ORIONSTAR

豹小秘SDK开发文档

北京猎户星空科技有限公司

版本：V4.19

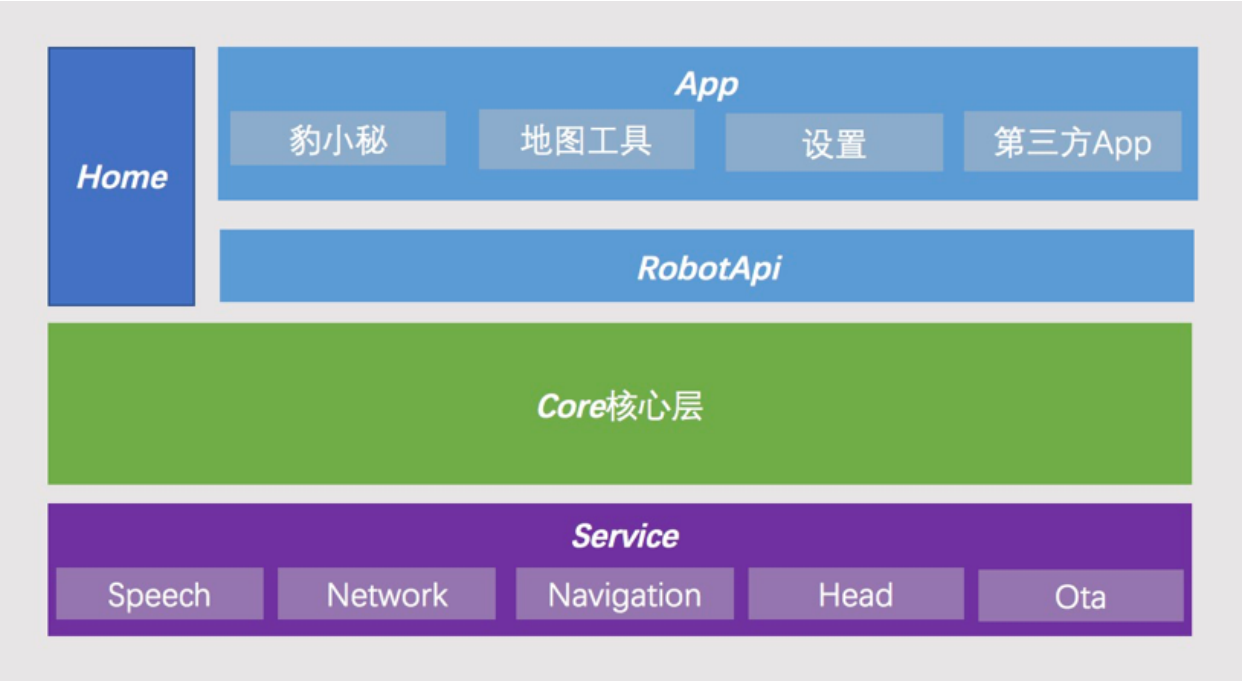
修订日期：2019-10-18

修订记录

版本	日期	描述
V1.0	2018-12-20	创建
V1.1	2019-01-23	新增语音最终识别结果回调 新增导航速度控制
V1.2	2019-03-04	废弃setExceptionIsClosed接口 修复未连接情况下调用接口崩溃问题 优化接口跨进程调用
V1.3	2019-03-29	新增开始自动回充接口 新增停止自动回充接口 新增脱离充电桩接口
V4.8	2019-04-24	新增系统服务 1. 新增转向目标点方向接口 2. 新增设置灯效接口 3. 新增txt转mp3接口
V4.10	2019-06-02	1. 语音接口，通过文本获取NLP结果 2. 系统接口，获取本机sn 3. 废弃关闭或开启语音接口setASREnabled
V4.12	2019-06-27	新增权限说明 startNavigation接口支持导航到指定坐标点 新增音频数据采集
V4.19	2019-10-18	新增获取系统版本接口 修复人脸数据不全问题 新增语音onQueryEnded回调参数说明

目录

RobotOS架构图



Home为默认launcher，可通过三指下拉，点击应用中心进入，其用来管理所有App对RobotApi的访问控制权，App必须在Home中点击启动，获得访问授权后，才能对Api进行访问

配置默认启动程序

配置Manifest文件

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
     </intent-filter>
     <intent-filter>
         <action android:name="action.orionstar.default.app" />
         <category android:name="android.intent.category.DEFAULT" />
     </intent-filter>
</activity>
```

App如果需要在开机后默认启动进入，需要在Manifest中配置

```
<intent-filter>
    <action android:name="action.orionstar.default.app" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

并在设置（三指下拉点击设置进入）- 开发者设置 – 开机启动程序中设置

注意：开机启动程序设置功能在OTA3才正式上线

权限

SDK的完整接入，需要在AndroidManifest.xml中声明以下权限

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/
>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

SDK接入

1. 创建接收语音请求及系统事件的回调

```
public class ModuleCallback extends ModuleCallbackApi {
    @Override
    public boolean onSendRequest(int reqId, String reqType, String reqTex
t, String reqParam)
            throws RemoteException {
        //接收语音指令，
        //reqTyp : 语音指令类型
        //reqText : 语音识别内容
        //reqParam : 语音指令参数
        return true;
    }

    @Override
    public void onRecovery() throws RemoteException {
        //控制权恢复，收到该事件后，重新恢复对机器人的控制
    }

    @Override
    public void onSuspend() throws RemoteException {
        //控制权被系统剥夺，收到该事件后，所有Api调用无效
    }
}
```

2. 连接Server

```
RobotApi.getInstance().connectServer(this, new ApiListener() {
```

```
    @Override
    public void handleApiDisabled() {

    }

    @Override
    public void handleApiConnected() {
        //Server已连接，设置接收请求的回调，包含语音指令、系统事件等
        RobotApi.getInstance().setCallback(new ModuleCallback());

    }

    @Override
    public void handleApiDisconnected() {
        //连接已断开
    }
});
```

