视频监控开发文档

## 前言

档是为了更好的阅读项目跟节点交互,制定本文档,项目跟节点交互主要应用了Tcp连接方式,当项目启动时会进行连接操作,连接需要互相确认.然后保存此链接,从而达到互相通信的目的.项目数据交互格式为JSON数据.双方生成JSON数据操JAR包为fastjson,每次发送信息成功后,必须等待对方发送”ok”确认信息.

文档交互API为节点方进行维护升级。

#{demo}

### 开发软件

Idea + Navicat Premium +Xshell

### 项目环境

#### 基本项目结构

项目属于maven项目,框架为:springmvc spring mybatis mysql websocket

权限校验为：shiro

页面：请求转发jsp页面，所用到的库 jquery bootstrap

视频播放所用内库为：videoJs 版本为 5.x

## 与节点交互文档

#### 交互流程总览

算法节点与系统交互是通过TCP交互方式进行连接.连接完成后会通过该方式来交换数据.数据交互文本格式为JSON格式.每次交互都要把JSON转换为相对应的类。所有连接操作在: TcpClientHandler 文件中完成。

#### 项目流程图片

##### 项目启动流程

|  |
| --- |
|  |

##### 项目交互API说明:

|  |
| --- |
|  |

##### 项目节点连接示意图

|  |
| --- |
|  |

#### 操作前提

* web项目中需要引入算法节点给的jar包，版本1的jar包为3.3.4,需要生成jar包到用户仓库,具体操作为:

|  |
| --- |
| Mvn install:install-file -Dfile=./ImageProcessApiInterface.jar -DgroupId=cn.swiftdata -DartifactId=ImageProcessApiInterface -Dversion=1.0 -Dpackaging=jar  注:   1. –Dfile jar包所在的路径 2. –Dversion 版本号   (需要下载maven) |

* 所有连接收到之后，双方都要进行确认，确认码为”OK”

##### 公共参数解析

|  |  |
| --- | --- |
| 参数名称 | 注释 |
| requestCode | 请求码(必须) |
| result | 操作结果 0为成功 1为失败 |
| Id | 生成通道的UUID,在节点做唯一标识符 |
| Message | 返回结果所携带的信息 |
| **Tag** | 存放web项目所携带的,用户uuid每一个发出的请求都需要携带 |

#### 开始建立连接

项目采用MQ发布订阅模式进行交互,与节点建立连接的方式为:

* 项目启动时,项目会自动读取所有node节点信息进行统一连接,如果连接成功会进行更新数据库和在项目中进行储存连接信息,详情看代码.
* 项目启动完成后可以由我方代表进行连接节点操作.服务地址为 /node/node，可以进行点击连接。连接完成后进行下面操作, 详情看代码.

当我们连接的时候会进行两次连接,如果成功,则返回true,否则返回false

##### 写建立连接交互类为: ReqConnection

交互连接分为,上行和下行,主要用于和算法节点进行沟通作用,交互连接为项目运行的基础.当项目运行的时候,会对,计算节点表进行扫表操作.逐个进行连接.也可以在节点页面,对节点进行连接:

###### 上行连接(写)

|  |  |
| --- | --- |
| 写入参数 | |
| 功能 | 用于与项目进行连接.每当与节点进行交互的时候,会进行连接，该连接返回true成功，返回false表示失败，失败后无法进行下一步操作，该操作与下行操作必须同时成功。如果有一个不成功则不会返回true。表示该节点无法连接 |
| hint | 发送信息，信息头 |
| url | 上传文件静态服务器地址,在resources/spring/uplode-config.properties,中配置 |
| 输出参数:无 | |
| 说明：每次进行连接需要发送信息头,信息头内容为调用为写服务还是读服务。如果是写服务需要携带URL地址。这样的目的是增加双边交互。只修改一次url地址：UplodePath负责传输该项。  实例:  *//启动socket* socket = **new** Socket(**ip\_address**, **ip\_port**); *//静态服务器地址拼接* String staticImageServer = **uplodePath**.getIp() +**":"** + **uplodePath**.getPort() + **"/"** + **uplodePath**.getFileName() +**"/"**; *//编码* staticImageServer = URLEncoder.*encode*(staticImageServer,**"utf-8"**); *//初始化连接* ReqConnection reqConnection = **new** ReqConnection(**"write"**, (staticImageServer).trim()); *//写入连接* socket.getOutputStream().write((JSON.*toJSONBytes*(reqConnection)); **byte**[] buf = **new byte**[2]; **int** read = socket.getInputStream().read(buf); *//得到返回值* String response = **new** String(buf, 0, read); *//判断返回值* **if** (!response.equalsIgnoreCase(**"OK"**)) {  closeSocket(socket);  **return false**; } *//标记成功* **socketWrite** = socket; **write** = **true**; **return true**; | |

