目录

一、开发环境说明

开发环境：Android Studio
Android SDK 版本：
minSdkVersion 21
compileSdkVersion 23
targetSdkVersion 23

二、API 说明

1、摄像头服务开启接口

功能：开启摄像头功能

```
/*
*@ context: 描述程序的上下文
*/
static boolean startCameraService(Context context);
```

2、摄像头服务关闭接口

功能：关闭摄像头功能

```
/*
*@ context: 描述程序的上下文
*/
static boolean stopCameraService(Context context);
```

3、人脸处理监听器设置接口

功能：设置人脸处理回调接口的对象

```
/*
*@ listener: 设置接收回调接口数据的对象
*/
setOnFaceProcessAvailableListener(OnFaceProcessAvailableListener listener);
```

4、人脸注册任务开启接口

功能：启动人脸注册任务

boolean mm_start_face_register();

5、人脸注册回调接口

功能：接收人脸注册结果

Status 取值定义:

```
interface Face_Status {
        int FACE_NORMAL = 0;    /*表示注册成功*/
        int FACE_NO_FACE = 1;   /*表示没检测到人脸*/
    }
```

```
/*
*@ status: 描述注册信息，比如是否注册成功，或者是否检测到人脸,用来提示用户使人脸在 UI 圆圈中合适的位置;
*@ img:     需要显示在注册的圆圈中的图片
*@ age:     检测出来人的年龄
*@ gender:  检测出来人的性别
*@ id:      成功注册后获得的唯一 id
*/
void Face_Register_CB_Func(int status, Bitmap img, int age, int gender, int id);
```

6、人脸识别任务开启接口

功能：启动人脸识别任务

```
boolean mm_start_face_recognition();
```

7、人脸识别回调接口

功能：接收人脸识别结果

```
/*
*@ img: 识别出来时人的图片
*@ age: 识别出来人的年龄
*@ gender: 识别出来人的性别
*@ id: 识别出注册时的唯一 id,如果为 0xffffffff 表示没有注册
*/
void Face_Recognition_CB_Func(Bitmap img, int age, int gender, int id);
```

8、人脸处理任务关闭接口

功能：关闭人脸处理任务,清空监听器

```
boolean mm_stop_face_process();
```

9、人脸删除接口

功能：用来删除某个注册的人脸

```
/*
*@ id: 要删除的人脸 id
*/
boolean mm_faceid_delete(int id);
```

10、手势处理监听器设置接口

功能：设置手势处理回调接口的对象

```
/*
*@ listener: 设置接收回调接口数据的对象
*/
setOnGestureProcessAvailableListener(OnGestureProcessAvailableListener
listener);
```

11、手势识别任务开启接口

功能：启动手势识别任务

```
boolean mm_start_gesture_recognition();
```

12、手势识别回调接口

功能：接收手势识别结果

手势 id 取值定义:

