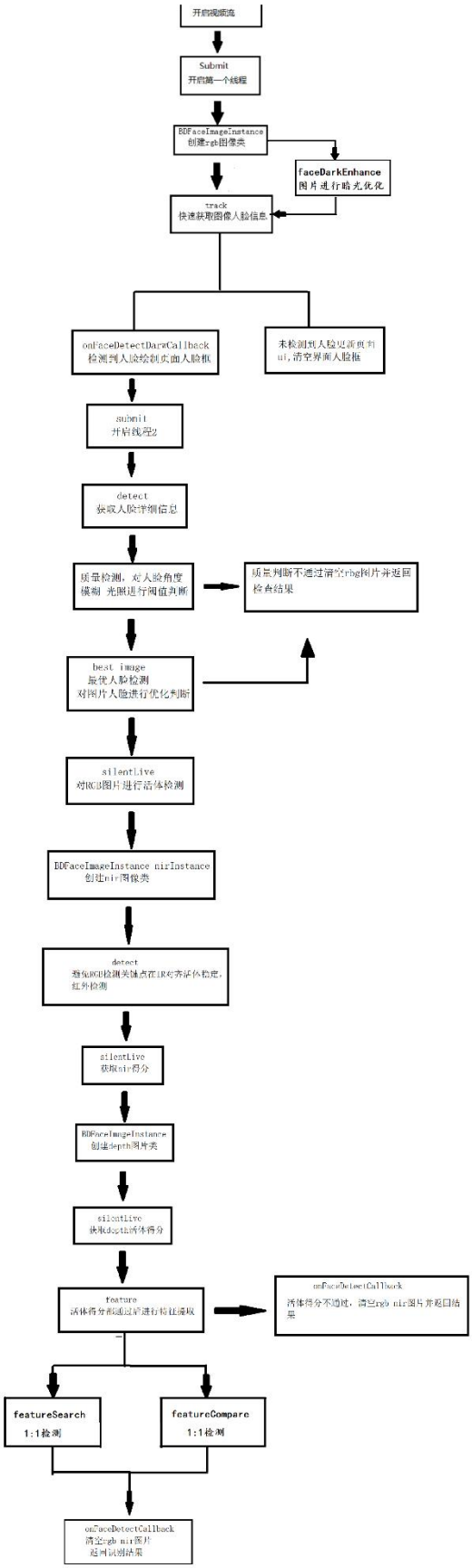
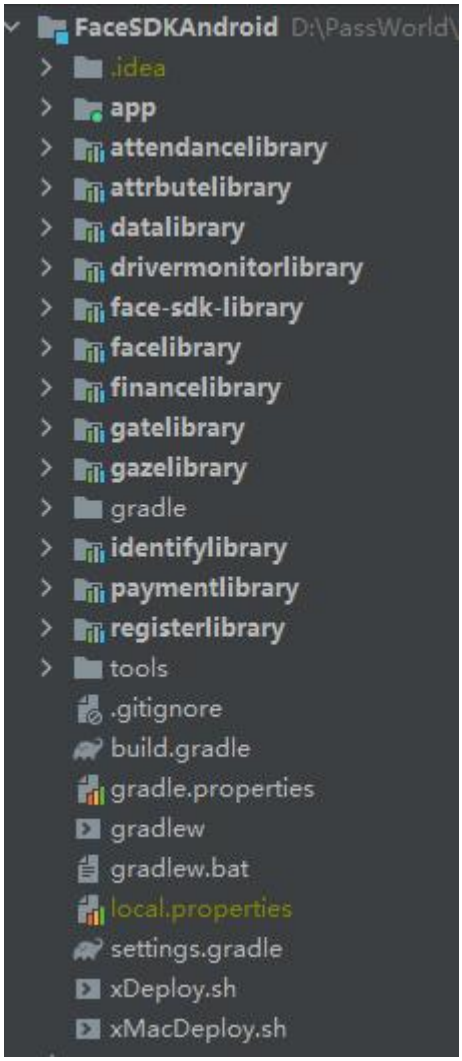
识别流程图

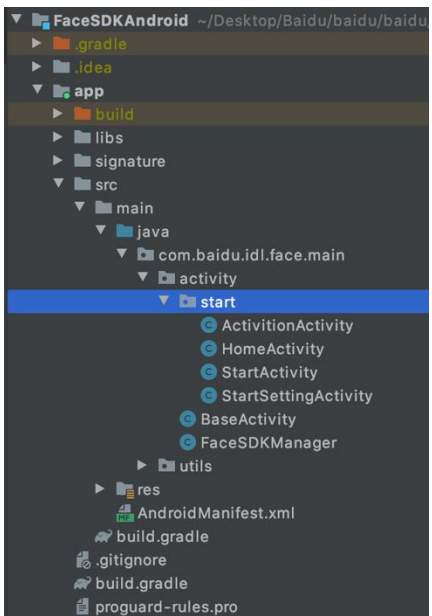

开启视频流
Submit
开启第一个线程
BDFaceImageInstance
创建rgb图像类
faceDarkEnhance
图片进行暗光优化
track
快速获取图像人脸信息
onFaceDetectDarkCallback
检测到人脸绘制出面人脸框
未检测到人脸更新页面ui,清空界面人脸框
submit
开启线程2
detect
获取人脸详细信息
质量检测，近人脸角度
模糊 光照进行阈值判断
质量判断不通过清空rbg图片并返回检查结果
best image
最优人脸检测
对图片人脸进行优化判断
silentLive
对RGB图片进行活体检测
BDFaceImageInstance nirInstance
创建nir图像类
detect
通过RGB检测关键点在IR对齐活体检测，
红外检测
silentLive
获取nir得分
BDFaceImageInstance
创建depth图片类
silentlive
获取depth活体得分
feature
活体得分都通过后进行特征提取
onFaceDetectCallback
活体得分不通过，清空rgb nir图片并返回结果
featureSearch
1:1检测
featureCompare
1:1检测
onFaceDetectCallback
清空rgb nir图片
返回识别结果

一：工程目录及功能入口



（工程目录图）

整个工程目录分为 5 个部分。

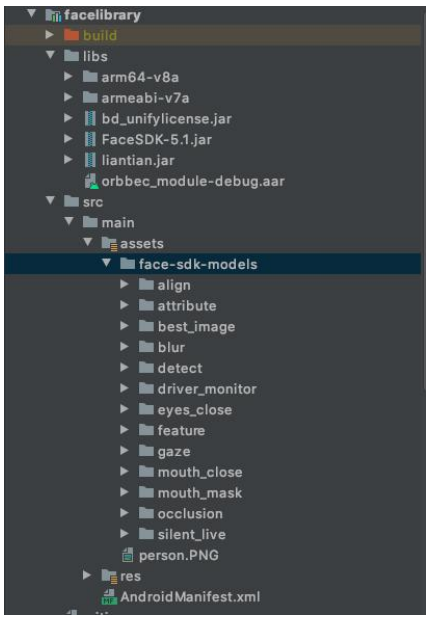


（1）app 主 module：

(1 图)

app module 包含 StartActivity 启动页，ActivitionActivity 激活 sdk 和 HomeActivity 主页（详细页面交互请下载运行 7.1demo）

(2) facelibrary 核心算法 module **:**



(2 图)

facelibrary 模块为 sdkDemo 核心功能模块，其他模块必须依赖。

1.libs 目录为动态库 so 和 jar 包，包含 arm64-v8 和 armeabi-v7a 两个平台.

2.assets 目录为模型文件。

assets/face-sdk-models 目录下存放各种模型能力，可根据自己的工程项目需要，选择相应的模型能力。

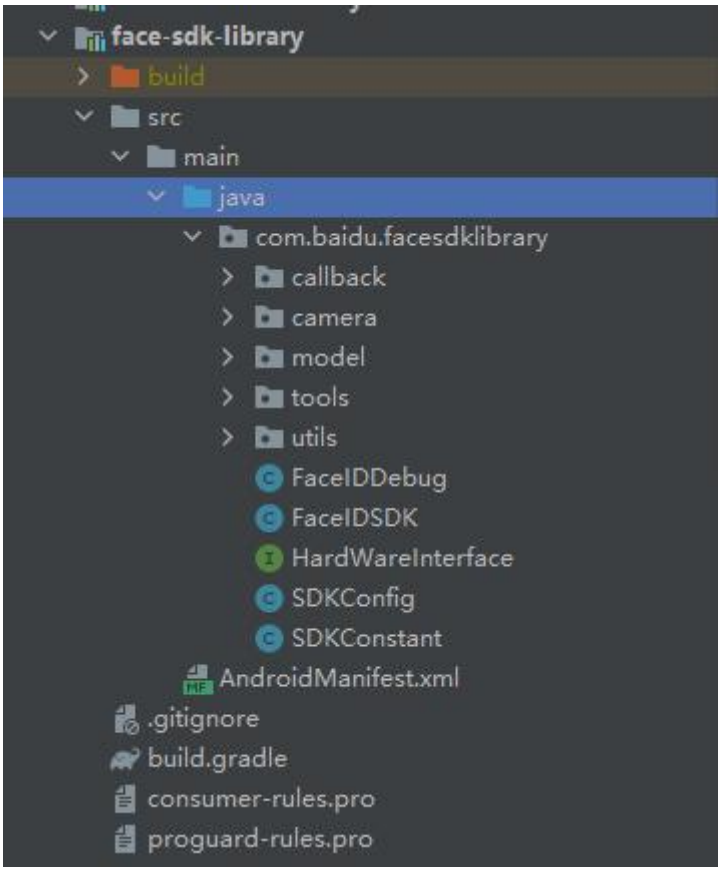
建议把整个 face-sdk-models 目前放到自己的工程的 assets 目录下

(3) facelibrary 核心算法 module **:**

face-sdk-models 模型包含:

1. align 对齐模型
2. attribute 属性模型
3. best_image 最优人脸模型
4. blur 质量模糊模型
5. detect 人脸检测模型
6. driver_monitor 驾驶行为模型
7. eyes_close 眼睛闭合模型
8. feature 识别模型
9. gaze 注意力模型
10. mouth_close 嘴巴闭合模型
11. mouth_mask 口罩模型
12. occlusion 遮挡模型
13. silent_live 活检模型

(4) face-sdk-library high-level api 核心 算法:



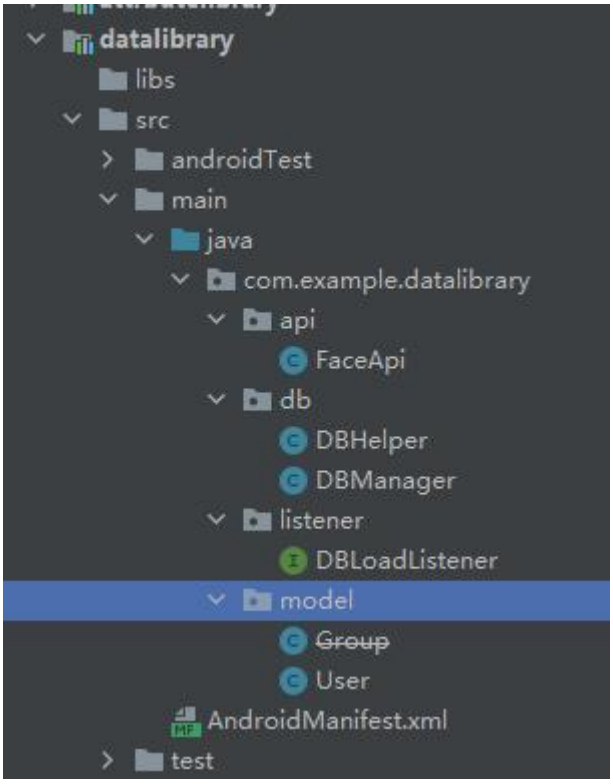
(图 3)

face-sdk-library 内部集成了识别流程，其他模块继承 face-sdk-library 只需要调取一个方法即可完成整体识别流程，目前只集成了活体检测，人脸特征提取，人脸比对，人脸属性分析，驾驶员行为分析 6 种接口方式

接口类型	接口需求	接口参数
人脸活体检测接口	将**人脸检测与对齐**、**质量检测**和**多模态活体检测**功能封装到一个接口内，用户可以通过调用该接口完成图像的活体判断。活体判断的结果为多模态联合判断，即：各种模态结果全部为真则结果为真人，任何一种模态判断为假则结果为假。	人脸检测：『最小检测人脸』 质量检测：『是否开启质量检测』『质量检测阈值』 活体检测：『活体检测模态』『活体检测阈值』
人脸特征抽取接口	将**人脸检测与对齐**、**质量检测**、**多模态活体检测**和**人脸特征抽取**功能封装到一个接口内，用户可以通过调用该接口完成图像中人脸特征抽取。可以通过制定的特征值抽取模型完成人脸特征的抽取。	人脸检测：『最小检测人脸』 质量检测：『是否开启质量检测』『质量检测阈值』 活体检测：『活体检测模态』『活体检测阈值』 特征抽取：『特征抽取模型』
1:1人脸比对接口	基于**『人脸特征值抽取接口』**完成人脸的1:1比对，用户通过调用该接口可以完成两个图像人脸相似度的比较，返回结果为两个图像的人脸相似度。	人脸检测：『最小检测人脸』 质量检测：『是否开启质量检测』『质量检测阈值』 活体检测：『活体检测模态』『活体检测阈值』 特征抽取：『特征抽取模型』 特征比对：『特征比对阈值』
人脸属性分析接口	将**人脸检测与对齐**、**人脸属性分析**功能封装到一个接口内，用户可以通过调用该接口得到图像中人脸的年龄、性别、是否佩戴眼镜、是否佩戴口罩等属性信息	人脸检测：『最小检测人脸』
驾驶员行为分析接口	将**人脸检测与对齐**、**驾驶员行为分析**功能封装到一个接口内，用户可以通过调用该接口得到图象中人脸的驾驶员相关属性：抽烟、进食、饮水、使用手机、注意力、安全带等	人脸检测：『最小检测人脸』

（4）dataLibrary 数据库 mode：

datalibrary 模块为数据库管理模块，其他 9 种功能模块需继承这个模块。



（图 4）

（5）9 个功能页面 module：

以下 9 模块属于功能页面模块（**看 1 图**），对应 demoAPP 首页的每个模块入口,跳转可进入相应的功能页面。9 个功能模块分别依赖都 facelibrary 模块（**facelibrary 请看 2**）