###### 下行连接(负责读取)

|  |  |  |
| --- | --- | --- |
| 写入参数 | | |
| hint | 发送信息，信息头(read) | |
| 功能 | 用于与项目进行连接.每当与节点进行交互的时候,会进行连接，该连接返回true成功，返回false表示失败，失败后无法进行下一步操作，该操作与下行操作必须同时成功。如果有一个不成功则不会返回true。表示该节点无法连接 |
| 输出参数:无 | | |
| 实例:同写连接无需携URL. | | |

#### 如何进行每次连接时调用

|  |
| --- |
| 当与算法节点连接成功后,会进行命令的传输调用,首先我们需要进行找到相对的handler，然后我们需要这个handle进行数据的发送。下面是寻找handler的事例. |
| *//通过通道ID得到通道* Monitoring monitoring = **this**.getMonitoringByUuid(uuid); Ipc ipc = **new** Ipc(); ipc.setUuid(monitoring.getFkA02()); *//通过IPC\_ID得到连接* TcpClientHandler handler = **cameraService**.getNodeByIpc(ipc); ReqTogglePlay reqToggleDetect = **new** ReqTogglePlay(uuid,isPlay); *//发送连接* handler.sendMessage(JSON.*toJSONString*(reqToggleDetect)); |

#### 对于BaseRequest中tag的说明

BaseRequest中的tag是一个固定变量,类型为JAVA中的Map。我们可以存放任意数据。然后这条消息被返回的时候。返回消息会在NodeMessage中存在。我们可以根据我们的key来找到对应的数据。从来进一步完善我们的操作，

当一个用户调用命令之后,在tag里面必须存放用户ID，用于给页面发送消息使用。标注key为: UserSessionConstant.***USER\_SESSION\_UUID***

#### 创建通道

创建交互类为ReqCreateCamera

|  |  |  |
| --- | --- | --- |
| 写入参数 | | |
| 使用说明 | 该方法是进行通知算法节点，打开一个摄像头,并且做好准备就绪工作.  当创建成功后。算法节点给系统发送消息(创建成功/失败)。系统会根据消息内容进行下一步操作. (普通用户无该权限) | |
| Id | Monitoring生成的id, | |
| username | Ipc连接账号 | |
| password | Ipc连接密码 | |
| ipAddress | 需要连接IP地址(IPC的连接地址) | |
| port | Ipc的端口号,默认为554 | |
| rtmpURL | 需要推流地址 | |
| 输出参数(trg) | | |
| **stream\_url** | | 推送到rtmp的nginx路径 |
| 代码示例:  *//创建通道* ReqCreateCamera camera = **new** ReqCreateCamera(monitoring.getUuid(),  newIpc.getAccessId(), newIpc.getAccessPwd(), newIpc.getIpAddr(), Integer.*valueOf*(newIpc.getPortNum()), rtmp\_url.getRtmpUrl());  *//推流地址* camera.**tag**.put(**"stream\_url"**, rtmp\_url.getRtmpUrl()); *//当前用户登陆ID* camera.**tag**.put(UserSessionConstant.***USER\_SESSION\_UUID***, user.getUuid()); | | |

#### 关闭通道

交互API: BaseRequest 基本API需要传入requestCode

|  |  |  |
| --- | --- | --- |
| 写入参数:无 | | |
| 使用说明 | 当通道不用的时候,可以点击关闭按钮.如果用完不关闭通道最大的缺点是资源占用(普通用户无该权限) | |
| 输出参数(trg) | | |
| stream\_url | | 推送到rtmp的nginx路径 |
| 代码示例:  *//调用关闭通道信息* BaseRequest baseRequest = **new** BaseRequest(BaseRequest.***CAM\_SHUTDOWN***, ipcId); baseRequest.**tag**.put(UserSessionConstant.***USER\_SESSION\_UUID***, loginUser.getUuid()); | | |

