

FireflyApi编程接口（SDK）

[概述](#)

[历史分支](#)

[开发环境](#)

[SDK 集成说明](#)

[FireflyApi 接口说明](#)

[1.1 ICard/身份证/](#)

[1.2 补光灯控制](#)

[1.3 信号控制](#)

[1.4 二维码](#)

[1.5 体温检测](#)

[1.6 雷达](#)

FireflyApi编程接口（SDK）

概述

FireflyApi SDK，包含了IDCard（身份证）、ICCard、二维码等相关功能，FireflyApi 根据自身的相关业务需求结合 SDK 的相关接口灵活的进行上层软件应用的开发。

历史分支

日期	版本	说明	修改人
2020-12-21	1.2	优化IDCard功能	肖大涛，宋建峰
2020-07-17	1.1	V1.1版	肖大涛，宋建峰
2020-05-21	1.0	初版完成	宋建峰

开发环境

1. Android Studio >=3.6.2
2. Gradle Version: 3.6.2
3. Gradle Plugin Version: 3.6.2
4. SDK Tool >=25.2.3

SDK 集成说明

- 1.将 Demo工程 fireflyApi/libs/armeabi-v7a 目录下的所有so包拷贝到 AS 工程对应的 libs/armeabi-v7a 文件夹下；
- 2.将 Demo工程 fireflyApi/libs/ 目录下的fireflyapi.jar包拷贝到 AS 工程对应的 libs 文件夹下；

详细请参照技术案例Demo工程配置和 doc/FireflyApi_instructions.png ；

FireflyApi 接口说明

1.1 ICCard/身份证/

1. 概述

当前设备主要三种硬件配置：

硬件配置	ICCard刷卡	身份证刷卡
身份证模块	支持	支持
nfc 模块	支持	不支持
无	不支持	不支持

2. 连接设备

启动监听服务，监听刷卡操作，建议在onResume()方法中执行；

```
/*
    打开后台监听服务
*/
IDCardUtil.getInstance().bindIDCardService(Context context);

/*
    nfc模块是以键值形式上报
*/
@Override
public boolean dispatchKeyEvent(KeyEvent event) {
    if (IDCardUtil.getInstance().handleEvent(event)) {return true;}

    return super.dispatchKeyEvent(event);
}
```

3. 检查功能支持

由于监听服务为异步启动，所以在连接设备后立刻进行功能支持检测不一定准确。
建议在6的监听回调onMachineConnect中再次进行功能支持检查

```
/*
    检测设备是否支持身份证识别
    boolean:true 表示支持、false 表示不支持
*/
boolean result =IDCardUtil.getInstance().isSupportIDCard();

/*
    检测设备是否支持ICCard识别
    boolean:true 表示支持、false 表示不支持
*/
boolean result = IDCardUtil.getInstance().isSupportICCard();
```

4. 设置刷卡模式

身份证模块虽然支持身份证和ICCard，但是不支持两者同时读取，需根据需要设置读卡模式

NFC模块 只支持 ICCard

```
/*
根据指定刷卡时的读取方式;
主要分为两种：READCARD_MODE_IDENTITY_CARD和READCARD_MODE_IC_CARD;

IDCardConfig.READCARD_MODE_IDENTITY_CARD = 0; //身份证模式
IDCardConfig.READCARD_MODE_IC_CARD = 1; //ICCard
IDCardConfig.READCARD_MODE_IDENTITY_CARD_UUID = 2; //身份证UUID模式
*/
IDCardUtil.getInstance().setModel(int readMode);
```

5. 设置读取ICCard的字节序模式

设置读取ICCard时的字节序模式，默认为大端

```
/*
boolean:true 大端、false 小端
*/
IDCardUtil.getInstance().setICCardEndianMode(boolean useBig);
```

6. 设置监听和回调函数

```
/*
绑定刷卡监听回调
context
callback 监听回调
*/
IDCardUtil.getInstance().setIDCardCallBack(IDCardUtil.IDCardCallBack callBack);

//监听回调
IDCardUtil.IDCardCallBack callBack = new IDCardUtil.IDCardCallBack() {

    //监听服务为异步启动,启动后若发现存在读卡设备,则回调onMachineConnect
    @Override
    public void onMachineConnect() {
        Log.i("firefly", "onMachineConnect ");
    }

    //身份证刷卡时执行回调
    @Override
    public void onSwipeIDCard(final IDCardBean info) {
        Log.i("firefly",
            "picture:" + info.getPicture() + "\n" +
            "name: " + info.getName() + "\n" +
            "sex: " + info.getSex() + "\n" +
            "nation: " + info.getNation() + "\n" +
            "birthDate: " + info.getBirthDateStr() + "\n" +
            "address: " + info.getAddress() + "\n" +
        );
    }
};
```

```

        "number: " + info.getNum() + "\n" +
        "issue: " + info.getIssue() + "\n" +
        "expiration date: " + info.getCreateTimeStr() + "-" + info.getValidTimeStr() + "\n" +
        "picture:" + info.getPhoto() + "\n");
    }

    // IC卡刷卡时执行回调
    @Override
    public void onSwipeICCard(final ICardBean info) {
        Log.i("firefly", "onSwipeICCard IC=" + info.getId());
    }

    // 当设置读卡模式为READCARD_MODE_IDENTITY_CARD_UUID时,刷身份证回调
    @Override
    public void onSwipeIDCardUUID(final String uuid) {
        Log.i("firefly", "onSwipeIDCardUUID uuid=" + uuid);
    }
}
};