（1）attendancelibrary 考勤模块

（2）attrbutelibrary 属性模块

（3）drivermonitorlibrary 驾驶行为模块

（4）financelibrary 金融模块

（5）gatelibrary 闸机模块

（6）gazelibrary 注意力模块

（7）identifylibrary 认证核验模块

（8）paymentlibrary 支付模块

（9）registerlibrary 人脸注册和人脸管理模块

（人脸 1:N 识别功能请参考考勤，闸机，支付模块）

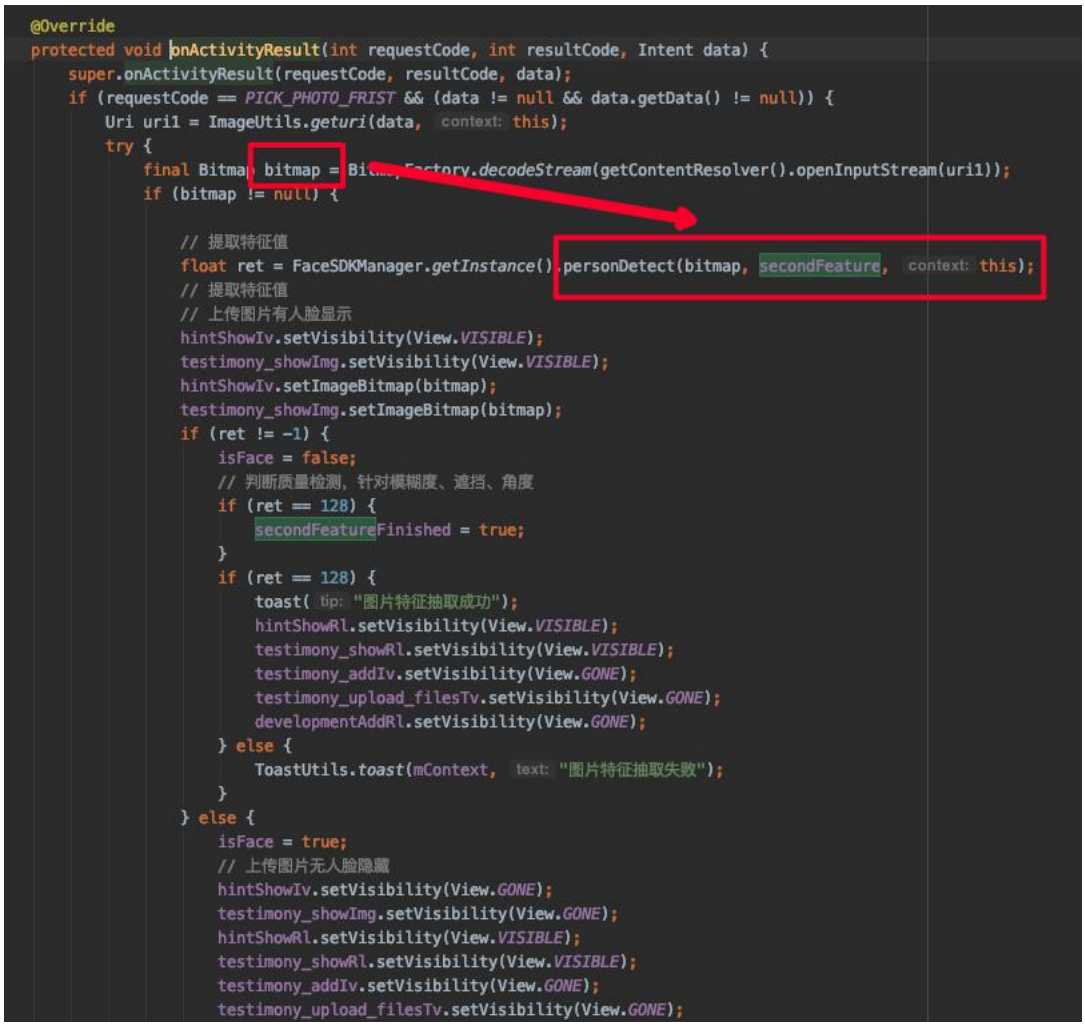
（人脸 1:1 识别功能请参考人证核验模块）

1》 1:1 识别功能-认证核验使用说明：

认证核验镜头模态和考勤，闸机，支付一样，分别为（1.单目 RGB，2.RGB + NIR, 3.RGB + DEPTH，开启双目摄像头方式请参考快速集成上手）,使用方式是：本地人像图片和预览流人像做对比，获取一个相似度，如果置信度大于阈值就认为是同一个人。

```
private void initListener() {
    if (FaceSDKManager.initStatus != FaceSDKManager.SDK_MODEL_LOAD_SUCCESS) {
        FaceSDKManager.getInstance().initModel( context: this, new SdkInitListener() {
            @Override
            public void initStart() {
            }

            @Override
            public void initLicenseSuccess() {
            }

            @Override
            public void initLicenseFail(int errorCode, String msg) {
            }

            @Override
            public void initModelSuccess() {
                FaceSDKManager.initModelSuccess = true;
                ToastUtils.toast( context: FaceRGBPersonActivity.this,  text: "模型加载成功，欢迎使用");
            }

            @Override
            public void initModelFail(int errorCode, String msg) {
                FaceSDKManager.initModelSuccess = false;
                if (errorCode != -12) {
                    ToastUtils.toast( context: FaceRGBPersonActivity.this,  text: "模型加载失败，请尝试重启应用");
                }
            }
        });
    }
}
```

1. 初始化加载模型



```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == PICK_PHOTO_FRIST && (data != null && data.getData() != null)) {
        Uri uri1 = ImageUtils.geturi(data, context: this);
        try {
            final Bitmap bitmap = BitmapFactory.decodeStream(getContentResolver().openInputStream(uri1));
            if (bitmap != null) {

                // 提取特征值
                float ret = FaceSDKManager.getInstance().personDetect(bitmap, secondFeature, context: this);
                // 提取特征值
                // 上传图片有人脸显示
                hintShowIv.setVisibility(View.VISIBLE);
                testimony_showImg.setVisibility(View.VISIBLE);
                hintShowIv.setImageBitmap(bitmap);
                testimony_showImg.setImageBitmap(bitmap);
                if (ret != -1) {
                    isFace = false;
                    // 判断质量检测，针对模糊度、遮挡、角度
                    if (ret == 128) {
                        secondFeatureFinished = true;
                    }
                    if (ret == 128) {
                        toast( tip: "图片特征抽取成功");
                        hintShowRl.setVisibility(View.VISIBLE);
                        testimony_showRl.setVisibility(View.VISIBLE);
                        testimony_addIv.setVisibility(View.GONE);
                        testimony_upload_filesTv.setVisibility(View.GONE);
                        developmentAddRl.setVisibility(View.GONE);
                    } else {
                        ToastUtils.toast(mContext, text: "图片特征抽取失败");
                    }
                } else {
                    isFace = true;
                    // 上传图片无人脸隐藏
                    hintShowIv.setVisibility(View.GONE);
                    testimony_showImg.setVisibility(View.GONE);
                    hintShowRl.setVisibility(View.VISIBLE);
                    testimony_showRl.setVisibility(View.VISIBLE);
                    testimony_addIv.setVisibility(View.GONE);
                    testimony_upload_filesTv.setVisibility(View.GONE);
```

2. 获取设备本地相机的人像图片作为 bitmap,提取这张 bitmap 的人脸特征值 A，

```
private void startTestOpenDebugRegisterFunction() {
    // TODO : 临时放置
    //  CameraPreviewManager.getInstance().setCameraFacing(CameraPreviewManager.CAMERA_USB);
    if (SingleBaseConfig.getBaseConfig().getRBGCameraId() != -1){
        CameraPreviewManager.getInstance().setCameraFacing(SingleBaseConfig.getBaseConfig().getRBGCameraId());
    }else {
        CameraPreviewManager.getInstance().setCameraFacing(CameraPreviewManager.CAMERA_FACING_FRONT);
    }
    CameraPreviewManager.getInstance().startPreview(mContext, mAutoCameraPreviewView,
            PREFER_WIDTH, PERFER_HEIGH, (data, camera, width, height) -> {
                // 摄像头预览数据进行人脸检测
                FaceSDKManager.getInstance().onDetectCheck(data,  nirData: null,  depthData: null,
                        height, width, mLiveType, new FaceDetectCallBack() {
                            @Override
                            public void onFaceDetectCallback(LivenessModel livenessModel) {
                                // 预览模式
                                checkCloseDebugResult(livenessModel);

                                // 开发模式
                                checkOpenDebugResult(livenessModel);
                            }

                            @Override
                            public void onTip(int code, String msg) {
                            }

                            @Override
                            public void onFaceDetectDarwCallback(LivenessModel livenessModel) {
```

3. 开启摄像头，调用人脸检测入口 onDetectCheck()

```
FaceSDKManager.java
e = System.currentTimeMillis();                    0 results
                                                   Replace  Replace all  Exclude
                                                   ⚠ 50  ⚠ 1  ✓ 65
1101                return;
1102            }
1103            livenessModel.setBdFaceImageInstanceCrop(cropInstance);
1104        }
1105
1106        long startCompareTime = System.currentTimeMillis();
1107        float score = FaceSDKManager.getInstance().getFaceFeature().featureCompare(
1108                BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_ID_PHOTO,
1109                livenessModel.getFeature(), secondFeature,  isPercent: true);
1110        livenessModel.setScore(score);
1111        if (score > SingleBaseConfig.getBaseConfig().getIdThreshold()) {
1112            /*faceId = livenessModel.getFaceInfo().faceID;
1113            trackTime = System.currentTimeMillis();
1114            faceAdoptModel = livenessModel;
1115            failNumber =
1116            isFail = false;*/
1117            setFail(livenessModel);
1118        } else{
1119            setFail(livenessModel);
1120        }
1121
```

比对接口

特征A

特征b

4. FaceSDKManager.getInstance().onDetectCheck()

人脸识别功能检测入口后会回调人脸检测识别结果返回（LivenessModel 里面可以获取到特征值 B）

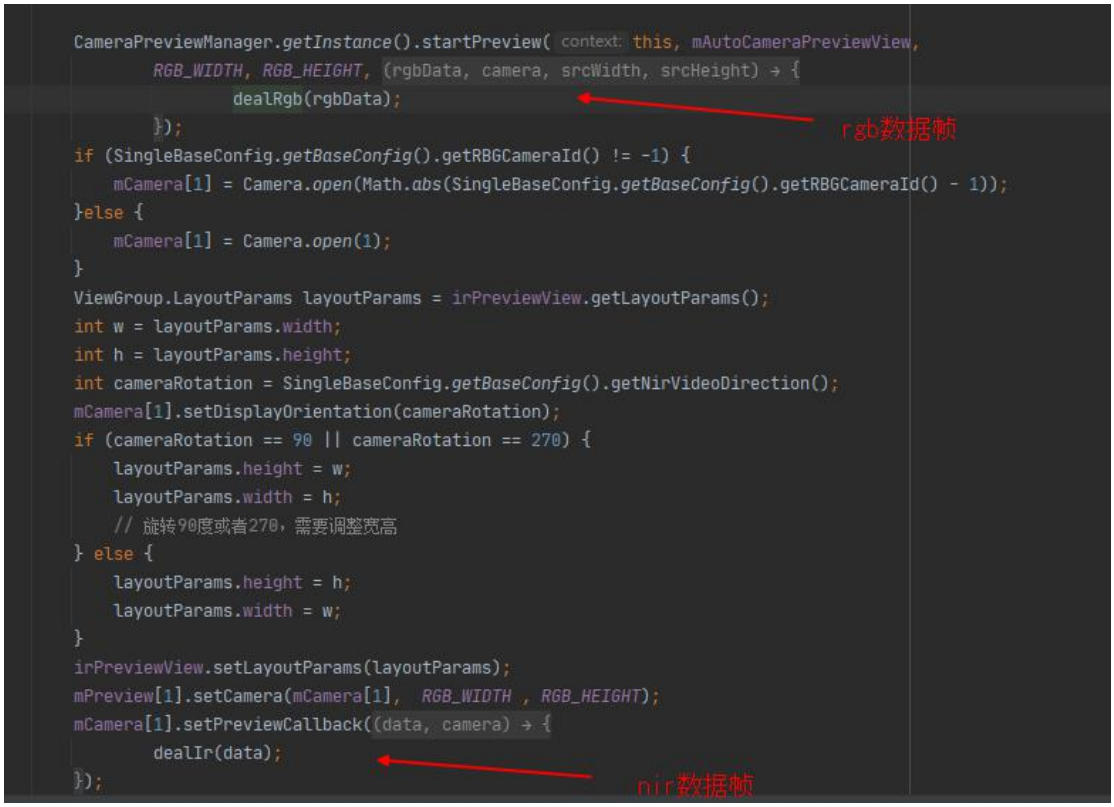
```
if (score > SingleBaseConfig.getBaseConfig().getIdThreshold()) {
    textCompareStatus.setTextColor(Color.parseColor( colorString: "#00BAF2"));
    textCompareStatus.setText("比对成功");
} else {
    textCompareStatus.setTextColor(Color.parseColor( colorString: "#FECD33"));
    textCompareStatus.setText("比对失败");
}
```

5. 最后调用 FaceSDKManager.getInstance().getFaceFeature().featureCompare()获取当前置信度大于默认阈值认为是同一个人，做相应文案提示。