#### 开始/关闭播放

交互APIReqTogglePlay

|  |  |  |
| --- | --- | --- |
| 写入参数 | | |
| 说明: | 当播放通道开启后会调用该方法，进行视频自动播放，或者用户点击播放，用户需要传入播放通道ID和是否进行播放参数。当返回消息时我们获取到tag到内容进行后续操作 | |
| Id | Monitoring生成的id, | |
| play | 是否播放 true/false | |
| 输出参数(trg) | | |
| play | | 执行后的结果存入数据库 |
| 代码示例:  *//执行播放命令* ReqTogglePlay reqToggleDetect = **new** ReqTogglePlay(cameraId,**true**); *//需要的数据放入tag* reqToggleDetect.**tag**.put(TcpInteractionConstant.***IS\_TOGGLE\_PLAY***,**true**); *//发送消息到节点* sendMessage(JSON.*toJSONString*(reqToggleDetect)); | | |

#### 录像相关

交互BaseRequest

|  |  |  |  |
| --- | --- | --- | --- |
| 写入参数 | | | |
| 说明: | 本系统携带录像功能.录像功能在算法节点进行处理. | | |
| Id | | | Monitoring生成的id, |
| requestCode | | | 参看 requestCode Api |
| 开始录像输出参数(无) | | | |
| 结束录像参数 | | | |
| response.getVideoUrl() | | 录像存放地址(注意解码) URLDecoder | |
| response.getImgUrl() | | 缩略图地址(注意解码) | |
| 代码示例:  *//开启录像*  BaseRequest baseRequest = **new** BaseRequest(BaseRequest.CAM\_START\_RECORD\_VIDEO, ipcId);  *//关闭录像*  BaseRequest baseRequest = **new** BaseRequest(BaseRequest.CAM\_STOP\_RECORD\_VIDEO, ipcId); | | | |

#### 截图相关(设置请求截图也是这个地址)

交互BaseRequest

|  |  |  |  |
| --- | --- | --- | --- |
| 写入参数 | | | |
| 说明: | 本系统携带截图功能.录像功能在算法节点进行处理. | | |
| Id | | | Monitoring生成的id, |
| requestCode | | | 参看 requestCode Api |
| 输出参数: | | | |
| response.getImgUrl() | | 图片所在地址 | |
| 代码示例:  *//当前播放截图*  BaseRequest baseRequest = **new** BaseRequest(BaseRequest.CAM\_SHOOT\_IMAGE,ipcId);  *//获取未划线截图*  BaseRequest baseRequest = **new** BaseRequest(BaseRequest. CAM\_SHOOT\_IMAGE\_ORIGINAL  , ipcId);  *//获取设置区域后的截图*  BaseRequest baseRequest = **new** BaseRequest(BaseRequest. CAM\_SHOOT\_IMAGE\_ROI  , ipcId); | | | |

#### 绊线相关

交互ReqSetROI

|  |  |  |
| --- | --- | --- |
| 写入参数 | | |
| 说明: | 当监控范围感觉太大的时候,可以通过设置绊线来监控固定的范围,超过绊线范围的位置不做监控. | |
| Id | | Monitoring生成的id, |
| rectsList | | 暂时用不到,设置为空对象 |
| irregularList | | 坐标点集合 |
| Width | | 画布宽度 |
| Hight | | 画布的宽度 |
| 代码示例:  *//定义一个点坐标* List<Point> pointList = **new** ArrayList<>(); *//定义点坐标集合* List<List<Point>> lists = **new** ArrayList<>(); *//根据通道ID查询通道* Monitoring monitoring = **monitoringService**.getMonitoringByUuid(ipcId); Ipc ipc = **new** Ipc(); ipc.setUuid(monitoring.getFkA02()); *//得到所有点的X,Y轴坐标* String[] pointsArray = points.replaceAll(**"[\\[\\]]"**, **""**).split(**","**); **for** (**int** i = 0; i < pointsArray.**length**; i += 2) {  *//得到X轴坐标* **int** x = Integer.*parseInt*(pointsArray[i]);  *//得到Y轴坐标* **int** y = Integer.*parseInt*(pointsArray[i + 1]);  *//组合成一个点* Point p = **new** Point(x, y);  *//插入点集合* pointList.add(p); } TcpClientHandler handler = **cameraService**.getNodeByIpc(ipc); lists.add(pointList); ReqSetROI reqSetROI = **new** ReqSetROI(ipcId, Collections.<Rectangle>*emptyList*(), lists, width, height); | | |
| 输出参数,  在tag中需要保存点集合。以备以后使用。 | | |