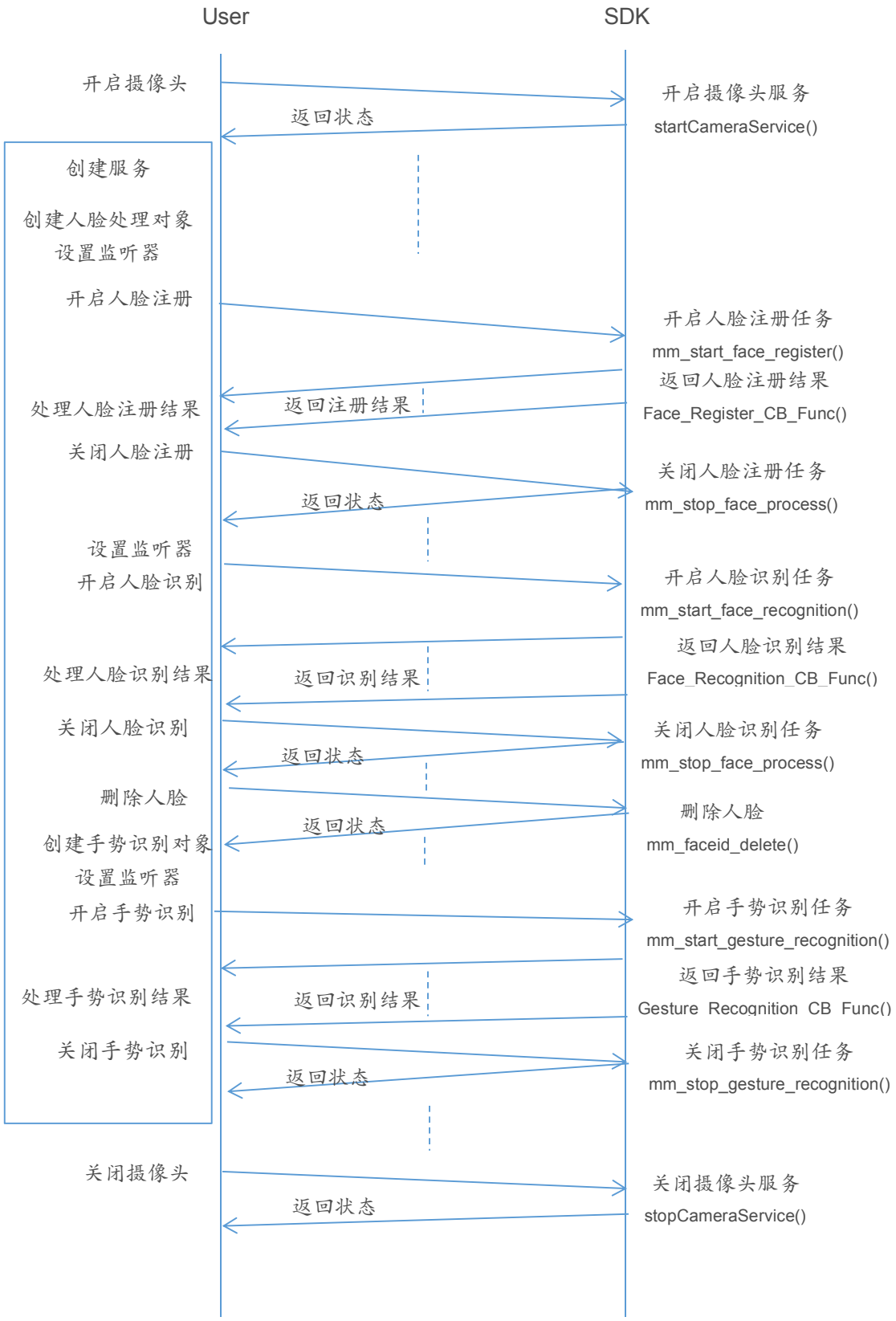
```
interface Gesture_Pose{
    GESTURE_POSE_0=0, /*表示手势 0*/
    GESTURE_POSE_1=1, /*表示手势 1*/
    GESTURE_POSE_2=2, /*表示手势 2*/
    GESTURE_POSE_3=3, /*表示手势 3*/
    GESTURE_POSE_4=4, /*表示手势 4*/
    GESTURE_POSE_MAC=0xff, /*表示没识别*/
}
/*
*@ id: 识别出来手的 pose
*/
void Gesture_Recognition_CB_Func(int id);
```

13、手势识别关闭接口

功能：关闭手势识别

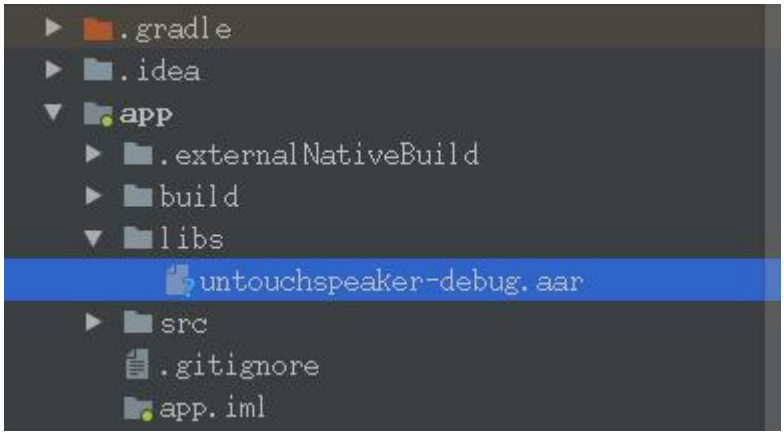
boolean mm_stop_gesture_recognition();

三、Use Case 处理流程



四、导入 AAR 包

1、将 AAR 包复制到 app 的 libs 目录下，如下图：



2、配置 build.gradle 文件：

在 app 下的 build.gradle 中加入

```
repositories{
    flatDir{
        dirs ‘libs’
    }
}
```