（驾驶行为，注意力，属性模块类似。都是只有检测能力，不做识别能力。

区别：驾驶行为用双目镜头（rgb + nir），其他两个模块是单目 rgb.）

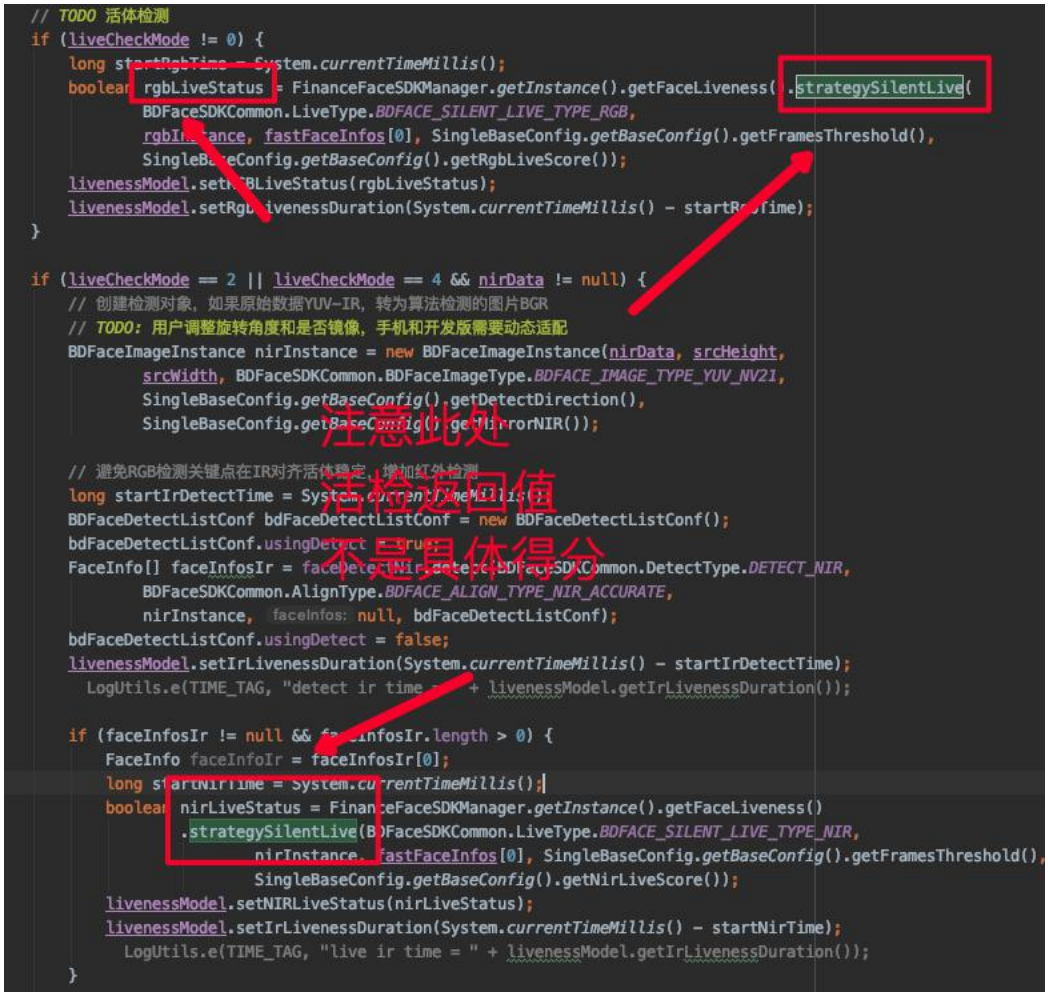
2》驾驶行为模块使用说明：


CameraPreviewManager.getInstance().startPreview(context: this, mAutoCameraPreviewView,
RGB_WIDTH, RGB_HEIGHT, (rgbData, camera, srcWidth, srcHeight) -> {
dealRgb(rgbData);
});
if (SingleBaseConfig.getBaseConfig().getRBGCameraId() != -1) {
mCamera[1] = Camera.open(Math.abs(SingleBaseConfig.getBaseConfig().getRBGCameraId() - 1));
}else {
mCamera[1] = Camera.open(1);
}
ViewGroup.LayoutParams layoutParams = irPreviewView.getLayoutParams();
int w = layoutParams.width;
int h = layoutParams.height;
int cameraRotation = SingleBaseConfig.getBaseConfig().getNirVideoDirection();
mCamera[1].setDisplayOrientation(cameraRotation);
if (cameraRotation == 90 || cameraRotation == 270) {
layoutParams.height = w;
layoutParams.width = h;
// 旋转90度或者270，需要调整宽高
} else {
layoutParams.height = h;
layoutParams.width = w;
}
irPreviewView.setLayoutParams(layoutParams);
mPreview[1].setCamera(mCamera[1], RGB_WIDTH , RGB_HEIGHT);
mCamera[1].setPreviewCallback((data, camera) -> {
dealIr(data);
});
rgb数据帧
nir数据帧

驾驶行为检测，只有单纯的检测功能和 RGB 活检，用于检测行为和注意力。不做识别功能。

3》 金融活检模块说明：

镜头开启和数据流获取，请参考（快速集成上手）



此模块支持检测帧数调整，默认为 2 帧（2 帧人脸都通过本次活体结果才为通过，注意上调帧数会导致活检过程变慢），无特征提取和对比识别功能。

4》 注册人脸库模块说明

（1） 人脸注册有 3 种镜头模式选择注册（注意人脸注册镜头模式以考勤，闸机，支付，认证核验模块其中最后一个设置的镜头模式应用于人脸注册，）

(2) 检测相关：

```
/**
 * 摄像头数据处理
 */
private void faceDetect(byte[] data, final int width, final int height) {
    if (mCollectSuccess) {
        return;
    }

    // 摄像头预览数据进行人脸检测
    int liveType = SingleBaseConfig.getBaseConfig().getType();
    // int liveType = 2;

    if (liveType == 0) { // 无活体检测
        FaceTrackManager.getInstance().setAliving(false);
    } else if (liveType == 1) { // 活体检测
        FaceTrackManager.getInstance().setAliving(true);
    }

    // 摄像头预览数据进行人脸检测
    FaceTrackManager.getInstance().faceTrack(data, width, height, new FaceDetectCallBack() {
        @Override
        public void onFaceDetectCallback(LivenessModel livenessModel) {
            checkFaceBound(livenessModel);
        }
```

(3) 活体相关：

```
if (faceInfos != null && faceInfos.length > 0) {
    livenessModel.setTrackFaceInfo(faceInfos);
    FaceInfo faceInfo = faceInfos[0];
    livenessModel.setFaceInfo(faceInfo);
    livenessModel.setLandmarks(faceInfo.landmarks);
    Log.e(TAG,  msg: "ldmk = " + faceInfo.landmarks.length);

    // 质量检测，针对模糊度、遮挡、角度
    if (onQualityCheck(livenessModel, faceDetectCallBack)) {
        // 活体检测
        if (isAliving) {
            startTime = System.currentTimeMillis();
            float rgbScore = FaceSDKManager.getInstance().getFaceLiveness().silentLive(
                    BDFaceSDKCommon.LiveType.BDFACE_SILENT_LIVE_TYPE_RGB,
                    rgbInstance, faceInfo.landmarks);
            Log.e(TAG,  msg: "score = " + rgbScore);
            livenessModel.setRgbLivenessScore(rgbScore);
            livenessModel.setRgbLivenessDuration(System.currentTimeMillis() - startTime);
        }
        // 流程结束销毁图片，开始下一帧图片检测，否着内存泄露
        // rgbInstance.destory();
        if (faceDetectCallBack != null) {
            faceDetectCallBack.onFaceDetectCallback(livenessModel);
        }
    } else {
        // 流程结束销毁图片，开始下一帧图片检测，否着内存泄露
        rgbInstance.destory();
    }
```

(4) 人脸特征提取

```
/**
 * 提取特征值
 *
 * @param model 人脸数据
 */
private void getFeatures(final LivenessModel model) {
    if (model == null) {
        return;
    }

    // 获取选择的特征抽取模型
    int modelType = SingleBaseConfig.getBaseConfig().getActiveModel();
    if (modelType == 1) {
        // 生活照
        FaceSDKManager.getInstance().onFeatureCheck(model.getBdFaceImageInstance(), model.getLandmarks(),
                BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_LIVE_PHOTO, new FaceFeatureCallBack() {
                    @Override
                    public void onFaceFeatureCallBack(final float featureSize, final byte[] feature, long time) {
                        runOnUiThread(() -> {
                                if (mCollectSuccess) {
                                    return;
                                }
                                displayCompareResult(featureSize, feature, model);
                                Log.e(TAG, String.valueOf(feature.length));
                            });

                    }
                });
    }
}
```

(5) 提取完毕的特征人脸信息，注册到本地数据库

```
// 特征提取成功
if (ret == 128) {
    // 抠图
    BDFaceImageInstance imageInstance = model.getBdFaceImageInstanceCrop();
    AtomicInteger isOutoBoundary = new AtomicInteger();
    BDFaceImageInstance cropInstance = FaceSDKManager.getInstance().getFaceCrop()
            .cropFaceByLandmark(imageInstance, model.getLandmarks(),
                    enlargeRatio: 2.0f,  correction: false, isOutoBoundary);
    if (cropInstance == null) {
        mFaceRoundProView.setTipText("抠图失败");
        mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_red);
        // 释放内存
        destroyImageInstance(model.getBdFaceImageInstanceCrop());
        return;
    }
    mCropBitmap = BitmapUtils.getInstaceBmp(cropInstance);
    // 获取头像
    if (mCropBitmap != null) {
        mCollectSuccess = true;
        mCircleHead.setImageBitmap(mCropBitmap);
    }
    cropInstance.destory();
    // 释放内存
    destroyImageInstance(model.getBdFaceImageInstanceCrop());

    mRelativeCollectSuccess.setVisibility(View.VISIBLE);
    mRelativePreview.setVisibility(View.GONE);
    mFaceRoundProView.setTipText("");

    for (int i = 0; i < faceFeature.length; i++) {
        mFeatures[i] = faceFeature[i];
    }
} else {
    mFaceRoundProView.setTipText("特征提取失败");
    mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_red);
}
```

人脸特征信息

```
} else if (id == R.id.btn_collect_confirm) {   // 用户名注册
    String userName = mEditName.getText().toString();
          if (TextUtils.isEmpty(userName)) {
              ToastUtils.toast(getApplicationContext(), "请先输入用户名");
              return;
          }
          if (userName.length() > 10) {
              ToastUtils.toast(getApplicationContext(), "用户名长度不得大于10位");
              return;
          }
    // 姓名过滤
    String nameResult = FaceApi.getInstance().isValidName(userName);
    if (!"0".equals(nameResult)) {
        ToastUtils.toast(getApplicationContext(), nameResult);
        return;
    }
    String imageName = userName + ".jpg";
    // 注册到人脸库
    boolean isSuccess = FaceApi.getInstance().registerUserIntoDBmanager( groupName: null
            userName, imageName,  userInfo: null, mFeatures);
    if (isSuccess) {
        // 保存人脸图片
        File faceDir = FileUtils.getBatchImportSuccessDirectory();
        File file = new File(faceDir, imageName);
        FileUtils.saveBitmap(file, mCropBitmap);
        // 数据变化，更新内存
        FaceApi.getInstance().initDatabases( isFeaturePush: true);
        // 更新UI
        mRelativeCollectSuccess.setVisibility(View.GONE);
        mRelativeRegisterSuccess.setVisibility(View.VISIBLE);
        mCircleRegSucHead.setImageBitmap(mCropBitmap);
    } else {
        ToastUtils.toast(getApplicationContext(),  text: "保存数据库失败，" +
                "可能是用户名格式不正确");
    }
```

人脸信息添加到本地数据库

注意：1.用户名为必填项，支持英文 汉字 数字 和下划线。

2.注册成功后的图片命名和格式为设置名.jpg,保存在 sd 卡下的 Success-Import 文件夹下，用于人脸库的图片显示。

3.demo 使用安卓原生数据库，开发者可根据自己的业务需求场景使用其他安卓开源数据库，添加保存人脸信息。

二：快速集成上手使用（1:N 识别功能）

1.新建工程项目依赖 facelibrary module

2.把 demo app libs 下的 arr 和 jar 放到自己的工程 app libs 目录。复制 demo AndroidManifest.xml 相应的安卓权限和介入配置到自己的工程中（安卓 6.0 需要动态申请相应的权限参考 demo 里 BaseActivity 类）

3. 选择相应的激活方式：

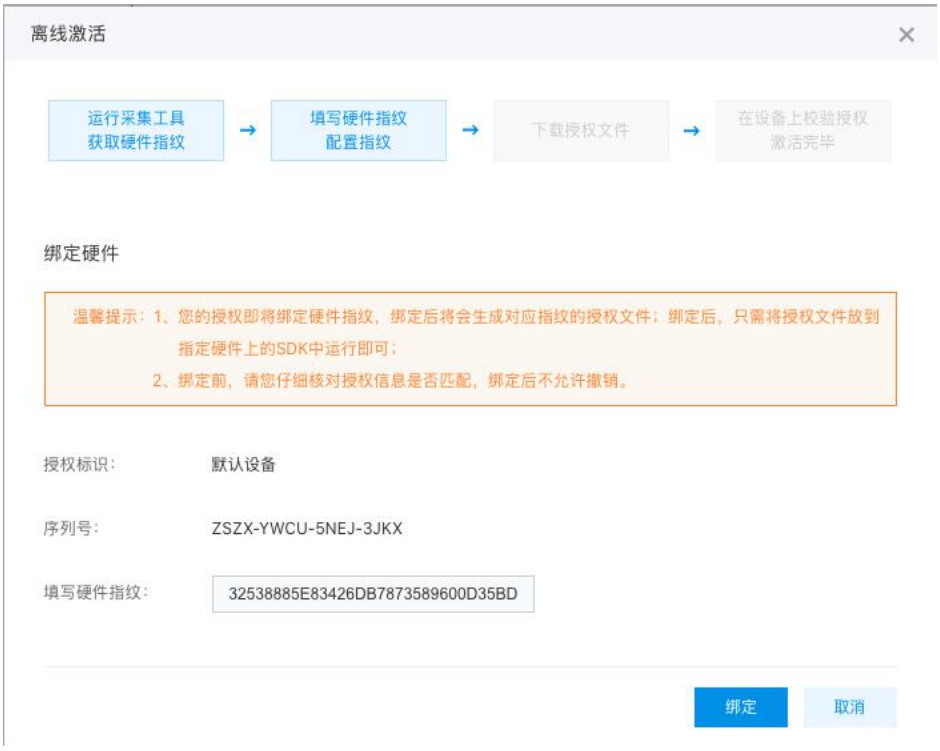
分为离线激活和在线激活（在线激活分两种：1.单设备授权。2.批量授权）

（1）在线激活


```
//  在线激活方式
//  licenseID :  官网申请的licenseID
faceAuth.initLicenseOnLine( context: this,  licenseID "xxxx-xxxx-xxxx-xxxx"  new Callback() {
    @Override
    public void onResponse(int code, String s) {
        if(code == 0){
            // 在线激活成功，跳转页面

        }
    }
});
```

（2）离线激活

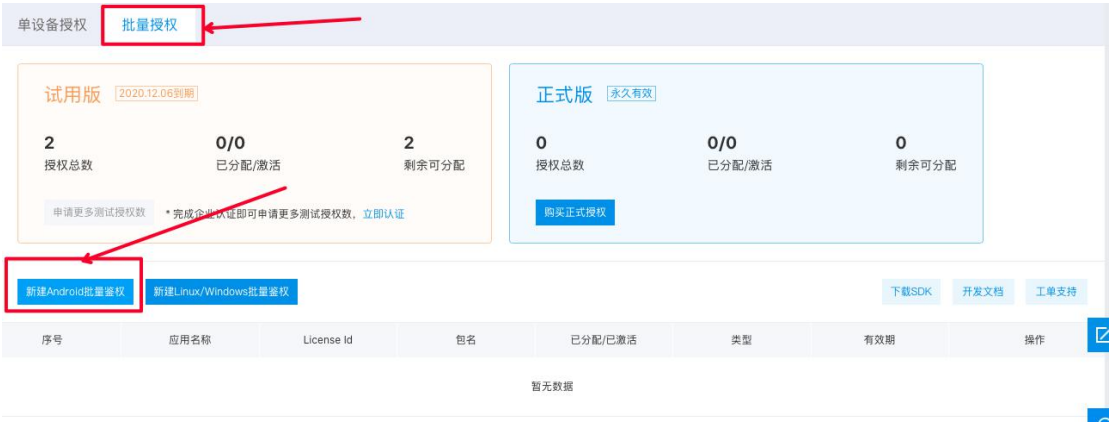

离线激活
运行采集工具
获取硬件指纹
填写硬件指纹
配置指纹
下载授权文件
在设备上校验授权
激活完毕
SDK初始化时会自动读取硬件指纹
请复制或记录完毕，进入下一步
下一步
取消