#### 开启/关闭绊线

交互ReqToggleROI

|  |  |  |
| --- | --- | --- |
| 写入参数 | | |
| 说明: | 如果在画面上设置了绊线，我们我们可以关闭/开启绊线来进行操作。 | |
| Id | | Monitoring生成的id, |
| useRoi | | true:启用绊线，false:关闭绊线 |
| 代码示例:  ReqToggleROI reqToggleROI = **new** ReqToggleROI(ipcId,isLineStatus); reqToggleROI.**tag**.put(UserSessionConstant.***USER\_SESSION\_UUID***,user.getUuid()); reqToggleROI.**tag**.put(TcpInteractionConstant.***GETISTOGGLEROI***,isLineStatus); | | |

#### 算法相关

交互ReqAlgSetting

|  |  |  |
| --- | --- | --- |
| 写入参数 | | |
| 说明: | 我们可以通过设置该方法来设置算法的各种参数。来进行算法进行过滤得到最佳监控目的。 | |
| Id | | Monitoring生成的id, |
| timeInSeconds | | 时间阀值(s): |
| detectionType | | 算法类型 |
| learnRate | | 学习率 |
| min | | 设置对象最小范围 |
| Max | | 设置对象最大范围 |
| percentage | | 人流密集度(只有人流对冲由此算法) |
| 输出参数(公共参数) | | |

\*本部分属于动态参数,数据由数据库提供

#### 公共参数设置

交互ReqAdvanceSettings

|  |  |  |
| --- | --- | --- |
| 写入参数 | | |
| 说明: | 我们可以根据我们每个通道的线条的粗细和线条颜色。 | |
| Id | | Monitoring生成的id, |
| Color(int) | | 画线的颜色 |
| thickness(int) 1-5 | | 线条粗细 |
| 输出参数(公共参数) | | |

\*本部分属于动态参数,数据由数据库提供,color为固定参数

#### 切换算法API

交互ReqSetAlgorithm

|  |  |  |
| --- | --- | --- |
| 说明: | 本系统可以支持算法切换.目前支持三种算法.用此API可以通知算法节点此通道需要执行的算法 | |
| 写入参数 | | |
| Id | | Monitoring生成的id, |
| detectionType(String) | | 算法类型（算法类的全路径） |
| 输出参数(公共参数) | | |
| 代码示例:  *//查询算法ID* Vaengine vaengine = **vaengineService**.getVaengineByEngineId(detectionType); *//调用算法API* ReqSetAlgorithm algorithm = **new** ReqSetAlgorithm(cameraId, detectionType); *// 放入用户信息* algorithm.**tag**.put(UserSessionConstant.***USER\_SESSION\_NAME***,user.getUuid()); *//放入该算法ID* algorithm.**tag**.put(TcpInteractionConstant.***DETECTION\_TYPE***,vaengine.getUuid()); | | |

#### 开始/结束分析

交互ReqToggleDetect

|  |  |  |
| --- | --- | --- |
| 说明: | 通道创建成功后，可以自动分析和暂停分析。每次调用此类 | |
| 写入参数 | | |
| Id | | Monitoring生成的id, |
| detect(boolean) | | 是否开始/结束分析 |
| 输出参数(公共参数) | | |
| 代码示例:  User user = UserLoginMessage.*getLoginUser*(); *//方法开始 通道ID 是否开启分析布尔* ReqToggleDetect reqToggleDetect = **new** ReqToggleDetect(uuid,isAnalyze); *//用户信息* reqToggleDetect.**tag**.put(UserSessionConstant.***USER\_SESSION\_UUID***,user.getUuid()); *//状态* reqToggleDetect.**tag**.put(TcpInteractionConstant.***ISANALYZE***,isAnalyze); | | |

注：tag需要放入detect需要写数据库

#### 是否开始自动取证

交互ReqToggleAutoCapture

|  |  |  |
| --- | --- | --- |
| 说明: | 算法可以根据当前通道参数设置来进行取证当前IPC监控所得的异常，然后保存进数据库中 | |
| 写入参数 | | |
| Id(String) | | Monitoring生成的id, |
| autoCaptureOn(boolean) | | 是否开始/结束自动取证, |
| 输出参数(公共参数) | | |
| 代码示例:  User user = UserLoginMessage.*getLoginUser*(); *//方法开始 通道ID 是否开启分析布尔* ReqToggleAutoCapture reqToggleAutoCapture = **new** ReqToggleAutoCapture(uuid,isAutoEvidence); *//是否自动取证 存表* reqToggleAutoCapture.**tag**.put(TcpInteractionConstant.***AUTOEVIDENCE***,isAutoEvidence); *//用户信息* reqToggleAutoCapture.**tag**.put(UserSessionConstant.***USER\_SESSION\_UUID***,user.getUuid()); | | |