在 dependencies 中加入

compile(name:’untouchspeaker-debug’,ext:’aar’)

Name 指定的是要导入的 aar 包名称；ext 指定的是要导入的包后缀

3、同步 gradle 即可。

4、如果调试需要更换 AAR 包，需要先把 libs 目录下的 AAR 包删除，然后屏蔽

```
repositories{
    flatDir{
        dirs ‘libs’
    }
}
```

和 compile(name:’untouchspeaker-debug’,ext:’aar’)，点击同步 gradle，然后再从步骤 1 开始导入新的 AAR 包即可。

五、创建人脸手势处理服务

1、创建 untouchService 类，继承自父类 Service，重载 onCreate，onStartCommand，onDestroy 和 onBind 方法；

2、在 Android Manifest.xml 文件中的<application>标签中声明服务：

```
<service android:name=".untouchService"
    android:enabled="true">
</service>
```

3、导入人脸处理包和手势处理包：

```
import com.example.untouch.untouchspeaker.FaceProcess;
import com.example.untouch.untouchspeaker.GestureProcess;
import com.example.untouch.untouchspeaker.OnFaceProcessAvailableListener;
import com.example.untouch.untouchspeaker.OnGestureProcessAvailableListener;
```

4、继承回调接口的监听器接口：

```
implements OnFaceProcessAvailableListener, OnGestureProcessAvailableListener
```

5、定义人脸处理对象和手势处理对象：

```
private FaceProcess faceProcess;
private GestureProcess gestureProcess;
```

6、在 onCreate 方法中设置监听器：

```
faceProcess = new FaceProcess();
faceProcess.setOnFaceProcessAvailableListener(this);
gestureProcess = new GestureProcess();
gestureProcess.setOnGestureProcessAvailableListener(this);
```

7、重载回调接口：

1）人脸注册回调接口：

```
@Override
public void Face_Register_CB_Func(int status, Bitmap bitmap, int age, int gender, int id){}
```

2）人脸识别回调接口：

```
@Override
public void Face_Recognition_CB_Func(Bitmap bitmap, int age, int gender, int id){}
```

3）手势识别回调接口：

```
@Override
public void Gesture_Recognition_CB_Func(int id){}
```

8、调用相应的方法：

1）在 onStartCommand 方法中调用人脸注册启动方法：

```
faceProcess.mm_start_face_register();
```

2）在人脸注册回调接口中判断注册成功，调用人脸处理关闭方法关闭人脸注册任务：

```
faceProcess.mm_stop_face_process();
```

设置人脸处理监听器：

```
faceProcess.setOnFaceProcessAvailableListener(this);
```

调用人脸识别启动方法：

```
faceProcess.mm_start_face_recognition();
```

3）在人脸识别函数中判断 id 号为 5 时，调用人脸处理关闭方法：

```
faceProcess.mm_stop_face_process();
```

调用人脸删除方法：

```
faceProcess.mm_faceid_delete(id);
```

调用手势识别启动方法：

```
gestureProcess.mm_start_gesture_recognition();
```

9、在服务停止方法 onDestroy 中调用人脸处理和手势识别关闭方法：

```
faceProcess.mm_stop_face_process();
gestureProcess.mm_stop_gesture_recognition();
```