```

7. 断开连接

移除监听回调并停止监听服务，建议在onStop()方法中执行；

```

IDCardUtil.getInstance().setIDCardCallBack(null);
IDCardUtil.getInstance().unBindIDCardService(Context context);

```

1.2 补光灯控制

补光灯分为4种：红外补光灯，白色补光灯，红色补光灯和绿色补光灯，

1. 红外补光灯:

新版机器支持红外补光灯开关(默认为关)，旧版机器不支持开关(默认为开).
故提供接口供用户使用。

注：闸机进行活体识别失败时可能是因为红外补光灯没有打开，导致IR活体检测失败

```

/*
    检测设备是否支持红外补光灯开关；
    Boolean:true 表示支持、false 表示不支持
*/
HardwareCtrl.isSupportInfraredFillLight();

/*
    红外线补光灯操作；
    open true 表示打开、false 关闭
*/
HardwareCtrl.setInfraredFillLight(open);

```

2. 白色补光灯:

新版机器支持白色补光灯亮度调节,旧版机器只支持开关

```
/*
    检测设备是否支持白色补光灯亮度调节
    Boolean:true 表示支持亮度调节、false 表示不支持
*/
HardwareCtrl.isFillLightBrightnessSupport();

/*
    获取亮度调节的最大值;
*/
int maxValue = HardwareCtrl.getFillLightBrightnessMax();

/*
    获取亮度调节的最小值;
*/
int minValue = HardwareCtrl.getFillLightBrightnessMin();

/*
    打开/关闭, 如果支持亮度调节, 则开灯时可以传入对应的亮度值;
    enable true 打开, 并设置对应的亮度值;
    false 关闭
*/
HardwareCtrl.ctrlLedWhite(enable,brightness);

/*
    打开/关闭;
    enable true 打开, 若当前版本支持亮度调节则设为亮度最大值
    false 关闭
*/
HardwareCtrl.ctrlLedWhite(enable);
```

3. 红色LED灯;

```
/*
    isChecked true 打开
    false 关闭
*/
HardwareCtrl.ctrlLedRed(isChecked);
```

4. 绿色LED灯;

```

/*
    isChecked true 打开
    false 关闭
*/
HardwareCtrl.ctrlLedGreen(isChecked);

```

1.3 信号控制

1. “rs485/232” 信号操作，

485串口地址：/dev/ttyS4 ， 波特率 :19200

232串口地址：/dev/ttyS3 ， 波特率 :19200

可以通过cat /dev/ttyS4 或者 cat /dev/ttyS3 查看设置的值变化；

```

/*
    获取串口serialPort
    485串口：/dev/ttyS4; 232串口：/dev/ttyS3
*/
SerialPort serialPort = HardwareCtrl.openSerialPortSignal(new File("/dev/ttyS3"), 19200, new
SerialPort.Callback() {

    //rs485/232发送信号后，接收到的返回值
    @Override
    public void onDataReceived(byte[] bytes, int i) {
        String result = StringUtils.bytesToHexString(bytes, size);
        Log.i("firefly", "result = "+result);
    }
});

/*
    发送 '48562311' 信号
*/
HardwareCtrl.sendSerialPortHexMsg(serialPort, "48562311")

/*
    页面退出时，关闭串口
*/
HardwareCtrl.closeSerialPortSignal(serialPort);

```

2. 韦根信号操作：

1) 监听 韦根输入信号

```

/*开始监听韦根输入*/
HardwareCtrl.openRecvWiegandSignal("/dev/wiegand");

/*监听回调*/
HardwareCtrl.recvWiegandSignal(new RecvWiegandCallBack() {
    @Override

```

```

public void recvWiegandMsg(int i) {
    Log.i("firefly", "result = "+i);
}
});

/*停止监听韦根输入*/
HardwareCtrl.closeRecvMiegandSignal();

```

2) 韦根输出

分为 韦根26和 韦根34

```

/*
    韦根26Output
    通过cat /sys/devices/platform/wiegand-gpio/wiegand26 查看设置的值变化。
*/
HardwareCtrl.sendWiegandSignal("123456789");

/*
    韦根34Output
    通过cat /sys/devices/platform/wiegand-gpio/wiegand34 查看设置的值变化。
*/
HardwareCtrl.sendWiegand34Signal("123456789");