注:tag需要放入detect需要写数据库

## 交互API相关文档

#### IpApiConstants

##### Web项目向算法节点发送的状态码

|  |  |  |
| --- | --- | --- |
| 记录id | 变量名 | 注释 |
| 0 | CAM\_CREATE | 创建通道 |
| 1 | CAM\_TOGGLE\_PLAY | 切换播放状态 |
| 2 | CAM\_TOGGLE\_VIEW | 是否切换到后台进行 |
| 3 | CAM\_TOGGLE\_DETECT | 切换算法(参照切换算法API) |
| 9 | CAM\_SHUTDOWN | 关闭当前通道 |
| 14 | CAM\_SHUTDOWN\_ALL | 关闭当前节点全部通道 |
| 11 | CAM\_START\_RECORD\_VIDEO | 开始录屏 |
| 12 | CAM\_STOP\_RECORD\_VIDEO | 结束录屏 |
| 10 | CAM\_SHOOT\_IMAGE | 截取图片 |
| 13 | CAM\_SET\_DETECTION | 设置算法(详情请查看算法API) |
| 15 | NODE\_GET\_CAMERA\_LIST | 获取当前节点所有通道 |
| 19 | CAM\_SET\_ROI | 设置基本参数(颜色/线条粗细) |
| 21 | NODE\_GET\_CHANEL\_INFO | 获取某通道参数 |
| 22 | CAM\_ADVANCE\_SETTING | 为通道算法添加高级参数 |
| 23 | CAM\_ALG\_SETTING | 为通道算法添加参数 |
| 24 | CAM\_GET\_ALG\_PARAMS | 获取通道参数(未使用) |
| 25 | CAM\_TOGGLE\_AUTO\_CAPTURE | 是否开启自动取证 |
| 26 | CAM\_TOGGLE\_ROI | 是否开启粄线设置 |
| 27 | CAM\_SHOOT\_IMAGE\_ORIGINAL | 获取原始图片未添加绊线 |
| 28 | CAM\_SHOOT\_IMAGE\_ROI | 获取添加添加绊线图片 |

节点主动发送的状态码

|  |  |  |
| --- | --- | --- |
| 记录id | 变量名 | 注释 |
| 8 | CAM\_AUTO\_PAUSED | 遇到错误暂停当前通道 |
| 16 | CAM\_AUTO\_STOP\_DETECT | 遇到错误停止当前通道算法 |
| 7 | CAM\_ERROR | 摄像头错误 |

##### 保留字段

|  |  |  |
| --- | --- | --- |
| 记录id | 变量名 | 注释 |
| 17 | CAM\_PRE\_ALARM\_SIGNAL |  |
| 18 | CAM\_ALARM\_SIGNAL |  |

#### CameraInfo(通道信息)

|  |  |  |
| --- | --- | --- |
| 变量名 | 注释 | 备注 |
| sourceType | 资源类型 | 文件资源(主版本不做开发) |
| id | 通道Id | 唯一ID |
| Username | 打开IPC用户名 |  |
| Password | 打开IPC密码 |  |
| ipAddress | IPC所在地址 |  |
| port | 端口 | 默认553 |
| streamURL | 推送到nginx播放流地址 |  |
| pathToFile | 文件地址 | 文件必须在节点 |
| currentAlg | 当前算法 |  |
| isPlaying | 是否播放 |  |
| isDetecting | 是否分析 |  |
| color | 当前线条颜色 | 默认红色 |
| thickness | 当前线条粗细 |  |
| timeInSeconds | 报警等待时间 | 默认5秒 |
| learnRate | 学习率 | 默认值是自适应 -1 至1 |
| min | 对象最小范围 | 10个等级 |
| max | 对象最大范围 | V1.0版本没有 |

#### ReqSetAlgorithm(算法分析相关)

|  |  |  |
| --- | --- | --- |
| 变量名 | 注释 | 备注 |
| ALG\_NORMAL | 无任何算法分析 |  |
| ALG\_ABNORMAL\_STOPPING | 非法停留分析 |  |
| ALG\_PEOPLE\_FLOW\_ANALISE | 人流对冲分析 |  |
| ALG\_REGION\_INVASION | 区域入侵 |  |