有与之对应一个目录，整体的Block结构如下：

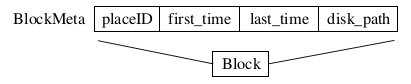


图2-8

具体存储信息：task\_id、camera\_id、时间戳、抓拍记录、图像信息。

其中：

抓拍记录:face\_id、包括了人脸信息、检测的人脸框的信息。人脸信息包括年龄、性别、是否遮挡。

### 类详细设计

Buffer类图

这里的kTimestampLen就是年月时加上一秒内的索引，也就是(YYYYMMDDHHSSFFFFFF)，client\_id就是对应的camera\_id。buffer用来存储对应摄像头数据。



图2-9

BufferID类图



图2-10

BufferGroup类图

用来管理buffer，其中，fetch\_buffer获取buffer用来写入record。fetch\_and\_put\_buffer获取新empty buffer，并归还full buffer存储成block,得到满的buffer的BlockID。read\_buffer获取client\_id的buffer用来读。



图2-11

BlockCache类图

通过DiskThread写入block，把数据块写入磁盘存储介质中，实现持久化存储。



图 2-12

# 数据挖掘

## 目标轨迹分析

### 功能描述

目标轨迹生成：输入一个人脸图片，返回目标的轨迹信息。

轨迹信息包括为：一系列按时间先后顺序排列的时间、地点

### 输入输出

输入：一个人的人脸图片、时间范围、地点范围（可选）、页码、页大小、face\_key（第一次请求时为空）。

输出：目标人脸对应的轨迹信息

### 实现细节

先解析data，如果请求的face\_key在track\_map\_for\_page中，则直接从track\_map\_for\_page中查找，并返回轨迹,如果不在，则进行后面的处理。

提取传入图片的人脸特征，根据请求的起止时间进行轨迹查询。查询（cal\_track函数）。解析时间和地点范围。地点为摄像头id。根据解析的时间和地址范围获取符合要求的所有数据（存储在vector<Item> 中）。cal\_tracker函数，就是搜寻member\_num个最接近目标的图片，将结果存入target(vector<Item>)中 。构建face\_key，用于下次快速查找。set\_track\_result\_by\_page 将数据集分页，结果存储在ack中发送给客户端。

### 接口

#### 请求(单张图片)

消息头部cmd名称：kTargetTrack

对应protobuf message：TargetTrack

功能：本请求需要输入目标人脸特征，找到目标人脸对应的轨迹信息。

负载：protobuf message。具体信息如下表所示。

表3-1 message参数

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 说明 | 备注 |
| image | bytes | 目标人脸图片 |  |
| client\_ids | repeated string | 摄像头id范围 | 可选 |
| timers | repeated string | 时间范围 | timer的格式为“20181031151801”字符串。  timers(0)表示开始时间，timers(1)表示结束时间。 |
| page\_info | PageInfo | 分页信息 |  |

表3-2 pageinfo

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 说明 | 备注 |
| page | int32 | 页码编号 |  |
| page\_size | int32 | 每一页的大小 |  |
| face\_key | string | 人脸索引 | 第一次发请求时，为””。  回复时，服务器端会设置该字段，用于下一次请求时，查找cache用的索引。 |

#### 回复（单张图片）

消息头部cmd名称：kTargetTrackAck。

对应protobuf message：TargetTrackAck。

功能：返回目标人员的轨迹信息。

负载：protobuf message。具体信息如下表所示。

表3-3 message参数

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 说明 | 备注 |
| person | required OnePerson | 人脸轨迹信息 | OnePerson说明见附录 |
| page\_info\_ack | PageInfoAck | 分页信息 |  |

表3-4 PageInfoAck

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 说明 | 备注 |
| page | int32 | 页码编号 |  |
| count | int32 | 轨迹总数目 |  |
| page\_total | int32 | 总的页码数 |  |
| face\_key | string | 人脸索引 | 第一次发请求时，为””。  回复时，服务器端会设置该字段，用于下一次请求时，查找cache用的索引。 |