注意：所有Api必须在连接到Server后才能调用

3. 设置回调

```
RobotApi.getInstance().setCallback(new ModuleCallback());
```

4. 结束指令

```
RobotApi.getInstance().finishModuleParser(reqId, result);
```

当收到的请求或语音指令处理完成后需调用finishModuleParser结束该指令，reqId 为onSendRequest回调里获得，result为执行结果

5. 注册状态监听

```
StatusListener statusListener = new StatusListener(){
    @Override
    public void onStatusUpdate(String type, String data) throws RemoteExce
ption {

    }
};
//注册状态监听
RobotApi.getInstance().registerStatusListener(type, statusListener);
//解除状态监听
RobotApi.getInstance().unregisterStatusListener(statusListener);
```

type为需要监听的状态类型，目前type支持以下几种：
Definition.STATUS_POSE：机器人当前的坐标，持续上报
Definition.STATUS_POSE_ESTIMAT：当前定位状态，定位状态发生改变时上报

人脸识别

1. 本地人脸识别

方法名称：startGetAllPersonInfo / stopGetAllPersonInfo
调用方式：

```
PersonInfoListener listener = new PersonInfoListener() {
    @Override
    public void onData(int code, List<Person> data) {
        //code : Definition.STATUS_INFO_UPDATE
        //data : 识别的人脸数据
    }
};
//开始
RobotApi.getInstance().startGetAllPersonInfo(reqId, listener);
//结束
RobotApi.getInstance().stopGetAllPersonInfo(reqId, listener);
```

参数说明：

- **reqId**：通用参数，可从ModuleCallback的onSendRequest回调里获得
- **listener**：人脸数据回调

注意：开始和结束的参数需要使用同一个listener *Person人脸信息*

```
int id; //人脸id
int angle; //角度
double distance; //距离
int headSpeed; //当前机器人头部转动速度
long latency; //数据延迟
int facewidth; //人脸宽度
int faceheight; //人脸高度
double faceAngleX; //人脸X轴角度
double faceAngleY; //人脸Y轴角度
double angleInView; //人相对于机器人头部的角度
int faceX; //人脸X坐标
int faceY; //人脸Y坐标
```

2. 获取人脸照片

方法名称：getPictureById
调用方式：

```
RobotApi.getInstance().getPictureById(reqId, faceId, count, new CommandLis
tener() {
    @Override
    public void onResult(int result, String message) {
        try {
            JSONObject json = new JSONObject(message);
            String status = json.optString("status");
            //获取照片成功
            if (Definition.RESPONSE_OK.equals(status)) {
                JSONArray pictures = json.optJSONArray("pictures");
                if (!TextUtils.isEmpty(pictures.optString(0))) {
                    //照片本地存储全路径
                    String picturePath = pictures.optString(0);
                }
            }
        } catch (JSONException | NullPointerException e) {
            e.printStackTrace();
        }
    }
});
```

参数

- **faceId**：人脸id，可通过本地人脸识别获得（Person的id）
- **count**：需要获取的照片数量，此参数目前无效，默认只有一张

注意：该接口保存的图片在使用完后需要手动删除

3. 远程注册

方法名称：startRegister
调用方式：

```
RobotApi.getInstance().startRegister(reqId, personName, timeout, tryCount,
secondDelay, new ActionListener() {
    @Override
    public void onResult(int status, String response) throws RemoteExcepti
on {
        if (Definition.RESULT_OK != status) {
            //注册失败
            return;
        }

        try {
            JSONObject json = new JSONObject(response);
```

```
            String remoteType = json.optString(Definition.REGISTER_REMOTE_
TYPE);
            String remoteName = json.optString(Definition.REGISTER_REMOTE_
NAME);
            if (Definition.REGISTER_REMOTE_SERVER_EXIST.equals(remoteTyp
e)) {
                //当前用户已存在
            } else if (Definition.REGISTER_REMOTE_SERVER_NEW.equals(remote
Type)) {
                //新注册用户成功
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }

    }
});
```

参数

- **personName**：注册名称
- **timeout**：注册超时时间，注册失败时会尝试重新注册，直到超时
- **tryCount**：失败重试次数，重试的时间大于timeout后，会放弃重试
- **secondDelay**：重试间隔时间

4. 远程识别

方法名称：getPersonInfoFromNet
调用方式：

```
RobotApi.getInstance().getPersonInfoFromNet(reqId, faceId, pictures, new C
ommandListener() {

    @Override
    public void onResult(int result, String message) {
        try {
            JSONObject json = new JSONObject(message);
            JSONObject info = json.getJSONObject("people");
            info.getString("name"); //名称
            info.getString("gender"); //性别
            info.getString("age"); //年龄
        } catch (JSONException | NullPointerException e) {
            e.printStackTrace();
        }
    }
});
```

参数

- **faceId**：人脸id，可通过本地人脸识别获得
- **pictures**：人脸照片，可通过getPictureById接口获得

基础控制

1. 直线运动

方法名称：goForward / goBackward
调用方式：

```
CommandListener motionListener = new CommandListener() {
    @Override
    public void onResult(int result, String message) {
        if ("succeed".equals(message)) {
            //调用成功
        } else {
            //调用失败
        }
    }
};

//速度为speed，持续前进
RobotApi.getInstance().goForward(reqId, speed, motionListener);

//速度为speed，前进距离为distance
RobotApi.getInstance().goForward(reqId, speed, distance, motionListener);

//速度为speed，持续后退
RobotApi.getInstance().goBackward(reqId, speed, motionListener);

//速度为speed，后退距离为distance
RobotApi.getInstance().goBackward(reqId, speed, distance, motionListener);
```

参数

- **speed**：运动速度，单位m/s，范围为 0 ~ 1.0, 大于1.0的速度按1.0运动
- **distance**：运动距离，单位m，值需大于0

2. 旋转运动

方法名称：turnLeft / turnRight
调用方式：

```
CommandListener rotateListener = new CommandListener() {
```

```
    @Override
    public void onResult(int result, String message) {
        if ("succeed".equals(message)) {
            //调用成功
        } else {
            //调用失败
        }
    }
};

//速度为speed，持续左转
RobotApi.getInstance().turnLeft(reqId, speed, rotateListener);

//速度为speed，左转angle度
RobotApi.getInstance().turnLeft(reqId, speed, angle, rotateListener);

//速度为speed，持续右转
RobotApi.getInstance().turnRight(reqId, speed, rotateListener);

//速度为speed，右转angle度
RobotApi.getInstance().turnRight(reqId, speed, angle, rotateListener);
```