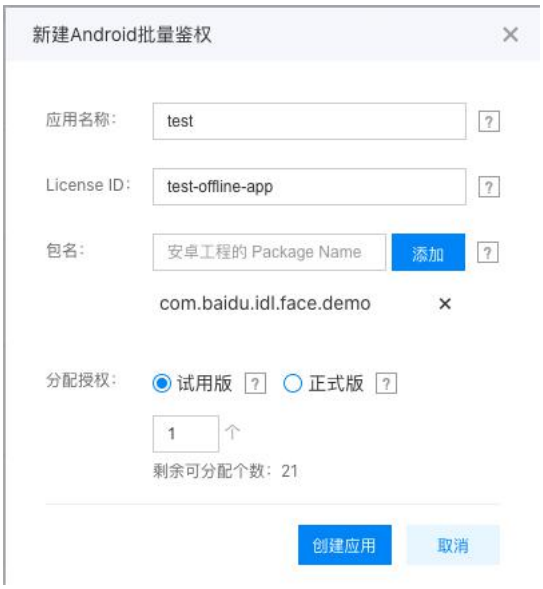

离线激活
运行采集工具
获取硬件指纹
填写硬件指纹
配置指纹
下载授权文件
在设备上校验授权
激活完毕
绑定硬件
温馨提示：1、您的授权即将绑定硬件指纹，绑定后将会生成对应指纹的授权文件；绑定后，只需将授权文件放到指定硬件上的SDK中运行即可；
2、绑定前，请您仔细核对授权信息是否匹配，绑定后不允许撤销。
授权标识：
默认设备
序列号：
ZSZX-YWCU-5NEJ-3JKX
填写硬件指纹：
32538885E83426DB7873589600D35BD
绑定
取消

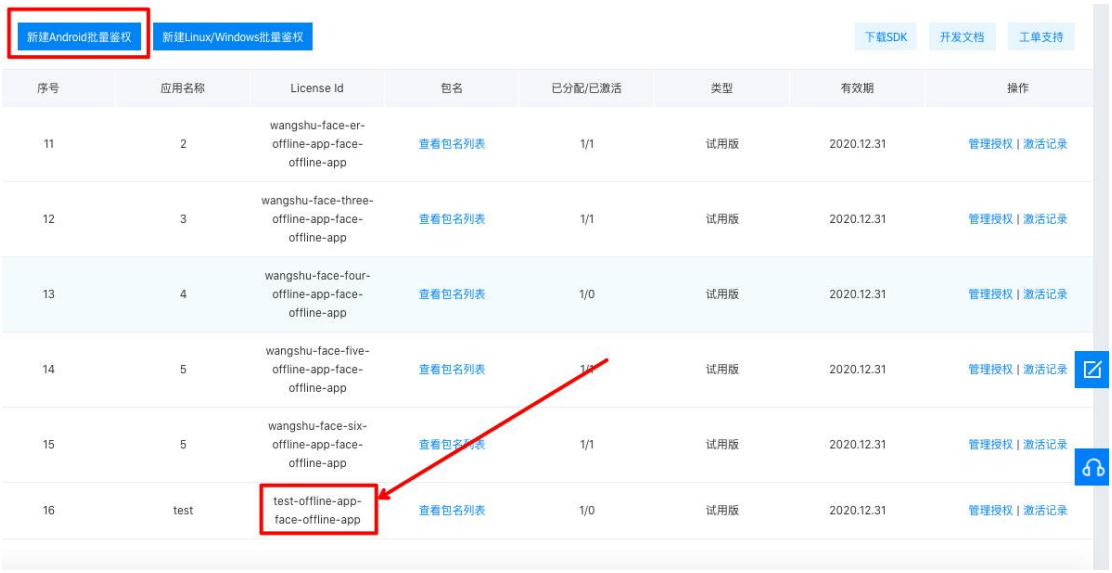

离线激活
运行采集工具
获取硬件指纹
填写硬件指纹
配置指纹
下载授权文件
在设备上校验授权
激活完毕
绑定硬件成功
请下载授权文件，将授权文件放到SDK指定位置即可，详情可查看SDK文档。
下载授权文件


```
//  离线激活方式
faceAuth.initLicenseOffLine( context: this, new Callback() {
    @Override
    public void onResponse(int code, String s) {
        if(code == 0){
            // 离线激活成功，跳转页面
        }
    }
});
```

（3）批量激活（属于在线激活）







详细请参考 ActivitionActivity 激活方式调用。

4. 激活成功后，初始化模型。

```
private void initListener() {
    if (FaceSDKManager.initStatus != FaceSDKManager.SDK_MODEL_LOAD_SUCCESS) {
        FaceSDKManager.getInstance().initModel( context: this, new SdkInitListener() {
            @Override
            public void initStart() {
            }

            @Override
            public void initLicenseSuccess() {
            }

            @Override
            public void initLicenseFail(int errorCode, String msg) {
            }

            @Override
            public void initModelSuccess() {
                FaceSDKManager.initModelSuccess = true;
                ToastUtils.toast( context: FaceRGBAttendanceActivity.this,  text: "模型加载成功，欢迎使用");
            }

            @Override
            public void initModelFail(int errorCode, String msg) {
                FaceSDKManager.initModelSuccess = false;
                if (errorCode != -12) {
                    ToastUtils.toast( context: FaceRGBAttendanceActivity.this,  text: "模型加载失败，请尝试重启应用");
                }
            }
        });
    }
```

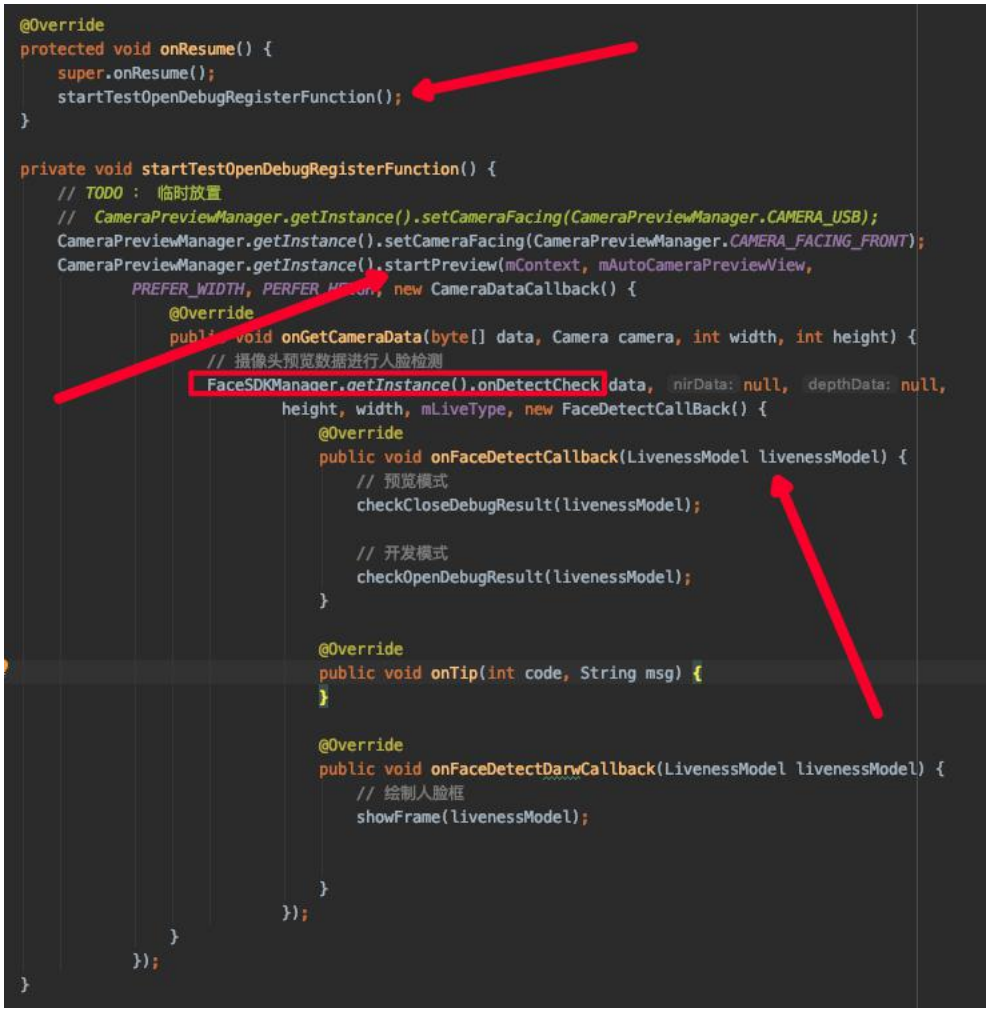
5. 初始化模型成功后，选择自己人脸识别的镜头模态。

(1.单目 RGB，2.RGB + NIR, 3.RGB + DEPTH)

****

(1) 例如: 考勤模块单目 RGB

(FaceRGBAttendanceActivity.class)



1.CameraPreviewManager.getInstance().startPreview()开启 RGB 摄像头预览，摄像头每一帧的图片数据 data 用于人脸检测识别。

2.FaceSDKManager.getInstance().onDetectCheck()人脸识别功能检测入口，及人脸检测识别结果返回。

（2） **例如: 考勤模块双目 RGB + NIR**

```
super.onResume();
if (mCameraNum < 2) {
    Toast.makeText( context: this,  text: "未检测到2个摄像头", Toast.LENGTH_LONG).show();
    return;
} else {
    try {
        startTestCloseDebugRegisterFunction();
        if (SingleBaseConfig.getBaseConfig().getRBGCameraId() != -1) {
            mCamera[1] = Camera.open(Math.abs(SingleBaseConfig.getBaseConfig().getRBGCameraId() - 1));
        }else {
            mCamera[1] = Camera.open(1);
        }
        ViewGroup.LayoutParams layoutParams = irPreviewView.getLayoutParams();
        int w = layoutParams.width;
        int h = layoutParams.height;
        int cameraRotation = SingleBaseConfig.getBaseConfig().getNirVideoDirection();
        mCamera[1].setDisplayOrientation(cameraRotation);
        if (cameraRotation == 90 || cameraRotation == 270) {
            layoutParams.height = w;
            layoutParams.width = h;
            // 旋转90度或者270，需要调整宽高
        } else {
            layoutParams.height = h;
            layoutParams.width = w;
        }
        irPreviewView.setLayoutParams(layoutParams);
        mPreview[1].setCamera(mCamera[1], PREFER_WIDTH, PERFER_HEIGH);
        mCamera[1].setPreviewCallback((data, camera) -> {
                dealIr(data);
        });
```

开启rgb显示

开启nir显示

nir图片帧回调

```
    // 设置USB摄像头
    if (SingleBaseConfig.getBaseConfig().getRBGCameraId() != -1){
        CameraPreviewManager.getInstance().setCameraFacing(SingleBaseConfig.getBaseConfig().getRBGCameraId());
    }else {
        CameraPreviewManager.getInstance().setCameraFacing(CameraPreviewManager.CAMERA_USB);
    }

    CameraPreviewManager.getInstance().startPreview( context: this, mAutoCameraPreviewView,
            PREFER_WIDTH, PERFER_HEIGH, (data, camera, width, height) -> {
                // 摄像头预览数据进行人脸检测
                dealRgb(data);
            });
```

sdk获取的rgb视频流

若未识别出则调取默认摄像头

rgb数据帧回掉

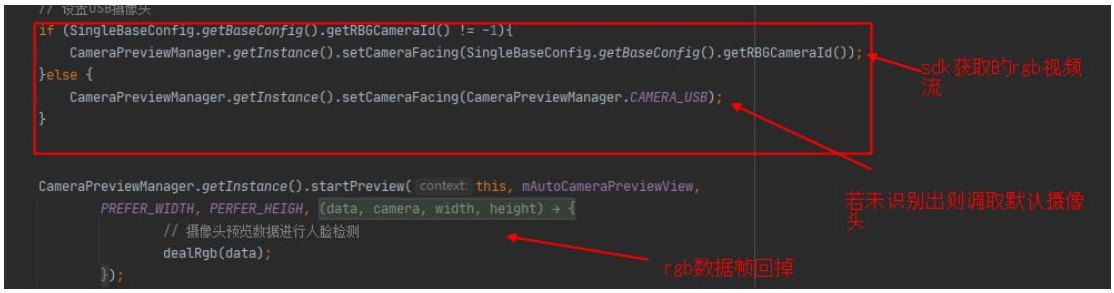
```
private synchronized void checkData() {
    if (rgbData != null && irData != null) {
        FaceSDKManager.getInstance().onDetectCheck(rgbData, irData,  depthData: null,
                PERFER_HEIGH, PREFER_WIDTH,  liveCheckMode: 2, new FaceDetectCallBack() {
                    @Override
                    public void onFaceDetectCallback(LivenessModel livenessModel) {
                        // 输出结果
                        // 预览模式
                        checkCloseDebugResult(livenessModel);
                        // 开发模式
                        checkOpenDebugResult(livenessModel);

                    }

                    @Override
                    public void onTip(int code, String msg) {
                    }

                    @Override
                    public void onFaceDetectDarwCallback(LivenessModel livenessModel) {
                        // 绘制人脸框
                        showFrame(livenessModel);
                    }
                });
        rgbData = null;
        irData = null;
    }
```

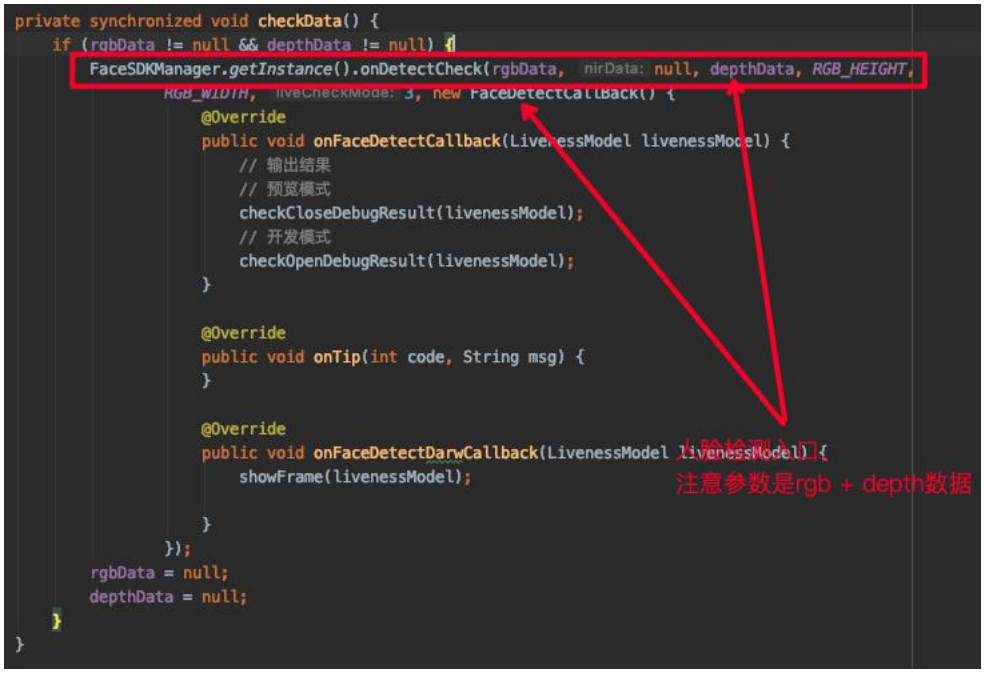
人脸检测入口

这里是 RGB + NIR 模态，其中 RGB 和上面单目 rgb 的打开镜头以及获取预览流数据基本一样（注意不同点的标注 CameraPreviewManager.getInstance().setCameraFacing(CameraPreviewManager.CAMERA_USB)），然后新增 NIR 镜头的打开方式和获取预览流数据的方式。