## 频次分析

### 功能描述

对每个人进行落脚点分析，返回每个人出现的频次，并返回每次出现的图片、时间、地点。

### 输入输出

输入：时间范围，页码、页大小、face\_key（第一次请求时为空）。

输出：该时间段内，每个人的落脚点分析结果。

### 实现细节

cal\_cluster总共有size个目标，然后将聚类，每个查询向量的最近邻的数量为member\_num。然后处理结果就是每个人的频次分析。

## 落脚点分析

### 功能描述

用户输入一个人的图像、时间、地点范围，客户端将请求发送给后台服务器，服务器首先计算目标人脸的轨迹，计算出轨迹后，统计轨迹中，经过每个地点的次数，作为落脚点分析结果。返回结果为：地点id-出现次数，出现时间；地点id-出现次数，出现时间。

### 输入输出

输入：目标人脸图片；起始时间、结束时间；起始地点、结束地点（可选），页码、页大小、face\_key（第一次请求时为空）。

输出：目标人脸在每个地点出现的次数和每一次的时间。

### 实现细节

先解析data,如果请求的face\_key在foothold\_map\_for\_page中，则直接从foothold\_map\_for\_page中查找，并返回轨迹,如果不在，则进行后面的处理。

先提取传入图片的人脸特征。然后解析时间和地点范围。根据解析的时间和地址范围获取符合要求的所有数据（存储在vector<Item> 中）。其中cal\_tracker函数，就是搜寻member\_num个最接近目标的图片，将结果存入target(vector<Item>)中 。将member\_num 个目标按照地点进行划分,如果目标的appearance，ack中没有，则添加目标的appearance并记录其出现的时间。然后构建face\_key，用于下次快速查找。将结果发往客户端。

### 接口

#### 请求

消息头部cmd名称：kFootholdAnalysis。

对应protobuf message：FootholdAnalysis。

功能:计算时间、地点约束条件下，目标人脸的落脚点。

负载：protobuf message。

表3-5 message参数

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 说明 | 备注 |
| image | bytes | 目标人脸图片 |  |
| client\_ids | repeated string | 摄像头id范围 | 可选 |
| timers | repeated string | 时间范围 | timer的格式为“20181031151801”字符串。  timers(0)表示开始时间，timers(1)表示结束时间。 |
| page\_info | PageInfo | 分页信息 | 见附录 |

#### 回复

消息头部cmd名称：kFootholdAnalysis Ack。

对应protobuf message：FootholdAnalysisAck。

功能：返回目标人脸经过的地点的时间和次数。

负载：protobuf message。

表3-6 message参数

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 说明 | 备注 |
| appearance | repeated AppearancePlace | 目标人脸出现的地点及次数 | AppearancePlace见附录 |
| page\_info\_ack | PageInfoAck | 分页信息 | 见附录 |

## 同行分析

### 功能描述

用户输入一个人的人脸图片、时间、地点范围，并输入阈值topN，客户端将请求发送给服务器，服务器计算目标人脸轨迹，并返回与目标人脸轨迹最相近的topN个同行人信息。

### 输入输出

输入：目标人脸图片；起始时间、结束时间（可选）；起始地址、结束地址（可选）；需要返回的轨迹数topN，页码、页大小、face\_key（第一次请求时为空）。

输出：与目标人脸轨迹最相近的topN个同行人信息（主要是face\_id）。

### 实现细节

先解析data,如果请求的face\_key在foothold\_map\_for\_page中，则直接从foothold\_map\_for\_page中查找，并返回轨迹,如果不在，则进行后面的处理。

先从传入的图片中获取人脸特征，解析时间和地点。根据解析的时间和地点获取数据。在cal\_tracker计算时，将时间延迟改为1s，这里会计算出伴随人员的信息存储在tracker.second。将目标人员的轨迹与items（符合解析的时间和地点，得到的所有数据）中的每个人出现的时间进行计算，相差小于1s则认为同行，将结果存储在tracker.second。然后将tracker.second进行聚类，每个查询向量的最近邻的数量为member\_num。最后处理就是找到和目标人员轨迹重叠次数最多的topn。

### 接口

#### 请求

消息头部cmd名称：kPartnerAnalysis。

对应protobuf message：PartnerAnalysis。

功能：本请求需要输入目标人脸特征和参数topn，找到与目标人轨迹相近的topn个同行人信息（主要是同行人face\_id）。

负载：protobuf message。

表3-7 message参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 说明 | 备注 |  |
| image | bytes | 目标人脸图片 |  |  |
| client\_ids | repeated string | 摄像头id范围 | 可选 |  |
| timers | repeated string | 时间范围 | timer的格式为“20181031151801”字符串。  timers(0)表示开始时间，timers(1)表示结束时间。 |  |
| topn | inte32 | 需要返回的同行人个数 |  |  |
| page\_info | PageInfo | 分页信息 | 见附录 |  |