参数

> - **speed**：旋转速度，单位：度/s，范围为 0 ~ 50 度/s
> - **angle**：旋转角度，单位：度，值需大于0

3. 停止

方法名称：stopMove
调用方式

```
RobotApi.getInstance().stopMove(reqId, new CommandListener() {
    @Override
    public void onResult(int result, String message) {
        if ("succeed".equals(message)) {
            //调用成功
        } else {
            //调用失败
        }
    }
});
```

注意：该接口只可用于停止前进、后退及旋转，不可用于停止导航及头部运动

4. 头部云台运动

方法名称：moveHead
调用方式：

```
RobotApi.getInstance().moveHead(reqId, hMode, vMode, hAngle, vAngle, new C
ommandListener() {
    @Override
    public void onResult(int result, String message) {
        try {
            JSONObject json = new JSONObject(message);
            String status = json.getString("status");
            if (Definition.CMD_STATUS_OK.equals(status)) {
                //操作成功
            }
        } catch (JSONException | NullPointerException e) {
            e.printStackTrace();
        }
    }
});
```

参数

- **hMode**：左右转动的模式，绝对运动：absolute，相对运动：relative
- **vMode**：上下运动的模式，绝对运动：absolute，相对运动：relative
- **hAngle**：左右转动角度，范围：-120 ~ 120
- **vAngle**：上下运动的角度，范围：0 ~ 90

5. 恢复云台初始角度

方法名称：resetHead
调用方式：

```
RobotApi.getInstance().resetHead(reqId, new CommandListener() {
    @Override
    public void onResult(int result, String message) {
        try {
            JSONObject json = new JSONObject(message);
            String status = json.getString("status");
            if (Definition.CMD_STATUS_OK.equals(status)) {
                //操作成功
            }
        } catch (JSONException | NullPointerException e) {
            e.printStackTrace();
        }
    }
});
```

6. 切换摄像头

方法名称：switchCamera
调用方式：

```
RobotApi.getInstance().switchCamera(reqId, camera, new CommandListener() {
    @Override
    public void onResult(int result, String message) {
        try {
            JSONObject json = new JSONObject(message);
            if (Definition.RESPONSE_OK.equals(json.getString("status"))) {
                //切换成功
            }
        } catch (JSONException | NullPointerException e) {
            e.printStackTrace();
        }
    }
});
```

参数

- **camera**：摄像头，String 类型
  Definition.JSON_HEAD_FORWARD 前置摄像头
  Definition.JSON_HEAD_BACKWARD 后置摄像头

7. 停止充电

方法名称：stopChargingByApp
调用方式：

```
RobotApi.getInstance().stopChargingByApp();
```

地图及位置服务

1. 建图及定位

建图及定位可使用系统集成的地图工具进行操作

2. 定位（设置机器人初始坐标点）

方法名称：setPoseEstimate
调用方式：

```
try {
    JSONObject params = new JSONObject();
    //x坐标
    params.put(Definition.JSON_NAVI_POSITION_X, x);
    //y坐标
    params.put(Definition.JSON_NAVI_POSITION_Y, y);
    //z坐标
    params.put(Definition.JSON_NAVI_POSITION_THETA, theta);
    RobotApi.getInstance().setPoseEstimate(reqId, params.toString(), new C
ommandListener() {
        @Override
        public void onResult(int result, String message) {
            if ("succeed".equals(message)) {
                //定位成功
            }
        }
    });
} catch (JSONException e) {
    e.printStackTrace();
}
```

3. 判断当前是否已定位

方法名称：isRobotEstimate
调用方式：

```
RobotApi.getInstance().isRobotEstimate(reqId, new CommandListener() {
    @Override
    public void onResult(int result, String message) {
        if (!"true".equals(message)) {
            //当前未定位
        } else {
            //当前已定位
        }
    }
});
```

4. 设置当前位置名称

方法名称：setLocation
调用方式：

```
RobotApi.getInstance().setLocation(reqId, placeName, new CommandListener()
{
    @Override
    public void onResult(int result, String message) {
        if ("succeed".equals(message)) {
```

```
                //保存位置点成功
            } else {
                //保存位置点失败
            }
        }
    });
```

参数

- **placeName**：位置名称

注意：调用该接口前需要确保已定位

5. 根据位置名称获取坐标点

方法名称：getLocation
调用方式：

```
RobotApi.getInstance().getLocation(reqId, placeName, new CommandListener()
{

    @Override
    public void onResult(int result, String message) {
        try {
            JSONObject json = new JSONObject(message);
            //当前位置是否存在
            boolean isExist = json.getBoolean(Definition.JSON_NAVI_SITE_EX
IST);
            if (isExist) {
                //x坐标
                double x = json.getDouble(Definition.JSON_NAVI_POSITION_
X);
                //y坐标
                double y = json.getDouble(Definition.JSON_NAVI_POSITION_
Y);
                //z坐标
                double z = json.getDouble(Definition.JSON_NAVI_POSITION_TH
ETA);

            }
        } catch (JSONException | NullPointerException e) {
            e.printStackTrace();
        }
    }
});
```

参数

- **placeName**：位置名称

注意：setLocation保存的位置会与地图关联，通过getLocation获取的时候也应该保持相同的地图，否则getLocation获取失败

6. 删除位置点

方法名称：removeLocation
调用方式：

```
RobotApi.getInstance().removeLocation(reqId, placeName, new CommandListene
r() {
    @Override
    public void onResult(int result, String message) {
        if ("succeed".equals(message)) {
            //删除位置点成功
        } else {
            //删除位置点失败
        }
    }
});
```

7. 获取当前地图所有位置点

方法名称：getPlaceList
调用方式：

```
RobotApi.getInstance().getPlaceList(reqId, new CommandListener() {

    @Override
    public void onResult(int result, String message) {
        try {
            JSONArray jsonArray = new JSONArray(message);
            int length = jsonArray.length();
            for (int i = 0; i < length; i++) {
                JSONObject json = jsonArray.getJSONObject(i);
                json.getDouble("x"); //x坐标
                json.getDouble("y"); //y坐标
                json.getDouble("theta"); //z坐标
                json.getString("name"); //位置名称
            }
        } catch (JSONException | NullPointerException e) {
            e.printStackTrace();
        }
    }
});
```