（详细代码参考 FaceNIRAttendanceActivity.class）

（3） 例如：考勤模块双目 RGB + Depth





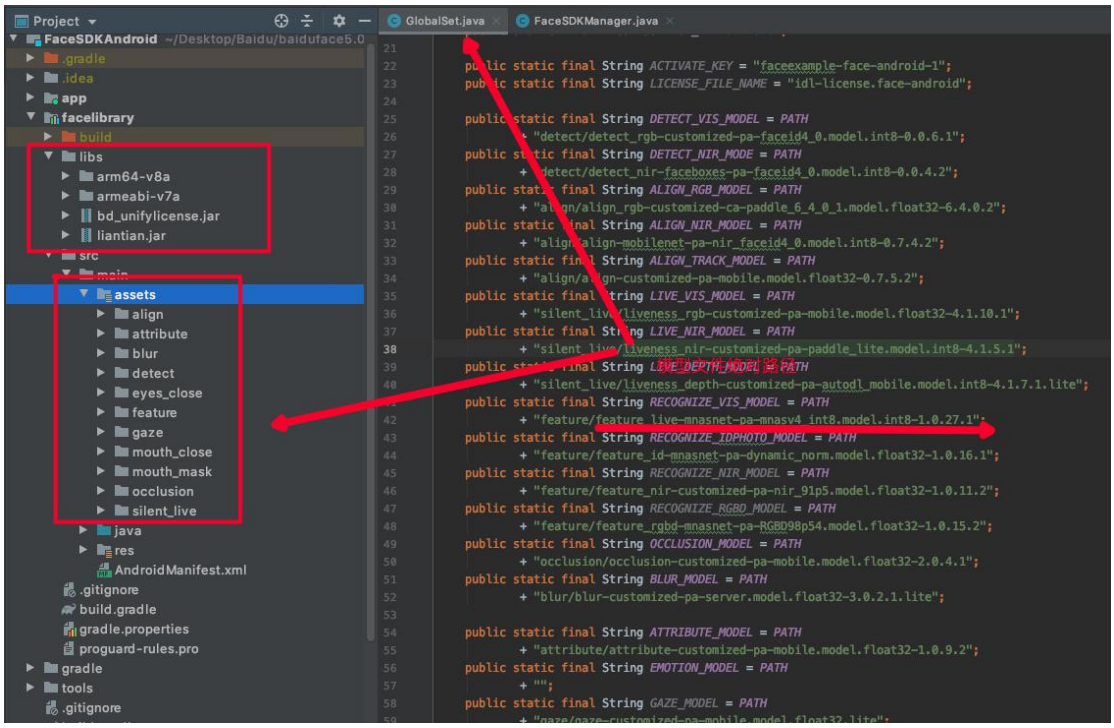
这里是 RGB + Depth 模态，其中 RGB 和上面单目 rgb 的打开镜头以及获取预览流数据基本一样（注意不同点的标注 CameraPreviewManager.getInstance().setCameraFacing(CameraPreviewManager.CAMERA_USB)），然后新增 Depth 镜头的打开方式和获取预览流数据的方式。

（详细代码参考 FaceDepthAttendanceActivity.class）

三：升级指导

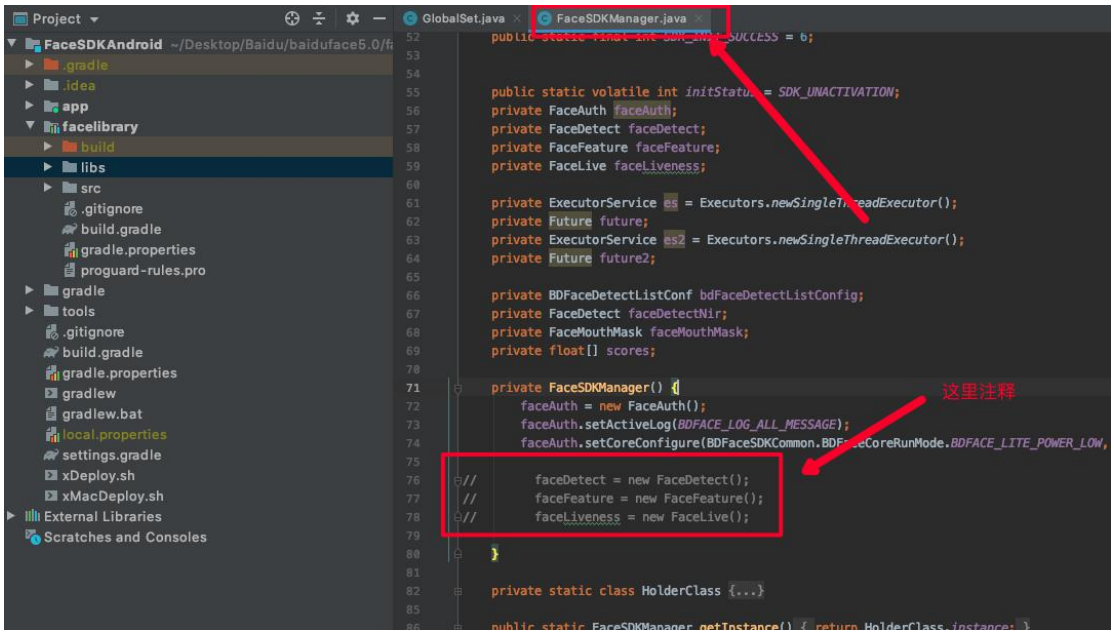
(1)3.v – 4.v

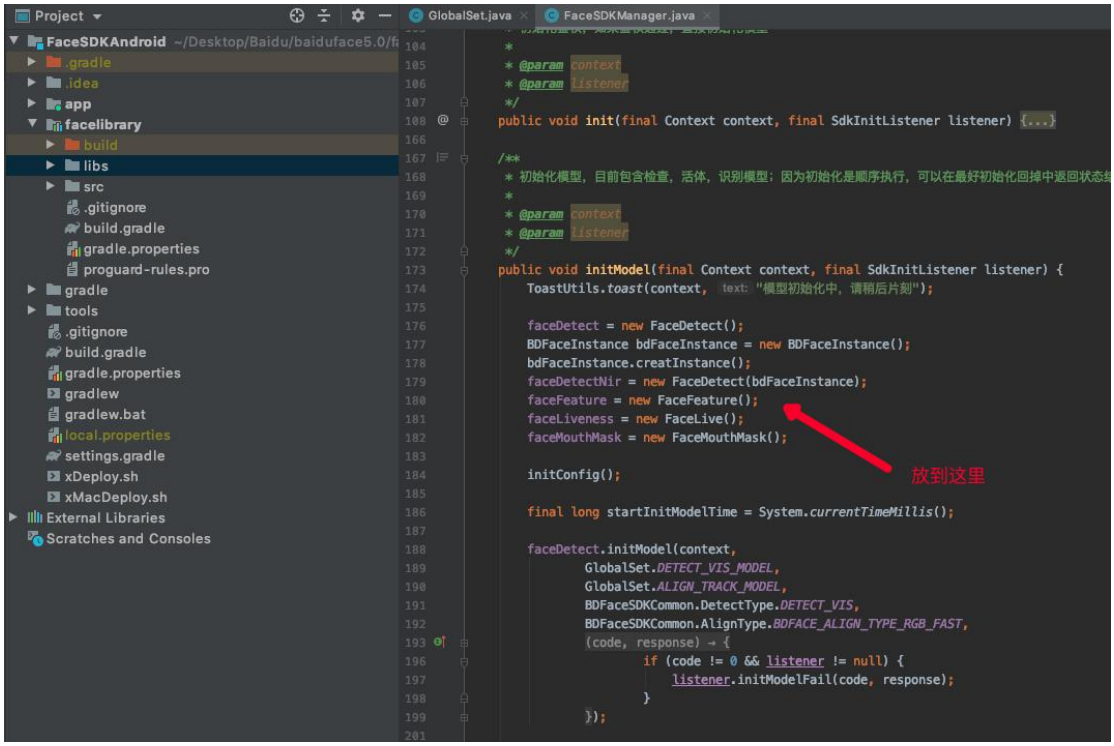
注意（更新版本要刷新人脸底库，重新导入人脸）



1. libs 下所有 jar 和文件替换成 4.0 的
2. assets 下所有模型文件替换成 4.0 的,

   GlobalSet 模型路径要对应到 assets 下相应的文件夹

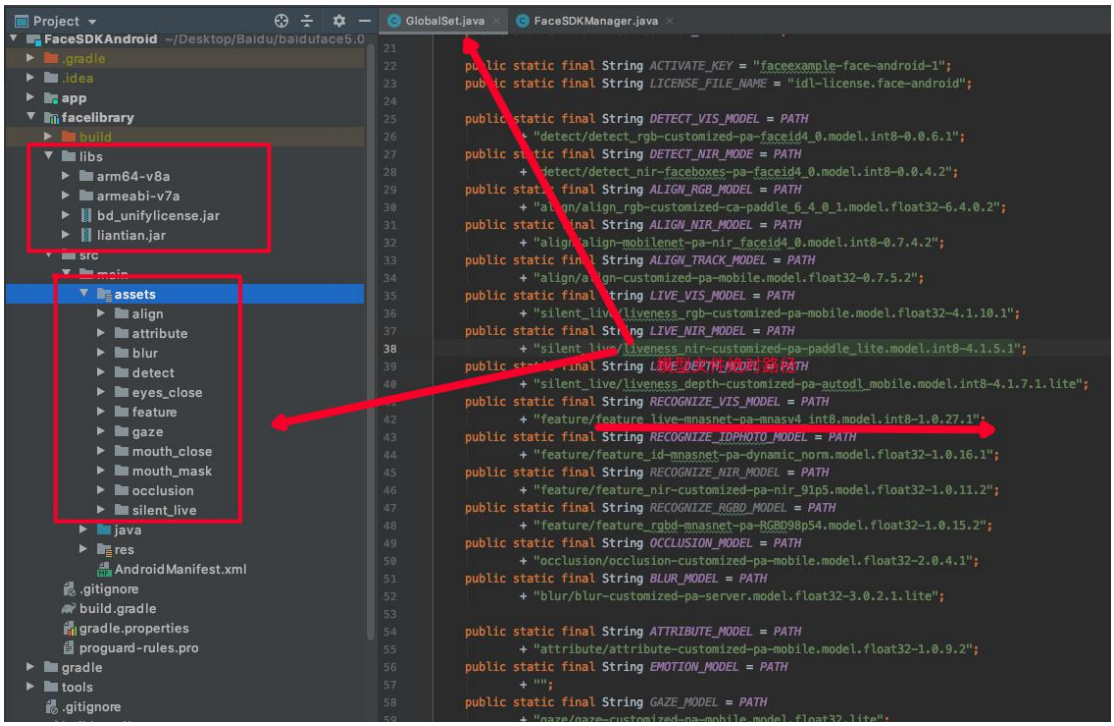




根据上方注释掉 FaceSDKManager 标注的代码。

(2)4.v – 5.v

注意（更新版本要刷新人脸底库，重新导入人脸）



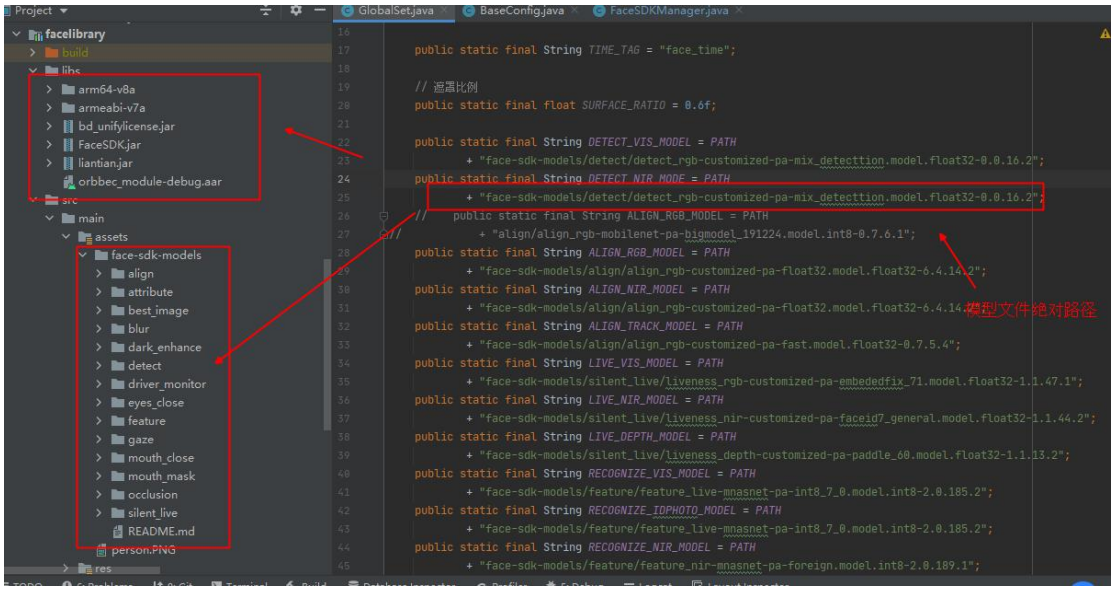
1.libs 下所有 jar 和文件替换成 5.0 的

2.assets 下所有模型文件替换成 5.0 的,

GlobalSet 模型路径要对应到 assets 下相应的文件夹

V6-v7.1

注意（更新版本要刷新人脸底库，重新导入人脸）



.libs 下所有 jar 和文件替换成 7.1 的

2.assets 下所有模型文件替换成 7.1 的,

GlobalSet 模型路径要对应到 assets 下相应的文件夹

四：常见问题

Q：批量授权分配授权次数问题?

A:

1）1 个 licenseID 可以绑定多个包名，如果运行到 1 台设备，即分配授权次数减少 1

2）1 个 licenseID 绑定 1 个包名，如果运行在 10 台设备上，即分配授权次数减少 10

Q:RGB 人脸检测预览流和 RGB 视频流回显图人脸角度不正?

A:如果人脸角度和视频流回显角度不正，请到各自模块设置页面的（镜头设置 - 人脸检测角度和视频流回显角度）调整

Q:人脸框和视频流镜像问题?

A:如果人脸的移动方向和预览流里的人脸移动相反，请到各自模块设置页面的（镜头设置 - 镜像设置 - 检测框镜像）调整。

Q: 展示人脸图失败的问题

```
if (getCropFace()) {
    BDFaceCropParam cropParam = new BDFaceCropParam();
    cropParam.foreheadExtend = 2.0f / 9;
    cropParam.chinExtend = 1.0f / 9;
    cropParam.enlargeRatio = 1.5f;
    cropParam.height = 640;
    cropParam.width = 480;
    BDFaceImageInstance cropInstance = FaceSDKManager.getInstance().getFaceCrop()
            .cropFaceByLandmarkParam(rgbInstance, faceInfos[0].landmarks, cropParam);
    if (cropInstance == null) {
        rgbInstance.destory();
        if (faceDetectCallBack != null) {
            faceDetectCallBack.onTip( code: -1,  msg: "抠图失败");
        }
        return;
    }
    livenessModel.setBdFaceImageInstanceCrop(cropInstance);
}
```

A:请注意相应的参数入参。可参考人脸注册模模块。

Q.SDK 激活鉴权错误码及含义?