```

3. 电平信号/继电器信号；

```

/*
    D0电平信号 isChecked true 表示打开、false 关闭
*/
LevelSignalUtil.sendSignalD0(isChecked);

/*
    D1电平信号 isChecked true 表示打开、false 关闭
*/
LevelSignalUtil.sendSignalD1(isChecked);

/*
    继电器信号 isChecked true 表示打开、false 关闭
*/
RelayUtil.sendRelaySignal(isChecked);

```

1.4 二维码

```

/*
    启动二维码扫描服务
*/
QRCodeUtil.getInstance().init();

```

```

/*
    检测设备是否支持二维码功能；
    ret:true 表示支持、false 表示不支持
*/
boolean ret = QrCodeUtil.getInstance().isQrCodeSupport();

/*
    添加二维码监听回调
*/
QrCodeUtil.getInstance().setQRCodeCallback(new QrCodeUtil.QRCodeCallback() {
    //监听服务为异步启动,启动后若发现存在二维码设备,则回调onConnect
    @Override
    public void onConnect() {
        Log.i("firefly", "QRCode onConnect:");
    }

    //二维码识别的内容回调方法
    @Override
    public void onData(final String s) {
        Log.i("firefly", "QRCode onData:"+s);
    }
});

/*
    二维码扫描补光灯操作
    QrCodeUtil.LED_STATE_AUTO //扫码时自动打开
    QrCodeUtil.LED_STATE_ON //常亮
    QrCodeUtil.LED_STATE_OFF //常灭
*/
QrCodeUtil.getInstance().setLedState(int state);

/*
    设置对焦灯状态
    QrCodeUtil.LED_STATE_AUTO //扫码时自动打开
    QrCodeUtil.LED_STATE_ON //常亮
    QrCodeUtil.LED_STATE_OFF //常灭
*/
QrCodeUtil.getInstance().setFocusLedState(int state);

/*
    设置二维码工作状态
    active 为false时不会回调数据而且会自动关闭照明灯
*/
QrCodeUtil.getInstance().setActive(boolean active);

/*
    页面退出时，关闭二维码扫描
*/
QrCodeUtil.getInstance().release();

```

参数说明:

1.5 体温检测

提供了温度校正接口，仅供参考使用


```
/*
    启动体温检测服务
*/
TemperatureUtil.getInstance().openDevice();

/*
    检测设备是否支持体温；
    Boolean:true 表示支持、false 表示不支持
*/
TemperatureUtil.getInstance().isSupport();

/*
    添加体温检测监听回调
*/
TemperatureUtil.getInstance().setTemperatureCallback(new TemperatureUtil.TemperatureCallback() {
    //监听服务为异步启动,启动后若发现存在测温设备,则回调onConnect
    @Override
    public void onConnect() {
        Log.i("firefly", "TemperatureCallback onConnect:");
    }

    //体温检测的内容回调方法
    @Override
    public void update(float ambientTemperature/*环境温度*/, float objectTemperature/*人体温度*/) {
        Log.i("firefly", "TemperatureCallback update:ambientTemperature="+ambientTemperature +
            "objectTemperature="+objectTemperature);

        Log.i("firefly", "体温校正后 TemperatureCallback update:ambientTemperature="+ambientTemperature +
            "objectTemperature="+TemperatureUtil.correctTemp(objectTemperature));
    }
});

/*
    温度校正函数，对体温进行一个简单的校正
    float: objectTemperature 人体温度
*/
TemperatureUtil.correctTemp(float objectTemperature) {
    Log.i("firefly", "TemperatureCallback onConnect:");
}

/*
    页面退出时，关闭测温功能
*/
TemperatureUtil.getInstance().closeDevice();
```

1.6 雷达

```
//监听雷达信号
//雷达信号以KeyEvent的形式触发,使用handleEvent(KeyEvent event)处理
```

返回值ret说明:

RadarUtil.EVENT_HANDLE_NOTHING_UNHANDLE //非雷达事件未处理

RadarUtil.EVENT_HANDLE_NOTHING_HANDLED //非雷达事件已处理

RadarUtil.EVENT_HANDLE_RADAR_IN //有物体进入

RadarUtil.EVENT_HANDLE_RADAR_OUT; //有物体离开

```
@Override
public boolean dispatchKeyEvent(KeyEvent event) {
    int ret = RadarUtil.handleEvent(event);
    if (ret == RadarUtil.EVENT_HANDLE_RADAR_IN) { //物体进入
        Log.i("firefly", "EVENT_HANDLE_RADAR_IN");
        return true;
    } else if (ret == RadarUtil.EVENT_HANDLE_RADAR_OUT) { //物体离开
        Log.i("firefly", "EVENT_HANDLE_RADAR_OUT");
        return true;
    } else if (ret == RadarUtil.EVENT_HANDLE_NOTHING_HANDLED) { //无物体
        Log.i("firefly", "EVENT_HANDLE_NOTHING_HANDLED");
        return true;
    }

    return super.dispatchKeyEvent(event);
}
```