8. 获取机器人当前坐标点

方法名称：getPosition
调用方式：

```
RobotApi.getInstance().getPosition(reqId, new CommandListener() {
    @Override
    public void onResult(int result, String message) {
        try {
            JSONObject json = new JSONObject(message);
            //x坐标
            double x = json.getDouble(Definition.JSON_NAVI_POSITION_X);
            //y坐标
            double y = json.getDouble(Definition.JSON_NAVI_POSITION_Y);
            //z坐标
            double z = json.getDouble(Definition.JSON_NAVI_POSITION_THET
A);
        } catch (JSONException | NullPointerException e) {
            e.printStackTrace();
        }
    }
});
```

注意：调用该接口前需要确保已定位

9. 判断机器人是否在位置点

方法名称：isRobotInLocations
调用方式：

```
try {
    JSONObject params = new JSONObject();
    params.put(Definition.JSON_NAVI_TARGET_PLACE_NAME, placeName); //位置名
称
    params.put(Definition.JSON_NAVI_COORDINATE_DEVIATION, range); //位置范
围
    RobotApi.getInstance().isRobotInlocations(reqId,
            params.toString(), new CommandListener() {
                @Override
                public void onResult(int result, String message) {
                    try {
                        JSONObject json = new JSONObject(message);
                        //是否在目标点
                        json.getBoolean(Definition.JSON_NAVI_IS_IN_LOCATIO
N);
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
```

```
            });
} catch (JSONException e) {
    e.printStackTrace();
}
```

参数

- **placeName**：位置名称
- **range**：位置范围，单位m

10. 获取当前地图名称

方法名称：getMapName
调用方式：

```
RobotApi.getInstance().getMapName(reqId, new CommandListener() {
    @Override
    public void onResult(int result, String message) {
        if (!TextUtils.isEmpty(message)) {
            //message为地图名称
            String mapName = message;
        }
    }
});
```

11. 切换地图

方法名称：switchMap
调用方式：

```
RobotApi.getInstance().switchMap(reqId, mapName, new CommandListener(){
    @Override
    public void onResult(int result, String message) {
        if ("succeed".equals(message)) {
            //切换地图成功
        }
    }
});
```

参数

- **mapName**：地图名称

注意：切换地图后需要重新定位

导航服务

1. 导航到指定位置

方法名称：startNavigation
结果回调：

```
ActionListener navigationListener = new ActionListener() {

    @Override
    public void onResult(int status, String response) throws RemoteExcepti
on {
        switch (status) {
            case Definition.RESULT_OK:
                if ("true".equals(response)) {
                    //导航成功
                } else {
                    //导航失败
                }
                break;
        }
    }

    @Override
    public void onError(int errorCode, String errorString) throws RemoteEx
ception {
        switch (errorCode) {
            case Definition.ERROR_NOT_ESTIMATE:
                //当前未定位
                break;
            case Definition.ERROR_IN_DESTINATION:
                //当前机器人已经在目的地范围内
                break;
            case Definition.ERROR_DESTINATION_NOT_EXIST:
                //导航目的地不存在
                break;
            case Definition.ERROR_DESTINATION_CAN_NOT_ARRAIVE:
                //避障超时，目的地不能到达，超时时间通过参数设置
                break;
            case Definition.ACTION_RESPONSE_ALREADY_RUN:
                //当前接口已经调用，请先停止，才能再次调用
                break;
            case Definition.ACTION_RESPONSE_REQUEST_RES_ERROR:
                //已经有需要控制底盘的接口调用，请先停止，才能继续调用
                break;
        }
    }

    @Override
```

```
    public void onStatusUpdate(int status, String data) throws RemoteExcep
tion {
        switch (status) {
            case Definition.STATUS_NAVI_AVOID:
                //当前路线已经被障碍物堵死
                break;
            case Definition.STATUS_NAVI_AVOID_END:
                //障碍物已移除
                break;
        }
    }
};
```

调用方式：

1. 默认导航速度

```
RobotApi.getInstance().startNavigation(reqId, destName, coordinateDeviatio
n, time, navigationListener);
```

2. 指定导航速度

   该调用方式在V4.2版本开始支持

```
RobotApi.getInstance().startNavigation(reqId, destName, coordinateDeviatio
n, time, linearSpeed, angularSpeed, navigationListener);
```

3. 指定坐标点

   该调用方式在V4.12版本开始支持

```
RobotApi.getInstance().startNavigation(reqId, pose, coordinateDeviation, t
ime, navigationListener);
```

参数：

- **destName**：导航目的地名称（必须先通过setLocation设置）
- **pose**：导航目的地坐标点
- **coordinateDeviation**：目的地范围，如果距离目的地在该范围内，则认为已到达
- **time**：避障超时时间，如果该时间内机器人的移动距离不超过0.1m，则导航失败
- **linearSpeed**：导航线速度，范围：0.1 ~ 0.85 m/s 默认值：0.7 m/s
- **angularSpeed**：导航角速度，范围：0.4 ~ 1.4 m/s 默认值：1.2 m/s
  最终导航速度是结合线速度和角速度换算后得到，不同的线速度和角速度对导航运动方式有影响，建议线速度和角速度保持一定规律：angularSpeed = 0.4 + (linearSpeed - 0.1) / 3 * 4

注意：调用该接口前需要确保已定位

2. 停止导航到指定位置

方法名称：stopNavigation
调用方式:

```
RobotApi.getInstance().stopNavigation(reqId);
```

注意：该接口只能用于停止startNavigation启动的导航

3. 导航到指定坐标点

方法名称：goPosition
结果回调:

```
CommandListener goPositionListener = new CommandListener() {
    @Override
    public void onResult(int status, String response) {
        switch (status) {
            case Definition.RESULT_OK:
                if ("true".equals(response)) {
                    //导航成功
                } else {
                    //导航失败
                }
                break;
        }
    }

    @Override
    public void onError(int errorCode, String errorString) throws RemoteEx
ception {
        switch (errorCode) {
            case Definition.ACTION_RESPONSE_ALREADY_RUN:
                //当前接口已经调用，请先停止，才能再次调用
                break;
            case Definition.ACTION_RESPONSE_REQUEST_RES_ERROR:
                //已经有需要控制底盘的接口调用，请先停止，才能继续调用
                break;
        }
    }

    @Override
    public void onStatusUpdate(int status, String data) {
        switch (status) {
            case Definition.STATUS_NAVI_AVOID:
                //当前路线已经被障碍物堵死
                break;
```

```
            case Definition.STATUS_NAVI_AVOID_END:
                //障碍物已移除
                break;
        }
    }
};
```