A:

SUCCESS	0	成功
LICENSE_NOT_INIT_ERROR	1	license未初始化
LICENSE_DECRYPT_ERROR	2	license数据解密失败
LICENSE_INFO_FORMAT_ERROR	3	license数据格式错误
LICENSE_KEY_CHECK_ERROR	4	license-key(api-key)校验错误
LICENSE_ALGORITHM_CHECK_ERROR	5	算法ID校验错误
LICENSE_MD5_CHECK_ERROR	6	MD5校验错误
LICENSE_DEVICE_ID_CHECK_ERROR	7	设备ID校验错误
LICENSE_PACKAGE_NAME_CHECK_ERROR	8	包名(应用名)校验错误
LICENSE_EXPIRED_TIME_CHECK_ERROR	9	过期时间不正确
LICENSE_FUNCTION_CHECK_ERROR	10	功能未授权
LICENSE_TIME_EXPIRED	11	授权已过期
LICENSE_LOCAL_FILE_ERROR	12	本地文件读取失败
LICENSE_REMOTE_DATA_ERROR	13	远程数据拉取失败
LICENSE_LOCAL_TIME_ERROR	14	本地时间校验错误
OTHER_ERROR	0xff	其他错误

Q.如何正确使用云端 url 特征值同步接口？

A:请求特征值同步接口获取到返回结果后，返回结果的 json 串 feature 字段的值需要 base64 解密成 byte[]在离线 sdk 中使用。

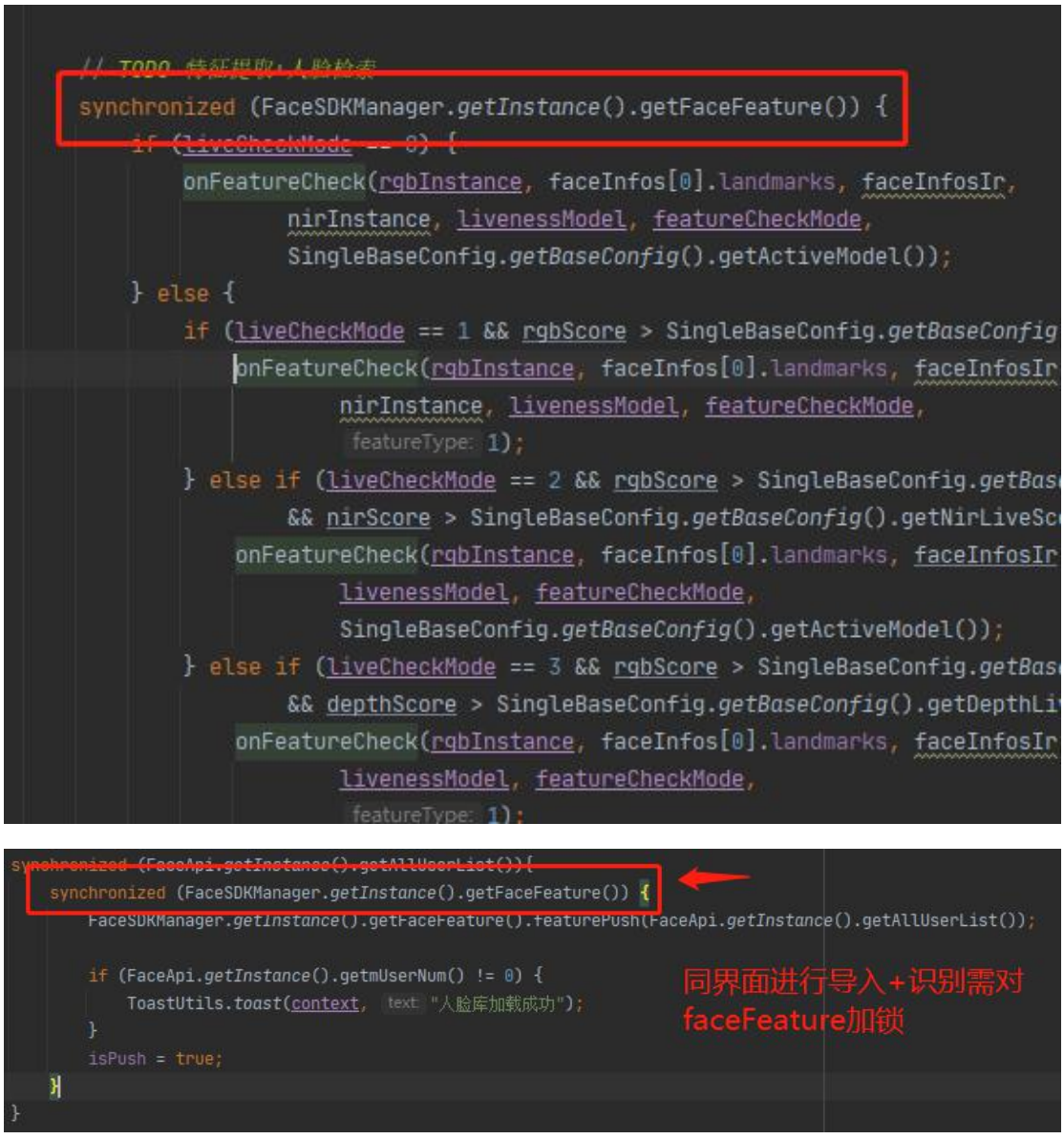
Q. 如何避免 SDK 升级后导致误识和漏识?

A:如果 SDK 升级后，发生误识和漏识的问题，需要刷新本地人脸库，重新把人脸信息导入到本地人脸库。

Q，多线程导致崩溃问题:

A:多线程导入+识别需创建两个检测识别内存，分别进行识别和导入

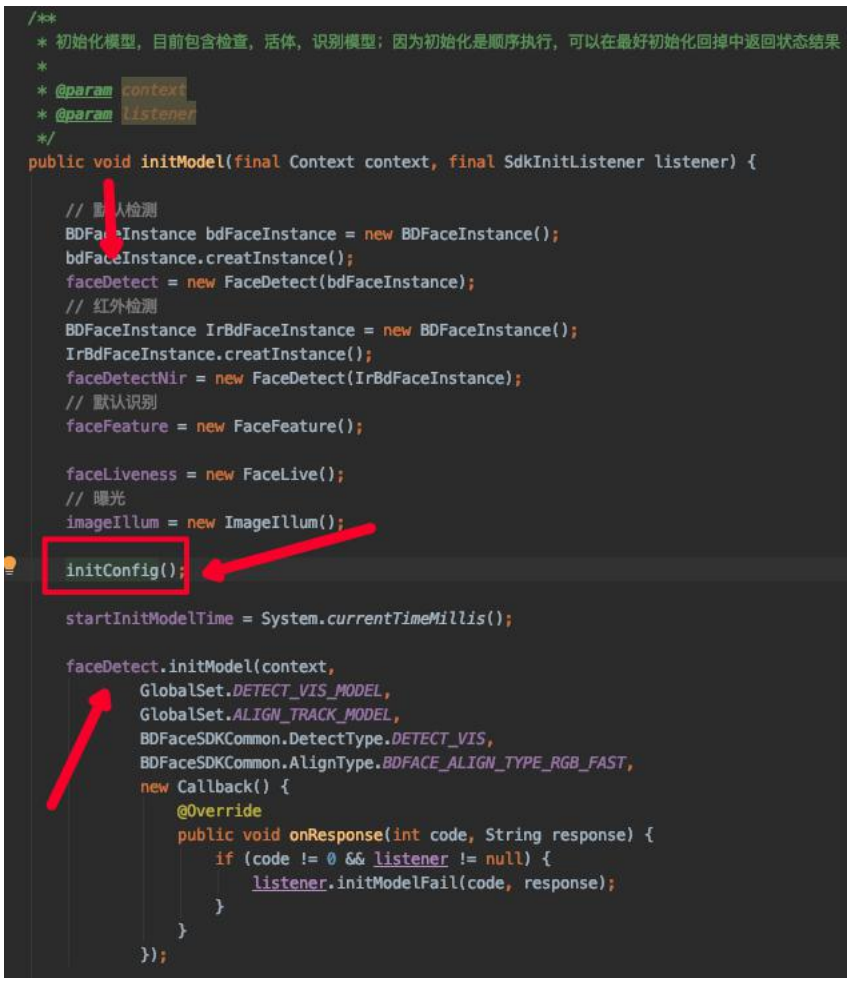
B，识别过程中进行批量导入将数据库数据 push 至内存需与视频流 1：n 识别部分加锁避免内存中数据列表重排序导致误识别

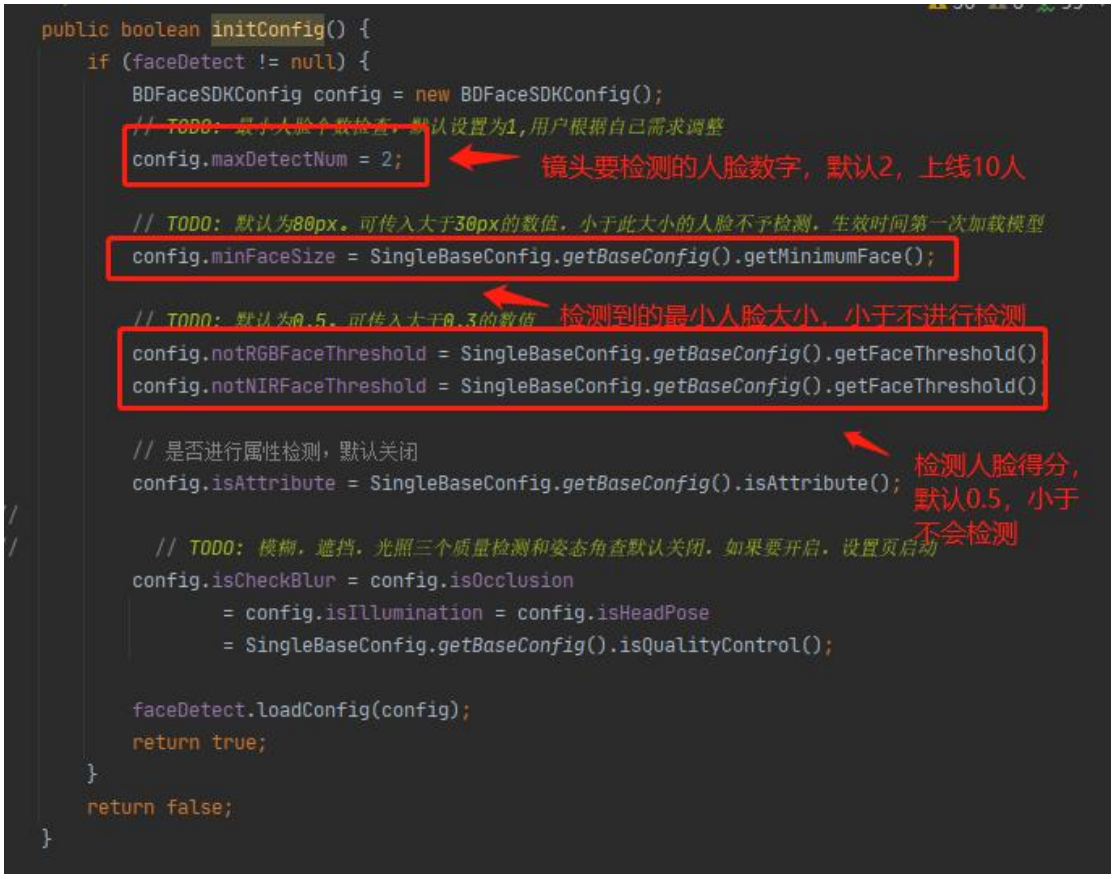


五：识别流程方法和阈值说明

FaceSDKManager.class

每个模块都有相应的 FaceSDKManager.class。


/**
* 初始化模型，目前包含检查，活体，识别模型；因为初始化是顺序执行，可以在最好初始化回掉中返回状态结果
*
* @param context
* @param listener
*/
public void initModel(final Context context, final SdkInitListener listener) {
// 默认检测
BDFaceInstance bdFaceInstance = new BDFaceInstance();
bdFaceInstance.creatInstance();
faceDetect = new FaceDetect(bdFaceInstance);
// 红外检测
BDFaceInstance IrBdFaceInstance = new BDFaceInstance();
IrBdFaceInstance.creatInstance();
faceDetectNir = new FaceDetect(IrBdFaceInstance);
// 默认识别
faceFeature = new FaceFeature();
faceLiveness = new FaceLive();
// 曝光
imageIllum = new ImageIllum();
initConfig();
startInitModelTime = System.currentTimeMillis();
faceDetect.initModel(context,
GlobalSet.DETECT_VIS_MODEL,
GlobalSet.ALIGN_TRACK_MODEL,
BDFaceSDKCommon.DetectType.DETECT_VIS,
BDFaceSDKCommon.AlignType.BDFACE_ALIGN_TYPE_RGB_FAST,
new Callback() {
@Override
public void onResponse(int code, String response) {
if (code != 0 && listener != null) {
listener.initModelFail(code, response);
}
}
});




```
public boolean initConfig() {
    if (faceDetect != null) {
        BDFaceSDKConfig config = new BDFaceSDKConfig();
        // TODO: 最大人脸个数检查，默认设置为1,用户根据自己需求调整
        config.maxDetectNum = 2;

        // TODO: 默认为80px。可传入大于30px的数值，小于此大小的人脸不予检测，生效时间第一次加载模型
        config.minFaceSize = SingleBaseConfig.getBaseConfig().getMinimumFace();

        // TODO: 默认为0.5，可传入大于0.3的数值
        config.notRGBFaceThreshold = SingleBaseConfig.getBaseConfig().getFaceThreshold();
        config.notNIRFaceThreshold = SingleBaseConfig.getBaseConfig().getFaceThreshold();

        // 是否进行属性检测，默认关闭
        config.isAttribute = SingleBaseConfig.getBaseConfig().isAttribute();
//
//          // TODO: 模糊，遮挡，光照三个质量检测和姿态角查默认关闭，如果要开启，设置页启动
        config.isCheckBlur = config.isOcclusion
                = config.isIllumination = config.isHeadPose
                = SingleBaseConfig.getBaseConfig().isQualityControl();

        faceDetect.loadConfig(config);
        return true;
    }
    return false;
}
```