#### 回复

消息头部cmd名称：kPartnerAnalysisAck

对应protobuf message：PartnerAnalysisAck

功能：与目标人轨迹相近的topn个同行人信息（主要是同行人face\_id）。

负载：protobuf message

表3-8 message参数

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 说明 | 备注 |
| topn\_partner | repeated OnePerson | topn个同行人信息 | 此处可只读取face\_id和task\_id字段 |
| page\_info\_ack | PageInfoAck | 分页信息 | 见附录 |

## 伴随分析

### 确定性伴随分析

就是多目标轨迹挖掘，传入多张人脸。

### 非确定性伴随分析

#### 功能描述

找出潜在的、与目标人员存在伴随关系的人的信息（主要是伴随人face\_id）。

#### 输入输出

输入：目标人脸图片；时间范围deltaT，页码、页大小、face\_key（第一次请求时为空）。

输出：在deltaT时间范围内，潜在伴随人的face\_id。

#### 实现细节

先解析data,如果请求的face\_key在uncertain\_accompany\_map\_for\_page中，则直接从foothold\_map\_for\_page中查找，并返回轨迹,如果不在，则进行后面的处理。

从传入的图片中获取人脸特征将jpg转为mat，提取人脸特征。然后解析时间和地点范围。根据解析的时间和地址范围获取符合要求的所有数据（存储在vector<Item> 中）。cal\_tracker函数，计算目标轨迹和伴随人员pair<vector<Item>, vector<Item>>得到符合的数据，cal\_cluster将伴随人员聚类，apriori求出频繁项集。

#### 接口

1. 请求

消息头部cmd名称：kUncertainAccompanyAnalysis。

对应protobuf message：UncertainAccompanyAnalysis。

功能：本请求需要输入目标人脸特征，根据目标人脸特征，找出与目标人员存在潜在伴随关系的其他人员。

负载：protobuf message。

表3-9 message参数

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 说明 | 备注 |
| target | OnePerson | 目标人脸轨迹 | OnePerson见附录 |
| accompany | Repeated OnePerson | 与目标有伴随关系的人员信息 | 此处可只读取face\_id和task\_id字段 |
| page\_info\_ack | PageInfoAck | 分页信息 | 见附录 |
| 字段 | 类型 | 说明 | 备注 |
| target | OnePerson | 目标人脸轨迹 | OnePerson见附录 |

#### 回复

消息头部cmd名称：kUncertainAccompanyAnalysisAck

对应protobuf message：UncertainAccompanyAnalysisAck

功能：返回目标人员的轨迹信息，并返回潜在伴随人员的信息（主要是伴随人face\_id）。

负载：protobuf message。

表3-10 message参数

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 说明 | 备注 |
| target | OnePerson | 目标人脸轨迹 | OnePerson见附录 |
| accompany | Repeated OnePerson | 与目标有伴随关系的人员信息 | 此处可只读取face\_id和task\_id字段 |
| page\_info\_ack | PageInfoAck | 分页信息 | 见附录 |

# 系统流程设计

## Dlogdb 系统启动时序图



1．Main()函数，先初始化ConfinManager配置文件和线程池ThreadPool。

2．slave\_server初始化并将muduo初始化，设置LoopThread数量，绑定ConnectionCallback,将MessageCallback绑定到PackageCodec::recv函数。

3．初始化完成后，等待服务的请求。

## 执行Task时序图



1. 收到客户端发送来的消息，然后解析req。
2. 解析req请求后，根据对应的cmd去找到对应的Task。
3. 将找到的Task分发到线程池，会有某一线程去执行Task。
4. Task执行完成后，会调用package\_codec::send函数，将执行完Task后的结果发送到客户端。