调用方式:

1. 默认速度

```
try {
    JSONObject position = new JSONObject();
    //x坐标
    position.put(Definition.JSON_NAVI_POSITION_X, x);
    //y坐标
    position.put(Definition.JSON_NAVI_POSITION_Y, y);
    //z坐标
    position.put(Definition.JSON_NAVI_POSITION_THETA, theta);
    RobotApi.getInstance().goPosition(reqId, position.toString(), goPositi
onListener);
} catch (JSONException e) {
    e.printStackTrace();
}
```

2. 指定速度

该调用方式在V4.2版本开始支持

```
try {
    JSONObject position = new JSONObject();
    //x坐标
    position.put(Definition.JSON_NAVI_POSITION_X, x);
    //y坐标
    position.put(Definition.JSON_NAVI_POSITION_Y, y);
    //z坐标
    position.put(Definition.JSON_NAVI_POSITION_THETA, theta);
    RobotApi.getInstance().goPosition(reqId, position.toString(),  linearS
peed, angularSpeed, goPositionListener);
} catch (JSONException e) {
    e.printStackTrace();
}
```

参数

- **position**: 参数为json格式的坐标点
  {
  “x”: “x坐标”,

"y": "y坐标",
"theta: "z坐标",
}

- **linearSpeed**：导航线速度，范围：0.1 ~ 0.85 m/s 默认值：0.7 m/s
- **angularSpeed**：导航角速度，范围：0.4 ~ 1.4 m/s 默认值：1.2 m/s
  最终导航速度是结合线速度和角速度换算后得到，不同的线速度和角速度对导航运动方式有影响，建议线速度和角速度保持一定规律：angularSpeed = 0.4 + (linearSpeed - 0.1) / 3 * 4

注意：调用该接口前需要确保已定位

4. 停止导航到指定坐标点

方法名称：stopGoPosition
调用方式：

```
RobotApi.getInstance().stopGoPosition(reqId);
```

注意：goPosition 与 startNavigation 两个启动导航的接口，都有一个对应的停止接口，必须一一对应，不能混用

5. 转向目标点方向

方法名称：resumeSpecialPlaceTheta

调用方式：

```
RobotApi.getInstance().resumeSpecialPlaceTheta(reqId,
    placeName,
    new CommandListener() {
        @Override
        public void onResponse(int result, String message) {
        }
  });
```

参数

- **reqId**: int类型 命令id
- **placeName**: String类型 目标点名称
- **listener**: CommandListener类型 消息回调

返回值

- int result 0 命令执行 / -1 没有执行

注意：调用该接口前需要确保已定位

基础场景

1. 引领

开始引领

方法名称：startLead

调用方式

```
LeadingParams params = new LeadingParams();
//引领目标人脸id
params.setPersonId(faceId);
//引领目的地
params.setDestinationName(dest);
//引领目标找不到多久后上报丢失状态
params.setLostTimer(lostTime);
//引领开始多久后开启人脸识别
params.setDetectDelay(delayTime);
//引领目标距离机器人多远后上报超距状态
params.setMaxDistance(maxDistance);
RobotApi.getInstance().startLead(reqId, params, new ActionListener() {

    @Override
    public void onResult(int status, String responseString) throws RemoteE
xception {
        switch (status) {
            case Definition.RESULT_OK:
                //成功将目标引领到目的地
                break;

            case Definition.ACTION_RESPONSE_STOP_SUCCESS:
                //在引领执行中，主动调用stopLead，成功停止引领
                break;

            default:
                break;
        }
    }

    @Override
    public void onError(int errorCode, String errorString) throws RemoteEx
ception {
        switch (errorCode) {
            case Definition.ERROR_NOT_ESTIMATE:
                //当前未定位
                break;

            case Definition.ERROR_SET_TRACK_FAILED:
```

```
            case Definition.ERROR_TARGET_NOT_FOUND:
                //引领目标未找到
                break;

            case Definition.ERROR_IN_DESTINATION:
                //当前已经在引领目的地
                break;

            case Definition.ERROR_DESTINATION_CAN_NOT_ARRAIVE:
                //避障超时，默认为机器人20s的前进距离不足0.1m,
                break;

            case Definition.ERROR_DESTINATION_NOT_EXIST:
                //引领目的地不存在
                break;

            case Definition.ERROR_HEAD:
                //引领中操作头部云台失败
                break;

            case Definition.ACTION_RESPONSE_ALREADY_RUN:
                //引领已经在进行中，请先停止上次引领，才能重新执行
                break;

            case Definition.ACTION_RESPONSE_REQUEST_RES_ERROR:
                //已经有需要控制底盘的接口调用，请先停止，才能继续调用
                break;

            default:
                break;
        }
    }

    @Override
    public void onStatusUpdate(int status, String data) throws RemoteExcep
tion {
        switch (status) {
            case Definition.STATUS_NAVI_OUT_MAP:
                //目标点不能到达，引领目的地在地图外，有可能为地图与位置点不匹配，请
重新设置位置点
                break;

            case Definition.STATUS_NAVI_AVOID:
                //当前引领路线已被障碍物堵死
                break;

            case Definition.STATUS_NAVI_AVOID_END:
                //障碍物已移除
                break;

            case Definition.STATUS_GUEST_FARAWAY:
                //引领目标距离机器人太远，判断标准通过参数maxDistance设置
```

```
                break;

            case Definition.STATUS_DEST_NEAR:
                //引领目标进入机器人maxDistance范围内
                break;

            case Definition.STATUS_LEAD_NORMAL:
                //正式开始导航
                break;

            default:
                break;
        }
    }
});
```

结束引领
方法名称：stopLead
调用方式：

```
RobotApi.getInstance().stopLead(reqId, isResetHW);
```

参数：

> - **isResetHW**： 引领时会切换摄像头到后置摄像头，isResetHW是用于设置停止引领时是否恢复摄像头状态，true：恢复摄像头为前置，false：保持停止时的状态

描述

> 引领是方便用户二次开发而实现的一个简单的业务场景，内部集成了人脸识别和导航，可在导航中持续识别引领目标，并可实时上报引领目标的状态，用户二次开发时可根据上报的状态进行处理，更快的实现自身业务需求