10、人脸手势处理服务完成方法说明：

```
import com.example.untouch.untouchspeaker.FaceProcess;
import com.example.untouch.untouchspeaker.GestureProcess;
import com.example.untouch.untouchspeaker.OnFaceProcessAvailableListener;
import com.example.untouch.untouchspeaker.OnGestureProcessAvailableListener;

public class untouchService extends Service implements OnFaceProcessAvailableListener, OnGestureProcessAvailableListener {

    private final String TAG="untouchService";
    private FaceProcess faceProcess;
    private GestureProcess gestureProcess;

    @Override
    public void onCreate()
    {
        Log.d(TAG, msg: "onCreate");
        super.onCreate();
        faceProcess = new FaceProcess();
        faceProcess.setOnFaceProcessAvailableListener(this);
        gestureProcess = new GestureProcess();
        gestureProcess.setOnGestureProcessAvailableListener(this);
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
    {
        Log.d(TAG, msg: "onStart and id is: "+startId);
        super.onStartCommand(intent, flags, startId);
        faceProcess.mm_start_face_register();
        return START_NOT_STICKY;
    }

    @Override
    public void onDestroy()
    {
        Log.d(TAG, msg: "onDestroy");
        super.onDestroy();
        faceProcess.mm_stop_face_process();
        gestureProcess.mm_stop_gesture_recognition();
    }
```

```
    @Override
    public void Face_Register_CB_Func(int status, Bitmap bitmap, int age, int gender, int id)
    {
        Log.d(TAG, msg: "the status is: "+status+" the age is: "+age+" the gender is: "+gender+" the id is: "+id);
        if(status == FaceProcess.Face_Status.FACE_NORMAL)
        {
            faceProcess.mm_stop_face_process();
            faceProcess.setOnFaceProcessAvailableListener(this);
            faceProcess.mm_start_face_recognition();
        }
    }

    @Override
    public void Face_Recognition_CB_Func(Bitmap bitmap, int age, int gender, int id)
    {
        Log.d(TAG, msg: "the age is: "+age+" the gender is: "+gender+" the id is: "+id);
        if(id == 5)
        {
            faceProcess.mm_stop_face_process();
            faceProcess.mm_faceid_delete(id);
            gestureProcess.mm_start_gesture_recognition();
        }
    }

    @Override
    public void Gesture_Recognition_CB_Func(int id)
    {
        Log.d(TAG, msg: "the gesture is: "+id);
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

六、检测硬件使用权限及请求相应权限

1、定义变量：

```
private static final int PERMISSIONS_REQUEST = 1;
private static final String PERMISSION_CAMERA = Manifest.permission.CAMERA;
private static final String PERMISSION_STORAGE = Manifest.permission.WRITE_EXTERNAL_STORAGE;
```

2、在 MainActivity 的 onCreate 方法中检测相应权限，如果没有则进行请求：

```
if (!hasPermission()) {
    requestPermission();
}
```

3、检测权限和请求权限方法：

```
private boolean hasPermission() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        //TYPE_SYSTEM_OVERLAY权限
        if(!Settings.canDrawOverlays(getApplicationContext())) {
            //启动Activity让用户授权
            Intent intent = new Intent(Settings.ACTION_MANAGE_OVERLAY_PERMISSION);
            intent.setData(Uri.parse("package:" + getPackageName()));
            startActivityForResult(intent, requestCode: 100);
        }
        return checkSelfPermission(PERMISSION_CAMERA) == PackageManager.PERMISSION_GRANTED &&
                checkSelfPermission(PERMISSION_STORAGE) == PackageManager.PERMISSION_GRANTED;
    } else {
        return true;
    }
}

private void requestPermission() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        if (shouldShowRequestPermissionRationale(PERMISSION_CAMERA) ||
                shouldShowRequestPermissionRationale(PERMISSION_STORAGE)) {
            Toast.makeText( context: MainActivity.this,
                    text: "Camera AND storage permission are required for this demo", Toast.LENGTH_LONG).show();
        }
        requestPermissions(new String[] {PERMISSION_CAMERA, PERMISSION_STORAGE}, PERMISSIONS_REQUEST);
        Log.d( tag: "waitRequestPermission", msg: "come here");
    }
}
```

七、启动服务

1、导入摄像头服务包：

```
import com.example.untouch.untouchspeaker.CameraProcess;
```

2、定义四个按钮，分别对应启动摄像头服务，关闭摄像头服务，启动人脸手势处理服务，关闭人脸手势处理服务：

```
private Button startCamera;
private Button closeCamera;
private Button startUntouchService;
private Button closeUntouchService;
```

3、继承按钮监听器接口：

```
implements View.OnClickListener
```

4、定义的四个按钮对象和界面上定义的四个按钮关联并设置监听器：

```
startCamera = (Button) findViewById(R.id.startCamera);
startCamera.setOnClickListener(this);

closeCamera = (Button) findViewById(R.id.closeCamera);
closeCamera.setOnClickListener(this);

startUntouchService = (Button) findViewById(R.id.startUntouchService);
startUntouchService.setOnClickListener(this);

closeUntouchService = (Button) findViewById(R.id.closeUntouchService);
closeUntouchService.setOnClickListener(this);
```

5、在按钮回调接口 onClick 中实现启动和关闭相应服务：

1）摄像头启动服务接口：

```
CameraProcess.startCameraService( context: this);
```

2）摄像头关闭服务接口：

```
CameraProcess.stopCameraService( context: this);
```

3）人脸手势处理服务启动接口：

```
Intent intent = new Intent( packageContext: MainActivity.this, untouchService.class);
startService(intent);
```

4）人脸手势处理服务关闭接口：