### 目标轨迹分析时序图



1. 执行Task时，先解析客户端发来的req，解析req中的信息。
2. 提取传入图片的人脸特征。
3. 解析时间和地点范围。
4. 根据解析的时间和地址范围获取符合要求的所有数据（存储在vector<Item> 中）。
5. 先回获取内存中的数据(recorder::get()),然后再去磁盘中查找符合条件的数据(EnvDefault::get()然后到blockcache::get())。在查找磁盘数据时，会有一个DiskThread专门进行磁盘的读取。
6. 然后进行目标轨迹分析，cal\_tracker函数，就是搜寻member\_num个最接近目标的图片，将结果存入target(vector<Item>)中。构建face\_key，用于下次快速查找。set\_track\_result\_by\_page 将数据集分页。
7. 结果存储在ack中发送给客户端。

### 频次分析时序图



1. 执行Task时，先解析客户端发来的req，解析req中的信息。
2. 提取传入图片的人脸特征。
3. 解析时间和地点范围。
4. 根据解析的时间和地址范围获取符合要求的所有数据（存储在vector<Item> 中）。
5. 先回获取内存中的数据(recorder::get()),然后再去磁盘中查找符合条件的数据(EnvDefault::get()然后到blockcache::get())。在查找磁盘数据时，会有一个DiskThread专门进行磁盘的读取。
6. 进行频次分析，其中cal\_tracker函数，就是搜寻member\_num个最接近目标的图片，将结果存入target(vector<Item>)中。cal\_cluster总共有size个目标，然后将聚类，每个查询向量的最近邻的数量为member\_num。然后处理结果就是每个人的频次分析。
7. 将最终的结果发往客户端。

### 落脚点分析时序图



1. 执行Task时，先解析客户端发来的req，解析req中的信息。
2. 提取传入图片的人脸特征。
3. 解析时间和地点范围。
4. 根据解析的时间和地址范围获取符合要求的所有数据（存储在vector<Item> 中）。
5. 先回获取内存中的数据(recorder::get()),然后再去磁盘中查找符合条件的数据(EnvDefault::get()然后到blockcache::get())。在查找磁盘数据时，会有一个DiskThread专门进行磁盘的读取。
6. 进行落脚点分析，其中cal\_tracker函数，就是搜寻member\_num个最接近目标的图片，将结果存入target(vector<Item>)中 。将member\_num 个目标按照地点进行划分,如果目标的appearance，ack中没有，则添加目标的appearance并记录其出现的时间。
7. 构建face\_key，用于下次快速查找。将结果发往客户端。

### 同行分析



1. 执行Task时，先解析客户端发来的req，解析req中的信息。
2. 提取传入图片的人脸特征。
3. 解析时间和地点范围。
4. 根据解析的时间和地址范围获取符合要求的所有数据（存储在vector<Item> 中）。
5. 先回获取内存中的数据(recorder::get()),然后再去磁盘中查找符合条件的数据(EnvDefault::get()然后到blockcache::get())。在查找磁盘数据时，会有一个DiskThread专门进行磁盘的读取。
6. 同行分析，在cal\_tracker计算时，将时间延迟改为1s，这里会计算出伴随人员的信息存储在tracker.second。将目标人员的轨迹与items（符合解析的时间和地点，得到的所有数据）中的每个人出现的时间进行计算，相差小于1s则认为同行，将结果存储在tracker.second。然后将tracker.second进行聚类，每个查询向量的最近邻的数量为member\_num。最后处理就是找到和目标人员轨迹重叠次数最多的topn。
7. 构建face\_key，用于下次快速查找。将结果发往客户端。

### 伴随分析时序图



1. 执行Task时，先解析客户端发来的req，解析req中的信息。
2. 提取传入图片的人脸特征。
3. 解析时间和地点范围。
4. 根据解析的时间和地址范围获取符合要求的所有数据（存储在vector<Item> 中）。
5. 先回获取内存中的数据(recorder::get()),然后再去磁盘中查找符合条件的数据(EnvDefault::get()然后到blockcache::get())。在查找磁盘数据时，会有一个DiskThread专门进行磁盘的读取。
6. 数据读取完成后，进行伴随分析。cal\_tracker函数，计算目标轨迹和伴随人员，将符合的数据存储在pair<vector<Item>, vector<Item>>。
7. cal\_cluster采用k-means算法将人员进行聚类。
8. 调用apriori算法进行不确定性伴随分析，将最终的结果发送到客户端。