2. 焦点跟随

开启焦点跟随
方法名称：startFocusFollow
调用方式：

```
RobotApi.getInstance().startFocusFollow(reqId, faceId, lostTimeout, maxDis
tance, new ActionListener() {

    @Override
    public void onStatusUpdate(int status, String data) {
        switch (status) {
            case Definition.STATUS_TRACK_TARGET_SUCCEED:
```

```
                    //跟随目标成功
                    break;

                case Definition.STATUS_GUEST_LOST:
                    //跟随目标丢失
                    break;

                case Definition.STATUS_GUEST_FARAWAY:
                    //跟随目标距离已大于设置的最大距离
                    break;

                case Definition.STATUS_GUEST_APPEAR:
                    //跟随目标重新进入设置的最大距离内
                    break;
            }
        }

        @Override
        public void onError(int errorCode, String errorString) {
            switch (errorCode) {
                case Definition.ERROR_SET_TRACK_FAILED:
                case Definition.ERROR_TARGET_NOT_FOUND:
                    //跟随目标未找到
                    break;

                case Definition.ACTION_RESPONSE_ALREADY_RUN:
                    //正在跟随中，请先停止上次跟随，才能重新执行
                    break;

                case Definition.ACTION_RESPONSE_REQUEST_RES_ERROR:
                    //已经有需要控制底盘的接口调用(例如：引领、导航)，请先停止，才能继续
调用
                    break;
            }
        }

        @Override
        public void onResult(int status, String responseString) {
            Log.d(TAG, "startTrackPerson onResult status: " + status);
            switch (status) {
                case Definition.ACTION_RESPONSE_STOP_SUCCESS:
                    //在焦点跟随过程中，主动调用stopFocusFollow，成功停止跟随
                    break;
            }
        }
    });
```

参数

- **faceId**：焦点跟随的目标人脸id
- **lostTimeout**：多久识别不到目标上报目标丢失状态

- **maxDistance**：目标距离多远上报超距状态

停止焦点跟随
方法名称：
调用方式：

```
RobotApi.getInstance().stopFocusFollow(reqId);
```

描述：

> 焦点跟随会根据用户指定的人脸id，持续识别并跟踪，头部云台会持续跟着目标转动，底盘也会跟着联动，直到目标静止，机器人正对目标

3. 巡航

开始巡航
方法名称：startCruise
结果回调：

```
ActionListener cruiseListener = new ActionListener() {

    @Override
    public void onResult(int status, String responseString) throws RemoteE
xception {
        Log.i(TAG, "startCruise onResult : " + status + " || " + responseS
tring);
        switch (status) {
            case Definition.RESULT_OK:
                //巡航完成
                break;

            case Definition.ACTION_RESPONSE_STOP_SUCCESS:
                //在巡航过程中，主动调用stopCruise，成功停止巡航
                break;
        }
    }

    @Override
    public void onStatusUpdate(int status, String data) throws RemoteExcep
tion {
        Log.i(TAG, "startCruise onStatusUpdate : " + status + " || " + dat
a);
        switch (status) {
            case Definition.STATUS_NAVI_OUT_MAP:
                //目标点不能到达，巡航点在地图外，请重新设置巡航路线
                break;
```

```
                case Definition.STATUS_START_CRUISE:
                    //开始巡航
                    break;

                case Definition.STATUS_CRUISE_REACH_POINT:
                    int index = Integer.parseInt(data);
                    //巡航已到达第几个巡航点，index为巡航点在路线集合中的下标
                    break;

                case Definition.STATUS_NAVI_AVOID:
                    //当前巡航路线已被障碍物堵死
                    break;

                case Definition.STATUS_NAVI_AVOID_END:
                    //障碍物移除
                    break;
            }
        }

        @Override
        public void onError(int errorCode, String errorString) throws RemoteEx
ception {
            Log.i(TAG, "startCruise onError : " + errorCode + " || " + errorSt
ring);
            switch (errorCode) {
                case Definition.ACTION_RESPONSE_ALREADY_RUN:
                    //巡航已经在进行
                    break;

                case Definition.ERROR_NOT_ESTIMATE:
                    //当前未定位
                    break;

                case Definition.ERROR_NAVIGATION_FAILED:
                    //巡航点导航失败
                    break;

                case Definition.ACTION_RESPONSE_REQUEST_RES_ERROR:
                    //已经有需要控制底盘的接口调用(例如：引领、导航等)，请先停止，才能继
续调用
                    break;
            }
        }
};
```

调用方式：

1. 默认速度

```
RobotApi.getInstance().startCruise(reqId, route, startPoint, dockingPoint
s, cruiseListener);
```

2. 自定义速度

```
RobotApi.getInstance().startCruise(reqId, route, startPoint, dockingPoint
s, linearSpeed, angularSpeed, cruiseListener);
```

参数

- **route**：巡航路线，坐标点集合
- **startPoint**：巡航起始点下标，从第几个点开始巡航
- **dockingPoints**：停靠点下标，默认到达巡航点0.5m内会直接开始导航到下一巡航点，停靠点必须坐标及方向符合才认为到达，然后继续下一目标点
- **linearSpeed**：导航线速度，范围：0.1 ~ 0.85 m/s 默认值：0.7 m/s
- **angularSpeed**：导航角速度，范围：0.4 ~ 1.4 m/s 默认值：1.2 m/s
  最终导航速度是结合线速度和角速度换算后得到，不同的线速度和角速度对导航运动方式有影响，建议线速度和角速度保持一定规律：angularSpeed = 0.4 + (linearSpeed - 0.1) / 3 * 4

停止巡航
方法名称：stopCruise
调用方式：

```
RobotApi.getInstance().stopCruise(reqId);
```

描述：

> 巡航是对导航的一次简单封装，支持一系列点的连续导航，支持起始点和停靠点的设置，会持续上报巡航状态。目前巡航只支持单次巡航，不支持循环持续巡航，如果需要的话可在单次巡航完成后直接开启下次巡航