```
Intent intent = new Intent( packageContext: MainActivity.this, untouchService.class);
stopService(intent);
```

6、按钮相应回调接口处理：

```
@Override
public void onClick(View v) {
    switch (v.getId())
    {
        case R.id.startCamera:
        {
            CameraProcess.startCameraService( context: this);
            break;
        }
        case R.id.closeCamera:
        {
            CameraProcess.stopCameraService( context: this);
            break;
        }
        case R.id.startUntouchService:
        {
            Intent intent = new Intent( packageContext: MainActivity.this, untouchService.class);
            startService(intent);
            break;
        }
        case R.id.closeUntouchService:
        {
            Intent intent = new Intent( packageContext: MainActivity.this, untouchService.class);
            stopService(intent);
            break;
        }
    }
}
```

7、完整方法说明：

```java
import com.example.untouch.untouchspeaker.CameraProcess;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private Button startCamera;
    private Button closeCamera;
    private Button startUntouchService;
    private Button closeUntouchService;
    private static final int PERMISSIONS_REQUEST = 1;
    private static final String PERMISSION_CAMERA = Manifest.permission.CAMERA;
    private static final String PERMISSION_STORAGE = Manifest.permission.WRITE_EXTERNAL_STORAGE;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        startCamera = (Button) findViewById(R.id.startCamera);
        startCamera.setOnClickListener(this);

        closeCamera = (Button) findViewById(R.id.closeCamera);
        closeCamera.setOnClickListener(this);

        startUntouchService = (Button) findViewById(R.id.startUntouchService);
        startUntouchService.setOnClickListener(this);

        closeUntouchService = (Button) findViewById(R.id.closeUntouchService);
        closeUntouchService.setOnClickListener(this);

        if (!hasPermission()) {
            requestPermission();
        }

    }
```

```java
    @Override
    public void onClick(View v) {
        switch (v.getId())        {
            case R.id.startCamera:
                CameraProcess.startCameraService( context: this);
                break;
            case R.id.closeCamera:
                CameraProcess.stopCameraService( context: this);
                break;
            case R.id.startUntouchService:
                Intent intent = new Intent( packageContext: MainActivity.this, untouchService.class);
                startService(intent);
                break;
            case R.id.closeUntouchService:
                Intent intent = new Intent( packageContext: MainActivity.this, untouchService.class);
                stopService(intent);
                break;
        }
    }

    private boolean hasPermission() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            //TYPE_SYSTEM_OVERLAY权限
            if(!Settings.canDrawOverlays(getApplicationContext())) {
                //启动Activity让用户授权
                Intent intent = new Intent(Settings.ACTION_MANAGE_OVERLAY_PERMISSION);
                intent.setData(Uri.parse("package:" + getPackageName()));
                startActivityForResult(intent, requestCode: 100);
            }
            return checkSelfPermission(PERMISSION_CAMERA) == PackageManager.PERMISSION_GRANTED &&
                    checkSelfPermission(PERMISSION_STORAGE) == PackageManager.PERMISSION_GRANTED;
        } else {
            return true;
        }
    }

    private void requestPermission() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            if (shouldShowRequestPermissionRationale(PERMISSION_CAMERA) ||
                    shouldShowRequestPermissionRationale(PERMISSION_STORAGE)) {
                Toast.makeText( context: MainActivity.this,
                        text: "Camera AND storage permission are required for this demo", Toast.LENGTH_LONG).show();
            }
            requestPermissions(new String[] {PERMISSION_CAMERA, PERMISSION_STORAGE}, PERMISSIONS_REQUEST);
        }
    }
}
```

八、接口说明

1、先调用摄像头启动服务，在调用人脸处理和手势识别服务；

2、先停止人脸处理和手势识别服务，再调用摄像头停止服务；

3、调用人脸注册方法后，会发送 5 次人脸注册失败，然后发送人脸注册成功，图片 bitmap 一直为空，age、gender、id 为从 0 开始累加的数；

4、调用人脸识别方法后，发送人脸识别结果，图片 bitmap 为空，age、gender、id 为从 0 开始累加的数；

5、调用手势识别方法后，会连续发送手势识别结果，id 为从 0 开始累加的数。

6、调用人脸处理关闭方法后，会同时关闭人脸注册和人脸识别功能，并清空监听器，所以如果只是关闭一个功能，再开启下一个功能之前需要重新注册监听器。