第一步：initModel（）初始化模型能力。code == 0 代表改模型能力初始化成功，否则 code != 0 代表失败。

请参照 code 返回码表，查找失败原因

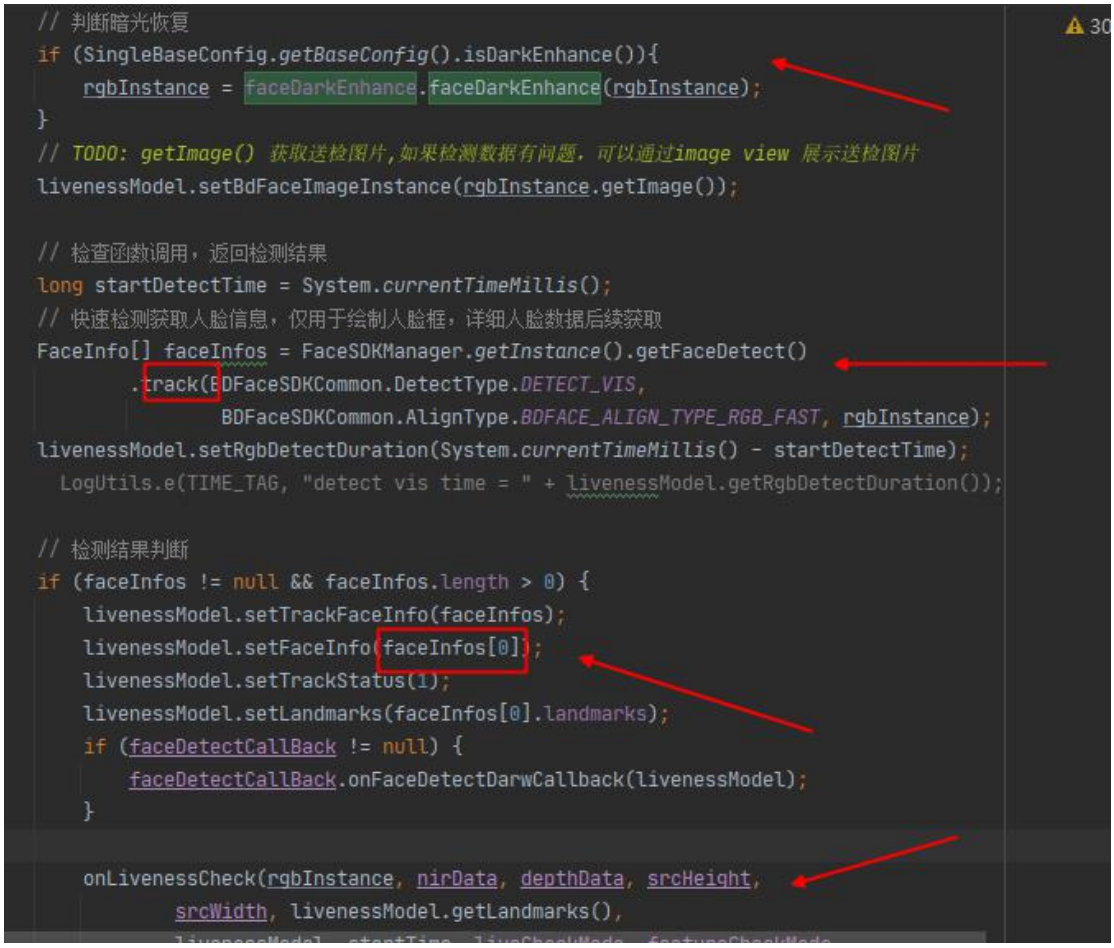
参数名	含义
code	code == 0 成功； code == 1 context 为null； code == -1 非法的参数； code == -2 内存分配失败； code == -3 实例对象为空； code == -4 模型内容为空；code == -5 不支持的能力类型； code == -6 不支持预测类型；code == -7 预测库对象创建失败； code == -8 预测库初始化失败；code == -9 图像数据为空； code == -10 人脸能力初始化失败；code == -11 能力未加载； code == -12 人脸能力已加载；code == -13 未授权； code == -14 人脸能力运行异常；code == -15 不支持的图像类型； code == -16 图像转换失败；

注意 initConfig（）在每个初始化模型能力的前面调用。

第二步：开始检测识别，从预览流中获取到图片帧数据后，调用 onDetectCheck()方法传相应的参数，这里再拿考勤模块举例：

```
private synchronized void checkData() {
    if (rgbData != null && irData != null) {
        FaceSDKManager.getInstance().onDetectCheck(rgbData, irData, depthData: null,
                PERFER_HEIGH, PREFER_WIDTH, liveCheckMode: 2, new FaceDetectCallBack() {
                    @Override
                    public void onFaceDetectCallback(LivenessModel livenessModel) {
                        // 输出结果
                        // 预览模式
                        checkCloseDebugResult(livenessModel);
                        // 开发模式
                        checkOpenDebugResult(livenessModel);

                    }

                    @Override
                    public void onTip(int code, String msg) {
                    }

                    @Override
                    public void onFaceDetectDarwCallback(LivenessModel livenessModel) {
                        // 绘制人脸框
                        showFrame(livenessModel);
                    }
                });
        rgbData = null;
        irData = null;
    }
}
```

```
/**
 * 检测-活体-特征- 全流程
 *
 * @param rgbData           可见光YUV 数据流
 * @param nirData           红外YUV 数据流
 * @param depthData         深度depth 数据流
 * @param srcHeight         可见光YUV 数据流-高度
 * @param srcWidth          可见光YUV 数据流-宽度
 * @param liveCheckMode     活体检测模式【不使用活体：0】；【RGB活体：1】；【RGB+NIR活体：2】；【RGB+Depth活体：3】；【RGB+NIR+Depth活体：4】
 * @param featureCheckMode  特征抽取模式【不提取特征：1】；【提取特征：2】；【提取特征+1：N检索：3】；
 * @param faceDetectCallBack
 */
public void onDetectCheck(final byte[] rgbData,
                          final byte[] nirData,
                          final byte[] depthData,
                          final int srcHeight,
                          final int srcWidth,
                          final int liveCheckMode,
                          final int featureCheckMode,
                          final FaceDetectCallBack faceDetectCallBack) {

    if (future != null && !future.isDone()) {
        return;
    }
```



```
// 判断暗光恢复
if (SingleBaseConfig.getBaseConfig().isDarkEnhance()){
    rgbInstance = faceDarkEnhance.faceDarkEnhance(rgbInstance);
}
// TODO: getImage() 获取送检图片,如果检测数据有问题，可以通过image view 展示送检图片
livenessModel.setBdFaceImageInstance(rgbInstance.getImage());

// 检查函数调用，返回检测结果
long startDetectTime = System.currentTimeMillis();
// 快速检测获取人脸信息，仅用于绘制人脸框，详细人脸数据后续获取
FaceInfo[] faceInfos = FaceSDKManager.getInstance().getFaceDetect()
        .track(BDFaceSDKCommon.DetectType.DETECT_VIS,
                BDFaceSDKCommon.AlignType.BDFACE_ALIGN_TYPE_RGB_FAST, rgbInstance);
livenessModel.setRgbDetectDuration(System.currentTimeMillis() - startDetectTime);
  LogUtils.e(TIME_TAG, "detect vis time = " + livenessModel.getRgbDetectDuration());

// 检测结果判断
if (faceInfos != null && faceInfos.length > 0) {
    livenessModel.setTrackFaceInfo(faceInfos);
    livenessModel.setFaceInfo(faceInfos[0]);
    livenessModel.setTrackStatus(1);
    livenessModel.setLandmarks(faceInfos[0].landmarks);
    if (faceDetectCallBack != null) {
        faceDetectCallBack.onFaceDetectDarwCallback(livenessModel);
    }

    onLivenessCheck(rgbInstance, nirData, depthData, srcHeight,
            srcWidth, livenessModel.getLandmarks(),
```

首先会对图片进行暗光恢复，之后调用 track(),注意 track()获取到的人脸信息 FaceInfo[]数组，仅用于绘制人脸框，faceInfo[0]代表最优最近的一个人脸（多人脸由 initConfig（）方法中的 config.maxDetectNum 设置人数，如果 config.maxDetectNum = 2 ，faceinfo[0]和 faceinfo[1]分别会拿到两个人的信息）

第三步：

```java
/**
 * 活体-特征-人脸检索全流程
 *
 * @param rgbInstance        可见光底层送检对象
 * @param nirData            红外YUV 数据流
 * @param depthData          深度depth 数据流
 * @param srcHeight          可见光YUV 数据流-高度
 * @param srcWidth           可见光YUV 数据流-宽度
 * @param landmark           检测眼睛，嘴巴，鼻子，72个关键点
 * @param livenessModel      检测结果数据集合
 * @param startTime          开始检测时间
 * @param liveCheckMode      活体检测模式【不使用活体：0】；【RGB活体：1】；【RGB+NIR活体：2】；【RGB+Depth活体：3】；【RGB+NIR+Depth活体：4】
 * @param featureCheckMode   特征抽取模式【不提取特征：1】；【提取特征：2】；【提取特征+1：N检索：3】；
 * @param faceDetectCallBack
 */
public void onLivenessCheck(final BDFaceImageInstance rgbInstance,
                            final byte[] nirData,
                            final byte[] depthData,
                            final int srcHeight,
                            final int srcWidth,
                            final float[] landmark,
                            final LivenessModel livenessModel,
                            final long startTime,
                            final int liveCheckMode,
                            final int featureCheckMode,
                            final FaceDetectCallBack faceDetectCallBack,
                            final FaceInfo[] fastFaceInfos) {

    if (future2 != null && !future2.isDone()) {
        // 流程结束销毁图片，开始下一帧图片检测，否着内存泄露
        rgbInstance.destory();
        return;
    }
```

```java
    BDFaceDetectListConf bdFaceDetectListConfig = new BDFaceDetectListConf();
    bdFaceDetectListConfig.usingQuality = bdFaceDetectListConfig.usingHeadPose
            = SingleBaseConfig.getBaseConfig().isQualityControl();
    bdFaceDetectListConfig.usingBestImage = SingleBaseConfig.getBaseConfig().isBestImage();
    FaceInfo[] faceInfos = FaceSDKManager.getInstance()
            .getFaceDetect()
            .detect(BDFaceSDKCommon.DetectType.DETECT_VIS,
                    BDFaceSDKCommon.AlignType.BDFACE_ALIGN_TYPE_RGB_ACCURATE,
                    rgbInstance,
                    fastFaceInfos, bdFaceDetectListConfig);

    // 重新赋予详细人脸信息
    if (faceInfos != null && faceInfos.length > 0) {
        livenessModel.setFaceInfo(faceInfos[0]);
        livenessModel.setTrackStatus(2);
        livenessModel.setLandmarks(faceInfos[0].landmarks);
    } else {
        rgbInstance.destory();
        if (faceDetectCallBack != null) {
            faceDetectCallBack.onFaceDetectCallback(livenessModel);
        }
        return;
    }
```

bast image最优人脸初始化

获取人脸详细信息，包括质量检测数据

获取最优人脸，与track同意

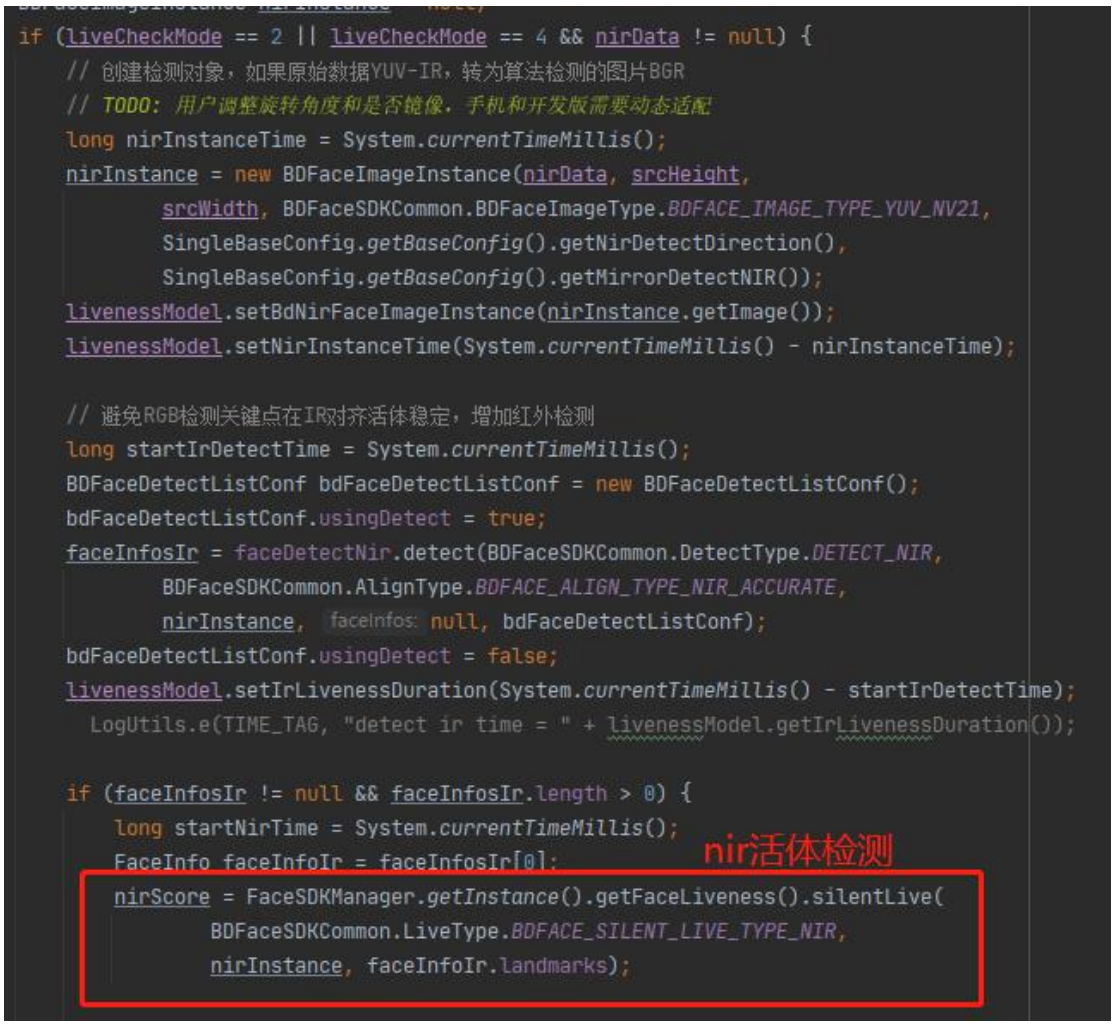
```
}
// 最优人脸控制 如果最优人脸没有通过则销毁BDFaceImageInstance，结束函数
if (!onBestImageCheck(livenessModel, faceDetectCallBack)) {
    rgbInstance.destory();
    if (faceDetectCallBack != null) {
        faceDetectCallBack.onFaceDetectCallback(livenessModel);
    }
    return;
}
// 质量检测未通过,销毁BDFaceImageInstance，结束函数
if (!onQualityCheck(livenessModel, faceDetectCallBack)) {
    rgbInstance.destory();
    if (faceDetectCallBack != null) {
        faceDetectCallBack.onFaceDetectCallback(livenessModel);
    }
    return;
}
```