4. 唤醒

开始唤醒
方法名称：wakeUp
调用方式：

```
RobotApi.getInstance().wakeUp(reqId, angle, new ActionListener() {

    @Override
    public void onResult(int status, String responseString) throws RemoteE
xception {
        switch (status) {
            case Definition.RESULT_OK:
                //唤醒完成
```

```
                break;

            case Definition.ACTION_RESPONSE_STOP_SUCCESS:
                //在唤醒过程中，主动调用stopWakeUp，停止唤醒
                break;
        }
    }

    @Override
    public void onError(int errorCode, String errorString) throws RemoteEx
ception {
        switch (errorCode) {
            case Definition.ERROR_MOVE_HEAD_FAILED:
                //头部云台移动失败
                break;

            case Definition.ACTION_RESPONSE_ALREADY_RUN:
                //当前正在唤醒中，需要先停止上次唤醒
                break;

            case Definition.ACTION_RESPONSE_REQUEST_RES_ERROR:
                //已经有需要控制底盘的接口调用(例如：引领、导航等)，请先停止，才能继
续调用
                break;
        }
    }
});
```

参数
angle：声源定位角度，通过ModuleCallback的onSendRequest回调获得，当语音唤醒时，reqType为Definition.REQ_SPEECH_WAKEUP，reqParams为声源定位的角度

停止唤醒
方法名称：stopWakeUp
调用方式：

```
RobotApi.getInstance().stopWakeUp(reqId);
```

描述：

> 唤醒可控制机器人转动到用户指定的角度，当角度小于45度时，只转动头部云台，大于45度时，头部云台及底盘均会转动，以最快的速度转动到声源定位的目标角度

语音服务

1. 创建语音服务回调

```
SkillCallback mSkillCallback = new SkillCallback() {
    @Override
    public void onSpeechParResult(String s) throws RemoteException {
        //语音临时识别结果
    }

    @Override
    public void onStart() throws RemoteException {
        //开始识别
    }

    @Override
    public void onStop() throws RemoteException {
        //识别结束
    }

    @Override
    public void onVolumeChange(int volume) throws RemoteException {
        //识别的声音大小变化
    }

    /**
     * @param status 0 : 正常返回
     *               1 : other返回
     *               2 : 噪音或single_other返回
     *               3 : 超时
     *               4 : 被强制取消
     *               5 : asr结果提前结束，未经过NLU
     *               6 : 全双工语意相同情况下，other返回
     */
    @Override
    public void onQueryEnded(int status) throws RemoteException {

    }

    @Override
    public void onQueryAsrResult(String asrResult) throws RemoteException
{
        //asrResult ： 最终识别结果
    }
};
```

2. 连接语音服务

```
final SkillApi skillApi = new SkillApi();
skillApi.connectApi(context, new ApiListener() {
    @Override
    public void handleApiDisabled() {
    }
```

```
    @Override
    public void handleApiConnected() {
        //语音服务连接成功，注册语音回调
        skillApi.registerCallBack(mSkillCallback);
    }

    @Override
    public void handleApiDisconnected() {
        //语音服务已断开
    }
});
```

3. TTS播放

方法名称：playText
调用方式

```
skillApi.playText(text, new TextListener() {
    @Override
    public void onStart() {
        //播放开始
    }

    @Override
    public void onStop() {
        //播放停止
    }

    @Override
    public void onError() {
        //播放错误
    }

    @Override
    public void onComplete() {
        //播放完成
    }
});
```

参数

> - **text**：TTS播放文本

4. 停止TTS播放

方法名称：stopTTS
调用方式：

```
skillApi.stopTTS();
```

5. 设置语音识别模式

方法名称：setRecognizeMode
调用方式：

```
skillApi.setRecognizeMode(isContinue);
```

参数

- **isContinue**：boolean类型，true 持续识别 false 单次识别

6. 关闭或开启语音识别

方法名称：setRecognizable
调用方式：

```
skillApi.setRecognizable(enable);
```

参数

- **enable**：boolean类型，true 开启 false 关闭

~~####7. 关闭或开启语音（语音识别、语音合成）~~

~~方法名称：setASREnabled~~

~~调用方式：~~

```
skillApi.setASREnabled(enable);
```

~~参数~~

- **enable**：boolean类型，true 开启 false 关闭

注意：该接口在4.10已经废弃，该接口涉及到两种调用场景 1. 不想识别，请使用setRecognizable 2. 创建Audio录音，目前已经支持多个Audio，直接创建即可，创建方法如下：new

AudioRecord(MediaRecorder.AudioSource.MIC, 16000,
AudioFormat.CHANNEL_IN_STEREO, AudioFormat.ENCODING_PCM_16BIT, 5 * 2 *16000);

8. 设置语音识别区域

方法名称：setAngleCenterRange
调用方式：

```
skillApi.setAngleCenterRange(angleCenter, angleRange);
```

参数

- **angleCenter**：中心角度，float类型，[0,360)
- **angleRange**：区间角度，float类型，[0,120]

9. 通过文本获取NLP结果

方法名称：queryByText
调用方式：

```
skillApi.queryByText(text);
```

参数

- **text**：需要query的文本，String类型

注意：该接口无返回值，query的结果通过ModuleCallback的onSendRequest，当作一条语音指令返回

10.音频数据采集

注意：所有参数为固定参数，不可变更，否则有可能采集不到数据

```
//音频采样率（固定16000）
private final int AUDIO_RATE = 16000;
//Buffer长度
private final int AUDIO_BUF_LEN = 640;

//声道（立体声）
int channelConfig = AudioFormat.CHANNEL_IN_STEREO;
int minRecBufSize = AudioRecord.getMinBufferSize(AUDIO_RATE, channelConfi
g, AudioFormat.ENCODING_PCM_16BIT);
int recBufSize = minRecBufSize * 2;
```

```
AudioRecord audioRecorder  =new AudioRecord(AudioSource.MIC, AUDIO_RATE, A
udioFormat.CHANNEL_IN_STEREO, AudioFormat.ENCODING_PCM_16BIT, recBufSize);

//数据采集
byte[] tempBufRec = new byte[AUDIO_BUF_LEN];
byte[] bfOutLeft = new byte[AUDIO_BUF_LEN / 2];
byte[] bfOutRight = new byte[AUDIO_BUF_LEN / 2];
int bufferSize = 5 * 2 * 16000;
int readBytes = audioRecorder.read(tempBufRec, 0, bufferSize);
if (readBytes  == bufferSize) {
    deinterleaveData(tempBufRec, bfOutLeft, bfOutRight, bufferSize);
    //分离后的数据，bfOutLeft为最终需要的数据
}

//左右声道数据分离
private void deinterleaveData(byte[] src, byte[] leftdest, byte[] rightdes
t, int len) {
    int rIndex = 0;
    int lIndex = 0;
    for (int i = 0; i < len; ) {
        leftdest[rIndex] = src[i];
        leftdest[rIndex + 1] = src[i + 1];
        rIndex = rIndex + 2;

        rightdest[lIndex] = src[i + 2];
        rightdest[lIndex + 1] = src[i + 3];
        lIndex = lIndex + 2;

        i = i + 4;
    }
}
```