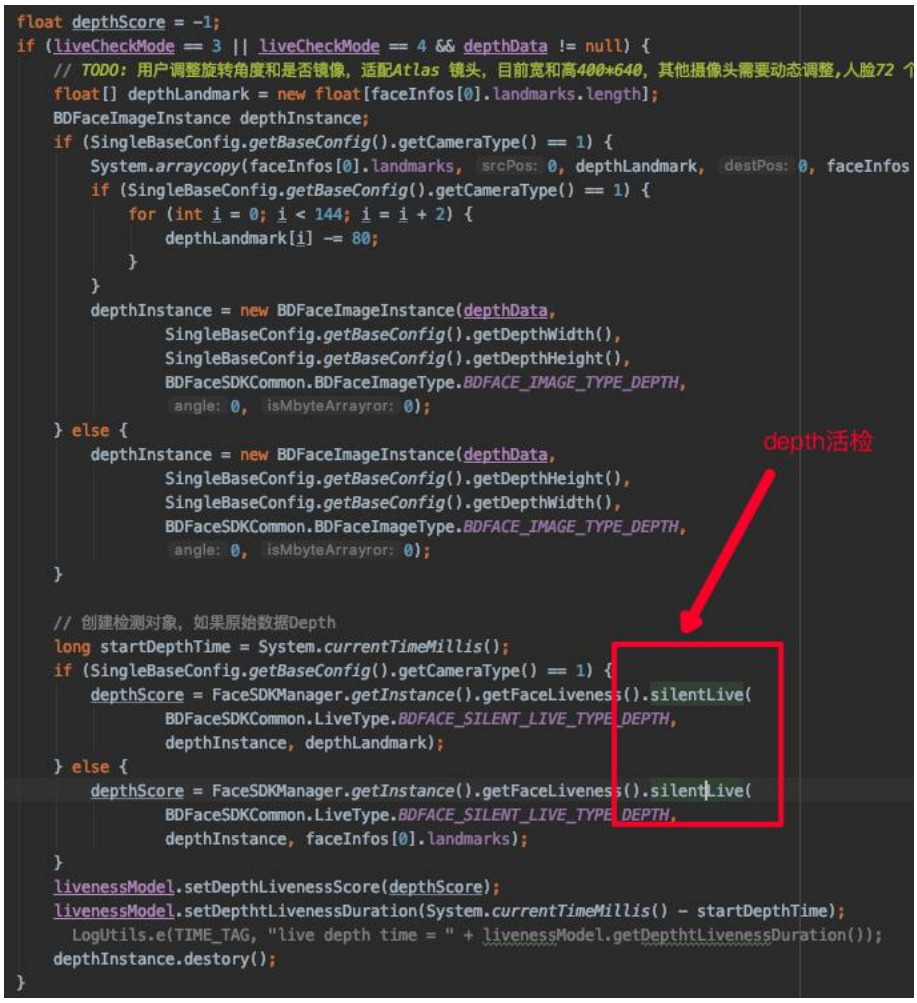
最优人脸过滤

质量检测过滤

```
    }
    // 获取LivenessConfig liveCheckMode 配置选项:【不使用活体: 0】;【RGB活体: 1】;【RGB+NIR活体: 2】;
    // TODO 活体检测
    float rgbScore = -1;
    if (liveCheckMode != 0) {
        long startRgbTime = System.currentTimeMillis();
        rgbScore = FaceSDKManager.getInstance().getFaceLiveness().silentLive(
                BDFaceSDKCommon.LiveType.BDFACE_SILENT_LIVE_TYPE_RGB,
                rgbInstance, faceInfos[0].landmarks);
        mRgbLiveList.add(rgbScore > mRgbLiveScore);
        livenessModel.setRgbLivenessDuration(System.currentTimeMillis() - startRgbTime);
        while (mRgbLiveList.size() > 6) {
            mRgbLiveList.remove( i: 0);
        }
        if (mRgbLiveList.size() > 2) {
```

RGB活体检测


if (liveCheckMode == 2 || liveCheckMode == 4 && nirData != null) {
// 创建检测对象，如果原始数据YUV-IR，转为算法检测的图片BGR
// TODO: 用户调整旋转角度和是否镜像，手机和开发版需要动态适配
long nirInstanceTime = System.currentTimeMillis();
nirInstance = new BDFaceImageInstance(nirData, srcHeight,
srcWidth, BDFaceSDKCommon.BDFaceImageType.BDFACE_IMAGE_TYPE_YUV_NV21,
SingleBaseConfig.getBaseConfig().getNirDetectDirection(),
SingleBaseConfig.getBaseConfig().getMirrorDetectNIR());
livenessModel.setBdNirFaceImageInstance(nirInstance.getImage());
livenessModel.setNirInstanceTime(System.currentTimeMillis() - nirInstanceTime);
// 避免RGB检测关键点在IR对齐活体稳定，增加红外检测
long startIrDetectTime = System.currentTimeMillis();
BDFaceDetectListConf bdFaceDetectListConf = new BDFaceDetectListConf();
bdFaceDetectListConf.usingDetect = true;
faceInfosIr = faceDetectNir.detect(BDFaceSDKCommon.DetectType.DETECT_NIR,
BDFaceSDKCommon.AlignType.BDFACE_ALIGN_TYPE_NIR_ACCURATE,
nirInstance, faceInfos: null, bdFaceDetectListConf);
bdFaceDetectListConf.usingDetect = false;
livenessModel.setIrLivenessDuration(System.currentTimeMillis() - startIrDetectTime);
LogUtils.e(TIME_TAG, "detect ir time = " + livenessModel.getIrLivenessDuration());
if (faceInfosIr != null && faceInfosIr.length > 0) {
long startNirTime = System.currentTimeMillis();
FaceInfo faceInfoIr = faceInfosIr[0];
nirScore = FaceSDKManager.getInstance().getFaceLiveness().silentLive(
BDFaceSDKCommon.LiveType.BDFACE_SILENT_LIVE_TYPE_NIR,
nirInstance, faceInfoIr.landmarks);
nir活体检测



然后调用 detect()获取详细的人脸信息包括质量检测，faceInfo[0]代表最优最近的一个人脸。如果开启了质量检测，并且通过质量检测后，就开始活体检测（包含 3 种类型 1.rgb 活体 2.NIR 活体 3.depth 活体）可选择自己的活体检测类型，调用 silentLive()得到活体检测得分。

```
if (featureType == 3) {
    // todo: 混合模态使用方式是根据图片的曝光来选择需要使用的type，光照的取值范围为：0~255之间
    AtomicInteger atomicInteger = new AtomicInteger();
    FaceSDKManager.getInstance().getImageIllum().imageIllum(rgbInstance, atomicInteger);
    int illumScore = atomicInteger.get();
    if (illumScore > SingleBaseConfig.getBaseConfig().getIllumination()) {
        if (faceInfos != null && nirInstance != null) {
            long startFeatureTime = System.currentTimeMillis();
            float featureSize = FaceSDKManager.getInstance().getFaceFeature().feature(
                    BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_NIR, nirInstance,
                    faceInfos[0].landmarks, feature);
            livenessModel.setFeatureDuration(System.currentTimeMillis() - startFeatureTime);
            livenessModel.setFeature(feature);
            // 人脸检索
            featureSearch(featureCheckMode, livenessModel, feature, featureSize,
                    BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_NIR);
        }

    } else {
        long startFeatureTime = System.currentTimeMillis();
        float featureSize = FaceSDKManager.getInstance().getFaceFeature().feature(
                BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_LIVE_PHOTO, rgbInstance, landmark, fe
        livenessModel.setFeatureDuration(System.currentTimeMillis() - startFeatureTime);
        livenessModel.setFeature(feature);
        // 人脸检索
        featureSearch(featureCheckMode, livenessModel, feature, featureSize,
                BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_LIVE_PHOTO);
    }
```

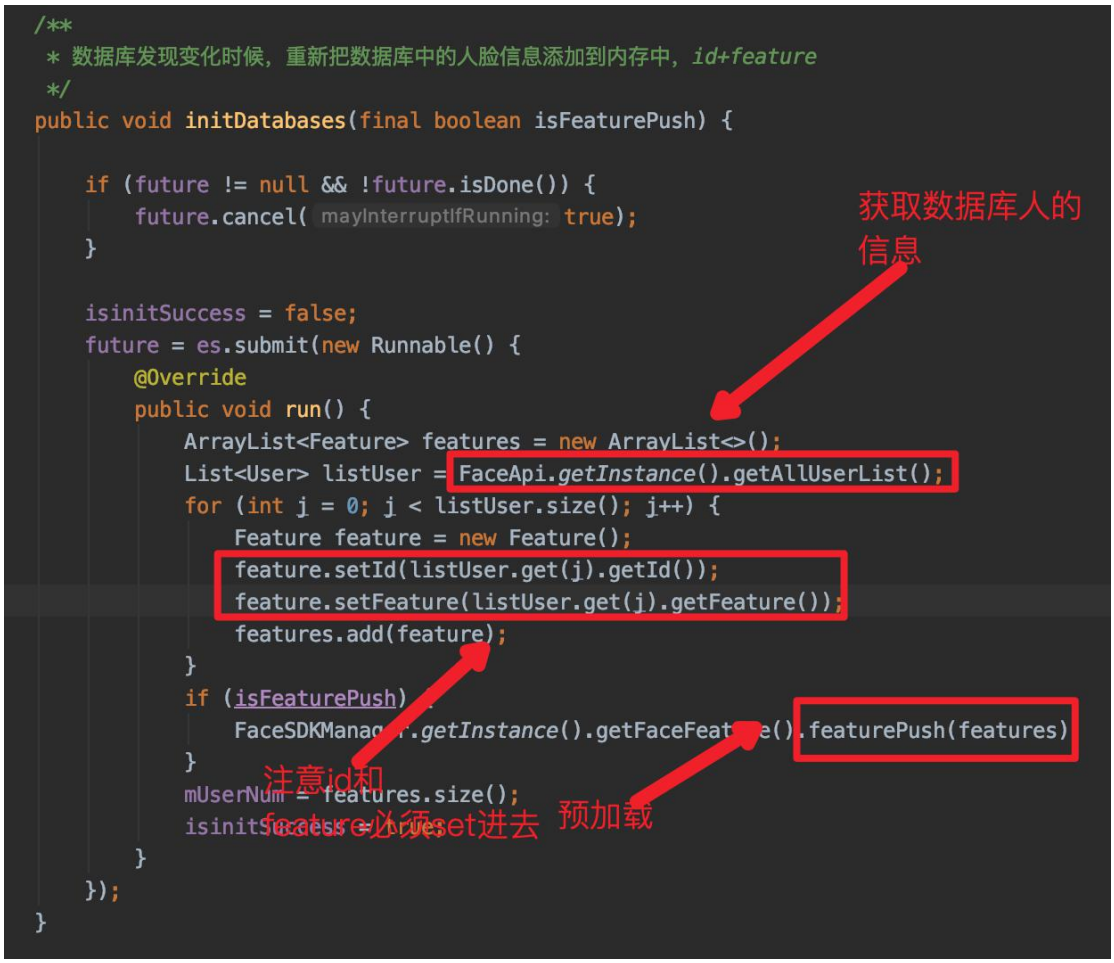
特征提取

```
private void featureSearch(final int featureCheckMode,
                           LivenessModel livenessModel,
                           byte[] feature,
                           float featureSize,
                           BDFaceSDKCommon.FeatureType type) {

    // 如果只提去特征，不做检索，此处返回
    if (featureCheckMode == 2) {
        livenessModel.setFeatureCode(featureSize);
        return;
    }
    // 如果提取特征+检索，调用search 方法
    if (featureSize == FEATURE_SIZE / 4) {

        // 特征提取成功
        // TODO 阈值可以根据不同模型调整
        long startFeature = System.currentTimeMillis();
        ArrayList<Feature> featureResult =
                FaceSDKManager.getInstance().getFaceFeature().featureSearch(feature,
                        type, topNum: 1, isPercent: true);

        // TODO 返回top num = 1 个数据集合，此处可以任意设置，会返回比对从大到小排序的num 个数据集合
        if (featureResult != null && featureResult.size() > 0) {

            // 获取第一个数据
            Feature topFeature = featureResult.get(0);
            // 判断第一个阈值是否大于设定阈值，如果大于，检索成功
            if (SingleBaseConfig.getBaseConfig().getActiveModel() == 1) {
                threholdScore = SingleBaseConfig.getBaseConfig().getLiveThreshold();
            } else if (SingleBaseConfig.getBaseConfig().getActiveModel() == 2) {
                threholdScore = SingleBaseConfig.getBaseConfig().getIdThreshold();
            } else if (SingleBaseConfig.getBaseConfig().getActiveModel() == 3) {
                threholdScore = SingleBaseConfig.getBaseConfig().getRgbAndNirThreshold();
            }
            if (topFeature != null && topFeature.getScore() >
                    threholdScore) {
                // 当前featureEntity 只有id+feature 索引，在数据库中查到完整信息
                User user = FaceApi.getInstance().getUserListById(topFeature.getId());
                if (user != null) {
                    livenessModel.setUser(user);
                    livenessModel.setFeatureScore(topFeature.getScore());
                }
            }
        }
        livenessModel.setCheckDuration(System.currentTimeMillis() - startFeature);
    }
```

开始1:N

得到置信度最相似的人

大于置信度阈值认为是同一个人



第四步：通过 track，detect 和 silentLive 之后，开始特征提取。特征提取用于 1：1 和 1:N。特征提取通过后，这里以考勤模块 1:N 举例。（注意在调用 1:N 方法前，一定要先预加载人脸库所有人的特征值作为特征底库，1:1 不需要预加载）

BaseConfig.class.

阈值指质量检测，活检检测，帧数设置，人脸大小设置等。可参考各个模块的 BaseConfig.class。您可以根据自己的使用场景调整这些阈值，用来控制严格或者宽松。

例如：质量检测

严格：遮挡阈值越大越严格。

宽松：遮挡阈值越小越宽松越容易通过质量检测。

```
// 模糊度设置，默认0.5。取值范围[0~1]，0是最清晰，1是最模糊
private float blur = 0.8f;
// 光照设置，默认0.8,取值范围[0~1]，数值越大，光线越强
private float illum = 0.8f;
// 姿态阈值
private float gesture = 15;
// 三维旋转之俯仰角度[-90(上)，90(下)]，默认20
private float pitch = 20;
// 平面内旋转角[-180(逆时针)，180(顺时针)]，默认20
private float roll = 20;
// 三维旋转之左右旋转角[-90(左)，90(右)]，默认20
private float yaw = 20;
// 遮挡阈值
private float occlusion = 0.8f;
// 左眼被遮挡的阈值，默认0.6
private float leftEye = 0.8f;
// 右眼被遮挡的阈值，默认0.6
private float rightEye = 0.8f;
// 鼻子被遮挡的阈值，默认0.7
private float nose = 0.8f;
// 嘴巴被遮挡的阈值，默认0.7
private float mouth = 0.8f;
// 左脸颊被遮挡的阈值，默认0.8
private float leftCheek = 0.8f;
// 右脸颊被遮挡的阈值，默认0.8
private float rightCheek = 0.8f;
// 下巴被遮挡阈值，默认为0.6
private float chinContour = 0.8f;
```

阈值推荐：

姿态：15

光照：0.8

模糊：0.8

左右眼：0.8

左右脸颊：0.8

鼻子：0.8

嘴巴：0.8

下巴：0.8