电量控制

1. 开始自动回充

方法名称：startNaviToAutoChargeAction
调用方式：

```
RobotApi.getInstance().startNaviToAutoChargeAction(reqId, timeout, new Act
ionListener() {

    @Override
    public void onResult(int status, String responseString) throws RemoteE
xception {
        switch (status) {
            case Definition.RESULT_OK:
                //充电成功
```

```
                break;

            case Definition.RESULT_FAILURE:
                //充电失败
                break;
        }
    }

    @Override
    public void onStatusUpdate(int status, String data) throws RemoteExcep
tion {
        switch (status) {
            case Definition.STATUS_NAVI_GLOBAL_PATH_FAILED:
                //全局路径规划失败
                break;

            case Definition.STATUS_NAVI_OUT_MAP:
                //目标点不能到达，引领目的地在地图外，有可能为地图与位置点不匹配，请
重新设置位置点
                break;

            case Definition.STATUS_NAVI_AVOID:
                //前往回充点路线已被障碍物堵死
                break;

            case Definition.STATUS_NAVI_AVOID_END:
                //障碍物已移除
                break;

            default:
                break;
        }
    }
});
```

参数

- **timeout**：导航超时时间，如果超过该时间未到达充电桩位置，则认为回充失败

2. 结束自动回充

方法名称：stopAutoChargeAction
调用方式：

```
RobotApi.getInstance().stopAutoChargeAction(reqId, true);
```

3. 停止充电并脱离充电桩

方法名称：stopChargingByApp
调用方式：

```
RobotApi.getInstance().stopChargingByApp();
```

系统功能

1. 关闭充电中系统接管接口

方法名称：disableBattery

调用方式：

```
RobotApi.getInstance().disableBattery();
```

参数

- 无

返回值

- 无

2. 恢复系统接管接口

方法名称：resetSystemStatus

调用方式：

```
RobotApi.getInstance().resetSystemStatus();
```

参数

- 无

返回值

- 无

3. 安装apk

方法名称：installApk

调用方式：

```
boolean status = RobotApi.getInstance().installApk(reqId, fullPathName, ta
skId);

结果通过广播通知，实例代码如下：
public class InstalledAppReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {

        if (intent != null
              && "ainirobot.intent.action.INSTALL_RESULT".equals(intent.ge
tAction())) {
            int taskId = intent.getIntExtra("task_id", 0);
            boolean result = intent.getBooleanExtra("result", false);
            String errorMsg = intent.getStringExtra("error_msg");
        }

    }
}
```

参数

- **reqId**: int类型命令id，传0
- **fullPathName** String类型 安装文件路径
- **taskID** String类型 安装任务id，安装多个应用时，区分安装进度

返回值

- boolean status true命令执行 / false命令没有执行(通常是binder died)

4. 设置灯效

方法名称：setLight

调用方式：

```
    JSONObject params = new JSONObject();
    try {
        params.put(Definition.JSON_LAMB_TYPE, type);
        params.put(Definition.JSON_LAMB_TARGET, 0);
        params.put(Definition.JSON_LAMB_RGB_START, startColor);
        params.put(Definition.JSON_LAMB_RGB_END, endColor);
        params.put(Definition.JSON_LAMB_START_TIME, startTime);
        params.put(Definition.JSON_LAMB_END_TIME, endTime);
```

```
            params.put(Definition.JSON_LAMB_REPEAT, loopTime);
            params.put(Definition.JSON_LAMB_ON_TIME, duration);
            params.put(Definition.JSON_LAMB_RGB_FREEZE, finalColor);
        } catch (JSONException e) {
            e.printStackTrace();
        }

        RobotApi.getInstance().setLight(0, params.toString(), null);
```

参数

- **reqId**: int类型 命令id
- **params**: String类型, 可包含如下参数:
  type - Lighting effect type
  start - Start color
  end - End color
  startTime - Start time
  endTime - End time
  repeat - Loop times
  onTime - Duration
  freese - end lighting color
- **listener**: ActionListener类型 消息回调

返回值

- int result 0 命令执行 / -1 没有执行

5. text转mp3

方法名称：textToMp3

调用方式:

```
boolean status = RobotApi.getInstance().textToMp3(reqId, text, path, name,
    new CommandListener(){
        @Override
        public void onResult(int result, String message) {

            if (Definition.RESULT_OK == result) {
                try {
                    JSONObject jsonObject = new JSONObject(message);
                    int code = jsonObject.optInt("code");
                    String errMessage = jsonObject.optString("message");
                    if (code == 0) {
                        mediaPlayer = new MediaPlayer();
                        mediaPlayer.setDataSource(path + File.separator +
 name);
```

```
                        mediaPlayer.prepare();
                        mediaPlayer.start();
                    } else{

                    }
                } catch (JSONException | IOException e) {
                    e.printStackTrace();
                }
            }else {

            }

        }

    });
```

参数

- **reqId**: int类型命令id，传0
- **text** String类型 需要转换的text
- **path** String类型 文件路径
- **name** String类型 文件名
- **listener** CommandListener 消息回调{“code”:0, “message”:”err msg”}

返回值

- int result 0 命令执行 / -1 没有执行

注意： 1. 调用接口需要app申请sdcard使用权限 2. 转换成功会在指定路径下生成/覆盖生成mp3文件 3. 请自己管理该文件

6. 获取本机SN

方法名称：getRobotSn

调用方式：

```
boolean status = RobotApi.getInstance().getRobotSn(
    new CommandListener(){
        @Override
        public void onResult(int result, String message) {
            if (Definition.RESULT_OK == result) {
                String serialNum = message;
            } else {

            }
```

```
        }
    });
```

参数

- **listener** CommandListener 消息回调{“code”:0, “message”:”err msg”}

返回值

- int result 1 命令执行 / -1 没有执行

7. 获取系统版本

方法名称：getVersion

调用方式：

```
String version = RobotApi.getInstance().getVersion();
```